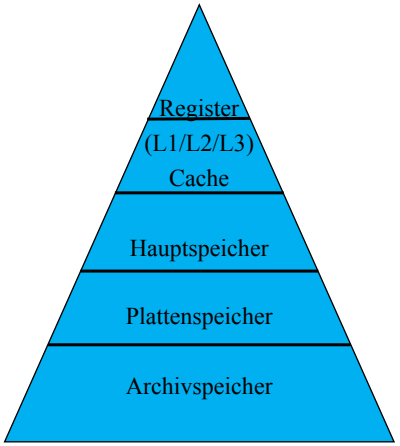
Kapitel 7

Physische Datenorganisation

- **Speicherhierarchie**
- **Hintergrundspeicher / RAID**
- **Speicherstrukturen**
- **B-Bäume**
- **Hashing**
- **R-Bäume**

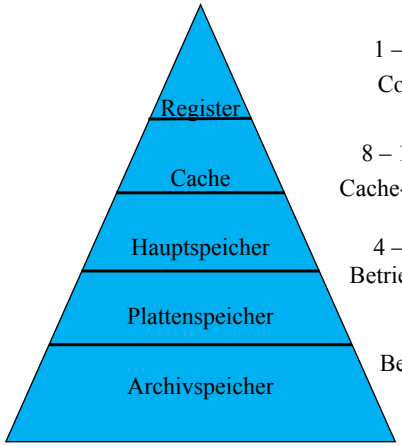
1

Überblick: Speicherhierarchie



2

Überblick: Speicherhierarchie



1 – 8 Byte
Compiler

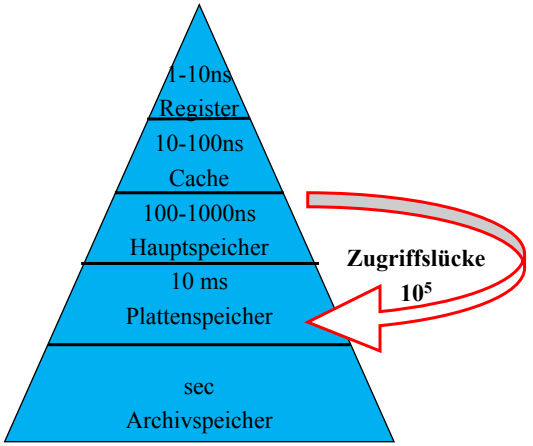
8 – 128 Byte
Cache-Controller

4 – 64 KB
Betriebssystem

Benutzer

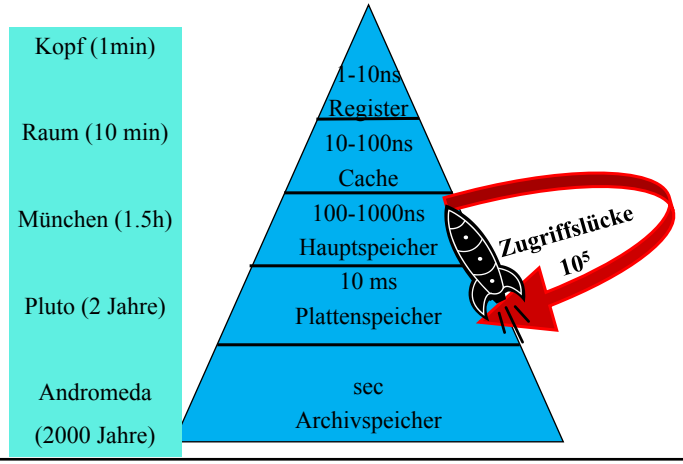
3

Überblick: Speicherhierarchie

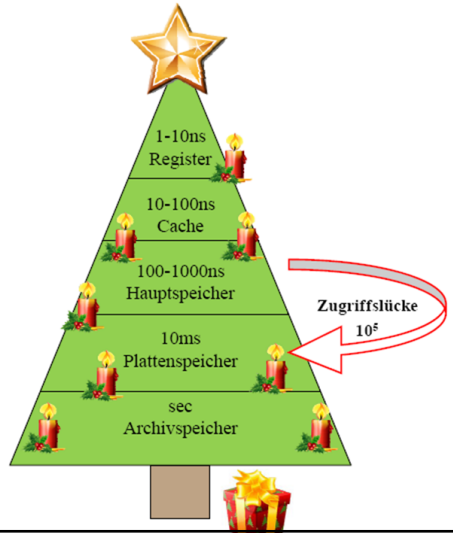


4

Überblick: Speicherhierarchie



5

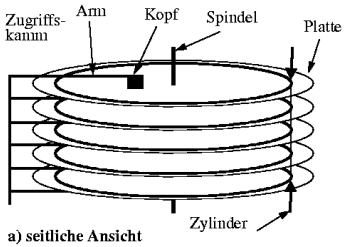


6

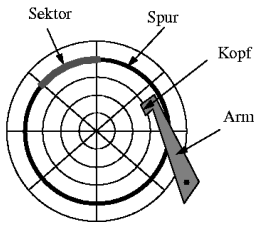
Magnetplattenspeicher

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

7

Lesen von Daten von der Platte

Seek Time: Arm positionieren
- 5ms

Latenzzeit: ½ Plattenumdrehung (im Durchschnitt)
- 10000 Umdrehungen / Minute
- ➔ Ca 3ms

Transfer von der Platte zum Hauptspeicher
- 100 Mb /s ➔ 15 MB/s

8

Random versus Chained IO

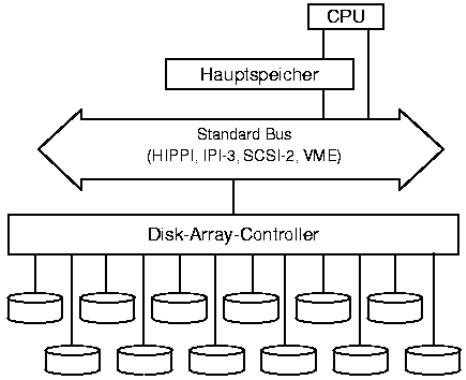
1000 Blöcke à 4KB sind zu lesen
Random I/O
- Jedesmal Arm positionieren
- Jedesmal Latenzzeit
- ➔ 1000 * (5 ms + 3 ms) + Transferzeit von 4 MB
- ➔ > 8000 ms + 300ms → 8s

Chained IO
- Einmal positionieren, dann „von der Platte kratzen“
- ➔ 5 ms + 3ms + Transferzeit von 4 MB
- ➔ 8ms + 300 ms → 1/3 s

Also ist chained IO **ein bis zwei Größenordnungen schneller** als random IO in Datenbank-Algorithmen unbedingt beachten !

9

Disk Arrays ➔ RAID-Systeme



10

Fehlertoleranz

„The Problem with Many Small Disks: Many Small Faults“

Disk-Array mit N Platten: ohne Fehlertoleranzmechanismen N-fach erhöhte Ausfallwahrscheinlichkeit
⇒ System ist unbrauchbar

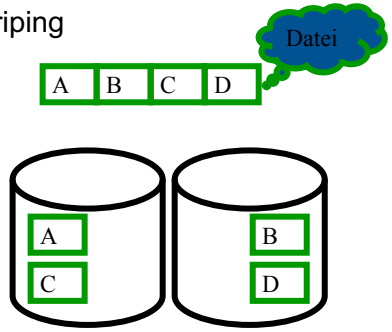
Begriffe
- Mean Time To Failure (MTTF): Erwartungswert für die Zeit (von der Inbetriebnahme) bis zum Ausfall einer Platte
- Mean Time To Repair (MTTR): Erwartungswert für die Zeit zur Ersetzung der Platte und der Rekonstruktion der Daten
- Mean Time To Data Loss (MTTDL): Erwartungswert für die Zeit bis zu einem nicht-maskierbaren Fehler

Disk-Array mit N Platten ohne Fehlertoleranzmechanismen: MTTDL = MTTF/N

Der Schlüssel zur Fehlertoleranz ist Redundanz => Redundant Arrays of Independent Disks (RAID)
- Durch Replikation der Daten (z.B. Spiegelplatten) – RAID1
- Durch zusätzlich zu den Daten gespeicherte Error-Correcting-Codes (ECCS), z.B. Paritätsbits (RAID-4, RAID-5)

11

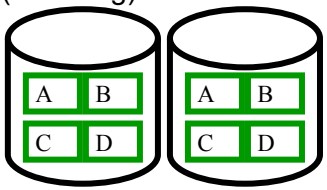
RAID 0: Striping



Lastbalancierung wenn alle Blöcke mit gleicher Häufigkeit gelesen/geschrieben werden
Doppelte Bandbreite beim sequentiellen Lesen der Datei bestehend aus den Blöcken ABCD...
Aber: Datenverlust wird immer wahrscheinlicher, je mehr Platten man verwendet (Stripingbreite = Anzahl der Platten, hier 2)

12

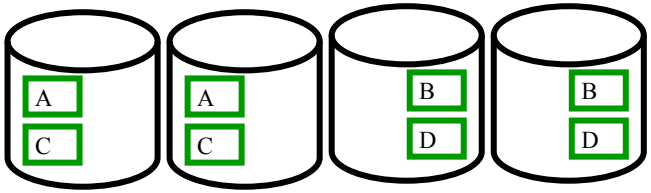
RAID 1: Spiegelung (mirroring)



Datensicherheit: durch Redundanz aller Daten (Engl. mirror)
Doppelter Speicherbedarf
Lastbalancierung beim Lesen: z.B. kann Block A von der linken oder der rechten Platte gelesen werden
Aber beim Schreiben müssen beide Kopien geschrieben werden
- Kann aber parallel geschehen
- Dauert also nicht doppelt so lange wie das Schreiben nur eines Blocks

13

RAID 0+1: Striping und Spiegelung

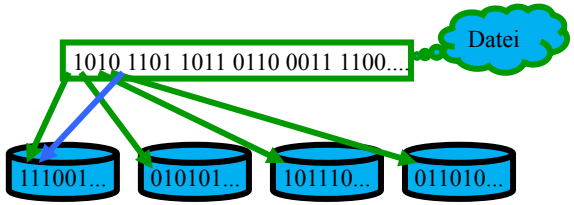


Kombiniert RAID 0 und RAID 1
Immer noch doppelter Speicherbedarf
Zusätzlich zu RAID 1 erzielt man hierbei auch eine höhere Bandbreite beim Lesen der gesamten Datei ABCD....
Wird manchmal auch als RAID 10 bezeichnet

14

RAID 2: Striping auf Bit-Ebene

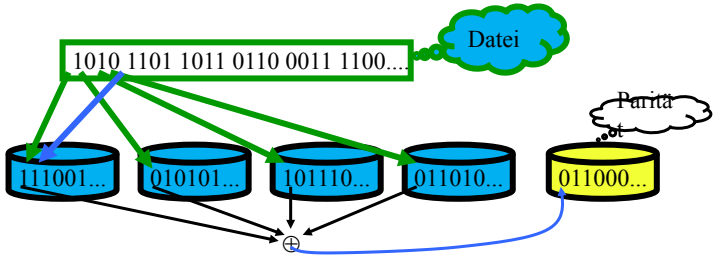
Anstatt ganzer Blöcke, wie bei RAID 0 und RAID 0+1, wird das Striping auf Bit- (oder Byte-) Ebene durchgeführt



Es werden zusätzlich auf einer Platte noch Fehlererkennungs- und Korrekturcodes gespeichert
In der Praxis nicht eingesetzt, da Platten sowieso schon Fehlererkennungscodes verwalten

15

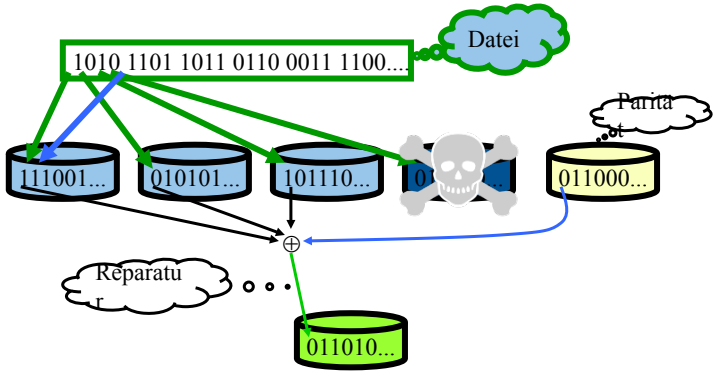
RAID 3: Striping auf Bit-Ebene, zusätzliche Platte für Paritätsinfo



Das Striping wird auf Bit- (oder Byte-) Ebene durchgeführt
Es wird auf einer Platte noch die Parität der anderen Platten gespeichert.
Parität = bit-weise xor ⊕
Dadurch ist der Ausfall einer Platte zu kompensieren
Das Lesen eines Blocks erfordert den Zugriff auf alle Platten
- Verschwendung von Schreib/Leseköpfen
- Alle marschieren synchron

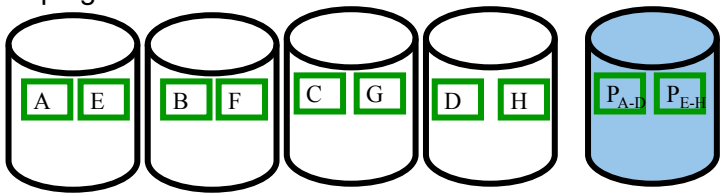
16

RAID 3: Plattenausfall



17

RAID 4: Striping von Blöcken

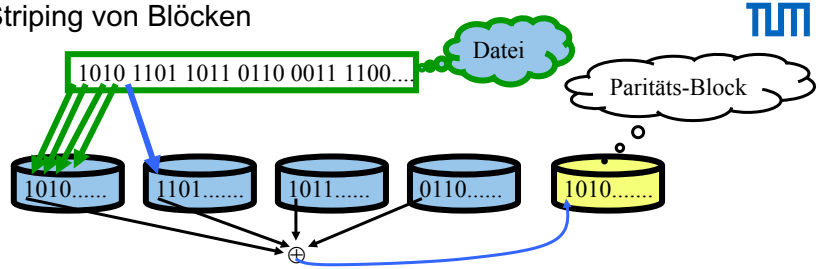


Bessere Lastbalancierung als bei RAID 3
Flaschenhals bildet die Paritätsplatte
Bei jedem Schreiben muss darauf zugegriffen werden
- Bei Modifikation von Block A zu A‘ wird die Parität P_{A-D} wie folgt neu berechnet:
- $P'_{A-D} := P_{A-D} \oplus A \oplus A'$

D.h. bei einer Änderung von Block A muss der alte Zustand von A und der alte Paritätsblock gelesen werden und der neue Paritätsblock und der neue Block A‘ geschrieben werden

18

RAID 4: Striping von Blöcken

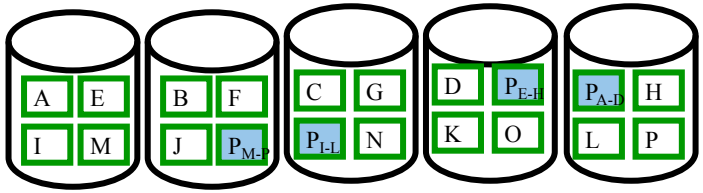


Flaschenhals bildet die Paritätsplatte
Bei jedem Schreiben muss darauf zugegriffen werden
- Bei Modifikation von Block A zu A‘ wird die Parität P_{A-D} wie folgt neu berechnet:
- $P'_{A-D} := P_{A-D} \oplus A \oplus A'$

D.h. bei einer Änderung von Block A muss der alte Zustand von A und der alte Paritätsblock gelesen werden und der neue Paritätsblock und der neue Block A‘ geschrieben werden

19

RAID 5: Striping von Blöcken, Verteilung der Paritätsblöcke



Bessere Lastbalancierung als bei RAID 4
die Paritätsplatte bildet jetzt keinen Flaschenhals mehr
Wird in der Praxis häufig eingesetzt
Guter Ausgleich zwischen Platzbedarf und Leistungsfähigkeit

20

Bewertung der Parallelität bei RAID

RAID 0
- ?

RAID 1
- ?

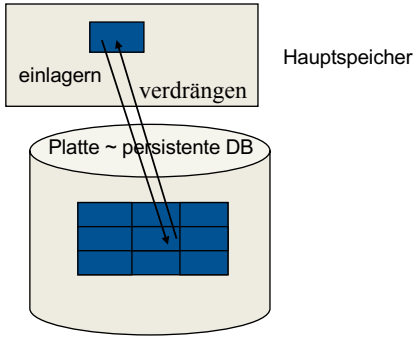
RAID 0+1
- ?

RAID 3
- ?

RAID 4
- ?

RAID 5
- ?

Systempuffer-Verwaltung

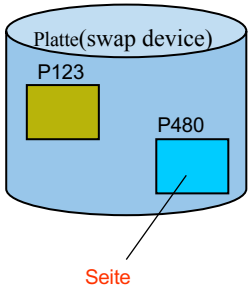


Ein- und Auslagern von Seiten

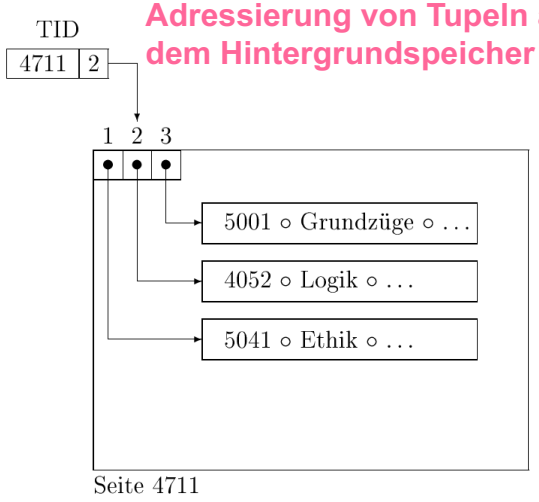
- Systempuffer ist in Seitenrahmen gleicher Größe aufgeteilt
- Ein Rahmen kann eine Seite aufnehmen
- „Überzählige" Seiten werden auf die Platte ausgelagert

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

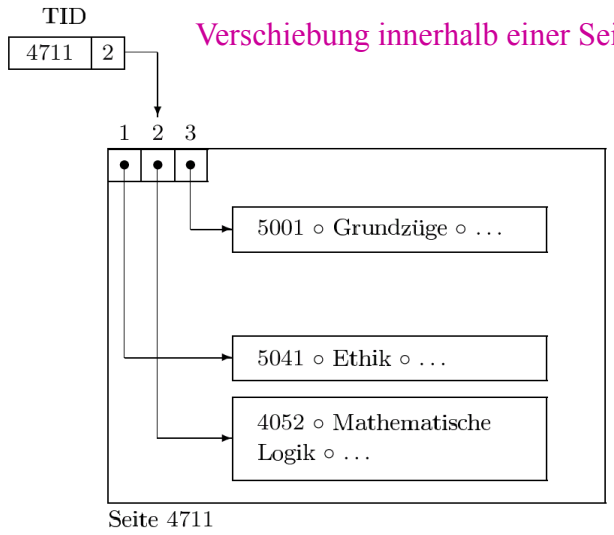
Seitenrahmen



Adressierung von Tupeln auf dem Hintergrundspeicher

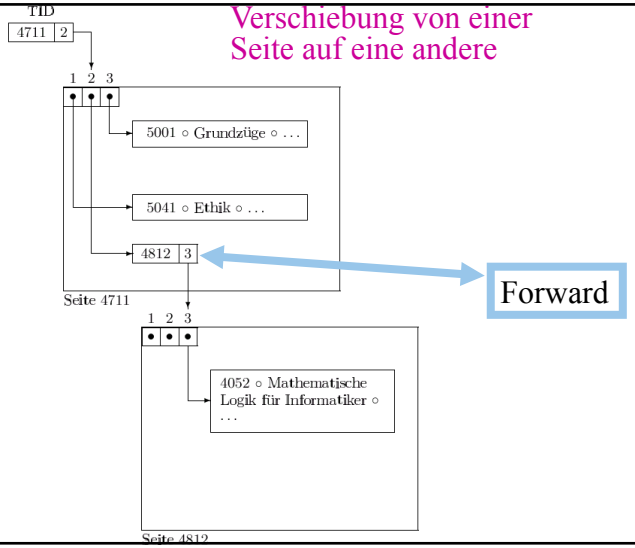


Verschiebung innerhalb einer Seite



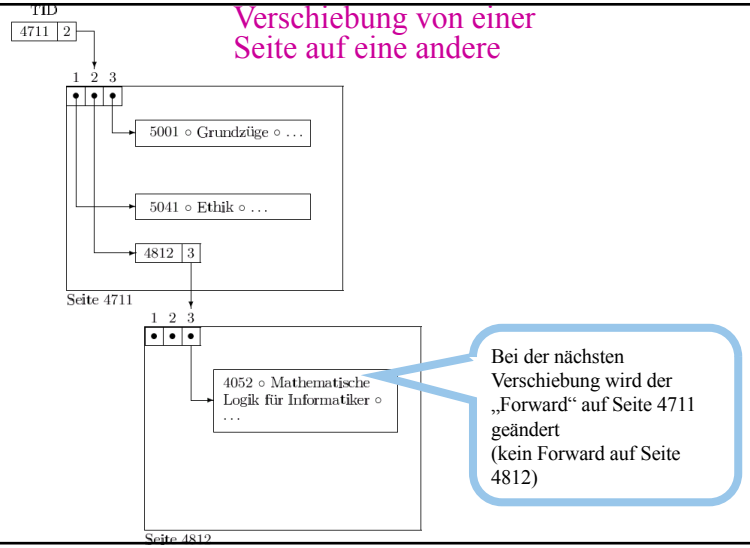
25

Verschiebung von einer Seite auf eine andere



26

Verschiebung von einer Seite auf eine andere



27

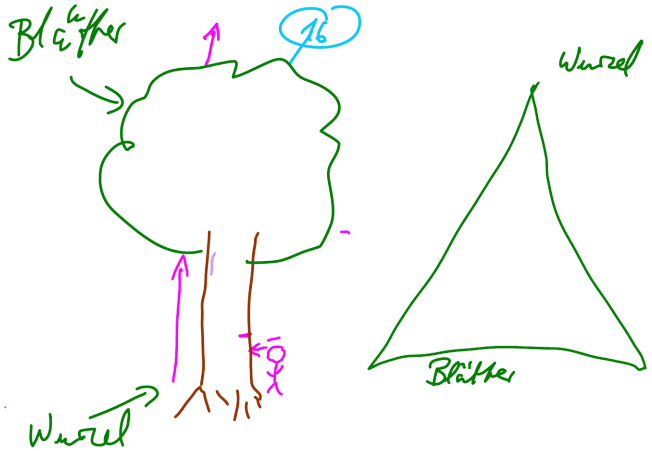
B-Bäume

Balancierte Mehrwege-Suchbäume

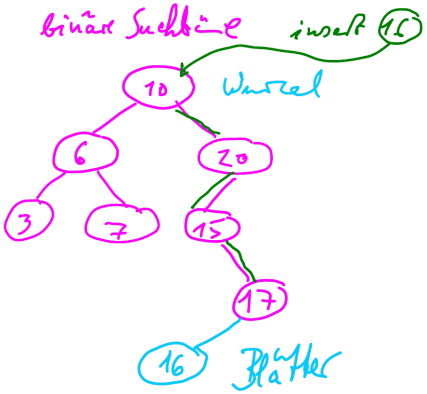
Für den Hintergrundspeicher

Erfunden von Rudolph Bayer (TUM Professor Emeritus)

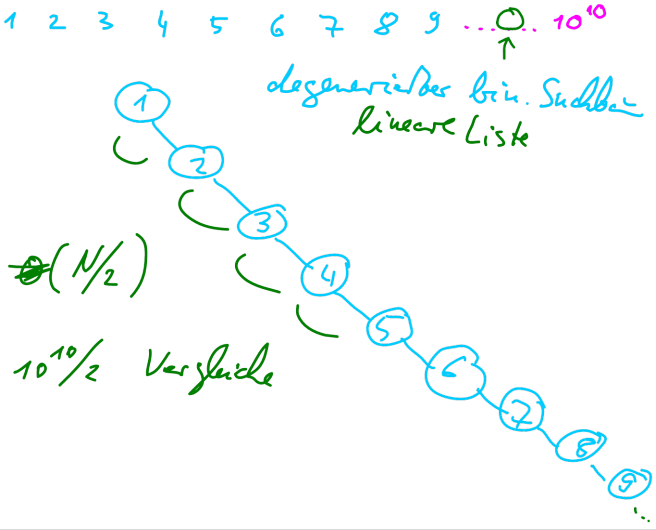
28



29



30

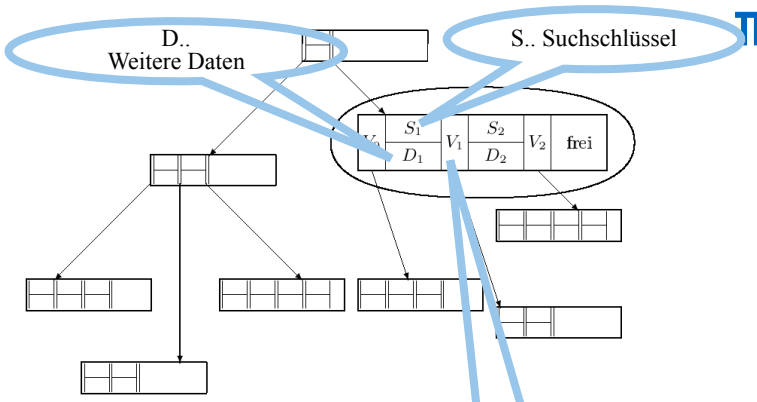


31

(Handschriftliche) Demo über Binäre Suchbäume

32

(Handschriftliche) Demo über Binäre Suchbäume

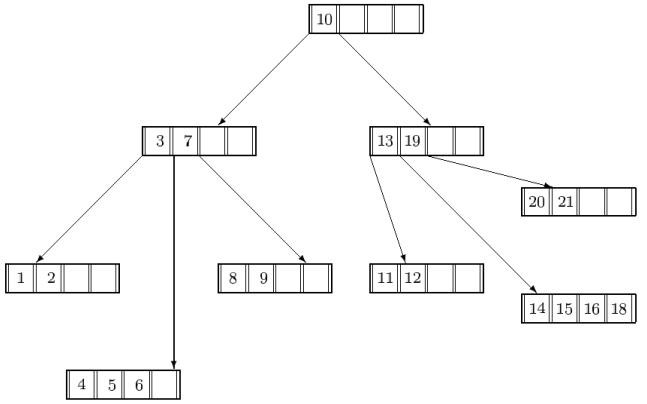


- ein Knoten entspricht einer Seite
- balanciert
- garantierte Auslastung $\geq 50\%$

B-Baum von Grad k:

1. Jeder Weg von der Wurzel zu einem Blatt hat die gleiche Länge.
2. Jeder Knoten außer der Wurzel hat mindestens k und höchstens $2k$ Einträge. Die Wurzel hat höchstens $2k$ Einträge. Die Einträge werden in allen Knoten sortiert gehalten.
3. Alle Knoten mit n Einträgen, außer den Blättern, haben $n+1$ Kinder.
4. Seien $S_1, \ldots, S_n$ die Schlüssel eines Knotens mit $n+1$ Kindern. $V_0, V_1, \ldots, V_n$ seien die Verweise auf diese Kinder. Dann gilt:
   (a) V_0 weist auf den Teilbaum mit Schlüsseln kleiner als S_1.
   (b) V_i $(i = 1, \ldots, n-1)$ weist auf den Teilbaum, dessen Schlüssel zwischen S_i und S_{i+1} liegen.
   (c) V_n weist auf den Teilbaum mit Schlüsseln größer als S_n.
   (d) In den Blattknoten sind die Zeiger nicht definiert.

Ein Beispielbaum

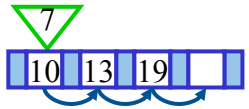


Einfügen eines neuen Objekts (Datensatz) in einen B-Baum

1. Führe eine Suche nach dem Schlüssel durch; diese endet (scheitert) an der Einfügestelle.
2. Füge den Schlüssel dort ein.
3. Ist der Knoten überfüllt, teile ihn
   - Lege einen neuen Knoten an und belege ihn mit den Schlüsseln, die rechts vom mittleren Eintrag des überfüllten Knotens liegen.
   - Füge den mittleren Eintrag im Vaterknoten des überfüllten Knotens ein.
   - Verbinde den Verweis rechts des neuen Eintrags im Vaterknoten mit dem neuen Knoten
4. Ist der Vaterknoten jetzt überfüllt?
   - Handelt es sich um die Wurzel, so lege eine neue Wurzel an.
   - Wiederhole Schritt 3 mit dem Vaterknoten.

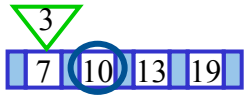
37

Sukzessiver Aufbau eines B-Baums vom Grad k=2



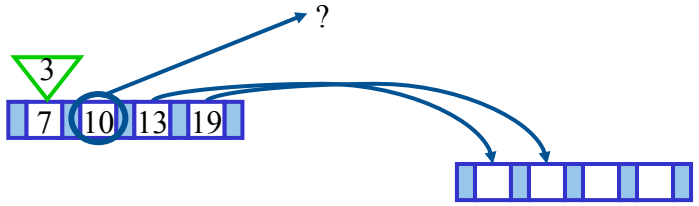
38

Sukzessiver Aufbau eines B-Baums vom Grad k=2



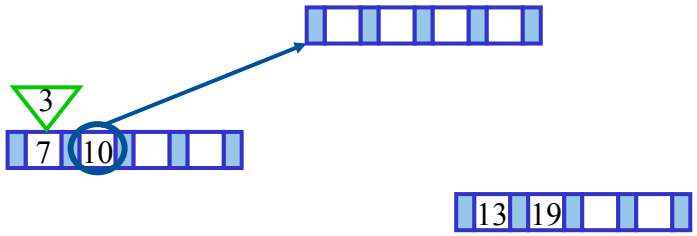
39

Sukzessiver Aufbau eines B-Baums vom Grad k=2



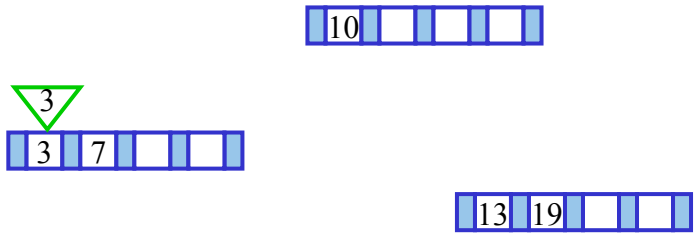
40

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



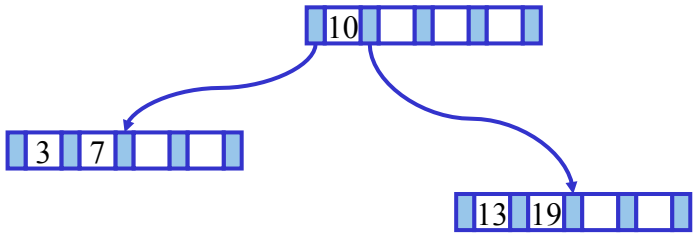
41

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



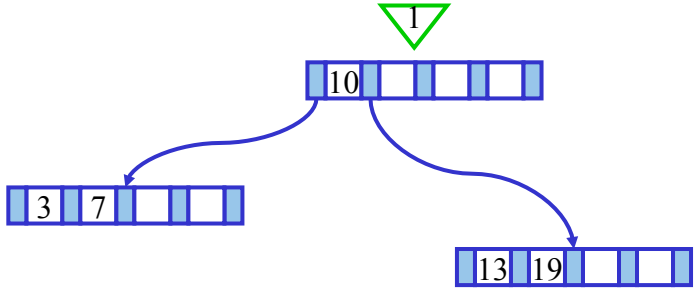
42

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



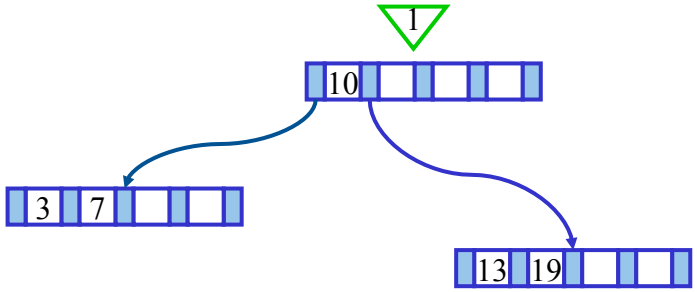
43

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



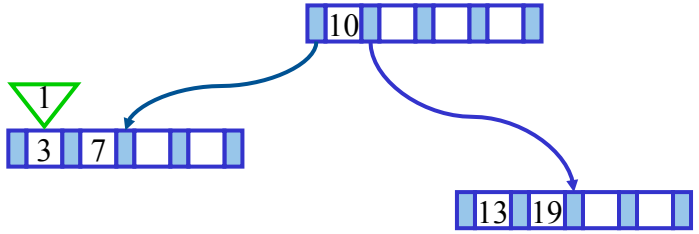
44

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



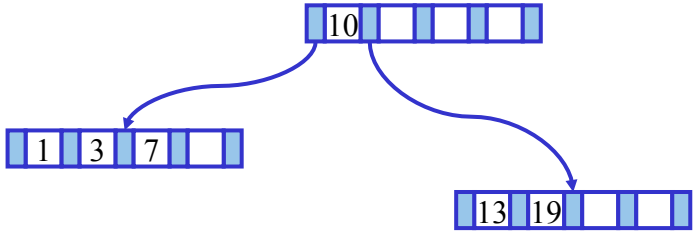
45

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



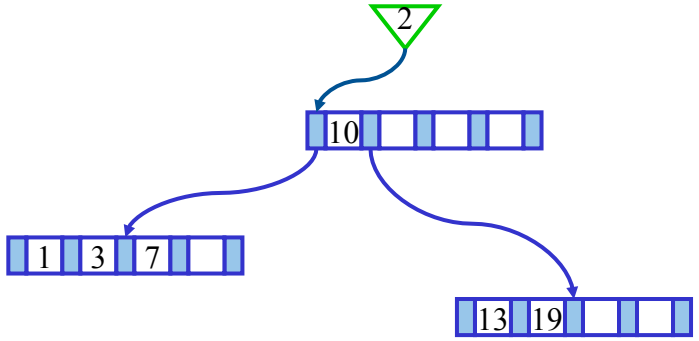
46

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



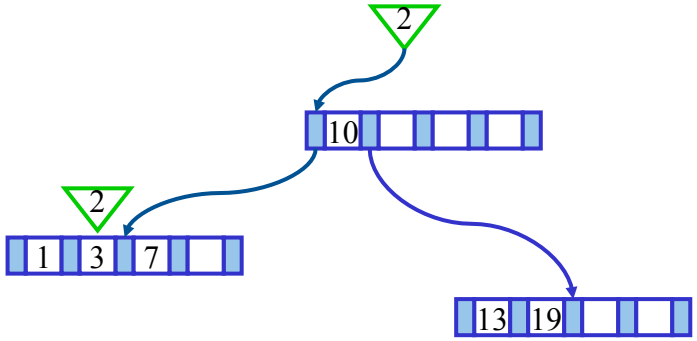
47

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



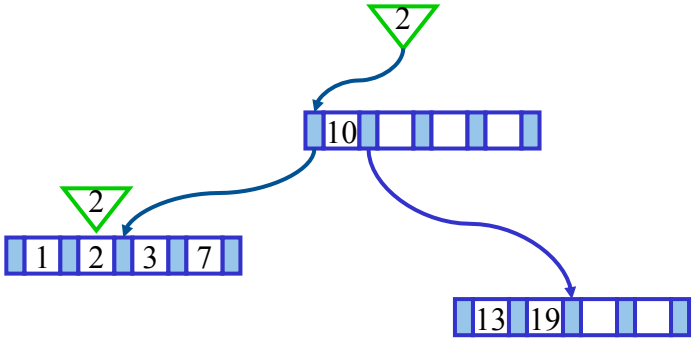
48

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



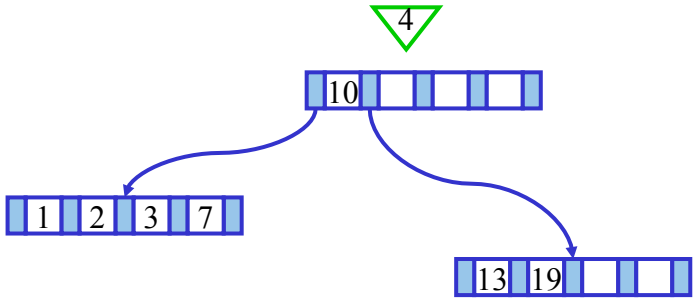
49

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



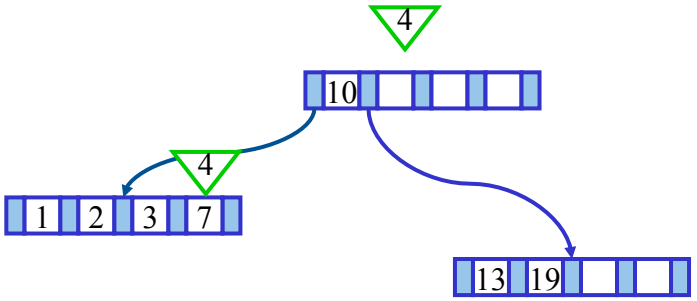
50

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



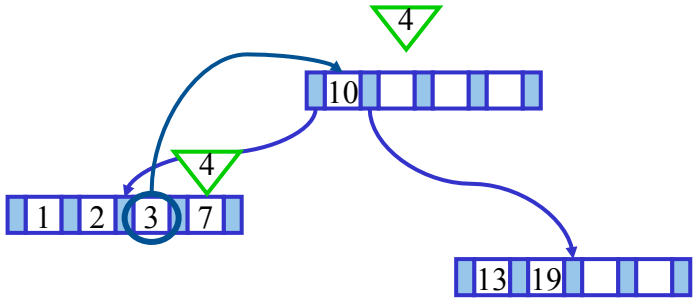
51

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



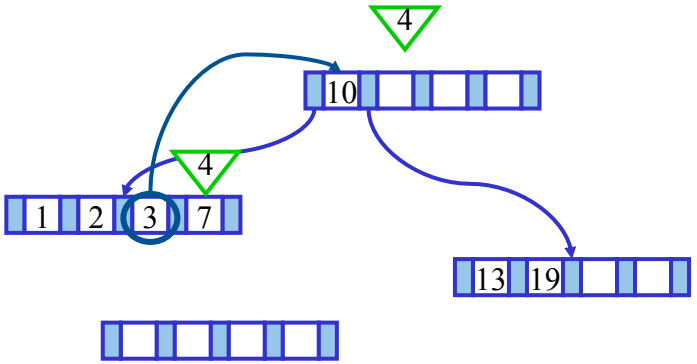
52

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



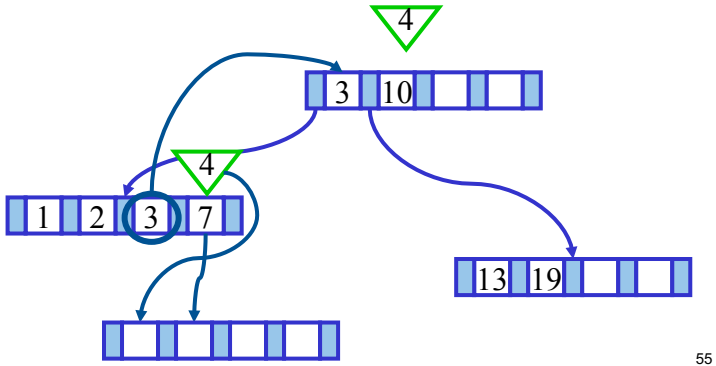
53

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



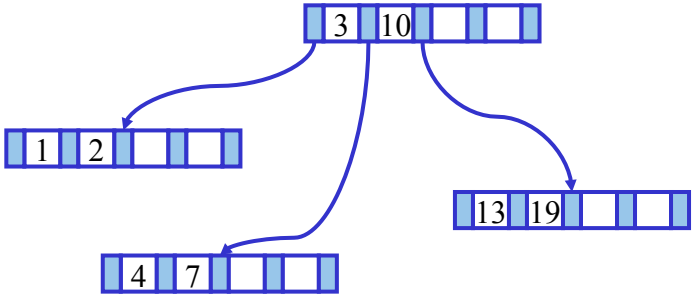
54

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



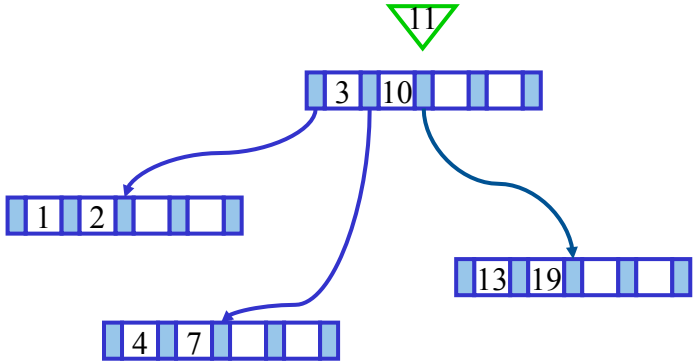
55

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



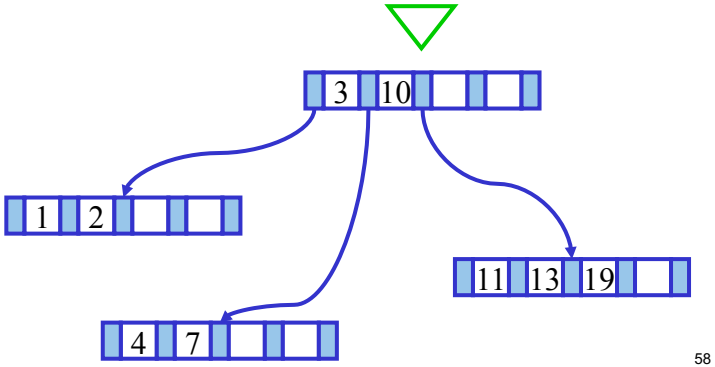
56

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



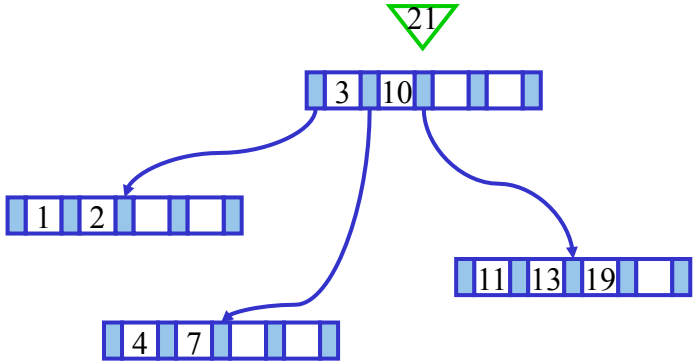
57

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



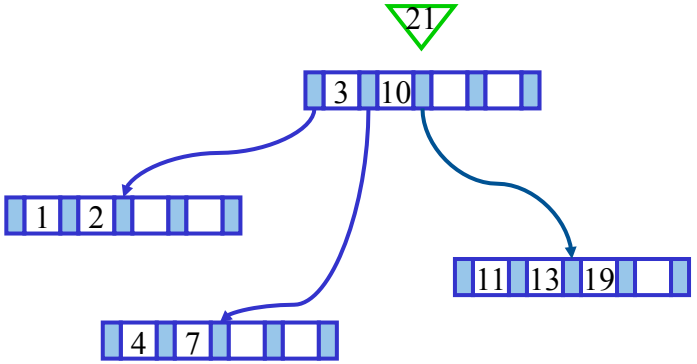
58

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



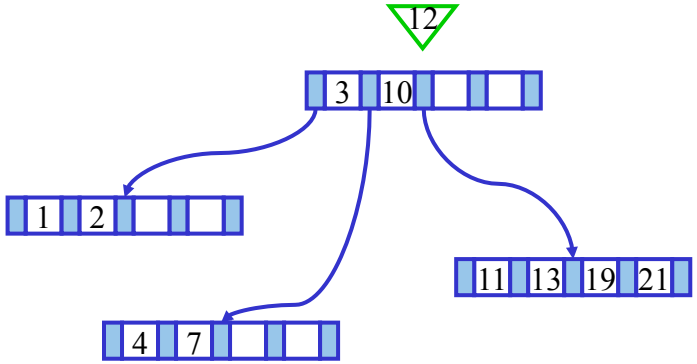
59

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



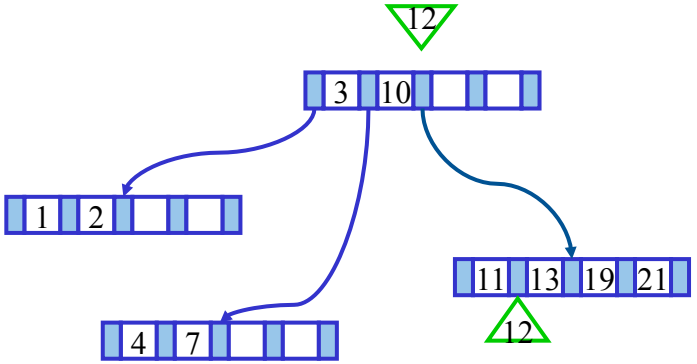
60

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



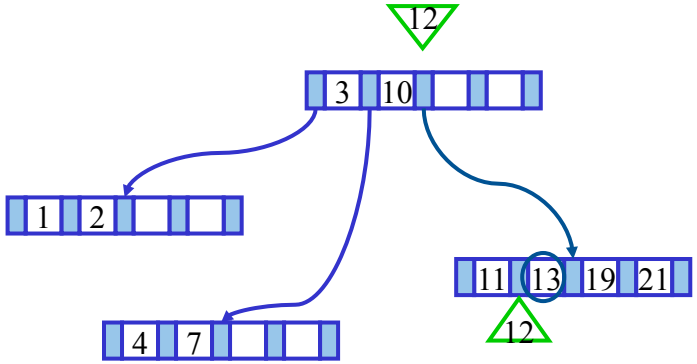
61

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



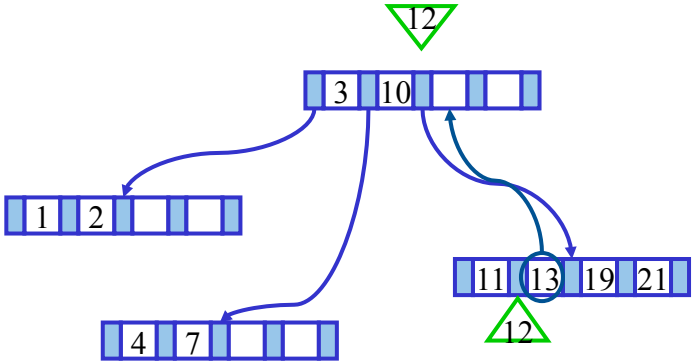
62

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



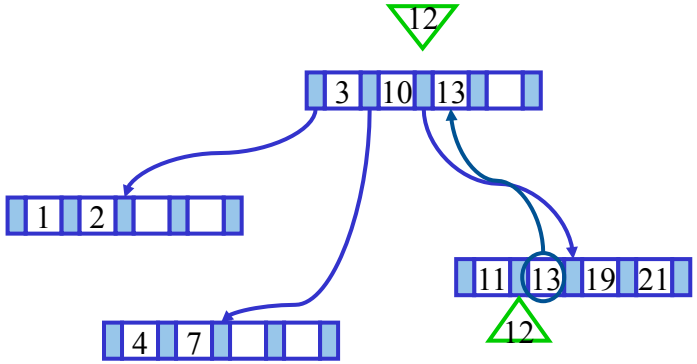
63

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



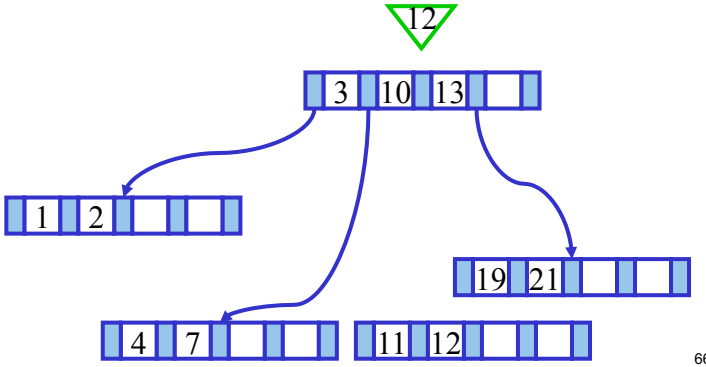
64

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



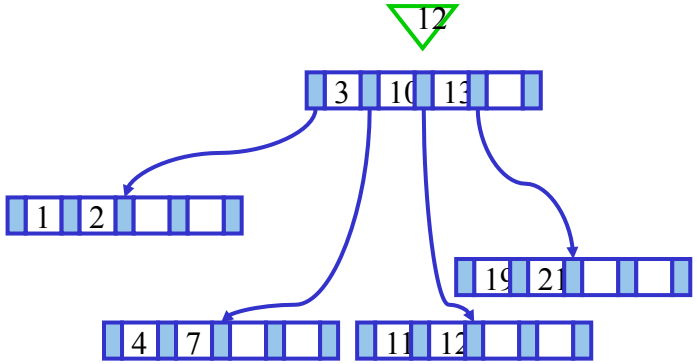
65

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



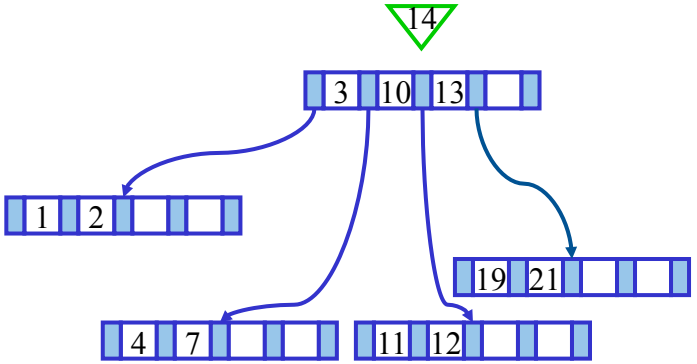
66

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



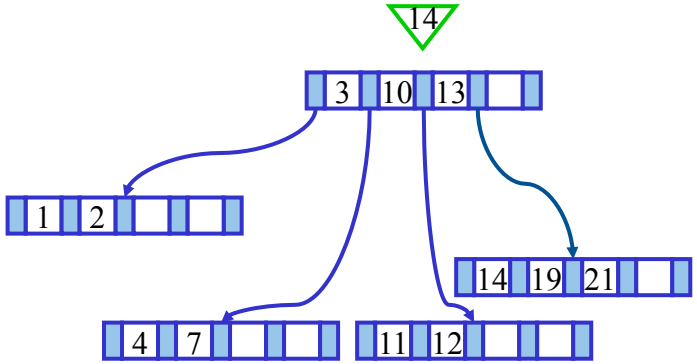
67

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



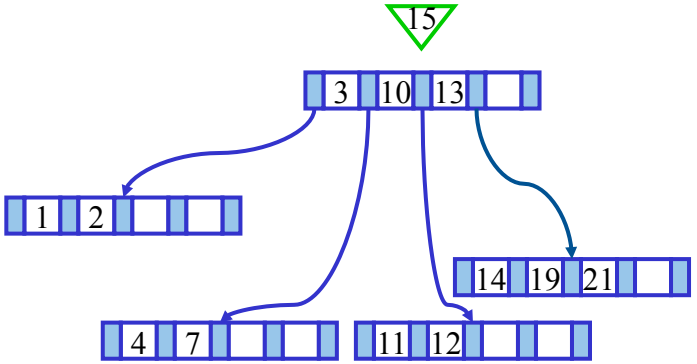
68

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



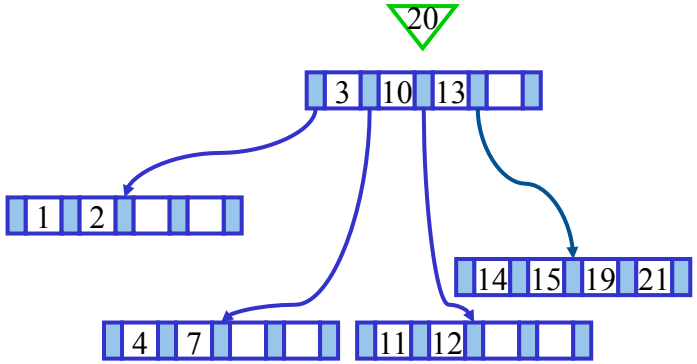
69

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



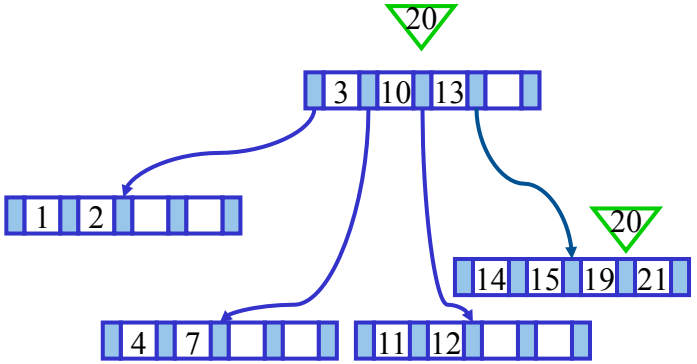
70

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



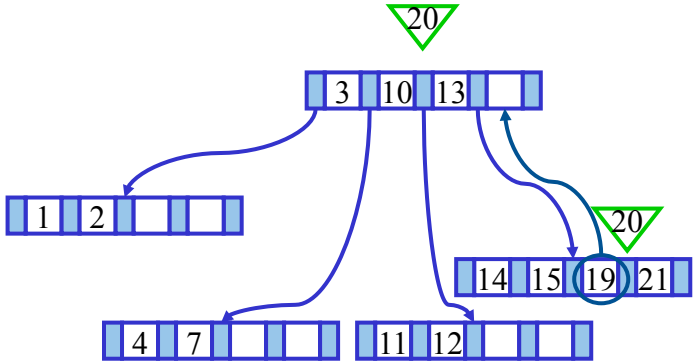
71

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



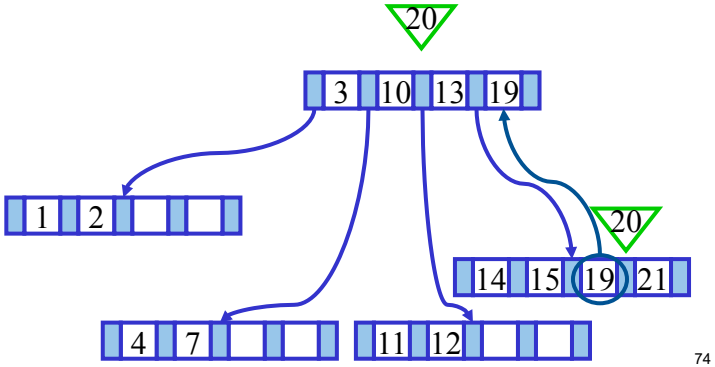
72

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



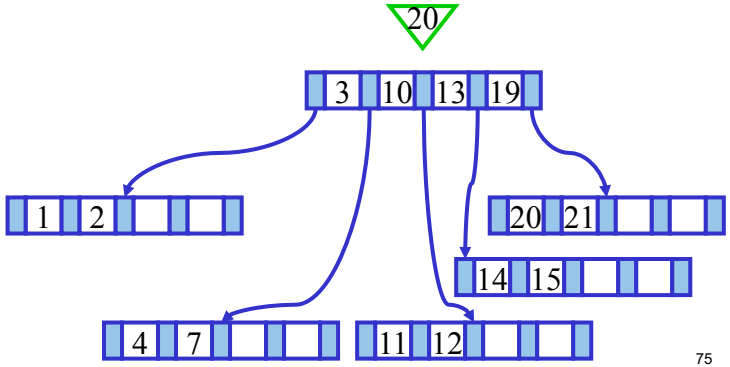
73

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



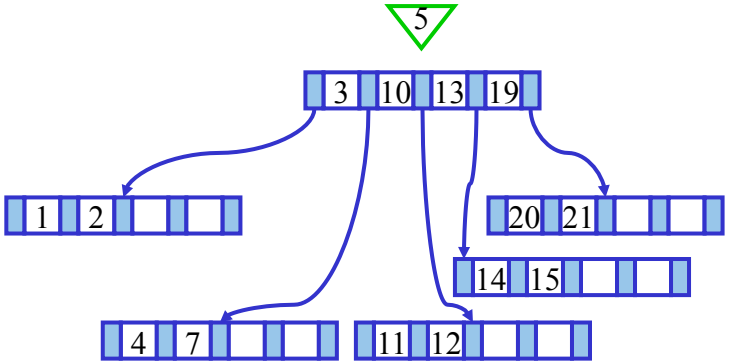
74

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



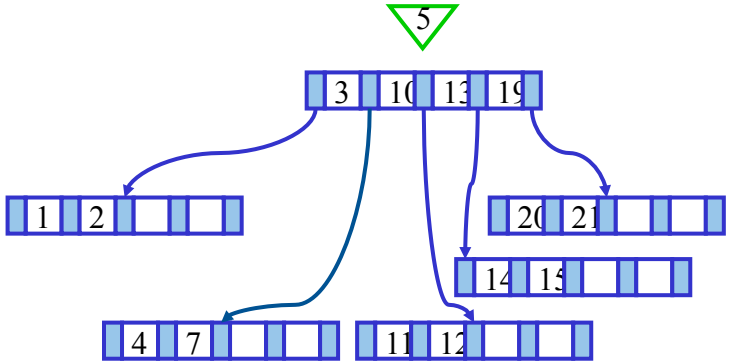
75

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



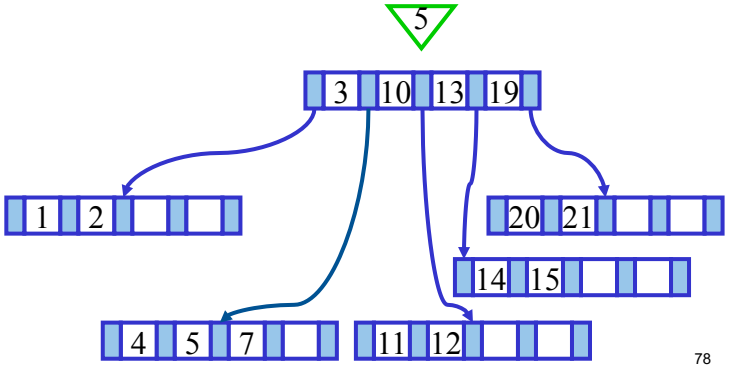
76

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



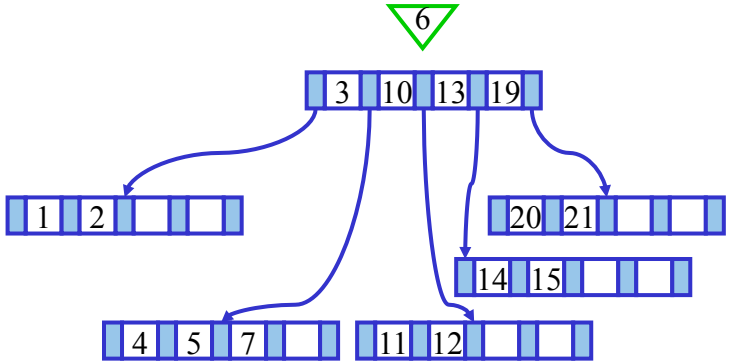
77

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



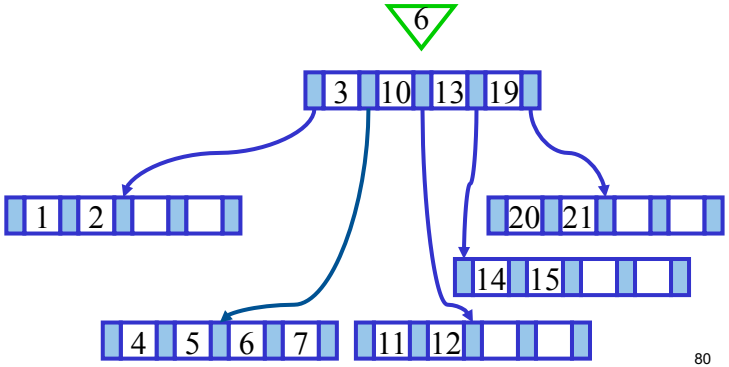
78

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



79

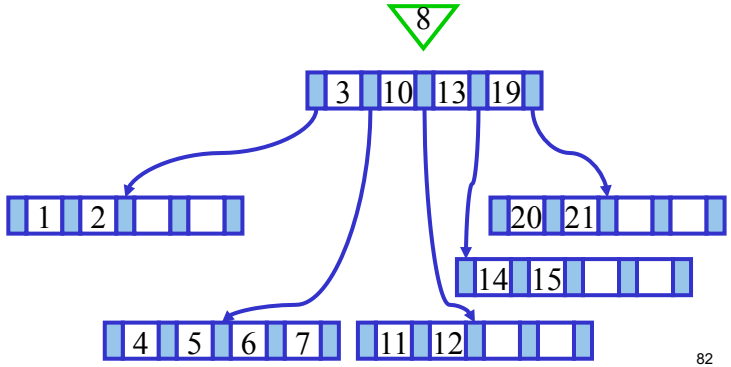
Sukzessiver Aufbau eines B-Baums
vom Grad k=2



80

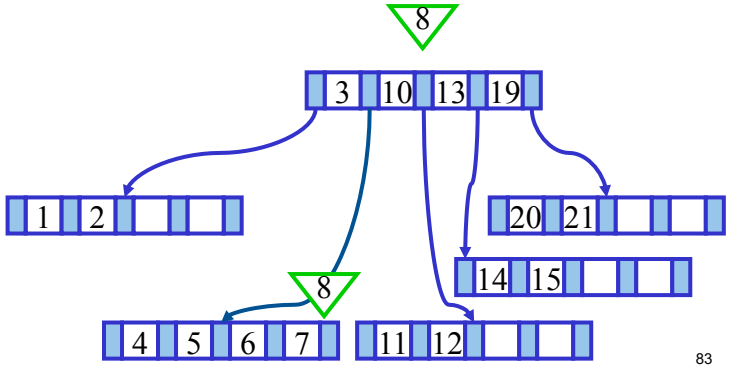
81

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



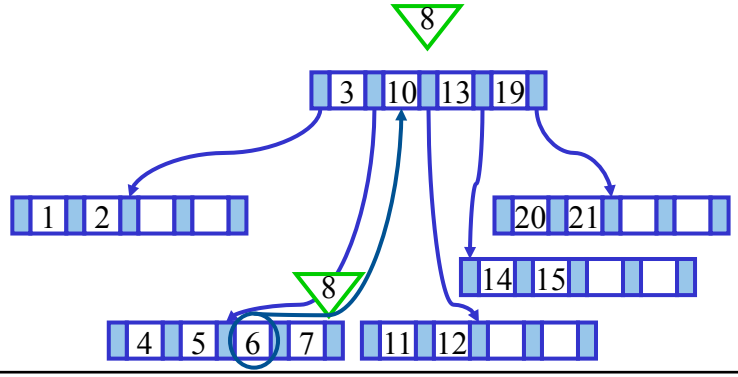
82

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



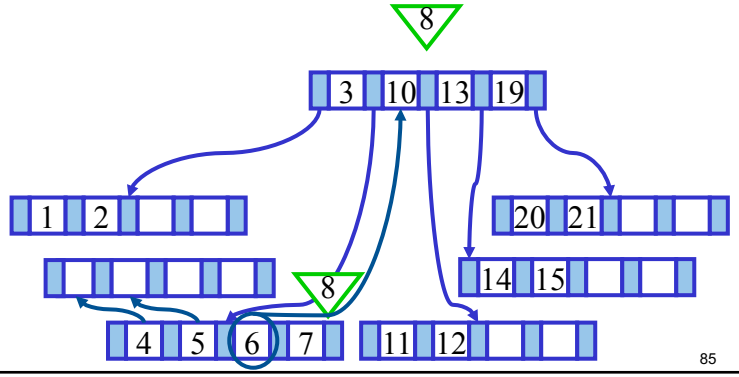
83

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



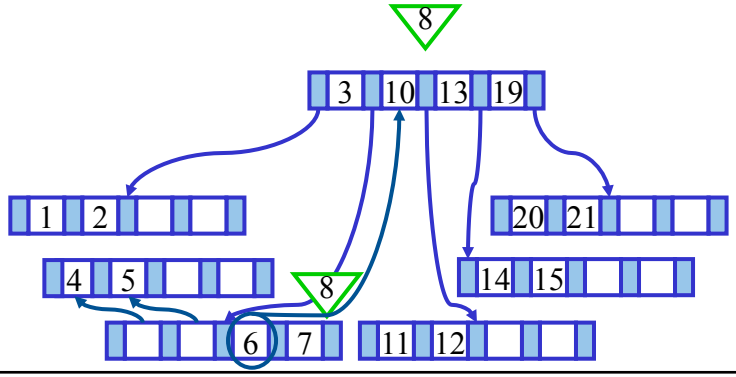
84

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



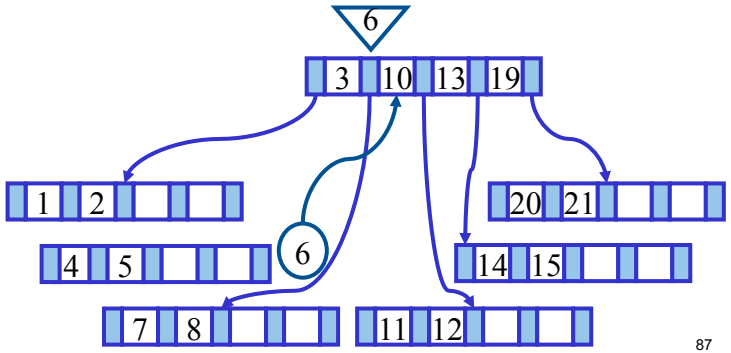
85

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



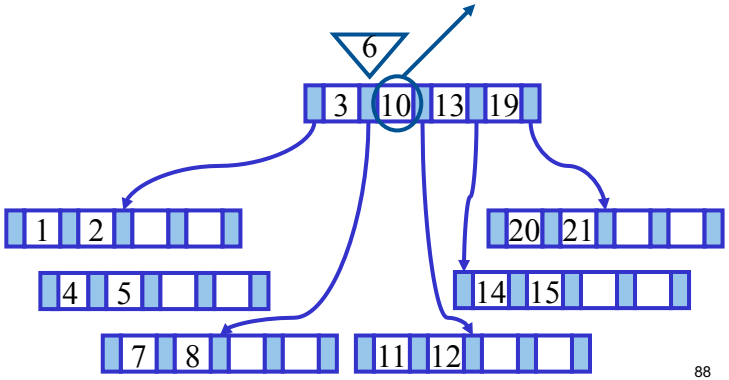
86

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



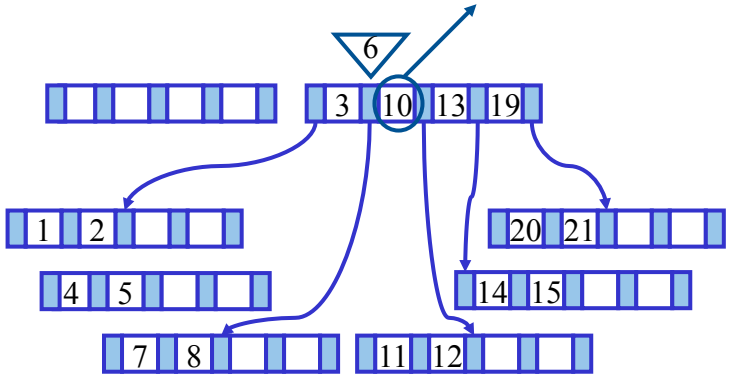
87

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



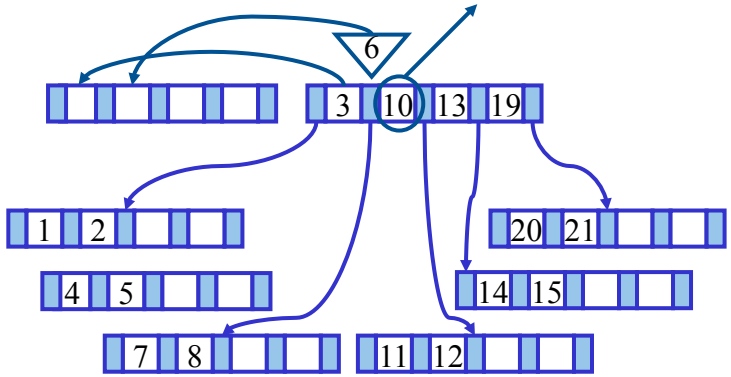
88

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



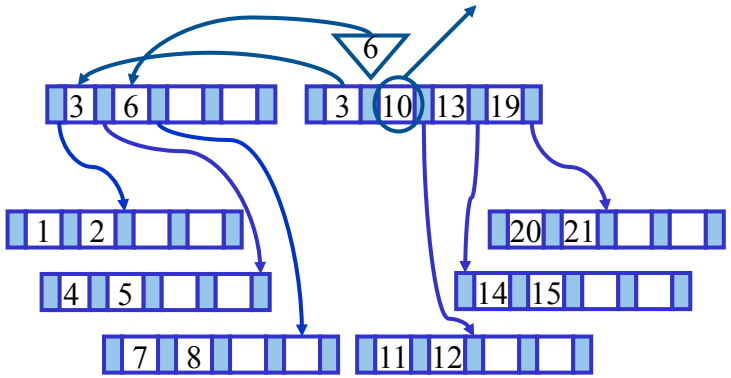
89

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



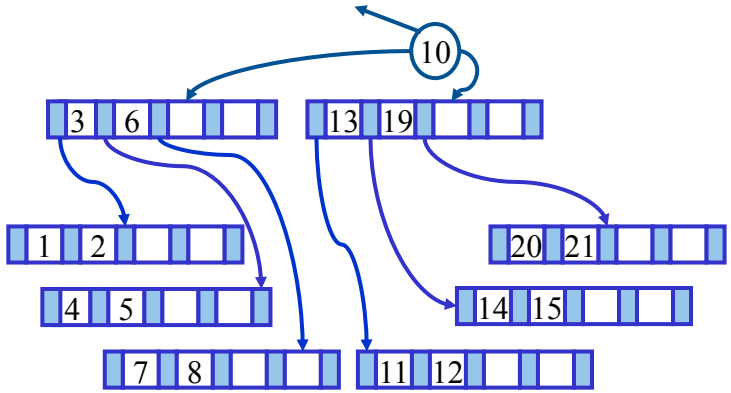
90

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



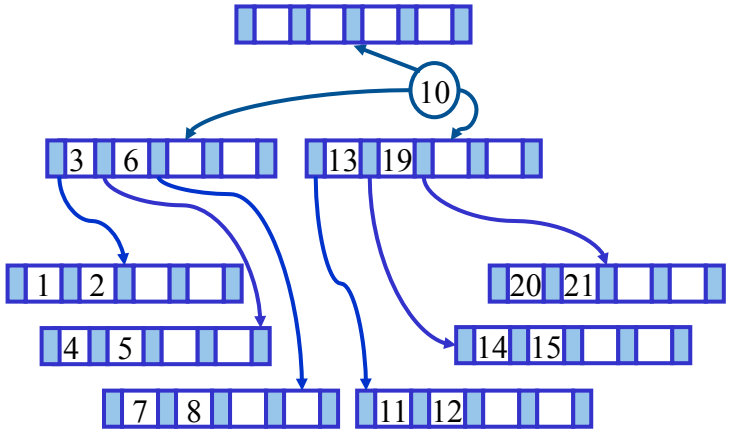
91

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



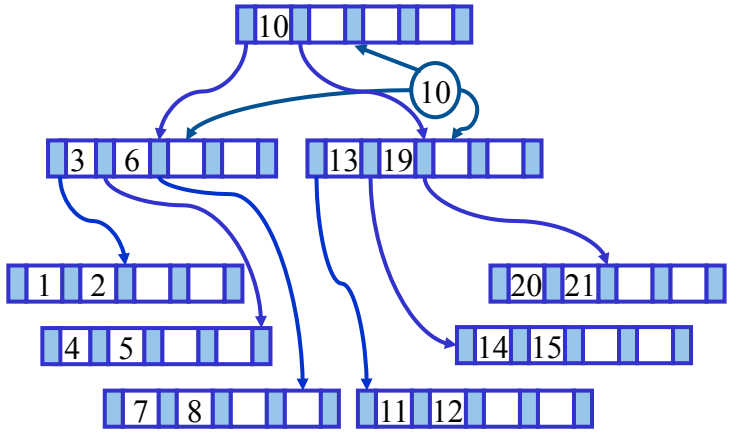
92

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



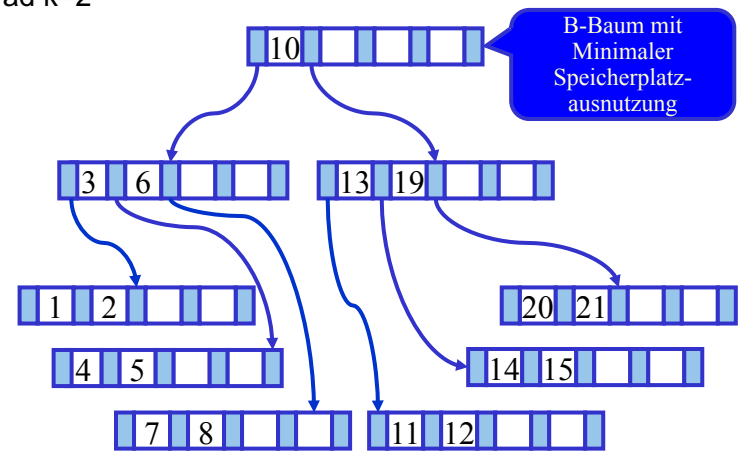
93

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



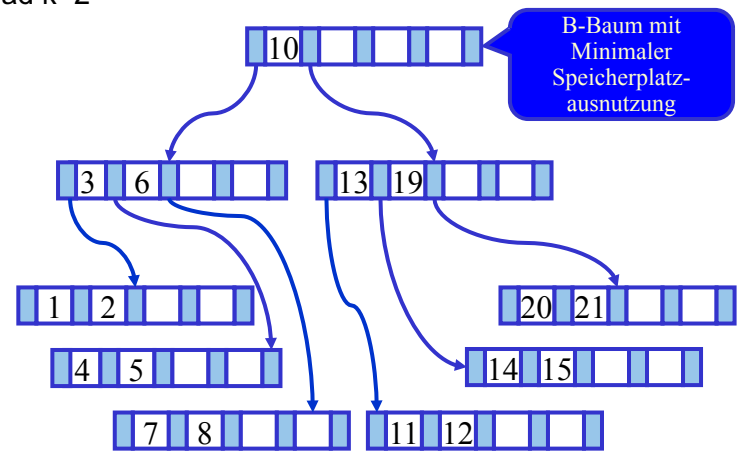
94

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



95

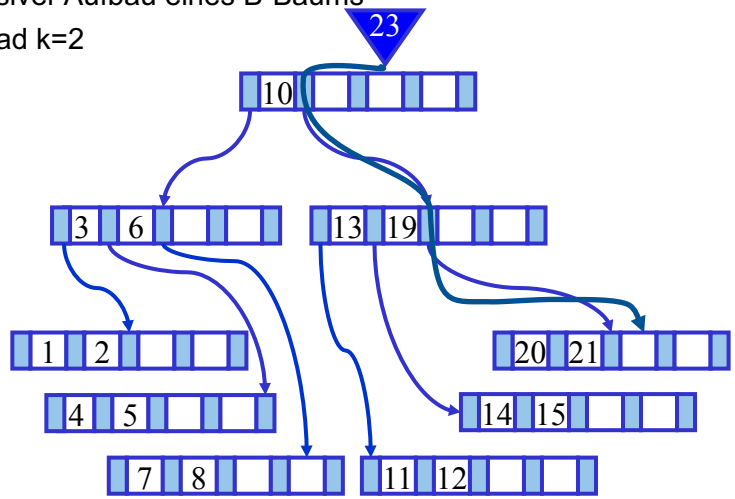
Sukzessiver Aufbau eines B-Baums
vom Grad k=2



96

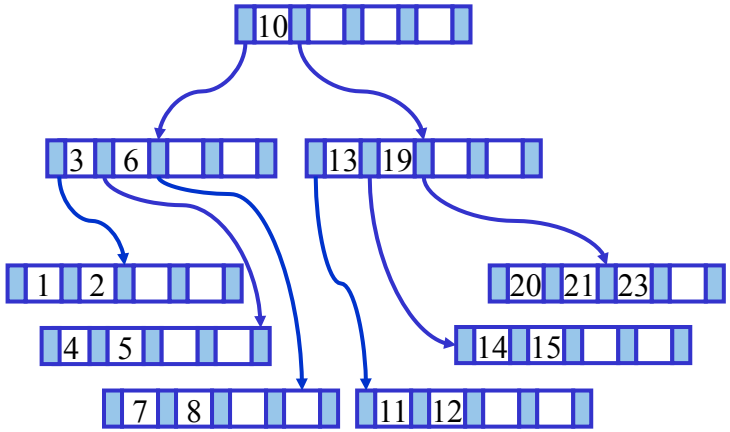
97

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



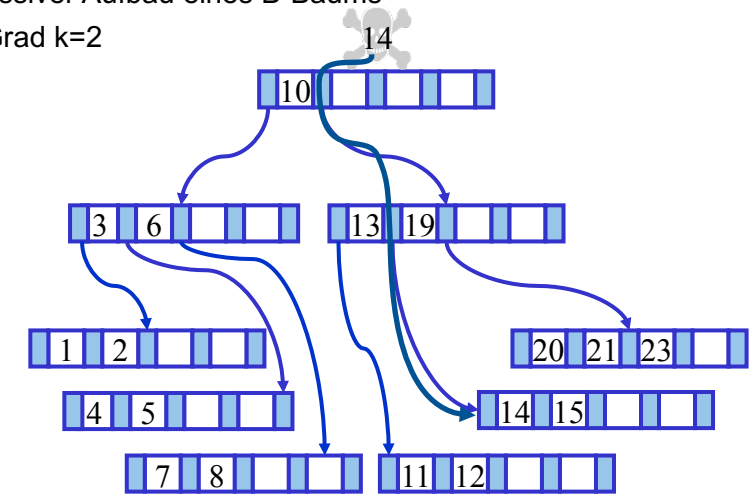
98

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



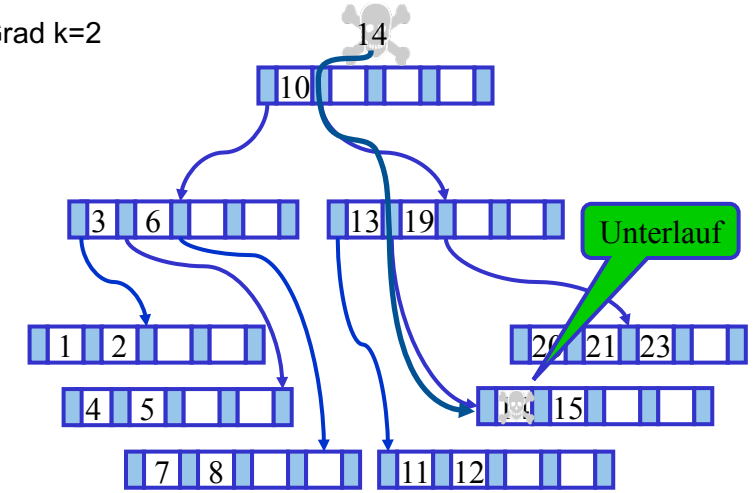
99

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



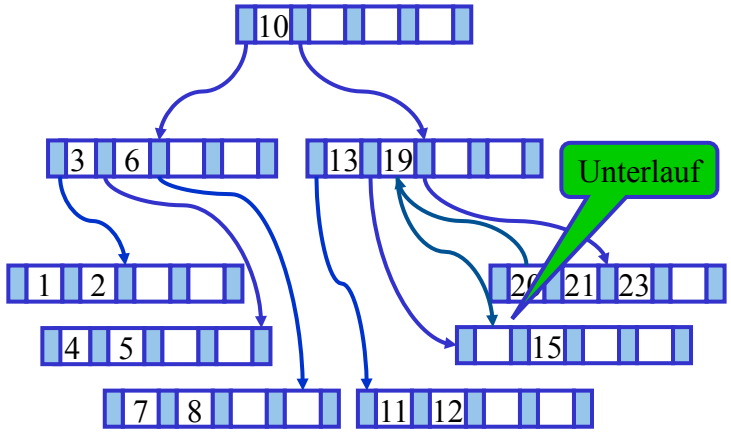
100

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



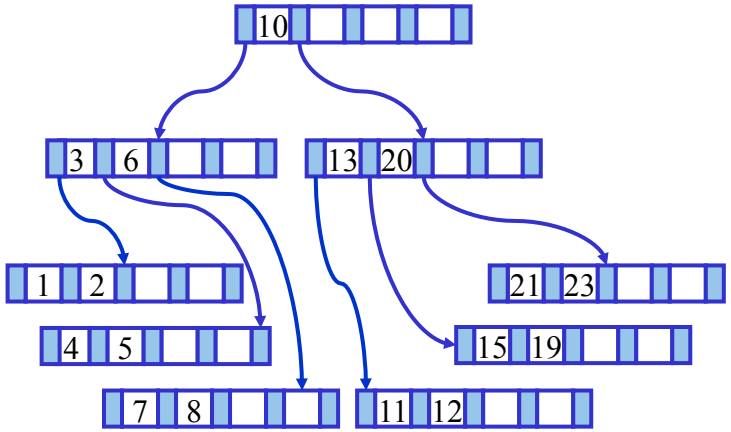
101

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



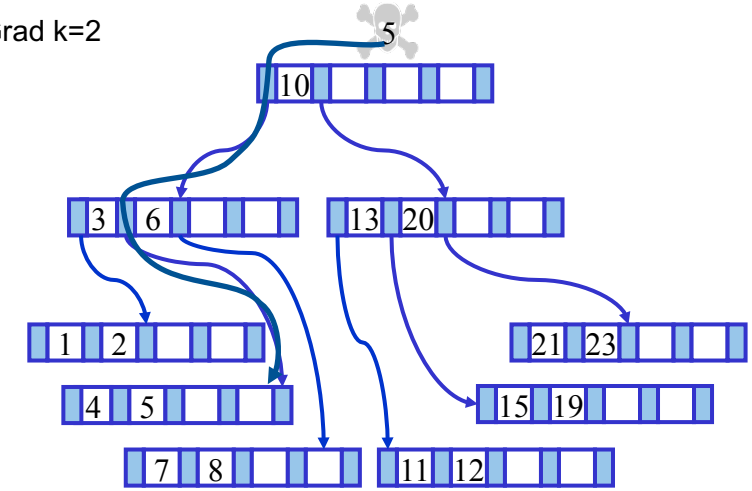
102

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



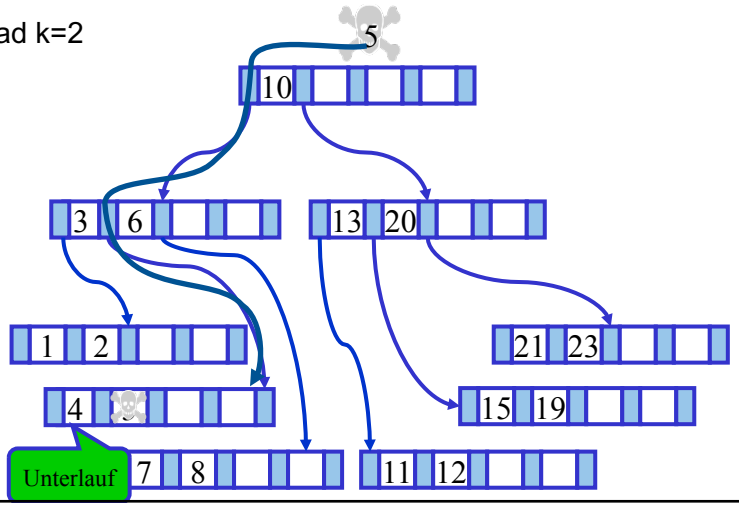
103

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



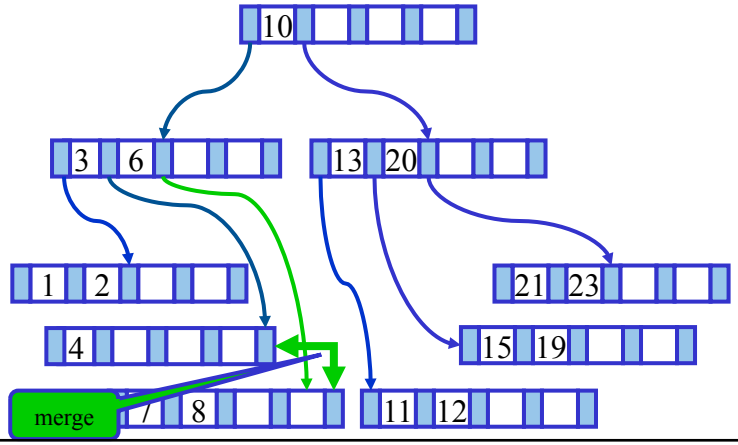
104

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



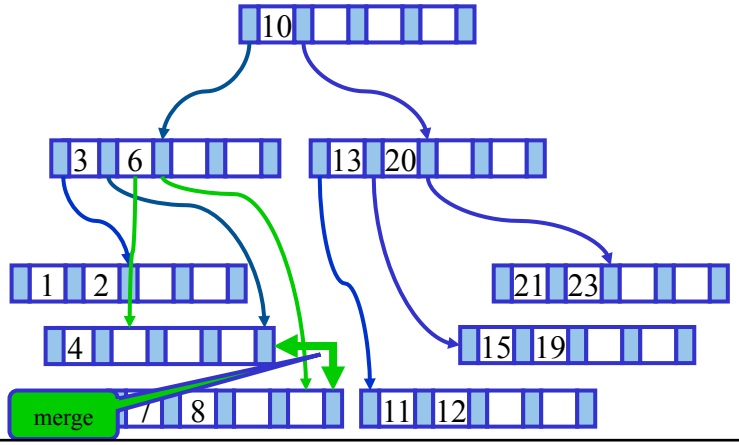
105

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



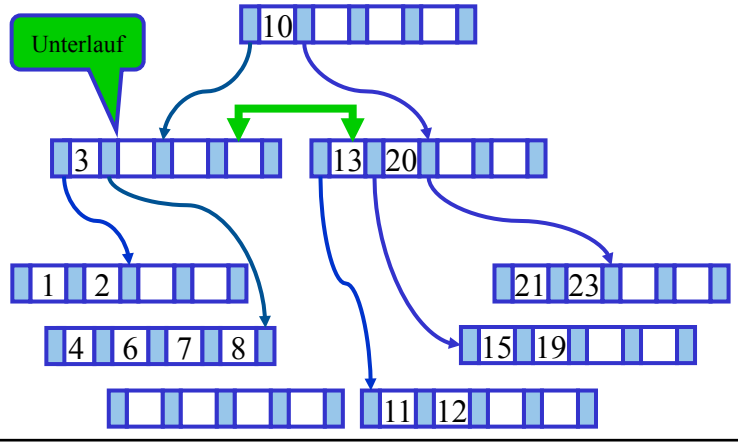
106

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



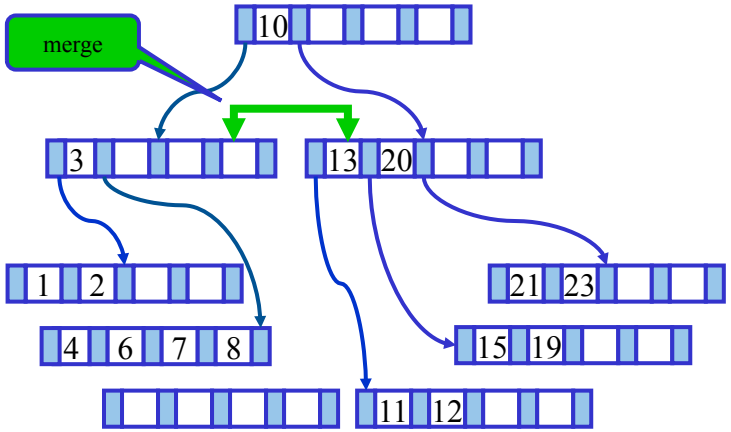
107

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



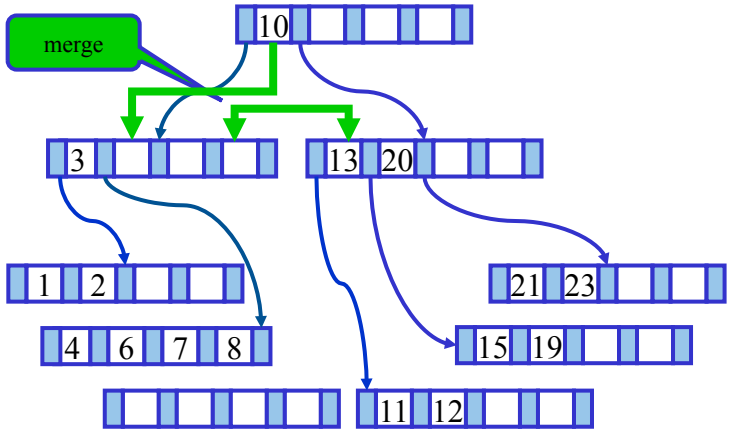
108

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



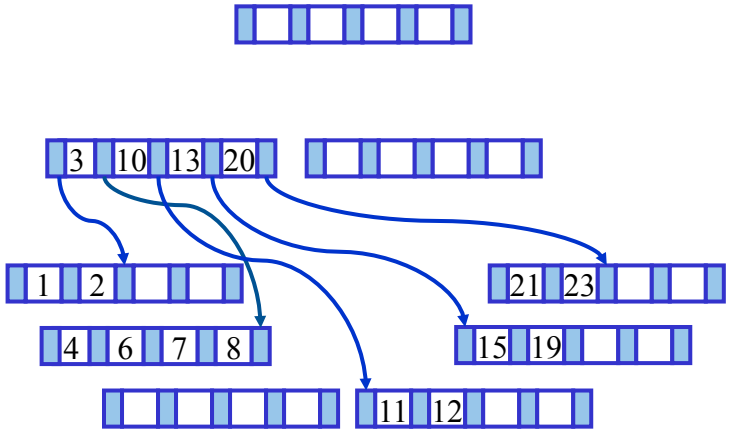
109

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



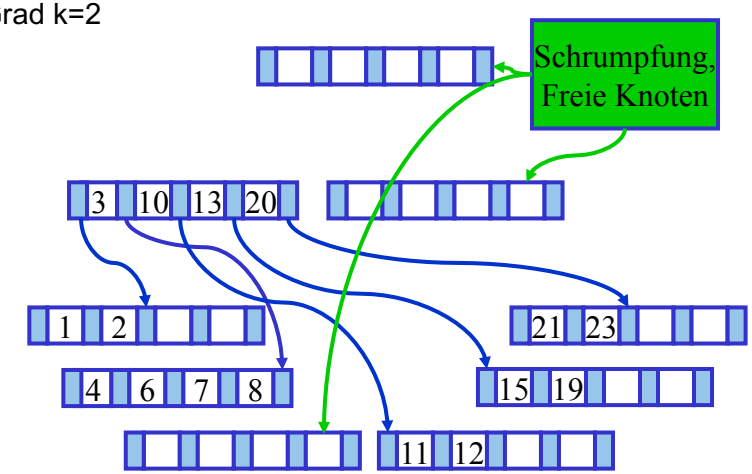
110

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



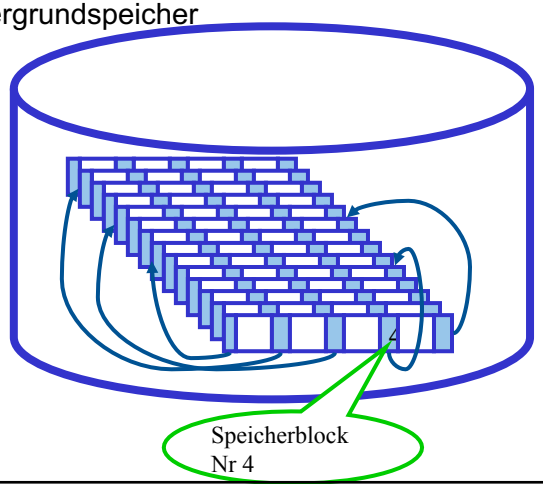
111

Sukzessiver Aufbau eines B-Baums
vom Grad k=2



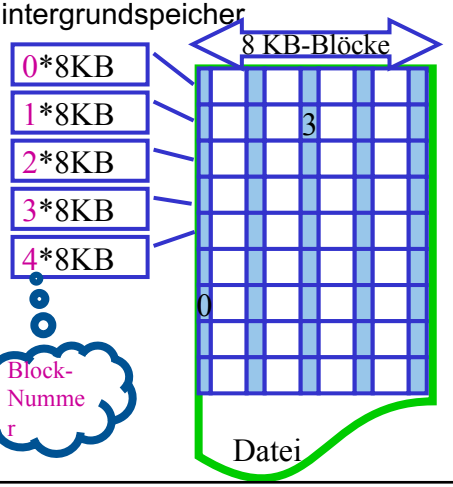
112

Speicherstruktur eines B-Baums
auf dem Hintergrundspeicher



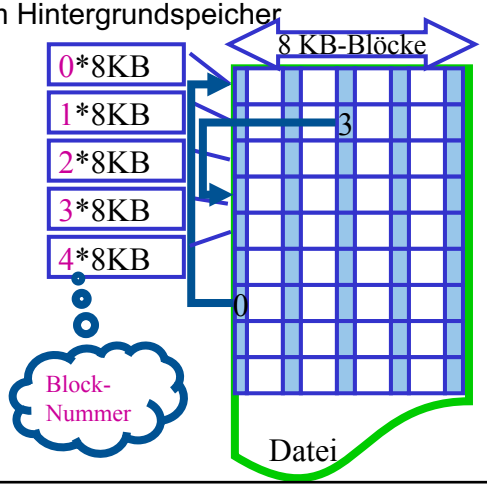
113

Speicherstruktur eines B-Baums
auf dem Hintergrundspeicher



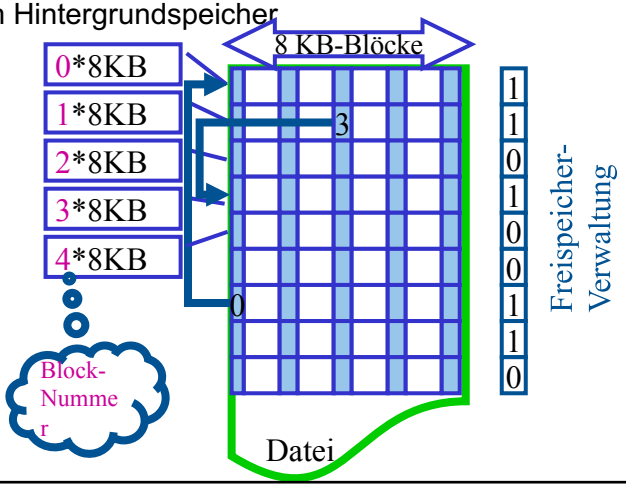
114

Speicherstruktur eines B-Baums
auf dem Hintergrundspeicher



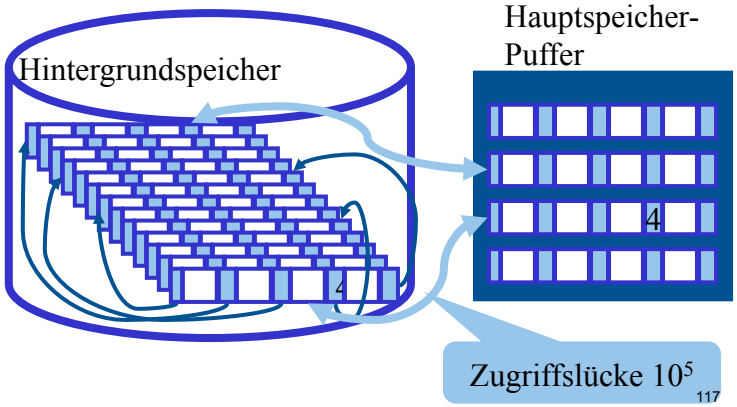
115

Speicherstruktur eines B-Baums
auf dem Hintergrundspeicher



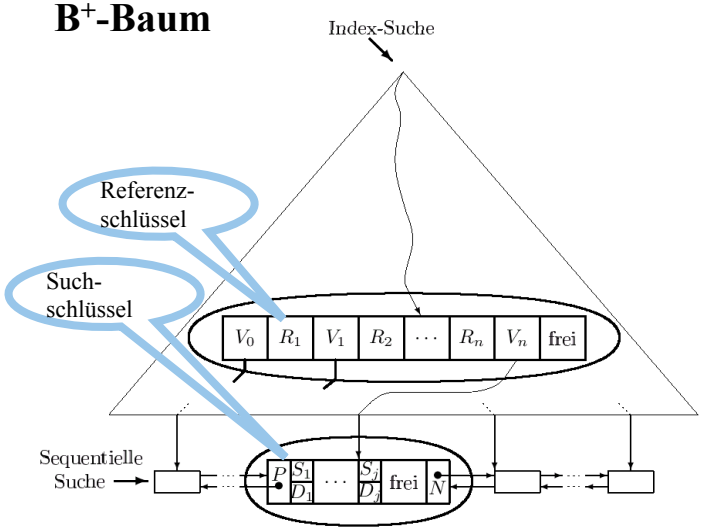
116

Zusammenspiel:
Hintergrundspeicher -- Hauptspeicher



117

B^+-Baum



118

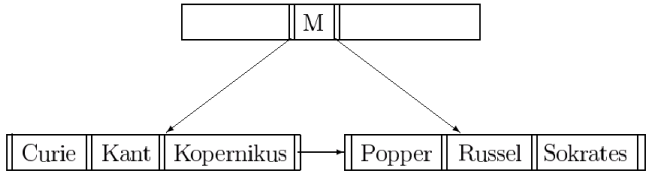
Ein B^+-Baum vom Typ (k, k^*) hat also folgende Eigenschaften:

1. Jeder Weg von der Wurzel zu einem Blatt hat die gleiche Länge.
2. Jeder Knoten – außer Wurzeln und Blättern – hat mindestens k und höchstens $2k$ Einträge. Blätter haben mindestens k^* und höchstens $2k^*$ Einträge. Die Wurzel hat entweder maximal $2k$ Einträge, oder sie ist ein Blatt mit maximal $2k^*$ Einträgen.
3. Jeder Knoten mit n Einträgen, außer den Blättern, hat $n+1$ Kinder.
4. Seien $R_1, \ldots, R_n$ die Referenzschlüssel eines inneren Knotens (d.h. auch der Wurzel) mit $n+1$ Kindern. Seien $V_0, V_1, \ldots, V_n$ die Verweise auf diese Kinder.
   (a) V_0 verweist auf den Teilbaum mit Schlüsseln kleiner oder gleich R_1.
   (b) V_i $(i = 1, \ldots, n-1)$ verweist auf den Teilbaum, dessen Schlüssel zwischen R_i und R_{i+1} liegen (einschließlich R_{i+1}).
   (c) V_n verweist auf den Teilbaum mit Schlüsseln größer als R_n.

119

Präfix-B^+-Bäume

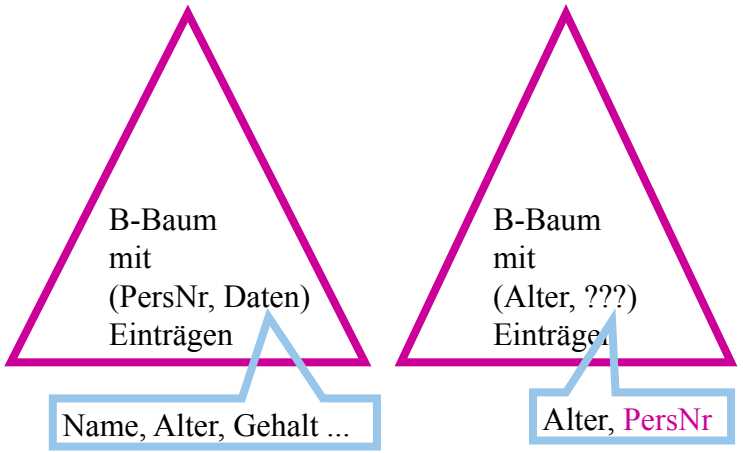
- Es werden nur *Referenz*schlüssel benötigt.



- beliebiger Referenzschlüssel R mit Kopernikus $\leq R <$ Popper

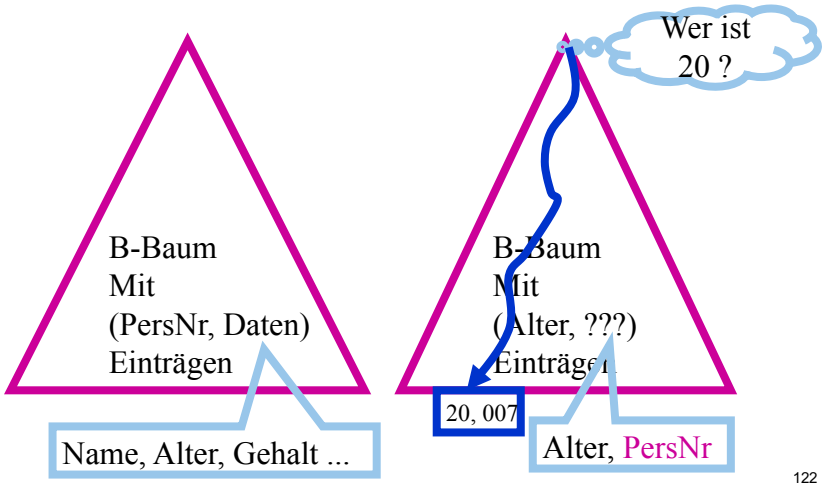
120

Mehrere Indexe auf denselben Objekten



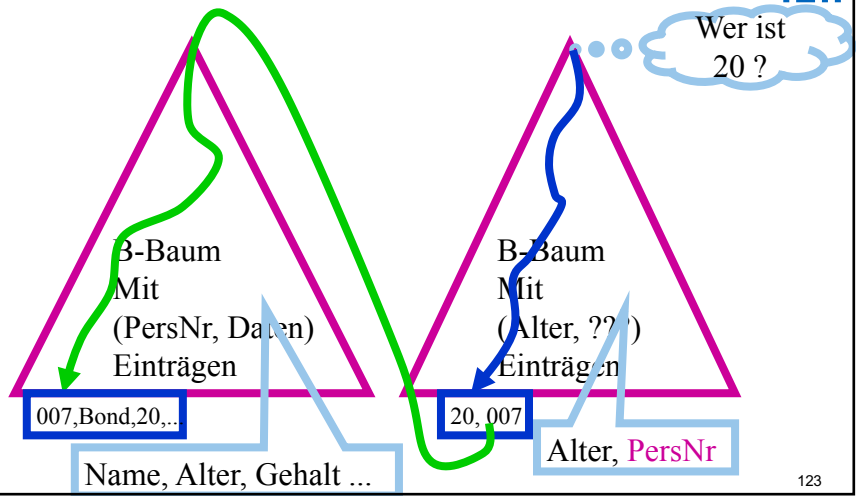
121

Mehrere Indexe auf denselben Objekten



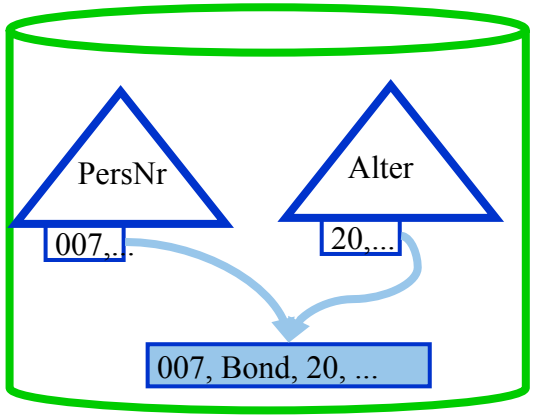
122

Mehrere Indexe auf denselben Objekten



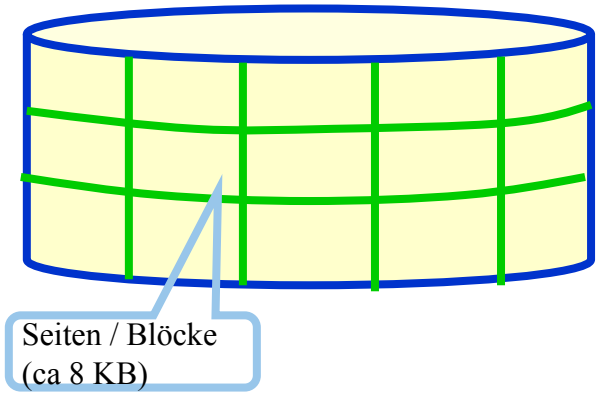
123

Eine andere Möglichkeit: Referenzierung über Speicheradressen



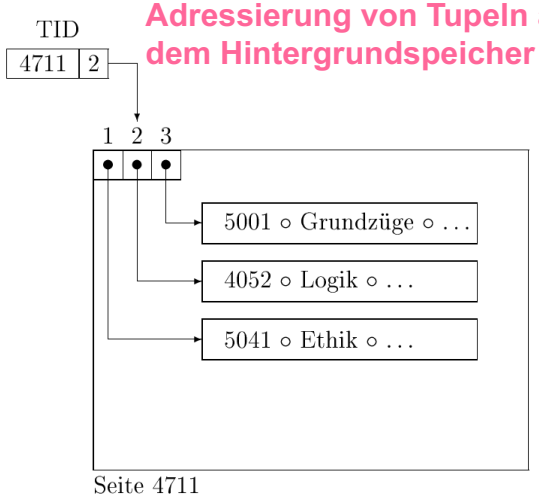
124

Realisierungstechnik für Hintergrundspeicher-Adressen



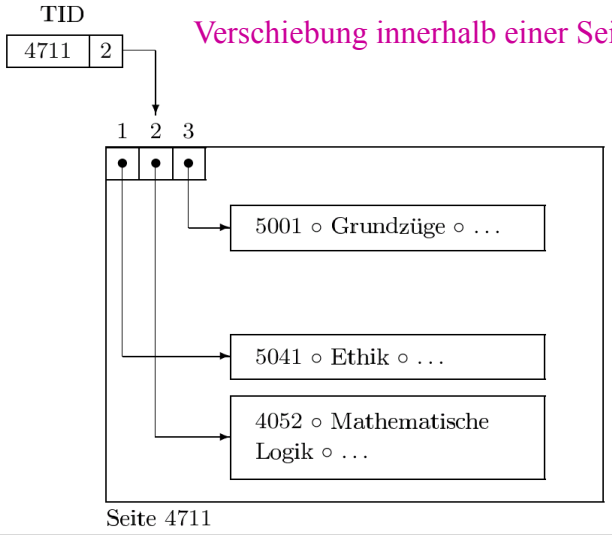
125

Adressierung von Tupeln auf dem Hintergrundspeicher



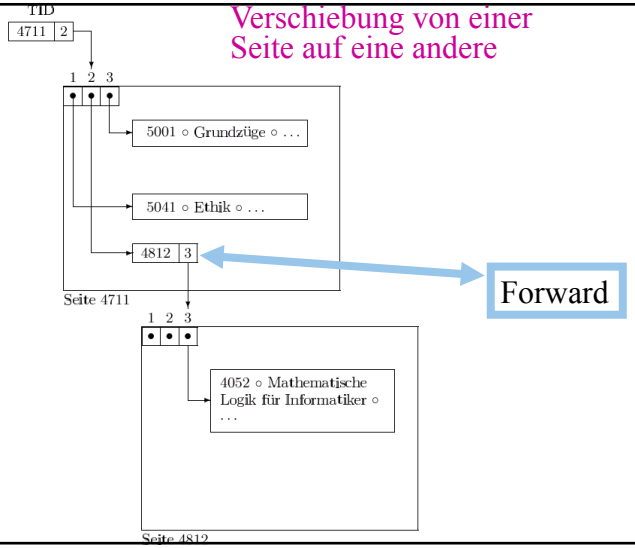
126

Verschiebung innerhalb einer Seite

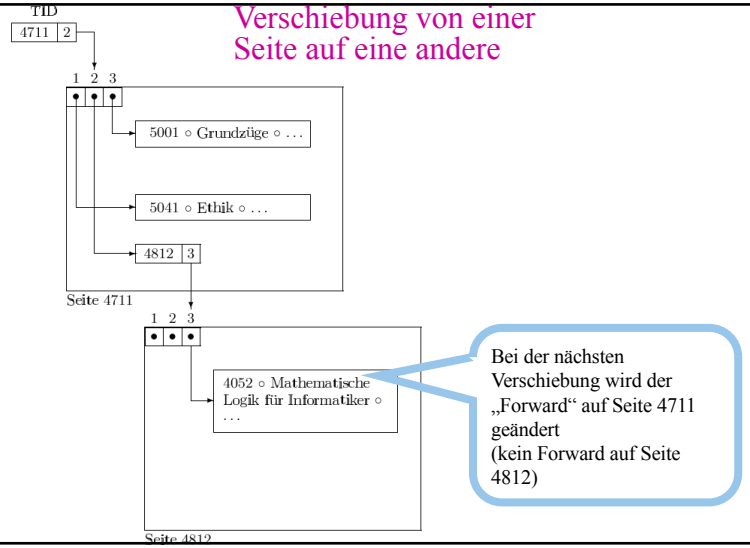


127

Verschiebung von einer Seite auf eine andere



128



„Statische“ Hashtabellen

À priori Allokation des Speichers
Nachträgliche Vergrößerung der Hashtabelle ist „teuer“

- Hashfunktion h(...) = ... mod N
- Rehashing der Einträge
  - h(...) = ... mod M
- In Datenbankanwendungen viele GB

Erweiterbares Hashing

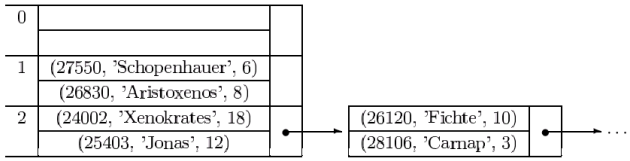
- Zusätzliche Indirektion über ein Directory
- Ein zusätzlicher Zugriff auf ein Directory, das den Zeiger (Verweis, BlockNr) des Hash-Bucket enthält
- Dynamisches Wachsen (und Schrumpfen) ist möglich
- Der Zugriff auf das Directory erfolgt über einen binären Hashcode

Statisches Hashing

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

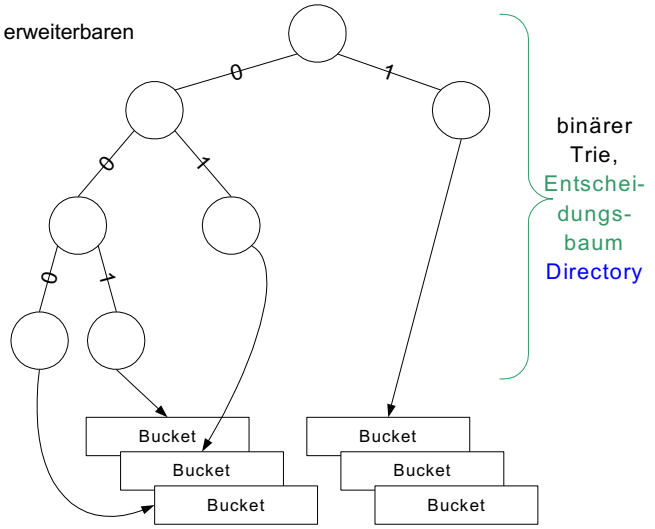
0	
1	(27550, 'Schopenhauer', 6)
2	(24002, 'Xenokrates', 18)
	(25403, 'Jonas', 12)

- Kollisionsbehandlung



$\Longrightarrow$ ineffizient bei nicht vorhersehbarer Datenmenge

Grundidee des erweiterbaren Hashings



133

Hashfunktion für erweiterbares Hashing

h: Schlüsselmenge → {0,1}*

Der Bitstring muss lang genug sein, um alle Objekte auf ihre Buckets abbilden zu können

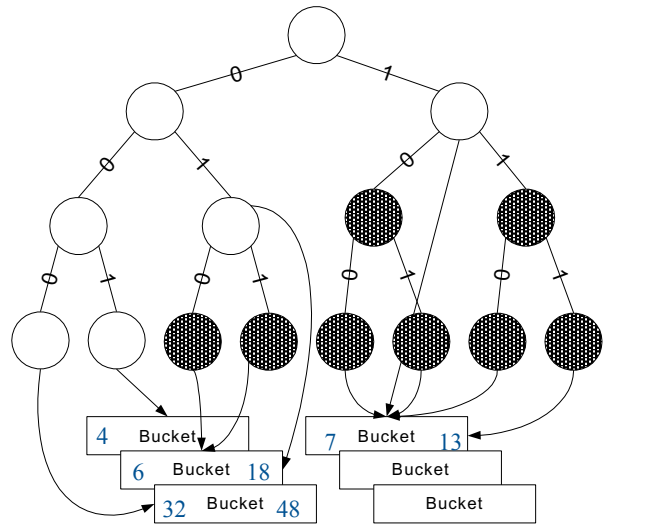
Anfangs wird nur ein (kurzer) Präfix des Hashwertes (Bitstrings) benötigt

Wenn die Hashtabelle wächst wird aber sukzessive ein längerer Präfix benötigt

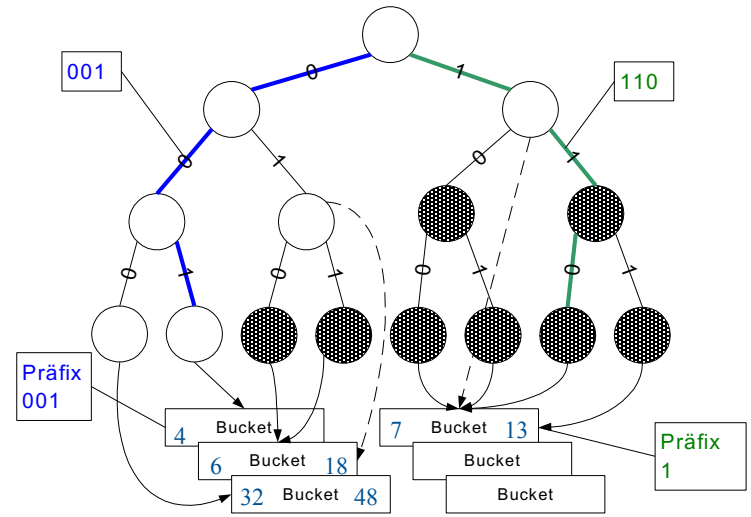
Beispiel-Hashfunktion: gespiegelte binäre PersNr

- 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)

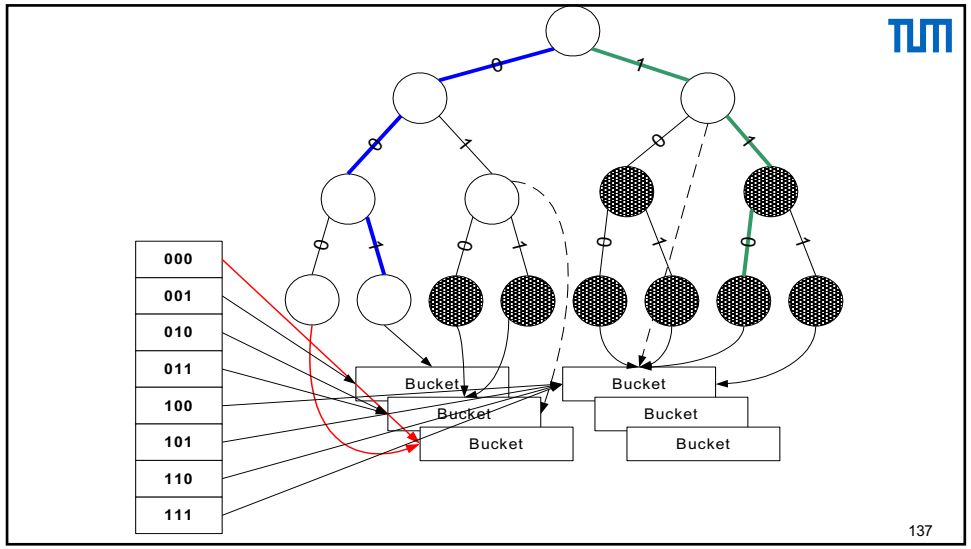
134

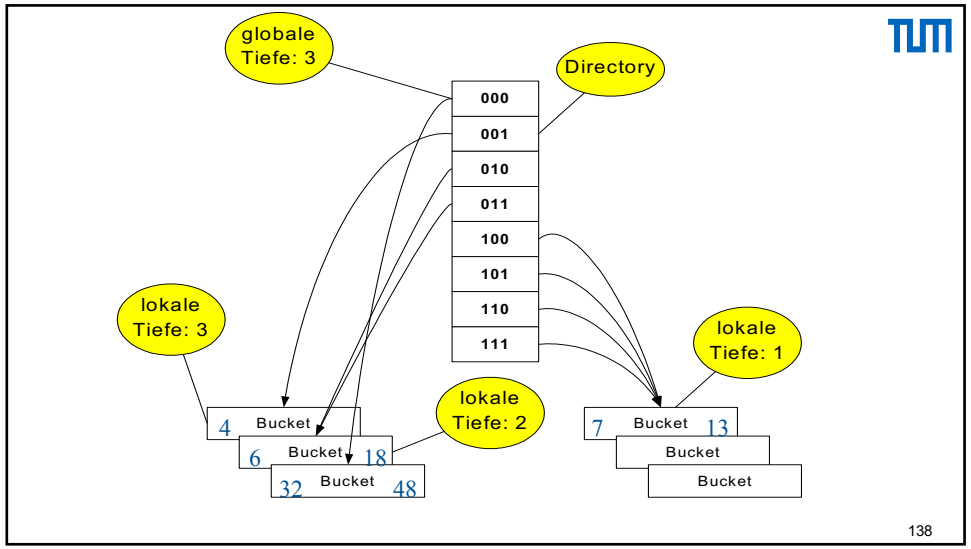


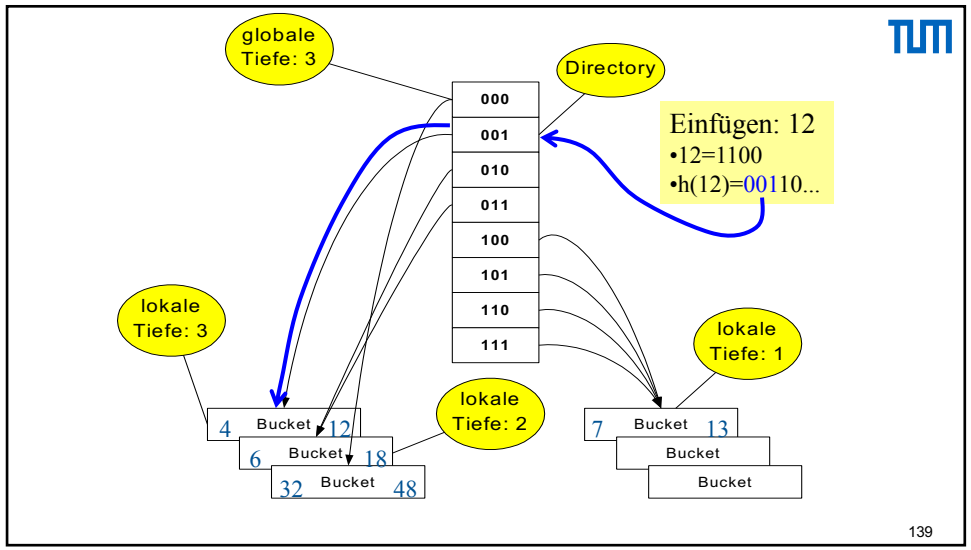
135

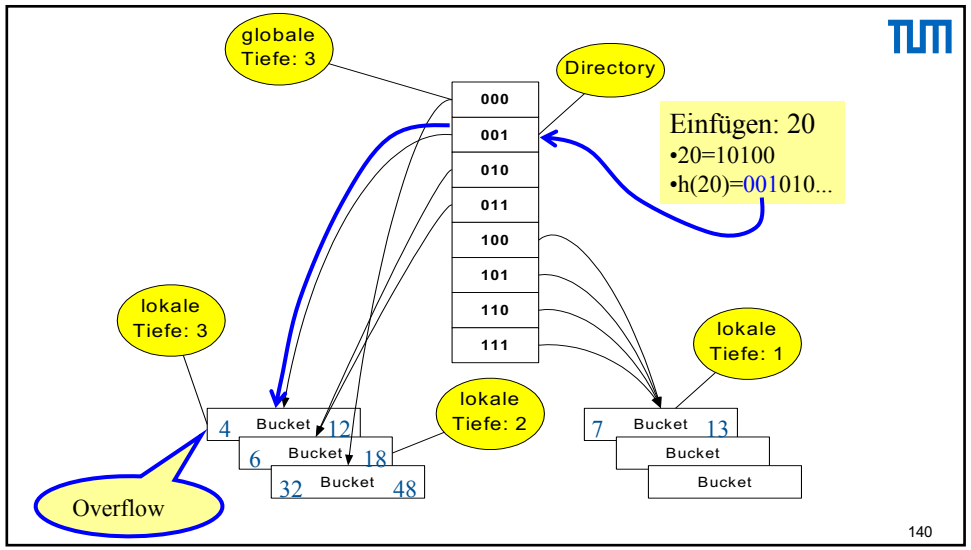


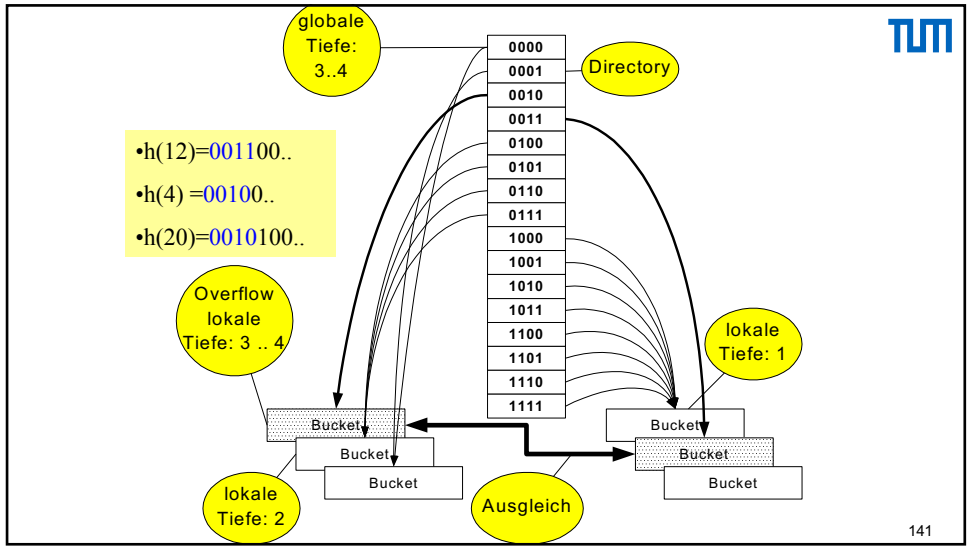
136

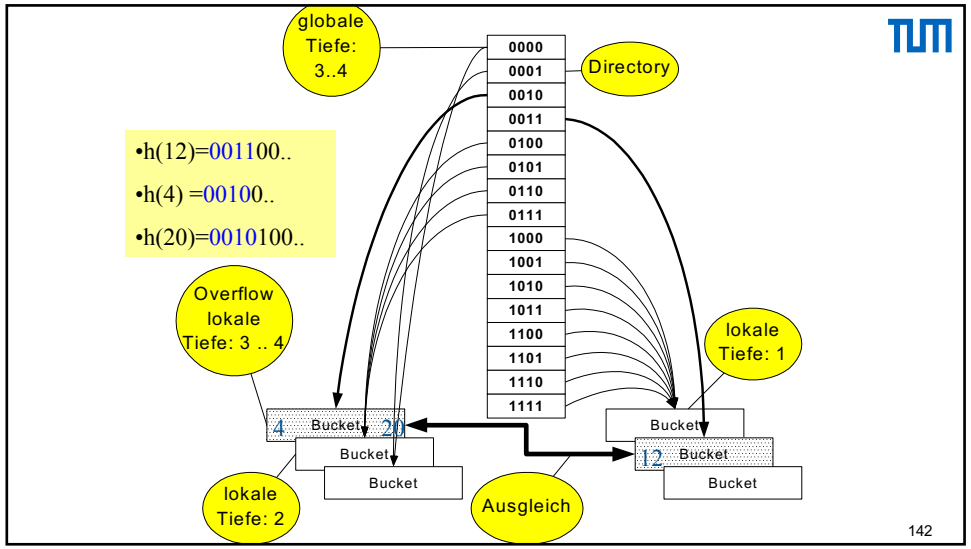

0
1
000
001
010
011
100
101
110
111
Bucket
Bucket
Bucket
Bucket
Bucket
Bucket
137

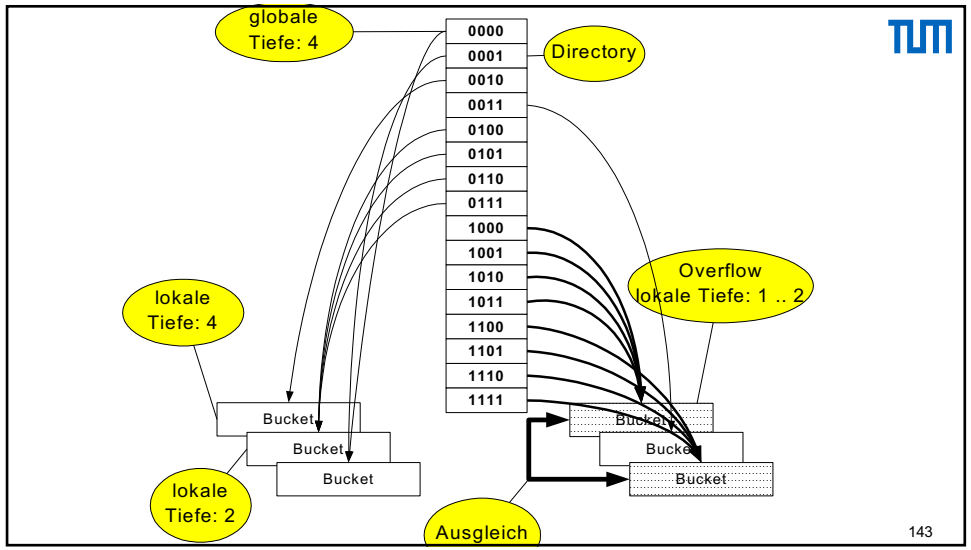

globale Tiefe: 3
Directory
000
001
010
011
100
101
110
111
lokale Tiefe: 3
lokale Tiefe: 1
lokale Tiefe: 2
4 Bucket
6 Bucket 18
32 Bucket 48
7 Bucket 13
Bucket
Bucket
138


globale Tiefe: 3
Directory
Einfügen: 12
•12=1100
•h(12)=00110...
000
001
010
011
100
101
110
111
lokale Tiefe: 3
lokale Tiefe: 1
lokale Tiefe: 2
4 Bucket 12
6 Bucket 18
32 Bucket 48
7 Bucket 13
Bucket
Bucket
139


globale Tiefe: 3
Directory
Einfügen: 20
•20=10100
•h(20)=001010...
000
001
010
011
100
101
110
111
lokale Tiefe: 3
lokale Tiefe: 1
lokale Tiefe: 2
4 Bucket 12
6 Bucket 18
32 Bucket 48
7 Bucket 13
Bucket
Bucket
Overflow
140

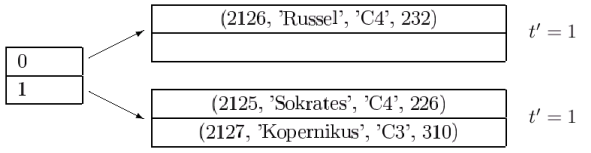






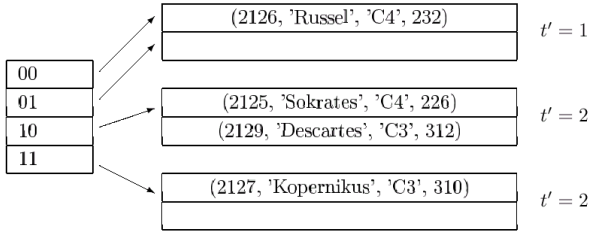
Demonstration des erweiterbaren Hashings

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



144

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



145

146

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

147

Grundlagen mehrdimensionaler Datenstrukturen

Wertebereiche $D_0,..., D_{k-1}$:
alle D_i sind endlich, linear geordnet und besitzen kleinstes ($-\infty_i$) und größtes (∞_i) Element

Datenraum $\boldsymbol{D} = D_0 \text{ x...x } D_{k-1}$

k-dimensionaler Schlüssel entspricht Punkt im Datenraum $p \in \boldsymbol{D}$

148

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

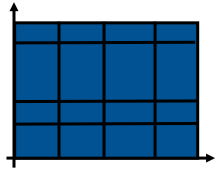
149

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



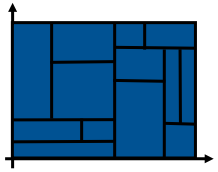
150

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



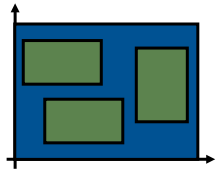
151

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



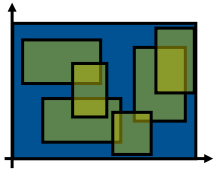
152

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



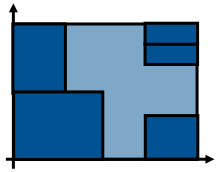
153

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



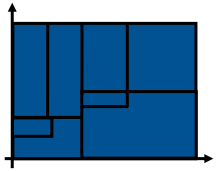
154

Charakterisierung mehrdimensionaler Datenstrukturen

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

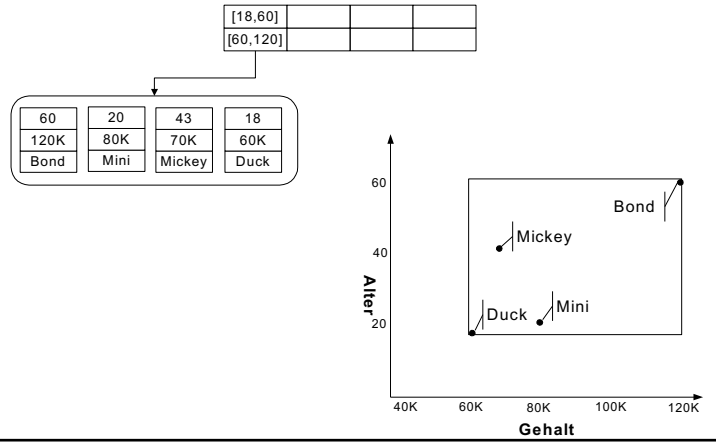
1. nur atomare Gebiete (beschreibbar durch ein Rechteck)
2. vollständig (die Vereinigung aller Gebiete ergibt den gesamten Datenraum)
3. disjunkt (die Gebiete überlappen nicht)

Z-B-Baum: vollständig,disjunkt



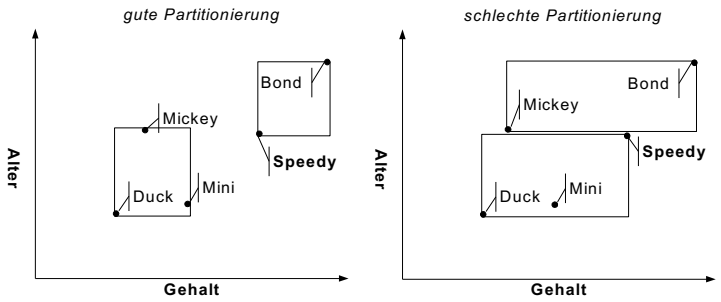
155

R-Baum: Urvater der baum-strukturierten mehrdimensionalen Zugriffsstrukturen



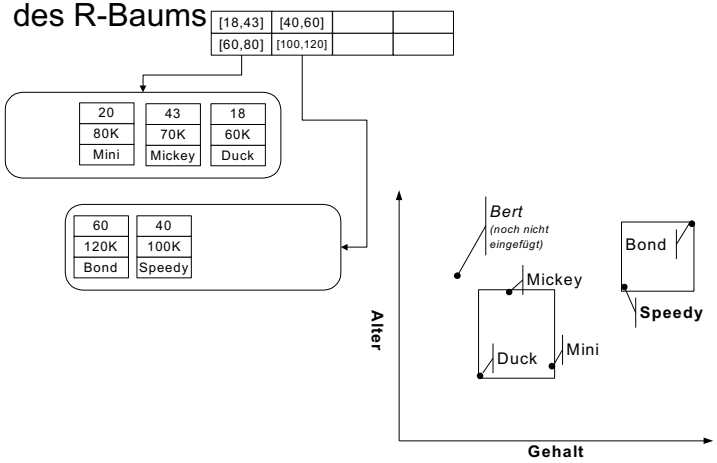
156

Gute versus schlechte Partitionierung



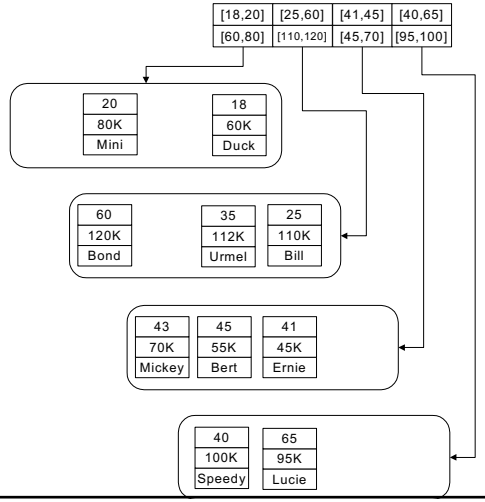
157

Nächste Phase in der Entstehungsgeschichte des R-Baums



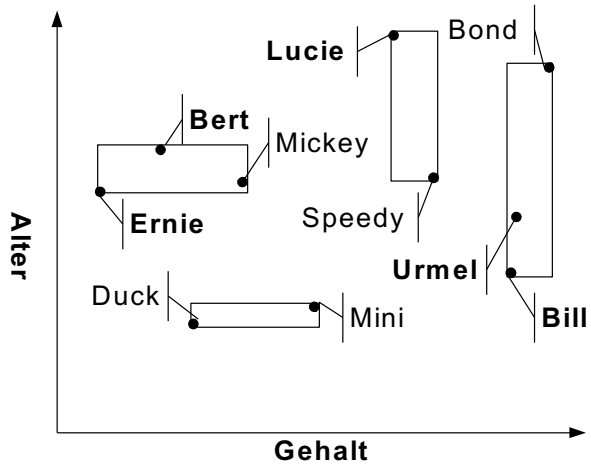
158

Nächste Phase



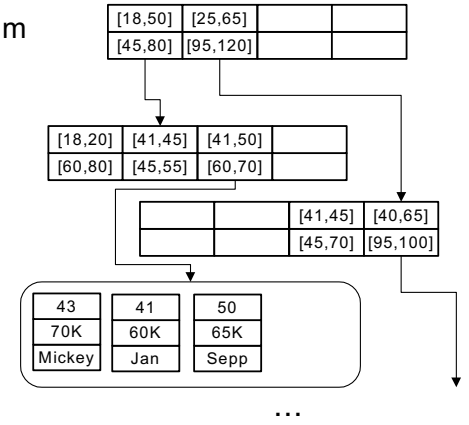
159

Datenraum

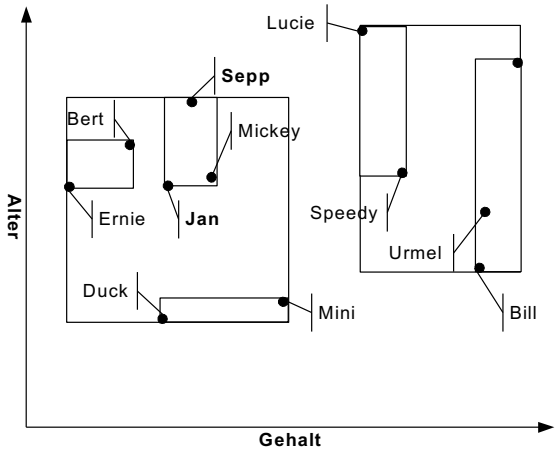


160

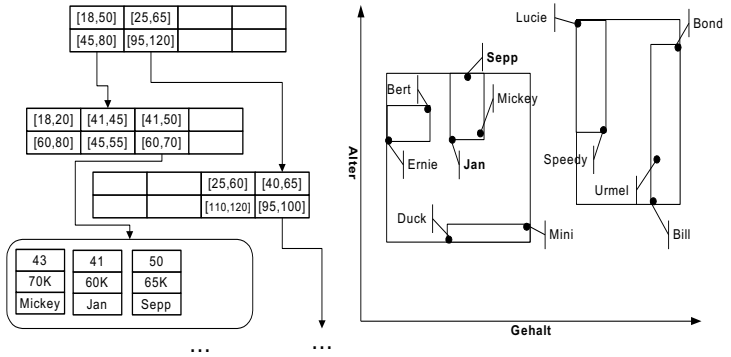
Wachsen des Baums:
nach oben – wie im B-Baum

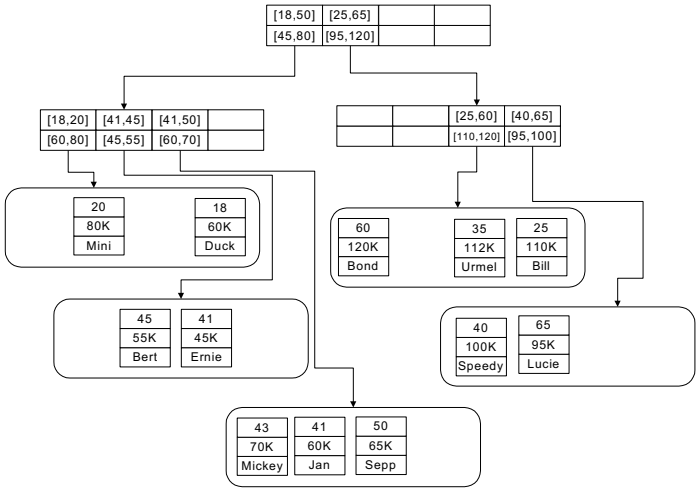


Datenraum

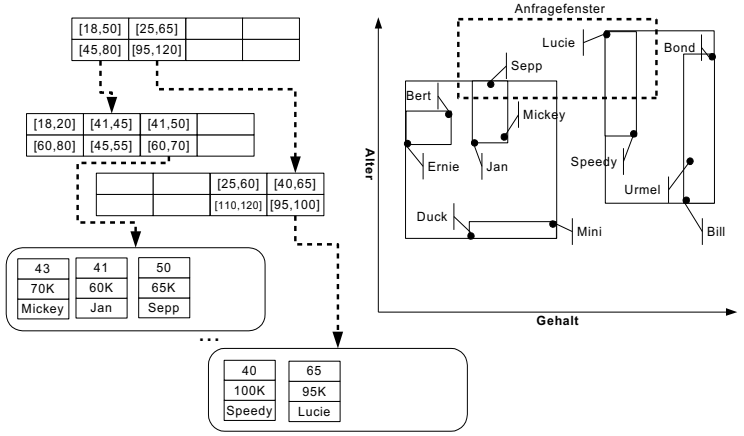


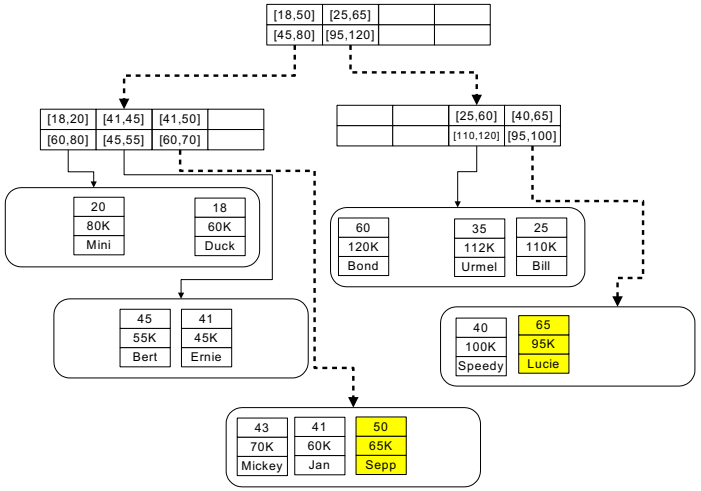
Datenraum und Speicherstruktur – Überblick



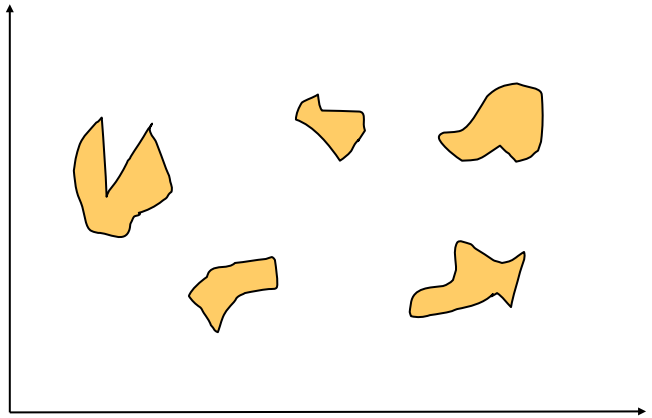


Bereichsanfragen auf dem R-Baum

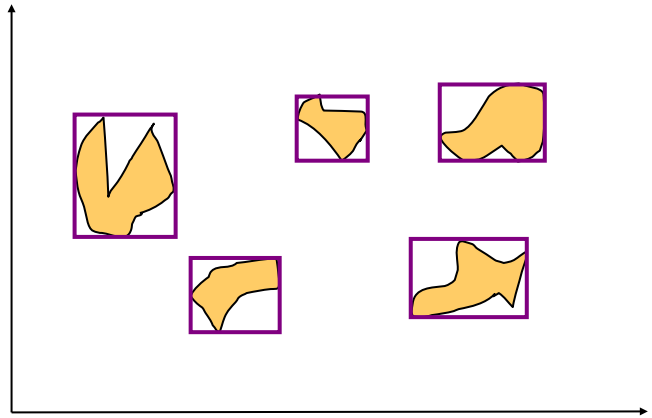




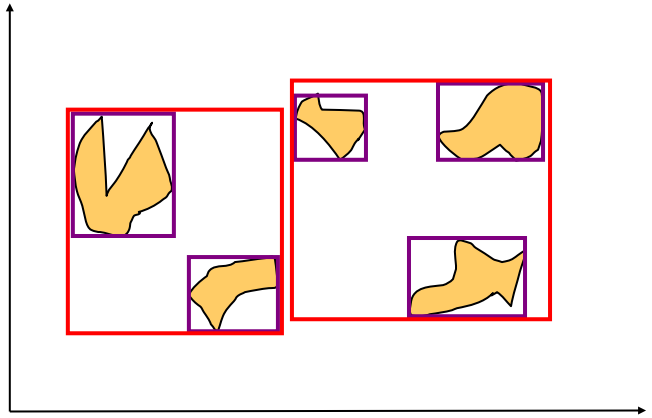
Indexierung räumlicher Objekte (anstatt Punkten) mit dem R-Baum



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



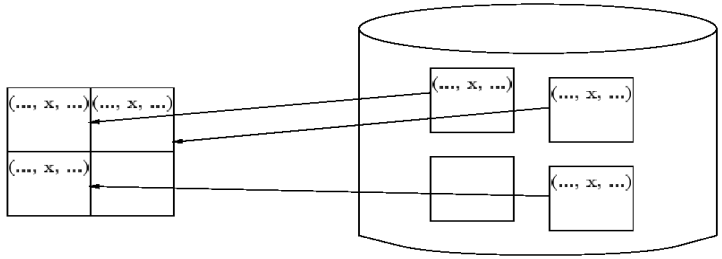
Indexierung räumlicher Objekte (anstatt Punkten) mit dem R-Baum



169

Objektballung / Clustering logisch verwandter Daten

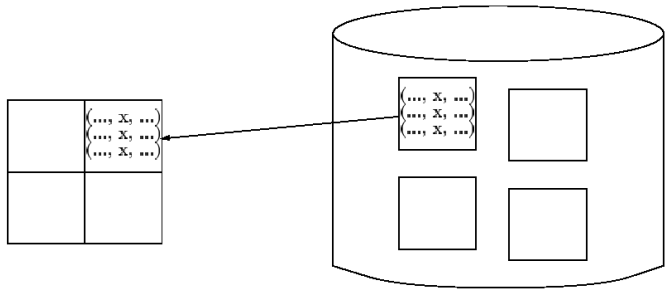
```
select *
from R
where A = x;
```



Hauptspeicher ← Zugriffslücke → Hintergrundspeicher

170

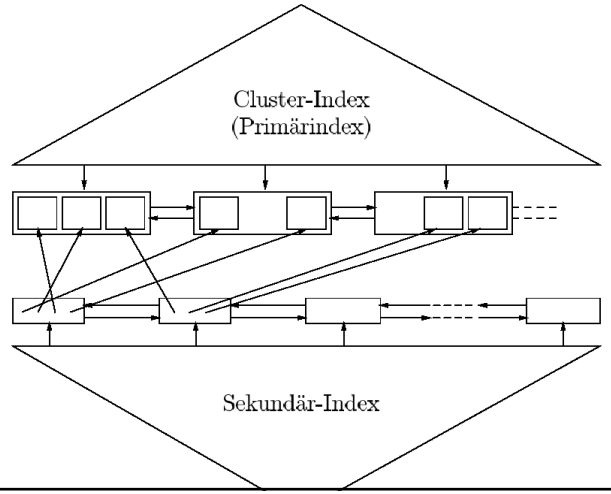
Hauptspeicher ← Zugriffslücke → Hintergrundspeicher



202

171

Indexe und Ballung



172

„verzahnte“ Ballung bei 1:N Beziehungen

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
```


Indexe in SQL

```
Create index SemsterInd
      on Studenten
      (Semester)
```

```
drop index SemsterInd
```