Vorlesung Implementierung von Datenbanksystemen

2. Dateiverwaltung

Prof. Dr. Klaus Meyer-Wegener

Wintersemester 2019/20

Lehrstuhl für Informatik 6 (Datenmanagement)
Friedrich-Alexander-Universität Erlangen-Nürnberg

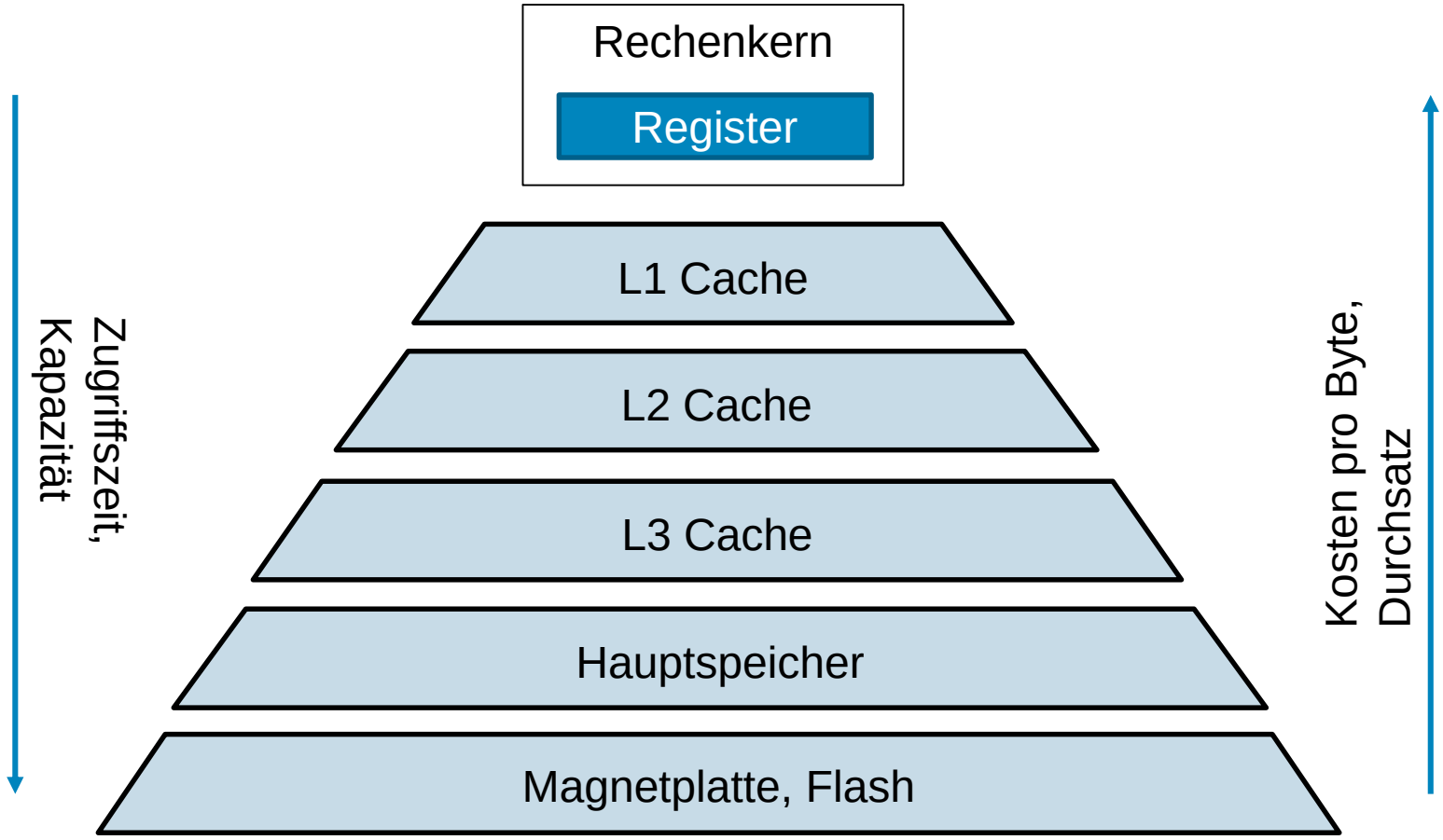
2. Dateiverwaltung

- **Basis-Datenverwaltungsdienst des Betriebssystems**
  - Grundlage für Datenbanksysteme
- **Schichtenbildung:**
  - Ganz unten: **physische Speichergeräte** (Hardware)
  - Darüber: **"logische" Speichergeräte** (mit verbesserter Störsicherheit)
  - Und: **Dateien**
- **Nichtflüchtig ("non-volatile"):**
  - Datenträger, rein passiv, dauerhafte Speicherung auch ohne Strom
- **(Hintergrund-) Speichergeräte ("storage"):**
  - Platzierung und Lokalisierung der Daten mittels Zugriffsmechanismen
  - Linear: Magnetband
  - Wahlfrei: für Datenverwaltung weiterhin am wichtigsten, **Magnetplattenspeicher**

2.1 Physische Speichergeräte

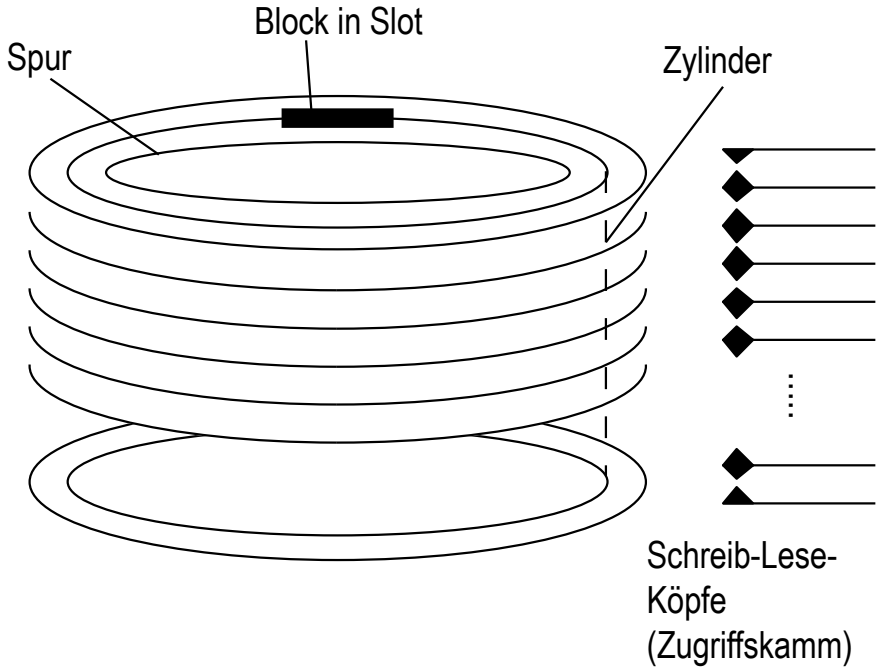
- **Magnetplattenspeicher**
  - Feste Struktur: Zylinder, Spuren, Slots (Sektoren)
  - Feste Größe
  - **Wahlfreier Zugriff** auf einen Slot-Inhalt (= Block): ~ 5 ms
  - Asynchrone Ein-/Ausgabe möglich
- **Magnetbandspeicher**
  - Kostengünstig, **sequenzieller Zugriff**, Sicherung und Archivierung
- **Alternative Speicher**
  - Optische Speicher (DVD, …), MEMS, Flash / SSDs, Holographie, …
    - Forschungsgegenstand
  - Am interessantesten wohl: **Flash**
    - Begrenzte Lebensdauer
    - Schreiben sehr viel langsamer als Lesen
  - Und: **Hauptspeicher** (Arbeitsspeicher, DRAM, "memory")
    - Seit neuestem (2019) sogar nichtflüchtig (NVM)

Speicherhierarchie



Magnetplattenspeicher

- **Schematischer Aufbau:**



- **Block:**
  - Datenmenge, Bytefolge
  - Elementare Einheit des Transports zwischen Platte und Hauptspeicher
- **Slot (Sektor):**
  - Platz auf einer Spur, der zur Aufnahme eines Blocks vorgesehen ist

2.2 Logische Speichergeräte

- **Erhöhung der Störsicherheit**
  - Verdecken von Gerätefehlern
    - Z.B. bei Paritätsfehler nochmal lesen
      und bei Schreibfehler nochmal schreiben
- **Einzelheiten dieser E/A-Prozeduren verdecken**
- **Satz von E/A-Prozeduren bereithalten, parametrisieren**
  - Verwaltung und Aufruf durch eigene Instanz: **Gerätetreiber**
    - E/A-Prozeduren gerätespezifisch, daher für jedes Gerät eigener Treiber
- **Außerdem: flexible Zuordnung von logischen Geräten (Laufwerken) zu den physischen**
  - Mehrere physische Geräte
    als ein großes logisches anbieten (z.B. RAID)
  - Physisches Laufwerk (bzw. Datenträger)
    in mehrere logische Platten aufteilen ("partitionieren")

Logische Speichergeräte (2)

- **Typische Lese- und Schreiboperation ("Dienste"):**

```
int Device::readBlock (
   int CylinderNo, int TrackNo, int SlotNo,
   char *BlockBuffer )
```

  - Rückgabewert: Quittung, Status
  - Speicherplatz für gelesenen Block (Puffer) muss vom Aufrufer vorher angelegt worden sein

```
int Device::writeBlock (
   int CylinderNo, int TrackNo, int SlotNo,
   char *BlockBuffer )
```

- Das heißt: **physische Adressierung** der Slots

Logische Speichergeräte (3)

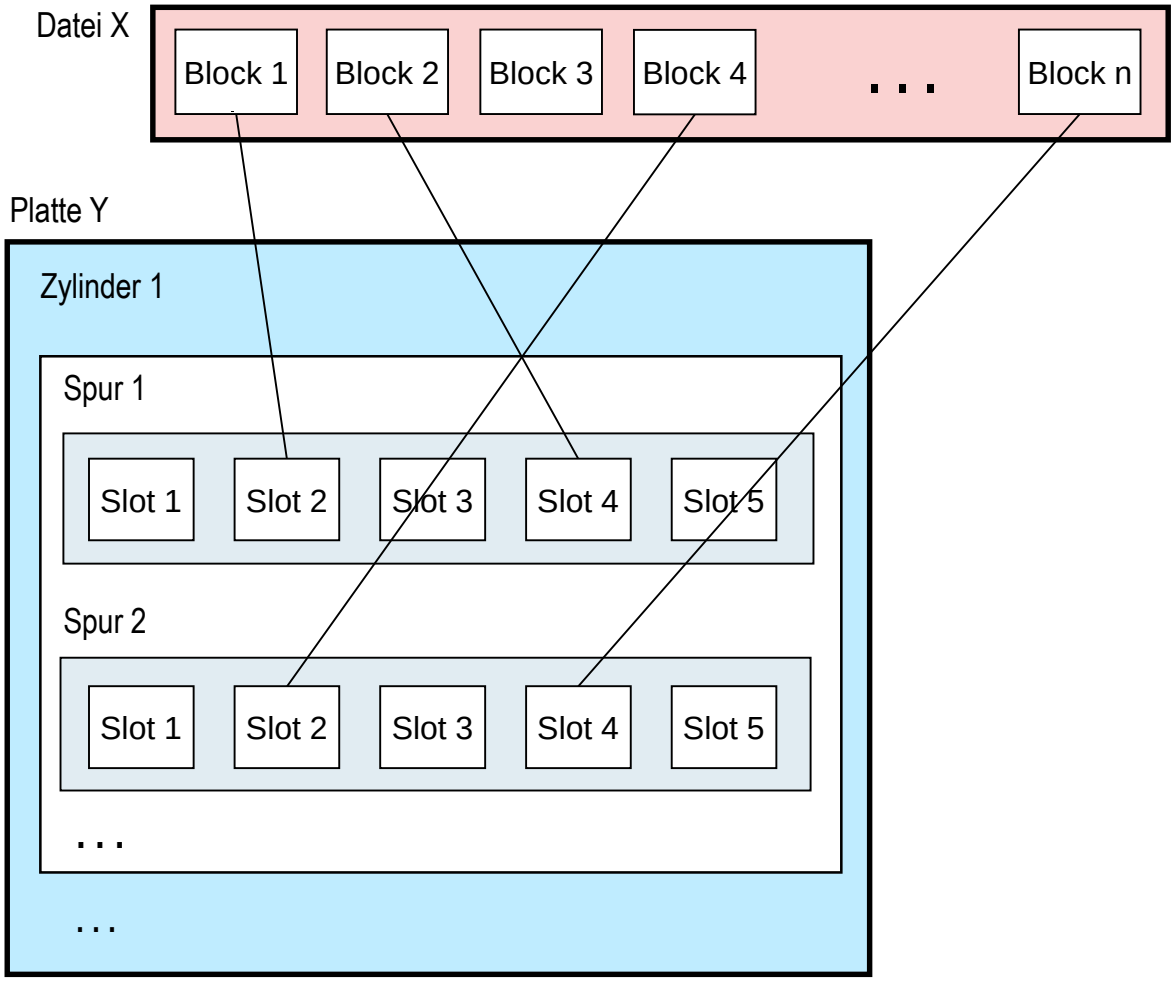
- **Annahme:**
  - Anwendungsprogramme arbeiten direkt mit dieser Ein-/Ausgabeschnittstelle
    - Frühzeit der Datenverarbeitung
- **Vorteile:**
  - Schnell
  - Maximale Speicherausnutzung – keine Verwaltungsdaten
- **Nachteile:**
  - Kein Schutz:
    - Alle Blöcke der Platte von allen Anwendungsprogrammen lesbar und schreibbar
  - Wissen über die Zuordnung der Blöcke (ggf. auch ganzer Spuren oder Zylinder) zu den Anwendungsprogrammen
    - nur außerhalb des Systems (in den Köpfen der Programmierer)
  - Reservierung zusätzlichen Speicherplatzes
    - nur durch Absprache unter den Programmierern
  - Zahlreiche Fehlermöglichkeiten
    - Z.B. Überschreiben eines vermeintlich freien Blocks
  - Oft starre Aufteilung einer Platte mit ungenutztem Platz; schwierig zu ändern

2.3 Dateiverwaltung eines Betriebssystems

- **Einrichten einer Indirektion**
  - zwischen Anwendungsprogramm und logischem Gerät
- **Verwaltung von Dateien:**
  - Haben einen **Namen**
  - **Folge** von Blöcken (linear)
    - Direkt ansprechbar über **laufende Nummer**
    - Auf der Platte aber nicht notwendigerweise sequenziell abgelegt
  - **Dynamisch erweiterbar** um zusätzliche Blöcke
- **Abstraktion**
  - Verborgen: Größe der Magnetplatte; Zylinder und Spuren

(Siehe auch Vorl. Systemprogrammierung 2)

Dateikonzept



Zugriffsoperationen (Dienste)

- **Sog. blockorientierte Zugriffsmethode ("Access Method") für eine Datei**
  - z.B. UPAM im BS2000 von Siemens, BDAM im MVS (heute: z/OS) von IBM
- **Gekennzeichnet durch Operationen wie die folgenden:**

```
BlockFile::BlockFile ( char *FileName, char Mode,
                       int *BlockSize )
```

  - **öffnet die Datei** mit dem angegebenen Namen
    und richtet eine temporäre Verwaltungsdatenstruktur für die Arbeit mit ihr ein ("Dateikontrollblock")
  - **Mode** gibt an, ob Datei gelesen, geschrieben oder ergänzt werden soll
  - Blockgröße definiert bei neuer Datei die Blockgröße
    oder liefert bei vorhandener Datei die verwendete Blockgröße zurück

```
BlockFile::~BlockFile ()
```

  - **schließt die Datei** und löscht den Dateikontrollblock

Schreib- und Leseoperationen


```
int BlockFile::append ( int NumberOfBlocks )
```

- erweitert Datei um die angegebene Zahl von Blöcken
- Rückgabewert gibt an, wie viele Blöcke tatsächlich angelegt werden konnten (bei Fehler: **0**).

```
int BlockFile::write ( int BlockNo, char *BlockBuffer )
```

- überschreibt den angegebenen Block mit dem im Blockpuffer bereitgestellten neuen Inhalt
- Rückgabewert ist ein Fehlercode, der z.B. anzeigt, dass die Blocknummer ungültig ist.

```
int BlockFile::read ( int BlockNo, char *BlockBuffer )
```

- liest den angegebenen Block der Datei in den Blockpuffer

Weitere Operationen

`int` **`BlockFile::size`** `()`

- gibt die aktuelle Größe der Datei in Blöcken an

`void` **`BlockFile::drop`** `( int` **`NumberOfBlocks`** `)`

- gibt angegebene Zahl von Blöcken am Ende (!) der Datei wieder frei
- Inverse Operation zu **append**

Operation zum Freigeben beliebiger Blöcke?

- Denkbar, aber in der Realisierung sehr aufwändig
- Außerdem verschieben sich alle Blocknummern hinter den gelöschten, was leicht zum Zugriff auf einen falschen Block führen kann!

Dateikatalog

- **Verwaltungsdatenstruktur**
  - Verzeichnis aller Dateien auf einem oder mehreren Speichergeräten
  - Selbst mit auf der Platte abgelegt
    - Meist auf einer festen Position (Zylinder 0, Spur 0, Block 0 o.ä.)
  - Kann hierarchisch gegliedert sein (UNIX, VMS, MS-DOS, …)
  - Leistet für jede Datei
    die **Abbildung des Dateinamens auf eine Folge von Blöcken**
    - Muss dann zu einer gegebenen Blocknummer
      die physische Slot-Adresse liefern können
- **Zusätzlich: Freispeicherverwaltung**
  - Liefert die unbenutzten Blöcke auf einer Platte
    (die momentan keiner Datei zugeordnet sind)
    - Also schnell zu finden bei **append**
  - Z.B. als Bitliste

Geräteunabhängigkeit

- **Vergleich:**
  - Programme, die mit blockorientierten Schreib-Lese-Operationen arbeiten, und Programme, die direkt auf die Platte zugreifen

Kriterium	direkt	blockorientiert
Geschwindigkeit	sehr schnell	langsamer wegen Katalogzugriff
Verwaltungsdaten	keine	Katalog
Schutz	keiner	Prüfung von Zugriffsrechten beim Öffnen
Erweiterung einer Datei	nur durch externe Absprache	systemverwaltet (Freispeicher, Extents)
Reorganisation (Verschieben von Blöcken)	Programmänderung	Änderung des Katalogs, Programme stabil
Wechsel des Plattenspeichertyps	Programmänderung	Änderung des Katalogs, Programme stabil

Geräteunabhängigkeit (2)

- **Anwendungsprogramme,
die blockorientierten Dateizugriff verwenden, sind
unabhängig von verwendeten Speichergeräten
(Magnetplatten, opt. Platten, Flash)**

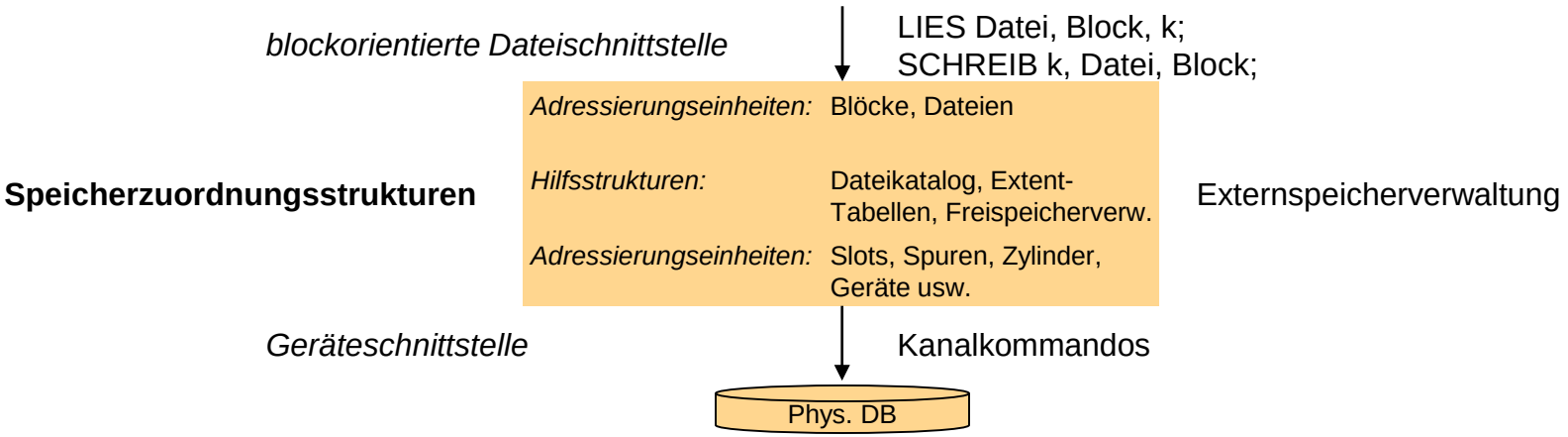
- **Das heißt:**
  - Speichergerät kann reorganisiert oder ausgewechselt werden,
ohne dass Programme geändert werden müssen
  - Für Programme ist Datei abstrakte Sicht auf diverse, evtl. verstreute Abschnitte (Slots) eines oder mehrerer Plattengeräte
    - Deshalb auch: "virtuelles Speichergerät" oder "logische Platte" – aber eigentlich noch abstrakter als das Logische Speichergerät von oben
  - Für dieselbe Datei verschiedene Abbildungen auf Speichergeräte möglich
    - Slots direkt hintereinander auch möglich!

Geräteunabhängigkeit (3)

- **Ein Fall von "Information Hiding":**
  - Abbildung der Blöcke einer Datei
    auf Slots eines Plattenspeichers
    wird dem Anwendungsprogramm verborgen

- **Im folgenden:**
  - Blockdatei selbst Basis für weitere Abstraktionen (weitere virtuelle Speicher)

Resultierende Schicht

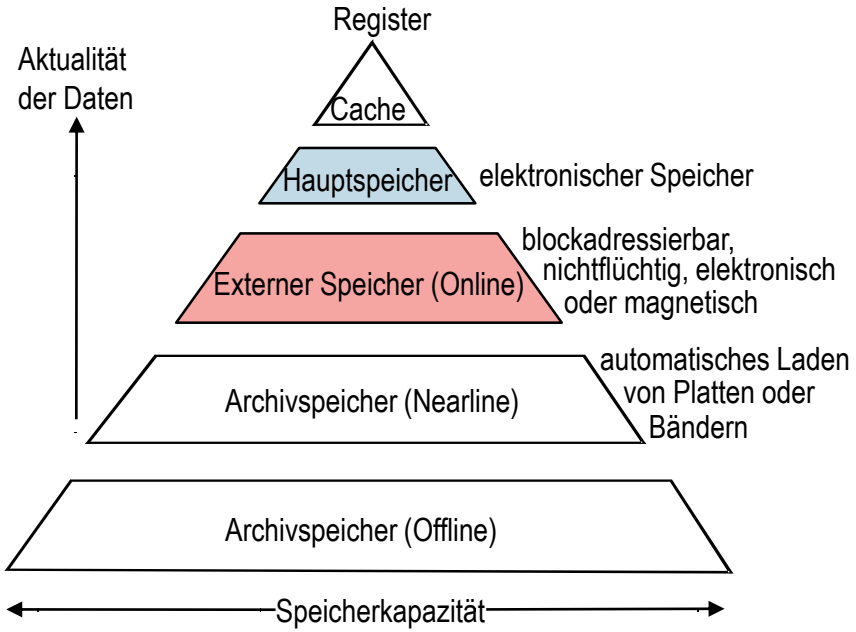


Anhang

- **Nicht mehr verwendete Folien**
- **Zum Vertiefen einzelner Aspekte und zum Nachschlagen bei Bedarf**

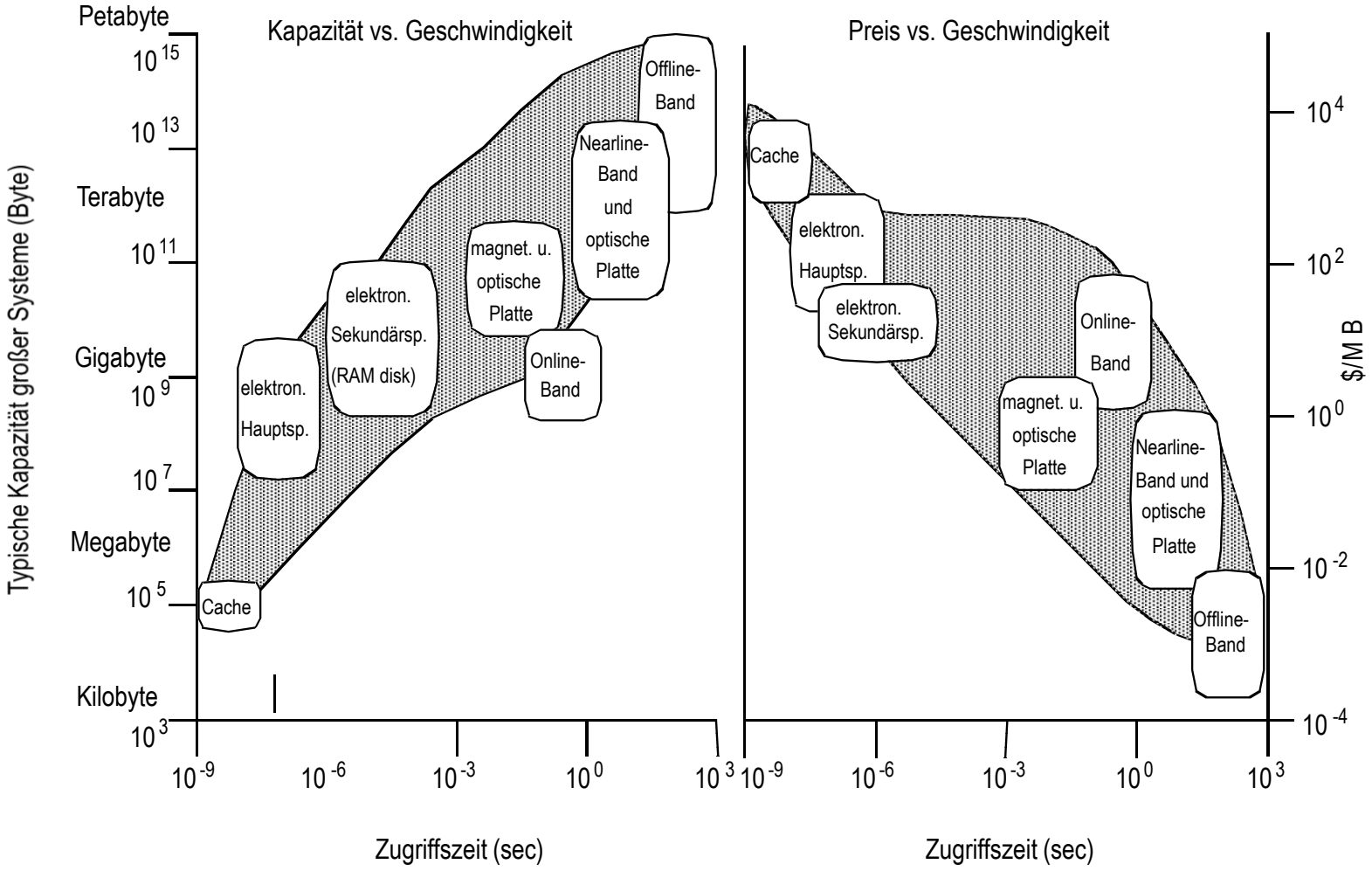
2.A.1 Physische Speichergeräte

- **Anforderungen**
  - Verwaltung externer Speichermedien / Unterstützung von Speicherhierarchien
  - Adressierung physischer Blöcke
  - Kontrolle des Datentransports vom/zum Hauptspeicher
  - Fehlertoleranzmaßnahmen (RAID-Systeme, ...)
- **Datenbankperspektive**
  - Betriebsysteme realisieren einen Verzeichnisdienst zur Verwaltung von Dateien mit unstrukturiertem Inhalt



Leistung versus Kosten bei Speichersystemen

(absolute Zahlen überholt, Verhältnisse aber heute ähnlich)



Zahlenbeispiel (veraltet)

- **Plattenlaufwerk Seagate Cheetah:**
  - 19036 Zylinder
  - 4 Spuren pro Zylinder
  - 947 Blöcke (Sektoren) pro Spur
  - 512 Byte pro Block
- **Daraus errechnen sich Kapazitäten von:**
  - 484.864 Byte (473,5 KB) pro Spur,
  - 1.939.456 Byte (1,85 MB) pro Zylinder und
  - 36.919.484.416 Byte (34,38 GB) pro Laufwerk
- **Drehzahl 10.000 UPM, also 166 Hz,**
  Dauer einer Umdrehung 6 ms
  - In einer Umdrehung max. die 473,5 KB einer Spur übertragen – theoretische **Datentransferrate** von 78.917 KB/s oder rund 77 MB/s
- **Positionierungszeit für den Zugriffskamm ("seek"-Zeit):**
  - durchschnittlich 6 ms, von Spur zu Spur 1,5 ms
- **Durchschnittliche Zugriffszeit: ~ 10 ms**

Optische Platten

- **CD-ROM**
  - Nur lesbar (wird "gedruckt")
  - 650 MB
  - Für DBS bisher nicht relevant
- **WORM**
  - Einmal schreibbar, dann nur noch lesbar ("write once, read many times")
  - 1 – 5 GB
  - Niedrigste Kosten pro Bit!
  - DBS: Protokolldaten, Versionen
- **Wiederbeschreibbare optische Platte**
  - Z.B. magneto-optische Platte
  - Wie alle optischen Platten relativ langsam (wahlfreier Zugriff 100 ms)
  - Eher Ersatz für Magnetband als für Magnetplatte

Zugriff auf Plattenspeicher (Dienste)

- **Typische Befehle an eine Plattensteuerung:**

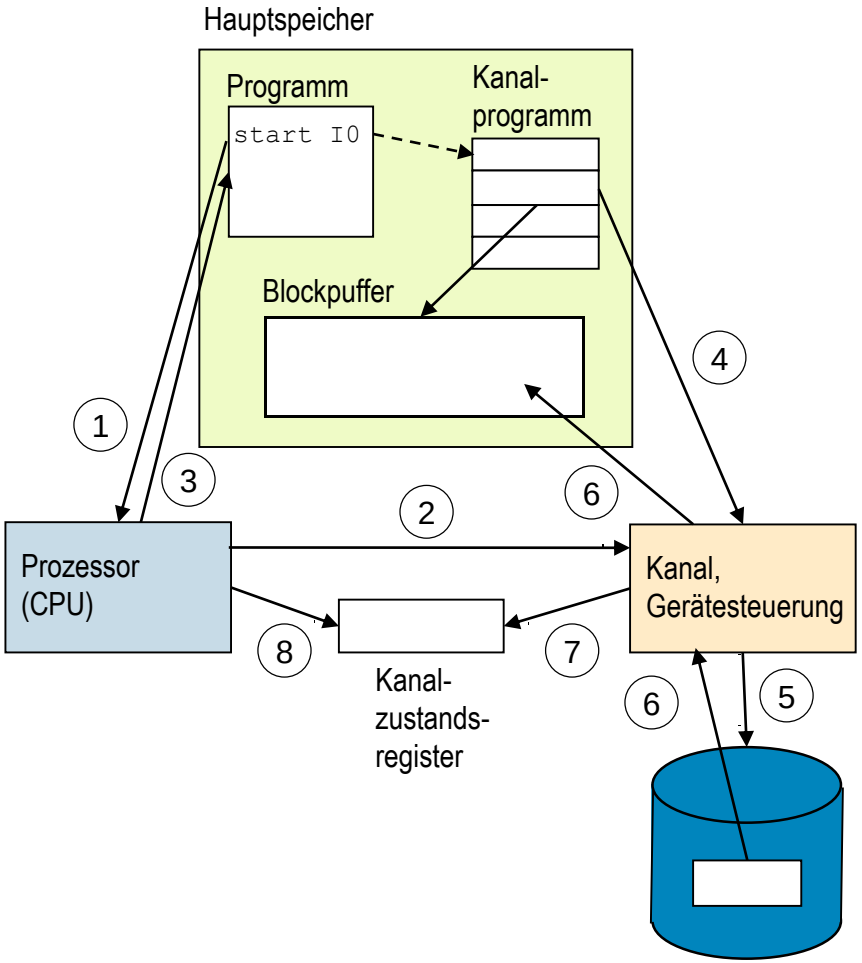
`SEEK n`	positionier Zugriffskamm auf Zylinder n
`SET HEAD n`	schalt Schreib-Lese-Kopf n ein
`SECTOR ROTATE n`	wart auf Anfang des Blocks n in der Spur
`READ a`	lies nächsten Block in den Hauptspeicher ab Adresse a
`WRITE a`	überschreib nächsten Block mit Hauptspeicherinhalt ab Adresse a
`VERIFY a`	vergleich nächsten Block mit Hauptspeicherinhalt ab Adresse a
`SELECT g`	Auswahl des Laufwerks g für die nächsten Befehle

Zugriff auf Plattenspeicher (2)

- **Folge zusammengehöriger Dienste**
  - zu Prozedur zusammenfassen
    - Z.B. Schreiben eines Blocks an eine bestimmte Stelle:
      Folge von **select, seek, set head, sector rotate, write, sector rotate** und **verify**
  - Abwicklung der Prozedur an *Spezialrechner* delegieren: **Kanal, Gerätesteuerung**
- **Minimalforderungen (notwendig für Persistenz):**
  - Erfolgreiche Schreiboperation hat tatsächlich Block an richtige Stelle geschrieben, ohne Nebenwirkungen
  - Bitfehler werden beim Lesen erkannt (z.B. Parität)
  - Alle Fehler werden korrekt diagnostiziert und gemeldet
    - Außerbetriebnahme, Fehlen des Datenträgers, mechanische Verklemmung, Schreibsicherung, Überlastung usw.
- **Ausführung der Operationen auch asynchron möglich:**
  - Abschluss der Operation (erfolgreich oder nicht)
    über Hardware-Register (Kanalzustandsregister) angezeigt

Kanal (Gerätesteuerung)

- **Lesen eines Blocks:**



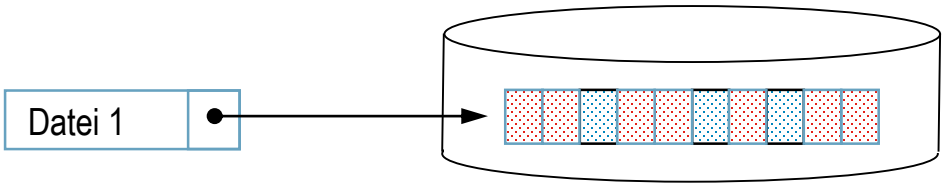
- **Schreiben eines Blocks**
  - analog
- **Lesen/Schreiben mehrerer Blöcke hintereinander**
  - mit einem Kanalprogramm möglich

2.A.2 Dateikatalog

- **Dateikatalog: Verwaltungsdatenstruktur**
  - zur Abbildung des Dateinamens auf eine Folge von Blöcken
  - Selbst mit auf der Platte abgelegt
    - Meist auf einer festen Position (Zylinder 0, Spur 0, Block 0 o.ä.)
  - Kann hierarchisch gegliedert sein (UNIX, VMS, MS-DOS)
  - Einstieg über den Dateinamen
    - Muss dann zu einer gegebenen Blocknummer die physische Slot-Adresse liefern können
- **Zusätzlich: Freispeicherverwaltung**
  - Unbenutzte Blöcke auf einer Platte (momentan keiner Datei zugeordnet)
  - Z.B. als Bitliste

Katalogeinträge

- **Möglichkeit A:**
  - Eintrag = (physische Slot-Adresse des ersten Blocks; Anzahl der Slots, die ab dieser Adresse belegt sind)

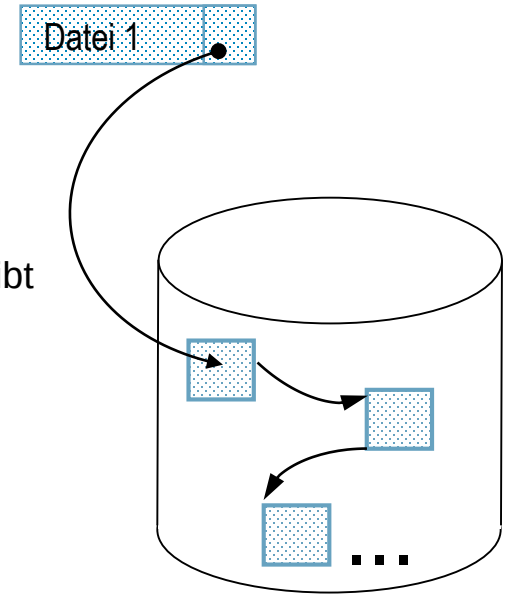


  - Benötigte Zahl von Blöcken auf physisch sequenzielle und lückenlose Folge von Slots abbilden (ggf. über Spur- und Zylindergrenzen hinweg)

    + Kompakter Dateikatalog (Eintrag pro Datei sehr klein)
    + Slot-Adresse von Block i kann errechnet werden
    + Sequenzielles Lesen aller Blöcke mit minimalen Zugriffskammbewegungen
    – Dynamische Erweiterung der Datei kann Umkopieren der ganzen Datei in größeren zusammenhängenden Bereich erfordern oder unmöglich sein
    – "Zerklüftung" der Platte: nicht nutzbare kleine Stücke zwischen den Dateien

Katalogeinträge (2)

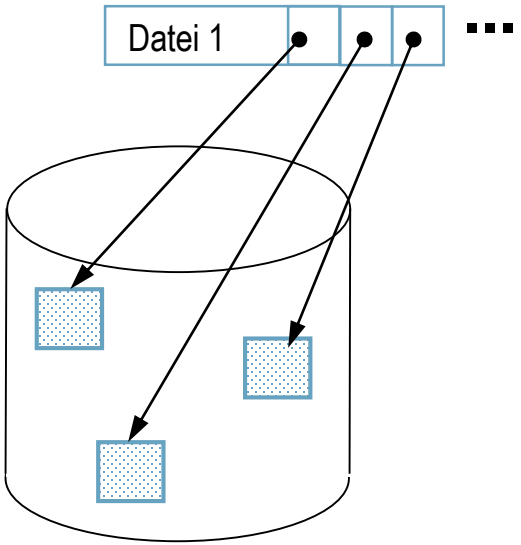
- **Möglichkeit B:**
  - Eintrag = (physische Slot-Adresse des ersten Blocks)
  - Verkettung der Blöcke als lineare Liste



    + Kompakter Dateikatalog
    + Erweiterung um neue Blöcke:
      immer möglich,
      wenn es auf der Platte irgendwo noch genug freie Slots gibt
    – Speicherplatz:
      in jedem Block muss Platz reserviert werden
      für Zeiger auf Nachfolger (Slot-Adresse, 6–8 Byte)
    – Sequenzielles Lesen der ganzen Datei:
      i.Allg. sehr langsam
      (viele Bewegungen des Zugriffskamms)
    – Zugriff auf Block i:
      erst nach Lesen der Blöcke 1 bis i–1 möglich
      (Kette entlanglaufen) – nicht akzeptabel!

Katalogeinträge (3)

- **Möglichkeit C:**
  - Eintrag = (Array mit den Slot-Adressen aller Blöcke)
  - Bekanntestes Beispiel: FAT



+ Zugriff auf Block i:
  problemlos
+ Erweiterung um neue Blöcke:
  immer möglich,
  wenn es auf der Platte irgendwo noch
  genug freie Slots gibt
– Sequenzielles Lesen aller Blöcke:
  langsam (viele Zugriffskammbewegungen)
– Katalog sehr groß,
  variabel lange Einträge, ändern sich bei jeder Erweiterung

Katalogeinträge (4)

- **Möglichkeit D:**
  - Eintrag = (Array von (Slot-Adresse; Zahl der von dieser Adresse an sequenziell belegten Slots))
    - 0 Katalog in der Regel deutlich kleiner als bei C, aber auch variabel lange Einträge
    - \+ Slot-Adresse von Block i berechenbar
    - \+ Erweiterung immer möglich, wenn irgendwo noch genügend freie Slots (ggf. in Teilstücken)
    - 0 Sequenzieller Zugriff um so besser, je weniger Teilstücke es gibt
    - – Kann zu C degenerieren
    - \+ Kann durch Reorganisation der Platte in A überführt werden (ohne Änderung der Anwendungsprogramme!)
  - **Extent**
    - Zusammenhängendes (physisch sequenzielles) Teilstück einer Datei
    - Typischerweise im Zuge einer Erweiterung belegt
    - Durch ein einzelnes Array-Element im Katalogeintrag repräsentiert
  - Katalogeintrag einer Datei: **Extent-Tabelle**