Enterprise Computing

Band 1
Einführung in z/OS

Prof. Dr. Martin Bogdan
Prof. Dr.-Ing. Wilhelm G. Spruth

bogdan@informatik.uni-leipzig.de
spruth@informatik.uni-leipzig.de

Juli 2013

Die Vorderseite des Buches zeigt einen zEC12 Rechner mit angeschlossener zEnterprise Blade Center Extension (zBX)

We dedicate this book to the genius of

Gene M. Amdahl
Gerry A. Blaauw
Fred P. Brooks

who invented the S/360 Architecture,

and to the vision of IBM Vicepresident

Bob O. Evans

and his team, who made it happen.

Vorwort

1. Struktur

Das vorliegende e-Book entstand aus einer zweisemestrigen e-learning Vorlesung „Enterprise Computing“, die mein Kollege Prof. Martin Bogdan und ich gemeinsam über viele Jahre als Teil des Bachelor Studienganges an der Universität Leipzig gehalten haben. Die Vorlesungsscripte sind auf dem Moodle Lernserver der Universität verfügbar.

Wir haben das umfangreiche Material in drei Bände gegliedert. Band 1 deckt im Wesentlichen die Vorlesung im Wintersemester ab; Band 2 die Vorlesung im darauffolgenden Sommersemester.

Die Vorlesungen werden von praktischen Übungen auf dem z9 Mainframe Rechner des Lehrstuhls Technische Informatik ergänzt. Die Übungsanweisungen (Tutorials) können Sie herunterladen unter http://jedi.informatik.uni-leipzig.de/de/zos.html.

Band 1 deckt diese Themen ab:

- **Grundlagen**
- **z/OS**
- **Hardware**
- **Ein/Ausgabe**
- **VSAM**
- **Transaktionen**
- **CICS**
- **CICS Connectivity**
- **WebSphere MQ (MQSeries)**

Band 2 deckt diese Themen ab:

- **Sysplex**
- **Virtualisierung und logische Partitionen**
- **Work Load Manager**
- **zBX, PureData System for Analytics**
- **Java Enterprise Edition**
- **RMI**
- **WebSphere Application Server**
- **Java Connection Architecture**
- **Transaktionsverarbeitung mit Java**
- **SOA**

Ein Verzeichnis der Akronyme und ein Stichwortverzeichnis für Band 1 und Band 2 befinden sich am Ende von Band 2.

Wir glauben, dass Fachbücher in zunehmendem Umfang als e-Books und in Farbe herausgebracht werden, und klassische gedruckte Bücher zunehmend ablösen werden.

Unser Buch liegt im pdf Format vor. Eine Änderung des Seitenumbruchs ist nicht möglich. Das ursprüngliche Word Dokument im 12 Point Ariel Font wurde mit Adobe Acrobat 8.0 konvertiert. Es werden immer ganze Seiten dargestellt.

Auf einem Nexus 7 Zoll Tablet mit einer Auflösung von 1280 x 800 im Hochformat wird der ganzseitige Text zwar sehr klein dargestellt, ist aber noch angenehm lesbar. Das Gleiche gilt für die Abbildungen.

Auf einem iPAD 2 mit einem 9,7 Zoll 1024 x 768 Bildschirm und 132 ppi ist die Darstellung hervorragend.

Auf einem 6 Zoll Kindle Paperwhite ist Text noch sehr gut lesbar, obwohl die Buchstaben sehr klein sind. Auch Abbildungen sind noch überraschend gut lesbar, die Konvertierung von Farbe in Graustufen erzeugt gute Ergebnisse.

Auf einem NOOK Simple Touch E-Ink Reader von Barnes und Noble sind unsere PDF Dateien marginal lesbar; außerdem gehen die Vorteile der farbigen Darstellung verloren. Nachteilig ist auch, dass die Zeichengröße sich nicht optimal einstellen lässt.

Auf einem 24 Zoll 1920 x 1200 Bildschirm lassen sich 3 Seiten parallel in drei Fenstern komfortabel darstellen.

2. Die Zielgruppe

Die Kapitel dieses Buches sind für Einsteiger in die Mainframe Technologie und das z/OS Betriebssystem z/OS gedacht. Als Vorkenntnisse ist das Wissen wünschenswert, welches Studenten der Informatik in den ersten 3-4 Semestern an einer deutschen Hochschule erwerben.

Nach dem Durcharbeiten des Lernstoffes sollten Sie in der Lage sein, bei Integrationsprojekten unter Anleitung produktiv mitzuarbeiten. Falls dabei Code erstellt wird, sind Kenntnisse in der entsprechenden Programmiersprache erforderlich. Dies wird vermutlich Cobol und/oder Java sein, eventuell aber auch PLI oder C/C++.

Die Kapitel dieses Buches sind keine Einführung in die Systemprogrammierung. Für eine Karriere als Systemprogrammierer brauchen Sie tiefergehende Kenntnisse in eine ganze Reihe von Fachgebieten, welche wir hier nicht vermitteln. Falls sie hieran interessiert sind, empfehlen wir Ihnen http://www.mainframe-academy.de.

3. Möglichkeiten der Weiterbildung

Unser Vorlesungs- und Übungsmaterial (Tutorien) steht im Netz, und kann ohne weitere Verpflichtungen heruntergeladen werden.

Eine zusätzliche Weiterbildung ist im Rahmen eines drei-Stufenplans möglich.

Stufe 1 Wer sich als Gasthörer der Uni Leipzig einträgt, erhält einen Zugriff auf unseren Moodle Server. Er kann die Multichoice Testfragen (circa 400 – 500) bearbeiten, das Forum benutzen und die Scripte für die praktischen Übungen herunterladen. Letztere sind auf einem z/OS Rechner Ihrer Wahl ausführfähig. In einem begrenzten Umfang gehen wir auf Fragen im Forum ein. Die Uni verlangt 40 € pro Semester als Einschreibe Gebühr; der Lehrstuhl hat keine finanziellen Vorteile.

Stufe 2 Gasthörer erhalten zusätzlich für 6 MonateZugriff auf unseren z9 Mainframe Server und Eclipse/RDz Server. Sie erhalten weiterhin eine Kopie der EJB 3.0 virtuellen Maschine. Wir

- **beantworten im Forum Fragen zu den oraktischen Übungsaufgaben,**
- **nehmen Screen Shots entgegen, und**
- **erstellen auf Wunsch einen Schein, der die erfolgreiche Durchführung der Übungen bestätigt.**

Hierfür wird eine Gebühr an den Lehrstuhl fällig um unsere Verwaltungskosten abzudecken..

Stufe 3 Herr Professor Bogdan nimmt eine mündliche Prüfung ab. Der Teilnehmer erhält ein Zertifikat mit einer Prüfungsnote. Die Prüfung erfolgt nach den gleichen Qualitätskriterien wie eine Leipziger Bachelor Studiengangsprüfung.

Hierfür wird eine Gebühr an Herrn Prof. Bogdan fällig.

4. Danksagung

Wir bedanken uns bei:

Mitarbeitern der Firma IBM, besonders Prof. Dr. Karl Ganzhorn, Herb Kircher, Paul Nemeth, Uwe Denneler, Elisaeth Puritscher, Dr. Peter Hans Roth, Roland Trauner, Dr. Jens Müller,

Kollegen und Mitarbeitern an der Universität Leipzig, besonders Herrn Dipl.-Inf. Frank Güttler,

Kollegen und Mitarbeitern an der Universität Tübingen, besonders Herrn Prof. Dr. Wolfgang Rosenstiel, Dekan der Mathematisch-Naturwissenschaftlichen Fakultät, sowie Herrn Dr. Axel Braun,

Diplom-, Bachelor und Masterstudenten an beiden Hochschulen,

Volker Falch und Wolfgang Greis, European Mainframe Academy,

meiner Frau Angela geb. Scheithauer, die mich bei der Erstellung des Buches ganz hervorragend unterstützte.

Böblingen, Juli 2013, Prof. Dr.-Ing. Wilhelm G. Spruth

Inhaltsverzeichnis

6. Data Sets, VSAM

7. Transaktionsverarbeitung

8. CICS Transaktionsserver

9. CICS Communication

10. MQSeries

1. Einführung

1.1 Eigenschaften eines Mainframes

1.1.1 Was ist Enterprise Computing ?

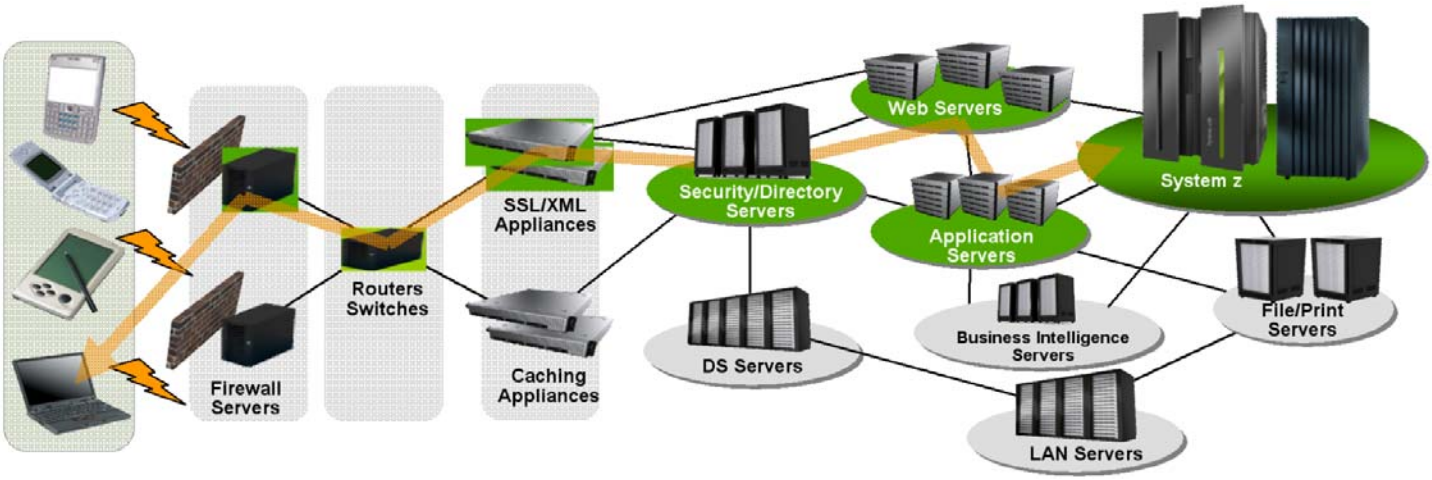


Abb. 1.1.1
IT Infrastruktur eines großen Unternehmens

Große Wirtschaftsunternehmen (Beispiele VW, Daimler, Deutsche Bank, Allianz, Bausparkasse Wüstenrot) und große staatliche Organisationen (Bespiele Rentenversicherung, Oberfinanzdirektion) benötigen eine umfangreiche IT-Infrastruktur für Ihren täglichen Betrieb. Diese Infrastruktur besteht aus bis zu mehreren 100 000 Klienten, zahlreichen Abteilungsservern, einer Netz Infrastruktur und einem (oder mehreren) zentralen Mainframe Rechnern.

Bei den Tausenden, Zehntausenden oder Hundertausenden von Klienten handelt es sich um Bildschirm-Arbeitsplätze, Tablets, Mobiltelefone, Registrierkassen, Geldausgabeautomaten, Kreditkarten Terminals, Industrieelektronik und anderen Geräten. Die dezentalen Abteilungsserver benutzen Betriebssysteme wie Windows, Linux, Mac OS, AIX, HP-UX, Solaris und andere. Häufig sind alle Hersteller und alle Betriebssysteme vertreten, die derzeitig auf dem Markt verfügbar sind oder in der Vergangenheit verfügbar waren. Dazu kommen die verschiedenen Kommunikationsnetze, (Lokal Area Netzwerke, häufig Ethernet, Internet sowie drahtlose Netze) mit den entsprechenden Routern, Switche, Controllern usw. Schließlich existiert ein zentrales Rechenzentrum, welches u.a. für die Speicherung und Konsistenz unternehmenskritischer Daten zuständig ist.

Es besteht ein deutlicher Trend, die dezentralen Abteilungsserver zu zentralisieren, und viele kleine Server durch wenige große Server zu ersetzen.

Als zentraler Rechenzentrumsserver wird bei großen Unternehmen und staatlichen Organisationen fast immer ein „Mainframe“ eingesetzt.

Das Zusammenspiel all dieser Elemente wird als Enterprise Computing bezeichnet. Wegen seiner Größe erfordert Enterprise Computing spezielle Einrichtungen.

1.1.2 Was ist ein Mainframe ?

Ein Mainframe ist der zentrale Server in großen Wirtschaftsunternehmen und staatlichen Organisationen. Von den 200 größten deutschen Unternehmen setzen 95 % einen Mainframe als ihren zentralen Server ein.

Mainframes werden (fast) ausschließlich von IBM hergestellt. Sie verwenden die System z Hardware und Architektur. In den allermeisten Fällen läuft auf einem Mainframe das z/OS Betriebssystem (andere Bezeichnungen OS/390, MVS). Neben z/OS gewinnt zLinux (Linux für Mainframes) an Bedeutung.

Rechner für maximale Rechenleistung werden als Supercomputer bezeichnet. Bei Mainframes spielen andere Faktoren als die reine Rechenleistung eine dominierende Rolle:

- **Ein/Ausgabeleistung**
- **Leistungsverhalten bei Transaktions- und Datenbankanwendungen.**
- **Zuverlässigkeit/Verfügbarkeit**
- **Sicherheit**

Neben IBM stellt Fujitsu/Siemens Hardware-kompatible Mainframes her, auf denen neben z/OS das hauseigene BS2000 Betriebssystem läuft. Inkompatible Großrechner von Unternehmen wie Bull und Unisys sowie Unix Großrechner von Hewlett Packard (HP) und Oracle/Sun können (mit Abstrichen) ähnliche Aufgaben wie Mainframes übernehmen.

Heutige Technologie ermöglicht es, eine bestimmte Hardware Architektur auf einer anderen Platform zu emulieren, z.B. die System z Architektur auf einem x86 (Intel, AMD) Rechner.

Das IBM System z Personal Development Tool (zPDT) besteht aus einem Dongle (USB Stick). Dieser kann mit einem regulären x86 (Intel, AMD) Rechner betrieben werden. Hiermit ist es möglich, z/OS auf einem x86-Rechner laufen zu lassen, allerdings mit stark verringerter Leistung. Siehe http://www.redbooks.ibm.com/redbooks/pdfs/sg247721.pdf .

Ähnliches leistet der Public Domain Hercules Emulator, der aber von IBM (im Gegensatz zu zPDT) nicht unterstützt wird. Siehe http://www.conmicro.cx/hercules/ .

Die Fujitsu Siemens SX Serie Systeme emulieren auf SPARC oder Intel Prozessoren das hauseigene BS2000 Betriebssystem.

1.1.3 Terminologie

IBM bezeichnet seine Hardware als System z, zSeries oder S/390 und das am meisten eingesetzte Betriebssystem als z/OS oder OS/390. Die früheren Rechner wurden als S/360 und S/370 bezeichnet, die Betriebssysteme als OS/360, OS/370 und MVS. Das heutige z/OS Betriebssystem wird immer noch häufig als MVS bezeichnet.

System z und z/OS weisen gegenüber S/390 und OS/390 eine zusätzliche 64 Bit-Unterstützung und andere Erweiterungen (z.B. Kryptografie) auf.

IBM garantiert, dass alle seit 1965 entwickelte S/360 Software unmodifiziert und ohne Recompilation auf den heutigen System z Rechnern läuft !!!

x86 ist die Abkürzung für eine Mikroprozessor-Architektur und der damit verbundenen Befehlssätze, welche unter anderem von den Chip-Herstellern Intel und AMD entwickelt werden. Wir benutzen den Begriff sowohl für die 32 Bit als auch für die 64 Bit Version.

1.1.4 Der Dinosaurier Mythos

Über viele Jahre (Jahrzehnte) wurden Mainframes als technologisch veraltete Dinosaurier betrachtet, die vermutlich bald von der Bildfläche verschwinden würden. In Wirklichkeit ist genau das Gegenteil der Fall.

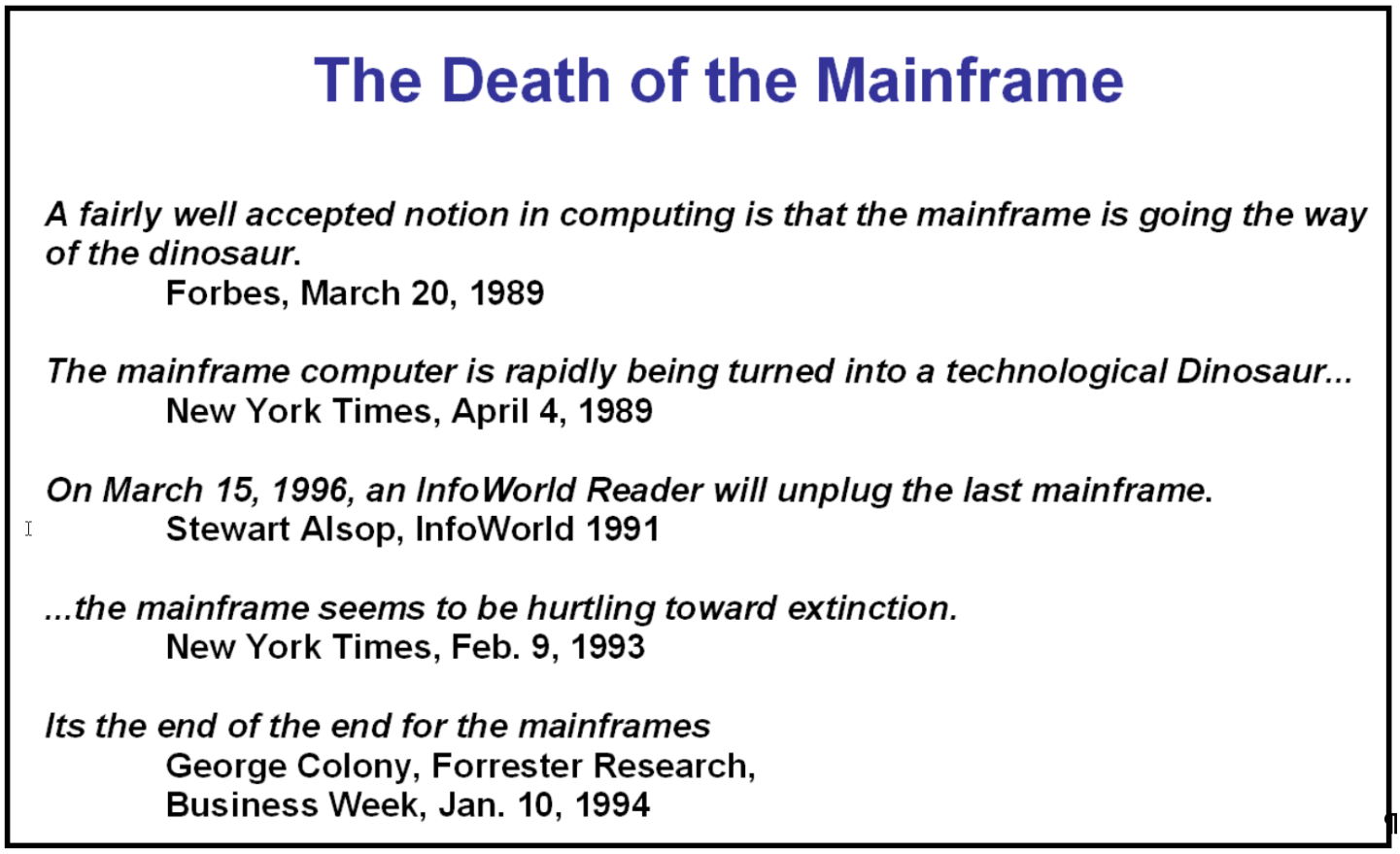


Abb. 1.1.2
Diese Liste reproduziert Aussagen von „IT Experten“

Abb. 1.1.3
The Hype

Die Anzeige in Abb. 1.1.3 wurde auf der CEBIT, Hannover, vom 13. bis 20. März 2002 gezeigt. Sie stellte den Sun Fire 15K Server als den "schnellsten kommerziellen Computer" dar.

Das Bild zeigt eine alte kranke Frau vor einem durch einen Ölteppich verschmutzten Strand.

Dies ist eine hervorragende Anzeige, welche die Botschaft auf eine sehr kraftvolle Art Weise herüberbringt. Leider war die Botschaft falsch.

Unternehmen, denen der Rechner verkauft wurde, erlebten anschließend in eine böse Überraschung. Der Sun Fire Linie von Systemen hatte viele Zuverlässigkeits- und Verfügbarkeitsprobleme, von denen einige auf den fehlenden Error Correction Code (ECC) im L1 und L2-Cache zurückzuführen waren. Die Reputation litt und Marktanteile gingen verloren. Sun Microsystems brauchte viele Jahre, um sich von dem Rückschlag zu erholen.

Peter Baston: “Unsafe At Any Speed?”
http://www.sparcproductdirectory.com/artic-2002-jan-pb.html oder als Mirror
http://www.cedix.de/VorlesMirror/Band1/UnsafeAtAnySpeed.pdf

1.1.5 Die Wirklichkeit

Nach wie vor werden zwischen 70 und 90 Prozent des weltweiten Datenbestands von Mainframe-Installationen verwaltet.

Abb. 1.1.4
Computerwoche 9/2006, 3. März 2006, S. 26

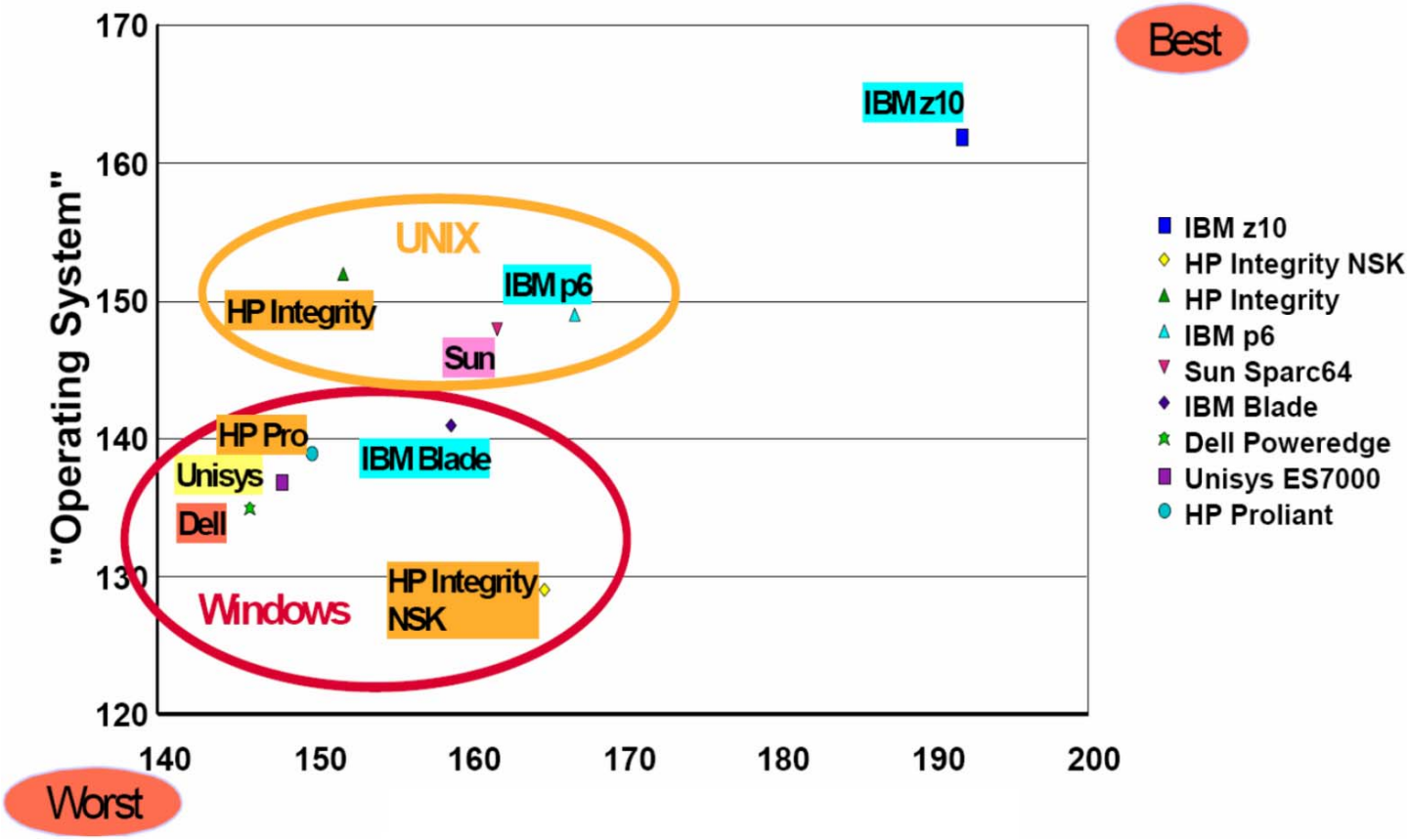


Abb. 1.1.5
Gartner, Server Scorecard Evaluation Model version 5, 4Q08

Gartner Research definiert als „Commercial large Systems“ Rechner mit einem Listenpreis von mehr als $ 250 000.

Für viele Jahre hat Gartner einen Jahresbericht (Server Scorecard Auswertung Model) publiziert, welcher die Merkmale von großen Systemen evaluiert. Mainframes waren immer Nr. 1, sowohl in der Gesamtwertung als auch in nahezu allen individuellen Merkmalen.

Einer der wichtigsten Gründe ist Spitzentechnologie. Mainframes bieten viele Software-und Hardware-Eigenschaften, die auf anderen Server-Platformen nicht verfügbar sind. Auf der anderen Seite kennen wir keine Unix-oder Windows-Server-Eigenschaften, die nicht auch auf Mainframes verfügbar sind.

1.1.6 Technologische Führungsposition

Nach wie vor werden Mainframes häufig mit veralteter Technologie assoziiert.

Überraschenderweise ist genau das Gegenteil der Fall. Mainframes verfügen über viele Hardware, Software und System Integrations-Eigenschaften, die entweder gar nicht, oder nur in rudimentärer Form, auf anderen Großrechnern verfügbar sind.

Der Bericht “System z and z/OS unique Characteristics” *) erläutert 40 führende technologische Hardware- und Software-Eigenschaften, die nur auf Mainframes zu finden sind. Im Gegensatz dazu sind keine führenden technologischen Eigenschaften auf anderen Großrechnern bekannt, die nicht auch auf Mainframes zu finden sind.

Gartner Research ist das weltweit führende Marktforschungsunternehmen auf dem Gebiet Informatik. Seit den 90er Jahren veröffentlich Gartner einen Bericht, in dem die wichtigsten Großrechner für den betriebswirtschaftlichen Einsatz miteinander verglichen werden. Hierbei werden sehr viele Einzelbewertungen zu einer Gesamtbewertung zusammengefasst.

Seitdem diese Untersuchungen veröffentlicht werden, nehmen Mainframes immer die Nr. 1 Position ein. Dies gilt nicht nur für die Gesamtwertung, sondern in nahezu allen Fällen auch für Einzelbewertungen.

Mit der wichtigste Grund hierfür ist, dass Mainframes über viele führenden technologische Eigenschaften verfügen, die auf anderen Plattformen nicht verfügbar sind. Dies war in den letzten Jahrzehnten so, ist auch heute der Fall und wird voraussichtlich auch in Zukunft so bleiben.

Der Bericht “System z and z/OS unique Characteristics” *) erläutert 40 führende technologische Hardware- und Software-Eigenschaften, die nur auf Mainframes zu finden sind. Im Gegensatz dazu sind keine führenden technologischen Eigenschaften auf anderen Großrechnern bekannt, die nicht auch auf Mainframes zu finden sind.

***) Wilhelm G. Spruth: System z and z/OS unique Characteristics.**
Universität Tübingen, Wilhelm Schickard Institut für Informatik
Technical Report WSI-2010-03, ISSN 0946-3852, April 2010, download at
http://www.cedix.de/Publication/Mirror/report.pdf

Dies sind einige Beispiele aus dem Bericht „System z and z/OS unique Characteristics“ :

- **Architektur, z.B. Hardware Protection verhindert Buffer overflows**
- **Hardware-Technologie, z.B. MLC Multi-Chip Module**
- **Ein-/Ausgabe-Architektur (siehe Veröffentlichung)**
- **Clustering, Sysplex**
- **Skalierung mit Hilfe der Coupling Facility (siehe Veröffentlichung)**
- **Stapelverarbeitung (Job Entry Subsystem)**
- **Partitionierung und PR/SM LPAR Mode (siehe Veröffentlichung)**
- **Hipersockets (z/OS – zLinux Integration)**
- **Goal-orientierter Workload Manager (siehe Veröffentlichung)**
- **CICS Transaction Server**
- **WebSphere Web Application Server und MQSeries**
- **Persistent Reusable Java Virtual Machine (siehe Veröffentlichung)**

Veröffentlichungen siehe www.cedix.de/Publication/publish.html

1.1.7 Verfügbarkeit Classes of 9s

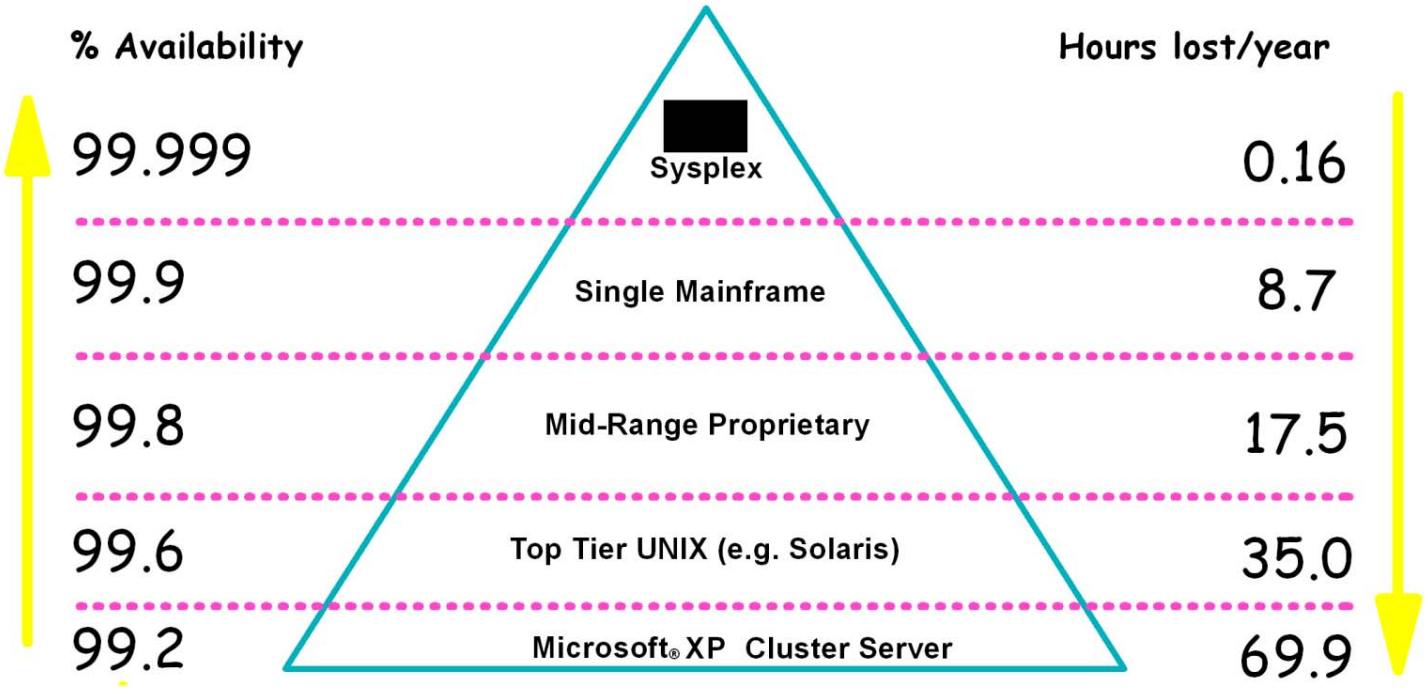


Abb. 1.1.6
Die Verfügbarkeitspyramide

Verfügbarkeit bedeutet, wenn ich auf den Rechner zugreifen will, ist er da. 99,999 % Verfügbarkeit bedeutet, der Rechner ist pro Jahr bis auf 0,16 Stunden, oder etwa 10 Minuten, erreichbar. Dies wird mit einer Sysplex Konfiguration erreicht. Näheres hierzu später.

Abb. 1.1.7
Kosten eines Rechner Ausfalls

Die Darstellung in Abb. 1.1.7 stammt aus einem Bericht der Firma Federal Express, dem größten Paketzustellungsunternehmen in den USA. Dies ist die Botschaft: Eine Minute Rechnerausfall kostet die Firma Amazon 4000 $. Bei Federal Express sind es 1 Million $ pro Minute.

Stellen Sie sich ein Logistik Zentrum von Federal Express vor. Mittels des voll automatischen Hochregal Lagers werden Lastwagen im 3 Minuten Takt be/entladen und abgefertigt. Fällt der zentrale Rechner aus, bilden sich in Bruchteilen einer Stunde kilometerlange Staus auf den Zufahrtstraßen und Autobahnen. Luftfracht-Anschlüsse werden verpasst und die Flugzeuge fliegen ohne Ladung.

Beispiel Toll Collect

Für das LKW Maut System der Bundesrepublik (Toll Collect) hat der Staat mit der Betreibergesellschaft eine Konventionalstrafe von 30 Mill. Euro für jede 60 Minuten Ausfallzeit vereinbart.

And in an independent survey of 520 CIOs interviewed on behalf of vendor Compuware, 71 percent were worried about a skills shortage and 80 percent said **mainframe outages were a major business risk.** It put the cost of a mainframe application outage at $13,931 per minute.

Abb. 1.1.8
Zuverlässigkeit und Verfügbarkeit sind kritische Mainframe Eigenschaften

http://www.cedix.de/VorlesMirror/Band1/Outage.pdf

Unternehmen und staatliche Organisationen sind bereit, sehr viel Geld für eine maximale Verfügbarkeit zu bezahlen.

1.1.8 Fehlerursachen

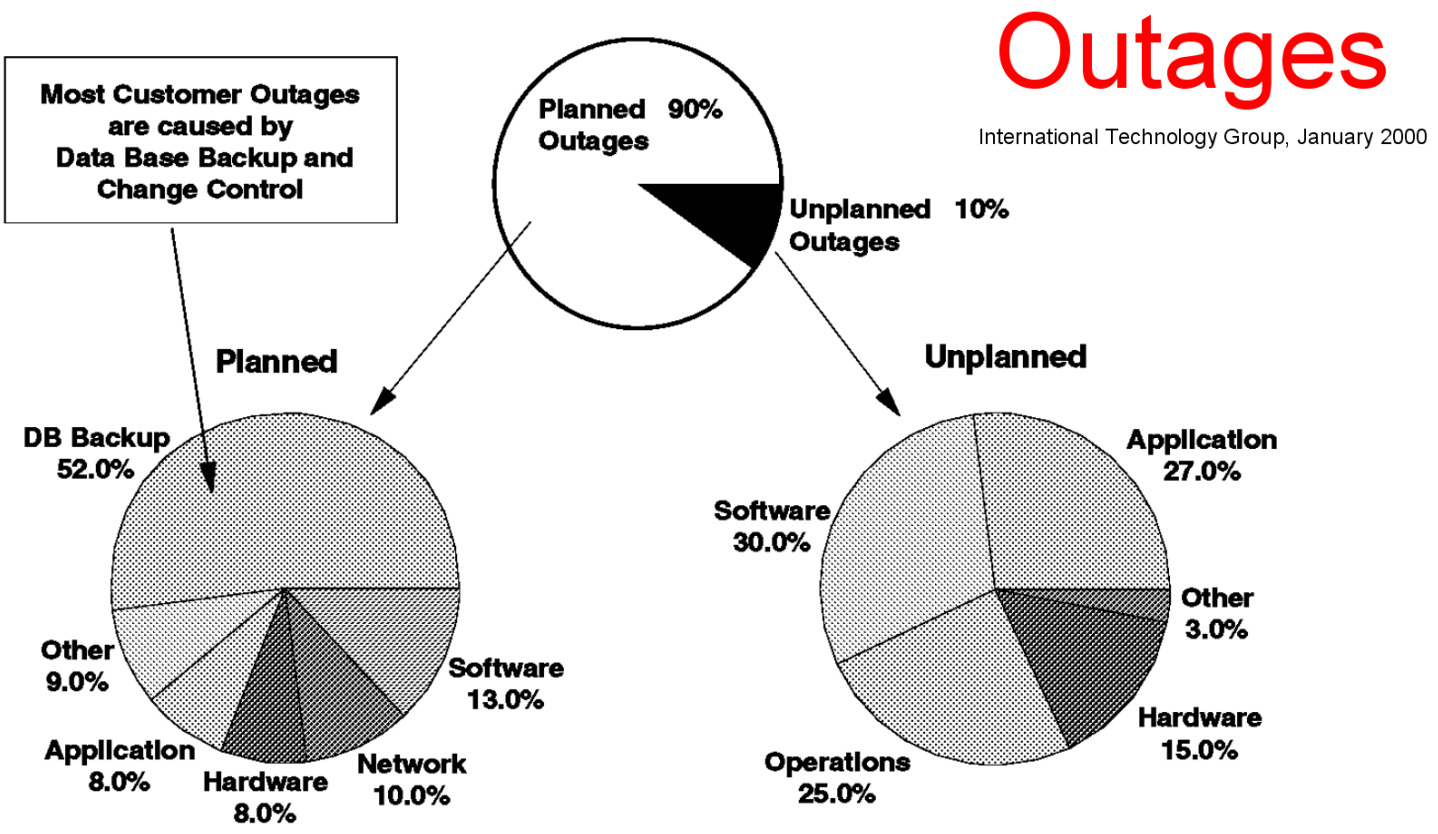


Abb. 1.1.9
Outage Ursachen

Als Outage (Non-availability, Nicht-Verfügbarkeit) bezeichnet man die Zeit, in der ein System für den Endbenutzer nicht verfügbar ist.

Outages können geplant oder ungeplant auftreten. Beispiele für geplante Outages sind:

- **Datenbank Backup**
- **Datenbank Reorganisation**
- **Release Wechsel, (z. B. Datenbank Versions-Upgrade)**
- **Hardware Erweiterung oder -Rekonfiguration**
- **Netzwerk Rekonfiguration**

Datenbank Backups (und Reorganisation) können den größten Beitrag zur Non-Availability von Mainframe Systemen leisten.

Beispiele für ungeplante Outages sind:

- **Permanente Hardware Fehler**
- **Permanente Software Fehler**
- **Datenprobleme (z.B. festgestellte Inkonsistenzen).**

Von den erwähnten 10 Minuten/Jahr Outage einer großen Mainframe Installation sind nur etwa 10 % (oder 1 Minute) nicht geplant. Dies ist ein statistischer Durchschnittswert; in der Praxis tritt eine unvorhergesehenes Aussetzen während der Lebensdauer des Systems nie auf.

Die restlichen 90 % (oder 9 Minuten) bedeuten, dass ein Datenbank Backup, eine Datenbank Reorganisation, ein Release Wechsel, eine Hardware Erweiterung oder –Rekonfiguration oder eine Netzwerk Rekonfiguration in den allermeisten Fällen während des laufenden Betriebs und unbemerkt für alle Benutzer erfolgen.

1.1.9 Wartung

Ein Mainframe verfügt über umfangreiche interne Diagnose und Selbst-Heilungseigenschaften. Die allermeisten Fehler repariert ein Mainframe von alleine.

Wird beispielsweise bei einer internen Datenübertragung ein Paritätsfehler entdeckt, veranlasst das System automatisch eine erneute Übertragung, in der Hoffnung, dass der Fehler nicht mehr auftaucht (automatic retry). In der Praxis stellt sich heraus, dass die allermeisten Hardware Fehler sog. transient errors sind, also bei einer Wiederholung nicht mehr auftreten.

Die meisten Register und Cache Speicher verwenden eine automatische Hamming Code Fehlerkorrektur. Die Hauptspeicher SIMMs verwenden neben der Hamming Code Fehlerkorrektur ein RAIM 5 Verfahren, ähnlich dem Plattenspeicher RAID 5 oder RAID 6. Es wird geschätzt, dass 30 – 40 % aller Schaltkreise auf einem CPU Chip der Fehlerdiagnose und automatische Fehlerkorrektur dienen.

Über alle transienten Fehler wird Buch geführt. Der Rechner ist über das Internet permanent mit einem IBM Diagnose Zentrum verbunden. Für die allermeisten Mainframe Rechner existiert ein Wartungsvertrag mit der IBM. Wenn die Anzahl der transienten Fehler einen Schwellwert überschreitet, wird ein Alarm ausgelöst. Als Folge kann es sein, dass ein IBM Außendiensttechniker unangemeldet auftaucht, um ein Problem zu beseitigen, von dem der Betreiber des Mainframes nicht weis, dass es überhaupt existiert.

1.1.10 Die kommende Revolution

Im Juli 2010 kündigte IBM die z196 Mainframe Modellreihe unter dem Namen zEnterprise an. Hierzu gehören die „zEnterprise BladeCenter Extension (zBX)“ und der „zEnterprise Unified Resource Manager (zManager)“.

Damit sind weitgreifende Änderungen in der Enterprise Computing Landschaft zu erwarten. Näheres hierzu in Band 2, Abschnitt 14.3 .

1.2 Total Cost of Ownership

1.2.1 Der zentrale Unternehmensserver

Abb. 1.2.1
Betriebswirtschaftliche Großrechner benötigen Funktionen, die Geld kosten.

Betriebswirtschaftliche Großrechner werden von Unternehmen wie Hewlett Packard (HP), Sun (Tochtergesellschaft von Oracle) und IBM hergestellt. IBM vertreibt neben den Mainframes noch die System p Linie (auch als „Power“ bezeichnet) mit dem AIX Betriebssystem.

Derartige Großrechner haben bis zu 256 CPU Cores, und Hauptspeicher Größen im Terabyte Bereich. Mainframes ausgenommen, benutzen sie in der Regel die gleichen Komponenten, die auch für individuelle Workstations mit einem Verkaufspreis von wenigen 1 000 $ eingesetzt werden. Als Großrechner enthalten sie aber viele zusätzliche Einrichtungen, die den Preis in die Höhe treiben.

Gezeigt ist ein Großrechner der Firma Sun. Das Unternehmen verkündet stolz, dass der Preis für eine Minimalversion nur wenig mehr als 1 Million $ beträgt. Voll ausgerüstete Systeme können auch 10 Millionen $ oder mehr kosten.

1.2.2 Total Cost of Ownership

Der Begriff “Total Cost of Ownership” (TCO) wurde in den 90er Jahren von der Firma Gartner generiert. Gartner stellte die Frage: Wenn ich einen Arbeitsplatz, der bisher keinen Computer hatte, mit einem PC ausstatte, was sind die Gesamtkosten.

Gartners Rechnung sah ganz grob so aus: 1 000 $ für die PC Hardware, 1 000 $ für Software Lizenzen, und weitere 8 000 $ für die Administration. Nachdem dieses Ergebnis ursprünglich angezweifelt wurde, wurde es in den Folgejahren mehr oder weniger akzeptiert. Seitdem ist TCO eine akzeptierte Messgröße in den Budget Diskussionen der Unternehmen.

Was fällt unter den Begriff Administration ? Dazu gehört die Existenz eines Support Centers, welches ein Mitarbeiter anrufen kann, wenn er ein Problem hat. Eine Faustformel besagt, dass ein Mitarbeiter im Support Center zwischen 30 und 150 PC Arbeitsplätze betreuen kann. Das Support Center deckt einen weiten Bereich unterschiedlicher Aufgaben ab, zum Beispiel Einspielen von Security Patches, Upgrades, Analyse und Beseitigung von Hardware- oder Software Problemen, Passwortverwaltung, Firewall Updates, Beseitigung von Netzwerkproblemen ....

Ein signifikanter Anteil der TCO wird durch das folgende Scenario beschreiben: „ Hallo Fritz, kannst Du mal rüberkommen, ich habe ein Problem mit meinem PC“. Als Folge sind Fritz und sein Kollege 2 Stunden damit beschäftigt, irgend etwas auf dem PC umzustrukturieren.

Das Unternehmen interessieren vor allem die Kosten, die dadurch entstehen, dass 2 + 2 Stunden produktive Arbeit nicht geleistet werden.

1.2.3 Betriebskosten Vergleich Mainframe versus Unix

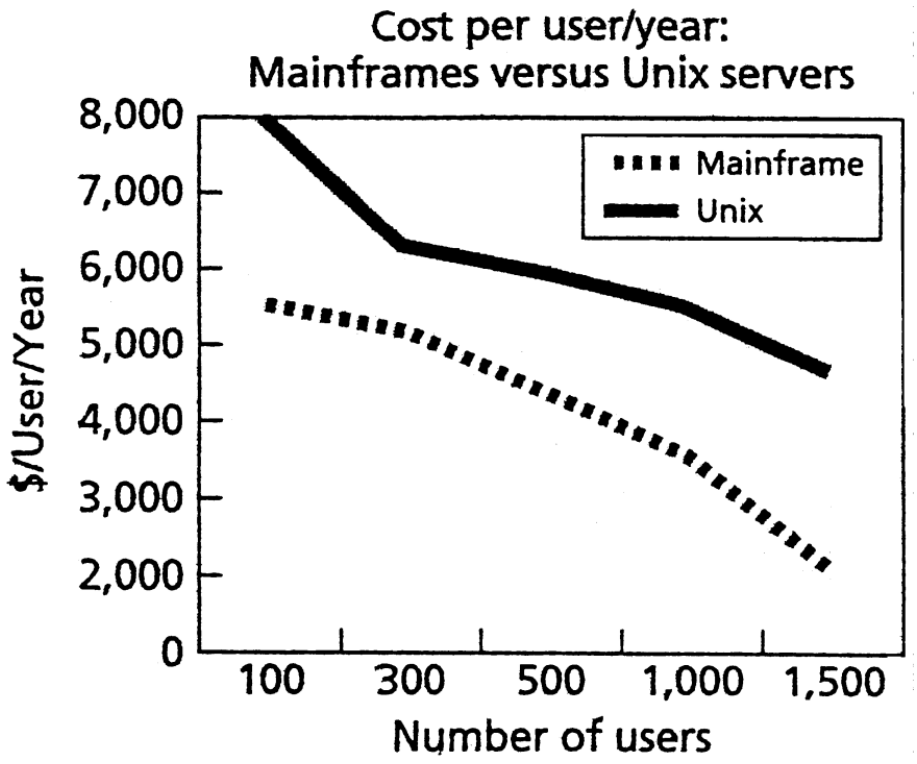


Abb. 1.2.2
Die TCO sinkt mit der Anzahl der an den Rechner angeschlossenen Benutzer

Ted Lewis: „Mainframes are dead, long live Mainframes.“ IEEE Computer, Aug. 1999, p. 104.

Die hier wiedergegebene Darstellung zeigt die TCO Kosten pro angeschlossenen Benutzer große Unix Rechner und für Mainframes. Allgemein kann gesagt werden, dass die TCO umso günstiger wird, je größer der Server, und je mehr Benutzer er bedient.

Die hier wiedergegebene Grafik stammt aus der Zeitschrift IEEE Computer und hat damit einen Spitzenwert an Glaubwürdigkeit. Sie stammt aus dem Jahre 1999; die Zahlen haben sich geändert, aber die Botschaft ist geblieben: Von allen großen Servern haben Mainframes mit Abstand die günstigste TCO:

Zahllose Untersuchungen bestätigen diese Aussagen.

Unternehmen können die TCO senken, wenn sie die Funktionen des Arbeitsplatz PCs einschränken und auf zentrale Server verlagern. Die erheblichen zusätzlichen Kosten für Server Hardware und Software werden in der Regel überproportional durch einen sinkenden Aufwand für die Administration kompensiert. Im Extremfall werden minimale PCs ohne Festplatte, nur mit einem Netzanschluss eingesetzt, wobei auf dem PC ein Browser als einzige Software läuft (sogenannte thin PCs). In Deutschland liefert die Firma IGEL Technology GmbH, https://www.igel.com/de/ derartige Geräte. Diese Rezentralisierung der IT Infrastruktur ist ein stark ausgeprägter Trend in allen großen Unternehmen und staatlichen Organisationen.

Als zentrale Server werden Windows, Linux, Unix und eben Mainframe Server eingesetzt. Die Hardware- und Software Kosten sind für große Windows Server am geringsten, Linux und Unix Server liegen in der Mitte, und Mainframe Server sind am teuersten.

Aber, Mainframes haben die mit Abstand besten und am weitesten entwickelten Werkzeuge für die System-Administration. Daher sind von allen großen Servern die Administrationskosten bei den Mainframes am Geringsten.

1.2.4 Das Problem der Skalierung

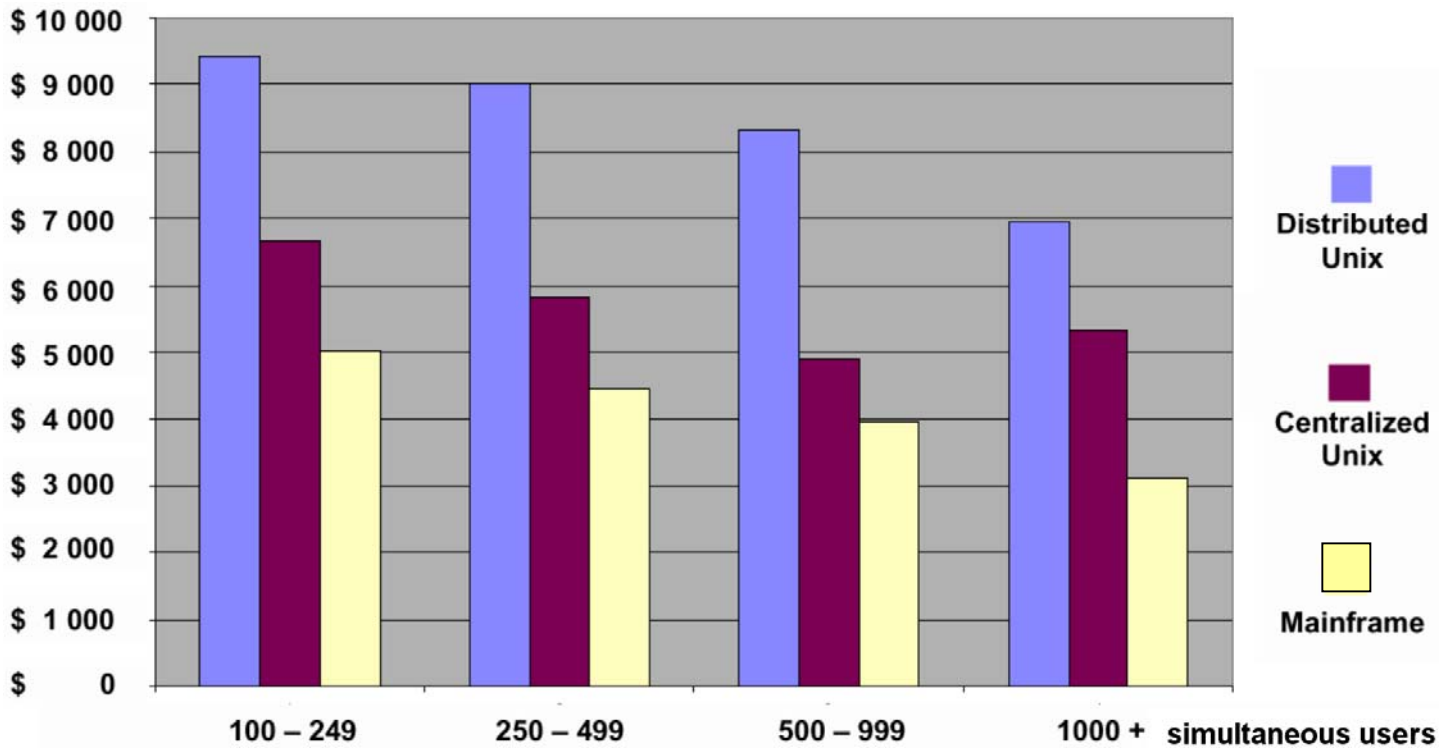


Abb. 1.2.3
Unix Rechner skalieren schlecht

In den Unternehmen existieren neben dem zentralen Mainframe zahlreiche kleinere Unix (und Windows) Server, die geografisch über das Unternehmen verstreut in den einzelnen Fachabteilungen zu finden sind. Es stellt sich heraus, dass eine Zentralisierung aller dezentralen Server im Rechenzentrum deutlich Kosten sparen konnte. Dies ist in der hier wiedergegebenen Grafik zu sehen. Aber .....

Nehmen wir an, Sie haben einen Server mit unzureichender Rechenleistung. Was machen Sie: Sie verdoppeln die Anzahl der CPUs, die Größe des Hauptspeichers, die Anzahl der Plattenspeicheranschlüsse usw.

Als Skalierung bezeichnet man die Eigenschaft eines Rechners, bei doppelt soviel Hardware Ressourcen die doppelte Leistung zu bringen. Manche Rechner skalieren überproportional: Doppelt soviel Hardware bringt mehr als doppelt soviel Leistung.

Die meisten Rechner skalieren unterproportional: Doppelt soviel Hardware bringt weniger als doppelt soviel Leistung.

Die Skalierungseigenschaften eines Rechners hängen von vielen Faktoren ab, z.B. Architektur, Hardware, Betriebssystem, Platten- und Bandspeicheranschlüsse (I/O Verhalten), Anwendungsprogrammen und vielen anderen Faktoren.

In der Abb. 1.2.3 ist zu sehen, dass bei zentralen Unix Servern für 1000 und mehr Benutzer die TCO wieder schlechter wird. Der Grund sind die schlechten Skalierungseigenschaften großer Unix Rechner. Mainframes besitzen dagegen hervorragende Skalierungseigenschaften.

Wir werden uns die Gründe hierfür später noch genauer ansehen.

1.2.5 Management Kosten

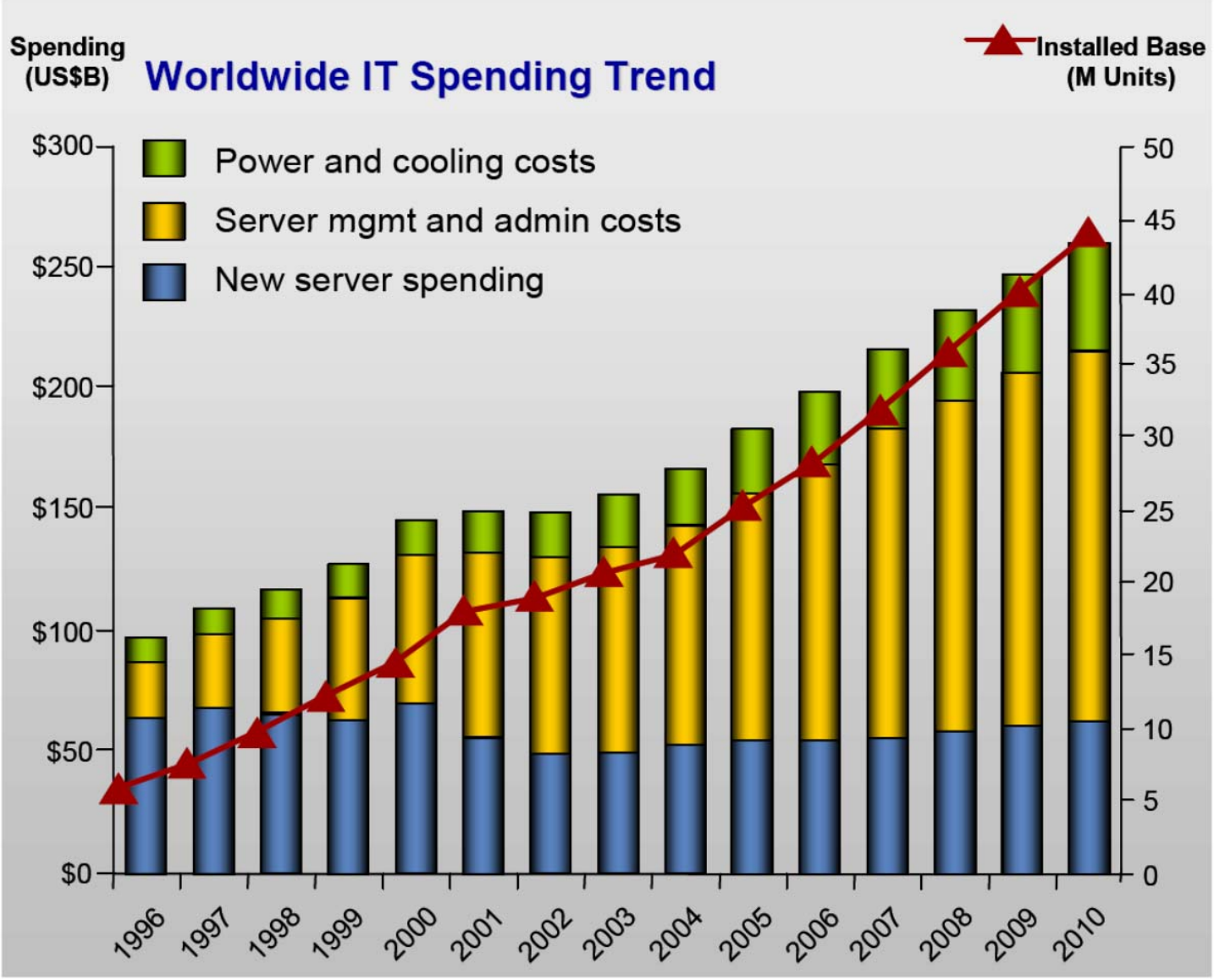


Abb. 1.2.4
Kostenentwicklung in den letzten 15 Jahren

Abb. 1.2.4 zeigt, dass die Anzahl der weltweit installierten Server in den letzten 15 Jahren stark gestiegen ist. Wegen der immer günstiger werdenden Kosten sind die Ausgaben für Softwarelizenzen und Hardware aber mehr oder weniger konstant geblieben. Die Administrations- und Management-Kosten (Kosten für Mitarbeiter, die den laufenden Betrieb der IT Infrastruktur aufrecht erhalten), sind in den letzten 15 Jahren überproportional gewachsen und sind heute sehr viel höher als die Kosten für die Hardware und Software.

Die Kosten für Energie und Klimatisierung steigen ebenfalls überproportional.

Dies erklärt, warum der Einsatz von Mainframes in den letzten Jahren trotz sehr hoher Hardware- und Softwarekosten ansteigt.

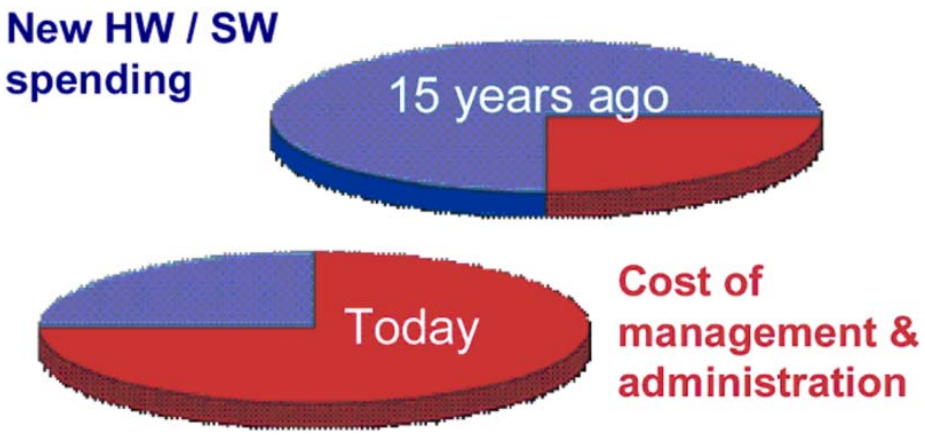


Abb. 1.2.5
Die Kostenentwicklung in den letzten 15 Jahren wird von anderen Quellen bestätigt

Die Grafik in Abb. 1.2.5 stammt von Tony Picardi, vom Marktforschungsunternehmen IDC. Sie wurde reproduziert in der Wochenzeitschrift The Economist, 28. Oktober 2004. Die Grafik bestätigt den stark anwachsenden Anteil der Personalkosten.

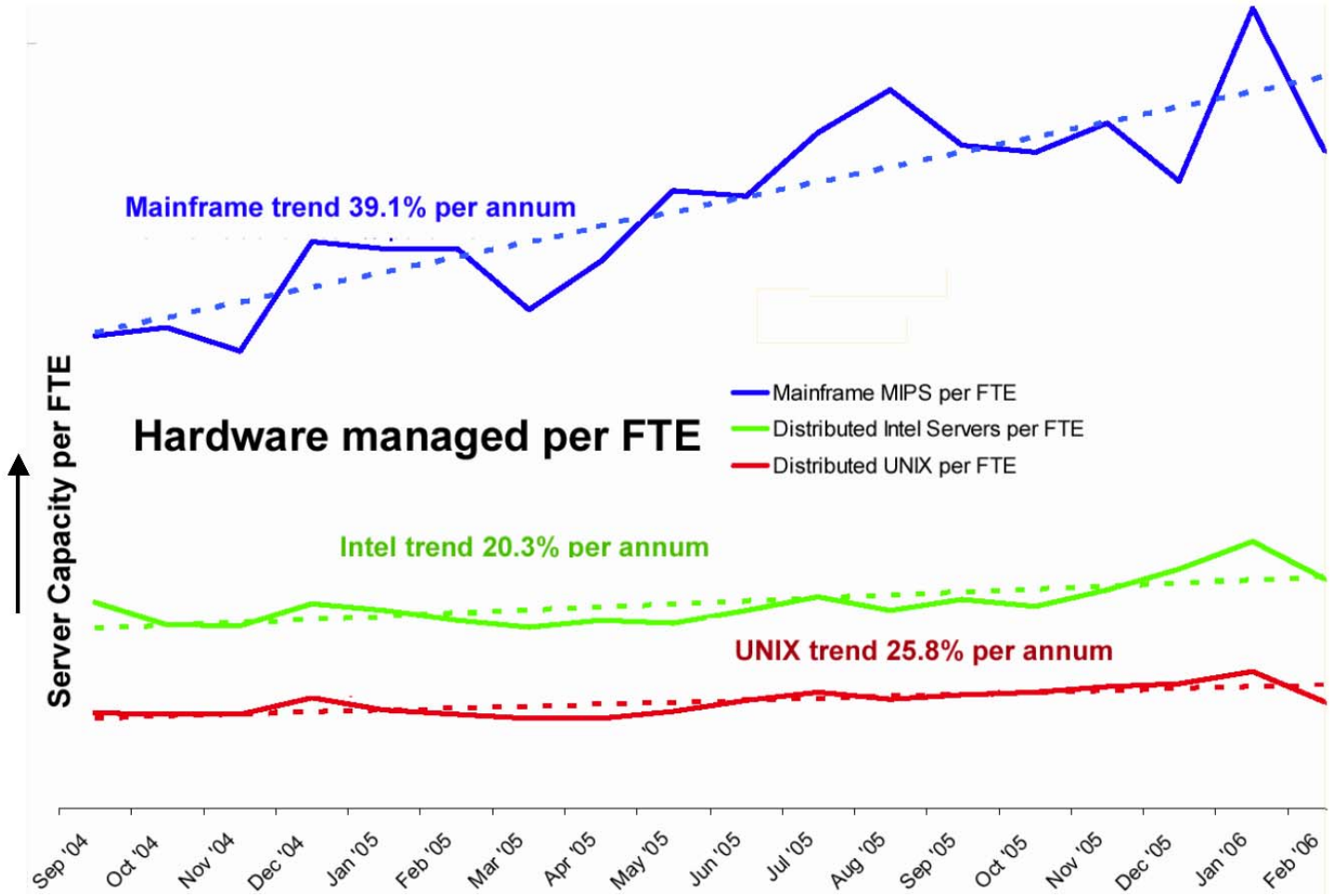


Abb. 1.2.6
Produktivitätswachstum der Mainframe Administration

Als “Full-Time Equivalent” (FTE) bezeichnet man den Prozentsatz an Zeit, in dem ein Mitarbeiter nützliche Arbeit leistet. Ein Vollzeit Job ist 1,0 und ein 50 % Teilzeit Job ist 0,5.

Die Produktivität eines FTE ist bei Mainframes stark gestiegen, während sie sich bei Windows und Linux/Unix Rechnern nur wenig verbesserte.

Abb. 1.2.7 zeigt, dass ein Mainframe Administrator heute wesentlich mehr Hardware betreuen kann als früher. Die Produktivität ist deutlich gestiegen. Bei Unix und Windows Administratoren ist ein sehr viel geringerer Produktivitätsgewinn festzustellen.

Der Grund ist, IBM gibt jährlich sehr viel Geld für die Verbesserung der Mainframe Wartungs- und Administrations-Infrastruktur aus.

IBM investiert jährlich etwa 1,2 Milliarden $ in die Entwicklung des „System z Stack“, eingeschlossen Hardware, Software, und Services.

Da die Software etwa 65 % des Mainframe Umsatzes generiert, wird etwa 65 % der 1,2 Milliarden Investition in die Weiterentwicklung der Software investiert.

1.2.6 Distributed Processing Kosten

Beispiel: Große Bank in der Schweiz 2Q2011.

In der IT Infrastruktur des Unternehmens wird der WebSphere Application Server eingesetzt. Das WebSphere Software Produkt ist für verschiedene Plattformen verfügbar.

In dem vorliegenden Fall wird WebSphere sowohl auf AIX (PowerPC Unix) als auch auf z/OS eingesetzt. Etwa 2/3 der Work Load läuft unter AIX, etwa 1/3 unter z/OS.

Etwa 7 Mitarbeiter werden für die Administration und Wartung der WebSphere AIX Installation benötigt. Die Administration und Wartung der WebSphere z/OS Installation wird von einem Mitarbeiter nebenbei (etwa 20% seiner Arbeitszeit) mit erledigt.

Auch wenn man Sonderfaktoren unterstellt, ist der Unterschied in den Administrationskosten erheblich.

Es existieren allerdings eine Reihe von Schwierigkeiten, einen durch eine TCO Analyse ermittelten potentiellen Kostenvorteil umzusetzen. Hierfür existieren mehrere Gründe:

- **Eine distributed Computing Infrastruktur bietet Flexibilitätsvorteile. Führungskräfte auf der mittleren Management Ebene wollen hierauf nur ungern verzichten**
- **Betroffene Fachkräfte werden überflüssig und müssen umgeschult werden. Betriebsrat und Gewerkschaft müssen konsultiert und beteiligt werden.**
- **Es fehlen Mainframe Spezialisten.**
- **Weitere Schwierigkeiten beim Umsetzen der Kostenvorteile sind in http://www.cedix.de/VorlesMirror/Band1/TCO01.pdf beschrieben.**

All dies bedeutet, dass die Rezentralisierung in Richtung Mainframe ein langwieriger Prozess ist, der sich über viele Jahre erstrecken wird.

1.2.7 Mainframe Auslastung

... und was für Anwendungen laufen auf einem Mainframe ?

Die Auslastung für ein typisches zSeries System besteht aus:

55%	**„Legacy „ Anwendungen (Anwendungen, die vor längerer Zeit entstanden sind)**
35 %	**Anwendungen, die in den letzten 1 - 2 Jahren geschrieben wurden**
10%	**Anwendungen, die im Rahmen von Konsolidierungsmaßnahmen von anderen Servern übernommen wurden, davon viele von Unix Rechnern**

Die große Mehrzahl der neu geschriebenen Anwendungen verwenden die Programmiersprache Cobol. Die Firma IBM übt starken Druck auf ihre Mainframe-Kunden aus, neue Anwendungen bevorzugt in Java zu schreiben, ist damit aber nur mäßig erfolgreich. Viele Führungskräfte in den Unternehmen glauben, dass für unternehmenskritische Anwendungen Cobol eine höhere Produktivität ermöglicht als Java.

Es existieren viele Meinungen, aber kaum wissenschaftliche Untersuchungen bezüglich eines Produktivitätsvergleiches Cobol versus Java. Vorhandene Indizien deuten darauf hin, dass die Produktivität von Cobol tatsächlich recht gut ist.

Die Firma Microfocus berichtet über steigende Umsätze für ihre Windows und Unix Cobol Compiler.

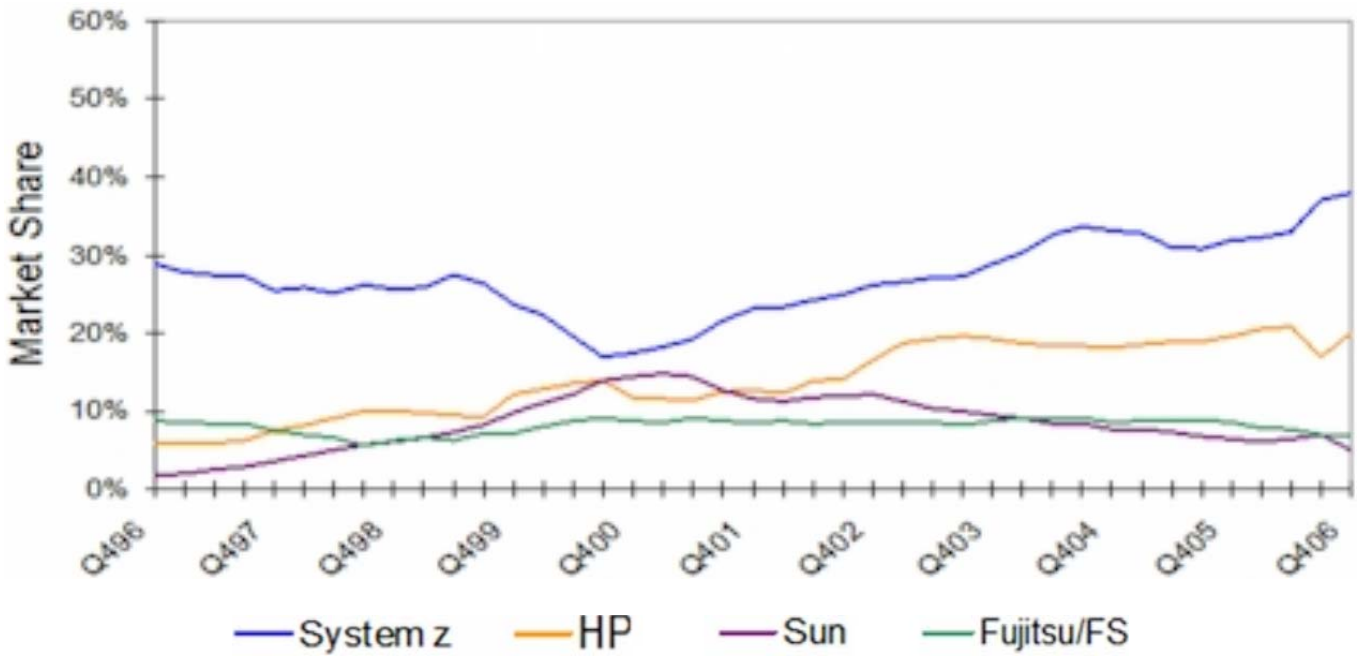


Abb. 1.2.7
Entwicklung der Marktanteile

Diese Investitionen erklären die wachsende Bedeutung von System z und z/OS. Der hier gezeigte Trend hat sich bis heute fortgesetzt.
http://www.cedix.de/VorlesMirror/Band1/Marketshare.pdf

1.2.8 Neue Installationen

Neue Mainframe Installationen bei Unternehmen, die bisher noch keinen Mainframe hatten, sind selten, da fast alle großen Firmen bereits seit vielen Jahren einen Mainframe als zentralen Server einsetzen. Neue Mainframe Installationen finden deshalb besonders in Entwicklungsländern wie z.B. China, Indien oder Russland statt.

Versucht man in solchen Ländern die veraltete IT-Infrastruktur eines Unternehmens zu modernisieren, existieren in vielen Fällen zum Mainframe keine Alternativen.

Trotzdem finden auch in Europa und den USA Neuinstallationen statt. Ein Grund ist, dass Mainframes über zahlreiche Funktionseigenschaften verfügen, die auf anderen großen Systemen nicht erhältlich sind.

1.2.9 Altersverteilung der Spezialisten

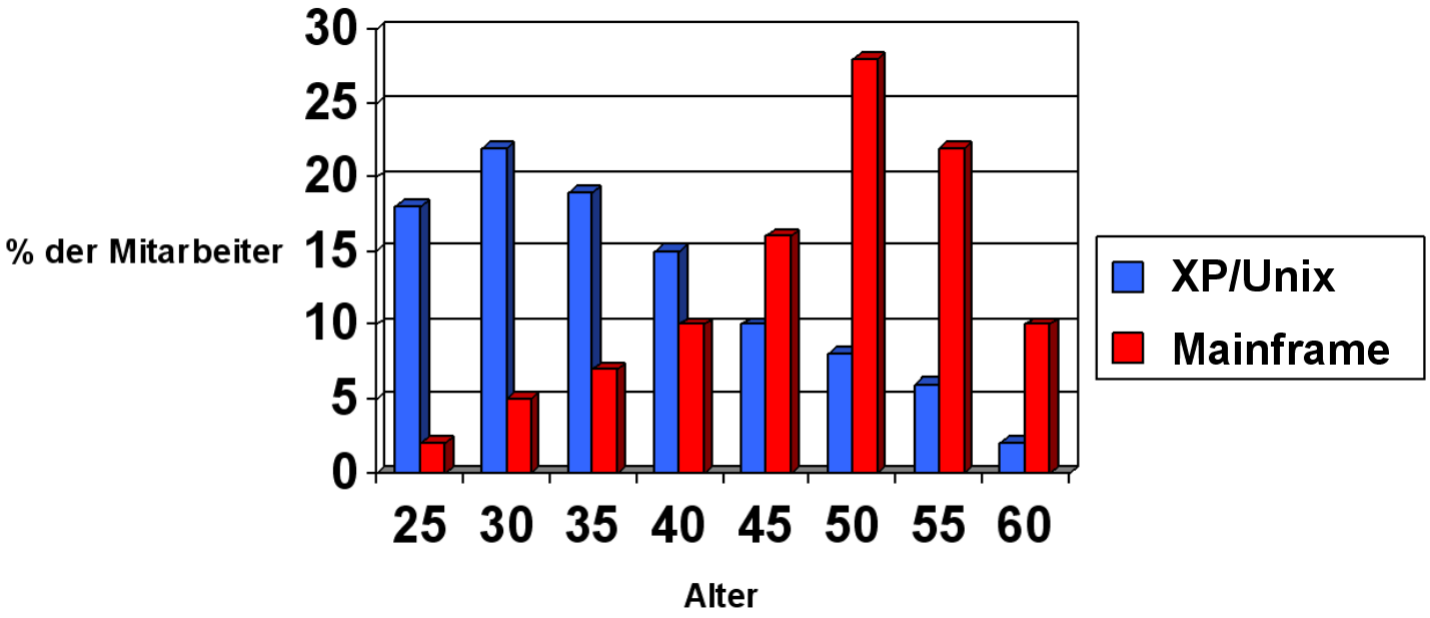


Abb. 1.2.8
Altersstruktur der Mainframe Spezialisten

Dies ist das größte Problem für die Zukunft der Mainframes. Die Unternehmen haben in den letzten 10 – 20 Jahren hauptsächlich Unix und Windows Spezialisten, aber kaum Mainframe Spezialisten eingestellt. Da Mainframes an den Hochschulen als veraltet gelten, wird kaum Mainframe Nachwuchs ausgebildet.

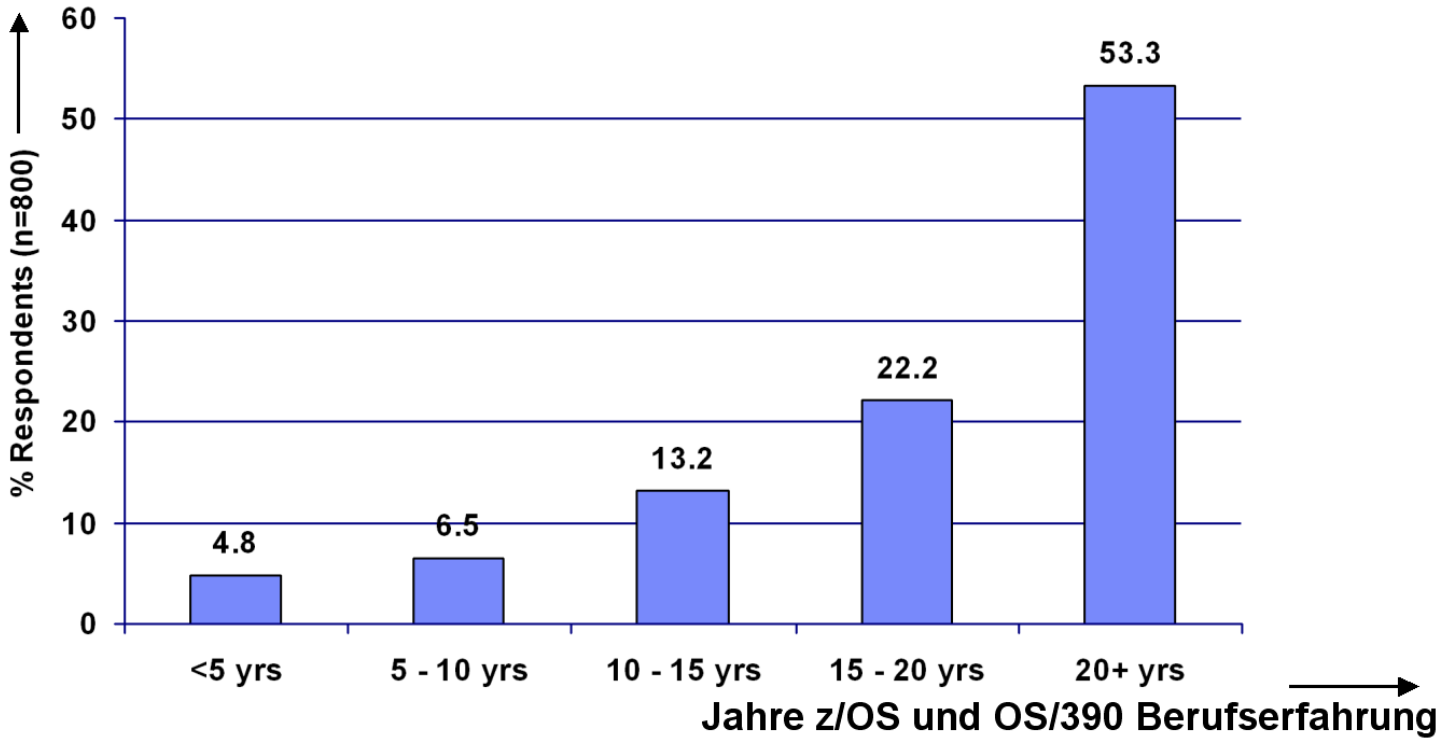


Abb. 1.2.9
Berufserfahrung der Mainframe Spezialisten

Abb. 1.2.9 bestätigt das Bild. Hochschulabsolventen mit Mainframe Kenntnissen haben hervorragende Berufsaussichten.

Dies wird durch die folgenden Beispiele untermauert.

Arbeitsmarkt Großrechnerexperten

Wer Mainframe und Java kombiniert, ist Gold wert

Datum: 07.06.2010
Autor(en): Gabi Visintin
URL: http://www.computerwoche.de/1937465

Großrechnerexperten mit Kenntnissen moderner Programmiersprachen haben auf dem Arbeitsmarkt gute Chancen.

Wer im IT-Bereich auf **Stellensuche** ist, stößt auch auf dieses Angebot: "Diese Perspektive bieten wir Ihnen: Sie sind für den Betrieb und die Optimierung von IT-Systemen und zugehöriger Prozessabläufe im **Großrechenzentrum** und im Produktionsbetrieb zuständig. Neue Konzepte für Ablaufprozesse und Produktionssystem setzen Sie um und achten dabei besonders auf Effizienz, **Sicherheit** und Verfügbarkeit." In der **Stellenanzeige** der **Datev** , eines Softwarehauses und IT-Dienstleisters aus Nürnberg, sind die letztgenannten drei Begriffe die Synonyme für die wichtigsten Eigenschaften eines Großrechners: Effizienz, Sicherheit und Verfügbarkeit.

Sichere Großrechner

Diese drei Stichworte sind der Grund dafür, warum die "Dinosaurier der Rechnerwelt" immer noch überall dort stehen, wo betriebskritische Daten und hohe Transaktionsvolumina verarbeitet werden: in Finanzinstituten und Versicherungen, bei Behörden und in Web-Zentren. Nach **IBM** -Angaben arbeiten die 50 weltweit führenden Banken mit **Mainframes** . 22 der Top-25-Einzelhändler in den USA zählt der IT-Konzern zum Anwenderkreis seines Großrechnersystems z.

Abb. 1.2.10
Computerwoche 26.5.2010

Executive Summary

Forrester conducted interviews with high-level managers and CIOs/CTOs of North American and European companies with revenues of more than $1 billion to better understand their use of the mainframe, their mainframe strategies, and their perceptions of factors that might impede the expansion of mainframe usage in their own organizations and/or the marketplace at large.

Key Findings

Forrester's study yielded four key findings:

- The mainframe is generally viewed as the most efficient, scalable, and reliable computing platform today. A majority of respondents said that they could not process a typical mainframe workload on any other platform. As the most recent mainframe platform provides a very efficient way to rationalize and consolidate their data centers, a majority of mainframe sites have started consolidation programs using Open Systems on the mainframe.

- All respondents recognized that a mainframe operation requires detailed and specialized knowledge. As the mainframe specialists are aging and as college and university graduates do not typically receive a mainframe education, there is a perceived risk of skill shortages despite education programs sponsored by several mainframe solution vendors.

Abb. 1.2.11

Aus einer Untersuchung des Marktforschungsunternehmens Forrester Consulting, November 2, 2009, „The Mainframe Opportunity“

Fachkräftemangel beim Mainframe

14.11.2012 | von Werner Kurzlechner

Laut einer BMC-Studie klagen Anwender klagen über fehlende Skills, während sich der IT-Nachwuchs kaum für Mainframes begeistern kann.

So haben sich die Prioritäten im Bereich der Anwendungsentwicklung in den vergangenen Jahren entwickelt.
Foto: BMC Software

Die IT ist fraglos ein Feld mit beschleunigten Innovations- und Evolutionszyklen. Aber nicht jeder Dinosaurier stirbt dort schnell aus. Mainframes beispielsweise sind keineswegs ein Relikt aus der Vergangenheit, sondern in fast allen Unternehmen weiterhin als Zukunftsmotor eingeplant. Das geht aus einer Studie des Softwarehauses BMC hervor, für die mehr als 1200 Entscheidungsträger befragt wurden. Ein Drittel der Teilnehmer kommt aus der Region EMEA.

Das Kernergebnis der Studie: 90 Prozent der Unternehmen betrachten Mainframe als langfristige Business-Lösung. Allerdings schränken zwei Fünftel diese Zukunftsperspektive auf bestehende Anwendungen ein. Die Hälfte sieht demgegenüber auch Spielraum für Wachstum und neuen Workload.

59 Prozent rechnen mit einem MIPS-Zuwachs. Davon 15 Prozent gehen von einem Wachstum der Maschinenbefehle pro Sekunde um mehr als 10 Prozent aus. 19 Prozent schreiben das alleine ihren Legacy-Anwendungen zu, 9 Prozent ausschließlich neuen Applikationen, 31 Prozent machen beide für die Erwartung verantwortlich. Einschränkend lässt sich jedoch festhalten, dass sich hier an der Einschätzung der Befragten gegenüber den Vorjahren kaum etwas verändert hat.

Abb. 1.2.12
Computerwoche 14.11.2012

Weitere Einzelheiten zum Thema Berufsaussichten und Mainframe Kenntnisse finden sie unter http://cedix.de/beruf/index.html .

1.3 Mainframe Architektur

1.3.1 Was ist eine Rechner Architektur ?

Als Rechner Architektur bezeichnen wir das Erscheinungsbild eines Rechners wie ihn der Assembler (oder Maschinensprachen-) Programmierer sieht.

Heute Prozessoren verwenden nur noch wenige Rechner Architekturen. Die wichtigsten sind:

- **x86 (PCs und Laptops, größtenteils von Intel und AMD hergestellt)**
- **ARM (Mobiltelefone, viele embedded Systems, wachsende Bedeutung)**
- **PowerPC (IBM Unix "System P" Rechner, Sony Playstation, CISCO Internet Router, Microsoft X-Box, Nintendo, embedded Systems in der Automobilindustrie, werden von Freescale und IBM hergestellt)**
- **Sparc (wird in großen Sun/Oracle und Fujitsu Rechnern eingesetzt)**
- **Itanium (wird in großen Hewlett Packard Unix Rechnern eingesetzt)**
- **System z (ausschließlich in Mainframes eingesetzt, heute fast nur noch von IBM hergestellt)**

1.3.2 An Architecture to stand the Test of Time

Für die Entwicklung der Mainframes ist eine Betrachtung des historischen Hintergrundes interessant. Am 7. April 1964 kündigte IBM die S/360-Rechnerfamilie (System 360) an. Die Ziffer „360" bezog sich dabei auf alle Richtungen eines Kompasses, um die universelle Anwendbarkeit, den weitgefächerten Bereich von Performance und Preis der Produkte sowie die Entwicklungsrichtungen des Unternehmens zu demonstrieren. Die Familie bestand ursprünglich aus 6 Zentraleinheiten und etwa 45 weiteren Ein-/Ausgabeeinheiten.

Die Existenz der S/360-Architektur ist den drei genialen Wissenschaftlern

Gene Amdahl,
Gerry Blaauw,
Fred Brooks

und der damaligen IBM-Entwicklungs-Organisation unter der Leitung von IBM-Vizepräsident B.O. Evans zu verdanken, siehe http://www.cedix.de/Literature/History/index.html .

Die Einführung der S/360-Architektur stellt einen Meilenstein in der Entwicklung des Computers dar. Zum damaligen Zeitpunkt waren die Unterschiede in den Rechner-Architekturen der einzelnen Hersteller sehr viel größer als dies heute der Fall ist. Viele Eigenschaften, die heute als selbstverständlich gelten, entstanden mit der Einführung der S/360-Architektur.

Einige von vielen Beispielen sind:

- **Die Entscheidung, dass ein Byte eine Länge von 8 Bit hat (und nicht z.B. 6 oder 7 Bit),**
- **die Tatsache, dass die Einheit der Hauptspeicheradressierung das Byte ist (und nicht ein 24-, 36- oder 48-Bit langes Wort),**
- **die Einführung von Mehrzweckregistern, die gleichzeitig als Adressregister und als Datenregister dienen,**
- **der Verzicht auf die direkte Hauptspeicher-Adressierung,**
- **der Unterschied zwischen Kernel (Supervisor)- und User (Problem)-Status,**
- **die Einführung des S/360-Kanals, der noch heute in der Form der SCSI und FICON Interfaces weiterlebt.**

Ein weiterer Meilenstein war die bewusste Entscheidung, die S/360-Architektur für eine sehr lange Lebenszeit auszulegen. Diesem dienten vor allem zwei Maßnahmen:

- **Die Verpflichtung und Garantie, dass Maschinencode auf allen damaligen sowie zukünftigen Rechnermodellen unverändert lauffähig sein würde. Diese Strategie wurde bis zu den heutigen System z Modelle eingehalten.**

- **Die Einrichtung eines Architektur-Boards mit der Aufgabenstellung, die S/360-Architektur nach wissenschaftlichen Grundsätzen weiterzuentwickeln.**

Diese und viele andere Entscheidungen von Amdahl, Blaauw und Brooks erwiesen sich als außerordentlich tragfähig und haben dazu geführt, dass sich die System z Architektur auch heute noch fortschrittlich und zukunftsorientiert darstellt.

Es existieren nur wenige Architektureigenschaften, die man mit dem heutigen Wissensstand anders und/oder besser machen würde. Dies ist besonders bemerkenswert, weil viele später entwickelte Rechnerarchitekturen mit neuartigen Entwicklungen versprochene Verbesserungen nicht liefern konnten, und in der Zwischenzeit wieder von der Bildfläche verschwunden sind. Ein Beispiel hierfür ist die Entwicklung der VAX-Architektur durch die Firma Digital Equipment Corporation (DEC).

DEC war in den 80er Jahren der weltweit zweitgrößte Computer Hersteller nach IBM. Im Jahre 1976 entwickelte DEC eine komplett neue Rechner-Architektur, die erheblich besser als die S/370 Architektur sein sollte. Nach größeren Anfangserfolgen musste DEC die VAX Architektur aufgeben, weil sie gegenüber S/370 schlicht nicht wettbewerbsfähig war. Die Firma Digital Equipment Corporation hat sich von diesem Desaster nie wieder erholen können.

1991 löste die Firma DEC ihre VAX-Architektur durch die Alpha-Architektur ab.

In dem Vorwort des Alpha-Architektur-Handbuches wurde explizit darauf hingewiesen, dass man die gleichen Entwurfsprinzipien angewendet habe, die von Amdahl, Blaauw und Brooks 1964 für die Einführung von S/360 entwickelt wurden:

The Alpha architecture is a RISC architecture that was designed for high performance and longevity. Following Amdahl, Blaauw, and Brooks, we distinguish between architecture and implementation:

Computer architecture is defined as the attributes of a computer seen by a machine language programmer. This definition includes the instruction set, instruction formats, operation codes, addressing modes, and all registers and memory locations that may be directly manipulated by a machine-language programmer.

Implementation is defined as the actual hardware structure, logic design, and datapath organization.

This architecture book describes the required behaviour of all Alpha implementations, as seen by the machine-language programmer.

a) Abb. 1.3.1

Alpha Architecture Reference Manual, Digital Press, Digital Equipment Corporation, 1992

1.3.3 Lebensdauer von Anwendungssoftware

Ein wichtiger Unterschied zwischen Enterprise Computing und anderen Einsatzgebieten der Informatik ist die erwartete Lebensdauer der damit entwickelten Anwendungen. Bei manchen Smartphones und Tablets wird akzeptiert, dass Software für ältere Modelle aufgrund deren begrenzten Langzeitsupports nicht mehr verfügbar ist. Der ideale Kunde eines Smartphone Herstellers ersetzt sein Mobiltelefon alle 24 Monate durch ein neues, verbessertes Modell, obwohl das alte Modell noch voll funktionsfähig ist. Inkompatibilitäten zwischen aufeinander folgenden Software Versionen sind in dieser Situation wegen des schnelllebigen Marktes kein gravierendes Problem.

Im Gegensatz dazu hat Enterprise Software eine Lebensdauer, die in Jahrzehnten gemessen wird. Ein Testfall waren die „Jahr 2000“ (y2k) Umstellungen von zweistelligen auf vierstellige Jahreszahlen. Obwohl der erforderliche Umstellungsaufwand von weltweit 300 Mrd. $ ein guter Anlass gewesen wäre, veraltete Software durch neuere, moderne Versionen zu ersetzen, ist dies fast nirgendwo geschehen.

In der Vergangenheit verstand man unter dem Begriff „Legacy Software“ viele Jahre alte Anwendungen, die auf Mainframe Rechnern ausgeführt wurden. In zunehmendem Maße existieren in den Unternehmen jedoch Legacy Anwendungen, die auf Windows (in unterschiedlichen Versionen), Linux, AIX, HP_UX, Solaris, Tru64 UNIX und vielen weiteren Betriebssystemen laufen.

Es wird geschätzt, dass Unternehmen etwa 50 % ihres jährlichen Budgets für Anwendungssoftware in Neuentwicklungen und 50 % in die Administration und Pflege von Legacy Software investieren.

1.3.4 Register Architektur

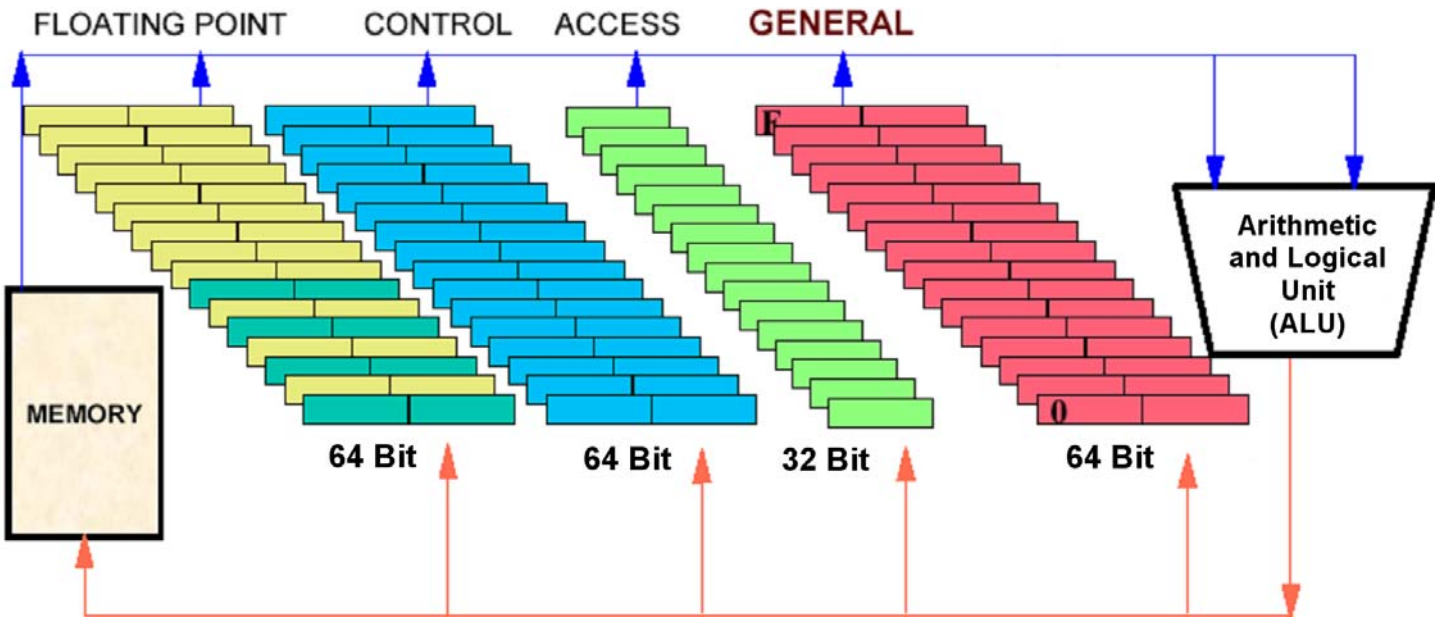


Abb. 1.3.2
System z Programmiermodell

Die System z Architektur ist eine 64 Bit Architektur und verfügt über je 16 Mehrzweckregister (General Purpose Register), Gleitkomma- und Steuerregister (Control Register) mit einer Länge von je 64 Bit, und 16 Access Registern mit einer Länge von je 32 Bit. Ähnlich wie x86, PowerPC und Sparc Architekturen ist sie aus einer ursprünglichen 32 Bit (31 Bit) Version weiterentwickelt worden. Hierbei unterstützen die System z-Rechner einen 31-Bit- und einen 64-Bit-Modus. Das Umschalten zwischen beiden Modi ist sehr einfach.

Anmerkung: der ursprüngliche System z Vorläufer benutzte 31 an Stelle von 32 Adressier-Bits.

1.3.5 Anzahl der Register mehrerer Mainstream Architekturen

Architecture	Integer registers	Double FP registers
x86	8	8
x86-64	16	16
IBM/360	16	4
Z/Architecture	16	16
Itanium	128	128
UltraSPARC	32	32
POWER	32	32
Alpha	32	32
6502	3	0
ARM	16	16

Abb. 1.3.3
Register Anzahl unterschiedlicher Architekturen

**Eine zu große Anzahl von Registern verringert die Performance, weil bei jedem Prozesswechsel der Inhalt der Register abgespeichert (saved) und neue Inhalte geladen werden müssen. Optimal sind durchschnittlich 25 Register,
abhängig vom Profile der bearbeiteten Anwendungen. Da 25 keine Binärziffer ist, sind 16 oder 32 Register optimal.**

1.3.6 16 und 32 Bit Maschinenbefehle

Die System z Maschinenbefehle haben alle eine einheitliche Länge von entweder 16 Bit, 32 Bit oder 48 Bit. Vor allem der geschickte Entwurf der 16 Bit Maschinenbefehle führt dazu, dass ein kompiliertes Programm auf einem System z Rechner fast immer 20 – 30 % weniger Platz im Hauptspeicher benötigt, als bei fast allen anderen Rechner Architekturen. Dies gilt spezifisch auch bei einem Vergleich der System z Architektur gegenüber der x86 Architektur.

Vor allem auf Grund der reduzierten Cache Bandbreite ist dies ein Performance Vorteil.

Es ist interessant zu sehen, dass die Firma Arm Holdings mit der Einführung der Thumb Technologie Erweiterung für die ARM Architektur ein ähnliches Ziel anstrebt. Vor allem die jüngste Thumb-2 Technologie Erweiterung weist an dieser Stelle überraschende konzeptuelle Ähnlichkeiten mit der 1964 entstandenen S/360 Architektur auf.

Hiermit wiederholt sich die Frage: Wie konnte es sein, dass Amdahl, Blaauw und Brooks beim Entwurf der S/360 Architektur im Jahre 1964 alles richtig gemacht haben ?

1.3.7 Decimal Floating Point

System z unterstützt drei Gleitkommaformate: Hexadezimal, IEEE 754 und Dezimal.

Das hexadezimale Format ist ein IBM proprietärer Binär-Standard und fast nur auf Mainframe Rechnern anzutreffen. Die meisten auf Mainframes gespeicherten Gleitkommadaten verwenden das hexadezimale Format.

Das IEEE 754 Format ist international weit verbreitet, und wird u.a. von der x86 Architektur benutzt. Auf Mainframes benutzt vor allem das zLinux Betriebssystem diesen Standard.

Das Dezimale Gleitkommaformat (DFP) ist erst seit wenigen Jahren verfügbar. Im Gegensatz zu den beiden anderen Formaten werden Mantisse und Exponent mit Dezimalziffern dargestellt. Dies führt zu einer gewissen Performance- und Genauigkeitseinbuße.

Die Benutzung von Dezimal Arithmetik ist in der Wirtschaft weit verbreitet. DFP vermeidet Rundungsfehler und andere Probleme bei der Konvertierung von Dezimaldaten in Binärdaten und umgekehrt, und gewinnt deshalb an Bedeutung.

Die Gleitkomma Einheit des System z Mikroprozessors unterstützt alle drei Gleitkomma-Arten. Ebenso speichert die z/OS DB2 Datenbank Daten in allen drei Formaten.

1.3.8 ASCII- und EBCDIC

Es existieren drei weit verbreitete Standards für die Darstellung alphanumerischer (Buchstaben, Ziffern, Sonderzeichen) Daten.

Die Darstellung von alphanumerischen Zeichen geht bei allen Rechnerarchitekturen auf uralte Wurzeln zurück. Bei vielen Rechnern ist dies die 7-Bit-ASCII-Darstellung (American Standard Code for Information Interchange), die ihren Ursprung in den Lochstreifen der Teletype-Maschinen hat und nachträglich auf 8 Bit erweitert wurde. Bei den Mainframe- (und einigen anderen) Rechnern ist dies die 8-Bit-EBCDIC-Darstellung (Extended Binary Coded Decimal Interchange Code, ausgesprochen [ebsidik]), die ihren Ursprung in den Lochkarten hat. Das Ergebnis ist eine Spaltung der IT-Welt. Etwa 60 % aller geschäftlich relevanten alphanumerischen Daten sind im EBCDIC-Format gespeichert und etwa 40 % im ASCII-Format. Wenn immer ASCII - und EBCDIC-Rechner miteinander kommunizieren, sind unschöne Konvertierungsverfahren erforderlich.

Mainframes unterstützen neben dem EBCDIC- auch das ASCII-Format. Letzteres wird z.B. von dem zLinux Betriebssystem genutzt, ist aber sonst nur wenig gebräuchlich. Man kann davon ausgehen, dass alphanumerische Daten auf einem Mainframe in der Regel im EBCDIC-Format vorliegen.

ASCII-Tabelle

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	Ä	Ö	Ü	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	ä				

EBCDIC-Tabelle

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	PF	HT	LC	DEL			SMM	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	TM	RES	NL	BS	IL	CAN	EM	CC	CU1	IFS	IGS	IRS	IUS
2	DS	SOS	FS		BYP	LF	ETB	ESC			SM	CU2		ENQ	ACK	BEL
3			SYN		PN	RS	UC	EOT				CU3	DC4	NAK		SUB
4	SP										¢	.	<	(	+	|
5	&										!	$	*	)	;	¬
6	-	/										,	%	_	>	?
7											:	#	@	'	=	"
8		a	b	c	d	e	f	g	h	i						
9		j	k	l	m	n	o	p	q	r						
A		~	s	t	u	v	w	x	y	z						
B										`						
C		A	B	C	D	E	F	G	H	I						
D		J	K	L	M	N	O	P	Q	R						
E	\		S	T	U	V	W	X	Y	Z						
F	0	1	2	3	4	5	6	7	8	9						

Abb. 1.3.4
ASCII- und EBCDIC Code Tabellen

Beispiele: **ASCII** **R = Hex 52 ;**
EBCDIC **R = Hex D9 ;**

Neben EBCDIC und ASCII können Mainframes auch Daten im UTF-8 bzw. UTF-16 Format verarbeiten. Die Java Plattform verwendet UTF16 in Character Arrays und Strings.

1.3.9 Big Endian versus Little Endian

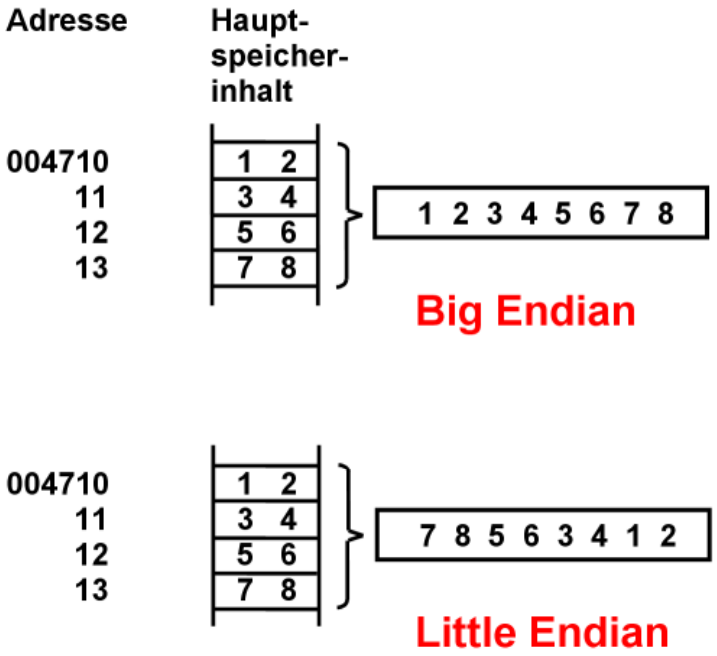


Abb. 1.3.5
b) Anordnung der Bytes im Hauptspeicher

Eine Hauptspeicheradresse adressiert ein einziges Byte im Hauptspeicher. Unter der „Endianess“ eines Rechners versteht man die Reihenfolge, in der eine Gruppe von Bytes im Hauptspeicher abgespeichert werden.

Nehmen wir einen Maschinenbefehl an, der 4 aufeinanderfolgende Bytes in ein Mehrzweckregister der Zentraleinheit ladet. Werden die Bytes in aufsteigender oder in absteigender Reihenfolge geladen ?

Wenn Halbworte oder Worte im Hauptspeicher gespeichert sind, dann befindet sich an der adressierten Hauptspeicherstelle:

- **Das wertniedrigste Byte bei Little Endian Rechnern**
- **Das werthöchste Byte bei Big Endian Rechnern**

Die Bytes eines Halbwortes oder Wortes werden bei Little Endian Rechnern in umgekehrter Reihenfolge abgespeichert wie bei Big Endian Rechnern.

Dies ist in Abb. 1.3.5 dargestellt. Es enthält die Hauptspeicheradresse 004710

- **das werthöchste Byte bei der Big Endian Architektur**
- **das wertniedrigste Byte bei der Little Endian Architektur**

Mainframes, die meisten Unix Rechner und das Internet verwenden die Big Endian Architektur. x86 und Ethernet verwenden die Little Endian Architektur.

Für Konvertierungszwecke enthält die x86 Architektur einen „Byte Swap“ Maschinenbefehl, mit dessen Hilfe die Endianess eines Feldes im Hauptspeicher umgedreht werden kann.

1.3.10 System z Speicherschutz

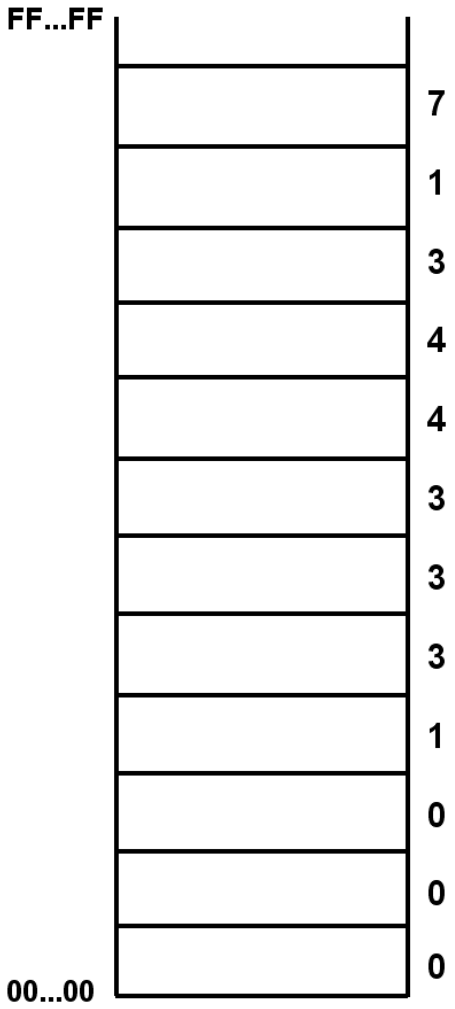


Abb. 1.3.6
Aufteilung des Hauptspeichers in 4 KByte Blöcke

Hardware-Speicherschutz (Storage Protection) ist eine einzigartige Einrichtung der Mainframes. Der Hardware-Speicherschutz teilt den Hauptspeicher in 4096 Byte große Blöcke auf und ordnet jedem dieser Blöcke einen 4 Bit Speicherschutzschlüssel (Storage Key) zwischen 0 ... 15 zu, der in einem getrennten Schnellspeicher abgespeichert wird. Der Speicherschutzschlüssel wird in einem 4-Bit-Feld des CPU Status Registers (Program Status Word, PSW*)* gespeichert. Bei jedem Speicherzugriff wird aus einem Schnellspeicher dieser Speicherschutzschlüssel ausgelesen und mit dem 4-Bit-Feld im PSW verglichen. Nur wenn dieser Vergleich positiv ausfällt (4-Bit-Felder sind identisch), erfolgt der Zugriff.

Die einzelnen Prozesse haben unterschiedliche Speicherschutzschlüssel. Somit wird verhindert, dass die Programme eines Prozesses auf Speicherbereiche eines anderen Prozesses zugreifen können.

Die hierfür erforderliche Logik ist in Abb. 1.3.7 dargestellt.

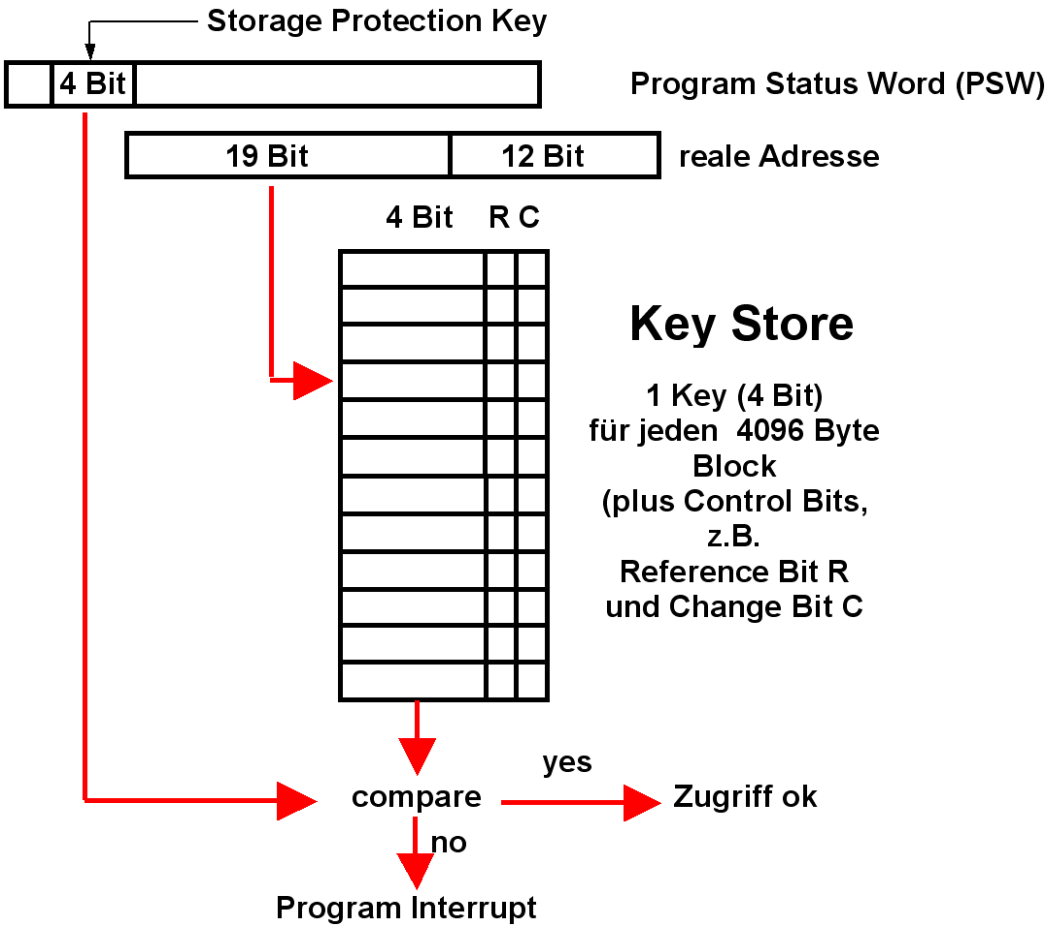


Abb. 1.3.7
Storage Keys

Bei jedem Hauptspeicher-Zugriff wird der Wert im Key Store für den adressierten 4 KByte Block mit dem Wert im PSW verglichen. Nur bei Übereinstimmung wird der Zugriff durchgeführt

Literatur: ABCs of z/OS System Programming, Volume 10, Section 1.24, S. 43
http://www.redbooks.ibm.com/redbooks/pdfs/sg246990.pdf

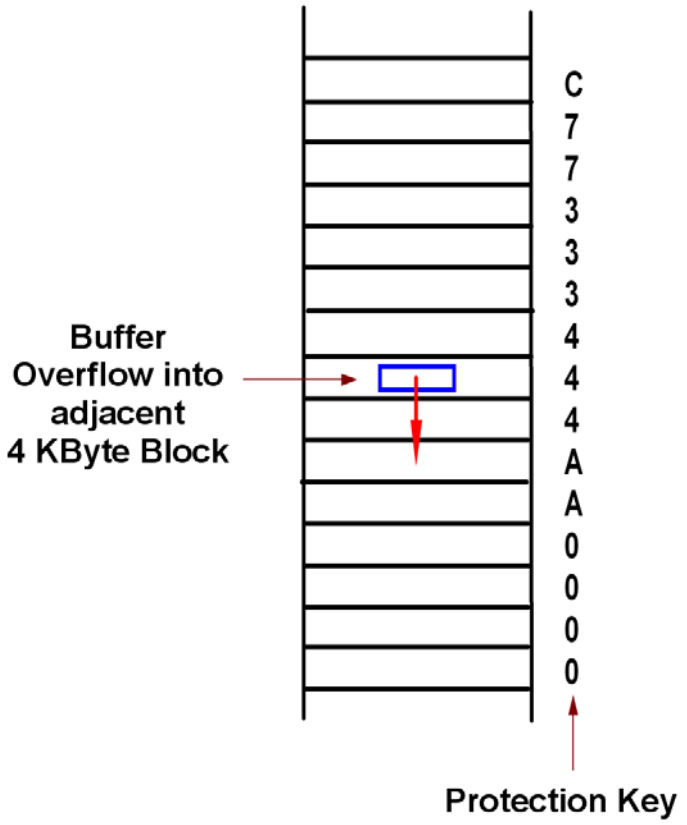


Abb. 1.3.8
Buffer Overflow Prevention

Der Hauptspeicher ist in 4 KByte große Blöcke mit unterschiedlichen Storage Keys aufgeteilt. Dies verhindert einen bekanntes Sicherheitsproblem, den Buffer-Overflow.

Versucht ein Prozess auf den (realen) Hauptspeicherbereich eines anderen Prozesses zuzugreifen, so verhindert die Storage Protection Einrichtung den Zugriff, wenn die Storage Keys ungleich sind.

Vor allem der Zugriff auf Betriebssystem-Kernel Funktionen wird damit ausgeschlossen.

1.3.11 Hardware Management Console

Ein PC verfügt über ein BIOS. Letzteres implementiert Maschinencode, der in einem Teil des Hauptspeichers liegt, auf den ein Benutzerprozess nicht zugreifen kann.

Eine äquivalente Mainframe-Funktion heißt Firmware , IBM intern auch als LIC (Licensed Internal Code) bezeichnet. Der Firmware-Bereich ist außerhalb des Adressbereichs der Maschinenbefehle im Hauptspeicher untergebracht. Er wird beim Hochfahren eines Rechners durch einen als IML (Initial Microcode Load) bezeichneten Vorgang in den Speicher geladen.

Firmware hat zahlreiche Funktionen. Er implementiert z.B. manche komplexe Maschineninstruktionen oder umfangreiche Diagnostik- und Fehlerbehandlungs-Funktionen. Die in Abschnitt 12.3 beschriebene PR/SM-Einrichtung wird ebenfalls mittels Firmware implementiert.

Wenn man beim Hochfahren eines PC eine Funktionstaste drückt, kann man BIOS-Funktionen aufrufen. Beim Mainframe ist diese Funktionalität sehr viel umfangreicher, während des laufenden Betriebs verfügbar, und wird als „Operator Facilities“ bezeichnet. Ein System-Administrator kommuniziert mit Hilfe der Operator Facilities mit dem Rechner. Er erledigt damit Aufgaben wie das Setzen von Datum und Zeit, das Reset von Subsystemen, Architectural Mode Selection, das Eingreifen in einen ausführenden Programmablauf oder ein Reagieren auf Maschinenfehler-Unterbrechungen. Ebenfalls dazu gehört die Funktion eines Boot-Managers, beim Mainframe als Initial Program Load (IPL) bezeichnet. Mainframe-Rechner verfügen über ein als „Hardware Management Console“ (HMC) bezeichnetes Bildschirm-Gerät, das ausschließlich der Nutzung der Operator Facilities durch den System-Administrator dient.

1.3.12 Control Unit

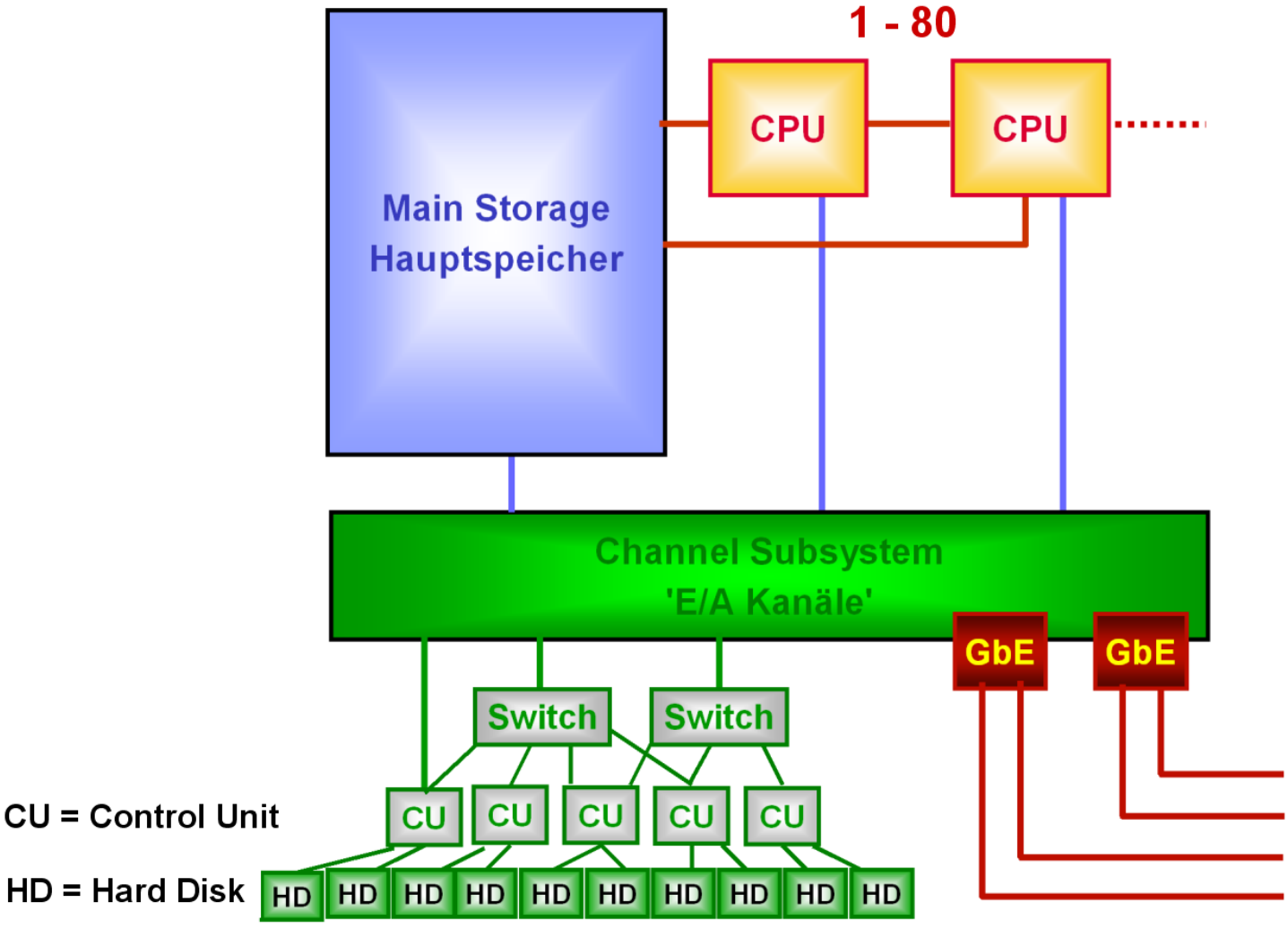


Abb. 1.3.9
Hardware Struktur eines Mainframe Systems

Ein Mainframe System besteht zumindest aus einer oder mehreren (bis zu 101) Prozessoren, einem Hauptspeicher, und Anschlüssen für Ein/Ausgabe (E/A) Geräte, besonders Festplattenspeicher (Hard Disk) und Netzwerkanschlüsse, häufig Gigabit oder 10 GBit Ethernet (GbE). Plattenspeicher und andere Input/Output Geräte (I/O) werden grundsätzlich über Control Units (CU) angeschlossen.

Bei einem PC erfolgt die Ansteuerung eines Plattenspeichers durch eine als I/O Driver bezeichnete Komponente des Betriebssystems. Die Ausführung des I/O Driver Codes benötigt CPU Zeit.

An viele PCs ist nur ein einziger Plattenspeicher angeschlossen. An einen Mainframe können Tausende, Zehntausende oder Hunderdtausende von Plattenspeichern angeschlossen sein. Zur Entlastung der CPU(s) wird die I/O Driver Abarbeitung auf dezentrale Spezialrechner ausgelagert, die als Control Units bezeichnet werden. Ohne diese Auslagerung wäre ein Mainframe Rechner niemals in der Lage, die erforderliche hohe Ein/Ausgabeleistung, besonders Plattenspeicherdurchsatz, zu erbringen.

Zum Anschluss der Control Units und Plattenspeicher wird ein Netzwerk von Verbindungen benötigt, die als Kanäle (Channels) bezeichnet werden. Kanäle werden den einzelnen I/O Anforderungen dynamisch zugeordnet, was ebenfalls sehr verarbeitungsaufwendig ist. Um hiervon die CPU(s) zu entlasten, erfolgt die Ansteuerung der Kanäle durch eine getrennte Komponente, das „Channel Subsystem“.

De Ein/Ausgabe Ansteuerung wird in Abb. 5.2.9 und in Band 2, Abschnitt 12.3 und 12.4 dargestellt.

1.3.13 Magnetbandspeicher

Fast alle Mainframe Installationen verwenden Magnetbänder um weniger häufig gebrauchte Daten zu speichern, und/oder um Daten zu archivieren.

Auf Magnetbänder (Tapes) wird mit Hilfe von Robotern zugegriffen. Diese verfügen über eigene Control Units, die von z/OS angesteuert werden. Ein Magnetband-Roboter ist in der Lage, eine Magnetbandkassette aus einem Regal zu entnehmen und in eine Magnetband Lese/Schreibstation einzulegen, um einen automatischen Zugriff zu ermöglichen. Der Umfang dieser Magnetbanddaten übertrifft den Umfang der Plattenspeicherdaten typischerweise um einen Faktor 10.

Eine typischerweise nochmals um einen Faktor 10 – 100 größere Datenmenge ist zu Archivierungszwecken ausgelagert.

Auch Magnetbandspeicher oder Magnetbandroboter werden über Control Units an den Mainframe Rechner angeschlossen.

1.3.14 Unterschiedliche Begriffe

Die Mainframe Welt benutzt häufig andere Begriffe als die PC oder Linux Welt. Hier ist ein kleiner Auszug aus meinem Wörterbuch:

z/OS, OS/390	Windows/Unix
Problem State	User Mode
Supervisor State	Kernel Mode
Region	Virtueller Adressenraum
Dataset	File
DASD	Plattenspeicher
Program Status Word	Status Register

DASD = Direct Access Storage Device

1.4 Weiterführende Information

Die meisten Kapitel dieses Buches enthalten einen Abschnitt mit der Überschrift „Weiterführende Information“. Hierbei handelt es sich um Information, die

- **zur Erläuterung hilfreich ist, oder**
- **ganz einfach interessant ist, oder**
- **zur Entspannung dienen soll.**

Teilweise handelt es sich dabei um YouTube Videos, mehrheitlich von der IBM Vertriebsorganisation. Videos aus anderen Quellen können Sie herunterladen und mit dem VLC Media Player (http://www.videolan.org/vlc/) oder einem Player Ihrer Wahl betrachten.

Das wichtigste Dokument

z/Architecture Principles of Operation:

http://publibz.boulder.ibm.com/epubs/pdf/dz9zr001.pdf

Dies ist die “Bibel” der Mainframe Architektur. Frühere Ausgaben gehen zurück bis in das Jahr 1964.

Berufsaussichten für Mainframe Spezialisten

Auf der Seite

http://www.cedix.de/beruf/index.html

finden Sie Informationen über die Berufsaussichten für junge Mainframe Nachwuchskräfte.

Ein Video zu gleichen Thema finden sie unter

http://www.linkedin.com/news?viewArticle=&articleID=799215892&gid=2196066&type=member&item=72622228&articleURL=http://www.abc.net.au/lateline/business/items/201109/s3326317.htm&urlhash=yx3e&goback=.gde_2196066_member_72622228%22

Sternstunde der Menschheit

Die Entstehung der Mainframes und der S/360 Architektur im Jahre 1964 ist eines der ganz ungewöhnlichen Ereignisse, so unwahrscheinlich, dass es eigentlich nie hätte passieren dürfen.

Die Vorgeschichte, die hierzu führte, ist in einigen Dokumenten beschrieben, die sich teilweise wie ein Kriminalroman lesen, und gut zur Entspannung eignen. Dies sind besonders

- **die Erinnerungen von IBM Vizepräsident Bob O. Evans, der seinerzeit für die Entwicklung von System S360 zuständig war http://www.cedix.de/Literature/History/boevans.pdf , und**

- **zwei Veröffentlichungen aus der Wirtschafts-Zeitung Fortune:**

  I.B.M.'s $ 5,000,000,000 Gamble
  http://www.cedix.de/Literature/History/FiveMillGamble1.pdf , und

  The rocky Road to the Marketplace
  http://www.cedix.de/Literature/History/RockyRoad1.pdf

Der wissenschaftliche Hintergrund ist beschrieben in

http://www.cedix.de/Literature/History/Amdahl.pdf

YouTube Videos

In YouTube existieren zahlreiche Videos zum Thema Mainframe, teilweise mit einem sehr vertriebsorientierten Hintergrund. Einige Beispiele sind:

"What is IBM zEnterprise System?"
http://www.youtube.com/watch?v=m9rC4yYbW2E

IBM launches "System zEC12" Mainframe
http://www.youtube.com/watch?v=DPcM5UePTY0

http://vimeo.com/4586385

Besonders faszinierend ist die Installation eines mittelständischen Unternehmens, der EFiS AG in Dreieich bei Frankfurt/M. Sie ist in einem Jahrhunderte-alten Gebäude untergebracht, und benutzt kühle Kellerluft an Stelle eines Klimagerätes. Ein Vorreiter für „grüne IT“.
http://www.youtube.com/watch?v=8fIF9WUx5gA

siehe auch http://efis.paymentgroup.de/fileadmin/files/documents/EFiS_PM_30.09.08.pdf

Siehe auch www.heise.de
http://www.heise.de/newsticker/meldung/IBMs-Mainframe-zEC12-mit-5-5-GHz-schnellen-Prozessoren-1675535.html

How to hack a z/OS System

z/OS verfügt über eine solide Sicherheits-Architektur. Sie kann jedoch umgangen werden, wenn die Sicherheitseinrichtungen fehlerhaft eingesetzt werden.

Die Mainframe Sicherheits-Architektur stellt einen soliden Schutzwall zur Verfügung. Mainframes haben die Reputation, dass sie immun gegen Viren, Trojaner und Hacker Angriffe sind, vorausgesetzt, der Systemadministrator hat die Sicherheitseinstellungen nicht deaktiviert. Es ist möglich, in diesem Schutzwall (eine begrenzte Anzahl von) Hintertürchen zu öffnen. Normalerweise würde der Sicherheitsadministrator sicherstellen, dass das nicht passiert.

Aber wenn aus Nachlässigkeit oder ähnlichen Gründen .....

Stuart C. Henderson zeigt in einem Referat, worauf der Systemadministrator achten muss:

http://www.stuhenderson.com/XBRKZTXT.PDF

Science Fiction

... and if you are interested to know, what Mainframes have to do with science fiction, read

Simba Wiltz: Mainframe- Beginnings,

http://www.amazon.com/Mainframe--Beginnings-Simba-Wiltz/dp/1401022871/ref=sr_1_2?s=books&ie=UTF8&qid=1328899842&sr=1-2

Well, maybe the relationship does not cover much more than the name.

2. Verarbeitungsgrundlagen

2.1 Multiprogrammierung

2.1.1 Rechnerstruktur

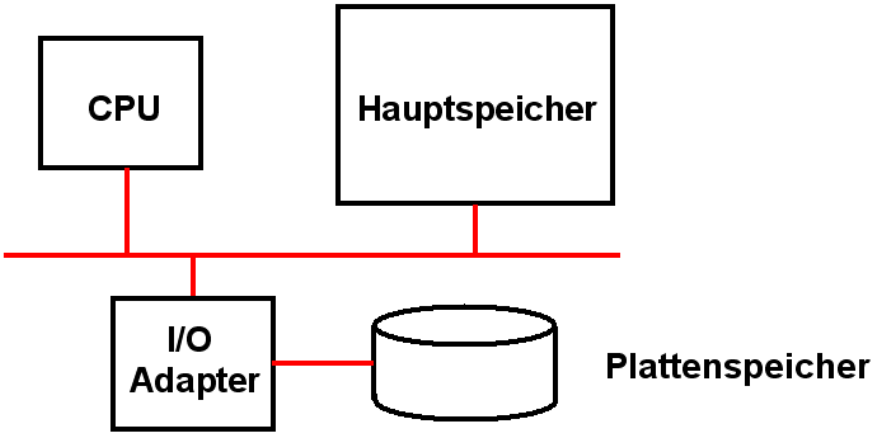


Abb. 2.1.1
Struktur eines Rechners

Ein moderner Rechner besteht aus einer CPU, welche ein Programm ausführt, einem Hauptspeicher (Main Storage, oft auch Arbeitspeicher genannt), welcher das auszuführende Programm und temporäre Daten enthält, und einem Anschluss (I/O Adapter) für einen (oder viele) Plattenspeicher, auf dem ausführbare Programme in Bibliotheken (Libraries) und Daten in Dateien und Datenbanken gespeichert sind. Der Hauptspeicher wird in Silizium Technologie implementiert, in der Form sog. DIMMs.

Ein auszuführendes Programm besteht aus einer Folge von Maschinenbefehlen (Machine Instructions). Das Format der Maschinenbefehle wird durch die Rechnerarchitektur bestimmt. Architekturen wie x86 (Pentium), PowerPC, Sparc, Itanium und System z (Mainframe) haben unterschiedliche Formate der Maschinenbefehle.

Ein Anwendungsprogramm, auch Benutzerprogramm genannt (application program, user program) wird heutzutage fast immer in einer höheren Programmiersprache wie Java, C++, Cobol, PL/1, ADA geschrieben, und durch einen Compiler (oder Interpreter) in ein Maschinensprache-Programm übersetzt, welches aus einer Folge von Maschinenbefehlen besteht. Programme in einer höheren Programmiersprache können nie direkt ausgeführt werden, sondern müssen vorher übersetzt (compiled oder interpreted) werden.

2.1.2 Quellprogramm und Maschinenprogramm

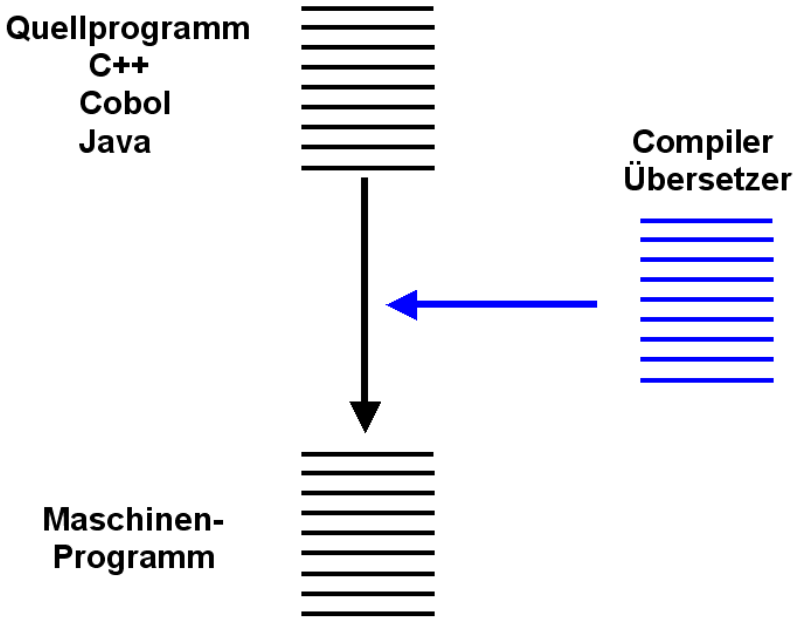


Abb. 2.1.2
Programm Übersetzung

Benutzerprogramme (andere Bezeichnung Anwendungsprogramm, engl.: user program, application program) werden in der Regel in einer höheren Programmiersprache wie C++, Cobol, PL/1, Java usw. geschrieben.

Ein Compiler ist ein Programm, dessen Ausführung ein Quellprogramm (Source Programm) als Input Daten benutzt, um daraus ein Maschinenprogramm, bestehend aus einzelnen Maschinenbefehlen, zu erstellen. Andere Bezeichnungen: Object Programm oder Binaries. Der Aufruf eines Compilers bewirkt eine Übersetzung (Compilation) eines Quellprogramms in ein Maschinenprogramm.

Meistens benutzt man ein weiteres Programm, einen Linker, um das übersetzte Maschinenprogrammen mit anderen, bereits früher übersetzten Programmen, zu einem ausführbaren Programm (executable program) zu verknüpfen. Ein Benutzerprogramm kann ans einzelnen Teilen (Modulen) bestehen, die einzeln geschrieben und übersetzt werden. Die übersetzten Maschinenprogramm Module werden in einem als Programmbibliothek (Library) bezeichneten Verzeichnis auf dem Plattenspeicher untergebracht.

Das Betriebssystem stellt zahlreiche weitere Maschinenprogramm Module zur Verfügung, die mit Hilfe des Linkers ebenfalls mit einem neu erstellten Maschinenprogramm verknüpft werden. Beispiele sind Ein/Ausgabe (I/O) Routinen oder die Socket Library für Intersytem Communication. Unter dem z/OS Betriebssystem stehen viele solcher Maschinenprogramme in einer mit dem Parameter SYS1 gekennzeichneten Library.

Ein Loader ist ein Maschinenprogramm, welches ein ausführbares Programm aus einer Library ausliest und in den Hauptspeicher lädt.

2.1.3 Hauptspeicheradressen

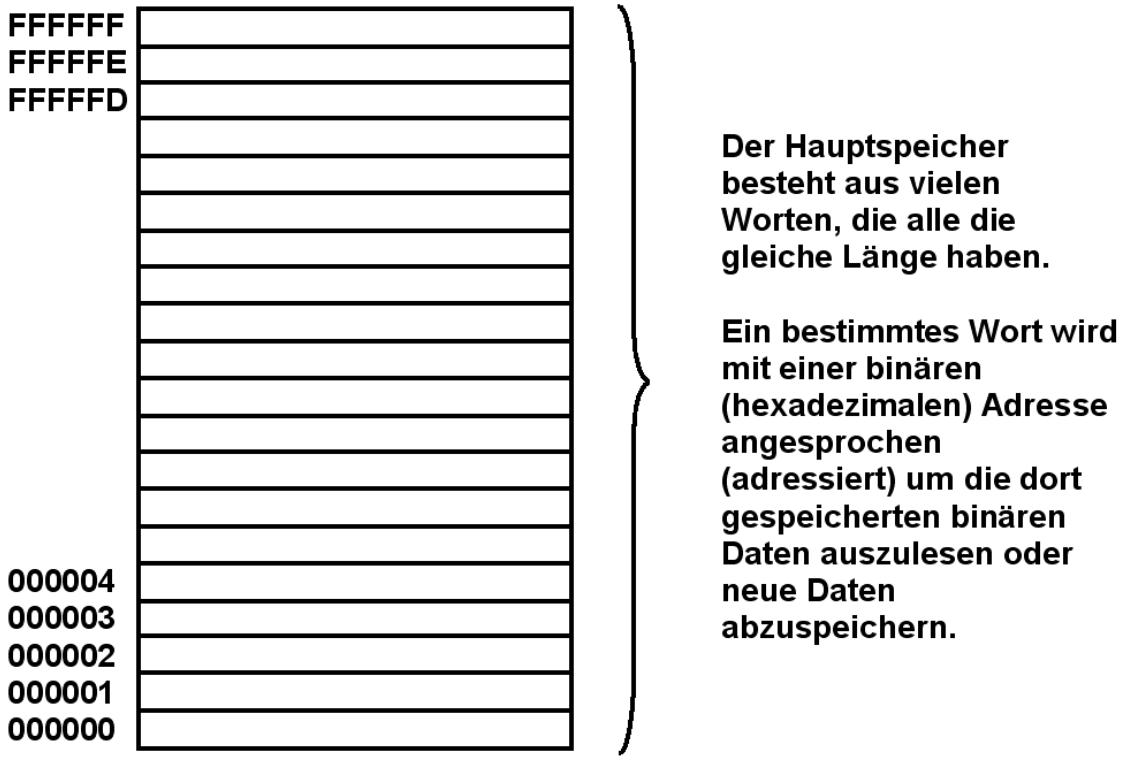


Abb. 2.1.3
Adressierung des Hauptspeichers

Der Hauptspeicher (main Storage) besteht aus lauter gleich langen Wörtern, die mit binären bzw. hexadezimalen Ziffern angesprochen (adressiert), gelesen und gespeichert werden.

Die Wortlänge eines Hauptspeichers ist entweder 8, 12, 16, 14, 32, 36 oder 48 Bit. Moderne Rechner benutzen fast ausschließlich eine Wortlänge von 8 Bit (ein Byte).

Im Hauptspeicher werden sowohl Maschinenprogramme als auch Daten gespeichert.

2.1.4 32 und 64 Bit Adressen

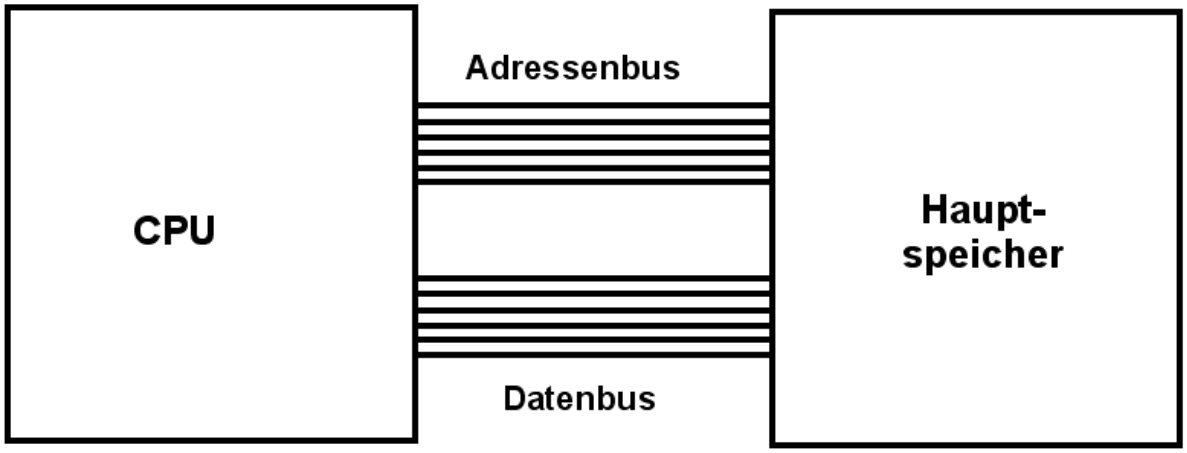


Abb. 2.1.4
Adressierung des Hauptspeichers

Wir unterscheiden zwischen 32 Bit und 64 Bit Architekturen. Bei einer 32 Bit Architektur ist die CPU mit dem Hauptspeicher mit einem Adressenbus und einem Datenbus verbunden. Der Adressenbus besteht aus 32 Leitungen, um eine von 2^{32} Adressen an den Hauptspeicher zu übergeben. Bei einer 64 Bit Architektur sind es 64 Leitungen, um eine von 2^{64} Adressen an den Hauptspeicher zu übergeben. Der Datenbus besteht aus 8 Leitungen, auf denen jeweils 1 Byte (8 Bit) zwischen Hauptspeicher und CPU übertragen wird.

Die Maschinenbefehle gehen von dieser Struktur aus, die wir als das logische Erscheinungsbild einer CPU bezeichnen. Die physische Implementierung eines Rechners weicht in der Regel von der logischen Sicht ab. So besteht z.B. ein Datenbus häufig aus 256 Leitungen, und es werden gleichzeitig 8 Bytes (256) Bits zwischen CPU und Hauptspeicher übertragen. Ein moderner Mainframe Rechner (z196) hat einen physischen Hauptspeicher mit einer maximalen Größe von 4 TByte (2^{42} Byte), für dessen Adressierung 42 Leitungen des Adressenbusses ausreichen.

Anmerkung: Bei der Mainframe Architektur verfügt die ehemalige 32 Bit Architektur nur über 31 Bit Adressen, und adressiert damit einen Hauptspeicher mit einer maximalen Größe von 2^{31} Adressen und 2^{31} Byte Speicherkapazität.

Im Gegensatz zum Hauptspeicher ist der Zugriff auf einen Festplattenspeicher sehr viel komplizierter. Für den Zugriff setzt die CPU ein spezielles Maschinenprogramm, einen I/O (Input/Output) Driver, ein.

Eine moderne CPU führt größenordnungsmäßig eine Milliarde (10^9) Maschinenbefehle pro Sekunde aus. Die Zugriffszeit zum Hauptspeicher (Cache, siehe Abb. 2.4.2) beträgt größenordnungsmäßig wenige 100 Picosekunden (10^{-9} bis 10^{-10} s, logische Sicht). Die Zugriffszeit zum Plattenspeicher beträgt größenordnungsmäßig 10 Millisekunden (10^{-2} s) . Dieser Unterschied in der Zugriffszeit hat einen sehr großen Einfluss auf die Struktur moderner Betriebssysteme.

2.1.5 Supervisor and User Space

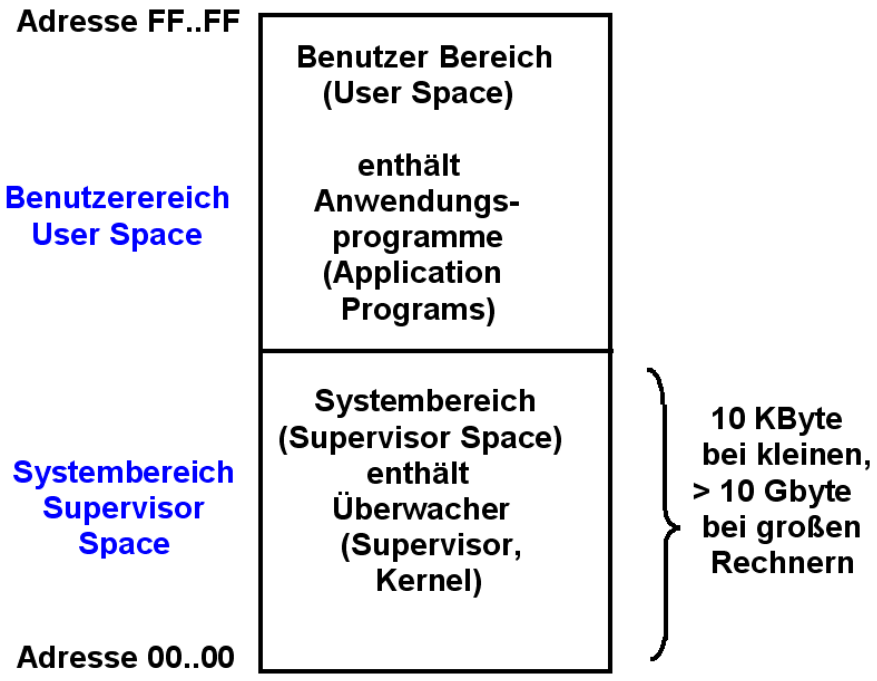


Abb. 2.1.5
Aufteilung des Adressenraums

Ein Betriebssystem enthält Programmroutinen, die von Benutzerprogrammen immer wieder gebraucht werden. Ein Beispiel ist der I/O Driver (Input/Output Driver), der das Lesen oder Schreiben von Daten vom/zum Plattenspeicher bewirkt.

Der Code des Betriebssystems befindet sich auf dem Plattenspeicher. Beim Hochfahren eines Rechners wird ein Teil des Betriebssystems in den Hauptspeicher geladen. Diesen Teil bezeichnet man als Betriebssystem-Kern, Kernel oder Überwacher (Supervisor). z/OS benutzt auch die Begriffe Nucleus oder „Basic Control Program“ (BCP).

Wir bezeichnen die Menge aller Hauptspeicheradressen als Adressenraum (Address Space). In dem hier dargestellten einfachsten Fall wird der Adressenraum des Hauptspeichers in 2 Teile aufgeteilt: einen Teil für den Überwacher (Supervisor Address Space) und einen zweiten Teil für das auf dem Rechner laufende Benutzerprogramm (Anwendungsprogramm) und seine Daten (User Address Space). Das ursprüngliche DOS Betriebssystem für den PC hatte diese Struktur.

Damit ein fehlerhaftes oder böswilliges Benutzerprogramm nicht unbefugt Programme oder Daten im Überwacher Adressenraum modifizieren kann, läuft der Rechner in jedem Augenblick entweder im Überwacherstatus (Supervisor State) oder im Benutzerstatus (User State, auch als Problem State bezeichnet).

Ein Programm, das im Überwacherstatus läuft, kann auf den gesamten Adressenraum des Hauptspeichers zugreifen. Ein Programm, das im User Modus läuft, hat Zugriffsrechte nur für den User Adressenraum.

2.1.6 Prozess

Was ist ein Prozess ?

Ein Maschinen-Programm lässt sich mit den Noten eines Musikstückes vergleichen. Ein Konzert ist die Aufführung eines Musikstückes entsprechend den verwendeten Musiknoten. Ein Prozess ist die Ausführung eines Maschinenprogramms auf einem Computer.

Der Begriff „Prozess“ ist ein zentrales Konzept in der modernen Datenverarbeitung. Ein Prozess ist die Ausführung von einem Programm (oder mehreren Programmen) um ein für einen Benutzer relevantes Problem zu bearbeiten. Beispiele für Prozesse sind die monatliche Lohn- und Gehaltsabrechnung in einem Unternehmen, die Verbuchung bei der Überweisung eines Geldbetrages, eine URL Abfrage im Internet oder die Stücklistenerstellung beim Bau eines Automobils.

Ein Benutzer Prozess läuft innerhalb des Benutzer Adressenraums. Systemprozesse bearbeiten irgendwelche Teilaufgaben des Betriebssystems.

Die Ausführung eines Prozesses bewirkt in der Regel, dass Daten abgeändert werden, die auf einem Festplattenspeicher permanent gespeichert sind. Ein Beispiel sind Kontodaten bei der Überweisung eines Geldbetrages. Hierzu lädt der Prozess die Daten vom Plattenspeicher in den Hauptspeicher, modifiziert sie und schreibt das Ergebnis auf den Plattenspeicher zurück.

Eine Transaktion ist eine spezielle Art eines Prozesses, bei dem garantiert wird, das alle Änderungen von Daten auf dem Plattenspeicher entweder vollständig oder gar nicht, nicht aber teilweise, erfolgen.

Ein Prozess wird von z/OS als „Task“ bezeichnet. Eine Task wird durch einen Bereich im Hauptspeicher definiert, der als „Task Control Block (TCB) bezeichnet wird. Linux verwendet hier die Bezeichnung „Process Control Bloch (PCB).

2.1.7 Struktur eines Benutzerprogramms

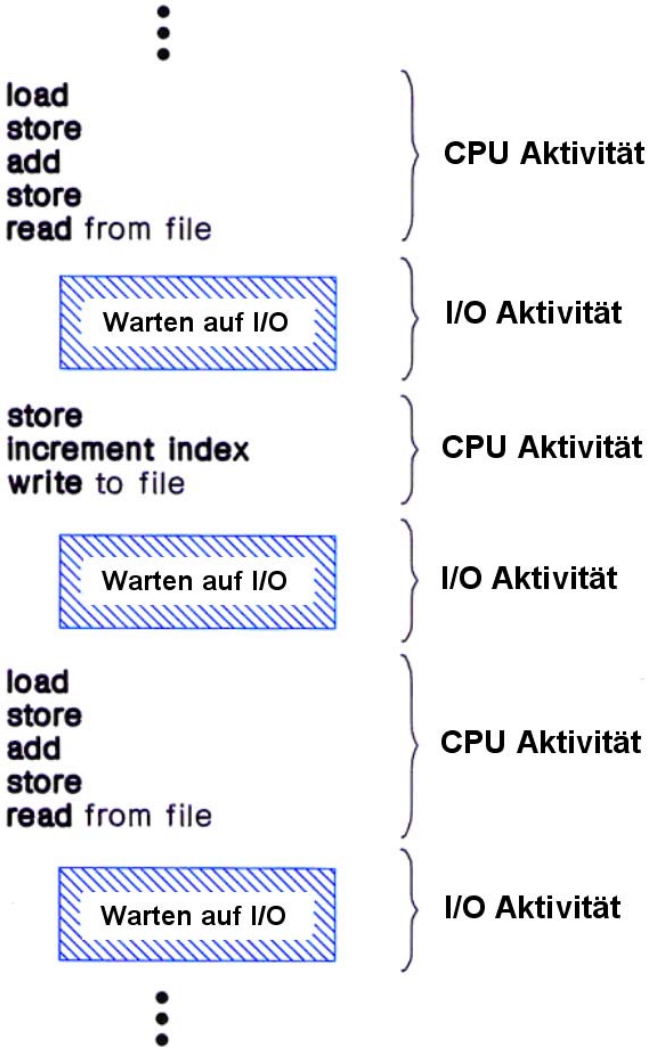


Abb. 2.1.6
Struktur eines Benutzerprogramms

Ein Benutzerprogramm besteht aus einer Folge von

- **Gruppen von Maschinenbefehlen**
- **I/O Operationen (Zugriff auf Festplattenspeicher-Daten).**

Die Programmausführung ist eine abwechselnde Folge von CPU und I/O Aktivitäten.

Beim Start einer I/O Operation muss das Anwendungsprogramm in der Regel warten, bis die I/O Operation abgeschlossen ist (z.B. die vom Plattenspeicher gelesenen Daten im Hauptspeicher eingetroffen sind).

Der Zugriff auf einem Plattenspeicher benötigt etwa 10 Millisekunden. Während dieser Zeit könnte die CPU etwa 10 Millionen Maschinenbefehle ausführen. Es wäre unökonomisch, wenn die CPU während der I/O Aktivität warten müsste.

2.1.8 Multiprogrammierung

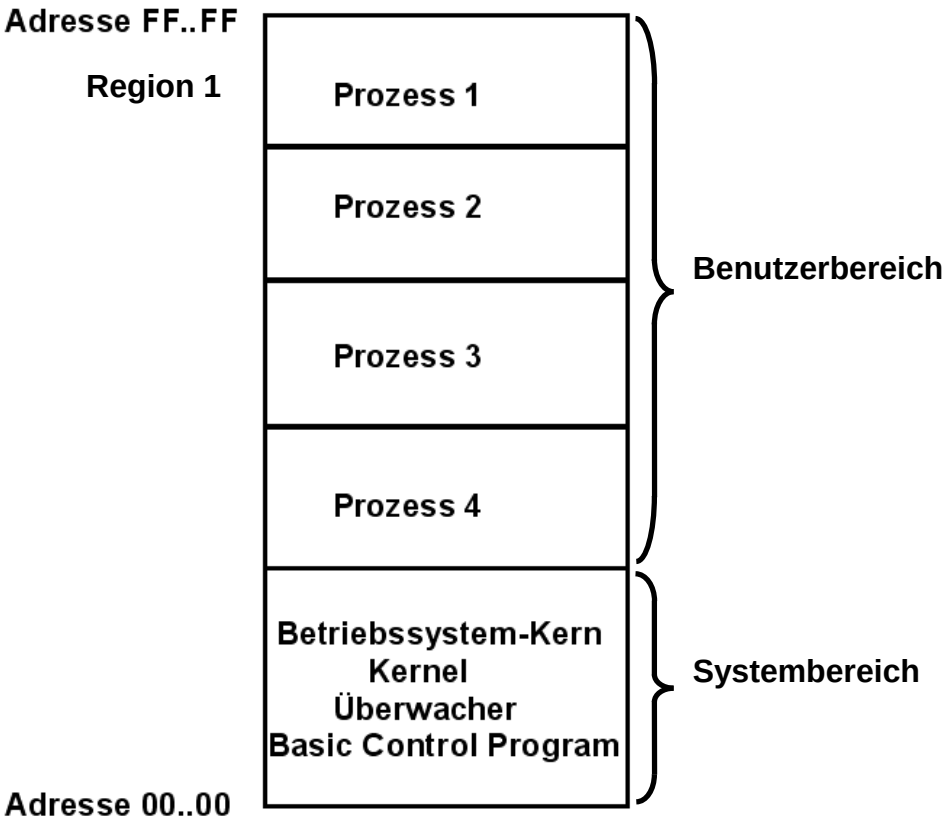


Abb. 2.1.7
Multiprogrammierung

Bei der Multiprogrammierung teilt man des Benutzer Adressenraum in mehrere Teilbereiche (Regions) auf und ordnet einem Prozess jeweils eine eigene Region zu.

Jetzt befinden sich die Programme (Code) mehrerer Prozesse gleichzeitig im Hauptspeicher. Wir nehmen in diesem Beispiel an, dass der Rechner eine einzige CPU hat. Wenn nun ein Prozess eine I/O Operation ausführt, wird er in einen Wartezustand versetzt. Ein anderer Prozess, dessen Programme und Daten sich ausführungsbereit in seiner Region des Benutzer Adressenraums befinden, wird nun von der CPU ausgeführt, bis er ebenfalls eine I/O Operation ausführen will. Jetzt beginnt die CPU einen dritten Prozess auszuführen usw.

In einem Mainframe Rechner werden auf diese Art typischerweise Tausende oder Zehntausende von Prozessen parallel ausgeführt. Diese Art der Datenverarbeitung, bei der mehrere Prozesse ineinander geschachtelt parallel verarbeitet werden, wird als „Multiprogrammierung“ (Multiprogramming) bezeichnet.

Prozesse werden normalerweise gestartet, verrichten ihre Arbeit und werden wieder beendet. Ein Daemon ist ein lang laufender Prozess, der häufig zusammen mit dem Hochfahren des Betriebssystems gestartet, und nie beendet wird.

2.1.9 Zustand von Prozessen

Wenn ein Prozess eine Plattenspeicher I/O Operation startet wird er in den Zustand „wartend“ (waiting) versetzt. Wenn die I/O Operation abgeschlossen ist, könnte er wieder von der CPU ausgeführt werden. Da die CPU vermutlich gerade mit einem anderen Prozess beschäftigt ist, wird er statt dessen in den Zustand „ausführbar“ (ready) versetzt. Irgendwann wird die CPU den Prozess weiter verarbeiten. Ein Prozess, der durch die CPU verarbeitet wird, wird als „laufend“ (running) bezeichnet.

Jeder Prozess rotiert wiederholt durch die Zustände laufend, wartend und ausführbar. Moderne Rechner haben meistens mehr als eine CPU, z.B. 101 bei einem zEC12 Mainframe. In der Regel verarbeitet jede CPU einen anderen Prozess. In diesem Fall können sich mehrere Prozesse im Zustand „laufend“ befinden.

- **Ein Prozess befindet sich im Zustand laufend, wenn er von einer der verfügbaren CPUs ausgeführt wird.**
- **Ein Prozess befindet sich im Zustand wartend, wenn er auf den Abschluss einer I/O Operation wartet.**
- **Ein Prozess befindet sich im Zustand ausführbar, wenn er darauf wartet, dass die Scheduler Komponente des Überwachers ihn auf einer frei geworden CPU in den Zustand laufend versetzt.**

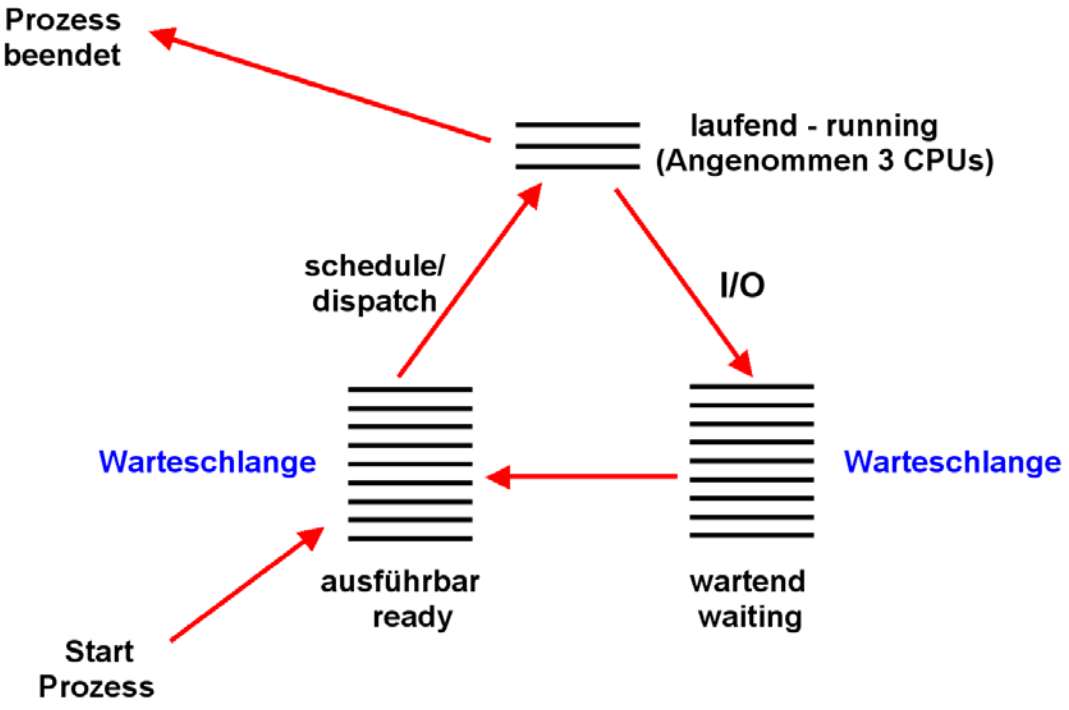


Abb. 2.1.8
Zustand von Prozessen

Jede Linie entspricht einem Prozess. Jeder Prozess wird durch einen Bereich im Hauptspeicher (Task Control Block, TCB) gekennzeichnet und repräsentiert. Die Scheduler Komponente des Überwachers unterhält drei Warteschlangen (Queues) für wartende, ausführbare und laufende Prozesse. In diese Warteschlangen werden die TCBs der Prozesse beim Wechsel des Zustandes eingeordnet.

Die Scheduler Komponente des Überwachers selektiert aus der Warteschlange ausführbarer Prozesse jeweils einen Kandidaten wenn immer eine CPU frei wird.

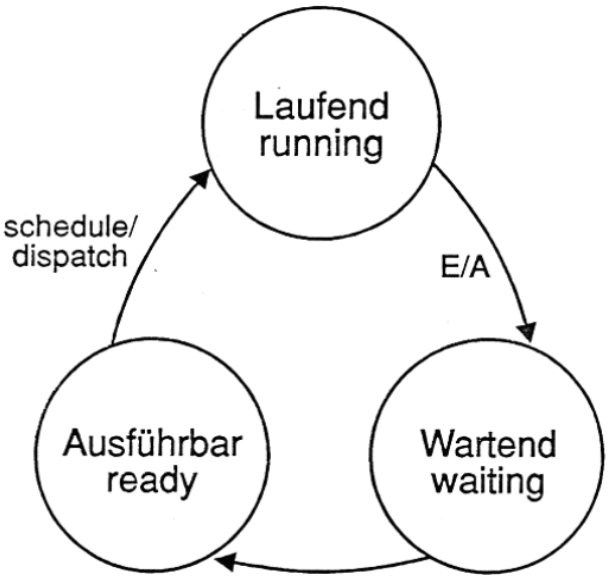


Abb. 2.1.9
Prozess Zustandsdiagramm

Dies ist eine andere Darstellung der obigen Abb. 2.1.8 Auf einem Mehrrechnersystem (Rechner mit mehr als einer CPU) können sich mehrere Prozesse im Zustand laufend befinden. Es ist auch möglich, dass ein Prozess mehrere CPUs beschäftigt.

Ausführbare Prozesse verfügen über Ressourcen, besonders über Platz im Hauptspeicher (Programm Code, Daten)

2.1.10 Zeitscheiben Steuerung (Time Slicing)

Bei der Ausführung eines Anwendungsprogramms treten I/O Anforderungen relativ häufig auf. In diesem Fall wird der betroffene Prozess in den Zustand wartend versetzt, und die Scheduler Komponente des Überwachers (Supervisor) selektiert aus der Menge (Queue) der ausführbaren Prozesse einen Prozess, der in den Zustand laufend versetzt wird.

In der großen Mehrzahl der Fälle wird dieser Prozess nach einem Zeitraum, der selten 1 ms überschreitet, in den Zustand wartend versetzt, weil er eine I/O Operation ausführt. Es gibt jedoch Situationen, in denen dies nicht der Fall ist. Um zu verhindern, dass ein Prozess eine CPU usurpiert, enthält der Scheduler eine Zeitscheiben (Time Slice) Steuerung, die einen laufenden Prozess nach einer gewissen Zeit (typischerweise wenige ms) unterbricht, in den Zustand ausführbar versetzt, und dafür einen anderen ausführbaren Prozess in den Zustand laufend versetzt. Dieser Vorgang wird als „Preemption“ bezeichnet.

2.1.11 Multiprogrammierung

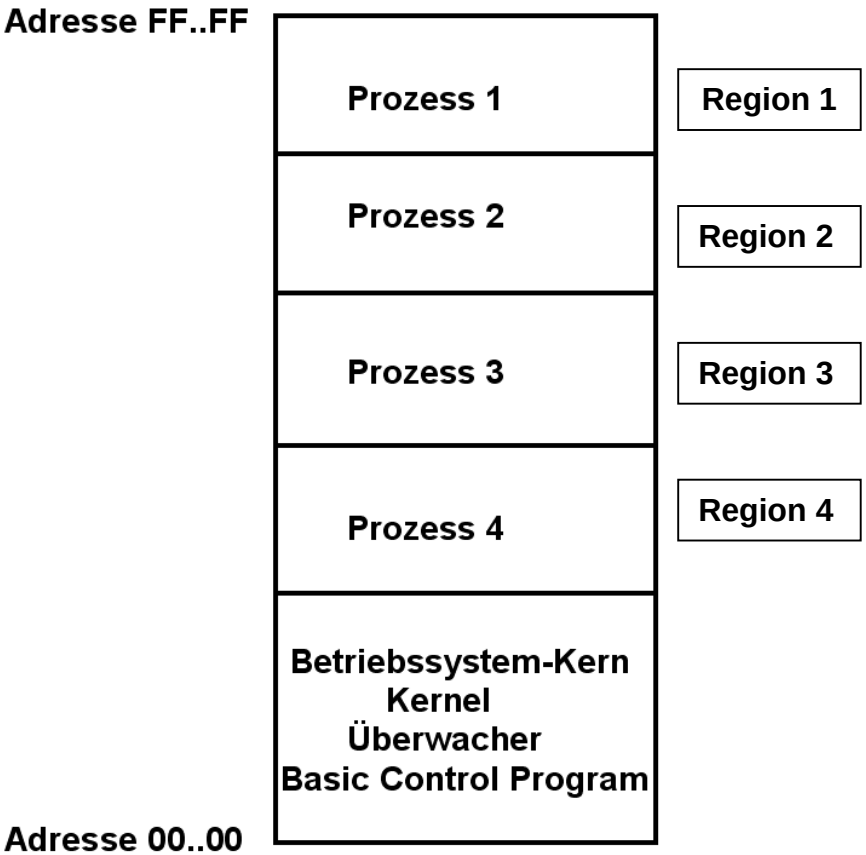


Abb. 2.1.10
Multiprogrammierung

Die Aufteilung des Benutzer Adressenraum in mehrere Regions für mehrere Prozesse erzeugt mehrere Probleme:

- **Es muss verhindert werden, dass fehlerhafte Programme eines Prozesses auf die Region eines anderen Prozesses zugreift.**

- **Bei Beendigung eines Prozesses wird die benutzte Region des Benutzer Adressenraums freigegeben. In ihr kann jetzt ein neuer Prozess gestartet werden.**

  Die einzelnen Prozesse benötigen aber unterschiedlich viel Speicherplatz. Die Regions haben deshalb eine unterschiedliche Größe. Die Größe der freigewordenen Region entspricht nicht notwendigerweise den Bedürfnissen des neuen Prozesses. Eine komplexe Speicherverwaltungskomponente (als Teil des Überwachers) ist erforderlich.

Zur Lösung dieses Problems führte IBM 1972 die virtuelle Speichertechnik (Virtual Storage) für eine neue Serie von Mainframe Rechnern (System /370) ein.

2.2 Virtual Storage

2.2.1 Virtueller Speicher

Die Befehlsadressen und effektiven Adressen der Operanden arbeiten bei einem modernen Rechner mit einer Illusion eines Speichers (virtueller Speicher), der eine andere und einfachere Struktur hat als der reale Hauptspeicher, in dem sich die Befehle und Operanden tatsächlich befinden.

Hierzu wird der virtuelle Speicher in Blöcke aufgeteilt, die alle die gleiche Größe (z.B. 4096 Bytes) haben. Diese Blöcke nennt man Seiten (pages).

Der Hauptspeicher wird in Blöcke aufgeteilt, die genauso groß wie die Seiten sind und als Platzhalter für die Aufnahme von Seiten dienen. Diese Blöcke nennt man Rahmen (frames oder pageframes).

Virtuelle Adressen adressieren Maschinenbefehle und Operanden im virtuellen Speicher.

Reale Adressen adressieren Maschinenbefehle und Operanden im (realen) Hauptspeicher.

Der virtuelle Speicher ist aus der Sicht des Programmierers ein kontinuierlicher, einfach zusammen-hängender, linearer Adressenraum.

Die Abbildungsvorschrift für die virtuelle (logische) in die reale (Physikalische) Adressumsetzung ist in einer Seitentabelle enthalten.

Wir benutzen die folgenden Begriffe:

Ein Adressenraum (Address Space) ist die lineare Folge von Adressen der Bytes (oder anderer adressierbarer Einheiten) eines Speichers.

Ein Virtueller Adressenraum ist die lineare Folge von Adressen der Bytes eines virtuellen Speichers.

Ein Realer Adressenraum ist die lineare Folge von Adressen der Bytes eines realen (tatsächlich existierenden) Hauptspeichers.

Mittels der Adressumsetzung (Dynamic Address Translation, DAT) wird der virtuelle Adressenraum der Befehle und Operanden eines Benutzerprozesses vom realen Adressenraum des Hauptspeichers getrennt. Die Adressumsetzung bewirkt die Abbildung von virtuellen Adressen auf reale Adressen.

2.2.2 Seiten und Rahmen

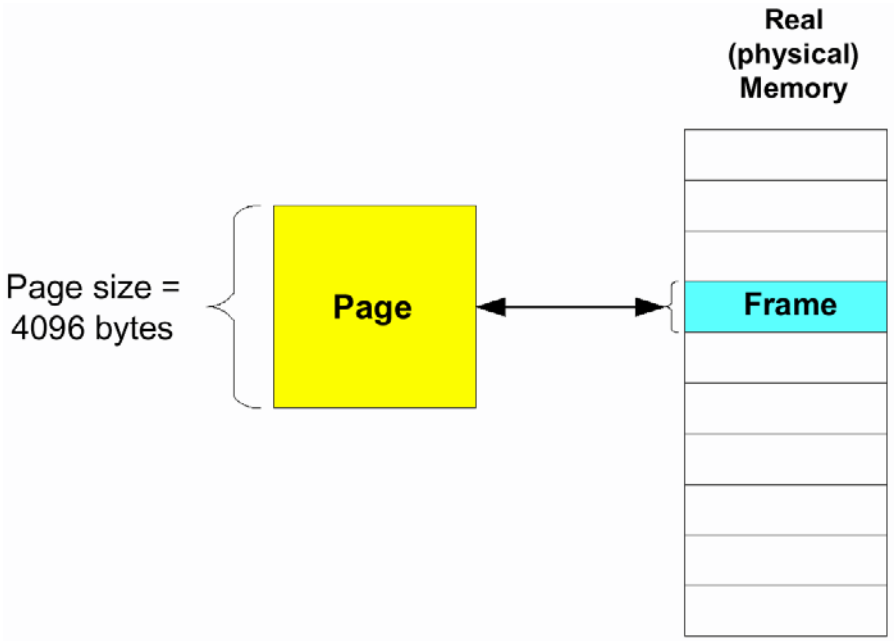


Abb. 2.2.1
Seiten und Rahmen

Für die Adressumsetzung wird der Virtuelle und der reale Adressenraum jeweils in 4096 Byte große Blöcke aufgeteilt. Die Blöcke des virtuellen Speichers werden als Seiten (Pages) und die Blöcke des realen Speichers als Rahmen (Frames) bezeichnet. Die Anordnung der Bytes innerhalb einer Seite oder eines Rahmens ist identisch und wird bei der Adressumsetzung nicht verändert. Es erfolgt aber eine willkürliche Zuordnung von Seiten- zu Rahmenadressen.

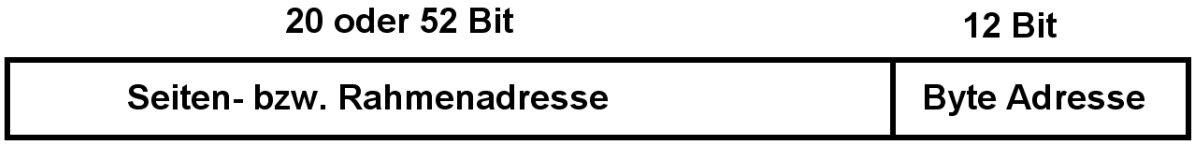


Abb. 2.2.2
Adressenformat

Eine virtuelle bzw. reale Adresse besteht deshalb grundsätzlich aus zwei Feldern:

- **Seiten- bzw. Rahmenadresse und**
- **Byteadresse**

2.2.3 Seitentabelle

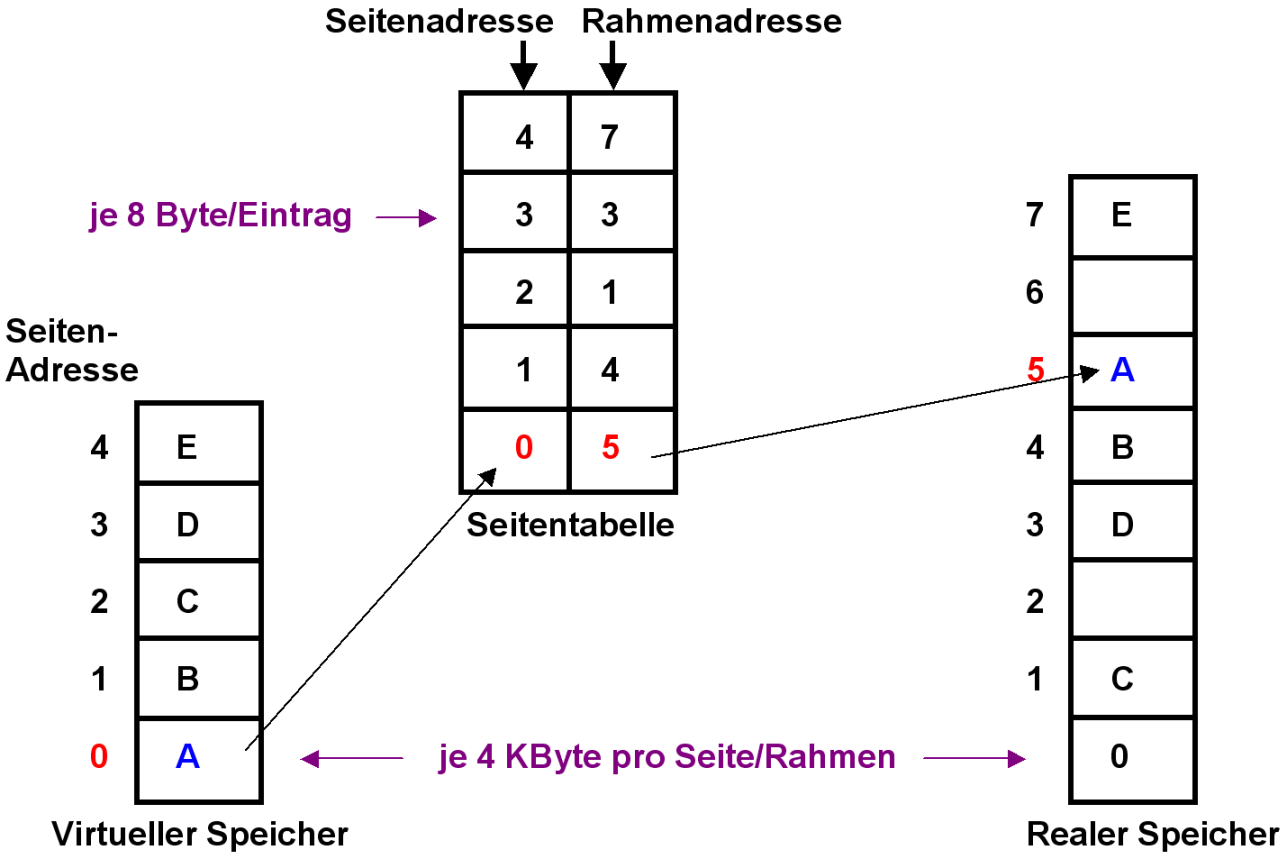


Abb. 2.2.3
Aufgabe der Seitentabelle

In dem hier gezeigten Beispiel erstreckt sich der virtuelle Speicher über 5 Seiten, der reale Hauptspeicher über 8 Rahmen. Virtuelle und reale Speicher können durchaus unterschiedliche Größen haben.

Die 5 Seiten des virtuellen Speichers speichern die Inhalte A, B, C, D und E. A hat die Seitenadresse 0. Die Adressumsetzung erfolgt mit Hilfe einer Seitentabelle (page table). Jede Seite des virtuellen Speichers hat einen Eintrag in der Seitentabelle. Der Eintrag für Seitenadresse 0 besagt, dass der Seiteninhalt A in dem Rahmen mit der Rahmen-Adresse 5 abgebildet wird.

2.2.4 Größe des virtuellen Speichers

Theoretisch ist es denkbar, dass bei einem Rechner mit 64 Bit Adressen jeder virtuelle Adressenraum eine Größe von 2^{64} Bit hat. In der Praxis ist das nicht machbar, unter anderem, weil die Größe der Seitentafel im realen Speicher proportional zu der Größe aller virtuellen Speicher ist.

Seitentabellen erfordern wenige Promille der virtuellen Speichergröße an realem Hauptspeicherplatz. Zwei Promille von 2^{64} Bytes = 16 Exabyte sind 32 Terabyte.

Ein Mainframe Rechner kann 10 000 Prozesse mit getrennten Seitentabellen unterhalten. Die maximale Hauptspeichergröße eines z196 Mainframe beträgt aber „nur“ 3 TByte.

Virtuelle Speicher sollen deshalb nur so groß eingerichtet werden, wie es der Prozess erfordert.

Die Seitentabelle befindet sich (entweder teilweise oder ganz) im Überwacherteil des realen Hauptspeichers.

Genau genommen verwendet ein Mainframe (und auch ein x86) Rechner nicht eine einzige Seitentabelle, sondern eine 2 – 5 stufige Hierarchie von Seitentabellen. Im Gegensatz dazu verwendet die PowerPC Architektur nur eine einzige Seitentabelle

Abb. 2.2.4 zeigt die 2-stufige Adressumsetzung, wie sie in den 32 (31) Bit Versionen der x86 und der System z Architektur implementiert ist.

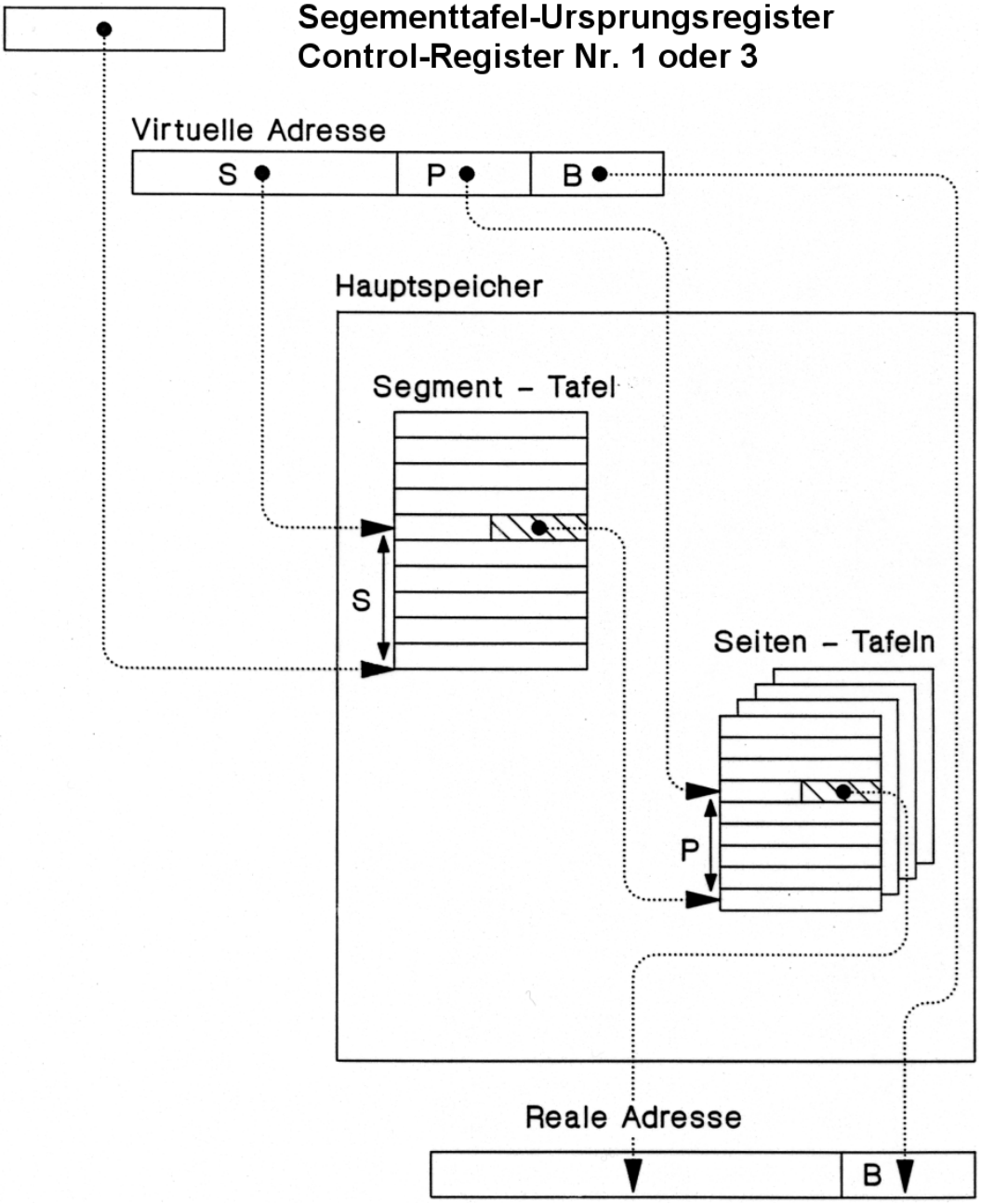


Abb. 2.2.4
Adressumsetzung Pentium, S/390

Die Adressumsetzung erfolgt durch zwei Arten von Tabellen (Segment- und Seitentabelle), die im Kernel Bereich des Hauptspeichers untergebracht sind (Siehe auch Band 1, Abschnitt 2.2.4). Die Anfangsadresse der Segmenttabelle steht in einem Control Register der Zentraleinheit, z.B. CR Nr. 1 bei der S/390 Architektur.

Die Adressumsetzung erfolgt bei jedem Hauptspeicherzugriff durch Hardware mit Unterstützung durch die Segmenttabelle und eine Seitentabelle im Kernel-Bereich. Sie kann durch den Programmierer nicht beeinflusst werden.

(Zur Leistungsverbesserung werden die derzeitig benutzten Adressen in einem Adressumsetzpuffer untergebracht.)

2.2.5 Externer Seitenspeicher

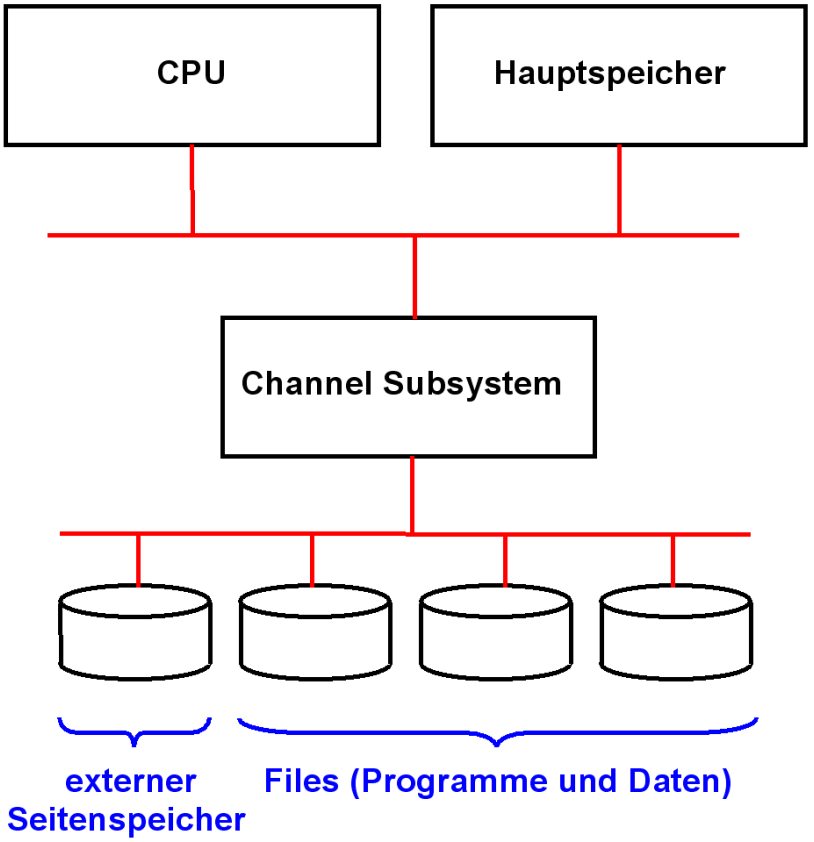


Abb. 2.2.5
Externer Seitenspeicher

Der virtuelle Speicher kann auch größer als der reale Hauptspeicher sein.

In diesem Fall wird ein Teil des realen Hauptspeichers (Real Storage) auf einen als externer Seitenspeicher (Auxiliary Storage oder Page Datasets) bezeichneten Plattenspeicher ausgelagert.

Bei einem Windows System ist dies z.B. der Bereich pagefile.sys auf der Partition C: . Bei einem Mainframe ist das in der Regel ein (oder mehrere) nur hierfür benutzter Festplattenspeicher. Der externe Seitenspeicher ist hierfür in 4096 Byte große Rahmen aufgeteilt.

Sehr grob betrachtet hat das Channel Subsystem eines Mainframes die Funktion eines I/O Adapters.

2.2.6 Speichern der Rahmen

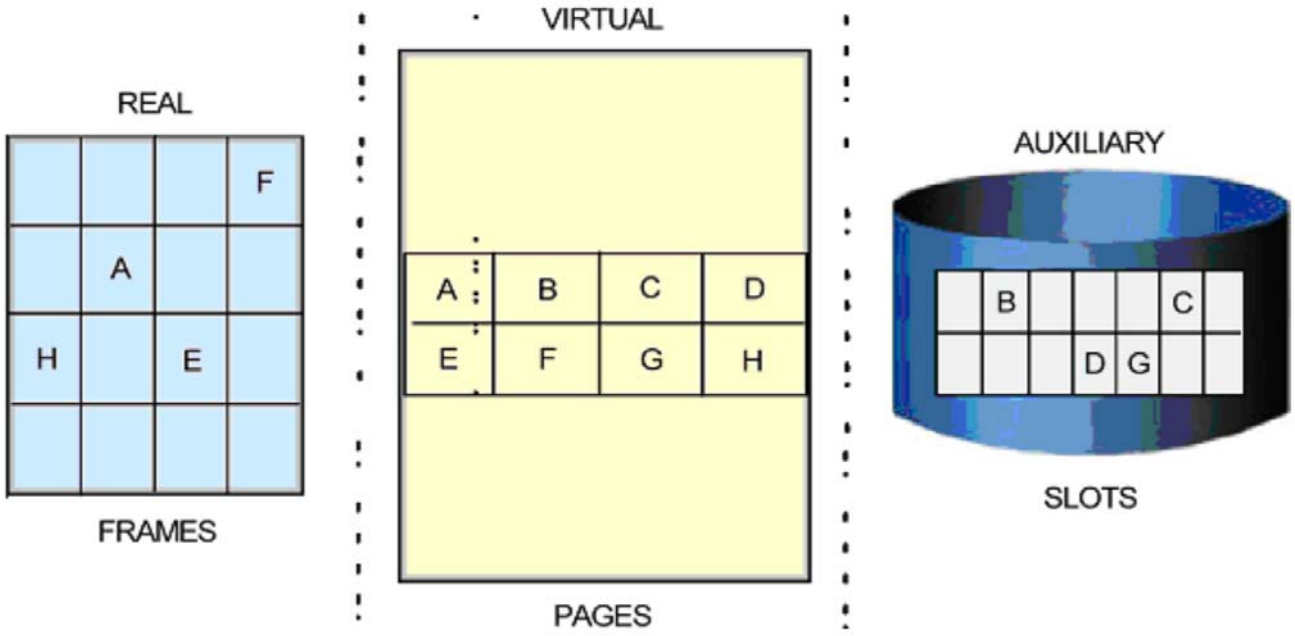


Abb. 2.2.6
Speichern der Rahmen

In dem hier gezeigten Beispiel enthält der virtuelle Speicher die Seiten mit den Inhalten A, B, C, D, E, F, G und H.

A, E, F und H sind in Rahmen des realen Hauptspeichers (Real Storage) abgespeichert. B, C, D und G befinden sich auf dem externen Seitenspeicher (Auxiliary Storage). z/OS bezeichnet die Rahmen des externen Seitenspeichers gelegentlich auch als „Slots“.

Wenn ein Benutzerprogramm auf B, C, D oder G zugreifen will, muss eine Komponente des Überwachers, der „Seitenüberwacher“ (Paging Supervisor), den Rahmen zuerst vom externen Seitenspeicher in den Hauptspeicher kopieren. Vermutlich muss er, um Platz zu schaffen, vorher den Inhalt einen anderen Rahmen des Hauptspeichers auf den externen Seitenspeicher auslagern.

2.2.7 Demand Paging

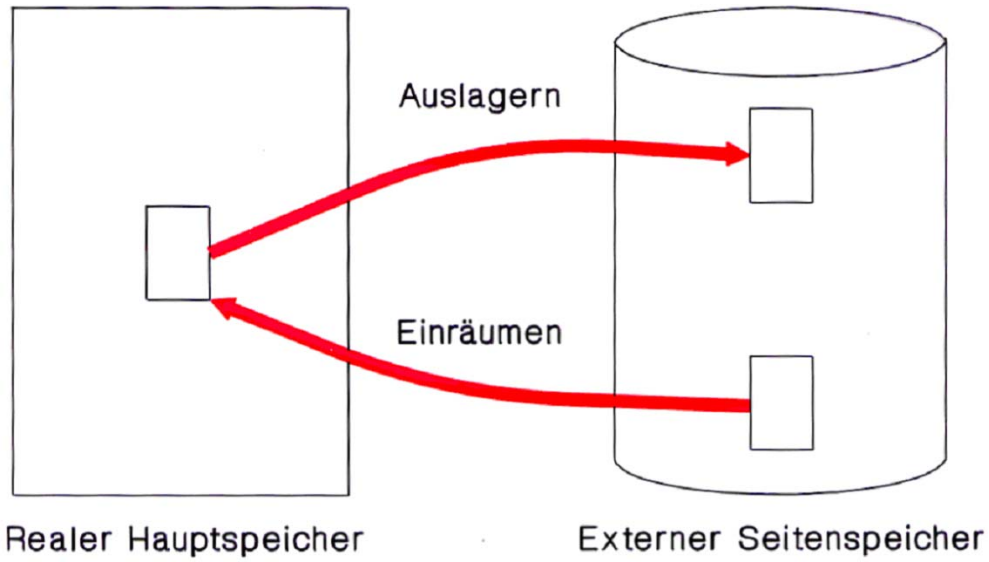


Abb. 2.2.7
Demand Paging

Mehrere Prozesse haben eigene (i n der Regel) unabhängige virtuelle Speicher. Jeder virtuelle Speicher kann größer als der reale Hauptspeicher sein.

Der größere Teil der Seiten eines virtuellen Speichers ist in jedem Augenblick auf einem " externen Seitenspeicher " (Auxiliary Storage), typischerweise ein Festplattenspeicher, ausgelagert. Der reale Speicher besteht somit aus 2 Teilen: dem realen Hauptspeicher und dem externen Seitenspeicher.

Beim Zugriff zu einer ausgelagerten Seite (nicht in einem Rahmen des Hauptspeichers abgebildet) erfolgt eine " Fehlseitenunterbrechung " (Page Fault).

Diese bewirkt den Aufruf einer Komponente des Überwachers (Seitenüberwacher, Paging Supervisor), der die benötigte Seite aus dem externen Seitenspeicher holt und in den Hauptspeicher einliest. Evtl. muss dafür Platz geschaffen werden, indem eine andere Seite dafür auf den externen Seitenspeicher gelegt wird.

Dieser Vorgang wird als „ Demand Paging " bezeichnet.

2.2.8 Abbildung des Virtuellen Speichers auf den realen Speicher

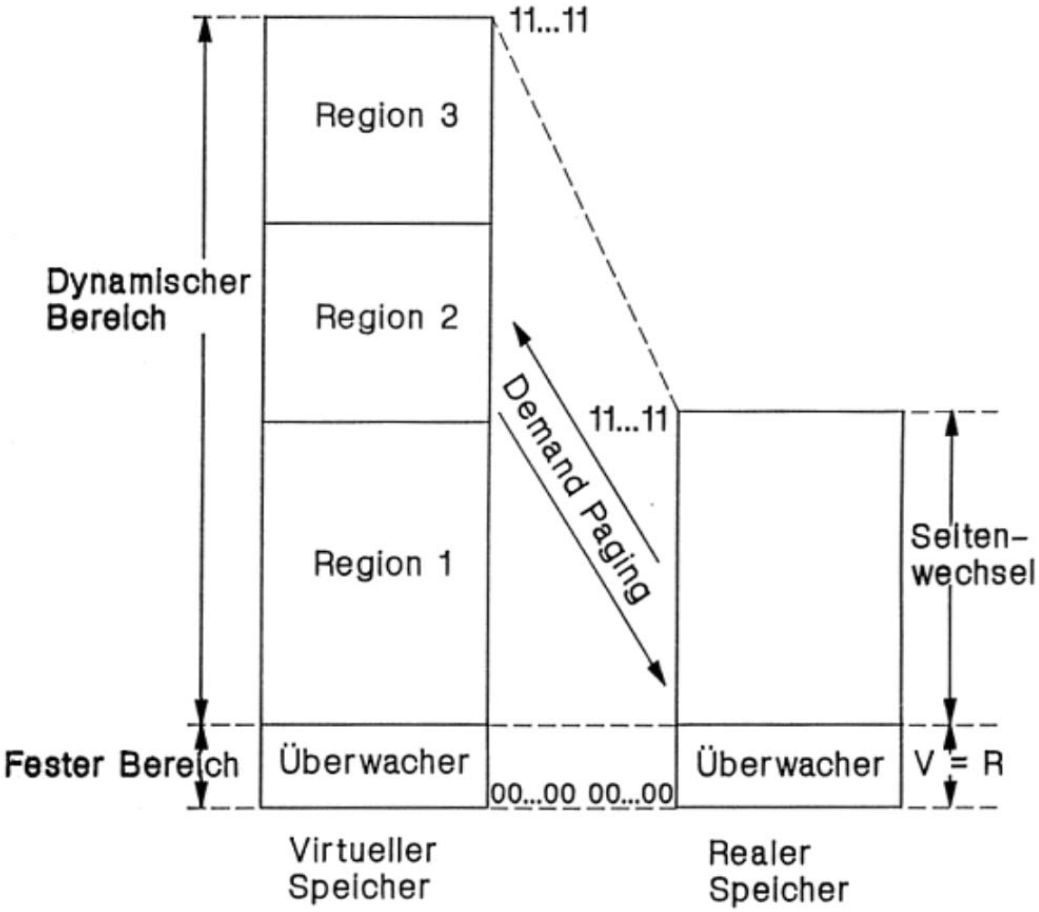


Abb. 2.2.8
Abbildung des virtuellen Speichers

Der Benutzer Adressenraum ist in der Regel größer als der reale Hauptspeicher. Ein Teil des Benutzerraums wirt deshalb in jedem Augenblick auf den externen Seitenspeicher ausgelagert.

Eine Möglichkeit der Nutzung des virtuellen Speicherkonzeptes besteht darin, den virtuellen Speicher in mehrere Regionen aufzuteilen, wobei in jeder Region ein Prozess läuft. Hierbei ist in jedem Augenblick ein Teil der Seiten einer jeden Region im realen Hauptspeicher abgebildet; der Rest befindet sich auf dem externen Seitenspeicher. Der Inhalt der Regionen verändert sich auf Grund des Demand Paging Konzeptes ständig; der Benutzer Adressenraum wird deshalb als der Dynamische Bereich bezeichnet.

Der Inhalt der Rahmen des realen Hauptspeichers ändert sich ständig, da mittels des Demand Paging ständig Seiten zwischen dem Hauptspeicher und dem externen Seitenspeicher hin- und herbewegt werden.

Hierzu gibt es eine Ausnahme: Seiten, die den Überwacher enthalten, werden aus Performance Gründen ständig im Hauptspeicher gehalten. Weiterhin sind die virtuellen Adressen des Überwachers identisch mit den realen Adressen. Man bezeichnet den Überwacher deshalb auch als den Residenten Überwacher. Der Überwacher belegt die untersten Adressen, beginnend mit der hexadezimalen Adresse 000---000 .

Streng genommen ist diese Darstellung stark vereinfacht. Tiefer gehende Details werden aber für die folgenden Erläuterungen nicht benötigt.

2.2.9 Abbildung der virtuellen Adressen

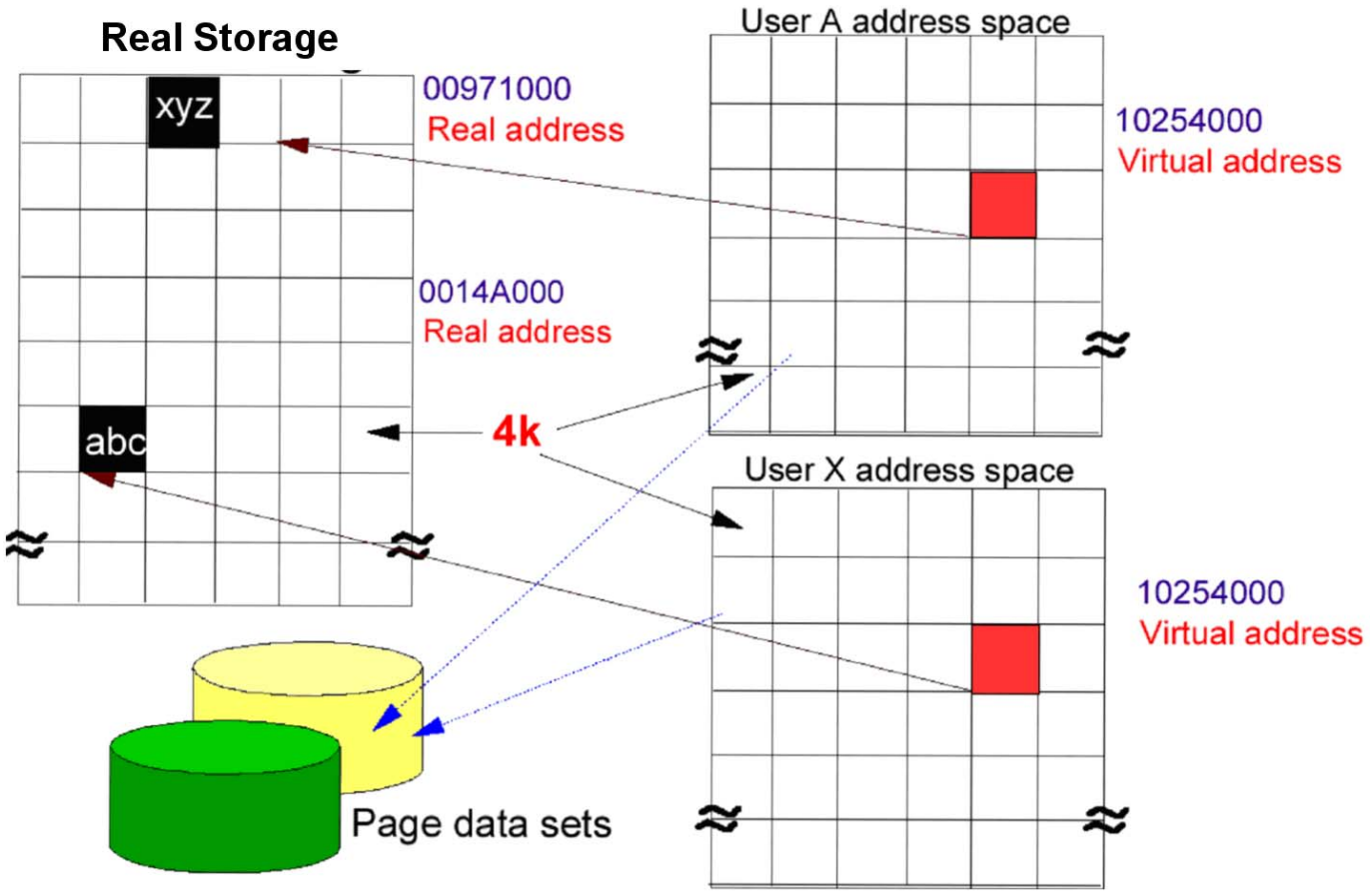


Abb. 2.2.9
Adressenabbildung

Gezeigt ist, wie zwei Seiten in getrennten virtuellen Adressräumen, aber mit identischen virtuellen Adressen, auf unterschiedliche reale Rahmenadressen abgebildet werden.

Seiten werden während der Prozessausführung mehrfach auf den Externen Seitenspeicher ausgelagert und später wieder in den Hauptspeicher eingeräumt. Hierfür werden vermutlich andere Rahmen benutzt; die reale Hauptspeicheradresse ändert sich zwischen Auslagern und Einräumen.

2.2.10 Mehrfache virtuelle Adressenräume

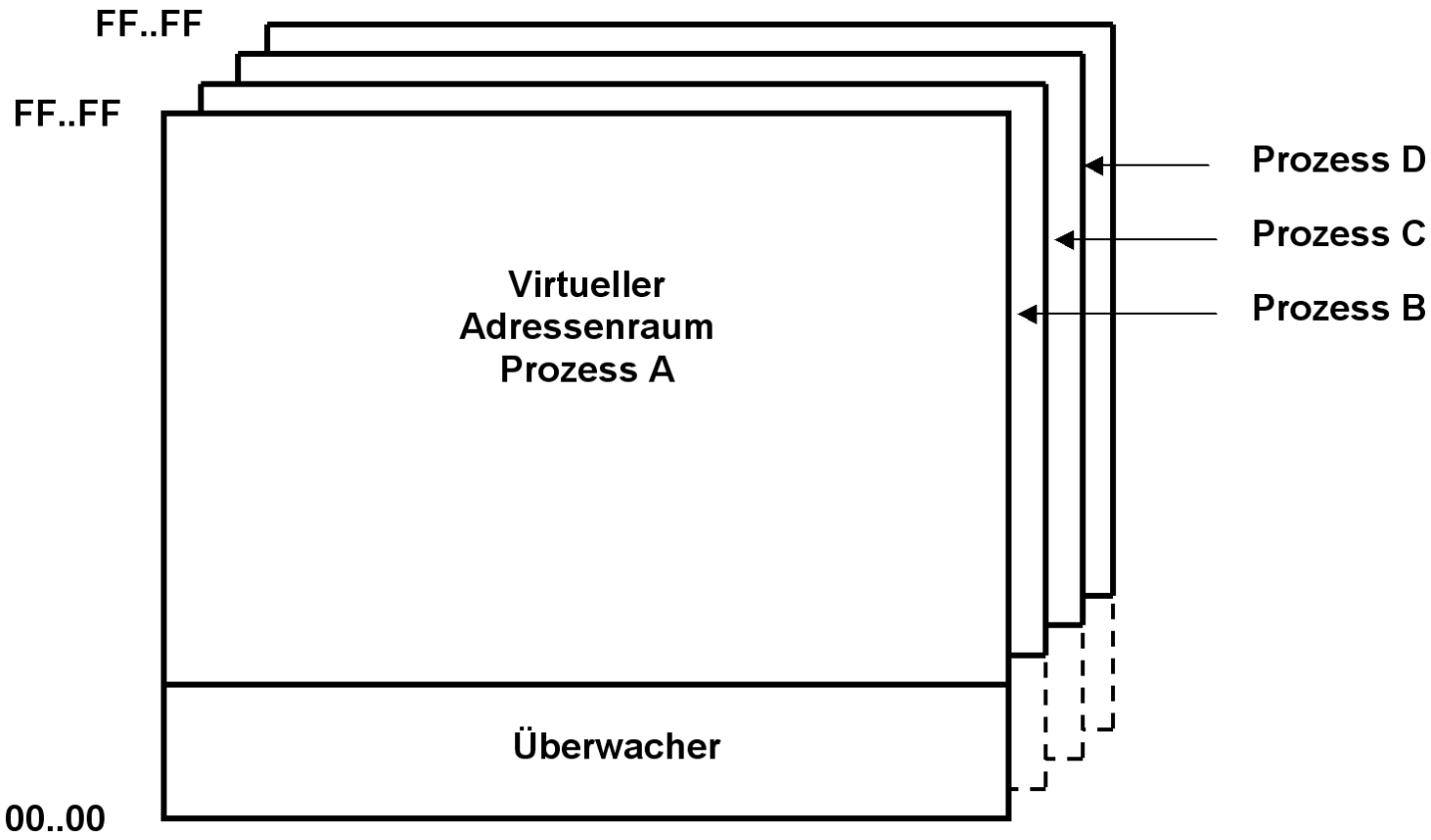


Abb. 2.2.10
Mehrfache virtuelle Adressenräume

Da der virtuelle Speicher sehr viel größer als der reale Hauptspeicher sein kann, ist es möglich, den Prozessen A, B, C und D jeweils einen virtuellen Speicher maximaler Größe zuzuorden. In diesem Fall benutzen unterschiedliche virtuelle Speicher identische Adressen. Der Adressenbereich der virtuellen Speicher geht von Hex 000..000 bis zu einer maximalen Größe, theoretisch bis zu FFF...FFF, z.B. 2^{31} oder 2^{64} Bytes. Da der reale Hauptspeicher viel kleiner ist, wird ein Großteil der Seiten auf dem externen Seitenspeicher abgespeichert.

Jeder Prozess hat einen eigenen virtuellen Adressenraum. z/OS bezeichnet den virtuellen Adressenraum (virtual Address Space) eines Prozesses auch als Region.

2.2.11 Feste Adressen für den Überwacher

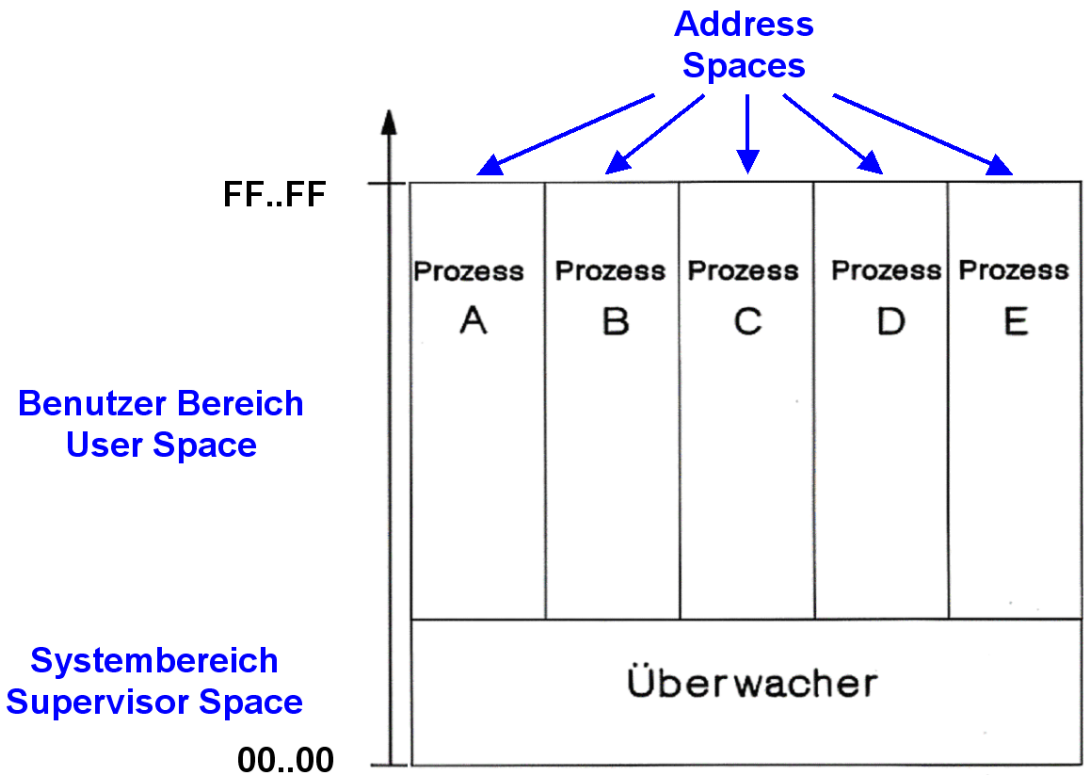


Abb. 2.2.11
Addressspaces können den ganzen Adressenbereich abdecken

Die Abbildung der virtuellen Adressen auf unterschiedliche reale Adressen erfolgt, in dem man jedem Prozess eine eigene Seitentabelle zuordnet.

Der Systembereich (Supervisor Space) mit dem Überwacher ist nur einmal vorhanden, und ist Bestandteil aller virtuellen Adressenräume. Für den Systembereich sind virtuelle und reale Adressen identisch.

z/OS benutzt dieses Verfahren. Der virtuelle Adressenraum eines Prozesses wird als (virtual) Address Space bezeichnet.

2.3 Betriebssystem Überwacher

2.3.1 System z Programmiermodell

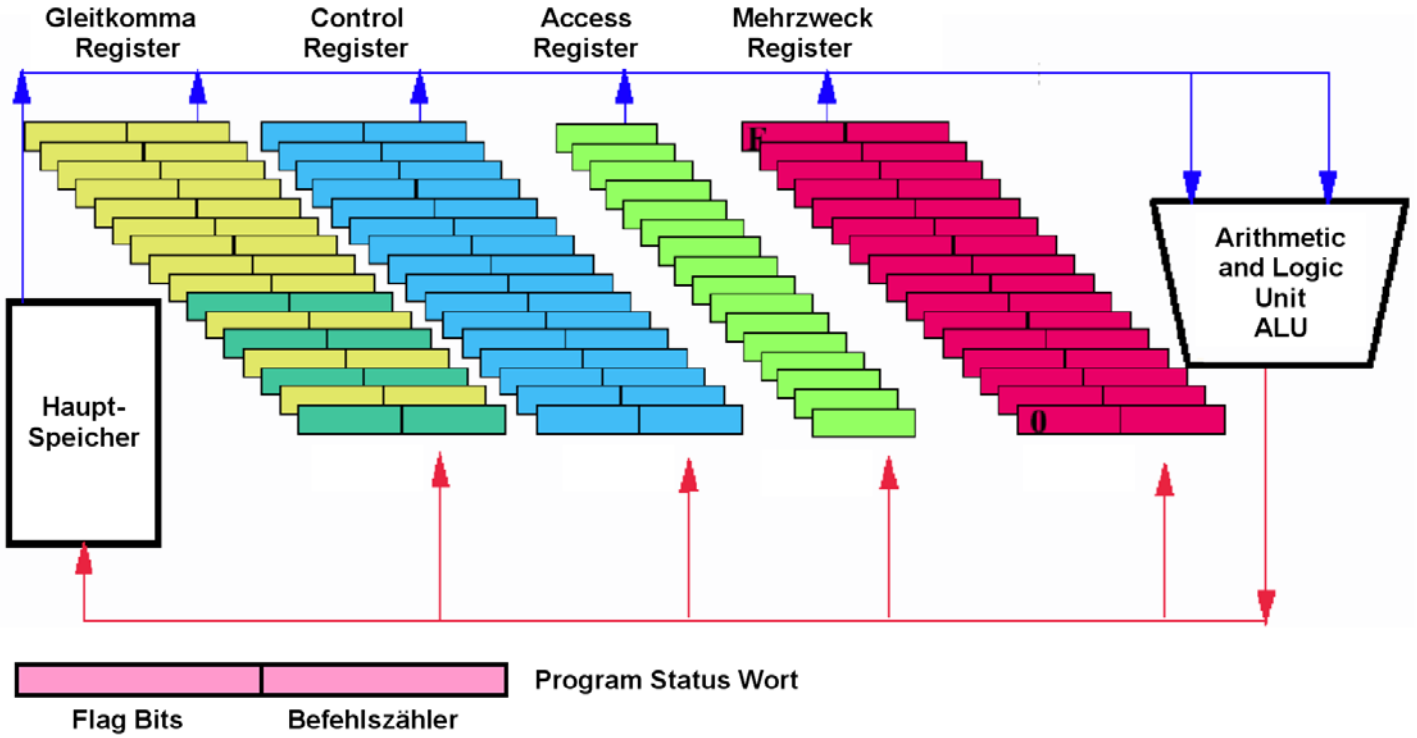


Abb. 2.3.1
Register, die dem Programmierer direkt oder indirekt zugänglich sind

Abbildung 2.3.1 zeigt die Bestandteile einer Mainframe CPU, die für den Programmierer sichtbar sind und die von Maschinenbefehlen manipuliert werden können. Man bezeichnet diese Teile auch als das Programmiermodell. Zu beachten ist, dass die CPU außerdem viele Teile und Funktionen enthält, die für einen Programmierer nicht zugänglich und unsichtbar sind.

Das Programmiermodell besteht aus dem Hauptspeicher, einer Verarbeitungseinheit (ALU) die Funktionen wie z.B. Addition ausführen kann und einer ganze Reihe von Registern, die Daten, Hauptspeicheradressen oder Steuerfunktionen speichern können. Der Bestand an System z Registern umfasst:

- **16 Mehrzweck Register (General Purpose Register, GPR, 64 Bit) speichern Adressen oder Daten.**
- **16 Gleitkommaregister (64 Bit) speichern Zahlen im Gleitkommaformat.**
- **16 Control Register (64 Bit) speichern Steuerfunktionen. Ein Beispiel ist Steuerregister 1, welches die Anfangsadresse der Seitentafel im Hauptspeicher enthält.**
- **16 Access Register (32 Bit) werden für spezielle Hauptspeicher Zugriffsfunktionen benutzt.**

Zusätzlich existiert ein 64 Bit Befehlszähler Register und ein Flag Bit Register. Der Befehlszähler enthält die Adresse des nächstens auszuführenden Maschinenbefehls. Das Flag Register enthält individuelle Bits. Ein Beispiel ist ein Bit, welches den Rechner entweder in den Benutzerstatus oder in den Überwacherstatus versetzt.

Als eine Besonderheit der System z Architektur werden Befehlzähler und Flag Bits zu einem einzigen 128 Bit langen Register, dem „Programm Status Wort“ (PSW) Register zusammengefasst. Vom Standpunkt der Ausführung eines Programms definiert der Inhalt des PSW in jedem Augenblick den Status der CPU.

Bei einem CPU Chip mit mehreren Cores enthält jedes Core einen eigenen Satz der hier gezeigten Register.

2.3.2 Überwacherstatus und Problemstatus

Eine CPU läuft in jedem Augenblick entweder im Überwacher Status (Supervisor State) oder im Benutzer Status (user State).

Der Überwacher Status bzw. Benutzer Status wird durch 1 Bit im Flag Bit Register Teil des Program Status Wortes der Zentraleinheit definiert.

Der Überwacher (Kernel) läuft im Überwacherstatus (Supervisor State, Kernel State).

Benutzerprogramme (Anwendungsprogramme) laufen im Problemstatus (Problem State, User State).

Auswirkungen sind:

- **Bestimmte „Privilegierte“ Maschinenbefehle können nur im Überwacherstatus ausgeführt werden**
- **Speicherschutz. Im Problemstatus kann nur auf einen Teil des virtuellen und des realen Hauptspeichers zugegriffen werden**

2.3.3 Unterbrechungen

Unterbrechungen (Interruptions) sind ein zentrales Steuerelement in einem jeden Rechner. Unterbrechungen bewirken eine Änderung des Status der Zentraleinheit als Folge von Bedingungen (Ursachen), die entweder

- **außerhalb der Zentraleinheit (CPU), oder**
- **innerhalb der Zentraleinheit**

auftreten.

Unterbrechungen bewirken den Aufruf und die Ausführung von speziellen Programmen (Unterbrechungsroutinen) außerhalb des normalen Programmablaufs.

Eine Unterbrechung bewirkt:

1. **Abspeichern des PSW (Programm-Status-Wort, Flag Bits und Befehlszählers) im Hauptspeicher.**
2. **Laden des Befehlszählers mit der Anfangsadresse einer Unterbrechungsroutine (Interrupt Handler).**
3. **Status-Initialisierung im PSW (z.B. Überwacherstatus setzen).**
4. **In der Regel: Abspeichern der Mehrzweckregister, evtl. auch Steuerregister, Gleitkommaregister durch die Unterbrechungsroutine.**

Typischerweise enthält ein Feld im Hauptspeicher zusätzliche Informationen über die Ursache der Unterbrechung.

Unterschiedliche Bedingungen können unterschiedliche Klassen von Unterbrechungen auslösen. Es existieren sechs Unterbrechungsklassen mit mehreren Unterbrechungsarten / Klasse

Maschinenfehler	**Beispiel Paritäts-Fehler (Bus, Hauptspeicher), fehlerhafte Addition**
Reset	**Beispiel: Setzt Zentraleinheit (und System) in jungfräulichen Zustand**
I/O	**Beispiel: Signalisiert den Abschluss einer E/A Operation**
Extern	**Beispiel: System externes Signal, z. B. Ablauf des Zeitgebers.**
Programm	**Beispiele: Division durch Null, Fehlseitenunterbrechung, illegaler OP Code, illegale Adresse**
Systemaufruf	**(SVC) Aufruf des Überwachers im Benutzerprogramm (eigentlich ein Maschinenbefehl)**

2.3.4 Unterbrechungsverarbeitung

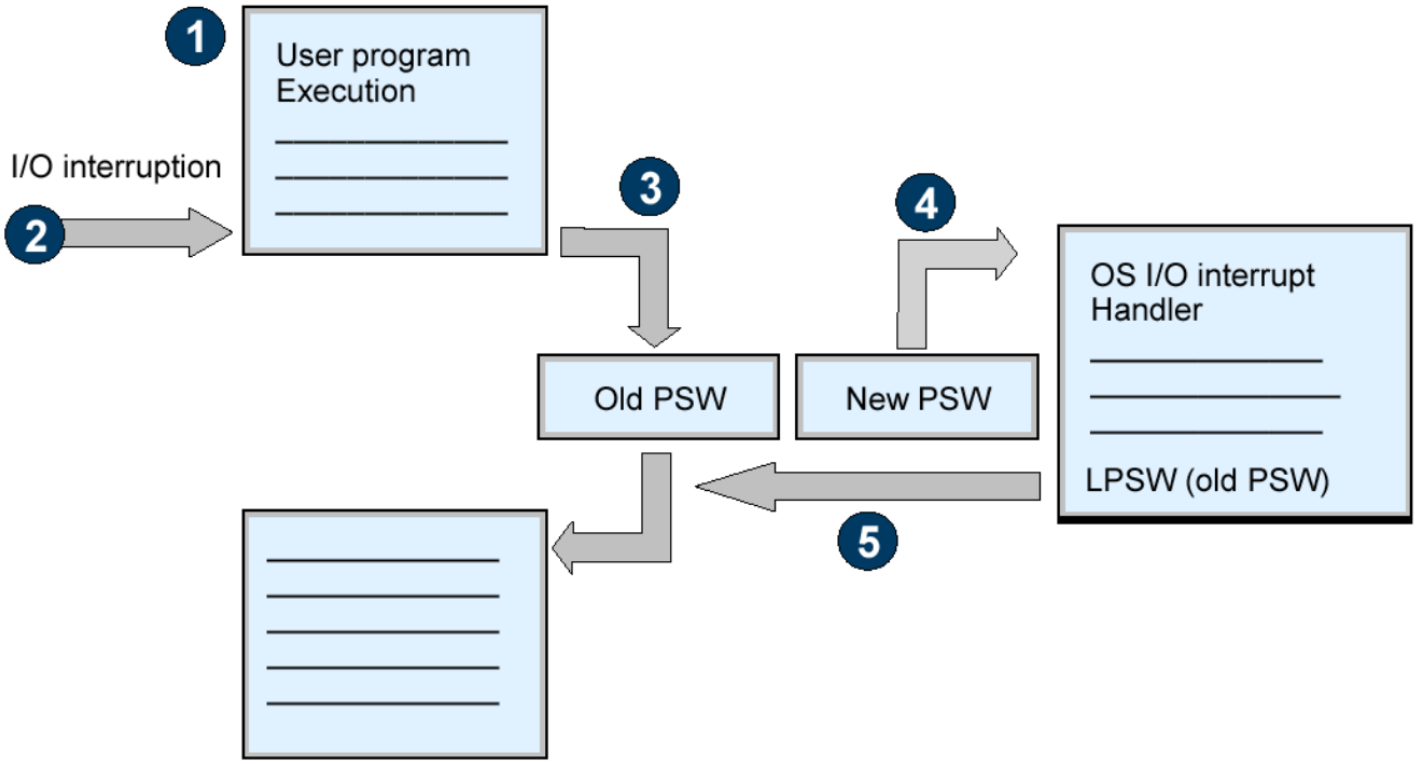


Abb. 2.3.2
Verarbeitung einer Unterbrechung

Eine Unterbrechung bewirkt das Abspeichern des derzeitig gültigen PSW (als old PSW bezeichnet) im Hauptspeicher an einer hierfür vorgesehenen Adresse und ein Laden eines neuen PSW (new PSW) in das PSW Register. Der Befehlszählerteil des neuen PSW enthält die Adresse des ersten Befehls der Unterbrechungsroutine. Die CPU befindet sich automatisch im Überwacherstatus.

Als allerletzten Befehl führt die Unterbrechungsroutine den Maschinenbefehl „LPSW (Load Programm Status) aus. Dieser bewirkt, dass das old PSW in das PSW Register geladen wird. Der Befehlszählerteil des PSW enthält die Adresse des Maschinenbefehls, den die CPU ohne Eintreten der Unterbrechung als nächstes ausgeführt hätte.

2.3.5 Schichtenmodell der Rechnerarchitektur

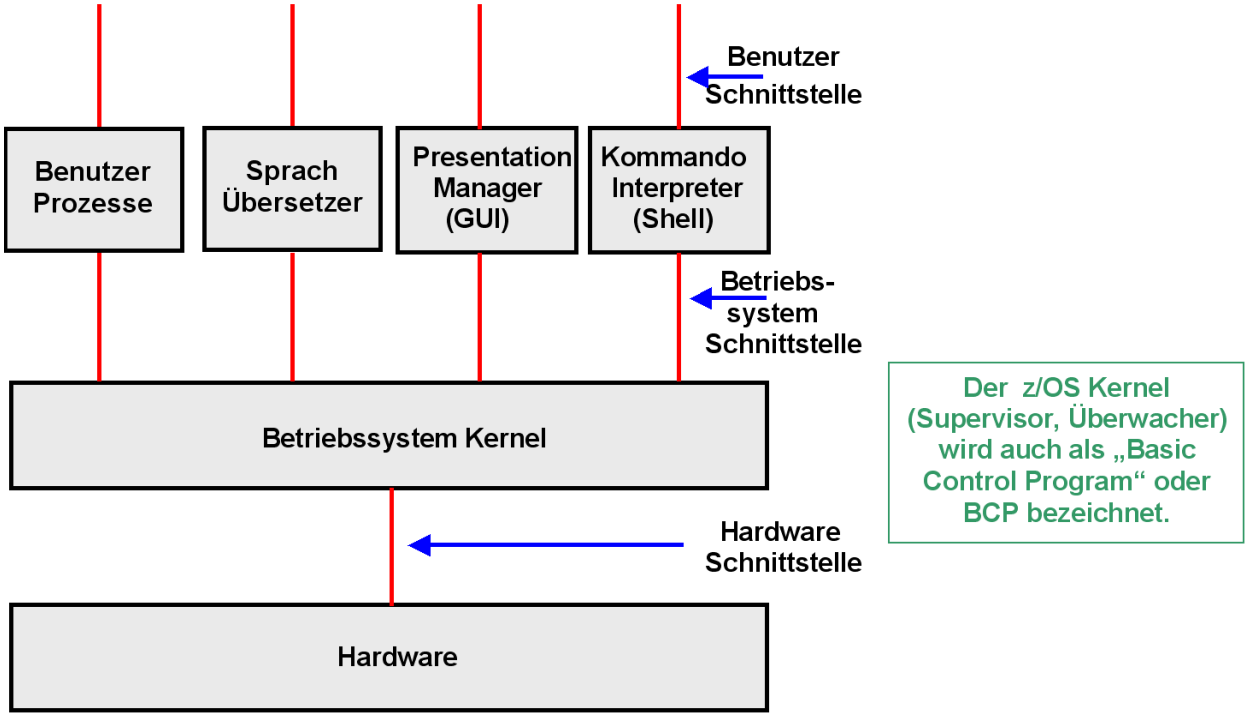


Abb. 2.3.3
Schichtenmodell der Rechnerarchitektur

Das Betriebssystem besteht aus einer zentralen Steuerungs- und Verwaltungsfunktion, dem Überwacher (Supervisor, Kernel), und weiteren Systemprogrammen oder Komponenten, die unter z/OS als „Subsysteme“ bezeichnet werden. Beispiele für Subsysteme sind der Kommandointerpreter (TSO), das Job Entry Subsystem(JES) oder das DB2 Datenbanksystem. Subsysteme sind Systemprozesse, die parallel zu den Benutzerprozessen laufen.

Der Überwacher (Supervisor) besteht aus :

1. **Datenbereichen (Control Blocks)**
2. **Programmteilen, welche Controlblock Daten manipulieren**

Der Überwacher ist In der Regel nicht strukturiert. In seinem Buch “Modern Operating Systems” published in 1992, pp.18, bezeichnet Prof. Andrew s. *Tanenbaum den Überwacher eines Betriebssystems als „.* The Big Mess“.

Der Überwacher enthält Funktionen die von vielen Prozessen gemeinsam genutzt werden. Häufig verbringt ein Prozess 50 % der Ausführungszeit (Pfadlänge) mit der Ausführung von Überwacherfunktionen (läuft 50 % der Zeit im Überwacherstatus).

Die Hardware des Rechners reagiert auf die Eingabe von Maschinenbefehlen und auf Unterbrechungen. Der Überwacher kann nur über Unterbrechungen aufgerufen werden. Er läuft im Überwacherstatus.

System - Aufrufe (System Calls) sind die einzige Möglichkeit für Benutzerprozesse, mit dem Überwacher zu kommunizieren. Unter z/OS implementiert der SVC Maschinenbefehl die System Call Funktion. Beim x86 hat der INT Maschinenbefehl die gleiche Funktion. Ein System Call ist eine Routine, die im Benutzer Status läuft, u.a. den SVC (oder INT) Maschinenbefehl ausführt, und dadurch den Übergang vom Benutzerstatus in den Überwacherstatus herbeiführt.

2.3.6 Systemaufruf (System Call)

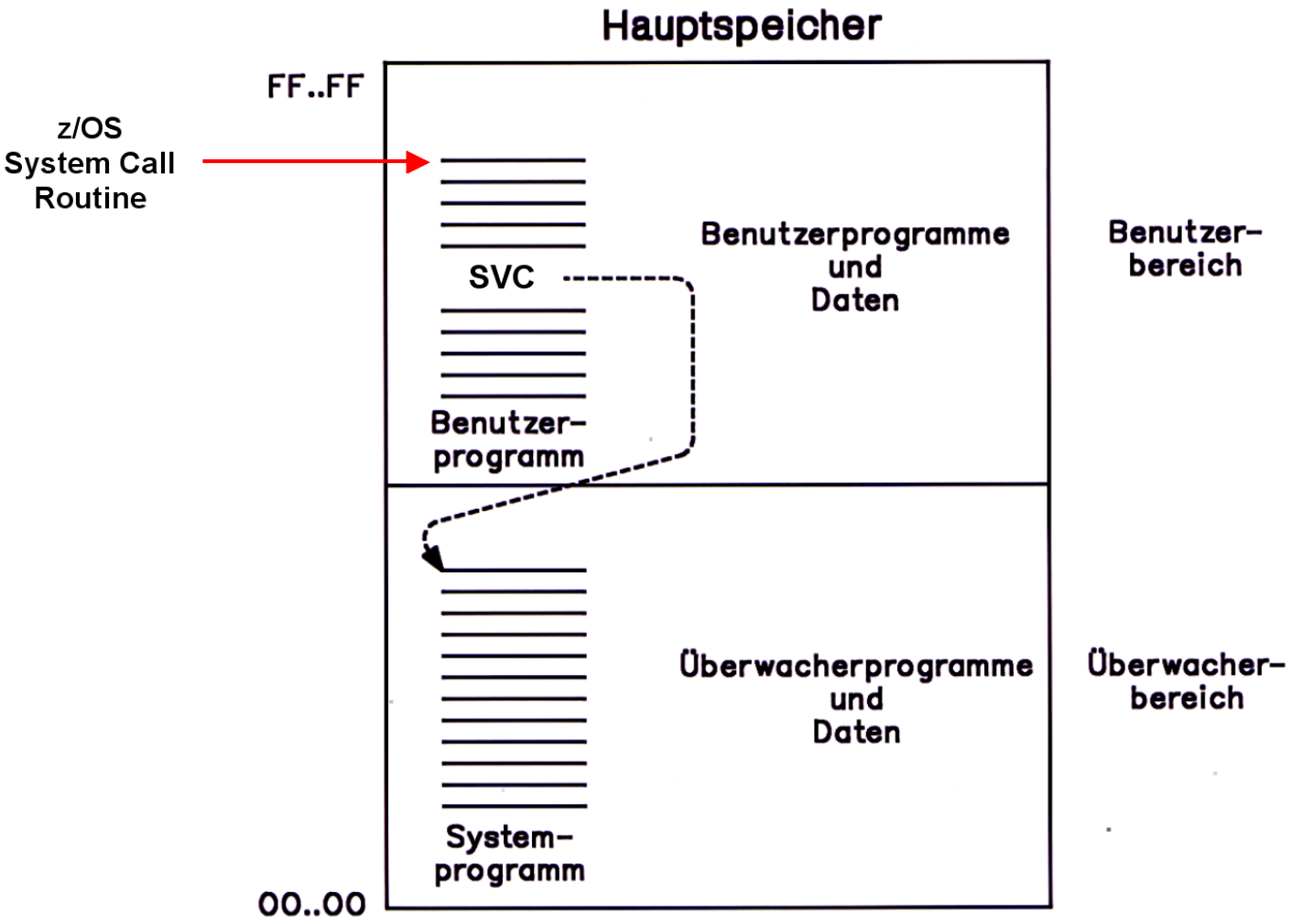


Abb. 2.3.4
Supervisor Call

Ein System z Benutzerprogramm kann eine Funktion des Überwachers durch Ausführung des SVC (Supervisor Call) Maschinenbefehls aufrufen. Der SVC Befehl übergibt hierzu einen Parameter, welcher die Art der gewünschten Funktion angibt. Ein Beispiel ist die Durchführung einer WRITE operation, welche Daten auf den Plattenspeicher schreibt. Die Ausführung des SVC Maschinenbefehls bewirkt eine Unterbrechung. Die Unterbrechungsroutine bewirkt unter anderem den Wechsel von Benutzerstatus in den Überwacher Status (Kernel Status).

2.3.7 Struktur des Supervisors

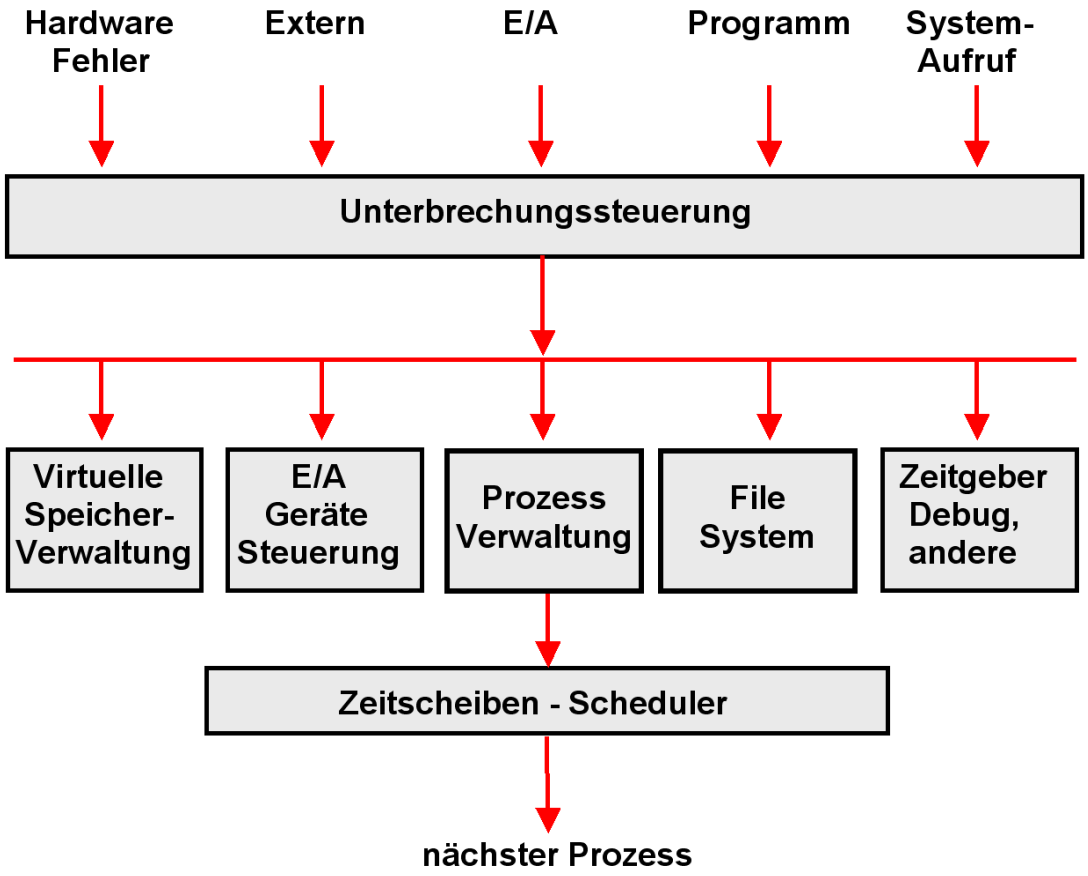


Abb. 2.3.5
Die wichtigsten Komponenten des Überwachers

Der Aufruf des Überwachers (Supervisor, Kernel) erfolgt grundsätzlich über Unterbrechungen.

Je nach Art der Unterbrechung werden unterschiedliche Komponenten des Überwachers aufgerufen.

Benutzerprozesse nehmen Dienste des Überwachers über eine architekturierte Schnittstelle, den Systemaufruf (System Call, Supervisor Call, SVC) in Anspruch. Diese Begriffe haben alle die gleiche Bedeutung.

Der Scheduler (Zeitscheibensteuerung) sucht den nächsten auszuführenden Prozess aus.

Der Überwacher wird über eine Unterbrechung aufgerufen. Dies ist der einzige Weg, über den man eine Funktion des Überwachers in Anspruch nehmen kann.

2.3.8 Funktionen des Überwachers

Die wichtigsten Überwacherfunktionen sind:

- **Die Unterbrechungssteuerung untersucht die Art der aufgetretenen Unterbrechung und ruft je nach Art eine andere Komponente des Überwachers auf.**
- **Die virtuelle Speicherverwaltung ordnet virtuellen und realen Speicherplatz zu. Sie bestimmt, welche Seiten in Rahmen des Hauptspeichers abgebildet werden und welche Seiten auf den externen Seitenspeicher (Auxiliary Store) ausgelagert werden. Sie lädt bei Bedarf Seiten vom externen Seitenspeicher in den Hauptspeicher.**
- **Die Input/Output Steuerung (I/O Supervisor) wird aufgerufen wenn ein Benutzerprogramm eine Lese oder Schreiboperation auf ein I/O Gerät (z.B. Festplattenspeicher) durchführen will. Sie nimmt auch eine I/O Unterbrechung entgegen, die z.B. besagt, dass ein Plattenspeicher eine I/O Operation erfolgreich abgeschlossen hat.**
- **Die Prozessverwaltung aktiviert und deaktiviert Prozesse. Aktive Prozesse verfügen über Ressourcen im Hauptspeicher. Bei deaktivierten Prozessen sind alle Resourcen (vermutlich) auf einen Plattenspeicher ausgelagert.**
- **Datenstrukturen auf Plattenspeichern werden mit Hilfe des File Systems verwaltet. Windows verwendet die NTFS und FAT32 File Systems. VSAM, PDSe und zFS sind die wichtigsten z/OS File Systems.**
- **Zahlreiche weitere Funktionen wie Zeitscheibensteuerung und Funktionen zum debuggen von Software sind vorhanden.**
- **Der Scheduler verwaltet die Warteschlangen der TCBs. Er versetzt Prozesse in den Zustand wartend, ausführbar oder laufend. Er steuert Prioritäten, nach denen entschieden wird, welcher Prozess vom Zustand wartend in den Zustand ausführbar überführt wird.**

Ein Prozess wird durch seinen TCB repräsentiert und dargestellt. Der TCB ist ein Bereich im Hauptspeicher und enthält Informationen über den Prozess.

Mehrzweck Register , Gleitkommaregister usw. sowie das PSW sind in einer CPU nur einmal vorhanden. Wird ein Prozess vom laufenden in den wartenden Zustand versetzt, wird der Inhalt der Register und des PSW in seinem TCB abgespeichert, ebenso alle weitere Information, die den Prozess charakterisiert. Wird der gleiche Prozess später einmal wieder vom ausführbaren in den laufenden Zustand versetzt, wird diese Information benutzt, um den bisherigen Zustand der CPU wieder herzustellen, indem der Inhalt der Register und des PSW mittels der im TCB gespeicherten Information wieder in den alten Zustand versetzt wird..

2.3.9 TCB Queues

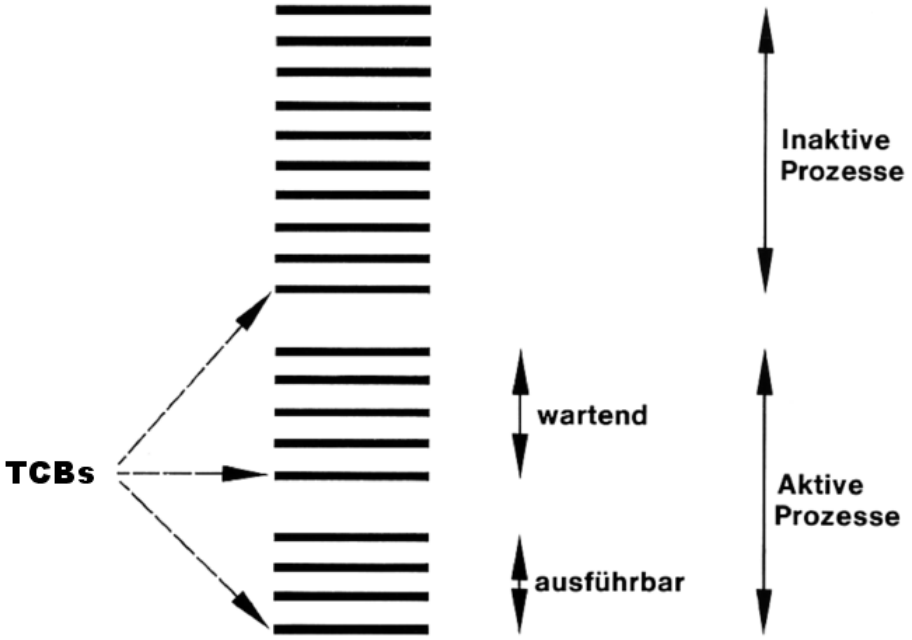


Abb. 2.3.6
Warteschlange der Task Control Blöcke

Ein Prozess wird durch einen Prozessleitblock beschrieben. Der Prozessleitblock enthält wichtige Funktionen für die Ausführung eines Prozesses.

z/OS verwendet die Bezeichnung Task Control Block (TCB). Andere Architekturen verwenden die Bezeichnung Process Control Block (PCB).

Zu jedem Zeitpunkt beanspruchen viele Prozesse Platz im Hauptspeicher und könnten ausgeführt werden. In einem Rechner mit einer CPU verarbeitet die CPU aber nur einen Prozess. In einem Mehrfachrechner mit n CPUs können gleichzeitig n Prozesse ausgeführt werden.

TCBs der wartenden und ausführbaren Prozesse werden vom Scheduler in Warteschlangen eingereiht und verwaltet.

Prozesse können inaktiv sein. Dies bedeutet, ihre TCBs und ihre Seiten sind auf einen Plattenspeicher ausgelagert. Der Scheduler entscheidet, ob und wann ein inaktiver Prozess aktiviert wird.

Ein Prozess wird durch seinen TCB repräsentiert und dargestellt. Der TCB ist ein Bereich im Hauptspeicher und enthält Informationen über den Prozess.

Mehrzweck Register , Gleitkommaregister usw. sowie das PSW sind in einer CPU nur einmal vorhanden. Wird ein Prozess vom laufenden in den wartenden Zustand versetzt, wird der Inhalt der Register und des PSW in seinem TCB abgespeichert, ebenso alle weitere Information, die den Prozess charakterisiert. Wird der gleiche Prozess später einmal wieder vom ausführbaren in den laufenden Zustand versetzt, wird diese Information benutzt, um den bisherigen Zustand der CPU wieder herzustellen.

2.3.10 Scheduler

Der Scheduler ist die Komponente des Überwachers, welcher drei Warteschlangen für die laufenden, wartenden und ausführbaren TCBs verwaltet. Er überführt bei gegebenen Anlass einen TCB von einer Warteschlange in eine andere.

Wenn eine CPU frei wird, selektiert der Scheduler aus der Warteschlange der ausführbaren TCBs einen Prozess und versetzt ihn in den Zustand laufend. Dies kann über Prioritäten gesteuert werden, und der z/OS Überwacher verfügt hierzu über eine Prioritätssteuerung.

In der Mehrzahl der Fälle bleibt ein Prozess nur kurze Zeit im Zustand laufend. Um zu verhindern, dass ein bestimmter Prozess eine CPU zu lange in Anspruch nimmt, verfügt der Scheduler über eine Zeitscheibensteuerung. Eine Zeitscheibe ist ein Zeitintervall von typischerweise wenigen Millisekunden. Verbleibt ein Prozess im Zustand laufend über seine Zeitscheibenlänge hinaus, erfolgt eine (Seitenfehler, page fault) Unterbrechung durch einen Zeitgeber. Diese bewirkt, dass der laufende Prozess in den Zustand ausführbar versetzt wird, und ein anderer Prozess dafür in den Zustand laufend versetzt wird.

In der Regel laufen unterschiedliche Prozesse auf den CPUs eines Rechners. Es ist jedoch möglich, dass ein Prozess mehr als eine CPU in Anspruch nimmt.

2.4 Cache

2.4.1 Halbleiter Speicher Chips

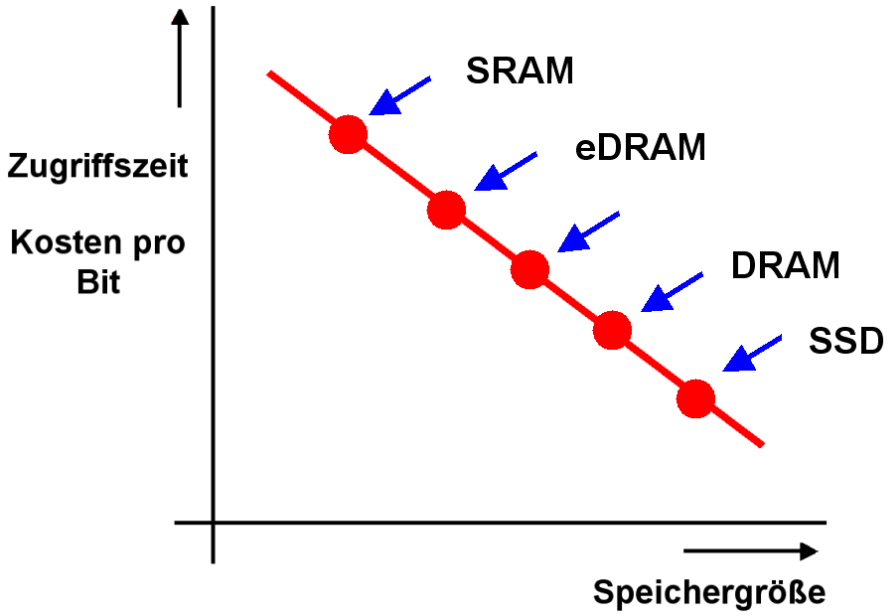


Abb. 2.4.1
Relation Zugriffszeit und Speichergröße

Es existieren eine ganze Reihe unterschiedlicher Technologien, mit denen man einen Speicher auf einem Halbleiterchip realisieren kann. Die Technologien unterscheiden sich in der Zugriffszeit auf den Speicher sowie der Speicherdichte, der Anzahl der Bits, die man pro mm^2 unterbringen kann. Die Speicherdichte beeinflusst die Kosten pro Bit. Allgemein gilt der hier dargestellte Zusammenhang: Kleine Speicher (Schnellspeicher) haben eine schnelle Zugriffszeit, brauchen viel Platz pro Bit und haben hohe Kosten pro Bit. Große Speicher haben eine längere Zugriffszeit, brauchen wenig Platz pro Bit und haben niedrige Kosten pro Bit.

Hauptspeicher Chips verwenden die DRAM (Dynamic Random Access Memory) Technologie. Das speichernde Element ist dabei ein Kondensator, der entweder geladen oder entladen ist. Über einen Schalttransistor wird er zugänglich und entweder ausgelesen oder mit neuem Inhalt beschrieben. Der Speicherinhalt ist (wie auch bei SRAM und eDRAM Chips) flüchtig (volatil), das heißt, die gespeicherte Information geht bei fehlender Stromversorgung oder zu später Wiederauffrischung verloren.

Festplattenspeicher sind dagegen statisch. Die gespeicherten Daten bleibt auch im abgeschalteten Zustand erhalten.

2.4 2 Cache Speicher

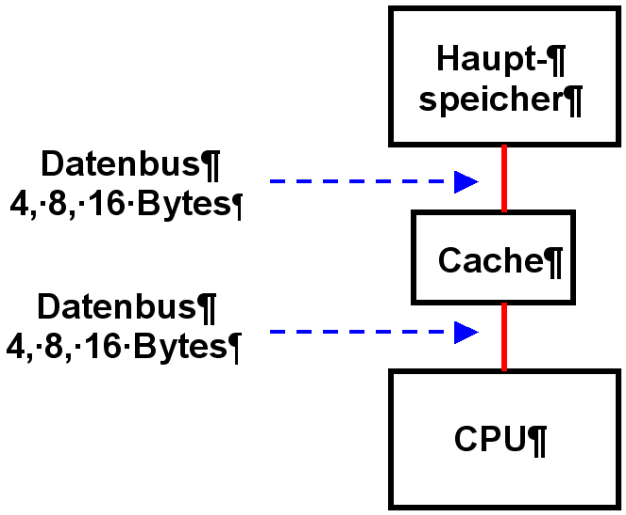


Abb. 2.4.2
Verbesserung der Hauptspeicher Zugriffszeit

Hauptspeicher Zugriffszeiten sind zu langsam, um mit der heute möglichen Verarbeitungsgeschwindigkeit einer CPU mithalten zu können. Deswegen schaltet man zwischen Hauptspeicher und CPU einen Cache Speicher. Der Cache Speicher verwendet eine SRAM (Static Random Access Memory) Technologie. Das speichernde Element ist dabei ein FlipFlop. Es existieren viele unterschiedliche SRAM Technologien, mit unterschiedlichen Zugriffszeiten, Speicherdichten und Kosten. Während mit DRAMs aufgebaute Hauptspeicher eine Zugriffszeit in der Gegend von 100 ns aufweisen, haben mit SRAMs aufgebaute Cache Speicher Zugriffszeiten zwischen 100 ps und 10 ns .

Wenn man von einem Cache redet meint man meistens einen Hauptspeicher Cache. Da es auch Plattenspeicher-Caches und andere Caches gibt ist die Unterscheidung wichtig.

Jeder moderne Rechner hat einen oder mehrere Caches. Die Existenz des Caches ist für den Benutzer praktisch unsichtbar. Es existieren auch keine Maschinenbefehle, mit denen der Programmierer den Inhalt des Caches manipulieren kann.

2.4.3 Struktur des Cache Speichers

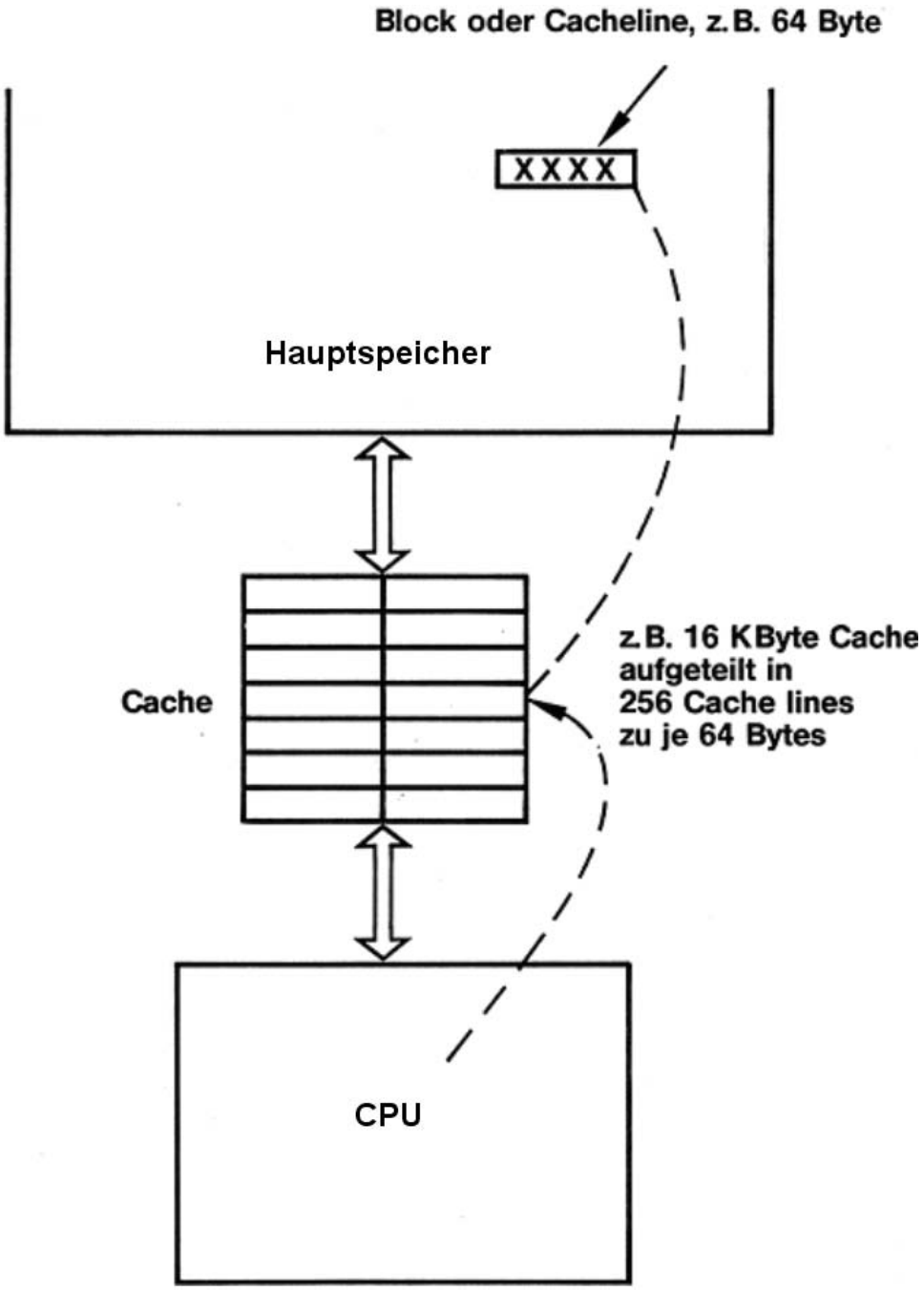


Abb. 2.4.3
Aufteilung des Cache Speichers in Cache Lines

Der Cache enthält in jedem Augenblick nur die Kopie einer Untermenge der Daten im Hauptspeicher. Diese Untermenge wird ständig ausgewechselt.

Hierzu werden Hauptspeicher und Cache Speicher in Blöcke mit einer identischen Größe aufgeteilt. Diese Blöcke werden als „Cachelines“ bezeichnet. Die Größe der Cachelines ist implementierungs-abhängig. Bei den z10 und z196 Mainframes sind es 256 Bytes, bei anderen Rechnern häufig weniger.

Ein Prozessor mit einer Cacheline-Size von 256 Byte wird aus dem Hauptspeicher immer nur Pakete dieser Größe in den Cache transportieren. Bei einem Hauptspeicher-Interface mit z.B. 256 Datenleitungen bedeutet das, dass jeder Cache Miss (wenn also angeforderte Daten nicht im Cache stehen) eine Adressierung und 8 Daten-Transferzyklen nach sich zieht.

2.4.4 Cache Directory

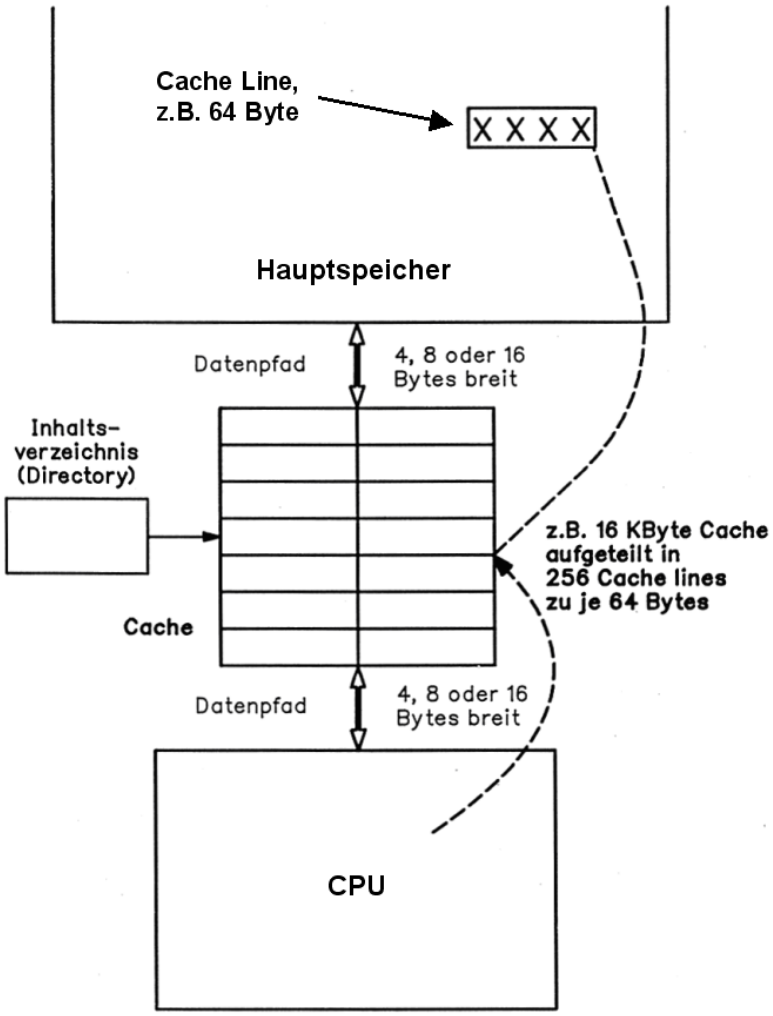


Abb. 2.4.4
Adressierung des Cache Speichers mittels des Cache Directories

Die Anordnung der Cachelines ist willkürlich und ändert sich ständig. Die CPU konsultiert ein „Cache Directory“ (Inhaltsverzeichnis) um eine bestimmte Cacheline innerhalb des Caches zu finden.

Das Cache Directory enthält je einen Eintrag für jede im Cache gespeicherte Cache Line. Der Eintrag enthält die Adresse der Cacheline im Hauptspeicher und im Cache.

Bei jedem Hauptspeicherzugriff durchsucht die CPU das Cache Directory in der Hoffnung, dass die benötigte Cacheline sich im Cache befindet. Wenn das nicht der Fall ist entsteht ein Cache Miss. Dies bewirkt, dass die benötigte Cache Line in ihrer Gesamtheit vom Hauptspeicher in den Cache geladen wird. Vermutlich muss dabei eine andere (nicht mehr benötigte) Cache Line aus dem Cache ausgelagert werden um Platz zu schaffen.

2.4.5 Split Cache

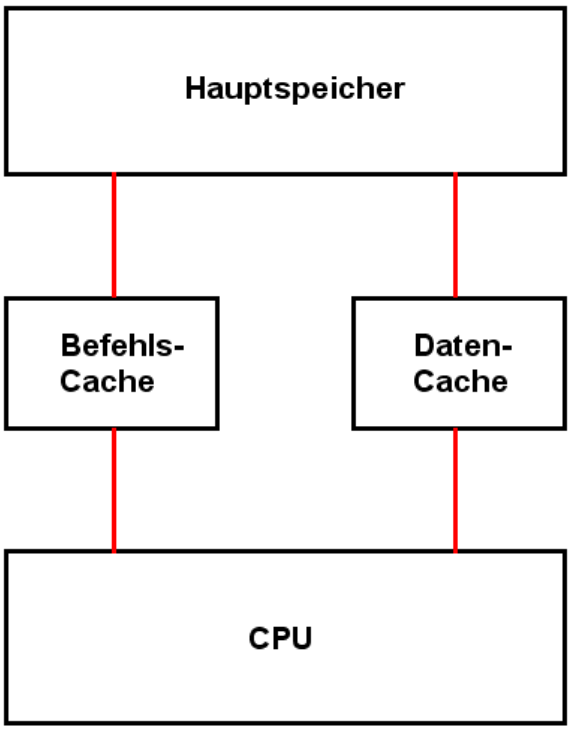


Abb. 2.4.5
Maschinen Befehle und Operanden in getrennten Cache Speichern

Im Hauptspeicher sind Programme (bestehend aus Maschinenbefehlen) und Daten gespeichert. Diese befinden sich in unterschiedlichen Hauptspeicherbereichen und damit auch in unterschiedlichen Cachelines.

Um den Durchsatz zu verbessern besteht der Cache häufig aus zwei unabhängigen Cache Speichern (Split Cache, Dual Cache): Der Befehlscache enthält nur Maschinenbefehle und der Datencache enthält nur Daten.

Der z9, z10, z196 und zEC12 Cache ist ein Dual Cache.

Der Befehlscache wird häufig als I-Cache (Instruction Cache) und der Datencache als D-Cache bezeichnet.

2.4.6 Second Level Cache

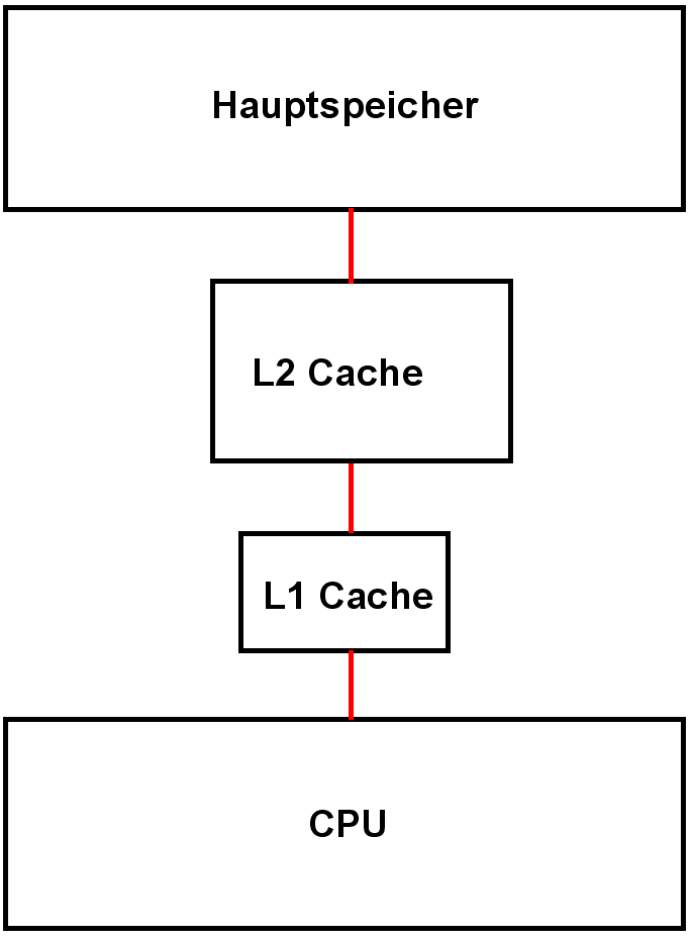


Abb. 2.4.6
SRAM Technologie mit unterschiedlichen Zugriffszeiten

Es existieren zahlreiche SRAM Technologien um Caches zu implementieren. Heute ist es üblich, mit einer Cache Hierarchie zu arbeiten. Hierbei wird ein „Level 1“ Cache (L1) mit besonders schnellen, aber teuren und platzaufwendigen SRAM Speicherzellen implementiert. Ein „Level 2“ Cache (L2) verwendet langsamere aber dafür kostengünstigere SRAM Zellen.

Hierbei wird häufig, z.B. beim z9 Mainframe, der L1 Cache als Split Cache und der L2 Cache als Uniform Cache implementiert

2.4.7 Mainframe Cache Hierarchien

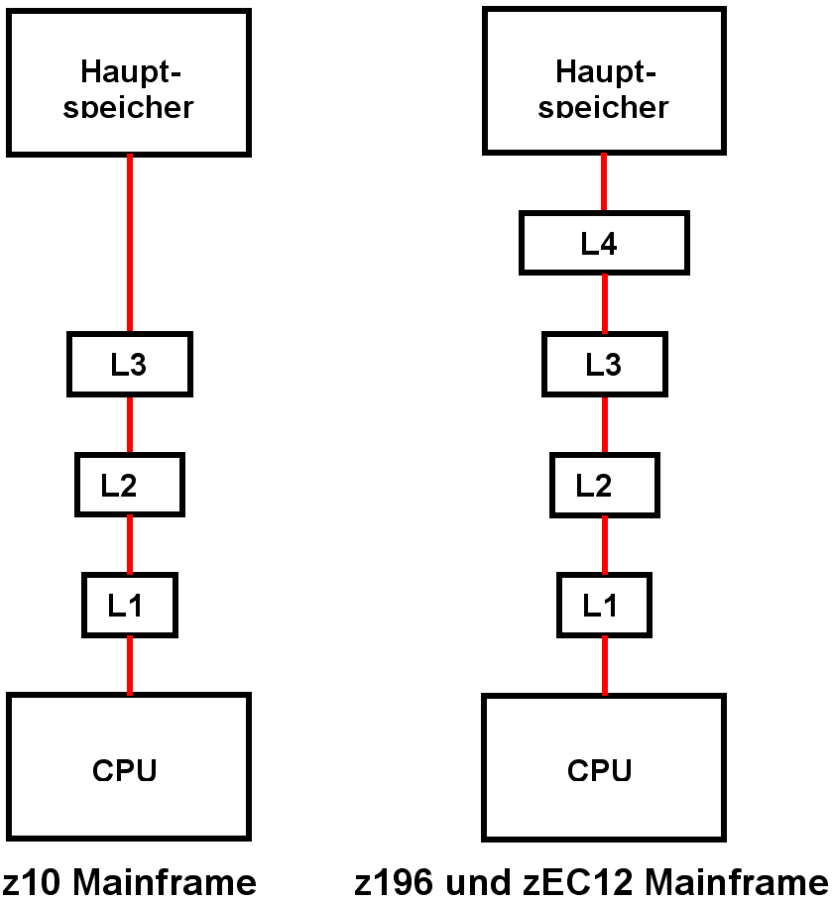


Abb. 2.4.7
Die L3 und L4 Stufen benutzen die langsamere eDRAM Technologie

Moderne Cache Hierarchien sind noch komplizierter. Gezeigt sind die 3-stufige z10 Cache Hierarchie und die 4-stufige z196 und zEC12 Cache Hierarchie. Sinnvoll wurde dies durch die Erfindung einer neuartigen als eDRAM bezeichneten Cache Speicherzellen-Technologie, die im z196 und zEC12 Mainframe die SRAM Zellen in den L3 und L4 Caches ersetzt.

2.5 Weiterführende Information

Die hier folgende Präsentation ist kein Prüfungsstoff.

Aber vielleicht interessiert Sie, wie sich die Rechentechnologie im Laufe der Jahrtausende entwickelt hat.

Sie können die Entwicklung verfolgen unter

http://www.cedix.de/VorlesMirror/Band1/History.pdf

Das folgende Video zeigt eine Demonstration des Antikythera Computers:
http://www.cedix.de/VorlesMirror/Band1/Antikythera.html
Vorsicht: der Download umfasst 67 MByte - bitte warten. Das Original ist zu finden unter:

http://www.cedix.de/VorlesMirror/Band1/Antikythera-M4-DivxQ85.avi

3. z/OS Betriebssystem

3.1 Job Control Language

3.1.1 System z und S/390 Betriebssysteme

Im Laufe der Jahre sind eine ganze Reihe von Betriebssystemen für die System z Plattform entstanden:

Name	Hersteller	
• z/OS	IBM	große Installationen (früher als OS/390oder MVS bezeichnet)
• z/VSE	IBM	mittelgroße Installationen
• z/VM	IBM	Virtualisierung, Software Entwicklung
• z/TPF	IBM	spezialisierte Transaktionsverarbeitung
• zLinux	Public Domain	Normale Suse oder Red Hat Adaption für System z Hardware
• UTS 4	Amdahl	Unix Betriebssystem, based on System V, Release 4 (SVR4)
• Open Solaris	Sun	Open Solaris Adaption für System z Hardware
• BS2000	Siemens/Fujitsu	von Siemens entwickelte Alternative zu OS/390

Alle System z bzw. S/390 Betriebssysteme sind Server Betriebssysteme, optimiert für den Multi-User Betrieb. Eine sehr ungewöhnliche Rolle spielt das z/TPF Betriebssystem.

3.1.2 System z Transaction Processing Facility

Das System z Transaction Processing Facility (z/TPF) Betriebssystem wurde ursprünglich als Platzreservierungssystem für die Fluggesellschaft American Airlines entwickelt und wird heute sowohl für Reservierungen für Fluggesellschaften, Hotels, Reisebüros, Mietwagenfirmen und Eisenbahngesellschaften als auch (vorrangig) für die Steuerung von Geldausgabe-Automaten eingesetzt. TPF unterscheidet nicht zwischen Kernel- und User-Status; sämtliche Anwendungen laufen aus Performance-Gründen im Kernel Status. Es existieren weltweit etwa 300 (sehr große) Installationen. Die Visa Kreditkartenverifizierung und das AMADEUS-Flugplatzreservierungssystem der Deutschen Lufthansa sind Beispiele für den Einsatz von TPF.

z/TPF kann nur für die Transaktionsverarbeitung eingesetzt werden, bietet nur sehr einfache Funktionen, ist aber mit dieser Einschränkung der weltweit leistungsfähigste Transaktionsserver. z/TPF Anwendungen werden auch heute noch häufig in Assembler programmiert.

Ein Beispiel ist die Firma Worldspan, ein weltweiter Anbieter von Reise-Reservierungs-Systemen. Worldspan hat sechs System z Rechner mit TPF installiert. Als Provider von elektronischen Datendiensten stellt Worldspan hiermit circa 700 Anbietern von Reiseangeboten und Millionen von Reisenden weltweit eine gemeinsame Plattform zur Verfügung.

Worldspan setzt die z/TPF Mainframe Server ein, um sowohl Reisebüros als auch Anbietern von Online-basierten Reisediensten die Möglichkeit zur Nutzung des weltweiten Global Distribution System (GDS) zu geben, über das zum Beispiel die Bestellung und Buchung von Reiseprodukten wie Flugzeugtickets, Hotels, Mietwagen und andere Reisedienstleistungen durchgeführt wird.

Durch die Nutzung von z/TPF ist Worldspan in der Lage, 17 000 Kundenanfragen pro Sekunde auf den Mainframes zu beantworten.

3.1.3 z/OS Grundstruktur

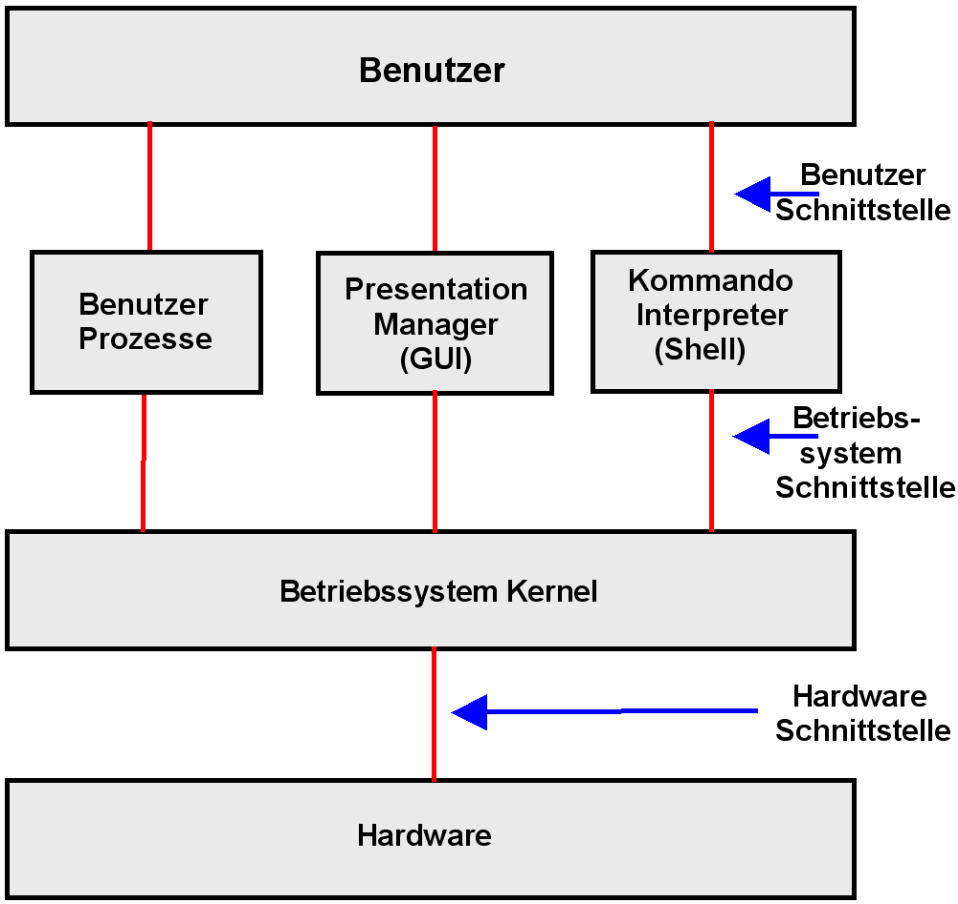


Abb. 3.1.1
Schichtenmodell der Rechnerarchitektur, Windows, Linux, Unix, z/OS

Bei allen Rechnern setzt der Betriebssystem Kernel (Überwacher, Supervisor, Basic Control Programm) direkt auf der Hardware auf. Der Kernel kann nur über wohl definierte Schnittstellen aufgerufen werden; in der Praxis sind das meistens Unterbrechungen.

Auf dem Kernel setzen Benutzerprozesse (von einem Benutzer geschriebenen Anwendungen) oder Systemprozesse auf. Systemprozesse werden auch als Subsysteme bezeichnet. Zwei der wichtigsten Systemprozesse sind der Presentation Manager (Graphical User Interface, GUI, z.B. Windows Desktop, Motiv, KDE) und der Kommando Interpreter (z.B. Windows DOS Shell, Unix Shell. TSO).

z/OS arbeitet fast ausschließlich mit verschiedenen Kommando Interpretern. GUIs werden vor allem für Anwendungen benutzt.

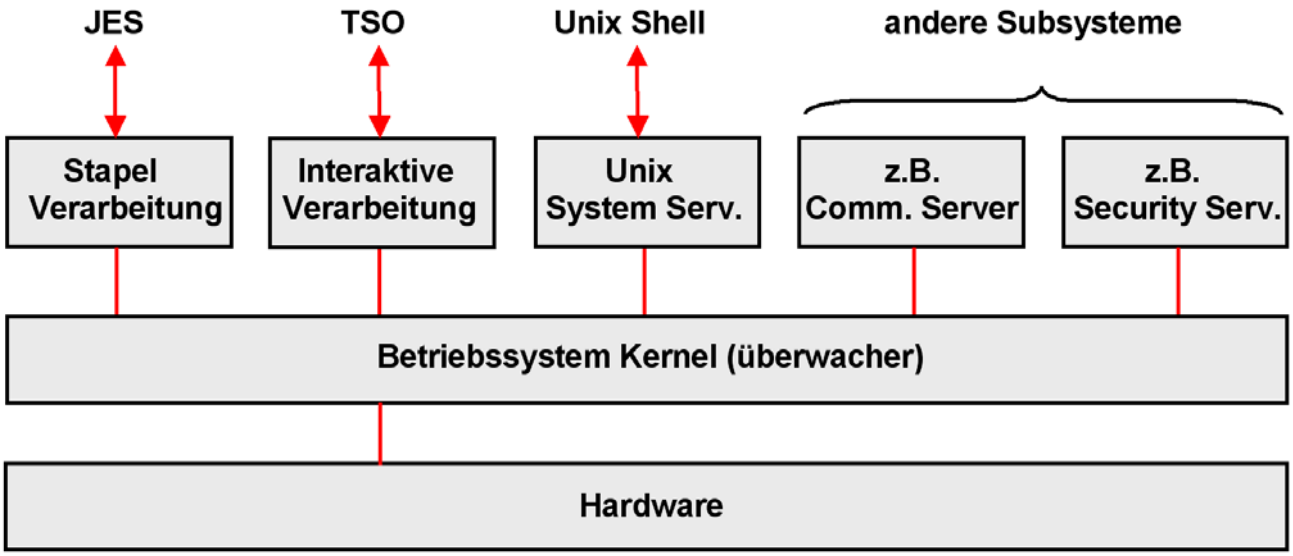


Abb. 3.1.2
z/OS Grundstruktur

Die drei wichtigsten z/OS Subsysteme (Shells) für die Steuerung des Systems sind:

- **JES (Job Entry Subsystem) für die Stapelverarbeitung**
- **TSO (Time Sharing Option) für die interaktive Verarbeitung (z.B. Programmentwicklung, Administration)**
- **Unix System Services, Posix compatibles Unix Subsystem**

Daneben existieren viele weitere Subsysteme, die Bestandteil des Betriebssystems sind, z.B. der Security Server, Communication Server, und viele andere

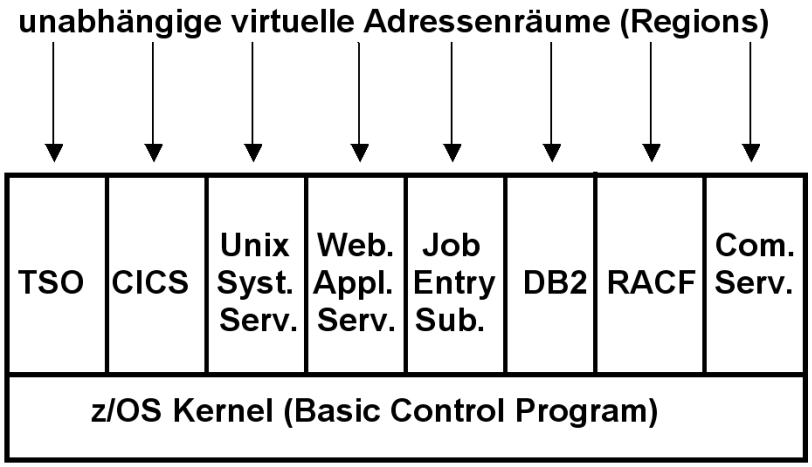


Abb. 3.1.3
z/OS Subsysteme in unterschiedlichen Regionen

Systemprozesse und Anwendungsprozesse laufen in getrennten virtuellen Adressenräumen. Der z/OS Kernel unterstützt eine Vielzahl von virtuellen Adressenräumen, die im z/OS Jargon als Regions bezeichnet werden.

Einige der Regions beherbergen Subsysteme, die Teil des Betriebssystems sind, aber im Benutzerstatus laufen. Gezeigt sind eine der wichtigsten Subsysteme, die wir uns im Einzelnen noch ansehen werden.

z/OS, wie auch Linux und Windows arbeitet mit dem Konzept eines Prozesses. Ausführbare Programme sind normalerweise in Programmbibliotheken auf einem Plattenspeicher abgespeichert. Ein Prozess ist der Aufruf und die Ausführung eines Programms.

z/OS, wie auch Linux und Windows arbeiten mit einer virtuellen Adressumsetzung (Address Translation). Hierbei existieren viele virtuelle Adressenräume, die mit der hexadezimalen Adresse Hex 00 ... 00 beginnen und eine einstellbare maximale Größe haben. z/OS bezeichnet diese virtuellen Adressenräume als (virtual) Address Spaces, Access Spaces oder auch als Regions. Die Begriffe sind austauschbar.

Moderne Rechner arbeiten multiprogrammiert. Auf einem Windows, Linux oder z/OS Rechner laufen in der Regel zahlreiche Prozesse parallel zueinander ab. Jeder der Prozesse läuft in einem eigenen virtuellen Adressenraum (Region, Address Space).

Manche der Prozesse sind Systemprozesse. Systemprozesse laufen als Teil des Betriebssystems. Wenn Sie unter Windows den Task Manager aufrufen (CTR+ALT+DEL), sehen Sie dort eine lange Liste von aktiven Systemprozessen in entsprechend vielen virtuellen Adressenräumen.

z/OS bezeichnet einige seiner Systemprozesse als Subsysteme. Subsysteme sind in sich abgeschlossene Softwareeinheiten. Manche Subsysteme wie JES, TSO, der Security Server oder der Communication Server sind ein Bestandteil des z/OS Betriebssystems. Andere Subsysteme wie die DB2 Datenbank, der WebSphere Application Server oder der CICS Transaktionsmanager sind optional, müssen extra installiert werden, und kosten zusätzliche Lizenzgebühren.

Im Gegensatz zu den Systemprozessen stehen die Benutzer- (Anwendungs-) Prozesse. Diese führen Programme (Anwendungen, Applications) aus, die vom Benutzer des z/OS Systems geschrieben wurden.

z/OS wurde Anfang 1966 unter dem Namen OS/360 als reines Stapelverarbeitungssystem eingeführt. Es wurde von Fred Brooks entwickelt, und diente als Vorlage für sein berühmtes Buch „The mythical Manmonth“ (http://en.wikipedia.org/wiki/The_Mythical_Man-Month).

Mit wachsendem Funktionsumfang wurde der Name immer wieder geändert: OS/360 → MFT→ MVT → MVS . 1996 erfolgte eine Namensänderung von MVS nach OS/390. 2000 wurde der Name z/OS im Zusammenhang mit der neuen 64 Bit Adressierung eingeführt.

An Stelle von z/OS wird der Name MVS heute noch häufig gebraucht, was nicht ganz korrekt ist.

Programme, die unter z/OS laufen, lassen sich gliedern in:

Subsysteme sind Programmprodukte wie Datenbanken und Transaktionsmonitore, die Laufzeitumgebungen für eigentliche Benutzerprogramme zur Verfügung stellen.

Benutzerprogramme können sein:

- **klassische z/OS (bzw. OS/390) Hintergrundprogramme (Batch Programs)**
- **Benutzeranwendungen, die unter der Kontrolle von Subsystemen wie TSO,CICS, IMS oder Websphere ablaufen, oder**
- **UNIX-Programme, welche die UNIX System Services unter z/OS ausnutzen.**

Systems Management Funktionen werden für die Steuerung und Überwachung des Ablaufes benötigt. Es gibt sehr viele solcher Funktionen, sowohl von IBM als auch von Drittanbietern wie z.B. Computer Associates (CA) oder BMC Software. .Sie dienen der Überwachung des Betriebssystems und, da sich das Betriebssystem in weiten Bereichen selbst steuert, zur Überwachung der Subsysteme und der Benutzeranwendungen.

3.1.4 Zwei Arten der Datenverarbeitung

Betriebswirtschaftliche Großrechner betreiben Datenverarbeitung auf zwei unterschiedliche Arten:

Bei der Interaktive Verarbeitung dient der Mainframe als Server in einer Client/Server Konfiguration. Zahlreiche Klienten (meistens PCs, aber auch Geldausgabeautomaten oder Registrierkassen im Supermarkt) nehmen Dienstleistungen des Servers in Anspruch. In Großunternehmen wie z.B. der Volkswagen AG können das mehrere 100 000 Klienten sein, die in der Fachsprache oft als Terminals bezeichnet werden.

Dies ist ein Beispiel für die interaktive Verarbeitung: Sie sitzen an Ihrem PC und haben einen Excel Prozess gestartet. Sie arbeiten mit einer großen Excel Tabelle und stoßen eine Berechnung an, die viele Sekunden oder Minuten dauert.

Während der Berechnung blockiert der EXEL Prozess, reagiert z.B. nicht auf Tastatur-Eingaben. Der Grund ist, die Bearbeitung wird als Teil des Excel Prozesses durchgeführt; während dieser Zeit kann der Prozess nichts anderes machen.

Der Gegensatz zur interaktiven Verarbeitung ist die Stapelverarbeitung (Batch Processing). Hier stößt ein Prozess, z.B. für länger laufende Verarbeitungsaufgaben, einen zweiten Prozess an.

Dies ist ein Beispiel für die Stapelverarbeitung: Sie sitzen an Ihrem PC und editieren ihre Masterarbeit mit Word for Windows. Dies ist ein interaktiver Prozess. Sie beschließen, die ganze Arbeit auf Ihrem Tintenstrahldrucker probeweise auszudrucken. Dies dauert viele Sekunden oder Minuten.

Während des Druckens blockiert der Word Prozess nicht. Sie können die Diplomarbeit weiter editieren.

Hierzu setzt Word for Windows neben dem Editierprozess einen getrennten Druckprozess auf, der als Stapelbearbeitungsprozess (batch job oder background Prozess) bezeichnet wird. Der Scheduler/Dispatcher des Betriebssystems stellt zeitscheibengesteuert beiden Prozessen CPU Zeit zur Verfügung.

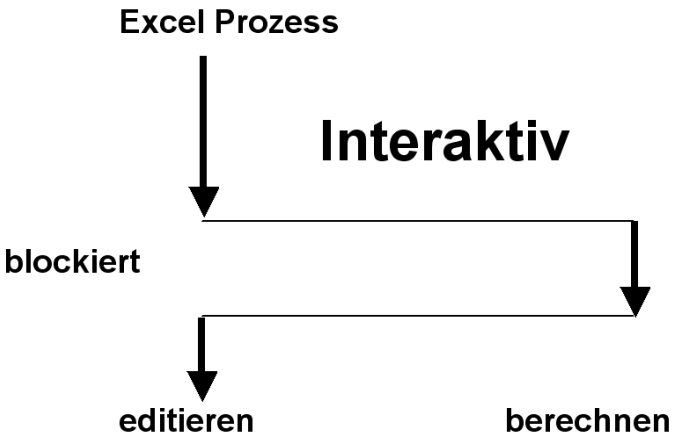


Abb. 3.1.4
Interaktive Verarbeitung

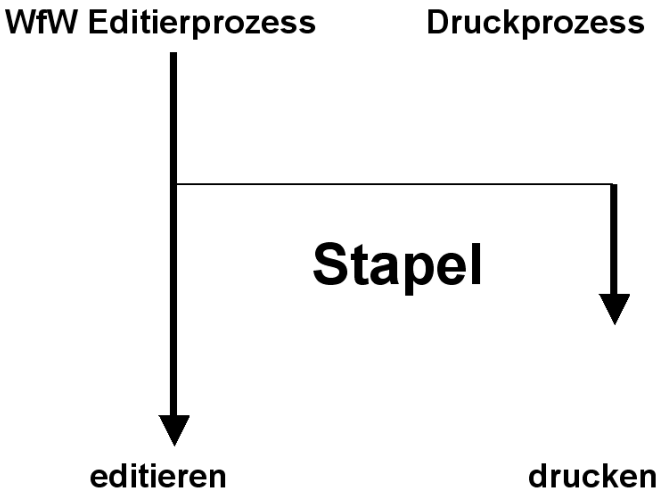


Abb. 3.1.5
Stapelverarbeitung

Abbildungen 3.1.4 und 3.1.5 zeigen diesen Zusammenhang im zeitlichen Ablauf.

Ein Mainframe kann ohne weiteres 100 000 Benutzer mit 100 000 angeschlossene Terminals bedienen. Hierzu wird für jeden Benutzer ein eigener interaktiver Prozess in einem eigenen virtuellen Adressenraum gestartet.

Parallel dazu laufen auf dem Mainframe zahlreiche Stapelverarbeitungsprozesse, die als Jobs bezeichnet werden.

Im Rechenzentrum der Credit Suisse, einer Schweizer Großbank in Zürich, waren es im Frühjahr 2010 durchschnittlich 86 000 Jobs pro Tag. Die Ausführungszeit der einzelnen Jobs konnte Sekunden, Minuten, Stunden, und in Extremfällen auch Tage betragen.

Starten, Überwachung des Ablaufs und Terminierung der einzelnen Jobs ist die Aufgabe eines z/OS Subsystems, des Job Schedulers.

3.1.5 Stapelverarbeitung unter Linux

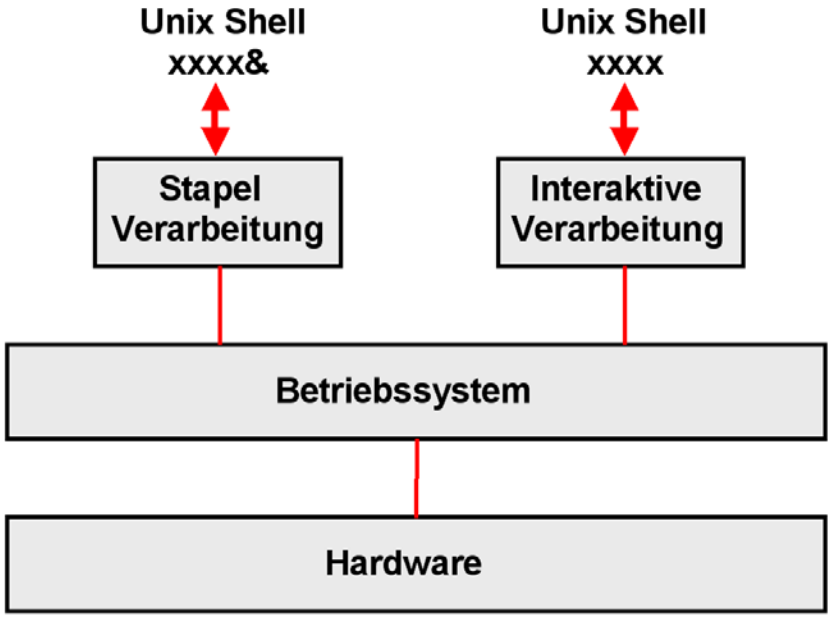


Abb. 3.1.6
Stapel- und interaktive Verarbeitung unter Linux

Unter Linux kann die Stapelverarbeitung (Batch Processing) als Sonderfall der interaktiven Verarbeitung betrieben werden. In der Shellsprache werden Batch-Aufträge durch ein nachgestelltes „&“ gekennzeichnet.

Der cron-Daemon ist eine Jobsteuerung von Unix bzw. Unix-artigen Betriebssystemen wie Linux, BSD oder Mac OS X, die wiederkehrende Aufgaben (cronjobs) automatisch zu einer bestimmten Zeit ausführen kann. cron ist die Software, die dem z/OS Batch Processing Subsystem (Job Entry Subsystem, JES) am nächsten kommt. Allerdings geht der JES Funktionsumfang weit über den von cron hinaus.

Cron startet Skripte und Programme zu vorgegebenen Zeiten. Der auszuführende Befehl wird in einer Tabelle, der sog. crontab, gespeichert. Jeder Benutzer des Systems darf eine solche crontab anlegen.

Diese Tabelle besteht aus sechs Spalten; Die ersten fünf dienen der Zeitangabe (Minute, Stunde, Tag, Monat, Wochentage), die letzte enthält den Befehl. Die einzelnen Spalten werden durch Leerzeichen oder Tabulatoren getrennt.

Häufig führt der cron-Daemon wichtige Programme für die Instandhaltung des Systems aus, wie zum Beispiel Dienste für das regelmäßige Archivieren und Löschen von Logdateien.

Beim Hochfahren eines Linux Systems wird als Erstes cron gestartet. cron wiederum startet einen Shell Prozess. Ähnlich wird beim Hochfahren eines z/OS Systems JES als erstes gestartet. Das Hochfahren selbst bewirkt ein weiterer Prozess, der Master Scheduler.

Griechisch chronos (χρόνος) bedeutet Zeit.

3.1.6 Beispiel eines DOS Jobs

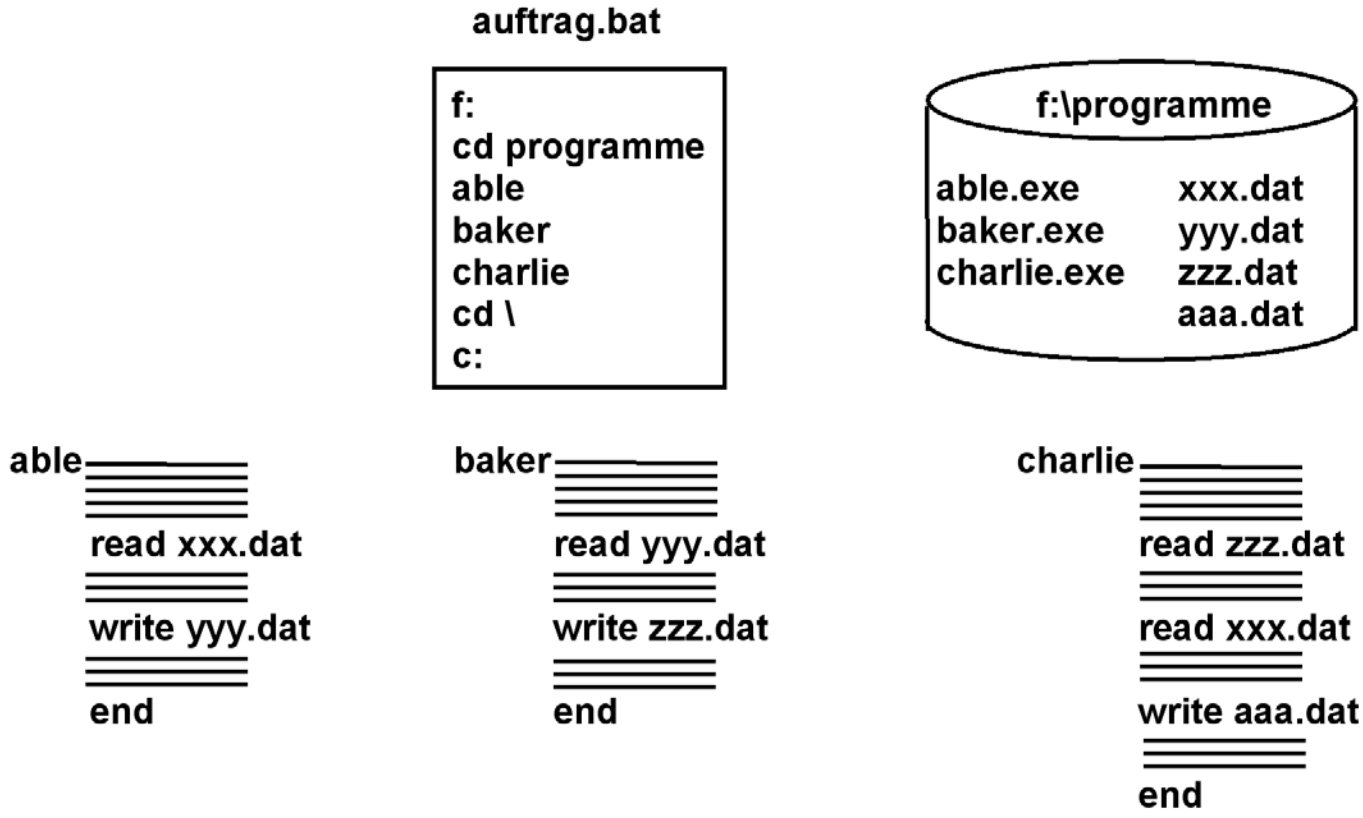


Abb. 3.1.7
Eine .bat File für die Ausführung eines einfachen Jobs

Als Beispiel sei angenommen, dass Sie unter Benutzung Ihrer Windows DOS Eingabeaufforderung drei Programme able, baker und charlie der Reihe nach ausführen wollen. Um dies zu automatisieren können Sie ein .bat Programm mit dem Namen auftrag.bat schreiben. auftrag.bat ist ein Job Script, welches die sequentielle Ausführung der drei Programme automatisch steuert. Es ist in der .bat Language geschrieben.

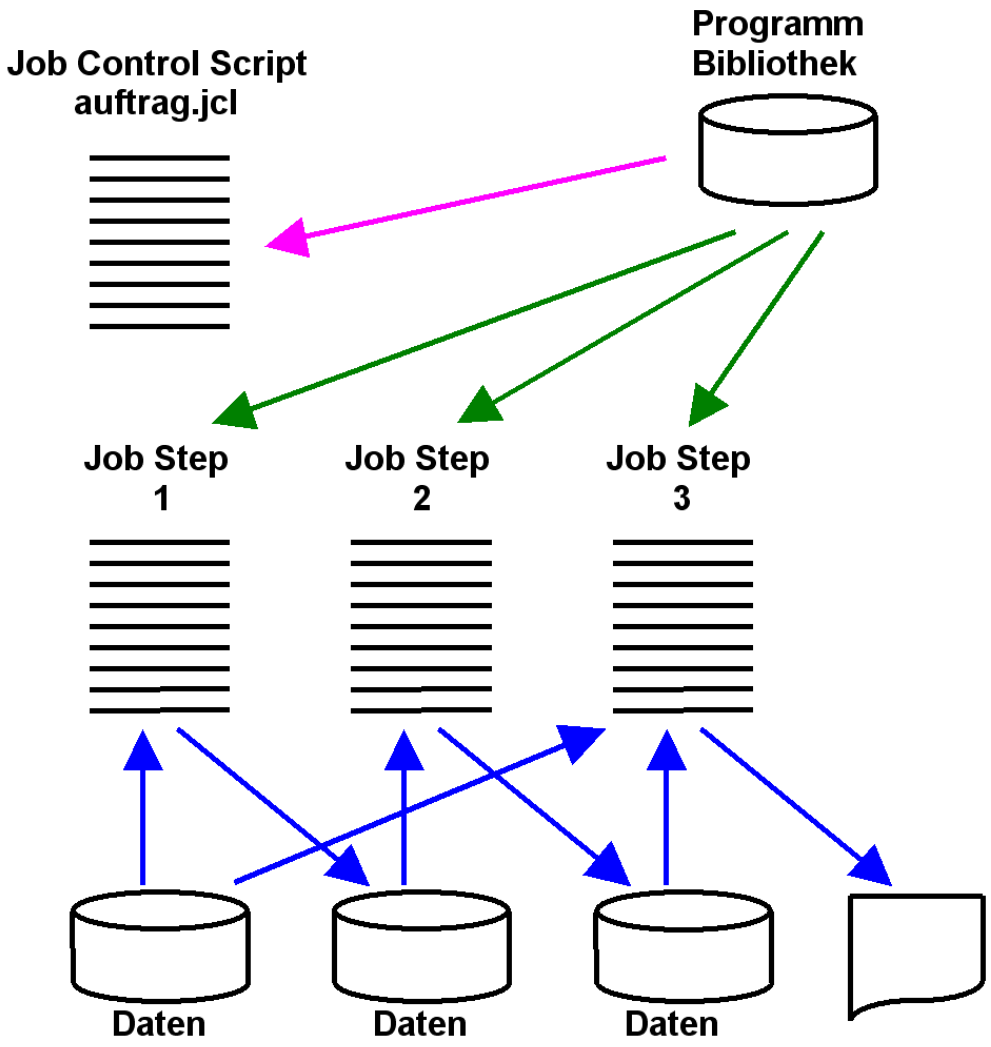


Abb. 3.1.8
Job Ausführung

Die Ausführung von auftrag.bat besteht aus drei Schritten, den Job Steps, die jeweils die Ausführung der drei Programme able, baker und charlie bewirken.

Unter z/OS verwendet man für die Job Steuerung eine eigene Scriptsprachen , die Job Control Language, JCL.

Da das JCL Programm die verwendeten Dateien angibt, ist ein „late Binding“ der verwendeten Dateien an die auszuführenden Programme möglich.

Eine „Cataloged Procedure“ ist ein JCL Programm, welches vom Benutzer für eine spätere Verwendung zwischengespeichert wird (.z.B. in einer vom Benutzer erstellten Library JCLLIB) und bei Bedarf mittels eines JCL Befehls aufgerufen wird.

3.1.7 Job Scheduler

Die Eigenschaften eines Stapelverarbeitungsprozesses sind häufig:

- **Lange Laufzeit**
- **wird häufig periodisch zu festgelegten Zeiten ausgeführt**
- **Hohes Datenvolumen. Es kann aus Tausenden oder Millionen von Datenbankelementen (z.B. Reihen in einer SQL Tabelle) bestehen**
- **Für die Daten möglicherweise komplexe Verarbeitungsanforderungen**
- **Der Verarbeitungsprozess benötigt möglicherweise große Datenmengen von einem anderen Rechner. Diese werden als ein große Datei zu einer bestimmten Zeit angeliefert.**
- **Der Stapelverarbeitungsprozess läuft asynchron zu irgendwelchen Benutzer-Aktionen. Er ist nicht Teil einer Benutzer Session in einem Online (Client/Server) System.**
- **Wird typischerweise nicht von einem Benutzer eines Online Systems gestartet, Normalerweise startet ein Online Benutzer keinen Stapelverarbeitungsprozesses, und wartet auch nicht auf eine Antwort.**

  (Eine Ausnahme sind die ersten Übungsaufgaben unseres Enterprise Computing Praktikums).

Ein reguläres Windows oder Linux Betriebssystem enthält nur primitive Job Scheduling Funktionen wie z.B. cron. Zum Füllen dieser Marktlücke stellen eine ganze Reihe von Software Unternehmen Job Scheduler für Apple, Windows, Unix und Linux Plattformen her. Beispiele sind:

- **cosbatch der Firma OSM**
  http://www.cosbatch.com/
- **Dollar Universe der Firma Orsyp,**
  http://www.orsyp.com/en.html
- **OpenPBS (Portable Batch System),**
  http://www.openpbs.org,
  open source package ursprünglich von der NASA entwickelt
- **PBS Pro der Firma PBS Grid Works,**
  http://www.pbsgridworks.com,
  enhanced commercial version von OpenPBS.
- **Global Event Control Server der Firma Vinzant Software,**
  http://www.vinzantsoftware.com/
- **ActiveBatch der Firma Advanced–System Concepts,**
  http://www.advsyscon.com/

All diese Produkte erreichen bei weitem nicht den Funktionsumfang von z/OS JES

3.1.8 Script Sprachen

Sie kennen vermutlich bereits eine der gängigen Scriptsprachen wie Pearl, PHP, Java Script, Tcl/Tk, Phyton, Ruby usw.

Sie haben vielleicht auch schon einen der erbitterten Religionskriege miterlebt, welche Scriptsprache die beste ist, und welche grottenschlecht ist. Glauben Sie mir, es gibt keine „beste“ Scriptsprache.

Eine weitere Scriptsprache ist REXX (*Restructured Extended Executor*). REXX wird vor allem auf Mainframes eingesetzt. REXX Interpreter sind aber auch für Windows, Apple, Linux, alle Unix Dialekte usw. verfügbar, und weiter verbreitet als allgemein angenommen. REXX wird vor allem von Leuten benutzt, die schon auf dem Mainframe damit gearbeitet haben.

Hier ist ein kurzes Programm in REXX

```
/* A short program to greet you                                    */
/* First display a prompt                                          */

say ‘Please type your name and then press ENTER:‘
parse pull answer                  /* Get the reply into answer */

/* If nothing was typed, then use a fixed greeting          */
/* otherwise echo the name politely                          */

if answer=‘‘   then say ‘ Hello Stranger! ‘
                    else say ‘ Hello ‘ answer ‘!‘
```

Abb. 3.1.9
Hello World in REXX

Auffallend ist, dass an Stelle des Schlüsselwortes “write” das Schlüsselwort “say” benutzt wird. Daran können sie immer erkennen, ob ein Script Programm in REXX geschrieben ist..

Neben REXX ist auch php unter z/OS verfügbar. Siehe
www.cedix.de/VorlesMirror/Band1/PHP01.pdf

Einige Script Sprachen sind sog. shell Scripte. Dazu gehören .bat Files unter Windows, Unix und Linux Shell Scripts und JCL unter z/OS. Shell Scripts sind aus Sequenzen von shell Kommandos entstanden. Auch die make file, die Sie als C++ Programmierer für das Übersetzen Ihres C++ Programms benutzen , benutzt eine eigenes Shell Script.

Shell Scripte sind in der Regel für den Anfänger unübersichtlicher und schwieriger zu verstehen als normale Script Sprachen wie Tcl/Tk, REXX oder PHP. Das gilt besonders auch für JCL.

Die JCL Syntax macht einen absolut archaischen Eindruck, und ist für den erstmaligen Benutzer ein Kulturschock. Das wird auch für Sie gelten, wenn sie Ihr erstes JCL Script schreiben, also „be prepared". JCL hat sicher viel dazu beigetragen, Mainframes zu dem Ruf „obsolete Technology" zu verhelfen.

Tatsache ist, nachdem Neulinge ihren ersten Schock überwunden haben (das ist nach wenigen hundert Zeilen JCL Code der Fall), beginnen sie JCL heiß und innig zu lieben. Es hat viele Versuche gegeben, JCL durch eine „moderne" Scriptsprache zu ersetzen – bisher mit wenig Erfolg. Tatsache ist, JCL ist relativ leicht erlernbar, sehr mächtig und sehr produktiv.

JCL wurde zusammen mit OS/360 (der ersten Version des heutigen z/OS) entwickelt und hat Ähnlichkeiten mit Unix Shell Scripts; ein Beispiel ist die Ähnlichkeit der Unix dd und des JCL DD Kommandos.

Es ist unmöglich, JCL zu verstehen, ohne sich mit der historischen Entwicklung auseinander zu setzen. Diese geht auf die frühere Benutzung von Lochkarten zurück. Lochkarten waren bis in die 70er Jahre beliebte und kostengünstige Elemente für die Speicherung von Daten.

3.1.9 Lochkarte

Die IBM Lochkarte, etwa 19 x 8 cm groß, wurde 1928 eingeführt. Vorläufer mit einem etwas anderen Format wurden von Herrmann Hollerith erfunden und 1890 bei der Volkszählung in den USA und wenig später auch in Deutschland eingesetzt.

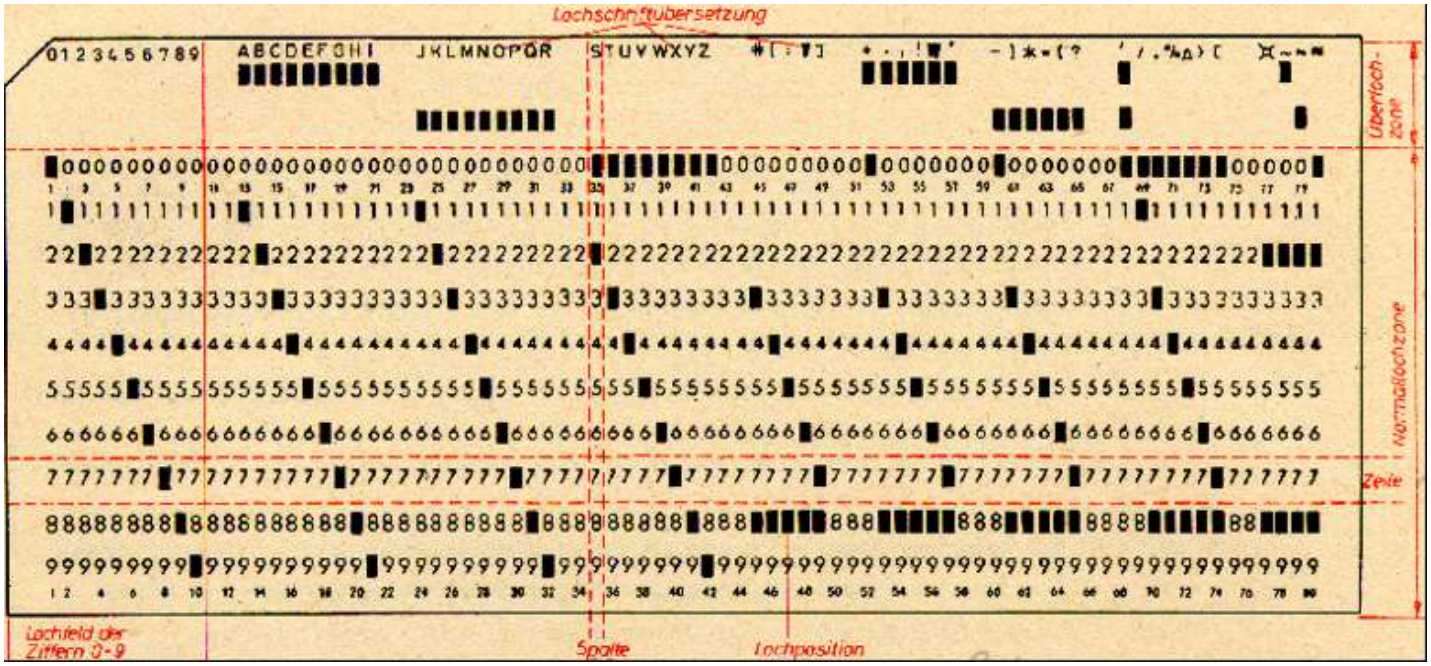


Abb. 3.1.10
Maximal 3 Lochungen in jeder Spalte

Mit maximal 3 Lochungen pro Spalte wurde das BCD Alphabet dargestellt. Die IBM Lochkarte besteht aus 80 Spalten. In jede Spalte konnte an einer von 12 Stellen jeweils ein Loch gestanzt werden. Die Position konnte in einem Lochkartenleser abgefühlt werden. Damit war es möglich, mittels der sog. BCD Kodierung (binary coded decimal) in jeder Lochkarte 80 alphanumerische Zeichen zu speichern.

Die Lochkarte ist die älteste Form eines maschinell lesbaren Speicher- und Steuerungsmediums. Die ersten durch Lochkarten gesteuerten Systeme fanden Verwendung in Musikautomaten, Drehorgeln und Webstühlen. Weitere Verwendungen der Lochkartentechnik fanden in vielen anderen mechanischen und elektromechanischen Rechen- und Sortiermaschinen statt, wie z. B. der US-Amerikanischen Volkszählung von 1890.

Eine Standard-Lochkarte (Hollerith-Lochkarte) aus den 30'er Jahren bestand aus dünnem Karton, hatte ein Format von 187,3 mm x 82,5 mm und war in 12 Zeilen zu je 80 Spalten eingeteilt. Die Informationen wurden vollautomatisch oder manuell mit einem Kartenlocher in Form von kleinen Rechtecken in die Karte eingestanzt. Das Prinzip der Datenspeicherung auf einer Lochkarte ist die Kodierung durch das Vorhandensein oder das Fehlen eines Loches an einer vorgegebenen Position. Die Lochkarten waren am linken oberen Rand abgeschnitten, um die Eindeutigkeit bei der stapelweisen "Fütterung" von optoelektronisch arbeitenden Lochkartenlesern zu gewährleisten.

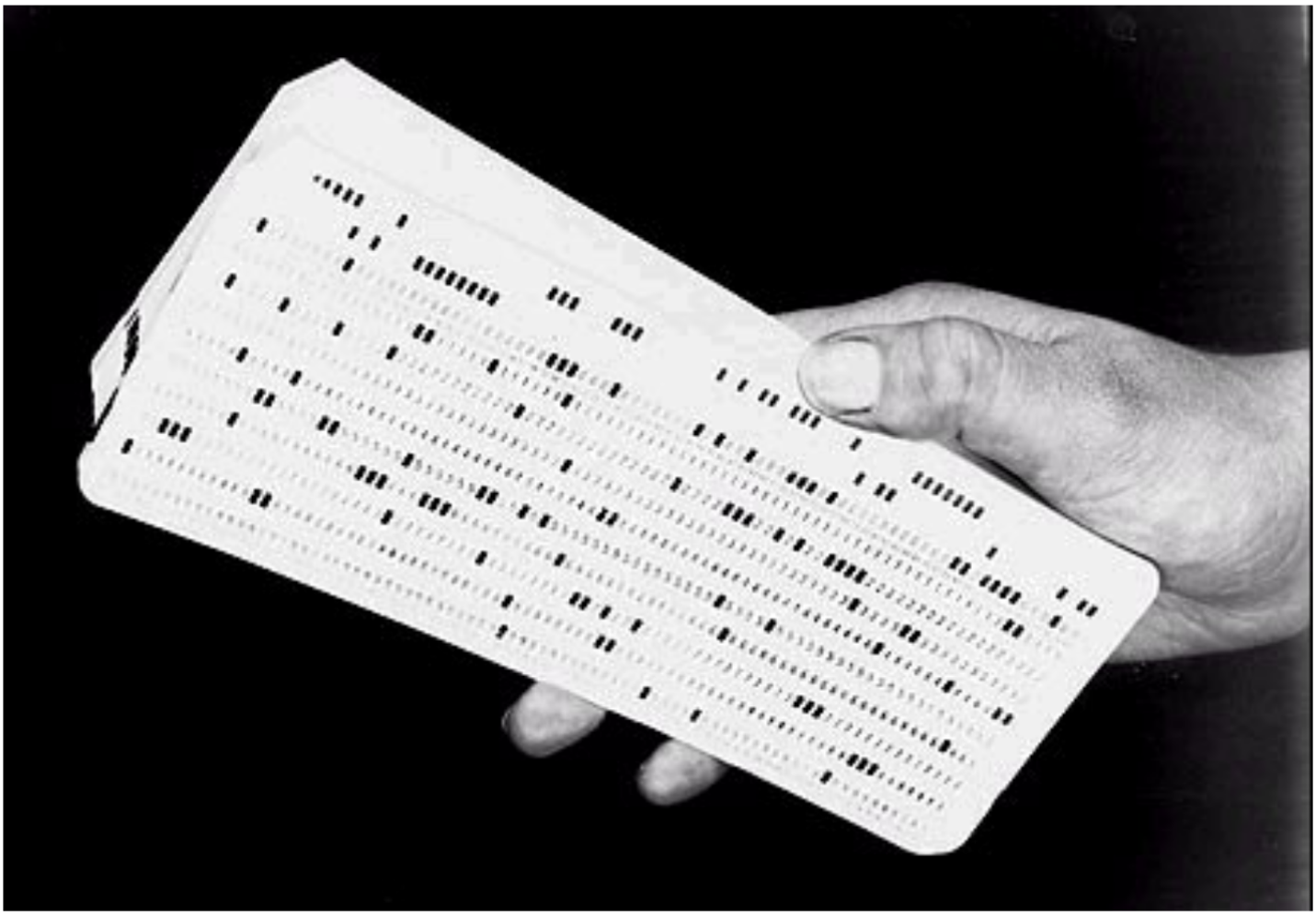


Abb. 3.1.11
Lochkartenstapel

Ein Programm, bestehend aus 500 Zeilen Code, wurde in den Rechner eingegeben als ein Stapel (hier gezeigt) bestehend aus 500 Lochkarten. Ein derartiger Stapel wurde als Lochkartendeck bezeichnet.

Ein Programm wurde dadurch erzeugt, indem man auf einem IBM 024 Locher, einem Gerät mit einer Schreibmaschinenähnlichen Tastatur, jeweils eine Zeile des Programms eingab, worauf der IBM 024 Locher mit dieser Information eine Lochkarte erstellte.

http://www-03.ibm.com/systems/z/os/zvse/about/history1960s.html

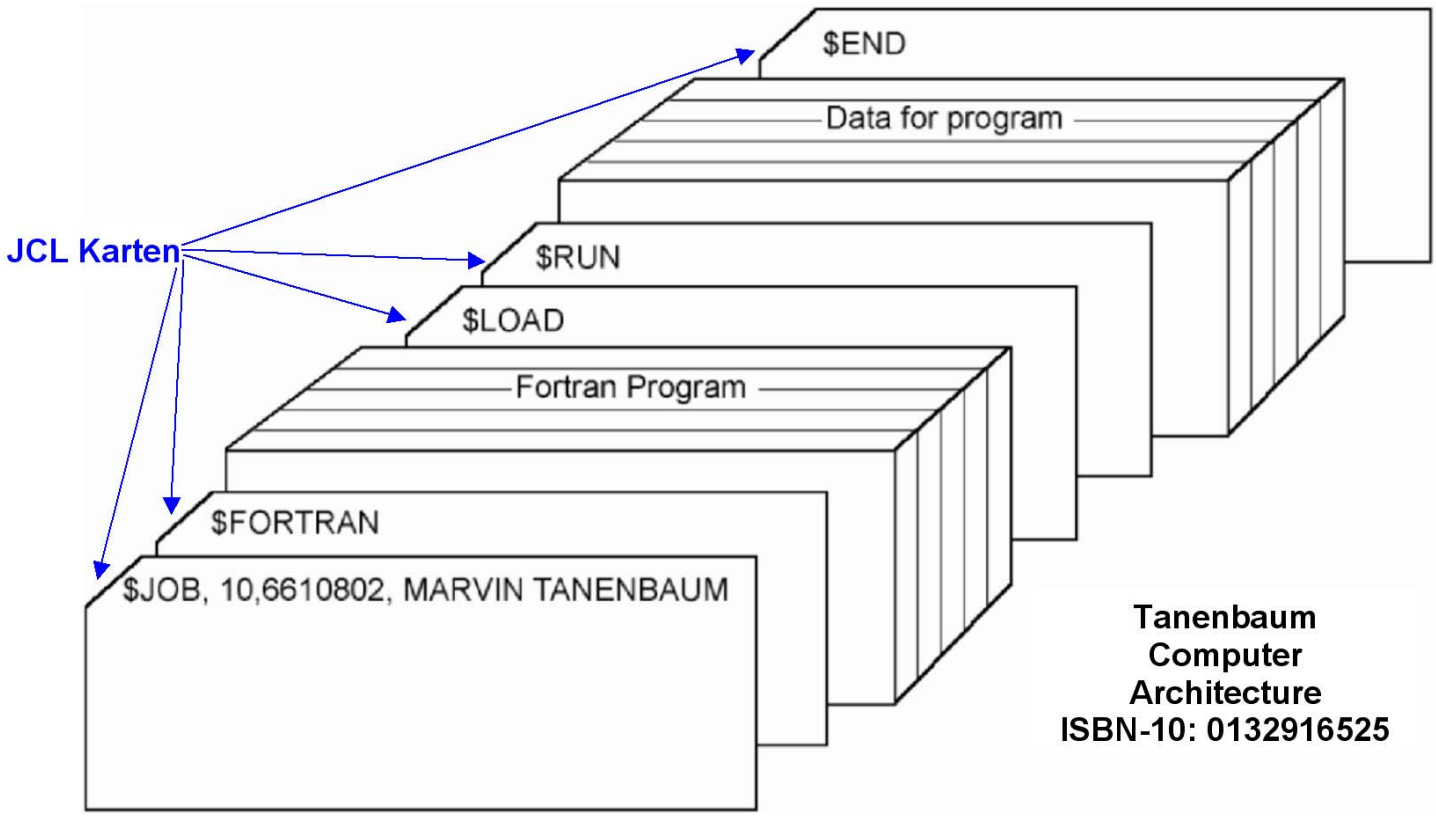


Abb. 3.1.12
Ein Programm und seine Daten als Lochkartendeck

Gezeigt ist die Implementierung eines FORTRAN Programms etwa Anno 1960. Es besteht aus einer Reihe von JCL Karten, ein JCL Statement pro Karte. Zwischen den JCL Karten befindet sich das FORTRAN Programm Karten Deck, jeweils eine Karte pro Fortran Statement, sowie ein weiteres Karten Deck mit den von dem vom Fortran Programm zu verarbeitenden Daten. Das gesamte Kartendeck wurde von dem Lochkartenleser des Rechners in den Hauptspeicher eingelesen und verarbeitet. Die Ausgabe erfolgte typischerweise mittels eines angeschlossenen Druckers.

Später wurde es üblich, das Fortran Programm und die Daten auf Magnetband (und noch später auf einem Plattenspeicher) zu speichern, und das Programmdeck und das Datendeck durch zwei Library Call Lochkarten zu ersetzen. Noch später speicherte man dann das JCL Script ebenfalls in einer Library auf dem Plattenspeicher.

3.1.10 JCL Format

Ein JCL Script besteht aus einzelnen JCL Statements. Ein JCL Statement passte früher auf eine Lochkarte mit 80 Spalten, und daran hat sich bis heute nichts geändert.

Wie auch bei anderen Programmiersprachen besteht ein JCL Statement aus mehreren (genau fünf) Feldern.

Abb. 3.1.13
Format eines JCL Statements

Um die Logik zum Parsen eines Statements zu vereinfachen, führte man einige Konventionen ein:

- **Jedes Feld beginnt immer an einer bestimmten Spalte der Lochkarte. Das vereinfachte damals das Parsen des Statements. Diese Spaltenabhängigkeit existiert auch heute noch !!! Wenn Sie dagegen verstoßen, gibt es eine Fehlermeldung. Warnung: Dies ist mit großem Abstand der häufigste Programmierfehler, den Anfänger beim Schreiben eines Mainframe Programms machen.**
- **Jedes JCL Statement beginnt mit den beiden Zeichen / / (forward slashes) . Damit war es möglich, die JCL Statement leicht von den Lochkarten des Fortran Decks und des Datendecks zu unterscheiden. Die beiden Slashes befinden sich in Spalte 1 und 2 .**
- **Das Label Feld beginnt in Spalte 3 , die maximale Länge ist 8 .**
- **Das Operation folgt dem Label Feld, getrennt durch ein Leerzeichen, in Spalte 12 .**
- **Das Parameter Feld (Operand Feld) folgt dem Operation Feld, getrennt durch ein (oder mehr) Leerzeichen**
- **Alles was hinter dem Operand Feld folgt, ist ein Kommentar. Zwischen Operand und Kommentar muss sich (mindestens) ein Leerzeichen befinden.**

z/OSMVS JCL Reference SA22-7597-10 Eleventh Edition, April 2006 , chapter 3,
see also http://www.cedix.de/Literature/Textbooks/jcl/CodeJCL.pdf.

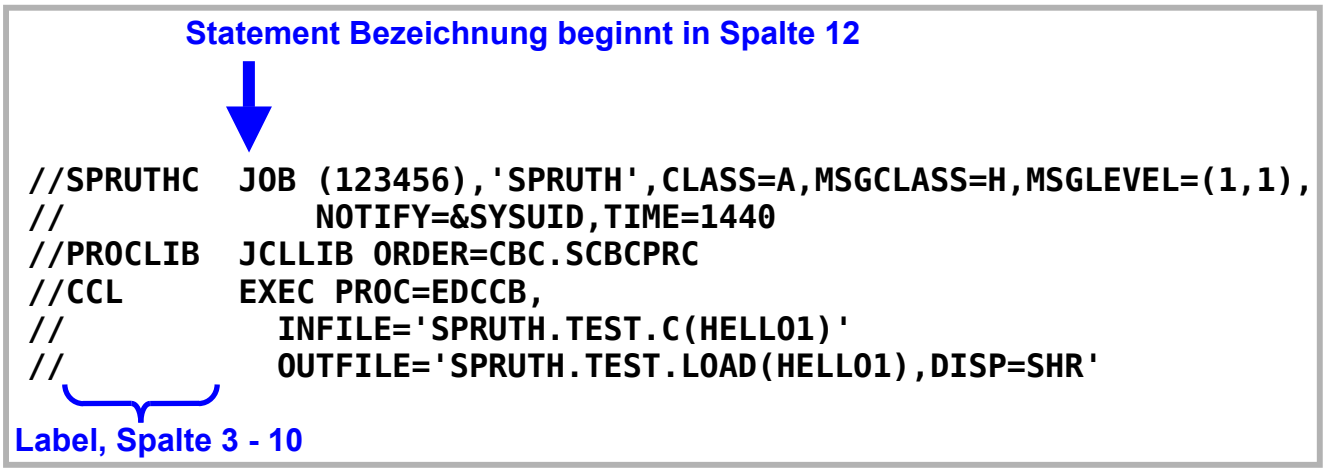


Abb. 3.1.14
Ein einfaches JCL Script

JOB Statement markiert den Anfang eines Jobs

EXEC Statement bezeichnet Prozedur, die ausgeführt werden soll

PROC Statement gibt die Adresse einer Procedure an

DD Statement bezeichnet die zu benutzenden Dateien (hier nicht verwendet)

Achten Sie darauf, dass Ihr JCL Script die richtigen Spalten benutzt !!!

Beispiel für einen JCL Befehl:

// DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)

RECFM, FB, LRECL und BLKSIZE sind Schlüsselwörter der JCL Sprache.

Der Befehl besagt, dass die hiermit angesprochene Datei (bzw. deren Data Control Block, DCB) ein Fixed Block (FB) Record Format (RECFM) hat (alle Datensätze haben die gleiche Länge), dessen Länge (Logical Record Length LRECL) 80 Bytes beträgt, und dass für die Übertragung vom/zum Hauptspeicher jeweils 5 Datensätze zu einem Block von (Blocksize BLKSIZE) 400 Bytes zusammengefasst werden.

Literatur
M.Winkler: „MVS/ESA JCL“. Oldenbourgh, 3. Auflage, 1999

3.1.11 Submit Command

TSO bzw. ISPF sind Command Line Shells, vergleichbar mit den korn, bash Shells unter Unix/Linux bzw. der DOS Eingabeaufforderung unter Windows. Solche Shells beinhalten eine primitive Ausführungsumgebung, die es ermöglicht, von der Kommandozeile aus ein Programm (z.B. xyz.exe) .direkt ausführen. Dies ist auch unter der TSO Shell möglich.

Professioneller ist die Programmausführung unter einem Anwendungsserver. Viele Anwendungsserver sind Bestandteil einer Client/Server Umgebung (sog. interaktive Server). Ein Beispiele hierfür ist z.B. der Web Application Server Ihrer persönlichen Home Page, auf dem eine CGI oder Java Servlet Anwendung läuft.

Der wichtigste Befehl in dem folgenden JCL Script ist „EXEC IGYWCL“. Er bewirkt, dass ein weiteres JCL Script mit dem Namen IGYWCL aus einer Programmbibliothek aufgerufen wird. IGYWCL enthält Anweisungen, die bewirken, dass der in der folgenden Zeile angegebene Data Set TEST.COB.(COB02) compiled und linked wird.

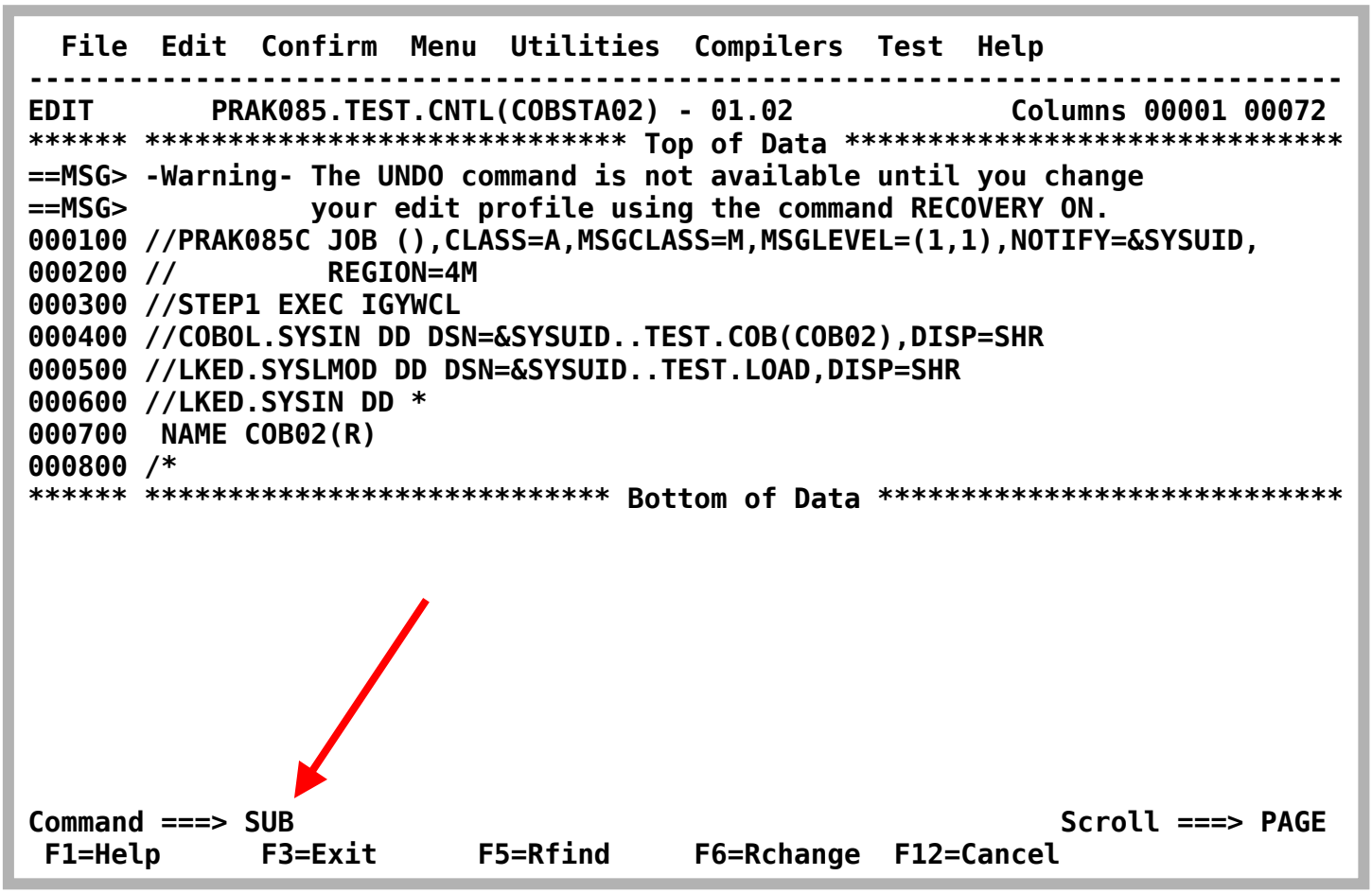


Abb. 3.1.15
Eingabe des Submit Commands

Andere Anwendungsserver implementieren Stapelverarbeitung, und das JES Subsystem unter z/OS ist eine führende Implementierung. Ein JES Job wird typischerweise durch die Ausführung eines JCL Scripts gestartet, und genau dies tun Sie bei der Durchführung ihres ersten Cobol Tutorials, bei dem Sie das JCL Script in Abb. 3.1.14 erzeugen.

Das Kommando „SUB“ (Abkürzung für Submit) bewirkt, dass das JCL Script zwecks Ausführung an das JES Subsystem übergeben wird. JES generiert hierfür einen Job mit einer eindeutigen Job Nummer, und übergibt ihn zwecks Ausführung an einen JES „Initiator“ (siehe Abschnitt 3.2).

3.1.12 Bedarf an JCL Talent

A February 2011 study conducted by SHARE entitled, "CLOSING THE IT SKILLS GAP: 2011 SHARE Survey for Guiding University & College IT Agendas", found that half of the companies surveyed hire new IT employees straight out of school, with relatively little actual work experience. The study also indicates a strong demand for mainframe skills with the finding, "In terms of platform-specific skills, companies seek applicants skilled in running two types of environments—database administration and mainframe administration. Specific mainframe administration skill areas also are in demand by a majority, or close to a majority, of companies in the survey — 55% seek mainframe administrative skills, and half are in need of skills involving JCL, or Job Control Language."

Abb. 3.1.16
Bedarf an JCL Talent

So archaisch JCL auch sein mag, es ist nach wie vor sehr populär. Nach Absolvierung des Einarbeitungs-Kulturschocks finden die Benutzer JCL sehr produktiv und mächtig.

http://www.mainframezone.com/blog/the-it-skills-shortage-where-will-we-find-mainframe-talent

3.2 Job Entry Subsystem

3.2.1 z/OS Stapelverarbeitung

In großen Unternehmen ist die Stapelverarbeitung (neben der interaktiven Verarbeitung) eine wichtige Aufgabe. Ein Job Entry-Subsystem (JES) hat die Aufgabe, Stapelverarbeitungsvorgänge zu automatisieren. Diese Funktion ist in den meisten Windows- und Unix-Betriebssystemen nur sehr rudimentär vorhanden. Job Control Subsysteme werden für Windows- und Unix-Betriebssysteme von unabhängigen Herstellern angeboten. Ein Beispiel dafür bildet COSbatch (www.cosbatch.com/) der Firma Open Systems Management Inc. (OSM).

Das z/OS Betriebssystem stellt eine spezielle Systemkomponente zur Verfügung, die für die Steuerung und Ablaufkontrolle aller Jobs (einschließlich aller TSO Sitzungen) zuständig ist. Diese Systemkomponente hat den Namen Job Entry Subsystem (JES). Aus historischen Gründen existieren davon zwei Varianten: JES2 und JES3, die sich in ihrer Funktion jedoch sehr ähnlich sind.

Beide Varianten haben die gleiche Hauptaufgabe, und sind in der Lage, die Jobverarbeitung auf einem einzelnen Rechner zu steuern. Der wesentliche Unterschied zwischen JES2 und JES3 liegt in der Steuerung einer System-Konfiguration, die aus mehreren Rechnern besteht. Ein einzelner Rechner wird in der Regel mit JES2 gesteuert.

Werden mehrere Mainframe Rechner in einem (als Sysplex bezeichneten) Verbund (Cluster) betrieben, erfolgt die Steuerung häufig durch JES3. Mehrere Rechner arbeiten gemeinsam, und jeder Job wird von JES3 je nach Auslastung einem der Rechner zur Verarbeitung zugewiesen.

Enterprise Batch Processing erfordert Sophistication und eine umfangreiche Funktionalität, die auf Windows, Unix und Linux Servern nicht verfügbar ist:

- **Zeit-abhängige Ausführung**
- **Maintaining relationships**
- **Adminstrative und Controlling Funktionen**
- **Einrichtungen für die Steuerung von System Resourcen**
- **Effektive Multiprogrammierung und Multiprocessing**
- **Accounting Einrichtungen**
- **Umgebung für die Stapelverarbeitung (Batch Execution)**

Können Sie sich vorstellen, in einem Unix oder Windows System mehr als 10 Batch Jobs gleichzeitig auszuführen und zu steuern ?

z/OS kann in einem (als Parallel Sysplex bezeichneten) Cluster Zehntausende von Jobs gleichzeitig ausführen, verwalten und steuern .

Die Übergabe eines neuen Jobs an das Betriebssystem, das Queuing von Jobs, die Prioritäts- und Ablaufsteuerung zahlreicher Jobs untereinander, Startzeit und Wiederholfrequenz, Durchsatzoptimierung sowie die Zuordnung oder Sperrung von Resourcen ist die Aufgabe eines Job Steuerungssystems.

3.2.2 Aufgaben der z/OS Stapelverarbeitung

z/OS Jobs können eine Verarbeitungsdauer von Sekundenbruchteilen, Minuten, Stunden und evtl. Tagen haben.

Ein Job wird submitted, es erfolgt eine lange Serie von Berechnungen, unterbrochen durch zahlreiche Ein/Ausgabe (I/O) Anforderungen, Ergebnisse der Berechnungen werden in eine Datei geschrieben, und der Job wird beendet (terminated) mit einem Return Code (RC), der die erfolgreiche oder nicht erfolgreiche Beendigung der Verarbeitung anzeigt.

Beispiele für z/OS Jobs sind:

- **Complex Data Base Queries**
- **Cheque and account processing, e.g. debits, credits, loans, credit rating**
- **Data Backups, Compression, Conversion**
- **Offline ATM (Automatic Teller Machine) processing**
- **System and Database maintenance**
- **Data replication and synchronization**
- **Processing exchanged B2B (Business to Business) data**
- **Tape processing**
- **Printing**
- **File system maintenance, healthchecks, compression…**
- **Configuration Jobs (Admin, RACF)**

3.2.3 Job Ausführung

Die Übergabe (Submission) eines Stapelverarbeitungs-Jobs an das Betriebssystem erfolgt durch einen Benutzer; bei einem Großrechner auch durch einen spezifisch hierfür abgestellten Bediener (Operator). Unter z/OS wird die Job Submission mit Hilfe des Job Entry Subsystems (JES) automatisiert.

Voraussetzung für die Ausführung eines Jobs ist, dass benutzte Programme und Dateien (Data Sets, Files) bereits existieren.

Die Erstellung und Eingabe (Submission) eines Jobs erfordert die folgenden Schritte:

1. **Zuordnung einer Datei, welche das Job Control Programm (JCL Script) enthalten soll.**

2. **Editieren und Abspeichern der JCL Datei**

3. **Submission (JES übernimmt die Ausführung des Jobs).**

Viele Jobs werden in einem Unternehmen immer wieder ausgeführt. Ein Beispiel ist die monatliche Lohn- und Gehaltsabrechnung und Geldüberweisung für alle Mitarbeiter eines Unternehmens. Hier wird jeden Monat das gleiche JCL Script ausgeführt. Eine „Cataloged Procedure“ ist ein JCL Programm, welches vom Benutzer für eine spätere Verwendung zwischengespeichert wird (z.B. in einer vom Benutzer erstellten Library JCLLIB) und bei Bedarf mittels eines JCL Befehls aufgerufen wird.

Das JCL Programm gibt die verwendeten Dateien an. In dem Beispiel der Gehaltsabrechnung sind die sich ändernden Gehaltslisten zusätzlich zu sich nur selten ändernden Dateien wie Personaldaten. In dem JCL Script werden die verwendeten Dateien mittels von DD Statements angegeben. Damit ist ein „late Binding“ der verwendeten Dateien an die auszuführenden Programme möglich.

3.2.4 Job Scheduler

JES arbeitet nach dem Master/Agent-Konzept. Der Master (entweder JES2 oderJES3)

- **initiiert die Jobs,**
- **ordnet zu Anfang der Bearbeitung die Eingabe und Ausgabe Ressourcen zu**
- **reiht sie je nach Priorität in Warteschlangen ein**
- **übernimmt während der Verarbeitung die Zuordnung von Ressourcen wie CPU´s und Hauptspeicher und**
- **kontrolliert den Ablauf der Jobs.**
- **Nach Abschluss der Bearbeitung übernimmt JES die Freigabe der Ressourcen. Sie werden damit für andere Jobs verfügbar.**

Die Agenten werden unter z/OS als Initiator bezeichnet. Dies sind selbständige Prozesse, die getrennt von JES in eigenen virtuellen Adressräumen (Regions) laufen. Sie

- **starten die Anwendungsprogramme des Jobs und**
- **senden Statusinformationen an die JES Steuerungskomponente, und/oder**
- **fertigen Laufzeitberichte an.**

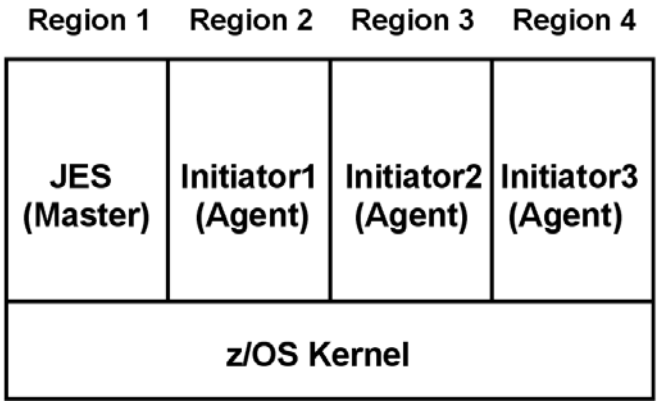


Abb. 3.2.1
Initiator

Zur Verbesserung der Auslastung unterhält ein z/OS System typischerweise mehrere Initiators. Ein Initiator unterhält eine Input Warteschlange, an die JES einen neuen Job übergibt. Der Initiator fertigt die Jobs in seiner Input Warteschlange unter Prioritätsgesichtspunkten ab. Jeder Initiator verarbeitet in jedem Augenblick nur einen einzigen Job, aber mehrere Initiatoren können mehrere Jobs gleichzeitig verarbeiten.

Jeder Initiator belegt einen eigenen virtuellen Adressenraum, in dem er selbst und ein von ihm gerade bearbeiteter Job untergebracht ist. Das z/OS Stapelverarbeitungs-Subsystem belegt somit einen Reihe von Adressräumen, einen für JES, und weitere für mehrere Initiators.

Wenn ein Job abgeschlossen wird, wird der Initiator des Adressraums aktiv und JES sendet den nächsten Job, der darauf wartet, verarbeitet zu werden.

JES2 kann bis zu 999 Initiators in 999 Address Spaces verwalten.

3.2.5 Verarbeitungsschritte einer Job Verarbeitung

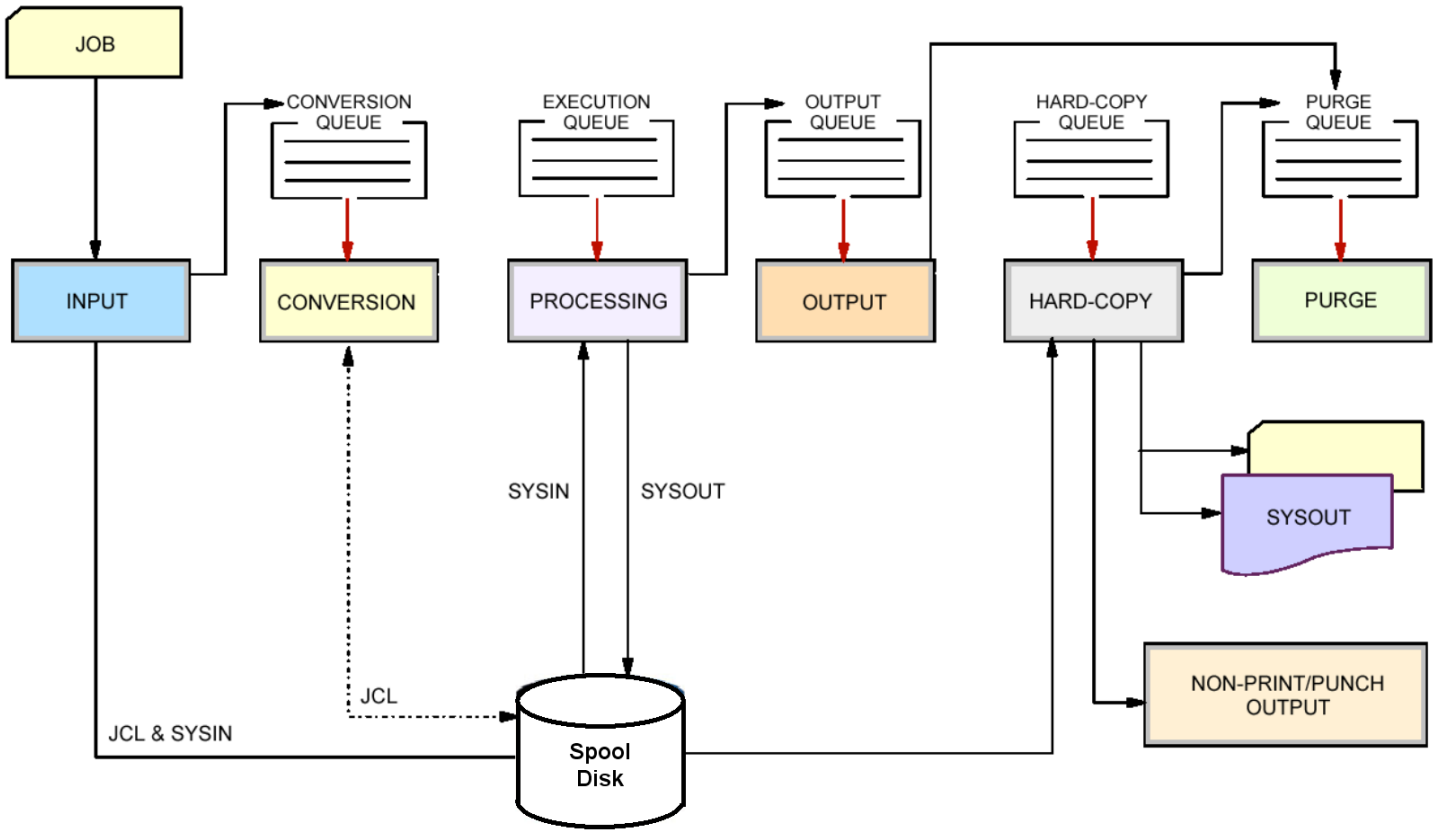


Abb. 3.2.2
Verarbeitungsablauf eines Batch Jobs

Zur Speicherung aller Informationen über einen Job (z.B. JCL Records oder Ausgabe-Daten) wird ein dem JES gehörender Dataset, der sogenannte SPOOL Dataset, benutzt in dem folgende Daten gespeichert werden;

- **Original JCL und ihre Interpretation durch JES,**
- **Uhrzeit, wann der Job die einzelnen Phasen oder Steps durchlaufen hat,**
- **Return-Codes für die einzelnen Steps,**
- **Benutzung aller Systemkomponenten, z.B. Rechenzeit, Speicherplatz, Zahl der Zugriffe auf Platten und Bänder.**

Die Ausführung eines Jobs durch einen Initiator erfolgt in 5 Schritten: Input, Conversion, Execution, Output und Purge. Spool Files sind temporäre Dateien, die nach Abschluss der Verarbeitung des Jobs nicht mehr gebraucht werden. Sie wurden traditionell auf Magnetband geschrieben; heute benutzt man hier häufig einen Festplattenspeicher.

INPUT

Ein Job wird über den Reader (Internal Reader) eingelesen und auf dem Spool Datenträger abgelegt. Er gelangt in die JES Input Queue und erhält eine Jobnummer.

CONVERSION

Die JCL wird im Falle von Prozeduren ergänzt und auf Gültigkeit überprüft. Es wird ein so genannter Internal Text erstellt, der als Kontrollblockstruktur die Batch Verarbeitung gegenüber dem JES repräsentiert.

EXECUTION

Nach einem bestimmten Algorithmus und auf der Basis von Prioritäten wird ein Job zur Ausführung ausgewählt. Die Kontrollblöcke des ausgewählten Jobs werden dazu einem freien Initiator übergeben.

OUTPUT

Die vom Programm erstellten Listen werden in der JES Output Queue auf dem Spool Datenträger gehalten, bis ein freier (heute meist virtueller) Drucker bereit ist, die Listen auszudrucken, oder bis der User/Operator den Output löscht (cancel).

PURGE

Erst wenn der gesamte Output auf einen Drucker ausgegeben wurde, werden die von einem Batchjob belegten Bereiche auf dem Spool Datenträger gelöscht.

Ein Job Scheduler steuert die zeitliche Ablauffolge einer größeren Anzahl von Jobs. Er dient als single point of control für die Definition und das Monitoring der Background Ausführung für einen einzelnen oder eine Gruppe (Cluster) von z/OS Mainframes.

3.2.6 Batch Processing Beispiel

Eine Stapelverarbeitung erfolgt häufig in mehreren Schritten (Job Steps). Der Initiator steuert die Ablauffolge der einzelnen Job Steps.

Beispiel: Monatliche Kreditabrechnung für 300 000 Kundenkonten in einer Bank. Diese könnte z.B. aus den folgenden Schritten (Job Steps) bestehen:

1. **Darlehnskonto abrechnen, Saldo um Tilgungsrate verändern**
2. **Tilgung und Zinsen im laufenden Konto (Kontokorrent) auf der Sollseite buchen**
3. **Globales Limit überprüfen**
4. **Bilanzpositionen (Konten)**
5. **G+V Positionen (Gewinn- und Verlust Konten)**
6. **Zinsabgrenzung monatlich für jährliche Zinszahlung**
7. **Bankmeldewesen (ein Kunde nimmt je 90 000.- DM bei 10 Banken auf, läuft am Stichtag)**

3.2.7 Eine unübersichtliche aber lehrreiche Darstellung der Subsysteme

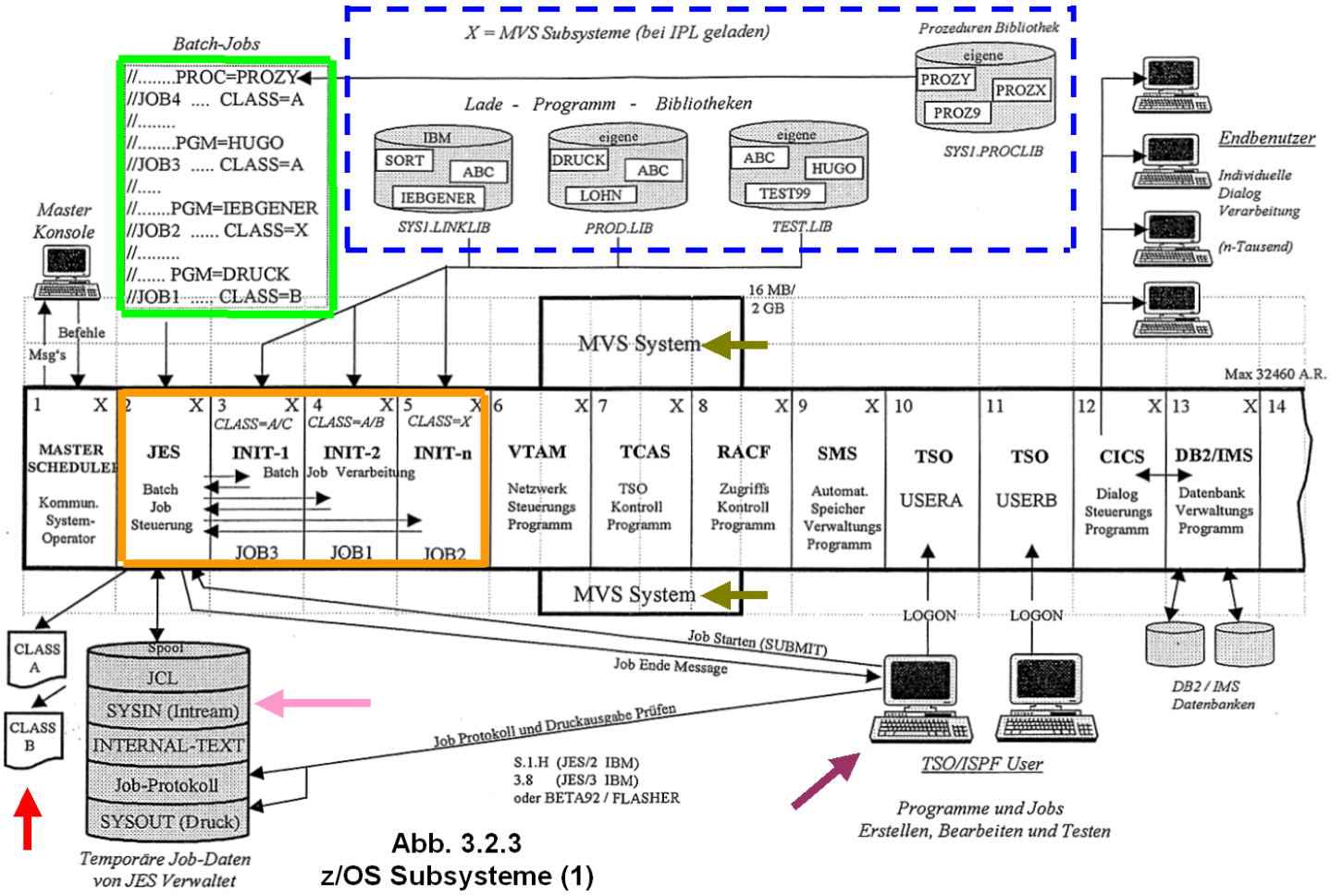


Abb. 3.2.3
z/OS Subsysteme (1)

Die Abb.3.2.3 zeigt als Beispiel ein z/OS System mit einer größeren Anzahl von Subsystemen, die jeweils in eigenen virtuellen Adressräumen (Regions) laufen.

An Stelle der korrekten Bezeichnung z/OS wird der noch immer geläufige Begriff MVS benutzt.

Das Stapelverarbeitungssystem besteht aus einer JES Region und drei Initiator Regions.

JES steuert derzeitig 4 Batch Jobs mit den Namen PROZY, HUGO, IEBGENER und Druck.

Die 4 JOBs werden entsprechend ihrer Prioritätsklassifizierung A, B und X von unterschiedlichen Initiators ausgeführt.

Das Programm PROZY ist in einer System-Library SYS1.PROCLIB untergebracht. IEBGENER befindet sich ebenfalls in einer Systemlibrary SYS1.LINKLIB. HUGO befindet sich in einer vom Benutzer angelegten Library. TEST.LIB. DRUCK schließlich befindet sich in einer ebenfalls vom Benutzer angelegten Library PROD.LIB.

Eine Spool File enthält temporäre Daten, die nach Abschluss der Verarbeitung wieder gelöscht werden können.

Ein Interaktiver Benutzer kann mittels seines TSO Terminals (siehe unten) einen neuen Job starten.

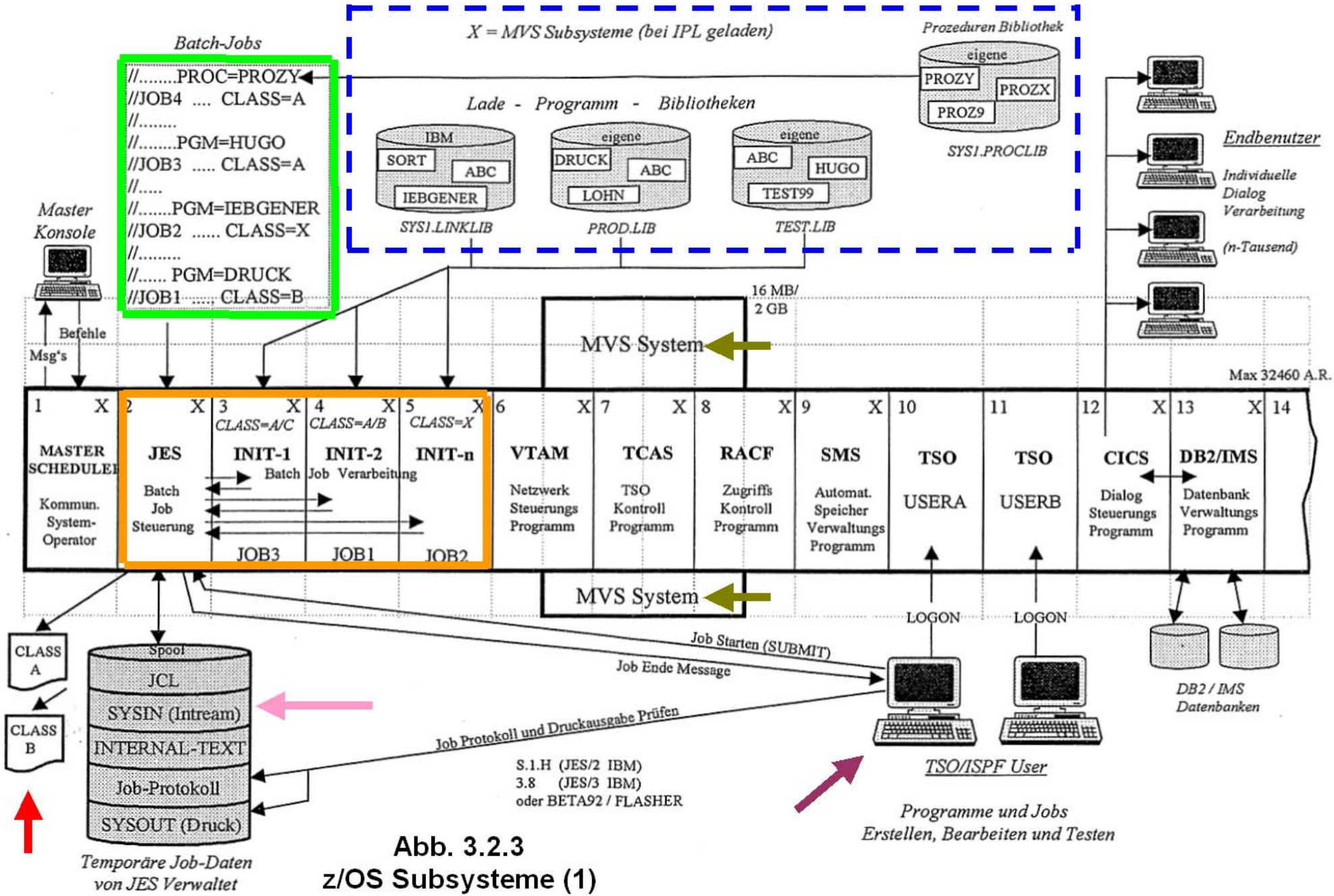


Abb. 3.2.3
z/OS Subsysteme (1)

3.2.8 Master Scheduler

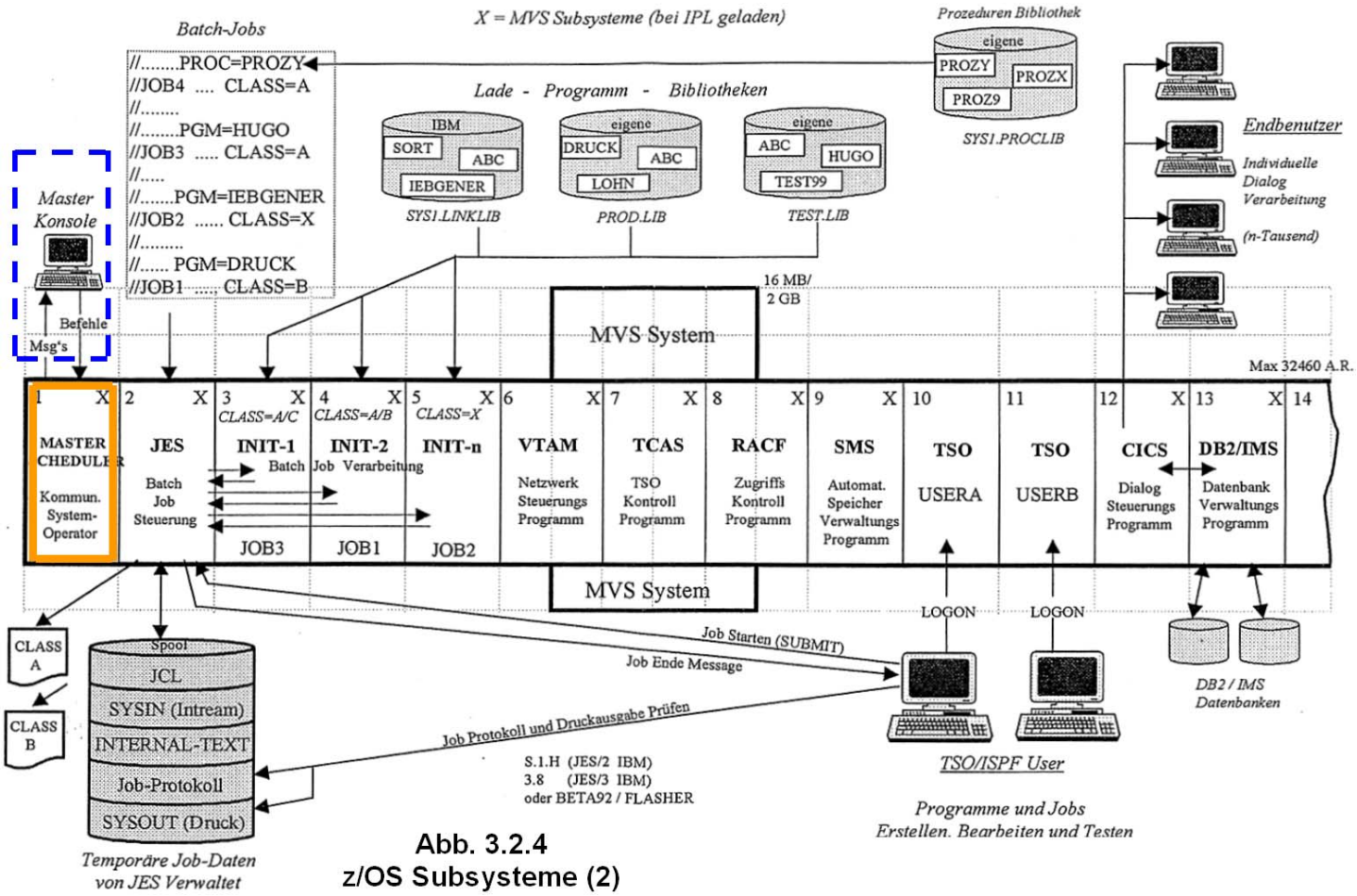


Abb. 3.2.4
z/OS Subsysteme (2)

Wenn Sie beim Booten Ihres PC eine spezielle Taste drücken, kommen Sie in das BIOS Menu. Hier können Sie Kommandos absetzen, um z. B. ein Booten von einer CD zu ermöglichen.

Unter z/OS ist diese Funktionalität nicht nur beim Hochfahren des Rechners, sondern auch während des laufenden Betriebs in der Form der „Master Konsole“ vorhanden. Im Vergleich zum BIOS sind dabei wesentlich erweiterte Funktionen vorhanden.

- **Mit Hilfe der z/OS Master Konsole kann ein System Administrator (Operator) den System Zustand beobachten, und bei Bedarf in das laufende System eingreifen.**
- **Die Steuerung der Master Console übernimmt eine z/OS Komponente der Master Scheduler. Dieser ist auch beim Hochfahren des Betriebssystems aktiv. Der Master Scheduler etabliert die Communication zwischen dem Betriebssystem und dem Job Entry Subsystem.**

Neben der z/OS Master Konsole existieren noch weitere Konsolen, z.B. eine Hardware Konsole nur für die Steuerung der Hardware. In modernen z/OS Versionen können diese Funktionen auf einer einzigen physischen Konsole vereinheitlicht werden.

3.2.9 Master Console

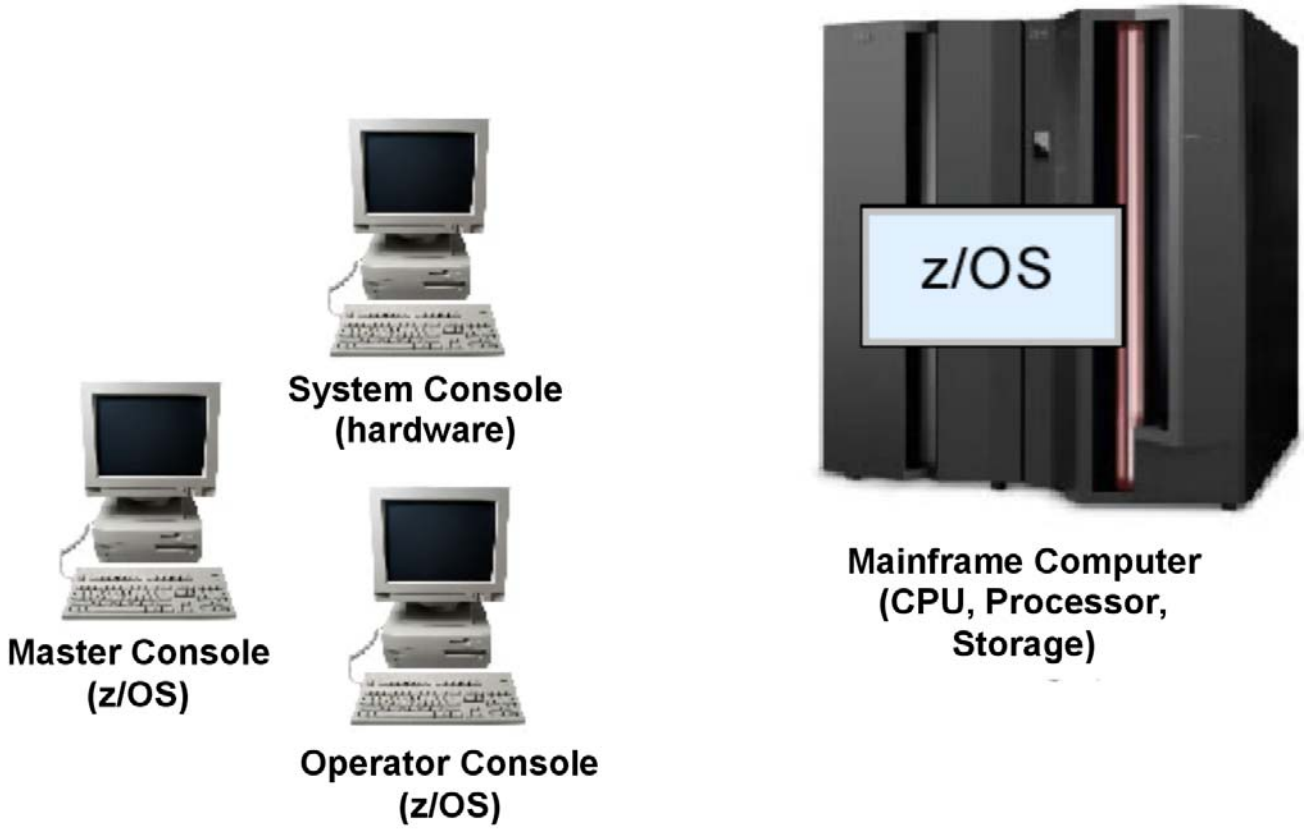


Abb. 3.2.5
Ein Mainframe System unterhält mehrere Konsolen

Die System Console (Teilfunktion der Hardware Management Console, HMC, Abschnitt 1.3.11) steuert die Hardware des Rechners. Für die Steuerung des z/OS Betriebssystems sind die Master Console und die Operator Console vorgesehen.

Wie oft starten Sie das Betriebssystem für Ihren PC oder Notebook? Wahrscheinlich mindestens einmal am Tag, vielleicht öfter, wenn Sie Upgrades oder Sicherheits-Patches installieren möchten. Das Starten von z/OS ist ein ähnlicher Prozess, geschieht aber weit weniger häufig, und geschieht nur mit sehr sorgfältiger vorheriger Planung.

z/OS Initialisierung, als Initial Program Load (IPL) bezeichnet, lädt eine Kopie des Betriebssystems von der Festplatte in den Hauptspeicher, und führt es aus. Dieses Verfahren besteht im wesentlichen aus:

- **System und Storage Initialisierung, einschließlich der Schaffung von Systemkomponente Adressräumen (Regions) für Systemkomponenten**
- **Master-Scheduler Initialisierung und Subsystem Initialisierung**

z/OS Systeme wurden entwickelt, kontinuierlich zu laufen. Viele Monaten können vergehen, ehe das Betriebssystem neu gestartet wird. Änderungen sind der häufigste Grund, das Betriebssystem neu zu starten, zum Beispiel:

- **Ein Testsystem kann täglich oder noch öfter IPLed werden.**
- **Ein Hochverfügbarkeits-Banking-System darf nur einmal im Jahr neu geladen werden, oder auch weniger häufig, um die Software zu aktualisieren.**
- **Äußere Einflüsse können oft die Ursache für IPLs sein, z.B. die Notwendigkeit, Stromversorgungs-Systeme in dem Maschinen Raum zu testen und zu pflegen.**
- **Manchmal verbraucht fehlerhafte Software Systemressourcen, die nur durch eine IPL aufgefüllt werden können. Diese Art von Verhalten ist in der Regel Gegenstand einer Untersuchung und Korrektur.**

Viele der Änderungen, die in der Vergangenheit ein IPL benötigten, können heute dynamisch erfolgen.

Das Herunterfahren von z/OS passiert so selten wie ein IPL.

3.3 TSO und Data Sets

3.3.1 Server Zugriff

Wir unterscheiden zwischen Arbeitsplatzrechnern (Klienten) und Servern. Arbeitsplatzrechner arbeiten im Single-User-Modus und können mit einem Single-User-Betriebssystem betrieben werden. Windows und CMS sind typische Single-User-Betriebssysteme. Windows fehlt die Benutzerverwaltung für den simultanen Multi-User-Betrieb. Unix und z/OS sind traditionelle Multi-User-Betriebssysteme.

Linux und Unix werden sowohl als Arbeitsplatzrechner- als auch als Server-Betriebssysteme eingesetzt. Windows Server 2008 R2 ist ein Server Betriebssystem, welches teilweise die gleichen Komponenten wie die Windows 7 Produktfamilie benutzt. z/OS ist ein reinrassiges Server-Betriebssystem. Andere Beispiele für Server-Betriebssysteme sind i/5 (OS/400), Tandem Non-Stop und DEC OpenMVS. Unix Betriebssysteme wie Solaris, HP-UX (unter Itanium) sowie AIX werden bevorzugt als Server Betriebssysteme eingesetzt

Bei einem Server-Betriebssystem unterscheiden wir zwischen einem interaktiven zeitscheibengesteuerten und einem Run-to-Completion Betrieb. Wenn mehrere Benutzer sich gleichzeitig in einen Unix-Server einloggen, geschieht dies typischerweise im interaktiven, Zeitscheiben-gesteuerten Modus. SQL Aufrufe und Transaktionen arbeiten in vielen Fällen im Run-to-Completion Modus. Run-to-Completion bedeutet den Wegfall der Zeitscheibensteuerung, was unter Performance Gesichtspunkten vorteilhaft sein kann. Hierbei ist es die Aufgabe des Entwicklers, seine Programme so zu schreiben, dass nicht ein bestimmtes Programm alle Resourcen eines Rechners usurpieren kann.

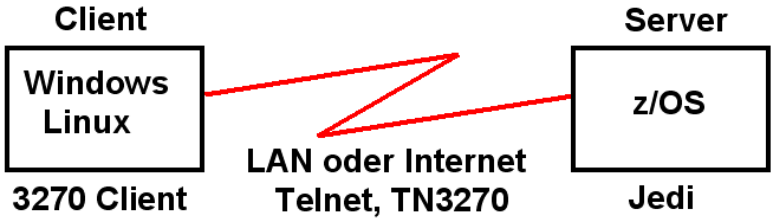


Abb. 3.3.1
Zeilenorientierter 3270 Client

Ein Server Zugriff erfordert spezielle Client Software. Hierfür existieren drei Alternativen:

1. **Selbstgeschriebene Anwendungen:**

   Sockets, RPC, Corba, DCOM, RMI

2. **Zeilenorientierte Klienten:**

Unix Server	**Telnet, Putty Client, FTP**
z/OS Server	**3270 Client (3270 Emulator)**
VMS Server	**VT 100 Client**

3. Klienten mit graphischer Oberfläche:

Windows Server	Citrix Client
WWW Server	Browser Client
SAP R/3 Server	SAPGUI Client
Unix Server	XWindows, Motif
z/OS und OS/390 Server	Servlet, JSP Client

Der 3270 Client, meistens als 3270 Emulator bezeichnet, ist der einfachste Klient für den Zugriff auf einen Mainframe Server. Unter http://jedi.informatik.uni-leipzig.de/de/access.html können Sie einen kostenlosen 3270 Emulator herunterladen.

3.3.2 Time Sharing Option

Time Sharing Option (TSO) ist eine z/OS zeilenorientierte Shell, vergleichbar mit der Windows „DOS Eingabeaufforderung“ oder den Unix/Linux Bourne, Korn, Bash oder C-Shells.

TSO wird hauptsächlich für angewendet für:

- Software Entwicklung und Test,
- System Administration.

Systemadministratoren (Systemprogrammierer) verwenden TSO um Files zu editieren, Steuerungen vorzunehmen, Systemparameter zu setzen und einen Job Status zu überprüfen.

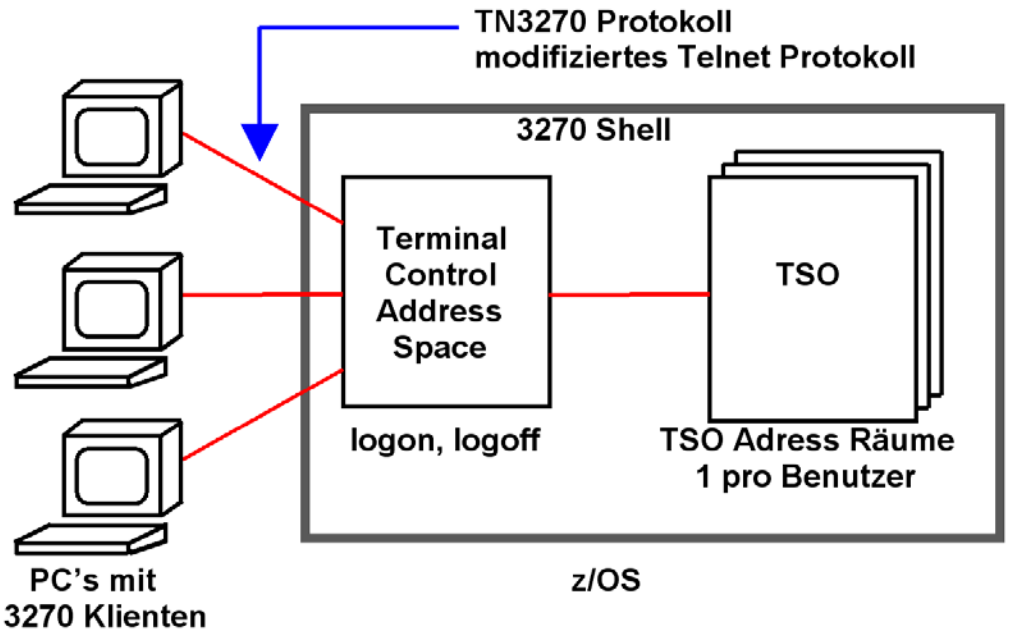


Abb. 3.3.2
TSO Adressenräume

Es ist eine „Full Screen“ und eine Command Level Schnittstelle verfügbar. Die Full Screen Komponente wird als ISPF (Interactive System Productivity Facility) bezeichnet.

ISPF ist eine wichtige Erweiterung / Ergänzung für die einfache TSO Shell. Hierbei wird auf dem Klienten ein Bildschirm wiedergegeben, der neben der Kommando Zeile eine Auflistung der möglichen Kommandos enthält. IBM nennt dies „Full Screen Mode“ und bezeichnet derartige Bildschirminhalte als Panels.

Daneben enthält ISPF einen Bildschirm Editor, den ISPF Editor. Der Editor lässt sich mit dem Unix/Linux vi Editor vergleichen. Beide Editoren sind hoffnungslos inkompatibel, kryptisch, schwer erlernbar, sehr mächtig, und werden von ihren Benutzern heiß und innig geliebt.

Der TSO Terminal Control Prozess (Terminal Control Access Space, TCAS) arbeitet in einem eigenen Adressenraum. Er nimmt Nachrichten von den einzelnen TSO Klienten entgegen und leitet sie an den für den Benutzer eingerichteten separaten TSO Adressenraum weiter.

Für jeden neuen TSO Benutzer wird beim login von TCAS ein eigener (virtueller) Adressenraum eingerichtet.

3.3.3 Eine unübersichtliche aber lehrreiche Darstellung der Subsysteme

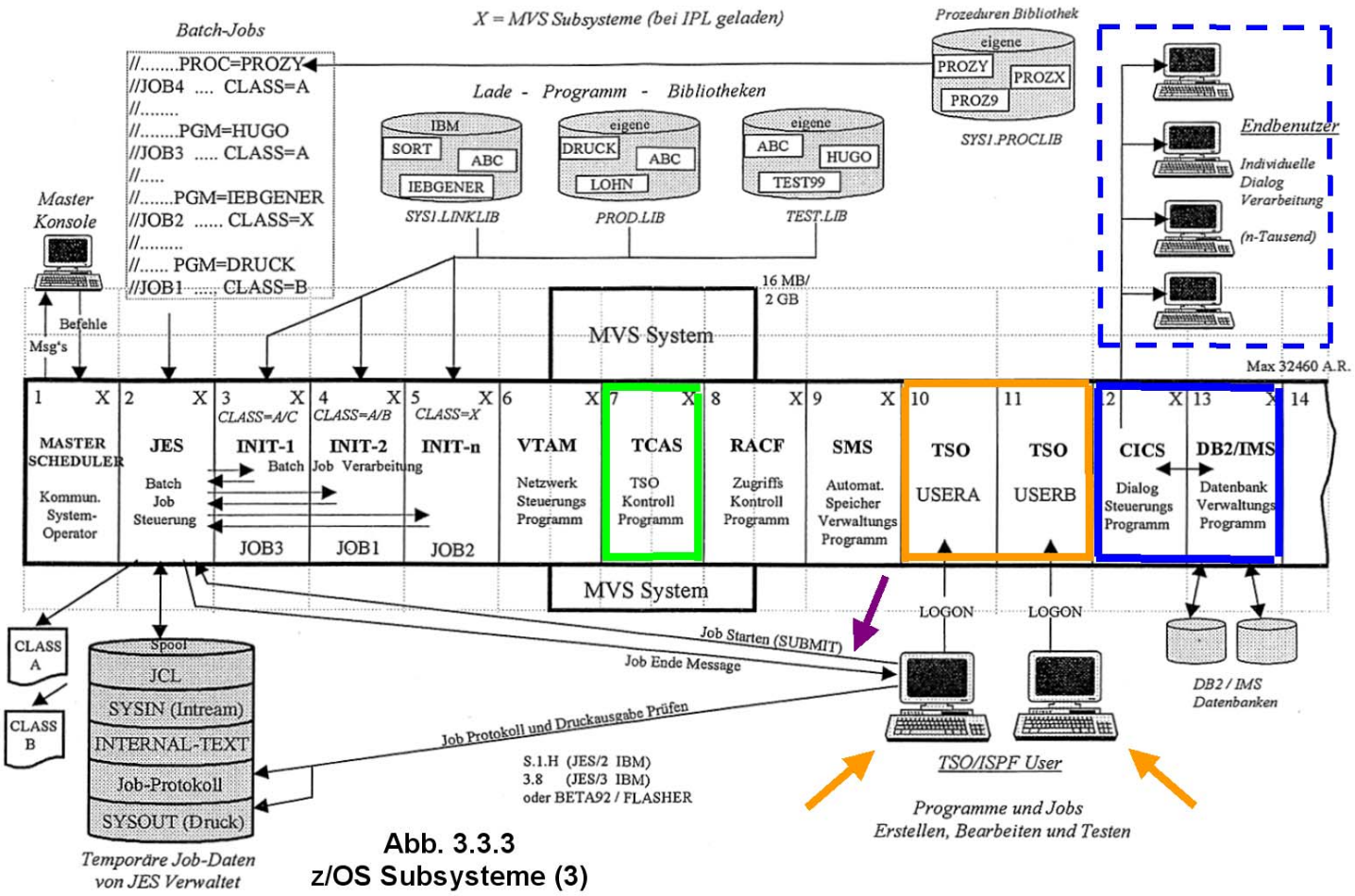


Abb. 3.3.3
z/OS Subsysteme (3)

**Abb. 3.3.3
z/OS Subsysteme (3)**

Die obige Abbildung zeigt ein z/OS System mit einer größeren Anzahl von Subsystemen, die jeweils in eigenen virtuellen Adressräumen (Regions) laufen.

Der TSO Terminal Control Access Space(TCAS) wird beim Hochfahren von z/OS gestartet.

Wenn immer ein Benutzer sich in TSO einlogged, richtet TCAS für ihn einen eigenen TSO Address Space eingerichtet.

In dem hier gezeigten Beispiel haben sich 2 Benutzer (User A und User B) jeweils mit ihrem eigenen Bildschirm Terminal eingelogged. Jeder der beiden TSO Benutzer hat nur Zugriff zu seinem eigenen Address Space.

Der linke der beiden TSO Benutzer übergibt gerade an JES einen Stapelverarbeitungsauftrag.

Parallel dazu existieren mehrere Address Spaces für (in diesem Beispiel) eine Kombination der CICS und der DB2 Subsysteme. CICS Programme kommunizieren intern direkt mit DB2 Programmen.

Parallel zu den TSO Benutzern haben sich 4 Benutzer mit Ihren Bildschirm Terminals in das CICS Subsystem eingelogged. Während bei TSO für jeden Benutzer ein eigener Address Space eingerichtet wird, existiert für alle Benutzer (plus dem CICS Subsystem selbst) nur ein einziger Address Space.

Dargestellt sind weiterhin virtuelle Adressenräume für den Master Scheduler, JES mit 3 Initiators, sowie für VTAM, RACF und SMS Subsysteme. Der Master Scheduler fährt z/OS hoch, steuert den laufenden Betrieb, und ermöglicht es einem Administrator mittels einer Master Konsole den laufenden Betrieb zu beobachten und in ihn einzugreifen. VTAM ist Teil des Communication Subsystems, RACF ist das Security Subsystem, und SMS ist weiter unten erläutert.

JES holt sich ein Script aus seiner Procedure Bibliothek. Die Initiators laden Programme aus Programmbibliotheken.

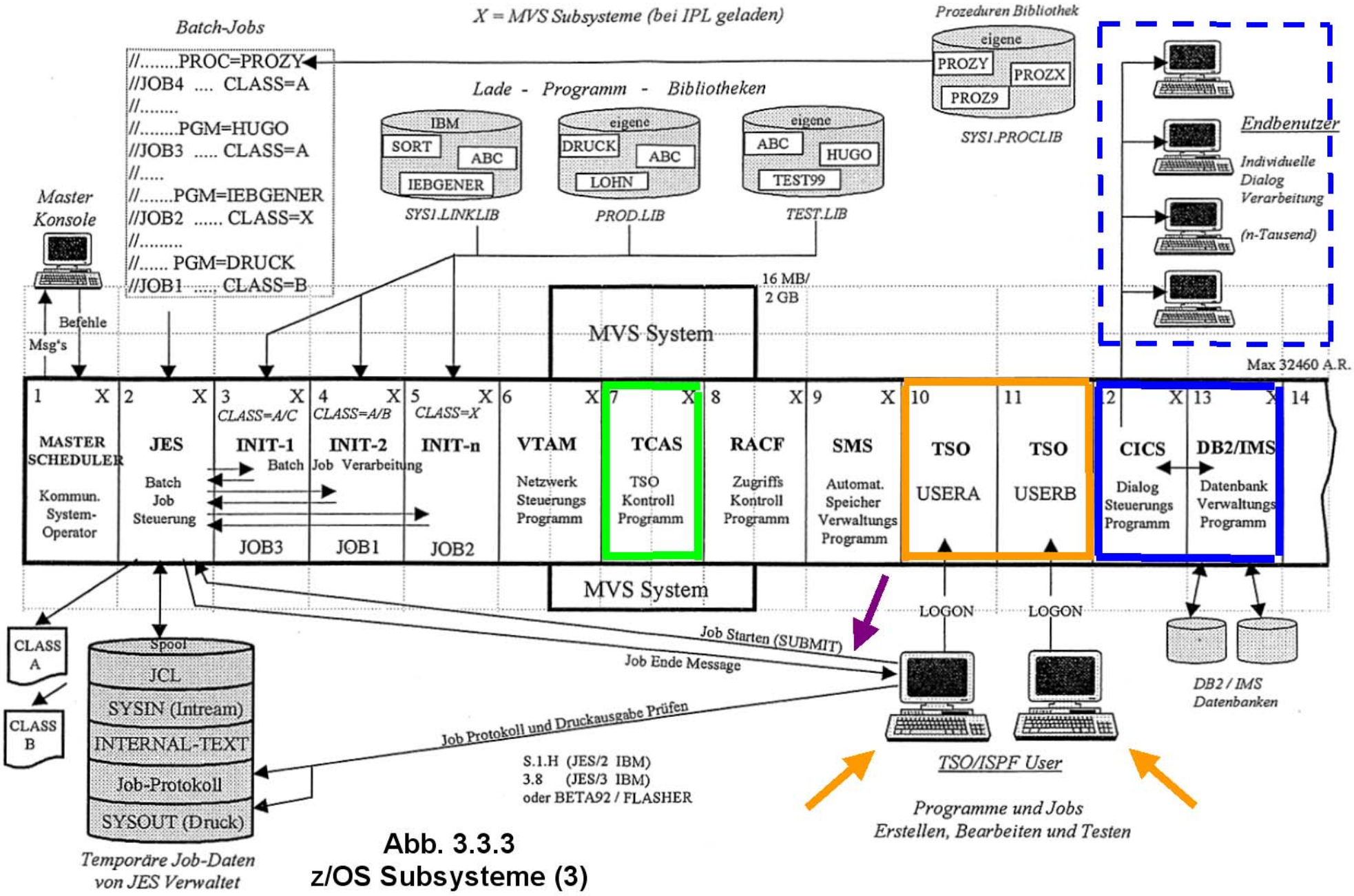


Abb. 3.3.3
z/OS Subsysteme (3)

3.3.4 TSO Logon

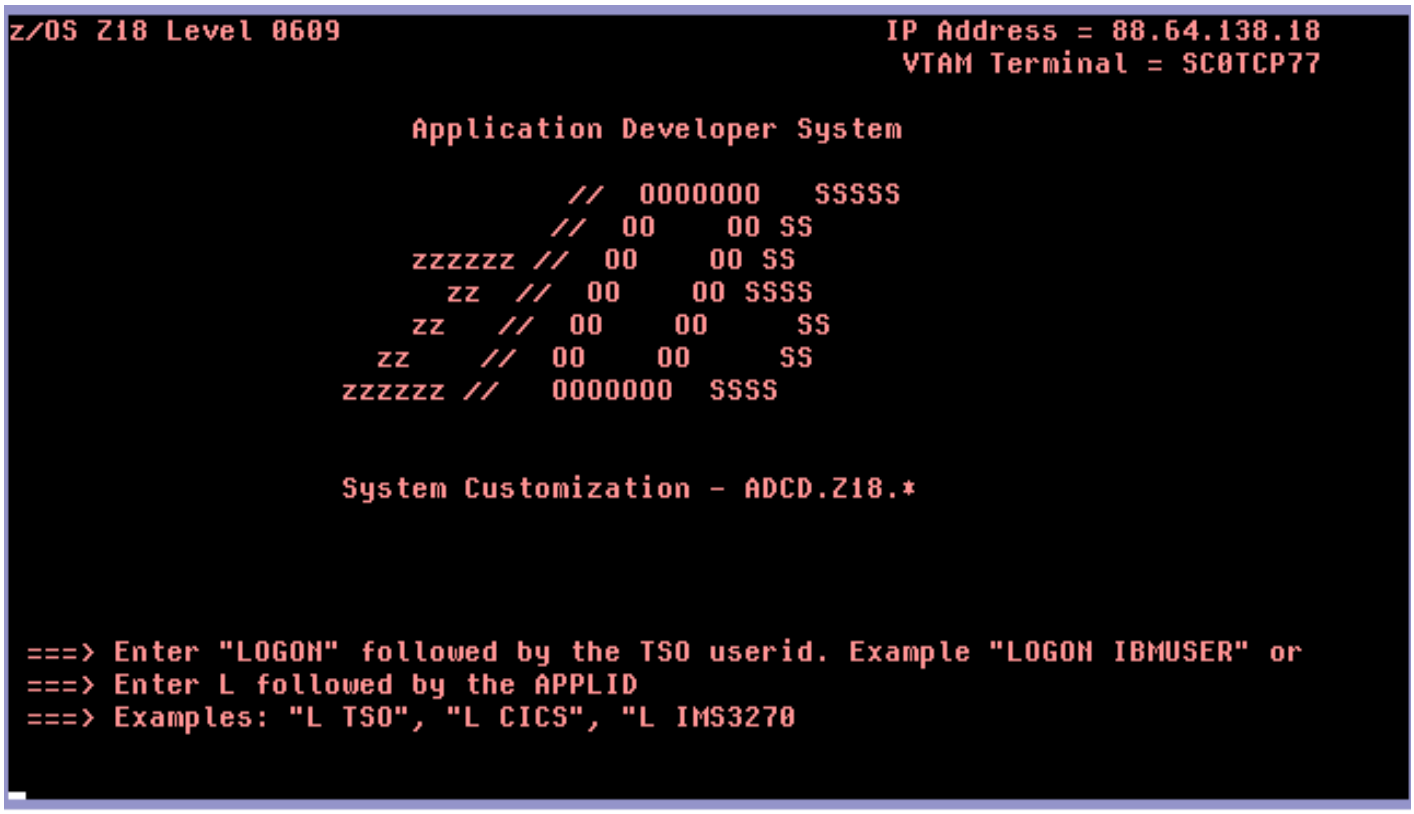


Abb. 3.3.4
z/OS Welcome Screen

Diesen Welcome Screen sehen sie, wenn Sie sich in unseren Mainframe Rechner jedi.informatik.uni-leipzig.de oder 139.18.4.30 einloggen. Von hier aus können Sie sich in eins von mehreren Subsystemen, wie z.B. TSO oder CICS einloggen

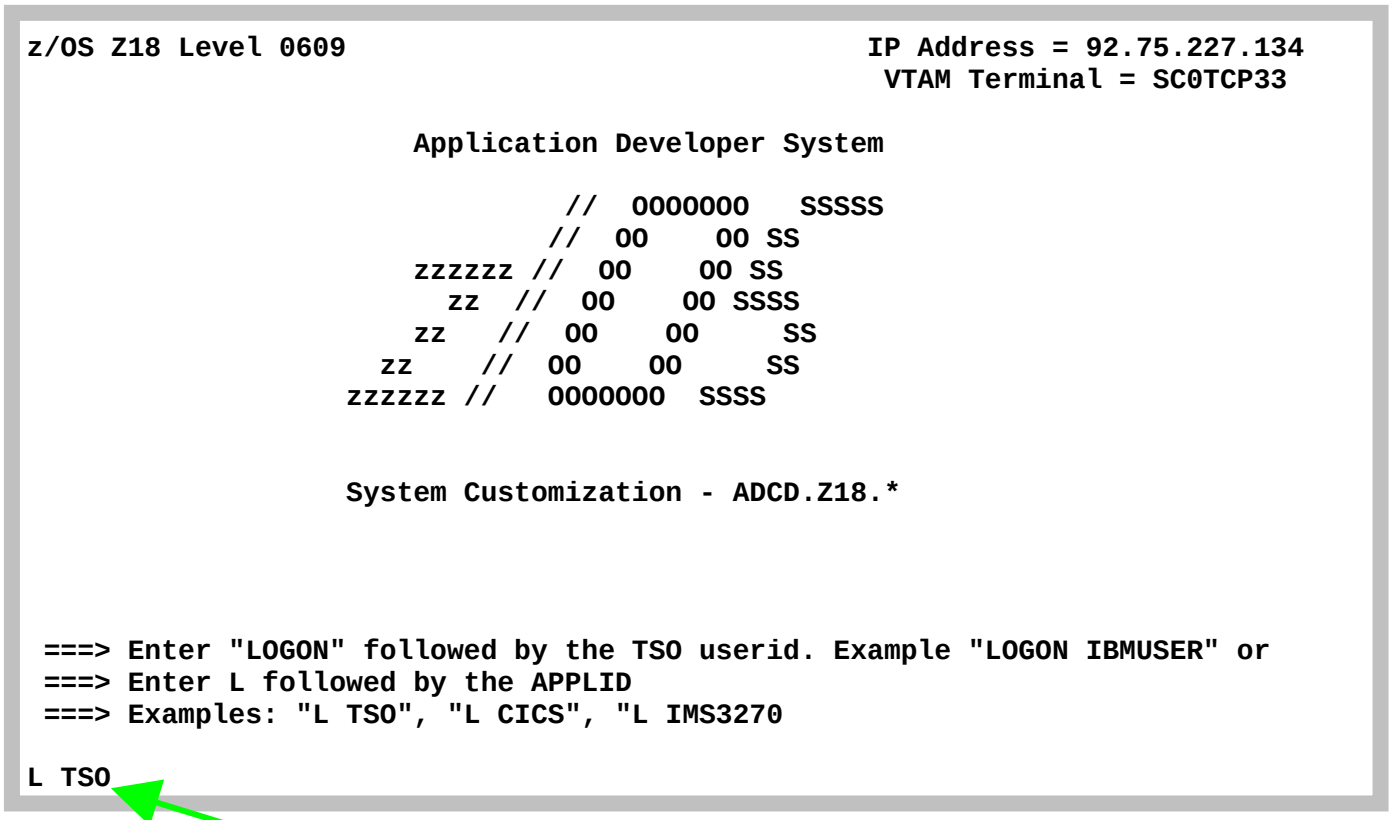


Abb. 3.3.5
Logon gibt an, welches von mehreren z/OS Subsystemen aufgerufen werden soll

Hier wird gerade das Logon (abgekürzt L) für das TSO Subsystem eingegeben.

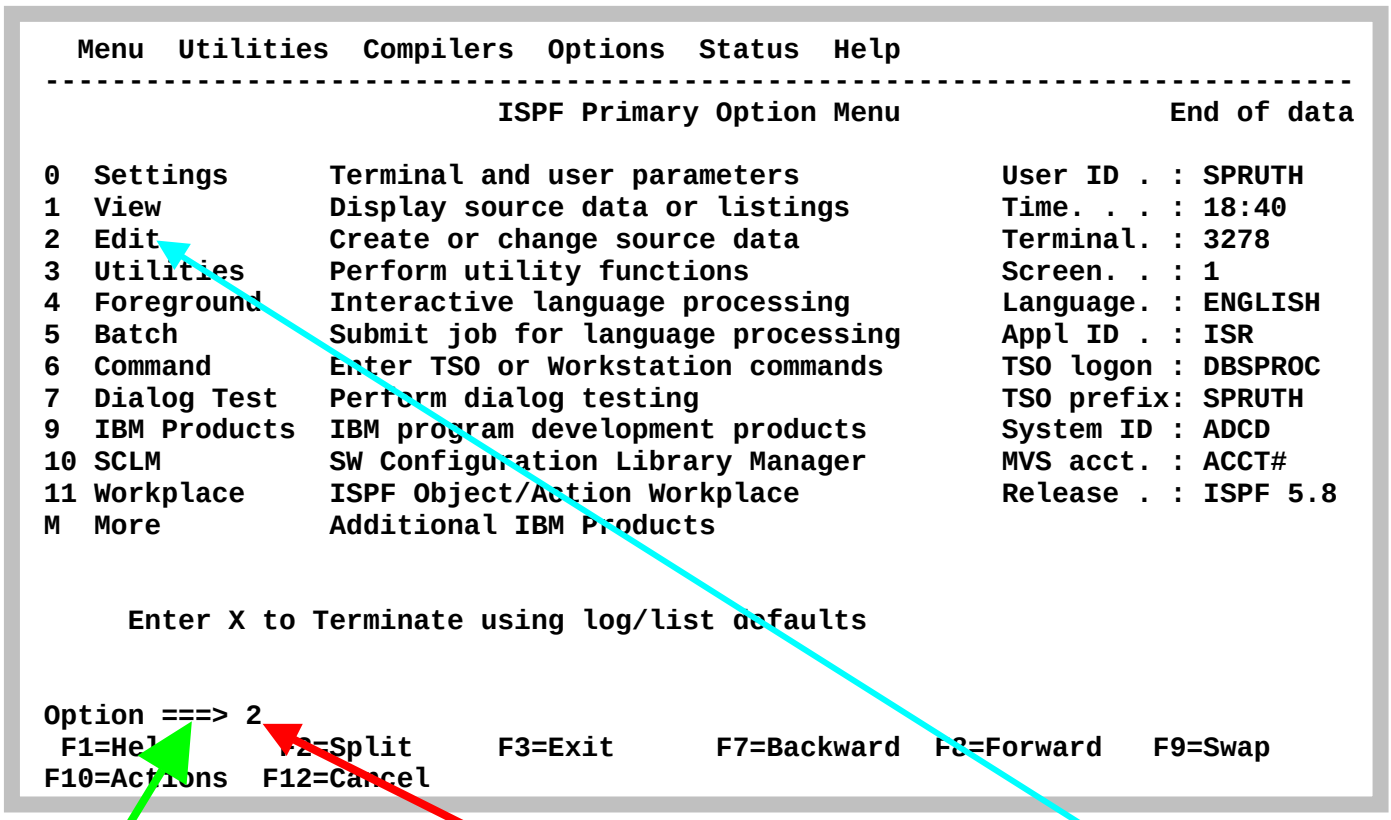


Abb. 3.3.6
Der Text gibt an, was auf der Kommandozeile eingegeben werden soll

===> ist das Cursor Symbol. Eingabe einer „2“ hinter dem Cursor ruft den „ISPF“ Editor auf.

3.3.5 Dateisystem

Betriebssysteme wie Unix, Linux oder Windows speichern Daten in Files (Dateien). Die Identifizierung von Dateien erfolgt über Dateiverzeichnisse, die selbst wie Dateien aussehen und wie Dateien behandelt werden können.

Ein Dateisystem (File System) verbirgt die Eigenschaften der physischen Datenträger (Plattenspeicher, CD, evtl. Bandlaufwerke) weitestgehend vor dem Anwendungsprogrammierer. Für ein Betriebssystem existieren typischerweise mehrere inkompatible File Systeme, z. B. FAT32 oder NTFS unter Windows, oder extf2, extf3 und Reiser unter Linux.

Unter Unix, Linux und Windows sind Dateien strukturlose Zeichenketten (unstrukturierte Menge von Bytes), welche über Namen identifiziert werden. Hierfür dienen Dateiverzeichnisse, die selbst wie Dateien aussehen und behandelt werden können.

Die meisten z/OS Dateien haben im Gegensatz zu Linux oder Windows Dateien zusätzliche Struktureigenschaften, und werden dann als Data Sets bezeichnet. Viele Data Sets bestehen aus Records. Hier ein Record Beispiel: Angenommen eine Datei, die das Telefonverzeichnis einer Stadt speichert. Die Datei besteht aus vielen Records (Einträgen). Jeder Record besteht aus Feldern, z.B. Nachnahme, Vornahme, Straße, Haus Nr., Tel. Nr. usw.

Im Gegensatz zu einer Unix File speichert eine Unix SQL Datenbank die Daten strukturiert in der Form von Tabellen (Tables), Relationen, Rows, Columns usw. Die Rows innerhalb einer Unix SQL Datenbank Tabelle enthalten Records, im Gegensatz zu Unix Files.

z/OS verwendet gleichzeitig mehrere Filesysteme. Eine z/OS „Access Method“ definiert das benutzte Filesystem und stellt gleichzeitig Routinen für den Zugriff auf die Records des Filesystems zur Verfügung. Das Filesystem wird durch die Formatierung eines physischen Datenträgers definiert. Bei der Formatierung der Festplatte wird die Struktur des Datasets festgelegt. Es gibt unter z/OS unterschiedliche Formatierungen und Zugriffsmethoden für Datasets mit direktem Zugriff , sequentiellen Zugriff oder index-sequentiellen Zugriff.

Genauso wie bei Unix existieren eine ganze Reihe von File Systemen (Access Methods) mit Namen wie BSAM, BDAM, QSAM, ISAM usw. Die wichtigste Access Method (File System) hat den Namen VSAM (Virtual Storage Access Method).

Dataset Zugriffe benutzen an Stelle eines Dateiverzeichnisse „Kontrollblöcke“, welche die Datenbasis für unterschiedliche Betriebssystemfunktionen bilden. Die Kontrollblöcke haben exotische Namen wie VTOC, DSCB, DCB, UCB, deren Inhalt vom Systemprogrammierer oder Anwendungsprogrammierer manipuliert werden können.

Unter z/OS werden Datenbanken ebenfalls in Data Sets abgespeichert. Eine Datenbank ist also eine höhere Abstraktionsebene als ein Data Set. Während DB2 unter z/OS hierfür normale z/OS Data Sets verwendet, benutzen Unix Datenbanken hierfür häufig Files mit proprietären Eigenschaften sowie direkte (raw) Zugriffe auf Dateien.

3.3.6 Blocks, physische und logische Records

In der Anfangszeit von OS/360 wurde bei einem Plattenspeicherzugriff ein einzelner Record zwischen Plattenspeicher und Hauptspeicher transportiert. Dies ist nicht sehr effektiv, weil Plattenspeicherzugriffe relativ lange dauern (häufig mehr als 10 ms). Records sind meistens nicht sehr lang; Sie würden überrascht sein, wie populär heute noch eine Record Länge von 80 Bytes ist (in Anlehnung an die 80 Spalten der IBM Lochkarte).

Man ging dazu über, bei jedem Plattenspeicherzugriff eine Gruppe von nebeneinanderliegenden Records auszulesen, in der Hoffnung, dass das Anwendungsprogramm demnächst einen benachbarten Record brauchen würde, der sich dann schon im Hauptspeicher befand. Besonders bei der sequentiellen Verarbeitung von Datenrecords ist das sehr effektiv.

Man bezeichnete diese Gruppe von gleichzeitig ausgelesenen Records als Block oder physischen Record, im Gegensatz zum eigentlichen Record, der dann als logischer Record bezeichnet wird.

Ein Block (physischer Record) besteht also aus mehreren logischen Records, welche die eigentlichen Verarbeitungseinheiten darstellen. Die Blockungsgröße definiert, wie viele logischen Records in einem physischen Record (Block) untergebracht werden können.

Wenn man heute von Records redet, meint man damit meistens logische Records.

3.3.7 Benutzung einer z/OS Datei

Wenn sie sich das erste Mal unter TSO einloggen ist Ihre erste Aufgabe die Zuordnung (allocate) von Data Sets.

Ihre Frage sollte sein: Was soll das ? „Das habe ich unter Windows noch nie gemacht“.

Dies ist falsch !!! Wenn Sie unter Windows eine neue Datei erstmalig anlegen, müssen Sie z. B. entscheiten, ob sie unter C: oder D: gespeichert wird. Wenn Ihr Rechner über 24 Plattenspeicher verfügt, z.B.

C:, D:, E:, ...Z: ,

wird die Verwaltung schon etwas schwieriger. Bei der Auswahl kann auch sein, dass einige der Platten (welche ?) mit FAT32 und andere mit NTFS formatiert sind, dass sie unterschiedliche Kapazität und/oder Performance Eigenschaften haben , und dass dies für Ihre Entscheidung relevant sein mag. Stellen Sie sich vor, Ihr Rechner hat eine normale Festplatte und zusätzlich eine besonders schnelle, aber kleine, Solid State Disk.

Was machen Sie wenn Ihr z/OS Rechner über 60 000 Plattenspeicher verfügt, dazu 1500 Solid State Disks ?

Sie verwenden für die Verwaltung eine z/OS Komponente Storage Management System (SMS). Wenn Sie Platz für eine neue Datei brauchen, melden Sie dies bei SMS an. Dieser Vorgang wird als Allocation bezeichnet.

Genau genommen besteht das z/OS Storage Management System aus mehreren Komponenten. Die wichtigste dieser Komponenten heißt DFSMS (Data Facility Storage Management System).

Die Menge aller von SMS verwalteten Datasets wird als „System Managed Storage“ bezeichnet. In der Vergangenheit war es möglich, Datasets auch ohne Benutzung von SMS zu verwalten. Das ist heute unüblich.

3.3.8 Erstellen einer Datei: Entscheidungen

Benutzer, die Datenverarbeitung betreiben, haben beim Umgang mit den Daten täglich viele Entscheidungen zu treffen. Zusätzlich zum Umgang mit Themen, die nur die Daten oder die Anwendung betreffen, müssen sie die Speicherverwaltungsmaßnahmen der Installationen kennen. Sie müssen sich außerdem mit den Themen auseinandersetzen, die Format, Bearbeitung und Positionierung der Daten betreffen:

- **Welchen Wert soll die Blockgröße haben? Wie viel Speicherplatz ist erforderlich?**
- **Welcher Festplattentyp soll verwendet werden? Sollen die Daten in den Cache geschrieben werden? Soll eine Fehlerbehebung durchgeführt werden?**
- **Welche Datenträger stehen für die Dateipositionierung zur Verfügung?**
- **Wie oft soll eine Sicherung oder Migration durchgeführt werden? Soll die Sicherung/Migration erhalten bleiben oder (wann ?) gelöscht werden?**

Wenn die Verwendung von Datenverarbeitungsservices vereinfacht werden soll, müssen dem System einfachere Schnittstellen zur Verfügung gestellt werden. Insbesondere JCL ist einer der Bereiche, in denen Vereinfachungen vorgenommen werden.

3.3.9 Data Facility Storage Management System

Angesichts der unübersichtlich großen Anzahl von Plattenspeichern, File Systemen und Data Sets werden letztere in Klassen eingeteilt. Hierfür ist eine z/OS Komponente zuständig, das Data Facility Storage Management System (DFSMS). Beim Neuanlegen eines Data Sets müssen angegeben werden:

Data Class

Eine Data Class fasst Data Sets mit identischen File System Attributen wie Record Format, Record Länge, Schlüssellänge bei VSAM Dateien, usw. zusammen.

Storage Class

Eine Storage Class fasst mehrere Festplattenspeicher mit identischen Performance Eigenschaften wie Antwortzeit (ms), Nutzung von Read/Write Caching, Verwendung der Dual Copy Funktion usw. zusammen. Beispielsweise könnte man z.B. alle Solid State Drives (Solid State Disks) zu einer Storage Class zusammenfassen

Management Class

Eine Management Class fasst mehrere Plattenspeicher zu einer Gruppe zusammen, wenn diese identisch verwaltet werden. Verwaltungseigenschaften sind z.B. Migration nach x Tagen, Anzahl Backup Versionen, schrittweiser Abbau der Backup Versionen, Löschen der Daten, ...

Eigenschaften der Management Class betreffen den Lebenszyklus eines Data Sets. Alle Dateien und Data Sets in einem Unternehmen haben einen Lebenszyklus: Sie entstehen irgendwann, werden verarbeitet und geändert und werden irgendwann wieder gelöscht. Das Was, wie und Wo ist in der Management Class festgelegt.

Es werden beispielsweise Eigenschaften wie die Folgenden festgelegt:

- **Das Anlegen (allocate) der Datei erfolgt durch den Benutzer**
- **Benutzung (Schreiben und/oder Lesen der Daten)**
- **Es werden Sicherungskopien angelegt**
- **Angaben zur Platzverwaltung (wann wird der Data Set freigegeben / erweitert / komprimiert)**
- **Auslagern von inaktiven Dateien, sowie Wiederbenutzung**
- **Ausmustern von Sicherungskopien**
- **Wiederherstellung von Dateien**
- **Löschen der Datei**

DFSMS bewirkt die Zuordnung eines Data Sets zu einer Storage Group, einer Gruppe von Plattenspeichern (Volume) in der gleichen Data Class, Storage Class und Management Class.

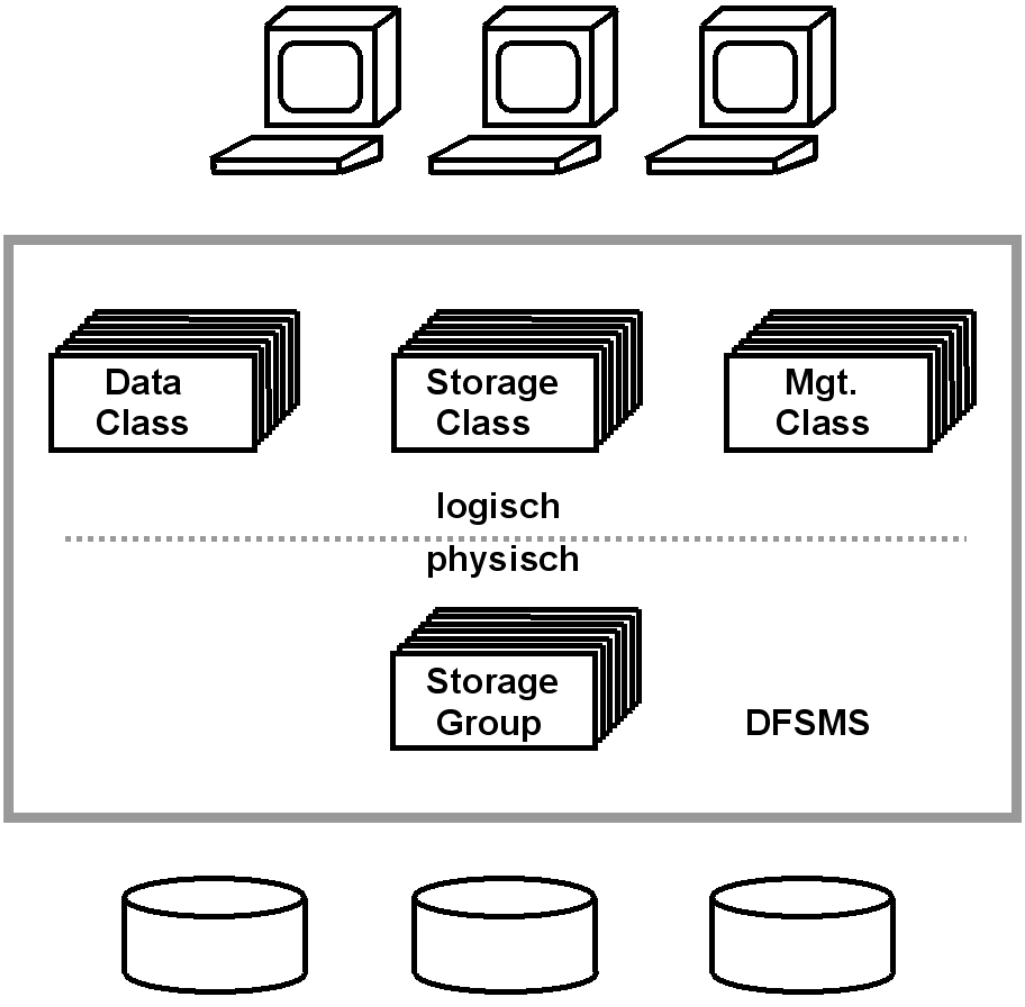


Abb. 3.3.7
DFSMS Klassen

Der Benutzer sieht nur die logische Sicht der Daten:

- **vorbereitete Datenmodelle in den Datenklassen**
- **in den Storage Klassen festgelegte Serviceanforderungen**
- **Management Kriterien für die Auslagerung der Daten**

Festplattenspeicher mit der gleichen Data Class, Storage Class und Management Class werden zu einer Storage Gropup zusammengefasst.

3.3.10 Allocating a new Data Set

```
  Menu  RefList  Utilities  Help
--------------------------------------------------------------------------------
                           Allocate New Data Set
                                                                        More:     +
Data Set Name  . . . : PRAKT20.TEST.DATASET

Management class . . . DEFAULT        (Blank for default management class)
Storage class  . . . . PRIM90         (Blank for default storage class)
 Volume serial . . . .                (Blank for system default volume) **
 Device type . . . . .                (Generic unit or device address) **
Data class . . . . . .                (Blank for default data class)
 Space units . . . . . KILOBYTE       (BLKS, TRKS, CYLS, KB, MB, BYTES
                                       or RECORDS)
 Average record unit                  (M, K, or U)
 Primary quantity  . . 16             (In above units)
 Secondary quantity    1              (In above units)
 Directory blocks  . . 2              (Zero for sequential data set) *
 Record format . . . . FB
 Record length . . . . 80
 Block size  . . . . . 320
 Data set name type  : PDS            (LIBRARY, HFS, PDS, or blank)  *
                                      (YY/MM/DD, YYYY/MM/DD
Command ===>
 F1=Help      F3=Exit     F10=Actions  F12=Cancel
```

Abb. 3.3.8
ISPF Allocate Screen

Abbildung 3.3.8 gibt den Bildschirminhalt (Screen) wieder, mit dem Sie das Allocating eines Datasets vornehmen.

In dem gezeigten Beispiel ist die Management Class mit „Default“ vorgegeben, desgleichen die Storage Class mit PRIM90.

Bei der Data Class haben Sie die Auswahl, die gewünschte Größe in Bytes, Kilobytes, Megabytes, Records, Blöcken, Plattenspeicher-Spuren (Tracks) oder Zylindern anzugeben.

Die angegebene Größe ist 16 KByte (Primary Quantity) mit einer Reserve von 1 KByte (Secondary Quantity) . Elemente innerhalb des Data Sets können mit 2 Directory Blocks adressiert werden.

Alle Records haben die gleiche Länge (Fixed Block, FB). Die Länge der logischen Records (Record Length) beträgt 80 Bytes. Vier logische Records werden zu einem physischen Record zusammengefasst
(Block size = 4 x 80 Bytes, = 320 Bytes).

Es existieren eine ganze Reihe von unterschiedlichen File Systemen für z/OS Data Sets. (z/OS bezeichnet die File Systeme als „Access Methods“). Das hier gewählte File System ist PDS (was immer das auch sein mag, wir schauen es uns später in Abschnitt 6.1 an).

3.4 z/OS Subsysteme

3.4.1 Die wichtigsten z/OS Subsysteme

Einige der wichtigsten z/OS Subsysteme (häufig auch als Server bezeichnet) sind:

CICS Transaction Manager
IMS Transaktionsmanager
DB2 Datenbank
IMS Datenbank
JES2oder JES3
Security Server (RACF, DCE Security, Firewall)
Netview, Systemview
WebSphere
UNIX System Services
Distributed Computing Services (DCE, NFS, DFS, FTP)
Run Time Language Support für

C/C++	**PL/1**
COBOL	**Fortran**
Object Oriented Cobol	**Assembler**

C/C++ Open Class Library

Weitere häufig anzutreffende Subsysteme stammen nicht von IBM, sondern von anderen Herstellern, z.B. Computer Associates und BMCsoftware.

3.4.2 z/OS Prozessverwaltung

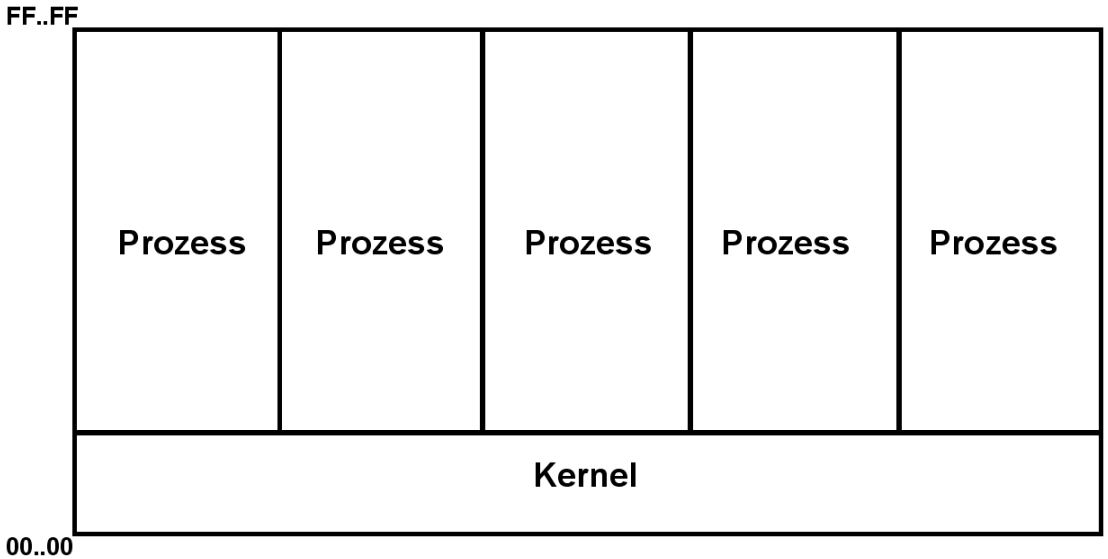


Abb. 3.4.1
Prozess Ansatz

Bei der Ausführung von Programmen unterscheiden wir zwischen Prozessen und Threads.

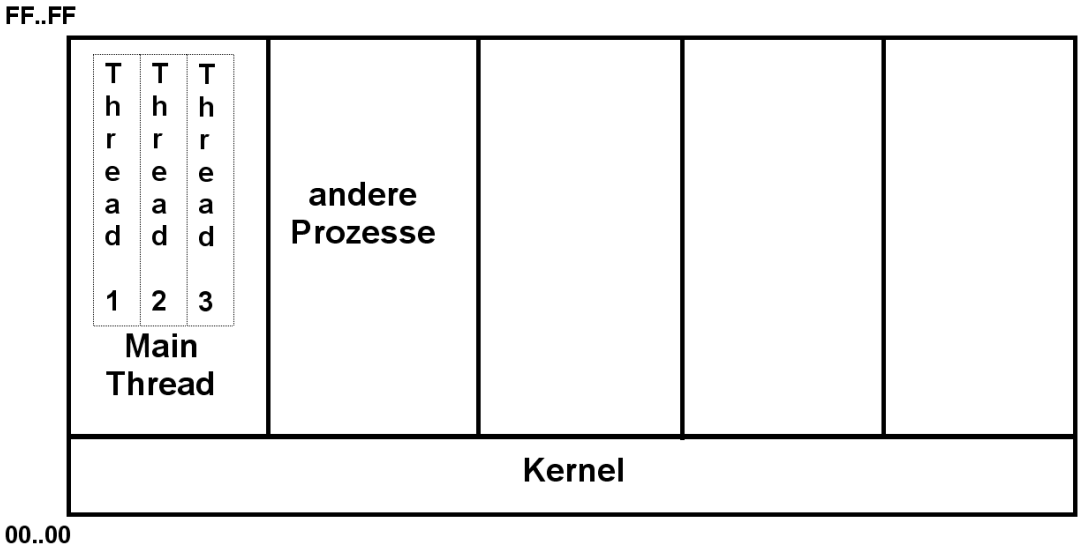


Abb. 3.4.2
Thread Ansatz

Prozesse laufen in getrennten virtuellen Adressenräumen, siehe Abb. 3.4.1. Threads sind unabhängige Ausführungseinheiten, die innerhalb des gleichen virtuellen Adressenraums ablaufen, siehe Abb. 3.4.2. Ein Wechsel zwischen Threads erfordert deutlich weniger Aufwand als ein Wechsel zwischen Prozessen.

Ein z/OS Prozess besteht aus mehreren Arbeitseinheiten, die als „Tasks“ bezeichnet werden. Eine „Main Task“ repräsentiert einen Prozess und typischerweise einen Address Space. Eine Main Task kann Subtasks generieren; eine Subtask entspricht in etwa einem Thread in Unix oder Windows. Subtasks laufen im gleichen Address Space wie die Main Task.

Wenn ein Prozess gestartet wird, erstellt z/OS hierfür einen Main Task Control Block (TCB). Ein TCB entspricht in etwa einem Unix Process Control Block (PCB).Das Programm kann weitere Subtasks erstellen mit Hilfe des ATTACH System Aufrufes. Hierbei wird jeweils ein eigener Subtask TCB erzeugt. Da der ATTACH Overhead relativ groß ist, implementieren zeitkritische Subsysteme wie z.B. CICS ihr eigenes Subtasking. Bei der Nutzung der Unix System Services (siehe weiter unten) wird empfohlen, die POSIX Funktion `pthread_create` an Stelle der z/OS ATTACH Makro zu benutzen.

Im Kernelmodus (Supervisor State) existieren zusätzliche Mechanismen, die als Service Request Blocks (SRB) bezeichnet werden. SRBs werden benutzt, um Systemroutinen auszuführen. Windows kennt im Kernelmodus einen ähnlichen Mechanismus, der als „Fibers“ (light weight threads) bezeichnet wird. Fibers werden durch die Anwendung gescheduled. Die CICS Portierung auf Windows benutzt Fibers.

3.4.3 DB2 und IMS Datenbanken

DB2 Universal Database (UDB) ist IBMs relationales Datenbank-Produkt. Es existiert eine identische Implementierung für alle UNIX-, Linux-, OS/2-, und Windows-Betriebssysteme.

Eine zweite getrennte Implementierung mit dem gleichen Namen und erweitertem Funktionsumfang ist für das z/OS Betriebssystem verfügbar, Obwohl es sich um zwei getrennte Implementierungen handelt, ist die Kompatibilität sehr gut.

Neben Oracle- und Microsoft-SQL ist DB2 eines der drei führenden relationalen Datenbankprodukte. Unter z/OS ist DB2 neben IMS die am häufigsten eingesetzte Datenbank. Andere populäre z/OS Datenbanksysteme sind IDMS der Fa. Computer Associates sowie Adabas der Fa. Software AG. Die z/OS Version von Oracle wird seit 2009 nicht mehr weiter entwickelt.

DB2 ist eine Server-Anwendung, die grundsätzlich in einem getrennten Adressraum läuft. Wie bei allen Server-Anwendungen ist ein Klient erforderlich, um auf einen DB2-Server zuzugreifen. Der Klient kann ein Anwendungsprogramm auf dem gleichen Rechner sein, oder auf einem getrennten Rechner laufen.

Der Zugriff kann mit Hilfe von SQL Statements erfolgen, die in einem Anwendungsprogramm eingebettet sind. Alternativ existieren eine Reihe spezifischer SQL Klienten-Anwendungen.

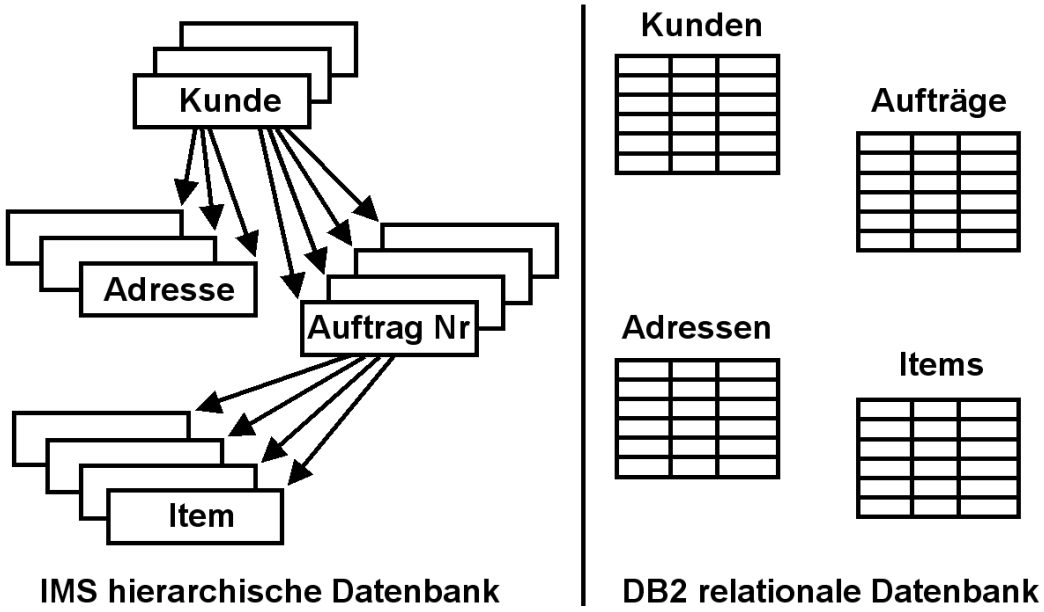


Abb. 3.4.3
Gegenüberstellung der IMS und DB2 Datenmodelle

Neben DB2 ist IMS ein häufig unter z/OS eingesetztes Datenbanksystem. IMS ist im Gegensatz zu DB2 ein nicht-relationales, hierarchisches Datenbanksystem.

IMS benutzt Bäume (Tree) an Stelle der Tabellen in DB2. Das IMS hierarchische Datenmodell besteht aus einer geordneten Menge von Bäumen, genauer aus Ausprägungen von Bäumen eines bestimmten Typs. Jeder Baum-Typ enthält eine Basis (Root-Segment) und keinen, einen oder mehrere Unterbaum-Typen (Segmente). Der Unterbaum-Typ seinerseits enthält ggf. weitere Unterbäume.

Ein IMS Baum besteht aus Segmenten. Ein Segment ist jeweils ein logischer Satz bestehend aus einem oder mehreren Feldern (Fields).Wie in Abbildung 3.4.4 dargestellt, kann ein Elternsegment mehrere Kindsegmente haben. Auch ein Kindsegment kann zur gleichen Zeit, ein Elternsegment sein, das heißt, es kann eigene Kindersegmente haben. Das Segment ohne Elternsegment, das sich an der Spitze des Baumes befindet, wird Wurzel Segment (Root Segment) genannt.

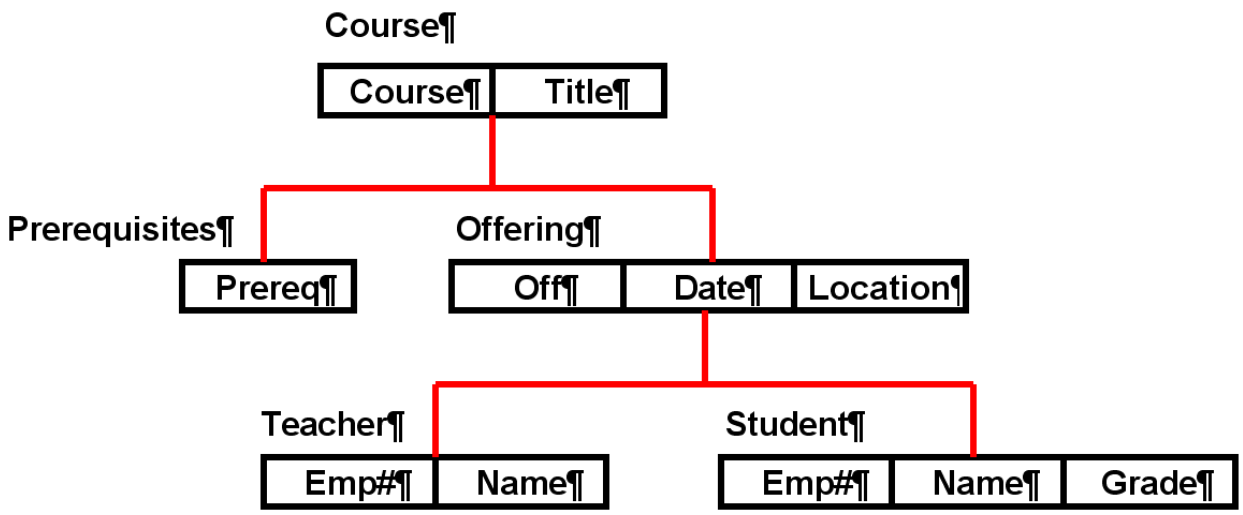


Abb. 3.4.4
Datenmodell einer IMS Datenbank

Eine IMS Datenbank besteht aus vielen Records, die alle die gleiche Struktur haben. Abb. 3.4.4 zeigt als Beispiel die Struktur eines Lehrgangs, der von einem Schulungsunternehmen angeboten wird. Abb. 3.4.5 zeigt als Beispiel einen (von vielen) Records mit dieser Struktur, der einen spezifischen Lehrgang, seine Termine und seine dazugehörigen Buchungen darstellt. Die verschiedenen Records der IMS Datenbank können eine unterschiedliche Anzahl von Ausprägungen jedes einzelnen Kindsegmentes haben.

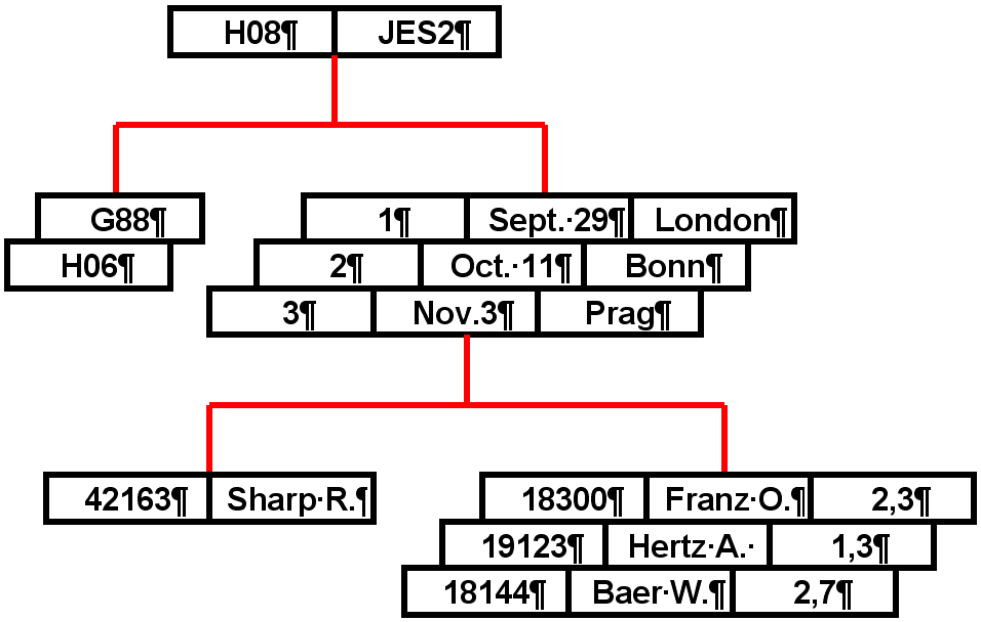


Abb. 3.4.5
Ausprägung eines einzelnen Records einer IMS Datenbank

Die logische Struktur der Datenbank ist in der Database Description (DBD) und in den Program Communication Blocks (PCB) definiert. Dabei entspricht ein DBD grob einer Tabelle und ein PCB einer Sicht in relationalen Datenbanken, genauer einem Create-Table- bzw. einem Create-View-Befehl.

Die Sprache DL/I (Data Language One) ist das IMS Äquivalent zu der SQL Sprache. Data Language/I (DL/I) Function Calls werden von Anwendungsprogrammen für Query und Update Vorgänge benutzt.

Die Baum Struktur von IMS schränkt die Flexibilität ein. Bei einfachen Datenmodellen ermöglicht IMS höhere Transaktionsraten als DB2.

Existierende z/OS Installationen verwenden relativ häufig IMS, allerdings nicht so häufig wie DB2.

Anmerkung: Ursprünglich wurde für die IMS Datenbank ein eigener Transaktionsmanager entwickelt. IMS besteht deshalb aus zwei Komponenten, dem Database Manager (IMS DB) und dem Transaction Manager (IMS DC). IMS DC ist eine Alternative zu dem CICS Transaktionsmanager (siehe Abschnitt 8.1), hat heute aber weniger Bedeutung. Häufig wird der CICS Transaktionsmanager für Zugriffe auf eine IMS Datenbank eingesetzt.

3.4.4 z/OS Communication Server

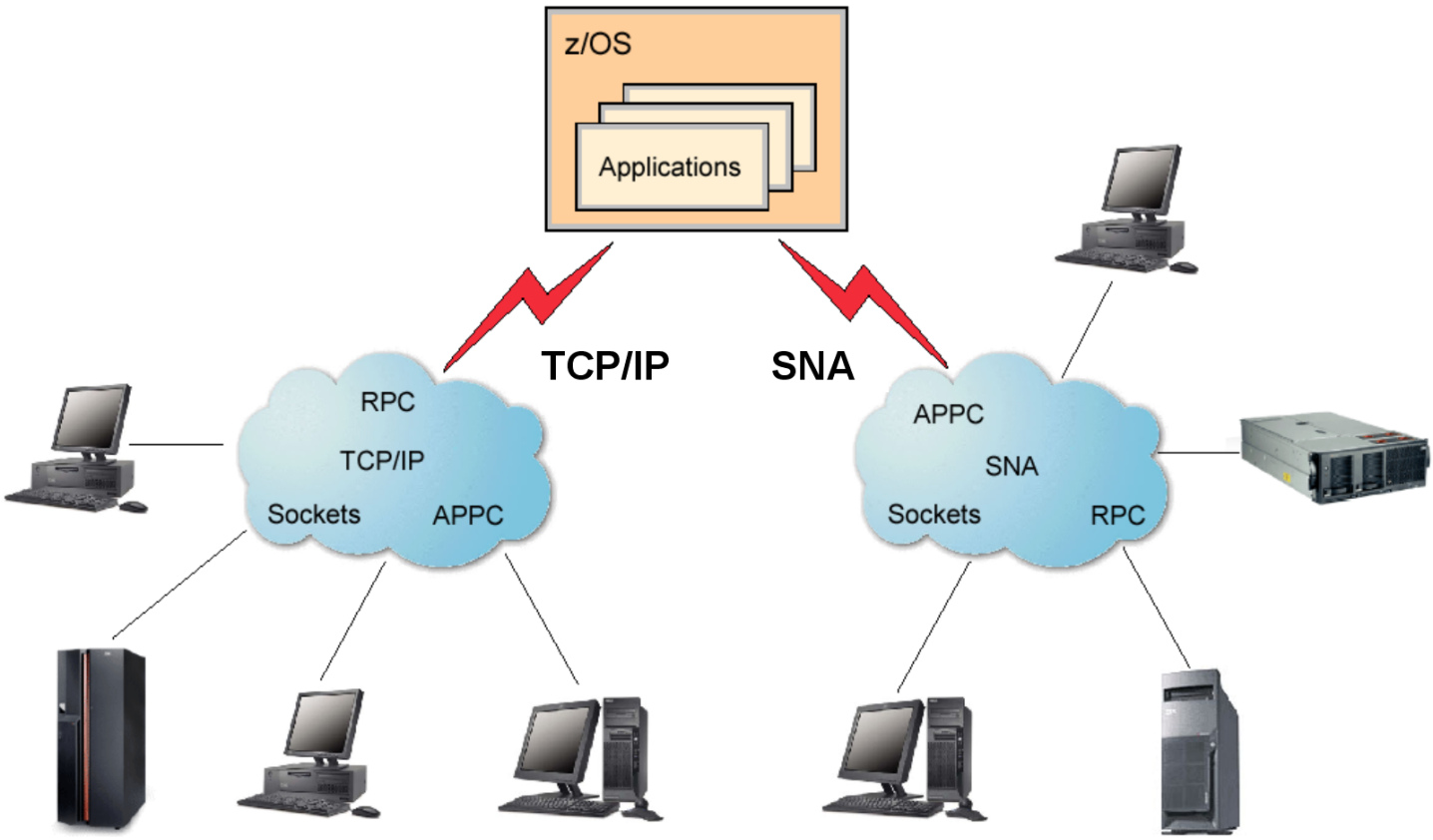


Abb. 3.4.6
Der z/OS Communication Server unterstützt sowohl TCP/IP als auch SNA

Der z/OS Communication Server ist ein eigenständiges Subsystem in einem eigenen virtuellen Adressenraum. Er implementiert die Software (Netzwerk Architektur Stacks) für eine ganze Reihe von Netzwerk Architekturen, besonders für TCP/IP und SNA.

Die Systems Network Architecture (SNA) wurde von IBM entwickelt und im Jahre 1974 vorgestellt. TCP/IP kam erst wesentlich später.

SNA ist vom Funktionsumfang her immer noch sehr modern und in etwa mit TCP/IP Version 6 vergleichbar. Bis in die 90er Jahre war SNA der de facto Standard in den allermeisten Mainframe Installationen. Danach erfolgte ein gradueller Wechsel zu TCP/IP, getrieben durch die Verbreitung des Internet. Heute sind physische SNA Netze fast überall durch TCP/IP Netze ersetzt worden.

Unter der Decke benutzen viele System Komponenten nach wie vor SNA. So kommuniziert Ihr 3270 Emulator über eine SNA Verbindung mit TSO oder CICS. Für den Benutzer ist dies unsichtbar, weil SNA Pakete in TCP/IP Pakete verpackt und über einen TCP/IP Tunnel versandt werden. Für die Kommunikation zwischen einem 3270 Emulator und einem Mainframe verwendet man hierzu ein aufgebohrtes Telnet Protokoll, welches als TN3270 bezeichnet wird. Port 23 ist der Standard Telnet Port, und aus diesem Grunde ist Port 23 der Standard Port für Zugriffe auf einen Mainframe Rechner.

Der z/OS Communication Server hat Zusatz Einrichtungen, mit denen getunnelte TCP/IP Pakete entpackt, und der Inhalt an den SNA Stack weitergegeben werden kann.

3.4.5 z/OS Security Server

Der z/OS Security Server ist ein eigenes Subsystem, welches für sicherheitsrelevante Belange zuständig ist. z/OS ist in Bezug auf Sicherheit allen anderen Betriebssystemen überlegen. Beispiel: Wir betreiben seit 13 Jahren einen Mainframe Server am Lehrstuhl Technische Informatik. Wir haben noch nie ein Security Update installiert und nach unserem Wissen existiert auch kein Virenscanner für z/OS. Zu diesen hervorragenden z/OS Sicherheitseigenschaften tragen viele Faktoren bei; einer davon ist der z/OS Security Server.

Die wichtigsten Komponenten sind:

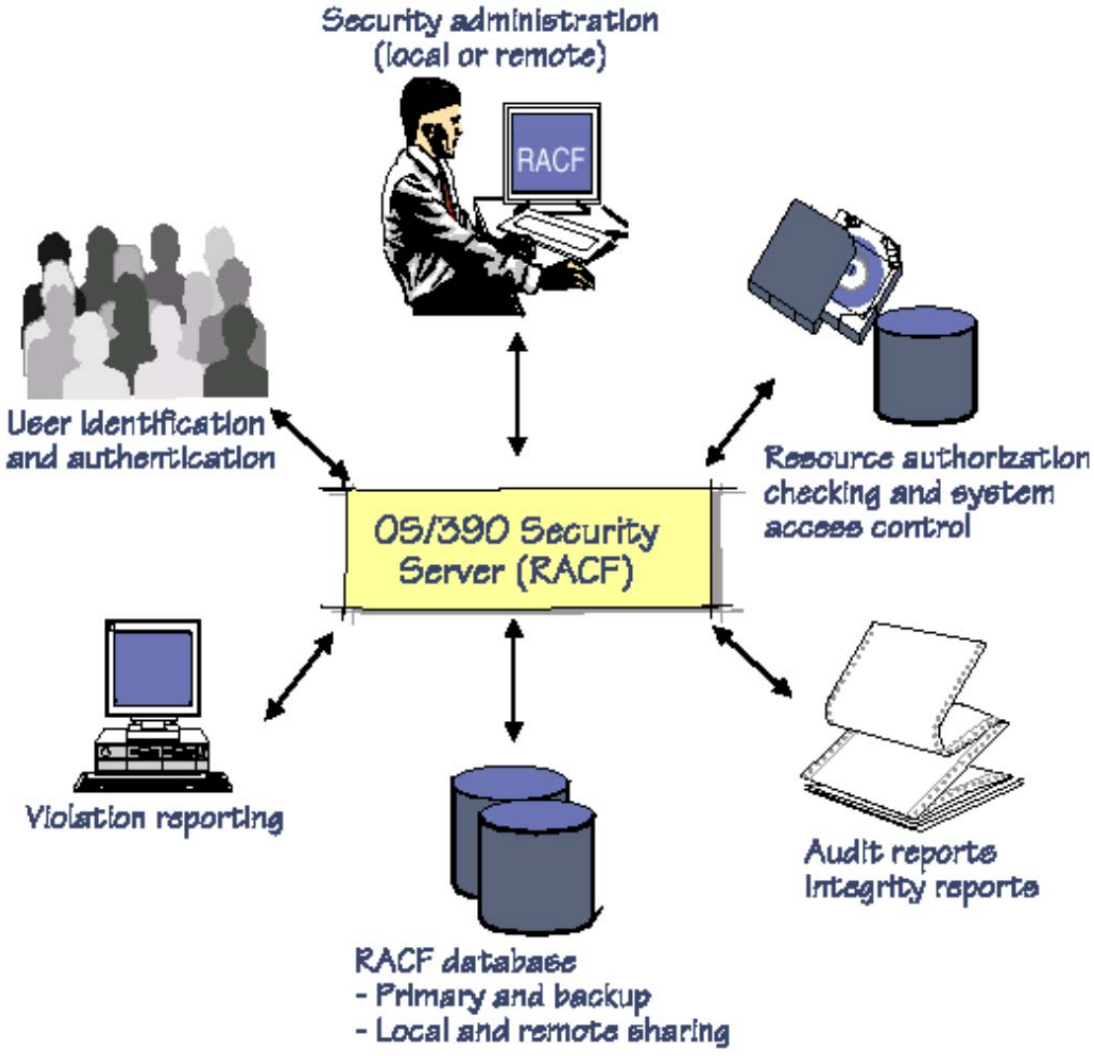
- **LDAP Server - Secure Directory Server**
- **Kerberos Network Authentification Service, einschließlich Kryptographie und Firewall Unterstützung. Hierfür existiert zusätzliche Hardware Unterstützung:**
  - **Zusätzliche Maschinenbefehle für kryptografische Operationen**
  - **Kryptographie Assist-Prozessoren**
- **Resource Access Control Facility (RACF)**

LDAP ist ein Verzeichnisdienst (Directory Service). Ein Verzeichnisdienst ist eine spezielle hierarchische Datenbank mit einer eigenen Abfragesprache. Es hat ähnliche Funktionen wie der DNS Namensdienst des Internets, kann aber darüber hinaus zusätzliche Informationen speichern. Verzeichnisdienste werden in der Regel dazu verwendet, Benutzerdaten zentral zu sammeln und Applikationen zur Verfügung zu stellen. Gegenüber einer dezentralen Speicherung z.B. mit einem eigenen Dienst für jedes Programm hat die zentrale Verwaltung den Vorteil, dass Benutzer und Rechte an zentraler Stelle nur einmal geändert werden müssen.

Kerberos ist ein verteilter Authentifizierungsdienst (Netzwerkprotokoll) für offene und unsichere Computernetze, zum Beispiel das Internet. Es wurde ursprünglich vom Massachusetts Institute of Technology entwickelt. Der Name leitet sich vom Höllenhund Kerberos aus der griechischen Mythologie ab, der den Eingang zur Unterwelt bewacht.

3.4.6 RACF

Das Resource Access Control Facility (RACF) Subsystem ist IBMs Implementierung der Sicherheitsschnittstelle SAF. Neben RACF existieren zwei relativ weit verbreitete alternative Subsystem Implementierungen der Firm Computer Associates mit den Namen ACF2 und Top Secret. Letztere können als Alternative zu RACF eingesetzt werden.



a) **Abb. 3.4.7**
RACF Aufgaben

Die Hauptfunktionen, die RACF erfüllt sind:

- **Identifikation und Verifikation der Benutzer mittels Benutzerschlüssel und Passwortprüfung (Authentifizierung),**
- **Schutz von Ressourcen durch die Verwaltung der Zugriffsrechte (Autorisierung),**
- **Speichern aller Sicherheitsinformationen in einer eigenen Systemdatenbank, der RACF Datenbank,**
- **Signalisieren, wenn Sicherheitsverletzungen auftauchen, und**
- **Logging der Zugriffe auf geschützte Ressourcen (Auditing).**

Literatur : IBM Form No. GC28-1912-06,
http://largescalecomputing.wikispaces.com/file/view/racf+overview.pdf .

RACF benutzt das Konzept von zu schützenden„Resourcen“. Resourcen werden in „Klassen“ aufgeteilt. Beispiele für Klassen sind:

- **Benutzer,**
- **Dateien,**
- **CICS Transaktionen,**
- **Datenbank Rechte,**
- **Terminals.**

Jedem Element einer Klasse wird ein „Profil“ zugeordnet. Das Profil besagt, welche sicherheitsrelevanten Berechtigungen vorhanden sind. Die Profile werden in der RACF Datenbank, abgespeichert.

Vor allem bei der Benutzer Identifikation entscheidet RACF:

- **ob der Benutzer für RACF definiert ist,**
- **ob der Benutzer ein gültiges Passwort, PassTicket, Operator Identification Card und einen gültigen Gruppen-Namen verwendet,**
- **ob der Benutzer das System an diesem Tag und zu dieser Tageszeit benutzen darf,**
- **ob der Benutzer autorisiert ist, auf das Terminal (Tag und Zeit) zuzugreifen,**
- **ob der Benutzer autorisiert ist, auf die Anwendung zuzugreifen,**
- **ob der Benutzer autorisiert ist, auf spezifische Daten zuzugreifen.**

Beispiel: Ein interaktiver Benutzer logged sich ein. Sein RACF Profil enthält den Benutzerschlüssel (Userid), die zu schützenden Ressourcen (Resources) und Gruppen (Groups).Der Login Prozess überprüft das Profil, ob die Login Berechtigung besteht. Er überprüft weiterhin, ob das Terminal, von dem der Benutzer sich einwählt, eine Zugangsberechtigung hat. Hierzu ruft der Login Prozess RACF auf, welches die entsprechenden Profile in seiner RACF Datenbank konsultiert.

Anschließend ruft der Benutzer ein Programm auf, welches auf eine Datei zugreift. Die OPEN Routine ruft RACF auf, welches das Profil der Datei bezüglich der Zugriffsrechte befragt.

Benutzer Profile enthalten „Capabilities“. Datei Profile enthalten „Access Control Listen“.

Zugriffsrechte können von spezifischen granularen Bedingungen abhängig gemacht werden; z.B. der Zugriff auf eine Datenbank kann von der Nutzung eines spezifischen Anwendungsprogramms abhängig gemacht werden, oder auf bestimmte Tageszeiten begrenzt sein.

Es existieren in einer Installation eine sehr große Anzahl an Profilen, die vom RACF Administrator gewartet werden müssen. Ein Problem ist die Wartung der Profile. Beispiel: Ein Unternehmen hat 100 000 Mitarbeiter und User. Einige wechseln ihren Job innerhalb des Unternehmens, was zu anderen Benutzer Berechtigungen führt. Die RACF Profile müssen geändert werden. Für neu eingestellte Mitarbeiter müssen RACF Profile angelegt werden, für ausgeschiedene Mitarbeiter gelöscht werden. Wenn neue Anwendungen installiert werden, bedingt dies ebenfalls einen Aufwand in der RACF Administration.

Die Sicherheit eines z/OS Rechners hängt davon ab, wie korrekt die Einträge in der RACF Datenbank sind.

3.4.7 Security Authorisation Facility

z/OS spezifiziert kritische Events innerhalb des Betriebssystems als sicherheitssensitive Verzweigungen. Die z/OS Security Authorisation Facility (SAF) des z/OS Kernels bewirkt an diesen Stellen den Aufruf einer Security Engine, eines Subsystem Prozesses, der im Benutzer Status läuft.

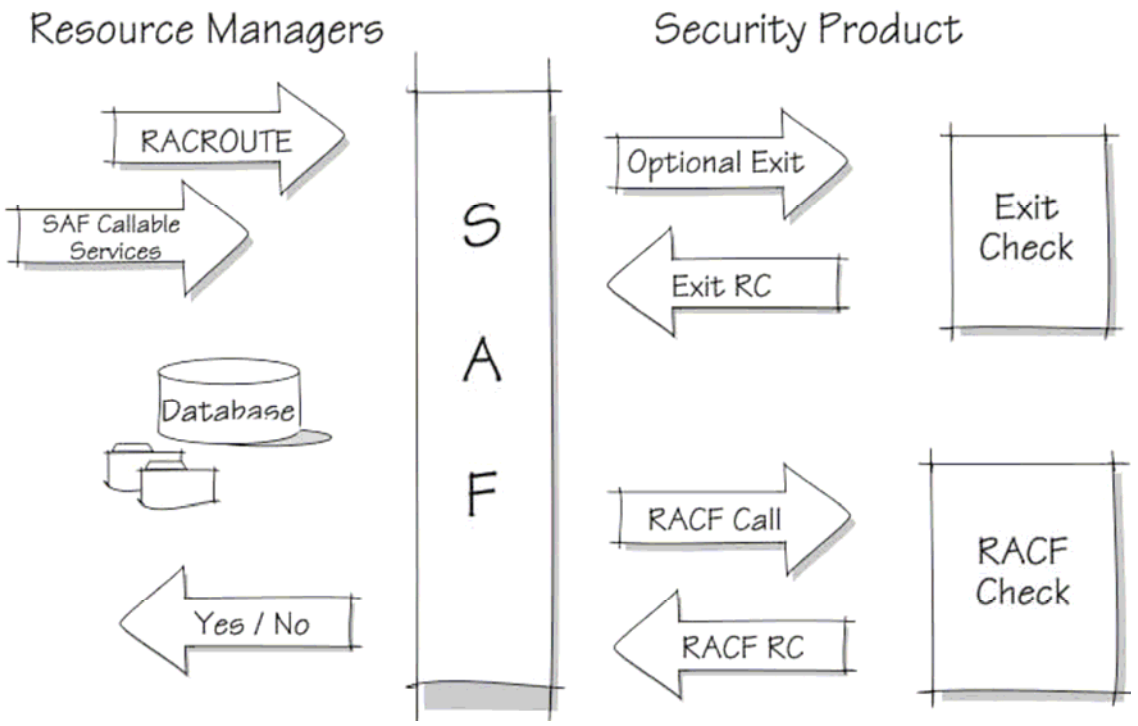


Abb. 3.4.8
Arbeitsweise der Security Authorisation Facility des z/OS Kernels

In z/OS ist diese Security Engine häufig RACF; alternative z/OS Security Engines sind ACF/2 oder TopSecret der Firma Computer Associates.

Zugriffsrechte können von spezifischen granularen Bedingungen abhängig gemacht werden; z.B. der Zugriff auf eine Datenbank kann von der Nutzung eines spezifischen Anwendungsprogramms abhängig gemacht werden, oder auf bestimmte Tageszeiten begrenzt sein.

SAF übergibt der externen Security Engine die pertinente Information. Die Security Engine kann dann auf der Basis von „Profilen“ über die Zugriffsrechte entscheiden.

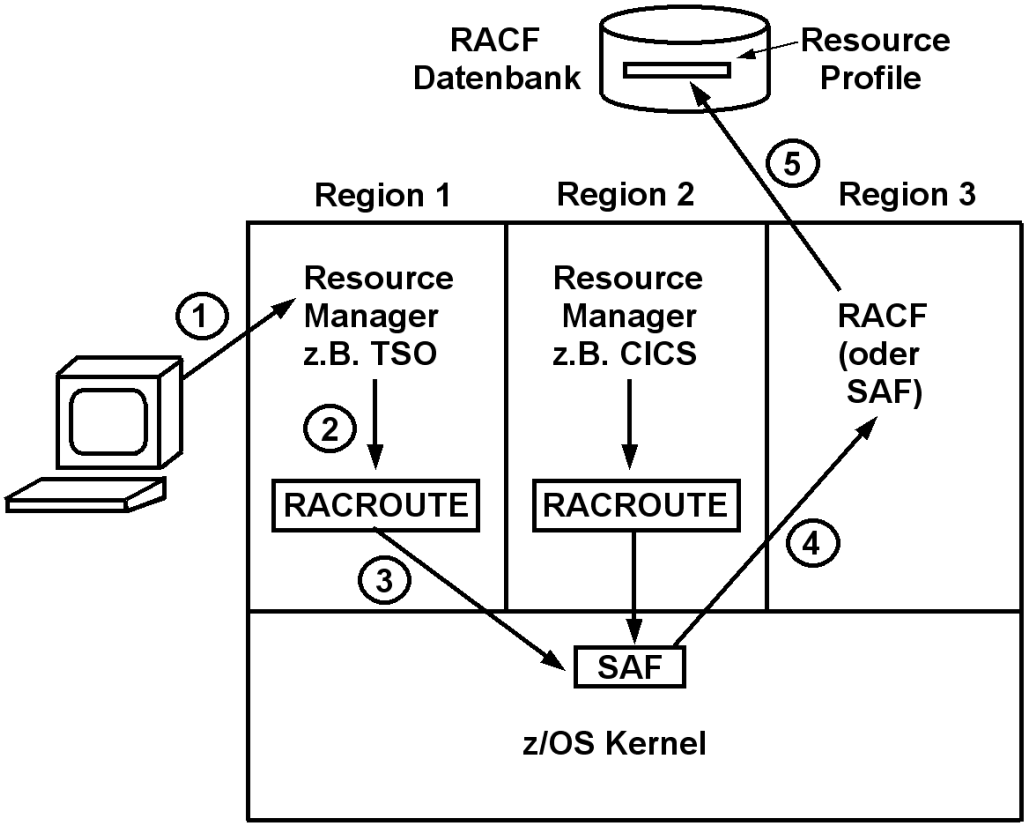


Abb. 3.4.9
b) RACF Arbeitsweise

Dies geschieht in den folgenden Schritten:

1. **Ein Benutzer greift auf eine Resource über einen Resource Manager zu, z.B. TSO**
2. **Der Resource Manager benutzt einen System Call „RACROUTE“ um auf die Security Access Facility (SAF) des z/OS Kernels zuzugreifen. SAF ist eine zentrale Anlaufstelle für alle sicherheitsrelevanten Aufgaben.**
3. **SAF ruft ein Zugriffskontrolle Subsystem auf. Dies ist normalerweise RACF.**
4. **RACF greift auf einen „Profile“ Datensatz in seiner eigenen RACF Datenbank zu und überprüft die Zugangsberechtigungen**
5. **Das Ergebnis teilt RACF dem anfragenden Resource Manager mit.**

3.5 Weiterführende Information

3.5.1 Mehr Details zum z/OS Betriebssystem

Unsere Lehrveranstaltung Enterprise Computing hat die Zielrichtung, das Wissen für die Entwicklung neuer z/OS Anwendungen sowie deren Integration in das Internet zu vermitteln. Wir beschränken uns dabei auf die hierfür erforderlichen z/OS Betriebssystem Grundlagen.

Systemadministratoren benötigen weitergehendes Wissen über das z/OS Betriebssystem sowie das Zusammenspiel mit den angeschlossenen Festplatten. Mehr Details zu z/OS sind in einem Vorlesungsscript enthalten, welches Prof. Spruth für eine Lehrveranstaltung 2009 an der IT Akademie Bayern verfasste.

Es ist verfügbar im pdf Format unter
http://www.cedix.de/bs/index.html

Information über die modernisierten z/OS Management Tools unter
www.cedix.de/VorlesMirror/Band1/Manage01.pdf

Benutzung von PHP unter z/OS
www.cedix.de/VorlesMirror/Band1/PHP01.pdf

3.5.2 JCL

Ramesh Krishna Reddy: JCL Tutorial:
http://www.mainframegurukul.com/srcsinc/drona/programming/languages/jcl/jcl.chapter1.html

(i) Introduction to JCL
http://www.cedix.de/Literature/Textbooks/jcl/jclintro.pdf

JCL Lehrbuch
http://www.cedix.de/Literature/Textbooks/jcl/JCLBuch.pdf

IBM: z/OS JCL Users Guide. SA22-7598-04, July 2004
http://www.cedix.de/Literature/Textbooks/jcl/MvsJclUserGuide.pdf

IBM: z/OS JCL Reference. GC28-1757-09 September 2000
http://www.cedix.de/Literature/Textbooks/jcl/MvsJclReference.pdf

3.5.3 Storage Management Subsystem

http://www.cedix.de/VorlesMirror/Band1/SMS01.pdf

enthält eine gute Übersicht über das Storage Management Subsystem

3.5.4 TSO/ISPF Benutzeroberfläche

Wenn Ihnen die TSO bzw. ISPF Benutzeroberfläche nicht gefällt, und Sie Eclipse mit RDz auch keinen Geschmack abgewinnen können:

Es existieren unzählige Software Produkte, welche mit „verbesserten“ Oberflächen arbeiten.

Ein Beispiel ist SELCOPYi, siehe

http://www.cbl.com/pdf/SELCOPYi_Brochure_zOS.pdf

oder Video Tutorials unter

http://www.cbl.com/cblidem.html

(läuft gut mit dem Windows Media Player).

3.5.5 Wie entwickelt man ein neues Betriebssystem ?

Zum Thema Software Engineering existieren Tonnen von Lehrbüchern und Fachaufsätzen.

Ein Lehrbuch schlägt alle anderen um Meilen.

z/OS begann Anfang 1966 unter dem Namen OS/360, als reines Stapelverarbeitungssystem. Es wurde von Fred Brooks entwickelt, und diente als Vorlage für sein 1975 erschienendes berühmtes Buch „The mythical Manmonth“ (http://en.wikipedia.org/wiki/The_Mythical_Man-Month).

Fred Brooks ist von der wissenschaftlichen Gemeinschaft auf Grund der von ihm vertretenen Thesen stark angefeindet worden. Heute gibt man ihm recht. Das Buch wird auch gerne als "The Bibel of Software Engineering" bezeichnet.

Die heute erhältliche Version des Buches enthält zusätzlich einen in der Zeitschrift IEEE Computer, vol. 20, no. 4, 1987 erschienenen Aufsatz „No Silver Bullet: Essence and Accidents of Software Engineering“ , verfügbar unter
http://www.cedix.de/Literature/Textbooks/NoSilverBullet.pdf

3.5.6 Wie war das noch mit den Lochkarten ?

Ein faszinierender Überblick über die Entwicklung der Lochkartentechnik ist zu finden in dem Buch

Günther Sandner, Hans Spengler:
Die Entwicklung der Datenverarbeitung von Hollerith Lochkartenmaschinen zu IBM Enterprise-Servern. ISBN-10: 3-00-019690-0, ISBM13: 978-3-00-019690-4.

Ein kostenloses Download ist verfügbar unter

http://www.cedix.de/Literature/History/SandnerSpengler.pdf

Zahlreiche Darstellungen von Lochkarten sind zu finden unter
http://www.google.de/search?q=punched+card&hl=de&lr=&as_qdr=all&prmd=imvns&tbm=isch&tbo=u&source=univ&sa=X&ei=wiqlUMu4FobV4QSk0oHICA&ved=0CCoQsAQ&biw=1516&bih=963

3.5.7 IMS

IMS Datenbanken werden nach wie vor sehr häufig eingesetzt. Eine Einführung (IMS Primer)in das IMS Datenbanksystem ist zu finden unter:

http://www.redbooks.ibm.com/redbooks/pdfs/sg245352.pdf

Eine andere, etwas kürzere Einführung:

http://www.mainframes360.com/2009/04/imsdb-tutorials.html

Bemerkung: Es existiert zusätzlich zu der IMS Datenbank (allgemeine Bezeichnung IMS DB) noch ein IMS Transaction Manager (allgemeine Bezeichnung IMS DC), der aber im Vergleich zum CICS Transaction Server keine große Bedeutung hat.

Hier zwei Video Tutorials

http://www.youtube.com/watch?v=Xs8nbkNSqdI

http://www.youtube.com/watch?v=lE7a0ZXzrow

IMS Zugriff über ein iPhone
http://imsmadesimple.tumblr.com/post/967487367/tutorial-access-ims-data-on-your-iphone

4. System z/Hardware

4.1 Microprocessor Technologie

4.1.1 Mainframe Alternativen

Hersteller	Name	Microprocessor	Betriebssystem
Fujitsu / Sun	Sunfire, M9000, M5-32	Sparc	Solaris
HP	Superdome	Itanium	HP-UX
IBM	System p	PowerPC	AIX

a) Abb. 4.1.1
Die wichtigsten Alternativen zu System z Mainframes

Die Firma Hewlett Packard (HP) vertreibt neben dem hauseigenen HP-UX (Unix) mehrere weitere Betriebssysteme, die zum Teil aus der Übernahme anderer Computer Firmen stammen:

- **MPE/iX ist eine HP-eigene Entwicklung, die 2010 eingestellt wurde.**
- **Tru64 UNIX wurde von der Firma Digital Equipment entwickelt und läuft auf "Alpha" Mikroprozessoren. Es wird nicht weiterentwickelt, und HP stellt die Unterstützung Ende 2012 ein.**
- **HP NonStop stammt von der Firma Tandem Computers und läuft auf Itanium Mikroprozessoren.**
- **Das Virtual Memory System (VMS) Betriebssystem wurde von der Firma Digital Equipment Corporation (DEC) entwickelt und läuft ebenfalls auf Itanium Mikroprozessoren. Microsoft benutzte VMS als Basis für die Entwicklung von Windows 2000.**

Mehrere Hersteller (Dell, HP, IBM, Unisys, andere) stellen große Konfigurationen mit x86 und Windows oder Linux als Alternative zu dem untersten Ende der System z Produktlinie her.

Die Firma Unisys produziert in kleinen Stückzahlen Großrechner mit dem OS2200 Betriebssystem , welches auf die UNIVAC 1100/2200-Serie zurückgeht, sowie Rechner mit dem Master Control Program (MCP) Betriebssystem, welches auf die Burroughs-B5000-Produktlinie zurückgeht.

Die französische Firma Bull S.A. produziert ebenfalls in kleinen Stückzahlen die „novascale gcos 7010 system" Rechner mit dem proprietären Betriebssystem GCOS (General Comprehensive Operating System), welches ursprünglich von General Electric in den USA entwickelt wurde.

Die meisten Implementierungen von betriebswirtschaftlichen Großrechnern verwenden die gleichen oder ähnlichen Technologien und Bausteine, wie sie auch für Arbeitsplatzrechner oder kleine Server eingesetzt werden. Dies hat den Vorteil, die Entwicklungskosten auf eine größere Stückzahl verteilen zu können. Es hat den Nachteil, dass die verwendeten Komponenten (Commodity Parts) nicht für den Einsatz in einem betriebswirtschaftlichen Großrechner optimiert wurden.

Zwei Beispiele sind:

- **Zuverlässigkeit und Verfügbarkeit richtet sich nach ökonomischen Kriterien , die für den PC Bereich etabliert werden. Durch zusätzliche Einrichtungen versucht man Verbesserungen zu erreichen, die dann aber ins Geld gehen.**
- **Ein/Ausgabe Einrichtungen (Input/Output, I/O) sind z.B. für den Anschluss von 5 Plattenspeichern optimiert, nicht aber für 5 000 oder 50 000 Plattenspeicher**

Es werden aber zahlreiche zusätzliche Komponenten benötigt, deren Entwicklung einen erheblichen Aufwand erfordert. Als Folge sind betriebswirtschaftlichen Großrechner alles andere als billig, wie das folgende Beispiel eines führenden Herstellers, der Firma Sun, zeigt.

4.1.2 Großrechner der Firmen Sun/Oracle und Hewlett Packard

Abb. 4.1.2
Minimal Konfiguration des Sun 25 K Servers

Die Firma Sun verkündet stolz, dass eine Minimalkonfiguration ihres Großrechners für wenig mehr als 1 Million $ zu haben ist.

Das kleinste Modell der Sun 25k Serie hatte 4 „System Boards“ mit je 4 Dual Core SPARC CPUs, oder insgesamt 16 CPUs. Maximal sind 16 bzw. 18 System Boards möglich. Spätere Modelle benutzten Quad Core CPU Chips. Die E25K System Boards (auch als Prozessor Boards bezeichnet) sind eine evolutionäre Weiterentwicklung der System Boards in der Sun 15k und Sun 10k Serie, welche single Core CPU Chips benutzten.

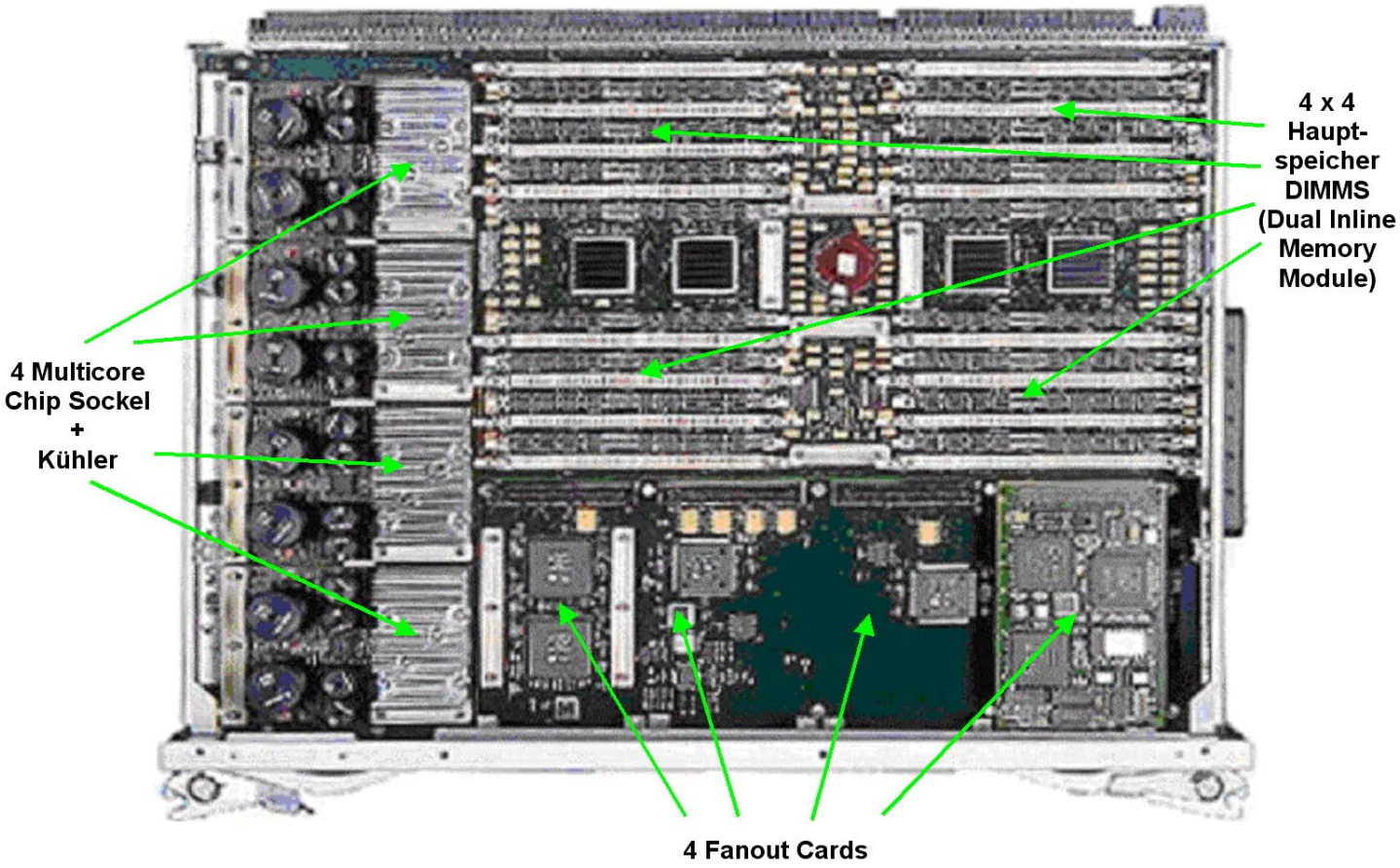


Abb. 4.1.3
b) Sun Fire 15K System Board

Die Sun System Boards werden auch in anderen Produkten der Firma Sun eingesetzt, z.B. in einer Low End Workstation mit einem einzigen System Board.

Die Firma Sun hat ihre Produkte kontinuierlich weiterentwickelt, wobei sich vor allem die Anzahl der CPU Cores pro CPU Chip gewachsen ist. Die Folge Modelle wurden als M9000 bezeichnet, und gemeinsam von Sun und von Fujitsu/Siemens vertrieben. Die neueste Version läuft unter dem Namen SPARC M5-32 Server.

Abb. 4.1.3 zeigt ein Sun System Board

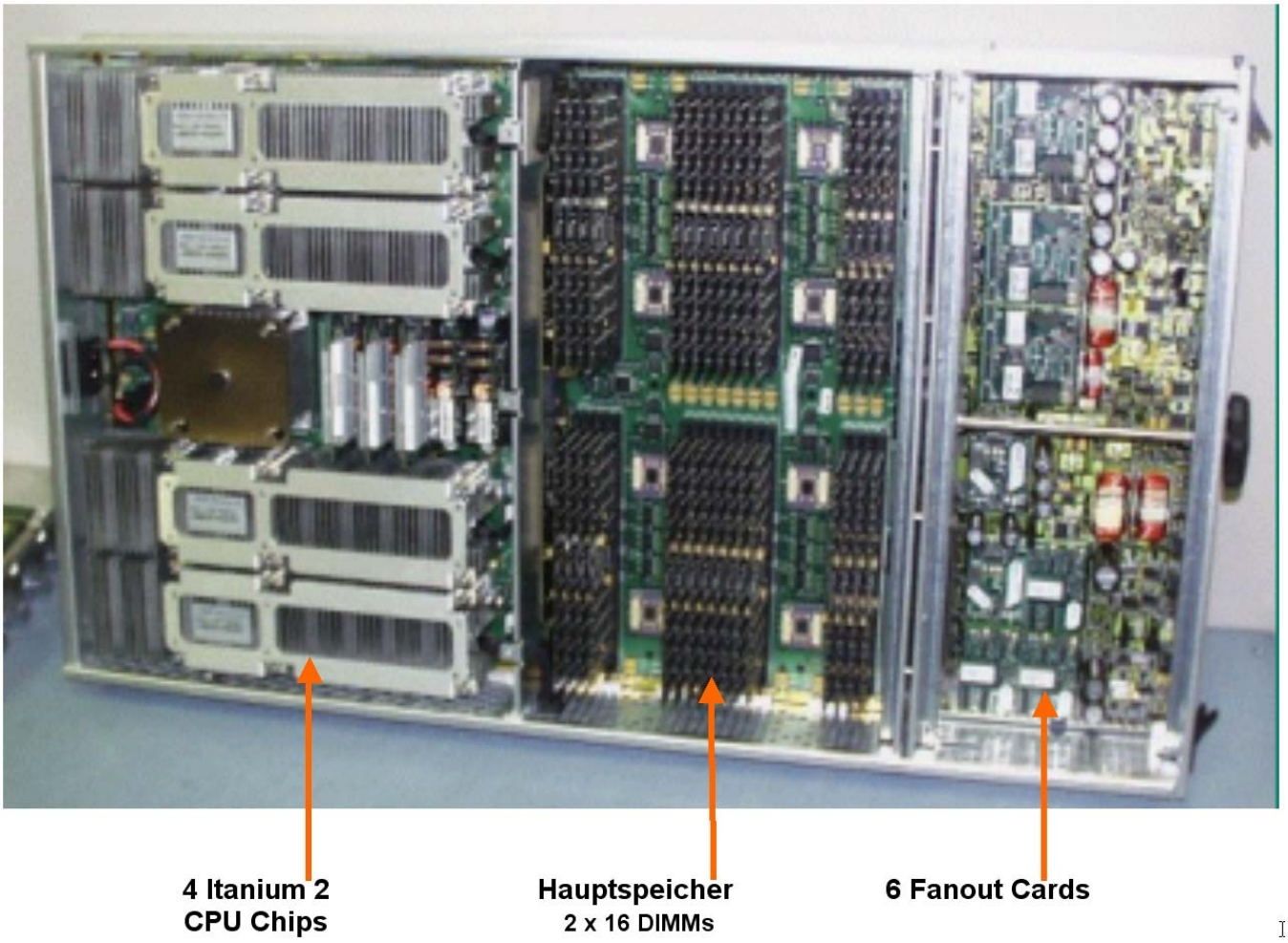


Abb. 4.1.4
Hewlett-Packard Superdome Cell Board

Hewlett Packard (HP) bezeichnet sein Prozessor Board als „Cell Board“. Ansonsten hat es sehr viel Ähnlichkeit mit dem System Board von der Firma Sun.

Die System- bzw. Cell Boards haben keinen Platz für den Anschluss von I/O Geräten, besonders Plattenspeichern, von denen evtl. Hunderte oder mehr angeschlossen werden müssen. Statt dessen existieren Fanout Cards, die mittels DMA (Direct Memory Access) auf den Hauptspeicher zugreifen können.

Die Fanout Cards sind über Kabel mit PCIe Card Steckplätzen auf einem „I/O Cage“ Board verbunden. Am häufigsten sind PCIe Adapter Karten für Plattenspeicher Anschlüsse mittels Fibre Channel SCSI Kabeln anzutreffen. Auf diese Art ist es möglich, Hunderte oder mehr Plattenspeicher an einen SUN oder HP Großrechner anzuschließen.

Spezifisch verfügen beide Processor Board Typen über Anschlüsse für 4 bzw. 6 Fanout Adapter Cards, die als Daughter Cards auf das Processor Board aufgesteckt werden. Typischerweise verwendet man für diese Slots PCIe Adapter Cards. Eine derartige PCIe Adapter Card verbindet das Prozessor Board über ein PCIe Kabel mit einem PCIe Board, welches eine Reihe von PCIe Karten-Slots verfügt.

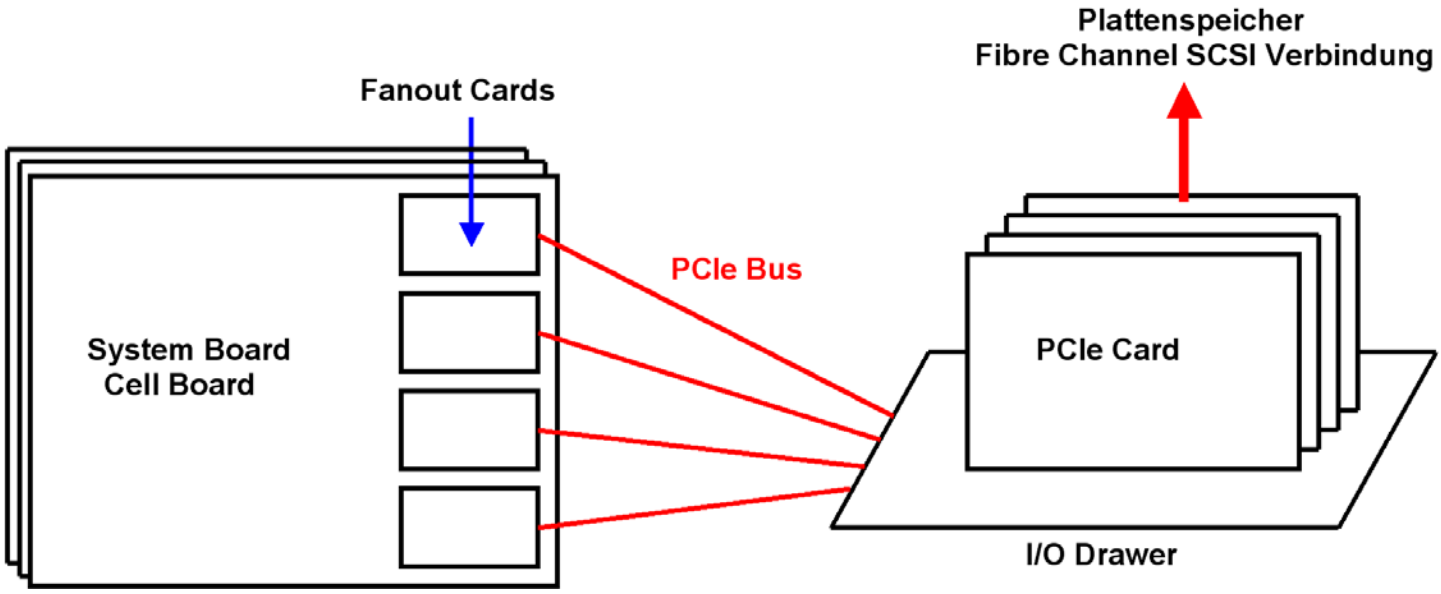


Abb. 4.1.5
c) Sun Fire, Superdome Konfiguration

System z verwendet an Stelle von Prozessor Boards sog. „Books“. Das Äquivalent zur Fanout Adapter Card wird als HCA (Host Channel Adapter) Card bezeichnet.

Sun (mit dem Co-Operationspartner Fujitsu), Hewlett Packard sowie IBM sind die drei führenden Hersteller von betriebswirtschaftlichen Großrechnern. Einige weitere Hersteller (z.B. Bull mit dem novascale gcos 9010 System oder Unisys mit dem ClearPath System) spielen eine eher untergeordnete Rolle.

Alle diese Hersteller außer IBM setzen aus der PC-Welt abgeleitete Technologien für ihre Processor Boards ein. Besonders verwenden die Prozessor Boards die gleich Printed Circuit Board (PCB) Technologie, die auch für die Mainboards der PCs benutzt wird.

Printed Circuit Boards bestehen aus einem elektrisch isolierenden Trägermaterial (Basismaterial), auf dem Kupferschichten aufgebracht sind. Die Schichtstärke beträgt typischerweise 35 µm Das Basismaterial besteht meistens aus mit Epoxydharz getränkte Glasfasermatten. Die Bauelemente werden in der Regel auf Lötflächen (Pads) aufgelötet. Multilayer Boards können aus bis zu 48 Schichten bestehen. PC Mainboards haben in der Regel unter 10 Schichten.

4.1.3 Mainframe Chip Technologie

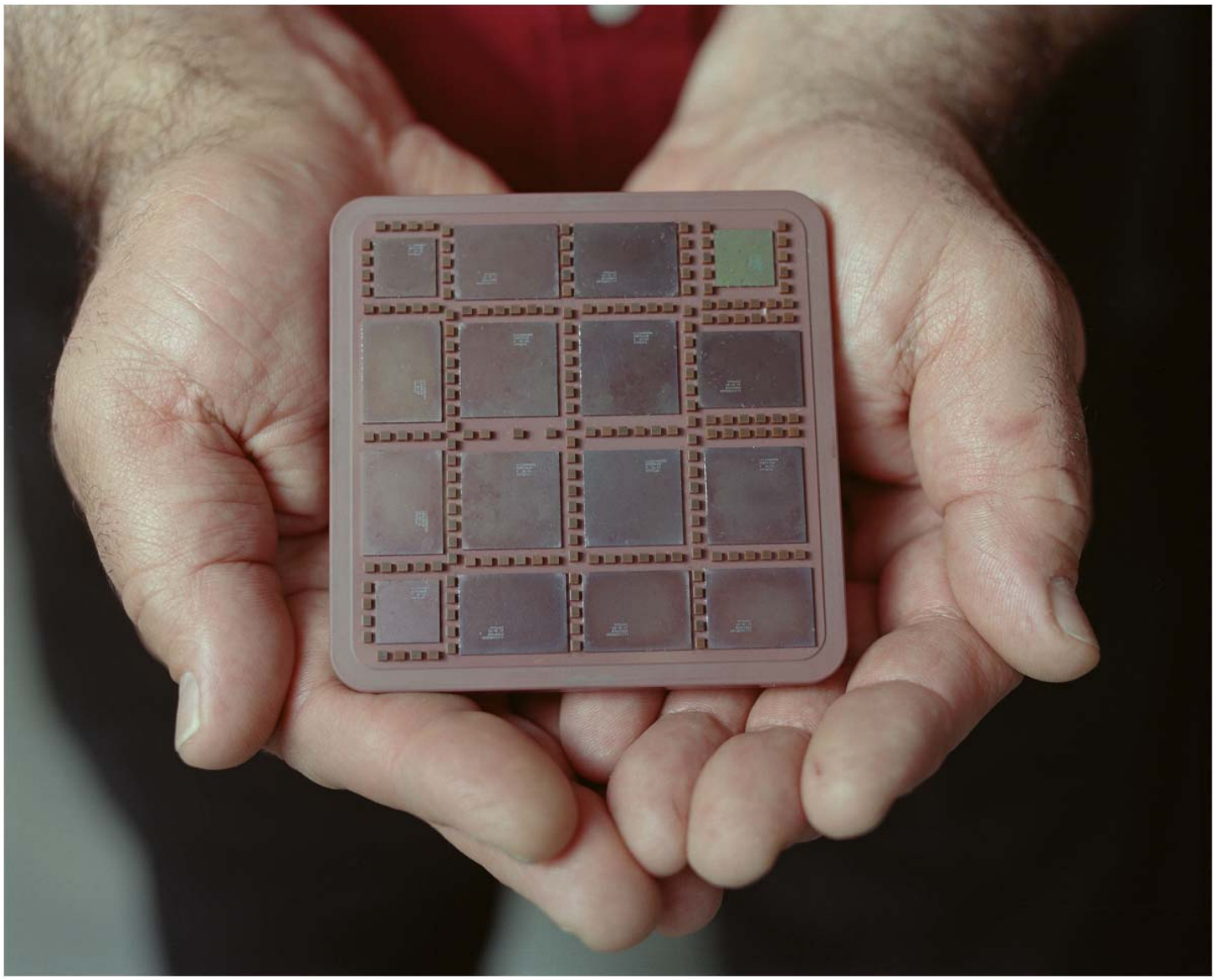


Abb. 4.1.6
Multi-Chip Module eines z9 Rechners

In der Abb. 4.1.6 ist ein MCM (Multi Chip Modul), das Kernstück eines z9-Rechners gezeigt. Das MCM besteht aus einem 95 x 95 mm großem Multilagen-Glas-Keramik-Träger. Auf dem Glas-Keramik-Modul sind 16 Chips aufgelötet

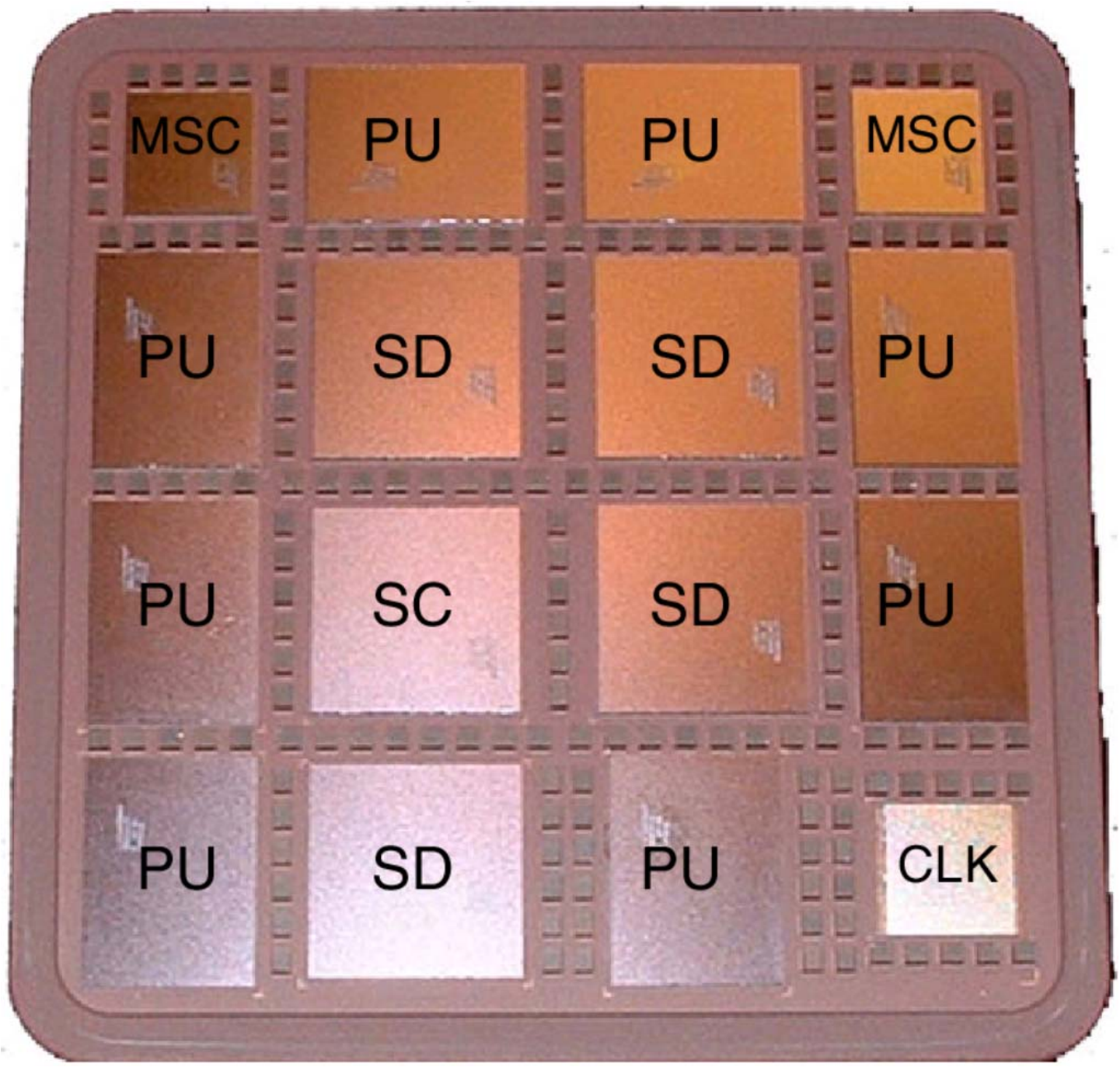


Abb. 4.1.7
Detailansicht des z9 Multichip Modules

Ab. 4.17 zeigt ein Multi Chip Module (MCM) Module, das Kernstück eines z9-Rechners. Auf dem Multi Chip Module befinden sich:

- **8 dual Core CPU Chips (labeled PU), insgesamt 16 CPU Cores,**
- **4 Level 2 (L2) Cache Chips labeled SD,**
- **1 L2 Cache Controller Chip labeled SC,**
- **2 Hauptspeicher Controller Chips labeled MSC,**
- **ein Clock Chip (CLK).**

Das z9 Multi Chip Module (MCM) benutzt eine Multilayer Ceramic (MLC) Technologie. Es besteht aus einem 95 x 95 mm großem Multilagen-Glas-Keramik-Träger mit 102 Verdrahtungslagen. MLC ermöglicht im Vergleich zur Printed Circuit Board Technologie besonders günstige Signallaufzeiten zwischen den Chips. Der Grund für die günstigeren Signallaufzeiten ist:

- **kleinere Abstände zwischen den Chips**
- **günstigere Dielektrizitätskonstante von MLC im Vergleich zu zur Printed Circuit Technologie**

Der Nachteil ist ein schwierigerer Produktionsprozess, den derzeitig in der Computerindustrie nur IBM einsetzt. Mehrere Unternehmen stellen MLC Substrate für Microwave Anwendungen her, z.B. http://www.ltcc.de/downloads/rd/pub/10-doc-plus-engl-2001.pdf

4.1.4 Eigenschaften des zEC12 CPU Chips

IBM bringt verbesserte Mainframe Modelle etwa im 2 ½ Jahres Rhythmus heraus. Die vier letzten Modelle sind:

Modell z9	**vertrieben seit Juli 2005**
Modell z10	**vertrieben seit Februar 2008**
Modell z196	**vertrieben seit Juli 2010**
Modell zEC12	**vertrieben seit August 2012**

Die neuen Modelle beinhalten in der Regel zahlreiche Verbesserungen, u.A. auch in der Halbleiter Technologie. Die MCM Technologie ist dagegen stabil und ändert sich nur wenig. Wir sehen deshalb auf dem MCM für das Modell zEC12 nur weniger, dafür aber größere Chips, wobei sich an den Abmessungen des Modules kaum etwas ändert.

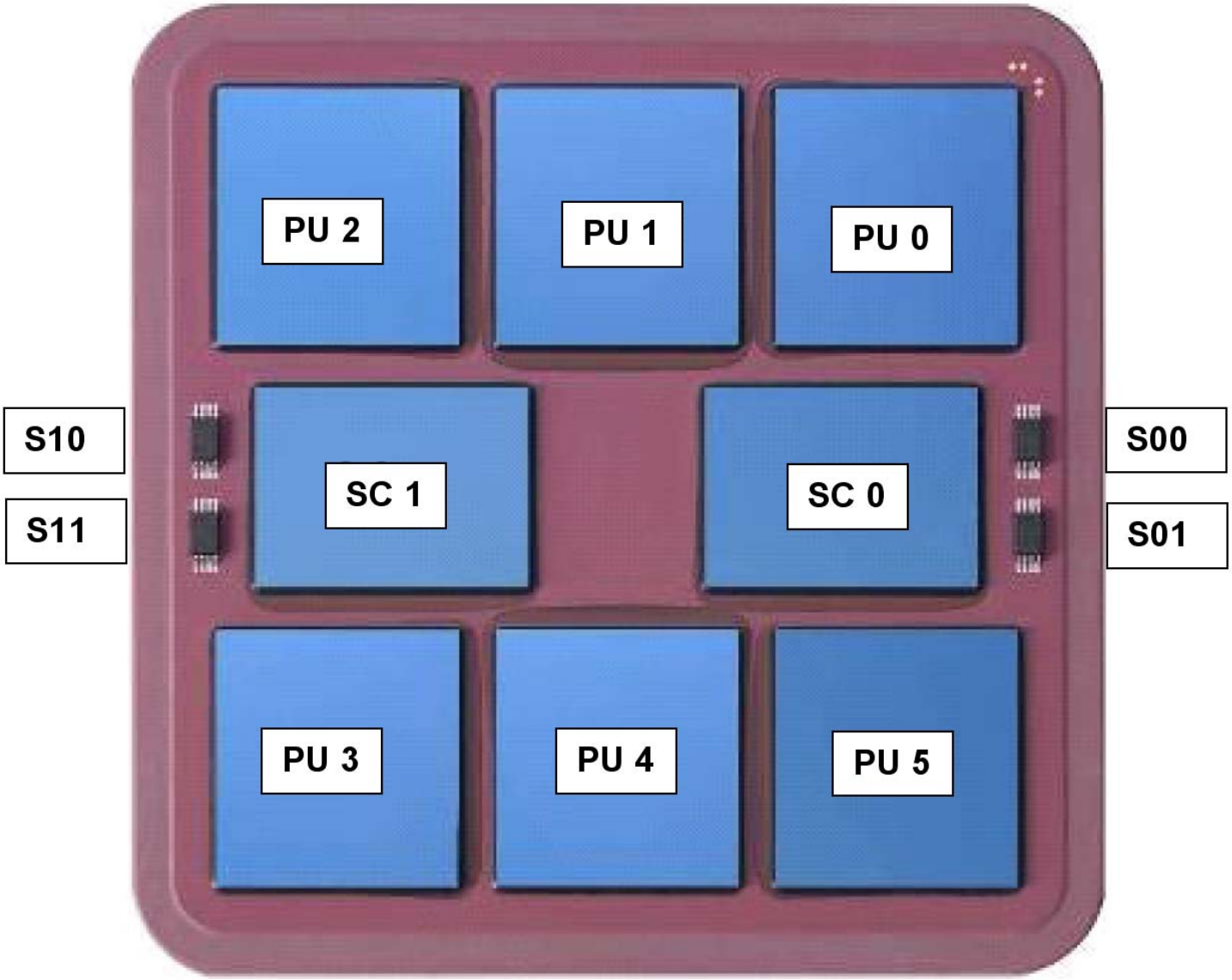


Abb. 4.1.8
zEC12 CPU Chip

Auf dem zEC12 Multi Chip Module (MCM) befinden sich:

- **6 Hex core CPU Chips (labeled PU), insgesamt 36 CPU Cores,**
- **2 L4 Cache Chips labeled SC,**
- **4 EPROM chips labeled S00, S01, S10 und S11. Sie dienen der Personalisierung (characterization) jedes einzelnen MCMs.**

Das MCM des Modells zEC12 besteht aus einem 96 x 96 mm großem Multilagen-Glas-Keramik-Träger mit 103 Verdrahtungslagen.

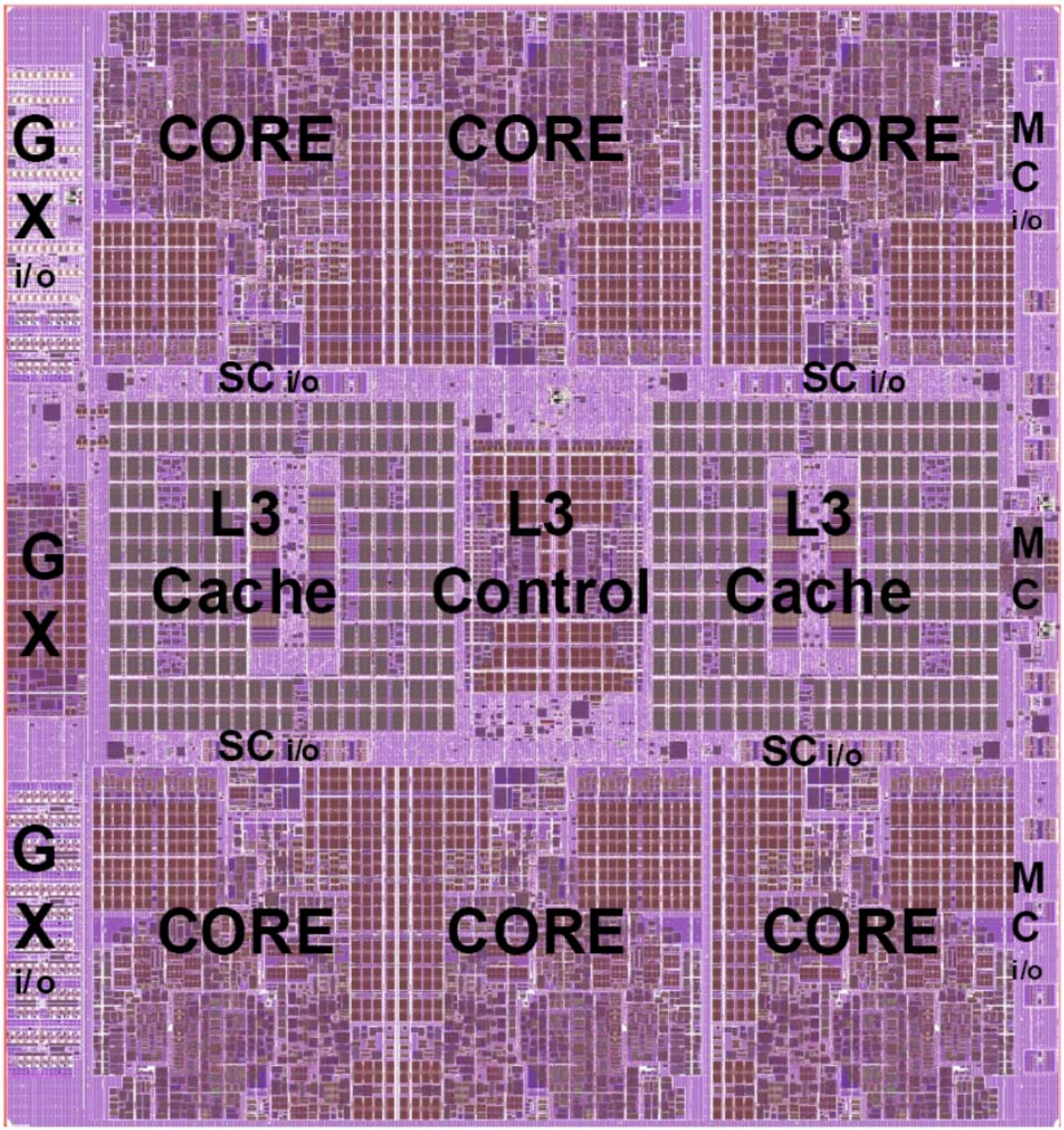


Abb. 4.1.9
Layout des zEC12 CPU Chips

Auf dem zEC12 CPU Chip befinden sich 6 CPU Cores. Jeder der sechs Cores hat einen eigenen L1-Cache mit 64 KByte für Befehle und 96 KByte für Daten. Neben jedem Core befindet sich ein privater L2-Cache mit 1 MByte für Befehle und 1 MByte für Daten.

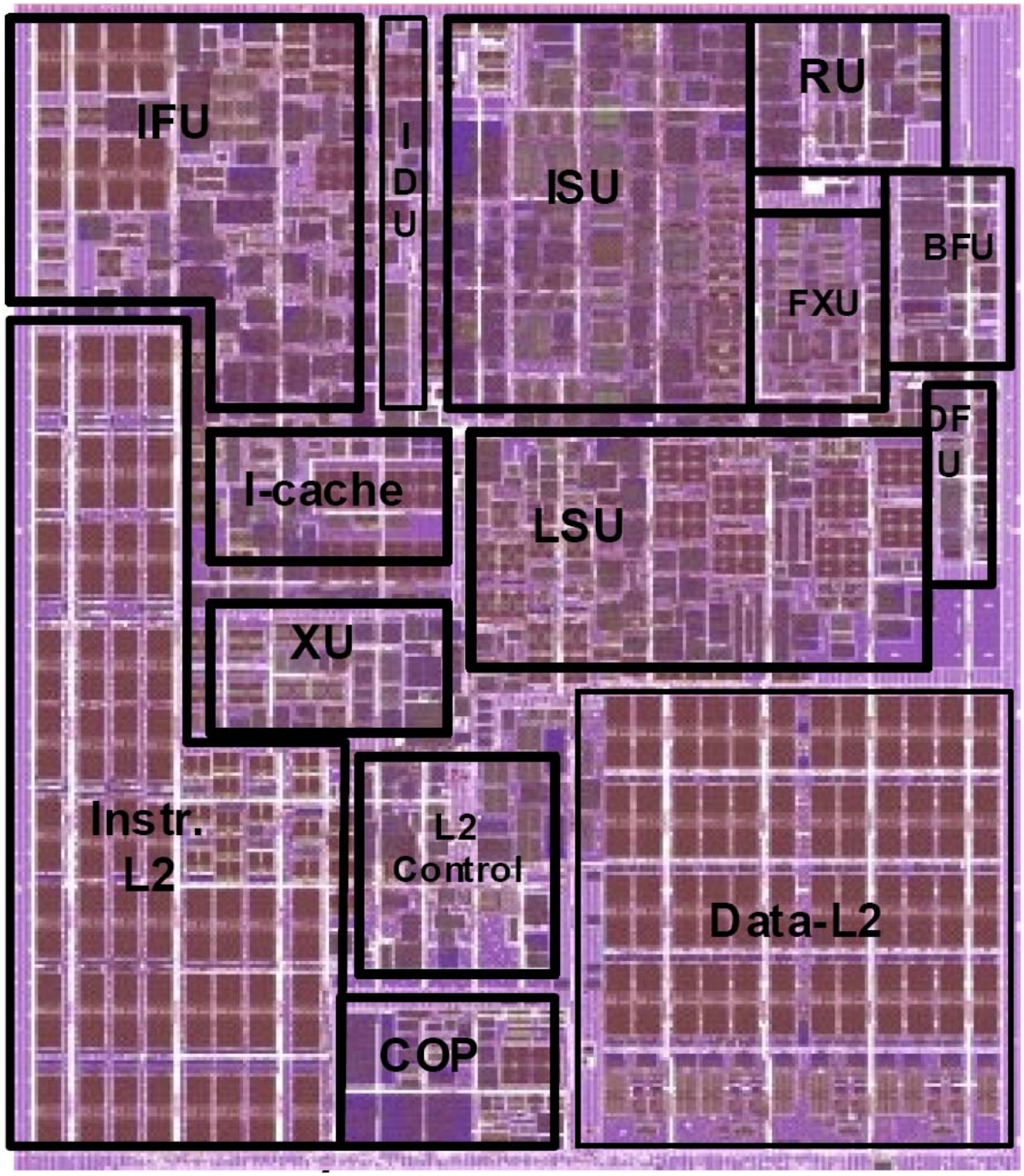


Abb. 4.1.10
Layout eines der 6 CPU Cores

Weiterhin befindet sich auf dem zEC12 CPU Chip ein L3-Cache mit 48 MByte, bestehend aus 2 Compartments. Der L3-Cache ist ein Store-in-Cache Speicher, der von allen Cores des CPU Chips gemeinsam genutzt wird. Die L3 Control Einheit verfügt über einen integrierten On-Chip-Kohärenz-Manager und einen Crossbar-Switch. Beide L3 Compartments können gleichzeitig bis zu 160 GByte/s Bandbreite an jeden Core übertragen. Der L3-Cache verbindet die sechs Kernen, GX I/O-Busse und Hauptspeicher-Controller (MCs) mit getrennten Hauptspeicher Controller (SC)-Chips. Die Hauptspeicher-Controller (MC) Funktion steuert weiterhin den Zugriff auf den Hauptspeicher. Der GX I/O-Bus steuert die Schnittstelle zu den Host Channel Adapter (HCA), welche die Verbindung zu getrennten I/O Adaptern herstellen. Insgesamt steuert das zEC12 CPU Chip den Datenverkehr zwischen den CPU Cores, I/O sowie dem L4-Cache auf dem SC-Chips.

Jedes CPU Chip hat 2,75 Milliarden (10^9) Transistoren. Die Abmessungen sind 23,5 x 21,8 mm. Es implementiert 6 CPU Cores und eine 4-stufige Cache Hierarchie, die aus L1, L2, L3 und L4 Caches besteht (näheres dazu später).

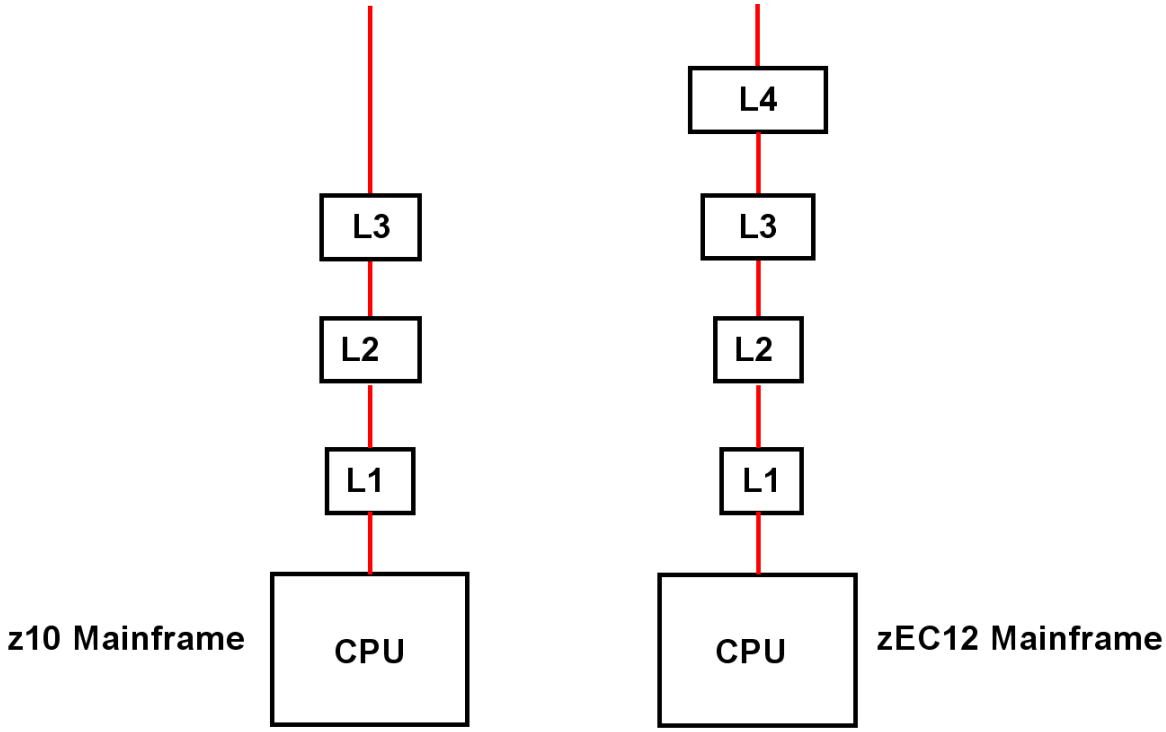
Im Vergleich dazu hatte die 1988 erschienene CMOS Implementierung eines S/370 Prozessors 200 000 Transistoren.

Der Energieverbrauch beträgt etwa 300 Watt pro zEC12 CPU Chip.

4.1.5 eDRAM

Ein Static Random Access Memory (SRAM) benötigt zwischen 4 – 10 Transistoren in einer Flip-Flop Schaltung für jedes gespeicherte Bit. Ein Dynamic Random Access Memory (DRAM) benötigt 1 Transistor und einen kleinen Kondensator für jedes gespeicherte Bit. Ein DRAM Speicher kann auf einer gegebenen Chip Fläche wesentlich mehr Bits unterbringen als ein SRAM Speicher, ist dafür aber wesentlich langsamer (z.B. 100 ns versus 1 ns Zugriffszeit).

Die Hauptspeicher nahezu aller Rechner verwenden fast immer DRAMS, häufig in der Form von SIMM oder DIMM Steckkarten. Cache Speicher werden fast immer in SRAM Technologie implementiert.



d) Abb. 4.1.11
Heutige Mainframes haben eine drei- oder vier-stufige Cache Hierarchie.

Die L3 und L4 Caches eines z196 Rechners werden in einer neuartigen eDRAM (embedded DRAM) Technologie implementiert. eDRAM ist nahezu so schnell wie SRAM, benötigt aber viel weniger Platz.

Ein embedded DRAM (eDRAM) ist ein auf DRAM basierender eingebetteter Speicher. Das bedeutet, das DRAM ist auf dem gleichen Chip wie der Mikroprozessor (die CPU) integriert (eingebettet). Im Gegensatz zu externen DRAM-Speichermodulen wird er häufig wie ein transistorbasiertes SRAM als Cache genutzt.

Das Einbetten des Speichers ermöglicht gegenüber externen Speichermodulen die Nutzung größerer Busse und höhere Arbeitsgeschwindigkeiten. Durch den geringeren Platzbedarf ermöglicht DRAM verglichen mit SRAM eine höhere Datendichte: somit kann bei gleicher Chip-Größe potentiell mehr Speicher genutzt werden. Jedoch machen die Unterschiede im Herstellungsprozess zwischen DRAM und Transistorlogik die Integration auf einem Chip kompliziert, das heißt, es sind in der Regel mehr Prozessschritte notwendig, was die Kosten erhöht.

eDRAM wird nicht nur als L3- und L4-Cache-Speicher im zEC12 Rechner, sondern auch in vielen Spielkonsolen genutzt, z.B. Xbox 360 von Microsoft, Wii von Nintendo und PlayStation 3 von Sony.

4.1.6 Sechs oder acht Cores pro Chip ?

Ein Blick auf das Layout eines Cores des zEC12 CPU Chips in Abb. 4.1.10 zeigt, dass die vier L2 Caches mit 4 x 1,5 MByte Speicherkapazität in etwa den gleichen Platz in Anspruch nehmen wie der gemeinsam genutzte L3 Cache mit insgesamt 24 MByte Speicherkapazität.

Weiterhin zeigt das zEC12 Chip Layout, dass die 6 CPU Cores (ohne L2 Cache) deutlich weniger als 50 % der Chip Fläche in Anspruch nehmen. Es wäre denkbar gewesen, auf dem CPU Chip 8 Cores unterzubringen.

Bei der Firma IBM entwickelt die gleiche Mannschaft neue PowerPC und System z CPU Chips. Nicht überraschend weisen beide CPU Core Implementierungen viele Gemeinsamkeiten auf. Die neuste Version des PowerPC Mikroprozessors wird als Power7 bezeichnet.

Während man sich beim Power7 für ein 8 Core Chip entschieden hat, ist das System z Team bei 6 Cores geblieben, um dafür eine sehr komplexe Cache Hierarchie mit maximalen Cache Größen unterzubringen.

Eine ähnliche Entscheidung hat man bei der Firma Sun/Oracle getroffen

Das neue (2013) T5 Sparc Chip der Firma Sun/Oracle hat 16 CPU Cores und 8 MByte L3 Cache. Eine Variante, das M5 Sparc Chip, benutzt identische CPU Cores, hat aber nur 6 an Stelle von 16 Cores. Der freiwerdende Platz wurde benutzt, um den L3 Cache von 8 MByte auf 32 MByte zu vergrößern.

Das T5 Chip ist für High CPU Performance Server vorgesehen, während das M5 Chip für betriebswirtschaftliche Großrechner vorgesehen ist, die evtl. mit Mainframes konkurrieren sollen.

Das M5 Chip wird in dem derzeitigen Spitzenprodukt, dem Sparc M5-32 Server eingesetzt, Nachfolger des bisherigen M9000 Servers. Ein M5-32 Server hat 32 M5 CPU Chips, oder bis zu 192 CPU Cores.

Einzelheiten unter:

http://www.oracle.com/technetwork/server-storage/sun-sparc-enterprise/documentation/o13-024-m5-32-architecture-1920556.pdf?ssSourceSiteId=ocomen

Wir werden in der Zukunft öfters Diskussionen erleben, ob mit wachsender Integrationsdichte es besser ist, die Anzahl der Cores zu vergrößern, oder mehr Platz für Cache Speicher zur Verfügung zu stellen.

4.1.7 Zusätzliche Eigenschaften des System z CPU Chips

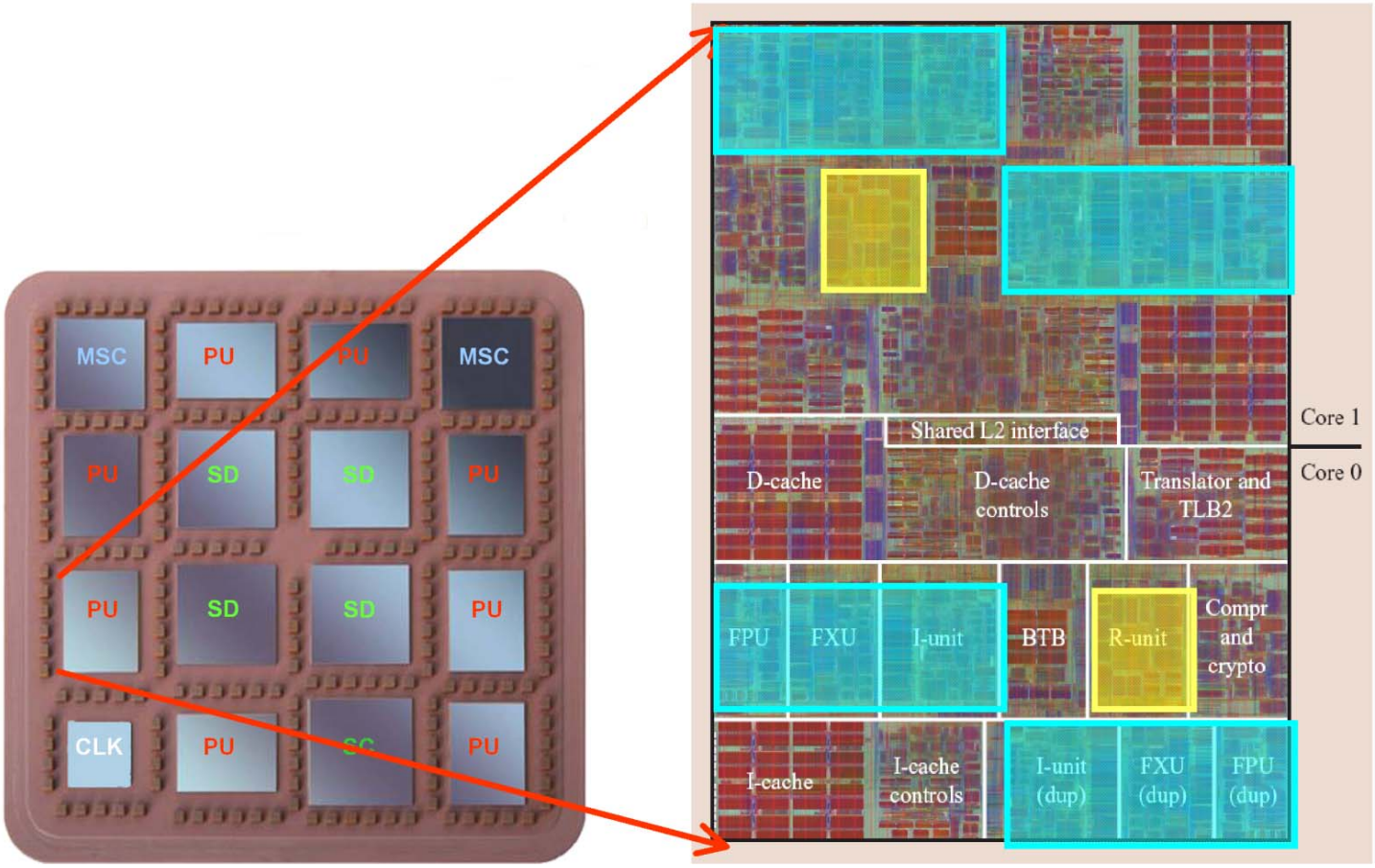


Abb. 4.1.12
Zusätzliche Baugruppen aus Zuverlässigkeitsgründen

Im Vergleich zum Power PC, x86, Sparc oder Itanium Chip wird bei System z ein sehr viel höherer Aufwand für Sicherheit und Verfügbarkeit getrieben. Dies sei am Beispiel des in Abb. 4.1.12 dargestellten z9 CPU Chips erläutert:

Es handelt sich um ein Dual Core Chip. Die obere und die untere Hälfte stellen je ein Core da. In der Mitte befindet sich die Schnittstelle zum L2 Cache (getrennte Chips auf dem gleichen MCM), die von beiden Cores gemeinsam genutzt wird.

Die wichtigsten Elemente in jedem Core sind die Instruction Unit (I-Unit), Fixed Point Execution Unit (FXU) und Floating Point Execution Unit (FPU). Jedes Core enthält alle drei Units in zweifacher Ausführung. Maschinenbefehle werden unabhängig und (nahezu aber nicht exakt in parallel) auf beiden Kopien der I-, FXU- and FPU Units ausgeführt. Mittels einer Compare Funktion wird verifiziert, dass beide Kopien das gleiche Ergebnis erzeugen. Wenn nicht, greifen automatische Fehlerbehebungsmaßnahmen ein, z.B. eine Maschinenbefehlswiederholung (instruction retry). Dies geschieht unbemerkt vom Betriebssystem oder Benutzerprogramm.

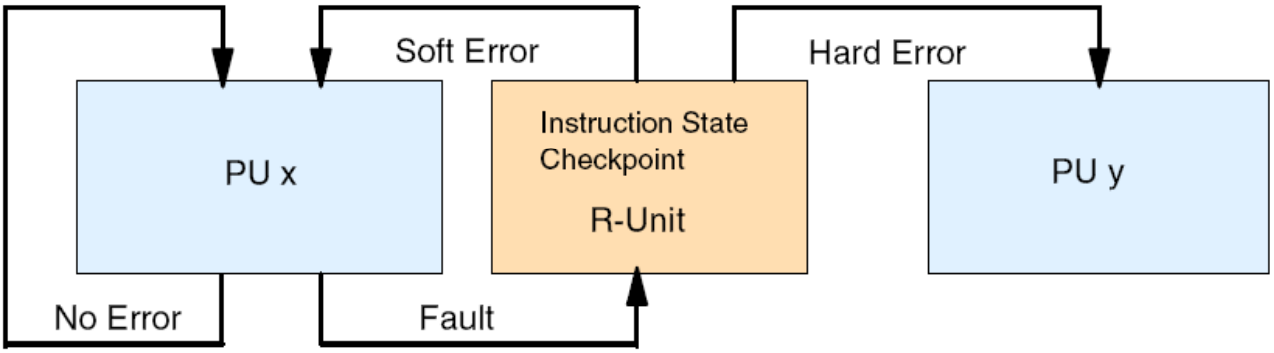


Abb. 4.1.
Recovery Unit

Von besonderem Interesse ist in diesem Zusammenhang die „Recovery Unit“ (RU). Wenn während der Ausführung eines Maschinenbefehls ein Fehler entdeckt wird, versucht die Instruction Unit des CPU Cores den Befehl ein zweites Mal auszuführen, in der Hoffnung, dass diesmal kein Fehler auftritt. Wenn eine Maschinenbefehlswiederholung nicht erfolgreich ist (z.B. ein permanenter Fehler existiert), wird ein Relocation Process gestartet. Dieser bewirkt, dass die Prozessausführung auf einem anderen CPU Core fortgesetzt wird. Das ist möglich, weil in jedem Augeblick den vollständigen Architekturstatus der CPU in ihrer R-Unit zwischenspeichert wird, wobei diese Zwischenspeicherung wiederum über Hamming Fehlerkorrekturcodes abgesichert ist.

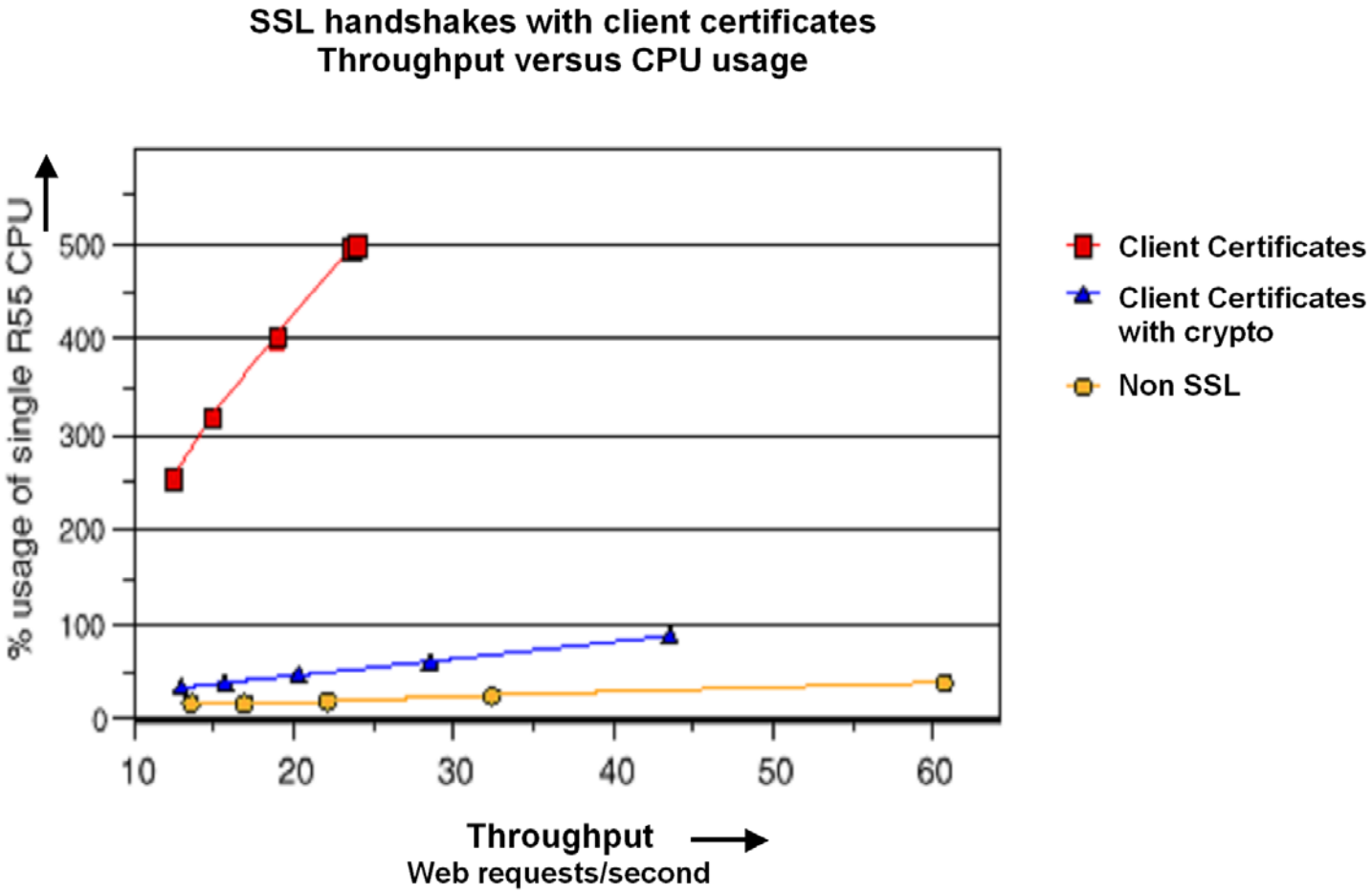


Abb. 4.1.13
Die Crypto Unit verbessert SSL Kommunikation

Der z9 Rechner hat eine Compression und Crypto Unit, beim zEC12 als Coprozessor bezeichnet. Die Crypto Unit wird u.a. eingesetzt, um die Verschlüsselung und Entschlüsselung von SSL (Secure Socket Layer) Nachrichten zu beschleunigen.

Die obige Abbildung demonstriert den Nutzen der Crypto Unit. Angenommen ist eine 6-Prozessor Einheit, die eine theoretische Leistung von 600 % verglichen mit einem einzelnen Prozessor erbringen kann.

Die gelbe Kurve stellt die CPU Auslastung für eine bestimmte Art von Web Requests pro Sekunde dar. Ohne SSL beträgt die Auslastung bei 60 Transaktionen/s weniger als 50 % der Leistung einer einzigen CPU.

Beim Einsatz von SSL, aber ohne Crypto Unit (rote Kurve) steigt die CPU Auslastung schon bei 25 Transaktionen/s auf 500 % (fünf CPUs).

Wird die Crypto Unit für die Verschlüsselung eingesetzt, entsteht die blaue Kurve. Die CPU Auslastung ist zwar deutlich höher als ohne SSL aber deutlich besser als SSL ohne Crypto.

Die hier wiedergegebenen Messdaten verwenden eine sehr einfache Web Request, die selbst nur wenig CPU Auslastung bewirkt. Bei komplexeren Web Requests ist der Unterschied weniger dramatisch.

4.1.8 Cache Struktur eines zEC12 Multichip Modules

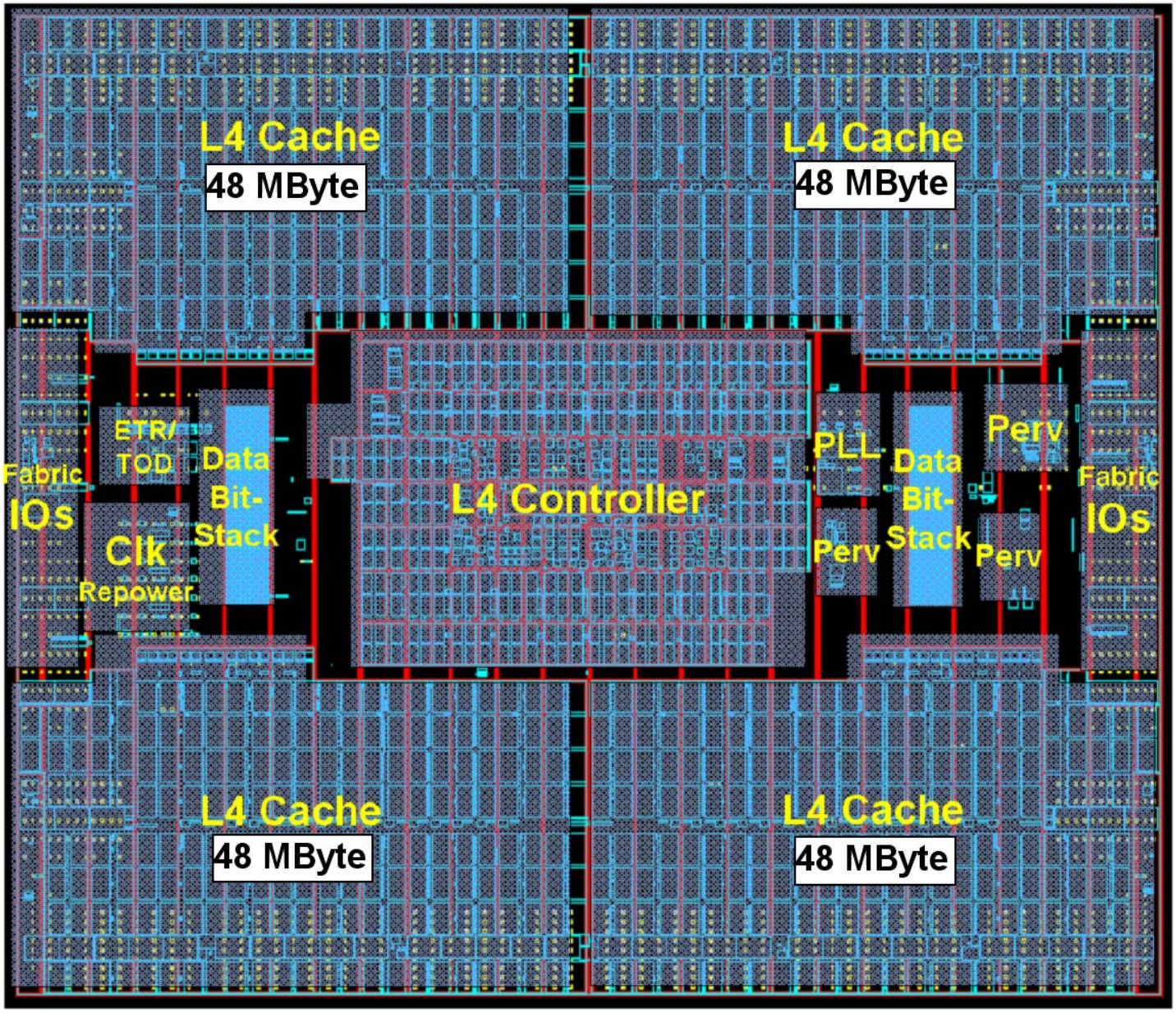


Abb. 4.1.14
zEC12 L4 Cache Chip

Neben den sechs CPU Chips befinden sich auf dem zEC12 MultiChip Module (MCM) noch zwei der hier gezeigten L4 Cache (SC) Chips. Jedes Chip hat Abmessungen von 28.4 x 23.9 mm, enthält 3.3 Milliarden (10^9) Transistoren und 2,1 Milliarde dynamische Speicherzellen (eDRAM). Neben dem Cache Controller speichert es 192 MByte.

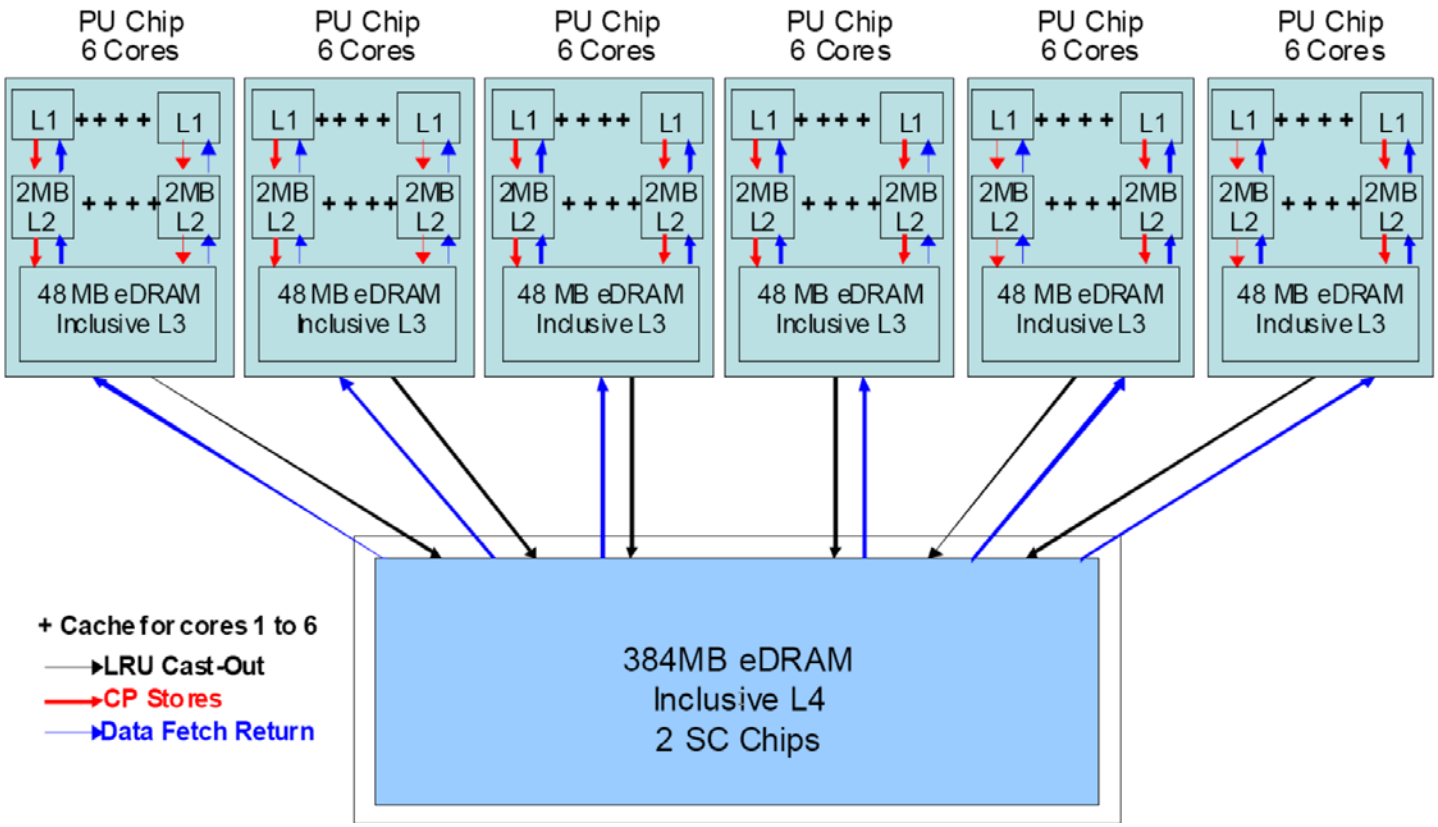


Abb. 4.1.15
zEC12 Cache Hierarchy

Die System z Prozessoren verwenden eine 4-stufige Cache Hierarchie.

Auf dem CPU Chip befinden sich 6 Cores. Jeder Core hat seine eigenen privaten L1 und L2 Cache. L1 und L2 verwenden unterschiedliche Technologien und haben unterschiedliche Zugriffszeiten.

Alle 6 Cores des CPU Chips verwenden einen gemeinsam genutzten (shared) L3 Cache.

Die 6 CPU Chips eines Multichip Modules verwenden einen gemeinsam genutzten L4 Cache.

Das MCM ist Bestandteil einer als „Book“ bezeichneten Baugruppe. Ein zEC12 Rechner enthält bis zu 4 derartiger Books und damit 4 MCMs. Die vier L4 Caches der vier MCMs sind miteinander verbunden und bilden einen von allen CPU Cores gemeinsam genutzten NUMA (Non-Uniform Memory Architecture) L4 Cache.

Ein NUMA Speicher ist dadurch gekennzeichnet, dass die Zugriffszeiten zu Teilen des Speichers unterschiedlich sein können.

4.2 Multichip Module

4.2.1 Wachstum der Chip Größe und Komplexität

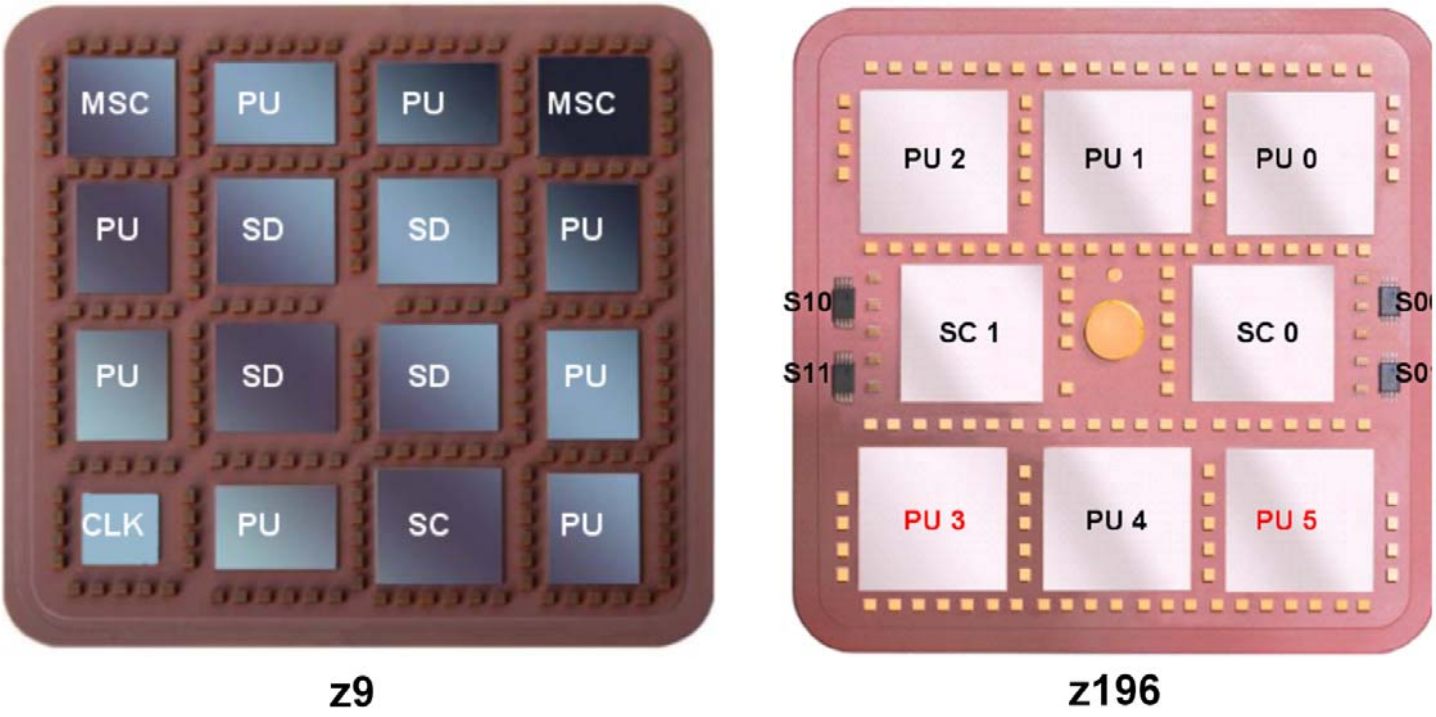


Abb. 4.2.1
Vergleich der z9 und z196 Multichip Mulitilayer Ceramic Module

Die Abmessungen und die Anzahl der Verdrahtungsebenen der beiden Module sind praktisch identisch. Die Anzahl der Chips ist jedoch halbiert und die Chips sind größer geworden. Die CPU Chips (PU) haben zwei (z9) bzw. vier (z196) Cores, und der gemeinsame Cache (SD/SC) wuchs von 40 (z9) auf 196 (z196) MByte.

IBM bringt etwa 2 ½ Jahre verbesserte Mainframe Modelle heraus. So erschien das Modell z9 im Juli 2005, Modell z10 im Februar 2008, Modell z196 im Juli 2010 und Modell zEC12 im Juli 2012. Bei unserem Mainframe Rechner an der Uni Leipzig handelt es sich um ein Modell z9 .

4.2.2 z9 Multi-Chip Module

Abb. 4.2.2
Halterung des z9 Multichip Modules

Multichip Module (MCM) benutzen die Multilagen Ceramic (MLC) Technologie. Ein MLC Module that etwa 100 Verdrahtungslagen und ist wenige mm dick. Gezeigt ist das Einpassen eines z9 MCM in eine Halterung.

Das MCM wird zusammen mit Hauptspeicher DIMMs und zusätzlichen Cards auf ein Printed Circuit Board aufgelötet.

Die MLC Module Technology hat sich seit 1980 evolutionär weiter entwickelt: Weniger und dafür größere Chips. Die Anzahl der Transistoren/Chip wuchs um etwa einen Faktor 10^6 .

4.2.3 Pin Grid Array

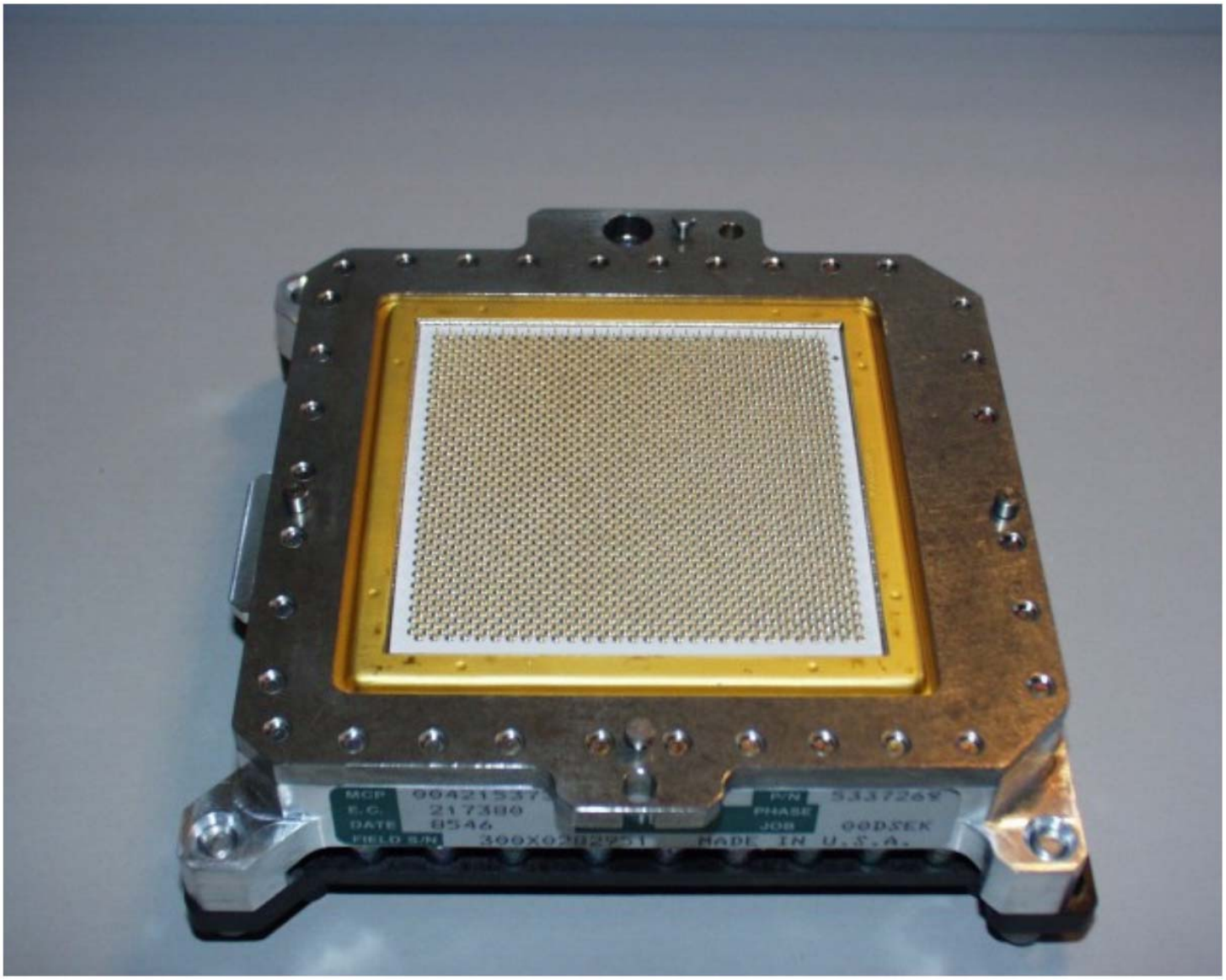


Abb. 4.2.3
Pin Grid Array des z9 Multichip Modules

Die Rückseite des MLC Modules enthielt früher vergoldete Kontaktstifte, welche die Verbindung mit einem Printed Circuit Board aufnehmen, welches gleichzeitig die Hauptspeicher DIMMs und die Host Channel Adapter Cards aufnimmt.

4.2.4 System z Modell 196 Multichip Module

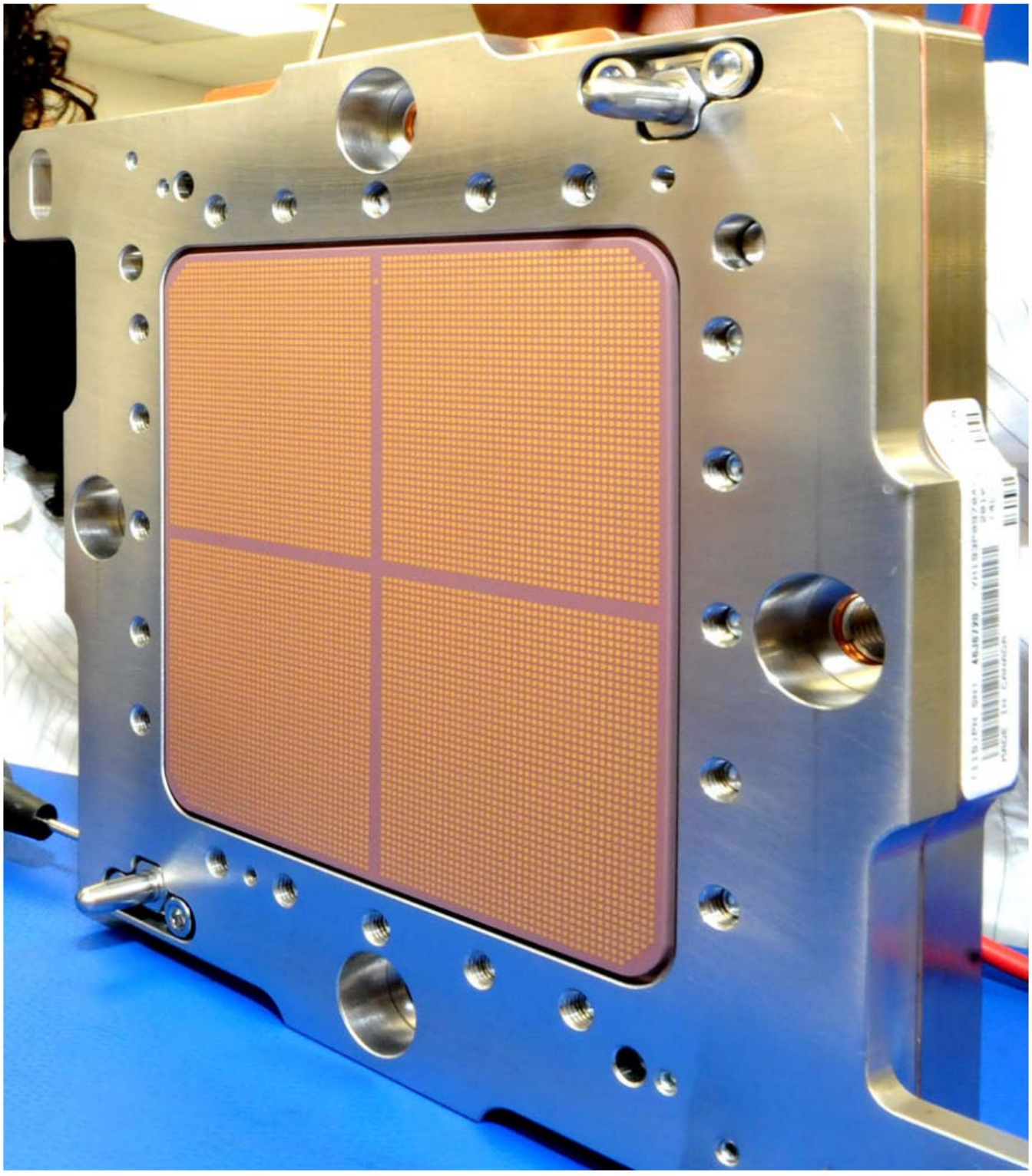


Abb. 4.2.4
Land Grid Array des z196 Multichip Modules

Im System z196 MCM sind die Kontaktstifte durch Kontaktpunkte (Lands) ersetzt. Es sind 7356 Lands vorhanden. Das 96 x 96 mm große MLC Modul hat 103 Verdrahtungslagen.

Ein Land Grid Array (LGA) ist ein Verbindungssystem für integrierte Schaltungen (IC, integrated circuit).

Beim LGA-System werden die Anschlüsse des integrierten Schaltkreises auf seiner Unterseite in Form eines schachbrettartigen Feldes (grid array) von Kontaktflächen (land) ausgeführt. Es ist eng verwandt mit dem PGA-System (Pin Grid Array), welches statt der Kontaktflächen die bekannten „Beinchen“ (pins) besitzt, und dem BGA-System (Ball Grid Array), welches Lötperlen benutzt.

LGA-Prozessoren werden meistens auf Sockel gesetzt, die federnde Kontakte enthalten, was eine geringere mechanische Beanspruchung der Kontakte zur Folge hat. Andere LGA-ICs werden aber oft auch wie PGA-ICs direkt verlötet. BGA-ICs sind hingegen ausschließlich zum Verlöten gedacht, sie bringen das nötige Lötzinn in Form der Lotperlen gleich mit. Alle drei Varianten sind hauptsächlich für ICs mit Hunderten bis über Tausend Anschlüssen gedacht.

Das Land Grid Array ist im Gegensatz zum Pin Grid Array für höhere Frequenzen geeignet und günstiger zu produzieren.

Es können 1800 Watt an Wärme abgeführt (gekühlt) werden.

4.2.5 MLC Substrate

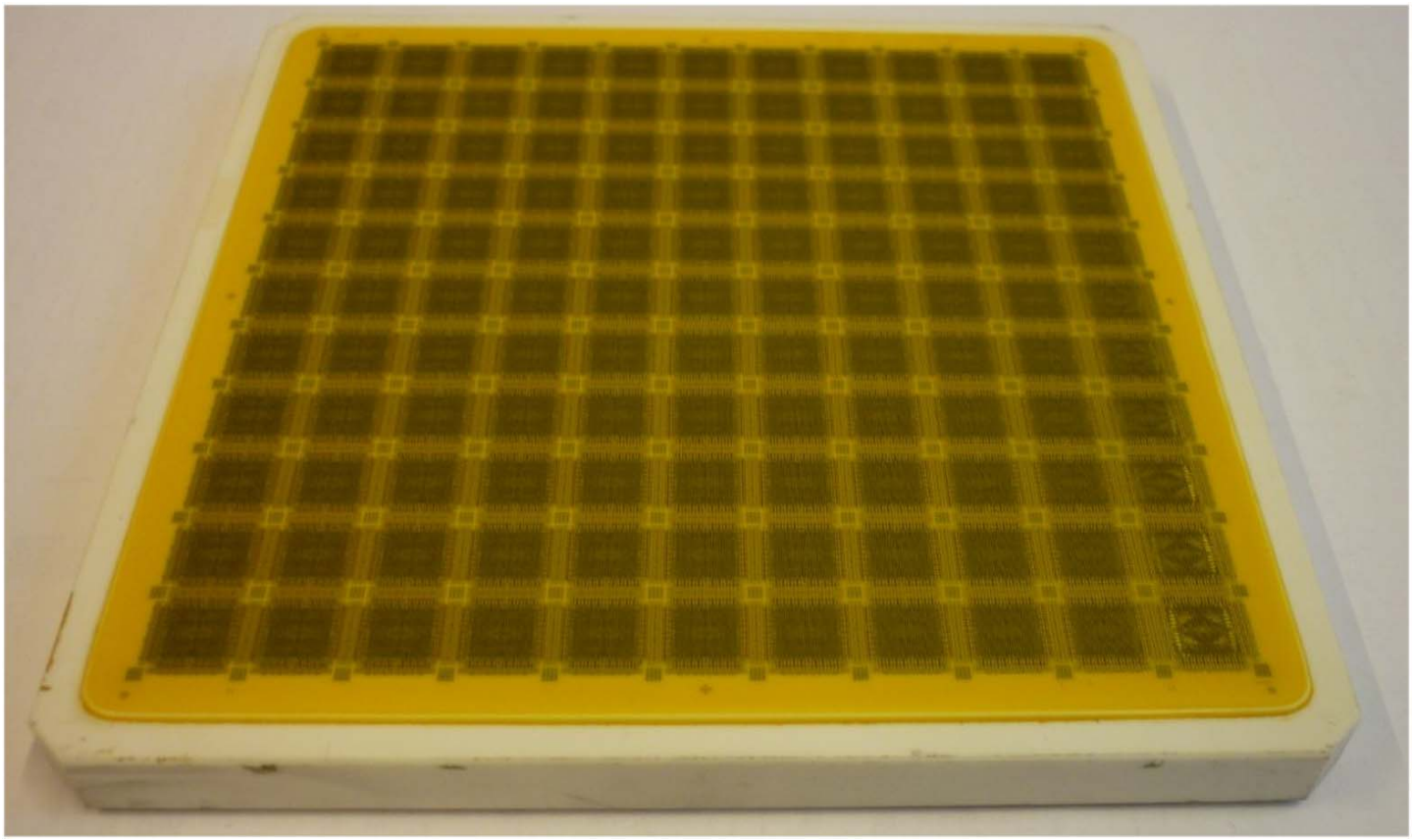


Abb. 4.2.5
2. Mainframe Multichip Multilayer Ceramic (MLC) Module mit 121 Chip Sites aus den 80er Jahren

In den 80er Jahren waren die Chips viel kleiner und hatten viel weniger Schaltkreise pro Chip (704 Circuits/Chip in Abb. 4.2.5). Eine CPU bestand aus vielen Chips.

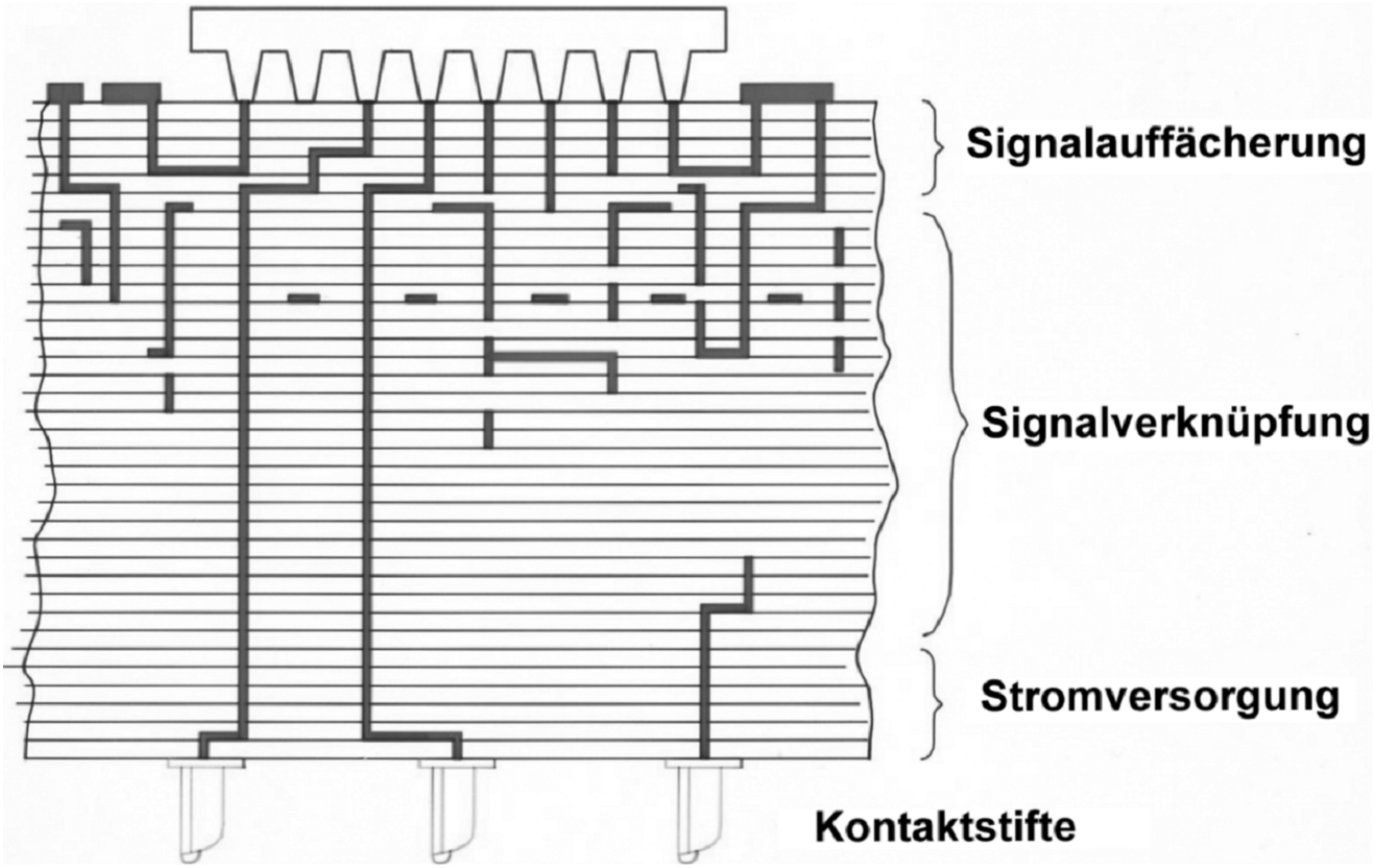


Abb. 4.2.6
Verdrahtungslagen eines Multilayer Ceramic Modules

Das zEC12 – Multilayer Ceramic (MLC) Multichip Module benutzt einen Glas-Keramik-Träger mit 103 Glas-Keramik Verdrahtungslagen und 7356 Land Grid Array (LGA) Connections. In dem 96 x 96 mm-Modul sind Leiterbahnen mit einer gesamten Länge von mehr als 500 Meter untergebracht. Innerhalb der verschiedenen Schichten entstehen komplexe Verdrahtungsmuster. Die senkrechten Verbindungen zwischen den Schichten bestehen aus leitenden Bohrungen (VIAs), die wiederum innerhalb einer Schicht in horizontalen Leiterbahnen weitergeführt werden und an einer Bohrung zu einer darunter- oder darüber liegenden Schicht enden usw. Die früher verwendeten Kontaktstifte werden heute durch Kontaktpunkte (Lands) ersetzt.

Das heute verwendete Glas-Keramik-Material tritt an Stelle der früher verwendeten Aluminiumoxid (AL2O3)-Keramik. Es hat eine um 1/3 geringere Dielektrizitätskonstante und damit eine kürzere Signallaufzeit der Leitungsverbindungen. Eine sehr ähnliche Packungstechnologie hat sich in der Hochfrequenztechnik bewährt.

Die Glas-Keramik-Technologie steht im Gegensatz zu den normalerweise in Printed Circuit Boards (PCB) verwendeten organischen Materialien und Wirebond Peripheral Interconnect-Verfahren mit Lead Frame- oder Pin Grid Array (PGA)-Verbindungen. Diese haben schmalbandige induktive Diskontinuitäten (Drähte, Pins, Leads) und lange Netze mit erhöhter Laufzeitverzögerung.

4.2.6 Thermal Conduction Module

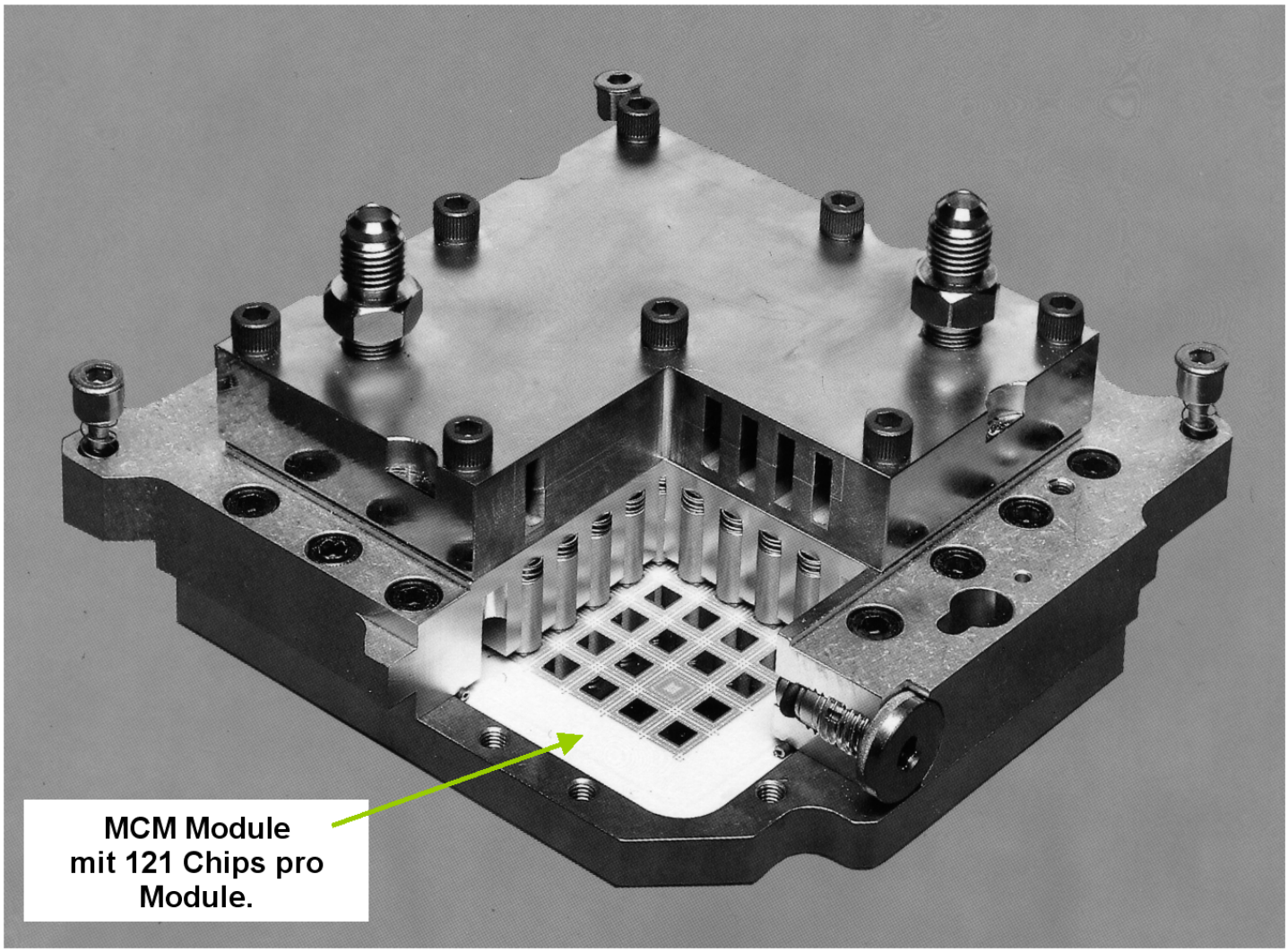


Abb. 4.2.7
Schnitt durch ein Thermal Conduction Module

Ein Thermal Conduction Module (TCM) ist eine Baugruppe, welche ein Multilayer Ceramic (MLC) Multichip Module (MCM) aufnimmt, und die für die Energieabfuhr (Kühlung) erforderliche Hardware enthält.

Die TCM und MLC Technologie wurde von IBM in den 80er Jahren zur Produktionsreife gebracht und seitdem kontinuierlich weiterentwickelt. An der Grundkonzeption hat sich allerdings erstaunlich wenig geändert.

Die meisten Mainframe Modelle benutzen seitdem diese Technologie. Sie zeichnet sich durch besonders hohe Zuverlässigkeit aus. Die elektrischen Eigenschaften bewirken eine besonders kurze Signallaufzeit zwischen den Chips eines MCM. Ein z196 TCM hat eine Kühlleistung von 1,8 KWatt, etwa soviel Energie wie ein Bügeleisen abstrahlt.

In Abb. 4.2.7 ist ein 1987 gefertigtes Thermal Conduction Module (TCM) dargestellt, mit 121 Chips auf dem MCM, und 704 circuits/chip. Mehrere dieser Module bildeten die CPU einer S/370 Modell 3081 Mainframe CPU.

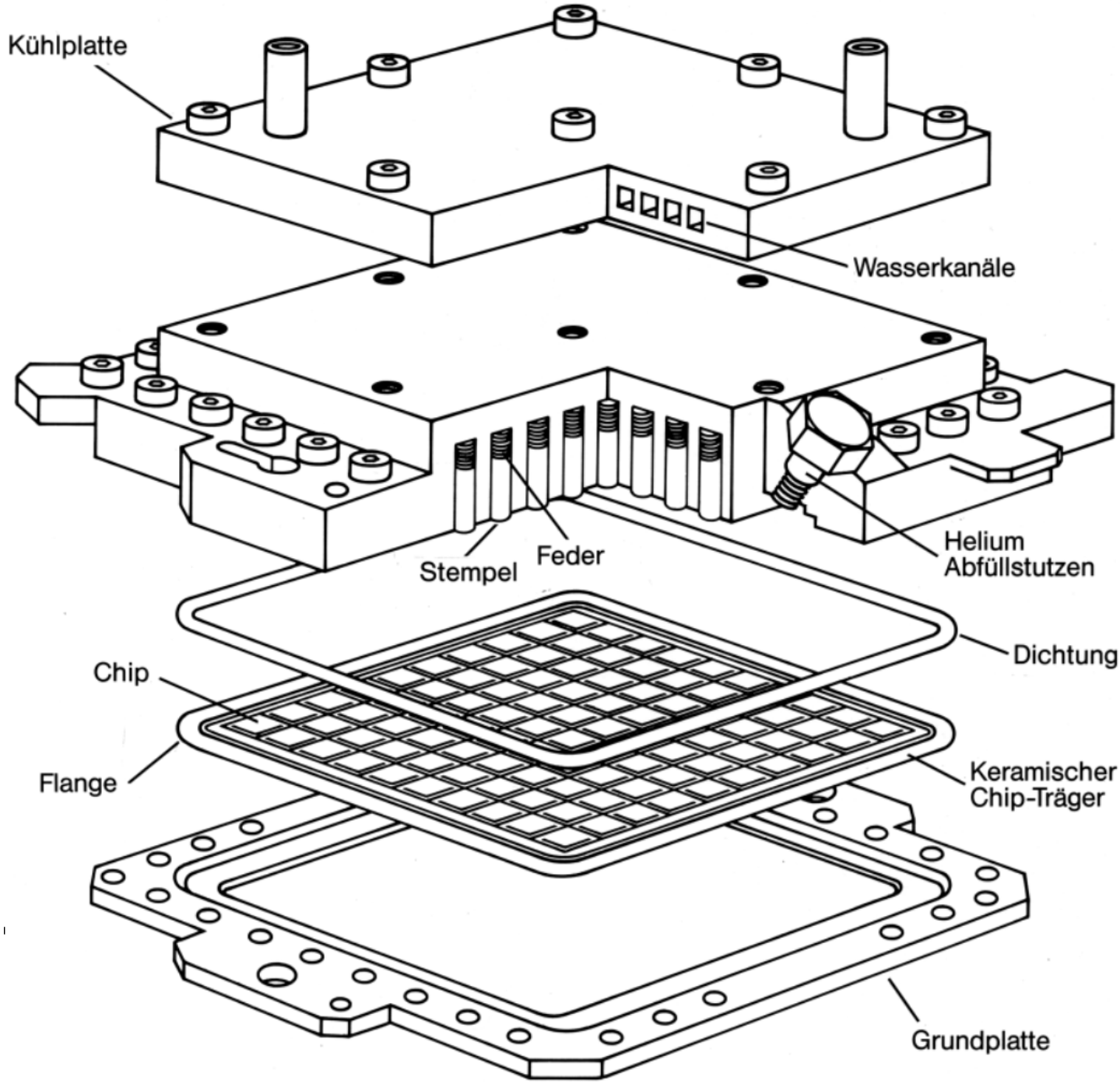


Abb. 4.2.8
Aufbau eines MCM Modules

Eine MCM-Baugruppe enthält neben dem keramischen Chip-Träger die Mechanik für die Energieableitung mittels Stempel und Kühlplatte. Diese Baugruppe wird als Thermal Conduction Module (TCM) bezeichnet und ist in der obigen Abbildung dargestellt.

Zu Kühlungszwecken sitzt auf jedem Chip ein Aluminium-Stempel, der die Verlustwärme ableitet . Eine Spiralfeder drückt den Stempel an die Chip Oberfläche an. Die das Chip berührende Oberfläche des Stempels ist konisch ausgebildet mit einem Konus Radius im Bereich vieler 100 Meter. Dies sorgt für einen guten Kontakt des Stempels mit der Chip-Oberfläche und für einen minimalem Luftspalt, auch wenn der Stempel geringfügig verkantet (siehe Abb. 4.2.9).

Die TCMs sind vielfach mit Helium an Stelle von Luft oder Stickstoff gefüllt. Die Wärmeleitfähigkeit von Helium ist größer als die jeder anderen bekannten Substanz.

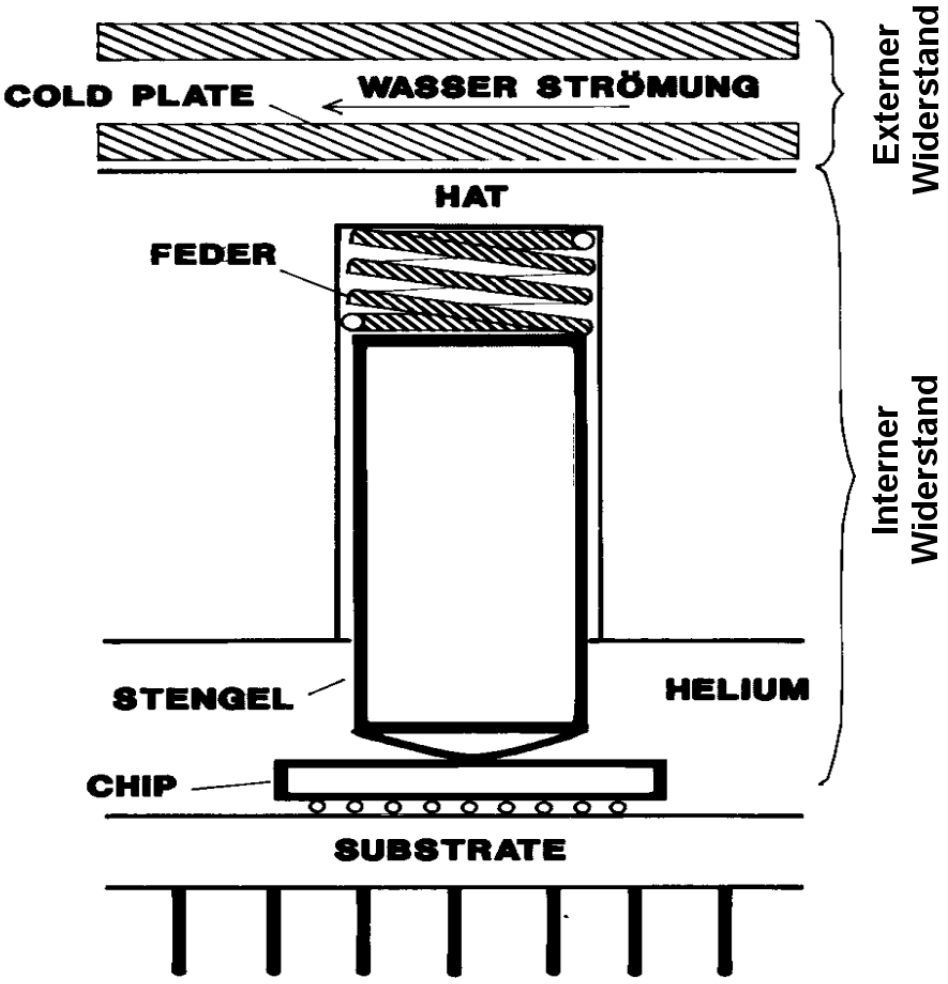


Abb. 4.2.9
Kühlung eines Chips

Die Aluminium-Stempel werden in einer Bohrung geführt und geben die Verlustwärme an die Umgebungsplatte weiter. Eine darüber liegende Kühlplatte wird entweder mit Luft oder mit Wasser gekühlt. Im letzteren Fall existiert ein geschlossener Kreislauf, in dem das Wasser seine Wärmeenergie an einen Radiator innerhalb des System z Frames weitergibt, ähnlich wie in einem Automobil. Es existieren auch System z Modelle, wo die Wasserkühlung extern erfolgt.

Bei dem hier vorgestellten Verfahren werden alle Chips selbst mit Luft (Helium) gekühlt. Es sind in der Vergangenheit viele Versuche unternommen worden, Chips direkt mit einer Flüssigkeit zu kühlen. Diese Verfahren haben sich in der Praxis nicht bewährt.

4.2.7 Intel Pentium Pro

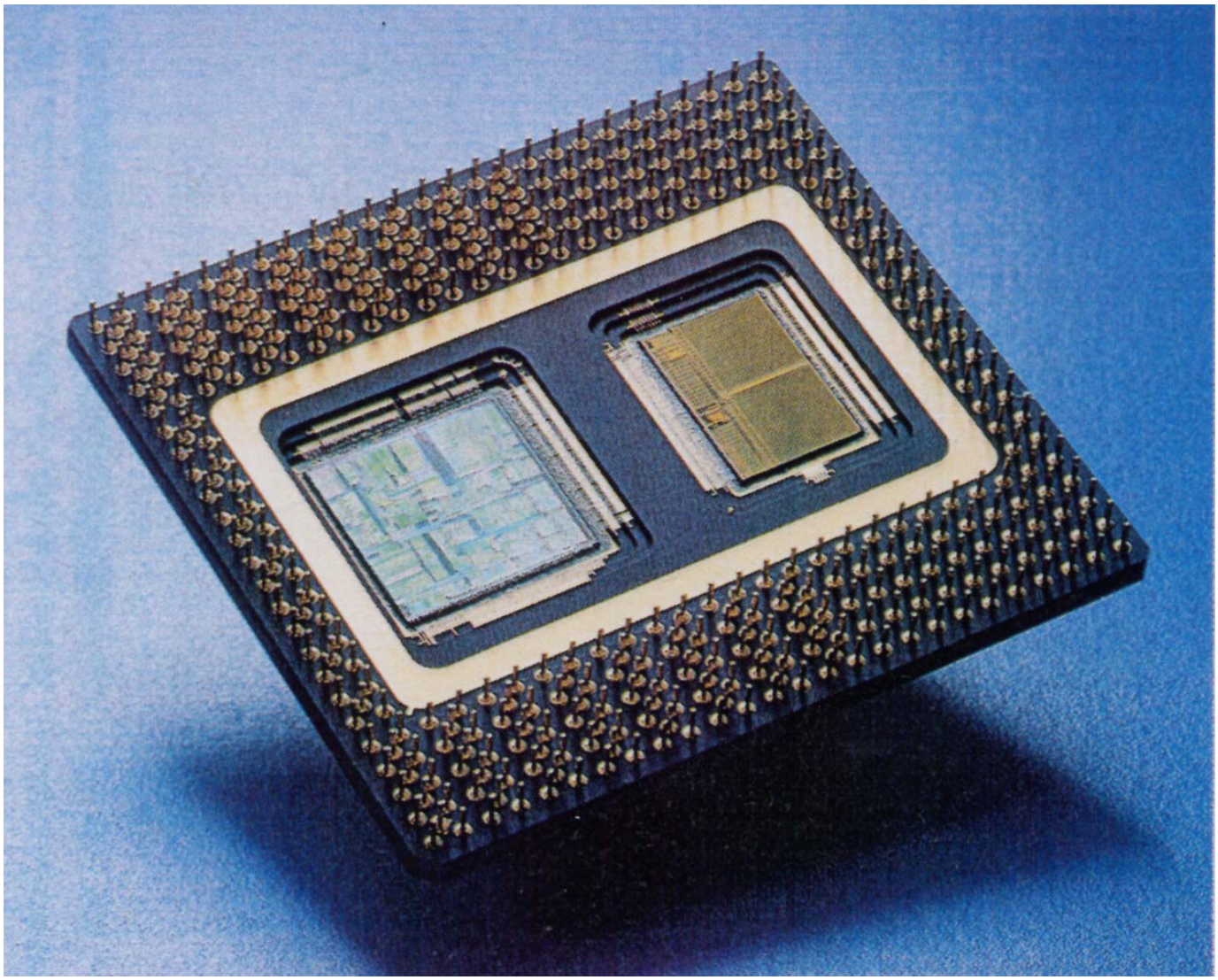


Abb. 4.2.10
Pentium Pro Dual Chip MLC Module

Bei der ursprünglichen Einführung des Pentium hatte die Firma Intel eine ähnliche MCM-Technologie für den Pentium Pro eingesetzt. Der 1995 von Intel herausgebrachte Pentium Pro Microprocessor verwendete ein 387 Pin Multi Layer Ceramic (MLC) Multi Chip Carrier (MCM) Module. Das Module enthielt zwei Chips, ein CPU chip und ein getrenntes L2 Cache Chip, beide auf dem gleichen MLC Substrate. Die MCM-Technologie ermöglichte besonders günstige Signallaufzeiten zwischen den Chips.

Der Pentium Pro hatte deshalb eine überdurchschnittliche Leistung. Er wurde für Server und für High-End Desktop Prozessoren eingesetzt und war auch in Supercomputern wie ASCI Red benutzt worden.

Der Pentium Pro wurde seinerzeit von den Server-Herstellern wegen seiner guten Leistungsdaten sehr geschätzt. Intel hatte aber Schwierigkeiten mit den Produktionskosten und hat deswegen die MCM-Technologie bis jetzt noch nicht wieder verwendet.

4.2.8 z196 Multichip Module

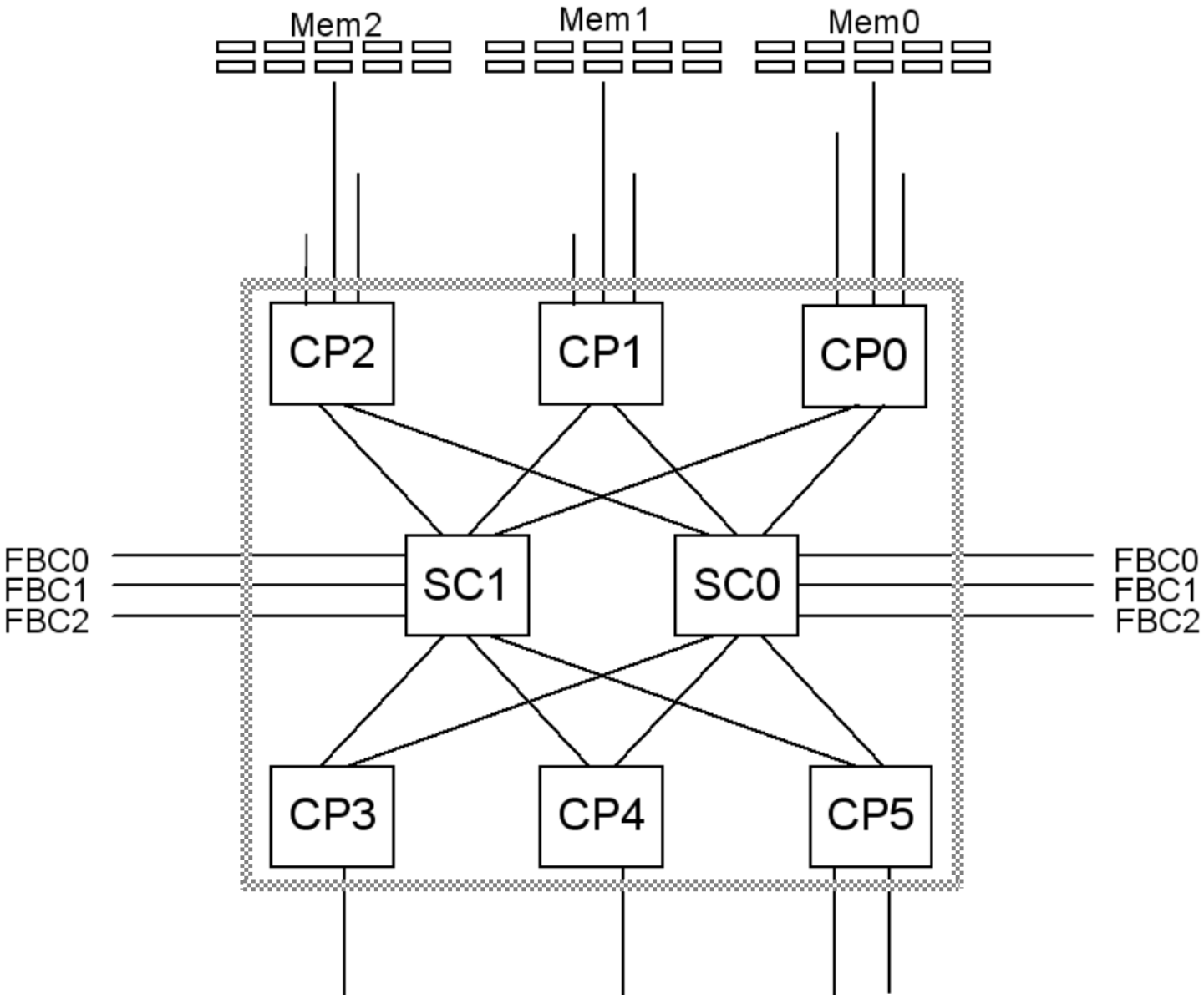


Abb. 4.2.11
Verbindungen der Chips auf einem zEC12 MCM

Auf dem zEC12 Multichip Module (siehe auch Abb. 4.1.8) befinden sich sechs CPU Chips (CP0 ..CP5) sowie zwei L4 Cache Chips SC0, SC1).

Jedes CPU Chip ist mit beiden L4 Cache Chips direkt verbunden Außerdem enthalten 3 der 6 CPU Chips eine Memory Management Unit und sind direkt mit den Hauptspeicher DIMMs verbunden, von denen bei Bedarf eine Cache Line nachgeladen werden kann.

Ein z 196 System kann vier als „Books“ bezeichnete Baugruppen enthalten. Jedes Book beinhaltet ein Multichip Module.

Die L4 Cache Chips aller Books sind miteinander über „Fabric Book Connectivity (FBC)“ Leitungen verbunden und bilden einen gemeinsam genutzten Cache.

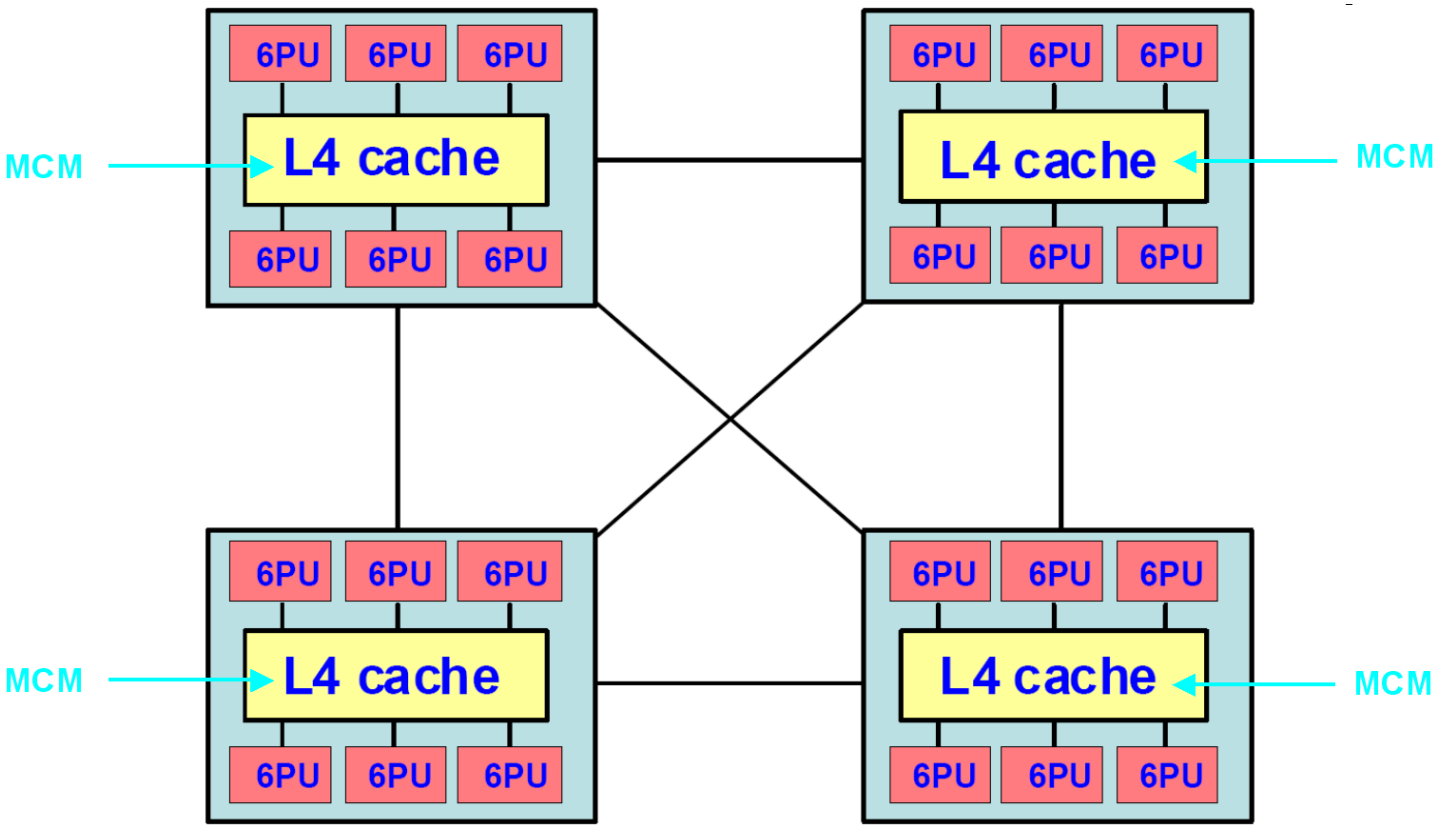


Abb. 4.2.12
Verbindung der L4 Caches zwischen vier zEC12 MCMs

Die vier L4 Caches (je 2 L4 Cache Chips) der vier z196 Books mit je 24 CPU Cores sind mittels einer Punkt zu Punkt Topologie über „Fabric Book Connectivity (FBC)“ Leitungen direkt miteinander verbunden. Direkter Datenaustausch zwischen den vier L4 Caches.

Alle 4 x 36 = 144 Cores greifen auf die L4 Caches aller Books direkt zu. Die vier L4 Caches implementieren einen NUMA (Non Uniform Memory Architecture) Cache (siehe Book 2, Abschnitt 12.1 und Abb. 12.1.2 – 12.1.4).

4.2.9 Datentransfer zwischen Hauptspeicher und Festplatten

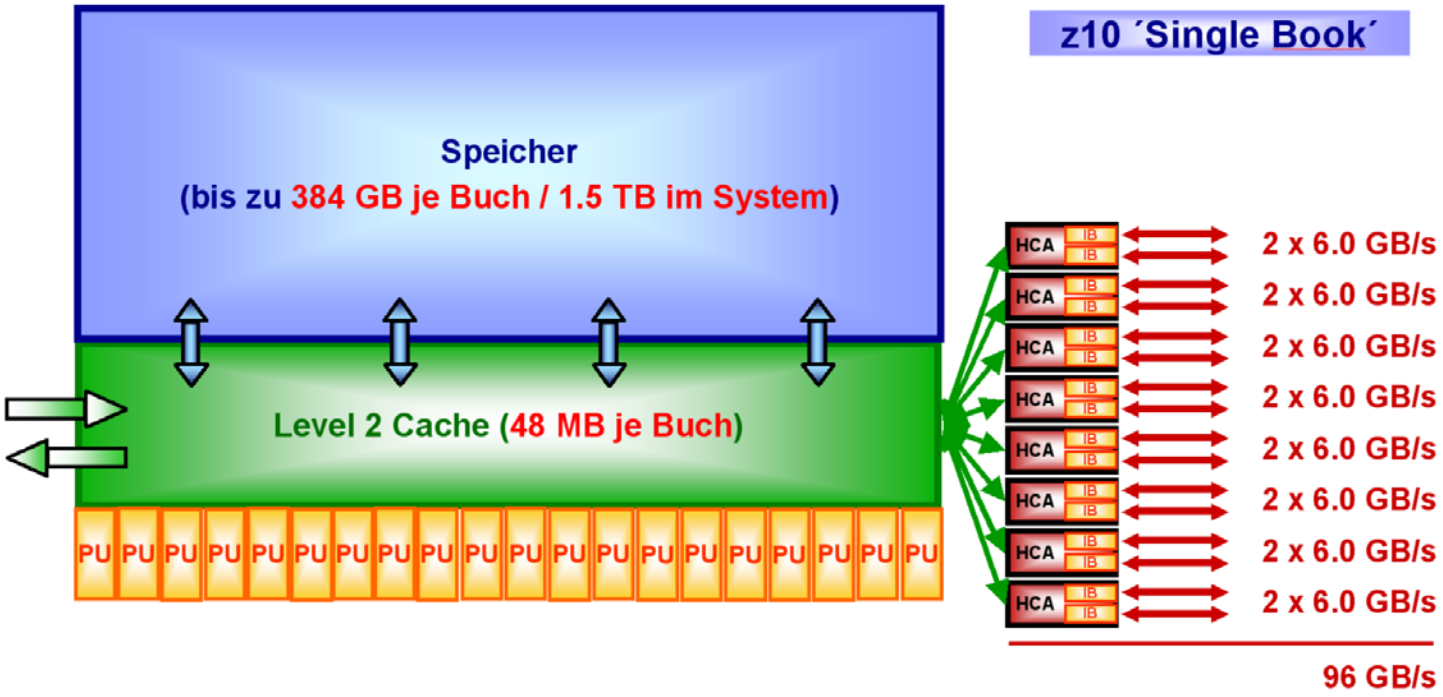


Abb. 4.2.13
Festplattenspeicher Anschluss eines MCMs

Dies ist ein weiteres Mainframe Alleinstellungsmerkmal: In allen nicht-Mainframe Servern (und allen PCs) bewirken die Fanout Adapter Karten einen Datentransfer zwischen Plattenspeicher und Hauptspeicher. In den Mainframe Rechnern erfolgt der Datentransfer zwischen Plattenspeicher und dem L4 Cache. Damit sind natürlich wesentlich höhere Datenraten möglich.

Dies wurde bisher auf Grund von Cache Kohärenzproblemen für unmöglich gehalten. Die eingesetzten Kohärenzalgorithmen wurden bisher auch noch nicht von IBM veröffentlicht.

4.3 Book und System Frame

4.3.1 Struktur eines Books

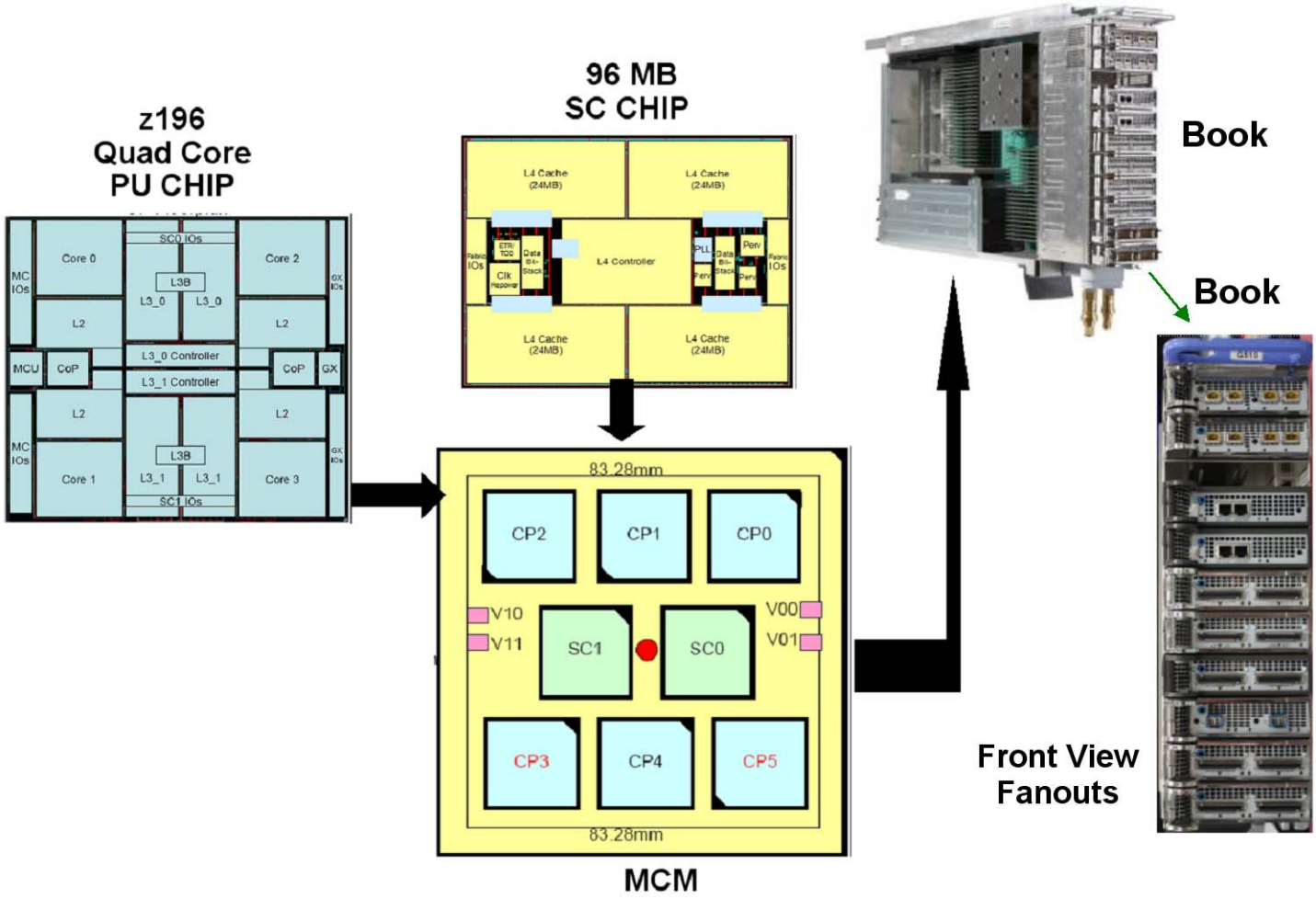


Abb. 4.3.1
Z196 CPU Chip, SC Chip und MCM als Teile eines Books

Abb. 4.3.1 zeigt ein CPU Chip (CP) und ein L4 Cache Chip (SC Chip). 6 + 2 dieser Chips befinden sich auf einem Multichip Module (MCM).

Das MCM ist Bestandteil eines Books.

Das MCM ist Bestandteil einer als „Book“ bezeichneten Baugruppe, welche neben dem MCM noch Steckplätze für 30 Hauptspeicher DIMMs (Dual Inline Memory Module) sowie „Fan Out Adapter“ Karten enthält. Fan Out Cards werden auch als Host Channel Adapter (HCA) Cards bezeichnet. Sie erfüllen die gleiche Funktion wie die Fanout Cards der Sun oder Hewlett Packard System Boards, siehe Abb. 4.1.5.

Hauptspeicher DIMMs haben eine Kapazität von je 4 GByte, 16 GByte oder 32 GByte. Die maximale physische Hauptspeicherkapazität beträgt somit 960 GByte, von denen nach Abzug für Fehlerkorrektur und Redundanz 768 GByte verfügbar sind.

Die Fanout Cards führen zu Steckkontakten auf der Vorderseite (Front View) des Books. Es existieren 8 Steckkontakte , die mit jeweils 2 Kabeln eine Verbindung zu einem „I/O Cage“ aufnehmen, in denen Steckkarten für I/O Anschlüsse untergebracht sind. Zwei weitere Steckkontakte (FSP) werden zur Verbindung zu zwei „Support Elementen“ verwendet, siehe Abb. 4.3.12, und Abschnitt 14.2.3 in Band 2.

Es existiert eine zweistufige Stromversorgung. Die primäre Stromversorgung (Bulk Supply) versorgt einen ganzen Rechner, und ist aus Zuverlässigkeitsgründen doppelt vorhanden. In jedem Book befindet sich eine zusätzliche sekundäre Stromversorgung (Distributed Converter Assembly, DCA) , welche kurzfristige Versorgungs-Schwankungen innerhalb eines Books ausgleicht.

Abb. 4.3.2 zeigt die Seitenansicht eines (geöffneten) Books, in der das Multichip Module (MCM), die Hauptspeicher DIMMs, die (sekundäre) Stromversorgung und (hinter der Abdeckung verborgen) die Fanout Cards zu sehen sind.

Voltage Tansformation Modules (VTM) bewirken Power Conversion in dem Book und benutzen Triple Redundancy. Die Redundanz schützt die Prozessoren vor einem Spannungsverlust als Folge des Versagens einer VTM Card. Triple Redundancy wird auch für die Humidity (Luftfeuchtigkeit) Sensoren zur Verbesserung der Zuverlässigkeit eingesetzt.

4.3.2 Baugruppen eines Books

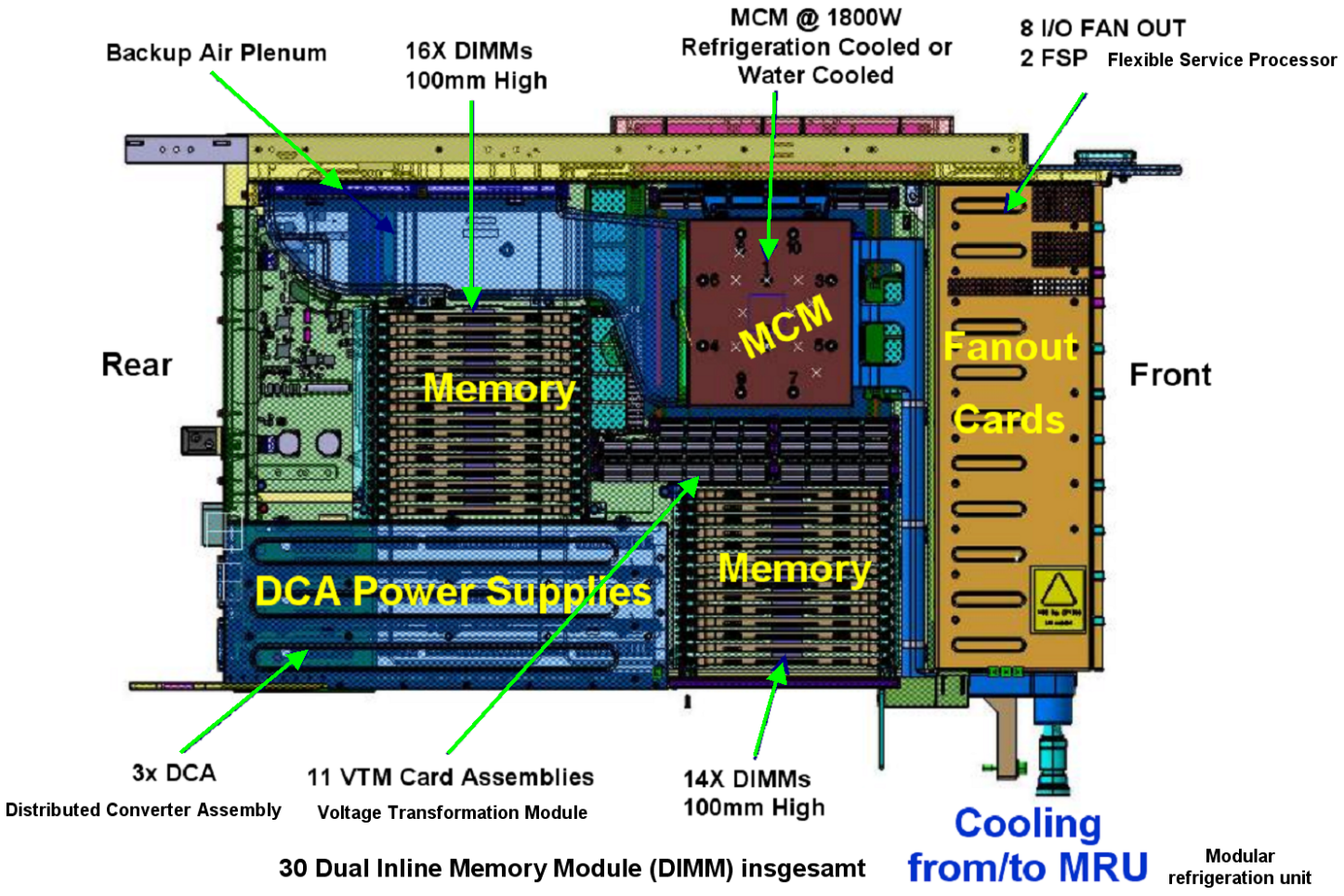


Abb. 4.3.4
Seitenansicht eines zEC12Books

Jedes zEC12 Book enthält die folgenden Komponenten:

- **Ein Multi-Chip-Modul (MCM) mit sechs Hex-Core-Mikroprozessor-Chips, und zwei Storage Control Chips mit insgesamt 384 MByte Level 4 Cache.**
- **Speicher DIMMs in 30 verfügbaren Slots. Dies ermöglicht von 60 GByte bis 960 GByte physischen Speicher pro Book.**
- **Eine Kombination von bis zu acht Host Channel Adapter (HCA) oder PCIe Fanout-Karten.**
- **PCIe Fanouts werden für 8Gbit/s Links zu den PCIe I/O-Karten verwendet. Die HCA-Optical Fanouts verbinden zu externen coupling links (Band 2, Abschnitt 11.2).**
- **Drei verteilte Wandler-Baugruppen (DCAs), die das Book mit Strom versorgen. Bei Verlust einer DCA ist genügend Power vorhanden (n +1 Redundanz), um den Strombedarf des Books zu befriedigen. Die DCAs können während des laufenden Betriebes gewartet und ausgetauscht werden.**
- **Zwei „Flexible Service-Prozessor“ (FSP)-Karten für die Systemsteuerung, siehe Band 2, Abschnitt 14.2.3.**

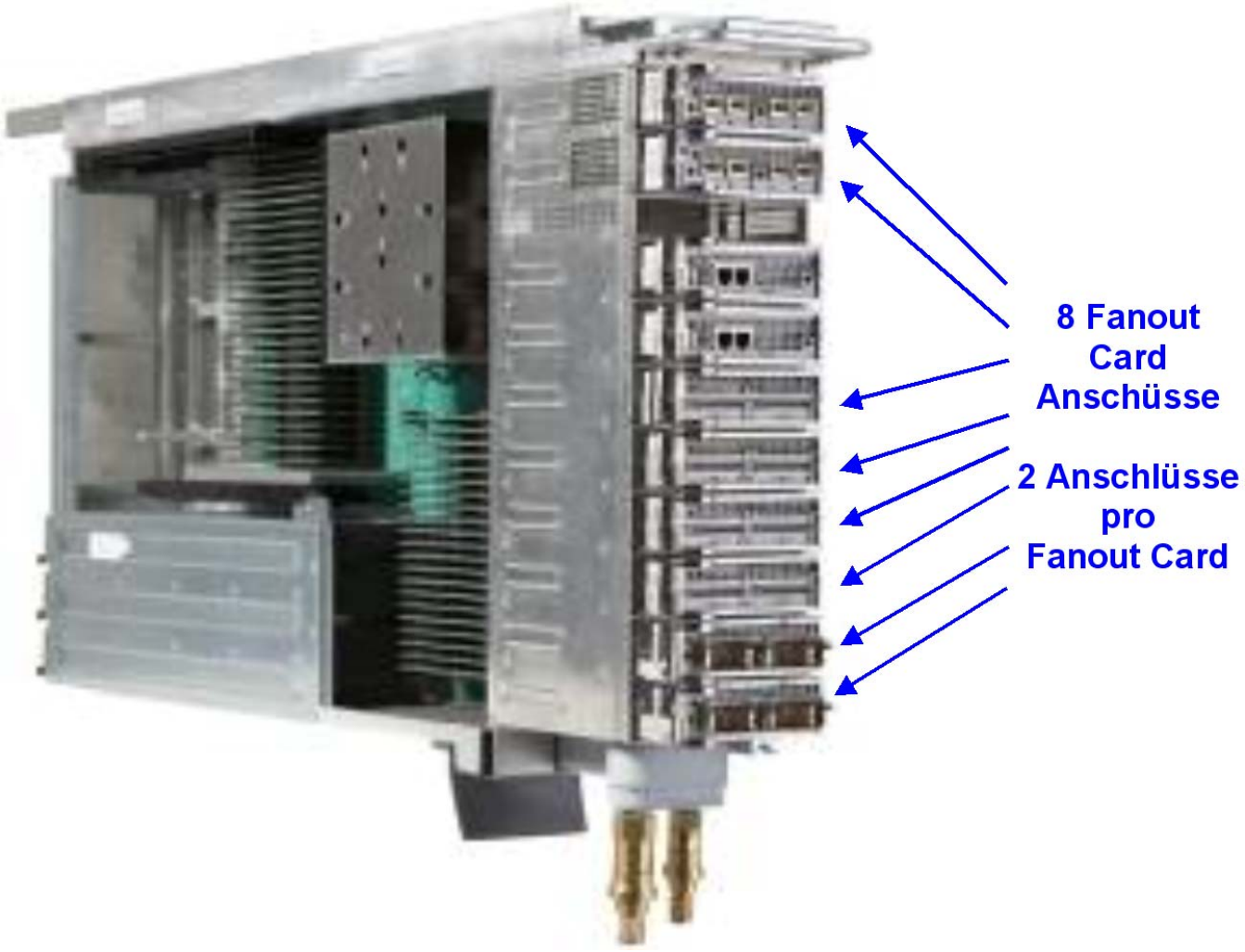


Abb. 4.3.3
Anschlüsse für I/O Adapter Karten

Fanout Cards werden auch als „Host Channel Adapter“ (HCA) Cards bezeichnet. Pro Fanout Card sind 2 nebeneinander-liegende Steckkontakte für Kabel-Verbindungen mit einem I/O Cage vorhanden.

Das MCM ist Bestandteil einer als „Book“ bezeichneten Baugruppe, welche neben dem MCM noch Steckplätze für 30 Hauptspeicher DIMMs (Dual Inline Memory Module) sowie „Fan Out Adapter“ Karten enthält. Fan Outs Cards werden auch als Host Connector Adapter (HCA) Cards bezeichnet. Sie erfüllen die gleiche Funktion wie die I/O Adapter Cards der Sun oder Hewlett Packard System Boards.

Hauptspeicher DIMMs haben eine Kapazität von je 4 GByte, 16 GByte oder 32 GByte. Die maximale physische Hauptspeicherkapazität beträgt somit 960 GByte, von denen nach Abzug für Fehlerkorrektur und Redundanz 768 GByte verfügbar sind.

Die Fanout Cards führen zu Steckkontakten auf der Vorderseite (Front View) des Books. Es existieren 8 Steckkontakte , die mit jeweils 2 Kabeln eine Verbindung zu einem „I/O Cage“ aufnehmen, in denen Steckkarten für I/O Anschlüsse untergebracht sind. Zwei weitere Steckkontakte (FSP) werden zur Verbindung zu zwei „Support Elementen“ verwendet, siehe Abb. 4.3.12.

4.3.3 Anschlüsse eines Books

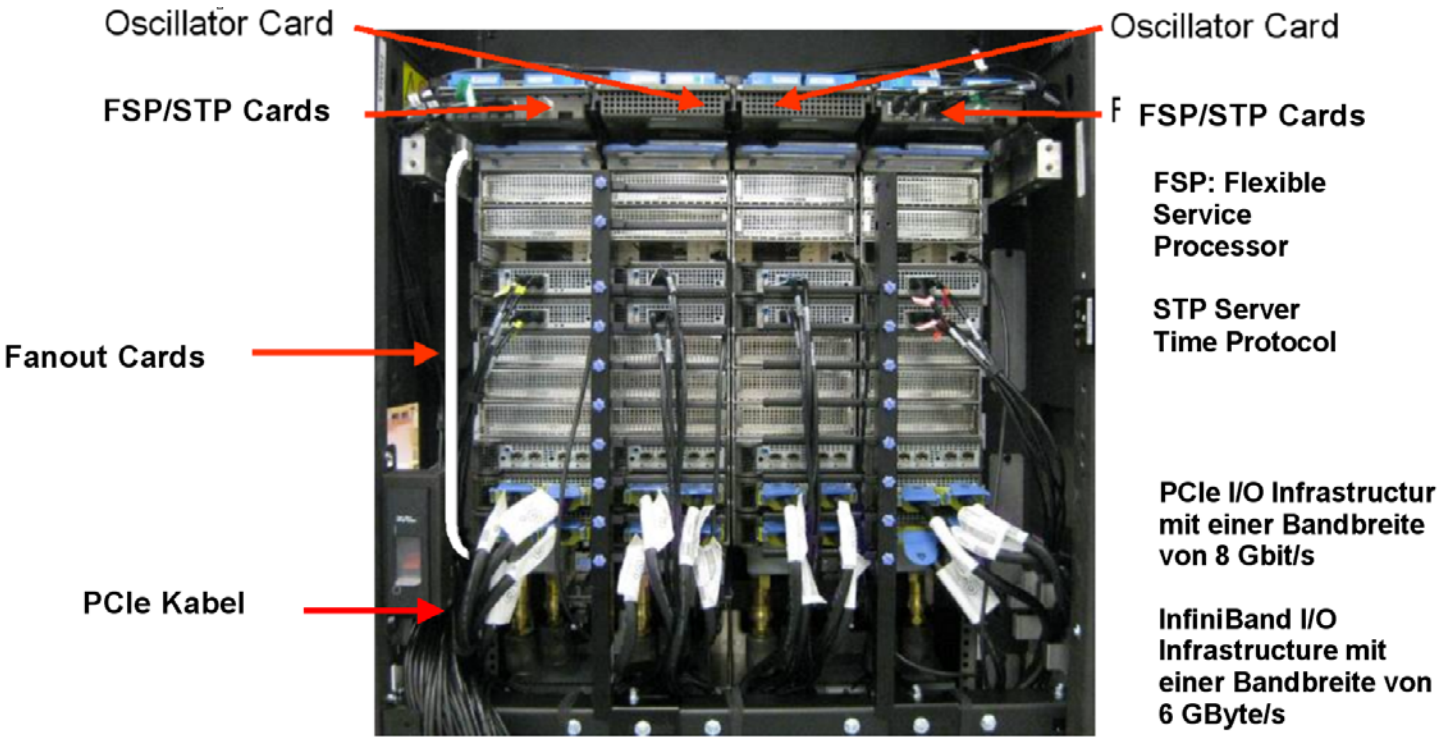


Abb. 4.3.4
PCIe Kabelverbindungen zur Front Seite eines Books

Abb. 4.3.4 zeigt die Vorderseite(n) von 4 nebeneinander stehenden Books. Zu sehen sind die Steckkontakte, über die mit Infiniband oder PCIe Kabeln die Verbindung zu einer (max. 3) I/O Drawers hergestellt werden, welcher I/O Adapter Karten (z.B. Plattenspeicher Anschlüsse) aufnimmt, die eine Verbindung zur Außenwelt übernehmen.

Zwei weitere Steckkontakte (FSP) werden zur Verbindung zu zwei „Support Elementen“ verwendet. Zwei Oscillator Karten stellen Clock Signale zur Verfügung und stellen eine Synchronisation aller CPUs mittels des Server Time Protocols (STP) sicher.

Abb. 4.3.5
Hier entfernt ein Entwicklungsingenieur ein Book aus einem z9 Rechner.

4.3.4 Verbindung MCM - Hauptspeicher

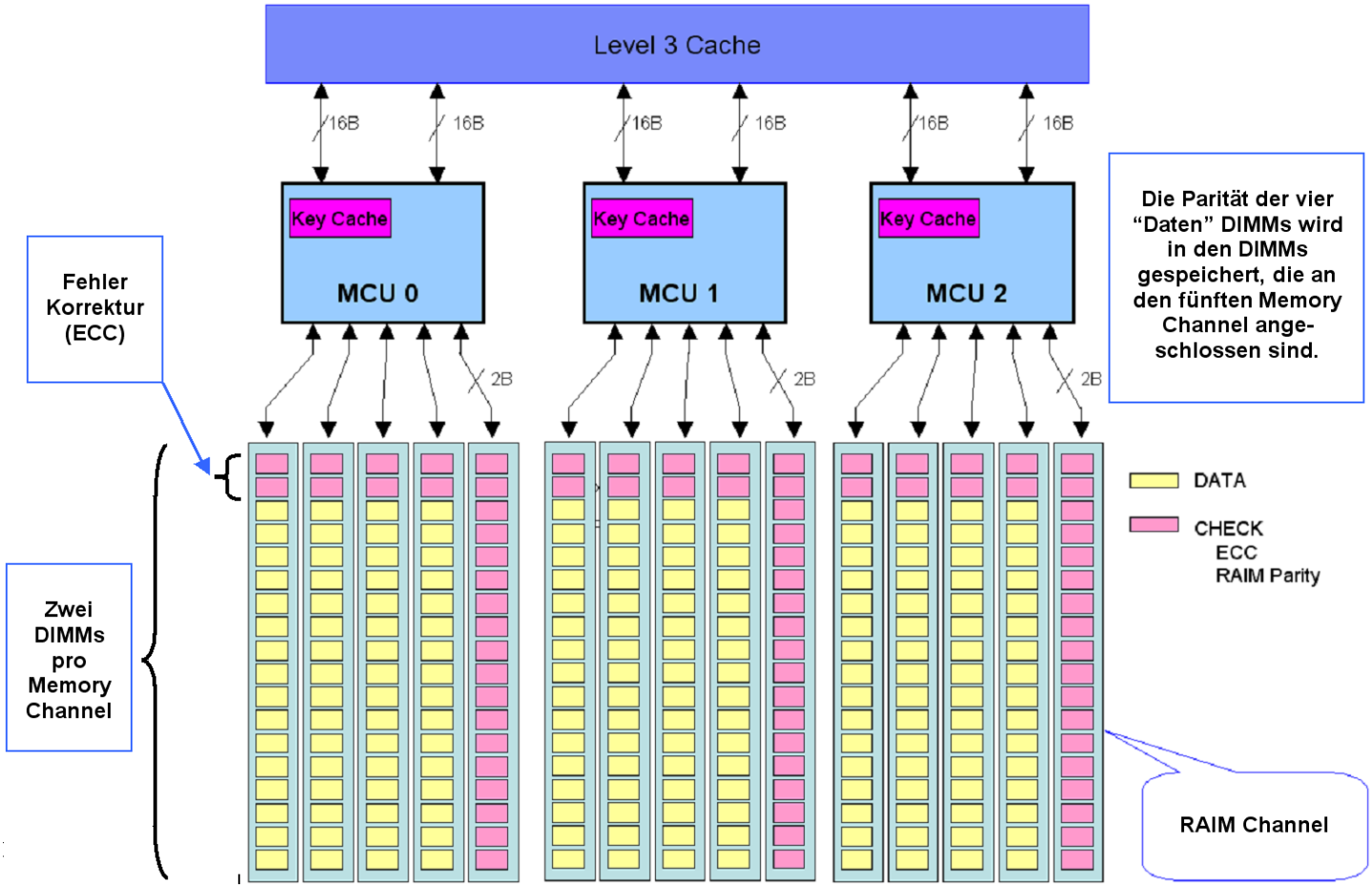


Abb. 4.3.6
Redundant Array of Independent Memories

Abb. 4.3.6 zeigt den Hauptspeicheranschluss.

Dieser erfolgt mit Hilfe von drei „Memory Control Units“ (MCU), welche über zwei je 16 Byte (2 x 128 Bit) breite Busse mit den L3 Caches der einzelnen CPU Chips verbunden sind. Die Memory Control Units enthalten zusätzlich einen Cache, der die am häufigsten gebrauchten Storage Keys (siehe Abschnitt 1.3.10) enthält.

Der Hauptspeicher selbst enthält 4 „Channels“ (Columns) von Hauptspeicher Quad High DIMMs (Dual Inline Memory Module), vom Aussehen her ähnlich wie die DIMMs in Ihrem PC. Jeder Channel verwendet eine spezielle Version des ReedSolomon Fehlerkorrektur Codes an Stelle des sonst für Hauptspeicher üblichen Hamming Fehlerkorrektur Codes.

Als weitere Fehlerkorrekturmaßnahme existiert ein 5. RAIM (Redundant Array of Independent Memory) Channel. RAIM benutzt das gleiche Verfahren wie RAID für Plattenspeicher (siehe Abschnitt 5.3.7). Wenn eine der 4 Colums ausfällt, kann der Hauptspeicherinhalt mit Hilfe der 5. Column wieder hergestellt werden. RAIM wurde erstmalig mit der z196 für kommerziell erhältliche Rechner eingeführt.

An jedem Channel hängen 2 Dual In-Line Memory Modules (DIMMs); jede einzelne MCU bedient 10 DIMMs. Jedes DIMM Module speichert 32 GByte. Jedes Book verfügt über DIMMs, für eine maximale Speicherkapazität von 960 GByte pro Book, oder 3 480 GByte für ein 4 Book System.

Da die RAIM DIMMs 20 % der Speicherkapazität in Anspruch nehmen, stehen dem Benutzer lediglich 768 GByte pro Book, oder 3 072 GByte pro System zur Verfügung. Davon werden 32 GByte für die „Hardware System Area“ (HSA) benötigt. Die HSA speichert Firmware, und wird in Band 2, Abschnitt 12.3.1 erläutert.

4.3.5 Struktur eines zEC12 Mainframe Systems

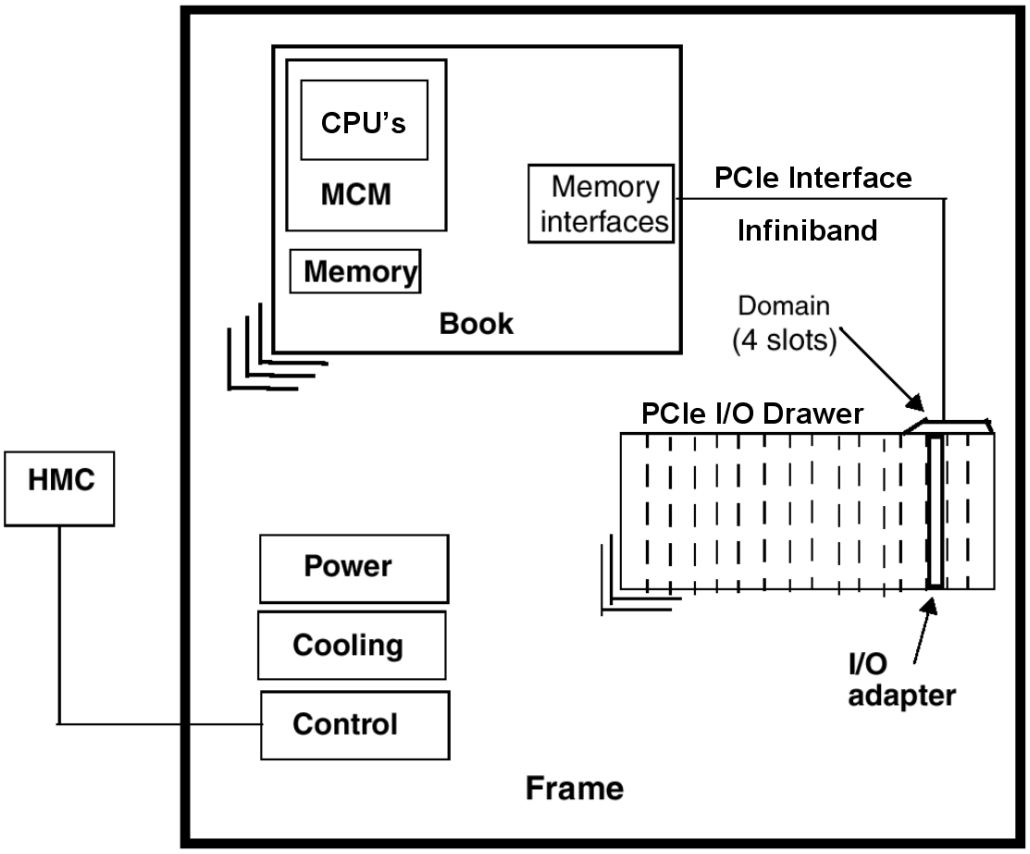


Abb. 4.3.7
Baugruppen eines Maiframe Rahmens

Bis zu 32 I/O Adapter Cards pro I/O Drawer, für Verbindungen zu Plattenspeichern, Magnetbändern und anderen I/O Geräten.

4.3.6 PCIe I/O drawer

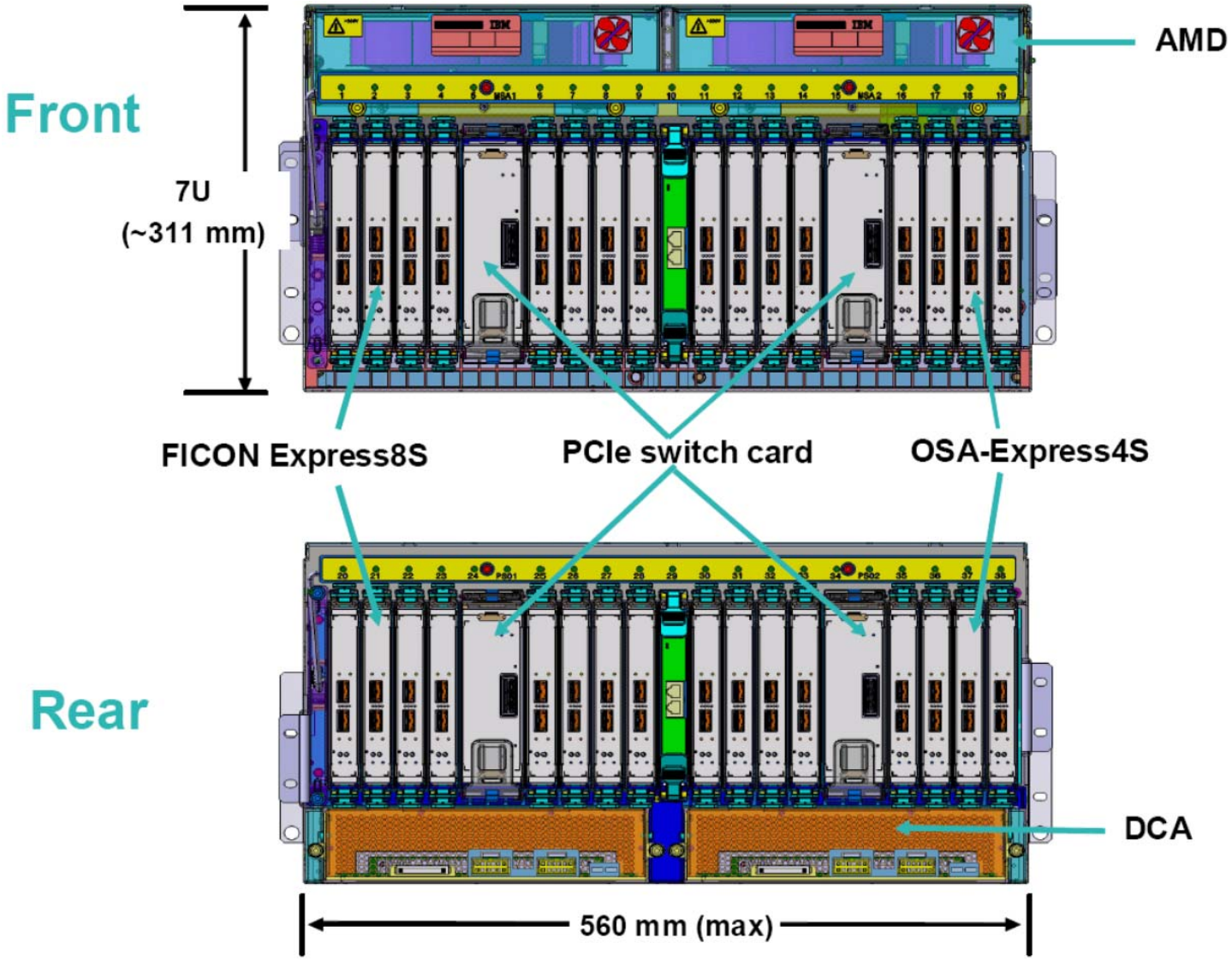


Abb. 4.3.8
Ein I/O Drawer kann bis zu 32 I/O Adapter Karten aufnehmen

Abb. 4.3.8 zeigt die interne Struktur eines z9, z10 z196 Mainframe oder zEC12 Systems. CPUs und Caches befinden sich auf einem Multi Chip Module (MCM). Ein einziges MCM, zusammen mit dem Hauptspeicher (main store) bilden eine als “Book” bezeichnete Baugruppe. Bis zu 4 Books sind möglich.

Die Hauptspeicherschnittstelle (memory interface) verbindet ein Book mit „I/O Adaptern“ (I/O Cards) die in einem „I/O Drawer” (auch als I/O Cage bezeichnet) untergebracht sind. Die Verbindung zwischen Book und I/O Drawer wird durch I/O Kabel hergestellt, welche das PCIe Protokoll implementieren, äquivalent zu dem PCIe Protokoll in einem PC. Die PCIe Drawer ist eine zweiseitige Drawer (I/O-Adapter auf beiden Seiten). Die Drawer enthält 32 I/O-Steckplätze, und kann bis zu 32 I/O Adapter Cards aufnehmen, für Verbindungen zu Plattenspeichern, Magnetbändern und anderen I/O Geräten. Letztere sind grundsätzlich in getrennten Gehäusen untergebracht.

Die PCIe I/O drawer nutzt PCIe als Infrastruktur. Die PCIe I/O-Bus-Infrastruktur Datenrate beträgt 8 Gbit/s. Bis zu 128 Kanäle (64PCIe I/O-Features) sind in einer I/O Drawer möglich. Die Drawer enthält 4 Switch Cards (zwei vorne, 2 hinten), und zwei DCAs für die redundante Stromversorgung.

4.3.7 Mainframe Frames

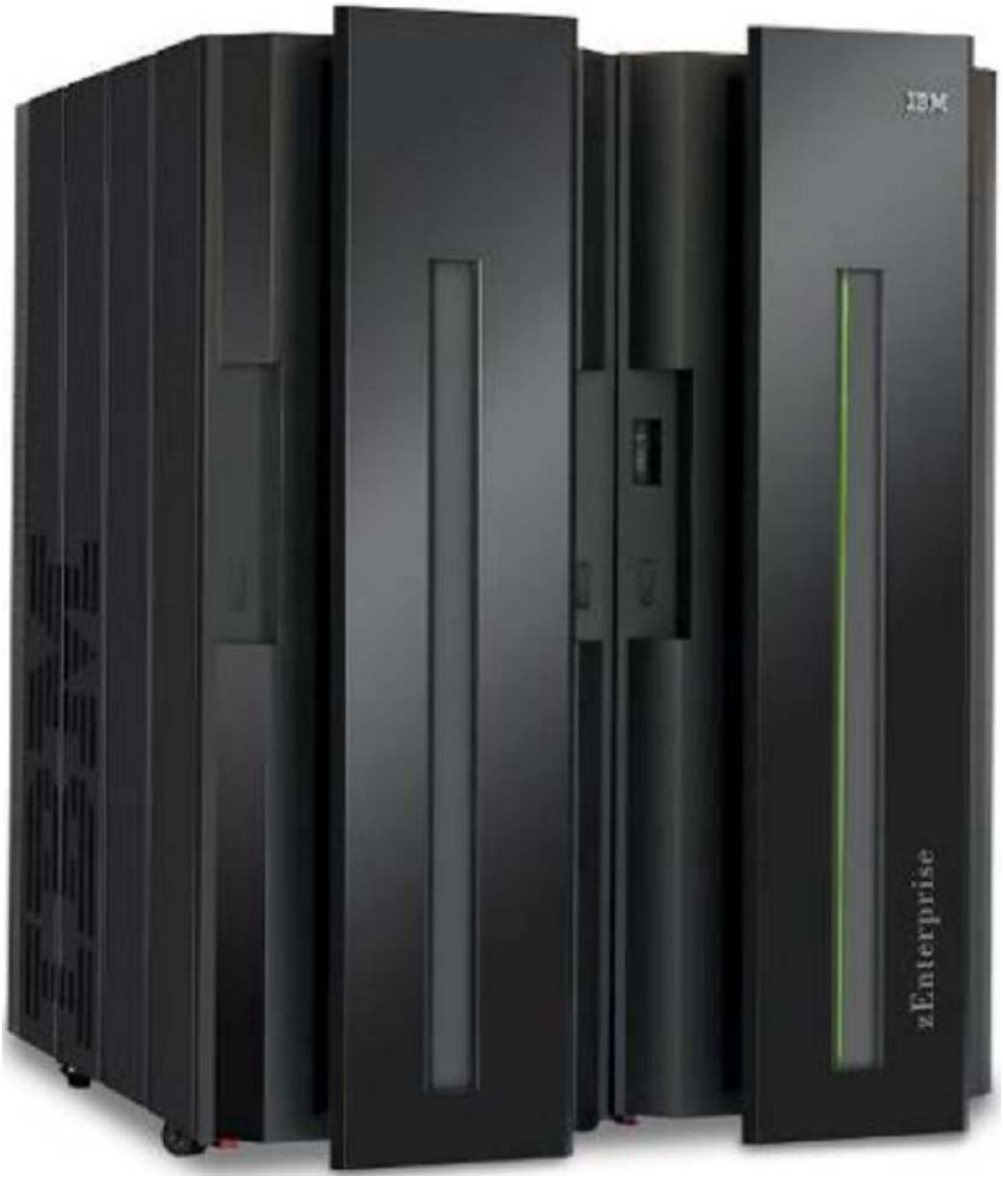


Abb. 4.3.9
Aussehen eines z196 Rechners

Ein z9, z10, z196 oder zEC12 Rechner besteht aus 2 meistens nebeneinander aufgestellten Rahmen, etwa 1,8 Meter hoch, welche von IBM als „Z Frame“ und „A Frame“ bezeichnet werden. Die Türen zu den beiden Frames sind künstlerisch gestaltet und enthalten viel leere Luft.

Ein zEC12 Rechner enthält bis zu 4 Books mit je 4 x 6 CPU Chips und 6 x 6 x 4 = 144 Prozessoren. Von diesen können 101 als CPUs eingesetzt werden, 16 arbeiten als „System Assisi Prozessoren“ (SAP; siehe Abschnitt 12.3.1) und 2 dienen als Reserve (Spares), die aktiviert werden können, wenn ein anderer Processor ausfällt.

Es sind 786 GByte Hauptspeicher pro Book möglich, insgesamt also 3 TByte für ein System mit 4 Books.

Es kann ein Rechner mit 1, 2, 3 oder 4 Books ausgeliefert werden. Unser z9 Rechner hat nur 1 Book.

Abb. 4.3.10
Aussehen eines zEC12 Rechners

Beim Entwurf der Türen für die zEC12 Rahmen durften sich die künstlerisch motivierten Designer austoben.

4.3. Geöffneter zEC12 Rechner

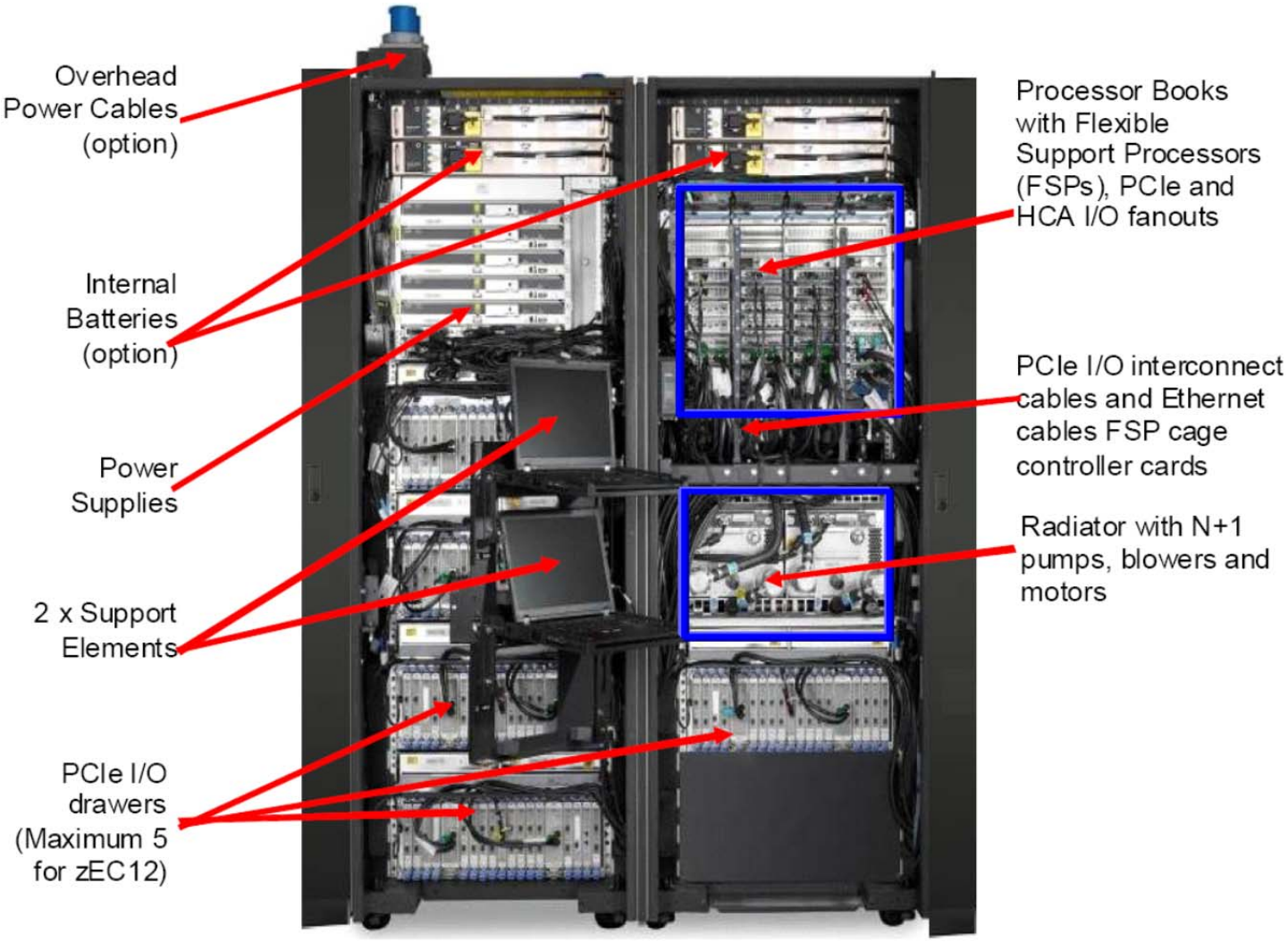


Abb. 4.3.11
Innenansicht eines zEC12 Rechners

Gezeigt ist ein geöffneter zEC12 Rechner (ohne Türen). Rechts oben sind 4 Books zu sehen mit Anschluss Steckern auf der Vorderseite. Hinter den Steckern sitzen Host Connector Adapter (HCA) Cards, welche die Verbindung zu dem MCM herstellen. Diese nehmen entweder PCIe Bus Kabel für die Verbindung zu den I/O Drawers, Infiniband Kabel für die Verbindung zu anderen Rechnern, oder FSB Kabel (diskutiert weiter unten) auf.

Zu beachten ist: Alle I/O Geräte, besonders auch Plattenspeicher, sind in getrennten Gehäusen untergebracht.

Ein Support Element (SE) ist ein herausklappbarer Laptop Computer, der innerhalb des linken Frames herausklappbar und fest verschraubt untergebracht ist. Er wird u.A. für die Initialisierung des z/OS Systems benutzt.

4.4 Weiterführende Information

4.4.1 Hauptspeicher DIMMs

512 MByte DIMM (Wikipedia)

Die Speicherkapazität eines Speichermoduls ergibt sich normalerweise als Produkt aus der Speicherkapazität der meist gleichartigen Speicherchips und deren Anzahl.

Als Beispiel sei hier ein Speichermodul genannt, das mit 16 Chips des Typs GM72V16821CT10K bestückt ist. Aus dem Datenblatt erfährt man, dass dieser Chip in zwei Bänken mit je 524.288 (= 219) Wörtern mit einer Wortbreite von jeweils 16 Bit organisiert ist (2×219×16). Daraus ergibt sich eine Speicherkapazität pro Chip von 2×219×16 bit = 224 bit = 16.777.216 bit. Mit 16 dieser Chips ergibt sich eine Speicherkapazität des Speichermoduls von 228 bit = 268.435.456 bit oder – mit 8 Bits pro Byte – 225 Byte = 33.554.432 Byte = 32 MByte.

Manche Speichermodule besitzen ein oder zwei zusätzliche Chips (gleichen oder anderen Typs), die für Fehlerkorrektur- bzw. Paritätsfunktionen zuständig sind. Hier werden für ein Byte häufig 9 Bits verwendet (8 Datenbits und 1 Prüfbit).

Der heute übliche DDR/DDR2-Speicher besitzt 64 Daten-Signalleitungen (beziehungsweise 72 bei ECC). Die einzelnen SDRAM-Chips sind so verschaltet, dass sie die gesamte Breite des Datenbusses belegen. Jeder Chip ist für bestimmte Datenleitungen zuständig. Ein Chip mit einer „×n“-Organisation kann n Datenleitungen versorgen. Für einen Datenbus mit 64 Leitungen sind folglich 64/n Chips mit der Organisation „×n“ erforderlich. Bei Modulen mit mehreren Bänken (siehe unten) sind mehrere Chips (2 oder 4) an den Datenleitungen parallel geschaltet. Folglich enthält ein Modul mit k Bänken 64/n×k Chips mit der Organisation „×n“.

Zusätzliche Eingangsleitungen regeln die Auswahl des Speicherbausteins (Chip Select) und die Schreib- bzw. Leserichtung (R/W) der Date

http://de.wikipedia.org/wiki/Speichermodul#Speicherkapazit.C3.A4t_.28Gr.C3.B6.C3.9Fe.29

Der zEC12 Rechner verwendet in der Maximalausrüstung DIMMs mit einer Kapazität von 32 GByte. Das sind 2GByte pro Speicherchip, oder 16 Gbit pro Speicherchip.

Eine Beschreibung des Memory Subsystems ist zu finden unter
http://www.cedix.de/VorlesMirror/Band1/MemorySubsystem.pdf

4.4.2 Multi-Layer Ceramics (MLC) Literatur

W. G. Burger, C. W. Weigel: Multi-Layer Ceramics Manufacturing. IBM Journal of Research and Development, Volume: 27 No.1,
Jan. 1983, p. 11 - 19

G. A. Katopis et al. : *MCM technology and design for the S/390 G5 system*. IBM Journal of Research and Development, Vol. 43, Nos. 5/6, 1999, p. 621.

A. J. Blodgett, D. R. Barbour: Thermal Conduction Module: A High-Performance Multilayer Ceramic Package.
Volume 26, Number 1, 1982, Page 30.

4.4.3 Mainframe Erdbeben Test

Die hohe Verfügbarkeit der Mainframes beruht auf einer sehr großen Anzahl einzelner Maßnahmen und Eigenschaften der Hardware, Software oder einer Kombination von beiden.

Sehr sichtbar ist dies, wenn man sich die Hardware anschaut. Alles ist grundsolide, kein Aufwand ist zu groß.

Ein Beispiel ist der „Erdbebentest“. Gefordert ist, dass bei einem Erdbeben mit dem Wert 9,0 auf der Richterskala der Mainframe Rechner dies ungestört überlebt, und dass spezifisch alle Software unbeeinträchtigt und ohne Absturz, (während des Bebens und hinterher) weiterläuft.

Das Video

http://www.cedix.de/VorlesMirror/Band1/earthquake.html

kann mit dem Microsoft Windows Media Player wiedergegeben werden. Das Download dauert einige Minuten. Es zeigt einen „Erdbebentest“, durchgeführt in der IBM Fabrik in Poughkeepsie, N.Y. Während des Testes laufen z/OS und alle anderen Subsysteme und Anwendungsprogramme ungestört weiter. Es ist auch direkt verfügbar unter http://www.cedix.de/VorlesMirror/Band1/eclipz z Earthquake test Aug 2007.wmv

Wenn Sie einen System z Rechner öffnen, fällt auf, wie solide der ganze mechanische Aufbau ist.

4.4.4 Eine interessante Vorführung:

Wer einen z196 Rechner einmal "auseinandernehmen" will: Hier gibt es die Möglichkeit! Klick auf Product Demonstration.

http://ibmtvdemo.edgesuite.net/servers/z/demos/zenterprise_196/index.html

Die Ausführung dieses sehr interessanten Videos erfordert im Browser Java enabled, sowie das Download eines Applets. Vielfach wird dies als ein Browser- Sicherheitsrisiko angesehen (die hier angegebene URL ist virenfrei). Benutzen Sie Chrome, wenn Firefox Probleme bereitet.

5. Input/Output

5.1 Festplattenspeicher Technologie

5.1.1 IBM 350 aus dem Jahre 1956

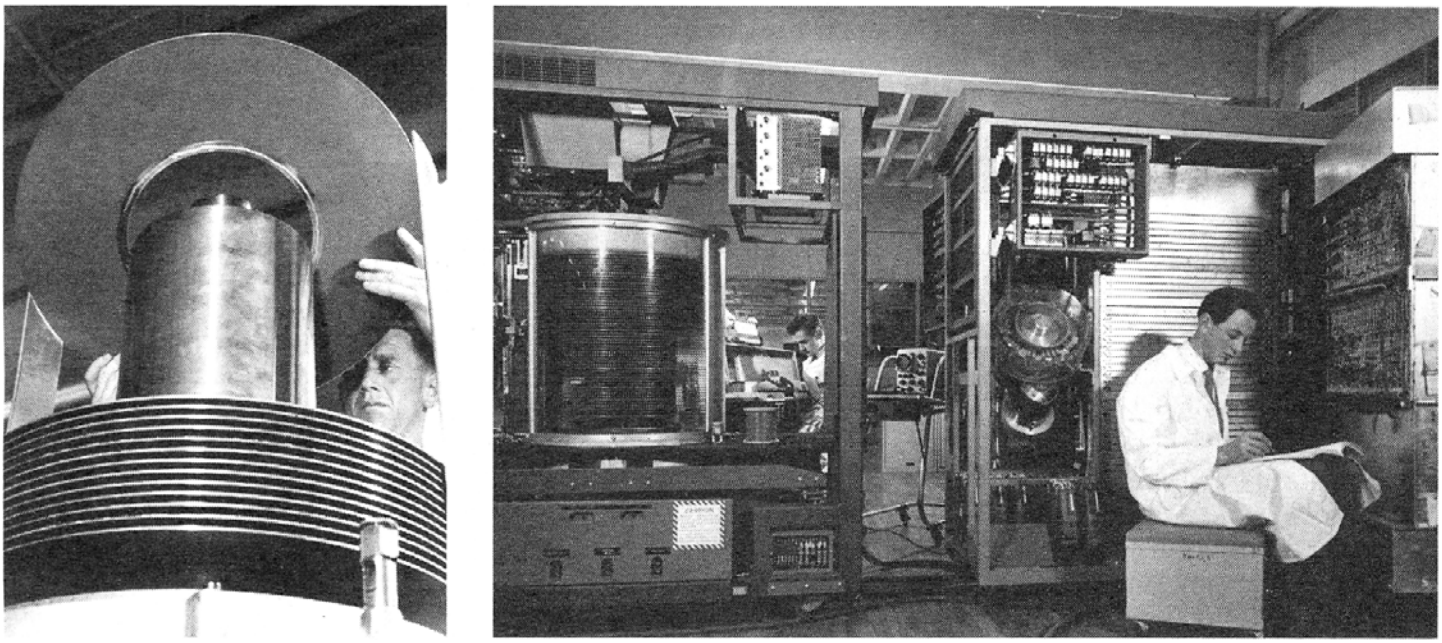


Abb. 5.1.1
IBM 350 Festplattenspeicher von 1956

Am 13. September 1956 stellte IBM das erste Festplattenspeichersystem der Welt vor, die "IBM 350 Plattenspeichereinheit", die zusammen mit dem "IBM 305 RAMAC" (Random Access Method of Accounting and Control)-Rechner ausgeliefert wurde. Der Festplattenspeicher im Kleiderschrankformat wog fast eine Tonne.

Auf einundfünfzig mit Eisenoxyd-beschichteten Platten mit einem Durchmesser von 61 cm (24-Zoll) speicherte die RAMAC sechs Millionen Zeichen. Hätte man damals schon Daten in Form von Bytes abgelegt, entspräche dies einer Kapazität von fünf Megabyte. Die IBM 350 Plattenspeichereinheit verbrauchte 2,5 Kilowatt an elektrischer Energie und wurde seinerzeit auf Mietbasis für 10 000 D-Mark pro Monat vertrieben.

Die modernen Festplattenscheiben verfügen nur über einen Durchmesser von 2,5 Zoll (6,35 cm) oder weniger. Das derzeit stärkste aktuelle Speichersystem der IBM, das System Storage DS8870, hat mit 2,3 PByte Kapazität eine um ca. 400 Millionen mal größere Speicherkapazität als die RAMAC.

5.1.2 Entwicklung der Festplattentechnologie

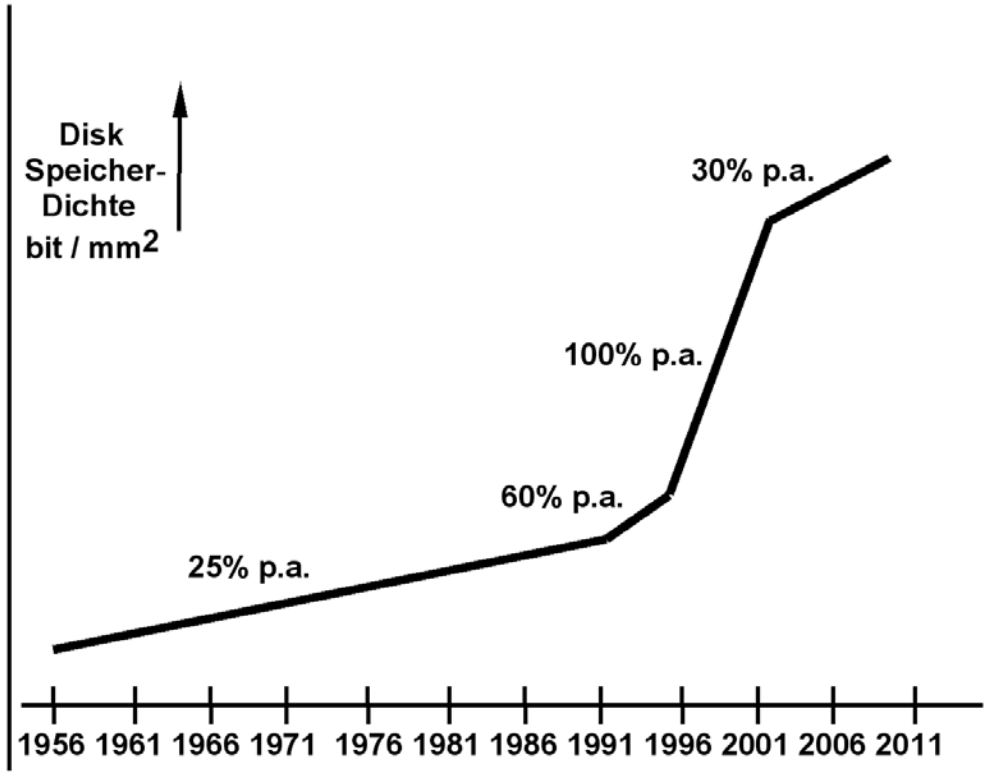


Abb. 5.1.2
Wachstumsrate der Festplatten Speicherdichte

Bei den magnetischen Festplattenspeichern wird der technologische Fortschritt daran gemessen, wie viele Bit auf einem Quadratmillimeter Oberfläche gespeichert werden können.

Dargestellt ist die 50 jährige Entwicklung mit dem jährlichen Wachstum der Plattenspeicherdichte. Auf der y-Achse ist die Speicherdichte mit einer logarithmischen Skala dargestellt. Das bedeutet, ein exponentielles Wachstum wird durch eine gerade Linie dargestellt.

Über mehrere Jahrzehnte betrug das Wachstum konstant etwa 25 % pro Jahr. In den 90er Jahren beschleunigte sich das Wachstum auf etwa 100 % pro Jahr. Ab Anfang der 2000er Jahre verlangsamt sich das Wachstum wieder auf etwa 30 % pro Jahr.

5.1.3 Baugruppen eines Festplattenspeichers

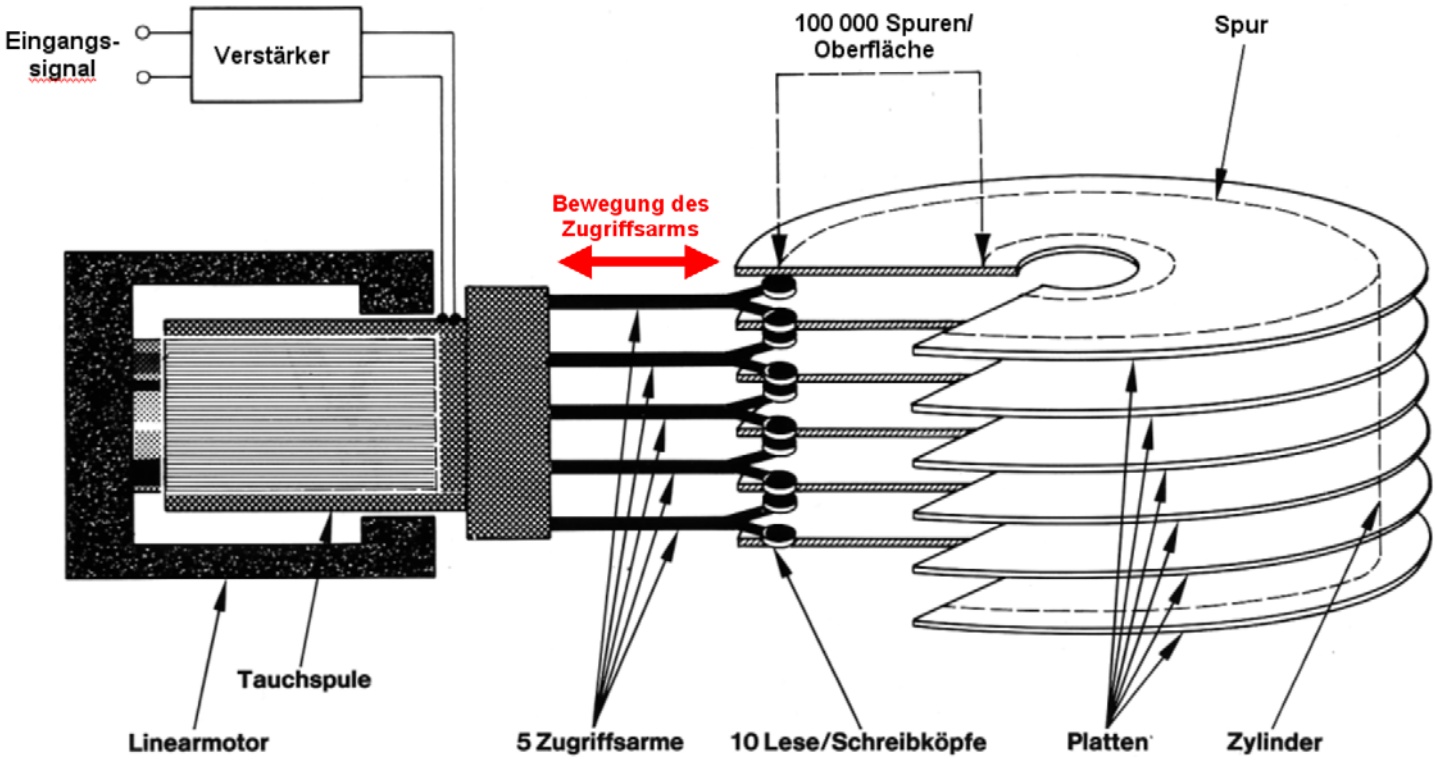


Abb. 5.1.3
Schnitt durch einen Festplattenspeicher

An der ursprünglichen Form eines Festplattenspeichers hat sich in den Jahrzehnten nicht viel geändert. Der Plattenspeicher besteht in der Regel aus mehreren Platten, von denen beide Oberflächen für die Aufzeichnung von Information genutzt wird. Die Information ist in kreisförmigen Spuren (Tracks) auf der Oberfläche angeordnet. Jeder Spur ist eine Spuradresse zugeordnet. Die Spuren selbst sind in Sektoren aufgeteilt. Die meisten Festplatten arbeiten heute mit einer Sektor Größe von 4096 Bytes. Früher waren 512 Byte Sektoren populär.

Pro Oberfläche gibt es mindestens einen Lese/Schreib-Kopf. Die Köpfe für alle Oberflächen sind auf einer gemeinsamen Zugriffsarmstruktur (slider) befestigt, welche alle Köpfe mittels eines Linearmotors (Actuator) auf eine der vielen Spuren bewegt.

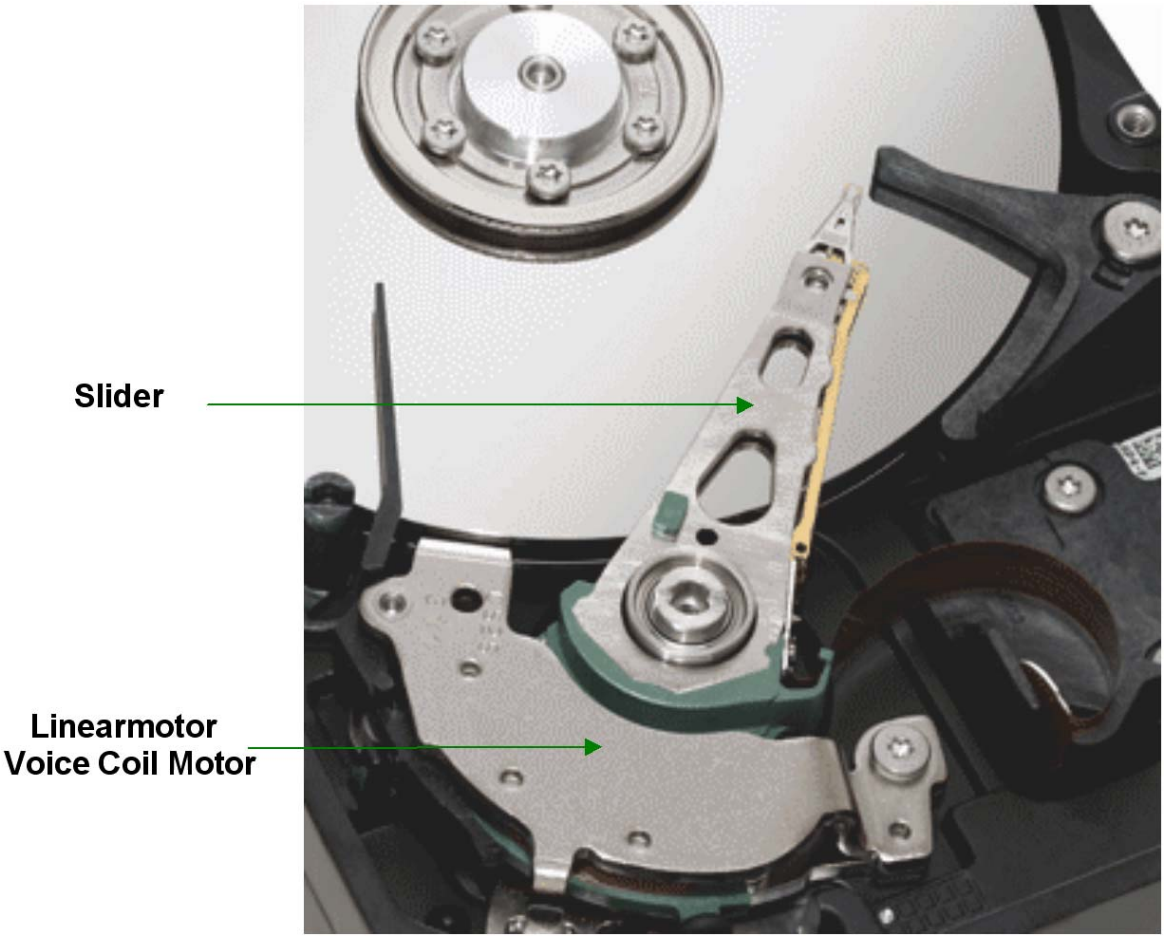


Abb. 5.1.4
Moderne Implementierung eines Festplattenspeichers

Abb. 5.1.4 zeigt eine Seagate Barracuda 7200.7 Festplatte der Firma Seagate (2006). Der Linearmotor bewegt sich senkrecht zur Achse des Sliders.

Eine Life Demo kann unter http://www.youtube.com/watch?v=L0nbo1VOF4M angeschaut werden.

Die Zugriffsarme (slider) werden durch einen Linearmotor angetrieben. Der Linearmotor wird oft auch als Voice Coil Motor bezeichnet, weil er das gleiche Prinzip benutzt wie der Linearmotor in einem Lautsprecher, der die Membrane antreibt. Eine Ansteuerungselektronik setzt eine Spuradresse in die entsprechenden Signale für den Linearmotor um.

5.1.4 Festplattenspeicher Zugriffszeiten

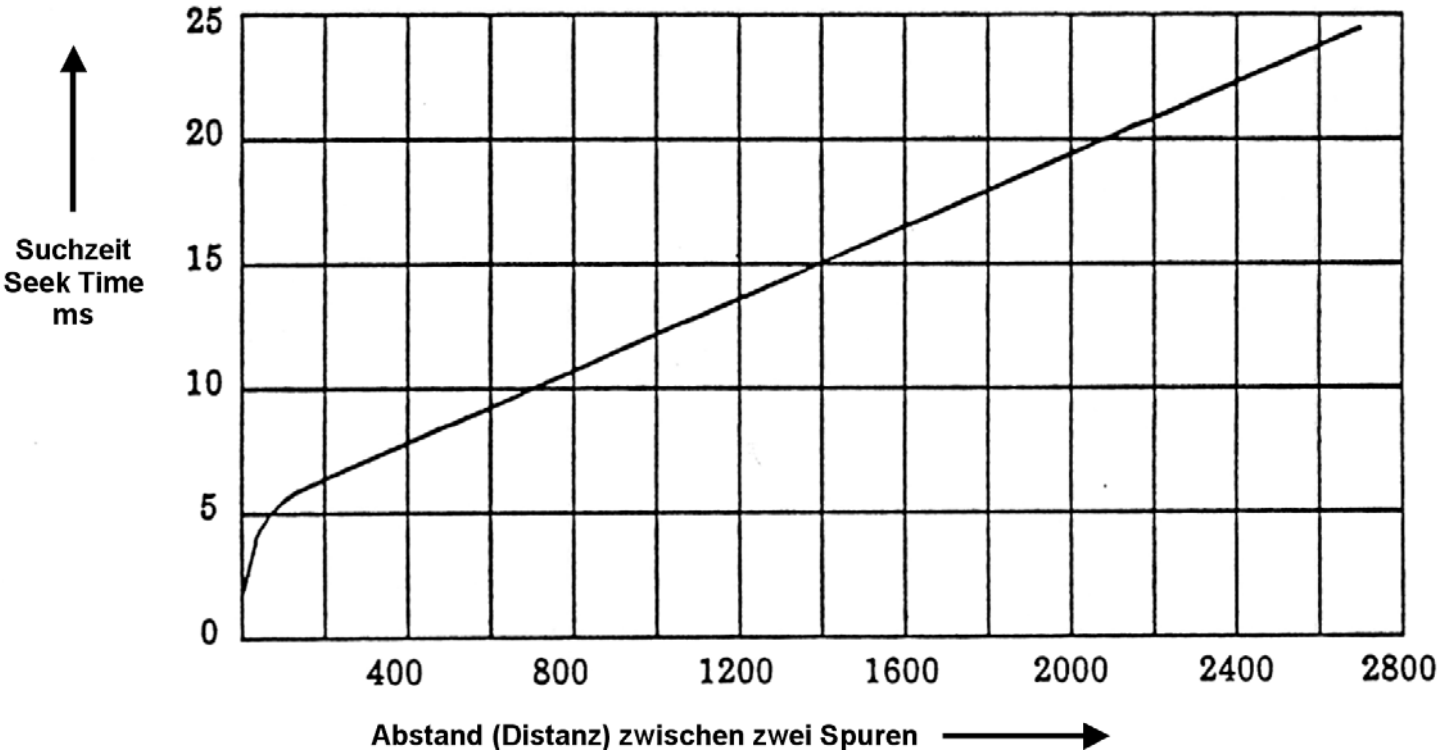


Abb. 5.1.5
Zugriffscharakteristik - Profil der Suchzeiten (Seek Time)

Die Bewegung durch den Linearmotor erfolgt relativ langsam. Der Wechsel von einer Spur auf eine unmittelbar benachbarte Spur kann mehrere Millisekunden betragen. Der Wechsel von einer Spur auf eine entfernte Spur kann bis zu 100 Millisekunden dauern. Diese Zeit ist kritisch für das Leistungsverhalten eines Plattenspeichers; sie wird als „Suchzeit“ (seek time) bezeichnet. Hersteller von Festplatten geben eine „mittlere Zugriffszeit“ an, die aus einer Mischung von vielen benachbarten und wenigen entfernten Zugriffen ermittelt wird. Zu fragen ist, ob diese Mischung realitätsnahe ist.

Die Speicherkapazität von Plattenspeichern ist in mehreren Jahrzehnten enorm gestiegen, von 5 MByte bei der ursprünglichen IBM RAMAC 305 bis zu mehren TByte heute (2013). Spuren werden sehr viel enger gepackt und auch die Anzahl der Bits auf einer Spur ist sehr viel größer geworden.

An der Zugriffscharakteristik hat sich aber nur wenig geändert.

Der Lese/Schreibkopf wird mit etwa 100 g beschleunigt und abgebremst (1 g = Anziehungskraft auf der Erdoberfläche). Jenseits von 100 g beginnt Metall sich zu verformen. Es ist daher in den letzten Jahrzehnten nicht möglich gewesen, die Geschwindigkeit der mechanischen Bewegung des Linearmotors deutlich zu verbessern.

Die Platte dreht sich gegenüber dem Kopf mit etwa $^2/_3$ Schallgeschwindigkeit. Es entsteht ein Luftstrom zwischen Platte und Kopf. Der Kopf hat eine Fläche von etwa 1 mm^2 · Er ist aerodynamisch wie der Flügel eines Flugzeuges geformt und erhält einen Auftrieb gegenüber der Plattenoberfläche. Der Auftrieb ist umso größer, je geringer die Entfernung Kopf – Platte ist. Eine Feder drückt den Kopf gegen die Plattenoberfläche. Es stellt sich eine Flughöhe ein, bei der Auftrieb und Federdruck im Gleichgewicht sind. Die Flughöhe beträgt etwa 50 nm.

Ein kleines Gedankenexperiment soll dies verdeutlichen. Vergrößern wir diese Struktur um einen Faktor 100 000. Das Ergebnis ist ein Flugzeug-ähnliches Gebilde mit einer Flügelweite von 100 Meter, dass mit nahezu Schallgeschwindigkeit mit einem Abstand von 5 mm über die Erdoberfläche fliegt. Jede Berührung des Kopfes mit der Plattenoberfläche bewirkt einen irreparablen Plattenspeichercrash.

5.1.5 Zylinder Organisation

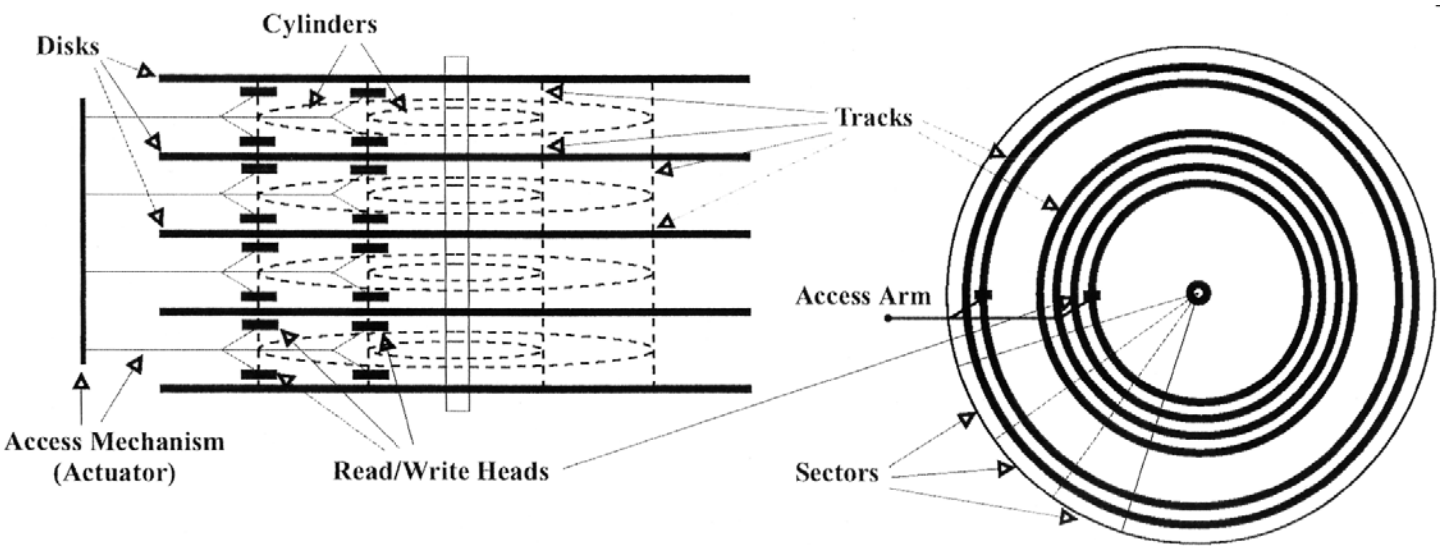


Abb. 5.1.6
a) Übereinander liegende Spuren

Um die Zugriffszeit zu verbessern werden häufig zwei (oder mehr) Lese/Schreibköpfe pro Plattenoberfläche eingesetzt. Dabei kann ein Kopf z.B. nur die äußeren Spuren und der andere Kopf nur die inneren Spuren abdecken. Hierdurch wird die maximale Distanz zwischen zwei Spuren halbiert.

Zusätzlich fasst man Spuren, die auf zwei oder mehr Oberflächen übereinander liegen, zu einem „Zylinder“ zusammen. Dadurch entstehen größere Speichereinheiten, die ohne eine Bewegung des Zugriffsarms adressiert werden können. Beim Wechsel von einer Spur zu einer anderen Spur des gleichen Zylinders muss nur die Elektronik der Lese/Schreibköpfe umgeschaltet werden, was im µs Bereich erfolgen kann.

Gebräuchliche Umdrehungsgeschwindigkeiten bei modernen Platten betragen:

3600 U/min	16,67 ms/Umdrehung
5400 U/min	11,11 ms/Umdrehung
7200 U/min	8,33 ms/Umdrehung
10 000 U/min	6 ms/Umdrehung
15 000 U/min	4 ms/Umdrehung

Bei diesen Umdrehungszeiten bewegt sich der Kopf gegenüber der Plattenoberfläche mit etwa $^2/_3$ Schallgeschwindigkeit. Bei einer größeren Geschwindigkeit würde der laminare Luftstrom zwischen Plattenoberfläche und Kopf abreißen und zu Turbulenzen führen. Dann kann der geringe Abstand zur Plattenoberfläche nicht mehr eingehalten werden.

Füllt man einen Plattenspeicher mit Helium statt mit Luft, sind höhere Umdrehungszahlen möglich. Die Schallgeschwindigkeit beträgt in Luft etwa 340 m/s, in Helium etwa 980 m/s .

In der Hardware und Betriebssystem-Software eines Rechners existieren sehr weitgehende Einrichtungen, die helfen, das an sich sehr schlechte Zugriffsverhalten eines Plattenspeichers zu verbessern. Es hat auch in den letzten Jahrzehnten viele Versuche gegeben, Plattenspeicher auf Grund Ihres schlechten Zugriffsverhaltens durch eine alternative Technologie zu ersetzen. Das ist über viele Jahre nicht erfolgreich gewesen. Mit der Einführung von Solid State Drives (SSD) wird sich das in der absehbaren Zukunft wahrscheinlich ändern.

5.1.6 Solid State Drive

Ein Solid State Drive (SSD,) ist ein Speichermedium, das wie eine herkömmliche magnetische Festplatte eingebaut und angesprochen werden kann, ohne eine rotierende Scheibe oder andere bewegliche Teile zu enthalten, da nur Halbleiterspeicherbausteine verwendet werden.

Der englische Begriff solid state in der Geschichte der Elektronik bedeutet, dass keinerlei bewegliche mechanische Teile wie Relais oder Röhren usw. verwendet werden, sondern Halbleiterbauteile, die durch die Entwicklungen in der Festkörperphysik möglich wurden. Insofern erscheint die Bezeichnung paradox, da ein eben gerade ohne bewegliche Teile auskommendes Medium als 'Drive' bzw. 'Disk' angesprochen wird. Dies geschieht in Analogie zu anderen Festplatten.

In SSD-Laufwerken werden in aller Regel Flash-Speicher Chips vom Typ "Nand" eingesetzt. Dabei wird die Information in Form von elektrischen Ladungszuständen der einzelnen Flash-Speicherzellen gespeichert. Das Speichern der Daten erfolgt nicht flüchtig, das heißt die Ladungen bleiben erhalten unabhängig davon, ob das Medium an die Stromzufuhr angeschlossen ist oder nicht. Das bedeutet, dass sich SSD-Speicher in der gleichen Art und Weise nutzen lassen wie Festplatten.

In den letzten Jahren werden zusätzlich zu (und bei mobilen Geräten an Stelle von) Plattenspeichern Solid State Drives eingesetzt.

Vorteile eines Solid State Drive sind Robustheit, schnelle Zugriffszeiten und Energieverbrauch. Nachteile sind Kapazität und Preis. In großen Mainframe Installationen findet man häufig zusätzlich zu den Plattenspeichern eine begrenzte Anzahl von SSDs für die Auslagerung von Daten für besonders performance-kritische Vorgänge.

Wenn sich SSDs als Alternative zu magnetischen rotierenden Plattenspeichern durchsetzen ist es denkbar, diese anders als herkömmliche Plattenspeicher zu nutzen.

5.1.7 Die Zukunft des Festplattenspeichers

Plattenspeicher behaupten ihre dominierende Rolle gegenüber SSDs, weil der Preisunterschied pro Byte Speicherkapazität etwa einen Faktor 10 beträgt.

Seit Jahrzehnten wird angenommen, dass SSDs in der Zukunft kostenmäßig gleichziehen werden. Das ist bis heute nicht geschehen.

Dennoch ist mit einem Zeithorizont von mehr als 10 Jahren anzunehmen, dass man dann den magnetischen Plattenspeicher als genauso archaisch ansehen wird, wie man heute die Lochkarte sieht. Dies wird in einem erheblichen Maße zu einer Änderung der Betriebssystem Struktur führen.

Panta rhei (griechisch ‚πάντα ῥεῖ'), „Alles fließt") sagte schon der griechische Philosoph Heraklit.

5.1.8 Direct Memory Access

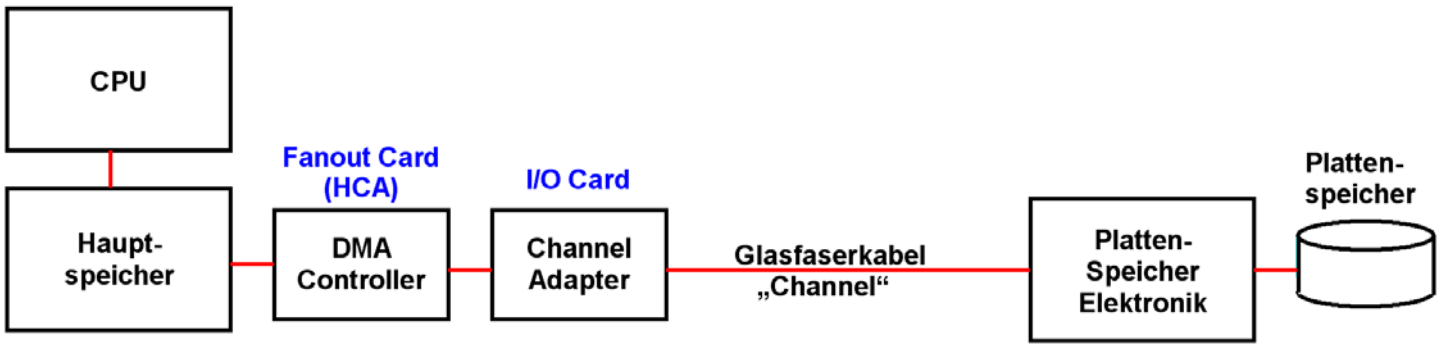


Abb. 5.1.7
Baugruppen einer Festplattenverbindung

Plattenspeicher werden mittels eines „Direct Memory Access" (DMA) Controllers an den Hauptspeicher angeschlossen. In einem zEC12 Mainframe ist dies eine Fanout Card innerhalb eines Books. Der DMA Controller (die Fanout Card) ist wiederum mit einer Channel Adapter Card in einem I/O Drawer verbunden.

Die Channel Adapter Card stellt mittels eines Glasfaserkabels die Verbindung zur Elektronik eines Plattenspeichers her. Das Glasfaserkabel wird als Channel, Ficon Channel oder Channel Path bezeichnet (die Begriffe sind weitgehend austauschbar). Ein Mainframe kann mehrere 100 Glasfaseranschlüsse aufweisen. Die Distanz Rechner – Plattenspeicher kann viele Kilometer betragen.

Plattenspeicher werden in getrennten Gehäusen, sog. „Enterprise Storage Servern" (ESS) untergebracht. Ein ESS kann Hunderte von Plattenspeichern enthalten. Für die Ansteuerung enthält ein ESS mehrere Spezialrechner, deren Firmware von außen nicht zugreifbar ist.

Der DMA Controller ist Bestandteil der Fanout Card (Host Channel Adapter). Der Channel Adapter ist als I/O Card implementiert, und befindet sich in einem PCIe I/O Drawer, siehe Abschnitt 4.3.6.

5.1.9 DMA Steuerung

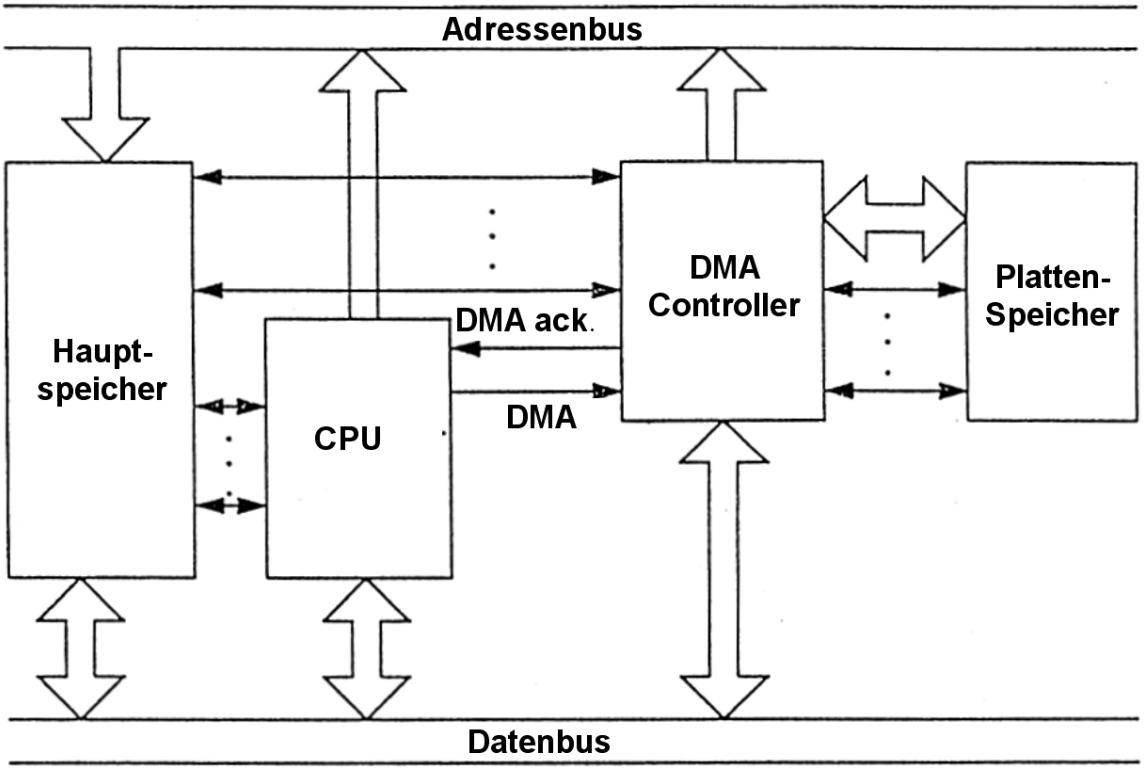


Abb. 5.1.8
Paralleler Zugriff auf den Hauptspeicher durch die CPU(s) und den DMA Controller

Der Processor (CPU) und der DMA (Direct Memory Access) Controller greifen mit Adress-, Daten- und Steuerleitungen gleichzeitig und parallel auf den Hauptspeicher zu. Jeder Hauptspeicherzyklus wird entweder von der CPU oder von dem DMA Controller genutzt.

Die Memory Bus Adapter (MBA) in dem z9 Rechner, bzw. Host Channel Adapter (HCA) in den z10/z196/zEC12 Rechnern werden gemeinsam auch als Fanout Card bezeichnet, und enthalten einen DMA Controller.

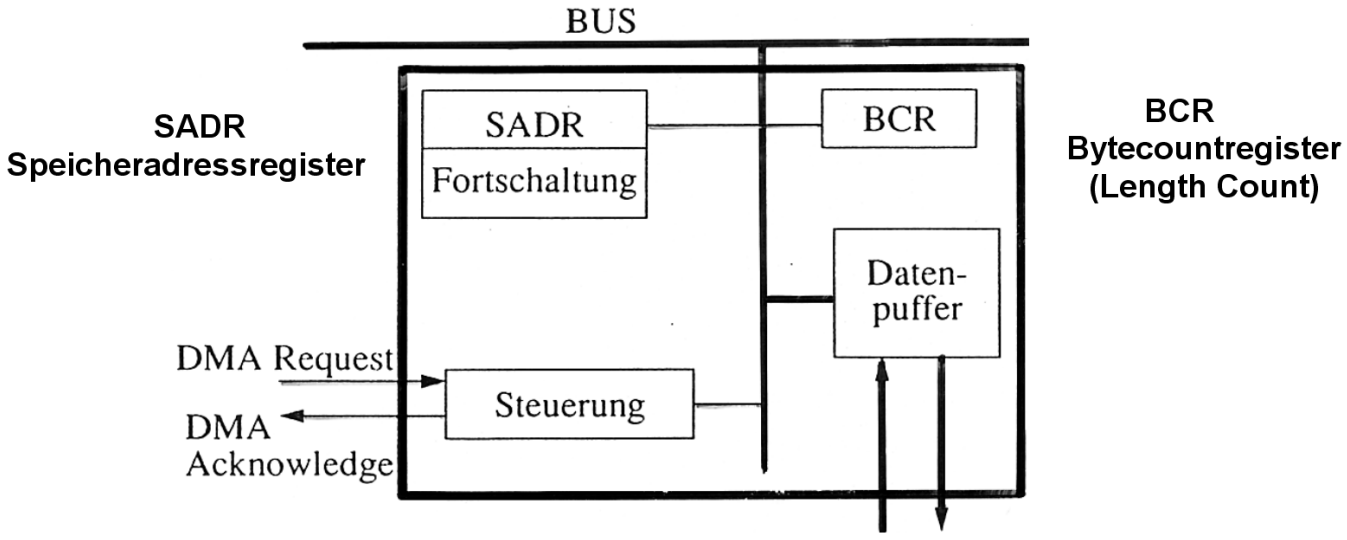


Abb. 5.1.9
Storage Address Register und Byte Count Register

Aufgaben der DMA-Steuerung:

- **Adressieren des Hauptspeichers durch Adressfortschaltung**
- **Adressieren der Geräteschnittstelle**
- **Steuerung der Buszugriffe für Lesen oder Schreiben**
- **Zählen der übertragenen Bytes**
- **Rückmelden an CPU**

Das Storage Address Register (SADR) enthält die Hauptspeicheradresse des zu übertragenden Bytes. Für die Übertragung eines größeren Datenblocks wird das SADR automatisch hochgezählt. Das Byte Count Register (BCR) enthält die Länge des zu übertragenden Datenblocks.

Für einen Übertragungsvorgang werden DMA-Steuerung und I/O Controller (I/O card) von der CPU initialisiert, z.B. Laden der Steuer(Control)- und Adressregister.

5.1.10 Unterbrechungsgesteuerte Ein/Ausgabe

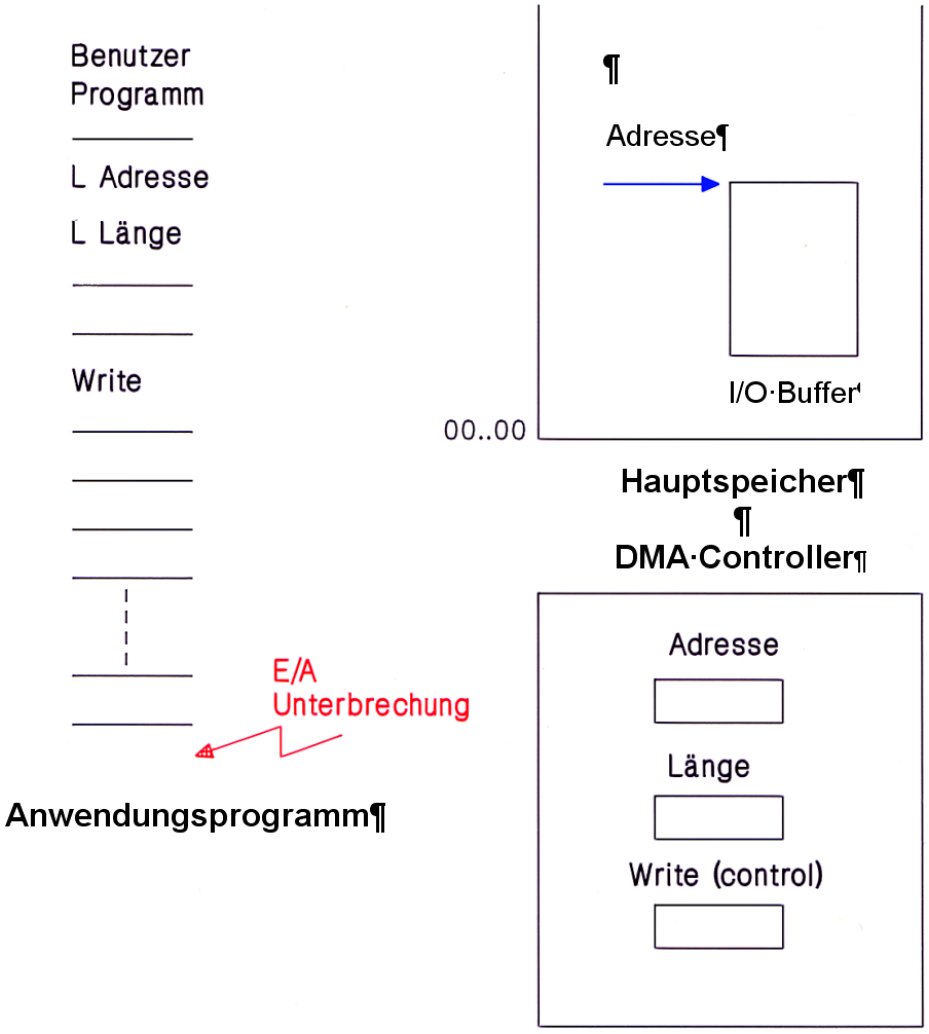


Abb. 5.1.10
Das Adress Register des DMA Controllers enthält die Hauptspeicheradresse des I/O Puffers

Die zu übertragenden Daten befinden sich in einem als I/O Buffer bezeichnetem Bereich des Hauptspeichers. Das Anwendungsprogramm definiert die Adresse und Länge des I/O Buffers,, und führt z.B. einen WRITE Befehl aus.

Der DMA Controller übernimmt Adresse und Länge in seine SADR und BCR Register und setzt sein Control Register auf Write.

Danach liest er automatisch und unabhängig von der CPU die Daten aus dem I/O Puffer und überträgt sie zum Plattenspeicher. Nach der Übertragung eines jeden Bytes wird der Length Count in dem BCR Register (Byte Count Register) heruntergezählt und das SADR Register wird inkrementiert. Die CPU arbeitet während dieser Zeit unabhängig weiter.

Wenn der Wert in dem Längenregister den Wert 0 erreicht, ist die I/O Operation beendet. Dies wird der CPU über eine I/O Unterbrechung mitgeteilt.

5.2 SCSI und FICON

5.2.1 Moderne Plattenspeicher-Anschlussarten

Moderne Plattenspeicher werden heute fast ausschließlich über ein serielles Protokoll angeschlossen. Die wichtigsten Protokolle sind:

- **SATA (serial ATA)** — **Nachfolger für parallel ATA (anderer Name ist IDE)**
- **SAS (serial attached SCSI)** — **Nachfolger für parallel SCSI**
- **iSCSI** — **Internet SCSI (benutzt Ethernet, von Mainframes nicht unterstützt)**
- **FC-SCSI** — **Fibre Channel SCSI**

SATA dominiert beim PC und anderen Arbeitsplatzrechnern. Die verschiedenen SCSI Arten dominieren bei Servern. Fibre Channel SCSI kann sowohl für Punkt–zu–Punkt Verbindungen als auch als FC-AL Version (Fibre Channel Arbitrated Loop) eingesetzt werden.

Seagate z.B. bietet z.B. (2010) die Barracuda Familie von Plattenspeichern mit der SATA und der SAS Schnittstelle an; Speicherkapazität bis zu 4,0 TByte. Für „Mission Critical Applications“ ist die Cheetah Familie von Plattenspeichern mit einer 4-Gb/s Fibre Channel interface verfügbar; Speicherkapazität bis zu 600 GByte. Cheetah Plattenspeicher sind laut Seagate für Anwendungen vorgesehen, „where system availability and reliability is of utmost importance“.

Letzteres ist vor allem bei Mainframes gegeben. Dafür wird die deutlich geringere Speicherkapazität in Kauf genommen. In anderen Worten: Plattenspeicher in Ihrem PC haben pro Einheit in der Regel eine deutlich höhere Speicherkapazität als die Plattenspeicher eines Mainframes.

5.2.2 S/360 Channel und SCSI

Abb. 5.2.1
b) Historische Entwicklung OEMI Channel und SCSI

Für den Anschluss von I/O Geräten System führte das System/360 im Jahre 1964 den „Selektor“ und 1970 den „Block Multiplex“ Kanal ein. IBM veröffentlichte diese I/O Schnittstelle unter dem Namen OEMI (Original Equipment Manufacturer Interface) , was dazu führte, dass viele unabhängige Hersteller I/O Geräte für den Anschluss an die damaligen Mainframes entwickelten und installierten. Das ist auch heute noch der Fall.

Ab 1882 wurde auf Initiative der Firmen Shugart und NCR eine Modifikation des OEMI Standards durch das ANSI (American National Standards Institute) unter dem Namen SCSI (Small Computer System Interface) für den Einsatz in kleineren Systemen veröffentlicht. Hierbei wurde die auf extreme Zuverlässigkeit ausgelegte OEMI Verkabelung (elektrische Interface) vereinfacht. Die logische Interface wurde weitestgehend beibehalten.

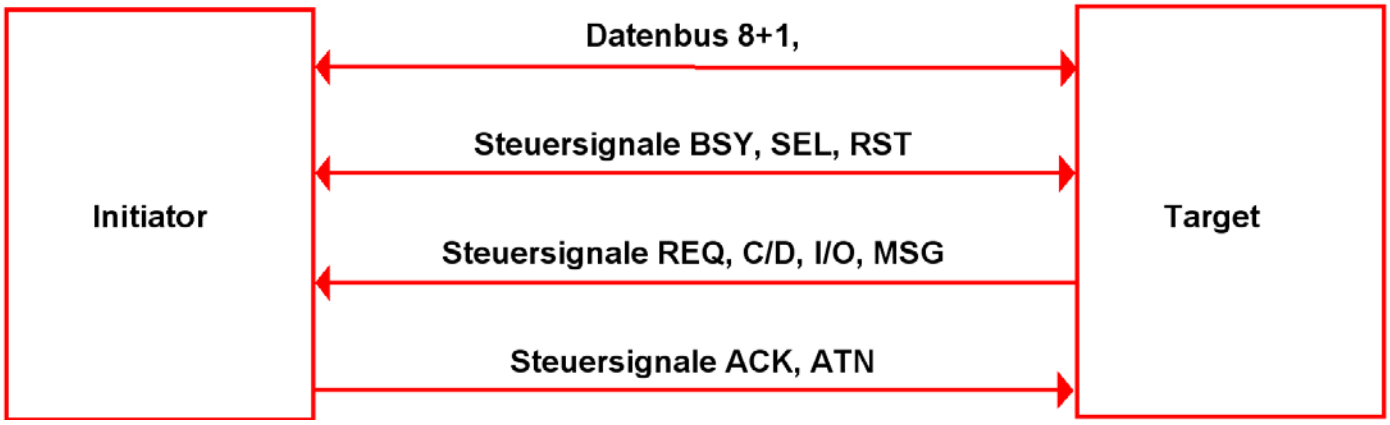


Abb. 5.2.2
OEMI Schnittstelle

Die logische OEMI und SCSI Interface bestand aus einem 8 Bit (plus Parity) Datenbus sowie einer Reihe von teils unidirektionalen, teils bidirektionalen Steuersignalen. Die OEMI Schnittstelle wird heute von IBM als „Parallel Channel“ bezeichnet. Ein Parallel Channel hat eine Datenrate von max. 4.5 MByte/s. und überbrückt Distanzen bis zu 130 m. Der Parallel Channel benutzt zwei Kupfer Kabel: *Bus* und *Tag*. . Ein Bus Kabel überträgt Information (ein

Byte in jeder Richtung). Daten auf dem Tag Kabel definieren die Bedeutung der Information auf dem Bus Kabel.

Später entwickelten sich die OEMI und SCSI Schnittstellen unabhängig voneinander weiter. Der ursprünglich 8 Bit breite Datenbus der SCSI-1 Schnittstelle wurde auf 16 und später 32 Bit verbreitet. Anschließend entstanden serielle Versionen, die bei IBM zu den Glasfaser-gestützten ESCON und dann den FICON Kanälen führten. Bei SCSI entstanden die Serial SCSI und die Glasfaser FC-SCSI Versionen.

Wegen der höheren Anforderungen im Großrechnerbereich hatten die IBM Kanäle immer einen deutlich höheren Funktionsumfang als die SCSI Schnittstellen. Auch heute kann der mit einem FICON Netzwerk erzielbare Durchsatz durch ein SCSI Netzwerk nicht erreicht werden.

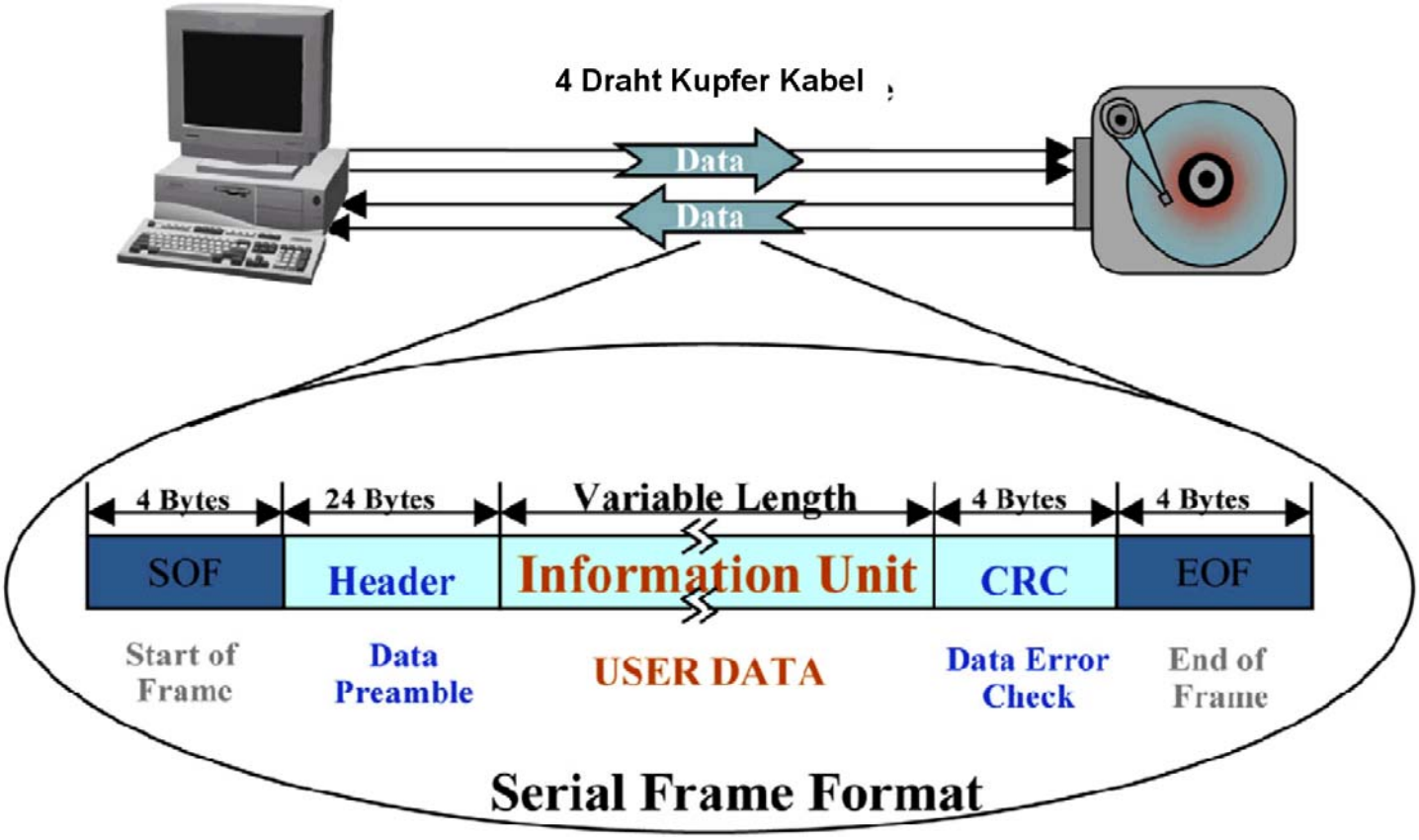


Abb. 5.2.3
Serial SCSI

Die ursprünglich parallel übertragenen OEMI oder SCSI Daten werden bei SAS (serial attached SCSI) in einen Rahmen (Frame) gepackt und über ein Kupferkabel seriell übertragen. Die Fibre Channel SCSI (FC-SCSI) Version verwendet statt dessen zwei Glasfaserkabel (für Hin- und Rückleitung.

Das aus dem Parallel Channel (OEMI) entwickelte serielle Fibre Channel FICON Protokoll benutzt ebenfalls Glasfasern. Es hat im Vergleich zu FC-SCSI einen wesentlich höheren Funktionsumfang und damit eine bessere I/O Leistung.

Der Parallel Channel dominierte die Mainframe I/O Konfigurationen bis zum Anfang dieses Jahrhunderts. Er hat viel Ähnlichkeit mit der parallelen SCSI Interface. Heutige Mainframes unterstützen den Parallel Channel nicht mehr.

Serielle Channels haben den älteren parallel Channels abgelöst. Es existieren zwei Serial Channel Typen:

- **Der (ältere) „ESCON Channel erlaubt Datenraten von 17 MByte/s. Er wurde 1990 eingeführt, und wird in zukünftigen Mainframe Modellen nicht mehr verfügbar sein.**
- **Ein FICON Channel erlaubt Datenraten bis zu 800 MByte/s. Das FICON Protokoll wurde 1997 eingeführt.**

Die über 10 Jahre alte „Shark“ Plattenspeichereinheit unseres eigenen Mainframe Rechners jedi.informatik.uni-leipzig.de wurde über ESCON Verbindungen angeschlossen. Unsere neuere DS6800 Plattenspeichereinheit verwendet FICON Verbindungen.

Serielle Channel verwenden Glasfaser Kabel an Stelle von Kupferverbindungen. Es können Entfernungen bis zu 100 km überbrückt werden. Außerdem verfügen sie über eine erweiterte I/O Adressierung.

Formal wird ein Channel als „Channel Path“ bezeichnet, und durch einen 8 Bit CHPID (Channel Path Identifier) gekennzeichnet. In der Umgangssprache werden Channel Path nach wie vor als Kanäle (Channels) bezeichnet, und auch Experten kennen den Unterschied nicht genau.

5.2.3 Fibre Channel

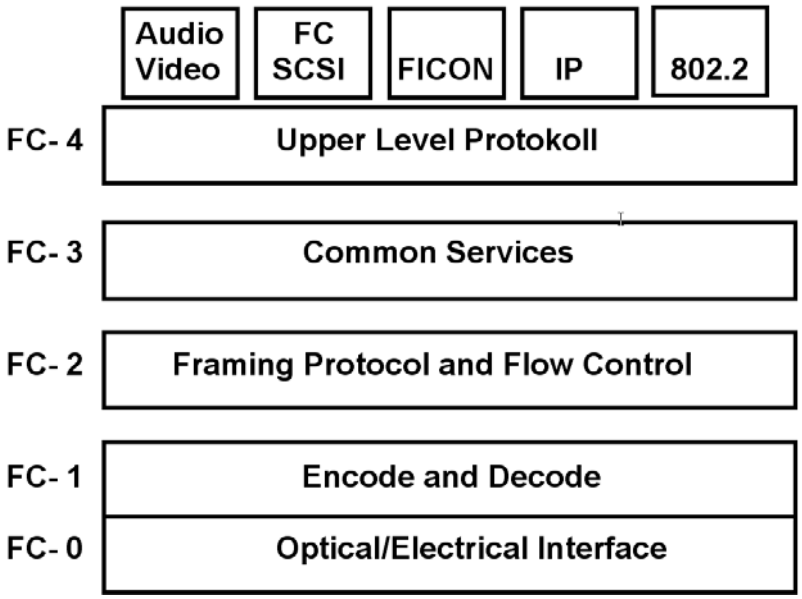


Abb. 5.2.4
c) Fibre Channel Architektur

Fibre Channel (FC) ist für serielle, kontinuierliche Hochgeschwindigkeitsübertragung großer Datenmengen konzipiert worden. Die erreichten Datenübertragungsraten liegen heute bei 8 Gbit/s, was im Vollduplex-Betrieb für Datentransferraten von 800 MB/s ausreicht. Als Übertragungsmedium findet man gelegentlich Kupferkabel (hauptsächlich innerhalb von Storage-Systemen; überbrückt bis zu 30 m), meistens aber Glasfaserkabel. Letzteres wird vor allem zur Verbindung von Rechnern mit Storage-Systemen oder aber von Storage-Systemen untereinander eingesetzt. Hierbei werden Entfernungen bis zu 10 km überbrückt. Der Zugriff auf die Festplatten erfolgt blockbasiert.

Es können generell zwei Arten von Fibre-Channel-Implementierungen unterschieden werden, die Switched Fabric, die meist als Fibre Channel Switched Fabric (FC-SW) bezeichnet wird und die Arbitrated Loop, kurz als FC-AL bekannt.

Bei der Fibre Channel-Switched Fabric werden Punkt-zu-Punkt-Verbindungen (Point To Point) zwischen den Endgeräten geschaltet. Beim Fibre Channel-Arbitrated Loop handelt es sich um einen logischen Bus, bei dem sich alle Endgeräte sich die gemeinsame Datenübertragungsrate teilen.

Das Fibre Channel-Switched Fabric ist die leistungsfähigste und ausfallsicherste Implementierung von Fibre Channel. In den meisten Fällen ist Switched Fabric gemeint, wenn nur von Fibre Channel gesprochen wird. Im Zentrum der Switched Fabric steht der Fibre Channel Switch (von IBM als „Director“ bezeichnet). Über dieses Gerät werden alle anderen Geräte miteinander verbunden, so dass es über den Fibre Channel Switch möglich wird, direkte logische Punkt-zu-Punkt-Verbindungen zwischen je zwei beliebigen angeschlossenen Geräten zu schalten.

FC-AL erlaubt es, bis zu 127 Geräte an einem logischen Bus zu betreiben. Dabei teilen sich alle Geräte die verfügbare Datenübertragungsrate (bis 8 GBit/s). Die Verkabelung kann sternförmig über einen Fibre Channel Hub erfolgen. Es ist auch möglich, die Geräte in einer Schleife (Loop) hintereinander zu schalten (Daisy Chain), da viele Fibre-Channel-Geräte über zwei Ein- bzw. Ausgänge verfügen. Dies ist z.B. beim IBM DSS 8700 Enterprise Storage Server der Fall.

Die Fibre Channel Architektur verwendet ein Schichtenmodell, vergleichbar mit (aber unabhängig von) den TCP/IP oder OSI Schichtenmodellen. Die unterste Schicht verwendet in den meisten Fällen optische Kabel. Wichtig ist besonders die oberste Schicht FC4. Hierüber ist es möglich, unterschiedliche Protokolle zu betreiben (siehe Abb. 5.2.4).

FC-SCSI ist eine serielle Form des SCSI Protokolls, die über Fibre Channel Verbindungen erfolgt. (Das als „Serial SCSI“ bezeichnete Protokoll benutzt keinen Fibre Channel).

FICON ist das universell von Mainframes eingesetzte Protokoll, um Rechner miteinander und mit I/O Geräten zu verbinden.

IP over Fibre Channel und Ethernet over Fibre Channel (IEEE 802.2) wurde standardisiert, wird aber nur wenig benutzt.

Ein spezieller Fibre Channel Adapter wird für die Echtzeitübertragung von Fernsehprogrammen benutzt.

Literatur:
http://www.answers.com/topic/fibre-channel?cat=technology

In der Praxis verwenden SCSI Platten bessere mechanische und elektronische Komponenten als SATA Platten. Dies bewirkt:

- **schnellere Zugriffszeiten**
- **höhere Zuverlässigkeit**
- **deutlich höhere Kosten**

Aus Zuverlässigkeitsgründen verwenden SCSI Platten selten die neueste Plattenspeichertechnologie. Dies ist einer der Gründe, warum für einen Personal Computer Plattenspeicher mit einer höheren Speicherkapazität erhältlich sind als dies im Mainframe Bereich der Fall ist.

FATA-Laufwerke sind SATA-Plattenlaufwerke mit einem Fibre Channel (FC) Anschluss. Sie arbeiten mit den mechanischen Komponenten von ATA-Festplatten, jedoch mit vorgeschalteter FC-Schnittstelle. In anderen Worten, es sind SATA Platten, bei denen die elektrische serielle ATA Schnittstelle durch eine Fibre Channel Schnittstelle ersetzt wurde. Die FATA-Technologie hat den Vorteil, dass sie in einer Mischung mit anderen FC-Laufwerken betrieben werden können.

FATA Platten werden auch als „Nearline-Platten“ bezeichnet. Sie werden im Großrechnerbereich dann eingesetzt, wenn Zuverlässigkeit und Zugriffszeit weniger wichtig sind, z.B. um Bilddateien (Images) zu archivieren.

5.2.4 Plattenspeicher Anschluss Alternativen

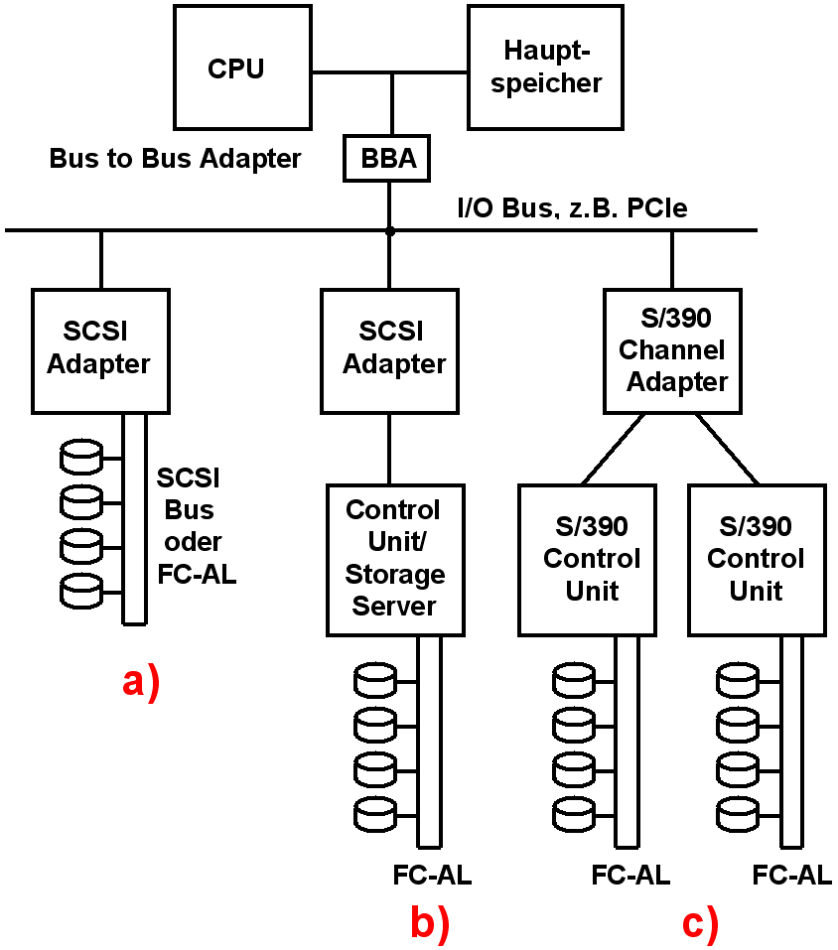


Abb. 5.2.5
Mögliche Plattenspeicher Anschlüsse an einen Server

a) Ein SCSI Adapter kann eine oder mehrere SCSI Platten mit dem I/O Bus eines Rechners verbinden.

b) Bei größeren Mengen an Plattenspeichern sind diese über eine Control Unit oder einen Storage Server mit dem SCSI Adapter verbunden.

c) Mainframes ersetzen den SCSI Adapter durch einen Channel Adapter. Plattenspeicher sind grundsätzlich über Control Units mit dem Channel Adapter verbunden .

5.2.5 S/360 I/O Konfiguration

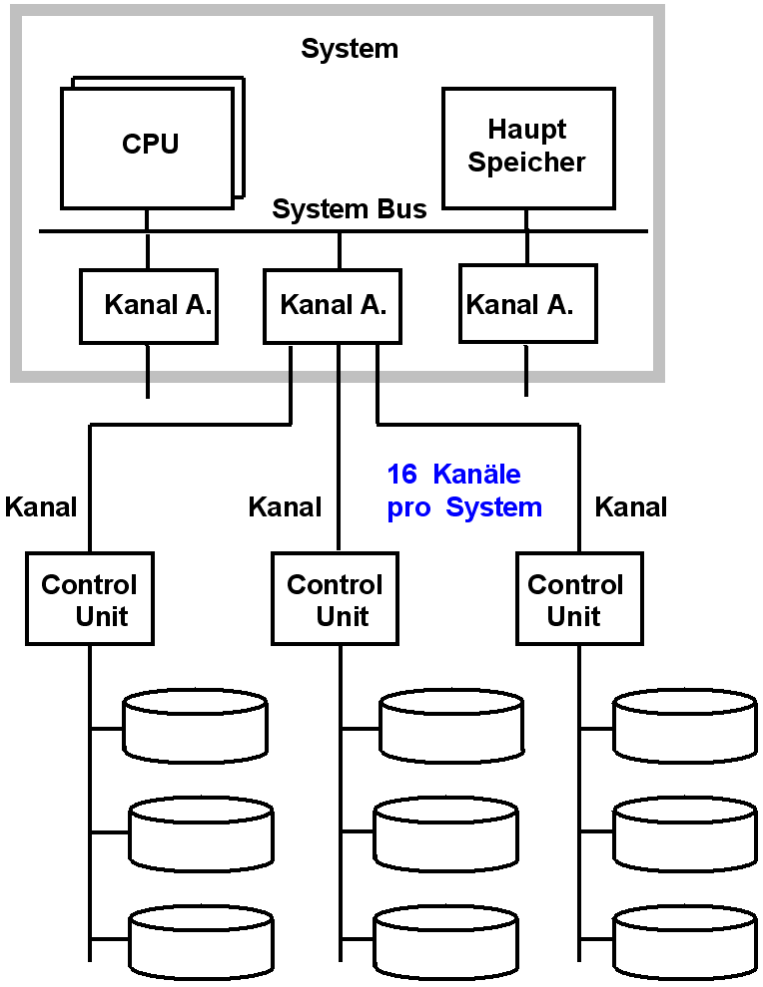


Abb. 5.2.6
S/360 Plattenspeicher Anschluss

Dargestellt ist die ursprüngliche S/360 I/O Konfiguration.

Plattenspeicher sind über Control Units, Kanal-Verbindungskabel (Channel Cables) und Kanal-Adapter mit dem Hauptspeicher des Systems verbunden. Die Verbindungskabel des „Parallel Channels“ waren bis zu 400 Fuß (130 m) lang. Die Kanal-Adapter konnten mittels DMA direkt auf den Hauptspeicher des Systems zugreifen.

Die Control Unit führte Befehle aus, die vom Kanal-Adapter aus dem Hauptspeicher ausgelesen und zwecks Ausführung an die Control Unit übergeben wurden.

Magnetbandgeräte und Drucker werden ähnlich wie Plattenspeicher über Control Units an einen Mainframe Rechner angeschlossen.

5.2.6 Aufgaben der Plattenspeicher-Steuereinheit

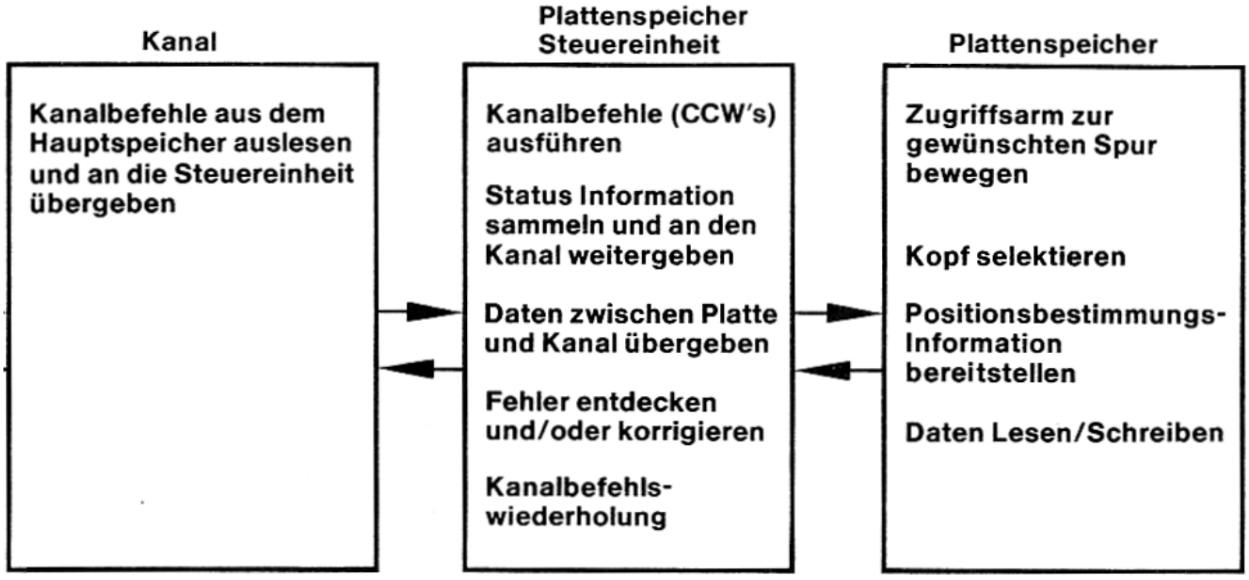


Abb. 5.2.7
Funktionen von Kanal Adapter, Steuereinheit und Festplatte

Dargestellt ist die Aufgabenaufteilung zwischen Kanal-Adapter, Steuereinheit (Control Unit) und der Plattenspeicher-Elektronik.

Der Kanal (die Kanal-Adapter Karte) ist nur dazu da, um stellvertretend für die Steuereinheit per DMA Daten und Kanalbefehle (CCWs) aus dem Hauptspeicher auszulesen und an die entfernte Steuereinheit weiter zu geben.

Der Plattenspeicher selbst enthält umfangreiche elektronische und logische Komponenten. Sie führen Steuerfunktionen aus wie:

- **Zugriffsarm auf die richtige Spur bewegen**
- **Verifizieren, dass dies geschehen ist**
- **Einen von mehreren Lese/Schreib-Köpfen (und damit eine Plattenoberfläche) selektieren**

5.2.7 Was ist Firmware ?

Komponenten, die früher mittels hart verdrahteter Transistorlogik erstellt wurden, verwenden heute häufig statt dessen einen dedizierten Mikroprozessor mit speziellem Code. Dieser Code hat die Eigenschaft, dass ein normaler Benutzer nicht darauf zugreifen, ihn ändern oder erweitern kann. Derartiger Code wird wahlweise als Microcode oder als Firmware bezeichnet ist.

Firmware Code eines Mainframes wird von Prozessoren mit der System z Architektur ausgeführt. Microcode eines Mainframes wird von nicht-System z Prozessoren ausgeführt, z.B. von PowerPC Prozessoren. Auf den in Abschnitt 12.3 erwähnten SAPs (System Assist Prozessor) läuft Firmware; auf der I/O Card befindet sich ein PowerPC Prozessor, der Microcode ausführt.

Außerhalb der Mainframes Welt ist der Unterschied zwischen Firmware und Microcode weniger sauber definiert. Firmware/Microcode läuft z.B. auf dem Prozessor, der einen WLAN Access Point, der Mikroprozessor Steuerung einer Geschirrspülmaschine oder der Mikroprozessor Steuerung des elektrischen Fensterhebers Ihres Mercedes S-Klasse Autos steuert.

Das in Abb. 5.2.8 erwähnte „Channel Subsystem“ besteht aus „System Assist Processoren (SAP)“ plus Firmware. Control Units und Enterprise Storage Server verwenden sehr leistungsfähige Prozessoren aber keine Firmware, da ihr Code nicht auf System z Prozessoren läuft.

5.2.8 System Assist Processor

Ein z196 Rechner hat bis zu 4 x 24 = 96 Prozessoren (Cores). Von diesen wird nur der größere Teil als CPUs eingesetzt. Mehrere Prozessoren, als „System Assist Processoren“ (SAP) bezeichnet und vorkonfiguriert, und führen ausschließlich Firmware Code aus. Bei einem maximal hochgerüsteten z196 Rechner sind dies mindestens 14 SAPs. Die SAPs weisen die gleiche Hardware Architektur auf wie die CPUs; auf ihnen läuft aber Firmware und kein z/OS. Bei der Installation eines neuen System z Rechners wird über eine Konfigurationsdatei eingestellt, wie viele der vorhandenen Prozessoren als CPUs bzw. SAPs eingesetzt werden.

SAPs und ihr Firmware Code wird vor allem für drei Funktionen benötigt:

1. **Fehlerbehandlungs- und Recovery Funktionen**
2. **Channel Subsystem**
3. **PR/SM Hypervisor Software für LPAR virtuelle Maschinen (siehe Abschnitt 12.3).**

Für Firmware (und dazugehörige Daten) sind in einem z196 Rechner 16 oder 32 GByte Speicherplatz vorgesehen. Von dem installierten physischen Speicher werden 16 oder 32 GByte abgeteilt und steht als „Hardware System Area“ (HSA) für Firmware Zwecke zur Verfügung. Der Rest kann als Hauptspeicher genutzt werden.

Bitte beachten: Den Begriff SAP (System Assist Processor) nicht verwechseln mit dem Namen der Firma SAP AG in Walldorf (Baden).

5.2.9 System z Festplattenspeicher Anschluss

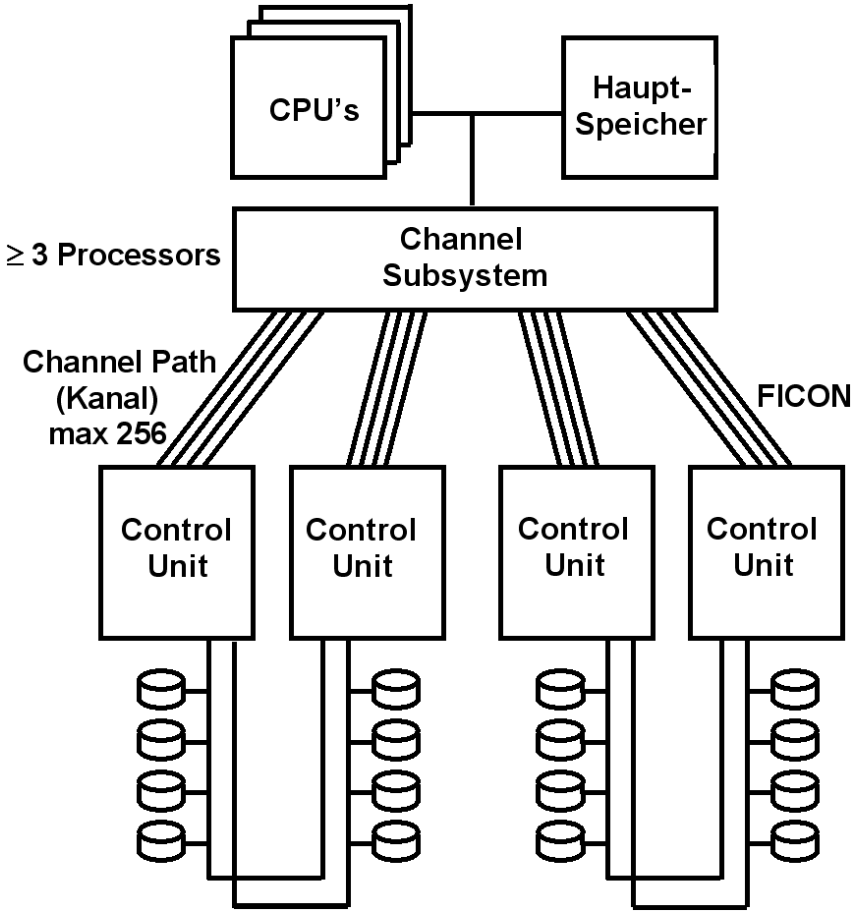


Abb. 5.2.8
Rolle des Channel Subsystems

Ein Mainframe kann über mehrere (bis zu 8) Kanäle mit einer bestimmten Control Unit verbunden werden, und ein I/O Gerät kann an mehr als eine Control Unit angeschlossen werden.

Das Channel Subsystem bildet die logische I/O Konfiguration, wie sie das Betriebssystem sieht, auf die physische Konfiguration ab.

Heute werden fast immer mehrere Control Units und angeschlossene Plattenspeicher zu einem physischen „Enterprise Storage Server" zusammengefasst. Control Units (und das Channel Subsystem) sind jedoch Teil der System z Architektur-Spezifikation; das Enterprise Storage System ist lediglich eine Implementierung, und ist nicht Teil der Architektur.

Ein Plattenspeicher ist typischerweise an zwei Control Units angeschlossen. Die Kommunikation mit der CPU erfolgt wahlweise über eine der beiden Control Units.

z/Architecture Principles of Operation, SA22-7201, Seite 2-6
www.cedix.de/Literature/Textbooks/PrincOps2010.pdf

5.2.10 System z Disk Storage Attachment

System z I/O Geräte werden über Steuereinheiten (Control Units) an das Kanal Subsystem angeschlossen. Viele Funktionen, die auf Ihrem PC von der CPU als I/O Driver Code ausgeführt werden, sind bei einem Mainframe in den Control Units implementiert. Sie belasten die CPUs nur wenig, und ermöglichen die Ansteuerung einer sehr großen Anzahl von Festplatten.

Steuereinheiten können über mehr als einen Kanalpfad an das Kanalsubsystem angeschlossen werden und I/O Geräte (z.B. Plattenspeicher) können an mehr als eine Steuereinheit angeschlossen werden.

z/OS kann auf ein beliebiges I/O Gerät über bis zu 8 unterschiedliche Kanalpfade zugreifen (und umgekehrt).

Der Zugriffsweg kann dynamisch geändert werden (DPS - Dynamic Path Selection). Eine I/O Operation muss nicht auf dem gleichen Weg abgeschlossen werden, auf dem sie gestartet wurde. Hiermit kann erreicht werden, dass sequentielle und zufallsbedingte Zugriffe sich nicht gegensätzlich beeinträchtigen.

Z.B. angenommen zwei Festplattenspeicher, die an die gleiche Steuereinheit angeschlossen sind. Ein Plattenspeicher überträgt einen großen Block sequentieller Daten, während der zweite Plattenspeicher gleichzeitig viele kurze Datenpakete mit einem zufallsbedingten Zugriffsmuster überträgt. Kein Plattenspeicher soll die Nutzung einer Verbindung für einen längeren Zeitraum usurpieren.

In anderen Worten, es gibt mehrere Wege, auf denen Daten (und Steuerinformation) zwischen Platte und CPU übertragen werden können.

Diese dynamische Weg-Steuerung ist rechenaufwendig. Deswegen belastet man mit dieser Aufgabe nicht die CPUs und das Betriebssystem, sondern überträgt sie einer getrennten Verarbeitungseinheit, dem Channel Subsystem. Eine Beispiel-Konfiguration ist in Abb. 5.2.9 wiedergegeben.

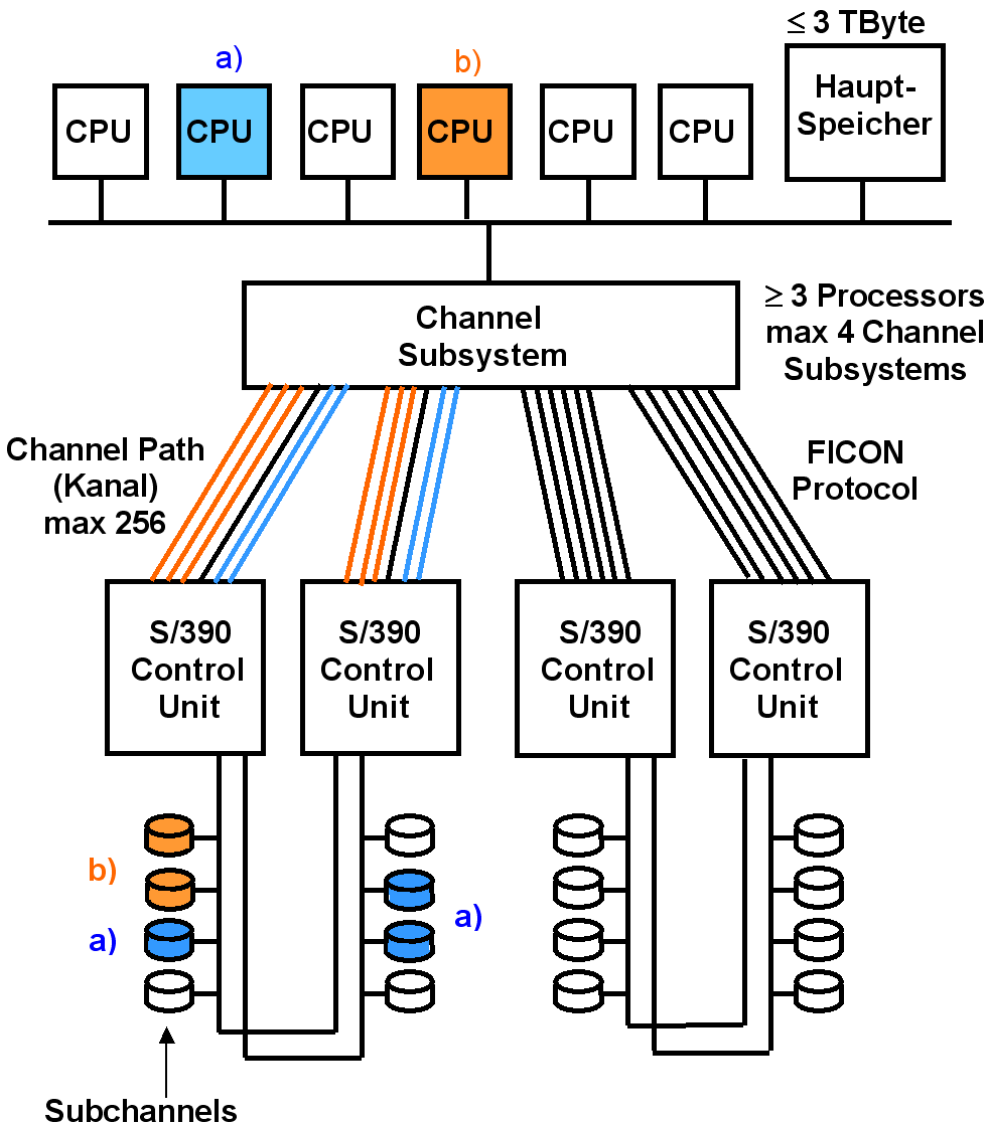


Abb. 5.2.9
Mehrfache Pfade zu einem I/O Gerät

Der auf der blauen CPU a) laufende Prozess greift auf die drei blauen Plattenspeicher a) zu. Diese können über die ersten beiden Control Units erreicht werden. Der blaue Prozess verwendet hierzu vier blau gezeichnete Kanäle, von denen jeweils zwei mit den beiden Control Units verbunden sind.

Ähnliches gilt für den Prozess auf der orange markierten CPU b).

An eine Control Unit können bis zu 8 Kanäle angeschlossen sein.

Das Channel Subsystem kann die Zuordnung von Kanälen zu Prozessen dynamisch ändern.

5.2.11 FICON Director

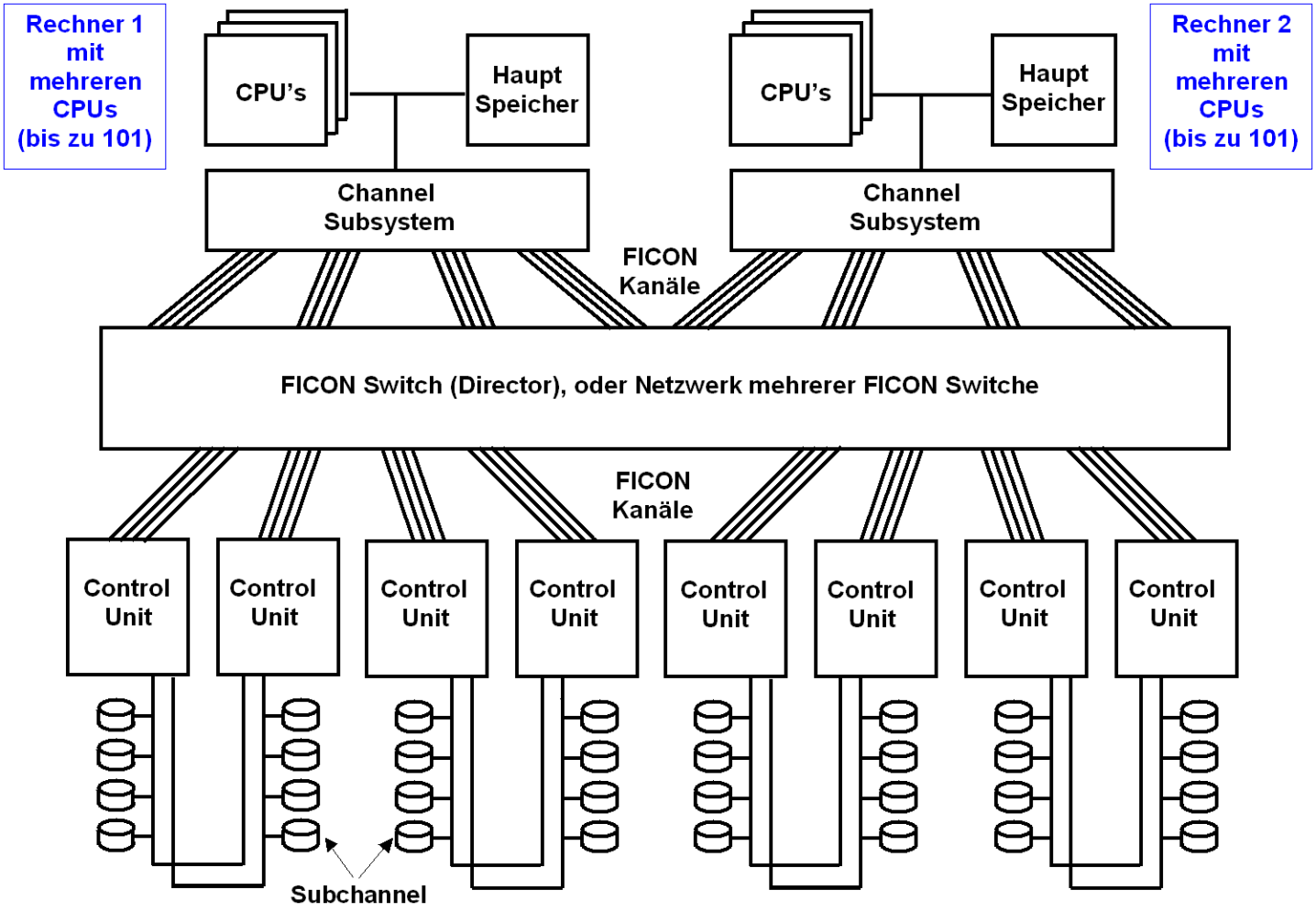


Abb. 5.2.10
FICON Netzwerk

Die obige Abbildung zeigt zwei Mainframe Rechner und zahlreiche Control Units mit ihren Plattenspeichern. Jeder zEC12 Mainframe Rechner kann bis zu 101 CPUs enthalten und verfügt über ein eigenes Channel Subsystem.

Es ist in der Regel notwendig, beide (bis zu 32) Mainframe Rechner mit allen Control Units und allen Plattenspeichern zu verbinden.

Da die Verkabelung sehr unübersichtlich ist, schaltet man einen FICON Switch (offizielle Bezeichnung „FICON Director“) zwischen die Channel Subsysteme und die Control Units. Vielfach wird an Stelle eines einzelnen FICON Switches ein ganzes Netzwerk von FICON Switches eingesetzt (nicht gezeigt).

Große Mainframe Installationen können über 10 000 Glasfaserkabel aufweisen.

Die I/O Anforderungen der Prozesse auf den einzelnen CPUs müssen ihren Weg durch das Netzwerk zu dem gewünschten Plattenspeicher finden. Die Channel Subsysteme der einzelnen Rechner haben die Aufgabe, das Routing der I/O Anforderungen über das FICON Netzwerk zu übernehmen. Ähnlich wie im Internet kann sich der Weg zwischen CPU und Plattenspeicher während der Ausführung einer I/O Operation dynamisch ändern.

Ein derartiges Netzwerk wird als „Storage Area Netzwerk“ (SAN) bezeichnet. Ein SAN benutzt das Fibre Channel Protokoll an Stelle der in Kommunikationsnetzen üblichen TCP/IP oder SNA Protokolle.

Die in Abb. 5.2.10 gezeigten beiden Channel Subsysteme haben eine weitere Aufgabe: Sie verbergen die komplexe I/O Konfiguration vor den z/OS Betriebssystemen, die auf den beiden Rechnern laufen. Ein z/OS Betriebssystem arbeitet mit der Illusion, dass alle Plattenspeicher über individuelle Punkt-zu-Punkt Verbindungen direkt an seine CPUs angeschlossen sind. Diese Verbindungen werden als „Subchannels“ bezeichnet; je ein Subchannel pro Plattenspeicher. Es ist die Aufgabe des Channel Subsystems, logische Subchannels auf physische Kanalpfade und die Topologie des FICON Netzwerkes dynamisch abzubilden.

5.2.12 Enterprise Storage Server

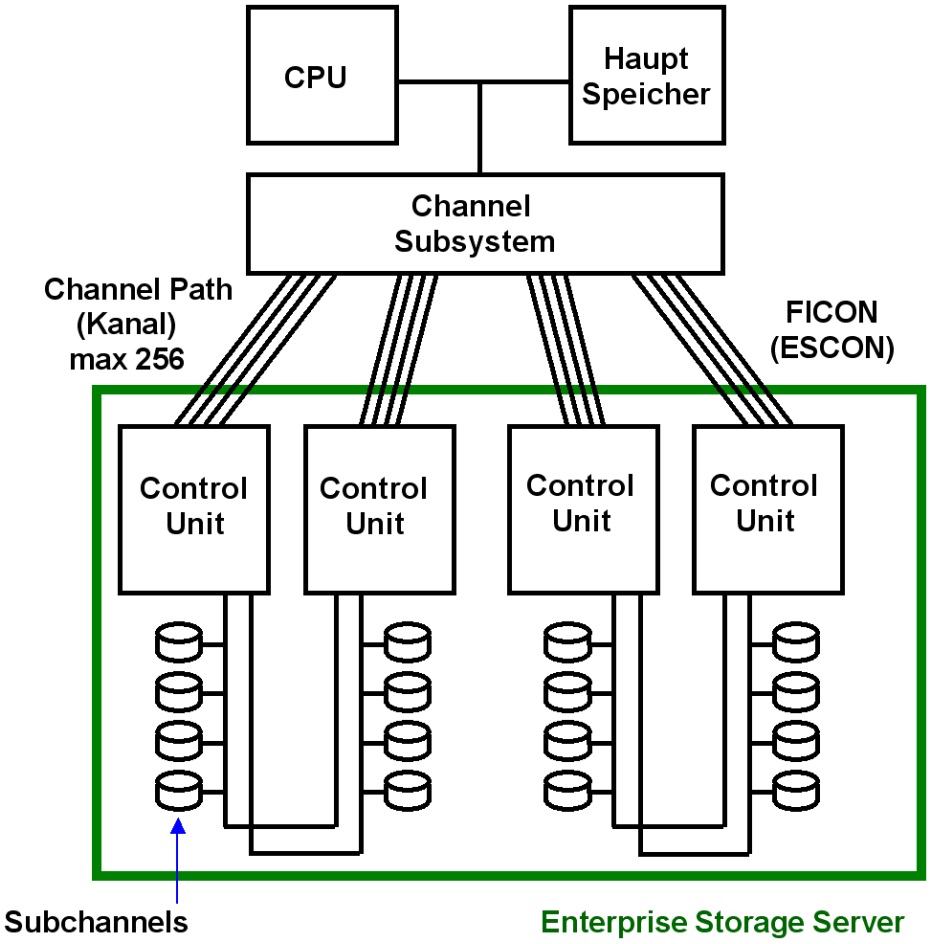


Abb. 5.2.11
Ein Enterprise Storage Server emuliert mehrere Control Units

Früher waren Control Units getrennte physische Einheiten in ihren eigenen Gehäusen.

Heute werden mehrere Control Units und angeschlossene Plattenspeicher zu einem physischen „Enterprise Storage Server“ (ESS) zusammengefasst, der auch die angeschlossenen Plattenspeicher enthält.

Der ESS emuliert mehrere logische Control Units, hat Anschlüsse für zahlreiche FICON Kanäle und bringt zahlreiche Plattenspeicher im gleichen Gehäuse unter.

Es können beliebig viele ESS angeschlossen werden.

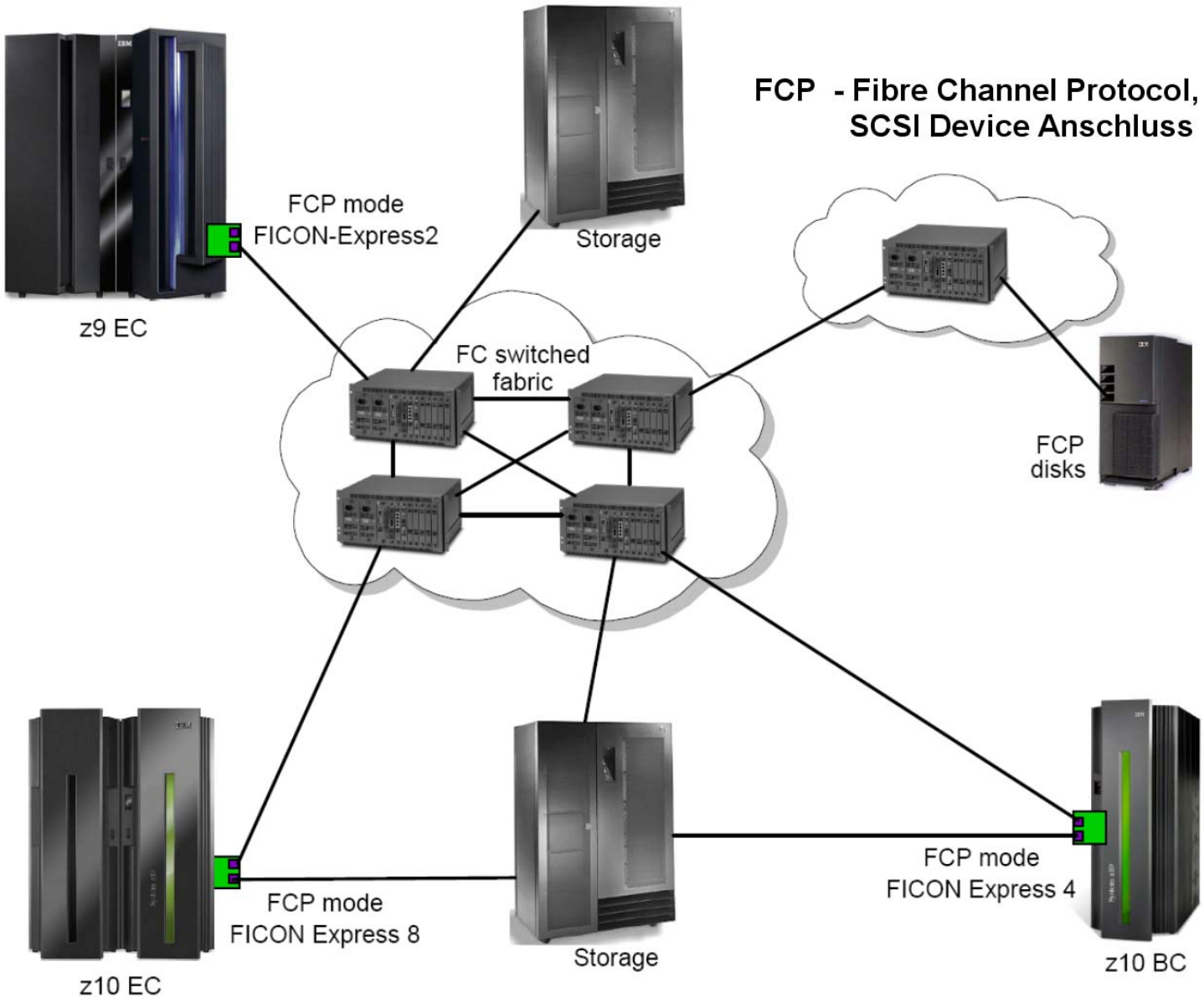


Abb. 5.2.12
Storage Area Netzwerk

Heutige Mainframes können alternativ neben FICON Plattenspeichern auch Fibre Channel SCSI Plattenspeicher anschließen. Dies ist z.B. beim Einsatz von zLinux üblich. Der Großteil der Daten einer Mainframe Installation wird aber auf FICON Platten gespeichert.

Über das Fibre Channel Storage Area Netzwerk können parallel FICON und FC SCSI Verbindungen geschaltet werden.

5.3 Mainframe I/O

5.3.1 Volume

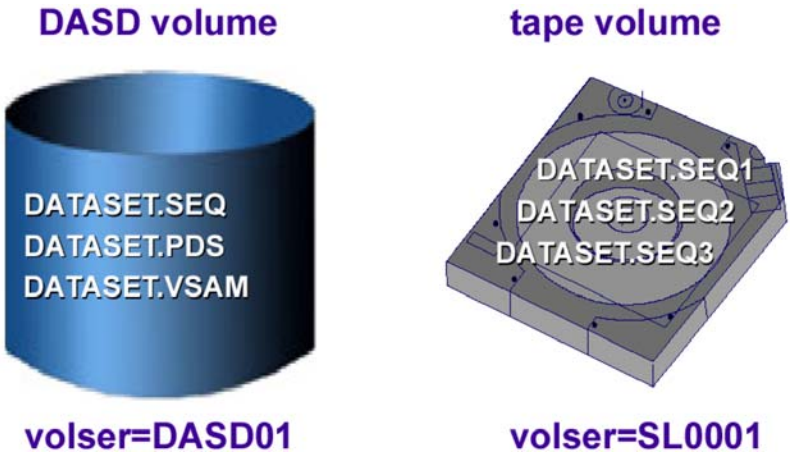


Abb. 5.3.1
Festplatten- und Magnetband - Volumen

Ein “Volume” (Datenträger) ist eine logische externe Speicher-Einheit, beispielsweise ein Festplattenspeicher oder eine Magnetbandkassette. Ein Volume speichert zahlreiche Dateien.

In einer Mainframe Installation ist ein jedes Volume durch eine eindeutige „Volume Serial Number“ (volser) gekennzeichnet, die auf dem Datenträger an einer bestimmten Stelle aufgezeichnet ist. Nicht alle vom Betriebssystem erfassten Volumen müssen in jedem Augenblick für das Betriebssystem zugreifbar sein. Beispielsweise kann eine Nachricht auf der Konsole den System-Administrator auffordern, eine Magnetbandkassette mit einer bestimmten Volume Serial Number manuell aus einem Regal zu entnehmen, und in eine angeschlossene Magnetbandeinheit einzulegen.

5.3.2 Channel und Control Unit

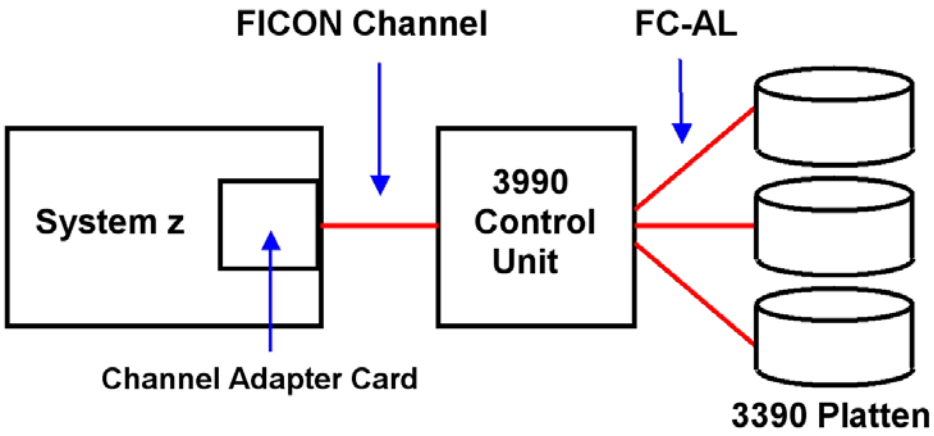


Abb. 5.3.2
Anschluss der Festplatten über eine Control Unit

Ein Kanal (Channel) ist ein Verbindungskabel zwischen einem Rechner und einer Plattenspeicher-Steuereinheit. Ein Channel Adapter ist eine Steckkarte im I/O Cage (I/O Drawer) eines Rechners, die Steckkontakte für das Kanalkabel enthält. Kanäle werden heute als FICON Fibre Channel Verbindungen implementiert.

Channel Adapter und Control Unit sind zwei physische Einheiten, die über ein Kanal-Kabel miteinander verbunden sind. Die Kombination stellt jedoch eine einzige logische Einheit dar. Die Aufteilung ist erforderlich, weil aus Platzgründen die Control Units in einer gewissen Entfernung voneinander und vom Rechner aufgestellt werden müssen.

In manchen Fällen ist es möglich, die Control Unit im gleichen Gehäuse wie die CPUs unterzubringen. In diesem Fall werden Control Unit und Channel Adapter als eine einzige Baugruppe implementiert. Ein Beispiel ist der OSA Adapter. Dies ist eine Steckkarte (I/O Card) im I/O Drawer eines Rechners, die zum Anschluss von Ethernet Verbindungen dient.

Note: Innerhalb IBM wird eine Festplatte regelmäßig als „DASD“ (Direct Access Storage Device, ausgesprochen „dasdi“ oder „däsdi“) bezeichnet.

5.3.3 Ausführen einer I/O Operation

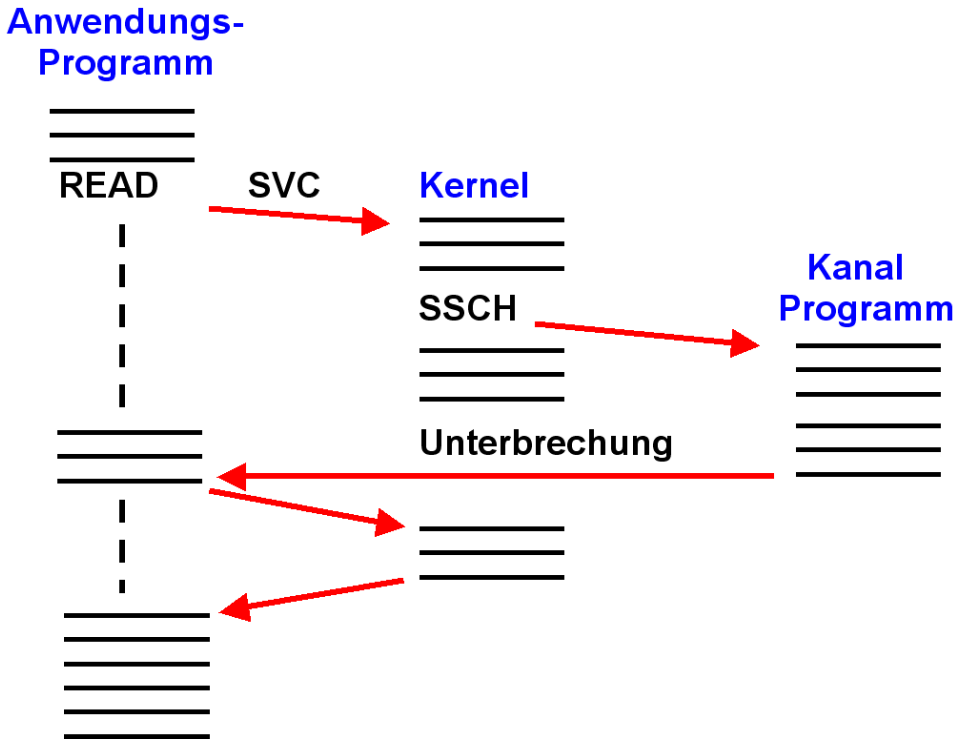


Abb. 5.3.3
d) Ausführen eines I/O Befehls

Unix Systeme steuern ihre Plattenspeicher über ein als I/O Driver bezeichnetes Programm, welches von der CPU ausgeführt wird. Die Funktion eines Unix I/O Drivers wird bei einem Mainframe zum allergrößten Teil in die Control Unit ausgelagert. Das dort ablaufende Programm wird als Channel Program bezeichnet. Ein Channel Program besteht aus einzelnen Befehlen, die als Channel Command Words (CCW) bezeichnet werden. Beim Starten einer I/O Operation werden die Befehle des Channel Programms aus dem Hauptspeicher ausgelesen, an die Control Unit übergeben und dort ausgeführt.

Abb. 5.3.3 zeigt das Zusammenspiel des Anwendungsprogramms im User State, des Kernels im Kernel State (Supervisor State) und des Kanalprogramms.

Das Anwendungsprogramm führt einen READ Befehl aus. Dies führt zu einem Aufruf des Kernels (Supervisor) über einen SVC Maschinenbefehl. Der Kernel ruft seine I/O Routinen auf und veranlasst zunächst die Erstellung, und danach die Ausführung des Kanalprogramms durch Kanal und Control Unit mittels eines Start Subchannel (SSCH) Maschinenbefehls. Die Control Unit führt die I/O Befehle (CCWs) der Reihe nach aus.

Der Abschluss der Kanalprogrammverarbeitung wird der CPU von der Control Unit mittels einer I/O Unterbrechung (Channel End Device End, CEDE) mitgeteilt.

Abb. 5.3.4
Einfaches Plattenspeicher - Steuerprogramm

Dargestellt ist ein einfaches Kanalprogramm und seine CCWs für die Datenübertragung vom/zum Plattenspeicher, wie es in den S/370 Rechnern gebräuchlich war. Heutige Kanalprogramme sind deutlich komplexer.

5.3.4 3390 Plattenspeicher

Control Units entlasten die CPUs, indem der I/O Driver Code außerhalb der CPUs in den Control Units ausgeführt wird. Zum Betriebssystem gehören nur Skelette für einige wenige I/O Driver Routinen, z.B. je eine generische Routine für Plattenspeicher, Magnetbänder, Drucker und Netzanschlüsse. Die I/O Driver Code Routinen werden als „Channel Programs“ bezeichnet, die jeweils aus einer Gruppe von Anweisungen, den „Channel Command Words“ (CCW) bestehen.

Das Anwendungsprogramm ergänzt das Channel Programm um Daten wie Adresse und Länge des I/O Buffers im Hauptspeicher oder die logische Adresse des I/O Gerätes (das Channel Subsystem bildet die logische auf die physische I/O Adresse ab). Der Betriebssystem Kernel überträgt daraufhin den Channel Program Code an die Control Unit, wo er ausgeführt wird.

Über viele Jahre war IBM – als Erfinder der Festplattenspeicher Technologie - der führende Hersteller von magnetischen Festplattenspeichern. Während dieser Zeit hatten IBM Festplattenspeicher einen Durchmesser von 14 Zoll.

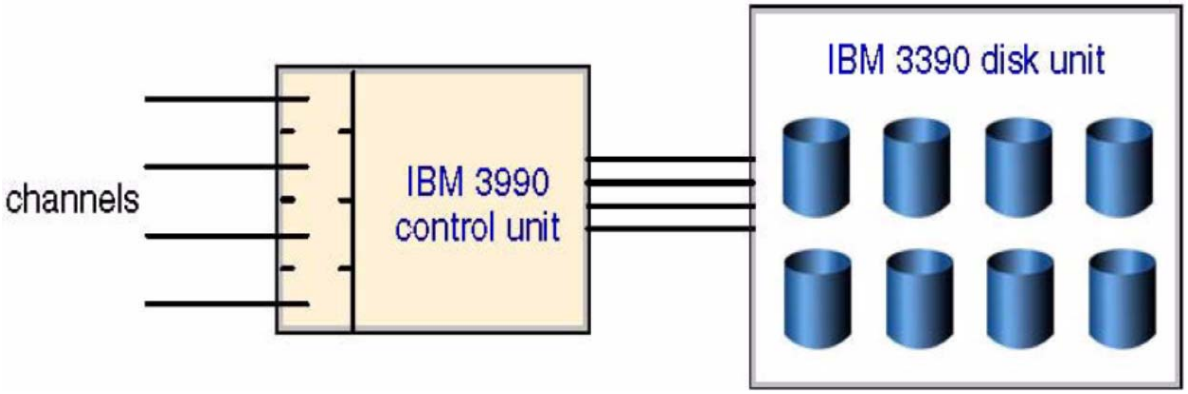


Abb. 5.3.5
Klassischer Anschluss eines 3390 Speichers an eine 3990 Control Unit

1993 wurde das letzte Modell dieser Art als IBM 3390 Plattenspeicher und der IBM 3990 Control Unit herausgebracht. Es hatte eine Speicherkapazität von 8,5 GByte. In der Folge wurde 5 ½ und 3 ¼ Zoll Festplatten eingesetzt.

Bis 1993 wurden für alle neuen Festplattenmodelle eigene Control Units und eigene Channel Programme entwickelt. Davon nahm man ab 1993 Abstand und standardisierte die Software Unterstützung auf das 3390 Modell.

Um dies zu erreichen, arbeitet der OS Kernel und die Control Unit mit der Illusion eines generischen 3390 Festplattenspeichers. Die Eigenschaften dieses physischen Plattenspeichers wurden für die Definition der logischen 3390 Plattenspeicher übernommen. Parameter wie Spuren pro Zylinder und Bytes pro Spur sind festgelegt, und sind wie bei allen 3390 Plattenspeichern identisch. Die Anzahl der Zylinder pro Plattenspeicher kann als Parameter definiert werden, woraus sich unterschiedliche Speicherkapazitäten ergeben.

Das Betriebssystem kennt nur 3390 Festplatten mit deren Struktur bezüglich Anzahl Zylinder, Anzahl Spuren, Bytes pro Spur usw. Das Nachfolgemodell des 3390 Plattenspeichers war der IBM „Enterprise Storage Server“ (ESS) . Das ursprüngliche „Shark“ Modell wurde später durch die „DS6000“ und „DS8000“ Enterprise Storage Server abgelöst.

Diese ESS benutzen bis zu 1024 Standard 3 ½ Zoll oder 2 ½ Zoll SCSI Festplatten mit bis zu je 600 GByte Speicherkapazität sowie zwei 4-way PowerPC Multiprozessoren für die Emulation mehrerer 3990 Control Units, für die Emulation der angeschlossenen 3390 Plattenspeicher und für weitere fortschrittliche Funktionen, u.A. die Verwaltung sehr großer Plattenspeicher-Caches. Ein Enterprise Storage Server kann unterschiedliche Arten von Festplatten enthalten, mit unterschiedlicher Speicherkapazität und Struktur. Es ist die Aufgabe des Enterprise Storage Servers, die virtuellen 3390 Plattenspeicher auf die tatsächlich eingesetzten physischen Festplatten und deren Struktur abzubilden. Spezifisch haben heutige Festplatten eine größere Speicherkapazität als die virtuellen 3390 Plattenspeicher. Es werden deshalb immer mehrere virtuelle 3390 Platten auf einer physischen Festplatte emuliert.

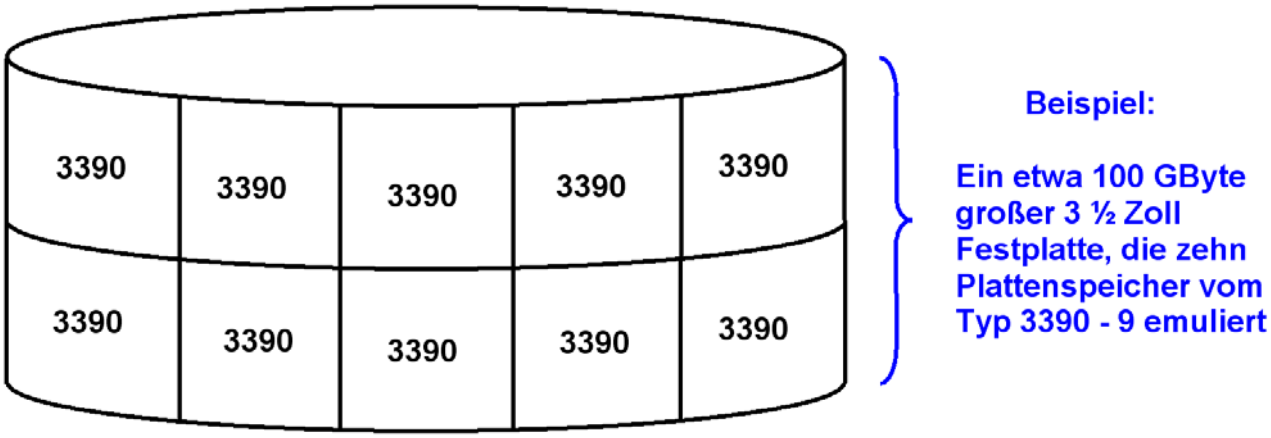


Abb. 5.3.6
Physisches und Logisches Volume

Emulierte 3390 Plattenspeicher werden als „Logical Volumes“ (LV) bezeichnet. Sie existieren als zwei unterschiedliche standardisierte Größen, die als „Modell 3“ und „Modell 9“ bezeichnet werden. Die Spur- (track) Geometrie innerhalb einer Modellserie ist immer identisch.

Eine 3390-Modell 3 Plattenspeicher ist in 3339 Cylinder aufgeteilt, mit 15 Tracks pro Cylinder. Ein Track hat hierbei eine Kapazität von 56,664 Bytes, woraus sich eine Gesamtkapazität von 2.84 GByte für die Festplatte ergibt. Das Modell 9 hat die 3-fache Anzahl von Spuren (10 017) und damit die 3-fache Kapazität des Modell 3 (8,51 GByte). Auch hier sind 56,664 Bytes pro Spur vorhanden. Das z/OS Betriebssystem ist häufig auf Model 3 Platten installiert.

Tatsächlich hängt die effektive Kapazität von der Größe der Blocksize ab, mit der Data Sets angelegt (allocated) werden, und der Struktur der Data Sets. So können z.B. VSAM Data Sets maximal 2,3 GByte an Daten auf einer Modell 3 Festplatte speichern; der Rest wird für Verwaltungsinformation benötigt.

	3390-3	3390-9
Zylinder pro Plattenstapel	3 339	10 017
Spuren pro Zylinder	15	15
Bytes pro Spur	56 664	56 664
Bytes pro Zylinder	849 960	849 960
MByte pro Plattenstapel	2 838	8 514
4096 Byte Blocks pro Spur	12	12

Abb. 5.3.7
DASD Speicherkapazität der Modelle 3390

Die in Abb. 5.3.6 wiedergegebenen Daten sind für das Allocate von Data Sets interessant. Beispielsweise sollte der Parameter BLKSIZE so gewählt werden, dass es bei 56 664 Bytes/Spur möglichst wenig Verschnitt gibt. Bei der Optimierung hilft ein BLKSIZE Calculator, siehe http://webspace.webring.com/people/lp/programmingstuff/blksize.htm .

Es bietet sich das "half-track blocking“ an: 2 Blöcke pro Spur. Eine Blockgröße von 27 998 Bytes (55 996 / 2 = 27 998) ermöglichst 2 Blöcke/Spur, oder eine Nutzung von 99 %. Eine Blockgröße von 28 000 Bytes gestattet nur einen Block pro Spur. Dies bedeutet einen Verschnitt von etwa 50 %.

Eine populäre Wahl ist, 349 logische Record zu je 80 Bytes in einen Block von 27920 Bytes zu packen.

5.3.5 System z I/O-Konfiguration

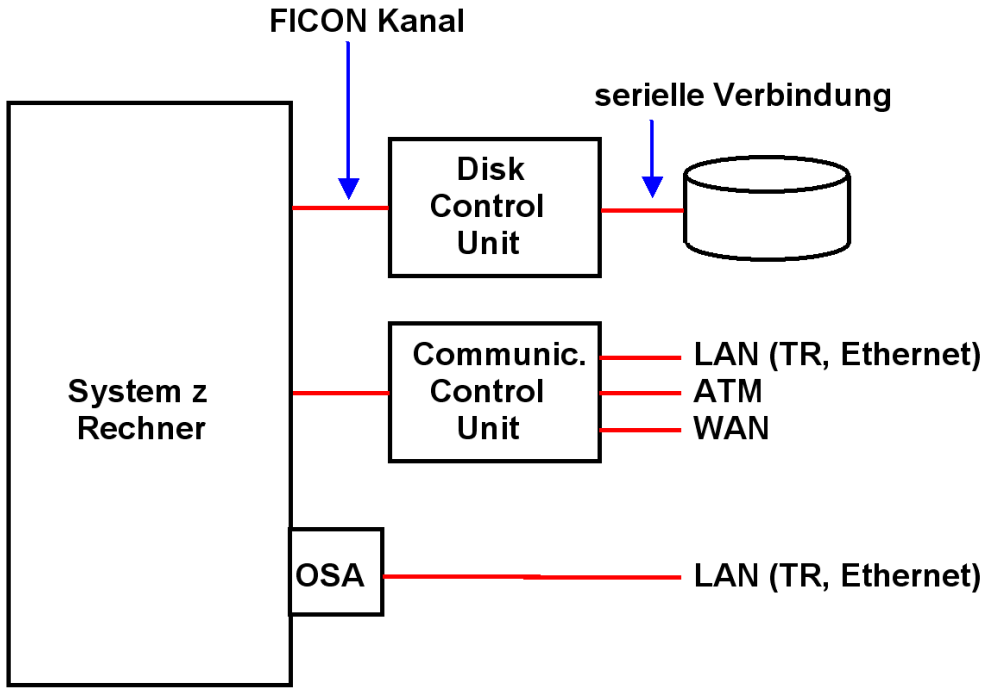


Abb. 5.3.8
Der OSA Adapter integriert die Control Unit Funktion

I/O Geräte werden grundsätzlich über Steuereinheiten (Control Units) angeschlossen. Steuereinheiten sind meistens in getrennten Boxen untergebracht, und über Glasfaser (z.B. FICON) an den System z Rechner angeschlossen.

Es existieren viele unterschiedliche Typen von Steuereinheiten. Die wichtigsten schließen externe Speicher (Platten, Magnetband-Archivspeicher) und Kommunikationsleitungen an.

Es existieren Steuereinheiten für viele weiteren Gerätetypen. Beispiele sind Belegleser für Schecks oder Drucker für die Erstellung von Rentenbescheiden.

5.3.6 Channel- to Channel Verbindung

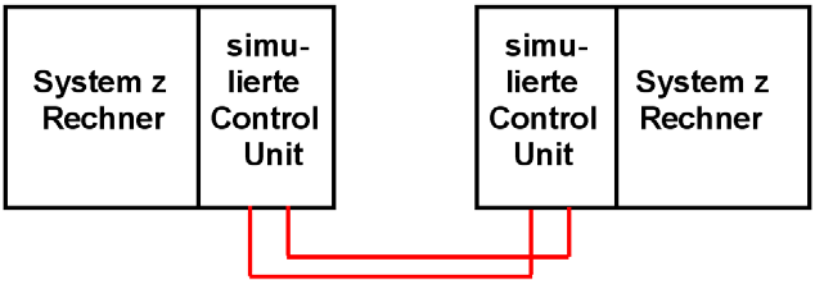


Abb. 5.3.9
Full Duplex Channel to Channel Verbindung

Die Channel- to Channel (CTC) Verbindung wird durch eine Hardware Einrichtung eines zSeries Systems verwirklicht, die dieses System gegenüber einem anderen zSeries System wie eine I/O Einheit erscheinen lässt.

Für eine Full Duplex Verbindung werden normalerweise zwei CTC Verbindungen eingesetzt.

Die „Cross-System Coupling Facility“ (XCF) ist eine Komponente des z/OS Betriebssystems. Sie verwendet das CTC Protokoll und die CTC Hardware. Sie stellt die Coupling Services bereit, mit denen z/OS Systeme innerhalb eines Clusters (Sysplex) miteinander kommunizieren.

5.3.7 RAID-System

Ein RAID-System (ursprünglich redundant array of inexpensive disks, heute redundant array of independent disks) dient zur Organisation mehrerer physischer Festplatten eines Computers zu einem logischen Laufwerk, das eine höhere Datensicherheit bei Ausfall einzelner Festplatten und/oder einen größeren Datendurchsatz erlaubt als eine physische Festplatte. Während die meisten in Computern verwendeten Techniken und Anwendungen darauf abzielen, Redundanzen (das Vorkommen doppelter Daten) zu vermeiden, werden bei RAID-Systemen redundante Informationen gezielt erzeugt, damit beim Ausfall einzelner Komponenten das RAID System als Ganzes seine Funktionalität behält.

Der Begriff wurde von Patterson, Gibson und Katz 1987 an der University of California, Berkeley in ihrer Arbeit „A Case for Redundant Array of Inexpensive Disks (RAID)“ zum ersten Mal verwendet. Darin wurde die Möglichkeit untersucht, kostengünstige Seagate Festplatten im Verbund als logisches Laufwerk zu betreiben, um die Kosten für einen großen (zum damaligen Zeitpunkt sehr teuren) 14 Zoll IBM Festplattenspeicher einzusparen. Dem gestiegenen Ausfallrisiko im Verbund sollte durch die Speicherung redundanter Daten begegnet werden, die einzelnen Anordnungen wurden als RAID-Level diskutiert.

Die weitere Entwicklung des RAID-Konzepts führte zunehmend zum Einsatz in Serveranwendungen, die den erhöhten Datendurchsatz und die Ausfallsicherheit nutzen. Der Aspekt der Kostenersparnis wurde dabei aufgegeben. Heute existiert meistens die Möglichkeit, in einem solchen System einzelne Festplatten im laufenden Betrieb zu wechseln.

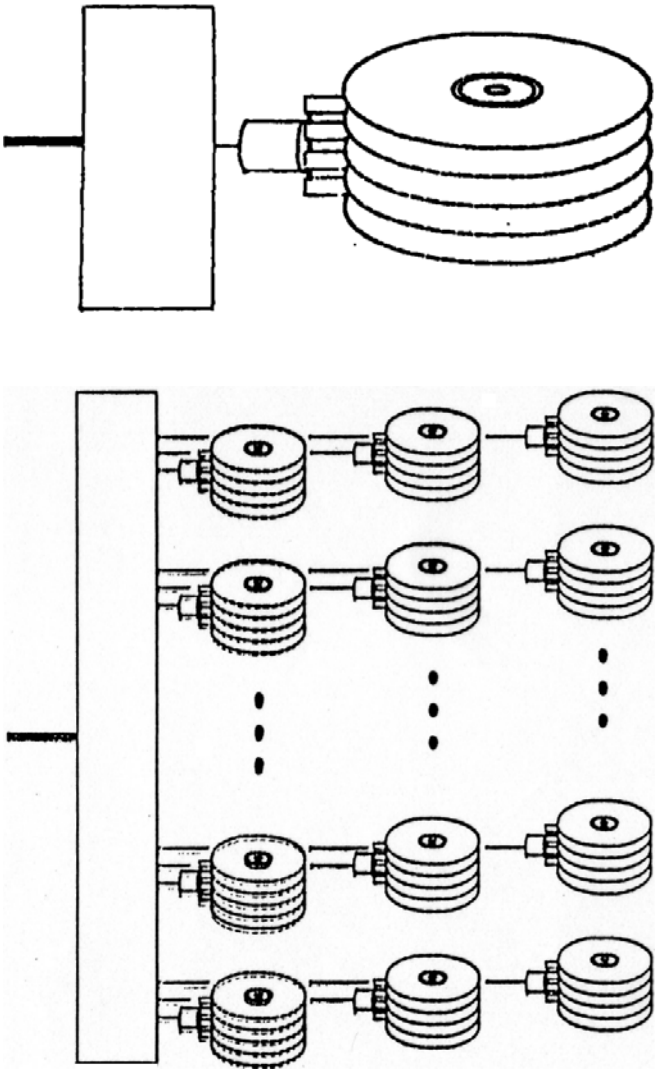


Abb. 5.3.10
Ersatz eines einzigen 14 Zoll Festplattenspeichers

Die Idee ist, dass das Disk Array für das Betriebssystem wie eine einzige Platte aussieht.

5.3.8 RAID-Level

Der Betrieb eines RAID-Systems setzt mindestens zwei Festplatten voraus. Die Festplatten werden gemeinsam betrieben und bilden einen Verbund, der leistungsfähiger ist als die einzelnen Festplatten. Mit RAID-Systemen kann man folgende Vorteile erreichen:

- **Erhöhung der Ausfallsicherheit (Redundanz)**
- **Steigerung der Daten-Transferraten (Leistung)**
- **Aufbau großer logischer Laufwerke**
- **Austausch von Festplatten und Erhöhung der Speicherkapazität während des Systembetriebes**
- **Kostenreduktion durch Einsatz mehrerer preiswerter Festplatten**

Die genaue Art des Zusammenwirkens der Festplatten wird durch den RAID-Level spezifiziert. Die gebräuchlichsten RAID-Levels sind RAID 0, RAID 1, RAID 5, RAID 6 und RAID 10.

Aus Sicht des Benutzers, eines Anwendungsprogramms oder des Betriebssystems unterscheidet sich ein logisches RAID-Laufwerk nicht von einer einzelnen physischen Festplatte.

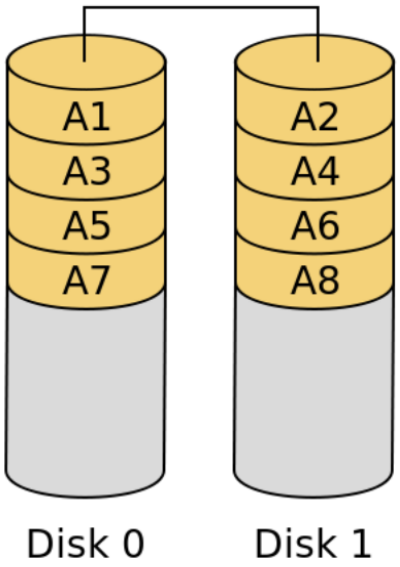


Abb. 5.3.11
RAID 0

RAID 0 bietet gesteigerte Transferraten, indem die beteiligten Festplatten in zusammenhängende Blöcke gleicher Größe (z.B. 16KB) in Streifen (eng. stripes) aufgeteilt werden, wobei jeder Streifen eines Datenblocks auf einer separaten Festplatte gespeichert wird. Diese Blöcke werden quasi im Reißverschlussverfahren zu einer großen logischen Festplatte angeordnet . Somit können Zugriffe auf allen Platten parallel durchgeführt werden (engl. striping, was „in Streifen zerlegen“ bedeutet).

Die Datendurchsatz-Steigerung (bei sequentiellen Zugriffen, aber besonders auch bei hinreichend hoher Nebenläufigkeit) beruht darauf, dass die notwendigen Festplatten-Zugriffe in höherem Maße parallel abgewickelt werden können.

Streng genommen handelt es sich bei RAID 0 nicht um ein wirkliches RAID, da es keine Redundanz gibt. Beim Ausfall einer Festplatte sind die Daten des gesamten RAID 0 Verbandes verloren.

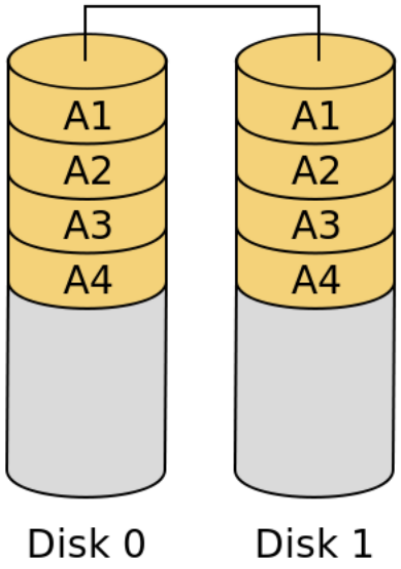


Abb. 5.3.12
RAID 1

RAID 1 ist der Verbund von mindestens 2 Festplatten. Ein RAID 1 speichert auf beiden Festplatten die gleichen Daten auch als Spiegelung bezeichnet. Beim Ausfall einer Platte sind die Daten identisch auf der zweiten Festplatte vorhanden. Beim Spiegeln von Festplatten an einem Kanal spricht man von Disk Mirroring, beim Spiegeln an unabhängigen Kanälen von Disk Duplexing (zusätzliche Sicherheit).

Fällt eine der gespiegelten Platten aus, kann die andere weiterhin alle Daten liefern. Besonders für sicherheitskritische Echtzeitanwendungen ist das unverzichtbar. RAID 1 ist eine einfache und schnelle Lösung zur Datensicherheit und Datenverfügbarkeit, besonders geeignet für kleinere Nutzkapazitäten. Lediglich die Hälfte der Gesamtkapazität steht als nutzbarer Bereich zur Verfügung.

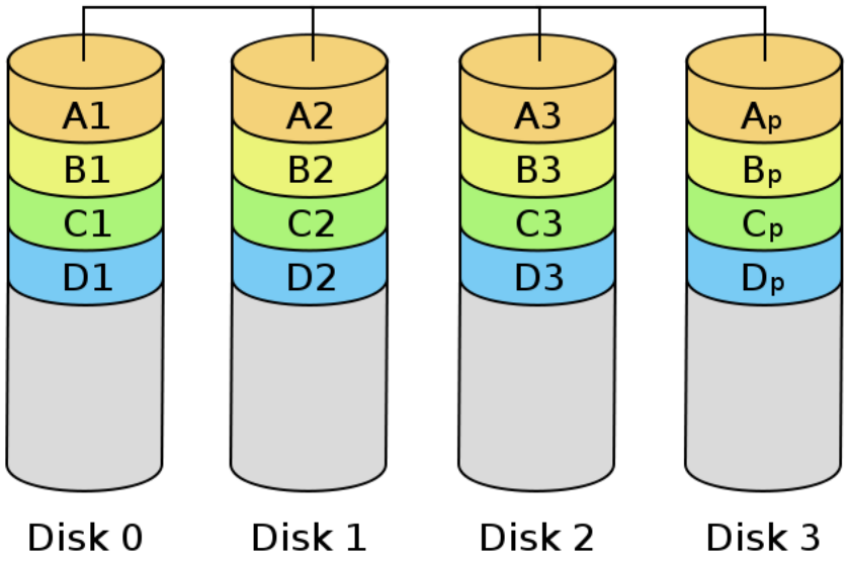


Abb. 5.3.13
RAID 3 und 4

RAID 4 verteilt wie bei RAID 0 die Daten auf mehrere Festplatten. Auf einem Sicherheitslaufwerk werden Paritätsdaten abgelegt. Durch diese Parität stehen selbst bei einem Ausfall einer Festplatte alle Daten weiterhin zur Verfügung. Lediglich die Kapazität einer Festplatte geht für die Redundanz verloren. Bei einem 4 Verbund mit 5 Festplatten stehen 80 Prozent der Gesamtkapazität als Nutzkapazität zur Verfügung.

Beim Schreiben kleiner Datenblöcke wird das Paritätslaufwerk sehr stark belastet was die Performance deutlich negativ beeinflusst. RAID 4 arbeitet im Gegensatz zu RAID 3 mit unabhängigen Datenpfaden für jedes Laufwerk. Dies bringt vor allem beim Schreiben und Lesen großer Dateien eine bessere Performance.

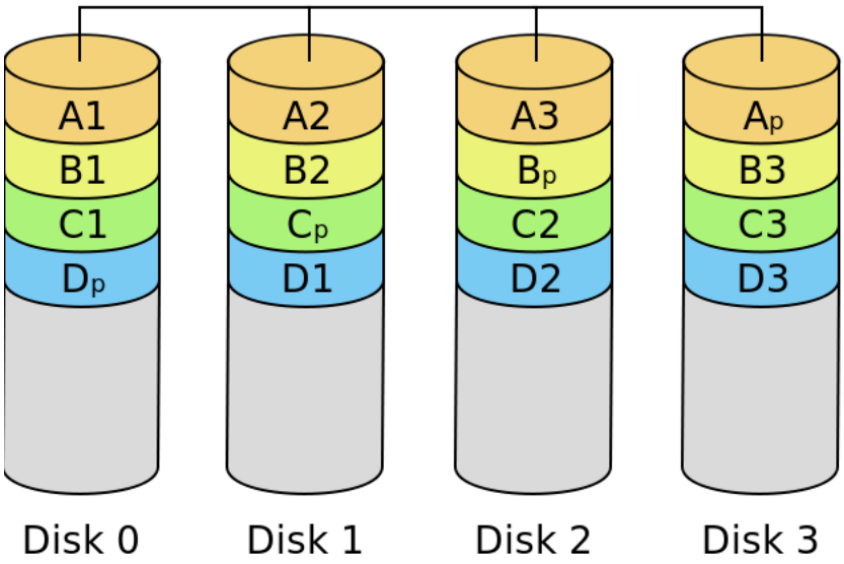


Abb. 5.3.14
RAID 5

Anders als bei RAID 4 verteilt RAID 5 die Paritätsdaten Ap, Bp. Cp und Dp gleichmäßig auf allen Festplatten im Verbund. Dies garantiert bei allen Zugriffen eine optimale Auslastung der Laufwerke. Selbst bei zufallsbedingten (random) Zugriffen, wie sie für ein multitasking multiuser Betriebssystem typisch sind, kann somit eine optimale Performance erreicht werden. RAID 5 bietet beim Ausfall einer Festplatte die gleiche Sicherheit und Datenverfügbarkeit wie RAID 4.

In schreibintensiven Umgebungen mit kleinen, nicht zusammenhängenden Änderungen ist RAID 5 benachteiligt, da bei zufälligen Schreibzugriffen der Durchsatz aufgrund des zweiphasigen Schreibverfahrens deutlich abnimmt (an dieser Stelle wäre eine RAID-0+1-Konfiguration vorzuziehen).

RAID 5 ist eine der kostengünstigsten Möglichkeiten, Daten auf mehreren Festplatten redundant zu speichern und dabei das Speichervolumen effizient zu nutzen.

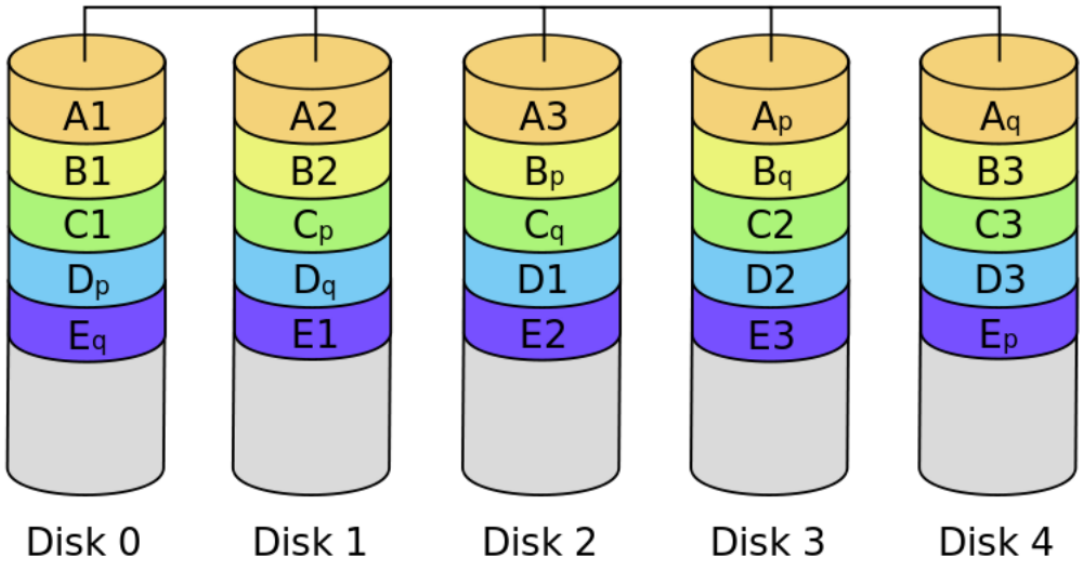


Abb. 5.3.15
RAID 6

Wenn bei RAID 5 eine Festplatte ausfällt, wird der laufende Betrieb dadurch zunächst nicht beeinträchtigt. Die fehlerhafte Festplatte muss baldigst durch eine neue Festplatte ersetzt werden. Nachdem dies geschehen ist, kann ein Unterprogramm des Enterprise Storage Servers den Inhalt der neue Platte aus den Inhalten der restlichen RAID 5 Platten automatisch generieren. Bis dies geschehen ist, ist die volle Ausfallsicherheit nicht mehr gewährleistet.

RAID 6 funktioniert ähnlich wie RAID 5, verkraftet aber den gleichzeitigen Ausfall von bis zu zwei Festplatten. Insbesondere beim intensiven Einsatz hochkapazitiver Festplatten kann die Wiederherstellung der Redundanz nach einem Plattenausfall viele Stunden dauern. Während dieser Zeit besteht kein Schutz gegen einen weiteren Ausfall.

Im Gegensatz zu RAID 5 gibt es bei RAID 6 mehrere mögliche Implementierungsformen, die sich insbesondere in der Schreibleistung und dem Rechenaufwand unterscheiden, und von unterschiedlichen Herstellern unter dem Namen RAID 6 vertrieben werden. Im allgemeinen gilt: Bessere Schreibleistung wird durch erhöhten Rechenaufwand erkauft.

Im einfachsten Fall wird eine zusätzliche XOR-Operation über eine orthogonale Datenzeile berechnet. Auch die zweite Parität wird rotierend auf alle Platten verteilt. Eine andere RAID 6 Implementierung rechnet mit nur einer Datenzeile, produziert allerdings keine Paritätsbits, sondern einen Zusatzcode, der 2 Einzelbit-Fehler beheben kann. Das Verfahren ist rechnerisch aufwändiger.

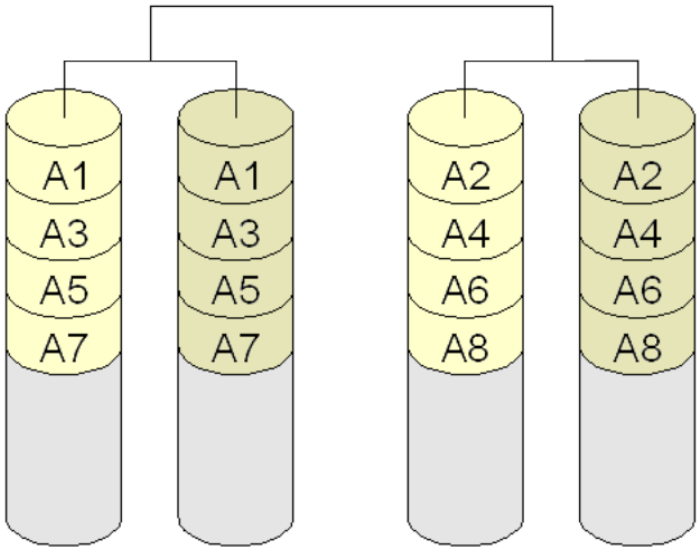


Abb. 5.3.16
RAID 10

Aus einer Kombination von RAID 0 (Performance) und RAID 1 (Datensicherheit) ist der RAID Level 10 entstanden. Rapid 10 ist eine Kombination aus RAID 0 + 1. Dabei werden immer $2 * n$ Platten zu einem RAID 0 zusammen gefasst - und dann per RAID 1 miteinander verbunden. Dabei ist $n \geq 2$.

RAID 10 Verbände bieten optimale Performance bei optimaler Ausfallsicherheit. Wie bei RAID 0 wird die optimale Geschwindigkeit allerdings nur bei sequentiellen Zugriffen erreicht und wie bei RAID 1 gehen 50 Prozent der Gesamtkapazität für die Redundanz verloren.

http://en.wikipedia.org/wiki/Standard_RAID_levels

5.3.9 Persistenz

Auf Festplatten abgelegte Daten haben den Vorteil, dass bei einem Stromausfall (oder beim Abschalten eines Rechners) Daten nicht verloren gehen.

Mit einem ausreichend hohem RAID Aufwand kann erreicht werden, dass Daten beliebige und auch sehr seltene Fehlerfälle intakt überstehen. Dieses Ziel wird in Mainframe Installationen häufig mit RAID 6 Systemen erreicht, die in zwei unterschiedlichen geografischen Standorten gespiegelt werden. Eine moderne Mainframe Installation geht heute davon aus, dass RAID Daten beliebig hohe Sicherheitsanforderungen erfüllen und nie verloren gehen.

Derartig gespeicherte Daten werden als persistent bezeichnet. Daten im Hauptspeicher eines Rechners sind nicht persistent, da sie bei einem Stromausfall verloren gehen, oder bei einem Hardware oder Software Fehler beschädigt werden können.

Die persistente Speicherung von Daten ist besonders bei der Transaktionsverarbeitung ein wichtiges Kriterium.

5.4 Enterprise Storage Server

5.4.1 Festplattenspeicher-Cache

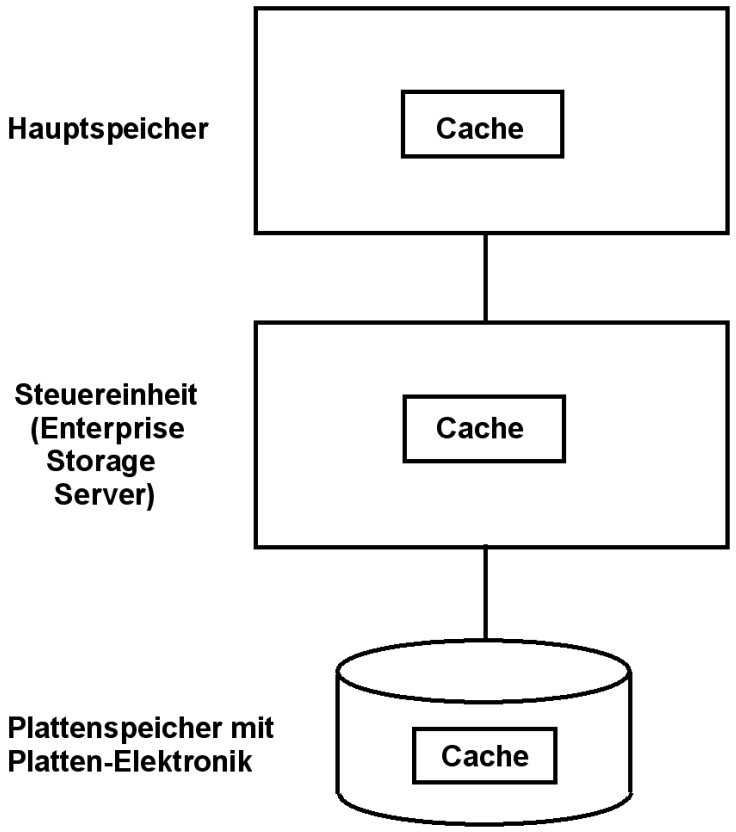


Abb. 5.4.1
Drei unterschiedliche Arten des Festplatten Cache

Zugriffe zu einem Festplattenspeicher benötigen Millisekunden. Um die Zugriffseigenschaften zu verbessern, wird bei jedem Lesezugriff ein größerer Block an Daten (evtl. eine ganze Spur oder ein ganzer Zylinder) in einen Cache Speicher gelesen. Vor allem bei einer Folge von sequentiellen Zugriffen können diese dann aus dem Cache befriedigt werden

Ein Cache für Plattenspeicherdaten kann sich im Hauptspeicher (Buffer Pool bei Datenbanken), und/oder in der Steuereinheit (Enterprise Storage Server) und/oder auf dem Plattenspeicher selbst befinden.

Unter z/OS wird der Plattenspeicher Cache im Hauptspeicher meistens als „Buffer Pool“ realisiert. Unabhängig davon unterhält der Enterprise Storage Server einen umfangreichen Plattenspeicher Cache. Auch die Elektronik eines Plattenspeichers unterhält heute in der Regel einen weiteren Cache.

In der historischen Entwicklung wurden zuerst Control Units um einen Cache-Speicher erweitert. Dieser ermöglichte es ihnen, einen Teil der Lese Zugriffe auf die physischen Platten zu vermeiden. Die ersten Control Unit Caches hatten Größen von nur 16 bis 32 MByte. Mit den Caches wurde es notwendig, Mikrocode zu entwickeln, der in der Control Unit ablief und eine effiziente Datenpufferung ermöglichte.

In den nächsten Schritten wurden die Caches größer, und dann wurden mit dem Aufkommen von preisgünstig zu produzierenden 3 ½ Platten die großen 14 Zoll Platten ersetzt. Jetzt wurden auch die Datenstrukturen und Plattenformate wie Count-Key-Data als real existierende Formate aufgelöst und durch die Control Units emuliert. Diese Control Units hatten damit bereits eine Komplexität erreicht, die sie zu eigenständigen Systemen machte.

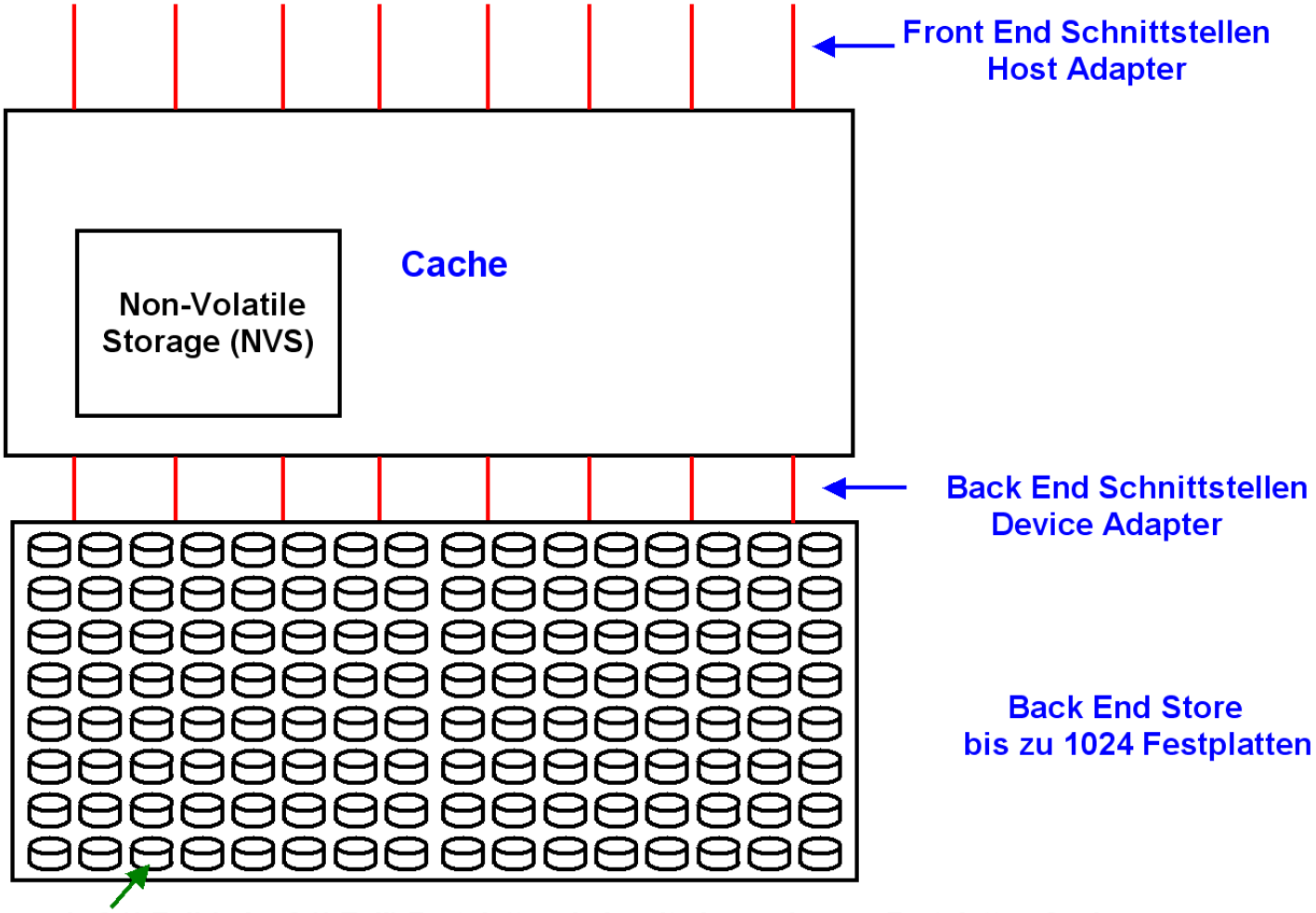
Lese-Zugriffe können mit Hilfe eines Caches deutlich beschleunigt werden. Wünschenswert ist es, auch Schreibvorgänge zu beschleunigen, in dem Daten zunächst in den Cache, und dann asynchron auf die Platte geschrieben werden.

Dies ist grundsätzlich problematisch. Wenn ein Problem auftritt, ehe das Schreiben der Daten aus dem Cache auf den Plattenspeicher abgeschlossen ist, gehen die Daten verloren. Ein typisches Beispiel ist ein Stromausfall. Wenn zum Zeitpunkt eines Stromausfalls der Cache nicht vollständig geleert wurde, sind Daten verloren gegangen.

Als Lösung bildet man einen Teil des Enterprise Storage Server (ESS) Caches als Non-Volatile Storage (NVS) aus. Dies ist ein Halbleiterspeicher mit einer eigenen Batterie zur Stromversorgung. Letztere stellt sicher, dass bei einem Stromausfall (oder in anderen Fehlerfällen) die NVS Daten nicht verloren gehen. Weitere Einrichtungen stellen sicher, dass eine I/O Operation als abgeschlossen gelten kann, wenn die Daten im NVS gelandet sind. Der entgültige Datentransfer zum Plattenspeicher erfolgt dann asynchron und unbemerkt vom Betriebssystem.

5.4.2 Enterprise Storage Server

Front end "Host Adapter" Anschlüsse für den Anschluss an FICON Kanäle



Normale 3 ½ Zoll (oder 2 ½ Zoll) Festplatten, jede mit einem eigenen Festplatten Cache

Abb. 5.4.2
Aufbau eines Enterprise Storage Servers

Im Wesentlichen besteht jeder Enterprise Storage Server aus vier Teilen:

1. Front End, welches die Schnittstelle zu den Rechnern darstellt (Host Adapter). Beim Anschluss an System z Rechner sind dies Host Adapter für FICON-Kanäle. UNIX-Systeme verwenden Host Adapter für das Fibre Channel SCSI Protokoll.

2. Ein oder (aus Zuverlässigkeitsgründen) zwei Multiprozessoren plus Cache, welcher aus zwei Teilen besteht: Einem Cache für Daten, die gelesen werden können, und einem Cache für Daten, die geschrieben werden sollen. Letzterer heißt Non-Volatile Storage (NVS) und bezeichnet damit einen Cache, der extra gegen Stromausfälle und andere Störfälle gesichert ist, z.B. durch eine Pufferung mit Batterien .

3. Back-End-Schnittstellen (Device Adapter), welche bei den meisten heutigen Enterprise Storage Servern FC-AL (Fibre Channel Arbitrated Loop) Anschlüsse sind.

4. **Back End Store. Dieser besteht aus zahlreichen Festplatten und kann unterschiedlich sein. So bauen einige Hersteller SCSI-Platten ein, während andere Hersteller FATA oder SATA Disk Arrays bevorzugen. Jede der Platten verfügt noch einmal, ähnlich wie PC-Platten, über einen eigenen kleinen Cache.**

Heutige Enterprise Storage Server besitzen sehr große Caches von z.B. 256 GByte. Der Non-Volatile Storage kann deutlich kleiner sein, da er nur zum vorübergehenden Zwischenspeichern der Schreibzugriffe benötigt wird. Zu schreibende Daten werden dann asynchron auf den Back End Store geschrieben, ohne dass die Anwendung davon etwas bemerkt.

Der Enterprise Storage Server übernimmt die Funktion mehrerer Control Units.

Die bedeutesten Hersteller von Enterprise Storage Servern sind die Firmen EMC, IBM, Hitachi, MaxData und StorageTek, die im internen Aufbau alle große Ähnlichkeiten haben. Als Beispiel wird im folgenden der IBM DS8700 Enterprise Storage Server beschrieben. In vielen Fällen setzen Mainframe Installationen Enterprise Storage Server anderer Hersteller ein; besonders Enterprise Storage Server der Firma EMC sind häufig anzutreffen.

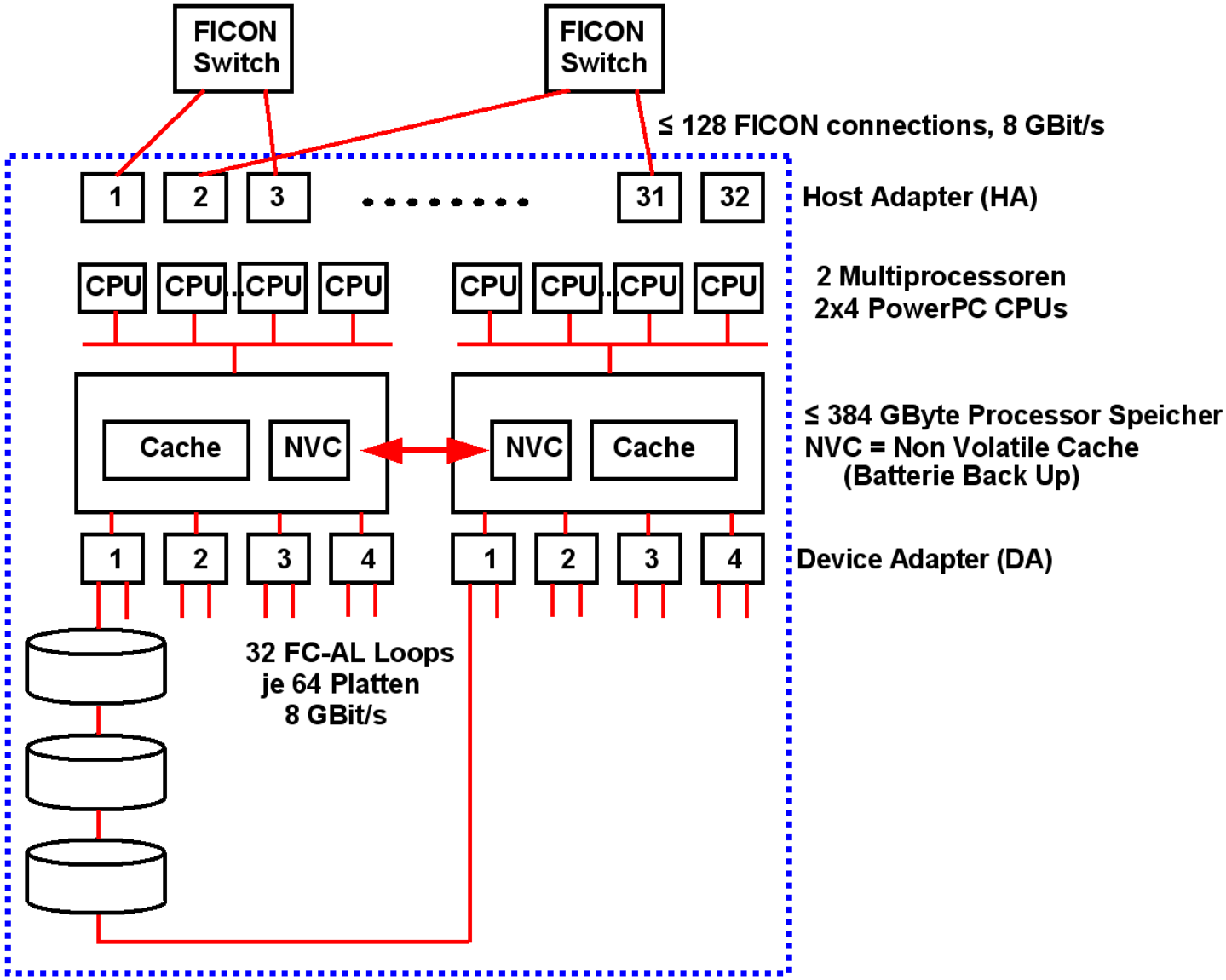


Abb. 5.4.3
IBM DS8700 Enterprise Storage Server

Zur Verbindung mit dem Host besitzt der DS8700 Enterprise Storage Server bis zu 32 Host Adapter (HA), die entweder je 4 FC-SCSI- Verbindungen oder je 4 FICON bzw. Fibre Channel Verbindungen nach außen implementieren.

Intern besteht der Storage Processor aus 2 unabhängigen Rechnern (Cluster), die jeweils aus (bis zu 4) PowerPC Prozessoren, dem Cache Storage und dem Non-Volatile Storage bestehen. Der Non-Volatile-Cache wird für die Zwischenspeicherung von Schreiboperationen benutzt. Die Idee ist: Wenn Daten einmal im ESS angekommen sind, gelten sie als sicher (persistent). Die Anwendung muss nicht das Schreiben auf den Festplatte abwarten.

Die beiden Cluster emulieren mehrere 3390 Control Units. Sie verfügen über getrennte Stromversorgungen und verhalten sich wie zwei unabhängige Rechner in der gleichen Box, außer dass sie über ein internes Netzwerk miteinander in Verbindung stehen. Zum Back Store besitzt jeder Cluster 8 Device Adapter mit je 4 FC-AL Ports. Die Adapter arbeiten immer paarweise, und die Plattenstränge (Disk Arrays) oder Ranks sind über eine FC-AL Loop mit den Device Adaptern verbunden.

FC-AL stellt eine serielle Kreisverbindung (Loop) für SCSI-Platten dar. Es existieren 2 Lese- und 2 Schreibverbindungen, von denen jede mit 40 MByte/s arbeitet, was eine Gesamtkapazität von 160 MByte/s ergibt.

Es werden 300, 450 oder 600 GByte Festplatten eingesetzt. Alternativ kann ein Teil auch aus Solid State Drives (SSD) bestehen. SSDs sind sehr teuer, bewähren sich aber für I/O-intensive Workloads. Sie ermöglichen eine bis zu 100fache Verbesserung des Throughput und bis zu 10fache bessere Antwortzeit als mit 15K U/min rotierende Festplatten. Sie verbrauchen auch weniger Energie als rotierende Festplatten.

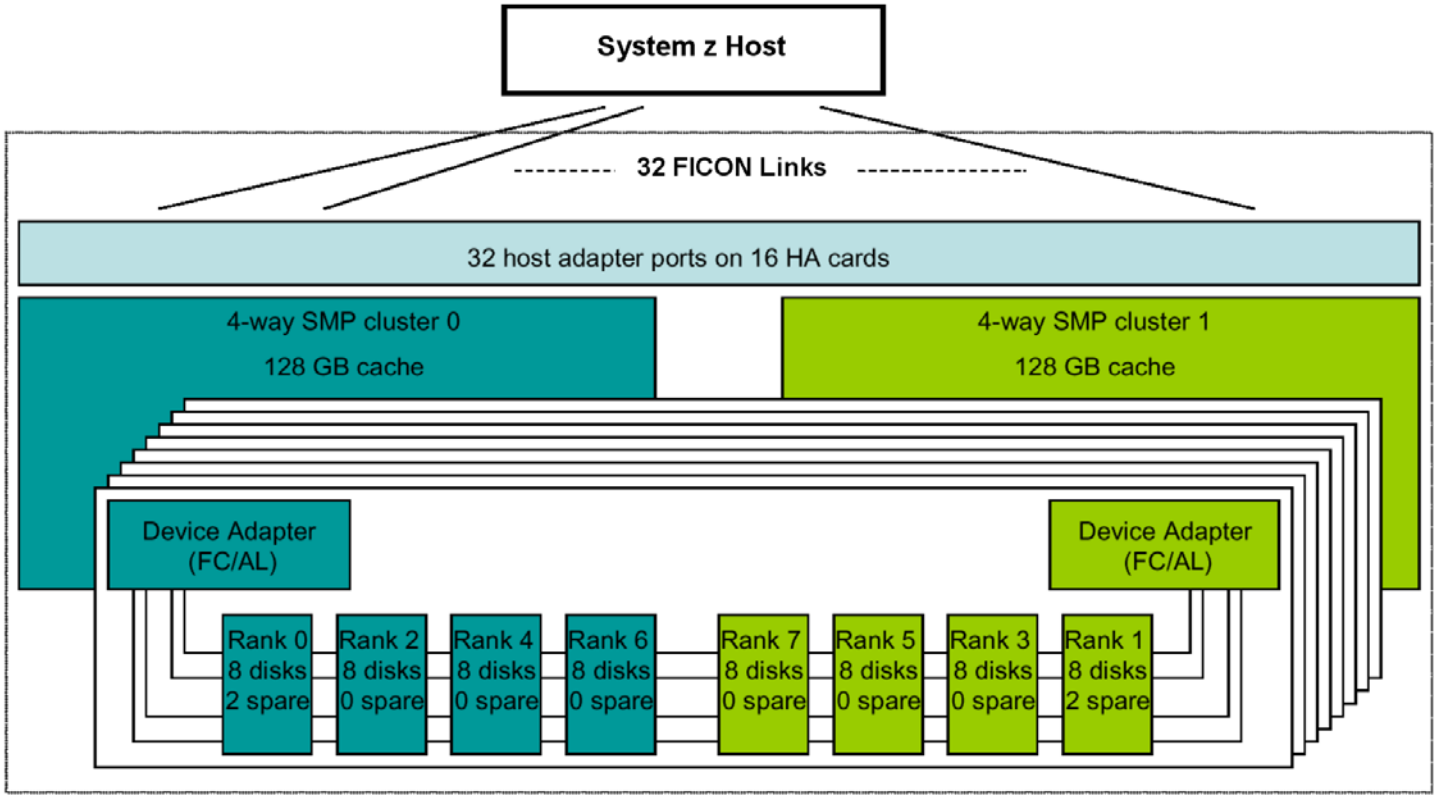


Abb. 5.4.4
Ranks eines DS8700 Enterprise Storage Servers

Abb. 5.4.4 demonstriert die Gliederung der Ranks in einer DS8700 Loop.

Dargestellt ist ein DS8700 Enterprise Storage Server mit 2 Cluster Prozessoren, je 4 x SMP, PowerPC, je 128 GByte Hauptspeicher/Cache sowie 2 x 8 Device Adaptoren . Jeweils eine FC-AL Loop mit 64 Festplatten ist an 2 Device Adaptoren angeschlossen. Jede FC-AL Loop besteht aus 2 Lese- und zwei Schreibverbindungen.

Jede FC-AL Loop besteht aus 64 aktiven Festplatten, aufgeteilt in 8 Ranks zu je 8 Platten. Die Ranks können als RAID 5, RAID 6 oder RAID 10 konfiguriert werden. Ein RAID 5 Rank würde aus 7 Platten für die Daten und einer Platte für Parity bestehen. Zusätzlich zu den 64 aktiven Platten einer FC_AL Loop existieren 2 Reserveplatten (Spares), die im Fehlerfall automatisch aktiviert (zugeschaltet) werden können. Die FC-AL Loop enthält somit 8 x 8 = 64 + 2 = 66 Festplatten.

Alle Festplatten sind als Hot Plug Steckplatten ausgeführt. Während des laufenden Betriebs können Platten entfernt und neu zugesteckt werden.

In dem hier gezeigten Beispiel enthält der Enterprise Storage Server 8 x 64 = 512 aktive Festplatten. Manche Modelle verfügen über 1024 aktive Platten. Wenn alle Plattenplätze mit 600 GByte Festplatten bestückt sind, ergibt dies eine gesamte Speicherkapazität von 633 TByte.

5.4.3 Nutzung durch unterschiedliche Server

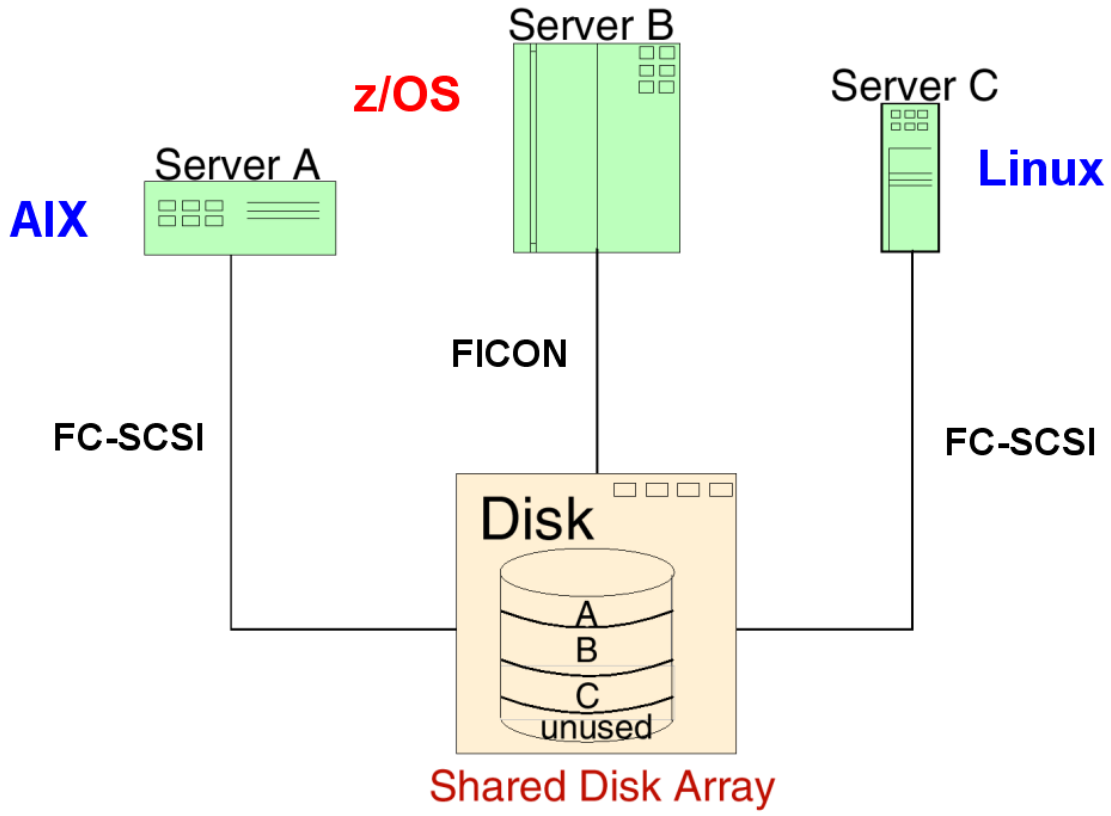


Abb. 5.4.5
Consolidated Storage

In einigen Situationen ist es attraktiv, mehrere physische Rechner mit unterschiedlichen Architekturen (z.B. z/OS, AIX, Solaris, Linux) an einen gemeinsamen Enterprise Storage Server, z. B. eine IBM DS8700 anzuschließen. Die Verbindungen zwischen den Servern und dem (den) Enterprise Storage Server(n) erfolgen über ein Fibre Channel Storage Area Network (SAN). Mainframes werden über FICON Front End Host Adapter, Unix und Linux Rechner über FC-SCSI Front End Host Adapter angeschlossen.

Unter Hot Plug versteht man den Austausch einer defekten Festplatte eines RAID Verbundes im laufenden Betrieb (oftmals auch als Hot Swap bezeichnet).

- **Beim Hot Plug wird die Ersatzfestplatte im Betrieb manuell getauscht**
- **Während Hot Plug besteht weiterhin volle Datenverfügbarkeit**
- **Die Einbindung der Ersatzfestplatte geschieht automatisch durch das Hot Plug Programm**

5.4.4 Datenarchivierung

In der Wirtschaft und öffentlichen Verwaltung hat die Datenarchivierung eine große Bedeutung. Der Gesetzgeber verlangt für manche Daten eine Archivierungsdauer von 30 Jahren. Bei Versicherungen kann die Archivierungsdauer auch noch länger sein.

Gelegentlich werden CDs und DVDs für die Archivierung eingesetzt. Das Standard Archivierungsmedium sind jedoch Magnetband Kassetten (Cartridges).

Bei manchen archivierten Daten fordert der Gesetzgeber eine Garantie, dass Daten nicht nachträglich modifiziert worden sind. Magnetbandkassetten sind deshalb in zwei Ausführungen verfügbar:

- **Read/Write Kassetten können mehrfach beschrieben werden. Daten können überschrieben werden.**
- **WORM (Write Once, Read Many) Kassetten können nur einmal beschrieben werden.**

Ein in jede Kassette eingebauter Mikroprozessor stellt die Eigenschaften einer WORM Kassette sicher. Weitere technologische Eigenschaften garantieren, dass eine betrügerische Änderung von Daten in einer WORM Kassette unmöglich ist.

Abb. 5.4.6
IBM 3592 WORM and R/W Kassetten

Die IBM 3592 Magnetbandkassette (Cartridge) hat Abmessungen von 24.5 mm H x 109 mm W x 125 mm D und verwenden ein ½ Zoll breites Magnetband mit einer Länge von 825 Meter. Die Speicherkapazität beträgt bis zu 4 TByte. Mainframes benutzen Datenkomprimierung für die Magnetbandspeicherung, was die Speicherkapazität zusätzlich um einen Faktor 2 – 3 erhöht.

IBM garantiert eine Lebensdauer der Kassetten von 10 Jahren. 30 Jahre sind wahrscheinlich. Die meisten Unternehmen kopieren archivierte Magnetband-Daten viel häufiger um, um auf der sicheren Seite zu sein (z.B. alle 5 Jahre).

Abb. 5.4.7
Magnetbandeinheit für 3592 WORM and R/W Kassetten

Der IBM System TS1130 Tape Drive (Magnetbandeinheit) liest und schreibt 3592 Kassetten mit einer Datenrate von 160 MByte/s.

5.4.5 Magnetband Library

Abb. 5.4.8
Ansicht der IBM 3494 Tape Library

Früher hat ein menschlicher Operator die Magnetbandkassetten in eine Magnetbandeinheit eingelegt und wieder entfernt. Heute verwendet man hier Magnetbandroboter, auch als Tape Libraries bezeichnet.

Die 3494 Enterprise Tape Library unterstützt ein Maximum von 6,240 Kassetten für eine Speicherkapazität von mehreren PetaBytes (PByte). Sie kann bis zu 132 Tape Drives in einer System z Umgebung unterstützen.

Der 3494 Cartridge Accessor mit dem dualen Gripper holt Kassetten aus einem Regal und legt sie in einen der Tape Drives ein.

Abb. 5.4.9
Dual Gripper 3494 Cartridge Accessor

Es können bis zu 265 Cartridge Exchanges/Stunde mit einem einzige Greifarm (Gripper), und bis zu 610 Exchanges/Stunde mit einem dualen Gripper und dualen aktiven Accessors durchgeführt werden.

5.4.6 Virtual-Tape-Server

Ein Virtual-Tape-Server (VTS, anderer Namen Virtual Tape Library oder Virtual I/O, VIO) ist ein Speicher auf Basis eines Festplatten Arrays.

Ein VTS stellt sich für angeschlossene Computer wie eine Tape-Library dar. Die Anzahl der Libraries und darin enthaltener Slots und Bandlaufwerke sind dabei frei konfigurierbar. Virtuelle Bänder können zumeist von dem VTS direkt auf „echte“ Bänder geschrieben werden, ohne dass die Backup-Software oder ein Server an diesem Vorgang beteiligt ist.

Magnetbandkassetten tolerieren nur eine begrenzte Anzahl von Zugriffen. Wenn überhaupt auf die Daten einer Magnetbandkassette zugegriffen wird, sind zahlreiche Zugriffe auf einzelne physische Datensätze oder Slots häufig die Folge. Eine VTL arbeitet so etwa wie ein mittels einer Festplatte implementierter Cache für eine Magnetbandkassette.

Weiterhin ist es möglich, Backup-To-Disk-Konzepte in bestehenden Datensicherungsumgebungen, die in der Regel auf Bandlaufwerken basieren, einzubinden. Ein Beispiel sind temporär genutzte Data Sets in der Stapelverarbeitung. JES nimmt hierfür in der Regel Magnetbandlaufwerke an.

5.4.7 Drucker Ausgabe

Mainframe Installationen haben sehr unterschiedliche Anforderungen bezüglich Printer I/O.

Die Annahme, dass mit wachsender Bildschirmausgabe und wachsender digitaler Speicherung von Dokumenten der Papierverbrauch rückläufig sein würde, hat sich fast nirgendwo bewahrheitet. Im Allgemeinen kann man davon ausgehen, dass der Papierverbrauch in Unternehmen und staatlichen Organisation Jahr für Jahr nach wie vor steigt.

Beispiele sind Konto Auszüge und Überweisungsbelege im Bankenbereich, Mitteilungen der staatlichen Rentenversicherung oder Abrechnungen von Krankenkassen. Derartige Dokumente werden teilweise auf dem Postweg in Briefumschlägen (Kuvert) versandt.

Drucker können wie Plattenspeicher über eine Printer-Control Unit und FICON Channel Kabel mit einer I/O Card eines Mainframe Systems verbunden werden. Mehrere Hersteller liefern derartige Produkte. Unter z/OS existiert ein generischer I/O Driver für die Drucker-Ansteuerung. Dieser überträgt ein entsprechendes Kanalprogramm an die Printer Control Unit.

Alternativ kann Print Output als Datei über einen Netzanschluss an einen getrennten Druck Server übergeben werden. Weitergehende Formatierungen erfolgen dezentral und unabhängig von z/OS.

Es existieren weitere exotische Ein/Ausgabegeräte. Ein Beispiel sind zentrale Check Lese Geräte für die automatische Verarbeitung von großen Mengen von Überweisungen und Bank-Schecks.

Abb. 5.4.10
Fiducia AG, Karlsruhe, Druck- und Kuvertierzentrum

Die Fiducia AG in Karlsruhe ist einer der großen deutschen IT Dienstleister. Ihr Rechenzentrum mit mehreren Mainframes bedient vor allen genossenschaftliche Einrichtungen wie Volksbanken und Raiffeisenbanken.

Das Druck- und Kuvertierzentrum der Fiducia AG in Karlsruhe ist über Control Units an die dortigen Mainframes angeschlossen. Dies sind einige der Leistungsmerkmale:

- **19 Mainframe Drucksysteme mit einer Gesamtleistung von 9 200 Seiten pro Minute.**
- **14 Kuvertierstraßen mit einer Gesamtleistung von 40.500 Kuvertierungen pro Stunde**
- **378 Millionen DIN-A4-Seiten pro Jahr**
- **140 Millionen kuvertierte Sendungen pro Jahr, davon 90 Millionen kuvertierte Kontoauszugssendungen**
- **1.500 Kunden für Druckdienstleistungen**

5.5 Weiterführende Information

Life Demo eines Festplattenspeicher Zugriffsarms

http://www.youtube.com/watch?v=L0nbo1VOF4M

Ein sehenswertes Video demonstriert ein altes Rechenzentrum in den 90er Jahren

http://www.youtube.com/watch?v=Cwj6pfhWBps&feature=related

In 1992 IBM released the 3495 Tape Library using a bright yellow robot which this video shows in great detail. This is a great look back to technology in the early 1990s.

http://www.youtube.com/watch?v=GwMn7YpF8r8&feature=fvwrel

Eine ähnliche Demo mit der Ultrium Tape Unit wird gezeigt in

http://www.youtube.com/watch?v=lNa_D1Hljww&feature=related

Youtube Video zu den Vorteilen eines Storage Area Network:

http://www.youtube.com/watch?v=KjqgHylfGL4

Youtube Video zum Thema IBM Impact Printer circa 1990

http://www.youtube.com/watch?v=kEWcvmUluyE

IBM war Marktführer mit mechanischen High Speed Druckern der Serien 1403 und 3203 von 1959 bis in die 1990's, siehe auch

http://www.youtube.com/watch?v=Cwj6pfhWBps

6. VSAM

6.1 Arten von Datasets

6.1.1 Datenspeicherung

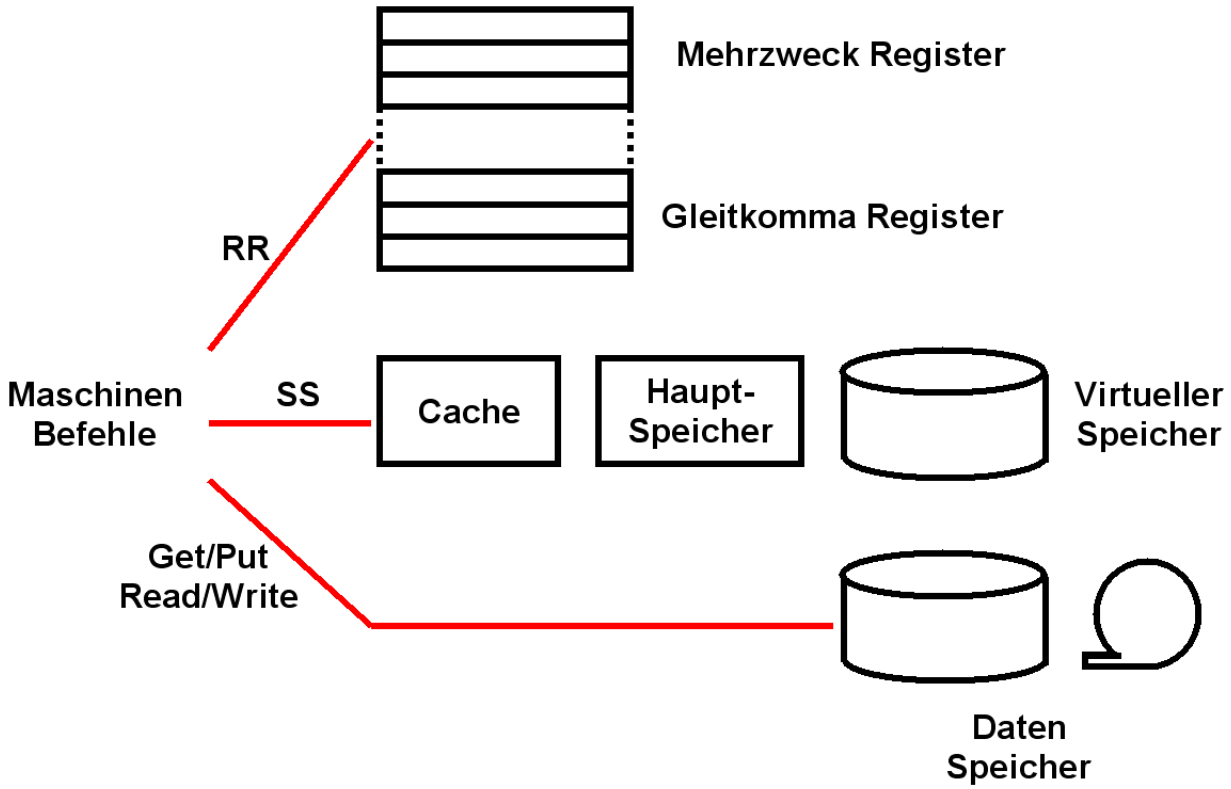


Abb. 6.1.1
Unterschiedliche Arten der Datenspeicherung

Ein Rechner speichert Daten auf drei unterschiedliche Arten:

- **In Mehrzweck- bzw. Gleitkommaregistern**
- **Im Hauptspeicher. Dies ist in der Regel ein virtueller Speicher mit einem externen Seitenspeicher (paging datasets) als Backup.**
- **In Dateien (Files und Datasets) oder Datenbanken.**

Um auf Mehrzweck Register (General Purpose Register) oder Gleitkommaregister (Abschnitt 1.3.4) zuzugreifen, benutzt er vor allem Maschinenbefehle im RR Format. Um auf den Hauptspeicher zuzugreifen, benutzt er vor allem Maschinenbefehle im SS Format. Um auf den Datenspeicher (Festplatte oder Magnetband) zuzugreifen, benutzt er Gruppen von Maschinenbefehlen (Makros), wie Get/Put oder Read/Write. Datenspeicher enthalten Dateien oder Datasets.

Bei einer Datei kann es sich um ein Quellprogramm, eine Makrobibliothek, ein ausführbares Programm, eine lineare Folge von Bits oder um eine Gruppierung von Datensätzen (data records) handeln, die von einem Programm verarbeitet werden.

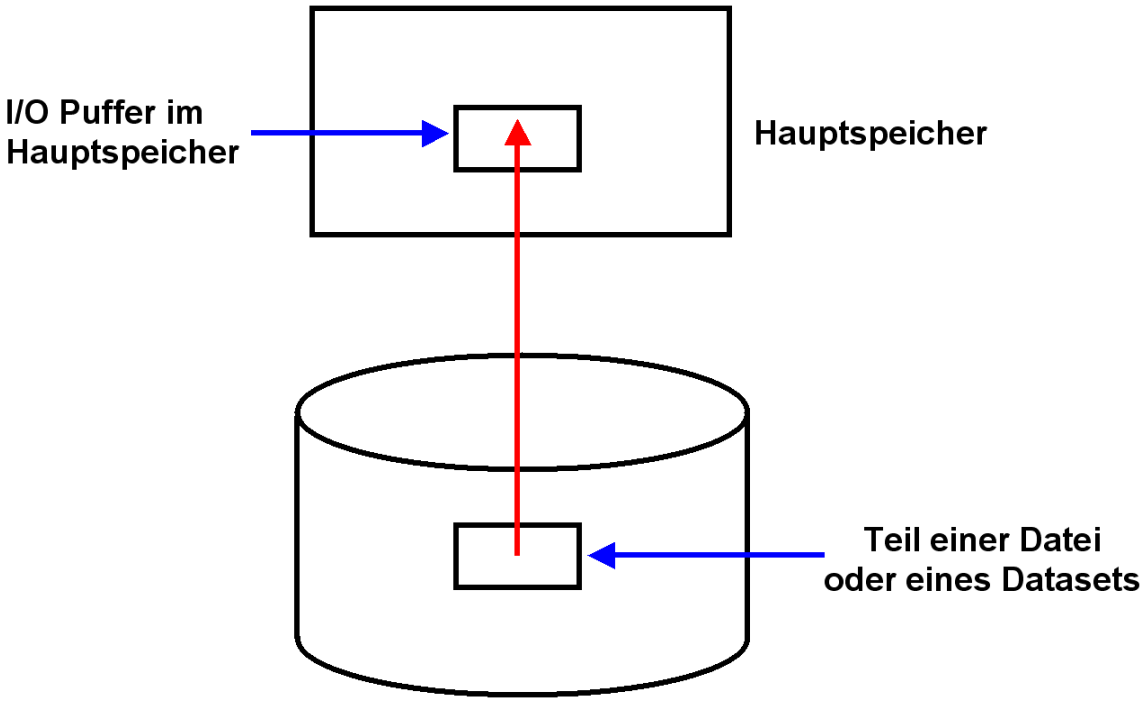


Abb. 6.1.2
Teile einer Datei werden im I/O Puffer gespeichert

Dateien, die zu verarbeitende Daten enthalten, können sehr groß sein. Nur ein kleiner Teil passt in der Regel in den Hauptspeicher. Bei einem READ Zugriff auf den Plattenspeicher wird eine Gruppe von Bytes aus einer Datei von der Festplatte in einen als I/O Buffer bezeichneten Bereich innerhalb des Hauptspeichers gelesen.

6.1.2 Festplattenspeicher versus File System

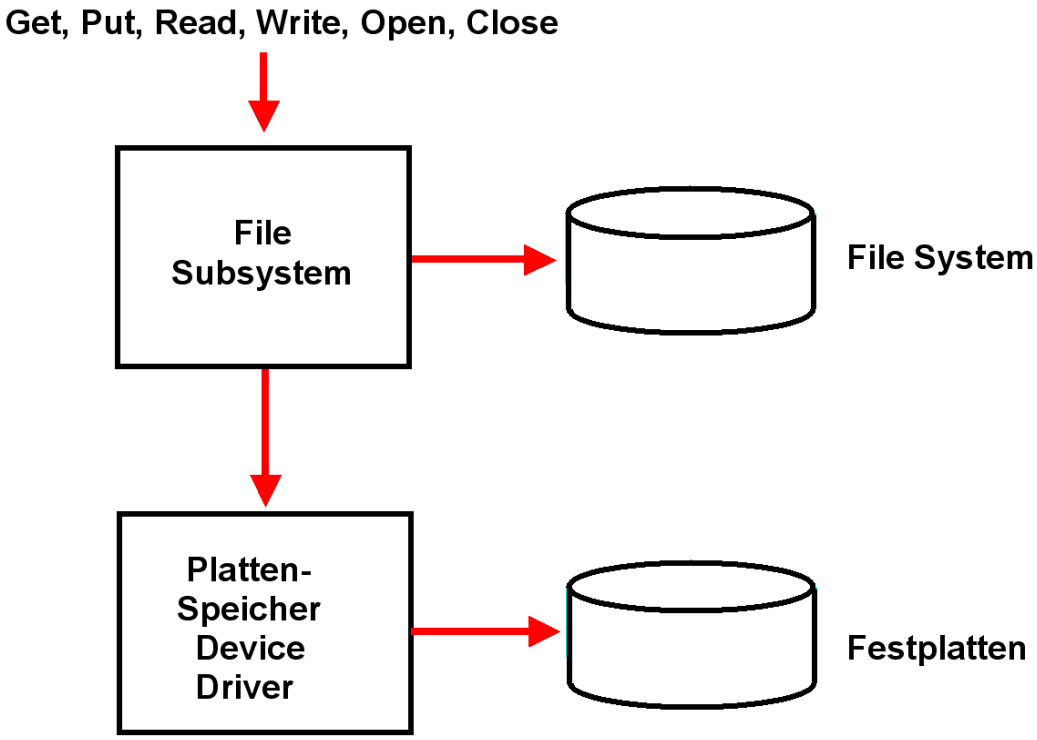


Abb. 6.1.3
Der Überwacher bildet das File System auf den Device Driver ab.

Anwendungsprogramme greifen selten oder nie direkt auf einen Festplattenspeicher zu. Stattdessen greifen sie auf eine abstrakte Struktur zu, die je nach Funktionsumfang als File System oder als Datenbank bezeichnet wird.

Es ist die Aufgabe einer Komponente des Betriebssystems, des File Subsystems oder des Datenbanksystems, diese abstrakte Struktur auf die konkrete Struktur der Festplatten abzubilden.

Das File Subsystem ist eine Komponente des Betriebssystems und arbeitet mit der logischen Sicht eines File Systems. Ähnlich ist ein Datenbanksystem eine Komponente des Betriebssystems und arbeitet mit der logischen Sicht einer Datenbank.

Der Plattenspeicher Device Driver (und/oder verwandte Komponenten) bildet ein oder mehrere File Systeme (oder Datenbanken) auf einem oder mehreren Festplattenspeichern ab.

6.1.3 Unix File I/O

Betriebssysteme wie Unix, Linux oder Windows speichern Daten in Files (Dateien). Die Identifizierung von Dateien erfolgt über Dateiverzeichnisse, die selbst wie Dateien aussehen und wie Dateien behandelt werden können.

Linux Files sind Bestandteil eines Linux File Systems. Es existieren viele unterschiedliche Linux File Systeme, (z.B. Ext2, Ext3, *Ext4,* ReiserFS), die jedoch alle sehr ähnliche Eigenschaften haben.

Eine Unix/Linux File ist eine unstrukturierte Menge von Bytes (strukturlose Zeichenketten), auf die mit Hilfe eines Dateinamens zugegriffen werden kann. Kleine Unix Files werden bei einem Zugriff oft vollständig in den Hauptspeicher gelesen.

Große Unix Files werden oft sequentiell gelesen. Es ist die Aufgabe einer Anwendung, die Abbildung von Byte-Sequenzen auf „Records“ (Strukturelemente des Programms, z.B. Structures in C/C++) vorzunehmen. Ein Direktzugriff wird mit Hilfe von Pointern programmiert, die gezielt auf eine bestimmte Zeichenfolge in der Datei aufsetzen.

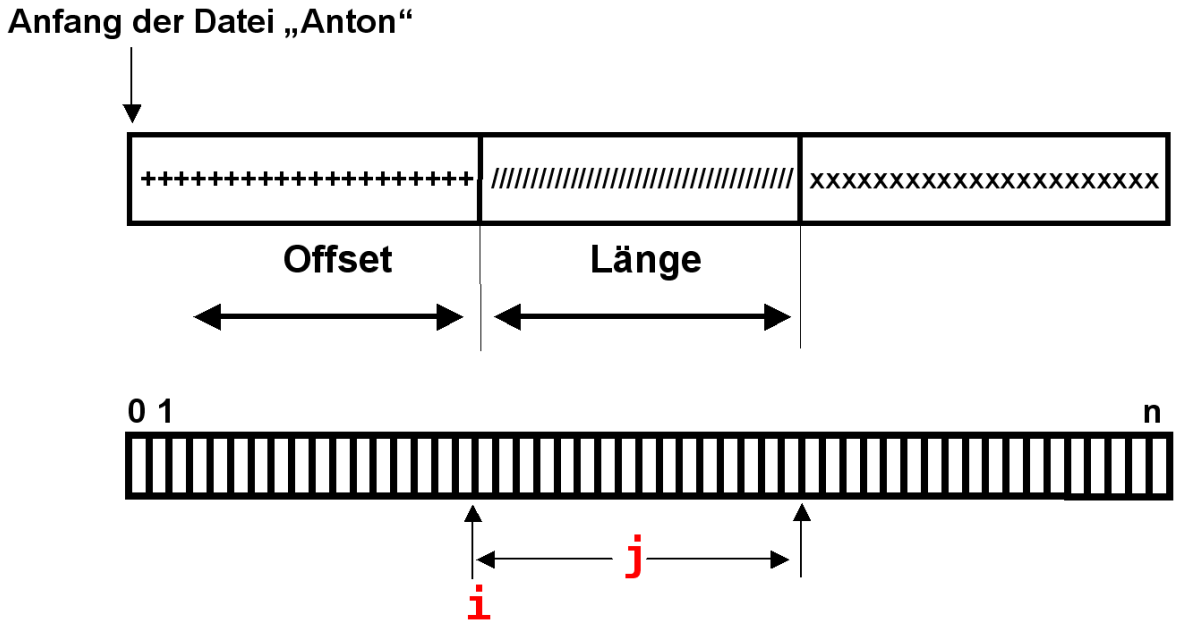


Abb. 6.1.4
Adressierung einer Folge von Bytes mit Hilfe von 2 Zeigern

Eine Unix, Linux oder Windows File besteht aus einer linearen Folge von Bytes mit den Adressen 0...n. Bei einem Lesezugriff

READ (Anton, `i`, `j`)

wird aus der Datei mit dem Namen Anton eine Gruppe von `j` aufeinanderfolgenden Bytes in einen Puffer im Hauptspeicher gelesen, beginnend mit Byte `i`.

`i` stellt einen Zeiger in die Datei dar und wird vom Anwendungsprogramm verwaltet.

Mit Hilfe des hier dargestellten Mechanismus können zusätzliche Programmpakete andere Dateiorganisationen implementieren, z.B. eine index-sequentielle Organisation.

6.1.4 Record I/O

Im Gegensatz zu einer Unix File speichert eine Unix (oder Windows) SQL Datenbank die Daten strukturiert in der Form von Tables (Relationen), Rows, Columns usw. Jede Row innerhalb einer Unix SQL Datenbank Tabelle stellt eine strukturierte Datenmenge dar, die aus Feldern besteht. Eine derartige strukturierte Datenmenge wird als „Record“ bezeichnet.

Ein Record (andere Bezeichnung Struktur) fasst verschiedene Komponenten zusammen, die im Allgemeinen unterschiedliche Datentypen besitzen. Meist besitzen Strukturen eine überschaubare Anzahl an Komponenten, die Anzahl ist jedoch nicht begrenzt.

Ein Beispiel ist der Datentyp `struct` in der Programmiersprache C++

```
struct beispiel {
  int personal_nummer;
  int alter;
  float Gehalt;
};
```

Anmerkung: Eine relationale Datenbank speichert Tabellen in irgendeiner Form auf einem Plattenspeicher ab. Es ist die Aufgabe der Datenbanksoftware, die entsprechende Abbildung vorzunehmen. Eine Unix/Linux Datenbank wird häufig mit Hilfe eines proprietären File Formats implementiert. Dieses greift direkt auf die Spuren einer Festplatte zu und benutzt hierzu eine Unix Feature. den „raw“ Zugriffsmechanismus.

Standard-Zugriffsmethoden für die Record- Ein/Ausgabe sind in das UNIX-System eingebaut und werden von der UNIX-Programmiersprache C/C++ unterstützt. Beispiel:

```
#include <stdio.h>
#define MAX_LEN 80
int main(void)
{
   FILE *fp;
   int len;
   char buf[MAX_LEN + 1];
   fp = fopen("MY_LIB/MY_FILE", "rb, type = record");
   while ((len = fread(buf, 1, MAX_LEN, fp)) != 0)
   {
      buf[len] = '\0';
      printf("%s\n", buf);
   }
   fclose(fp);
   return 0;
}
```

z/OS verwendet auch Files und File Systeme, die vergleichbar mit den Unix/Linux Files und File Systemen sind. z/OS unterhält hierfür u.a. die Unix System Services (USS), die über ein Unix-kompatibles Hierarchical File-System verfügen.

Die allermeisten z/OS Daten werden jedoch in „Datasets“ gespeichert. Ein Dataset ist im Gegensatz zu einer File eine strukturierte Menge von Records. Eine z/OS-Anwendung manipuliert Records an Stelle einer unstrukturierten Menge von Bytes.

z/OS verwendet wie Unix gleichzeitig mehrere Filesysteme (als „Access Methods“ bezeichnet), mit Namen wie BSAM, BDAM, QSAM, ISAM, PDS, PDSe usw. Die wichtigste Access Method (File System) hat den Namen VSAM (Virtual Storage Access Method).

Eine z/OS „Access Method“ definiert das benutzte Filesystem und stellt gleichzeitig Routinen für den Zugriff auf die Records des Filesystems zur Verfügung. Das Filesystem wird durch die Formatierung eines physischen Datenträgers (z.B. Plattenspeicher) definiert. Bei der Formatierung der Festplatte wird die Struktur des Datasets und die zu benutzende Access Method festgelegt.

Dataset Zugriffe benutzen an Stelle eines Dateiverzeichnisse „Kontrollblöcke“, welche die Datenbasis für unterschiedliche Betriebssystemfunktionen bilden. Die Kontrollblöcke haben exotische Namen wie VTOC, DSCB, DCB, UCB, deren Inhalt vom Systemprogrammierer oder Anwendungsprogrammierer manipuliert werden können.

Im Gegensatz zu Unix/Linux werden Datenbanken unter z/OS ebenfalls in Datasets abgespeichert. Eine Datenbank ist also eine höhere Abstraktionsebene als ein Dataset. Während DB2 unter z/OS hierfür normale z/OS Datasets verwendet, benutzen Unix Datenbanken hierfür häufig Files mit proprietären Eigenschaften sowie direkte (raw) Zugriffe auf Dateien.

6.1.5 Datasets und Files in z/OS und Unix

z/OS	UNIX
DataSets	HFS files
Record-oriented (F(B), V(B), U)	Byte-oriented
Several general methods VSAM (ESDS, KSDS, RRDS, LDS) Non-VSAM (SAM, PDS-E)	Hierarchical file structure Directory / subdir / filename / Path info max. 1023 char.
Dataset name (max. 44 chars.) UPPER CASE names	File name max. 256 char. Mixed case, case sensitive names

Abb. 6.1.5
z/OS Datasets und Unix Files

Die hier dargestellte Gegenüberstellung der Unix files und z/OS Datasets fasst die Unterschiede nochmals zusammen.

6.1.6 Arten von Datasets

Unter Dataset Organisation versteht man die Art, wie die Elemente einer Datei zueinander angeordnet sind. Eine Anordnung von Elementen bedingt eine Struktur.

z/OS Datasets haben unterschiedliche Organisationsformen, die unterschiedliche Arten des Zugriffs auf die Records ermöglichen. Diese sind:

1. Sequentiell (SEQUENTIAL)
Die Records in der Datei sind fortlaufend organisiert. Datensätze (Records) werden in der Reihenfolge der Ankunft geschrieben. Der Zugriff auf die Datei erfolgt sequentiell in der Reihenfolge in der die Records gespeichert sind. Um auf den 15. Datensatz zuzugreifen muss das System die 14 vorhergehenden Datensätze lesen. Sequentielle Datasets sind eine Voraussetzung für die Speicherung auf Magnetband oder eine Drucker-Ausgabe, können aber auf DASD gespeichert werden (was häufig geschieht).

2. Indiziert (INDEXED)
Nach dem Laden befinden sich die Datensätze in ihrer natürlichen Reihenfolge auf dem Datenträger. Die einzelnen Records in der Datei werden mit unterschiedlichen Schlüsseln gespeichert. Durch eine zusätzlich gespeicherte Indextabelle ist ein direkter Zugriff auf individuelle Records über diesen Schlüssel möglich. Indexsätze werden durch die Organisation (Access Method) automatisch erstellt und verwaltet.

3. Direkt (RANDOM)
Die Records in der Datei sind fortlaufend nummeriert. Der Zugriff auf einen Record erfolgt direkt über die Rekord-Nummer, die das Anwendungsprogramm kennen muss.

4. Partitioned
Datasets mit einer Partitioned Organisation speichern Daten in untergliederten Komponenten, die als „Member“ bezeichnet werden. Auf einen Member kann direkt über seinen Namen zugegriffen werden. Ein Member verhält sich ähnlich wie eine File in einem Unix File System.

6.1.7 Sequentielle Data Sets

```
DECLARE
   1 MASTER
      2  CATALOG_NO   CHARACTER (5),
      2  UNIT_PRICE   FIXED (4,2),
      2  TITLE        CHARACTER (30),
   1 TRANSACTION,
      2  CATALOG_NO   CHARACTER (5),
      2  QUANTITY     FIXED (4),
      2  CUSTOMER     CHARACTER (40),
   1 REPORT,
      2  CUSTOMER     CHARACTER (40),
      2  CATALOG_NO   CHARACTER (5),
      2  TITLE        CHARACTER (30),
      2  QUANTITY     FIXED (4),
      2  UNIT_PRICE   FIXED (4,2),
      2  TOTAL_PRICE  FIXED (6,2),
   PAGE_COUNT FIXED (2) INITIAL (1);
```

Abb. 6.1.6
Record Declaration in einem PL/I Programm

Die in einem PL/I Programm deklarierten z/OS Records werden z.B. sequentiell in einem Dataset gespeichert. Auf sie kann mit einer einfachen Open – Read – Write – Close Sequenz zugegriffen werden. Die Manipulation mit Offset, Länge wie bei Unix Dateien entfällt bzw. erfolgt automatisch.

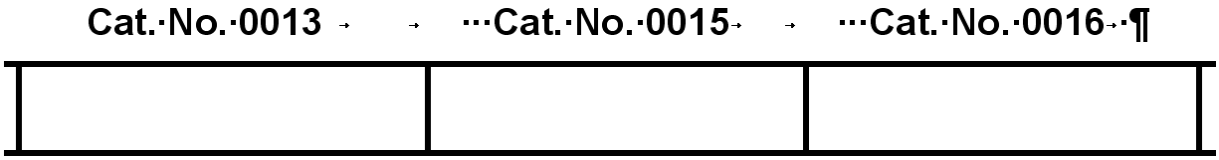


Abb. 6.1.7
Sequentielle Speicherung der Records auf einem Datenträger

6.1.8 Access Methods

Der Betriebssystem Kernel verfügt über als „Access Methods“ bezeichnete Routinen. Um z.B. auf einen bestimmten Record in einer Index-sequentiell organisierten Datei zuzugreifen, braucht ein Anwendungsprogramm lediglich einen READ Befehl mit Angabe des Datei Namens und des Index-Wertes des Records. Die Access Method macht den Rest.

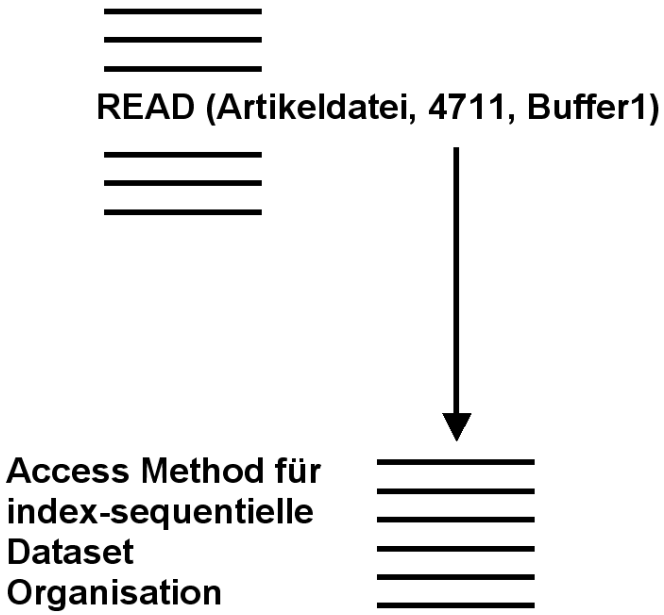


Abb. 6.1.8
Aufruf einer Access Method

Das Anwendungsprogramm gibt nur den Namen der Datei (Artikeldatei), den Indexwert eines Records (4711) sowie den Namen eines Speicherbereichs im Hauptspeicher (Buffer1) an, in den ein Record mit der Artikelnr. = 4711 gespeichert werden soll. Die Access Method des Kernels macht den Rest.

Unter Unix/Linux müsste der Anwendungsprogrammierer entweder die Funktion der Access Method selbst schreiben, oder ein vorgefertigtes Unterprogramm benutzen, welches aus einem sequentiellen Bit Strom den gewünschten Record herausfiltert.

6.1.9 Blocking

In der Anfangszeit der Mainframe Entwicklung benutzte man „Basic“ Access Methods. Wenn im Anwendungsprogramm ein READ oder WRITE Befehl ausgeführt wurde, wurde durch das Betriebssystem ein einziger Record von der Festplatte gelesen oder auf die Festplatte geschrieben.

Sehr bald ging man zu „Queued“ Access Methods über. Bei der Ausführung eines READ Befehls wurde von der Festplatte immer eine Gruppe benachbarter Records gelesen und in einen entsprechend größeren Buffer Bereich des Hauptspeichers gelesen. Mit etwas Glück befand sich der gewünschte Record bei einem folgenden READ Befehl bereits im Hauptspeicher, und man konnte sich den aufwendigen Lesevorgang vom Plattenspeicher ersparen.

Hierzu fasste man mehrere Records (auch als „logische“ Records bezeichnet) zu einem Block (auch als „physischer Record bezeichnet) zusammen. Beim Zugriff auf die Festplatte wurde immer ein Block an Stelle eines einzelnen Records gelesen.

Bei der Definition eines Datasets mittels Allocate machen wir beispielsweise diese Angaben:

```
Primary quantity  . .  16          (In above units)
Secondary quantity     1           (In above units)
Directory blocks  . .  2           (Zero for sequential data set) *
Record format . . . .  FB
Record length . . . .  80
Block size  . . . . .  320
Data set name type     PDS         (LIBRARY, HFS, PDS, LARGE, BASIC, *
```

Abb. 6.1.9
Angabe der Block Größe

Wir haben die (logische) Record Länge mit 80 Byte definiert, und haben eine Block (physischer Record) Größe von 320 Bytes festgelegt. Jeder Block nimmt also 4 logische Records auf, und bei einem Lese Zugriff auf den Festplattenspeicher werden immer 320 Bytes ausgelesen.

Für eine Diskussion über optimale Block Größen siehe Kapitel 5, Input/Output, Abschnitt 5.3.

6.1.10 Dataset Namen

Jeder Dataset benötigt einen Namen, mit dem er gespeichert und verarbeitet werden kann.

Regeln:

- **Ein Name besteht aus einem oder mehreren (Normalfall) Qualifiern, die jeweils durch einen Punkt miteinander verbunden sind.**
- **Jeder Qualifier besteht aus 1 - 8 Zeichen, wobei Buchstaben, Ziffern und die drei Zeichen #, § und $ erlaubt sind. Der Name darf nicht mit einer Ziffer beginnen.**
- **Die Gesamtlänge des Namens ist maximal 44 Zeichen, dabei zählen die Punkte mit.**
- **Für die Namen der Member in einem Partitioned Dataset gelten die gleichen Regeln wie für einen Qualifier.**

Gültige Dataset Namen:

USER1.TEST.DATA
SYS1.PARMLIB
MAX.PROGRAM.VERSION1

Ungültige Dataset Namen:

USER3-X.BEISPIEL.LIST	**(Bindestrich nicht erlaubt)**
8TEST.LISTE	**Ziffer am Anfang)**
EMIL.TESTPROGRAMM1	**Qualifier > 8 Zeichen)**

Ein Member in einem Partitioned Dataset (PDS) wird folgendermaßen angesprochen:

USER1.PROGRAM.LIBRARY(PROG1)

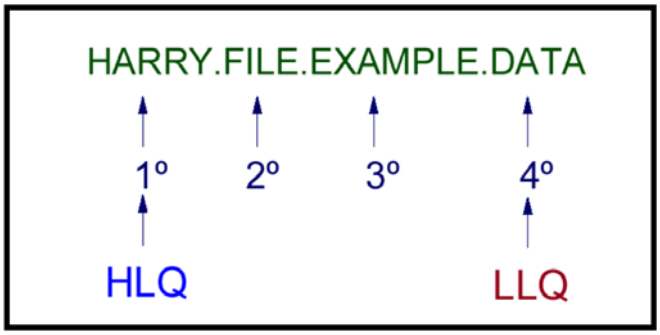


Abb. 6.1.11
High und Low Level Qualifier

Der Name eines Datasets kann aus einem oder aus mehreren Segmenten bestehen. Die Segmente werden als Qualifier bezeichnet und durch Dezimalpunkte getrennt. Jedes Segment des Namens repräsentiert eine Ebene (Level) der Qualifikation. Zum Beispiel besteht der Dataset Name HARRY.FILE.EXAMPLE.DATA aus 4 Segmenten. Das erste Segment (HARRY) wird als High-Level Qualifier (HLQ) ; das letzte Segment (DATA) wird als Low-Level Qualifier (LLQ) bezeichnet.

z/OS kann so konfiguriert werden, dass der High Level Qualifier stets identisch mit der User ID ist, z.B. PRAK123. In vielen z/OS Installationen wird davon Gebrauch gemacht, so auch bei dem z9 Rechner des Lehrstuhls Technische Informatik.

6.1.11 Non-VSAM und VSAM Datasets

Bei der Einführung von OS/360 in den 60er Jahren waren bereits unterschiedliche Rekord-organisierte Datasets verfügbar, z.B. BSAM, QSAM, BDAM, ISAM usw. Für sie wurde später der gemeinsame Name „non-VSAM“ eingeführt. 1973 führte IBM VSAM ein. VSAM-Datasets müssen sich auf einem Plattenspeicher (DASD) befinden und können auf vier verschiedene Arten organisiert sein:

- **ESDS - ähnlich einem sequentiellen Dataset**
- **KSDS - ähnlich einem indexsequentiellen Dataset**
- **RRDS - ähnlich einem direkt organisierten Dataset**
- **LDS - ein ESDS ohne jegliche Steuerinformationen**

VSAM sollte die bisherigen non-VSAM Datasets ablösen. Da aber noch immer viele Uralt Anwendungen im Einsatz sind, ist das bis heute nicht vollständig geschehen. Neue Anwendungsprogramme verwenden fast immer VSAM.

Ähnlich führte OS/360 das Partitioned Dataset (PDS) Format ein. Später wurde die verbesserte Version PDSe eingeführt. Noch später entstand HFS (Hierarchical File System), ein Unix kompatibles File System, welches ebenfalls als partitioned bezeichnet werden kann, aber einen anderen Einsatzzweck wie PDSe hat.

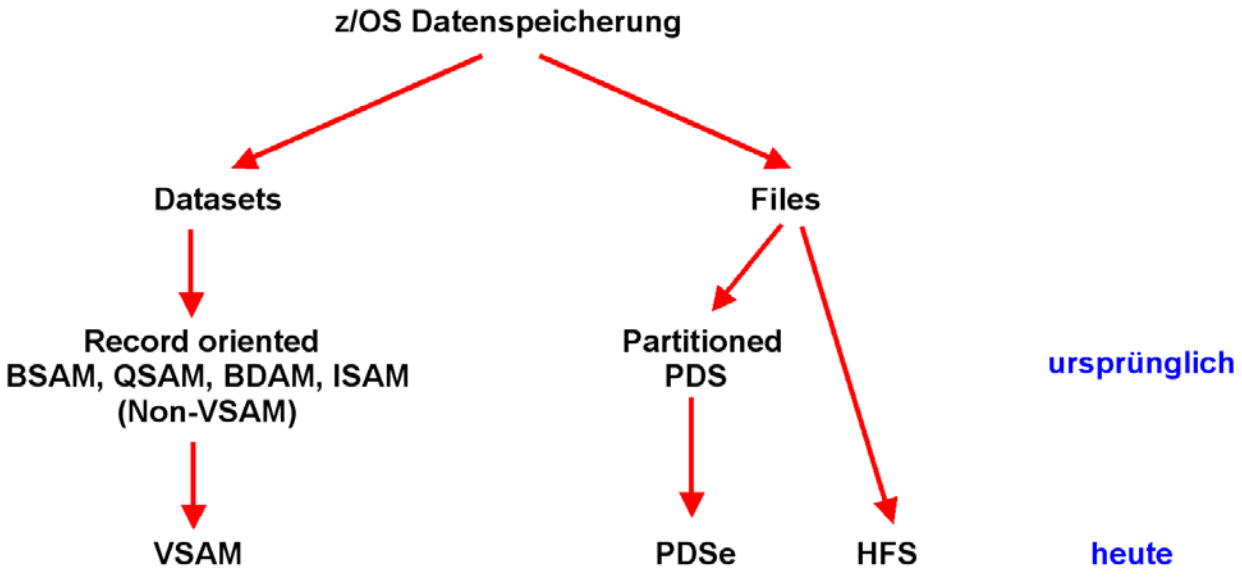


Abb. 6.1.10
Übersicht über die z/OS Datasets und Files

z/OS bezeichnet PDS ebenfalls als einen Dataset, obgleich ein PDS möglicherweise, nicht aber notwendigerweise, aus Records besteht.

6.1.12 Partitioned Dataset (PDS)

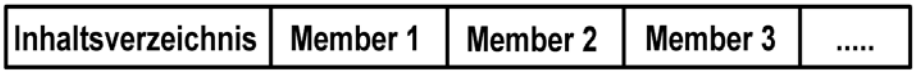


Abb. 6.1.12
Partitioned Dataset (PDS)

Ein Partitioned Dataset (PDS) ist eine Art Mini-Unix File-System. Es verfügt über ein einfaches Inhaltsverzeichnis (Directory) und Platz für mehrere Dateien, die als „Member“ bezeichnet werden. Jeder einzelne Member des Datasets kann ausgewählt, geändert oder gelöscht werden, ohne die anderen Member zu beeinflussen.

Ein Member, wird in sequentieller Reihenfolge geschrieben, und es wird dem Member ein NAME im Inhaltsverzeichnis des PDS zugeordnet.

Ein Member ist entweder eine strukturlose Zeichenkette (Byte Stream) ähnlich einer Unix File, oder kann aus einer Folge von Records bestehen.

Das Inhaltsverzeichnis (Directory) ist ein Index, der vom Betriebssystem verwendet wird, um ein Member in einem PDS zu lokalisieren. Alle Einträge (Namen der Member) im Inhaltsverzeichnis sind in alphabetisch aufsteigende Reihenfolge angeordnet. Die einzelnen Member können jedoch in jeder beliebigen Reihenfolge innerhalb des Partitioned Dataset gespeichert werden.

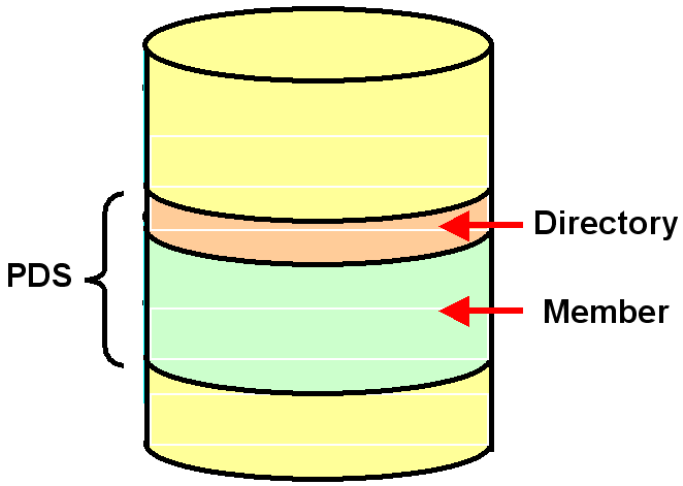


Abb. 6.1.13
Spezifikation für Bibliotheken

Ein PDS Dataset besteht aus zwei Bereichen, dem Directory und dem Member Bereich.

Das Verzeichnis (Directory) besteht aus 256-Byte-Blöcken. Beim Anlegen (Allocate) eines neuen PDS Datasets wird nicht nur die Größe des Datasets sondern auch die Größe des Directory statisch festgelegt.

Das Directory enthält Zeiger auf die einzelnen Member. Im Directory steht der Member-Name an der Stelle, wo er in aufsteigender alphabetischer Reihenfolge einzuordnen ist. Das Betriebssystem greift auf einen Member zu, indem es das Directory nach dem entsprechenden Eintrag (Namen) durchsucht.

Der Directory Bereich enthält Member-Namen und -Adressen. Der Member Bereich enthält sequentielle Member.

Ein PDS wird häufig als LIBRARY oder Bibliothek bezeichnet. Partitioned Datasets werden oft für Programm-Bibliotheken eingesetzt, wobei ein Member ein Programm in dieser Bibliothek darstellt. Nutzungen sind:

- **Program Source Libraries,**
- **Object Libraries,**
- **Load Module Libraries,**
- **Macro Libraries, sowie**
- **Text Files.**

Angenommen, es wurde eine Anforderung an z/OS gestellt, das Member CCC aus der Datei MY.PDS aufzurufen.

z/OS durchsucht sequentiell das Verzeichnis, bis es den Eintrag für CCC findet. Der Verzeichniseintrag enthält die Adresse auf dem Festplattenspeicher für CCC. z/OS geht zu dieser Adresse, lädt CCC in den Hauptspeicher und ruft anschließend CCC auf.

Speicherplatz, der für z/OS Datasets allocated wird, beginnt immer am Anfang einer Festplattenspur (Track). Mit Hilfe von PDS kann mehr als eine Datei auf einer Spur gespeichert werden. Hiermit kann Festplatten-Speicherplatz optimal genutzt werden, wenn viele Files vorhanden sind, die alle wesentlich kleiner als eine Spur sind. Eine Spur eines 3390 DASD speichert 56,664 Bytes.

Der ursprüngliche PDS wurde 1989 durch PDSe (Partitioned Dataset extended) ersetzt/erweitert. Unterschiede zwischen dem ursprünglichen Partitioned Dataset (PDS) und der verbesserten Version Partitioned Dataset extended (PDSe) sind für den Benutzer weitgehend unsichtbar und betreffen vor allem das Leistungsverhalten. Während es möglich ist, auch heute noch PDS zu benutzen, wird fast immer PDSe verwendet. Wenn man von einem PDS spricht, ist damit in der Regel ein PDSe gemeint.

PDSe Member werden in 4 KByte Seitenrahmen gespeichert. Jeder Member benötigt eine ganzzahlige Anzahl von Seiten an Speicherplatz, mindestens aber einen Seitenrahmen.

6.1.13 Unterschied zwischen PDS/PDSe und Unix File System

Ein PDS oder PDSE besteht aus einem Directory und einzelnen Members. Der PDSE Dataset mit dem Namen MYLIB kann die Member ABLE, BAKER und CHARLIE enthalten. Die Bezeichnung der einzelnen Member ist MYLIB.(ABLE), MYLIB.(BAKER) und MYLIB.(CHARLIE). Die Hierarchie ist einstufig:

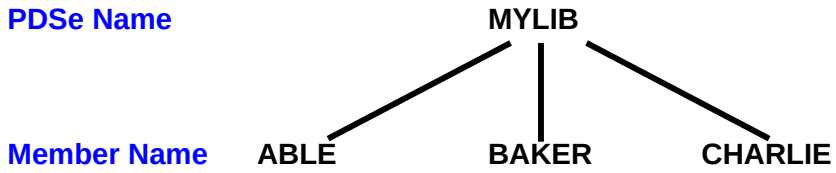


Abb. 6.1.14
PDSe Hierarchie

Eine mehrstufige Hierarchie ist im Gegensatz zu Unix/Linux nicht möglich. Eine Unix/Linux File könnte die Bezeichnung

```
anton/berta/caesar/dora/emil/friedrich.exe
```

haben.

Während ein Unix/Linux System in der Regel nur ein einziges monolytisches File System benutzt, kann ein z/OS System viele PDS Files enthalten. Dies vereinfacht die Administration und macht die Namensgebung für Programmbibliotheken übersichtlicher.

6.1.14 z/OS Hierarchical File System

Neben PDS und PDSE existiert unter z/OS ein „Hierarchical File System“.

Ein z/OS Hierarchical File System (HFS) hat die gleichen Eigenschaften wie ein traditionelles Unix File System. Files sind in eine mehrstufige Hierarchie gegliedert.

Im Zusammenhang mit der häufig stattfindenden Rezentralisierung besteht der Bedarf, existierende Unix Anwendungen nach z/OS zu portieren. z/OS HFS wird vor allem eingesetzt, wenn dabei die File System Struktur dieser Unix Anwendungen erhalten bleiben soll. Bei neuen Anwendungen ist es möglich, für die Speicherung von Daten an Stelle von VSAM oder non-VSAM Datasets das Hierarchical File System zu spezifizieren. Davon wird aber nur selten Gebrauch gemacht. Neue z/OS Anwendungen verwenden in der Regel PDSe für die Speicherung von Programmen und VSAM für die Speicherung von Daten.

6.2 VSAM Struktur

6.2.1 VSAM

VSAM (Virtuell Storage Access Method) ist der wichtigste z/OS Data Set Typ. VSAM wurde speziell für das Arbeiten in einer z/OS virtual Storage Umgebung entwickelt.

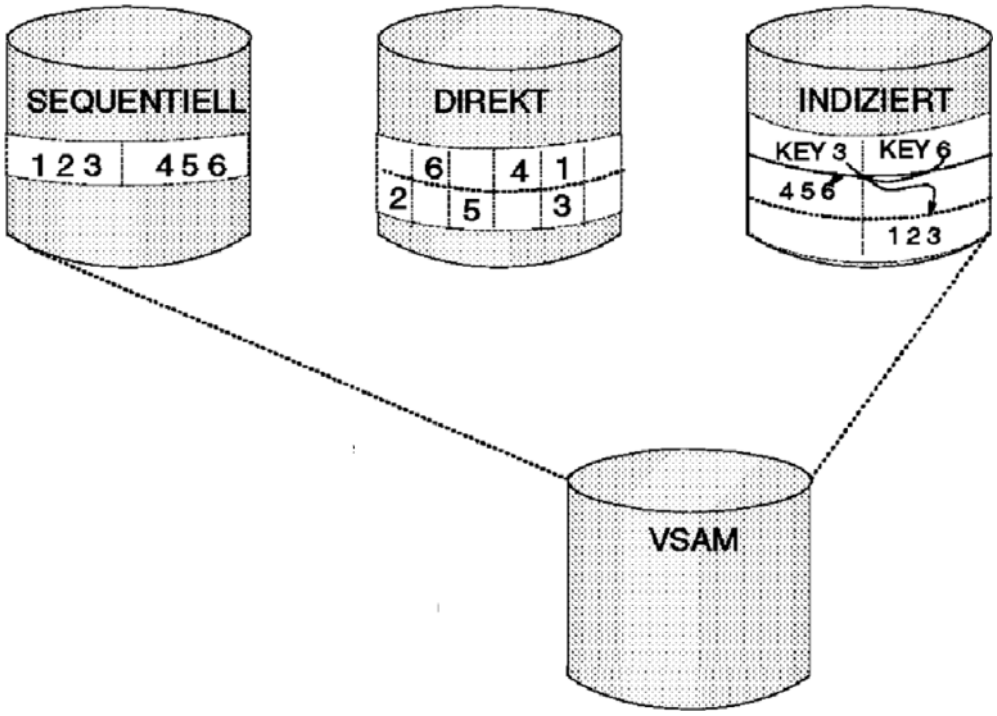


Abb. 6.2.1
Unterschiedliche Arten von VSAM Datasets

Der Begriff Virtual Storage Access Method (VSAM) beschreibt sowohl einen Datensatz-Typ (Organisation) als auch die Zugriffsmethode, mit der auf verschiedene Anwender-Datentypen zugegriffen wird. Als Zugriffs Methode stellt VSAM weit komplexere komplexe Funktionen zur Verfügung als andere bisherige Access Methods. VSAM speichert Daten in einem speziellen Format auf der Festplatte ab, das nicht für andere Access Methods nicht verständlich ist.

VSAM wird hauptsächlich für die Speicherung von Anwendungsdaten verwendet. In vielen Fällen wird VSAM in Situationen eingesetzt, in denen unter Linux oder Windows der Einsatz einer relationalen Datenbank erforderlich wäre. VSAM ist an dieser Stelle deutlich performanter. VSAM ist nicht für Source-Programme, JCL, oder ausführbare Module vorgesehen. VSAM Datasets können nicht routinemäßig mit ISPF angezeigt oder bearbeitet werden.

VSAM ist der Nachfolger der älteren non-VSAM Data Sets. Es wurde mit dem Ziel, entwickelt die bisher verwendeten sequenziellen, Index-sequenziell und direkten non-VSAM Datasets zu ersetzen.

Neue Anwendungen verwenden meistens VSAM. Jedoch sind immer noch eine sehr große Anzahl von non-VSAM Datasets in Benutzung.

6.2.2 VSAM Dataset Typen

Es existieren die folgenden unterschiedlichen Arten von VSAM Data Sets:

Entry Sequence Data Set (ESDS)

Diese Form von VSAM speichert Records in sequentieller Reihenfolge. Datensätze können sequentiell abgerufen werden. ESDS wird auch als Container verwendet, um ein vollständiges z/OS Hierarchical File System zu speichern.

Key Sequence Data Set (KSDS)

Dies ist die häufigste Verwendung von VSAM. Jeder Datensatz enthält ein (oder mehrere) Schlüsselfelder. Ein Datensatz kann über seinen Schlüssel gelesen (oder eingeschoben) werden. Dies ermöglicht einen Zugriff über einen Index auf die Daten. Die Records können von unterschiedlicher Länge sein.

Relative Record Data Set (RRDS)

Dieses VSAM-Format ermöglicht das Abrufen von Datensätzen nach ihrer Nummerierung; Datensatz 1, Satz 2, und so weiter. Dies ermöglicht einen random (direkten) Zugang zu den Daten, vorausgesetzt das Anwendungsprogramm kennt die Nummer des Records. Das Anwendungsprogramm muss eine Möglichkeit haben, die gewünschte Datensatz-Nummer zu ermitteln.

In den meisten RRDS Datasets haben alle Records die gleiche Länge. Ein VRRDS (variable relative Rekord Dataset) ermöglicht Records mit unterschiedlicher Länge.

Linear Data Set (LDS)

Dies ist ein Bit-Stream Datensatz (ähnlich einer Unix-Datei). Eine beträchtliche Anzahl von z/OS System Funktionen benutzen dieses Format; es wird aber nur selten von Anwendungsprogrammen verwendet.

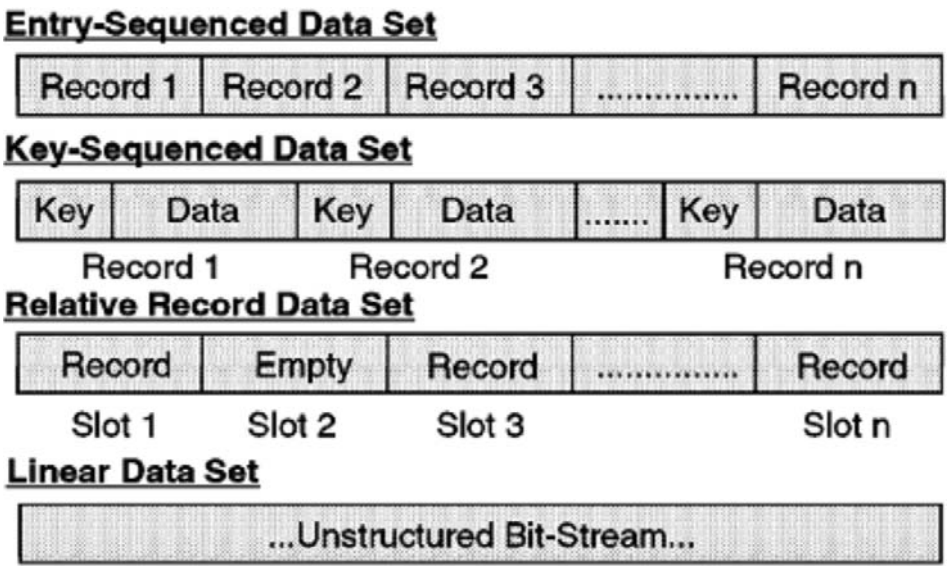


Abb. 6.2.2
Übersicht der VSAM Datasets

Ein Relative Record Data Set wird auch als Direct Record Data Set bezeichnet.

ESDS Datasets sind für die sequentielle Verarbeitung von Daten optimiert.

6.2.3 Benutzung von VSAM

Die VSAM Zugriffsmethode (Access Method) dient als Schnittstelle zwischen Anwendungsprogrammen und dem Betriebssystem. Ein Anwendungsprogramm ruft VSAM Zugriffsmethode Routinen als normale Unterprogramme auf.

In Assembler Language Programmen erfolgt die Benutzung durch einen Aufruf von VSAM Makros. Bei der Benutzung von Hochsprachen wie Cobol, C++ oder PL/1 wandelt der Compiler I/O-Anweisungen (z.B. WRITE) in Aufrufe an die entsprechenden VSAM Routinen um. Wenn die I/O-Anforderung abgearbeitet wurde, wird die Steuerung an das Anwendungsprogramm zurückgegeben.

Beim Zugriff auf einen Record durch ein Anwendungsprogramm geht VSAM durch die folgenden Schritte:

1. **VSAM interpretiert den Aufruf des Anwendungsprogramms und bestimmt, welche Dienste erwünscht sind.**
2. **VSAM erstellt die erforderlichen Input oder Output (I/O) request(s) an das Betriebssystem.**
3. **Das Betriebssystem führt die physischen I/O-Operation(en) zwischen Festplatte und Hauptspeicher durch.**
4. **VSAM lokalisiert und extrahiert die gewünschten Daten zur Rückgabe an das Anwendungsprogramm.**

Zwei wichtige Alternativen beim Zugriff auf einen VSAM Record sind:

1. **Beim Zugriff auf einen Festplattenspeicher (Direct Access Storage Device, DASD) transportiert VSAM (bzw. z/OS) einen größeren Block (Control Interval) mit mehreren Records vom/zum virtuellen Hauptspeicher. Der vom Anwendungsprogramm gewünschte Record befindet sich möglicherweise in einem Control Interval, welches bereits in den virtuellen Speicher geladen wurde. Eine physische I/O-Operationen ist in diesem Fall nicht erforderlich.**
2. **Aufgrund der Art wie VSAM Daten speichert und der Vielfalt der Verarbeitungsoptionen kann es sein, dass mehrere physische I/O-erforderlich sind, um einen einzigen logischen Record in den virtuellen Speicher zu laden.**

6.2.4 Control Interval

VSAM Datasets werden anders als non-VSAM Datasets auf dem Plattenspeicher abgespeichert..

VSAM speichert Records in Blöcken, die als Control Intervals bezeichnet werden Ein Control Interval (CI) ist ein zusammenhängender (contiguous) Bereich auf einem DASD (disk drive), welcher sowohl Records als auch Steuerinformation speichert. Daten werden zwischen Hauptspeicher und Plattenspeicher immer ganze Control Intervals bewegt. Die Größe der CIs kann von einem VSAM Dataset zum nächsten VSAM Dataset unterschiedlich sein; alle CIs innerhalb eines spezifischen VSAM Datasets haben jedoch die gleiche Länge. Eine sehr häufig benutzte Control Interval Größe ist 4 KByte, identisch mit der Größe eines 4 KByte Virtual Storage Page Frames.

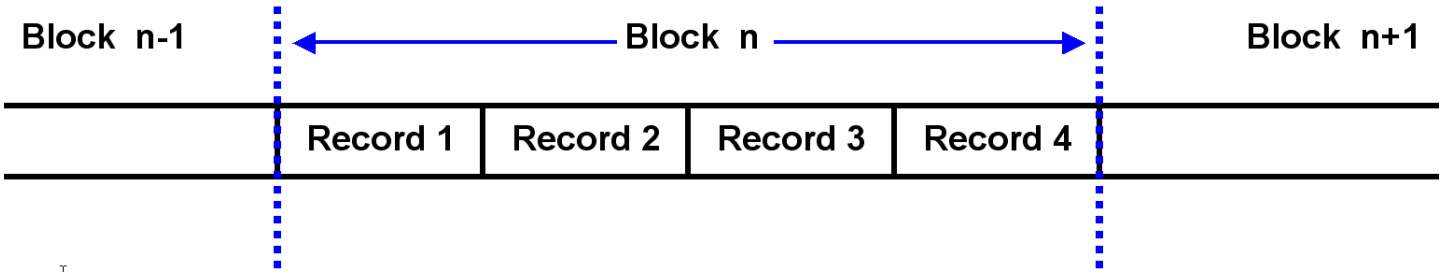


Abb. 6.2.3
Ein Non-VSAM Block bestehend aus mehreren logischen Records

In einer normalen Queued Access Method (z.B. QSAM) fasst man mehrere (logische) Records zu einem Block (physischer Record) zusammen. Der physische Record ist die Dateneinheit, die zwischen Festplatte und I/O Puffer im Hauptspeicher gelesen und geschrieben wird.

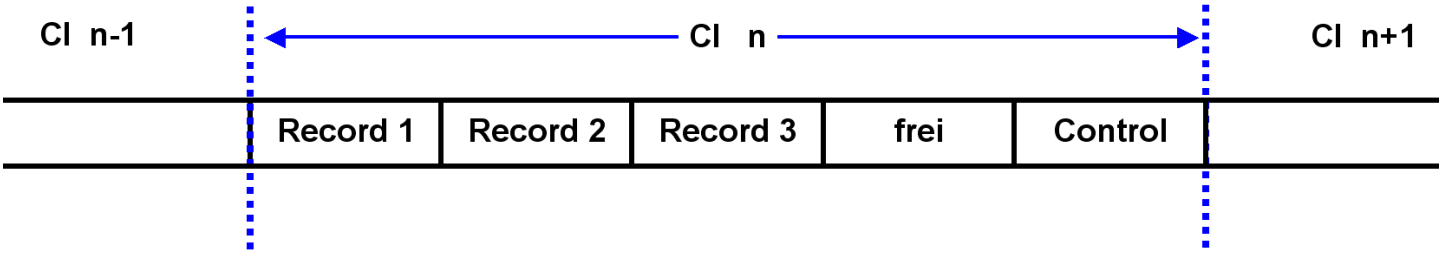


Abb. 6.2.4
Struktur eines Control Intervals

Ein VSAM Control Interval (CI) unterscheidet sich von einem Block (physischer Record dadurch, dass es neben mehreren (logischen) Records noch zusätzliche Control Information über die Records des CI speichert. Außerdem kann in einem CI freier Platz für die zukünftige Aufnahme weiterer Records vorhanden sein.

Ein Control Interval besteht aus

- **mehreren logischen Records,**
- **freien Platz, plus einem**
- **Control Interval Definition Field (CIDF) sowie mehreren**
- **Record Definition Fields (RDFs).**

In so fern unterscheidet sich ein CI von den Blöcken (physischen Records) anderer Dataset Access Methods, bei denen ein Block lediglich eine Aneinanderreihung von (logischen) Records enthält.

Abb. 6.2.5
RDF und CIDF Felder

Ein Control Intervals (CI) enthält typischerweise mehrere (logische) Records. Die Standard CI Größe ist 4KByte, kann aber bis zu 32KByte betragen.

Das Control Interval enthält

- · Records,
- · Ungenutzten (freien) Speicherplatz,
- · Record-Descriptor Felder (RDF) und
- · ein einziges Control Interval Descriptor Feld (CIDF).

Die CIDF ist ein 4-Byte-Feld. Es enthält Informationen über die Größe und Lokation des freien Speicherplatzes. Ein RDF ist ein 3-Byte-Feld. Es beschreibt die Länge der Records. VSAM benutzt häufig Records mit variabler (unterschiedlicher) Länge. In diesem Fall existiert ein RDF für jeden Record. Für eine Folge von Records mit fester Länge benutzt man 2 RDFs. Ein RDF gibt den Wert der Länge an, der zweite RDF besagt, wie viele Records mit gleicher Länge vorhanden sind.

In so fern unterscheidet sich ein CI von den Blöcken (physischen Records) anderer Dataset Access Methods, bei denen ein Block lediglich eine Aneinanderreihung von (logischen) Records enthält.

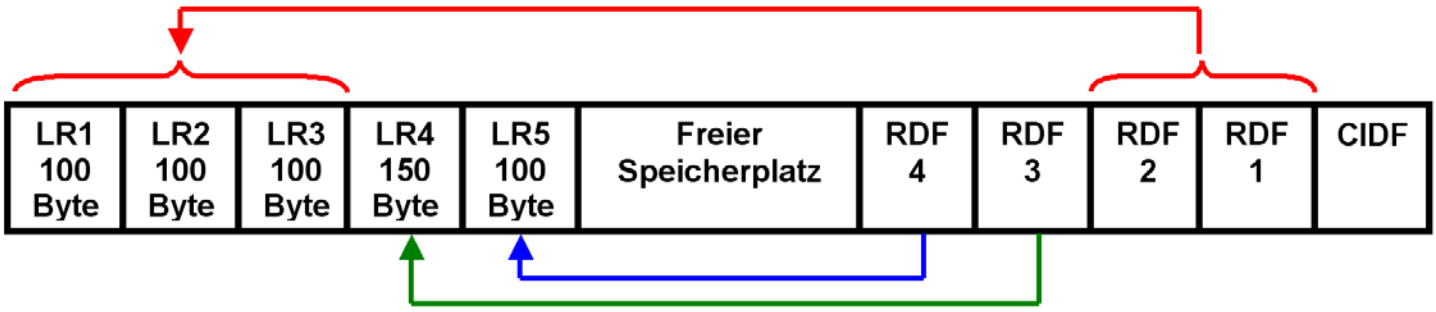


Abb. 6.2.6
Aufeinanderfolgende (contiguous) Records mit identischer Länge

Das in Abb. 6.2.6 gezeigte Beispiel zeigt ein Control Interval mit mehreren Records sowie frei verfügbaren Platz für die Aufnahme weiterer Records. Der Datenbereich enthält 5 logische Records mit jeweils einer Länge von 100 Bytes, 100 Bytes, 100 Bytes, 150 Bytes und 100 Bytes.

Es existiert 1 Control Interval Definition Field (CIDF), welches die Lokation und Länge des freien (nicht genutzten) Speicherplatzes angibt. Von den 4 Record Definition Fields (RDF) definieren die beiden ersten RDFs die Länge (100 Bytes) und die Anzahl (3) der ersten Gruppe von 3 Records mit je einer Länge von 100 Bytes. Die beiden nächsten RDFs definieren die Länge der folgenden beiden Records.

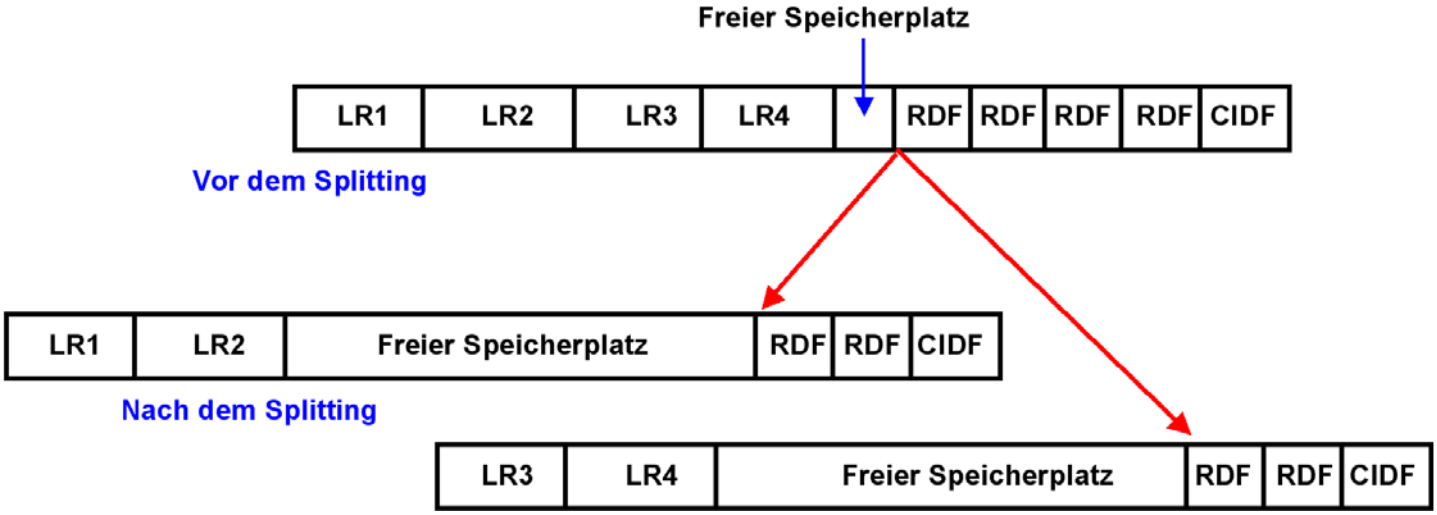


Abb. 6.2.7
Splits

Splits treten auf, wenn der ungenutzte Speicherplatz innerhalb eines Control-Interval nicht mehr groß genug ist, um einen zusätzlichen neuen Rekord aufzunehmen. In diesem Fall wird die Hälfte der Datensätze in ein neues Control Interval bewegt. Dies stellt mindestens 50% freien Speicherplatz in jedem Control Interval bereit. Anschließend werden die Control Information Felder in jedem Control Interval aktualisiert.

Wenn Speicherplatz in einer Control Area knapp wird, wird die Control Area ebenfalls gesplittet.

6.2.5 Control Area

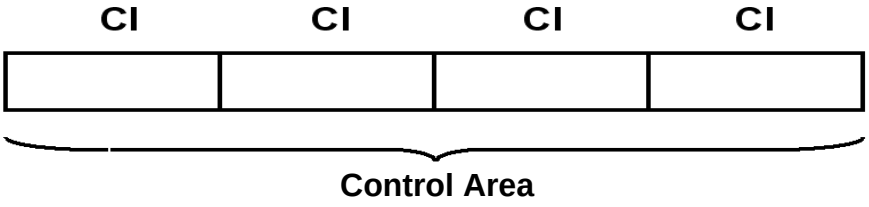


Abb. 6.2.8
Mehrere Control Intervals werden zu einer Control Area zusammengefasst

Mehrere Control Intervals in einem VSAM Dataset werden in einem zusammenhängender (contiguous) Bereich auf einem DASD (disk drive) zusammengefasst, der als Control Area bezeichnet wird. Eine Control Area hat häufig die Größe eines Zylinders (15 Spuren) einer 3390 Festplatte, oder 849960 Bytes.

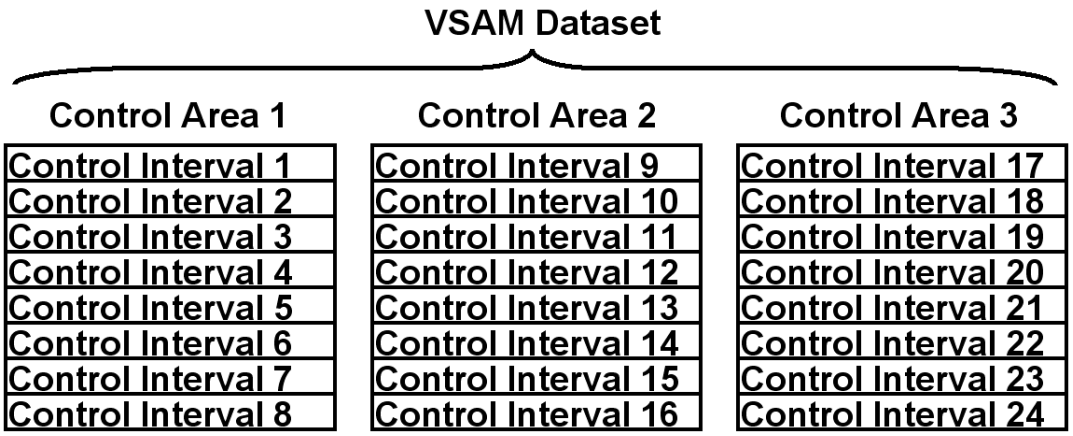


Abb. 6.2.9
Beispiel eines VSAM Data Sets mit 3 Control Areas

Eine Control Area ist ein spezifisches VSAM Construct. Ein Control Area wird von zwei oder mehreren Control-Intervals gebildet, und in einem zusammenhängenden (contiguous) DASD Bereich gespeichert. Ein VSAM Dataset besteht häufig aus einer größeren Anzahl von Control Areas.

In vielen Fällen hat eine Control Area die Größe eines IBM Typ 3390 Festplatten Zylinders (849 960 Bytes). Die Größe der CA wird definiert, wenn der Dataset allocated wird.

Ein VSAM Dataset kann aus vielen Control Areas bestehen. Mit einer Control Interval Größe von 4 KByte ist eine maximale Größe eines VSAM Datasets von 16 TByte möglich.

6.2.6 Buffering

Buffering ist einer der wichtigsten Aspekte was die I/O-Leistung betrifft. Ein VSAM Buffer ist ein virtueller Speicherbereich, in den ein CI während einer I/O-Operation übertragen wird.

Ein Bufferpool ist ein Satz von Buffern, von denen jeder ein CI aufnimmt. Ein Bufferpool kann mehrere CIs des gleichen VSAM Datasets enthalten. Zusätzlich kann ein Anwendungs-programm auf mehrere VSAM-Datasets zugreifen. So kann der Bufferpool häufig CIs von unterschiedlichen VSAM-Datasets enthalten.

Ein Ressourcen Pool ist ein Bufferpool mit zusätzlicher Steuerinformation. Diese beschreibt den Pool und die CIs in dem Ressourcen Pool.

Eine Datenbank nutzt auch Bufferpools. Datenbanken wie DB2 und IMS transportieren Daten zwischen Festplatte und Hauptspeicher in Blöcken, die als „Slots" bezeichnet werden. Ein Slot hat eine Größe von 4096 Byte.

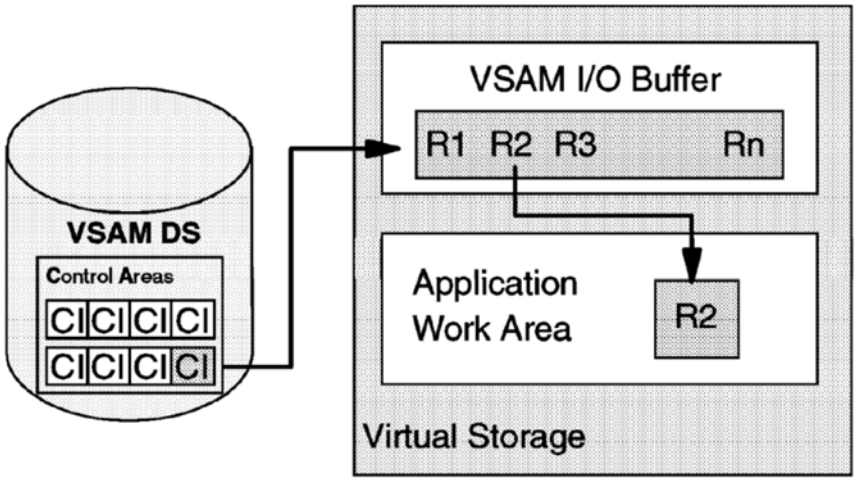


Abb. 6.2.10
Ein VSAM I/O Buffer speichert meistens mehrere Records

Wenn ein Anwendungsprogramm einen VSAM Record liest, wird das ganze Control Interval, das den Datensatz enthält, von dem DASD in einen VSAM I/O Buffer in den virtuellen Speicher des Anwendungsprozesses kopiert. Anschließend wird der gewünschte Record aus dem VSAM I/O-Buffer in einen separaten Buffer im Adressenbereich des Anwendungsprogramms kopiert.

6.2.7 Spanned Records

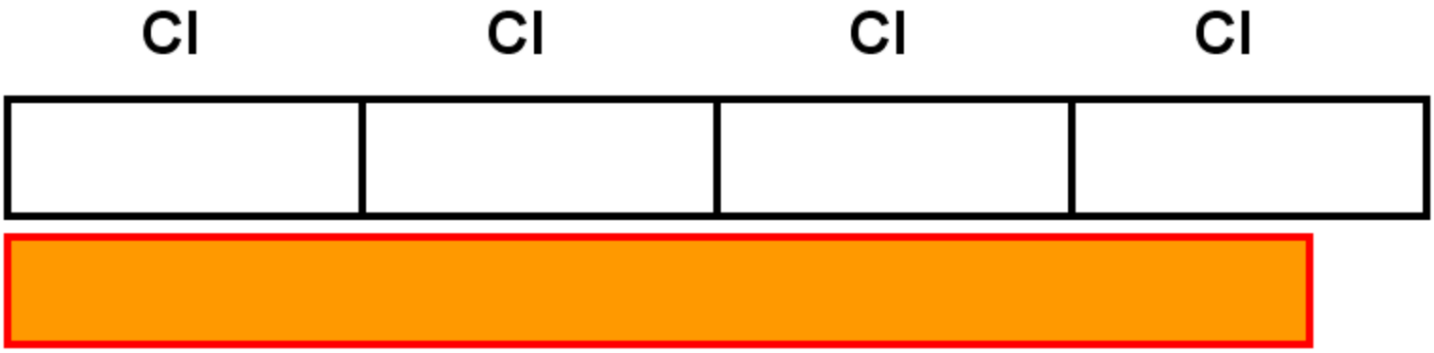


Abb. 6.2.11
Beispiel eines Spanned Records, der sich über 4 Control Intervals erstreckt

Spanned Records sind logische Records, die größer als ein Control Interval sind. Sie werden benötigt, wenn die Anwendung sehr lange logischen Records erfordert. Um mit Spanned Records zu arbeiten, muss der Dataset zum Zeitpunkt der Erstellung mit dem Spanned Records Attribut definiert werden. Spanned Records dürfen Control Interval Grenzen überschreiten. Die RDFs beschreiben, ob der Record spanned ist oder nicht.

Ein Spanned Record beginnt immer auf einer Control Intervalgrenze und füllt eine oder mehrere Control Intervalle innerhalb einer einzigen Control Area. Ein Control Interval, welches einen Teil eines Spanned Records beherbergt, enthält keine weiteren Records. In anderen Worten, der freie Platz am Ende des letzten Segments wird nicht mit dem nächsten Record gefüllt. Dieser Freiraum wird nur verwendet, um den Spanned Rekord zu verlängern.

Ein “Spanned Record” erstreckt sich über mehrere Control Intervalle, kann aber nicht größer als eine Control Area sein.

6.2.8 z/OS Catalog

Unter Windows kann eine Datei wahlweise in der Partition C:, D:, E: bis Z: gespeichert werden. Der Benutzer muss wissen, wo die Datei gespeichert ist, wenn er darauf zugreifen will. Auch können 2 verschiedene Dateien unter dem gleichen Namen in zwei verschiedenen Partitionen gespeichert werden.

Unter z/OS hat jeder Dataset einen eindeutigen Namen, meistens beginnend mit der Benutzer-ID. Zum Beispiel könnte der Benutzer prak123 einen Record in einem Dataset mit dem Namen PRAK123.TEST01.COBOL speichern.

Eine z/OS-Struktur, der „Katalog", ist ein systemweites Verzeichnis, in dem der Speicherort aller Datasets (VSAM, non-VSAM, PDSE) enthalten ist. Er enthält Informationen, auf welchem Plattenlaufwerk der Dataset PRAK123.TEST01.COBOL gespeichert ist. Bedenken Sie, eine z/OS-Installation kann viele Tausende von Festplatten enthalten. Unterschiedliche Arten von Datasets (VSAM, non-VSAM, PDS) können katalogisiert werden. Werden auf einem Windows Rechner unterschiedliche File Systeme (z.B. NTFS, FAT32) in unterschiedlichen Regions eingesetzt, ist dies nicht ohne weiteres möglich.

In der Praxis werden z/OS Datasets fast immer katalogisiert.

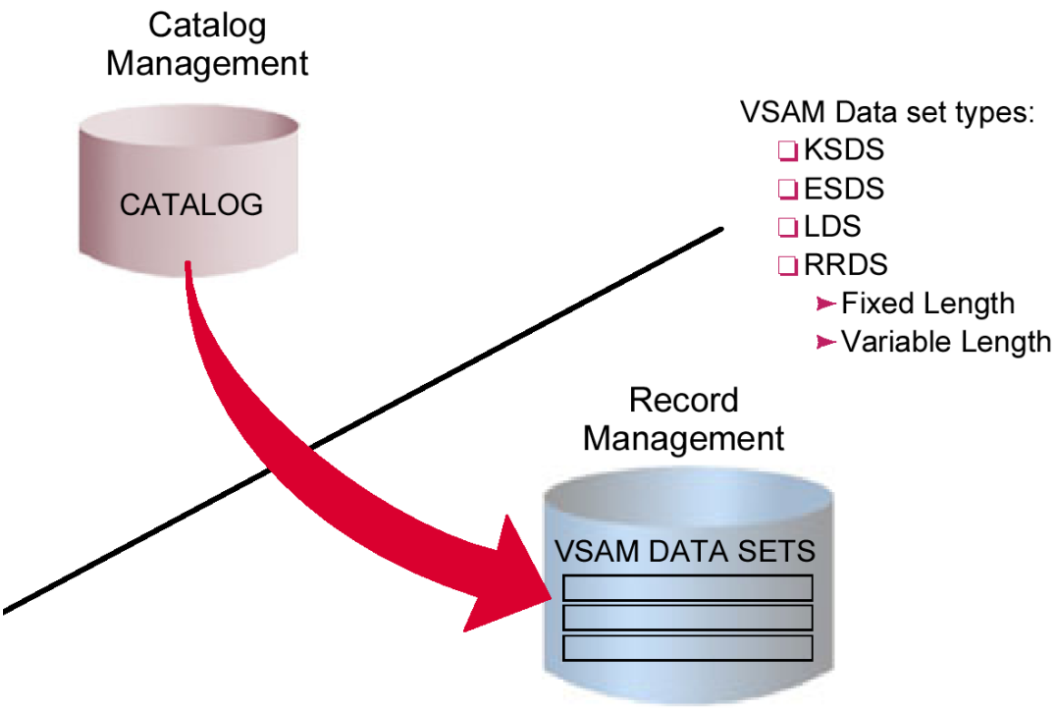


Abb. 6.2.12
Der z/OS Katalog verwaltet auch non-VSAM und PDS Data Sets

Die VSAM Access Method definiert, wie VSAM Datasets in Katalogen organisiert und verwaltet werden. Darüber hinaus legt sie fest, wie Records in einer VSAM Dataset organisiert sind. Damit hat die VSAM Access Method zwei Hauptfunktionen:

- **Das Record Management verwaltet die Records eines VSAM Datasets**
- **Das Catalog Management enthält Funktionen vergleichbar mit einem Unix File Directory, optimiert für die Verwaltung von einer wesentlich größeren Anzahl von Datasets.**

Für Details siehe IBM Redbook: "VSAM Demystified". September 2003, SG24-6105-01
http://www.redbooks.ibm.com/redbooks/pdfs/sg246105.pdf

6.2.9 IDCAMS

Ein VSAM Data Set hat eine komplexe Struktur, und das Anlegen eines neuen Data Sets ist ebenfalls kompliziert.

Hierfür existiert eine Utility unter dem Namen IDCAMS.

IDCAMS ist ein Dienstprogramm unter z/OS zum Anlegen und Verwalten von VSAM Dateien. Mit IDCAMS können Systemspezialisten auch Kataloge administrieren.

Der Name setzt sich, wie bei IBM-Programmen üblich, aus einem Drei-Zeichen-Präfix IDC und der Abkürzung AMS (für Access Method Services) zusammen.

IDCAMS kann mittels JCL in einem Batchjob ausgeführt werden. Eine weitere Möglichkeit ist der Aufruf aus einem Anwenderprogramm. Hier ist es möglich, den Standardinput (SYSIN) und Standardoutput (SYSPRINT) statt mit Dateien mit eigenen Unterprogrammen zu behandeln.

6.3 ESDS und KSDS

6.3.1 Entry Sequenced Dataset

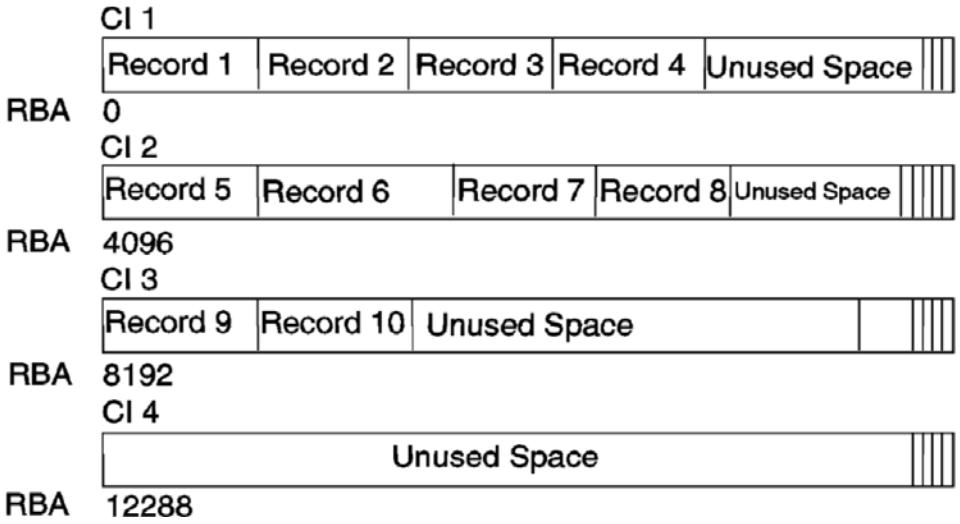


Abb. 6.3.1
Record Anordnung in einem ESDS

Ein Entry Sequenced Dataset (ESDS) ist mit einem sequentiellen Dataset vergleichbar. Er enthält Records mit fester oder variabler Länge. Records werden in der Reihenfolge ihres Eintreffens in dem Dataset abgespeichert, nicht auf Grund des Wertes in einem Schlüsselfeld. Alle neuen Datensätze werden am Ende des Datasets gespeichert.

Dargestellt ist ein ESDS mit 4 Control Intervallen (CI 1, CI 2, CI 3, CI 4), die jeweils 4096 Bytes lang sind. Der Dataset enthält 10 Datensätze und einigen ungenutzte freien Speicherplatz. Der Offset jedes CI vom Anfang des Datasets wird als relative Byte-Adresse (RBA) bezeichnet. Die 4 CIs beginnen bei den relativen Byte-Adressen (RBA) 0, 4096, 8192 und 12228, gezählt vom Anfang des Datasets.

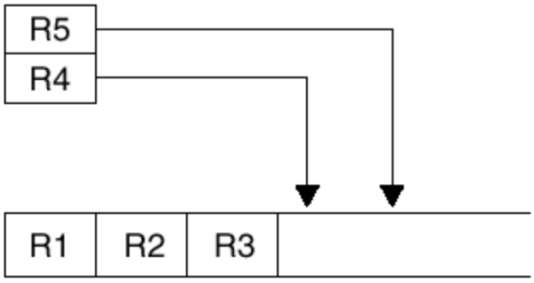


Abb. 6.3.2 ESDS Verarbeitung
Einfügen von neuen Records

Records werden in der Reihenfolge ihres Eintreffens in dem Dataset abgespeichert, nicht auf Grund des Wertes in einem Schlüsselfeld.

Neue Records werden im Anschluss an die bisher gespeicherten Records abgespeichert. Vorhandene Records können nicht gelöscht werden. Wenn Sie einen Record löschen möchten, müssen Sie diesen Record als inaktiv kennzeichnen. Soweit es VSAM betrifft, wird der Record nicht gelöscht. Records können aktualisiert werden, aber sie können nicht verlängert werden. Um die Länge eines Records in einem ESDS zu vergrößern, müssen Sie ihn am Ende des Datasets als neuer Record speichern.

Für einen ESDS werden zwei Arten von READ Verarbeitung unterstützt:

- **Sequentieller Zugriff (die häufigste).**
- **Direkter (oder zufälliger) Zugang. Dies erfordert, dass das Anwendungsprogramm die Relative Byte-Adresse (RBA) des Records benutzt. Die RBA muss irgendwie dem Programm bekannt sein.**

Typische sequentielle Lesevorgänge in einem Anwendungsprogramm benutzen Befehle wie GET NEXT (Dataset Name, ...). Dies lädt denjenigen Record aus dem Bufferpool, der dem zuletzt abgerufenen Record folgt.

Abb. 6.3.3
Beispiel von RBAs eines Entry-Sequenced Datasets (X’62’ = Decimal 98)

Zusätzlich zum sequentiellen Zugriff erlaubt ein ESDS einen direkten (random) Zugriff. Hierzu muss das Anwendungsprogramm die relative Byte-Adresse (RBA) des gewünschten Records angegeben.

Wenn ein Record geladen oder hinzugefügt wird, zeigt VSAM seine relative Byte-Adresse (RBA) an. Der RBA ist ein Zeiger, der das Offset (displacement) in Bytes dieses Records vom Anfang des Datasets enthält. Der erste Record in einem Dataset hat den RBA Wert 0. Der Wert des RBA für den zweiten (und nachfolgenden Records) hängt von der Länge der bisherigen Records ab.

Der RBA eines logischen Records hängt nur von der Position des Records innerhalb des Datasets ab. Der RBA wird immer als Fullword binäre ganze Zahl ausgedrückt.

6.3.2 Key Sequenced Dataset

Jeder Key Sequenced Dataset (KSDS) Record besteht aus mehreren Feldern. Eines dieser Felder kann als Schlüsselfeld (Key Field) deklariert werden.

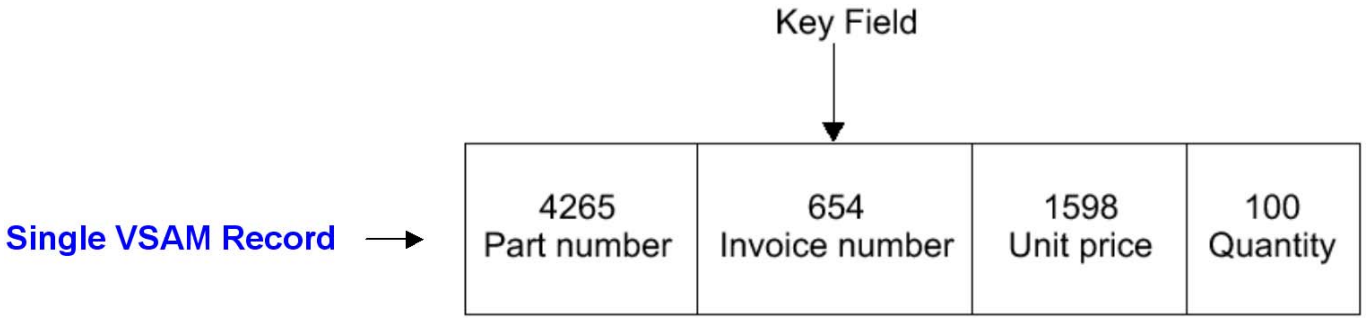


Abb. 6.3.4
Schlüssel Feld (Key Field)

In Abb. 6.3.4 besteht jeder VSAM Record aus 4 Feldern. Die Rechnungsnummer wird als Schlüssel-Feld verwendet. Für den gezeigten Record ist der Wert des Schlüsselfeldes 654.

Das Schlüsselfeld muss sich in der gleichen Position in jedem Record eines Key Sequenced Datasets befinden (das zweite Feld in der Abbildung). Der Offset des Schlüsselfeldes vom Anfang des Records und die Schlüssellänge sind benutzerdefiniert. Der Schlüssel jedes Records muss eindeutig sein. Nachdem ein Record in einen VSAM Dataset geladen wurde kann der Wert des Schlüssels nicht mehr verändert werden. Der gesamte Record muss gelöscht und neu geschrieben werden.

In einem Key Sequenced Dataset werden logische Datensätze in aufsteigender Schlüsselreihenfolge entsprechend des Wertes in dem Schlüsselfeld platziert. Der Schlüssel enthält einen eindeutigen Wert, wie z.B. eine Personalnummer oder Rechnungsnummer. Dies bestimmt die Sortierreihenfolge (Collating Sequence) der Records in dem KSDS Dataset. Für einen bestimmten KSDS wird die Schlüssellänge auf einen konstanten Wert im Bereich von 1 bis 255 Bytes festgelegt. VSAM unterstützt keine Schlüssel mit variabler Länge innerhalb eines einzigen KSDS. Das Schlüsselfeld wird als ein binärer Wert behandelt.

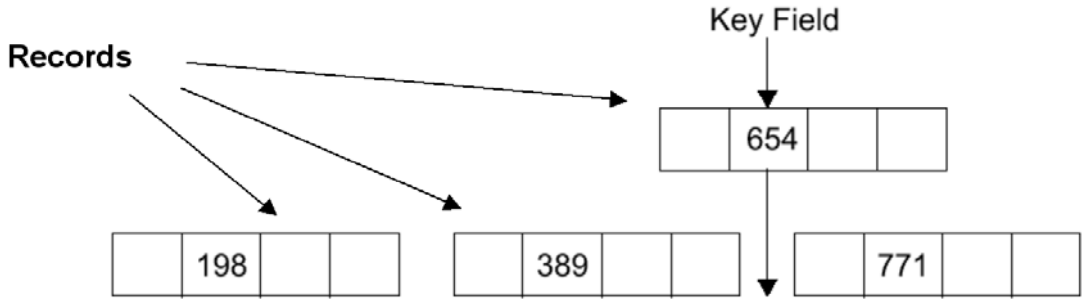


Abb. 6.3.5
Einfügen eines neuen Records

Wenn Records aktualisiert werden, ihre Länge verändert wird, neue Records eingefügt werden, Records gelöscht werden, werden alle Records mit höheren Schlüssel-Werten innerhalb des Control Intervals verschoben.

In dem gezeigten Beispiel enthält das VSAM KSDS CI drei Records mit den Schlüssel Werten 198, 389 und 771. Ein neuer Record mit dem Schlüssel Wert 654 wird nach Record 389 und vor Record 771 eingeordnet. Das Schlüsselfeld der Records bestimmt die Reihenfolge, in der die Records gespeichert werden.

In einem KSDS können Records sowohl sequentiell als auch direkt (indexed) abgearbeitet werden, entsprechend ihren Schlüsselwerten. Die Vorteile des KSDS sind:

- **Sequentielle Verarbeitung ist zum Abrufen von Datensätzen in der sortierten Reihenfolge nützlich**
- **Die direkte Verarbeitung von Records ist bei on-line Anwendungen nützlich.**

6.3.3 Collating Sequence

Der Begriff Collating Sequence bezieht sich auf die Reihenfolge, in der Zeichenfolgen platziert werden, wenn sie sortiert werden sollen.

Ein typisches Beispiel ist die bekannte "alphabetische Reihenfolge", in der "Alfred" vor "Zeus" auftritt, weil „A" vor "Z" im Alphabet erscheint. In einem Computer-System treten jedoch zusätzliche Reihenfolge Probleme auf:

- **Groß-und Kleinschreibung: ein Großbuchstabe "A" und ein kleines "a" werden in der Regel als der gleiche Buchstabe betrachtet. So erscheint Ab zwischen AA und AC. Aber es kann sein, dass Sie die Records anders sortieren möchten.**
- **Nationale Sonderzeichen, Akzente, Tilden: Verschiedene Sprachen verwenden diese Marken über und um Buchstaben herum, aber ein Sprecher mag diese Buchstaben gleich behandeln.**

In einem Computersystem, wird zwangsläufig jedem Buchstaben einen einzigartiger numerischer Code (wie in ASCII, EBCDIC oder Unicode Zeichensätzen) zugewiesen. Die ordnungsgemäße und übliche Anordnung von Zeichenketten entspricht nicht einen einfachen numerischen Vergleich dieser Zeichensatz Codes. Vielmehr wird die Reihenfolge anhand der Sortierreihenfolge (Collating Sequence) bestimmt.

In der deutschen Sprache werden Umlaute (Ä, Ö, Ü) in der Regel ebenso wie ihre nicht-Umlaut Versionen behandelt. Der Buchstabe ß wird immer als ss sortiert. Dies ergibt die alphabetische Reihenfolge Arg, ärgerlich, Arm, Assistent, Aßlar, Assoziation. Für Telefon-Verzeichnisse und ähnliche Listen von Namen werden die Umlaute wie die Buchstaben-Kombinationen "ae", "oe", "ue" behandelt. Dies ergibt die alphabetische Reihenfolge Udet, Übelacker, Uell, Ülle, Ueve, Uffenbach.

Die Sortierfolge in einer EBCDIC Umgebung ist nicht das Gleiche wie die Sortierfolge in einen ASCII-Umgebung. VSAM-Schlüssel werden nach der EBCDIC Sortierfolge sortiert.

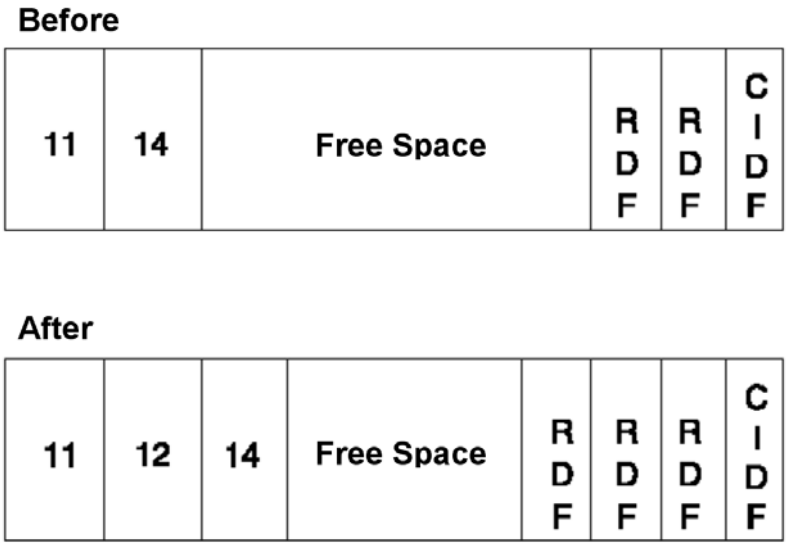


Abb. 6.3.6
Einfügen eines neuen Records

In der Abb. 6.3.6 sind zwei logische Records in dem oberen Control-Interval (CI) gespeichert. Die beiden Records haben die Schlüssel 11 und 14. Das untere Control-Interval zeigt, was passiert, wenn Sie einen Record mit einem Schlüssel von 12 einfügen..

1. **Record 12 wird in seiner korrekten Sortierfolge in dem CI eingesetzt.**
2. **Das CI-Definition Feld (CIDF) wird aktualisiert, um die Reduzierung des verfügbaren freien Speicherplatzes zu registrieren.**
3. **Ein entsprechendes Record Definition Field (RDF) wird an der entsprechenden Stelle eingefügt, um die Länge des neuen Records zu beschreiben.**

Wenn ein Record gelöscht wird, wird der Vorgang umgekehrt durchgeführt. Der Speicherplatz des gelöschten Records und des entsprechenden RDF wird als freier Speicherplatz zurückgewonnen.

6.3.4 VSAM Index

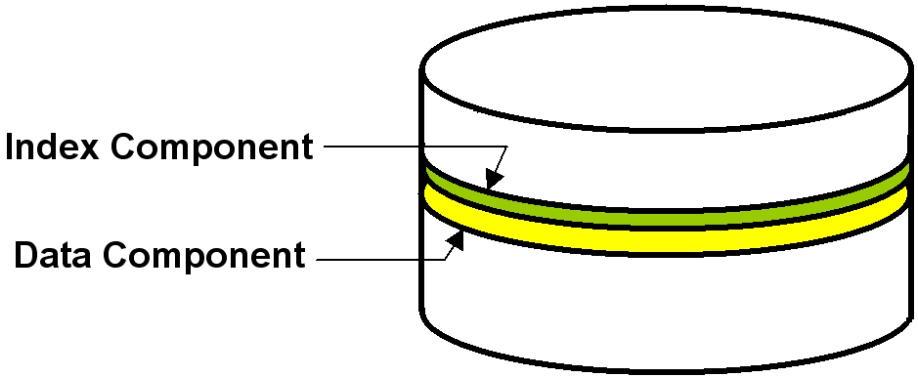


Abb. 6.3.7
VSAM Key Sequenced Dataset

Auf einem Plattenspeicher nimmt ein Key Sequenced Dataset (KSDS) zwei lineare Speicherbereiche ein, sogenannt Komponenten. Es gibt zwei Arten von Komponenten, die Daten-Komponente und die Index-Komponente. Die Daten Komponente ist der Teil einer VSAM Datei, welcher die Daten Records enthält. Alle VSAM Datasets haben eine Daten-Komponente. Andere VSAM Organisationen, zum Beispiel Entry Sequenced Datasets (ESDS), haben nur die Daten-Komponente.

- **Die Data Component ist der Teil des VSAM Datasets der die (Daten) Records enthält. Alle VSAM Datasets beinhalten eine Datenkomponente.**
- **Die Index Component ist eine Sammlung von Records (Index logical Records), welche**
  - **Data Keys der Records der Datenkomponente enthalten, sowie deren**
  - **Adressen (RBA, Relative Byte Address).**

Unter Verwendung des Indexes (Index Component) ist VSAM in der Lage, einen logischen Datensatz aus der Datenkomponente abzurufen, wenn eine Anforderung nach einem Datensatz mit einem bestimmten Schlüssel erfolgt.

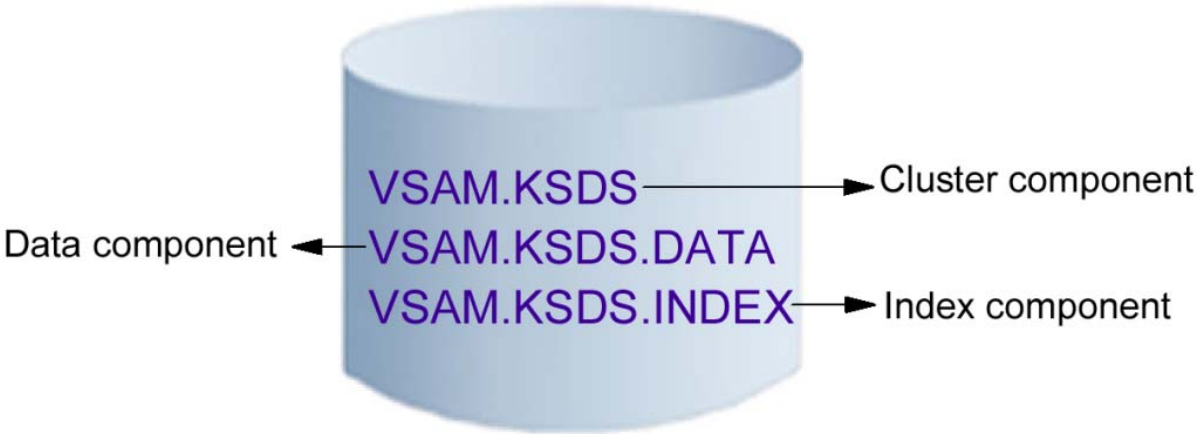


Abb. 6.3.8
Rolle der Cluster Component

Die Datenkomponente und die Index-Komponente sind wirklich zwei unabhängige VSAM Datasets.

Ein VSAM-Cluster ist die Kombination der Daten Komponente (Dataset) und der Index-Komponente (Dataset) eines KSDS. Das Cluster Konzept vereinfacht die VSAM Verarbeitung. Es bietet eine Möglichkeit, Index und Daten-Komponenten als eine Einheit zu behandeln, und mit einem eigenen Namen zu katalogisieren. Es kann auch jede Komponente einen eigenen Namen haben. Dies ermöglicht es, die Daten Komponente getrennt von der Index-Komponente zu verarbeiten.

Jetzt kommt der entscheidende VSAM Frage: "Was in VSAM entspricht einem z/OS Dataset?" Die beste Antwort ist, es hängt von den Umständen ab. Wenn Sie zum Beispiel den Cluster-Namen in einer DD-Anweisung eines JCL Script benutzen, dann entspricht der Cluster dem Dataset. Wenn Sie sich auf eine Komponente beziehen, dann entspricht diese dem Dataset.

(Die Bedeutung des Begriffs "Cluster", im Kontext mit VSAM, ist nicht identisch mit der Bedeutung des Begriffs "Cluster" in einer parallel Processing Konfiguration.)

In unserer VSAM Übungsaufgabe definieren wir einen VSAM Cluster mit dem Namen PRAKT20.VSAM.STUDENT, der aus einer Data Component PRAKT20.VSAM.STUDENT.DATA sowie einer Index Component PRAKT20.VSAM.STUDENT.INDEX besteht.

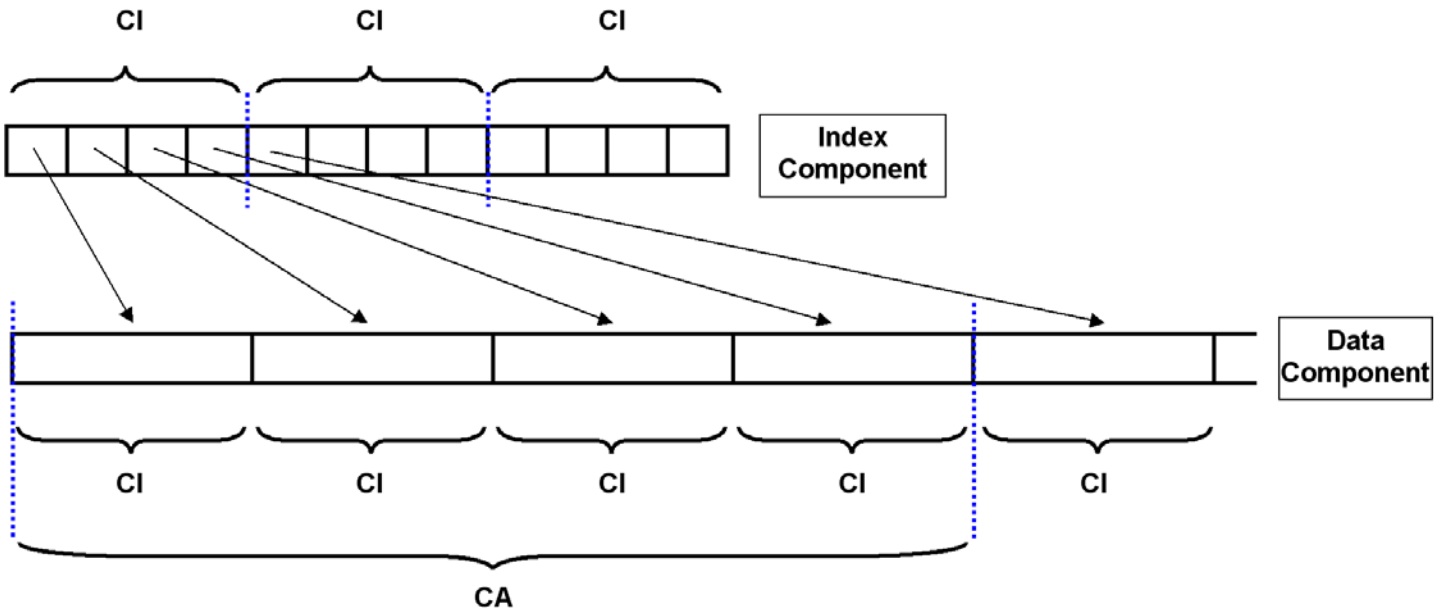


Abb. 6.3.9
Gliederung der Index Component und der Data Component

Die Index-Komponente besteht aus mehreren CIs (und wahrscheinlich auch aus mehreren CAs). Jeder CI in der Index-Komponente besteht aus mehreren Index Records, wobei jeder Index Record in der Index-Komponente auf ein CI in der Daten Komponente zeigt.

In dem in Abb. 6.3.9 gezeigten Beispiel enthält jedes CI in der Index-Komponente 4 Records, und jede CA in der Data Component enthält 4 CIs.

6.3.5 Multilevel Index Component

Ein VSAM Index (Index Component) kann aus einer einzigen Ebene oder aus mehr als einer Ebene bestehen. Jede Ebene enthält Zeiger auf die nächst tiefere Ebene.

Die Suche in einem großen Index kann sehr zeitaufwendig sein. Ein mehrstufiger Index wird aufrechterhalten, um die Index-Suche zu verkürzen. VSAM teilt die Index Komponente in zwei Teile auf: Sequence-Set und Index-Set. Die unterste Ebene der Index Komponente wird als Sequence-Set bezeichnet. Die Pointer in der untersten Ebene zeigen direkt (mit Hilfe einer RBA) auf ein Control-Interval (CI) innerhalb einer Control Area (CA) der Daten Komponente.

Der Sequence Set enthält

- **einen Index-Eintrag für jedes CI in der Daten Komponente, und damit auch**
- **ein Index CI für jedes CA in der Daten-Komponente.**

Bei kleinen VSAM Datasets würde die Index Komponente nur aus einem Sequence Set bestehen. Größere Index Sets unterhalten als Bestandteil ihrer Index Komponente eine oder mehrere Index Ebenen zusätzlich zu dem Sequence Set. Man bezeichnet die Ebenen oberhalb des Sequence Set als den Index Set. Er kann beliebig viele Ebenen enthalten. Sequence-Set und Index Set bilden zusammen die Index-Komponente eines KSDS.

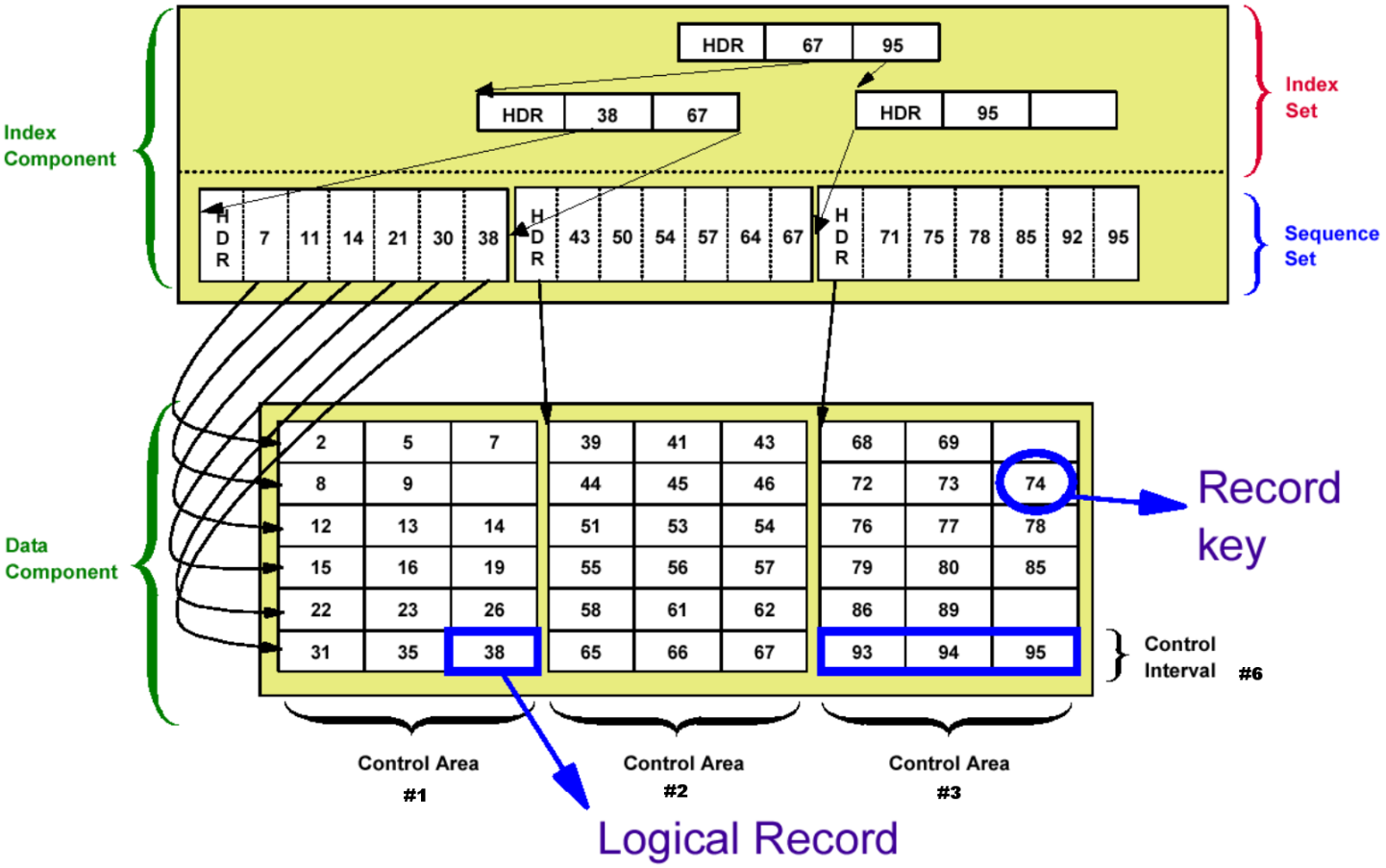


Abb. 6.3.10
Beispiel Index Component und Data Component

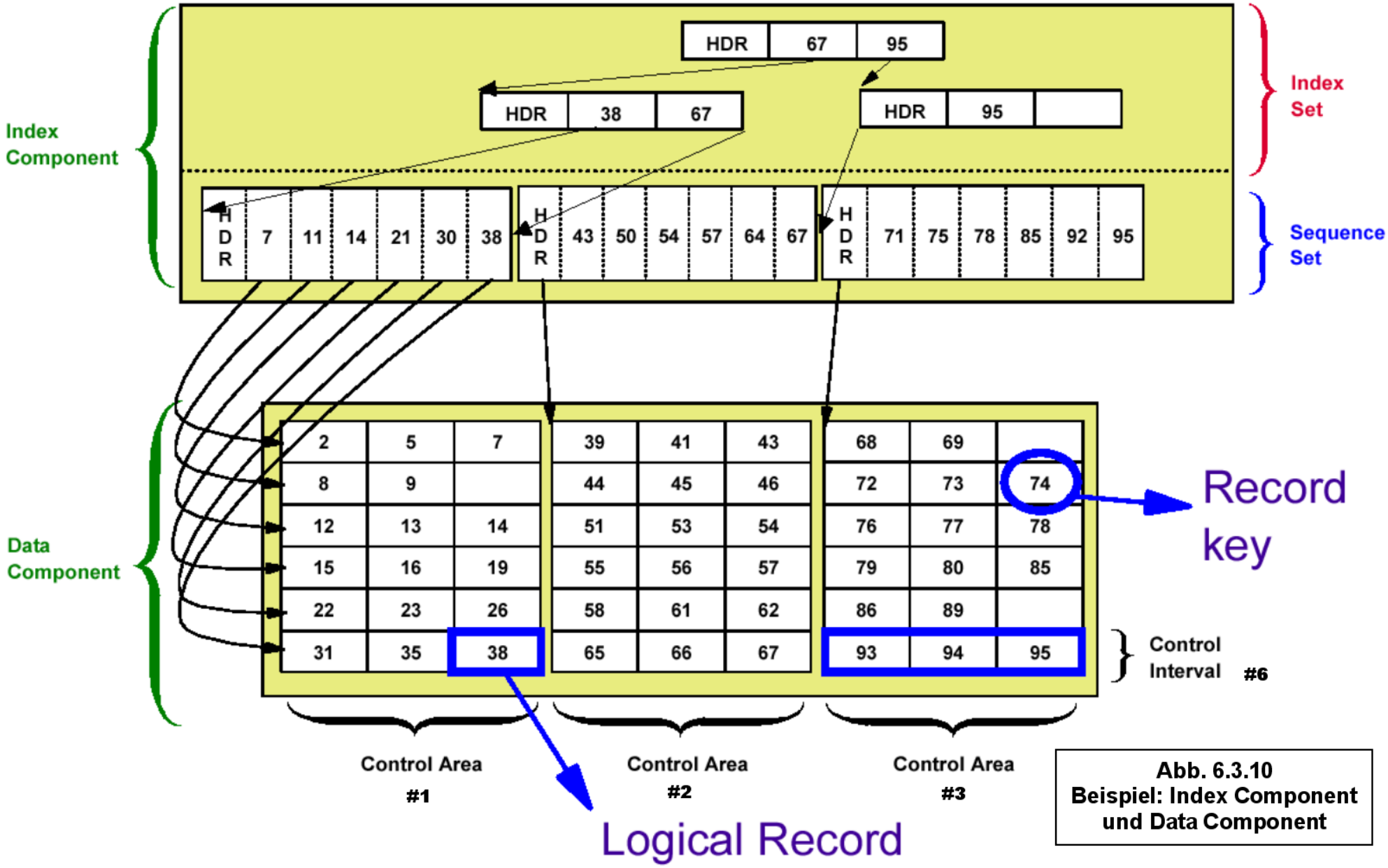


Abb. 6.3.10
Beispiel: Index Component und Data Component

Ein Eintrag in einem Index Set Datensatz enthält den höchstmöglichen Key in einem Index-Datensatz in der nächst tieferen Ebene, und einen Zeiger auf den Anfang dieses Index-Datensatzes. Die höchste Ebene des Index enthält immer einen einzelnen Index CI.

In dem in Abb. 6.3.10 Beispiel gezeigten Beispiel besteht die Daten Komponente eines VSAM Key Sequenced Dataset (KSDS) aus 3 Control Areas. Jede CA besteht aus 6 Kontrollintervallen, und jedes CI speichert genau 3 Records. Es kann nützlich sein, wenn Sie die Seite mit dem Beispiel ausdrucken um den folgenden Text besser zu verstehen.

Der Sequence Set in der Index-Komponente enthält 3 CIs, eine CI für jede CA in der Daten Komponente. Jedes CI in dem Sequence Set enthält 6 Records, einen Datensatz für jedes CI in der Daten-Komponente. Jeder Sequence Set Record enthält den höchsten Key in dem zugehörigen Daten-Komponente CI.

Die erste CI in der Daten-Komponente hat die Schlüssel 2, 5 und 7. Der erste Datensatz in dem Sequence-Set enthält einen Zeiger (RBA) an den Datensatz mit dem Schlüssel 7 in der Daten-Komponente.

Das zweite CI in der Daten-Komponente hat die Keys 8, 9, und einen freien Platz. Das dritte CI in der Daten-Komponente hat die Keys 12, 13 und 14. Wenn der freie Speicherplatz in dem zweiten CI in der Daten Komponente später gefüllt wird, kann ihr Key einen Wert nicht höher als 11 haben (oder es wäre an anderer Stelle platziert werden). Deshalb enthält der zweite Datensatz in dem Sequence Set einen Zeiger (RBA) auf einen potentiellen Datensatz mit dem Schlüssel 11 in der Daten-Komponente.

Für die zweite CA in der Daten-Komponente enthält der Sequence Set eine neue CI. Der erste Datensatz des CI enthält den Wert 43, weil das der Schlüssel des letzten Datensatzes in CI Nr. 1 von CA Nr. 2 in der Daten-Komponente ist.

Das letzte CI in der ersten CA der Datenkomponente hat die Keys 31, 36 und 38. Die Index Ebene unmittelbar über dem Sequence-Set enthält einen Indexeintrag für jedes Sequence Set Control-Interval. Damit enthält der erste Datensatz in der ersten CI des Index Set den Wert 38. Der zweite Datensatz in dem ersten CI des Index Set enthält den Wert 67. Dies ist der höchste Key in CA 2 der Daten-Komponente.

Die unterste Ebene des Index Set enthält 2 CIs, jeweils ein CI für die beiden CAs 1 und 2 der Daten-Komponente, und ein weitere CI für CA 3 der Daten-Komponente. Diese beiden CIs werden durch eine zweite Ebene des Index Sets adressiert.

Alle Einträge werden in aufsteigender Key Reihenfolge gehalten.

6.3.6 Weitere VSAM Dataset Formate

Es existieren zwei weitere VSAM Dataset Formate, die nicht so häufig wie die VSAM Entry Sequenced (ESDS) und VSAM Key Sequenced (KSDS) Dataset Formate eingesetzt werden:

In einem Relative Record Dataset (RRDS) werden die Records in Slots mit einer festen Länge geladen. Eine weitere Version mit einer variablen Slot Länge hat den Namen Variable-Length RRDS (VRRDS). In beiden Fällen werden die Records durch die Relative Record Nummern (RRNs) ihrer Slots adressiert.

Ein Verarbeitungsprogramm benutzt RRNs für einen direkten (random) Zugriff auf die Records.

Ein Linear Dataset (LDS) ist ein VSAM Dataset mit einem Control Interval Größe von 4096 Byte bis 32768 Byte. Ein LDS enthält nur eine zusammenhängende Kette von Datenbytes. Er hat keine Control Informationen in seinem CI, das heißt, keine Record Definition Felder und kein Control Interval Definition Feld (CIDF). Daher sind alle LDS Bytes Datenbytes. Wenn der LDS logischen Records enthält, müssen diese durch das Anwendungsprogramm geblocked und entblocked werden. Records existieren nicht aus der Sicht von VSAM. So weit erfolgt die Verarbeitung ähnlich wie in einem Unix Hierarchical File System.

Ein LDS wird für spezielle Anwendungen benutzt, die es erfordern, große Datenmengen im Hauptspeicher zu halten.

6.3.7 Zusammenfassung

Ein Control Interval ist die Einheit der Daten, die zwischen Festplattenspeicher und virtuellen Speicher übertragen wird, wenn eine I/O-Anforderung auftritt. Es enthält Records, freien Speicherplatz und Steuerinformationen.

Eine Gruppe von Control Intervals bildet eine Control Area (CA). Eine typische Größe ist ein Zylinder eines IBM Modell 3390 Plattenlaufwerks.

Der Index ist ein Merkmal eines KSDS. Ein Sequence Set ist der Teil des Index, der auf die Control Area und das Control Interval eines Records verweist. Der Index Set ist der andere Teil des Index. Es hat mehrere Ebenen mit Zeigern, die letztendlich auf den Sequence Set zeigen.

Ein Control Interval SPLIT ist die Übertragung von Records von einem CI in ein neues CI, um Platz zu schaffen. Das Ergebnis sind zwei halb leere CIs. Ein Control Interval Split erfordert eine Reihe von I/O Operationen, was CPU Performance kostet. Um dies zu verringern sollte immer ausreichend freier Platz in den CIs vorgesehen werden.

6.4 Weiterführende Information

Das Standard Textbuch für VSAM ist als IBM Redbook verfügbar:
“VSAM Demystified”. September 2003, SG24-6105-01
http://www.cedix.de/VorlesMirror/Band1/VSAM01.pdf

Zahlreiche Unternehmen bieten Software Produkte an, mit denen man auf die VSAM Daten eines z/OS Systems direkt zugreifen kann, z.B. über ein Tablet oder ein iPhone.

Ein Beispiel (von vielen) ist die Firma (http://www.attunity.com/index.aspx). Eines ihrer Produkte wird in dem Video
„Unleash VSAM Data From the Mainframe -- 'Off Platform'!“ vorgestellt:

http://www.youtube.com/watch?v=-qcKFhFzpTg
oder
http://www.youtube.com/watch?v=-qcKFhFzpTg&NR=1&feature=endscreen

Weitere Web Präsentationen unter

www.attunity.com/webinars.aspx?newsId=1540

Ein ausführliches VSAM Tutorial ist unter
http://www.cedix.de/VorlesMirror/Band1/VSAM02.pdf
zu finden

7. Transaktionsverarbeitung

7.1 Einführung

7.1.1 Client/Server-Modell

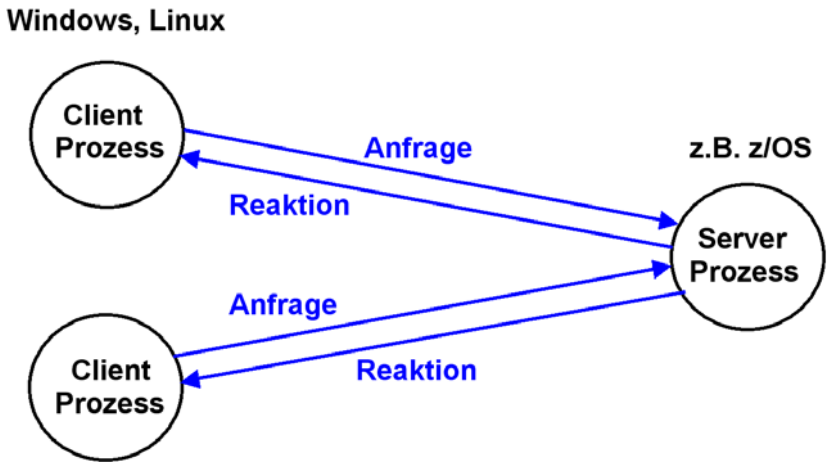


Abb. 7.1.1
Client/Server-Modell

Prozesse auf einem Klienten-Rechner nehmen die Dienstleistungen eines Servers in Anspruch. Ein Server bietet seine Dienste (Service) einer Menge a priori unbekannter Klienten (Clients) an.

Klient:	**Nutzer eines Server Dienstes**
Server:	**Rechner, der Dienst-Software ausführt**
Dienst (Service):	**Software-Instanz, die auf einem oder mehreren Server Rechnern ausgeführt wird**
Interaktionsform:	**Anfrage / Reaktion (Request/Reply)**

Ein Rechner kann gleichzeitig mehrere Serverdienste (Services) anbieten.

Der größere Teil (etwa 60-70 %) aller Anwendungen in Wirtschaft und Verwaltung läuft (interaktiv) auf einer Client/Server Konfiguration. Etwa 30-40 % laufen als Stapelverarbeitungsprozesse (batch processing)

Client/Server Systeme werden auch als „Interaktive“ Systeme, im Gegensatz zur Stapelverarbeitung, bezeichnet.

Die meisten interaktiven Anwendungen und viele Stapelverarbeitungsanwendungen werden als „Transaktionen“ ausgeführt.

7.1.2 Datenbanken

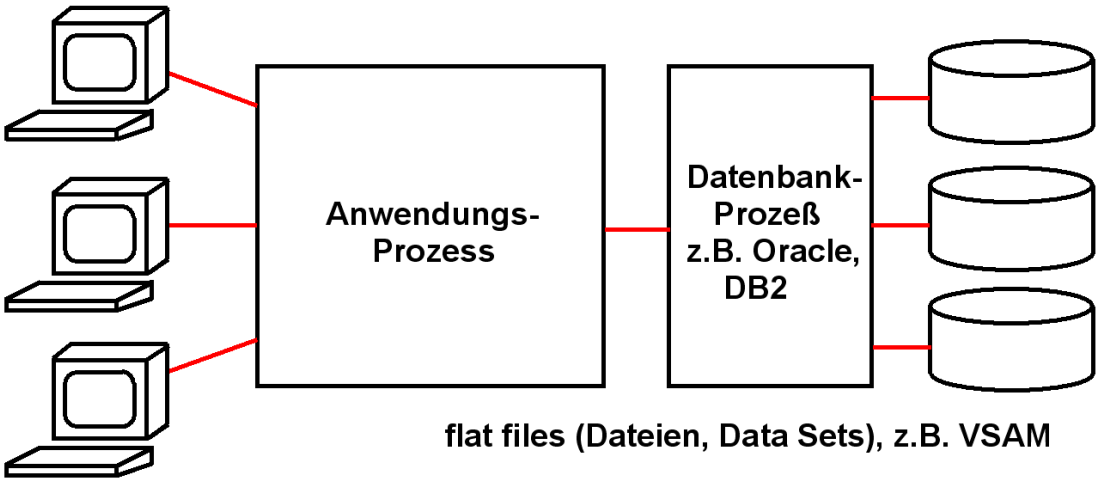
Für einfache Interaktionen eines Anwendungsprogramms mit seinen Daten werden „Flat Files“ (z.B. VSAM) eingesetzt. Unter z/OS verringern Access Methods den Programmieraufwand für den Zugriff. Access Methods werden größtenteils vom Betriebssystem Kernel ausgeführt, aber VSAM ist eng in den Benutzer- (Anwendungs-) Prozess integriert.

Komplexere Interaktionen unterstützen ein Anwendungsprogramm mit Hilfe eines getrennten „Datenbank“-Prozesses.

Eine Datenbank besteht aus einer Datenbasis (normalerweise Plattenspeicher) und Verwaltungsprogrammen (Datenbank Software), welche die Daten entsprechend den vorgegebenen Beschreibungen abspeichern, auffinden oder weitere Operationen mit den Daten durchführen.

Wenn wir im Zusammenhang mit Client/Server Systemen von einer Datenbank sprechen meinen wir in der Regel damit die Verwaltungsprogramme (Datenbank im engeren Sinne).

Ein Datenbankprozess läuft fast immer in einem eigenen virtuellen Adressenraum. Häufig sind es sogar mehrere virtuelle Adressenräume (z.B. drei im Fall von z/OS DB2).



PC
Registrierkasse
Geldausgabe-
automat

flat files (Dateien, Data Sets), z.B. VSAM

nicht relationale Datenbanken, z.B.
ADABAS
IMS

relationale Datenbanken, z.B.
DB2
Oracle
MS SQL
mysql

Abb. 7.1.2
Datenbank Server in einer typische Client/Server Anwendung

Es existieren unterschiedliche Arten von Datenbanken. Die drei wichtigsten Grundtypen sind:

Relationale Datenbanken, z.B.:

- **Oracle**
- **DB2 – wichtigste Datenbank unter z/OS**
- **Mircrosoft SQL**

Nicht-relationale Datenbanken, unter z/OS häufig eingesetzt:

- **IMS**
- **ADABAS**

Objekt orientierte Datenbanken (wenig Bedeutung unter z/OS), z.B.

- **Poet**

Auf einem Mainframe dominieren die DB2, IMS und ADABAS Datenbanksysteme.

Der Server Teil einer Client/Server Anwendung besteht aus zwei Teilen: einem Anwendungsprozess und einem Datenbankprozess.

Dargestellt in Abb. 7.1.2 ist eine logische Struktur. 2-Tier, 3-Tier oder n-Tier Konfigurationen unterscheiden sich dadurch, wie diese Funktionen auf physische Server abgebildet werden.

Einige der Alternativen sind:

In einer 2-Tier Konfiguration befindet sich der Anwendungsprozess auf dem gleichen Rechner wie der Klient.

In einer 3-Tier Konfiguration befinden sich Anwendung und Datenbank als getrennte Prozesse auf getrennten Rechnern.

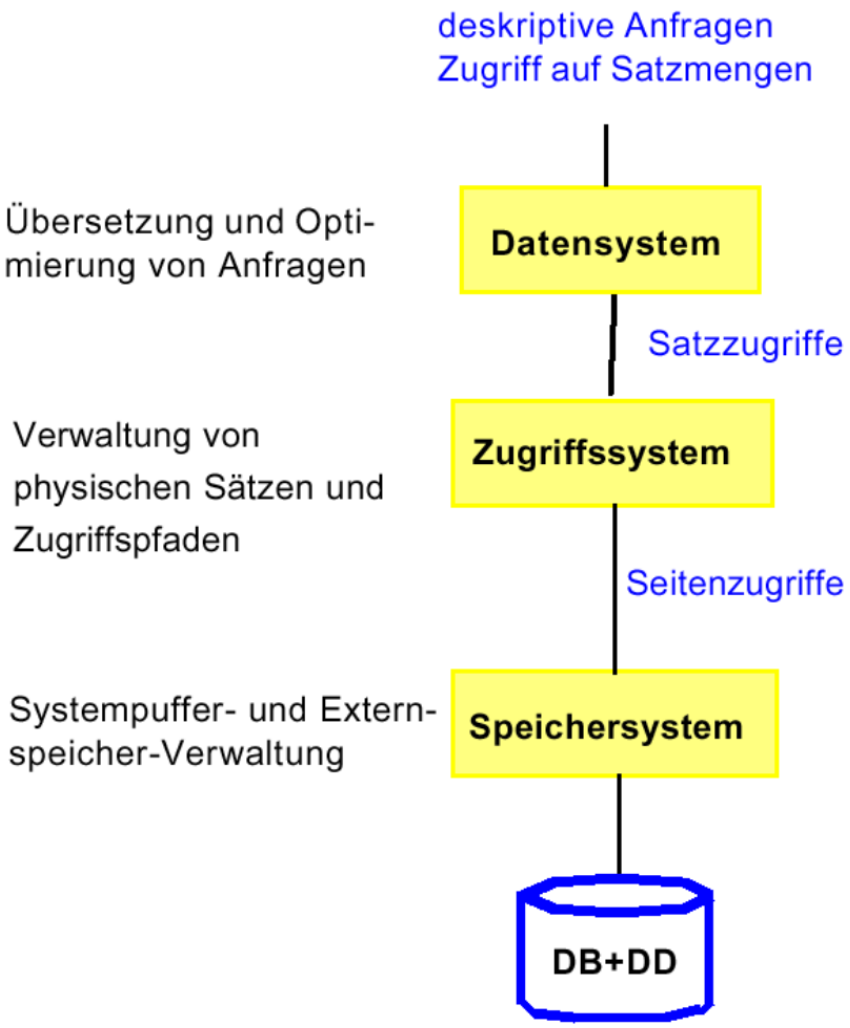


Abb. 7.1.3
Drei Basiskomponenten eines Datenbanksystems

Ein Datenbanksystem ist ein eigenständiger Systemprozess, der in einem eigenen virtuellen Adressenraum (Region) läuft (bei z/OS DB2 sind es 3 Regions).

Das Kernelement ist das Zugriffssystem. Es bildet die auf dem Plattenspeicher befindlichen ungeordneten Daten in der Form von Tabellen (z.B. DB2) oder Hierarchien (z.B. IMS) ab.

Das Speichersystem optimiert die Zugriffe auf die Daten eines Plattenspeichers, z. B. durch die Verwaltung von Ein/Ausgabepuffern (Buffer Pool).

Das Datensystem erlaubt Abfragen und Änderungen der Datenbestände mittels einer benutzerfreundlichen Schnittstelle, z.B. SQL.

Ein wichtiger Bestandteil einer Relationalen Datenbank ist ihr Schema. Das Schema legt fest, welche Daten in der Datenbank gespeichert werden und wie diese Daten in Beziehung zueinander stehen. Den Vorgang zum Erstellen eines Schemas nennt man Datenmodellierung.

Ein Datenbanksystem hat drei Basiskomponenten:

Das Datensystem übersetzt Anfragen von Transaktionsprogrammen, die z.B. in SQL (Structured Query Language) gestellt werden:

```
Select * FROM PERS
WHERE ANR = 'K55'
```

Das Zugriffssystem setzt die Anfrage in logische Seitenreferenzen um.

Das Speichersystem setzt die logische Seitenreferenzen in physische Seitenreferenzen um. Das DB2 Datenbanksystem speichert alle Daten in „Slots“, deren Größe 4096 Bytes, also der Größe eines Seitenrahmens, entspricht.

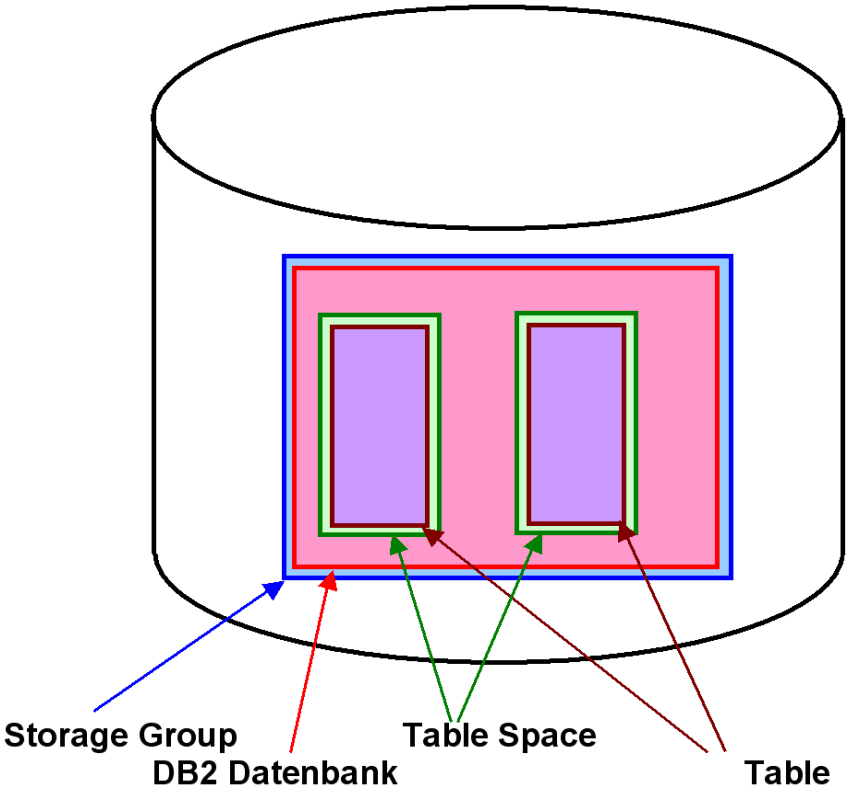


Abb. 7.1.4
Platzzuweisung für eine DB2 Datenbank

Aus der Sicht des Benutzers besteht eine DB2 Datenbank aus mindestens einer, meistens aber mehreren Tabellen.

Auf einem Plattenspeicher wird hierfür Platz in der Form einer Storage Group angelegt. Diese nimmt eine DB2 Datenbank auf.

Innerhalb der Storage Group werden eine oder mehrere Table Spaces angelegt. Jeder Table Space nimmt eine DB2 Table auf.

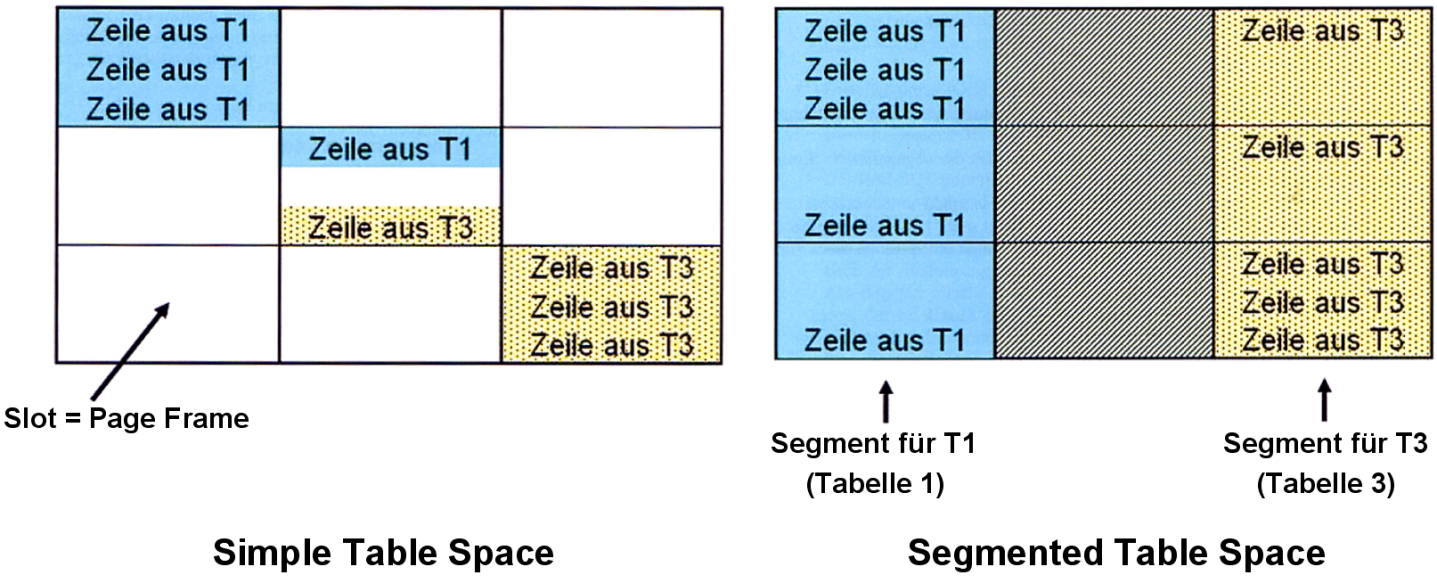


Abb. 7.1.5
Unterschiede in der Datenspeicherung

Ein Table Space besteht aus Slots. Slots sind typischerweise 4096 Bytes groß. Bei einen Zugriff auf die in einer DB2 Tabelle gespeicherten Daten wird ein Slot aus dem Plattenspeicher ausgelesen, und in einen als „Buffer“ bezeichneten 4096 Byte großen Rahmen des realen Hauptspeichers (und die entsprechende Seite des virtuellen Speichers) transportiert.

Ein Table Space kann auch zwei oder mehrere DB2 Tabellen aufnehmen. Bei einem „simple Table Space“ können die Zeilen von zwei Tabellen in dem gleichen Slot untergebracht sein. Der Plattenspeicherbereich von einem „segmented Table Space“ ist in mehrere Segmente aufgeteilt. Ein Segment enthält eine bei der Anlage des Table Spaces angegebene Anzahl von Slots (4, 8, ..64). Jedes Segment enthält Einträge von nur einer Tabelle.

7.1.3 Pufferverwaltung im Datenbanksystem

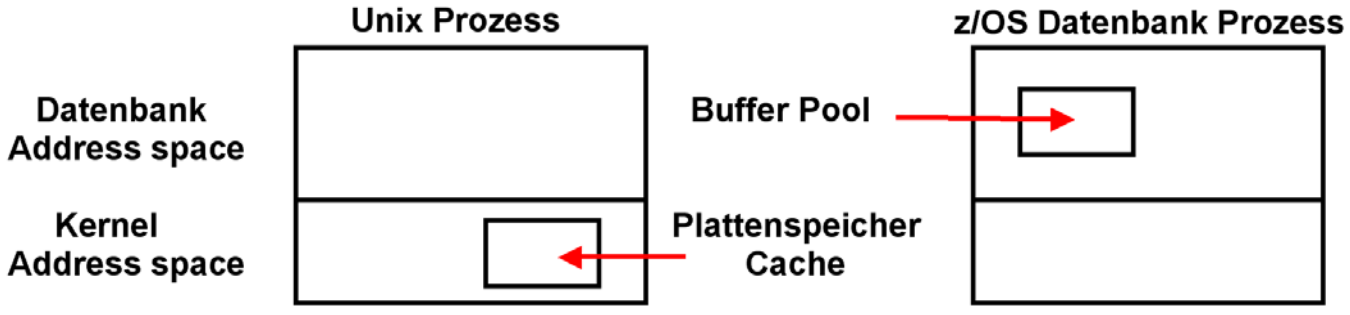


Abb. 7.1.6
Unterschiede zwischen Unix und z/OS

Unix Systeme verbessern den Zugriff auf Daten, indem sie im Hauptspeicher einen Plattenspeicher-Cache für alle Arten von I/O Daten verwalten. Dieses Konzept existiert nicht unter z/OS.

z/OS DB2 und z/OS IMS speichern eine Anzahl von derzeitig benutzten Daten Items in I/O-„Buffern“ im Hauptspeicher. Ein Buffer speichert in der Regel mehrere Records, z.B. ein VSAM Control Intervall oder einen mehrere Zeilen umfassenden Teil einer DB2 Tabelle. Die Menge aller derzeitig angelegten Buffer wird als Bufferpool bezeichnet.

Der Bufferpool befindet sich im Addresspace des Datenbanksystems, welches eine aufwendige Verwaltung seiner I/O Puffer vornimmt. Die Suche im Datenbankpuffer ist in Software implementiert. Typische Referenzmuster in einem Datenbank-System sind:

- **Sequentielle Suche (z.B.: Relationen- Scan)**
- **Hierarchische Pfade (z.B.: Suchen über Bäume)**
- **Zyklische Pfade**

7.1.4 Komponenten der Transaktionsverarbeitung

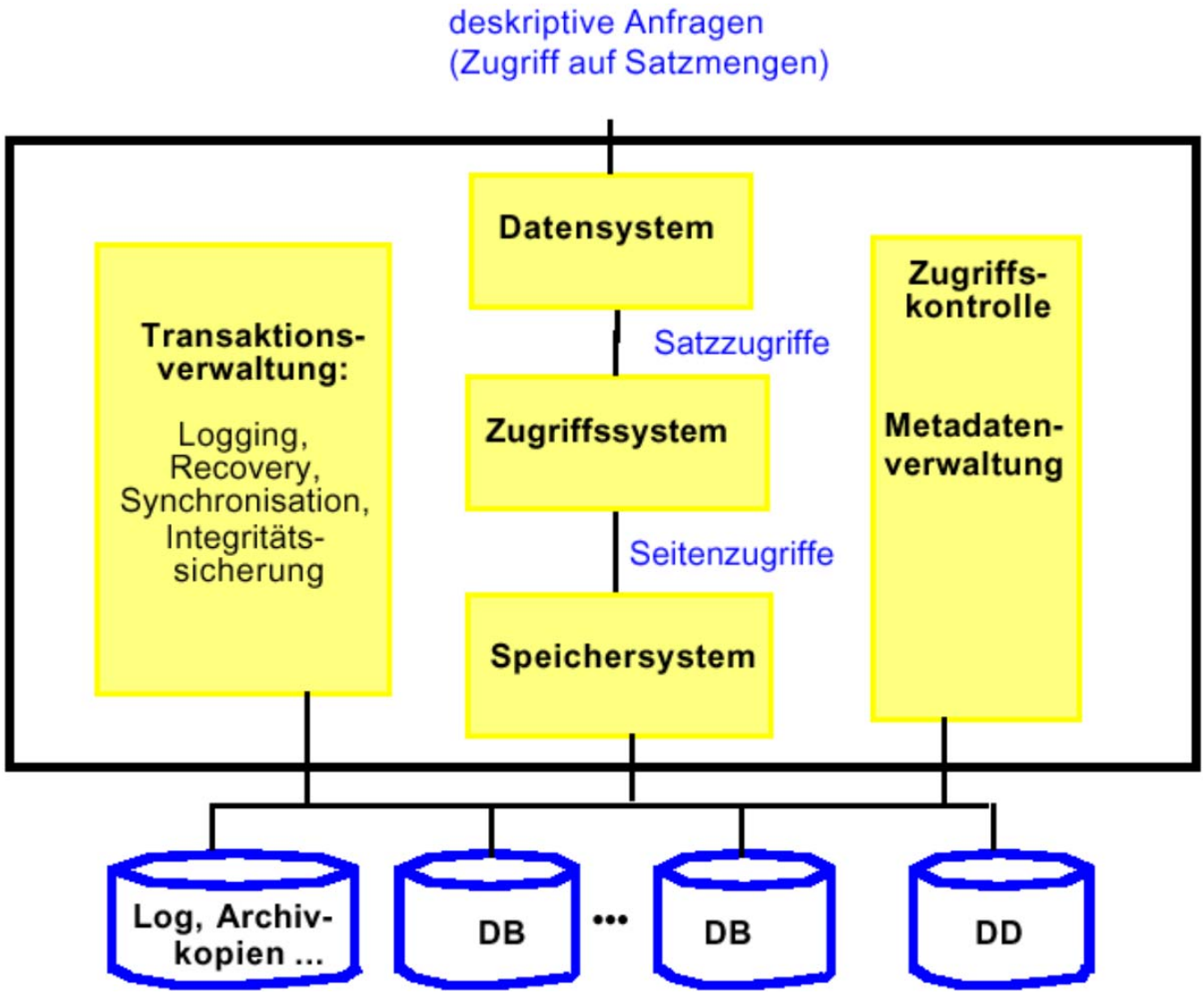


Abb. 7.1.7
Aufbau eines Datenbanksystems

Neben den Komponenten Datensystem, Zugriffssystem und Speichersystem enthält ein Datenbanksystem in der Regel eine Komponente für die Transaktionsverarbeitung, Einrichtungen für die Zugriffskontrolle, Administrationskomponenten sowie Einrichtungen, welche die gespeicherten Daten beschreiben (Metadaten, Data Definition (DD)).

Schließlich sind Einrichtungen für die Verwaltung einer Log-Datei sowie für die Archivierung von Daten vorhanden.

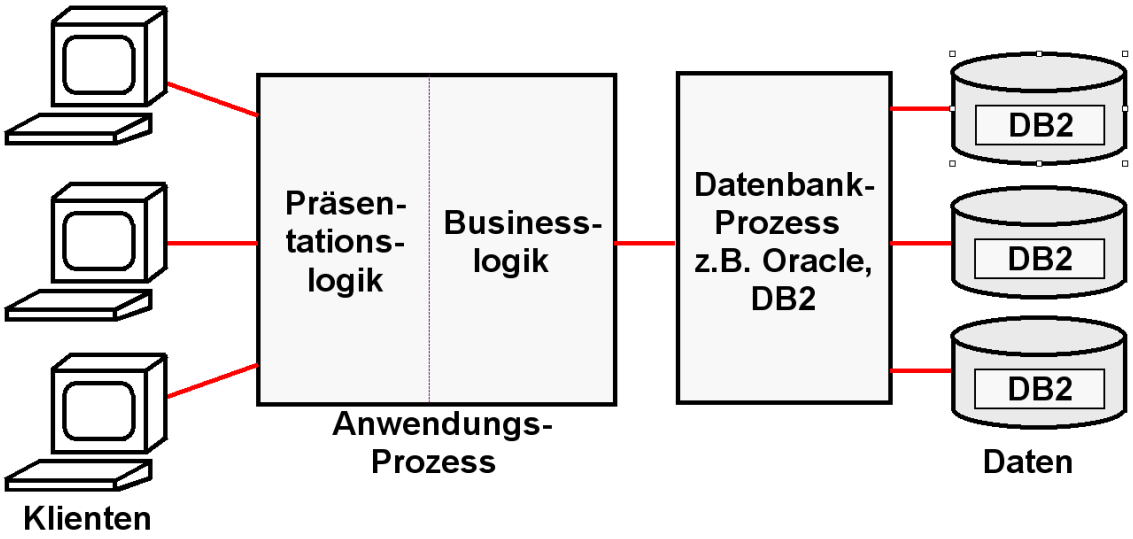


Abb. 7.1.8
Aufteilung des Anwendungsprogramms

Der Anwendungsprozess besteht aus zwei Teilen:

Business Logic (Anwendungslogik) verarbeitet die Eingabedaten des Endbenutzers und erzeugt Ausgabedaten für den Endbenutzer, z.B. in der Form einer wenig strukturierten Zeichenkette (oft als „Unit Record“ bezeichnet).

Präsentationslogik formt die rohen Ausgabedaten in eine für den Endbenutzer gefällige Form um, z.B. in Form einer grafischen Darstellung. Unterschiedliche Klienten können die gleiche Business Logik benutzen um mit unterschiedlicher Präsentationslogik die Ausgabedaten unterschiedlich darzustellen.

7.1.5 Zwei-Tier Client/Server Architektur

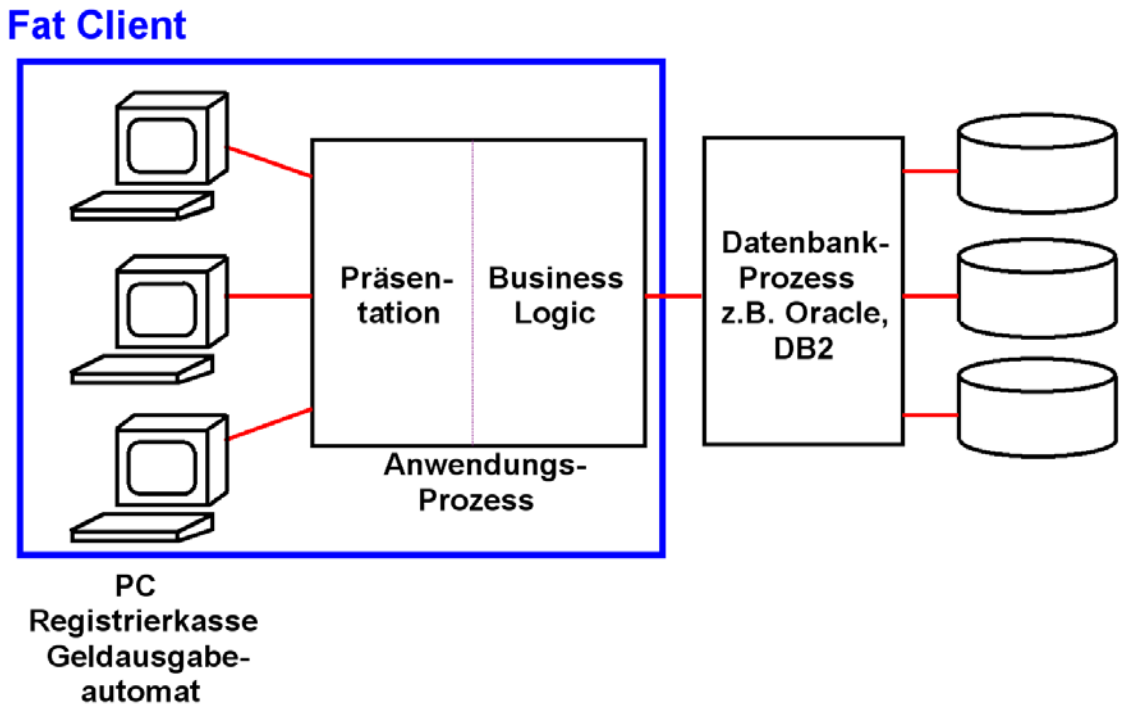


Abb. 7.1.9
Der Anwendungsprozess läuft auf dem Klienten

In einer als „Fat Client“ bezeichneten Konfiguration laufen Business Logic und Präsentationslogik beide auf dem Klienten. Auf dem Server läuft nur der Datenbankprozess.

Dies ist eine häufig in kleinen Betrieben anzutreffende Konfiguration, auch als „2-Tier“ bezeichnet. z.B. teilen sich 12 PC Bildschirm-Arbeitsplätze einer Abteilung einen Datenbankserver.

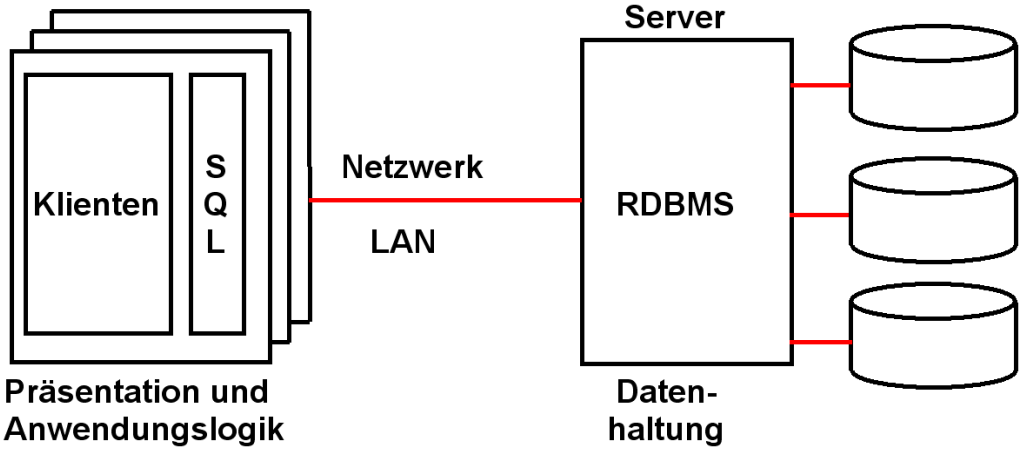


Abb. 7.1.10
SQL Client Komponente

Auf dem Klientenrechner befindet sich eine SQL-Client Komponente, die in der Lage ist, SQL Anfragen über das Netz an den Datenbankserver zu senden.

Typische Umgebungen haben folgende Eigenschaften:

< 200 Klienten
< 100 000 Transaktionen / Tag
LAN Umgebung
1 oder wenige Server
Mäßige Sicherheitsanforderungen

Die Anwendungsentwicklung verwendet häufig Power Builder oder Visual Basic. 2-Tier Konfigurationen werden häufig in reinen Microsoft Umgebungen bei kleinen Unternehmen eingesetzt. Diese Konfiguration skaliert sehr schlecht; deshalb ist sie in größeren Unternehmen nur sehr selten anzutreffen.

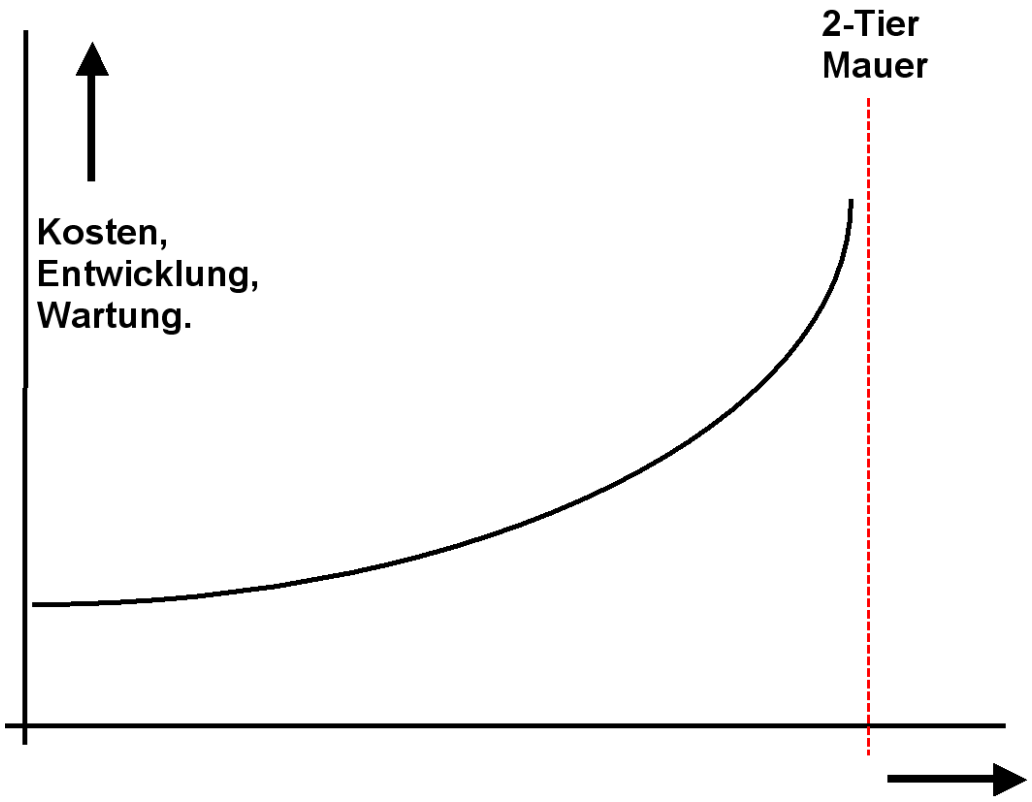


Abb. 7.1.11
Skalierung der 2-Tier Architektur

Die 2-Tier Konfiguration ist sehr populär, leicht zu programmieren, ist aber nur bis zu einer gewissen maximalen Größe möglich (als 2-Tier Mauer bezeichnet). Gründe sind:

- **Datenvolumen auf dem Netzwerk**
- **Datensatz Lock Contention. Was passiert, wenn 2 Klienten gleichzeitig auf den gleichen Datensatz zugreifen ?**
- **Verteilung (Administration) der Klienten Software auf einer Vielzahl von Klienten Rechnern**

7.1.6 Drei-Tier Client/Server Architektur

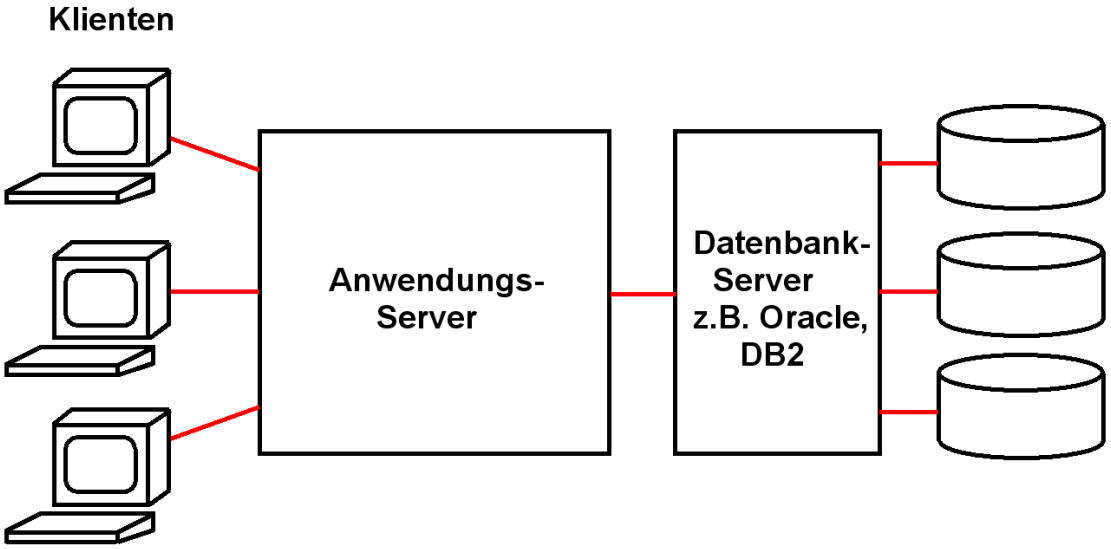


Abb. 7.1.12
Alle Anwendungen laufen auf einem getrennten Anwendungsserver

Bei der 3-Tier Konfiguration existiert ein von den Klienten getrennter Anwendungsserver, wobei entweder zwei getrennte Rechner für Anwendung und Datenbank benutzt werden, oder alternativ der Anwendungs- und der Datenbankprozess auf dem gleichen Rechner laufen. Letzteres ist eine typische z/OS Konfiguration. Für den Anwendungsserver existieren zwei Arten:

- **Präsentationslogik läuft auf dem Klienten. Business Logik läuft auf dem Anwendungsserver. Beispiel: SAPGUI des SAP R/3 Systems**
- **Präsentationslogik und Business Logik laufen auf dem Anwendungsserver Beispiel: Servlets, Java Server Pages und Enterprise Java Beans.**

Die 3-Tier Konfiguration skaliert wesentlich besser als die 2-Tier Konfiguration:

- **Service Anforderungen generieren weniger Datenvolumen**
- **Anwendungsserver optimiert Lock Contention**
- **Mehrfache Anwendungsserver sind möglich (Anwendungsreplikation)**

Die Zugriffskontrolle erfolgt auf Service Basis.

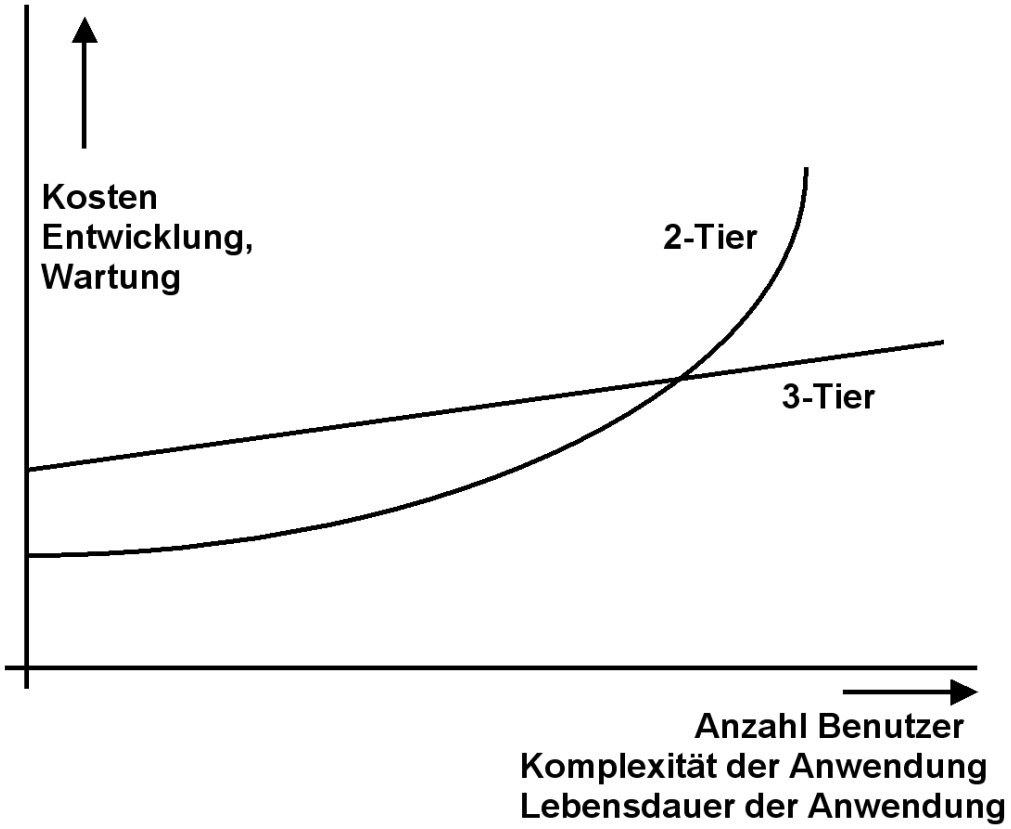


Abb. 7.1.13
Überwindung der 2-Tier Mauer

Bei geringer Belastung ist die 2-Tier Konfiguration überlegen – im Mainframe Bereich ein nicht sehr realistischer Fall.

Bei großer Belastung skaliert die 3-Tier Konfiguration wesentlich besser als die 2-Tier Konfiguration.

7.1.7 Transaktionen

Transaktionen sind Stapelverarbeitungs- oder Client/Server-Anwendungen, welche die auf einem Server gespeicherten Daten von einem definierten Zustand in einen anderen überführen. Eine Transaktion wird wie jede Anwendung als ein Prozess oder Thread ausgeführt.

Eine Transaktion ist eine atomare Operation. Die Transaktion wird entweder ganz oder gar nicht durchgeführt.

Eine Transaktion ist die Zusammenfassung von mehreren Datei- oder Datenbankoperationen, welche entweder

erfolgreich abgeschlossen wird, oder
die Datenbank(en) unverändert lässt

Die Datei/Datenbank bleibt in einem konsistenten Zustand: Entweder der Zustand vor Anfang, oder der Zustand nach erfolgreichem Abschluss der Transaktion

Im Fehlerfall, oder bei einem Systemversagen werden alle in Arbeit befindlichen Transaktionen abgebrochen und alle evtl. bereits stattgefundenen Datenänderungen automatisch rückgängig gemacht. Dieser Vorgang wird als Rollback, Backout oder Abort bezeichnet; die Begriffe sind Synonyme.

Wird eine Transaktion abgebrochen, werden keine Daten abgeändert.

7.1.8 ACID Eigenschaften

Eine Transaktion ist ein Anwendungsprozess, der die ACID Eigenschaften implementiert:

Atomizität (Atomicity)

Eine Transaktion wird entweder vollständig ausgeführt oder überhaupt nicht.

Der Übergang vom Ursprungszustand zum Ergebniszustand erfolgt ohne erkennbare Zwischenzustände, unabhängig von Fehlern oder Crashes. Änderungen betreffen Datenbanken, Messages, Transducer und andere.

Konsistenzerhaltung (Consistency)

Eine Transaktion überführt die transaktionsgeschützten Daten des Systems von einem konsistenten Zustand in einen anderen konsistenten Zustand.

Diese Eigenschaft garantiert, dass die Daten der Datenbank nach Abschluss einer Transaktion schemakonsistent sind, d. h. alle im Datenbankschema spezifizierten Integritätsbedingungen erfüllen.

Daten sind konsistent, wenn sie nur durch eine Transaktion erzeugt oder abgeändert werden.

Isolation

Die Auswirkungen einer Transaktion werden erst nach ihrer erfolgreichen Beendigung für andere Transaktionen sichtbar.

Single User Mode Modell. Selbst wenn 2 Transaktionen gleichzeitig ausgeführt werden, wird der Schein einer seriellen Verarbeitung gewahrt.

Dauerhaftigkeit (Durability)

Die Auswirkungen einer erfolgreich beendeten Transaktion gehen nicht verloren.

Das Ergebnis einer Transaktion ist real, mit allen Konsequenzen. Es kann nur mit einer neuen Transaktion rückgängig gemacht werden. Die Zustandsänderung überlebt nachfolgende Fehler oder Systemcrashes.

Bei Einhaltung der ACID Eigenschaften wird angenommen, dass

- **alle Daten auf Plattenspeichern (oder Solid State Disks) mit RAID oder vergleichbaren Eigenschaften, oft mit zusätzlicher Spiegelung, gespeichert werden und**
- **alle Fehlerfälle wie Plattenspeichercrashes oder andere katastrophale Ausfälle unbeschadet überstehen.**

Als Persistenz bezeichnet man eine Datenspeicherung, welche die Durability Bedingung der ACID Eigenschaften erfüllt.

7.1.9 Beispiel für eine Transaktionsverarbeitungsanwendung

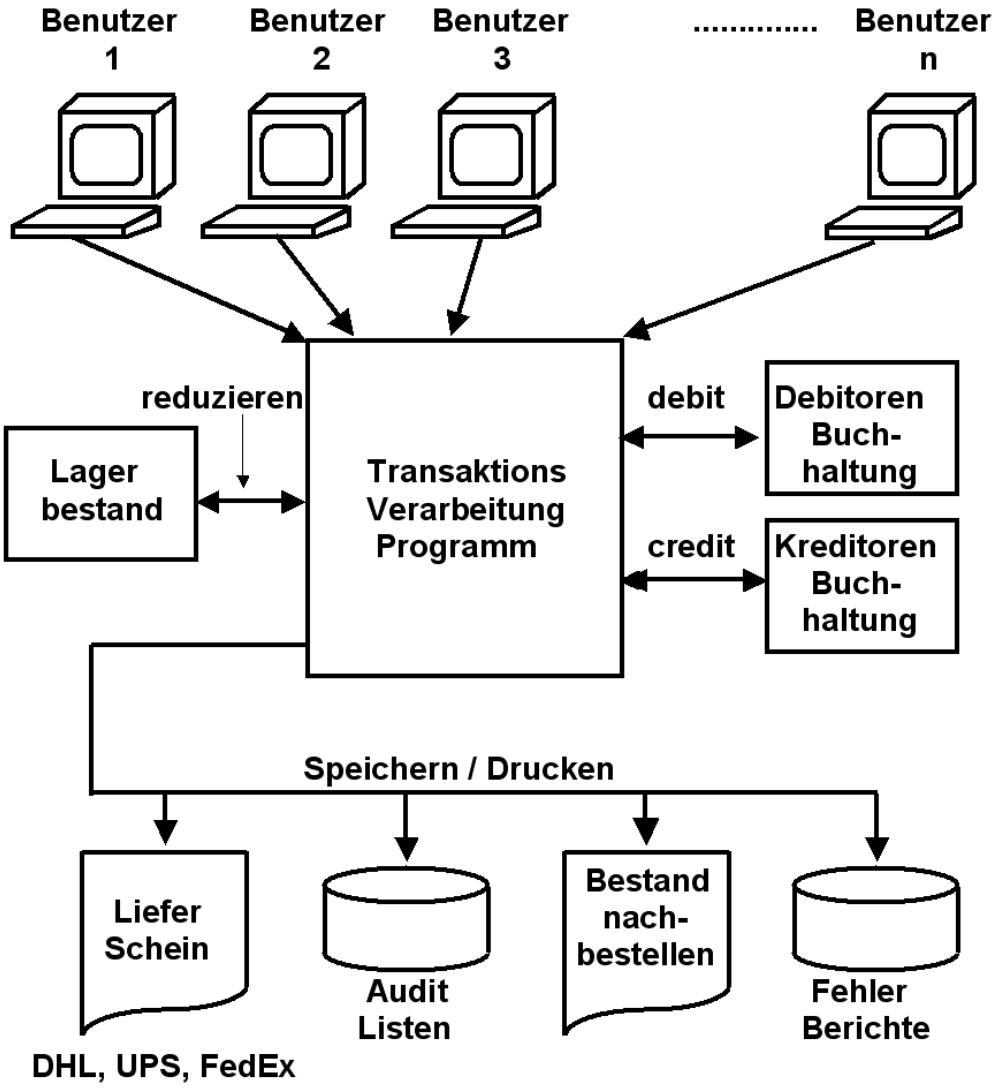


Abb. 7.1.14
Auftragseingang-Bearbeitung

Das Beispiel in Abb. 7.1.14 zeigt das Auftragseingangssystem eines Handelsunternehmens (z. B. Otto Versand), das Aufträge ausliefert, Bezahlungen aufzeichnet, den Orderstatus überprüft und den Warenbestand im Lager überwacht. Die Benutzer sind Kunden, die sich mittels Ihres PCs über das Internet einloggen und jeweils eine Transaktion in der Form eines Bestellvorgangs auslösen. Der Code für ein Beispiel dieser Art ist als ein weit verbreitetes Benchmark (TPC-C) verfügbar (http://www.tpc.org/tpcc/default.asp).

Bei der Bestellung wird der Lagerbestand des bestellten Artikels (z.B. Herrenpullover Gr.43, Bestell Nr. 4711) um die bestellte Menge verringert. Die Debitoren- und Kreditoren-Buchhaltung nehmen Finanzbuchungen vor. Ein Paketdienst (z.B. DHL, UPS, Fedex usw.) erhält einen elektronischen Lieferschein und Auslieferungsauftrag. Ein Zulieferant erhält eine Nachbestellung um den Bestand des bestellten Artikels wieder aufzufüllen. Es werden Audit-Listen und Fehlerberichte erstellt.

Eine teilweise ausgeführte Transaktion wird evtl. wieder rückgängig gemacht, z.B. weil der bestellende Kunde seinen Kreditrahmen überzogen hat oder die angegebene Versandanschrift von keinem Paketdienst angefahren werden kann.

Etwa 80 % aller betrieblichen Anwendungen werden als Transaktionen verarbeitet. Die transaktionalen Verarbeitungsabläufe erfolgen teilweise interaktiv als Client/Server Anwendungen, teilweise als Stapelverarbeitung (Batch Processing).

Interaktive Beispiele:

Auskunftsysteme
Buchungssysteme (z.B. Flugplatzreservierung)
Geldausgabeautomaten
Auftragsbearbeitung, Buchbestellung bei Amazon
Angebot bei eBay abgeben

Stapelverarbeitung Beispiele:

Monatliche Lohn/Gehaltsabrechnung
Jährliche Bilanz erstellen

Das Verhältnis der Rechnerbelastung interaktiv zu Stapel beträgt bei den meisten Unternehmen etwa 60-70% zu 30-40%.

7.1.10 Transactional Memory

Transationaler Memory ist eine zukünftige Entwicklung mit noch unbekannten Potential. Es handelt sich um ein Hauptspeicherkonzept für parallele Berechnungseinheiten, die auf gemeinsame Speicherbereiche zugreifen, wie z.B. Threads oder Mehrprozessorsysteme. Ziel ist es, damit die Ausführungsgeschwindigkeit gegenüber bisherigen Synchronisationsverfahren zu steigern, sowie die Schwierigkeiten der Synchronisierung zu lösen

Transationaler Memory wird mit Hilfe von einer Gruppe von neuartigen Maschinenbefehlen implementiert. Diese erlauben eine Gruppe von regulären Maschinenbefehlen zu definieren, die atomar ausgeführt werden soll.

Transationaler Memory wurde erstmalig im IBMs BlueGene/Q und zEC12 Rechner eingeführt. Intels nächste Prozessorgeneration Haswell wird ebenfalls Transactional Memory in Hardware unterstützten.

TSX: Transactional Synchronisation Extentions, so heißt die Befehlserweiterung, deren Beschreibung man jetzt als bei Intel in der Neufassung der "Intel Architecture Instruction Set Extensions, Programming Reference" herunterladen kann (PDF). Mit TSX soll die Synchronisation zwischen Threads verbessert und vor allem beschleunigt werden. Threads müssen sich bei Zugriffen auf gemeinsame Bereiche miteinander synchronisieren, was viel Zeit kosten kann. Bei Transactional Memory arbeiten die Threads stattdessen zunächst einmal unsynchronisiert und erst beim "Commit" wird überprüft, ob es einen Konflikt gegeben hat. In dem Fall muss die Transaktion verworfen und wiederholt werden, aber das ist vergleichsweise selten.

Konzepte in Software gibt es schon lange, doch die sind bislang zumeist zu ineffizient. Intels TSX bieten dem Programmierer zwei Schnittstellen: Hardware Lock Elison (HLE) mit den neuen Präfixen XACQUIRE und XRELEASE sowie eine Variante namens Restricted Transactional Memoyr RTM, die die neuen Instruktionen XBEGIN, XEND und XABORT bietet. HLE ist die klassische Form, die sich in bestehende Programmkonzepte mit "mutual exclusion" leicht einbringen lässt, RTM ist flexibler, erfordert aber eine Neufassung des Konzepts.

http://www.heise.de/newsticker/meldung/Transactional-Memory-fuer-Intels-Haswell-Prozessor-1432877.html
10.02.2012

7.2 SQL

7.2.1 Structured Query Language

SQL (Structured Query Language) ist eine Programmiersprache für die Abfrage (query) und Abänderung (modify) von Daten und für die Administration von relationalen Datenbanken. Für die hierarchische IMS Datenbank existiert DL/1 als Alternative Programmiersprache.

SQL erlaubt

- **retrieval,**
- **insertion,**
- **updating, and**
- **deletion von Daten.**

In den 70er Jahren entwickelte eine Gruppe am IBM San Jose Research Laboratory das "System R" relationale Database Management System. Diese Arbeit basierte auf einer bahnbrechenden Veröffentlichung von IBM Fellow Edgar F. Codd: "A Relational Model of Data for Large Shared Data Banks". Donald D. Chamberlin und Raymond F. Boyce (beide IBM) entwickelten in 1974 einen ersten Prototypen unter dem Namen System/R. Die "Structured English Query Language" (SEQUEL) wurde für die Manipulation von Daten des System R entworfen. Das Acronym SEQUEL wurde später in SQL abgeändert.

Das erste relationale Datenbank Produkt wurde von der Firma Relational Software herausgebracht (später in Oracle umbenannt). IBM zögerte mit dem eigenen DB2 Produkt, weil das Leistungsverhalten gegenüber IMS als unzureichend angesehen wurde. Die Firma Oracle erbrachte aber den Nachweis, dass relationale Datenbanken ein attraktives Produkt trotz des deutlich schlechteren Leistungsverhaltens sind, und wurde damit Marktführer auf dem relationalen Datenbank-Sektor.

Das Leistungsverhalten von Relationalen Datenbanken wurde in den folgenden Jahrzehnten deutlich verbessert. IMS und Adabas haben aber nach wie vor ein besseres Leistungsverhalten, und werden deshalb immer noch eingesetzt.

Heute zerfällt der relationale Datenbankmarkt in 2 Teile: Relationale Datenbanken für Unix/Linux/Windows und für z/OS. Unter Unix/Linux/Windows hat Oracle nach wie vor einen höheren Marktanteil als DB2. Unter z/OS dominiert DB2; Oracle hat hier eine nur minimale Bedeutung. Ein interessantes Streitgespräch ist zu finden unter

http://mainframeupdate.blogspot.com/2010/02/oracle-versus-ibm-and-db2.html

Die DB2 Version unter Unix/Linux/Windows (distributed Version) ist kompatibel mit der DB2 Version unter z/OS. Die z/OS Version weist aber zahlreiche zusätzliche Funktionen auf, die in der Unix/Linux/Windows nicht vorhanden sind, z.B. Unterstützung für den Sysplex, die Coupling Facility, den Workload Manager, CICS usw.

SQL wurde später von der ANSI und danach von der ISO standardisiert.. Die meisten Datenbank Management Systeme implementieren diese Standards, oft unter Hinzufügung proprietären Erweiterungen. Die proprietären Erweiterungen bringen in der Regel wesentliche Vorteile, und werden deshalb gerne benutzt. Trotz aller Standardisierungsbemühungen ist es aus diesem Grunde in der Regel nicht möglich, SQL Code von einer Datenbank auf eine andere Datenbank ohne Eingriffe zu portieren.

Dies gilt für die Nutzung unter Cobol, Fortran, REXX, C++, PL/1 oder ADA. Für die Nutzung unter Java existieren die Java Standards SQLJ und JDBC mit besserer Kompatibilität.

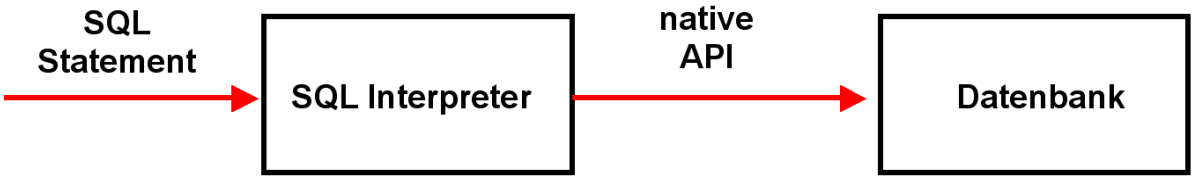


Abb. 7.2.1
Übersetzung von SQL Statements

Neben SQL existiert für jede Datenbank eine nicht standardisierte, proprietäre, native Application Programming Interface (API). Ein Interpreter übersetzt bei jeder Anfrage das SQL Statement in die native API. Da dies Aufwand bedeutet, sind Zugriffe direkt in der nativen API in der Regel performanter als Zugriffe in SQL.

7.2.2 Embedded SQL

Ein Anwendungsprogramm implementiert Datenbankzugriffe über eingebettete SQL-Anweisungen. Diese werden durch "exec sql" eingeleitet und durch ein spezielles Symbol (Delimiter, ";" in C++) beendet. Dies erlaubt einem Precompiler die Unterscheidung der exec sql Anweisungen von anderen Anweisungen.

```
main( )
{
   ........
   ........
   exec sql insert into PERS (PNR, PNAME) values
                  (4711, ´Ernie´);
   ........
   ........
}
```

Abb. 7.2.2
Embedded SQL Beispiel 1

Cobol verwendet statt dessen „END EXEC als Delimiter:

```
  EXEC SQL
       UPDATE CORPDATA/EMPLOYEE
           SET SALARY = SALARY * :PERCENTAGE
           WHERE COMM >= :COMMISSION
  END-EXEC.
```

Abb. 7.2.3
Embedded SQL Beispiel 2

Ein Programm kann viele EXEC SQL Statements enthalten

```
exec sql include sqlca; /* SQL Communication Area*/
main ()
{
exec sql begin declare section;
 char X[8];
 int   GSum;
exec sql end declare section;
exec sql connect to dbname;
exec sql insert into PERS (PNR, PNAME) values (4711, 'Ernie');
exec sql insert into PERS (PNR, PNAME) values (4712, 'Bert');
printf ( "ANR ? ") ; scanf(" %s", X);
exec sql select sum (GEHALT) into :GSum from PERS  where ANR = :X;
printf ("Gehaltssumme %d\n", GSum)
exec sql commit work;
exec sql disconnect;
}
```

Abb. 7.2.4
Embedded SQL Beispiel 3

Eingebettete SQL Anweisungen werden durch "EXEC SQL" eingeleitet und durch spezielles Symbol (hier ";" oder „END EXEC“) beendet, um einem Precompiler die Unterscheidung von anderen Anweisungen zu ermöglichen

Der Oracle oder DB/2 Precompiler greift sich die exec sql Befehle heraus und übersetzt sie in Anweisungen, die der C-Compiler versteht.

Die connect Anweisung baut die Verbindung zwischen Klienten und Server auf.

Es wird eine Kopie von einem Teil der DB Tabelle erstellt, gegen die alle SQL Befehle Änderungen vornehmen. Die commit Anweisung macht die vorhergehenden SQL Befehle entgültig.

Der Kommunikationsbereich SQLCA dient der Rückgabe von Statusanzeigern u.a..

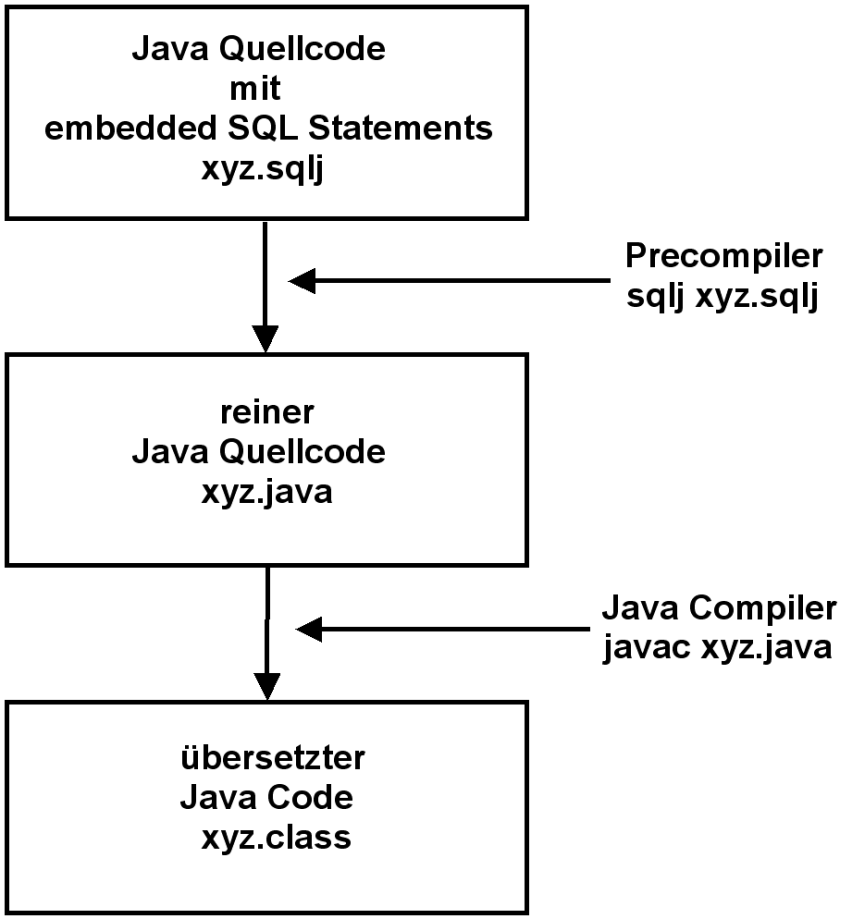


Abb. 7.2.5
SQLJ Precompiler

Embedded SQL bedeutet, wir fügen SQL Kommandos in normale Programmiersprachen wie Cobol, PL/1, C/C++ oder Java ein.

Der Cobol, C/C++ oder Java Compiler versteht die SQL Kommandos nicht. Sie müssen zunächst mit einem Pre-Compiler in Funktionsaufrufe der entsprechenden Programmiersprache übersetzt werden.

Der Precompiler übersetzt embedded SQL Kommandos in entsprechende Datenbank API Calls.

Relationale Datenbanken wie DB2, Oracle, Sybase erfordern unterschiedliche Precompiler

Literatur: IBM Redbook e-business Cookbook for z/OS Volume II: Infrastructure,July 2002, section 4.5.

7.2.3 Zugriff auf eine SQL Datenbank

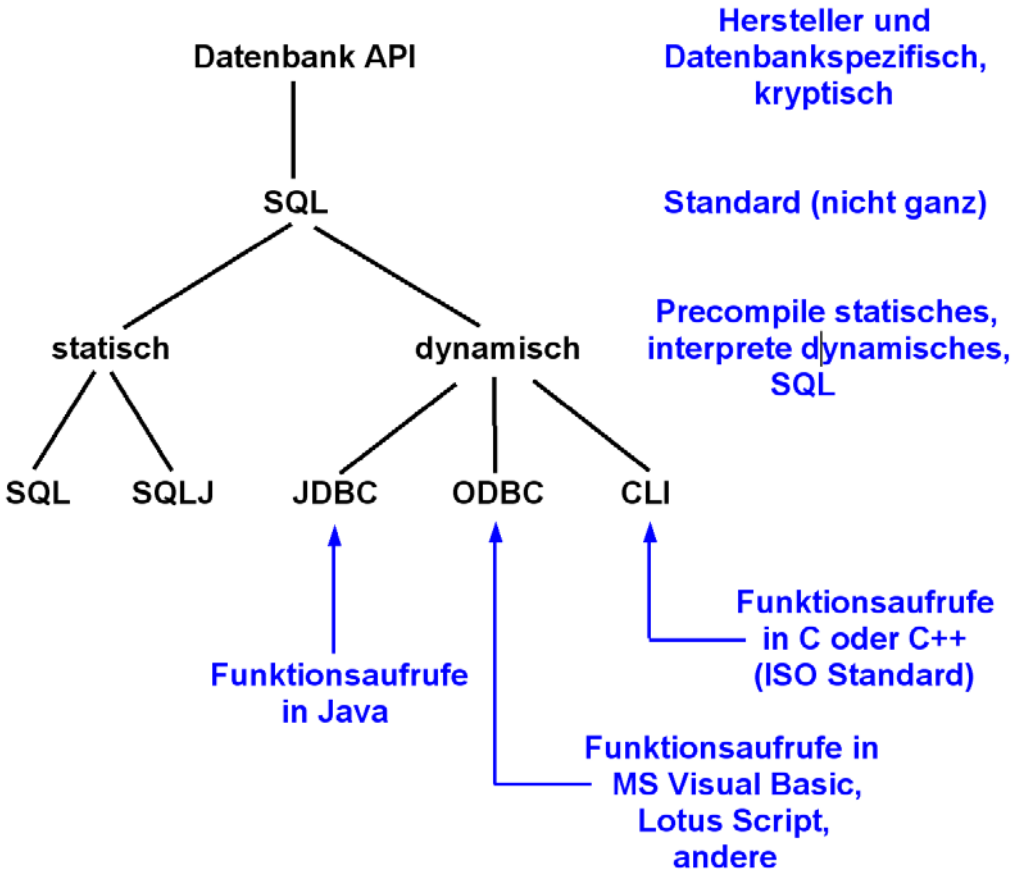


Abb. 7.2.6
Alternative Zugriffsverfahren

Wir unterscheiden zwischen statischem und dynamischem SQL.

Statisches SQL arbeitet mit einem Precompiler. Alle Angaben zu dem Datenbankzugriff müssen zur Compile Zeit festgelegt werden.

Dynamisches SQL ermöglicht Angabe von Parametern erst zur Laufzeit. Z.B. Benutzer gibt gewünschten Datenbanknamen und Tabellennamen in Datenfeld einer HTML Seite ein.

JDBC, ODBC und CLI ist dynamisches SQL für unterschiedliche Programmiersprachen

DB2connect ist ein Klient für statische SQL Zugriffe auf eine DB2 Datenbank. SQLJ ist die Java - spezifische Version von DB2connect.

7.2.4 Vor- und Nachteile von statischem gegenüber dynamischem SQL

```
#sql iterator SeatCursor(Integer row, Integer col,
         String type, int status);
     Integer status = ?;
     SeatCursor sc;
     #sql sc = {
     select rownum, colnum from seats where status <= :status
        };
while(sc.next())                                    SQLJ
          {
 #sql { insert into categ values(:(sc.row()),
         :(sc.col())) };
          }
sc.close();
```

Abb. 7.2.7
Statisches SQL Programmierbeispiel

```
Integer status = ?;
PreparedStatement stmt = conn.prepareStatement("select
         row, col from seats where status <= ?");
if (status == null) stmt.setNull(1,Types.INTEGER);
else   stmt.setInt(1,status.intValue());
ResultSet sc = stmt.executeQuery();
while(sc.next())
{
int row = sc.getInt(1);
boolean rowNull = sc.wasNull();                    JDBC
int col = sc.getInt(2);
boolean colNull = sc.wasNull();
PreparedStatement stmt2 =
         conn.prepareStatement("insert into categ
         values(?, ?)");
if (rowNull) stmt2.setNull(3,Types.INTEGER);
else stmt2.setInt(3,rownum);
if (colNull) stmt2.setNull(4,Types.INTEGER);
else stmt2.setInt(4,colnum);
stmt2.executeUpdate();
stmt2.close();
}
sc.close();
stmt.close();
```

Abb. 7.2.8
JDBC Programmierbeispiel

Was sind die Vor- und Nachteile von SQLJ gegenüber JDBC in Bezug auf Performance?

Die Antwort ist, es hängt sehr stark von den Umständen ab. Ein guter Vergleich ist zu finden unter
http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.db2.udb.doc/ad/c0005785.htm .

Ein dynamischer Datenbankzugriff auf eine relationale Datenbank aus einem Java Anwendungsprogramm heraus verwendet das JDBC (Java Data Base Connectivity) Zugriffsverfahren. Es bietet höhere Flexibilität als das statische Verfahren. Z.B. muss der Name der Datenbank erst im Augenblick des Zugriffs und nicht schon zur Compile-Zeit angegeben werden.

Ein Nachteil ist häufig eine schlechtere Performance. Auch ist der Programmieraufwand größer, wie ein Vergleich mit dem vorhergehenden statischen SQLJ Beispiel zeigt.

Im Allgemeinen hat ein Anwendungsprogramm mit dynamischen SQL größere Start-up (oder anfängliche) Kosten pro SQL Statement, weil das SQL Statement vor der Benutzung erst übersetzt (compiled) werden muss. Nach der Übersetzung sollte die Ausführungszeit beim dynamischen SQL ähnlich sein wie beim statischen SQL. Wenn mehrere Klienten das gleiche dynamische Programm mit den gleichen Statements aufrufen, benötigt nur der erste Benutzer den zusätzlichen Übersetzungsaufwand.

7.3 Stored Procedures

7.3.1 Zwei-Tier Client/Server Architecture

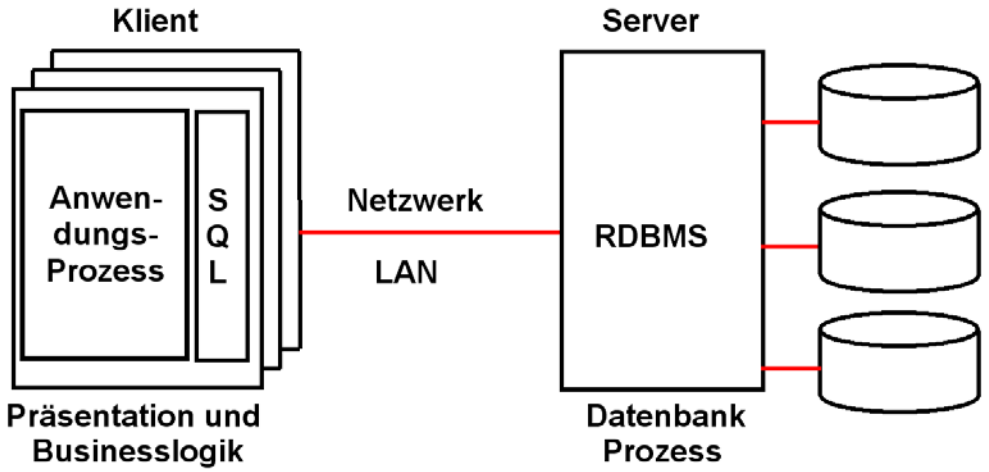


Abb. 7.3.1
Ein SQL Client wird für die Kommunikation mit einer relationalen Datenbank benutzt

Der Anwendungsprozess auf dem Klientenrechner sendet SQL Befehle an den Datenbankrechner. Auf dem Client Rechner befindet sich eine SQL Client Komponente, die in der Lage ist, SQL Anfragen über das Netz an den Datenbankserver zu senden.

Der Datenbankprozess stellt sicher, dass jedes einzelne SQL Statement unter Einhaltung der ACID Bedingungen ausgeführt wird.

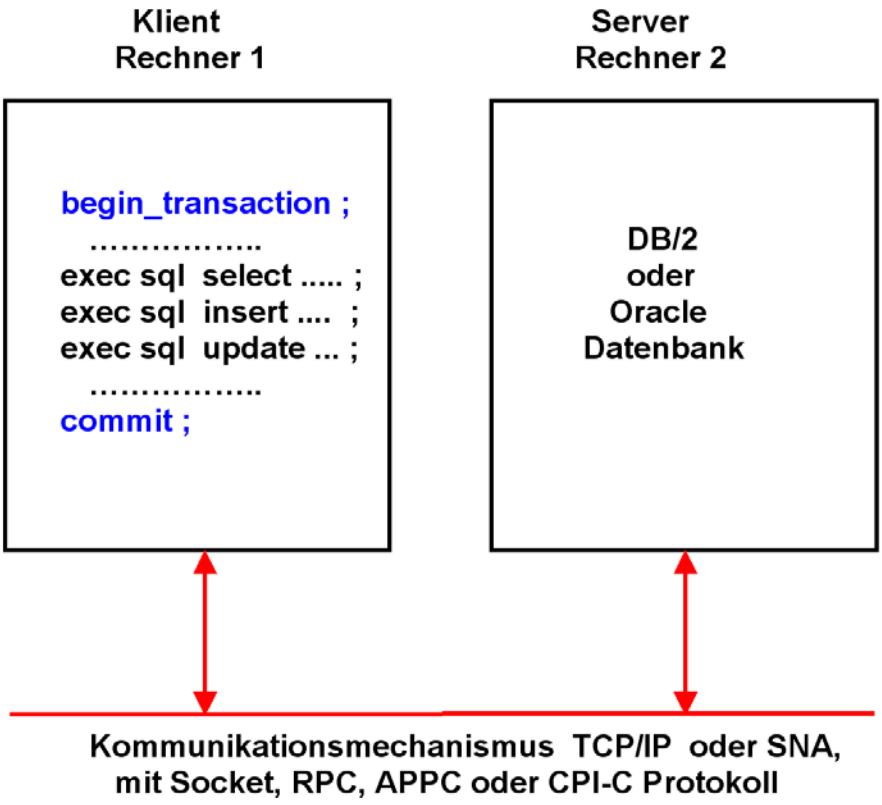


Abb. 7.3.2
ACID Eigenschaften für eine Gruppe von SQL Aufrufen

Der Anwendungsprozess verfügt über einen SQL Klienten, der die Kommunikation mit dem Datenbank Prozess herstellt. Dabei ist es gleichgültig, ob der Datenbank Prozess auf dem gleichen Rechner wie der Anwendungsprozess läuft, oder auf einem entfernten (remote) Rechner läuft, der mit dem Anwendungsrechner über TCP/IP verbunden ist.

In vielen Fällen führt der Klient eine Reihe von EXEC SQL Kommandos aus, die insgesamt als Gruppe ACID Eigenschaften haben sollen.

Der Rechner führt die Anweisungen zwischen

```
exec sql begin_transaction
```

und

```
exec sql commit
```

als Arbeitseinheit (Work Unit) entsprechend den ACID Anforderungen aus.

Dies stellen die beiden Schlüsselwörter begin_transaction und commit sicher.

Eine Gruppe mit mehrfachen SQL Statements, die ACID Eigenschaften haben, wird als „embedded SQL“ Transaktion bezeichnet.

7.3.2 Embedded SQL Transaktion mit mehrfachen SQL Statements

```
DebitCreditApplication( )
{   receive input message;
    exec sql begin_transaction ;      /* start transaction */
    Abalance = DoDebitCredit (.....);
                        •
                        •
                        •
      exec sql  select ..... ;
          exec sql  insert ....  ;
          exec sql  update ... ;
                        •
                        •
                        •
    if (Abalance < 0 && delta < 0)
         {   exec sql rollback ;    }
    else {
         send output message;
         exec sql commit ;             /* end transaction */
         }
}
```

Abb. 7.3.3
Beispiel einer embedded SQL Transaktion

In dem gezeigten Code Fragment fängt der transaktionale Abschnitt mit exec sql begin_transaction an, und hört mit entweder exec sql commit oder mit exec sql rollback auf. Rollback bewirkt, dass alle Datenbank-Änderungen wieder rückgängig gemacht werden.

An Stelle von Schlüsselworten wie rollback sind auch Schlüsselworte wie abort oder backout gebräuchlich.

Eine Embedded SQL Transaktion mit mehrfachen SQL Statements stellt ein relativ simples Programmiermodell dar. Sie ist einfach zu entwerfen und zu implementieren. Da der ganze Anwendungscode auf dem Klienten läuft, kann ein einzelner Programmierer die Verantwortung für die ganze Anwendung übernehmen.

Es existiert jedoch eine Reihe von Nachteilen.

Da die gesamte Anwendung auf dem Klienten-Rechner läuft, ist für jedes EXEC SQL Statement eine getrennte Netzwerk I/O Operation erforderlich. Dies kann die Performance stark herabsetzen. Weiterhin muss das Anwendungsprogramm über detaillierte Kenntnisse des Datenbank-Designs verfügen. Jede Änderung im Datenbank-Design erfordert eine entsprechende Änderung im Anwendungsprogramm. Schließlich erfordern mehrfache Kopien des Anwendungsprogramms auf mehreren Arbeitsplatzrechnern einen erhöhten Administrationsaufwand.

Stored Procedures lösen diese Probleme.

7.3.3 Stored Procedure

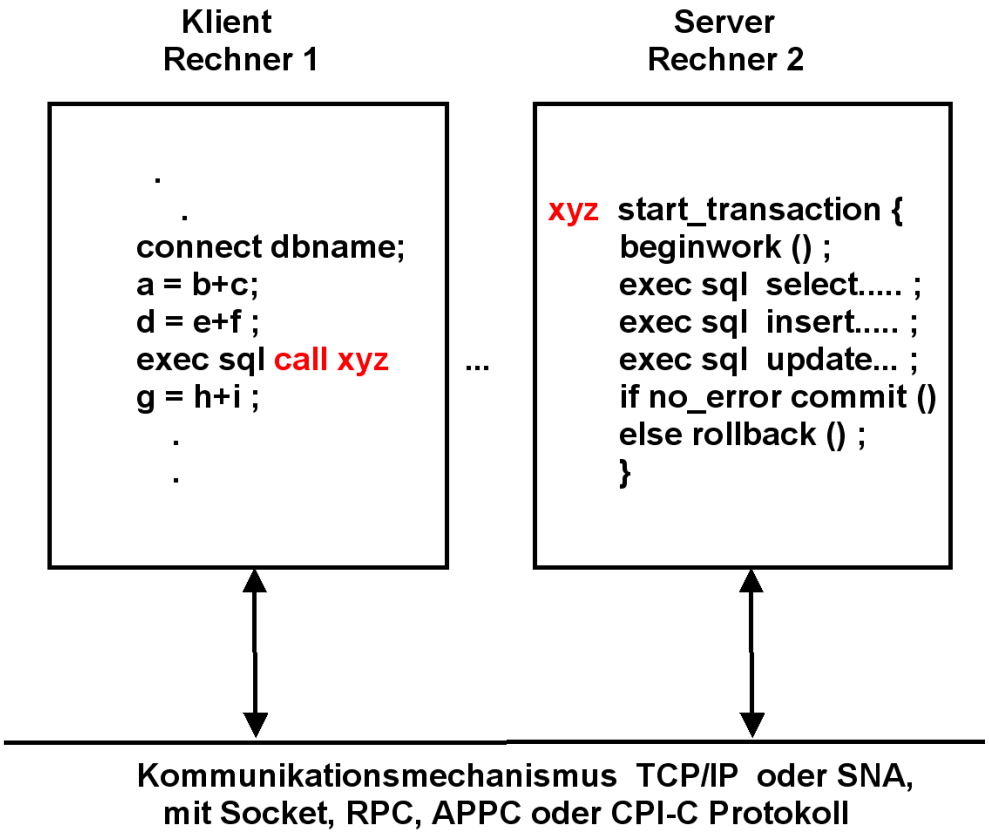


Abb. 7.3.4
ACID Eigenschaften für eine Gruppe von SQL Aufrufen

Mit Hilfe einer "Stored Procedure" wird die Gruppe von SQL Aufrufen nicht auf dem Klienten sondern auf dem Server ausgeführt. Der Klient ruft die Stored Procedure mit Hilfe eines „call" statements auf.

Die Stored Procedure führt eine Gruppe von zusammenhängenden SQL Statements aus. Die Gruppe hat ACID Eigenschaften.

Mit Hilfe des „connect" Statements wählt der Klient die Datenbank mit dem Namen „dbname" (an Stelle einer evtl. anderen angeschlossenen Datenbank) aus.

Abb. 7.3.5 stellt dar, wie durch den Einsatz einer Stored Procedure die Anzahl der SQL Zugriffe über das Netz deutlich reduziert werden kann. Dies verringert das Datenvolumen, welches über das Netz geht.

Zusätzlich wird auch der Aufwand für die TCP/IP Verarbeitung deutlich verringert. Die Stored Procedure läuft in der Regel auf dem gleichen Server wie die Datenbank, und zwar in einem eigenen Address Space, getrennt von dem Datenbank Address Space. Die Kommunikation zwischen den beiden Address Spaces erfolgt aber über den Betriebssystem Kernel und erfordert wesentlich weniger CPU Zyklen als die TCP/IP Verarbeitung.

Als Ergebnis laufen Stored Procedures deutlich performanter als dies bei einzelnen SQL Aufrufen der Fall ist.

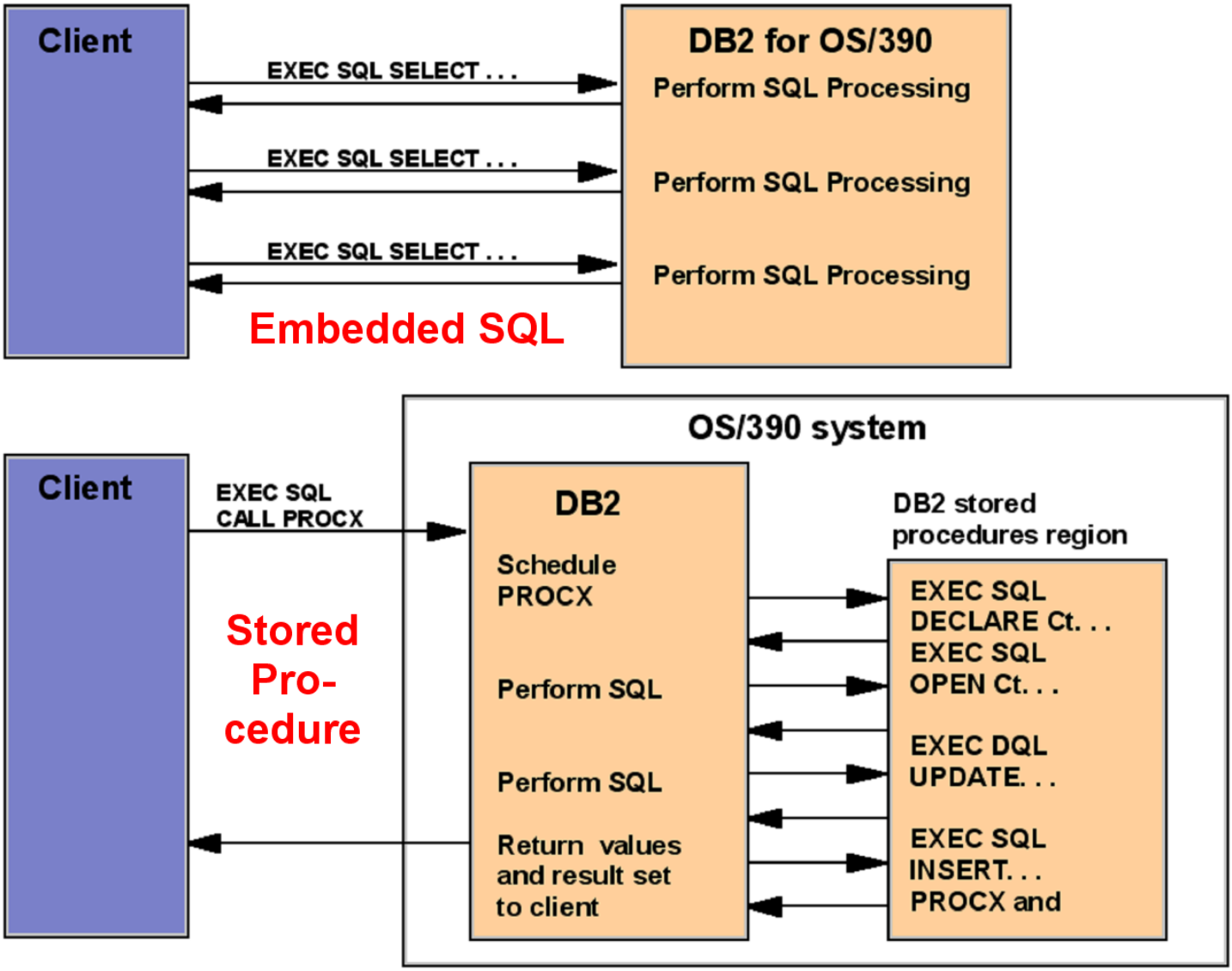


Abb. 7.3.5
Eine Stored Procedure verursacht weniger Kommunikation über das Netz

Stored Procedures werden manchmal als „TP light“ bezeichnet, im Gegensatz zu einem „TP heavy“ Transaktionsmonitor, siehe Abschnitt 7.4. Letzterer startet eigene Prozesse für mehrfache Aufrufe; innerhalb der Prozesse können nochmals Threads eingesetzt werden.

```
exec sql include sqlca; /* SQL Communication Area*/
main ()
{
exec sql begin declare section;
  char X[8];
  int  GSum;
exec sql end declare section;
exec sql insert into PERS (PNR, PNAME) values (4711, 'Ernie');
exec sql insert into PERS (PNR, PNAME) values (4712, 'Bert');
printf ( "ANR ? ") ; scanf(" %s", X);
exec sql select sum (GEHALT) into :GSum from PERS
  where ANR = :X;
printf ("Gehaltssumme %d\n", GSum)
exec sql commit work;
exec sql disconnect;
}
```

Abb. 7.3.6
Beispiel für eine Stored Procedure

Die in Abb. 7.3.6 gezeigte Stored Procedure enthält neben SQL Anweisungen auch Statements in einer regulären Programmiersprache. Eine Stored Procedure kann relativ komplexen Programm Code enthalten.

Traditionell verwenden Stored Procedures unter z/OS hierfür reguläre Programmiersprachen wie COBOL, PL/I, C/C++, Assembler, REXX, and Java (external high-level language stored procedure). z/OS unterstützt neben den traditionellen "external high-level language stored procedures" noch "SQL procedures". SQL procedures sind Stored Procedures, die in der Sprache SQL/PSM (SQL Persistent Modules) geschrieben sind. SQL/PSM ist ein ISO Standard.

Es existiert eine ähnliche, aber proprietäre Programmiersprache PL/SQL (Procedural Language/SQL) für Oracle Datenbanken, die sich in der Syntax an die Programmiersprache Ada angelehnt.

Stored Procedures werden vom Datenbank Server in einer Library abgespeichert und mit einem Namen aufgerufen.

Das Klienten Anwendungsprogramm stellt Verbindung zur Datenbank her mit

```
EXEC SQL CONNECT TO dbname
```

und ruft Stored Procedure auf mit

```
EXEC SQL CALL ServProgName (parm1, parm2)
```

Hierbei ist:

parm1 **Variablen Name (Puffer) der eine Struktur definiert (als SQLDA bei DB2 bezeichnet), die zum Datenaustausch in beiden Richtungen benutzt wird.**

parm2 **Variablen Name (Puffer) der eine Struktur definiert die für Return Codes und Nachrichten an den Klienten benutzt wird.**

Stored Procedures bündeln SQL Statements bei Zugriffen auf Relationale Datenbanken. Sie ersetzen viele, vom Klienten an den Server übergebene, SQL Statements durch eine einzige Stored Procedure Nachricht

Beispiel:

Bei einem Flugplatzreservierungssystem bewirkt eine Transaktion die Erstellung oder Abänderung mehrerer Datensätze:

- **Passenger Name Record (neu)**
- **Flugzeugauslastung (ändern)**
- **Platzbelegung (ändern)**
- **Sonderbedingungen (z.B. vegetarische Verpflegung) (ändern)**

7.3.4 Stored Procedure Datenbank Client

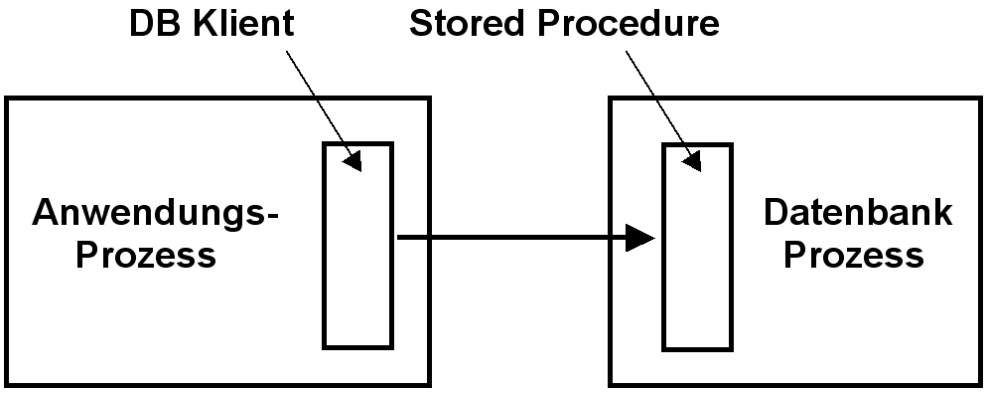


Abb. 7.3.7
Der Aufruf einer Stored Procedure erfolgt nicht über SQL

Bei einer Stored Procedure führt nicht der Anwendungsprozess, sondern der Datenbankprozess, die Gruppe von SQL Statements aus.

Datenbank Hersteller, z.B. IBM, Oracle, Sybase etc. stellen eigene (proprietäre) Datenbank-Klienten zur Verfügung, die mit den Stored Procedures der Datenbank kommunizieren. Datenbank Klienten sind nicht kompatibel miteinander.

Der Klient enthält einen Treiber (DB Klient), der ein proprietäres "Format and Protocol" (FAP) definiert.

FAP unterstützt mehrere Schicht 4 Stacks (TCP/IP, SNA, NetBios, ...)

7.3.5 DB2 Stored Procedures

DB2 ist verfügbar für z/OS, z/VSE, i5/OS (OS/400), and als „DB2 UDB“ (Universal Data Base) für alle verbreiteten Unix (einschließlich Linux) und Windows Betriebssysteme. Es benötigt standardmäßig 3 Address Spaces unter z/OS.

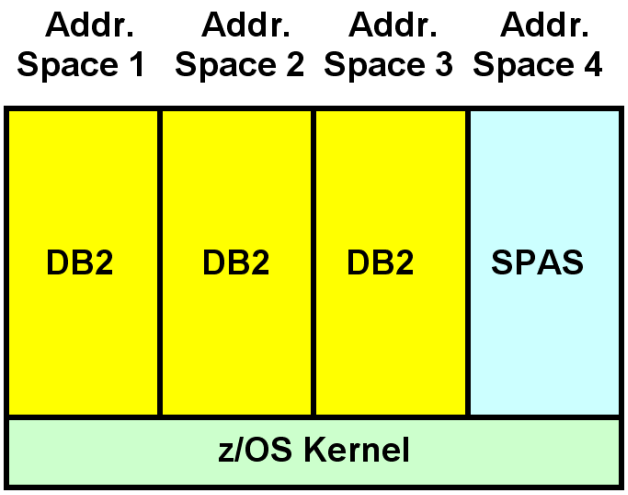


Abb. 7.3.8
Unter z/OS werden Stored Procedures in einem eigenen Address Space (SPAS) ausgeführt

Die Benutzung von Stored Procedures erfordert Verhandlungen mit dem Datenbank Administrator. Das Entwickeln von Stored Procedures ist eine komplexe Aufgabe; das Debuggen kann schwierig sein.

7.3.6 Leistungsverhalten

Das Lehrbuch von R. Orfali, D. Harkey: „Essential Client/Server Survival Guide“. John Wiley, 1994, S. 178, enthält einen detaillierten Leistungsvergleich mit einer einfachen Transaktion, für die der vollständige Code vorliegt. Die Messungen wurden unter dem OS/2 Betriebssystem auf einem Intel 80486 Rechner durchgeführt.

Dies sind die Messergebnisse in ausgeführten Transaktionen pro Sekunde:

Dynamic SQL	**2,2**	**Transaktionen/s**
Static SQL	**3,9**	**Transaktionen/s**
Stored Procedure	**10,9**	**Transaktionen/s**

Zu sehen ist ein deutlicher Performance Vorteil bei der Benutzung von statischen gegenüber dynamischen SQL (Faktor 1,75). Noch deutlicher ist der Performance Vorteil beim Einsatz von Stored Procedures (Faktor 2,79).

Mit einem modernen Rechner wäre die Transaktionsrate (Anzahl ausgeführter Transaktionen pro Sekunde) sehr deutlich höher. Am Verhältnis der Ausführungszeiten würde sich vermutlich nicht viel ändern.

7.3.7 Probleme mit Stored Procedures

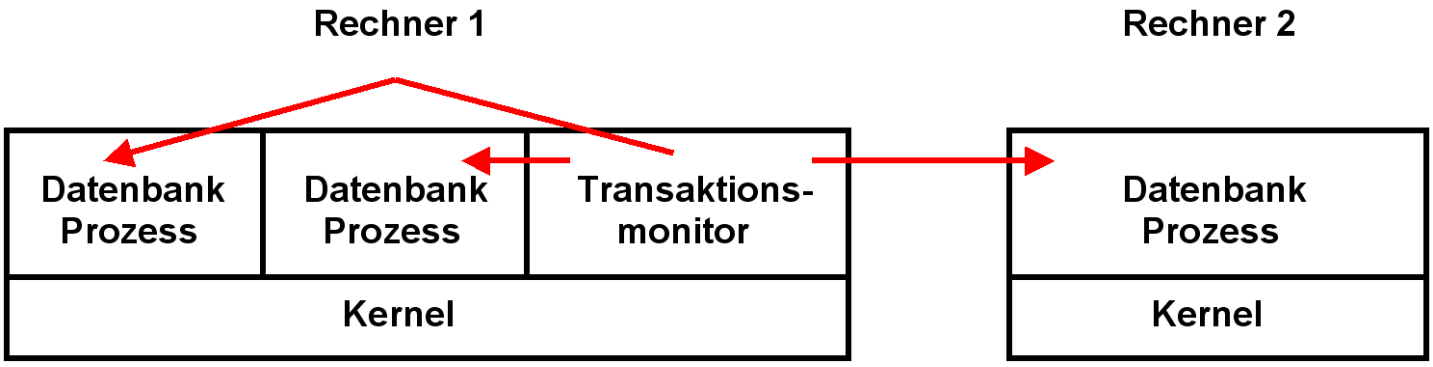


Abb. 7.3.9
Eine Stored Procedure ist Bestandteil einer Datenbank

Die Stored Procedure Laufzeitumgebung ist Bestandteil eines Datenbankprozesses. Häufig ist es jedoch erforderlich, dass eine transaktionale Anwendung auf zwei oder mehr Datenbanken zugreift. Die Zugriffe müssen die ACID Bedingungen erfüllen.

Ein weiteres Problem tritt auf, wenn eine Transaktion auf mehrere heterogene Datenbanken, z.B. von unterschiedlichen Herstellern zugreift. Beispielsweise ist bei der Buchung einer Urlaubsreise in einem Reisebüro ein Zugriff auf die Datenbank einer Fluglinie sowie auf die Datenbank eines Hotels erforderlich. Diese Datenbanken sind nicht identisch.

Zur Lösung implementiert man die Laufzeitumgebung für die Ausführung von Transaktionen als einen getrennten Prozess, der unabhängig vom Datenbankprozess ist, und als „Transaktionsmonitor“ oder „Transaktionsserver“ bezeichnet wird.

Der Transaktionsmonitor (TP Monitor) Prozess übernimmt die Rolle des DB2 Stored Procedure Address Space (SPAS).

Es existieren weitere Vorteile beim Einsatz eines Transaktionsmonitors an Stelle von Stored Procedures:

- **Lastverteilung - Leistungsverhalten - Skalierung**
- **Prioritätssteuerung**
- **Verfügbarkeit (High Availability)**

In Mainframe Installationen werden Transaktionen ganz überwiegend mittels eines Transaktionsmonitors und nicht mittels Stored Procedures ausgeführt. Auch das Leistungsverhalten eines Transaktionsmonitors ist häufig deutlich besser.

7.3.8 Beispiel für Transaktionsmonitore

Transaktionsmonitore sind Softwarepakete (auch als Middleware bezeichnet), die von verschiedenen Herstellern vertrieben werden. Die wichtigsten sind:

- **CICS der Firma IBM**
- **SAP R/3 der Firma SAP**
- **Transaction Server (MTS) der Firma Microsoft**
- **Tuxedo der Firma Bea (heute ein Tochterunternehmen von Oracle)**

Daneben von Bedeutung sind :

- **IMS-DC der Firma IBM**
- **TPF (Transaction Processing Facility) der Firma IBM**
- **UTM der Firma Siemens**
- **NonStop / Guardian / Pathway der Firma Hewlett Packard**

Für die objekt-orientierte Programmierung sind von wachsender Bedeutung:

- **Corba OTS (Object Transaction Service)**
- **EJB JTS (Enterprise Java Bean Transaction Service)**

Unter allen Transaktionsmonitoren hat CICS eine absolut dominierende Position. Weit mehr als die Hälfte aller täglich weltweit ausgeführten Transaktionen werden von CICS verarbeitet.

7.3.9 Begriffe

Die Begriffe Transaction Monitor, Transaction Server und Transaction Service werden (grob gesehen) austauschbar verwendet und bedeuten mehr oder weniger das Gleiche. Ein Transaction Manager ist eine Komponente, die mehrere Transaction Monitore steuert, und sollte mit den Begriffen Transaction Monitor, Transaction Server und Transaction Service nicht verwechselt werden.

Resource Manager ist ein Überbegriff. Ein Resource Manager kann – muss aber nicht – ein Transaktionsmonitor sein.

7.4 Transaction Processing Monitor

7.4.1 Locking

Ein fundamentales Problem der Transaktionsverarbeitung besteht darin, dass ein Transaktionsmonitor aus Performance Gründen in der Lage sein muss, Hunderte oder Tausende von Transaktionen gleichzeitig auszuführen.

Um die Isolation (das `i` in ACID) zu gewährleisten muss der Anschein erweckt werden, dass alle Transaktionen der Reihe nach bearbeitet werden, also dass Transaktion Nr. 2 erst dann an die Reihe kommt, wenn die Verarbeitung von Transaktion Nr. 1 abgeschlossen ist. Wenn beide Transaktionen auf unterschiedliche Daten zugreifen, ist es kein Problem sie trotzdem parallel zu verarbeiten. Wenn beide Transaktionen aber auf die gleichen Daten zugreifen, muss sichergestellt werden, dass keine Verletzung der ACID Eigenschaften auftritt. Dies geschieht mit Hilfe von Locks (Sperren).

Angenommen zwei Transaktionen, die auf die beiden zwei Variablen d1 und d2 zugreifen.

Anfangswerte: d1 = 15, d2 = 20

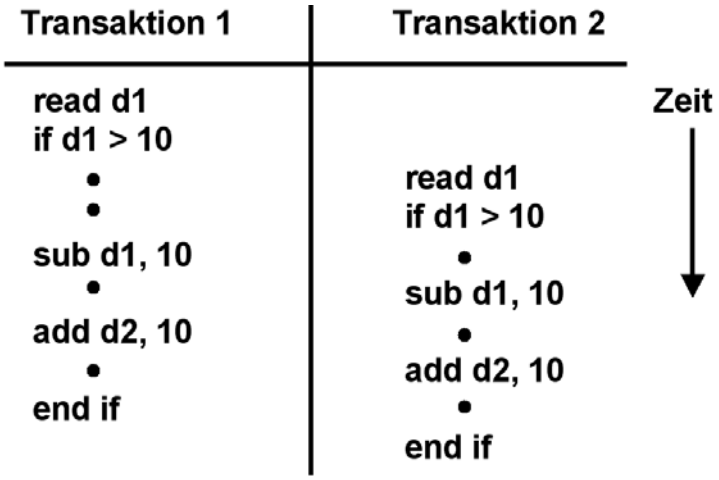


Abb. 7.4.1
Anfangswerte: d1 = 15, d2 = 20

Das Ergebnis ist: d1 = - 5, d2 = 40, obwohl die if-Bedingung ein negatives Ergebnis verhindern sollte.

7.4.2 Concurrency Control

Concurrency Control (Synchronisierung) ist zwischen den beiden Transaktionen erforderlich, um dies zu verhindern. Der Begriff, der dies beschreibt ist Serialisierung (serializability).

Hierfür wird jedes Datenfeld, auf das zwei Transaktionen gleichzeitig zugreifen können, mit einem Lock (einer Sperre) versehen.

Eine Transaktion, welche auf dieses Datum zugreifen will, erwirbt zunächst das Lock. Dies geschieht dadurch, dass das Lock, welches normalerweise den Wert 0 hat, auf 1 gesetzt wird.

Die Transaktion verrichtet nun ihre Arbeit. Nach Abschluss wird das Lock wieder freigegeben (auf 0 gesetzt).

Eine andere Transaktion, die während dieser Zeit ebenfalls versucht, auf das gleiche Datenfeld zuzugreifen, wird daran gehindert, weil das Lock den Wert 1 hat. Sie muss warten, bis das Lock wieder den Wert 0 hat.

Bei großen Datenbeständen können viele Millionen von Locks für unterschiedliche Datenobjekte vorhanden sein

In der Praxis unterscheidet man zwischen Read und Write Locks. Read Locks verhindern „Dirty Reads“. Write Locks verhindern “Lost Updates”.

Mehrere Transaktionen können gleichzeitig ein Read Lock besitzen. Sie können sich gleichzeitig über den Inhalt eines Datenfeldes informieren. Solange der Inhalt des Datenfeldes nicht geändert wird, ist noch nichts passiert.

Wenn eine Transaktion den Inhalt des Datenfeldes ändern will, konvertiert sie das Read Lock in ein Write Lock. Dies bewirkt, dass eine Nachricht an alle anderen Transaktionen geschickt wird, die ein Read Lock für das gleiche Datenfeld besitzen. Letztere wissen nun, dass der Wert, den sie gelesen haben, nicht mehr gültig ist.

Hierzu ist erforderlich, dass nur eine Transaktion in jedem Augenblick ein Write Lock für ein bestimmtes Datenfeld besitzen darf. Write Locks werden deshalb auch als „Exclusive Locks“ bezeichnet.

Angenommen zwei Transaktionen, die auf die beiden zwei Variablen d1 und d2 zugreifen.

Anfangswerte: d1 = 15, d2 = 20

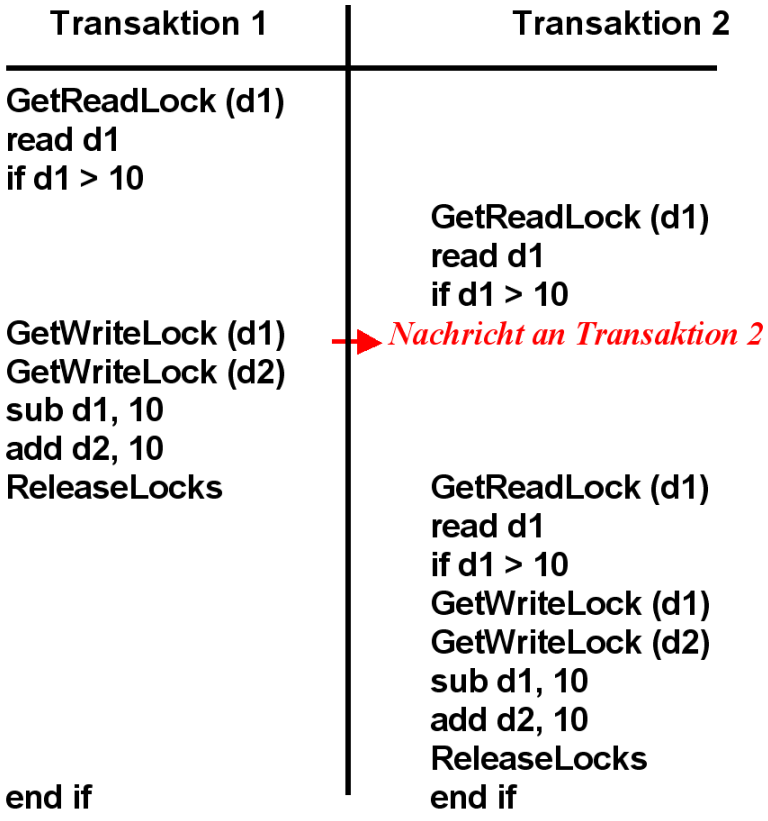


Abb. 7.4.2
Benachrichtigung an Transaktion 2 betr. Ungültiges ReadLock

Ergebnis: d1 = + 5, d2 = 30. Transaktion 2 ändert die beiden Variablen d1 und d2 nicht mehr

Fred Brooks erfand Locks in 1964 bei der Entwicklung von OS/360, damals als "exclusive control" bezeichnet.

7.4.3 Two-Phase Locking

In Transaktionssystemen und Datenbanksystemen werden Locks (Sperren) benutzt, um Datenbereiche vor einem unautorisierten Zugriff zu schützen. Jedem zu schützenden Datenbereich ist ein Lock fest zugeordnet. Ein Lock ist ein Objekt welches über 4 Methoden und zwei Zustände S und E verfügt. Die Methode

- **GetReadLock** **reserviert S Lock (shared),**
- **GetWriteLock** **reserviert E Lock (exclusive),**
- **PromoteReadtoWrite** **bewirkt Zustandswechsel S $\rightarrow$ E,**
- **Unlock** **gibt Lock frei.**

Mehrere Transaktionen können ein S Lock für den gleichen Datenbereich (z.B. einen Datensatz) besitzen. Nur eine Transaktion kann ein E Lock für einen gegebenen Datenbereich besitzen. Wenn eine Transaktion ein S Lock in ein E Lock umwandelt, müssen alle anderen Besitzer des gleichen S Locks benachrichtigt werden.

Normalerweise besitzt eine Transaktion mehrere Locks.

In einer Two-Phase Locking Transaktion finden alle Lock Aktionen zeitlich vor allen Unlock Aktionen statt. Eine Two-Phase Transaktion hat eine Wachstumsphase (growing), während der die Locks angefordert werden, und eine Schrumpf- (shrink) Phase, in der die Locks wieder freigegeben werden.

Two Phase Locking ist nicht zu verwechseln mit dem 2-Phase Commit Protokoll der Transaktionsverarbeitung (siehe in diesem Abschnitt weiter unten).

7.4.4 Programmierung eines Locks

Die Implementierung eines Locks wird durch spezielle Maschinenbefehle unterstützt, z.B. `Test und Set` sowie `Compare und Swap` im Falle von System z. Diese ermöglichen eine atomare (gleichzeitige) Änderung mehrerer Datenelemente.

Höhere Programmiersprachen benutzen diese Befehle um Lock Funktionen zu ermöglichen, z.B. in C++:

Um eine Funktion `incr` zu implementieren, die einen Zähler x hochzählt, könnte Ihr erster Versuch sein:

```
void incr()
     {
          x++;
     }
```

Abb. 7.4.3
Zählerbeispiel 1

Dies garantiert nicht die richtige Antwort, wenn die Funktion `incr` von mehreren Threads gleichzeitig aufgerufen wird. Das Statement x++ besteht aus drei Schritten: Abrufen des Werts x; Erhöhung um 1, Ergebnis in x speichern. In dem Fall, dass zwei Threads dies im Gleichschritt tun, lesen beide den gleichen Wert, erhöhen um 1, und speichern den um 1 erhöhten Wert. x wird nur um 1 an Stelle von 2 inkrementiert. Ein Aufruf von `incr()` verhält sich nicht atomar; es ist für den Benutzer sichtbar, dass er aus mehreren Schritten besteht. Wir könnten das Problem durch die Verwendung eines Mutex lösen, welcher nur von einem Thread zu einem Zeitpunkt gesperrt werden kann:

Ein Mutex benutzt Maschinenbefehle wie `Test und Set` oder `Compare und Swap` um eine atomare Ausführung der Increment Funktion zu gewährleisten.

```
void incr()
     {
          mtx.lock();
          x++;
          mtx.unlock();
     }
```

Abb. 7.4.4
Zählerbeispiel 2

In Java könnte das so aussehen

```
void incr()
     {
          synchronized(mtx) {
               x++;
          }
     }
```

Abb. 7.4.5
Zählerbeispiel 3

Hans-J. Boehm, Sarita V. Adve: You Don't Know Jack About Shared Variables or Memory. Models. Communications of the ACM, February 2012, vol. 55, no. 2.

Sie können ein transaktionales Anwendungsprogramm schreiben, welches diese Konstrukte benutzt. Das ist jedoch extrem aufwendig und fehleranfällig. Sie benutzen deshalb einen Transaction Processing Monitor, der Ihnen diese Arbeit abnimmt.

7.4.5 Struktur eines Transaction Processing Monitors

Das Zusammenspiel der einzelnen Komponenten wird in Abb. 7.4.6 dargestellt.

Endbenutzer kommunizieren mit dem TP Monitor mit Hilfe von Nachrichten. Klienten (Arbeitsplatzrechner) werden häufig als „Terminals“ bezeichnet.

1. Presentation Services empfangen Nachrichten und bilden die Datenausgabe auf dem Bildschirm des Benutzers ab.

2. Eingabe-Nachrichten werden mit einer TRID (Transaction ID) versehen und in einer Warteschlange gepuffert.

3. Aus Zuverlässigkeitsgründen muss diese einen Systemabsturz überleben und ist persistent auf einem Plattenspeicher gesichert. Eine ähnliche Warteschlange existiert für die Ausgabe von Nachrichten.

4. Die Benutzer-Autorisation erfolgt über ein Sicherheitssystem, z.B. Kerberos.

5. Der Scheduler verteilt eingehende Bearbeitungsanforderungen auf die einzelnen Server Prozesse.

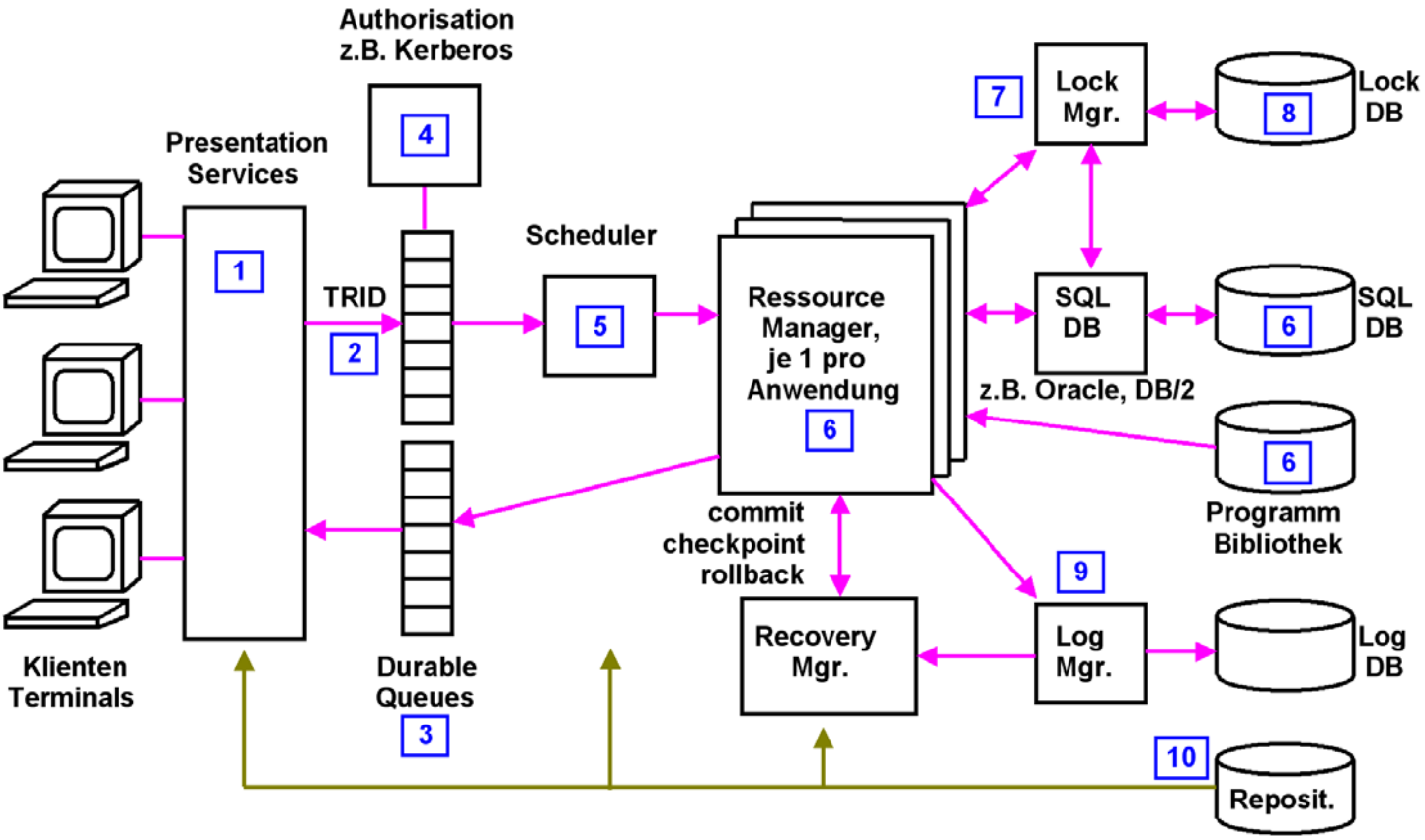


Abb. 7.4.6
TP Monitor

6. **Ein TP Monitor bezeichnet seine Server Prozesse als Ressource Manager. Es existiert ein Ressource Manager pro (aktive) Anwendung. Ressource Manager sind multithreaded; ein spezifischer Ressource Manager für eine bestimmte Anwendung ist in der Lage, mehrere Transaktionen gleichzeitig zu verarbeiten.**

7. **Der Lock Manager blockiert einen Teil einer Datenbanktabelle. Alle in Benutzung befindlichen Locks werden in einer Lock Datei gehalten,**

8. **Aus Performance Gründen befindet sich diese in der Regel im Hauptspeicher. In Zusammenarbeit mit dem Datenbanksystem stellt der Lock Manager die „Isolation“ der ACID Eigenschaft sicher.**

9. **Der Log Manager hält alle Änderungen gegen die Datenbank fest. Mit Hilfe der Log Datenbank kann der Recovery Manager im Fehlerfall den ursprünglichen Zustand der Datenbank wiederherstellen (Atomizität der ACID Eigenschaft).**

10. **In dem Repository verwaltet der TP Monitor Benutzerdaten und -rechte, Screens, Datenbanktabellen, Anwendungen sowie zurückliegende Versionen von Anwendungen und Prozeduren.**

7.4.6 Fehlerbehandlung

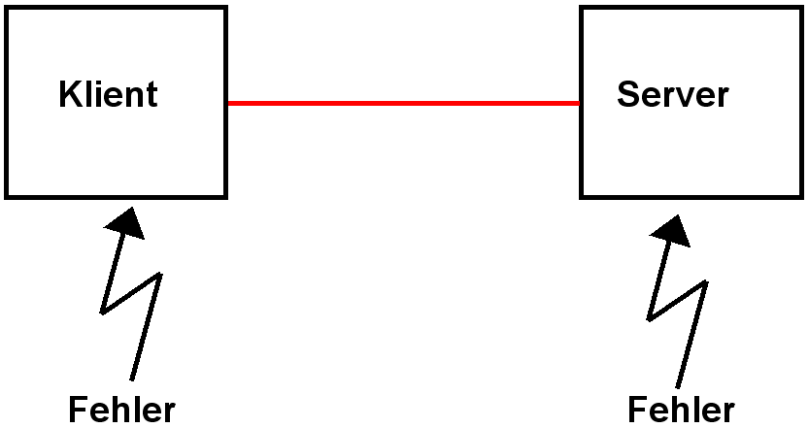


Abb. 7.4.7
Absturz beim Klienten oder beim Server

In einer Client/Server Konfiguration existieren drei grundsätzliche Fehlermöglichkeiten:

1. **Fehler in der Verbindung zwischen Client und Server**
2. **Server kann Prozedur nicht beenden (z.B. Rechnerausfall, SW-Fehler)**
3. **Client-Prozess wird abgebrochen ehe Antwort vom Server eintrifft. Er kann die Antwort des Servers nicht entgegennehmen.**

Fehler in der Verbindung zwischen Client und Server werden in der Regel von TCP in Schicht 4 automatisch behoben und sind deshalb unkritisch. Die beiden anderen Fehlermechanismen erfordern Fehlerbehandlungsmaßnahmen.

7.4.7 Ausfall des Servers oder des Klienten

1. Idempotente Arbeitsvorgänge

"Idempotent" sind Arbeitsvorgänge, die beliebig oft ausgeführt werden können; Beispiel: Lese Block 4 der Datei xyz. Nicht idempotent ist die Überweisung eines Geldbetrages von einem Konto auf ein anderes.

2. Nicht-idempotente Arbeitsvorgänge

Problem: Wie wird festgestellt, ob Arbeitsgang durchgeführt wurde oder nicht?

"Genau einmal" (exactly once).
Überweisung von Konto x nach y

"Höchstens einmal" (at most once).
Stimmabgabe Bundestagswahl

"Wenigstens einmal" (at least once). Ideal für idempotente Vorgänge.
Update Virenscanner

Transaktionen stellen eine „exactly once“ Semantik sicher.

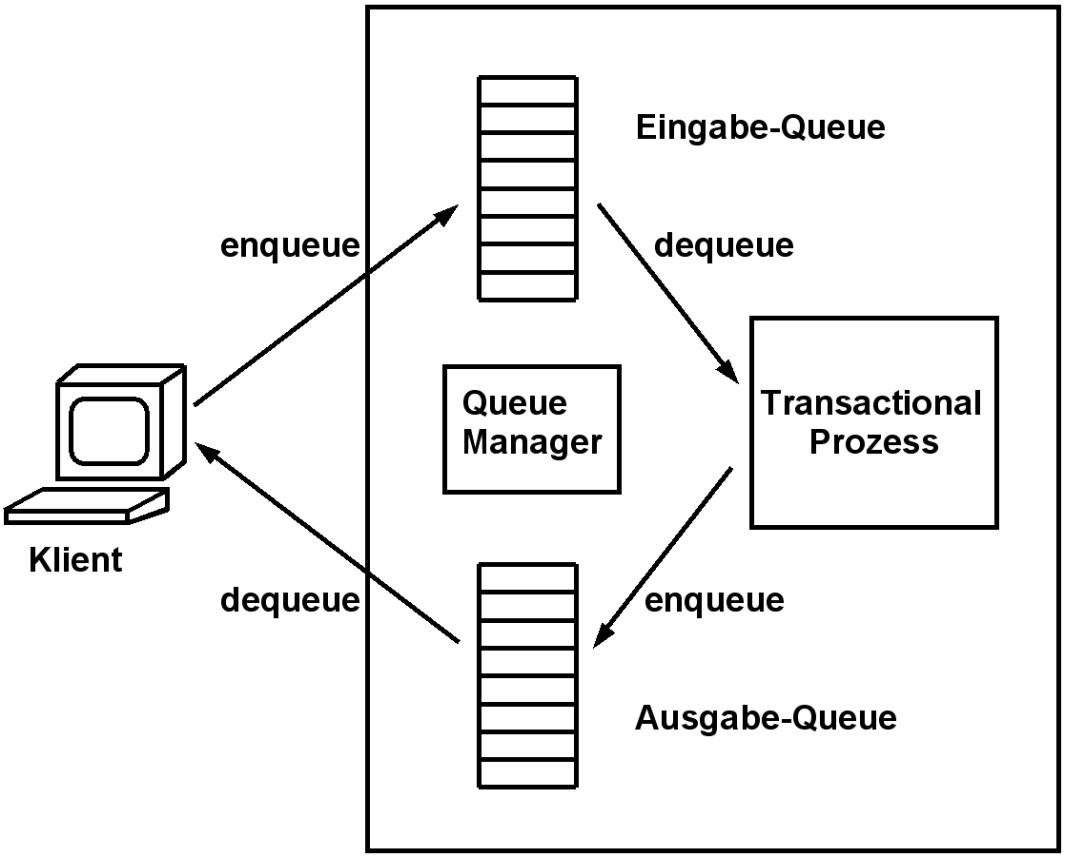


Abb. 7.4.8
Aufgaben der Warteschlangen

Jede einzelne Transaktion wird in 3 Subtransaktionen aufgelöst, die alle eigene ACID Eigenschaften haben.

1. **Subtransaktion 1 nimmt die Eingabenachricht entgegen, stellt sie in die Eingabe Queue, und commits.**
2. **Subtransaktion 2 dequeues die Nachricht, verarbeitet sie, stellt das Ergebnis in die Ausgabe-Queue, löscht den Eintrag in der Eingabequeue und commits.**
3. **Subtransaktion 3 übernimmt das Ergebnis von der Ausgabequeue, übergibt das Ergebnis an den Klienten, löscht den Eintrag in der Ausgabequeue und commits.**

Wenn der Klient abstürzt (Waise), stellt der Server das Ergebnis der Verarbeitung in die Ausgabe Queue. Die Transaktion ist damit vom Standpunkt des Servers abgeschlossen. Das Verarbeitungsergebnis kann beim Wiederanlauf des Klienten aus der Ausgabe-Queue abgeholt werden.

Getrennte Warteschlangen (Queues) für eingehende und ausgehende Nachrichten lösen viele Probleme. Die Queues sind auf einem RAID Plattenspeicher persistent gespeichert. Für den TP Monitor ist die Transaktion abgeschlossen, wenn die Ergebnisse in der Ausgabe Queue festgehalten wurden, auch wenn der Klient zwischenzeitlich ausgefallen ist.

Die Queues nehmen weiterhin eine Prioritätensteuerung und eine Lastverteilung auf mehrere Server vor.

7.4.8 Sicherheitssystem

Unter z/OS werden die Basis-Sicherheitserfordernisse durch den z/OS Security Server abgedeckt, z.B. mit Hilfe von RACF oder einer äquivalenten Software Komponente, z.B. ACS von der Firma Computer Associates. Diese ist für den Login-Process zuständig und regelt weiterhin Zugriffsrechte, z.B. auf bestimmte Daten.

In vielen Fällen ist es zusätzlich erforderlich, eine weitergehende Klienten-Autentifizierung durchzuführen und/oder Daten kryptografisch verschlüsselt zu übertragen. Hierfür werden Sicherheitsprotokolle wie SSL (Secure Socket Layer), TLS (Transport Layer Security), IPsec oder Kerberos eingesetzt.

Abb. 7.4.9
Herkules kämpft mit dem Höllenhund Kerberos

Kerberos, ursprünglich vom Massachusetts Institute of Technology (MIT) entwickelt, gehört zum z/OS Lieferumfang und ist in Mainframe Installationen weit verbreitet. Daneben werden SSL (TLS) und IPsec unter z/OS eingesetzt. Kerberos hat gegenüber SSL den Vorteil, dass ausschließlich symmetrische kryptografische Schlüssel einsetzt werden, was besonders bei kurzen Transaktionen die Performance verbessert.

Kerberos (Κέρβερος) ist der Name des dreiköpfigen Höllenhundes, der in der antiken griechischen Mythologie den Hades (die Unterwelt) bewacht.

7.4.9 Zustand von Prozessen

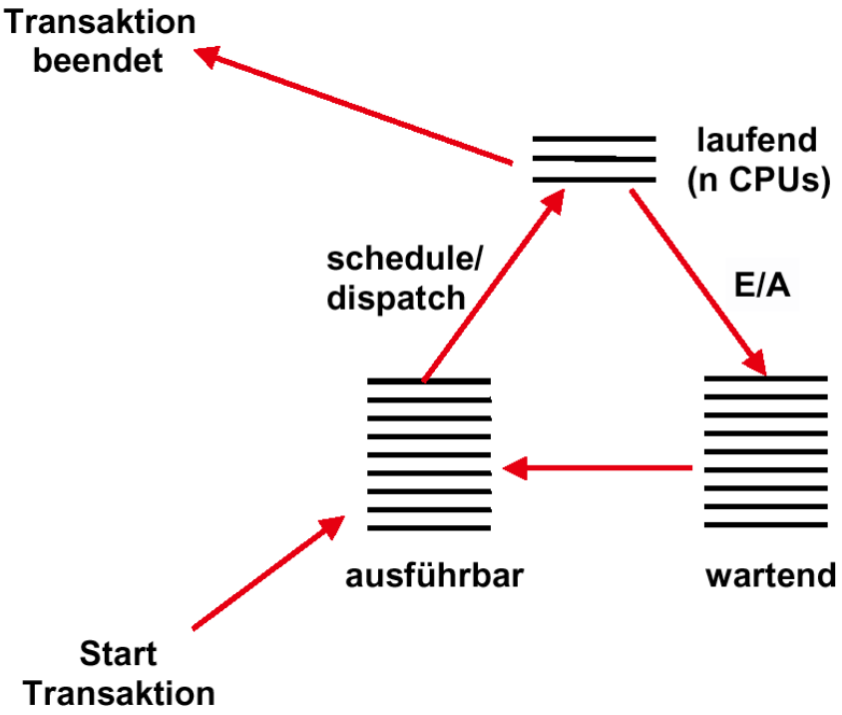


Abb. 7.4.10
Zustandswechsel für aktive Transaktionen

Jede Transaktion rotiert wiederholt durch die Zustände laufend, wartend und ausführbar. Der Scheduler des TP Monitors selektiert aus der Queue ausführbare Transaktionen jeweils einen Kandidaten wenn immer eine CPU frei wird.

In einem Rechner laufen immer zahlreiche Prozesse gleichzeitig ab. Die Prozesse befinden sich immer in einem von drei Zuständen. Hierfür werden drei Wartschlangen benutzt, in denen sich die TCBs der Prozesse befinden.

Ein Prozess befindet sich im Zustand laufend, wenn er von einer der verfügbaren CPUs ausgeführt wird. (Die allermeisten Mainframes sind Mehrfachrechner und verfügen über mehrere CPUs, bis zu 196 bei einem zEC12Rechner).

Ein Prozess befindet sich im Zustand wartend, wenn er auf den Abschluss einer I/O Operation wartet.

Ein Prozess befindet sich im Zustand ausführbar, wenn er darauf wartet, dass die Scheduler/Dispatcher Komponente ihn auf einer frei geworden CPU in den Zustand laufend versetzt.

7.4.10 TP Monitor Repository

Das Repository des TP Monitors speichert alle Daten, die für die interne Ablaufsteuerung benötigt werden. Andere Bezeichnungen sind Katalog oder Dictionary.

Im Repository werden katalogisiert:

- **Benutzer Rechte**
- **Workstation IDs (Terminal IDs)**
- **Screens**
- **Anwendungen**
- **Prozeduren**
- **Tabellen**
- **Rollen (Wer kann/darf/soll wann was machen)**
- ●
- ●
- ●
- ●

Z.B. Hinzufügen eines Feldes in einen Screen ändert Client Software, Server Schnittstelle, fügt eine weitere Spalte in eine SQL Tabelle ein.

7.4.11 Log File Konzept

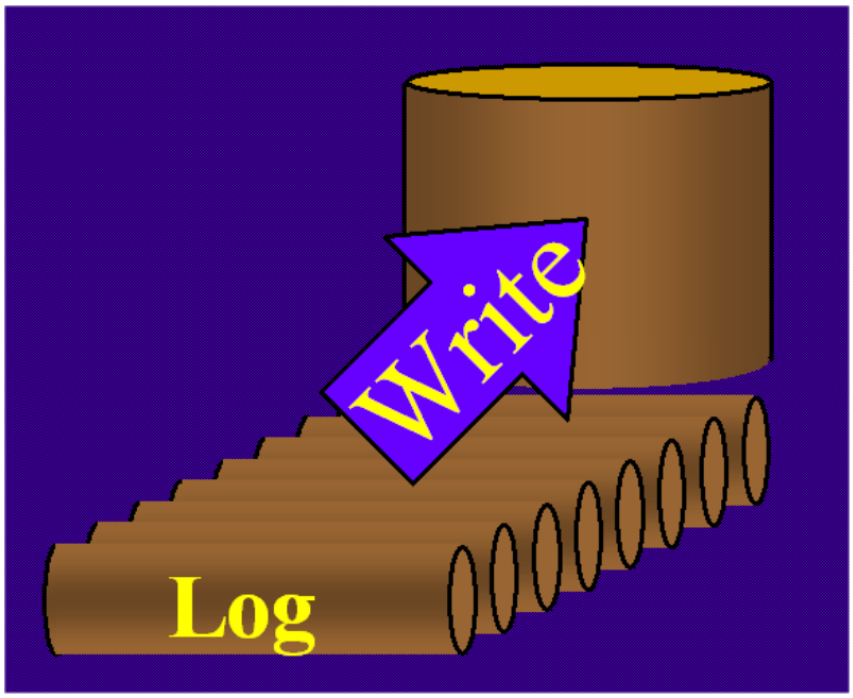


Abb. 7.4.11
Änderungen in der Datenbank werden in einem Log festgehalten

Das Log ist eine historische Aufzeichnung aller Änderungen des Zustands (state) des TP Systems. Log plus alter Zustand ergibt neuen Zustand. Das Log ist eine sequentielle Datei. Das vollständige Log enthält die historische Entwicklung aller Datenbankänderungen.

Beim „Commit“ einer Transaktion wird das Log persistent gespeichert (z.B. auf einer gespiegelten oder einer RAID Festplatte).

Das Log wird zur Wiederherstellung einer Transaktion im Falle eines Fehlers benutzt:

- **System failure: lost in-memory updates**
- **Media failure (lost disk)**

Bewirkt die Dauerhaftigkeit (Durability) einer Transaktion.

Es existiert ein statisches Log, welches erlaubt, alle historisch abgelaufenen Transaktionsabläufe wiederherzustellen (reconstruct), sowie ein dynamisches Log, dessen Einträge nach erfolgreicher Durchführung einer Transaktion wieder gelöscht werden können.

Im Fall, dass eine Datenbank zerstört wird, kann mit Hilfe einer (nicht den neuesten Stand enthaltenden) Backup Kopie sowie des statischen Logs die zerstörte Datenbank wieder hergestellt werden, indem man gegen die Backup Kopie alle seither stattgefundene Transaktionen laufen lässt.

7.4.12 Backward Recovery

Andere Bezeichnungen sind: backout, roll back oder abort.

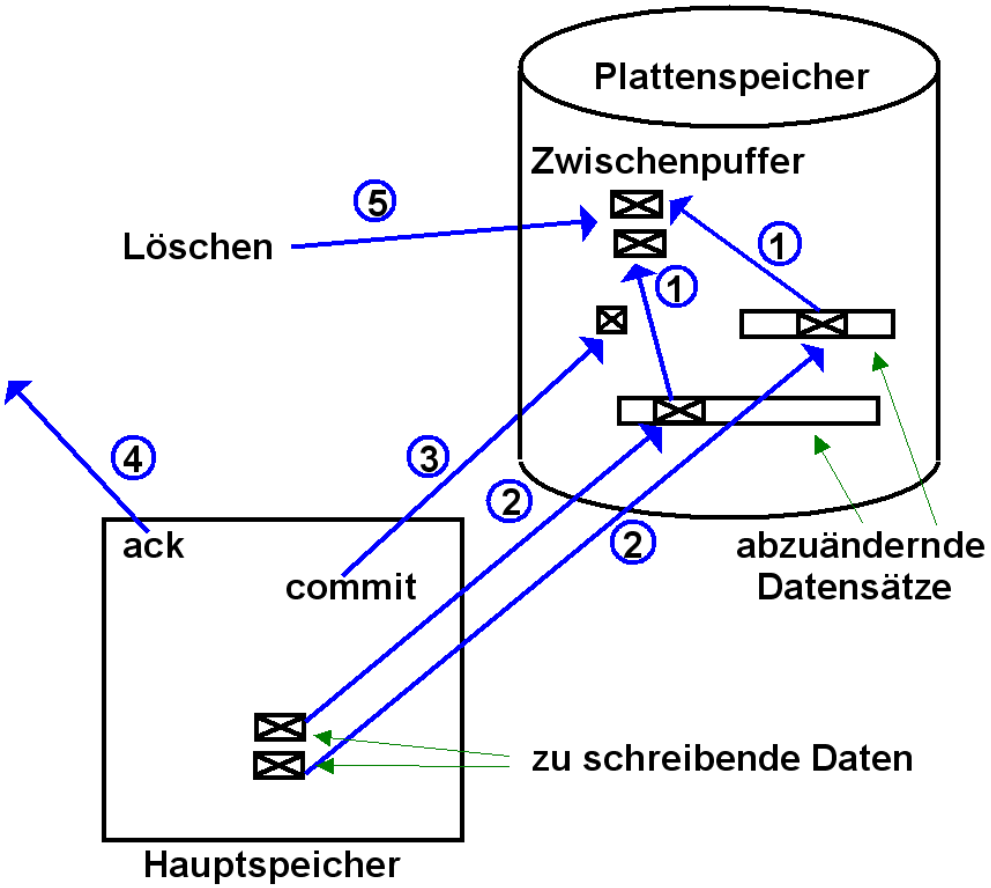


Abb. 7.4.12
Abändern von 2 Datensätzen

Zur Ermöglichung einer eventuellen Recovery finden bei jedem Write Vorgang die folgenden Schritte statt:

1. **abzuändernde Information in Puffer zwischenspeichern**
2. **Datensätze überschreiben**
3. **Commit Status festschreiben. Damit ist der ACID-relevante Teil der Transaktion abgeschlossen.**
4. **erfolgreiches Commit dem Benutzer mitteilen**
5. **Zwischenpuffer löschen**

Der Zwischenpuffer ist Teil eines dynamischen Logs.

Gewisse Daten werden von einem Transaction Processing Monitor als recoverable resource behandelt. Hierzu gehören:

- **Alle von Anwendungsprogrammen verwendeten Dateien und Datenbanken**
- **Gewisse transiente Daten und temporary storage queues**

Für diese Daten wird recovery information vom Transaction Monitor logged oder aufgezeichnet (recorded). Hierfür dient ein dynamisches Log. Wenn eine Transaktion erfolgreich abgeschlossen wird, wird die recovery Information in dem dynamischen Log gelöscht. Unabhängig davon existiert ein statisches Log, in dem alle transaktional vorgenommenen Datenänderungen durch erfolgreich durchgeführte Transaktionen festgehalten werden.

Wenn eine Transaktion nicht ausgeführt werden kann (z.B. Fehler im Anwendungsprogramm, im Datenzugriff oder aus anderen Gründen), dann führt der Transaction Monitor ein dynamic transaction backout (DTB) oder Rollback aus.

Dies macht alle bisherigen, vor dem Transaktionsabbruch durchgeführten Updates (Datenänderungen) rückgängig. DTB verhindert, dass corrupted Data von anderen Tasks oder Transaktionen benutzt werden.

Der Abbruch einer Transaktionsverarbeitung wird in der z/OS Terminologie als „abnormal end“, oder abgekürzt “abend“ bezeichnet.

7.4.13 Distributed Transaction

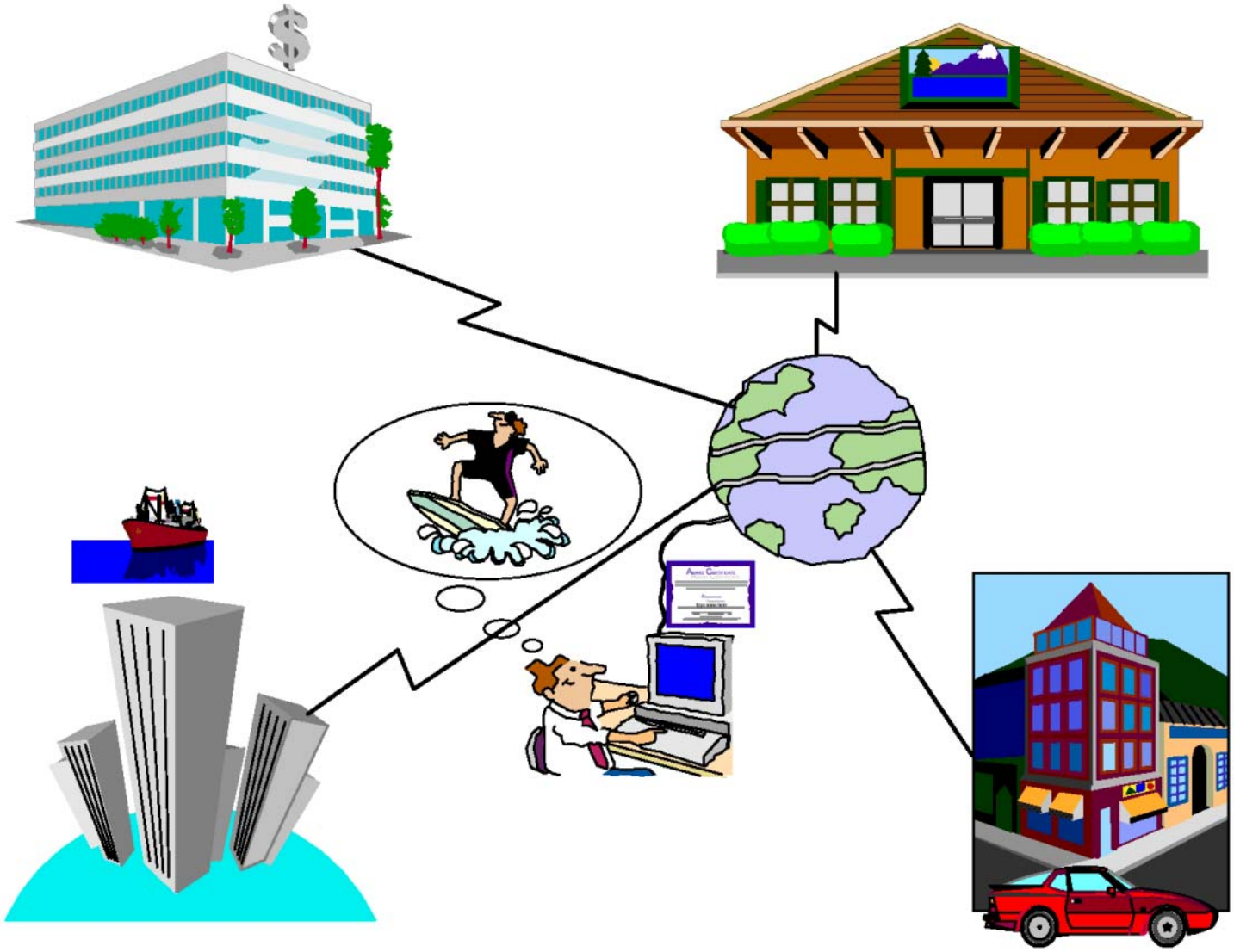


Abb. 7.4.13
Urlaubsbuchung mittels einer Distributed Transaction

Angenommen, ein Benutzer geht zu einem Reisebüro um dort eine Urlaubsreise zu buchen. Nachdem er den Flug, das Hotel und den Mietwagen selektiert hat, beschließt das Reisebüro, die Reise über das Internet zu buchen

Hierzu startet das Reisebüro eine Transaktion, welche

- **den Interessenten authentifiziert und in der Datenbank des Reisebüros speichert,**
- **die Flugreservierung im Datenbanksystem der Fluglinie aufzeichnet,**
- **die Raumreservierung im Computer des Hotels vornimmt,**
- **das Gleiche im Datenbanksystem der Autovermietung vornimmt,**
- **den gesamten Preis vom Bankkonto des Interessenten abbucht,**
- **Überweisungen auf die Bankkonten der anderen Teilnehmer der Transaktion vornimmt,**
- **für den Benutzer eine Multimedia File mit Information über das Paket an Reservierungen erstellt, und**
- **eine Bestätigung über die erfolgreiche Durchführung an den Interessenten sendet.**

Offensichtlich sind hier mehrere TP Monitore auf geographisch verstreuten Systemen involviert.

Wichtig ist, dass die gesamte Datenverarbeitung als eine atomare Transaktion durchgeführt wird. Der Interessent wird nicht zufrieden sein, wenn Hotel und Mietwagen gebucht wurden, das Geld vom Konto abgebucht wurde, aber der Flug wegen Überbuchung nicht verfügbar ist. Deshalb muss jeder Teil der Transaktion erfolgreich durchgeführt werden, oder die Transaktion insgesamt darf nicht ausgeführt werden.

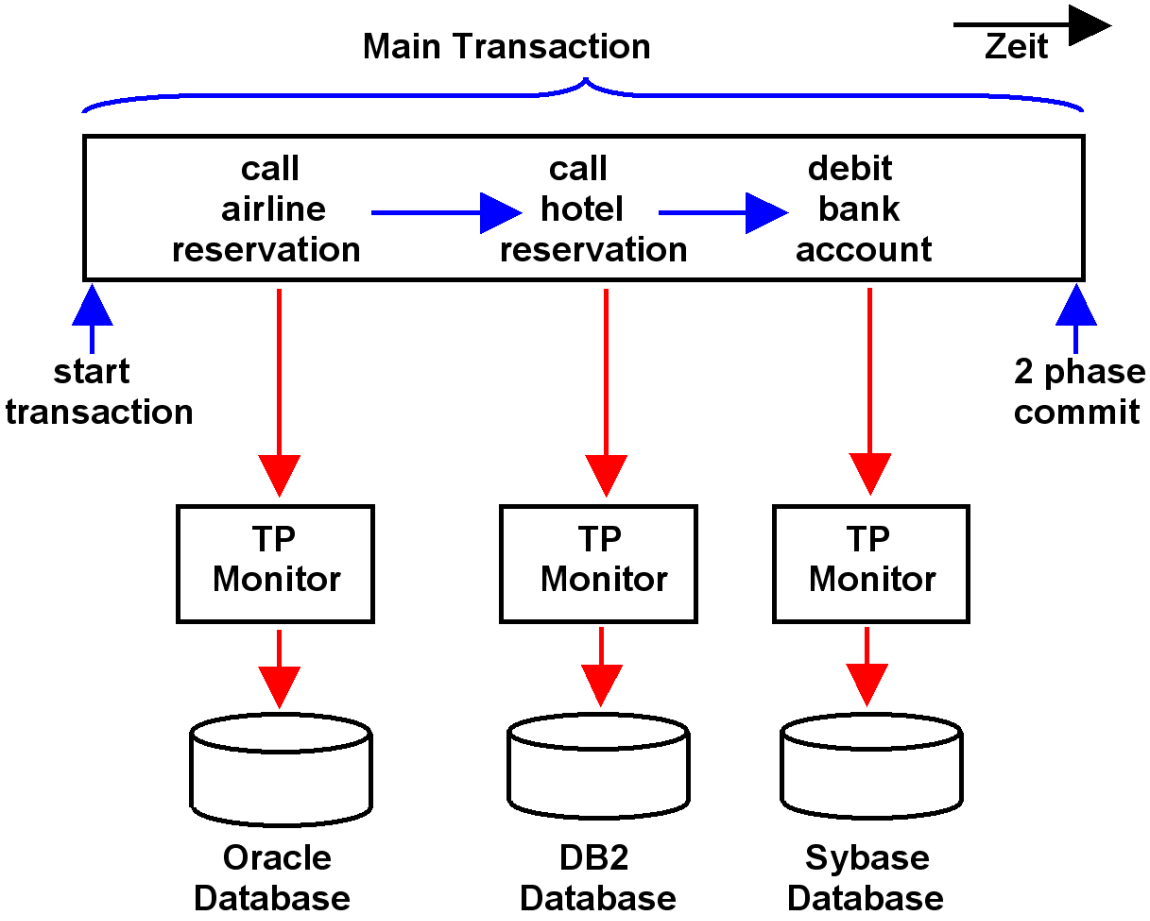


Abb. 7.4.14
Verteilte Transaktion auf mehreren Transaktionsmonitoren

In dem in Abb. 7.4.14 gezeigten Beispiel beinhaltet die Transaktion für die Urlaubsreservierung (Main Transaction) Aufrufe für drei unterschiedliche TP Monitore und drei verschiedene Datenbanken.

Dieser Fall wäre mit einer nested Transaktion lösbar. Nested Transaktionen werden in der Praxis aber nicht eingesetzt.

An deren Stelle tritt das Two-Phase Commit Protokoll, welches Bestandteil fast aller TP Monitore ist, besonders auch der unter z/OS verfügbaren TP Monitore.

Die Two-Phase Commit Management Komponente wird auch als Sync-Point Manager bezeichnet.

7.4.14 Two-phase Commit Protokoll

Das 2-Phase Commit Protokoll steuert die gleichzeitige Änderung mehrerer Datenbanken, z.B. bei der Urlaubsreise-Buchung die Änderung der Datenbanken der Fluggesellschaft, des Hotels und des Bankkontos von dem die Bezahlung der Reise abgebucht wird. Das Update der drei Datenbanken erfolgt durch unabhängige TP Monitore.

Ein Problem tritt auf, wenn eines der Updates nicht erfolgen kann, z.B. weil das Hotel ausgebucht ist. Daher sind atomare Transaktionen erforderlich

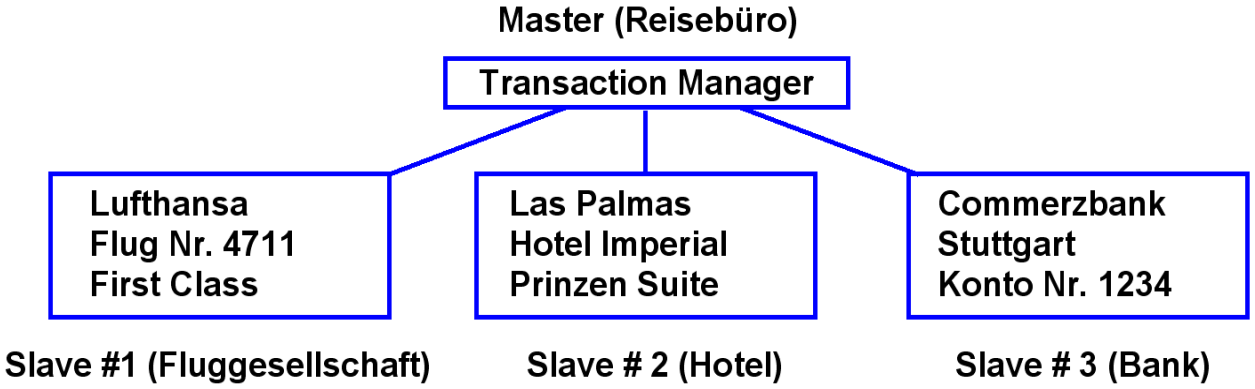


Abb. 7.4.
Master steuert Slaves

Die Konsistenz wird erreicht durch einen „Master“, der die Arbeit von „Slaves“ überwacht. Der TP Monitor des Reisebüros übernimmt die Rolle des Masters, die TP Monitore der Fluggesellschaft, des Hotels und der Bank sind die Slaves.

Der Master sendet Nachrichten an die drei Slaves. Jeder Slave markiert die Buchung als vorläufig und antwortet mit einer Bestätigung (Phase 1).

Wenn alle Slaves ok sagen sendet der Master eine Commit Nachricht, ansonsten eine Rollback Nachricht (Phase 2).

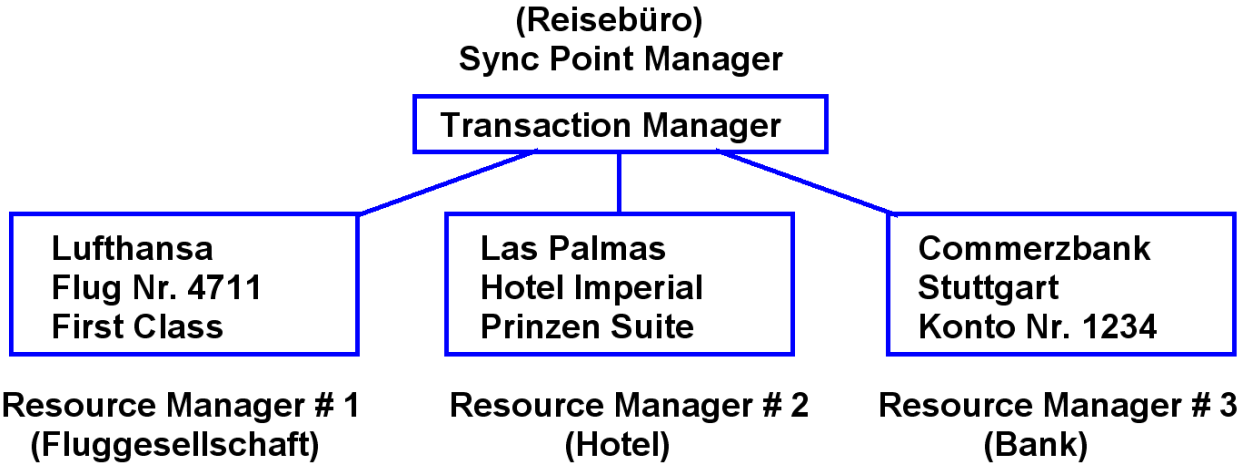


Abb. 7.4.
X/Open Terminologie

Der X/Open Standard verwendet eine andere Terminologie.

Der Master wird auch als Transaction Manager, Recovery Manager oder Sync Point Manager bezeichnet. Slaves werden als Resource Manager bezeichnet.

IBM bezeichnet den Master häufig als Sync Point Manager.

Master	Slave
Begin atomic action Send Request 1 ..... Send Request n Send „ Prepare to commit“	
	if action can be performed then begin Lock data Store initial state on disk Store requests on disk Send „OK“ end else Send „Failure“
Ende Phase 1	
if all slaves said „OK“ then send „Commit“ else send „Rollback“ Wait for acknowledgements	
	if master said commt then begin Do work Unlock data end Send „Acknowledgement“
Ende Phase 2	

Abb. 7.4.
2-Phase Commit Code Snipplet

Der Master fragt bei allen beteiligten Slaves an, ob die gewünschte Aktion machbar ist. Jeder Slave markiert die Anforderung als vorläufig und schickt eine Bestätigung an den Master (Phase 1).

Wenn alle Slaves positiv antworten sendet der Master eine commit Aufforderung, worauf die Slaves die vorläufige Änderung permanent machen. Wenn einer der Slaves seine Transaktion nicht durchführen kann sendet der Master an die Slaves eine Nachricht die vorläufige Änderung wieder rückgängig zu machen (Phase 2).

Alle Slaves bestätigen den Abschluss der 2-Phase Commit Transaktion.

In Phase 1 übergibt das Anwendungsprogramm die 2-Phase Aufforderung an den Syncpoint Manager. Dieser sendet einen PREPARE-Befehl an allen Ressource-Manager. In Reaktion auf den Befehl PREPARE, antwortet jeder an der Transaktion beteiligte Ressource-Manager an den Syncpoint Koordinator er ist bereit oder auch nicht.

Wenn der Syncpoint Manager Antworten von allen Resource Managern erhalten hat wird die Phase 2 eingeleitet. Der Syncpoint Manager sendet einen Commit oder Rollback-Befehl auf der Grundlage der vorherigen Antworten. Wenn auch nur einer der Resource Manager eine negativen Antwort gesendet hat, veranlasst der Syncpoint Manager ein Rollback der vorläufigen Änderungen in allen Resource Managern.

CICS enthält einen eigenen Syncpoint Manager, der mit dem EXEC CICS SYNCPOINT Command aufgerufen wird. Daneben und alternativ kann CICS einen generischen Recovery Manager „z/OS Resource Recovery Services“ (RRS), der von z/OS Resource Managern wie WebSphere, IMS, DB2, aber auch von CICS (an Stelle des CICS Syncpoint Managers) benutzt werden kann.

7.4.15 z/OS Resource Recovery Services

Ein Prozess, der transaktionale Anwendungen ausführt, wird als Resource Manager bezeichnet. Unter z/OS können mehrere Resource Manager gleichzeitig tätig sein. Beispiele sind die CICS, IMS und WebSphere Transaktionsmonitore, sowie Stored Procedures mehrerer Datenbanken wie IMS und DB2.

z/OS Resource Recovery Services (RRS) stellt einen Recovery Manager zur Verfügung, der das Two-Phase Commit Protokoll beinhaltet, und den jeder Resource Manager innerhalb von z/OS nutzen kann. Es ermöglicht Transaktionen, ein Update geschützter (protected) Resourcen vorzunehmen, die von unterschiedlichen Resource Managern (z.B. unterschiedlichen TP Monitoren) verwaltet werden.

RRS wird von neuen Resource Managern eingesetzt, sowie für die Nutzung neuartiger Funktionen durch existierende Resource Managers. Neu entwickelte Resource Manager Produkte unter z/OS (z.B. WebSphere Enterprise Java Beans) verwenden RRS, an Stelle einer nicht vorhandenen eigenen Two-phase Commit Protocol Komponente.

Existierende Tansaction Server wie CICS verfügen bereits über viele Funktionen, die in RRS enthalten sind, können stattdessen aber auch RRS benutzen. Vermutlich wird sich RRS in der Zukunft zu einer zentralen z/OS Komponente für die Transaktionssteuerung entwickeln.

Ursprünglich besaß jede Komponente von z/OS, die Transaktionen ausführen konnte (z.B. CICS, IMS/DC, andere) ihren eigenen Two-Phase Commit Recovery Manager. Dieser kann durch RRS ersetzt werden.

http://www.redbooks.ibm.com/abstracts/sg246980.html

7.5 Weiterführende Information

7.5.1 Die wichtigste Literatur zum Thema Transaktionen

J. Gray, A. Reuter:
"Transaction Processing".
Morgan Kaufmann, 1993.

fast 20 Jahre alt – immer noch das Standard Werk - 1070 Seiten !

P. A. Bernstein:
"Principles of Transaction Processing".
Morgan Kaufmann, 1997.

Wesentlich kürzer

J. Horswill:
"Designing and Programming CICS Applications". O'Reilly, 2000

Das beste CICS-spezifische Lehrbuch

Mark Little, Jon Maron, Greg Pavlik:
Java Transaction Processing :
Design and Implementation
Prentice Hall 2004, ISBN 0-13-035290-X

Transaktionale Anwendungen werden in Zukunft in Java entwickelt - vielleicht

7.5.2 Weitere Informationen

Das Internet enthält Tonnen von Material zu Themen wie Datenbanken, Stored Procedures und Transactionsverarbeitung.

Eine DB2 Tutorial Introduction ist verfügbar unter

http://www.youtube.com/watch?v=6noAMY3PcB8

Das folgende Video

http://www.youtube.com/watch?v=LwhkvIdombM

enthält eine detaillierte DB2 Installatonsanweisung

Videos der „Independent DB2 Users Group (idug) sind hier zu finden

http://www.idug.org/p/cm/ld/fid=85

und hier

http://www.idug.org/p/cm/ld/fid=86

Ein DB2 Stored Procedures Tutorial ist verfügbar unter

http://solutions.devx.com/ibm/skillbuilding/archives/introduction-to-db2-stored-procedures.html

oder sehr detailliert in

http://publib.boulder.ibm.com/infocenter/idm/v2r1/index.jsp?topic=/com.ibm.datatools.routines.tutorial.doc/topics/routines_introduction.html

Eine Einführung in die Transactionsverarbeitung ist hier zu finden

http://www.codeproject.com/Articles/522039/A-Beginners-Tutorial-for-Understanding-Transaction

Herr Kolja Treutlein, ehem. Student an der Uni Tübingen, erstellte ein DB2 Stored Procedures Tutorial im Rahmen seiner Diplom Arbeit. Download unter

http://www-ti.informatik.uni-tuebingen.de/~spruth/DiplArb/Treutlein.pdf

Eine Einführung in Two Phase Commit ist hier zu finden

http://docs.mongodb.org/manual/tutorial/perform-two-phase-commits/

8. CICS Transaktionsserver

8.1 CICS Übersicht

8.1.1 Implementierung eines Transaktionsmonitors

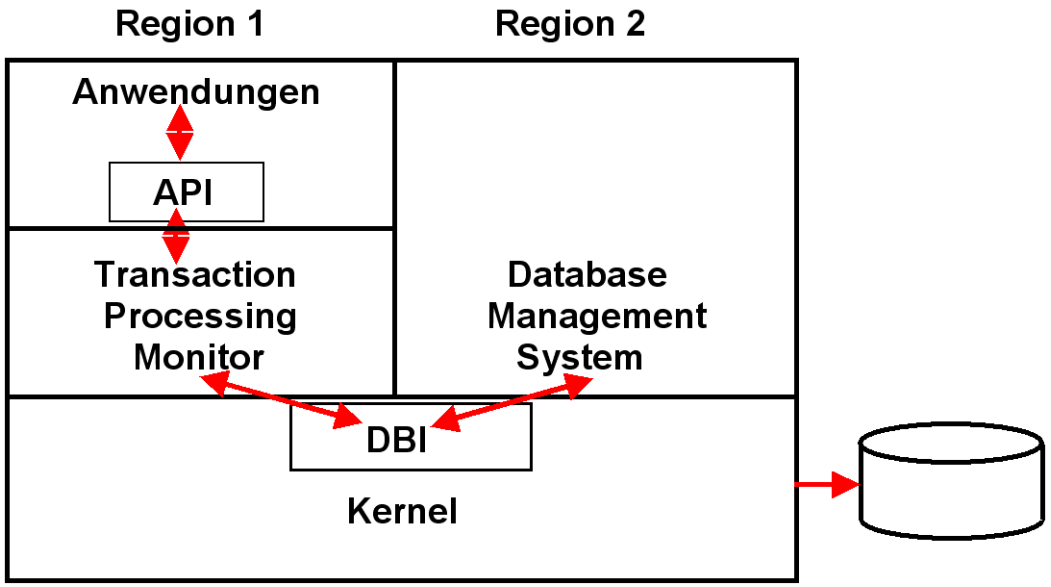


Abb. 8.1.1
Alternative 1: TP Monitor und Datenbanksystem sind getrennte Subsysteme

Unter z/OS laufen der Transaction Processing Monitor (TP Monitor, Transaction Server) und das Datenbank-System als unabhängige Anwendungsprozesse in getrennten virtuellen Adressenräumen unter dem Betriebssystem-Kernel. Der TP Monitor unterstützt sowohl Stapelverarbeitungs- als auch interaktive Transaktionen.

TP Monitor und Datenbanksystem kommunizieren über eine Kernel Schnittstelle (Datenbank Interface, DBI) miteinander.

Der Aufruf von Kernel Funktionen ist sehr aufwendig. Deshalb vermeidet der TP-Monitor nach Möglichkeit die Nutzung der Betriebssystem-Funktionen. Um Leistung und Durchsatz zu optimieren, kann er z.B. eigene Message Behandlungs- und Queuing-Einrichtungen enthalten.

8.1.2 Transaction Processing Facility

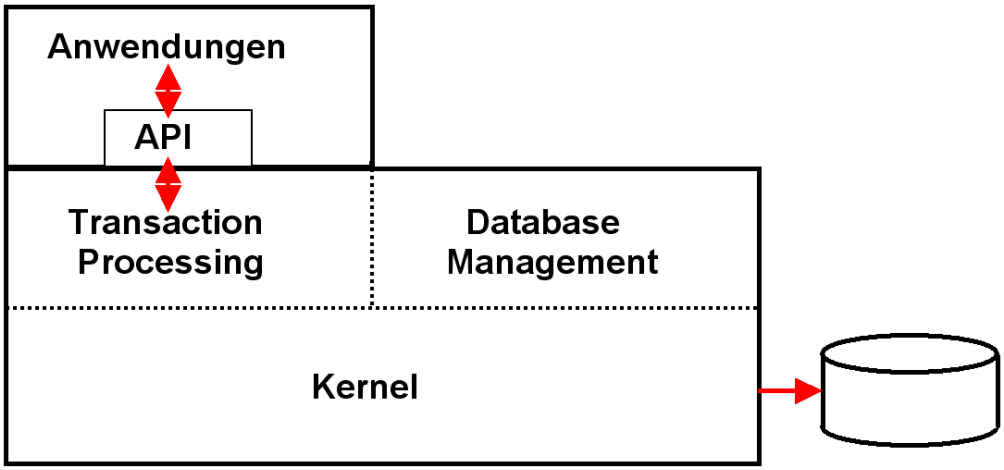


Abb. 8.1.2
Alternative 2: TP Monitor und Datenbanksystem sind Bestandteil des Kernels

Es wäre denkbar, die TP-Monitor- und Datenbank Funktionen in das Betriebssystem einzubauen. Dies ist z.B. beim Betriebssystem *Guardian* von HP/Compaq/Tandem und beim IBM-Betriebssystem TPF der Fall. TP Monitor und Datenbanksystem laufen im Kernel Status, evtl. auch die Anwendungen.

Dies bedingt sehr große Sorgfalt beim Schreiben des Codes. Durch Vermeidung des Wechsels zwischen User Status und Kernel Status (Problem and Supervisor State) ist ein signifikanter Leistungsgewinn möglich.

TPF *(Transaction Processing Facility)* ist ein eigenständiges Mainframe Betriebssystem, das nur eine einzige Anwendung erlaubt, nämlich die Transaktionsverarbeitung. Bei TPF laufen Anwendungen, Transaction-Monitor und Überwacher gemeinsam im Überwacher-Status.

Zu den Eigenschaften von TPF gehören:

- **Keine Einrichtungen für Softwareentwicklung (erfolgt auf einem anderen Rechner)**
- **Run-to-Completion (non-preemptive) Scheduler/Dispatcher**
- **Betriebssystem Aufruf erfordert ca. 500 Maschinenbefehle**
- **E/A Operation erfordert etwa 500 - 800 Maschinenbefehle**
- **>10 000 Transaktionen/s**

TPF wird eingesetzt, wenn besonders hohe Raten von relativ einfachen Transaktionen auftreten. Beispiele hiefür bilden Systeme für die Flugplatzreservierung, Geldausgabeautomaten und Kreditkartenverifizierung.

TPF ist aus dem Saber-Flugplatz-Reservierungssystem hervorgegangen, das 1959 gemeinsam von American Airlines und IBM entwickelt wurde. Später ist es in ACP und dann in TPF umbenannt worden.

Marriott Hotels, eine der großen internationalen Hotelketten, betreibt ein heterogenes Rechenzentrum. Es enthält ein zEnterprise 196 und ein System z10 Mainframe in dem primären Rechenzentrum und ein System z10 Mainframe-Backup-System in der remote Disaster Recovery Site. Darüber hinaus enthält die Marriott IT-Infrastruktur eine Mischung aus kleinen und Midrange-Systemen, die das gesamte Spektrum der Windows-, Linux-und UNIX-Betriebssysteme beinhalten. "Wir betreiben auch virtualisierte Ressourcen und evaluieren derzeitig die Installation einer zBX zEnterprise BladeCenter Extension".

"Unser zEnterprise System verwendet ein TPF-Betriebssystem, das wir im Laufe der Jahre für unsere Reservierungs-Verarbeitung optimiert haben. Wir glauben, dass dies uns einen strategischen Wettbewerbsvorteil" sichert“.

Mainframe Executive, Sept/Oct 2011.

8.1.3 Customer Information Control System (CICS)

CICS (ausgesprochen kiks, manchmal auch ziks) ist der am weitesten verbreitete, IBM proprietäre Transaktionsmonitor der Firma IBM. Die offizielle Bezeichnung ist:
CICS Transaction Server.

CICS ist unter den System z-Betriebssystemen z/OS und z/VSE, sowie in modifizierter Form (als Encina Erweiterung) unter AIX, HP-UX, Solaris sowie Windows Server verfügbar.

CICS hat eine Spitzenposition bezüglich Durchsatz, Zuverlässigkeit und Verfügbarkeit.

Mit CICS werden weltweit mehr als 30 Milliarden Transaktionen pro Tag verarbeitet.

Pro Tag wird mit CICS Transaktionen weltweit ein Geldbetrag von > 1 Billion (10^{12}) Dollar transferiert. Mehr als 30 Millionen Sachbearbeiter benutzen CICS bei ihrer täglichen Arbeit. 900 000 Benutzer können gleichzeitig auf einem CICS Rechner arbeiten.

Seit einigen Jahren benutzt IBM die offizielle Bezeichnung CICS Transaction Server oder abgekürzt CICS TS. In der Umgangssprache benutzt man weiterhin den Begriff CICS TP Monitor.

Derzeitig (2013) ist die Version CICS TS 5.1 verfügbar. Weit verbreitet ist noch Version CICS TS 3.2. Auf unserem Universitätsrechner ist beides installiert.

„TX Series for Multiplatforms“ ist eine CICS Implementierung für Unix, Linux und Windows, und ist weitestgehend mit der z/OS CICS Version kompatibel.

http://www.cedix.de/Literature/Textbooks/CicsIntro.pdf

Eine ganze Reihe von CICS Anwendungsprogrammen sind vor Jahrzehnten geschrieben worden und entsprechen in ihrer Struktur nicht mehr modernen Software Engineering Anforderungen.

Es hat immer wieder Überlegungen gegeben, diese in ihrer Struktur veralteten Programme umzuschreiben. Dies geschieht in der Praxis so gut wie gar nicht.

In der Weltwirtschaft sind weder das Geld und erst recht nicht die erforderliche Anzahl von Programmierern vorhanden um existierende Anwendungen ohne Not umzuschreiben. Vorhandene Ressourcen werden dringend für anstehende Aufgaben benötigt.

Die vorhandenen Anwendungen arbeiten zuverlässig und performant. In vielen Fällen fehlen Glaube und Überzeugung, dass das Neuschreiben existierender Anwendungen eine messbare Verbesserung bringen würde.

Frage:

Schätzen Sie bitte, wie groß die weltweiten Investitionen aller derzeitig in Betrieb befindlichen CICS Anwendungen sind

- **10 000 Mannjahre ?**
- **10 Millionen Mannjahre ?**
- **10 Milliarden Mannjahre ?**

Antwort:

10 Millionen Mannjahre dürften etwa richtig sein. Angenommen Entwicklungskosten von 100 000 $/Mannjahr ergibt eine Investition von 1 Billion $ in für Benutzer geschriebener CICS Anwendungssoftware.

- **20 000 System z Servers haben durchschnittlich 1 Mill. Zeilen aktiven CICS Anwendungscode (zwischen 200 000 und 50 Millionen pro Server), kumulativ 20 Milliarden Lines of Code (LOC).**
- **Bei einer Produktivität von 2 000 LOC/Mannjahr ergibt sich eine Investition von 10 Millionen Mannjahren.**
- **Angenommen 100 000 $/Mannjahr ergibt eine Investition von 1 Billion $ (10^{12} $) in CICS Anwendungssoftware. Zum Vergleich, das USA 1999 GNP war 9 Billion $.**

10 Milliarden Mannjahre können nicht richtig sein. Bei 100 000 $ / Mannjahr wären das etwa 1000 Billionen $. So viel Geld gibt es auf dem Planeten nicht.

There are approximately 950 000 CICS programmers worldwide.

http://www-3.ibm.com/developer/solutionsevent/pdfs/spector_lunchtime_keynote.pdf

http://www.cedix.de/Literature/Textbooks/CicsIntro02.pdf

http://www.cedix.de/VorlesMirror/Band1/Unicom.pdf

8.1.4 Multiprogrammierte Verarbeitung von Transaktionen

Ein Hochleistungs-Transaktionssystem verarbeitet in jedem Augenblick Hunderte oder Tausende von Transaktionen gleichzeitig und parallel. Pro CPU ist typischerweise nur eine Transaktion laufend, die anderen sind ausführbar oder warten auf den Abschluss einer Ein-/Ausgabeoperation. Hunderte oder Tausende paralleler Transaktionen sind denkbar.

Im Regelfall setzt der TP-Monitor als ein getrennter Anwendungsprozess auf dem Betriebssystem auf.

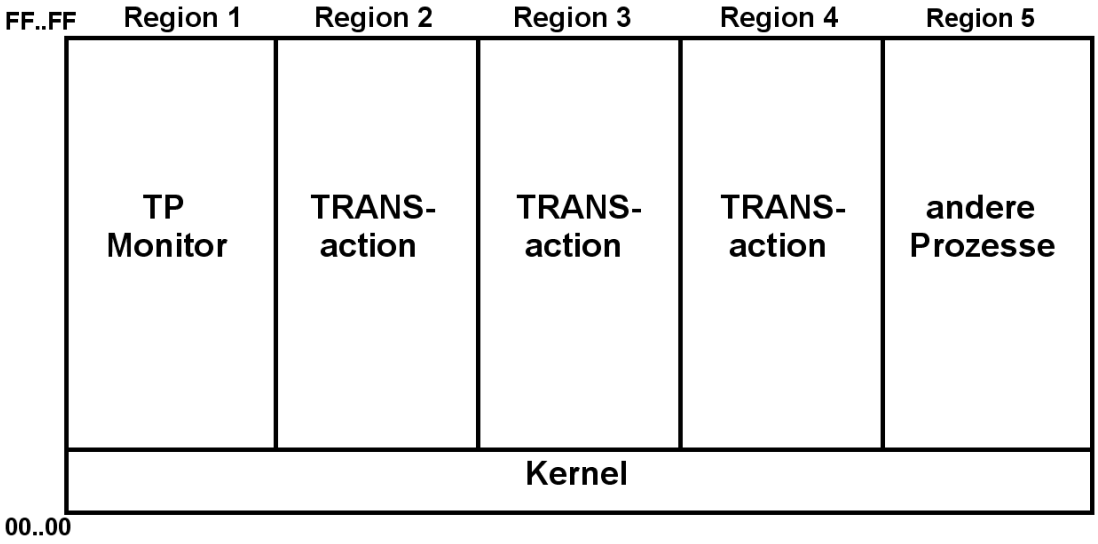


Abb. 8.1.3
Prozess-Ansatz

Für die Transaktions-Verarbeitung gibt es zwei Alternativen:

Beim Prozess-Ansatz läuft jede Transaktion als selbständiger Prozess in einem eigenen virtuellen Adressenraum (z/OS Region). Vorteil: hervorragende Isolation.

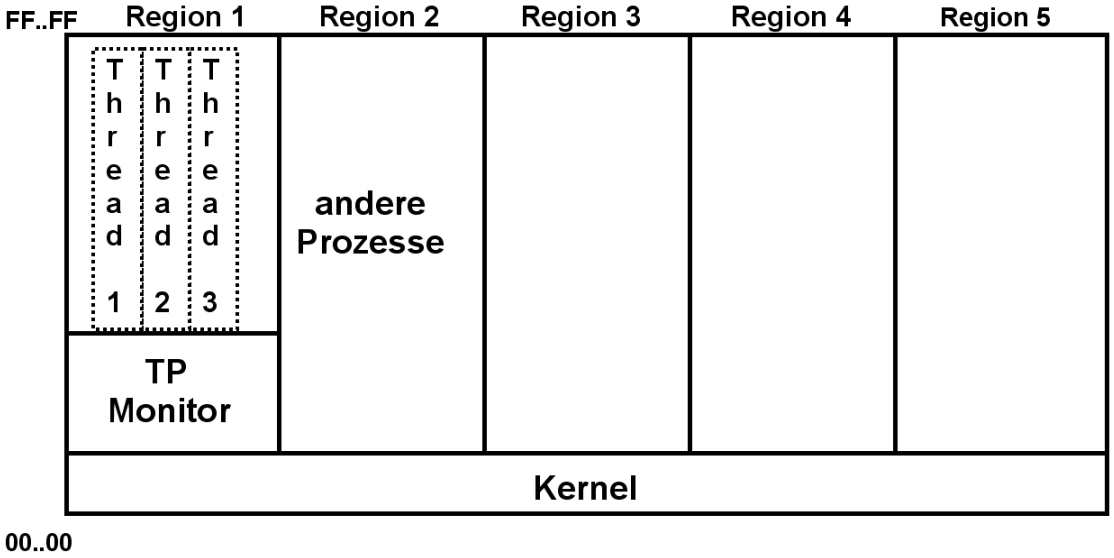


Abb. 8.1.4
Thread-Ansatz

Beim Thread-Ansatz laufen alle Transaktionen als Threads gemeinsam mit dem TP Monitor in einem einzigen virtuellen Adressenraum. Vorteil: Leistungsverhalten.

1 Prozess pro Transaktion bedeutet, dass jede Transaktion in einem eigenen virtuellen Adressenraum läuft. Vorteilhaft ist, dass die Isolation der Transaktionen untereinander (das i in ACID) optimal gewährleistet ist. Nachteilig ist der höhere Verarbeitungs-Aufwand im Vergleich zum Thread Ansatz.

Die z/OS Version von CICS benutzt einen Tread-ähnlichen Ansatz. Der Transactions-Monitor und alle CICS Anwendungen laufen im gleichen virtuellen Adressenraum (CICS Region). CICS Struktur-Eigenschaften gewährleisen die Isolation der Threads untereinander und gegenüber dem TP Monitor.

Operationen über Adressraumgrenzen hinweg benötigen viele Mikrosekunden. Operationen im gleichen Adressraum benötigen Mikrosekunden-Bruchteile. Ein Faktor 100 Geschwindigkeitsunterschied ist möglich. Deshalb vermeidet der CICS TP-Monitor nach Möglichkeit die Nutzung der Betriebssystem-Funktionen.

Dazu läuft der vollständige CICS Transaktionsmonitor als eine z/OS Anwendung im Problemstatus. Gleichzeitig stellt CICS eine Laufzeitumgebung für zahlreiche Anwendungen (hier Transaktionen) zur Verfügung. Eine derzeitige Laufzeitumgebung wird als „Middleware“ bezeichnet. Andere Middleware Beispiele sind:

- **Web Application Server**
- **Message Based Queuing**
- **Corba**
- **RMI**
- **Remote Procedure Call Middleware, z.B. DCE**

8.1.5 Rolle der Exec CICS Kommandos

Bei allen transaktionskritischen Aktivitäten rufen CICS Anwendungsprogramme den Betriebssystem-Kernel nie direkt auf. Stattdessen führt ein CICS Anwendungsprogramm spezielle EXEC CICS Kommandos aus, um Betriebssystem-Funktionen zu bewirken wie:

- **Dateizugriffe,**
- **Klienten-Kommunikation**
- **Prozesswechsel oder**
- **Funktionen, die System-Ressourcen erfordern.**

EXEC CICS Kommandos werden vom „CICS Nucleus“ im User Status ausgeführt. Beispielsweise kann ein EXEC CICS WRITE Kommando die Funktion mehrerer System Calls beinhalten:

- **Zugriff auf die Lock Datenbank**
- **Zugriff auf die Log Datenbank**
- **Speicherung der Daten**

Der CICS TS Transaktionsmonitor verhält sich wie ein Mini-Betriebssystem unterhalb des tatsächlichen Betriebssystems. CICS stellt damit eine Laufzeitumgebung für den Ablauf von CICS Transaktionen zur Verfügung.

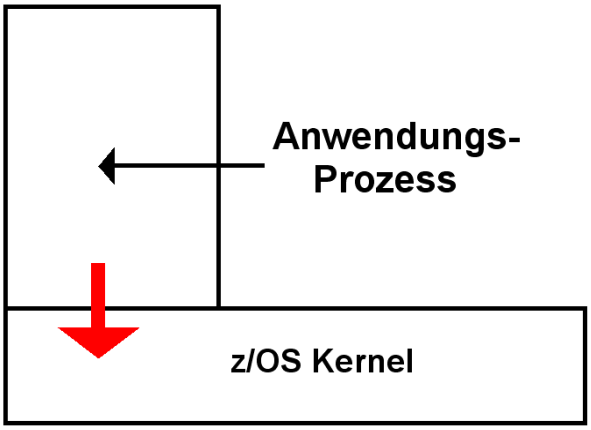


Abb. 8.1.5
System Calls

Ein normaler Anwendungsprozess nimmt Dienstleistungen des Kernels über System Calls wie z.B. OPEN oder READ in Anspruch.

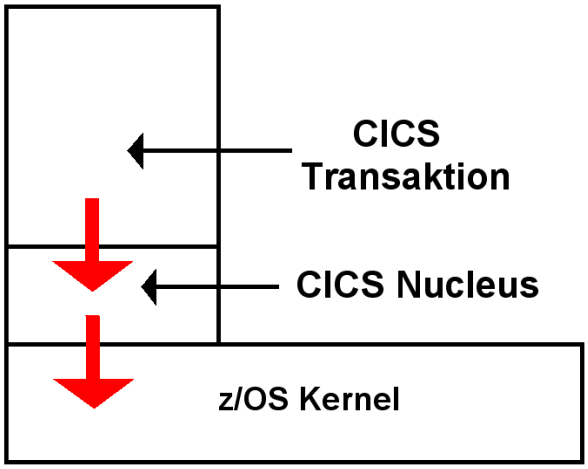


Abb. 8.1.6
EXEC CICS Kommandos

Eine CICS Transaktion verwendet statt dessen EXEC CICS Kommandos, die Dienstleistungen einer speziellen CICS Komponente (CICS Nucleus) in Anspruch nehmen.

Der CICS Nucleus ruft bei Bedarf den Kernel über normale System Calls auf.

Alle laufenden CICS Anwendungsprogramme befinden sich zusammen mit dem CICS Nucleus in einem einzigen virtuellen Adressenraum.

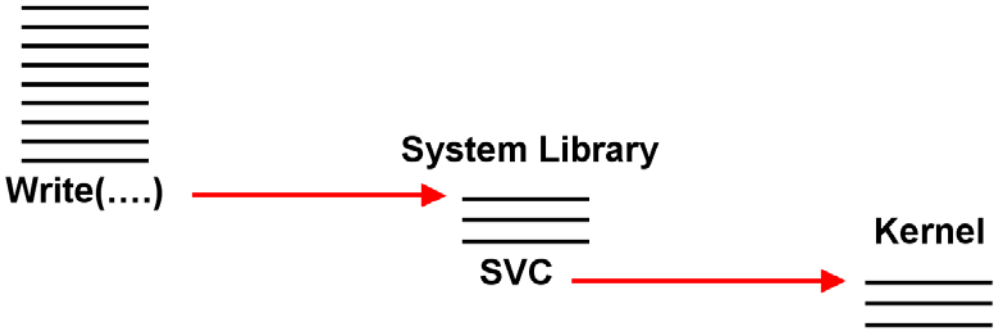


Abb. 8.1.7
Normale Schreiboperation

Eine normale Write Operation führt zum Aufruf einer Library Routine, die in den Kernel mittels eines SVC Maschinenbefehls verzweigt und diese ausführt.

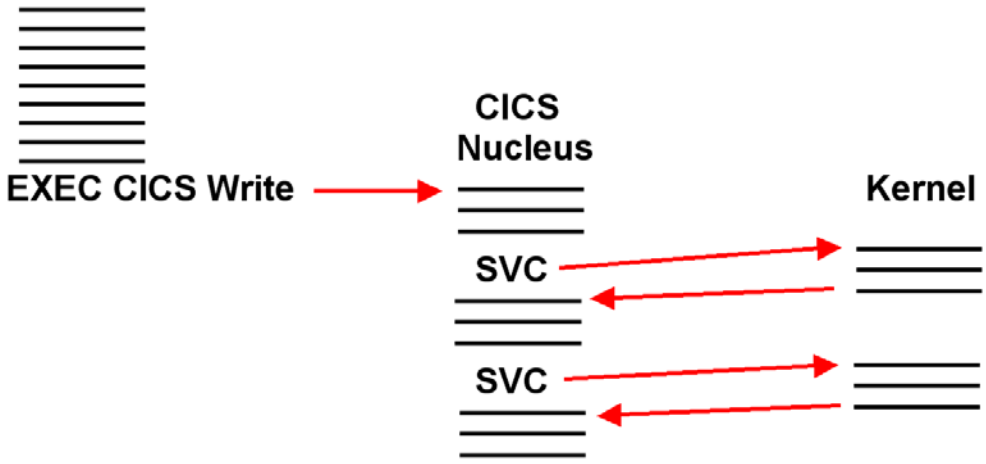


Abb. 8.1.8
CICS Schreiboperation

CICS ersetzt die Write Operation durch eine EXEC CICS Write Operation. Diese führt zum Aufruf einer Routine des CICS Nucleus, welche die ACID Eigenschaften sicherstellt, indem u.A. Einträge in eine LOG Datei und eine Lock Datei erfolgen, Voraussetzungen für ein evtl. Rollback geschaffen werden, usw.

Anders als reguläre Programme führen CICS Anwendungsprogramme bei transaktionskritischen Aktivitäten (Einhaltung der ACID Bedingungen) keine direkten Aufrufe des Betriebssystems aus. Stattdessen enthalten CICS Anwendungsprogramme Aufrufe des CICS Nucleus um Funktionen wie Terminal I/O, File I/O, Program Control usw. auszuführen, welche Kernel Funktionen erfordern.

CICS TS verhält sich wie ein Mini-Betriebssystemkernel unterhalb des tatsächlichen Betriebssystems um eine Umgebung für die Ausführung von Anwendungsprogrammen bereitzustellen.

Aufrufe des CICS Nucleus fangen immer mit der Sequenz EXEC CICS an, und hören mit dem Statement Delimiter der Programmiersprache auf, in der das ECEC CICS Statement eingebettet ist, z. B ; in C++ oder END in Cobol.

Beispiel für einen Aufruf des CICS Nucleus innerhalb eines C++-Programms:

```
EXEC CICS SEND MAP(“label04”) MAPSET(“s04set”) ERASE;
```

Ein Bildschirminhalt (eine Map) mit dem Namen label04, welche zu einer Gruppe ähnlicher Maps (einem Mapset mit dem Namens04set) gehört, wird an einen Klientenrechner gesendet.

Beispiel für einen Aufruf des CICS Nucleus innerhalb eines COBOL-Programms

```
EXEC CICS
WRITEQ TS QUEUE(‘ACCTLOG’) FROM(ACCTDTLO)
LENGTH(DTL-LNG)
END EXEC
```

Ein existierender Datensatz „ACCTDTLO“ wird in eine temporäre Warteschlange ACCTLOG geschrieben, die als Log zur Datensicherung dient

8.1.6 Überblick über die CICS Befehle

Bildschirm

SEND MAP
SEND CONTROL
SEND
RECEIVE MAP

Programmsteuerung

LINK
RETURN
XCTL
LOAD
RELEASE

Zwischenspeichern

WRITEQTS (Temporary Storage)
READQTS
DELETEQTS
WRITEQTD (Transient Data)
READQTD
DELETEQTD

Zeitsteuerung

ASKTIME
FORMATTIME
START
RETRieVE

Systemsteuerung

ADDRESS
ASSIGN

VSAMf

READ
WRITE
UNLOCK
DELETE
STARTBR
READNEXT
READPREV
RESETBR
ENDBR

Hauptspeichersteuerung

GETMAIN
FREEMAIN

Andere

END DEQ
SUSPEND
WRITEJOURNALNUM
HANDLE CONDITION
IGNORE CONDITION

Abb. 8.1.9
Überblick über die CICS Befehle

8.1.7 Beispiel für Embedded SQL

```
EXEC SQL INCLUDE SQLCA;
 EXEC SQL BEGIN DECLARE SECTION;
 char vname[20];
 char nname[20];
 EXEC SQL END DECLARE SECTION;

 main()
 {
   EXEC SQL DECLARE C1 CURSOR FOR
    SELECT VNAME,NNAME FROM PRAKT20.TAB020;
   EXEC SQL OPEN C1;
   EXEC SQL FETCH C1 INTO :vname, :nname;
   memcpy(map5020.map5020i.vnam1i,vname,20);
   memcpy(map5020.map5020i.nnam1i,nname,20);
   EXEC SQL FETCH C1 INTO :vname, :nname;
   memcpy(map5020.map5020i.vnam2i,vname,20);
   memcpy(map5020.map5020i.nnam2i,nname,20);
   EXEC SQL FETCH C1 INTO :vname, :nname;
   memcpy(map5020.map5020i.vnam3i,vname,20);
   memcpy(map5020.map5020i.nnam3i,nname,20);
   EXEC SQL FETCH C1 INTO :vname, :nname;
   memcpy(map5020.map5020i.vnam4i,vname,20);
   memcpy(map5020.map5020i.nnam4i,nname,20);
   EXEC SQL CLOSE C1;

   EXEC CICS SEND MAP("map5020") MAPSET("set5020") ERASE;

 }
```

Abb. 8.1.10
Code Beispiel in C++

Der hier wiedergegebene Code in C++ enthält eine Reihe von EXEC SQL Statements für Zugriffe auf eine Datenbank sowie ein einziges EXEC CICS Statement, mit dem eine Nachricht an einen Terminal gesendet wird.

Das EXEC CICS SEND MAP Statement sendet die mittels der SQL Statements aus der Datenbank ausgelesenen Daten an den Bildschirm.

8.1.8 Erstellen einer CICS - DB2 Anwendung

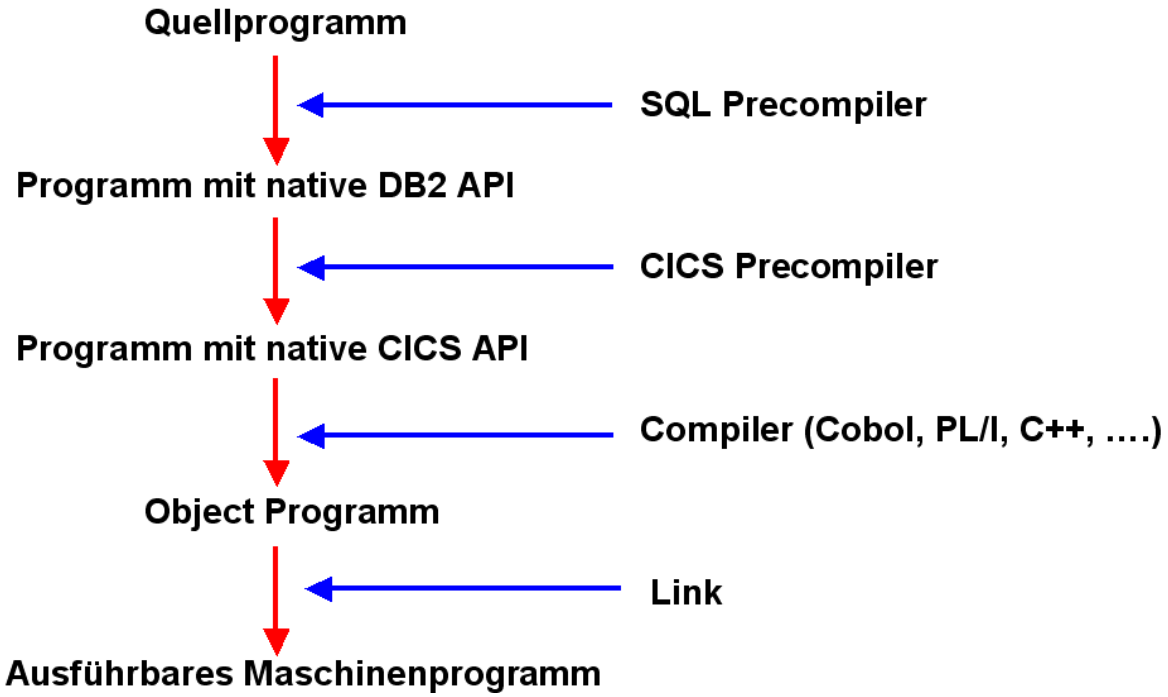


Abb. 8.1.11
Übersetzung des Quellprograms

Das in Abb. 8.1.10 dargestellte Quellprogramm wird vor der Übersetzung zunächst von einem SQL Precompiler verarbeitet, welcher die EXEC SQL Statements durch Quellcode Makros oder Bibliotheksroutine-Aufrufe ersetzt, die der angesprochenen Datenbank entsprechen. Der Code in diesen Makros würden z.B. bei einer Oracle Datenbank anders aussehen als bei einer DB2 Datenbank.

Anschließend bewirkt der CICS Precompiler eine ähnliche Vorgehensweise für alle EXEC CICS Statements.

Danach wird der so entstandene Quellcode durch einen regulären Compiler übersetzt.

8.2 Ausführungsbeispiel einer CICS Transaktion

8.2.1 Die NACT Transaktion

Im Folgenden schlüpfen wir in die Rolle eines Sachbearbeiters eines Unternehmens, der von seinem Arbetsplatzrechner eine Transaktion aufruft und durchführt.

Die Transaktion hat den Namen NACT. Sie ist auf dem z/OS Rechner der Universität Leipzig http://jedi.informatik.uni-leipzig.de installiert, und ist Bestandteil eines dort verfügbaren Vorrats an Übungen.

NACT wird in einer fiktiven Firma, einem Kaufhaus mit dem Namen KanDoIT eingesetzt. KanDoIT hat Tausende von Kunden, an welche Kreditkarten des Kaufhauses ausgegeben wurden. Die Kundendatei ist als VSAM Datei implementiert, mit je einem Record pro Kunde. Jeder Record enthält Namen, Adresse, Tel. Nr., Kreditgrenze, derzeitiger Kontostand, usw. Jeder Kunde ist durch eine 5-stellige KontoNr. (VSAM Index) identifiziert, die dem Kunden von Hand zugeordnet wurde. In der Praxis würde sie nicht von Hand sondern automatisch erzeugt werden; diese Komponente fehlt, um die Anwendung einfach zu halten.

Eine detaillierte Beschreibung der NACT Transaktion ist in einem ausgezeichneten Lehrbuch:

J. Horswill: “Designing & Programming CICS Applications”. O´Reilly, 2000, ISBN 1-56592-676-5

enthalten, welches leider vergriffen, aber als Kindle Ausgabe bei Amazon erhältlich ist. Die Business Logik der Anwendung ist in Cobol geschrieben und nahezu unverändert von einer älteren Transaktion ACCT übernommen. Für diese existiert eine hervorragende Dokumentation in dem CICS Application Programming Primer:

http://www.cedix.de/Literature/Textbooks/Cobol/CicsCobPrimer.pdf

Der CICS Application Programming Primer enthält eine sehr detaillierte – Codezeile für Codezeile – Beschreibung der NACT bzw. ACCT Cobol Business Logik.

Eine ebenfalls sehr gute Beschreibung ist in einer Diplomarbeit der Universität Leipzig zu finden:

Tobias Busse: Generation of a Java front end for a standalone CICS application accessed through MQSeries.

http://www.cedix.de/DiplArb/Busse04.pdf

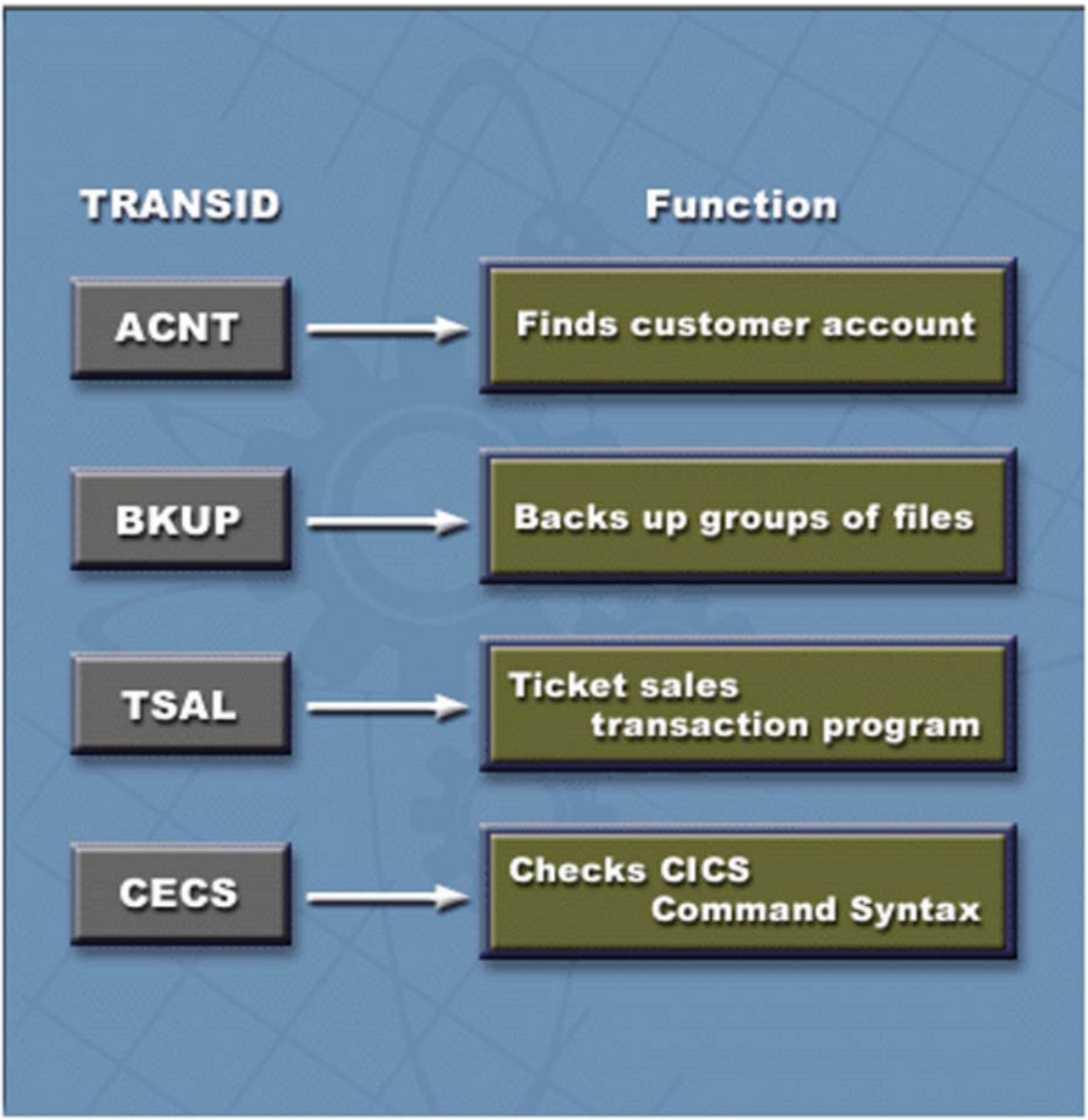
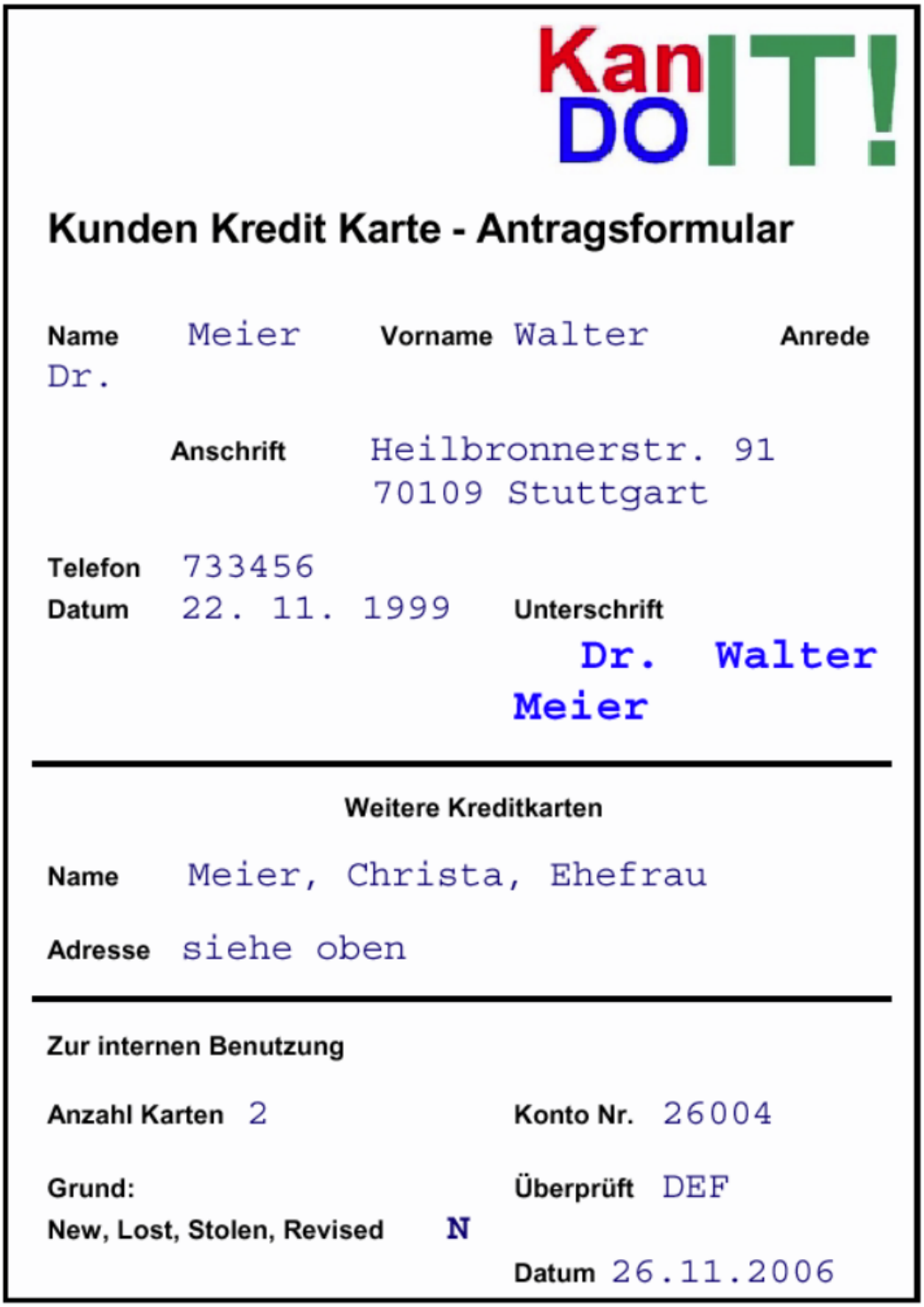


Abb. 8.2.1
Ein bestimmtes Transaktionsprogramm wir mittels einer TRID aufgerufen

NACT ist eine Transaction ID (TRID). Mit einer TRID wird eine (von vielen) Anwendungen aufgerufen, die CICS als Transaktionen ausführt. Die TRID identifiziert die Transaktion. Ein normaler Benutzer ruft einen Dienst eines CICS Servers auf, indem er eine TRID eingibt.

CICS TRIDs sind grundsätzlich immer 4 Zeichen lang.

Einige Transaktionen und Ihre TRIDs sind Bestandteil des CICS TP Monitors. Z. B. implementiert die CEDA Transaktion eine Kommandozeilen-Shell.



KanDOIT!

Kunden Kredit Karte - Antragsformular

Name Meier Vorname Walter Anrede Dr.

Anschrift Heilbronnerstr. 91
70109 Stuttgart

Telefon 733456

Datum 22. 11. 1999 Unterschrift Dr. Walter Meier

Weitere Kreditkarten

Name Meier, Christa, Ehefrau

Adresse siehe oben

Zur internen Benutzung

Anzahl Karten 2 Konto Nr. 26004

Grund: Überprüft DEF

New, Lost, Stolen, Revised N

Datum 26.11.2006

Abb. 8.2.2
Antragsformular für eine Kreditkarte

In unserem Beispiel besucht ein Neukunde, Dr. Walter Meier, das Kreditbüro der Firma KanDoIT und beantragt eine Kreditkarte. Der Sachbearbeiter der Firma KanDoIT lässt ihn das in Abb. 8.2.2 dargestellte Formular ausfüllen.

Der Sachbearbeiter ordnet dem Kunden die Konto Nr. 26004 zu und signiert das Antragsformular mit dem Kürzel seines Namens DEF. Als Grund für die Kreditkartenausgabe wird N (neu) angegeben.

Field	Length	Occurs	Total
Account Number (Key)	5	1	5
Surname	18	1	18
First Name	12	1	12
Middle initial	1	1	1
Title (Jr, Sr, and so on)	4	1	4
Telephone number	10	1	10
Address line	24	3	72
Other charge name	32	4	128
Cards issued	1	1	1
Date issued	6	1	6
Reason issued	1	1	1
Card code	1	1	1
Approver (initials)	3	1	3
Special codes	1	3	3
Account status	2	1	2
Charge limit	8	1	8
Payment history:	(36)	3	108
-Balance	8		
-Bill date	6		
-Bill amount	8		
-Date paid	6		
-Amount paid	8		

Abb. 8.2.3
Kundenrecord des VSAM Datasets

In der Kunden Kreditkartenverwaltung ist die Kundendatei als Key-sequenced VSAM Datei angelegt. Die Struktur eines Records ist in Abb. 8.2.3 abgebildet.

8.2.2 Komponenten der NACT Transaktion

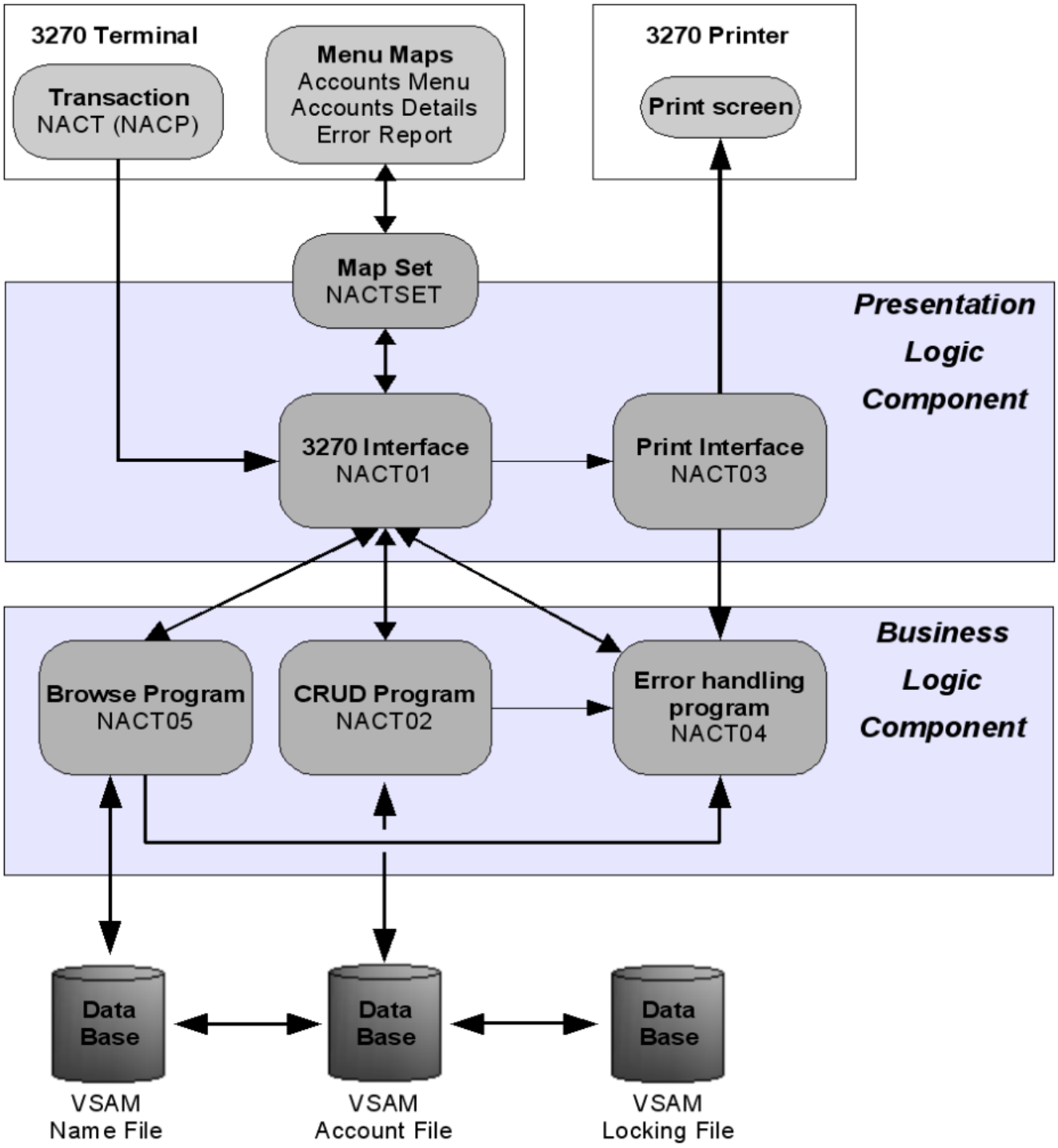


Abb. 8.2.4
Komponenten der NACT Transaktion

Die Struktur der NACT Transaktion besteht aus einem Kliententeil und einem CICS Server Teil. Der Klienten Teil besteht aus einem 3270 Emulator. Zusätzlich ist ein Arbeitsplatzdrucker vorgesehen, der in der Implementierung auf dem z/OS Rechner der Universität Leipzig fehlt.

Der Server Teil besteht aus einer Presentation Logik (in BMS implementiert), und einer Business Logik (in Cobol implementiert.

CRUD (Create, Read, Update, Delete) ist das Kernelement der Business Logic.

8.2.3 Ausführungsbeispiel

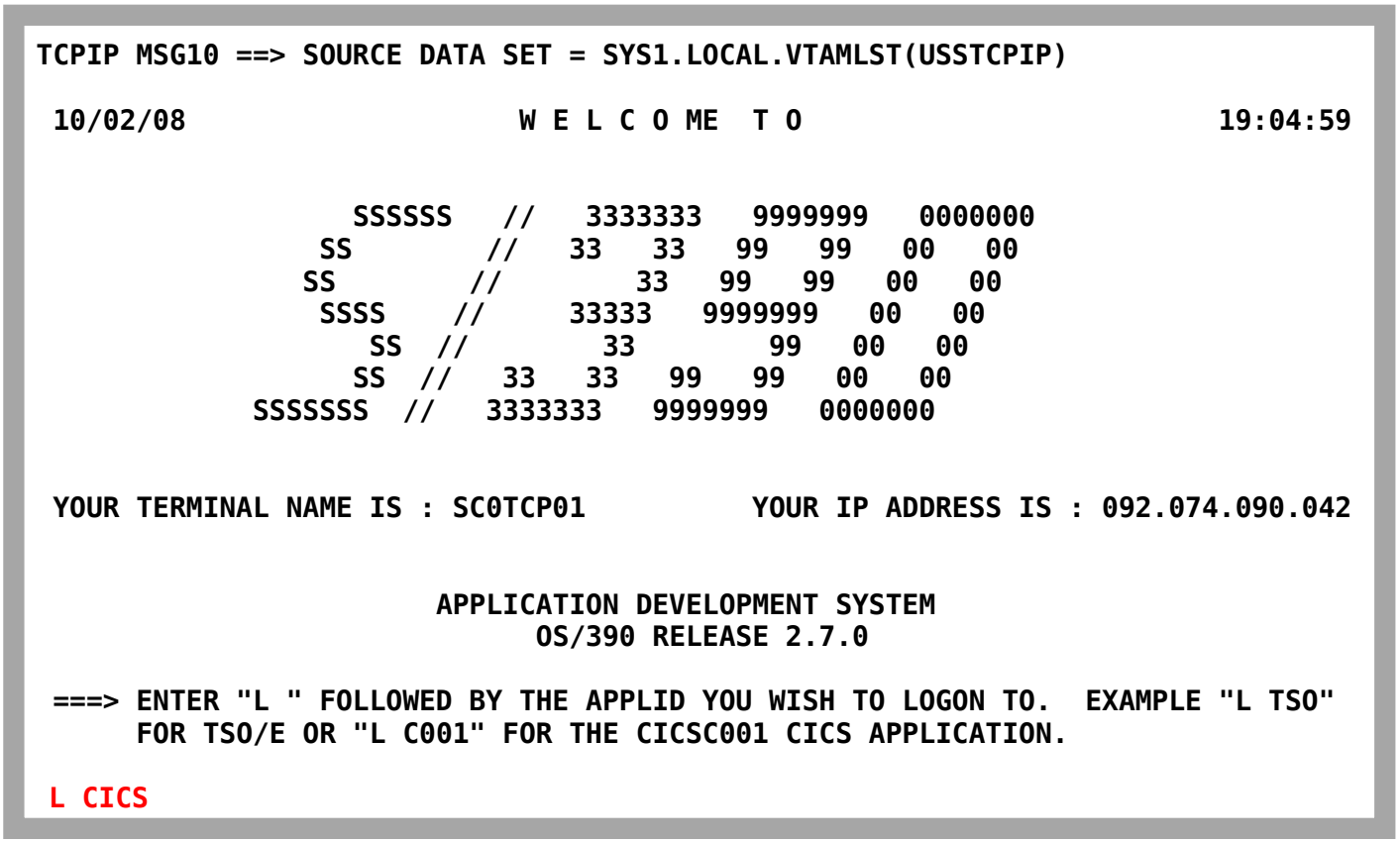


Abb. 8.2.5
Welcome Screen

Die folgende Bildschirmfolge stellt den Ablauf einer Transaktion dar, in der Dr. Walter Meier eine neue Kreditkarte beim Großkaufhaus KanDoIT beantragt.

Auf dem Eingangsbildschirm des z/OS Rechners wird das CICS Subsystem mit dem Kommando L(ogon) CICS aufgerufen. An Stelle von CICS könnten hier auch andere Subsysteme aufgerufen werden, z.B. IMS oder TSO.

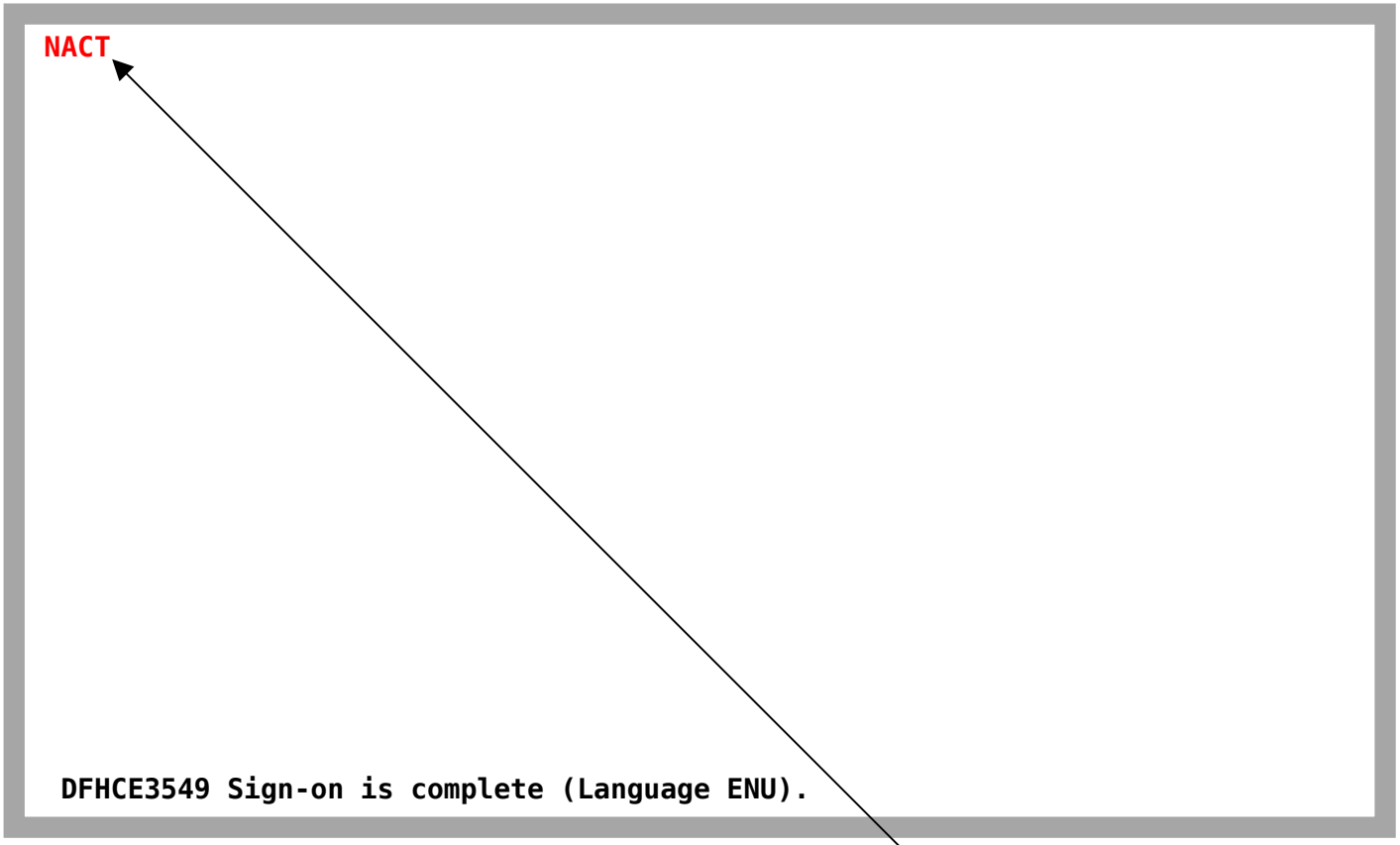


Abb. 8.2.6
TRID Selektion

Auf eine sehr kryptische Weise fordert CICS den Benutzer auf, die TRID der Transaktion einzugeben, hier NACT.

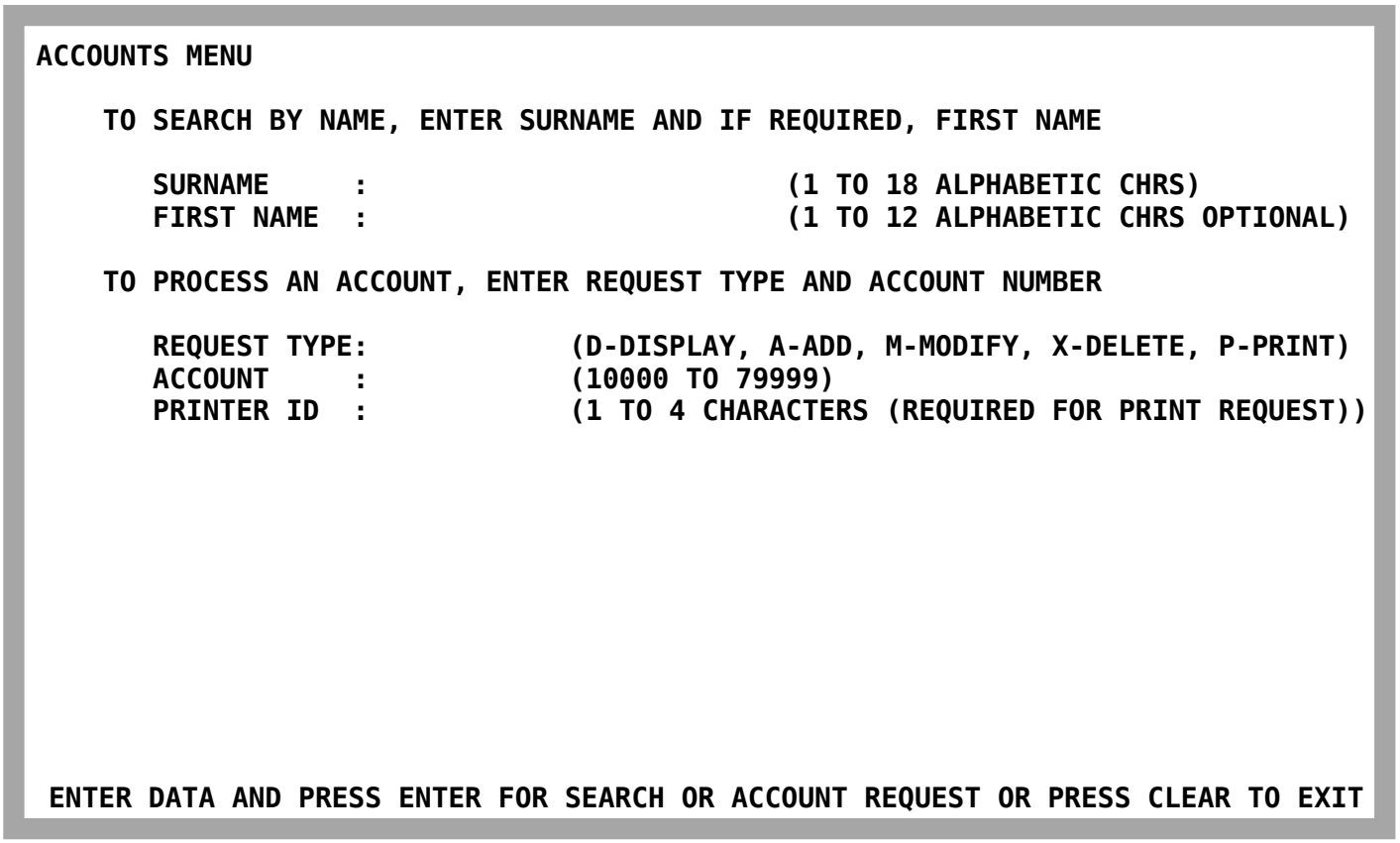


Abb. 8.2.7
Leere Accounts Menu MAP

Dies ist der Eingabescreen der NACT Transaktion. Er enthält vorgefertigten Text sowie 5 Felder (hier unsichtbar), in die der Benutzer an seinem Terminal Daten eingeben kann.

Der hier dargestellte Bildschirminhalt wird von CICS als „MAP“, von anderen Transaktionsmonitoren auch als View oder Screen bezeichnet. Zu einer Transaktion gehören in der Regel mehrere unterschiedliche MAPs.

Die NACT Transaktion verwendet zwei unterschiedliche Maps, mit den Namen „Accounts Menu“ bzw. „Accounts“. Der Name wird jeweils links oben in den Bildschirm angegeben.

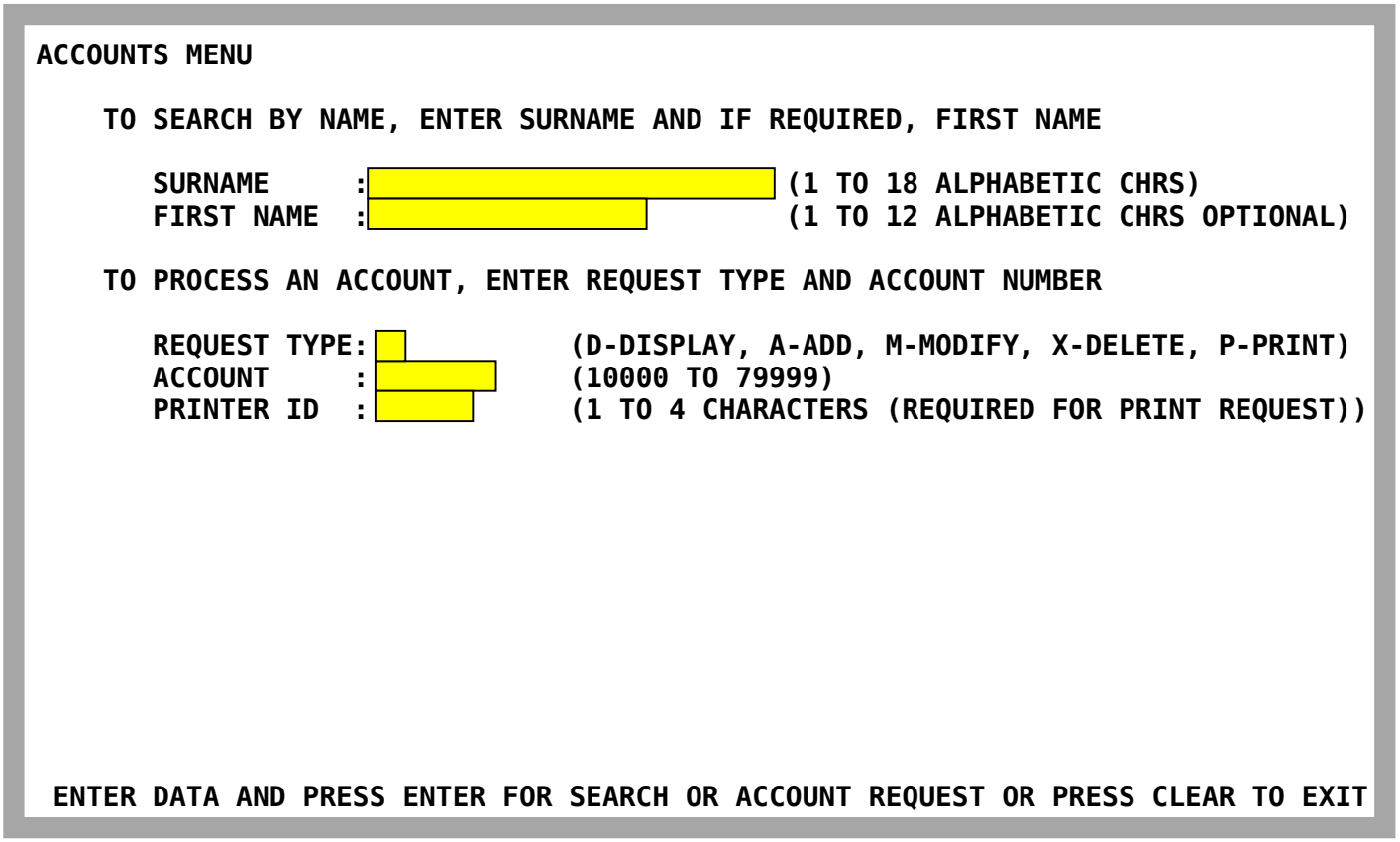


Abb. 8.2.8
Eingabefelder der Accounts Menu Map

Dies ist der gleiche Eingabescreen der NACT Transaktion. Zum Unterschied gegenüber der vorherigen Darstellung sind hier die möglichen (an sich unsichtbaren) 5 Eingabefelder gelb dargestellt. Die Felder haben eine Länge von jeweils 18, 12, 1, 5 und 4 Zeichen. Vom Benutzer wird erwartet, dass er dies weiß.

Bitte Zurückhaltung mit Ihren Verbesserungsvorschlägen.

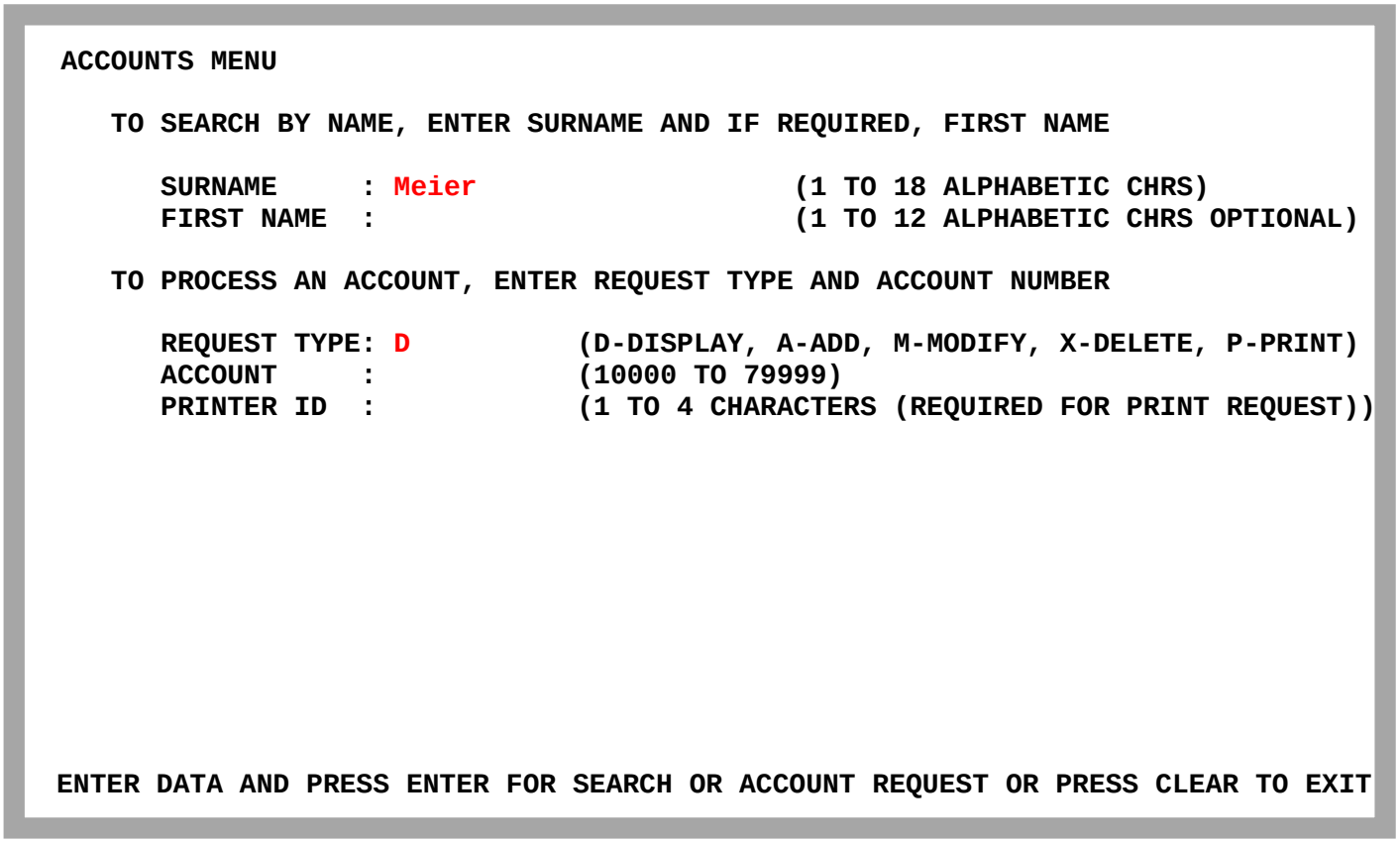


Abb. 8.2.9
Dateneingabe in die Accounts Menu Map

Der Sachbearbeiter gibt den Namen Meier und den Request Type D (für Display) ein. Er will überprüfen, ob der Name Walter Meier schon vorhanden ist.

In der hier gewählten Darstellung werden die Eingaben des Sachbearbeiters in rot und die Antworten von CICS in blau dargestellt.

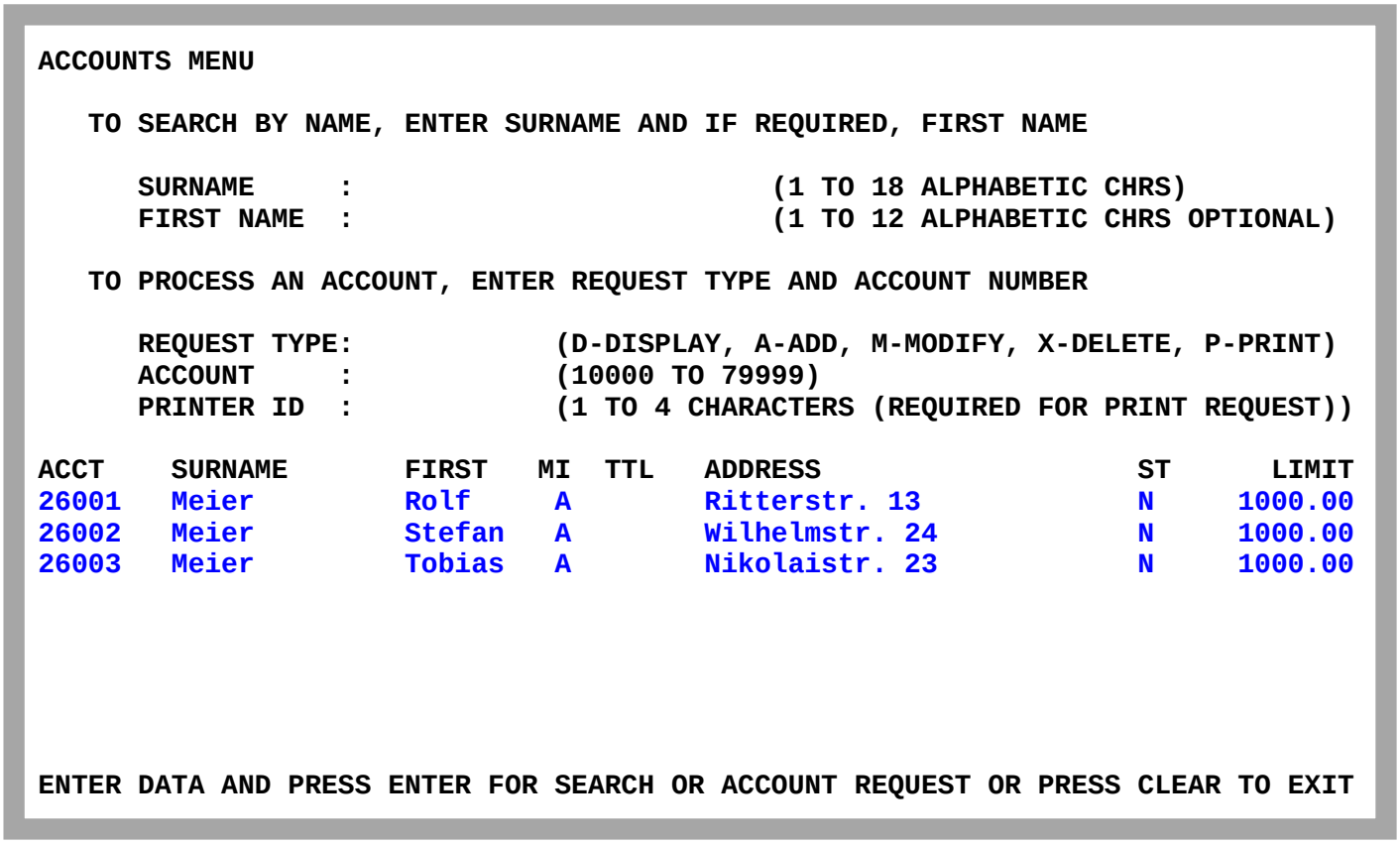


Abb. 8.2.10
Antwort von CICS

CICS findet in seiner VSAM Kundendatei drei Einträge mit dem Namen Meier, allerdings keinen Dr. Walter Meier.

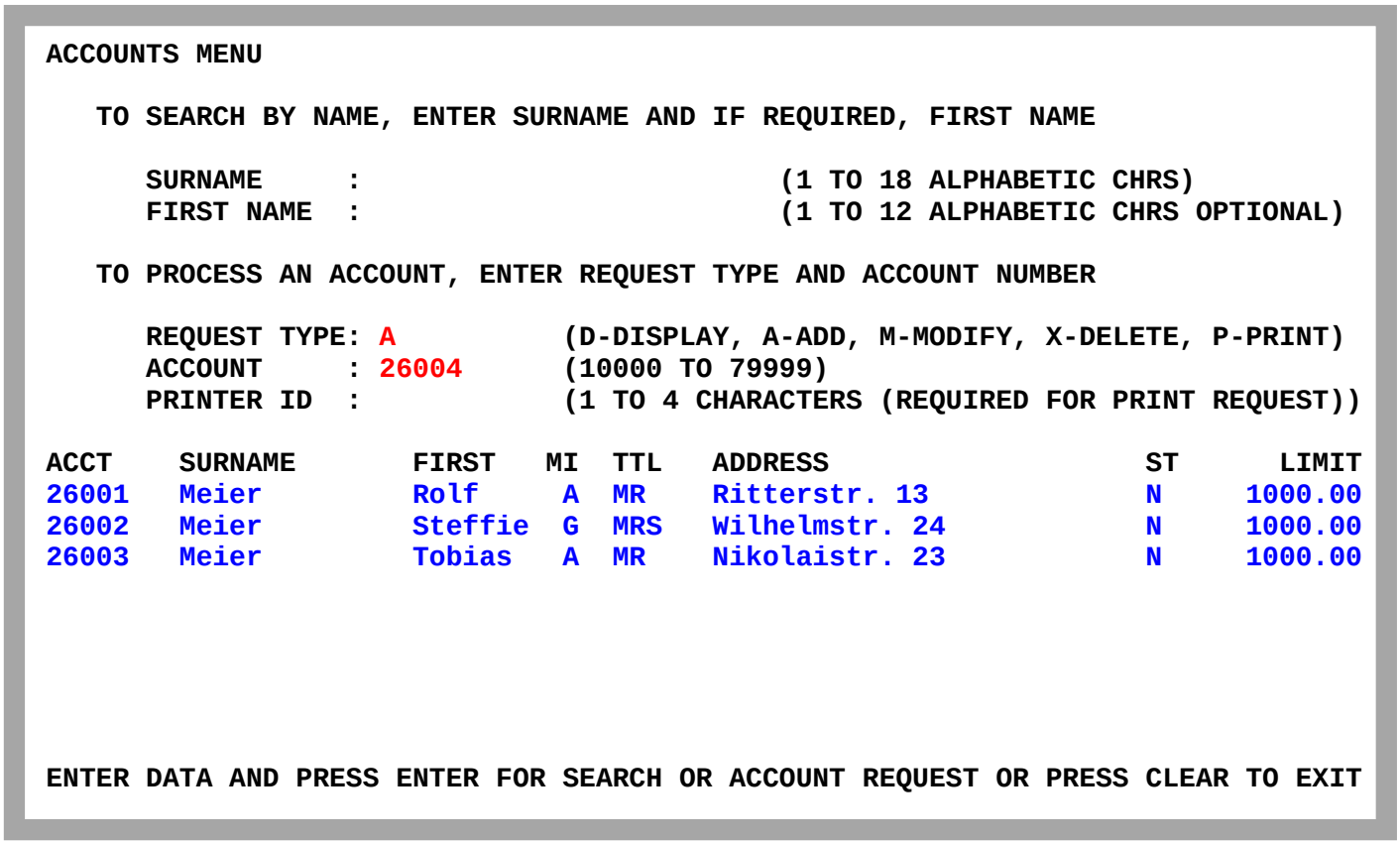


Abb. 8.2.11
Eingabe der Aufforderung „Add“ in die Accounts Menu Map

Der Sachbearbeiter gibt als Request Type A (für add) ein. Damit soll ein neuer Record für Herrn Dr. Walter Meier angelegt werden.

Gleichzeitig ordnet er dem neuen Kunden die Konto Nr. 26004 zu.

```
ACCOUNTS                 ADD ACCOUNT NUMBER 26004

SURNAME      :                    (18 CHRS) TITLE      :      (4 CHRS OPTIONAL)
FIRST NAME   :                    (12 CHRS) MIDDLE INIT:      (1 CHR  OPTIONAL)
TELEPHONE    :                    (10 DIGS)
ADDRESS LINE1:                         (24 CHRS)
        LINE2:                         (24 CHRS)
        LINE3:                         (24 CHRS OPTIONAL)

CARDS ISSUED :           (1 TO 9)          CARD CODE  :      (1 CHR)
DATE ISSUED  :           (MM DD YY)        REASON CODE:      (N,L,S,R)
APPROVED BY  :           (3 CHRS)

UPTO 4 OTHERS WHO MAY CHARGE (EACH 32 CHRS OPTIONAL)
    O1:                                    O2:
    O3:                                    O4:
SPECIAL CODE1:   CODE2:   CODE3:   (EACH 1 CHR OPTIONAL)
NO HISTORY AVAILABLE AT THIS TIME       CHARGE LIMIT                STATUS



NOTE:- DETAILS IN BRACKETS SHOW MAXIMUM NO. CHARACTERS ALLOWED AND IF OPTIONAL

FILL IN AND PRESS "ENTER," OR "CLEAR" TO CANCEL
```

Abb. 8.2.12
Leere Accounts MAP

CICS stellt eine andere MAP auf den Bildschirm, die Accounts Map. In diesem Beispiel verwenden wir nur diese zwei MAPs, die meisten Transaktionen haben eine sehr viel größere Anzahl von MAPs.

Die Eingabefelder sind wiederum unsichtbar.

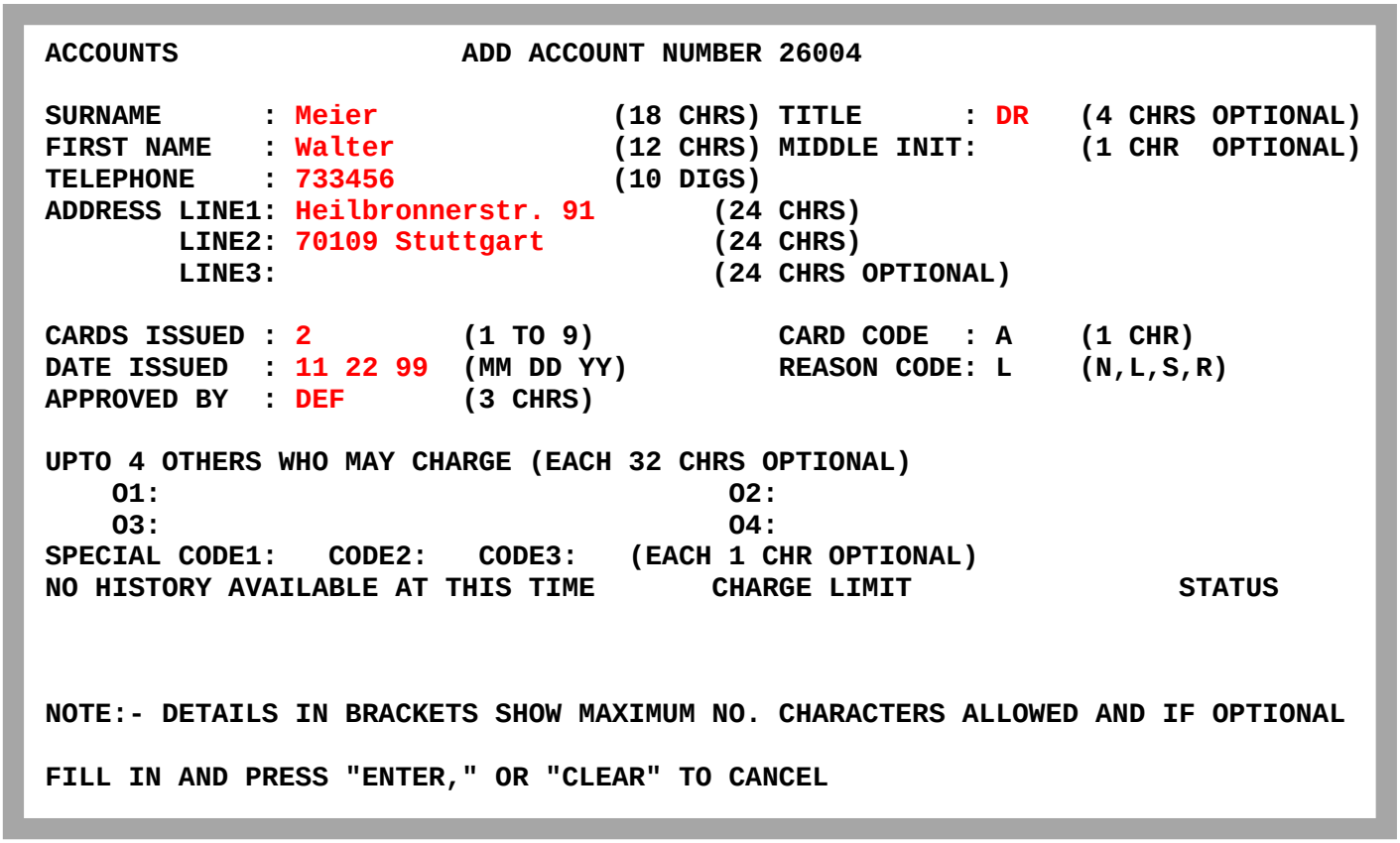


Abb. 8.2.13
Daten Eingabe in die Accounts Map

Der Sachbearbeiter überträgt die Daten aus dem Antragsformular und betätigt die Entertaste. Dies bewirkt, dass die eingegebenen Daten in den neuen VSAM Record für Herrn Dr. Walter Meier übernommen werden.

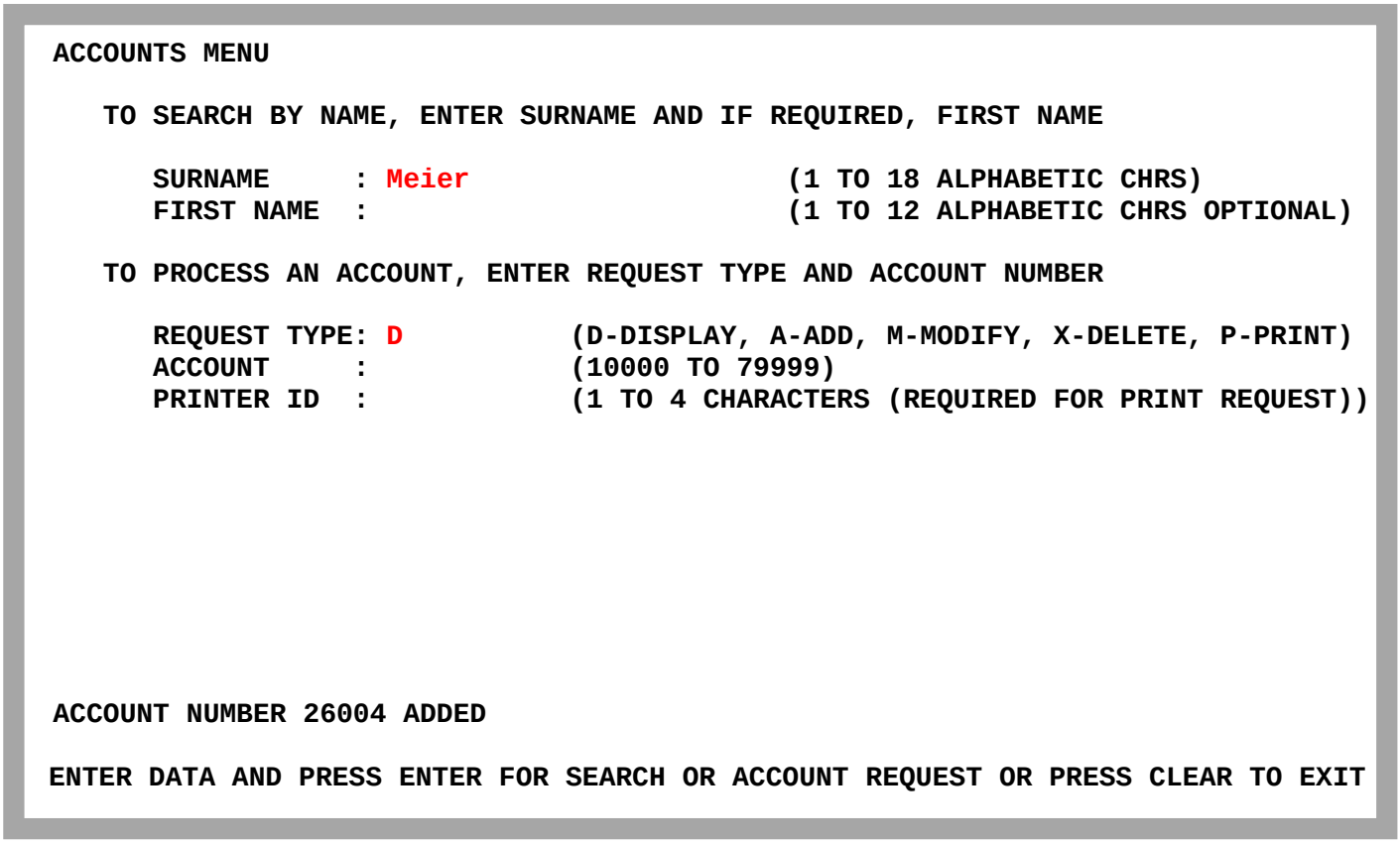


Abb. 8.2.14
Zurück zur Accounts Menu Map

Es erscheint wieder die erste Map. Der Sachbearbeiter ruft nochmals die D (display) Funktion für alle Meier's in der Kundendatei auf.

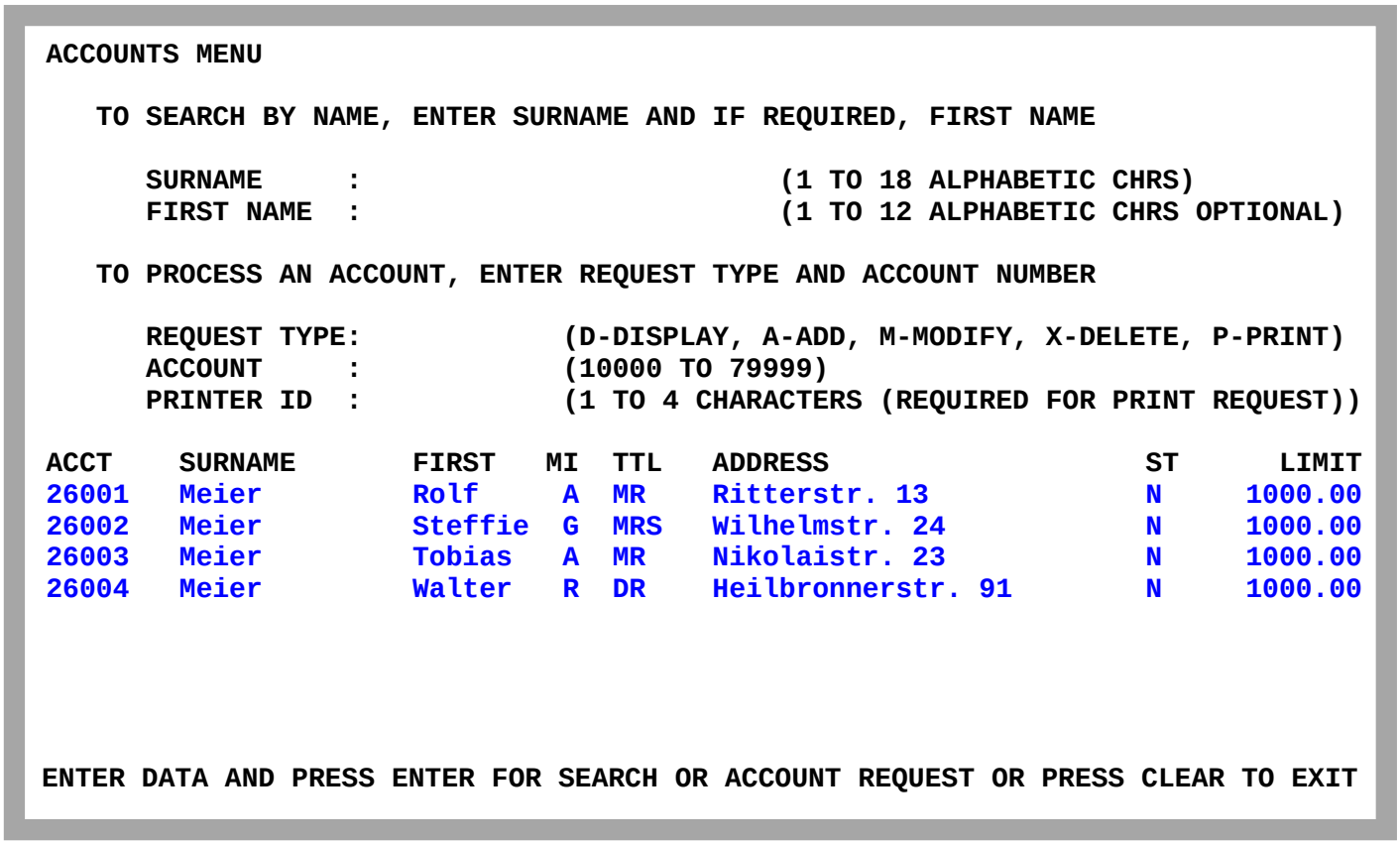


Abb. 8.2.15
CICS Ausgabe

CICS zeigt an, dass es nun auch einen Kunden Record für Herrn Dr. Walter Meier gibt.

Damit wurde die Transaktion erfolgreich abgeschlossen.

8.2.4 Moderne Oberflächen

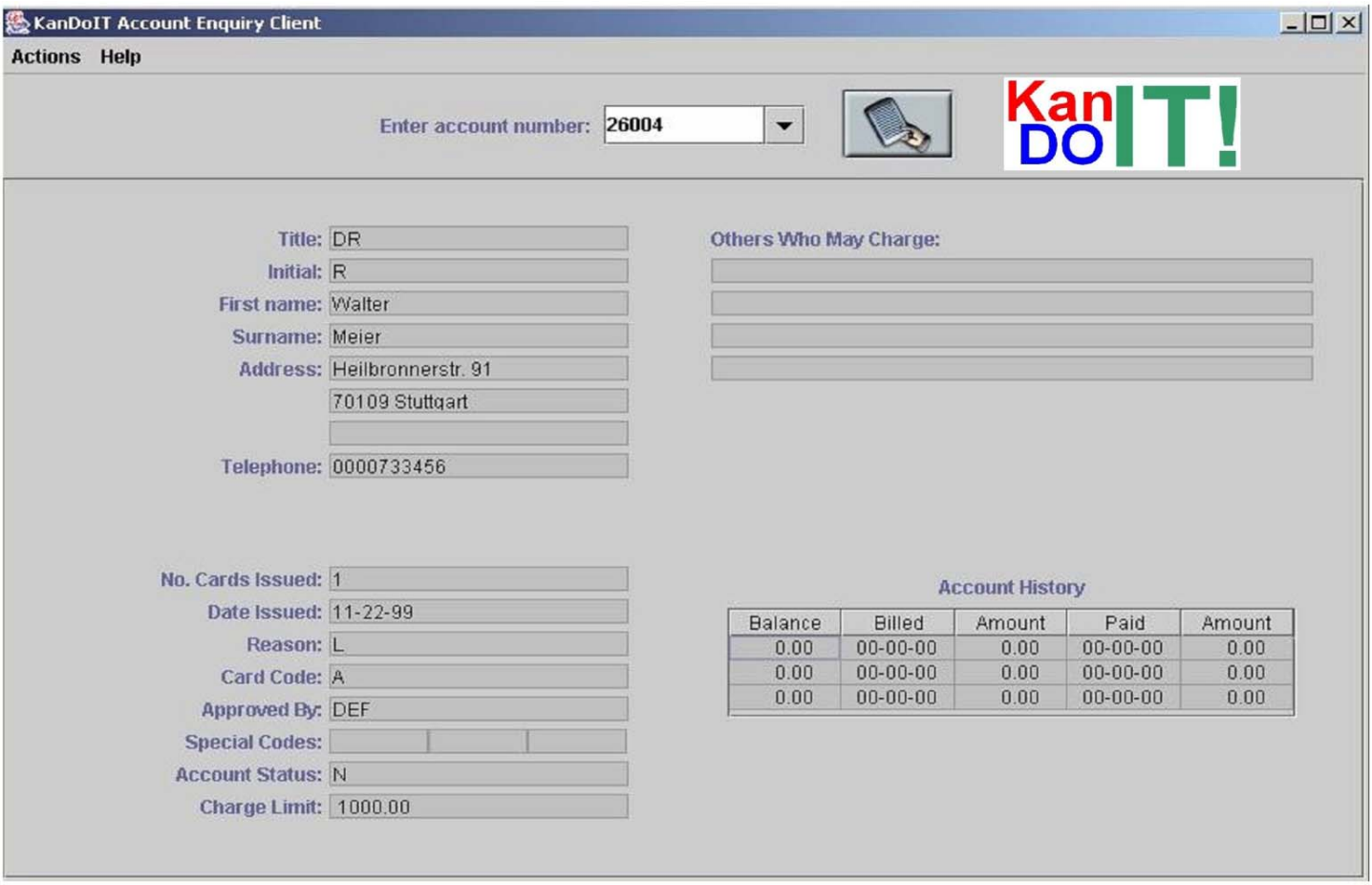


Abb. 8.2.16
Grafische Bildschirmdarstellung

In dem hier gezeigten Beispiel benutzen alle Screens das 1971 entstandene „3270 Protokoll“ und die BMS (Basic Mapping Support) Präsentationslogik, welche Bestandteil von jedem CICS TP Monitor ist.

Eine Alternative sind moderne Benutzeroberflächen, die in der großen Mehrzahl der Fälle mit Hilfe von Java programmiert werden.

In Wirtschaft und Verwaltung wurden in den letzten Jahren für die allermeisten CICS Anwendungen hiermit ausgestattet. Dies geschah fast immer als Alternative (und nicht als Ersatz) zu der existierenden BMS Präsentationslogik.

Eine grafische Ausgabe der Abb. 8.2.13 ist in Abb. 8.2.16 gezeigt. In den meisten Unternehmen kann der CICS Benutzer zwischen mehrere Darstellungsalternativen auf seinem Bildschirm wählen. Interessanterweise ist die klassische BMS Wiedergabe bei relativ vielen Benutzern nach wie vor recht populär.

8.3 CICS Nucleus

8.3.1 Ablauf einer CICS Transaktion

Unter CICS werden viele Transaktionen gleichzeitig und parallel verarbeitet. Hierbei wird die „Isolation“ der vier ACID Bedingungen strikt eingehalten.

Unter dem z/OS Betriebssystem laufen alle CICS Anwendungen und Dienste, sowie die CICS Nucleus Komponenten im Problemstatus, ungeschützt voneinander, innerhalb eines einzigen virtuellen Adressenraums. Anwendungen und Ressource Manager laufen als Thread-ähnliche Konstrukte innerhalb dieses Adressenraums.

Thread-ähnlich: Scheduling erfolgt durch den CICS Nucleus, und nicht – wie bei echten Threads – durch die Scheduler/Dispatcher Komponente des Betriebssystem Kernels.

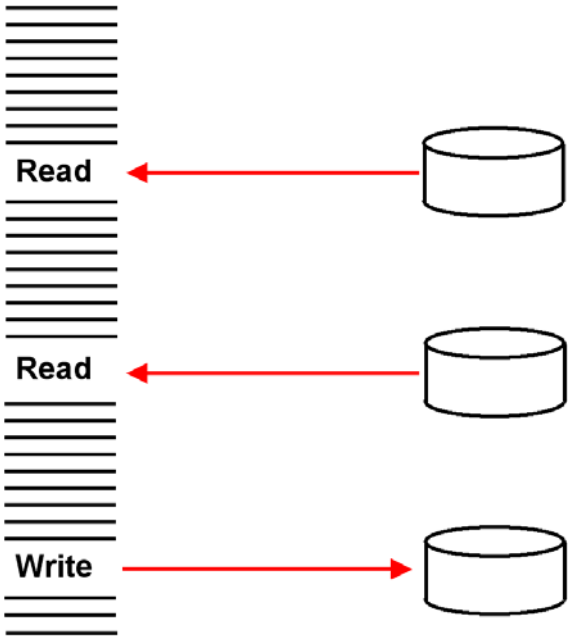


Abb. 8.3.1
Ablauf einer CICS Transaktion

Das Programm für die Ausführung einer Transaktion besteht aus Folgen von Maschinenbefehlen, in die in unregelmäßigen Abständen I/O Operationen eingebaut sind (READ, WRITE, OPEN, CLOSE, CONTROL ....).

Während des Zeitabschnittes in der die Read/Write Operation durchgeführt wird, verarbeitet die CPU eine andere Transaktion. Dies geschieht deshalb, weil eine Befehlsausführung etwa 1 ns benötigt, während ein Plattenspeicherzugriff etwa 10 ms braucht, also 7 Größenordnungen (10^7) länger dauert.

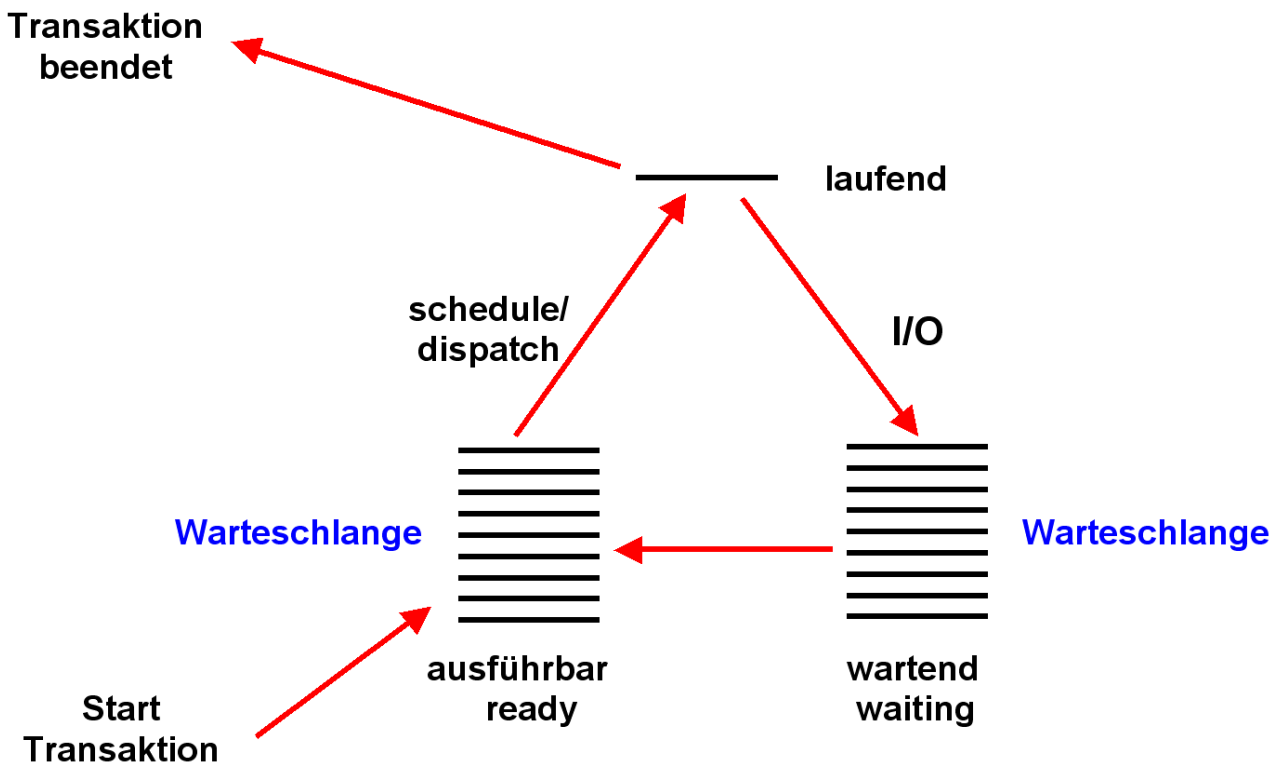


Abb. 8.3.2
Scheduling von Transaktionen

Jede Transaktion rotiert wiederholt durch die Zustände laufend, wartend und ausführbar. Der CICS eigene Scheduler selektiert aus der Queue ausführbare Transaktionen jeweils einen Kandidaten wenn immer die CPU frei wird (wir nehmen hier an, dass die CICS Region nur eine CPU benutzt).

Alle Anwendungsprogramme, die von Transaktionen aufgerufen werden, befinden sich in einer Programmbibliothek. Beim erstmaligen Aufrufen einer Transaktion wird überprüft, ob sich das entsprechende Programm bereits im Hauptspeicher befindet. Wenn nein, wird das Programm zunächst in den Hauptspeicher geladen.

Es kann aber sein, dass eine andere Transaktion das gleiche Anwendungsprogramm bereits benutzt oder benutzt hat. In diesem Fall ist kein Laden aus der Programmbibliothek erforderlich. Wenn mehrere Transaktionen das gleiche Programm verwenden, ist dieses nur einmal im Hauptspeicher vorhanden.

Die Anwendungsprogramme sind deshalb „quasi reentrant“ geschrieben: Eine Transaktion, deren Ausführung unterbrochen wird, hinterlässt de Anwendungscode so, dass eine andere Transaktion den gleichen Code ohne Probleme ausführen kann.

8.3.2 CICS Nucleus

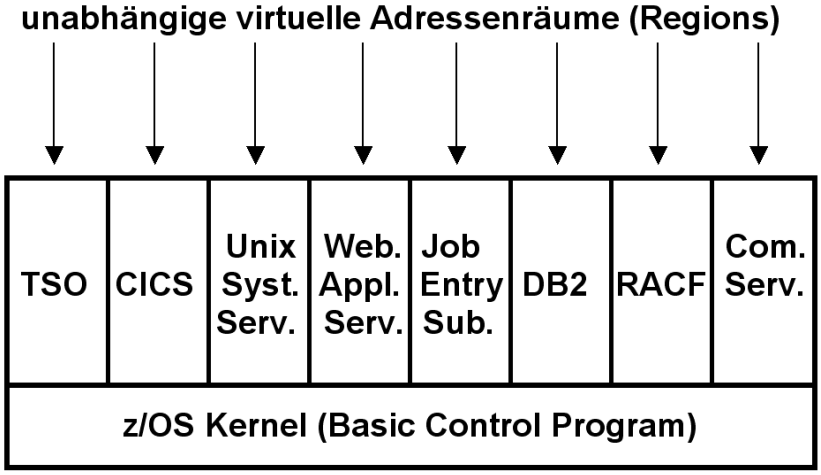


Abb. 8.3.3
z/OS Subsysteme

Der z/OS Kernel unterstützt eine Vielzahl von virtuellen Adressenräumen, die im z/OS Jargon als Regions bezeichnet werden.

Manche Regions beherbergen Subsysteme, die Teil des Betriebssystems sind, aber im Benutzerstatus laufen. Einige der (zahlreichen) Subsysteme sind:

- **CICS Transaktionsverarbeitung**
- **TSO Shell, Entwicklungsumgebung**
- **USS Unix kompatible Shell, Entwicklungsumgebung**
- **WAS WebSphere Web Application Server**
- **JES Job entry Subsystem**
- **DB2 relationale Datenbank**
- **RACF Sicherheitssystem**
- **Communications Server**

CICS verhält sich wie ein Mini-Betriebssystem unterhalb des eigentlichen Betriebssystem und stellt eine Umgebung für die Ausführung von CICS Anwendungsprogrammen (Transaktionen) zur Verfügung.

Unter z/OS läuft CICS als ein z/OS Prozess in einem einzigen virtuellen Adressenraum (Region). Dieser Adressenraum enthält einmal die CICS Laufzeitumgebung (als CICS Nucleus bezeichnet). Weiterhin laufen alle CICS Transaktionen in der gleichen Region unter Kontrolle des CICS Nucleus. CICS Anwendungsprogramme benutzen EXEC CICS Befehle für alle Schnittstellen (interfaces), die bezüglich Gewährleistung der ACID Eigenschaften relevant sind. Der CICS Nucleus wiederum greift auf den z/OS Überwacher zu.

CICS ist die Schnittstelle zwischen Anwendungsprogrammen, Datenbanken und Netzwerk-(Communication) Komponenten.

Aus Sicht des z/OS Betriebssystems existiert nur eine einzige Anwendung und nur ein Prozess – der CICS Transaktionsmonitor mit allen derzeitig ausgeführten Transaktionen. Die Ausführung einer CICS Transaktion wird als „Task“ bezeichnet. In der CICS Region sind gleichzeitig viele (möglicherweise tausende) von CICS Tasks aktiv. Eine CPU führt in jedem Augenblick nur eine dieser Tasks aus (laufend, running); die übrigen Tasks sind ausführbar (ready) oder wartend (waiting).

CICS verwendet den System z Architektur „Storage Protection Key“ Mechanismus, um den CICS Nucleus von den Anwendungen zu isolieren, die im gleichen Adressenraum laufen. Der CICS Nucleus verwendet normalerweise Storage Protection Key = 8, was einen Zugriff auf den ganzen CICS virtuellen Adressenraum ermöglicht. CICS Anwendungen laufen normalerweise mit Storage Protection Key = 9, was den Zugriff auf den von den Anwendungen benutzten Adressenraum begrenzt.

"Quasi Reentrant" - eine CICS-Anwendung wird nur bei der Ausführung einer CICS API (EXEC CICS ...) unterbrochen.

8.3.3 CICS Domänen (Domains)

Der CICS Nucleus besteht aus einer Gruppe von Domänen (domains). Jede Domäne enthält eine Gruppe von Objekten, die einen gemeinsamen Satz von Funktionen ausführen.

Domänen enthalten als „Manager“ bezeichnete Programm Module, Tabellen und Steuerblöcke. Einige wenige Domänen sind für den größten Teil des Verarbeitungsablaufes einer Transaction zuständig:

- **Terminal Manager Domäne**
- **Transaction Manager (XM, auch als Task Manager bezeichnet) Domäne**
- **Program Manager (PG) Domäne**
- **Storage Manager (SM) Domäne**
- **File Manager Domäne**

Die „Manager“ wurden in der Vergangenheit als “Controls” bezeichnet: Terminal Control, Task Control, Program Control, Storage Control und File Control. Die Begriffe Control und Manager haben unter CICS die gleiche Bedeutung.

Anwendungsprogramme (Transaktionen) greifen auf Manager (CICS Management Module) zu um Terminal I/O, File I/O, Program loading und Zugriffe auf andere System Ressourcen für sie durchzuführen.

Anwendungsprogramme (Transaktionen) teilen sich die CICS Region mit anderen CICS Komponenten. Neben den Anwendungsprogrammen und den Management Modulen existieren weitere Arten von Objekten in der CICS Region:

- **Tabellen (Tables) – Die Management Module benutzen CICS System Tables um Informationen über Terminals, Dateien und Anwendungsprogramme zu erhalten und zu verwalten, die Teil des CICS Systems sind. Die Tabellen werden in den Hauptspeicher geladen, wenn CICS gestartet wird. Sie bleiben aktiv, bis CICS heruntergefahren wird. (Beides sind Vorgänge, die nur sehr selten stattfinden; CICS bleibt normalerweise 24 Stunden, 7 Tage/Woche, im Betrieb).**
- **Steuerblöcke (Control blocks) – Die Management Module erstellen Steuerblöcke, mit denen der Status aller Transaktionen festgehalten wird, die in jedem Augenblick aktiv sind. Die Steuerblöcke werden freigegeben, wenn sie nicht mehr benötigt werden, z. B. wenn eine Transaktion das System verlässt.**
- **System Data Sets – Die Management Module und Anwendungsprogramme benutzen CICS System Data Sets für Transaction Logging, System Recovery, und für das Speichern von Ergebnissen, die von anderen Anwendungsprogrammen oder Transaktionen verwendet werden.**

8.3.4 CICS Nucleus Komponenten

CICS läuft als lang laufender Stapelverarbeitungsjob in einem einzigen virtuellen Adressenraum (Region in z/OS Terminologie). CICS Anwendungsprogramme laufen „run to completion“; Interactivität wird programmtechnisch gewährleistet, indem ihre maximale Ausführungszeit eine vorgegebene Grenze nicht überschreitet.

Die CICS Nucleus Komponenten (Terminal Manager, Task Manager, Program Manager, Storage Manager and File Manager) laufen in dem gleichen virtuellen Adressenraum wie alle Anwendungen. Jede Nucleus Komponente hat eine zugeordnete Tabelle: TCT, PCT, PPT, FCT. Über den Scratchpad werden Sessions eingerichtet: Der State einer Transaktion ist für die Folgetransaktion verfügbar.

Die CICS Nucleus Komponenten laufen in Domains. Domains enthalten Programme, Tabellen und Steuerblöcke.

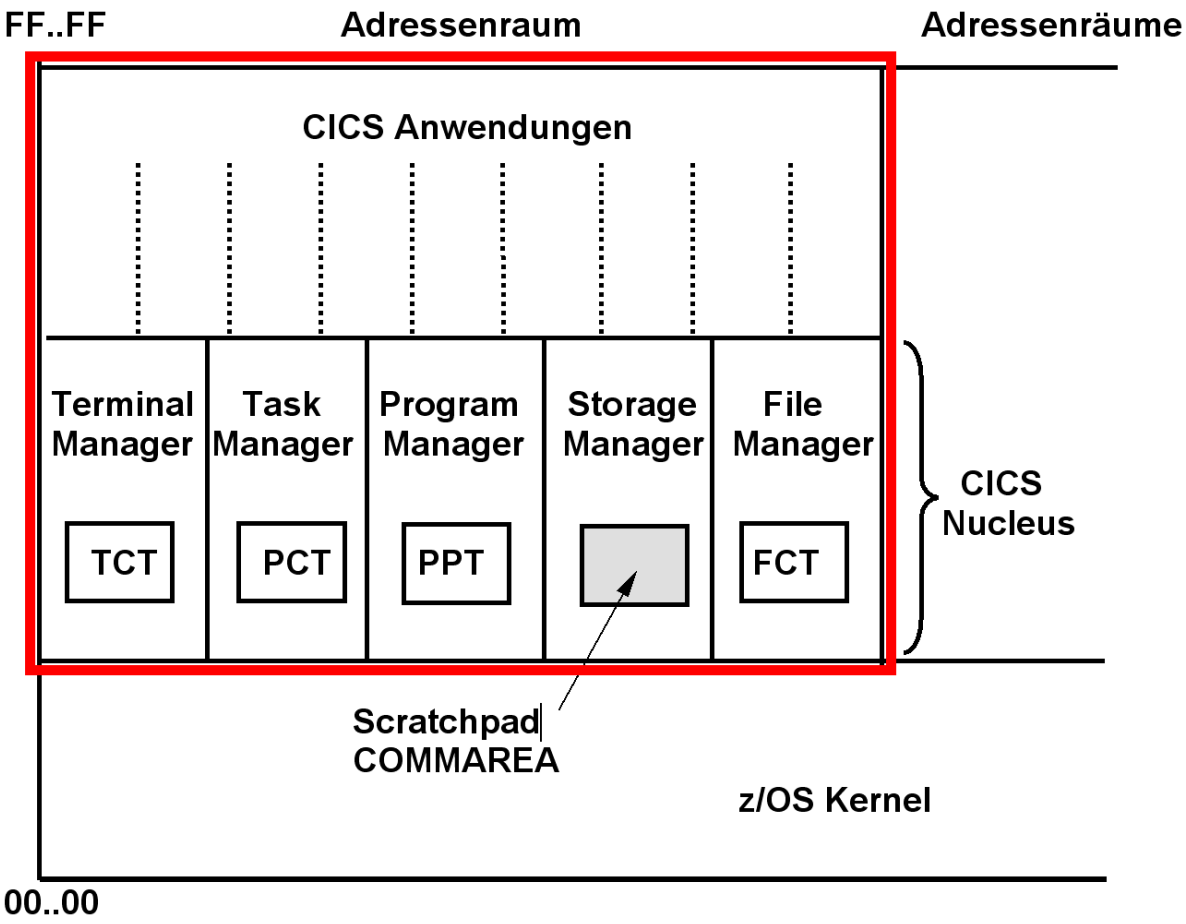


Abb. 8.3.4
CICS Bestandteile

Die Überwachung und Steuerung der Transaktionsverarbeitung erfolgt vor allem durch drei Komponenten:

- **Der Task Manager ist für den Empfang von *Transaction Requests* zuständig, sowie für die Erstellung und Steuerung von *Tasks*, welche die Transaction Requests verarbeiten.**

- **Der Program Manager ist zuständig für das Laden von Anwendungsprogrammen in den Hauptspeicher sowie deren anschließende Ausführung. Auch wenn ein Programm von mehreren gleichzeitig laufenden Transaktionen benutzt wird, befindet sich nur eine einzige Kopie des Programms im Hauptspeicher.**

- **Der Storage Manager ist zuständig für die Zuordnung von (virtuellem) Speicherplatz, der für die Transaktionsverarbeitung benötigt wird.**

Terminal Manager. Task Manager. Program Manager, Storage Manager und File Manager wurden früher als Terminal Control, Task Control Program Control, Storage Control und File Control bezeichnet.

8.3.5 Task Manager

Eine Task ist die Ausführung einer Transaktion durch CICS. Während der Ausführung hat die Task die Verfügungsgewalt über eine CPU. Wenn die Task von CICS die Ausführung eines Dienstes verlangt, (z. B. Einlesen von Daten) wird sie in den Wartezustand versetzt.

Während diese Task wartet, wird die nächste ausführbare (ready to run) Task vom CICS Nucleus gestartet. Auf diese Art bewirkt CICS Multitasking innerhalb seiner Region ohne Mitwirkung des Scheduler/Dispatchers des Betriebssystem Kernels. Dies ist der Unterschied zu der Verarbeitung von normalen Threads innerhalb eines Address Spaces.

Wenn eine Task erstmalig gestartet wird (dispatched for execution), ruft der Task Manager als erstes den Program Manager (PG) auf. PG ruft das erste Anwendungsprogramm (von möglicherweise mehreren Anwendungsprogrammen) auf, das für die Ausführung der Transaktion benötigt wird.

8.3.6 Program Manager

Um das gewünschte Anwendungsprogramm aufzufinden, durchsucht der Programm Manager (PG) den Processing Program Table (PPT) nach dem Namen des Moduls, welches als nächstes aufgerufen werden soll.

CICS lädt lediglich eine einzige Kopie des Anwendungsprogramms in den Hauptspeicher, auch wenn mehrere Transaktionen den gleichen Anwendungsprogrammcode gleichzeitig benutzen wollen. Damit das funktioniert, müssen Anwendungsprogramme quasi-reentrant geschrieben werden.

- **Wenn sich das gewünschte Anwendungsprogramm bereits im virtuellen Speicher befindet, ruft PG es auf, und ermöglicht die Ausführung wie gewünscht.**

- **Wenn sich das gewünschte Anwendungsprogramm nicht im virtuellen Speicher befindet, lokalisiert PG es in der CICS Program Library, lädt es in den virtuellen Speicher und ruft es auf.**

Wenn das derzeitig laufende Programm terminiert, übernimmt PG wiederum die Verantwortung und entscheidet was als Nächstes zu erfolgen hat.

An der Ausführung einer einzelnen CICS Task können mehrere Anwendungsprogramme beteiligt sein. Die Verarbeitung kann seriell erfolgen (ein Programm nach dem anderen). Alternativ kann ein Programm ein anderes als ein Unterprogramm aufrufen. PG steuert die Übergabe von einem Programm auf ein anderes.

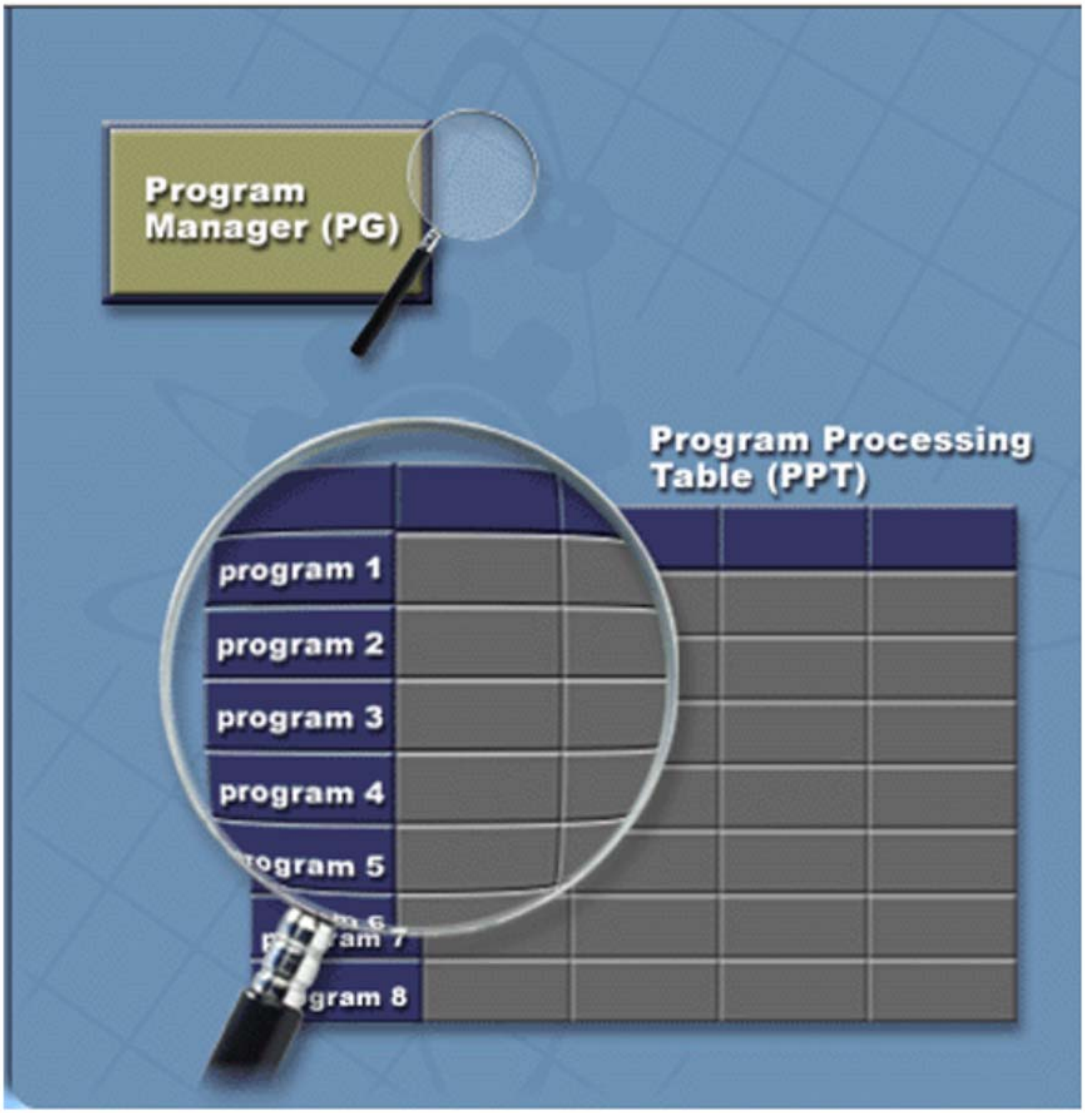


Abb. 8.3.5
Program Processing Table

Der Program Manager ist die zuständige Domäne für:

- **Auffinden des Anwendungsprogramms**
- **Laden des Anwendungsprogramms**
- **Starten des Anwendungsprogramms**

Er benutzt hierfür seinen Program Processing Table (PPT).

8.3.7 Storage Manager

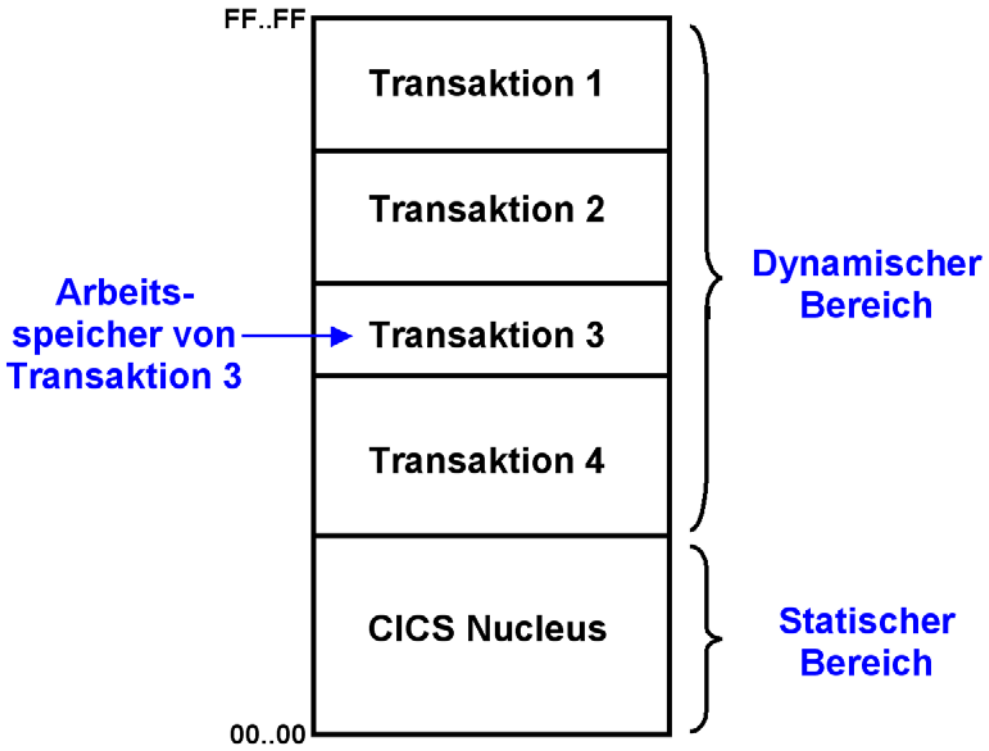


Abb. 8.3.6
Aufteilung des CICS virtuellen Speicherraums

Der Storage Manager (SM) ist die Domäne, die für die Zuordnung (allocation) von virtuellem Speicher verantwortlich ist, der für die Ausführung einer Transaktion benötigt wird.

Unter z/OS verwaltet CICS den virtuellen Speicherplatz seiner Region. Der Storage Manager (SM) verwaltet den dynamischen Bereich des virtuellen Speichers. Dies ist der Teilbereich des virtuellen Speichers, der übrig bleibt nachdem CICS (der CICS Nucleus) geladen wurde.

Dynamischer Speicher wird für Programme, Ein/Ausgabe Bereiche und Arbeitsbereiche genutzt.

Auf Anforderung der anderen CICS Domain Managers bewirkt der Storage Manager die Zuordnung, Freigabe und Verwaltung von verfügbarem virtuellem Speicherplatz.

Vor der Einführung der 64 Bit Adressierung betrug die maximale Größe der CICS Region 2^{31} = 2 GByte. Für tausende von Transaktion muss virtueller Speicherplatz zugeordnet, und nach Abschluss wieder freigegeben werden.

Auch nach der Einführung der 64 Bit Adressierung ist virtueller Speicherplatz eine kritische Ressource. Platz in dem Dynamischen Bereich der CICS Region (Dynamic Storage Area, DSA), muss vom Storage Manager verwaltet und den einzelnen Transaktionen zugeordnet werden. Da der reentrant Program Code von mehreren Transaktionen benutzt werden kann, befindet er sich außerhalb des Arbeitsspeichers einer jeden Transaktion.

8.3.8 COMMAREA

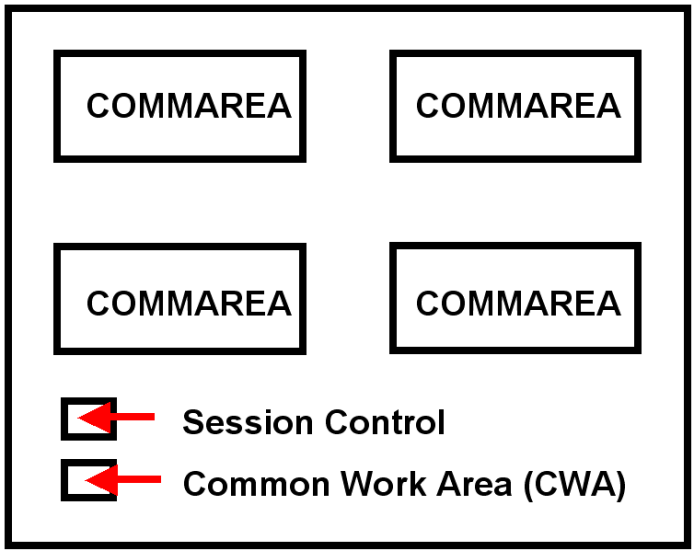


Abb. 8.3.7
Scratchpad

Der Scratchpad-Bereich innerhalb des virtuellen Speichers wird von der Storage Manager Komponente des CICS Nucleus für interne Verarbeitungsabläufe benutzt.

Innerhalb des Scratchpads wird für jeden aktiven Klienten eine COMMAREA eingerichtet.

COMMAREA (Communication Area) ist eine Art Ein/Ausgabepuffer, über den eine Transaktion Daten mit dem Klienten (Terminal) austauschen kann.

COMMAREA ist von so zentraler Bedeutung für die CICS Transaktionsverarbeitung, dass es sich lohnt, den Namen zu merken: COMMAREA

In einem Präsentationslogik-Anwendungsprogramm (z.B. in Java) bezeichnet der Begriff COMMAREA heute fast immer einen Ein-/Ausgabe Puffer.

Der Inhalt eines COMMAREA Ein/Ausgabepuffers wird häufig als Record, Ein-/Ausgaberecord oder Unit Record bezeichnet.

Eine CICS Anwendung kann, muss aber nicht, COMMAREA als I/O Puffer benutzen. Es ist möglich, die Ein/Ausgabe unter Umgehung von COMMAREA zu programmieren. Dies verbessert die Performance, und wurde in der Vergangenheit häufig gemacht. Viele (nicht alle) früher geschriebene CICS Anwendungen machen hiervon Gebrauch.

Heute gilt dies als sehr schlechter Programmierstiel. Praktisch alle in den letzten 10 Jahren geschriebenen CICS Anwendungen nutzen COMMAREA.

Der Execute Interface Block (EIB) ist ein CICS Bereich, auf den Anwenderprogramme mit EXEC CICS Kommandos zugreifen und CICS Informationen abfragen können. Der EIB besteht aus einer Reihe von Feldern. z.B. steht die Länge der COMMAREA in dem Parameter EIBCALEN des EIB.

Für CICS Anwendungsprogramme gibt es zwei Arten, Daten temporär zu speichern: Temporary Storage und Transient Data.

Temporary Storage (TS) ist eine gewöhnliche Datei. TS ist als VSAM-Entry Sequenced Data Set (ESDS) organisiert und wird von CICS unter dem Namen DFHTEMP angesprochen

DFHTEMP wird als Queue benutzt. Entweder ist dies eine einzige TS Queue, auf die viele Programme zugreifen. Häufiger sind Situationen, bei denen für jeden Benutzer eine benutzerspezifische Queue angelegt wird. Anlegen und Löschen einer TS-Queue ist Aufgabe des Anwendungsprogrammierers. Das Cobol Statement

EXEC CICS WRITEQ TS QUEUE('ACCTLOG') FROM(ACCTDTLO) LENGTH(DTL-LNG) END EXEC

ist ein Beispiel für die Nutzung der TS Queue. TS Queue Sätze sind bis zu 32767 Bytes lang.

Transient Data (TD) ist ebenfalls VSAM-ESDS organisiert und wird von CICS unter dem Namen DFHINTRA angesprochen

Die Zwischenspeicherung in der TS-Queue bietet sich an, wenn Daten innerhalb von CICS übergeben werden sollen. Transient Data (TD) sind vorgesehen, um Daten zwischen CICS und seiner Umgebung auszutauschen.

8.3.9 File Manager

Das File Manager CICS Module ist für den Zugriff auf Daten zuständig, welche in Data Sets gespeichert sind, z.B.:

- **VSAM files**
- **Temporary Storage Queues**
- **Transient Data Queue**

Bei einem Datenbank Zugriff benutzt CICS Locking Einrichtungen der Datenbank. Derartige Einrichtungen sind bei VSAM nicht vorhanden. Sie werden deshalb von dem CICS File Manager zur Verfügung gestellt.

Neben Datenbanken kann CICS mit Daten arbeiten, die in VSAM Data Sets gespeichert sind. Hiervon wird häufig Gebrauch gemacht. VSAM (im Gegensatz zu DB2) beinhaltet jedoch keine Funktion die es ermöglicht, Files von unabhängigen Anwendungen in unterschiedlichen CICS Regions gemeinsam zu nutzen (share), mit voller Update Fähigkeit. Der File Manager (mittels seiner DFSMS RLS Komponente) macht dies möglich.

Die beiden wichtigsten CICS Module, die bei einem VSAM Datenzugriff beteiligt sind, sind der File Manager und der File Control Table (FCT).

Der File Manager benutzt die Daten in der File Control Table (FCT) für einen Zugriff auf den Dataset. Jeder FCT Eintrag enthält:

- **den Namen des Data Sets,**
- **die benutzte Access Method (z.B. VSAM),**
- **die mögliche Zugriffsart (z.B. update, read-only),**
- **andere Data Set Attribute, z.B. die maximale Anzahl von Tasks, die gleichzeitig auf den Data Set zugreifen können.**

8.3.10 Weitere Funktionen

Eine CICS Region besteht aus mehreren Domains, von denen jede ihre eigenen Resourcen und Funktionen hat.

Domains kommunizieren miteinander über Schnittstellen, die als "Gates" bezeichnet werden.

CICS verfügt über Domains, die für die eigentliche Transaktionsverarbeitung zuständig sind (z.B. die Storage Manager, Task Manager und Program Manager Domains), Daneben existieren CICS Domains, die zuständig sind für

- **das Laden von Anwendungsprogrammen (Loader Domain)**
- **Dispatching CICS Messages (Message Domain)**
- **Interfacing mit externen Security Systemen (Security Manager Domain)**

Zwei sehr wichtige Domains sind die Domain Manager Domain sowie die Application Domain. Die Domain Manager Domain unterhält einen Katalog mit Kenndaten über alle anderen Domains.

Die Application Domain enthält einige Schlüssel-Komponenten, wie:

- **Application und System services,**
- **Extended recovery facility (XRF) für Fehlersituationen,**
- **Intercommunication Segmente wie Multiregion operation (MRO) und Inter-System Communication (ISC) Systemsteuerung (wird in Abschnitt 9.3 besprochen).**

8.4 CICS Ablaufsteuerung

8.4.1 Bildschirm Wiedergabe

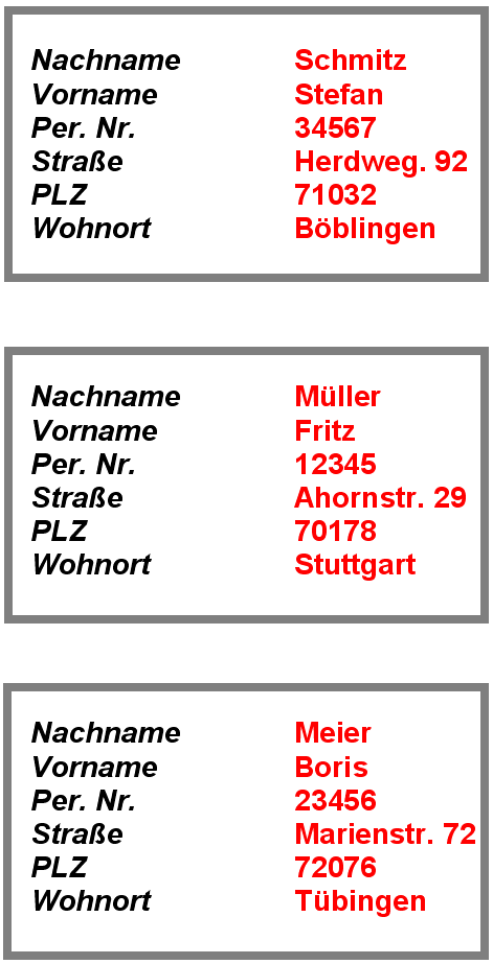


Abb. 8.4.1
Statische und dynamische Bildschirm Information

Vorgefertigte Daten, die bei einer Bildschirmausgabe des gleichen Typs immer wieder identisch sind, werden als „MAP“ bezeichnet.

In Abb. 8.4.1 sind drei Bildschirmwiedergaben dargestellt, welche die gleiche Map benutzen.

Die in schwarzen Buchstaben dargestellte Information ist Bestandteil der Map. Sie dient CICS als Schablone für die auszugebende Information.

Die in roten Buchstaben dargestellte Information wird von CICS bei unterschiedlichen Anfragen erzeugt und in Felder eingestellt, die innerhalb der Map hierfür vorgesehen sind.

Eine bestimmter Typ einer Transaktion verwendet in der Regel mehrere Maps (z.B. zwei bei der NACT Transaktion, wesentlich mehr bei den meisten Transaktionen).

Die Summe aller Maps eines Transaktions-Typs werden als Mapset bezeichnet.

Für den Benutzer am Bildschirm besteht eine Transaktion in der Regel aus mehreren Schritten.

Der Benutzer ruft beispielsweise CICS auf, identifiziert sich und gibt eine TRID (einen Transaktionstyp) ein, trifft eine Auswahl zwischen mehreren Alternativen aus einem Auswahlmenu, und erhält schließlich eine Antwort.

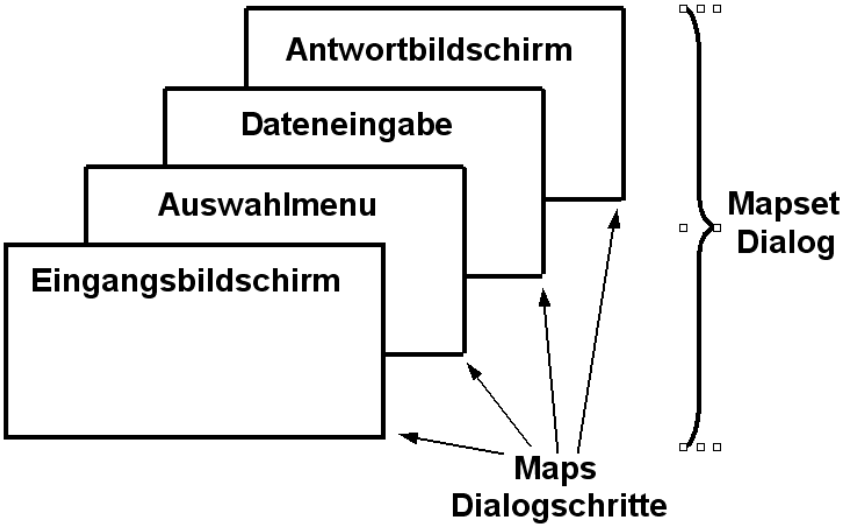


Abb. 8.4.2
Die verschiedenen Maps einer CICS Anwendung bilden den Mapset

Der statische Inhalt eines Bildschirms wird als „Map“ bezeichnet (Dialogschritte bei SAP R/3). Eine Transaktion wird in der Regel mehrere unterschiedliche Bildschirme (Maps) benutzen.

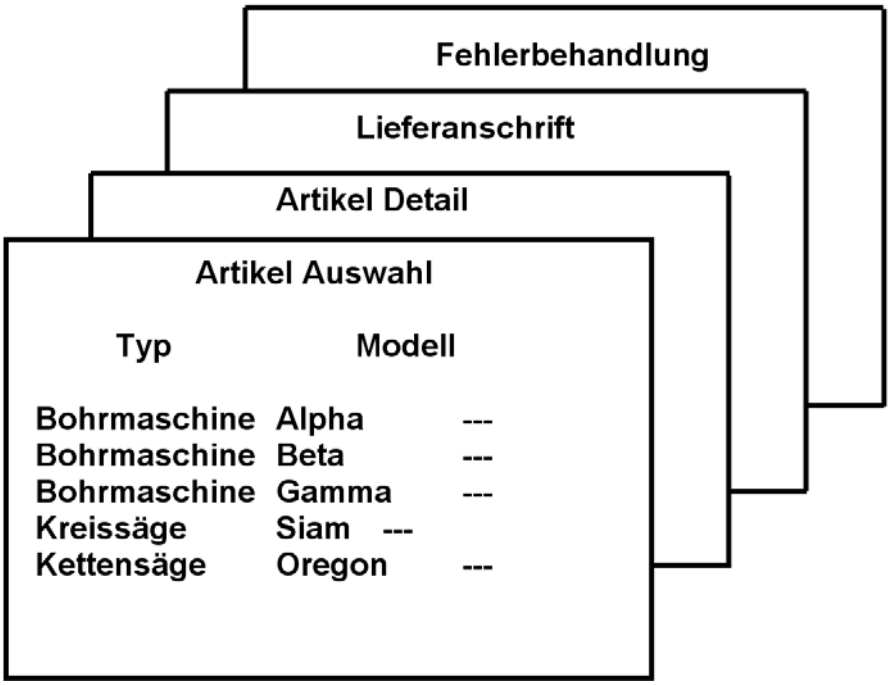
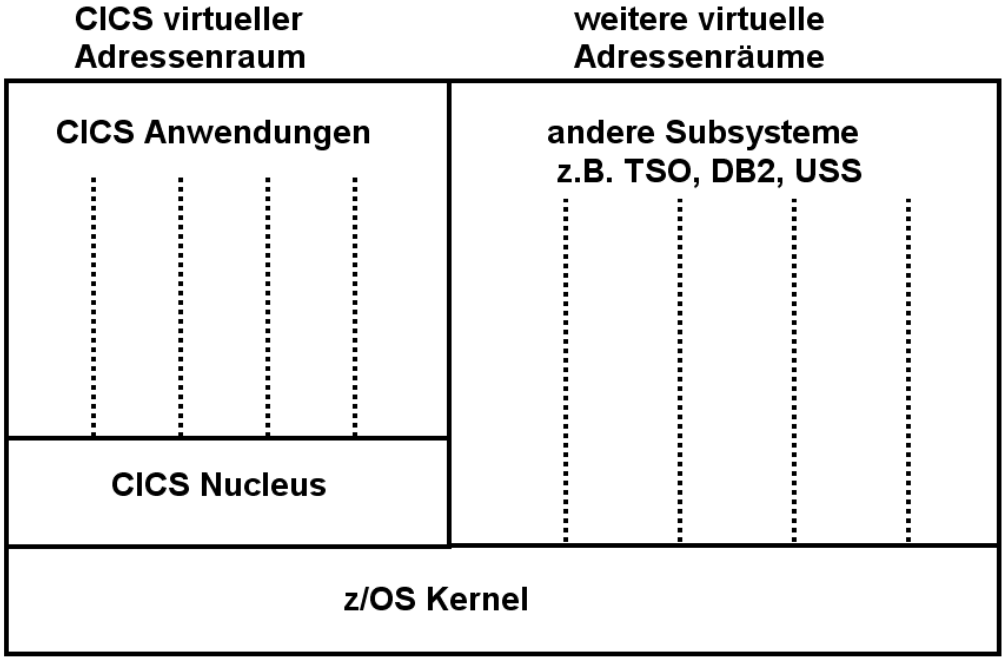


Abb. 8.4.3
Beispiel eines Mapset für einen Lieferauftrag, bestehend aus 4 Maps

Eine Map enthält ein Gerüst generischer Information. Dieses wird während der Transaktionsausführung mit spezifischer Information angereichert. Alle zu einer Transaktion gehörigen Maps werden als Mapset bezeichnet (Dialog bei SAP R/3).

8.4.2 CICS Erstellung der Bildschirm Ausgabe



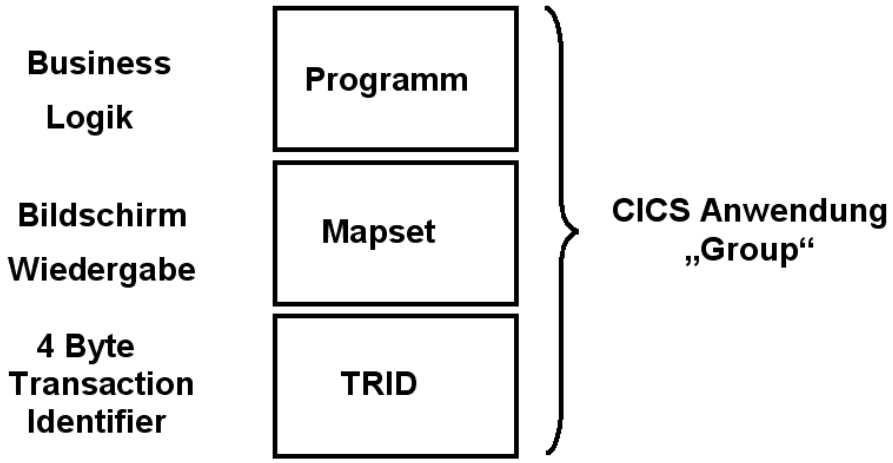


Abb. 8.4.4
Bestandteile einer Group

Die Erstellung der Ausgabe erfolgt in 3 Schritten:

1. **Die richtige Map des Mapsets laden (statische Information)**
2. **Den Mapset mit dynamischer Information ergänzen.**
3. **Beides an den Terminal (Klienten Rechner) senden.**

Die Ausführung der Transaktionsverarbeitung bewirkt die Erstellung von dynamischer Information. Die statische Information liegt in Form einer vorgefertigten Map bereits vor.

Anwendungen, die auf einem Transaktionsserver laufen, werden normalerweise in einer Entwicklungsumgebung erstellt, die nicht auf dem Server läuft.

Eine CICS Anwendung wird ebenfalls außerhalb von CICS entwickelt. Alle Komponenten der Anwendung werden als „Group“ zusammengefasst. Die Group erhält einen Namen, der CICS bekannt gegeben wird.

Anschließend wird die Group in der CICS Programm Bibliothek installiert. Eine einfache Anwendung besteht aus 3 Teilen: Programm, Mapset und Transaction ID (TRID).

Die Anwendungsentwicklung kann auf einem externen Rechner oder in einer anderen LPAR erfolgen. Beispiele sind die CICS Anwendungsentwicklung unter TSO und ISPF, unter dem z/VM Betriebssystem oder unter Windows und Eclipse mit den RDz (Rational Developer for System z) plugin. Im letzteren Fall erzeugt ein Crosscompiler z/OS executable code.

8.4.3 Entwicklung und Installation einer neuen CICS Anwendung

CICS ist ein logischer Server, der (neben anderen logischen Servern) auf einem physischen z/OS Server Läuft. Auf CICS laufen nur fertige Anwendungen; neue Anwendungen können nicht auf dem CICS Transaction Server entwickelt werden.

Ähnliches gilt für Java Anwendungen, die unter z/OS typisch auf einem Web Application Server laufen. Ein weiteres Beispiel sind MQSeries Anwendungen (Kapitel 10), die unter z/OS unter einem „Queue Manager“ ausgeführt werden.

Für die Entwicklung neuer CICS Anwendungen braucht man eine Entwicklungsumgebung. In unserer ersten CICS Übungsaufgabe entwickeln wir eine einfache Hello World CICS Anwendung unter TSO und installieren sie dann unter CICS, um sie dort auszuführen. Andere populäre Entwicklungsumgebungen für CICS Anwendungen sind z/VM und besonders Eclipse mit dem RDz (Rational Developer for System z) Plugin.

Alle unter CICS installierten Anwendungen existieren in der Form von Gruppen (Group). Eine Group enthält alle Komponenten, aus denen eine CICS Anwendung besteht. Als Minimum besteht eine Group aus einem ausführbaren CICS Anwendungsprogramm, einer TRID, sowie (falls die CICS interne Präsentationslogik-Komponente benutzt wird) aus einem Mapset.

Die Installation einer neuen Anwendung unter CICS besteht aus den folgenden Phasen:

Mit Hilfe der Definition (define) wird CICS mitgeteilt, dass eine neue Group mit dem Namen xxxx besteht. Zu dieser Group gehören die folgenden Komponenten:

- **Anwendungsprogramm yyyy**
- **Mapset zzzz**
- **TRID aaaa**

Bei der eigentlichen Installation werden

- **die einzelnen Referenzen werden aufgebaut, z.B. neue TRID in TRID Table einfügen,**
- **Laden der Komponenten, z.B. Laden des Anwendungsprogramms in die CICS Programmbibliothek.**

Diese Schritte sind für jede neue CICS-Anwendung erforderlich.

8.4.4 Ablaufsteuerung einer CICS Transaktion

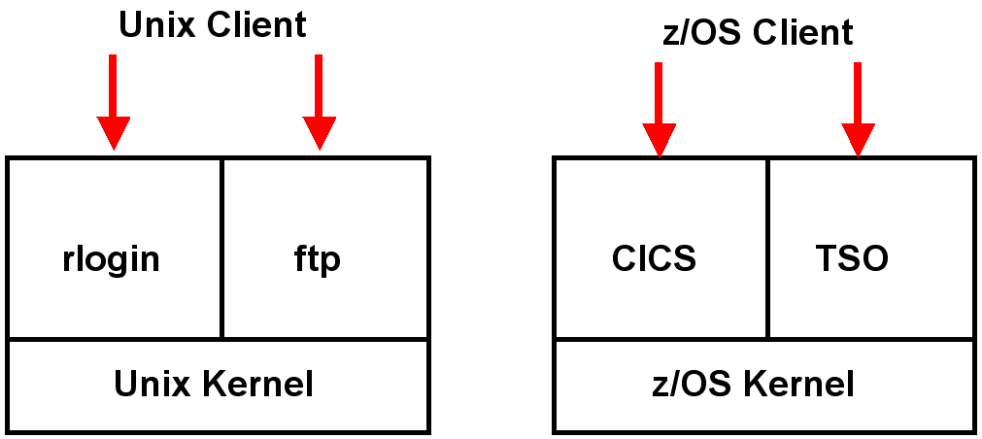


Abb. 8.4.5
Aufbau einer Sitzung

Ein Klient logt sich in das CICS Subsystem ein. (Unter z/OS stehen mehrere Subsysteme für ein remote Login zur Verfügung, z.B. TSO, CICS, IMS, andere ....

Äquivalent besteht z.B. bei einem Unix System die Möglichkeit, sich mit rlogin oder ftp in zwei unterschiedliche Shells einzuloggen.

CICS verfügt über mehrere Shells, von denen CEDA die wichtigste ist.

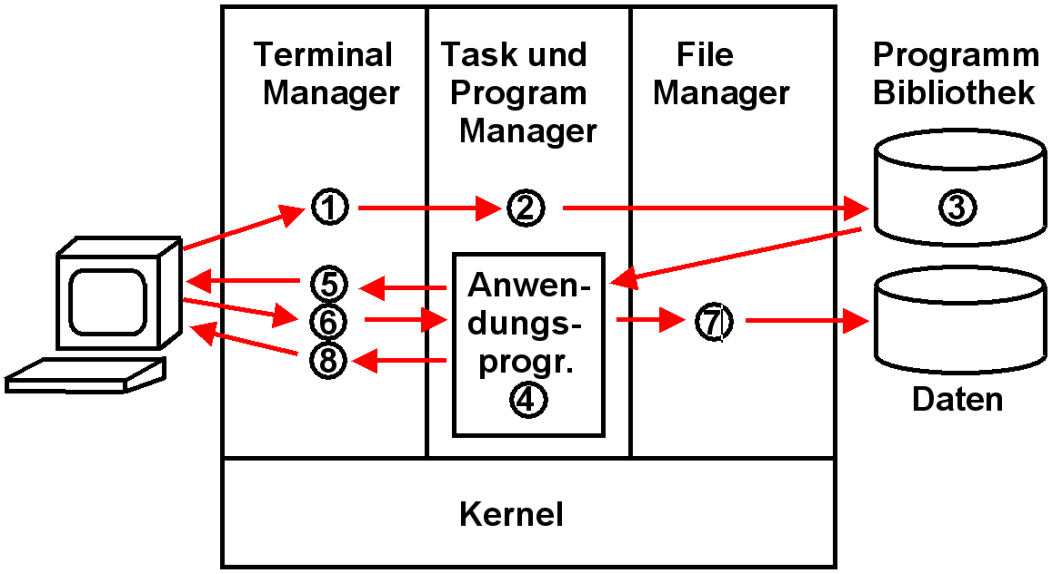


Abb. 8.4.6
Ablaufsteuerung einer CICS Transaktion

Beim Logon erstellt das CICS Subsystem eine Sitzung (Session). Die login Nachricht enthält die Adresse des Klienten. Der CICS Terminal Manager identifiziert den Klienten. Dies geschieht in vielen Fällen einmal pro Tag, und der Benutzer startet danach viele Transaktionen.

Bein Start einer neuen Transaktion verifiziert der Terminal Manager, dass bereits ein gültiges Logon und Authentifizierung vorliegt. Hierfür benutzt Terminal Manager Information, die in seinem Terminal Control Table (TCT) enthalten ist. Dieser enthält einen gültigen Eintrag für jeden eingeloggten Terminal.

- **Eine neue Transaktion ruft CICS mit Hilfe ihrer 4 Byte Transaction ID (TRID) auf. Der CICS Terminal Manager prüft, ob eine laufende Transaktion (eine Session, die fast immer aus mehreren Schritten besteht) mit dem Klienten bereits existiert. Wenn nein, werden die ersten 4 Bytes der Nachrichten als TRID interpretiert. Terminal Manager übernimmt die Eingabenachricht und speichert sie ab**

- **Die TRID am Anfang der Nachricht wird von CICS als solche identifiziert und an Task Manager weitergereicht. (siehe Abb. 8.4.6). CICS interpretiert die Nachricht als Transaktion und ruft das entsprechende Anwendungsprogramm auf.**

- **Das Anwendungsprogramm befindet sich entweder schon im Arbeitsspeicher oder wird vom Program Manager aus der Programmbibliothek geladen. Der Processing Program Table PPT der Program Manager Domain enthält Information über alle CICS interne und alle Benutzer geschriebenen Anwendungen.**

- **Ein CICS Prozess (Task) wird erzeugt. Der Prozess führt das Anwendungsprogramm aus. Information über alle laufenden Transaktionen ist im Program Control Table (PCT) Table festgehalten. Task Manager liest aus seinem PCT Table die zu der TRID gehörige Group aus, darunter Referenzen auf Mapset und Anwendungsprogramm. Das Anwendungsprogramm liest die Nachricht des Klienten.**

- **Terminal Manager baut ein Bildschirm Menü auf (z.B. mit BMS, oder mit Java Präsentationslogik) welche dem Benutzer eine Spezifikation der durchzuführenden Aktivität ermöglicht.**

- **Weitere Eingaben werden von Terminal Manager entgegengenommen und zur Verarbeitung weitergereicht. Die gelesenen Daten (Unit Record) werden von Terminal Manager aufbereitet.**

- **File Manager liest/schreibt gewünschte Daten aus einem VSAM Data Set. Für Zugriffe auf DB2 oder IMS Datenbanken existieren ähnliche SQL oder DL/I Komponenten.**

Wenn vom Klienten die nächste Nachricht eintrifft, erinnert sich CICS Terminal Manager, dass eine Sitzung bereits besteht. Weitere Eingaben werden von Terminal Manager entgegengenommen und zur Verarbeitung Task Manager weitergereicht.

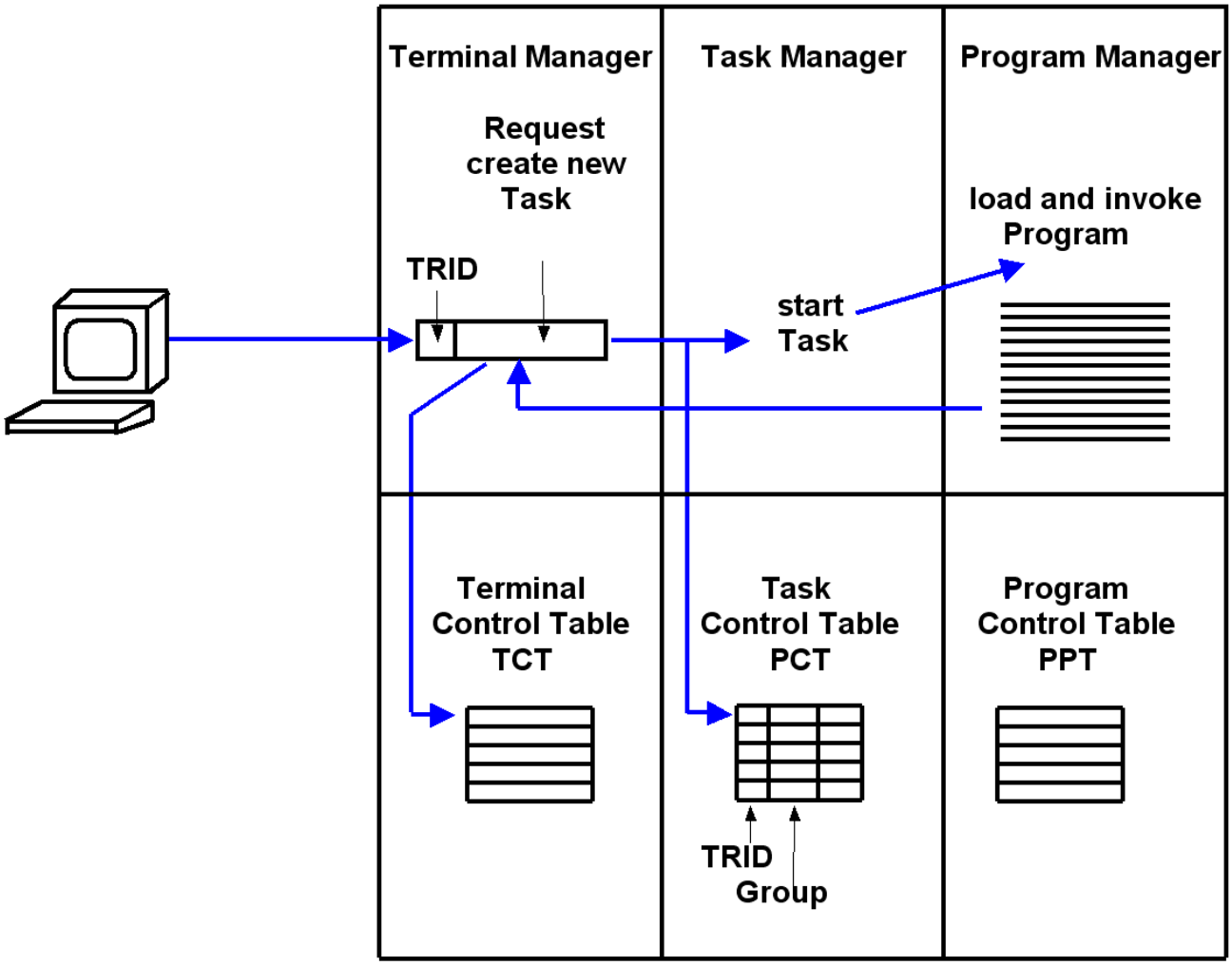


Abb. 8.4.8
Zusammenspiel von Terminal Manager, Task Manager und Program Manager

Task Manager legt für jede TRID, welche in das CICS System eintritt, einen Control Block, (Task Control Area, TCA) als Teil des Task Control Table (PCT) an. Die TCA enthält einen Pointer auf den PCT Eintrag für die TRID und den entsprechenden TCT Eintrag für das Terminal. Task Manager identifiziert an Hand der TRID, um welche von vielen möglichen Transaktionsarten es sich handelt. Für die Ausführung einer bestimmten Transaktionsart sind eine Reihe von Komponenten erforderlich, z.B. mindestens ein Anwendungsprogramm, Kenndaten für die Terminal Ein/Ausgabe wie z.B. ein Mapset (siehe Abb. 8.4.3), die TRID selbst, usw.

Bei Bedarf bittet Task Manager den Programm Manager, das in der Group definierte Anwendungsprogramm aus einer Bibliothek zu laden und startet dann die Transaktion, indem der entsprechende Task Control Block (TCB) in die Warteschlange ausführbarer Prozesse eingereiht wird.

8.4.5 VSAM bzw. Datenbank Zugriff durch ein Anwendungsprogramm

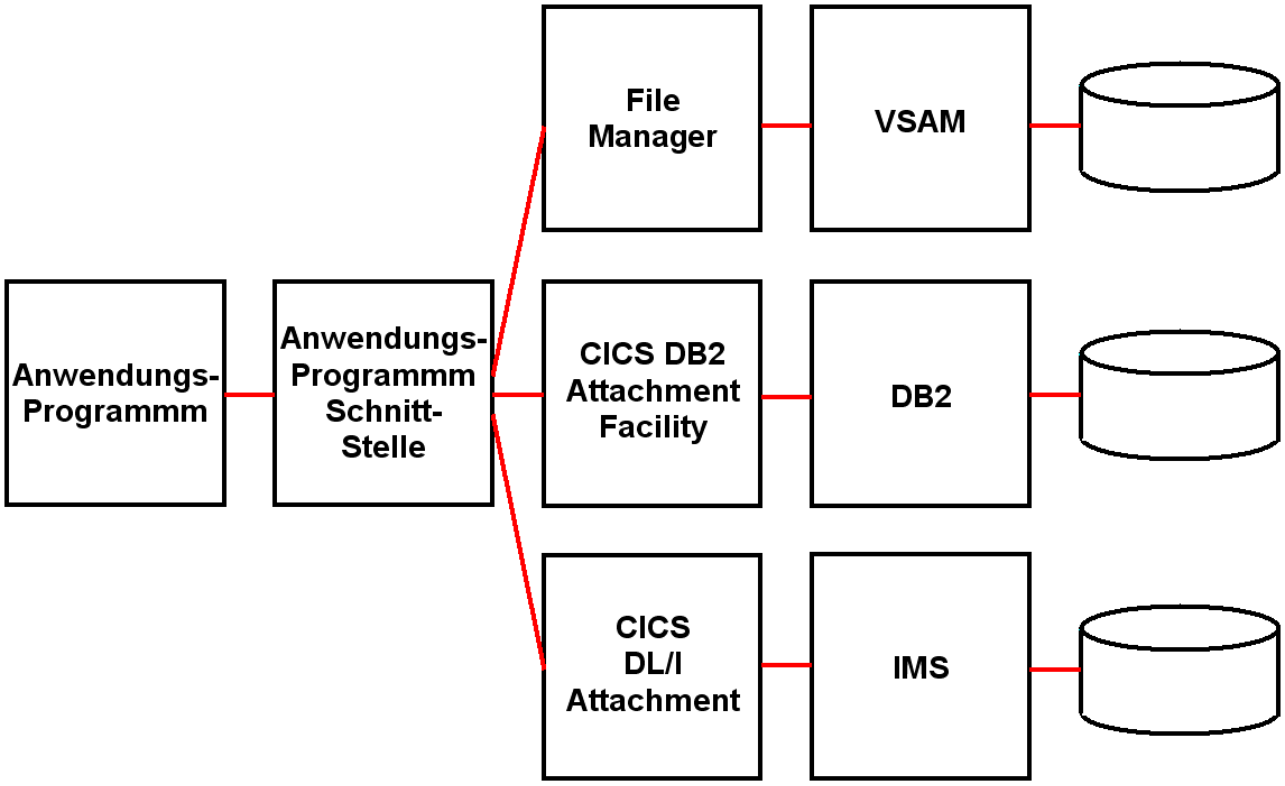


Abb. 8.4.9
Datenbank bzw. VSAM Data Set Zugriff

CICS stellt drei Funktionen für den Datenzugriff zur Verfügung, jeweils mit unterschiedlichem Funktionsumfang. Diese Funktionen müssen von dem Anwendungsprogramm über entsprechende APIs aufgerufen werden.

Ein CICS Programm greift auf DB2 über SQL Befehle zu. Mehrere CICS Module mit dem Namen CICS/DB2 Attachment Facility stellen die Verbindung von der CICS Region zu der DB2 Region her.

Als erster Schritt muss das CICS Anwendungsprogramm eine Connection (Verbindung) zu der gewünschten DB2 Datenbank herstellen. Die CONNECT-Anweisung verbindet eine Anwendung mit einem Datenbankserver. Dieser Server wird der aktuelle Server für den Prozess. Danach kann des CICS Anwendungsprogramm EXEC SQL Befehle wie SELECT, INSERT, UPDATE und DELETE enthalten. Diese EXEC SQL Befehle werden von der CICS/DB2 Attachment Facility in der CICS Region an die DB2 Region zur Verarbeitung weitergereicht. Nach der Verarbeitung gibt DB2 das Ergebnis an das CICS Anwendungsprogramm zurück.

Für IMS benutzt CICS DL/1 Befehle, für VSAM die entsprechenden EXEC CICS READ, WRITE usw. Befehle, siehe Abb. 8.1.9

8.4.6 CICS Shell

Der CICS Application Server braucht zur Bedienung eine Command Line Interface (shell).

Die einzigen Programme, die mit einem Benutzer interagieren, sind Transaktionen, die jeweils durch eine TRID gekennzeichnet sind. Die CICS Shell ist ebenfalls als eine Gruppe von CICS internen Transaktionen implementiert.

Diese CICS internen Shell Transaktionen sind, wie jede andere Transaktion, durch eine 4 Zeichen TRID gekennzeichnet, und werden über diese aufgerufen. Im Gegensatz zu normalen Transaktionen werden sie aber nicht von einem Anwendungsprogrammierer entwickelt, sondern sind schon vorhanden, wenn CICS erstmalig installiert wird.

Die wichtigsten TRIDs dieser Shell Transaktionen sind CEDA, CEDB und CEDC, sowie CESF und CESN für login und logoff. Zu CICS gehören viele weitere interne Transaktionen

Dies ist eine Liste der CICS internen Transaktionen:

CDBC	DBCTL IMS Database Control Menu
CDBI	IMS Database Control Inquiry
CDBM	Database Control Interface
CEBR	Temporary storage browse
CECI	Command-level interpreter
CECS	Command-level syntax checker
CEDA	Dynamic addition of various tables
CEDB	Update CICS CSD data set
CEDC	Interrogate CICS CSD data set
CEDF	Execution diagnostic facility (EDF)
CEMT	All master terminal functions
CEOT	Terminal status
CESF	Signoff (previously CSSF)
CESN	Signon (previously CSSN)
CEST	Inquire or set terminals, lines, control units or tasks
CETR	Trace control facility
CIND	In-doubt Testing Tool
CMAC	Display messages and codes
CMSG	Message switching
CRTE	Route transaction to control units or tasks
CSFE	Terminal test function, trace control and storage freeze
CSPG	Terminal paging
CWTO	Write to operator

Die Mehrzahl dieser Transaktionen werden Sie niemals benutzen

Der CICS System Definition (CSD) Dataset ist eine VSAM KSDS File, welche CICS Resource Definitionen speichert.

8.5 Weiterführende Information

Eine CICS Übersicht finden Sie hier

http://www.cedix.de/VorlesMirror/Band1/IntroandOverview.pdf

und hier ein 140 Seiten Lehrbuch

http://de.scribd.com/doc/48888220/CICS

Hier wird ein Logon zu CICS gezeigt

http://www.youtube.com/watch?v=OFjpRnmMKQ0

Das folgende Video enthält ein Video der IBM Vertriebsorganisation, welches die Vorzüge von CICS anpreist. Interessant ist es vor allem, weil es vor dem Hintergrund des IBM Entwicklungslabors in Hursley (Südengland) gedreht wurde. Das Labor ist in einem landschaftlich wunderschön gelegenen Schloss untergebracht, welches im 19. Jahrhundert von einem englischen Adeligen errichtet wurde.

http://www.youtube.com/watch?v=8KoyvfieQZ8

Zum Thema Cobol

CICS Anwendungen können in vielen Programmiersprachen entwickelt werde, z.B. C++, Java, REXX, Ada, und andere. Die am häufigsten eingesetzte Programmiersprache ist aber nach wie vor Cobol,

Eine Einführungen in Cobol ist zu finden unter

http://www.csis.ul.ie/cobol/course/Default.htm

und

http://www.csis.ul.ie/cobol/

Information über die Zukunft der Programmiersprache Cobol finden sie unter

http://www.cedix.de/VorlesMirror/Band1/TestOfTime.pdf

Die Programmiersprache Cobol hat den Vorteil der besonders effektiven Darstellung von Zahlen

http://www.youtube.com/watch?v=XvA9J2oL4qM

oder

http://cedix.de/VorlesMirror/Band1/Numeric.pdf

Cobol Software hat die Reputation, besonders wartungsfreudig zu sein, deutlich besser als alle anderen Programmiersprachen. Siehe hierzu heise online > News > 2011 > KW 49 > :

09.12.2011 17:25 « Vorige | Nächste »

Unterhalt von Java-Software teurer als von Cobol-Programmen

vorlesen / MP3-Download

Zum zweiten Mal hat die US-Firma CAST Software Unternehmenssoftware unter anderem in Hinblick auf Sicherheit, Leistung und Wartbarkeit untersucht. Dabei kamen 745 Anwendungen mit 365 Millionen Zeilen Code aus 160 Organisationen auf den Prüfstand.

J2EE-Anwendungen machen mit knapp 46 Prozent den Löwenanteil der untersuchten Programme aus, gefolgt von 16 Prozent, die verschiedene Techniken mischen. Auf den dritten Platz kommt Cobol-Software mit 11 Prozent. Vertreten sind außerdem die Kategorien .NET, ABAP, C, C++, Oracle Forms, Oracle CRM/ERP und Visual Basic.

Bei der statischen Code-Analyse mit dem hauseigenen Werkzeug "Application Intelligence Platform" ermittelte CAST eine als "technische Schuld" (technical debt) bezeichnete Größe zum Messen der Code-Qualität. Sie beschreibe "die Kosten für die Behebung jener Probleme in einer Anwendung, die das Unternehmen einem ernsthaften Risiko aussetzen."

Am höchsten liegen, so CAST, die durchschnittlichen Schulden bei J2EE-Anwendungen, während Cobol- und ABAP-Anwendungen sehr niedrige Werte haben. Dazu dürfte auch beitragen, dass Cobol-Programme im Durchschnitt wesentlich älter sind als etwa in .NET oder Java geschriebene, sodass viele Bugs bereits entdeckt und beseitigt sind.

http://www.heise.de/ix/meldung/Unterhalt-von-Java-Software-teurer-als-von-Cobol-Programmen-1392849.html

9. CICS Communication

9.1 Das 3270 Protokoll

9.1.1 Business- und Präsentationslogik

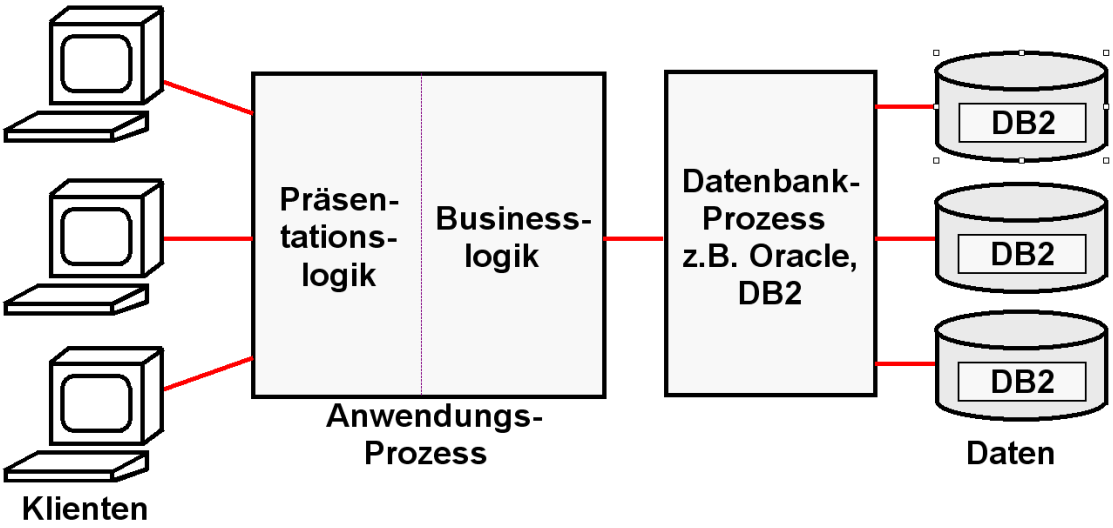


Abb. 9.1.1
Aufteilung des Anwendungsprogramms

Ein sauber strukturiertes CICS Programm besteht aus zwei Teilen: Business Logik und Präsentations-Logik.

Business Logik ist der Teil, in dem Berechnungen erfolgen und Daten in einer Datenbank gelesen/geschrieben werden.

Präsentations- Logik ist der Teil, in dem die Ergebnisse der Berechnungen so aufgearbeitet werden, dass sie dem Benutzer in einer ansprechenden Art auf dem Bildschirm dargestellt werden können.

Business Logik wird in Sprachen wie C, C++, COBOL, PL/1, Java usw. geschrieben.

Für die Präsentations-Logik gibt es viele Möglichkeiten. Die modernste Alternative benutzt Java Servlets und Java Server Pages und einen Web Application Server um den Bildschirminhalt innerhalb eines Web Browsers darzustellen.

Die älteste (und einfachste) Alternative verwendet das CICS BMS (Basic Mapping Support) Subsystem. BMS Programme werden in der BMS Sprache geschrieben.

9.1.2 Endgeräte für die Transaktionsverarbeitung

Es existieren viele unterschiedliche Arten von Endgeräten (Klienten) für die Transaktionsverarbeitung:

- **Arbeitsplatzrechner**

  Browser GUI (Java Swing Classes)
  Windows GUI
  Motiv, KDE, Gnome
  SAPGUI
  3270 CUI

GUI	**Graphical User Interface**
CUI	**Character User Interface**

- **Hand Held Geräte, z.B. Tablet, Mobiltelephon**
- **Geldausgabeautomaten, Kontoauszugsdrucker**
- **Supermarkt Registrierkasse, Tankstellen Zapfsäule**
- **Produktionssteuerungselektronik**

Endgeräte (Klienten) für das Arbeiten mit CICS sind keinesfalls nur Arbeitsplatzrechner mit Tastatur und Bildschirm. Neue Geräte erscheinen ständig. Ein Beispiel ist der Fahrkartenautomat der deutschen Bundesbahn

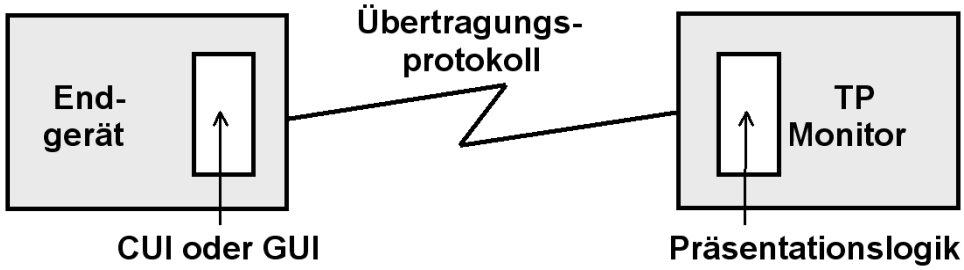


Abb. 9.1.2
Für die Verbindung mit CICS benötigt der Klient eine spezielle Komponente

Die GUI oder CUI ist ein Prozess in den Endgeräten, welcher für die visuelle Ein/Ausgabe zuständig ist. Es ist die Aufgabe der Präsentationslogik, Information von/zu den Endgeräte GUIs oder CUIs in geeigneter Form aufzubereiten.

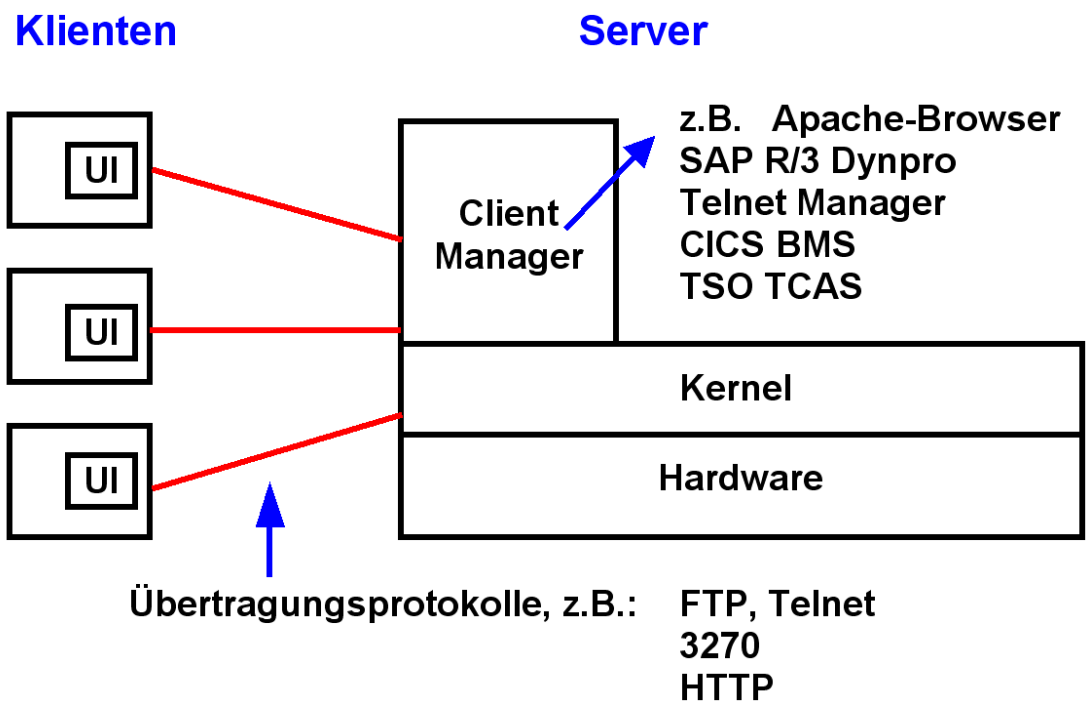


Abb. 9.1.3
Client-Server User Interfaces

Auf der Klientenseite wird eine User Interface (UI) benötigt, z.B. ein 3270 Emulator oder ein Browser.

Auf der Serverseite wird ein Client Manager benötigt, z.B. CICS Terminal Manager oder Apache Web Server.

Für jedes Paar (UI) – Client Manager wird ein entsprechendes Übertragungsprotokoll in Schicht 5 des OSI Modells benötigt. Schicht 3 – 4 benutzen heute meistens TCP/IP.

9.1.3 Alternativen der CICS Bildschirmausgabe

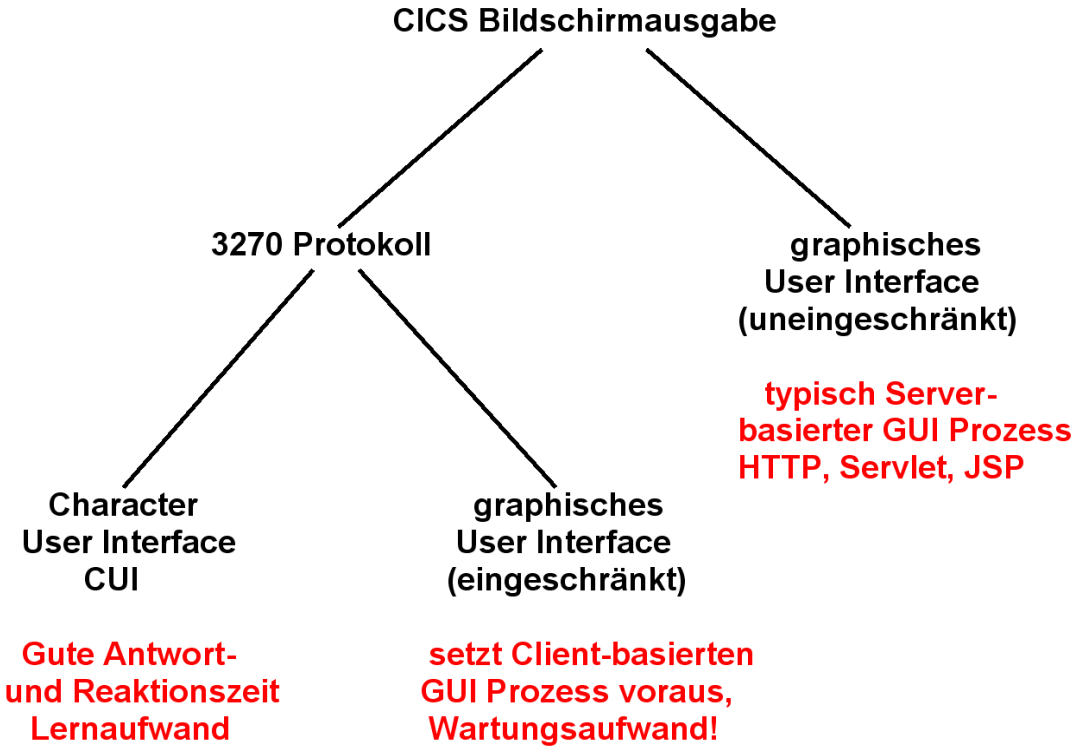


Abb. 9.1.4
existieren zwei unterschiedliche Möglichkeiten für eine grafische Benutzerschnittstelle

Die traditionelle CICS Bildschirmausgabe verwendet den Basic Mapping Support (BMS), sowie das weit verbreitete 3270 Übertragungsprotokoll, welches auch von vielen nicht-IBM Software Produkten verwendet wird, z.B. vom System SAP R/3.

BMS und 3270 können in einer grafischen und der ursprünglichen Zeichen-orientierten Version eingesetzt werden. Die grafische Version ist aber im Funktionsumfang eingeschränkt.

Deshalb existiert eine vom 3270 Protokoll unabhängige grafische Version ohne Einschränkungen.

Für CICS existieren viele Alternativen, mit grafischen Oberflächen unabhängig vom 3270 Protokoll zu arbeiten. Zwei weit verbreitete Alternativen sind das CICS Transaction Gateway (CTG) und die MQSeries CICS Bridge. Auch Web Service Schnittstellen gewinnen an Bedeutung. Einzelheiten hierzu im Abschnitt 18.3, Java Connection Architecture.

Interessanterweise ist die Verwendung der 3270/CUI nach wie vor weit verbreitet. Neu entwickelte Anwendungen stellen häufig neben einer GUI auch eine 3270/CUI zur Verfügung. Viele Benutzer schätzen sie, weil hiermit eine bessere Produktivität möglich ist.

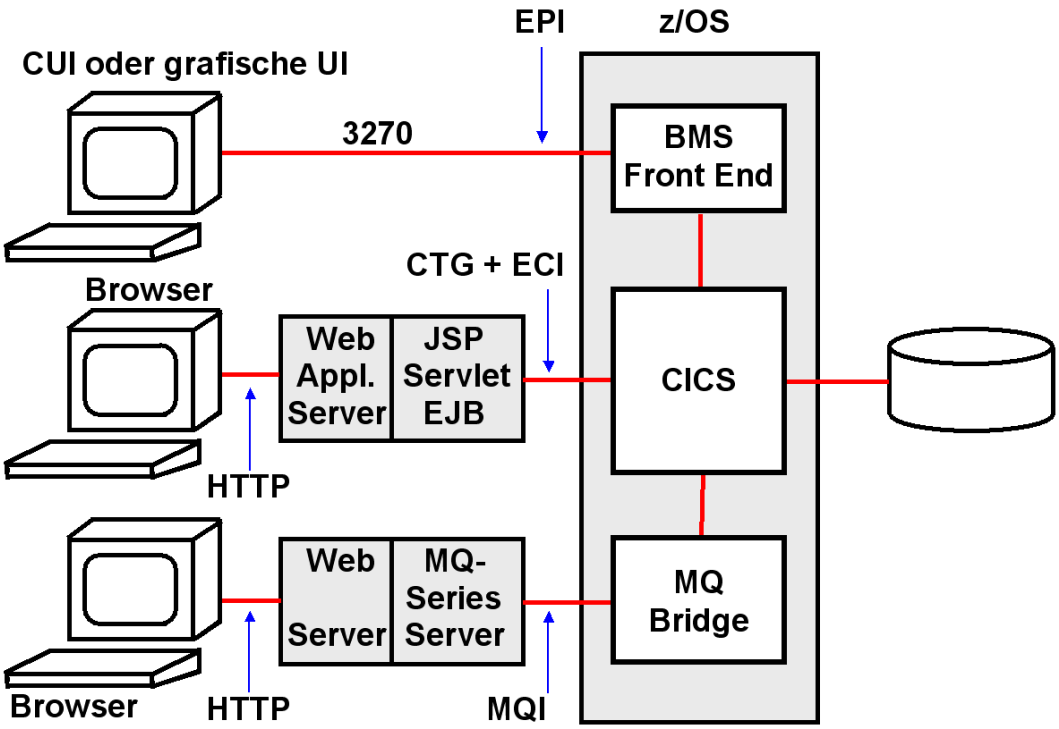


Abb. 9.1.5
Alternativen der CICS Klienten Anbindung

Die hier gezeigten Alternativen benutzen unterschiedliche Schnittstellen:

EPI **Die BMS Maps werden weiter verwendet. Keine Änderung der Information, die auf dem Bildschirm wiedergegeben wird. Die Darstellung der Information kann geändert werden.**

ECI **Die Presentation Service Komponente von CICS (BMS)wird nicht genutzt. Direkter Zugriff auf COMMAREA. Häufig im Zusammenhang mit dem CICS Transaktion Gateway (CTG) benutzt (siehe Kapitel 18).**

MQSeries **Asynchrone Übertragung durch Message oriented Middleware (siehe Kapitel 10).**

Es existieren viele weitere Möglichkeiten, z.B. CICS-Corba Bridge (EJBs) oder SOAP.

9.1.4 3270 Bildschirm

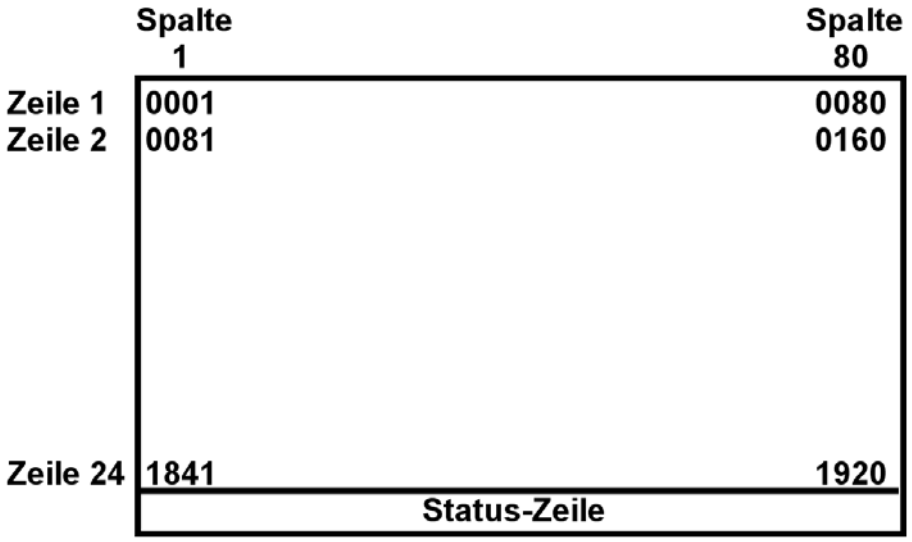


Abb. 9.1.6
Positionen im Bildschirm Puffer (Presentation Space)

Das 3270 Protokoll unterstellt im Terminal einen 1920 Byte großen Bildschirm Puffer, der aus 24 Zeilen mit je 80 Byte besteht. Jedes Byte kann einzeln adressiert werden. Der Bildschirmpuffer wird auch als „Presentation Space“ bezeichnet.

Der 3270 Datenstrom lädt die Information in den Presentation Space. Die interne Logik platziert den Inhalt des Puffers an die entsprechende Stelle des Bildschirms, der hierfür in 80 Spalten und 24 + 1 Zeilen aufgeteilt ist.

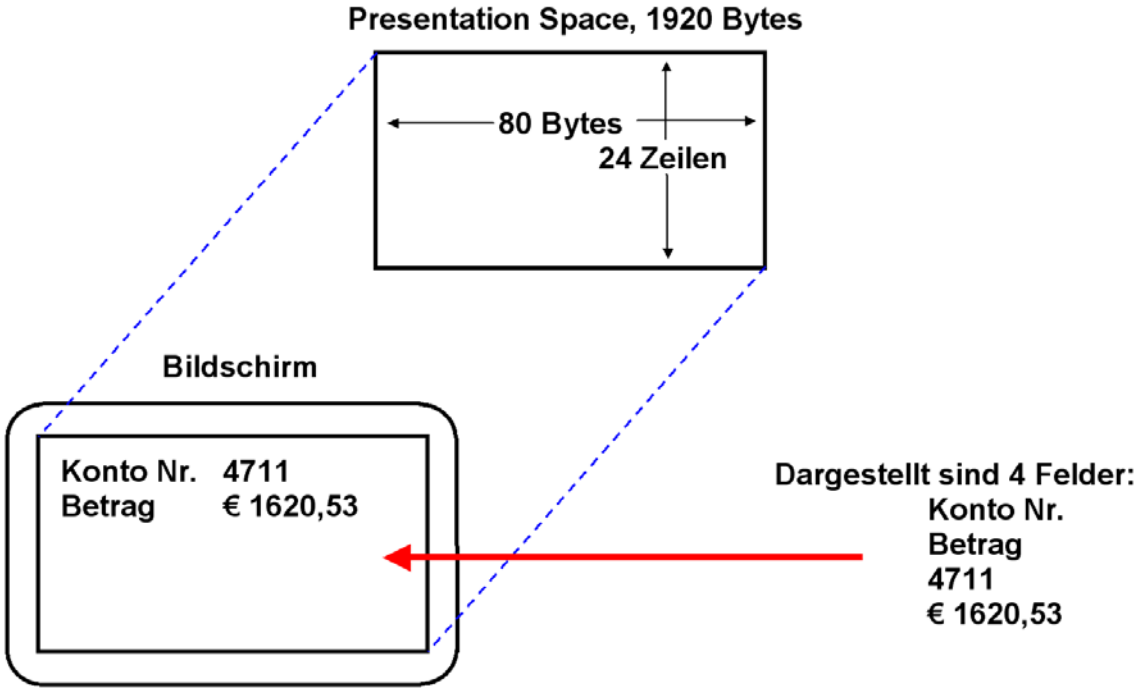


Abb. 9.1.7
3270 Bildschirmdarstellung

Es wird eine Nachricht übertragen, die einen 24 x 80 = 1920 Byte großen Presentation Space mit Character Daten füllt und vom CICS Terminal interpretiert wird.

Der Pufferinhalt wird auf dem Bildschirm als 24 Zeilen mit 80 Zeichen/Zeile wiedergegeben. Jede der 1920 Byte Positionen kann einzeln adressiert werden. In der Regel werden Gruppen von Bytes (Felder) adressiert. Die Felder erscheinen auf dem Bildschirm an der Stelle, an der sie in dem Presentation Space gespeichert sind.

```
ACCOUNTS MENU

   TO SEARCH BY NAME, ENTER SURNAME AND IF REQUIRED, FIRST NAME

      SURNAME     :                          (1 TO 18 ALPHABETIC CHRS)
      FIRST NAME  :                          (1 TO 12 ALPHABETIC CHRS OPTIONAL)

   TO PROCESS AN ACCOUNT, ENTER REQUEST TYPE AND ACCOUNT NUMBER

      REQUEST TYPE:            (D-DISPLAY, A-ADD, M-MODIFY, X-DELETE, P-PRINT)
      ACCOUNT     :            (10000 TO 79999)
      PRINTER ID  :            (1 TO 4 CHARACTERS (REQUIRED FOR PRINT REQUEST))

ACCT    SURNAME       FIRST    MI  TTL   ADDRESS                  ST      LIMIT
26001   Meier         Rolf      A        Ritterstr. 13            N     1000.00
26002   Meier         Stefan    A        Wilhelmstr. 24           N     1000.00
26003   Meier         Tobias    A        Nikolaistr. 23           N     1000.00




ENTER DATA AND PRESS ENTER FOR SEARCH OR ACCOUNT REQUEST OR PRESS CLEAR TO EXIT
```

Abb. 9.1.8
Beispiel eines CICS Basic Mapping Support α/n Bildschirms

Der Basic Mapping Support ist eine CICS Präsentationslogik Komponente, die eine zeichenorientierte (alpha/numerisch, α/n) Bildschirm-Ein/Ausgabe ermöglicht.

9.1.5 3270 Protokoll

Das 3270 Protokoll wurde ursprünglich für nicht-intelligente Terminals eingesetzt.

Es arbeitet mit einem zeichenorientierten Bildschirm, bestehend aus 24 Zeilen mit je 80 α/n Zeichen sowie einem Bildschirmpuffer mit identischer Organisation. Der ursprüngliche 3270 Terminal verfügte hierfür über einen 1920 Byte großen Speicher. Eine Hardware Logik aus diskreten Bausteinen stellte jedes Byte in dem Bildschirmpuffer (Presentation Space) automatisch in die entsprechende Position auf den Bildschirm.

Jede der 24 x 80 = 1920 Positionen ist vom Anwendungsprogramm im CICS Server individuell adressierbar.

Normalerweise werden „Felder“ angesprochen. Ein Feld ist eine Folge von Zeichenpositionen. Felder können gelesen und geschrieben werden

Eine CICS-Utility, Basic Mapping Support (BMS) erleichtert dem Anwendungsprogrammierer die Entwicklung von „Screens“ (Bildschirminhalten).

Auf heutigen Arbeitsplatzrechnern wird die 3270 Bildschirmdarstellung mit Hilfe eines Programms implementiert, welches als „3270 Emulator“ bezeichnet wird. Gelegentlich wird auch die Bezeichnung „3270 Klient“ verwendet. Der 3270 Emulator hat eine Funktion vergleichbar mit dem Telnet Klienten für das Telnet Protokoll auf Unix/Linux Rechnern, z.B. „putty“.

Das Anwendungsprogramm erzeugt die Datenausgabe an den Bildschirm in der Form einer seriell übertragenen Datenstroms mit dem folgenden Format:

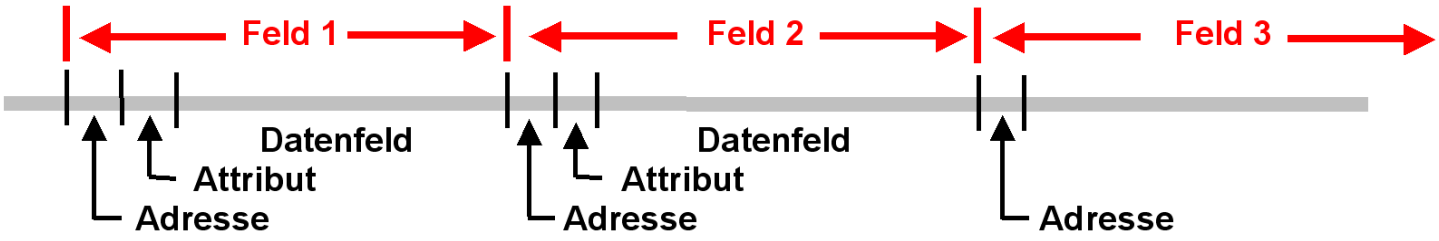


Abb. 9.1.9
3270 Datenstrom

Der 3270 Bildschirm besteht aus 24 α/n Zeilen mit je 80 fixed Font-Width Zeichenpositionen pro Zeile. Das Adressenfeld kennzeichnet eine der 24 x 80 = 1920 Zeichenpositionen auf dem Bildschirm (Zeile und Spalte).

Das Datenfeld enthält eine variable Anzahl von α/n Zeichen, welche auf dem Bildschirm in einem Feld wiedergegeben werden.

Das Attributfeld (3 Bytes bei BMS/CICS) enthält Steuerzeichen, welche Informationen über die Art der Wiedergabe des folgenden Datenfeldes enthalten, z.B. Darstellung in roter Farbe, blinkender Cursor, Font, andere...

Das 3270 Protokoll verwendet eine Untermenge der 256 Zeichen des ASCII oder EBCDIC Zeichensatzes zur Datenwiedergabe auf dem Bildschirm. Die restlichen Zeichen werden als Steuerzeichen für Steuerungszwecke eingesetzt.

Es existieren Sonderversionen für kyrillische, griechische, hebräische usw. Zeichensätze und für die Schreibweise von rechts nach links (z.B. hebräisch).

9.1.6 Entwicklung der Präsentationslogik

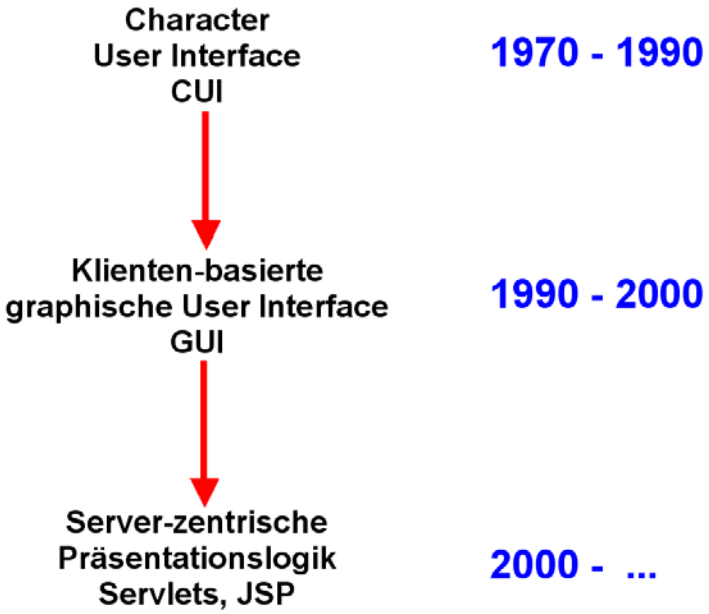


Abb. 9.1.10
Unterschiedliche Entwicklungsstufen

CICS Terminals verwendeten ursprünglich die 3270/CUI Oberfläche und Darstellung. Später kam eine 3270 Protokoll basierte GUI dazu, die ein entsprechendes Umsetzungsprogramm in jedem Terminal erforderte.

Vom 3270 Protokoll unabhängige grafische Versionen entstanden gleichzeitig mit der Entwicklung von Java. Obwohl nicht grundsätzlich erforderlich, wird hierbei die Präsentationslogik in der überwiegenden Mehrzahl der Fälle in Java Servlets und Java Server Pages geschrieben.

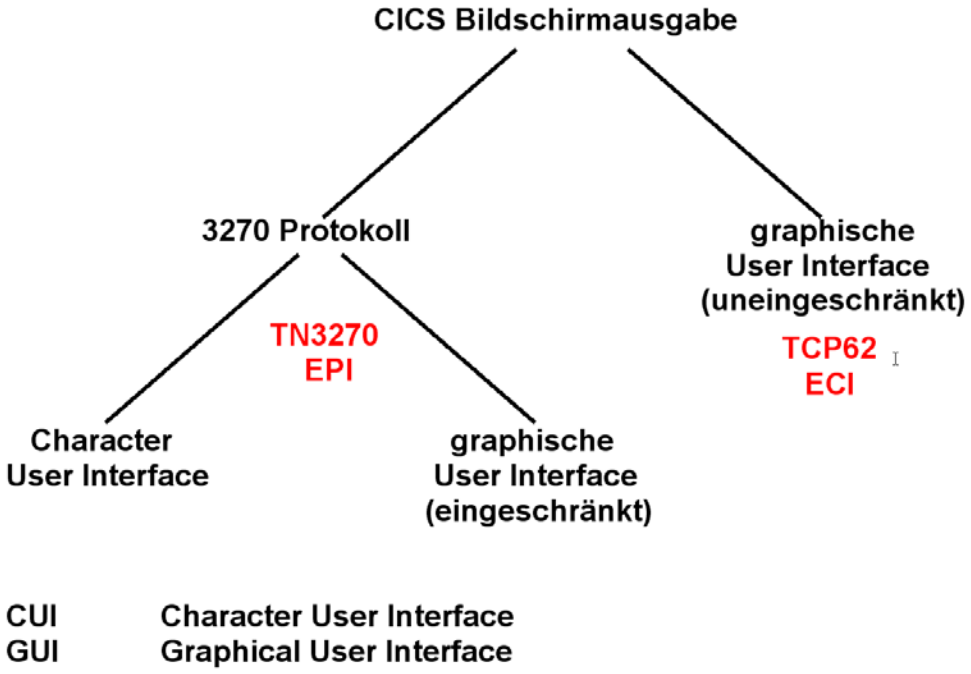


Abb. 9.1.11
Alternativen der Bildschirmausgabe

CICS arbeitete ursprünglich ausschließlich mit dem SNA Protokoll: das 3270 Protokoll ist Bestandteil von SNA. Später wurde eine TCP/IP Version geschaffen, die als TN3270 bezeichnet wird.

Anwendungen auf dem Klienten können mit Hilfe der „External Programming Interface“ (EPI) über TN3270 mit CICS kommunizieren. Dies gilt auch für die eingeschränkte graphische User Interface über 3270.

Die uneingeschränkte graphische User Interface verwendet statt TN3270 das TCP62 Protokoll und statt EPI die ECI (External Communication Interface) Schnittstelle. Die Hintergründe werden in Kapitel 18 diskutiert.

9.1.7 Basic Mapping Support (BMS)

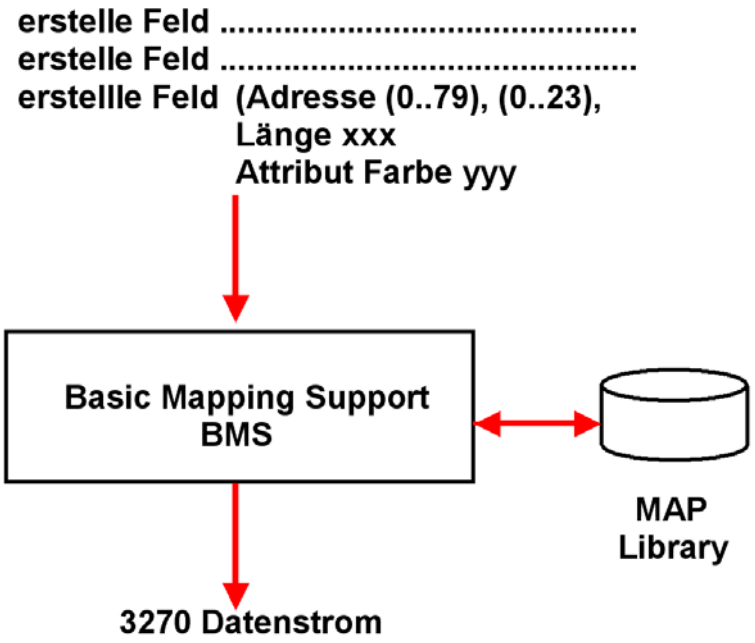


Abb. 9.1.12
BMS Programmierung

Eine Map (auch als Screen bezeichnet) definiert die Wiedergabe von Daten auf dem Bildschirm

Bei einer CUI (Character User Interface) definiert die Map:
Feld Nr. 1 am Ort mit der Adresse aaa,
Feld 2 am Ort mit der Adresse bbb, usw.

wobei aaa und bbb Adressen des 24 x 80 Presentation Spaces sind.

Diese Beschreibung erfolgt mit Hilfe einer eigenen Sprache, der BMS-Sprache.

Bei einer eingeschränkten GUI (Graphical User Interface) redefiniert der GUI Prozess innerhalb eines PC zusätzlich Aussehen und Anordnung der 3270 Datenelemente im Presentation Space Buffer.

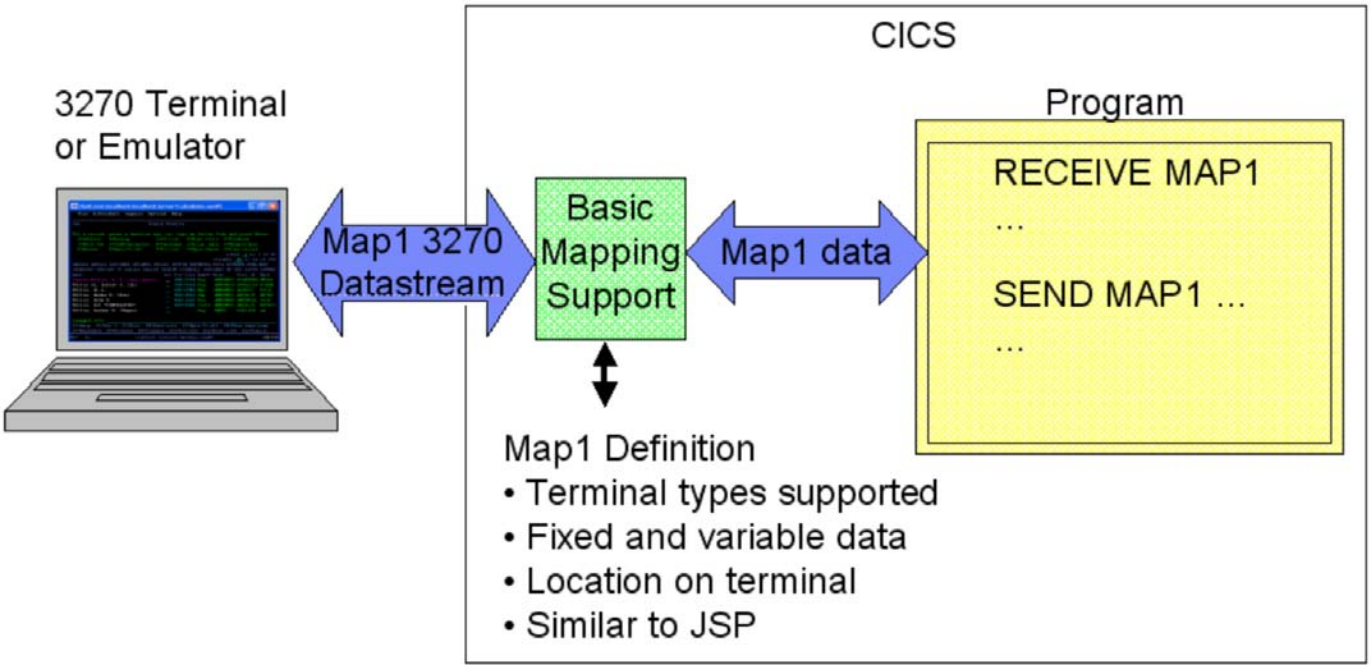


Abb. 9.1.13
BMS ist ein Teil des CICS Terminal Managers

Die zu sendende Map wird mit Hilfe einer eigenen BMS-Sprache beschrieben. Dieser Vorgang wird als Map Definition bezeichnet.

Die so definierte Map wird mit Hilfe der BMS Komponente des CICS Terminal Managers in einen seriellen Datenstrom übersetzt, der mittels des 3270 Protokolls in den Presentation Space Buffer geschrieben wird.

Die CICS Kommandos EXEC CICS SEND MAP und EXEC CICS RECEIVE MAP werden hierfür benutzt.

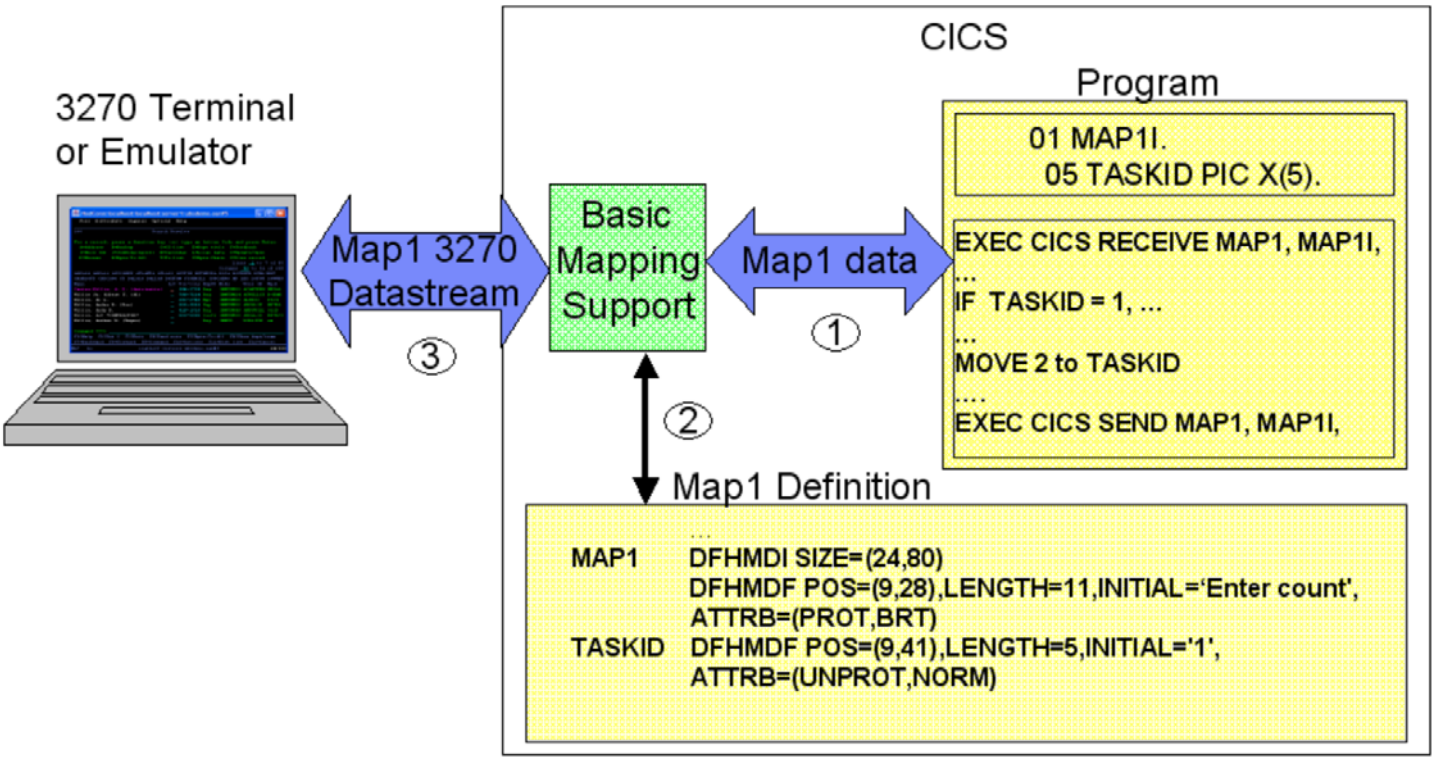


Abb. 9.1.14
BMS Beispiel

Gezeigt wird die Map Definition in der BMS Sprache. Die Ausführung eines EXEC CICS SEND MAP Kommandos bewirkt die Übersetzung der MAP Definition in einen 3270 Datenstrom und dessen Transmission an den Presentation Space Buffer des angesprochenen CICS Terminals.

Die Bildschirmwiedergabe besteht aus einer Menge von Feldern. Diese werden durch zwei Komponenten dargestellt: Map und Inhalt.

BMS (Basic Mapping Support) als Bestandteil von CICS erzeugt und verwaltet „“Maps“. Jede Map beschreibt das Aussehen eines Bildschirm-Fensters des Klienten, spezifisch die Anordnung und die Art (Attribut) von Feldern. Die Felder werden mit statischen oder dynamischen Ausgabedaten (Inhalt) gefüllt. Statische Ausgabedaten sind Teil der Map; dynamische Ausgabedaten werden durch das Anwendungsprogramm erstellt. Je nach Transaktionsart wird die dazugehörige Map in den Klienten geladen.

Jeder Klient enthält ein CUI Programm, welches die Daten auf dem Bildschirm ausgibt. Die Map bestimmt, wie und wo die Ausgabedaten in dem Fenster dargestellt werden.

Das Anwendungsprogramm kann einige der Felder mit Daten füllen. Nach Betätigung der Enter Taste werden diese in der Form einer 3270 Nachricht an den CICS Terminal übertragen.

9.1.8 Benutzung der Maps

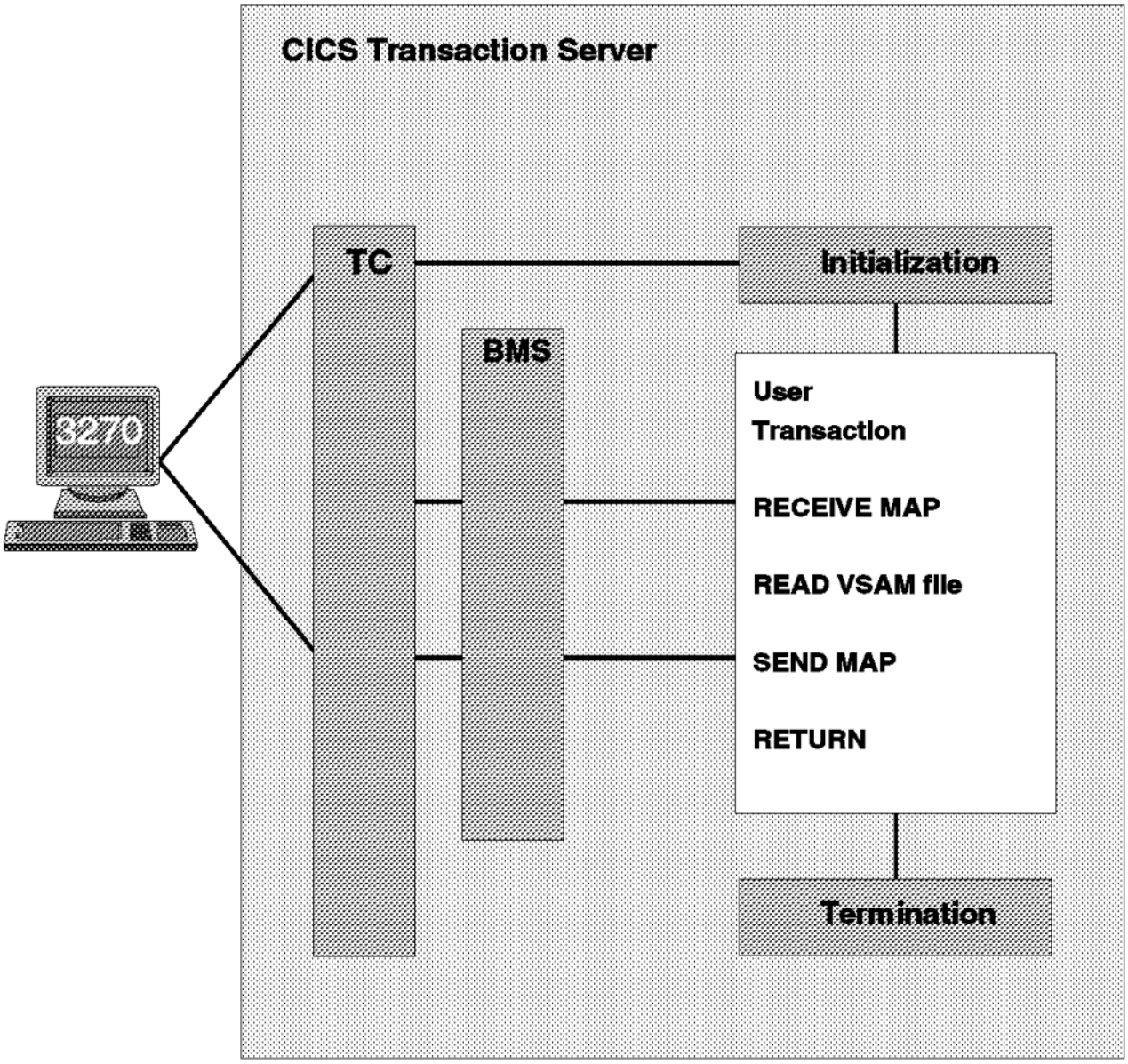


Abb. 9.1.15
Ablauf einer Transaktion

Gezeigt ist der typische Ablauf einer CICS Transaktion.

Die CICS BMS Komponente ist ein Bestandteil des CICS Terminal Managers (Terminal Control, TC).

Nach Initialisierung einer neuen Transaktion liest diese mit Hilfe eines EXEC CICS RECEIVE MAP Kommandos den Inhalt des Presentation Space Buffers.

Bei der anschließenden Verarbeitung wird z.B. ein VSAM Dataset gelesen.

Das Ergebnis der Verarbeitung wird mittels eines EXEC CICS SEND MAP Kommandos an den CICS Terminal zurückgeschickt.

9.2 3270 Bildschirmausgabe Alternativen

9.2.1 Darstellung auf dem Bildschirm

Die Datenübertragung vom /zum CICS Terminal erfolgt normalerweise über einen Input/Output Puffer mit dem Namen COMMAREA.

COMMAREA ist Teil des Scratchpad Bereiches, der vom CICS Storage Manager unterhalten wird.

```
struct adresse { char vorname  [20];
                 char nachname [20];
                 char plz      [20];
                 char ort      [20];
                 char strasse  [20];
                 char tel      [20];
               };

main()
{
...
EXEC CICS RECEIVE MAP(„adresse“) MAPSET(„wgsset“);
...
EXEC CICS SEND MAP(„adresse“) MAPSET(„wgsset“);
...
}
```

Abb. 9.2.1
Beispiel Struktur in C/C++

Die Daten, welche ein Anwendungsprogramm in den COMMAREA I/O Puffer zwecks Ausgabe an den Terminal stellt, werden normalerweise in der Form einer Struktur dargestellt, die oft auch als „Unit Record“ bezeichnet wird.

Abb. 9.2.1 zeigt ein Beispiel in der Programmiersprache C/C++ .

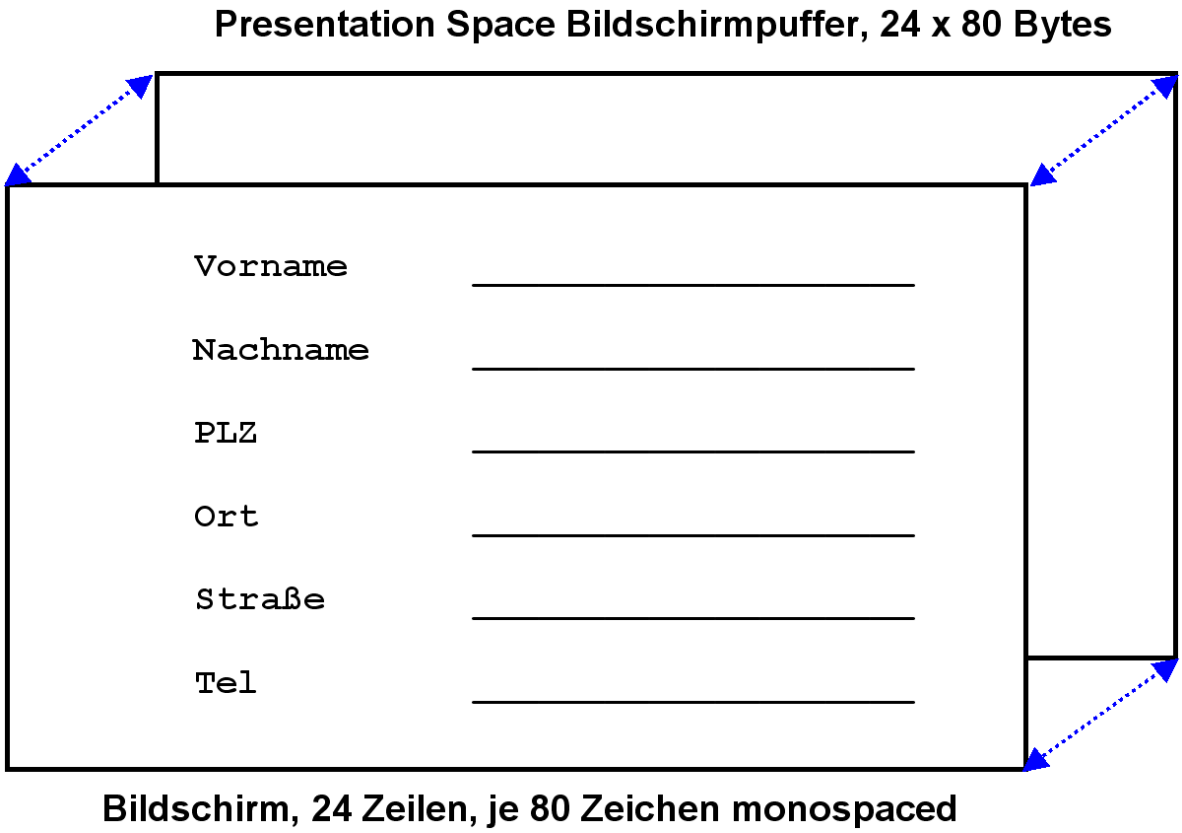


Abb. 9.2.2
Abbildung des Presentation Spaces auf den Bildschirm

Der Presentation Space Bildschirmpuffer hat eine Struktur von 24 Worten zu je 80 Byte. Jede Wortadresse und Byte Adresse innerhalb eines Wortes entspricht genau der Zeilen- und Spaltenadresse. auf der das Byte auf dem Bildschirm wiedergegeben wird.

9.2.2 Kommunikation mit dem Bildschirm

```
#include <//'PRAKT20.LIB(MSET020)'>

main()
{
EXEC CICS SEND MAP("map020") MAPSET("s04set") ERASE;
}
```

Abb. 9.2.3
Hello World in C++

Im Folgenden schauen wir uns ein einfaches CICS Hello World Programm in C/C++ an, einschließlich seiner BMS Präsentationslogik.

Dargestellt ist der CICS Hello World Programm Code. Es wird eine Map mit dem Namen `map020` gesendet, die ein Teil des Mapsets `s04set` ist.

```
File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
 ------------------------------------------------------------------------------
 EDIT       SPRUTH.CICS.TEST04(MAP04) - 01.02               Columns 00001 00072
 ****** ***************************** Top of Data ******************************
 ==MSG> -CAUTION- Profile changed to CAPS ON (from CAPS OFF) because the
 ==MSG>           data does not contain any lower case characters.
 ==MSG> -Warning- The UNDO command is not available until you change
 ==MSG>           your edit profile using the command RECOVERY ON.
 000001 //PREPARE  JOB (),CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID
 000002 //ASSEM    EXEC DFHMAPS,MAPNAME='S04SET',RMODE=24
 000003 //SYSUT1   DD *
 000004 S04SET  DFHMSD TYPE=MAP,MODE=INOUT,LANG=C,STORAGE=AUTO,TIOAPFX=YES
 000005 *       MENU MAP.
 000006 map020  DFHMDI SIZE=(24,80),CTRL=(PRINT,FREEKB)
 000007         DFHMDF POS=(9,23),ATTRB=(ASKIP,NORM),LENGTH=34,                X
 000008                INITIAL='WELCOME TO THE MAGIC WORLD OF CICS'
 000009         DFHMDF POS=(12,27),ATTRB=(ASKIP,NORM),LENGTH=26,               X
 000010                INITIAL='MAY THE FORCE BE WITH YOU!'
 000011         DFHMSD TYPE=FINAL
 000012         END
 000013 /*
 000014 //
 Command ===>                                                 Scroll ===> PAGE
  F1=Help      F3=Exit      F5=Rfind     F6=Rchange  F12=Cancel
```

Abb. 9.2.4
MAP für das Hello World Programm

…und hier ist der ISPF Editor Screen mit der Beschreibung der MAP in der BMS Sprache. Der Mapset unseres Hello World Programms besteht aus einer einzigen Map, die mit Hilfe von EXEC CICS SEND MAP gesendet wird.

9.2.3 BMS Programmierung

```
S04SET  DFHMSD TYPE=MAP,MODE=INOUT,LANG=C,STORAGE=AUTO,TIOAPFX=YES
*       MENU MAP.
map020  DFHMDI SIZE=(24,80),CTRL=(PRINT,FREEKB)
        DFHMDF POS=(9,23),ATTRB=(ASKIP,NORM),LENGTH=34,
               INITIAL='WELCOME TO THE MAGIC WORLD OF CICS'
        DFHMDF POS=(12,27),ATTRB=(ASKIP,NORM),LENGTH=26,
               INITIAL='MAY THE FORCE BE WITH YOU!'
        DFHMSD TYPE=FINAL
(1)                     END
```

Abb. 9.2.5
BMS Programm

Ein BMS Programm verwendet ausschließlich drei Arten von Befehlen, nämlich

DFHMSD	**Data Facility Hierarchical Map Set Definition.** **Mit Hilfe dieses Befehls wird ein Mapset definiert.**
DFHMDI	**Data Facility Hierarchical Map Definition Information.** **Mit Hilfe dieses Befehls wird eine Map definiert.**
DFHMDF	**Data Facility Hierarchical Map Data Field.** **Mit Hilfe dieses Befehls wird ein Feld innerhalb einer Map definiert.**

Der Mapset hat den Namen S04SET, die einzige Map dieses Mapsets hat den Namen map020. Jede Map innerhalb des Mapsets muss durch einen Namen gekennzeichnet sein.

Der Parameter TIOAPFX=YES in dem DFHMSD Befehl fügt ein 12-Byte Feld an den Anfang jeder Map ein.

… und hier verraten wir Ihnen ein Geheimnis: Es gibt überhaupt keine eigene BMS Sprache. Was Sie hier sehen ist in Wirklichkeit ein Assembler Programm, welches ausschließlich aus Assembler Makros besteht, nämlich den drei Makros DFHMSD. DFHMDI und DFHMDF. IBM hat sich die Entwicklung einer eigenen BMS Sprache erspart, und stattdessen Assembler Macros verwendet. EXEC DFHMAPS in der JCL File bewirkt ein Assembler Compile, Link and Go.

Die beiden DFHMDF Befehle in der JCL File definieren 2 Felder in der Map, die mit Hilfe des EXEC CICS SEND MAP Befehls an den Terminal gesendet werden. Diese beiden Felder werden mit statischer Information initiiert. Für das erste der beiden Felder ist dies:

WELCOME TO THE MAGIC WORLD OF CICS

Angegeben wird, in welcher Zeile (Zeile 9) und Spalte (Spalte 23) dieses Feld in den Presentation Space Puffer geladen werden soll.

Es ist grundsätzlich möglich, CICS Maps mit Hilfe von Sprachen wie Cobol, C/C++ oder PL/1 zu erstellen. Dies geschah häufig in der Anfangszeit von CICS, wird heute aber praktisch so gut wie nie mehr gemacht.

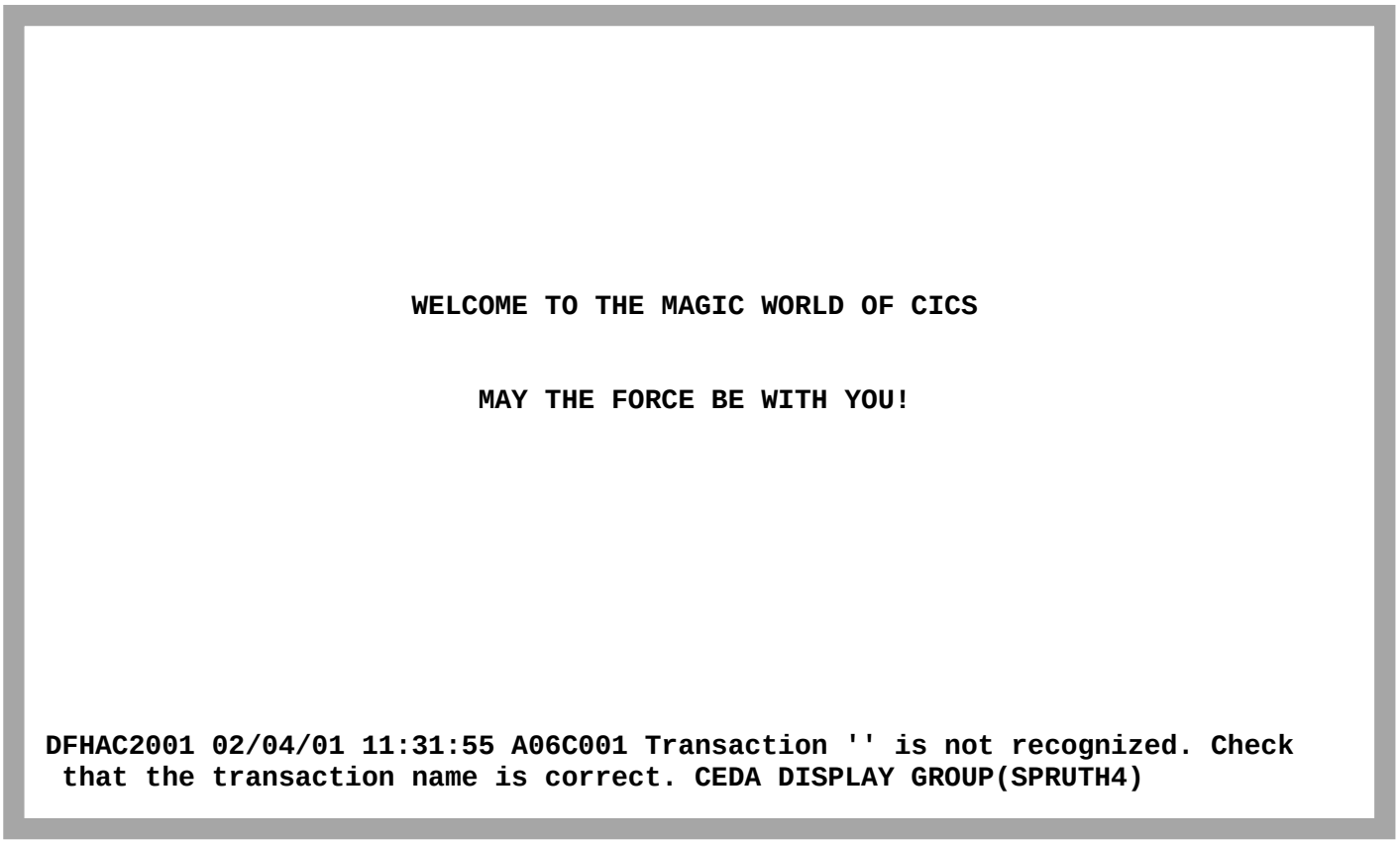


Abb. 9.2.6
Bildschirm Darstellung

… und dies ist das tolle Ergebnis unseres CICS Hello World Programms.

Im Anhang Abschnitt 9.2.8 befindet sich ein erweitertes Mapset-Beispiel mit 2 Maps.

9.2.4 Screen Scraping

CICS Terminals verwendeten ursprünglich die 3270/CUI Oberfläche und Darstellung. Später kam eine 3270 Protokoll basierte GUI dazu, die ein entsprechendes Umsetzungsprogramm in jedem Terminal erforderte.

Mit der Verfügbarkeit von PCs begann man, diese an Stelle der bisherigen Terminals einzusetzen. Dies ermöglichte auf dem Klientenrechner Programme, die den Inhalt des Presentation Puffer für die Erstellung einer (eingeschränkten) graphischen Ausgabe benutzte. Hierfür wurde die EPI (External Programming Interface) Schnittstelle zur Verfügung gestellt.

Diese Art der Verarbeitung wird auch heute noch vor allem für ältere CICS Anwendungen eingesetzt und als Screen Scraping bezeichnet.

Das Klienten Anwendungsprogramm (z.B. in Java geschrieben) empfängt den 3270 Datenstrom über die EPI (External Programming Interface) Schnittstelle und erzeugt eine graphische Darstellung der Datenausgabe. Der Vorteil ist, dass diese Art der Bildschirmdarstellung transparent für die Server-seitigen CICS Anwendungsprogramme ist.

Ein weiterer Vorteil ist eine einfache, schnelle und kostengünstige Umstellung auf eine graphische Darstellung. Es existieren zahlreiche Software Produkte von unterschiedlichen Herstellern, welche die Umstellung erleichtern. Ein Beispiel ist „Host-on-Demand“ von IBM; es existieren viele ISV (Independent Software Vendor) Alternativen (z.B. von der Firma Attachmate).

Erreicht wird eine gefälligere Darstellung, aber es fehlen manche Funktionen. Nicht möglich ist zum Beispiel eine Scroll Funktion mittels eines Scroll Balken, weil im 3270 Protokoll hierfür die Vorraussetzungen fehlen.

Probleme:

- **2000 Benutzer erfordern die Wartung für 2000 Emulatoren auf den Arbeitsplatzrechnern**
- **Schwierigkeiten mit der Wartung der Maps. Maps können nicht mehr bequem geändert werden, z.B. "move field 1 Byte“.**

Eine Server-seitige Lösung umgeht das Administrationsproblem. Z.B. „Host Access Transformation Services“ (HATS) von IBM läuft auf einem Server (z.B. einer zLinux Maschine) und konvertiert eine 3270 Nachricht in Browser-fähiges HTML Format.

9.2.5 Beispiel Computer-Schachprogramm

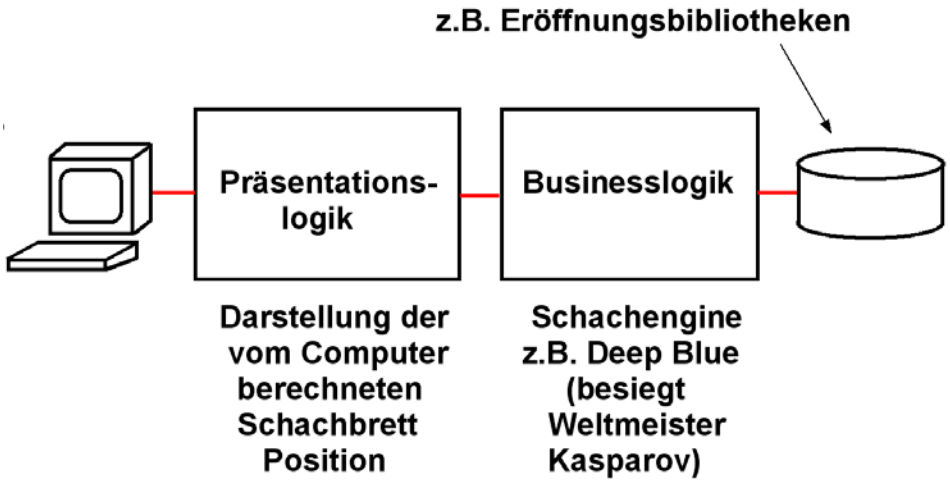


Abb. 9.2.7
Gliederung in Präsentationslogik und Businesslogik

Als Beispiel schreiben wir ein Computer Schachprogramm. (Wir übersehen, dass man ein Mainframe vermutlich nicht zum Schachspielen benutzen würde, und wenn doch, das Schachprogramm wahrscheinlich nicht unter CICS laufen würde).

Die Business Logik läuft als CICS Anwendung. Die Präsentationslogik wird einmal mit Hilfe von BMS erstellt und als 3270/CUI dargestellt, und ein zweites Mal mit Hilfe von Screen Scraping graphisch dargestellt.

```
Mein Zug: c5 x d4

Position:

Ke1, DF3, Ta1, h1, Lc1, e2, Ba2, c3, e5, g2, h2

Ke8, Dd8, Ta8, h8, Lc8, f8, Ba7, b7, d4, f7, g7,
h7

Ihr Zug _______
```

Abb. 9.2.8
Minimale Präsentationslogik

Abb. 9.2.8 zeigt die Darstellung mit Hilfe der BMS 3270/CUI Präsentationslogik. Der Inhalt des Schachbretts ist mit Hilfe der „International Chess Notation“ dargestellt. Wiedergegeben sind die Positionen der einzelnen Figuren auf dem Schachbrett. Der Computer spielt mit den schwarzen Steinen, und eine seiner Figuren auf dem Feld c5 hat gerade eine weiße Figur auf dem Feld d4 geschlagen.

Der dargestellte Bildschirminhalt ist in dieser Form im Presentation Space Puffer des Terminals abgelegt, und wird unverändert auf dem Bildschirm wiedergegeben. Mitglieder eines Schachclubs sind mit dieser Notation vertraut.

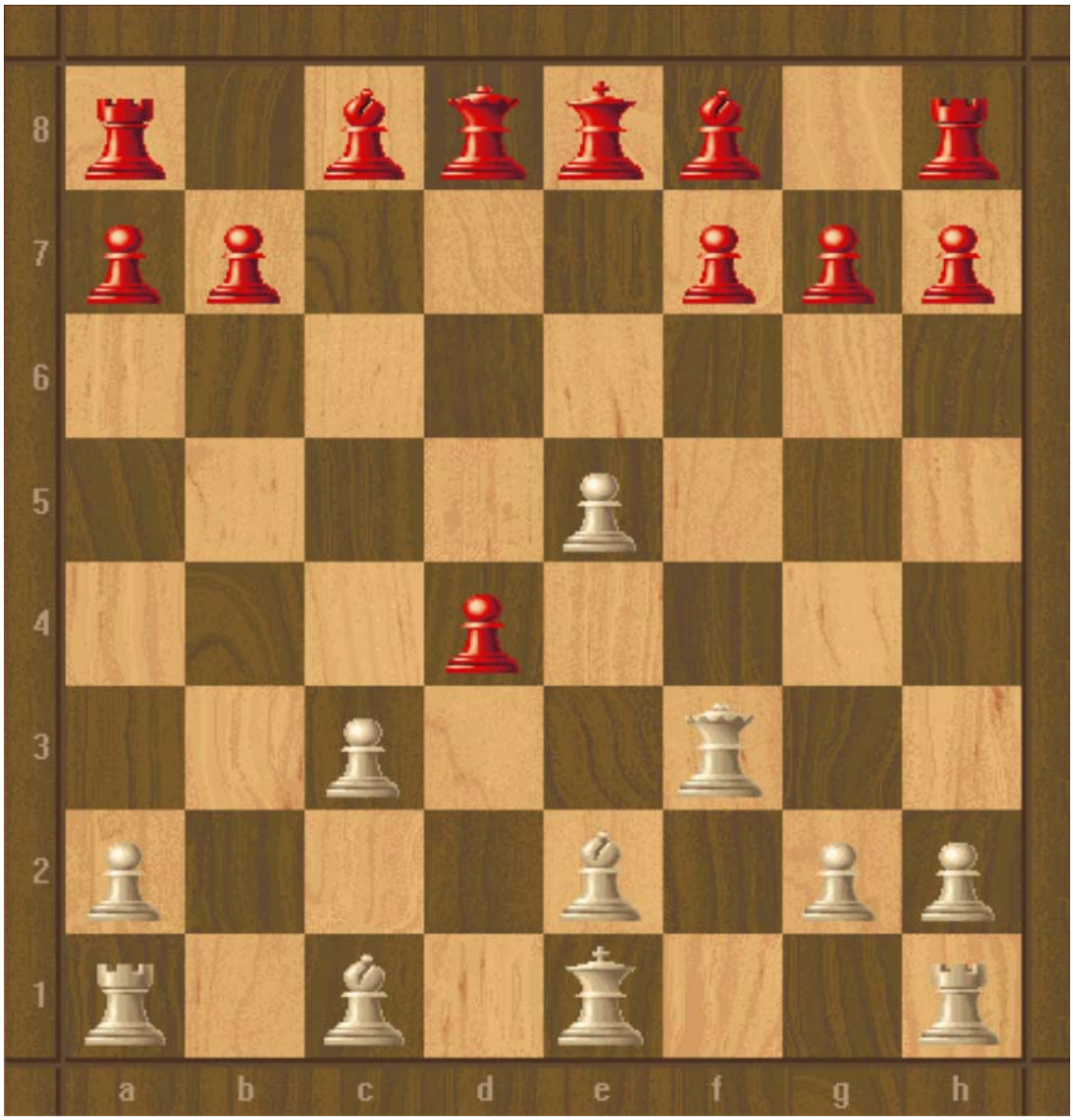


Abb. 9.2.9
Graphische Präsentations- Logik

Alternativ greift ein Java Programm mit Hilfe der EPI Schnittstelle auf den Inhalt des Presentation Space Buffers zu und bereitet die Darstellung, wie in Abb. 9.2.9 gezeigt, graphisch auf.

Wichtig ist, dass die gezeigten beiden Bildschirm-Darstellungen inhaltlich identisch sind.

Es sind zusätzliche Funktionen möglich. Z. B. kann die zuletzt gezogene Figur blinkend dargestellt werden, Figuren können mit der Maus bewegt werden, usw.

9.2.6 Screen Scraping in der Praxis

Unser Schachbeispiel ist sicher sehr einprägsam, aber sicher auch sehr realitätsfern. Ein mehr realistisches Beispiel würde z.B. den in Abb. 9.2.10 gezeigten Screen produzieren.

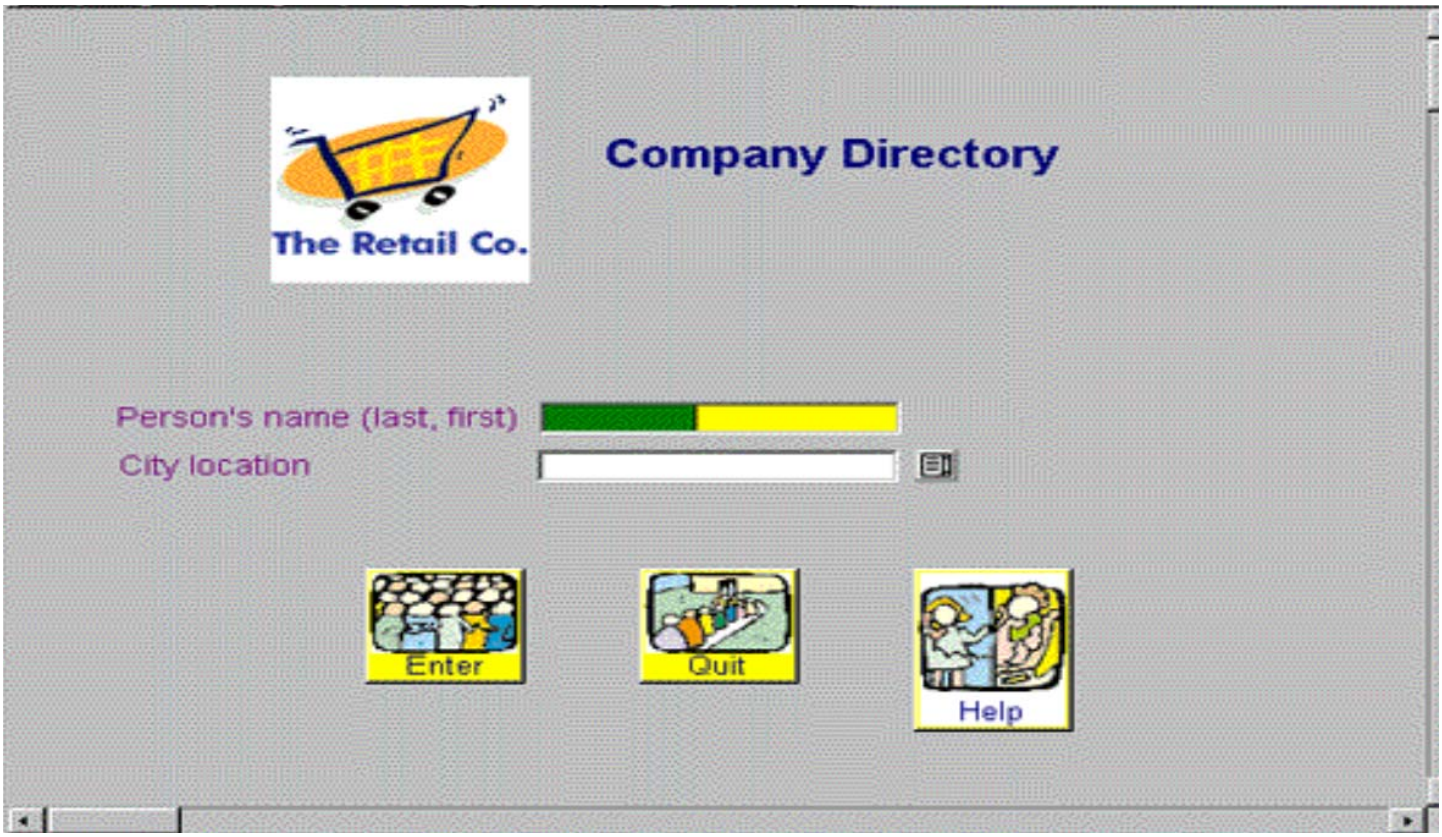


Abb. 9.2.10
Screen Scraping Ergebnis

Zu beachten ist, dass auch Aktionen mit der Maus möglich sind, die über die EPI Schnittstelle in 3270 Aktionen umgesetzt werde können, oder aber lokal (z.B. eine Help Funktion) abgearbeitet werden.

Nachteilig ist beim Screen Scraping Ansatz, dass ein Teil der Präsentationslogik auf dem Klientenrechner ausgeführt wird. Dies verursacht zusätzliche Administrationskosten.

Unintelligente 3270 Terminals haben den Vorteil, dass nicht sehr viel schief gehen kann. Das Terminal funktioniert entweder, oder er funktioniert nicht. Im Problemfall kommt der Techniker mit einem Ersatzterminal, prüft mit einem einfachen Testgerät ob der Kabelanschluss ok ist (ja/nein), wenn ja tauscht den Terminal aus und nimmt den alten Terminal mit in die Werkstatt. Für diese Tätigkeit ist keine Spezialausbildung erforderlich. PCs mit zusätzlicher Software als Terminal Ersatz erfordern schnell im Problemfall einen Spezialisten.

Zur Abhilfe setzt man heute als Terminals gerne PCs mit nur rudimentären Funktionen ein, z. B. ohne Plattenspeicher und ohne USB Anschluss, auf denen nur ein Browser läuft. Derartige PCs werden als „Thin Clients“ bezeichnet. Die 3270 Emulations- und Screen Scraping Software wird auf einen Server verlagert, der eine ganze Reihe von derartig abgerüsteten Arbeitsplatzrechnern bedient. Auf dem Server läuft Software, welche 3270 Daten in eine Browser Darstellung umsetzt.

Es gibt zahlreiche Software Produkte, die eine derartige Umsetzung durchführen. Ein Beispiel ist „Host Access Transformation Services“ (HATS) von der IBM. Wir werden später in Kapitel 18, „Java Connection Architecture“, ein Beispiel zeigen.

Die Firma IGEL Technology GmbH in 28199 Bremen ist ein führender europäischer Hersteller von Thin Client PC Hardware, siehe https://www.igel.com/de/. Besonders in der öffentlichen Verwaltung werden Thin Client PCs gerne eingesetzt.

9.2.7 Probleme des Screen Scraping Ansatzes

Viele Hersteller haben das 3270 Protokoll für ihre Client/Server Produkte eingesetzt, aber es existieren Probleme des Screen Scraping Ansatzes

Das 3270 Protokoll lässt einige Funktionen nicht zu, z.B.:

- **Scroll Bar**
- **Mehrere Fenster**

Als Lösung bieten sich zwei Ansätze an:

- **Erweiterung des 3270 Protokolls um die fehlenden Funktionen. Dies wurde z.B. von der Firma SAP für ihre System R/3 SAPGUI implementiert.**

- **Java und http für die Präsentationslogik. Wir werden dies später in dem Abschnitt 18.3 diskutieren.**

9.2.8 Anhang

Zur Vertiefung zeigen wir ein einfaches CICS Programm, welches zwei Maps in seinem Mapset verwendet.

Die erste Map (Eingabe Map) stellt einen Eingabe Screen dar, in den der Benutzer zwei Zahlen in zwei dafür vorgesehene Felder eingeben kann.

Die zweite Map stellt einen Ergebnis Screen dar. Hier wird das Ergebnis der Addition der beiden Zahlen wiedergegeben.

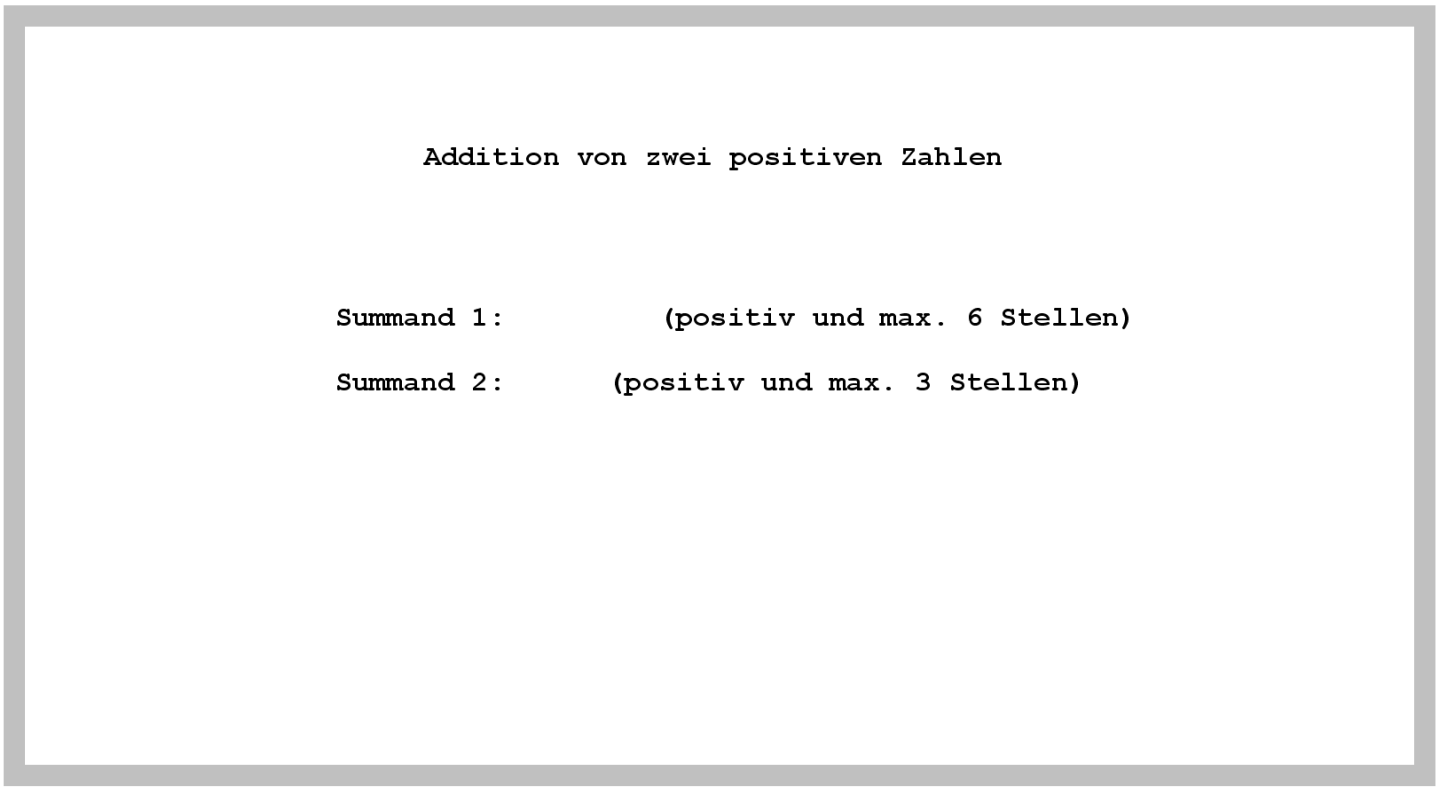


Abb. 9.2.11
Dies ist die Eingabe Map

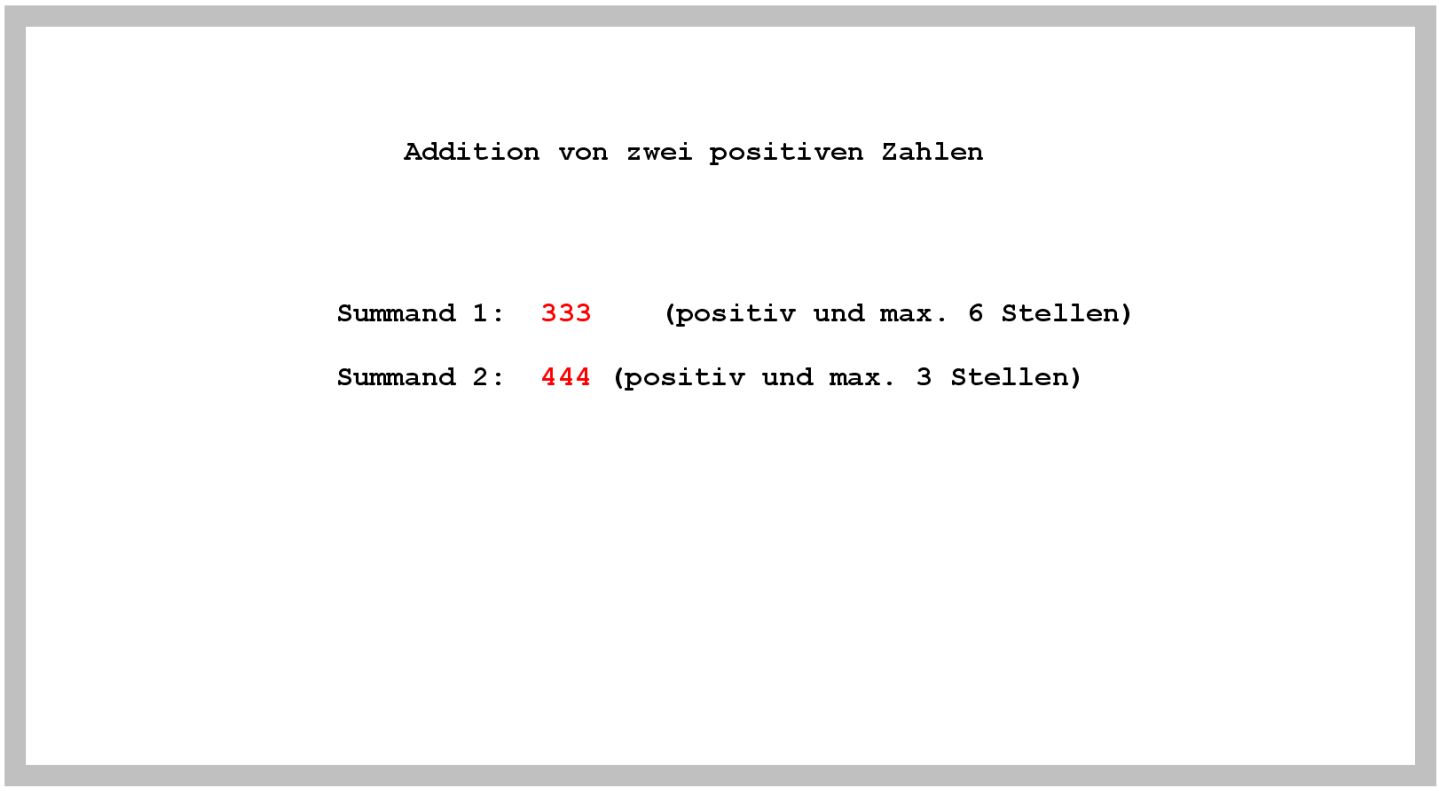


Abb. 9.2.12
Input in die Eingabe Map

Dies ist die Eingabe Map, nachdem der Benutzer 2 Zahlen in die dafür vorgesehen (nicht markierten Felder) eingegeben hat, ...

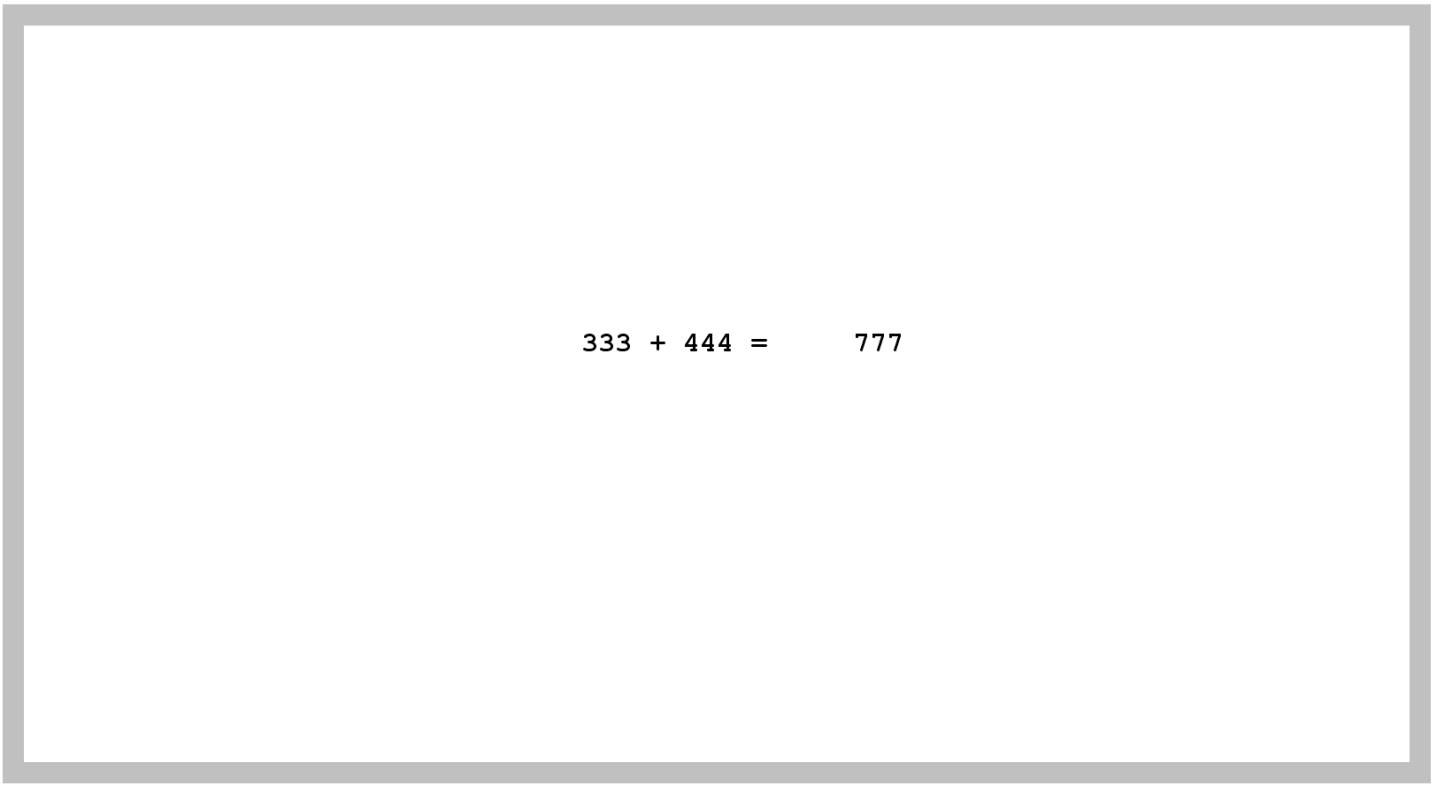


Abb. 9.2.13
Ausgabe Map

… und dies ist die CICS Ausgabe. Die nächsten beiden Abbildungen zeigen das dafür benutzte BMS Programm, welches aus zwei ISPF Panels besteht.

```
  File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       PRAKT20.CICS.BMS(MAPSET) - 01.25               Columns 00001 00072
****** ***************************** Top of Data ******************************
==MSG> -CAUTION- Profile changed to CAPS OFF (from CAPS ON) because data
==MSG>           contains lower case characters.
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000100 //PRAKT20M JOB (),CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID
000101 //ASSEM    EXEC DFHMAPS,MAPNAME='M3BM020',RMODE=24
000102 //SYSUT1   DD *
000110 M3BM020 DFHMSD TYPE=MAP,MODE=INOUT,LANG=COBOL2,STORAGE=AUTO,TIOAPFX=YES
000120 *       Die Eingabemap des Mapsets.
000200 EINGMAP DFHMDI SIZE=(24,80),LINE=1,COLUMN=1,CTRL=FREEKB
000400         DFHMDF POS=(5,22),LENGTH=34,ATTRB=(ASKIP,NORM),                X
000500                INITIAL='Addition von zwei positiven Zahlen'
000700         DFHMDF POS=(10,18),LENGTH=10,ATTRB=(ASKIP,NORM),               X
000800                INITIAL='Summand 1:'
000900 A       DFHMDF POS=(10,30),LENGTH=6,ATTRB=(UNPROT,NUM,IC)
000910         DFHMDF POS=(10,37),LENGTH=28,ATTRB=(ASKIP,NORM),               Z
000920                INITIAL='(positiv und max. 6 Stellen)'
Command ===>                                                 Scroll ===> PAGE
  F1=Help       F3=Exit       F5=Rfind      F6=Rchange   F12=Cancel
```

Mapset

Map # 1

Abb. 9.2.14
Die erste Hälfte des BMS Programms

```
  File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       PRAKT20.CICS.BMS(MAPSET) - 01.25               Columns 00001 00072
001100         DFHMDF POS=(12,18),LENGTH=10,ATTRB=(ASKIP,NORM),               E
001110                INITIAL='Summand 2:'
001200 B       DFHMDF POS=(12,30),LENGTH=3,ATTRB=(UNPROT,NUM)
001210         DFHMDF POS=(12,34),LENGTH=28,ATTRB=(ASKIP,NORM),               W
001211                INITIAL='(positiv und max. 3 Stellen)'
001212 *       Die Ausgabemap des Mapsets.
001320 AUSGMAP DFHMDI SIZE=(24,80),CTRL=(PRINT,FREEKB)
001330 SUMMND1 DFHMDF POS=(11,29),ATTRB=(ASKIP,NORM),LENGTH=6
001340         DFHMDF POS=(11,36),ATTRB=(ASKIP,NORM),LENGTH=1,INITIAL='+'
001341 SUMMND2 DFHMDF POS=(11,38),ATTRB=(ASKIP,NORM),LENGTH=3
001342         DFHMDF POS=(11,42),ATTRB=(ASKIP,NORM),LENGTH=1,INITIAL='='
001350 SUMME   DFHMDF POS=(11,44),ATTRB=(ASKIP,BRT),LENGTH=7
001410         DFHMSD TYPE=FINAL
001500         END
001600 /*
001700 //
****** **************************** Bottom of Data ****************************

Command ===> SUB                                             Scroll ===> PAGE
 F1=Help       F3=Exit       F5=Rfind      F6=Rchange   F12=Cancel
```

Map # 2

Abb. 9.2.15
Die zweite Hälfte des BMS Programms, in der die Map für die Ergebnisausgabe definiert wird.

9.3 Multiregion and Intersystem Communication

9.3.1 CICS Interprocess Communication

Als Pfadlänge bezeichnet man die Zahl der während der Abarbeitung einer Transaktion ausgeführten Maschinenbefehle. Eine typische CICS oder IMS Transaktionen hat eine Pfadlänge von mehr als 100 000 Maschinenbefehlen.

Transaktionen mit einer Pfadlänge von 1 Million Maschinenbefehlen sind nicht ungewöhnlich.

Annahme: Eine CPU führt 1 Milliarde Maschinenbefehle / Sekunde aus.

Bei einer Pfadlänge von 250 000 Maschinenbefehlen sind 4 000 Transaktionen / Sekunde theoretisch möglich, wenn man Overhead vernachlässigt.

Große Installationen bewältigen 5000 Transaktionen / Sekunde. Spitzenwert von 9 000 Transaktionen / Sekunde sind schon aufgetreten. Das IBM Entwicklungslabor in Hursley, Südengland hat 19 000 Transaktionen / Sekunde demonstriert.

Derartig hohe Transaktionsraten bedingen mehrere CPUs in einer „Symmetrischen Multiprozessor“ (SMP) Konfiguration und häufig auch mehrere Mainframe Rechner..

Hardware	klein/einfach	mittel	groß/komplex
Endbenutzer	100	10 000	> 100 000
CPUs	1 - 3	5 -20	> 100
Plattenspeicher TeraByte	1 - 10	10 – 100	100 – 5 000
Magnetband Terabyte	100	100 – 1000	> 10 000
Software			
Transaktions-programme X)	400	4 000	40 000
Quell-programme XX)	1 000	10 000	100 000

Abb. 9.3.1
Anzahl der Komponenten eines Transaktionsverarbeitungssystems

X) einschl. Berichte, Maps XX) einschl. alte Versionen

Pfadlänge pro Transaktion: 100 000 - 1 000 000 Maschinenbefehle

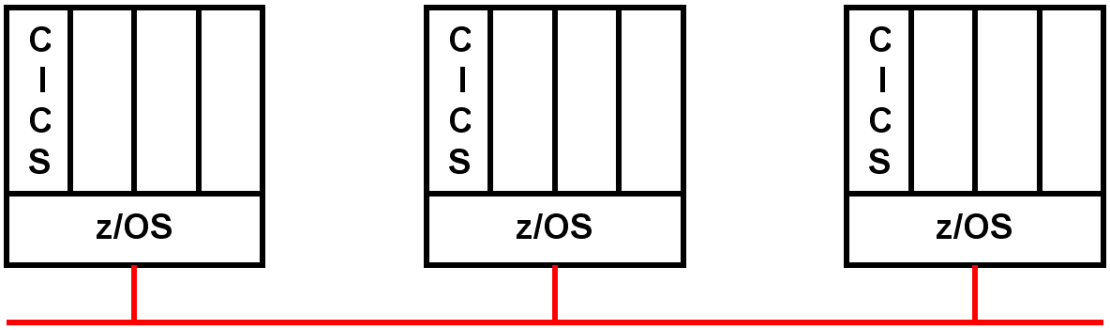


Abb. 9.3.2
Verbund mehrerer CICS Instanzen

Häufig ist gewünscht, dass 2 CICS Transaktionen, die auf unterschiedlichen CICS Transaktions-Monitoren laufen, miteinander kommunizieren können. Es existieren 2 Arten von CICS Interprocess Communication:

- **Multiregion operation (MRO)**
- **Intersystem communication (ISC)**

Multiregion Operation (MRO) wird für die Communication von zwei CICS Regions benutzt, die sich auf dem gleichen Mainframe System befinden.

Ein Spezialfall ist ein Verbund aus mehreren Mainframe Rechnern, wobei auf jedem Rechner eine CICS Region installiert ist.

Für einen Cluster bestehend aus mehren Mainframes innerhalb eines Rechenzentrums existiert eine als „Parallel Sysplex“ bezeichnete Integrationssoftware, die ebenfalls MRO zwischen unterschiedlichen z/OS Instanzen ermöglicht. Sind die z/OS Rechner in geographisch voneinander getrennten Rechenzentren untergebracht, spricht man von einem Geographically Dispersed Parallel Sysplex (GDPS, siehe Abschnitt 14.1.1), Das Thema Sysplex wird in Kapitel 12 behandelt.

Intersystem Communication (ISC) wird für die Communication von zwei CICS Systemen benutzt, die lediglich über TCP/IP und das Internet miteinander verbunden sind. Hierbei können die CICS Instanzen alle auf z/OS, oder aber auch auf Unix oder Windows Rechnern installiert sein.

9.3.2 CICS Multiregion Operation

CICS spezifische MRO-Standards sind:

Transaction Routing

Transaktionen (Eingabe vom Klienten) können zwecks Ausführung von einem CICS System einem anderen CICS System unverändert übergeben werden

Function Shipping

Eine Anwendung in einem CICS System kann auf Daten eines anderen CICS Systems zugreifen

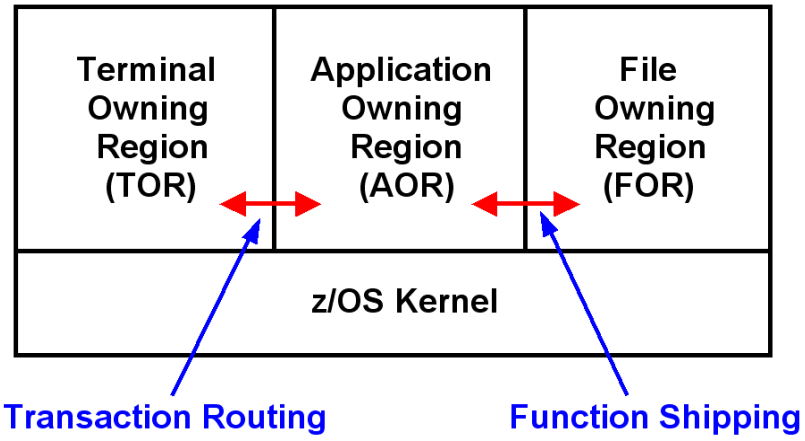


Abb. 9.3.3
Benutzung von Tranaction Routing und Function Shipping

CICS Multiregion Operation (MRO) ermöglicht es zwei oder mehr CICS Systemen, die auf dem gleichen z/OS System oder innerhalb des gleichen Parallel Sysplex laufen, miteinander zu kommunizieren.

Relativ häufig wird CICS auf 3 Regionen (virtuelle Adressenräume) innerhalb des gleichen Rechners aufgeteilt, die jede über einen eigenen CICS Transaktionsmonitor verfügen, und über MRO miteinander kommunizieren.

Die Terminal Owning Region verwaltet alle angeschlossenen Terminals. Sie gibt über das „Transaction Routing“ Protokoll die Transaktion an die Application Owning Region weiter.

Die Application Owning Region führt die Anwendung aus. Wenn diese Daten (z.B. von einem VSAM Data Set) benötigt werden, werden diese von der File Owning Region über das Function Shipping Protokoll zur Verfügung gestellt.

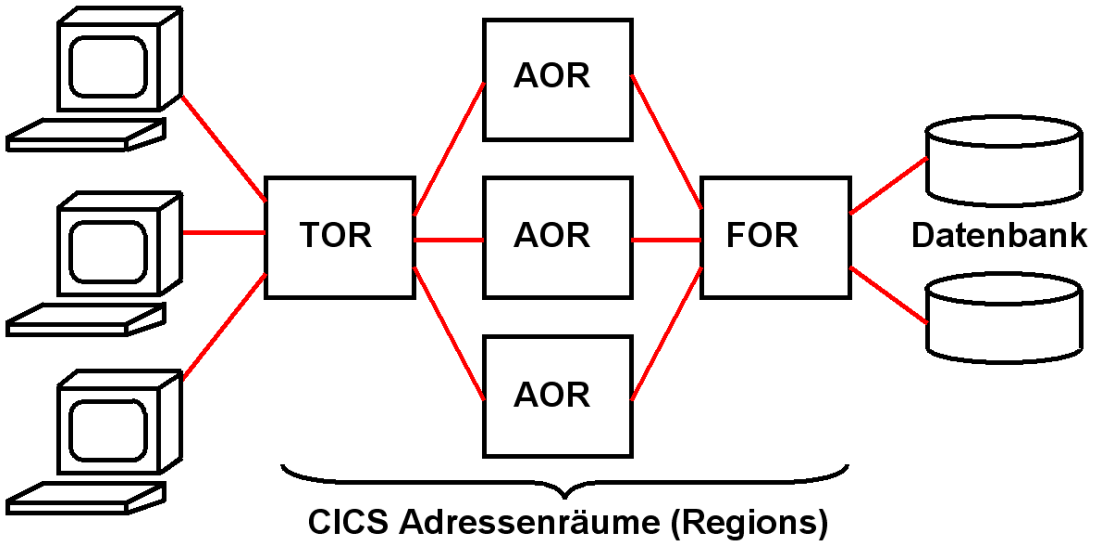


Abb. 9.3.4
Verteilung auf mehrere Adressenräume

Gezeigt ist ein Beispiel mit 5 CICS Instanzen (Ausprägungen), die in 5 virtuellen Adressenräumen (Regions) laufen. In einem symmetrischen Multiprozessor können die 5 Instanzen auf 5 CPUs verteilt werden.

TOR	**Terminal Owning Region**
AOR	**Application owning Region**
FOR	**File owning Region (Resource Region)**

Die AORs können für bestimmte Anwendungen, spezielle Zugriffsarten, Datenbankverbindungen usw. optimiert sein. AORs können auf unterschiedliche Regions eines Sysplex verteilt sein.

Es existieren Mainframe Installationen mit über 100 CICS Instanzen in ebenso vielen CICS Regions.

Warum mehrfache AORs ?

Prozesse unter z/OS sind durch einen Task Control Block (TCB) gekennzeichnet. Der TCB ist ein Speicherbereich, der z.B. den Inhalt der Mehrzweck-, Gleitkomma-, Controlregister und das Programm Status Wort (PSW) aufnimmt, wenn der Prozess von dem laufenden in den wartenden Zustand versetzt wird.

CICS einschließlich Nucleus und aller Anwendungsprogramme läuft in einem einzigen z/OS Address Space und wird in den Augen des z/OS Kernels durch einen einzigen TCB (Task Control Block) repräsentiert.

Innerhalb des CICS Address Spaces laufen gleichzeitig Hunderte oder tausende von Transaktionen, die durch die Scheduler Komponente des CICS Nucleus gescheduled werden. Jede Transaktion wird durch einen „quasi-reentrant“ TCB (QR TCB) repräsentiert. Im Gegensatz zu normalen TCBs werden die QR TCBs nicht vom Betriebssystem Kernel, sondern vom CICS Nucleus gescheduled und verwaltet. Dies geschieht unsichtbar für den z/OS Kernel.

Da CICS nur über einen einzigen TCB verfügt, kann der z/OS Kernel es auch nur für eine einzige CPU schedulen. Nur ein einziger QR TCB ist in jedem Augenblick active (running). Multiple AORs bedeuten multiple CICS Instanzen und erlauben, die Transaktionsverarbeitung auf mehreren CPUs zu verteilen. Der z/OS Work Load Manager (WLM, diskutiert in Kapitel 14) ordnet neue Transaktionen den einzelnen AORs zu.

9.3.3 Quasireentrant Programme

Multithreading ermöglicht es, dass eine einzelne Kopie eines Anwendungsprogramms von mehreren Transaktionen gleichzeitig ausgeführt wird. Zum Beispiel kann eine Transaktion anfangen, ein Anwendungsprogramm auszuführen. Wenn ein EXEC CICS Befehl erreicht wird, wird die Transaktion in den Wartezustand versetzt. Der CICS Dispatcher scheduled eine andere Transaktion, welche die gleiche Kopie des Anwendungsprogramms ausführt.

Multithreading verlangt, dass alle CICS Anwendungsprogramme quasi-reentrant geschrieben sind. „Quasi-Reentrant" bedeutet, dass das Anwendungsprogramm vor und nach jedem EXEC CICS Befehl in einem konsistenten Zustand sein muss. Das Anwendungsprogramm muss seriell wiederverwendbar zwischen Einstiegs- und Ausstiegspunkten sein. Quasi Reentrance garantiert, dass jeder Aufruf eines Anwendungsprogramms unbeeinflusst von früheren Durchläufen, oder dem Multithreading durch mehrere CICS Tasks ist. CICS Anwendungsprogramme mit der EXEC CICS-Schnittstelle gehorchen dieser Regel automatisch. Für COBOL, C/C++ und PL/1 Programmen wird Reentrance erreicht, indem jede Transaktion eine eigenen Speicherbereich für die Speicherung von Variablen erhält.

Eine gewisse Ähnlichkeit mit Java Objekten ist unverkennbar.

CICS Anwenderprogramme (Transaktionen) laufen unter einer CICS managed Task Control Block (TCB). Wenn die Programme als Quasi-Reentrant definiert sind (das CONCURRENCY Attribut der Programm Ressourcen-Definition gesetzt ist), läuft CICS immer unter dem CICS „Quasi-Reentrant (QR) TCB“. Die Voraussetzungen für ein Quasi-Reentrant Programm in einem Multithreading Kontext sind weniger streng, als wenn das Programm gleichzeitig auf mehreren CPUs ausgeführt werden soll.

CICS Transaktionen können alternativ unter dem “Open TCB” an Stelle des QR TCBs laufen. Open TCBs werden vom Betriebssystem Kernel (und nicht vom CICS Nucleus) gescheduled und können damit mehreren CPUs zugeordnet werden. In anderen Worten, die Multiprocessor Eigenschaften eines Mainframes werden genutzt. Hiermit kann der Gesamtdurchsatz verbessert werden; allerdings erfordert das Scheduling durch den Betriebssystem Kernel mehr Aufwand (CPU Zyklen) als das Scheduling durch den CICS Nucleus. Vor allem Java Programme machen vom Open TCB Gebrauch.

Damit Open TCB Programme die Integrität der gemeinsam genutzten Ressourcen wahren , müssen Serialisierungs-Techniken verwendet werden, welche den gleichzeitigen Zugriff auf gemeinsam genutzte Ressourcen verhindern. Programme, welche die Serialisierung beim Zugriff auf gemeinsam genutzte Ressourcen sicherstellen, werden als „threadsafe“ oder "reentrant“ bezeichnet.

9.3.4 AORs und VSAM

AORs laufen in getrennten z/OS Regions. Die dort laufenden Programme können unabhängig voneinander auf die gleichen DB2 Daten zugreifen. Das DB2 Datenbanksystem verfügt über entsprechende Lock Management Einrichtungen.

Die weltweiten, von CICS verarbeiteten VSAM Datenbestände, haben eine ähnliche Größe wie die von CICS verarbeiteten DB2 Datenbestände. Häufig greift eine CICS Anwendung parallel sowohl auf VSAM wie auf DB2 Daten zu. VSAM fehlen jedoch die Lock Management Einrichtungen, die für einen parallelen Zugriff durch mehrere AORs erforderlich sind.

Abhilfe schafft eine als „VSAM Record Level Sharing“ (RLS) bezeichnete Funktion, die von DFSMS (siehe Abschnitt 3.3 „z/OS Betriebssystem“) als getrenntes Subsystem (SMSVSAM) zur Verfügung gestellt wird. CICS RLS ermöglicht es, dass zahlreiche CICS Anwendungen in unterschiedlichen AORs parallel auf die gleichen VSAM Data Sets mit voller Update Fähigkeit zugreifen können.

9.3.5 CICSPlex

Wenn mehrere CICS Instanzen in getrennten Regions installiert werden, nennt man die resultierende Architektur einen CICSPlex. Als CICSPlex wird eine Gruppe von logisch verbundene CICS Regions betrachtet. Ein typischer CICSPlex ist eine miteinander verbundene Gruppe von TORs, AORs, FORs usw. Auf einem Sysplex-Verbund von Mainframe Rechnern wird normalerweise ein (oder mehrere) CICSPlex eingesetzt.

Jede CICS Instanz verwaltet eine Region (Address Space). Jede CICS Region kann für eine bestimmte Aufgabe optimiert sein, und dennoch leicht mit anderen CICS Regions kommunizieren.

Ein System von miteinander verbundenen CICS Regions kann z.B. enthalten:

- **Test Regions für das Austesten neuer Anwendungen**
- **Function-spezifische Regions für die Steuerung von Terminals, Benutzer Interfaces oder System Services**
- **Nutzung spezifischer Regions für das Management bestimmter Anwendungen, wie z.B. Datenbanken**

Eine spezielle CICS Komponente, der CICSPlex System Manager (CICSPlex SM)ist für die Steuerung einer Vielzahl von CICS Regions zuständig. In zunehmendem Maß finden wir Unternehmen mit Dutzenden oder Hunderten von CICS Regions.

Ein Beispiel ist die Installation einer Schweizer Großbank. Die Union Bank of Switzerland unterhält in zwei Standorten in der Nähe von Zürich zwei Rechenzentren, die als Geographically Dispersed Parallel Sysplex (GDPS) zusammengeschaltet sind. Ein weiterer GDPS befindet sich im Staate New Jersey in den USA.

Die CICS Konfiguration des Schweizer GDPS besteht aus (3Q 2006):

- **11 CICSPlexes**
- **943 CICS Regions**

Die CICS Konfiguration des USA GDPS besteht aus:

- **4 CICSPlexes**
- **194 CICS Regions**

9.3.6 Intersystem Communication (ISC)

CICS Multi Region Operation findet innerhalb eines einzigen z/OS Systems oder innerhalb eines Sysplex statt, der darauf abzielt, ein „Single System Image“ anzubieten. Eine CICS Kommunication zwischen zwei (ansonsten unkoordinierten) Rechnern wird als CICS Intersystem Communication (CICS ISC) bezeichnet.

Es existieren viele, nicht CICS spezifische Arten, von Intersystem Communication (ISC).

Einfache Beispiele für ISC-Standards sind Sockets, Named Pipes und APPC. In verteilten Systemen ist ISC häufig schwierig zu entwickeln, debug, portieren, besonders für unterschiedliche Plattformen, z.B. auf Grund von Abhängigkeiten von der Rechnerhardware (Byte Order, Adressierung).

Der RPC (Remote Procedure Call) ist eine klassische Software für die Implementierung von Client/Server Konfigurationen. Ein RPC liefert automatisch Funktionen, die beim Programmieren mit Sockets vom Programmierer erstellt werden müssen, z.B. Fehlerbehandlung, Adressierung und Datendarstellung. RPC erleichtert gegenüber Sockets die Entwicklung und Wartung von verteilten Anwendungen.

Es existieren eine ganze Reihe von inkompatiblen RPC Standards, z.B. Sun RPC, DCE RPC, SOAP (Web Services), sowie objekt-orientierte Versionen wie CORBA und RMI. Der RPC ist ein synchrones Protokoll, d.h. der Klient wartet (blockiert) auf die Antwort des Servers. Im Gegensatz dazu ist MQSeries ein weit verbreitetes asynchrones RPC Protokoll.

Normalerweise wird ISC durch Bibliotheken eines Betriebssystems implementiert.

CICS Distributed Program Link (DPL) ist eine CICS-spezifische, besonders einfach zu benutzende, Art des Remote Procedure Call. Voraussetzung ist, dass sowohl der Client als auch der Server unter CICS laufen. DPL benutzt TCP/IP, SNA oder NetBios in der Netzwerk Schicht 4.

Ein Programm eines CICS-Systems kann mit Hilfe des DPL Kommandos EXEC CICS LINK eine Verbindung mit einem Programm eines anderen CICS-Systems aufnehmen.

Es existieren viele Vereinfachungen gegenüber dem normalen RPC, da einheitliche Datendarstellung, Namenskonventionen, Fehlerbehandlung, u.s.w. Z.B. existieren keine Datenrepräsentations-Probleme, da alle CICS Implementierungen einen einheitlichen EBCDIC Standard einsetzen.

CICS TP Monitore existieren für viele unterschiedliche Betriebssystem-Plattformen, neben z/OS z.B. für Windows, i5/OS, oder Unix. Eine DPL Kommunikation zwischen CICS TP Monitoren auf unterschiedlichen Platformen ist sehr einfach zu implementieren.

Neben DPL existieren weitere CICS ISC Protokolle mit spezifischen Eigenschaften, z.B. Distributed Transaction Programming (DTP).

9.3.7 Distributed CICS

Der CICS Transaction Server wurde ursprünglich für Mainframe Rechner entwickelt. Heute existiert zusätzlich eine relativ weit verbreitete Version, die als „Distributed CICS“ bezeichnet wird und die auf anderen Plattformen läuft, z.B.:

- **Solaris,**
- **Microsoft Windows,**
- **IBM AIX,**
- **Hewlett-Packard UNIX (HP-UX) und**
- **Linux**

Distributed CICS Systeme sind über das Imternet miteinander verbunden

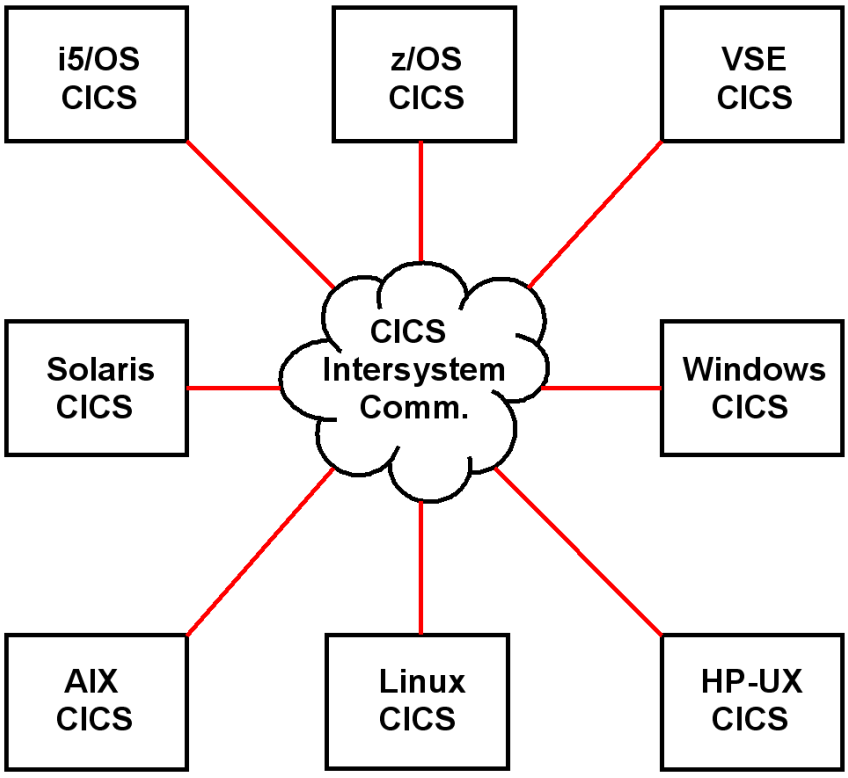


Abb. 9.3.5
Verbund von Mainframe und Distributed CICS Instanzen

Distributed CICS verfügt über die gleiche Application Programming Interface (API) und die gleiche System Programming Interface (SPI) wie der Mainframe CICS Transaction Server.

Es unterstützt eine Vielzahl von Datenbanken wie DB2, Oracle, IBM Informix, Microsoft SQL Server und Sybase. Datenintegrität wird durch die Unterstützung der XA-Schnittstelle bereitgestellt. XA ist die Industrie-Standard-Schnittstelle für das Commitment und die Recovery von transaktionalen Daten, einschließlich des Two-Phase Commit-Prozesses. Weiterhin verfügt Distributed CICS über einen „Structured File Server“ (SFS), eine Record-orientierte VSAM Emulation, welche indexierte, relative und sequentielle File Zugriffe ermöglicht. Unterstützt wird das Äquivalent eines Open TCB, nicht aber eines QR TCB.

Anwendungen können in C, C++, PL/I, COBOL, und Java geschrieben werden und sind problemlos auf die Mainframe CICS Version portierbar.

Spezifisch ist hiermit eine sehr einfache und problemlose Intersystem Communication möglich.

http://www.redbooks.ibm.com/redbooks/pdfs/sg247185.pdf

Als Beispiel können CICS Programme auf unterschiedlichen Plattformen (z.B. z/OS, Linux, Windows) Daten über die CICS COMMUNICATION AREA (COMMAREA) austauschen

In COBOL ist dies ein Teil der DATA DIVISION/LINKAGE SECTION, z.B.

```
01 DFHCOMMAREA.
    05 PROCESS-SW           PIC X.
       88 INITIAL-ENTRY     VALUE ‘0’.
       88 VERIFICATION      VALUE ‘1’.
    05 ACCOUNT-NUMBER       PIC X(10).
            •
            •
            •
EXEC LINK PROGRAM(ACCTPGM)
      COMMAREA(DFHCOMMAREA)
      LENGTH(11)
END-EXEC.
```

Abb. 9.3.6
COMMAREA als Cobol Copybook

9.4 Weiterführende Information

Die Benutzung von mobilen Endgeräten für einen Zugriff auf CICS Anwendungen hat in den letzten Jahren zunehmend an Bedeutung gewonnen. Zahlreiche Hersteller von Software Produkten sind mit eigenen Entwicklungen in diesen Markt eingestiegen.

Ein Beispiel ist die Firma Hostbridge, welche Software Produkte für die CICS Communication herstellt. Ein Beispiel für einen Zugriff auf den z/OS CICS Transaction Server mittels eines iPads oder anderer Android-Endgeräte ist zu finden unter

www.cedix.de/VorlesMirror/Band1/HostBridgeWIRE.pdf

Das folgende Video demonstriert, wie man mit einem normalen iPhone auf eine CICS VSAM Anwendung zugreift.

http://testiphone.com/?address=on&url=http://zserveros.demos.ibm.com:9080/iPhone/egl.html

und dieses Video demonstriert, was man mit dem CICS iPhone alles machen kann.

http://www.youtube.com/watch?v=5JyJ0XXR_3c

Das nächste Video enthält eine detaillierte Vorführung, wie mit Hilfe des Rational Developers for System z (RDz) eine iPhone Anwendung entwickelt wird, welche auf eine existierende z/OS COBOL Transaktion zugreift.

http://www.youtube.com/watch?v=5Fu66xMQdcY

Ein Tutorial über CICS for Intersystem Communication finden sie hier

http://docstore.mik.ua/univercd/cc/td/doc/product/ibd/ctrc/ctrchost.htm

10. WebSphere MQ

10.1 Übersicht

10.1.1 Client/Server-Operation

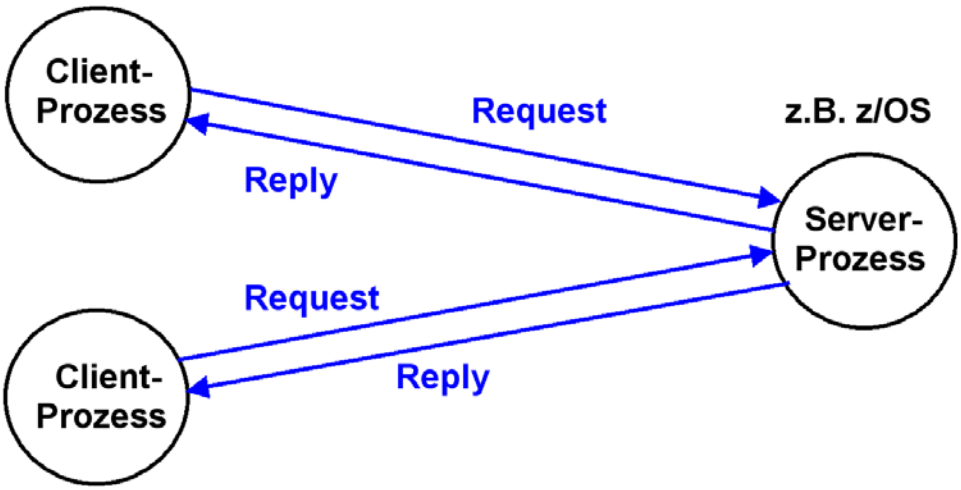


Abb. 10.1.1
Remote Procedure Call

In einer Client/Server Konfiguration bietet ein Server seine Dienste (Service) einer Menge a priori unbekannter Klienten (Clients) an. Client-Prozesse auf einem Client-System (z.B. einem PC) rufen Dienste (Services) auf einem Server-System auf. Ein Beispiel ist der Zugriff auf eine URL auf einem Apache-Webserver. Als Dienstleistung gibt Apache das gewünschte Dokument zurück. Ein Server bietet Dienste für jeden Client an, der seine Dienste anfordert.

Wir verwenden die folgenden Begriffe:

- **Client:** **ruft einen Dienst (Service) auf einem Server auf**
- **Server:** **Rechner, der Dienst-Software (als Service bezeichnet) ausführt**
- **Service:** **Software Prozess, der auf einem Server ausgeführt wird (möglicherweise auf mehreren Servern)**
- **Interaktion:** **Anfrage/Reaktion (Request/Reply)y**

Ein physischer Server kann gleichzeitig viele verschiedene Serverdienste (Services) anbieten. Beispielsweise bietet ein CICS Server multiple CICS Servies (identifiziert durch unterschiedliche TRIDs) einer Vielzahl von CICS Klienten an.

Der größere Teil aller Anwendungen in Wirtschaft und Verwaltung läuft auf Client/Server Systemen.

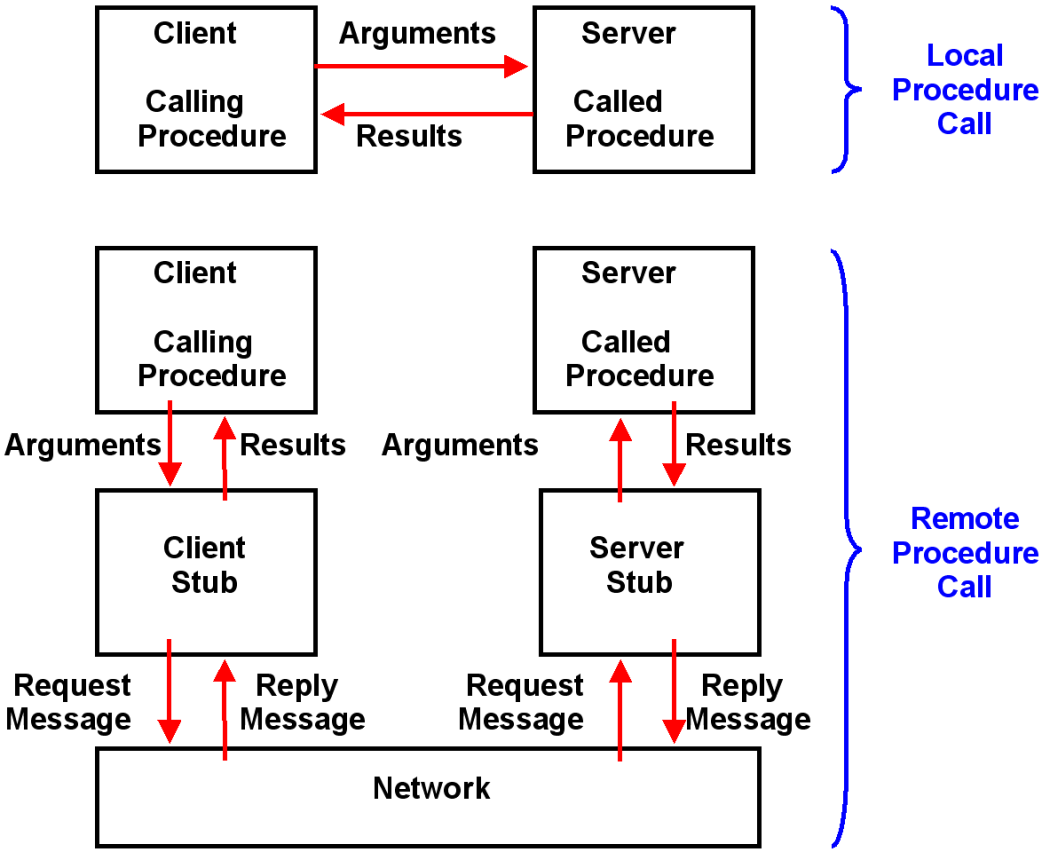


Abb. 10.1.2
Laufzeitumgebung des Remote Procedure Calls

Local Procedure Call

Eine aufgerufene Prozedur (Server) wird innerhalb des gleichen Adressraums wie die aufrufende Prozedur (Client) ausgeführt.

Remote Procedure Call

Client und Server laufen als zwei getrennte Prozesse, in der Regel auf zwei verschiedenen Rechnern. (Eine Konfiguration mit zwei getrennten Prozessen, die jeweils in ihrem eigenen virtuellen Adressraum auf dem gleichen Rechner laufen, ist ebenfalls möglich).

Beide Prozesse kommunizieren über Stubs. Stubs bilden lokale Prozeduraufrufe auf Netzwerk RPC Funktionsaufrufe ab.

Der klassische RPC benutzt die Bezeichnung Server Stub. Java und CORBA verwenden den Begriff Skeleton anstelle von Server-Stub. Die Bezeichnungen Server Stub und Skeleton sind austauschbar.

Stubs bzw. Skeletons implementieren eine RPC Runtime. Sie übersetzen Aufrufe der Client API in Sockets und dann in Nachrichten, die über das Netz übertragen werden.

Der Remote Procedure Call (RPC) erweckt den Anschein, als ob ein Client eine Prozedur aufruft, die sich in dem gleichen Adressraum befindet. In Wirklichkeit ruft die Client-Anwendung eine lokale Stub Prozedur auf, (anstelle des eigentlichen Codes, der die aufgerufene Prozedur implementiert). Der Client und Server haben jeweils einen eigenen Adressraum, und können sich (Normalfall) auf unterschiedlichen Maschinen befinden.

10.1.2 Funktionsweise des RPC

Stubs werden kompiliert und mit der Client-Anwendung verbunden. Statt dem eigentlichen Code, der die Remote-Verfahren implementiert, bewirkt der Client-Stub-Code:

- **Ermitteln der erforderlichen Parameter von dem Client-Adressraum.**
- **Übersetzen der Parameter in ein Standard-Format (Network Data Representation, NDR) für die Übertragung über das Netzwerk.**
- **Anruf von Funktionen in der RPC-Client-Laufzeitbibliothek, um die Anforderung und seine Parameter an den Server zu senden.**

Der Server führt die folgenden Schritte aus, um die Remote Procedure aufzurufen:

- **Die Server RPC-Laufzeitbibliothek Funktionen akzeptieren die Anfrage und rufen die Server-Stub Prozedur auf.**
- **Der Server Stub empfängt die Parameter von dem Netzwerk-Puffer und konvertiert sie von dem Netzwerk Übertragungsformat (NDR) in das Format, welches der Server benötigt.**
- **Der Server-Stub ruft die eigentliche Prozedur auf dem Server auf.**

Die Remote-Prozedur läuft jetzt. Sie erzeugt möglicherweise Ausgabeparameter und einen Rückgabewert. Wenn die Remote-Prozedur abgeschlossen ist, werden die resultierenden Daten in einer ähnlichen Folge von Schritten an den Client zurückgegeben:

- **Die Remote-Prozedur übergibt Ihre Ausgabeparameter und Rückgabewert Daten an den Server-Stub.**
- **Der Server-Stub wandelt die Daten auf das Format um, das für die Übertragung über das Netzwerk benötigt wird, und übergibt sie an die RPC-Laufzeitbibliothek.**
- **Die Server RPC-Laufzeitbibliothek überträgt die Daten über das Netzwerk an den Client Rechner-Computer.**

Der Client akzeptiert die Daten und übergibt sie an die aufrufende Funktion:

- **Die Client RPC-Laufzeitbibliothek erhält die Remote-Prozedur Rückgabewerte und gibt sie an den Client-Stub. weiter.**
- **Der Client-Stub wandelt die Daten aus dem NDR-Format in das Format um, das der Client-Rechner erwartet. Der Stub schreibt Daten in den Client-Speicher und liefert das Ergebnis an das aufrufende Programm auf dem Client.**
- **Die aufrufende Prozedur fährt fort, als ob sich die aufgerufene Prozedur im gleichen Adressraum befunden hätte.**

10.1.3 RPC Implementierungen

Sockets für TCP/IP , sowie LU 6.2 und APPC für SNA, sind Protokolle auf der niedrigsten Verbindungsstufe (Schicht 5 des OSI Modells, siehe Abschnitt 18.1).

Darüberliegende Protokolle (Schicht 6 und 7) werden allgemein als Remote Procedure Call (RPC) bezeichnet. Es existieren eine Vielzahl von RPCs, deren APIs (Application Programming Interface) in den meisten Fällen inkompatibel sind:

Klassische RPCs

- **SUN RPC (Standard in Solaris und vielen Linux Versionen),**
- **DCE RPC (unterstützt von Microsoft und IBM, besonders auch von z/OS),**
- **CPI-C (von IBM, ursprünglich für SNA entwickelt).**

Klassische RPCs sind immer noch populär aufgrund der überlegenen Leistung. Linux, Unix und z/OS-Systeme unterstützen sowohl den Sun RPC als auch den DCE RPC.

Spezialisierte RPC

- **DPL**

CICS Distributed Program Link, sehr performant, leistungsfähig und einfach zu bedienen, aber nur nutzbar zwischen CICS-Servern.

Objekt orientierte RPCs

- **Corba**
- **RMI**
- **DotNet der Fa. Microsoft, auch als .Net / COM+ / DCOM / ActiveX / DNA / ASP.NET bezeichnet. Die Abgrenzung der Begriffe ändert sich häufig. Wir benutzen durchgängig den Begriff DotNet. DotNet verwendet eine Erweiterung des DCE RPC.**

Corba, RMI und DotNet sind objektorientiert RPCs. RPC-Services werden als Objekte implementiert. Ein Client ruft eine Methode eines bestimmten Objekts auf.

Web Services RPC (SOAP RPC)

- **Der Web Service RPC benutzt das Simple Object Access Protocol (SOAP).**

Für den RPC sind Prozedur, Unterprogramm, Subroutine und Methode (objektorientierte Programmierung) austauschbare Begriffe. Eine Procedur implementiert z.B. einen serverseitigen Service. Ein Prozedur-Aufruf (request) besteht aus einer Nachricht, welche den Namen der aufgerufenen Prozedur sowie eine Liste der übergebenen Parameter enthält.

10.1.4 Synchroner und Asynchroner RPC

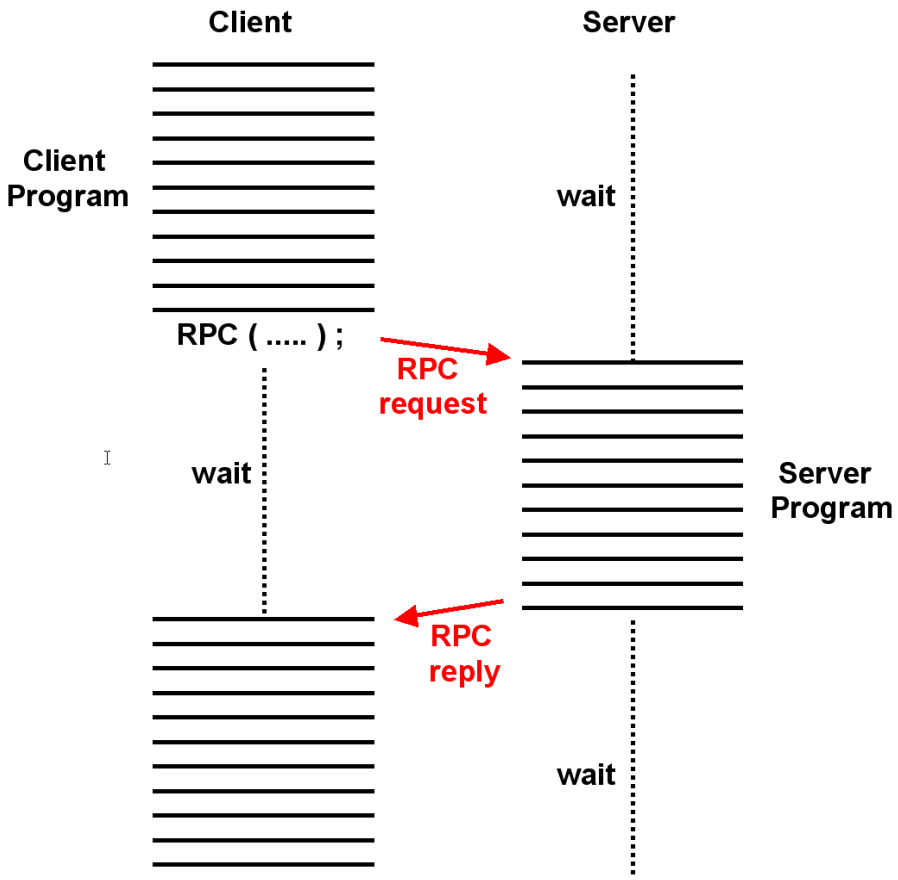


Abb. 10.1.3
Synchroner RPC

Die meisten RPCs werden als "synchrone RPCs" implementiert.

Der Klient sendet eine Nachricht an den Server. Das Eintreffen der Nachricht bewirkt den Aufruf eines von mehreren Services, die der Server bereitstellt. Der Klient wartet (blockiert), bis das Ergebnis des Service Aufrufes verfügbar ist.

Ein asynchrones RPC-Client-Programm wartet nicht auf eine Antwort von dem Server. MQSeries und MDB (Message Driven Beans, siehe Abschnitt 15.4.9) sind Beispiele für einen asynchronen Service Aufruf: Der Klient wartet nicht auf eine Antwort des Servers.

Electronic Mail ist ein Beispiel für einen eher primitiven asynchronen RPC.

10.1.5 Message Based Queuing

Message Based Queuing (MBQ) wird auch als Message Oriented Middleware (MOM) bezeichnet

MBQ ist ein Verfahren zur asynchronen Programm zu Programm-Kommunikation. Es kann verwendet werden, um einen asynchronen RPC implementieren. Programme bekommen die Fähigkeit, dass sie Informationen senden und empfangen können, ohne dass eine direkte Verbindung zwischen ihnen besteht. Programme kommunizieren, indem sie Nachrichten in Message-Queues hinterlegen, und Nachrichten von Message-Queues abrufen.

Führende MBQ Produkte sind:

- **IBM MQSeries, jetzt umbenannt in WebSphere MQ**
- **Microsoft MS Message Queue Server (MSMQ)**
- **Oracle BEA MessageQ**

MQSeries ist seit 1993 verfügbar und ist der unangefochtene Marktführer. Das Produkt ist für mehr als 35 Betriebssystem-Plattformen verfügbar, einschließlich AIX, DG / UX, HP-UX, i5/OS, Sequent, Sinix, Solaris, Tandem, TPF, TrueUnix, VMS / VAX, Windows und z/OS.

Anders als bei anderen MBQ Produkte ist Microsoft MSMQ nur auf Windows-Plattformen verfügbar. Es biete auch keine native Unterstützung für JMS (Java Messaging Service) an.

Apache ActiveMQ ist das populärste Open-Source MBQ Produkt.

Message Based Queuing ist attraktiv für die Integration von heterogenen Hardware, Software und Anwendungsumgebungen.

E-Mail wie SMTP, das Simple Mail Transport Protocol, und X.500 sind primitive MBQ Anwendungen. Merkmale sind:

- **Eine Nachricht wird gesendet.**
- **Die Nachricht kann (oder auch nicht) am Ziel ankommen.**
- **Der Empfänger ist (in der Regel) ein Mensch, der beschließt, die Mail, sofort oder später oder auch gar nicht zu beantworten.**

MBQ unterscheidet sich von elektronischer Post:

- **MBQ bietet ACID Eigenschaften, die sicherstellen, dass eine Nachricht genau einmal ausgeliefert wird.**

Ein MBQ Nachricht wird durch ein Anwendungsprogramm empfangen, das weiß, was damit zu tun ist.

Message Based Queuing ist eine Methode der Programm-zu-Programm-Kommunikation. Programme innerhalb einer Anwendung kommunizieren miteinander durch das Schreiben und Abrufen von anwendungsspezifischen Daten (Nachrichten) zu/von Warteschlangen (Queues), ohne über eine private, dedizierte logische Verbindung miteinander verknüpft zu sein. Messaging bedeutet, dass Programme miteinander durch Senden von Daten in Nachrichten kommunizieren und nicht durch einen direkt Aufruf.

Queuing bedeutet, dass Programme über Nachrichten kommunizieren, die in Queues gespeichert werden. Anders als bei einem synchronen RPC, müssen Programme, die über Queues kommunizieren, nicht gleichzeitig ausgeführt werden. Ein Java-Objekt kann eine Methode eines anderen Java-Objekts durch Senden einer Nachricht aufrufen. Dies ist jedoch eine synchronen und keine asynchrone Kommunikation.

Beim asynchronen Messaging fährt das sendende Programm mit seiner Verarbeitung fort, ohne auf eine Antwort seiner Nachricht zu warten. Im Gegensatz dazu wartet ein synchrones RPC Programm auf eine Antwort, ehe es die Verarbeitung fortgesetzt.

Beim Message Queuing muss sich der Programmierer nicht darum kümmern, ob das Zielprogramm beschäftigt oder nicht verfügbar ist. Er macht sich noch nicht einmal Gedanken, ob der Server heruntergefahren oder die Verbindung zum Server gestört ist. Der Programmierer sendet Nachrichten an eine entfernte (remote) Queue, die einem Anwendungsprogramm zugeordnet ist. Letzteres kann zu diesem Zeitpunkt verfügbar sein oder auch nicht. WebSphere MQ kümmert sich um den Transport zu der Zielanwendung. Ggf. startet es diese sogar.

Für den Anwender ist das zugrunde liegende Kommunikationsprotokoll transparent.

10.1.6 WebSphere MQ

Das IBM MQSeries Software-Produkt ist seit langer Zeit verfügbar. Andere Hersteller vertreiben alternative MBQ Produkte. IBM MQSeries hat jedoch im Markt eine ähnlich dominierende Bedeutung wie CICS.

Kürzlich hat IBM entschieden, MQSeries in WebSphere MQ umzubenennen. Sowohl die alte Bezeichnung MQSeries, der neue Begriff WebSphere MQ und die Kurzform MQ werden mehr oder weniger synonym verwendet.

IBM WebSphere entstand als IBMs Java Application Server-Produkt, heute bekannt unter dem Namen WebSphere Application Server (WAS). IBM hat beschlossen, den Namen WebSphere wiederzuverwenden, um eine Reihe von mehr oder weniger verwandten Software-Produkten (einschließlich WAS) zu kennzeichnen. Neben dem WebSphere Application Server enthält die WebSphere-Familie Produkte wie den WebSphere Designer, den WebSphere Integration Developer, WebSphere Process Server oder WebSphere Portal Server. Es gibt über 200 Software-Produkte in der WebSphere-Familie. Einige der Produkte sind gut miteinander integriert, andere nicht. WebSphere MQ ist ein ziemlich eigenständiges Add-on, das unabhängig von dem WebSphere Java Application Server eingesetzt werden kann.

Die meisten Personen, wenn sie den Namen WebSphere benutzen, meinen damit den WebSphere Application Server (WAS).

Es gibt 2 Versionen von WebSphere MQ:

- **Integrierte Version,**
- **Unabhängige (nicht integrierte) Version, die nur wenige Beziehungen mit dem Rest der WebSphere-Produktfamilie hat.**

Viele Personen benutzen immer noch den Namen MQSeries für die unabhängige (nicht integrierte) Version von WebSphere MQ. Der WebSphere Application Server (WAS) Implementierung des Java Message Service (JMS)-Standards verwendet hierfür eine integrierte Version von WebSphere MQ.

10.1.7 Program-to-Program Communication

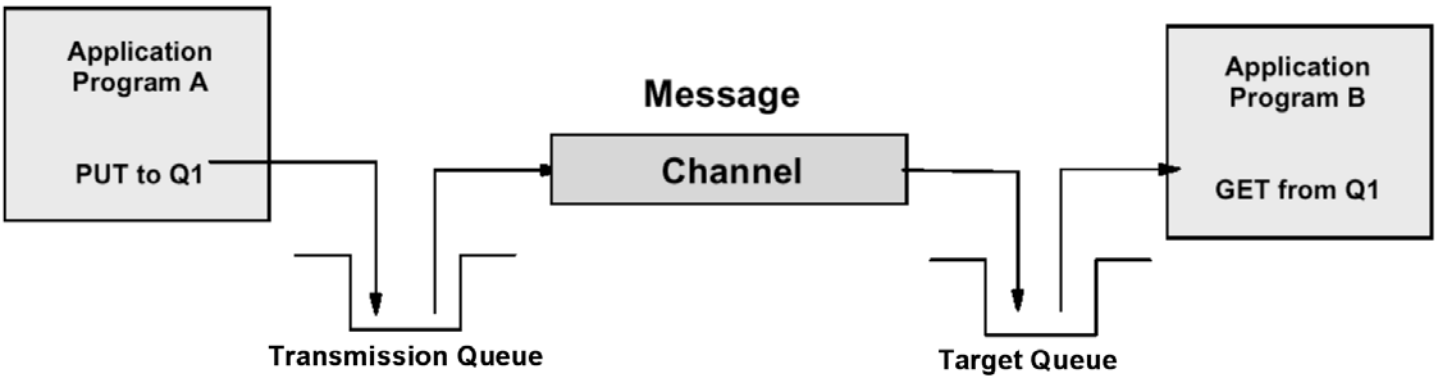


Abb. 10.1.4
MQSeries Operation

Ein Anwendungsprogramm kann Nachrichten an einen anderen Anwendungsprogramm senden, das auf einem entfernten System, wie etwa einem Server oder einem Host läuft.

Der sendende Anwendungsprogramm A speichert die Nachricht in eine lokale Transmission Queue mit dem MQPUT Befehl . Von dort aus wird die Nachricht über eine "Message Channel" zu einer entfernten Target Queue übertragen. Das empfangende Anwendungs-Programm B kann nun die Nachricht aus der Target Queue mit dem MQGET Befehl zu einem Zeitpunkt seiner Wahl abrufen.

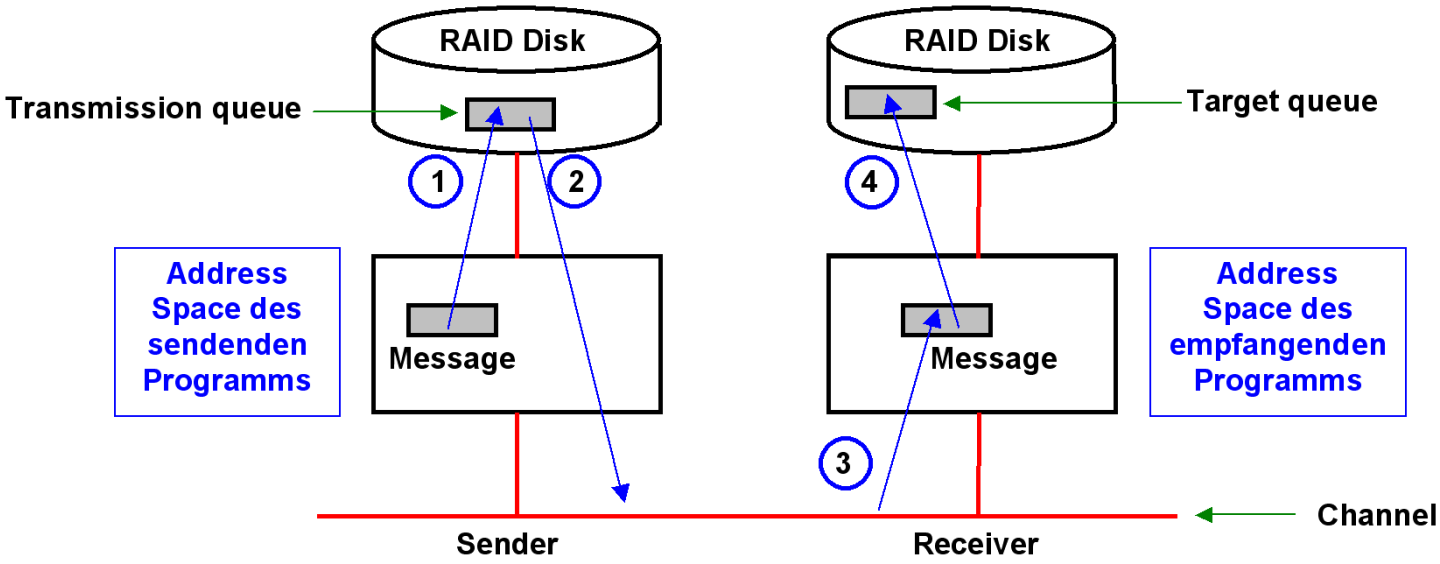


Abb. 10.1.5
Persistente und Non-Persistente Nachrichten

Die "Message Channel" (oder Channel) ist ein MQSeries Konstrukt, das benutzt wird, um Nachrichten von einer Transmission Queue in eine Target Queue zu übertragen. Es ist ein Schicht 5 Protokoll, vergleichbar mit Telnet, 3270, http oder SMTP. Es benutzt TCP/IP (oder SNA) als den darunter liegenden Schicht 4 Transportmechanismus.

WebSphere MQ unterscheidet zwischen persistenten und nicht-persistente Nachrichten. Die Übertragung von persistenten Nachrichten ist gewährleistet, und überlebt Systemausfälle, ähnlich wie Nachrichten, die CICS in seine Log Files schreibt. Nicht-persistente Nachrichten können nach einem Systemausfall nicht wiederhergestellt werden.

Die Persistenz wird mit Hilfe des Speicherns der Nachricht in einem lokalen RAID Festplatten-Subsystem erreicht.

Persistente Nachrichten implementieren eine Kommunikation mit ACID-Eigenschaften (die Zustellung der Nachricht genau einmal wird garantiert). Die Nachricht wird dauerhaft (persistent) durch den Absender gespeichert, ehe sie gesendet wird. Der Empfänger speichert die Nachricht ebenfalls persistent, ehe sie verwendet wird, und bestätigt den Empfang an den Absender.

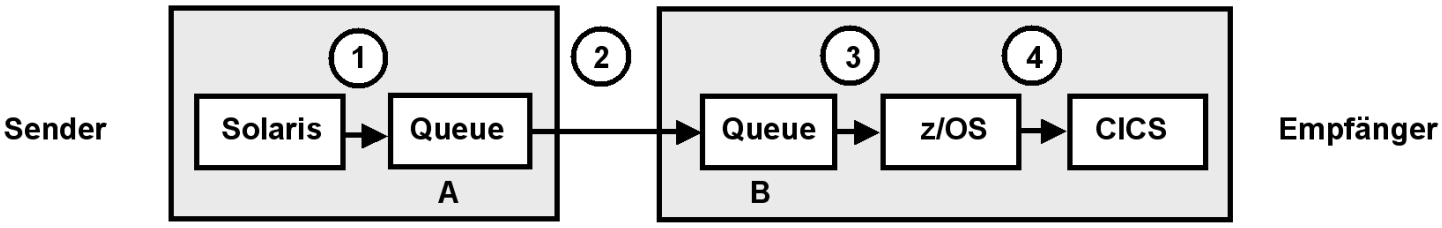


Abb. 10.1.6
Datenübertragung unter ACID Bedingungen

Die persistente Nachrichtenübertragung erfolgt in mehreren Schritten.

Schritt 1: Der Sender speichert die Nachricht persistent in der lokalen Transmission Queue A.

Schritte 2 - 3: Die Nachricht wird über das Netzwerk an die Target Queue B übertragen. Wenn die Übertragung erfolgreich war, sendet der Empfänger eine Bestätigung. Wenn erfolglos, wird der Vorgang beliebig oft wiederholt.

Schritt 4: Der empfangende Anwendungsprogramm (ein CICS Programm in diesem Beispiel) kann nun die Nachricht aus Queue B zu einem Zeitpunkt seiner Wahl abrufen. Dieser Zeitpunkt kann beliebig weit in der Zukunft liegen.

Wenn ein Anwendungsprogramm eine Nachricht in einer Queue speichert, gewährleistet MQSeries, dass die Nachricht:

- **Sicher gespeichert wird,**
- **recoverable ist, und**
- **nur einmal, und genau einmal (exactly once) an die empfangende Anwendung ausgeliefert wird.**

Die Kommunikation ist einseitig.

10.1.8 Middleware

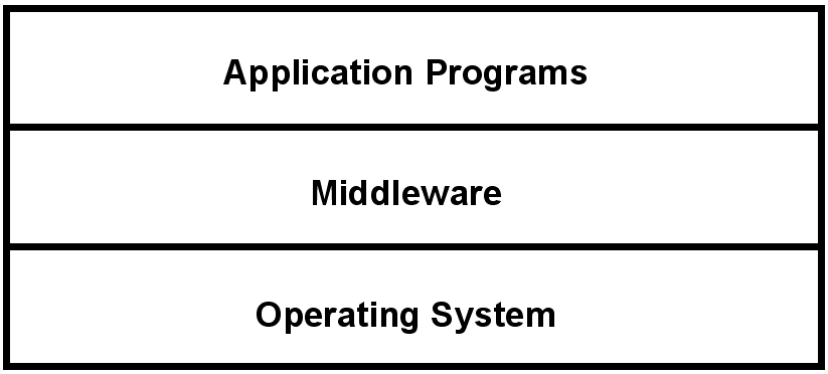


Abb. 10.1.7
Middleware ist eine Laufzeitumgebung für Anwendungssoftware

Der CICS Nucleus ist eine Runtime-Ausführungsumgebung, welche die gleichzeitige Ausführung von mehreren unabhängigen Anwendungsprogrammen (Transaktionen) ermöglicht. Ähnliche ermöglicht ein RPC-Server oder CICS Server die gleichzeitige Ausführung von mehreren unabhängigen Anwendungsprogrammen. Der WebSphere Application Server (WAS) ist eine Laufzeit-Ausführungsumgebung, die gleichzeitige Ausführung von mehreren unabhängigen Java-Anwendungsprogrammen ermöglicht, unter Benutzung von Einrichtungen wie einem Java Servlet-Container und einem Enterprise Java Bean (EJB) Container.

Der Queue Manager ist eine Runtime-Ausführungsumgebung, die die gleichzeitige Ausführung von mehreren unabhängigen WebSphere MQ Anwendungsprogrammen ermöglicht.

Software wie der CICS Nucleus, der RPC-Server, der Web Application Server und der Queue Manager wird gemeinhin als Middleware bezeichnet. Sie alle stellen eine gemeinsame Laufzeitumgebung für mehrere unabhängige Anwendungsprogramme zur Verfügung.

10.1.9 Queue Manager

Der Kern von WebSphere MQ ist der (Message) Queue-Manager (MQM), ein WebSphere MQ Run-Time Programm.

Der Queue Manager stellt eine Laufzeitumgebung für MQ Anwendungen dar, ähnlich wie der CICS Nucleus eine Laufzeitumgebung für CICS Anwendungen darstellt.

Ehe ein Anwendungsprogramm WebSphere MQ auf einem Rechner verwendet, muss ein Queue Manager gestartet werden. Der Queue-Manager besitzt und verwaltet die Ressourcen, die von WebSphere MQ verwendet werden. Zu diesen Ressourcen gehören:

- **Page Sets, die WebSphere MQ Objektdefinitionen und Message Daten enthalten,**
- **Log Files, die verwendet werden, um Nachrichten und Objekte wiederherzustellen falls ein Queue Manager Fehler auftritt,**
- **Prozessor Speicherplatz,**
- **Anschlüsse, über die verschiedenen Anwendungsumgebungen (z.B. CICS, IMS, Batch) auf die WebSphere MQ API zugreifen können,**
- **Den WebSphere MQ Channel-Initiator (später besprochen), der die Kommunikation zwischen WebSphere MQ auf einem sendenden und einem empfangenden Rechner ermöglicht,**

Der Queue-Manager hat einen Namen, und Anwendungen können sich mit einem entfernten Queue Manager über seinen Namen verbinden.

Die Aufgabe des Queue Managers ist es, Warteschlangen und Nachrichten für Anwendungen zu verwalten. Es stellt eine API, die Message Queuing Interface (MQI), zur Kommunikation mit Anwendungen zur Verfügung. Anwendungsprogramme benutzen Funktionen des Queue Managers über API-Aufrufe (Commands). Zum Beispiel legt der MQPUT API-Aufruf eine Nachricht in eine Queue, die durch einentferntes Programm über den API-Aufruf MQGET gelesen werden kann. Dieses Szenario ist in Abb. 10.1.9 dargestellt.

Der Queue Manager kann Nachrichten für den Fall von Anwendung- oder Systemausfällen zwischenspeichern. Nachrichten werden in einer Warteschlange gespeichert, bis eine Empfangsbestätigung durch die MQI eintrifft.

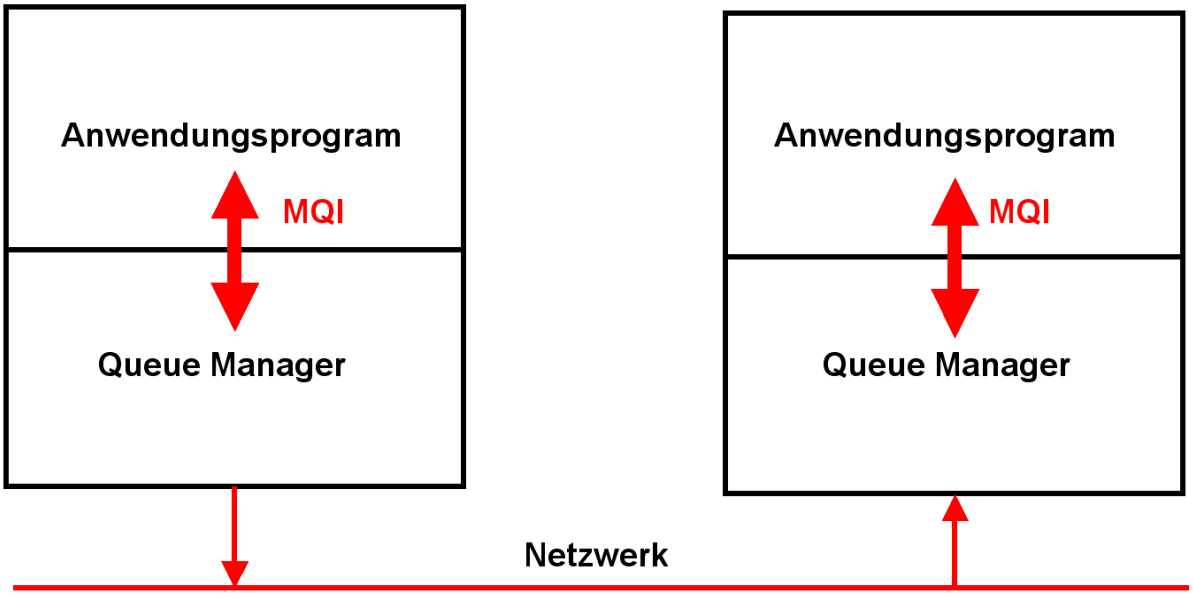


Abb. 10.1.8
MQI als Schnittstelle zwischen Queue Manager und Anwendungsprogramm

WebSphere MQ ermöglicht es Anwendungsprogrammen, miteinander über ein Netzwerk ungleicher Komponenten, wie unterschiedlicher Prozessoren, Subsysteme, Betriebssysteme, Kommunikationsprotokolle und Programmen, die in unterschiedlichen Programmiersprachen geschrieben sind, zu kommunizieren.

Abb. 10.1.8 zeigt die wichtigsten Teile eines WebSphere MQ-Anwendung zur Laufzeit. Anwendungsprogramme benutzen WebSphere MQ-API-Aufrufe, die Message Queue Interface (MQI), um mit einem Queue Manager zu kommunizieren. Die MQI ist eine konsistente Application Program Interface (API) für viele, sehr unterschiedliche Plattformen. Der Queue Manager ist die Laufzeit-Umgebung (runtime environment) von WebSphere MQ.

Anwendungsprogramme können in verschiedenen Programmiersprachen geschrieben werden, einschließlich Java. Der gleiche Queuing-Mechanismus gilt für alle Plattformen. Dies gilt auch für die derzeitig 13 API-Aufrufe der MQI.

10.1.10 Message Queuing Interface

Die wichtigsten Message Queuing Interface (MQI) Befehle sind:

MQCONN	**Verbinden (Connect) mit einem (in der Regel entfernten) Queue Manager**
MQOPEN	**Öffnen (Open) einer spezifischen Queue**
MQPUT	**Eine Message in eine Transmission Queue stellen**
MQGET	**Eine Message aus einer Target Queue auslesen**
MQCLOSE	**Schließen (Close) einer Queue**
MQDISC	**Verbindung zu einem Queue Manager auflösen (Disconnect)**

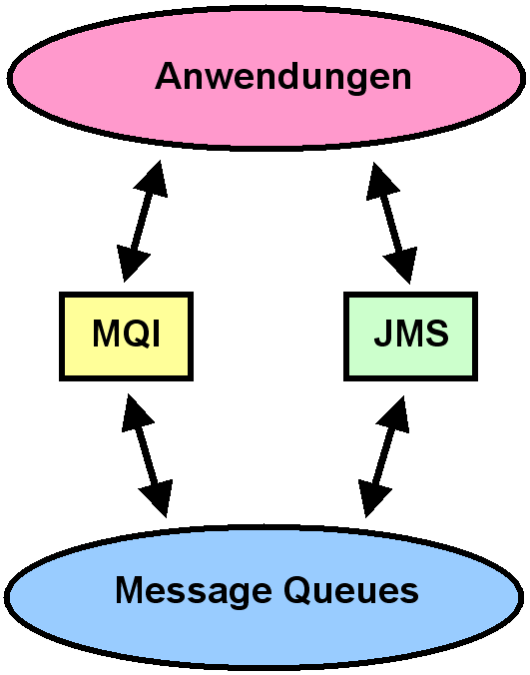


Abb. 10.1.9
JMS als Alternative zu MQI für Java Anwendungen

An Stelle der MQI kann Websphere MQ die „Java Message Service“ (JMS) Interface benutzen. Die beiden wichtigsten APIs sind:

Message Queue Interface (MQI)
- **MQI Funktionen sind verfügbar in den folgenden Sprachen:**
  z/OS Assembler, C/C++, COBOL, Lotus Script, Java, PL/1, VisualBasic, C#, viele andere.

Java Message Service (JMS)
- **Java Standard, der von WebSphere MQ unterstützt wird**
- **Abstracts WebSphereMQ Details**

JMS ist eine von mehreren Schnittstellen für JEE / Enterprise Java Beans.

Die älteste und am weitesten verbreitete API ist die MQI. JMS kann nur von Java-Anwendungen verwendet werden.

10.1.11 Queue Transmission

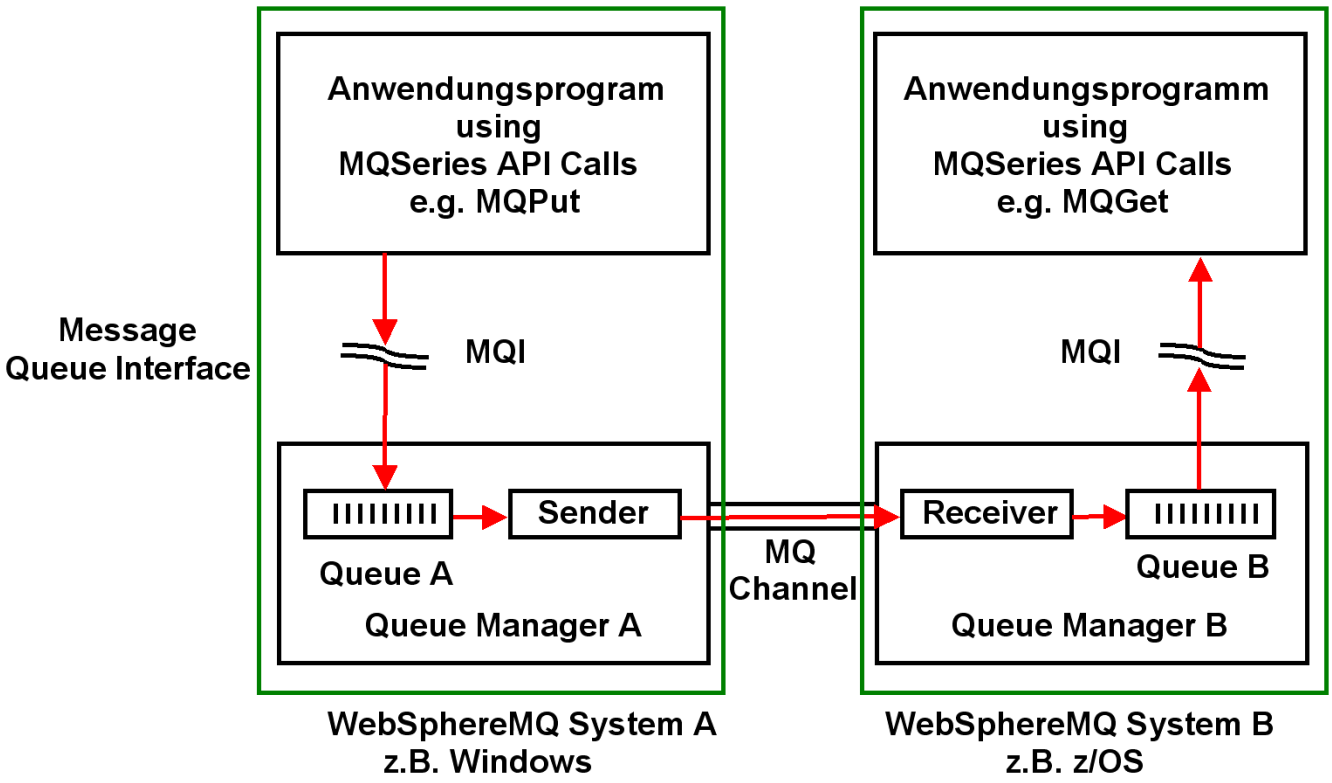


Abb. 10.1.10
WebSphere MQ Operation

Das Anwendungsprogramm stellt eine Nachricht in die Queue A (Transmission Queue), mit Hilfe des MQPUT Calls, ein Bestandteil der Message Queue Interface (MQI). Queue A ist Bestandteil von Queue Manager A. Dieser benutzt sie, um die Nachricht über den MQ-Channel an den Empfänger von Queue Manager B zu senden. Queue Manager B besitzt Queue B (Target Queue, auch als Request Queue, Application Queue, oder Destination Queue bezeichnet), wo die Nachricht gespeichert wird. Das empfangende Anwendungsprogramm kann die Nachricht aus der Target Queue B zu einem von ihm gewählten Zeitpunkt abrufen, mit Hilfe der MQI, beispielsweise mit dem MQI Befehl MQGET.

10.1.12 Sneakernet

Die Infrastruktur eines großen Unternehmens besteht neben dem Mainframe aus einer Vielzahl unterschiedlicher Rechner mit unterschiedlichen Betriebssystemen. Sehr häufig dienen die Ergebnisse auf einem Rechner als Input für den nächsten Rechner.

In der Vergangenheit war es üblich, die Ergebnisse eines Rechners in der Form von Magnetbändern oder Lochkarten zu Fuß zum nächsten Rechner zu tragen. Dies geschah durch einen menschlichen Operator, daher der Ausdruck „Sneakernet“. Noch in den 90er Jahren wurden jeden Abend umfangreiche Daten, die im Werk Bremen der Firma Daimler Benz anfielen, in der Form von Magnetbändern in einen PKW geladen, um am nächsten Morgen im Werk Sindelfingen verarbeitet zu werden.

Andrew S. Tanenbaum: „Never underestimate the bandwidth of a station wagon full of tapes hurtling down the highway.“

WebSphere MQ ist ein perfekter Ersatz für das Sneakernet: Einheitliche Schnittstellen, Sicherheit, sowie asynchrones Verhalten.

10.1.13 Unterstützung einer Service Oriented Architecture

WebSphere MQ ist eine Schlüsselkomponente bei der Umsetzung einer Enterprise Application Integration (EAI) oder einer Service-Oriented Architecture (SOA)-Strategie, in dem es das Messaging-Backbone für über 80 verschiedene Platformen bereitstellt. Die wachsende Bedeutung von EAI und SOA und das Wachstum von Web Services und anderen Verbindungs-Mechanismen sind eindeutig wichtige Entwicklungen.

Einzelheiten hierzu in Band 2, Abschnitt 20.4.

10.2 Queues und Channels

10.2.1 Messages

Abb. 10.2.1
WebSphere MQ Nachrichtenformat

Eine Nachricht besteht aus zwei Teilen:

1. **Message Header (Nachrichtenkopf),**
2. **Daten, die von einem Programm zu einem anderen geschickt werden.**

Der Message Header beinhaltet in jedem Fall einen „Message Descriptor“ (MQMD), kann aber weitere Information beinhalten. Der Message Descriptor identifiziert die Nachricht (Message-ID) und enthält Steuerinformationen (auch als Attribute bezeichnet) wie Nachrichtentyp, Name der Target Queue, Expiration Time, Korrelations-ID, die Priorität und den Namen der lokalen Queue für eine mögliche Antwort.

Eine Nachricht kann bis zu 100 MByte lang sein.

10.2.2 Was ist eine Queue

Die Begriffe “Queue” und “Message Queue” sind austauschbar.

Eine Queue ist eine Datenstruktur die verwendet wird, um Nachrichten zu speichern, bis sie von einer Anwendung abgefragt werden. Es ist einfach ein Speicherplatz der Nachrichten enthält. Die Nachrichten werden entweder von einem Anwendungsprogramm in die Queue gesetzt, oder durch einen Queue Manager als Teil seiner Operation.

Der Queue-Manager ist für die Queues verantwortlich, die er besitzt, für die Speicherung aller Nachrichten, die er empfängt, und das Auslesen von Nachrichten als Reaktion auf eine Anforderung eines Anwendungsprogramms.

Es gibt lokale Queues, die im Besitz des lokalen Queue-Managers sind, und entfernte (remote) Queues (z.B. eine Target Queue), die einem anderen Queue-Manager gehören.

Eine Queue ist ein benanntes Objekt (bis zu 48 Zeichen lang, so lang wie ein z/OS-Dataset-Name), der von einem Queue Typ definiert wird. Der Queue Typ kann eine lokale Queue spezifizieren. Alternativ kann es eine lokale Definition einer Target-Queue sein, die sich auf einem entfernten Rechner mit einem entfernten Queue-Manager befindet.

Lokale Queues sind physische Queues; der Queue-Manager kann eine Nachricht in ihnen speichern. Ein Queue Manager behandelt nicht lokale Queues als Beschreibungen von Queues, die Definitionen enthalten, die durch einen entfernten lokalen Queue-Manager verwendet werden können.

Der MQPUT Befehl speichert eine Nachricht in der Transmission Queue. Es definiert auch in seinem Message Descriptor (MQMD) eine entfernte Target-Queue (die MQI Syntax nennt dies ein "Remote-Queue-Object"). Diese Informationen werden von dem lokalen Queue-Manager verwendet, um die Nachricht an die entsprechende Target Queue eines anderen entfernten Queue-Managers zu übertragen.

10.2.3 Was ist ein Channel?

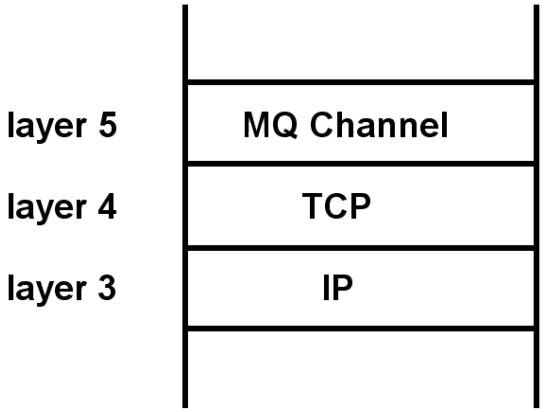


Abb. 10.2.2
Der MQ Channel ist eine höhere Abstraktionsebene als TCP/IP (oder SNA)

Alle Netzwerk-Kommunikation in WebSphere MQ wird über einen "Channel" (auch als "Message Channel" oder "MQ Channel" bezeichnet) durchgeführt. Insbesondere die Übertragung von Nachrichten von einer Transmission Queue an eine Target Queue eines entfernten Queue Managers erfolgt über einen Channel..

Ein Channel ist eine Punkt-zu-Punkt-Verbindung, die zwei verschiedene Queue-Manager miteinander verknüpft. Es ist eine logische Verbindung in der Schicht 5 des OSI-Modells. Telnet, 3270, SMTP, http und ftp sind weitere Beispiele für Schicht 5 Protokolle. Ein Channel wird durch ein sendendes und ein empfangendes- Programm realisiert, die über ein Netzwerk miteinander verbunden sind.

Channels verbergen die zugrundeliegenden Schicht 3 und Schicht 4 Kommunikationsprotokolle (zum Beispiel TCP und IP) vor den Anwendungen. Anwendungsprogrammierer brauchen nicht die physische Adresse des Programms zu kennen, dem sie eine Nachricht senden. Sie stellen ihre Nachrichten in einer Queue, und überlassen es dem Queue-Manager, sich um die Adresse des Ziel-Rechners zu kümmern, und darum, wie die Nachricht dorthin kommt. WebSphere MQ weiß, was zu tun ist, wenn das entfernte System nicht verfügbar ist oder das Zielprogramm nicht läuft oder beschäftigt ist.

Der Begriff "Channel" kann verschiedene Dinge in verschiedenen Umgebungen bedeuten. Eine MQ-Channel ist etwas ganz anderes als z.B. ein FICON-Channel (Abschnitt 5.2).

10.2.4 MQ Message Channel

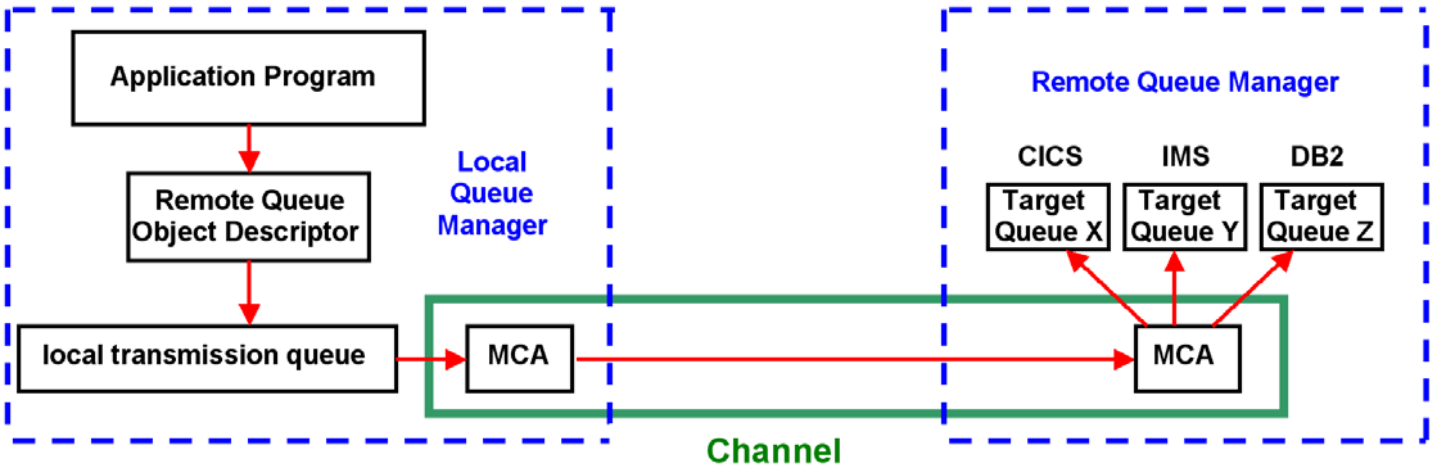


Abb. 10.2.3
Eine MQ Channel Verbindung wird mittels 2 MCAs erstellt

Wie funktioniert die Nachrichtenübertragung?

Ein Message Channel Agent (MCA) ist ein Programm, welches das Senden und Empfangen von Nachrichten steuert. Es gibt einen MCA an jedem Ende eines Channels. Ein MCA nimmt Nachrichten aus der Transmission Queue und legt sie auf die Kommunikationsverbindung. Der andere MCA empfängt Nachrichten und liefert sie an eine Target Queue des entfernten Queue Managers. Einem MQ-Channel wird durch ein Paar von MCAs und das Kommunikationsnetz dazwischen implementiert.

Jeder MQ Message Channel besteht aus zwei Message Channel Agents (MCA), je einem auf dem sendenden und dem empfangenden Rechner. Jeder MCA übernimmt eine der folgenden Rollen:

- **Der sendende MCA öffnet eine bestimmte Transmission Queue für exklusiven Input. Dies bedeutet, keine zwei Message Channels können konfiguriert werden, um Nachrichten von der gleichen Transmission Queue zu übernehmen. Der MCA erhält Nachrichten von einer bestimmten Transmission Queue und sendet sie an den Partner MCA.**

- **Der empfangende MCA empfängt Nachrichten von dem sendenden MCA. Für jede Nachricht, entfernt er den Transmission Queue Message Deskriptor aus der Nachricht und liest deren Inhalt. Er öffnet die Target Queue, die in dem Message Deskriptor für diese Nachricht angegeben ist und schreibt dann die Nachricht in die Target Queue.**

Bitte beachten: Ein Channel ist mit einer einzigen Transmission Queue verbunden, kann aber mit mehreren Target Queues verbunden sein.

MQSeries benutzt den Begriff „Objekt“, um damit Queues und Channels zu bezeichnen.

Um Nachrichten von einem lokalen Queue-Manager an einen entfernten Queue-Manager zu senden, muss der lokale Queue Manager eine Kommunikationsverbindung zum entfernten Queue Manager aufsetzen (zum Beispiel mit dem MQCONNECT Befehl). Der empfangende MCA muss auf dem entfernten Queue-Manager gestartet sein, um Nachrichten über die Kommunikationsverbindung zu empfangen. Diese Einweg-Verbindung, bestehend aus dem sendenden MCA, der Kommunikationsverbindung, und dem empfangenden MCA, wird als Channel bezeichnet. Der sendende MCA übernimmt Nachrichten von einer Transmission Queue und sendet sie über einen Channel an den empfangenden MCA. Dieser empfängt die Nachrichten und legt sie auf eine Target Queue.

Der Remote „Message Queue Object Descriptor“ (MQOD) enthält die Message Descriptor Information, die für den Transmission Queue Header benötigt wird.

MQ-Channels sind unidirektional. Für die bidirektionale Kommunikation müssen zwei Channels (ein Channel Paar) bestehend aus einem Sender Channel und einem Empfänger Channel definiert werden.

10.2.5 Target Queue Definition

Wenn Sie ein WebSphere MQ-Queue öffnen, führt der MQOPEN Befehl eine Namensauflösung durch

Die MQPUT Kommando eines Anwendungsprogramme schreibt eine Nachricht in einer virtuellen Remote Queue, die nur eine Definition einer Target Queue auf einem entfernten Rechner darstellt (remote Queue-Definition). Es ist die Aufgabe des lokalen Queue-Managers, die Nachricht in eine lokale Transmission Queue zu laden, und an die richtige Target Queue weiterzuleiten. Dieses Target Queue ist lokal für einen einem entfernten Queue-Manager auf einem Remote System.

Auf dem entfernten System (und seine Queue-Manager) können mehrere Target Queues existieren, die mit unterschiedlichen Anwendungsprogrammen assoziiert sind, zum Beispiel einer CICS-Transaktion, einer IMS-Transaktion oder einer DB2 Stored Procedure.

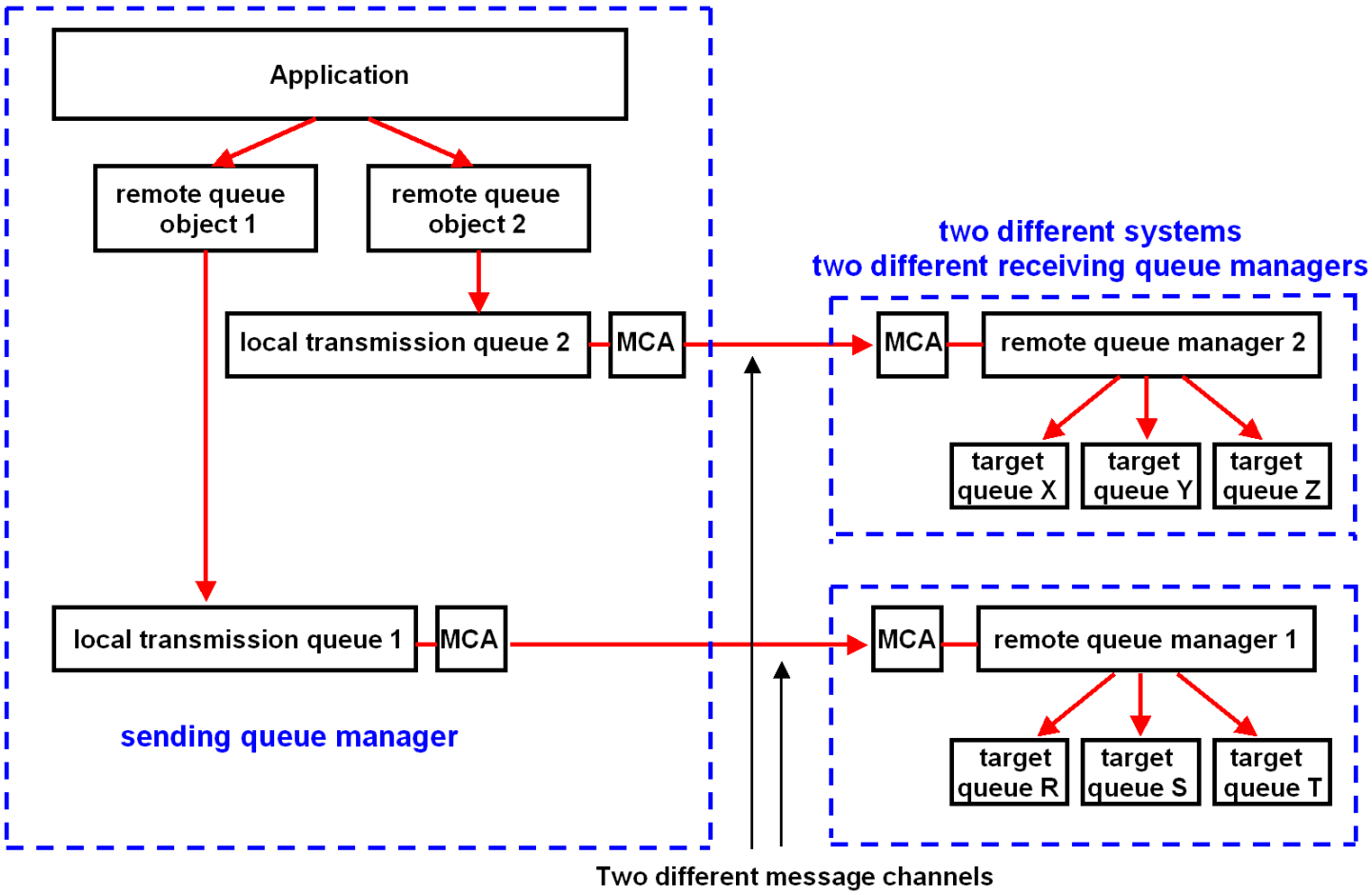


Abb. 10.2.4
Für jede Channel Verbindung wird eine getrennte lokale Transmission Queue benötigt

Eine einzelne Transmission Queue und ein einzelner MQ Message Channel kann mehrere Target Queues auf einem bestimmten Remote-System bedienen. Der Message Deskriptor in der Nachricht, die an das entfernte System gesendet wird, definiert die Target Queue, welche die Nachricht empfangen soll.

Wie in der folgenden Abbildung dargestellt, kann ein lokales System gleichzeitige mehrere WebSphere MQ Verbindungen zu mehreren Remote-Systemen aufrechterhalten. Jede Verbindung benötigt eine separate Transmission Queue und den damit verbundenen MQ Message Channel. Die MQPUT Kommando eines Anwendungsprogramme schreibt eine Nachricht an eine von mehreren entfernten Target Queues. Der lokale Queue-Manager stellt die Nachricht in die entsprechende Transmission Queue. Damit wird die Nachricht automatisch an den richtigen Remote Queue Manager ausgeliefert.

10.2.6 WebSpere MQ Netzwerk

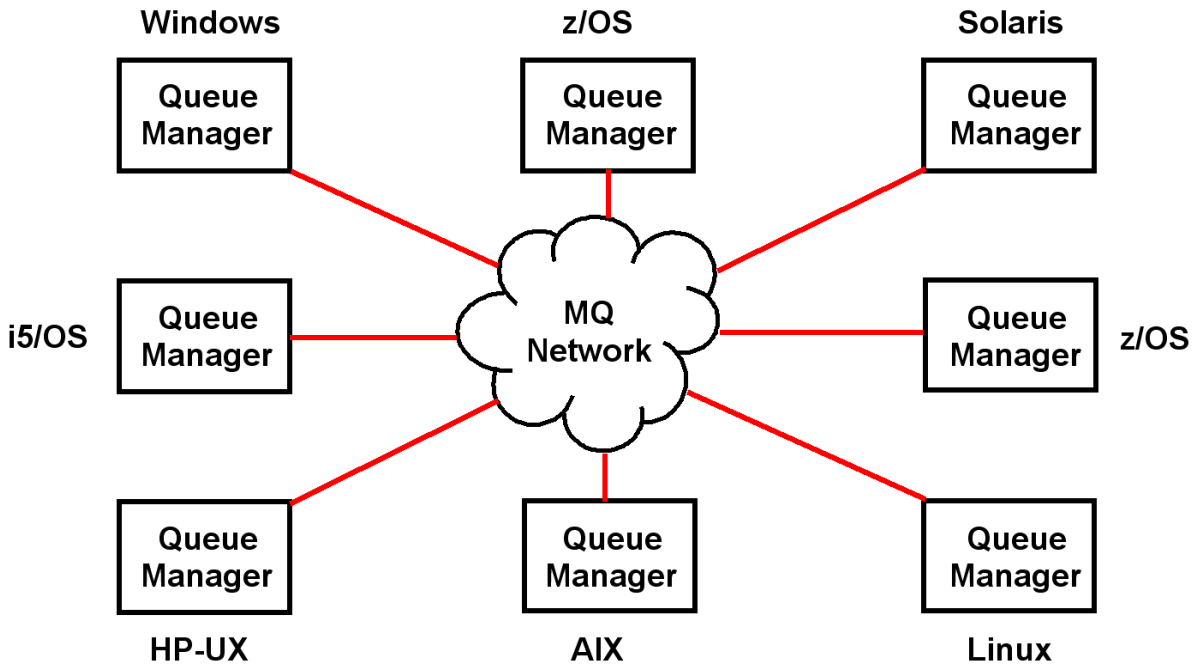


Abb. 10.2.5
Ein weltweites MQ Netzwerk kann Router ähnliche Funktionen haben

Alle Queue Manager in einem Netzwerk von WebSphere MQ Systeme müssen eindeutige Namen haben. Sie ordnen die Namen den Queue Managern zu, wenn Sie das WebSphere MQ-Netz definieren und installiere, genau so, wie Sie IP-Adressen definieren, wenn Sie eine IP-Netzwerk aufzubauen.

Denken Sie daran, genauso wie bei CICS oder Apache, WebSphere MQ ist es egal, auf welcher Betriebssystem-Plattform es läuft.

10.2.7 z/OS Channel Initiator

Unter z/OS läuft WebSphere MQ als z/OS-Subsystem in seinen eigenen Adressraum (Adressraum # 2 in dem in Abb. 10.2.6 gezeigten Beispiel). Innerhalb des Subsystems wird der Queue-Manager durch das Ausführen eines JCL-Prozedur gestartet. Diese spezifiziert die z/OS-Datasets, die Informationen über die Log Files, die Object Definitionen und Message Data (page sets) enthalten. Alle Queue-Manager in Ihrem Netzwerk müssen eindeutige Namen haben, auch wenn sie auf verschiedenen Systemen und Plattformen laufen.

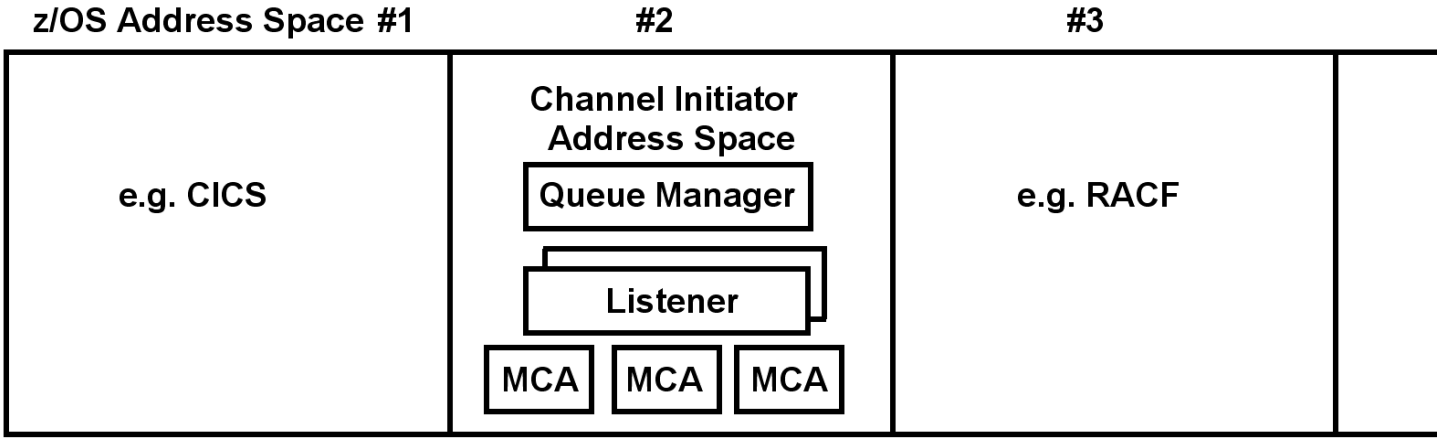


Abb. 10.2.6
Channel Initiator und Listener

Ein Channel Initiator ist ein Bestandteil eines Queue-Managers. In z/OS existiert ein Channel Initiator für jeden Queuemanager (es können mehrere Queue-Manager in getrennten Regionen existieren). Der Channel Initiator läuft zusammen mit seinen Queuemanager innerhalb desselben z/OS Adressenraums. Er beherbergt alle Message Channel Agents (MCAs) für einen Queue-Manager. Tausende von MCA Prozessen können innerhalb des Channels Initiators gleichzeitig laufen. Der Channel Initiator überwacht die vom System definierte Queue SYSTEM.CHANNEL.INITQ, die Initiation Queue für eine Transmission Queue (siehe hierzu den folgenden Abschnitt 10.3.3 und Abb. 10.3.5 dieses Kapitels).

Sie können den Channel Initiator benutzen, um Channels zu starten.

Um Nachrichten zu empfangen, muss ein Listener-Programm auf der Empfängerseite gestartet worden sein.

Das Abhören (monitor) von TCP/IP-Ports geschieht mit Hilfe des WebSphere MQ für z/OS Channel-Initiators. Der Listener, ein WebSphereMQ Komponente, und ein Teil des empfangenden Queuemanagers, überwacht Nachrichten, die auf einem TCP/IP-Port ankommen. Er erstellt einen Message Channel Agent (MCA), der diese Verbindung verarbeitet. Wenn eine Nachricht eintrifft, startet er den Message Channel Agent. Die MCA speichert die Nachricht in die Target-Queue, die in dem Message Descriptor angegeben ist.

Port 1414 ist der Standard TCP/IP Port für WebSphere MQ. Mehrere TCP/IP Listener mit verschiedenen Ports können innerhalb des Channels Initiators gestartet werden. Jeder Listener hört einem bestimmten TCP/IP-Port ab (monitors the Port). Ein Listener wird mit der START LISTENER Befehl innerhalb des Queue-Manager Subsystems gestartet.

Zusätzlich zu den üblichen TCP/IP Listeners unterstützt WebSphere MQ für z/OS auch SNA LU 6.2 Listener.

Das Anwendungsprogramm, das die eingehende Nachricht verarbeiten soll, kann manuell oder automatisch gestartet werden. Um das Programm automatisch zu starten, muss eine Initiation Queue und ein Anwendungsprogramm mit der lokalen Queue assoziiert werden, und der Trigger-Monitor muss laufen. Dies wird in Abschnitt 10.3 diskutiert.

10.2.8 WebSphere MQ auf dem gleichen z/OS System

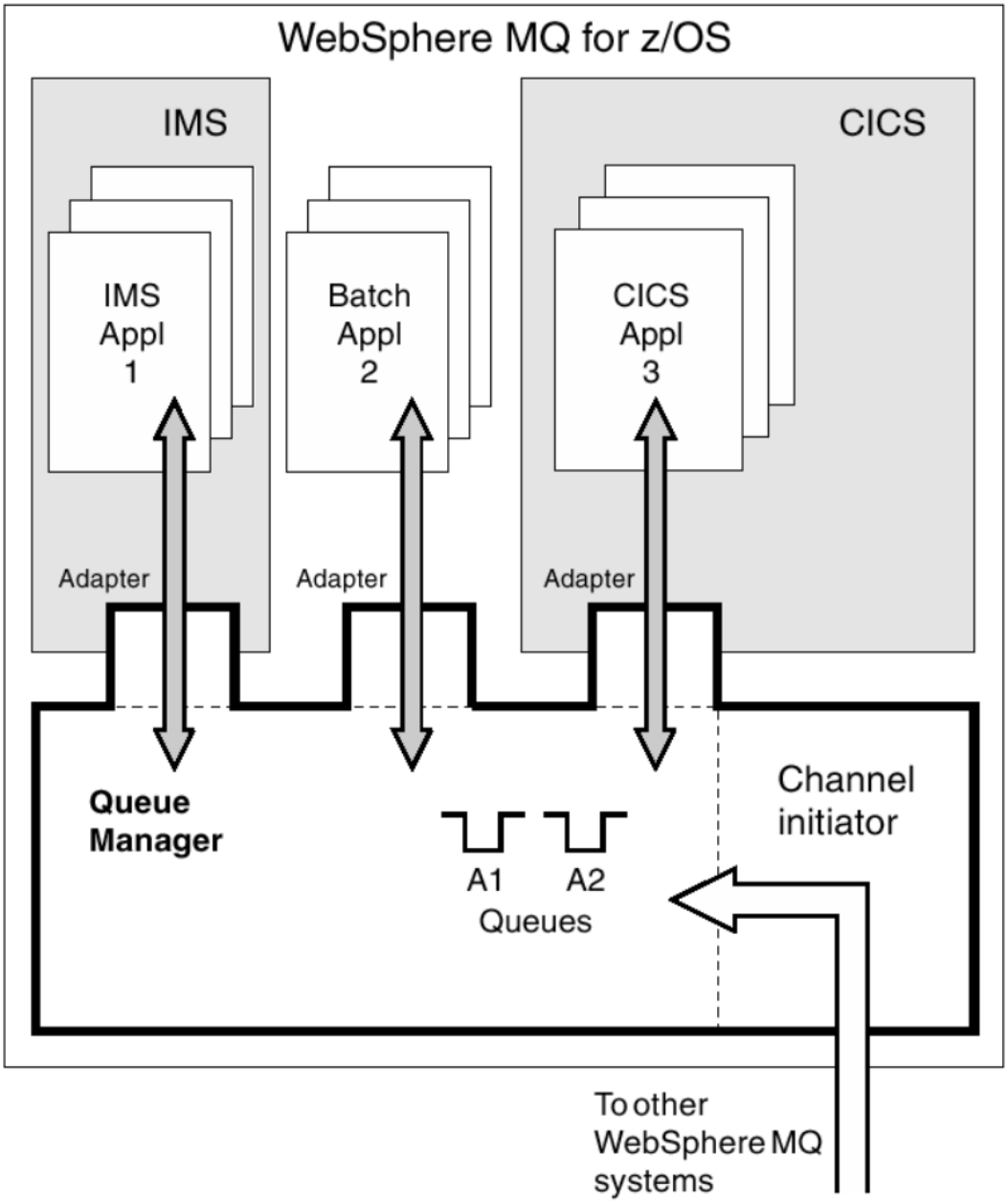


Abb. 10.2.7
WebSphere MQ kann benutzt werden, um Prozesse auf dem gleichen Rechner zu verbinden

MQ Kommunikation wird in der Regel zwischen entfernten Systemen durchgeführt. Es kann aber auch für eine Kommunikation zwischen zwei Regions auf dem gleichen z/OS System eingesetzt werden. Abb. 10.2.7 ist ein Beispiel für eine Programm-zu-Programm-Kommunikation in einem einzigen z/OS-System, zum Beispiel zwischen einer CICS Region und einer IMS-Region.

Eine Übungsaufgabe (Tutorial) für dieses Scenario läuft auf unserem Rechner. Sie ist zu finden unter

http://www.cedix.de/VorlesMirror/Band1/TutorialMQ1.pdf

10.2.9 MQPUT

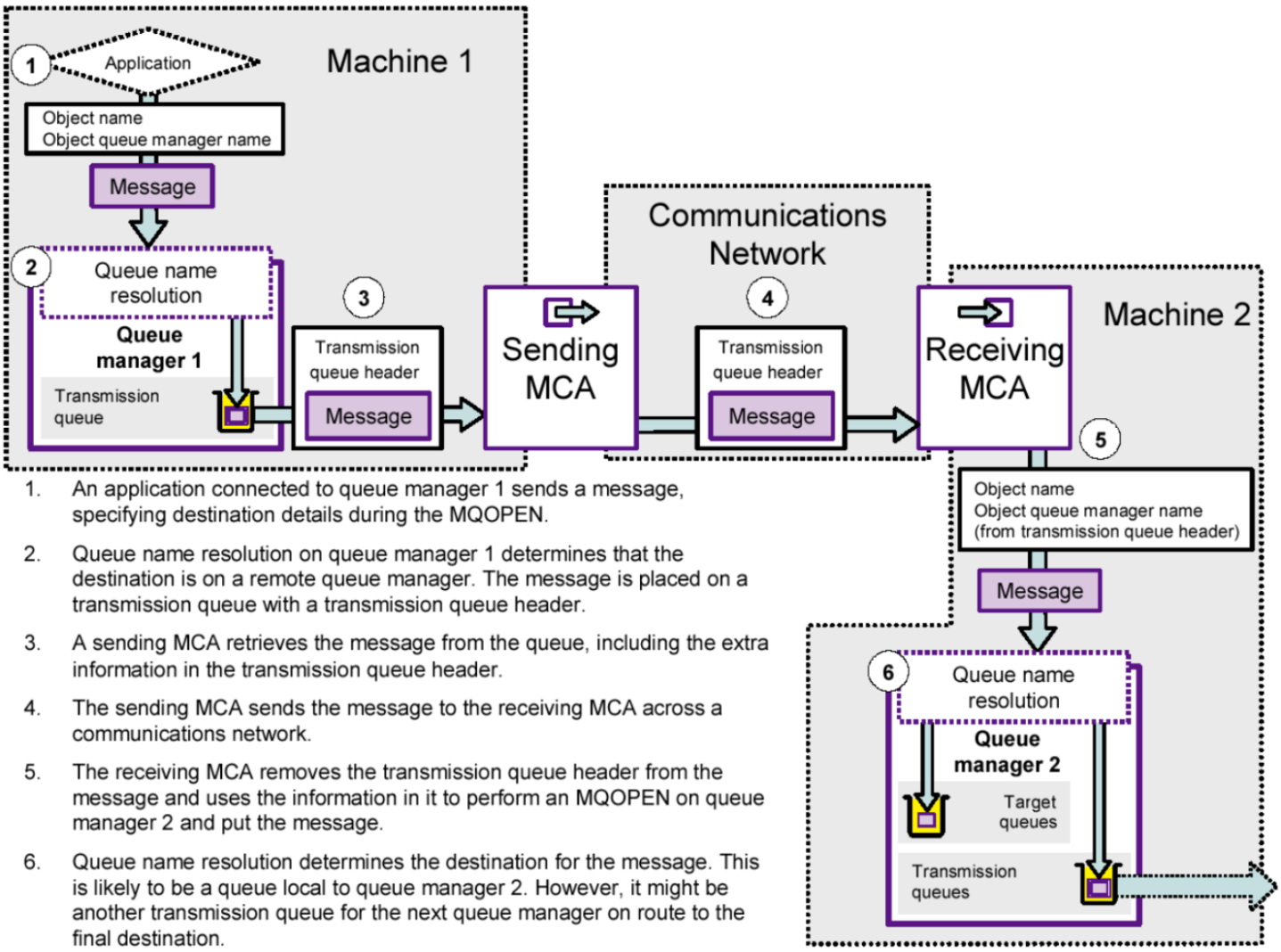
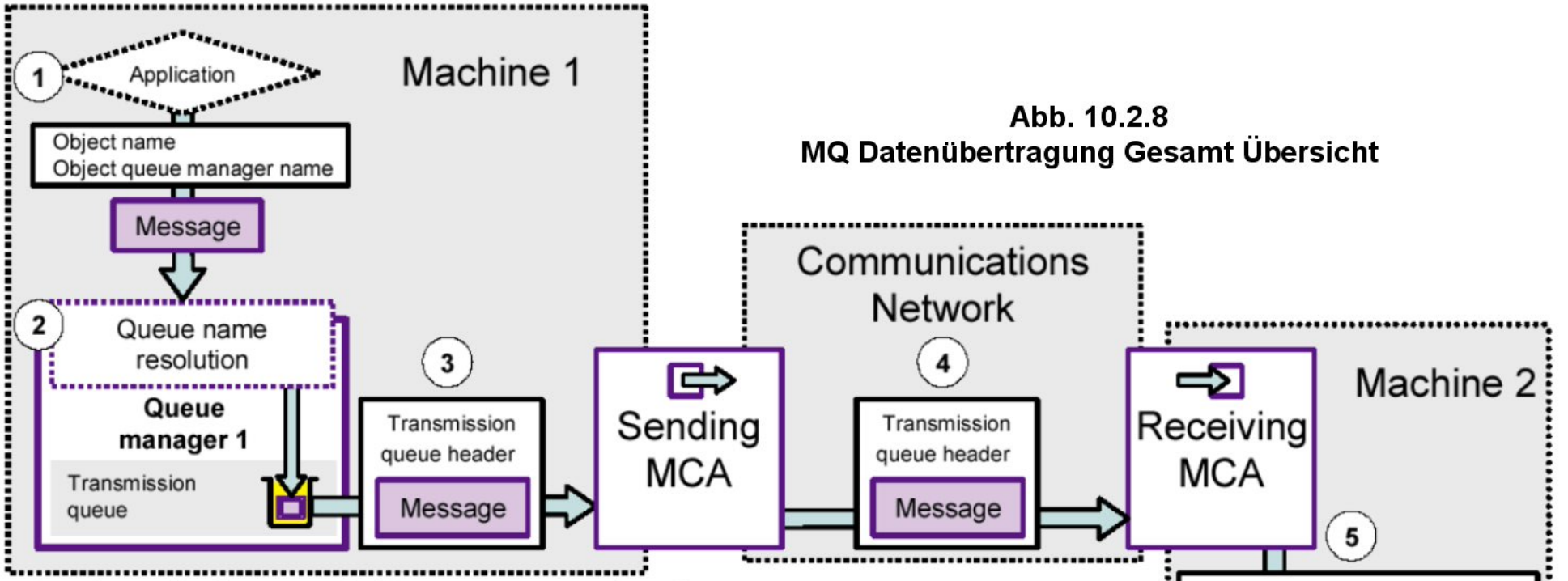


Abb. 10.2.8
MQ Datenübertragung Gesamt Übersicht

Beim Ausführen eines MQPUT Calls übergibt das Anwendungsprogramm neben der Nachricht einen Remote Message Queue Object Descriptor (MQOD) an den lokalen Queue Manager. Dieser enthält spezifisch den Namen des Entfernten Queue Managers und den Namen der entfernten Target Queue.

Abb. 10.2.8
MQ Datenübertragung Gesamt Übersicht



1. An application connected to queue manager 1 sends a message, specifying destination details during the MQOPEN.
2. Queue name resolution on queue manager 1 determines that the destination is on a remote queue manager. The message is placed on a transmission queue with a transmission queue header.
3. A sending MCA retrieves the message from the queue, including the extra information in the transmission queue header.
4. The sending MCA sends the message to the receiving MCA across a communications network.
5. The receiving MCA removes the transmission queue header from the message and uses the information in it to perform an MQOPEN on queue manager 2 and put the message.
6. Queue name resolution determines the destination for the message. This is likely to be a queue local to queue manager 2. However, it might be another transmission queue for the next queue manager on route to the final destination.

Während der Queue Namensauflösung durch den sendenden Queue-Manager, (wo der sendende MCA läuft), generiert der lokale Queue Manager hieraus den Message Descriptor, der in den Message Header (Transmission Queue Header) eingebaut wird. Er wird verwendet, um die Nachricht an den Queue-Manager zu übertragen, auf dem der empfangende MCA läuft. Dieser schreibt die Nachricht auf eine von mehreren Target Queues, die von dem empfangenden Queue-Manager gesteuert werden. Der Transmission Queue Header enthält die folgenden Informationen:

- **Remote Queue-Name:**
  Der Name der Target Queue für die Nachricht, die von der Queue-Manager während der Queue Namensauflösung ermittelt wurde. Wenn der MCA im entfernten Queue-Manager die Nachricht in eine Queue zu schreiben versucht, entnimmt er diesen Namen der Queue dem MQ Object Descriptor (MQOD) beim Öffnen der Queue. MQOD ist eine Struktur (in C++) oder ein Copybook (in Cobol), die als Parameter in dem MQOPEN Anruf des sendenden Anwendungsprogramms benutzt wurde.

- **Remote Queue Manager Name:**
  Dies ist der Name des Target-Queue-Managers, der die Target Queue hostet. Dies ist möglicherweise nicht der Name des Queue Managers, der die Nachricht empfängt. Dieser Fall kann auftreten, wenn dieser Queue Manager nicht die Final Destination für die Nachricht ist. Eine MQ Channel Verbindung kann über ein Netzwerk von Queue Managen zu dem empfangenden Queue Manager erstellt werden.

10.3 Trigger

10.3.1 MQ Client/Server Communication

In einer WebSphere MQ Client/Server-Konfiguration arbeiten ein Rechner und sein Queue Manager als Client, und ein anderer Rechner und sein Queue-Manager arbeitet als Server.

Streng genommen stellt WebSphere MQ eine Peer-to-Peer-Kommunikation dar. Es ist jedoch nützlich, den sendenden Queue-Manager als Client und dem empfangenden Queue-Manager als Server zu betrachten, weil WebSphere MQ häufig in einem Request / Response-Modus betrieben wird. Das sendende System erwartet (asnchronously) eine Reaktion von dem empfangenden System beim Senden einer Nachricht (Request-Message), und das empfangende System reagiert durch das Senden einer Response Message.

In diesem Text benutzen wir den Begriff MQ-Client für einen Queue-Manager, der eine Nachricht an eine Target Queue sendet, und den Begriff MQ-Server für einen Queue-Manager, der die Target Queue unterhält, und der potentiell mit einer Antwort-Nachricht an den MQ Client reagiert.

Bitte denken Sie daran, WebSphere MQ ist eine unidirektionale Kommunikation. Um in einem Request / Response-Modus zu arbeiten, muss der MQ Server eine zweite unidirektionale Verbindung vom MQ-Server an den MQ-Client verwalten.

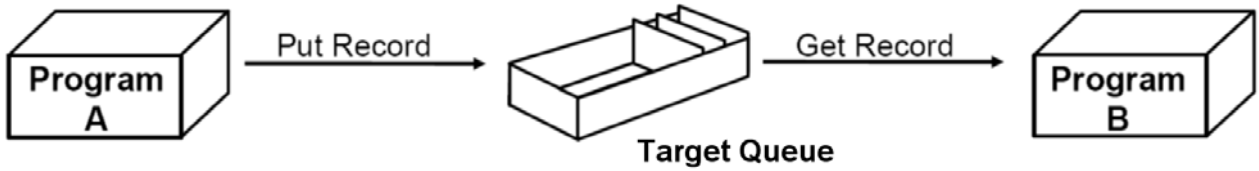


Abb. 10.3.1
Fire and forget

In seiner einfachsten "Fire and Forget"-Form, überträgt das sendende WebSphere MQ Programm A eine Nachricht an einen empfangenden Programm B und vergisst die ganze Sache. Es liegt an der empfangenden Programm B, die Nachricht irgend wann von der Target Queue zu extrahieren.

Die Nachricht befindet sich entweder auf dem sendenden Rechner oder dem Target Rechner, je nachdem, ob die Nachricht bereits übermittelt wurde oder nicht. Programm A kümmert das alles nicht.

Wenn das empfangende Programm B Programm nicht verfügbar ist, wird die Nachricht in einer Queue bleiben und später verarbeitet. Programm B kann den ganzen Tag lang warten, ehe es mit einem MQGET Befehl die Nachricht ausliest.

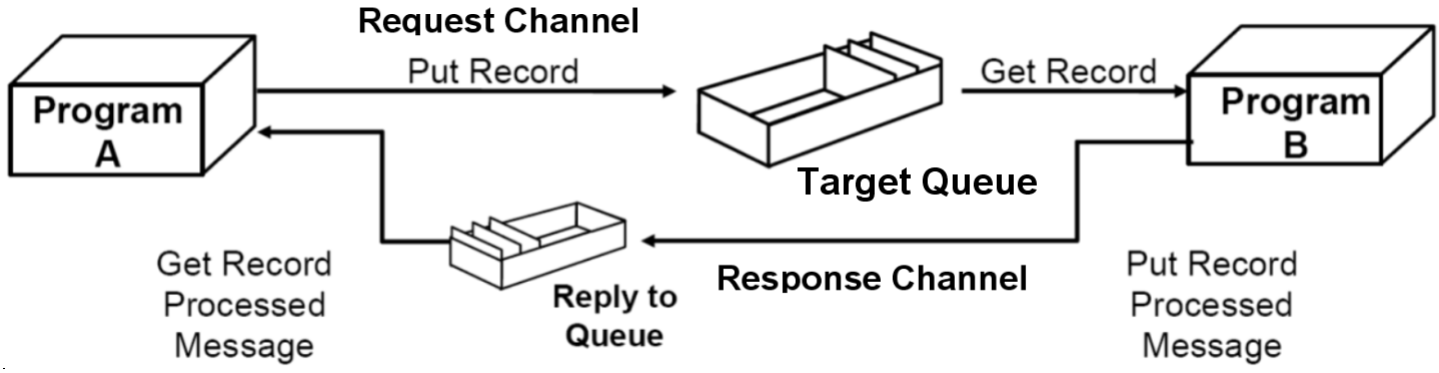


Abb. 10.3.2
Request / Response Modus

Eine Alternative ist, Programm B automatisch zu starten, wenn eine Nachricht eintrifft (oder nachdem eine bestimmten Anzahl von Nachrichten eingetroffen sind).

Obwohl WebSphere MQ asynchron arbeitet, kann ein "Request / Response" Modus einen synchronen Betrieb simulieren. Es wird recht häufig auf diese Weise verwendet. Dazu enthält der Message Descriptor der Request Nachricht eine "Reply to ...."-Anzeige.

Da WebSphere MQ asynchron und unidirektional arbeitet, werden zwei MQ-Kanäle mit zwei Message Channel Agent (MCA) Paaren für einen Request / Response Betrieb benötigt.

10.3.2 MQI Channel

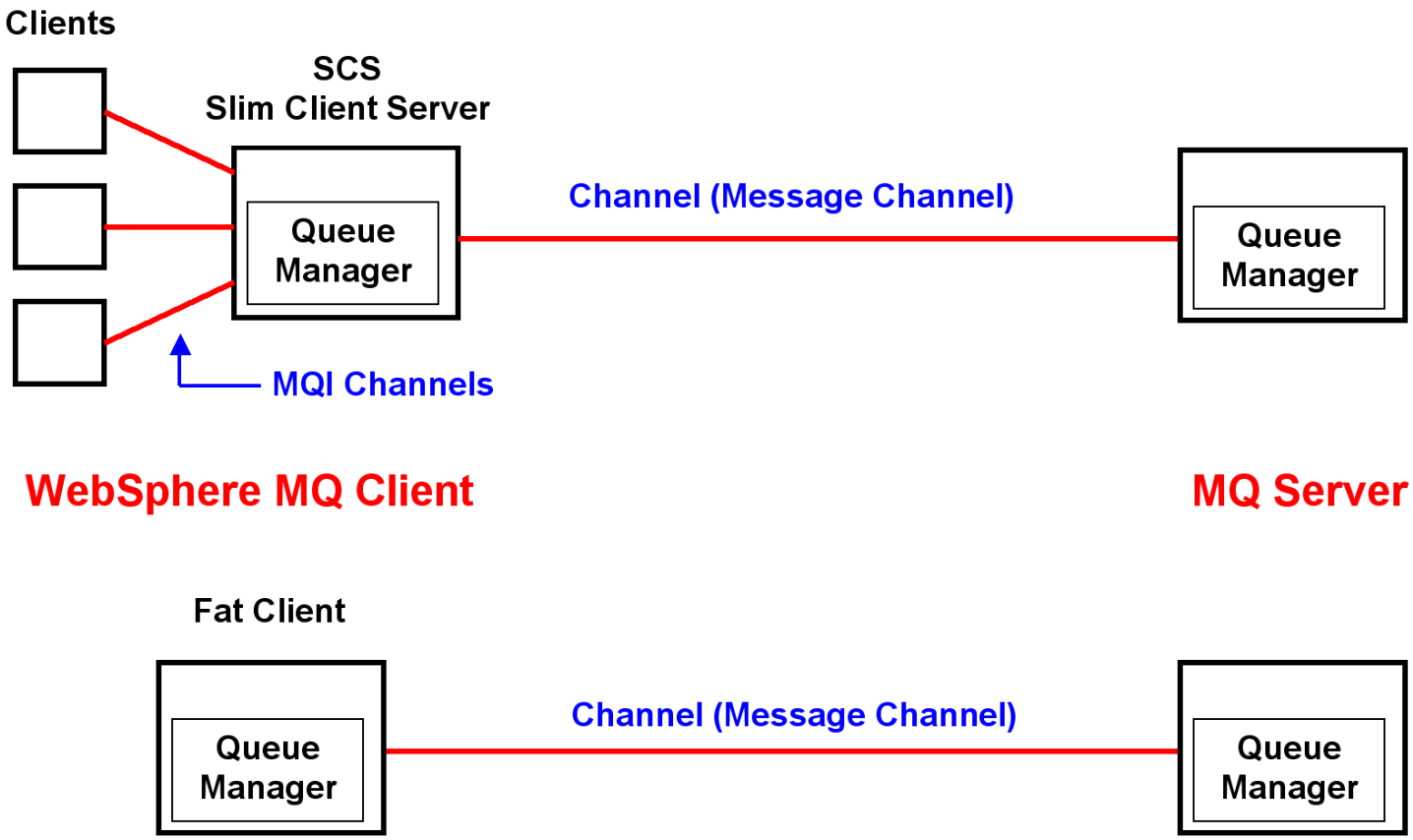


Abb. 10.3.3
MQI Channels verbinden Thin Clients mit einem Thin Client Server

Bei den WebSphere MQ Netzwerk-Kommunikationsverbindungen existieren zwei verschiedene Arten von Channels: Message Channel und MQI Channel

Message Channel

Ein Message Channel (in der Regel nur als Channel bezeichnet) ist das, was wir bisher diskutiert haben. Er verbindet zwei Queue Manager durch Message Channel Agents (MCAs) auf jeder Seite. Ein Message Channel ist unidirektional, besteht aus zwei Message Channel Agents (ein Sender und ein Empfänger) und einem Kommunikationsprotokoll. Ein MCA überträgt Nachrichten von einer Transmission Queue zu einer Kommunikationsverbindung, und von einer Kommunikationsverbindung zu einem Target Queue. Für eine bidirektionale Kommunikation ist es notwendig, ein Paar von Kanälen, bestehend aus einem Sender und einem Empfänger-Channel zu definieren.

Der Queue-Manager überträgt Nachrichten an andere Queue-Manager über Channels. Hierzu werden vorhandenen Netzwerk-Einrichtungen, in der Regel TCP / IP, benutzt.

Channels sind unidirektional. Die meisten der Zeit verstehen wir unter einem Channel einen Message Channel.

MQI Channel

Ein spezieller Fall eines Channels ist der Message Queue Interface (MQI) Channel. Es verbindet einen WebSphere MQ-Client (auch als Slim-Client bezeichnet) mit einem Queue-Manager. WebSphere MQ Slim Clients verfügen nicht über einen eigenen Queue-Manager.

Ein MQI Channel ist bidirektional.

WebSphere MQ unterscheidet zwischen verschiedenen Arten von Clients:

- **Slim client (oder WebSphere MQ client)**
- **Fat client**

Der Unterschied zwischen einem Slim Client und einem Fat Client besteht in der Art, wie Nachrichten gesendet werden. Die Transmission Queue befindet sich entweder in dem Endbenutzer-Arbeitsplatz (Fat Client) oder auf einem „Slim Client-Server“ (SCS). Ein Slim Client muss an einen WebSphere SCS angeschlossen sein, der die Queue-Manager Services zur Verfügung stellt. An einen SCS sind in der Regel viele Slim Clients angeschlossen.

Fat Clients haben eine lokalen Queue-Manager, Slim Clients nicht.

Ein Slim WebSphere MQ-Client, der sich nicht mit einem SCS verbinden kann, kann nicht arbeiten, weil sich der Queue-Manager und die Queues für den Slim Client auf dem SCS befinden. WebSphere MQ-Clients sind häufiger Slim Clients und seltener Fat Clients.

Ein Fat Client enthält einen eigenen Queue Manager. Für eine Gruppe von Slim Clients stellt ein WebSphere MQ Slim Client Server (SCS) Queue Manager Services zur Verfügung.

Hinweis: Die "WebSphere MQ-Client für Java" ist ein Slim Client.

10.3.3 Trigger Event

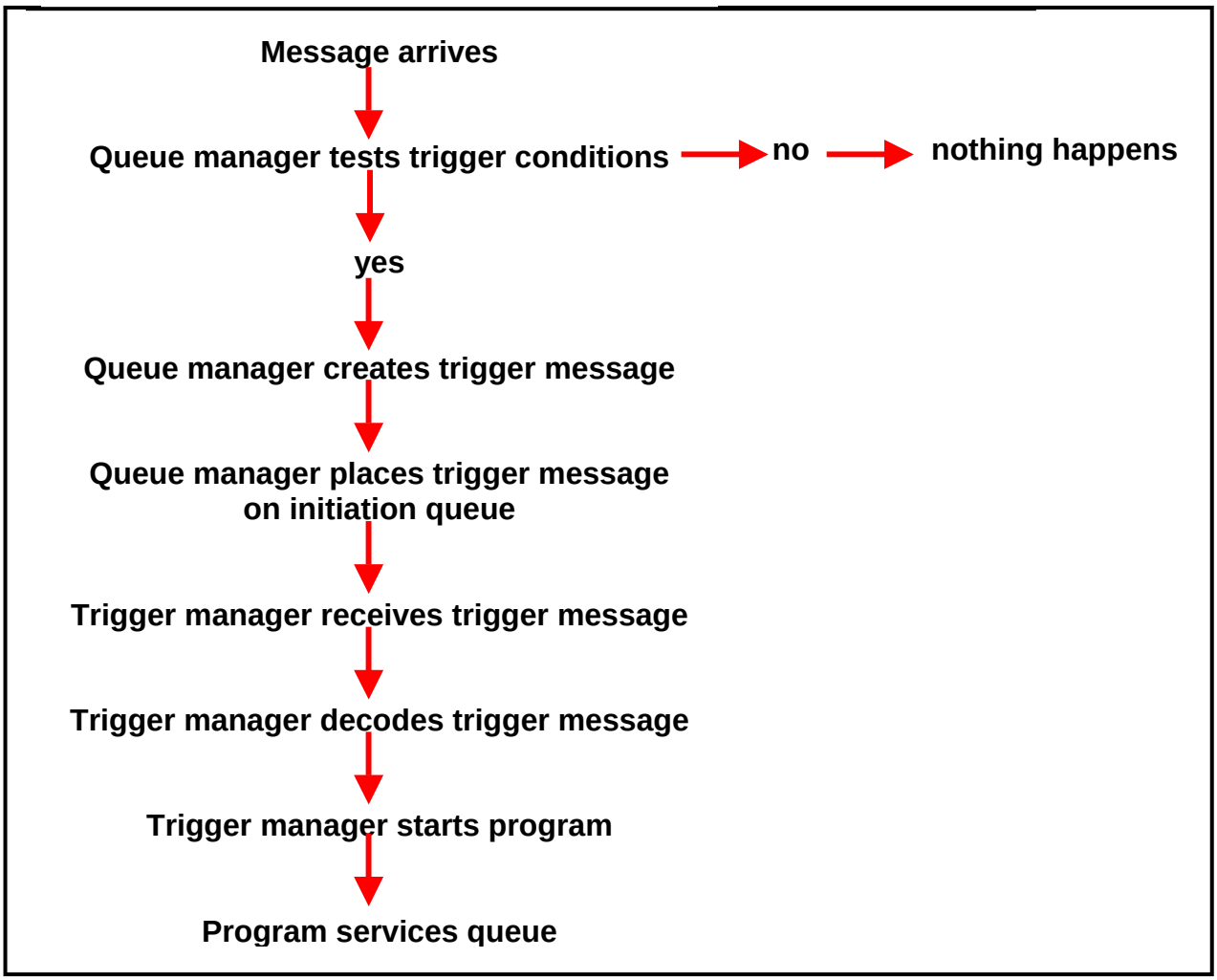


Abb. 10.3.4
Verarbeitung eines Trigger Events

Die Target Anwendung kann die Nachricht aus der Target Queue zu einem beliebigen Zeitpunkt extrahieren. Dies kann zu einer langen Wartezeit führen, ehe eine Antwort gesendet wird. Um pseudosynchron zu arbeiten, muss die Target-Anwendung "getriggert" werden. Die Target Anwendung wird informiert, dass eine Nachricht in der Target Queue angekommen ist, die ihre Aufmerksamkeit erfordert. Dazu wird ein „Triggering Mechanismus" benötigt.

Ein "Trigger-Event" ist ein Ereignis, das den entfernten Queue-Manager motiviert, eine "Trigger-Message" zu generieren. Ein Trigger-Event wird in der Regel in dem Deskriptor einer Nachricht angegeben.

Wenn Triggerung für eine Target Queue aktiviert ist und ein Trigger-Event auftritt, sendet der Queue Manager eine "Trigger Message" an eine lokale Queue, die als Initiation Queue bezeichnet wird. Das Vorhandensein der Trigger-Nachricht in der Initiation Queue zeigt einen Trigger Event an. Eine Initiation Queue kann mehrere Target Queues bedienen, aber eine Target Queue ist immer einer bestimmten Initiation Queue zugeordnet.

Nehmen wir die folgende Situation an: Program A sendet eine Nachricht an ein Remote Programm B, das getriggert werden soll (siehe Abb. 10.3.4 und 10.3.5):

WebSphere MQ Application Programming Guide, March 2003, SC34-6064-03, p.196

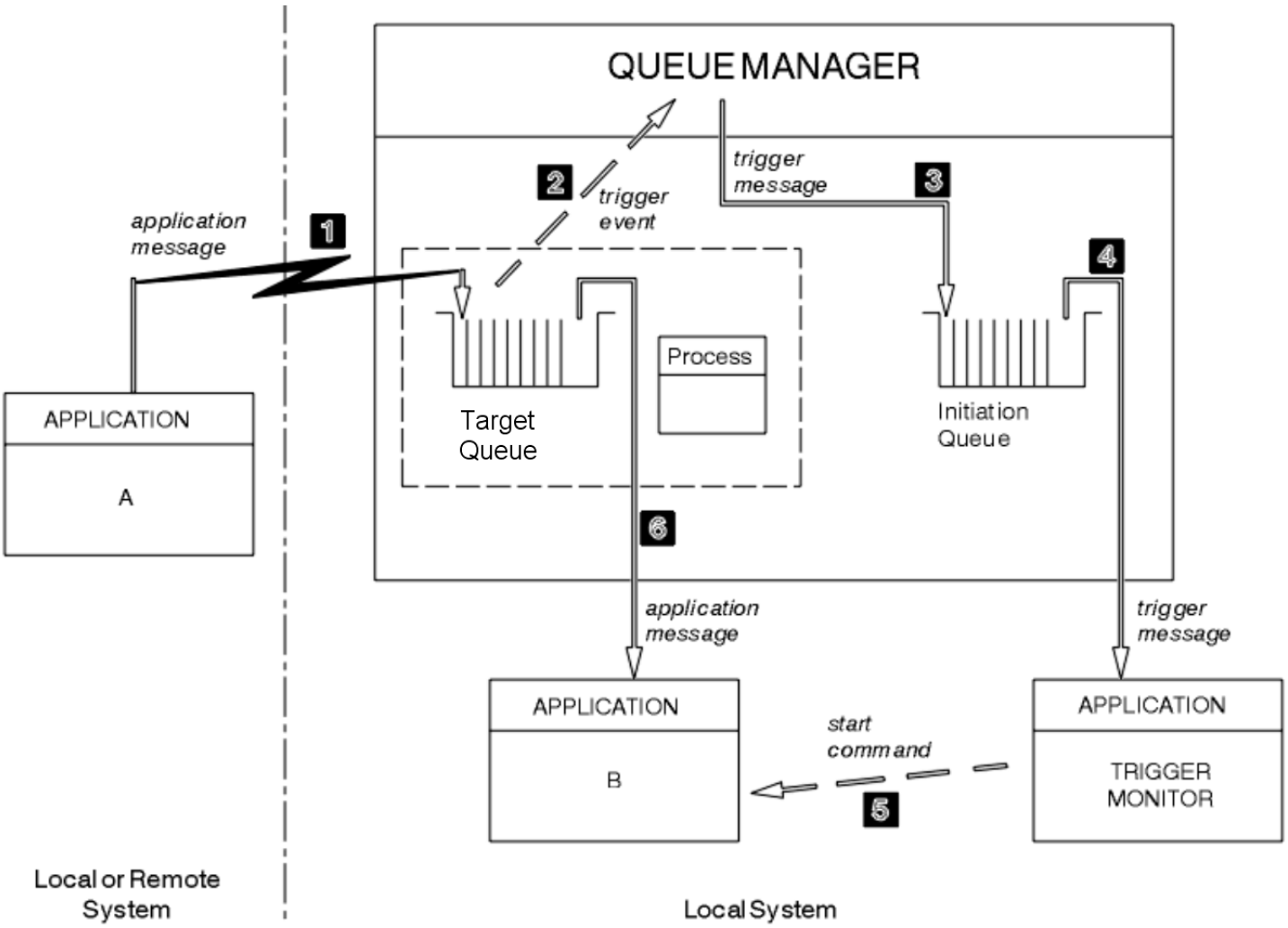


Abb. 10.3.5
Aufgabe der Initiation Queue

Die Ankunft einer Nachricht in einer getriggerten Queue startet eine komplexe Folge von Ereignissen.

1. **Programm A, das entfernt zu dem empfangenden Queue-Manager ist, stellt eine Nachricht in die Target Queue. Beachten Sie, dass kein Anwendungsprogramm diese Nachricht erwartet.**
2. **Der empfangende Queue-Manager überprüft die Nachricht. Er stellt fest, ob die Bedingungen erfüllt sind, unter denen er ein Trigger-Event erzeugen muss (Daten im Message Descriptor). Wenn ja, wird ein Trigger Event generiert.**
3. **Der Queue-Manager erstellt eine Trigger-Nachricht und legt sie in die spezifische Initiation Queue, welche der Target Queue zugeordnet ist. Dazu muss eine spezielle Anwendung (Trigger-Monitor) das Erscheinen der Trigger Message in der Initiation Queue zur Kenntnis nehmen..**
4. **Der Trigger Monitor liest die Trigger-Nachricht von der Initiation Queue.**
5. **Der Trigger-Monitor startet das Programm B (die Server-Anwendung).**
6. **Programm B öffnet die Target Queue und liest die Nachricht.**

10.3.4 WebSphereMQ Zugriff auf eine CICS Transaction

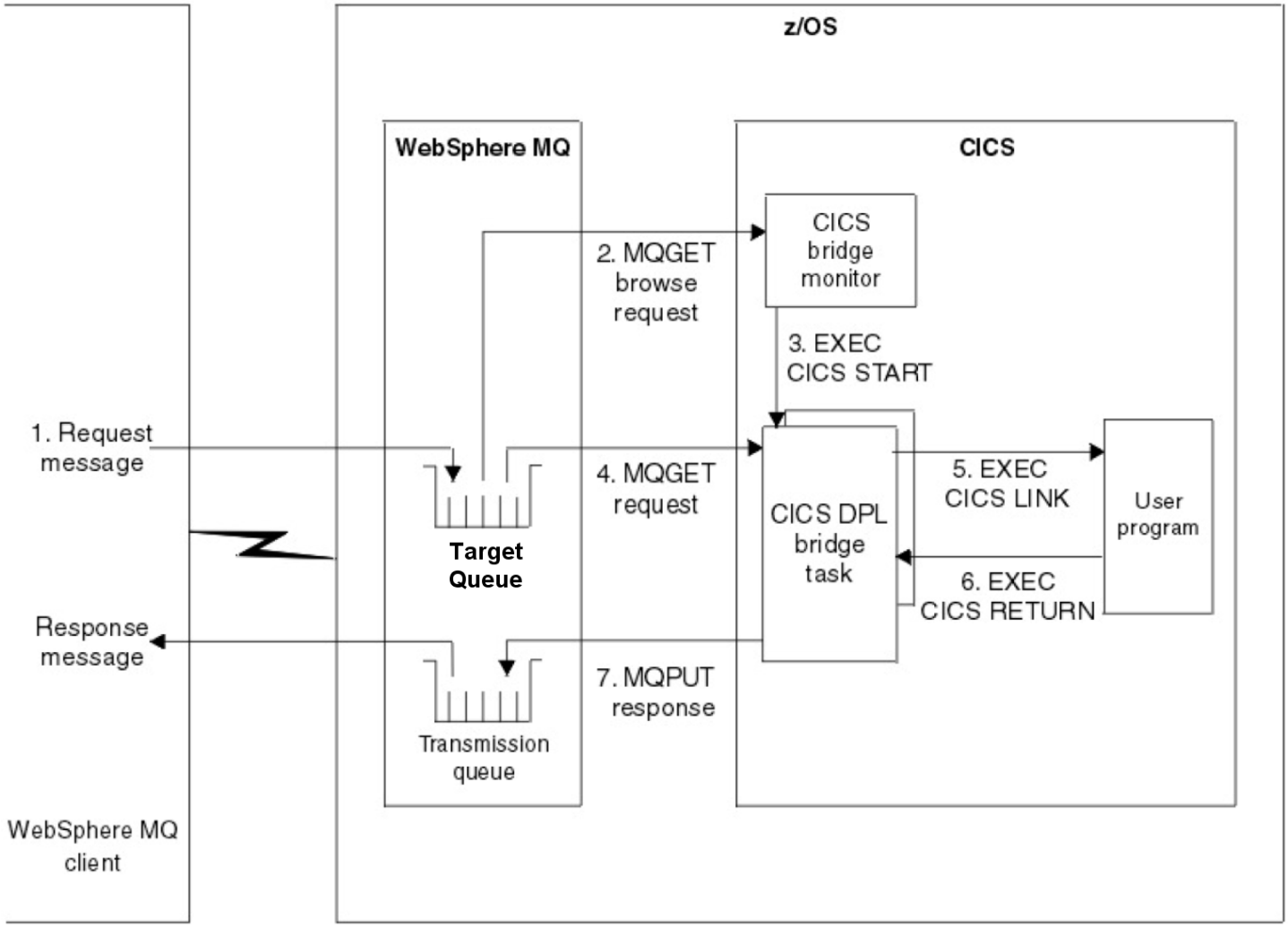


Abb. 10.3.6
Der CICS Bridge Monitor startet mittels des normalen DPL ein CICS Anwendungsprogramm

WebSphereMQ im Request/Response-Modus ist eine Alternative zur Verwendung eines 3270-Emulator beim Zugriff auf eine CICS Transaktion. Dies kann mit Hilfe des "MQ-CICS-Bridge" Software-Produktes durchgeführt werden. Die MQ-CICS-Bridge besteht aus 2 Komponenten, dem Bridge Monitor und der DPL Bridge Task.

Abb. 10.3.6 zeigt, wie es funktioniert:

1. **Eine Nachricht, die eine CICS Transaktion aufrufen soll, wird in der Target Queue empfangen. Die Nachricht enthält einen Trigger Event.**
2. **Der CICS Bridge Monitor übernimmt die Funktion des Trigger Monitors.**
3. **Der CICS Bridge Monitor startet eine spezielle "CICS DPL Bridge" Task.**
4. **Die CICS DPL Bridge Task liest (und entfernt) die Nachricht aus der Target Queue.**
5. **Der CICS DPL Bridge Task verwendet den Distributed Program Link (DPL) Aufruf und greift auf eine CICS Region zu (z.B. ein Application Owning Region - AOR). Diese startet die Transaktion und führt sie aus.**
6. **Die Transaktion übergibt das Ergebnis an die CICS DPL Bridge Task, z.B. über eine COMMAREA.**
7. **Der CICS DPL Bridge Task verwendet einen MQPUT Anruf, um das Ergebnis in die WebSphere MQ Transmission Queue zu speichern.**
8. **Der Queue-Manager transportiert die Nachricht in der Transmission Queue über eine MQ-Channel an den WebSphere MQ-Client.**

Eine detaillierte Beschreibung ist zu finden in einer Masterarbeit von Tobias Busse:
http://www.cedix.de/DiplArb/Busse04.pdf

10.3.5 Multiple Target Queues

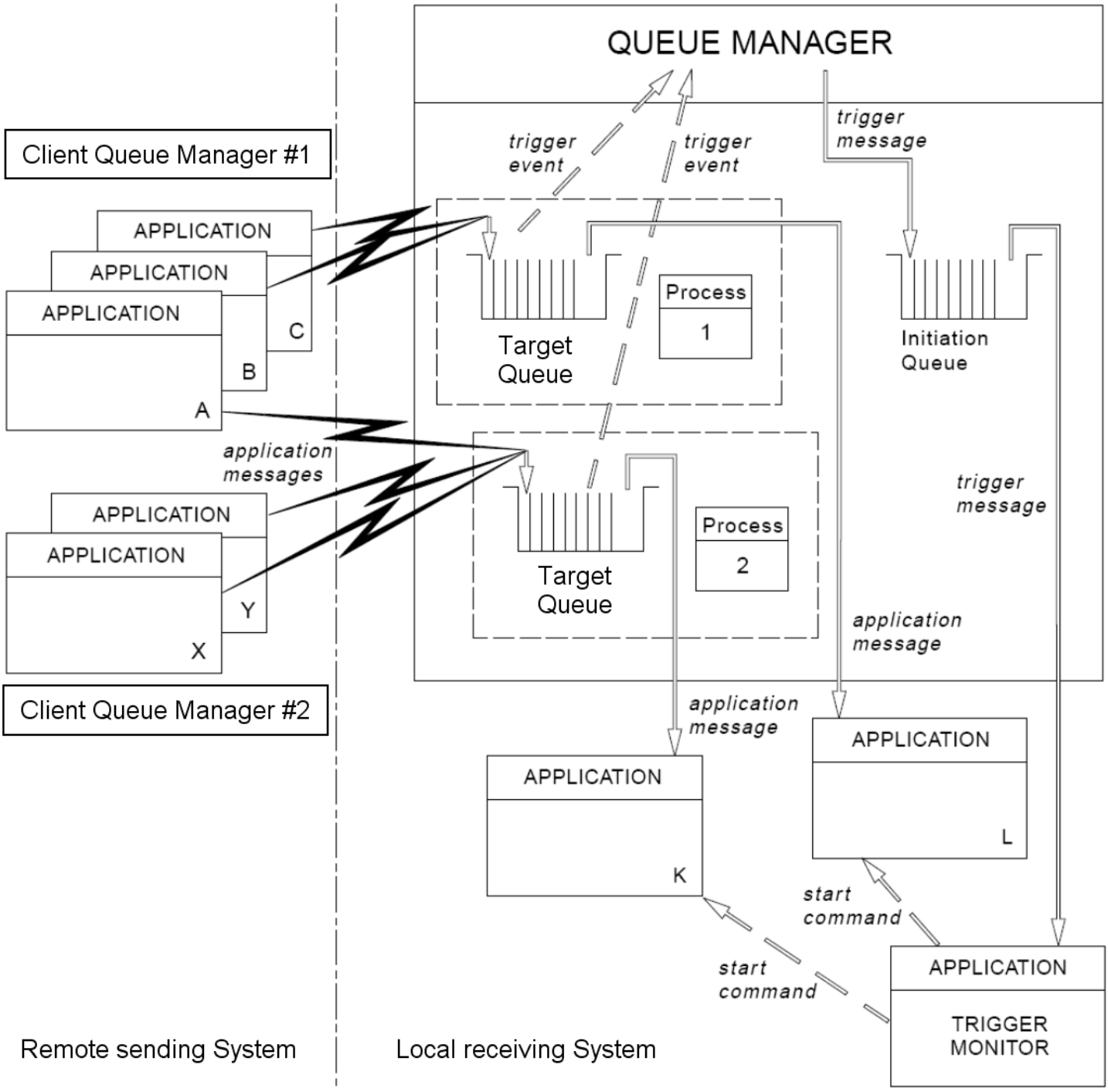


Abb. 10.3.7
Unterschiedliche Anwendungen kommunizieren mit unterschiedlichen Target Queues

Abb. 10.3.7 zeigt eine Situation, wo mehrere Programme, die entfernt zu dem empfangenden Queue-Manager sind, Nachrichten an zwei verschiedenen Target Queues senden. Mehrere Nachrichten können die gleiche Target Queue adressieren. Es ist angenommen, alle eingehenden Nachrichten zeigen Trigger-Events in ihren Deskriptoren an.

Obwohl mehrere Target Queues in Betrieb sind, und jede einem anderen Anwendungsprogramm zugeordnet ist, kann eine einzige Initiation Queue und ihr Trigger Monitor alle Triggering Events bedienen.

Abb. 10.3.7 zeigt zwei Server-Anwendungsprogramme K und L, die Nachrichten von verschiedenen Client-Anwendungsprogrammen A, B, C, X, Y empfangen können. Programme K und L werden von zwei verschiedenen Target Queues bedient.

Mit einer Target Queue ist ein Prozess-Definition Objekt assoziiert, welches Details über die Anwendung enthält, die die Nachricht verarbeiten soll. Der Queue Manager stellt die Informationen in eine Trigger Message, die in einer Initiation Queue gespeichert wird.

Der Trigger Monitor extrahiert diese Informationen aus der Trigger-Nachricht und startet die entsprechende Anwendung, um die Nachricht in jeder Target Queue zu behandeln. Nur eine Initiation Queue ist erforderlich, um entweder Anwendung K oder Anwendung L zu triggern.

10.4 MQI Programmierung

10.4.1 MQI Application Programming Interface

Die MQI API besteht aus 13 API Kommandos. Dies sind die 6 am häufigsten benutzten Kommandos:

MQCONN	**Verbindung mit einem (normalerweise entfernten) Queue Manager herstellen**
MQOPEN	**Öffnen (Open) einer spezifischen Queue**
MQPUT	**Eine Message in eine Queue schreiben**
MQGET	**Eine Message aus einer Queue auslesen**
MQCLOSE	**Eine Queue schließen (Close)**
MQDISC	**Verbindung mit einem Queue Manager auflösen**

Die übrigen 7 Kommandos sind:

MQPUT1	**Kombination von MQOPEN + MQPUT + MQCLOSE**
MQINQ	**Eigenschaften eines Objektes erfragen**
MQSET	**Eigenschaften (Properties) eines Objektes setzen**
MQCONNX	**Standard oder fast Path Bindings**
MQBEGIN	**Eine Unit of Work starten (database coordination)**
MQCMIT	**Commit eine Unit of Work**
MQBACK	**Back out**

Ein Client-Anwendungsprogramm benutzt verschiedene WebSphere MQ MQI Kommandos.

Es beginnt mit den MQCON und MQOPEN Kommandos. In der Regel beziehen diese sich auf einen entfernten QUEUE-Manager und eine entfernte Target-Queue. Bedenken Sie, dass der Remote Queue Manager mehrere Target Queues enthalten kann. Mit MQOPEN wählen wir die richtige Target Queue.

Das Anwendungsprogramm führt dann seinen Code aus, in der Regel durch den Einsatz mehrerer MQPUT und/oder MQGET Anrufe.

Wenn es damit fertig ist, beendet es seine Arbeit durch Ausgabe von MQCLOSE und MQDISC Kommandos.

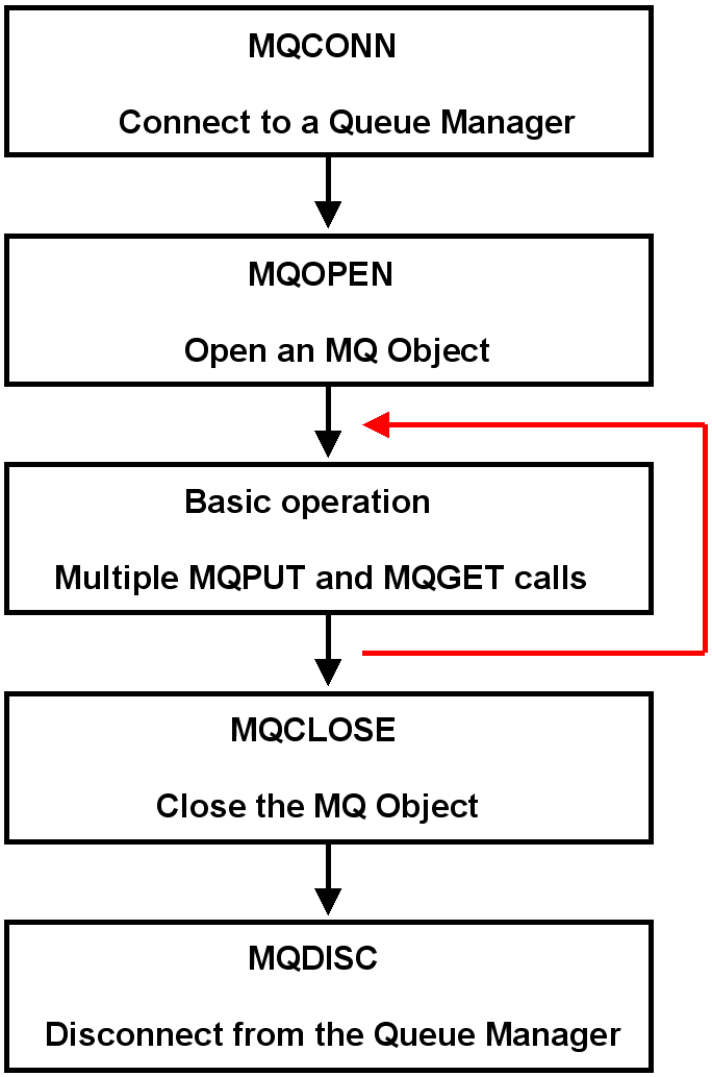


Abb. 10.4.1
Folge von MQI Kommandos in einem Client-Anwendungsprogramm

10.4.2 MQI Call Parameters

Es gibt zwei Typen von Parametern, die von allen Kommandos benutzt werden:

Handles

Diese werden von den Queue-Manager MQCONN und MQOPEN Kommandos zurückgegeben, und werden dann als Eingangsgrößen für die nachfolgenden MQPUT und MQGET Kommandos verwendet.

Der Begriff „Handle“ wird hier benutzt, um eine Message Channel Verbindung oder eine Target Queue eindeutig zu definieren.

Return codes

Zwei Return-Codes sind für alle Kommandos gebräuchlich: ein Completion-Code und ein Reason-Code.

Der Completion-Code gibt an, ob die Kommandoausführung erfolgreich war (mit einem MQCC_OK), oder ob ein Fehler aufgetreten ist (mit einem MQCC_FAILED).

Der Reason-Code ist MQCC_NONE, wenn der Completion-Code MQCC_OK ist. Wenn nicht, wird ein anderer Wert zurückgegeben, der die Ursache der Warnung oder des Fehler im Completion-Code erklärt.

10.4.3 Verbinden mit dem Queue Manager

Um eine MQSeries Aktivität mittels der MQI Schnittstelle zu starten, sollten Sie sich zunächst mit einem (in der Regel entfernten) QUEUE-Manager mit einem der beiden verfügbaren Verbindungs-Kommandos, MQCONN oder MQCONNX verbinden. Beispielsweise unter Verwendung von MQCONN:

```
MQCONN (QMgrName, Hconn, CompCode, Reason)
```

Als Input für MQCONN müssen wir den symbolischen Namen des (in der Regel entfernten) Queue-Managers (QMgrName) angeben.

Das Kommando gibt die folgenden Werte zurück:

- **Eine Connection Handle (`Hconn`) zu dem Queue-Manager. Hconn wird durch das Anwendungsprogramm in den folgenden Kommandos wendet, um diesen spezifischen Queue Manager zu adressieren.**
- **Die Ergebnis-Codes (Completion- und Reason-Codes).**

MQCONN bewirkt, dass der lokale Queue Manager eine Transmission Queue einrichtet, und ein Message Channel mit seinen beiden MCAs erstellt wird.

10.4.4 Öffnen von MQSeries Objekten

Das MQOPEN Kommando ermöglicht es einer Anwendung, Nachrichten in eine Queue zu schreiben oder Nachrichten aus einer Queue zu lesen.

```
MQOPEN (Hconn, ObjDesc, Options, Hobj, CompCode, Reason)
```

Das Kommando hat diese Input Parameter:

- **Eine Connection Handle. Der Wert von `Hconn` war von dem vorangehenden MQCONN Kommando zurückgegeben worden.**
- **Eine Beschreibung der Target Queue, die wir öffnen (Open) möchten. Dies geschieht in der Form eines MQ Object Descriptor (MQOD). MQOD ist eine Structure (in C++) oder ein Copybook (in Cobol). Diese Struktur identifiziert die Target Queue, die geöffnet werden soll.**
- **Eine oder mehrere Optionen. Eine mögliche Option ist es z.B., der Anwendung zu erlauben, eine Nachricht in die Target Queue zu stellen.**

Das Kommando gibt die folgenden Werte zurück:

- **Eine Object Handle Hobj, die den Zugriff auf die Target Queue mittels ihres Namens spezifiziert.**
- **Eine modifizierte Object-Descriptor Structure (MQOD, wenn eine Abänderung erforderlich ist).**
- **Die Result-Codes (Completion- und Reason-Codes).**

Weiterführende Information finden Sie in: MQ Application Programming Guide:
http://publib.boulder.ibm.com/iseries/v5r2/ic2924/books/csqzal05.pdf
Und in dem MQSeries Primer:
http://www.cedix.de/Literature/Textbooks/MQSerPrimer.pdf

10.4.5 Schreiben einer Nachricht in eine Queue

Wir benutzen das MQPUT Kommando, um eine Nachricht in eine Queue zu schreiben:

```
MQPUT (Hconn, Hobj, MsgDesc, PutMsgOpts, BufferLength, Buffer, CompCode,
Reason)
```

Dieses Kommando erhält als Input Parameter:

- **Eine Connection Handle Hconn, die durch das MQCONN Kommando zurückgegeben wurde.**
- **Eine Queue Handle Hobj , die von dem MQOPEN Kommando zurückgegeben wurde.**
- **Eine Beschreibung der Nachricht, die Sie in die Queue stellen wollen, in der Form eines Message Descriptors.**
- **Control Informationen, in Form einer Put-Message Optionen (MQPMO) Struktur.**
- **Die Länge der Daten in dieser Nachricht.**
- **Die Nachricht selbst, enthalten in einem Puffer.**

Das Kommando gibt diese Werte zurück:

- **Die Result-Codes (Completion- und- Reason-Codes).**
- **Aktualisierte Message Descriptor und Optionen, wenn der Aufruf erfolgreich ausgeführt wurde.**

10.4.6 Lesen einer Nachricht aus einer Queue

Wir benutzen das MQGET Kommando, um Nachrichten aus einer Queue zu lesen:

```
MQGET (Hconn, Hobj, MsgDesc, GetMsgOpts, BufferLength, Buffer, DataLength,
CompCode, Reason)
```

Die Eingabeparameter für diesen Aufruf sind:

- **Eine Connection Handle `Hconn`.**
- **Eine Queue Handle `Hobj`.**
- **Eine Beschreibung der Nachricht, die wir aus der Queue lesen wollen, in der Form einer MQMD Structure.**
- **Control Information in Form einer Get Message Options (MQGMO) Struktur.**
- **Die Größe des Puffers, in dem die Nachricht gespeichert werden soll.**
- **Die Adresse des Puffers.**

Die Ausgabe Parameter dieses Aufrufs sind:

- **Die Result-Codes (Completion- und Reason-Codes).**
- **Die Message in dem angegeben Puffer, wenn das Kommando erfolgreich abgeschlossen wurde.**
- **Die Get Message Options Struktur, modifiziert, um den Namen der Queue zu zeigen, aus dem die Nachricht abgerufen wurde.**
- **Die Message Descriptor Struktur, mit Informationen über die eingelesene Nachricht.**
- **Die tatsächliche Länge der Nachricht.**

10.4.7 Schließen des MQ Objektes

Um ein MQ Object zu schließen (Close) benutzen wir das MQCLOSE Kommando.

```
MQCLOSE (Hconn, Hobj, Options, CompCode, Reason)
```

Dieses Kommando benutzt die folgenden Eingabe Parameter:

- **Eine Connection Handle `Hconn`.**
- **Eine Queue Handle `Hobj` des Remote-Queue-Objekts das wir schließen wollen wir.**
- **Die Close Optionen.**

Die folgenden Parameter werden zurückgegeben:

- **Die Result-Codes (Completion und Reason Codes).**
- **Die Object Handle `Hobj`, auf den Wert MQHO_UNUSABLE_HOBJ zurückgesetzt.**

Typischerweise wird eine Queue gelöscht, sobald das Programm, das sie geschaffen hat, ein MQCLOSE Kommando für diese Queue ausführt. In diesem Fall wird die Close Option MQCO_NONE erzeugt.

10.4.8 Codefragment

Das folgende Codefragment zeigt die APIs, um eine Nachricht in eine Queue zu schreiben und eine Antwort von einem anderen Queue zu erhalten. Die C++ MQI wird benutzt.

Das Codefragment verwendet die folgenden Definitionen. Die Felder CompCode und Reason enthalten die Fertigstellung Codes für die APIs. Wenn Sie interessiert sind, können Sie sie in der Application Programming Reference Dokumentation finden:
http://publib.boulder.ibm.com/iseries/v5r2/ic2924/books/csqzak05.pdf

```
                                        // Definitions
MQHCONN HCon;                  // Connection handle
MQHOBJ HObj1;                  // Object handle for queue 1
MQHOBJ HObj2;                  // Object handle for queue 2
MQLONG CompCode, Reason;       // Return codes
MQLONG options;
MQOD od1 =      {MQOD_DEFAULT};  // Object descriptor for queue 1
MQOD od2 =  {MQOD_DEFAULT};  // Object descriptor for queue 2
MQMD md =   {MQMD_DEFAULT};  // Message descriptor
MQPMO pmo = {MQPMO_DEFAULT}; // Put message options
MQGMO gmo = {MQPMO_DEFAULT}; // Get message options

// 1 Connect application to a queue manager.
strcpy (QMName,"MYQMGR");
MQCONN (QMName, &HCon, &CompCode, &Reason);
// 2 Open a queue for output
strcpy (od1.ObjectName,"QUEUE1");
MQOPEN (HCon,&od1, MQOO_OUTPUT, &Hobj1, &CompCode, &Reason);
// 3 Put a message on the queue
MQPUT (HCon, Hobj1, &md, &pmo, 100, &buffer, &CompCode, &Reason);
// 4 Close the output queue
MQCLOSE (HCon, &Hobj1, MQCO_NONE, &CompCode, &Reason);
// 5 Open input queue
options = MQOO_INPUT_AS_Q_DEF;
strcpy (od2.ObjectName, "QUEUE2");
MQOPEN (HCon, &od2, options, &Hobj2, &CompCode, &Reason);
// 6 Get message
gmo.Options = MQGMO_NO_WAIT;
buflen = sizeof(buffer - 1);
memcpy (md.MsgId, MQMI_NONE, sizeof(md.MsgId);
memset (md.CorrelId, 0x00, sizeof(MQBYTE24));
MQGET (HCon, Hobj2, &md, &gmo, buflen, buffer, 100, &CompCode, &Reason);
// 7 Close the input queue
options = 0;
MQCLOSE (HCon, &Hobj2,options, &CompCode, &Reason);
// 8 Disconnect from queue manager
MQDISC (HCon, &CompCode, &Reason);
```

1 Dieses Statement verbindet die Anwendung mit dem dem QUEUE-Manager mit dem Namen MYQMGR. Wenn der Parameter QMName keinen Namen enthält, dann wird der Default-QUEUE-Manager verwendet. MQ speichert die Handle des Queue-Managers in der Variablen HCon. Diese Handle muss in allen nachfolgenden APIs genutzt werden.

2 Um eine Queue zu öffnen muss der Name der Queue in den Objektdeskriptor kopiert werden, der für diese Queue verwendet wird. Dieses Kommando öffnet QUEUE1 nur für Output (Open Option MQOO_OUTPUT). Die Handle der Queue und Werte in dem Objektdeskriptor werden zurückgegeben. Die Handle Hobj1 muss in dem MQPUT Kommando angegeben werden.

3 MQPUT platziert die Nachricht, die in einem Puffer steht, in eine Queue. Parameter für MQPUT sind:

- **Die Handle des Queue-Manager (aus MQCONN)**
- **Die Handle der Queue (von MQOPEN)**
- **Den Message Descriptor**
- **Eine Struktur, die Optionen für den MQPUT Befehl enthält (siehe MQ Application Programming Reference)**
- **Die Länge der Nachricht**
- **Der Puffer, der die Daten enthält**

4 Dieses Kommando schließt die Output Queue. Da die Queue vordefiniert ist, findet kein Close Processing statt (MQOC_NONE).

5 Dieses Kommando öffnet Queue2 nur für die Eingabe, unter Benutzung der Queue-Voreinstellungen (Defaults).

6 Für das Get Kommando wird die nowait Option verwendet. MQGET braucht die Länge des Puffers als Eingangs Parameter. Da keine Nachrichten-ID oder Korrelations-ID angegeben ist, wird die erste Nachricht aus der Queue gelesen. Sie können einen Warteintervall (in Millisekunden) hier spezifizieren. Sie können den Return Code überprüfen, um herauszufinden, ob die Zeit abgelaufen ist und keine Nachricht eingetroffen ist.

7 Dieses Kommando schließt die Eingabe-Queue.

8 Die Anwendung schließt die Verbindung mit dem Queue-Manager.

Die gezeigten Codebeispiele benutzen C++ und die MQI API. Sie würden sehr ähnlich mit anderen Programmiersprachen wie Cobol oder PL/1.aussehen

Java MQ-Code sieht anders aus, weil der JEE (Java Enterprise Edition) Standard für JMS (Java Message Service) eine eigene API festlegt, die sich von der MQI API unterscheidet. Java-Anwendungen verwenden normalerweise die JMS API anstelle der MQI API.

Es gibt 2 Versionen von WebSphere MQ. Die "non-integrated"-Version benötigt keinen WebSphere Java Application Server (WAS), und wird in der Regel für andere Programmiersprachen als Java verwendet. Die integrierte Version ist Teil der WebSphere Java Application Server (WAS), nutzt aber die gleiche Code-Basis wie die nicht-integrierte Version.

MQ Code Beispiele für Cobol, C + + und Java sind verfügbar unter;

http://www.cedix.de/Literature/Textbooks/MQ/index.html

Wenn Sie glauben, all dies ist kompliziert: Die meisten Unternehmen stehen vor dem Problem, bestehende Anwendungs-Software zu integrieren, die von unabhängiger Seite in der Vergangenheit entwickelt wurde, in verschiedenen Programmiersprachen implementiert wurde, und auf verschiedenen Hardware-, Betriebssystem-und Middleware-Platformen läuft. Viele Experten betrachten die Verwendung von WebSphere MQ als den einfachsten und effizientesten Weg, um die Integration von Anwendungen zu erreichen.

10.5 Weiterführende Information

Eine sehr gute Einführung
Dieter Wackerow: MQSeries Primer. www.redbooks.ibm.com/redpapers/pdfs/redp0021.pdf
Auch hier erhältlich:
http://www.cedix.de/Literature/Textbooks/MQSerPrimer.pdf

Ein interessantes Video
“WebSphere MQ Zero to Hello World in under 5 Minutes”
unter Benutzung von Linux ist zu finden unter
http://www.youtube.com/watch?v=wSCHLBftjDw

Vier vergleichsweise einfache WebSphere MQ Tutorials sind beschrieben unter
http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp?topic=%2Fcom.ibm.mq.explorer.tutorials.doc%2Fbi00112_.htm

Vielleicht interessiert Sie hier ein MQSeries Video vom IBM Vertrieb
"IBM WebSphere MQ on System z"
http://www.youtube.com/watch?v=rAJMXO2o59M

und WebSphere MQ im Allgemeinen
http://www.youtube.com/watch?v=kbxFDP6aWhw