Speicherverwaltung: Platten und Dateien

Platten und Dateien

- DBMS speichert Informationen auf externen Datenträgern: Magnetplatten (“Hard“ Disks) oder Bänder (Tapes)
- Wir betrachten nachfolgend nur Platten (Bänder als sequentielle Datenträger ungeeignet)
- Zwei Hauptoperationen:
  - READ: Datentransfer von Platte in den Hauptspeicher (RAM), auch Fetch genannt
  - WRITE: Datentransfer vom RAM auf Platte
  - Beides sind Operationen mit hohen Kosten im Vergleich zu in-memory Operationen, müssen deshalb sorgfältig geplant werden!
  - An den Lese- und Schreiboperationen sind mehrere Komponenten eines DBMS beteiligt, z.B. zur Identifikation eines Datensatzes auf dem externen Datenträger
- Bedeutende Auswirkungen auf den Entwurf eines DBMS

Warum nicht alles im Hauptspeicher behalten?

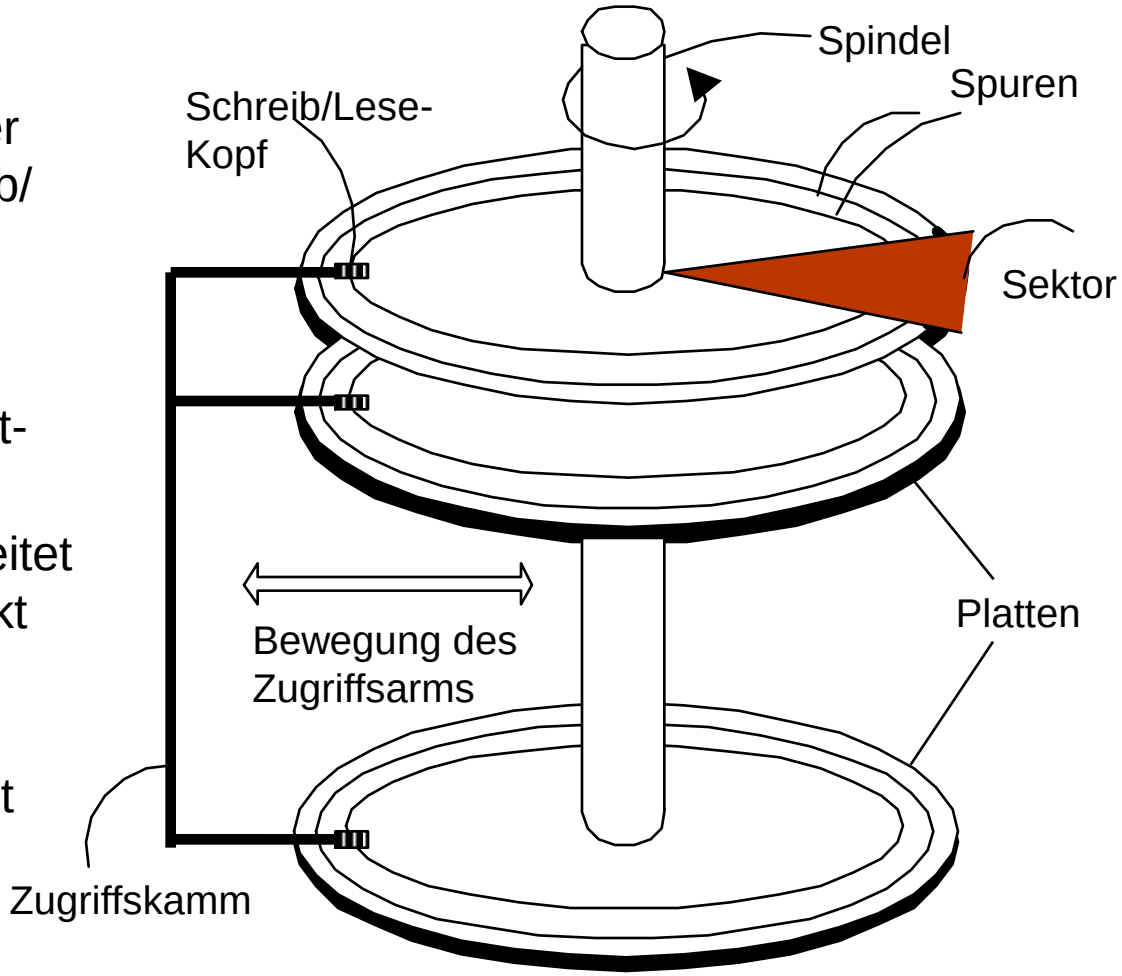
- *Zu hohe Kosten*. Für 14 € gibt‘s heute 4 GB DDR3-RAM, für 60 € 1 TB SATA Disk [Stand 2011]
- *Hauptspeicher ist flüchtig*. Wir möchten Daten zwischen einzelnen Programmläufen speichern. (Offenkundig!)
- *Adreßraum* eines Hauptspeichers nicht ausreichend (oft nur 32-Bit Adressen, d.h. 2^{32} Bytes adressierbar)
- Typische Speicherhierarchie:
  - Primärspeicher:
    - Cache für sehr schnellen Datenzugriff (64 KB - 32 MB, 6 ns)
    - Main Memory (RAM) für häufig genutzte Daten (MB - GB, 25 ns)
  - Sekundärspeicher:
    - Disks für die aktuelle Datenbank (GB - TB, 12 ms)
  - Tertiärspeicher:
    - Tapes für die Archivierung älterer Versionen der Daten oder Sicherheitskopien für Recovery-Zwecke (billigstes Speichermedium).
  - Neu: Flash-Memory als „extended buffer pool“ oder „extended disk“

Platten

- Bevorzugtes Speichermedium für Sekundärspeicher
- Hauptvorteil gegenüber Bändern: Direktzugriff (*random access*) gegenüber *sequentiellem* Zugriff
- Daten werden gespeichert und gelesen in Einheiten, genannt *Blöcke* oder *Pages*.
- Blöcke werden in konzentrischen Ringen angeordnet = *Spuren (Tracks)*
- Spuren können auf beiden Seiten der Platten angelegt werden
- Im Gegensatz zum RAM hängt die Zeit zum Auffinden einer Page vom Speicherort auf der Platte ab.
  - Deshalb hat die relative Plazierung von Pages auf der Platte großen Einfluß auf die Performance des DBMS.

Komponenten einer Platte

- Plattenumdrehung (ca. 10.000/min)
- Zugriffskamm wird hinein- oder hinausbewegt, um den Schreib/ Lesekopf auf die gewünschte Spur zu positionieren
- Zylinder = alle übereinander angeordneten Spuren von Plattenoberflächen
- Nur ein Schreib/Lesekopf arbeitet zu einem bestimmten Zeitpunkt (liest oder schreibt)
- Blockgröße ist das Vielfache einer Sektorgröße (welche fest ist)



Zugriff auf eine Page

- Adressierung von Blöcken über Zylinder-, Spur- und Sektor-Nr.
- Zugriffskamm muß über entsprechenden Zylinder geführt werden, auf den Sektor warten, der irgendwann vorbeirotiert und schließlich den Blockinhalt lesen und übertragen
- Zugriffszeit (read / write) auf einen Block:
  - Zugriffsbewegungszeit (seek time)

    Bewegung der Zugriffsarme, um den Schreib/Lesekopf auf der richtigen Spur zu positionieren
  - Umdrehungswartezeit (latency time, rotational delay)

    Warten, bis der gesuchte Block am Kopf vorbeirotiert
  - Übertragungszeit (transfer time)

    Tatsächlicher Transport der Daten von/zu der Plattenoberfläche
- Seek time (1..20 ms) und Umdrehungswartezeit (0..10 ms) dominieren
- Transfer-Rate ca. 125 MB/s
- Mittlere Zugriffszeit: ca. 12.. 18 ms
- Reduzierung der I/O-Kosten heißt: Seek time und Umdrehungs-wartezeit verringern! (Hardware- vs. Software-Lösungen)

Anordnung von Pages auf einer Platte

- ‘Next‘ Block - Konzept
  - Blöcke auf der gleichen Spur, gefolgt von
  - Blöcken auf dem gleichen Zylinder, gefolgt von
  - Blöcken auf benachbartem Zylinder
- Blöcke sollten in einer Datei sequentiell angeordnet sein (‘next‘), um die Zugrifsbewegungszeit und die Umdrehungswartezeit
- Prefetching
  - Derzeit noch nicht (aber vielleicht in Kürze) benötigte Pages werden gleich mit gelesen
- Es gilt: Sequentielles Lesen (Scan) ist immer viel schneller als wahlfreier Zugriff (Random Access)

Speicherbedarf von Informationen und Speicherkapazität

Größe	Information oder Speichermedium
1 KB	~ 1.000
0.5 KB	Buchseite als Text
30 KB	Eingescannte komprimierte Buchseite
1 MB	~ 1.000.000
5 MB	Die Bibel als Text
20 MB	Eingescanntes Buch
500 MB	CD-ROM; Oxford English Dictionary
1 GB	~ 1.000.000.000
4.7 GB	Digital Versatile Disk (DVD)
10 GB	Komprimierter Spielfilm
100 GB	Ein Stockwerk einer Bibliothek
1 TB	~ 1.000.000.000.000
1 TB	Bibliothek mit 1M Bänden
11 TB	Erstes großes Data Warehouse (Wal-Mart)
2 PB	Library of Congress (weltgrößte Bibliothek): 155.3 Mio Objekte, darunter 35 Mio Bücher, Gesamtumfang aller digitalisierten Objekte [Stand 2011],

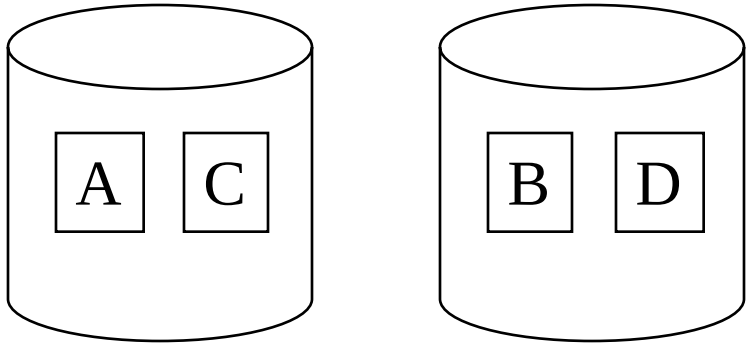
RAID

- Disk Array: Anordnung mehrerer Platten, die in Abstraktion als einzige große Platte angesehen werden können (Steuerung erfolgt durch Controller)
- Ziele:
  - Steigerung der Performance durch Parallelität des Zugriffs
  - Verfügbarkeit (Ausfallsicherheit, Zuverlässigkeit)
- Zwei Techniken:
  - Data Striping:
    - Daten werden partitioniert
    - die Größe einer Partition heißt Striping Unit
    - Verteilung der Daten: bitweise, byteweise oder blockweise
    - Partitionen werden über mehrere Platten verteilt (meist nach Round-Robin-Algorithmus)
  - Redundanz: Mehr Platten -> Mehr Fehler
    - Redundante Informationen erlauben Wiederherstellung der Daten im Fehlerfall.
    - Entweder Duplikate halten oder nur Kontrollinfos (z.B. Paritätsbits)

RAID Levels: Level 0

Level 0: Keine Redundanz

- Datenblöcke werden im Rotationsprinzip auf (physische) Platten verteilt
- Striping mit Block-Granularität
- Zuverlässigkeit sinkt linear mit der Anzahl der Platten
- Beste Write-Performance aller RAID-Levels
- Lastbalancierung bei vielen parallel anstehenden Lesetransaktionen (sehr gut beim sequentiellen Lesen)
- Wahlfreier Zugriff nicht beschleunigt

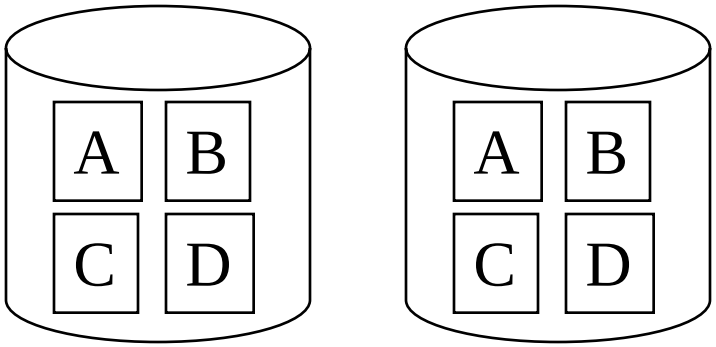


Blöcke: A, B, C, D
2 Platten

RAID Levels: Level 1

Level 1: Spiegelung

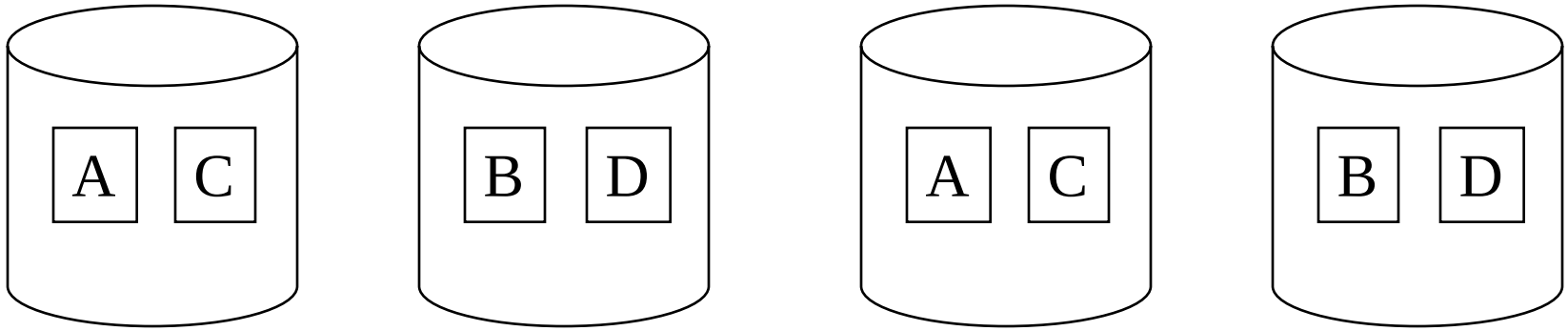
- Zwei identische Kopien
- Jede Platte hat eine Spiegelplatte (Mirror image), d.h. zwei identische Kopien
- Teuerste Lösung: jedes Write muß auf beiden Platten erfolgen
- Parallele Reads auf verschiedenen Platten möglich
- Maximale Datentransfer-Rate = Transfer-Rate einer Platte
- 50%ige Nutzung des Plattenspeichers



RAID Levels: Level 0 + 1

Level 0+1: Striping und Spiegelung

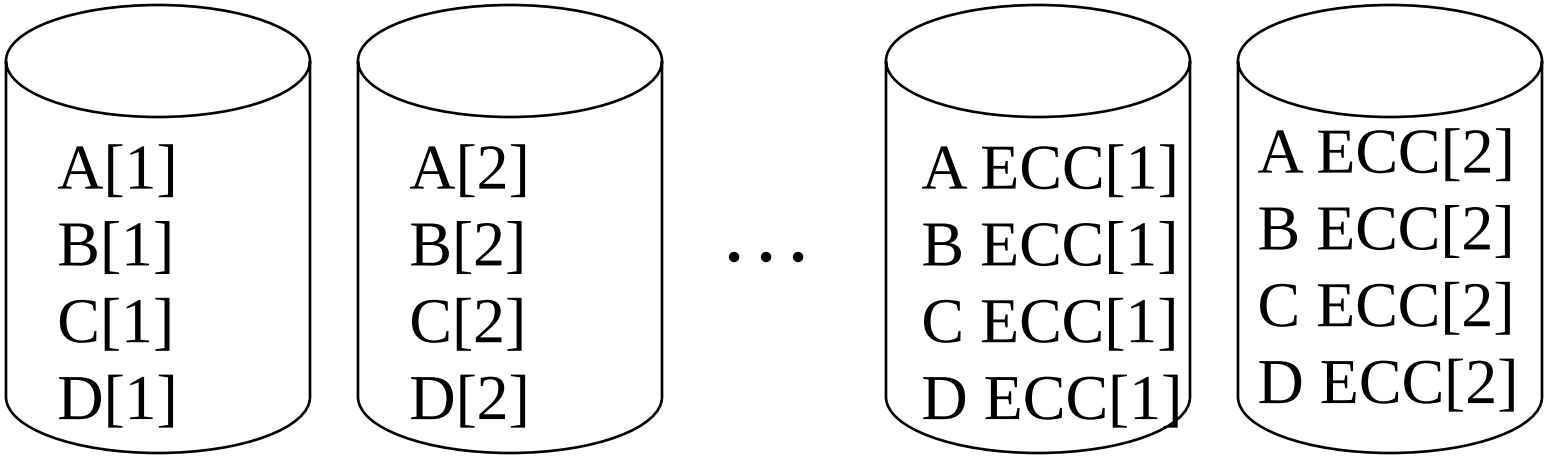
- Kombination aus Level 0 (Striping) und Level 1 (Spiegelung)
- Parallele Reads möglich
- Write-Kosten analog zu Level 1 (beide Platten involviert)
- Maximale Transfer-Rate = Aggregierte Bandbreite aller Platten



RAID Levels: Level 2

Level 2: Error-Correcting Codes

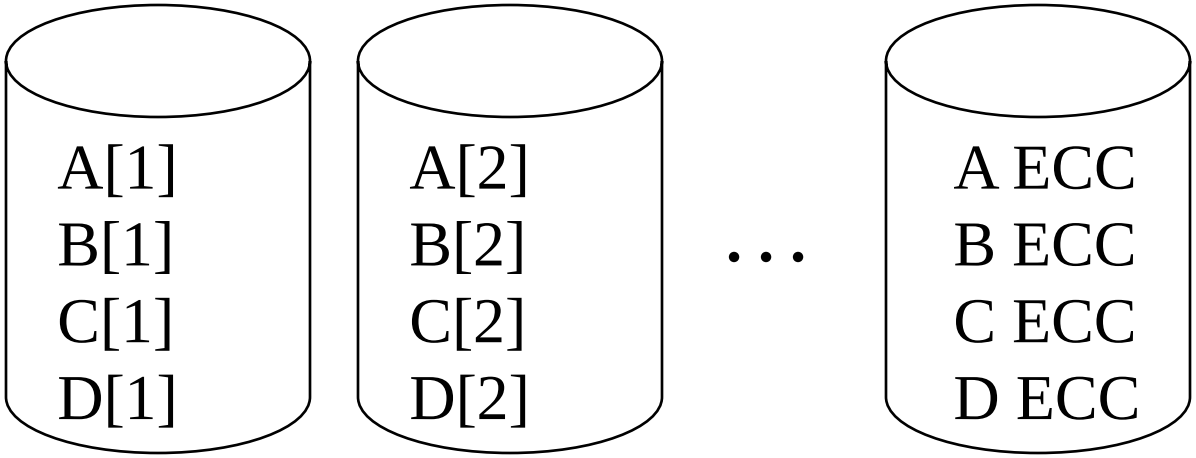
- Striping auf Bitebene
- Speicherung zusätzlicher Paritätsbits oder Error Correcting Codes (ECC) auf zusätzlichen Platten
- Regel: 1. Bit auf 1. Platte, ..., i. Bit auf Platte (i mod n); Kontrollinfos auf zusätzlichen Platten
- Read erfordert 8 parallele Lese-Operationen (für 1 Byte)
- Zugriff auf einzelne Daten nicht schneller, aber höherer Gesamtdurchsatz



RAID Levels: Level 3

Level 3: Bit-Interleaved Parity

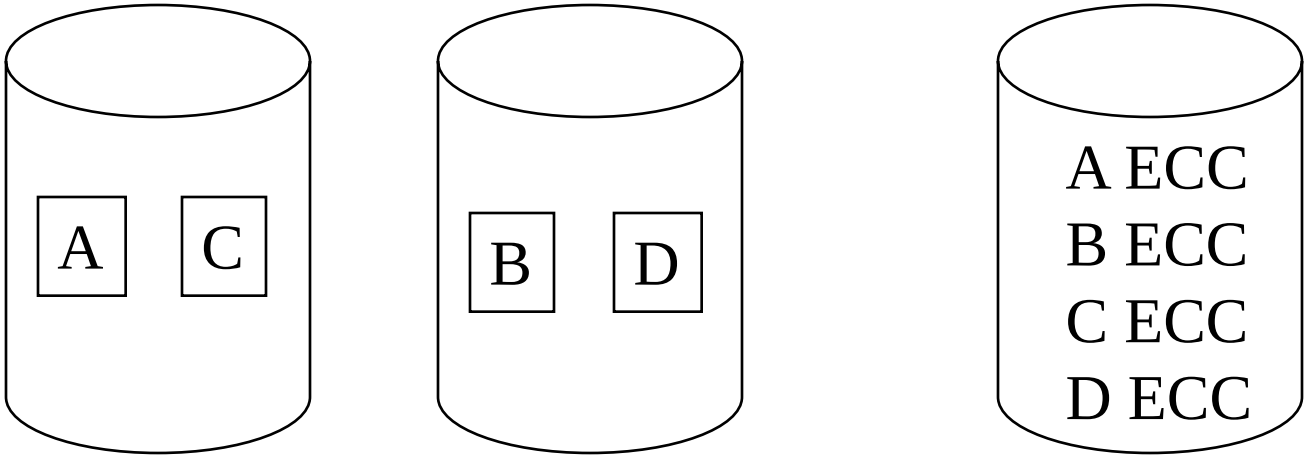
- Striping auf Bitebene
- Nur eine Check Disk (ein Paritätsbit ausreichend), somit platzsparender als Level2
- Read erfordert n Platten; Write erfordert n+1Platten parallel
- Disk Array kann einen Request gleichzeitig verarbeiten
- Zugriff auf einzelne Daten nicht schneller, aber höherer Gesamtdurchsatz



RAID Levels: Level 4

Level 4: Block-Interleaved Parity

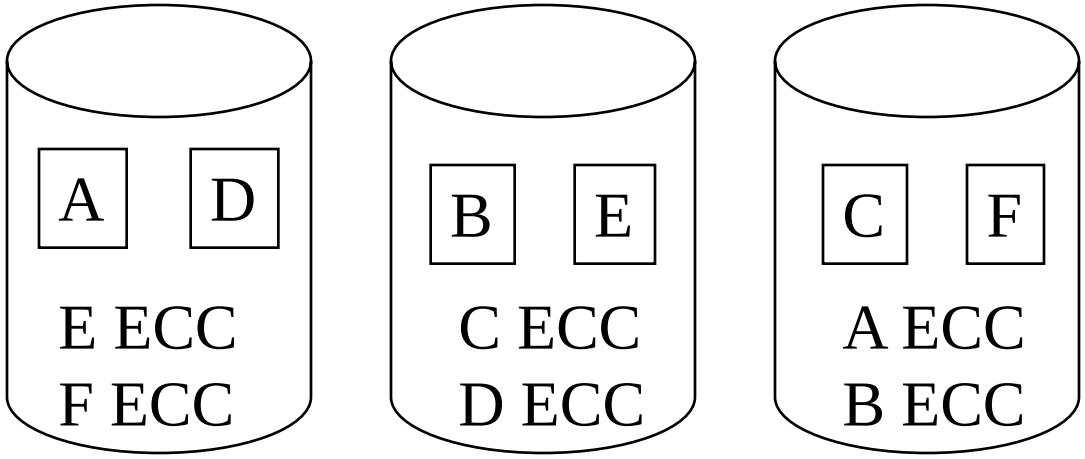
- Striping auf Blockebene
- Nur eine Check Disk zur Aufnahme des Paritätsbits
- Kleine Datenmengen können effizienter gelesen werden, da nur eine physische Platte betroffen ist
- große Read-Requests können die volle Bandbreite des Disk Array nutzen
- Write eines einzelnen Blocks betrifft 2 Platten: Data und Check Disk (1. Lese Datenblock und Paritätsbit-Block 2. Modifiziere beide Blöcke 3. Schreibe die Blöcke zurück)



RAID Levels: Level 5

Level 5: Block-Interleaved Distributed Parity

- Ähnlich wie Level 4
- Blöcke mit den Paritätsbits sind über alle Datenplatten verteilt
- Vorteile: Write-Requests können parallel ausgeführt werden (kein Bottleneck mehr durch einzige Check Disk); höhere Parallelität beim Lesen (da alle Platten genutzt werden)
- Beste Performance aller RAID Levels für kleine und große Read-Requests und große Write-Requests



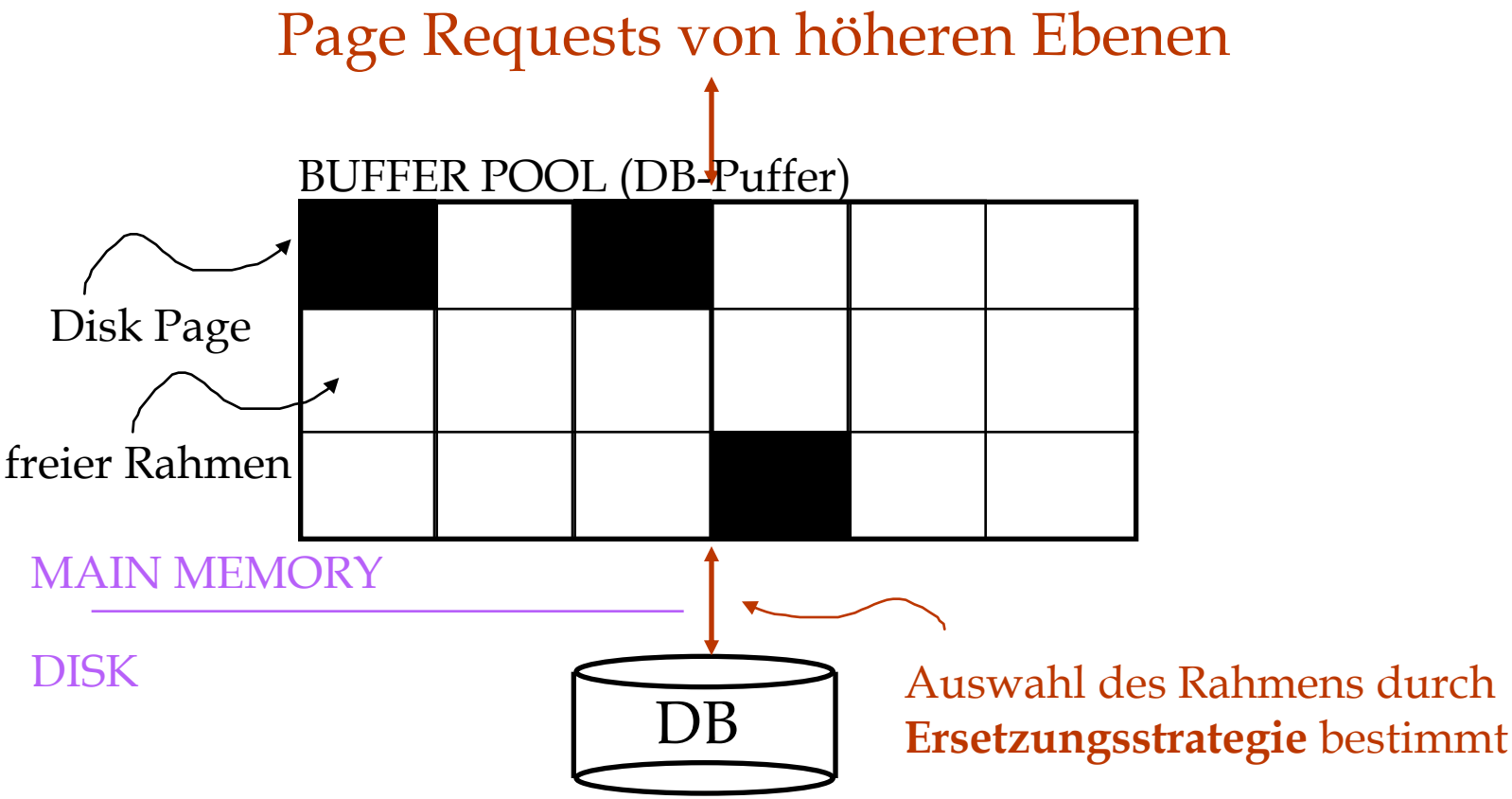
Verwaltung des Hintergrundspeichers

- Annahme: Es gelte Sekundärspeicher = Plattenspeicher
- Niedrigste Ebene in einem DBMS verwaltet den Speicherplatz auf der Platte (*Disk Space Management*)
- Konzept einer Seite (Page): Dateneinheit
- Zuordnung der physischen Blöcke zu sogenannten Seiten (Pages) mit festen Faktoren (1,2,4,8...)
- Annahme 1 Block = 1 Seite
- Adressierung über Seiten-Nr., die dann in Block/Sektor-Nr. übersetzt wird
- Operationen_
  - Allokation / Deallokation von Pages
  - Read / Write von Page
- Ein Request für eine Sequenz von Pages muß erfüllt werden durch sequentielle Allokation von Pages auf der Platte (z.B. bei logisch aufeinanderfolgenden Daten)
- Nach vielen Updates Reorganisation erforderlich (Freispeicher-verwaltung), jedoch für höhere Schichten des DBMS nicht sichtbar

Betriebssystemdateien

- Verschiedene Varianten, wie Konzepte des Betriebssystems durch ein DBMS genutzt werden^:
- Jede Relation oder jeder Zugriffspfad in genau einer BS-Datei
  - Feinstruktur der Datenbank auch im Dateisystem sichtbar
- DBMS legt ein oder mehrere Dateien durch das BS an und verwaltet Relationen und Zugriffspfade innerhalb dieser Dateien
- DBMS steuert selbst die Platte an und arbeitet mit den Blöcken in ihrer Ursprungsform (*raw device*), d.h. Dateisystem wird am BS vorbei re-implementiert (z.B. wenn BS die Anforderungen nicht erfüllt)
  - DBMS kann betriebssystemunabhängig implementiert werden
  - größere Dateigrößen benötigt (z.B. mehr als 4 GB)
  - Mediumübergreifende Datenbanken
  - Betriebssystemseitige Pufferverwaltung genügt nicht den Anforderungen an des DBMS

Pufferverwaltung in einem DBMS



- *Daten müssen im RAM sein, damit DBMS darauf operieren kann*
- *Tabelle mit <frame#, pageid> pairs wird bewirtschaftet.*

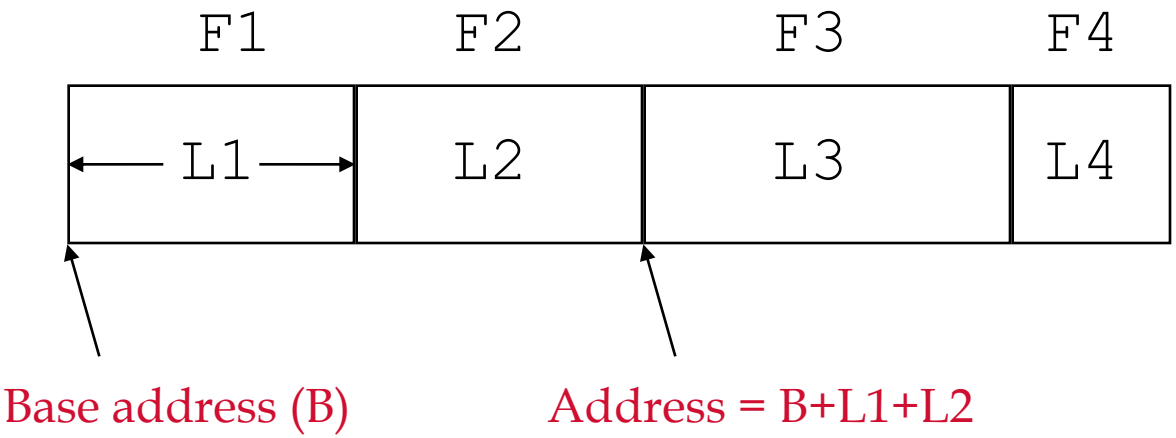
Wenn eine Seite benötigt wird ...

- Wenn die angeforderte Seite nicht im Puffer ist:
  - Auswahl eines Pufferrahmens zur Ersetzung (wenn sämtliche Puffer-Rahmen belegt) nach einer bestimmten Strategie
  - Wenn der zu verdrängende Rahmen in der Zwischenzeit im Puffer verändert wurde (*dirty*), wird er zunächst auf Platte geschrieben
  - Anderenfalls kann die betroffene Seite direkt überschrieben werden
  - Einlesen der gesuchten Seite in den gewählten Rahmen
- Rückgabe der Adresse der gelesenen Seite
- Wenn Requests vorhergesehen werden können (z.B. beim sequentiellen Lesen) ist ein Pre-Fetch von Seiten möglich
- Synchronisationsprobleme (z.B. gleichzeitiger Request auf dieselbe Page von mehreren Benutzern) werden schon in höheren Schichten des DBMS gelöscht (z.B. durch Sperrenverwaltung)
- Angeforderte Seiten werden mit einem Zähler (*pin count*) verwaltet
- Auswahl eines Rahmens zur Ersetzung durch eine Ersetzungsstrategie:
  - Least-recently used (LRU)
  - Most-recently used (MRU) u.a. ...
- Strategie hat großen Einfluß auf I/O - abhängig vom Zugriffsmuster

DBMS vs. Filesystem des Betriebssystems

- Betriebssystem umfaßt Plattenspeicher- und Puffermanagement --> warum nicht nutzen fürs DBMS?
- Es gibt Unterschiede bei den Betriebssystemen --> Portabilitäts-probleme
- Beschränkungen: z.B. keine plattenübergreifenden Files
- Puffermanagement erfordert spezielle Fähigkeiten:
  - Markieren einer Seite im Puffer (inkrementieren Verwendungszähler), Zwingen einer Seite auf Platte (z.B. wichtig für Concurrency Control und Recovery)
  - Abstimmung von Ersetzungsstrategien und Unterstützung von Prefetching basierend auf Zugriffsmustern in typischen Datenbank-operationen

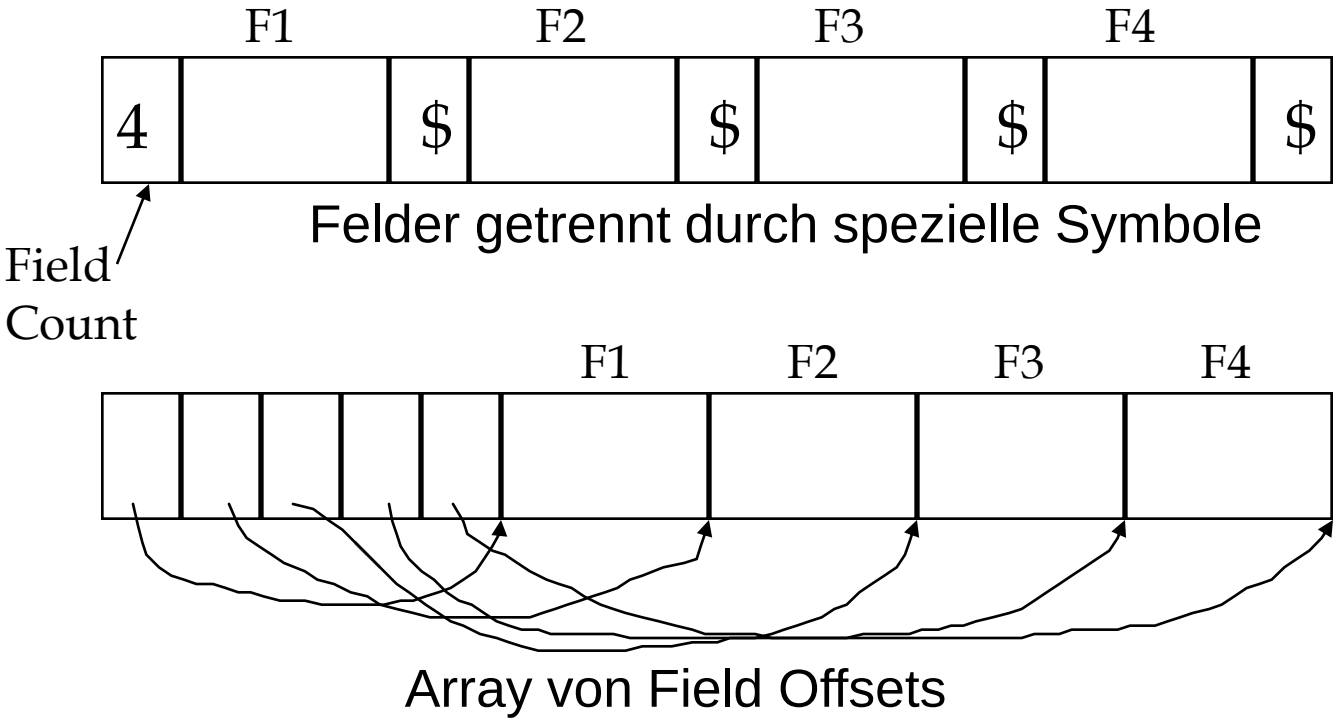
Satzformate in einem File: Feste Länge



- Informationen über Feldtypen sind gleich für alle Datensätze in einem File: gespeichert in *System Catalog.*
- Finde das i-te Feld erfordert Scannen des Datensatzes

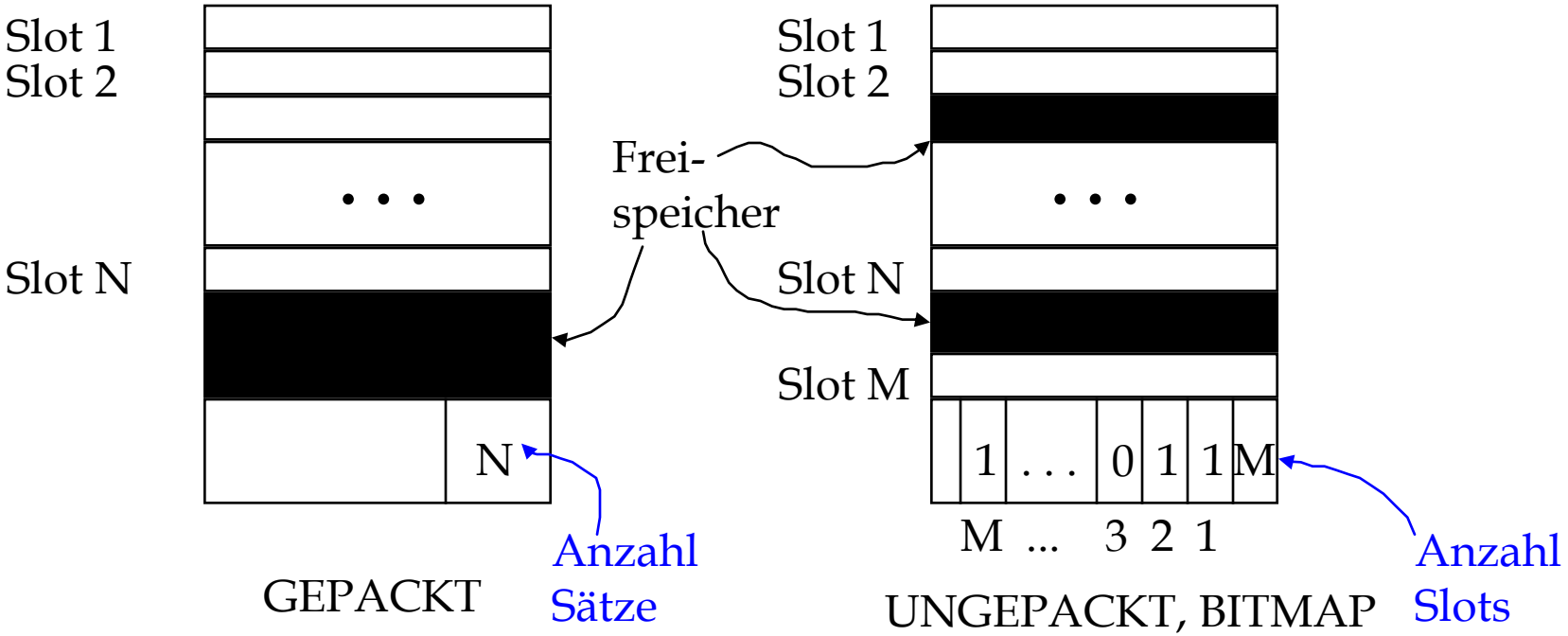
Satzformate: Variable Länge

- Zwei alternative Formate (# Felder ist fix):



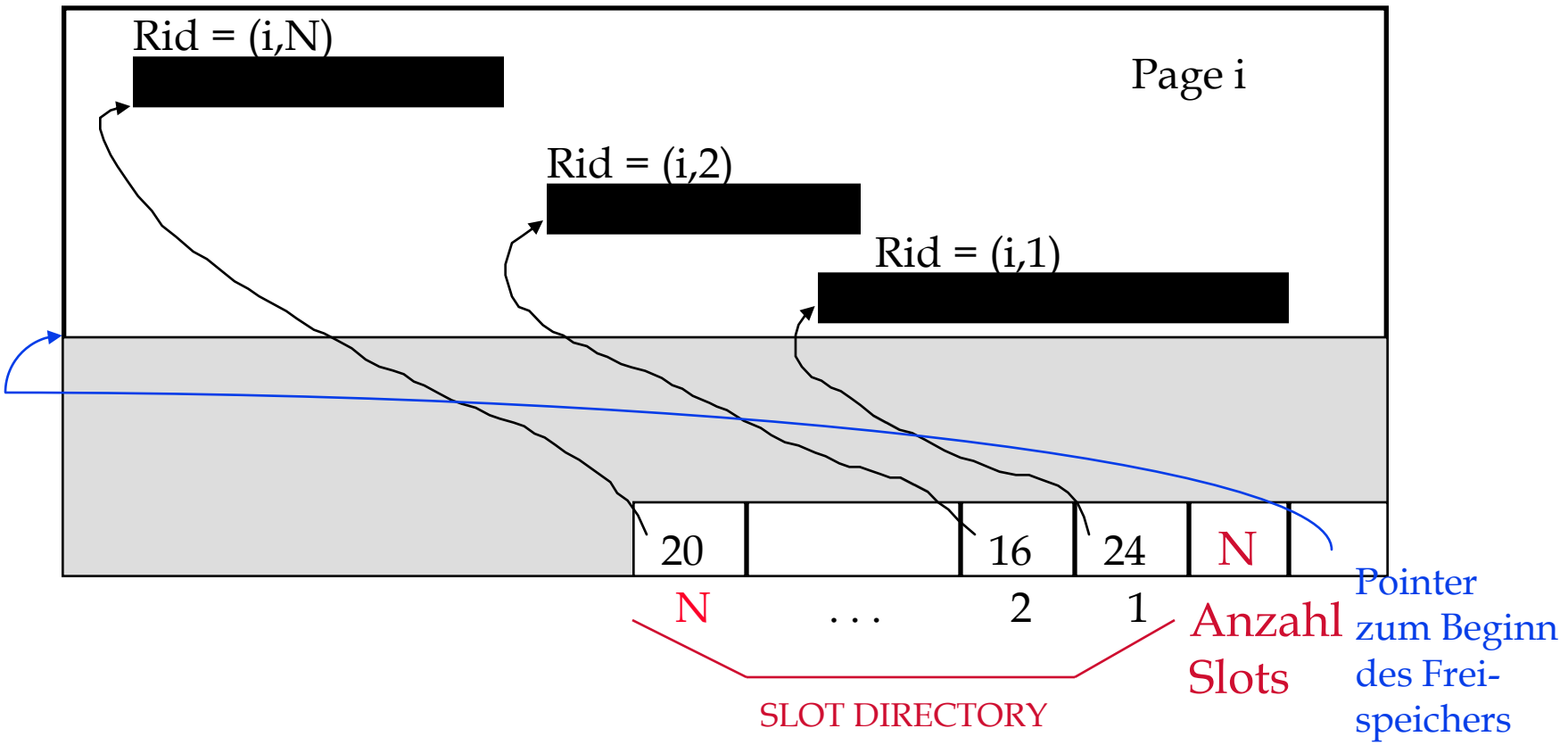
- Zweite Variante erlaubt direkten Zugriff aufs i-te Feld, effiziente Speicherung von Nulls; geringer Directory-Overhead

Seitenformate: Sätze fester Länge



- *Record id = <page id, slot #>. (Slot = Tupel-Identifikator)* Erster Ansatz: Verschieben von Sätzen in Freispeicher (z.B. beim Löschen), so daß gilt: Satz i im Slot i. Dies ändert jedoch die ID des Satzes (meist nicht akzeptabel).
- Bessere Alternative mit Bitmap.

Seitenformate: Sätze variabler Länge



- Verschieben von Sätzen in einer Seite ohne Veränderung der ID möglich (somit auch attraktiv für Sätze fester Länge)
- Im Slot Directory (Liste von Satz-Zeigern) stehen Offset und Länge für jeden Satz

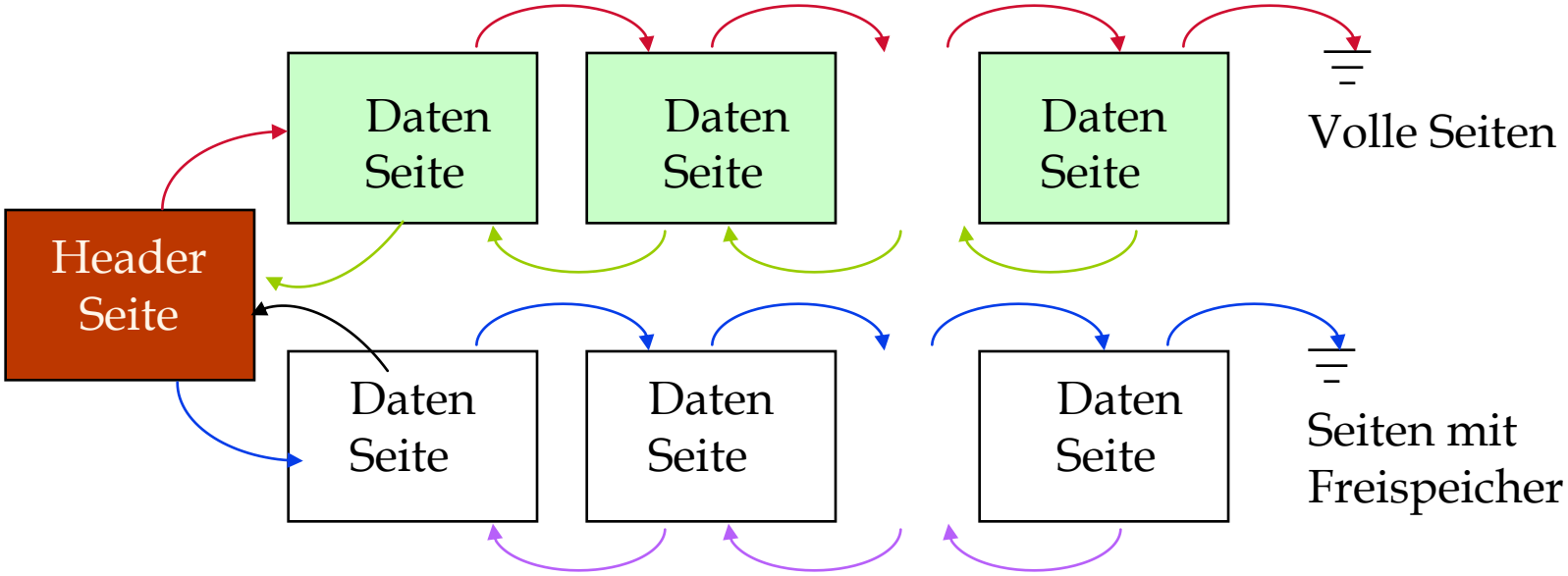
Datei aus Sätzen

- Für I/O-Operationen genügen Seiten/Blöcke
- Höhere Schichten eines DBMS operieren auf Sätzen (Records) und Dateien (Files) aus Sätzen
- File:
  - Ansammlung von Seiten, die jeweils eine bestimmte Anzahl von Sätzen enthalten
  - Unterstützt folgende Operationen:
    - Einfügen / Löschen / Modifizieren von Sätzen
    - Lesen eines bestimmten Satzes (spezifiziert durch eine Record-ID)
    - Scannen aller Sätze (möglich mit Einschränkungen auf eine Teilmenge von Sätzen, die zurückgegeben werden sollen)

Ungeordnete Heap-Dateien

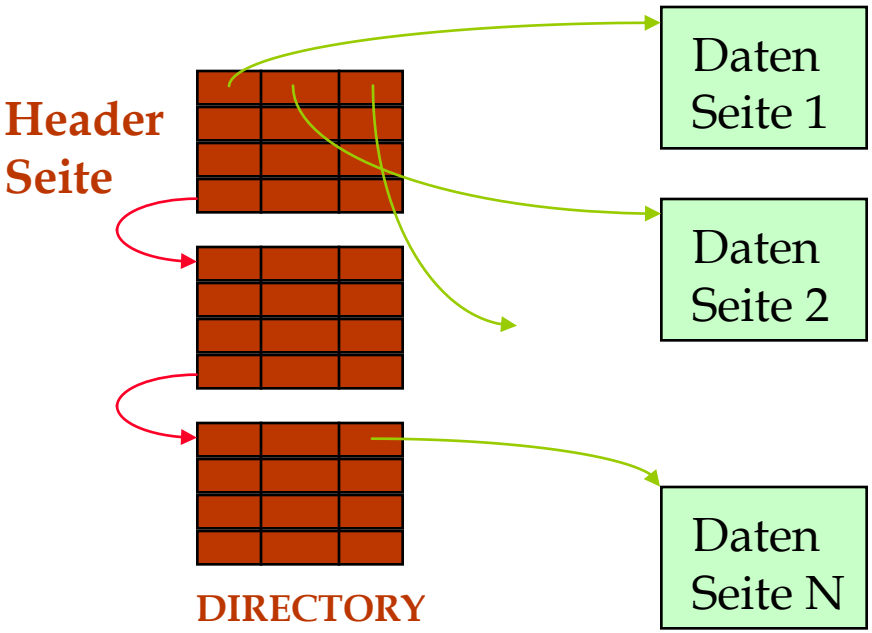
- Einfachste Dateistruktur:
  - Enthält Sätze ohne bestimmte Ordnung
- Beim Wachsen und Verkleinern der Datei werden Seiten auf der Platte allokiert und deallokiert
- Zur Unterstützung von Operationen auf Satzebene wird benötigt:
  - Kontrolle der Seiten in einer Datei
  - Kontrolle des Freispeichers in einer Seite
  - Kontrolle der Sätze auf einer Seite
- Für die Implementierung dessen gibt es zahlreiche Alternativen

Implementierung einer Heap-Datei als Liste



- Header-Seite und Name der Heap-Datei müssen an bekanntem Ort auf der Platte gespeichert sein
- Jede Seite enthält zwei Pointer und die Daten

Heap-Datei mit Seitenverzeichnis



- Das Directory ist eine Sammlung von Seiten; Implementierung als verkettete Liste ist dazu eine Alternative.
- Der Eintrag für eine Seite kann die Anzahl freier Bytes auf der Seite beinhalten.
- Effiziente Suche im Verzeichnis möglich (z.B. nach Datenseiten mit genug Platz)
- *Diese Lösung erfordert weniger Platz als eine verkette Liste aller Seiten*

Indexe (Einführung)

- Heap-Datei erlaubt uns, Sätze zu suchen
  - durch Angabe der Record-ID
  - durch das sequentielle Scannen aller Sätze
- Auch gewünscht: Suche nach Sätzen mit einem bestimmten Wert in einem oder mehreren Feldern, z.B.
  - Finde alle Studenten im Fachbereich “Informatik“
  - Finde alle Studenten mit Durchschnitt < 3
- Indexe sind spezielle Dateien, die es gestatten, wert-basierte Anfragen effizient zu beantworten

System-Kataloge

- Für jeden Index:
  - Struktur (z.B. B+ Baum) und die Schlüsselfelder zum Suchen
- Für jede Relation:
  - Name, Dateiname, Dateistruktur (z.B. Heap File)
  - Für jedes Attribut: Attributname und Typ
  - Für jeden Index: Indexname
  - Integritätsbedingungen
- Für jede Sicht:
  - Sichtname und Definition
- Weitere Daten:
  - Statistik
  - Autorisierungsinformationen (Benutzer, Gruppen, Rollen, Rechte)
  - Konfigurationsdaten (z.B. Größe des Datenbank-Puffers)

Systemkataloge sind selbst als Relationen gespeichert

Beispiel

Attr_Cat(attr_name, rel_name, type, position)

attr_name	rel_name	type	position
attr_name	Attribute_Cat	string	1
rel_name	Attribute_Cat	string	2
type	Attribute_Cat	string	3
position	Attribute_Cat	integer	4
sid	Students	string	1
name	Students	string	2
login	Students	string	3
age	Students	integer	4
gpa	Students	real	5
fid	Faculty	string	1
fname	Faculty	string	2
sal	Faculty	real	3

Zusammenfassung

- Platten stellen billigen, nicht-flüchtigen Speicher bereit
  - Direktzugriff möglich, Kosten hängen vom Speicherungsort der Seite auf der Platte ab;
  - wichtig: Daten sequentiell anordnen, um Zugriffszeit (Zugriffs-bewegungszeit, Umdrehungswartezeit) zu minimieren
- Puffermanager bringt die Seiten in den RAM
  - Seite bleibt im Puffer, bis sie wieder freigegeben wird
  - Seite auf Platte schreiben, wenn deren Rahmen zur Ersetzung ausgewählt wurde (kann nach der Freigabe der Seite erfolgen)
  - Auswahl eines Rahmens zur Ersetzung basiert auf Ersetzungsstrategie (z.B. MRU)
  - Pre-Fetch von gleichzeitig mehreren Seiten sinnvoll (z.B. bei Kenntnis über das Zugriffsmuster)
- DBMS vs. Betriebssystem-Unterstützung
  - DBMS benötigt Features, die in vielen BS nicht existieren,
    - z.B. Zwingen einer Seite auf Platte (“Force to Disk“)
    - Dateien über Plattengrenze
    - Vorhersage von Zugriffsmustern (unterstützt Prefetching und Seitenersetzung

Zusammenfassung (Forts.)

- Sätze variabler Länge mit Verzeichnis der Feld-Offsets unterstützen Direktzugriff auf das i-te Feld und erlauben Null-Werte
- Verschiebbare Sätze variabler Länge in einer Seite möglich (mit Directory aus Record-Zeigern)
- Dateiebene kontrolliert Seiten in einer Datei, Seite als Abstraktion einer Menge von Sätzen
  - Verkettete Liste von vollen Seiten und welchen mit Freispeicher
  - Directory-Struktur (Verwaltung der Seiten einer Datei)
- Index unterstützt effizientes Retrieval von Sätzen (basierend auf Werten der Felder)
- Relationen des Systemkatalogs speichern Informationen über Relationen, Indexe, Views usw. (Informationen, die für alle Sätze einer bestimmten Menge gleich sind)