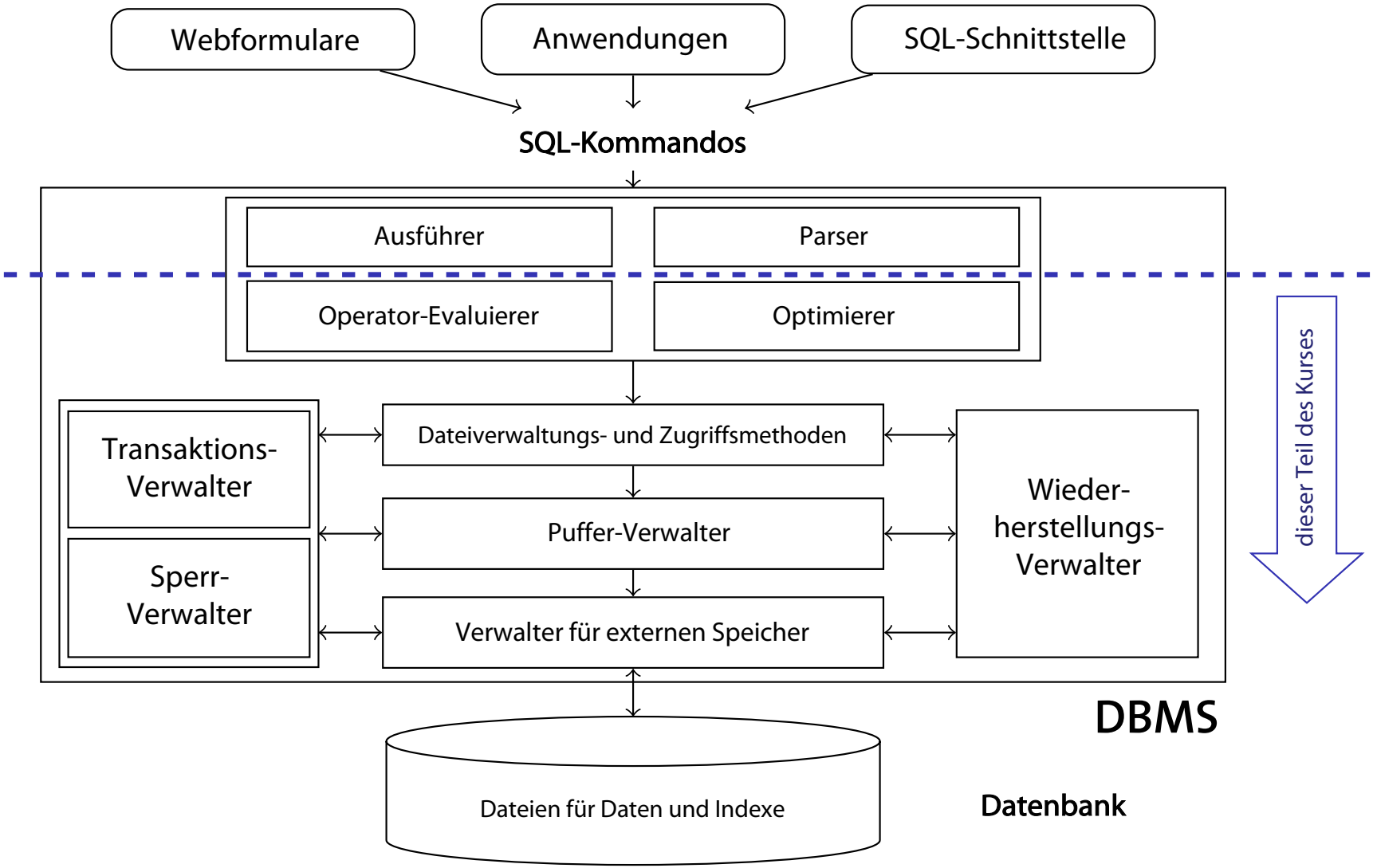
Datenbanken

Dr. Özgür Özçep
Universität zu Lübeck
Institut für Informationssysteme

Architektur eines DBMS

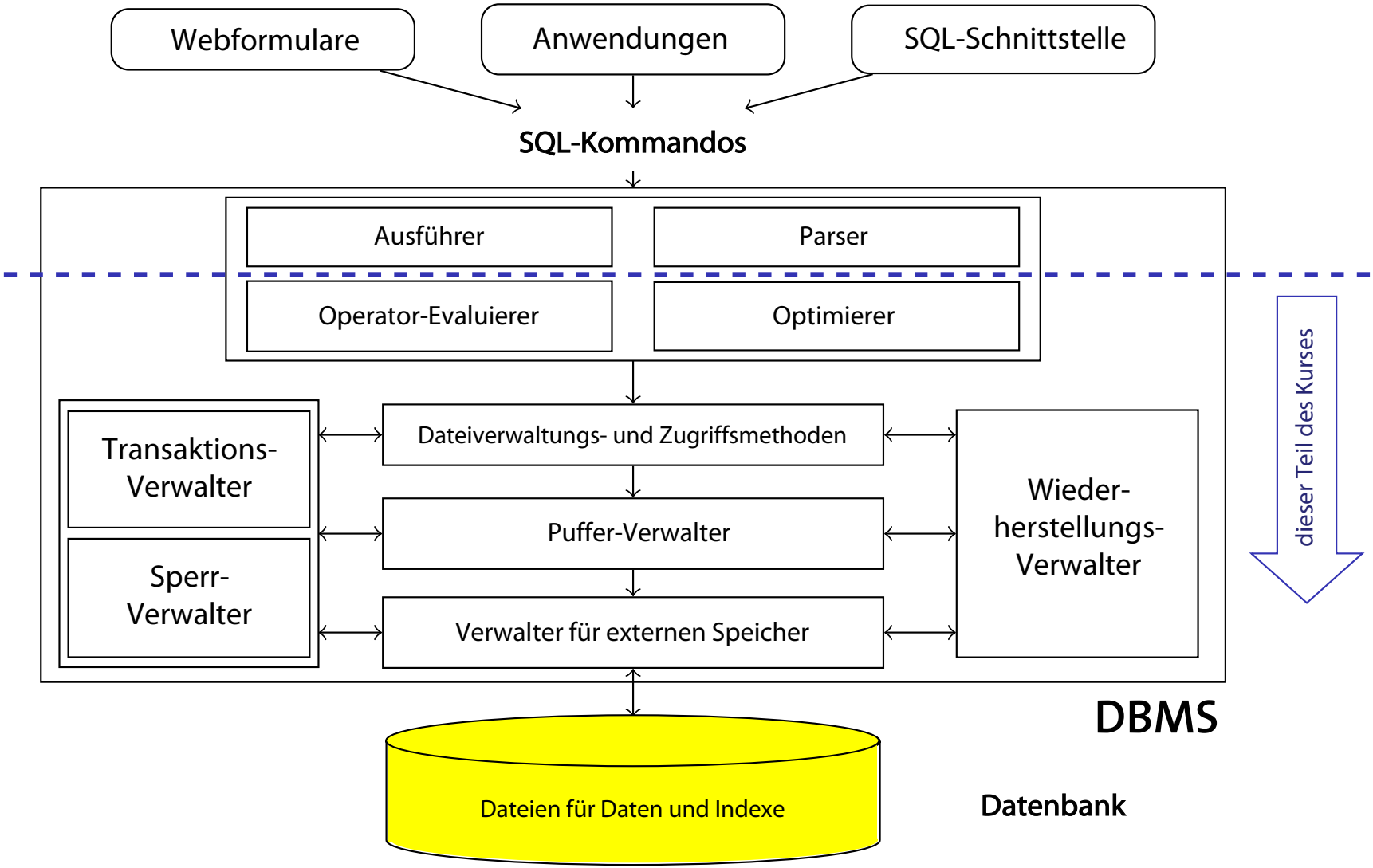


Danksagung

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

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

Speicher: Platten und Dateien

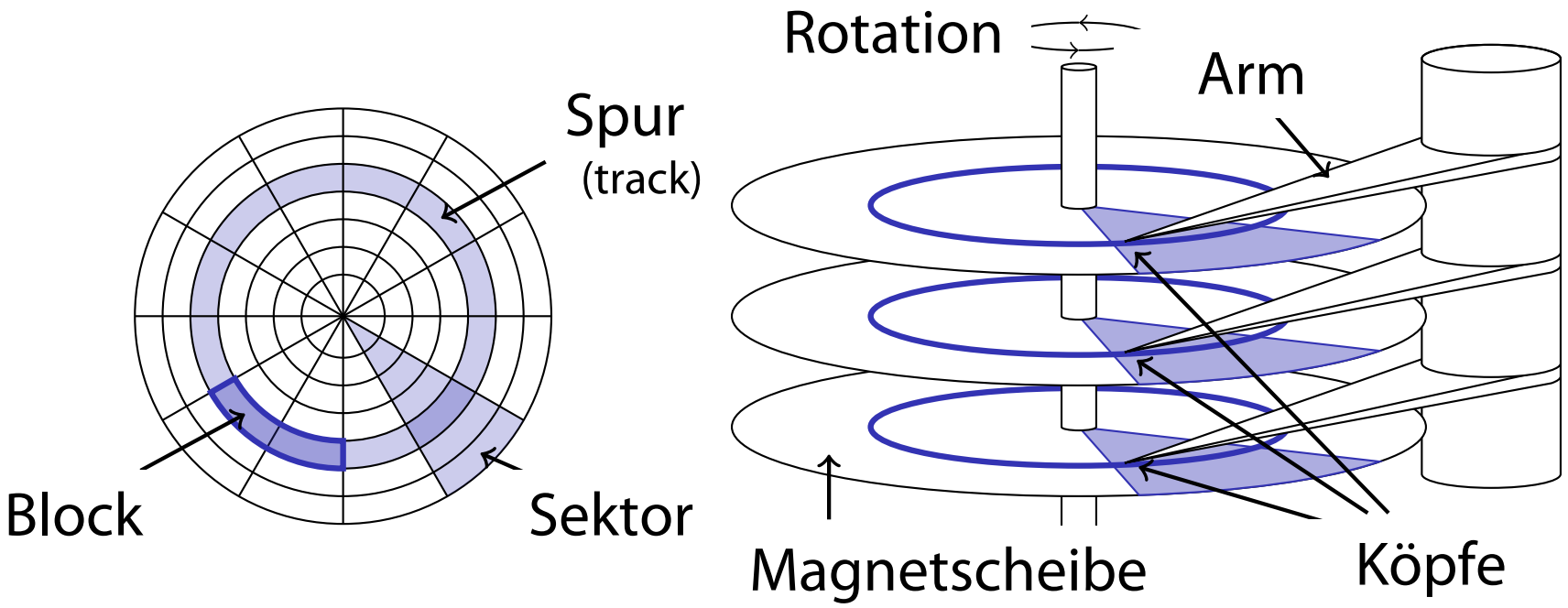


Speicherhierarchie

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

- Zur CPU: Schnell aber klein
- Zur Peripherie: Langsam aber groß
- Cache-Speicher zur Verringerung der Latenz
- Blockweises Lesen/Schreiben ab Flash/SSD (Block etwa 4K)

Magnetische Platten / Festplatten



- Schrittmotor positioniert Arme auf bestimmte Spur
- Magnetscheiben rotieren ständig

Photo: http://www.metallurgy.utah.edu/

	MLC-NAND-Flash-Laufwerk 1,0″ bis 3,5″	RAM-Disk als Teil des Arbeitsspeichers	Festplatte 1,0 bis 3,5″
Größe (keine Raidlaufwerke)	bis 16 TB	bis 32 GB je Modul	bis 12 TB
Preis pro TB (Stand Mai 2016)	ab ≈ 218 €[72]	ab ≈ 2.690 €[73]	ab ≈ 26 €[74]
Anschluss	IDE/(P)ATA, SATA, mSATA, PCIe, M.2	hauptsächlich DIMM-Connector	SCSI, IDE/(P)ATA, SATA, SAS
Lesen (kein RAID)	bis 510 MB/s[75]	bis 51.200 MB/s[76]	bis ca. 227 MB/s[77]
Schreiben (kein RAID)	bis 490 MB/s[75]	bis 51.200 MB/s[76]	bis ca. 160 MB/s[77]
Mittlere Zugriffszeit lesen	ab 0,031 ms[78]	0,000.02 ms	ab 3,5 ms
Mittlere Zugriffszeit schreiben	ab 0,023 ms[78]	0,000.02 ms	ab 3,5 ms
Überschreibbar (Zyklen)	3 bis 10 tausendmal (MLC)	$> 10^{15}$ [79]	ca. 10 Mrd. (3 Jahre)[80]
Lagerbar bei	–45–85 °C	–25–85 °C	–40–70 °C
Stoßfestigkeit – Betrieb[81]	1.500 *g*	ca. 1.000 *g* (rüttelfest verlötet)	60 *g*
Stoßfestigkeit – Lagerung	1.500 *g*	ca. 1.000 *g* (ähnlich SSD)	350 *g*
Verbrauch – Ruhe	0,1–1,3 W	1 W pro SDRAM-Modul	4 W und höher
Verbrauch – Zugriff	0,5–5,8 W[82]	8 W pro SDRAM-Modul	6 W und höher
Verhalten beim Herunterfahren	problemlos	Datenverlust, falls keine Sicherung auf SSD/Festplatte stattfindet	problemlos
Verhalten bei Stromausfall	mit Stützkondensator problemlos,[83] sonst Datenverlust möglich	Datenverlust	Datenverlust möglich
Lautlos	Geräusche möglich bei Defekt[84][85]	ja	nein
Bemerkungen	unterstützen meistens S.M.A.R.T.	Größe begrenzt durch Hauptplatine oder Adapter nötig, nicht bootfähig	unterstützen S.M.A.R.T.

Quelle: Wikipedia Solid-State Drive
S.M.A.R.T. = Verschleiß- und Ausfallsvorhersage

Zugriffszeit bei Festplatten

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. Schreibezeit (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
- Mittlere Suchzeit: 10 ms
- Transferrate: ca. 50 MB/s

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

Hitachi Travelstar 7K200 (für Laptops)

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

Antwort:

- t_s = 10ms
- t_r = (1/7200 rpm) * 60 * 1000 * ½ ms= 4,17ms
- t_{tr} = 8KB/(50 * 1024 KB)* 1000ms = 0,16ms

- Also: t = 10ms + 4,17ms + 0,16ms = 14,33ms

Sequentieller vs. Wahlfreier Zugriff

Beispiel: Lese 1000 Blöcke von je 8 KB

- Wahlfreier Zugriff:
  - $t_{rnd} = 1000 \cdot 14{,}33\ ms = 14330ms$
- Sequentieller Zugriff:
  - Travelstar 7K200 hat 63 Sektoren pro Spur, mit einer Track-to-Track-Suchzeit $t_{s,track\text{-}to\text{-}track}$ von 1 ms
  - Ein Block mit 8 KB benötigt 16 Sektoren
  - $tseq = t_s + t_r + 1000 \cdot t_{tr} + 16 \cdot 1000/63 \cdot t_{s,track\text{-}to\text{-}track}$
    $= 10\ ms + 4{,}17\ ms + 160\ ms + 254\ ms \approx 428\ ms$

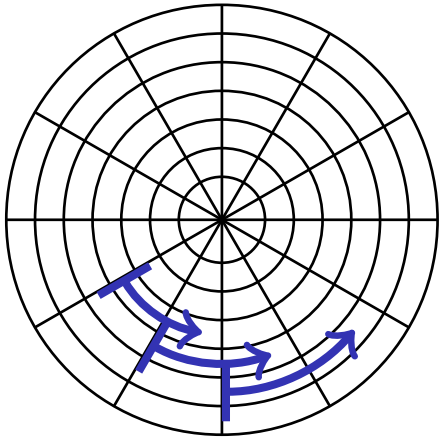
Einsicht: Sequentieller Zugriff **viel** schneller als wahlfreier Zugriff: Vermeide wahlfreie I/O wenn möglich

Wenn 428 ms / 14330 ms = 3% einer 8MB Datei wahlfrei benötigt wird, kann man gleich die ganze Datei lesen, sofern Blöcke hintereinander stehen.

Tricks zur Performanzsteigerung

Spurverschiebung (track skewing)

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



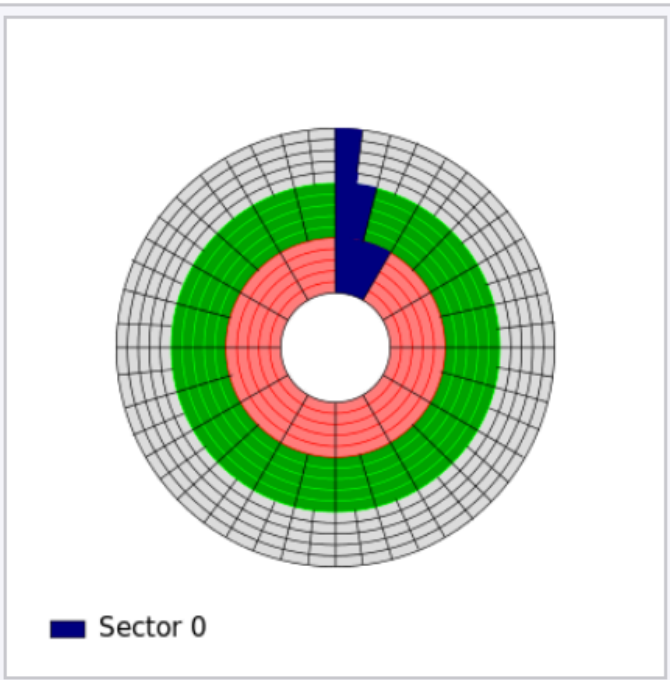
Anfrageplanung (request scheduling)

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

Einteilung in unterschiedliche Zonen (zoning)

Mehr Sektoren in den längeren äußeren Spuren
unterbringen

https://en.wikipedia.org/wiki/Zone_bit_recording



Physical layout of sectors in a zone-bit disc: As distance from the centre increases, the number of sectors in a given angle increases from one (red) to two (green) to four (grey).

JAN SCHAUMANN - http://panix.netmeister.org/tmp/DiskStructure-ZBR.svg

Verbesserung der Festplattentechnologie

Latenz (Suchzeit und Rotationsverzögerung) der Platten über die letzten 10 Jahre
nur marginal verbessert (≈ 10% pro Jahr)

Aber:

- Durchsatz (Transferraten) um ≈ 50% pro Jahr verbessert
- Kapazität der Festplatten um ≈ 50% pro Jahr verbessert

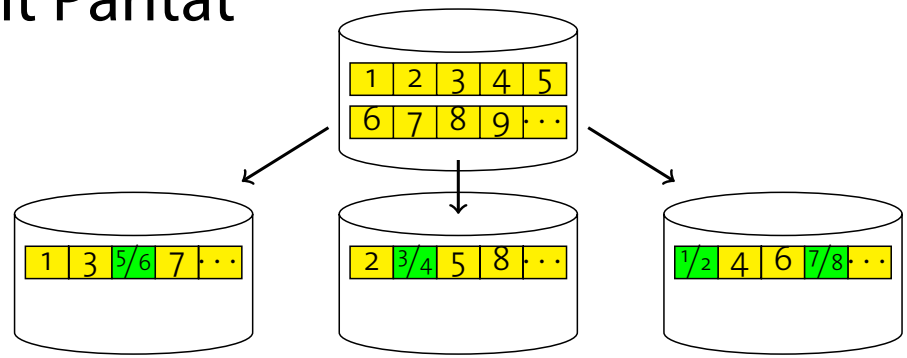
Daher:

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

Werden Festplatten durch SSDs ersetzt?

Wege zur Verbesserung der I/O-Performanz

- Latenzproblem kaum zu vermeiden
  - Durchsatz kann recht leicht gesteigert werden durch Ausnutzung von Parallelität
  - Z.B. durch Streifenbildung mit Parität (RAID-5)



- Transaction Processing Performance Council, kurz TPC
  - Bereitstellung standardisierter Benchmarks
  - Angaben über die Leistungsfähigkeit von Transaktions- und Datenbankmanagementsystemen

TPC-C: Online-Transaction-Processing (OLTP)

- Simulation der Datenverarbeitung eines Großhändlers
- Mehrere Nutzer führen Warentransaktionen aus
  - Bestellungen und Auslieferungen
  - Bezahlungen, Prüfungen des Bearbeitungsstatus
  - Überwachung des Warenbestands

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

Historie: Bestes System 2013 (Oracle 11g auf SPARC T5-8 Server):

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

TPC-C Result Highlights (V2.0 - for Non-TPC Members)

As of 29-May-2017 at 11:12 AM [GMT]

Dell PowerEdge T620

Reference URL: http://www.tpc.org/1794

Benchmark Stats

Result ID:	114112501
Status:	Accepted Result
Report Date:	11/25/14
TPC-C Rev:	5.11.0

System Information

Total System Cost:	21,160 USD
Performance:	112,890 tpmC
Price/Performance:	.19 USD per tpmC
TPC-Energy Metric:	Not reported
Availability Date:	11/25/14
Active Expiration Date:	11/25/17
Database Manager:	SQL Anywhere 16
Operating System:	Microsoft Windows 2012 Standard x64
Transaction Monitor:	Microsoft COM+

Server Specific Information

CPU Type:	Intel Xeon E5-2670 v2 2.5GHz
Total # of Processors:	2
Total # of Cores:	20
Total # of Threads:	40
Cluster:	N

Client Specific Information

# of Clients:	1
CPU Type:	Intel Xeon E5-2670 v2 2.5GHz
Total # of Processors:	2
Total # of Cores:	20
Total # of Threads:	40

Jet-speed HHA2212

Reference URL: http://www.tpc.org/1795

KRW = südkoreanischer WON

Benchmark Stats

Result ID:	117050801
Status:	Accepted Result
Report Date:	05/08/17
TPC-C Rev:	5.11.0

System Information

Total System Cost:	241,936,000 KRW
Performance:	139,909 tpmC
Price/Performance:	1,730.00 KRW per tpmC
TPC-Energy Metric:	Not reported
Availability Date:	05/09/17
Active Expiration Date:	05/08/20
Database Manager:	Goldilocks v3.1 Standard Edition
Operating System:	CentOS 6.6
Transaction Monitor:	Red Hat JBOSS Web Server

10x

Server Specific Information

CPU Type:	Intel Xeon E5-2630 v3 2.40 GHz
Total # of Processors:	2
Total # of Cores:	16
Total # of Threads:	16
Cluster:	N

Client Specific Information

# of Clients:	3
CPU Type:	Intel Xeon E5-2630 v3 2.40 GHz
Total # of Processors:	2
Total # of Cores:	16
Total # of Threads:	32

Netzwerk-Speicher ist kein Flaschenhals

- Durchsatz SSD: >500 MB/s (Serial ATA)
- SDRAM: 50 Gbit/s (Latenz: ~ ns)
- Ethernet
  - 100 Gbit/s (Latenz: ~ µs)
  - 400 Gbit/s (Terabit Ethernet (TbE) um 2020)

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_2)“
    - 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

M. Brantner, D. Florescu, D. Graf, D. Kossmann, T. Kraska, Building a database on S3, Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD '08), S. 251-264, 2008

Google's Spanner 2012

Google Spanner's Most Surprising Revelation: NoSQL Is Out And NewSQL Is In

MONDAY, SEPTEMBER 24, 2012 AT 9:12AM

Google recently released a paper on Spanner, their planet enveloping tool for organizing the world's monetizable information. Reading the Spanner paper I felt it had that chiseled in stone feel that all of Google's best papers have. An instant classic. Jeff Dean foreshadowed Spanner's humungousness as early as 2009. Now Spanner seems fully online, just waiting to handle "millions of machines across hundreds of datacenters and trillions of database rows." Wow.

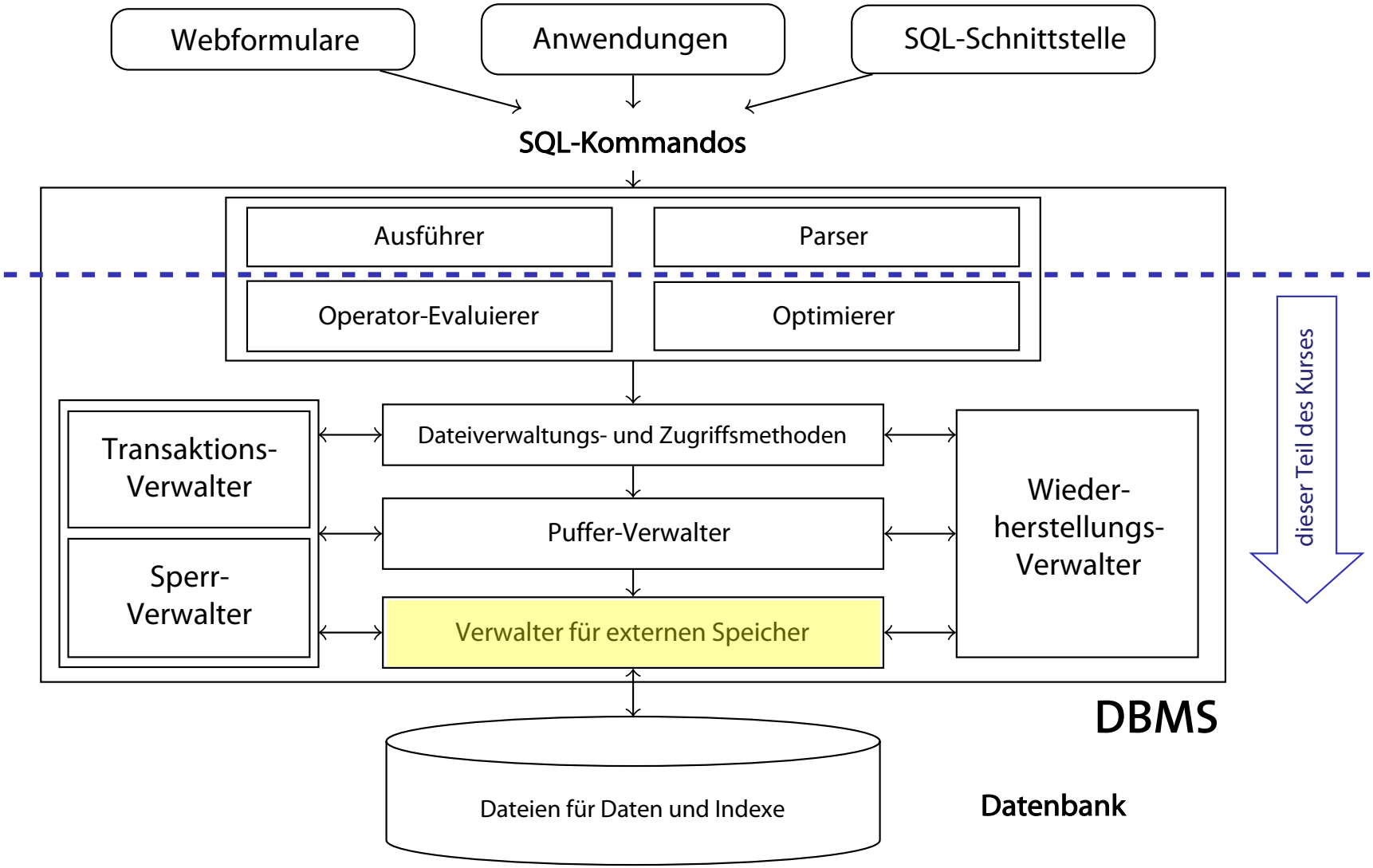
The Wise have yet to weigh in on Spanner en masse. I look forward to more insightful commentary. There's a lot to make sense of. What struck me most in the paper was a deeply buried section essentially describing Google's motivation for shifting away from NoSQL and to NewSQL. The money quote:

> ***We believe it is better to have application programmers deal with performance problems due to overuse of transactions as bottlenecks arise, rather than always coding around the lack of transactions.***

This reads as ironic given Bigtable helped kickstart the NoSQL/eventual consistency/key-value revolution.

http://highscalability.com/blog/2012/9/24/google-spanners-most-surprising-revelation-nosql-is-out-and.html

Architektur eines DBMS



Verwaltung des externen Speichers

- Abstraktion von technischen Details der Speichermedien
- Konzept der Seite (page) mit typischerweise 4-64KB als Speichereinheit 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

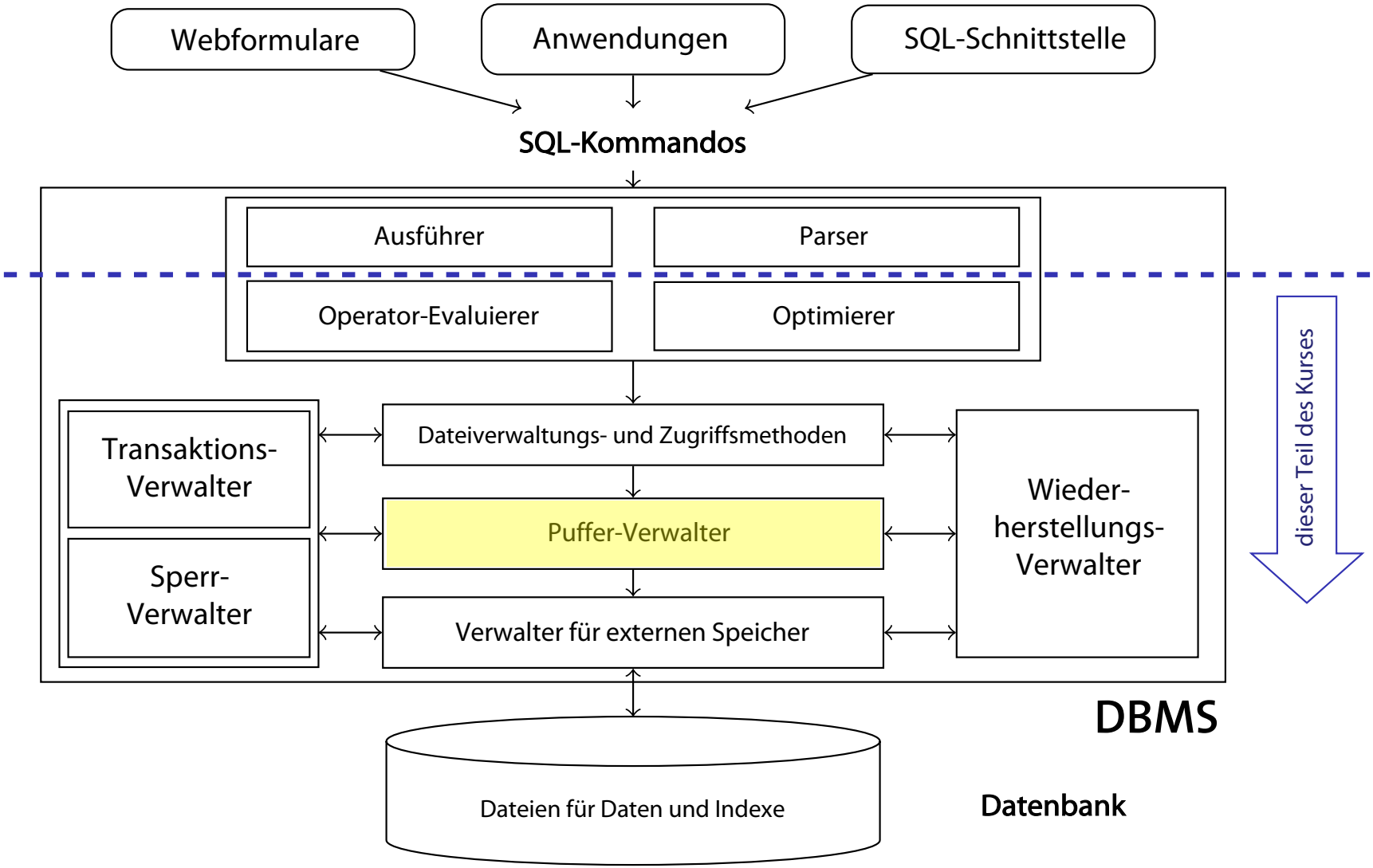
Verwaltung leerer Seiten

Verwendete Techniken:

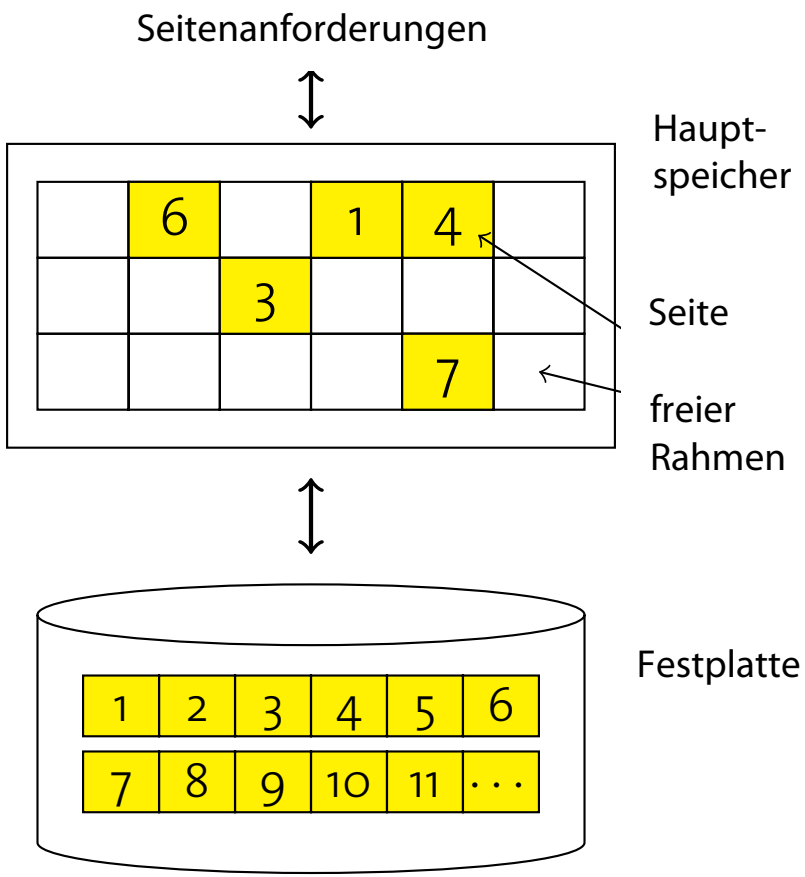
1. Liste der freien Seiten
   - Hinzufügung falls Seite nicht mehr verwendet
2. Bitmap mit einem Bit für jede Seite
   - Umklappen des Bits k, wenn Seite k (de-)alloziert wird
   - Finden von hintereinanderliegenden Seiten einfacher

Persistent als Verwaltungsinformationen zu speichern

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
- Genutzt von Transaktionen (mit richtiger Verschachtelung von pin und unpin)

Frage: Wofür ist der dirty-Flag nötig?

Frage: Wofür ist der dirty-Flag nötig?

Antwort: Nur modifizierte Seiten müssen auf die Festplatte zurückgeschrieben werden.

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;
```

Frage: Warum werden Seiten nicht direkt beim unpin zurückgeschrieben?

Frage: Warum werden Seiten nicht direkt beim unpin zurückgeschrieben?

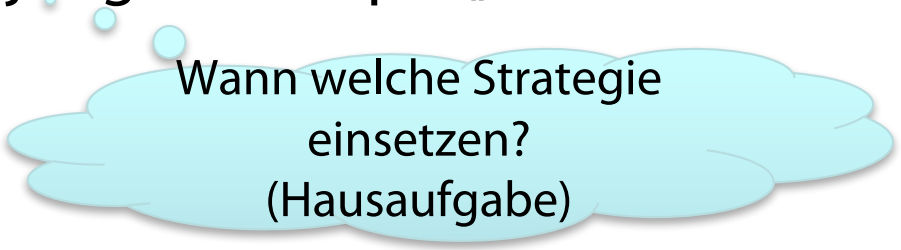
Antwort: Werden wir später besser verstehen (Transkationsmanagment).

- Von Vorteil wäre: Unmittelbares Zurückschreiben würde Fehlererholung vereinfachen (denn nicht unmittelbares Zurückschreiben kann zu Verlusten führen, die durch zusätzliche Protokolle abzufangen sind).
- Nachteil ist:
  - Hohe I/O-Kosten (jedes Unpin führt direkt zum Schreiben auf die Festplatte)
  - Mäßige Antwortzeiten für Transkationen.

Verdrängungsstrategien/-heuristiken

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

- Least Recently Used (LRU)
  - Verdrängung der Seite mit am längsten zurückliegendem 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



Pufferverwaltung in der Praxis

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

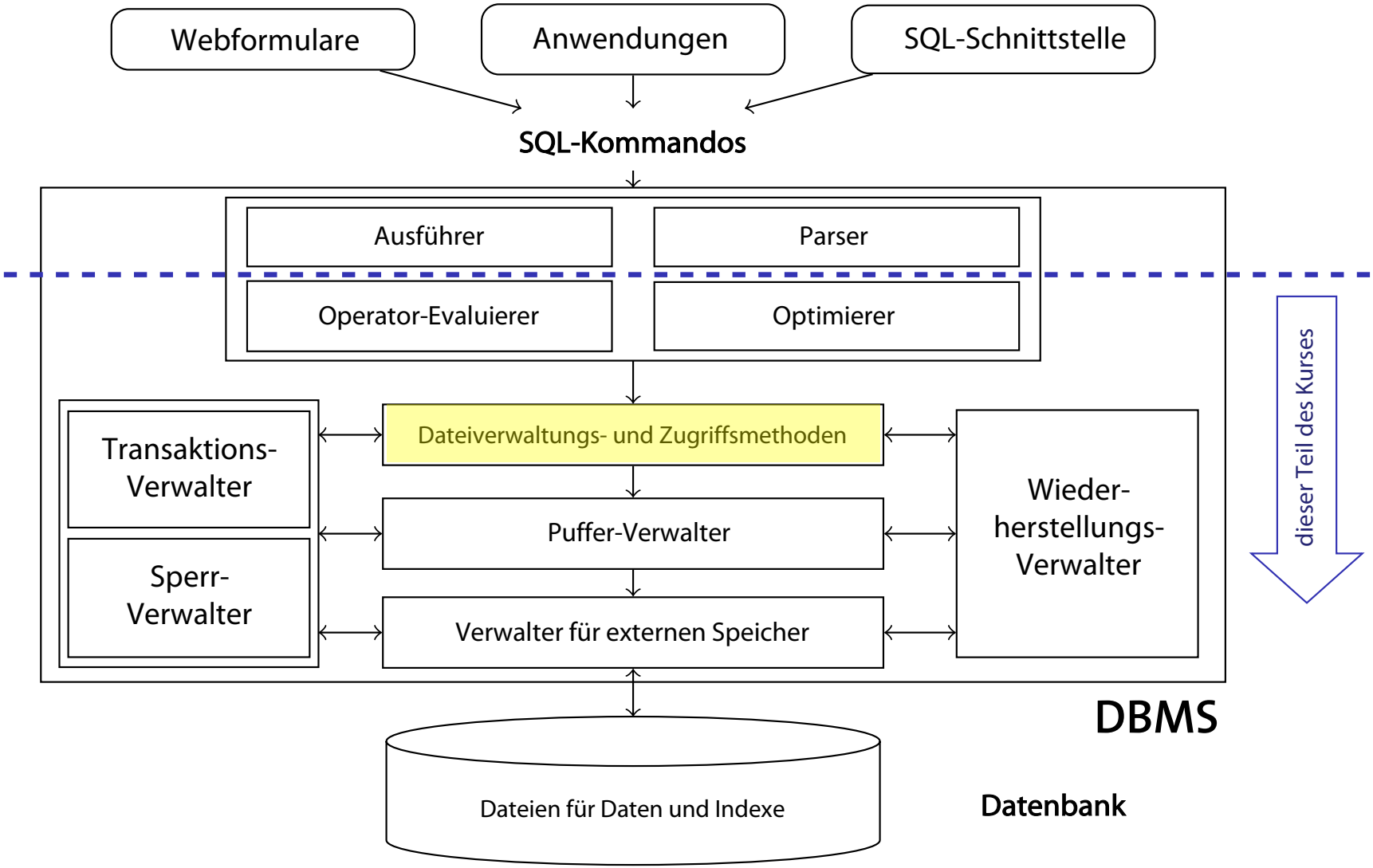
Datenbanken vs. Betriebssysteme

- Haben wir nicht gerade ein Betriebssystem entworfen?
- Ja
  - Verwaltung für externen Speicher und Pufferverwaltung ähnlich
  - Memory-mapped files
- 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

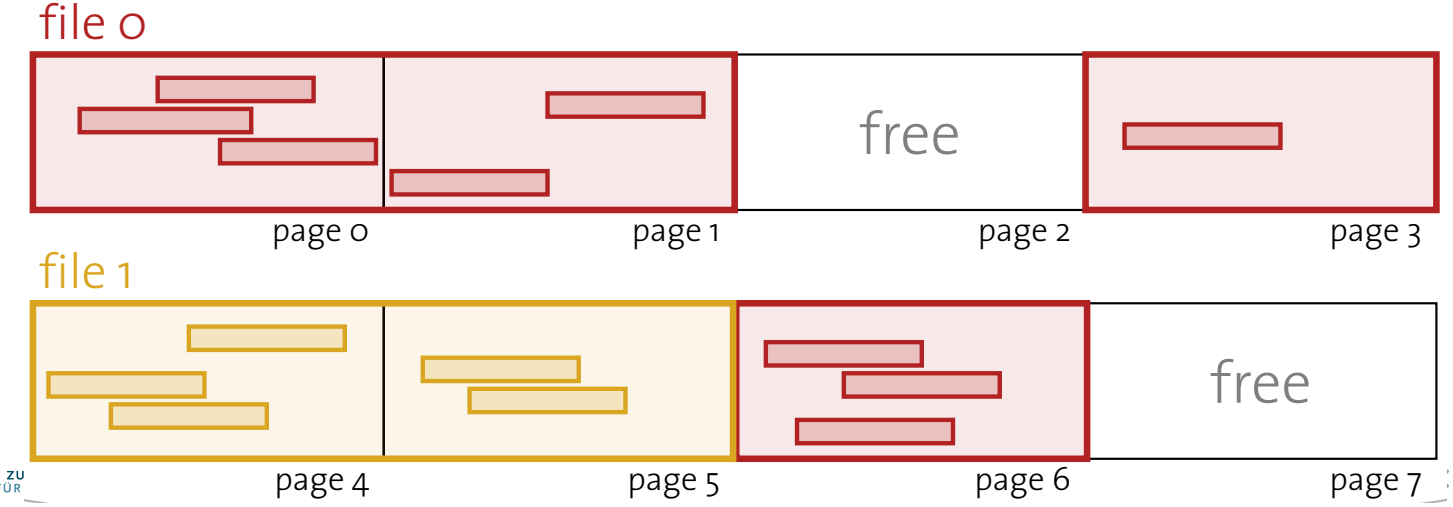
- Gegenseitige Störung möglich
  - Doppelte Seitenverwaltung
  - DMBS-Transaktionen vs. Transaktionen auf Dateien organisiert vom Betriebssystem (journaling)
  - DBMS Pufferbereiche durch Betriebssystem ausgelagert auf Festplatte
- (Daher:) DBMSe schalten Betriebssystemdienste aus
  - Direkter Zugriff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

Architektur eines DBMS



Datenbank-Dateien

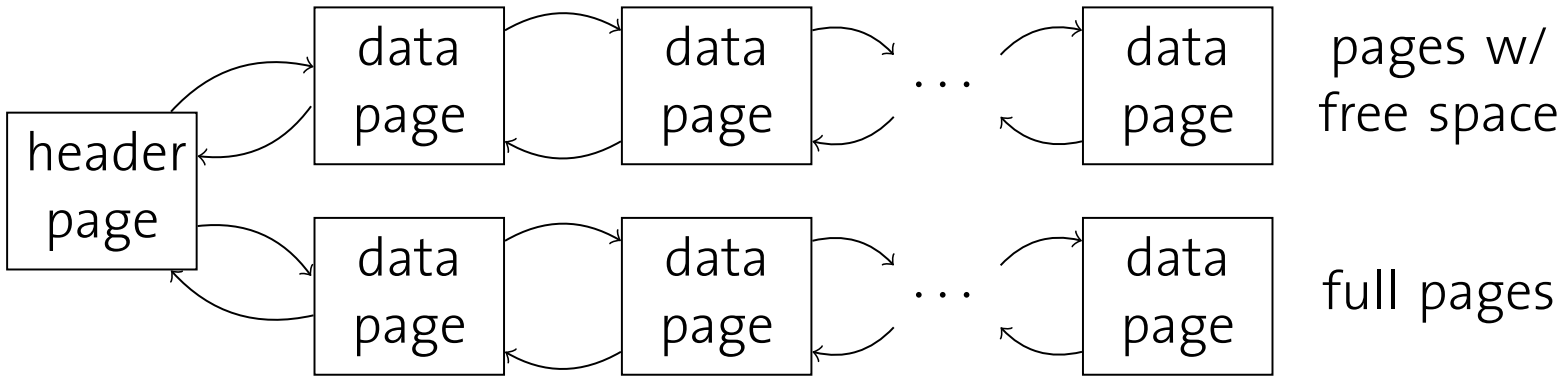
- Seitenverwaltung ist unbeeinflusst vom Inhalt
- DBMS verwaltet aber auch Inhalte: Tabellen von Tupeln, Indexstrukturen, ...
- Tabellen sind Dateien von Datensätzen (records)
  - Datei besteht aus einer oder mehrerer Seiten
  - Jede Seite speichert eine oder mehrere Datensätze
  - Jeder Datensatz korrespondiert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



Heap-Dateien

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

Verkettete Liste von Seiten



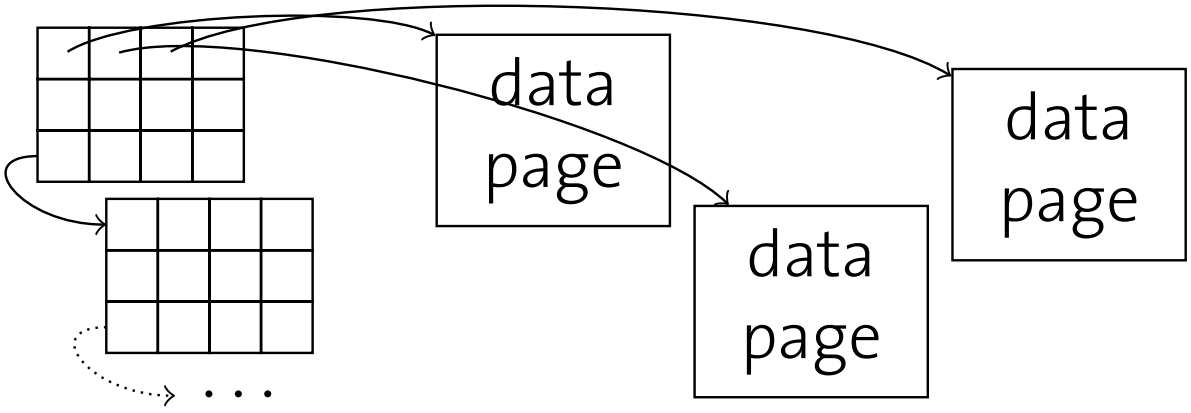
✓ Einfach zu implementieren

✗ 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 auf den Seiten. (Abwägungssache: Wenig Speicherverbrauch vs. Genauigkeit der Angabe)

✓ Suche nach freien Plätzen effizient

✗ Zusatzaufwand für Verzeichnisspeicher

Freispeicher-Verzeichnis

Welche Seite soll für neuen Datensatz gewählt werden?

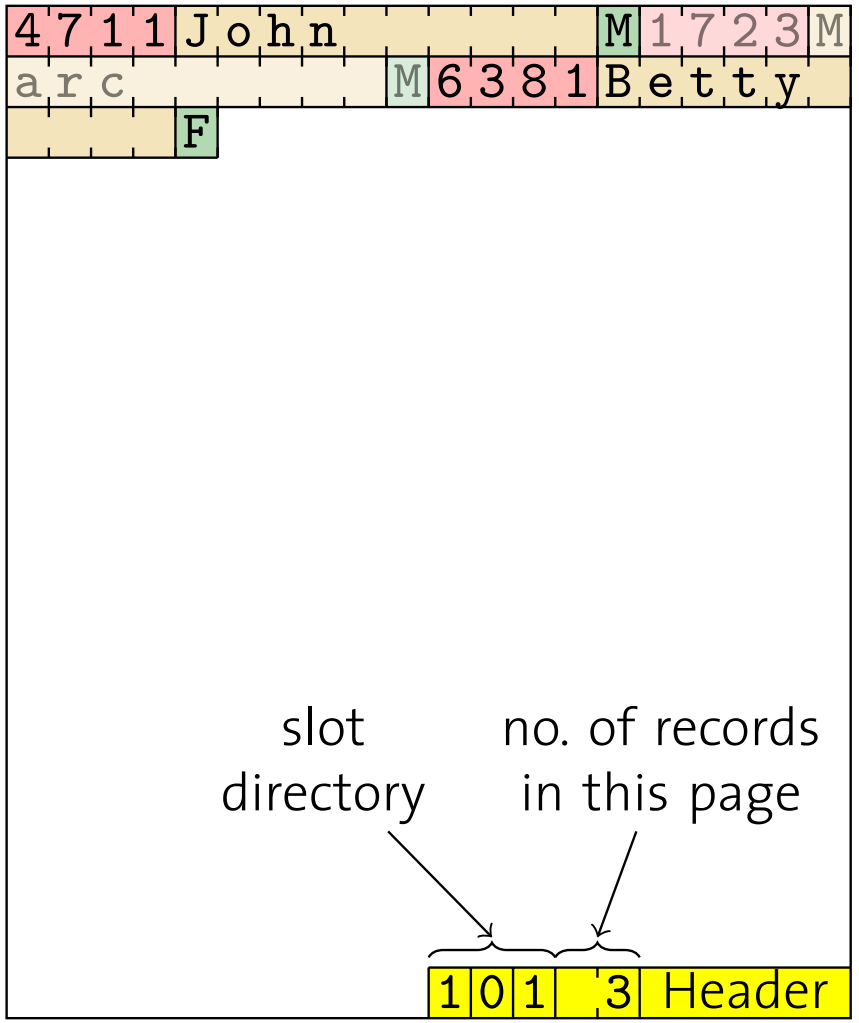
- Append Only
  - Immer in letzte Seite einfügen, sonst neue Seite anfordern
- Best-Fit
  - Alle Seiten müssen betrachtet werden, Reduzierung der Fragmentierung
- First-Fit
  - Suche vom Anfang, nimm erste Seite mit genug Platz
  - Erste Seiten füllen sich schnell, werden immer wieder betrachtet
- Next-Fit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

Inhalt einer Seite (für Datensätze fester Länge)

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

- Datensatz-Kennung (record identifier, rid):

  (seitennr, slotnr)
- Datensatz-Position (Versatz auf der Seite)
  Slotnnr x Bytes pro Slot
- Datensatz gelöscht?
  - rid sollte sich nicht ändern
  - Slot-Verzeichnis (Bitmap)

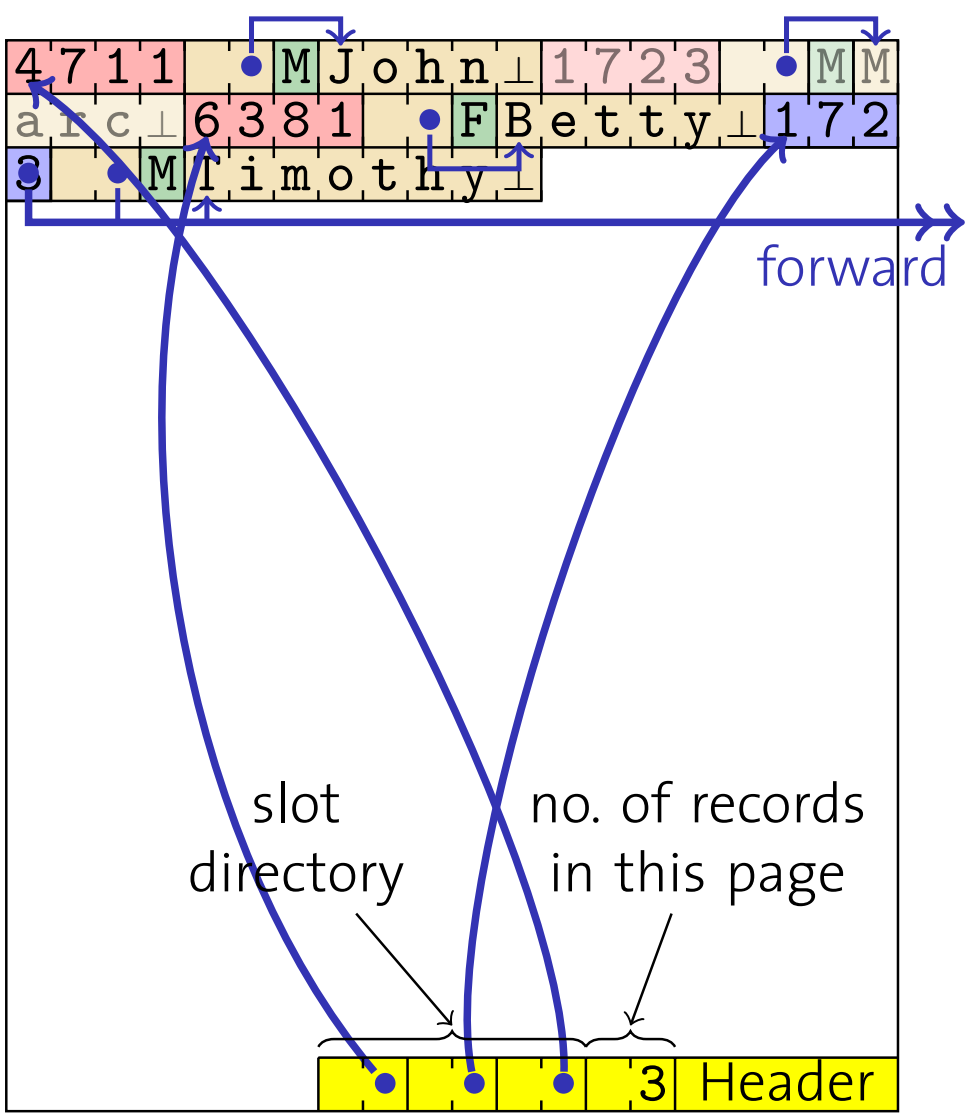


Inhalte einer Seite: Felder variabler Länge

- Felder variabler Länge zum Ende verschoben
  - Platzhalter zeigt auf Position

Warum?

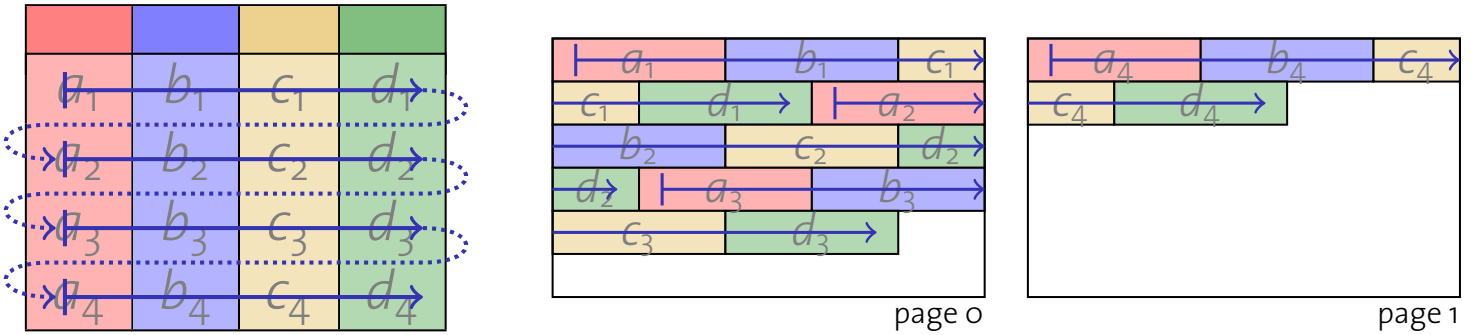
- In Slot-Verzeichnis Zeiger auf Start eines Datensatzes
- Datensätze können auf Seite verschoben werden (z.B. wenn sich Feldgröße ändert)
- Einführung einer Vorwärtsreferenz, wenn Datensatz nicht auf Seite passt



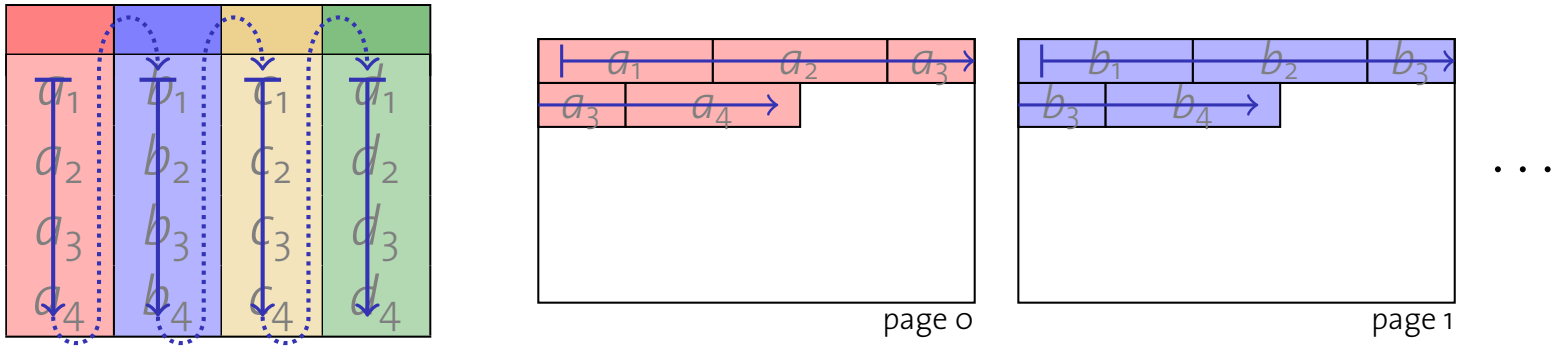
Was passiert bei Updates?

Alternative Seiteneinteilungen

- Im Beispiel wurden Datensätzen zeilenweise angeordnet:



- Spaltenweise Anordnung genauso möglich:

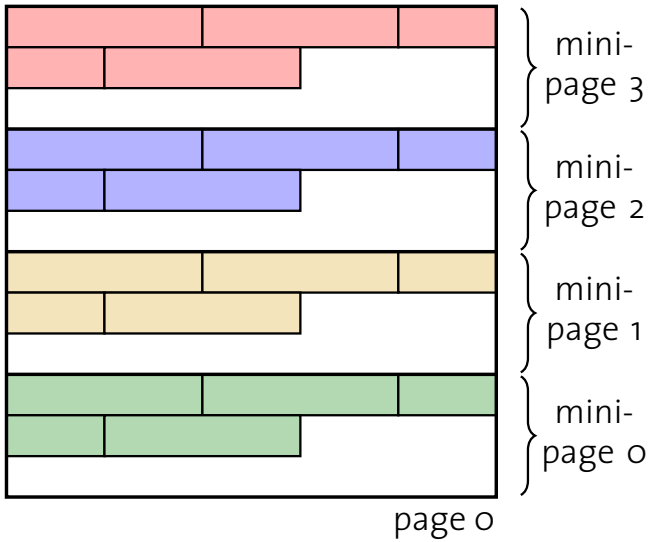


Alternative Seitenanordnungen

Vorgestellte Schemata heißen auch:

- Row-Store
- Column-Store

- Anwendungen für verschiedene Lasttypen und Anwendungs-kontexte (z.B. OLAP)
- Unterschiedliche Kompressionsmöglichkeiten
- Kombination möglich
  (PAX: partition attribute across)
  - Tabelle horizontal partitionieren in Seiten
  - Teiltabellen kolumnenweise in Mini-Seiten



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