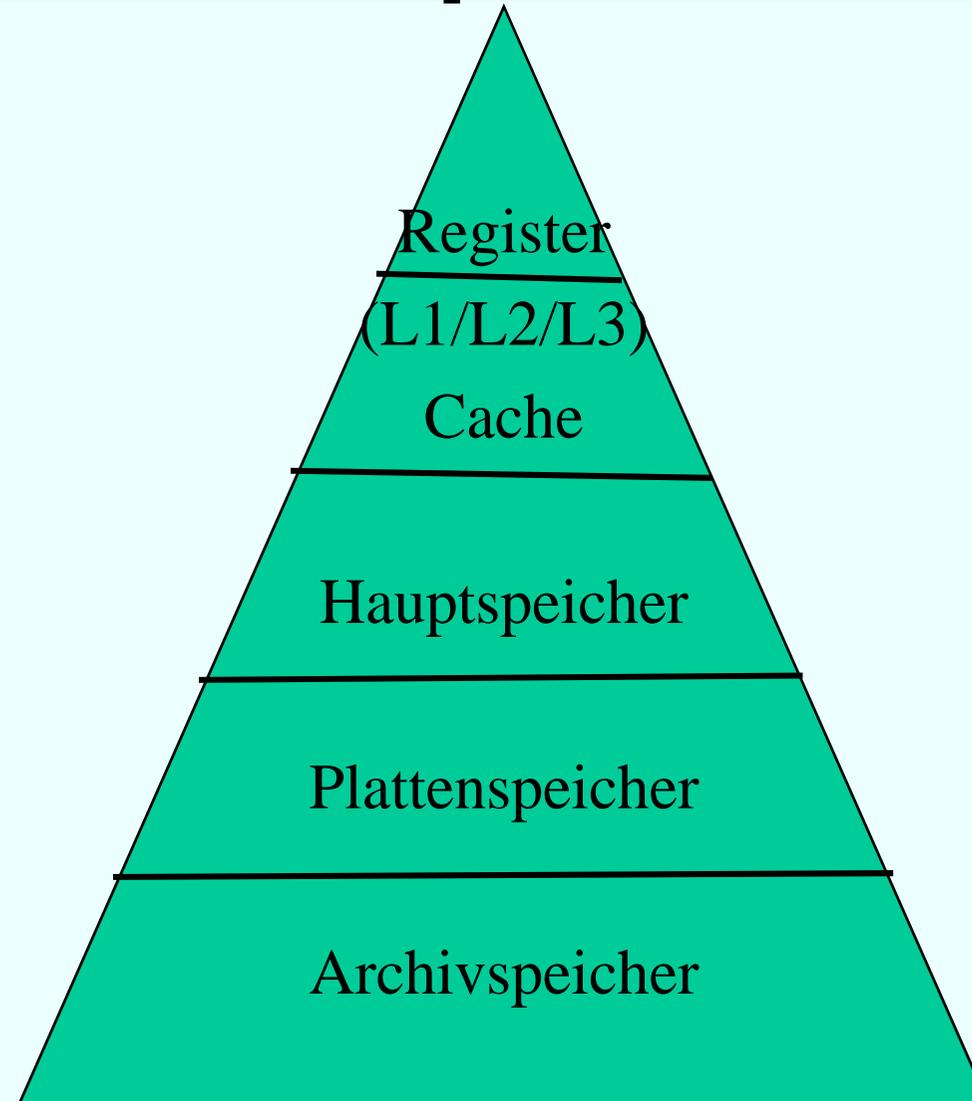


Physische Datenorganisation

Themenbereiche:

- Speicherhierarchie
- Hintergrundspeicher
- Speicherstrukturen
- ISAM
- B-Bäume
- Hashing
- Clustering

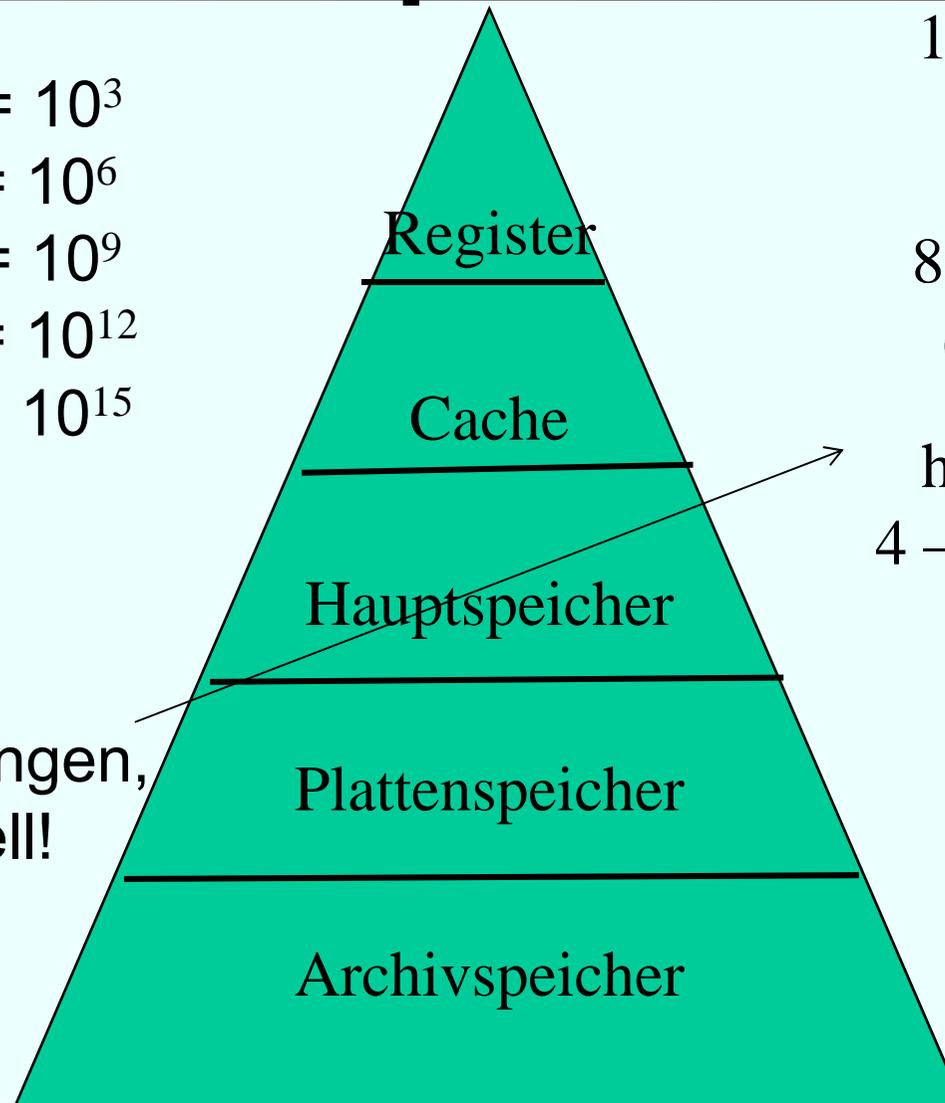
Überblick: Speicherhierarchie



Überblick: Speicherhierarchie

1 K (Kilo) = 10^3
1 M (Mega) = 10^6
1 G (Giga) = 10^9
1 T (Tera) = 10^{12}
1 P (Peta) = 10^{15}

Ungefähre
Größenordnungen,
veraltet schnell!



1 – 8 Byte/Register
Compiler

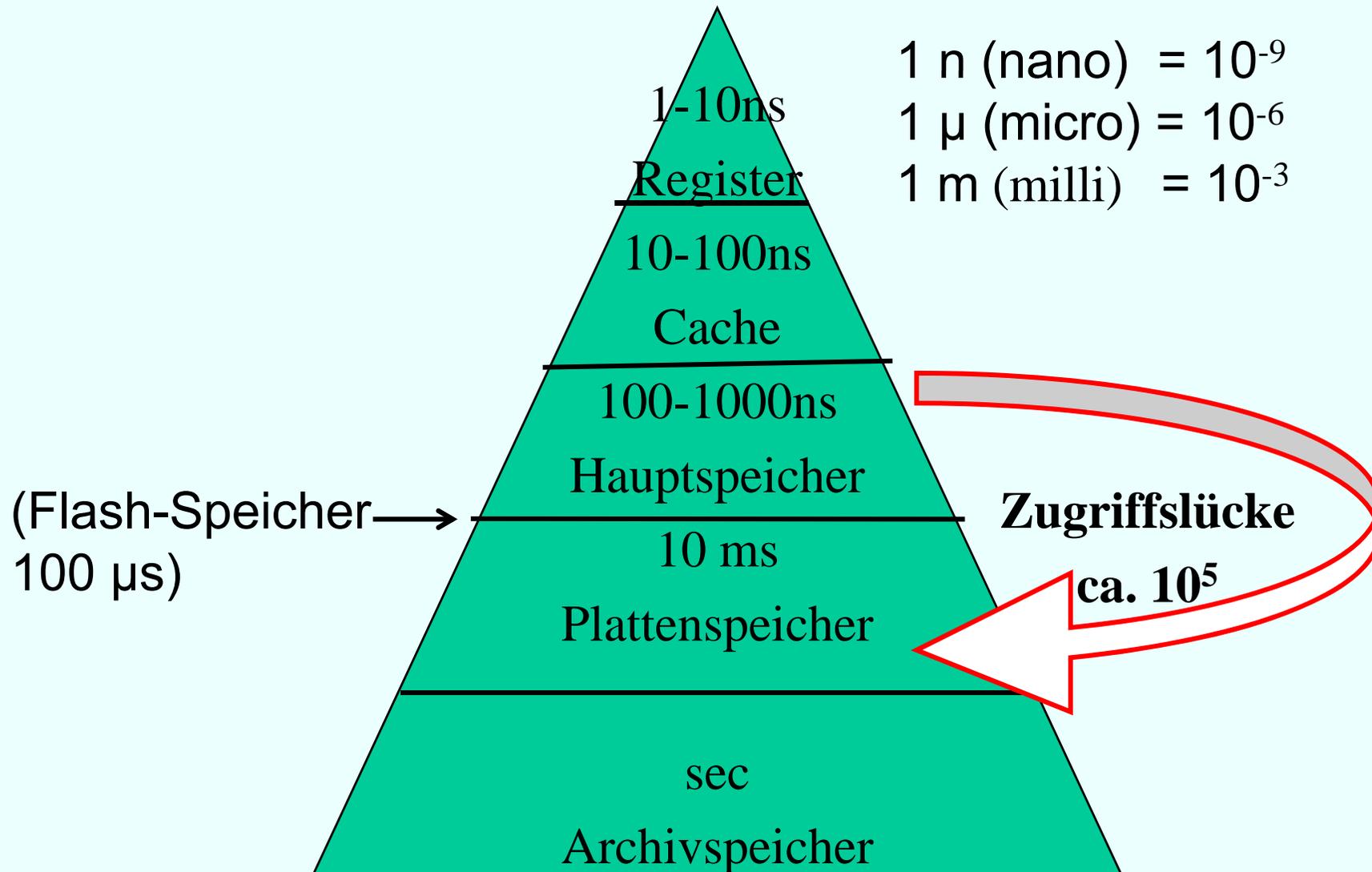
8 – 128 Byte/Cache
Cache-Controller

hoher GB-Bereich,
4 – 64 KB Blockgröße
Betriebssystem

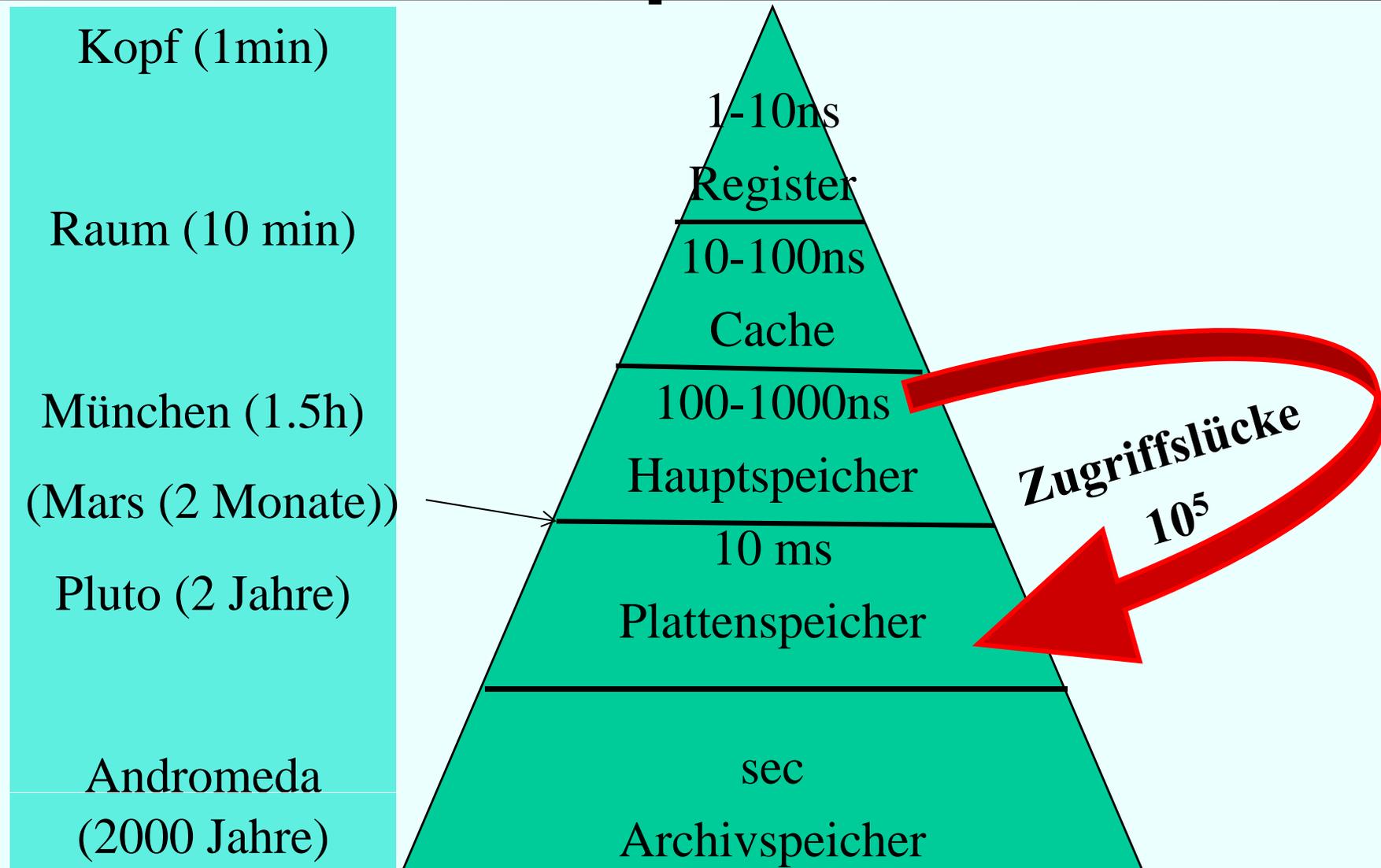
im Bereich TB
Benutzer

im Bereich PB
Benutzer

Überblick: Speicherhierarchie



Überblick: Speicherhierarchie



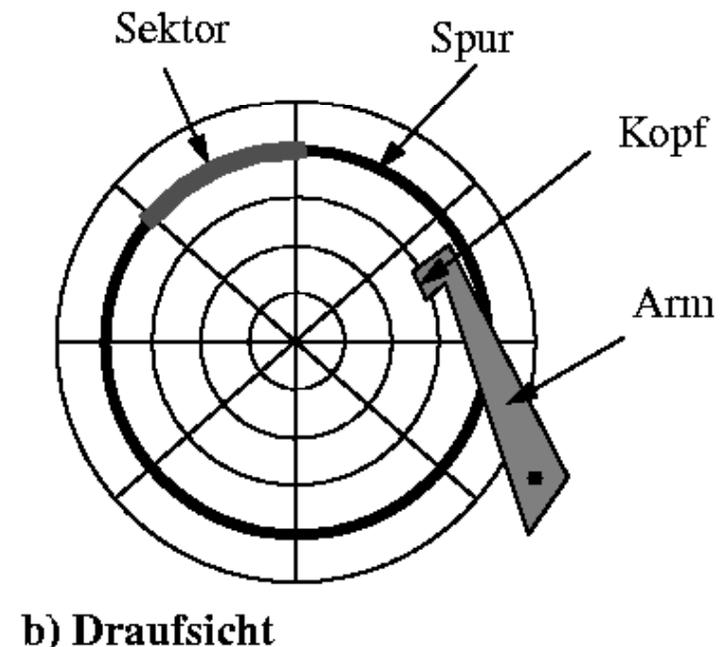
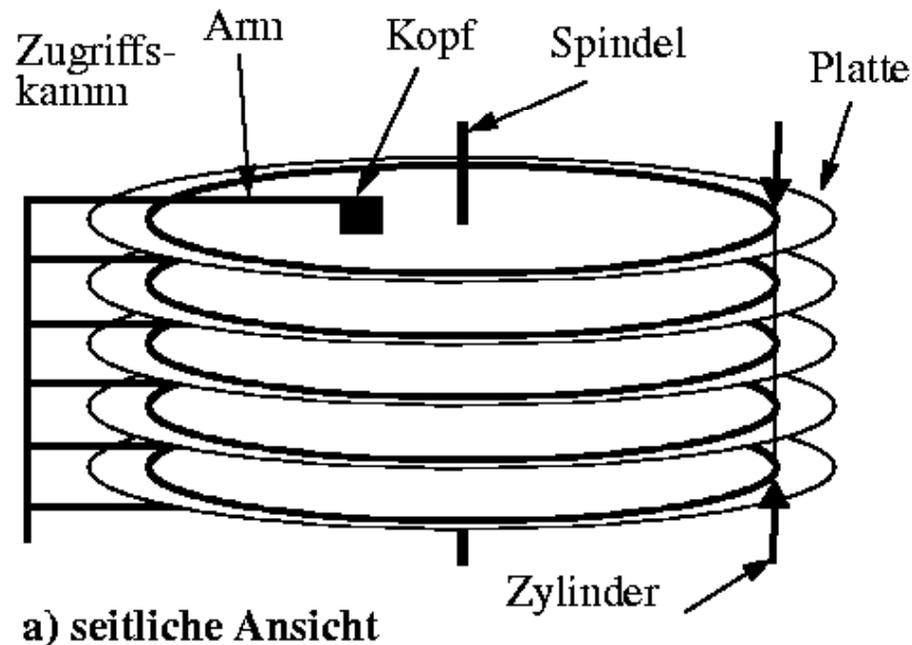
Magnetplattenspeicher

Aufbau

- mehrere gleichförmig rotierende Platten, für jede Plattenoberfläche ein Schreib-/Lesekopf
- jede Plattenoberfläche ist eingeteilt in Spuren
- die Spuren sind formatiert als Sektoren fester Größe (Slots)
- Sektoren (typischerweise 1 - 8 KB) sind die kleinste Schreib-/Leseinheit auf einer Platte

Adressierung

- Zylindernummer, Spurnummer, Sektornummer
- jeder Sektor speichert selbstkorrigierende Fehlercodes; bei nicht behebbaren Fehlern erfolgt automatische Abbildung auf Ersatzsektoren



Lesen von Daten von der Platte

Seek Time: Arm/Kopf positionieren

Latenzzeit: Rotation zum Anfang des Blocks
 $\frac{1}{2}$ Plattenumdrehung (im Durchschnitt)

Lesezeit: Transfer von der Platte zum Hauptspeicher

Random versus Chained IO

Random I/O

Jedesmal Arm positionieren

Jedesmal Latenzzeit

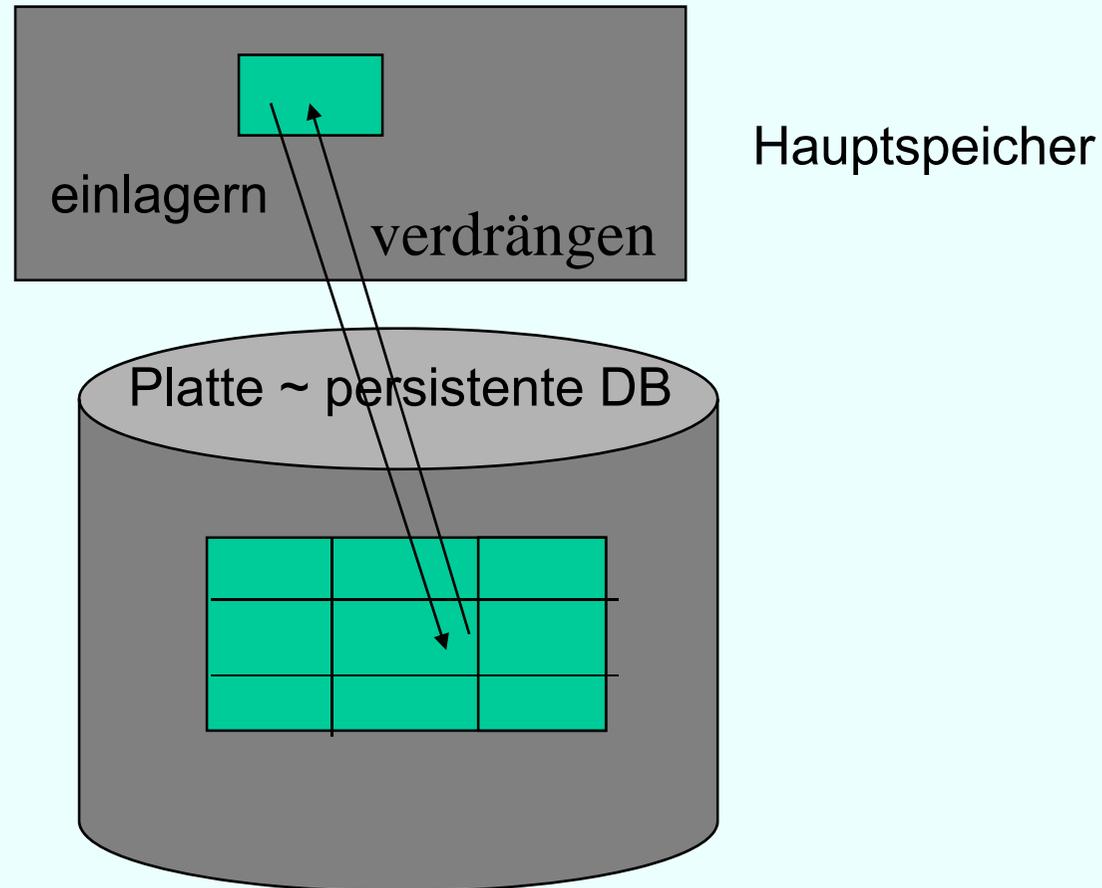
Chained IO

Einmal positionieren, dann „von der Platte kratzen“

Chained IO ist **ein bis zwei Größenordnungen schneller** als random IO

→ in Datenbank-Algorithmen beachten !

Systempuffer-Verwaltung



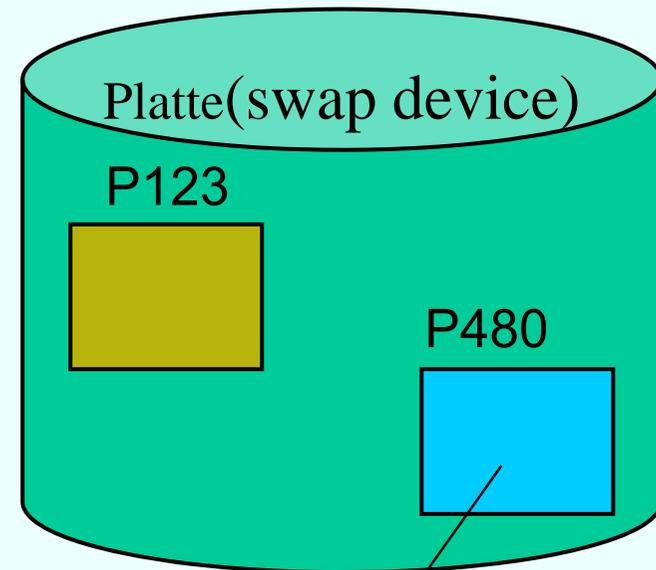
Ein- und Auslagern von Seiten

- Systempuffer ist in Seitenrahmen gleicher Größe aufgeteilt
- Ein Rahmen kann eine Seite aufnehmen
- „Überzählige“ Seiten werden auf die Platte ausgelagert

Hauptspeicher

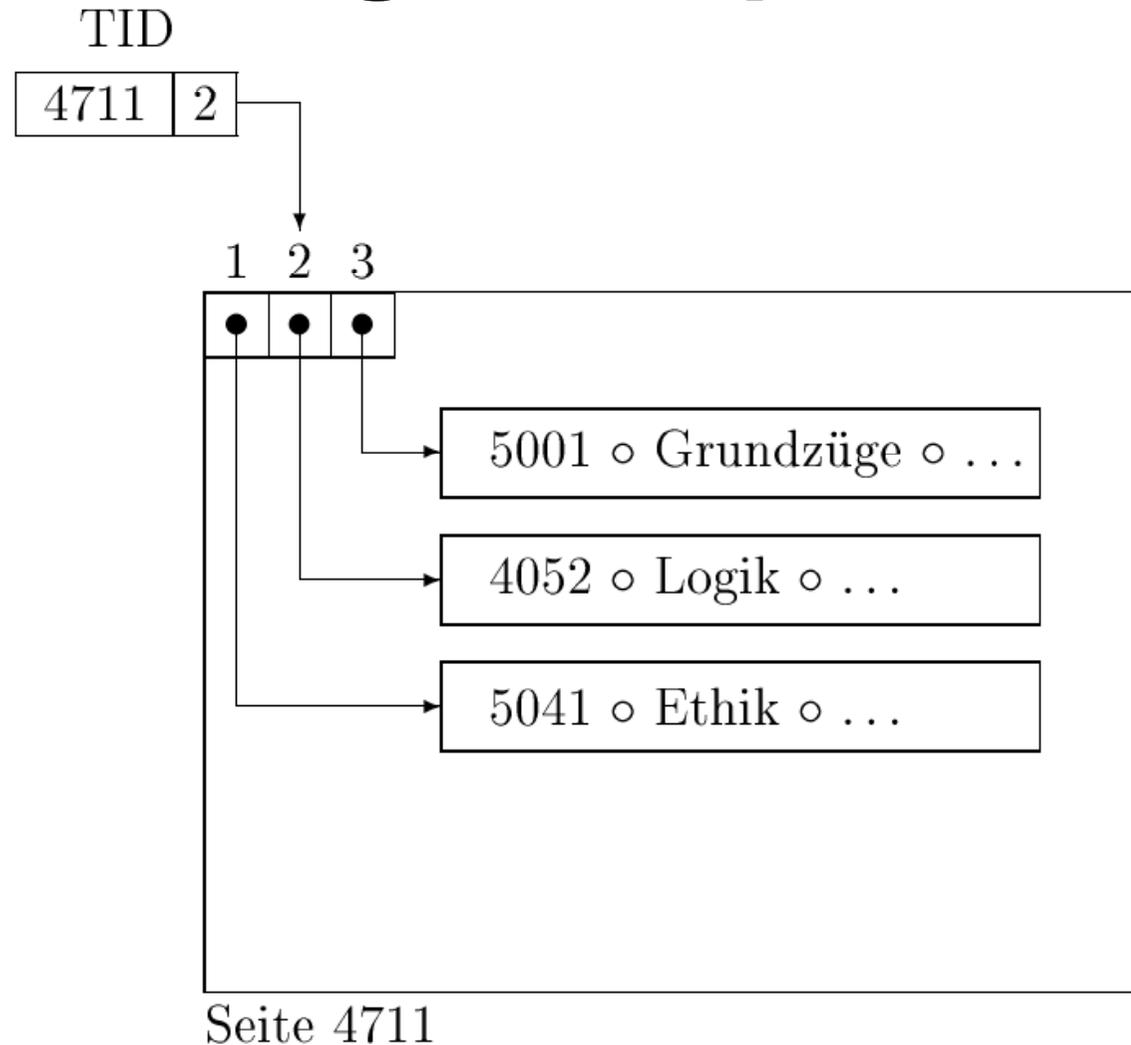
0	4K	8K	12K
16K	20K	24K	28K
32K	36K	40K	44K
48K	52K	56K	60K

Seitenrahmen

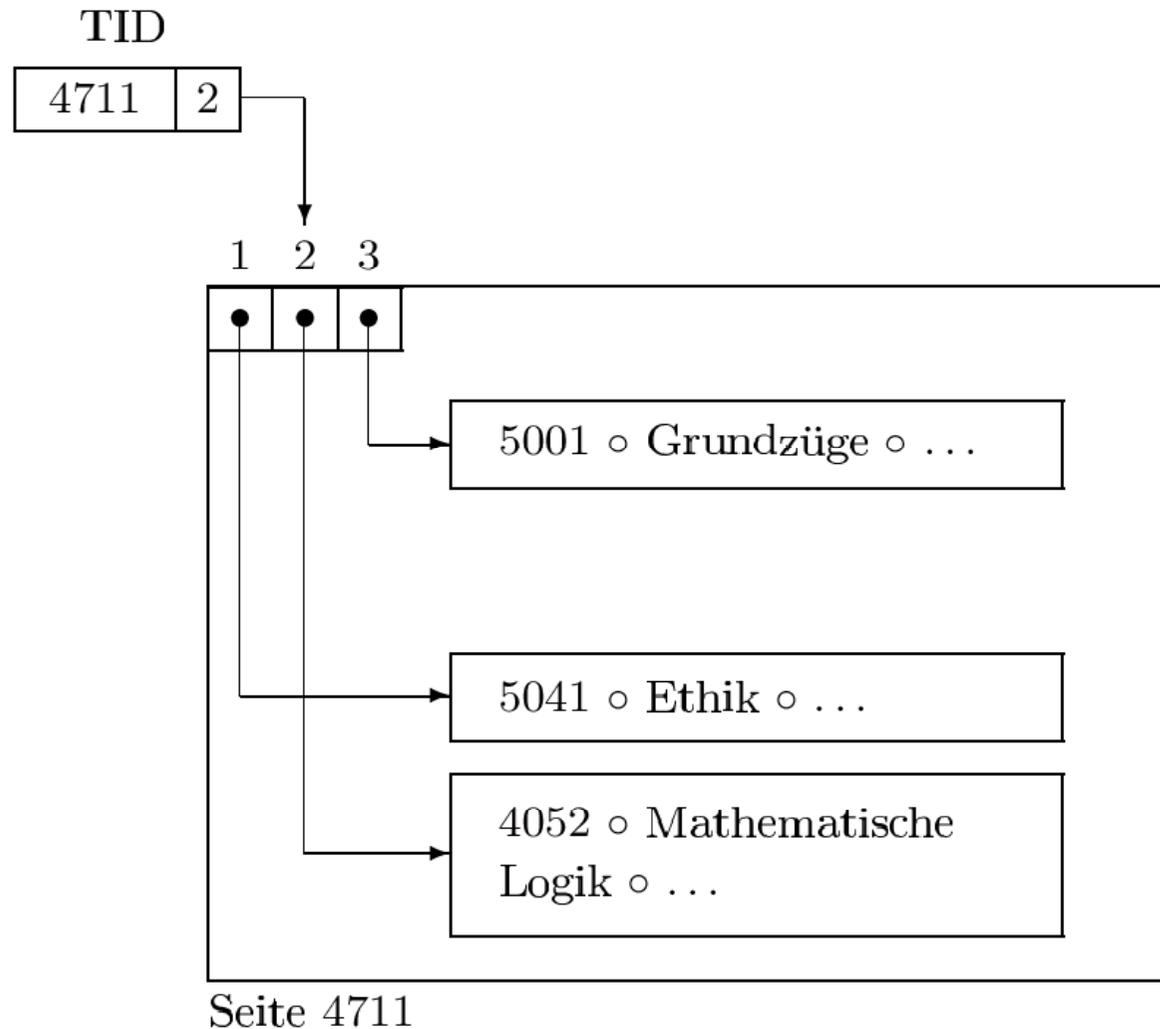


Seite

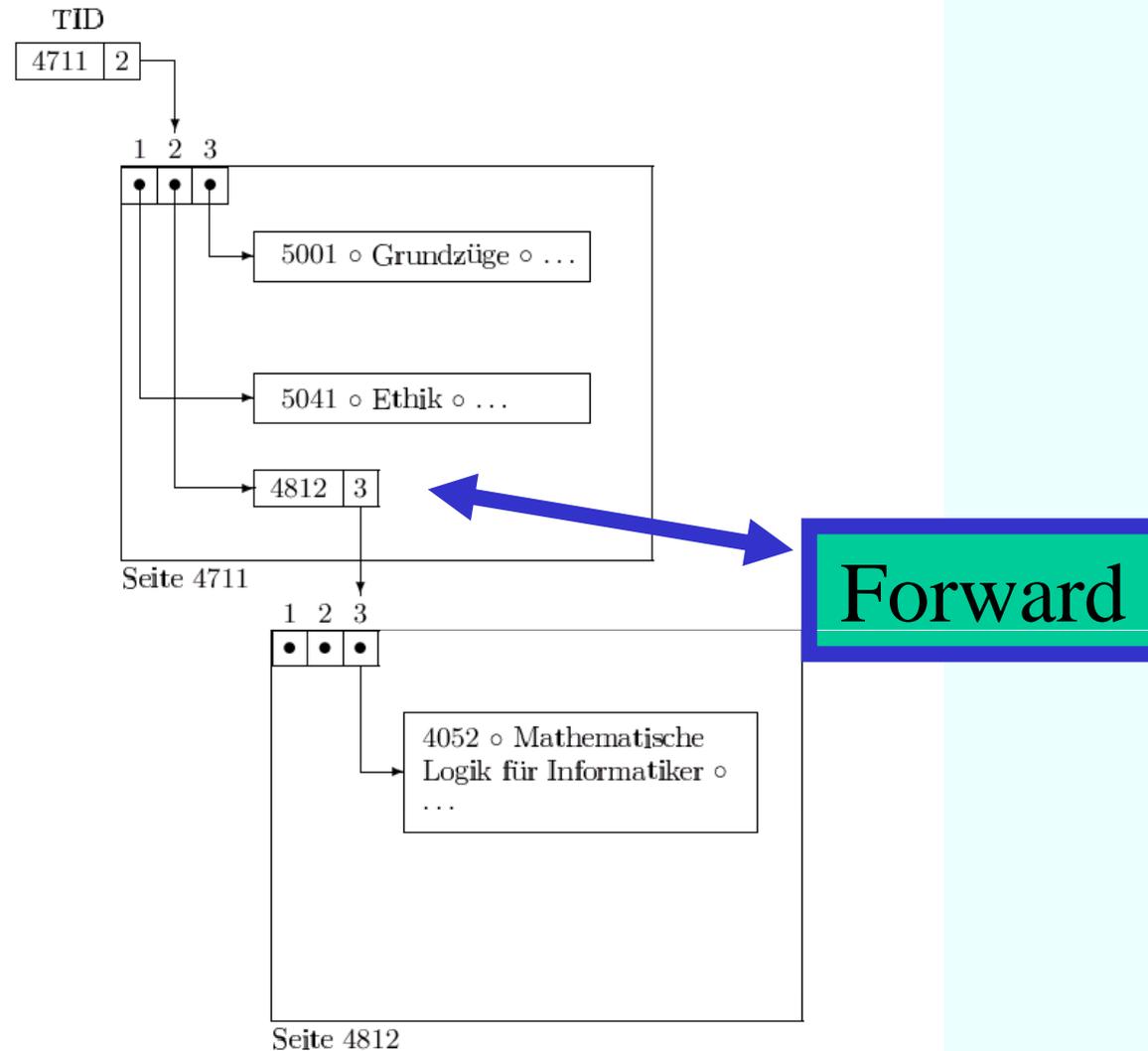
Adressierung von Tupeln auf dem Hintergrundspeicher



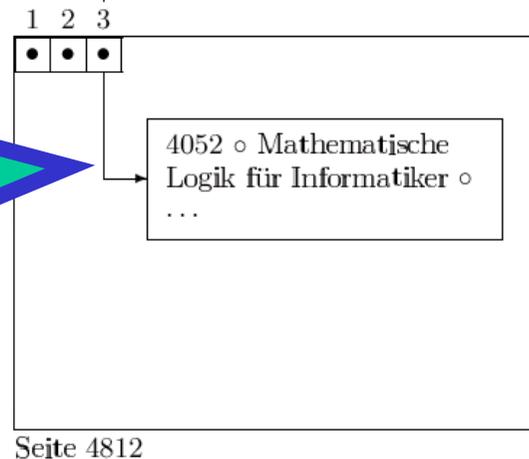
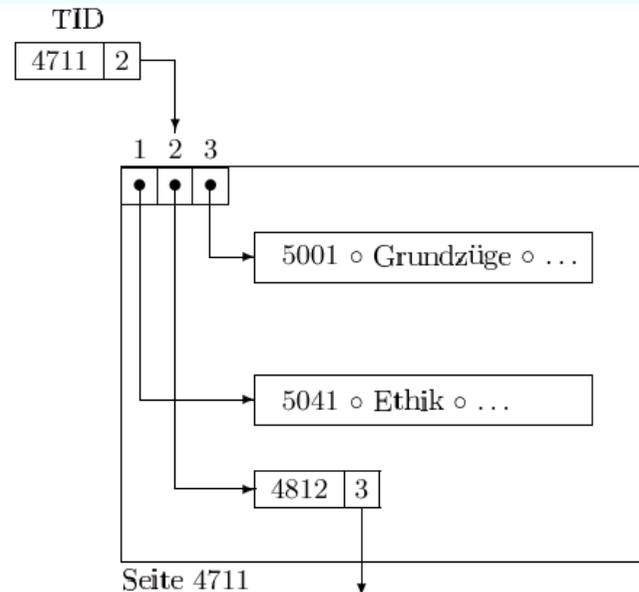
Verschiebung innerhalb einer Seite



Verschiebung von einer Seite auf eine andere



Verschiebung von einer Seite auf eine andere



Bei der nächsten Verschiebung wird der „Forward“ auf Seite 4711 geändert (kein Forward auf Seite 4812)

Datentransfer

Einfache Anfragebeantwortung:

Tupel aller beteiligten Relationen nacheinander in den Hauptspeicher holen

→ die teuerste Art ☹

Bei näherer Betrachtung stellt man folgendes fest:

- Oft erfüllt nur ein Bruchteil der Tupel die Anfragebedingungen
- Anfragen haben oft ähnliche Prädikate
- Festplatten erlauben wahlfreien Zugriff

Indexstrukturen

- Indexstrukturen nutzen diese Eigenschaften von Anfragen aus, um das transferierte Datenvolumen klein zu halten
- Sie erlauben schnellen assoziativen Zugriff auf die Daten
- Nur der Teil der Daten, der zur Beantwortung der Anfrage wirklich gebraucht wird, wird in den Hauptspeicher geholt
- Zwei bedeutende Indexierungsansätze
 - Hierarchisch (Bäume)
 - Partitionierung (Hashing)

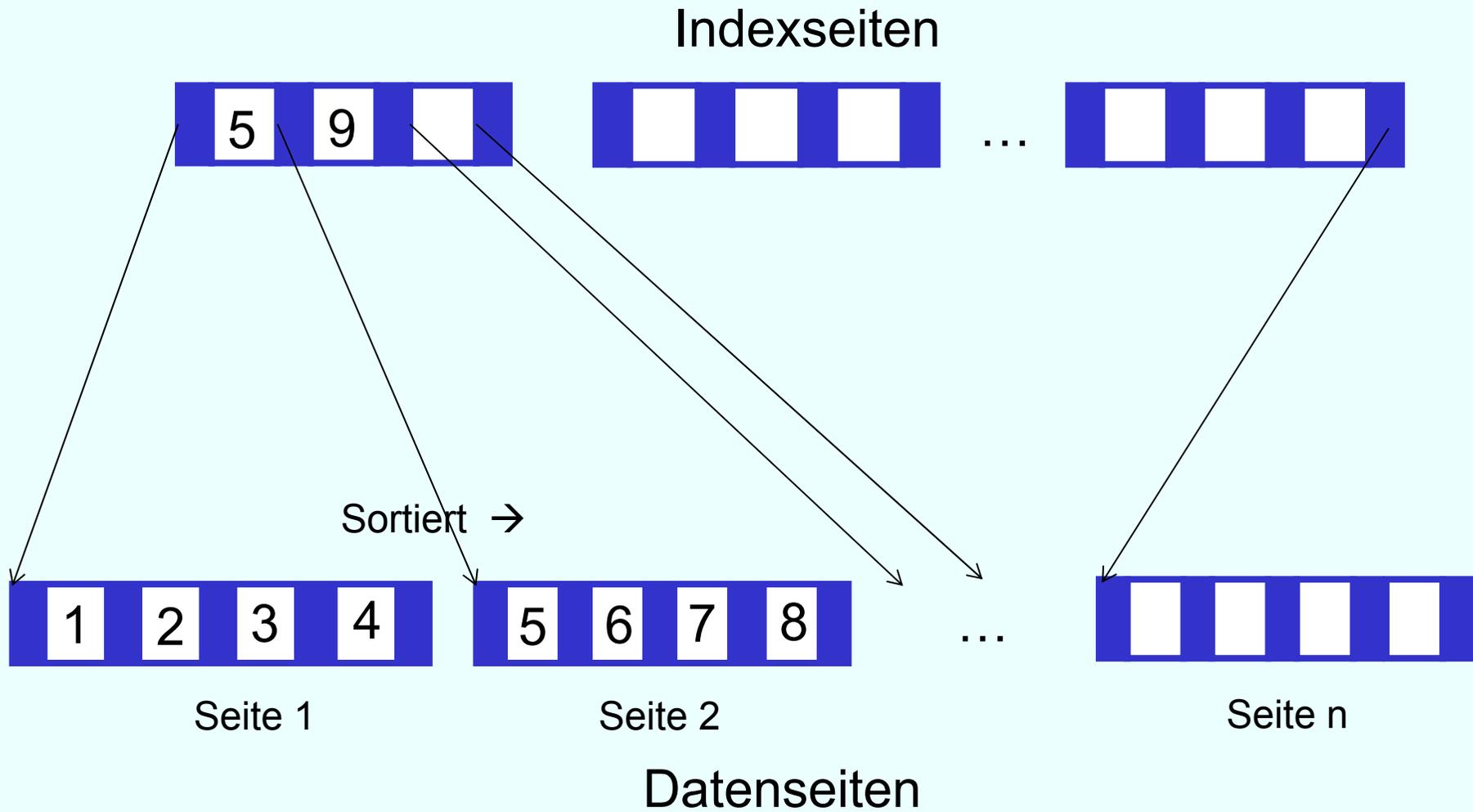
Hierarchische Indexe

Wir betrachten zwei hierarchische Indexstrukturen:

- ISAM (Index-Sequential Access Method)
- B-Bäume

- ISAM war Vorgänger von B-Bäumen
- Hauptidee ist die Tupel auf dem indexierten Attribut zu sortieren und eine Indexdatei darüber anzulegen
- Ähnlich wie ein Daumenindex an der Seite eines Buches, durch den man schnell durchblättern kann

Beispiel



Beispiel cont.

- Student mit Matrikelnummer 13542 wird gesucht
- Während Anfragebearbeitung geht man durch die **Indexseiten** und sucht die Stelle, an der 13542 passt
- Von dort aus wird die referenzierte **Datenseite** geholt
- **Vorteil:** Anzahl der Indexseiten ist normalerweise sehr viel kleiner als die Anzahl der Datenseiten, d.h. es wird I/O gespart
- Es können **auch Bereichsanfragen** beantwortet werden (z.B. bei einer Suche nach allen MatrNr zwischen 765 und 1232: zuerst die erste passende Datenseite finden und von dort aus sequentiell durch die Datenseiten bis MatrNr 1232 laufen)

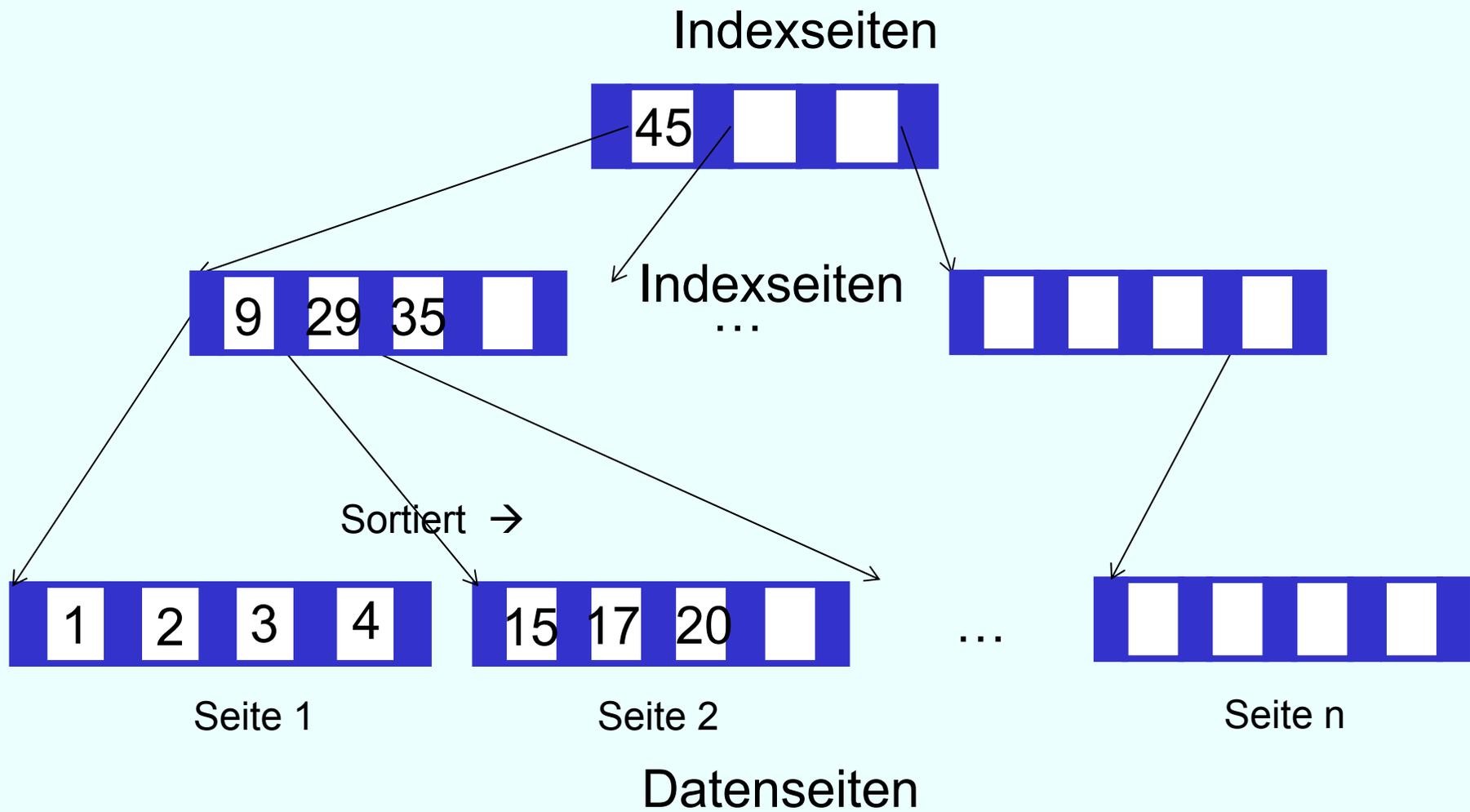
Probleme mit ISAM

- Obwohl Suche auf ISAM einfach und schnell ist, kann die **Instandhaltung des Indexes teuer** werden
- Wenn ein Tupel auf eine gefüllte Datenseite eingefügt werden soll, muss **Platz geschaffen werden**: die Datenseite wird auf zwei Seiten aufgeteilt (wir müssen Sortierung beibehalten)
- Das erzeugt wiederum einen **neuen Eintrag** auf einer **Indexseite**
- Wenn auf der Indexseite auch kein Platz mehr ist, müssen die **Einträge verschoben** werden, um Platz zu schaffen

Weitere Probleme

- Obwohl die Anzahl der Indexseiten kleiner als die Anzahl der Datenseiten, kann **Durchlauf der Indexseiten** trotzdem **lange dauern**
- Idee: warum richtet man nicht **Indexseiten für die Indexseiten** ein?
- Das ist im Prinzip die Idee eines **B-Baums**

Idee



B-Bäume

Bäume in der Informatik

... haben Knoten

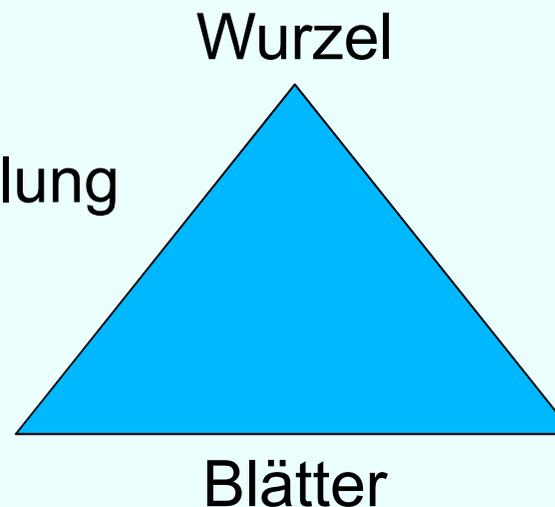
... haben Kanten

... haben eine Wurzel (oben!)

... haben Blätter (unten!)

... sind balanciert oder nicht (dann eher Kette)

Schematische Darstellung
balancierter Baum:



Eigenschaften B-Baum

Ein B-Baum vom Grad i hat folgende Eigenschaften:

- Jeder Pfad von der Wurzel zu einem Blatt hat die gleiche Länge
- Jeder Knoten (außer der Wurzel) hat mindestens i und höchstens $2i$
- Einträge (in obigem Beispiel $i = 2$)
- Die Einträge in jedem Knoten sind sortiert
- Jeder Knoten (außer Blätter) mit n Einträgen hat $n + 1$ Kinder

Eigenschaften B-Baum

- Seien $p_0, k_1, p_1, k_2, \dots, k_n, p_n$ die Einträge in einem Knoten (p_j sind Zeiger, k_j Schlüssel)
Dann gilt folgendes:
 - Der Unterbaum der von p_0 referenziert wird, enthält nur Schlüssel kleiner als k_1
 - p_j zeigt auf einen Unterbaum mit Schlüsseln zwischen k_j und k_{j+1}
 - Der Unterbaum der von p_n referenziert wird, enthält nur Schlüssel größer als k_n

Einfügealgorithmus

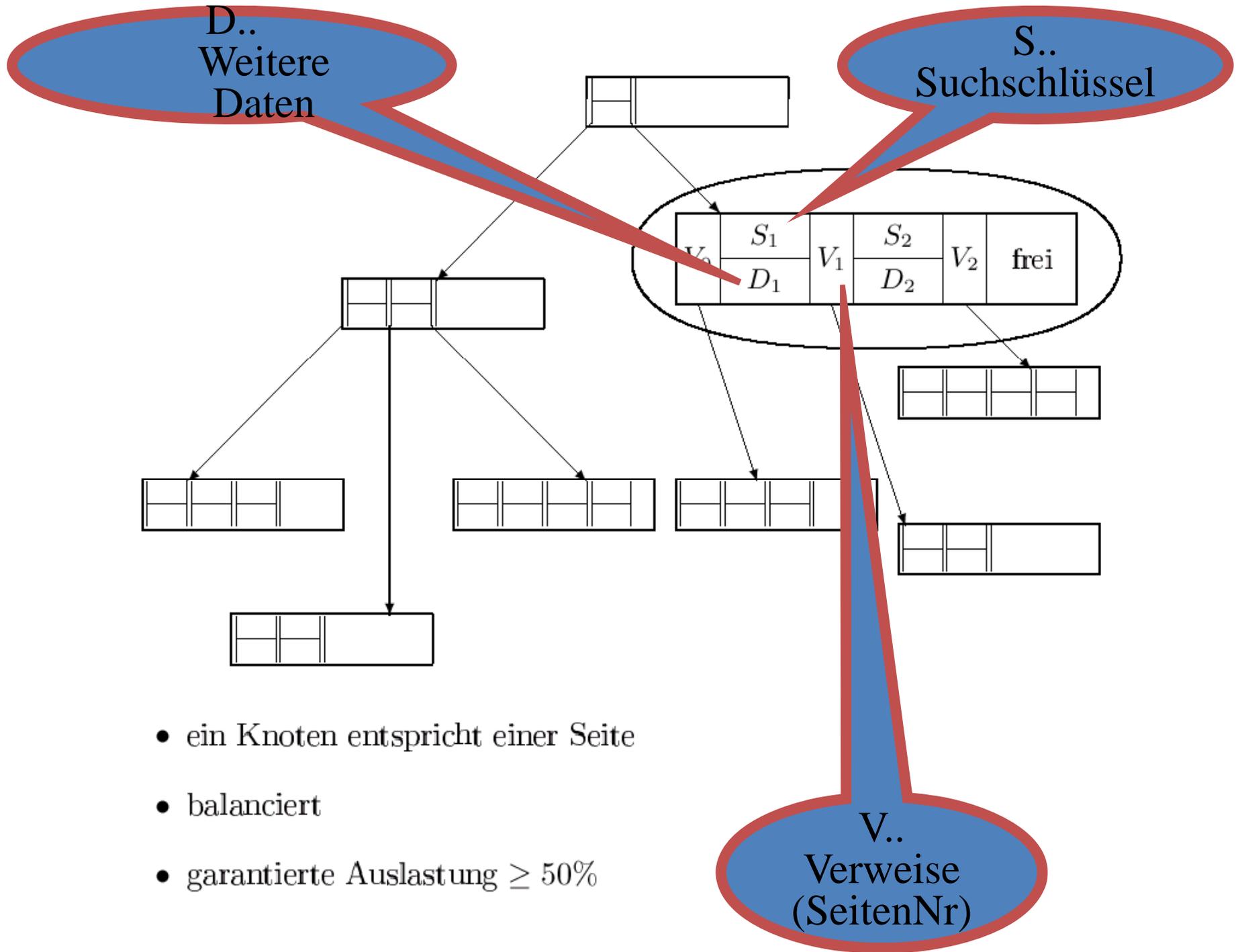
1. Finde den richtigen Blattknoten, um den neuen Schlüssel einzufügen
2. Füge Schlüssel dort ein
3. Falls kein Platz mehr da
 - i. Teile Knoten und ziehe Median heraus
 - ii. Füge alle Knoten kleiner als Median in linken Knoten, alle größer als Median in rechten Knoten
 - iii. Füge Median in Elternknoten ein und passe Zeiger an
4. Falls kein Platz in Elternknoten
 - i. Falls Wurzelknoten, kreierte neuen Wurzelknoten und füge Median ein, passe Zeiger an
 - ii. Ansonsten, wiederhole 3. mit Elternknoten

Löschalgorithmus

- In einem Blattknoten kann ein Schlüssel einfach gelöscht werden
- In einem inneren Knoten muss Verbindung zu den Kindern bestehen bleiben
 - Deshalb wird der nächstgrößere (oder nächstkleinere) Schlüssel gesucht (in entsprechendem Kindknoten)
 - Dieser Schlüssel wird an die Stelle des gelöschten Schlüssels geschrieben

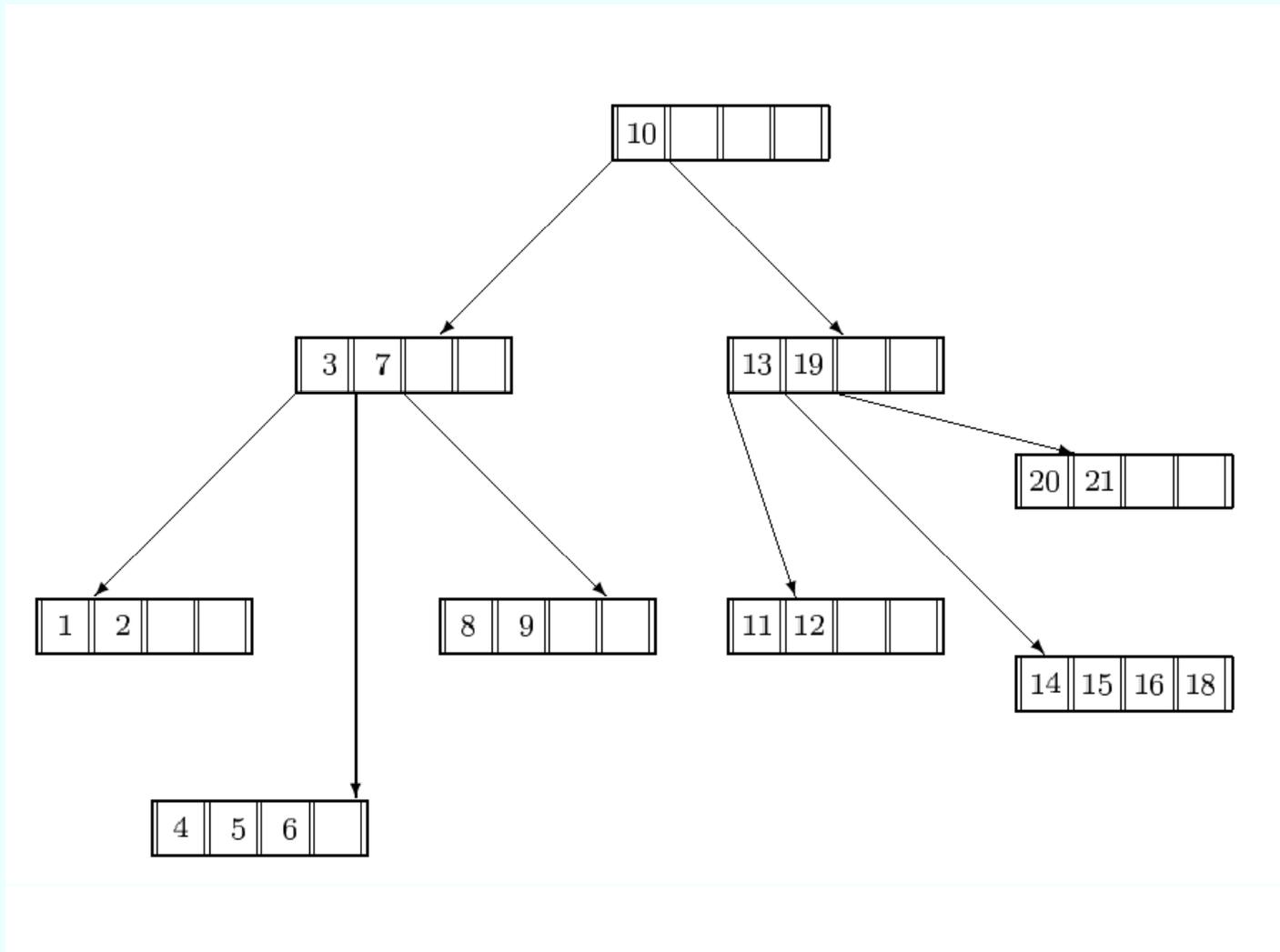
Löschalgorithmus cont.

- Nach Löschen eines Schlüssels kann ein Knoten unterbelegt sein (weniger als i Einträge haben)
- Dann wird dieser Knoten mit einem Nachbarknoten verschmolzen
- Das kann eine Unterbelegung im Elternknoten hervorrufen, d.h. Elternknoten muss ebenfalls verschmolzen werden
- Da dieses Verfahren relativ aufwändig ist und Datenbanken eher wachsen als schrumpfen, wird diese Verschmelzung oft nicht realisiert



- ein Knoten entspricht einer Seite
- balanciert
- garantierte Auslastung $\geq 50\%$

Beispielbaum



Sukzessiver Aufbau eines B-Baums vom Grad $k=2$

Siehe

<http://www->

[db.in.tum.de/research/publications/books/DBMSeinf/EIS](http://www-db.in.tum.de/research/publications/books/DBMSeinf/EIS),
Kapitel 7, ab Folie 51

Im Internet gibt es eine Reihe von

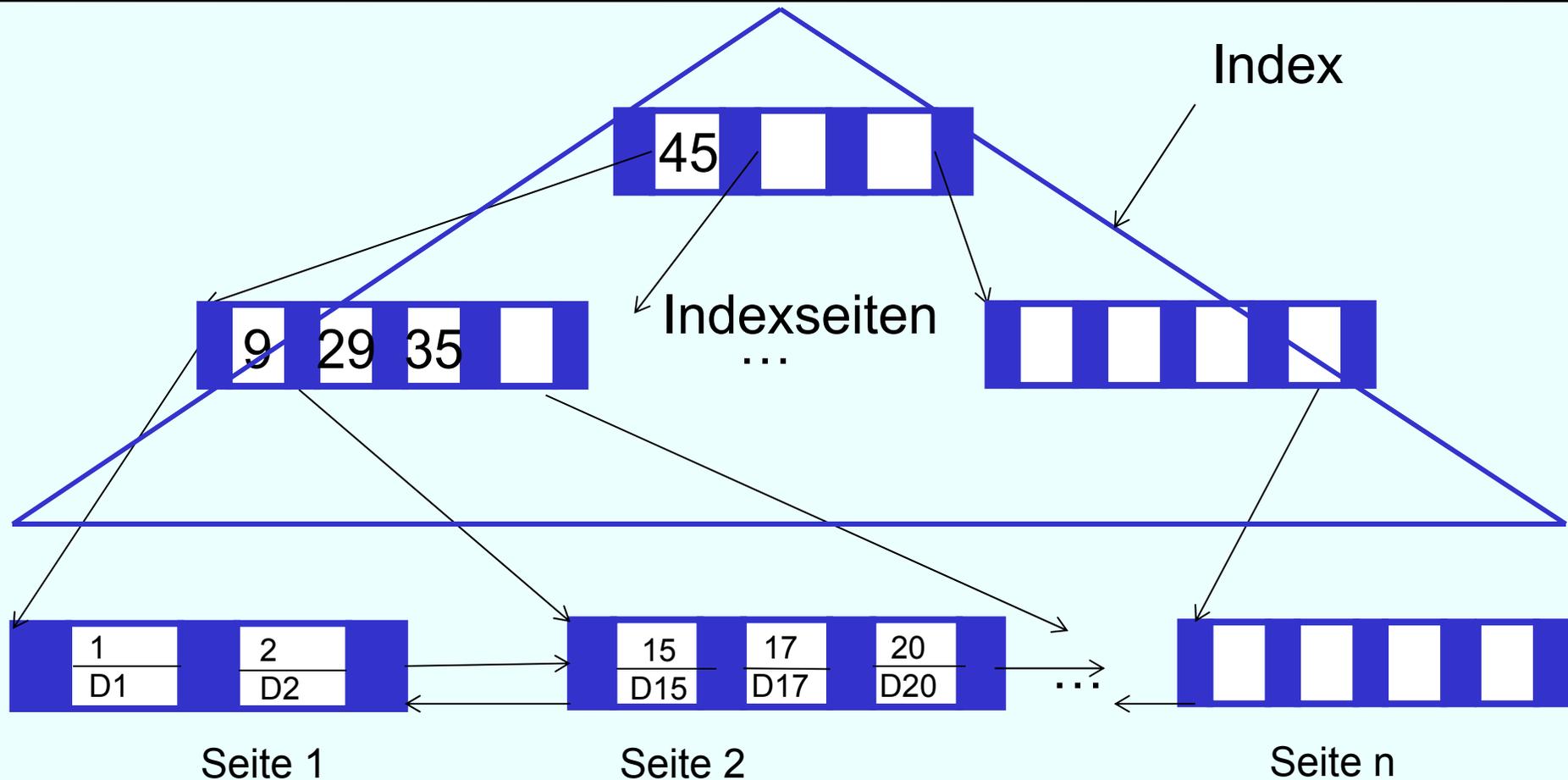
Animationsprogrammen für B-Bäume – **ohne Gewähr!**

<https://www.cs.usfca.edu/~galles/visualization/BTree.html>
sieht ganz gut aus (auch B+-Baum: [.../BPlusTree.html](https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html))

B+-Bäume

- Performanz eines B-Baums ist stark abhängig von der Höhe des Baumes: im Schnitt $\log_k(n)$ Seitenzugriffe, um ein Datenelement zu lesen
(k =Verzweigungsgrad, n =Anzahl indexierter Datensätze)
- → möglichst hoher Verzweigungsgrad der inneren Knoten
- Abspeichern von Daten in inneren Knoten reduziert den Verzweigungsgrad
- B+-Bäume speichern nur Referenzschlüssel in inneren Knoten, die Daten selbst werden in Blattknoten gespeichert
- Meistens sind die Blattknoten noch verkettet, um schnelle sequentielle Suche zu ermöglichen

Struktur B+-Baum



Datenseiten, sortiert,
doppelt verkettet

Präfix B+-Bäume

- Weitere Verbesserung ist der Einsatz von Referenzschlüssel**präfixen**, z.B. bei langen Zeichenketten
- Es muss nur irgendein Referenzschlüssel gefunden werden, der linken vom rechten Teilbaum trennt:
 - Müller-Pfefferkorn $\leq P <$ Schmidtchen
 - Systemprogramm $\leq ? <$ Systemprogrammierer

Mehrere Indexe auf den selben Daten

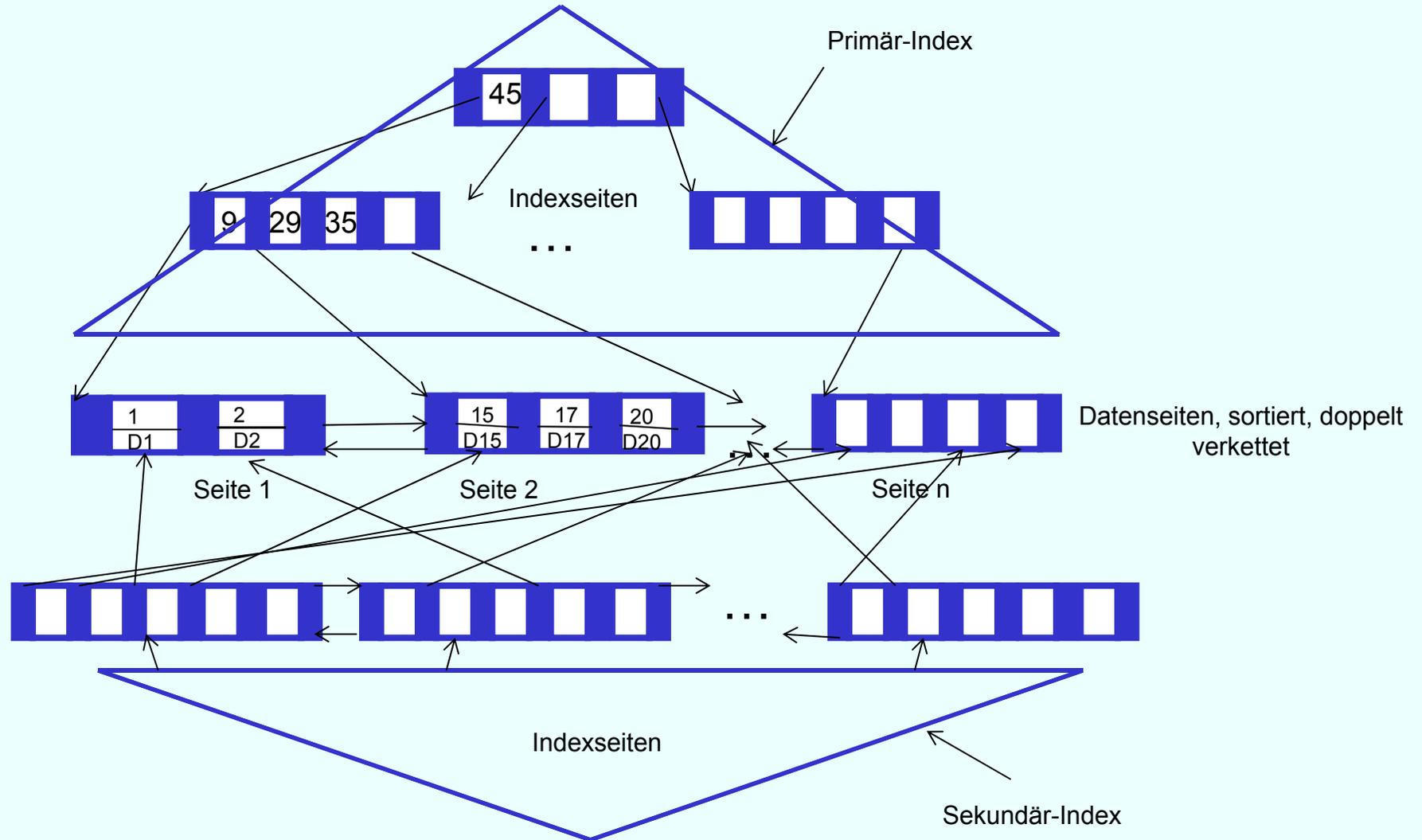
Primärindex - Sekundärindexe

Studenten		
MatrNr	Name	Semester
25403	Jonas	12
29120	Theophrastos	2
29555	Feuerbach	2
27550	Schopenhauer	6
⋮	⋮	⋮

Wann

- Index auf MatrNr?
- Index auf Name?
- Index auf Semester?

Sekundärindexe



DDL: Kreiere Index

```
CREATE [UNIQUE] INDEX index_name  
ON table_name (column_name1 [, column_name2, ...])
```

Beispiel:

```
CREATE INDEX full_name  
ON Person (Name, Vorname)
```

Partitionierung

Was ist Hashing?

- to hash = zerhacken
- Speicherung der Tupel in einem festgelegten Speicherbereich
- Hashfunktion: Abbildung von Tupeln (Schlüsselwerte) in eine festgelegte Menge von Funktionswerten
- optimale Hashfunktion:
 - injektiv (keine gleichen Funktionswerte für unterschiedliche Argumente)
 - surjektiv (kein Speicherverschnitt)
- typische Hashfunktion $h: h(x) = x \bmod N$
Menge von Funktionswerten somit $\{0, \dots, N-1\}$

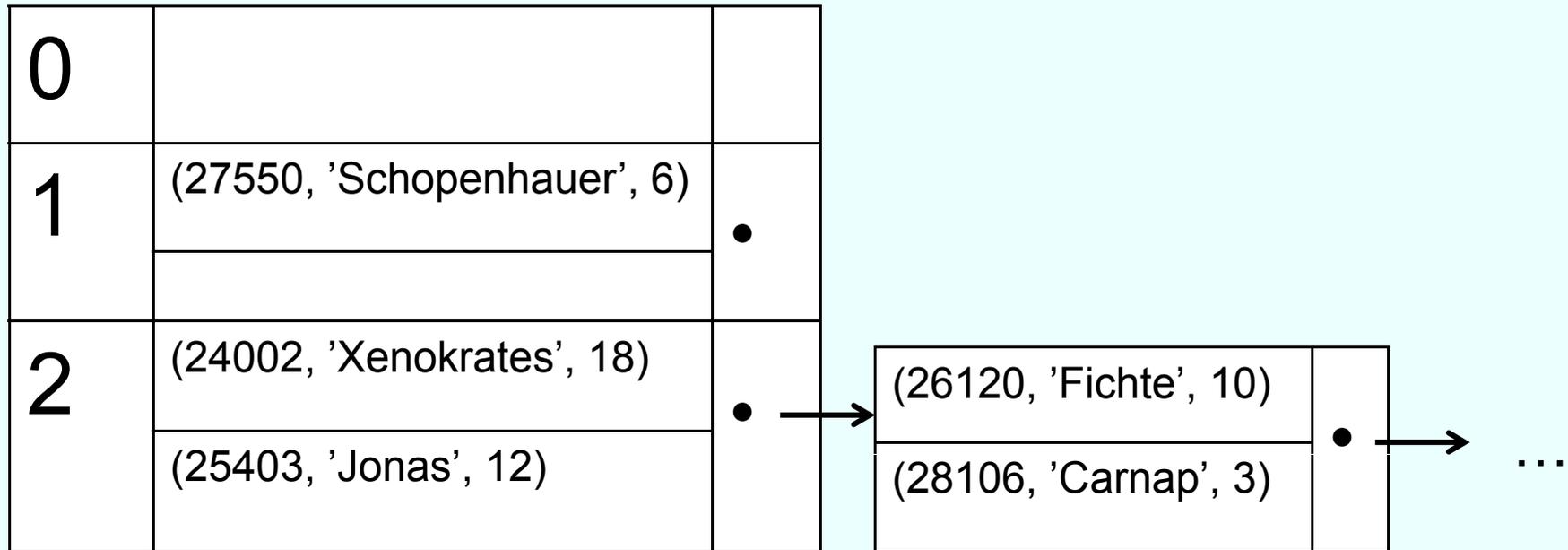
Beispiel Hashing

- Beispiel-Hashfunktion $h(x) = x \bmod 3$

0	
1	(27550, 'Schopenhauer', 6)
2	(24002, 'Xenokrates', 18)
	(25403, 'Jonas', 12)

Kollisionen

- Kollisionsbehandlung



- Ineffizient bei nicht vorhersehbarer Datenmenge
- Ausweg: erweiterbares (dynamisches) Hashing
→ zusätzliche Indirektion über Directory

Vor- / Nachteile Hashing

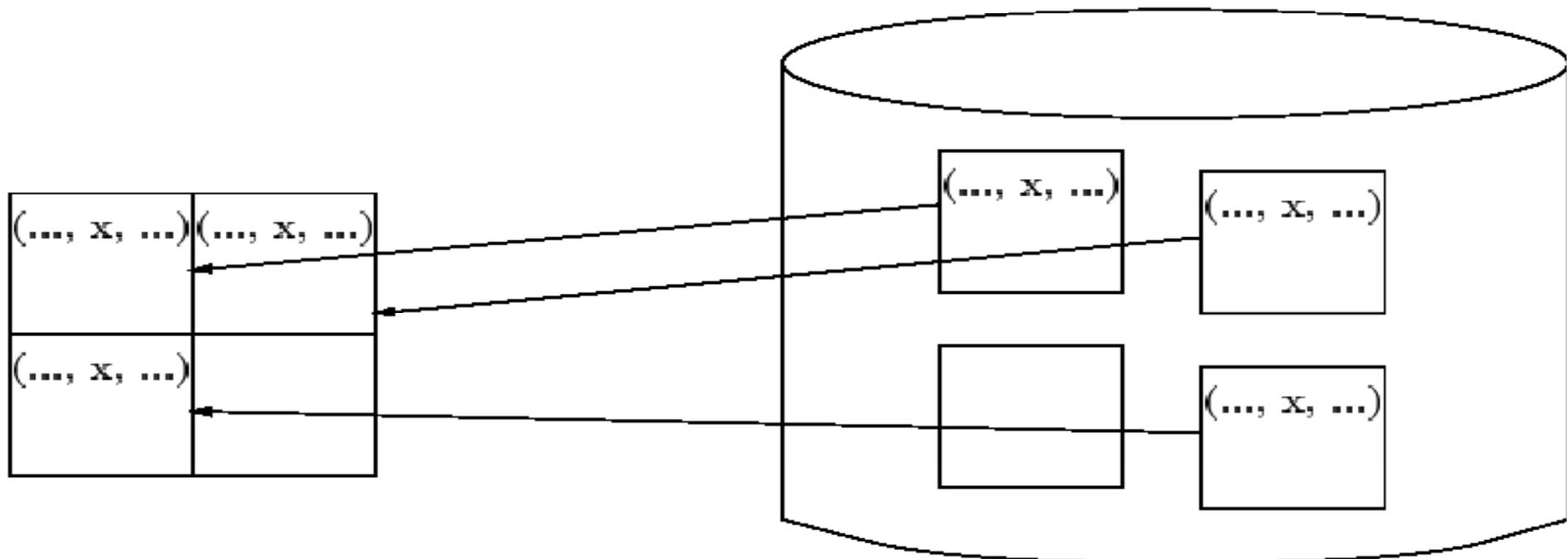
- + wenig Hintergrundspeicherzugriffe
konstanter Aufwand: $O(1)$, i.d.R. 1-2
- + einfache Implementierung

- Kollisionsbehandlung notwendig
- Vorreservierung des Speicherbereichs
- nicht dynamisch bzw. nur mit Nachsteuerung
- **keine Bereichsanfragen, nur Punktanfragen**

Objektballung

Clustering logisch verwandter Daten

```
select *  
from R  
where A = x;
```

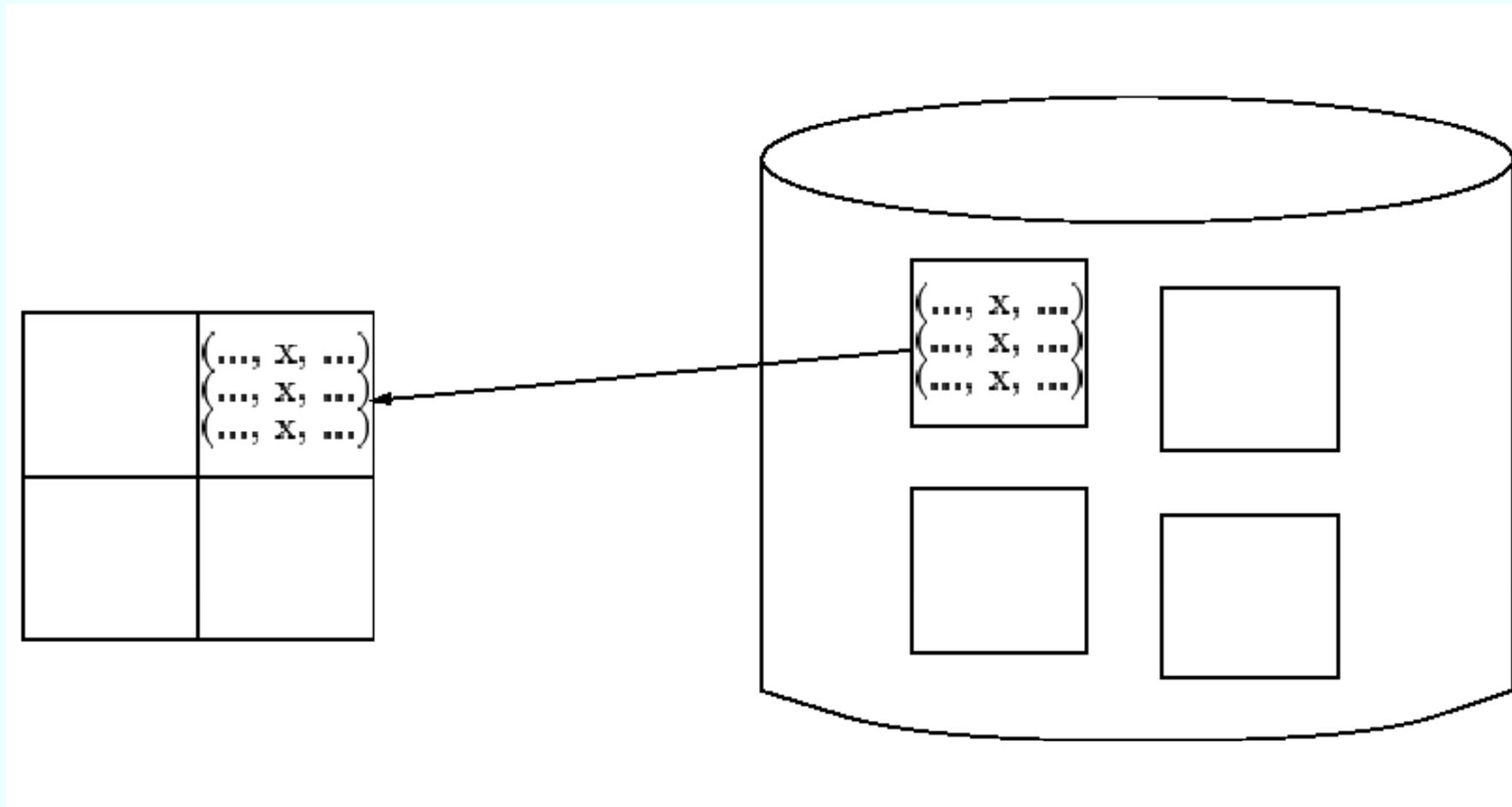


Hauptspeicher ← Zugriffslücke → Hintergrundspeicher

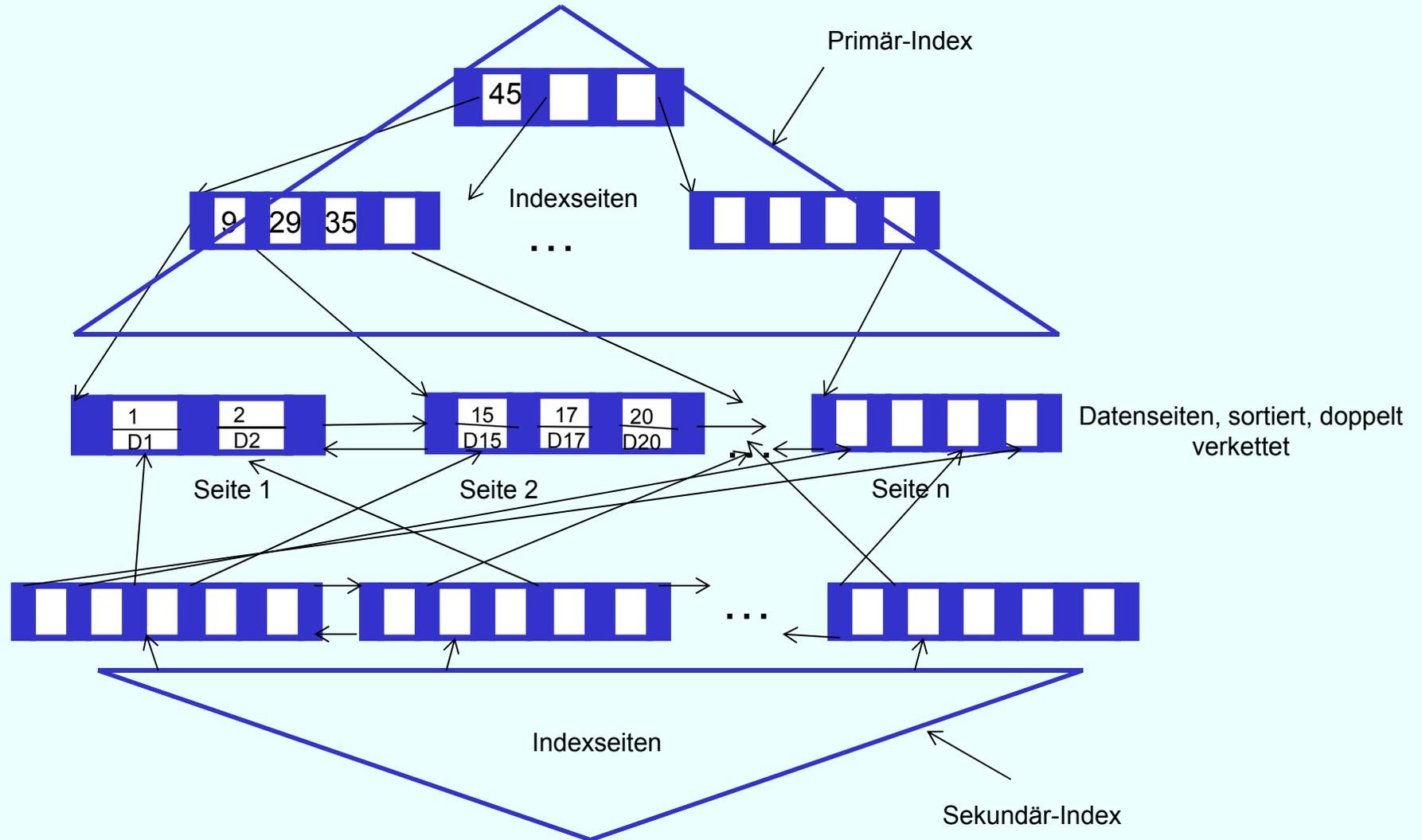
Objektballung

Clustering logisch verwandter Daten

Besser:



Erinnerung



Verschränkte Speicherung

Seite P_i

2125	o Sokrates	o C4	o 226	•
5041	o Ethik	o 4	o 2125	•
5049	o Mäeutik	o 2	o 2125	•
4052	o Logik	o 4	o 2125	•
2126	o Russel	o C4	o 232	•
5043	o Erkenntnistheorie	o 3	o 2126	•
5052	o Wissenschaftstheorie	o 3	o 2126	•
5216	o Bioethik	o 2	o 2126	•

Seite P_{i+1}

2133	o Popper	o C3	o 52	•
5259	o Der Wiener Kreis	o 2	o 2133	•
2134	o Augustinus	o C3	o 309	•
5022	o Glaube und Wissen	o 2	o 2134	•
2137	o Kant	o C4	o 7	•
5001	o Grundzüge	o 4	o 2137	•
4630	o Die 3 Kritiken	o 4	o 2137	•
		o		•