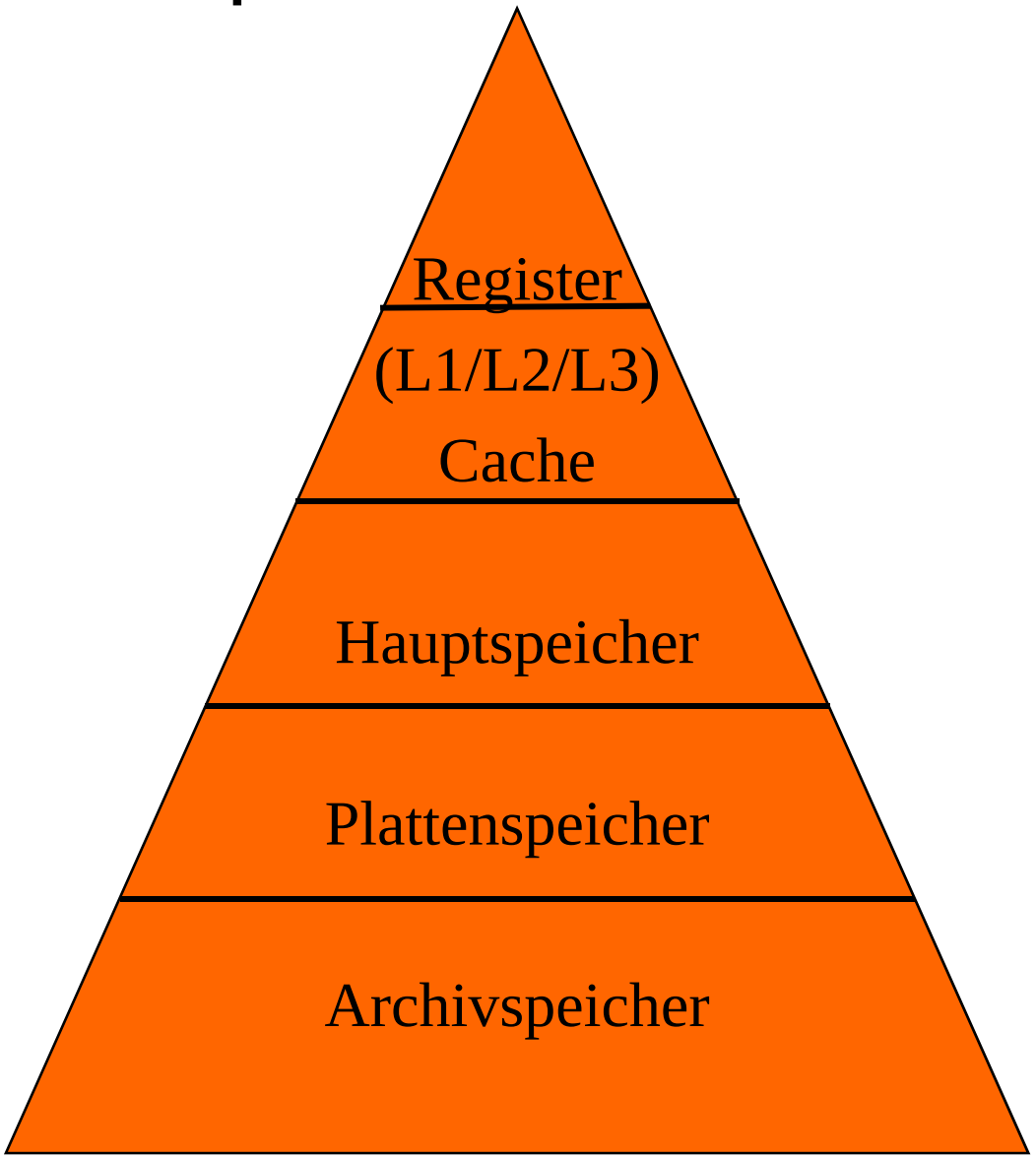
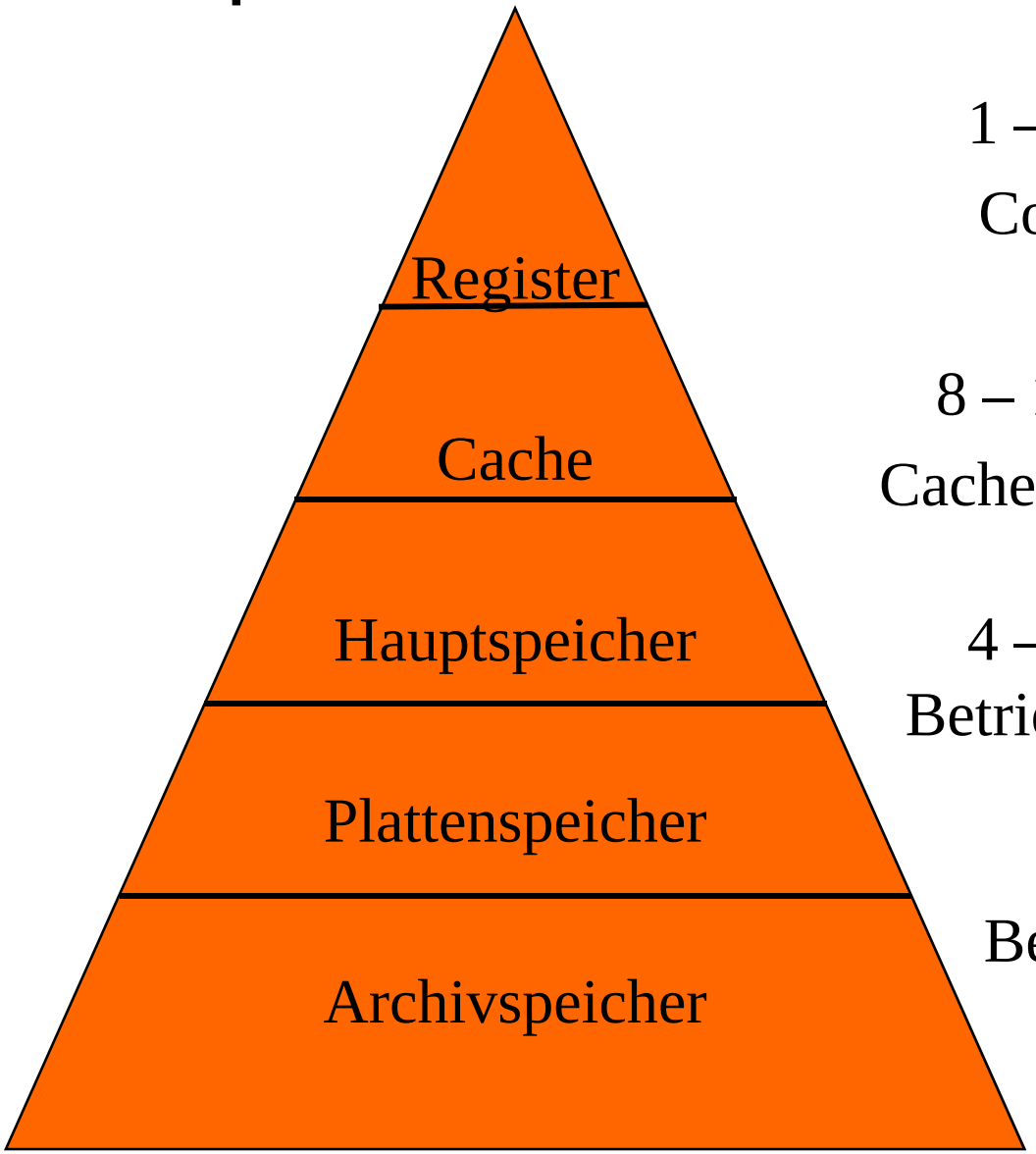
Kapitel 7
Physische Datenorganisation

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

Überblick: Speicherhierarchie



Überblick: Speicherhierarchie



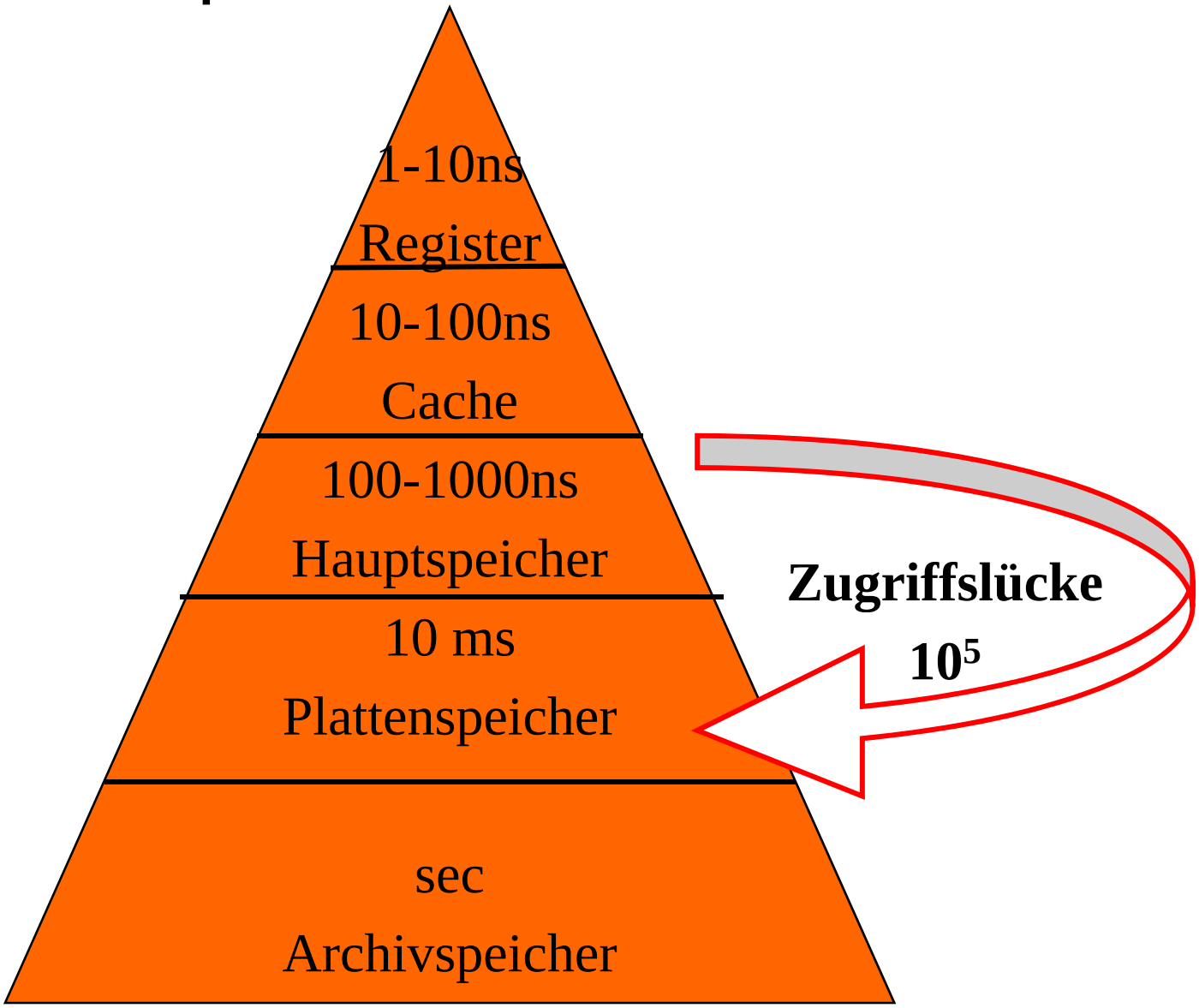
1 – 8 Byte
Compiler

8 – 128 Byte
Cache-Controller

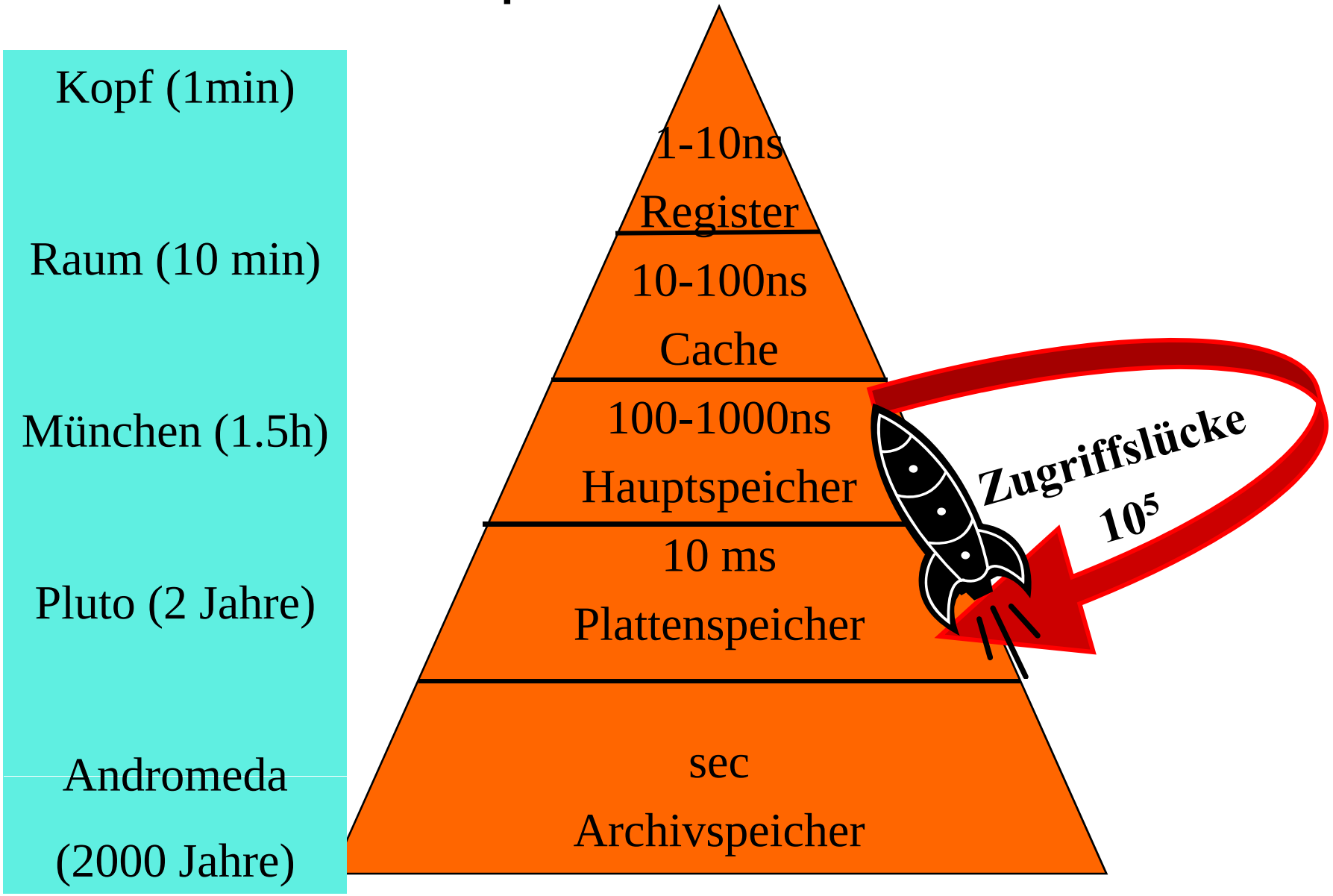
4 – 64 KB
Betriebssystem

Benutzer

Überblick: Speicherhierarchie



Überblick: Speicherhierarchie



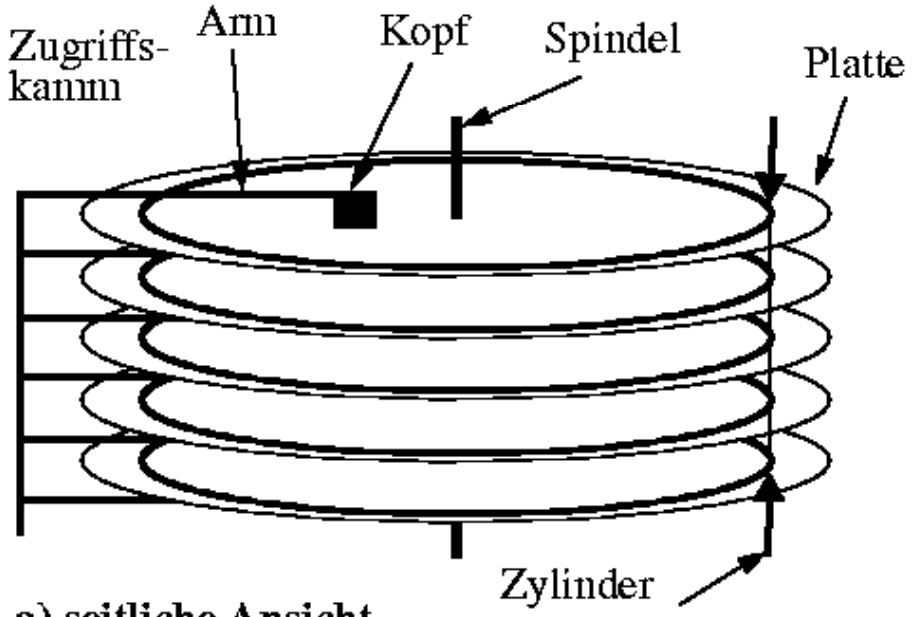
Magnetplattenspeicher

Aufbau

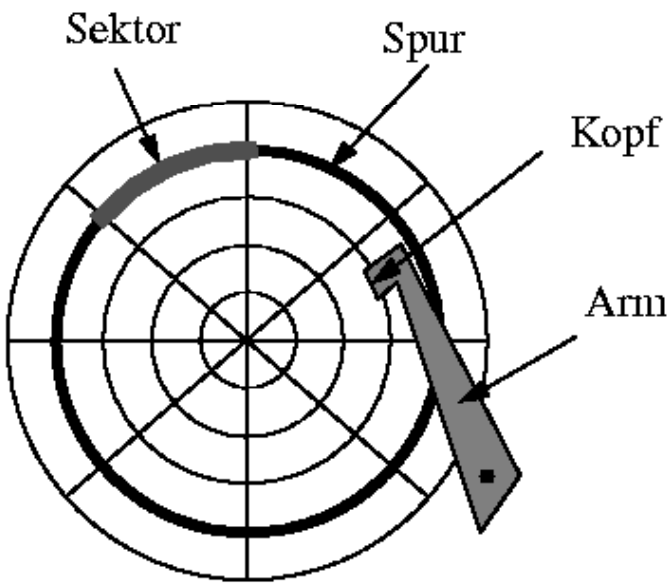
- mehrere gleichförmig rotierende Platten, für jede Plattenoberfläche ein Schreib-/Lesekopf
- jede Plattenoberfläche ist eingeteilt in Spuren
- die Spuren sind formatiert als Sektoren fester Größe (Slots)
- Sektoren (typischerweise 1 - 8 KB) sind die kleinste Schreib-/Leseeinheit auf einer Platte

Adressierung

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



a) seitliche Ansicht



b) Draufsicht

Magnetplatten: Technische Merkmale

Merkmal / Magnetplattentyp	typische Werte 1998	IBM 3390 (1990)	IBM 3380 (1985)	IBM 3330 (1970)
t_{smin} = Zugr.bewegung(Min)	1 ms	k. A.	2 ms	10 ms
t_{sav} = " (Mittel)	8 ms	12.5 ms	16 ms	30 ms
t_{smax} = " (Max.)	16 ms	k. A.	29 ms	55 ms
t_r = Umdrehungszeit	6 ms	14.1.ms	16.7 ms	16.7 ms
T_{cap} = Spurkapazität	100 KB	56 KB	47 KB	13 KB
T_{cyl} = #Spuren pro Zyl.	20	15	15	19
N_{dev} = #Zylinder	5000	2226	2655	411
u = Transferrate	15 MB/s	4.2 MB/s	3 MB/s	0. 8 MB/s
Nettokapazität	10 GB	1.89 GB	1.89 GB	0.094 GB

Typische Werte in 2000:

- 5 - 100 GB Kapazität, 10 - 30 MB/s
- 2 - 6 ms Umdrehungszeit, 5 - 10 ms seek
- 20 $ / GB (SCSI-Platten) bzw. 7 $ / GB (IDE-Platten)

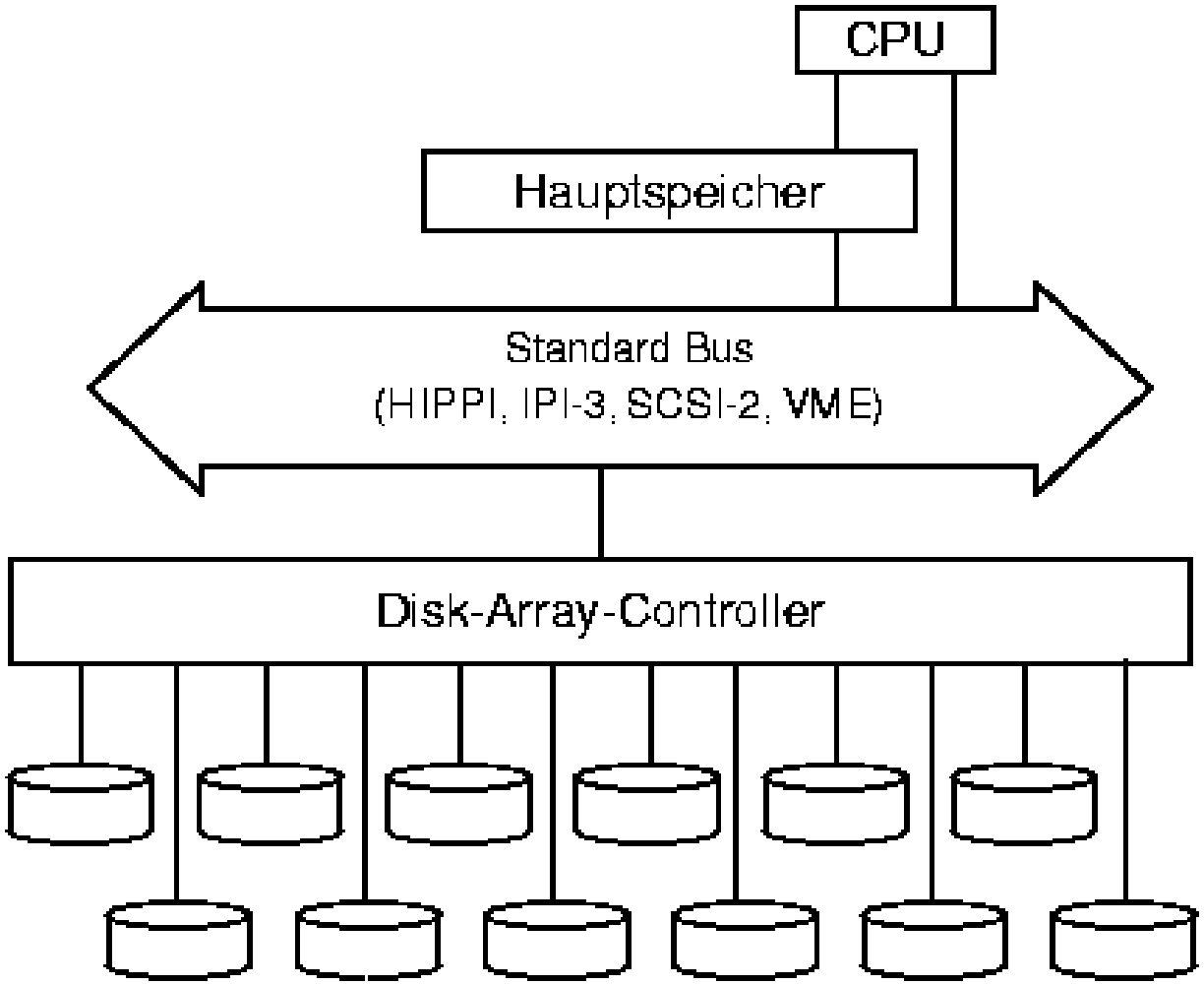
Lesen von Daten von der Platte

- 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

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 !

Disk Arrays ➔ RAID-Systeme



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

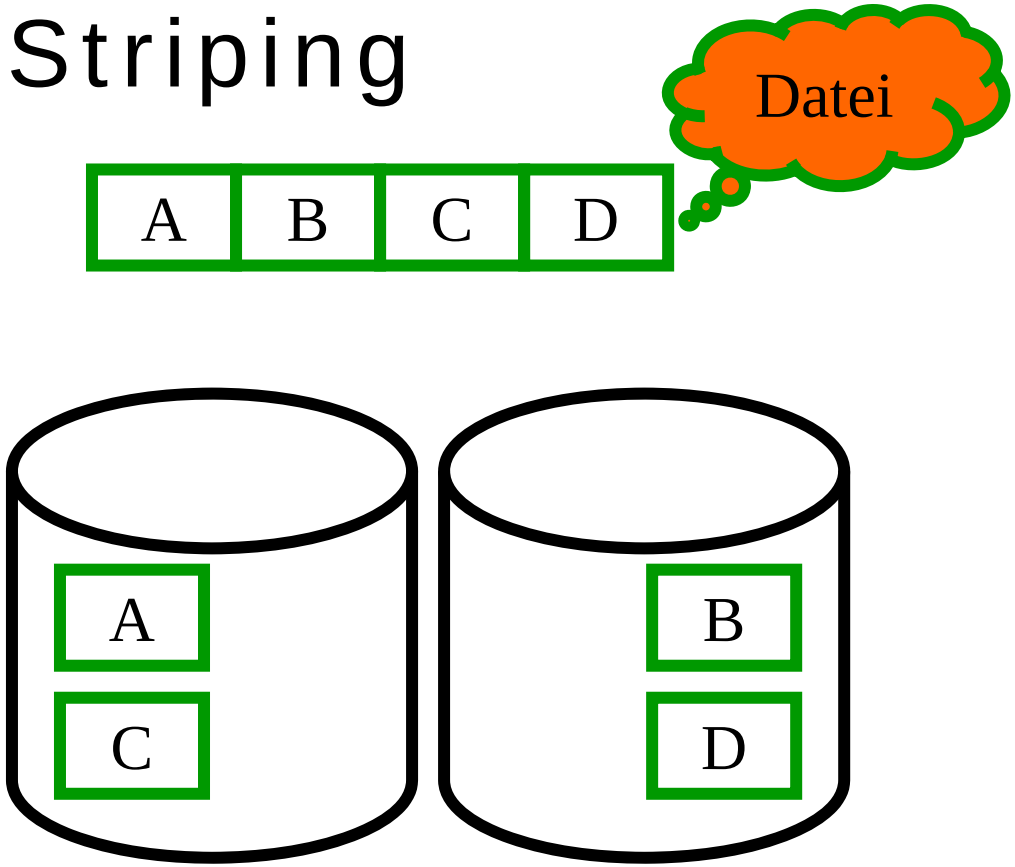
- Mean Time To Failure (MTTF): Erwartungswert für die Zeit (von der Inbetriebnahme) bis zum Ausfall einer Platte
- Mean Time To Repair (MTTR): Erwartungswert für die Zeit zur Ersetzung der Platte und der Rekonstruktion der Daten
- Mean Time To Data Loss (MTTDL): Erwartungswert für die Zeit bis zu einem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

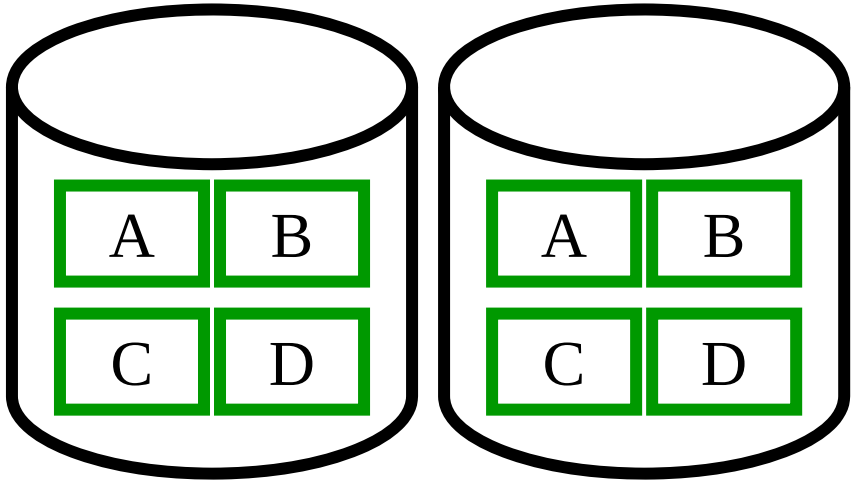
- durch Replikation der Daten (z. B. Spiegelplatten) - RAID1
- durch zusätzlich zu den Daten gespeicherte Error-Correcting-Codes (ECCs), z.B. Paritätsbits (RAID-4, RAID-5)

RAID 0: Striping



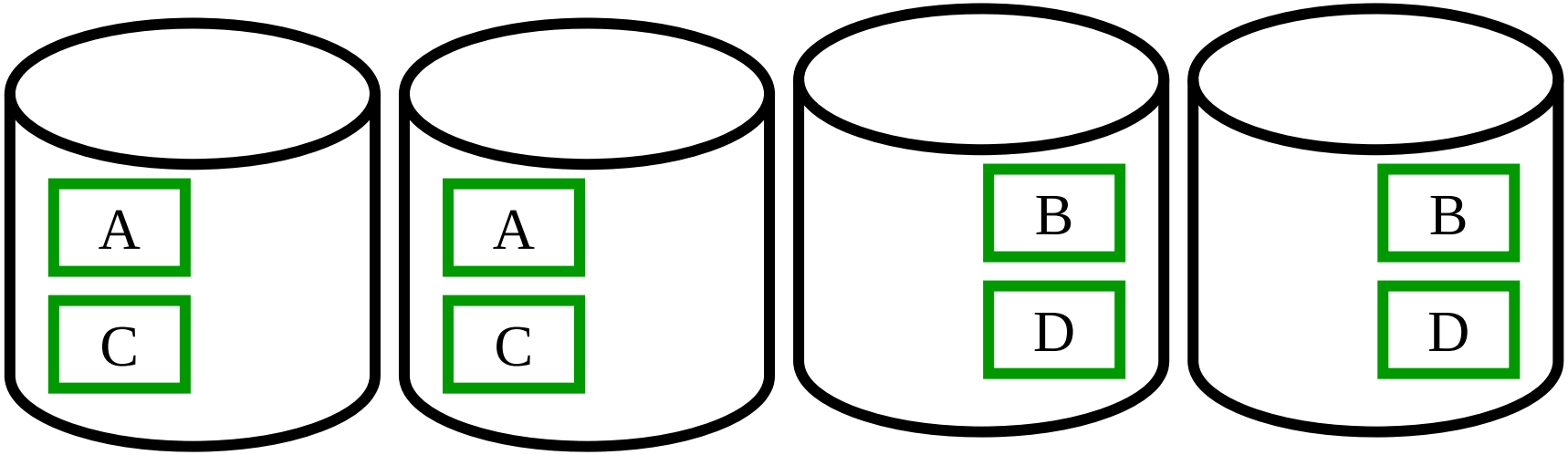
- Lastbalancierung wenn alle Blöcke mit gleicher Häufigkeit gelesen/geschrieben werden
- Doppelte Bandbreite beim sequentiellen Lesen der Datei bestehend aus den Blöcken ABCD...
- Aber: Datenverlust wird immer wahrscheinlicher, je mehr Platten man verwendet (Stripingbreite = Anzahl der Platten, hier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

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

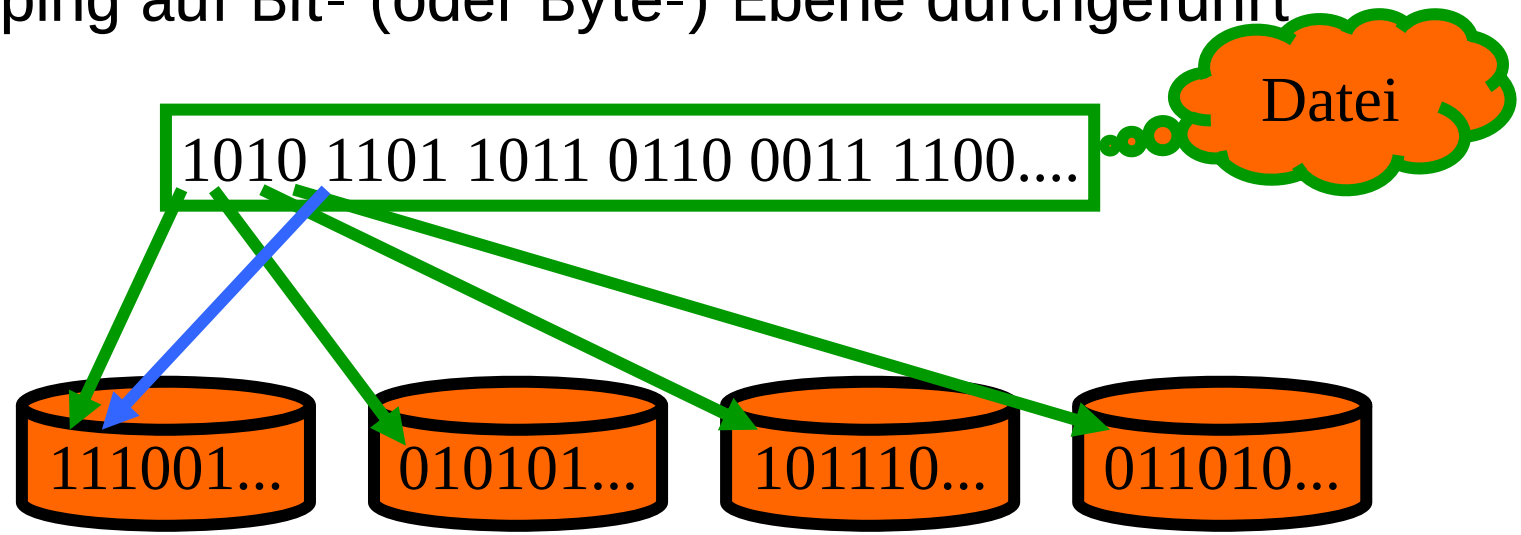
RAID 0+1: Striping und Spiegelung



- 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

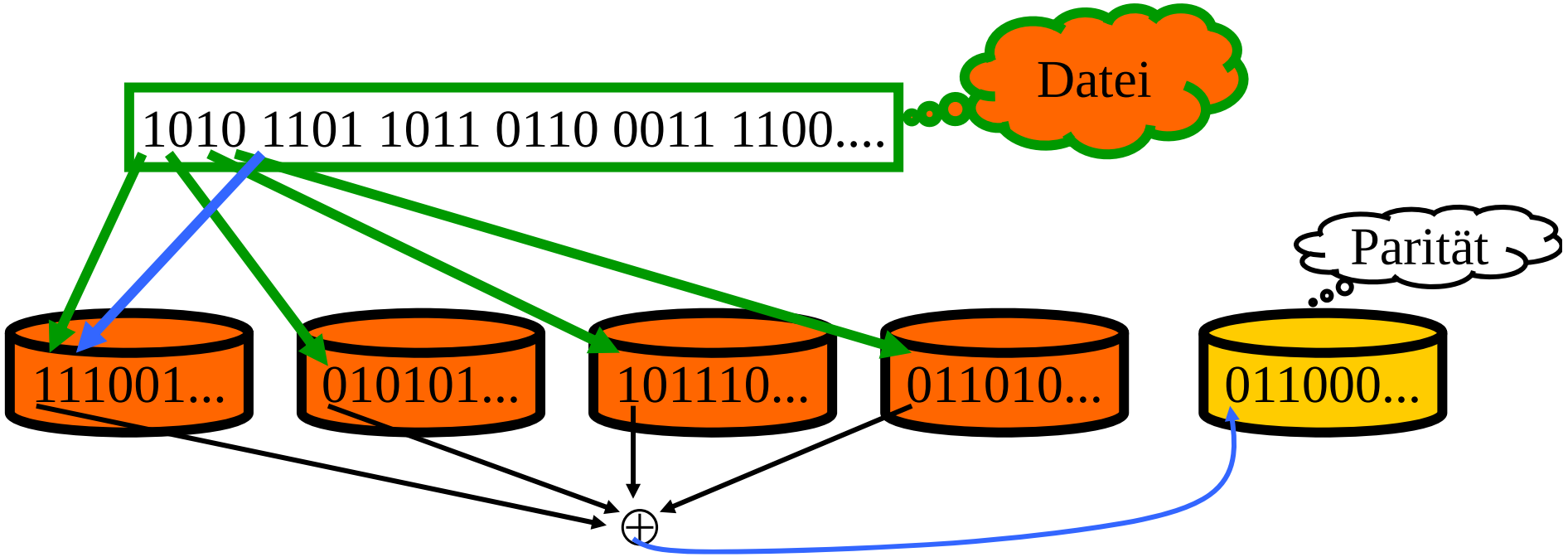
RAID 2: Striping auf Bit-Ebene

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



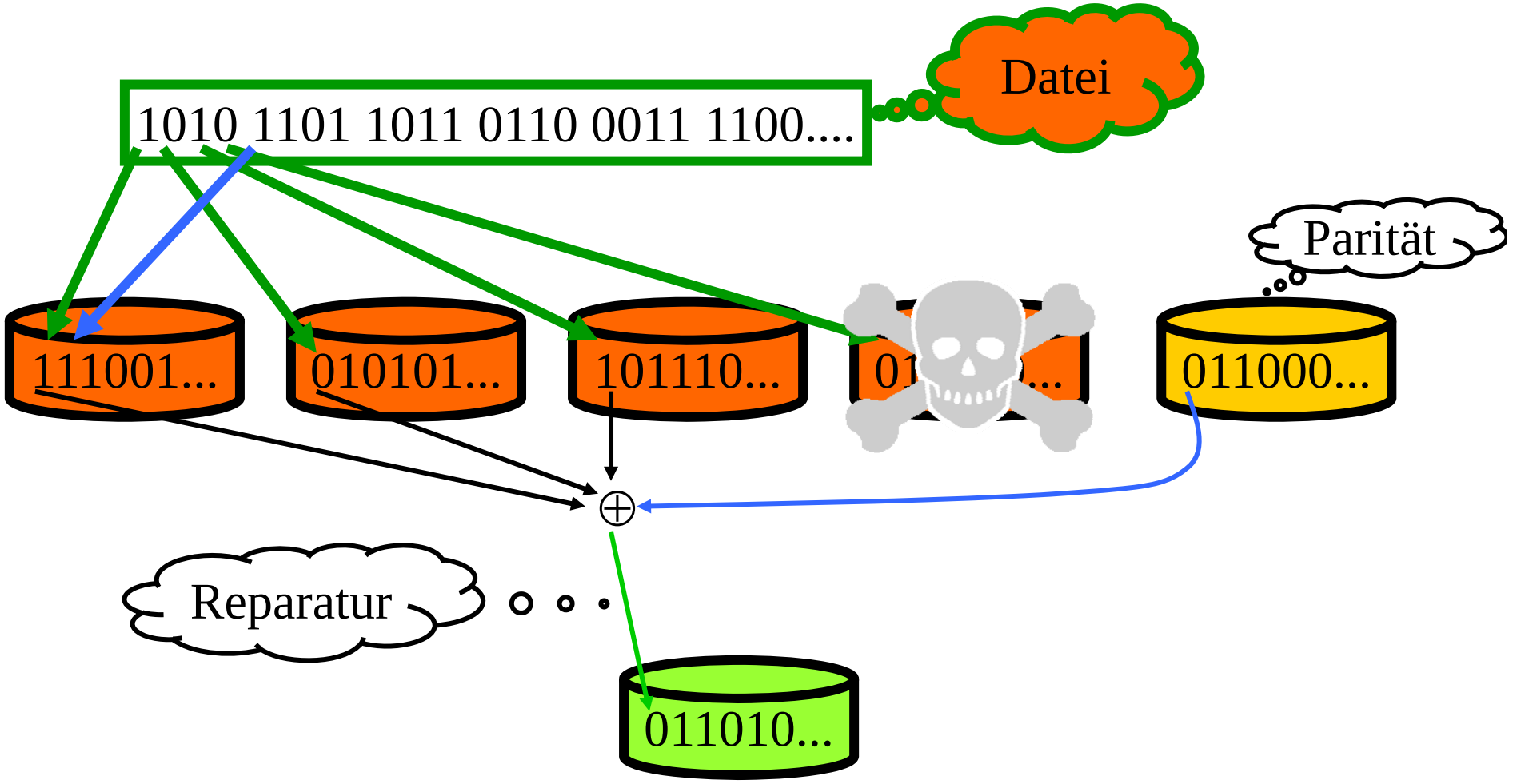
- Es werden zusätzlich auf einer Platte noch Fehlererkennungs- und Korrekturcodes gespeichert
- In der Praxis nicht eingesetzt, da Platten sowieso schon Fehlererkennungscodes verwalten

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

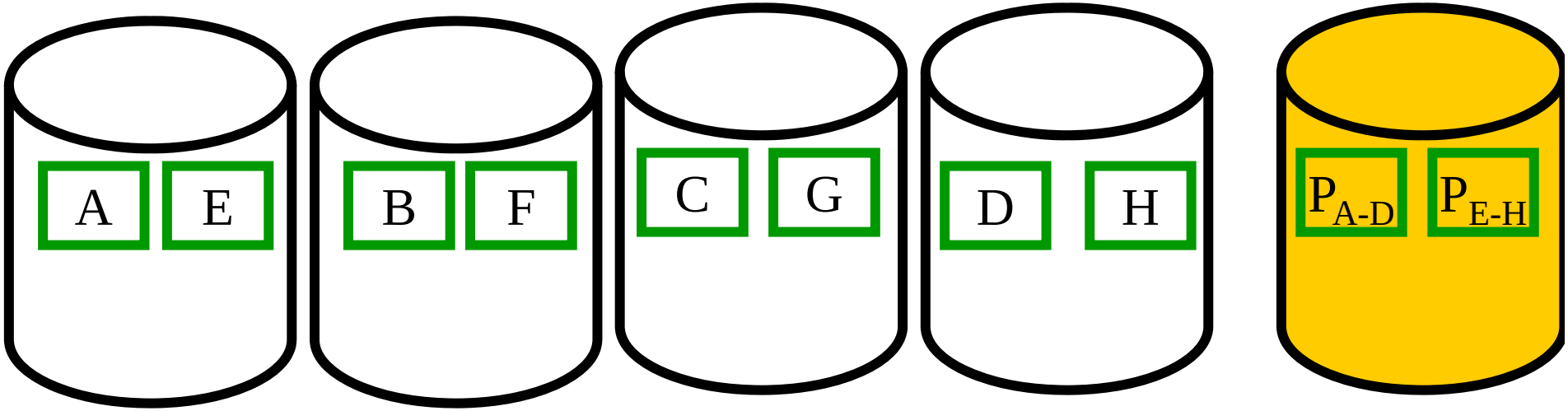


- Das Striping wird auf Bit- (oder Byte-) Ebene durchgeführt
- Es wird auf einer Platte noch die Parität der anderen Platten gespeichert. Parität = bit-weise xor ⊕
- Dadurch ist der Ausfall einer Platte zu kompensieren
- Das Lesen eines Blocks erfordert den Zugriff auf alle Platten
  - Verschwendung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

RAID 3: Plattenausfall

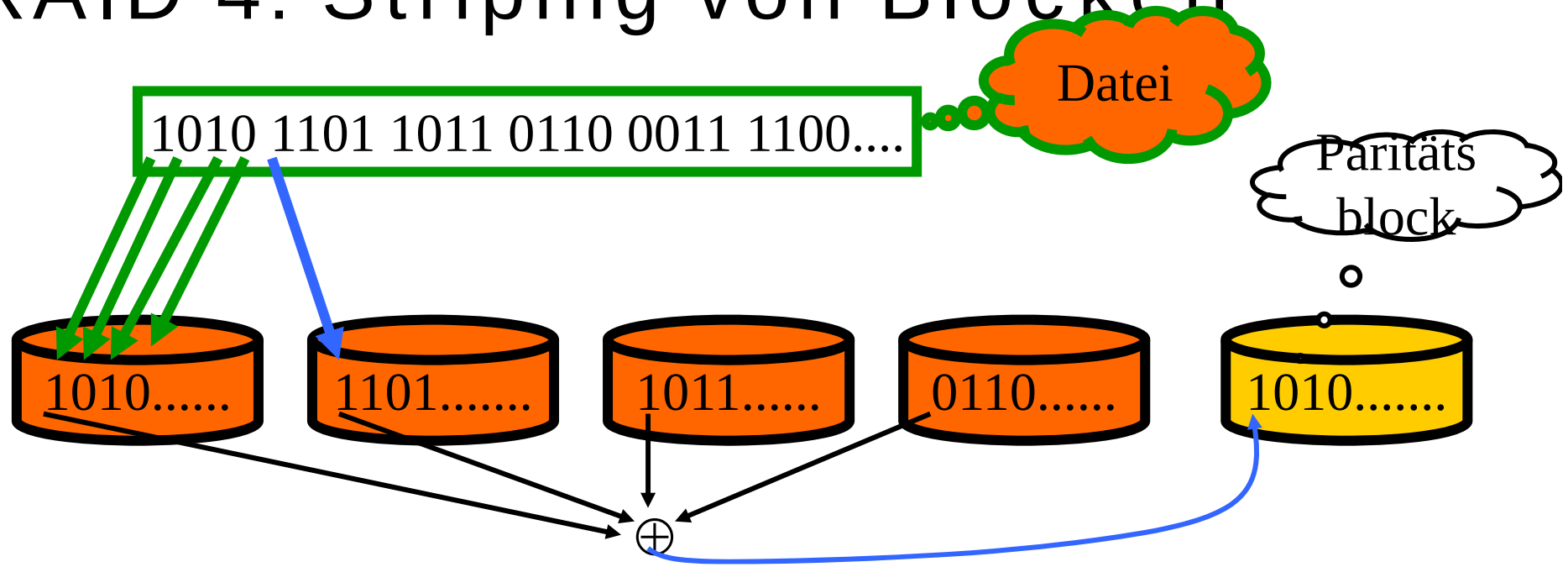


RAID 4: Striping von Blöcken



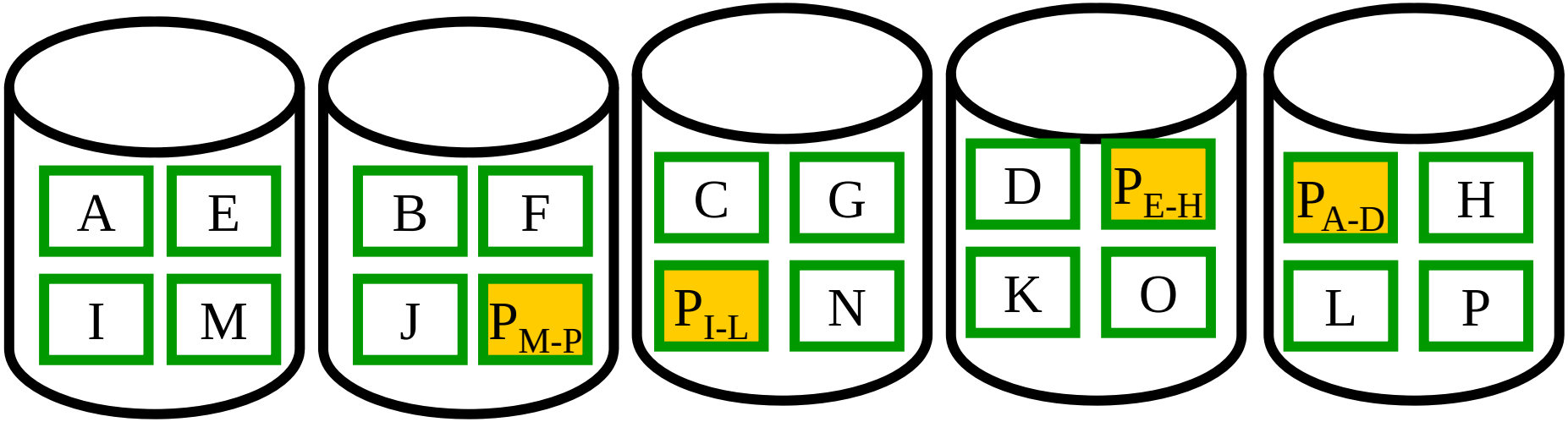
- Bessere Lastbalancierung als bei RAID 3
- Flaschenhals bildet die Paritätsplatte
- Bei jedem Schreiben muss darauf zugegriffen werden
  - Bei Modifikation von Block A zu A‘ wird die Parität P_{A-D} wie folgt neu berechnet:
  - $P'_{A-D} := P_{A-D} \oplus A \oplus A'$
- D.h. bei einer Änderung von Block A muss der alte Zustand von A und der alte Paritätsblock gelesen werden und der neue Paritätsblock und der neue Block A‘ geschrieben werden

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 $P_{A\text{-}D}$ wie folgt neu berechnet:
  - $P'_{A\text{-}D} := P_{A\text{-}D} \oplus A \oplus A'$
- 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

RAID 5: Striping von Blöcken, Verteilung der Paritätsblöcke



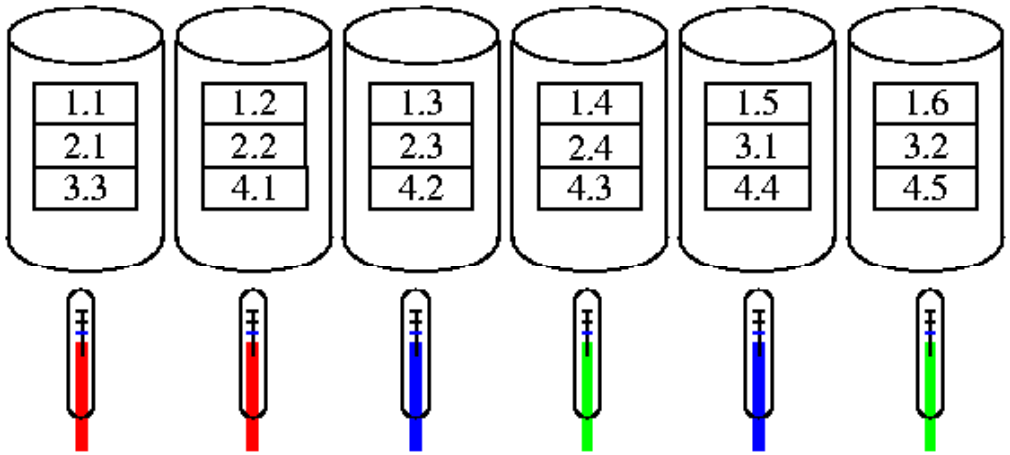
- Bessere Lastbalancierung als bei RAID 4
- die Paritätsplatte bildet jetzt keinen Flaschenhals mehr
- Wird in der Praxis häufig eingesetzt
- Guter Ausgleich zwischen Platzbedarf und Leistungsfähigkeit

Lastbalancierung bei der Blockabbildung auf die Platten

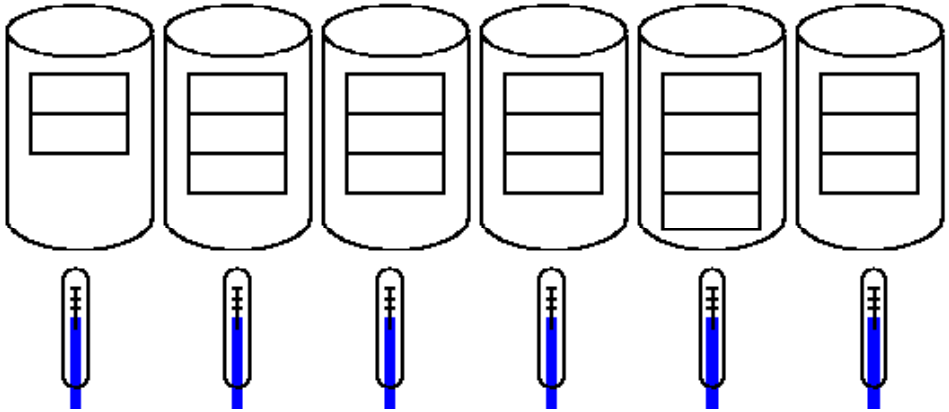
Vergleich von Greedy-Verfahren und Round-Robin-Allokation

Datei1	1.1	1.2	1.3	1.4	1.5	1.6
Hitze:	10	4	4	3	2	1
Datei 2	2.1	2.2	2.3	2.4		
Hitze:	8	5	5	1		
Datei 3	3.1	3.2	3.3			
Hitze:	5	5	5			
Datei 4	4.1	4.2	4.3	4.4	4.5	
Hitze:	7	4	3	2	1	

Round-Robin-Allokation



Greedy-Methode

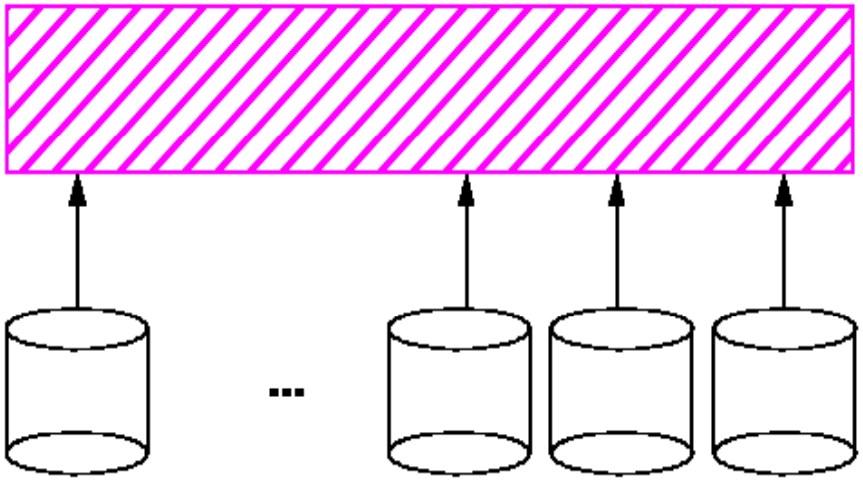


Parallelität bei Lese/Schreib-Aufträgen

Voraussetzung: ***Declustering*** von Dateien über mehrere Platten

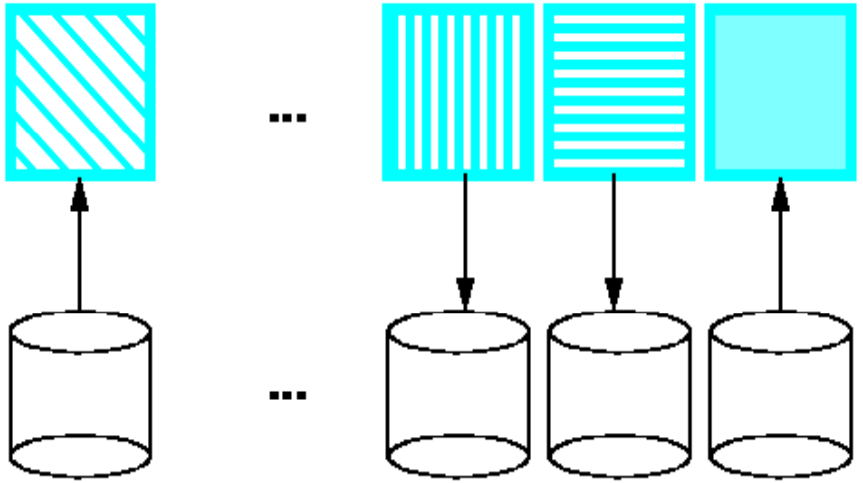
2 generelle Arten von E/A-Parallelität

Intra-E/A-Parallelität (Zugriffsparallelität)



1 E/A-Auftrag wird in mehrere, parallel ausführbare Plattenzugriffe umgesetzt

Inter-E/A-Parallelität (Auftragsparallelität)

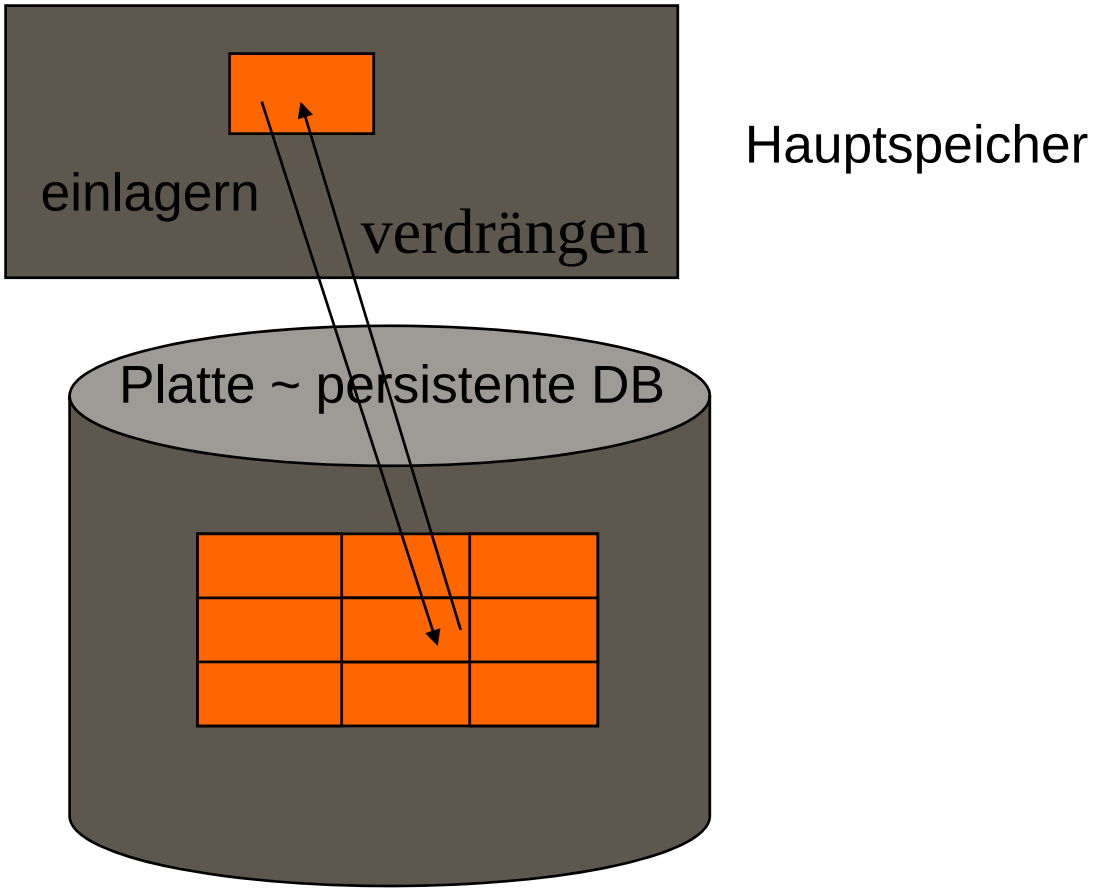


Mehrere unabhängige E/A-Aufträge können parallel ausgeführt werden, sofern die betreffenden Daten über verschiedene Platten verteilt sind

Bewertung der Parallelität bei RAID

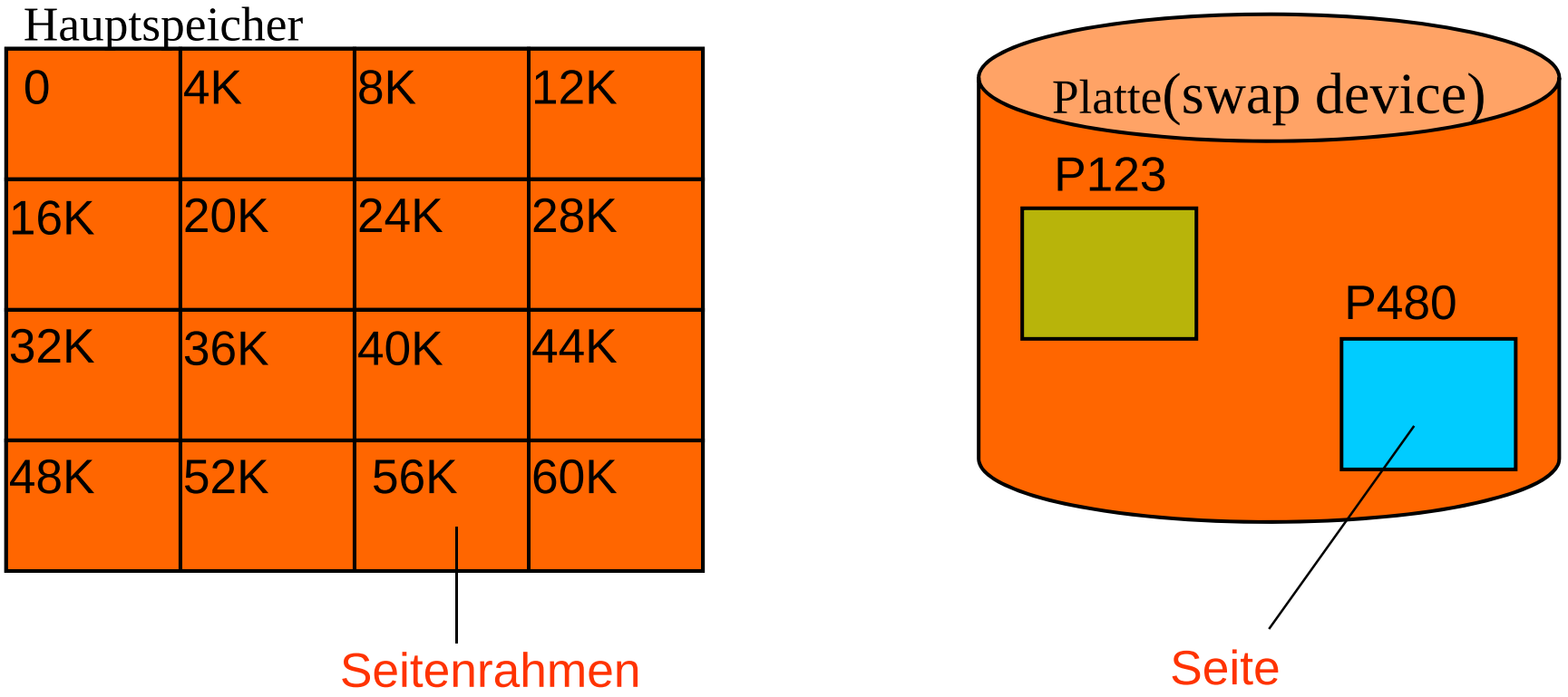
- RAID 0
  - ?
- RAID 1
  - ?
- RAID 0+1
  - ?
- RAID 3
  - ?
- RAID 4
  - ?
- RAID 5
  - ?

Systempuffer-Verwaltung

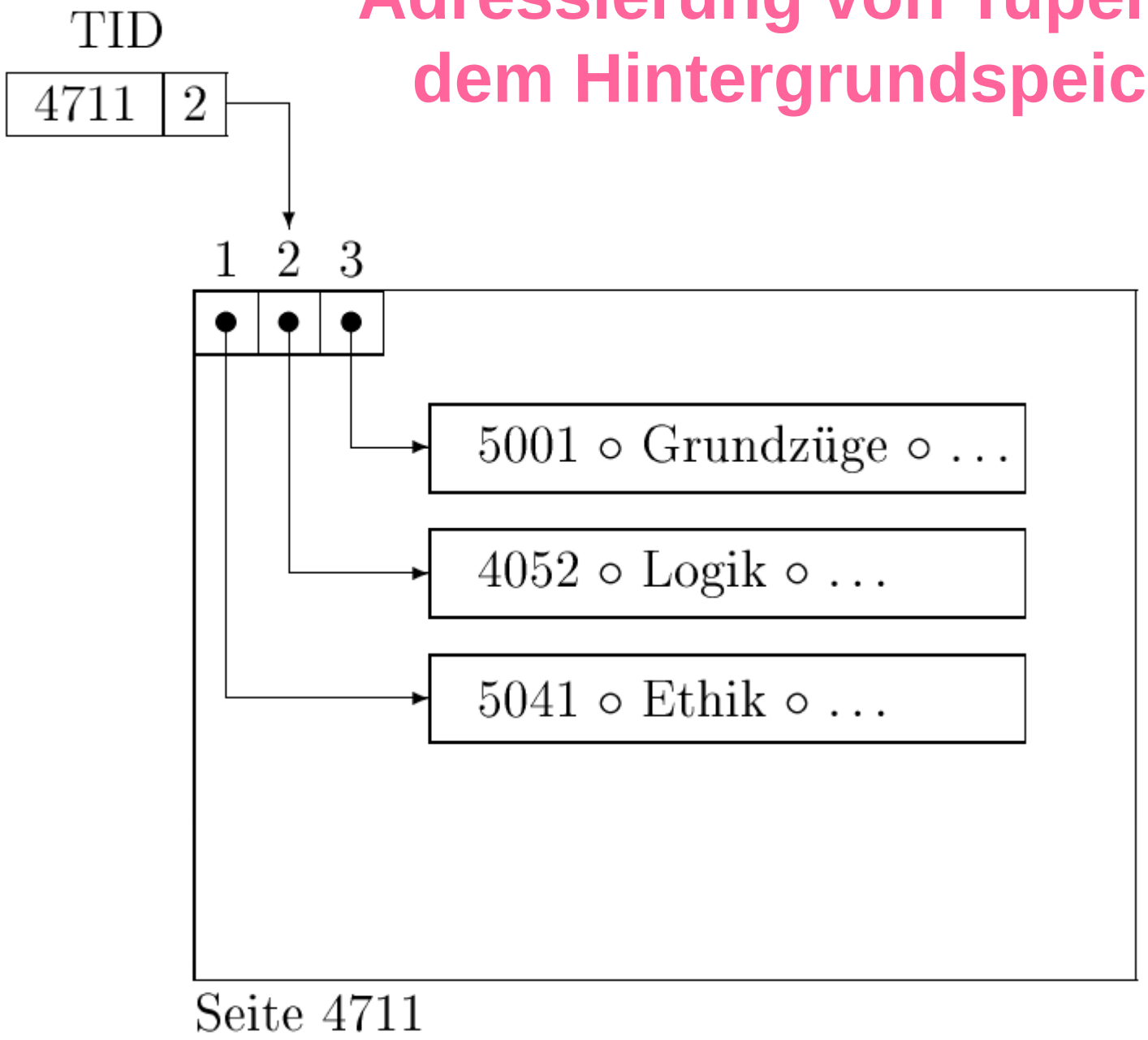


Ein- und Auslagern von Seiten

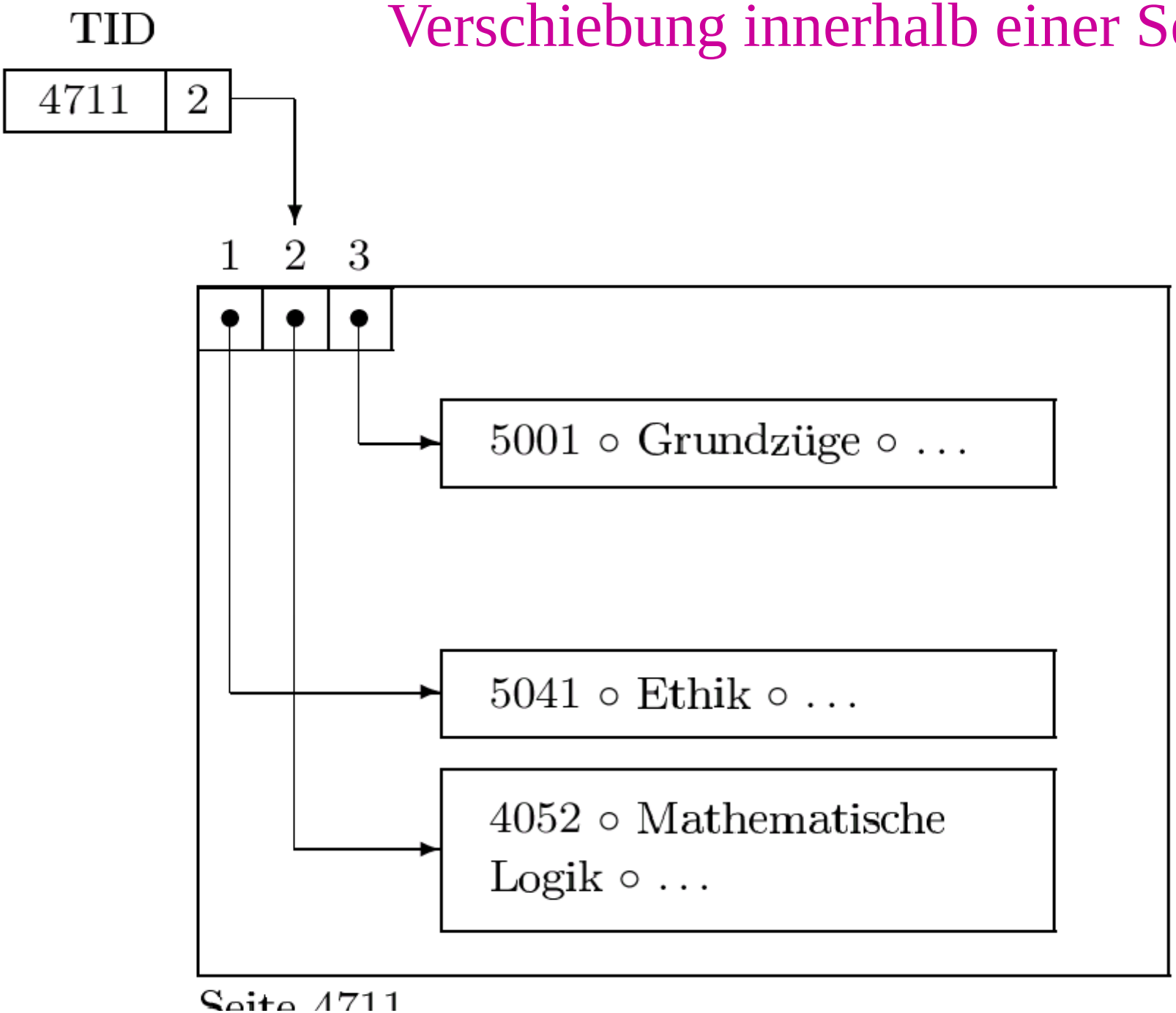
- Systempuffer ist in Seitenrahmen gleicher Größe aufgeteilt
- Ein Rahmen kann eine Seite aufnehmen
- „Überzählige“ Seiten werden auf die Platte ausgelagert



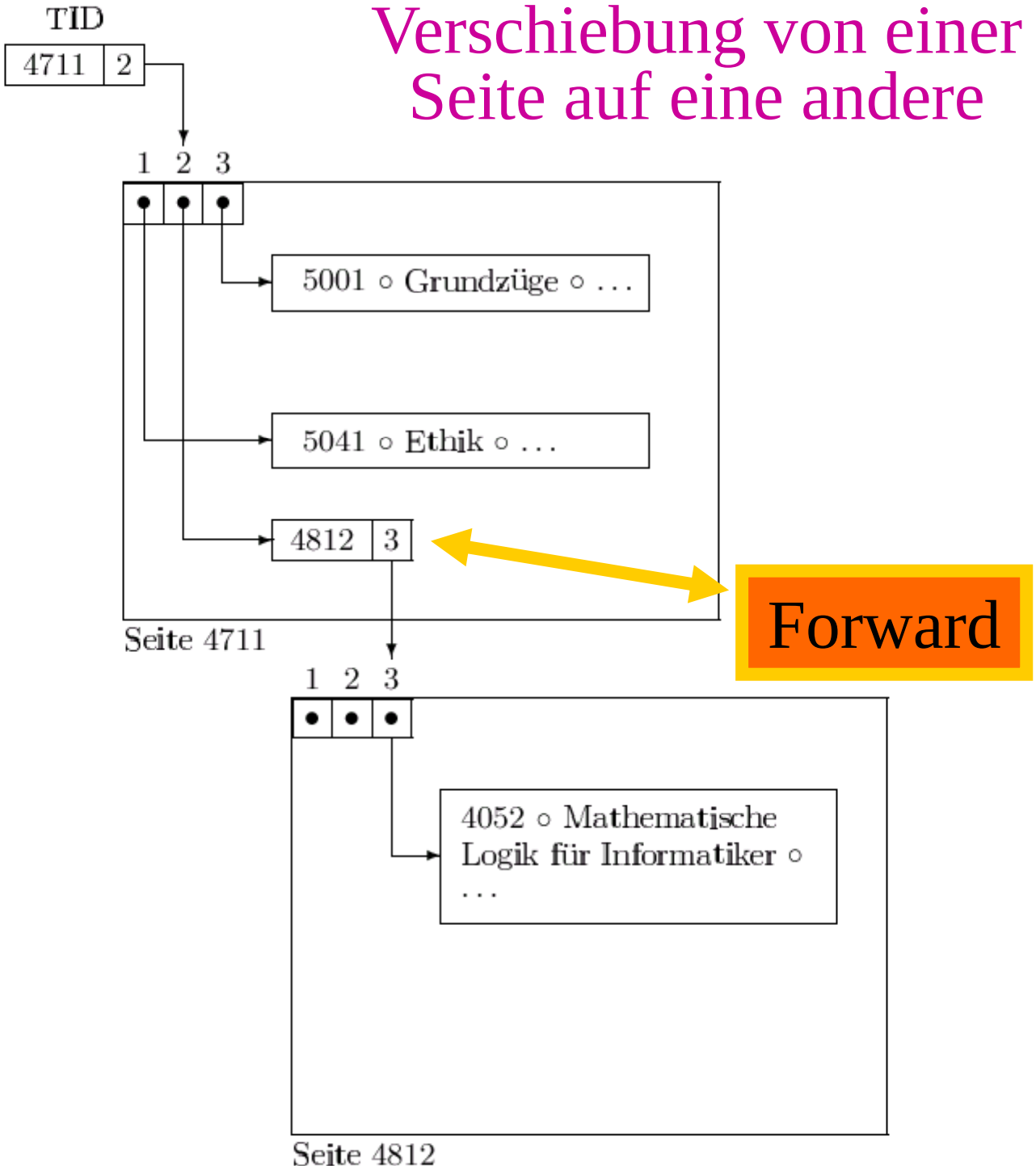
Adressierung von Tupeln auf dem Hintergrundspeicher



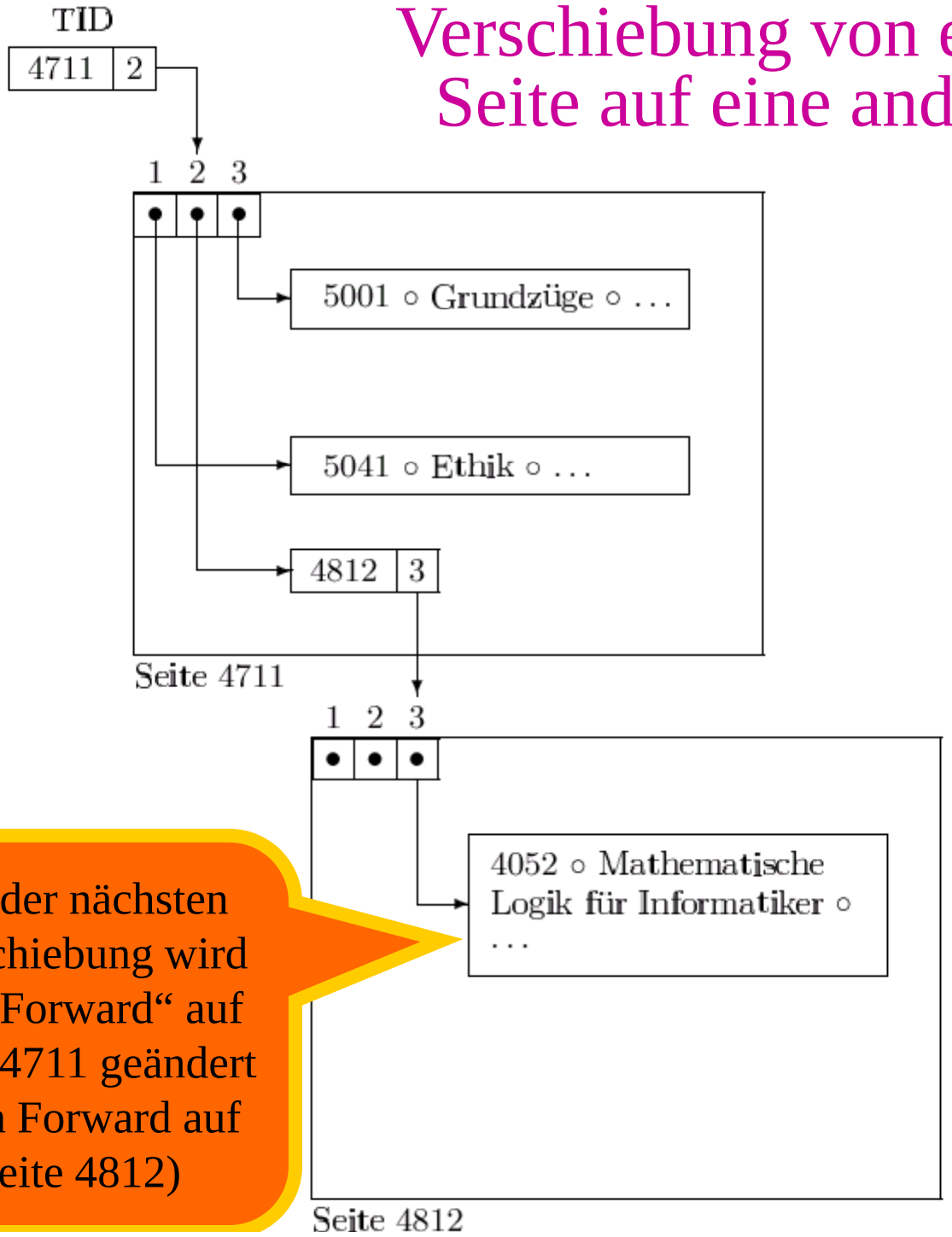
Verschiebung innerhalb einer Seite



Verschiebung von einer Seite auf eine andere



Verschiebung von einer Seite auf eine andere



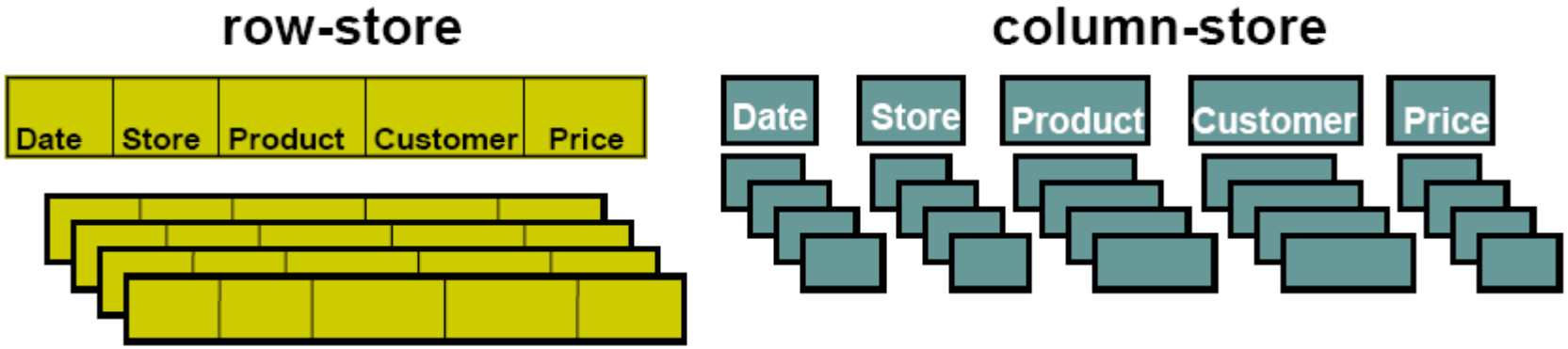
Bei der nächsten Verschiebung wird der „Forward“ auf Seite 4711 geändert (kein Forward auf Seite 4812)

Neue Entwicklungen

- Hauptspeicher-Datenbanksysteme
  - Times Ten
  - Transact in Memory
  - Monet DB
  - TREX von SAP
- Columns Store versus Row Store
  - C-Store / Vertica
  - Monet
  - TREX

Re-use permitted when acknowledging the original © Stavros Harizopoulos, Daniel Abadi, Peter Boncz (2009)

What is a column-store?



+ easy to add/modify a record

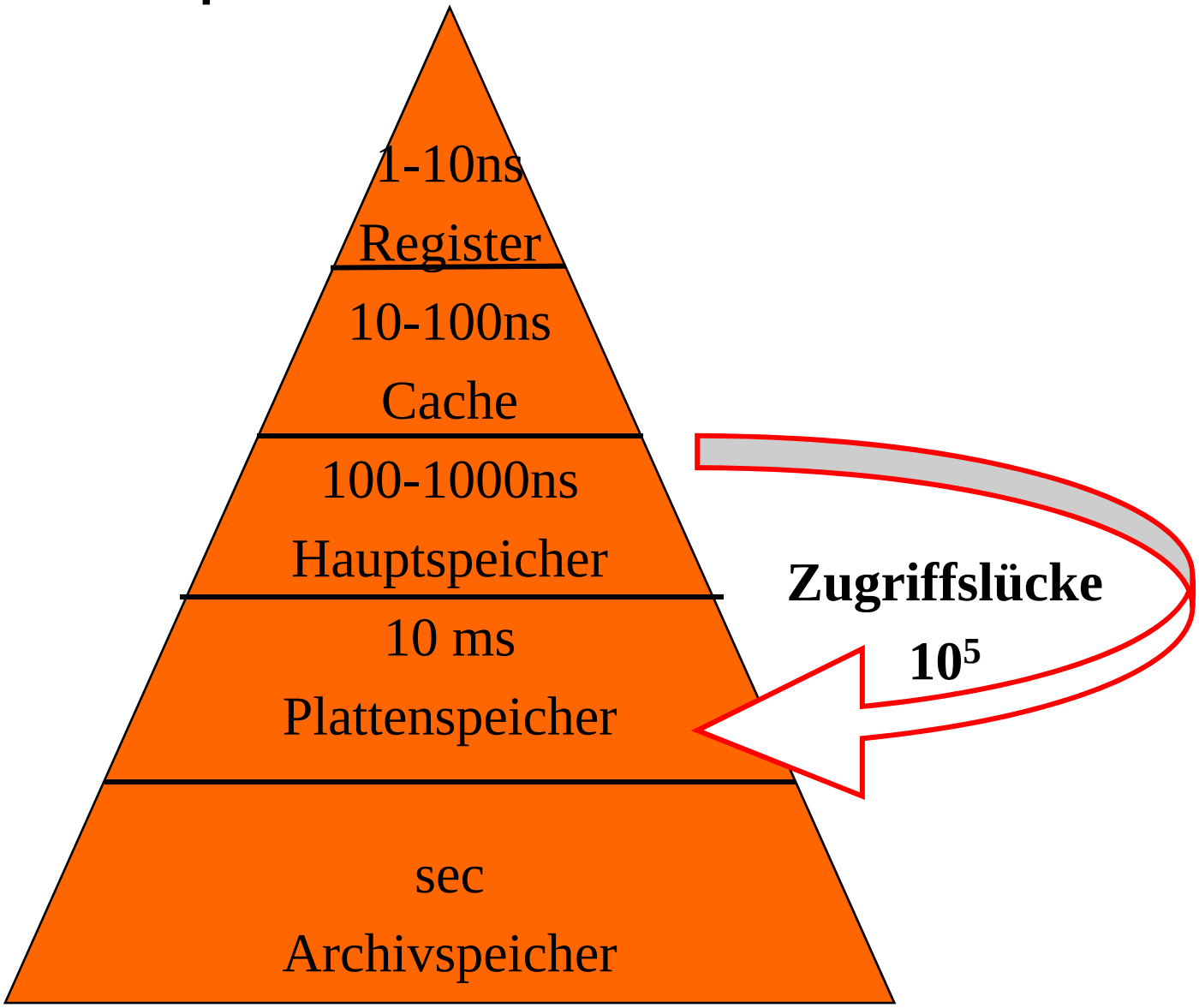
- might read in unnecessary data

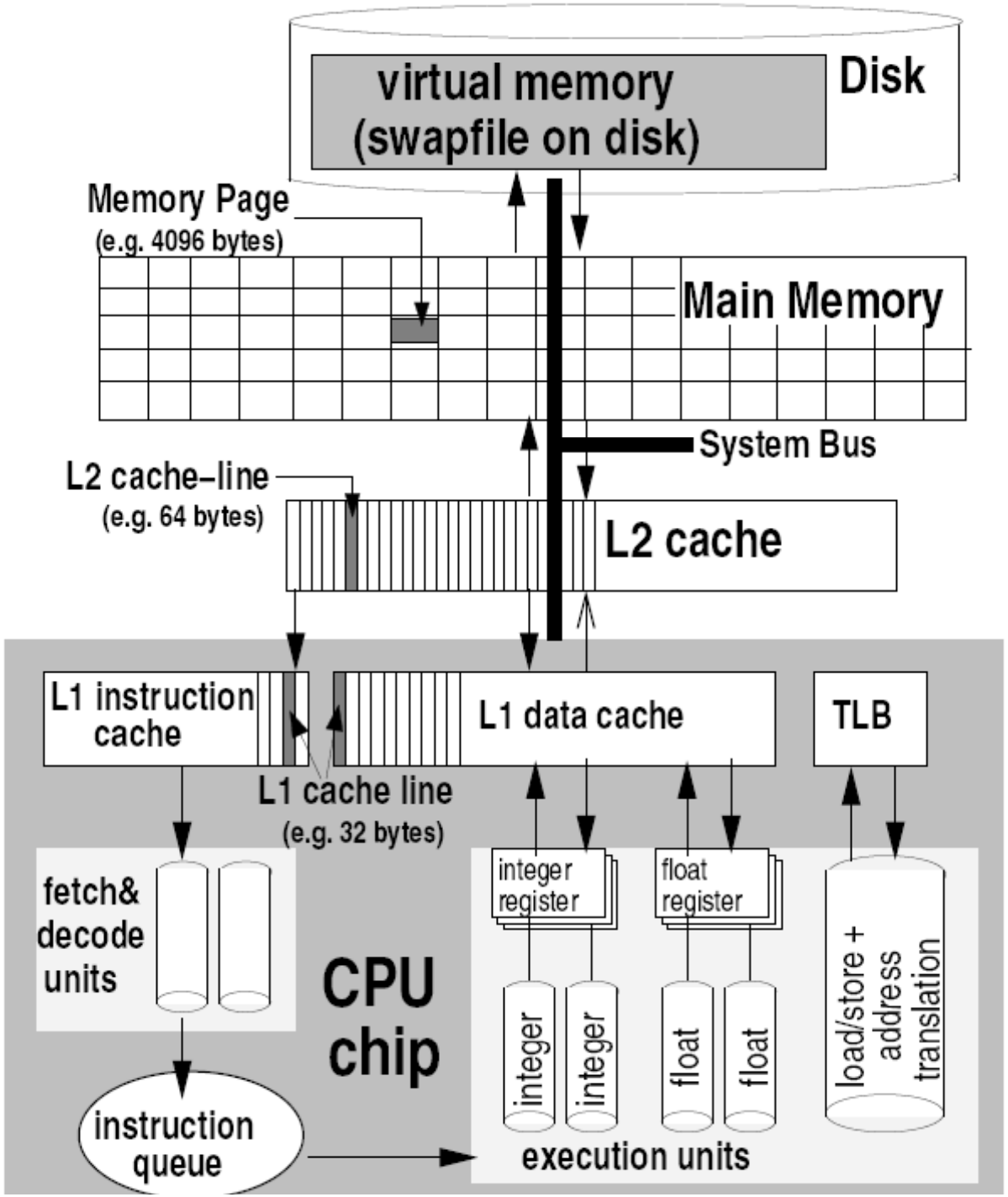
+ only need to read in relevant data

- tuple writes require multiple accesses

=> suitable for read-mostly, read-intensive, large data repositories

Überblick: Speicherhierarchie




virtual memory
(swapfile on disk)
Disk
Memory Page
(e.g. 4096 bytes)
Main Memory
System Bus
L2 cache–line
(e.g. 64 bytes)
L2 cache
L1 instruction
cache
L1 data cache
TLB
L1 cache line
(e.g. 32 bytes)
fetch&
decode
units
CPU
chip
integer
register
float
register
load/store +
address
translation
integer
integer
float
float
instruction
queue
execution units

Row Store versus Column Store

Verkäufe				
Produkt	Kunde	Preis	Filiale	...
Handy	Kemper	345	Schwabing	...
Radio	Mickey	123	Bogenhausen	...
Handy	Minnie	233	Schwabing	...
Kühlschrank	Urmel	240	Augsburg	...
Beamer	Bond	740	London	...
Handy	Lucie	321	Bogenhausen	...

Row Store versus Column Store

Produkt	
ID	Produkt
0	Handy
1	Radio
2	Handy
3	Kühlschrank
4	Beamer
5	Handy

Kunde	
ID	Kunde
0	Kemper
1	Mickey
2	Minnie
3	Urmel
4	Bond
5	Lucie

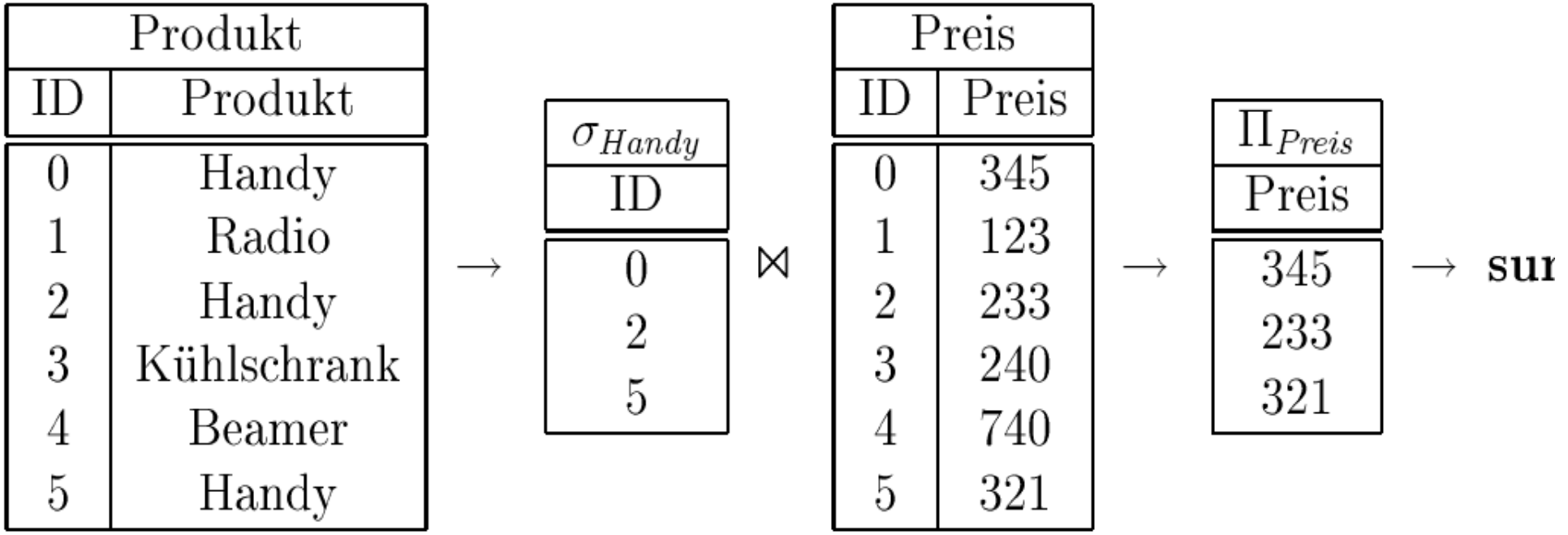
Preis	
ID	Preis
0	345
1	123
2	233
3	240
4	740
5	321

Filiale	
ID	Filiale
0	Schwabing
1	Bogenhausen
2	Schwabing
3	Augsburg
4	London
5	Bogenhausen

Anfragebearbeitung

select sum(Preis)
from Verkäufe
where Produkt = 'Handy'

Die schrittweise Abarbeitung dieser Anfrage ist nachfolgend illustriert



Komprimierung

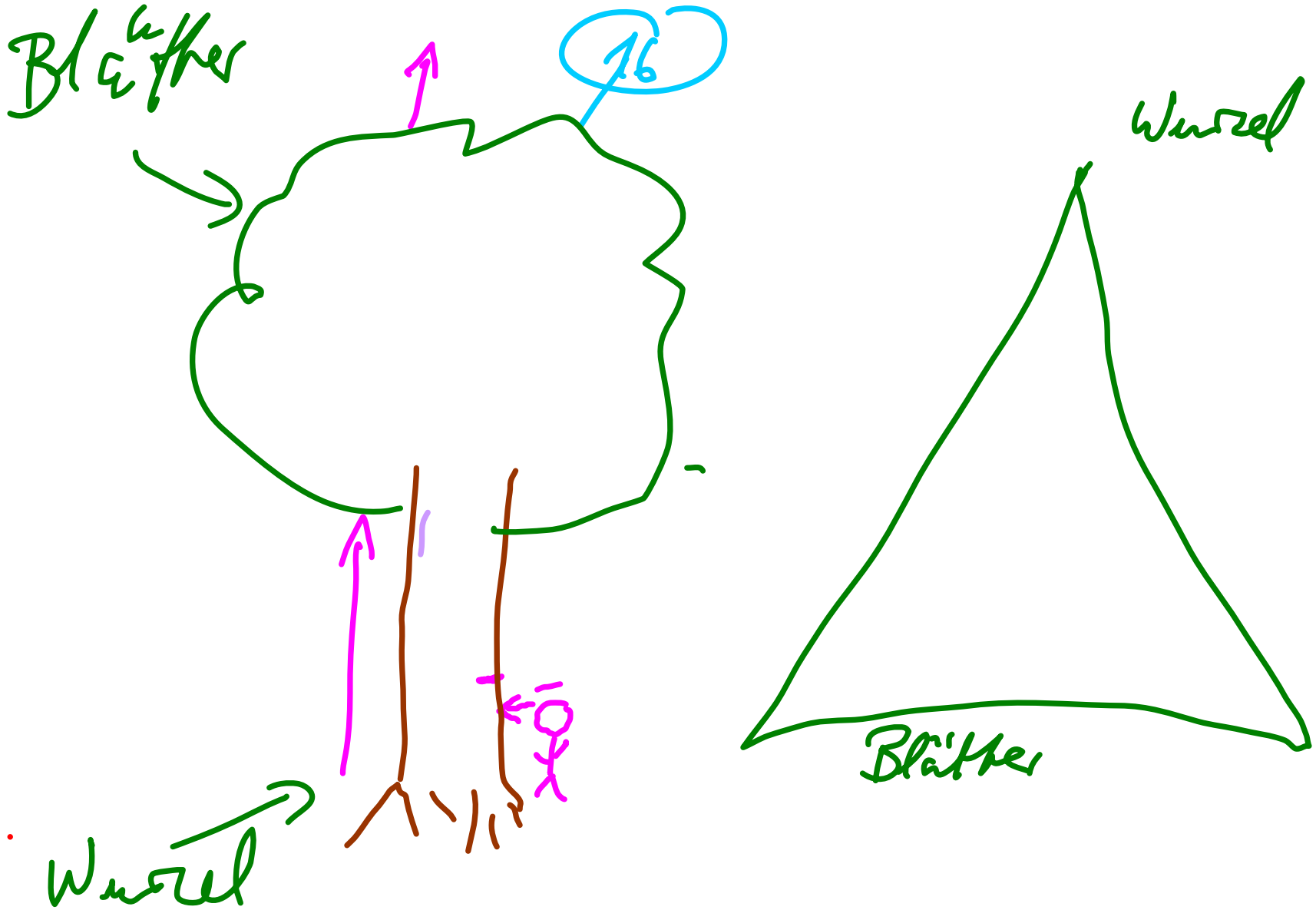
Dictionary	
ID	Wort
0	Augsburg
1	Beamer
2	Bogenhausen
3	Handy
4	Kühlschrank
5	London
6	Radio
7	Schwabing
...	...

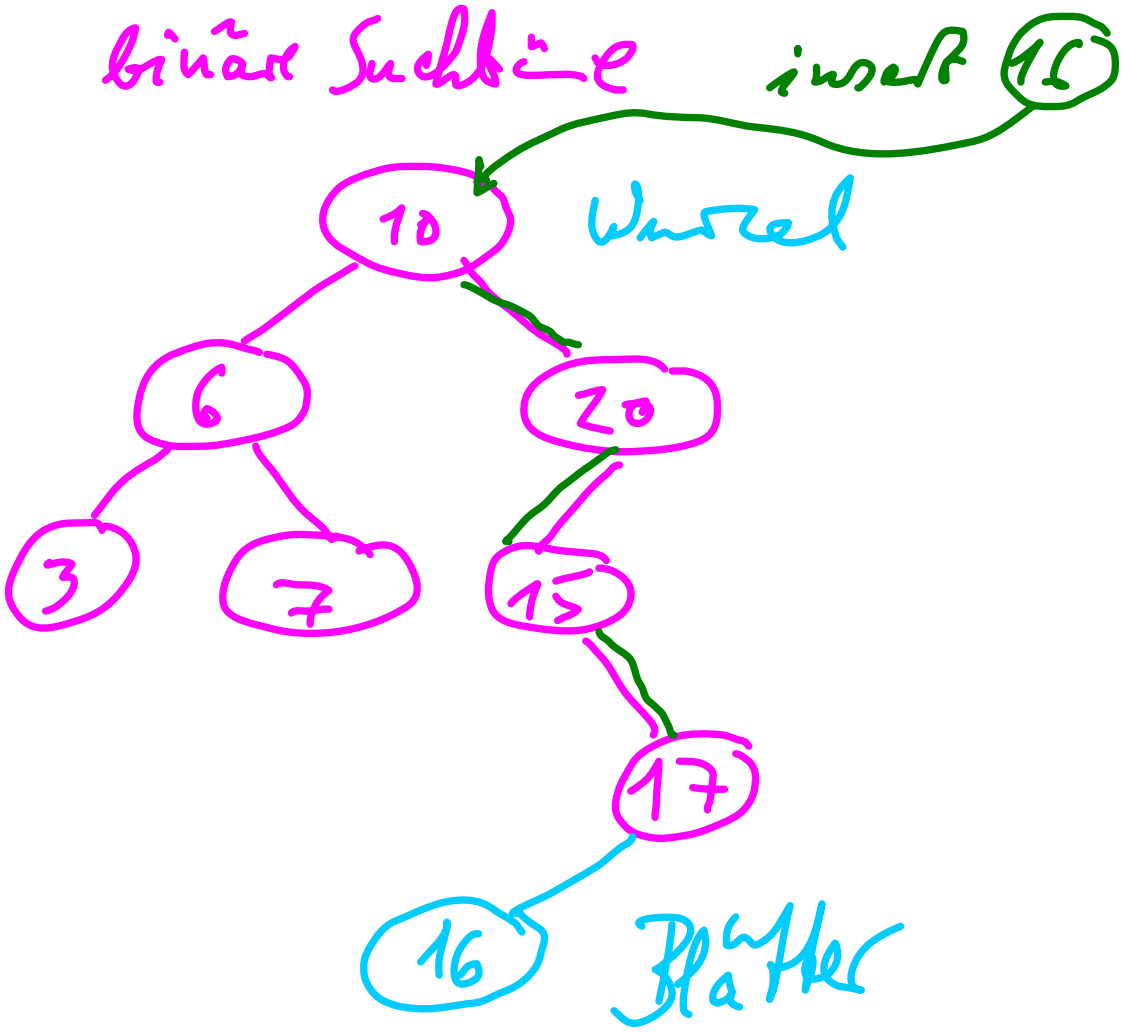
Produkt	
ID	Produkt
0	3
1	6
2	3
3	4
4	1
5	3

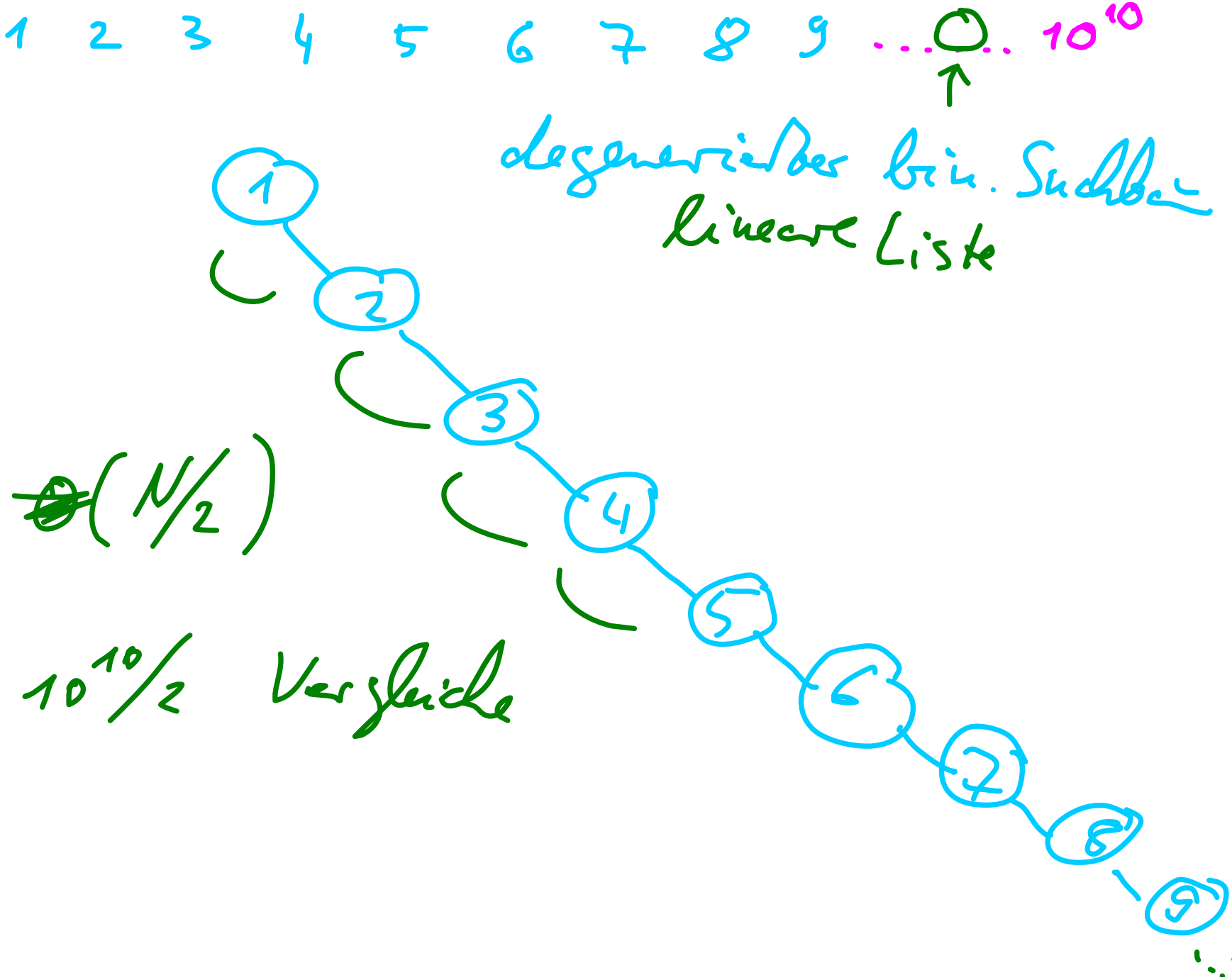
Filiale	
ID	Filiale
0	7
1	2
2	7
3	0
4	5
5	2

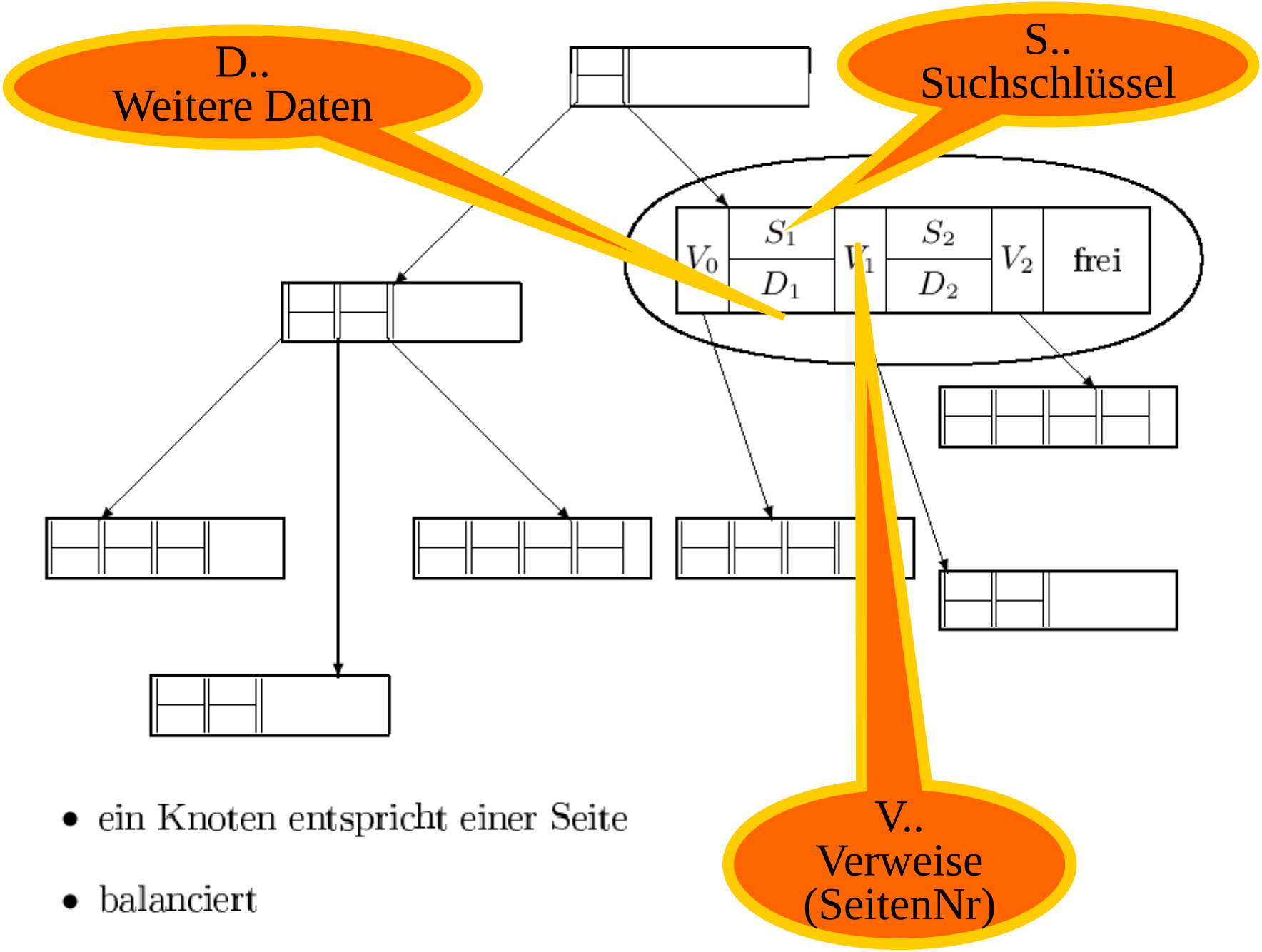
B-Bäume

Balancierte Mehrwege-Suchbäume
Für den Hintergrundspeicher


Blätter
16
Wurzel
Wurzel
Blätter


binäre Suchbäume
insert 15
10
Wurzel
6
20
3
7
15
17
16
Blätter


1 2 3 4 5 6 7 8 9 ... 10^10
degenerierter bin. Suchbaum
lineare Liste
1
2
3
4
5
6
7
8
9
Θ(N/2)
10^10/2 Vergleiche

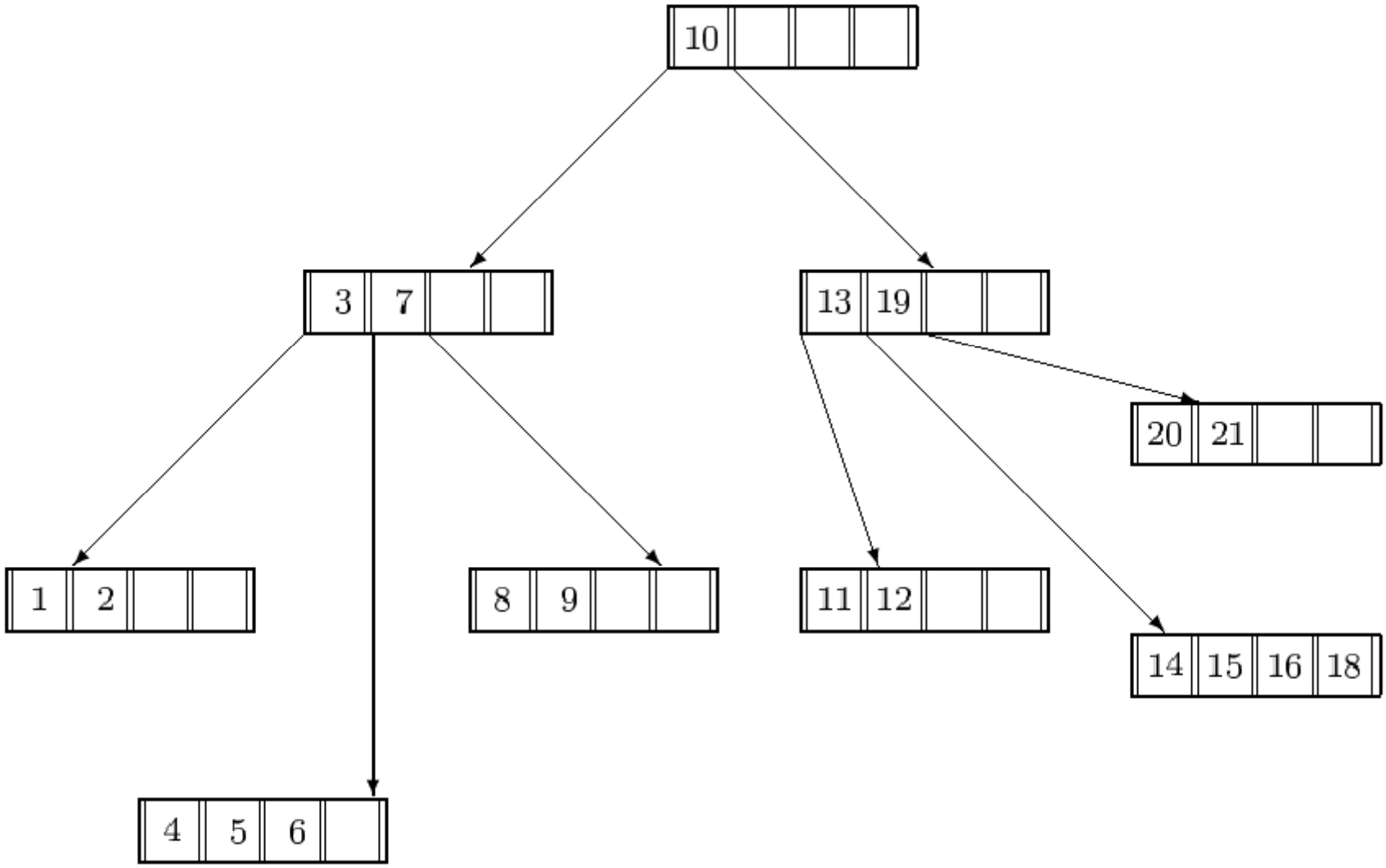


- ein Knoten entspricht einer Seite
- balanciert
- garantierte Auslastung $\geq 50\%$

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

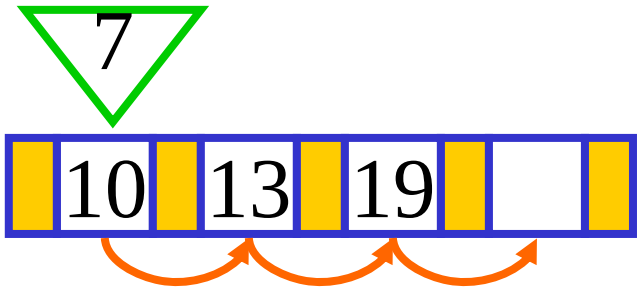
1. Führe eine Suche nach dem Schlüssel durch; diese endet (scheitert) an der Einfügestelle.

2. Füge den Schlüssel dort ein.

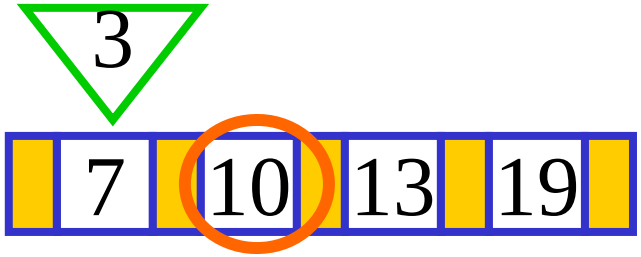
3. Ist der Knoten überfüllt, teile ihn
   - Lege einen neuen Knoten an und belege ihn mit den Schlüsseln, die rechts vom mittleren Eintrag des überfüllten Knotens liegen.
   - Füge den mittleren Eintrag im Vaterknoten des überfüllten Knotens ein.
   - Verbinde den Verweis rechts des neuen Eintrags im 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.

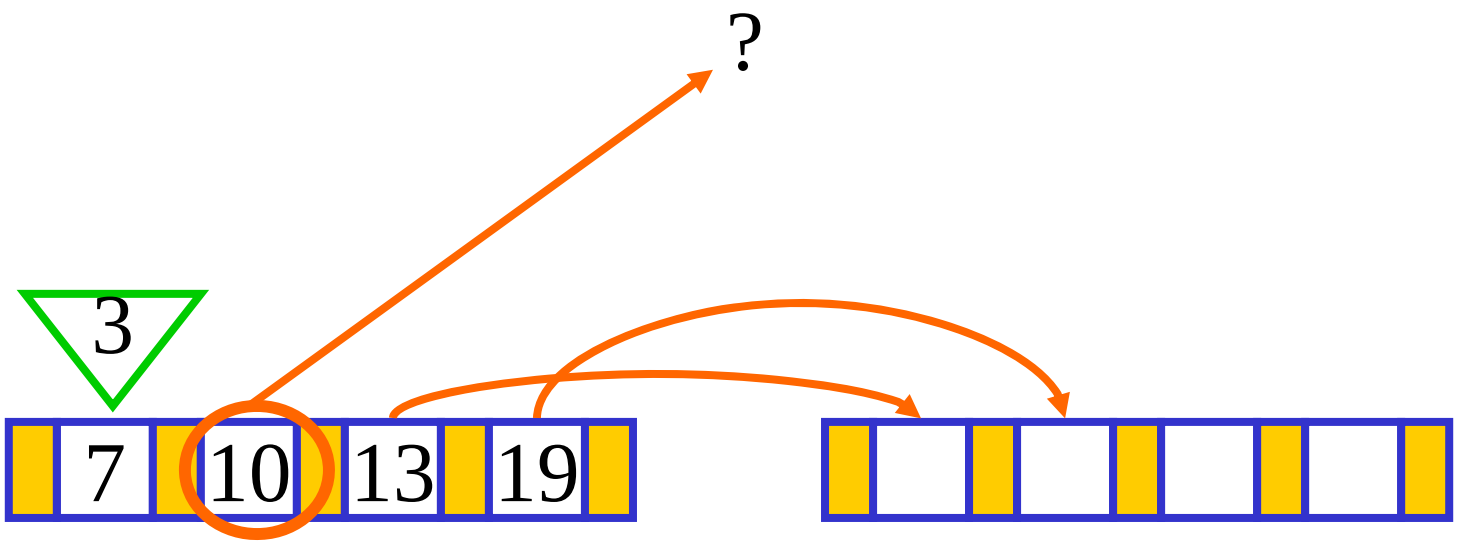
Sukzessiver Aufbau eines B-Baums vom Grad k=2



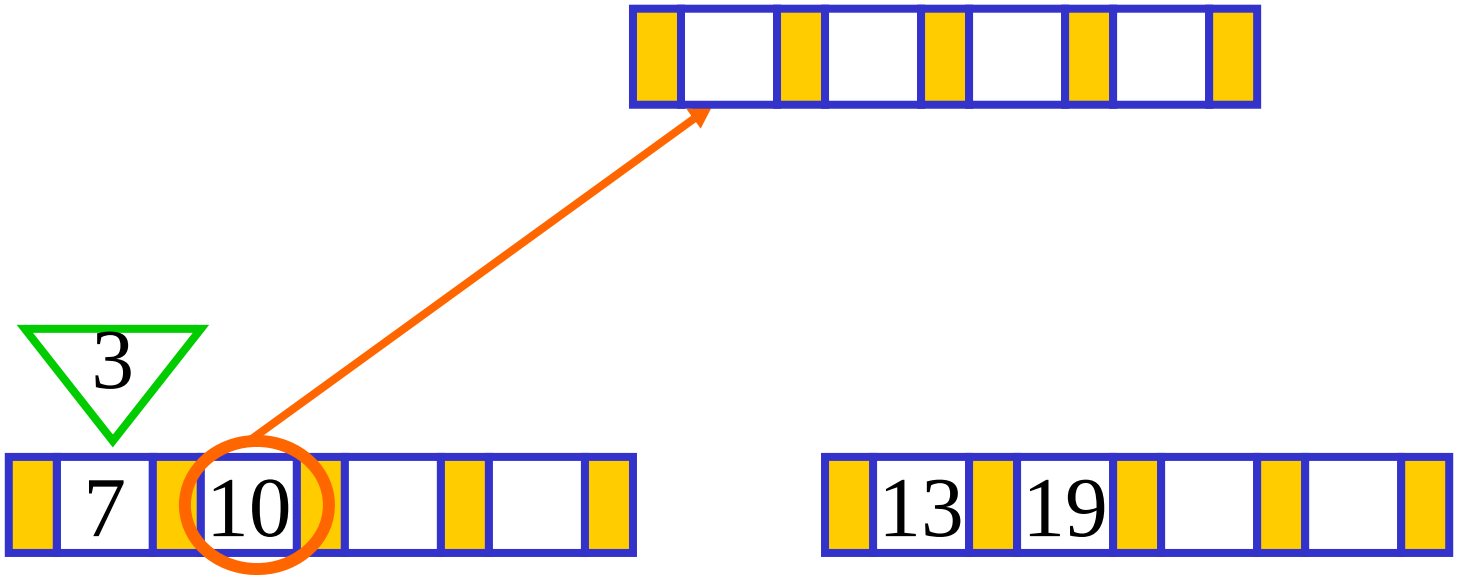
Sukzessiver Aufbau eines B-Baums vom Grad k=2



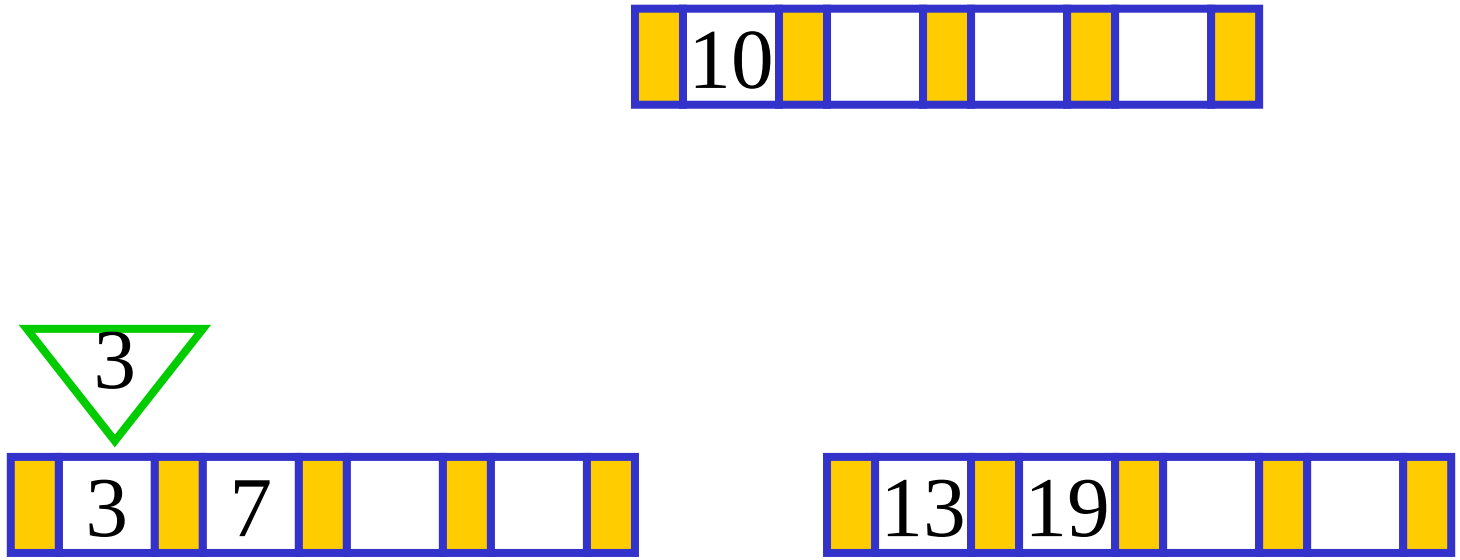
Sukzessiver Aufbau eines B-Baums vom Grad k=2



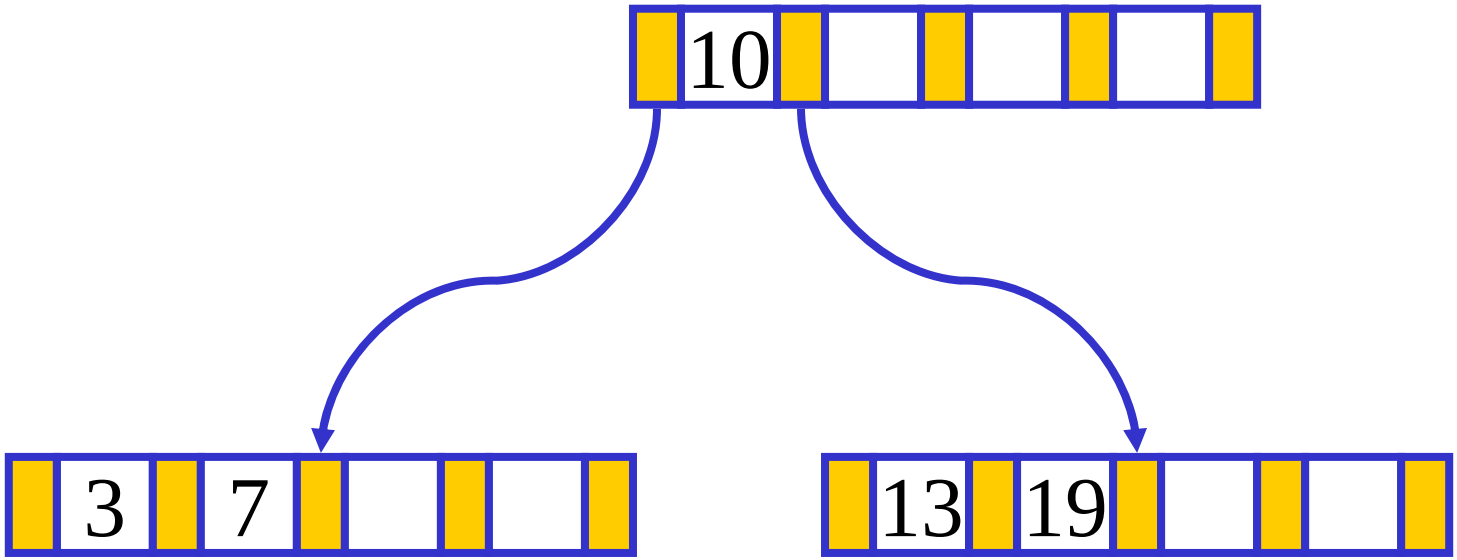
Sukzessiver Aufbau eines B-Baums vom Grad k=2



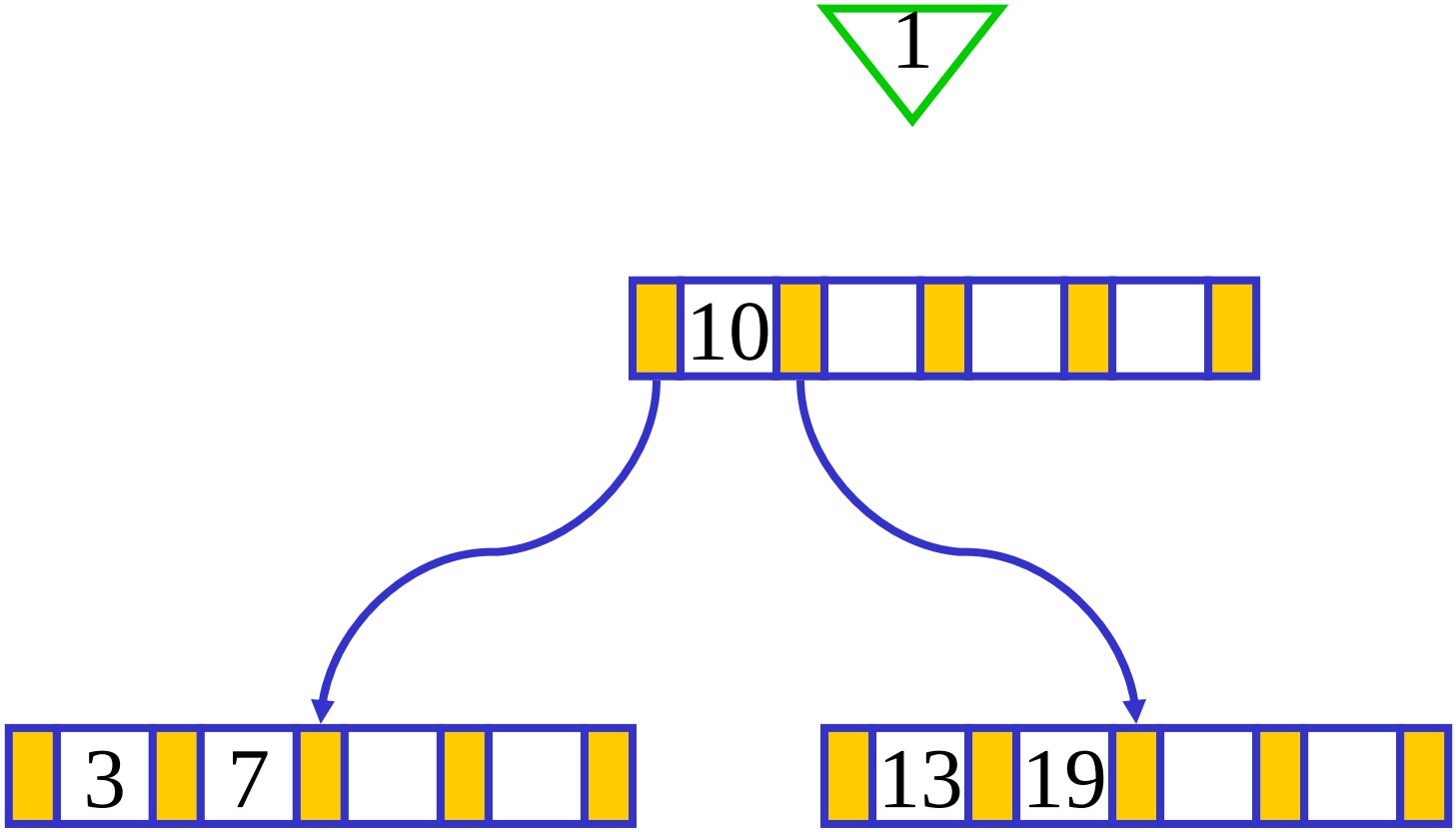
Sukzessiver Aufbau eines B-Baums vom Grad k=2



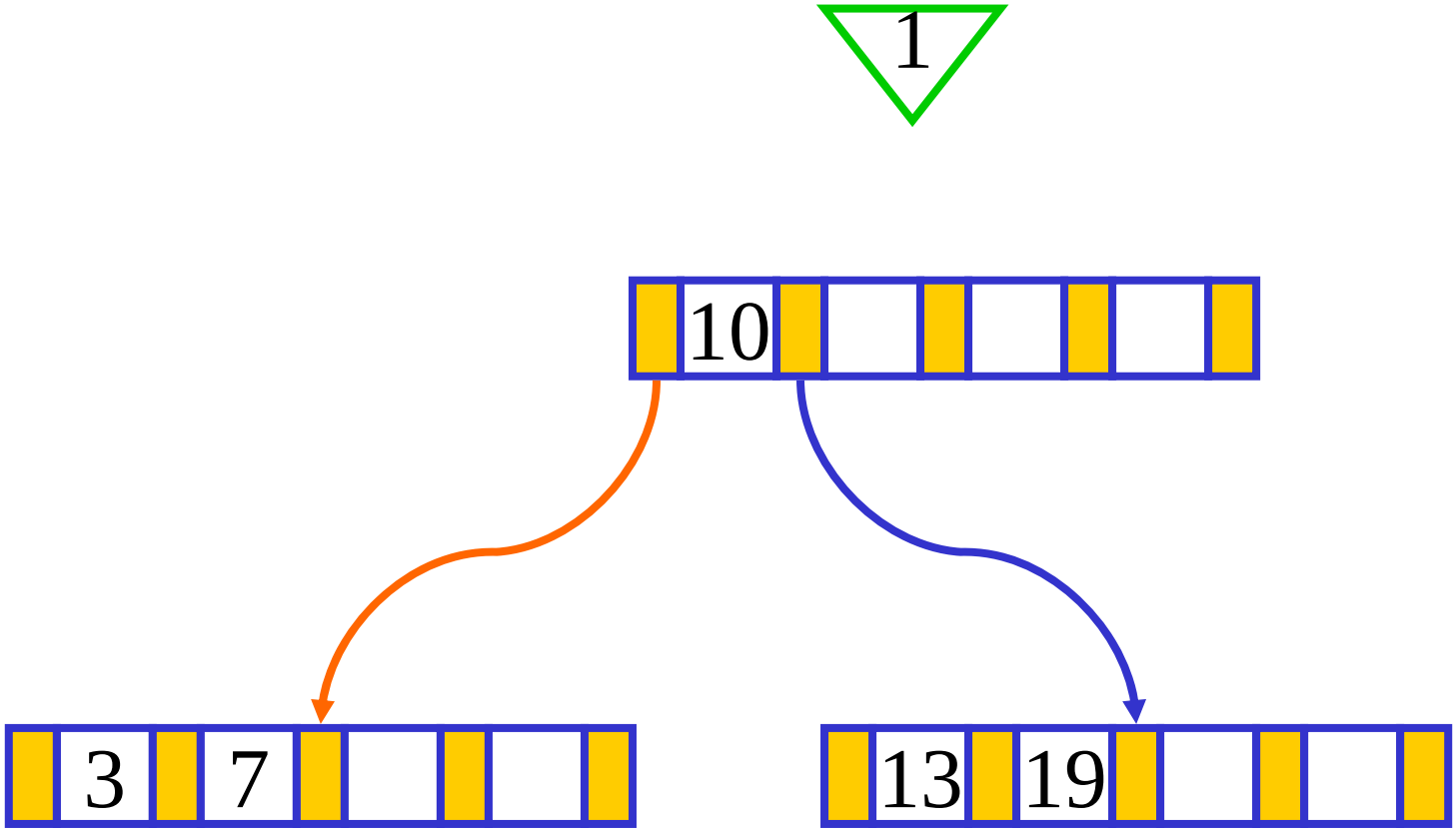
Sukzessiver Aufbau eines B-Baums vom Grad k=2



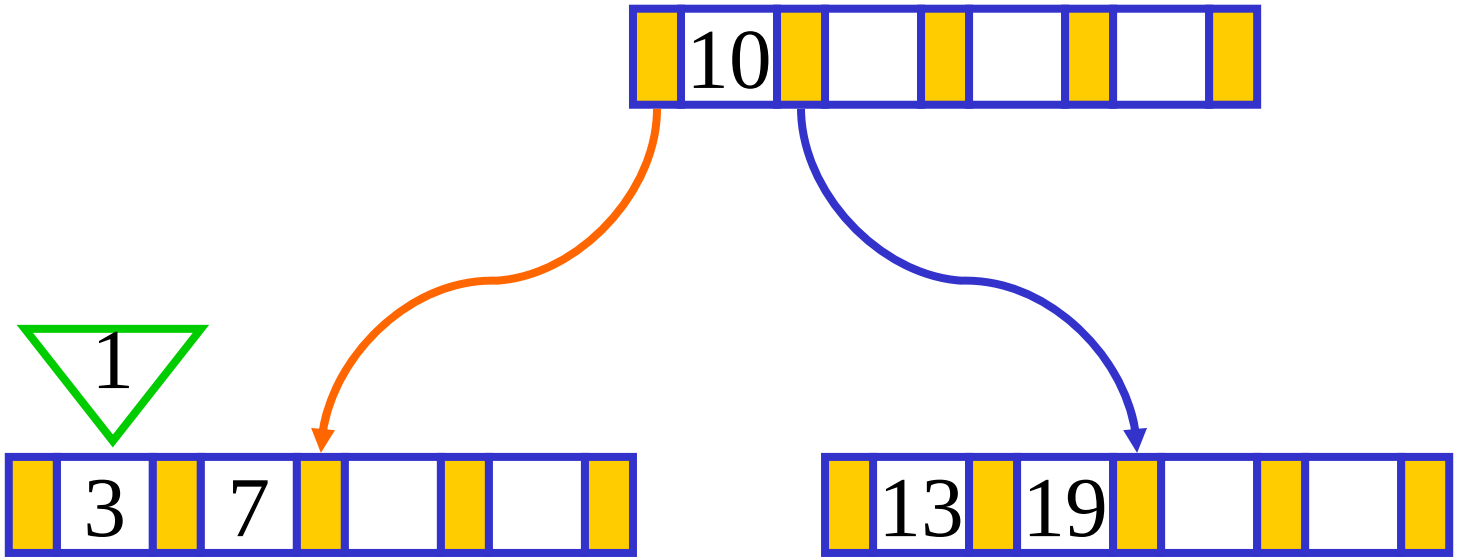
Sukzessiver Aufbau eines B-Baums vom Grad k=2



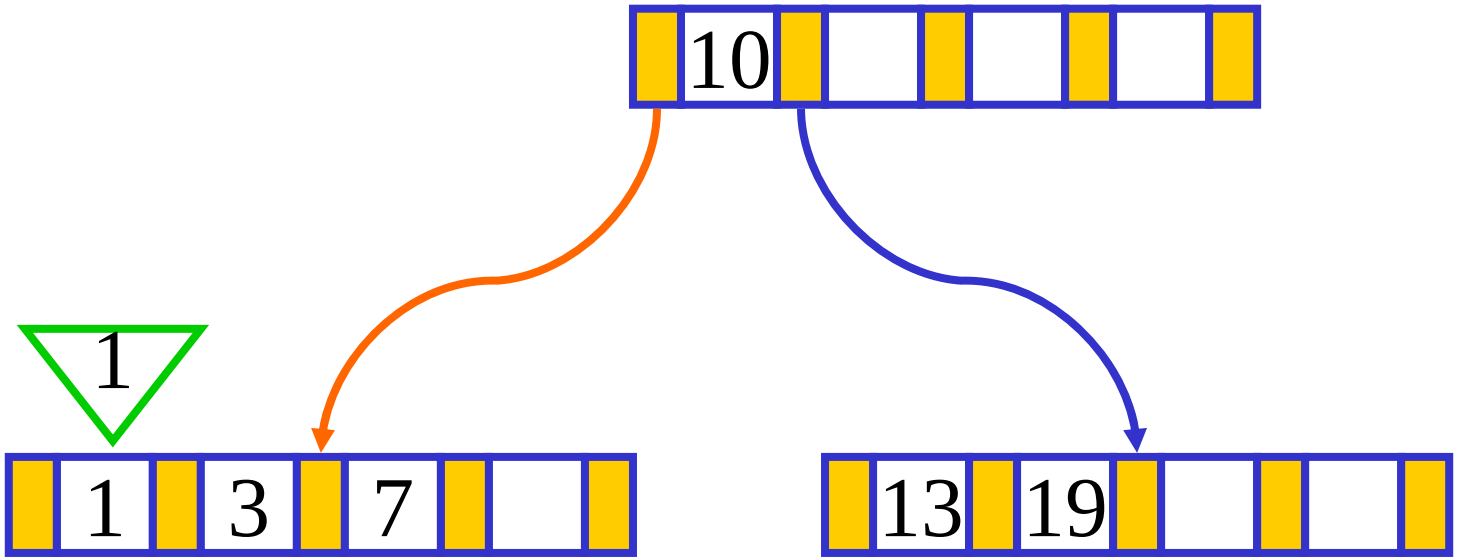
Sukzessiver Aufbau eines B-Baums vom Grad k=2



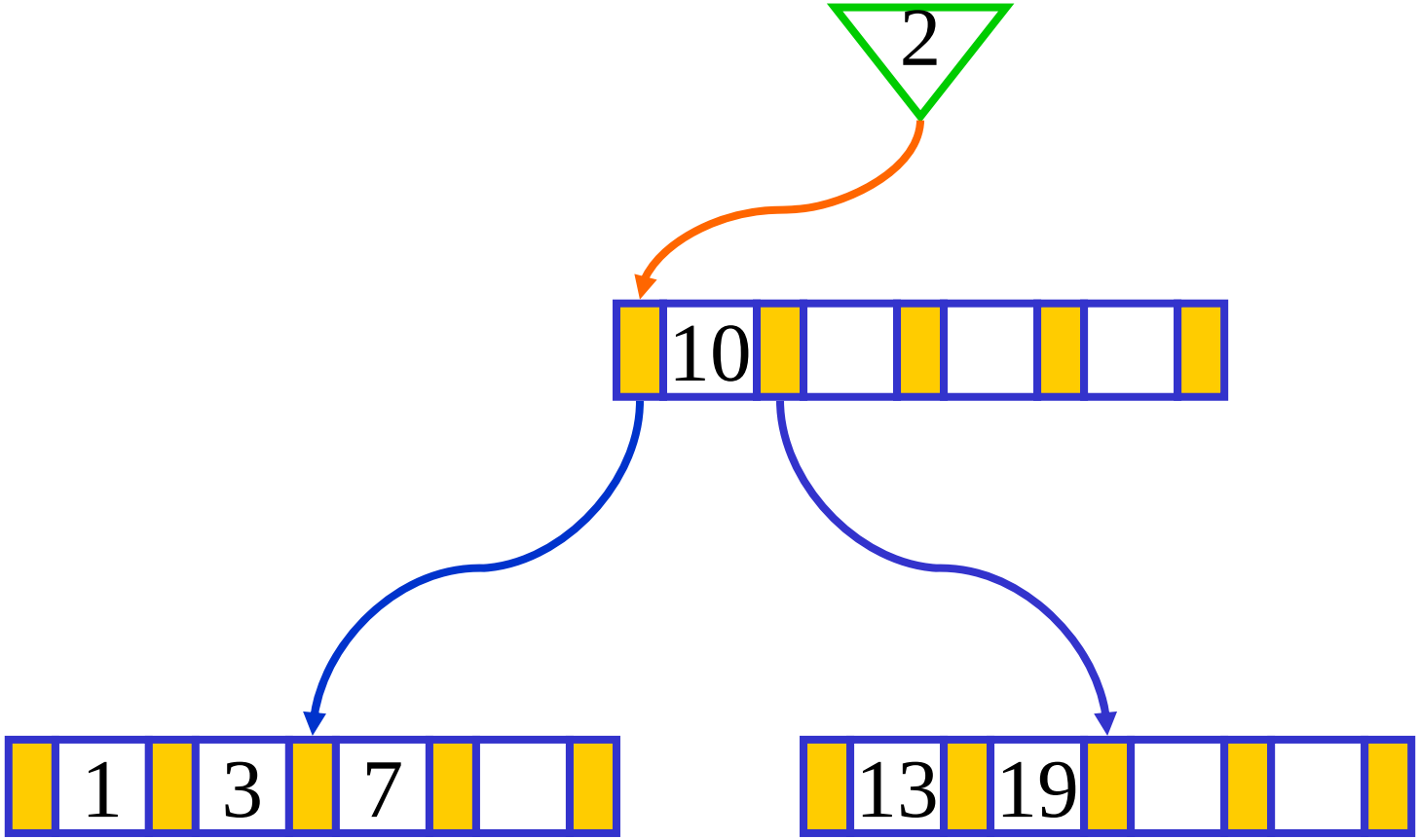
Sukzessiver Aufbau eines B-Baums vom Grad k=2



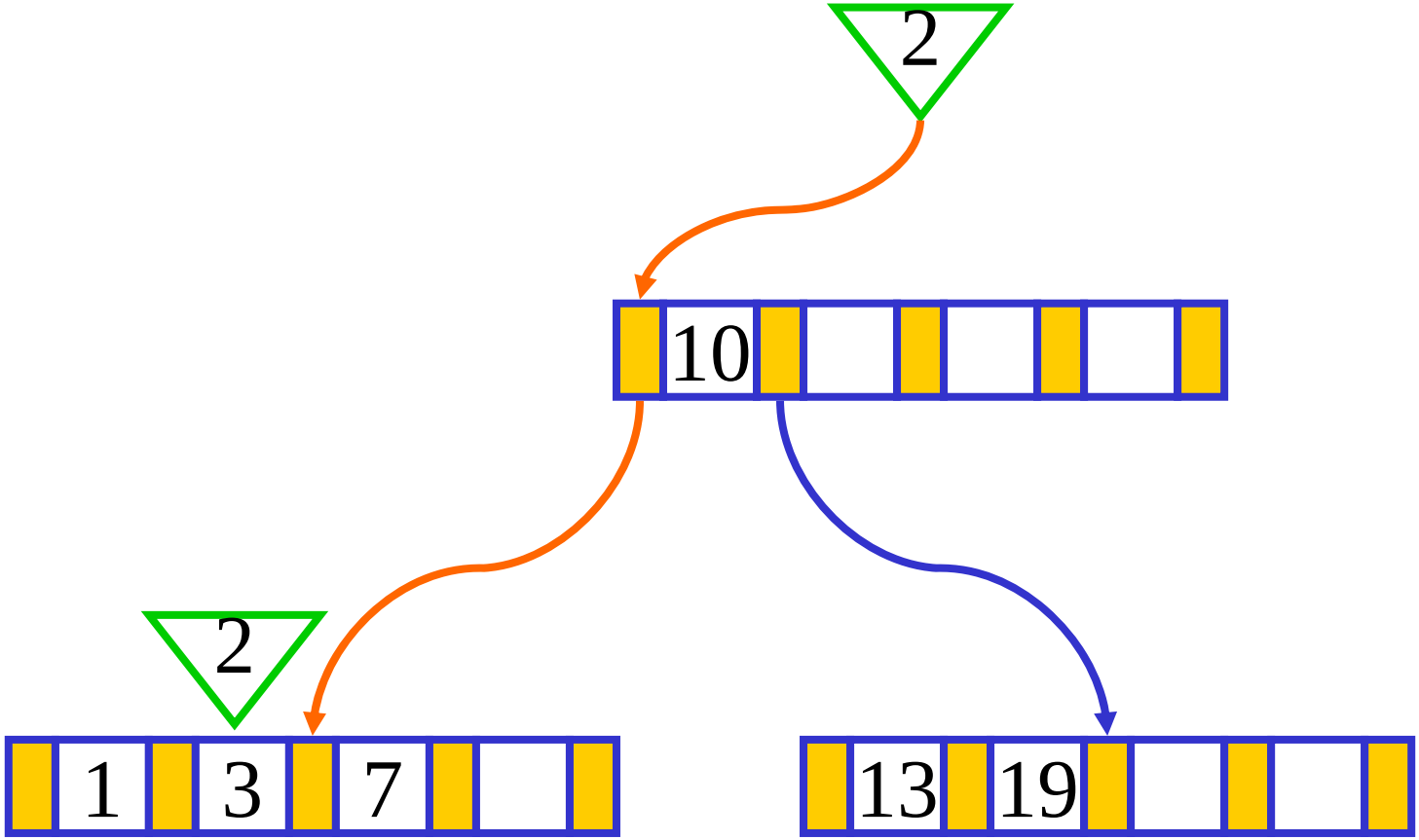
Sukzessiver Aufbau eines B-Baums vom Grad k=2



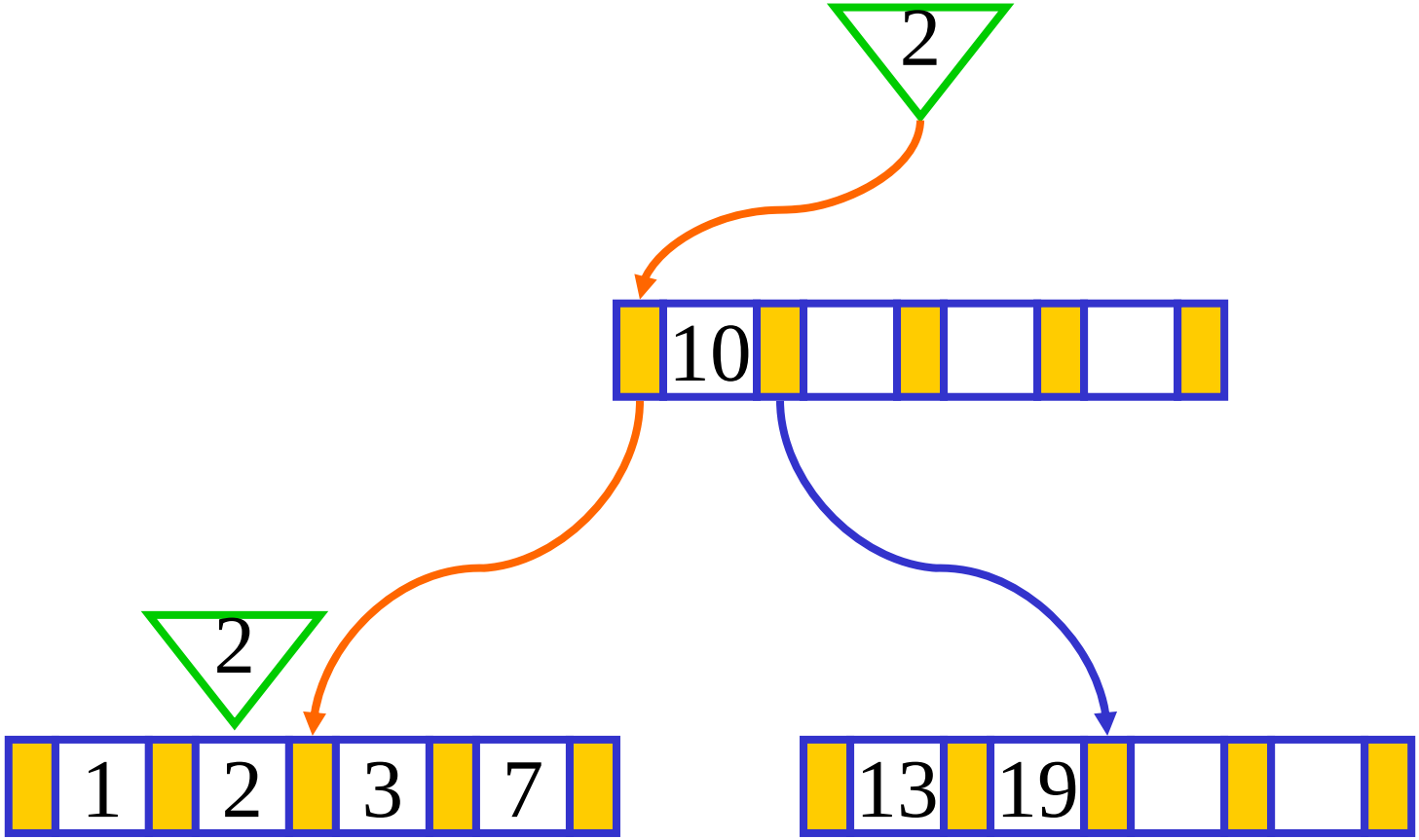
Sukzessiver Aufbau eines B-Baums vom Grad k=2



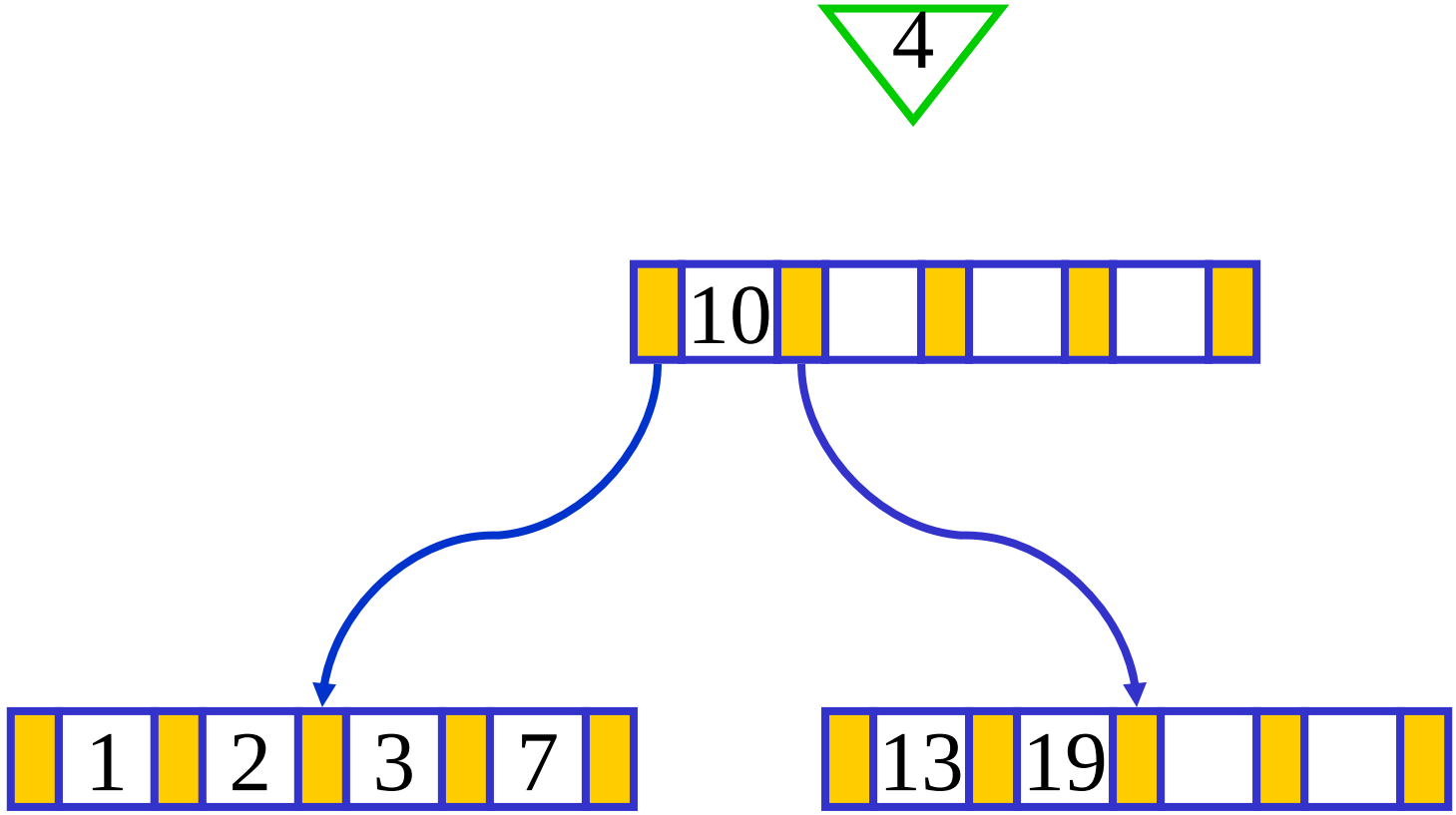
Sukzessiver Aufbau eines B-Baums vom Grad k=2



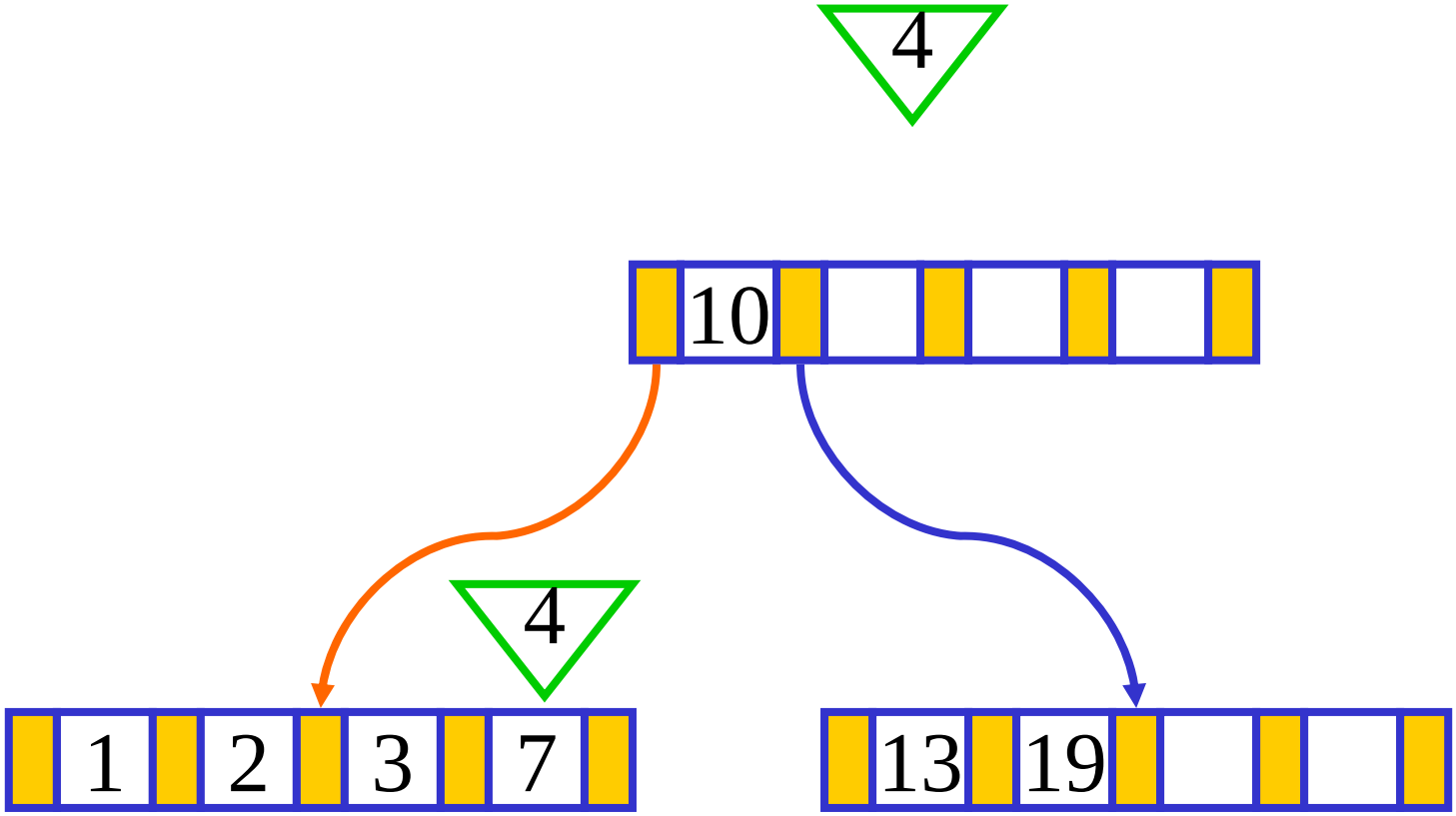
Sukzessiver Aufbau eines B-Baums vom Grad k=2



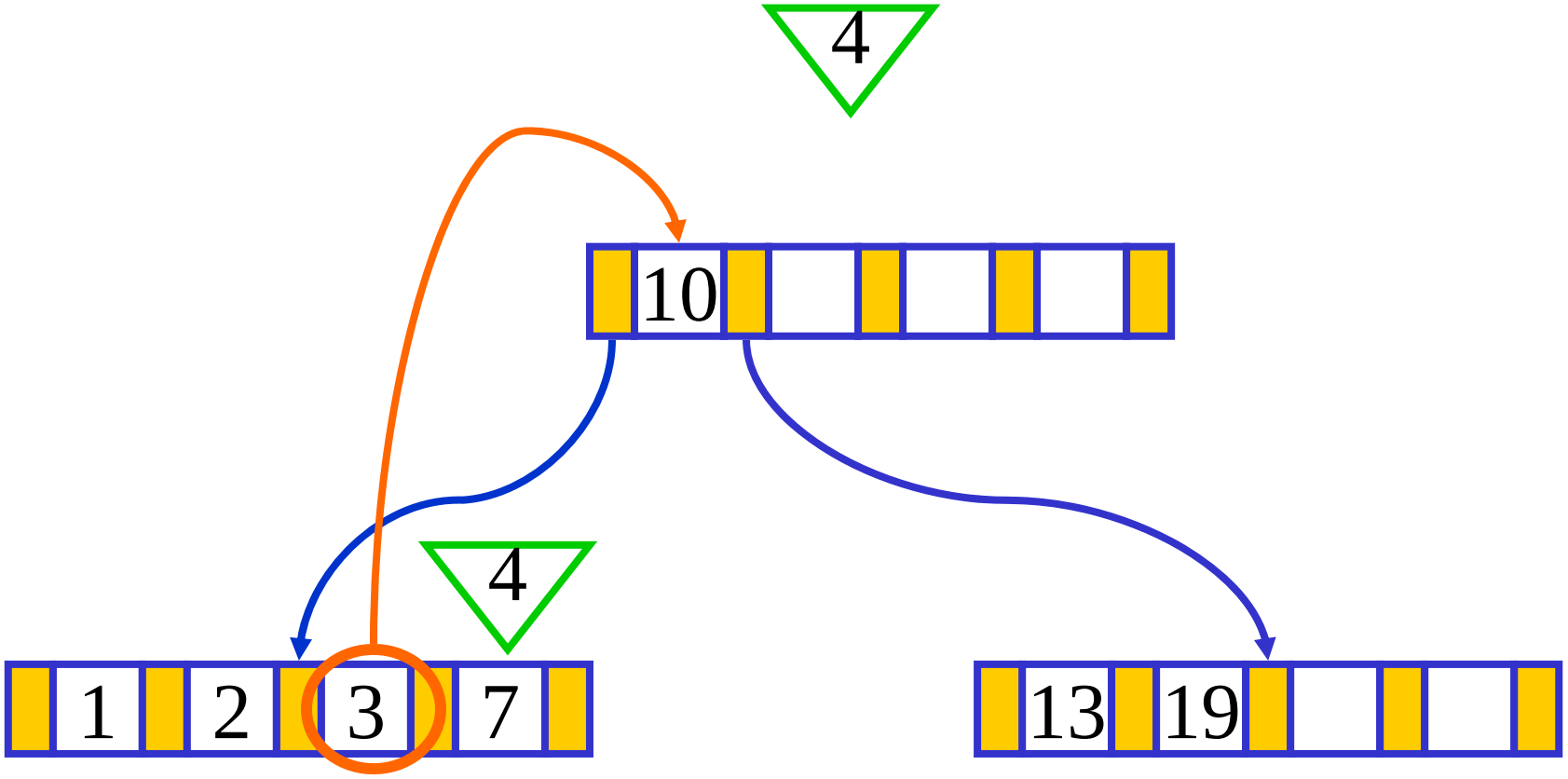
Sukzessiver Aufbau eines B-Baums vom Grad k=2



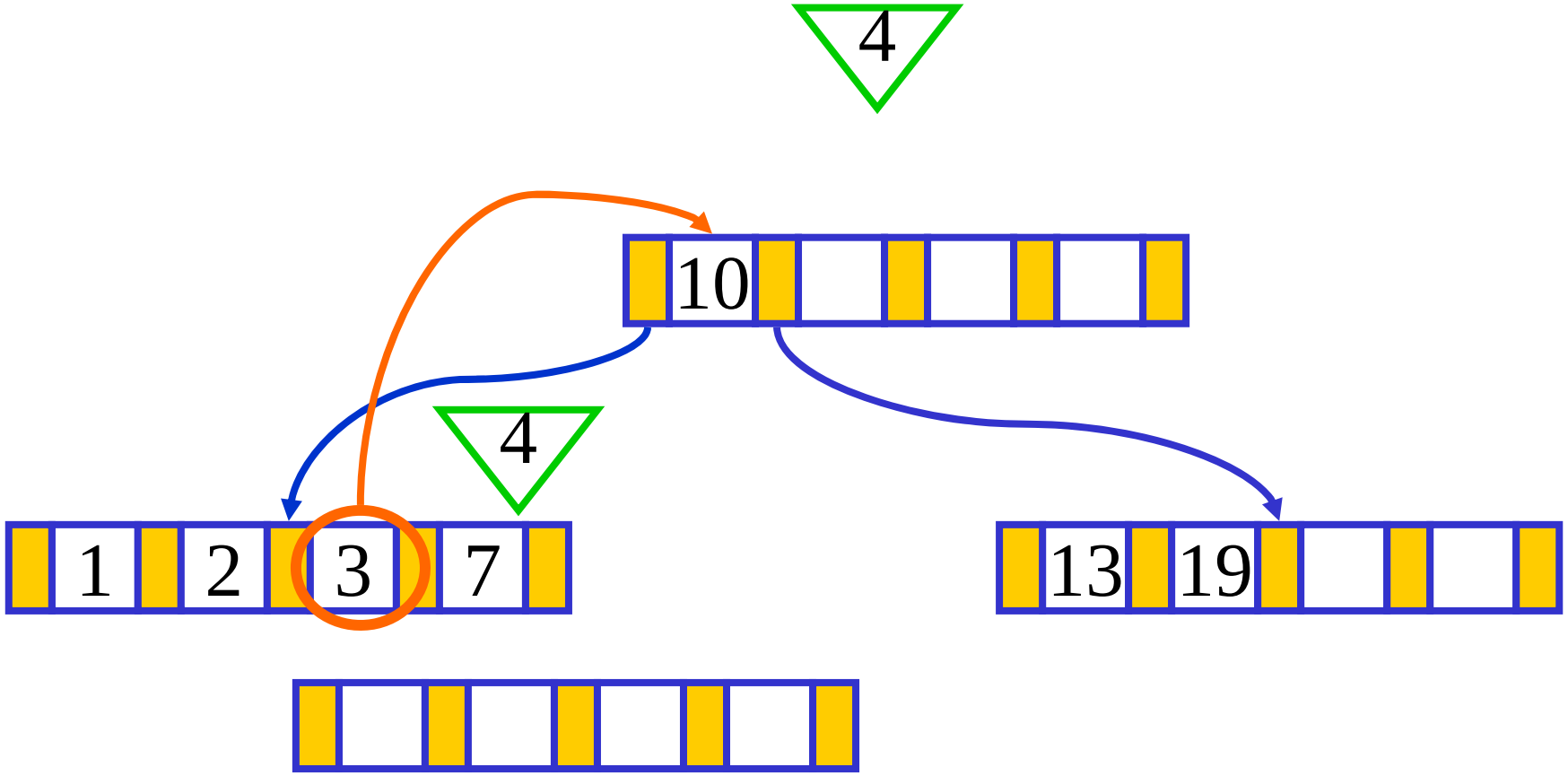
Sukzessiver Aufbau eines B-Baums vom Grad k=2



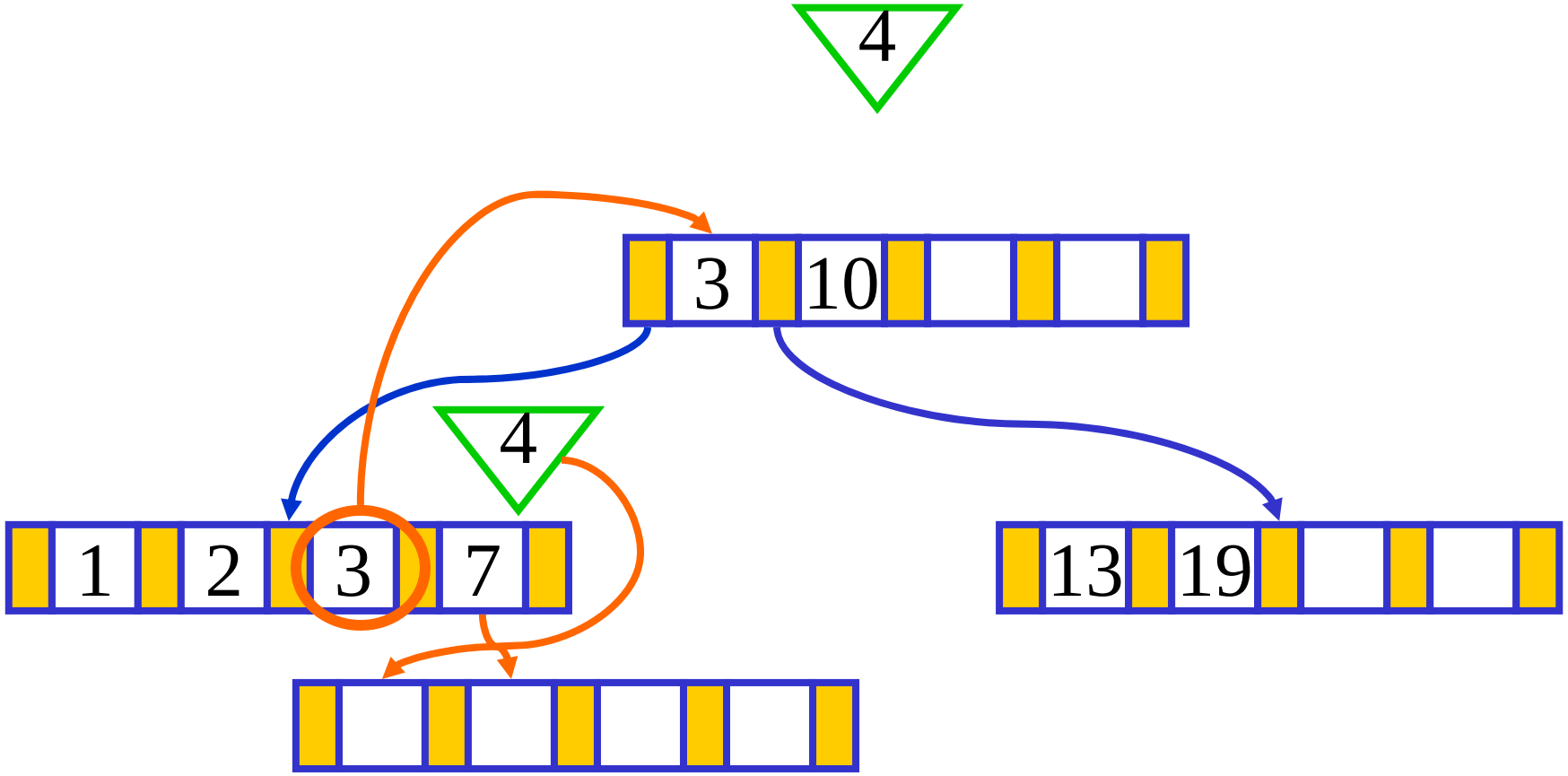
Sukzessiver Aufbau eines B-Baums vom Grad k=2



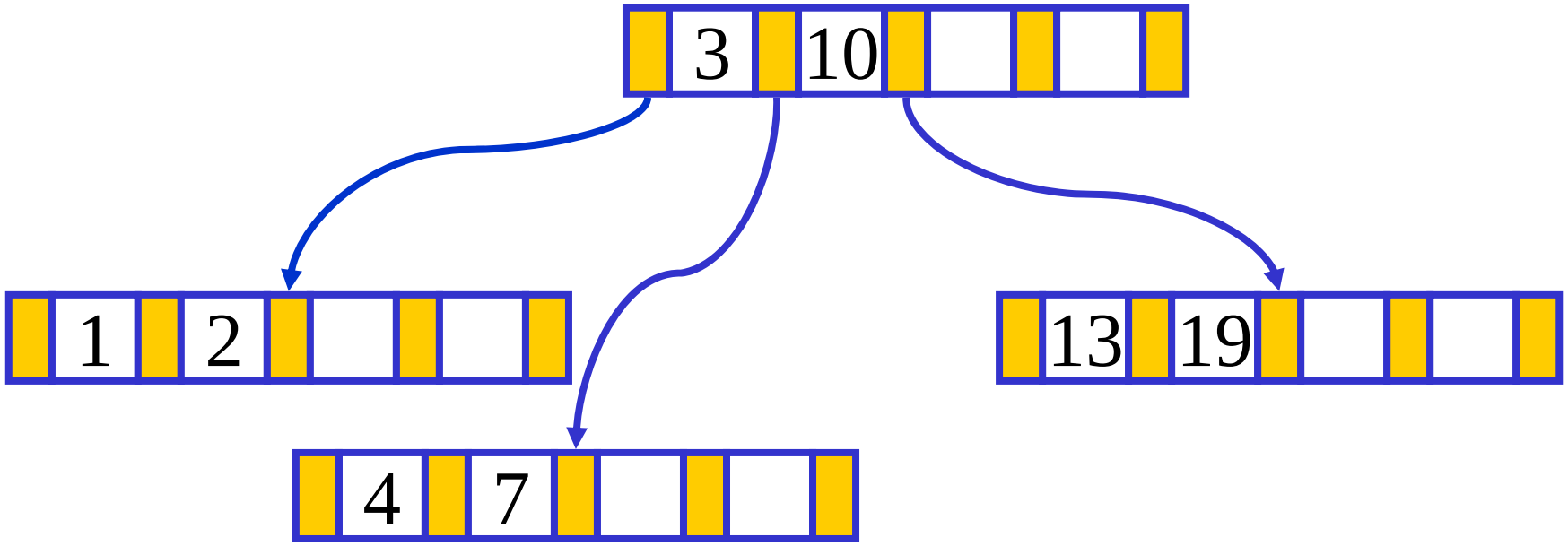
Sukzessiver Aufbau eines B-Baums vom Grad k=2



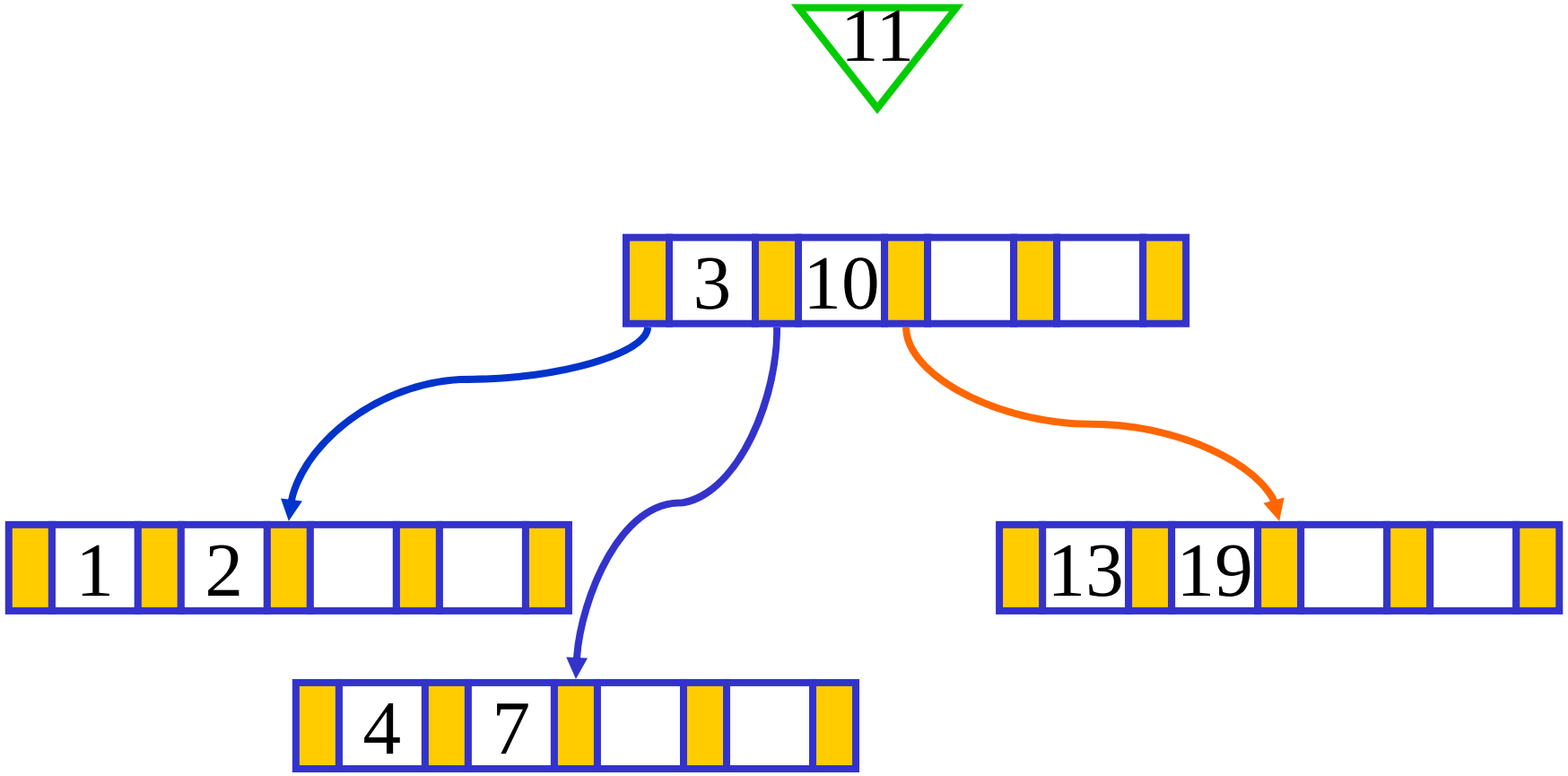
Sukzessiver Aufbau eines B-Baums vom Grad k=2



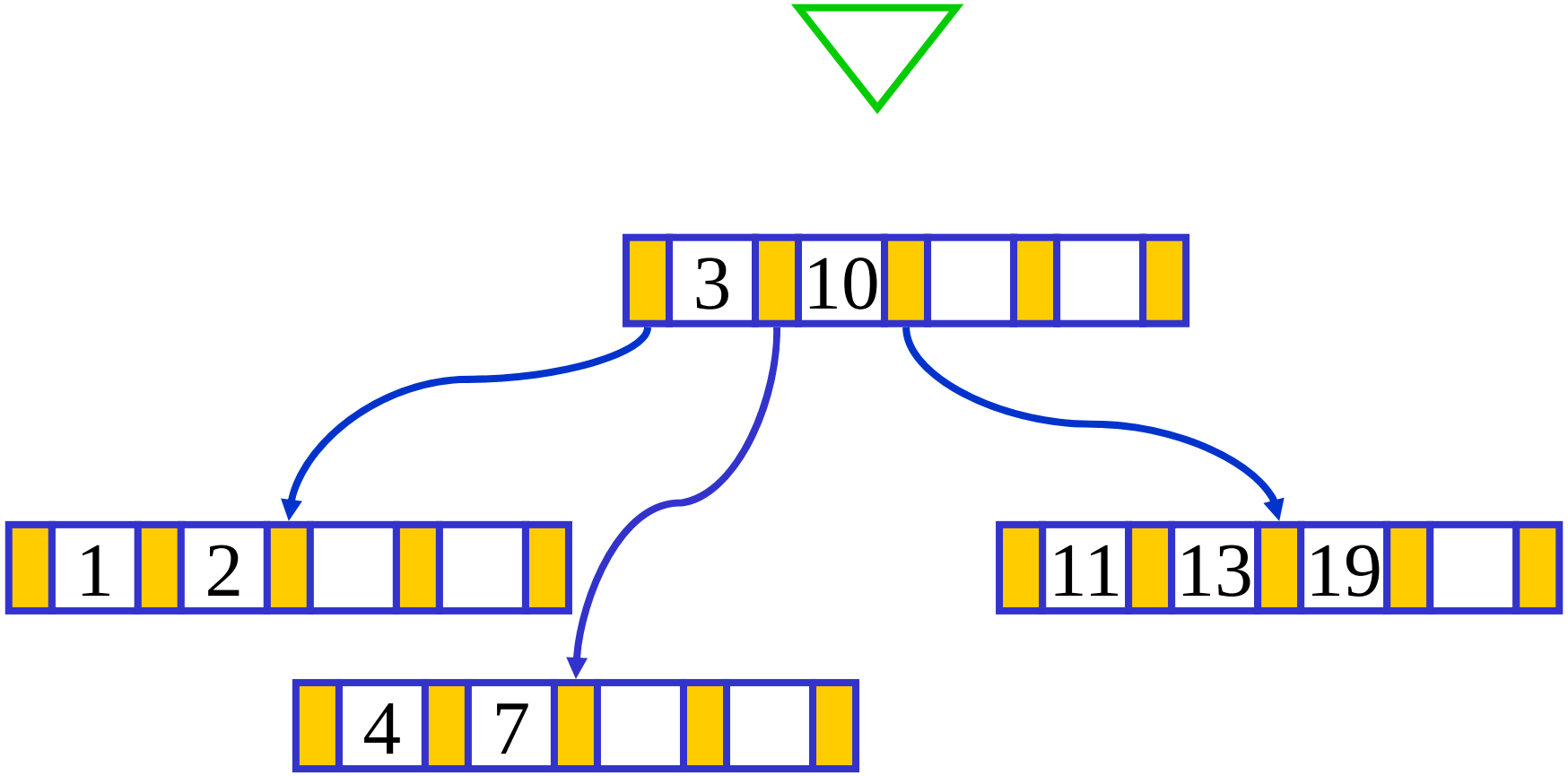
Sukzessiver Aufbau eines B-Baums vom Grad k=2



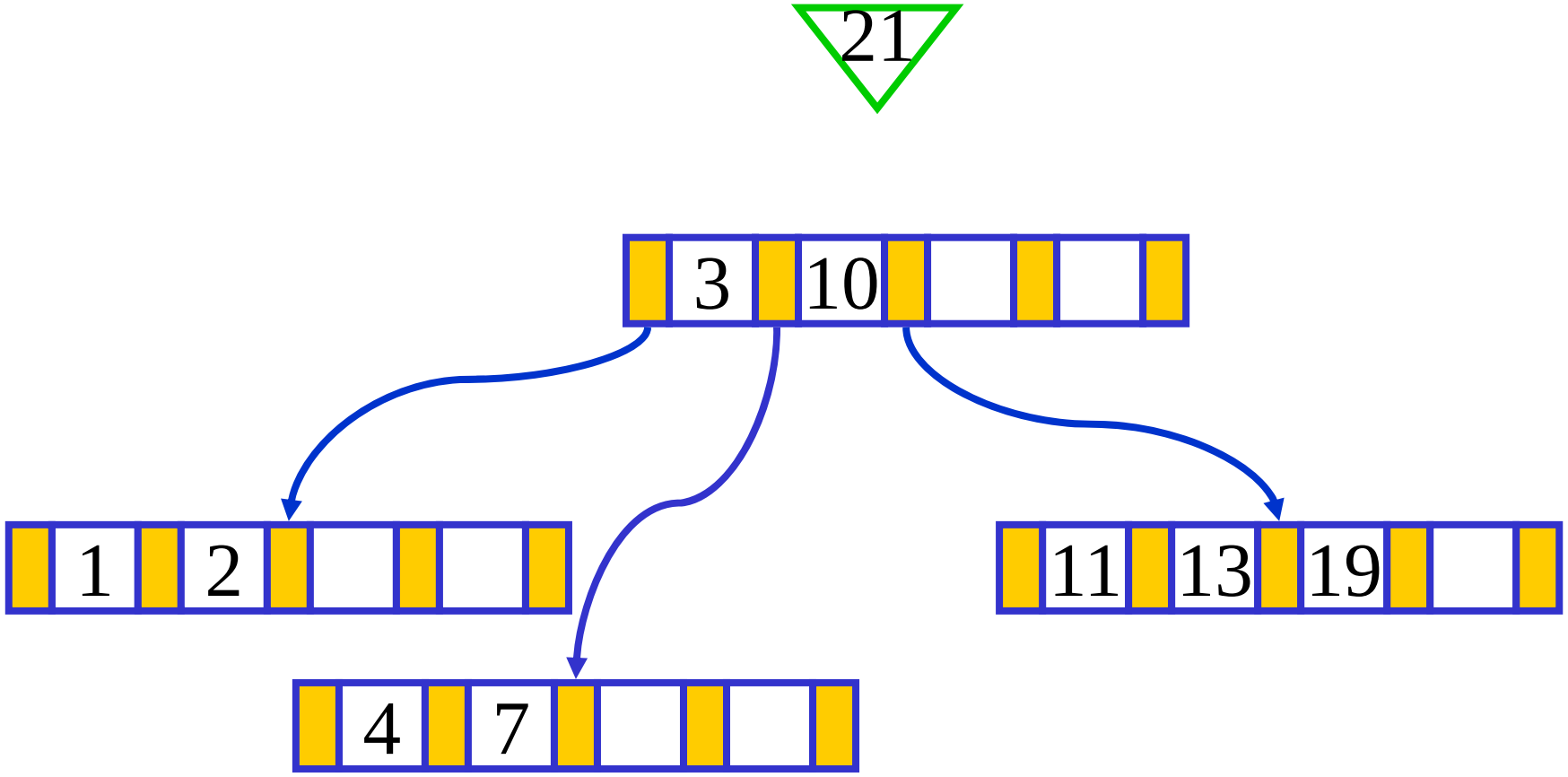
Sukzessiver Aufbau eines B-Baums vom Grad k=2



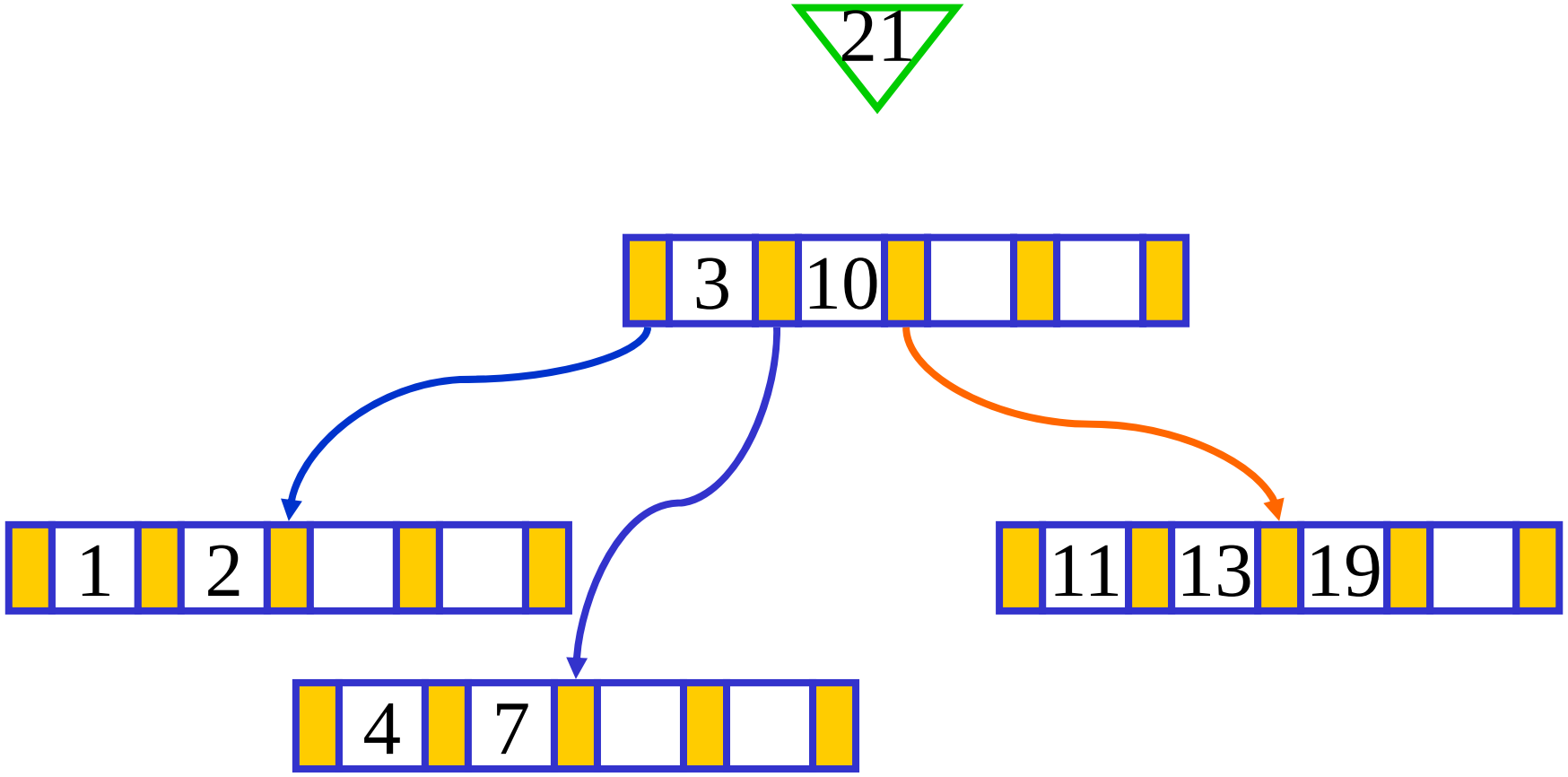
Sukzessiver Aufbau eines B-Baums vom Grad k=2



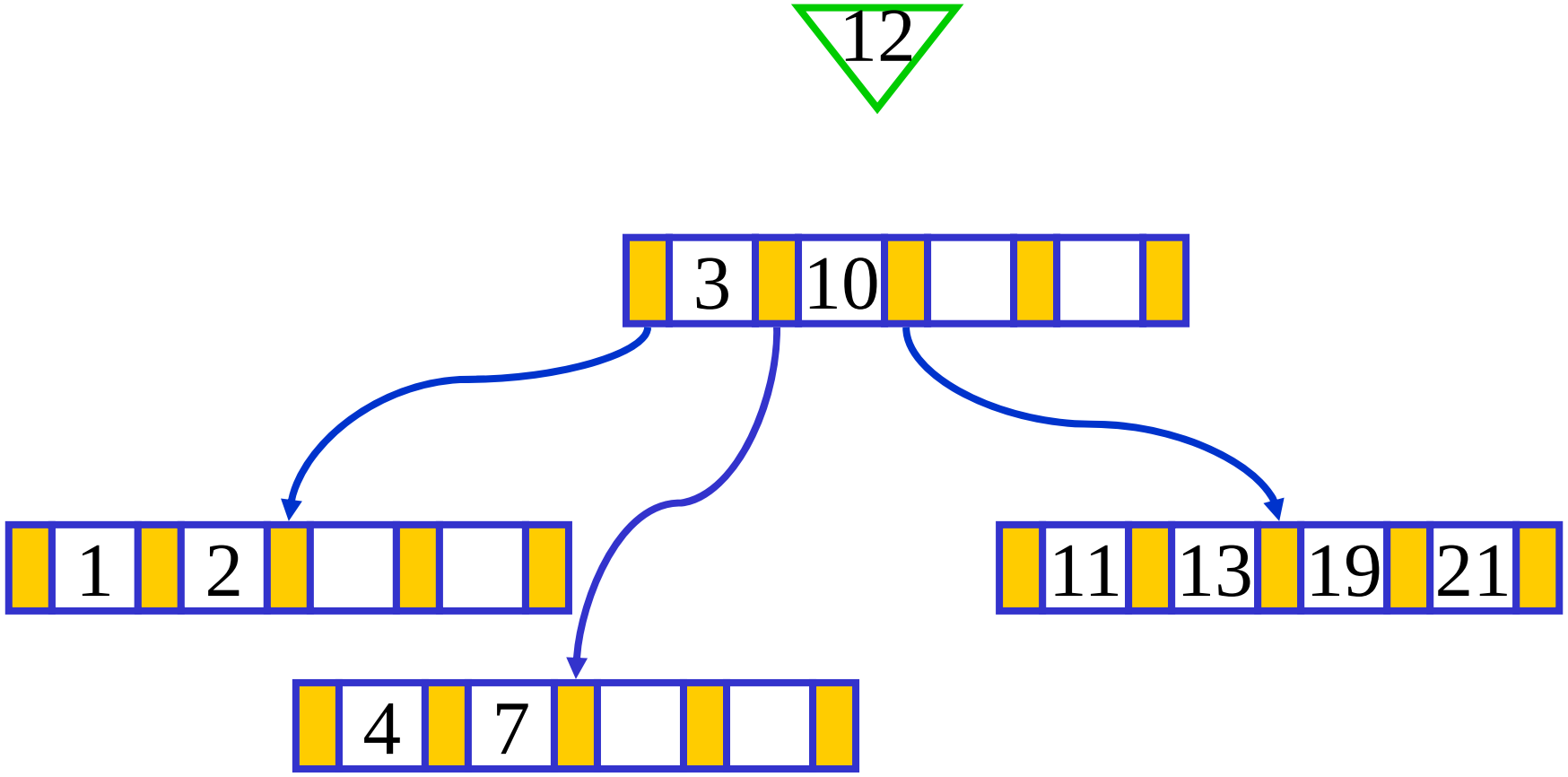
Sukzessiver Aufbau eines B-Baums vom Grad k=2



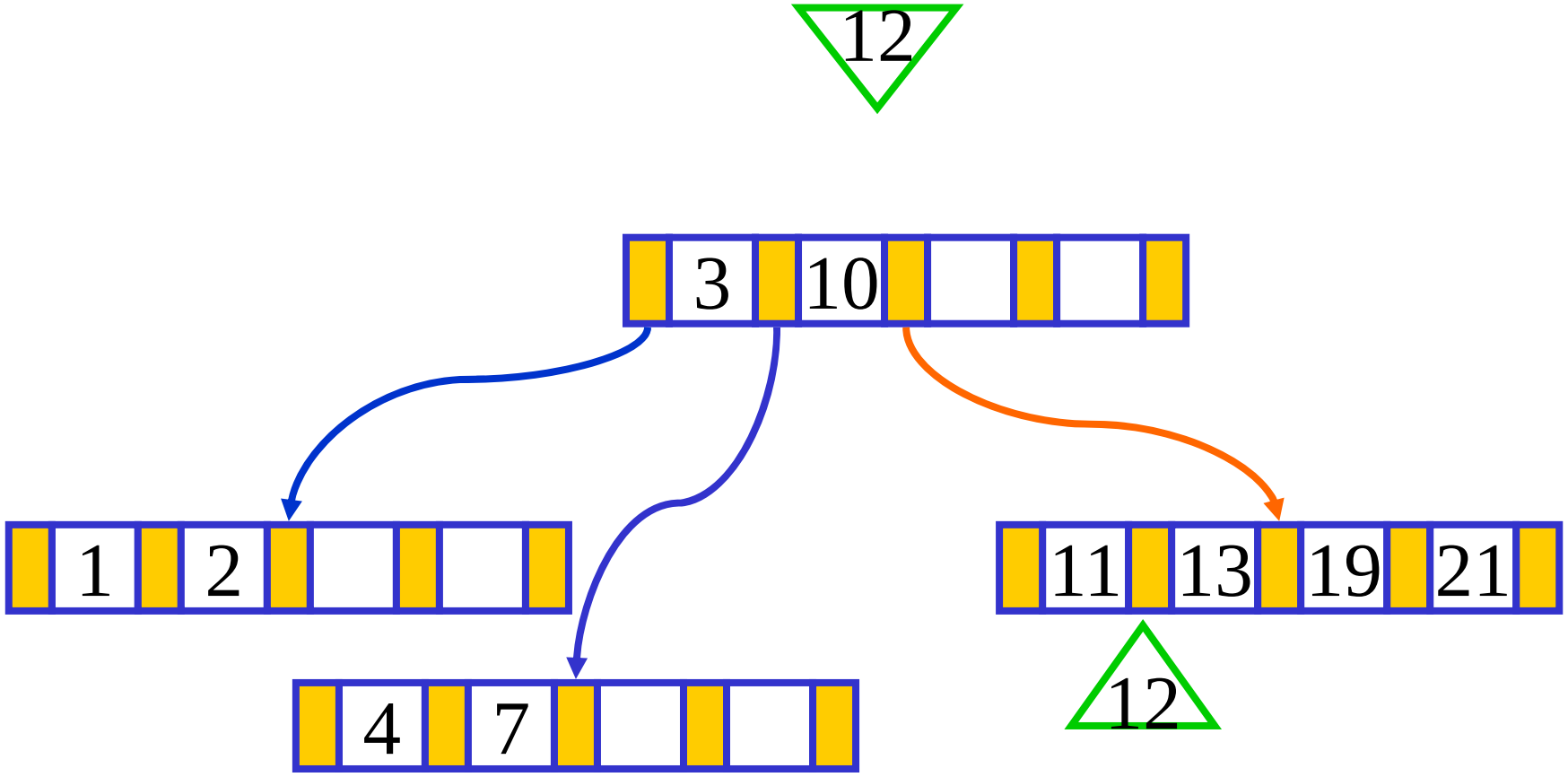
Sukzessiver Aufbau eines B-Baums vom Grad k=2



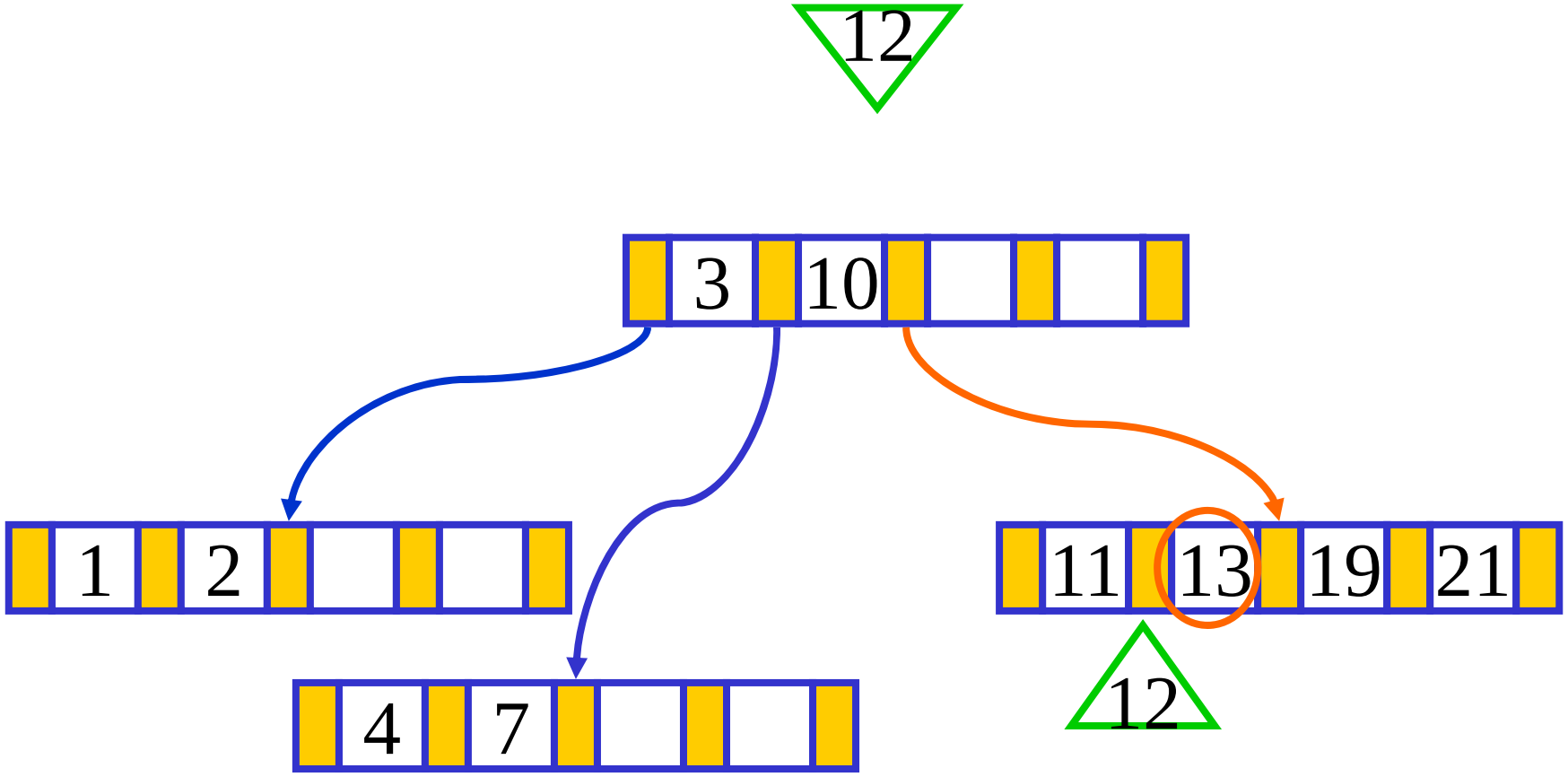
Sukzessiver Aufbau eines B-Baums vom Grad k=2



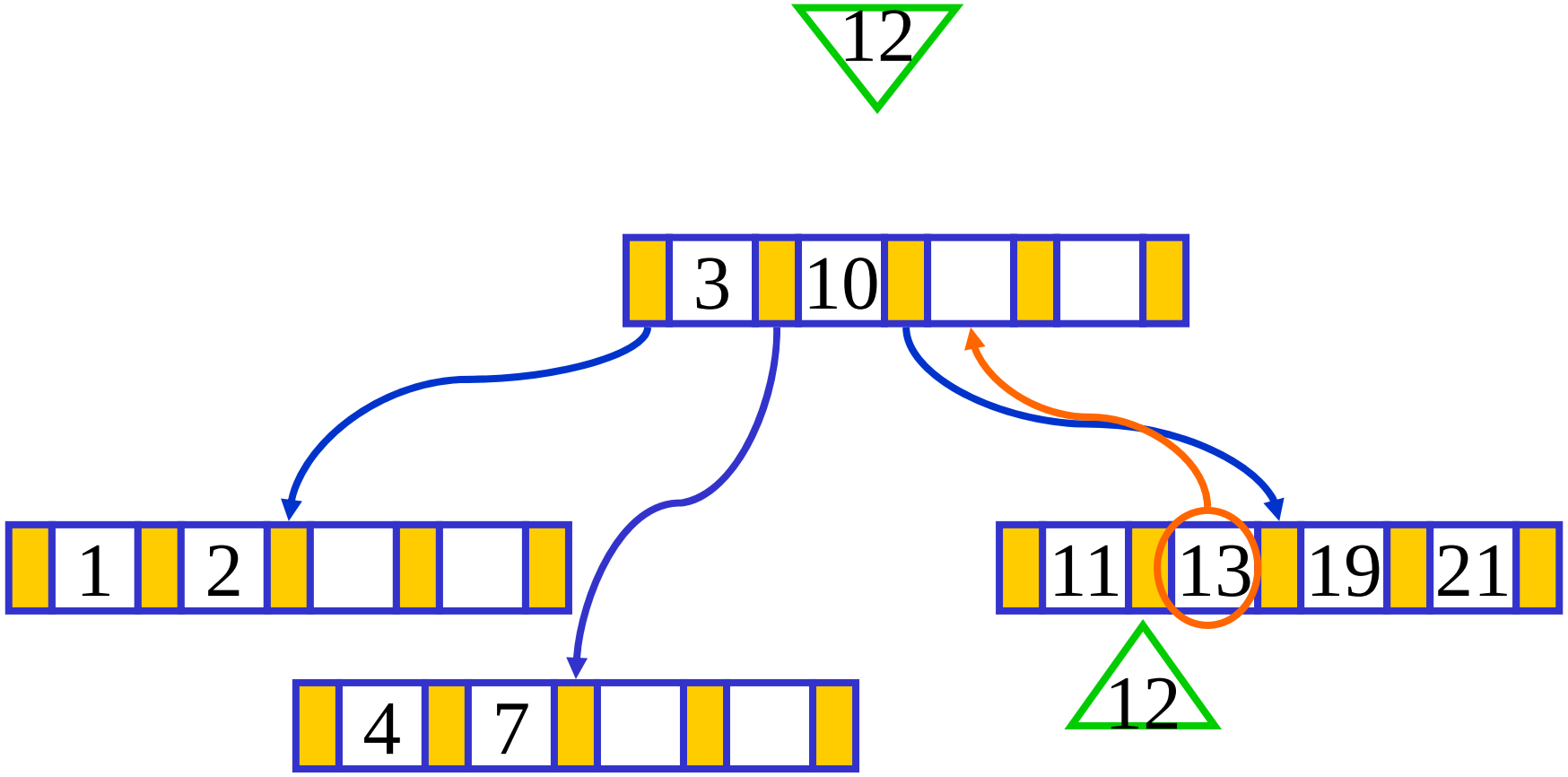
Sukzessiver Aufbau eines B-Baums vom Grad k=2



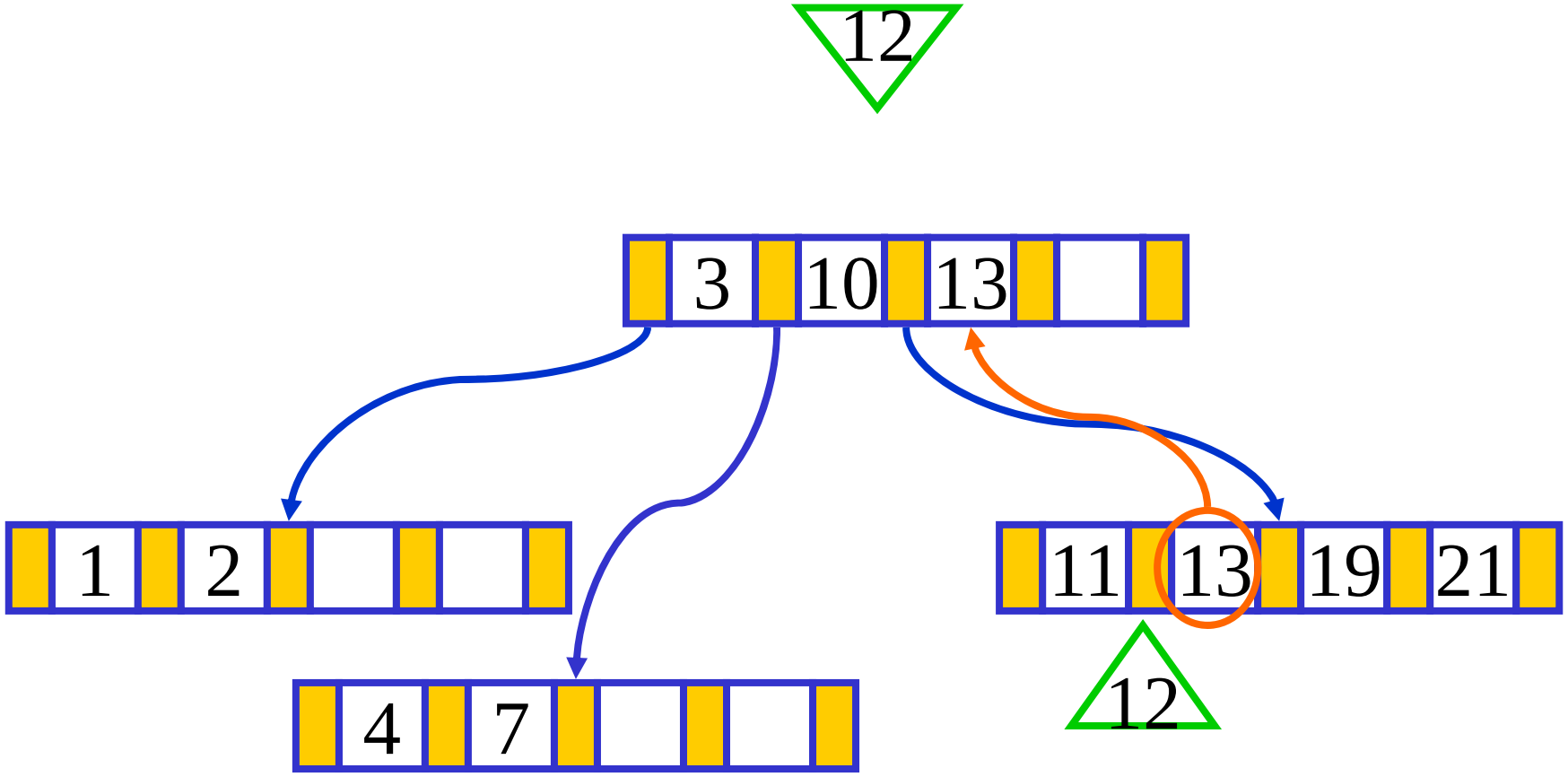
Sukzessiver Aufbau eines B-Baums vom Grad k=2



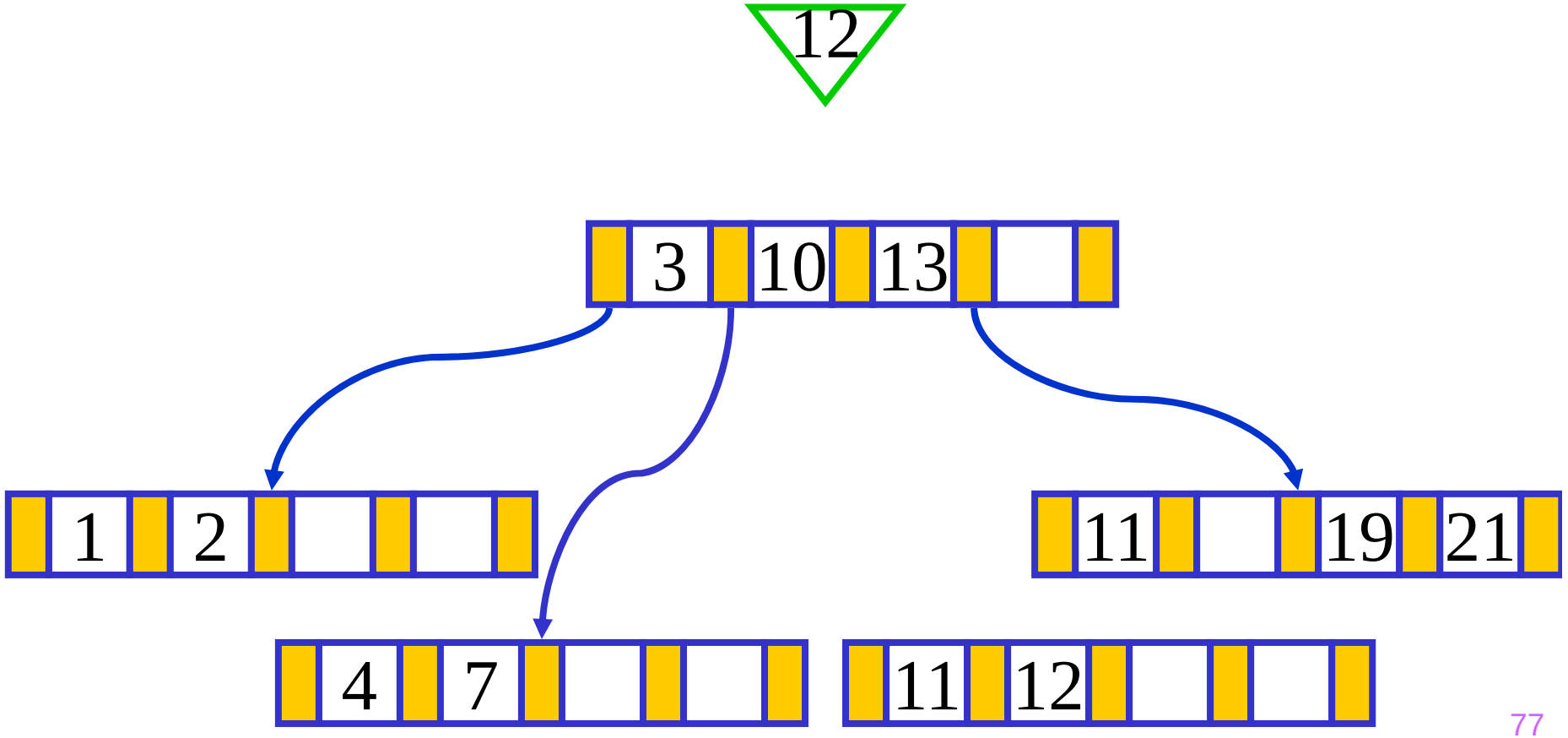
Sukzessiver Aufbau eines B-Baums vom Grad k=2



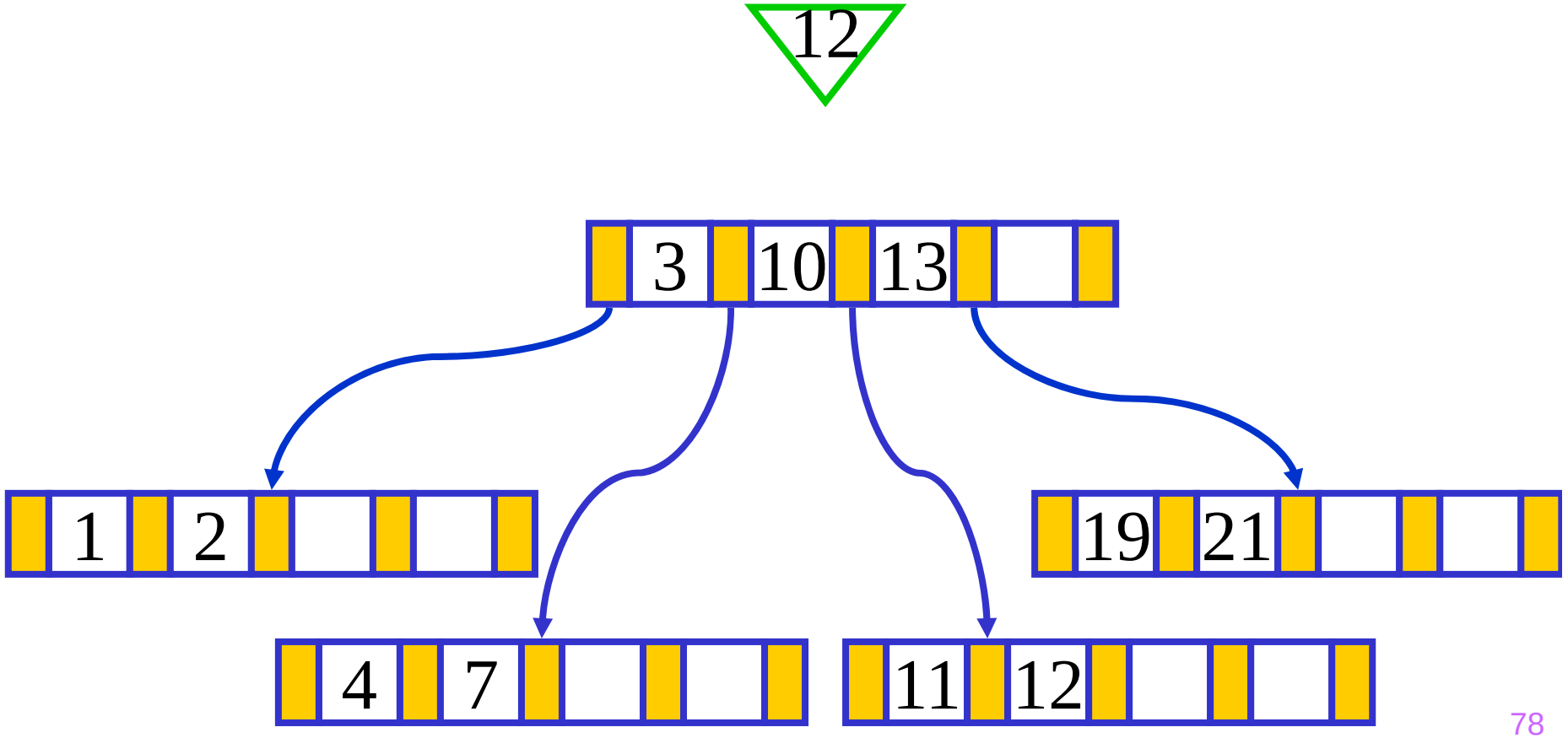
Sukzessiver Aufbau eines B-Baums vom Grad k=2



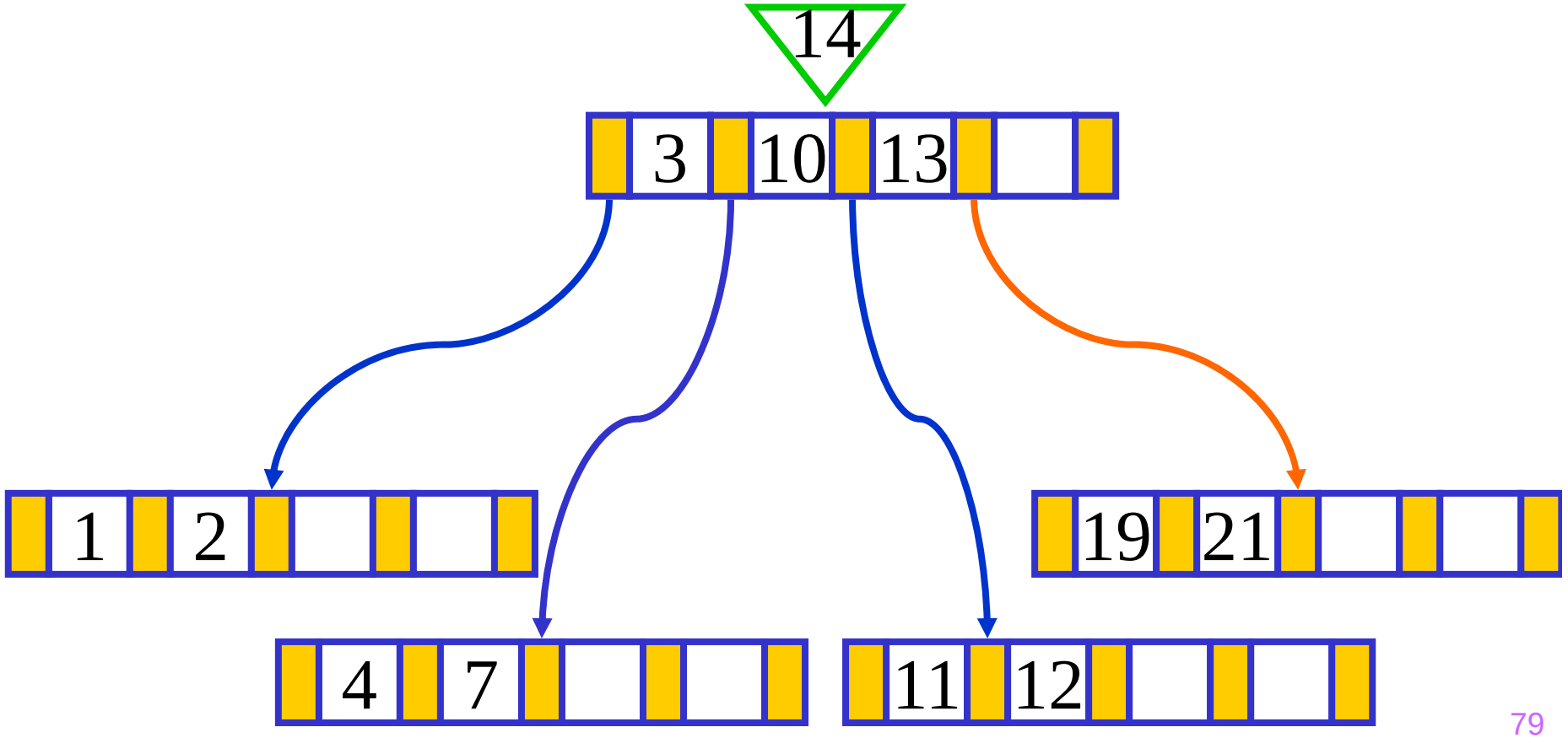
Sukzessiver Aufbau eines B-Baums vom Grad k=2



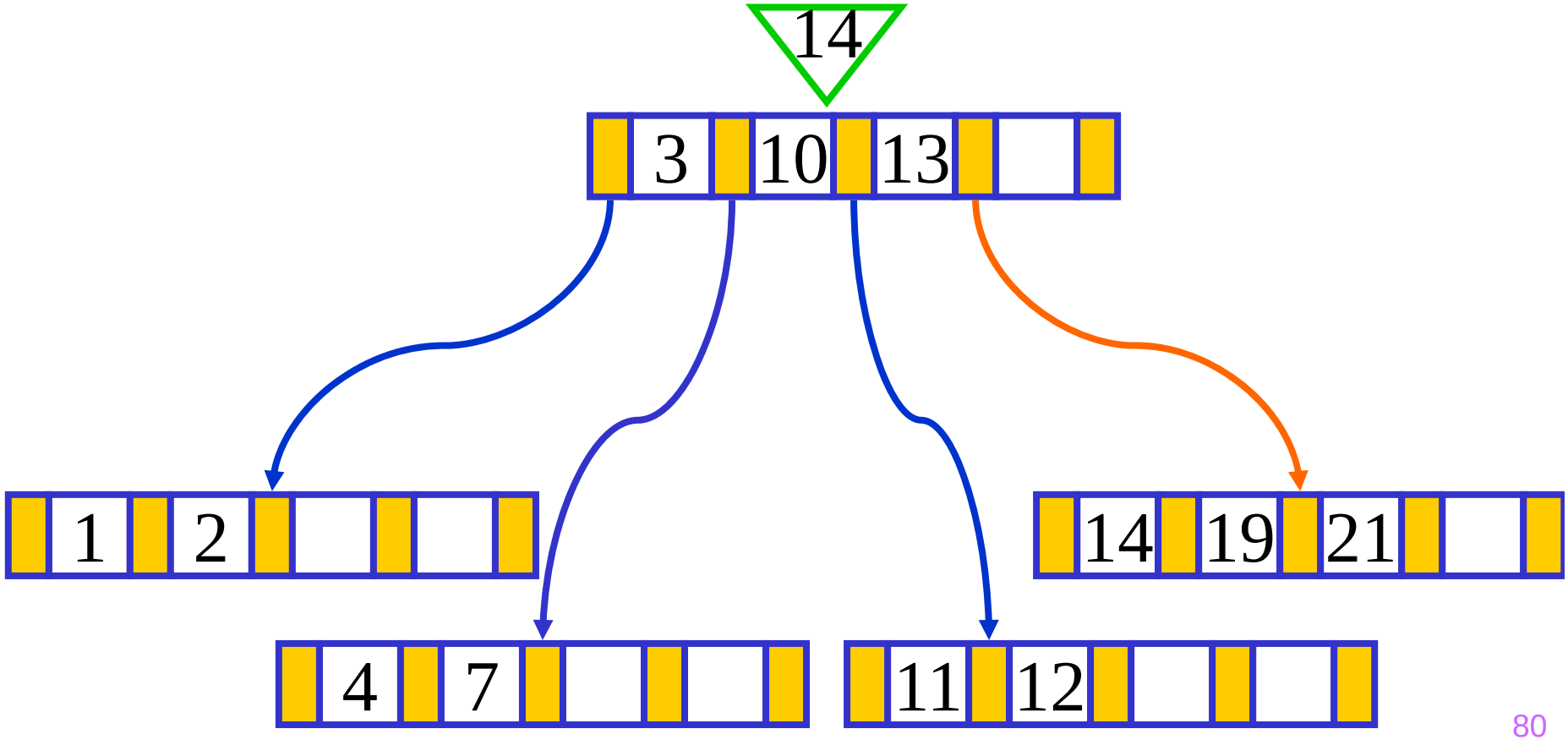
Sukzessiver Aufbau eines B-Baums vom Grad k=2



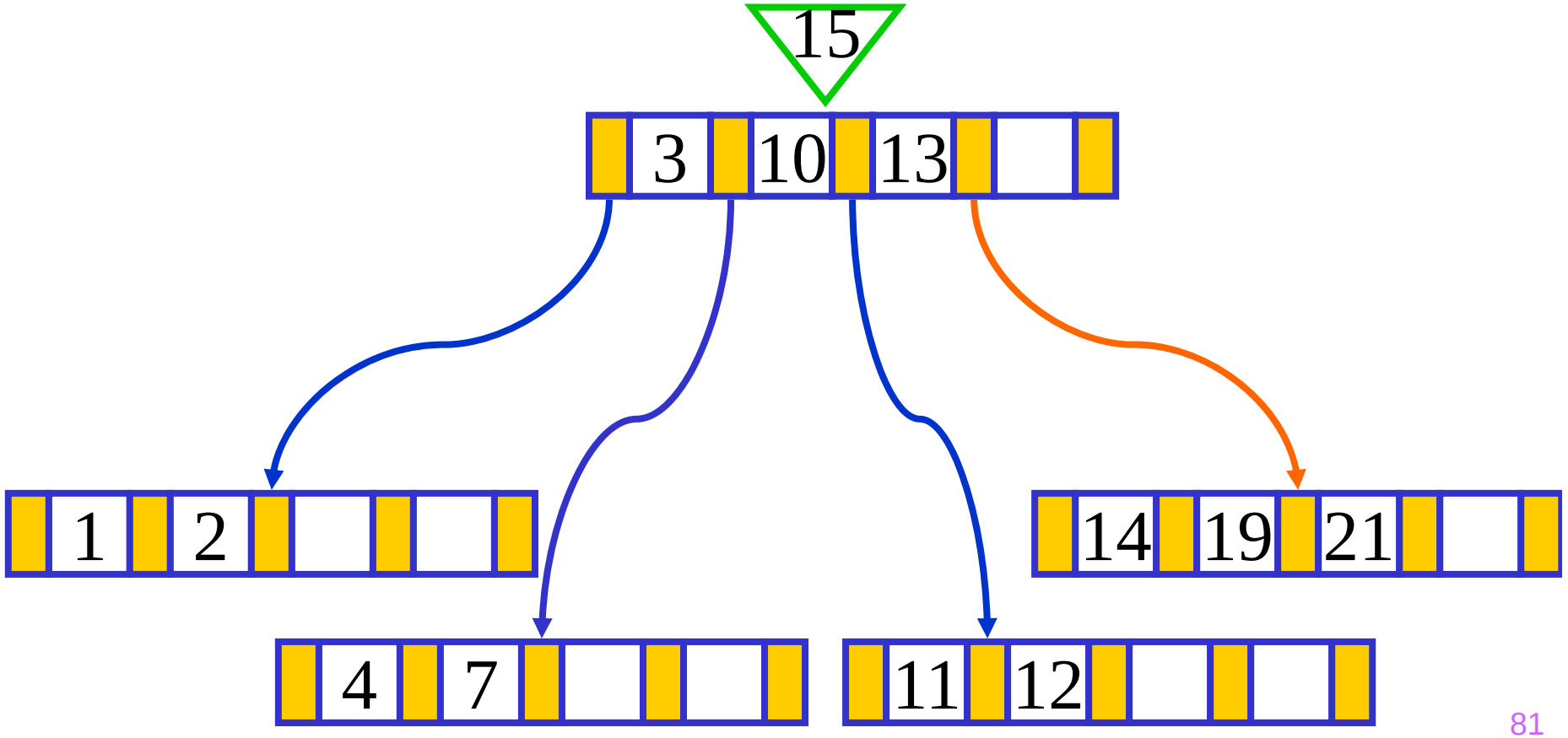
Sukzessiver Aufbau eines B-Baums vom Grad k=2



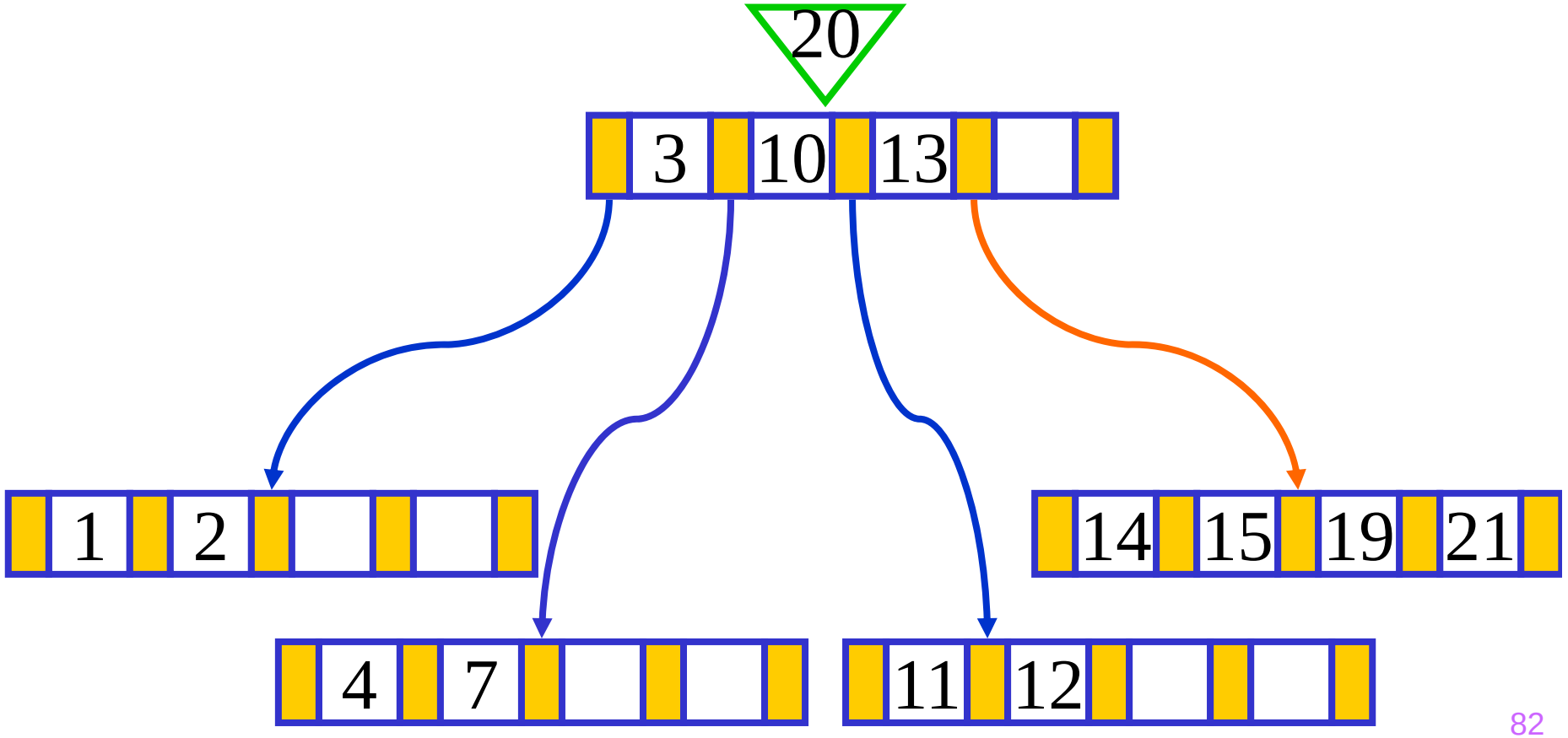
Sukzessiver Aufbau eines B-Baums vom Grad k=2



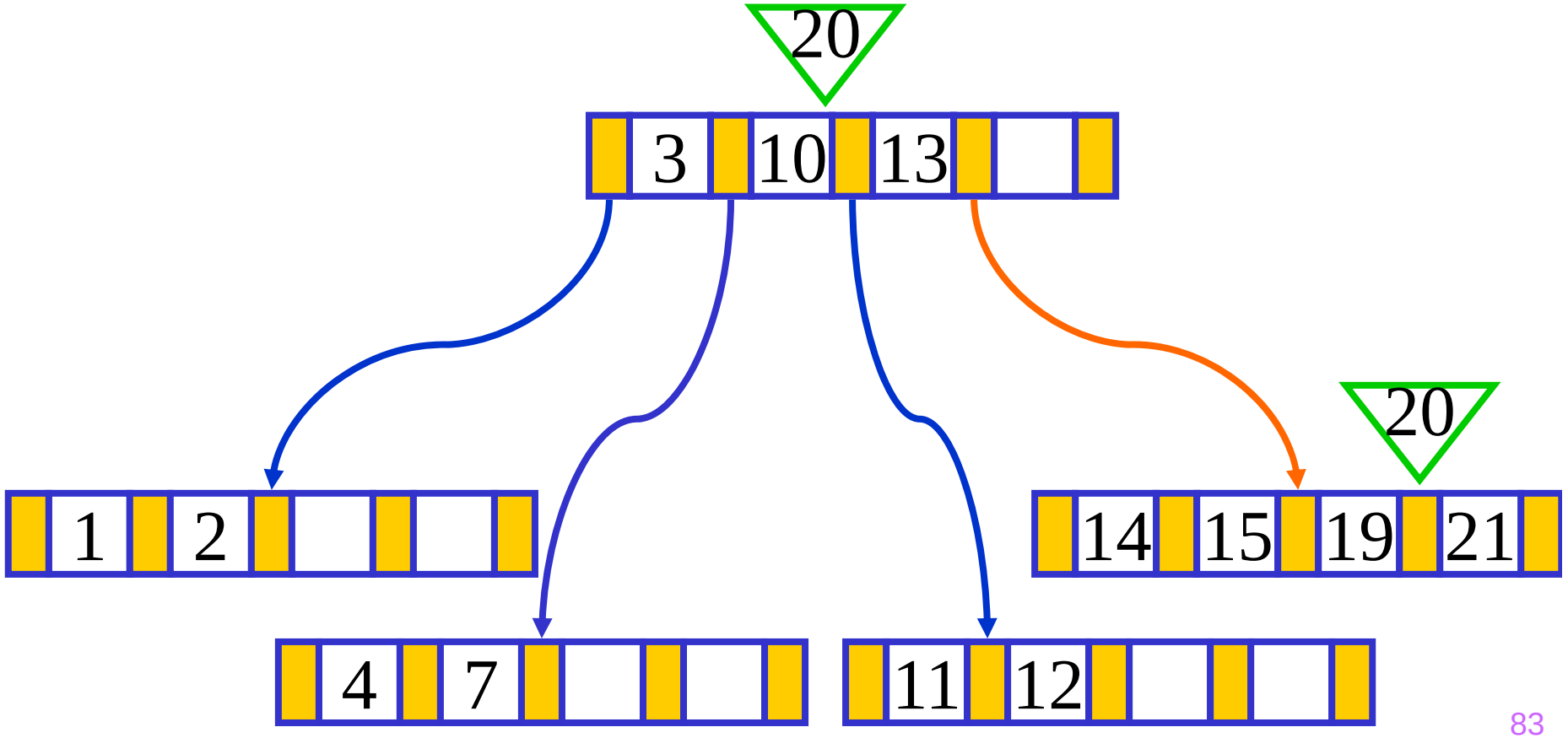
Sukzessiver Aufbau eines B-Baums vom Grad k=2



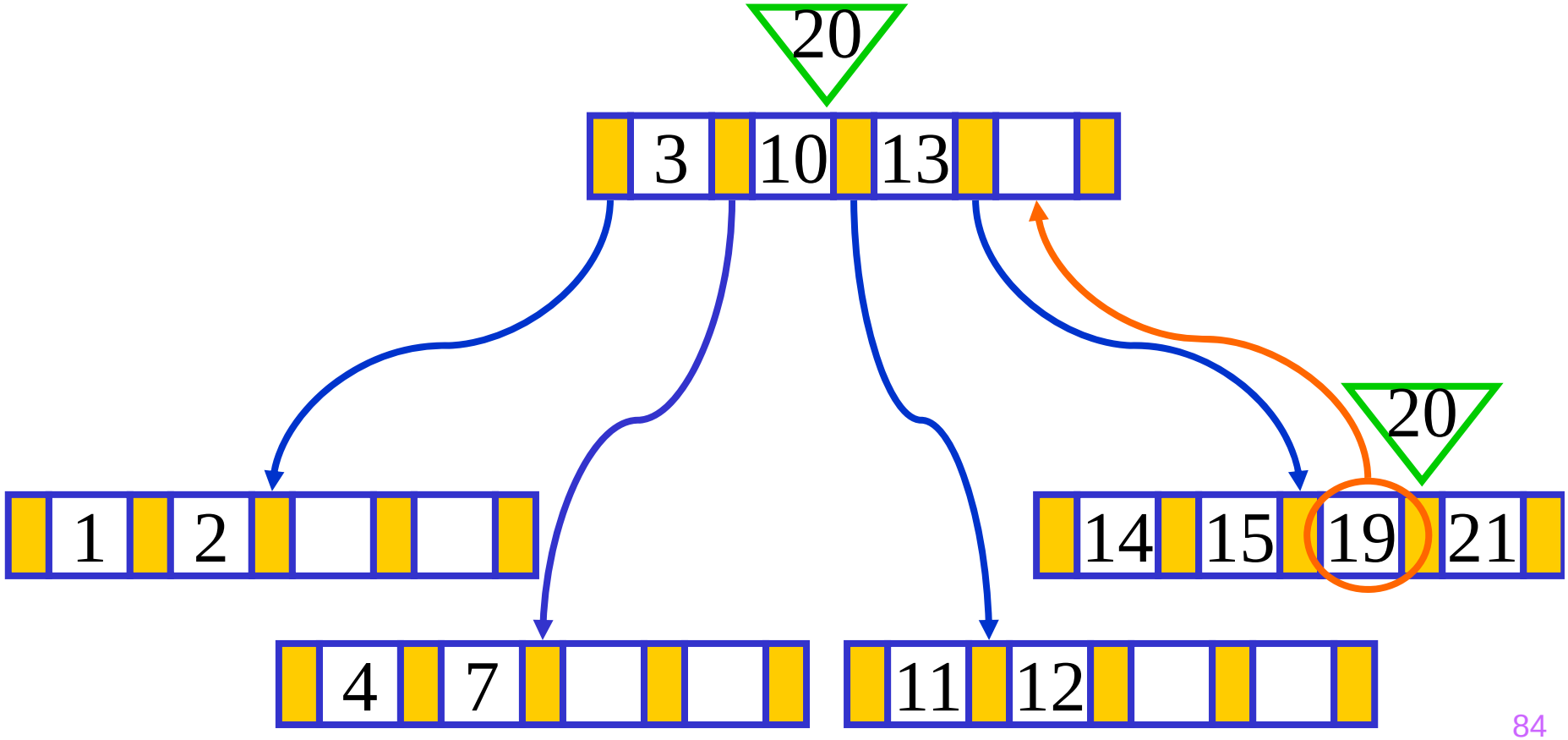
Sukzessiver Aufbau eines B-Baums vom Grad k=2



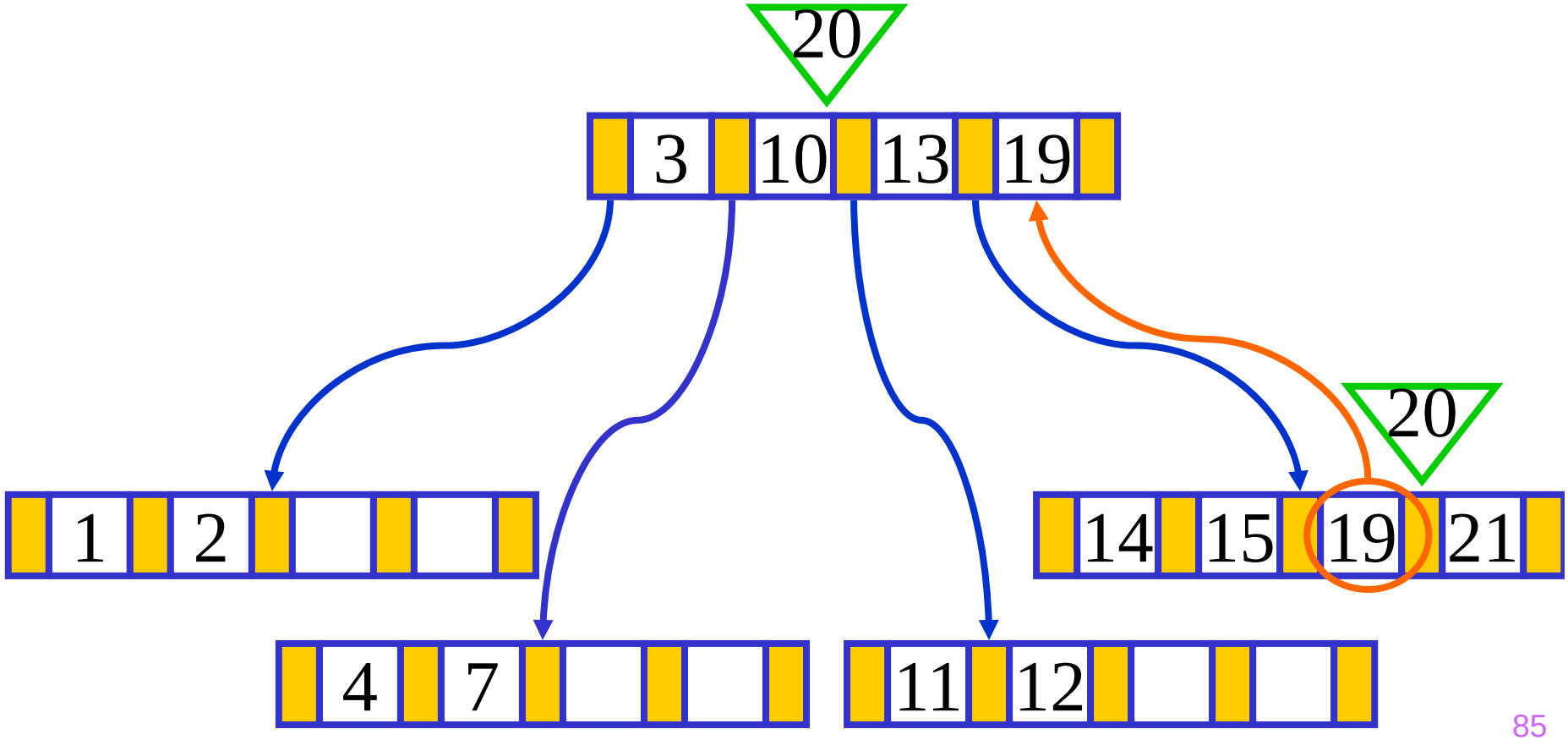
Sukzessiver Aufbau eines B-Baums vom Grad k=2



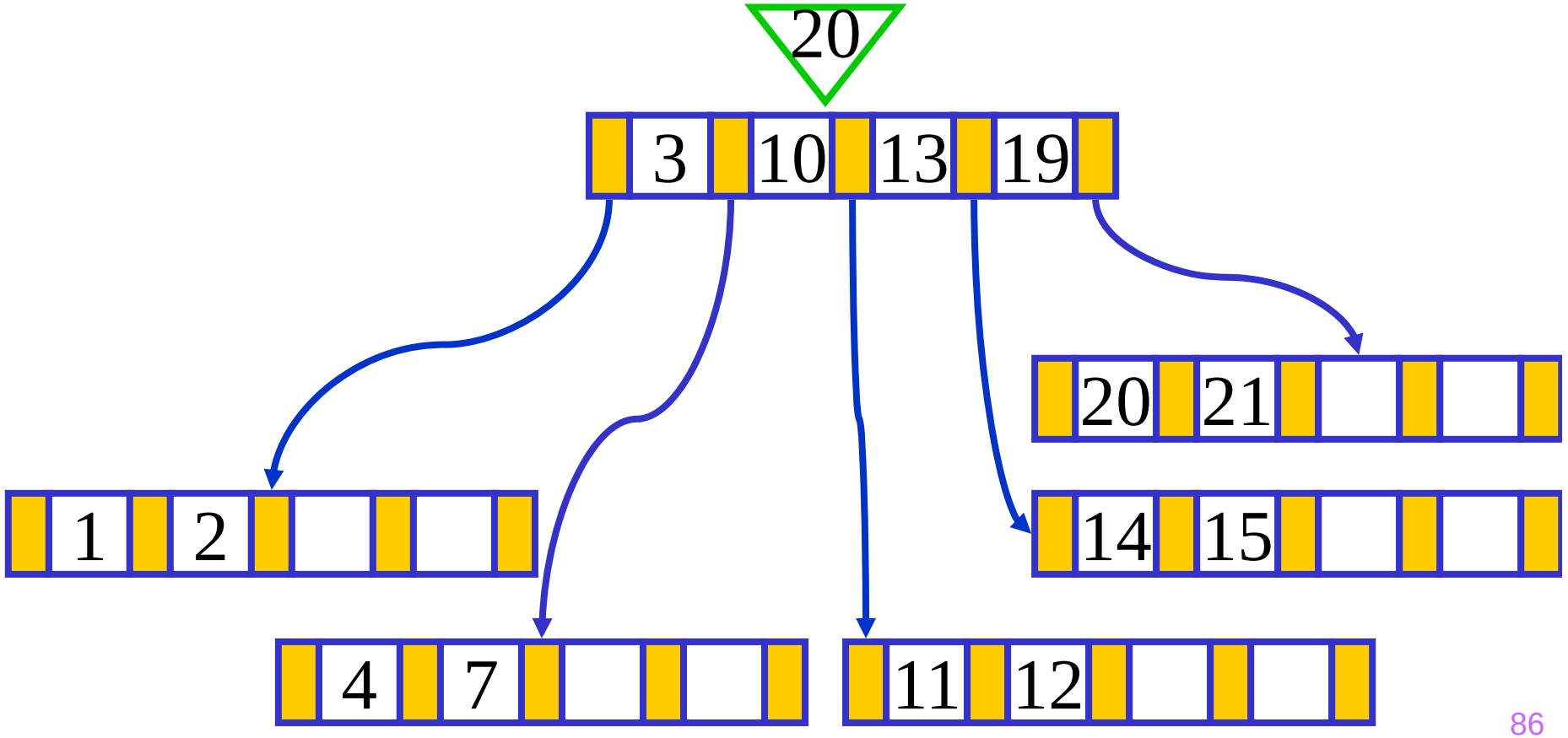
Sukzessiver Aufbau eines B-Baums vom Grad k=2



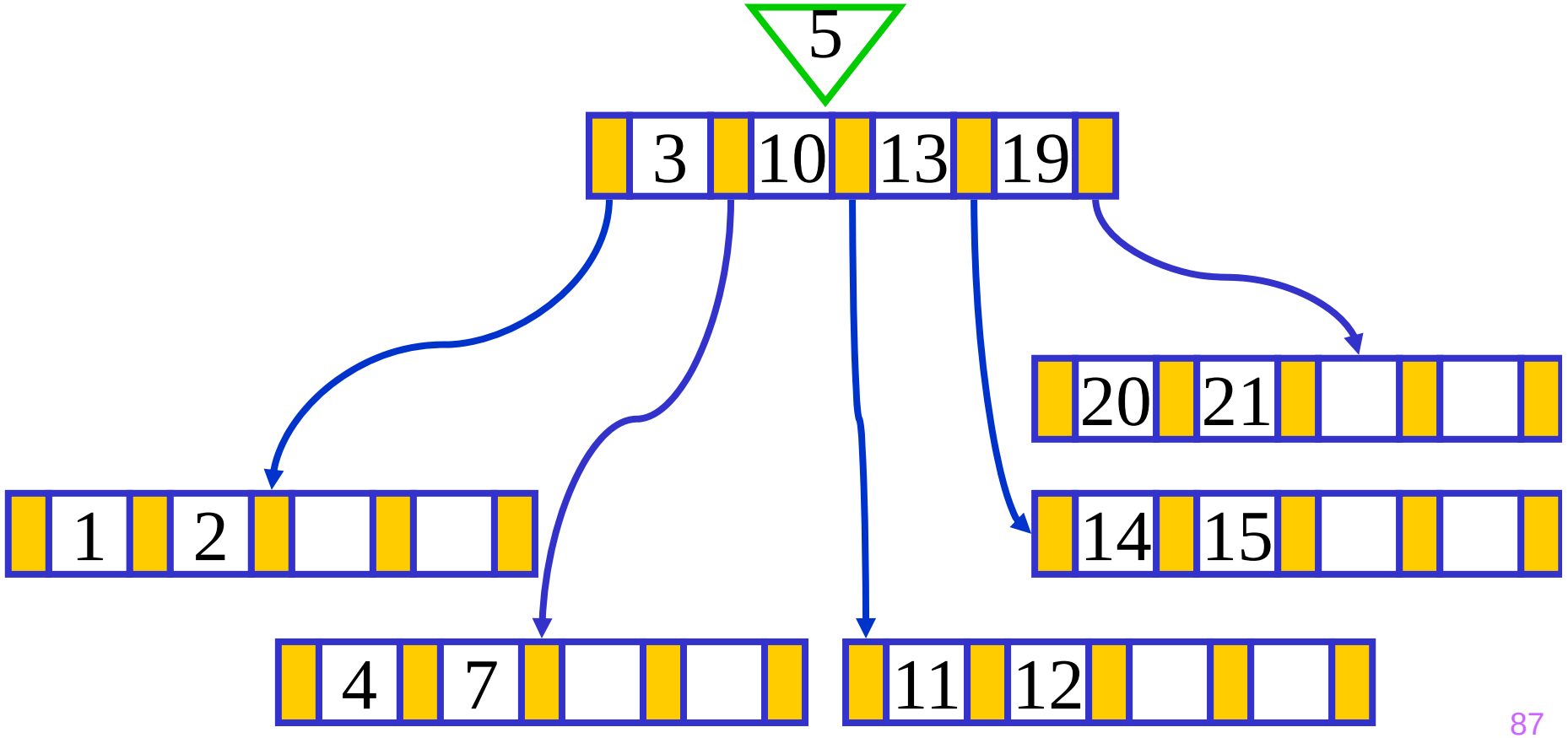
Sukzessiver Aufbau eines B-Baums vom Grad k=2



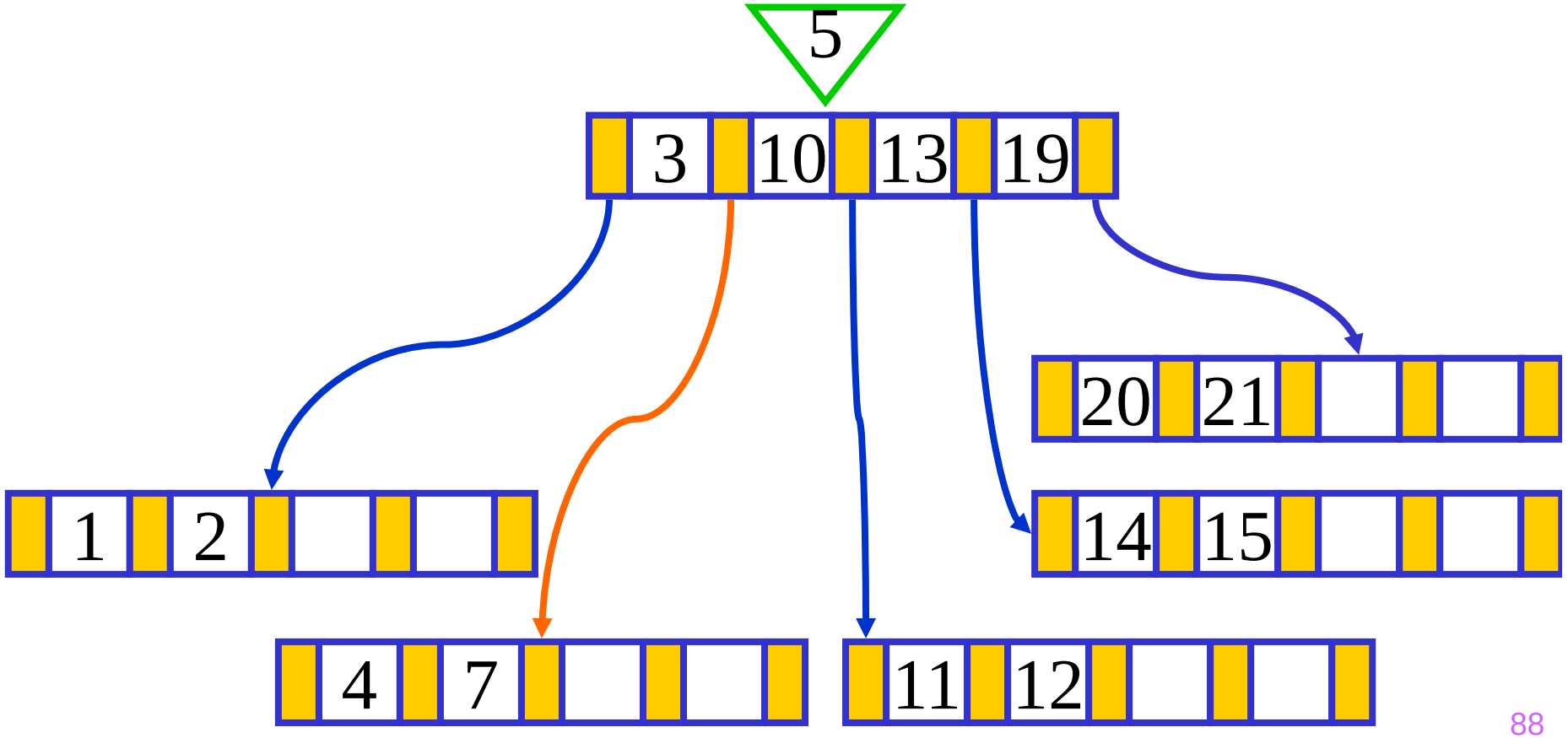
Sukzessiver Aufbau eines B-Baums vom Grad k=2



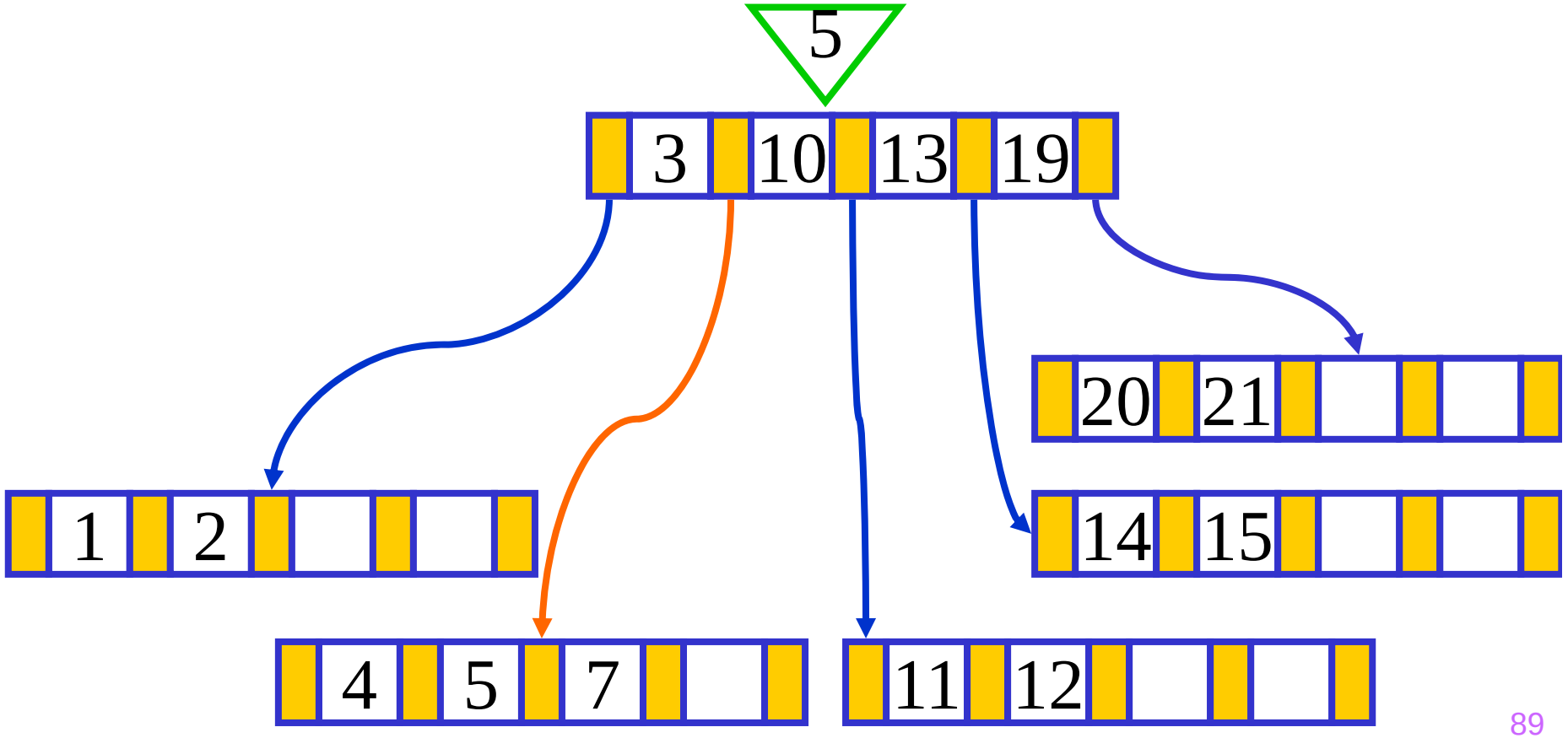
Sukzessiver Aufbau eines B-Baums vom Grad k=2



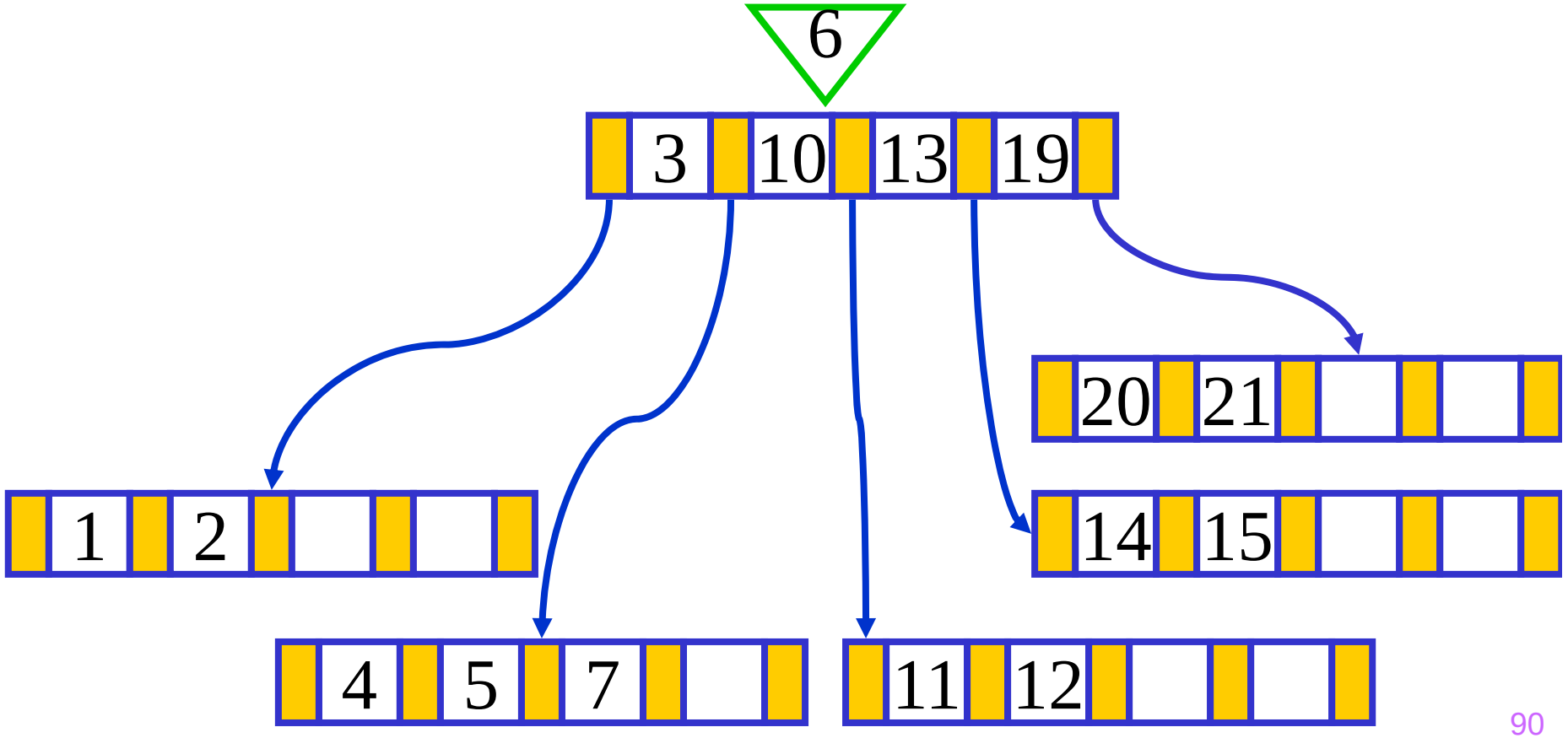
Sukzessiver Aufbau eines B-Baums vom Grad k=2



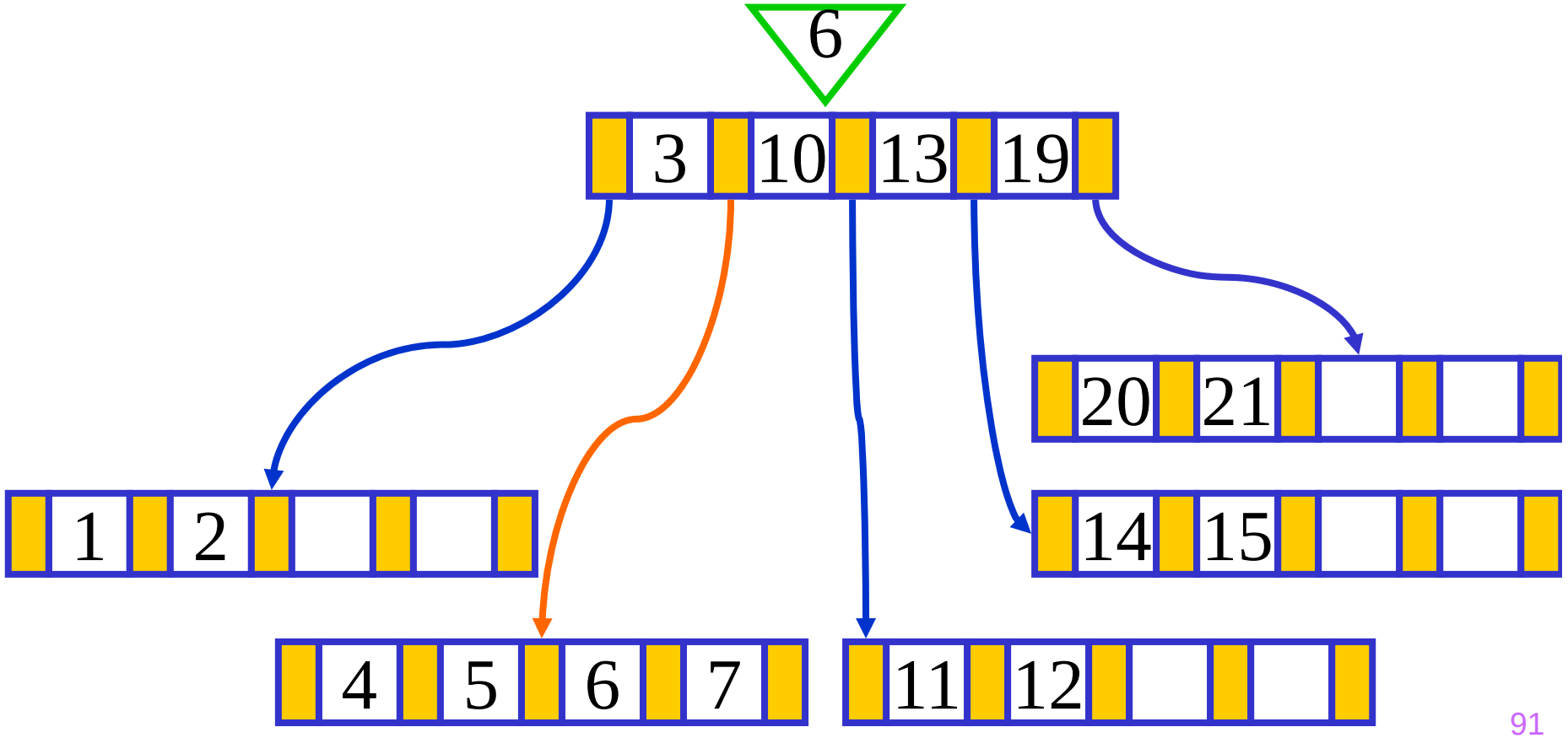
Sukzessiver Aufbau eines B-Baums vom Grad k=2



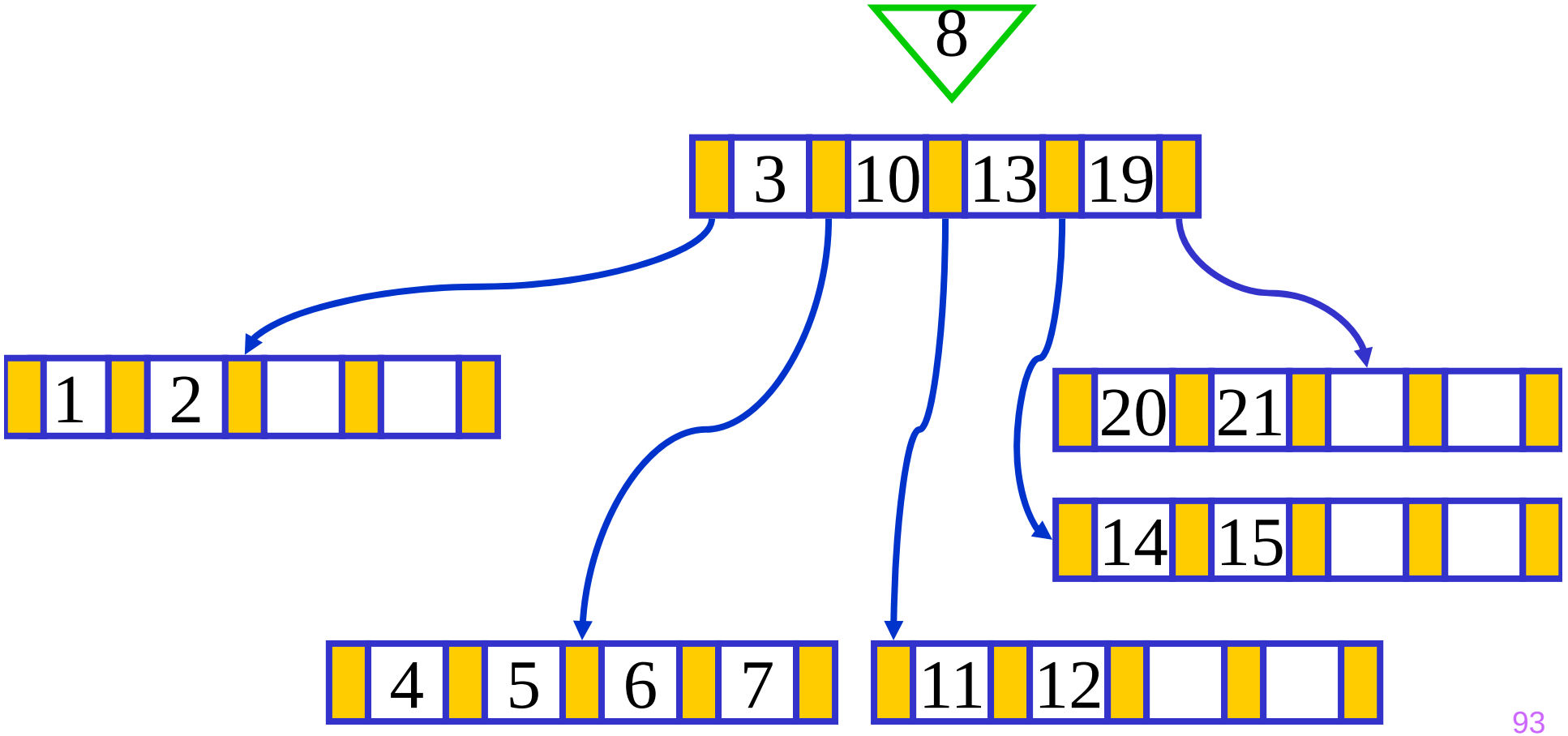
Sukzessiver Aufbau eines B-Baums vom Grad k=2



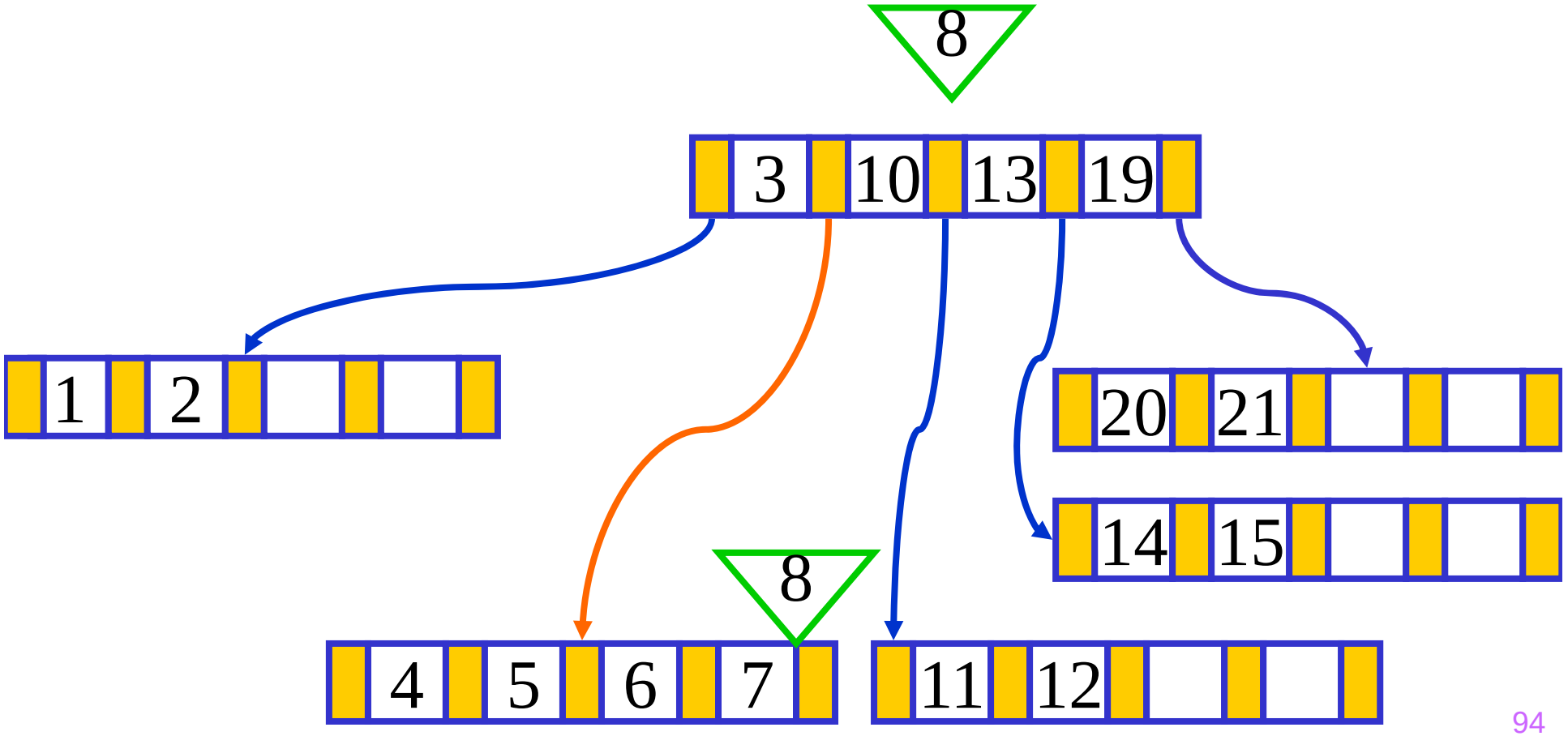
Sukzessiver Aufbau eines B-Baums vom Grad k=2



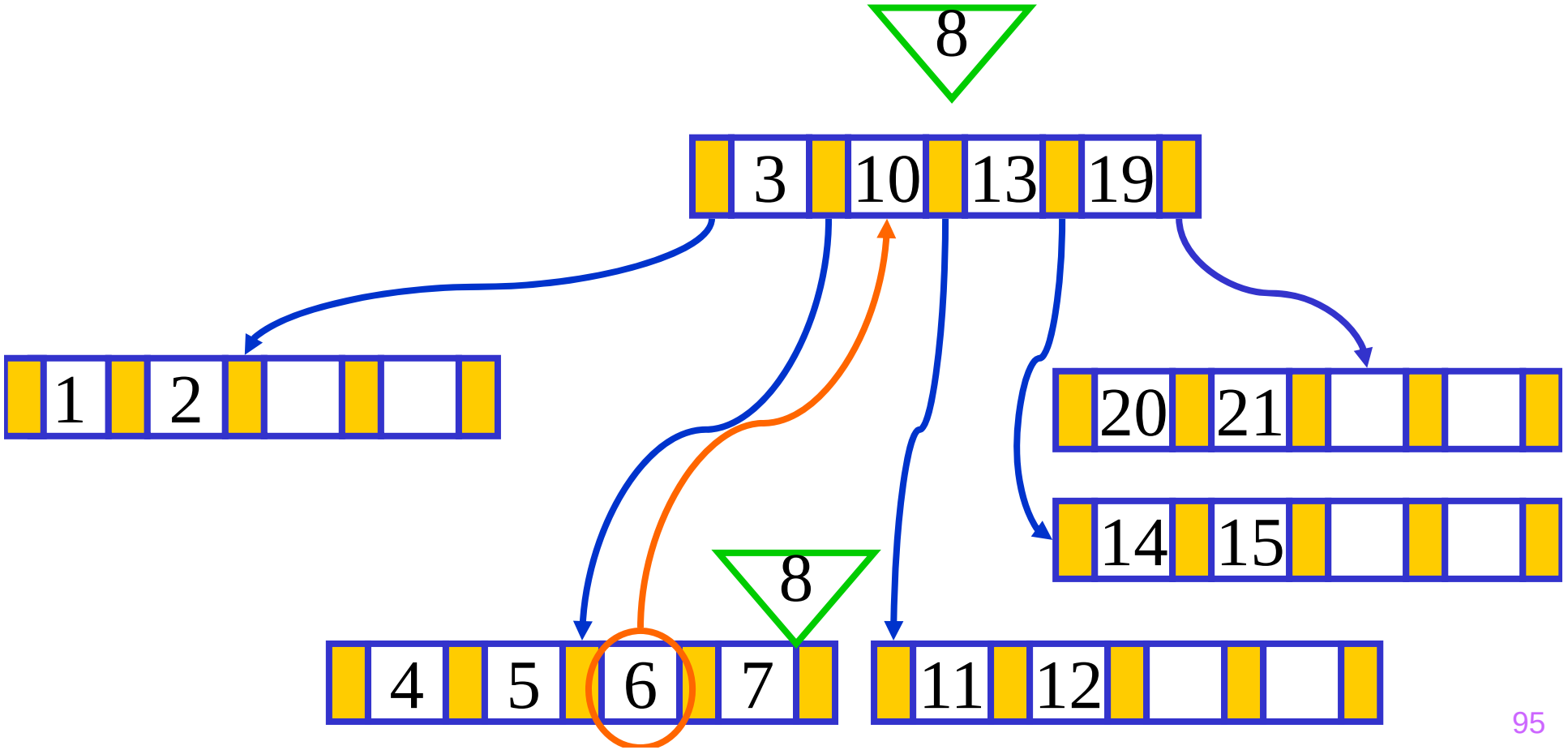
Sukzessiver Aufbau eines B-Baums vom Grad k=2



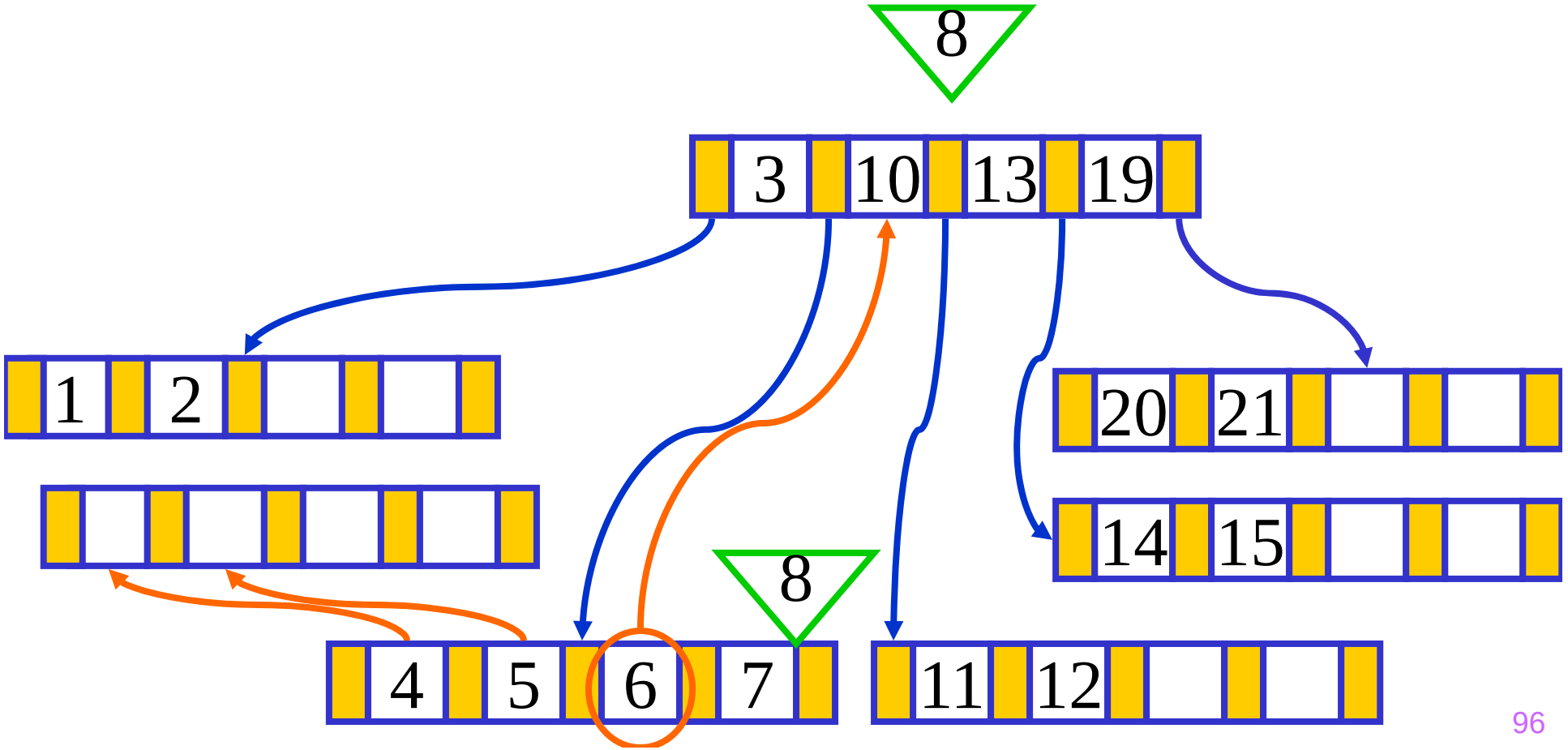
Sukzessiver Aufbau eines B-Baums vom Grad k=2



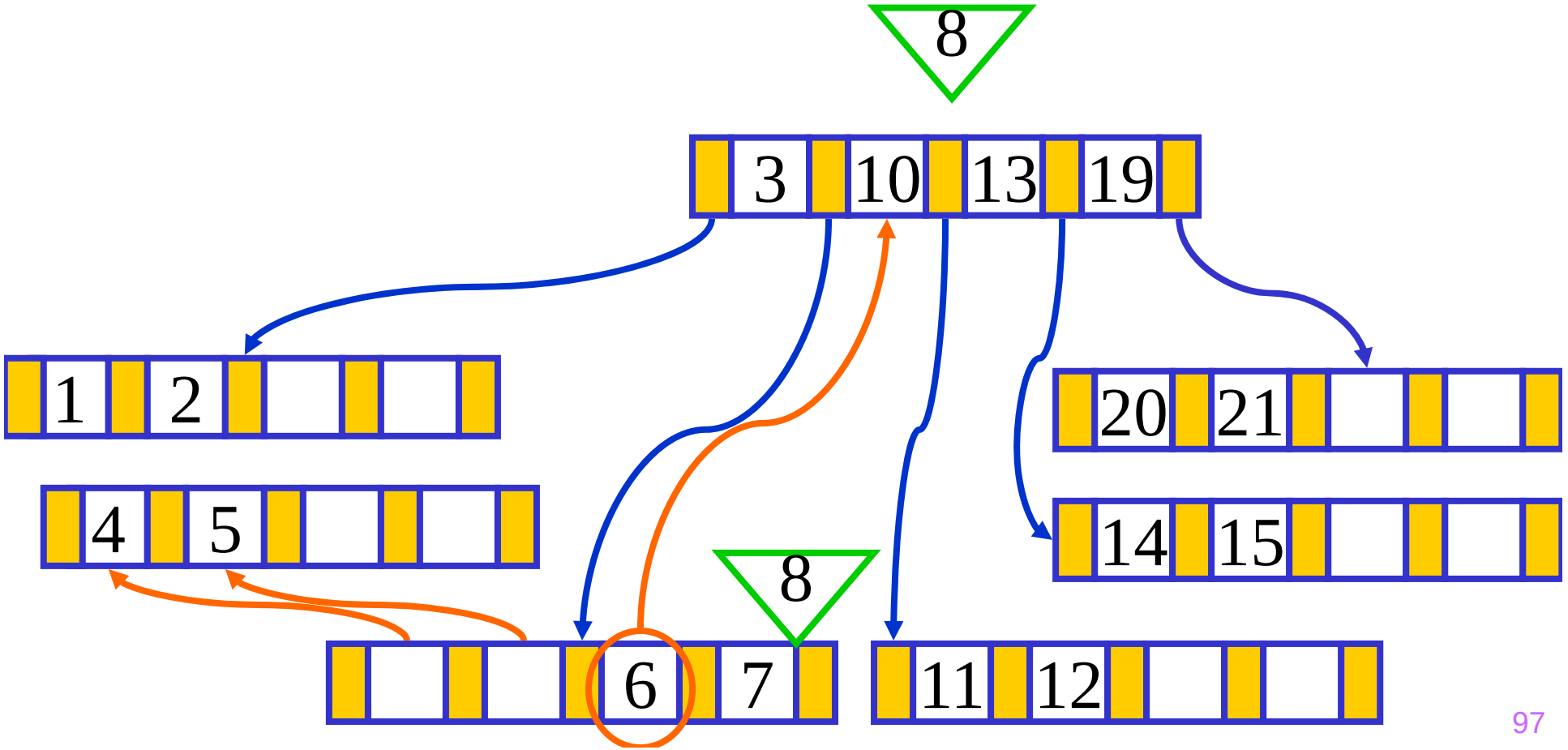
Sukzessiver Aufbau eines B-Baums vom Grad k=2



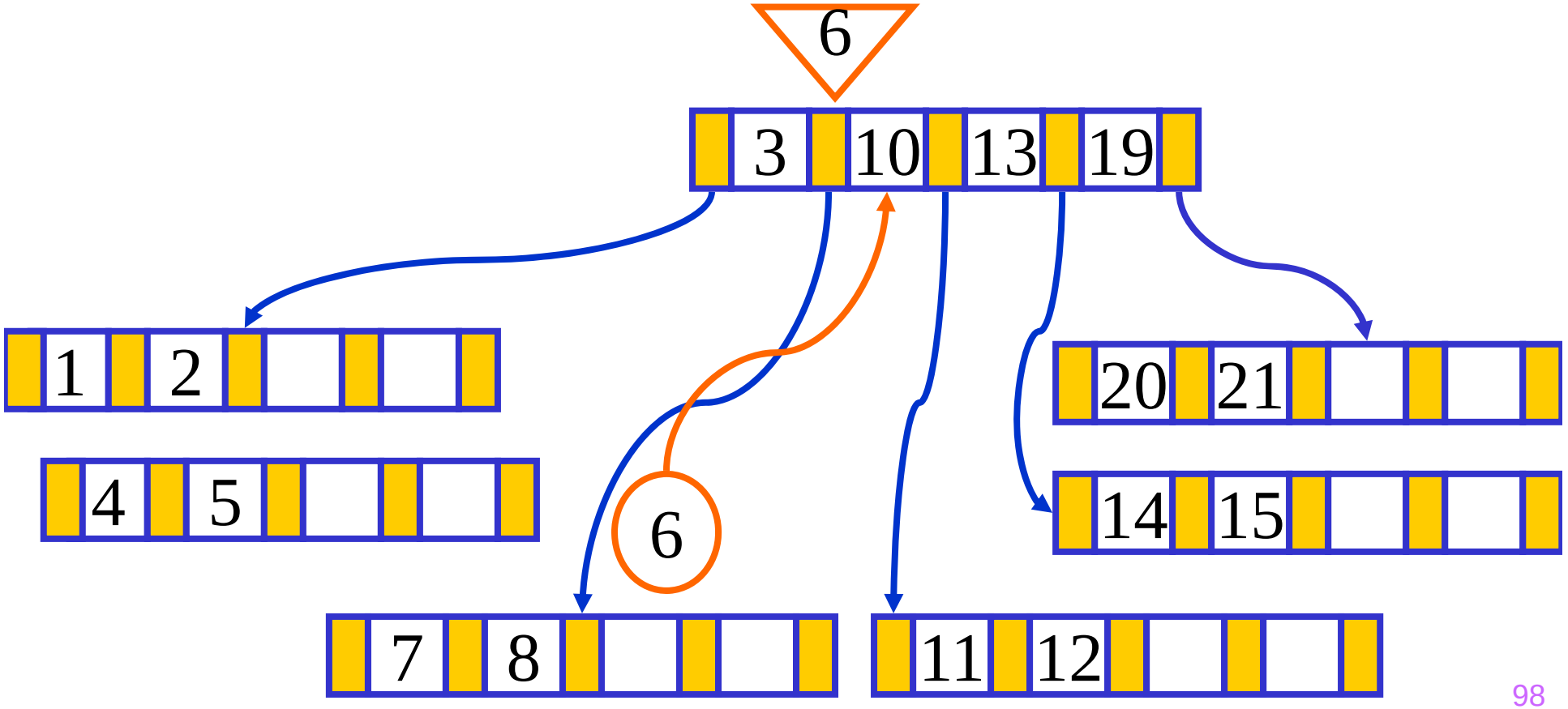
Sukzessiver Aufbau eines B-Baums vom Grad k=2



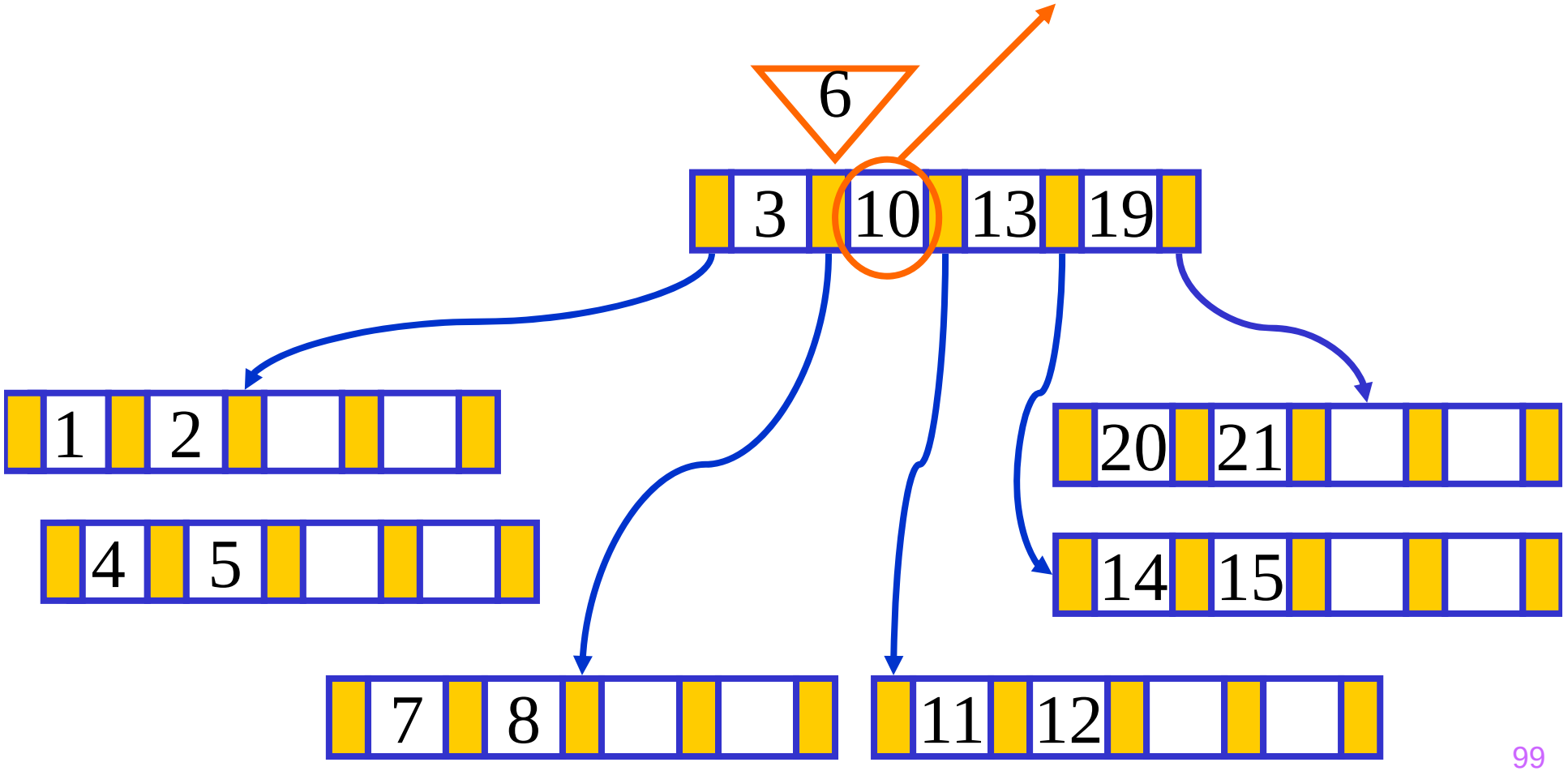
Sukzessiver Aufbau eines B-Baums vom Grad k=2



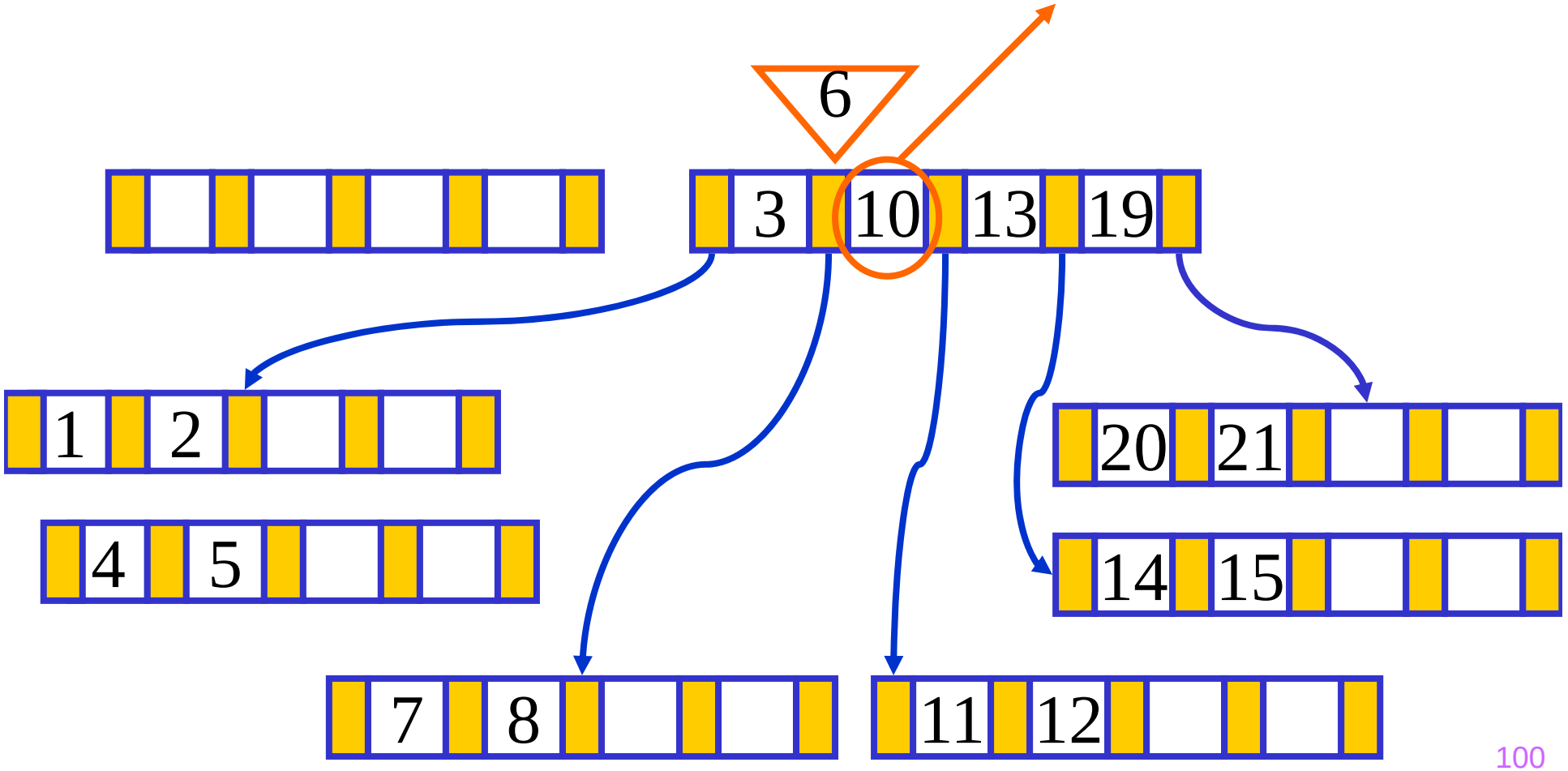
Sukzessiver Aufbau eines B-Baums vom Grad k=2



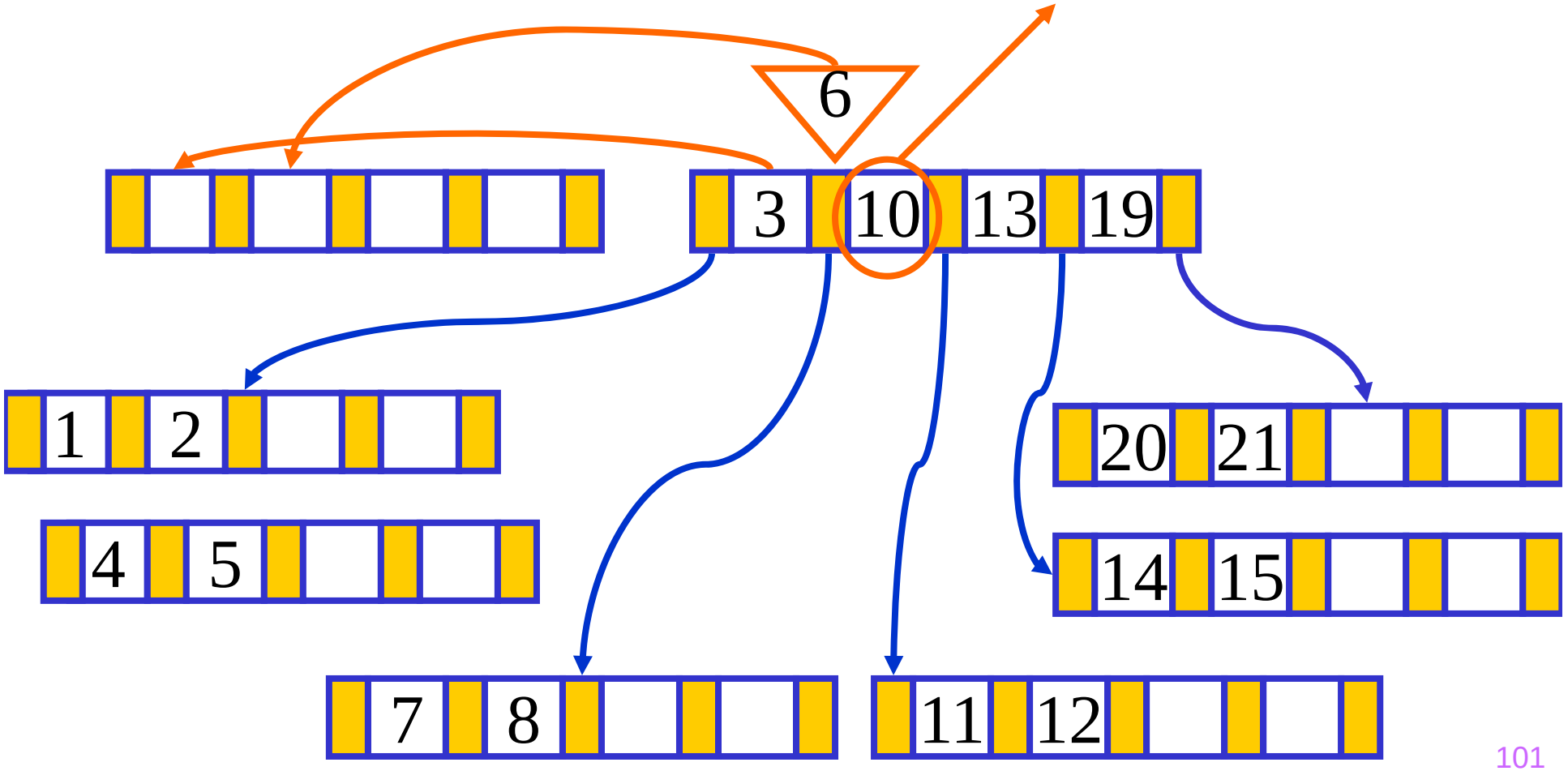
Sukzessiver Aufbau eines B-Baums vom Grad k=2



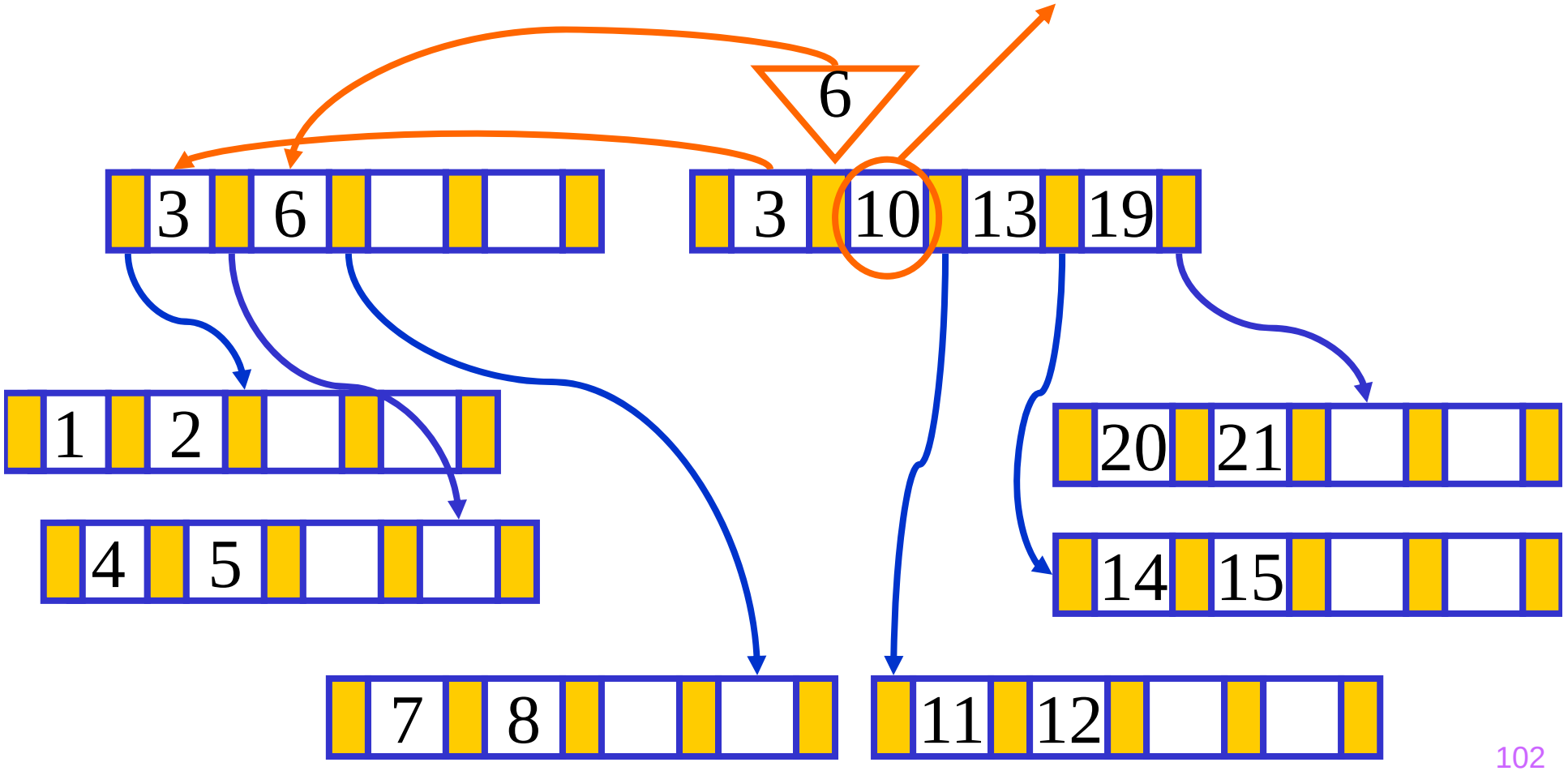
Sukzessiver Aufbau eines B-Baums vom Grad k=2



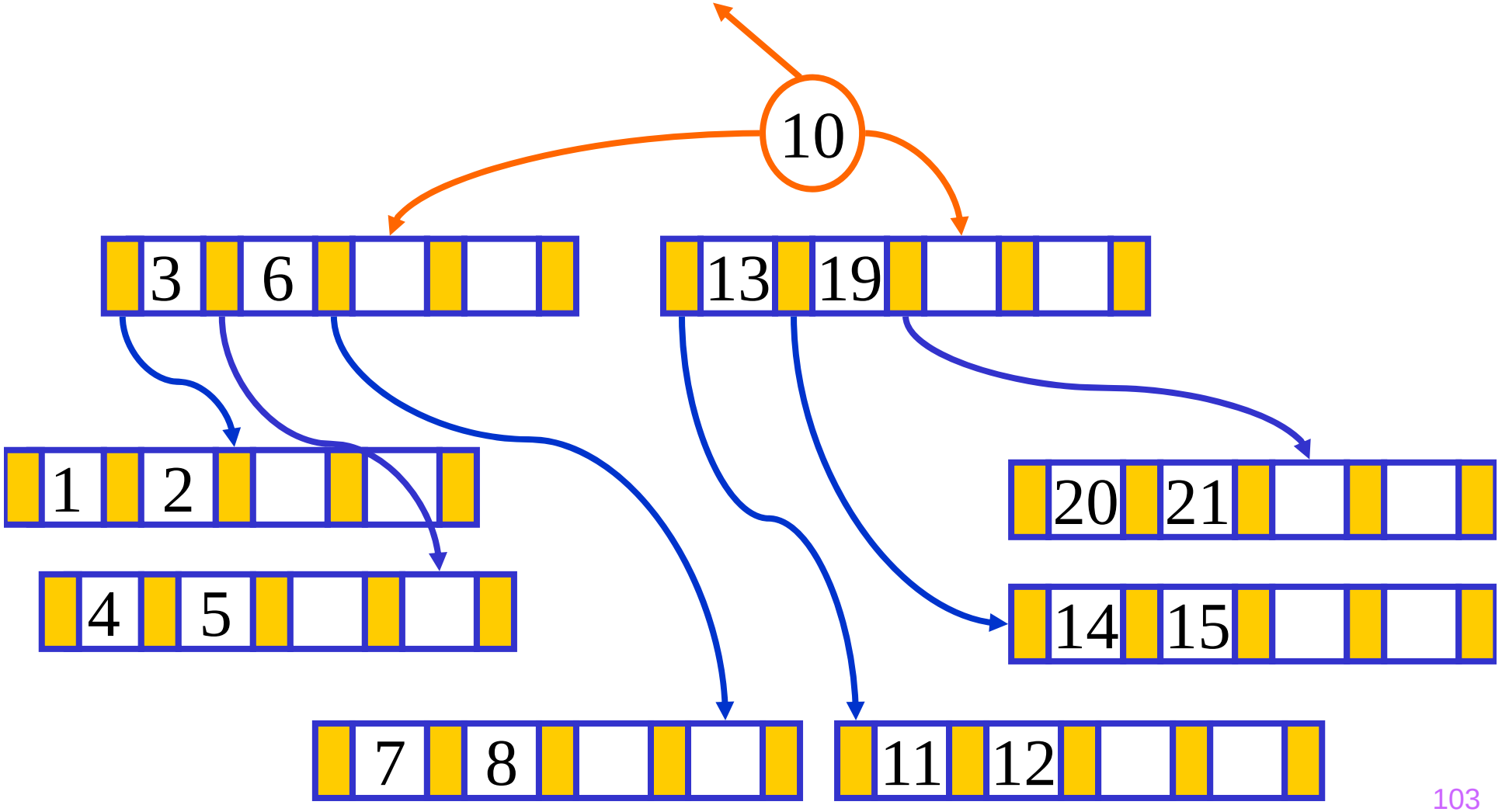
Sukzessiver Aufbau eines B-Baums vom Grad k=2



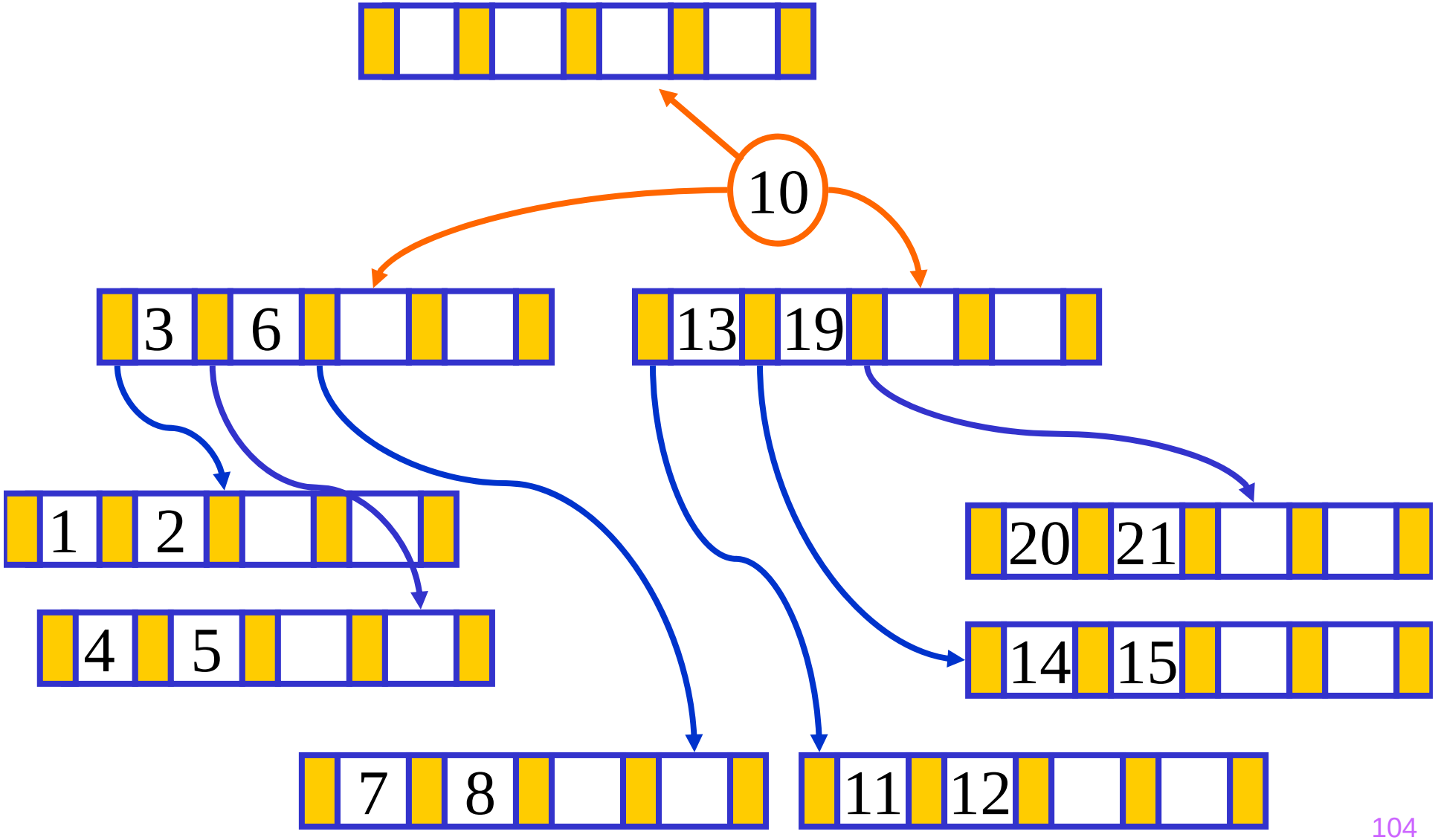
Sukzessiver Aufbau eines B-Baums vom Grad k=2



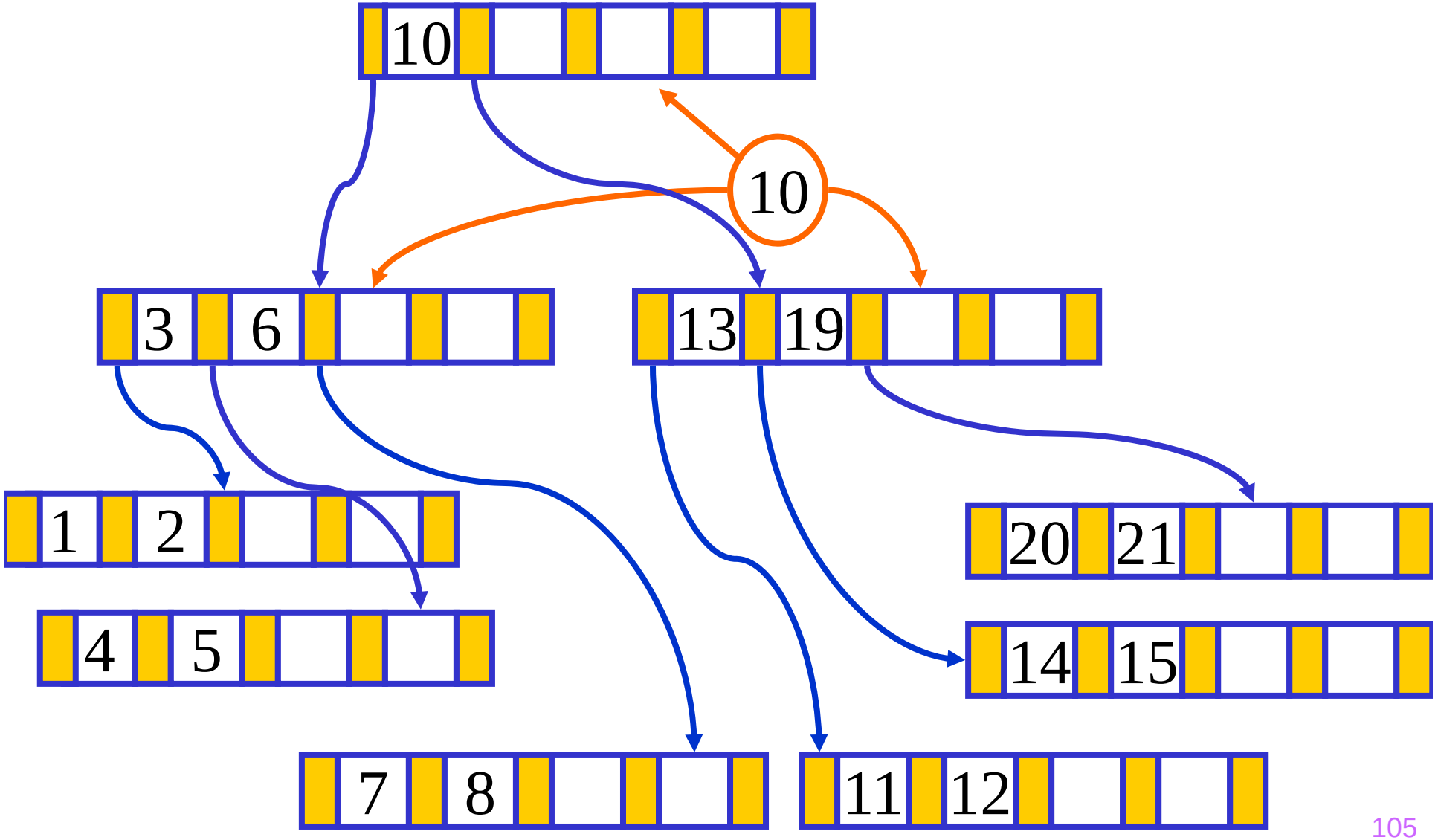
Sukzessiver Aufbau eines B-Baums vom Grad k=2



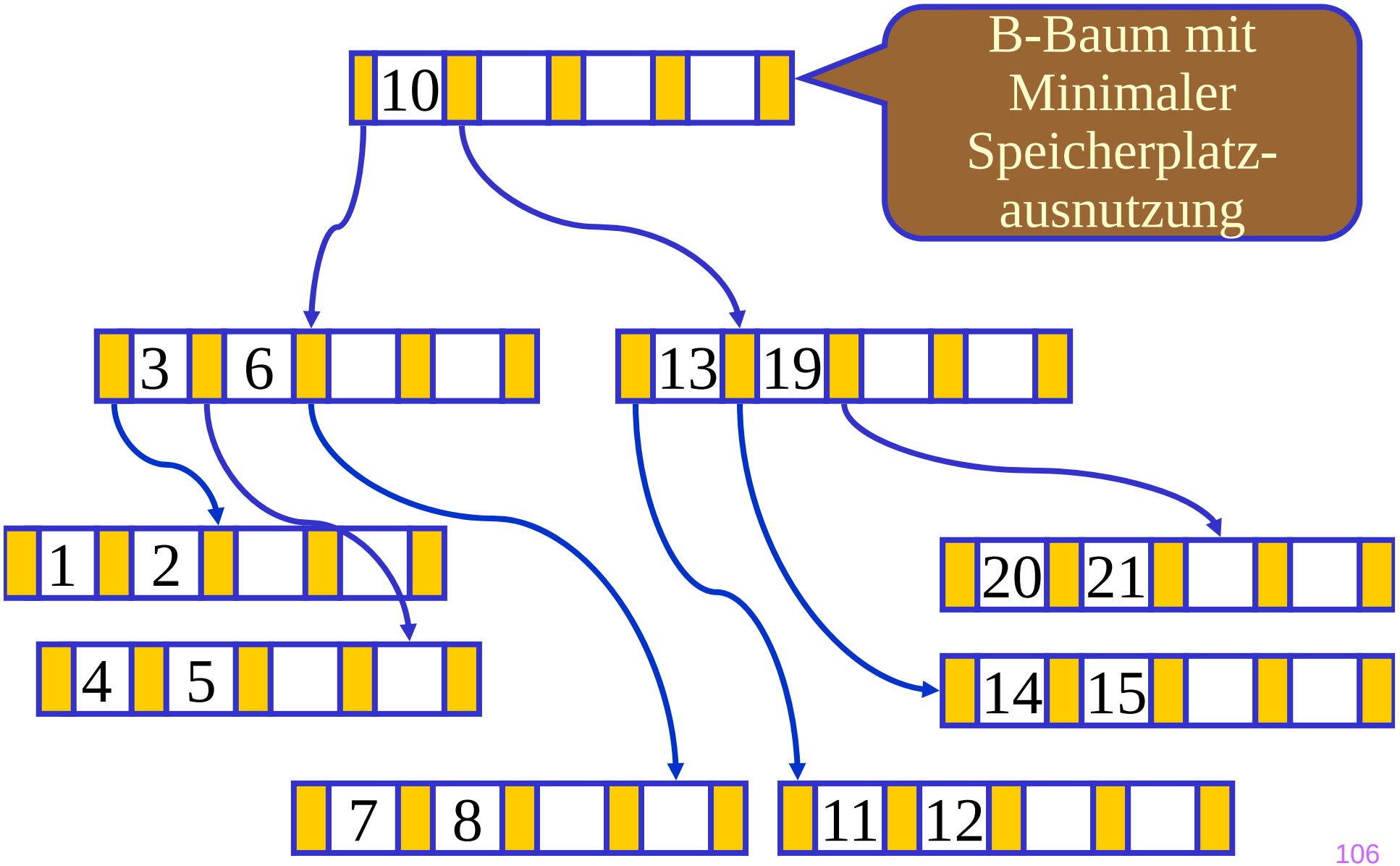
Sukzessiver Aufbau eines B-Baums vom Grad k=2



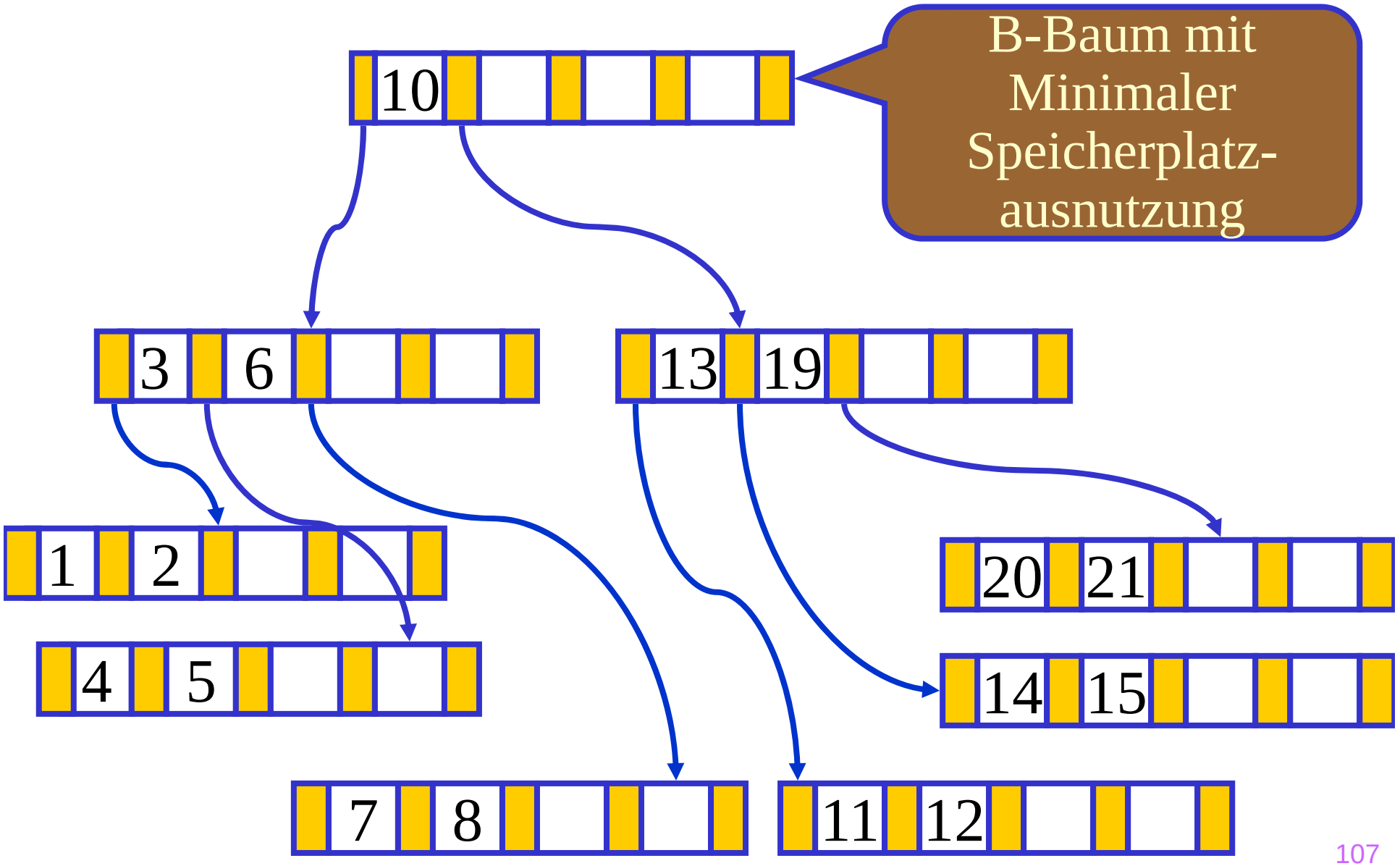
Sukzessiver Aufbau eines B-Baums vom Grad k=2



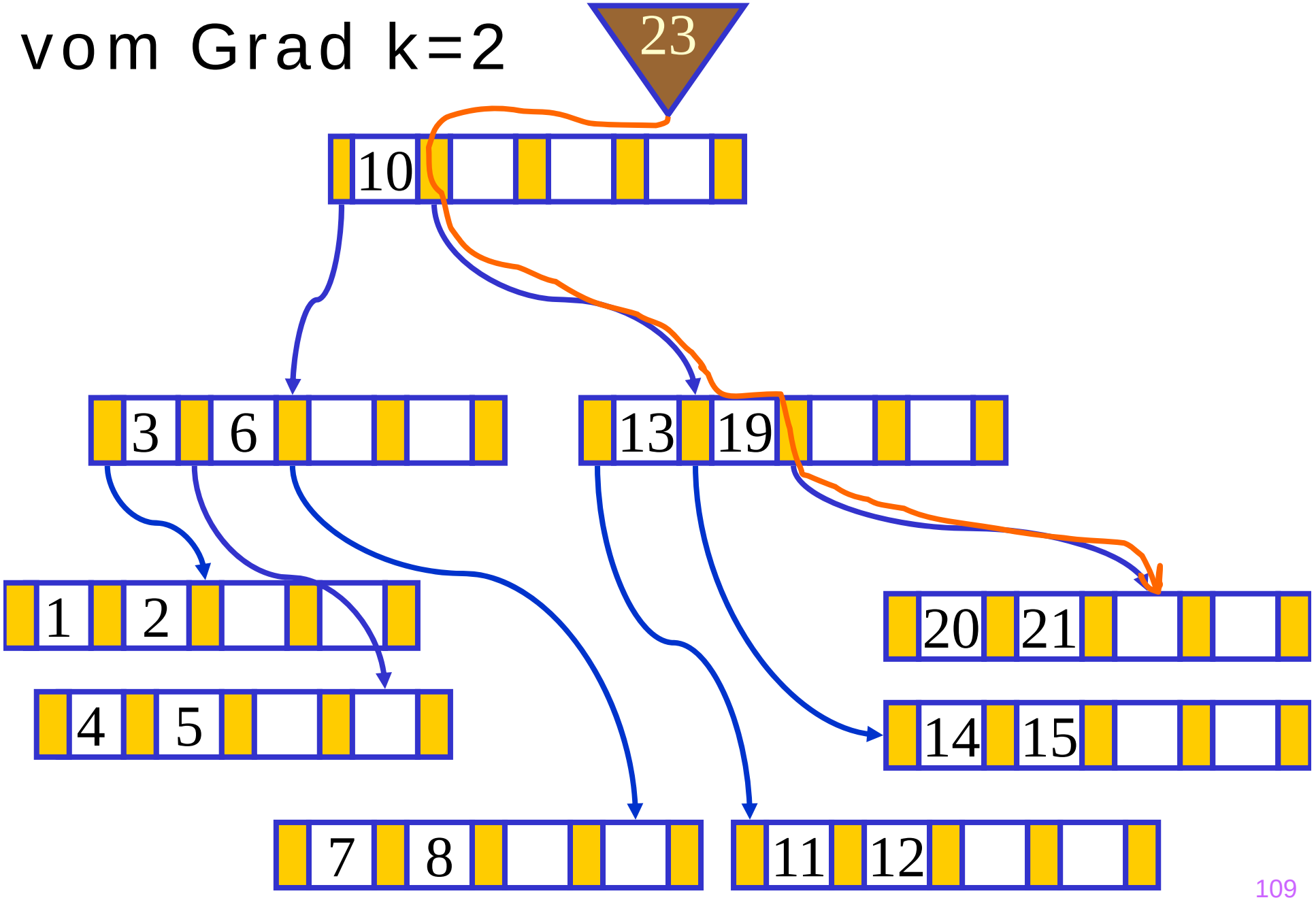
Sukzessiver Aufbau eines B-Baums vom Grad k=2



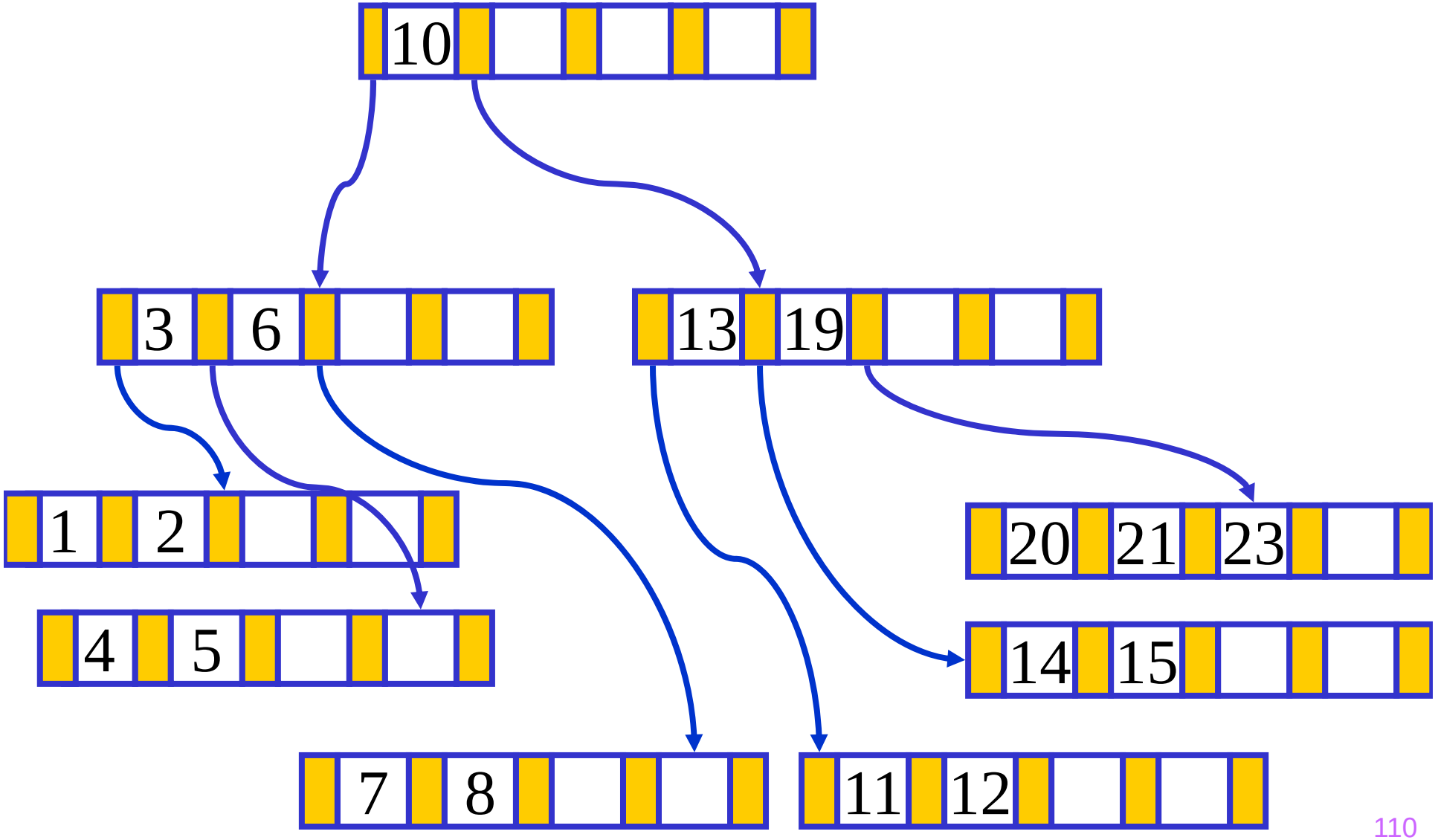
Sukzessiver Aufbau eines B-Baums vom Grad k=2



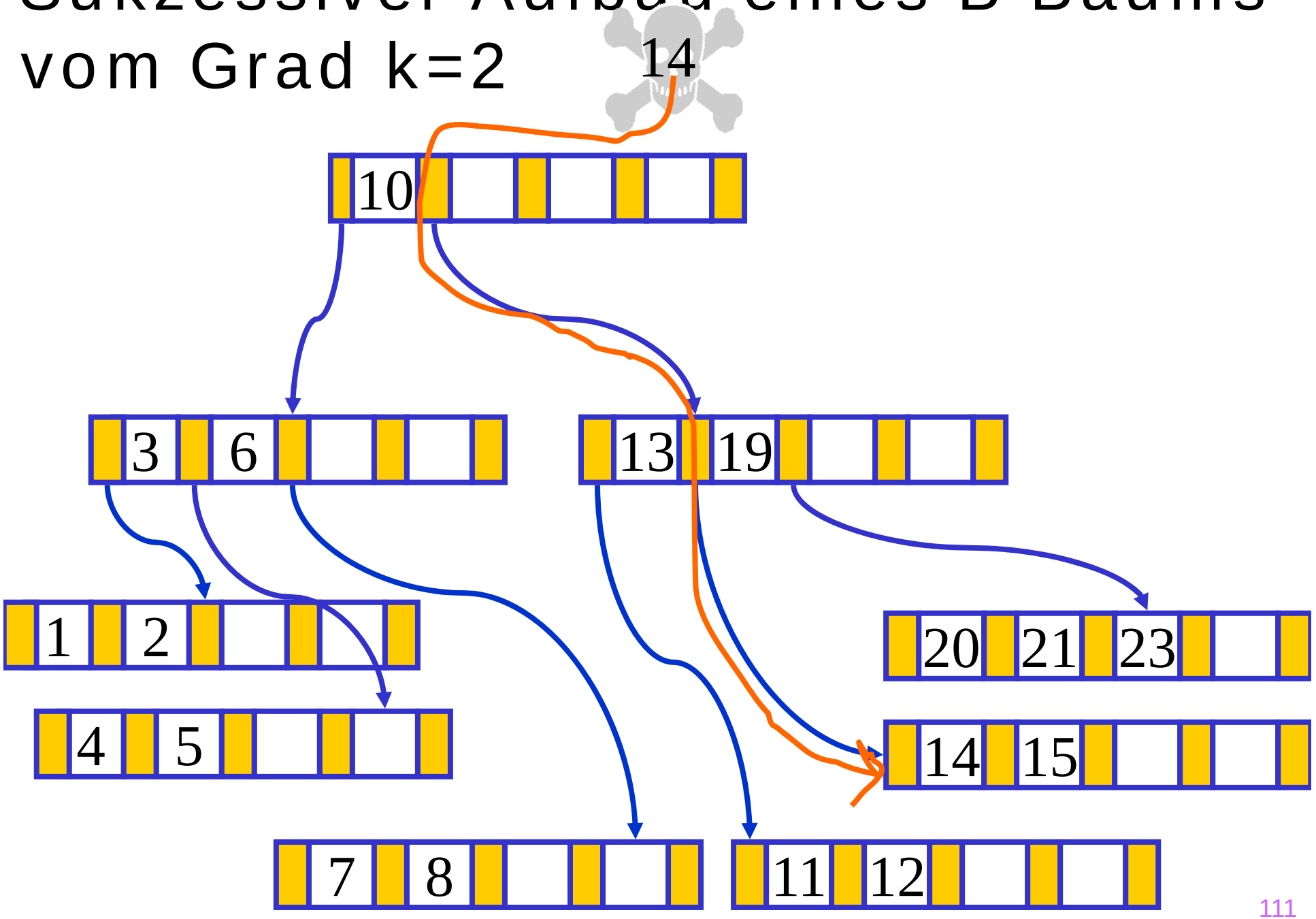
Sukzessiver Aufbau eines B-Baums vom Grad k=2



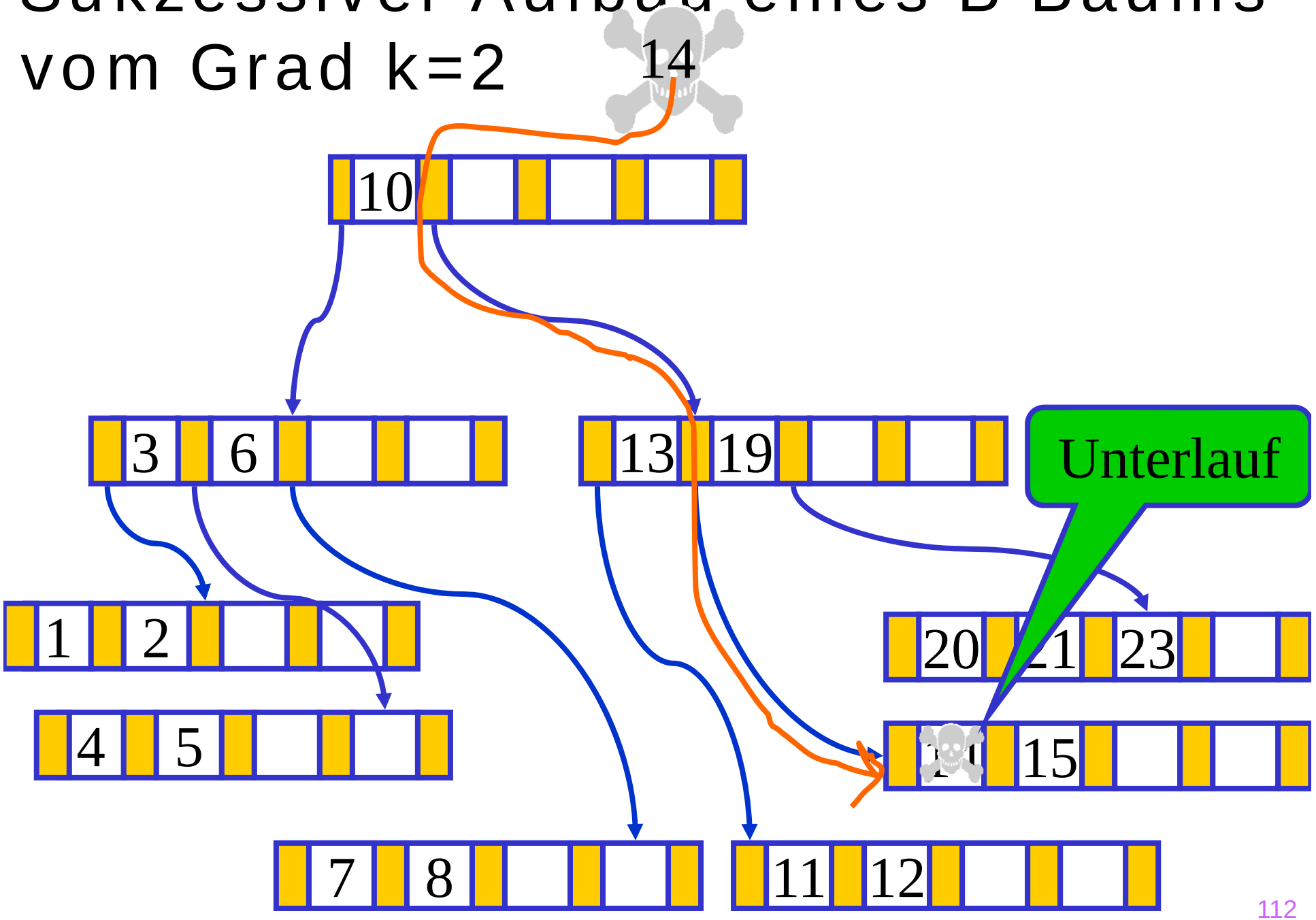
Sukzessiver Aufbau eines B-Baums vom Grad k=2



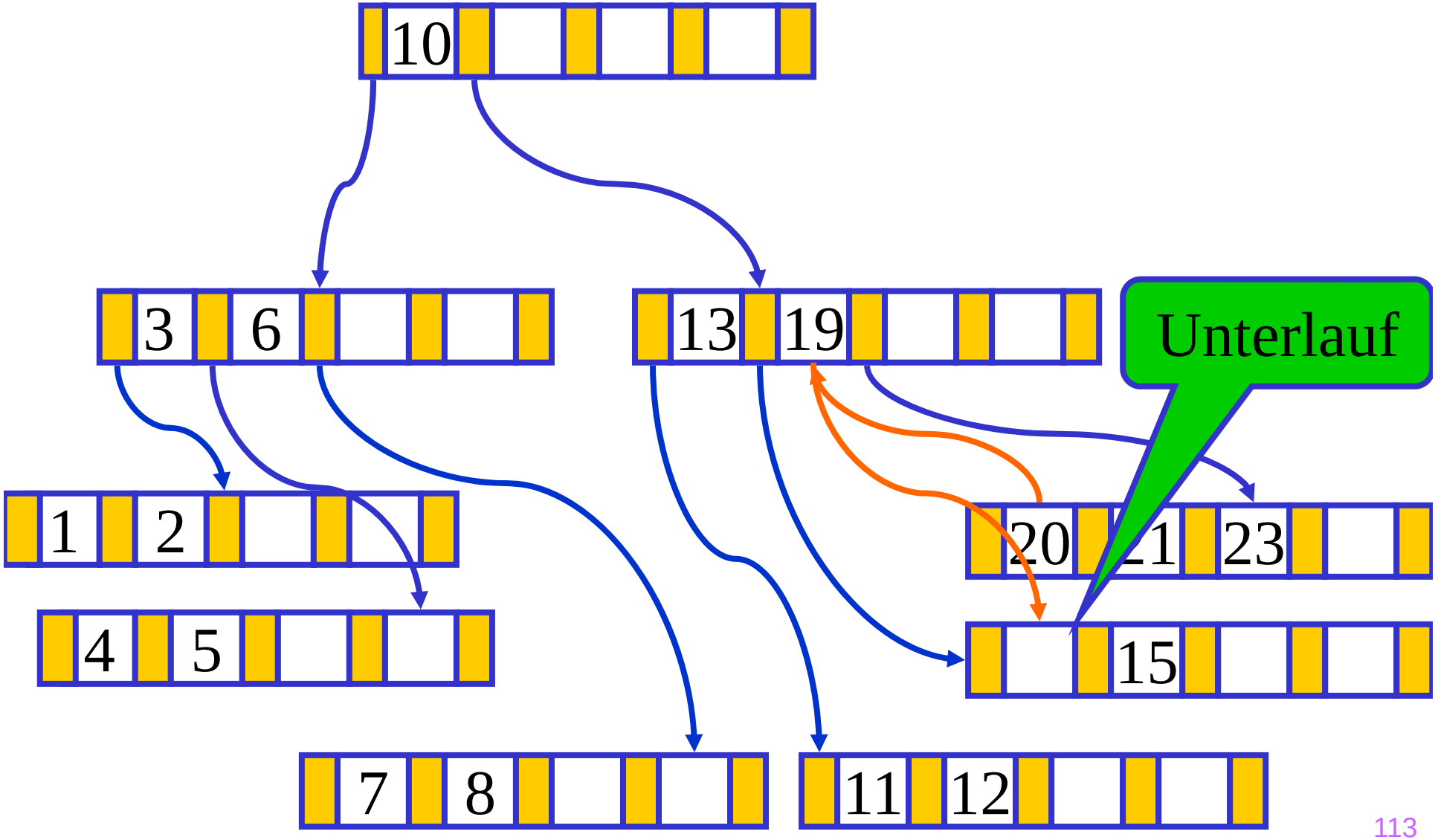
Sukzessiver Aufbau eines B-Baums vom Grad k=2



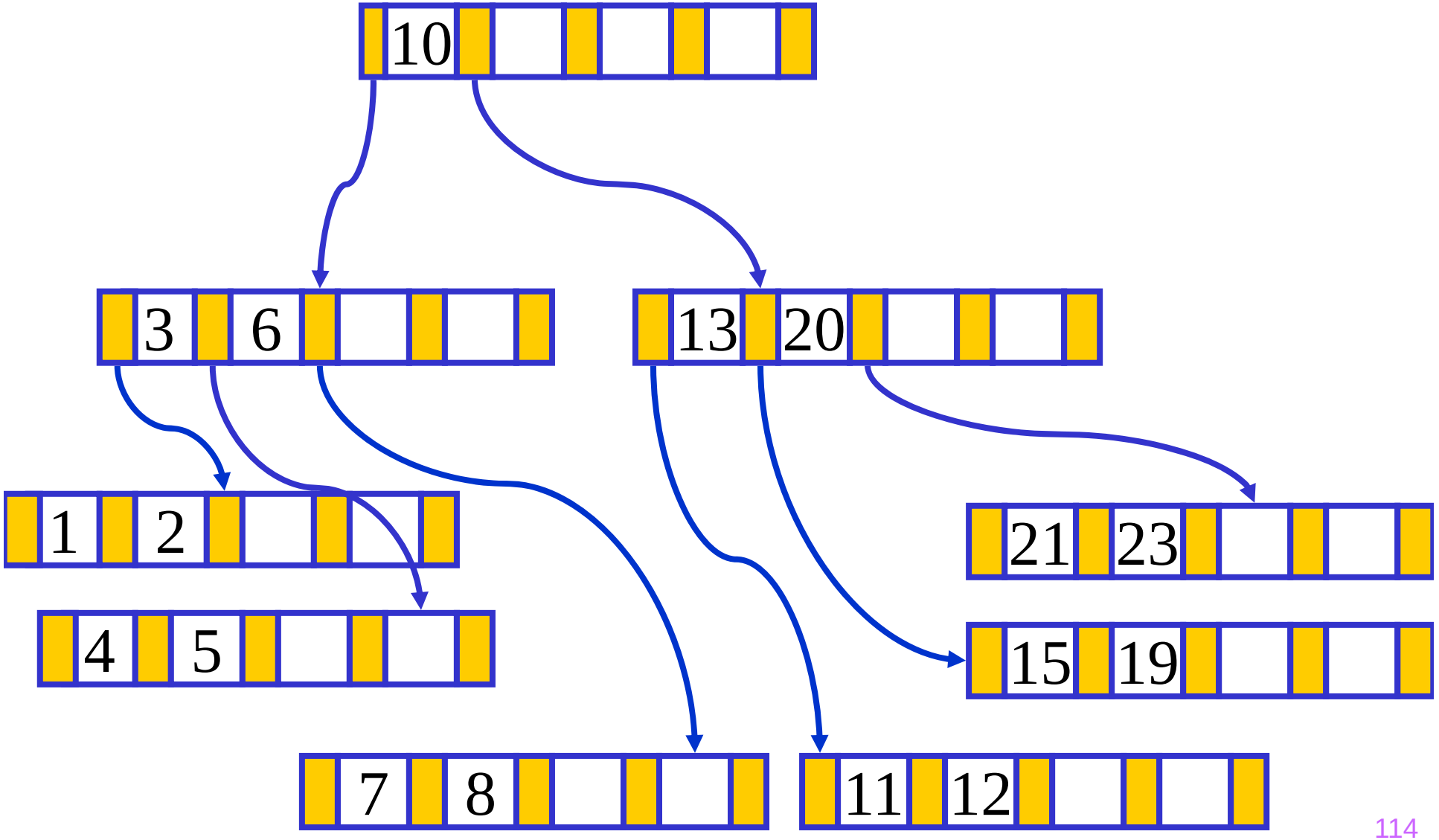
Sukzessiver Aufbau eines B-Baums vom Grad k=2



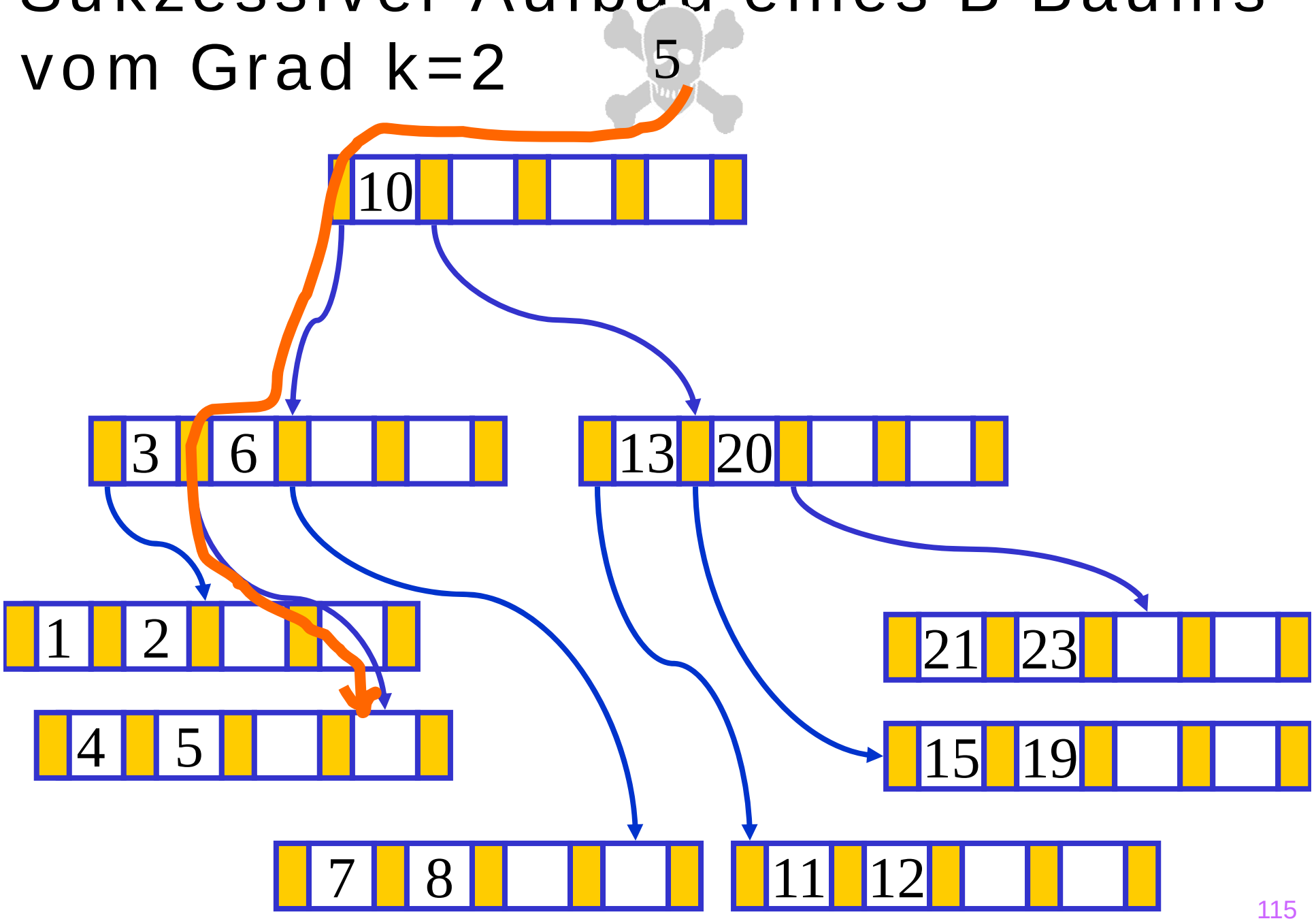
Sukzessiver Aufbau eines B-Baums vom Grad k=2



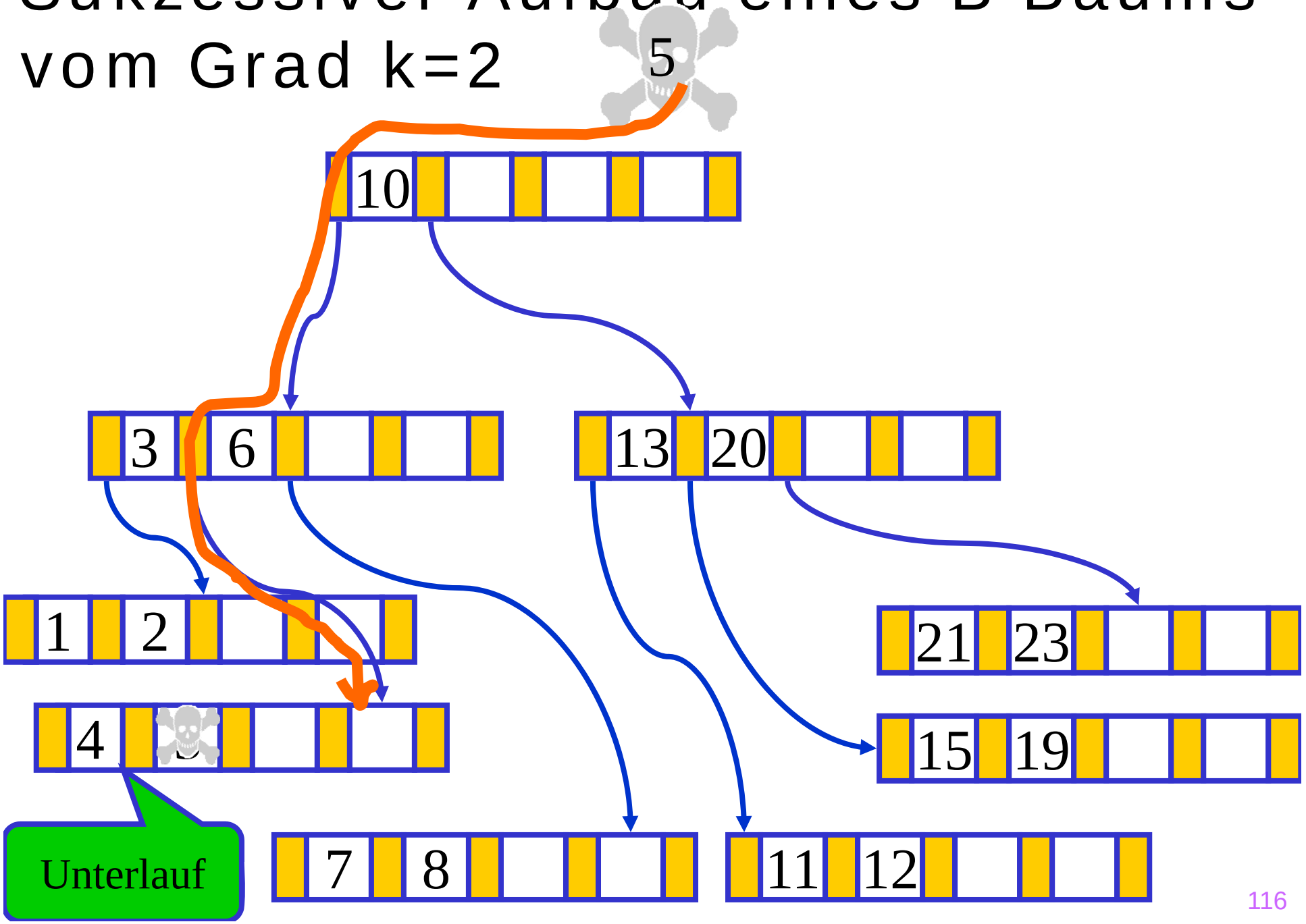
Sukzessiver Aufbau eines B-Baums vom Grad k=2



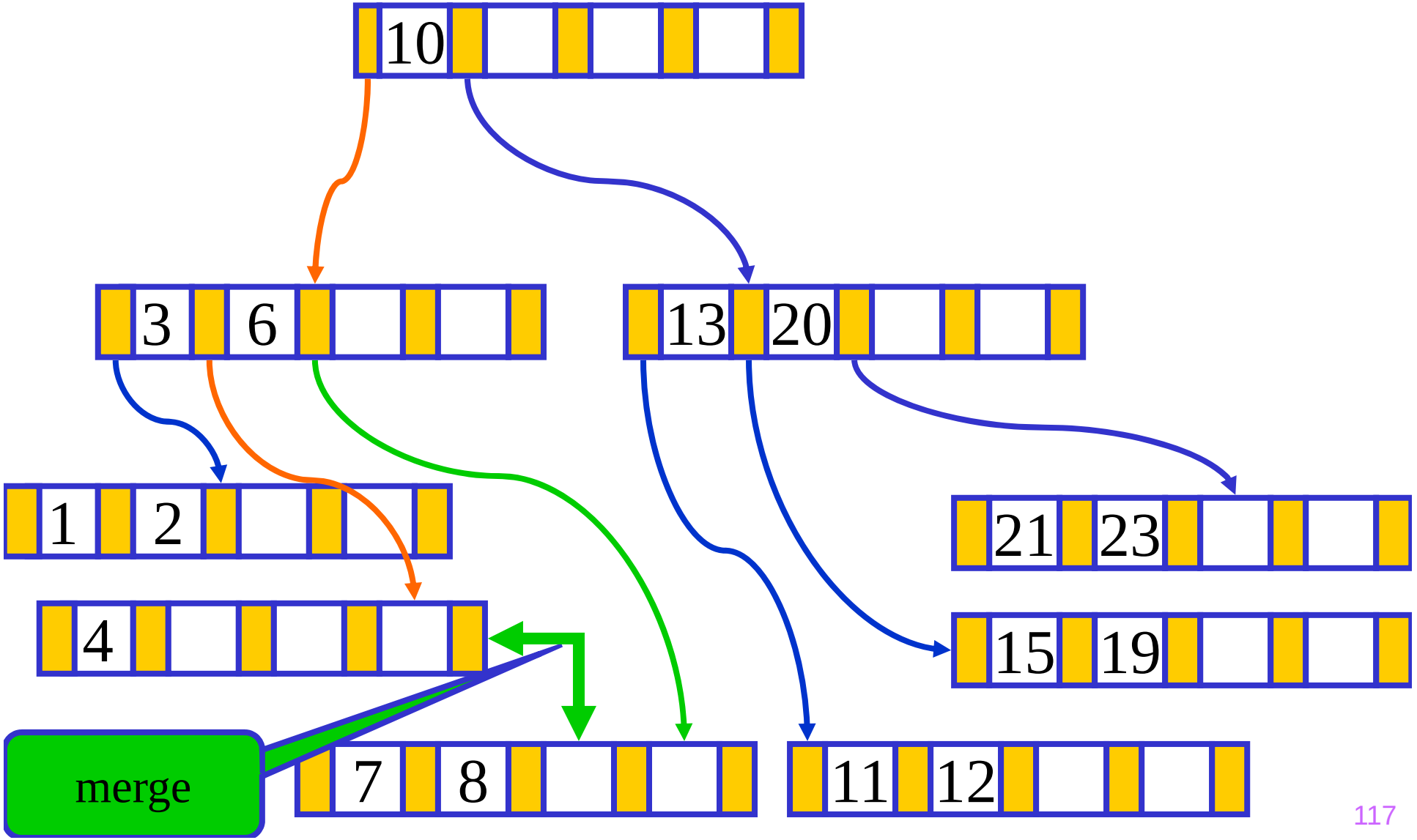
Sukzessiver Aufbau eines B-Baums vom Grad k=2



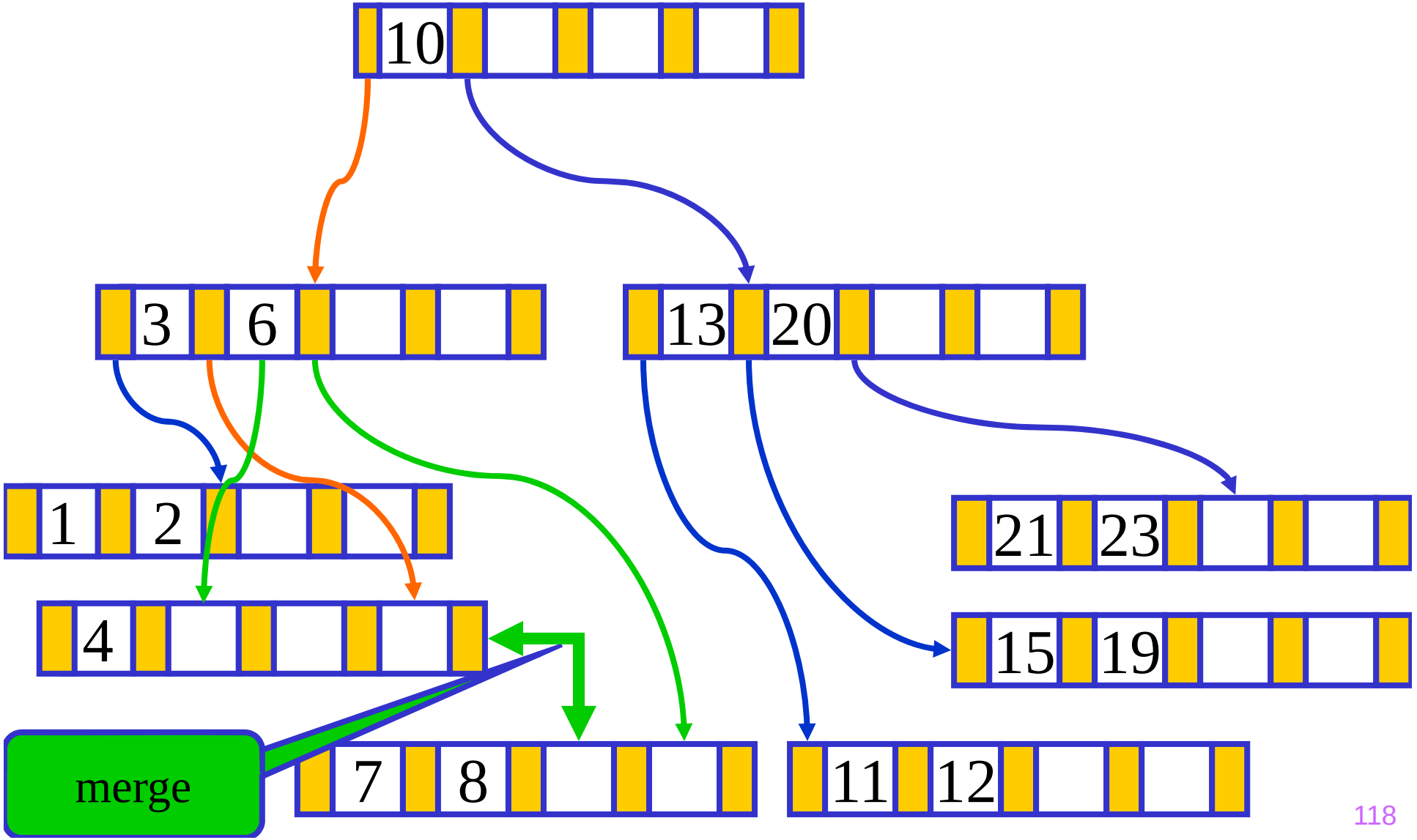
Sukzessiver Aufbau eines B-Baums vom Grad k=2



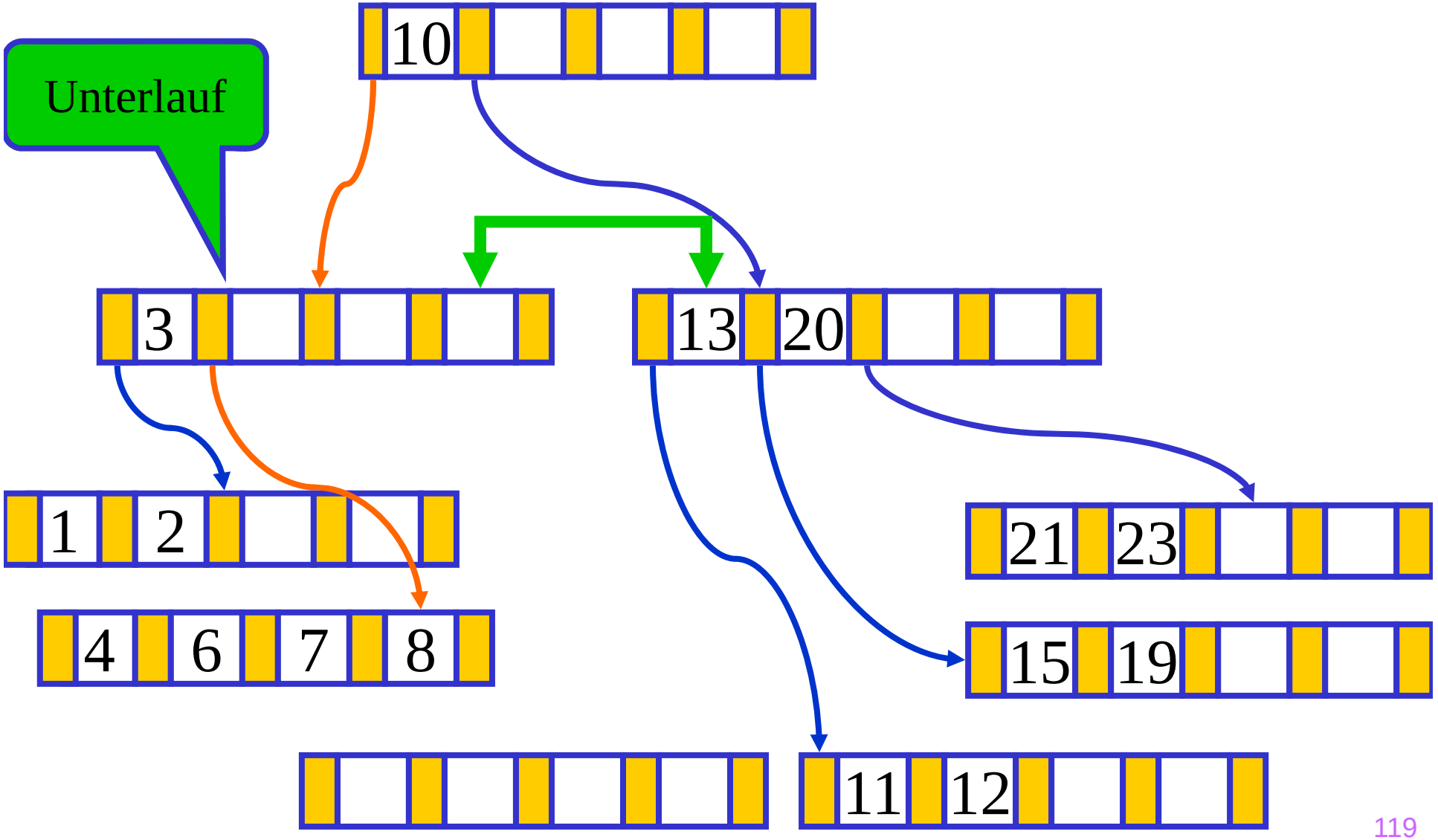
Sukzessiver Aufbau eines B-Baums vom Grad k=2



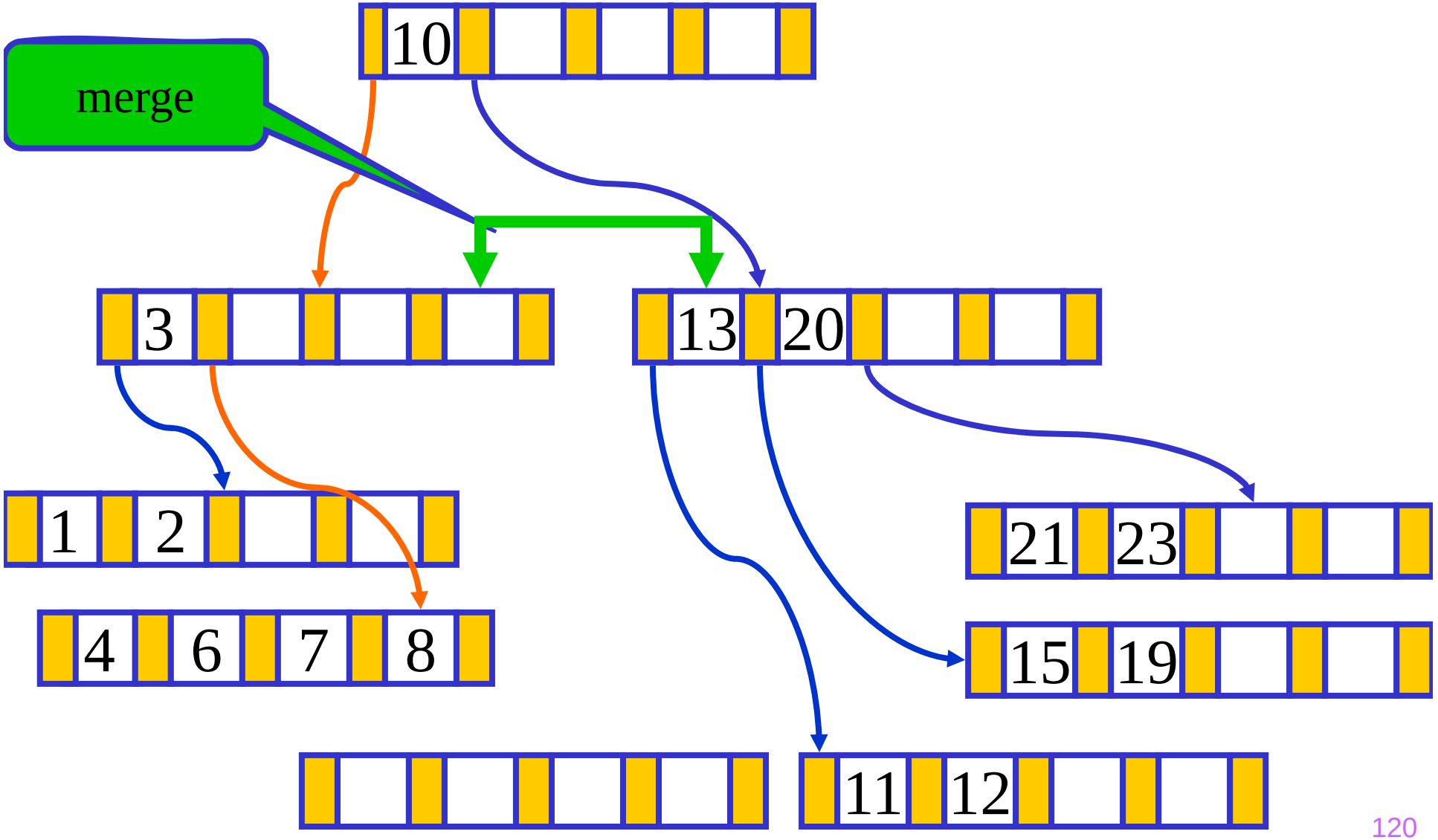
Sukzessiver Aufbau eines B-Baums vom Grad k=2



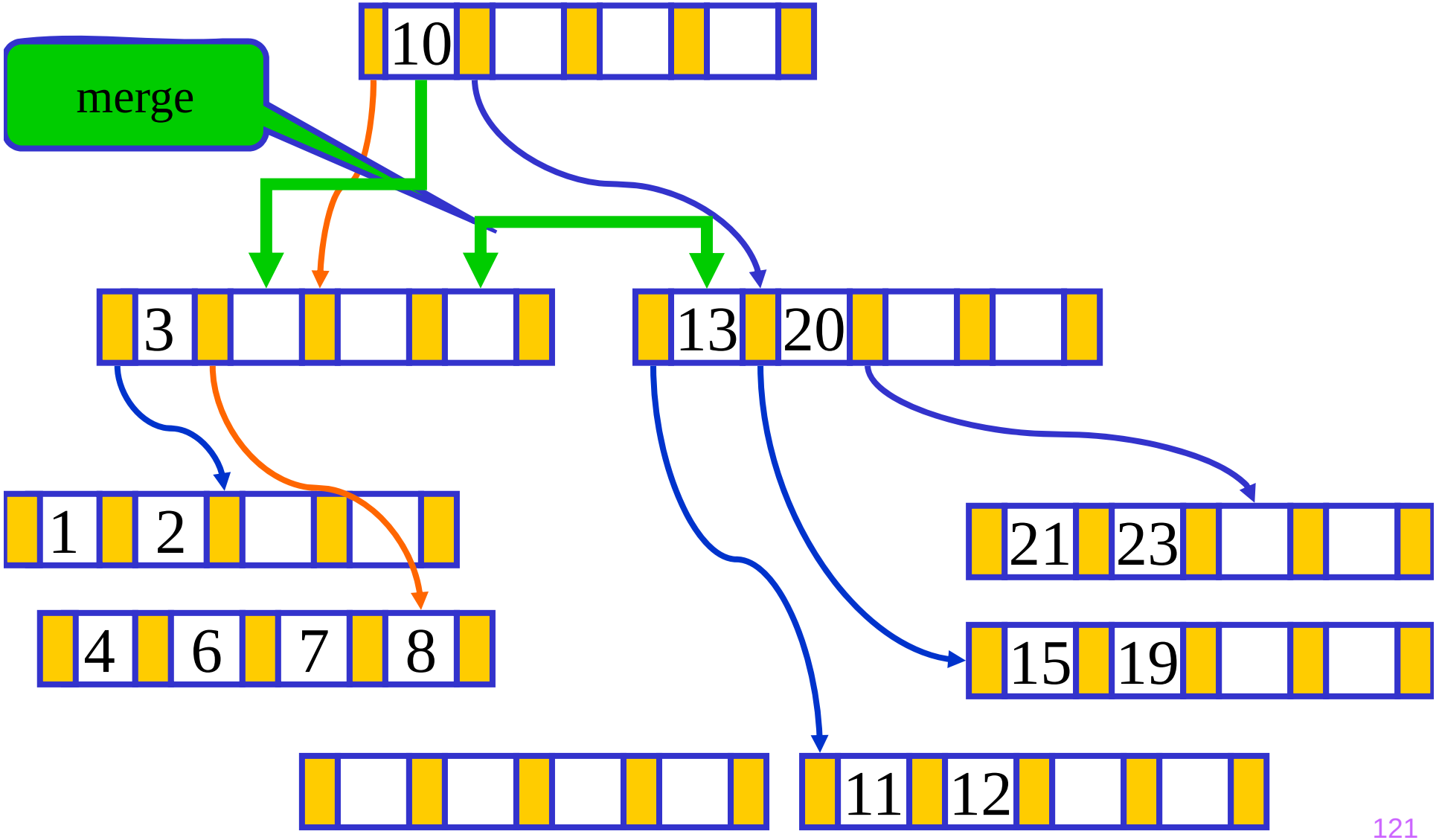
Sukzessiver Aufbau eines B-Baums vom Grad k=2



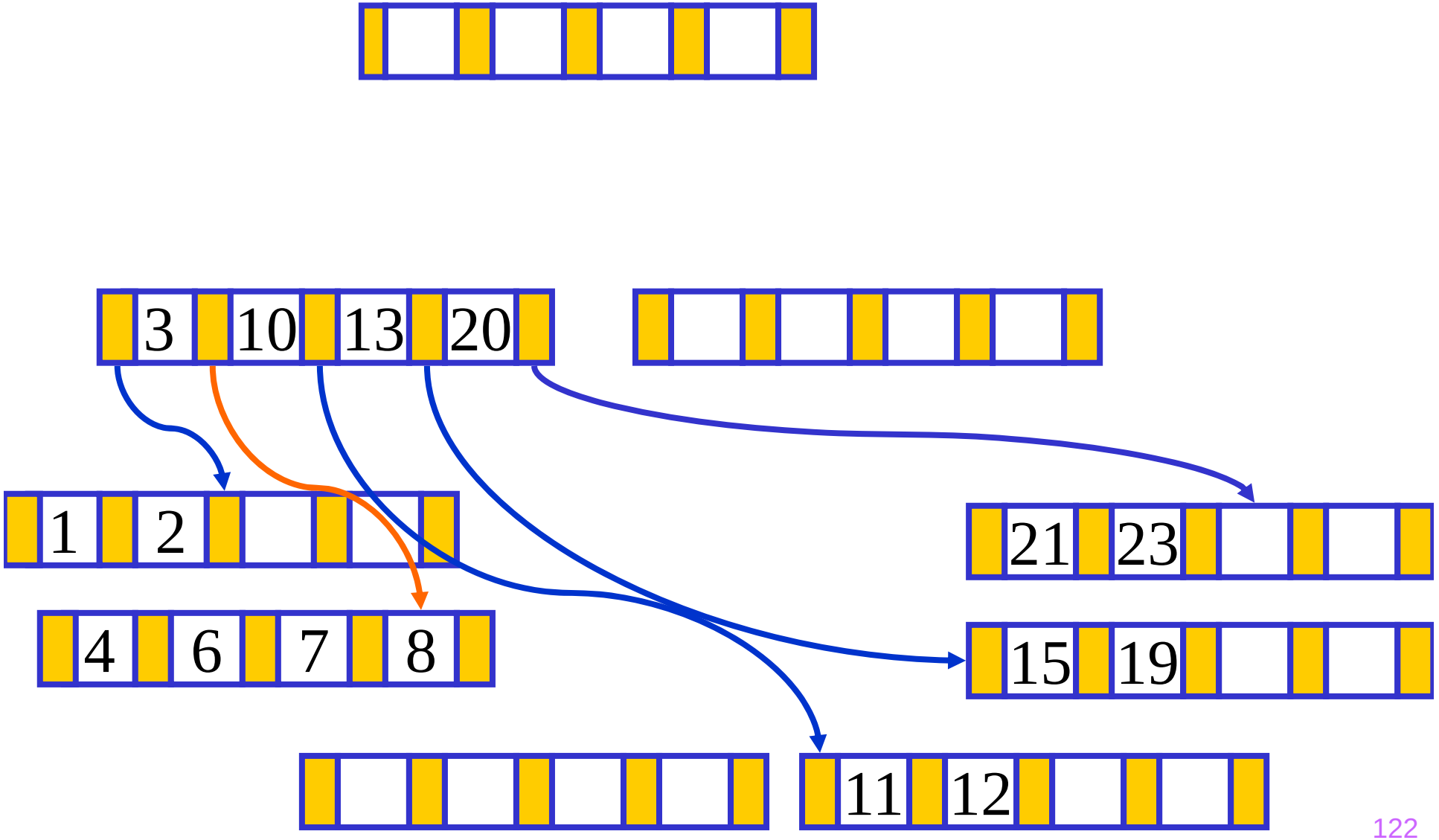
Sukzessiver Aufbau eines B-Baums vom Grad k=2



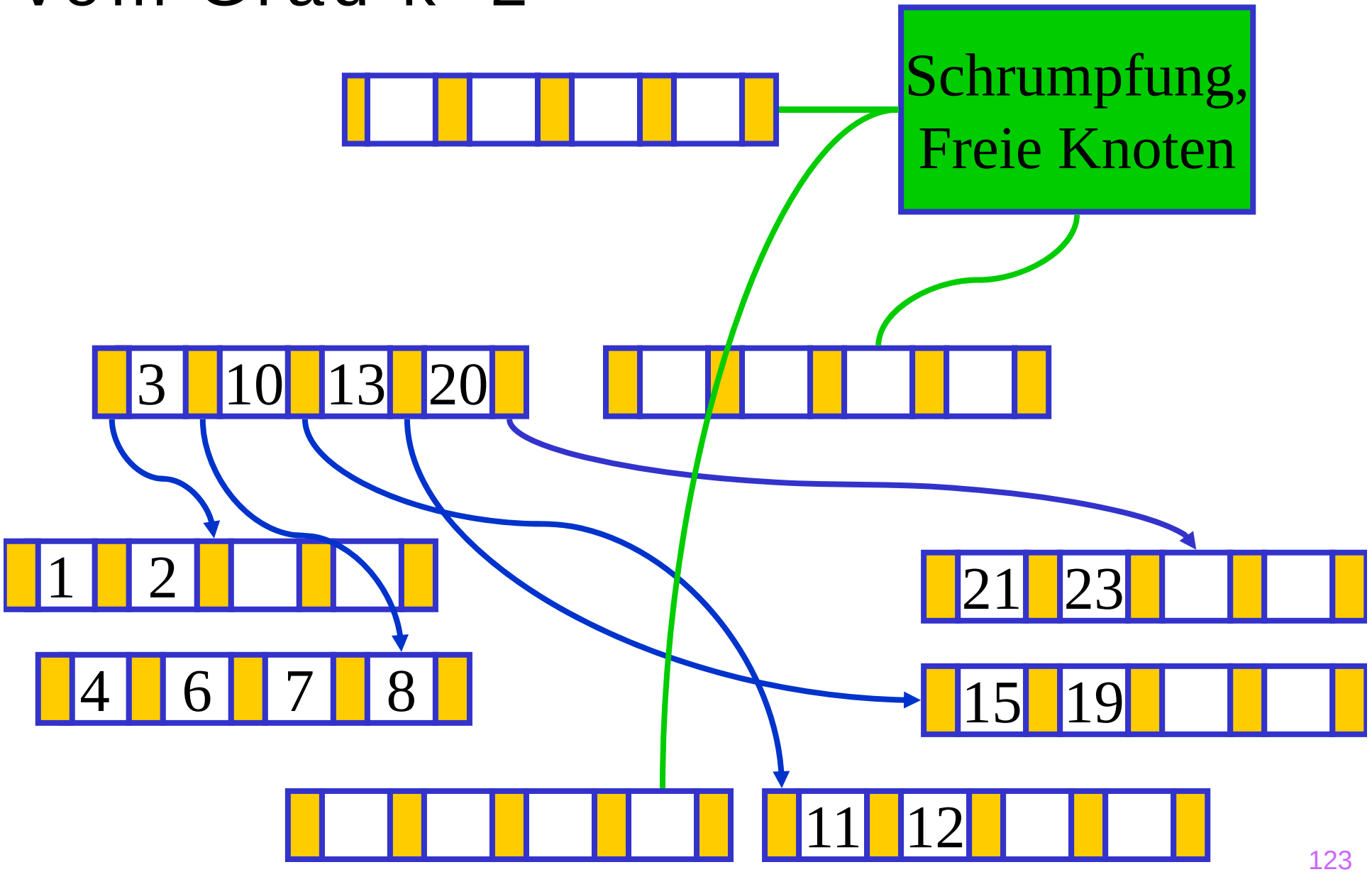
Sukzessiver Aufbau eines B-Baums vom Grad k=2



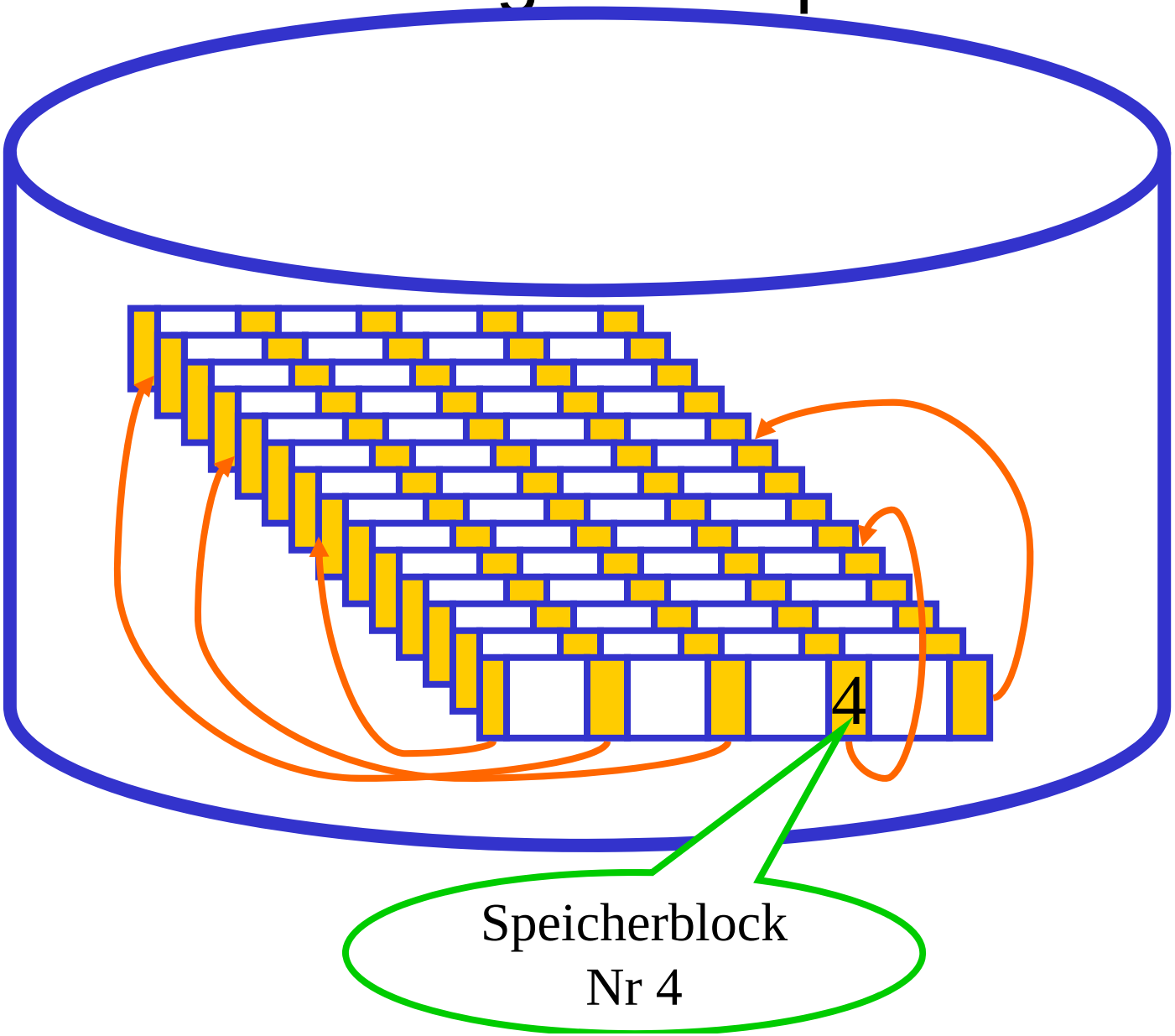
Sukzessiver Aufbau eines B-Baums vom Grad k=2



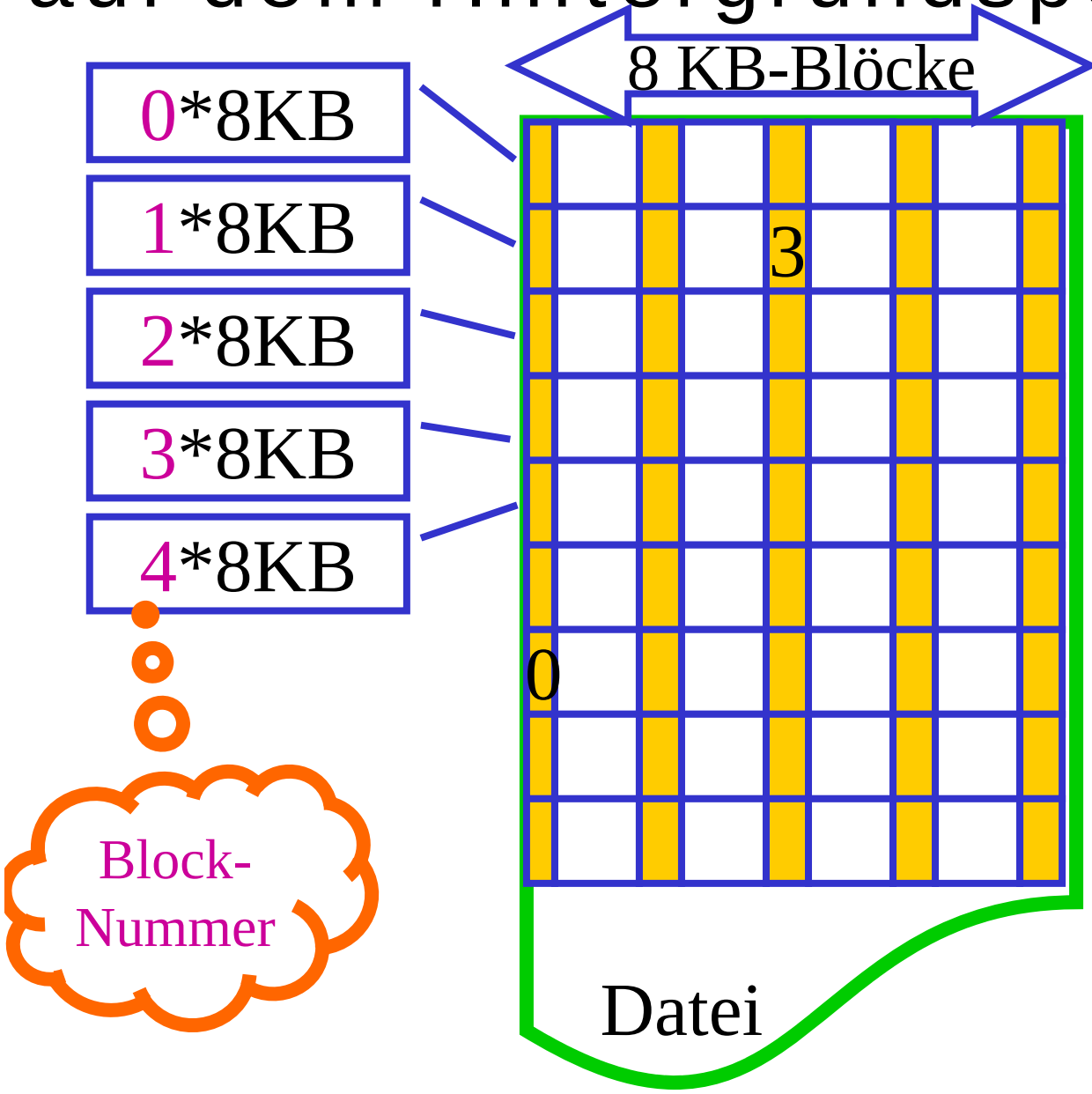
Sukzessiver Aufbau eines B-Baums vom Grad k=2



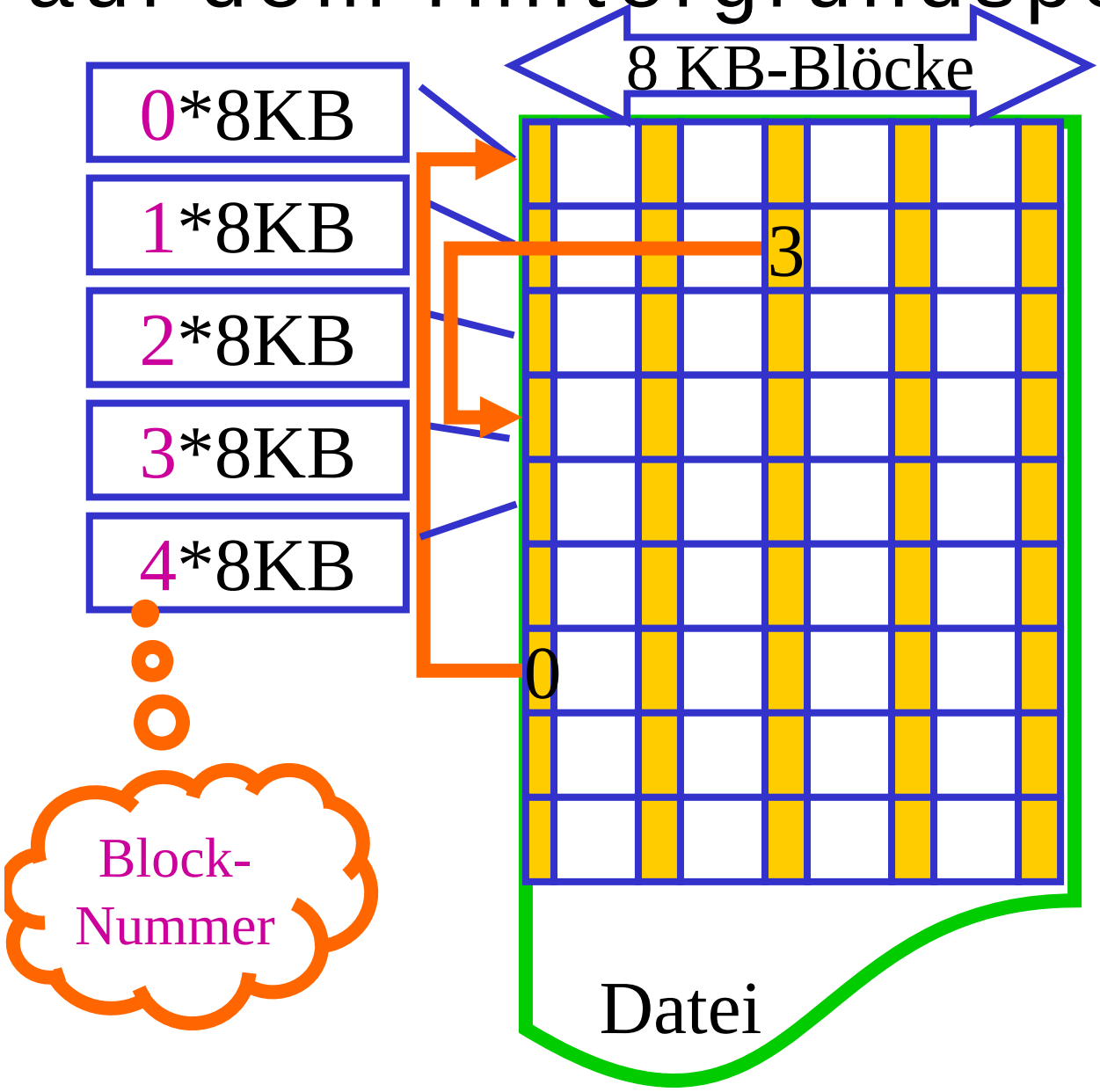
Speicherstruktur eines B-Baums auf dem Hintergrundspeicher



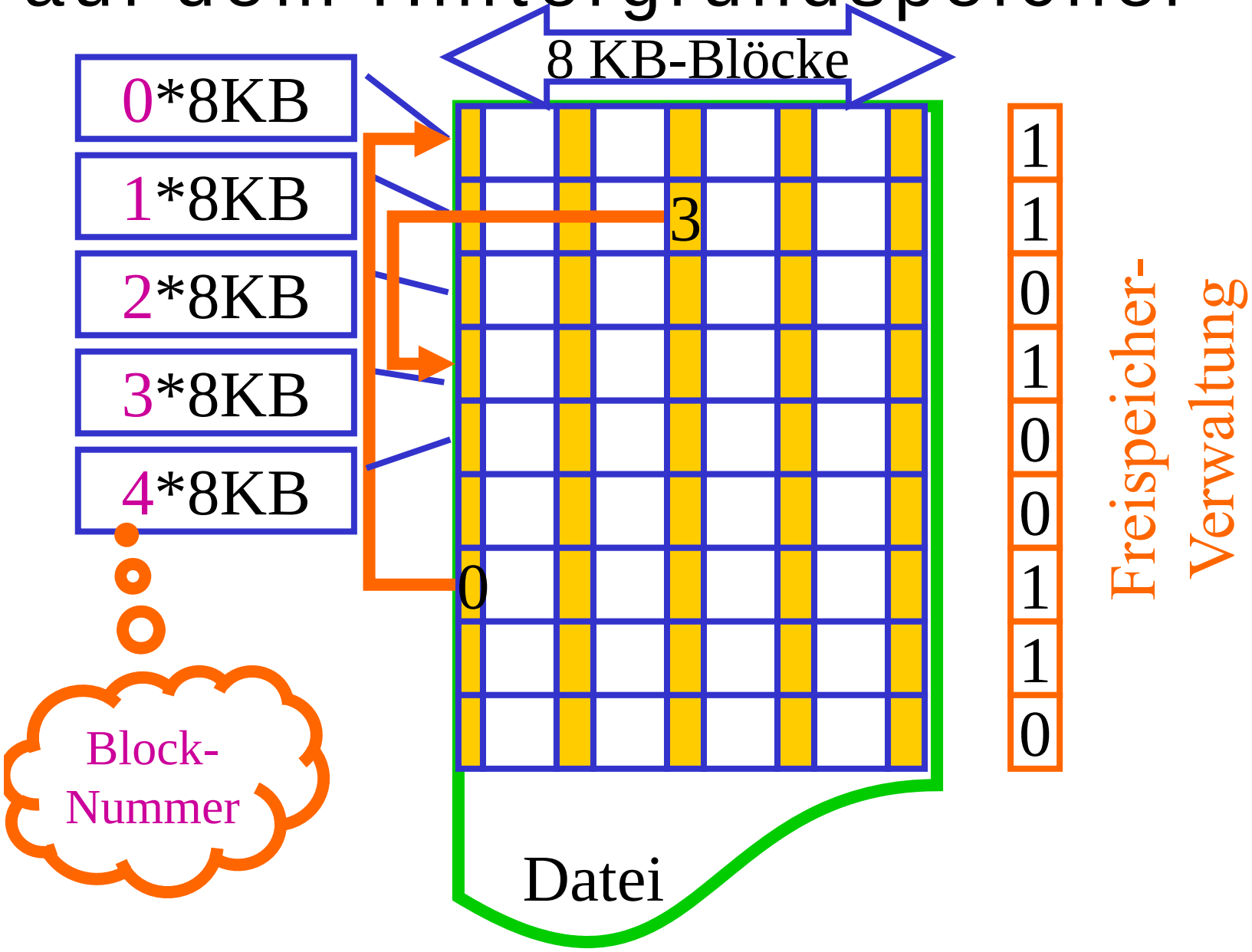
Speicherstruktur eines B-Baums auf dem Hintergrundspeicher



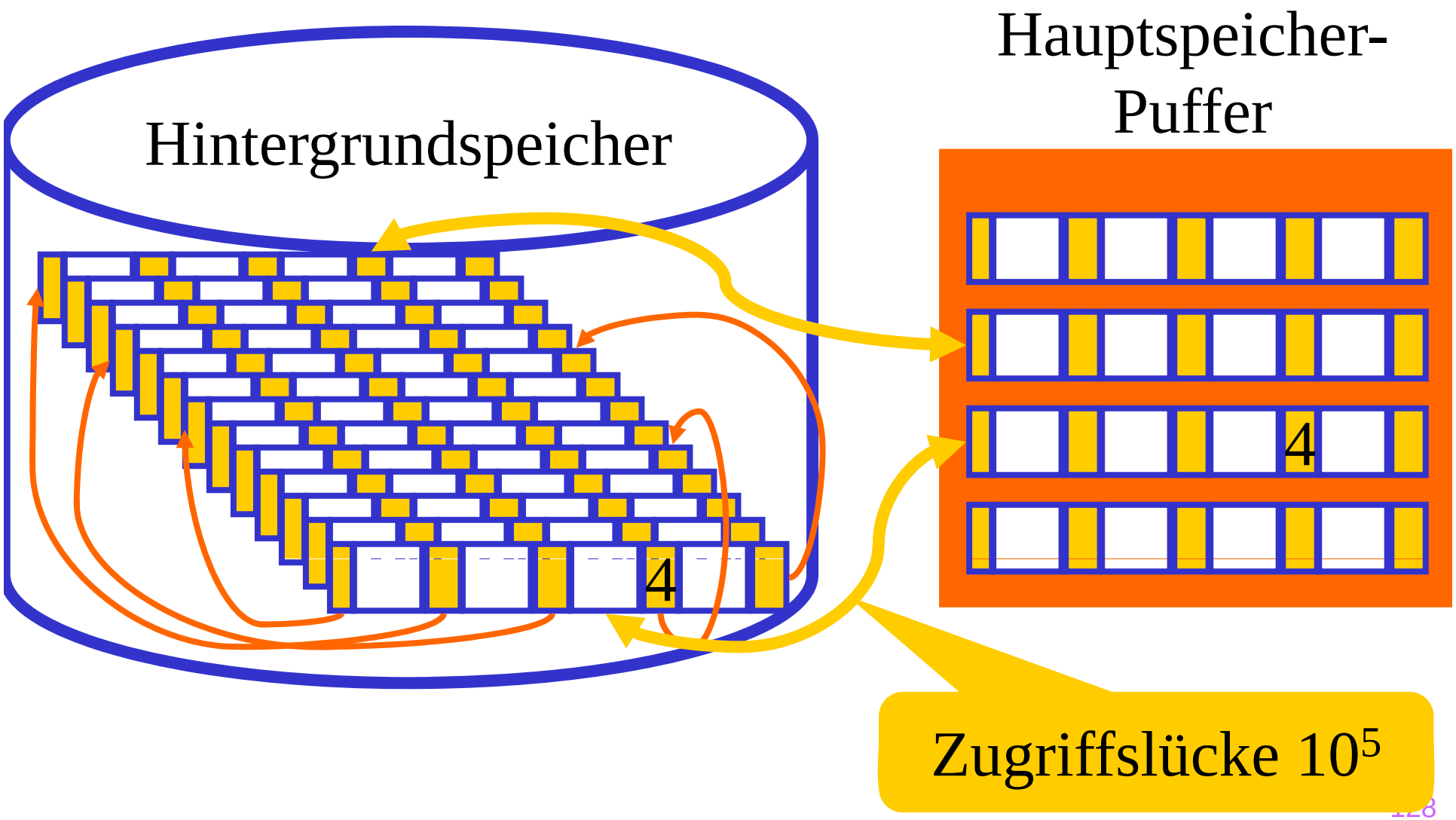
Speicherstruktur eines B-Baums auf dem Hintergrundspeicher



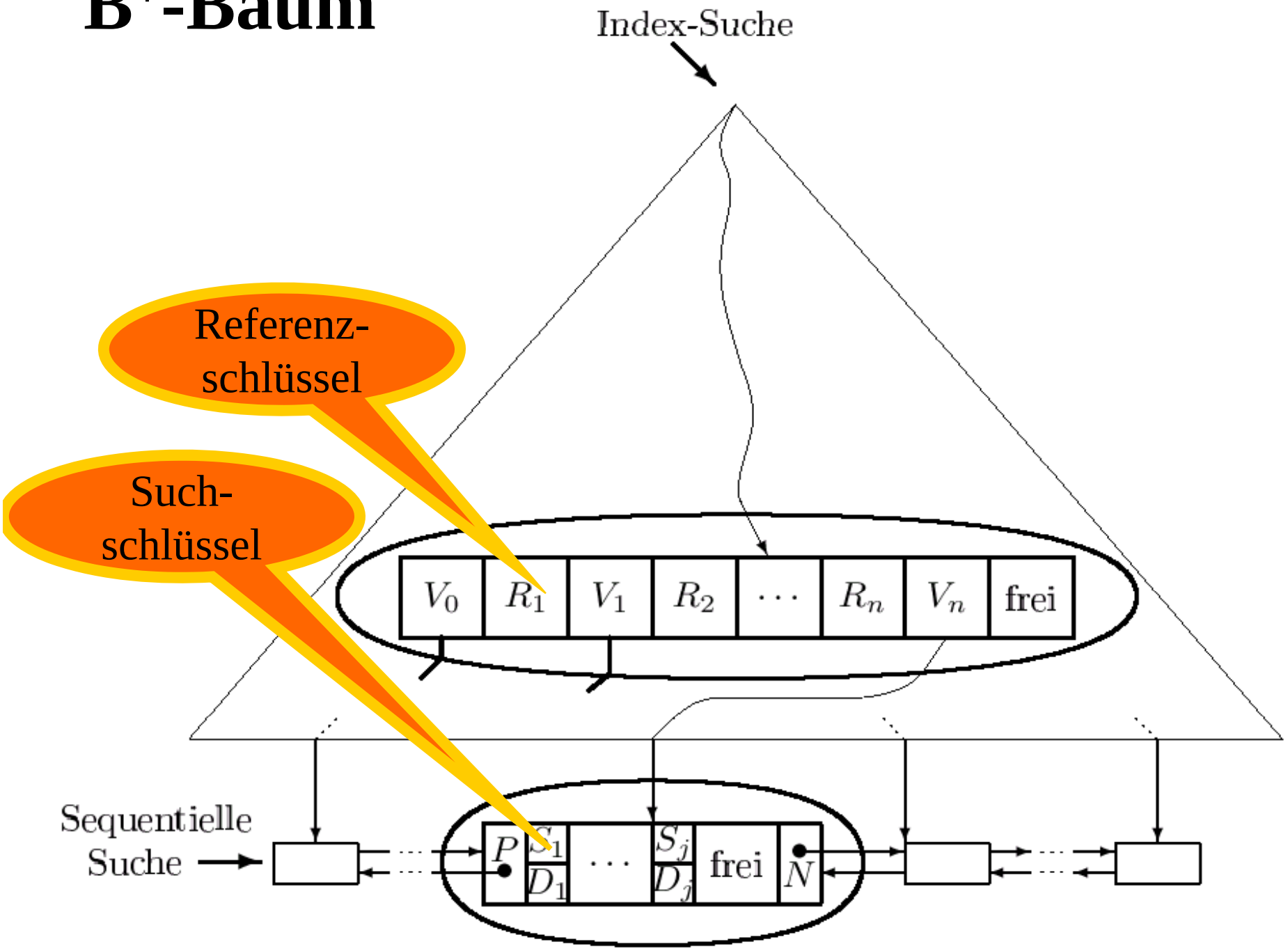
Speicherstruktur eines B-Baums auf dem Hintergrundspeicher



Zusammenspiel:
Hintergrundspeicher -- Hauptspeicher



B$^+$-Baum

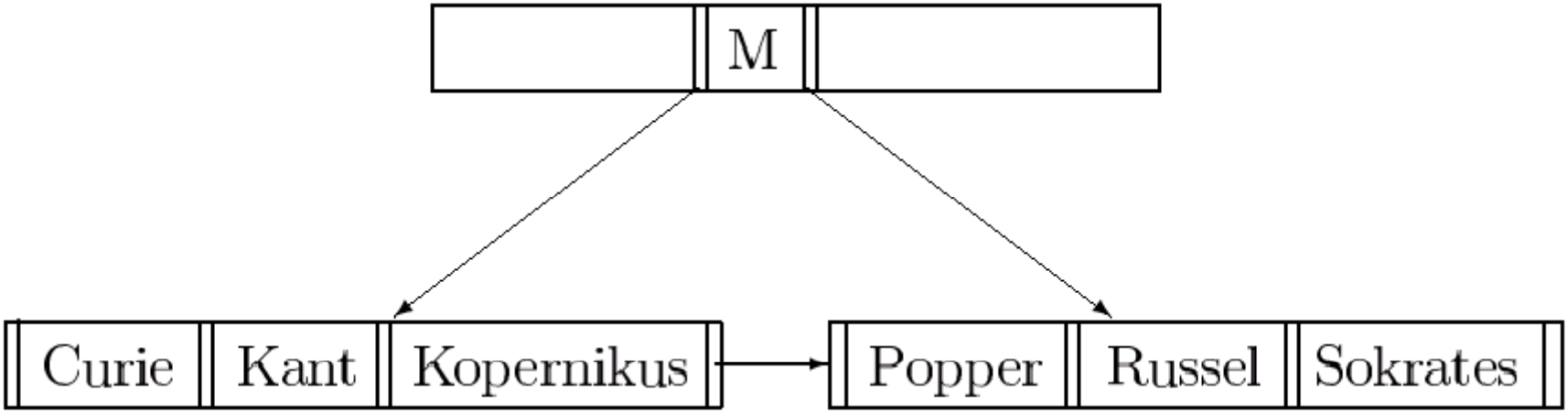

Index-Suche
Referenz-
schlüssel
Such-
schlüssel
V_0 R_1 V_1 R_2 $\cdots$ R_n V_n frei
Sequentielle
Suche
P S_1 D_1 $\cdots$ S_j D_j frei N

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.

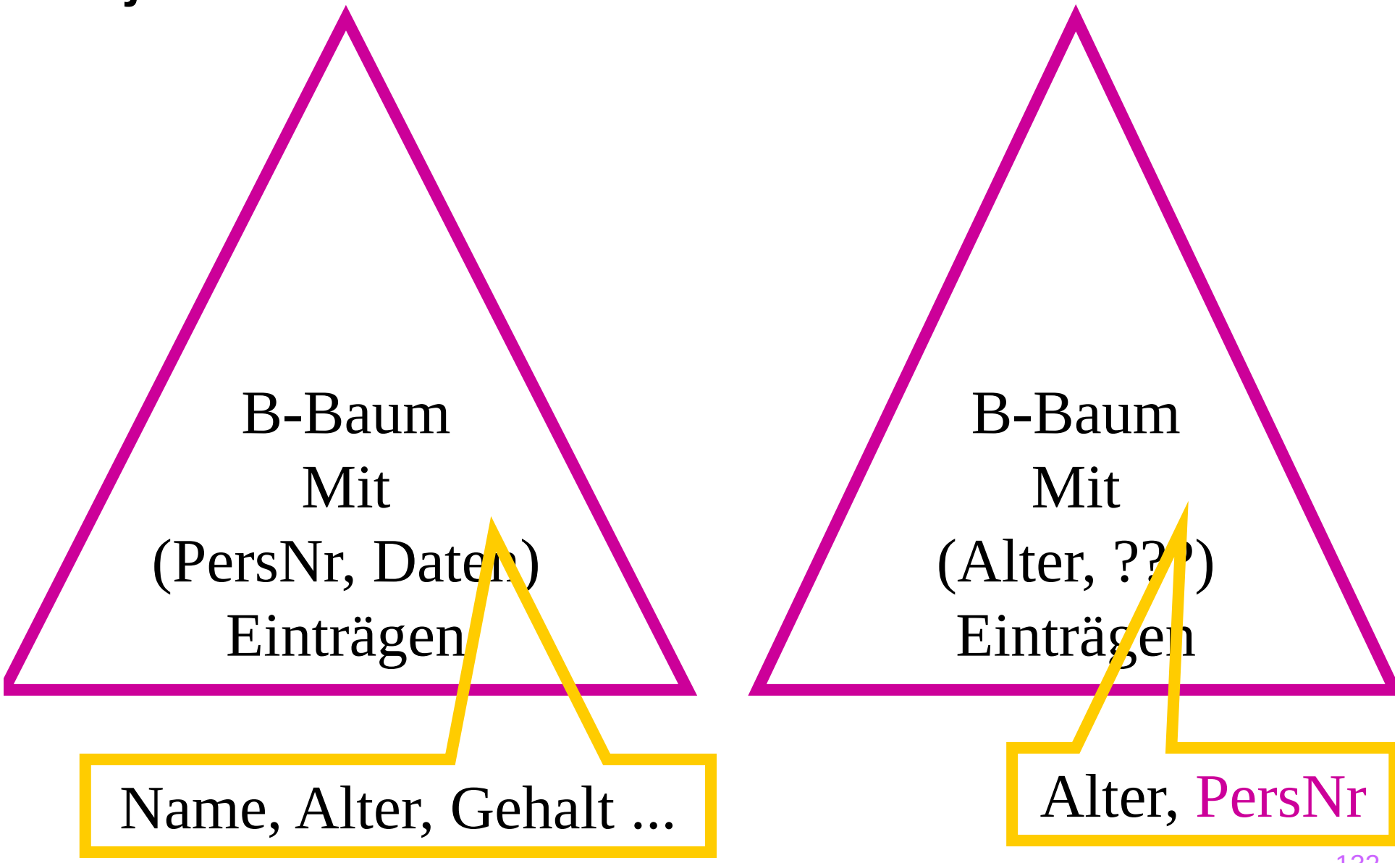
Präfix-B$^+$-Bäume

- Es werden nur *Referenz*schlüssel benötigt.

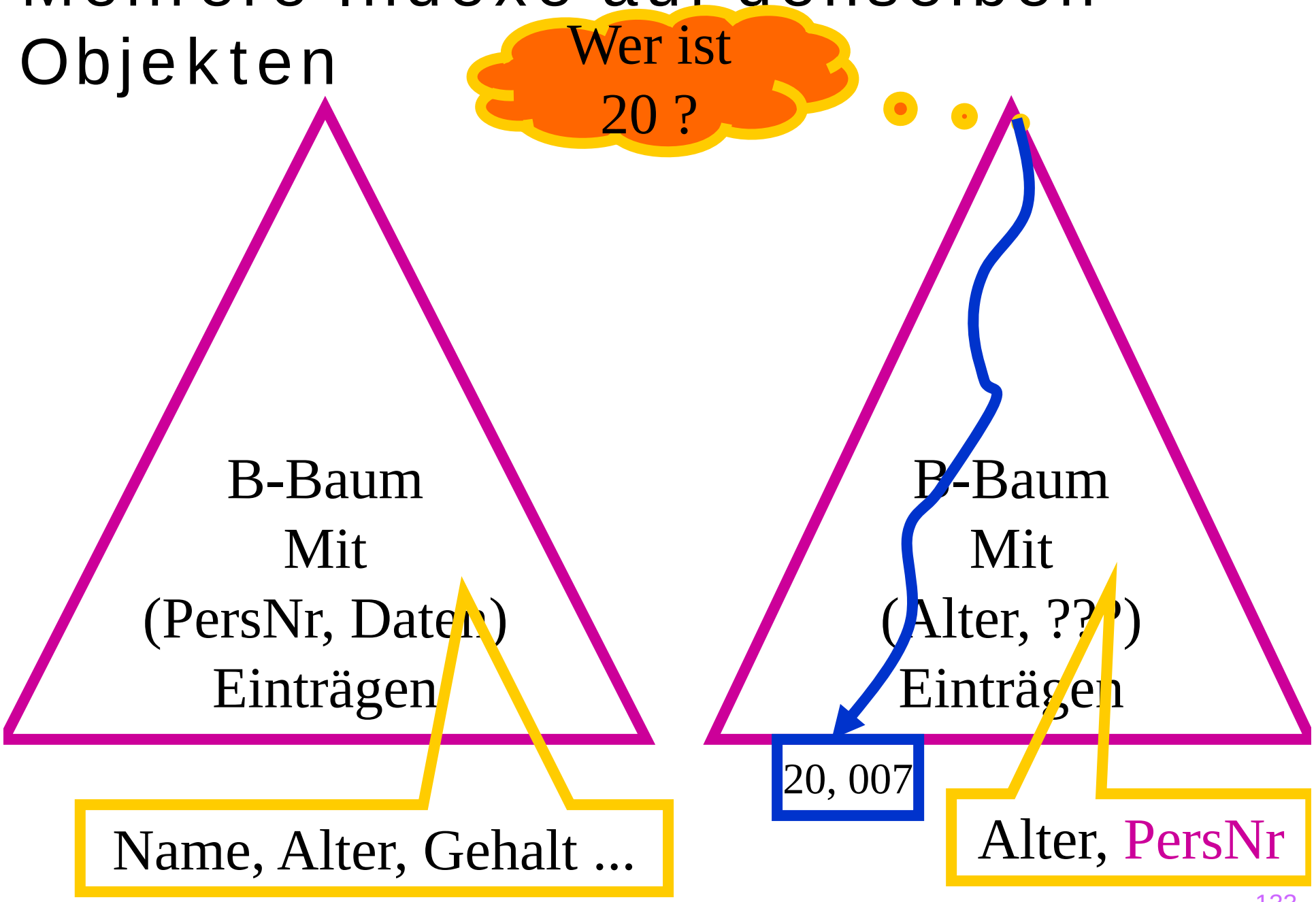


- beliebiger Referenzschlüssel R mit
  Kopernikus $\leq R <$ Popper

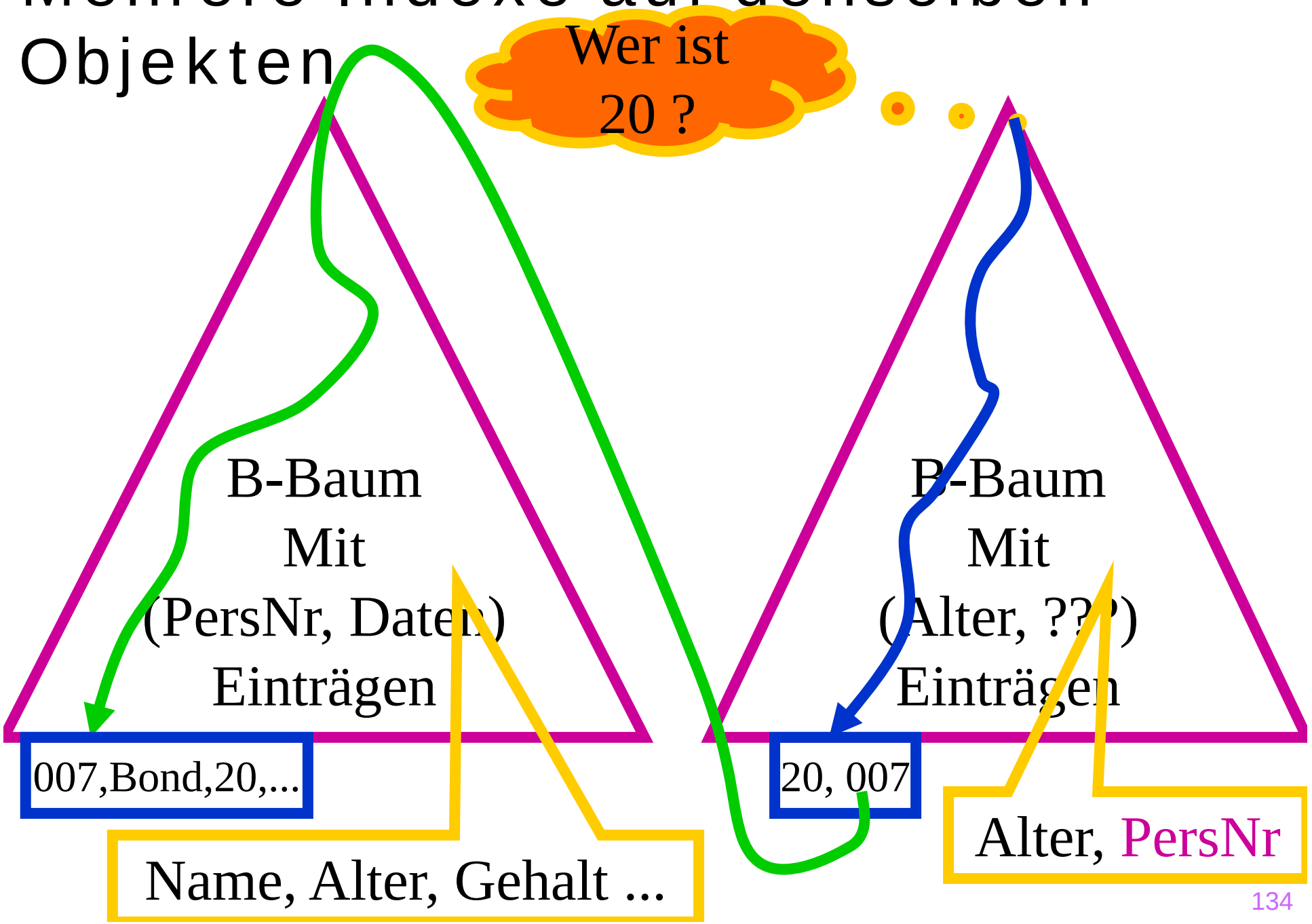
Mehrere Indexe auf denselben Objekten



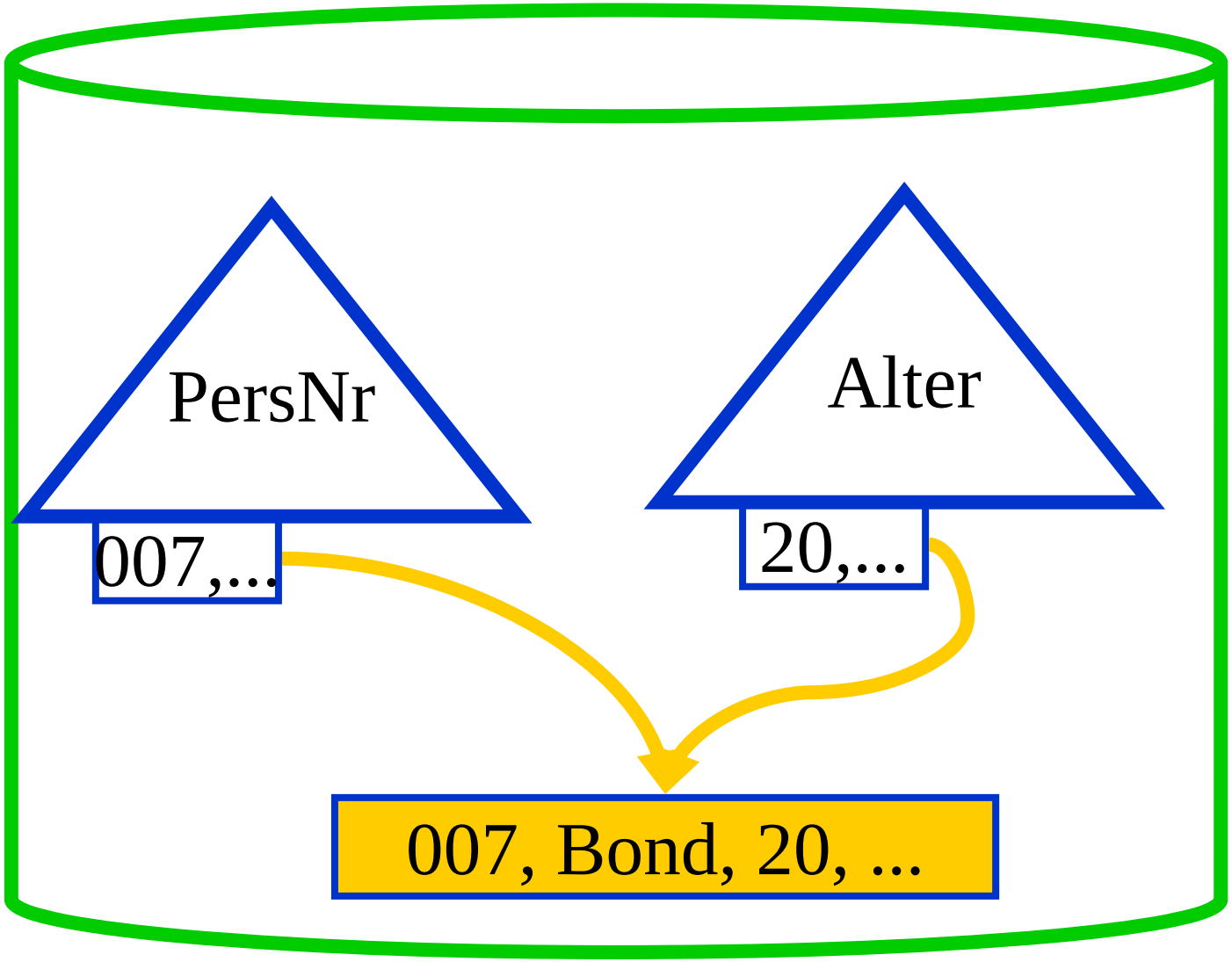
Mehrere Indexe auf denselben Objekten



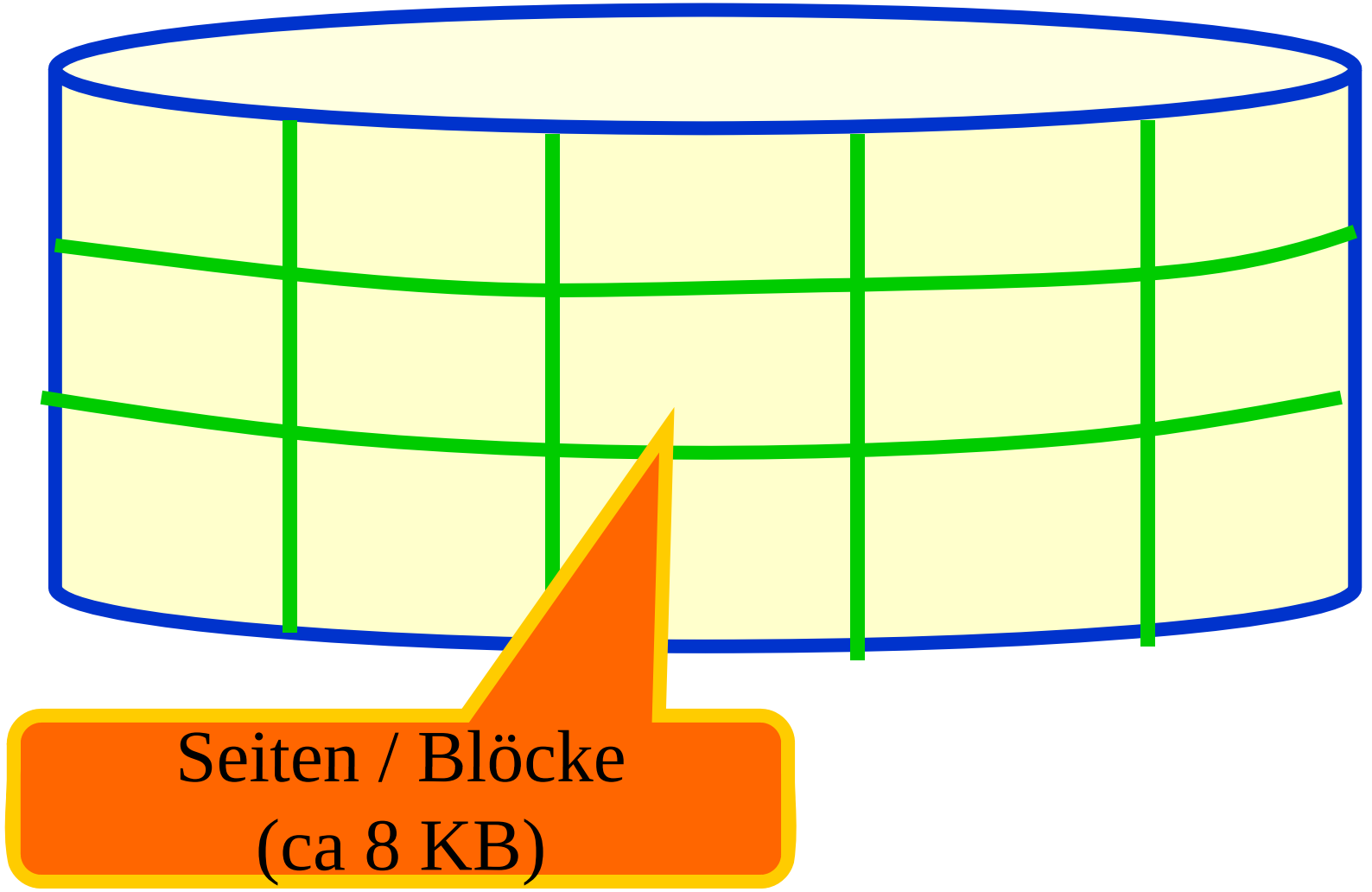
Mehrere Indexe auf denselben Objekten



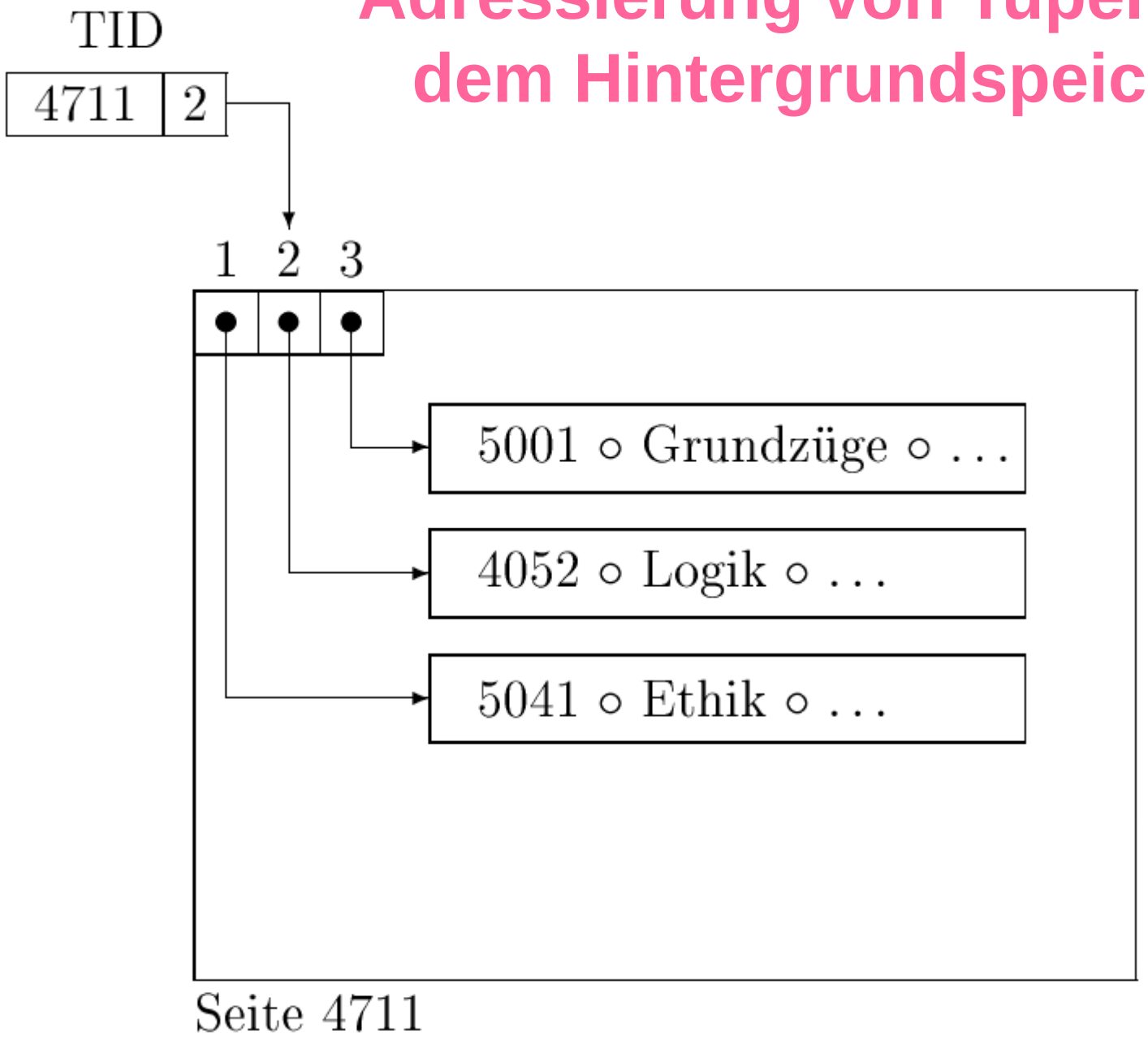
Eine andere Möglichkeit: Referenzierung über Speicheradressen



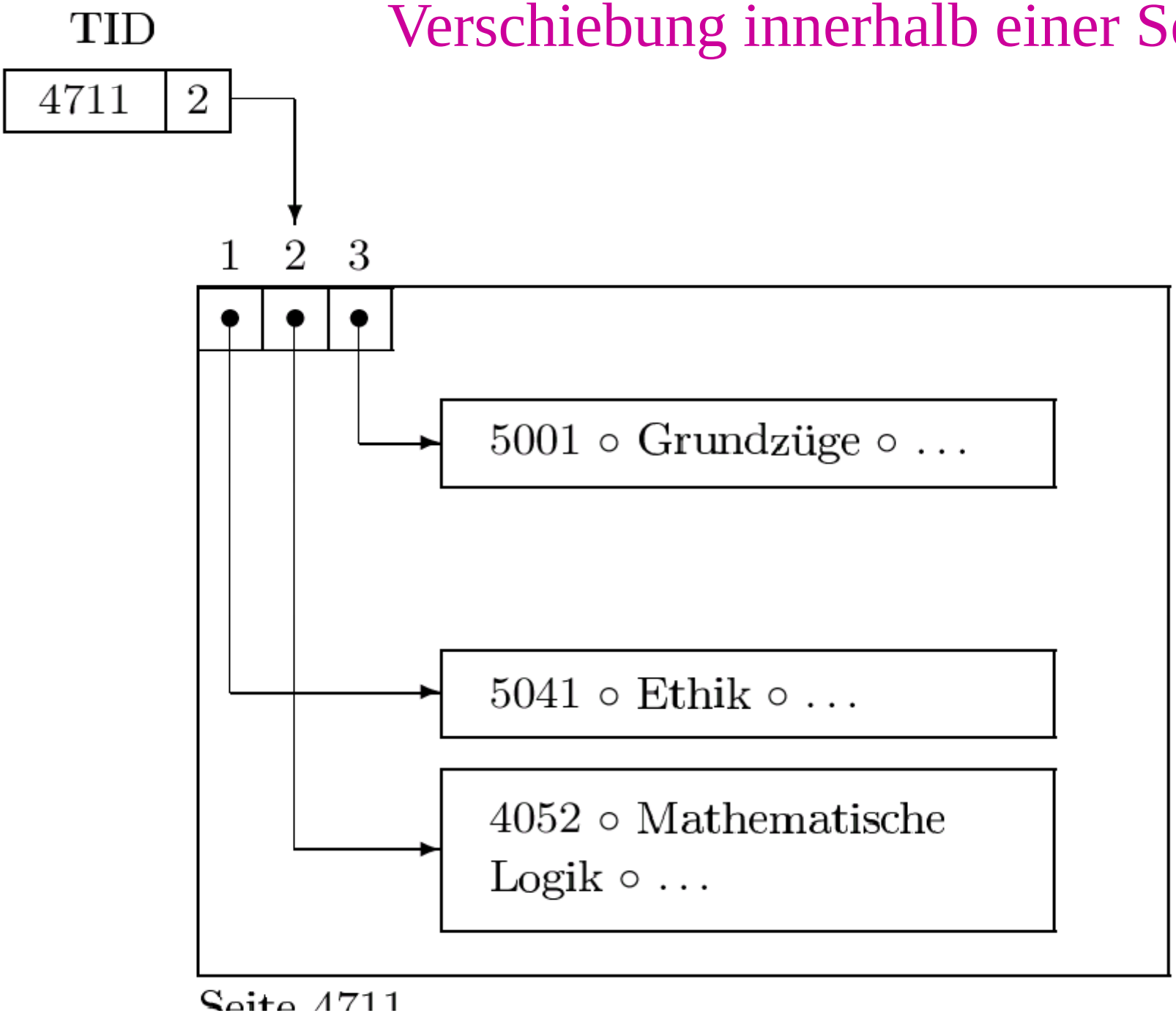
Realisierungstechnik für Hintergrundspeicher-Adressen



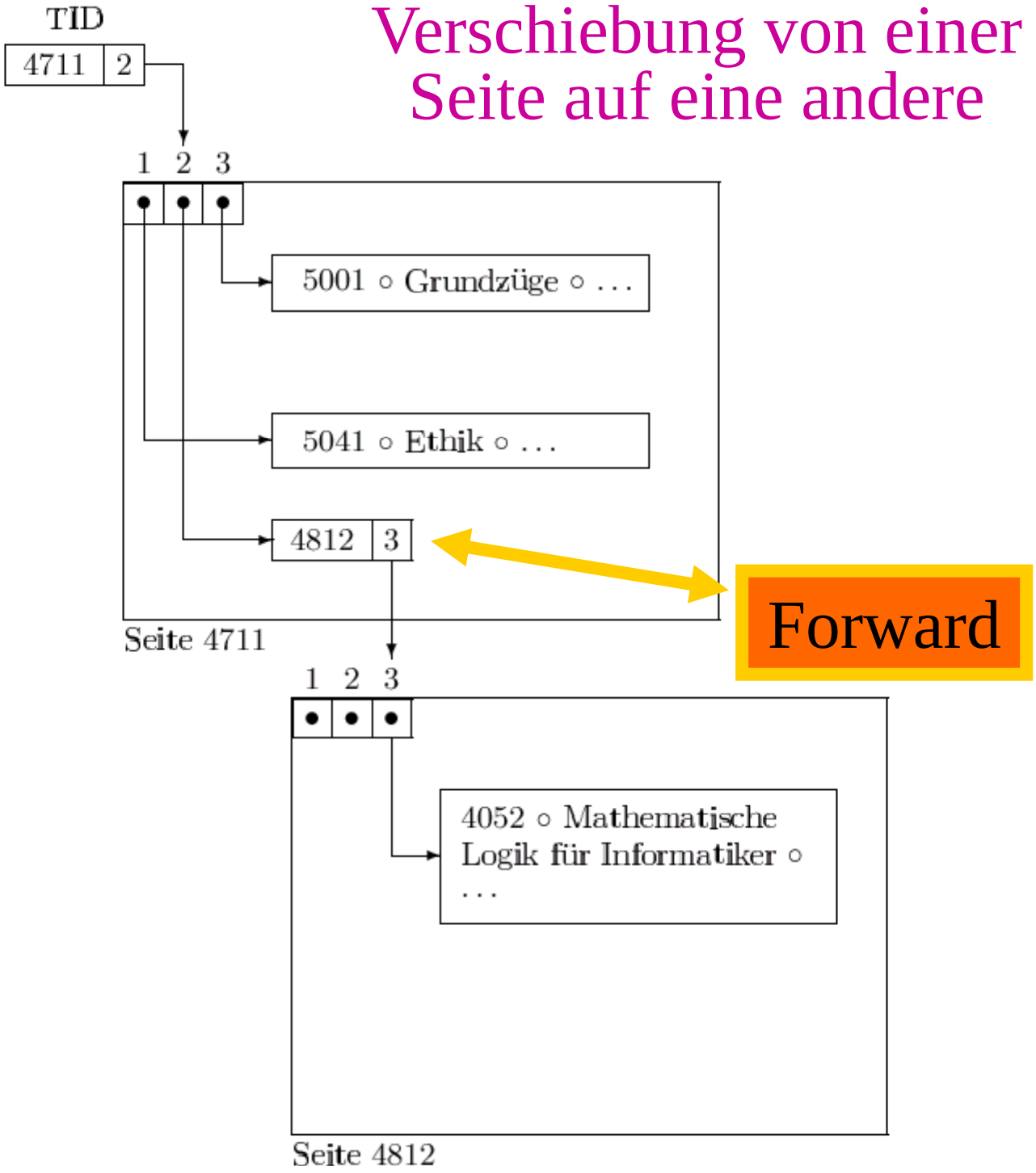
Adressierung von Tupeln auf dem Hintergrundspeicher



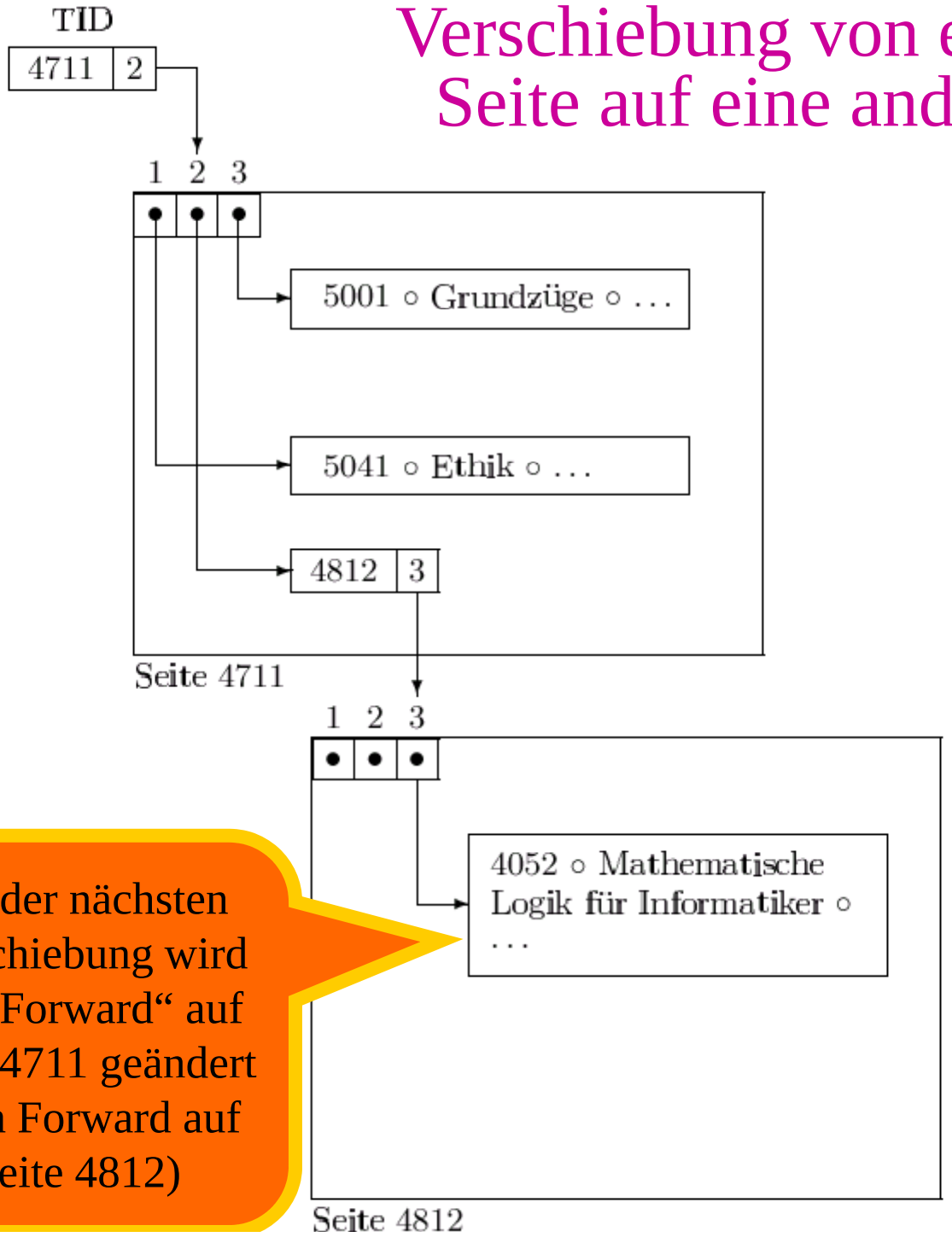
Verschiebung innerhalb einer Seite



Verschiebung von einer Seite auf eine andere



Verschiebung von einer Seite auf eine andere



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

„Statische“ Hashtabellen

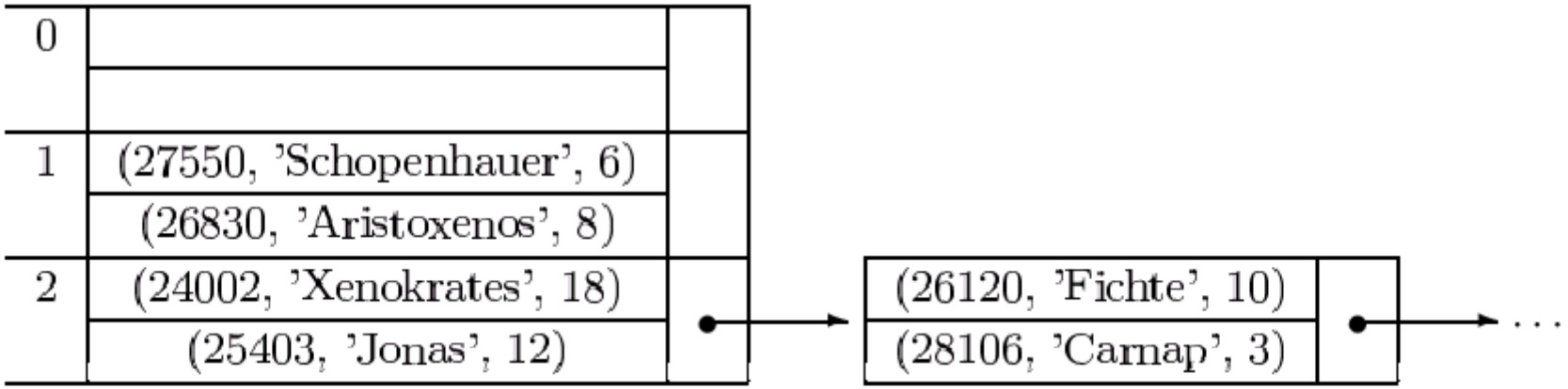
- À priori Allokation des Speichers
- Nachträgliche Vergrößerung der Hashtabelle ist „teuer“
  - Hashfunktion h(...) = ... mod N
  - Rehashing der Einträge
    - h(...) = ... mod M
  - In Datenbankanwendungen viele GB
- Erweiterbares Hashing
  - Zusätzliche Indirektion über ein Directory
  - Ein zusätzlicher 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

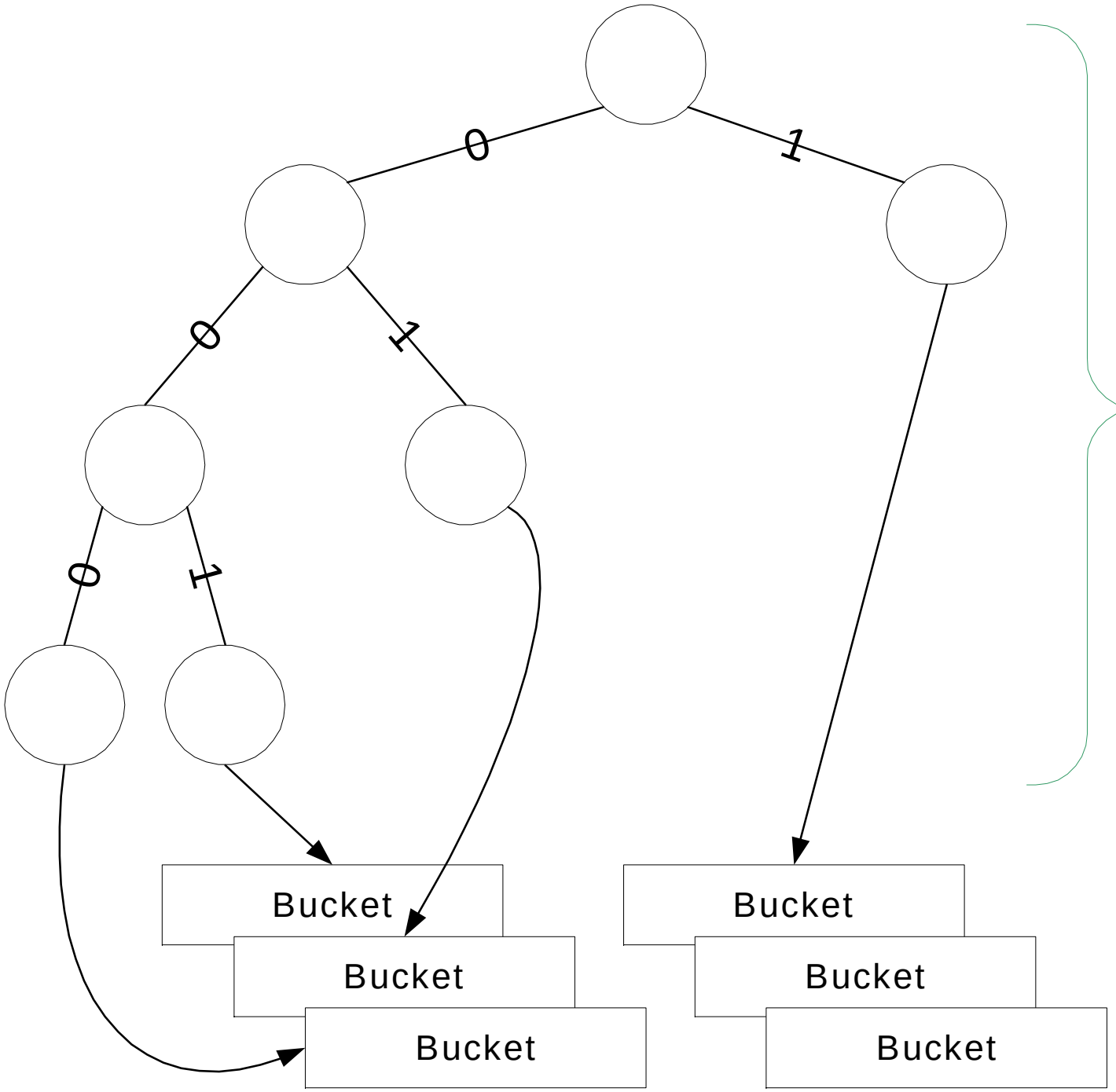
- Hashfunktion $h(x) = x \bmod 3$

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

- Kollisionsbehandlung



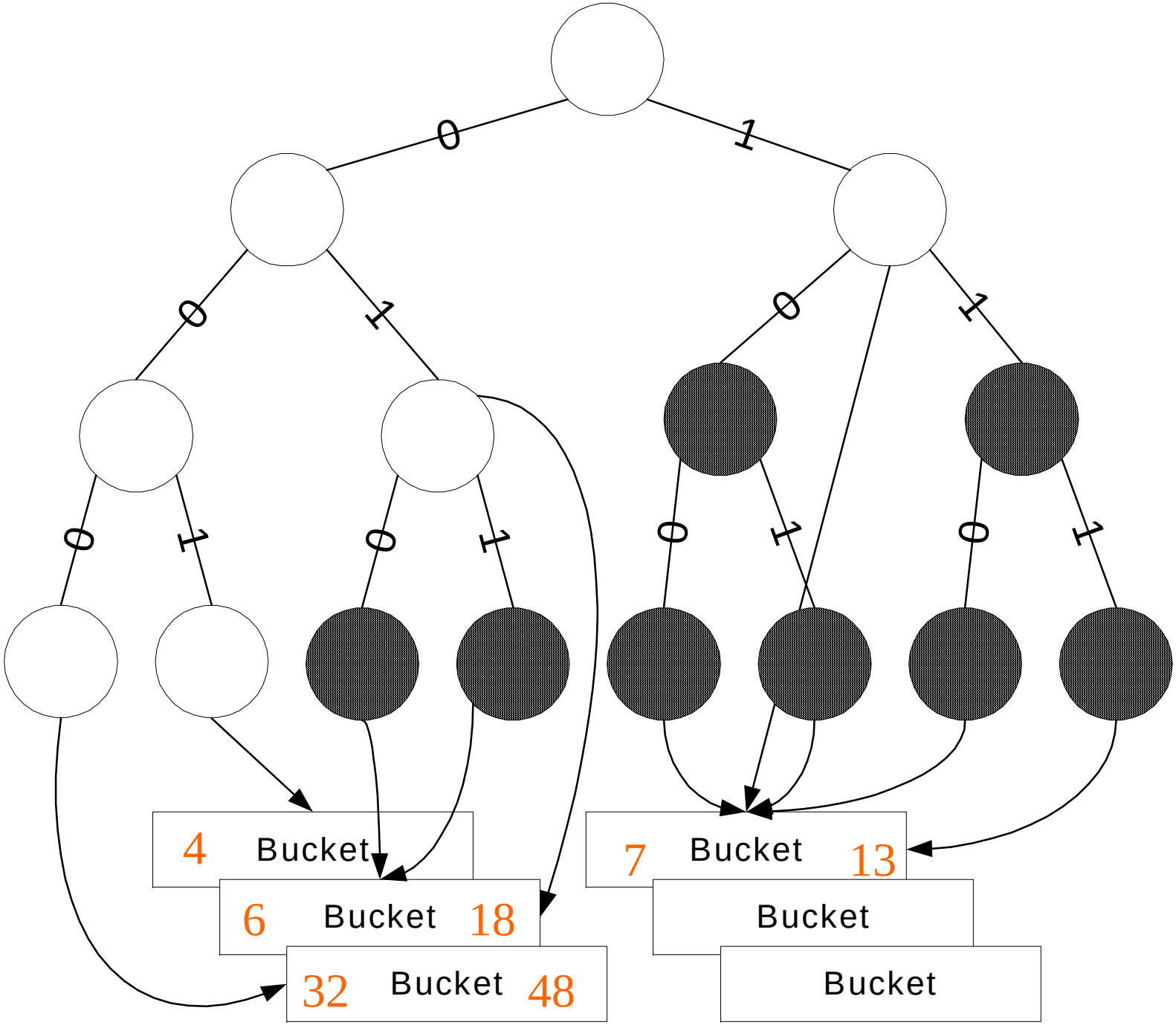
$\Longrightarrow$ ineffizient bei nicht vorhersehbarer Datenmenge

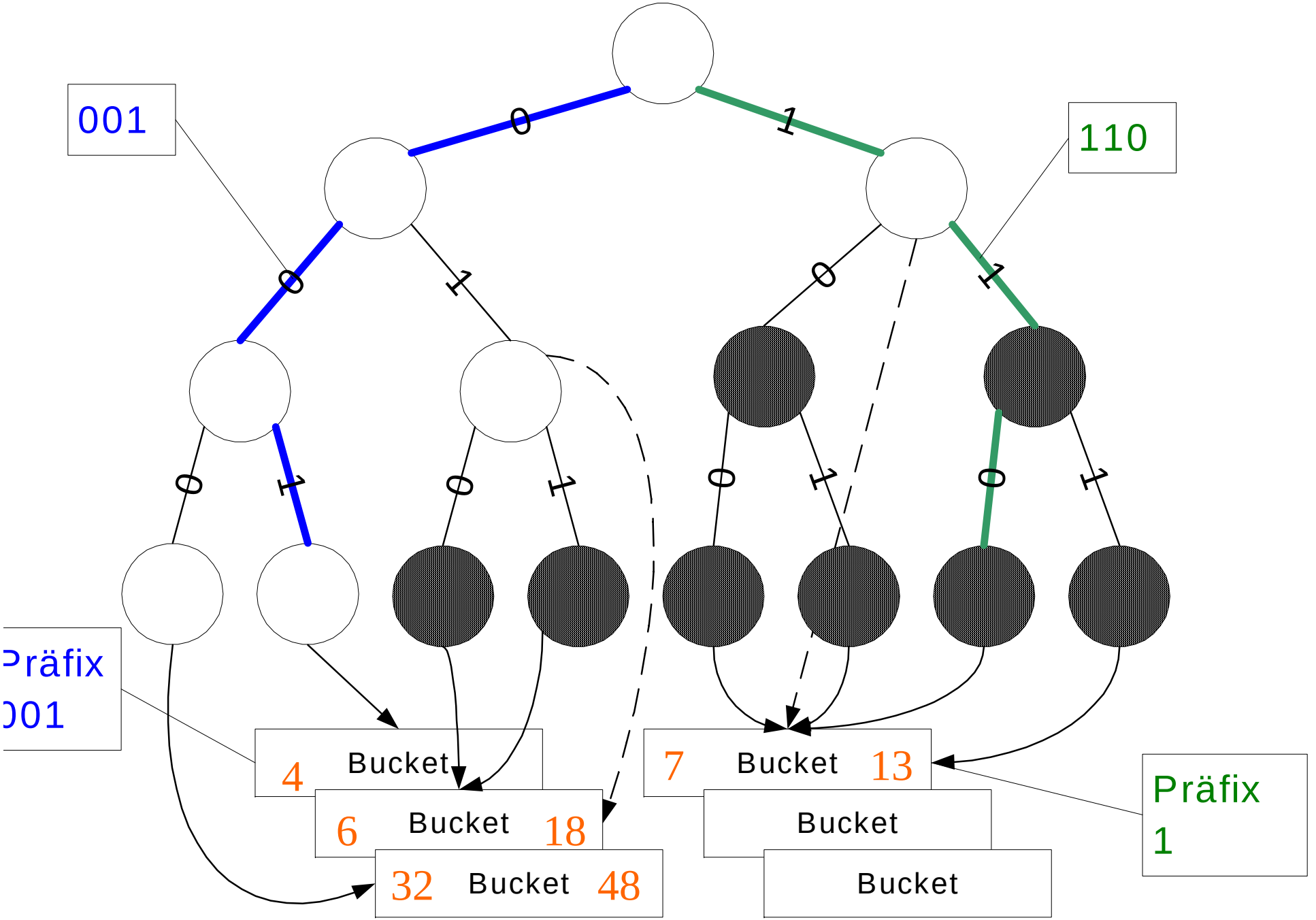


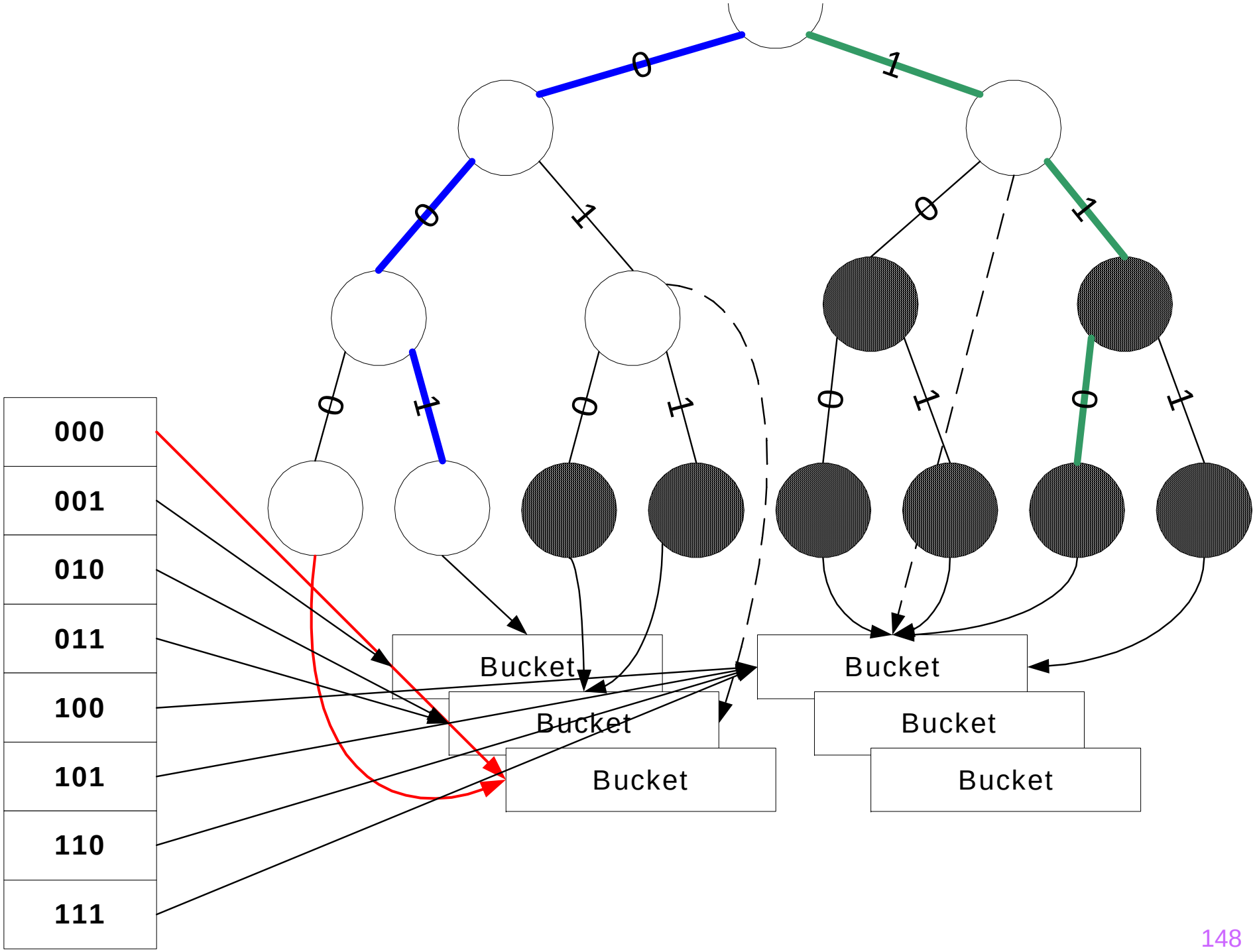
binärer Trie, Entschei-dungs-baum Directory

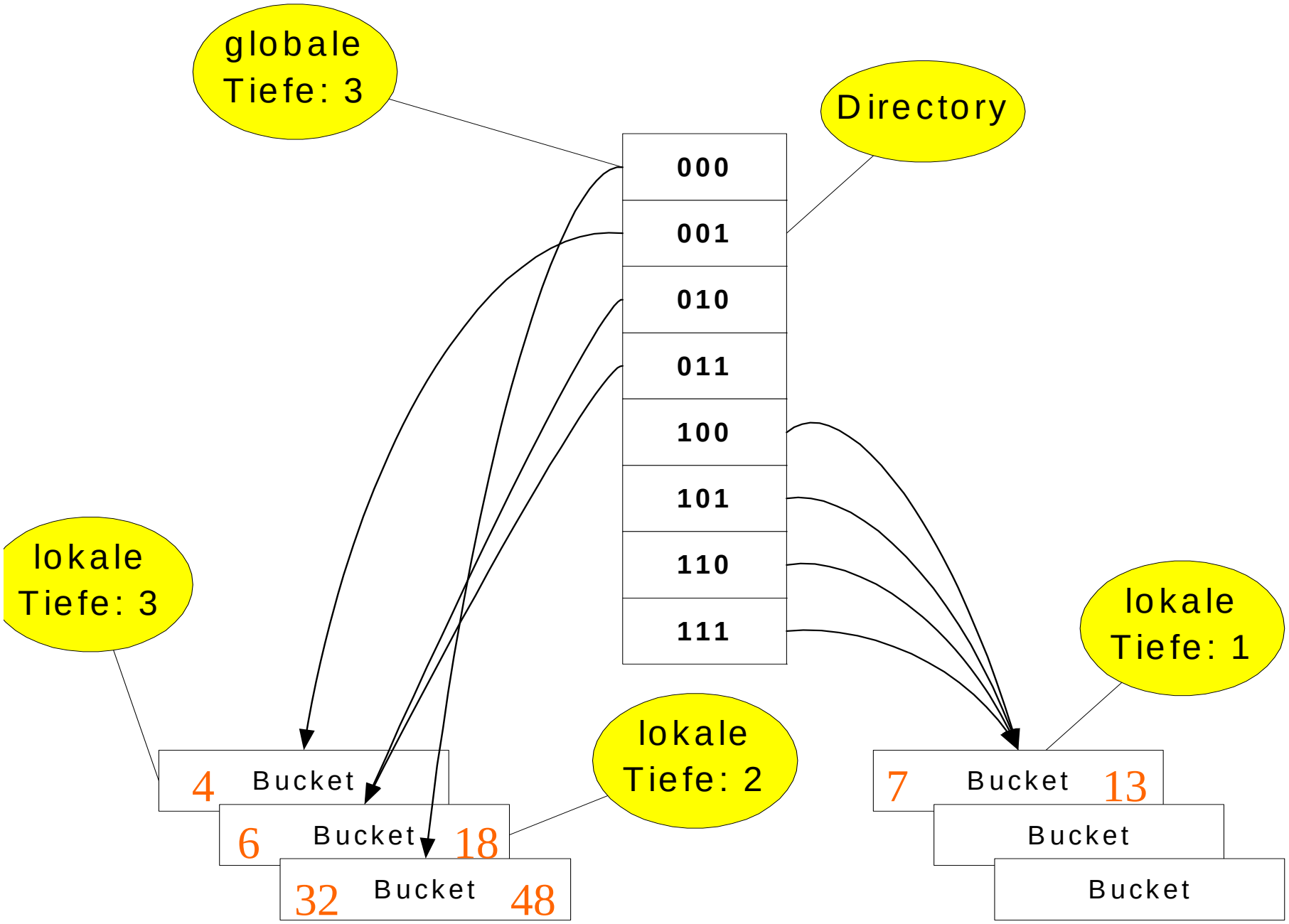
Hashfunktion für erweiterbares Hashing

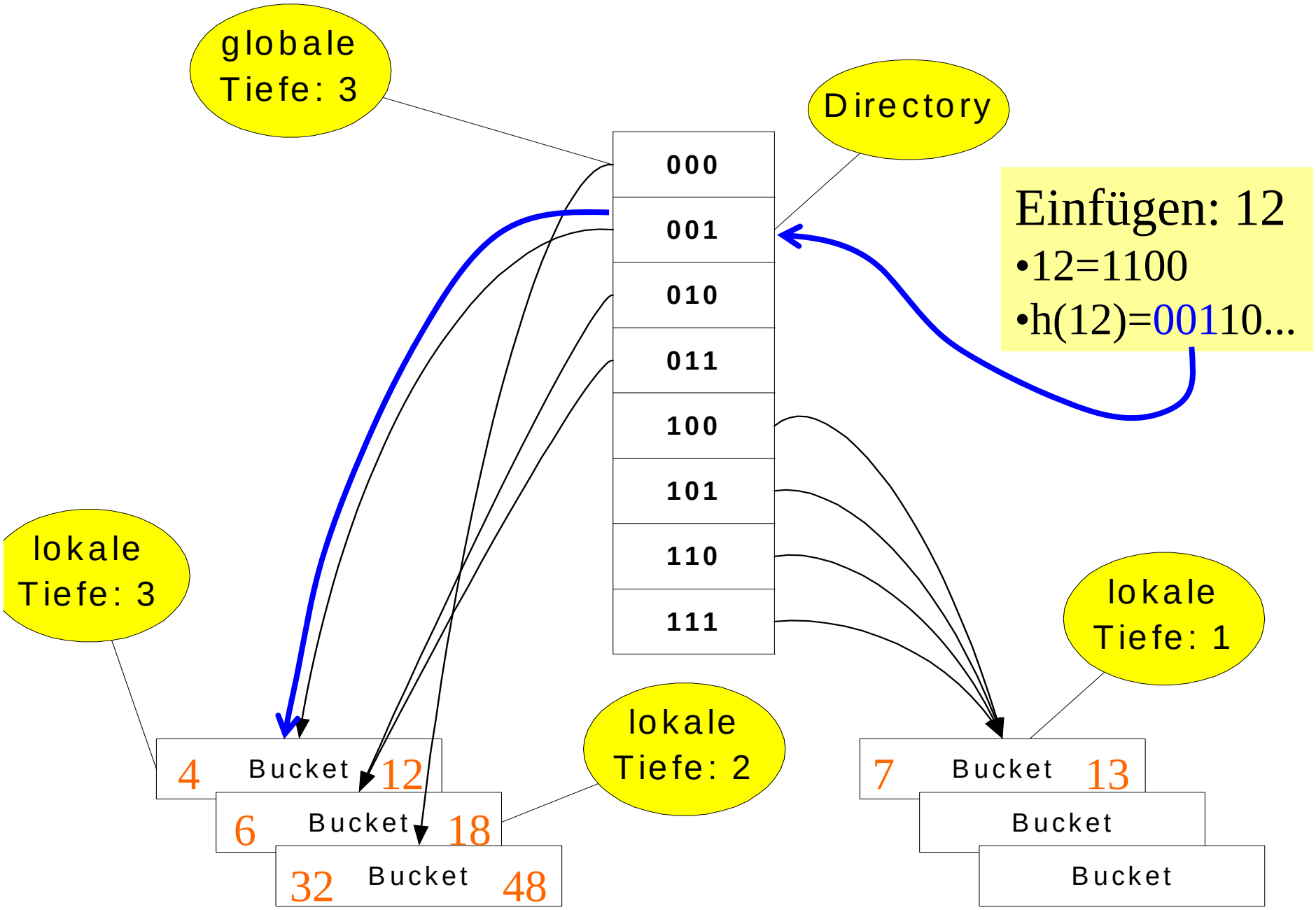
- h: Schlüsselmenge $\rightarrow$ {0,1}*
- Der Bitstring muss lang genug sein, um alle Objekte auf ihre Buckets abbilden zu können
- Anfangs wird nur ein (kurzer) Präfix des Hashwertes (Bitstrings) benötigt
- Wenn die Hashtabelle wächst wird aber sukzessive ein längerer Präfix benötigt
- Beispiel-Hashfunktion: 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)

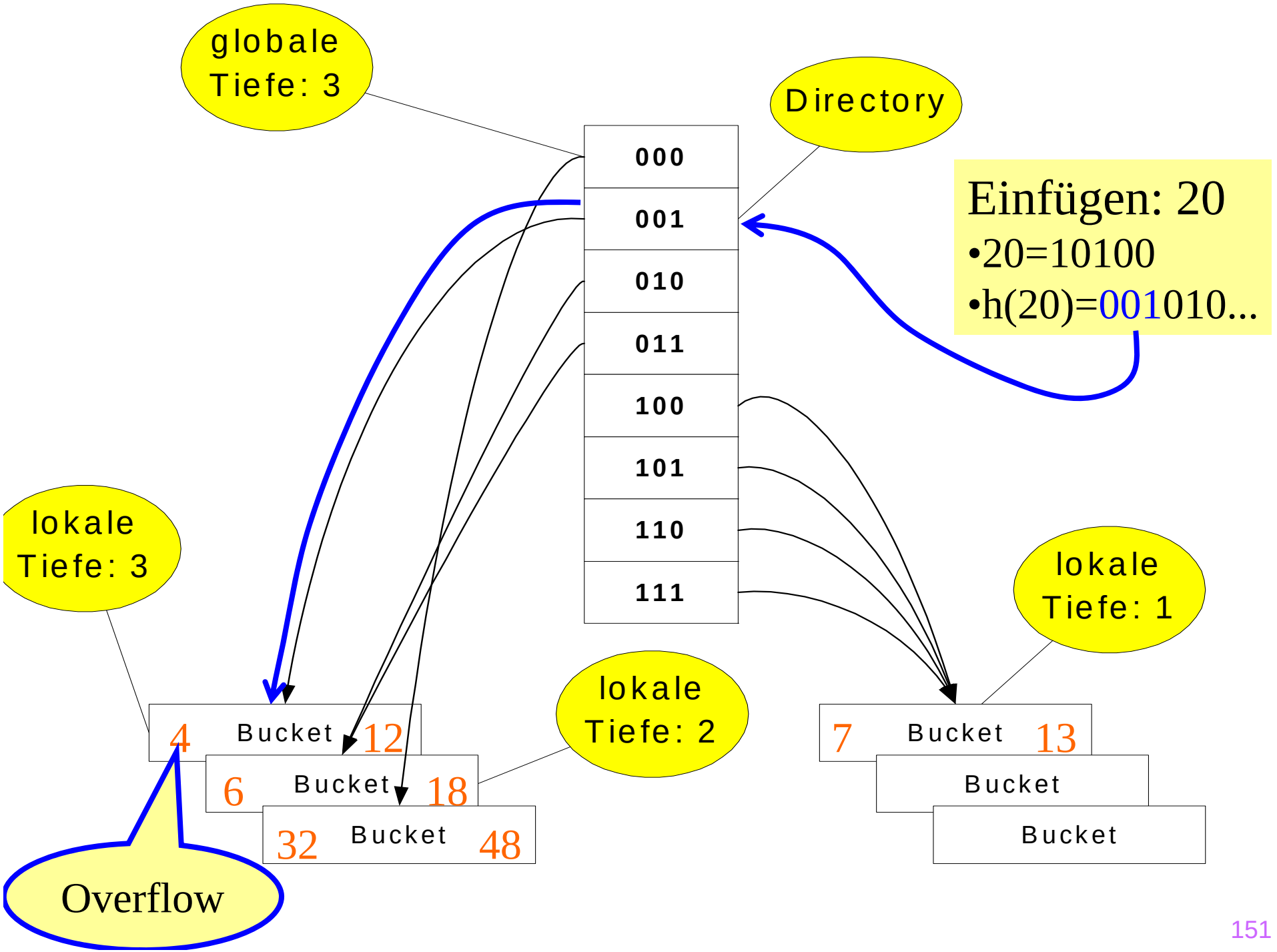

0
1
0
1
0
1
0
1
0
1
0
1
0
1
4
Bucket
6
Bucket
18
32
Bucket
48
7
Bucket
13
Bucket
Bucket

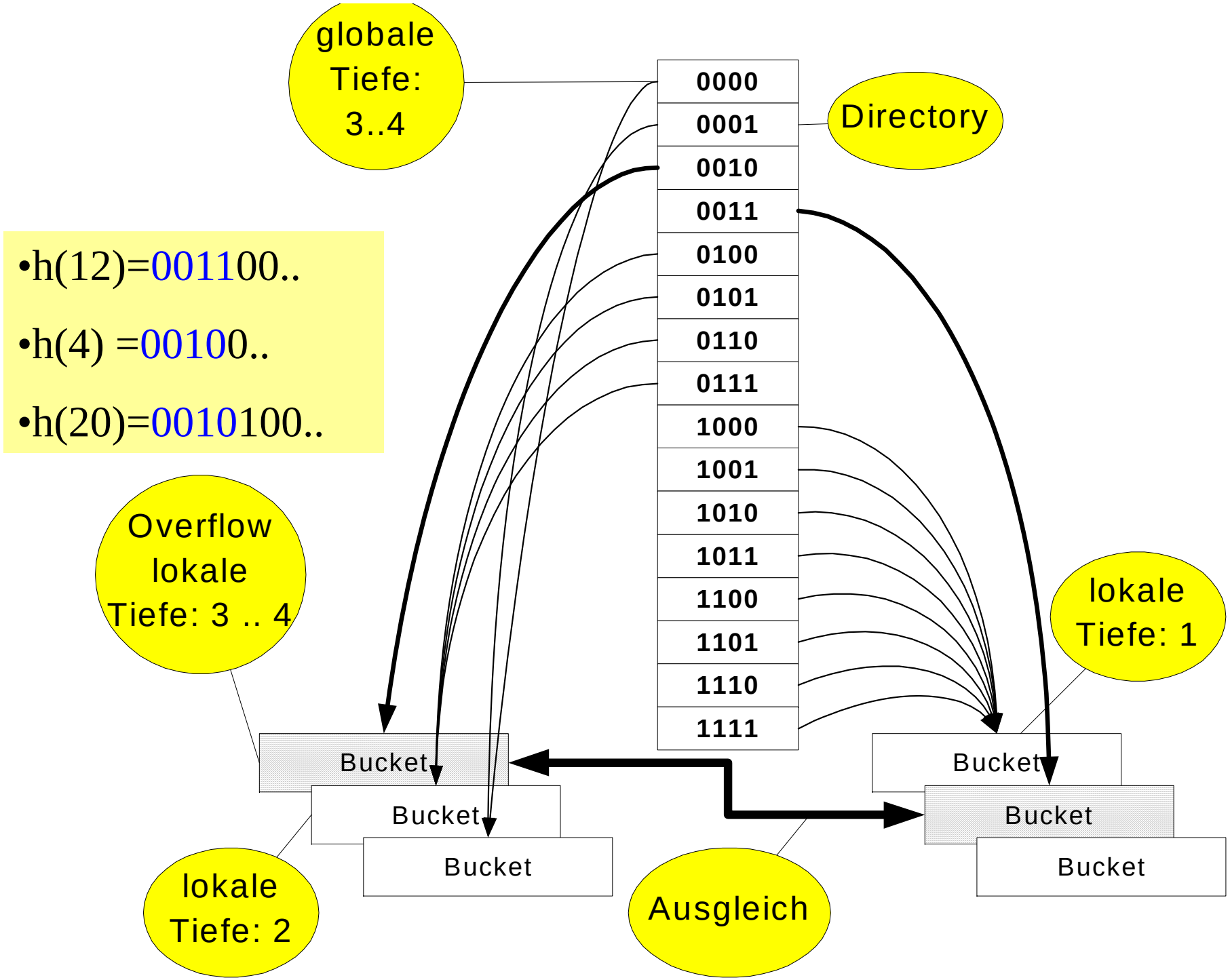

001
110
0
1
0
1
0
1
0
1
0
1
0
1
0
1
Präfix
001
4
Bucket
6
Bucket
18
32
Bucket
48
7
Bucket
13
Bucket
Bucket
Präfix
1

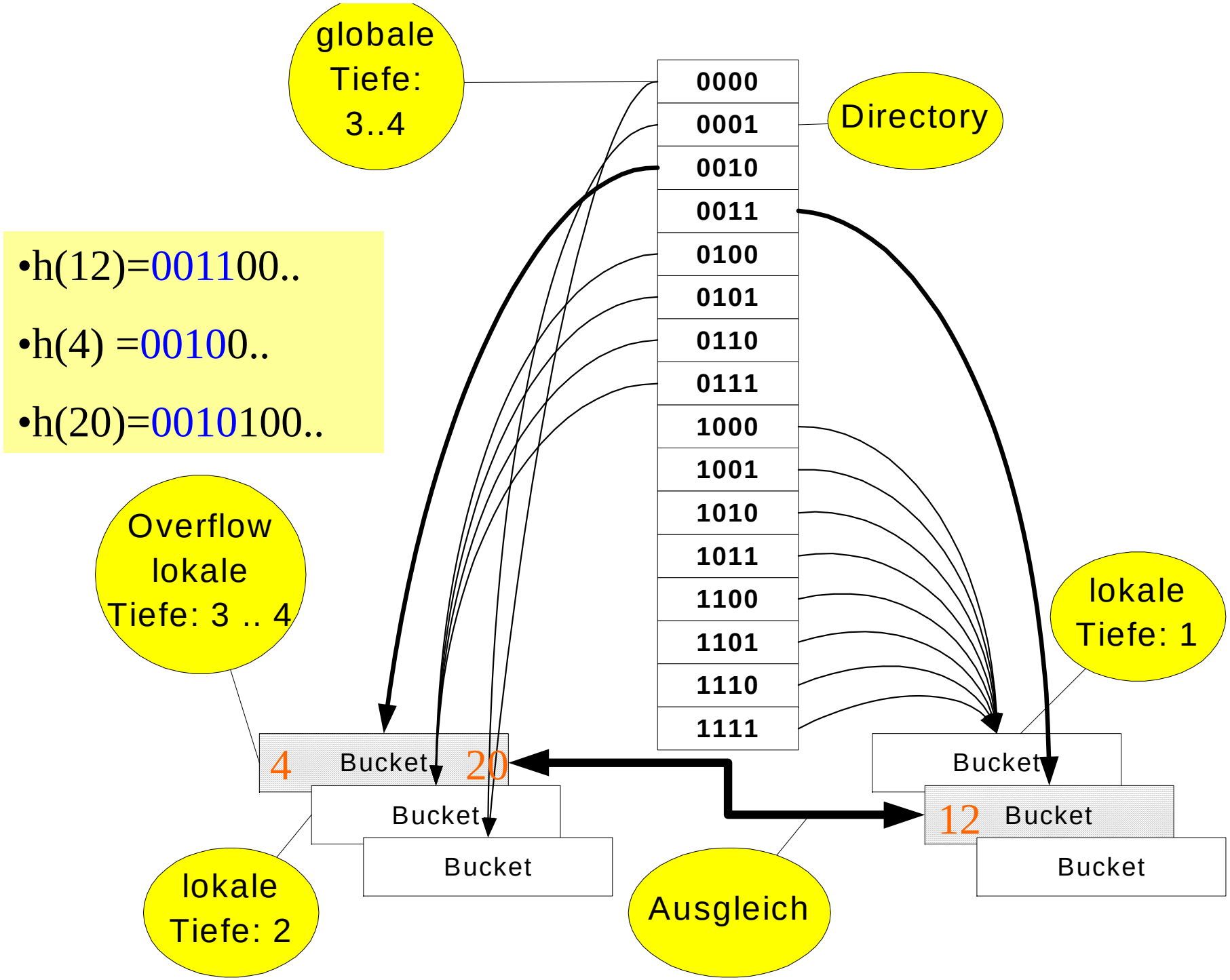

000
001
010
011
100
101
110
111
Bucket
Bucket
Bucket
Bucket
Bucket
Bucket

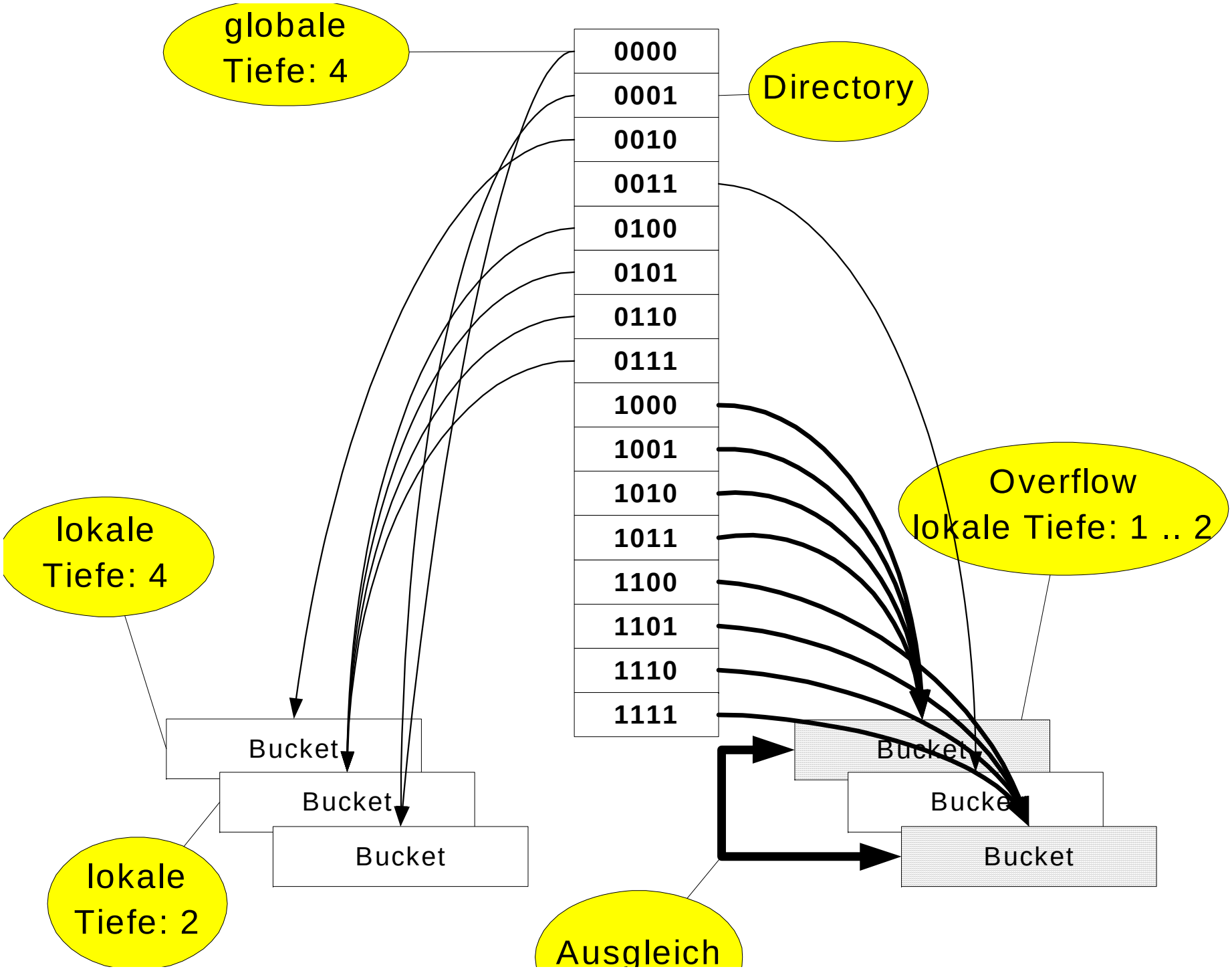

globale
Tiefe: 3
Directory
000
001
010
011
100
101
110
111
lokale
Tiefe: 3
lokale
Tiefe: 2
lokale
Tiefe: 1
4
Bucket
6
Bucket
18
32
Bucket
48
7
Bucket
13
Bucket
Bucket


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


globale
Tiefe: 3
Directory
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
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
Directory
•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

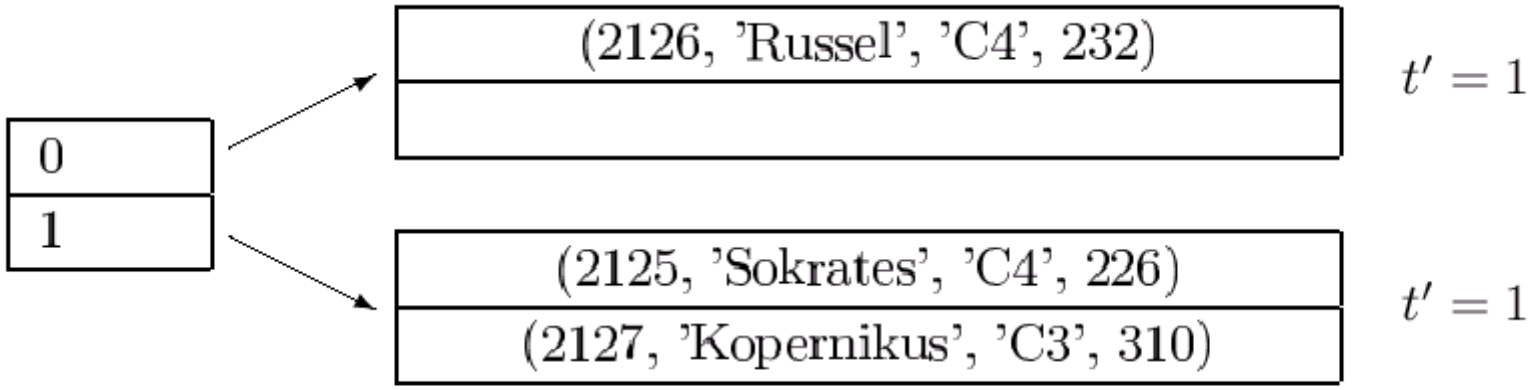

globale
Tiefe:
3..4
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
Directory
•h(12)=001100..
•h(4) =00100..
•h(20)=0010100..
Overflow
lokale
Tiefe: 3 .. 4
lokale
Tiefe: 1
4
Bucket
20
Bucket
Bucket
Bucket
12
Bucket
Bucket
lokale
Tiefe: 2
Ausgleich


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

atei Bearbeiten Dokument Werkzeuge Anzeige Fenster Hilfe

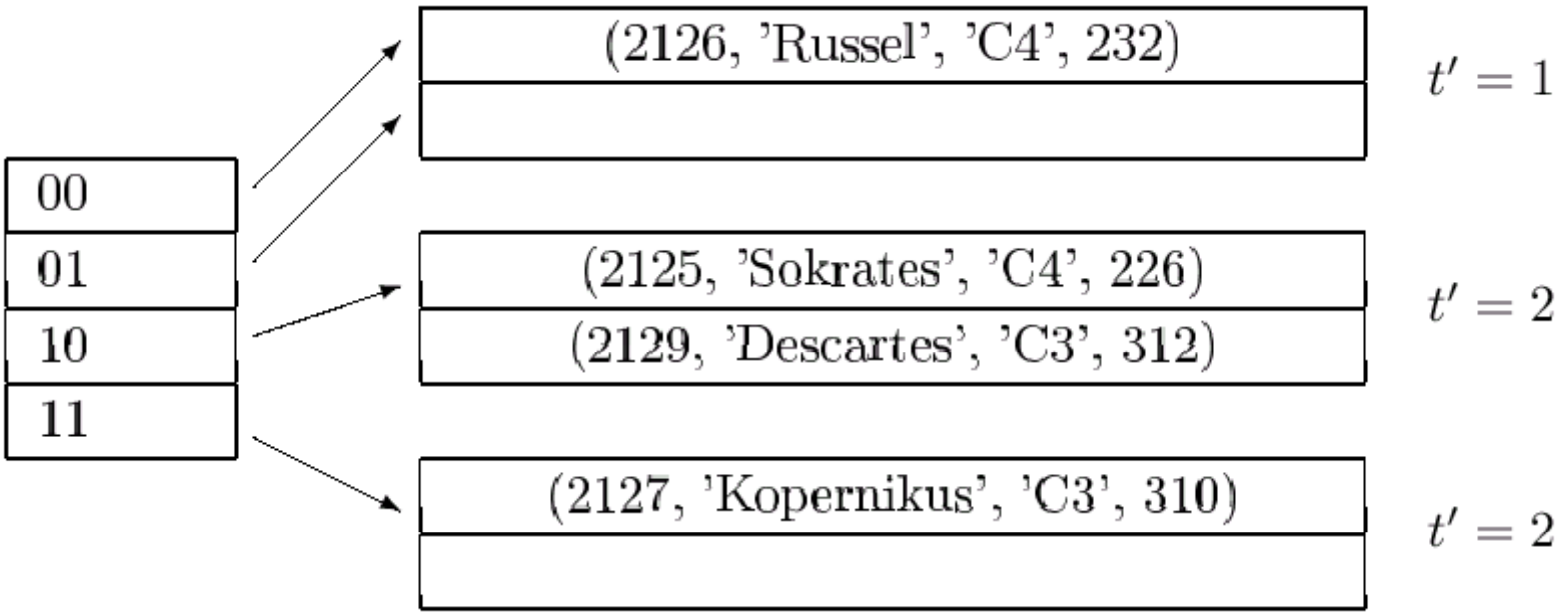
Demonstration des erweiterbaren Hashings

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



150% 200 von 511 210,3 x 297 mm

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

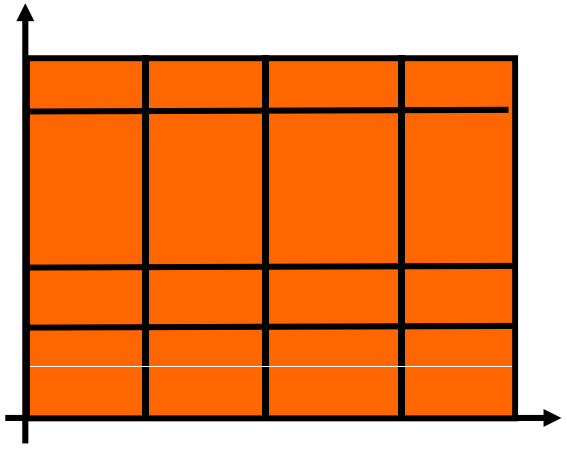
1. Exact Match Query
   spezifiziert Suchwert für jede Dimension D_i
2. Partial Match Query
   spezifiziert Suchwert für einen Teil der Dimensionen
3. Range Query
   spezifiziert ein Suchintervall $[ug_i, og_i]$ für alle Dimensionen
4. Partial Range Query
   spezifiziert ein Suchintervall für einen Teil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

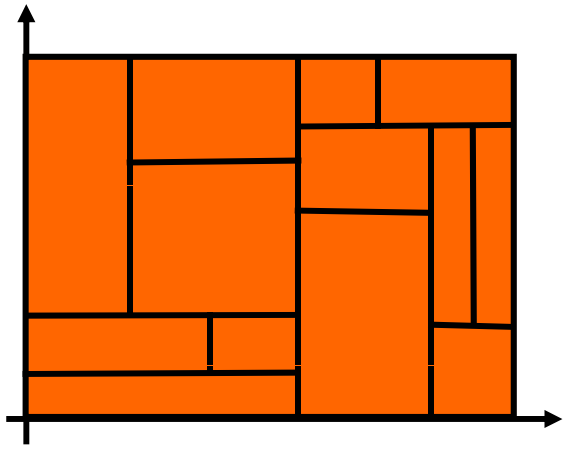


Charakterisierung mehrdimensionaler Datenstrukturen

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

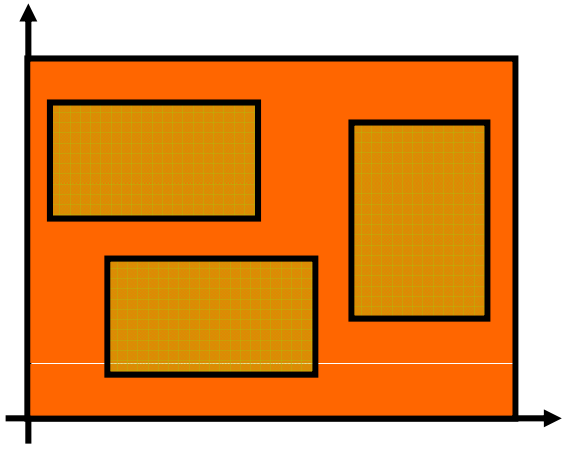


Charakterisierung mehrdimensionaler Datenstrukturen

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

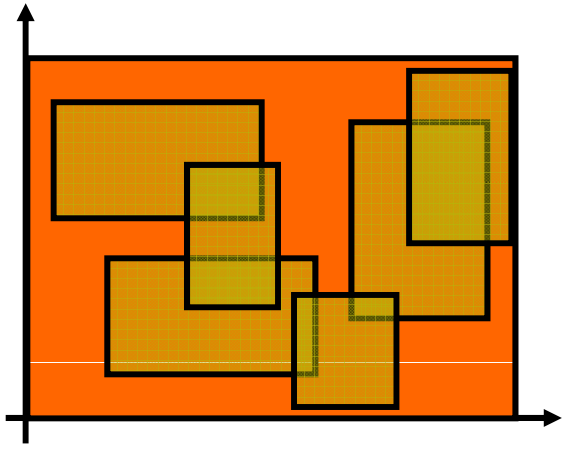


Charakterisierung mehrdimensionaler Datenstrukturen

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

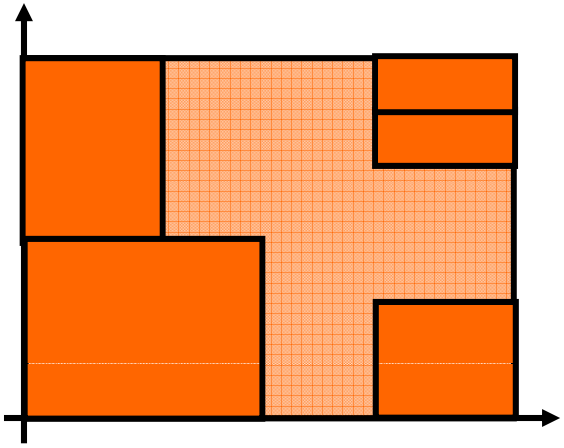


Charakterisierung mehrdimensionaler Datenstrukturen

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

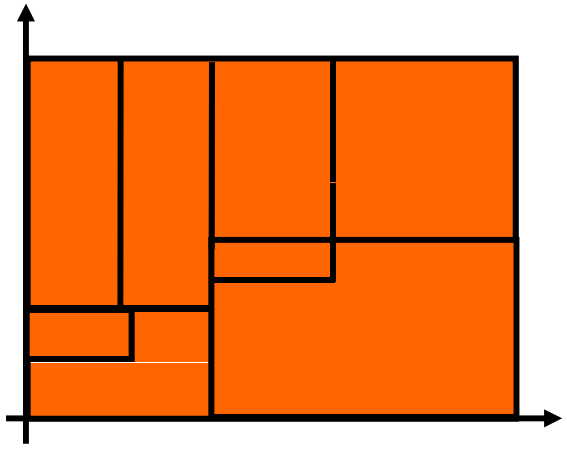


Charakterisierung mehrdimensionaler Datenstrukturen

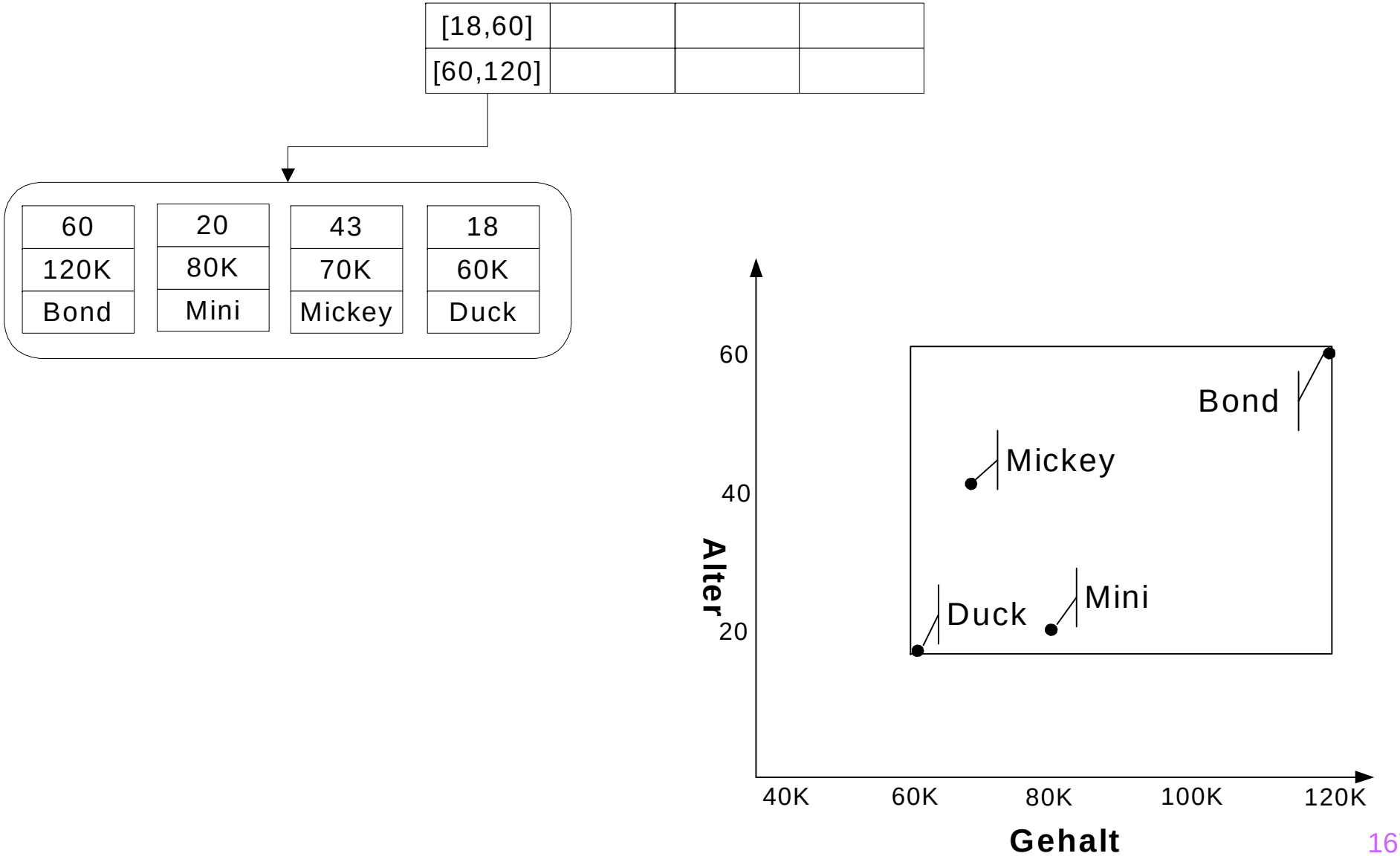
Mehrdimensionale Zugriffsstrukturen können gemäß der Art der Aufteilung des Datenraums in Gebiete charakterisiert werden:

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

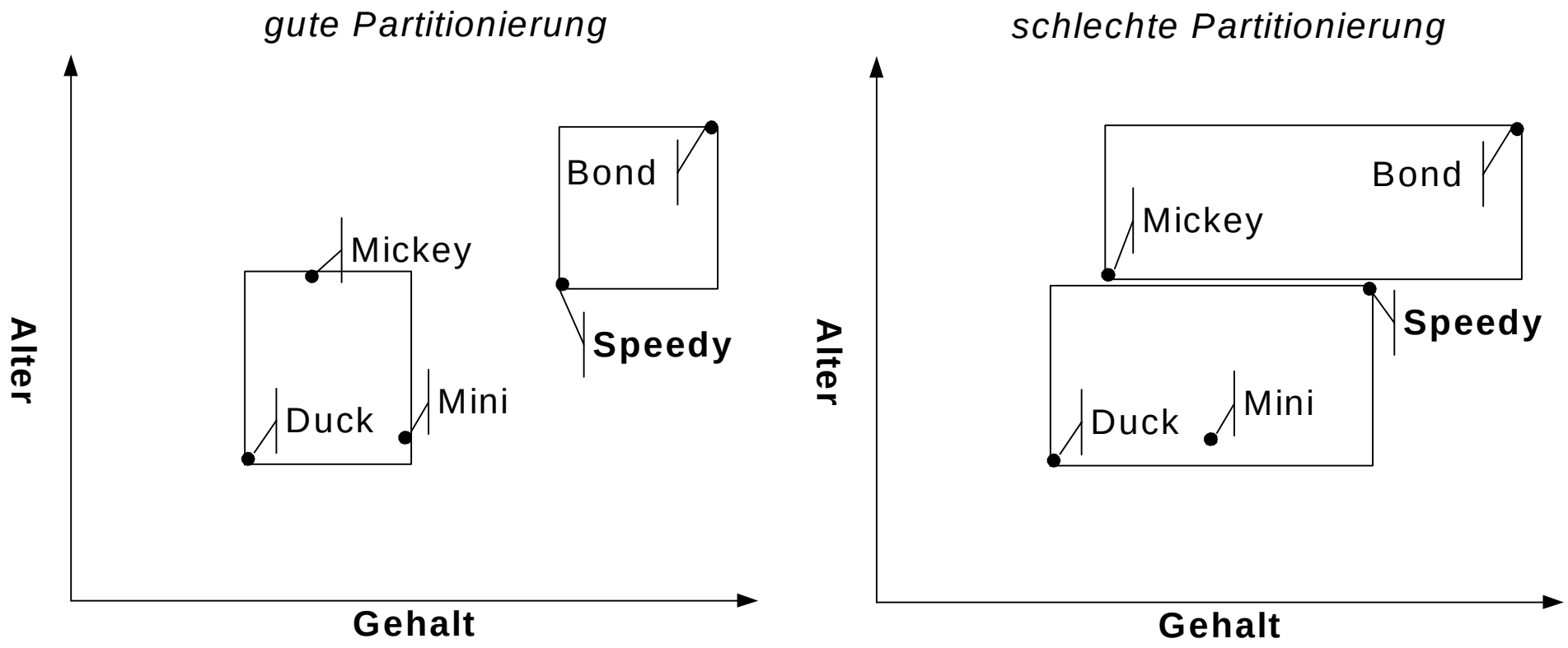
Z-B-Baum: vollständig,disjunkt



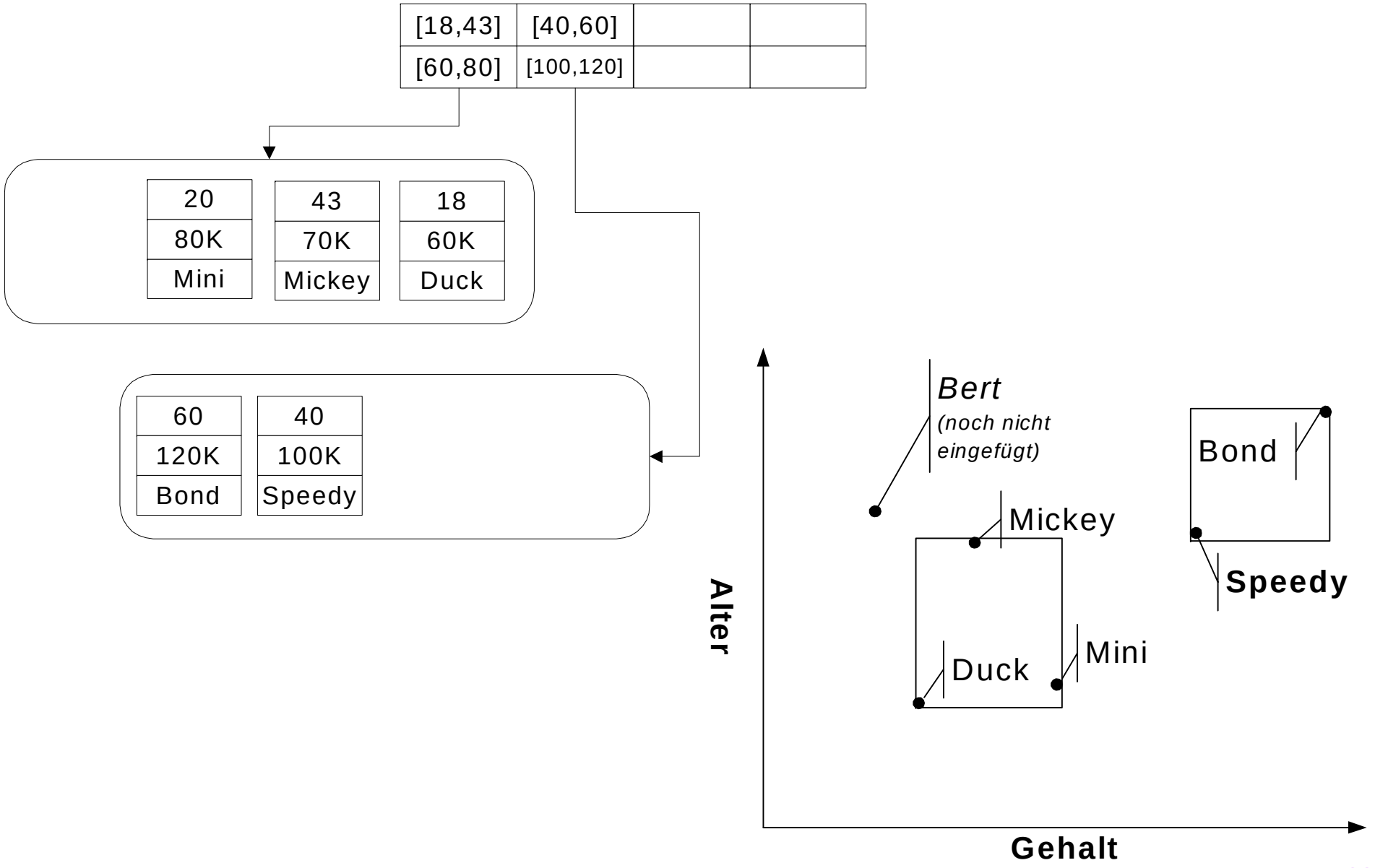
R-Baum: Urvater der baum-strukturierten mehrdimensionalen Zugriffsstrukturen



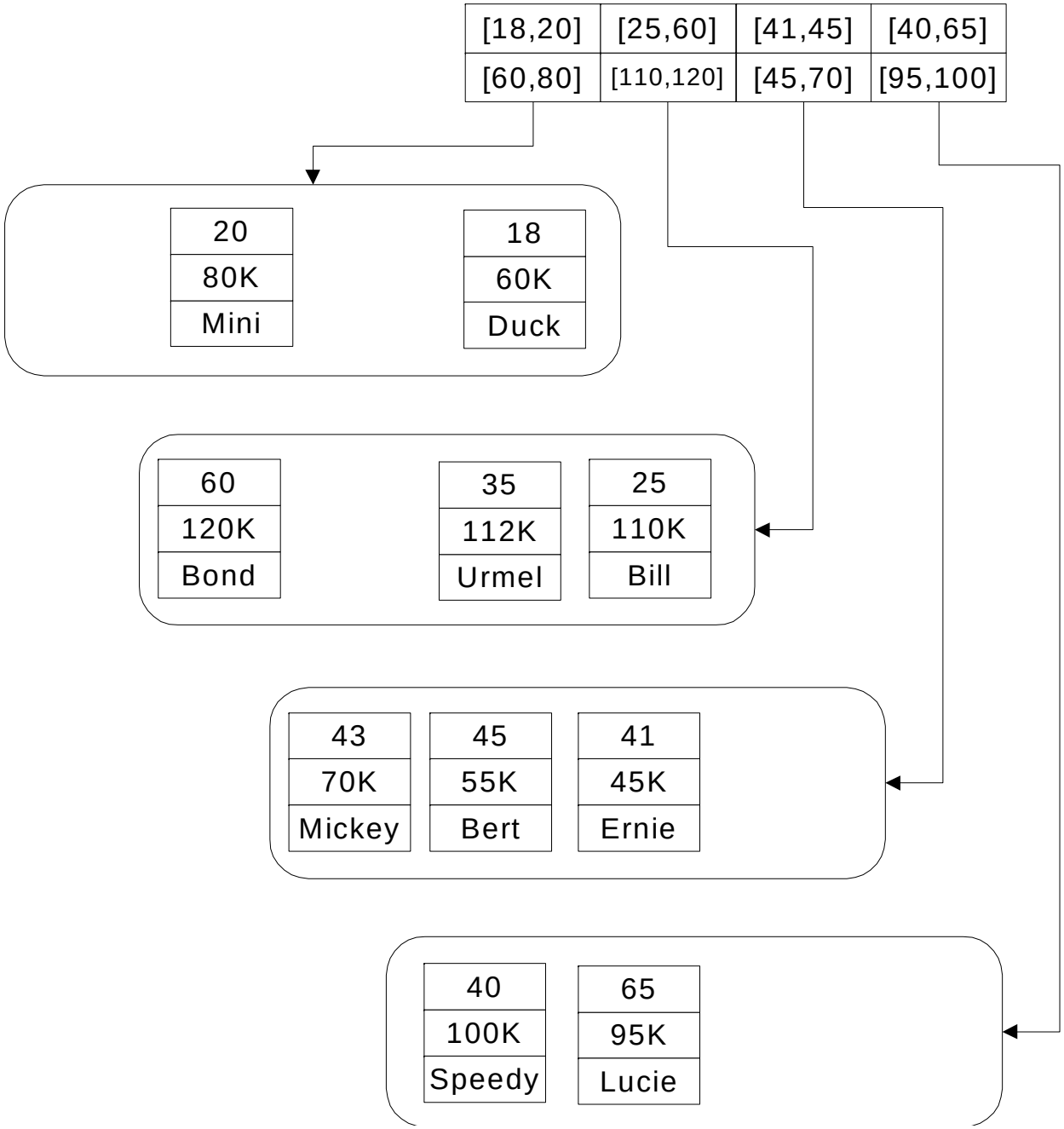
Gute versus schlechte Partitionierung

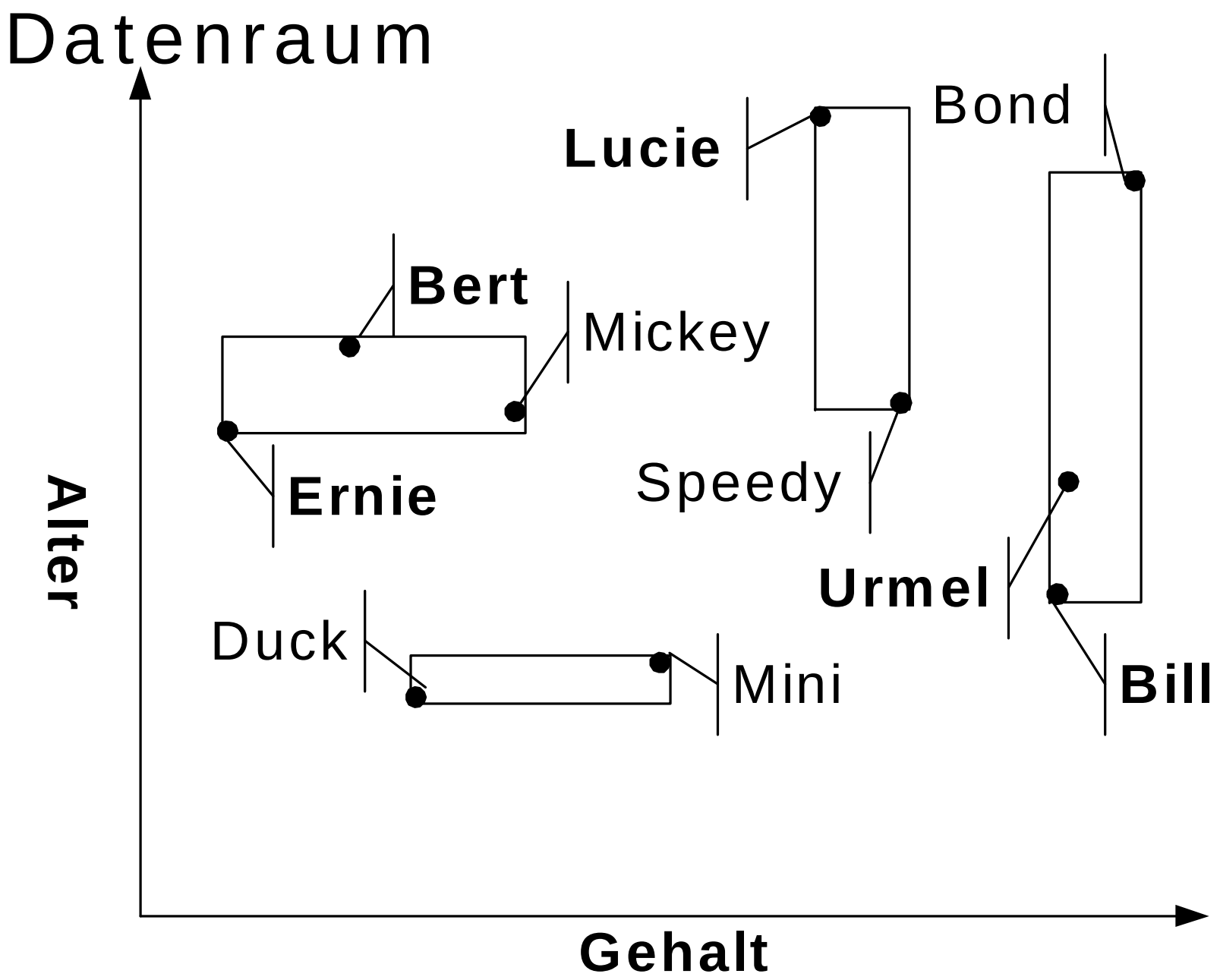


Nächste Phase in der Entstehungsgeschichte des R-Baums

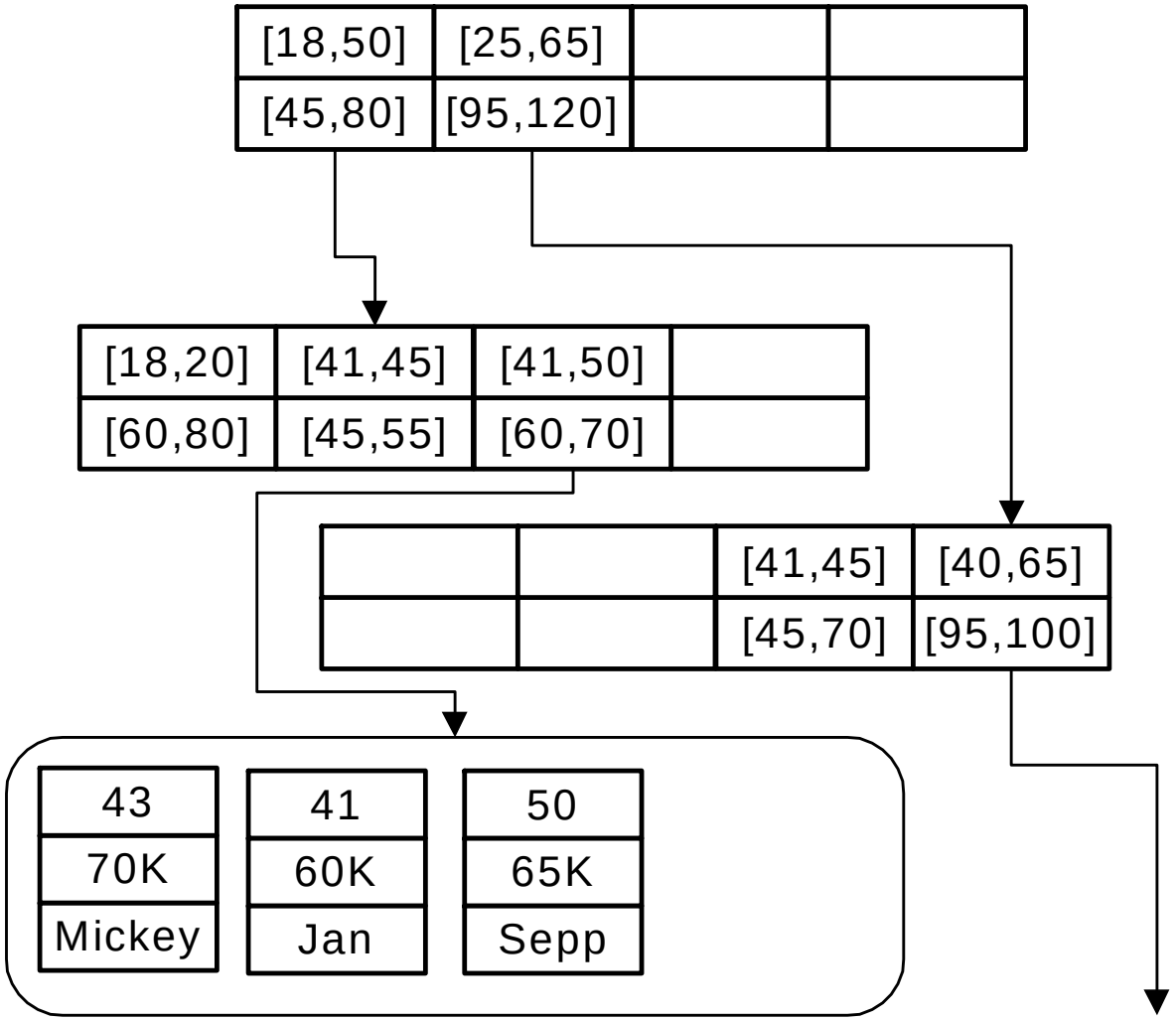


Nächste Phase



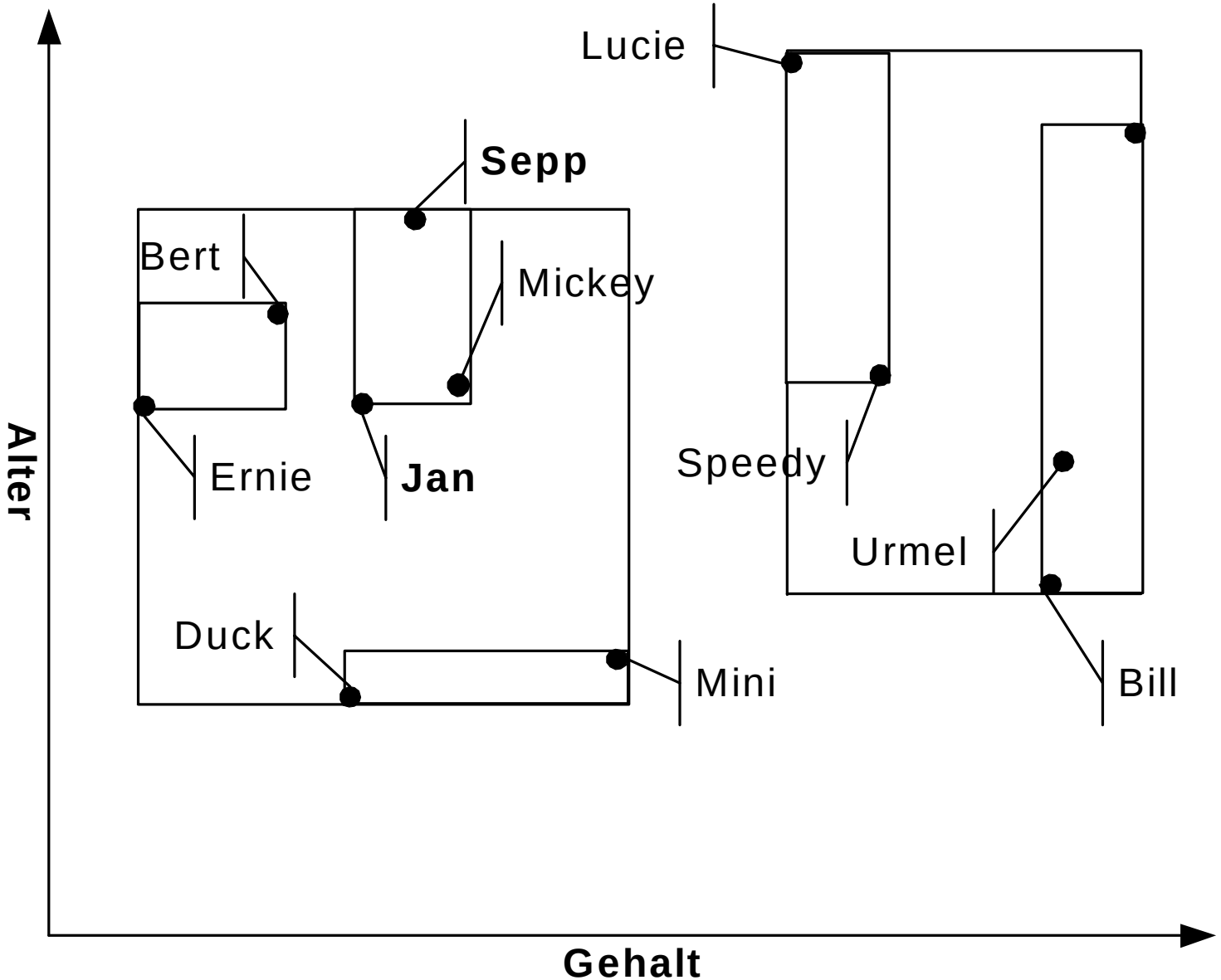

Datenraum
Alter
Gehalt
Lucie
Bond
Bert
Mickey
Ernie
Speedy
Urmel
Duck
Mini
Bill

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

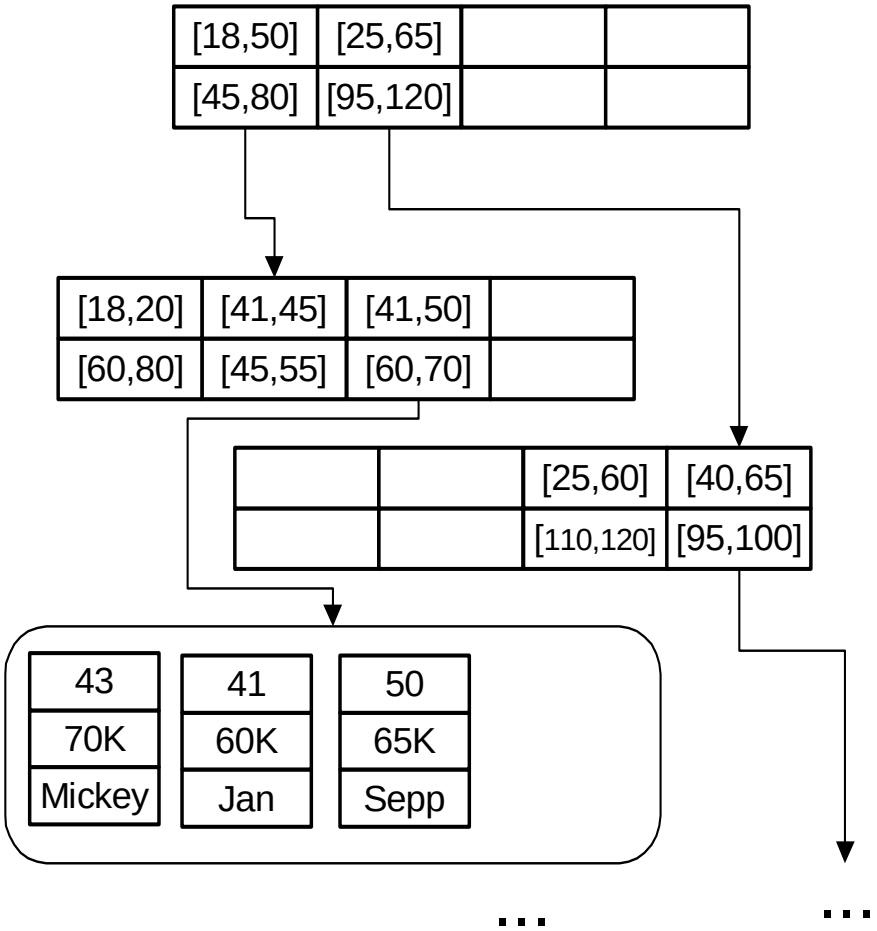


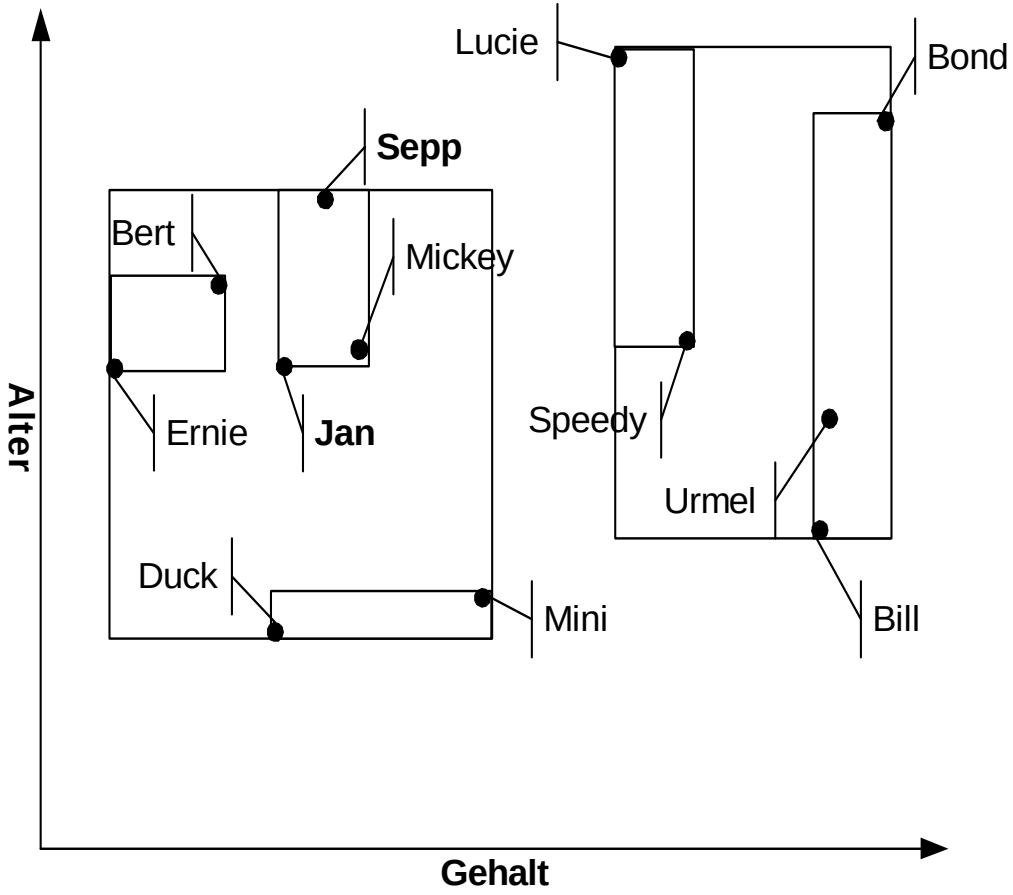
...

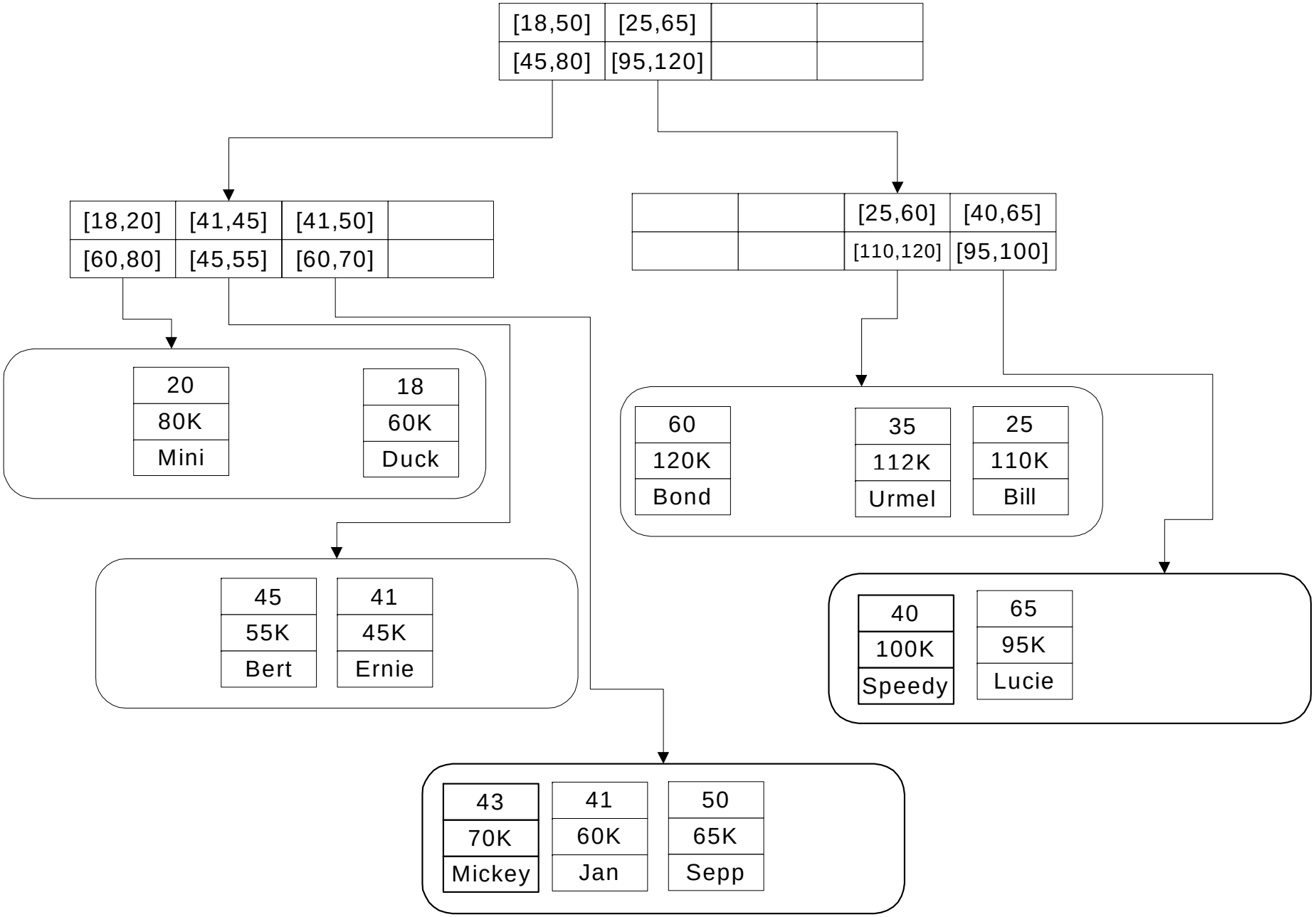
Datenraum


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

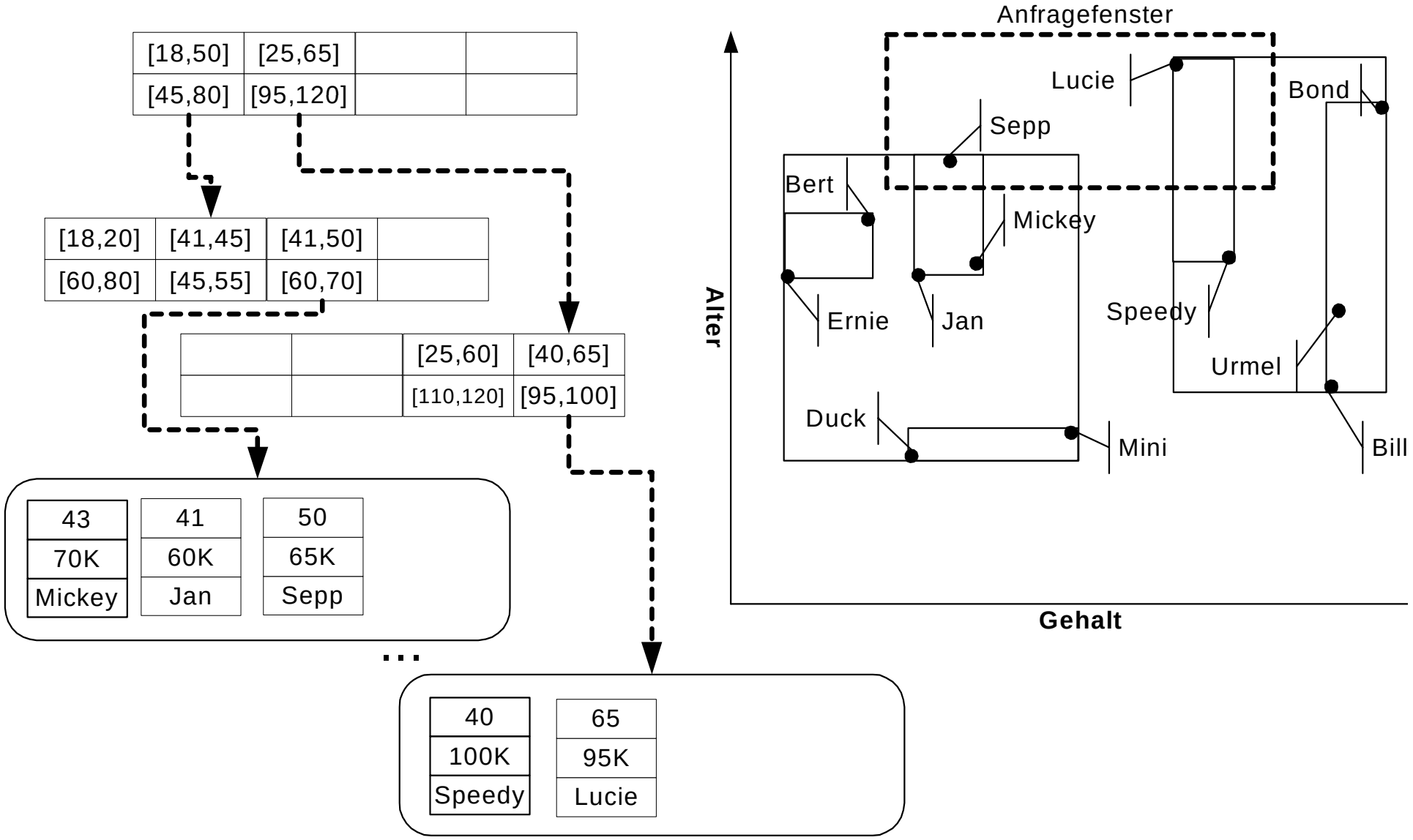
Überblick

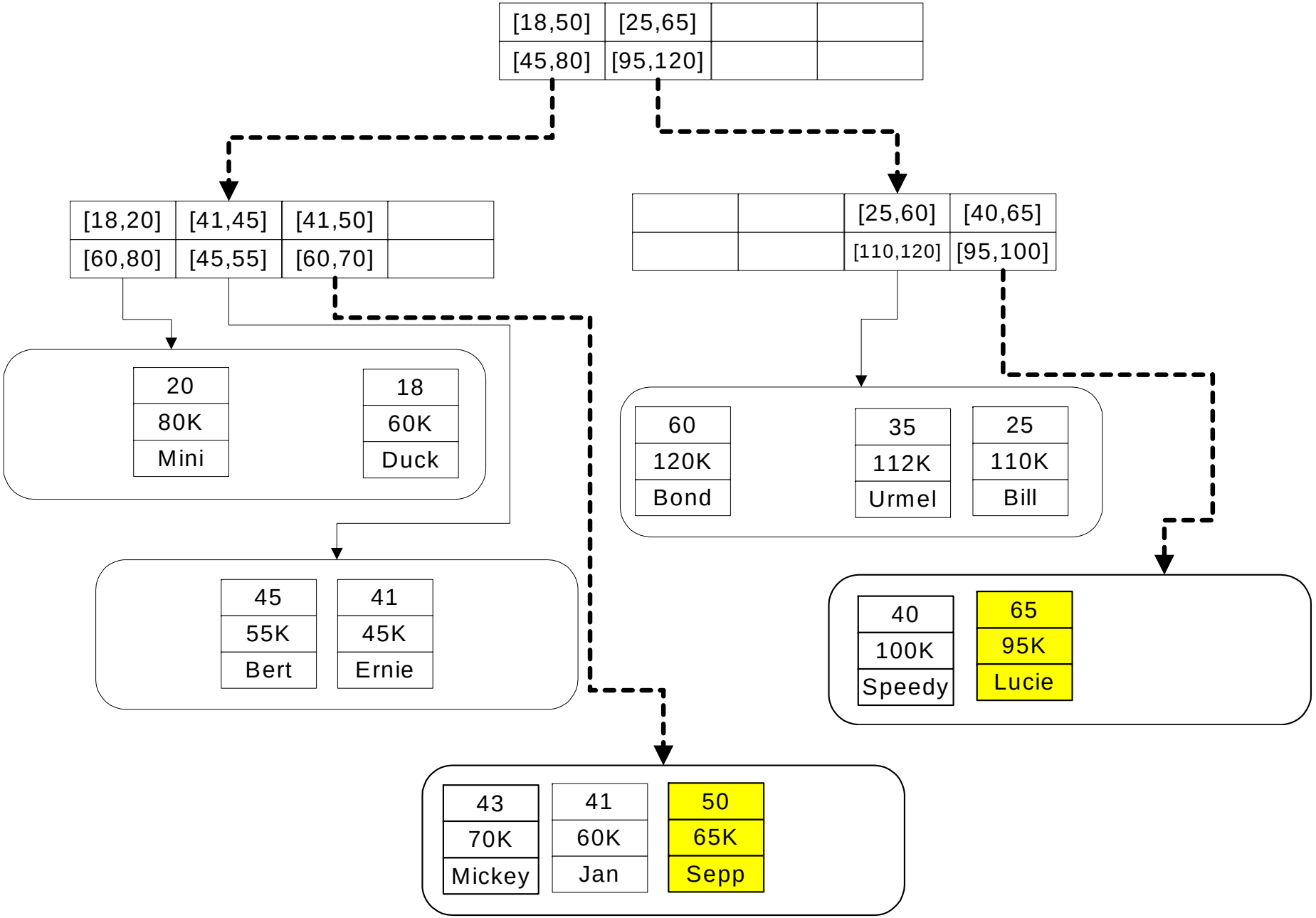

[18,50]
[25,65]
[45,80]
[95,120]
[18,20]
[41,45]
[41,50]
[60,80]
[45,55]
[60,70]
[25,60]
[40,65]
[110,120]
[95,100]
43
70K
Mickey
41
60K
Jan
50
65K
Sepp
...
...


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

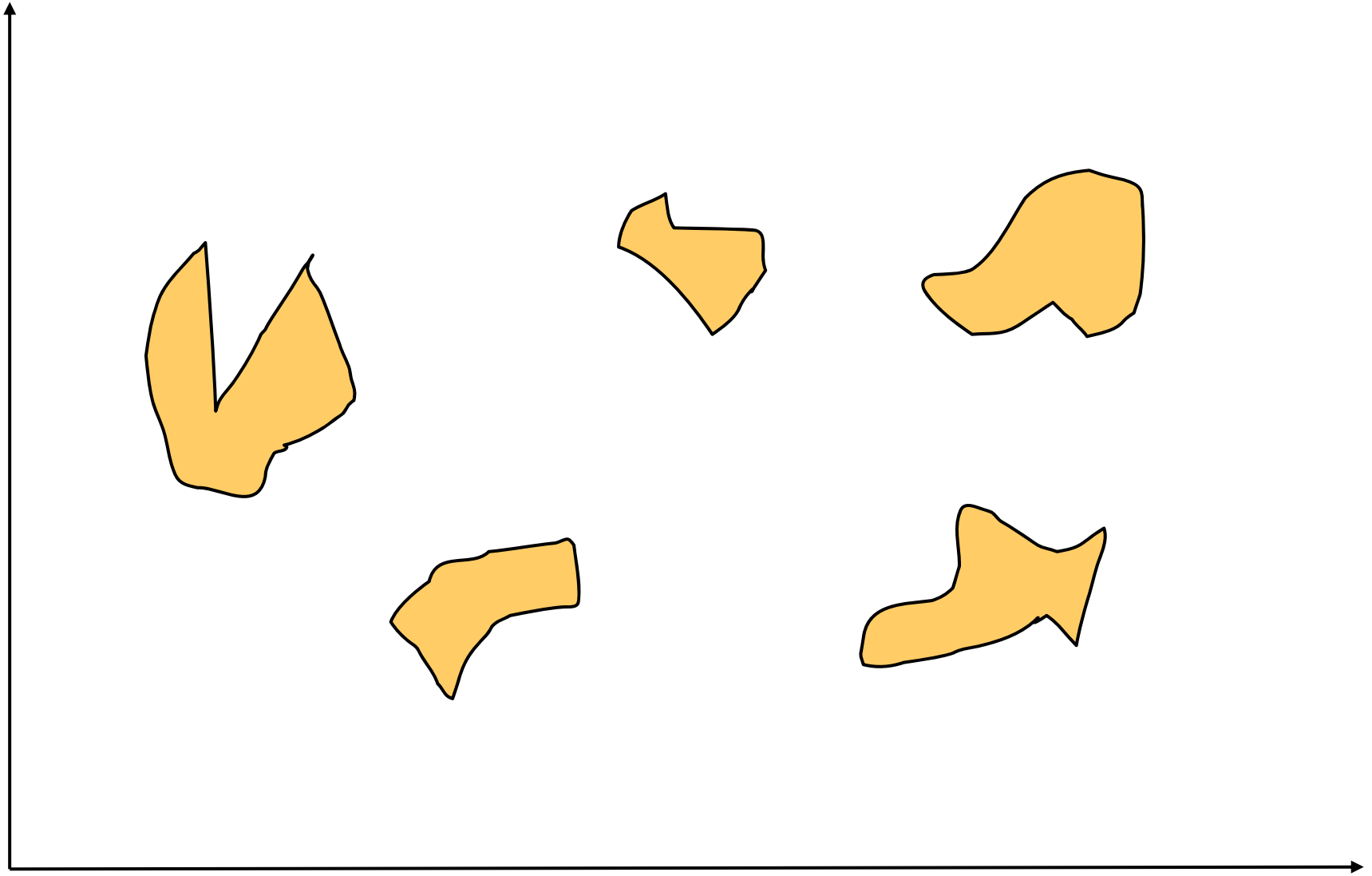

[18,50]
[25,65]
[45,80]
[95,120]
[18,20]
[41,45]
[41,50]
[60,80]
[45,55]
[60,70]
[25,60]
[40,65]
[110,120]
[95,100]
20
80K
Mini
18
60K
Duck
60
120K
Bond
35
112K
Urmel
25
110K
Bill
45
55K
Bert
41
45K
Ernie
40
100K
Speedy
65
95K
Lucie
43
70K
Mickey
41
60K
Jan
50
65K
Sepp

Bereichsanfragen auf dem R-Baum

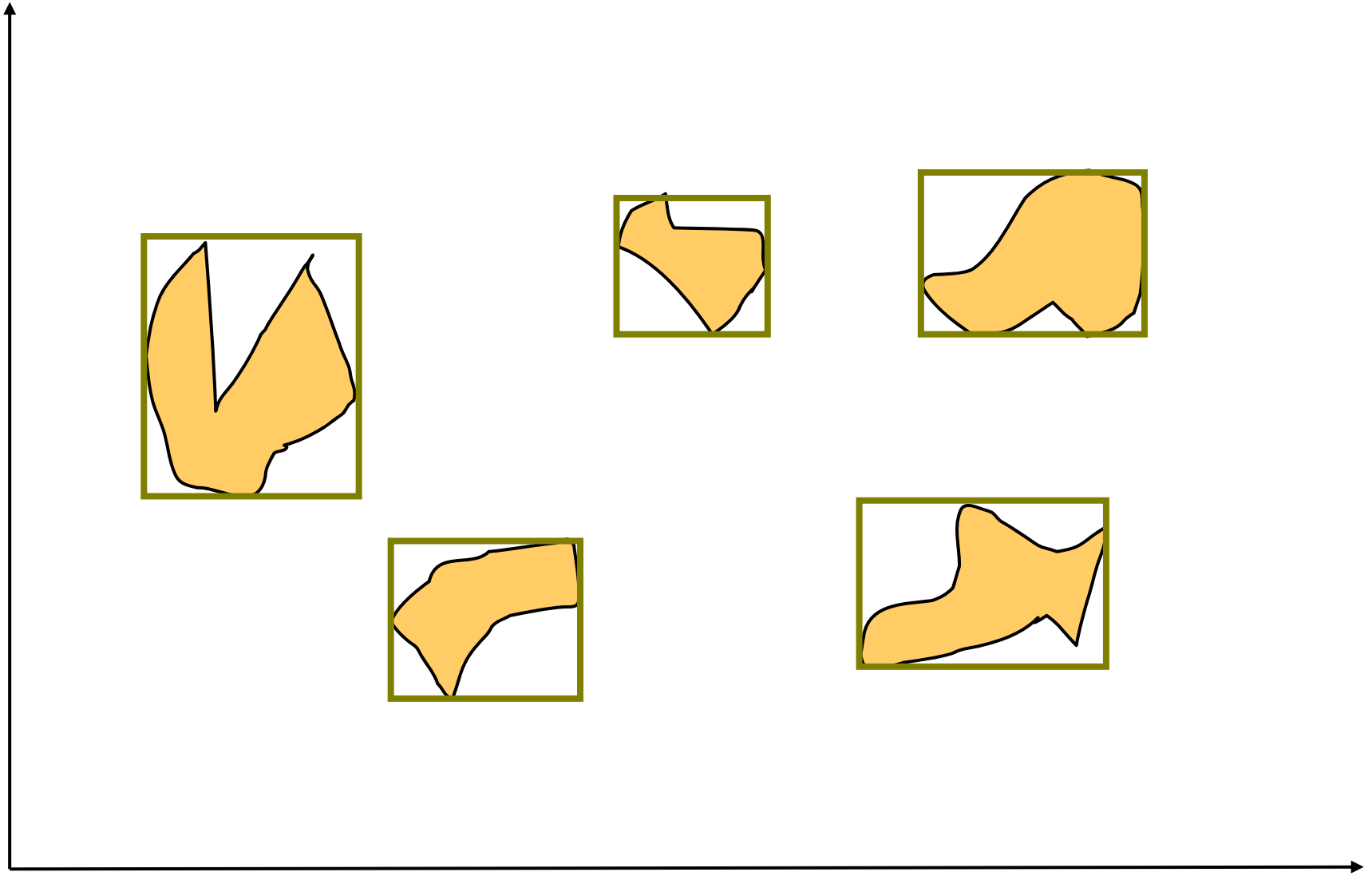



[18,50]
[25,65]
[45,80]
[95,120]
[18,20]
[41,45]
[41,50]
[60,80]
[45,55]
[60,70]
[25,60]
[40,65]
[110,120]
[95,100]
20
80K
Mini
18
60K
Duck
60
120K
Bond
35
112K
Urmel
25
110K
Bill
45
55K
Bert
41
45K
Ernie
40
100K
Speedy
65
95K
Lucie
43
70K
Mickey
41
60K
Jan
50
65K
Sepp

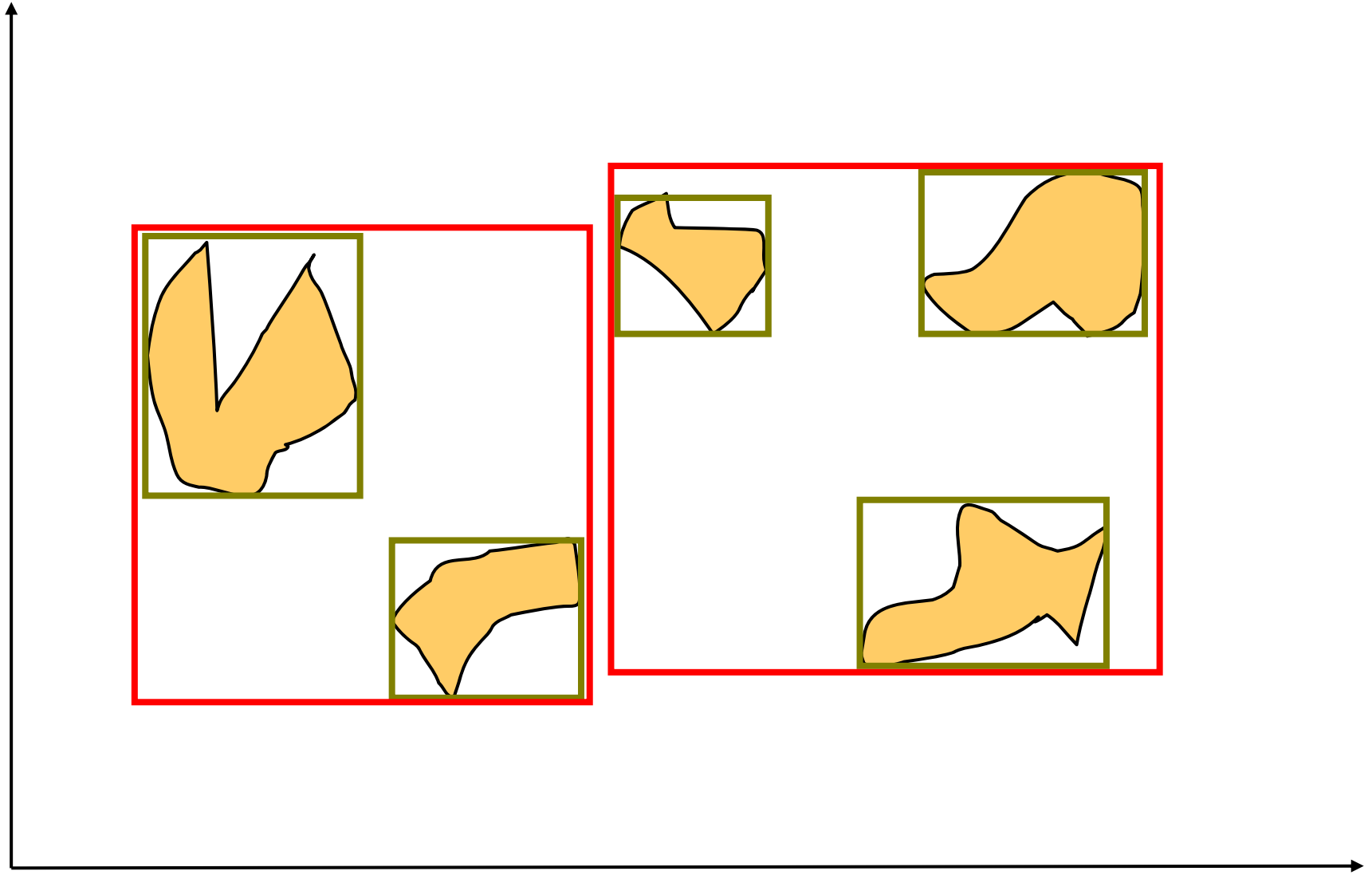
Indexierung räumlicher Objekte (anstatt Punkten) mit dem R-Baum



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



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



Bitmap-Indexe

w_{18}	w_{19}
18	**19**
0	0
1	0
1	0
0	0
0	1
...	...

Kunden				
KundenNr	Name	wiealt	Geschlecht	...
007	Bond	43	m	...
4013	Mini	18	w	...
4315	Mickey	18	m	...
4711	Kemper	43	m	...
5913	Twiggy	19	w	...
...	...	...	...	...

G_m	G_w
m	**w**
1	0
0	1
1	0
1	0
0	1
...	...

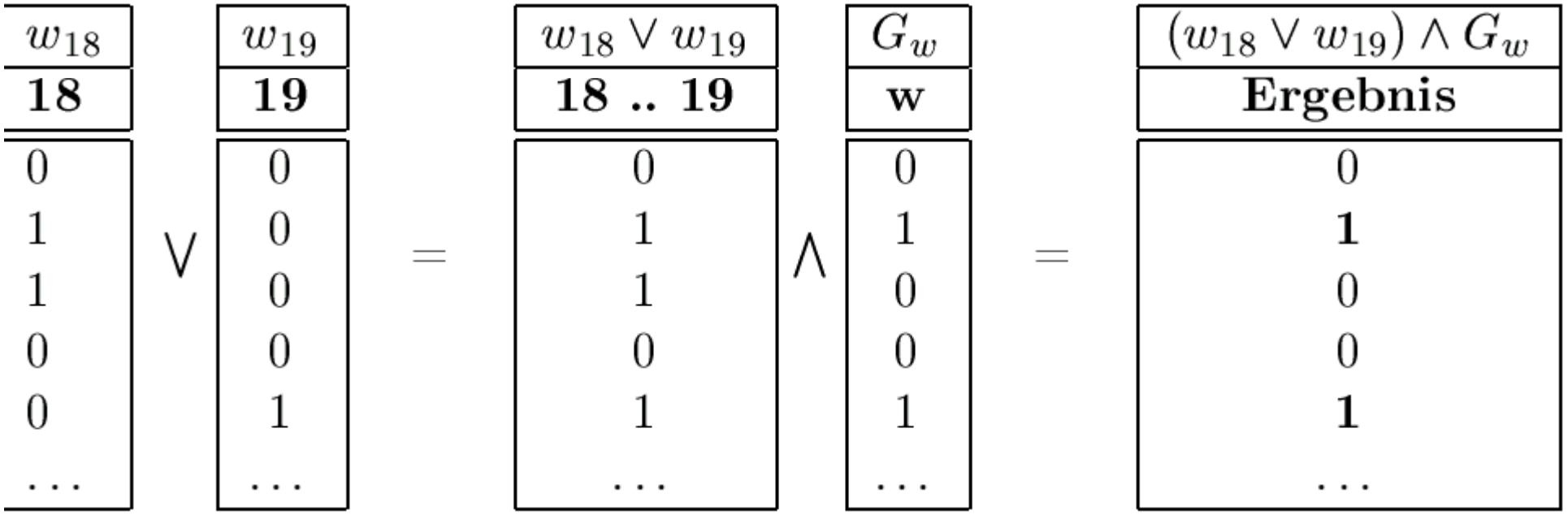
- Optimierung durch Komprimierung der Bitmaps
- Ausnutzung der dünnen Besetzung
  - Runlength-compression
    - Grundidee: speichere jeweils die Länge der Nullfolgen zwischen zwei Einsen
  - Mehrmodus-Komprimierung:
    - bei langen Null/Einsfolgen speichere deren Länge
    - Sonst speichere das Bitmuster

Beispiel-Anfrage und Auswertung

```
select k.Name, ...
from Kunden k
where k.Geschlecht = 'w' and
        k.wiealt between 18 and 19;
```

$$(w_{18} \vee w_{19}) \wedge G_w$$

Bitmap-Operationen



ile Go Page Scroll Tools View Window Help

Verkäufe		
TID	...	KundenNr
i	...	007
ii	...	4711
iii	...	007
iv	...	007
v	...	4711
vi	...	007
...	...	...

Join-Index	
TID-V	TID-K
i	*II*
ii	*I*
iii	*II*
iv	*II*
v	*I*
vi	*II*
...	...

Kunden		
TID	KundenNr	...
I	4711	...
II	007	...
III	...	...
...	...	...

bb. 17.17: Klassischer Join-Index

Bitmap-Join-Index

Verkäufe		
TID	...	KundenNr
i	...	007
ii	...	4711
iii	...	007
iv	...	007
v	...	4711
vi	...	007
...	...	...

J_I	J_{II}	J_{III}
0	1	...
1	0	...
0	1	...
0	1	...
1	0	...
0	1	...
...	...	...

...

Kunden		
TID	KundenNr	...
I	4711	...
II	007	...
III	...	...
...	...	...

Verkäufe		
TID	...	KundenNr
i	...	007
ii	...	4711
iii	...	007
iv	...	007
v	...	4711
vi	...	007
...	...	...

Join-Index	
TID-V	TID-K
i	*II*
ii	*I*
iii	*II*
iv	*II*
v	*I*
vi	*II*
...	...

Kunden		
TID	KundenNr	...
I	4711	...
II	007	...
III	...	...
...	...	...

bb. 17.17: Klassischer Join-Index

Bitmap-Join-Index

Verkäufe		
TID	...	KundenNr
i	...	007
ii	...	4711
iii	...	007
iv	...	007
v	...	4711
vi	...	007
...	...	...

J_I	J_{II}	J_{III}	
0	1	...	
1	0	...	
0	1	...	...
0	1	...	
1	0	...	
0	1	...	
...	...	...	

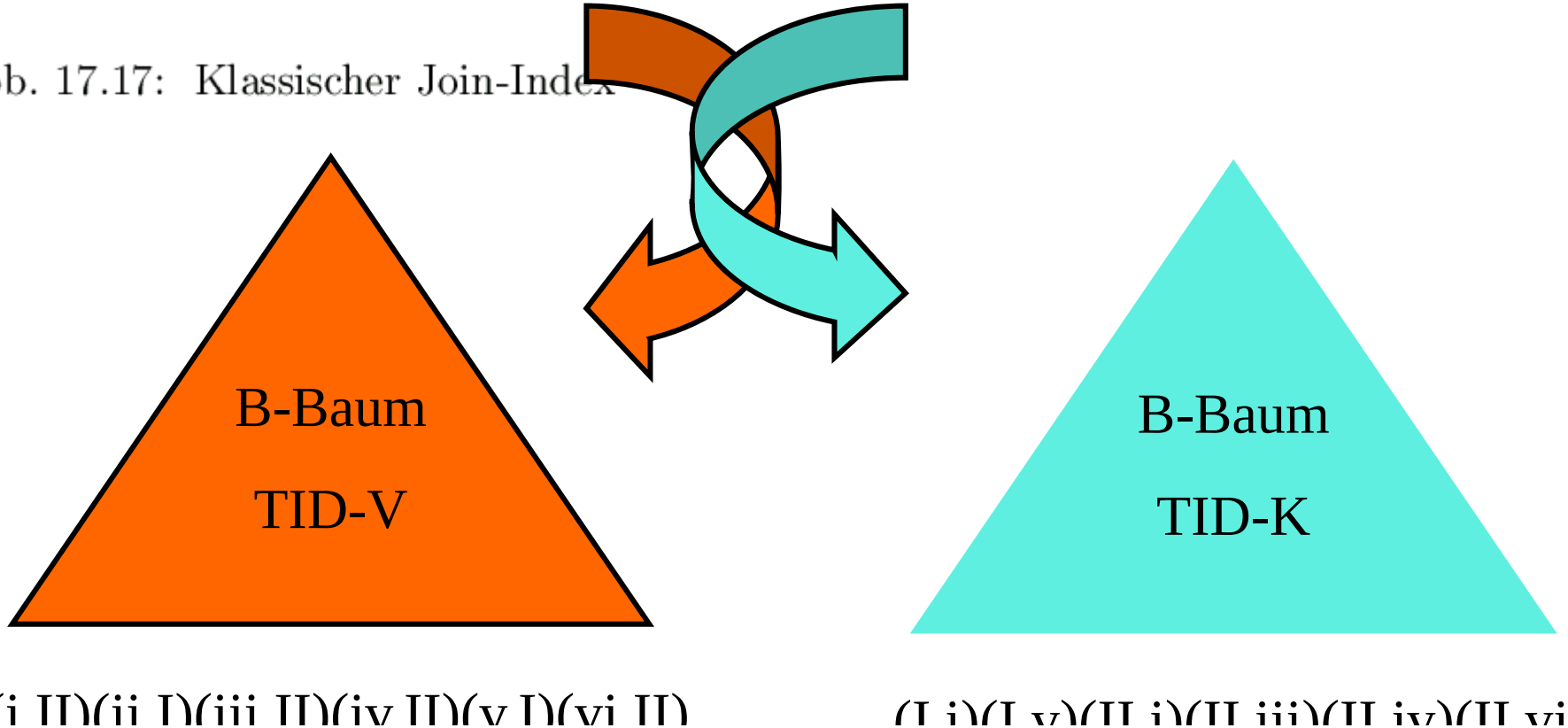
Kunden		
TID	KundenNr	...
I	4711	...
II	007	...
III	...	...
...	...	...

Verkäufe		
TID	...	KundenNr
i	...	007
ii	...	4711
iii	...	007
iv	...	007
v	...	4711
vi	...	007
...	...	...

Join-Index	
TID-V	TID-K
i	*II*
ii	*I*
iii	*II*
iv	*II*
v	*I*
vi	*II*
...	...

Kunden		
TID	KundenNr	...
I	4711	...
II	007	...
III	...	...
...	...	...

bb. 17.17: Klassischer Join-Index

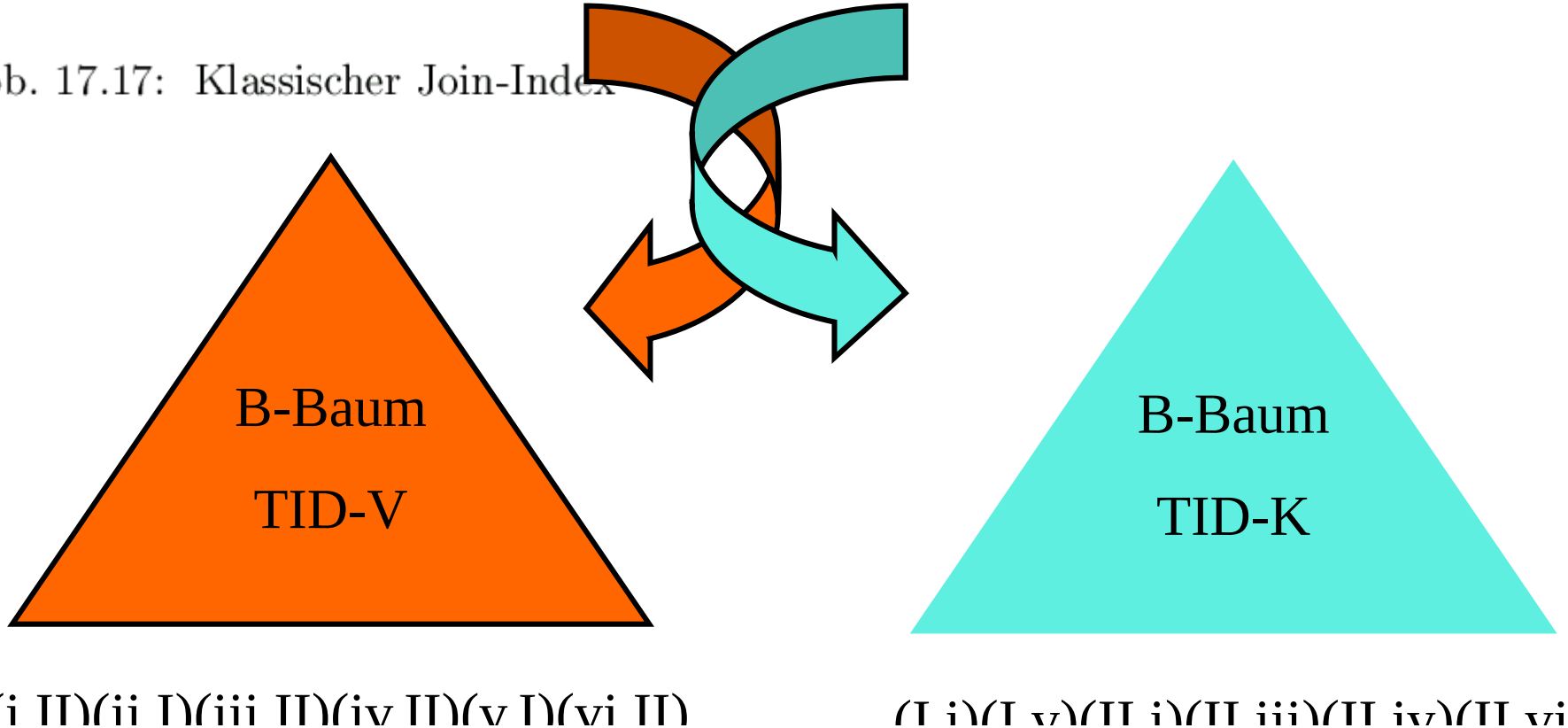


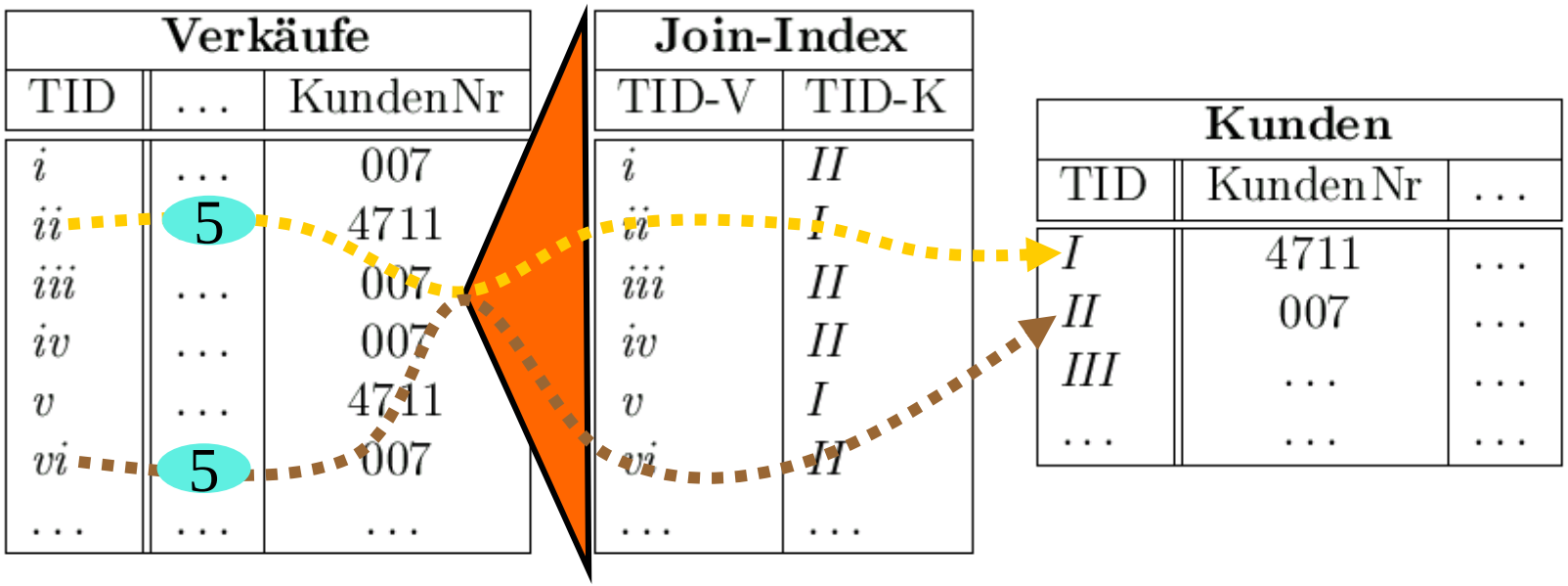
Verkäufe		
TID	...	KundenNr
i	...	007
ii	...	4711
iii	...	007
iv	...	007
v	...	4711
vi	...	007
...	...	...

Join-Index	
TID-V	TID-K
i	*II*
ii	*I*
iii	*II*
iv	*II*
v	*I*
vi	*II*
...	...

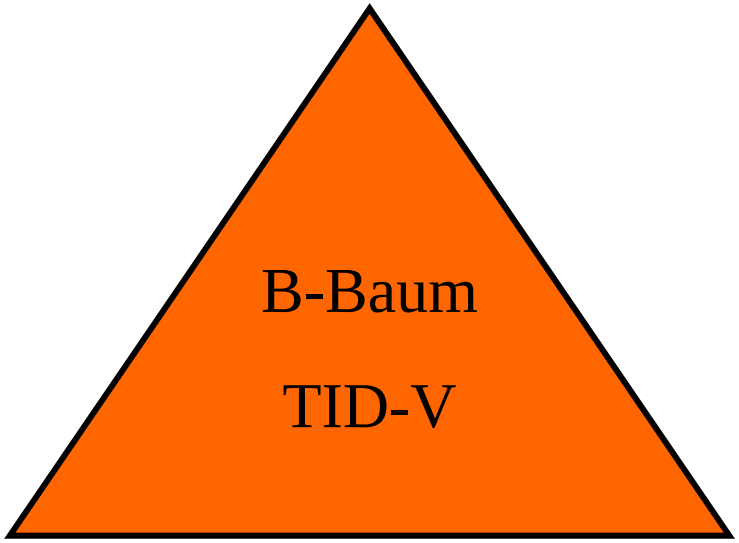
Kunden		
TID	KundenNr	...
I	4711	...
II	007	...
III	...	...
...	...	...

bb. 17.17: Klassischer Join-Index



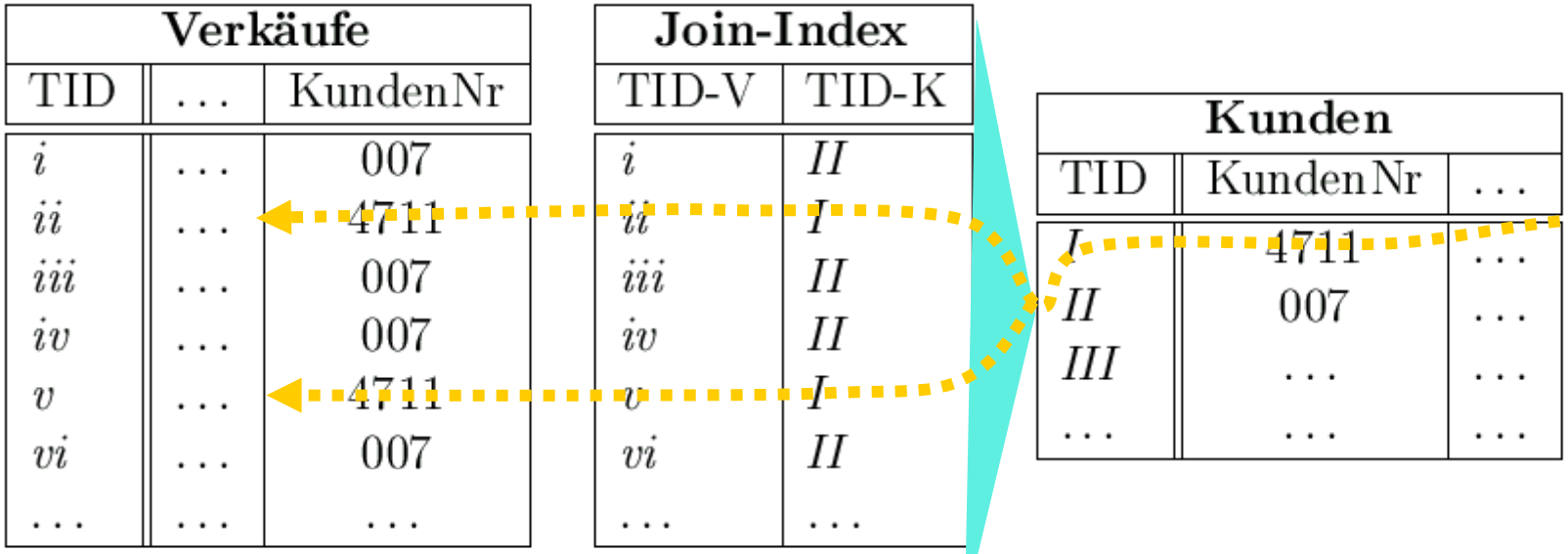


bb. 17.17: Klassischer Join-Index



(i,II)(ii,I)(iii,II)(iv,II)(v,I)(vi,II)

```
Select k.*
From Verkäufe v, Kunden k
Where v.ProduktID = 5 And
      v.KundenNr = k.KundenNr
```



bb. 17.17: Klassischer Join-Index

B-Baum

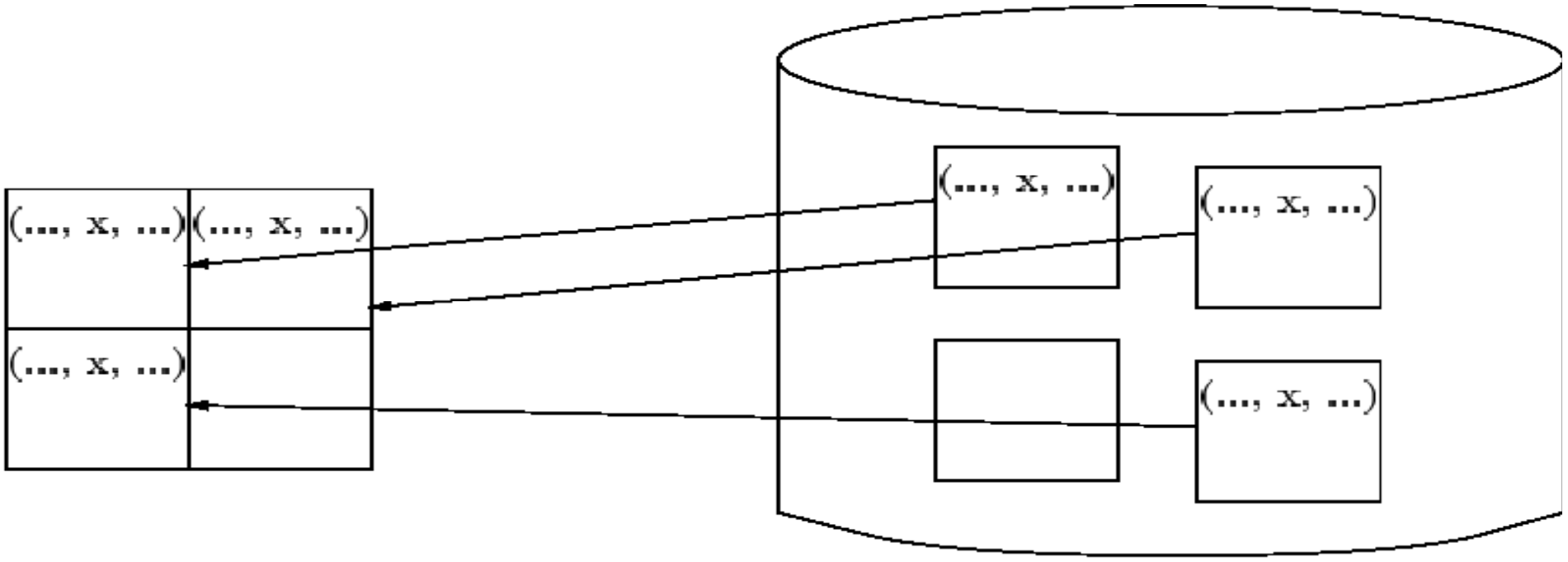
TID-K

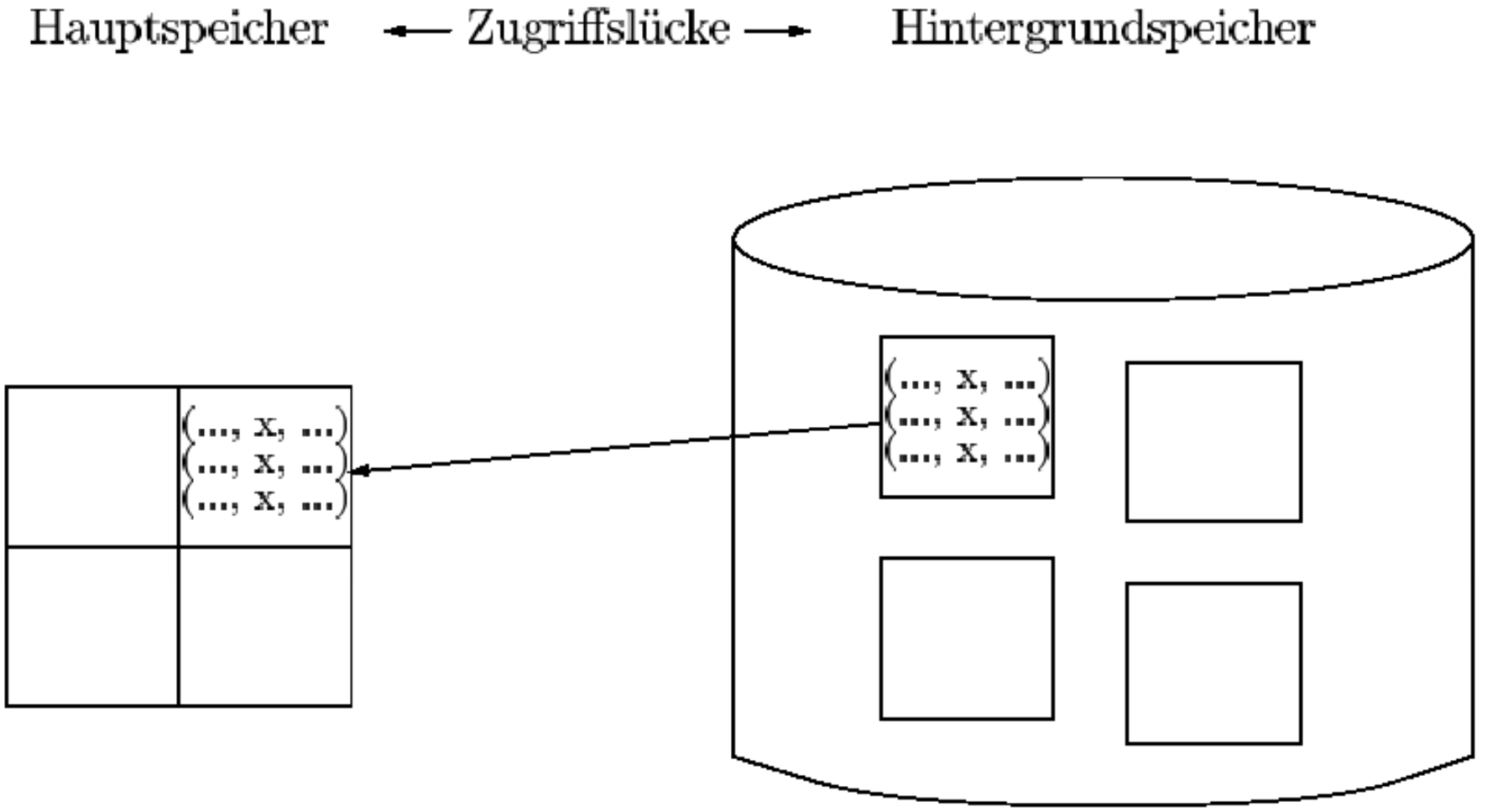
```
Select v.*
From Verkäufe v, Kunden k
Where k.KundenNr = 4711 and
      v.KundenNr = k.KundenNr
```

(I,i)(I,v)(II,i)(II,iii)(II,iv)(II,vi)...

Objektballung / Clustering logisch verwandter Daten

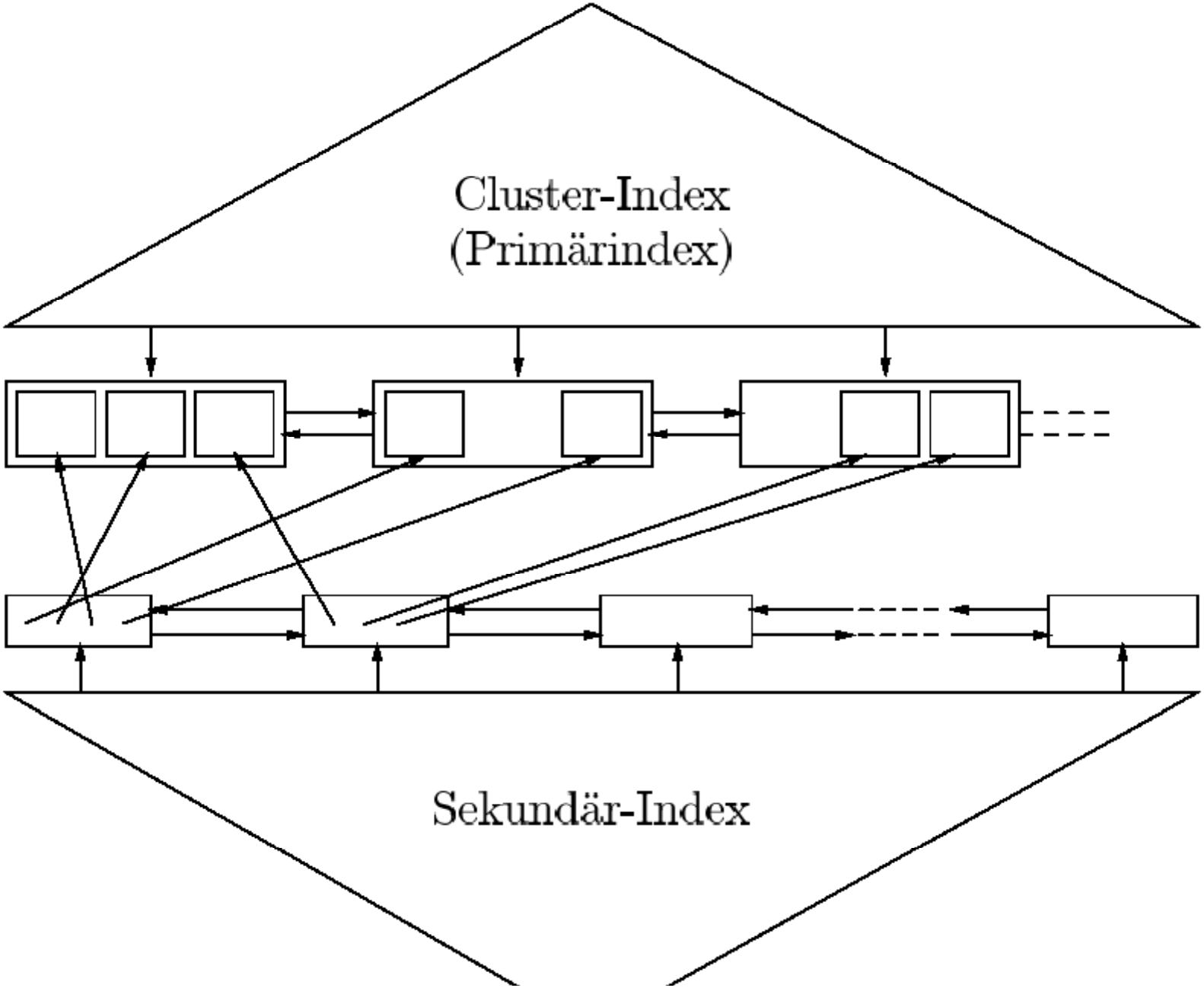
select $*$
from R
where $A = x$;




Hauptspeicher
Zugriffslücke
Hintergrundspeicher
(..., x, ...)
(..., x, ...)
(..., x, ...)
(..., x, ...)
(..., x, ...)
(..., x, ...)

202

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
```

Indexe in SQL

Create index SemsterInd
on Studenten
(Semester)

drop index SemsterInd