UNIVERSITÄT LEIPZIG

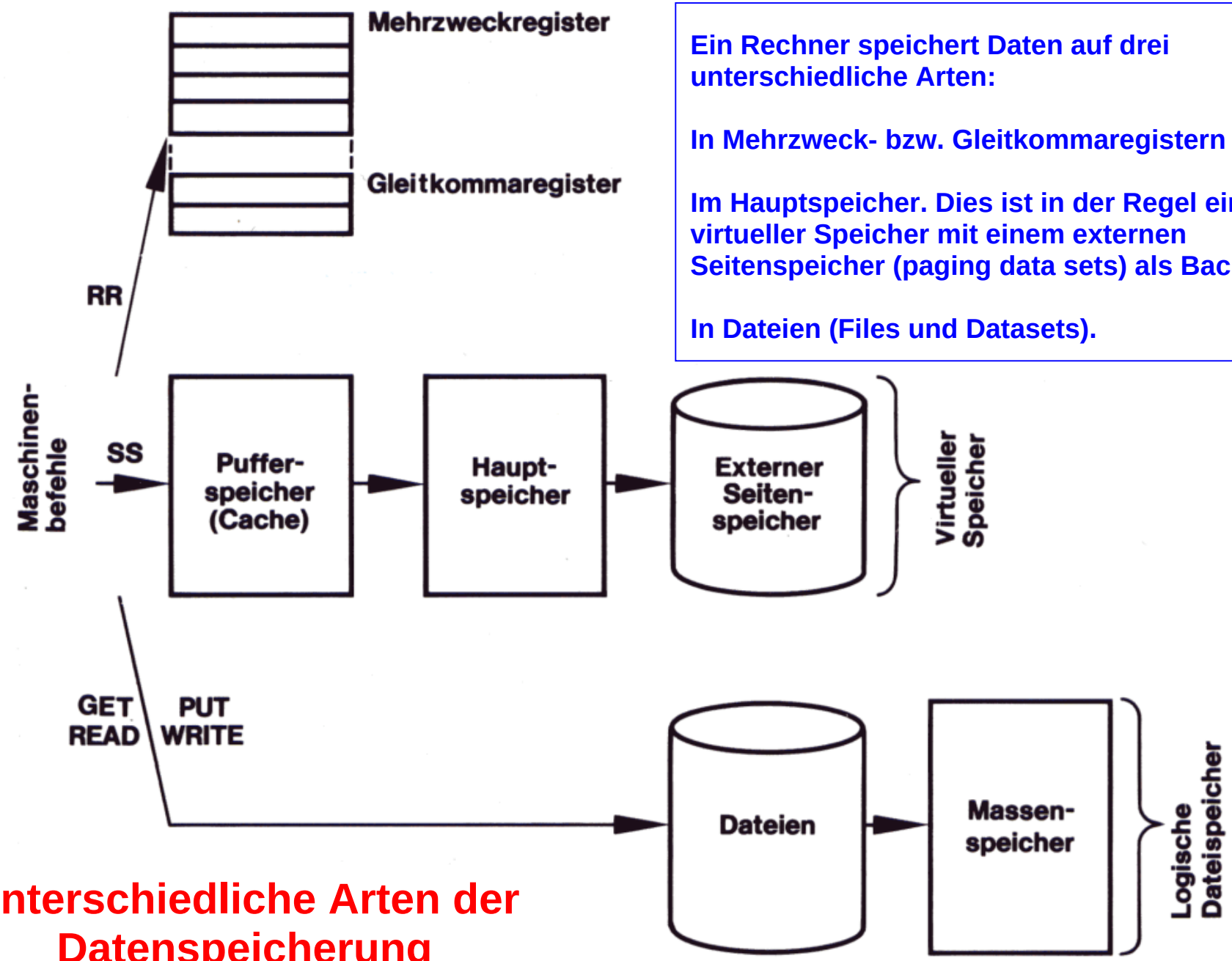
Enterprise Computing
Einführung in das Betriebssystem z/OS

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

WS2012/2013

Datasets Teil 1

Arten von Datasets



Ein Rechner speichert Daten auf drei unterschiedliche Arten:

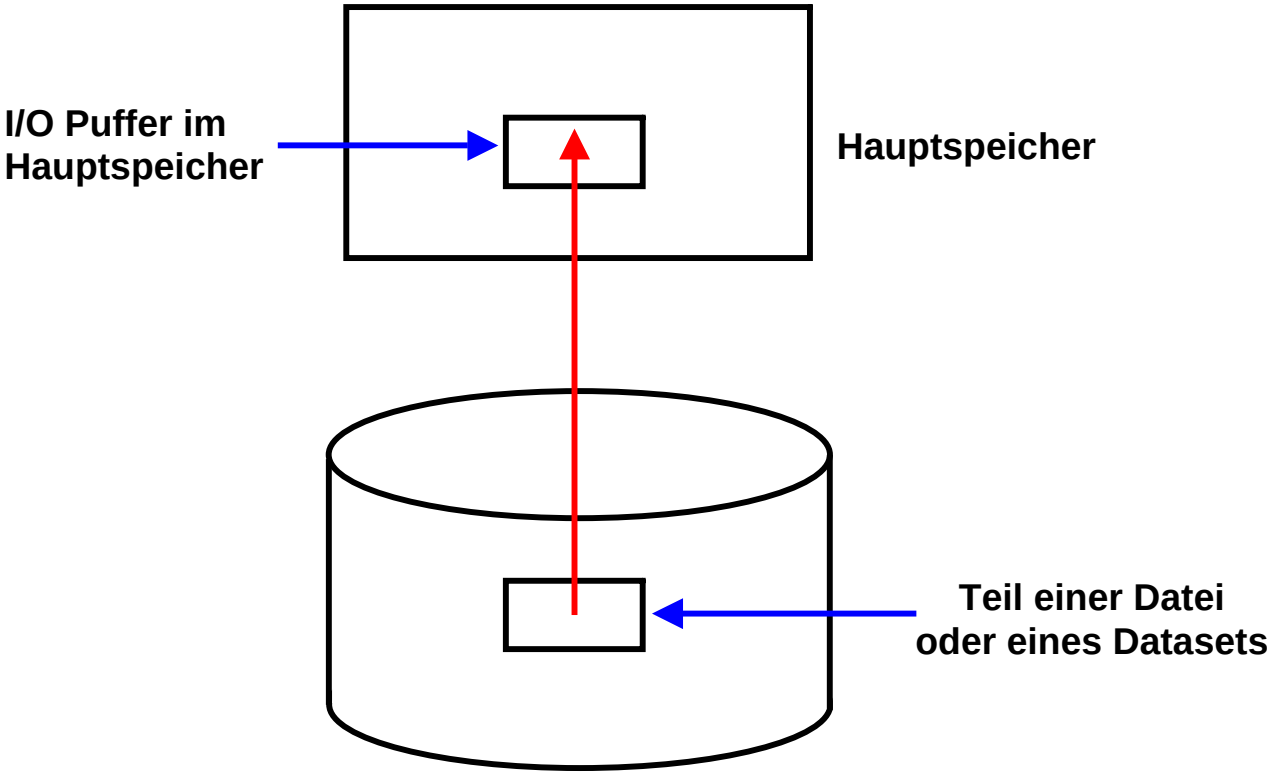
In Mehrzweck- bzw. Gleitkommaregistern

Im Hauptspeicher. Dies ist in der Regel ein virtueller Speicher mit einem externen Seitenspeicher (paging data sets) als Backup.

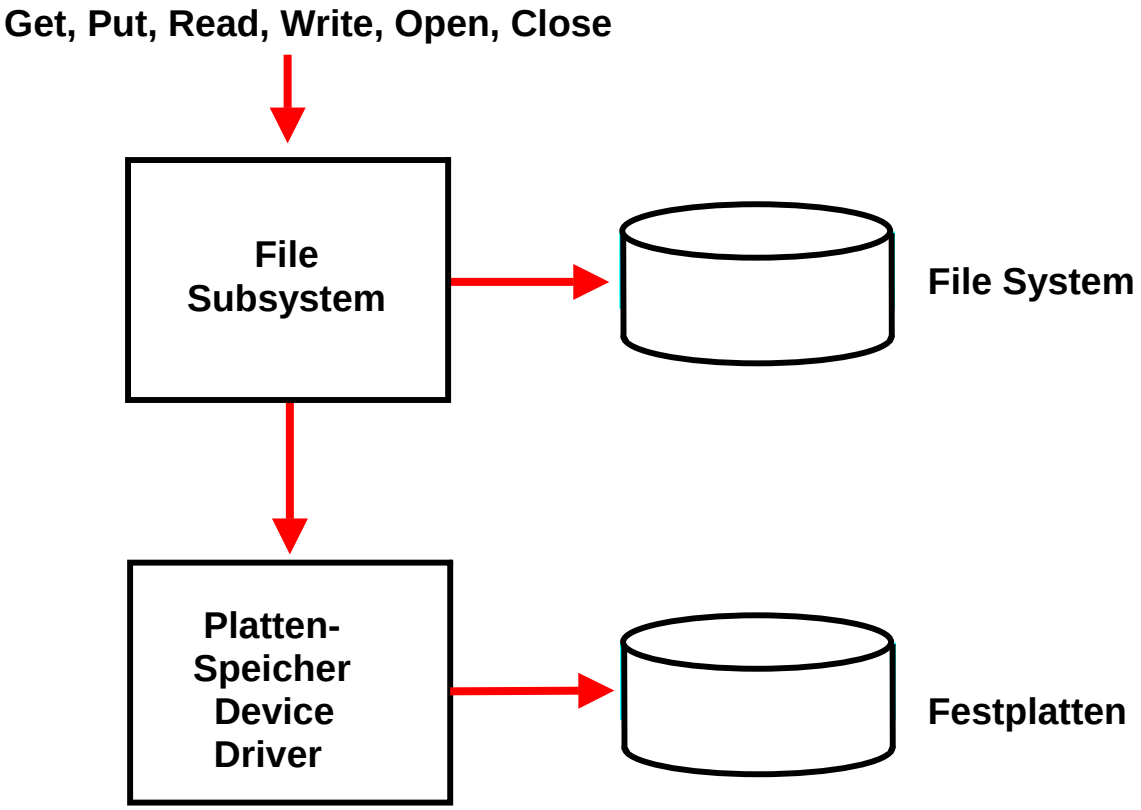
In Dateien (Files und Datasets).

Unterschiedliche Arten der Datenspeicherung

Es kann sich bei einer Datei 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.



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.



Anwendungsprogramme greifen selten oder nie direkt auf einen Plattenspeicher 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.

Plattenspeicher versus File System

Das File Subsystem ist eine Komponente des Überwachers und arbeitet mit der logischen Sicht eines File Systems.

Der Plattenspeicher Device Driver (und/oder verwandte Komponenten) bildet ein oder mehrere File Systeme auf einem oder mehreren Plattenspeichern ab.

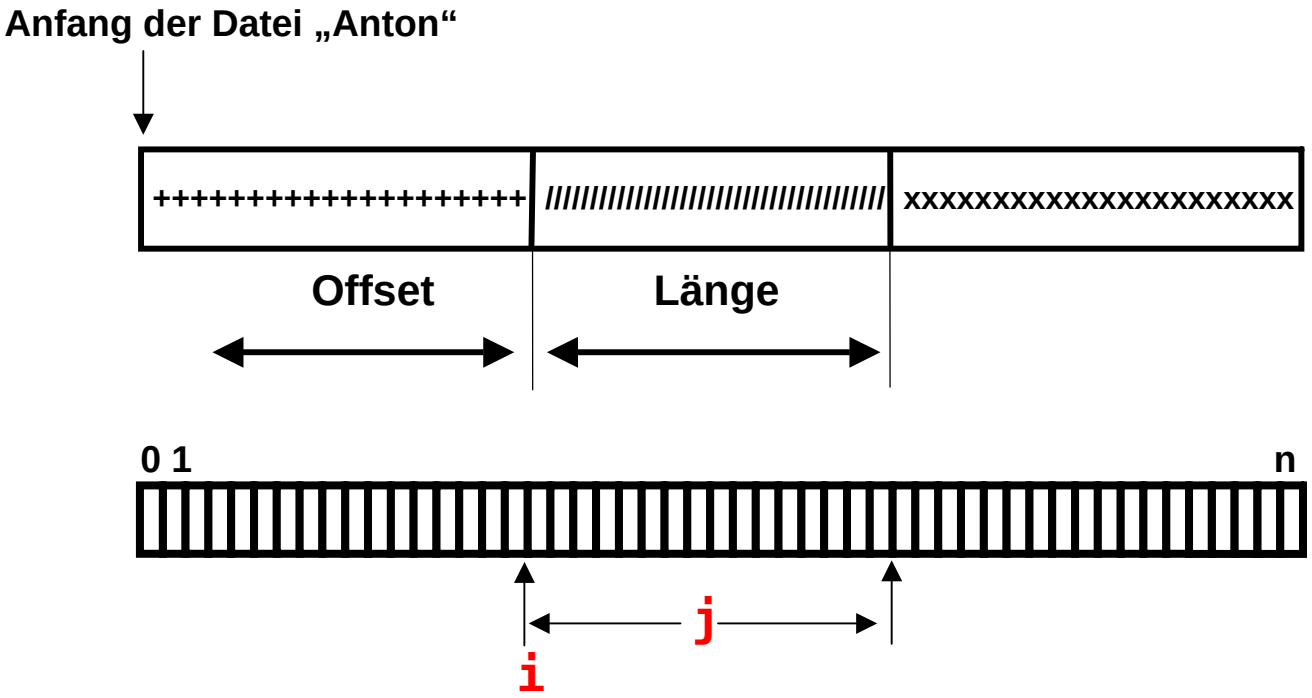
Datenspeicherung unter Linux

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.



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.

Unix Record I/O

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;
}
```

Record

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 name { Komponente_1, ..., Komponente_n };
```

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.

Datenspeicherung unter z/OS

z/OS verwendet zwar 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 Formattierung 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.

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

Die hier dargestellte Gegenüberstellung der Unix files und z/OS Datasets fasst die Unterschiede nochmals zusammen.

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).

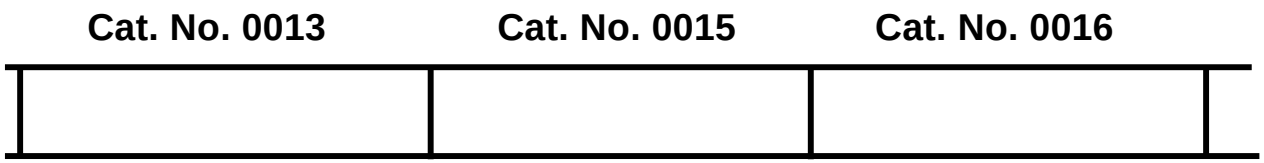
2. indiziert (INDEXED
Nach dem Laden befinden sich die Datensätze in ihrer natürlichen Reihenfolge auf dem Datenträger. Die einzelnen Records in der Datei werden mit unterschiedlichen Schlüsseln gespeichert. Durch eine zusätzlich gespeicherte Indextabelle ist ein direkter Zugriff auf individuelle Records über diesen Schlüssel möglich. Indexsätze werden durch die Organisation (Access Method) automatisch erstellt und verwaltet.

3. Direct - wahlfrei (RANDOM)
Die Records in der Datei sind fortlaufend nummeriert. Der Zugriff auf die Datei erfolgt direkt über die Record-Nummer, die das Anwendungsprogramm kennen muss.

4. Partitioned
Datasets mit einer partitioned Organisation speichern Daten in untergliederten Komponenten, die als „Member“ bezeichnet werden.

```
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);
```

Record Declaration in einem PL/I Programm

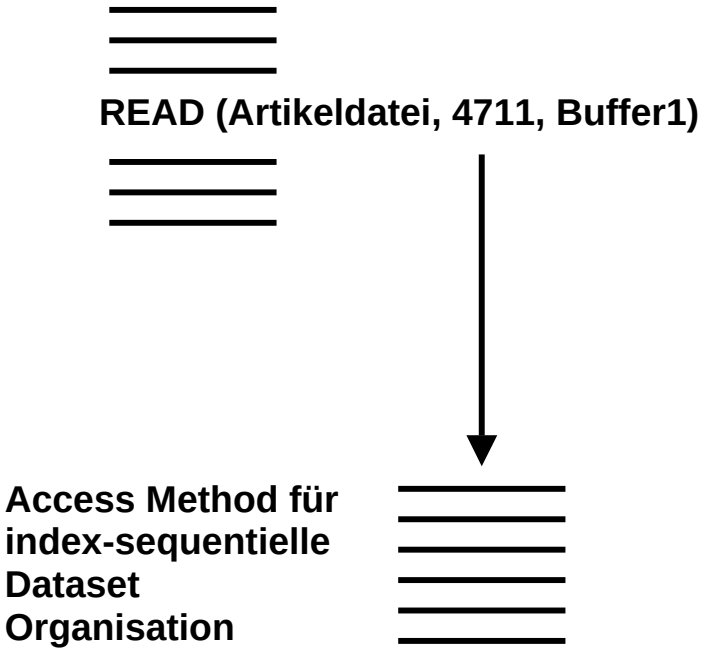


Speicherung der Records auf einem Datenträger

Sequential Dataset Beispiel

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.

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.



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.

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.

In unserem Tutorial 1a haben wir erstmalig einen Dataset allocated. Dort machten wir 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, *
```

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 Thema Input/Output Teil 3.

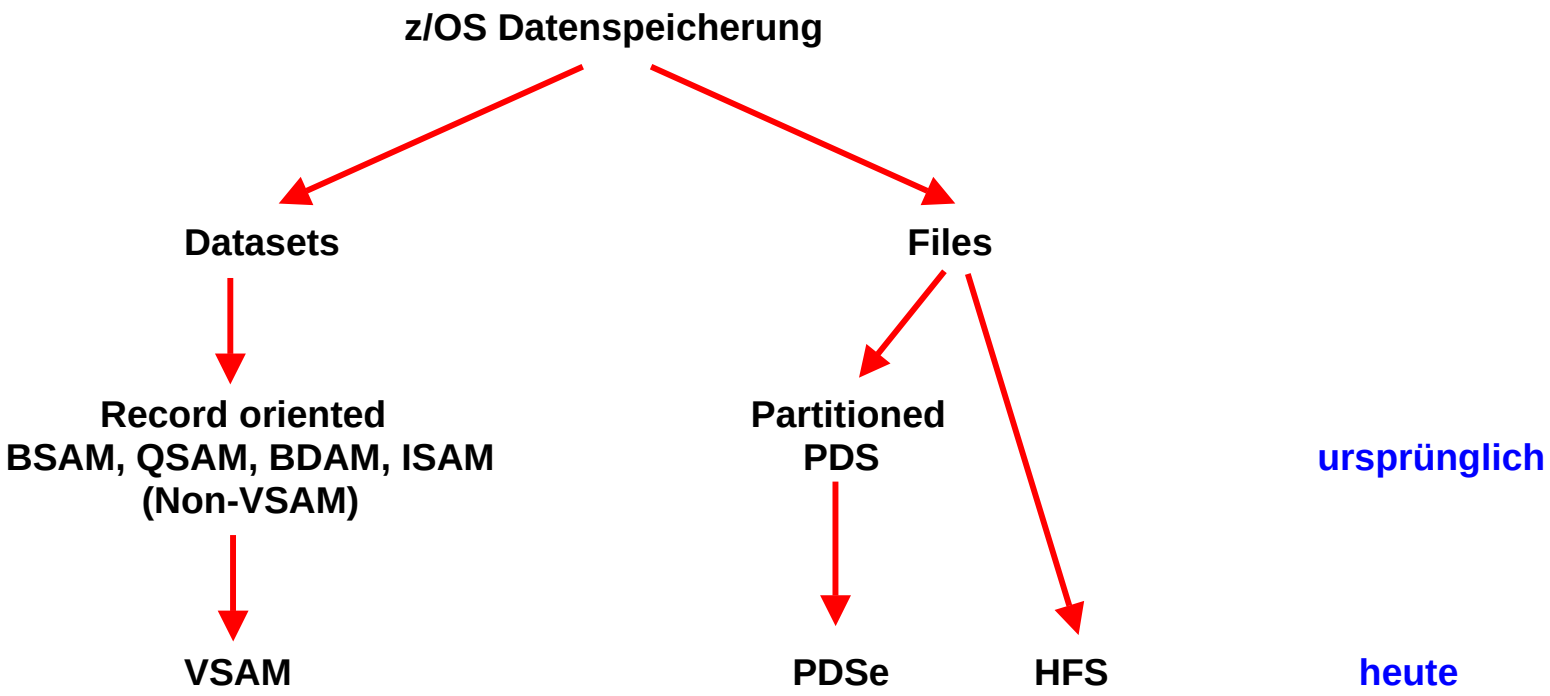
Non-VSAM und VSAM Datasets

Bei der Einführung von OS/360 in den 60er Jahren waren bereits unterschiedliche Record-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, was aber bis heute nicht vollständig erfolgt ist. 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.



Übersicht über die z/OS Datasets und Files

Wir beschreiben die einzelnen Arten von Datasets und fangen mit den Partitioned Datasets an. IBM bezeichnet PDS ebenfalls als einen Dataset, obgleich ein PDS möglicherweise, nicht aber notwendigerweise, aus Records besteht.

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:

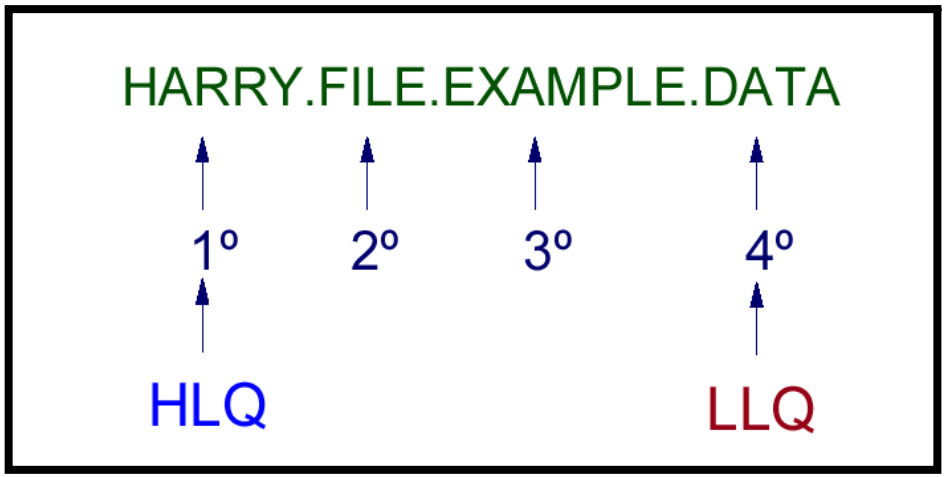
USER1.TEST.DATA
SYS1.PARMLIB
MAX.PROGRAM.VERSION1

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)



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.

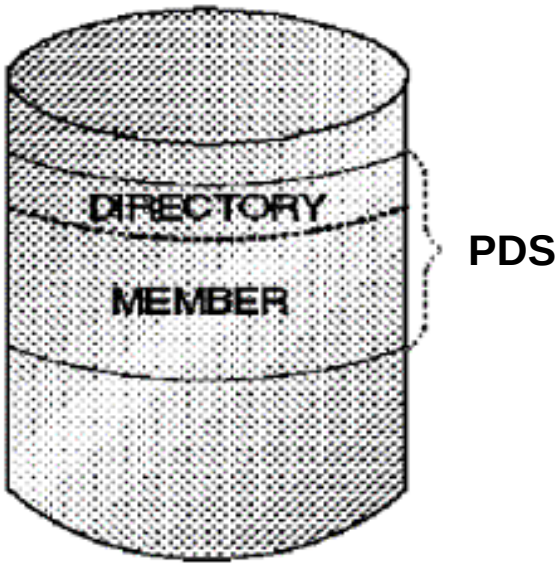
Partitioned Dataset (PDS)

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.



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.

Partitioned Organisation – Nutzung

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 Plattenspeicher für CCC. z/OS geht zu dieser Adresse, lädt CCC in den Hauptspeicher und ruft anschließend CCC auf.

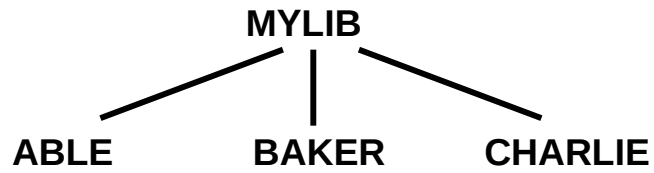
Speicherplatz der für z/OS Datasets allocated wird, beginnt immer am Anfang einer Festplattenspur (Track). Mit Hilfe von PDS kann mehr als eine Datei auf einer Spur gespeichert werden. Hiermit kann Festplatten-Speicherplatz optimal genutzt werden, wenn viele Files vorhanden sind, die alle wesentlich 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.

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



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.

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 Data Sets 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.