Datenbanksysteme SS 2011

Anfang von Kapitel 4:
Physikalische Datenorganisation

Oliver Vornberger

Institut für Informatik
Universität Osnabrück

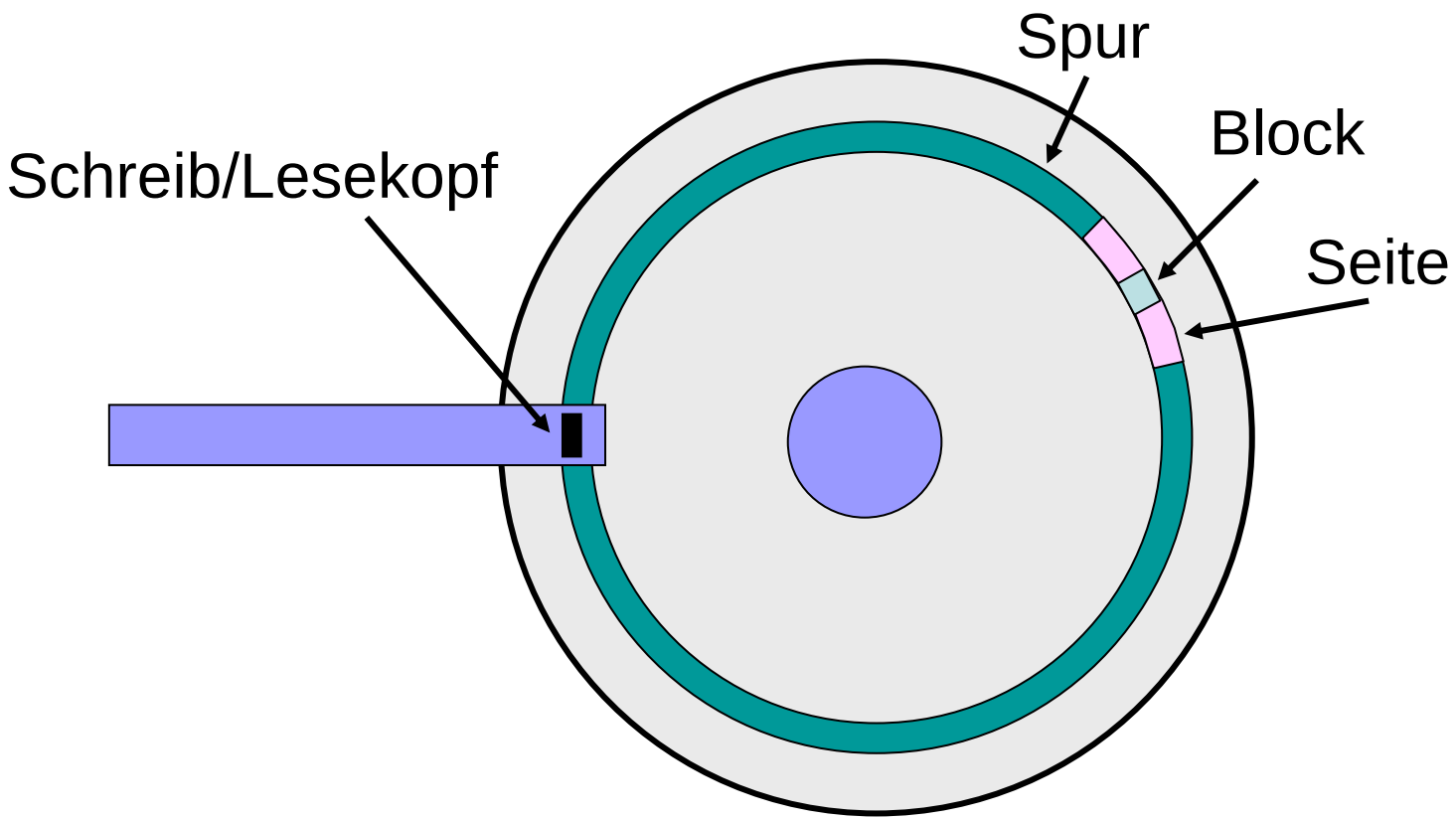
Vorlesung vom 19.04.2011

Speicherhierarchie

GB	TB
10 €	100 €
GHertz	10 ms

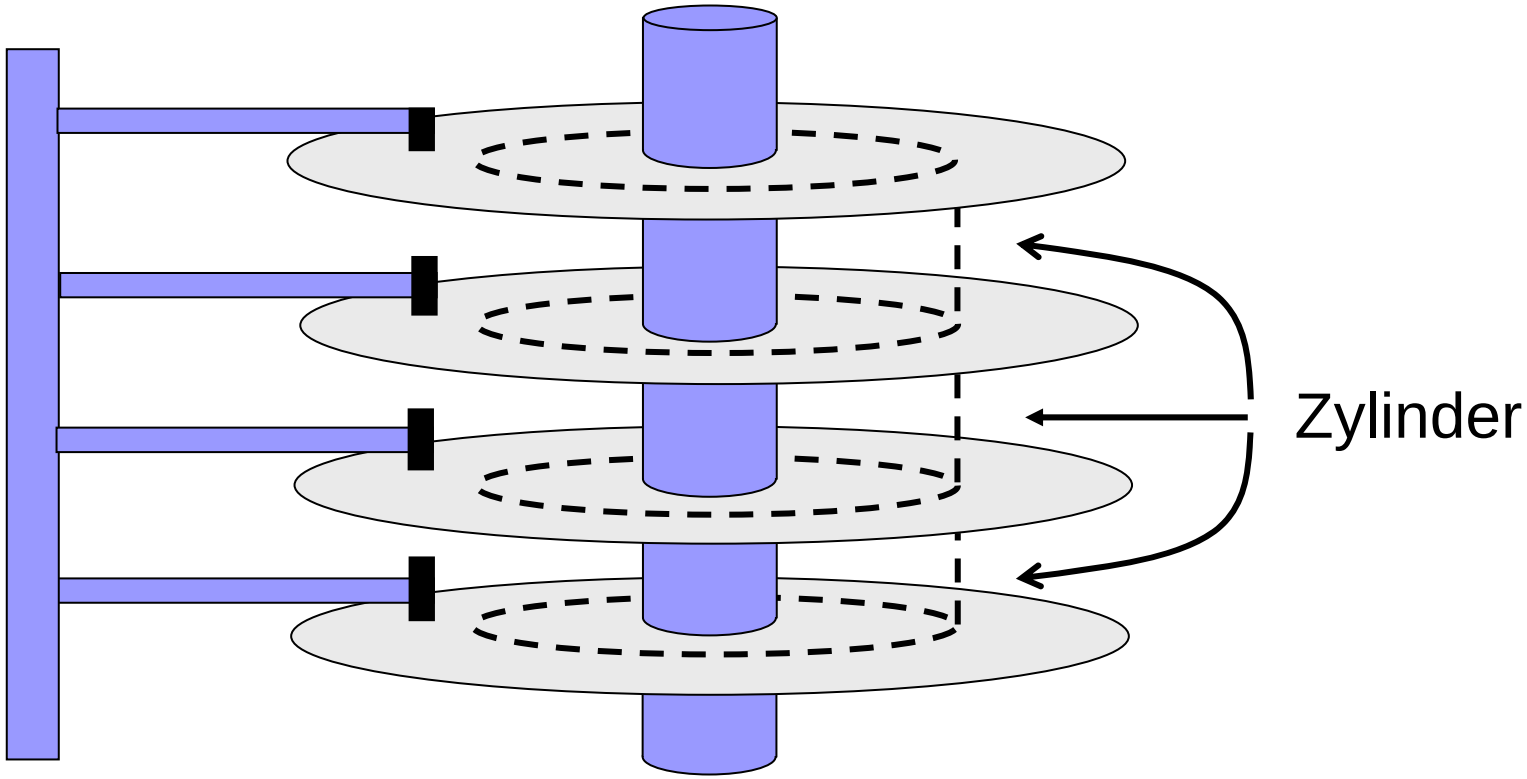
	Primär	Sekundär	Tertiär
Größe	klein	groß [10^{3}]	sehr groß
Tempo	schnell	langsam [10^{-7}]	sehr langsam
Preis	teuer	billig [10^{-2}]	billig
Granularität	fein	grob	grob
Stabilität	flüchtig	stabil	stabil

Festplatte: von oben



Zugriff = Positionieren + Warten + Lesen

Festplatte: seitlich



Physikalische Datenorganisation

Record: Datensatz fester oder variabler Länge
mit Feldern bestimmten Typs

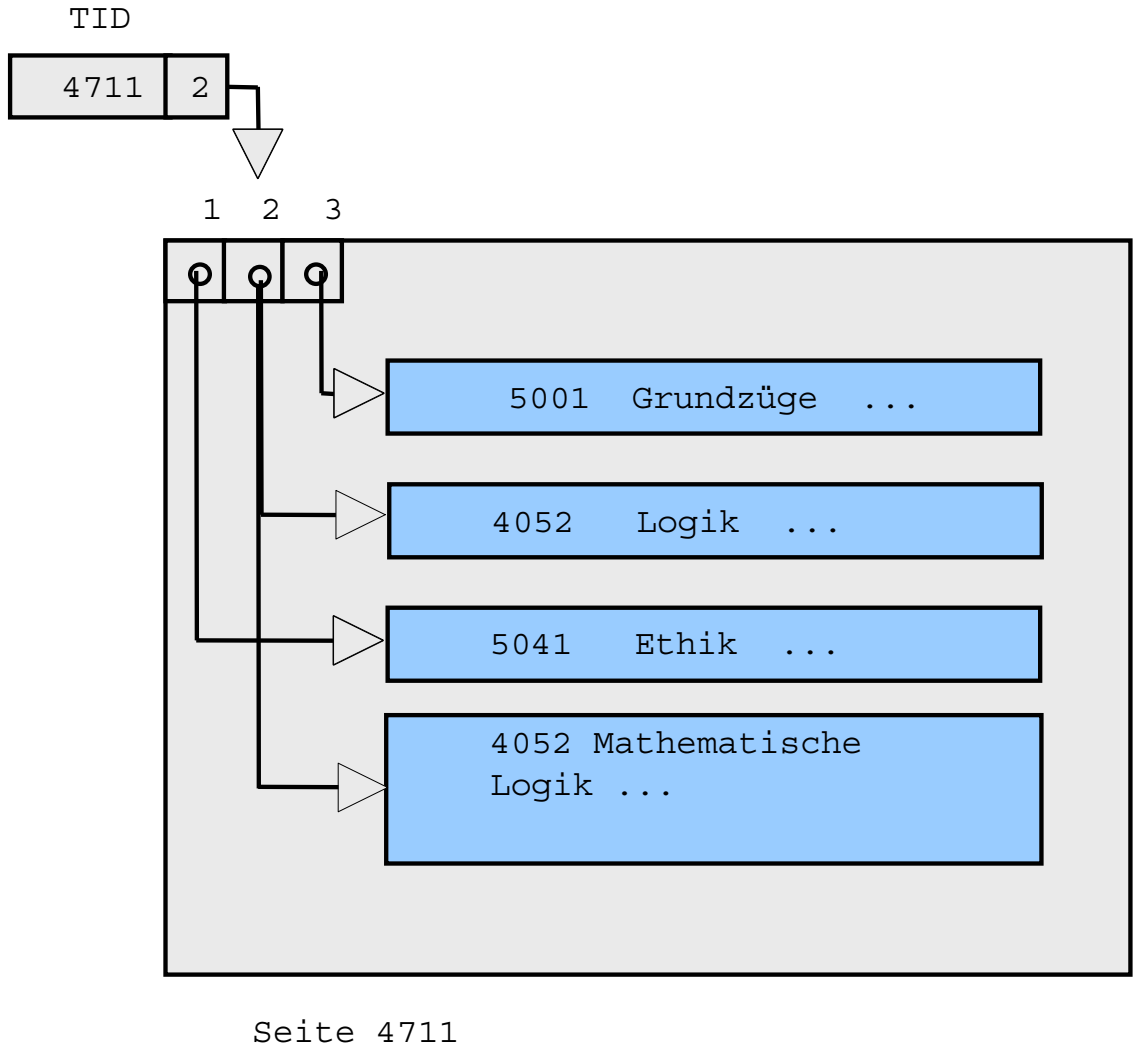
Block: Speichereinheit im Hintergrundspeicher
(2^9 - 2^{12} Bytes)

File: Menge von Blöcken

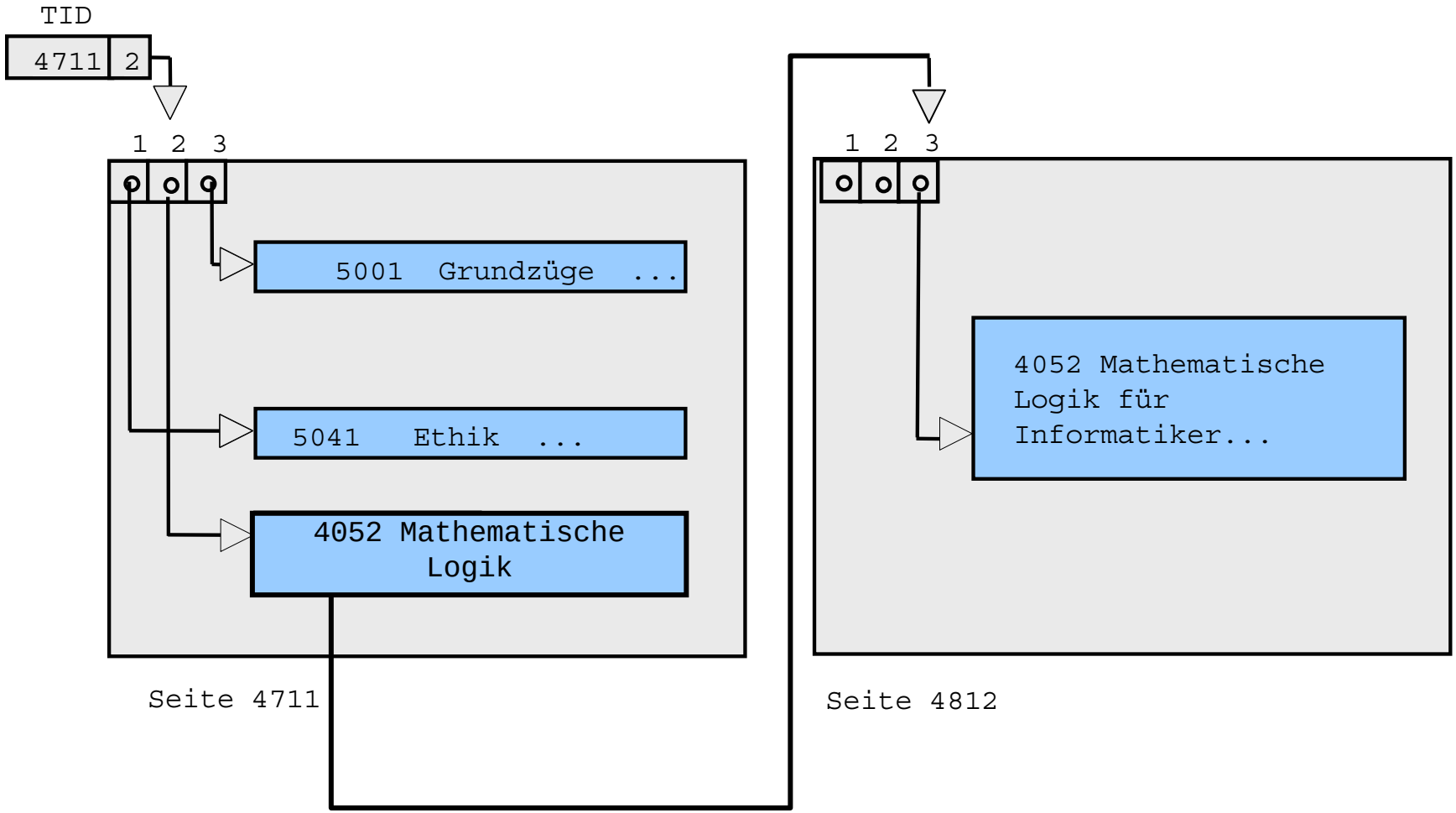
Pinned record: Blockadresse + Offset

Unpinned record: Blockadresse + Recordschlüssel
Blockadresse + Tupelidentifikator

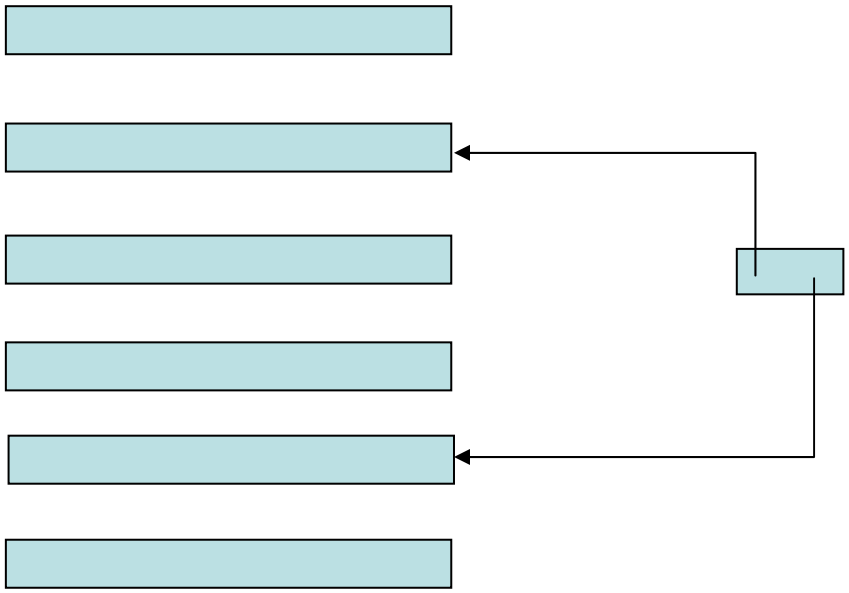
Tupelidentifikator: Verschieben innerhalb der Seite



Tupelidentifikator: Verdrängen auf andere Seite



Implementierung des E-R-Modells



- pro Entity ein Record mit den Attributen als Datenfelder
- pro Relationship ein Record mit den TIDs der beteiligten Entities

Speicher-Operationen

- INSERT: Einfügen eines Records
- LOOKUP: Suchen eines Records
- MODIFY: Modifizieren eines Records
- DELETE: Löschen eines Records

Heap-File

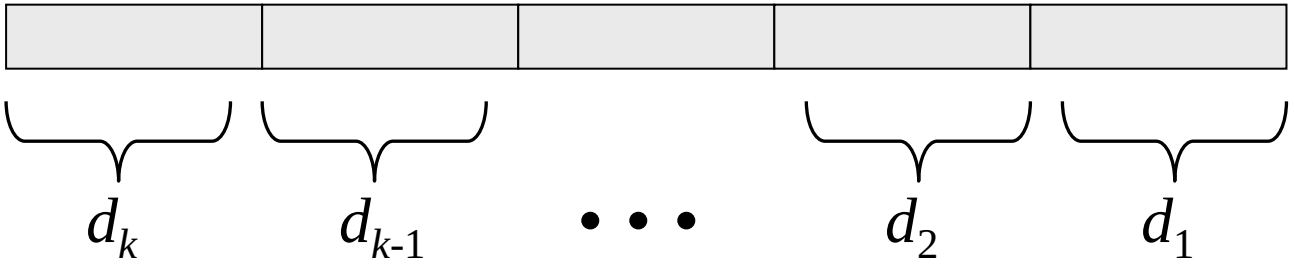
- INSERT: Record am Ende einfügen
- LOOKUP: Gesamtes File durchsuchen
- MODIFY: Record überschreiben
- DELETE: Lösch-Bit setzen

Hashing

- alle Records sind auf Buckets verteilt
- ein Bucket = verzeigerte Liste von Blöcken
- Bucketdirectory enthält Einstiegsadressen
- Hashfunktion liefert zuständiges Bucket
- Wertebereich: **`[0 .. N-1]`**
- Pro Datenrecord ein Frei/Belegt-Bit

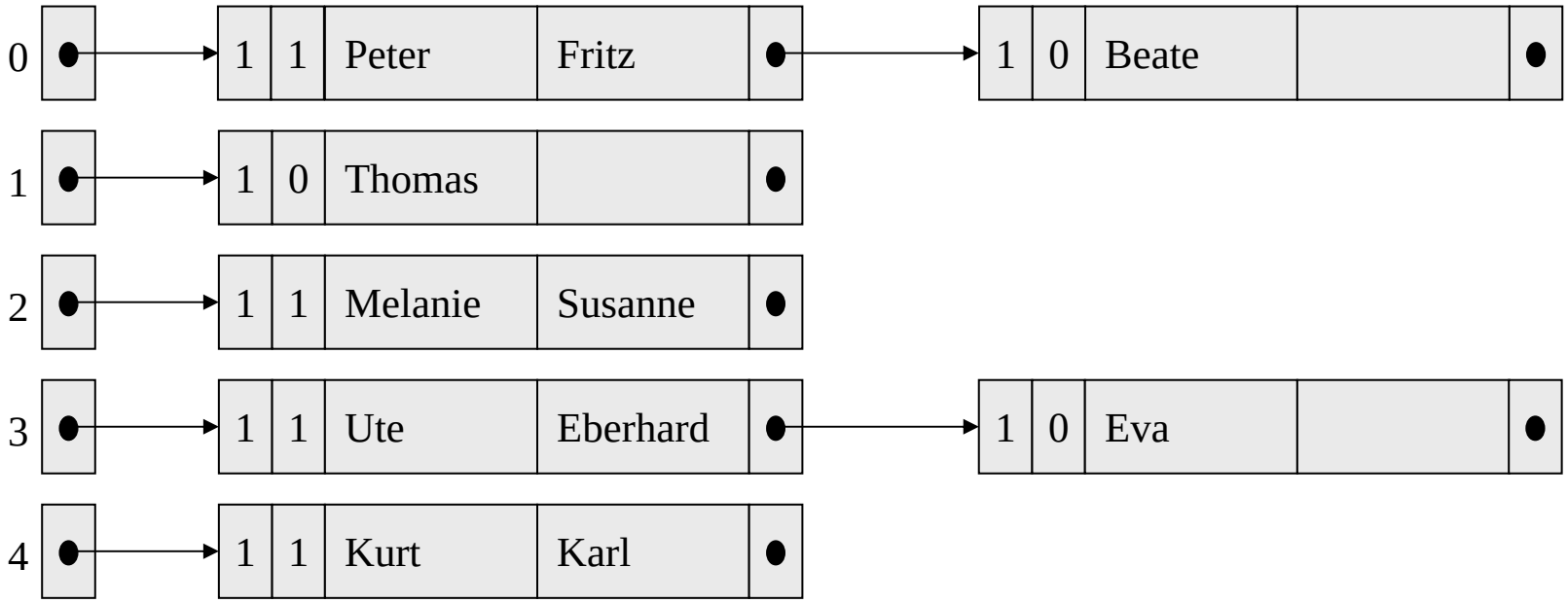
Beispiel für Hash-Funktion

Zerlege Schlüssel *v* in *k* Gruppen zu je *n* Bits.
Fasse jede Gruppe als Zahl auf.



$$h(v) = (d_k + d_{k-1} + \ldots + d_2 + d_1) \bmod N$$

Beispiel für Hashorganisation (|v| mod 5)



Hash-Operationen für Schlüssel *v*

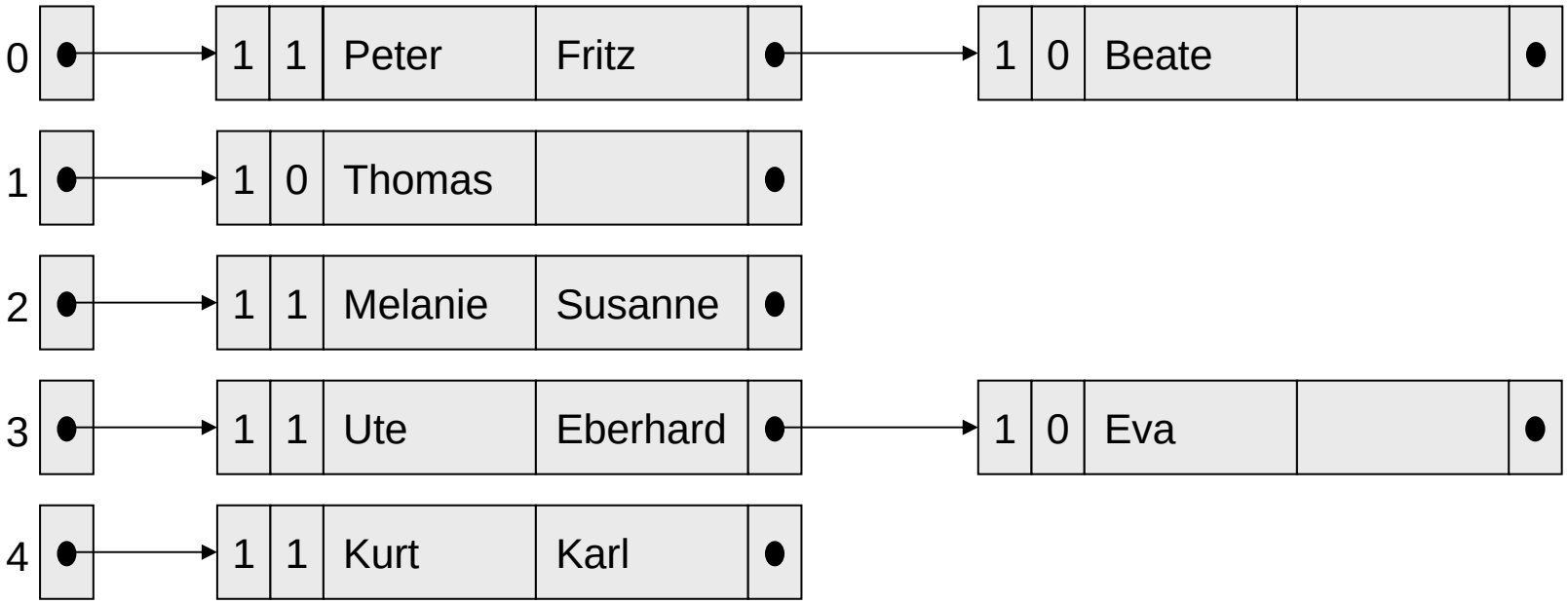
- **LOOKUP:**
  Berechne $h(v) = i$. Lies zuständigen Directory-Block, durchsuche alle Blöcke im Bucket

- **DELETE:**
  LOOKUP, falls nicht gefunden: Fehler.
  Sonst: Löschbit setzen.

- **INSERT:**
  Zunächst LOOKUP. Falls Satz mit *v* vorhanden: Fehler.
  Sonst: Freien Platz im Block überschreiben oder neuen Block anfordern.

- **MODIFY:**
  Falls Schlüssel beteiligt: DELETE und INSERT
  Andernfalls: LOOKUP und überschreiben.

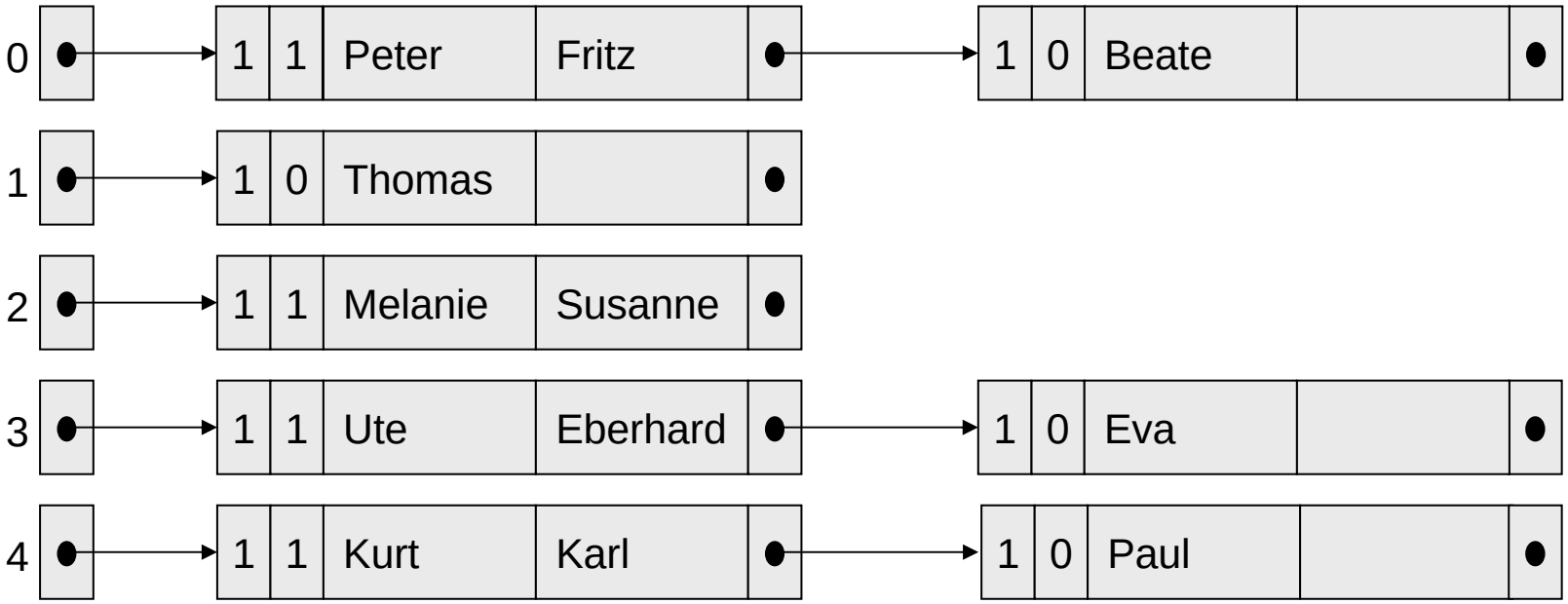
Beispiel für Hashorganisation



Hashorganisation: Ausgangslage $h(s) = |s| \bmod 5$

Paul einfügen

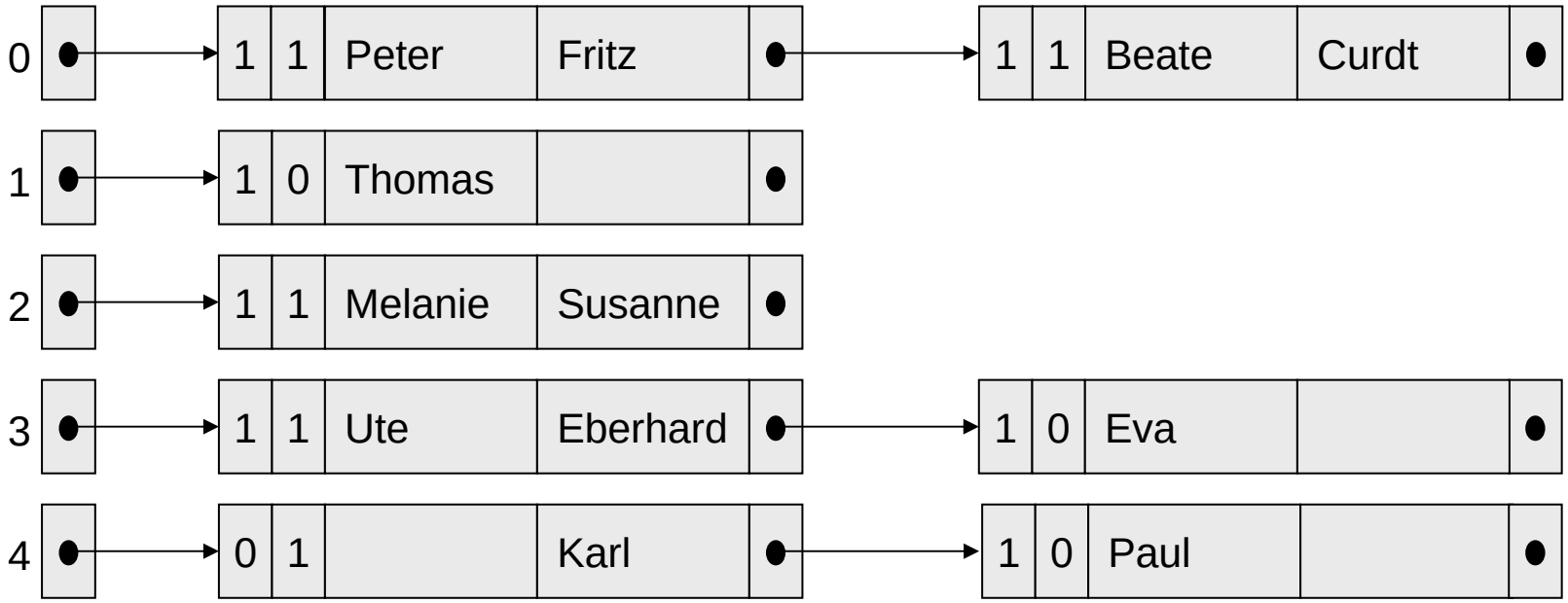
Beispiel für Hashorganisation



Hashorganisation: nach Einfügen von Paul

Kurt umbenennen nach Curdt

Beispiel für Hashorganisation

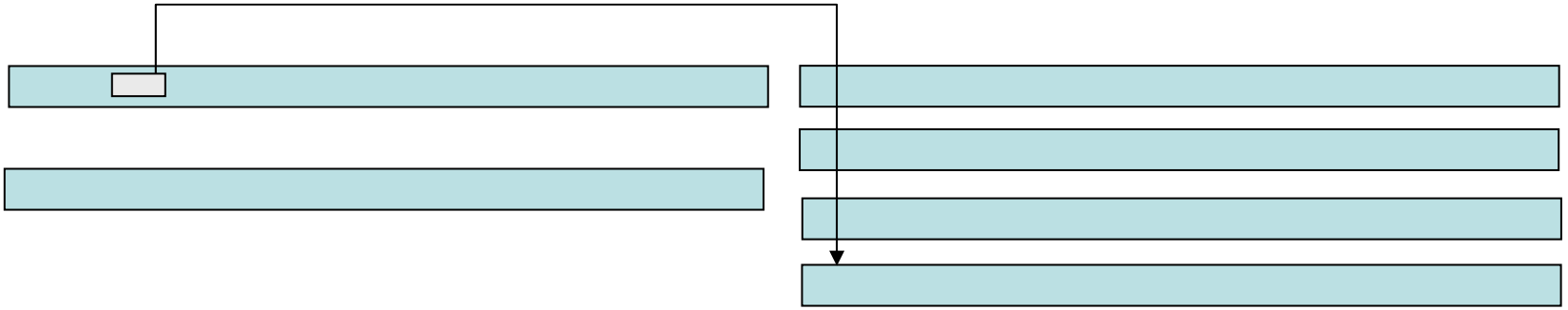


Hashorganisation: nach Umbenennen von Kurt in Curdt

Probleme beim Hashing

- Keine Sortierung
- Keine Bereichsabfragen
- Blocklisten werden immer länger
- Reorganisation erforderlich

ISAM (Index sequential access method)



- Index-Datei mit Verweisen in die Hauptdatei.
- Index-Datei enthält Tupel
  < Schlüssel, Blockadresse>
  sortiert nach Schlüsseln.
- Liegt **<*v*,*a*>** in der Index-Datei, so sind alle Record-Schlüssel im Block, auf den *a* zeigt, größer oder gleich *v*.

ISAM-Operationen für Record mit Schlüssel *v*

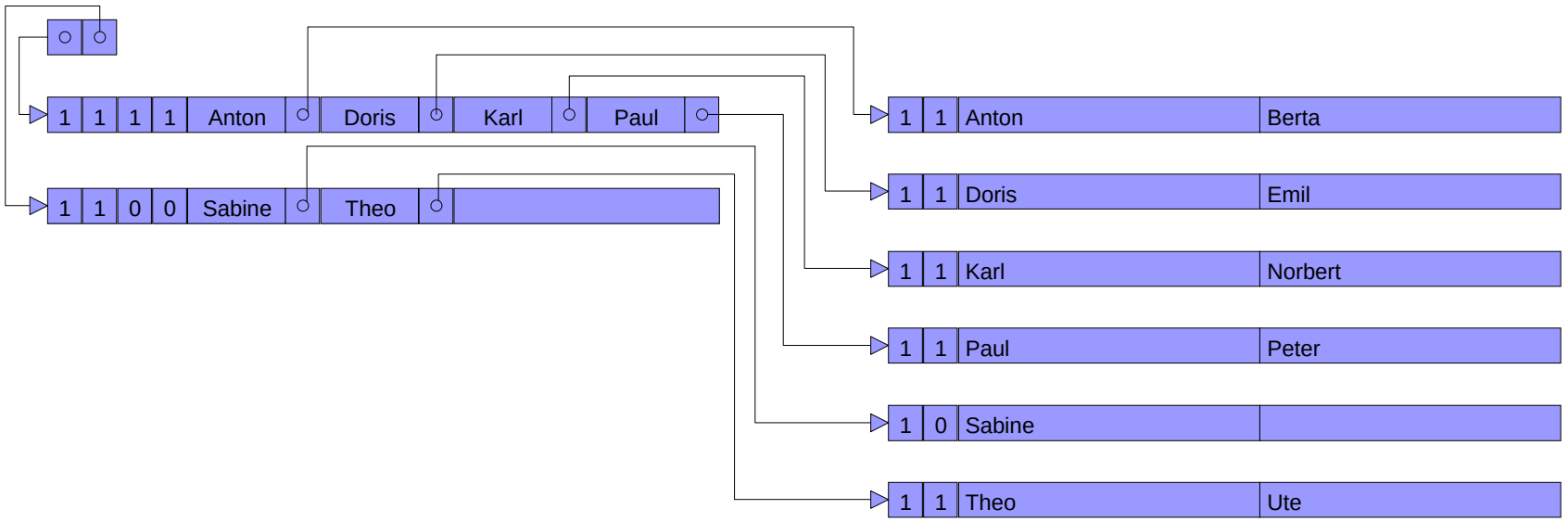
- **LOOKUP** (für Schlüssel v):
  Suche in Index-Datei den letzten Block mit erstem Eintrag $v_2 \leq v$. Suche in diesem Block das letzte Paar (v_3, a) mit $v_3 \leq v$. Lies Block mit Adresse *a* und durchsuche ihn nach Schlüssel *v*.

- **INSERT:**
  Zunächst LOOKUP. Falls Block noch Platz für Record hat: einfügen. Falls Block voll ist: Nachfolgerblock oder neuen Block wählen und Index anpassen.

- **DELETE:**
  Analog zu INSERT

- **MODIFY:**
  Zunächst LOOKUP. Falls Schlüssel an Änderung beteiligt: DELETE + INSERT. Sonst: Record ändern, Block zurückschreiben.

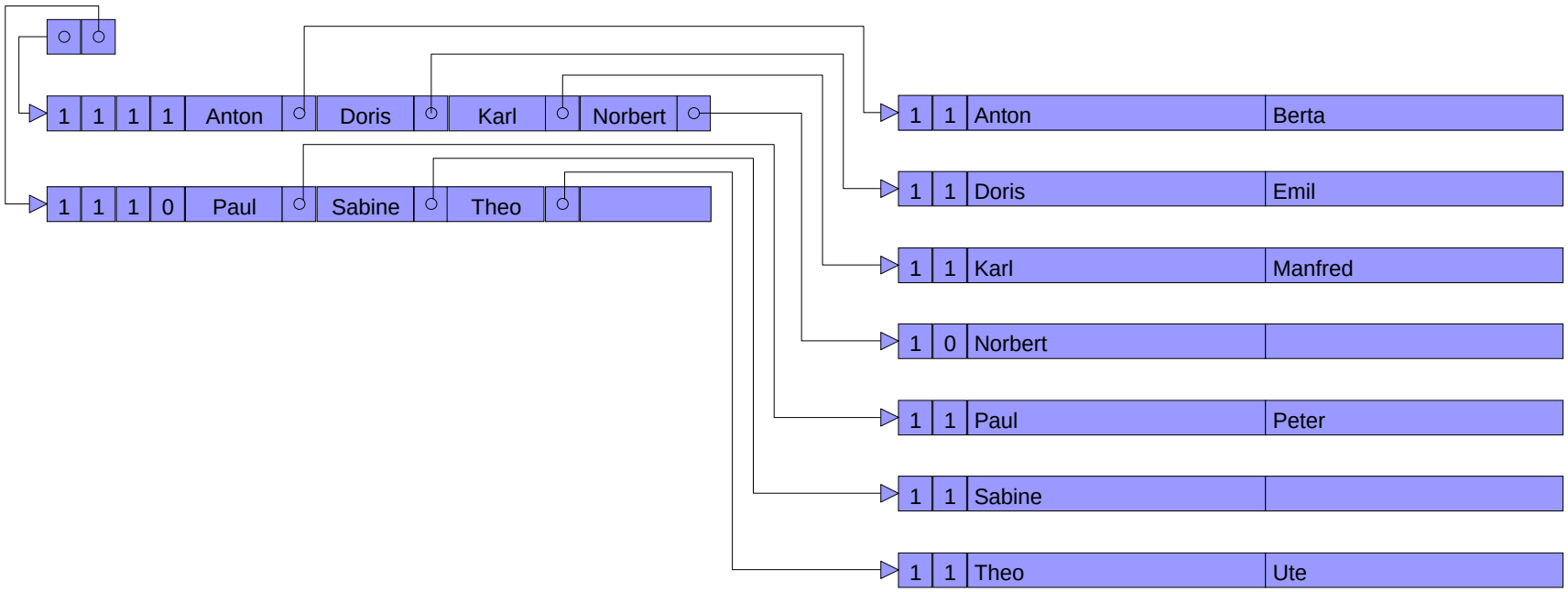
Beispiel für Indexorganisation



Index-Organisation: Ausgangslage

Manfred einfügen

Beispiel für Indexorganisation



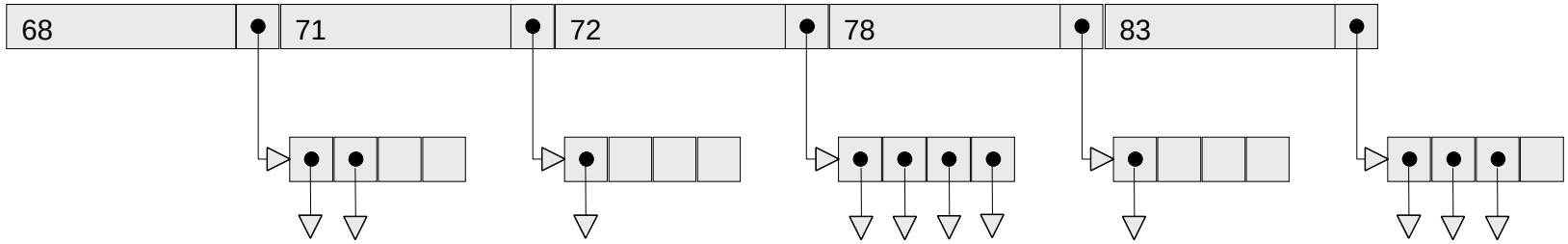
Index-Organisation: nach Einfügen von Manfred

Sekundär-Index

Sekundärindex besteht aus Index-File mit Einträgen der Form **<Attributwert, Adresse>**.

Liegt **<v,a>** im Sekundärindex, so verweist a auf Block mit Verweisen auf Records in Hauptdatei mit Attribut ≥ **v**

Sekundär-Index für Gewicht



Beispiel zur physikalischen Speicherung

Gegeben: 300.000 Records

Attribut	Bytes
Pers-Nr.	15
Vorname	15
Nachname	15
Straße	25
PLZ	5
Ort	25

Platzbedarf pro Record: 100 Bytes.

Die Blockgröße betrage 1024 Bytes.

Fragen zur Zahl der Records

Wieviel Daten-Records passen in einen zu 100% gefüllten Datenblock?

1024 / 100 = 10

Wieviel Daten-Records passen in einen zu 75% gefüllten Datenblock?

10 * 0,75 = 7-8

Wieviel Schlüssel / Adresspaare passen in einen zu 100% gefüllten Indexblock?

1.024 / (15+4) = 53

Wieviel Schlüssel / Adresspaare passen in einen zu 75% gefüllten Indexblock?

1.024 / (15+4)*0,75 ≈ 40

Heapfile versus ISAM

Welcher Platzbedarf entsteht beim Heapfile?

300.000 / 10 = 30.000 Blöcke

Wieviel Blockzugriffe entstehen im Mittel beim Heapfile?

30.000 / 2 = 15.000

Welcher Platzbedarf entsteht im Mittel bei ISAM?

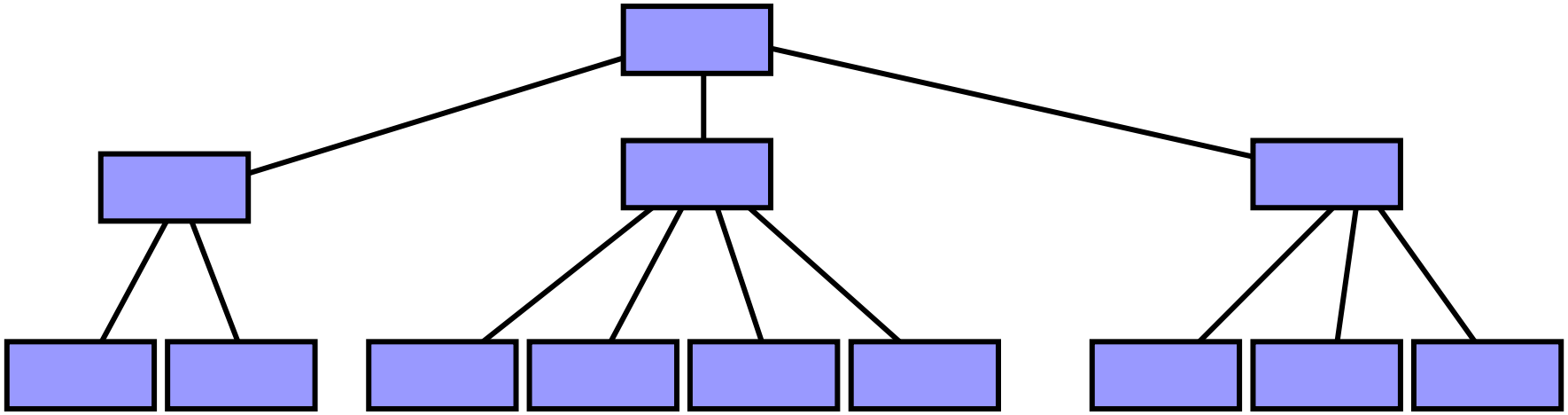
300.000 / 7,5 $\approx$ 40.000 zu 75% gefüllte Datenblöcke +

40.000 / 40 $\approx$ 1.000 zu 75% gefüllte Indexblöcke

Wieviel Blockzugriffe entstehen im Mittel bei ISAM?

$\log_2(1.000) + 1 \approx 11$ Blockzugriffe

B*-Baum



- Jeder Weg von der Wurzel zu einem Blatt hat dieselbe Länge.
- Jeder Knoten außer der Wurzel und den Blättern hat mindestens *k* Nachfolger.
- Jeder Knoten hat höchstens 2 • *k* Nachfolger.
- Die Wurzel hat keinen oder mindestens 2 Nachfolger.