

---

# Datenbanken

Prof. Dr. Ralf Möller

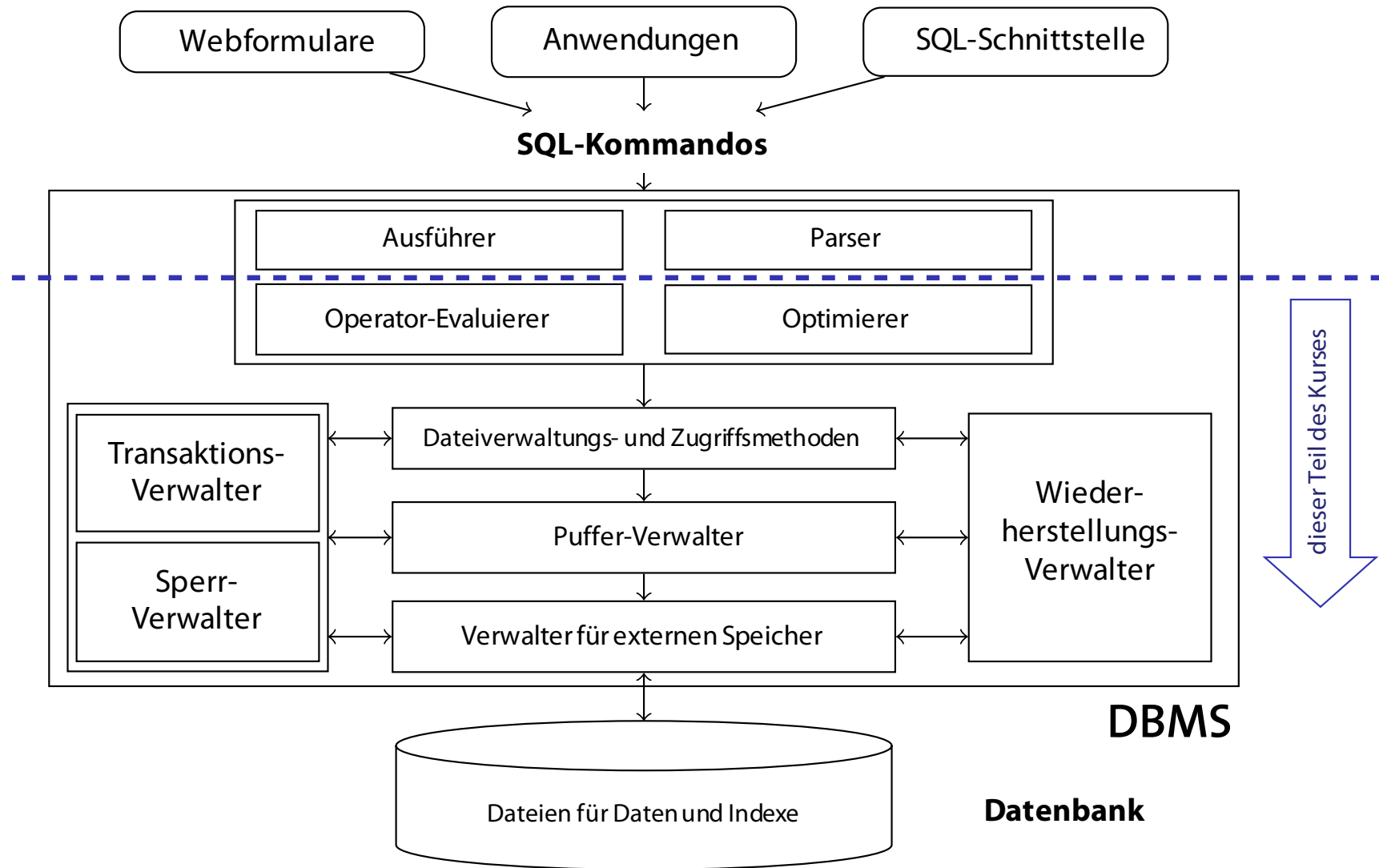
**Universität zu Lübeck**

**Institut für Informationssysteme**

Karsten Martiny (Übungen)



# Architektur eines DBMS



# Danksagung

---

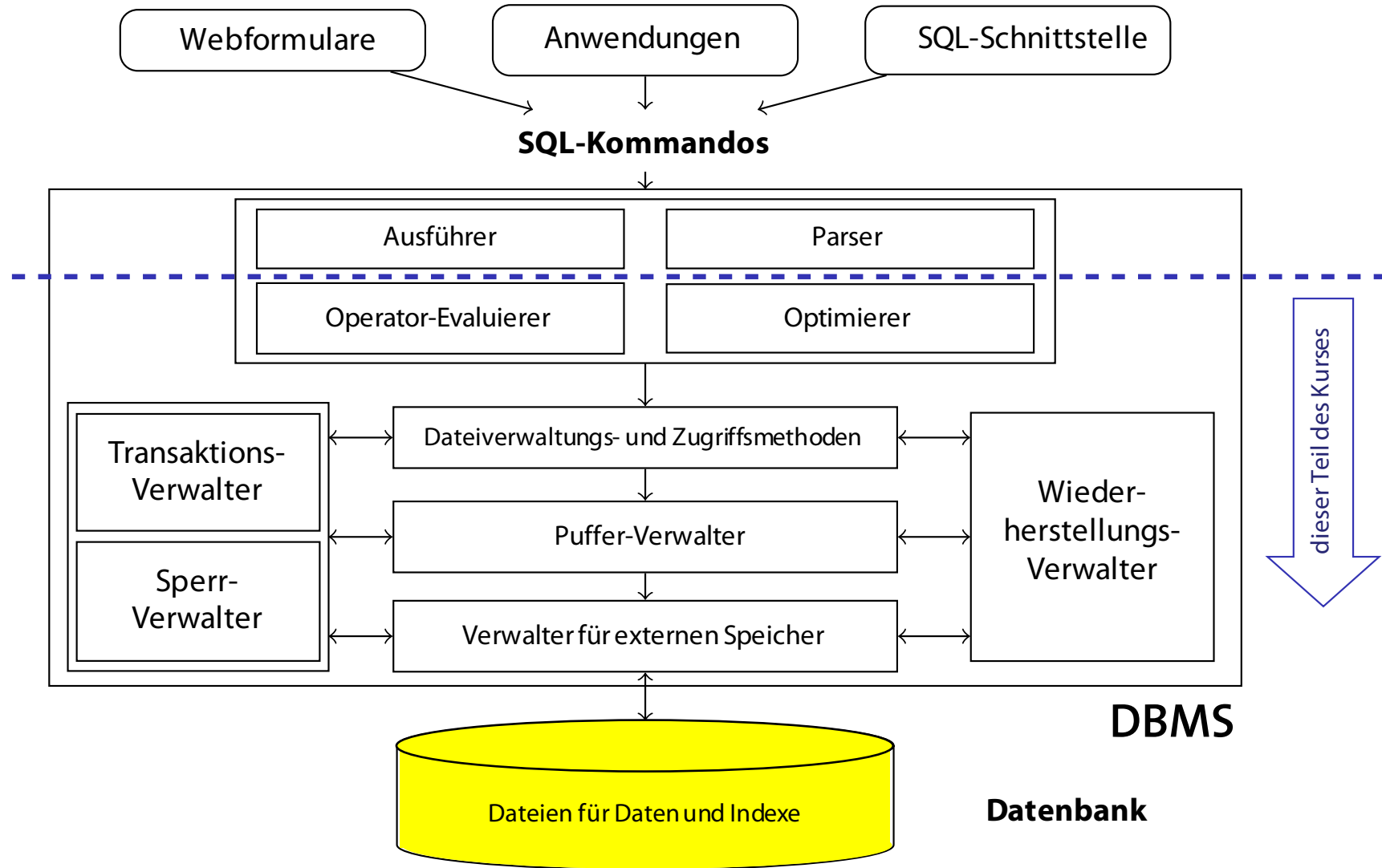
- Diese Vorlesung ist inspiriert von den Präsentationen zu dem Kurs:

„Architecture and Implementation of Database Systems“  
von Jens Teubner an der ETH Zürich

- Graphiken wurden mit Zustimmung des Autors aus diesem Kurs übernommen



# Speicher: Platten und Dateien



# Speicherhierarchie

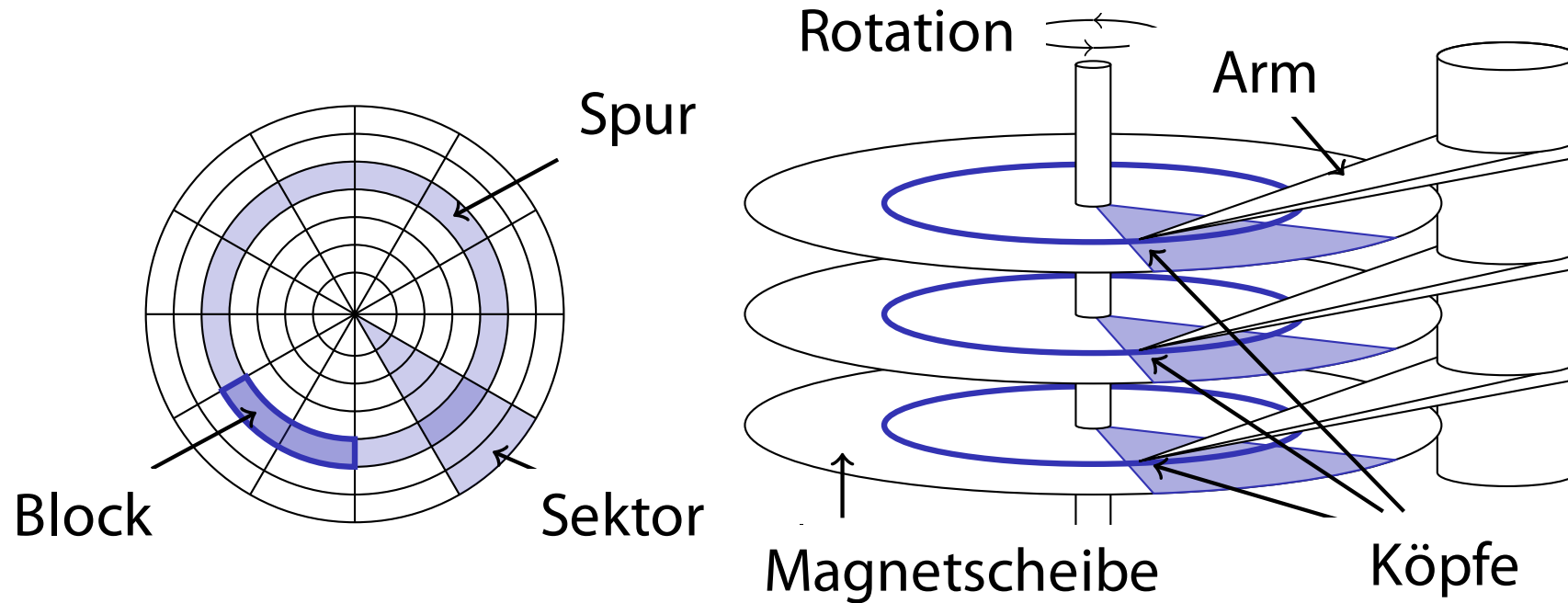
---

	Kapazität	Latenz
• CPU (mit Registern)	Bytes	< 1 ns
• Cache-Speicher	Kilo-/Mega-Bytes	< 10 ns
• Hauptspeicher	Giga-Bytes	20-100 ns
• Flash-Speicher	Giga/Tera-Bytes	30-250 $\mu$ s
• Festplatte	Tera-Bytes	3-10 ms
• Bandautomat	Peta-Bytes	variierend

- Zur CPU: Schnell aber klein
- Zur Peripherie: Langsam aber groß
- Cache-Speicher zur Verringerung der Latenz

# Magnetische Platten / Festplatten



- Schrittmotor positioniert Arme auf bestimmte Spur
- Magnetscheiben rotieren ständig
- Organisation in Blöcke
- Transfer erfolgt blockweise (lesend und schreibend)



Photo: <http://www.metalurgy.utah.edu/>

# Zugriffszeit

---

Konstruktion der Platten hat Einflüsse auf Zugriffszeit (lesend und schreibend) auf einen Block

1. Bewegung der Arme auf die gewünschte Spur (Suchzeit  $t_s$ )
2. Wartezeit auf gewünschten Block bis er sich unter dem Arm befindet (Rotationsverzögerung  $t_r$ )
3. Lesezeit bzw. Schreibzeit (Transferzeit  $t_{tr}$ )

Zugriffszeit:  $t = t_s + t_r + t_{tr}$

# Hitachi Travelstar 7K200 (für Laptops)

- 4 Köpfe, 2 Magnetplatten, 512 Bytes/Sektor,
- Kapazität: 200 GB
- Rotationsgeschwindigkeit: 7200 rpm
- Mittelere Suchzeit: 10 ms
- Transferrate: ca. 50 MB/s

Wie groß ist die Zugriffszeit auf einen Block von 8 KB?



# Sequentieller vs. Wahlfreier Zugriff

**Beispiel:** Lese 1000 Blöcke von je 8 KB

- **Wahlfreier Zugriff:**

- $t_{\text{rnd}} = 1000 \cdot 14.33 \text{ ms}$

- **Sequentieller Zugriff:**

- Travelstar 7k200 hat 63 Sektoren pro Spur, mit einer Track-to-Track-Suchzeit  $t_{s,\text{track-to-track}}$  von 1 ms

- Ein Block mit 8 KB benötigt 16 Sektoren

- $t_{\text{seq}} = t_s + t_r + 1000 \cdot t_{\text{tr}} + 1000 \cdot 16/63 \cdot t_{s,\text{track-to-track}}$   
 $= 10 \text{ ms} + 4.17 \text{ ms} + 160 \text{ ms} + 254 \text{ ms} \approx 428 \text{ ms}$

**Einsicht:** Sequentieller Zugriff **viel** schneller als wahlfreier Zugriff: Vermeide wahlfreie I/O wenn möglich

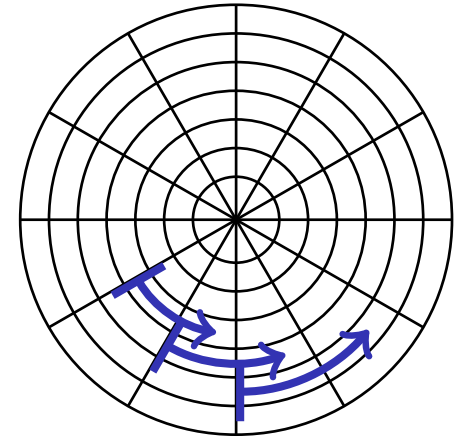
**Beobachtung:** Wenn  $428 \text{ ms} / 14330 \text{ ms} = 3\%$  einer 8MB Datei benötigt wird, kann man gleich die ganze Datei lesen

# Tricks zur Performanzsteigerung

---

## Spurverschiebung (track skewing)

Verschiebe Sektor 0 einer jeden Spur, so dass Rotationsverzögerung bei sequentielltem Abgriff minimiert wird



## Anfrageplanung (request scheduling)

Falls mehrere Blockanfragen befriedigt werden müssen, wähle die Anfrage, die die kleinste Armbewegung bedarf (SPTF: shortest positioning time first)

## Einteilung in unterschiedliche Zonen (zoning)

Mehr Sektoren in den längeren äußeren Spuren unterbringen

# Verbesserung der Festplattentechnologie

---

Latenz der Platten über die letzten 10 Jahre nur marginal verbessert ( $\approx 10\%$  pro Jahr)

## **Aber:**

- Durchsatz (Transferraten) um  $\approx 50\%$  pro Jahr verbessert
- Kapazität der Festplatten um  $\approx 50\%$  pro Jahr verbessert

## **Daher:**

- Kosten für wahlfreien Zugriff über die Zeit hinweg relativ gesehen immer bedeutsamer

# Wege zur Verbesserung der I/O-Performanz

---

Latenzproblem kaum zu vermeiden

## Aber:

- Durchsatz kann recht leicht gesteigert werden durch Ausnutzung von **Parallelität**
- Idee: Verwende mehrere Platten und greife parallel auf Daten zu

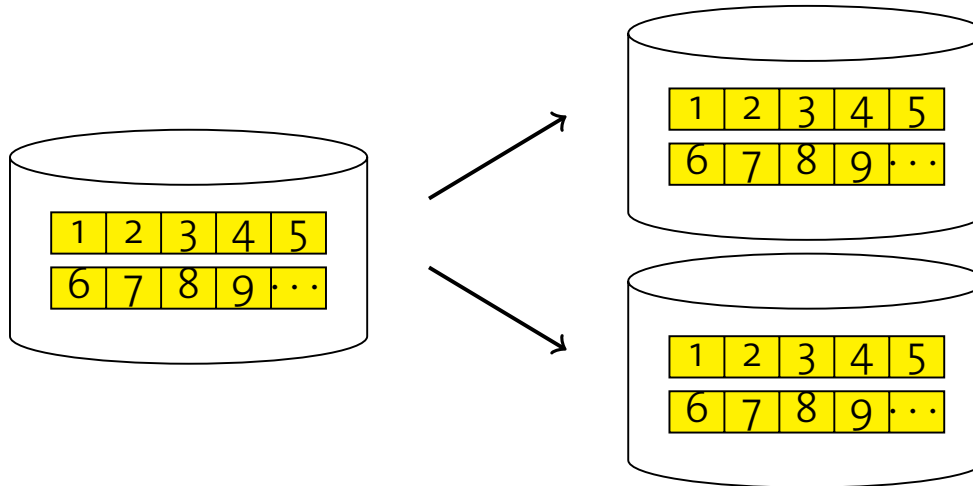
### **TPC-C: Ein Industrie-Laufzeittest für OLTP (V5.11)**

Kennzeichen des im Jahre 2013 besten Systems (Oracle 11g auf SPARC T5-8 Server):

- Server-CPU: SPARC T5 3,6 GHz, #Prozessoren: 8, #Kerne (total): 128
- Client-CPU: Intel Xeon E5-2690 2,9 GHz, #Clients: 8, #Proz. 32, #Kerne: 256
- In der Summe 8,5 Mio Transaktionen pro Minute
- Kosten: \$4.663.073 USD, mit \$0,55 USD/tpmC

# Spiegelung von Festplatteninhalten

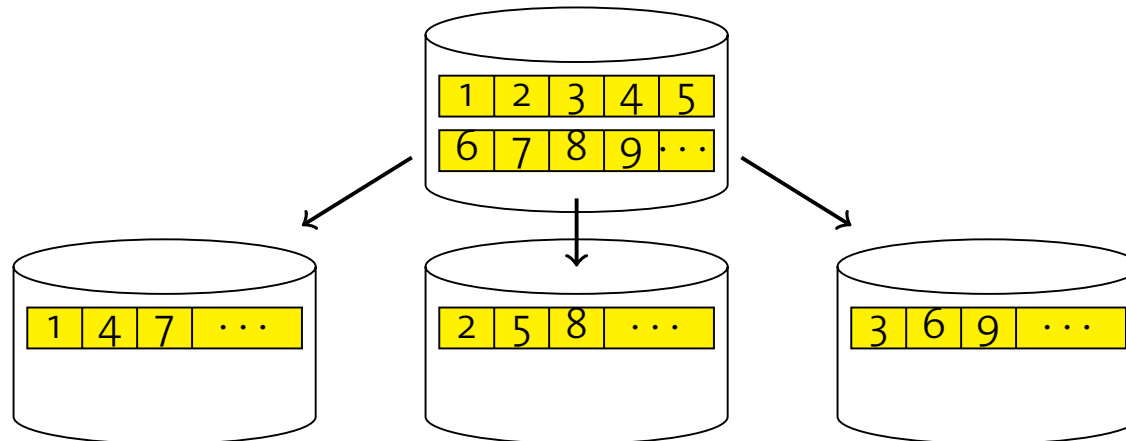
- Replizierung von Daten auf mehrere Platten



- I/O-Parallelität nur für Lesezugriffe
- Erhöhte Fehlertoleranz (überlebt Plattenfehler)
- Als RAID1 bekannt (Spiegelung ohne Parität)  
(RAID: Redundant Array of Inexpensive Disks)

# Speicherung mit Streifenbildung

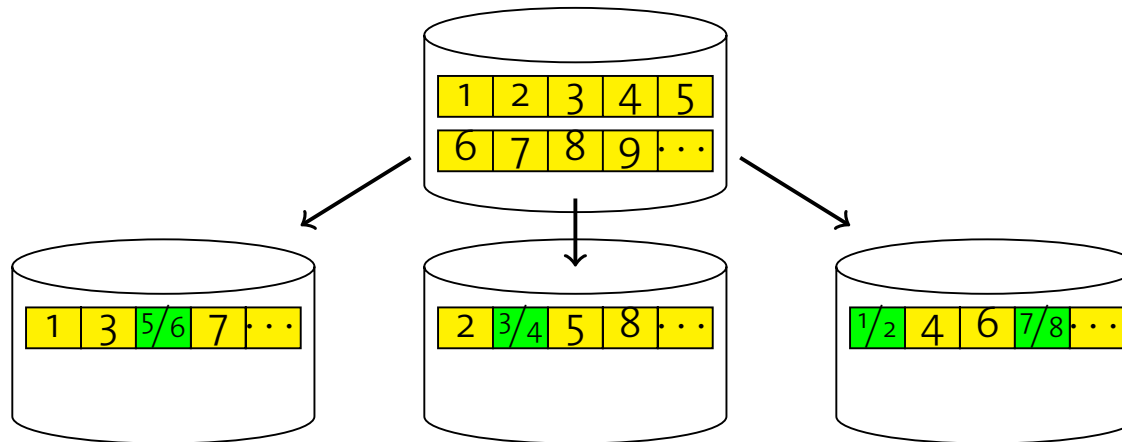
- Verteilung der Daten auf mehrere Platten



- Volle I/O-Parallelität
- Hohe Fehlerrate (hier: dreimal höheres Ausfallrisiko)
- Auch als RAID-0 bekannt (Streifenbildung ohne Parität)

# Streifenbildung mit Parität

- Verteile Daten und Paritätsinformation über Platten



- Hohe I/O-Parallelität
- Fehlertoleranz: Eine Platte kann ausfallen, ohne dass Daten verloren gehen
- RAID-5 (Streifenbildung mit verteilter Parität)

# Solid-State Disks als Alternative zur Festplatte

Anpassung von Datenbanken auf Geräteeigenschaften ist immer noch Forschungsgegenstand

	<b>MLC-NAND-Flash-Laufwerk 1,0" bis 3,5"</b>	<b>RAM-Disk als Teil des Arbeitsspeichers</b>	<b>Festplatte 1,0" bis 3,5"</b>
<b>Größe (keine Raidlaufwerke)</b>	bis 4 TB	bis 32 GB je Modul	bis 8 TB
<b>Preis pro TB (Stand Oktober 2014)</b>	ab $\approx 330$ € <sup>[7]</sup>	ab $\approx 4990$ € <sup>[105]</sup>	ab $\approx 27,30$ €
<b>Anschluss</b>	S-ATA, P-ATA, mSATA, PCIe	hauptsächlich DIMM-Connector	S-ATA, P-ATA, SCSI, SAS
<b>Lesen (kein RAID)</b>	bis 510 MB/s <sup>[106]</sup>	bis 51200 MB/s <sup>[107]</sup>	bis ca. 227 MB/s <sup>[108]</sup>
<b>Schreiben (kein RAID)</b>	bis 490 MB/s <sup>[106]</sup>	bis 51200 MB/s <sup>[107]</sup>	bis ca. 160 MB/s <sup>[108]</sup>
<b>Mittlere Zugriffszeit lesen</b>	ab 0,031 ms <sup>[109]</sup>	0,000.02 ms	ab 3,5 ms
<b>Mittlere Zugriffszeit schreiben</b>	ab 0,023 ms <sup>[109]</sup>	0,000.02 ms	ab 3,5 ms
<b>Überschreibbar (Zyklen)</b>	3 bis 10 tausendmal (MLC)	$> 10^{15}$ <sup>[110]</sup>	ca. 10 Mrd. (3 Jahre) <sup>[111]</sup>

Quelle: Wikipedia Solid-State Disk



# Netzwerk-Speicher ist kein Flaschenhals

---

- Durchsatz Festplatte (SSD):  $>500$  MB/s (Serial ATA)
- SDRAM: 50 Gbit/s (Latenz:  $\sim$  ns)
- Ethernet
  - 100-Gbit/s heute (Latenz:  $\sim$   $\mu$ s)
  - 400 Gbit/s erwartet in 2017

Warum also nicht Datenbank-Speicher über das Netzwerk referenzieren?

# Speichernetzwerk (Storage Area Network, SAN)

---

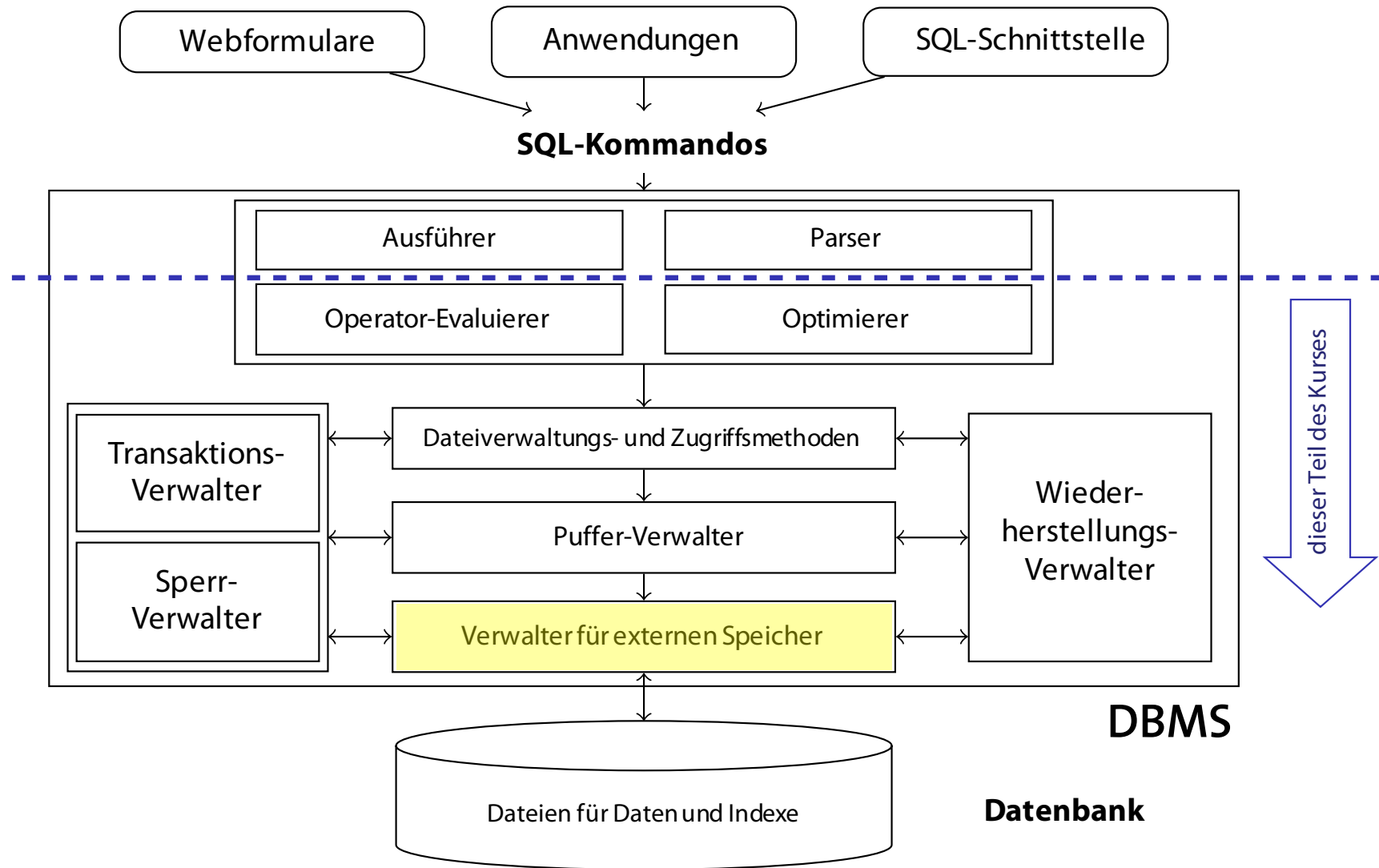
- Block-basierter Netzwerkzugriff auf Speicher
  - Als logische Platten betrachtet (Suche Block 4711 von Disk 42)
  - Nicht wie bei NFS (Network File System)
- SAN-Speichergeräte abstrahieren von RAID oder physikalischen Platten und zeigen sich dem DBMS als logische Platten
  - Hardwarebeschleunigung und einfachere Verwaltung
- Üblicherweise lokale Netzwerke mit multiplen Servern und Speicherressourcen
  - Bessere Fehlertoleranz und erhöhte Flexibilität

# Cloud-Speicher

---

- Cluster von vielen Standard-PCs (z.B. Google, Amazon)
  - Systemkosten vs. Zuverlässigkeit und Performanz
  - Verwendung massiver Replikation von Datenspeichern
- CPU-Zyklen und Disk-Kapazität als Service
  - Amazons „Elastic Compute Cloud (EC<sub>2</sub>)“
    - Kosten pro Stunde <10 Cent
  - Amazons „Simple Storage System (S3)“
    - „Unendlicher“ Speicher für Objekte in einer Größe zwischen 1 Byte und 5 GB mit Key-Value-Struktur
      - Latenz: 100 ms bis 1s
- Datenbank auf Basis von S3 entwickelt in 2008

# Architektur eines DBMS



# Verwaltung des externen Speichers

---

- Abstraktion von technischen Details der Speichermedien
- Konzepte der Seite (page) mit typischerweise 4-64KB als Speichereinheiten für die restlichen Komponenten
- Verzeichnis für Abbildung

Seitennummer → Physikalischer Speicherort

wobei der physikalische Speicherort

- eine Betriebssystemdatei inkl. Versatz,
- eine Angabe Kopf-Sektor-Spur einer Festplatte oder
- eine Angabe für Bandgerät und -nummer inkl. Versatz

sein kann

# Verwaltung leerer Seiten

---

## Verwendete Techniken:

1. Liste der freien Seiten
  - Hinzufügung falls Seite nicht mehr verwendet
2. Bitmap mit einem Bit für jede Seite
  - Umklappen des Bits  $k$ , wenn Seite  $k$  (de-)alloziert wird

# Aufgabe

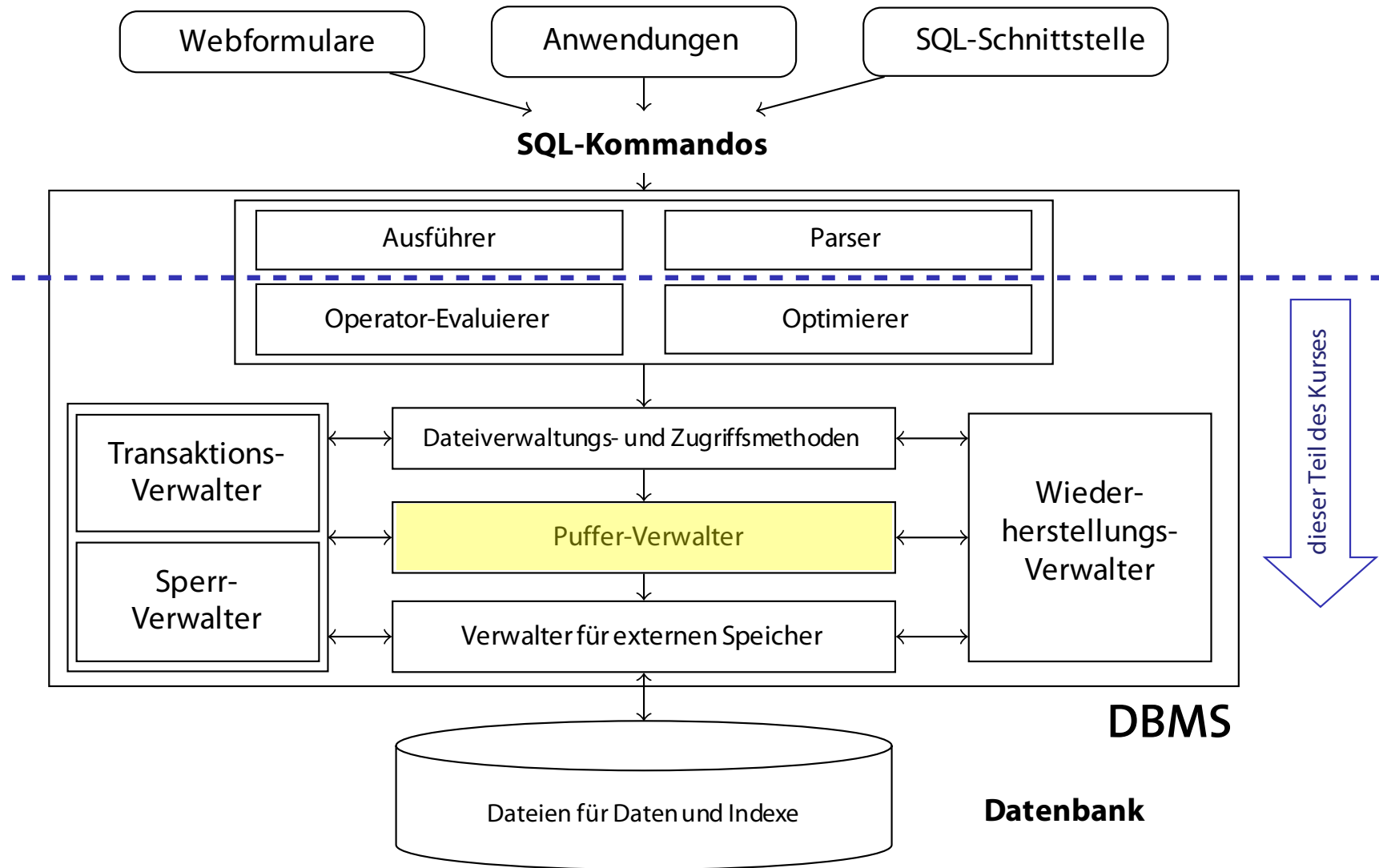
Verwendete Techniken:

1. Liste der freien Seiten
  - Hinzufügung falls Seite nicht mehr verwendet
2. Bitmap mit einem Bit für jede Seite
  - Umklappen des Bits  $k$ , wenn Seite  $k$  (de-)alloziert wird

Zur Erhöhung des sequentiellen Zugriffs sollten hintereinanderliegende Seiten verwendet werden.

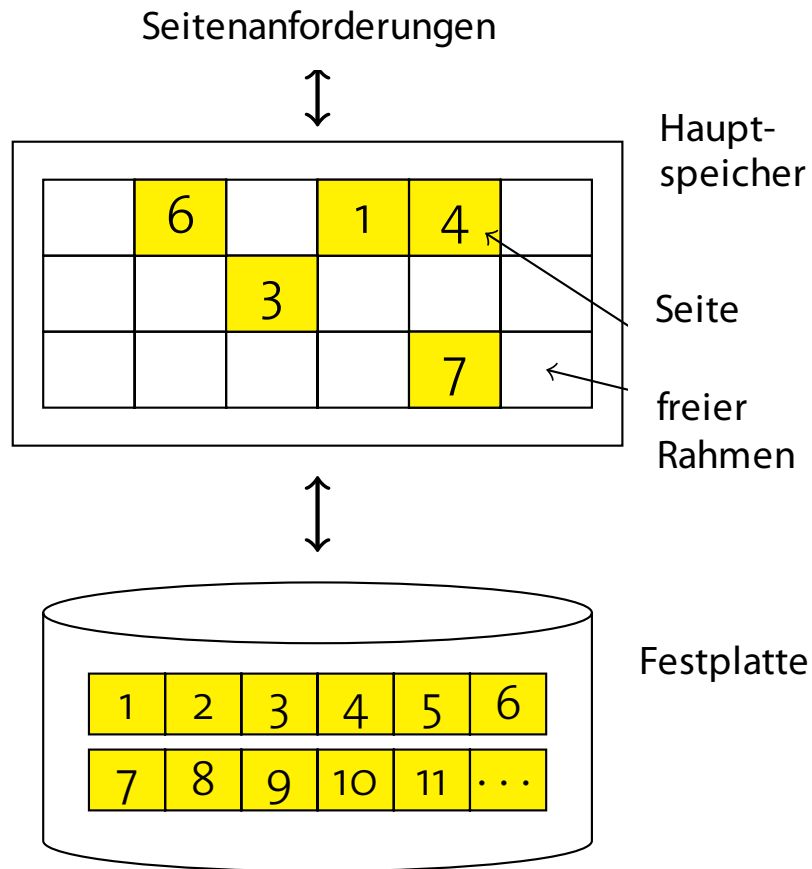
Welche Technik, 1. oder 2., würden Sie wählen, um dieses zu unterstützen?

# Architektur eines DBMS





# Puffer-Verwalter



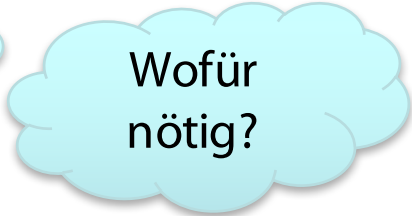
- Vermittelt zwischen externem und internem Speicher (Hauptspeicher)
- Verwaltet hierzu einen besonderen Bereich im Hauptspeicher, den Pufferbereich (buffer pool)
- Externe Seiten in Rahmen des Pufferbereichs laden
- Verdrängungsstrategie falls Pufferbereich voll

# Schnittstelle zum Puffer-Verwalter

---

Funktion **pin** für Anfragen nach Seiten und **unpin** für Freistellungen von Seiten nach Verwendung

- **pin(pageno)**
  - Anfrage nach Seitennummer pageno
  - Lade Seite in Hauptspeicher falls nötig
  - Rückgabe einer Referenz auf pageno
- **unpin(pageno, dirty)**
  - Freistellung einer Seite pageno zur möglichen Auslagerung
  - dirty = true bei Modifikationen der Seite



Wofür  
nötig?

# Implementation von pin()

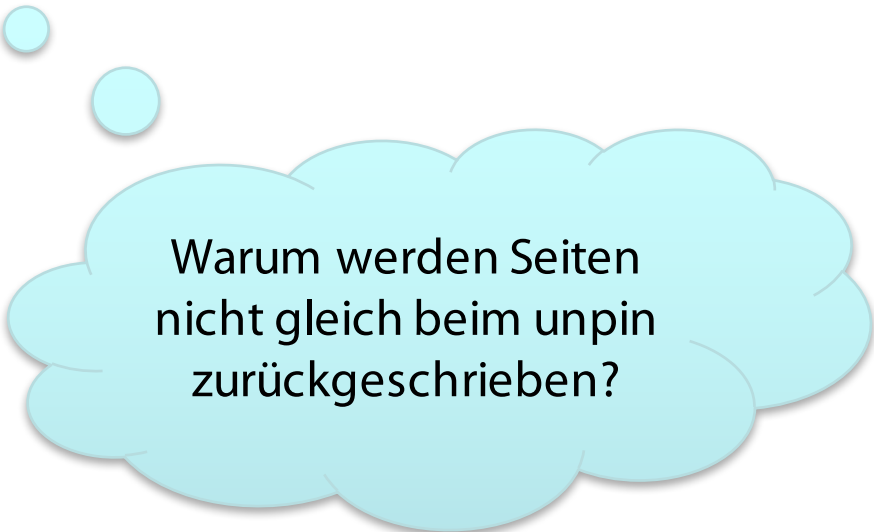
---

```
1 Function: pin(pageno)
2 if buffer pool already contains pageno then
3   |   pinCount (pageno) ← pinCount (pageno) + 1;
4   |   return address of frame holding pageno ;
5 else
6   |   select a victim frame v using the replacement policy ;
7   |   if dirty (v) then
8   |     | write v to disk ;
9   |   read page pageno from disk into frame v ;
10  |   pinCount (pageno) ← 1 ;
11  |   dirty (pageno) ← false ;
12  |   return address of frame v ;
```

# Implementation von unpin()

---

```
1 Function: unpin(pageno, dirty)  
2 pinCount (pageno) ← pinCount (pageno) - 1;  
3 if dirty then  
4   | dirty (pageno) ← dirty;
```



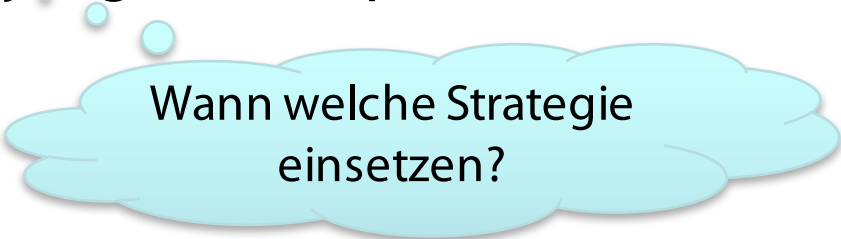
Warum werden Seiten  
nicht gleich beim unpin  
zurückgeschrieben?

# Verdrängungsstrategien

---

Die Effektivität des Puffer-Verwalters hängt von der gewählten Verdrängungsstrategie ab, z.B.:

- **Least Recently Used (LRU)**
  - Verdrängung der Seite mit längszurückliegendem unpin()
- **LRU-k**
  - Wie LRU, aber k-letztes unpin(), nicht letztes
- **Most Recently Used (MRU)**
  - Verdrängung der Seite mit jüngstem unpin()
- **Random**
  - Verdrängung einer beliebigen Seite



Wann welche Strategie einsetzen?

# Pufferverwaltung in der Praxis

---

- Prefetching
  - Antizipation von Anfragen, um CPU- und I/O-Aktivität zu überlappen
    - Spekulatives Prefetching: Nehme sequentiellen Seitenzugriff an und lese im Vorwege
    - Prefetch-Listen mit Instruktionen für den Pufferverwalter für Prefetch-Seiten
- Fixierungs- oder Verdrängungsempfehlung
  - Höherer Code kann Fixierung (z.B. für Indexseiten) oder schnelle Verdrängung (bei seq. Scans) empfehlen
- Partitionierte Pufferbereiche
  - Z.B. separate Bereiche für Index und Tabellen

# Datenbanken vs. Betriebssysteme

---

- Haben wir nicht gerade ein Betriebssystem entworfen?
- Ja
  - Verwaltung für externen Speicher und Pufferverwaltung ähnlich
- Aber
  - DBMS weiß mehr über Zugriffsmuster (z.B. Prefetching)
  - Limitationen von Betriebssystemen häufig zu stark für DBMS (Obergrenzen für Dateigrößen, Plattformunabhängigkeit nicht gegeben)

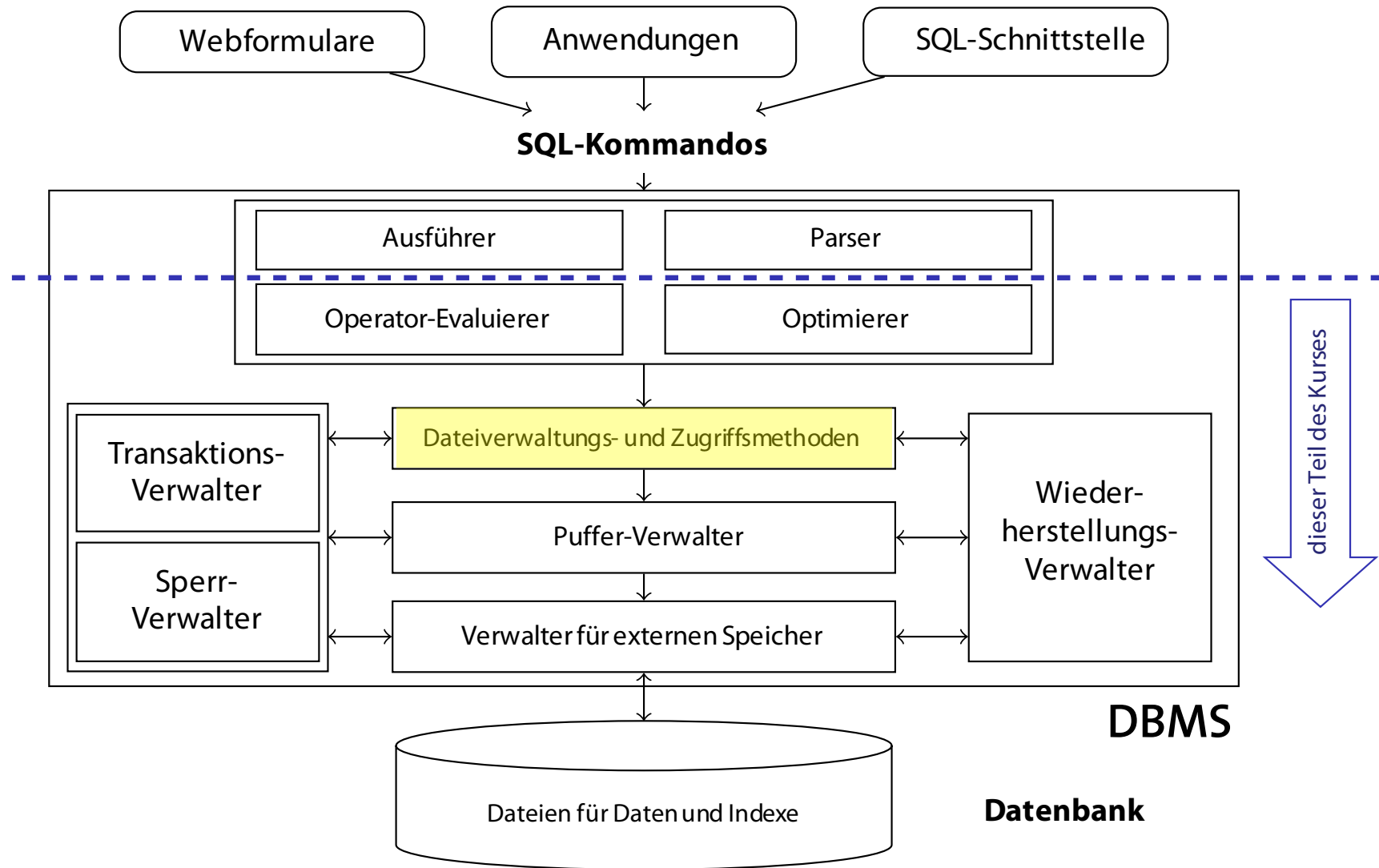
# Datenbanken vs. Betriebssysteme

---

- Gegenseitige Störung möglich
  - Doppelte Seitenverwaltung
  - DMBS-Transaktionen vs. Transaktionen auf Dateien organisiert vom Betriebssystem (journaling)
  - DBMS Pufferbereiche durch Betriebssystem ausgelagert
  - DBMS schalten Betriebssystemdienste aus
    - Direkter Zugriff auf Festplatten
    - Eigene Prozessverwaltung
    - ...

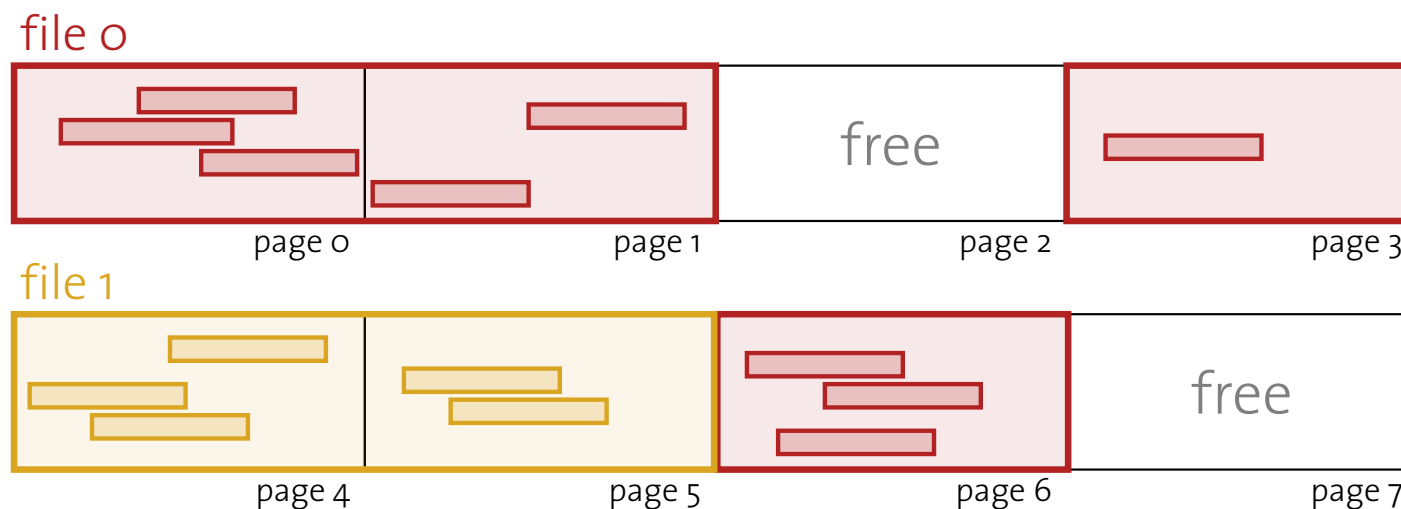


# Architektur eines DBMS



# Datenbank-Dateien

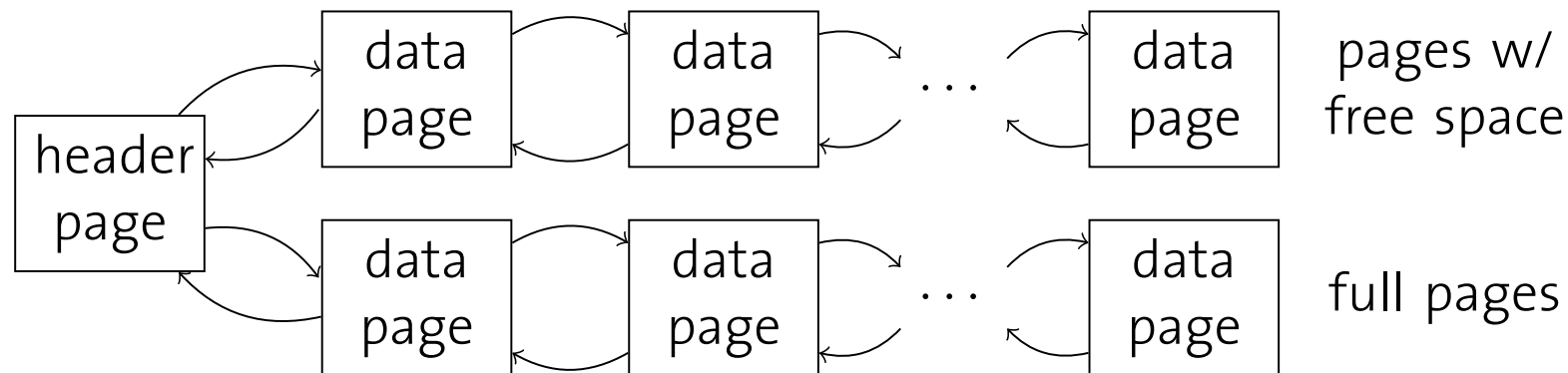
- Seitenverwaltung unbeeinflusst vom Inhalt
- DBMS verwaltet Tabellen von Tupeln, Indexstrukturen, ...
- Tabellen sind Dateien von Datensätzen (records)
  - Datei besteht aus einer oder mehrerer Seiten
  - Jede Seite speichert eine oder mehrere Datensätze
  - Jeder Datensatz korrespondiert zu einem Tupel



# Heap-Dateien

- Wichtigster Dateityp: Speicherung von Datensätzen mit willkürlicher Ordnung (konform mit SQL)

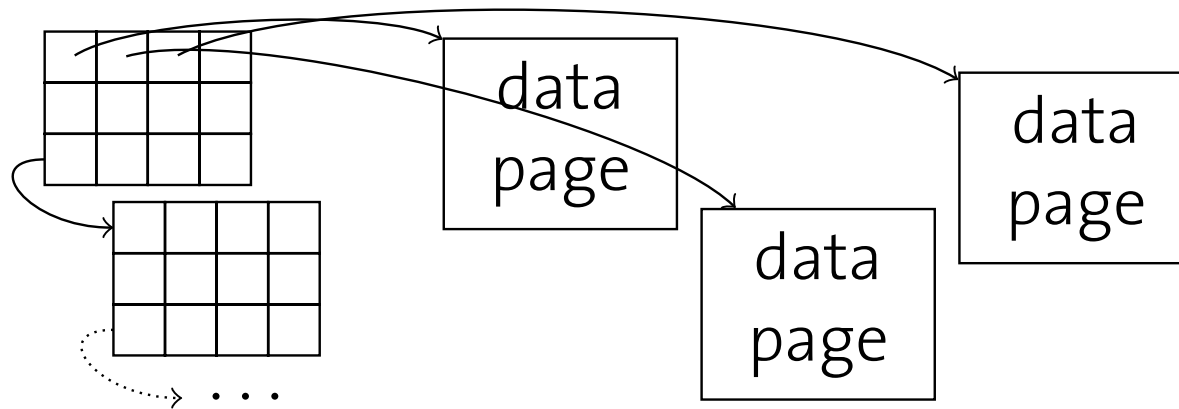
## Verkettete Liste von Seiten



- ✓ Einfach zu implementieren
- ✗ Viele Seiten auf der Liste der freie Seiten (haben also noch Kapazität)
- ✗ Viele Seiten anzufassen bis passende Seite gefunden

# Heap-Dateien

- Verzeichnis von Seiten



- ✗ Verwendung als Abbildung mit Informationen über freie Plätze (Granularität ist Abwägungssache)
- ✓ Suche nach freien Plätzen effizient
- ✗ Zusatzaufwand für Verzeichnisspeicher

# Freispeicher-Verzeichnis

---

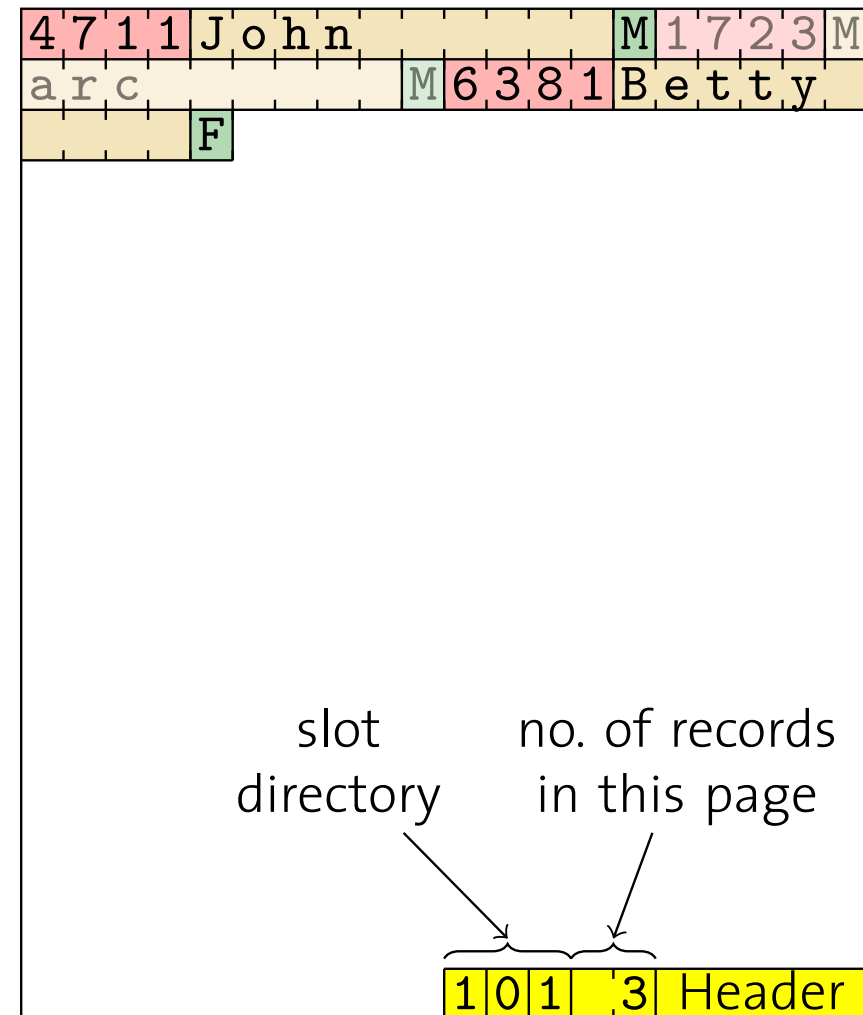
Welche Seite soll für neuen Datensatz gewählt werden?

- **Append Only**
  - Immer in letzte Seite einfügen, sonst neue Seite anfordern
- **Best-Fit**
  - Alle Seiten müssen betrachtet werden, Reduzierung der Fragmentierung
- **First-Fit**
  - Suche vom Anfang, nehme erste Seite mit genug Platz
  - Erste Seiten füllen sich schnell, werden immer wieder betrachtet
- **Next-Fit**
  - Verwalte Zeiger und führe Suche fort, wo Suche beim vorigen Male endete

# Inhalt einer Seite

ID	NAME	SEX
4711	John	M
1723	Marc	M
6381	Betty	F

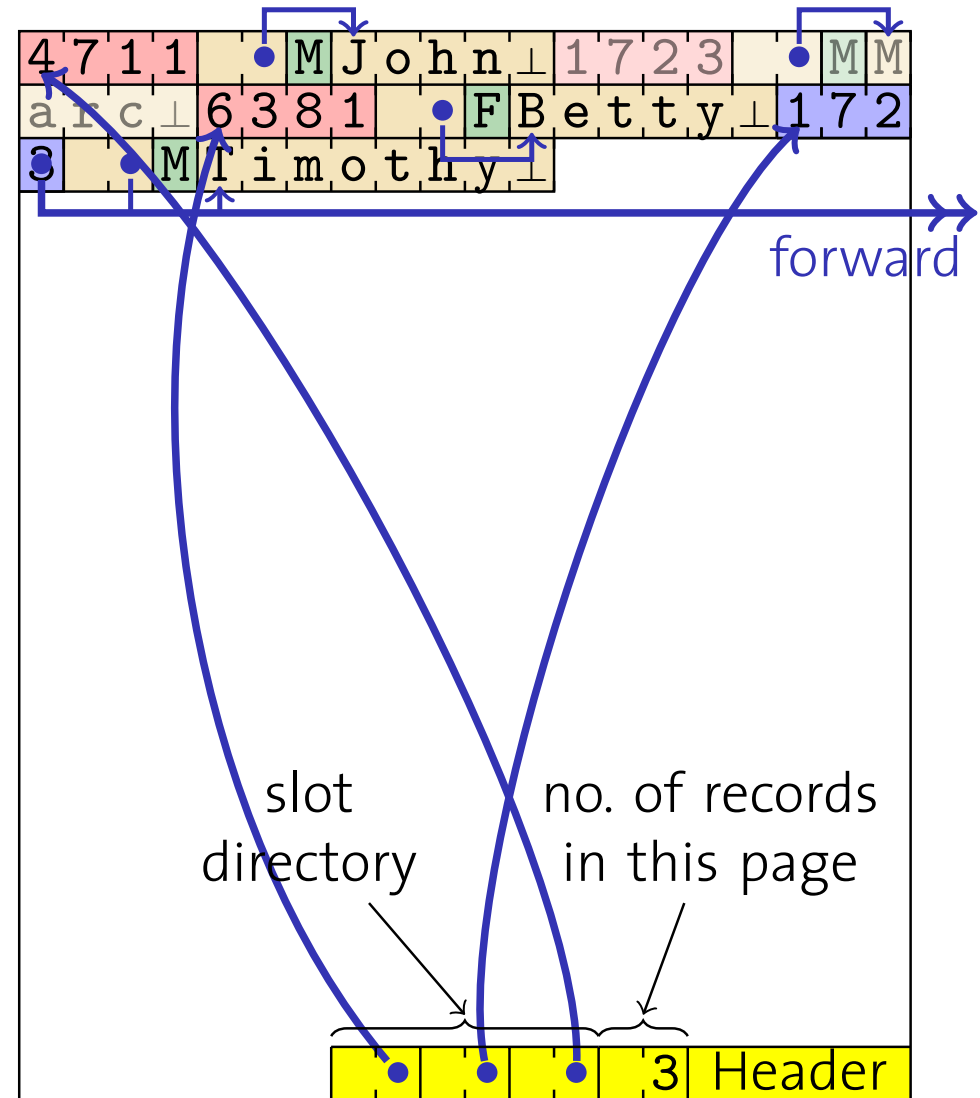
- Datensatz-Kennung  
(record identifier, rid)
- Datensatz-Position  
(Versatz auf der Seite)  
Slotno x Bytes pro Slot
- Datensatz gelöscht?
  - rid sollte sich nicht ändern
  - Slot-Verzeichnis (Bitmap)



# Inhalte einer Seite: Felder variabler Länge

- Felder variabler Länge zum Ende verschoben
  - Platzhalter zeigt auf Position
- Slot-Verzeichnis zeigt auf Start eines Feldes
- Felder können auf Seite verschoben werden (z.B. wenn sich Feldgröße ändert)
- Einführung einer Vorwärtsreferenz, wenn Feld nicht auf Seite passt

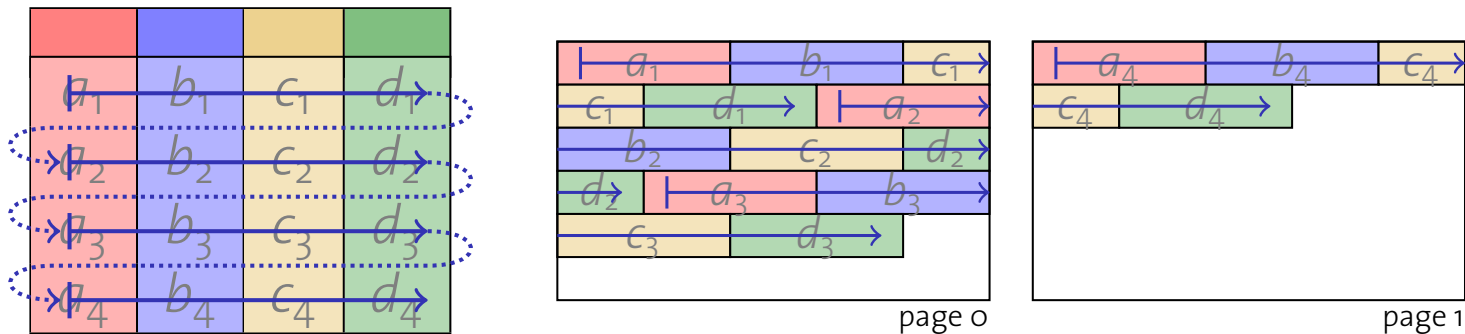
Warum?



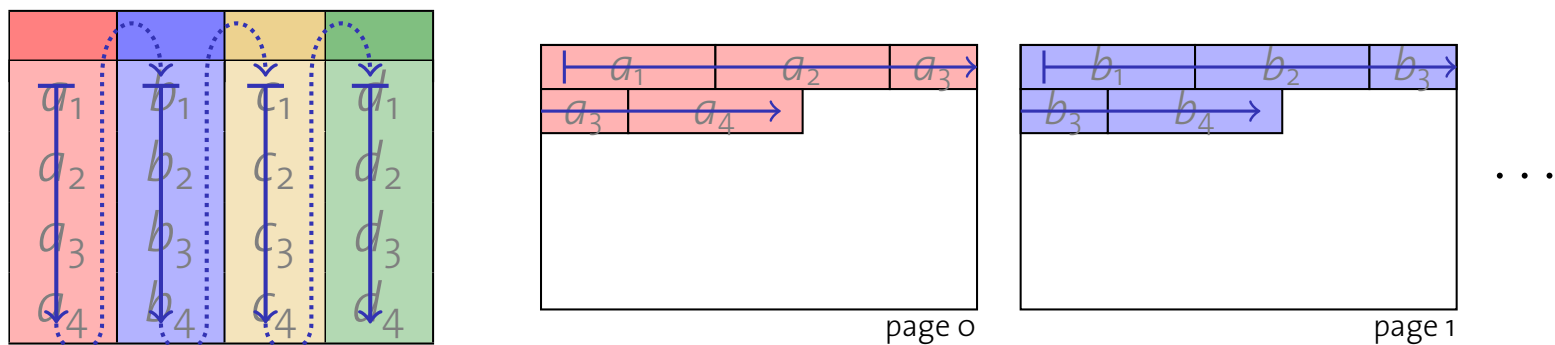
Was passiert bei Updates?

# Alternative Seiteneinteilungen

- Im Beispiel wurden Datensätzen zeilenweise angeordnet:



- Spaltenweise Anordnung genauso möglich:

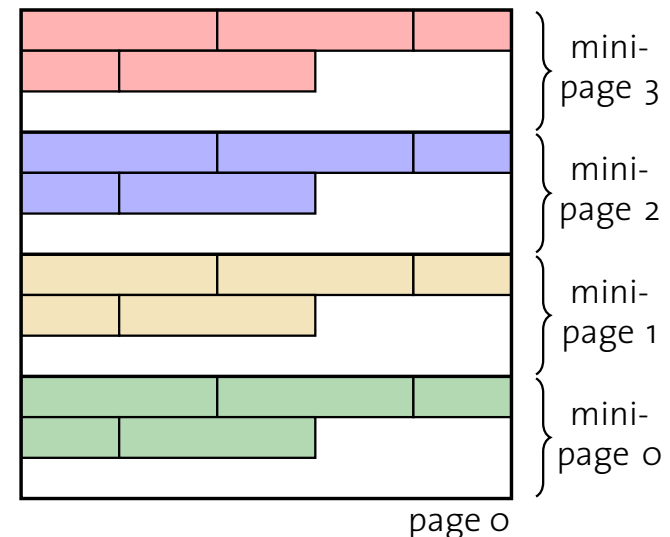




# Alternative Seitenanordnungen

Vorgestellte Schemata heißen auch:

- Row-Store
- Column-Store
- Anwendungen für verschiedene Lasttypen und Anwendungskontexte (z.B. OLAP)
- Unterschiedliche Kompressionsmöglichkeiten
- Kombination möglich:
  - Unterteilung einer Seite in Miniseiten
  - mit entsprechender Aufteilung



# Zusammenfassung

---

- **Kennzeichen von Speichermedien**
  - Wahlfreier Zugriff langsam (I/O-Komplexität)
- **Verwalter für externen Speicher**
  - Abstraktion von Hardware-Details
  - Seitennummer → Physikalischer Speicherort
- **Puffer-Verwalter**
  - Seiten-Caching im Hauptspeicher
  - Verdrängungsstrategie
- **Dateiorganisation**
  - Stabile Record-Bezeichner (rids)
  - Verwaltung statischer und dynamischer Felder

