6. VSAM

6.1 Arten von Datasets

6.1.1 Datenspeicherung

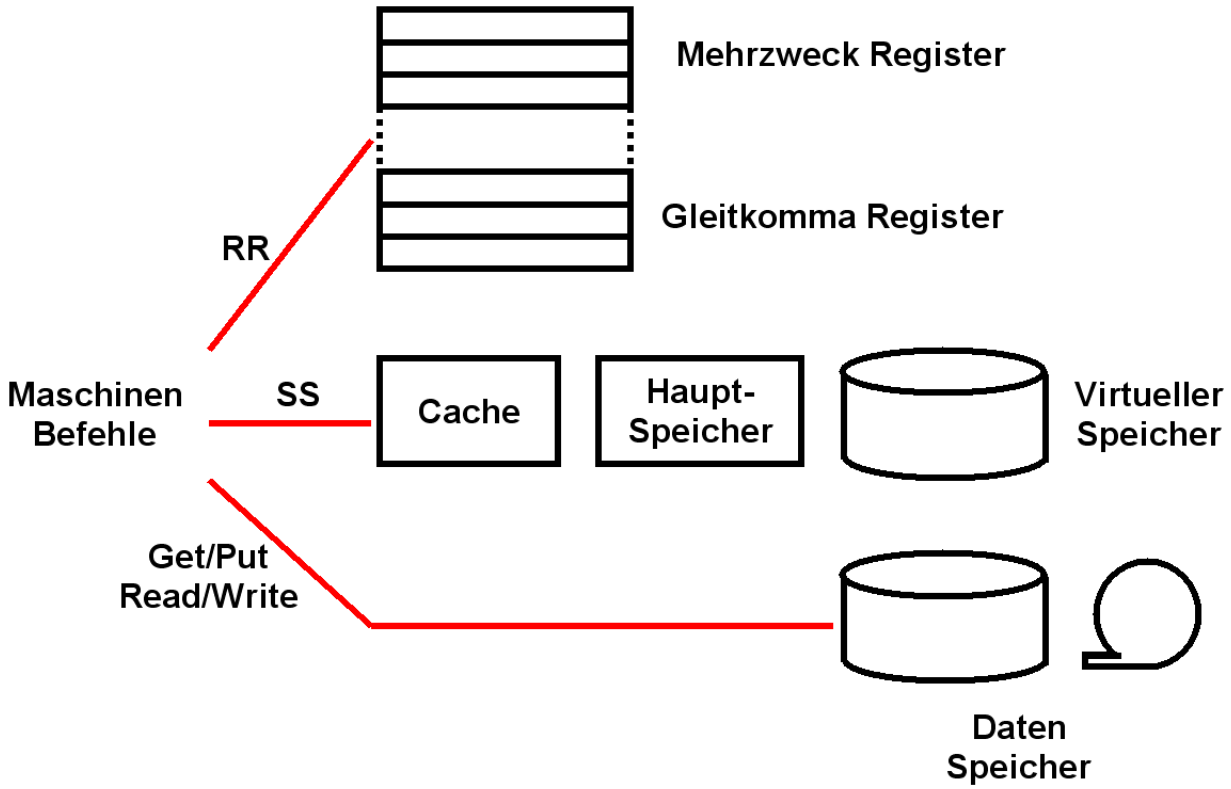

Abb. 6.1.1
Unterschiedliche Arten der Datenspeicherung

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.

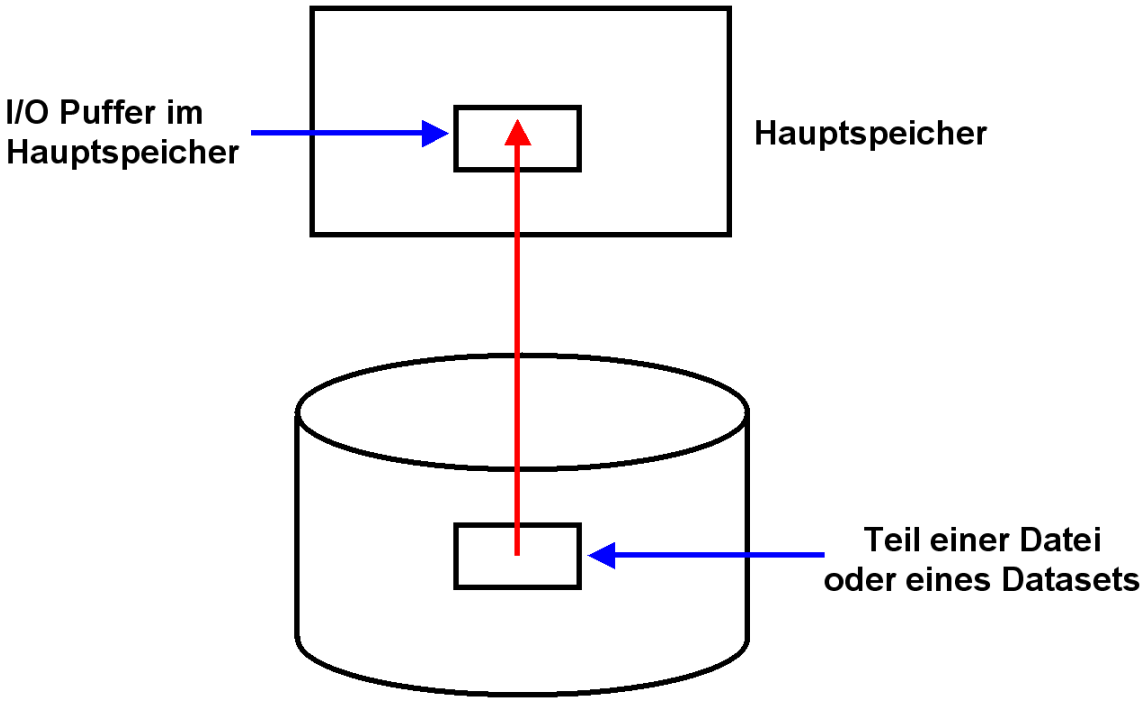

Abb. 6.1.2
Teile einer Datei werden im I/O Puffer gespeichert

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

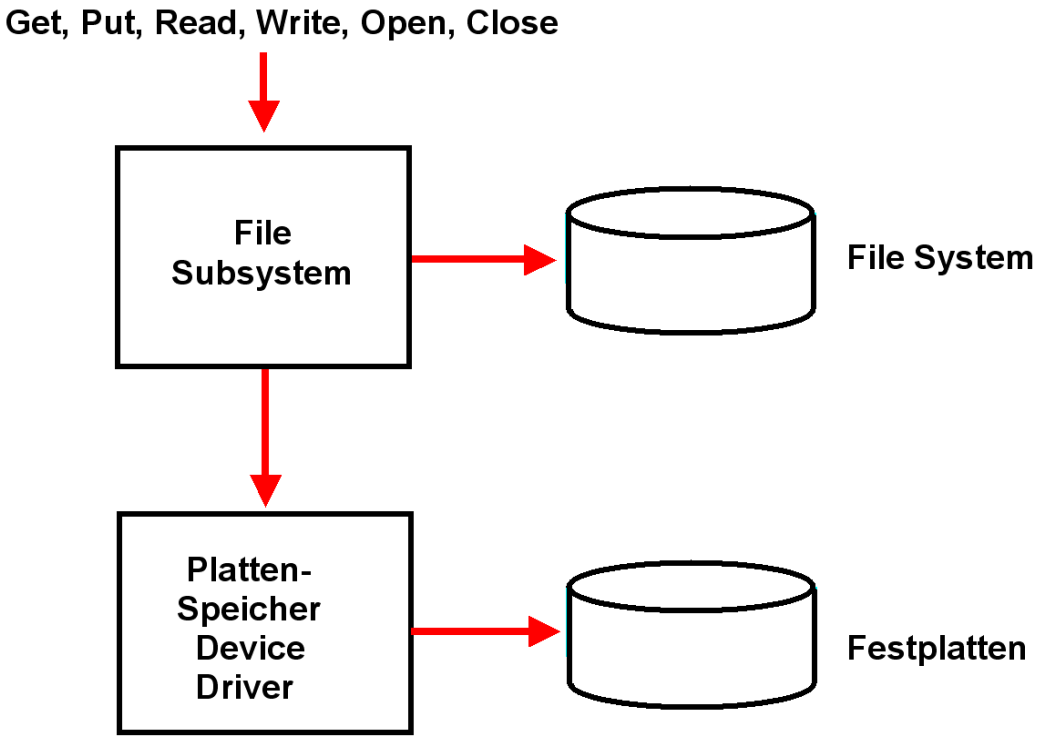

Abb. 6.1.3
Der Überwacher bildet das File System auf den Device Driver ab.

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.

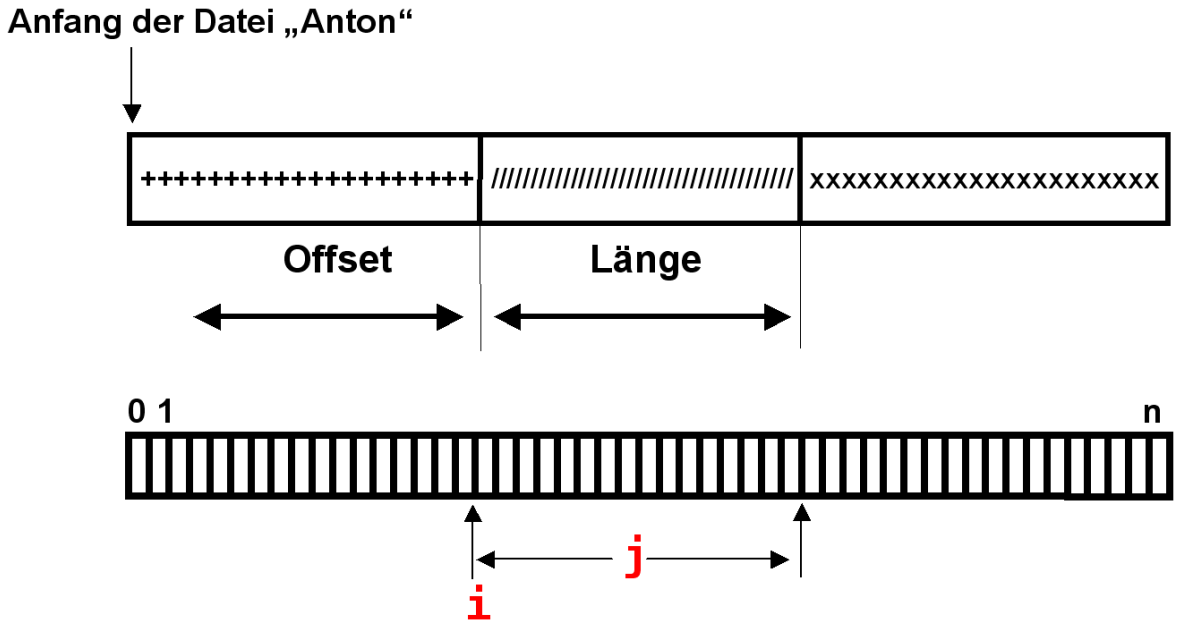

Abb. 6.1.4
Adressierung einer Folge von Bytes mit Hilfe von 2 Zeigern

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.

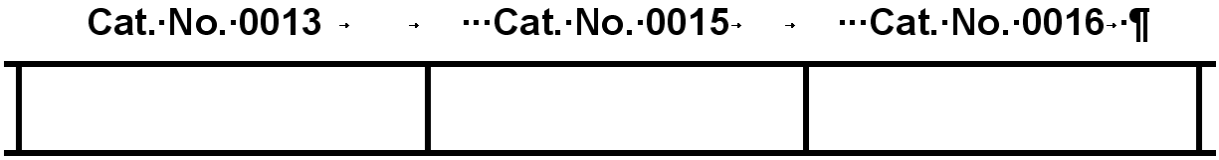

Abb. 6.1.7
Sequentielle Speicherung der Records auf einem Datenträger

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.

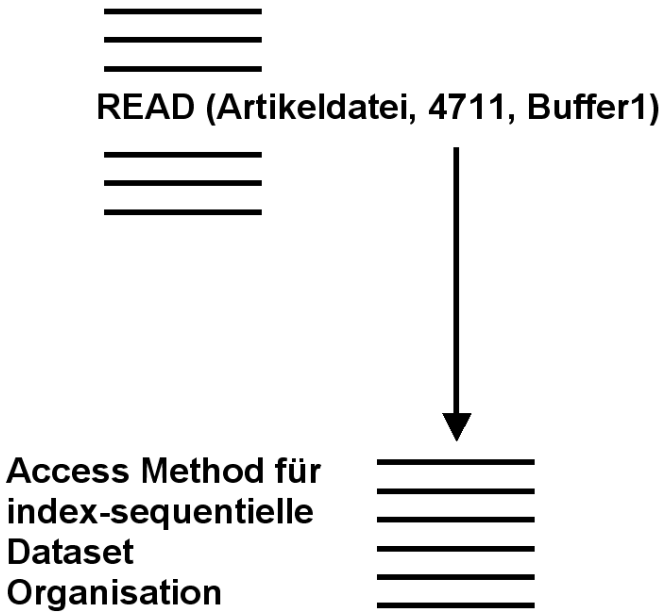

Abb. 6.1.8
Aufruf einer Access Method

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)

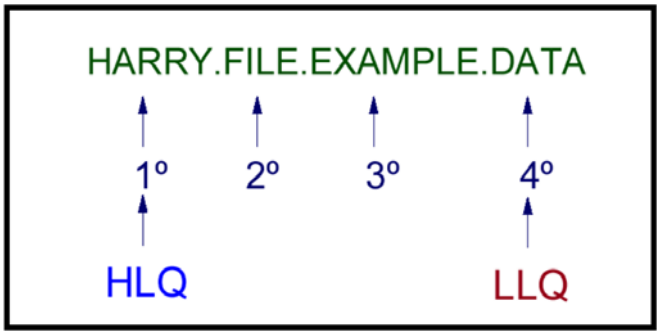

Abb. 6.1.11
High und Low Level Qualifier

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.

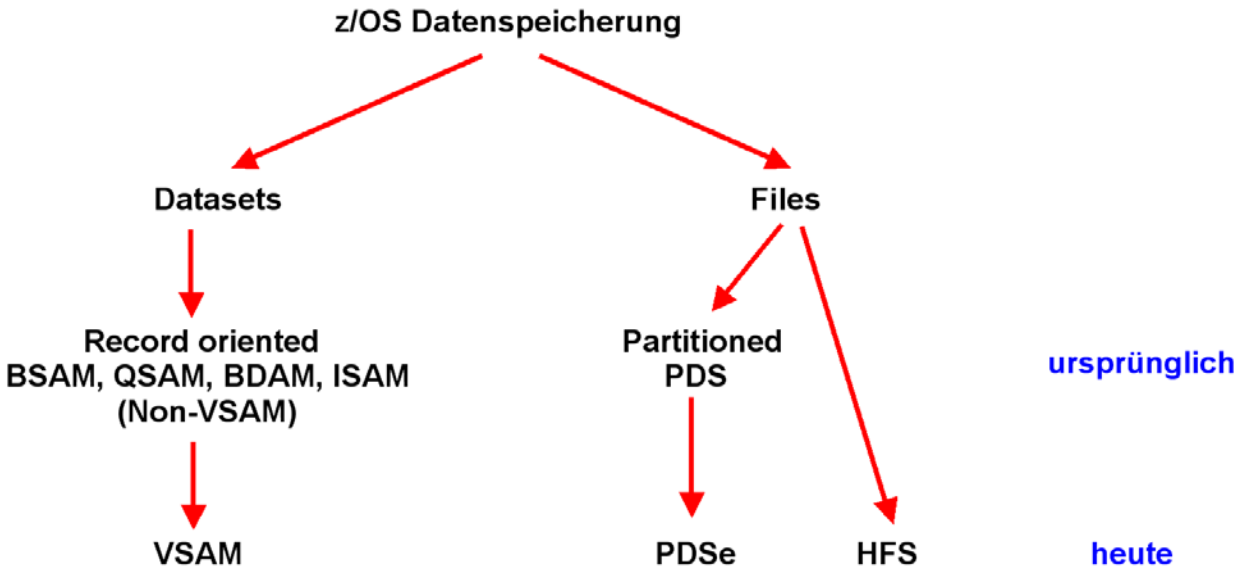

Abb. 6.1.10
Übersicht über die z/OS Datasets und Files

z/OS bezeichnet PDS ebenfalls als einen Dataset, obgleich ein PDS möglicherweise, nicht aber notwendigerweise, aus Records besteht.

6.1.12 Partitioned Dataset (PDS)

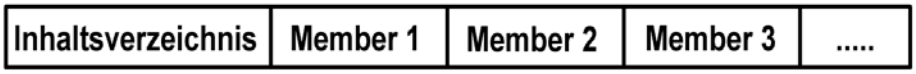

Abb. 6.1.12
Partitioned Dataset (PDS)

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.

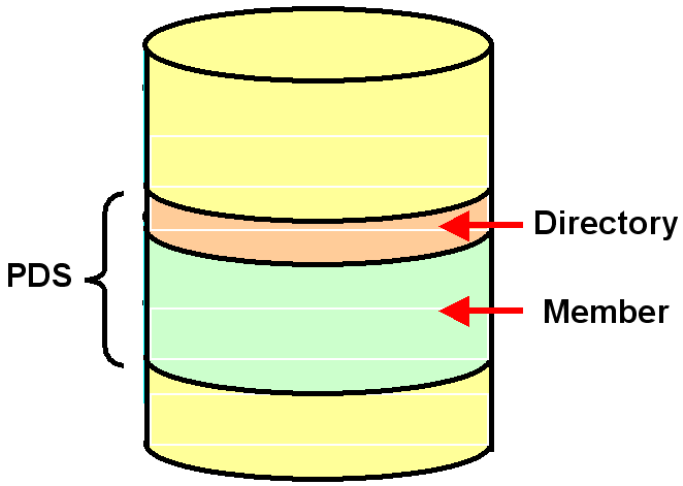

Abb. 6.1.13
Spezifikation für Bibliotheken

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:

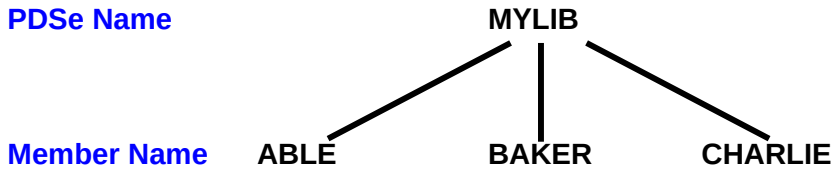

Abb. 6.1.14
PDSe Hierarchie

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.

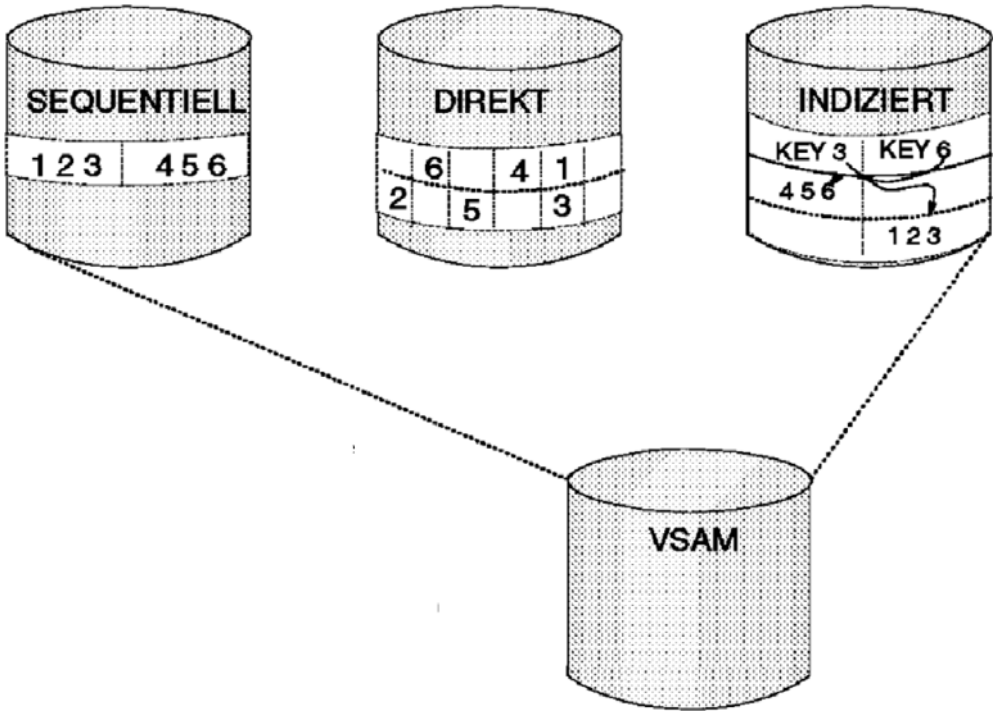

Abb. 6.2.1
Unterschiedliche Arten von VSAM Datasets

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.

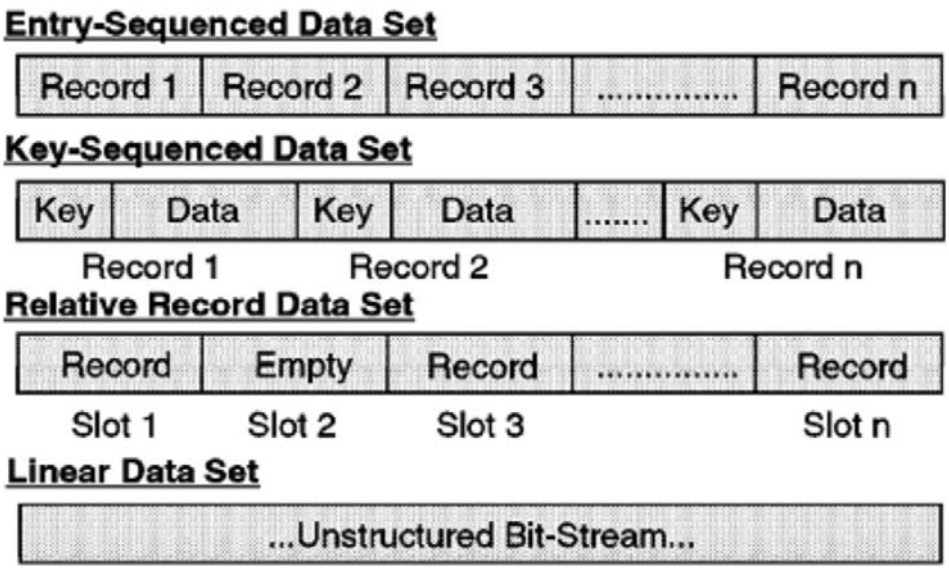

Abb. 6.2.2
Übersicht der VSAM Datasets

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.

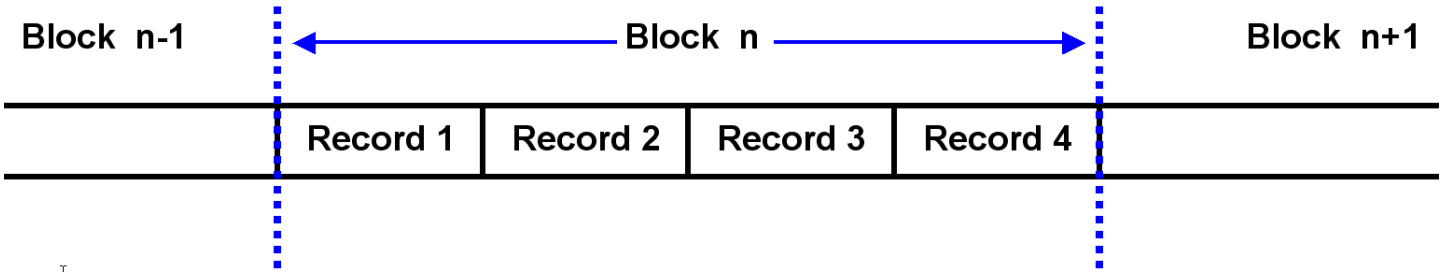

Abb. 6.2.3
Ein Non-VSAM Block bestehend aus mehreren logischen Records

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.

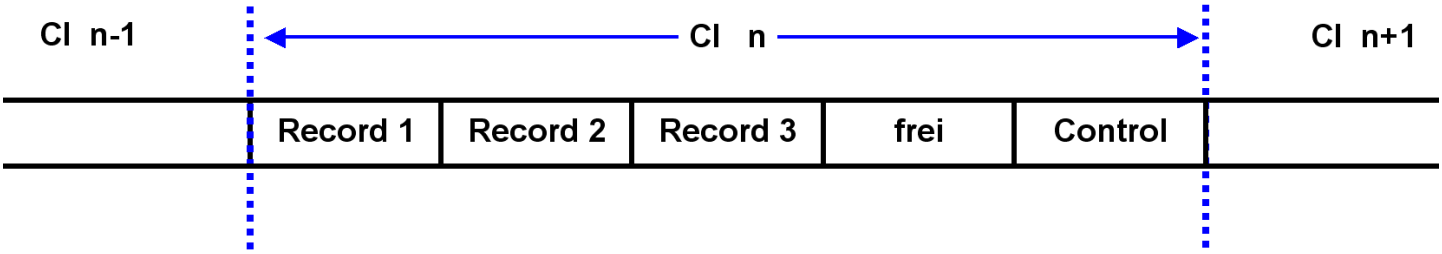

Abb. 6.2.4
Struktur eines Control Intervals

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.

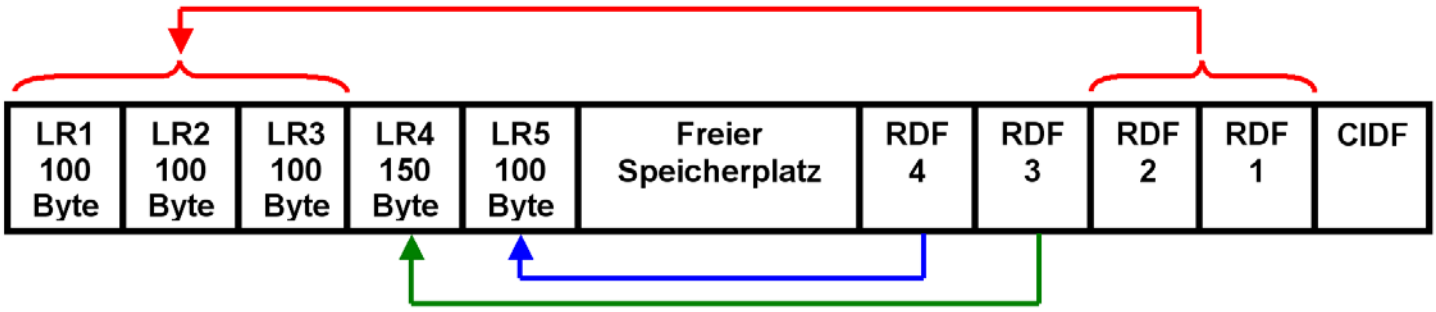

Abb. 6.2.6
Aufeinanderfolgende (contiguous) Records mit identischer Länge

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.

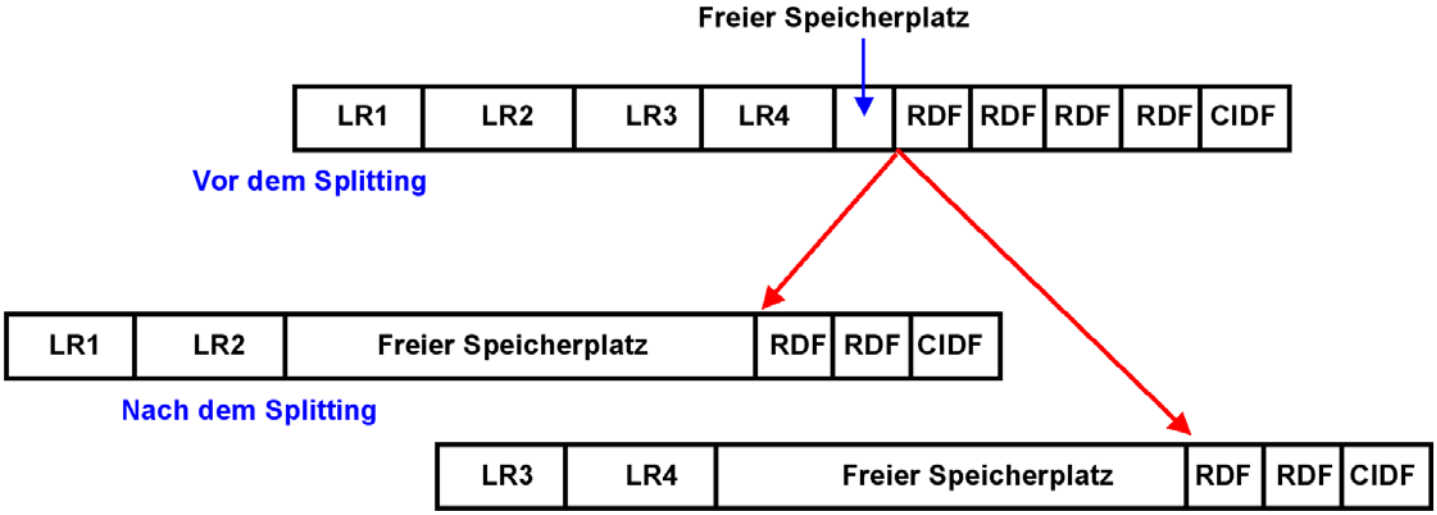

Abb. 6.2.7
Splits

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

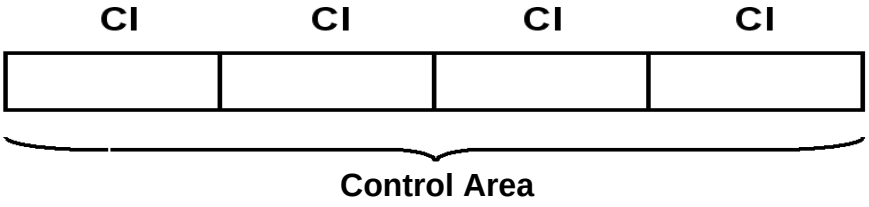

Abb. 6.2.8
Mehrere Control Intervals werden zu einer Control Area zusammengefasst

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.

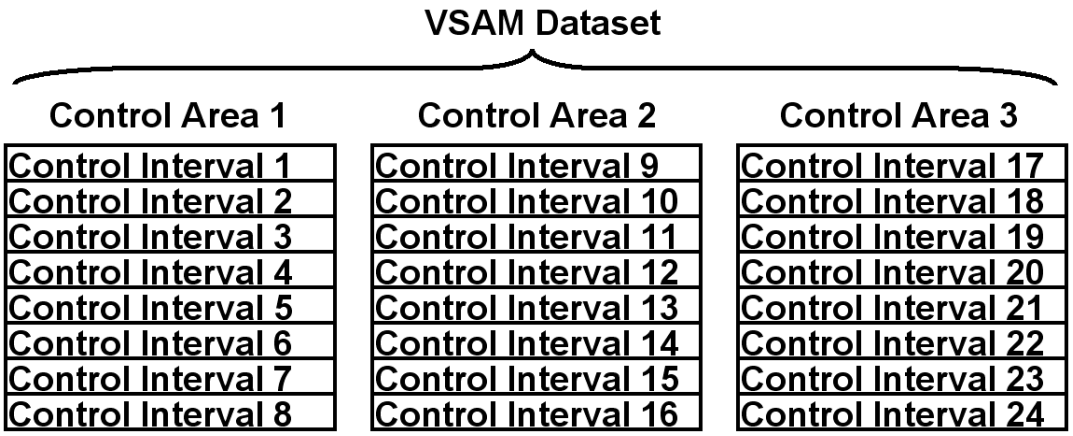

Abb. 6.2.9
Beispiel eines VSAM Data Sets mit 3 Control Areas

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.

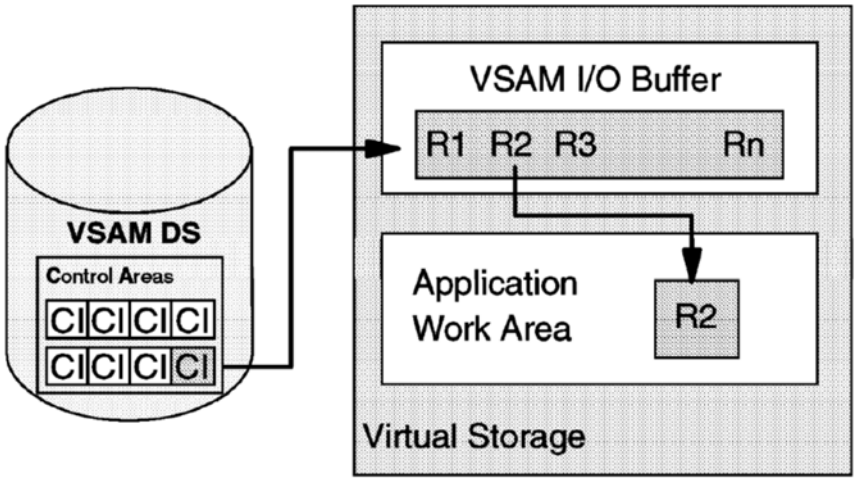

Abb. 6.2.10
Ein VSAM I/O Buffer speichert meistens mehrere Records

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

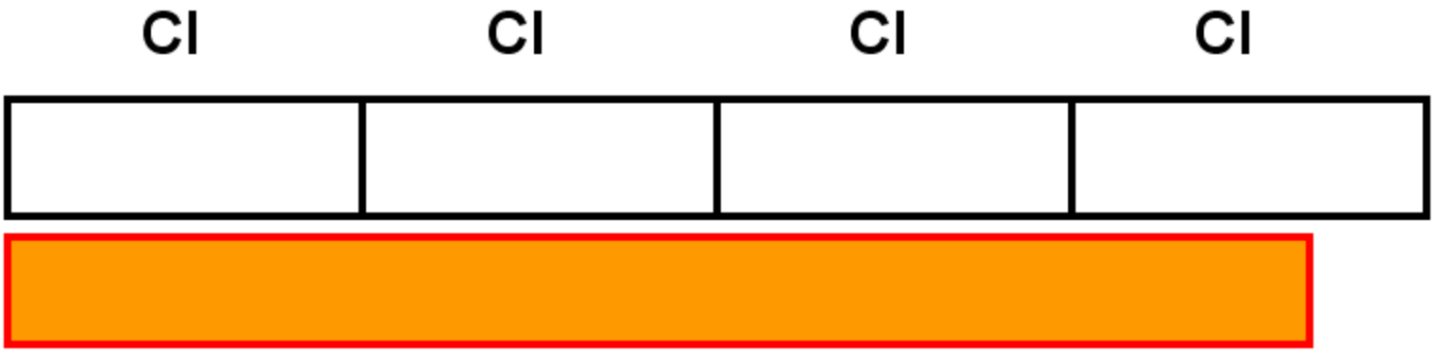

Abb. 6.2.11
Beispiel eines Spanned Records, der sich über 4 Control Intervals erstreckt

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.

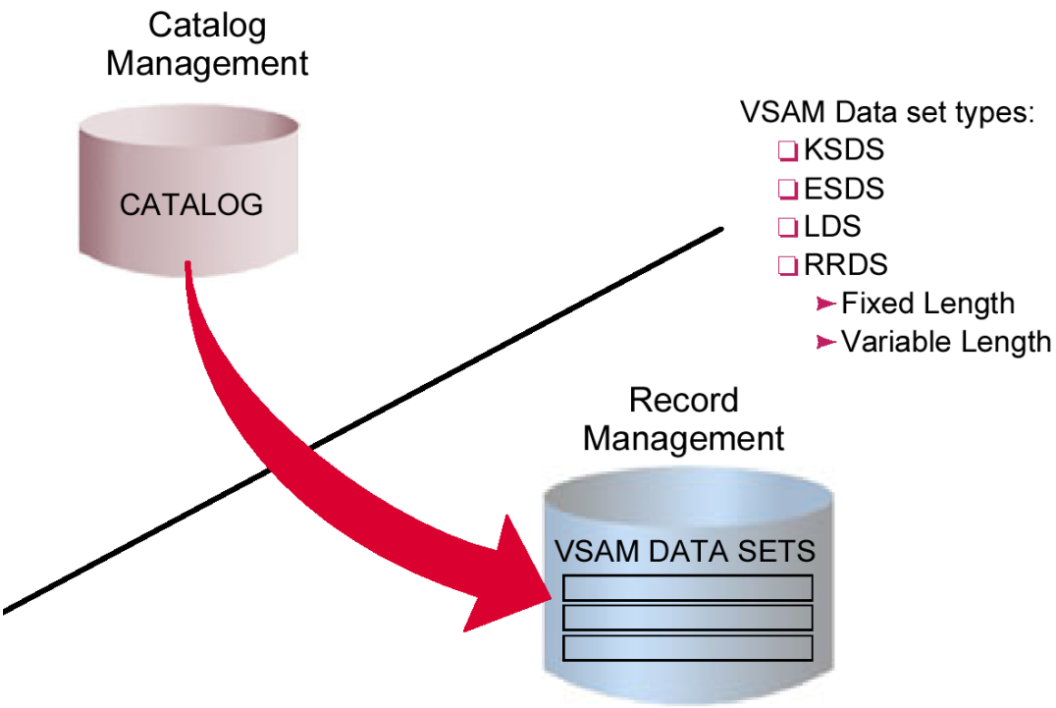

Abb. 6.2.12
Der z/OS Katalog verwaltet auch non-VSAM und PDS Data Sets

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

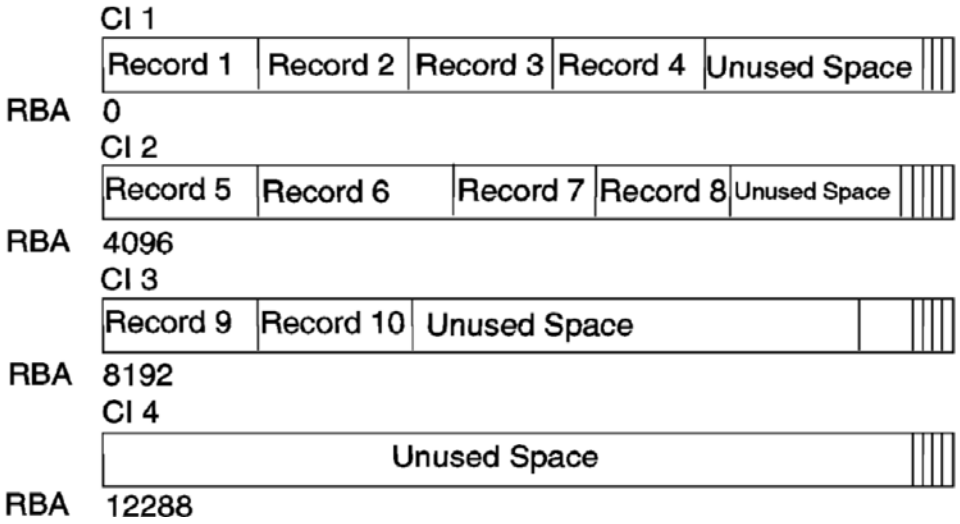

Abb. 6.3.1
Record Anordnung in einem ESDS

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.

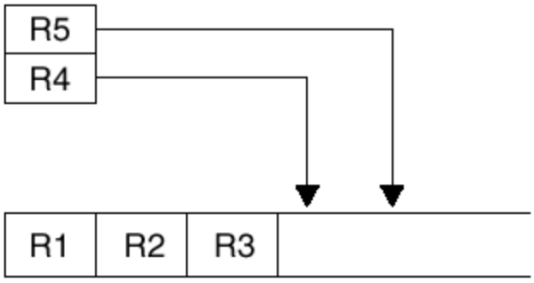

Abb. 6.3.2 ESDS Verarbeitung
Einfügen von neuen Records

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.

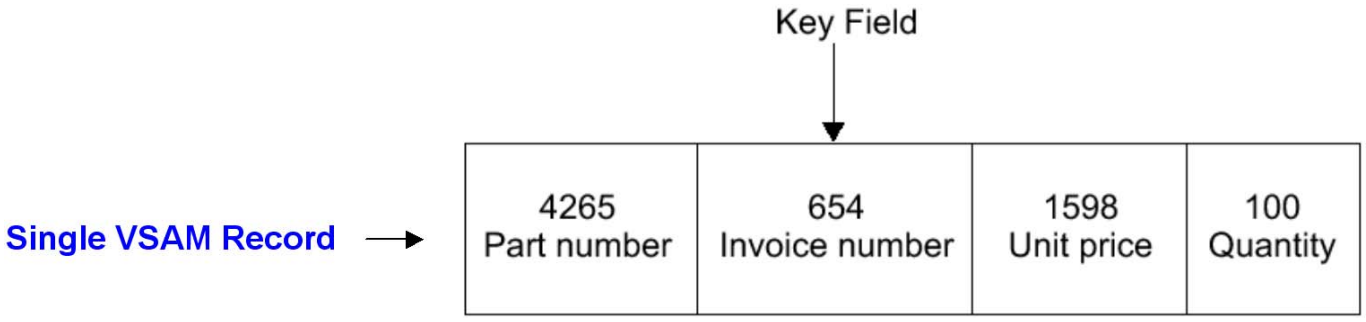

Abb. 6.3.4
Schlüssel Feld (Key Field)

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.

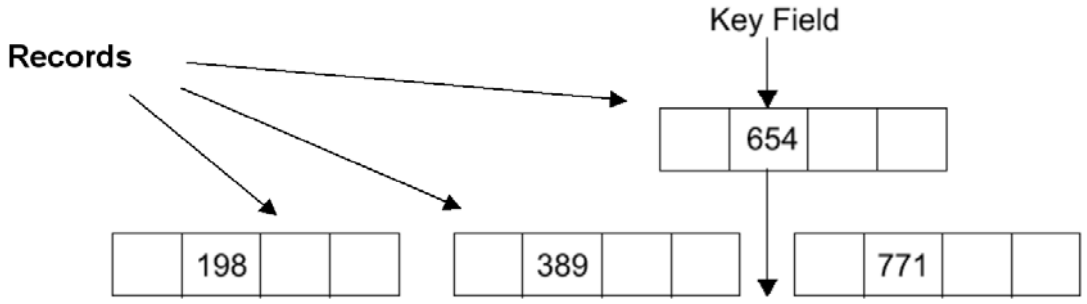

Abb. 6.3.5
Einfügen eines neuen Records

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.

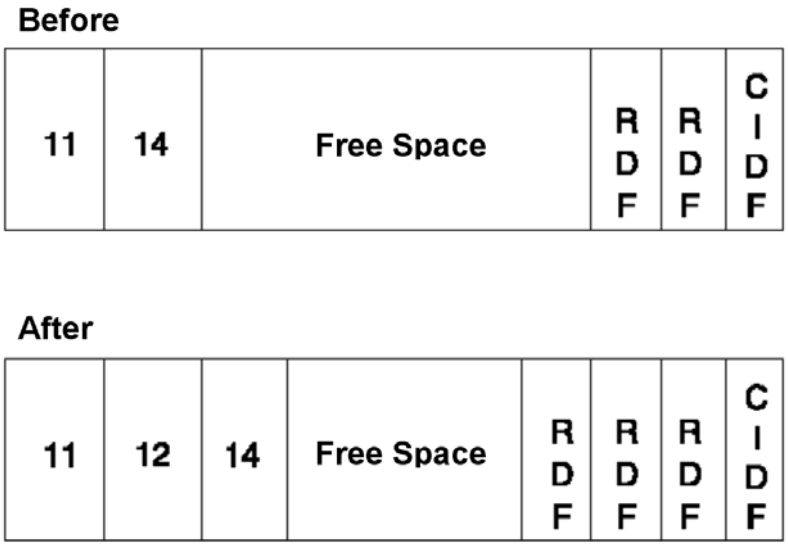

Abb. 6.3.6
Einfügen eines neuen Records

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

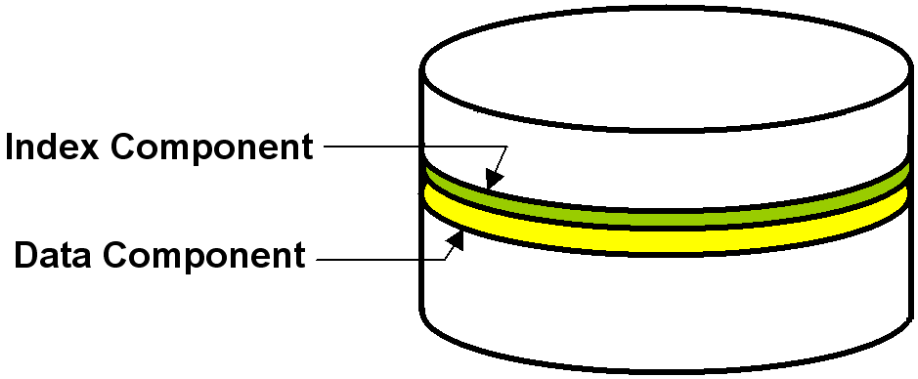

Abb. 6.3.7
VSAM Key Sequenced Dataset

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.

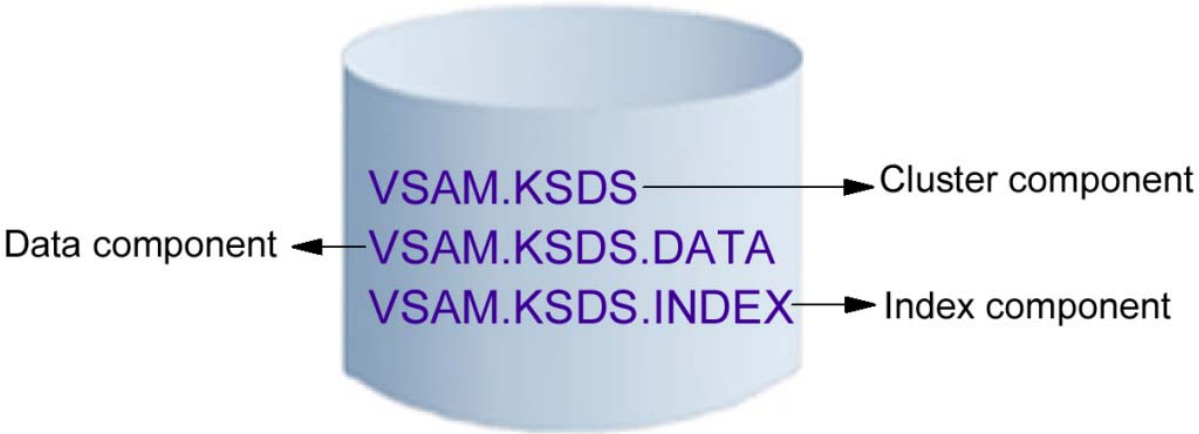

Abb. 6.3.8
Rolle der Cluster Component

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.

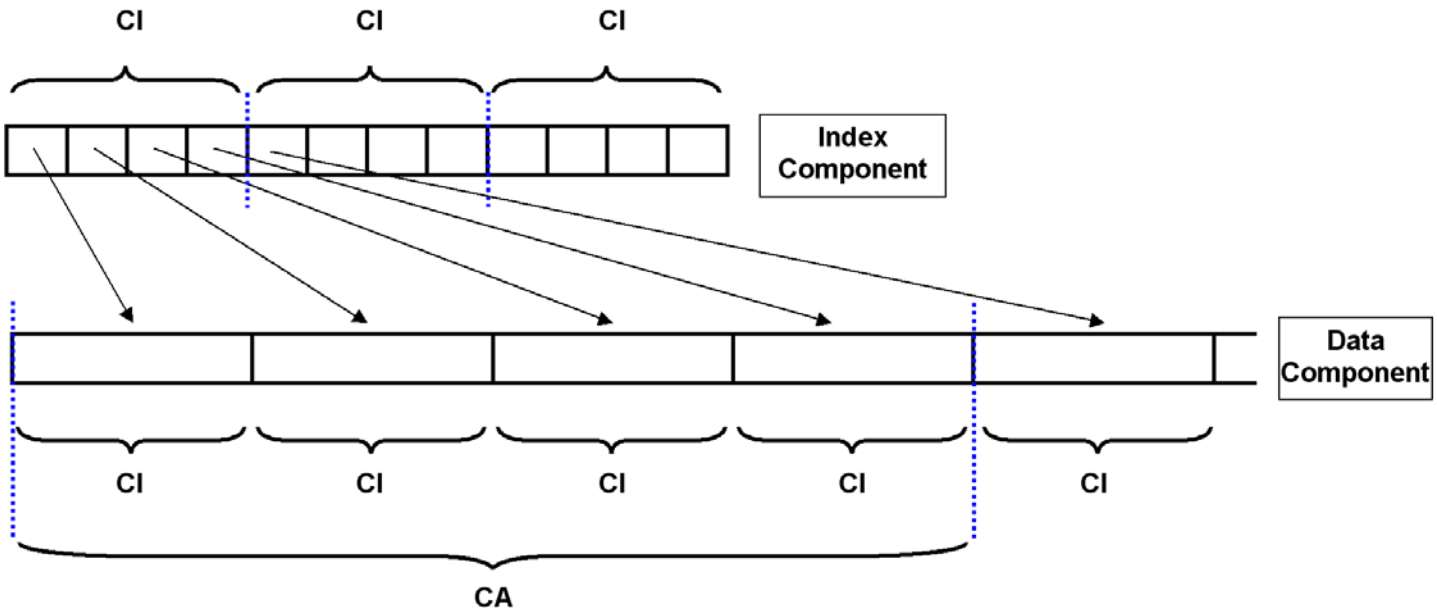

Abb. 6.3.9
Gliederung der Index Component und der Data Component

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.

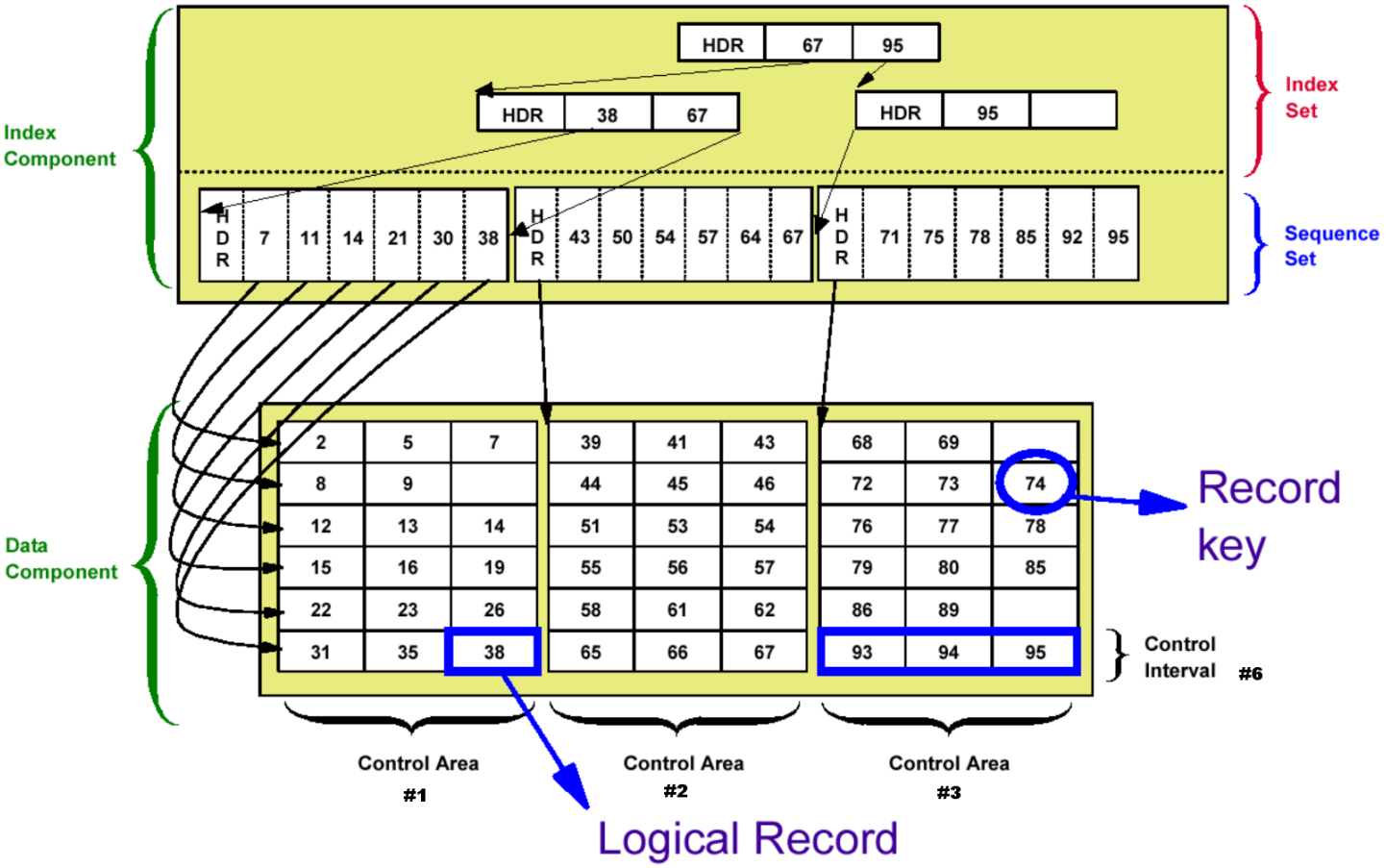

Abb. 6.3.10
Beispiel Index Component und Data Component

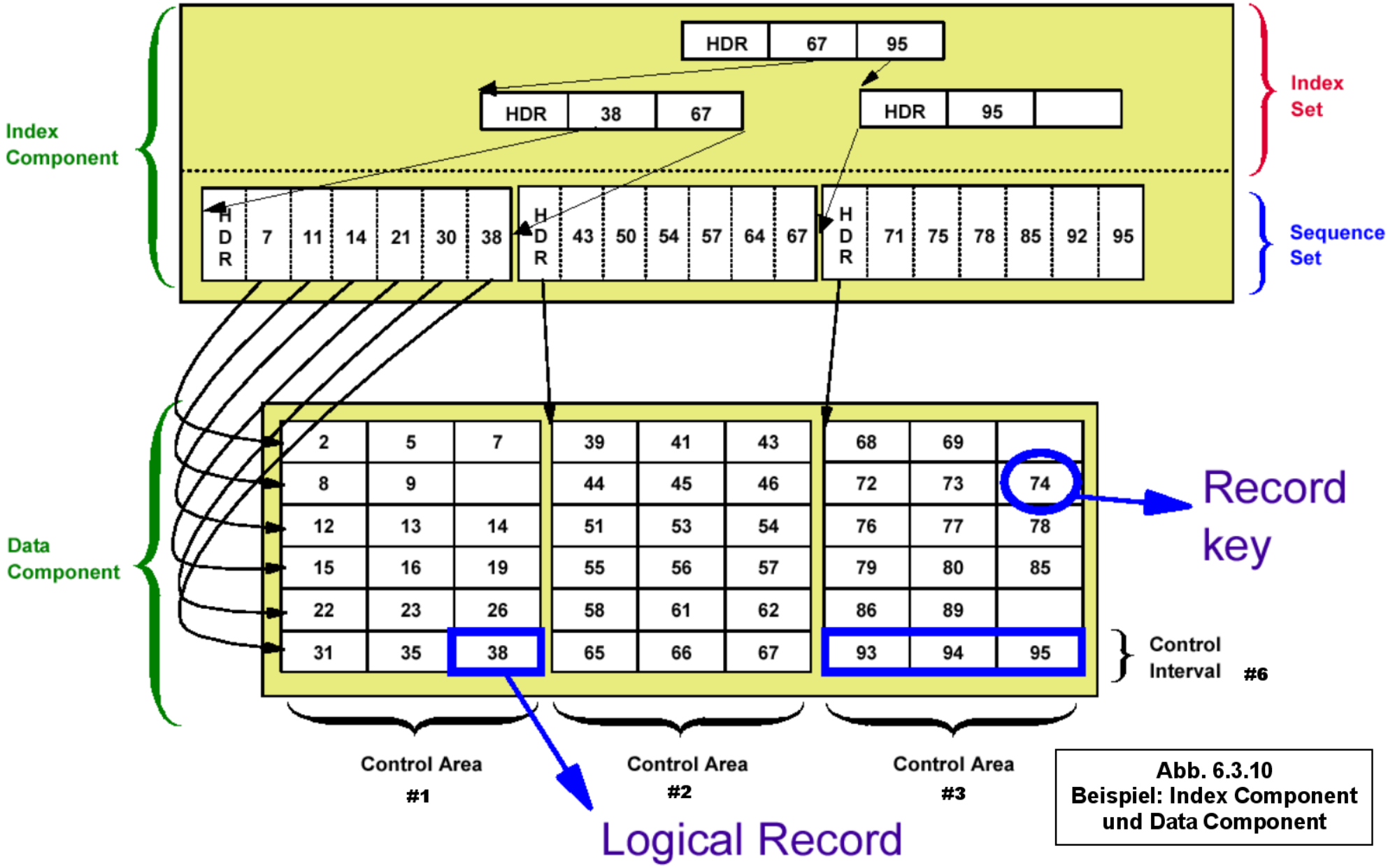

Abb. 6.3.10
Beispiel: Index Component und Data Component

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