19. Java Transaction Processing

19.1 EJB Transaktionseigenschaften

19.1.1 Locking Funktion für eine Transaktion

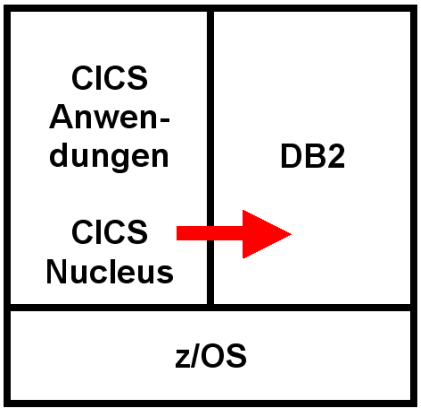

Abb. 19.1.1
Wird nur eine einzige Datenbank benutzt, übernimmt diese das Locking

In einer einfachen Konfiguration greift eine transaktionale Anwendung auf eine einzige Data Bank (zum Beispiel DB2) zu, die auf dem gleichen Rechner untergebracht ist. Mehrere unabhängige Anwendungen können auf diese Datenbank zur gleichen Zeit zugreifen. DB2 enthält hierfür eine eigene Transaktionsverarbeitungsfunktion, die z.B. auch bei Stored Procedures eingesetzt wird. Diese ordnet den einzelnen Transaktionen je eine Signatur zu, überwacht die transaktionalen Zugriffe und ist für den Transaktionsabschluss und die Fehlerbehandlung zuständig.

Bei der Benutzung von VSAM zur Datenspeicherung übernimmt der CICS File Manager die Locking Funktion.

19.1.2 Transaction Manager und Transaction Service

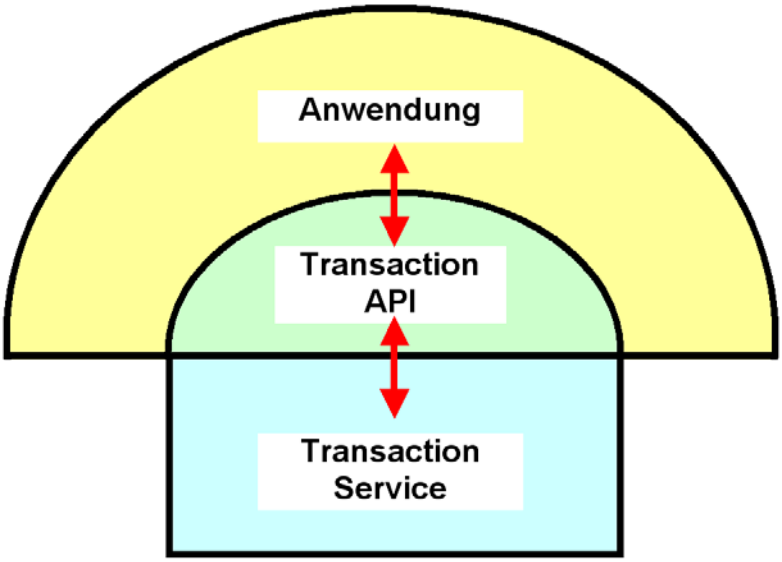

Abb. 19.1.2
Erläuterung der Begriffe

Formal unterscheidet man bei dem JEE Standard zwischen der Java Transaction API und einer Transaction Service Implementation. Die Java Transaction API ist Teil der JEE Spezifikation und ist die Schnittstelle zum Transaction Service.

Der Transaction Service implementiert einen Transaktionsmonitor vergleichbar zu CICS oder Tuxedo und ist die mit Abstand umfangreichste Komponente eines Web Application Servers. Der Transaction Service ist nicht Teil der JEE Spezifikation und ist eine proprietäre Implementierung des Herstellers des Web Application Servers. So verwendete früher der WebLogic Application Server den hauseigenen Tuxedo Transaktionsmonitor. Distributed WebSphere verwendete hier die distributed CICS Version.

19.1.3 Distributed Transaction System

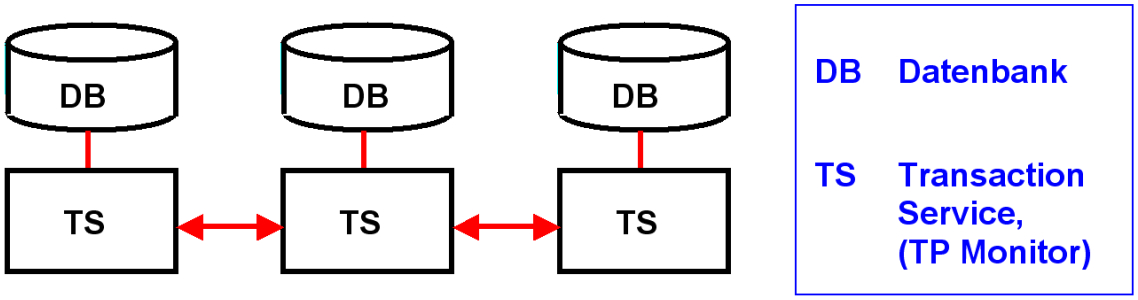

Abb. 19.1.3
Mehrere Transaktionsmanager implementieren eine globale Transaktion

In einem distributed Transaction System wird eine globale Transaktion auf mehreren Rechnern ausgeführt. Dabei können Datenbanken (DB) auf mehreren Rechnern mit ihrem eigenen lokalen Transaktionsmonitoren (hier als Transaction Service, TS, bezeichnet) an der Ausführung der Transaktion beteiligt sein. Eine globale Transaktion wird meistens mittels des 2-Phase-Commit Protokolls implementiert, siehe Band 1, Abschnitt 7.4.14.

Enterprise Java Beans und JEE Server unterstützen neben lokalen Transaktionen, bei denen nur auf eine lokale Datenbank zugegriffen wird, auch globale Transaktionen in verteilten Umgebungen mit mehreren Datenbanken.

19.1.4 Distributed Transaction Processing (DTP)

Wenn Information in einer Datenbank geändert wird, muss sichergestellt sein, dass die Änderung korrekt erfolgt. Nehmen wir an, Sie unterhalten bei Ihrer Bank zwei Konten, ein laufendes Konto (Checking Account) und ein Anlagekonto (Savings Account), und Sie wollen Geld von Ihrem Anlagekonto auf Ihr laufendes Konto überweisen. Wird die Transaktion nur teilweise ausgeführt, indem das Geld von Ihrem Anlagekonto abgebucht, aber nicht Ihrem laufenden Konto gutgeschrieben wird, wird Sie das nicht zufrieden stellen.

Weiteres Beispiel: Sie wollen einen Flug von San Francisco nach Paris buchen, aber ein direkter Flug ist nicht verfügbar. Wenn Sie keine Buchungsbestätigung sowohl für die Verbindung San Francisco – Chicago, als auch eine weitere Buchungsbestätigung Chicago – Paris erhalten, werden Sie diesen Flug nach Paris nicht buchen (commit). Sie werden ein "roll back" für diesen Flug nach Paris durchführen, weil eine Buchungsbestätigung nur für einen Teil des Fluges für Sie nicht annehmbar ist.

In beiden Beispielen sind mehrere kleinere Teiltransaktionen erforderlich um eine gesamte (globale) Transaktion durchzuführen. Wenn ein Problem mit einer Teiltransaktion besteht, wollen Sie ein Commit der gesamten Transaktion (Geldtransfer, oder diesen spezifischen Flug nach Paris buchen) nicht durchführen. Statt dessen wollen Sie ein Rollback für jede Teiltransaktion durchführen. Für eine erfolgreiche Geldüberweisung oder Parisreise-Buchung wollen Sie sicherstellen, dass alle Teiltransaktionen koordiniert und gesteuert werden um die globale Transaktion erfolgreich durchzuführen.

Für eine derartige koordinierte globale Transaktionsverarbeitung enthält die JEE Platform ein "distributed Transaction Processing Environment". Dieses besteht aus einem JEE Application Server, JEE Application Komponenten sowie einem JEE Connector Architecture (JCA) Resource Adapter. Hiermit können Resourcen über physische Rechnergrenzen hinweg transparent und koordiniert abgeändert werden.

In einer einfachen Konfiguration greift eine EJB Anwendung auf eine einzelne Data Bank (zum Beispiel DB2) zu. In einer komplexeren Situation greift eine EJB Transaktion auf mehrere "Resources" (Datenbanken, Files, Queues) zu, die von mehreren Transaction Processing Monitoren verwaltet werden, die evtl. wiederum auf getrennten Rechnern laufen und von unabhängigen Herstellern stammen. Eine derartige globale Transaktion erfordert ein "Distributed Transaction Processing" (DTP) Environment. Ein Beispiel ist die verteilte Flat Transaction und das 2-Phase Commit Protocol.

19.1.5 Two Phase Commit

Angenommen, ein CICS Programm will parallel eine Änderung einer CICS eigenen VSAM Datei und einer externen DB2 Datenbank durchführen. Das CICS Programm wird hierzu ein EXEC CICS Syncpoint Kommando ausführen. Dies löst eine 2-Phase Commit Operation aus. Siehe hierzu Band 1, Abschnitt 7.4.14.

Existierende Transaction Monitore wie CICS, IMS und DB2 verfügen hierzu über eigene Commit Coordinator Komponentenen. Das Update der VSAM Datei und der DB2 Datenbank wird erst entgültig, wenn der CICS Commit Coordinator Phase 2 des 2-Phase Commit Protokolls erfolgreich abgeschlossen hat.

Neu entwickelte Resource Manager Produkte unter z/OS (z.B. WebSphere Enterprise Java Beans) verwenden z/OS Resource Recovery Services (RRS) an Stelle einer nicht vorhandenen eigenen Two-phase Commit Protocol Komponente.

19.1.6 Resource Recovery Services

In der Vergangenheit haben z/OS Resource Manager wie IMS, CICS und DB2 ihre eigenen Commit Coordinator Funktion für die Two-Phase Commit Verarbeitung entwickelt. Hierbei übernimmt einer der beteiligten Resource Manager die Rolle des Commit Coordinators. Dies erforderte die Entwicklung von getrenntem Commit Coordinator Code jeweils für IMS, CICS und DB2.

Mit der wachsenden Anzahl verfügbarer z/OS Resource Manager entstand das Bedürfnis für einen generischen Commit Coordinator (von IBM als Sync Point Manager bezeichnet). Diese Rolle übernimmt eine neue z/OS Komponente, die Resource Recovery Services (RRS). RRS ist ein 2-Phase Commit Coordinator Service, der beliebigen z/OS Subsystemen zur Verfügung steht, so auch für WebSphere unter z/OS.

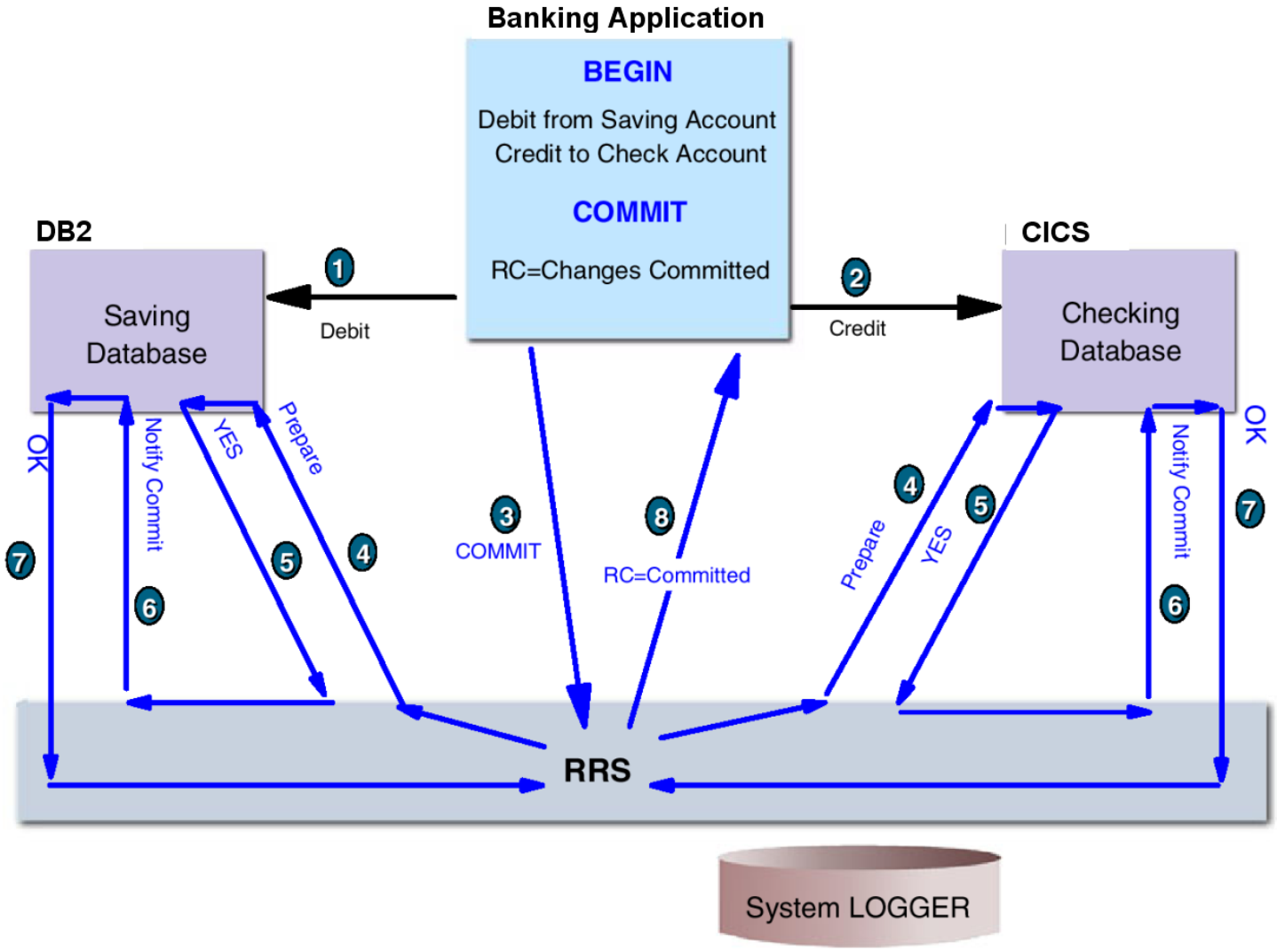

Abb. 19.1.4
RRS Funktionsablauf

Heute können Resource Manager wie IMS oder CICS die RRS Komponente an Stelle ihrer eigen Commit Coordinator Funktionen verwenden. Neue Entwicklungen wie WebSphere unter z/OS müssen RRS verwenden.

Das CICS Transaction Gateway kooperiert mit dem

- **Java Transaction Manager einer beliebigen JEE Platform (u.a. WebSphere), den**
- **Resource Recovery Services (RRS) unter z/OS, und**
- **CICS,**

um konsistente Änderungen für CICS und andere geschützte Resourcen durchzuführen.

Vom Standpunkt von CICS aus gesehen, agiert RRS als ein "external coordinator" für die Änderung oder Recovery von Resourcen. Vermutlich wird sich RRS in der Zukunft zu einer zentralen z/OS Komponente für die Transaktionssteuerung entwickeln.

Literature: http://www.redbooks.ibm.com/redbooks/pdfs/sg246980.pdf

19.1.7 X/Open

Das X/Open Konsortium wurde 1984 von mehreren UNIX Systems Herstellung mit der Zielsetzung gegründet, offene Standards auf dem Gebiet der Informationstechnologie zu etablieren. Spezifisch wurde eine einheitlich Unix Spezifikation für die Interoperability von Anwendungen und die Portierung von Software angestrebt.

Es existieren eine ganze Reihe von unterschiedlichen X/Open Standards. Sie gelten für alle Sprachen, einschließlich Java.

X/Open XA (kurz XA) ist ein von X/Open spezifizierter Informatik-Standard für Distributed Transaction Processing (die Abarbeitung von Transaktionen, die über mehrere Systeme verteilt sind). Mithilfe des XA-Standards können verschiedene „Ressource Manager“ (wie Datenbanken, Transaktionsmonitore, Applikationsserver, Messaging Systeme) innerhalb einer Transaktion angesprochen werden, unter Bewahrung der ACID-Eigenschaften von Transaktionen.

Die XA Spezifikation beschreibt, was ein Ressource Manager leisten muss um gemäß XA verteilte Transaktionen zu ermöglichen. Damit wird die Vorgangsweise und Schnittstelle zwischen einem globalen „Transaktion Manager“ und den lokalen Resource Managern festgelegt. Ein Transaktionsmanager ist somit etwas anderes als ein Transaktionsmonitor (Transaktionsserver). Ressource Manager, die den XA Standard erfüllen nennt man „XA compliant“. Der XA-Standard basiert auf dem 2-Phasen-Commit-Protokoll.

Die Java Implementierung des X/Open XA definiert die Schnittstelle (Interfaces und Exception-Classes), über die Java-Programme mit Transaktionsmanagern kommunizieren können. Transaktionsmanager ihrerseits implementieren üblicherweise die Java Transaction API (JTA) Programmierschnittstelle, Teil des EJB Standards. JTA stellt die Standardschnittstelle für XA fähige Transaktionsserver dar.

19.1.8 Globale Transaktion und der 2-Phase Commit Process

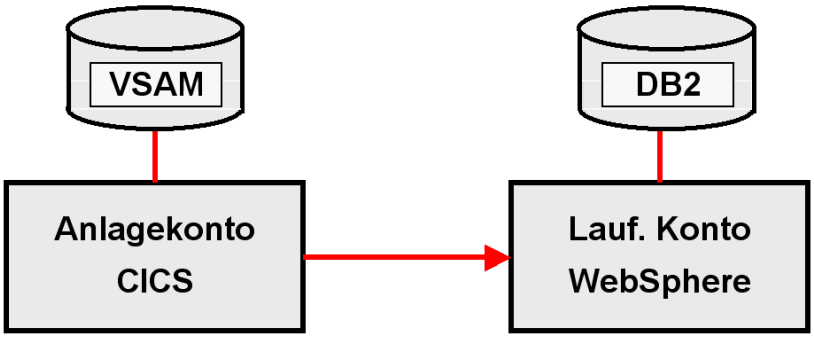

Abb. 19.1.5
Beispiel für eine globale Transaktion

Ein JEE-konformer Anwendungsserver (wie z.B. der WebSphere Application Server) benutzt den XA Transaction Manager für die Kommunikation zwischen den Anwendungs-Komponenten (z.B. Java Servlets oder Enterprise Java Beans) und externen Resource Managern (z.B. CICS oder DB2). Die Koordination einer globalen Transaktion erfolgt typischerweise über Java Connector Architecture (JCA) konforme Resource Adapter, wie z.B. dem CICS Transaction Gateway.

Wenn ein Transaction Manager (TM) eine globale Transaction mit mehr als einem Resource Manager koordiniert, verwendet die Commit Coordinator Funktion des Transaction Managers das Two-Phase Commit Protokoll.

In dem vorher erwähnten Bank-Beispiel unterhielten Sie bei Ihrer Bank zwei Konten, ein laufendes Konto (Checking Account) und ein Anlagekonto (Savings Account). Sie wollten Geld von Ihrem Anlagekonto (Savings Account) auf Ihr laufendes Konto (Checking Account) überweisen. Angenommen, Ihr Anlagekonto wird von einer CICS Anwendung bearbeitet, deren Daten in einem VSAM Data Set gespeichert sind, und Ihr laufendes Konto wir von einer z/OS WebSphere EJB Anwendung bearbeitet, deren Daten in einer DB2 Datenbank gespeichert sind. In diesem Fall würde die Commit Coordinator Funktion des WebSphere Transaction Managers die Änderungen zwischen VSAM und DB2 koordinieren.

19.1.9 Transaktionale Konnektoren

Konnektoren sind spezielle EJBs, welche eine Verbindung zu einem Backend System wie CICS oder DB2 herstellen. WebSphere für z/OS unterstützt zwei Arten von Konnektoren: non-transactional und transaktional. Die Connector Verarbeitung variiert je nach Konfiguration für jede installierte Instanz eines Connectors.

Die Transaktion Policy der EJB bestimmt, ob die Verarbeitung im Rahmen einer globalen Transaktion ausgeführt wird, oder nicht. Wenn ja, ist der Connector für die Steuerung der globalen Transaktionsverarbeitung zuständig. Für transaktionale Connectoren wird die Art der Transaktionsverarbeitung zu dem Zeitpunkt festgelegt, wenn eine Anforderung an das Ziel Enterprise Information System (EIS) gesendet wird.

z/OS WAS ist in der Lage, in Kooperation mit z.B. CICS eine distributed Transaction durchzuführen, die das 2-Phase Commit Protokoll erfordert. Diese benutzt den transaktionalen Typ von Connector. Letzterer ist konfiguriert, mit der Ressource Recovery-Service (RRS) Komponente von z/OS eine Two-Phase Commit Verarbeitung durchzuführen. RRS beinhaltet die Commit Coordinator Funktion für die 2-Phase Commit Verarbeitung; z/OS WAS beinhaltet keine eigene Commit Coordinator Funktion.

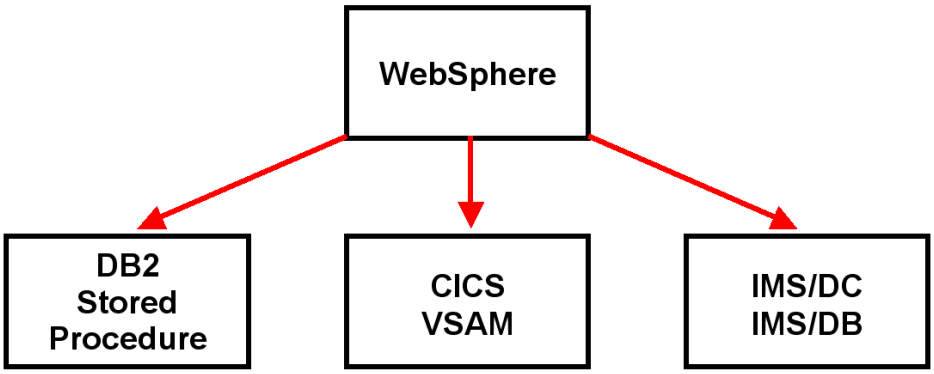

Abb. 19.1.6
WebSphere als Commit Koordinator

Beispiel: Ein WebSphere Anwendungsprogramm führt eine globale Transaktion durch, an der DB2, CICS und IMS/DC als Resource Manager beteiligt sind.

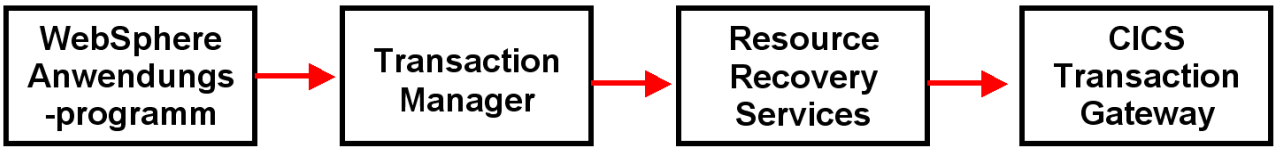

Abb. 19.1.7
Ablauf der Commit Koordination

Dies sind die Merkmale:

- **WebSphere übernimmt die Rolle des Transaktionsmanagers**
- **Die 2-Pase Commit Funktion wird durch Resource Recovery Services implementiert**
- **Das CICS Transaction Gateway ist ein JCA konformer Konnektor, dessen Konfiguration für eine globale Transaktion enabled wurde.**
- **Die JCA Konnektoren für DB2 und IMS/DC wurden ebenso enabled.**

19.1.10 X/Open Distributed Transaction Processing Model

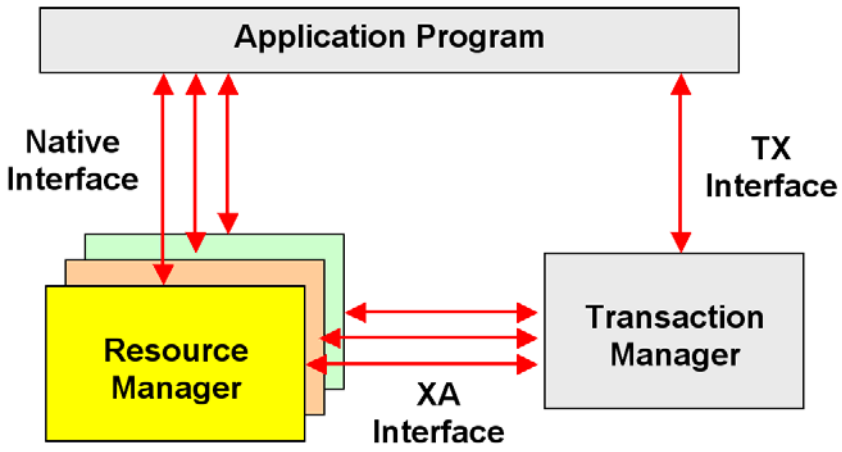

Abb. 19.1.8
X/Open Interfaces

Vorsicht: Als Transaktion Manager wird manchmal ein Transaktionsserver (Transaktionsmonitor) wie CICS bezeichnet. X/Open DTP bezeichnet statt dessen als Transaktion Manager eine Funktion, die einen 2-Phase Commit Coordinator enthält. Das X/Open Distributed Transaction Processing (DTP) Modell ist eine Spezifikation für das Management von Transaktionen, deren Operationen auf unterschiedlichen Rechnern oder auf Datenbanken unterschiedlicher Hersteller (z.B. Oracle, DB2) erfolgen.

DTP besteht aus 3 Kernkomponenten:

- **Einem Application Program. Dieses definiert Transaktionsgrenzen und spezifiziert die Aktionen, die eine Transaktion darstellen.**
- **Resource Managers (RM) sind Subsysteme oder Komponenten wie CICS, IMS, oder DB2. Diese verwalten Resourcen, die von den Transaktionen in Anspruch genommen werden, wie z.B. Datenbanken, Files oder Queues. Alternativ kann ein RM auch ein Data Access Komponente wie ODBC oder JDBC sein.**
- **Ein Transaction Manager (TM). Dieser markiert Transaktionen mit Idendifiern, überwacht den Fortschritt und ist zuständig für den erfolgreichen Abschluss (commit), oder alternativ für Fault Recovery (Rollback). Ein TM beinhaltet vor allem einen 2-Phase Commit Coordinator.**

RRS (Resource Recovery Services, Abschnitt 11.ist ein Beispiel einer XA Transaction Manager Implementierung.

19.1.11 X/Open Interfaces

Der X/Open Standard ist für beliebige Sprachen anwendbar, einschließlich Java.

Das Application Program, der Resource Manager (RM) und der Transaction Manager (TM) kommunizieren über drei spezifische Interfaces: Native, TX, and XA.

- **Über die Native interface sendet das Application Program Requests direkt an die Resource Managers. Diese Schnittstelle ist Resource Manager spezifisch; embedded SQL oder EXEC CICS sind Beispiele.**

- **Die TX Interface ist das Medium zwischen dem Application Program und dem Transaction Manager, zum Beispiel der Commit Coordinator Komponente von CICS. Die API verwendet TX Calls (über dieTX Interface), um den Transaction Manager zum Start, Commit oder Roll Back einer globalen Transaktion aufzufordern. Diese Schnittstelle ist Transaction Manager spezifisch.**

- **Die XA Interface ist das Medium zwischen den Resource Managern und dem Transaction Manager. Mit Hilfe von XA Calls fordert der Transaction Manager den Resource Manager zum Transaction Start, Commit, oder Roll Back auf. Der Transaction Manager ist auch für eine Recovery zuständig.**

Die X/Open XA Interface ist ein offener Standard um Änderungen für multiple Resources zu koordinieren, wobei die Integrität dieser Änderungen gewährleistet wird. Beispiele sind Resourcen, die eine persistente Speicherung beinhalten wie Datenbanken, Queues und File Systeme. Mehrere unterschiedliche Transaction Processing Monitore benutzen typischerweise die XA Interface für eine gegenseitige Kommunikation. Ein gleichzeitiger Zugriff auf mehrere Datenbanken ist damit möglich.

19.2 Enclaves

19.2.1 Language Environment

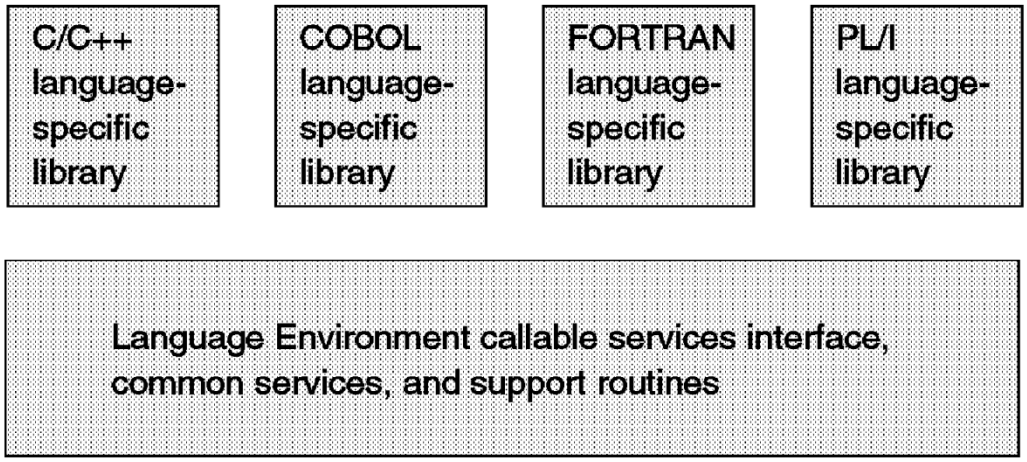

Abb. 19.2.1
LE Übersicht

Das z/OS „Language Environment“ (LE) Programm Management Modell bietet einen Rahmen, in dem der Code von Anwendungsprogrammen ausgeführt werden kann, die in verschiedenen Sprachen geschrieben wurden.

LE bietet eine gemeinsame Laufzeitumgebung für die z/OS Versionen bestimmter Hochsprachen (High Level Language) an, nämlich C/C++, COBOL, Fortran, PLI und Java. LE kann als eine Sammlung von Ressourcen, Bibliotheken und Betriebssystem-spezifischen Diensten und Schnittstellen betrachtet werden.

LE kombiniert häufig verwendete Laufzeit-Dienste, wie Routinen für

- **Mathematische Funktionen (z.B. transzendentale Funktionen wie Wurzel, arctan, x^{π}, ...)**
- **Laufzeit Message-Handling,**
- **Condition Handling,**
- **Storage-Management, und**
- **Datum und Uhrzeit.**

Das bedeutet z.B., dass es nur eine arctan Object Code Routine gibt, die von unterschiedlichen Programmiersprachen genutzt wird. LE stellt dafür eine Reihe von Schnittstellen zur Verfügung, die konsistent für alle unterstützten Programmiersprachen sind.

Language Environment besteht aus:

- **Basic-Routinen, die den Start und Stop von Programmen, Zuweisung von Speicherkapazitäten, sowie Handhabungsbedingungen unterstützen.**
- **Gemeinsame bibliothekarischen Dienstleistungen, wie z. B. mathematische Funktionen sowie Datum und Uhrzeit Dienste.**
- **Sprach-spezifische Teilen der Laufzeitbibliothek, da viele Sprach-spezifische Aufrufe von Language Environment Routinen unterstützt werden müssen. Dabei ist das Verhalten konsistent für alle unterstützten Sprachen.**
- **Einrichtungen für die Kommunikation zwischen Programmen, die in verschiedenen Sprachen geschrieben sind.**

POSIX Unterstützung ist in der Language Environment Grundausstattung sowie in der C/C++ spezifischen Library vorhanden.

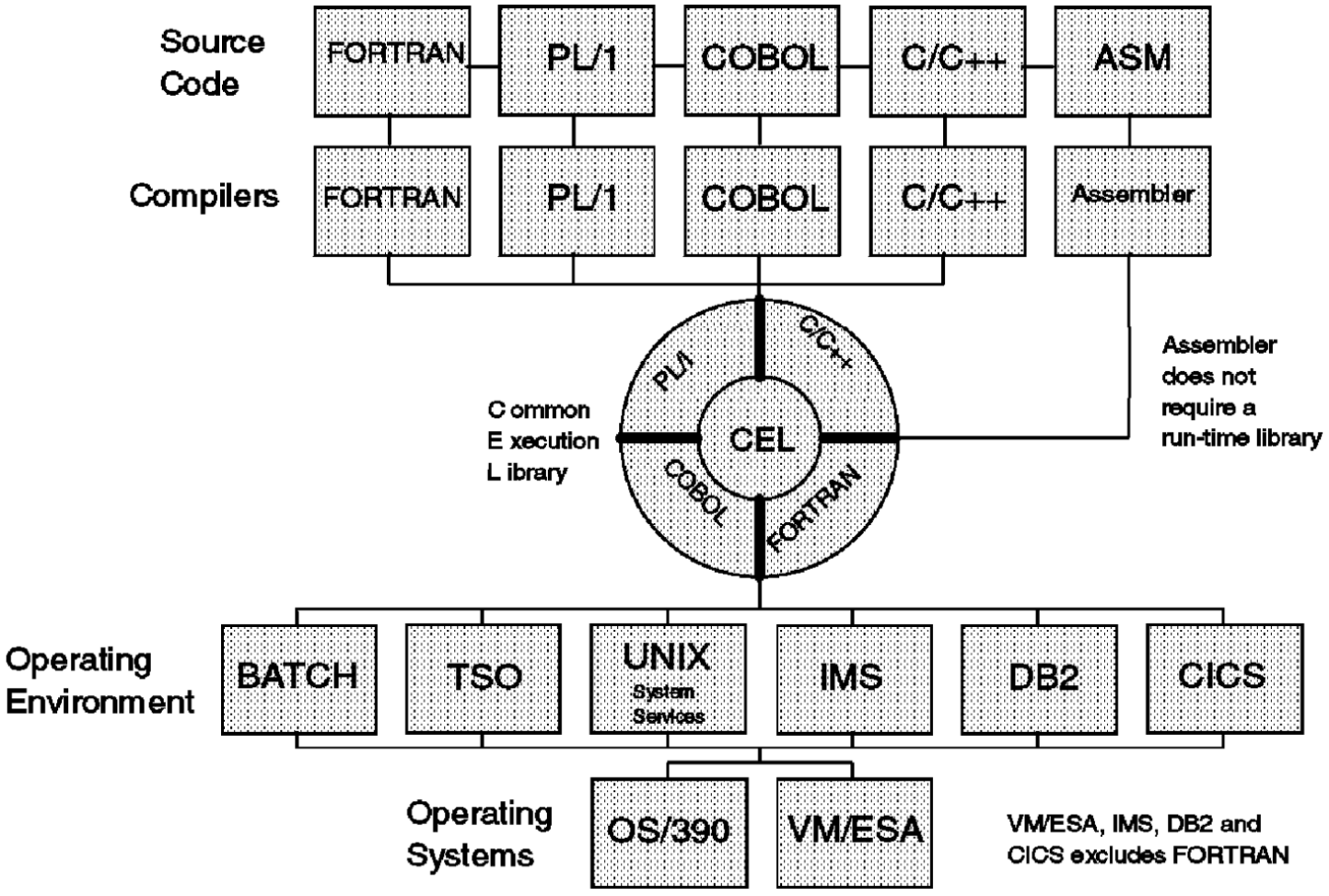

Abb. 19.2.2
Language Environment unter z/OS

Jede High Level Language hat ihre spezifischen Laufzeit und SYSLIB Bibliotheken. Zusätzlich benutzt sie zusammen mit anderen Sprachen eine gemeinsame Common Execution Library (CEL).

Die auf diese Weise hergestellten Load Modules können in verschiedenen Ausführungsumgebungen (Batch, CICS, DB2 usw.) unter z/OS oder z/VM ausgeführt werden.

19.2.2 Übersetzung eines neu entwickelten Programms

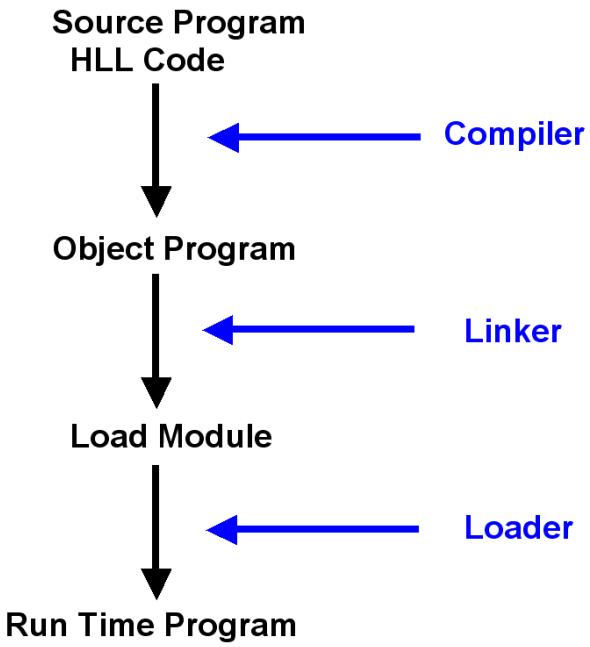

Abb. 19.2.3
Übersetzung eines Programms

Der Compiler übersetzt ein Source Programm (in Cobol, Java, C++, ...) in ein Object Programm (Binaries). Das Object Programm besteht aus Maschinenbefehlen.

Der Linker verkettet das Objekt Programm, gemeinsam mit einem oder mehreren zu einem anderen Zeitpunkt entstandenen Objekt Programm(en), sowie mit Routinen aus einer Programmbibliothek und aus der Common Execution Library zu einem ausführbaren Gesamtprogramm, dem Load Module. Die Common Execution Library besteht aus Object Modulen.

Alle Komponenten des Load Modules haben eindeutige Adressen; der zusammenhängende Adressenbereich beginnt typischerweise mit der Adresse 000...000 .

Der Loader lädt das Load Module vom Plattenspeicher in den Hauptspeicher, typischerweise auf eine Adresse, die nicht mit 000...000 beginnt. Hierzu werden alle Adressen durch den Loader verschoben (relocated).

19.2.3 Language Environment Definition

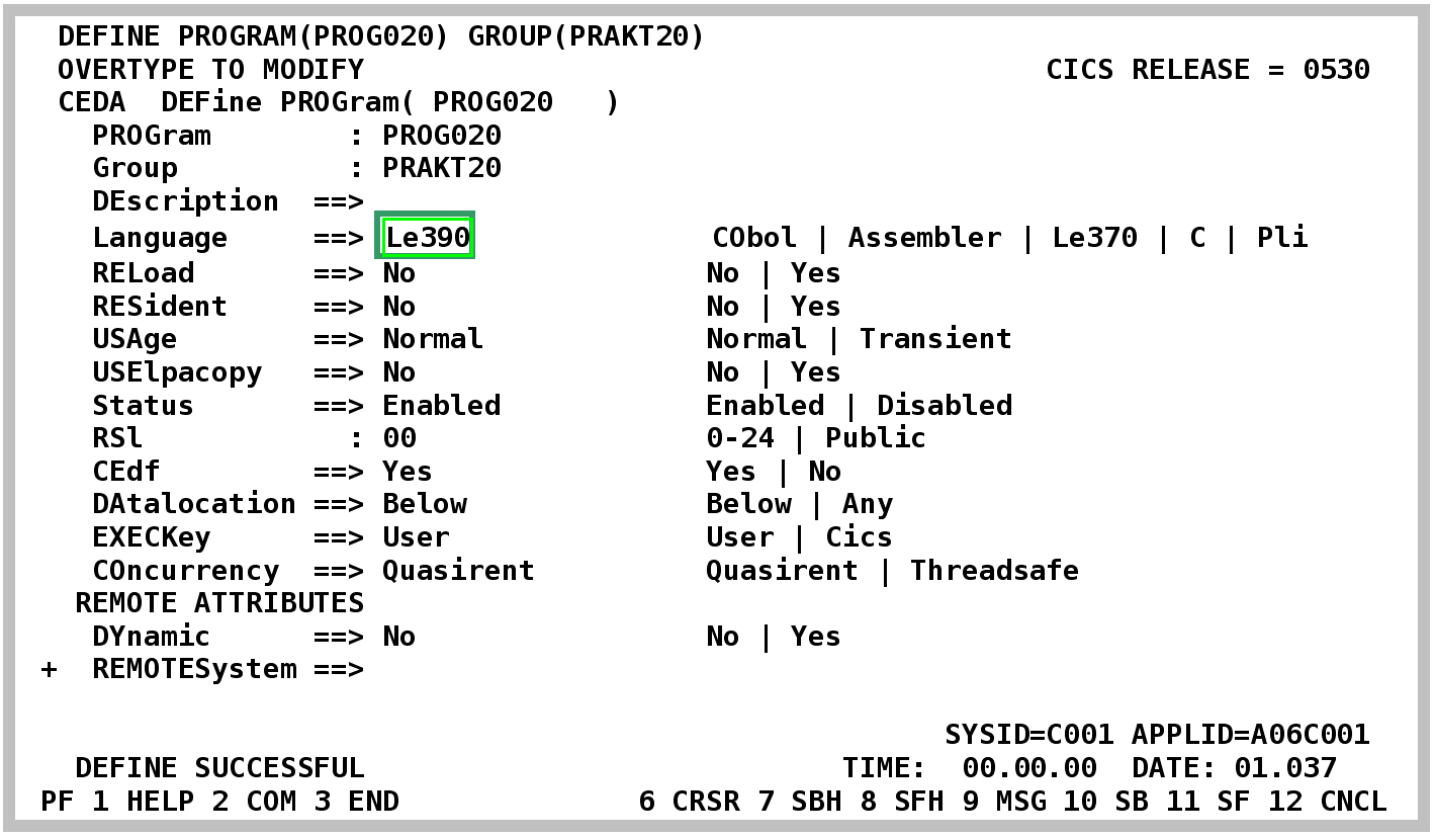

Abb. 19.2.4
LE Definition für ein CICS Programm

Bei der Bearbeitung einer CICS Übungsaufgabe sieht man den in Abb. 19.2.4 dargestellten Bildschirm. Der Benutzer wird gebeten, eine Sprache für das CICS-Programm angeben. Die richtige Antwort ist Le390 (oder Le370). Le390 ist jedoch keine Sprache, es spezifiziert das z/OS Language Environment als die Ausführungsumgebung für eine CICS-Anwendung.

19.2.4 Parallele Verarbeitung von Transaktionen

Ein Hochleistungs-Transaktionssystem verarbeitet in jedem Augenblick gleichzeitig Hunderte oder Tausende von Transaktionen. Für die Verarbeitung gibt es zwei Alternativen:

- **1 Prozess pro Transaktion**
- **1 Thread pro Transaktion)**

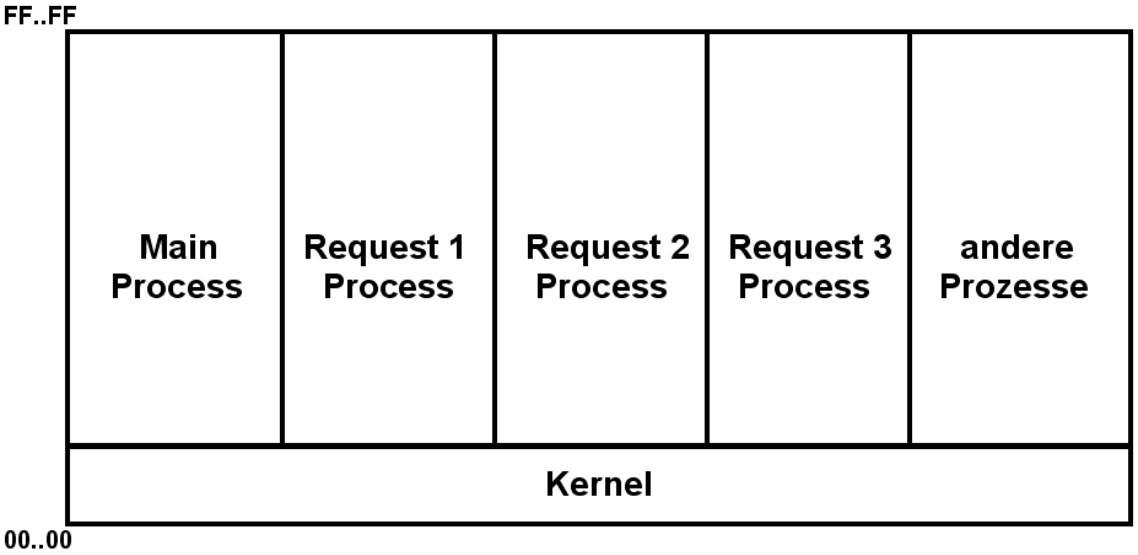

Abb. 19.2.5
Ansatz mit mehreren Prozessen

1 Prozess pro Transaktion bedeutet, dass jede Transaktion in einem eigenen virtuellen Adressenraum (z/OS Region) läuft. Vorteilhaft ist, dass die Isolation der Transaktionen untereinander (das i in ACID) optimal gewährleistet ist. Nachteilig ist der höhere Verarbeitungsaufwand im Vergleich zum Thread Ansatz.

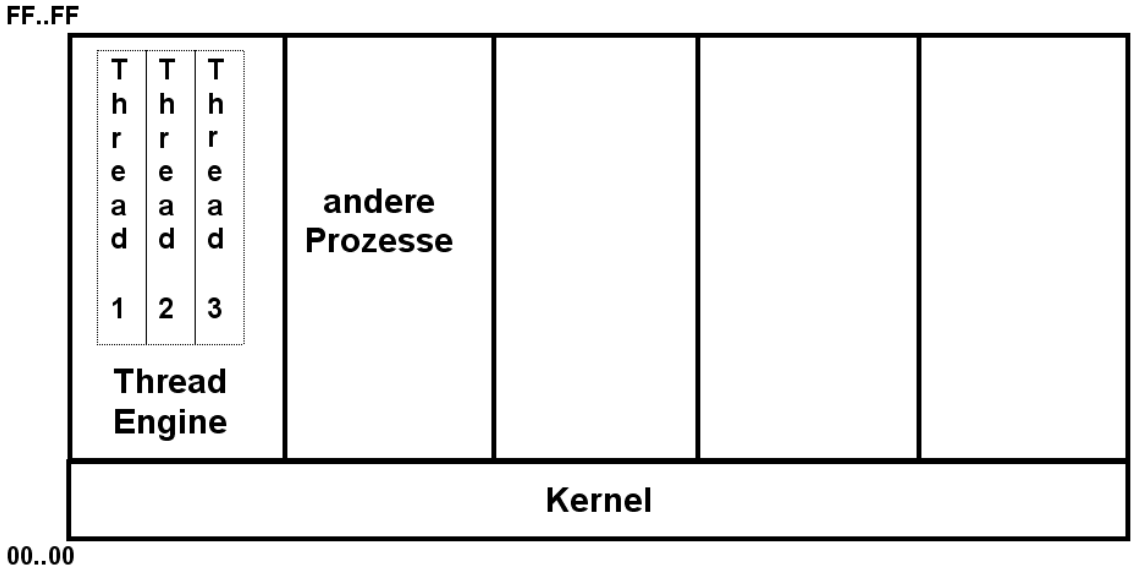

Abb. 19.2.6
Thread Ansatz

Threads haben den Nachteil, dass die Isolation der Threads gegeneinander gewährleistet werden muss.

Bei der z/OS Version von CICS laufen alle Anwendungen unter einem Thread ähnlichen Mechanismus in einem einzigen Adressenraum.

19.2.5 CICS Enclaves

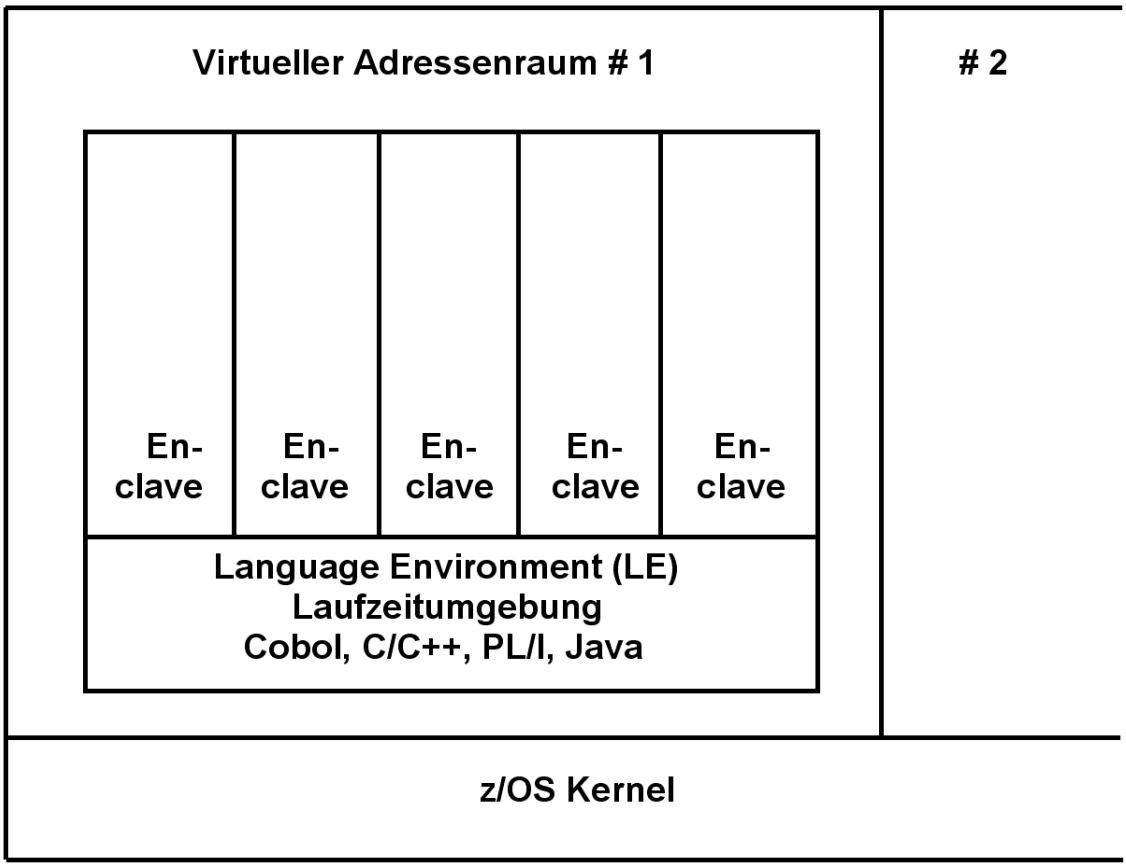

Abb. 19.2.7
Enclaves in der CICS Region

Die grundlegende CICS Konfiguration besteht aus einer einzigen z/OS Region, die den CICS Nucleus beinhaltet, sowie bis zu mehr als 1 000 gleichzeitig ausgeführten Transaktionen beherbergt.

Transaktionen müssen ACID Anforderungen erfüllen: Atomicity, Consistency, Isolation, Durability.

Die Isolation ist oft schwierig zu implementieren. CICS verwendet etwas ähnliches wie Threads, den „Enclave“ Mechanismus.

Enclaves sind ein Teil des z/OS Language Environment (LE). Enclaves sind Thread-ähnliche Einheiten. Sie sind Laufzeit-Einheiten, die das Äquivalent einer multithreaded Verarbeitung durch den CICS Nucleus ermöglichen. Enclaves stellen sicher, dass parallel ausgeführte Transaktionen innerhalb eines einzigen z/OS Adressraums (Region) voneinander isoliert sind.

Language Environment Enclaves können eingesetzt werden, um unterschiedliche Anwendungen innerhalb des gleichen virtuellen Adressenraums voneinander zu isolieren. Sie werden u.a. in transaktionalen Subsystemen wie CICS, IMS und DB2 benutzt. Zahlreiche Enclaves laufen in einem einzigen CICS Virtuellen Adressenraum. CICS Anwendungen (Transaktionen) werden in getrennten Enclaves ausgeführt.

Im Falle von CICS ist die LE Laufzeitumgebung Bestandteil des CICS Nucleus.

Enclaves benutzen hierfür einen Speicherschutzschlüssel (Storage Protection Key). Der Hardware Speicherschutz arbeitet unabhängig und zusätzlich zu dem üblichen Speicherschutz über Segment- und Seitentafeln und ist eine Einrichtung der z/Series Hardware Architektur, die auf anderen Plattformen nicht verfügbar ist. Der CICS Nucleus verwendet typischerweise Speicherschutzschlüssel Nr. 8, die Anwendungen in ihren Enclaves Speicherschutzschlüssel Nr. 9. Der CICS Nucleus ist somit vor falschen Zugriffen durch fehlerhafte Anwendungen in den Enclaves geschützt. Siehe Band 1, Abschnitt 1.3.10.

Im Falle von Java läuft in jeder Enclave eine JVM, und in dieser eine Java Transaktion unter einem eigenen Open TCB (JTCB). Spezifisch ist es hiermit möglich, mehrere JVMs innerhalb eines Adressraums laufen zu lassen, siehe Band 1, Abschnitt 9.3.3.

19.2.6 LE Program Management

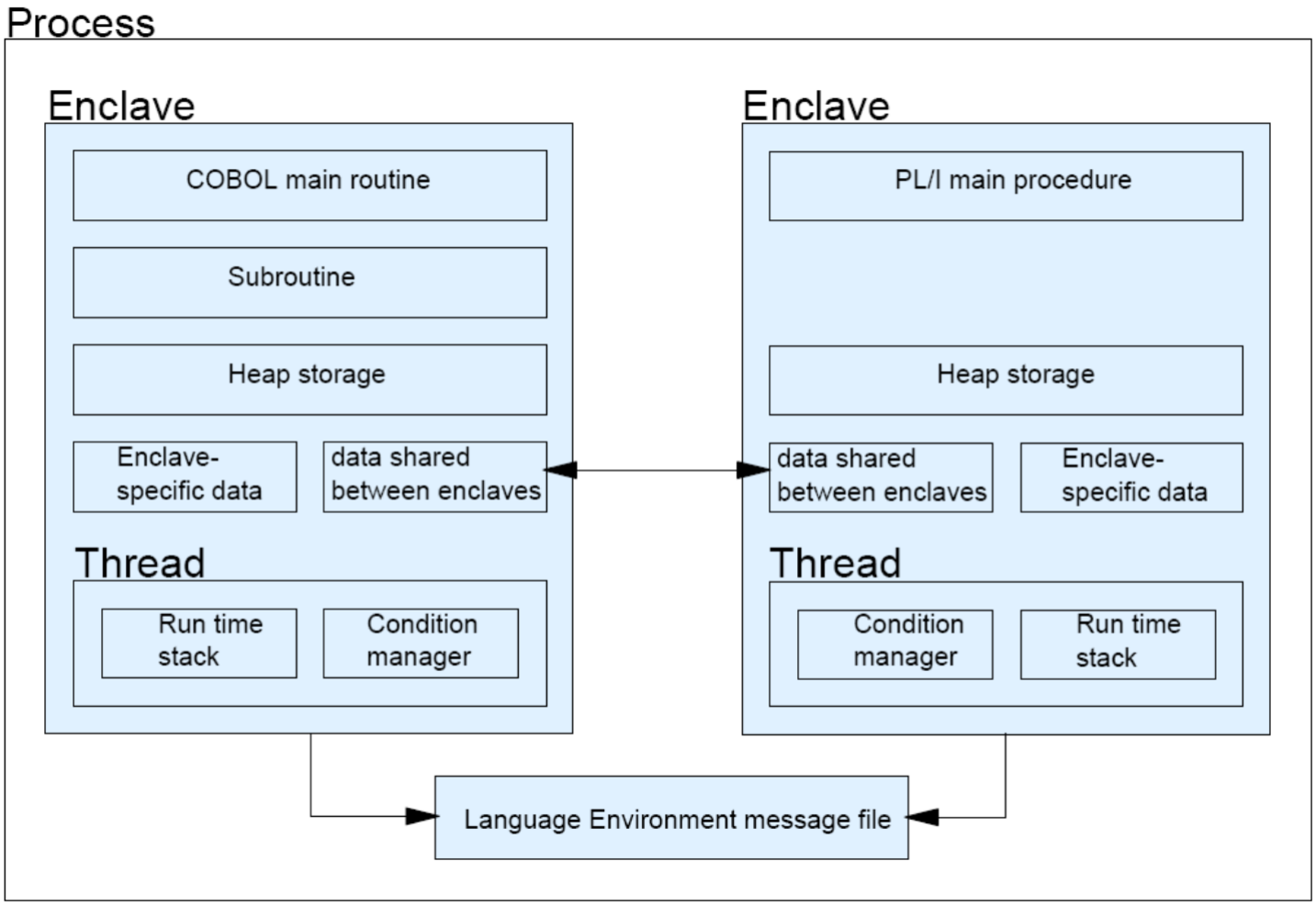

Abb. 19.2.8
Struktur der Enclave

Drei Elemente:

- **Prozess,**
- **Enclave und**
- **Thread**

sind der Kern des Language Environment Programm-Management-Modells.

Ein Prozess kann mehrere Enclaves enthalten. Jede Enclave enthält ihren eigenen Programm-Code. Der Code kann in verschiedenen Sprachen geschrieben werden, darunter auch Cobol und Java.

Die Threads innerhalb der gleichen Enclave benutzen gemeinsam den gleiche Code, und ebenso den Heap-Storage. Jeder Thread hat seinen eigenen Stack.

Die höchste Ebene des Language Environment Programm-Modells ist der Prozess. Jeder Prozess hat seinen eigenen Adressenraum. Ein Prozess ist eine Sammlung von Ressourcen, sowohl Programmcode als auch Daten.

Ein Prozess besteht in der Regel aus einer Enclave und ist logisch getrennt von anderen Prozessen. Jeder Prozess hat seinen eigenen Speicherplatz; Prozesse haben keinen gemeinsamen Speicherplatz. Prozesse sind gleichberechtigt und unabhängig voneinander. Es besteht keine hierarchische Gliederung.

Prozesse können neue Prozesse erstellen. Sie kommunizieren miteinander mit Hilfe einer Language Environment definierten Kommunikation, z. B. um anzuzeigen, dass ein Prozess beendet wurde.

Die Enclave ist eine Sammlung von Routinen, die eine Anwendung darstellen. Eine Enclave ist das Äquivalent von einer der folgenden Bezeichnungen:

- **Eine Run Unit in COBOL**
- **Ein Programm, (Main C-Funktion und seine Sub-Funktionen), in C/C++**
- **Einer Main Procedure und alle seine Unterprogramme in PLI**
- **Einer JVM und ihre Klassen in Java**

Die Enclave besteht aus einem Hauptprogramm und einer beliebige Anzahl von Unterprogrammen. Die Main Routine ist als erstes in einer Enclave auszuführen; alle nachfolgenden Routinen werden als Unterprogramme behandelt.

Externe Daten sind nur innerhalb der Enclave, in der sie sich befinden, verfügbar. Der Geltungsbereich der externen Daten ist die Enclave. Auch wenn externen Daten identisch Namen in verschiedenen Enclaves aufweisen, sind sie nur innerhalb ihrer Enclave gültig.

Die Threads in einer Enclave können weitere Enclaves erstellen, diese können mehr Threads erstellen, usw.

Für Sonderfälle existiert ein Mechanismus, mit dem Enclaves Daten gemeinsam nutzen können. Ein Beispiel ist die PLI Standard SYSPRINT File. Sie ist shared von mehreren Enclaves innerhalb einer Anwendung.

Es existiert eine Language Environment Message-File, die von mehreren Enclaves gemeinsam genutzt werden kann. Dies ist ein nützlicher zentraler Speicherort für Nachrichten, die während der Ausführung von Programmen innerhalb der Enclaves generiert werden können.

Einer der wichtigen Eigenschaften von CICS Enclave Transaktionen ist, dass der z/OS Work Load Manager sie unabhängig voneinander verwalten kann, auch wenn mehrere Enclaves in dem gleichen Adressenraum laufen.

Den einzelnen aktiven Enclaves in einem Adressraum können ihre eigenen unabhängigen Dispatching und I/O-Prioritäten zugeordnet werden, abhängig von den WLM Goals, die der Benutzer definiert hat. Wichtige Enclave Transaktionen können ihre WLM Goals nicht verpassen, weil weniger wichtige Transaktionen im gleichen Adressraum ausgeführt werden.

Jedes Enclave besteht aus mindestens einem Thread.

Ein Thread ist ein Ausführungs-Konstrukt, das aus synchronen Aufrufen und Terminierungen von Routinen besteht. Ein Thread wird von dem System mit seinem eigenen Runtime-Stack, Befehlszähler und Registern dispatched. Threads können gleichzeitig mit anderen Threads innerhalb einer Enclave existieren.

Der Runtime Stack steuert die Ausführung eines Threads. Er enthält außerdem den Befehlszähler, Register und Condition-Handling Mechanismen. Jeder Thread stellt eine unabhängige Instanz einer Routine dar, die in einer Enclave unter unter Benutzung der Enclave Ressourcen läuft.

Threads benutzen gemeinsam alle Ressourcen einer Enclave. Ein Thread kann den ganzen Speicher innerhalb einer Enclave adressieren. Alle Threads sind gleichberechtigt und unabhängig voneinander und nicht hierarchisch miteinander verbunden.

Weil Threads mit ihrem eigenen Run-Time Stacks arbeiten, können sie gleichzeitig innerhalb einer Enclave laufen und ihren Speicherplatz verwalten. Weil sie gleichzeitig ausgeführt werden können, können Threads verwendet werden, um eine parallele Verarbeitung von Anwendungen (und von ereignisgesteuerten Anwendungen) zu implementieren.

In einer z/OS Enclave kann jeder Thread mit unabhängigen eigenen Performance Zielen ausgeführt werden. Mit z/OS Workload-Management (WLM) kann man z.B. z/OS Performance-Ziele für einzelne DB2-Server-Threads etablieren.

Threads innerhalb einer Enclave erfordern eine kritische Handhabung um zu gewährleisten, dass sie sich nicht gegenseitig beeinflussen. Die Isolation muss gewährleistet sein. Deshalb benutzen CICS-Anwendungen in der Regel keine Threads um mehrere Transaktionen in einem einzigen Enclave laufen zu lassen.

Allerdings verwendet der neue CICS „JVM Server“ Threads! Das OSGi Framework innerhalb jeder JVM stellt die erforderliche Isolation (teilweise) sicher.

19.3 CICS und Java Standard Edition

19.3.1 Task Control Block

Prozesse (und Threads) unter z/OS werden von einem Task Control Block (TCB) gesteuert. Unix benutzt die Bezeichnung Process Control Block (PCB).

Ein TCB ist eine Datenstruktur in dem z/OS-Kernel Adressraum, der die erforderliche Informationen enthält, um einen bestimmten Prozess zu verwalten. Der TCB ist die Manifestation eines Prozesses in z/OS.

Ein TCB umfasst:

- **Den Identifier des Prozesses (Prozessidentifier, oder PID).**
- **Register Inhalte für den Prozess (Mehrzweck, Gleitkomma, Control, Access Register) einschließlich des Programm-Status Wortes (PSW) für den Prozess.**
- **Identifizierung des Adressraums für den Prozess.**
- **Priorität: Ein Prozess mit höherer Priorität bekommt die erste Präferenz (Gegenstück zum "nice"-Command in Unix-Betriebssystemen).**
- **Prozess Accounting Informationen, wie z.B.: wann wurde der Prozess zuletzt ausgeführt, wie viel CPU-Zeit hat er akkumuliert, usw.**
- **I/O Information (z.B. I/O Devices, die dem Prozess zugeordnet wurden, Liste der geöffneten Data Sets, usw.).**

In einer z/OS-Umgebung kann ein Programm in einem von zwei Modi ausgeführt werden:

- **TCB-Modus, in der Regel als Task-Modus bezeichnet,**
- **SRB-Modus (Service Request-Block Modus).**

Die meisten Programme - und alle im Userstatus laufenden Programme – werden als „Task“ ausgeführt. Jeder Thread der Ausführung wird durch einen Task Control Block (TCB) repräsentiert. Ein Programm kann in mehrere Tasks unterteilt werden, was eine Ausführung auf mehreren Prozessoren ermöglicht. Programme im Task Modus können I/O ausführen, auf Events warten und alle Arten von System-Diensten nutzen.

SRBs sind leichtgewichtige und effiziente z/OS Ausführungs-Threads, die nur im Supervisor State verfügbar sind. Der SRB-Modus wird von manchen Kernel Routinen benutzt, um bestimmte Performance-kritischen Funktionen auszuführen. Wenn zum Beispie ein I/O-Interrupt auftritt, dispatches z/OS einen SRB zu dem entsprechenden Adressen Raum, um einen ECB (Event Control Block) zu benachrichtigen, und um möglicherweise andere Verarbeitungsvorgänge auszuführen.

Der SRB-Modus ist weitgehend ungenutzt außerhalb des Betriebssystems und außerhalb von Middleware-Produkten wie DB2. Ein Grund dafür ist: Der SRB-Modus ist nur im Supervisor-Status verfügbar. Die meisten Programmierer glauben, SRB-Modus-Funktionen sind zu schwierig zu testen und zu debuggen. Als Folge läuft fast aller z/OS Anwendungs-Code im Task-Modus. Der SRB-Modus wird nur dort eingesetzt, wo er absolut notwendig ist.

Windows hat Prozesse Control Blöcke ähnlich zu z/OS TCBs. Das Windows Äquivalent eines SRB wird als "Fibre" bezeichnet.

19.3.2 CICS Quasi-reentrant TCB

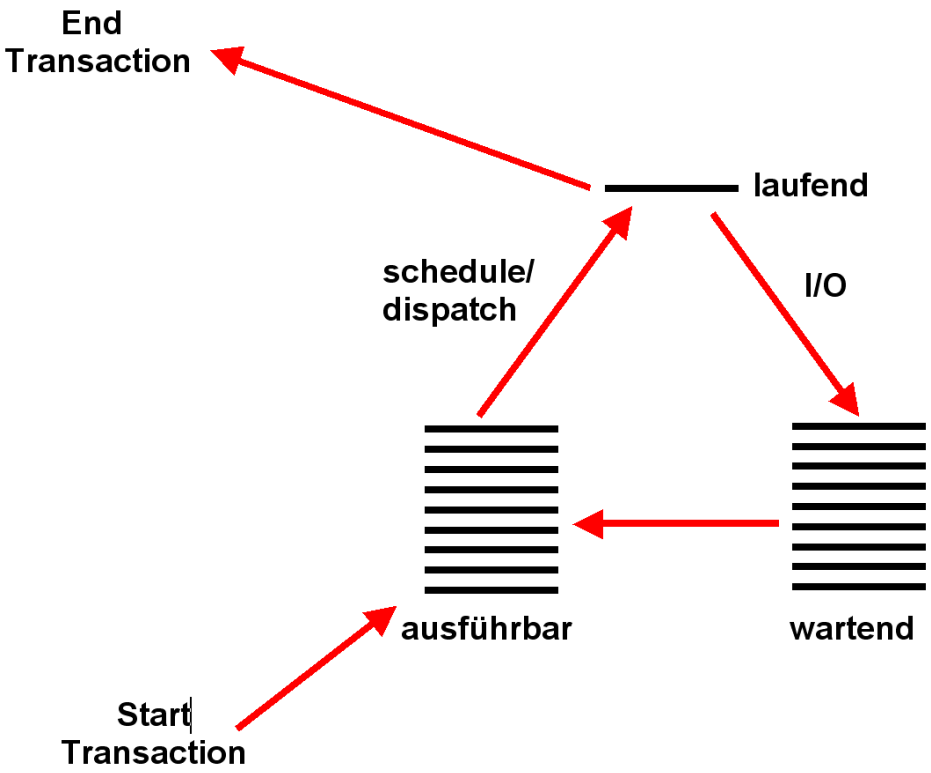

Abb. 19.3.1
Transaction State

Traditionell laufen alle CICS Transaktionen in einem einzigen Adressraums unter der Steuerung eines einzigen Prozesses (dem CICS Nucleus) und einem einzigen TCB (genannt der Quasi Reentrant TCB (QR TCB). Jedes Mal, wenn die Transaktion einen EXEC CICS-Befehl ausführt, wird die Steuerung an den CICS Nucleus transferiert. Der Programmierer muss sicherzustellen, dass die Pfad Länge zwischen EXEC CICS Kommandos kurz genug ist, um eine akzeptable Reaktionszeit (Antwortzeit) zu garantieren.

Ein in Java geschriebenes CICS-Programm benutzt keine EXEC CICS Befehle. Auch kann ein Java-Programm Ressourcen außerhalb der Kontrolle von CICS verwenden. RMI/IIOP ist ein Beispiel. Die Pfad Länge kann nicht gesteuert werden. Daher ist es nicht möglich, dass Java CICS Transaktionen mit einem QR TCB arbeiten.

19.3.3 Quasi-reentrant TCB

Unter z/OS, existieren mehrere Varianten des TCB. Es gibt zwei verschiedene Alternativen für das Betreiben eines CICS Subsystems, wobei jede mit einer anderen Art von TCB arbeitet.

In der traditionellen Alternative laufen alle Anwendungsprogramme unter einer einzigen, CICS-managed, Task Control Block (TCB), dem quasi-reentrant (QR) TCB. In diesem Fall müssen alle CICS Anwendungsprogramme quasi-reentrant geschrieben sein.

CICS quasi-reentrant Anwendungsprogramme werden durch den CICS Dispatcher unter dem QR TCB aufgerufen. Bei der Ausführung unter diesem TCB kann ein Programm sicher sein, dass keine anderes quasi-reentrant Programm ausgeführt werden kann, bis es die Kontrolle (die Verfügungsgewalt über die einzige CPU) aufgrund eines EXEC CICS-Commands aufgibt. An diesem Punkt wird die Transaktion unterbrochen (Waiting oder Ready State). Während dieser Warte Zeit kann eine andere Transaktion das gleiche, quasi-reentrant Anwendungsprogramm ausführen. Dies bedeutet, ein quasi-reentrant Anwendungsprogramm kann gleichzeitig von mehr als einer Task benutzt werden, obwohl nur eine Task es in jedem Augenblick ausführen kann.

Es existiert nur eine einzige Kopie des quasi-reentrant Anwendungsprogramms im Hauptspeicher. Um sicherzustellen, dass parallel laufende Transaktionen, welche das gleiche quasi-reentrant Anwendungsprogramm benutzen, sich nicht mit der Nutzung des Arbeitsspeichers stören, unterhält CICS eine getrennte Kopie des Arbeitsspeichers für jede Transaktion. Wenn somit eine Kopie eines Anwendungsprogramms gleichzeitig von 11 Tasks benutzt wird, existieren 11 Kopien des Arbeitsspeichers in dem entsprechenden dynamischen CICS Speicherbereich (Dynamic Storage Area, DSA), siehe Band 1, Abschnitt 8.3.7.

Achten Sie darauf, wenn ein Programm langwierige Berechnungen durchführt, weil ein Anwendungsprogramm die Kontrolle (Verfügungsgewalt über die CPU) von einem EXEC CICS Command bis zum nächsten behält. Die Verarbeitung von anderen Transaktionen unter dem QR TCB ist während dieser Zeit ausgeschlossen. CICS terminiert eine Transaktion, die nicht vor Ablauf eines angegebenen Zeit Intervalls die Kontrolle abgibt..

Der CICS QR TCB bietet Schutz durch die ausschließliche Kontrolle der globalen Ressourcen nur dann, wenn alle Transaktionen, die auf diese Ressourcen zugreifen unter dem QR TCB laufen. Es bietet keine automatischen Schutz vor anderen Prozessen, die gleichzeitig unter einem anderen (offenen) TCB ausgeführt werden.

19.3.4 Open TCB

Wenn alle Transaktionen unter einem QR TCB laufen, impliziert dies, dass eine CICS Region nur mit einer einzigen CPU läuft. Es ist aus diesem Grund, dass mehrere Application Owning Regions (AOR) beliebt sind (siehe Abb. 9.3.4). Jede AOR läuft auf einer anderen CPU.

Das Open Transaction Environment (OTE) ist ein Umfeld, in dem

- **Eine einzelne CICS-Region mehrere CPUs benutzen kann, und**
- **CICS Anwendungscode nicht-CICS Dienste (Einrichtungen außerhalb des Umfangs der CICS API) innerhalb des CICS Adressraum ausführen kann, ohne Interferenz mit anderen Transaktionen.**

.

Das Problem ist, ruft eine Transaktion einen Service außerhalb der CICS-Adressenraums auf, kann diese Transaktion für eine lange Zeit zu sperren, ohne dass der CICS Nucleus dessen bewusst sein muss. Aus diesem Grund läuft jede Anwendung (Transaction), die das Open Transaction Environment (OTE) nutzt, unter einem eigenen open TCB. In der CICS Region können mehrere aktive (parallel laufende) Programme mit eigenen open TCBs existieren anstatt eines einzigen aktiven Programms mit einem QR TCB. Während der QR TCB vom CICS Nucleus dispatched wird, erfolgt das open TCB Dispatching durch den z/OS Kernel. Wenn eine open TCB Transaktion einen non-CICS Service aufruft, die den TCB blockiert, hat dies keine Auswirkungen auf andere open TCB CICS Anwendungen.

Es gibt auch andere Verwendungen für das Open Transaction Environment. Allerdings hat das alles seinem Preis. Die Komplexität wächst, und die Performance kann schlechter werden. So ergänzt der offene TCB den QR TCB, aber ersetzt ihn nicht, und der QR TCB bleibt sehr beliebt. In einer gegebene z/OS-Installation können einige CICS Regionen mit dem QR TCB, und andere mit dem open TCB laufen.

Es existieren mehrere Varianten des open TCB. Einige von ihnen sind speziell für CICS Java-Anwendungen bestimmt.

Details unter
http://www.cedix.de/VorlesMirror/Band2/OTE.pdf
http://www.cedix.de/VorlesMirror/Band2/OpenTCB.pdf

19.3.5 Programming CICS Applications in Java Enterprise Edition

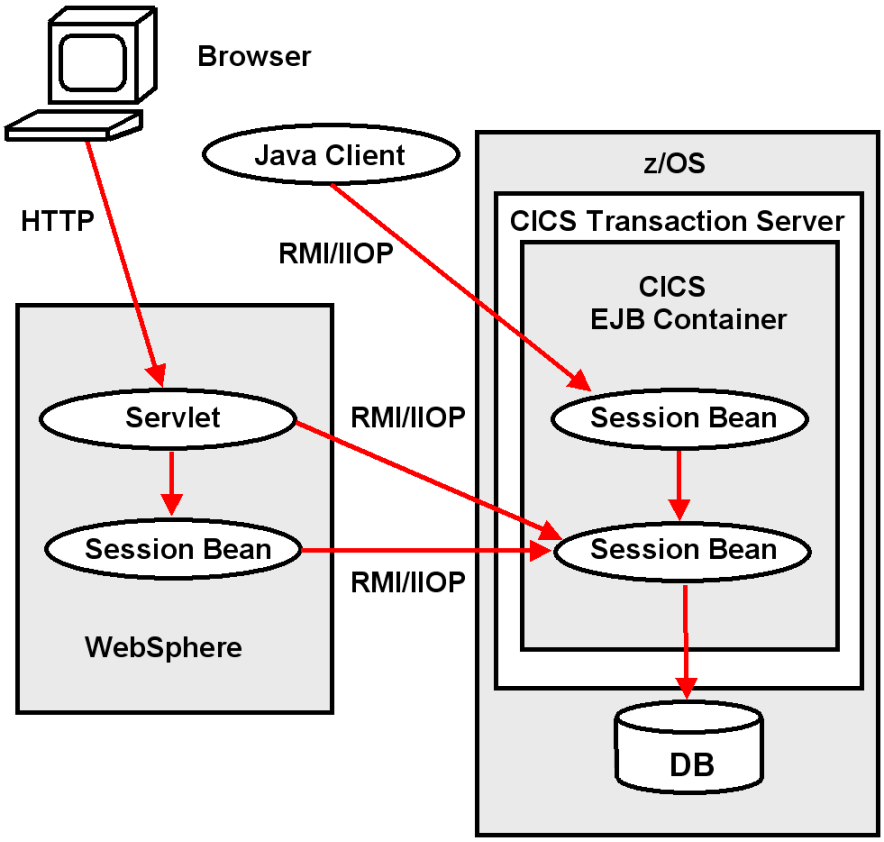

Abb. 19.3.2
Benutzung von Enterprise Java Beans in einer CICS Anwendung

Die meisten CICS-Anwendungen wurden bisher in Cobol, PL/I oder Assembler programmiert. C/C++ wurde weniger häufig verwendet. Mit der Einführung von Java wurde es möglich, CICS-Anwendungen auch in Java zu programmieren. Viele Experten gehen davon aus, dass Java die dominierende Sprache werden wird, um neue CICS-Anwendungen zu programmieren. Es gibt drei alternative Möglichkeiten, um CICS-Anwendungen in Java zu schreiben:

1. **Der z/OS "High Performance Java Compiler" hat eine Option, Objekt-Code wie ein Cobol oder PL/I Compiler zu generieren. Dieser Objekt Code wird gelinked und geladen wie jede andere Code. Der Java Quellcode ist auf andere Plattformen übertragbar, der Objekt-Code aber nicht. Der Java Standard unterstützt diese Möglichkeit nicht, sie existiert außerhalb des Java Standards.**

2. **Alternativ kann der z/OS High Performance Java Compiler Byte Code generieren. Der Byte-Code wird innerhalb einer JVM, die als Teil des CICS Transaction Server installiert wird, ausgeführt. Es wird die JSE an Stelle der JEE benutzt.**

   Es gibt jedoch ein Problem mit diesem Ansatz. Die Java-Sprache erlaubt nicht den Einsatz von EXEC-Anweisungen. Deshalb wird die EXEC CICS-Schnittstelle durch eine Reihe von Java Bibliothek Aufrufe ersetzt, durch die „JCICS“ Schnittstelle.

3. **Beide bisher genannten Ansätze benutzen keinen EJB-Container. Wenn Sie JEE Einrichtungen mit CICS verwenden möchten, können Sie eine JEE EJB Container (mit einer eigenen JVM) innerhalb des CICS Transaction Servers installieren, siehe Abb. 9.3.2 . Dies ist in Wirklichkeit ein Corba ORB. Es ermöglicht die Verwendung von RMI/IIOP.**

Die letzten beiden Ansätze erfordern die Verwendung des open TCB, da die Java-Programme Einrichtungen außerhalb von CICS verwenden können.

CICS unterstützt keine Entity Beans. Es benutzt eigene Verfahren, um die Persistenz zu managen.

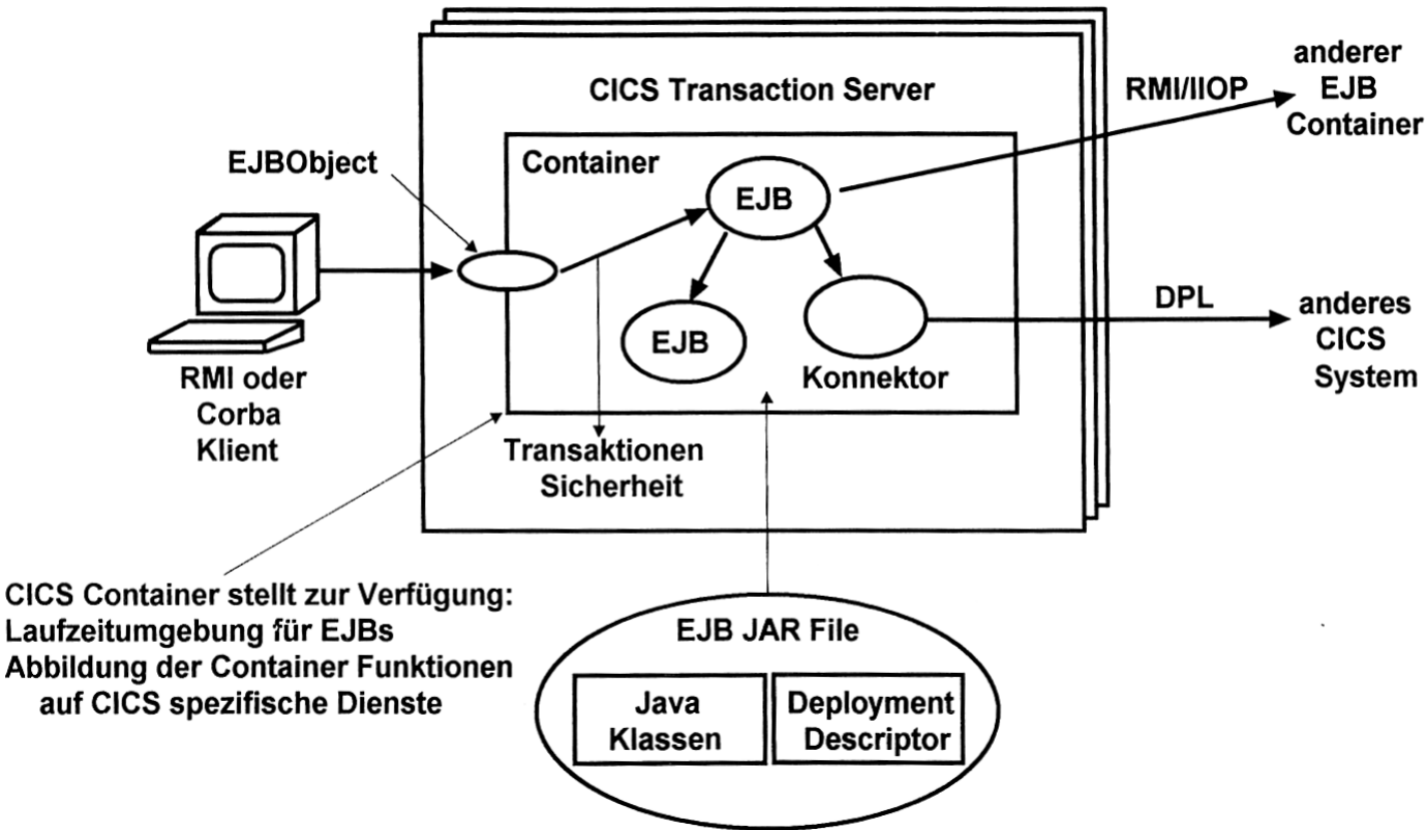

Abb. 19.3.3
CICS EJB Container

CICS Session Beans können über RMI/IIOP aufgerufen werden. Jede Java-Klasse oder jeder Corba Client ist in der Lage, mit RMI/IIOP auf eine CICS Session Bean zuzugreifen. CICS Session Beans können ihrerseits anderen EJBs außerhalb CICS via RMI/IIOP aufrufen. Außerdem können sie mittels DPL mit Programmen in anderen CICS Servern kommunizieren, auch wenn diese z.B. in Cobol geschrieben wurden, siehe Abb. 19.3.3.

Im Vergleich zu anderen Umgebungen können Enterprise Beans in einem CICS EJB Container alle CICS Transaction Management Services wie System Log Management, Performance Optimization, Runaway und Deadlock Detection sowie Monitoring und Statistics nutzen.

Der CICS EJB Container benutzt das X/Open Distributed Transaction Processing Model.

19.3.6 Programmieren von CICS Transaktionen in der Java Standard Edition

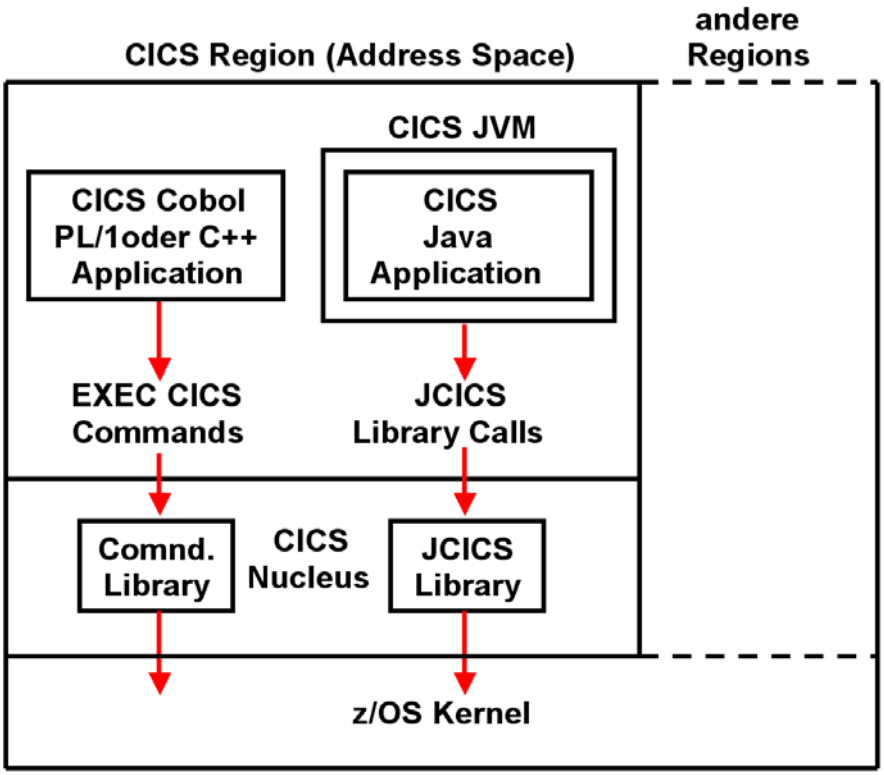

Abb. 19.3.4
JCICS Java Anwendungsprogrammierung

JEE Server wie WebSphere, Web Logic, Netweaver, und Open-Source Produkte wie Jboss und Geronimo sind sehr beliebte Transaction Processing Produkte. CICS kann ebenfalls einen JEE-Server integrieren. Jedoch ist die JSE (Java Standard Edition) eine überraschend attraktive Alternative für CICS Java-Programme. Dies ist der Grund:

- **Ein JEE Server integriert einen Transaktion Monitor in eine bestehendes JEE Infrastruktur, die viele Funktionen aufweist, die nicht direkt mit der Transaktionsverarbeitung zu tun haben. JNDI und RMI/IIOP sind Beispiele. Viele dieser Funktionen werden von CICS eigentlich nicht benötigt, und stellen einen überflüssigen Overhead dar.**
- **CICS integriert Java in einer vorhandenen Transaktion Processing Infrastruktur. OTE ermöglicht es, JSE Programmen direkt unter CICS laufen zu lassen.**

Wie alle anderen Java-Programme laufen CICS JSE Programme innerhalb einer JVM. Sie können Seite an Seite mit Cobol, PL/I und C/C++ CICS Programmen ausgeführt werden.

Java hat aber keine Einrichtungen um die EXEC CICS Aufrufe zu implementieren. Deshalb benutzen Java-Programme statt dessen die JCICS Bibliotheksroutinen. Die JCICS Bibliotheksaufrufe ersetzen eins zu eins die EXEC CICS Commands und nehmen damit den Platz des EXEC CICS Befehle ein.

19.3.7 Parallele Ausführung von JSE Java Transaktionen

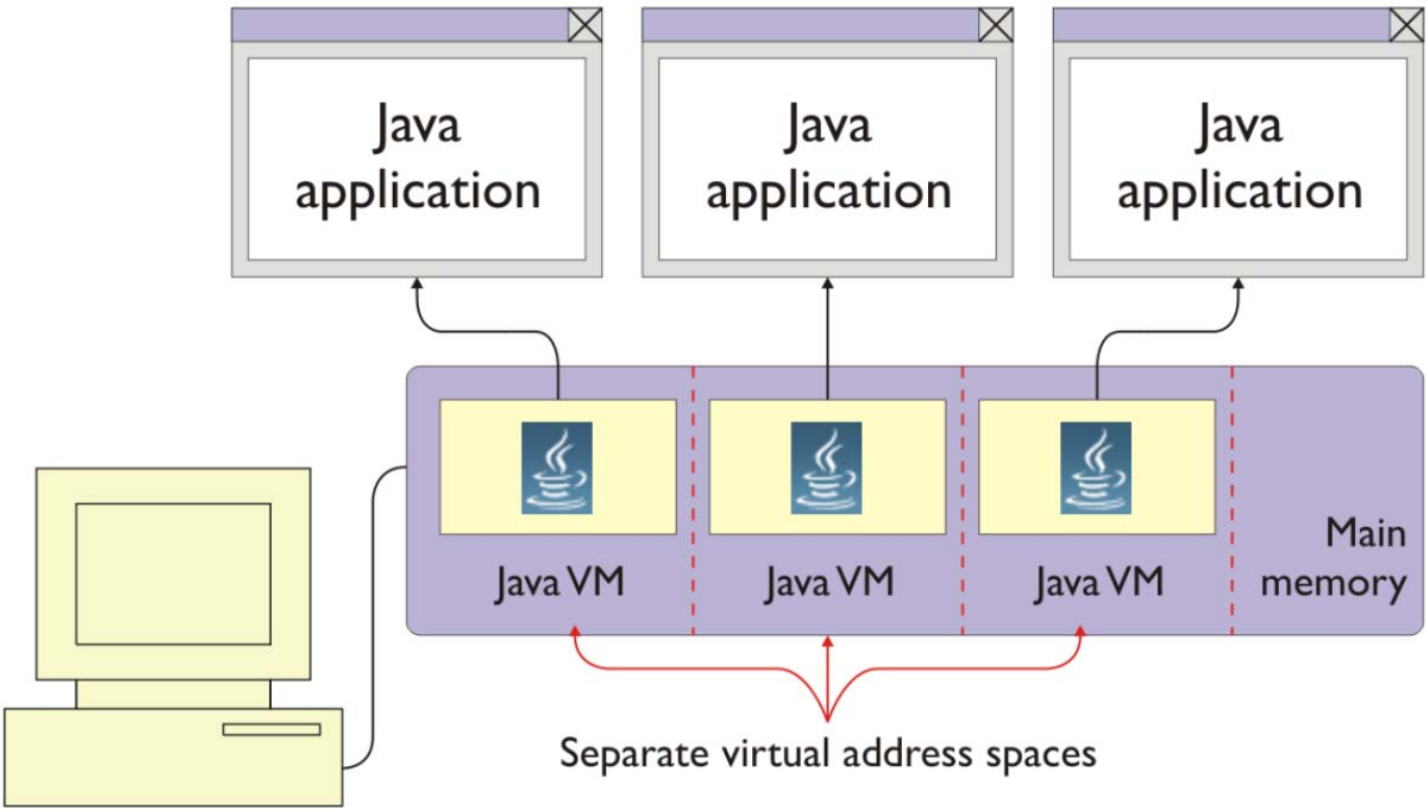

Abb. 19.3.5
Multiprogrammierung mit Java

Der einfachste Weg, JSE Transaktionen parallel auszuführen ist, jeder Transaktion eine separaten JVM zuzuordnen, und eine getrennte z/OS Region jeder JVM. Das ist ziemlich aufwendig in Bezug auf den Verbrauch von CPU-Zyklen. Unter CICS laufen daher alle JSE-Anwendungen genauso wie Cobol und PL/I Anwendungen innerhalb eines einzigen Adressraums.

Es gibt zwei JSE Alternativen, dies zu tun: die JVM Pool Architektur und die JVM Server Architektur. Wir diskutieren die JVM Pool Architektur zuerst.

19.3.8 CICS JVM Pool Architektur

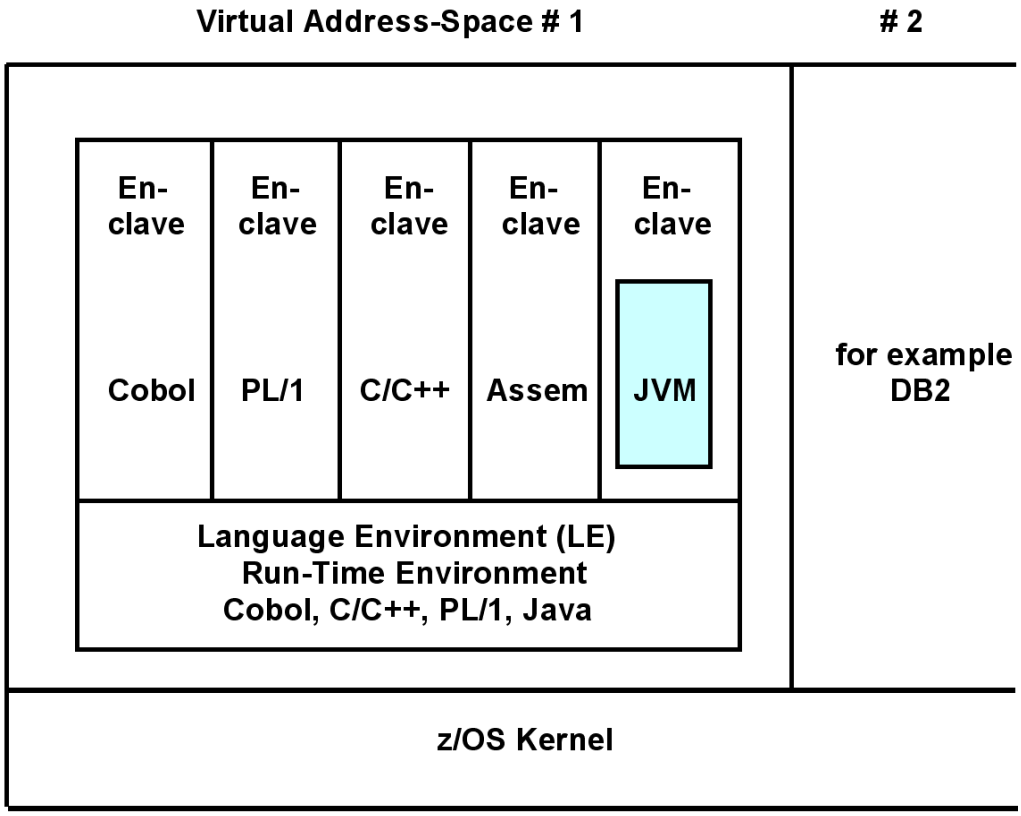

Abb. 19.3.6
Multiple JVMs

CICS-Transaktionen, die in Cobol, C/C++, PL/I oder Assembler geschrieben sind, laufen in getrennten Enclaves; eine Enclave für jede Transaktion.

CICS Java-Transaktionen werden in getrennten JVMs ausgeführt, jeweils eine JVM pro Enclave. Java Threads werden in der Pool Architektur nicht benutzt.

Cobol, PL/I, C/C++ und Java Enclaves können gleichzeitig innerhalb einer CICS-Region laufen.

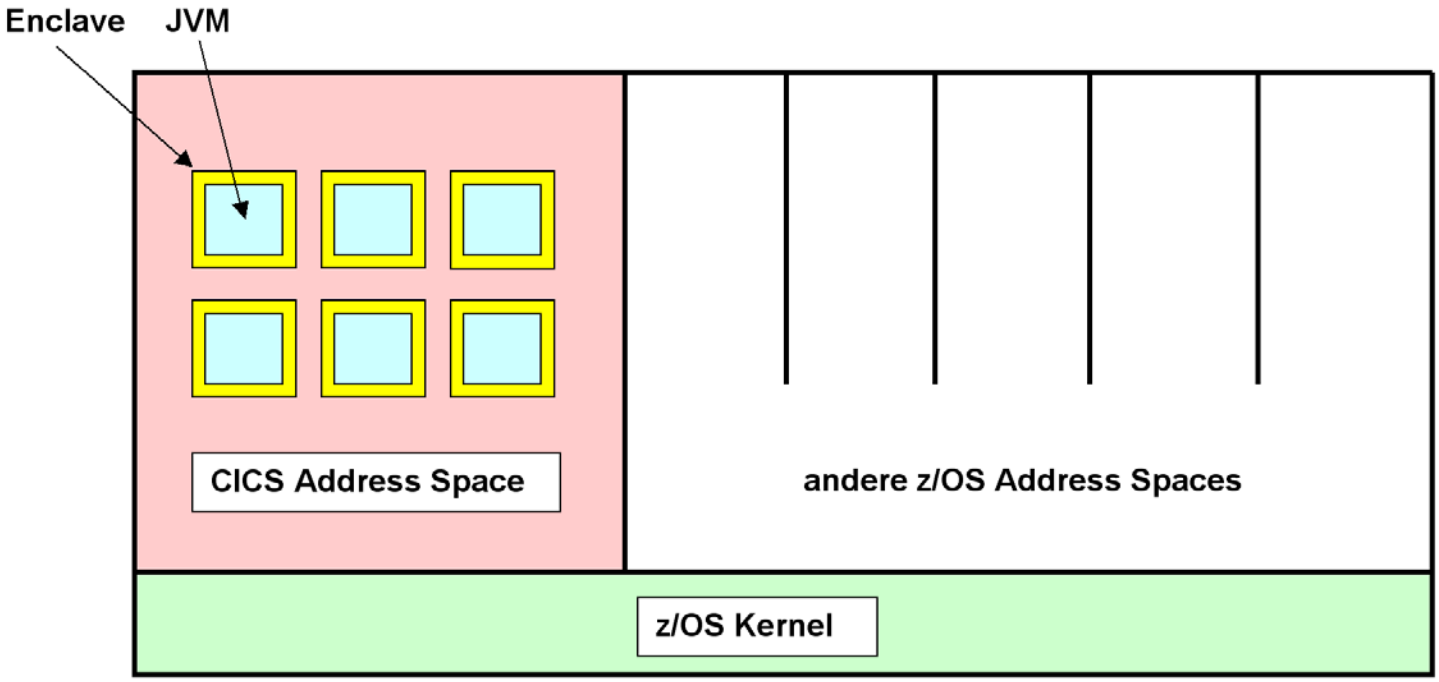

Abb. 19.3.7
JVM und Enclave

Ein wichtiger Punkt ist es die gegenseitige Isolation von mehreren Java Transaktionen (das „I“ in ACID) zu gewährleisten. Die JVM Pool Architektur ist ein gradliniger Ansatz. Hier unterhält CICS mehrere JVMs in seinen Adressraum, als JVM Pool bezeichnet. Einzelne CICS Tasks werden jeweils einer JVM in dem Pool zugeordnet, und nur eine einzige Java Anwendung läuft in jeder JVM. Getrennte JVMs garantieren die Isolation zwischen den Java Anwendungen.

Theoretisch sind > 100 JVMs möglich. Allerdings wird viel Speicherplatz benötigt (~ 20MByte + Heap). Die Anzahl der gleichzeitig laufenden Transaktionen ist durch die Anzahl der JVMs begrenzt, die in eine Region passen. Das Ergebnis sind maximal 20 gleichzeitige JVMs in einer 2 GByte Region.

19.3.9 JVM Zuordnung

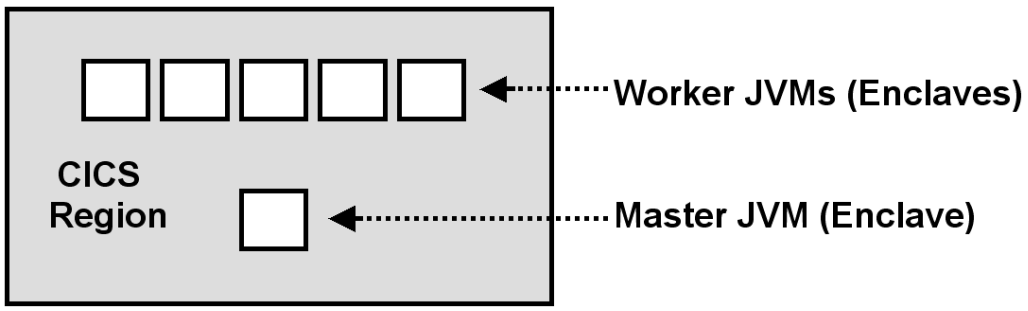

Abb. 19.3.8
Master JVM und Enclave

LE Enclaves können eingesetzt werden um unterschiedliche Anwendungen innerhalb des gleichen virtuellen Adressenraums voneinander zu isolieren. Sie werden u.a. in transaktionalen Subsystemen wie CICS, IMS und DB2 benutzt. Spezifisch ist es hiermit möglich, mehrere JVMs innerhalb eines CICS Adressenraums laufen zu lassen.

Es existiert immer eine Enclave für jede JVM. Eine Master JVM Eine Master JVM cloned je nach Bedarf mehrere Worker JVMs und steuert sie. Transaktionen werden in den Worker JVMs ausgeführt.

CICS unterhält eine „Shared Class Cache Facility für die JVM. Mehrere JVMs können gemeinsam einen einzigen Cache mit Class Files nutzen. Die gemeinsam genutzte Cache ersetzt den System-Heap und die Anwendungsklassen für die JVMs. JVMs, die gemeinsam genutzte Klassen verwenden, starten schneller und haben einen geringere Speicherplatzbedarf als JVMs, die dies nicht tun.

Die Master JVM initialisiert u.A. den Shared Class Cache. Die Master JVM wird nicht für die Ausführung von Java Anwendungen benutzt.

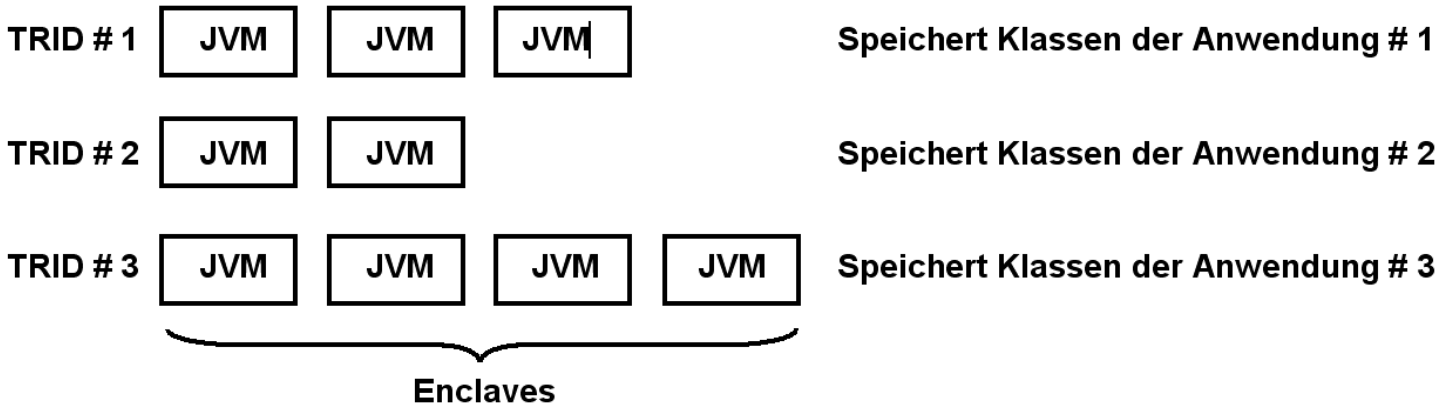

Abb. 19.3.9
Enclaves mit identischen geladenen Klassen

Wenn eine neue Transaktion eintrifft, sucht CICS eine freie Enclave mit Ihrer vorinstallierten JVM, um in dieser die Transaktion auszuführen. Wenn möglich, sucht CICS eine JVM aus, in der die gleiche Anwendung, gekennzeichnet durch ihre TRID, bereits gelaufen ist. In diesem Fall müssen die Anwendungsklassen nicht neu geladen werden.

Nehmen wir an, in einer Java CICS Region sind 20 JVMs vorhanden. In jeder JVM sind die Anwendungsklassen einer spezifischen Java CICS Anwendung gespeichert. Häufig sind die Klassen einer Anwendung in mehr als einer JVM gespeichert.

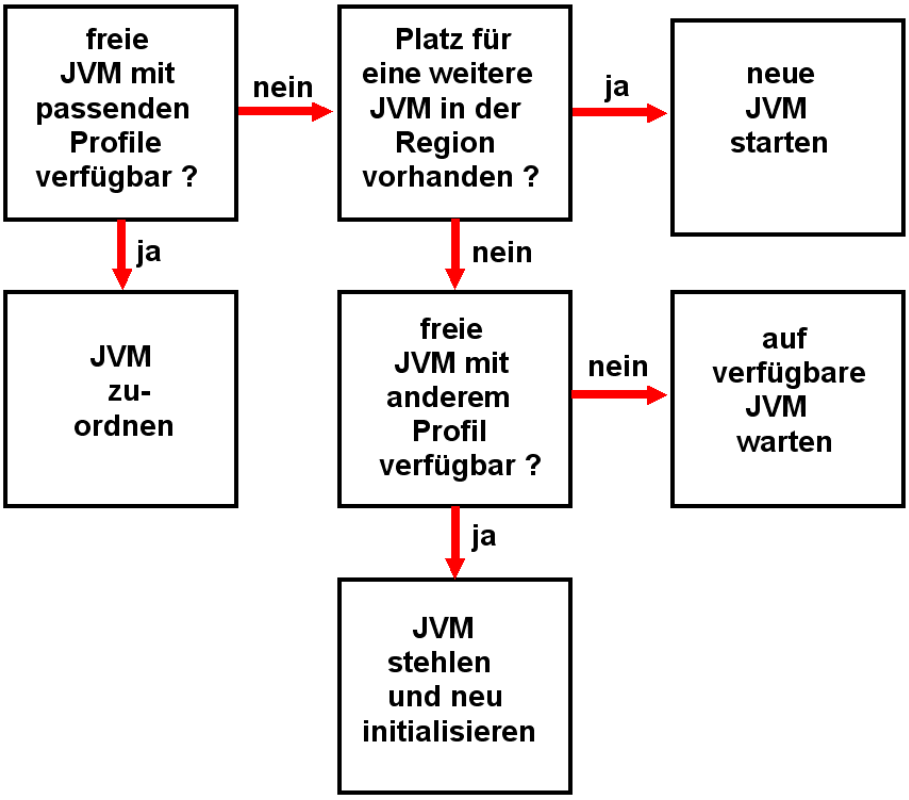

Abb. 19.3.10
Auswahlalgorithmus für die Zuordnung einer JVM

Dargestellt ist der Algorithmus, mit dem beim Starten einer weiteren Transaktion eine passende JVM für deren Bearbeitung ausgewählt wird.

Ein bestimmter Transaktionstyp wird durch seine TRID gekennzeichnet, und kann von mehreren Transaktionen parallel ausgeführt werden. Unterschiedliche Transaktionstypen unterscheiden sich durch unterschiedliche Anwendungsklassen. Wenn eine neue Transaktion gestartet wird, wird nach Möglichkeit eine freie JVM ausgewählt, in der die richtigen Anwendungsklassen bereits geladen sind. Wenn das nicht möglich ist, werden die benötigten Anwendungsklassen in eine verfügbare Worker JVM nachgeladen.

19.3.10 Ausführung einer Folge von Transaktionen

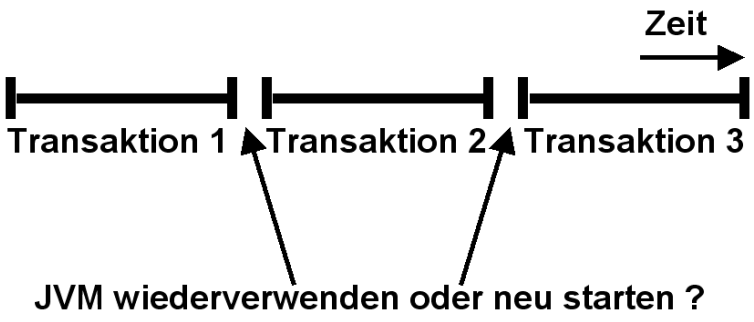

Abb. 19.3.11
Serielle Nutzung einer JVM

Eine Transaktion kann die Ausführung der Folge-Transaktion beeinflussen, indem sie den Zustand (State) der JVM ändert. Beispiele für eventuelle sicherheitskritische Reste der vorhergehenden Transaktion sind:

- **überschriebene statische Variablen**
- **Starten von Threads**
- **geladene native Bibliotheken.**

Klassischer Ansatz: Für jede Transaktion wird eine neue JVM gestartet und nach Abschluss der Transaktion wieder beendet.

Das Hoch- und Herunterfahren einer JVM hat jedoch einen erheblichen Zeitaufwand zur Folge. Bei jeder Initialisierung einer JVM werden

- **60 System Klassen geladen,**
- **700 Array-Objekte und**
- **1000 non-Array-Objekte allokiert und angelegt.**

Pfadlänge bis zu 100 Millionen Maschinenbefehle, ca. 0,1 Sekunde Verarbeitungsdauer, sind möglich.

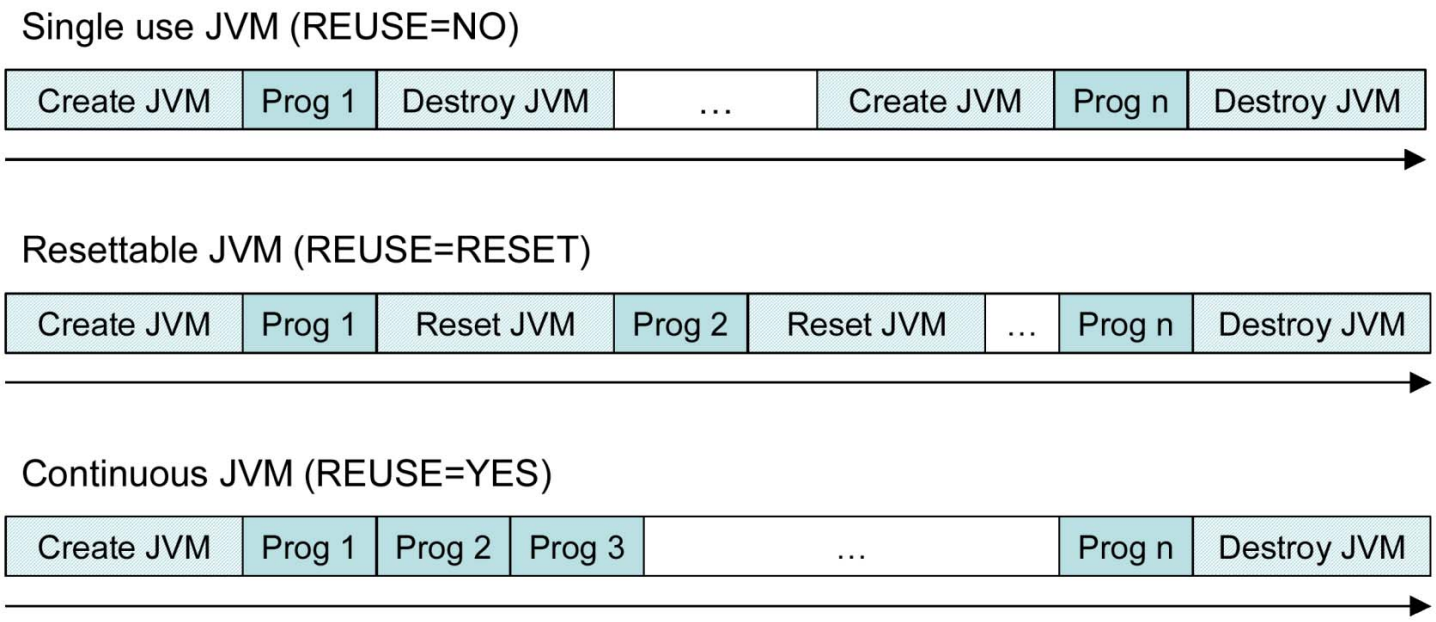

Abb. 19.3.12
CICS JVM Modus

In the CICS JVM Pool Architecture kann eine JVM kann in einem der folgenden Modi laufen:

1. **Single Use mode. Die JVM wird zerstört, und für die nächste Transaktion wird eine neue JVM gestartet.**
2. **Resettable mode. Eine Zusatzeinrichtung bewirkt, dass der ursprüngliche Zustand der JVM wiederhergestellt wird.**
3. **Continuous mode. Es ist Aufgabe des Anwendungs-Programmierers, sicherzustellen, den dass der Zustand der JVM nicht verändert wird..**

Der Single Use Modus hat den Nachteil, dass der Aufwand für das Starten einer JVM sehr hoch ist. Dies hat zur Folge, dass der Single Use Modus aus Performance Gründen nur in Ausnahmefällen eingesetzt werden kann.

Der Continuous Use Modus hat das Problem, dass die JVM nach Beendigung einer Transaktion sich im gleichen Zustand wie zu Beginn einer Transaktion befinden muss. Dies hat der Anwendungsprogrammierer zu gewährleisten. Was hierzu zu tun ist, ist jedoch nicht ausreichend dokumentiert, und erfordert sehr tief gehende Java Kenntnisse, über die ein durchnittlicher Java Programmierer nicht notwendigerweise verfügt.

Dieses Problem löst der Resettable Modus, der jedoch nicht Teil des JEE Standards ist, und somit eine unerwünschte IBM proprietäre Erweiterung des Standards darstellt ist.

Dr. Jens Müller hat in seiner Diplomarbeit: " Anwendungs- und Transaktionsisolation unter Java ", Mai 2005, eine detaillierte Untersuchung der JVM Isolationseigenschaften durchgeführt. Siehe http://www.cedix.de/VorlesMirror/Band2/javaisol.pdf. Viele der Isolationsprobleme können durch eine sorgfältige Programmierung umgangen werden. Im Vergleich zu Cobol CICS Programmen ist dies jedoch eine zusätzliche Herausforderung.

19.3.11 CICS JVM Server Architecture

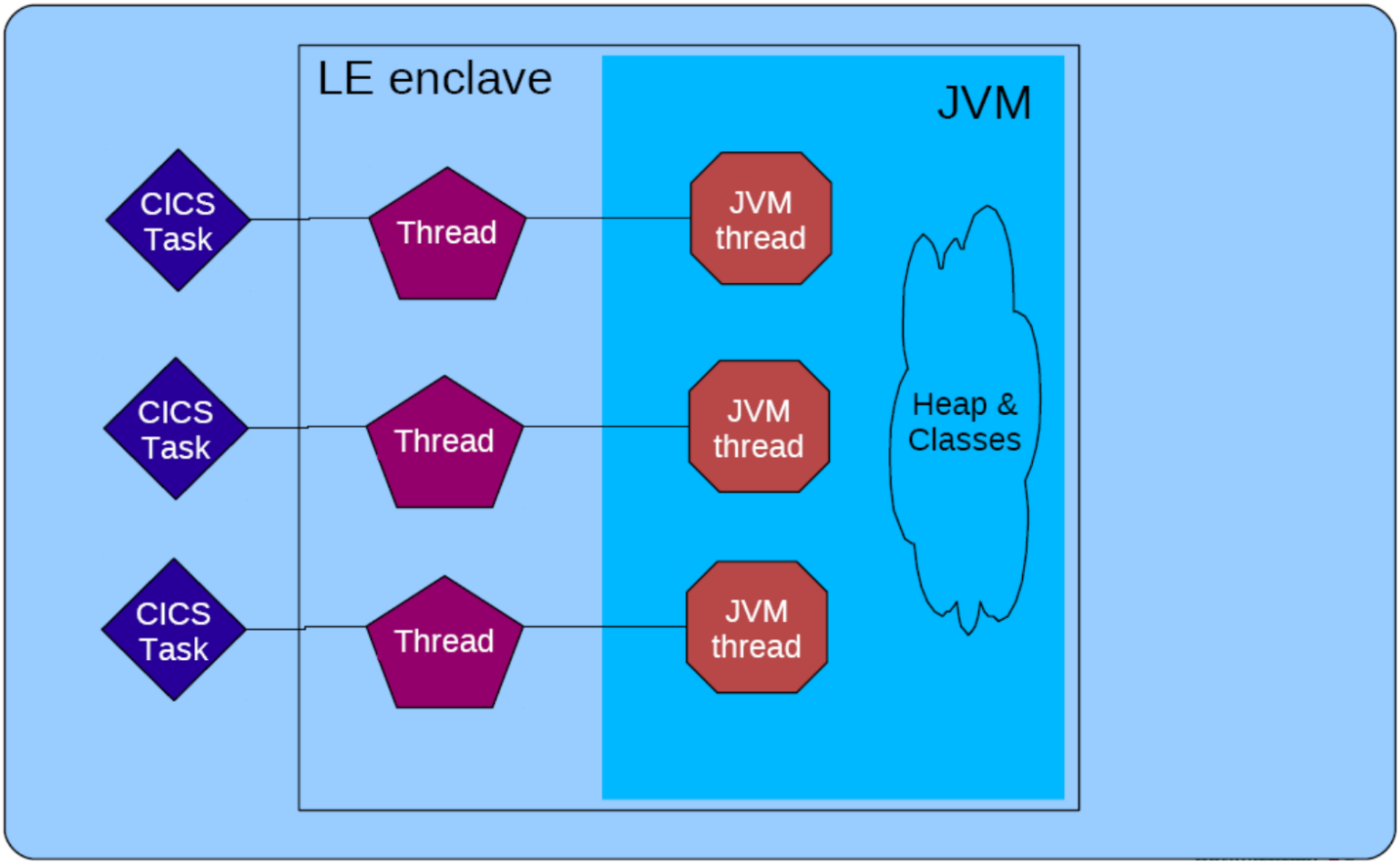

Abb. 19.3.13
Enclave Threads und JVM Threads

Der "JVM Server" ist eine neue Implementierung der JVM. Es laufen mehrere CICS Transaktionen als unabhängige JVM Threads innerhalb einer einzigen JVM. Jeder JVM Thread wird auf einen LE Enclave Thread abgebildet. Die JVM Server Architektur wird möglicherweise die JVM Pool Architektur ablösen.

Über viele Jahre benutzten CICS und WebSphere die gleiche, als „Sovereign“ bezeichnete Implementierung der JVM. Unterschiedliche Optimierungspotentiale führten dazu, dass WAS heute die „J9“ JVM verwendet. Diese und die Sun „HotSpot“ JVM gelten als die performantesten verfügbaren JVMs. Die J9 wird auch in der JSE und der JME eingesetzt.

CICS entwickelte statt dessen die „JVM Server“ JVM, die spezifisch für die CICS Bedürfnisse optimiert wurde. Die JVM Server JVM ermöglicht es, mehrere Transaktionen als Java Threads innerhalb einer einzigen JVM laufen zu lassen.

Die JVM Server Umgebung ermöglicht die gegenseitig Isolation von mehreren Java Anwendungen innerhalb einer JVM durch den Einsatz von Industrie-Standard OSGi-Bundles. Siehe http://www.cedix.de/VorlesMirror/Band2/OSGiArchi.pdf. Das OSGi-Framework innerhalb des JVM Servers bietet erforderliche Quarantäneeinrichtungen für mehrere gleichzeitig ausgeführte CICS Java Programme innerhalb der gleichen JVM. Dazu müssen CICS Java-Anwendungen in einem OSGi-Bundle verpackt werden, und dann als CICS BUNDLE Ressource mit Verweis auf die OSGi-Bundle deployed werden.

Der JVM-Server bietet auch integrierte Statistikfunktion zur Überwachung der JVM Garbage Collection. Dies erleichtert das Performance Tuning von Java CICS Anwendungen.

Der OSGI Ansatz wird auch von dem Android Betriebssystem als Laufzeitumgebung für Java Apps verwendet, siehe
http://felix.apache.org/site/presentations.data/OSGi%20on%20Google%20Android%20using%20Apache%20Felix.pdf,
oder als Mirror unter http://www.cedix.de/VorlesMirror/Band2/OSGI.pdf.

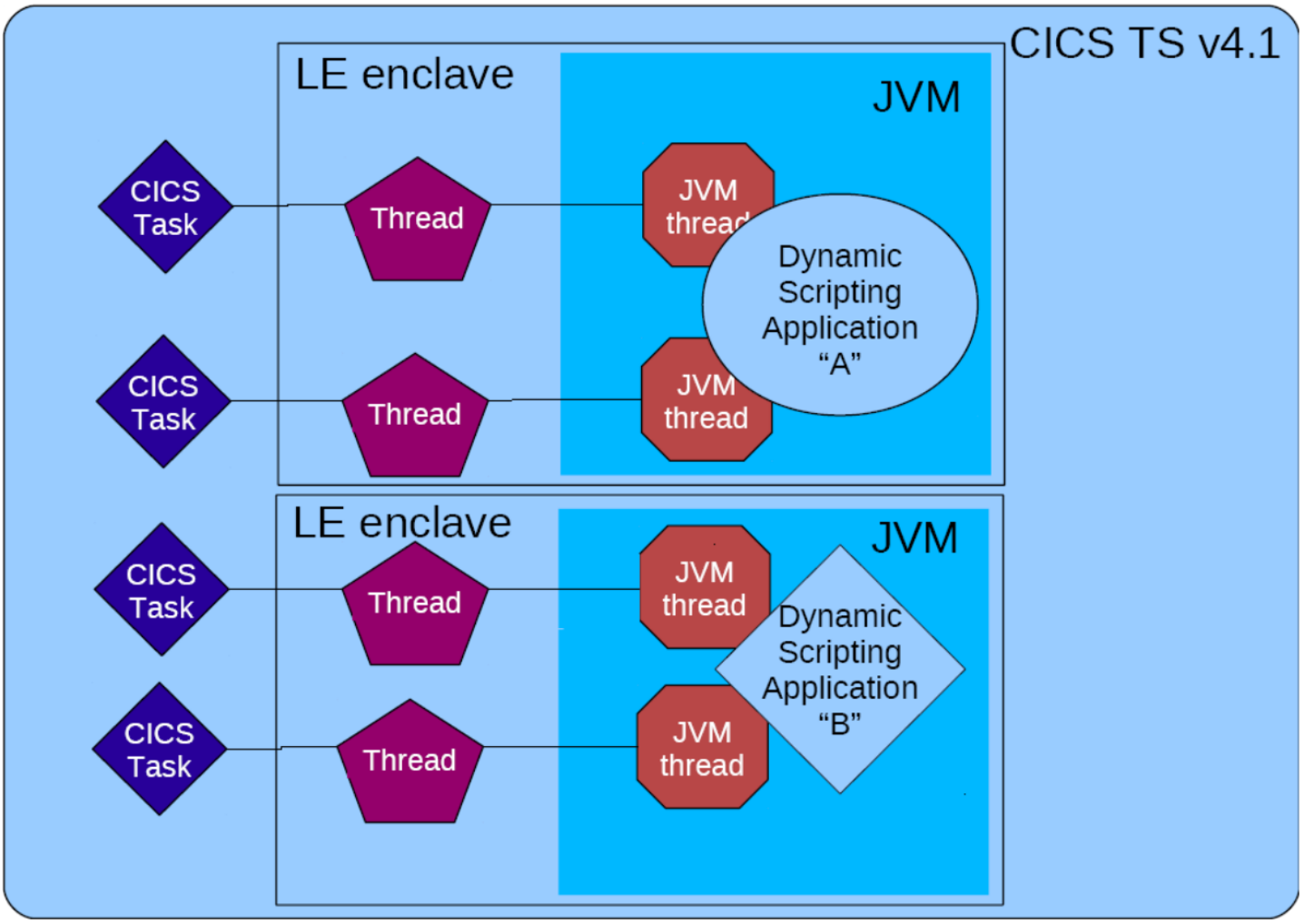

Abb. 19.3.14
Multiple JVMs mit Multiple Threads pro JVM

Es ist möglich, mehrere JVM Server in einem einzigen CICS Adressenraum auszuführen, genauso wie bei der CICS JVM Pool Architektur . Eine praktische Grenze ist etwa 20 JVMs innerhalb eines CICS Adressenraums.

19.3.12 Facit

Wie vollständig ist die Isolierung der Threads innerhalb des JVM Servers ?

Besser als alles, was es vorher gab, aber nicht so vollständig, wie man es gerne hätte.

Robert Harbach hat in seiner Master Thesis: "CICS JVM Server Application Isolation", März 2012, eine detaillierte Untersuchung der JVM Server Isolationseigenschaften durchgeführt. Während die Einführung des OSGi-Frameworks viele Probleme löst, müssen eine Reihe von Isolation Fragen noch geklärt werden. Eine sorgfältige Programmierung kann diese Probleme umgehen. Das Entwicklungsrisiko wächst jedoch mit der Größe und Komplexität einer neuen CICS Java-Anwendung.

Die Master Thesis kann heruntergeladen werden unter

http://www.cedix.de/VorlesMirror/Band2/harbach.pdf

Es ist eine dringende Erfordernis, dass eine zukünftige Version des Java Standards das Thread Isolationsproblem der JVM an der Wurzel anpackt, und eine zufriedenstellende Lösung anbietet. Dieses und verwandte Probleme haben dazu geführt, dass sich Java bisher nur für die Präsentationslogik von neuen Java Anwendungen durchgesetzt hat, während der Business Logik teil häufig immer noch in Cobol oder PL/I programmiert wird.

19.5 Weiterführende Information

Eine Übersicht über den Java Tranaction Service ist enthalten in
http://www.torsten-horn.de/techdocs/jee-transaktionen.htm.

Die erwähnte Diplomarbeit von Dr. Jens Müller: Anwendungs- und Transaktionsisolation unter Java, Mai 2005, können Sie herunterladen unter
http://www-ti.informatik.uni-tuebingen.de/~spruth/DiplArb/jmueller.pdf

Siehe auch
http://www.cedix.de/VorlesMirror/Band2/javaisol.pdf

Eine weitere Arbeit zum Thema Java Isolation finden Sie unter
http://www.cedix.de/VorlesMirror/Band2/JavaIsol05.pdf

Die erwähnte Masterarbeit von Robert Harbach: "CICS JVM Server Application Isolation", March 2012, können Sie herunterladen unter
http://www.cedix.de/VorlesMirror/Band2/harbach.pdf

Siehe auch
http://www.cedix.de/VorlesMirror/Band2/GIIS16-1.pdf

Ein Oracle Tutorial finden Sie unter
http://docs.oracle.com/javase/tutorial/jdbc/basics/transactions.html

Eine Video Aufnahme einer Vorlesung zum Thema Java Transaction Management ist zu finden unter
http://freevideolectures.com/Course/2979/Java-EE-Programming-Spring-2012/12

Ein Vortrag „Understanding, Monitoring and Managing z/OS Enclaves" ist zu finden unter
http://www.cedix.de/VorlesMirror/Band2/EnclaveWLM.pdf

Eine gute Übersicht über verteilte Transaktionsverarbeitung ist zu finden unter
http://publib.boulder.ibm.com/infocenter/txformp/v7r1/index.jsp?topic=/com.ibm.cics.tx.doc/concepts/c_xopen_intface_fr_accsg_res_mgrs.html