Informationssysteme

Sommersemester 2016

Prof. Dr.-Ing. Sebastian Michel
TU Kaiserslautern

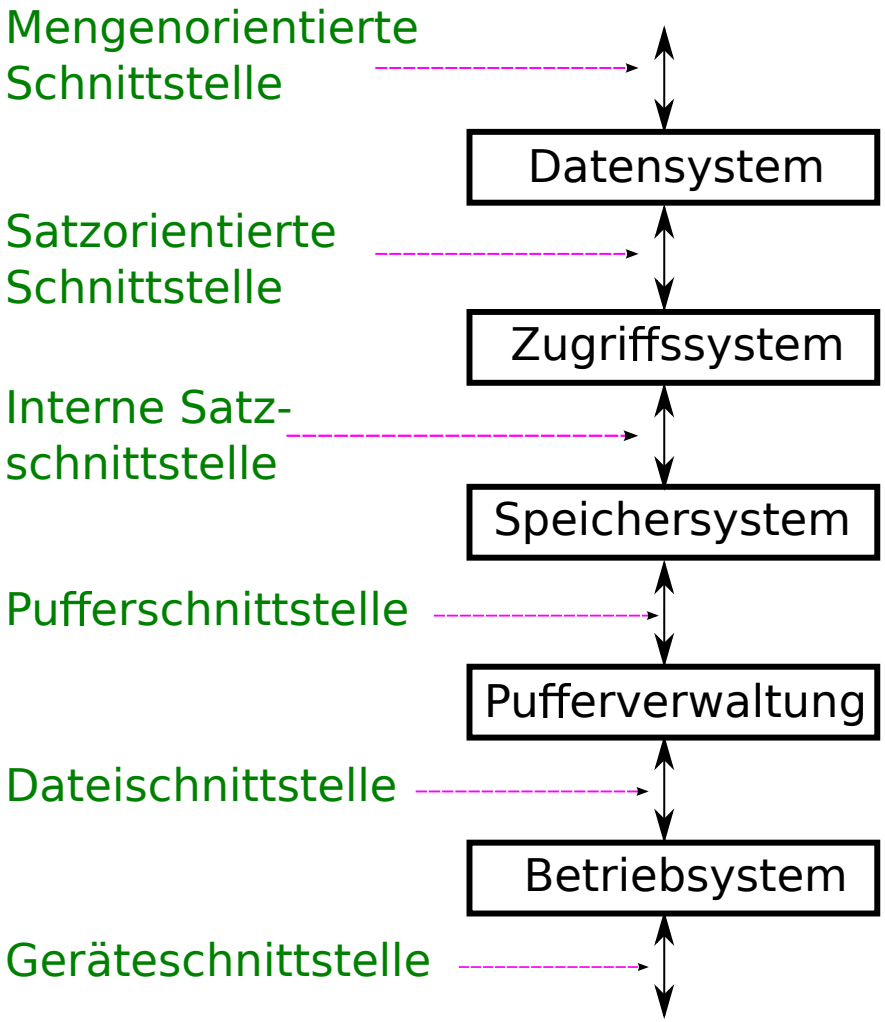
smichel@cs.uni-kl.de

Übersicht und Ausblick

Ausblick auf kommende Vorlesungen

- Zugriffskosten
- Physische Datenorganisation
- Pufferverwaltung
- Indexstrukturen
- Regelbasierte Anfrageoptimierung

5 Schichtenmodell einer Datenbank

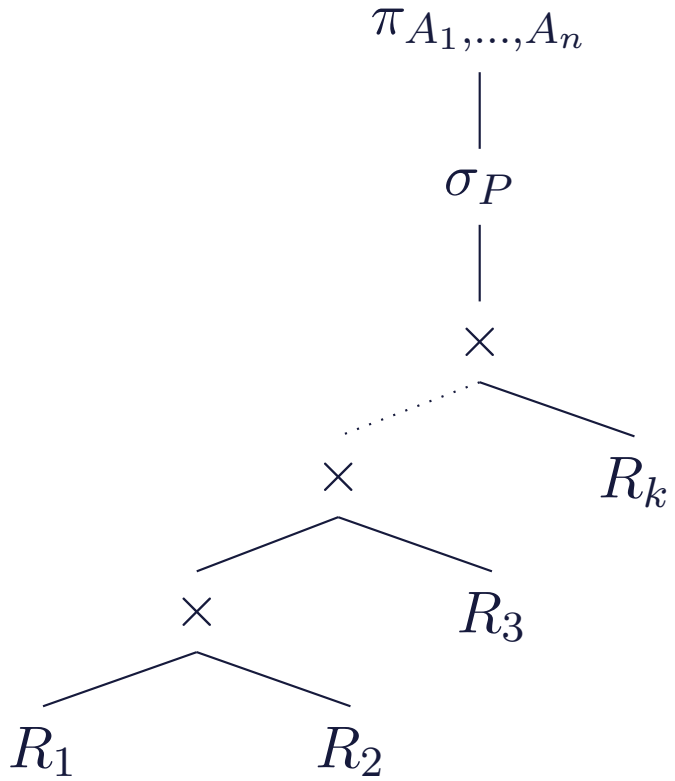


Übersicht Anfrageverarbeitung

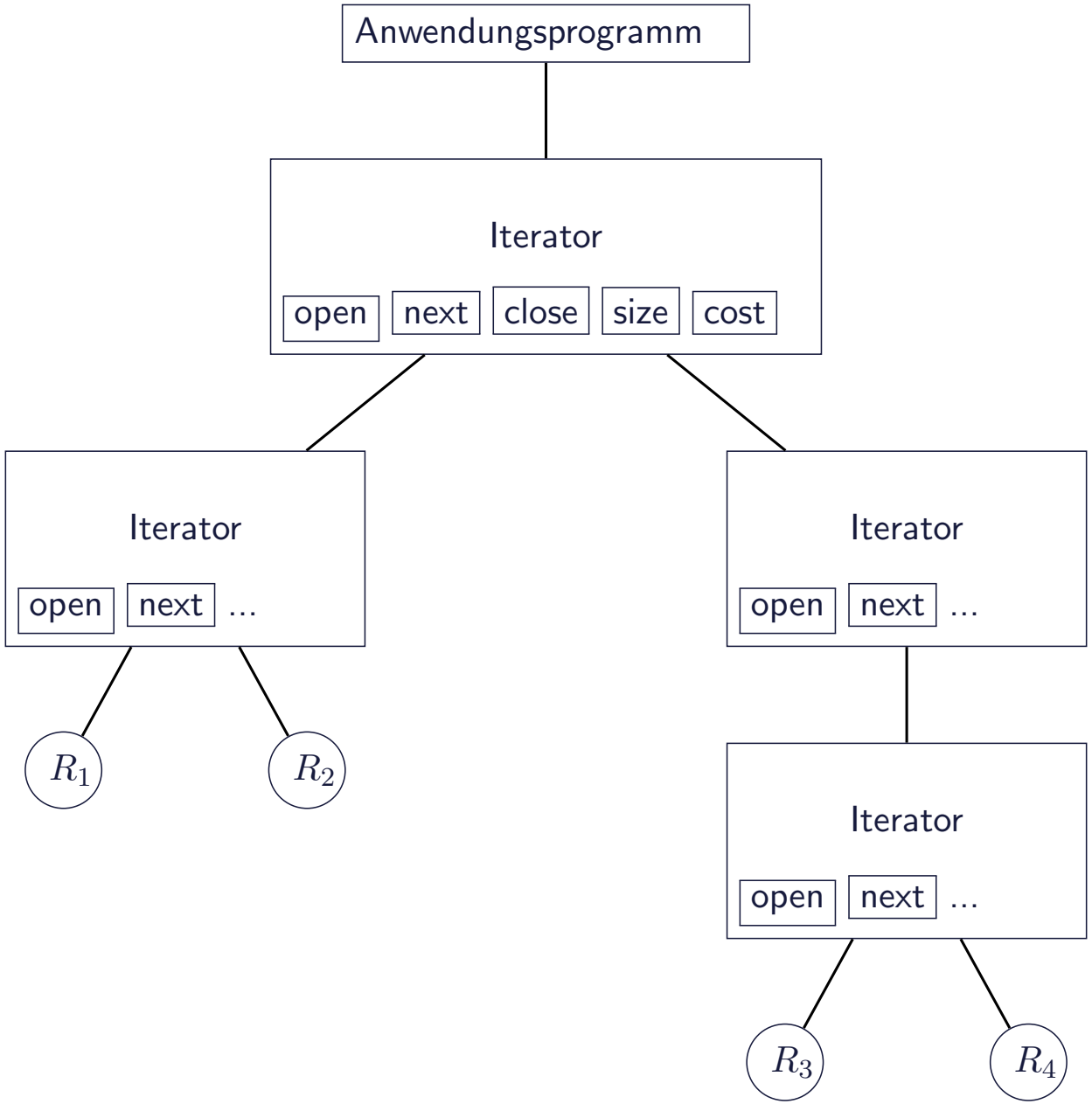
- **Eingabe:** Anfrage als Text (String)
- Kompilierzeitsystem (compile time system) übersetzt und optimiert die Anfrage
- Zwischenprodukt: **Anfrage als Anfrageplan** (query plan)
- Laufzeitsystem (runtime system) führt die Anfrage aus
- **Ausgabe:** Ergebnisse der Anfrage

Übersetzung von SQL in Baum aus Operatoren

select A_1, ..., A_n
from R_1, ..., R_k
where P;



Was sind Operatoren, wie läuft Verarbeitung ab und wie werden Operatoren Implementiert?


Anwendungsprogramm
Iterator
open
next
close
size
cost
Iterator
open
next
...
Iterator
open
next
...
R_1
R_2
Iterator
open
next
...
R_3
R_4

Iterator

- **Open:** Konstruktor, initialisiert, öffnet die Eingabe
- **Next:** Liefert das nächste Ergebnis
- **Close:** Schließt die Eingabe
- **Cost und Size:** Geben Informationen über die geschätzten (!) Kosten

Empfehlenswert: Übersichtsartikel von G. Graefe: Query Evaluation Techniques for Large Databases, ACM Computing Surveys, 1993, volume 25, number 2, pages 73-170.

Pull-basierte Verarbeitung

- **Operatoren werden in Form von Iteratoren implementiert**
- **Datenfluss hierbei ist "von unten nach oben"**
- **Konsument-Produzent Verhältnis**
- Der konsumierende Iterator bezieht Tupel von seinen Eingaben, die ebenfalls Iteratoren sind, durch deren `next()` Schnittstelle

- **Im Allgemeinen werden Zwischenergebnisse nicht explizit materialisiert.**

Anmerkung: Push-basierte Verarbeitung

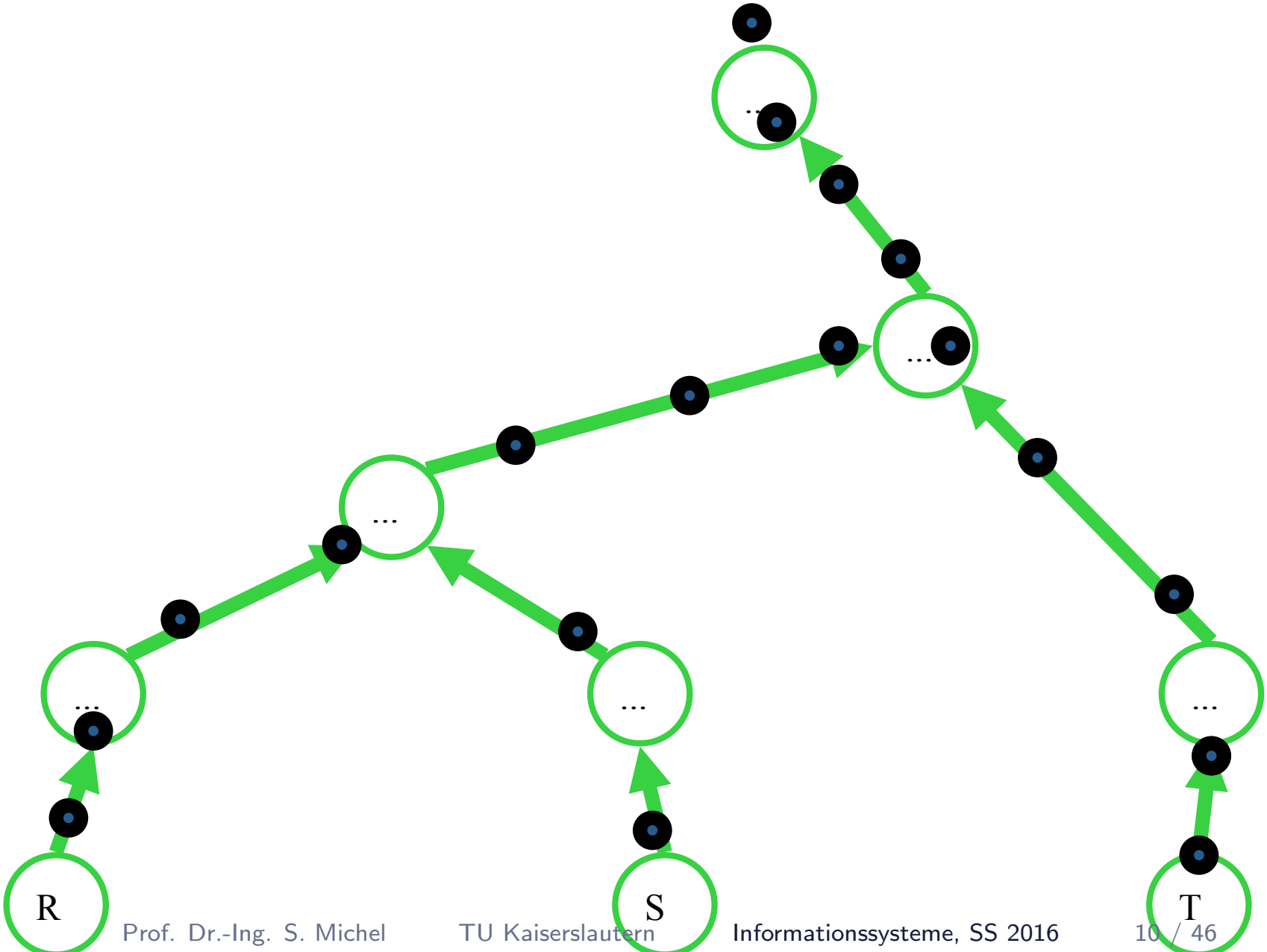
Es gibt insbesondere bei Systemen zur Datenstromverarbeitung die sogenannte Push-basierte Verarbeitung. Dort registrieren sich "Konsumenten" an Datenquellen oder anderen Operatoren. Falls diese neue Daten haben, werden die Daten an die Konsumenten weiter gereicht (aktiv).

Blockierende Operatoren

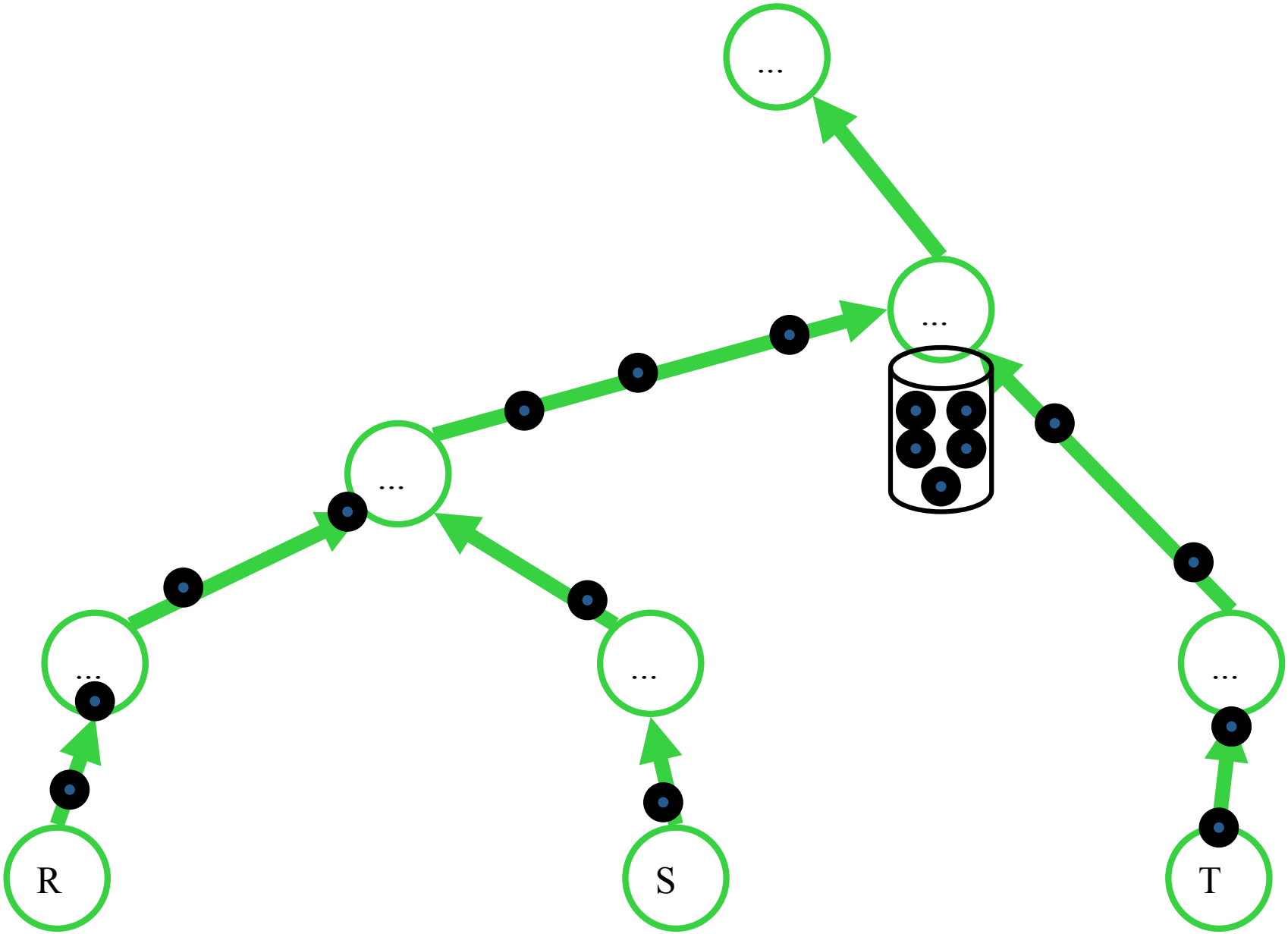
Idealerweise

- **Operatoren blockieren den Datenfluss nicht**
- D.h. beim Aufruf von `next()` werden darunter liegende Operatoren angefragt via next und das Ergebnis direkt weiter geleitet. Nur wenige Tupel werden dabei gelesen.
- **Erlaubt Pipelining**
- Im Gegensatz dazu: Operatoren, die den Datenfluss "blockieren", sogenannte Pipeline-Breaker

Pipelining vs. Pipeline-Breaker



Pipelining vs. Pipeline-Breaker



Pipeline-Breaker

- Sortieren
- Duplikate eliminieren (unique, distinct)
- Aggregation: min, max, avg, ...
- Joins (je nach Implementierung)
- Union (je nach Implementierung)

Implementierung von Operatoren

- **Bei der Erzeugung eines physischen Anfrageplans muss entschieden werden wie genau die Anfrage ausgeführt werden soll**

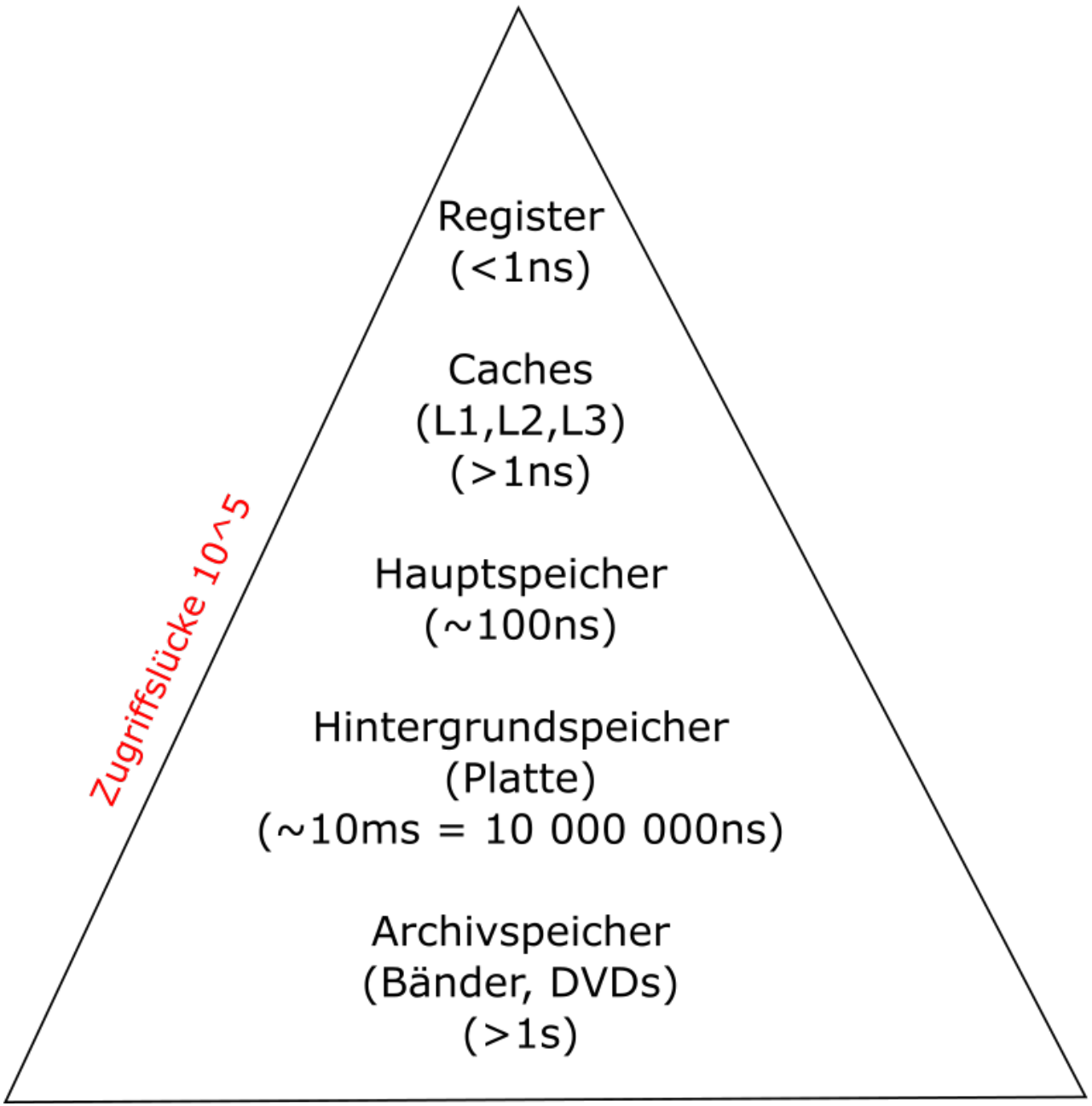
- Welche Implementierungen gibt es für die diversen Operatoren?
- Welche Implementierung ist effizienter?
- Und wie kann dies überhaupt berechnet/vorhergesagt werden? (Kostenmodelle)
- Gibt es Indexstrukturen, die ausgenutzt werden können?
- Falls ja, macht es auch Sinn einen Index zu benutzen?
- ...

Implementierung von Operatoren wird in der **VL Datenbanksysteme** besprochen.

Was kostet wieviel?

- L1 cache reference 0,5 ns
- L2 cache reference 7 ns
- Main memory reference 100 ns
- Compress 1K bytes with Zippy 10 000 ns
- Send 2K bytes over 1 Gbps network 20 000 ns
- Read 1 MB sequentially from memory 250 000 ns
- Round trip within same datacenter 500 000 ns
- Disk seek 10 000 000 ns
- Read 1 MB sequentially from network 10,000,000 ns
- Read 1 MB sequentially from disk 30 000 000 ns
- Send packet CA $\rightarrow$ Netherlands $\rightarrow$ CA 150 000 000 ns

Numbers by Jeff Dean (Google)



Analogie

Kopf
(1 Min)

Raum
(10 Min)

Stadt
(1 Std)

Pluto
(2 Jahre)

Andromeda
(2000 Jahre)

Problemstellung

Langfristige Speicherung und Organisation der Daten im Hauptspeicher?

- Speicher ist (noch) flüchtig!
- Speicher ist (noch) zu teuer und zu klein.
- Kostenverhältnis Festplatte vs. DRAM

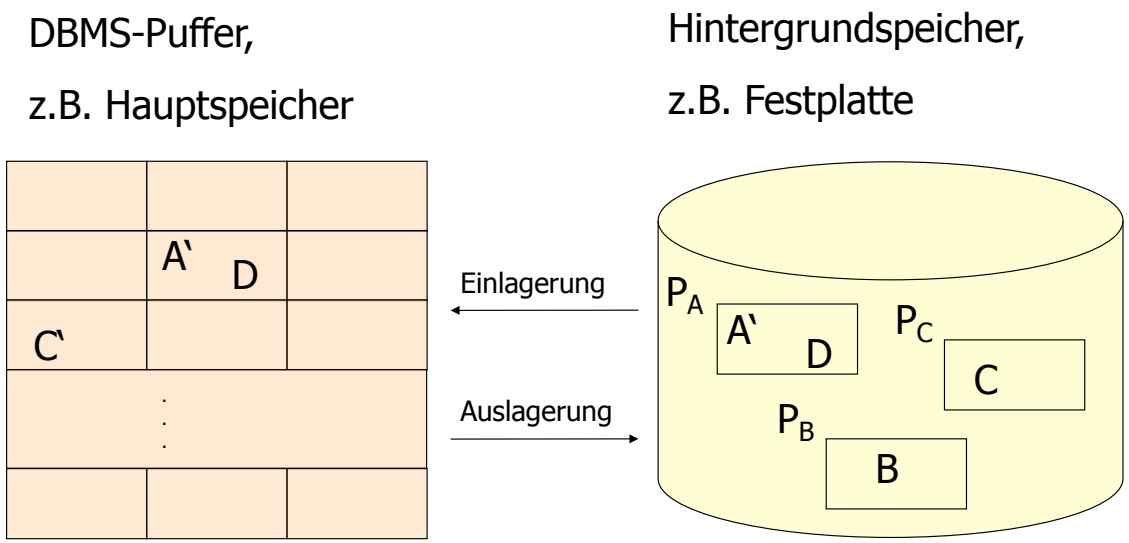
Mindestens zweistufige Organisation der Daten erforderlich

- Langfristige Speicherung auf großen, billigen und nicht-flüchtigen Speichern (Externspeicher)
- Verarbeitung erfordert Transport zum/vom Hauptspeicher

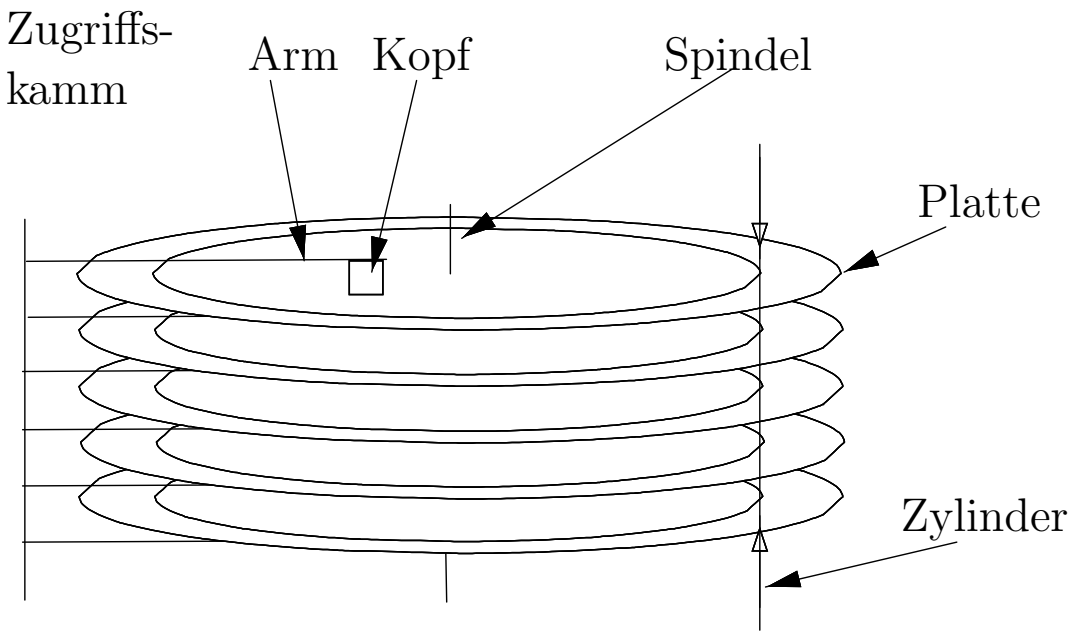
Zweistufige Speicherhierarchie

Vereinfachte E/A-Architektur:

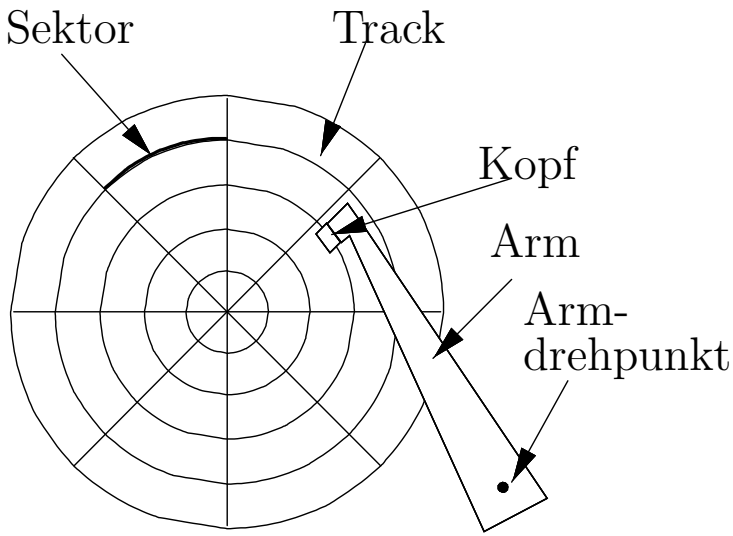
- Modell für Externspeicheranbindung
- Vor Bearbeitung sind die Seiten in den Datenbank-Puffer zu kopieren
- Geänderte Seiten müssen zurückgeschrieben werden



Aufbau einer (klassischen) Festplatte



a. Seitenansicht



b. Draufsicht

Aufbau Festplatte (Tracks, Sektoren, Zonen)

- **Track:** Teil eines Zylinders auf einer Platte
- **Sektor:** Teil eines Tracks. Anzahl Sektoren pro Track ursprünglich gleich für alle Tracks
- Aber, auf Zylinder/Tracks weiter "außen" passen mehr Sektoren. Daher, aktuelle Hardware, variable Anzahl von Tracks: äußere Tracks haben mehr Sektoren als innere Tracks; Zylinder sind in Zonen unterteilt.

Blöcke

- Aka. Sektoren, Aka. physical Record. Kleinste Transfereinheit bei Block-Storage-Devices. Seit wenigen Jahren typischerweise 4KB oder sonst (historisch) 512 Byte groß.
- **Achtung**, es gibt auch Unterschiede zu Blöcken bzw. Seiten des Dateisystems, dort kann ein Block auch mehrere Festplattenblöcke umfassen.

Beispiel

Die Festplatte SAMSUNG HD103SJ, die in meinem Desktop PC im Büro eingebaut ist hat laut (Linux tool) `hdparm -i` (oder `hdparm -I` für mehr Details):

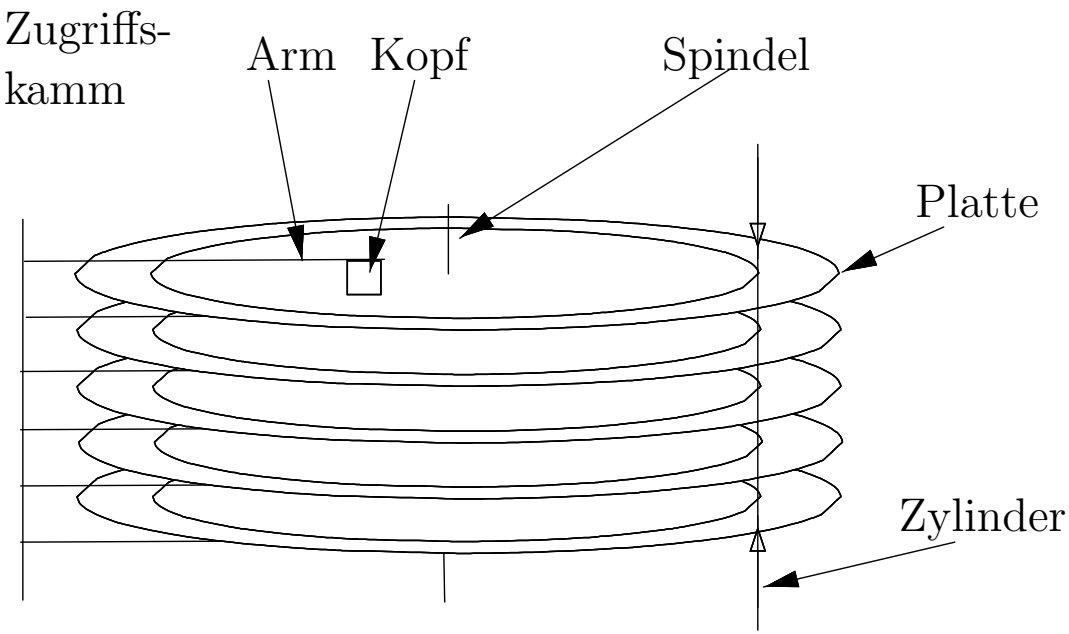
- 1953525168 Sektoren
- a 512 Bytes.

Das macht: 1 TB

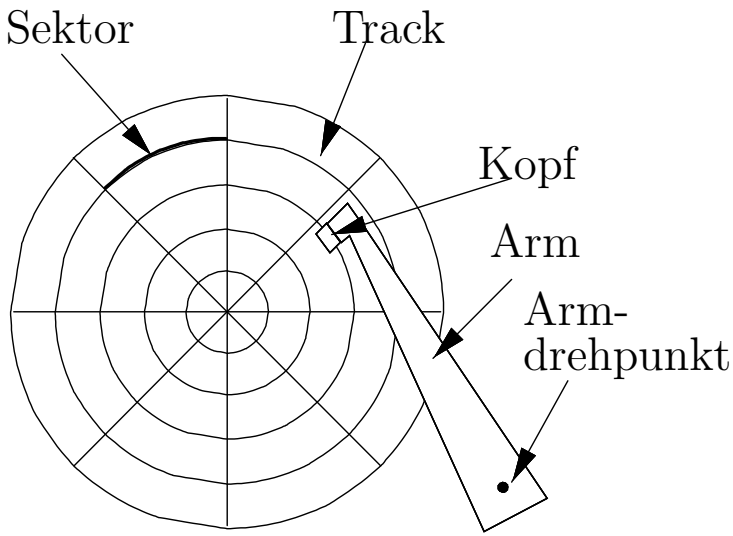
Ausgabe (Auszug) von hdparm -I /dev/sda

```
CHS current addressable sectors:   16514064
LBA    user addressable sectors:  268435455
LBA48  user addressable sectors: 1953525168
Logical  Sector size:                   512 bytes
Physical Sector size:                   512 bytes
device size with M = 1024*1024:      953869 MBytes
device size with M = 1000*1000:     1000204 MBytes (1000 GB)
```

Aufbau einer (klassischen) Festplatte

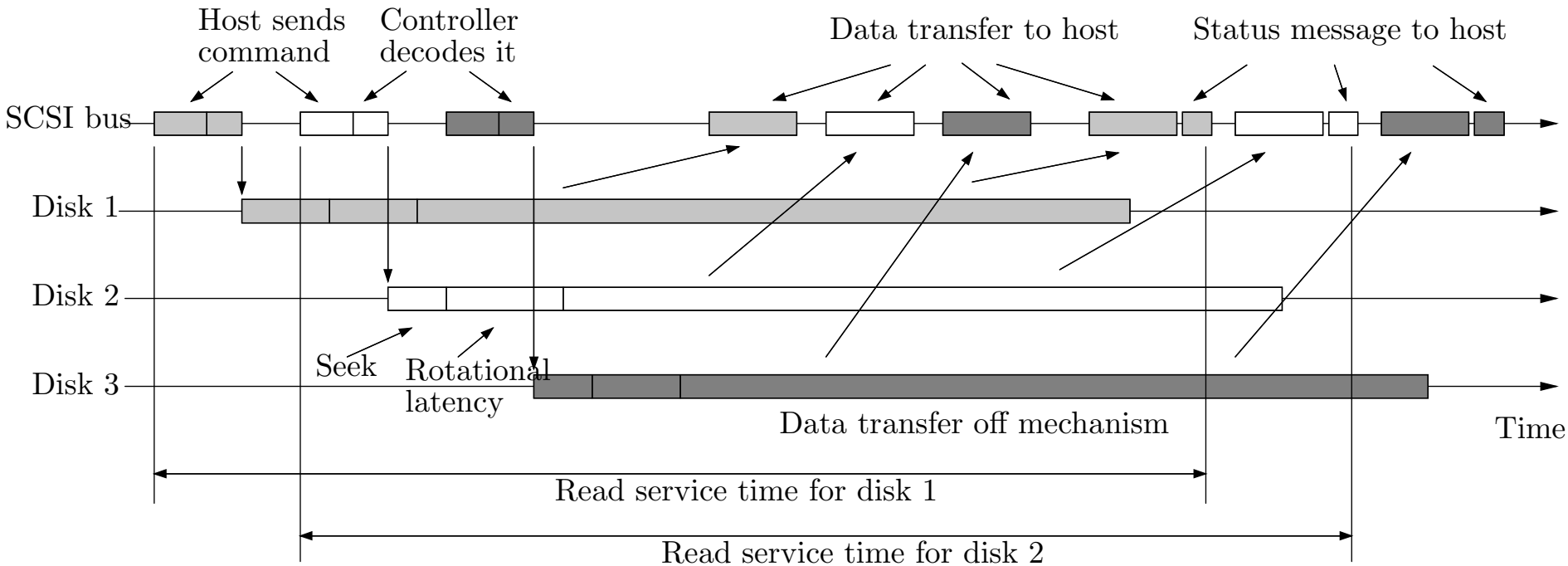


a. Seitenansicht



b. Draufsicht

Reading/Writing a Block



Schwierig die Kosten genau zu berechnen

Kosten eines Festplattenzugriffs hängen ab von:

- der aktuellen **Position des Lesekopfs**
- der **Position (Drehung/Winkel) der Platte**

Diese Informationen sind zur Zeit der Übersetzung der Anfrage **nicht bekannt**.

Daher: Kosten über mehrere Zugriffe (Mittel), einfaches Modell.

Einfaches Kostenmodell

Parameter des Kostenmodells:

- Durchschnittliche **Latenzzeit** (average latency time):
  Durchschnittliche Zeit für Positionierung (seek+rotational delay)
  - Durchschnittliche Zugriffszeit für einen einzelnen Zugriff
- **Lese- / Schreibrate** (sustained read/write rate):
  - Nach Positionierung: Datentransferrate bei sequentiellem Zugriff

Beispiel: Performance Parameter für Beispiel-Festplatte

Modell aus 2004		
Parameter	**Wert**	**Abkürzung**
Kapazität (capacity)	180 GB	D_{cap}
Latenz (average latency time)	5 ms	D_{lat}
Leserate (sustained read rate)	100 MB/s	D_{srr}
Schreibrate (sustained write rate)	100 MB/s	D_{swr}

Dann: **Zeit um n Bytes zu lesen** ist geschätzt als $D_{\text{lat}} + n/D_{\text{srr}}$.

Sequenzielle und Wahlfreie Zugriffe (Sequential vs. Random I/O)

Es wird unterschieden zwischen zwei verschiedenen Arten von Zugriffen (I/O):

- **Sequenzielle (sequential)** I/O und
- **Wahlfreie (random)** I/O.

In unserem einfachen Kostenmodell:

- für sequenzielle Zugriffe: es gibt eine Positionierung des Lesekopfes, danach wird mit Leserate gelesen.
- für wahlfreie Zugriffe: es gibt eine Positionierung pro gelesener Einheit – typischerweise einer Seite von z.B. 8 KB

Beispielanwendung des einfachen Kostenmodells

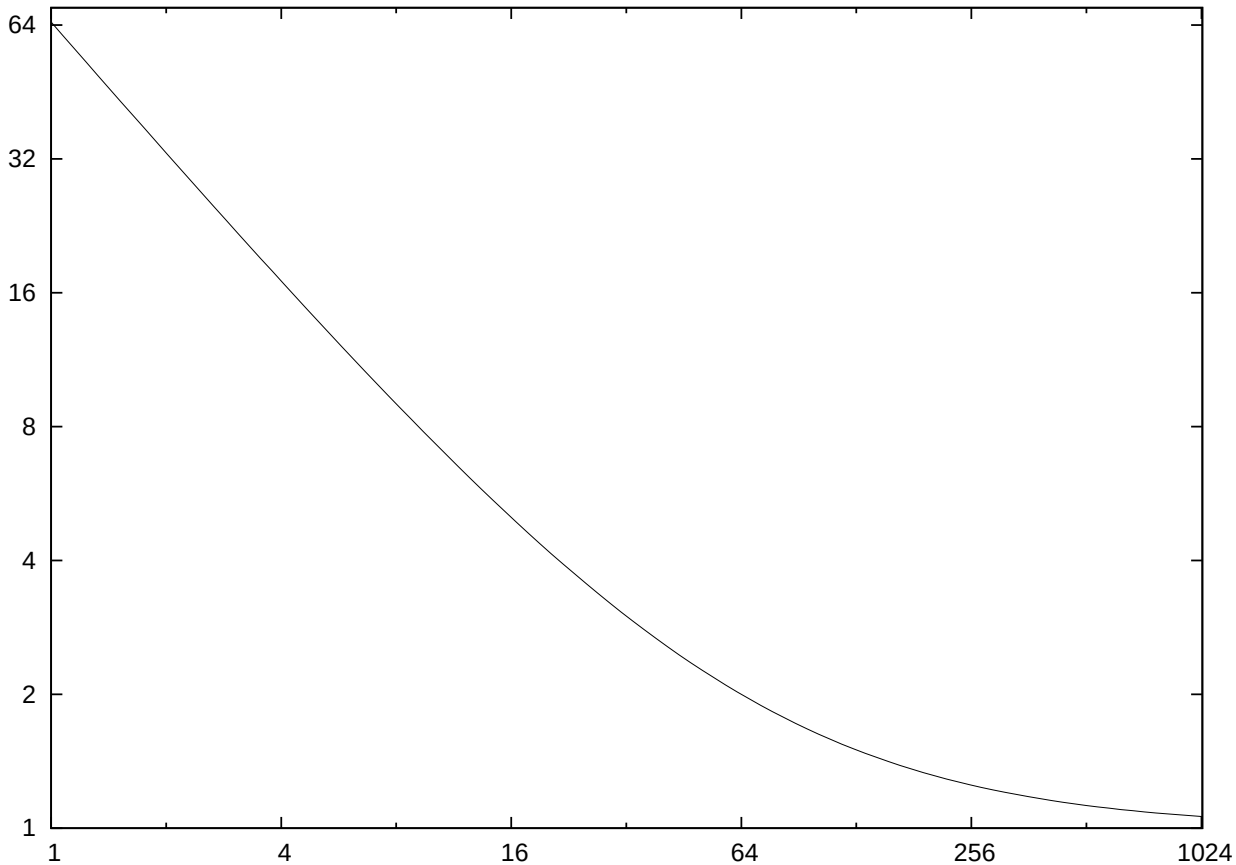
Lese 100 MB

- Sequenzielles Lesen: 5 ms + 1 s
- Lesen durch wahlfreie Zugriffe (Seitengröße 8KB): 65 s

Time to Read 100 MB

x-Achse: Größe der zu lesenden Chunks in 8 KB

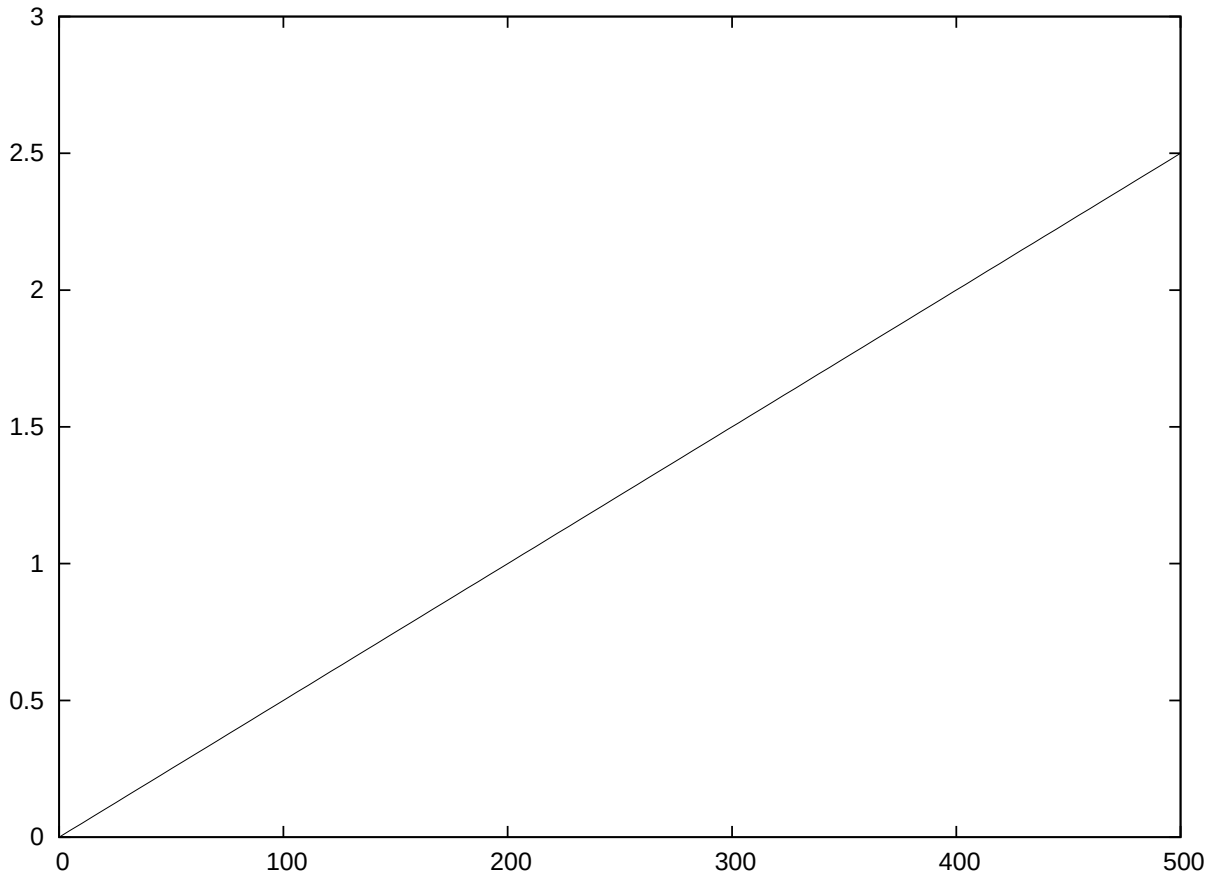
y-Achse: Sekunden



Time to Read n Random Pages

x-Achse: n

y-Achse: Sekunden



Beispiel: Berechnung Zugriffskosten

- Lesen einer Datei von 100 MB Größe, gespeichert in 12800 8 KB Seiten.
- In unserem einfachen Modell kostet das wahlfreie (random access) Lesen von 200 Seiten ungefähr genauso lange wie das Lesen der gesamten 100 MB im sequenziellen Modus.

Das heißt, das Lesen von $1/64$ einer 100 MB Datei im wahlfreien Zugriff dauert genauso lang wie das Lesen der gesamten Datei im sequenziellen Modus.

Break-Even-Point im einfachen Kostenmodell

Sei a die Positionierungs-Zeit, s die Leserate, p die Seitengröße und d die Anzahl an fortlaufend abgelegten Bytes. Dann ist der **Break-Even-Point** gegeben durch

$$\begin{aligned} n * (a + p/s) &= a + d/s \\ n &= (a + d/s)/(a + p/s) \\ &= (as + d)/(as + p) \end{aligned}$$

a und s sind Parameter, die durch die Festplatte gegeben sind (also unveränderlich). Für gegebenes d hängt der Break-Even-Point nur von der Seitengröße ab.

Lessons Learned

- **Sequenzielles Lesen ist sehr viel schneller als wahlfreies Lesen.**
- **Das Datenbanksystem sollte dies idealerweise ausnutzen.**

Möglichkeiten

- Sorgfältig ausgewähltes physisches Layout auf der Festplatte (z.B. Zylinder- oder Track-Aligned, Clustering)
- Festplatten Scheduling, multi-page Requests
- Prefetching
- Puffer

  und nicht zu vergessen:
- Effiziente und robuste Algorithmen (Implementierungen) der algebraischen Operatoren

Neuere Entwicklungen: SSDs

- Was ändert sich bei SSDs (Solid State Disks) gegenüber traditionellen mechanischen Festplatten?
- **Sehr viel mehr IOPs** (Input/Output Operations Per Second) möglich, Unterschied in Größenordnungen: Z.B. 1000 Leseoperationen a 4KB Block pro Sekunde einer SSD gegenüber 100 pro Sekunde einer normalen Festplatte.
- Dennoch sequenzielles Lesen günstiger als wahlfreies Lesen.
- **Was ändert dies für die Anfrageoptimierung?** Siehe Papier unten.

	Disk	Flash
Model	WD VelociRaptor 10Krpm	OCZ RevoDrive
Capacity	300gb	120gb
Price	$164	$300
Random Read	10ms	90µs
Seq. Read	120mb/s	190mb/s

Pelley et al. Do Query Optimizers Need to be SSD-aware?, ADMS Workshop, 2011.

Neuere Entwicklungen: In-Memory Datenbanken

In-Memory Datenbanken

- Verfügbarer RAM oft ausreichend groß, um gesamte Datenbank zu halten.
- Oft betrachtet in Zusammenhang mit Mehrkernsystemen und NUMA (**Non Uniform Memory Access**) Architekturen.
- Wo ist der neue **Flaschenhals für die Performance**?

Physische Organisation einer Datenbank

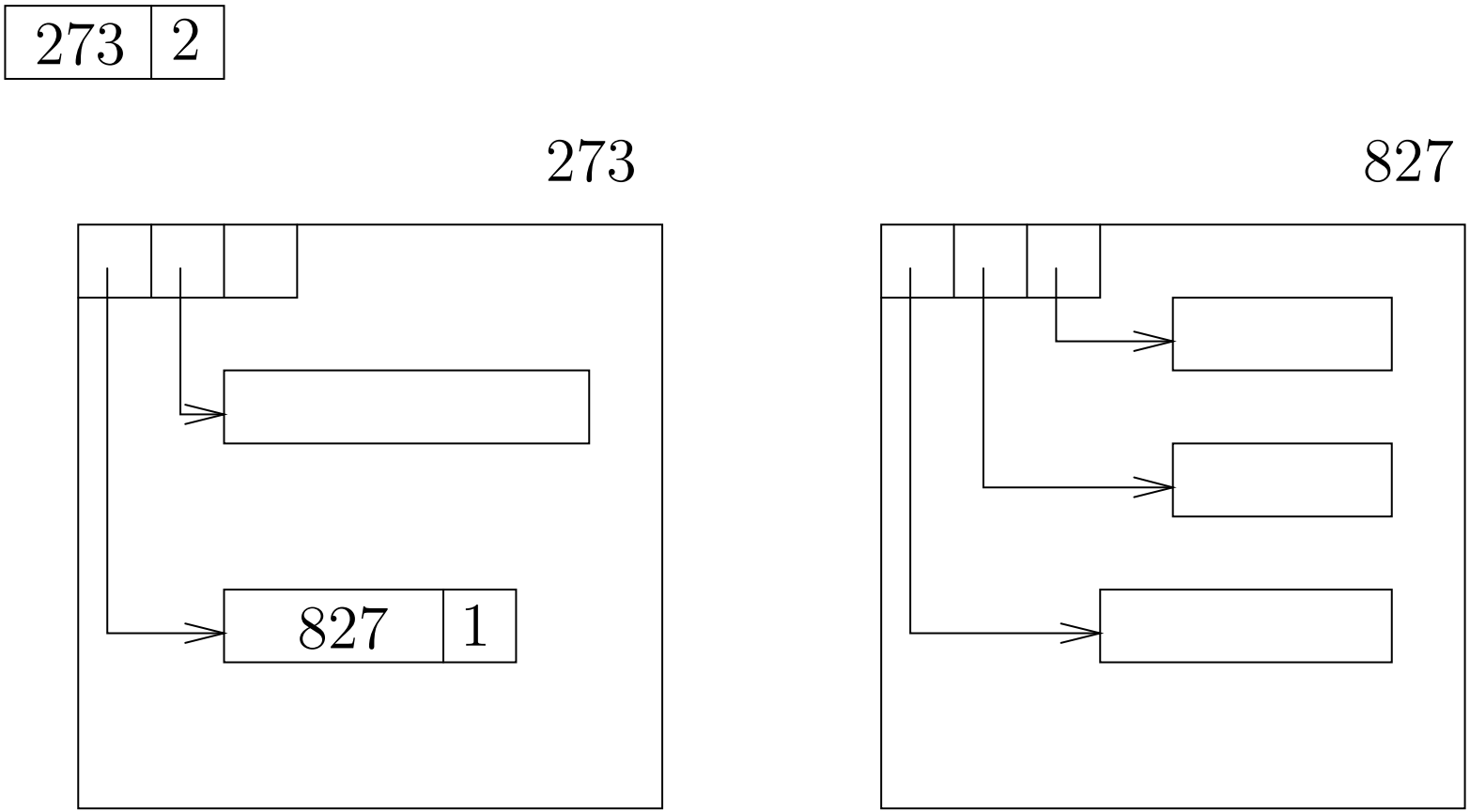
Das Datenbanksystem organisiert den physikalischen Speicher in verschiedene Schichten.

- **Datenbank**: Menge von Dateien
- **Datei**: Sequenz von Blöcken.
- **Segmente**: Organisationseinheit im DBMS (bzgl. Sperren, Rechten, etc.)

Zugriffseinheiten

- **Segmente**
- **Seiten** werden in Segmenten gespeichert
- **Seite** enthält Sätze
- **Satz**: In einer Seite gespeicherte Sequenz von Bytes. Menge von echten Daten, verschiedene Felder.
- Bzw. man redet auch von **Tupeln** im DB (Relationen) Kontext

Seite



- Seite (Page) ist organisiert in Anzahl von Bereiche (Slots)
- Slots zeigen auf Daten
- ... oder auch auf andere Seiten

Tuple Identifier (TID)

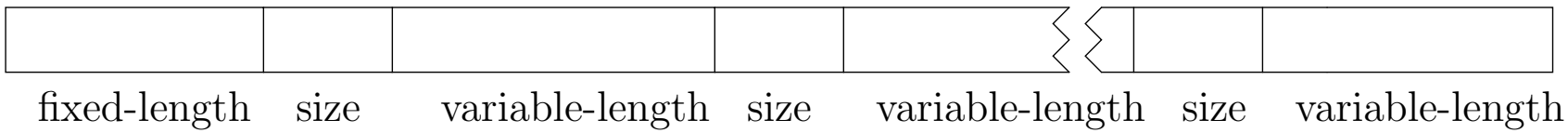
Ein TID ist eine Paar bestehend aus

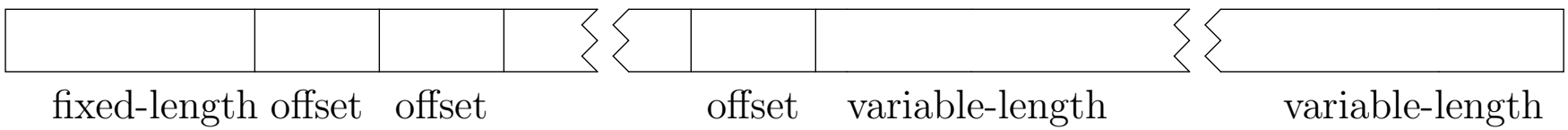
- **Seiten ID** (z.B. Datei/Segment Nummer plus Nummer der Seite)
- **Slot Nummer**

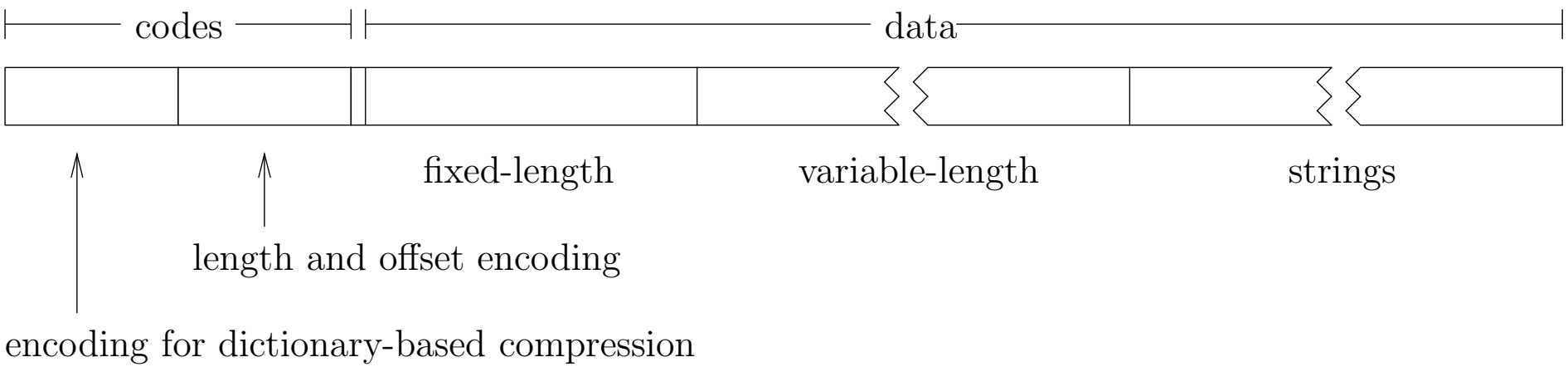
TID wird manchmal auch Row Identifier (RID) genannt

Satz Layout

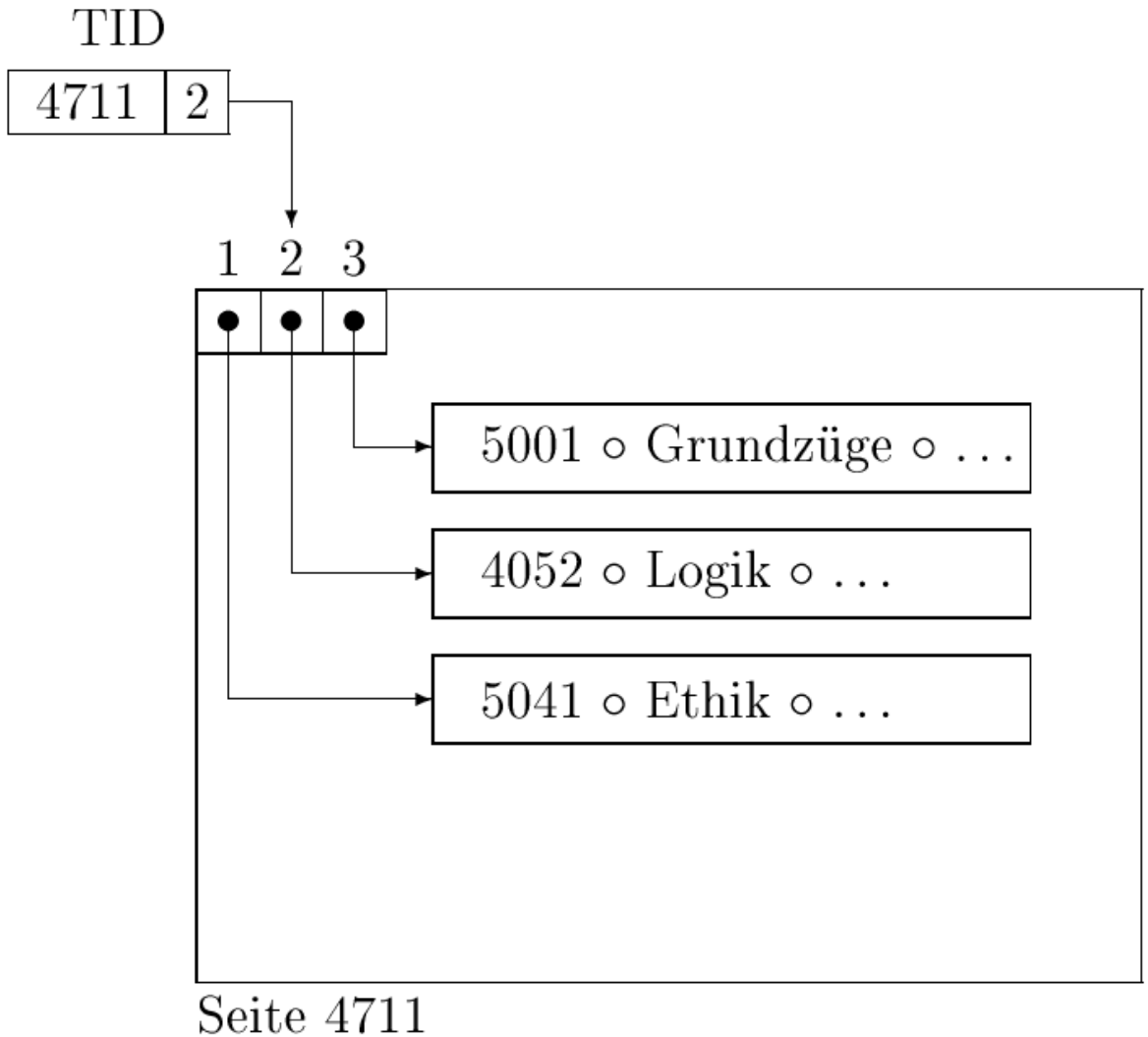
Verschiedene mögliche Layouts:



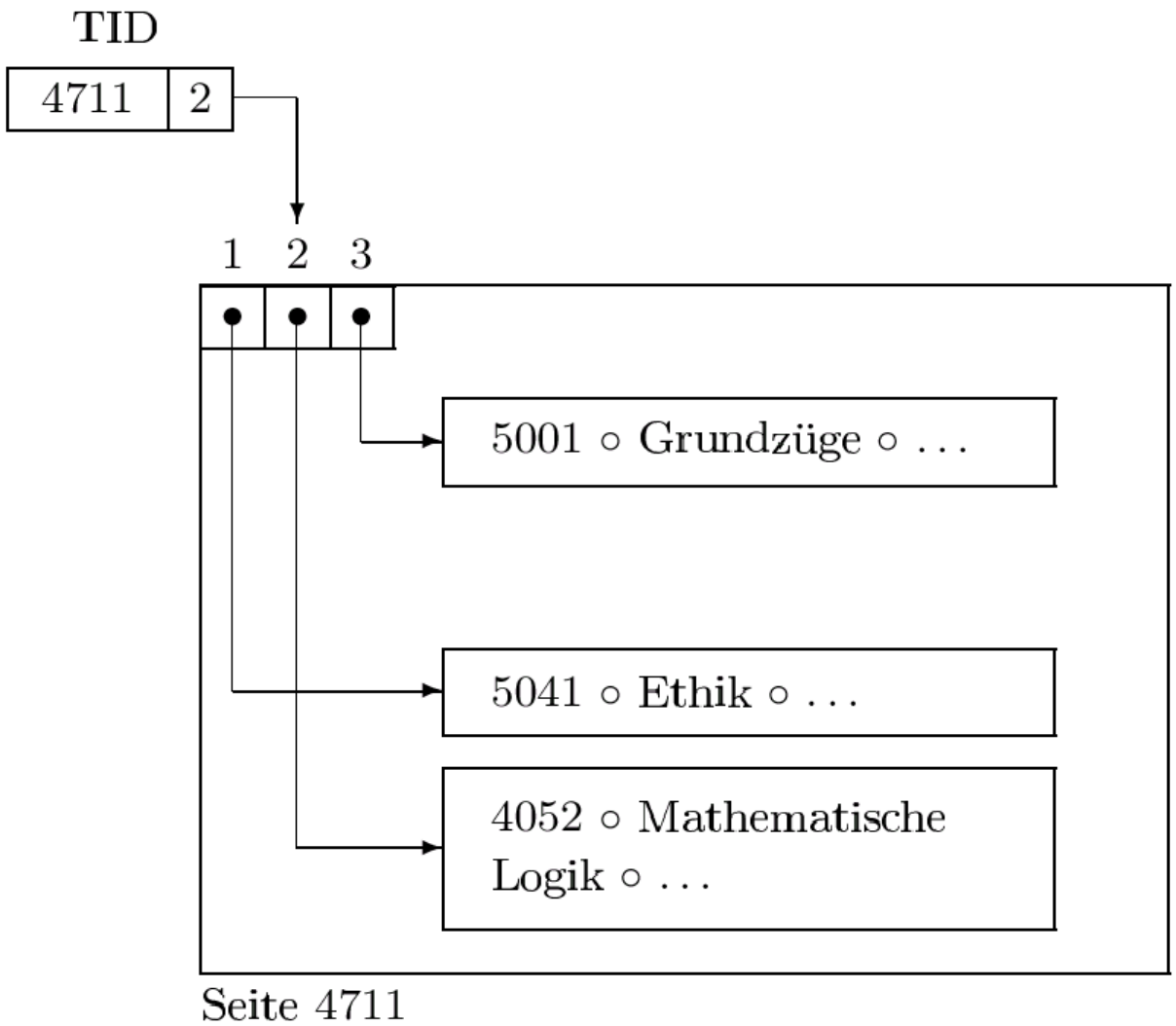




Speicherung von Tupeln auf Seiten



Verschieben von Tupeln innerhalb einer Seite



Verdrängen von Tupeln auf andere Seiten

- Falls eine Seite zu klein wird.
- Verschiebe Tupel in eine andere (z.B. neue, leere) Seite
- Füge in ursprünglicher Seite eine TID hinzu die auf den neuen Ort verweise

Was passiert bei mehrfachem Verschieben?

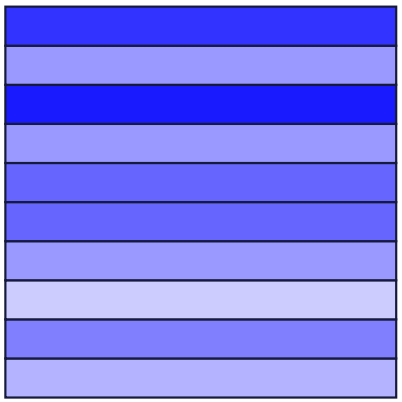
- Verweis in der ursprünglichen Seite wird angepasst.

$\Rightarrow$ Länge der Verweiskette auf zwei beschränkt

Organisation der Dateien

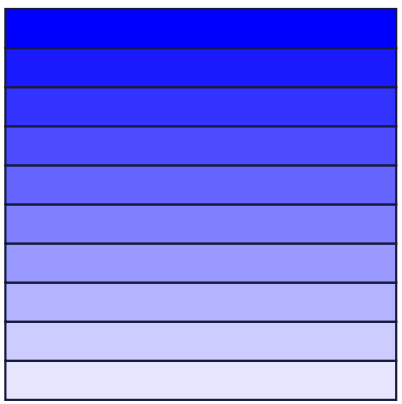
Haufen

- Einfachster Weg eine Datei zu organisieren ist neu ankommende Sätze in die Seite (den Block) am Ende der Datei einzufügen.
- Einfügen ist sehr effizient.
- Suche ist teuer.



Sortierte Dateien

- Gegeben ein Schlüssel anhand dessen Sätze geordnet werden können
- Effizientere Suche (binär) - Obwohl so nicht realisiert (siehe z.B. B+ Baum)



Beispiel: Sortierte Datei (Hier nach Name)

	Name	MatrNr	Semester
Block 1	Aaron	443421	10
	Adam	233499	1
	...		
	Acosta	561921	1
Block 2	Allen	581722	9
	Anderson	339163	8
	...		
	Archer	965492	1
Block 3	Arnold	672961	3
	Arnold	759311	1
	...		
	Atkins	173522	8
	⋮		
Block n	Wright	672961	4
	Wyatt	197646	7
	...		
	Zimmer	524145	12

Suche in Dateien

"If you don't find it in the index,
look very carefully through the entire catalog"
— Sears, Roebuck, and Co., Consumers' Guide, 1897

(aus dem Buch von Ramakrishnan & Gehrke)

Lineare Suche

Binäre Suche in sortierten Dateien

Suche via Indexen

Grundidee eines Index

Abbildung: Schlüssel $\rightarrow$ Menge von Einträgen

Beispiele:

- MatrikelNummer $\rightarrow$ persönliche Daten des Studenten
- PLZ $\rightarrow$ Name und andere Informationen einer Stadt
- Term $\rightarrow$ alle Dokumente in denen diesesr Term enthalten ist

Natürlich möchte man bei diesen häufig auftretenden Anfragen die Antwortzeit gering halten.

Dafür wird ein Index angelegt: Ein Index materialisiert diese Abbildung!

Betrachten wir im nächsten Kapitel.