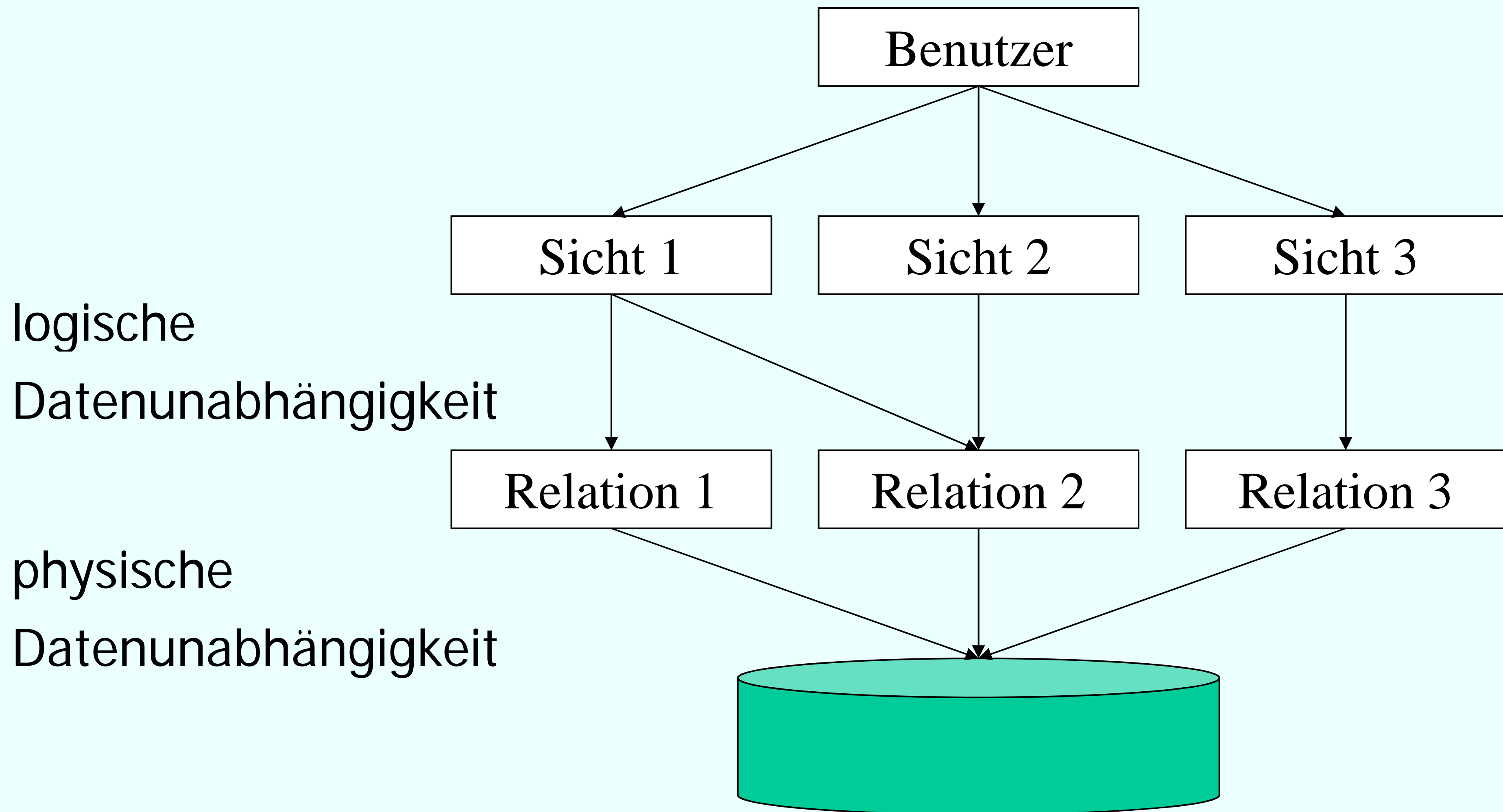
(Sichten, systemgenerierte Werte) Indexe

Wiederholung Sichten:

- Virtuelle Relation
- keine Materialisierung
- Abspeicherung Anfragetext
- Vereinfachung von Anfragen
- für den Datenschutz
- für Statistik
- Untertypen / Obertypen bei Generalisierung

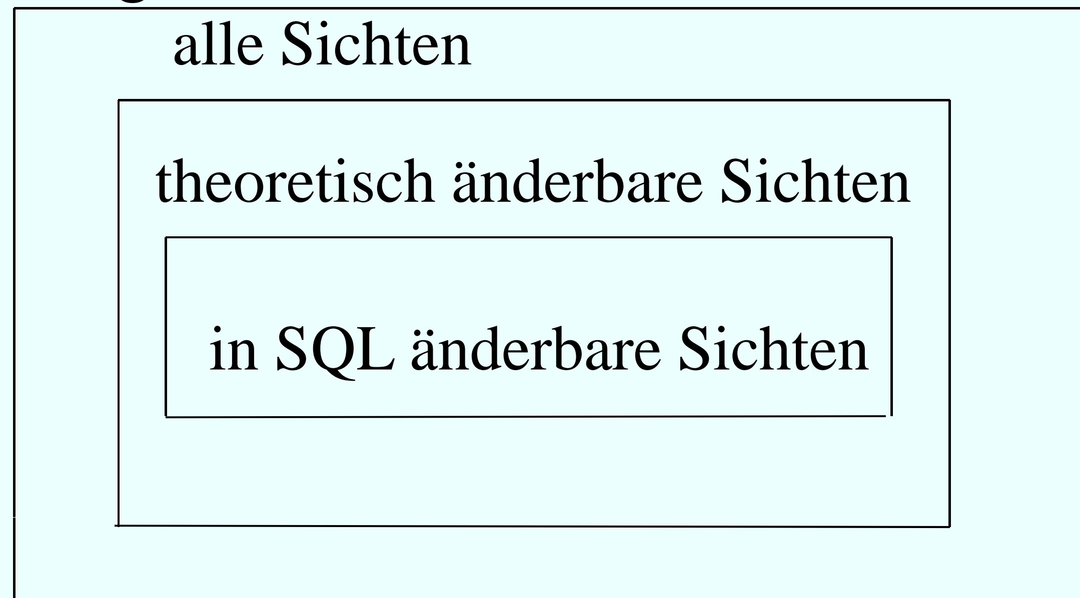
Sichten zur Gewährleistung von Datenunabhängigkeit



Änderbarkeit von Sichten

in SQL

- nur eine Basisrelation
- Schlüssel muss vorhanden sein
- keine Aggregatfunktionen, Gruppierung und Duplikateliminierung



Sichten

Lebensdauer, Gültigkeit

- Löschen: **`DROP VIEW`** ***`view-name`***
- Ungültige (inoperative) Views:
  - Basisrelation wird gelöscht
  - Rechteverlust des View-Erstellers
  - View-Definition bleibt erhalten (ungültig markiert, kann durch Neudefinition reaktiviert werden)
- Auswertung:
  - Ersetzen der Sicht durch ihre Definition (~ Makro)
  - **keine** Speicherung (Materialisierung) der Sichtauswertung

System-generierte Werte

Künstlich erzeugte Werte, Surrogate, ohne Semantik, meist als Schlüssel, z.B. in DB2:

```
create table dept (
   deptno smallint not null
         generated always as
         identity (start with 10,
         increment by 1),
   deptname varchar(50) not null);
```

Einfügen mit:

```
insert into dept values(default, 'I3');
```

Sequenzen

Erzeugt monotone Folge, z.B. in DB2:

```
create sequence SEQNAME
 start with 1
 increment by 1
 no maxvalue
 no cycle
 cache 24;
```

Abrufen:

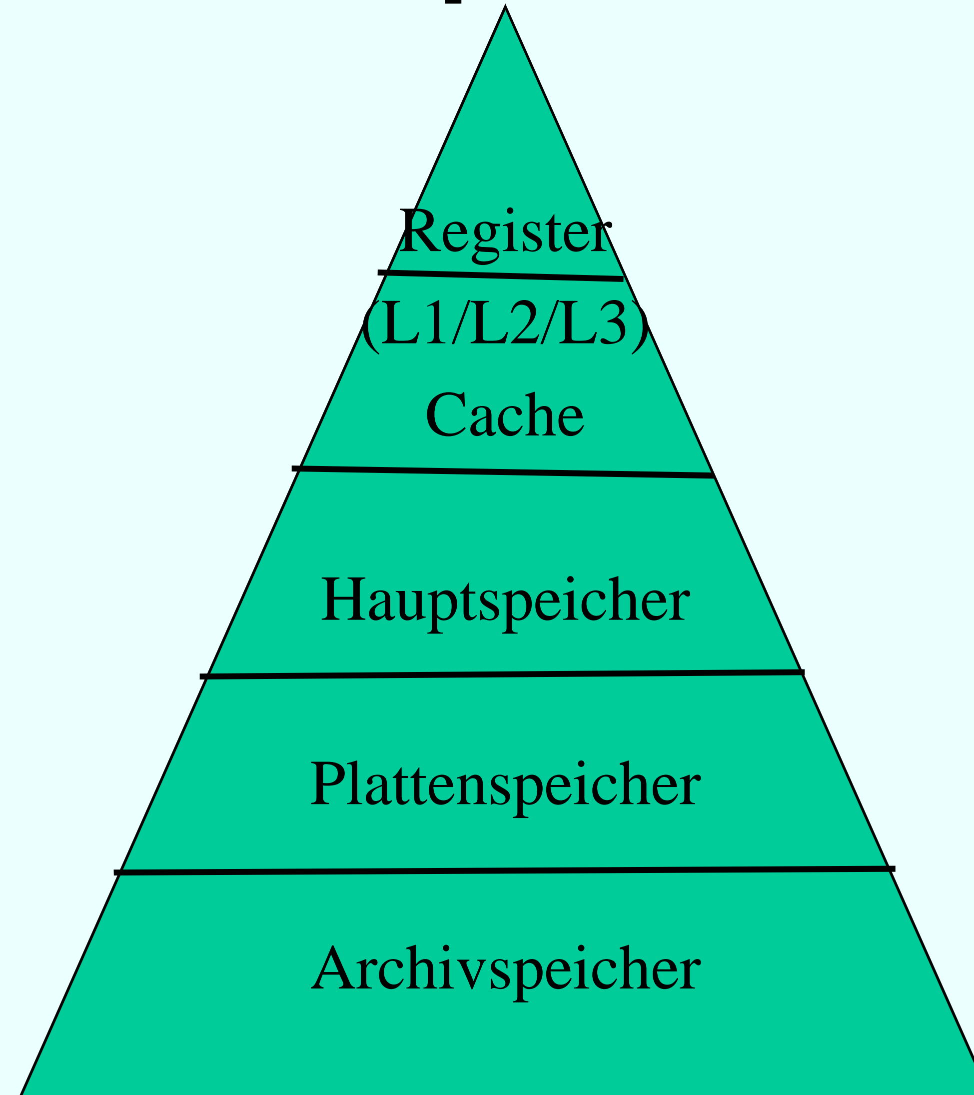
```
insert into dept values (nextval for
SEQNAME, 'I3');
```

Physische Datenorganisation

Themenbereiche:

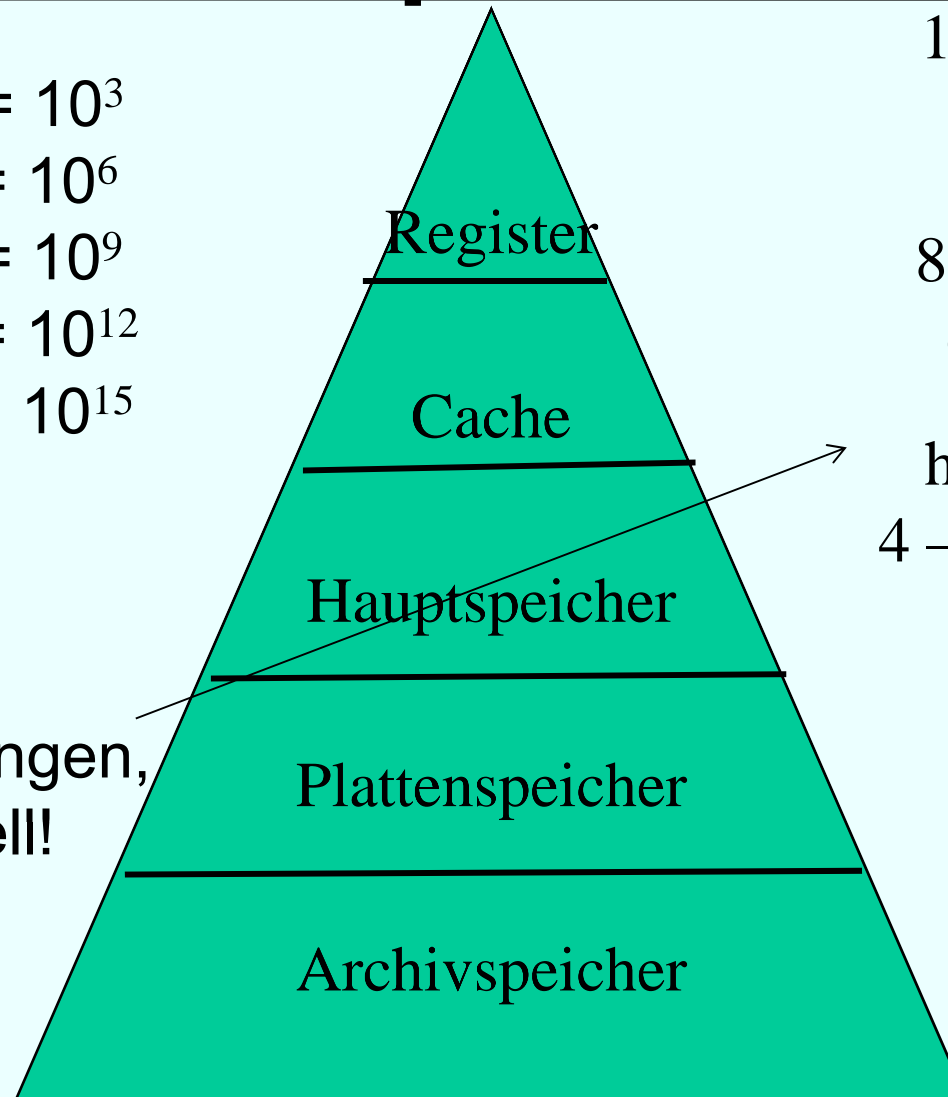
- Speicherhierarchie
- Hintergrundspeicher / RAID
- Speicherstrukturen
- ISAM
- B-Bäume
- Hashing
- Clustering

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

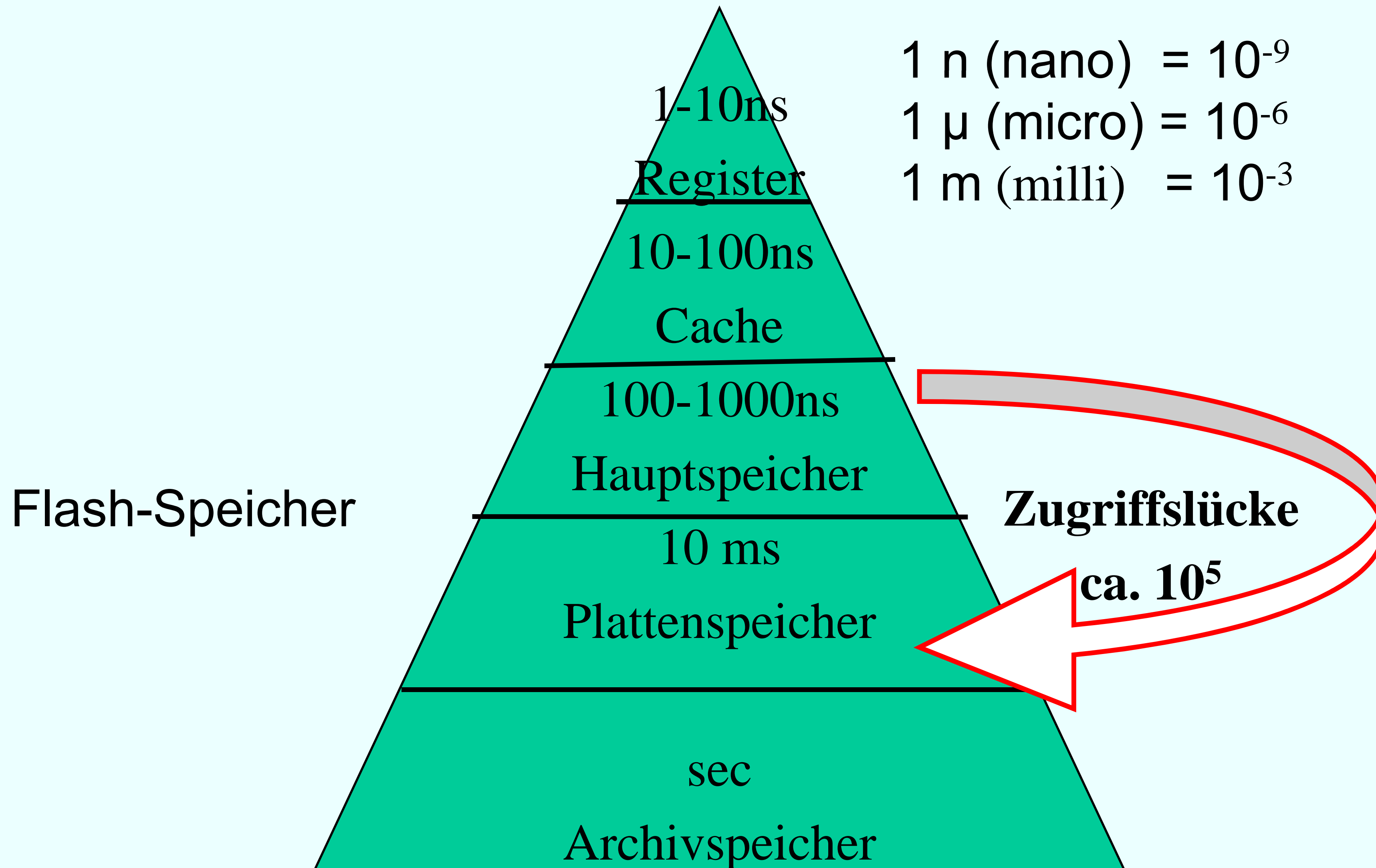
8 – 128 Byte/Cache
Cache-Controller

hoher GB-Bereich,
4 – 64 KB Blockgröße
Betriebssystem

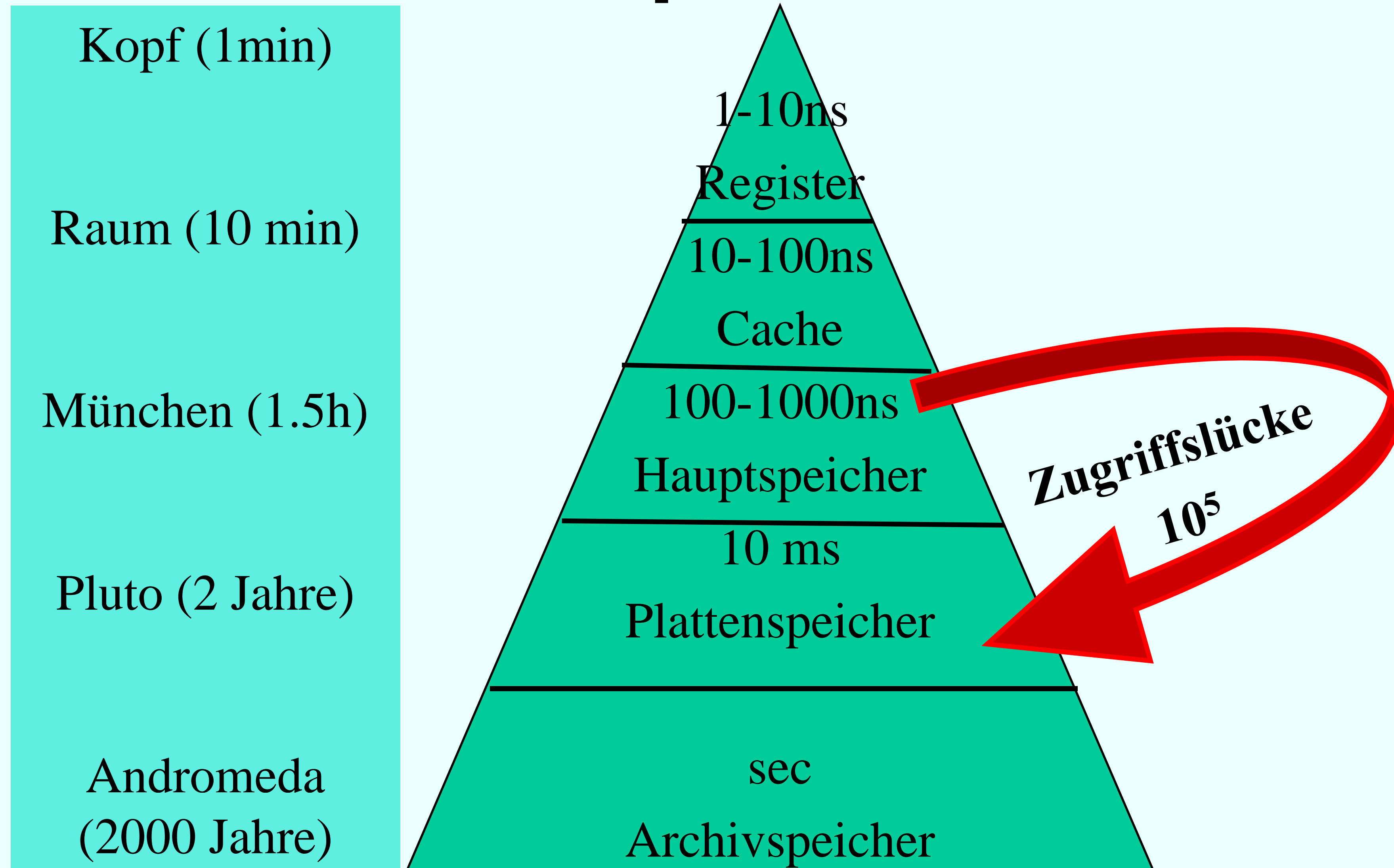
im Bereich TB
Benutzer

im Bereich PB
Benutzer

Überblick: Speicherhierarchie



Überblick: Speicherhierarchie



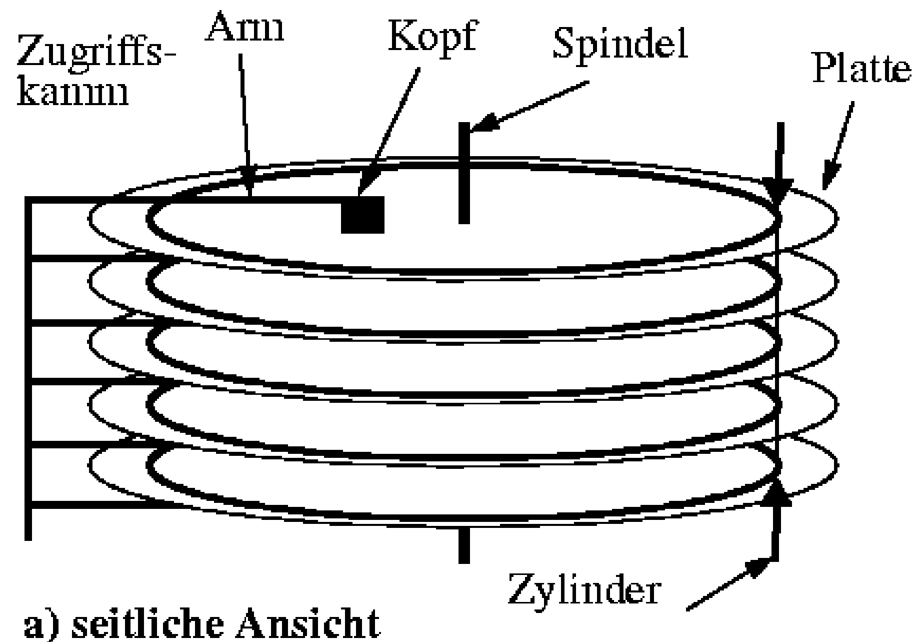
Magnetplattenspeicher

Aufbau

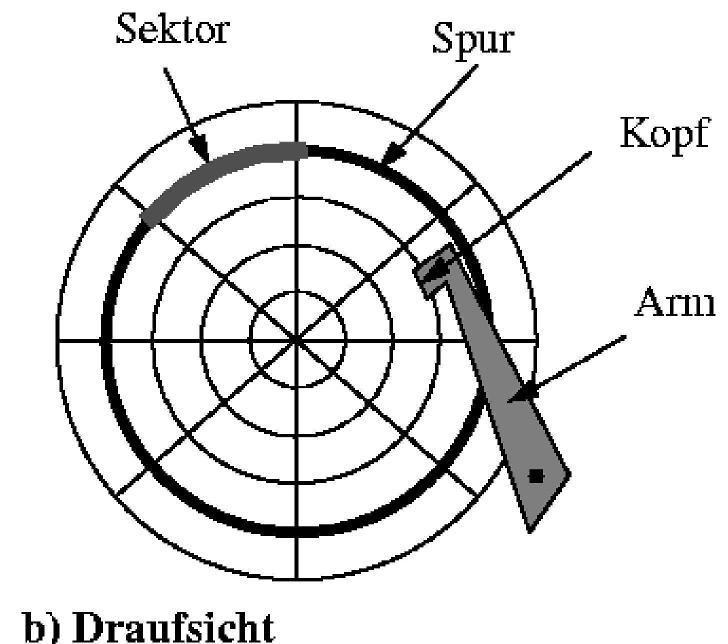
- mehrere gleichförmig rotierende Platten, für jede Plattenoberfläche ein Schreib-/Lesekopf
- jede Plattenoberfläche ist eingeteilt in Spuren
- die Spuren sind formatiert als Sektoren fester Größe (Slots)
- Sektoren (typischerweise 1 - 8 KB) sind die kleinste Schreib-/Leseeinheit auf einer Platte

Adressierung

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



a) seitliche Ansicht



b) Draufsicht

Lesen von Daten von der Platte

Seek Time: Arm/Kopf positionieren

Latenzzeit: Rotation zum Anfang des Blocks
½ Plattenumdrehung (im Durchschnitt)

Lesezeit: Transfer von der Platte zum Hauptspeicher

Random versus Chained IO

Random I/O

- Jedesmal Arm positionieren
- Jedesmal Latenzzeit

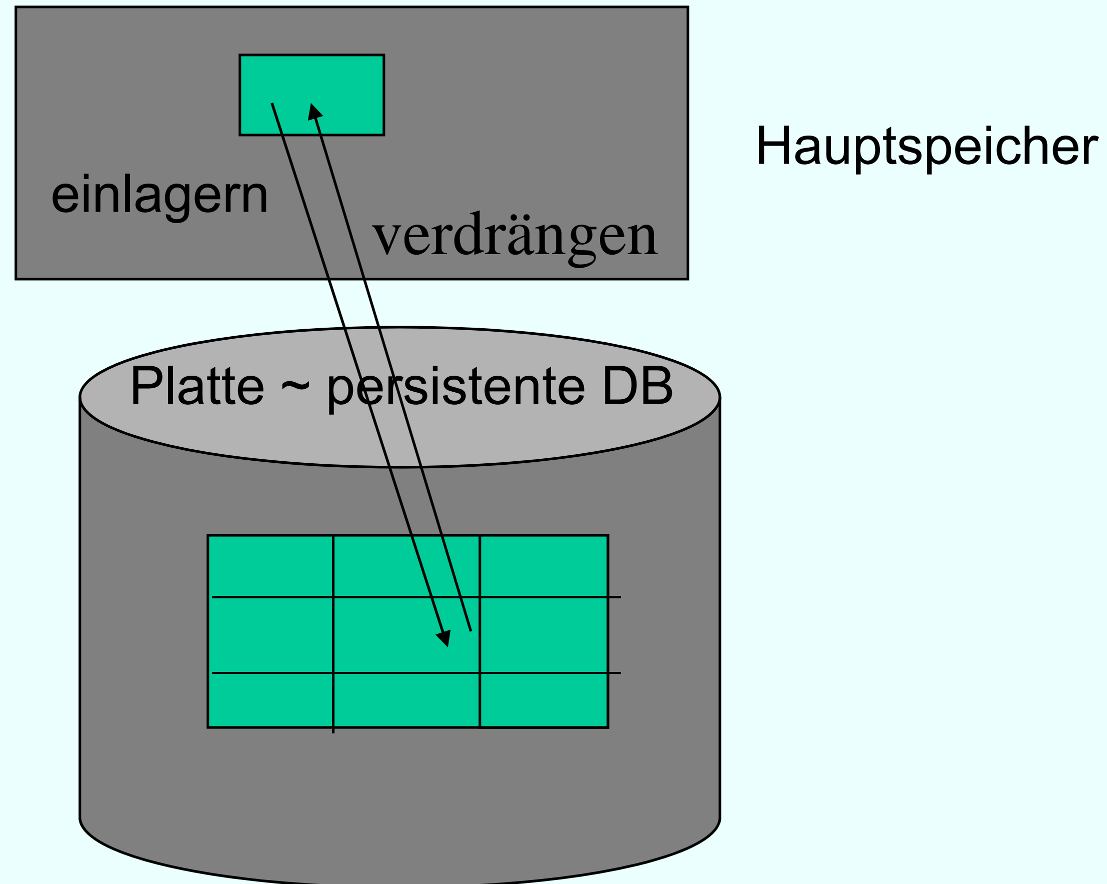
Chained IO

- Einmal positionieren, dann „von der Platte kratzen“

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

→ in Datenbank-Algorithmen beachten !

Systempuffer-Verwaltung



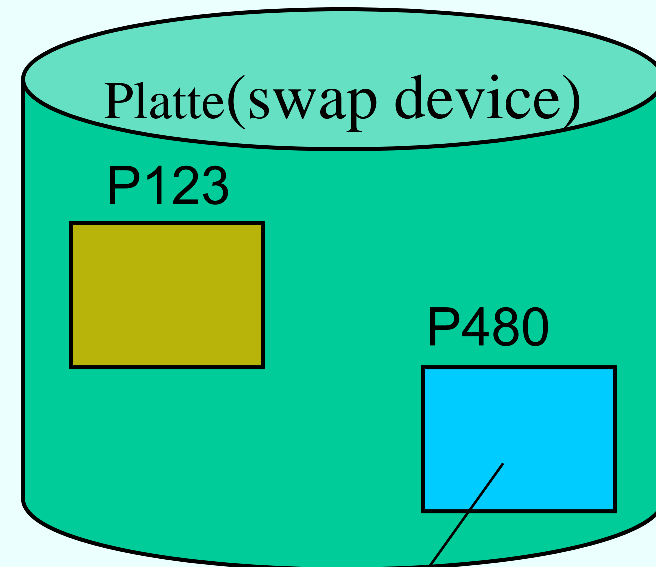
Hauptspeicher

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

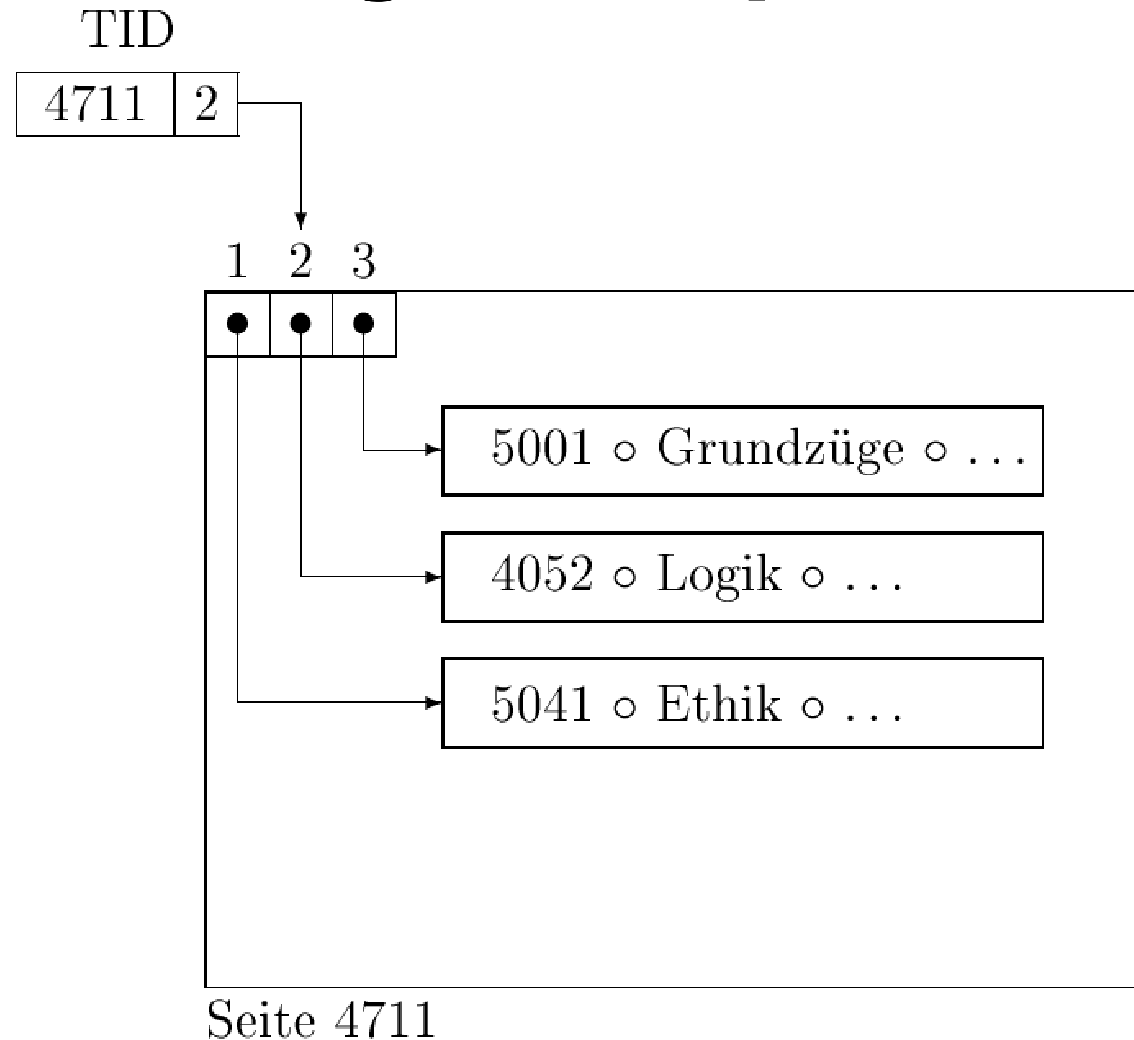
0	4K	8K	12K
16K	20K	24K	28K
32K	36K	40K	44K
48K	52K	56K	60K



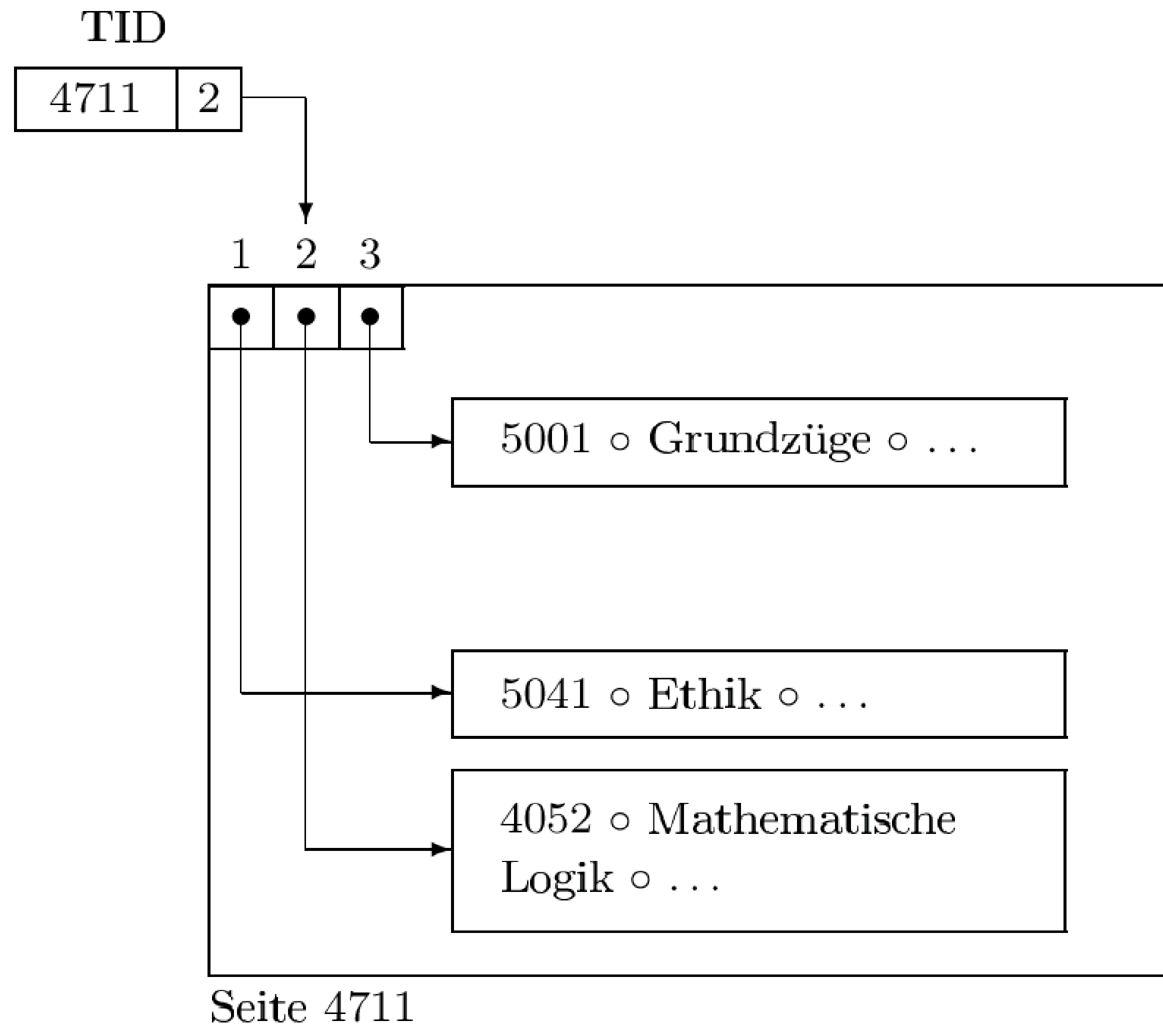
Seitenrahmen

Seite

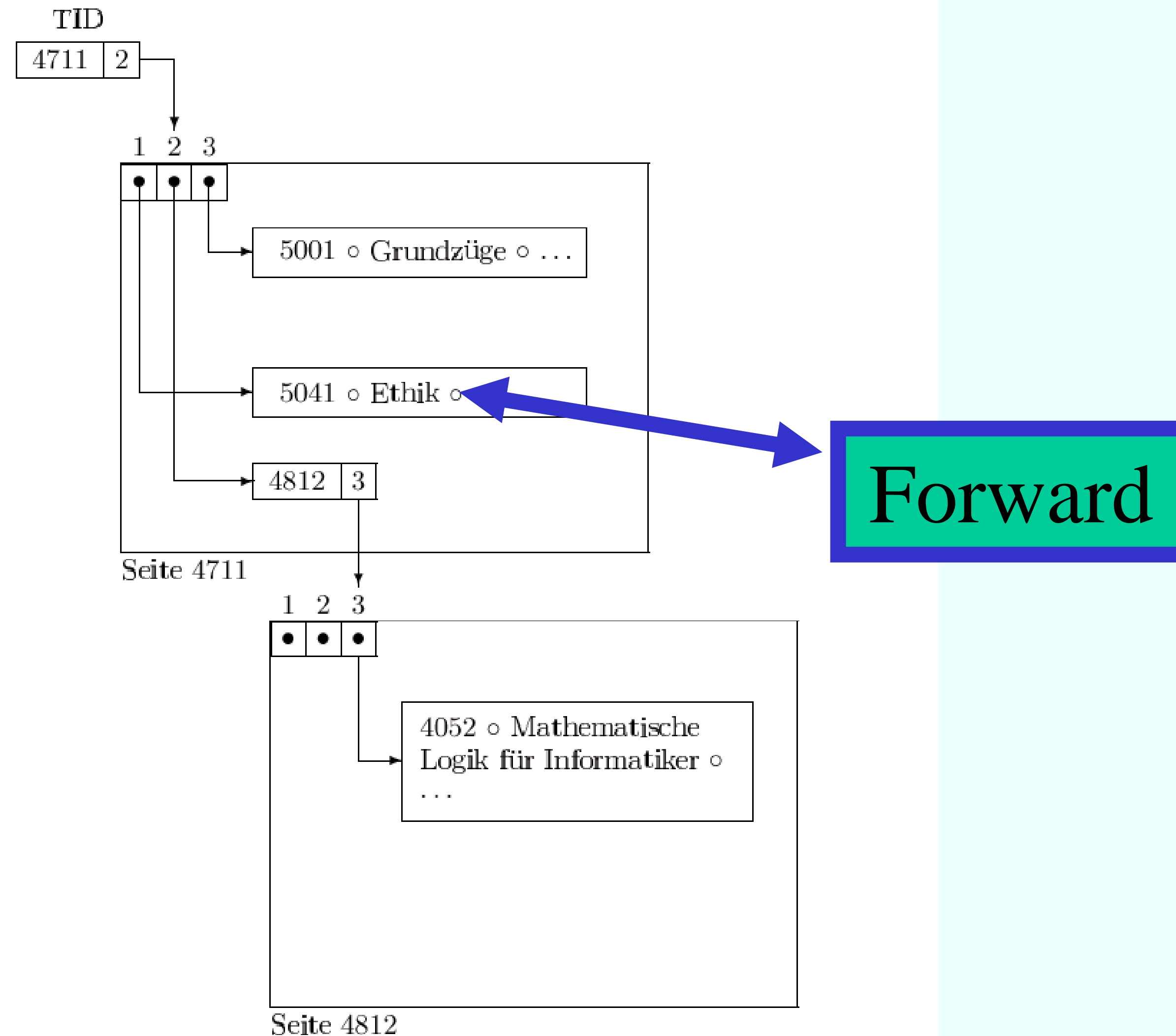
Adressierung von Tupeln auf dem Hintergrundspeicher



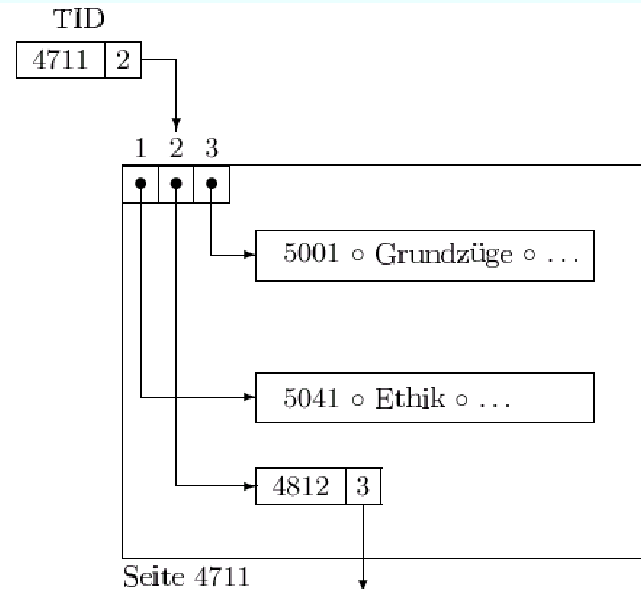
Verschiebung innerhalb einer Seite

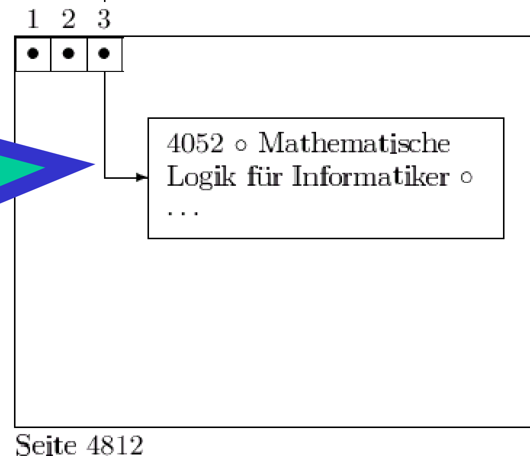


Verschiebung von einer Seite auf eine andere



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)

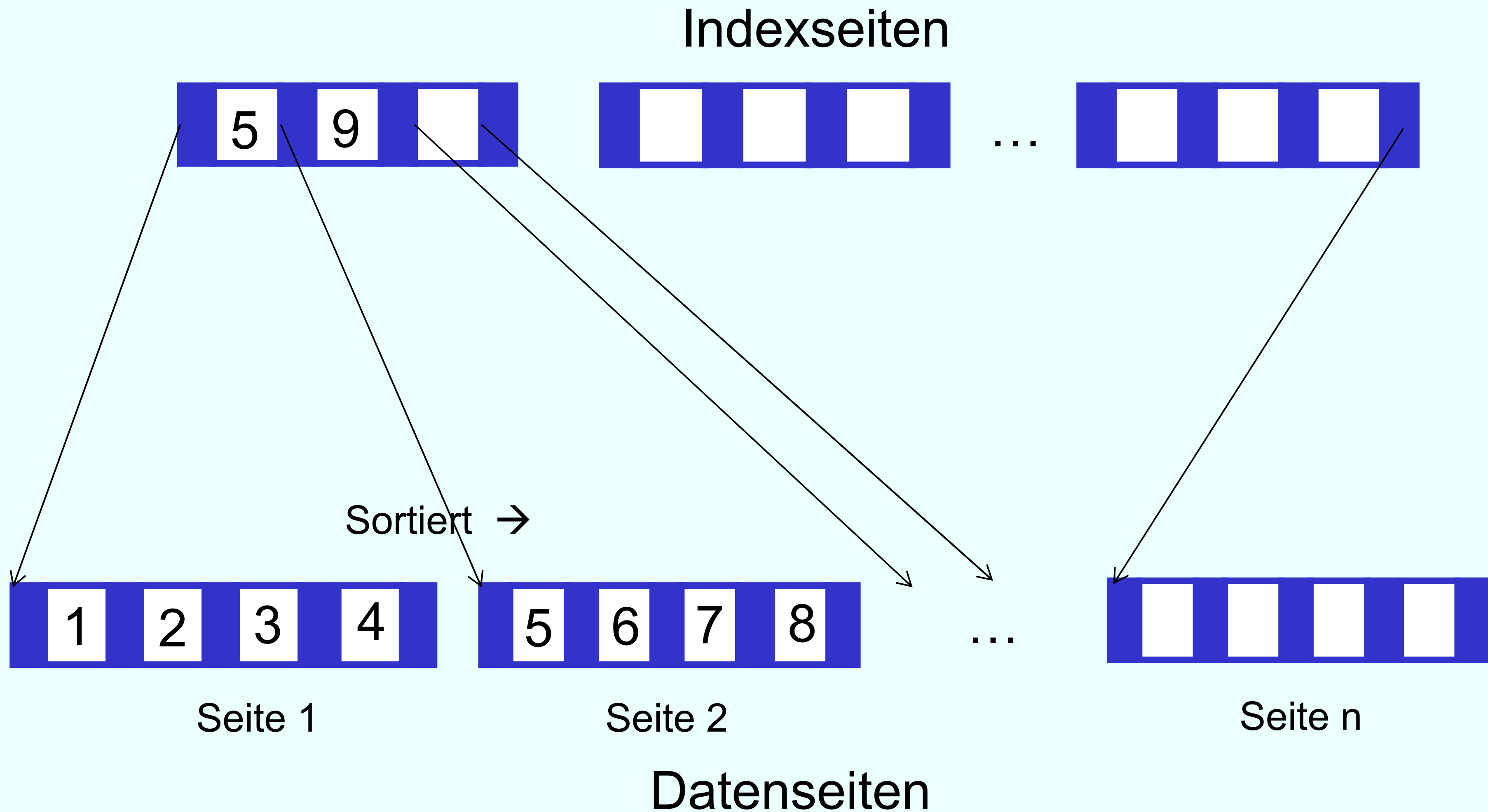
Hierarchische Indexe

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)

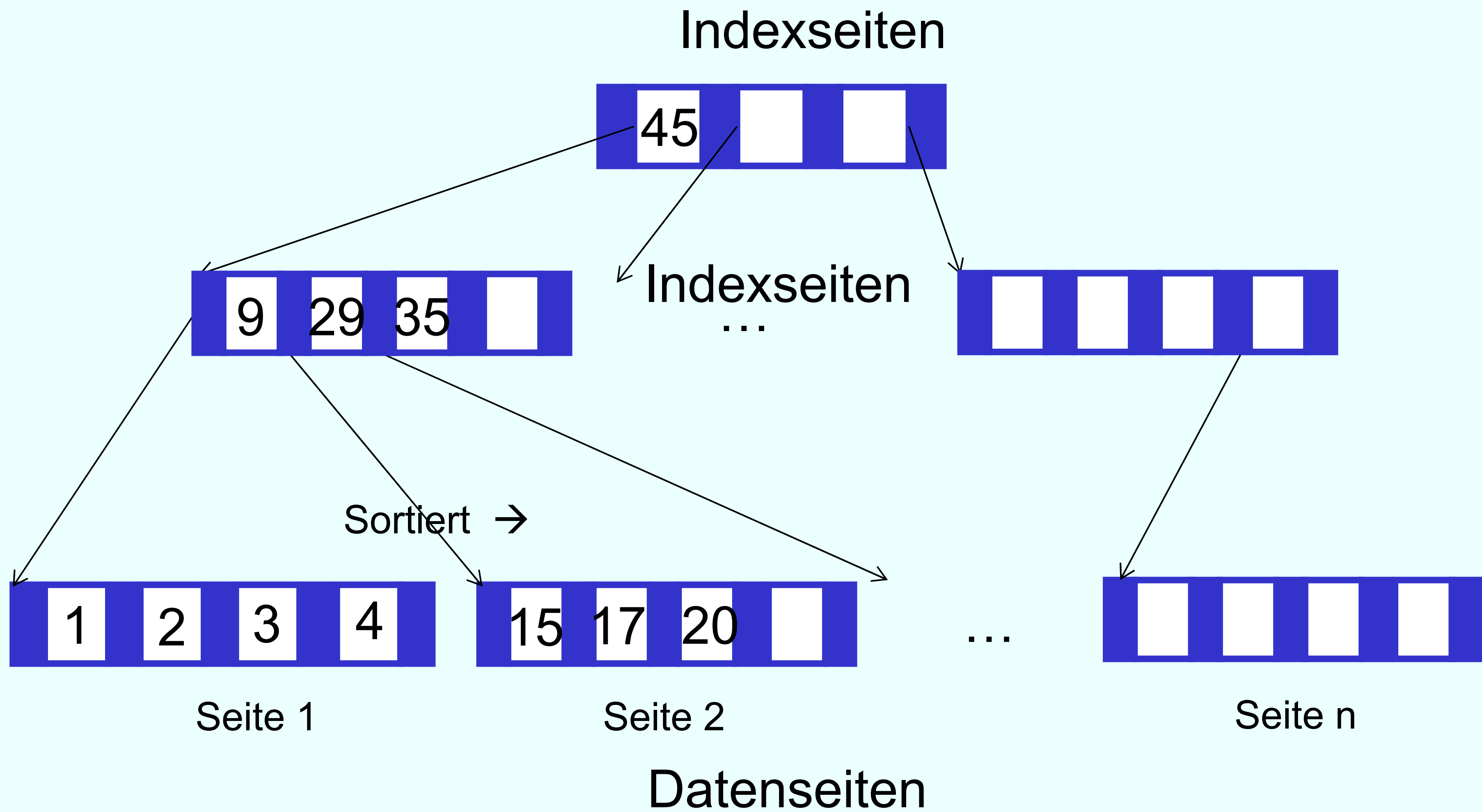
Probleme mit ISAM

- 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-Baum**s

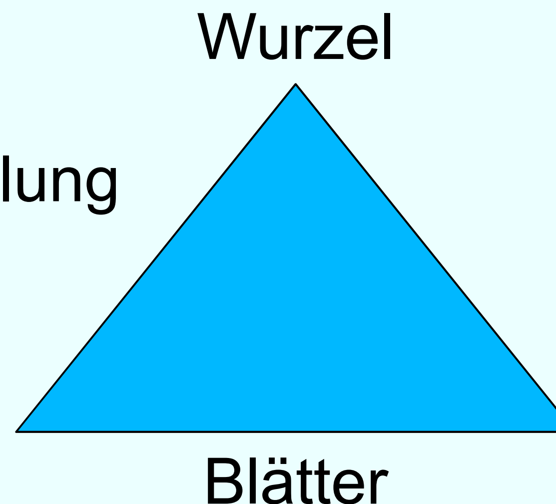
Idee



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 2*i*
- 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 $p0$, $k1$, $p1$, $k2$, . . . kn, pn die Einträge in einem Knoten (pj sind Zeiger, kj Schlüssel)
  Dann gilt folgendes:
  - Der Unterbaum der von $p0$ referenziert wird, enthält nur Schüssel kleiner als $k1$
  - pj zeigt auf einen Unterbaum mit Schlüsseln zwischen kj und $kj+1$
  - Der Unterbaum der von pn referenziert wird, enthält nur Schlüssel größer als kn

Einfügealgorithmus

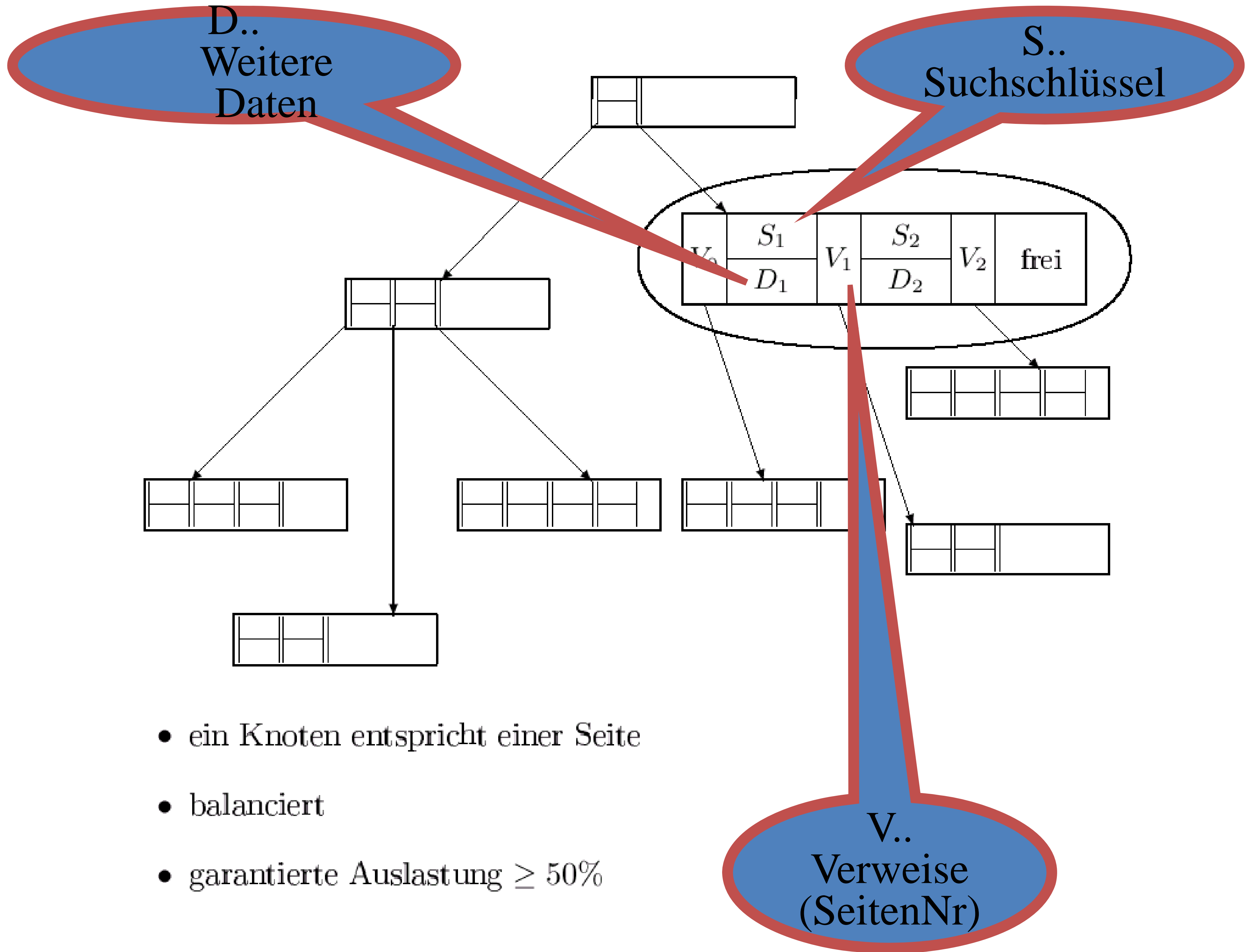
1. Finde den richtigen Blattknoten, um den neuen Schlüssel einzufügen
2. Füge Schlüssel dort ein
3. Falls kein Platz mehr da
   i. Teile Knoten und ziehe Median heraus
   ii. Füge alle Knoten kleiner als Median in linken Knoten, alle größer als Median in rechten Knoten
   iii. Füge Median in Elternknoten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

Löschalgorithmus

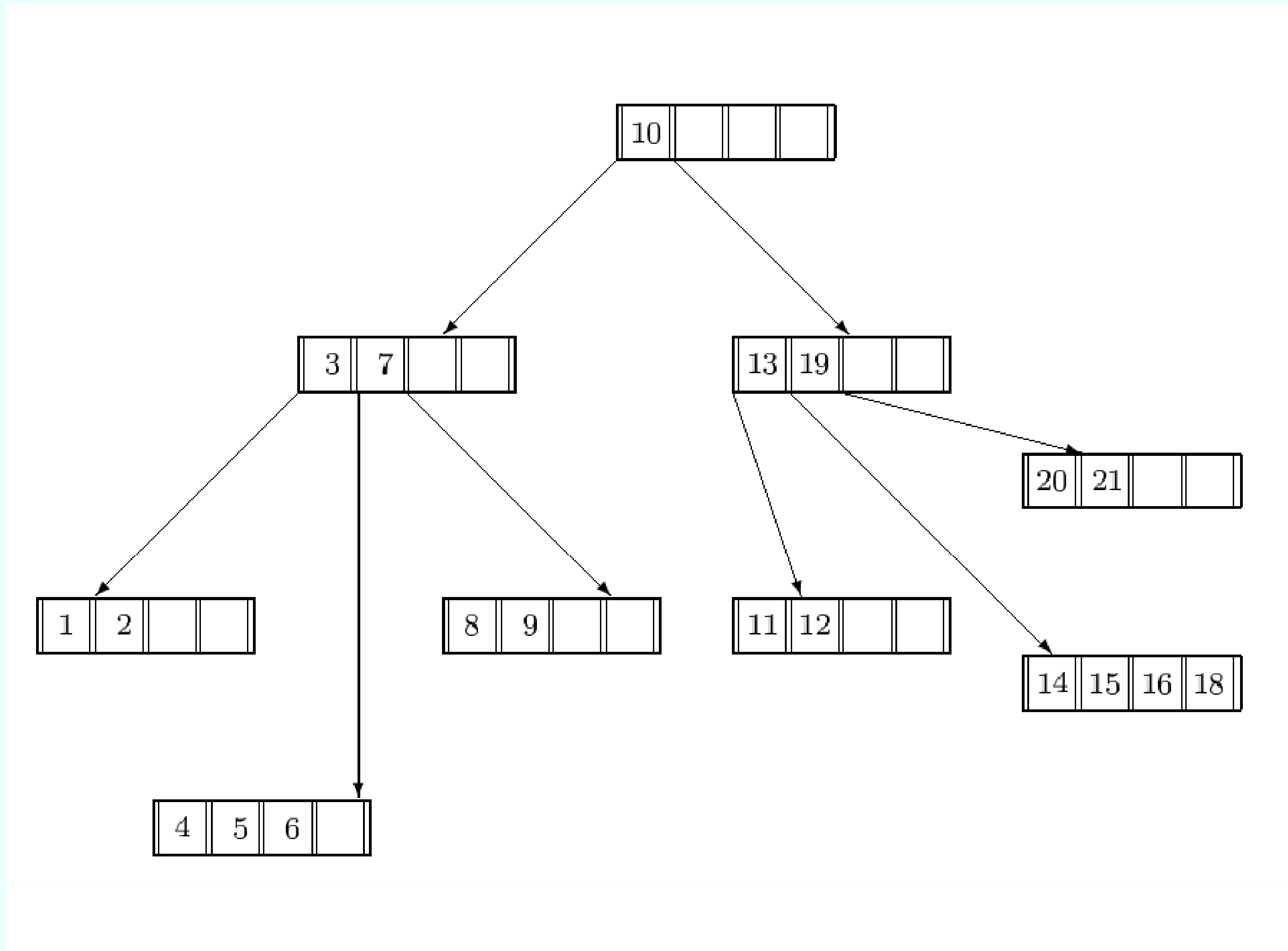
- In einem Blattknoten kann ein Schlüssel einfach gelöscht werden
- In einem inneren Knoten muss Verbindung zu den Kindern bestehen bleiben
  - Deshalb wird der nächstgrößere (oder nächstkleinere) Schlüssel gesucht (in entsprechendem Kindknoten)
  - Dieser Schlüssel wird an die Stelle des gelöschten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


D..
Weitere
Daten
S..
Suchschlüssel
S_1
S_2
V_1
V_2
frei
D_1
D_2
V..
Verweise
(SeitenNr)
• ein Knoten entspricht einer Seite
• balanciert
• garantierte Auslastung ≥ 50%

Beispielbaum



Sukzessiver Aufbau eines B-Baums vom Grad k=2

Siehe
http://www-db.in.tum.de/research/publications/books/DBMSeinf/EIS, Folie 49 – 123

Im Internet gibt es eine Reihe von Animationsprogrammen für B-Bäume – **ohne Gewähr!**

z.B. http://slady.net/java/bt/ sieht ganz gut aus oder auch http://people.ksp.sk/~kuko/bak/ (auch andere Suchbäume)