Datenbanken

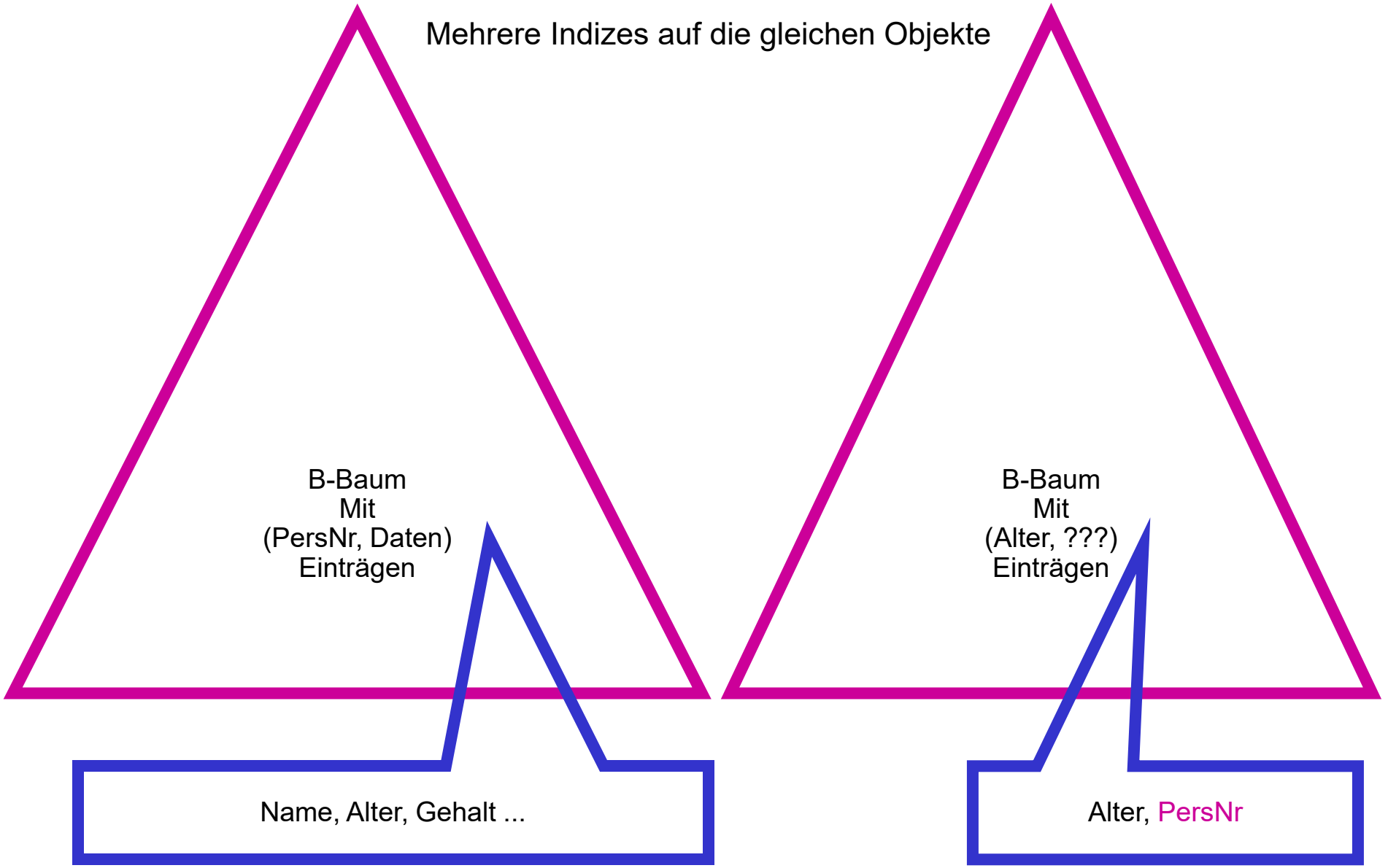
Gliederung

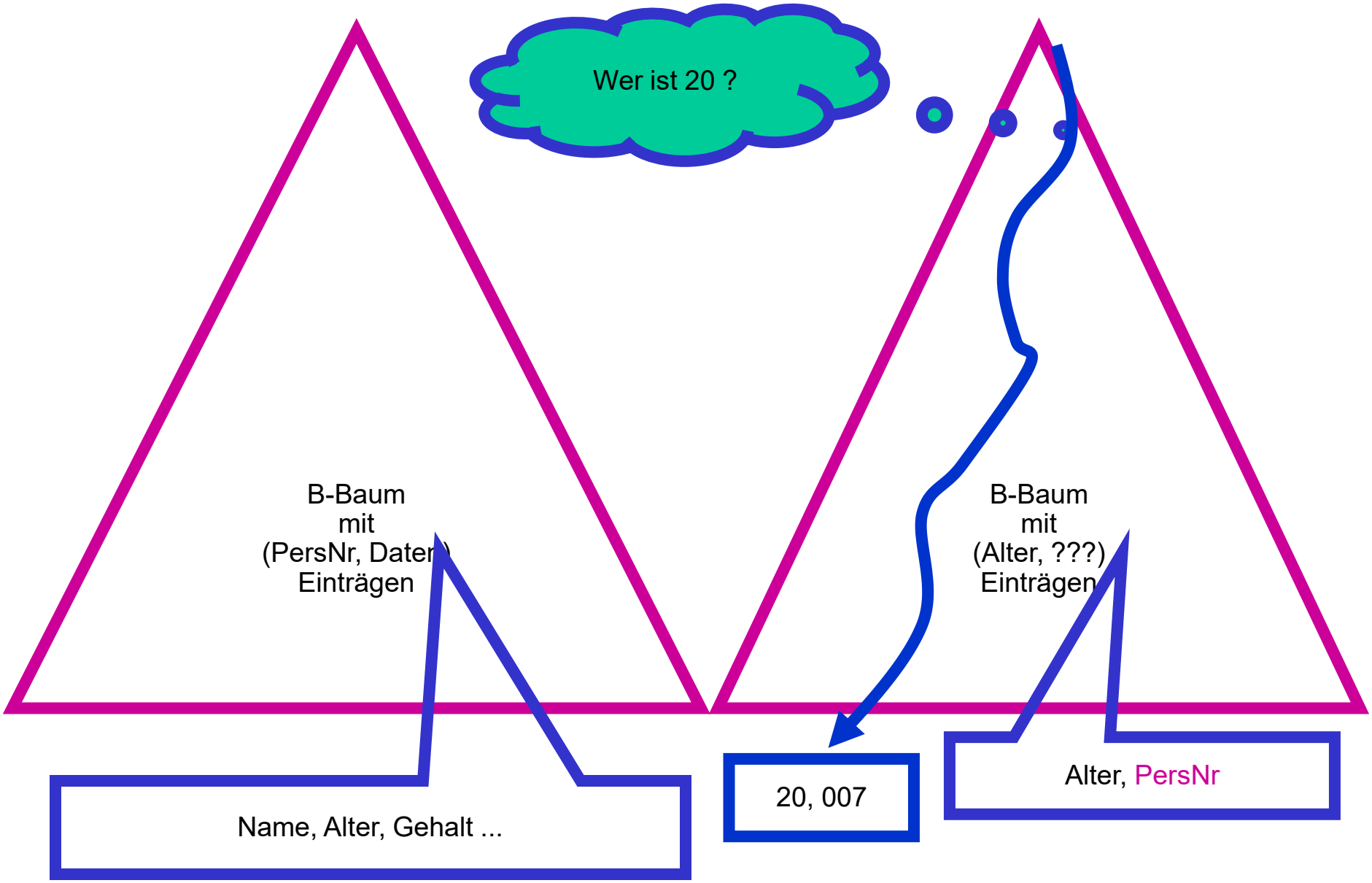
Gliederung

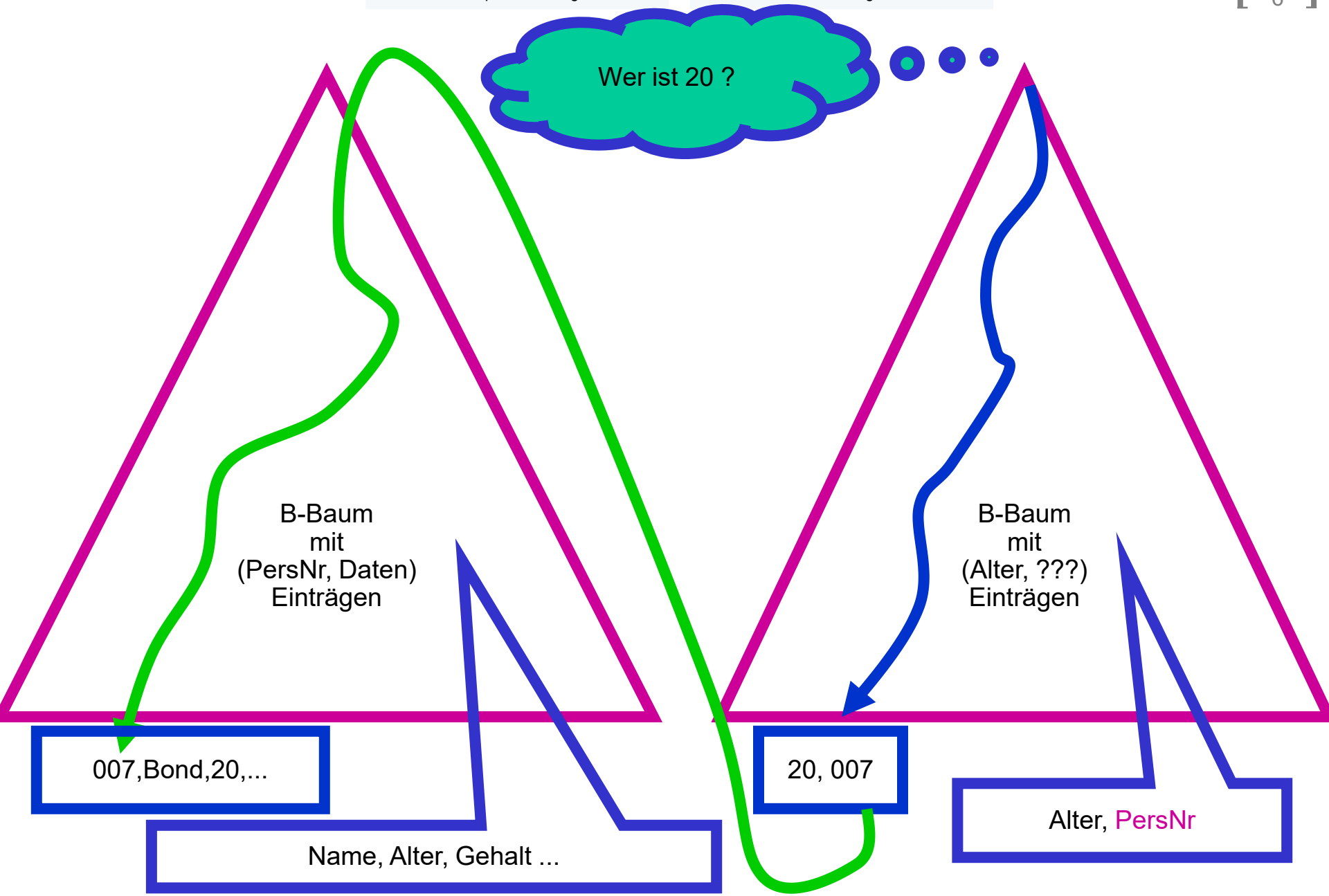
4. Physische Datenorganisation

- Aufbau eines DBMS
- Deklarative vs. Prozedurale Programmiersprache
- Ablauf einer Anfrageverarbeitung
- Aufbau und Organisation von Speichermedien
- Aufbau von Dateien und Zugriff auf die Daten

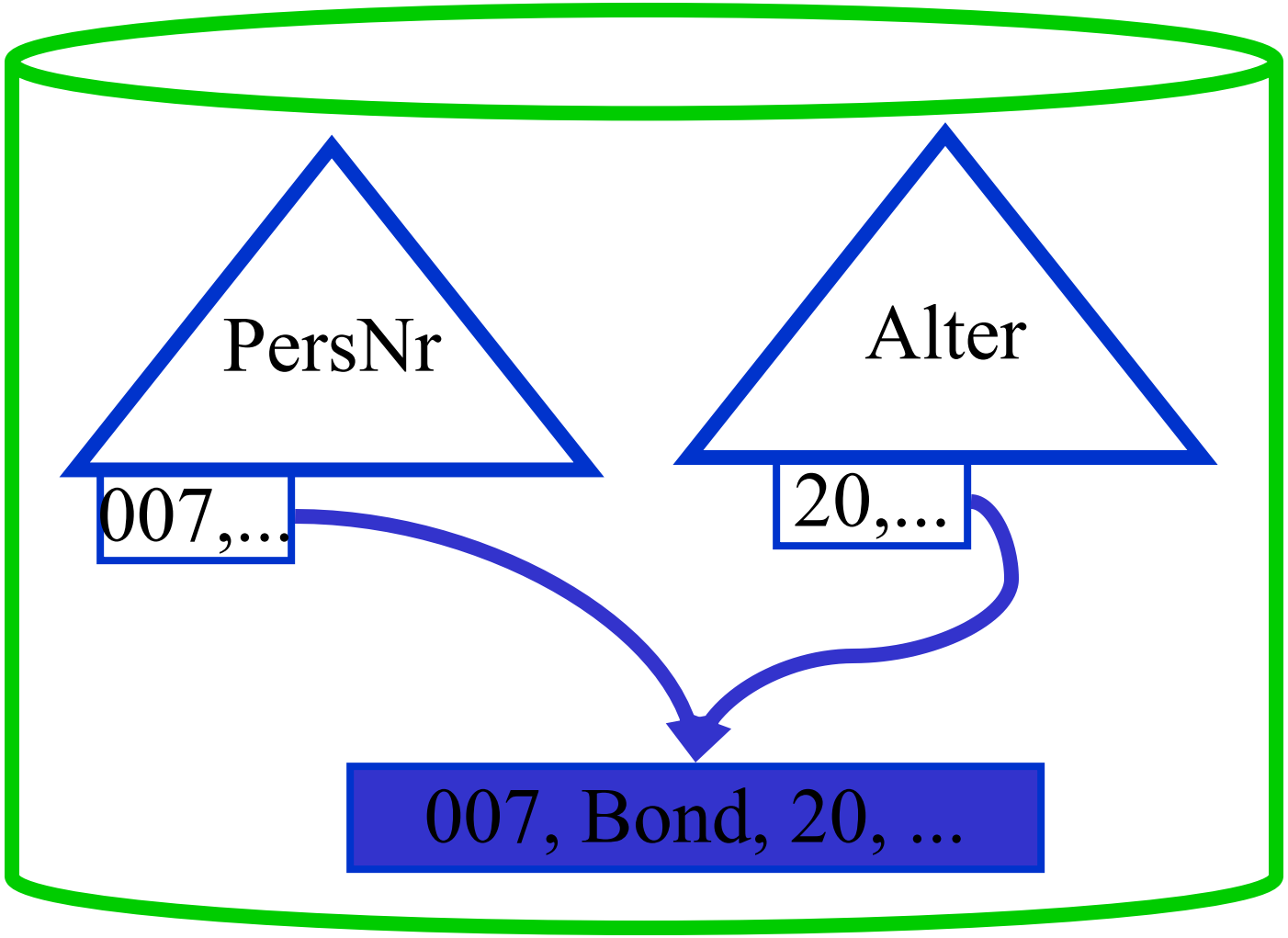
Mehrere Indizes auf die gleichen Objekte



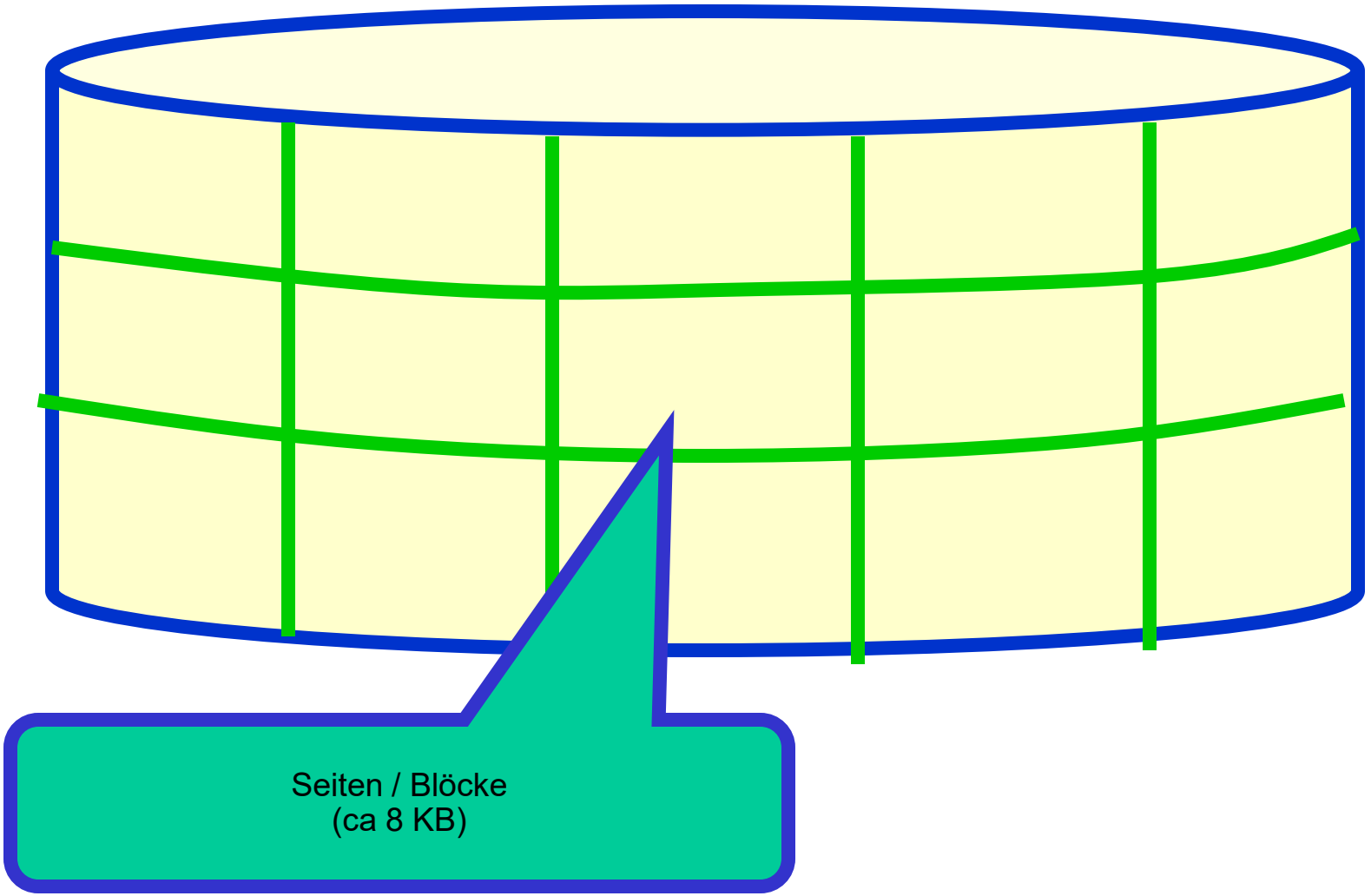

Wer ist 20 ?
B-Baum
mit
(PersNr, Daten)
Einträgen
B-Baum
mit
(Alter, ???)
Einträgen
Name, Alter, Gehalt ...
20, 007
Alter, PersNr


Wer ist 20 ?
B-Baum
mit
(PersNr, Daten)
Einträgen
B-Baum
mit
(Alter, ???)
Einträgen
007,Bond,20,...
Name, Alter, Gehalt ...
20, 007
Alter, PersNr

Eine andere Möglichkeit: Referenzierung über Speicheradressen



Realisierungstechnik für Hintergrundspeicher-Adressen



„Statische“ Hashtabellen

- á priori Allokation des Speichers
- Nachträgliche Vergrößerung der Hashtabelle ist „teuer“
  - Hashfunktion h(...) = ... mod N
  - Rehashing der Einträge
    - h(...) = ... mod M
  - In Datenbankanwendungen viele GB

- Bsp: Matrikelnummer, Sportverein o.ä.

Matr.No. xxx.xxx = 10^6	, # Studierende =	6000
Reserve + 20 %		7200
Nächste Primzahl		7207

„Statische“ Hashtabellen

Bsp: Matrikelnummer

Matr.No.: xxx.xxx = 10^6 # Studierende = 6000

Reserve + 20 % 7200

Nächste Primzahl 7207

f(x) = H mod(7207)

Matr. No	f(x)
7207	0
7208	1
14416	2 > 21623
7206	
21623	

„Statische“ Hashtabellen

- Erweiterbares Hashing
  - Zusätzliche Indirektion über ein Directory
  - Ein zusätzlicher Zugriff auf ein Directory, das den Zeiger (Verweis, BlockNr) des Hash-Bucket enthält
  - Dynamisches Wachsen (und Schrumpfen) ist möglich
  - Der Zugriff auf das Directory erfolgt über einen binären Hashcode

„Statische“ Hashtabellen

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

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

Buckets
Kapazität:
Seitengröße
/ Satzgröße

- Kollisionsbehandlung

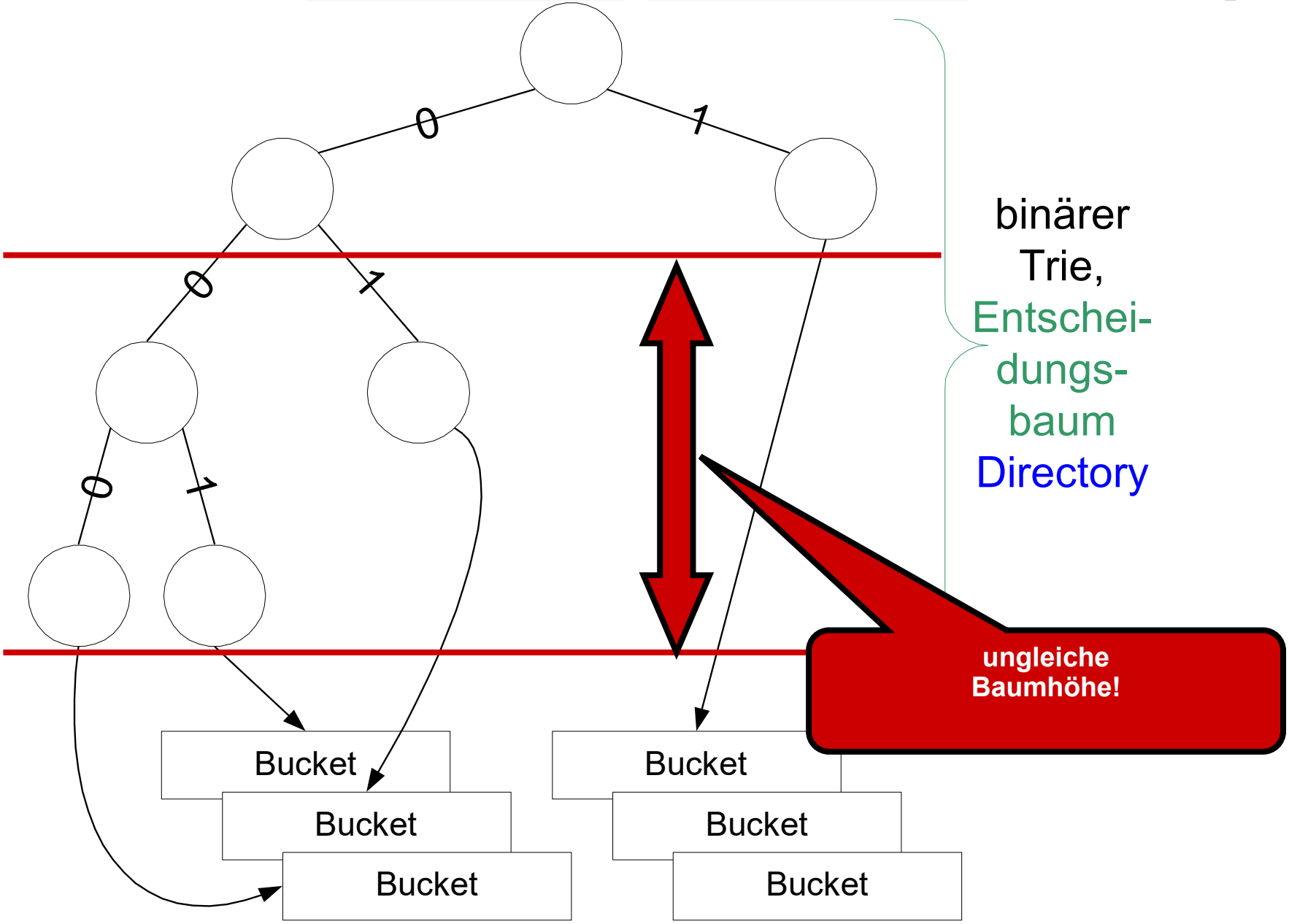
0		
1	(27550, 'Schopenhauer', 6)	
	(26830, 'Aristoxenos', 8)	
2	(24002, 'Xenokrates', 18)	→ (26120, 'Fichte', 10) → ...
	(25403, 'Jonas', 12)	(28106, 'Carnap', 3)

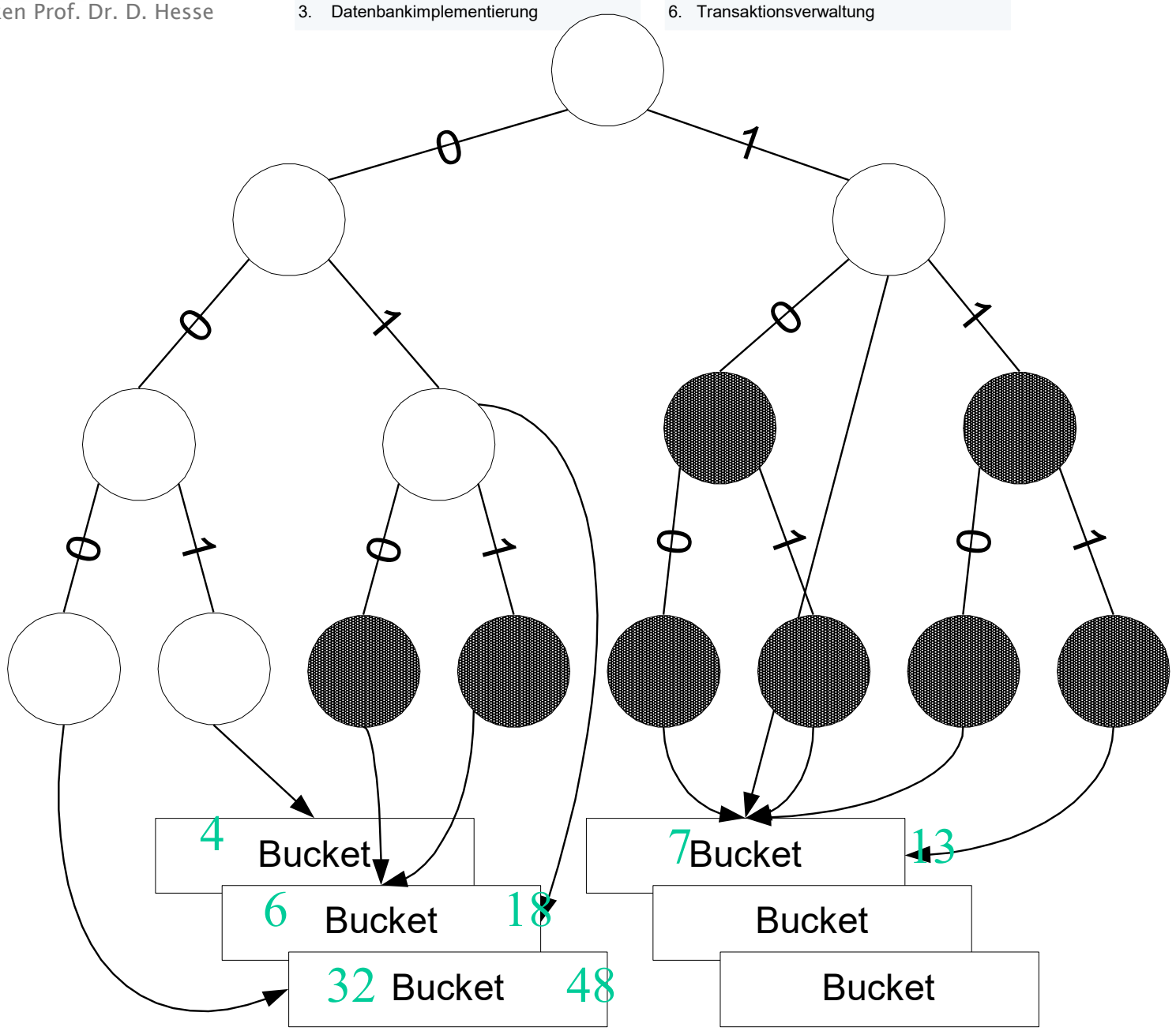
Überlauf-bucket

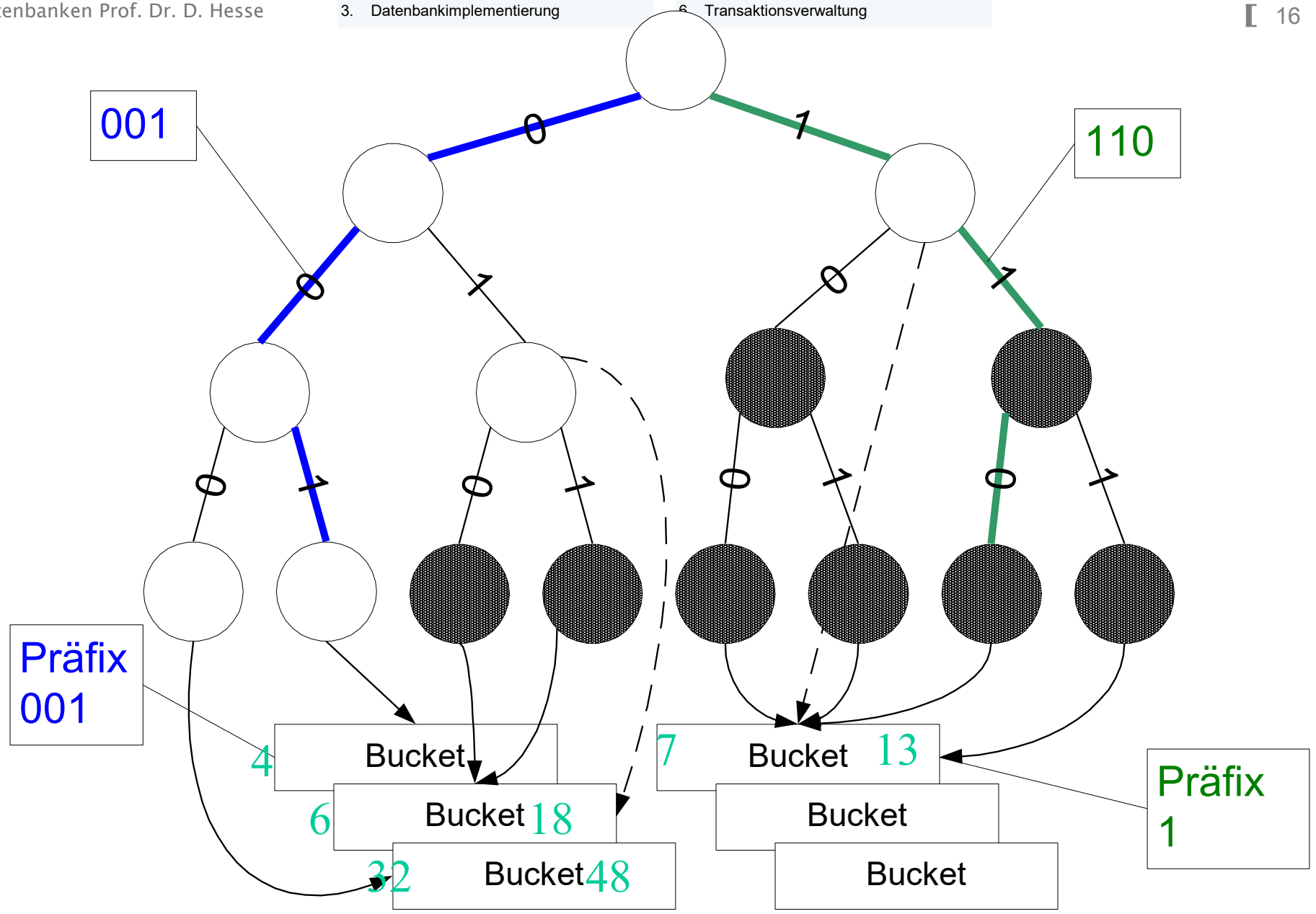
$\Longrightarrow$ ineffizient bei nicht vorhersehbarer Datenmenge

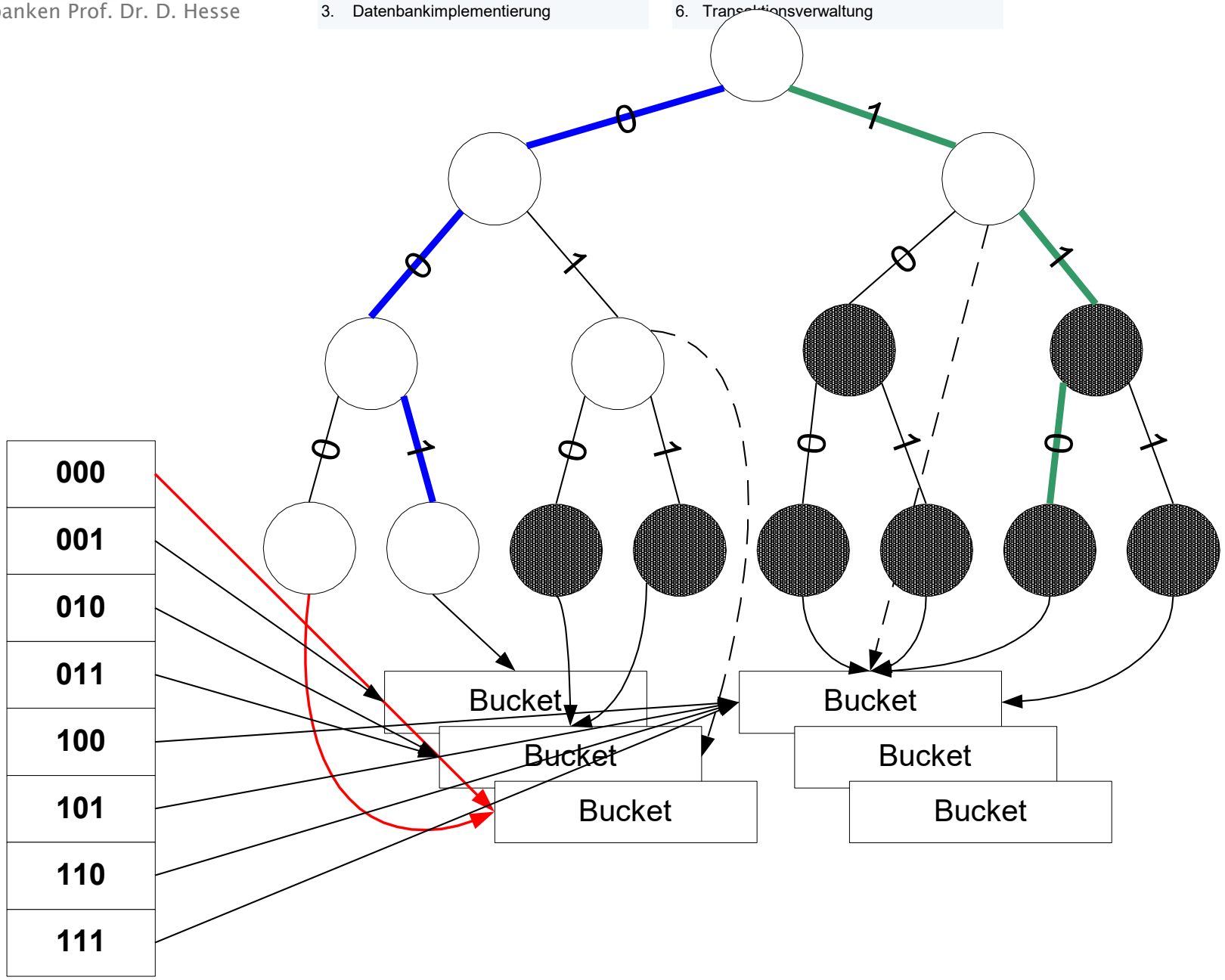
Hashfunktion für erweiterbares Hashing

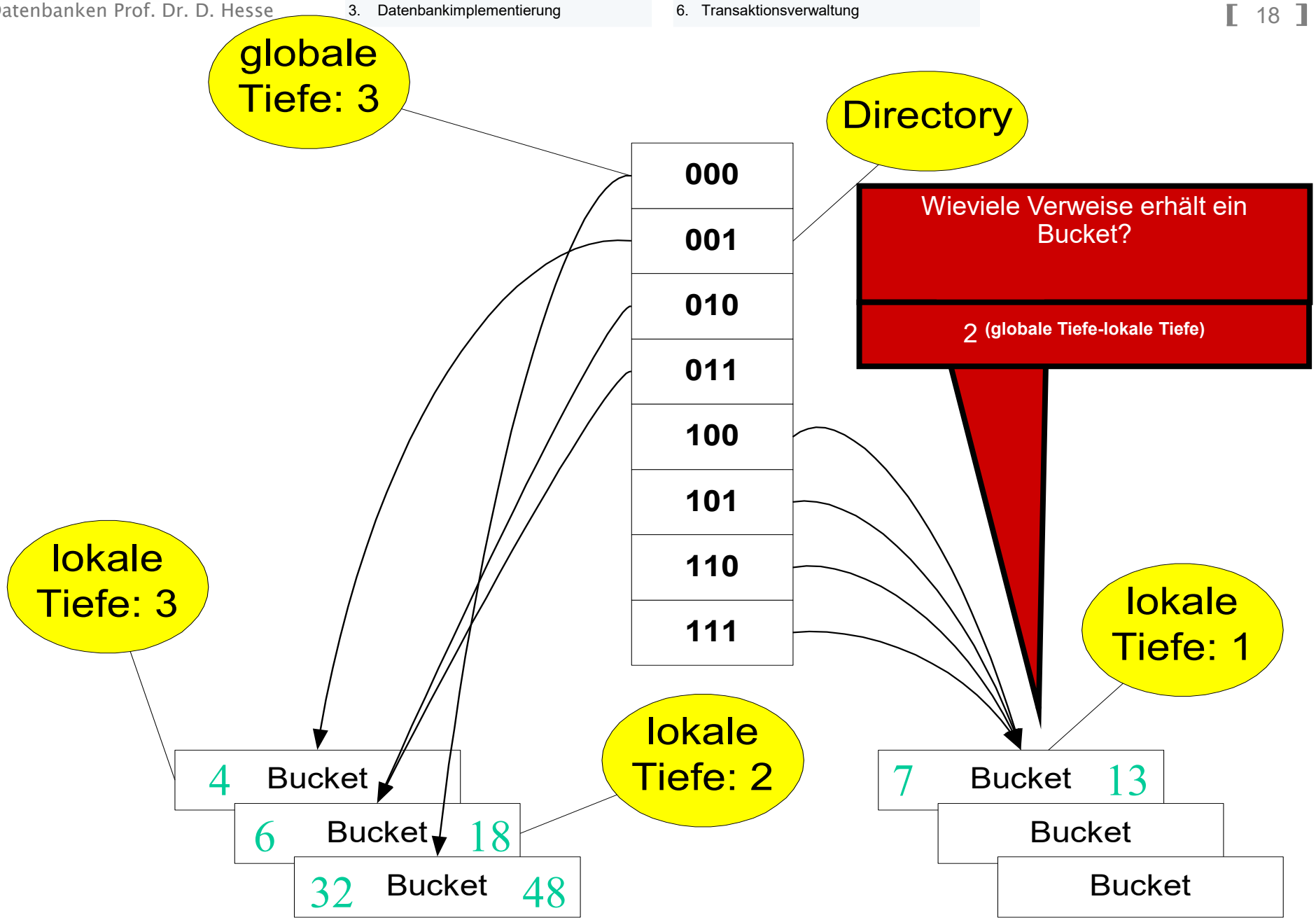
- h: Schlüsselmenge → {0,1}*
- Der Bitstring muss lang genug sein, um alle Objekte auf ihre Buckets abbilden zu können
- Anfangs wird nur ein (kurzer) Präfix des Hashwertes (Bitstrings) benötigt
- Wenn die Hashtabelle wächst wird aber sukzessive ein längerer Präfix benötigt
- Beispiel-Hashfunktion: gespiegelte binäre PersNr
  - h(004) = 001000000... (4=0..0100)
  - h(006) = 011000000... (6=0..0110)
  - h(007) = 111000000... (7 =0..0111)
  - h(013) = 101100000... (13 =0..01101)
  - h(018) = 0100100000... (18 =0..010010)
  - h(032) = 000001000... (32 =0..0100000)
  - H(048) = 000011000... (48 =0..0110000)

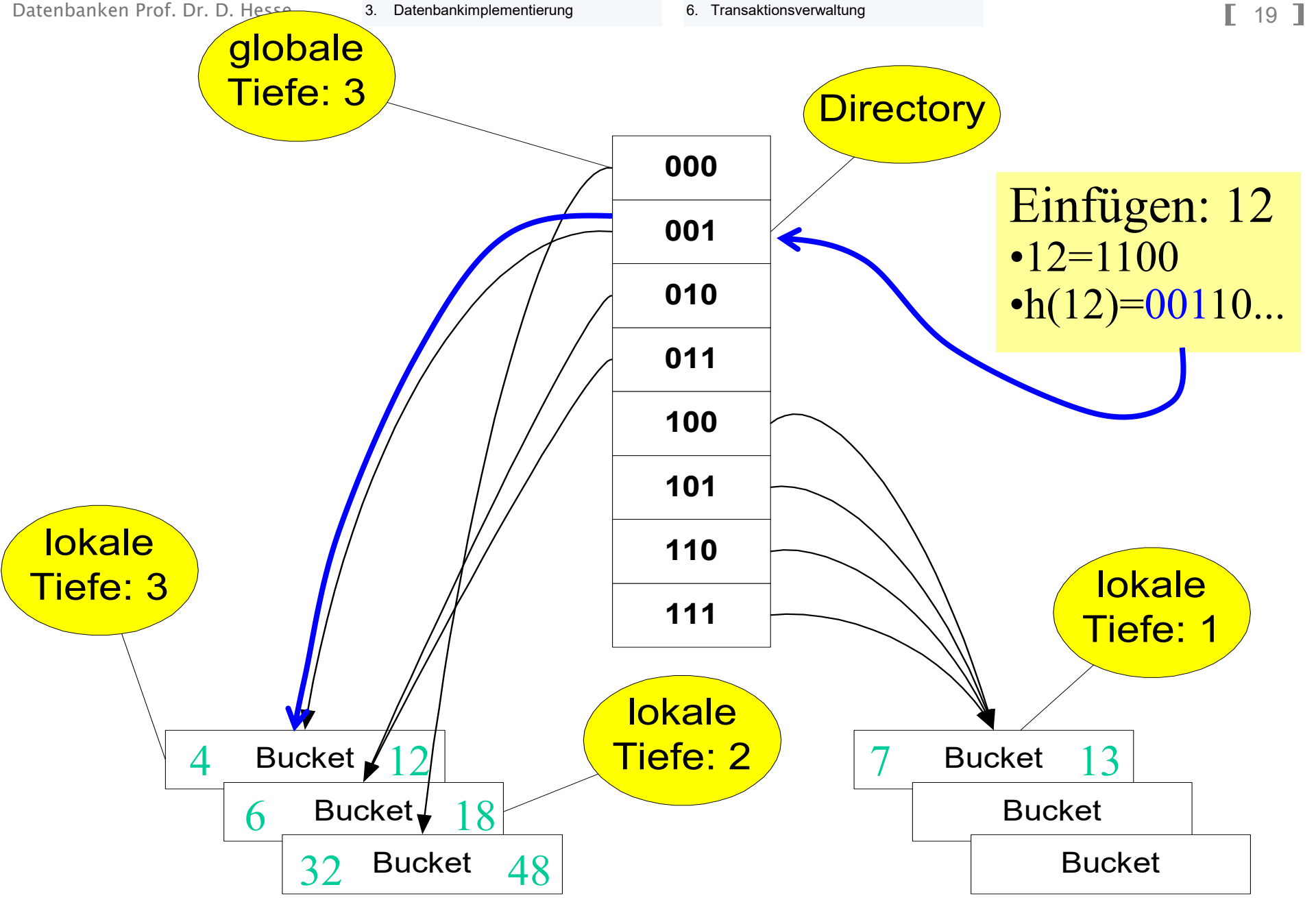

0
1
0
1
0
1
Bucket
Bucket
Bucket
Bucket
Bucket
Bucket
binärer Trie, Entschei-dungs-baum Directory
ungleiche Baumhöhe!

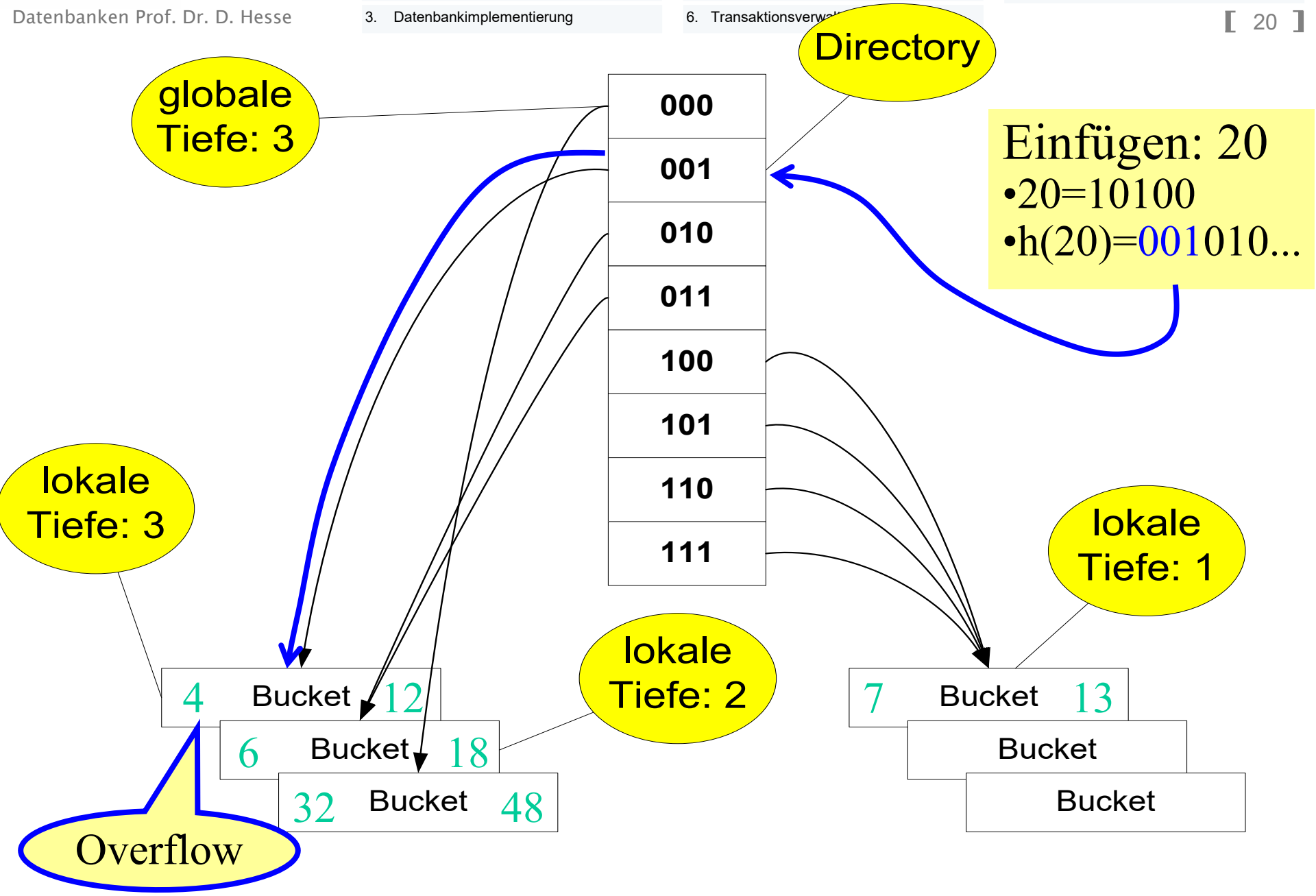

0
1
0
1
0
1
0
1
0
1
0
1
0
1
4
Bucket
6
Bucket
18
32
Bucket
48
7
Bucket
13
Bucket
Bucket

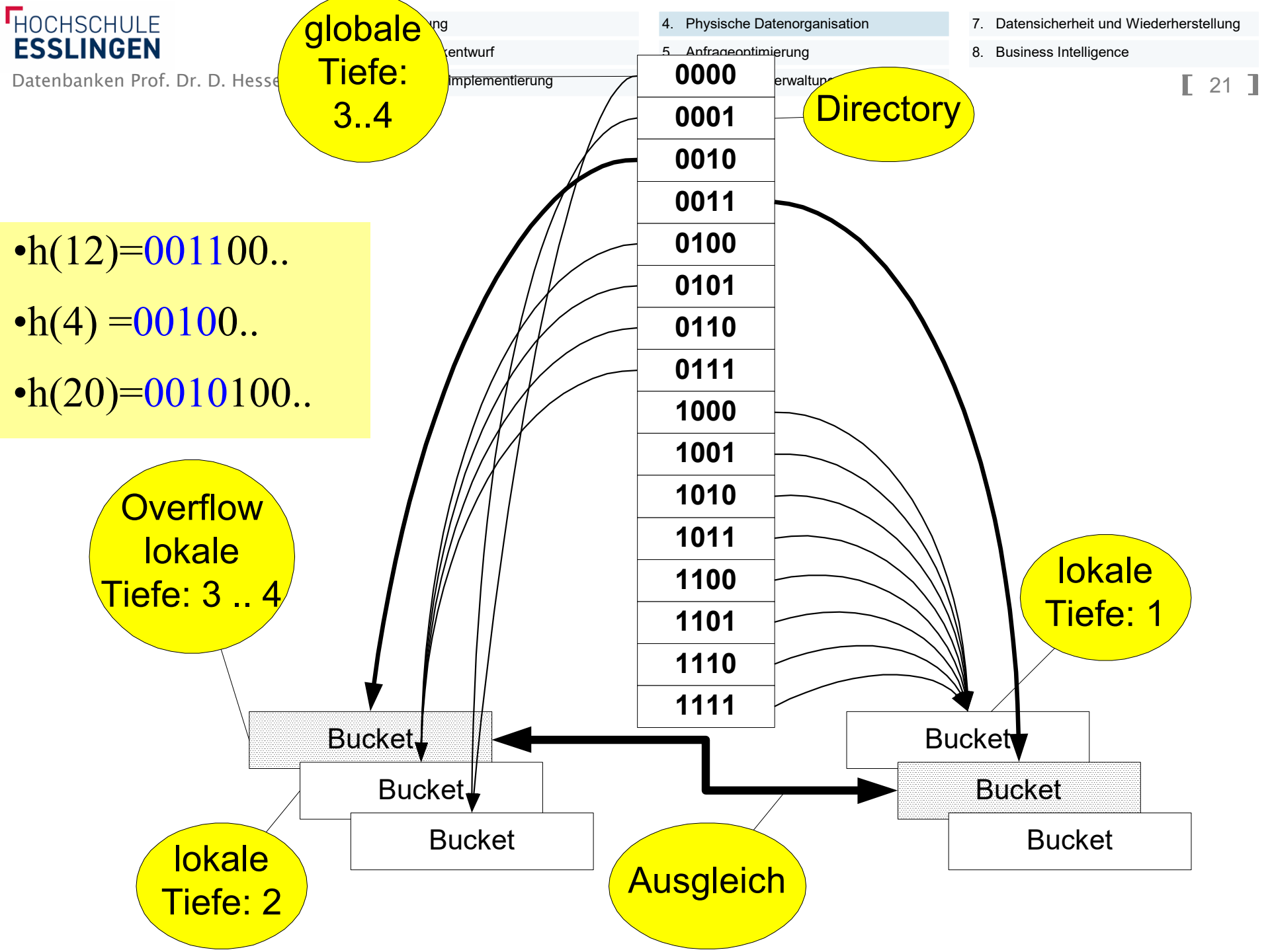

001
110
0
1
Präfix
001
4
Bucket
6
Bucket 18
32
Bucket 48
7
Bucket 13
Bucket
Bucket
Präfix
1

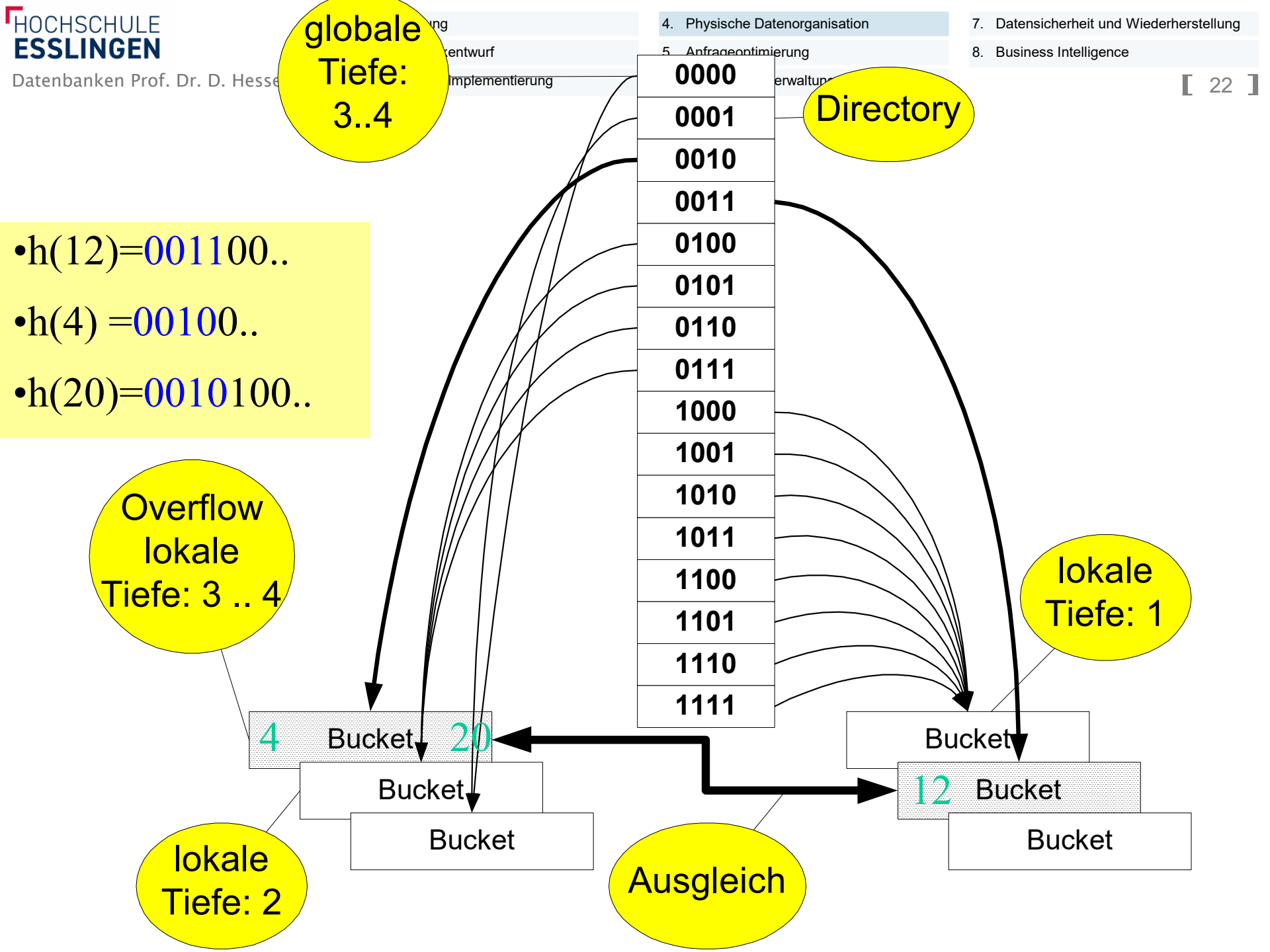

0
1
000
001
010
011
100
101
110
111
Bucket
Bucket
Bucket
Bucket
Bucket
Bucket

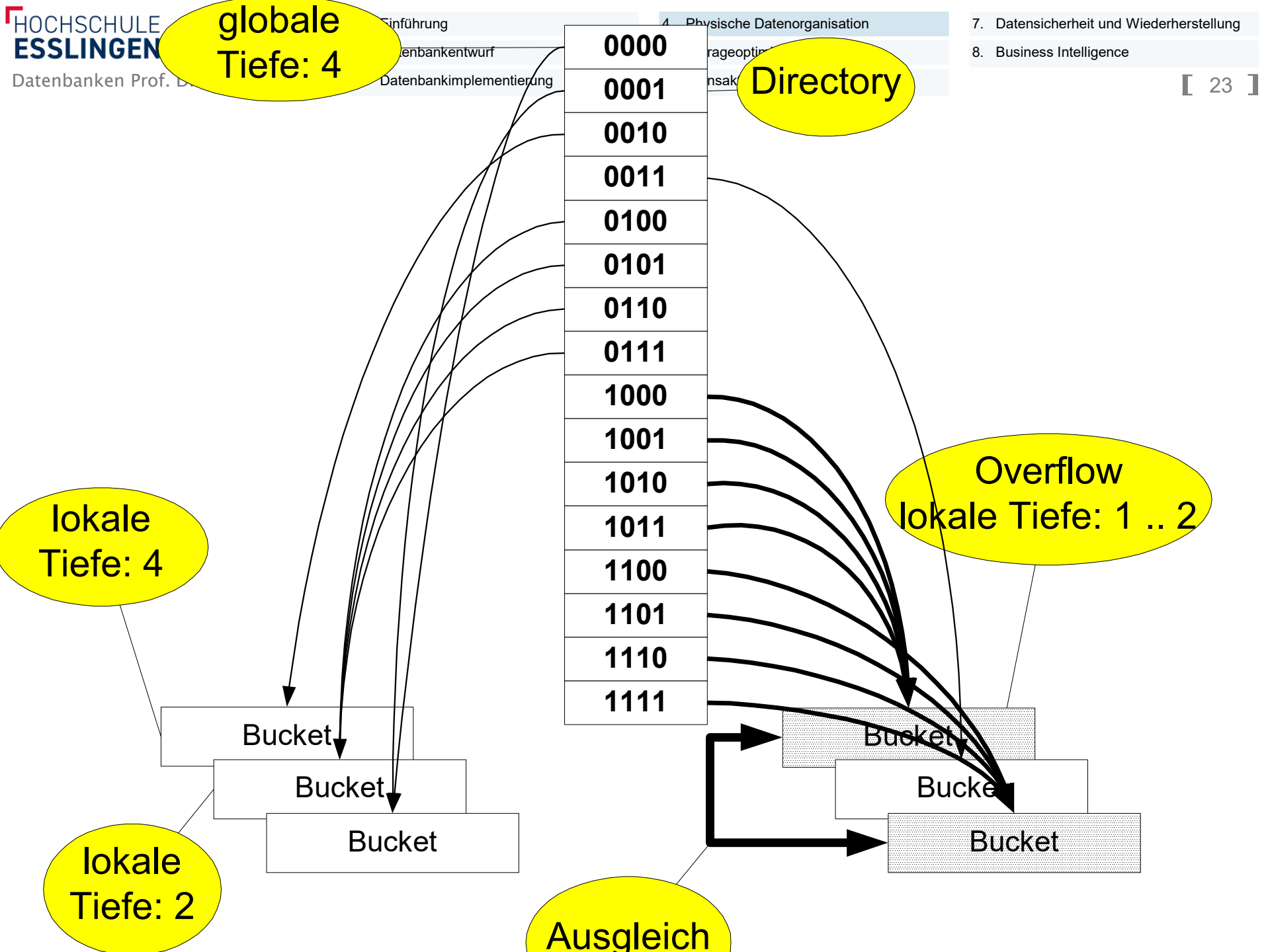

globale Tiefe: 3
Directory
000
001
010
011
100
101
110
111
Wieviele Verweise erhält ein Bucket?
2 (globale Tiefe-lokale Tiefe)
lokale Tiefe: 3
lokale Tiefe: 2
lokale Tiefe: 1
4 Bucket
6 Bucket 18
32 Bucket 48
7 Bucket 13
Bucket
Bucket


globale Tiefe: 3
Directory
000
001
010
011
100
101
110
111
Einfügen: 12
•12=1100
•h(12)=00110...
lokale Tiefe: 3
lokale Tiefe: 2
lokale Tiefe: 1
4 Bucket 12
6 Bucket 18
32 Bucket 48
7 Bucket 13
Bucket
Bucket


Directory
globale Tiefe: 3
000
001
010
011
100
101
110
111
Einfügen: 20
•20=10100
•h(20)=001010...
lokale Tiefe: 3
lokale Tiefe: 2
lokale Tiefe: 1
4 Bucket 12
6 Bucket 18
32 Bucket 48
7 Bucket 13
Bucket
Bucket
Overflow

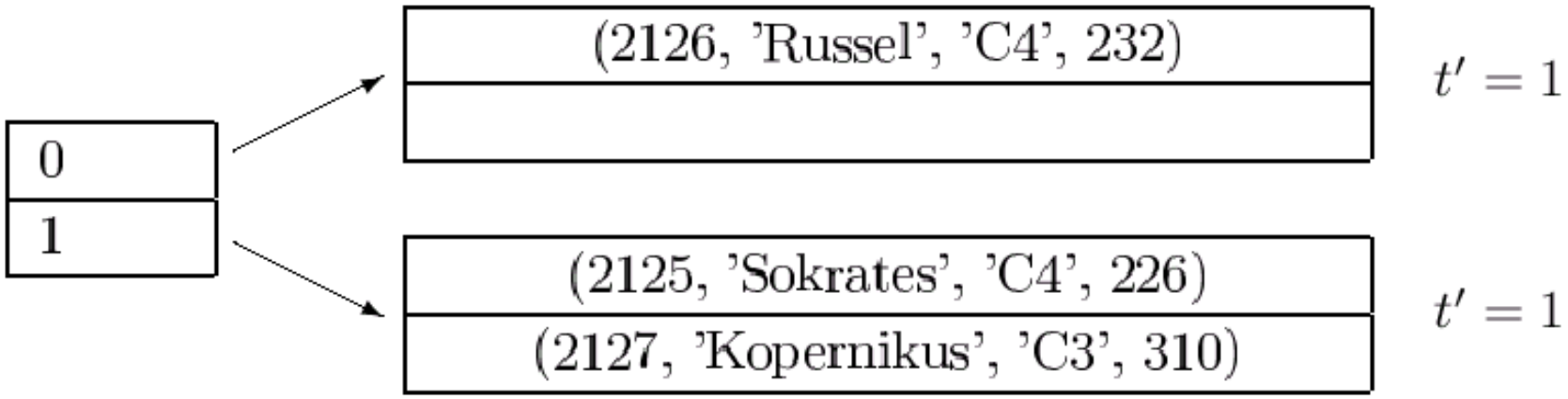

globale Tiefe: 3..4
Directory
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
•h(12)=001100..
•h(4) =00100..
•h(20)=0010100..
Overflow lokale Tiefe: 3 .. 4
lokale Tiefe: 1
Bucket
Bucket
Bucket
Bucket
Bucket
Bucket
lokale Tiefe: 2
Ausgleich


globale Tiefe: 3..4
Directory
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
•h(12)=001100..
•h(4) =00100..
•h(20)=0010100..
Overflow lokale Tiefe: 3 .. 4
lokale Tiefe: 1
4 Bucket 20
Bucket
Bucket
Bucket
12 Bucket
Bucket
lokale Tiefe: 2
Ausgleich

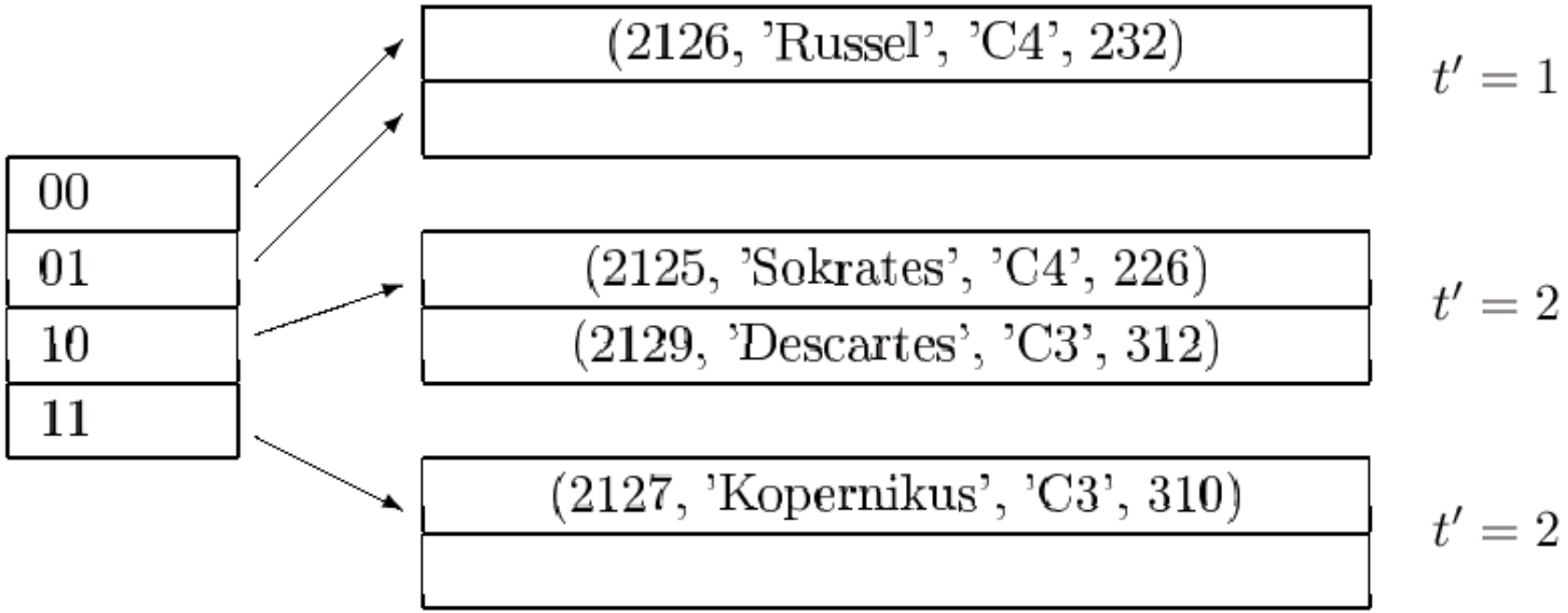

globale Tiefe: 4
Directory
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
lokale Tiefe: 4
Overflow
lokale Tiefe: 1 .. 2
Bucket
Bucket
Bucket
Bucket
Bucke
Bucket
lokale Tiefe: 2
Ausgleich

Demonstration des erweiterbaren Hashings

x	$h(x)$	
	d	p
2125	1	01100100001
2126	0	11100100001
2127	1	11100100001



x	$h(x)$	
	d	p
2125	10	1100100001
2126	01	1100100001
2127	11	1100100001
2129	10	0010100001



Mehrdimensionale Datenstrukturen

- Wertbasierter Zugriff auf der Grundlage mehrerer Attribute, dies einzeln oder in beliebigen Kombinationen.
- Typische Anforderungen aus CAD, VLSI-Entwurf, Kartographie,...
- Anfragen decken den Bereich ab zwischen
  - mehrdimensionalem Punktzugriff (EMQ) und
  - mehrdimensionalen Bereichsanfragen (RQ)
- Lösung mit eindimensionalen Indexen
  - erfordert konjunktive Zerlegung der Anfrage in Einattributanfragen und Schnittmengenbildung
  - bedingt hohe Speicherredundanz
- Problemstellung:
  - Mehrdimensionale Nachbarschaftsverhältnisse

Grundlagen mehrdimensionaler Datenstrukturen

- Wertebereiche $D_0,\ldots, D_{k-1}$:

alle D_i sind endlich, linear geordnet und besitzen kleinstes ($-\infty_i$) und größtes (∞_i) Element

- Datenraum $\boldsymbol{D} = D_0 \times \ldots \times D_{k-1}$
- k-dimensionaler Schlüssel entspricht Punkt im Datenraum $p \in \boldsymbol{D}$

Grundlagen mehrdimensionaler Datenstrukturen

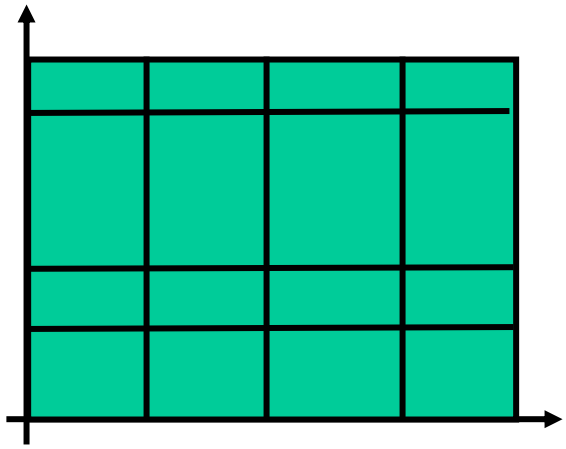
1. Exact Match Query
   spezifiziert Suchwert für jede Dimension D_i
2. Partial Match Query
   spezifiziert Suchwert für einen Teil der Dimensionen
3. Range Query
   spezifiziert ein Suchintervall $[ug_i, og_i]$ für alle Dimensionen
4. Partial Range Query
   spezifiziert ein Suchintervall für einen Teil der Dimensionen

Charakterisierung mehrdimensionaler Datenstrukturen

Mehrdimensionale Zugriffsstrukturen können gemäß der Art der Aufteilung des Datenraums in Gebiete charakterisiert werden:

1. nur atomare Gebiete (beschreibbar durch ein Rechteck)
2. vollständig (die Vereinigung aller Gebiete ergibt den gesamten Datenraum)
3. disjunkt (die Gebiete überlappen nicht)

Grid-File (Gitter-Datei): atomar, vollständig, disjunkt

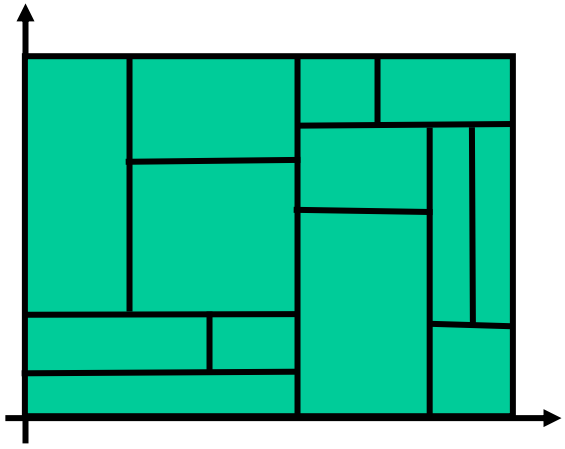


Charakterisierung mehrdimensionaler Datenstrukturen

Mehrdimensionale Zugriffsstrukturen können gemäß der Art der Aufteilung des Datenraums in Gebiete charakterisiert werden:

1. nur atomare Gebiete (beschreibbar durch ein Rechteck)
2. vollständig (die Vereinigung aller Gebiete ergibt den gesamten Datenraum)
3. disjunkt (die Gebiete überlappen nicht)

K-D-B-Baum: atomar, vollständig, disjunkt

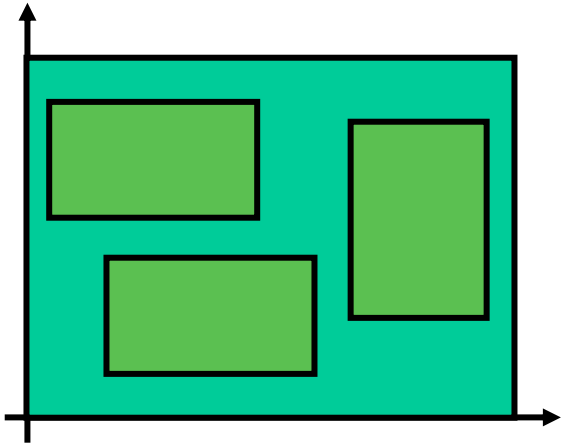


Charakterisierung mehrdimensionaler Datenstrukturen

Mehrdimensionale Zugriffsstrukturen können gemäß der Art der Aufteilung des Datenraums in Gebiete charakterisiert werden:

1. nur atomare Gebiete (beschreibbar durch ein Rechteck)
2. vollständig (die Vereinigung aller Gebiete ergibt den gesamten Datenraum)
3. disjunkt (die Gebiete überlappen nicht)

R^+-Baum: atomar, disjunkt

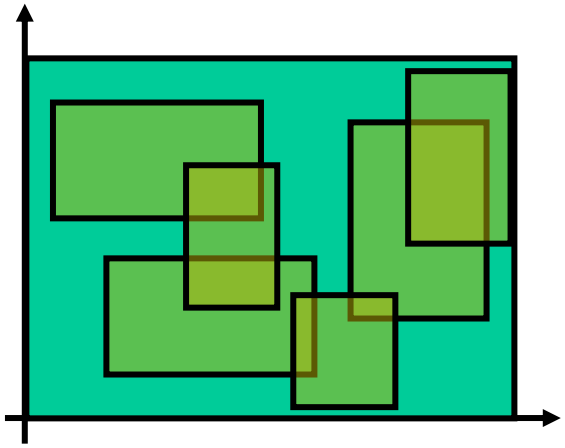


Charakterisierung mehrdimensionaler Datenstrukturen

Mehrdimensionale Zugriffsstrukturen können gemäß der Art der Aufteilung des Datenraums in Gebiete charakterisiert werden:

1. nur atomare Gebiete (beschreibbar durch ein Rechteck)
2. vollständig (die Vereinigung aller Gebiete ergibt den gesamten Datenraum)
3. disjunkt (die Gebiete überlappen nicht)

R-Baum: atomar

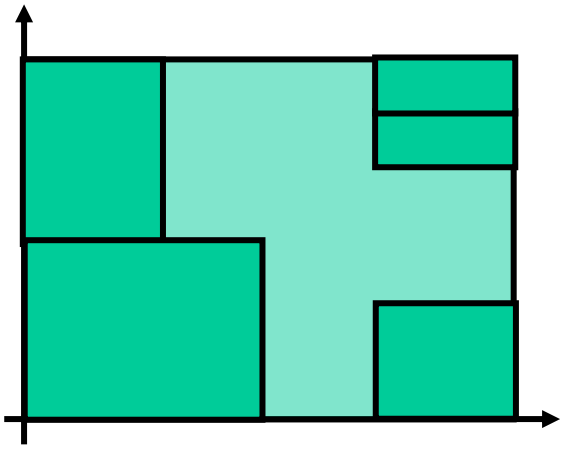


Charakterisierung mehrdimensionaler Datenstrukturen

Mehrdimensionale Zugriffsstrukturen können gemäß der Art der Aufteilung des Datenraums in Gebiete charakterisiert werden:

1. nur atomare Gebiete (beschreibbar durch ein Rechteck)
2. vollständig (die Vereinigung aller Gebiete ergibt den gesamten Datenraum)
3. disjunkt (die Gebiete überlappen nicht)

Buddy-Hash-Baum: atomar, disjunkt

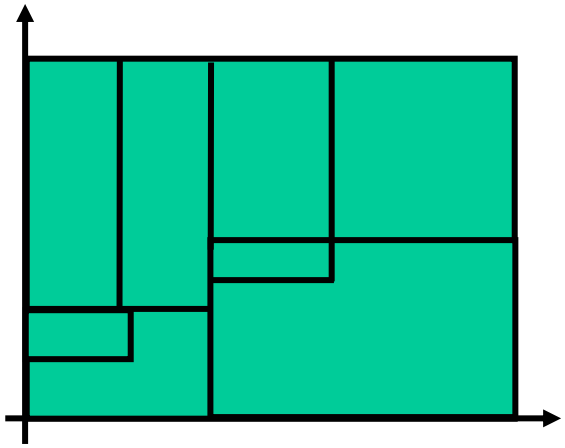


Charakterisierung mehrdimensionaler Datenstrukturen

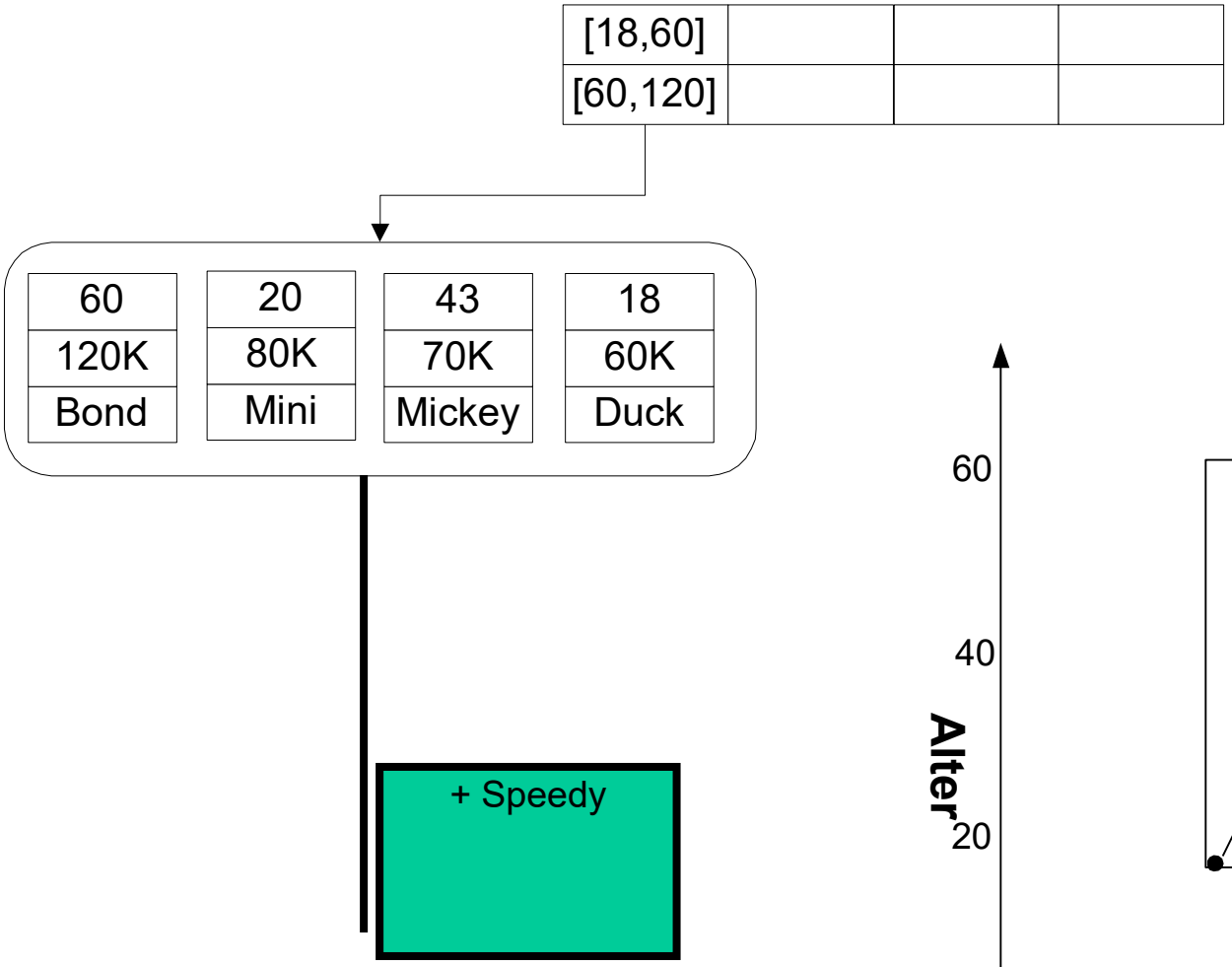
Mehrdimensionale Zugriffsstrukturen können gemäß der Art der Aufteilung des Datenraums in Gebiete charakterisiert werden:

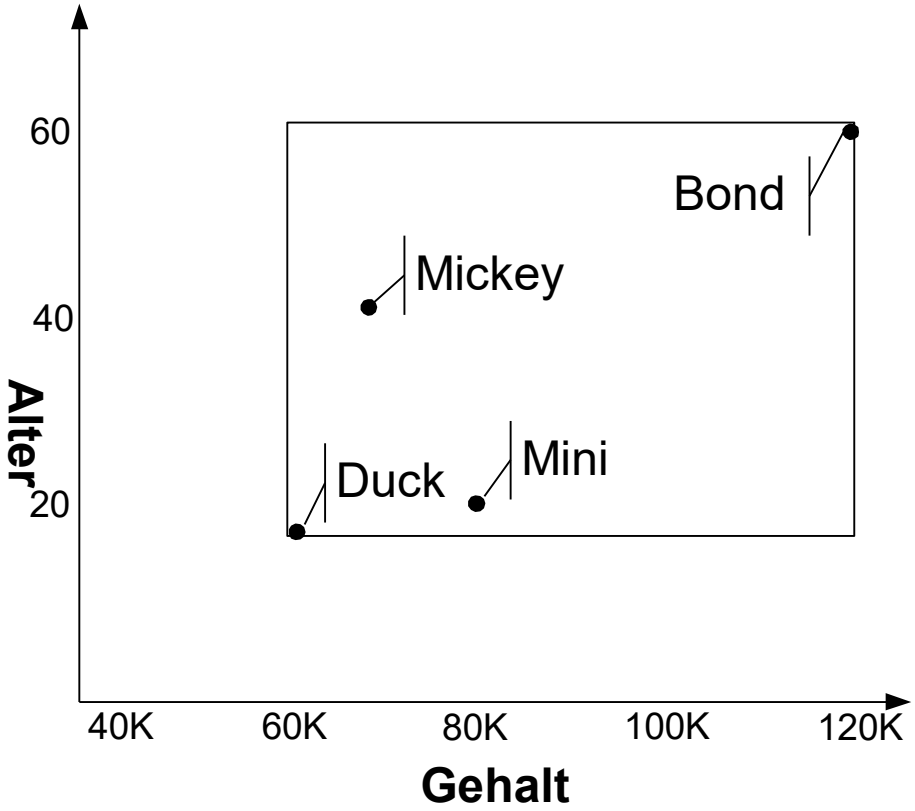
1. nur atomare Gebiete (beschreibbar durch ein Rechteck)
2. vollständig (die Vereinigung aller Gebiete ergibt den gesamten Datenraum)
3. disjunkt (die Gebiete überlappen nicht)

Z-B-Baum: vollständig,disjunkt



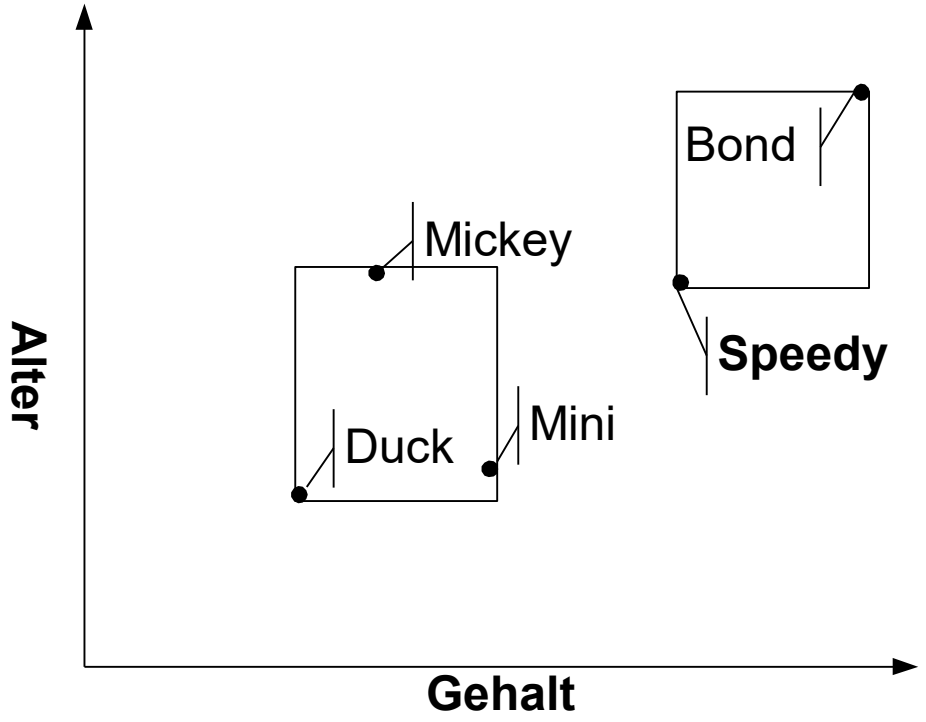
R-Baum: Urvater der baumstrukturierten mehrdimensionalen Zugriffsstrukturen



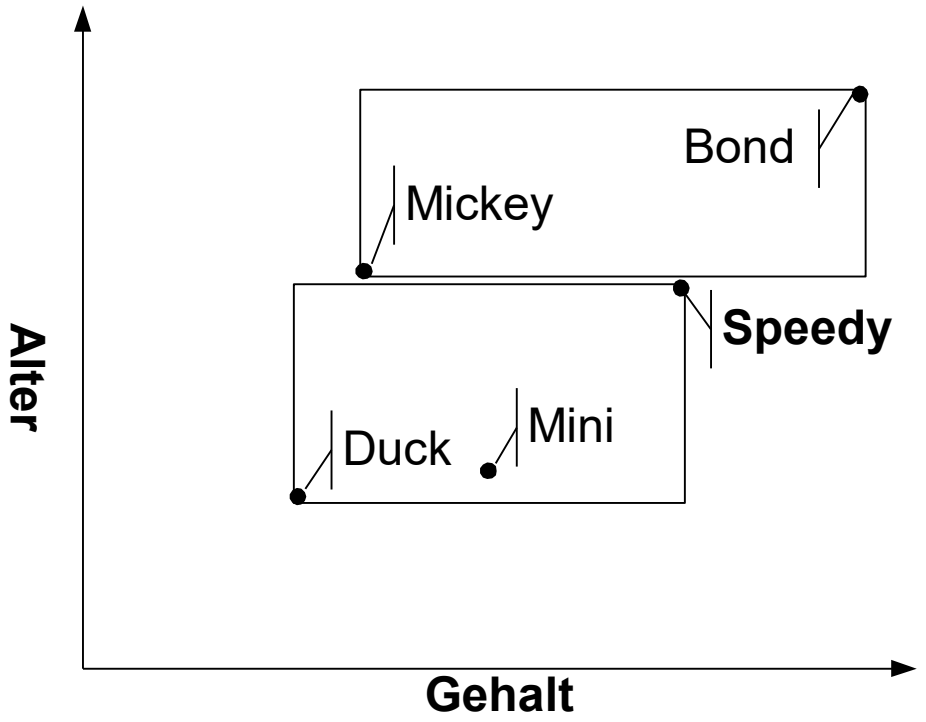


Gute versus schlechte Partitionierung

gute Partitionierung



schlechte Partitionierung

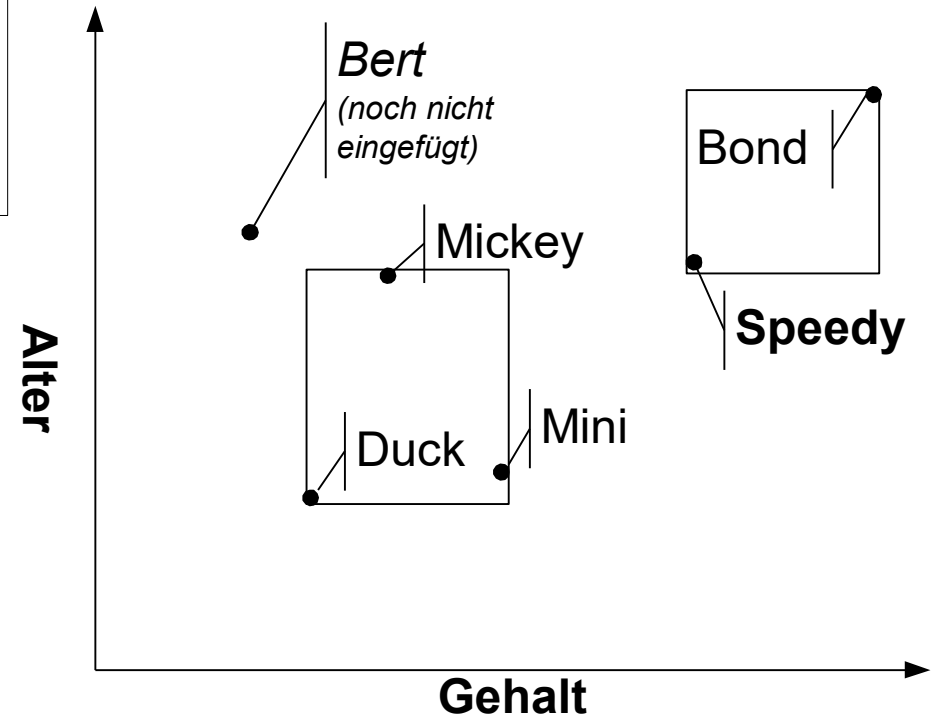


Nächste Phase in der Entstehungsgeschichte des R-Baums

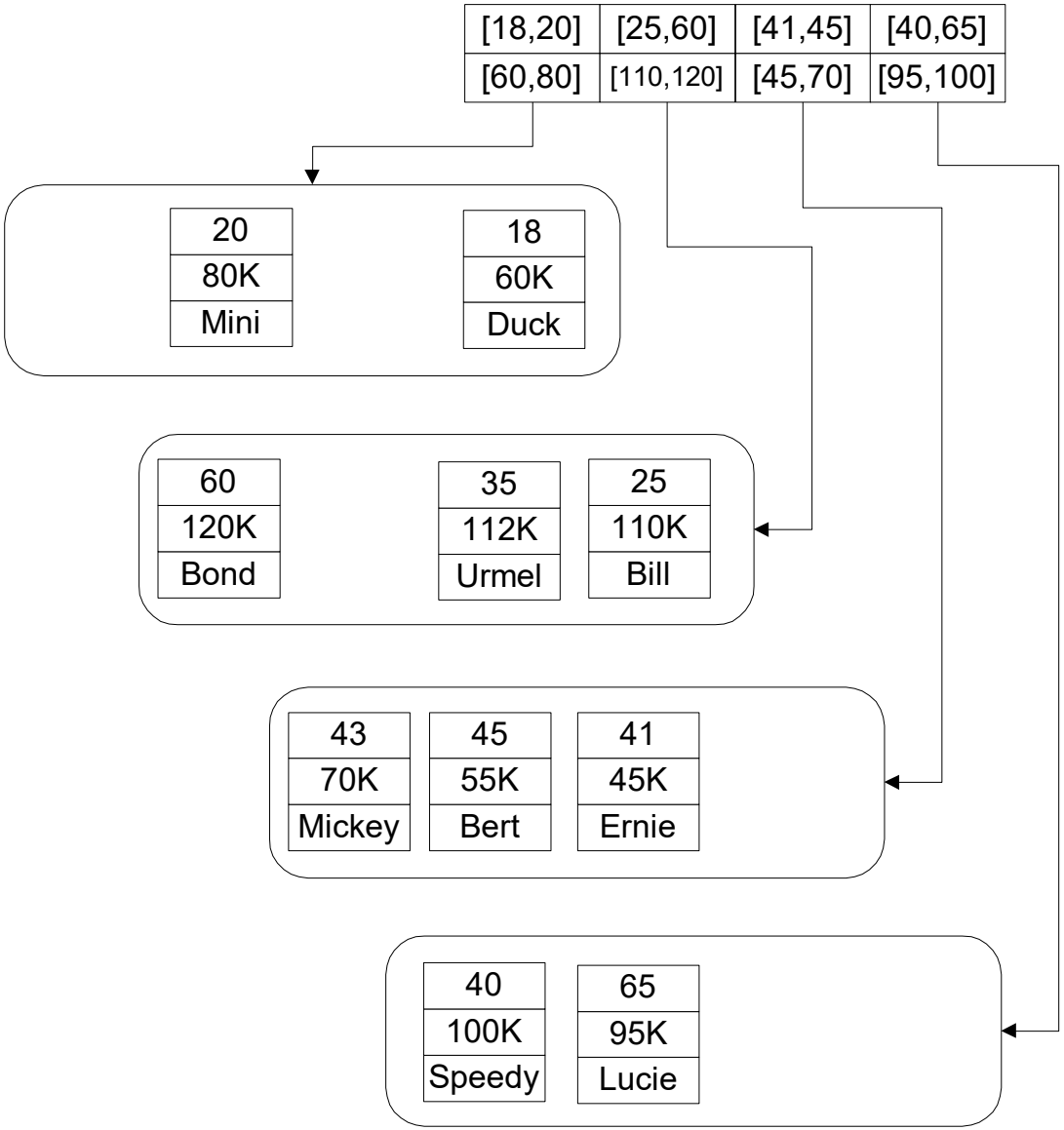
[18,43]	[40,60]		
[60,80]	[100,120]		

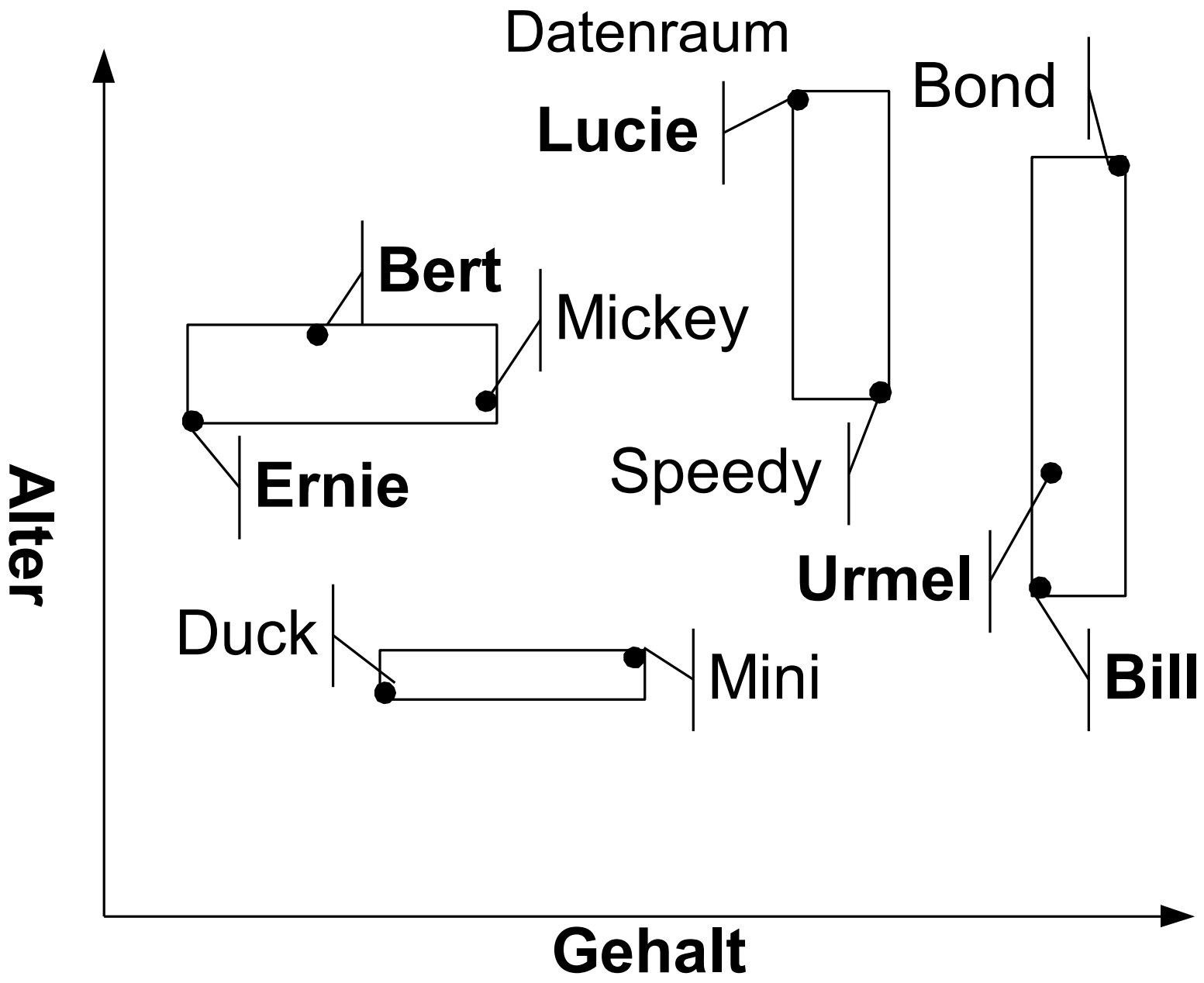
20	43	18
80K	70K	60K
Mini	Mickey	Duck

60	40
120K	100K
Bond	Speedy

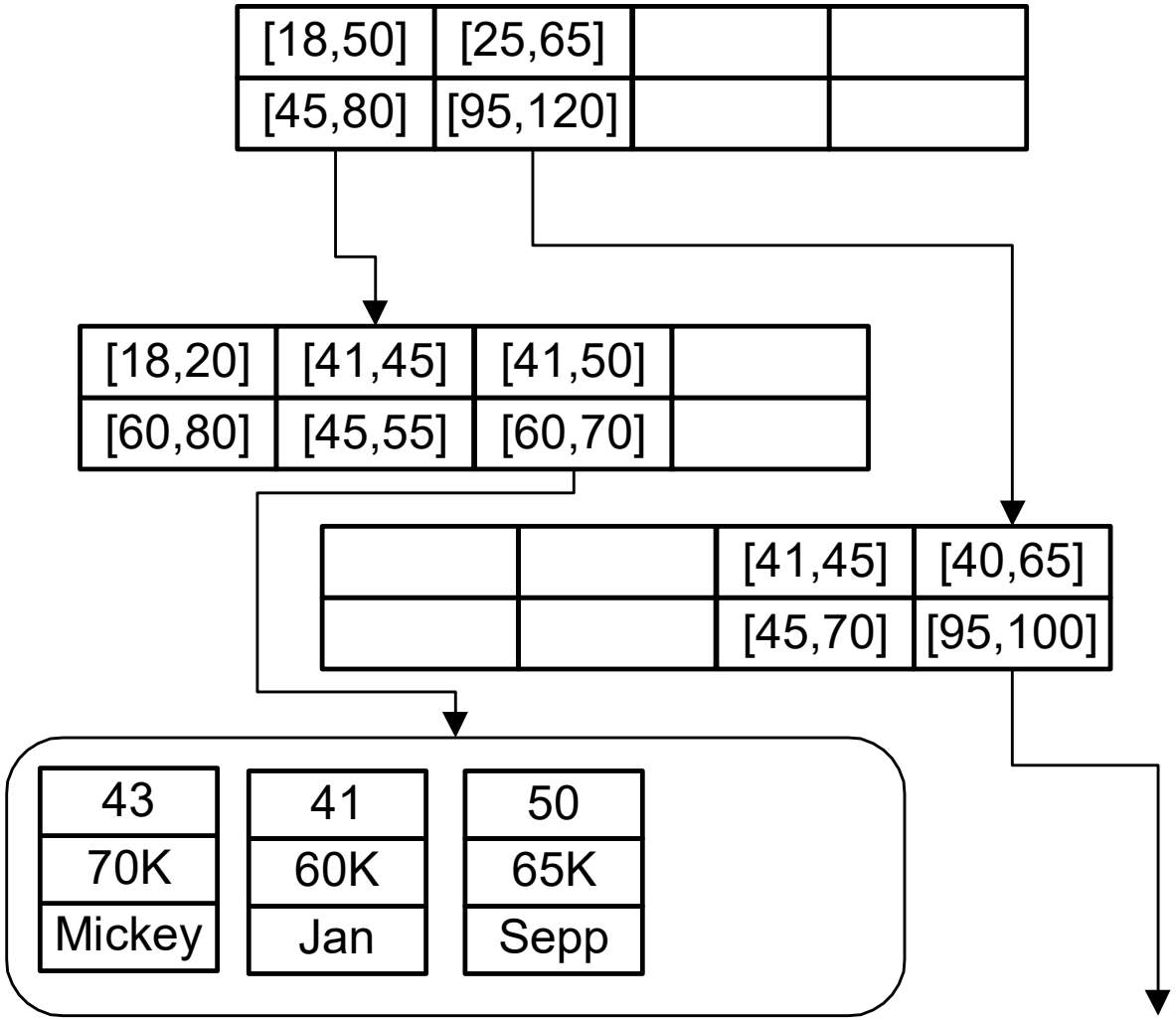


Nächste Phase



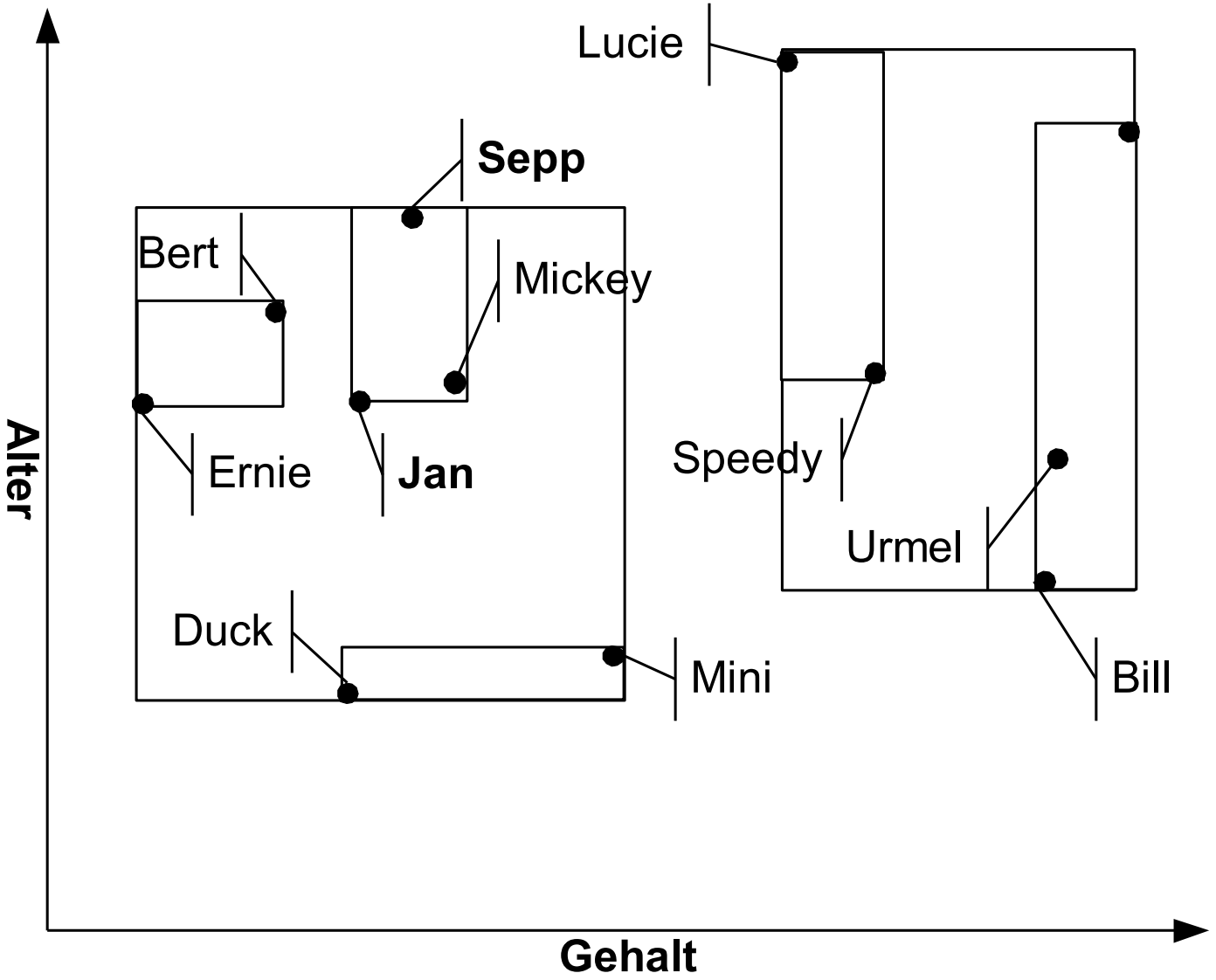

Datenraum
Lucie
Bond
Bert
Mickey
Ernie
Speedy
Urmel
Duck
Mini
Bill
Alter
Gehalt

Wachsen des Baums: nach oben – wie im B-Baum

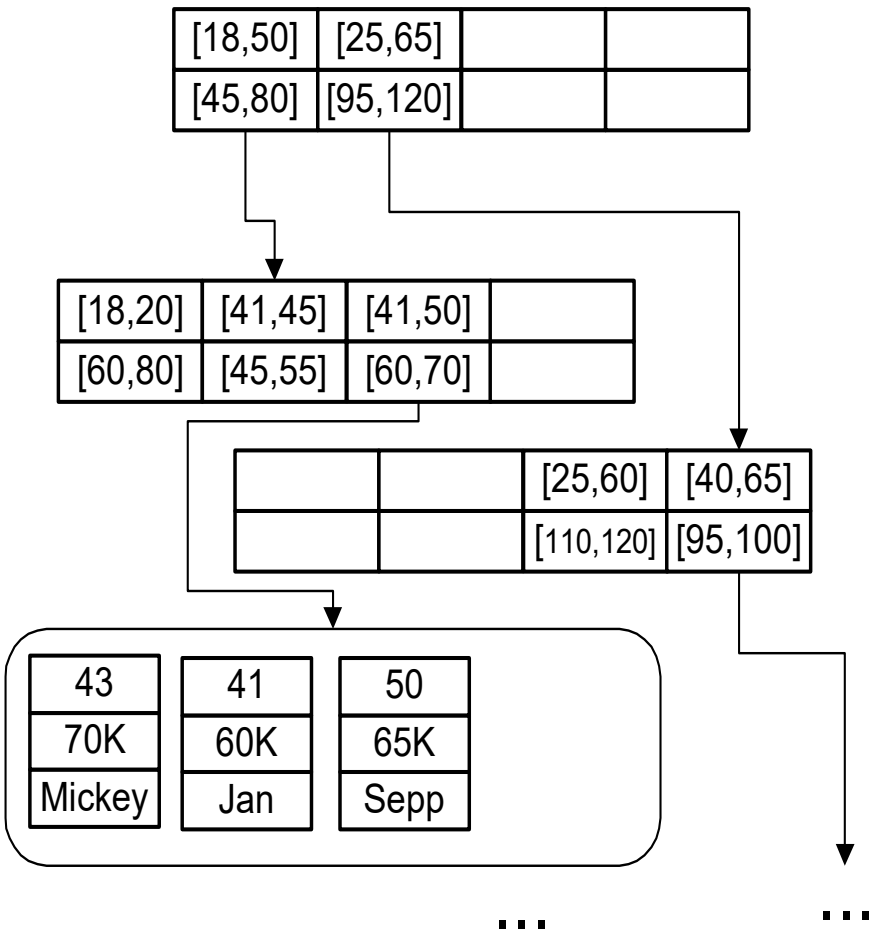


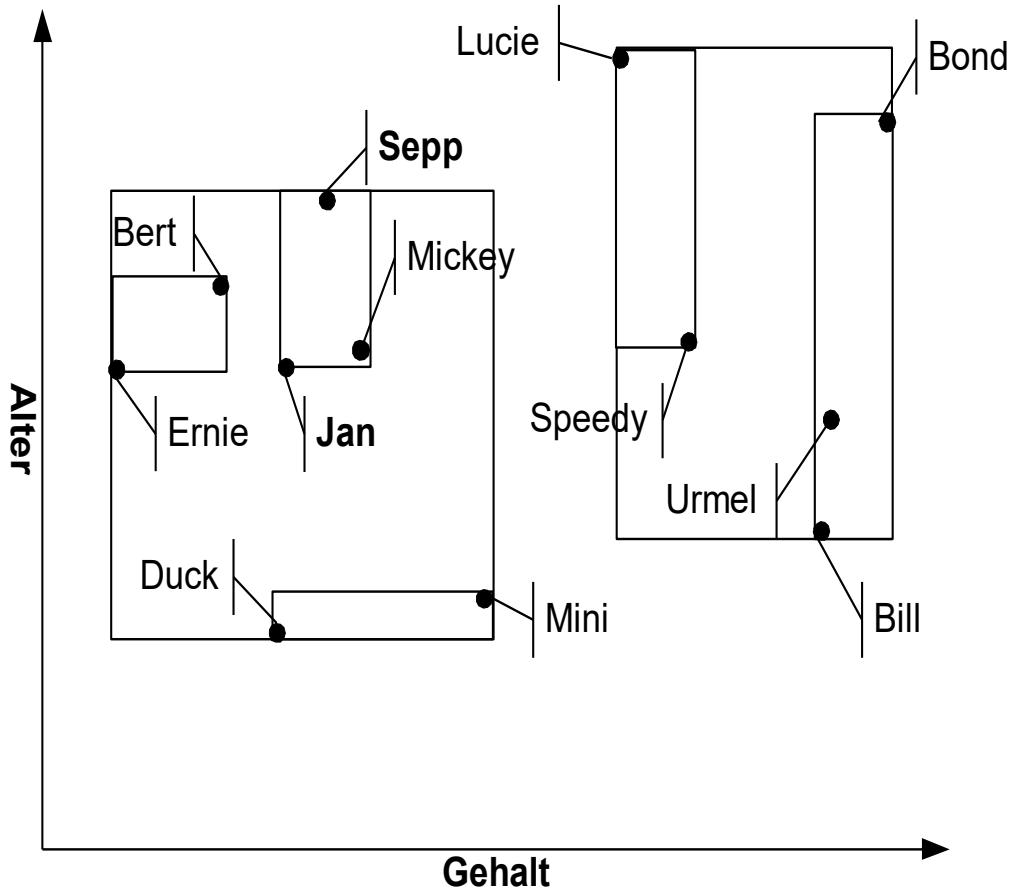
...

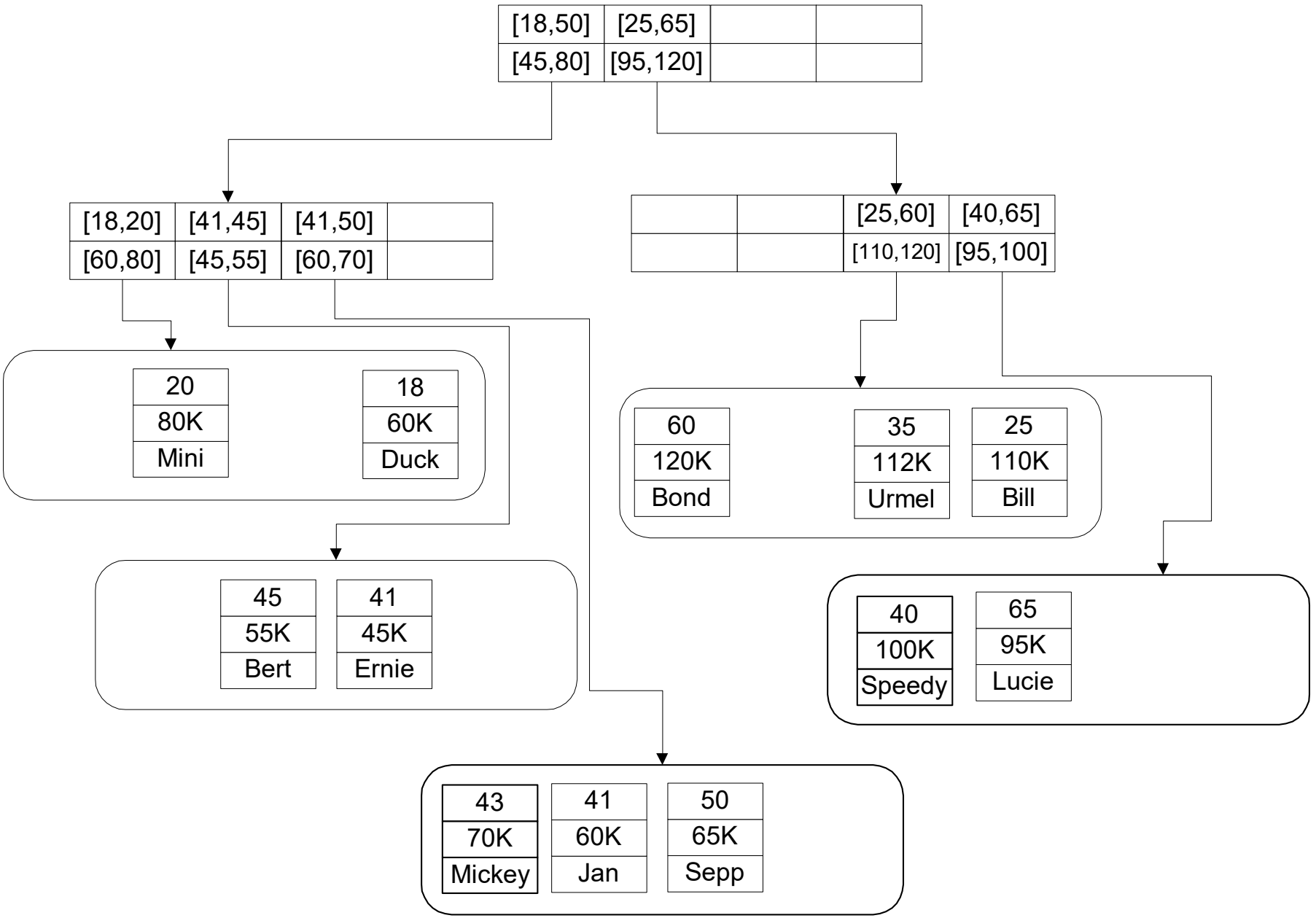
Datenraum


Lucie
Sepp
Bert
Mickey
Ernie
Jan
Speedy
Urmel
Duck
Mini
Bill
Alter
Gehalt

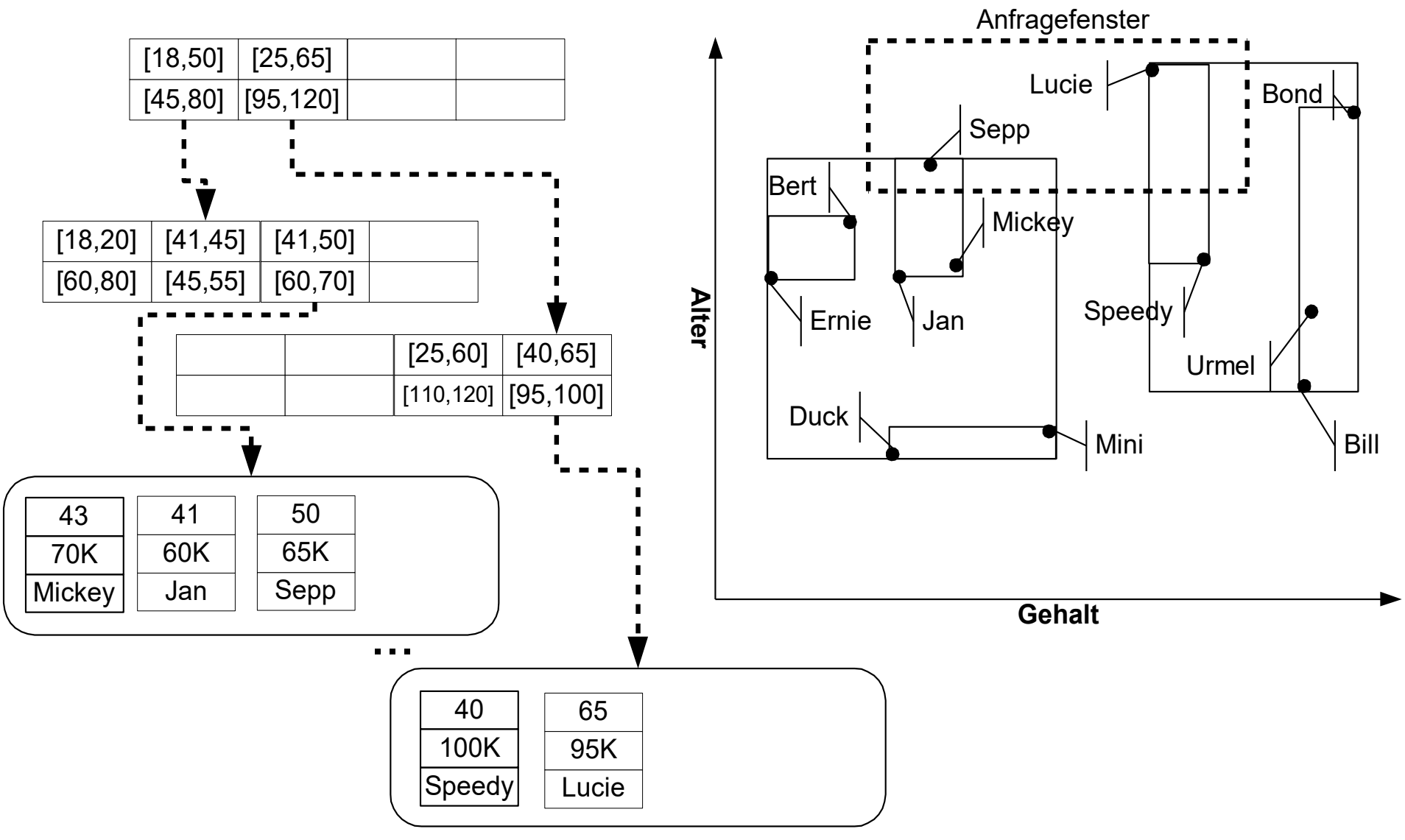
Datenraum und Speicherstruktur – Überblick

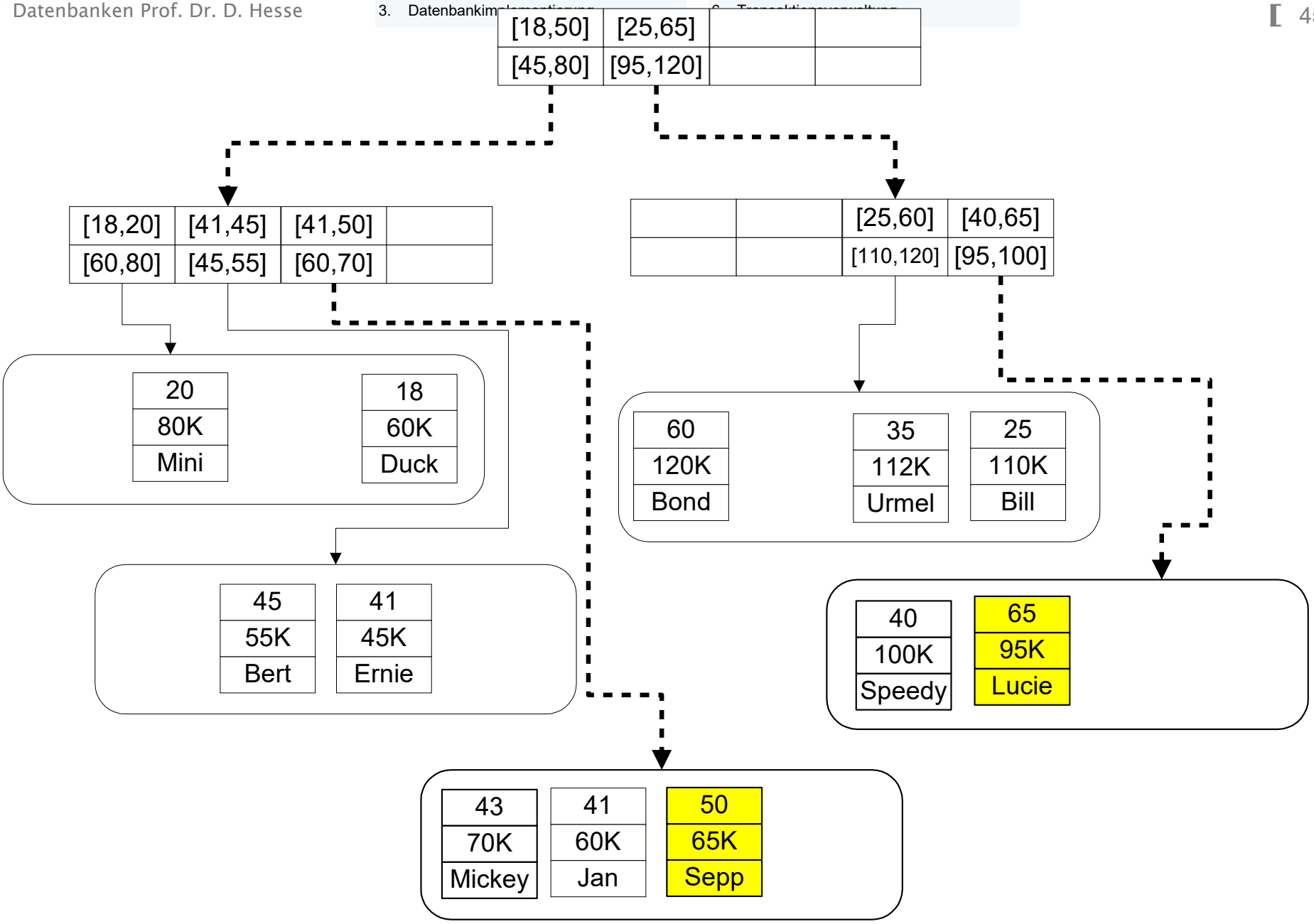




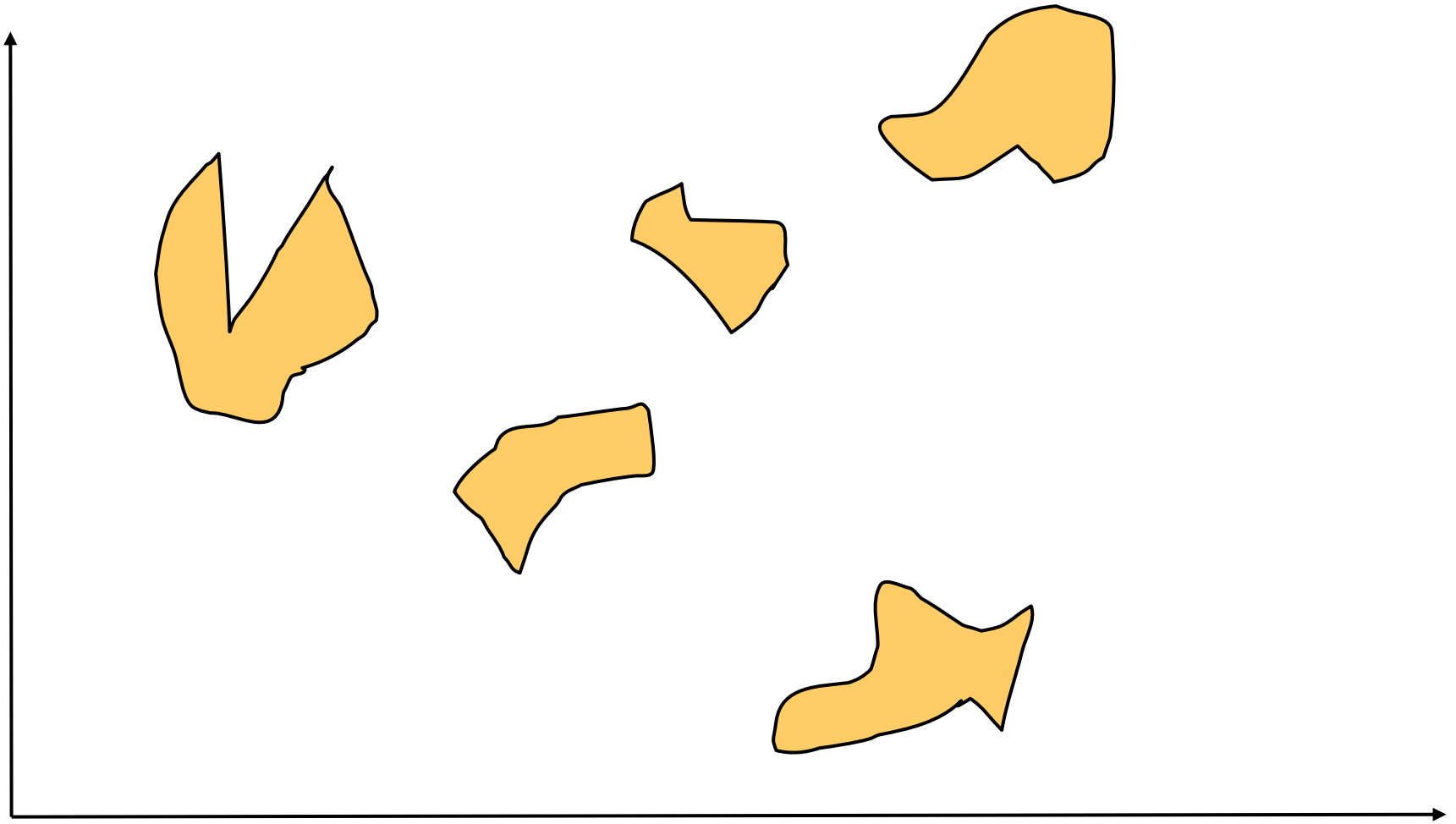

[18,50] [25,65]
[45,80] [95,120]
[18,20] [41,45] [41,50]
[60,80] [45,55] [60,70]
[25,60] [40,65]
[110,120] [95,100]
20
80K
Mini
18
60K
Duck
60
120K
Bond
35
112K
Urmel
25
110K
Bill
45
55K
Bert
41
45K
Ernie
40
100K
Speedy
65
95K
Lucie
43
70K
Mickey
41
60K
Jan
50
65K
Sepp

Bereichsanfragen auf dem R-Baum

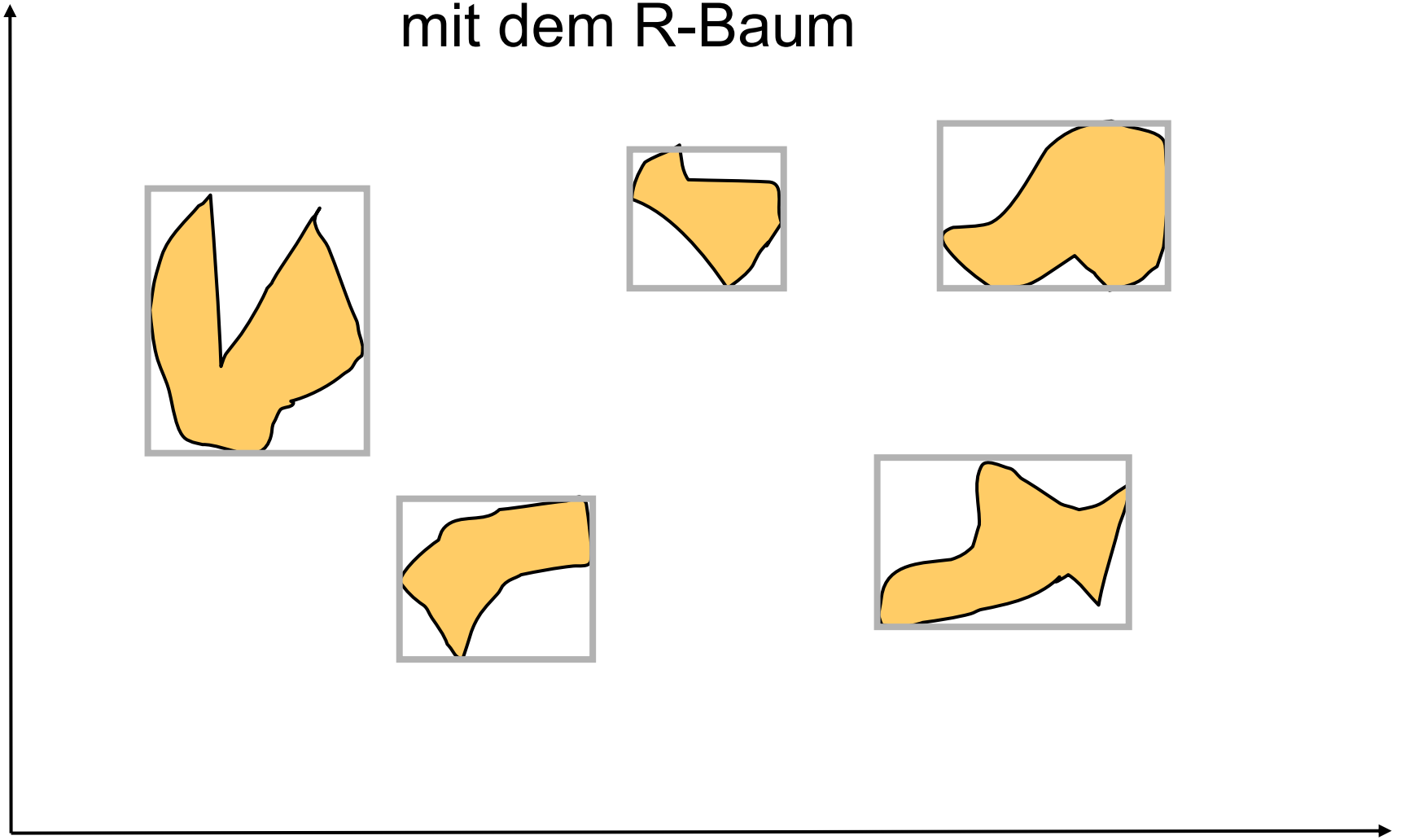



[18,50] [25,65]
[45,80] [95,120]
[18,20] [41,45] [41,50]
[60,80] [45,55] [60,70]
[25,60] [40,65]
[110,120] [95,100]
20
80K
Mini
18
60K
Duck
60
120K
Bond
35
112K
Urmel
25
110K
Bill
45
55K
Bert
41
45K
Ernie
40
100K
Speedy
65
95K
Lucie
43
70K
Mickey
41
60K
Jan
50
65K
Sepp

Indizierung räumlicher Objekte (anstatt Punkten) mit dem R-Baum



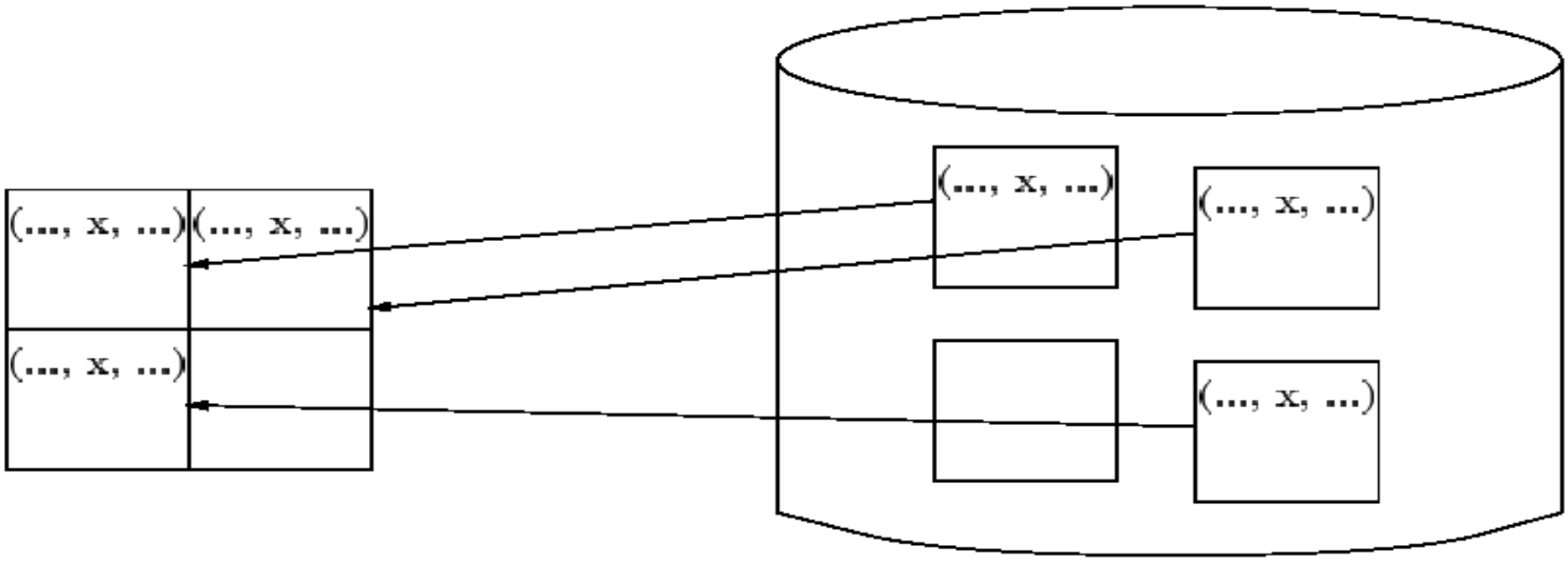
Indizierung räumlicher Objekte (anstatt Punkten) mit dem R-Baum



Indizierung räumlicher Objekte (anstatt Punkten) mit dem R-Baum

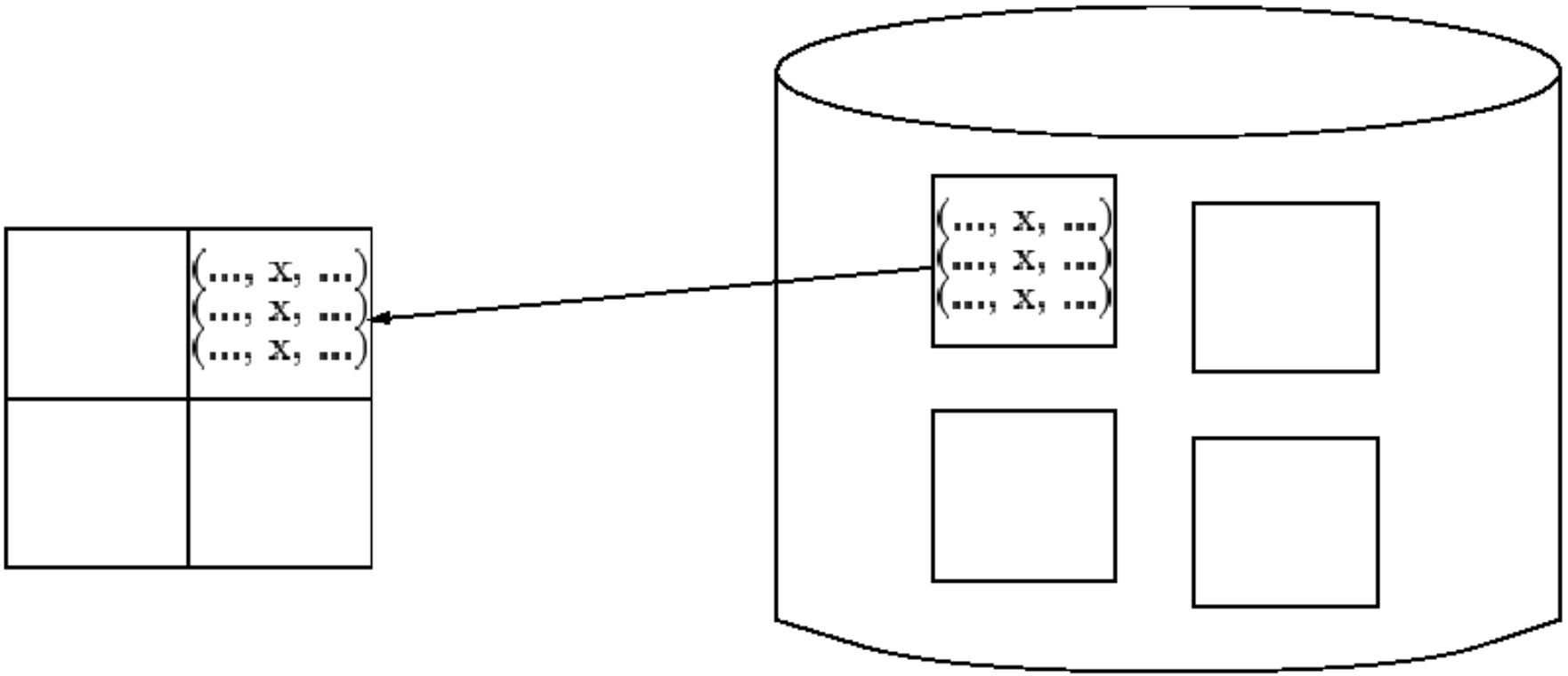
Objektballung / Clustering logisch verwandter Daten

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

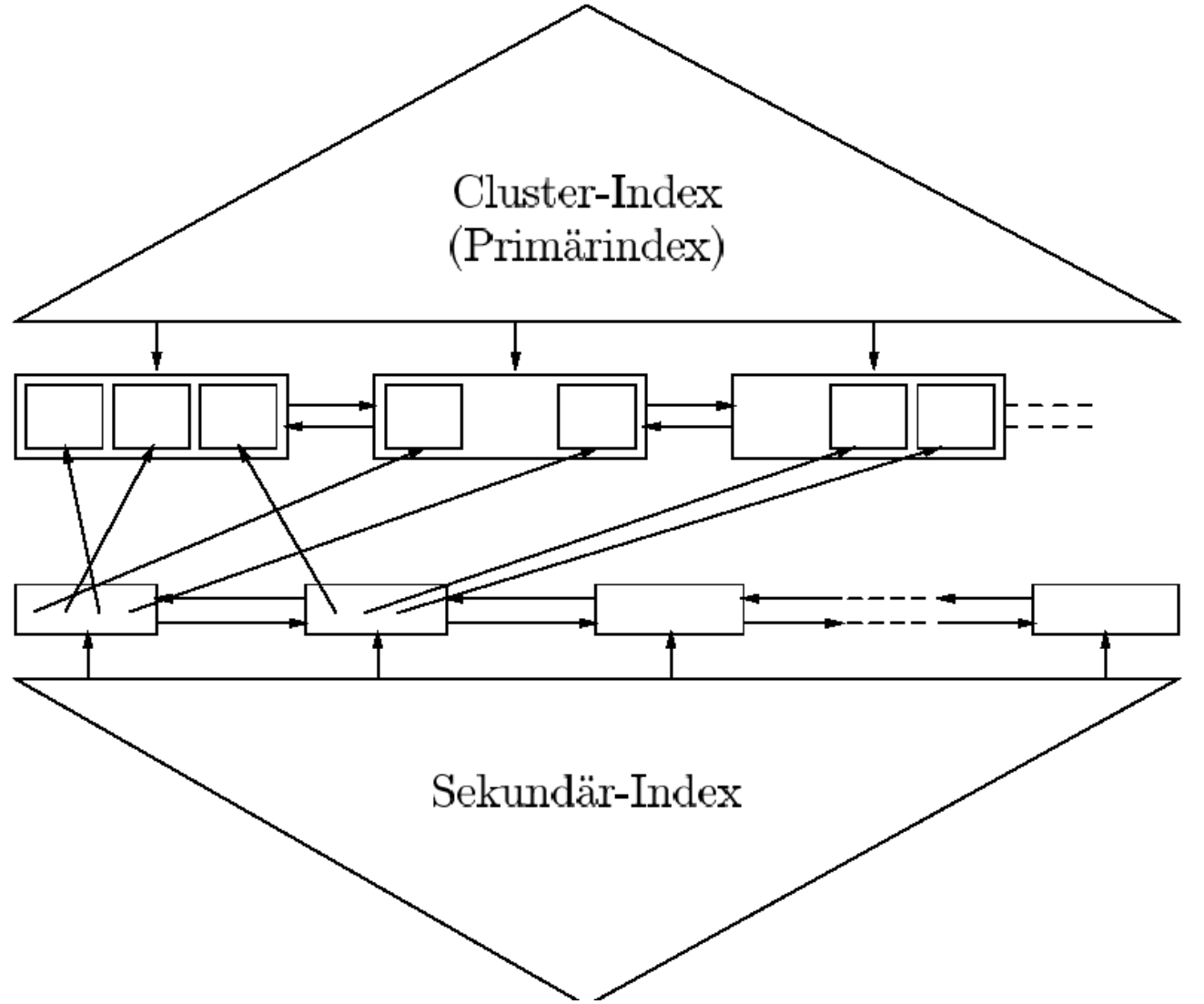


Hauptspeicher ← Zugriffslücke → Hintergrundspeicher

Hauptspeicher ← Zugriffslücke → Hintergrundspeicher



Indexe und Ballung



Seite P_i

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

Seite P_{i+1}

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

Unterstützung eines Anwendungsverhaltens

```
Select Name
From Professoren
Where PersNr = 2136
```

```
Select Name
From Professoren
Where Gehalt >= 90000 and Gehalt <= 100000
```

Indizierung in SQL

Create index SemsterInd
on Studenten
(Semester)

drop index SemsterInd