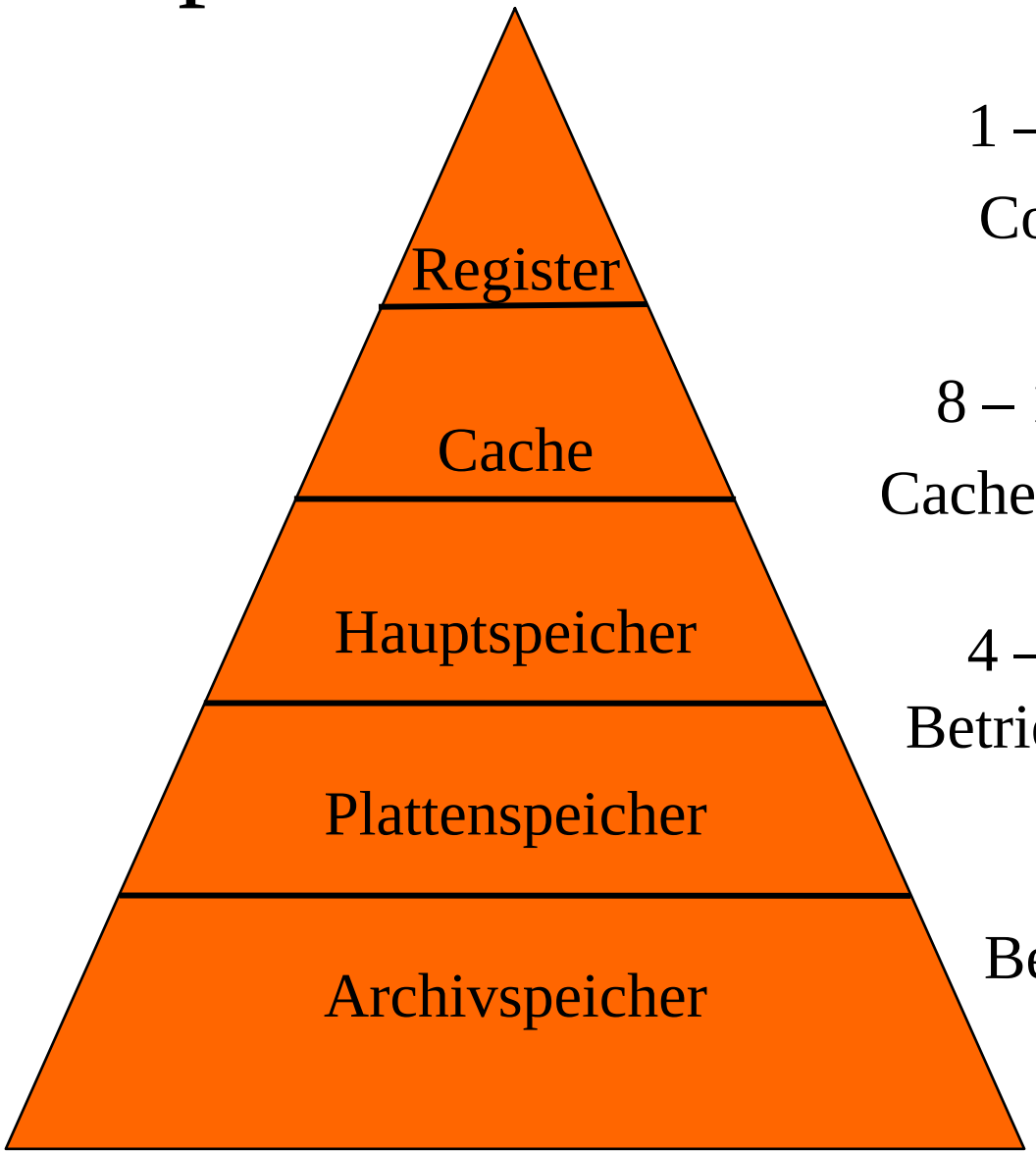
Physische Datenorganisat

- Speicherhierarchie
- Hintergrundspeicher / RAID
- B-Bäume
- Hashing
- (R-Bäume)
- Objektballung
- Indexe in SQL

Kapitel 7

Überblick: Speicherhierarchie



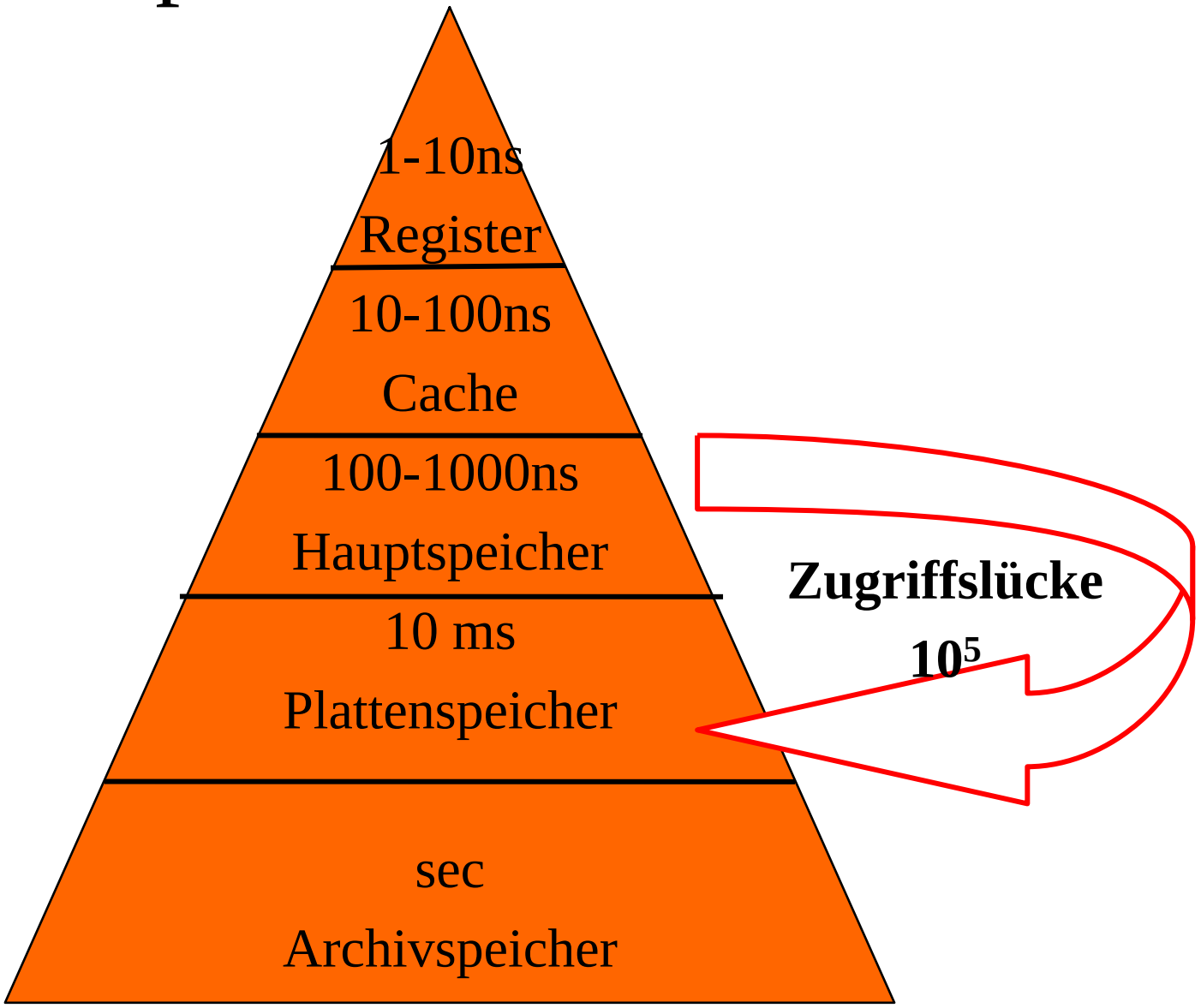
1 – 8 Byte
Compiler

8 – 128 Byte
Cache-Controller

4 – 64 KB
Betriebssystem

Benutzer

Überblick: Speicherhierarchie



Überblick: Speicherhierarchie

Literaturrecherche

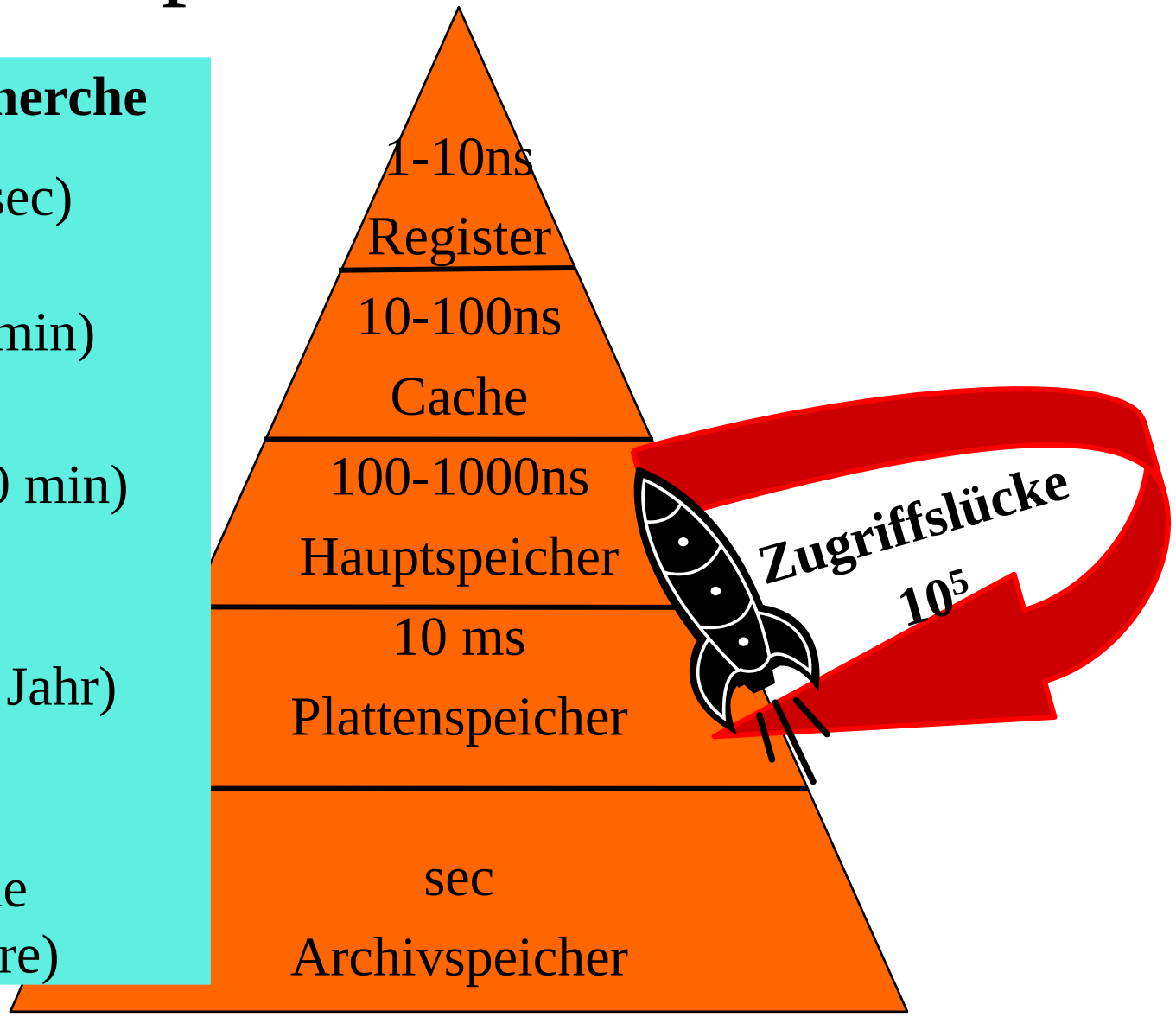
Kopf (30 sec)

Internet (5 min)

Uni-Bibl. (50 min)

Fernleihe (1 Jahr)

Archäologie (1000 Jahre)



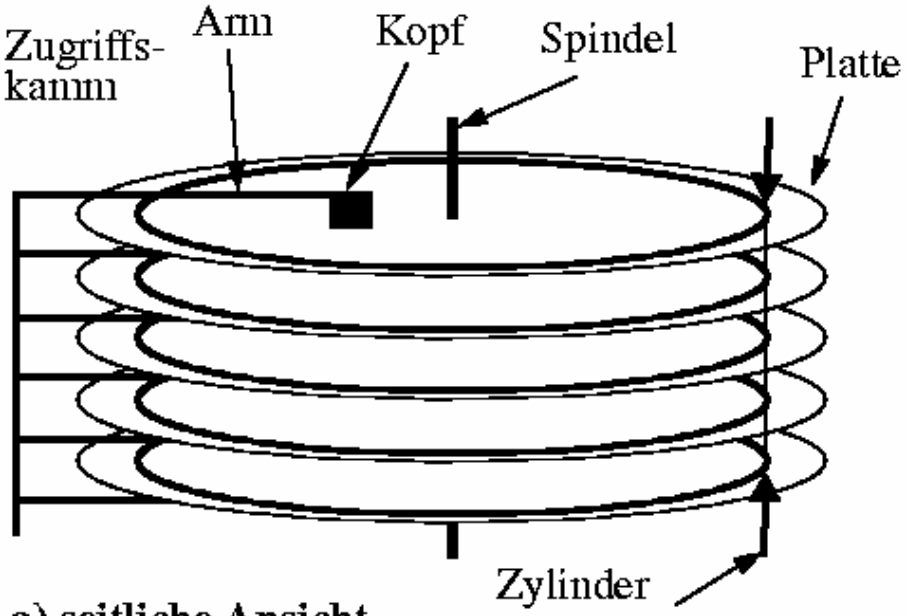
Magnetplattenspeicher

Aufbau

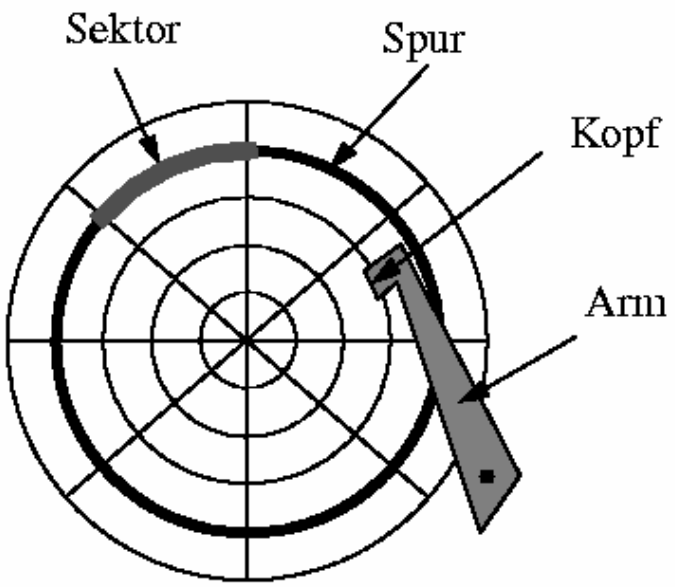
- mehrere gleichförmig rotierende Platten, für jede Plattenoberfläche ein Schreib-/Lesekopf
- jede Plattenoberfläche ist eingeteilt in Spuren
- die Spuren sind formatiert als Sektoren fester Größe (Slots)
- Sektoren (typischerweise 1 - 8 KB) sind die kleinste Schreib-/Leseeinheit auf einer Platte

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)

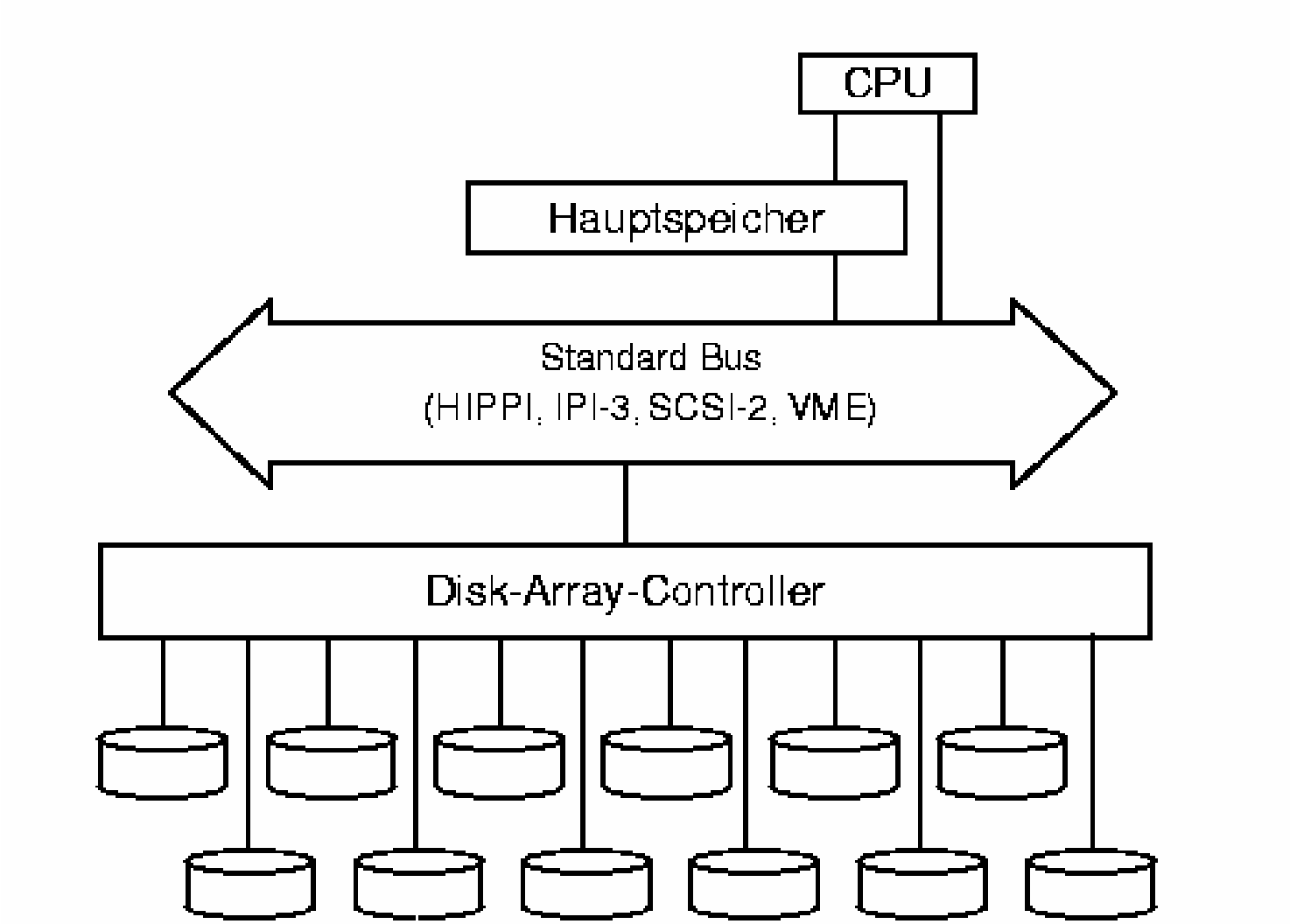
Lesen von Daten von der Platte

- 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 I/O

- 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 → 8.3 s
- Chained I/O
  - Einmal positionieren, dann „von der Platte kratzen“
  - ➔ 5 ms + 3ms + Transferzeit von 4 MB
  - ➔ 8ms + 300 ms → 0.38 s
- Also ist chained I/O **ein bis zwei Größenordnungen schneller** als random I/O.
- Ist bei Datenbank-Algorithmen unbedingt zu 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

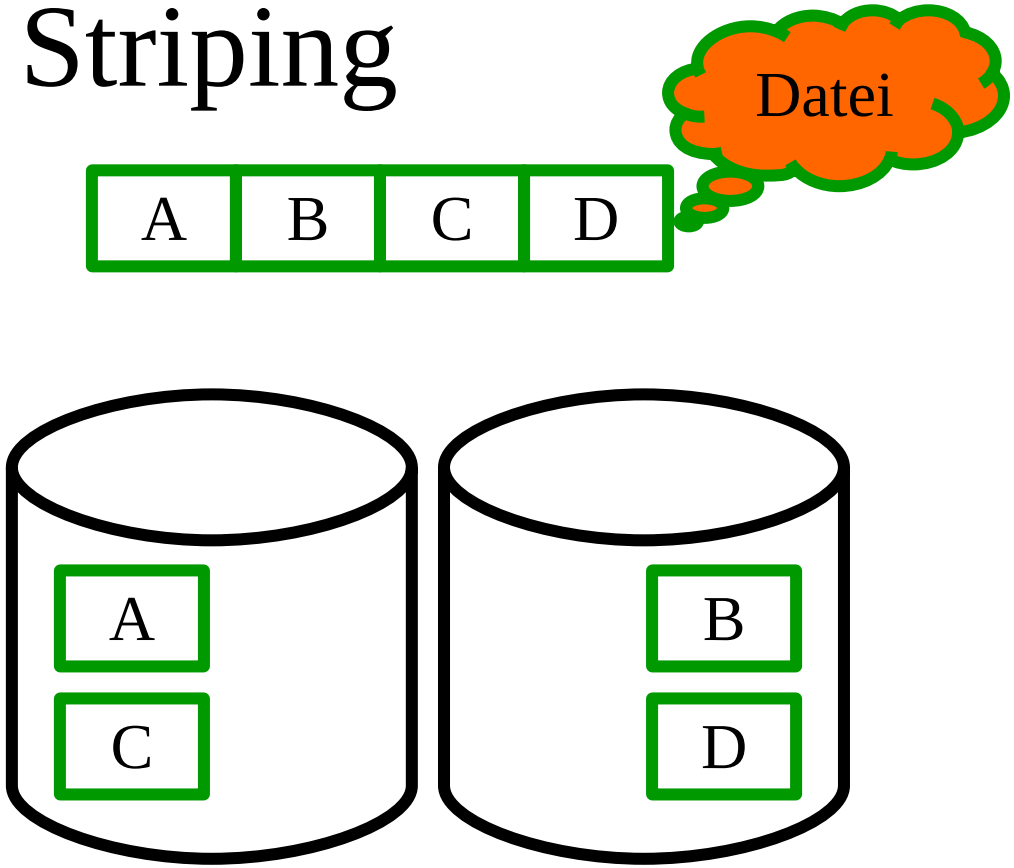
- Mean Time To Failure (MTTF): Erwartungswert für die Zeit (von der Inbetriebnahme) bis zum Ausfall einer Platte
- Mean Time To Repair (MTTR): Erwartungswert für die Zeit zur Ersetzung der Platte und der Rekonstruktion der Daten
- Mean Time To Data Loss (MTTDL): Erwartungswert für die Zeit bis zu einem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

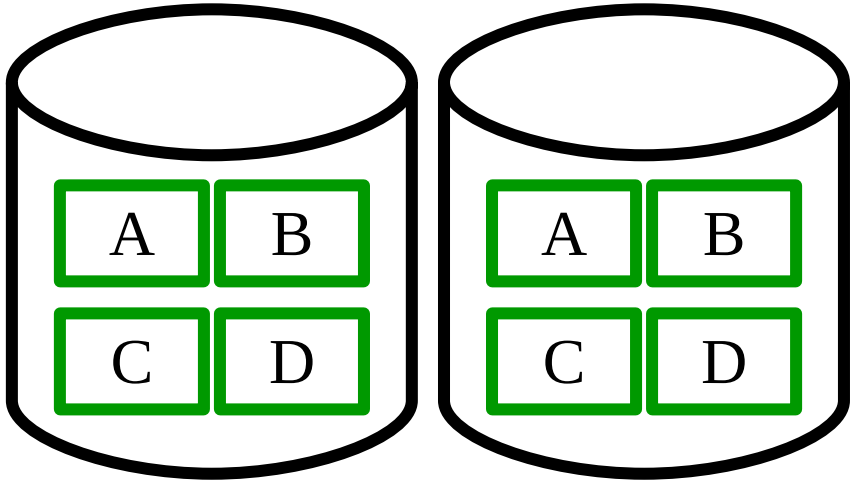
- durch Replikation der Daten (z. B. Spiegelplatten) - RAID1
- durch zusätzlich zu den Daten gespeicherte Error-Correcting-Codes (ECCs), z.B. Paritätsbits (RAID-4, RAID-5)

RAID 0: Striping



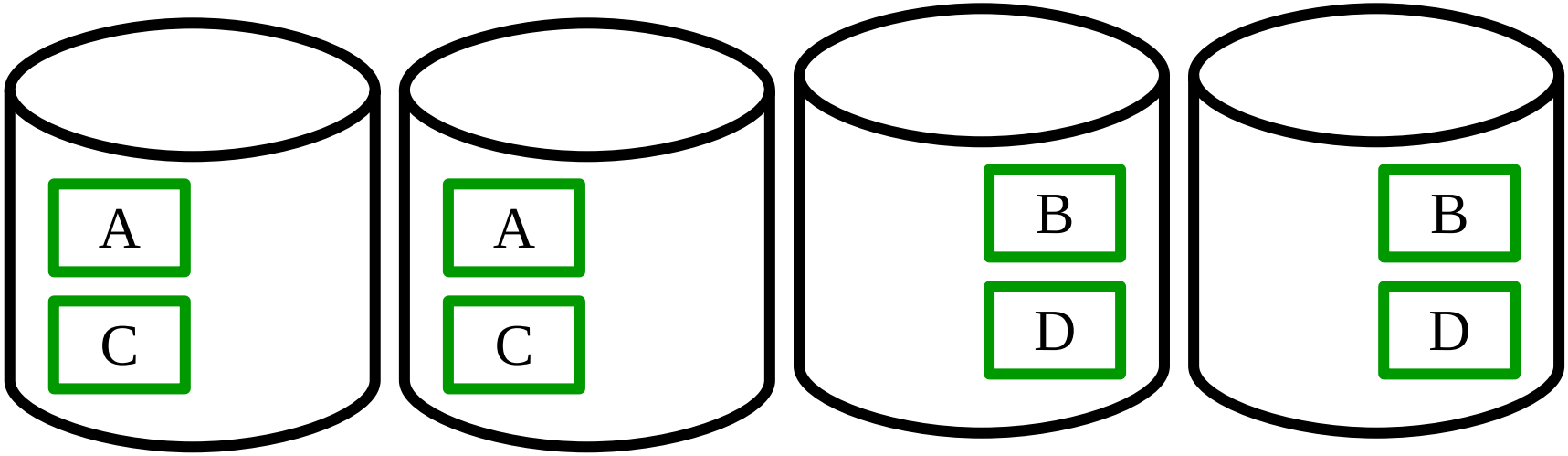
- Lastbalancierung, wenn alle Blöcke mit gleicher Häufigkeit gelesen/geschrieben werden
- Doppelte Bandbreite beim sequentiellen Lesen der Datei bestehend aus den Blöcken ABCD...
- Aber: Datenverlust wird immer wahrscheinlicher, je mehr Platten man verwendet (Stripingbreite = Anzahl der Platten, hier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

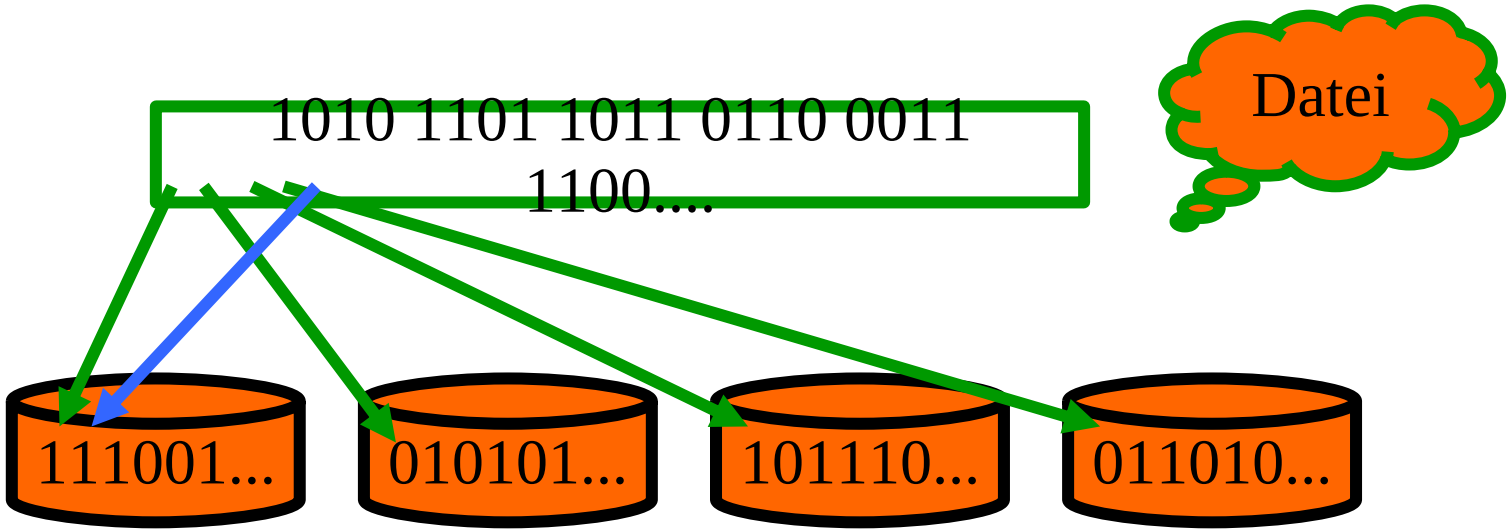
RAID 0+1: Striping und Spiegelung



- 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.

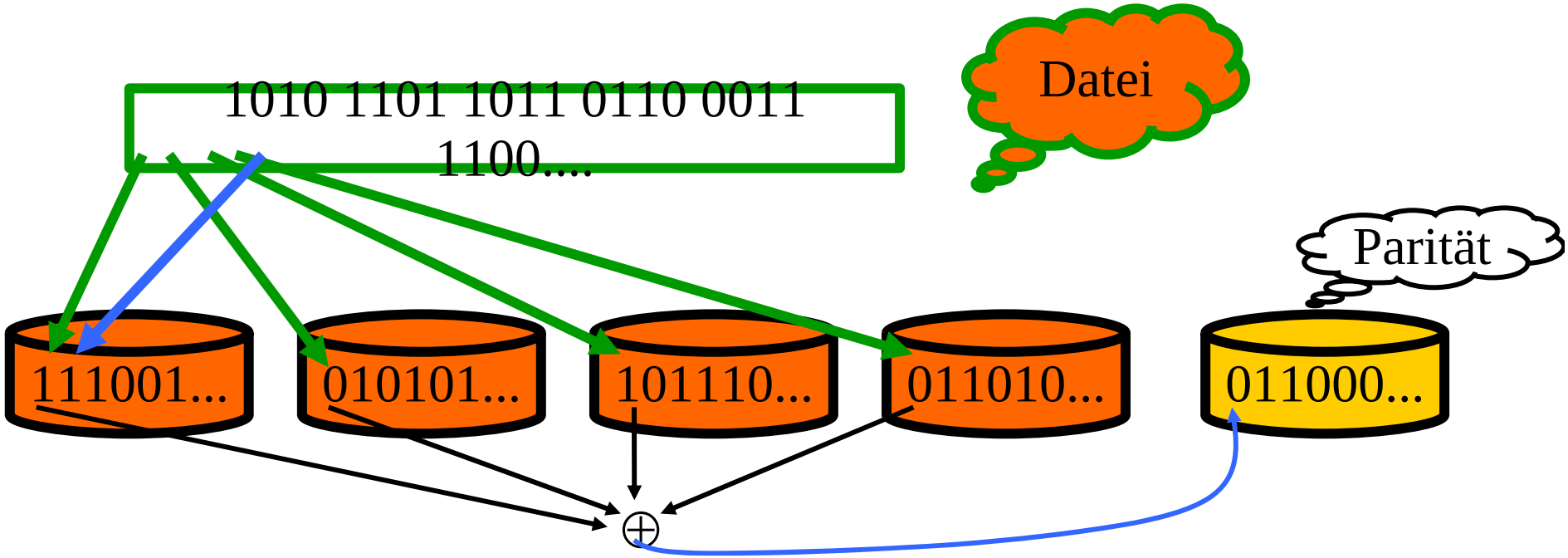
RAID 2: Striping auf Bit-Ebene

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



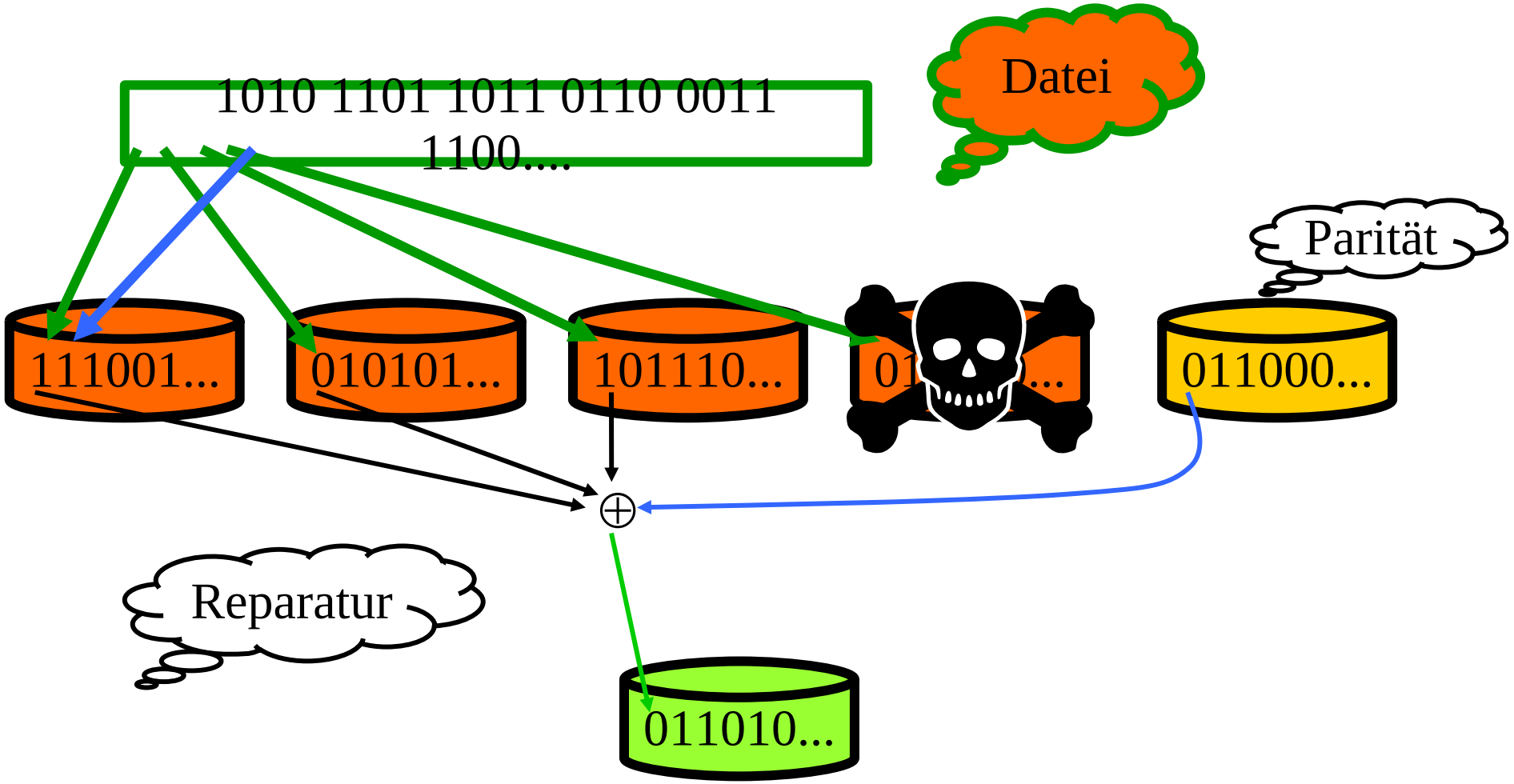
- Es werden zusätzlich auf einer Platte noch Fehlererkennungs- und Korrekturcodes gespeichert
- In der Praxis nicht eingesetzt, da Platten sowieso schon Fehlererkennungscodes verwalten

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

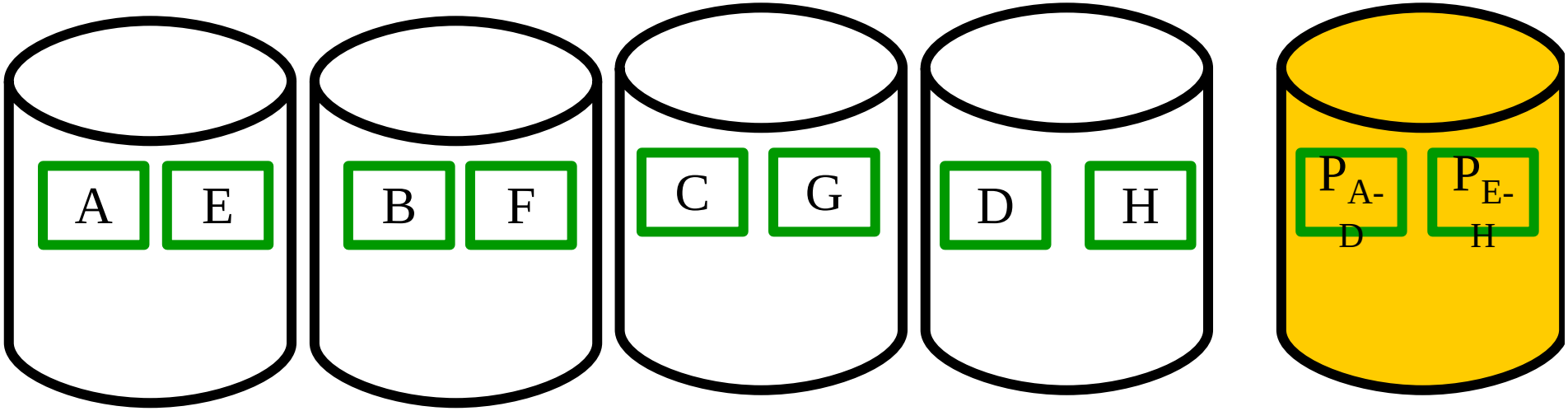


- Das Striping wird auf Bit- (oder Byte-) Ebene durchgeführt.
- Es wird auf einer Platte noch die Parität der anderen Platten gespeichert.
  Parität = bitweise xor (⊕)
- Dadurch ist der Ausfall einer Platte zu kompensieren.
- Das Lesen eines Blocks erfordert den Zugriff auf alle Platten:
  - Verschwendung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

RAID 3: Plattenausfall

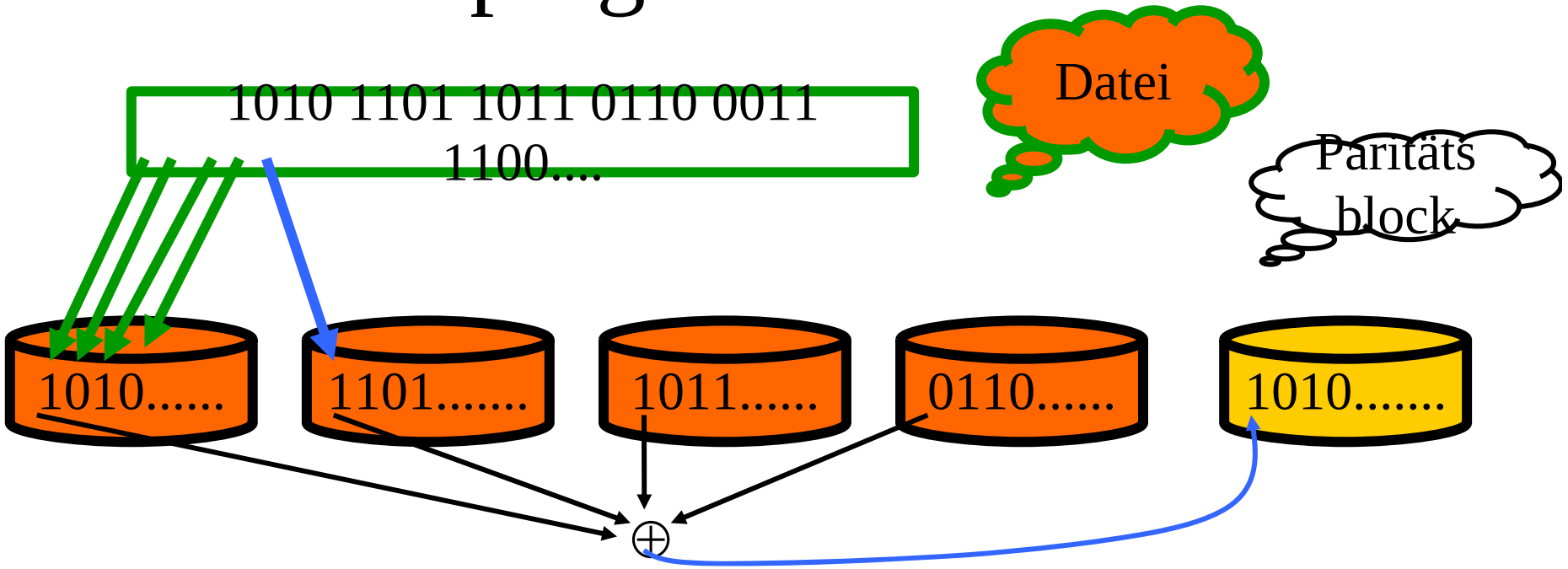


RAID 4: Striping von Blocken



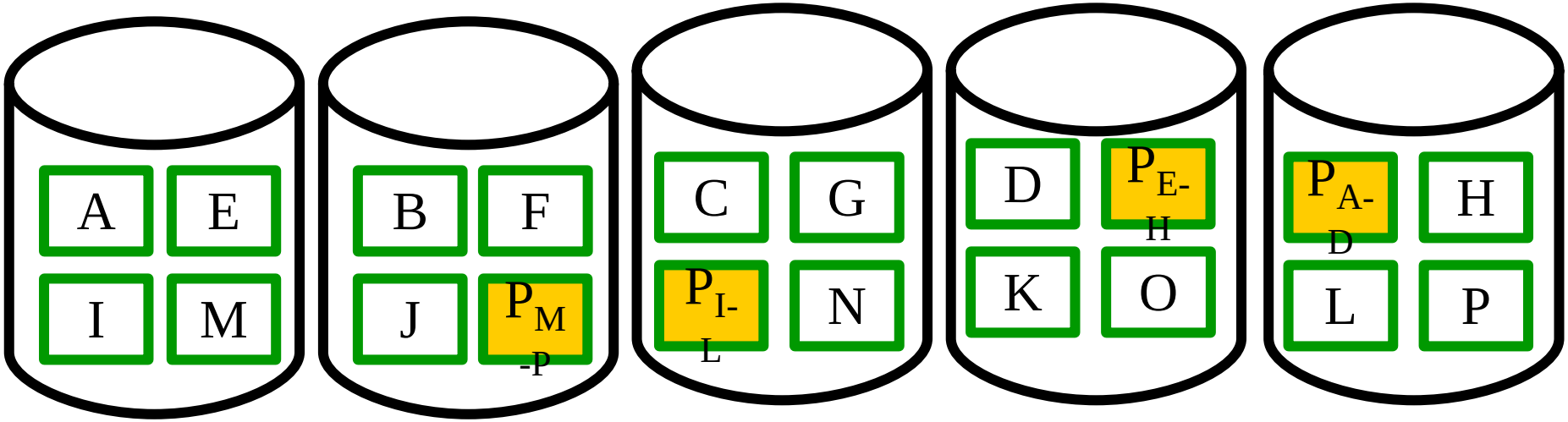
- Bessere Lastbalancierung als bei RAID 3
- Flaschenhals bildet die Paritätsplatte
- Bei jedem Schreiben muss darauf zugegriffen werden
  - Bei Modifikation von Block A zu A‘ wird die Parität $P_{A\text{-}D}$ wie folgt neu berechnet:
  - $P'_{A\text{-}D} := P_{A\text{-}D} \oplus A \oplus A'$
- D.h. bei 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 4: Striping von Blocken



- 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



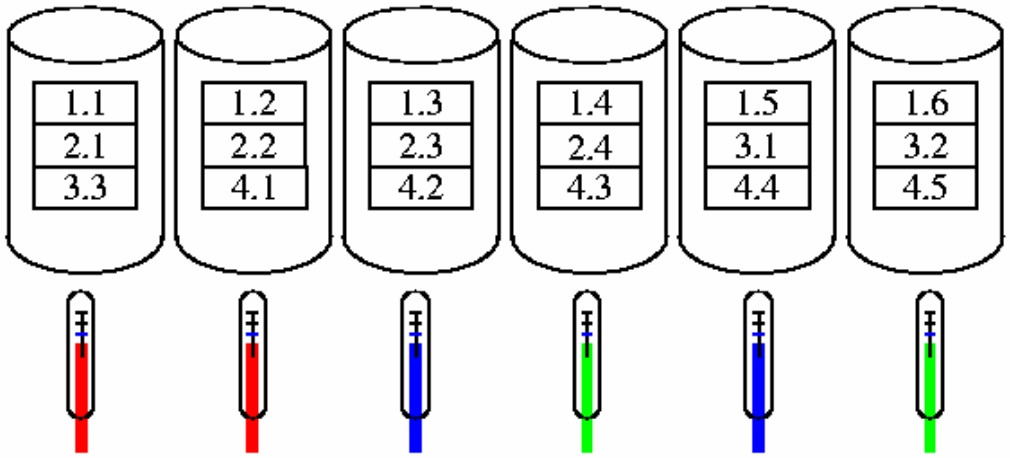
- Bessere Lastbalancierung als bei RAID 4
- die Paritätsplatte bildet jetzt keinen Flaschenhals mehr
- Wird in der Praxis häufig eingesetzt
- Guter Ausgleich zwischen Platzbedarf und Leistungsfähigkeit

Lastbalancierung bei der Blockabbildung auf die Platten

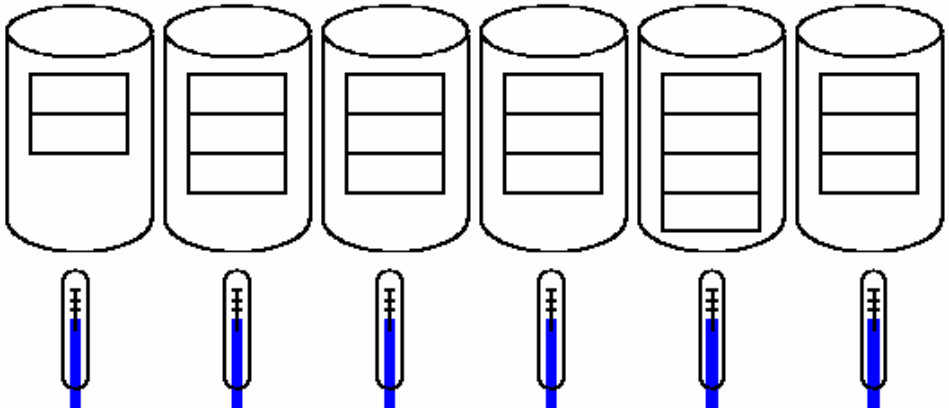
Vergleich von Greedy-Verfahren und Round-Robin-Allokation

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

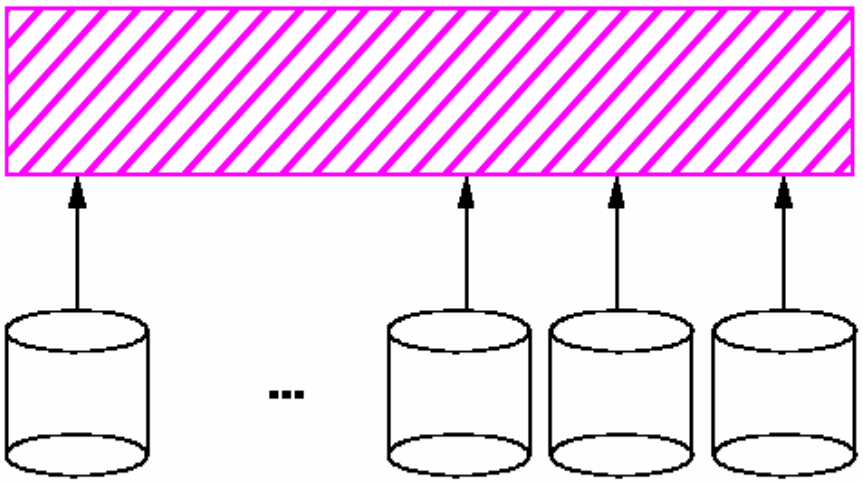


Parallelität bei Lese/Schreib-Aufträgen

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

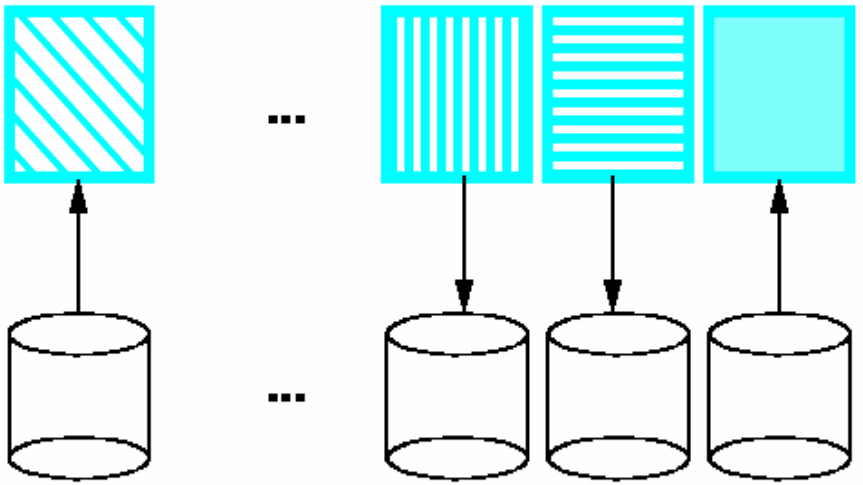
2 generelle Arten von E/A-Parallelität

Intra-E/A-Parallelität (Zugriffsparallelität)



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

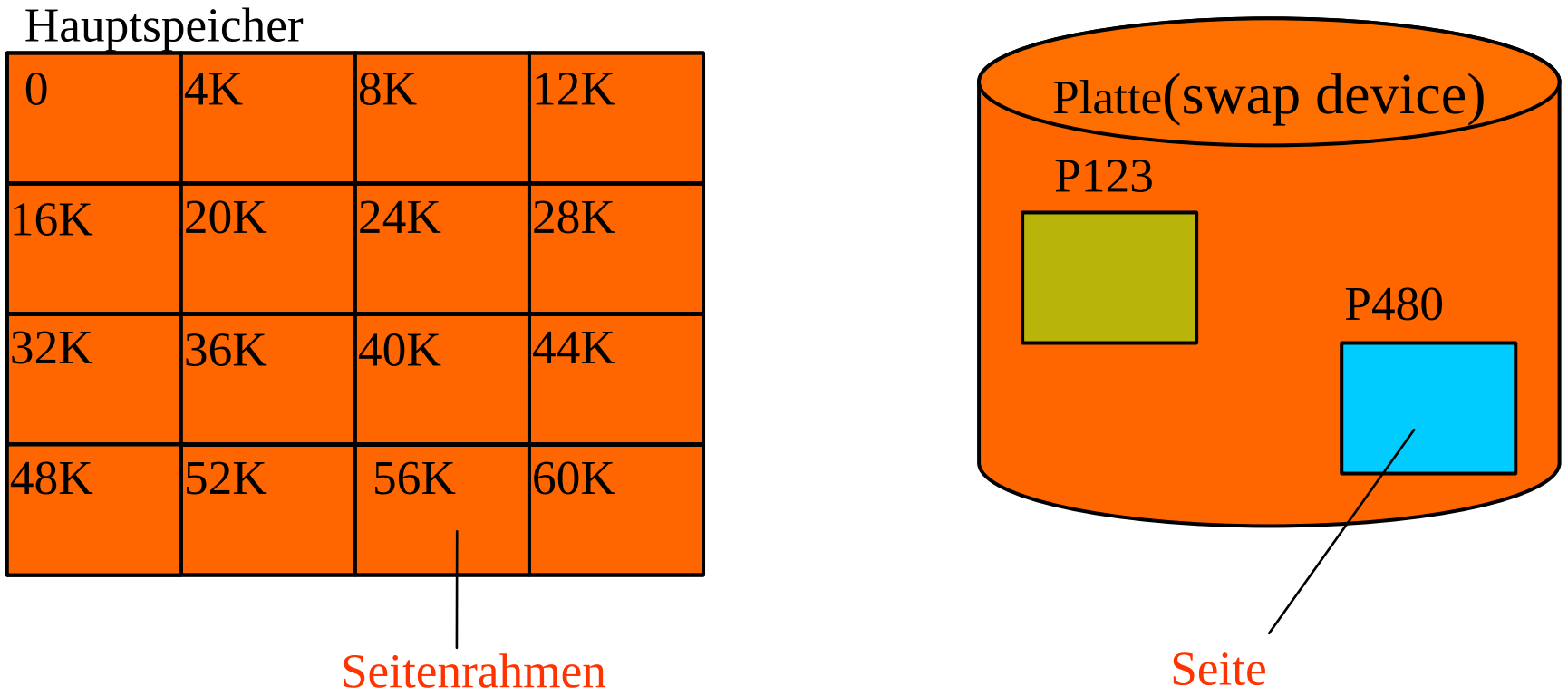
Inter-E/A-Parallelität (Auftragsparallelität)



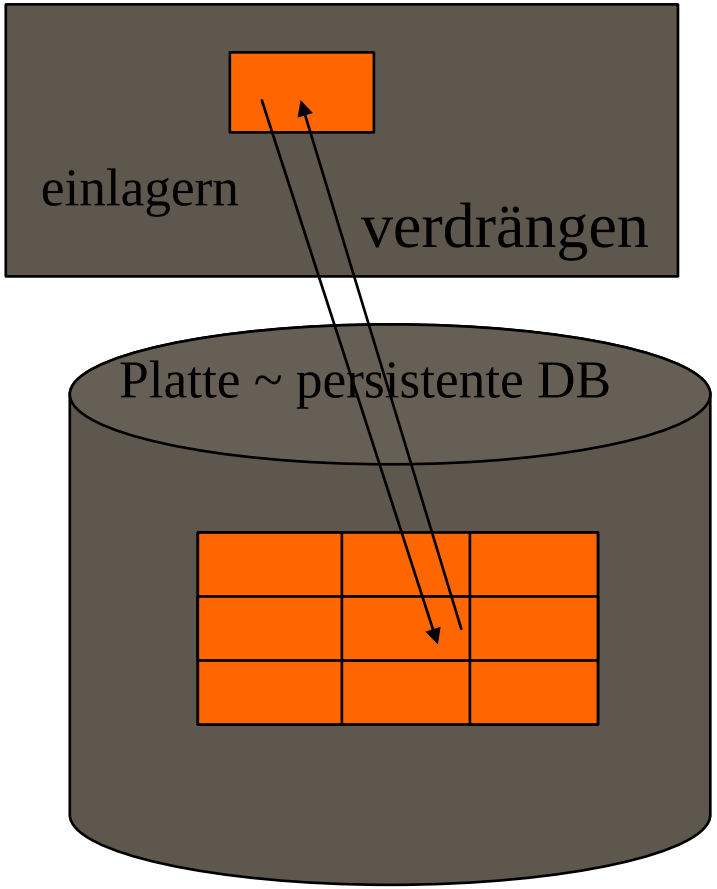
Mehrere unabhängige E/A-Aufträge können parallel ausgeführt werden, sofern die betreffenden Daten über verschiedene Platten verteilt sind

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



Systempuffer-Verwaltung



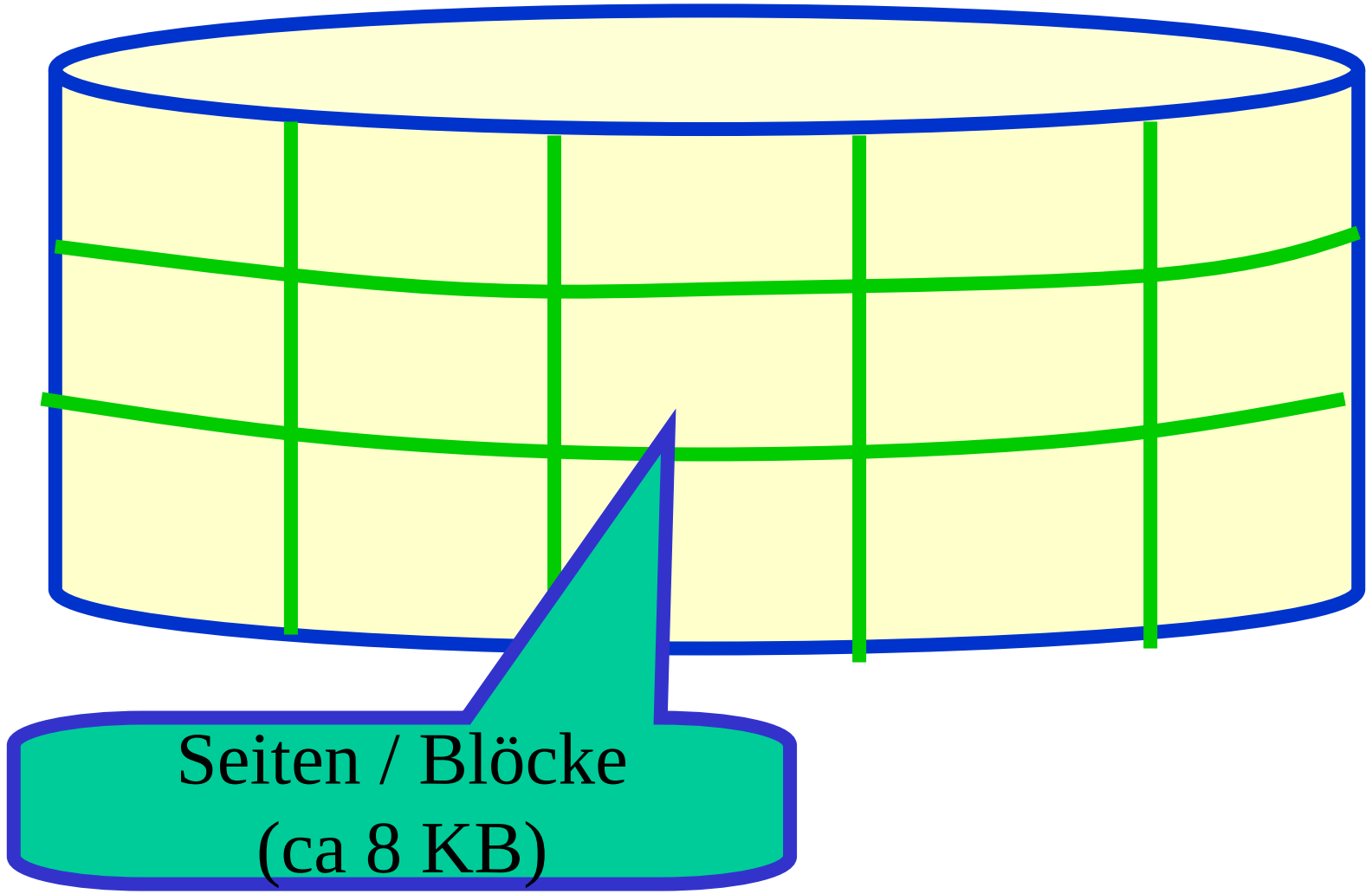
Hauptspeicher

Verdrängungsstrategien:

- Least-Recently-Used (LRU)
- First-in-first out (FIFO)
- Second Chance
- Zähler (simulierte Uhr)

(Ziel: Annäherung der Belady-Strategie: Seite, die am längsten nicht mehr benötigt wird, verdrängen)

Realisierungstechnik für Hintergrundspeicher-Adressen



Adressierung von Tupeln auf dem Hintergrundspeicher

Naiver Ansatz:

- jedes Tupel hat gleiche Länge *l*
- *i*. Tupel hat Position (*i* -1) x *l*

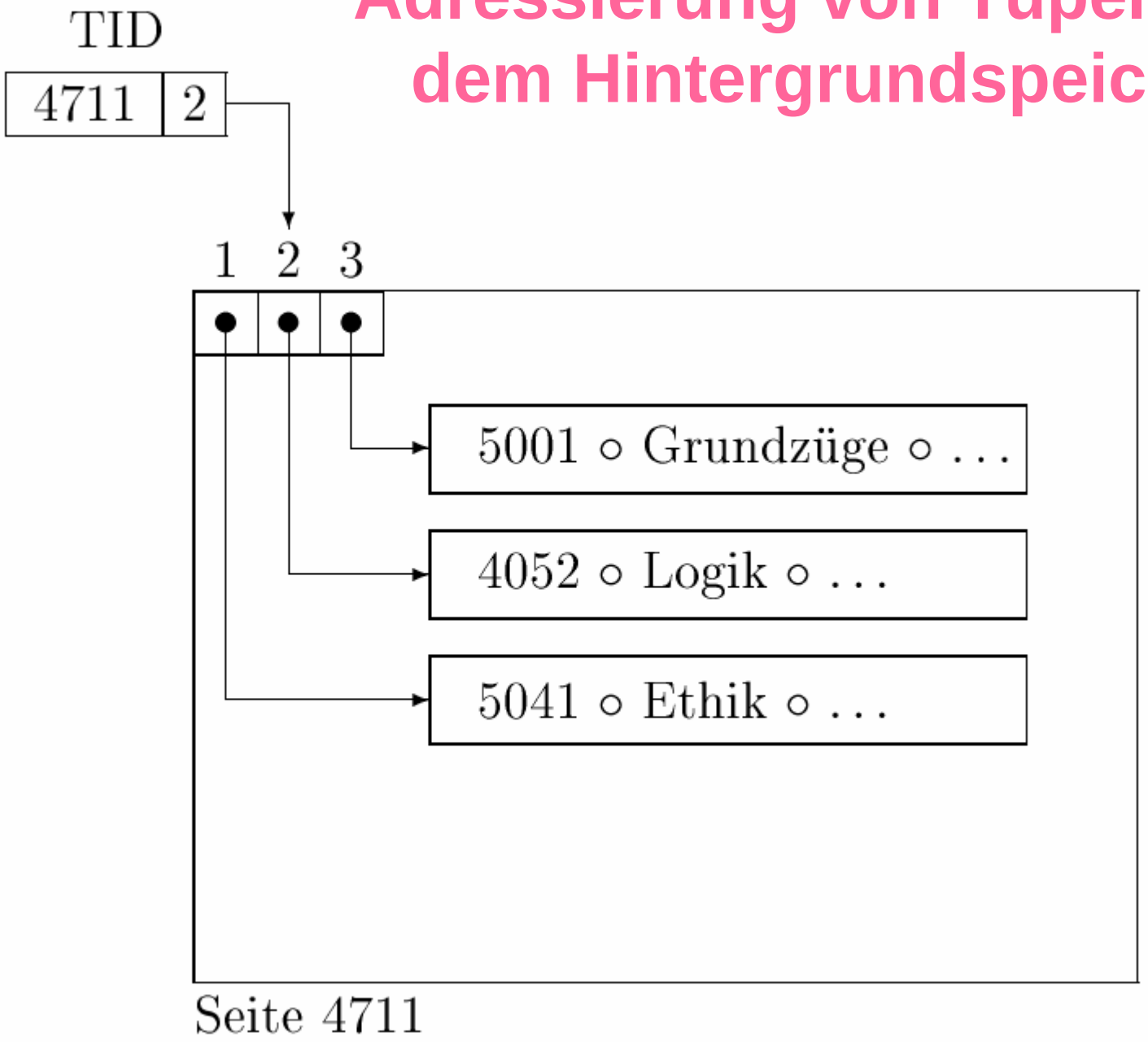
Komplexe Anforderungen verlangen flexiblere Adressierung:

- variable Feldlängen
- Logdateien
- mehrfacher Bezug auf Werte

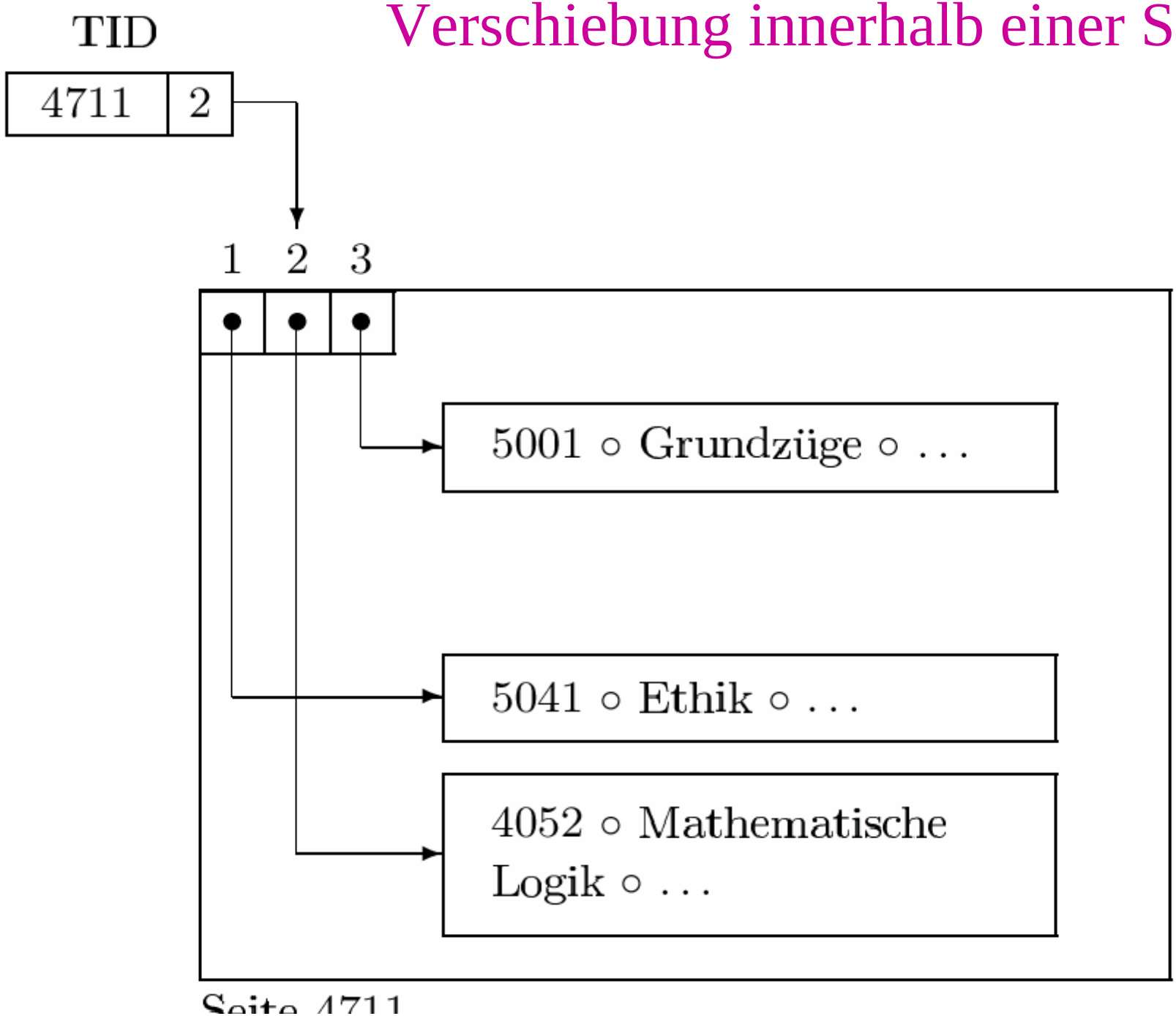
Lösung:

- nur die Seiten werden nach obigem Schema addressiert.
- innerhalb einer Seite werden Zeiger verwendet.

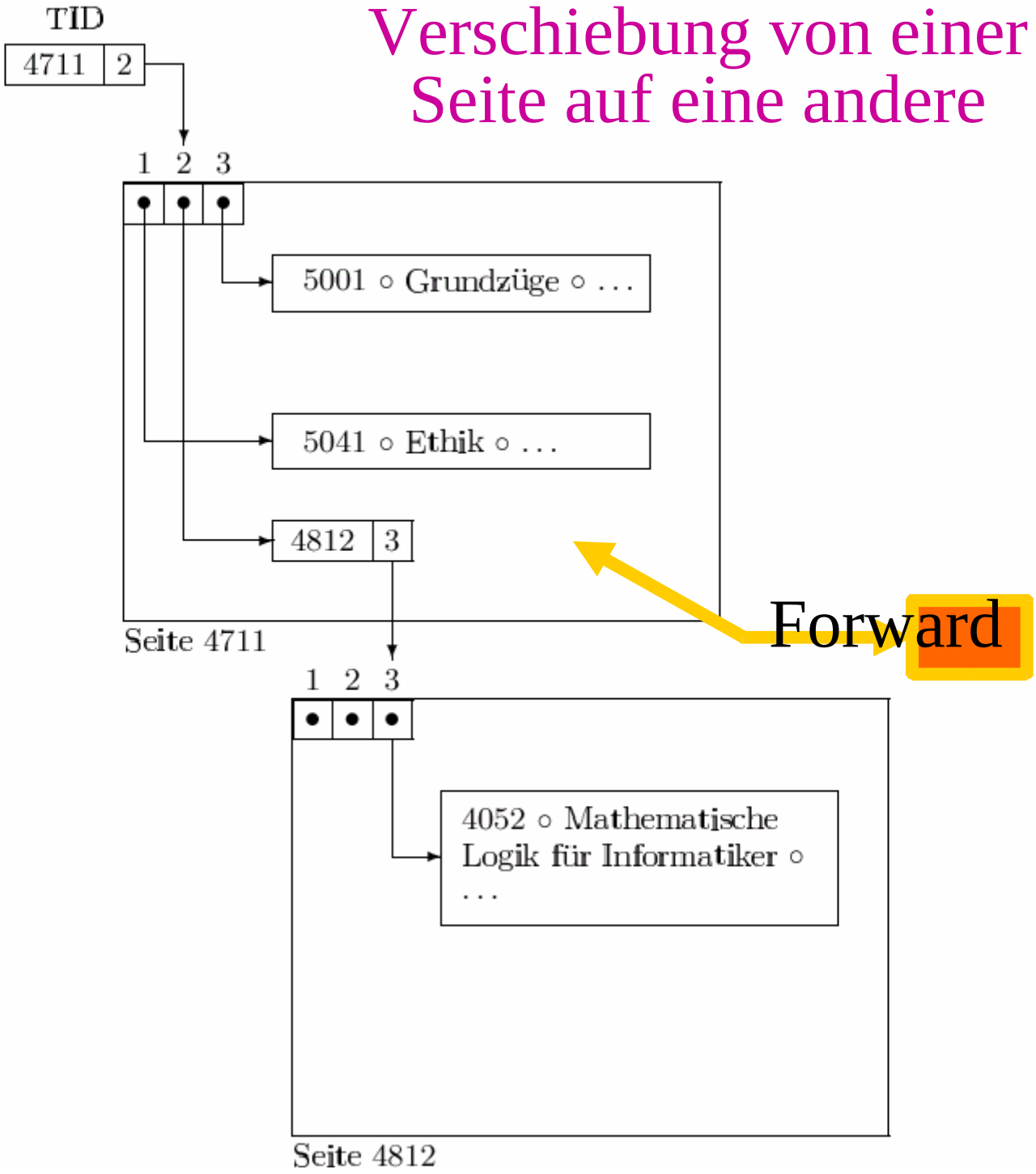
Adressierung von Tupeln auf dem Hintergrundspeicher



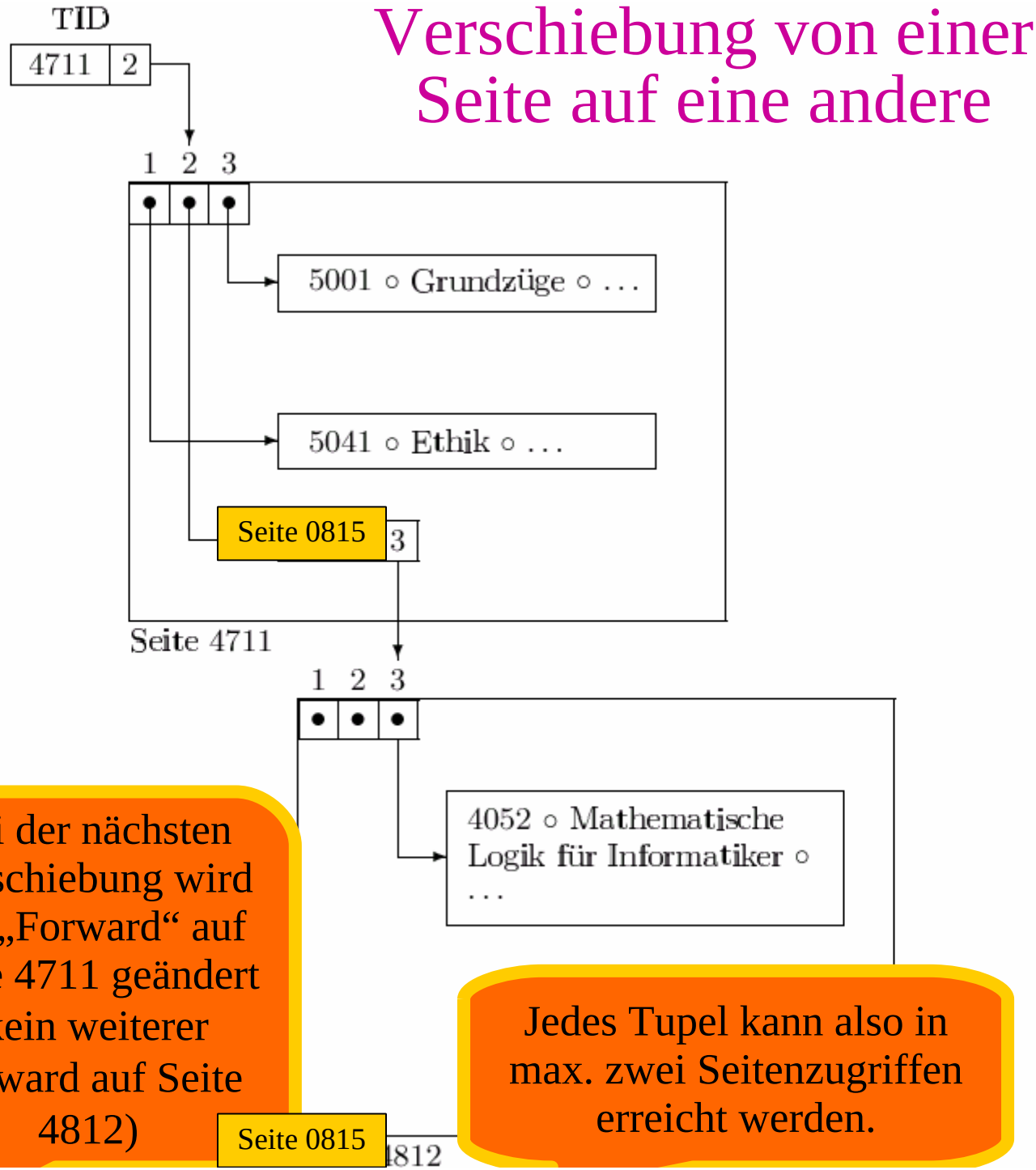
Verschiebung innerhalb einer Seite



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 weiterer Forward auf Seite 4812)

Jedes Tupel kann also in max. zwei Seitenzugriffen erreicht werden.

Indexstrukturen

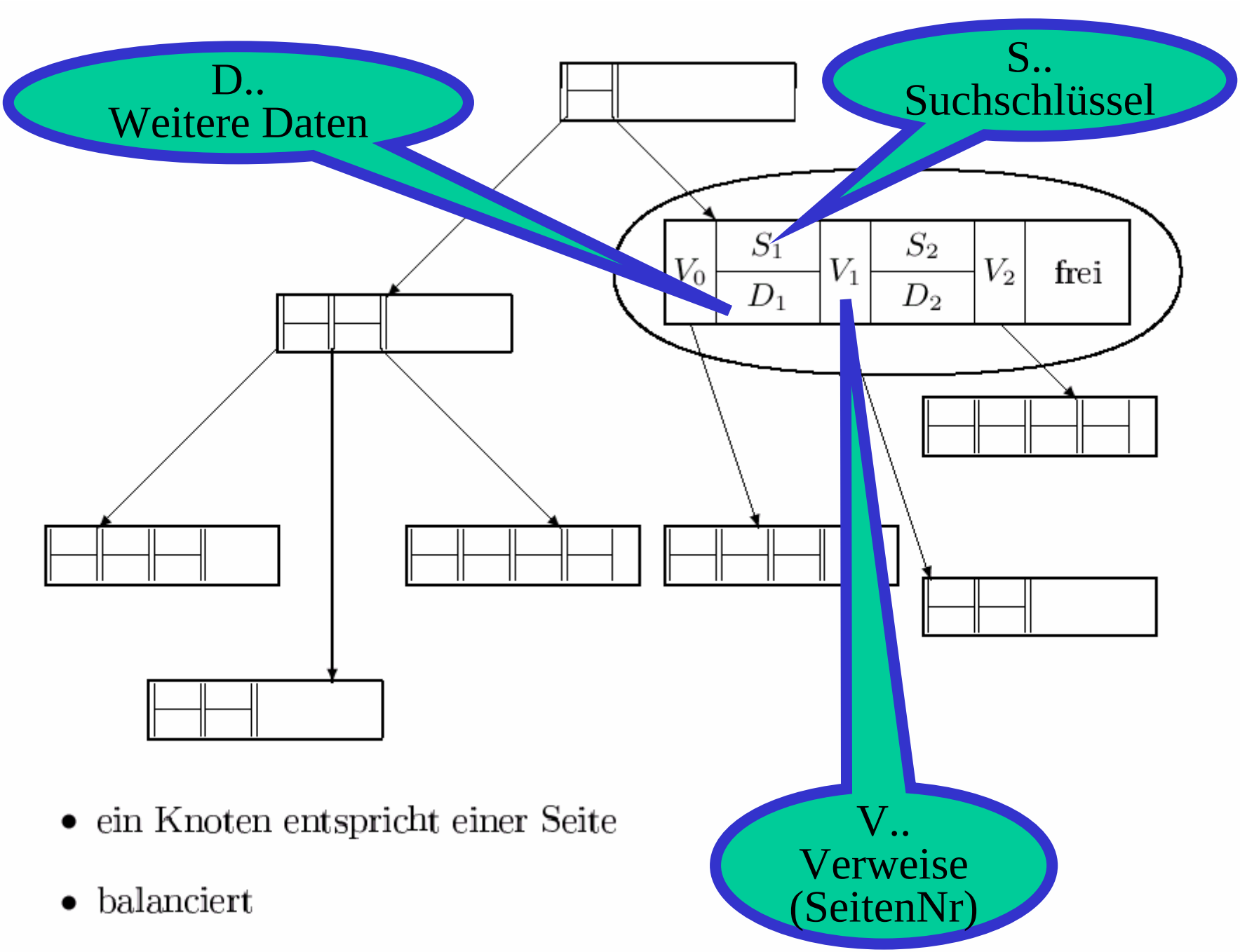
- Oft werden bei Anfragen nur wenige Tupel benötigt.
- Mit einem Index muss nicht die ganze Datei durchsucht werden.

- Wir betrachten hier:
  - B-Bäume
  - Hashing

- Ein Index ist immer mit Wartungsaufwand und Speicherverbrauch verbunden.
- Es ist in der konkreten Anwendung abzuwägen, ob sich dieser Aufwand rechnet.

B-Bäume

- Binärbäume sind gute Suchstrukturen, aber nur für Verwendung im Hauptspeicher geeignet.
- B-Baum = balancierter Mehrwege-Suchbaum für den Hintergrundspeicher
- Ein Knoten entspricht einer Seite im Hintergrundspeicher.
- Die maximale Anzahl der Seitenzugriffe wird durch die Höhe des Baums begrenzt.
- Aufgrund der Balancierung sind alle Blätter gleich weit von der Wurzel entfernt.

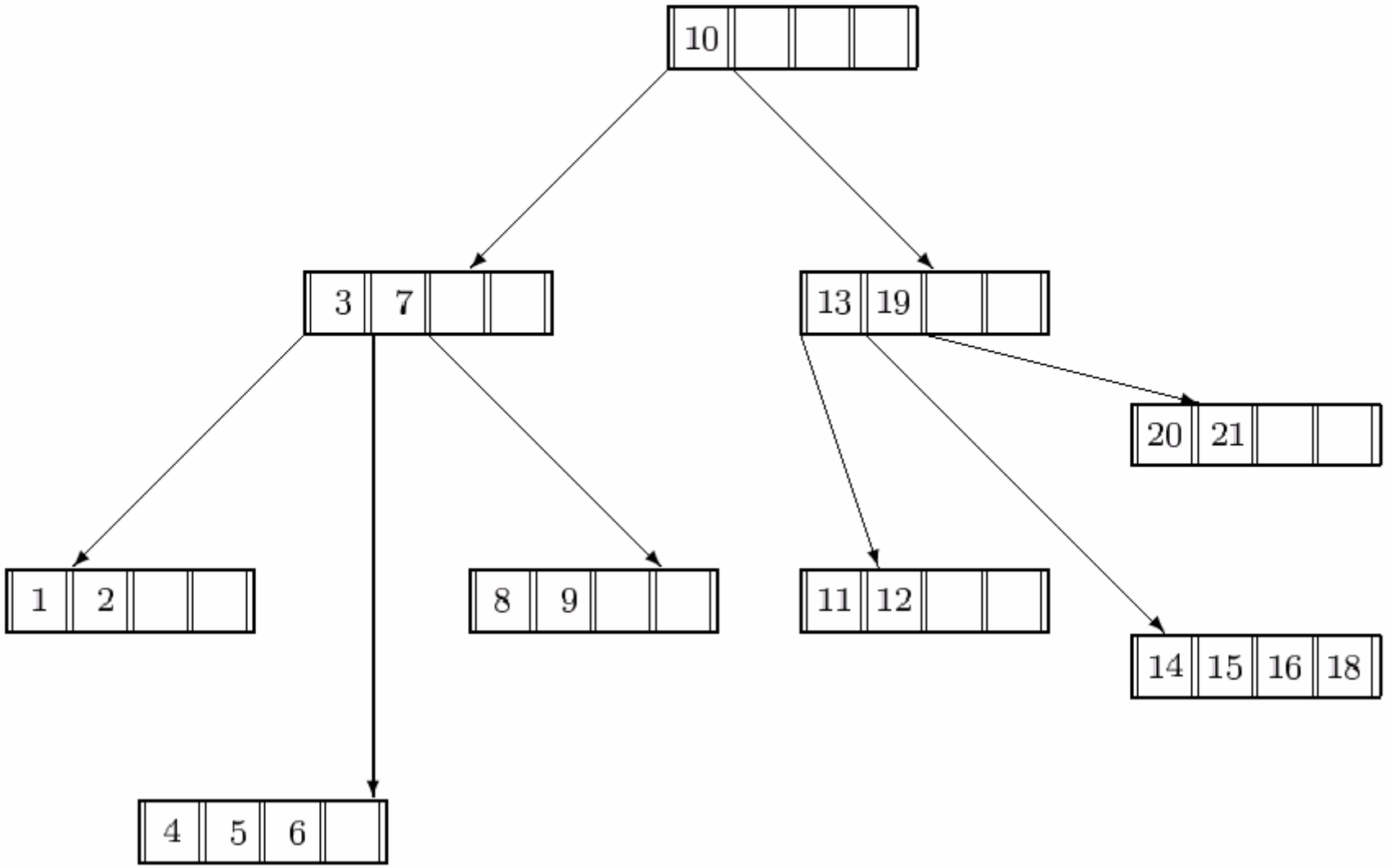


- ein Knoten entspricht einer Seite
- balanciert
- garantierte Auslastung $\geq 50\%$

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

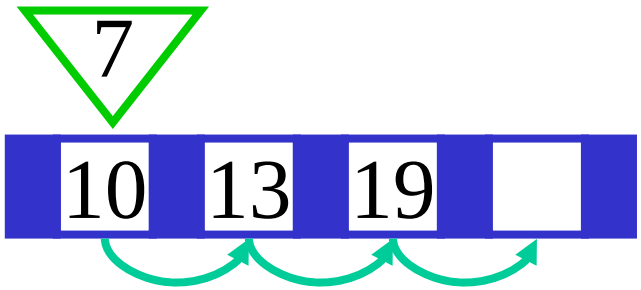
1. Führe eine Suche nach dem Schlüssel durch; diese endet (scheitert) an der Einfügestelle.

2. Füge den Schlüssel dort ein.

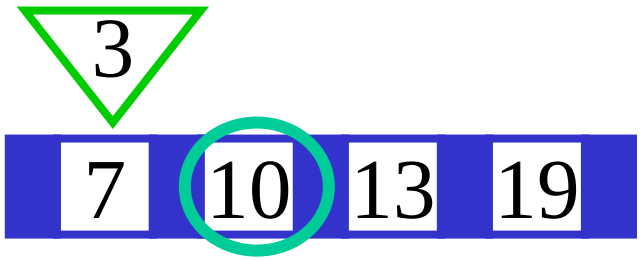
3. Ist der Knoten überfüllt, teile ihn
   - Lege einen neuen Knoten an und belege ihn mit den Schlüsseln, die rechts vom mittleren Eintrag des überfüllten Knotens liegen.
   - Füge den mittleren Eintrag im Vaterknoten des überfüllten Knotens ein.
   - Verbinde den Verweis rechts des neuen Eintrags im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

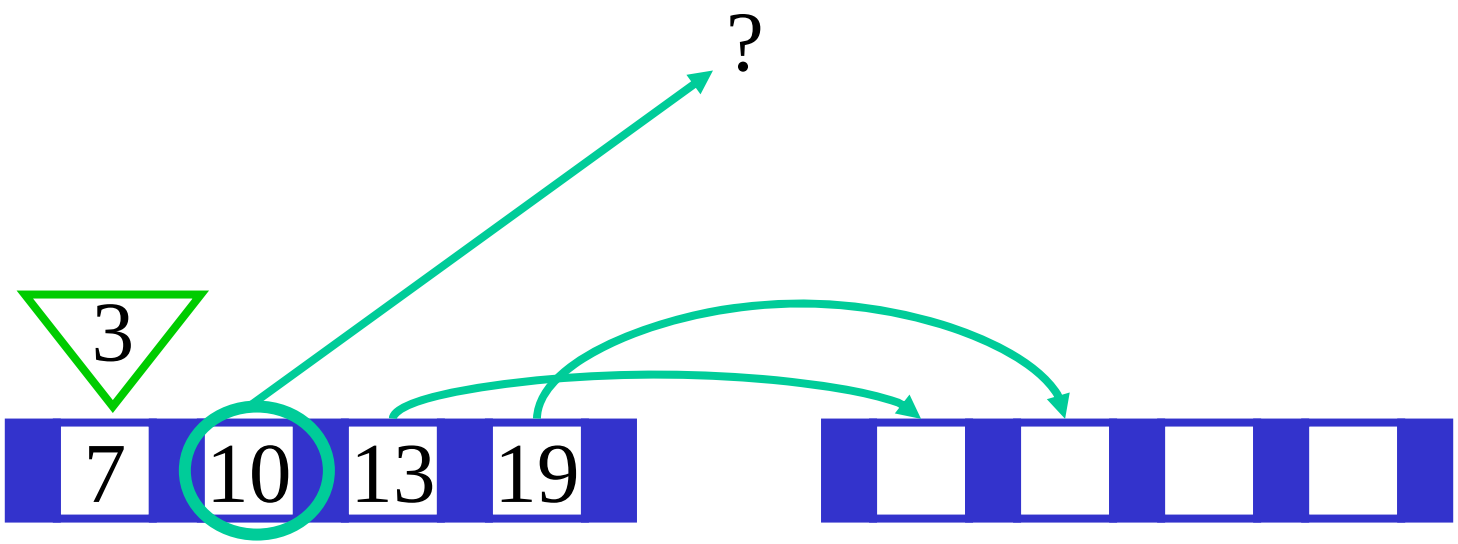
Sukzessiver Aufbau eines B-Baums vom Grad k=2



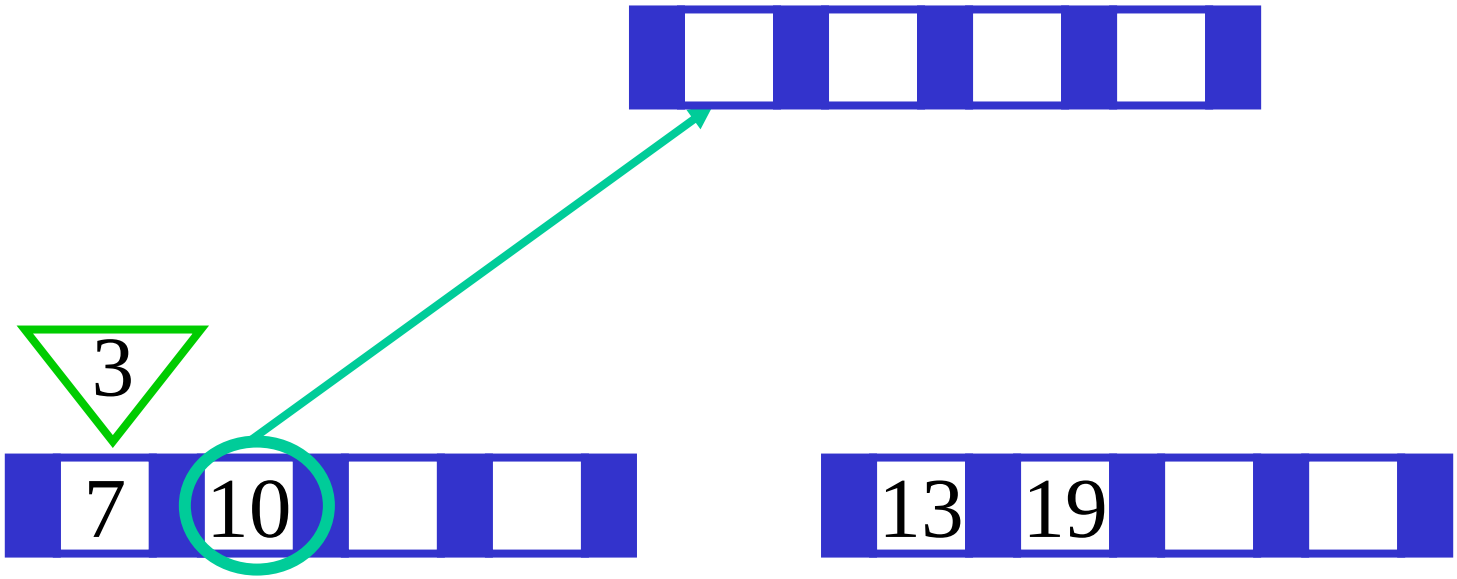
Sukzessiver Aufbau eines B-Baums vom Grad k=2



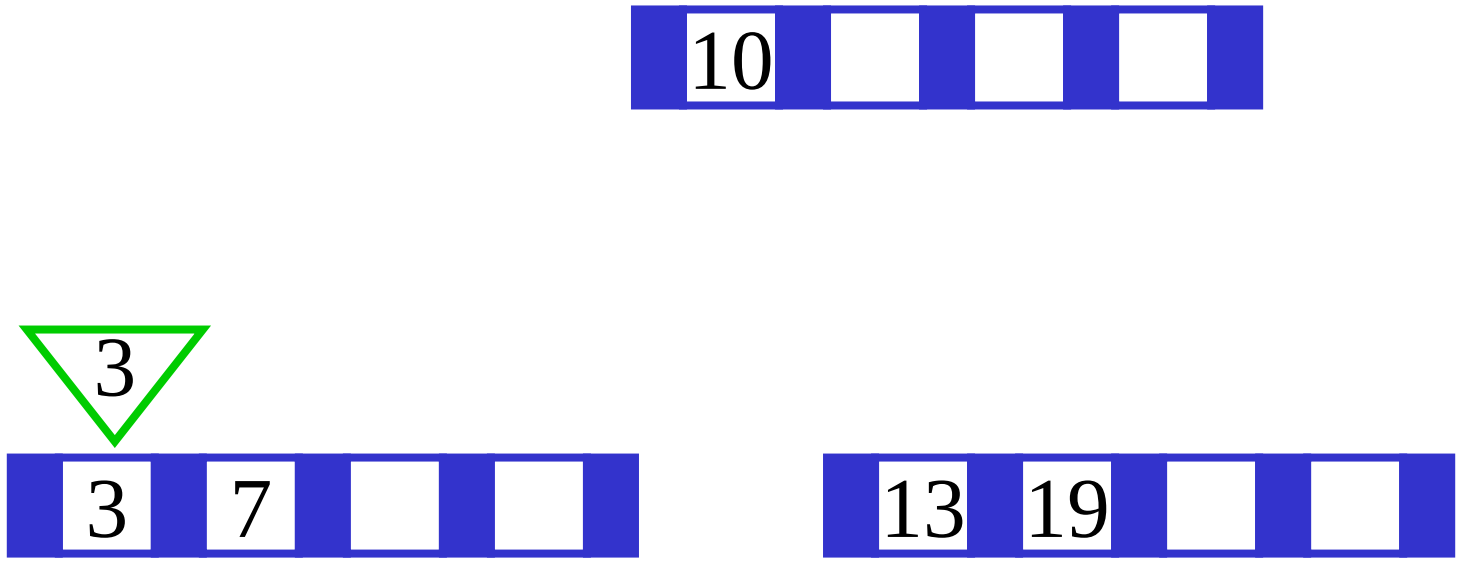
Sukzessiver Aufbau eines B-Baums vom Grad k=2



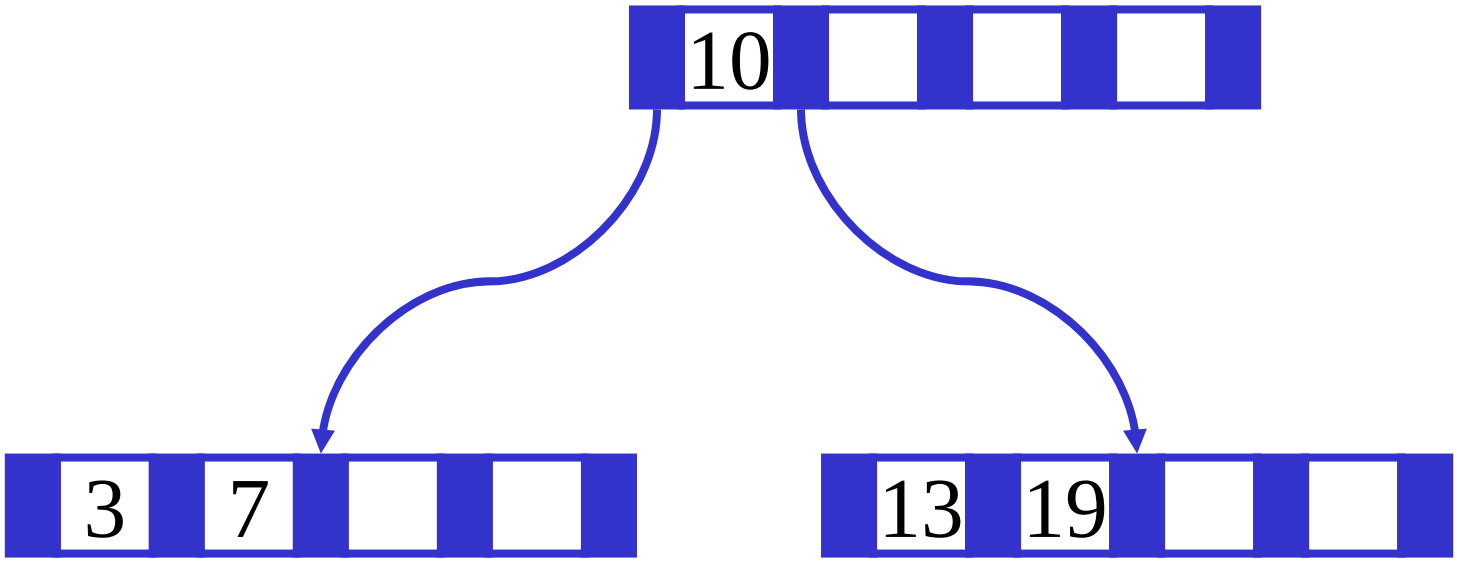
Sukzessiver Aufbau eines B-Baums vom Grad k=2



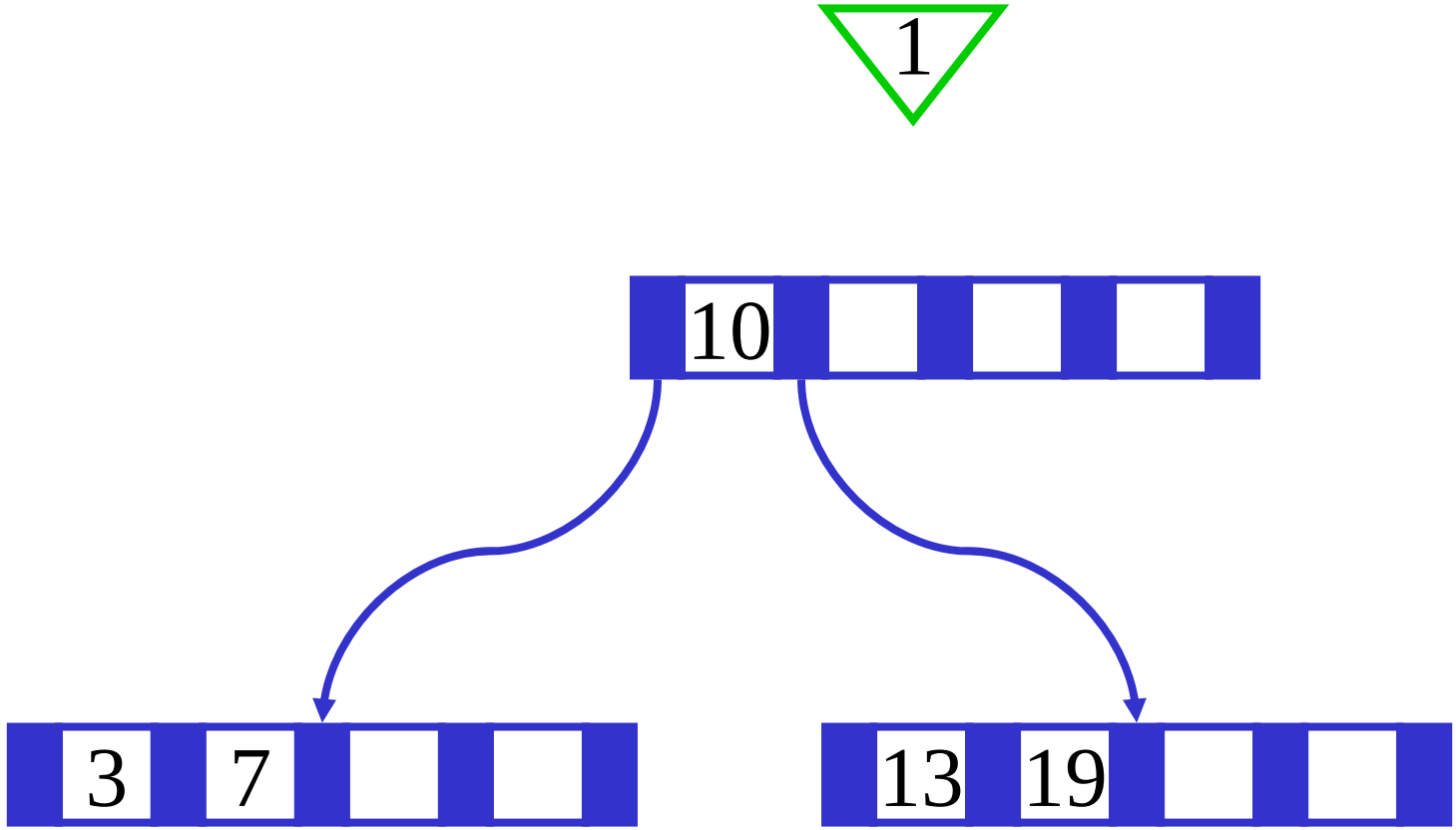
Sukzessiver Aufbau eines B-Baums vom Grad k=2



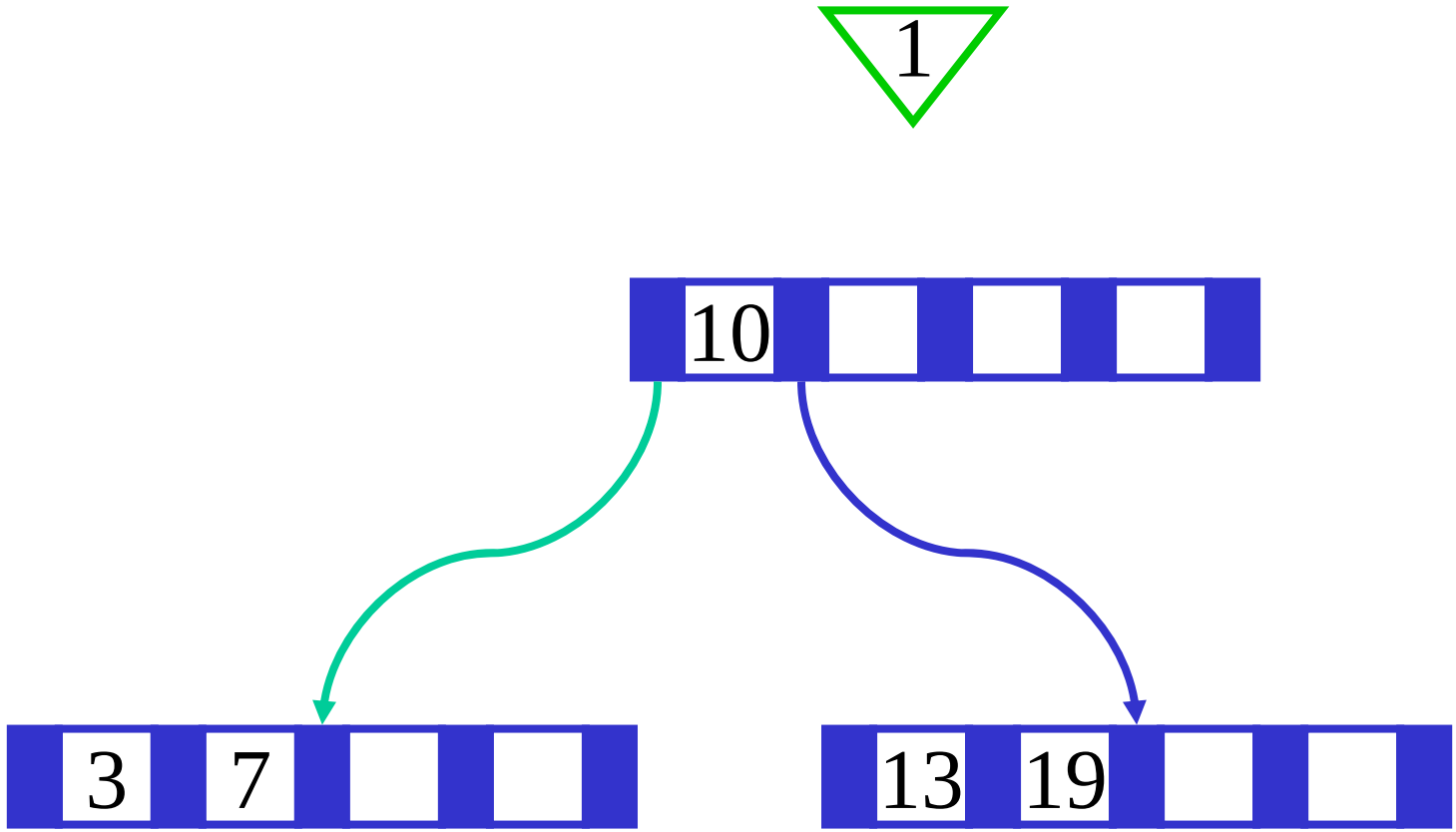
Sukzessiver Aufbau eines B-Baums vom Grad k=2



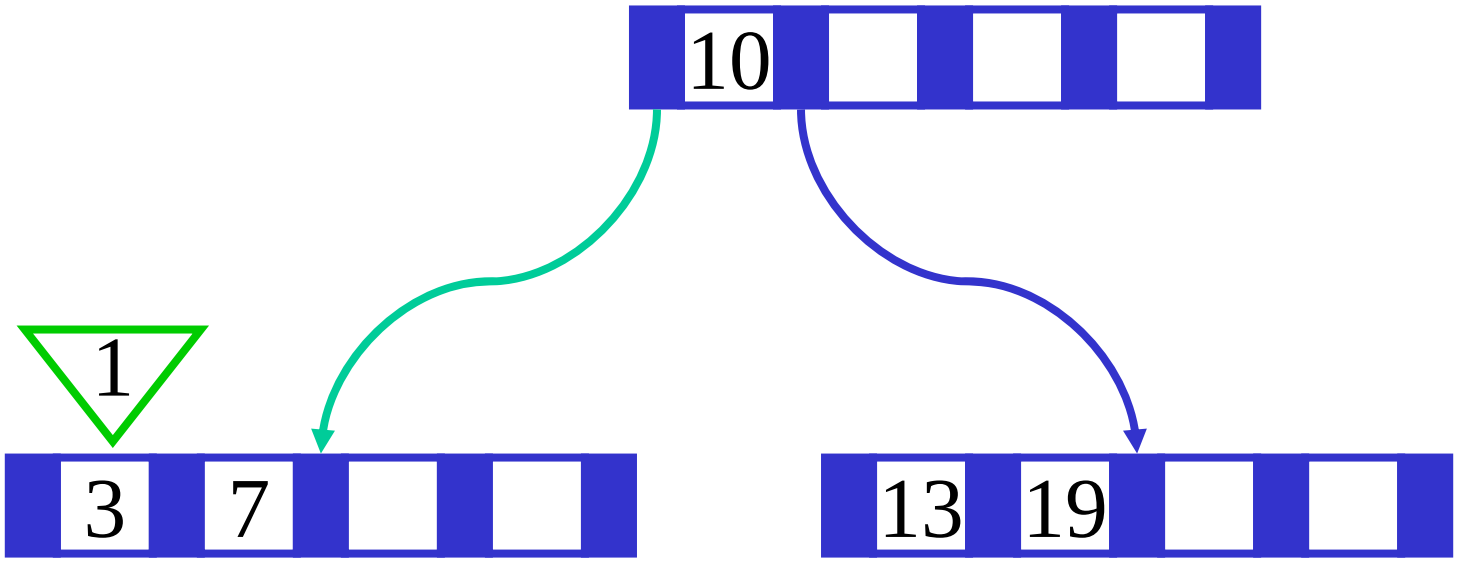
Sukzessiver Aufbau eines B-Baums vom Grad k=2



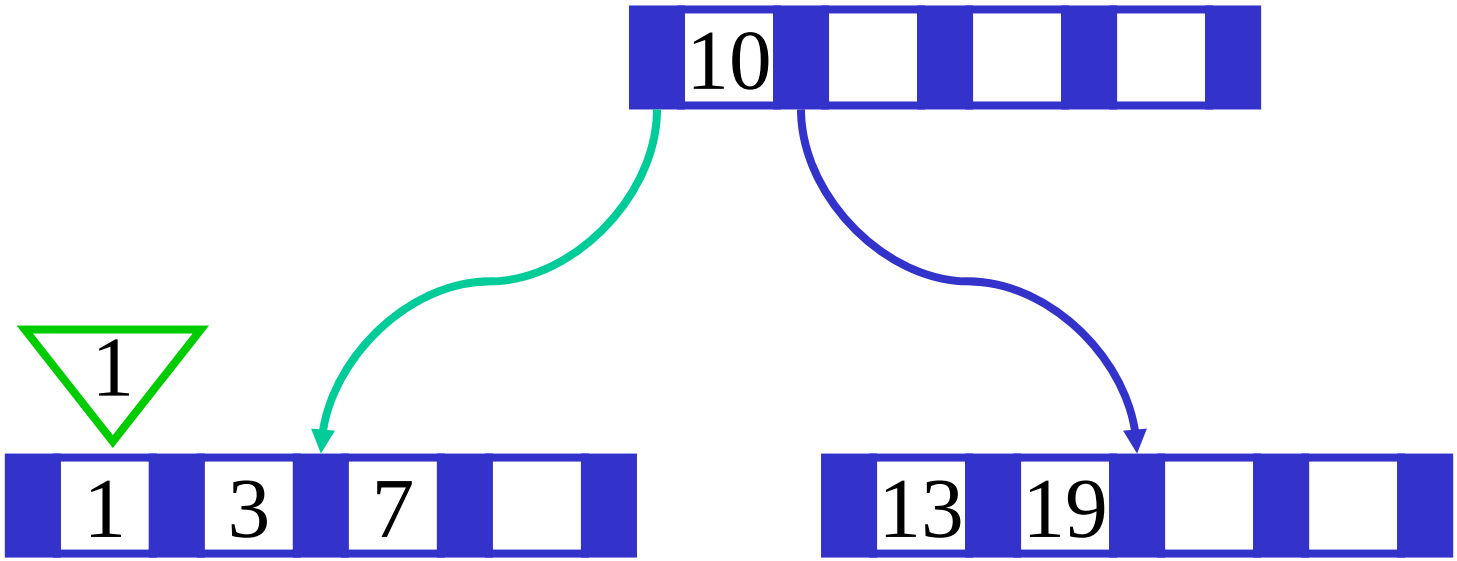
Sukzessiver Aufbau eines B-Baums vom Grad k=2



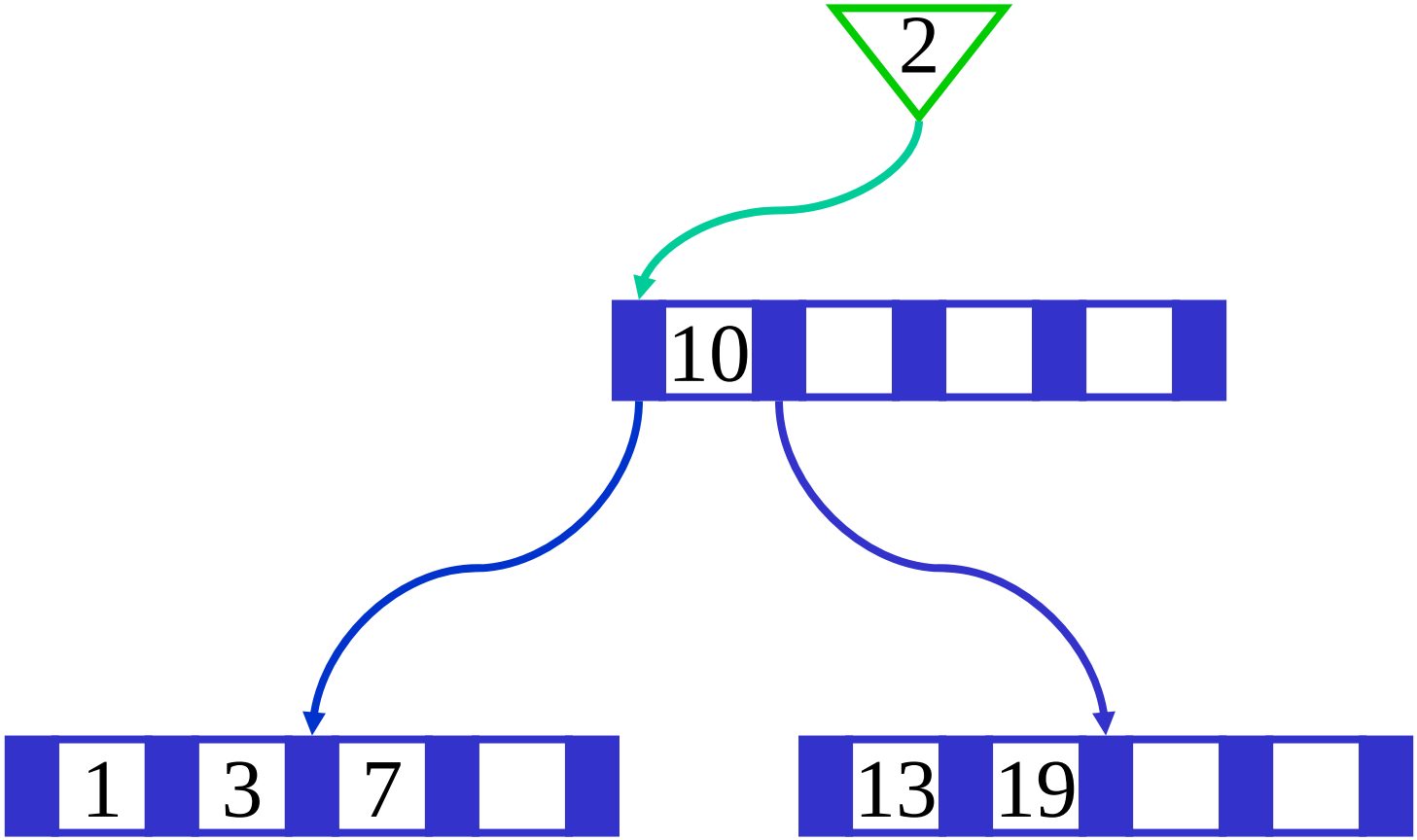
Sukzessiver Aufbau eines B-Baums vom Grad k=2



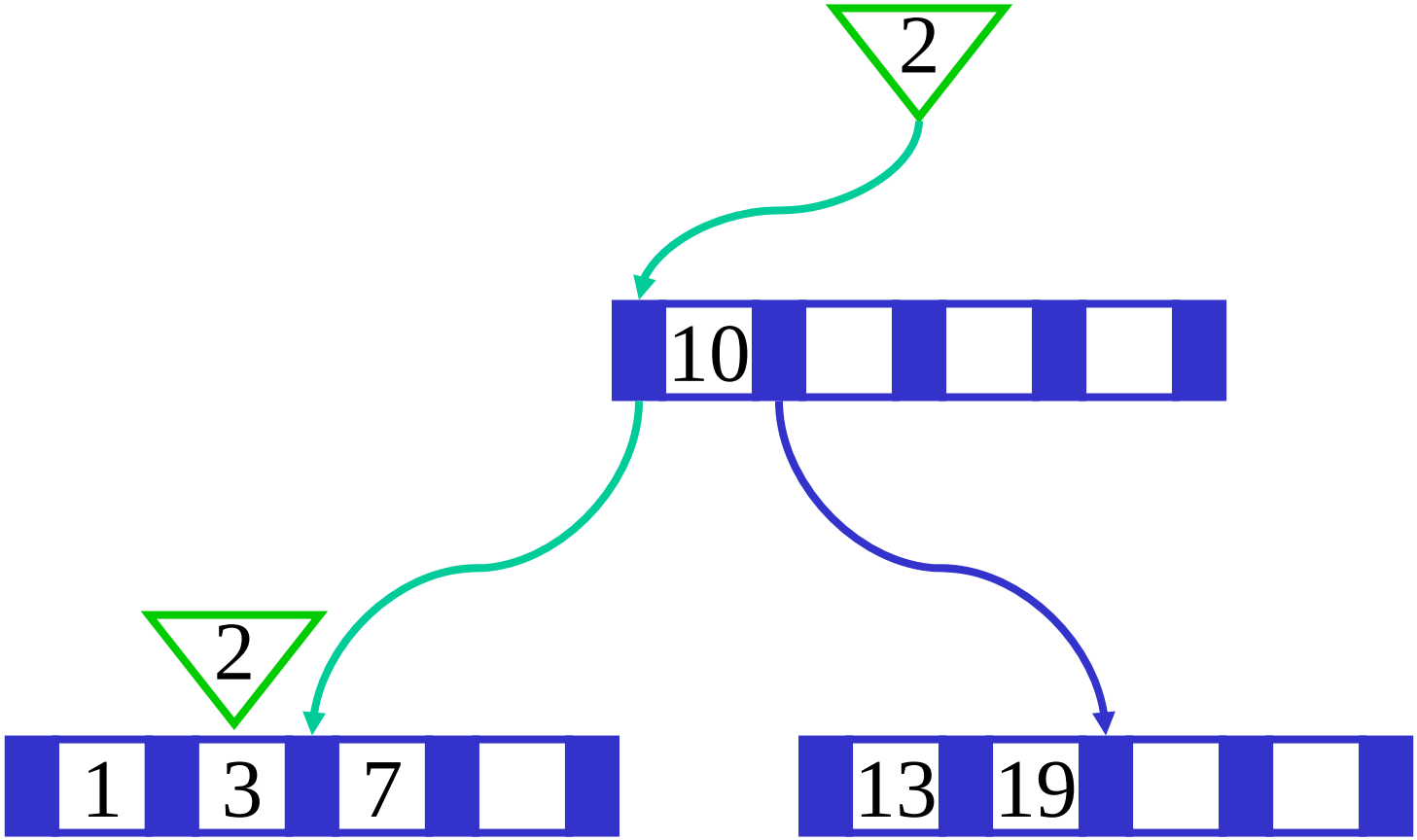
Sukzessiver Aufbau eines B-Baums vom Grad k=2



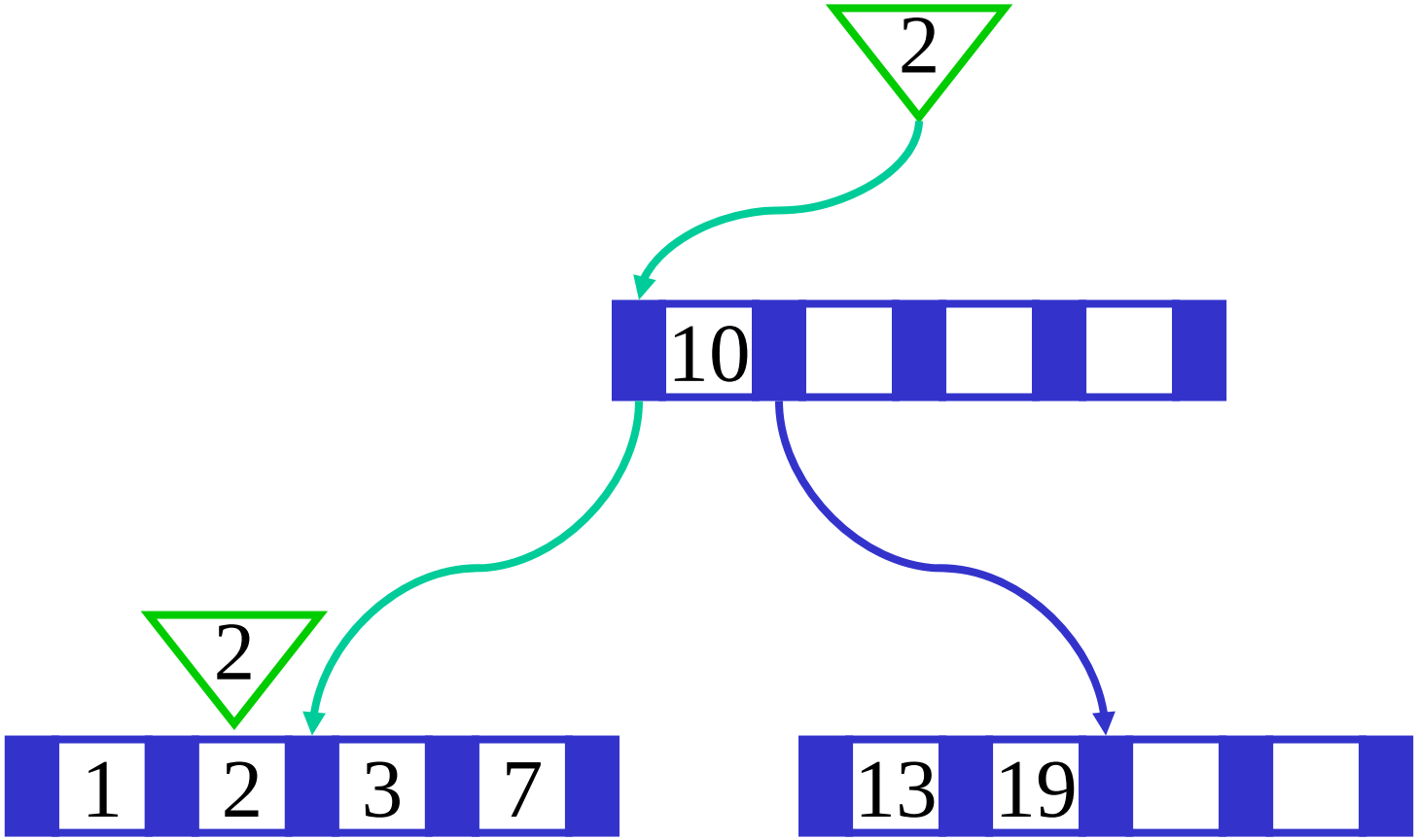
Sukzessiver Aufbau eines B-Baums vom Grad k=2



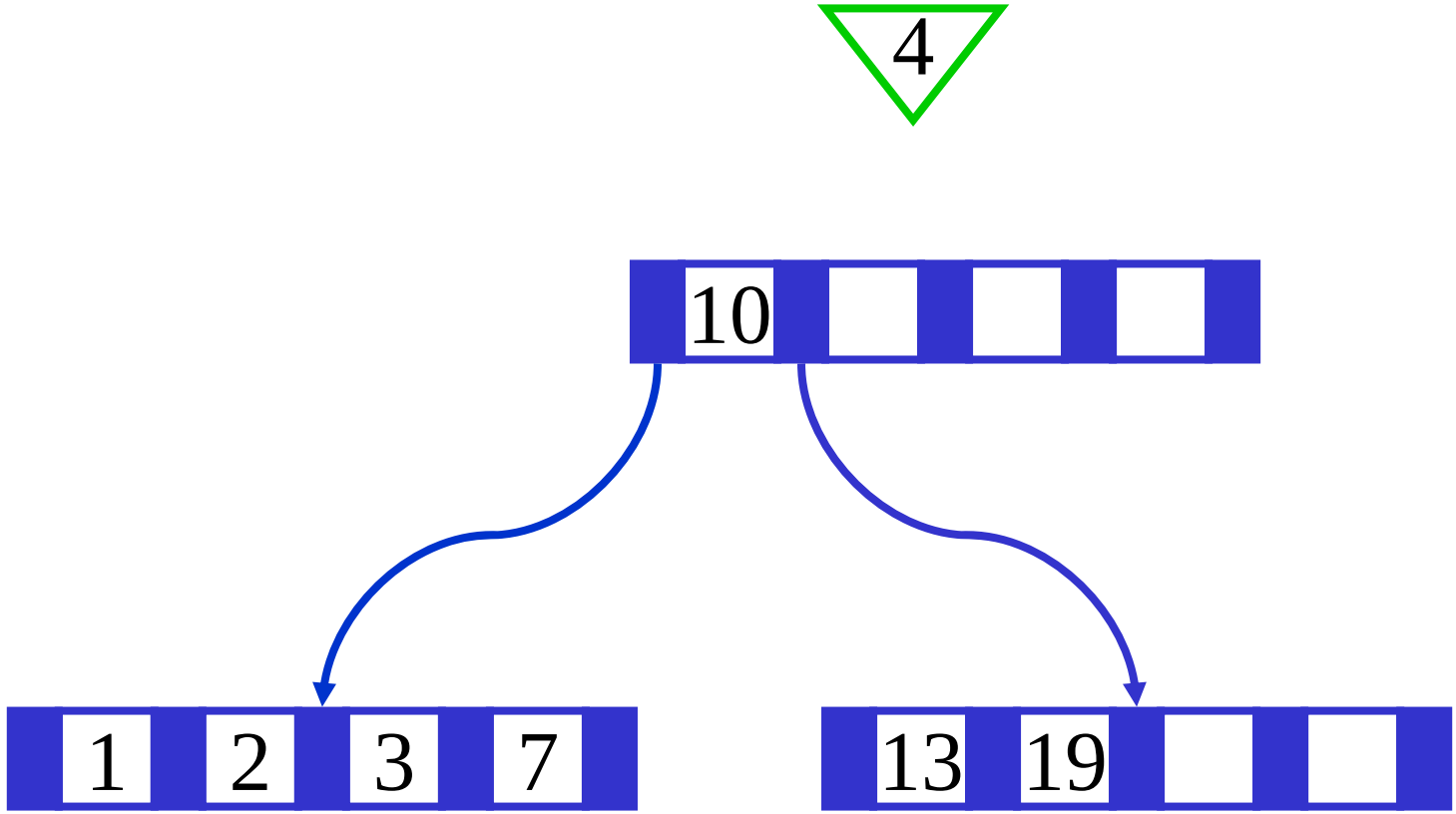
Sukzessiver Aufbau eines B-Baums vom Grad k=2



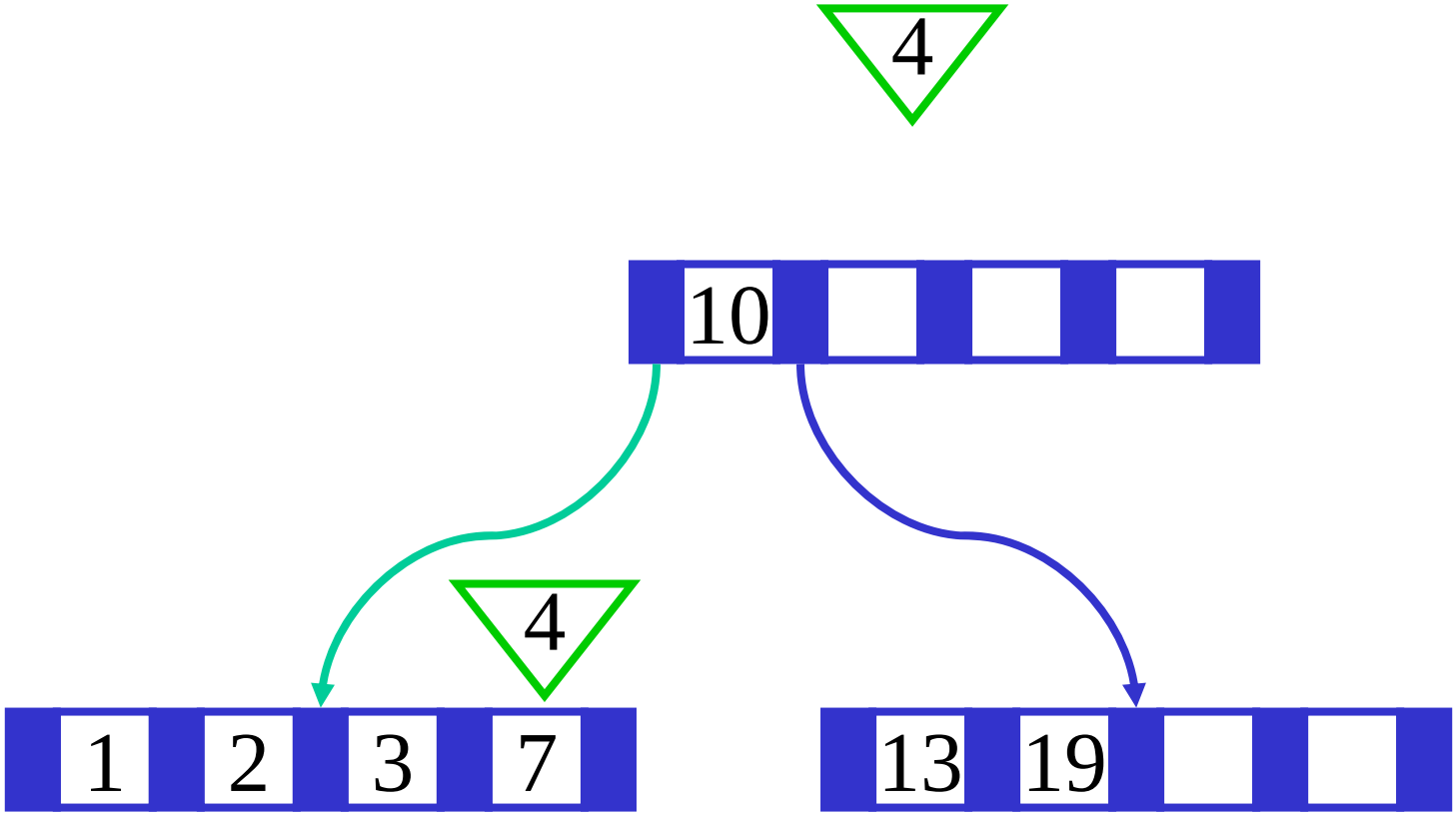
Sukzessiver Aufbau eines B-Baums vom Grad k=2



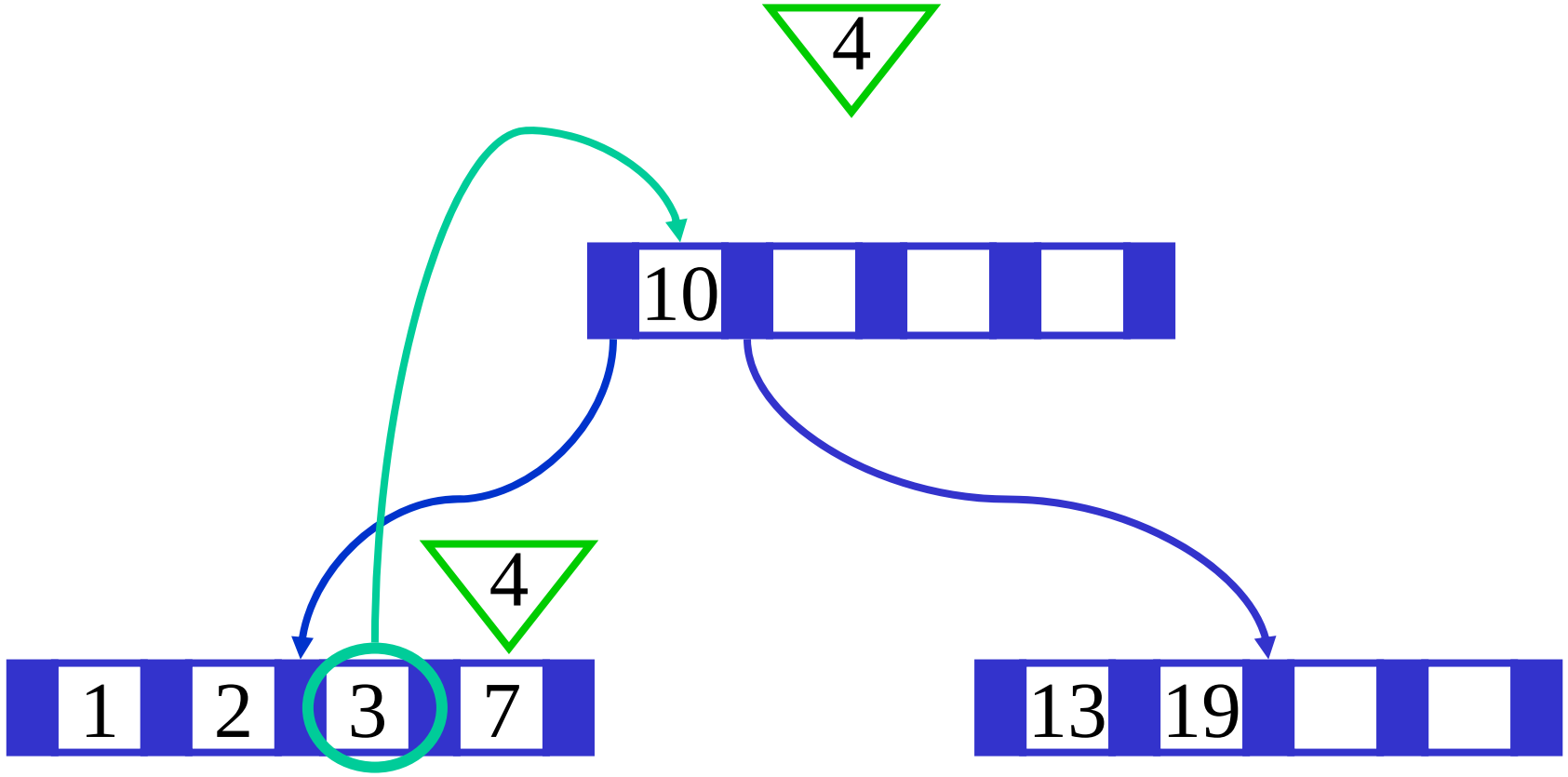
Sukzessiver Aufbau eines B-Baums vom Grad k=2



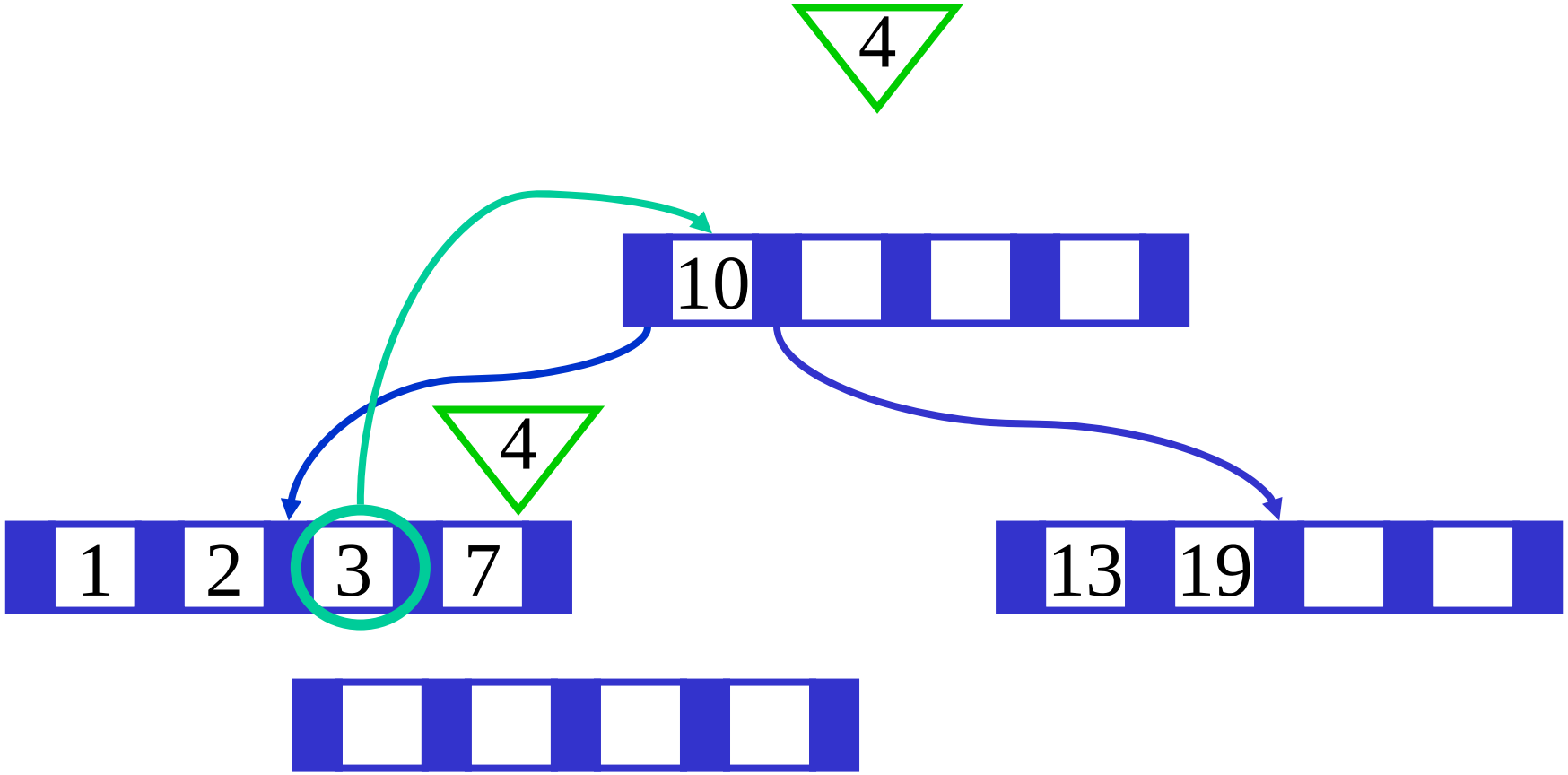
Sukzessiver Aufbau eines B-Baums vom Grad k=2



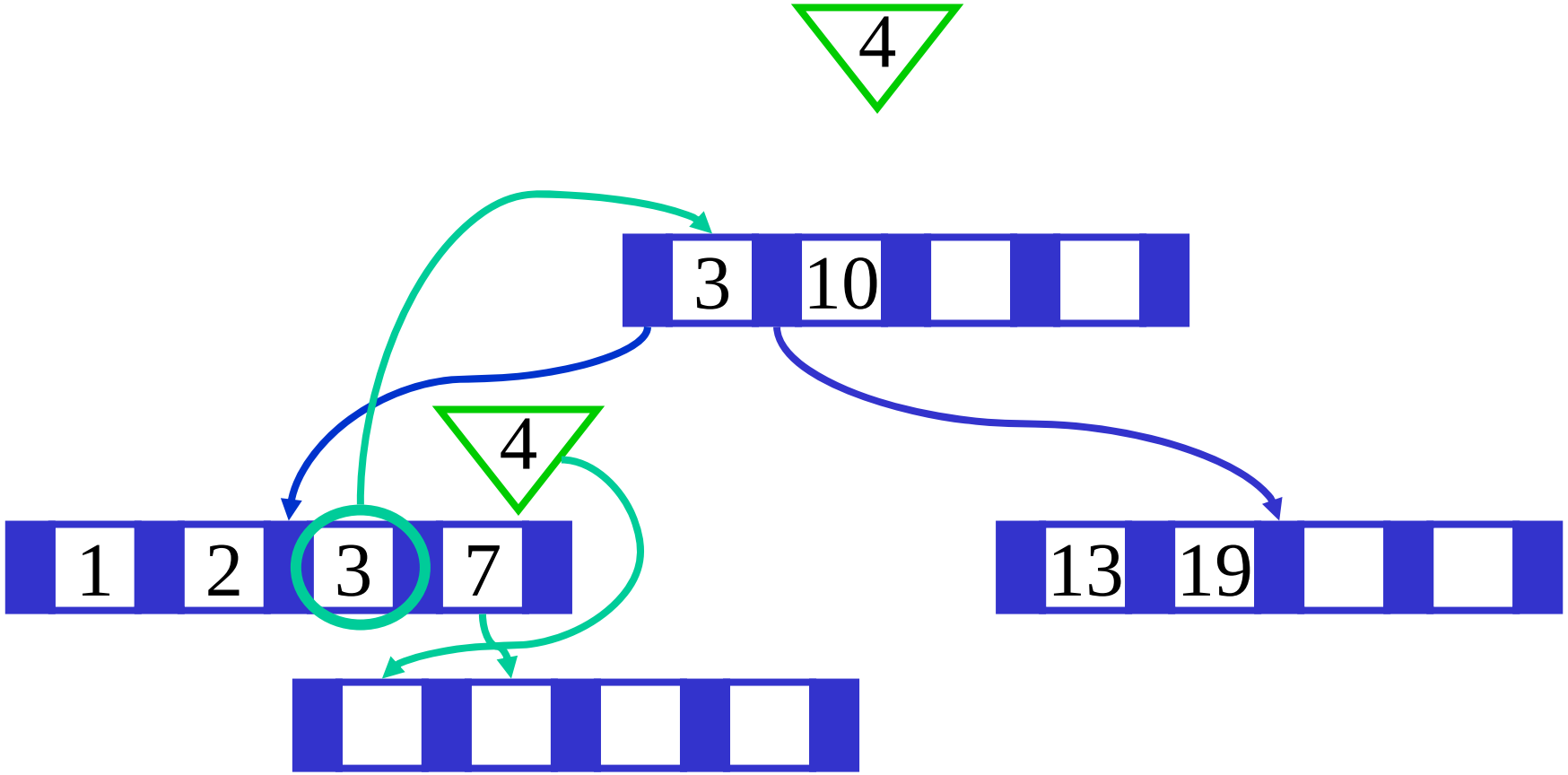
Sukzessiver Aufbau eines B-Baums vom Grad k=2



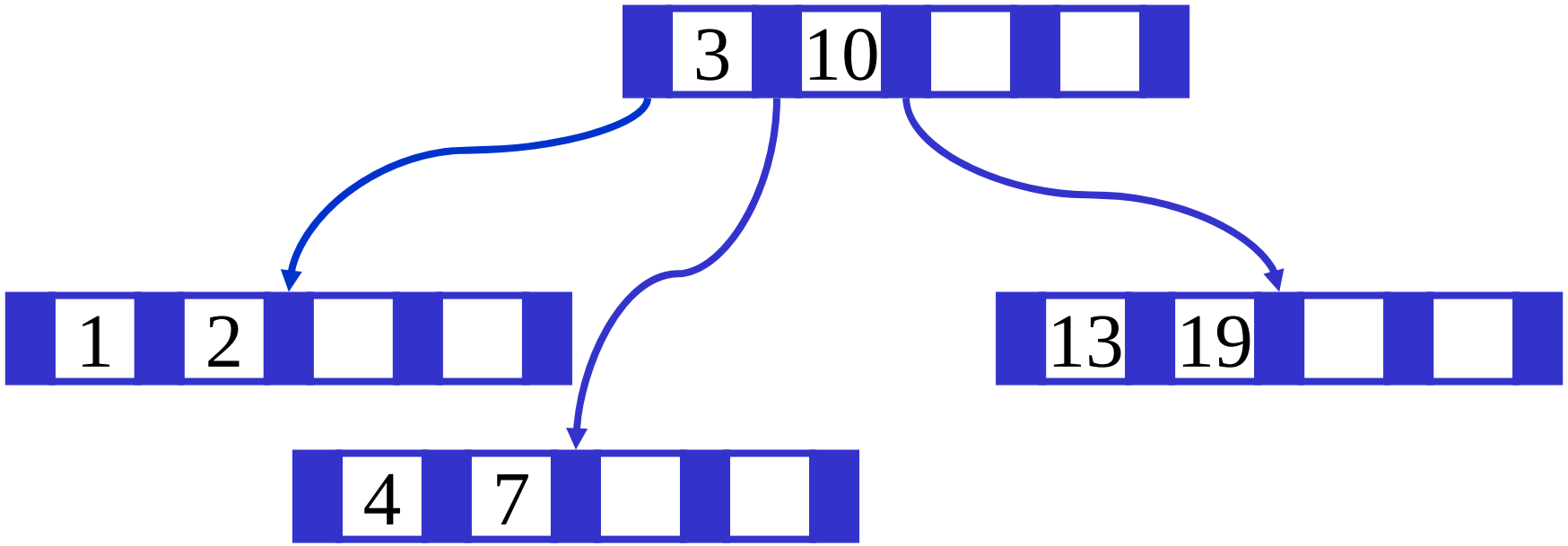
Sukzessiver Aufbau eines B-Baums vom Grad k=2



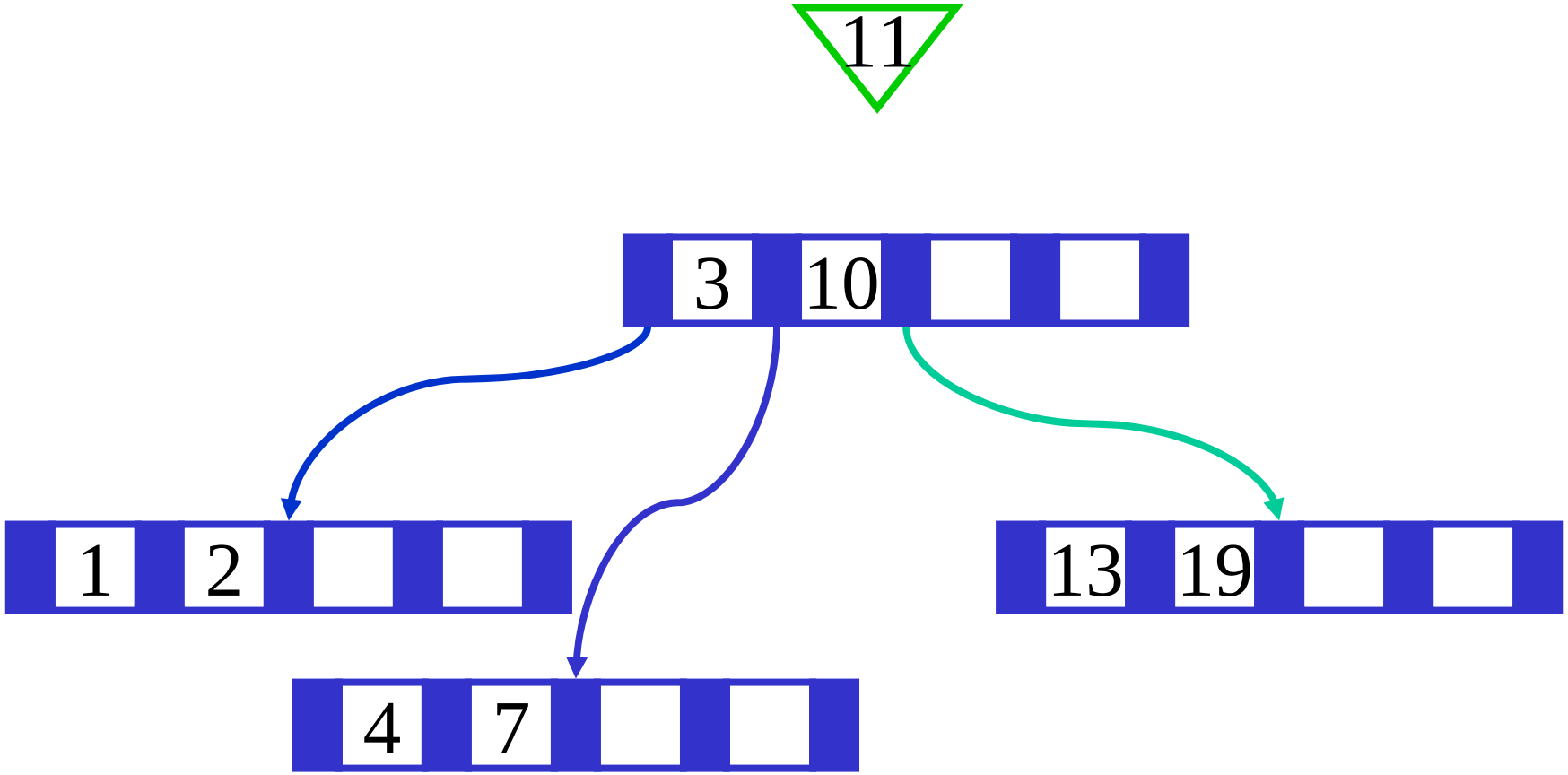
Sukzessiver Aufbau eines B-Baums vom Grad k=2



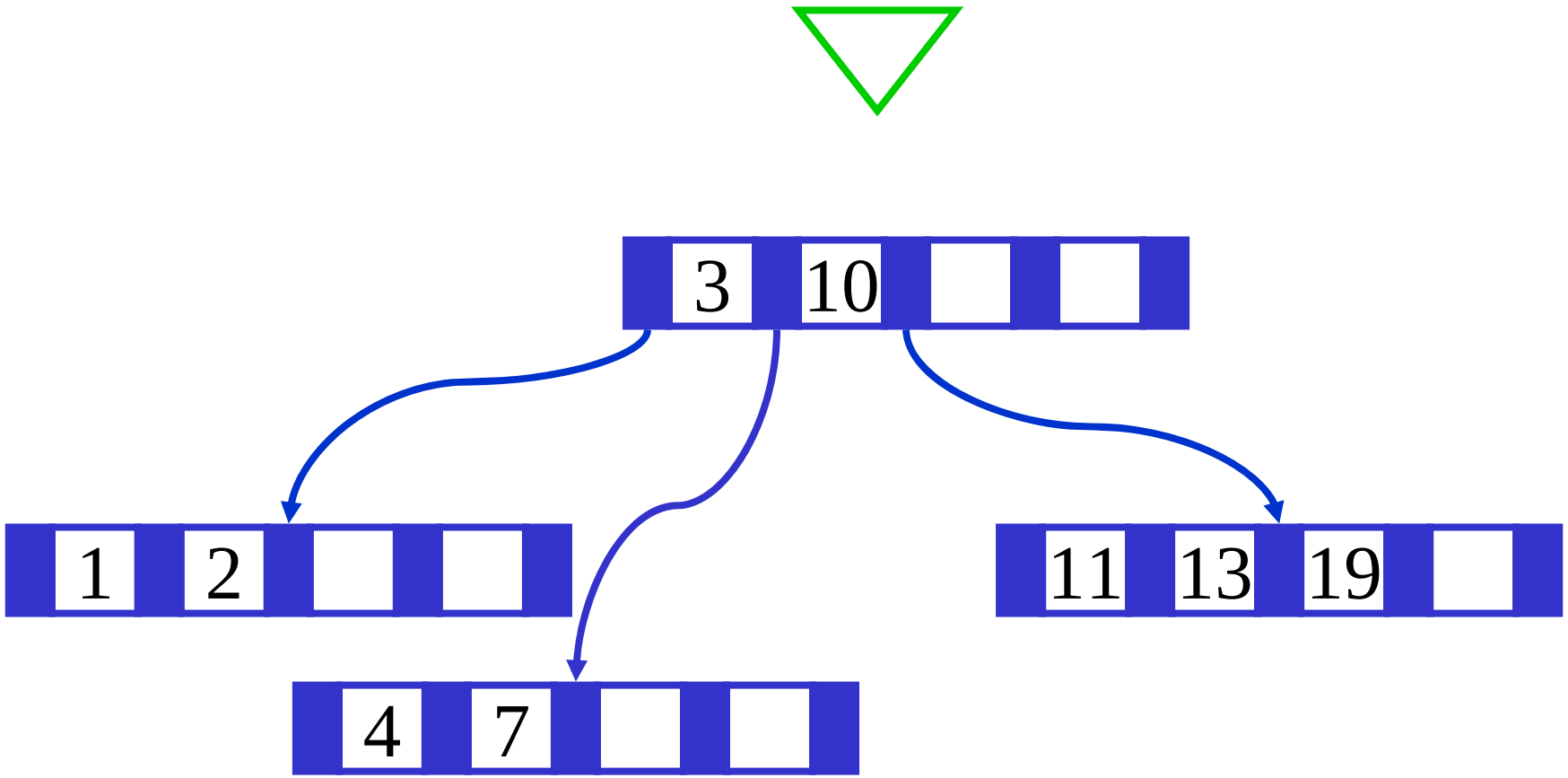
Sukzessiver Aufbau eines B-Baums vom Grad k=2



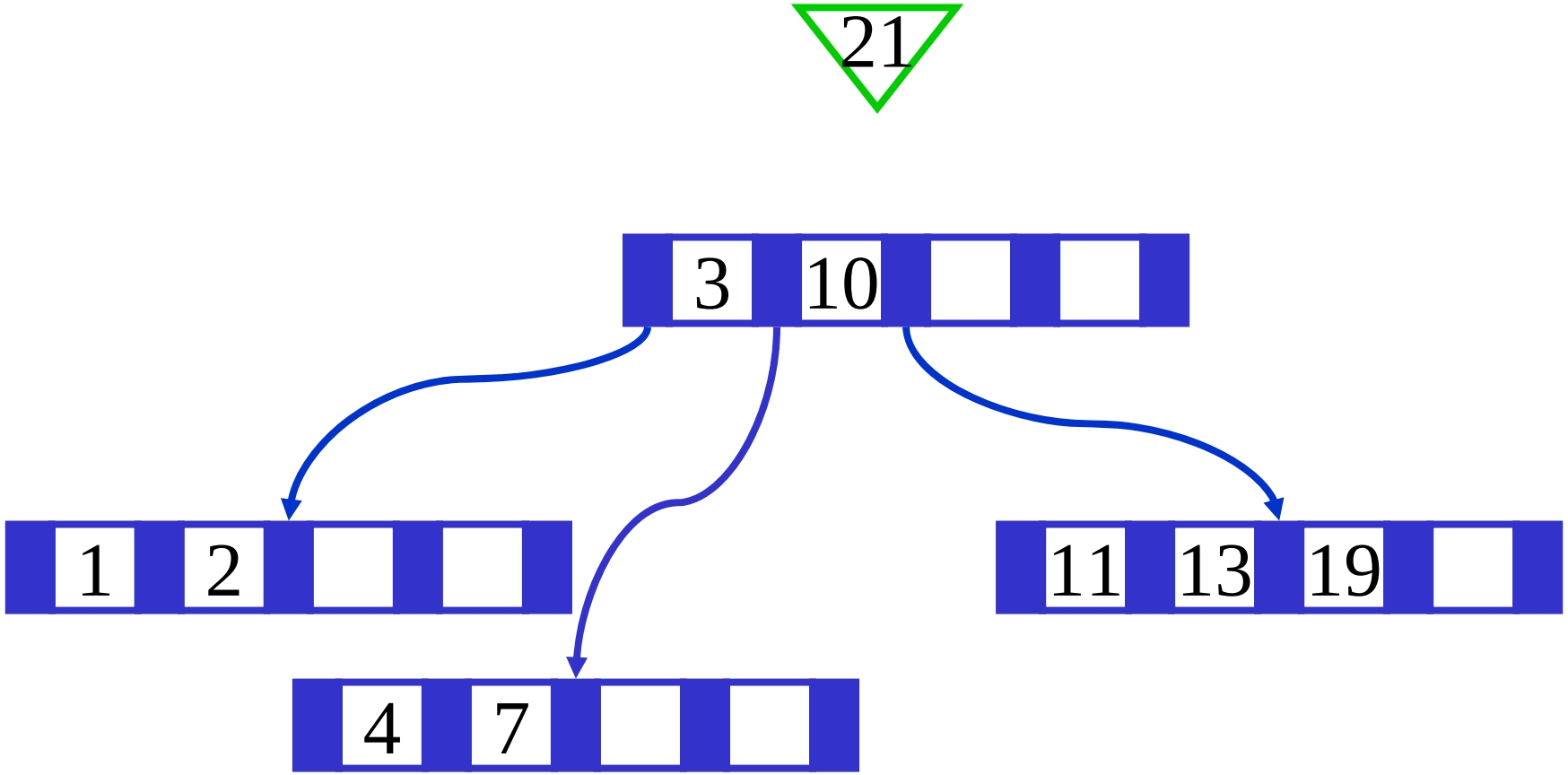
Sukzessiver Aufbau eines B-Baums vom Grad k=2



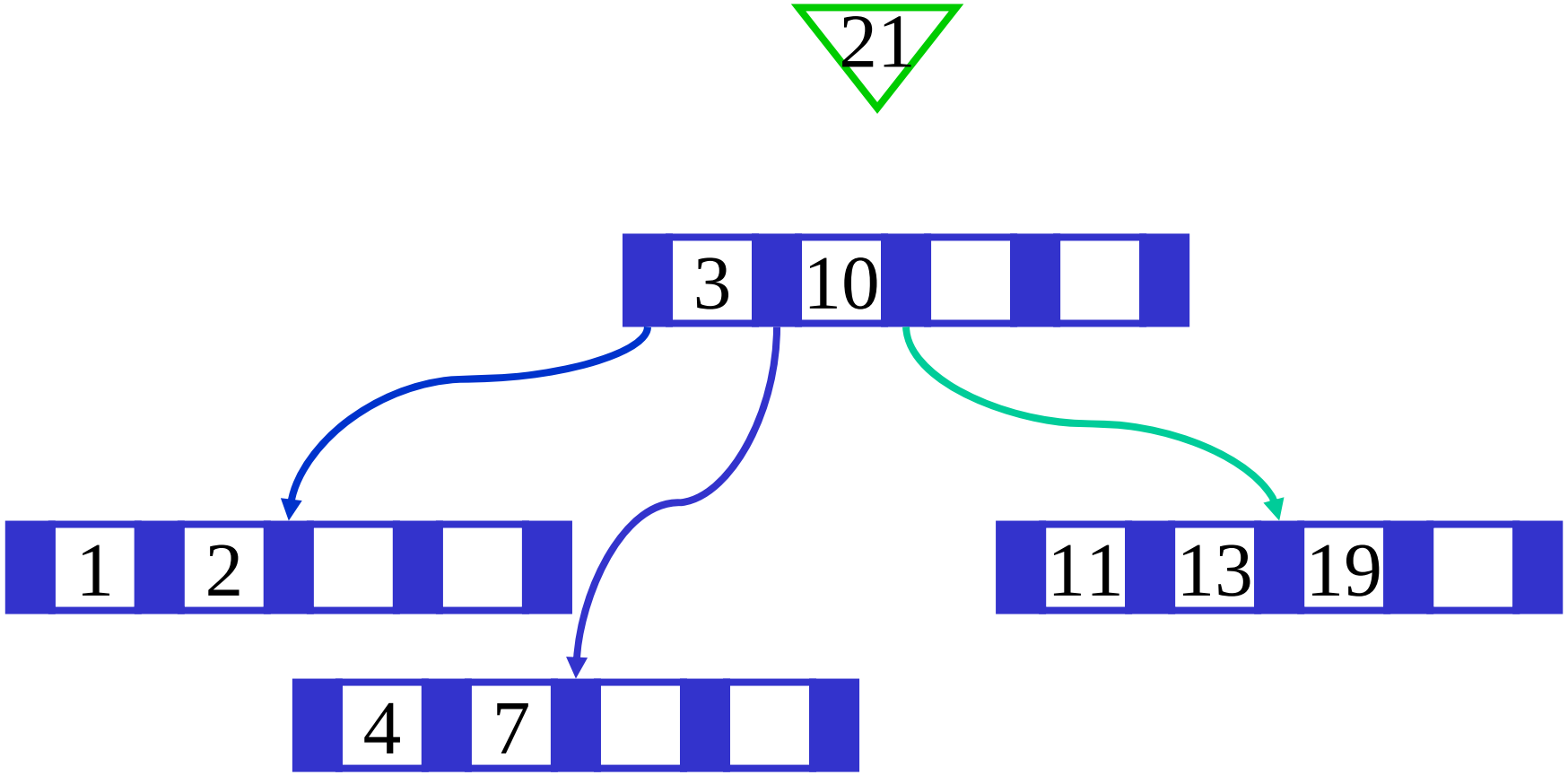
Sukzessiver Aufbau eines B-Baums vom Grad k=2



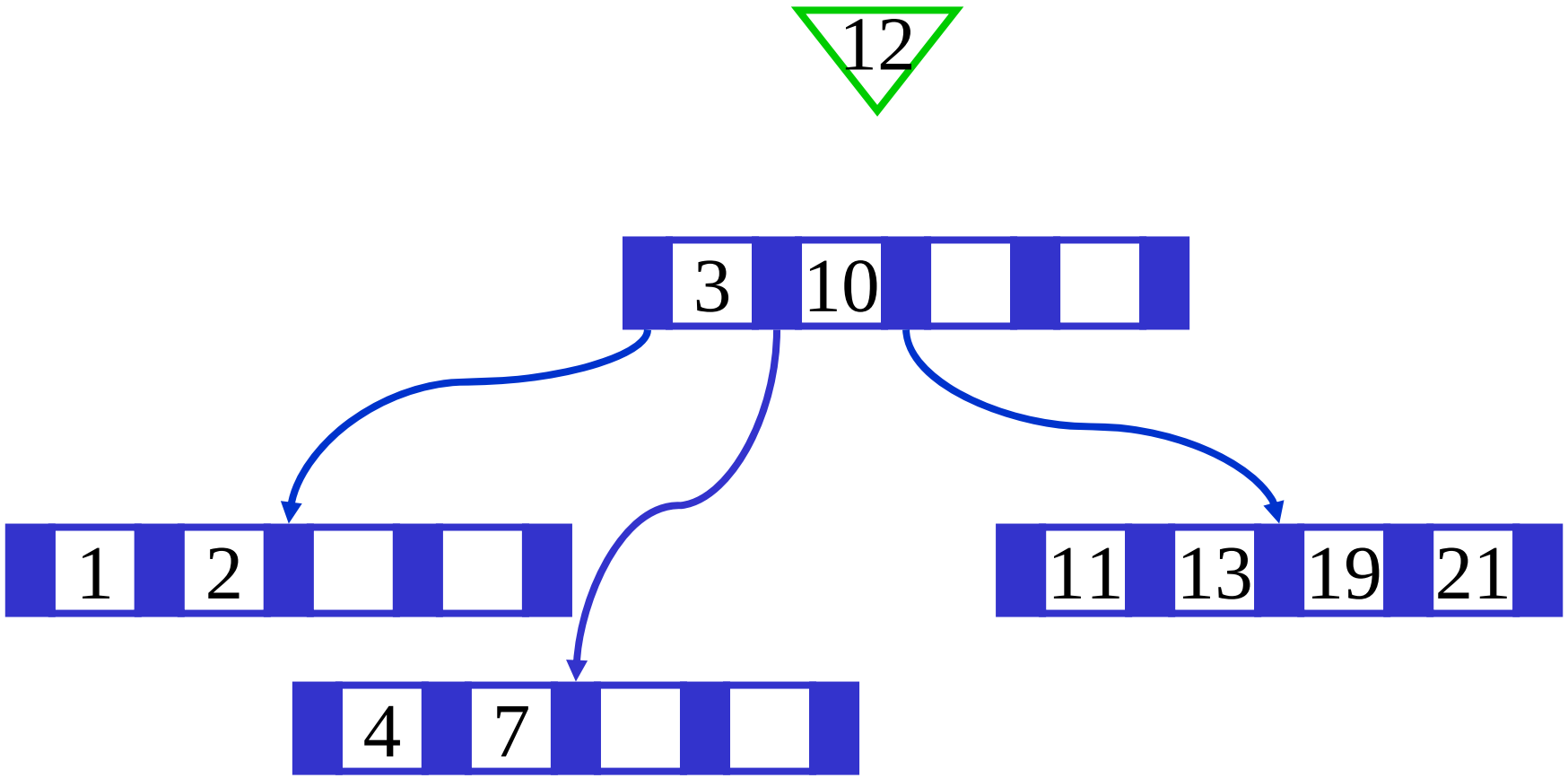
Sukzessiver Aufbau eines B-Baums vom Grad k=2



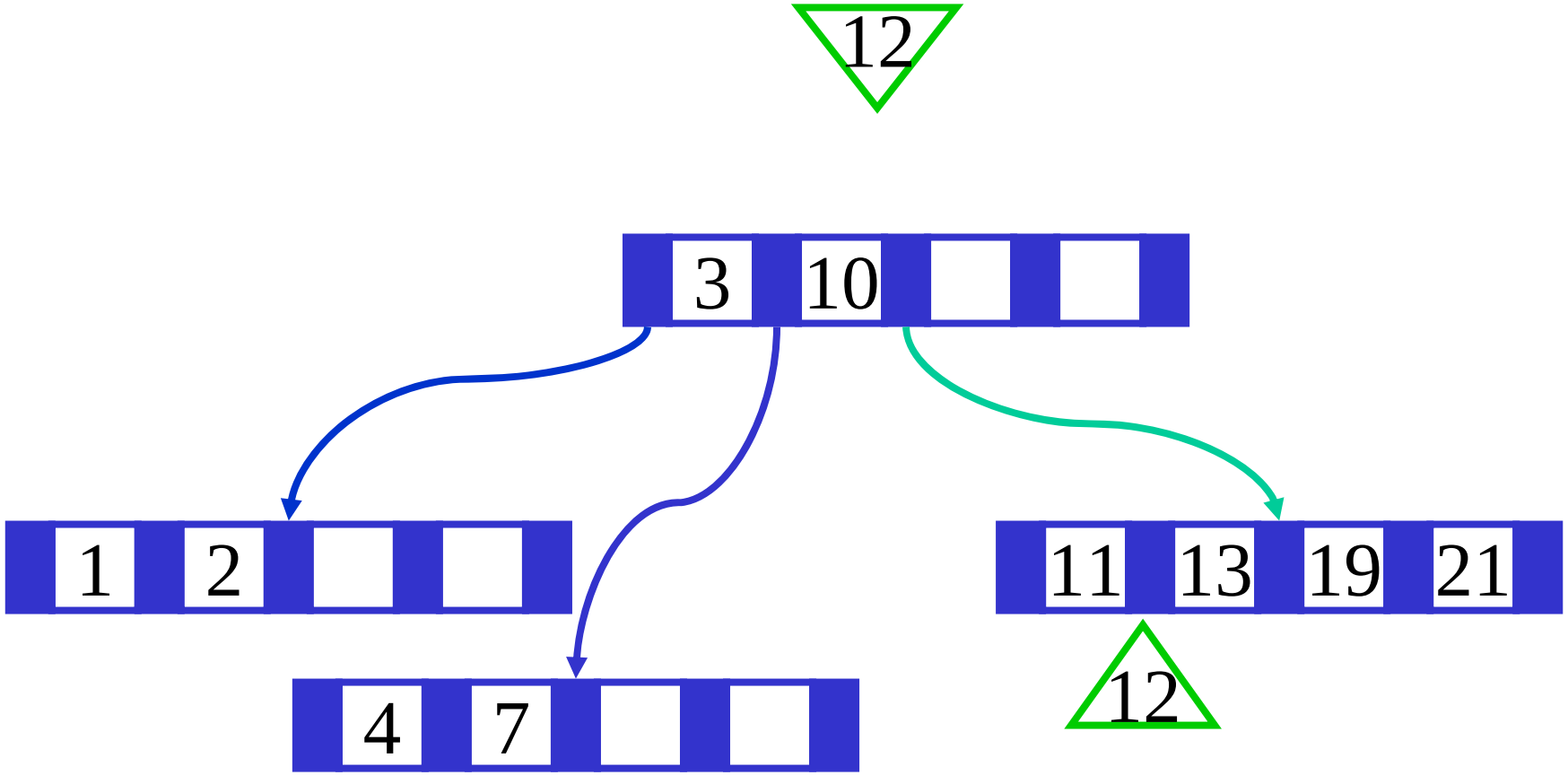
Sukzessiver Aufbau eines B-Baums vom Grad k=2



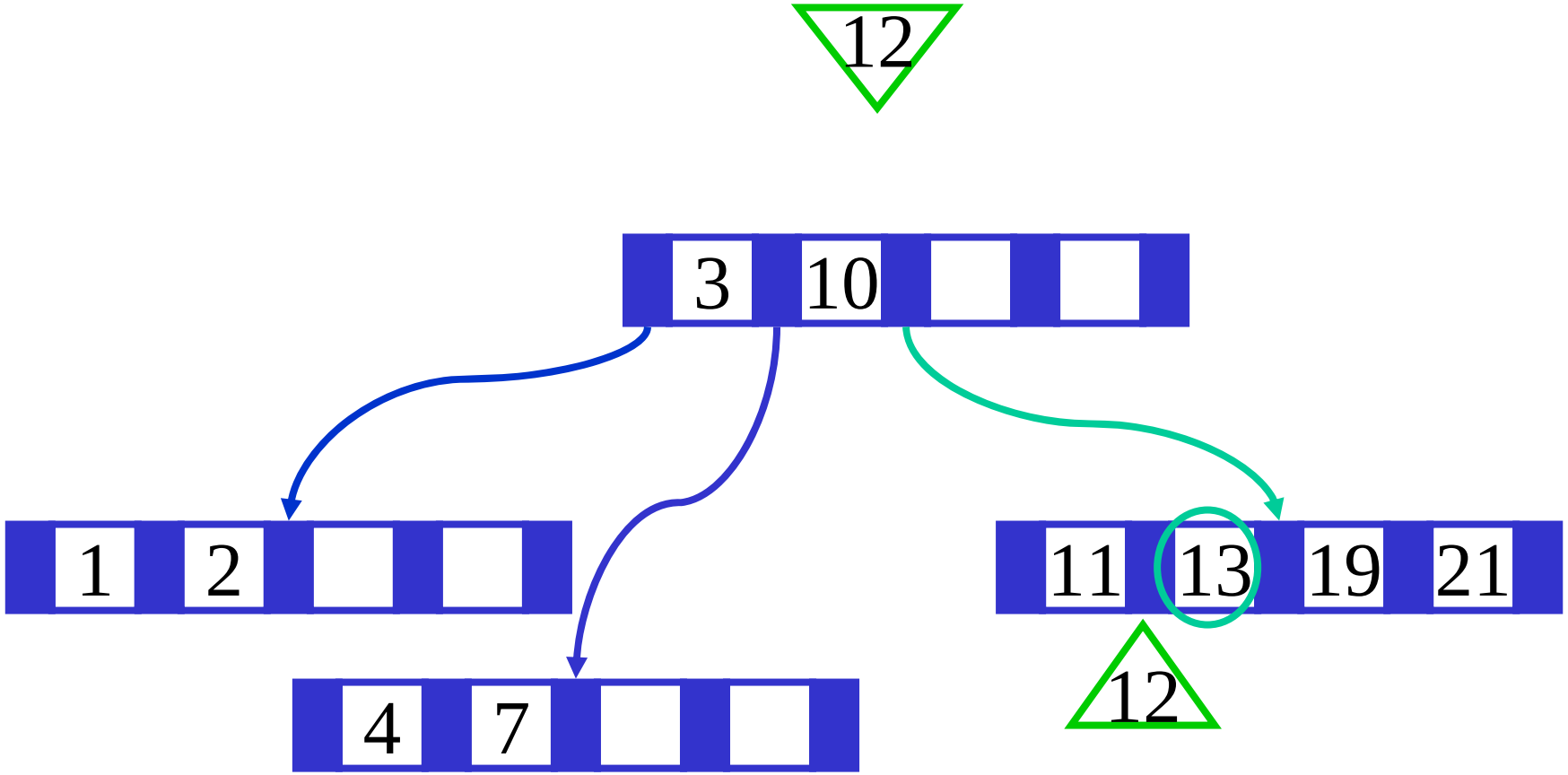
Sukzessiver Aufbau eines B-Baums vom Grad k=2



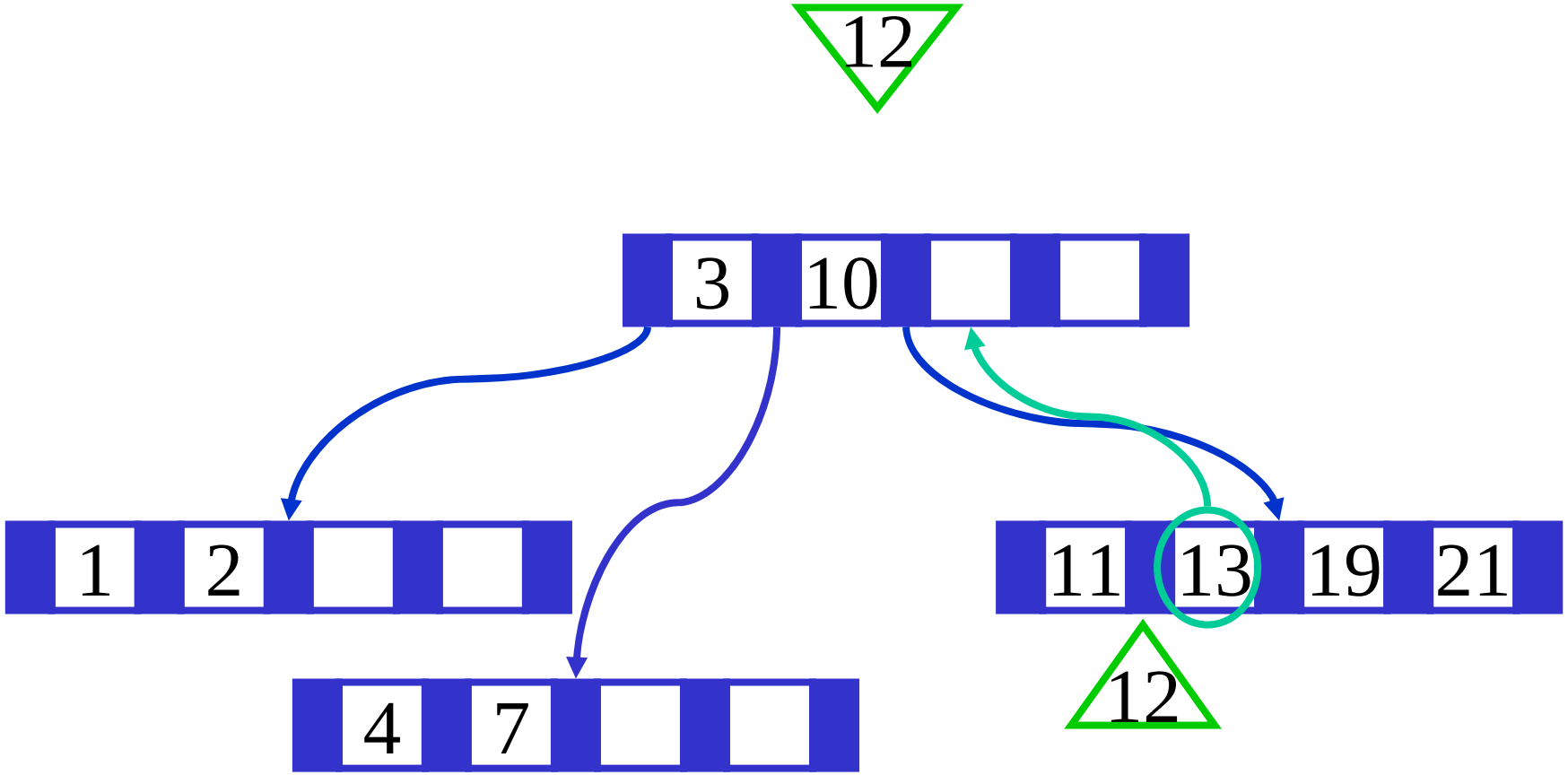
Sukzessiver Aufbau eines B-Baums vom Grad k=2



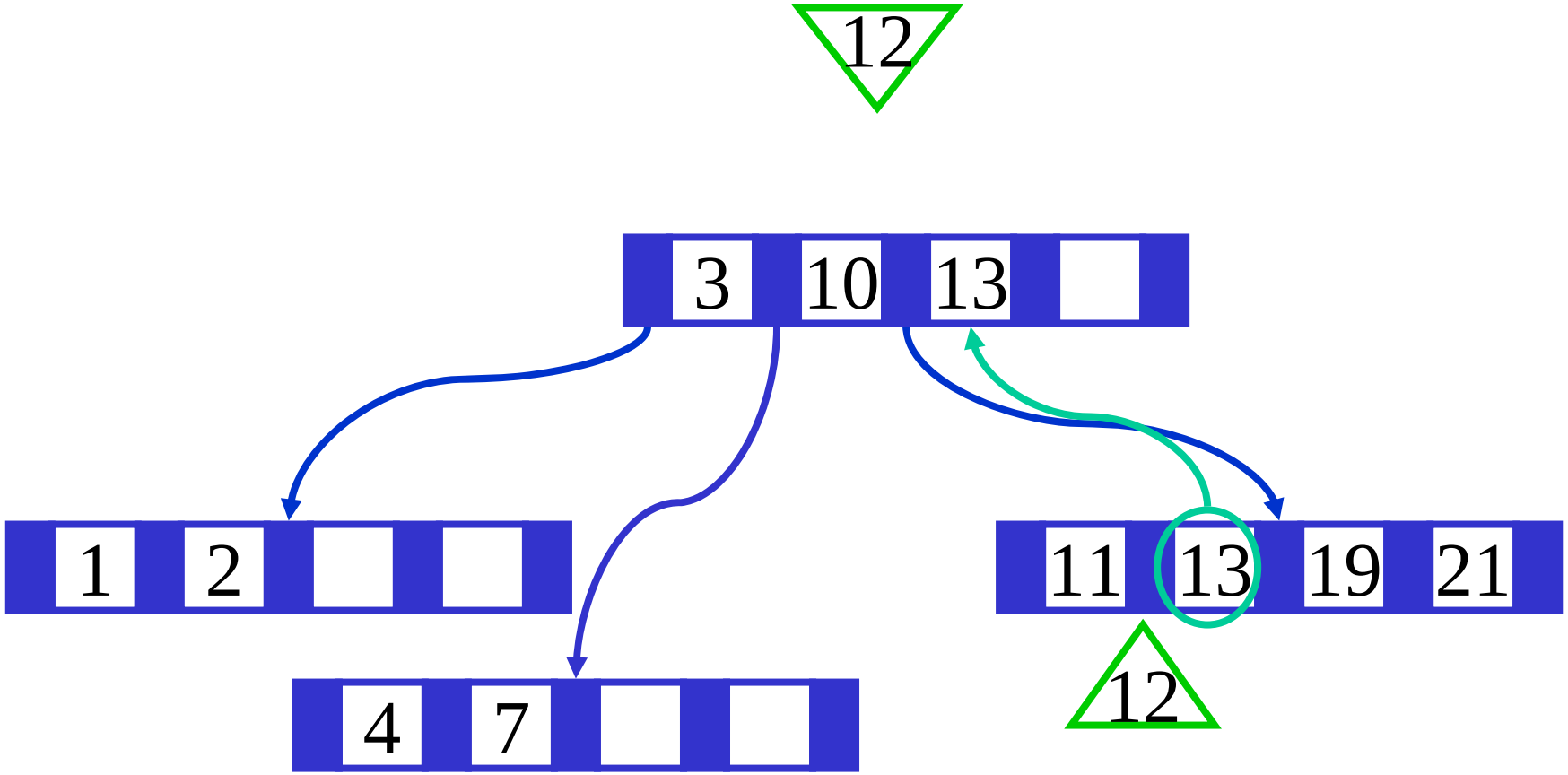
Sukzessiver Aufbau eines B-Baums vom Grad k=2



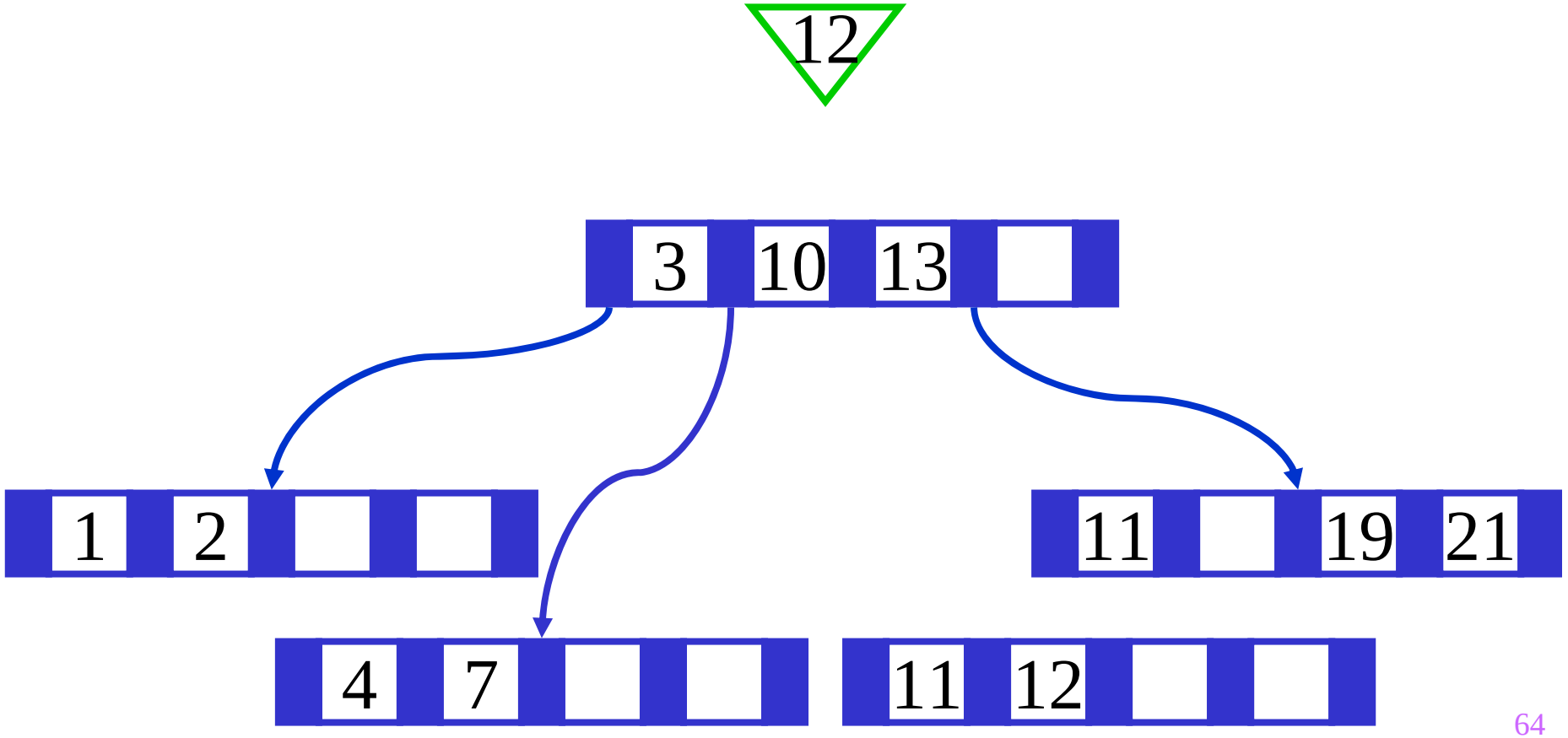
Sukzessiver Aufbau eines B-Baums vom Grad k=2



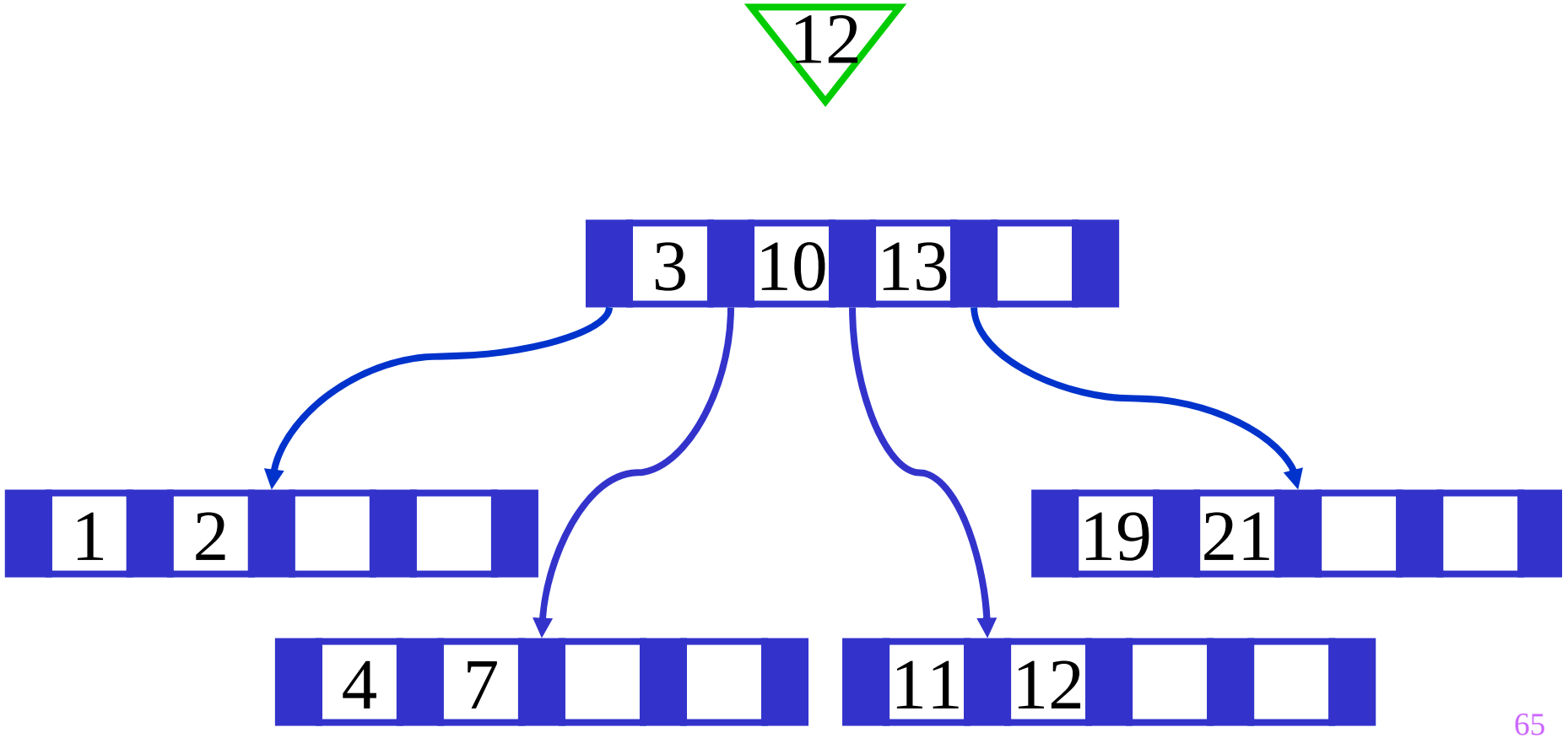
Sukzessiver Aufbau eines B-Baums vom Grad k=2



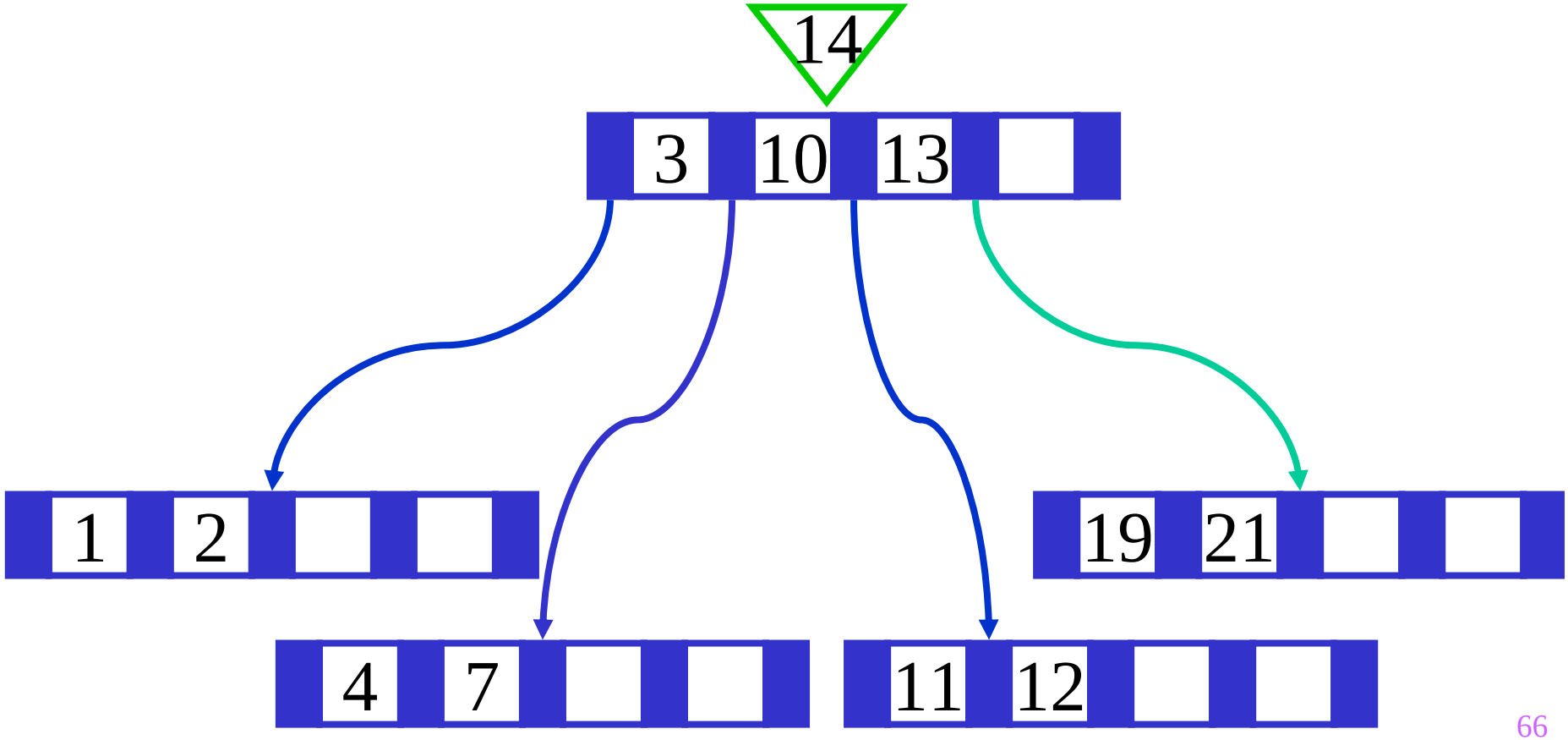
Sukzessiver Aufbau eines B-Baums vom Grad k=2



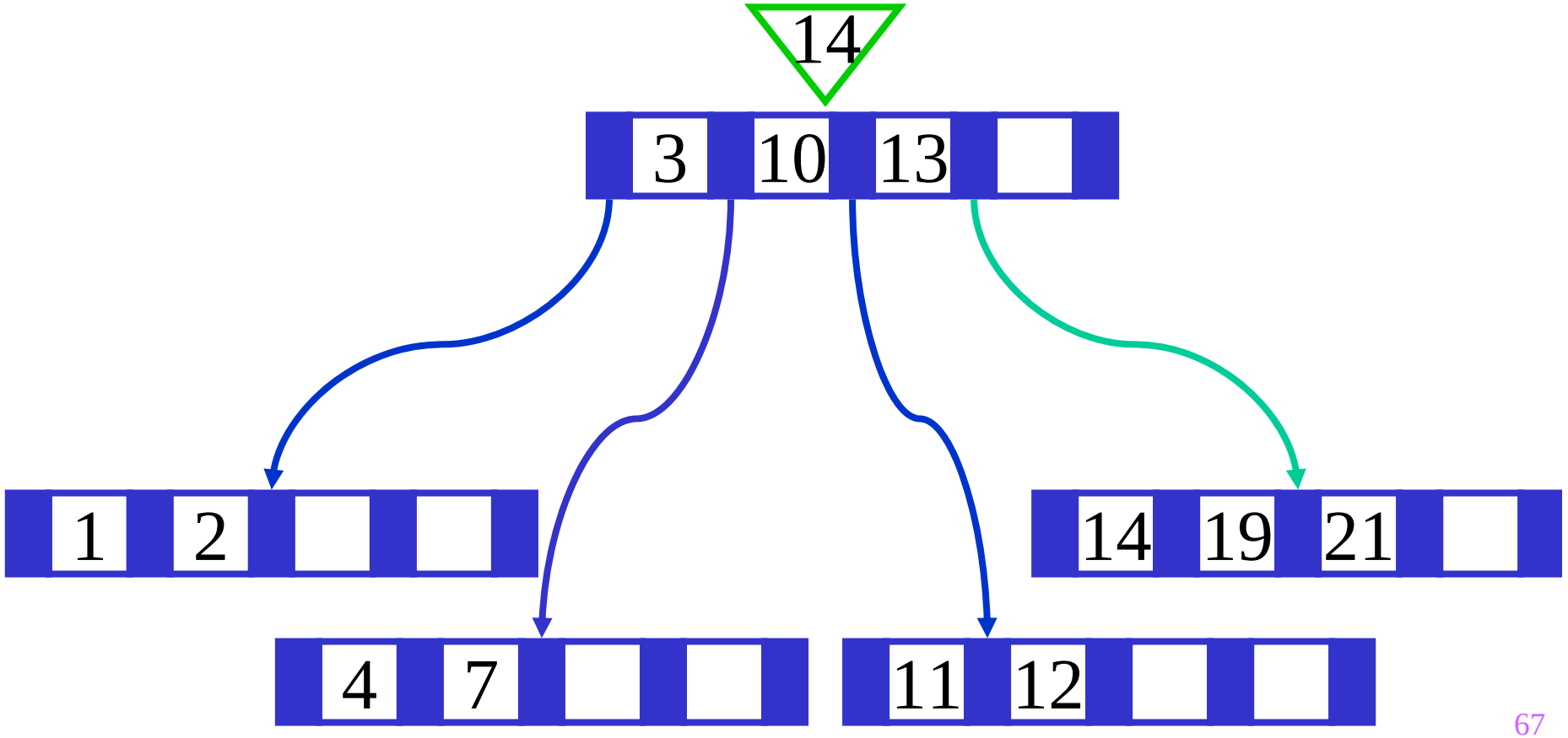
Sukzessiver Aufbau eines B-Baums vom Grad k=2



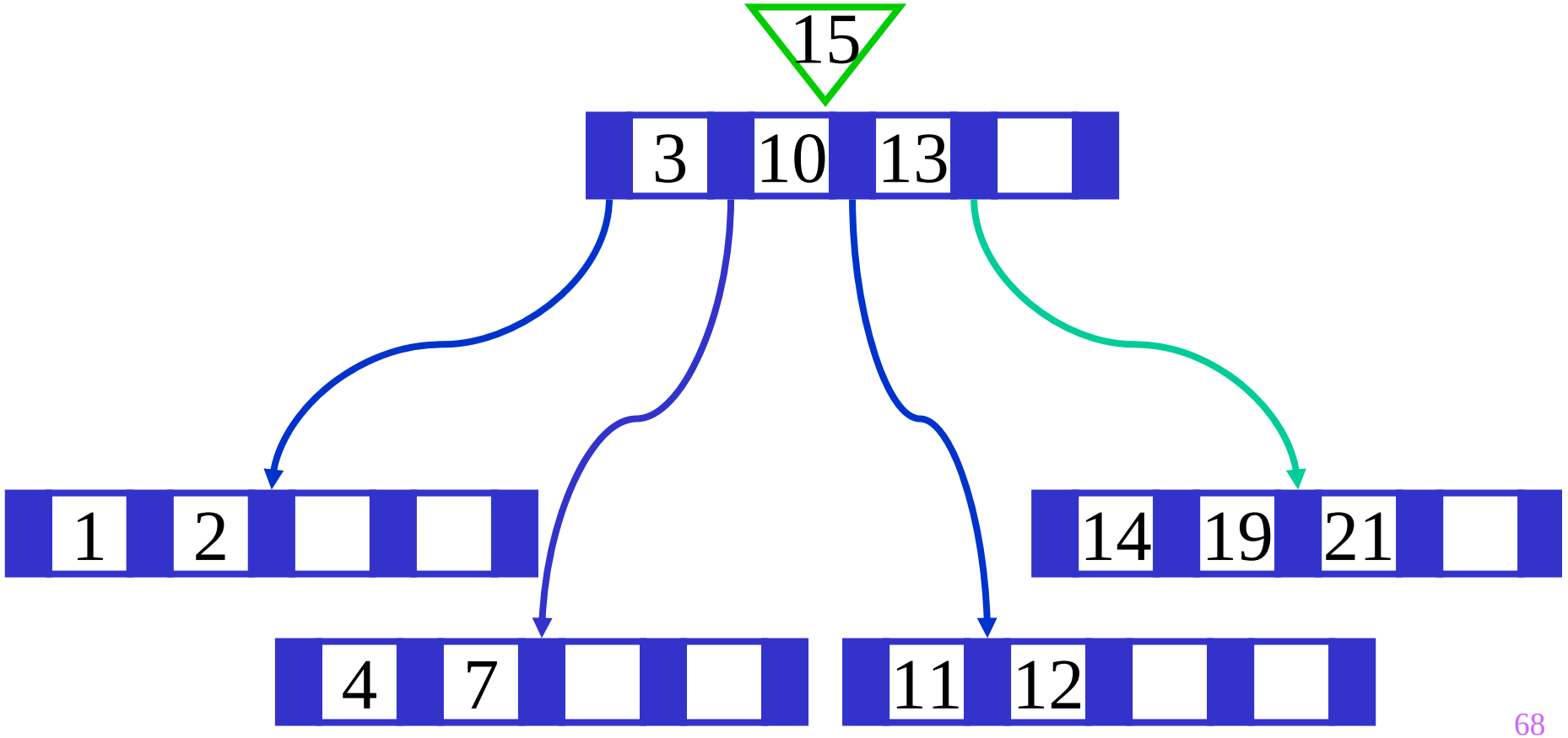
Sukzessiver Aufbau eines B-Baums vom Grad k=2



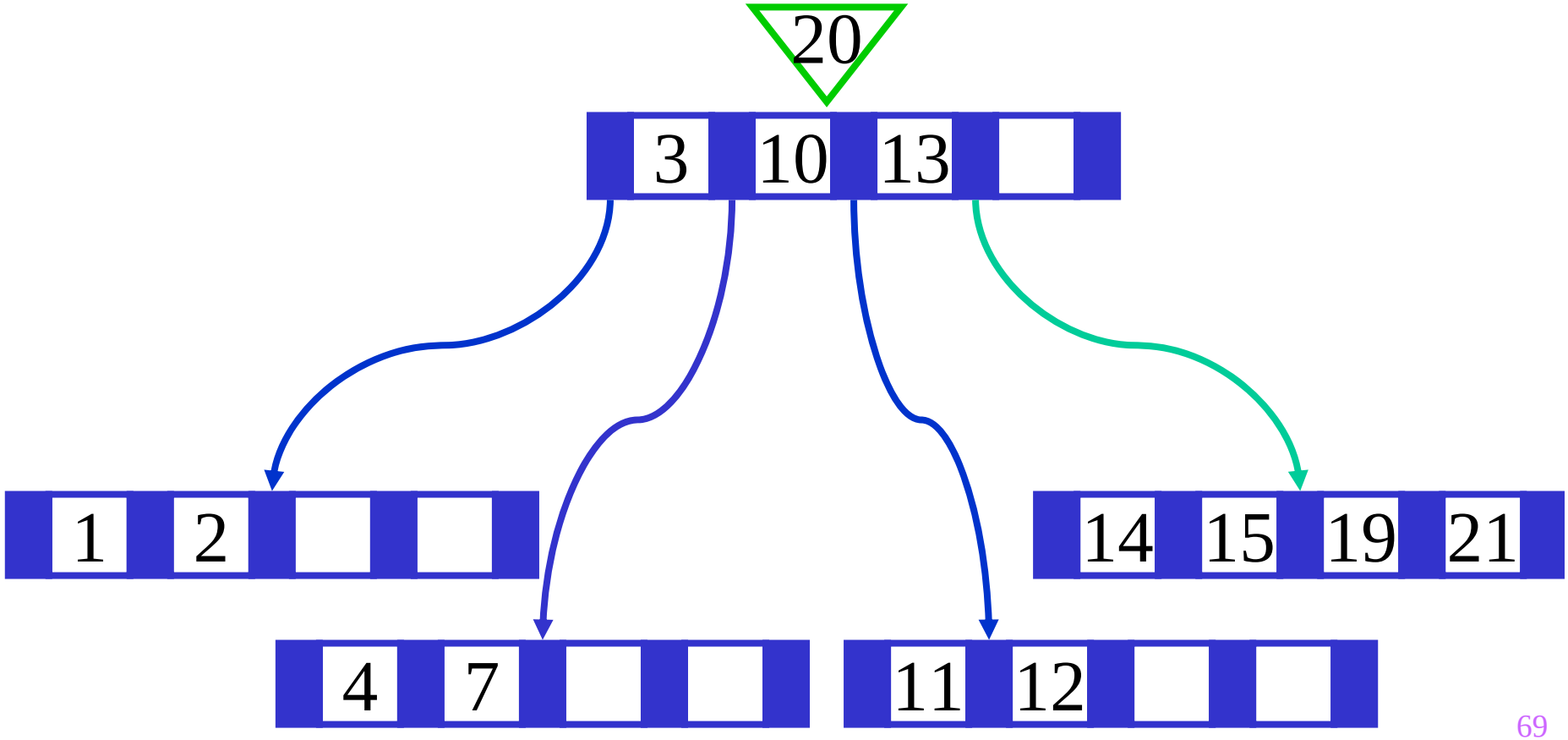
Sukzessiver Aufbau eines B-Baums vom Grad k=2



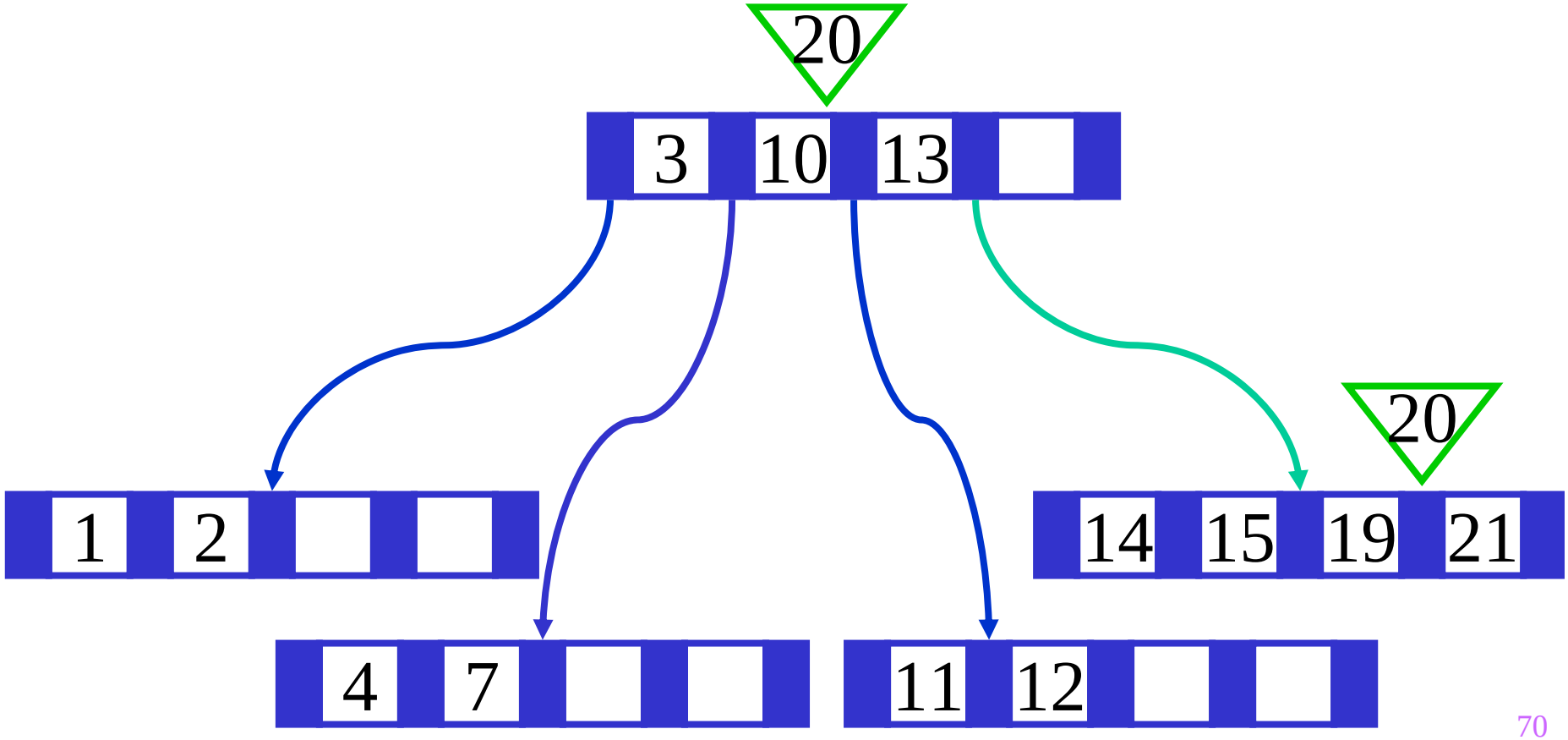
Sukzessiver Aufbau eines B-Baums vom Grad k=2



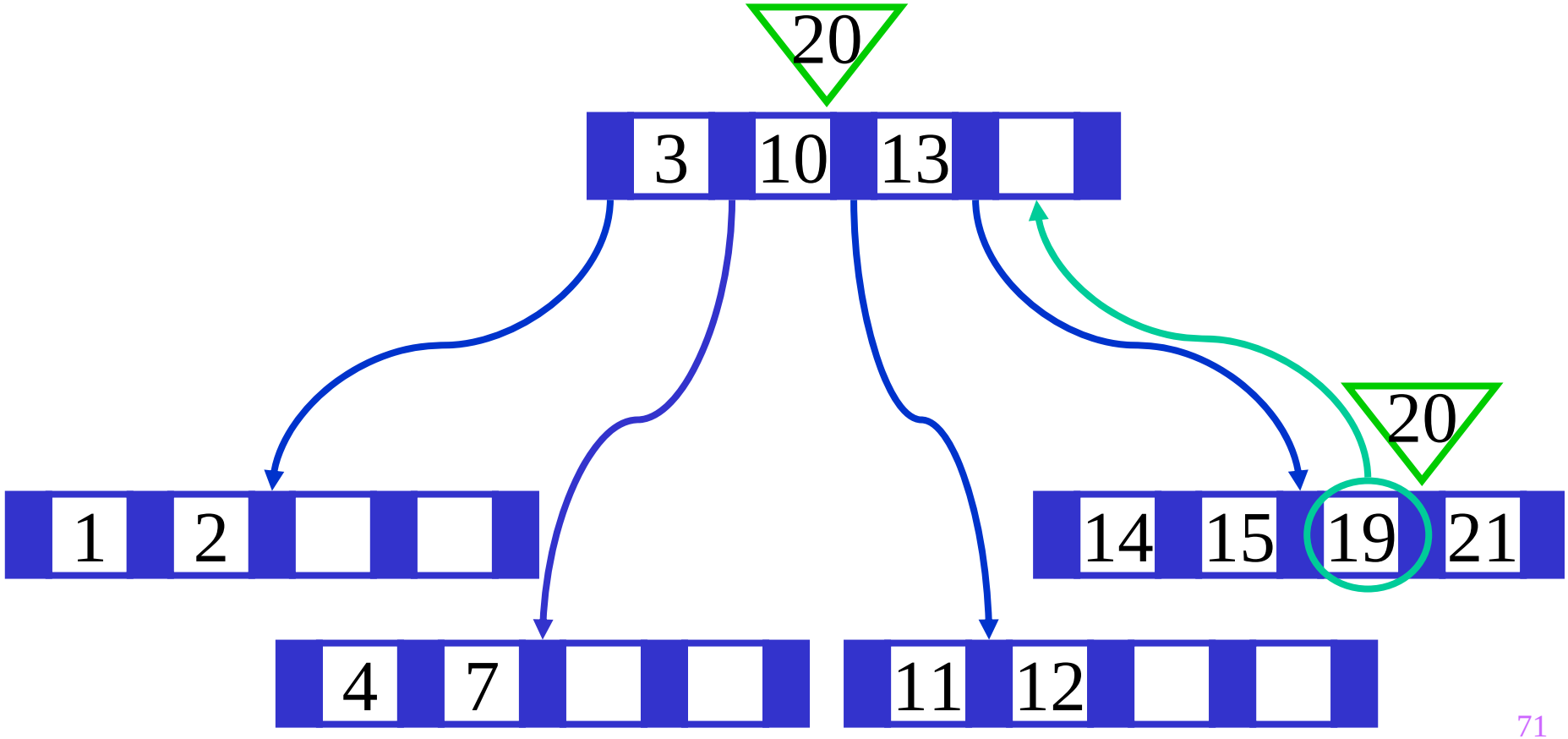
Sukzessiver Aufbau eines B-Baums vom Grad k=2



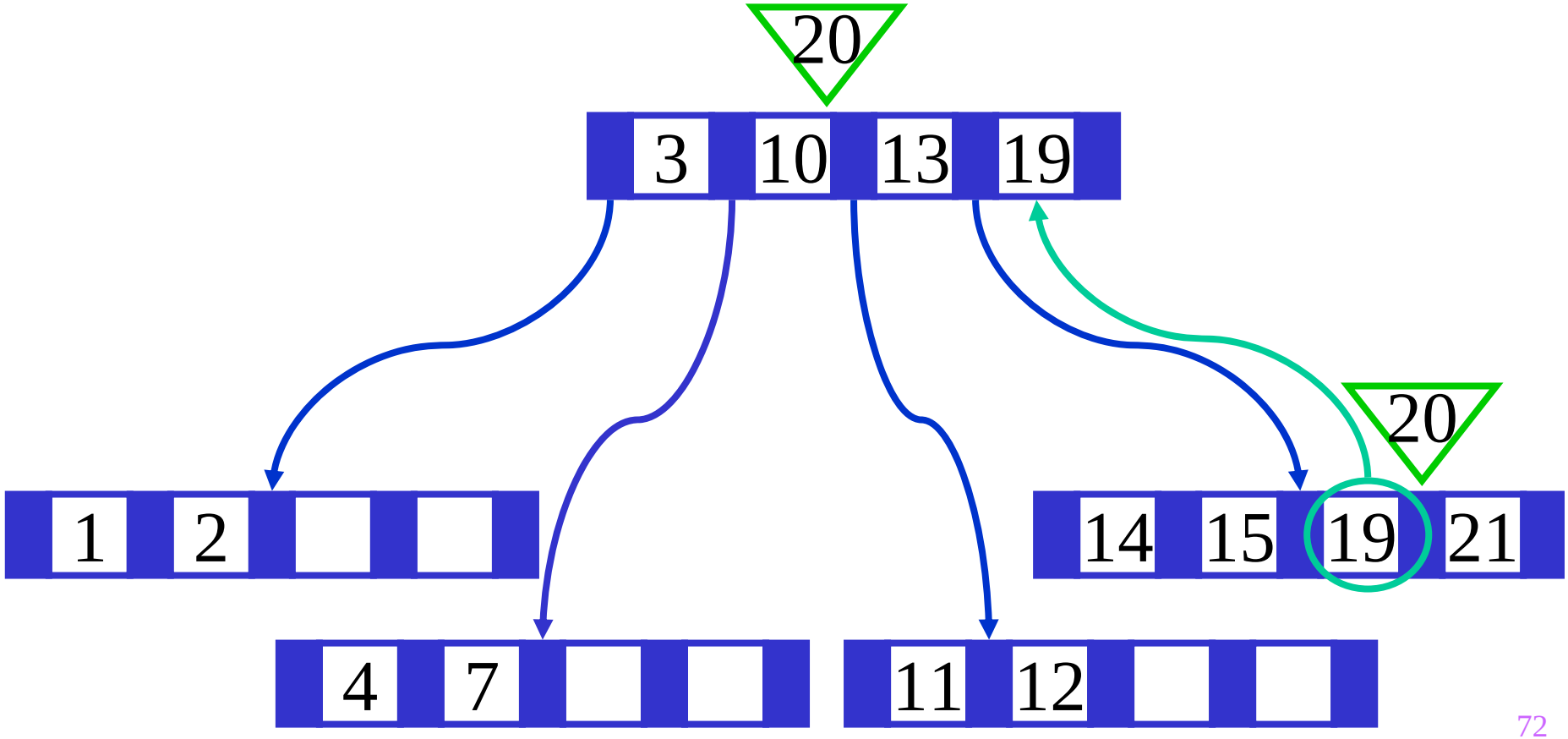
Sukzessiver Aufbau eines B-Baums vom Grad k=2



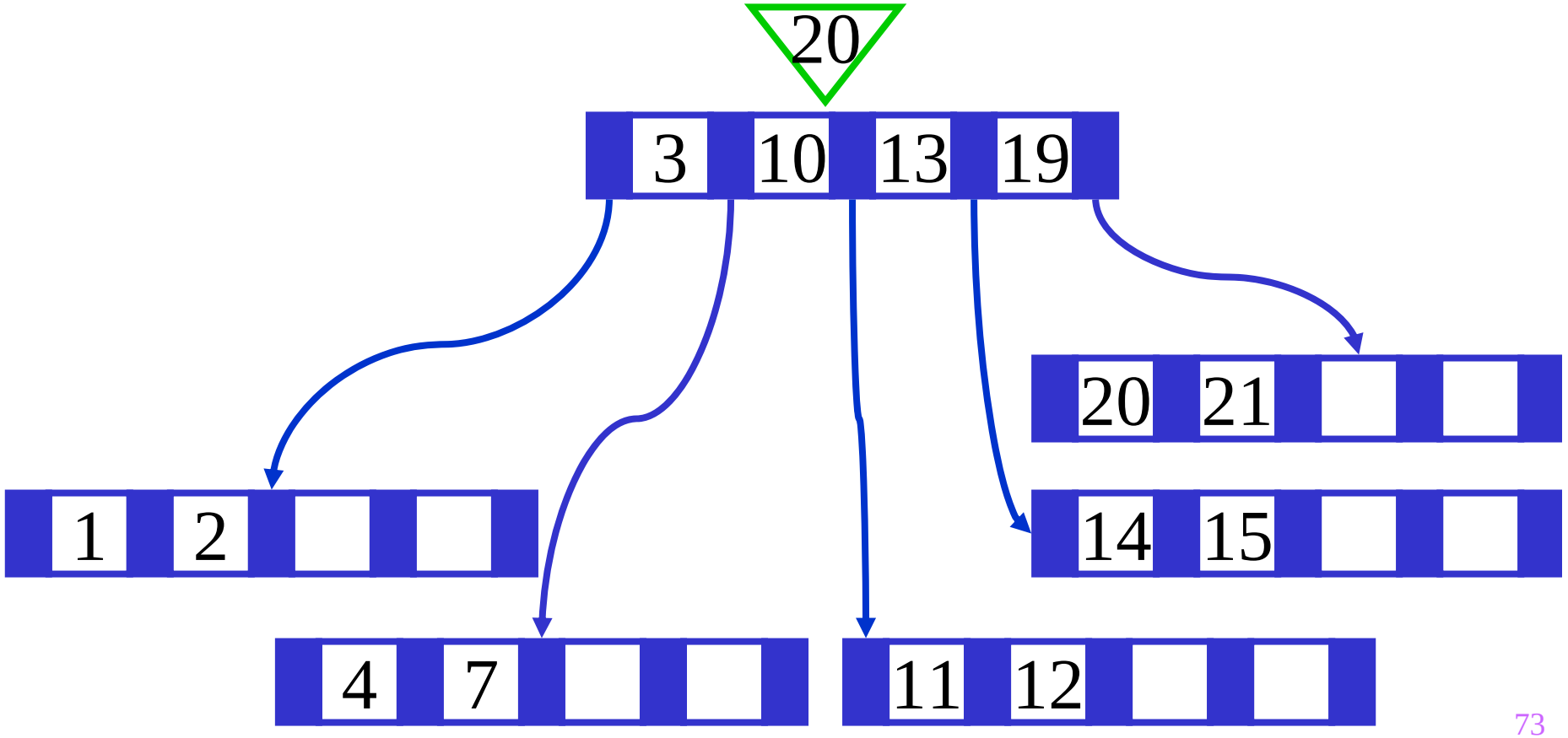
Sukzessiver Aufbau eines B-Baums vom Grad k=2



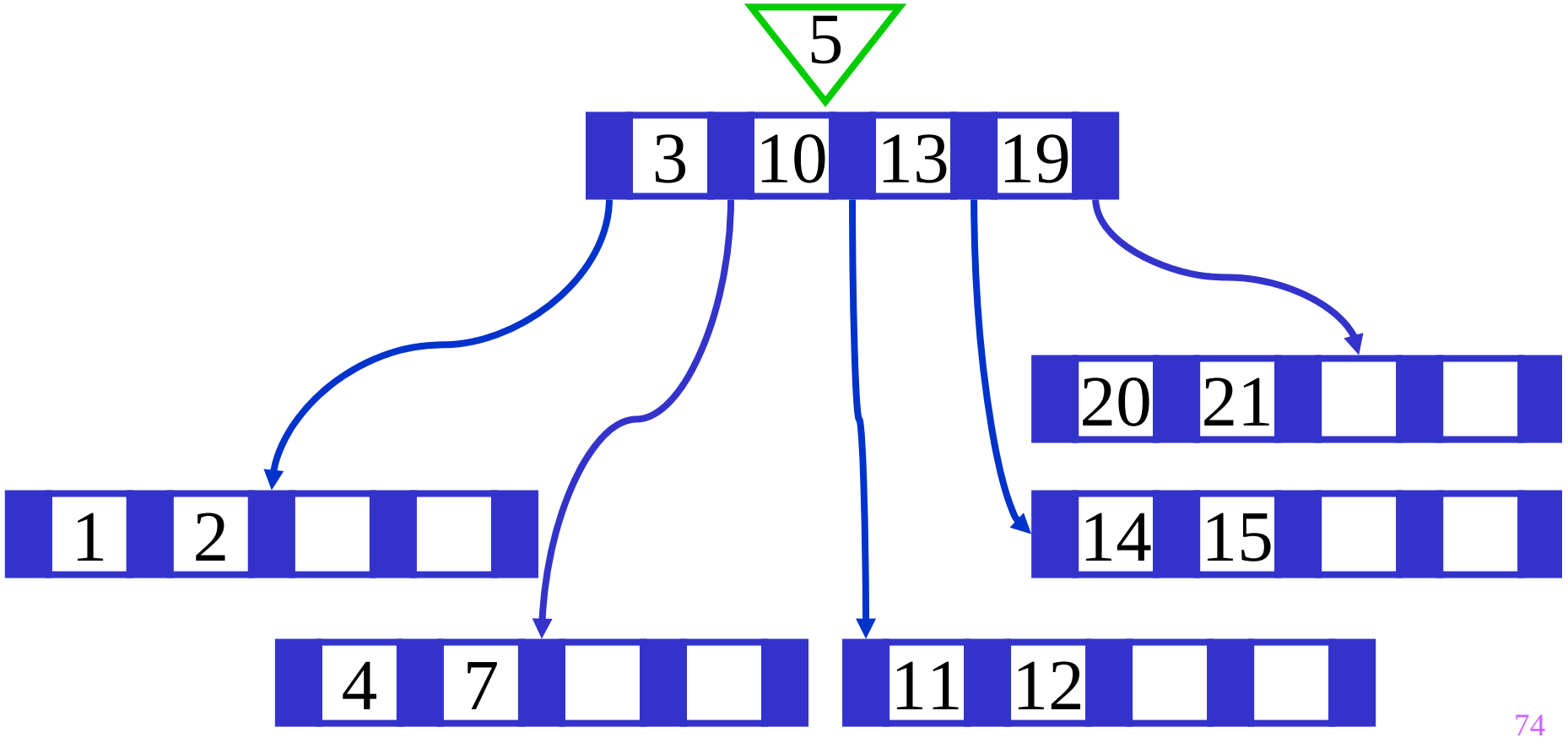
Sukzessiver Aufbau eines B-Baums vom Grad k=2



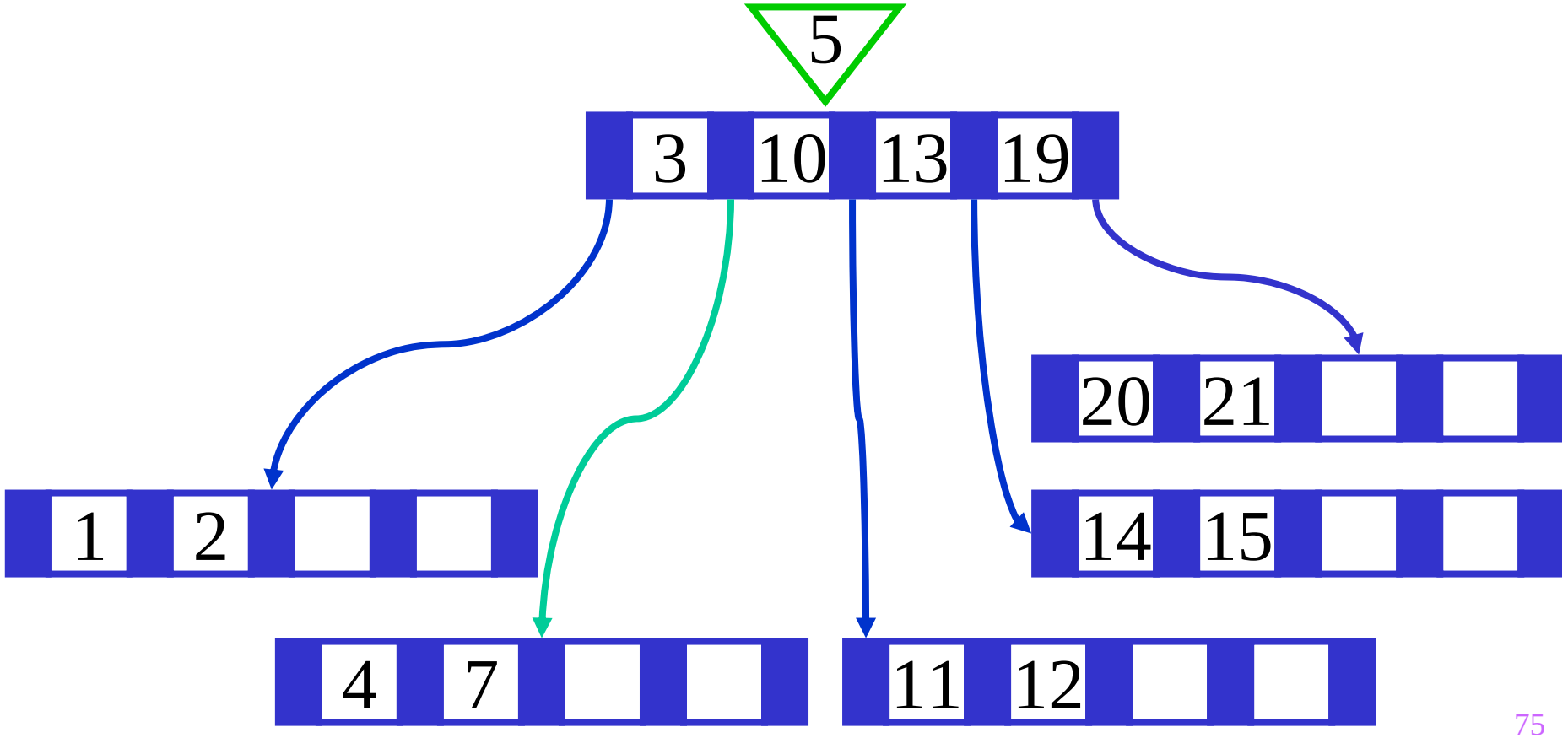
Sukzessiver Aufbau eines B-Baums vom Grad k=2



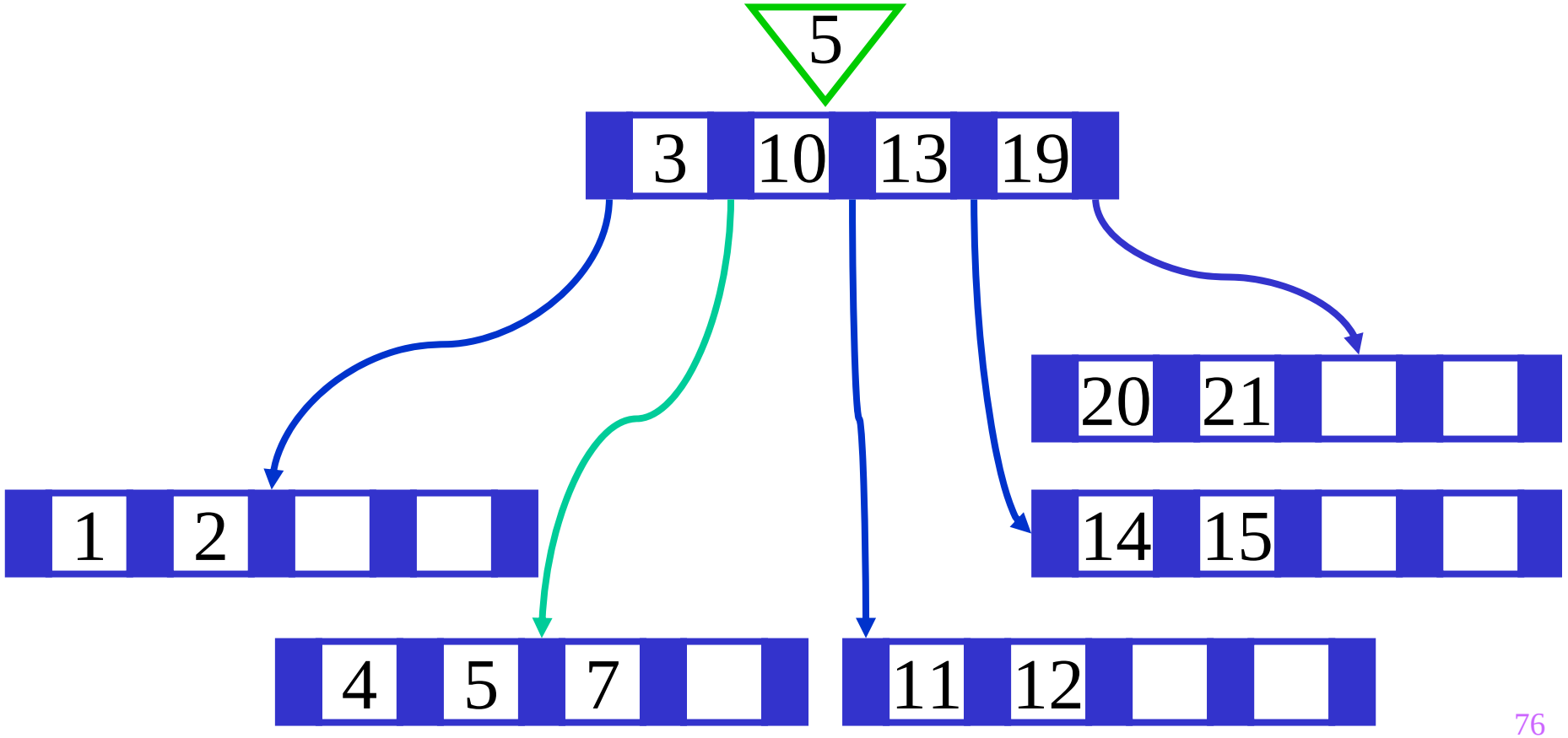
Sukzessiver Aufbau eines B-Baums vom Grad k=2



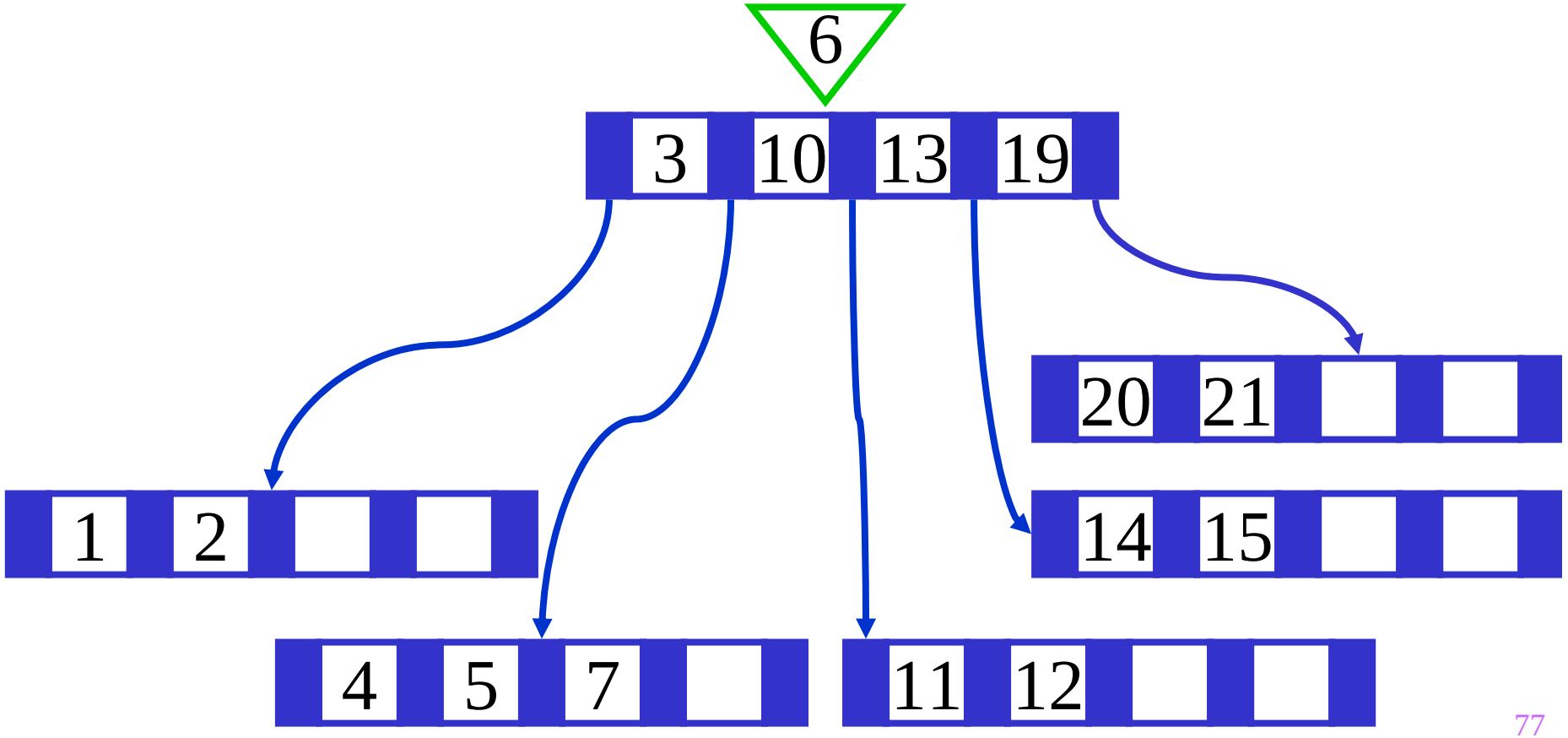
Sukzessiver Aufbau eines B-Baums vom Grad k=2



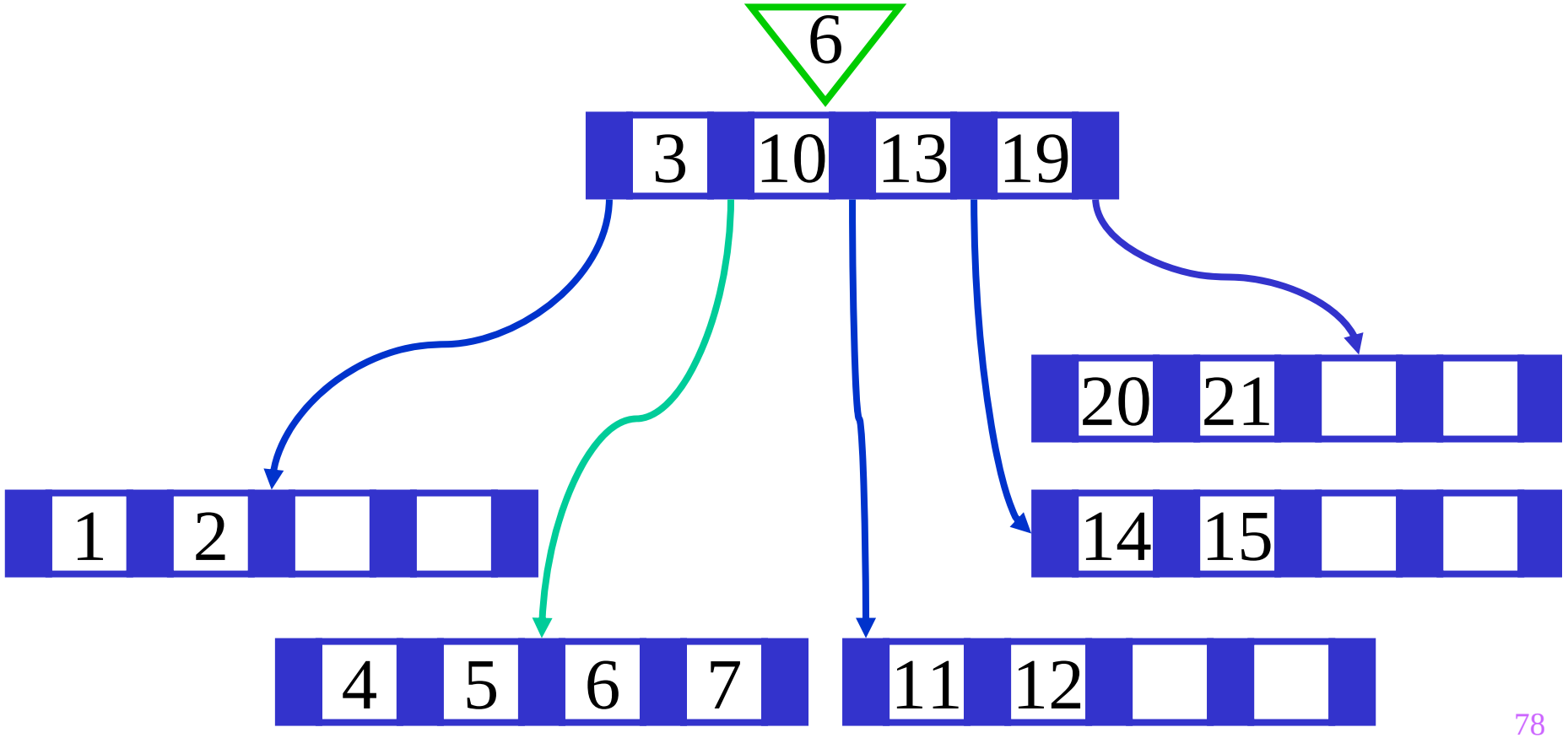
Sukzessiver Aufbau eines B-Baums vom Grad k=2



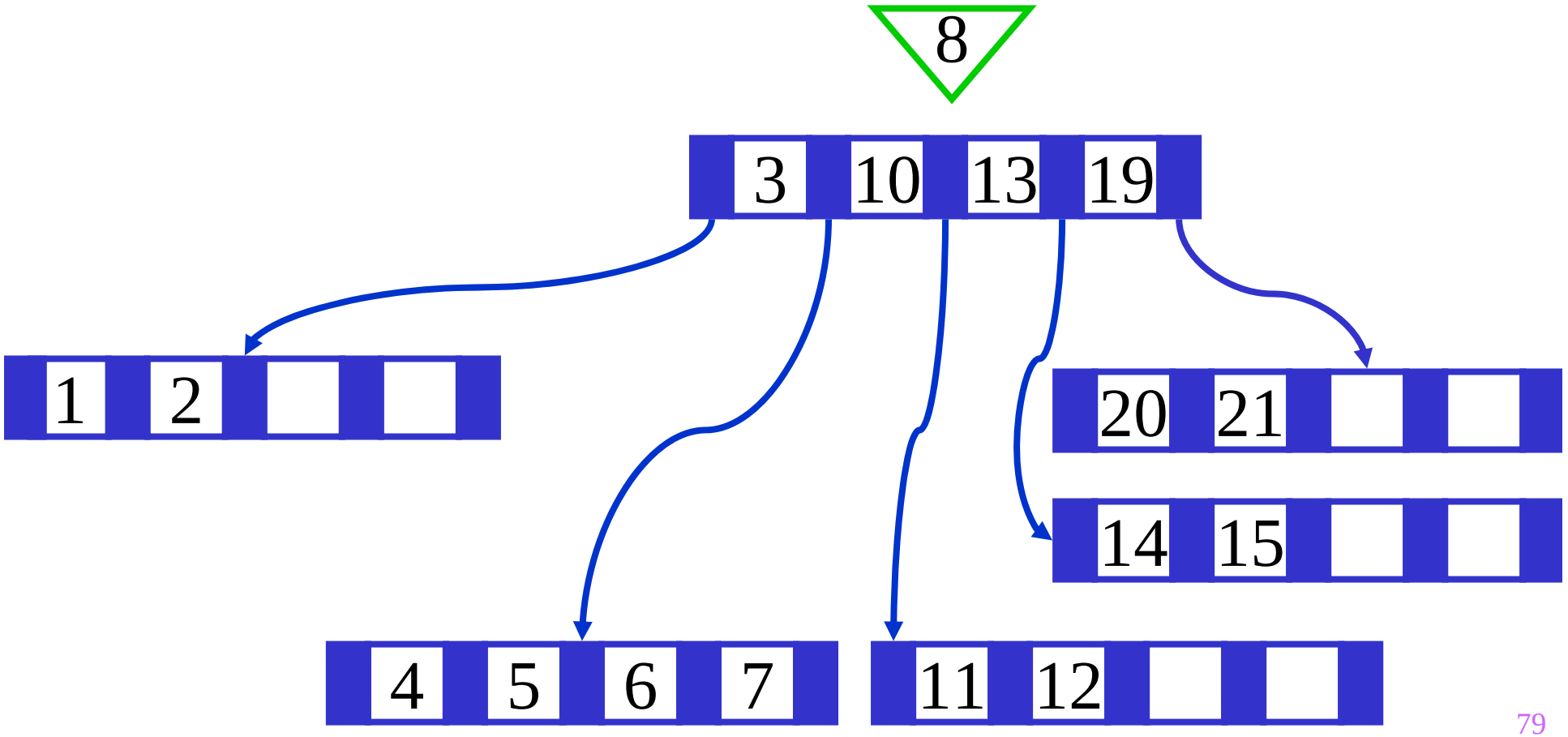
Sukzessiver Aufbau eines B-Baums vom Grad k=2



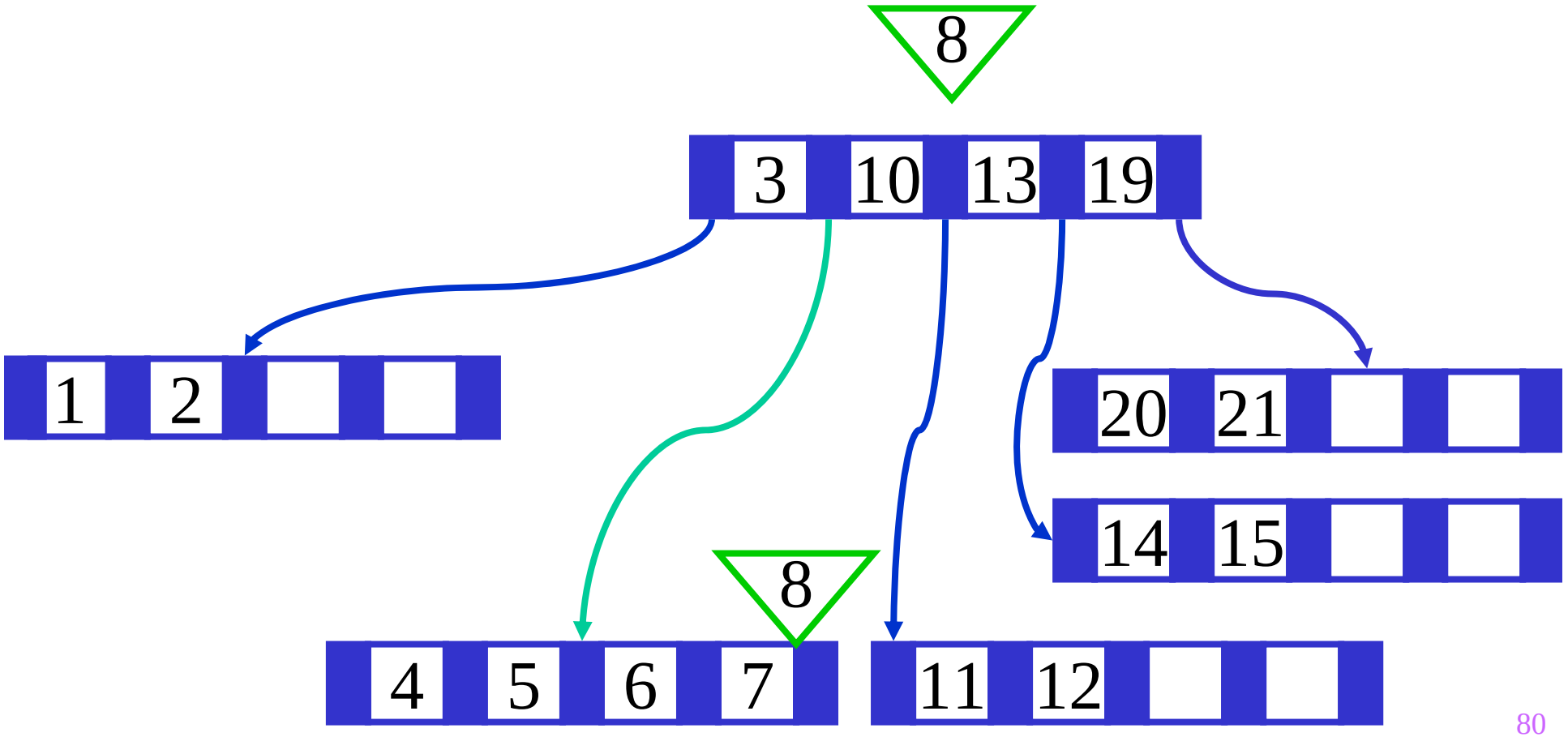
Sukzessiver Aufbau eines B-Baums vom Grad k=2



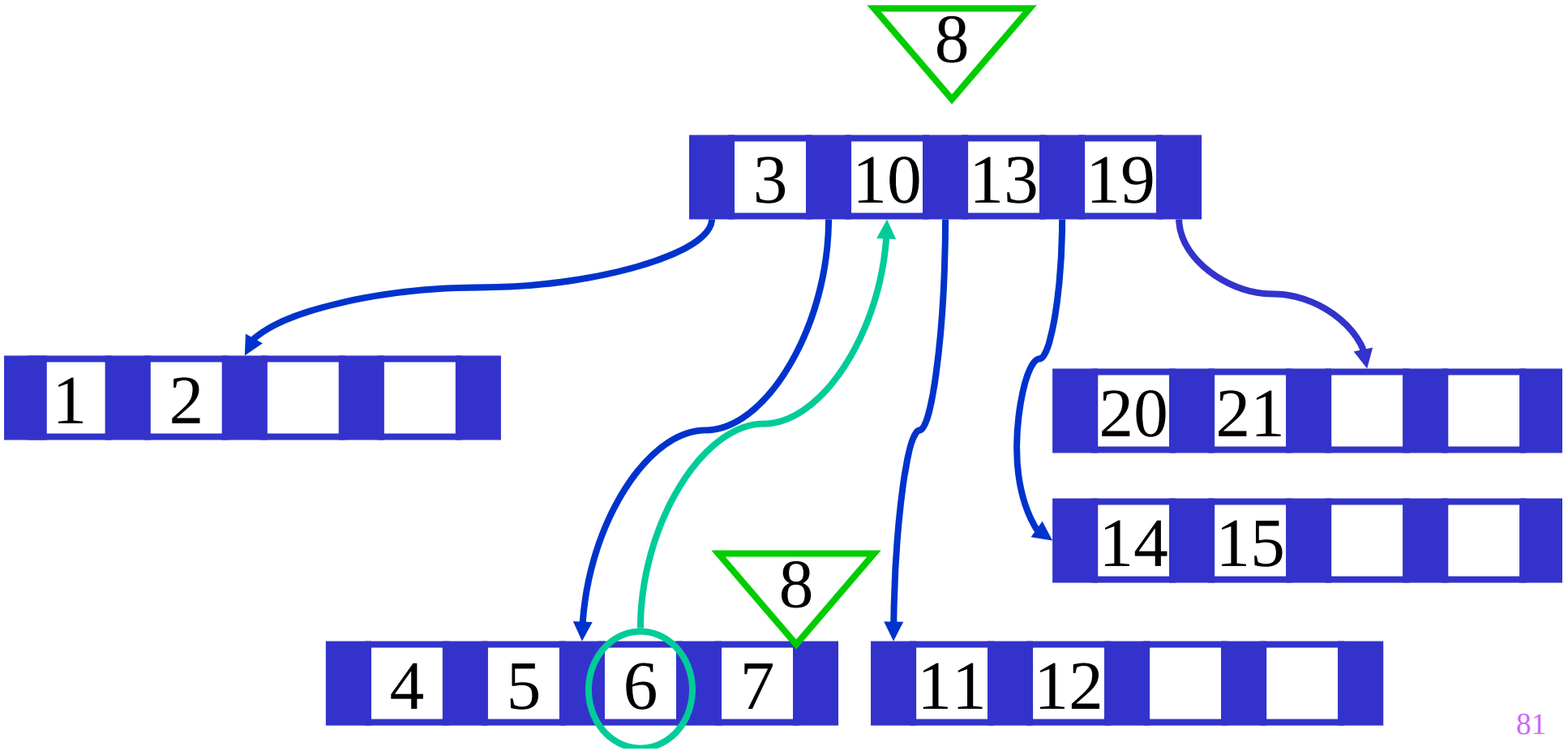
Sukzessiver Aufbau eines B-Baums vom Grad k=2



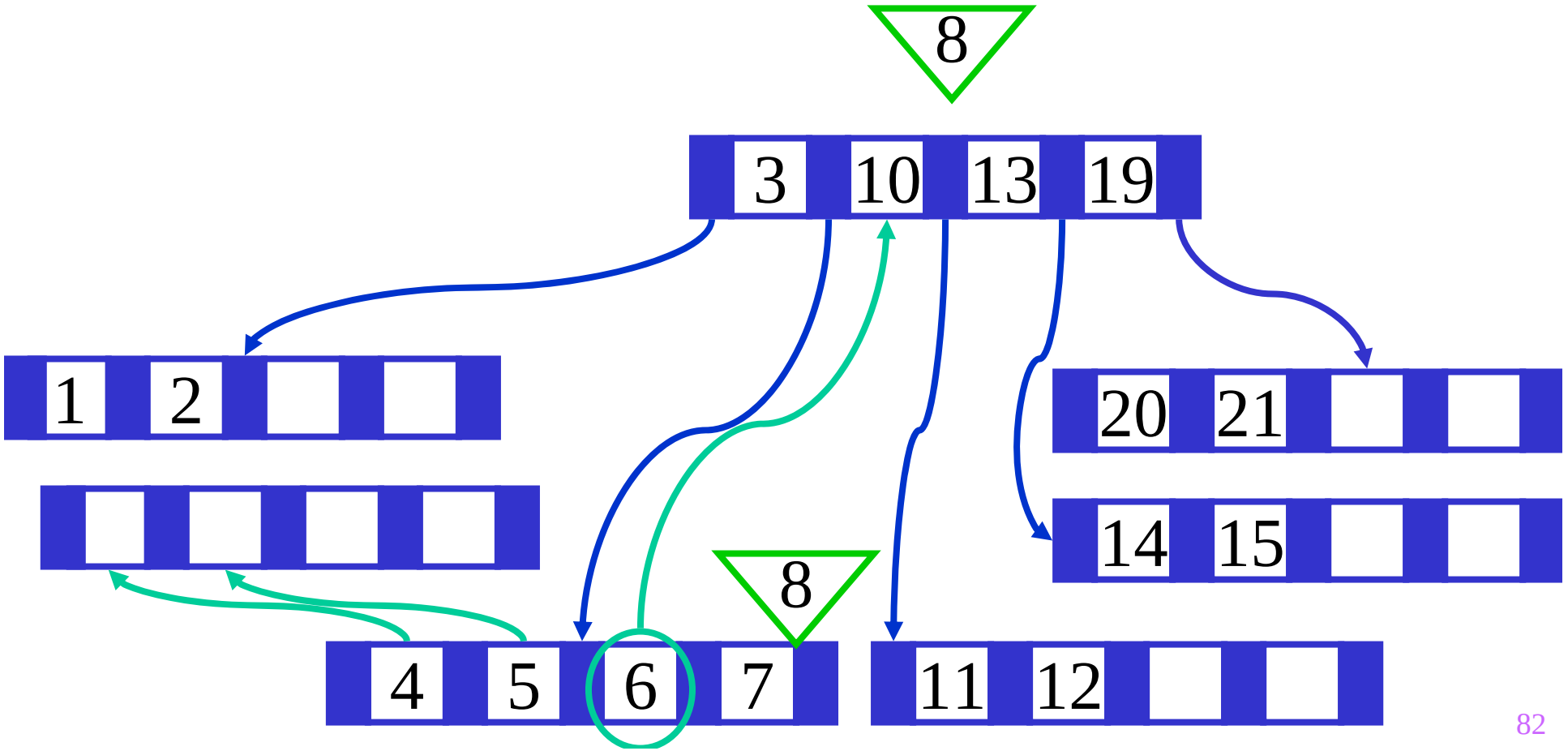
Sukzessiver Aufbau eines B-Baums vom Grad k=2



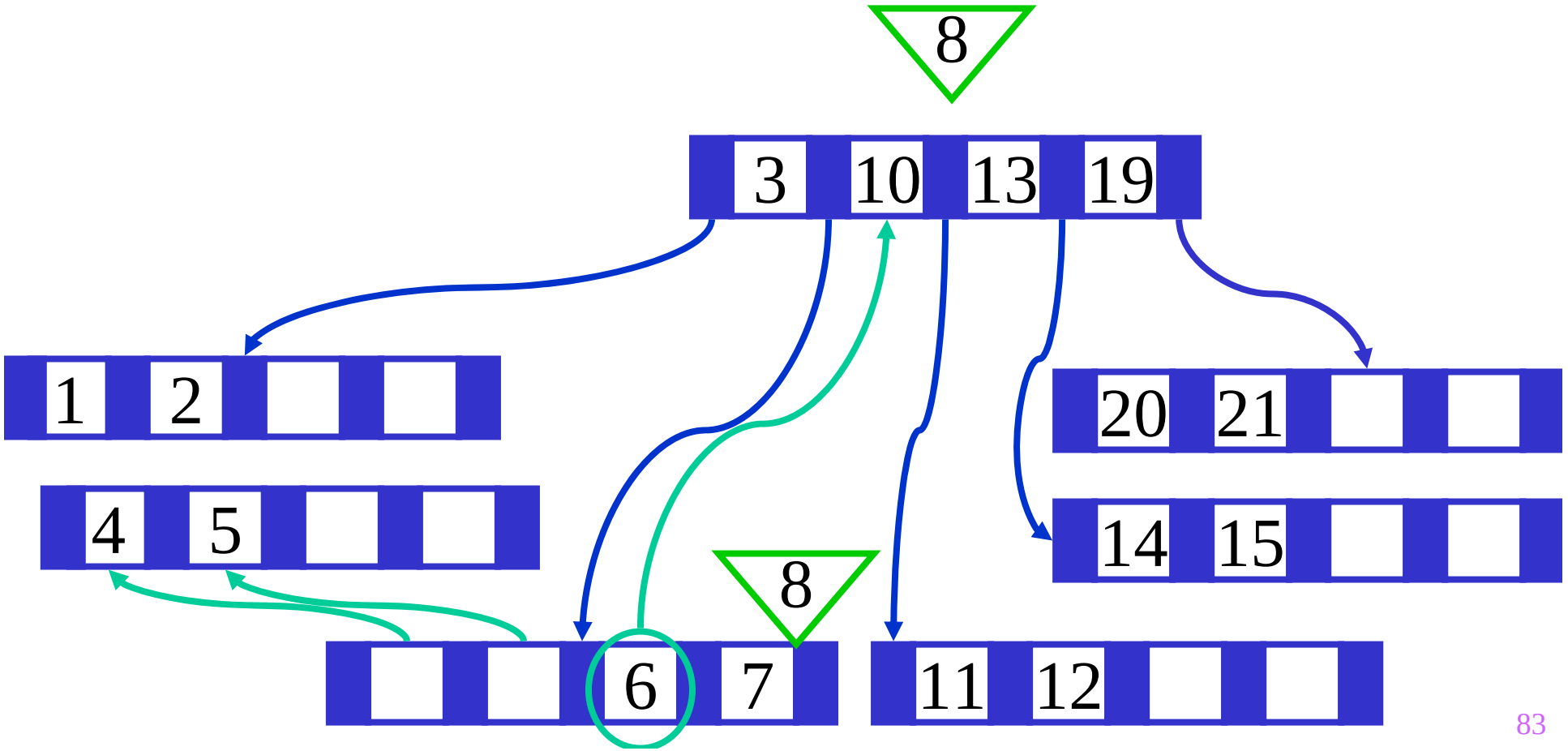
Sukzessiver Aufbau eines B-Baums vom Grad k=2



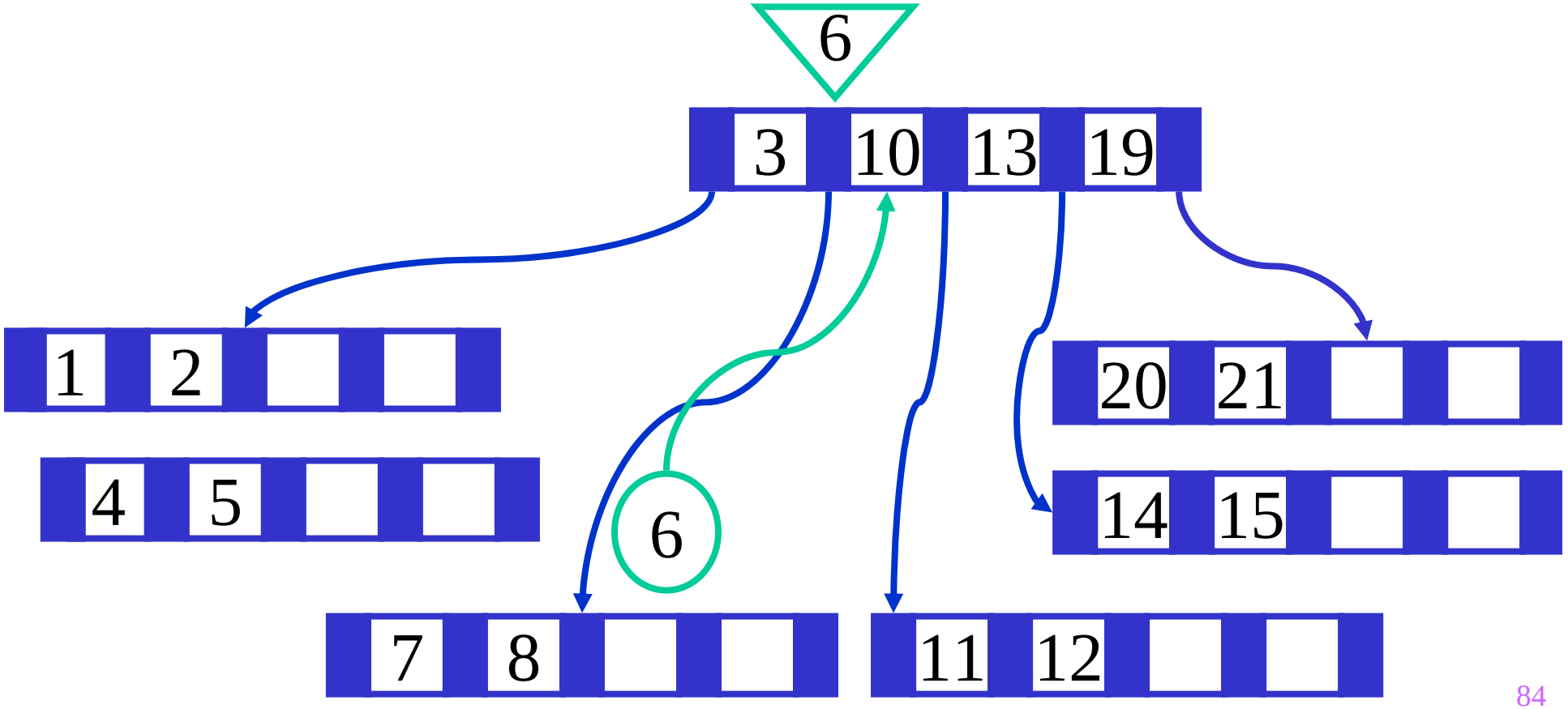
Sukzessiver Aufbau eines B-Baums vom Grad k=2



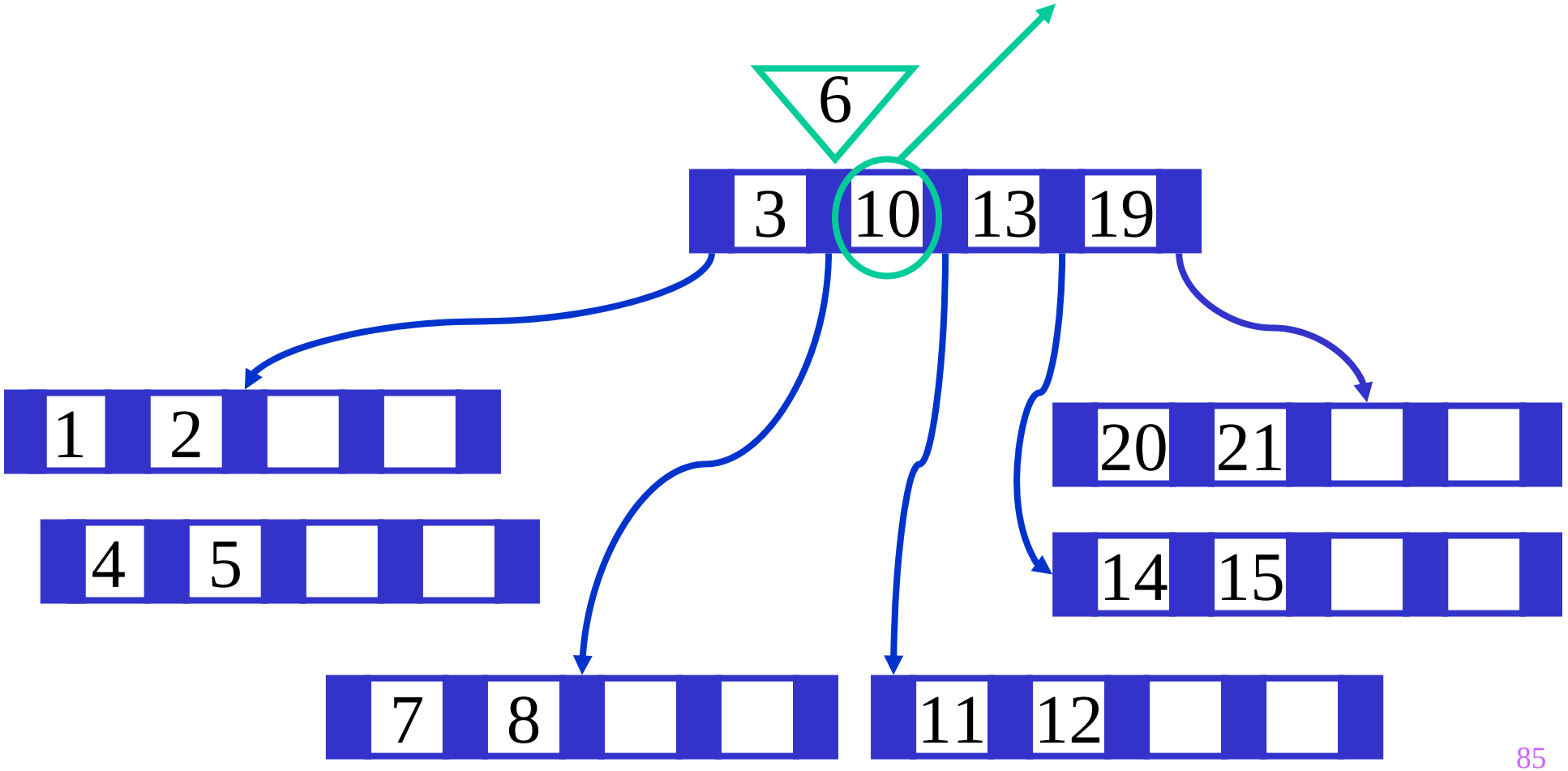
Sukzessiver Aufbau eines B-Baums vom Grad k=2



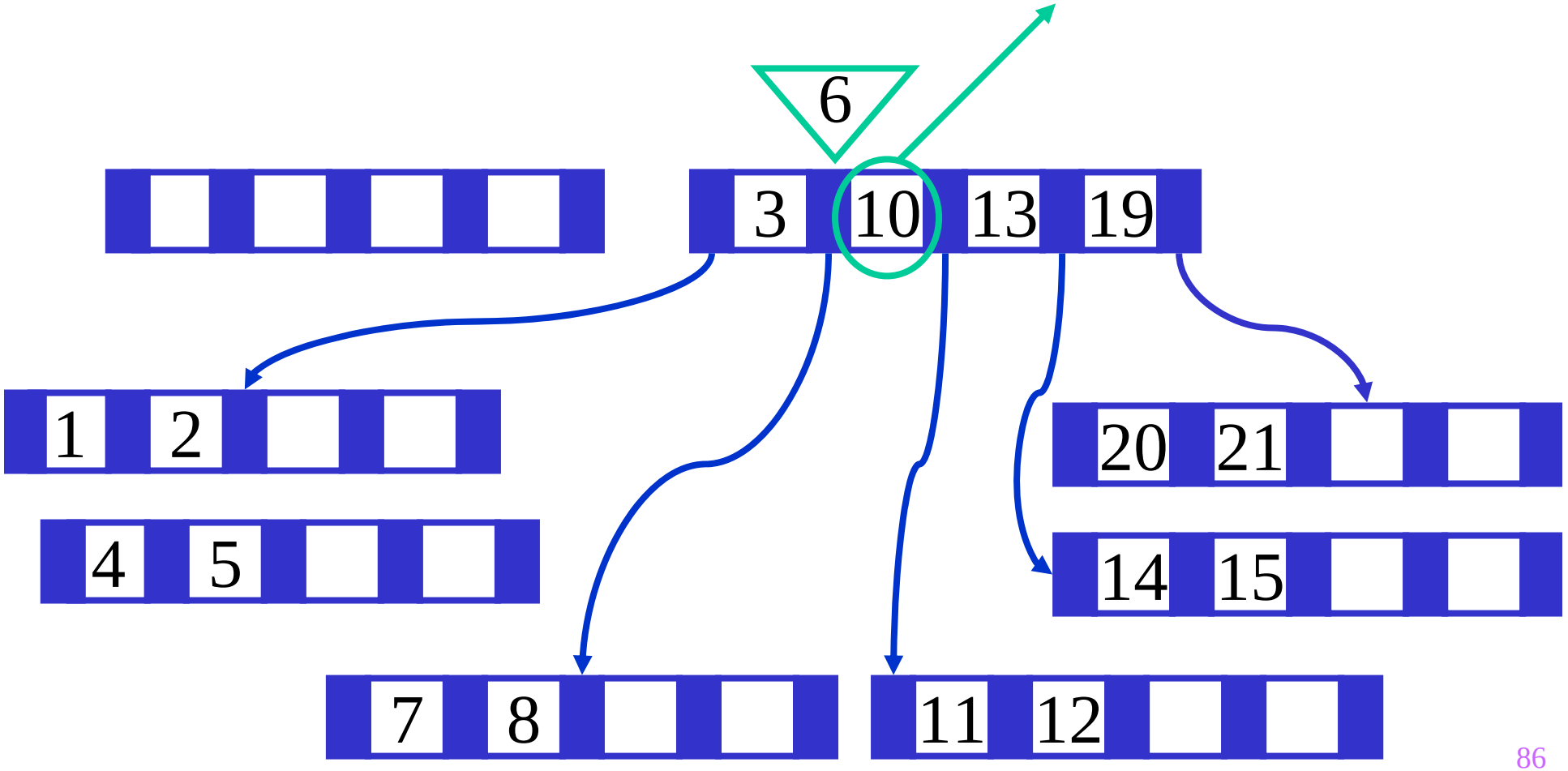
Sukzessiver Aufbau eines B-Baums vom Grad k=2



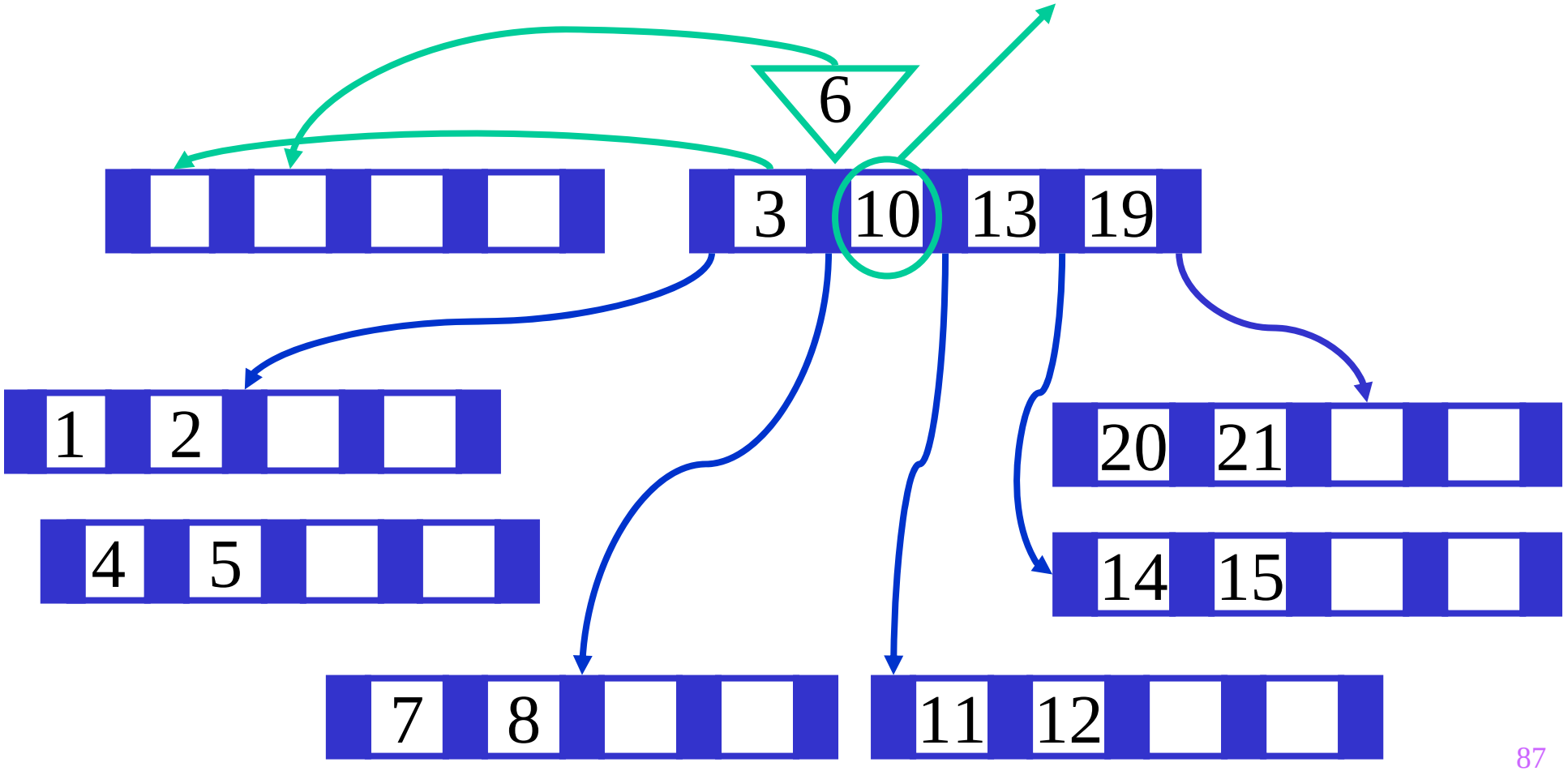
Sukzessiver Aufbau eines B-Baums vom Grad k=2



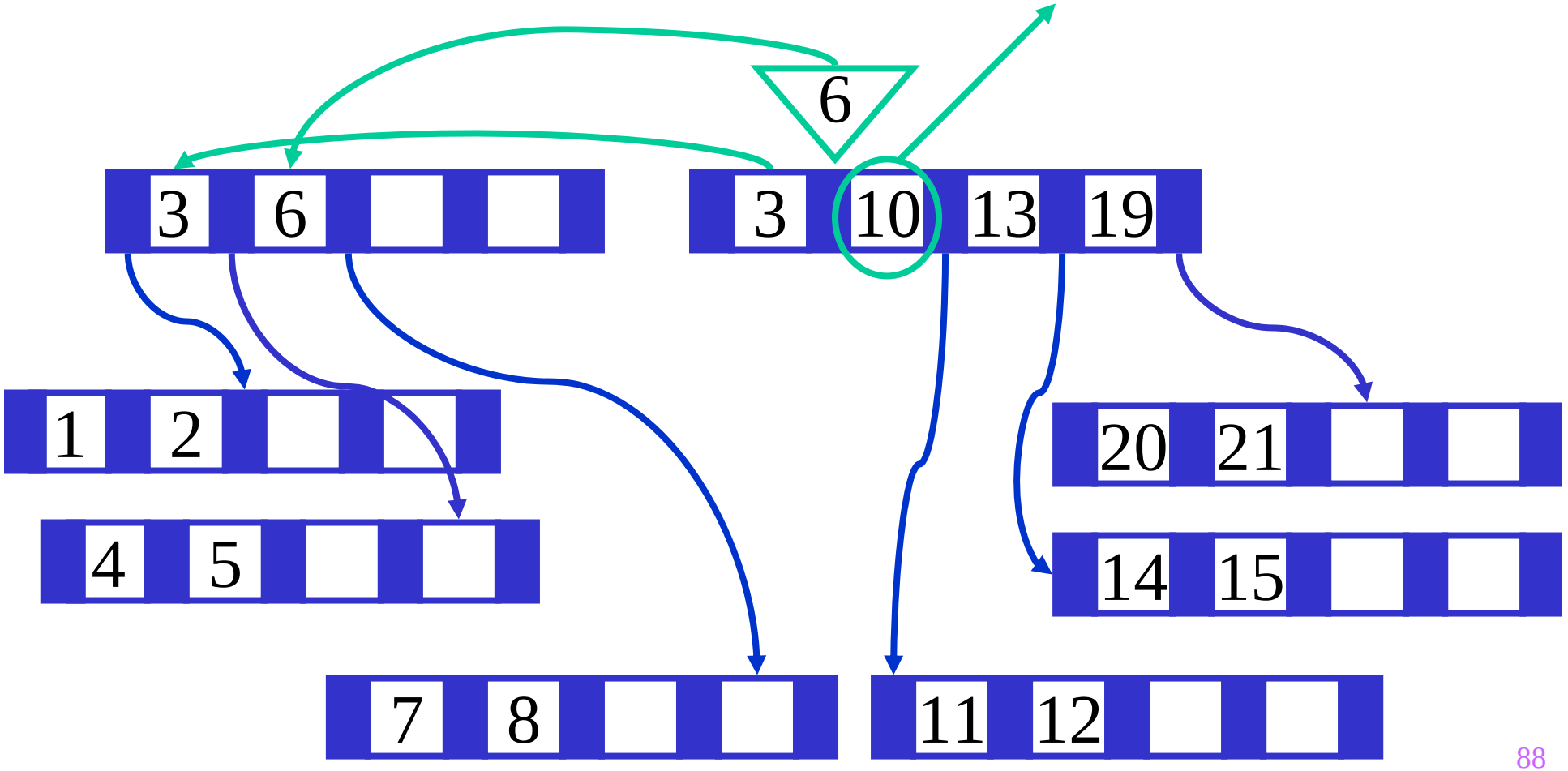
Sukzessiver Aufbau eines B-Baums vom Grad k=2



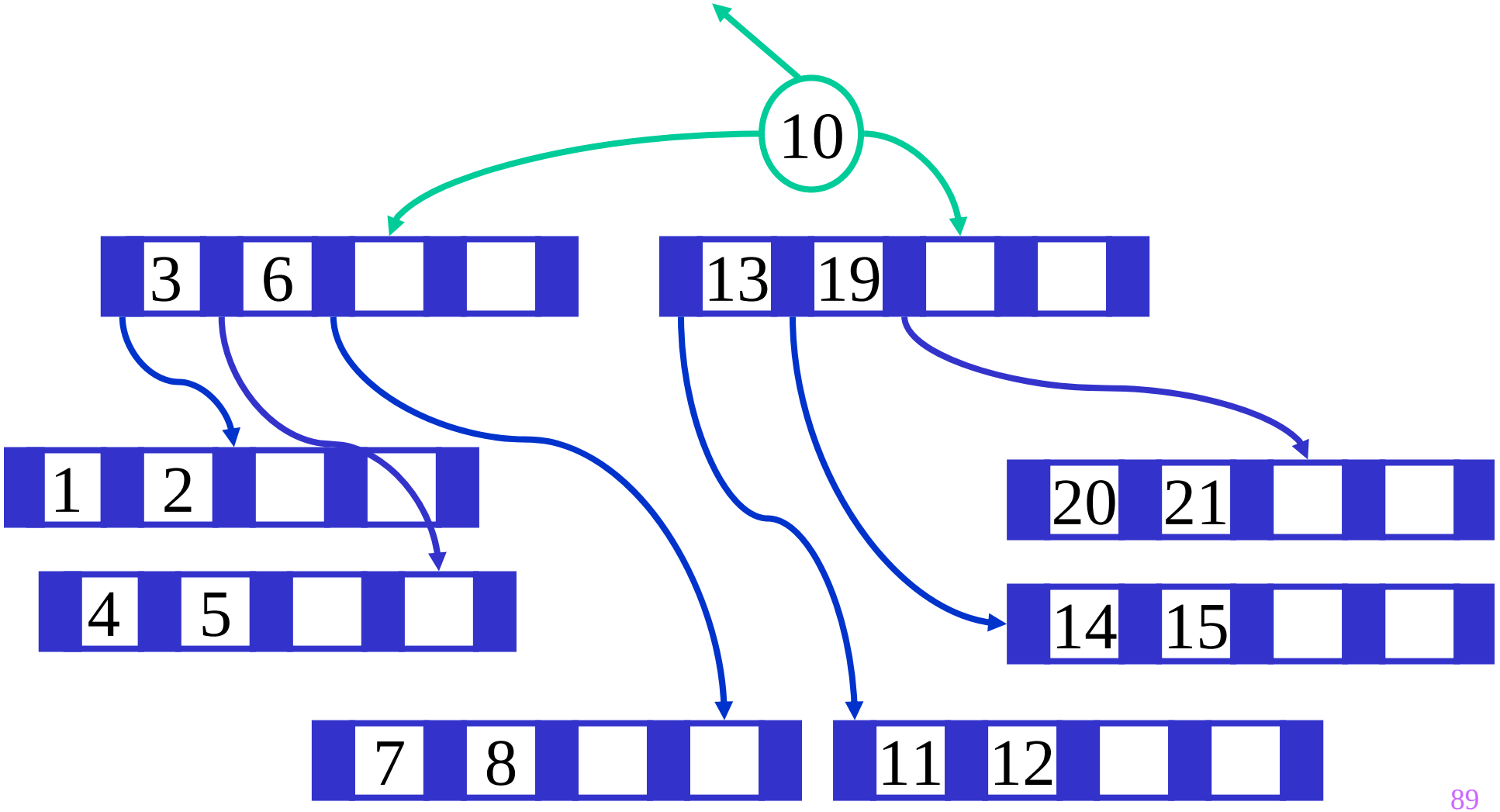
Sukzessiver Aufbau eines B-Baums vom Grad k=2



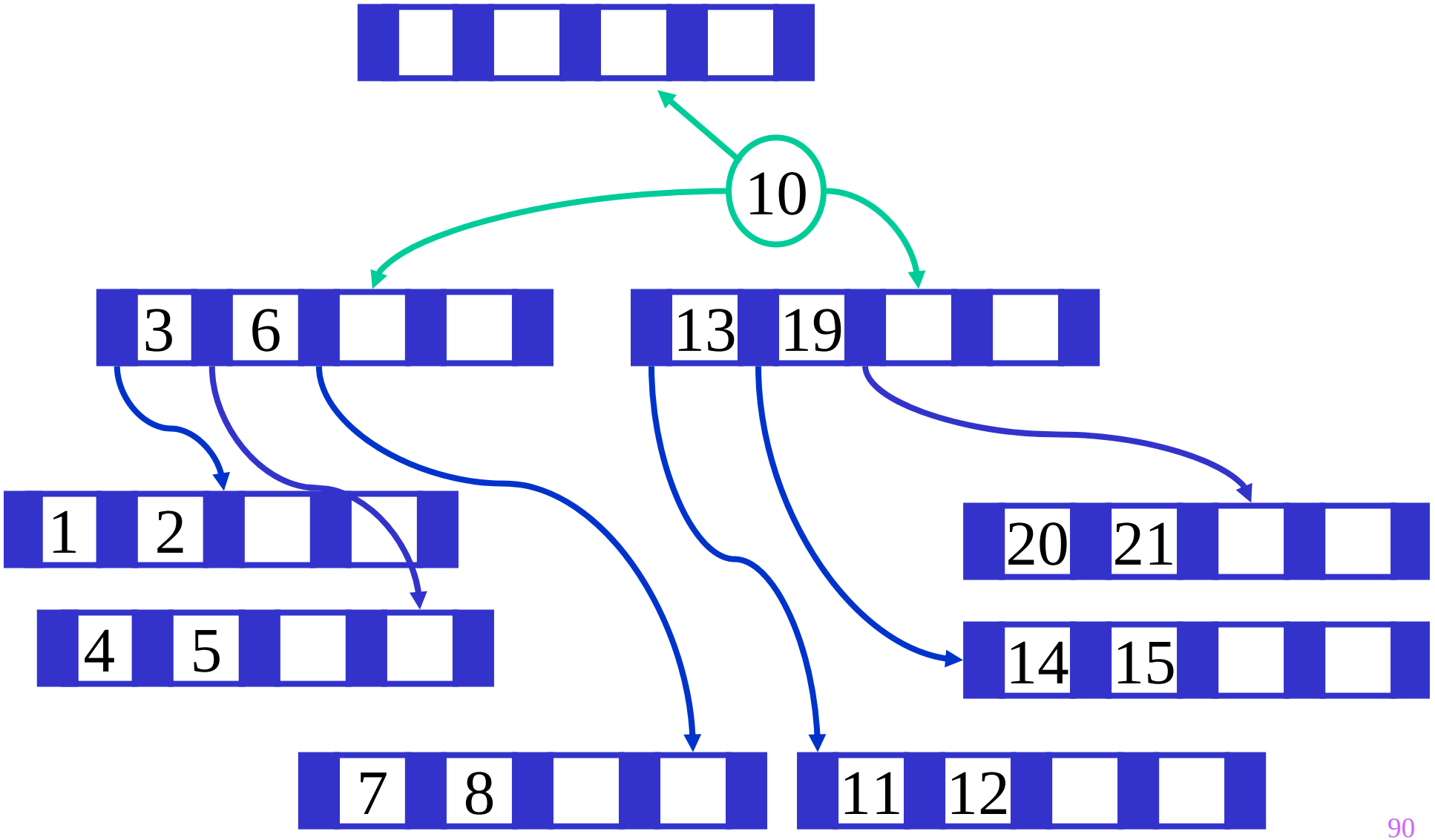
Sukzessiver Aufbau eines B-Baums vom Grad k=2



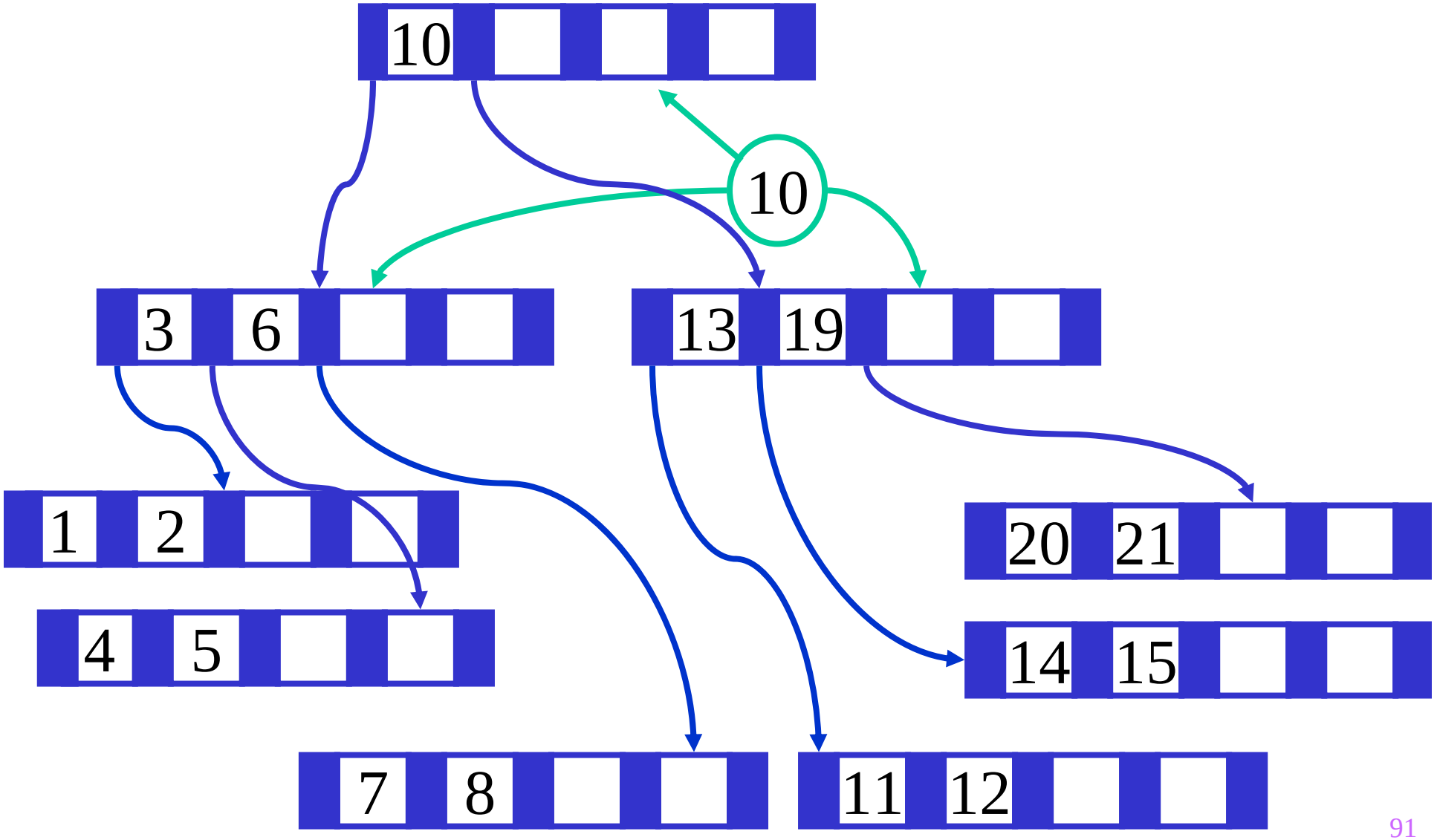
Sukzessiver Aufbau eines B-Baums vom Grad k=2



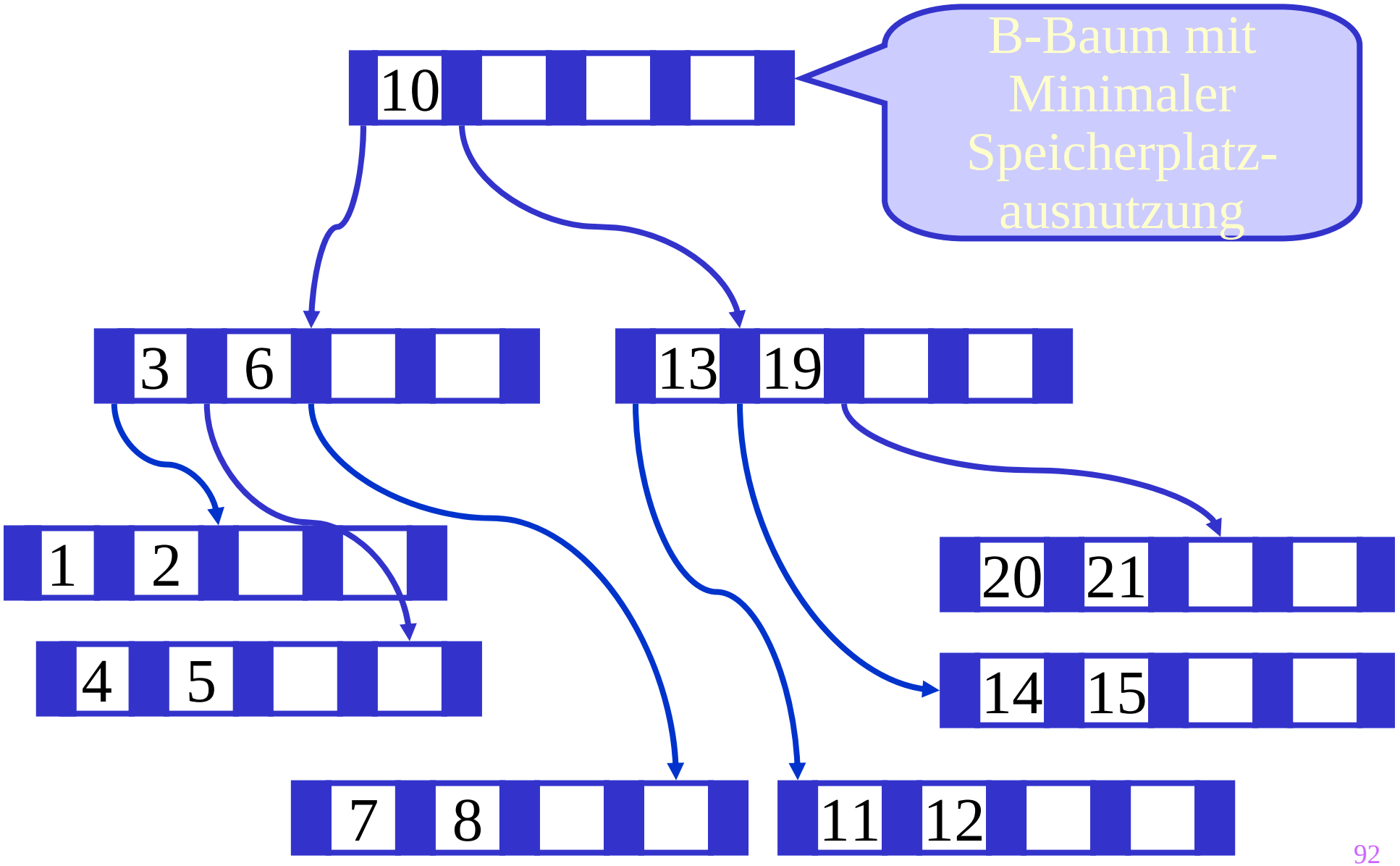
Sukzessiver Aufbau eines B-Baums vom Grad k=2



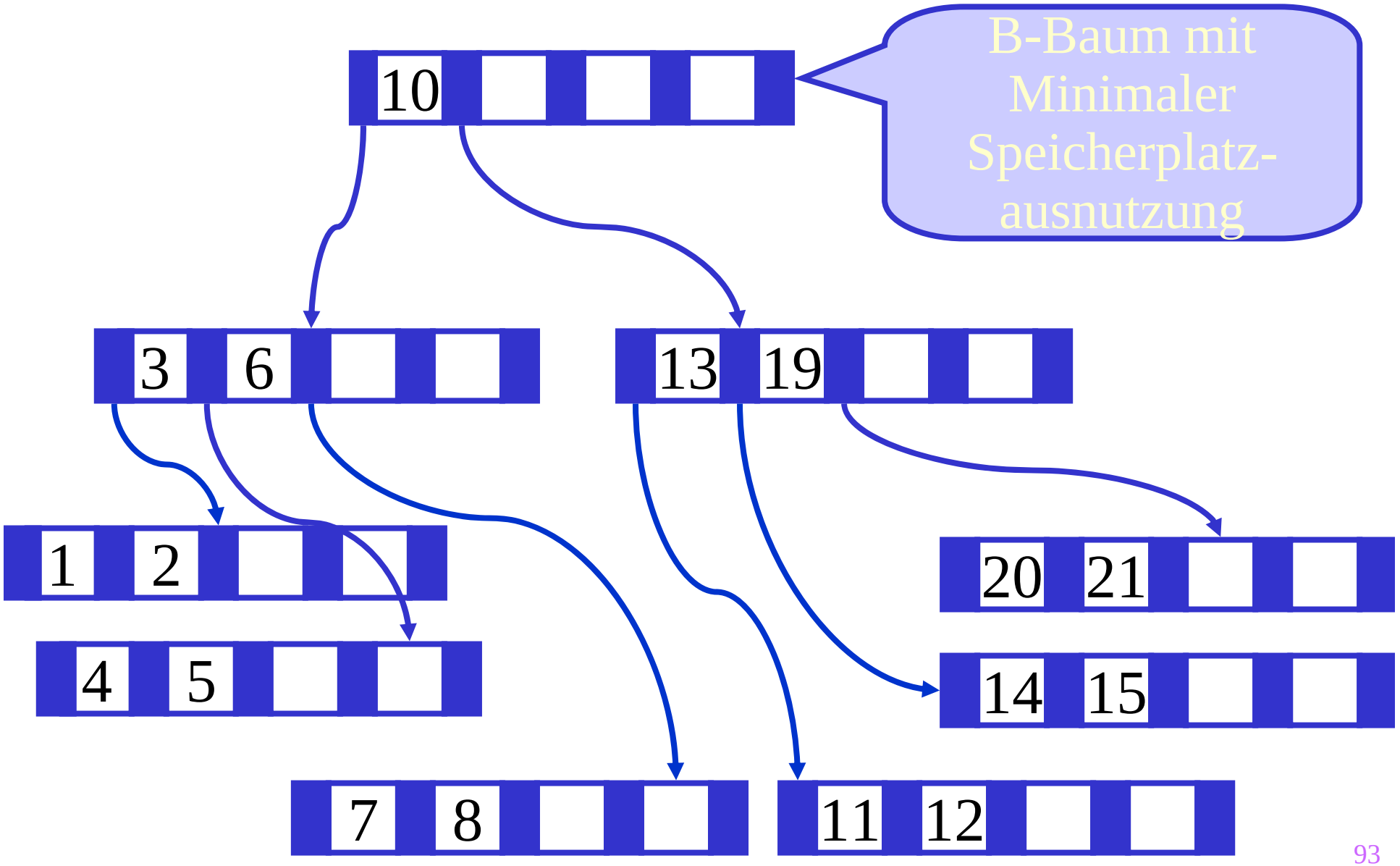
Sukzessiver Aufbau eines B-Baums vom Grad k=2



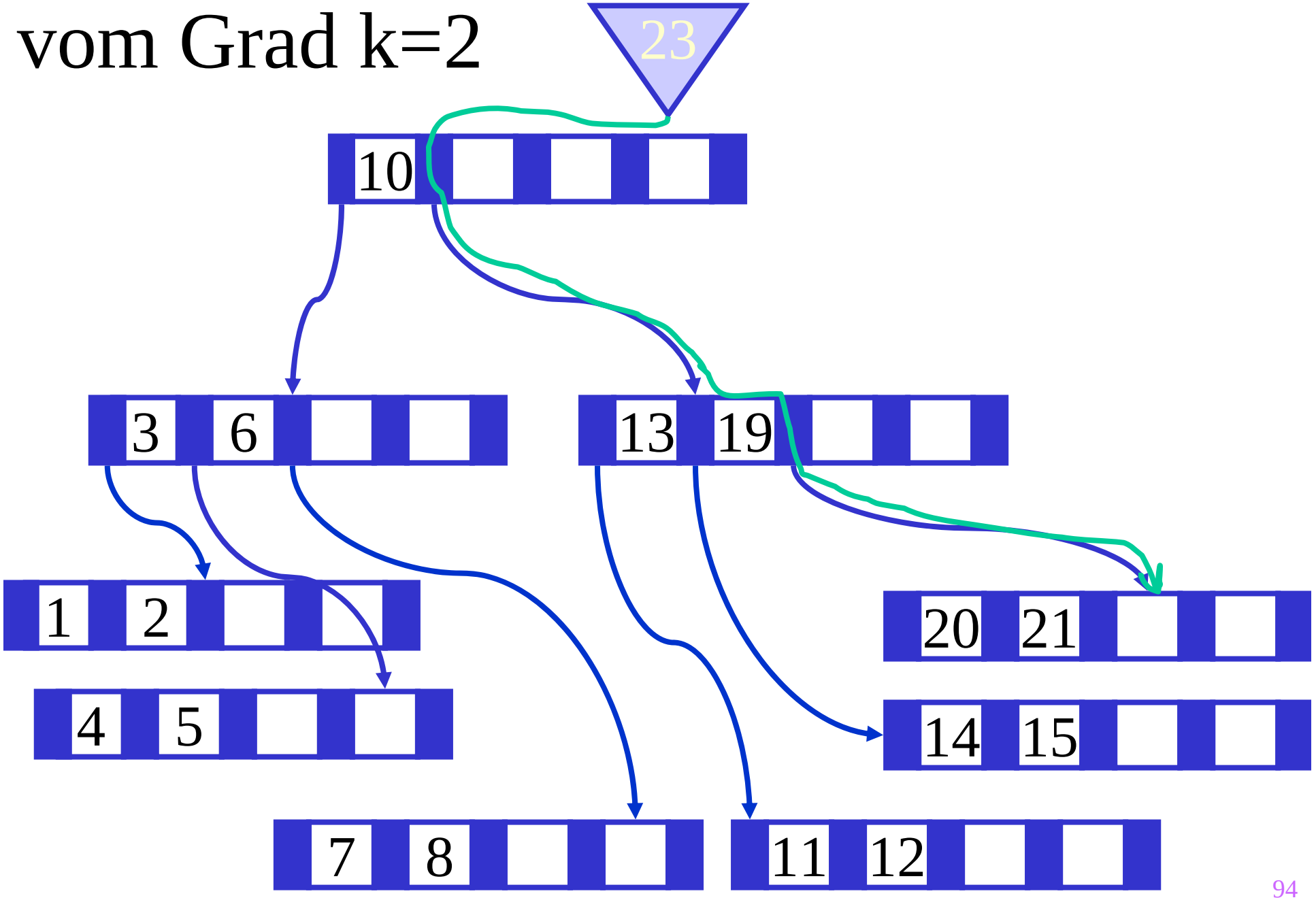
Sukzessiver Aufbau eines B-Baums vom Grad k=2



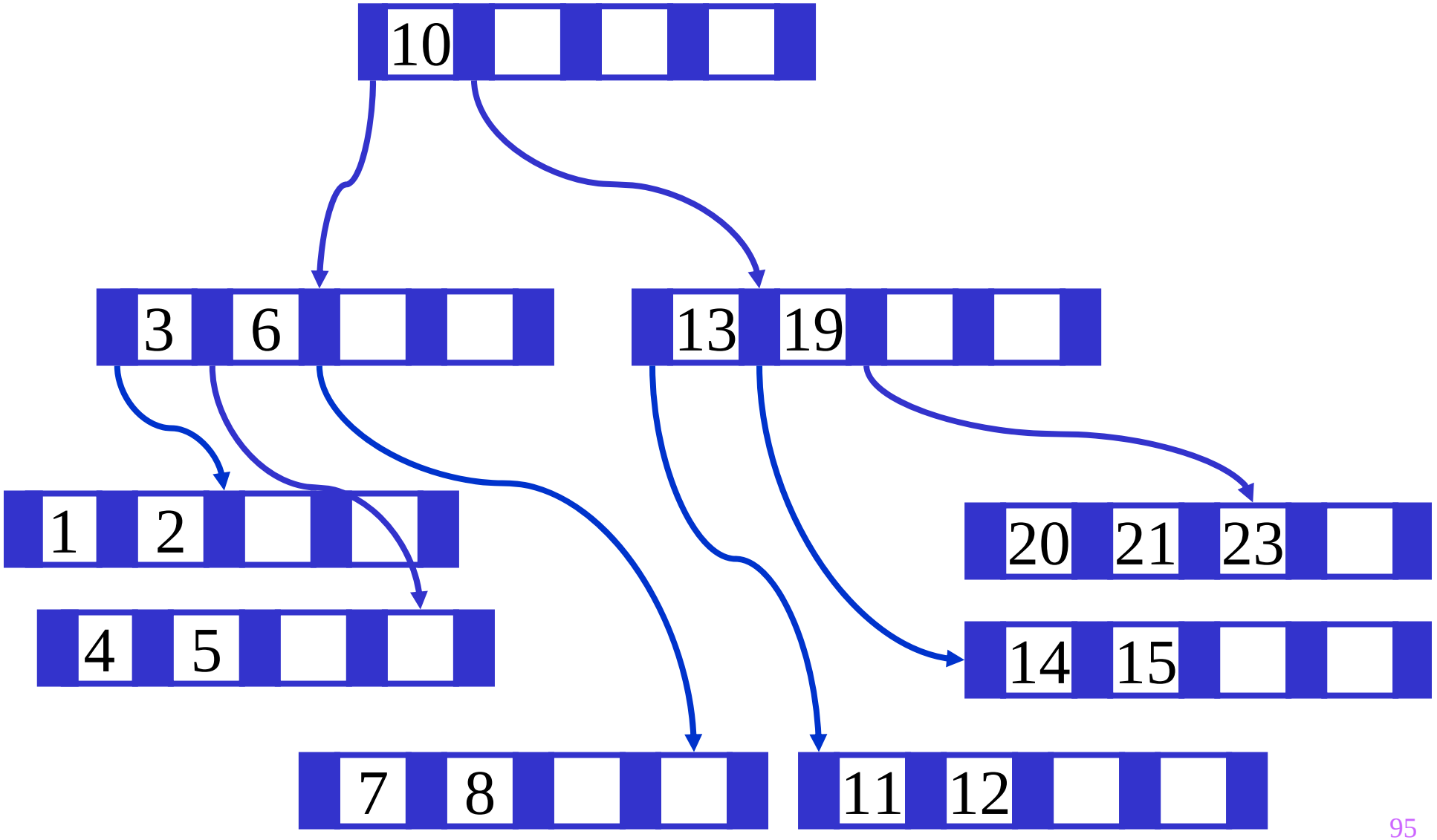
Sukzessiver Aufbau eines B-Baums vom Grad k=2



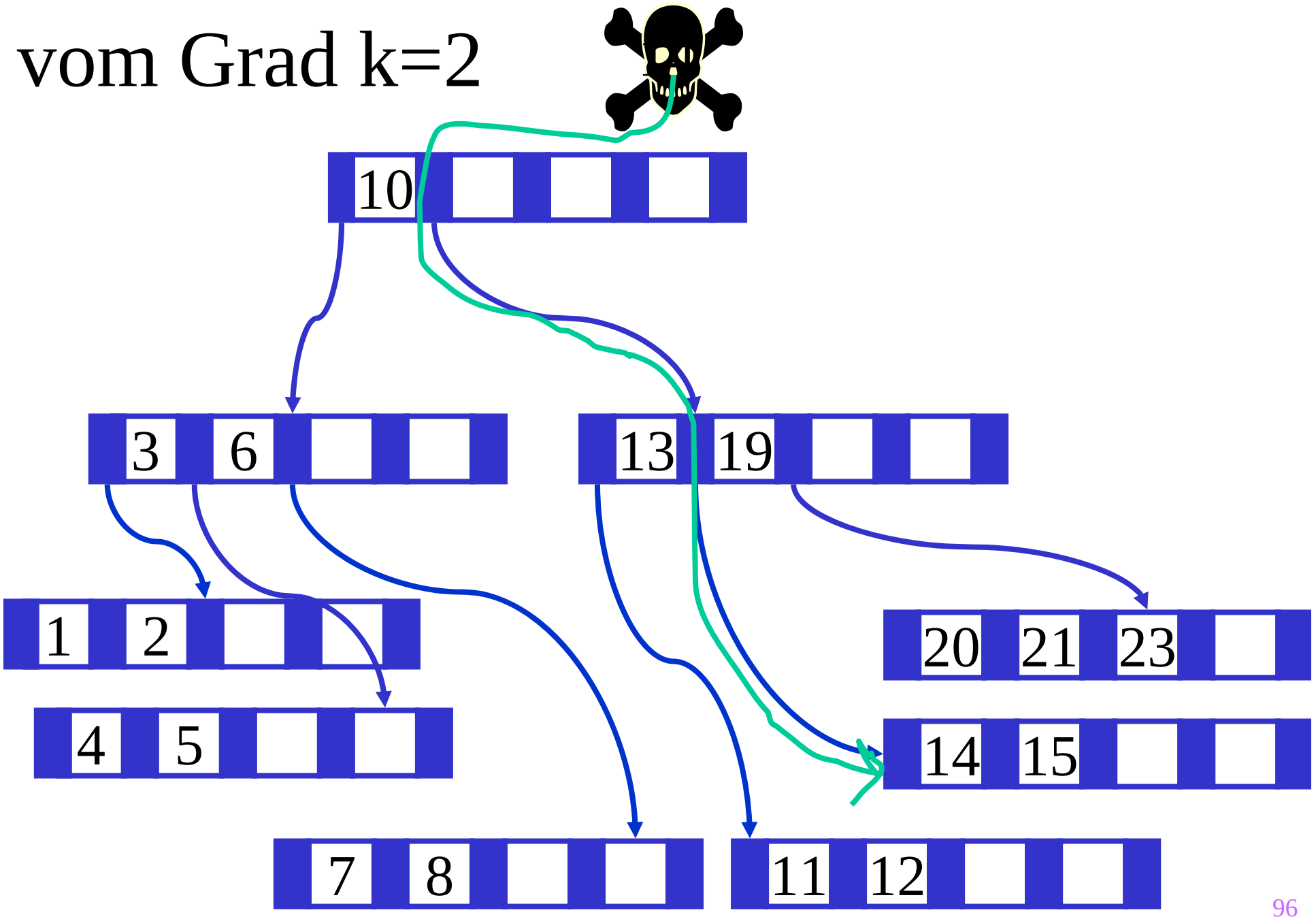
Sukzessiver Aufbau eines B-Baums vom Grad k=2



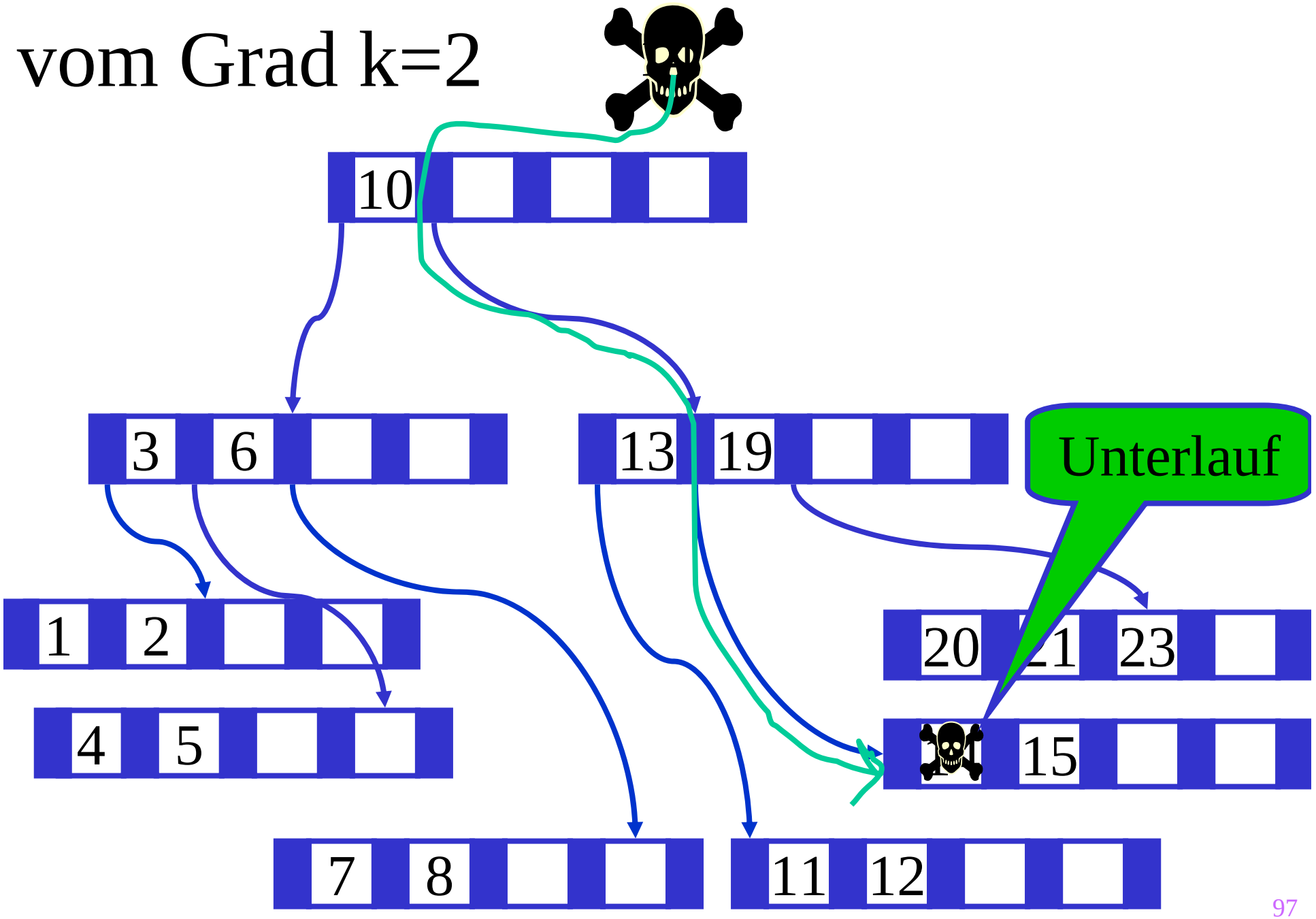
Sukzessiver Aufbau eines B-Baums vom Grad k=2



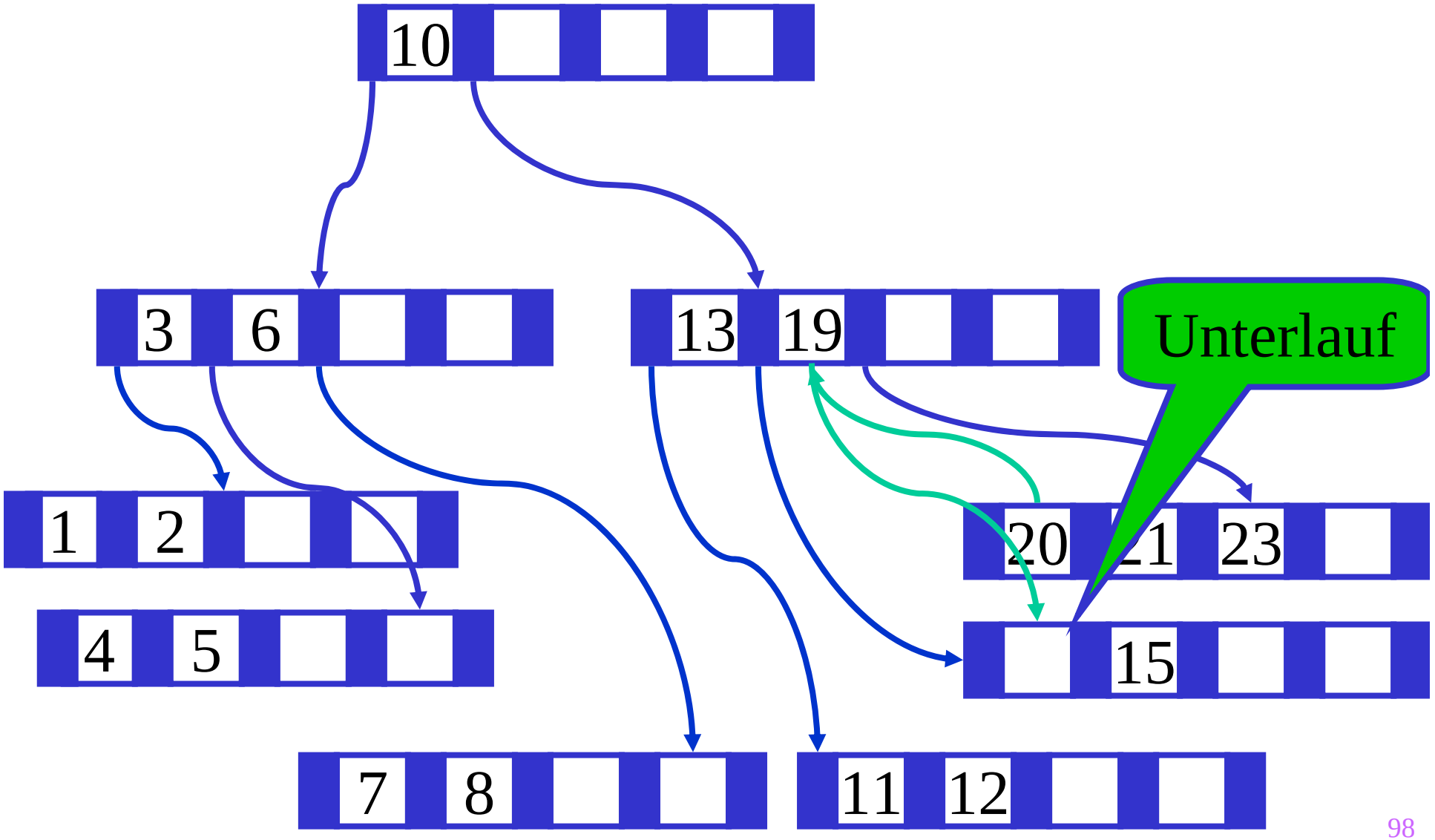
Sukzessiver Aufbau eines B-Baums vom Grad k=2



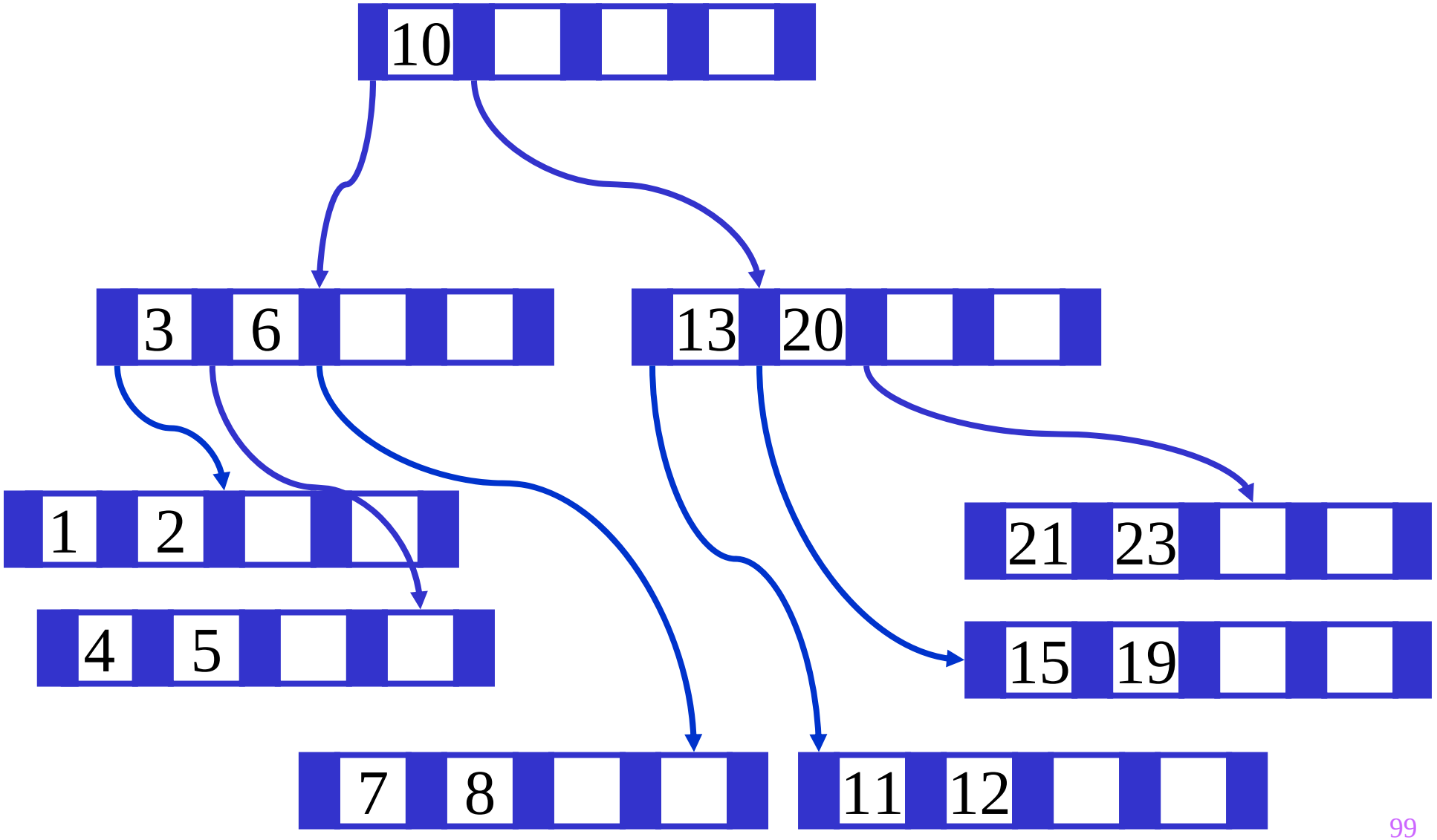
Sukzessiver Aufbau eines B-Baums vom Grad k=2



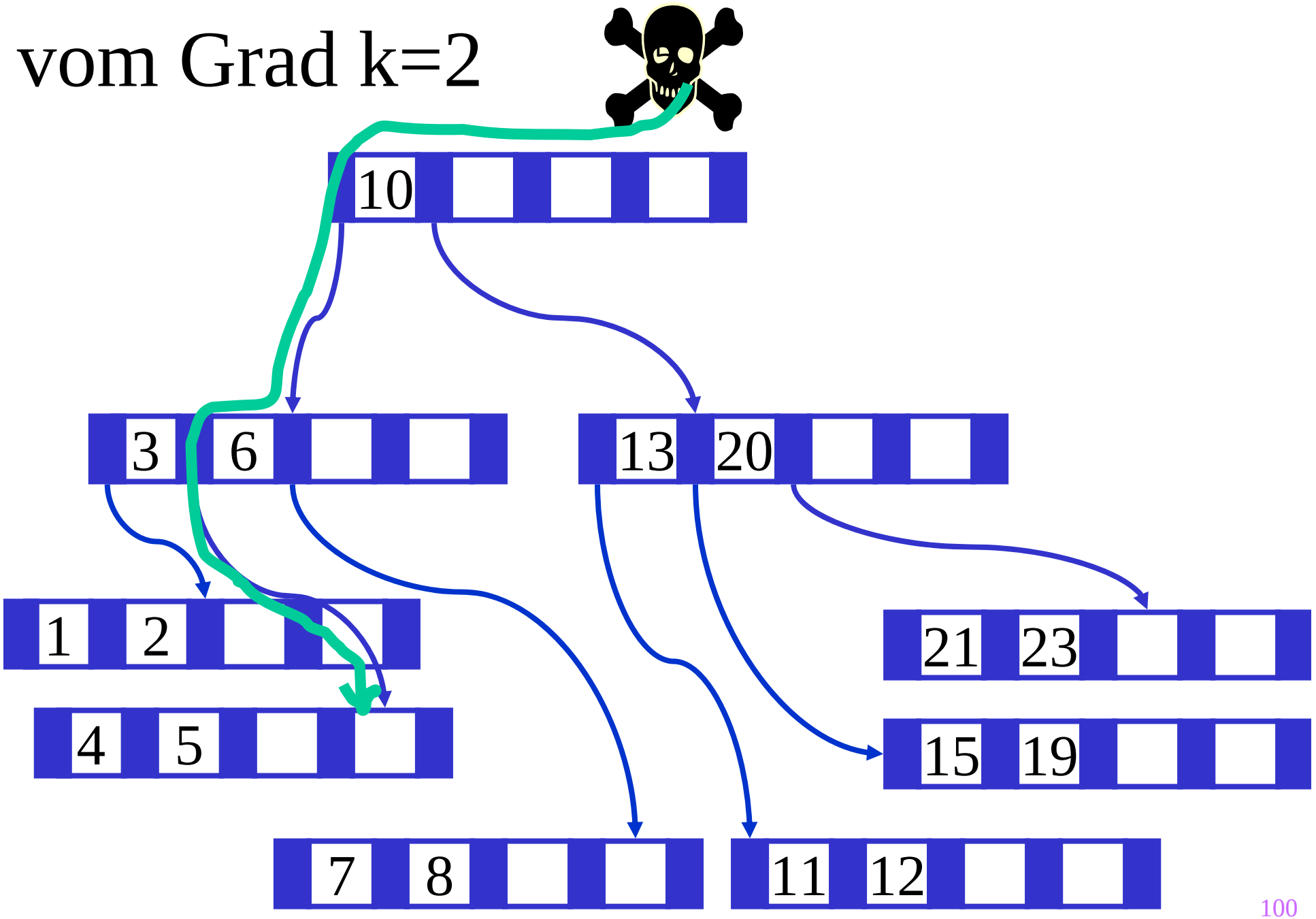
Sukzessiver Aufbau eines B-Baums vom Grad k=2



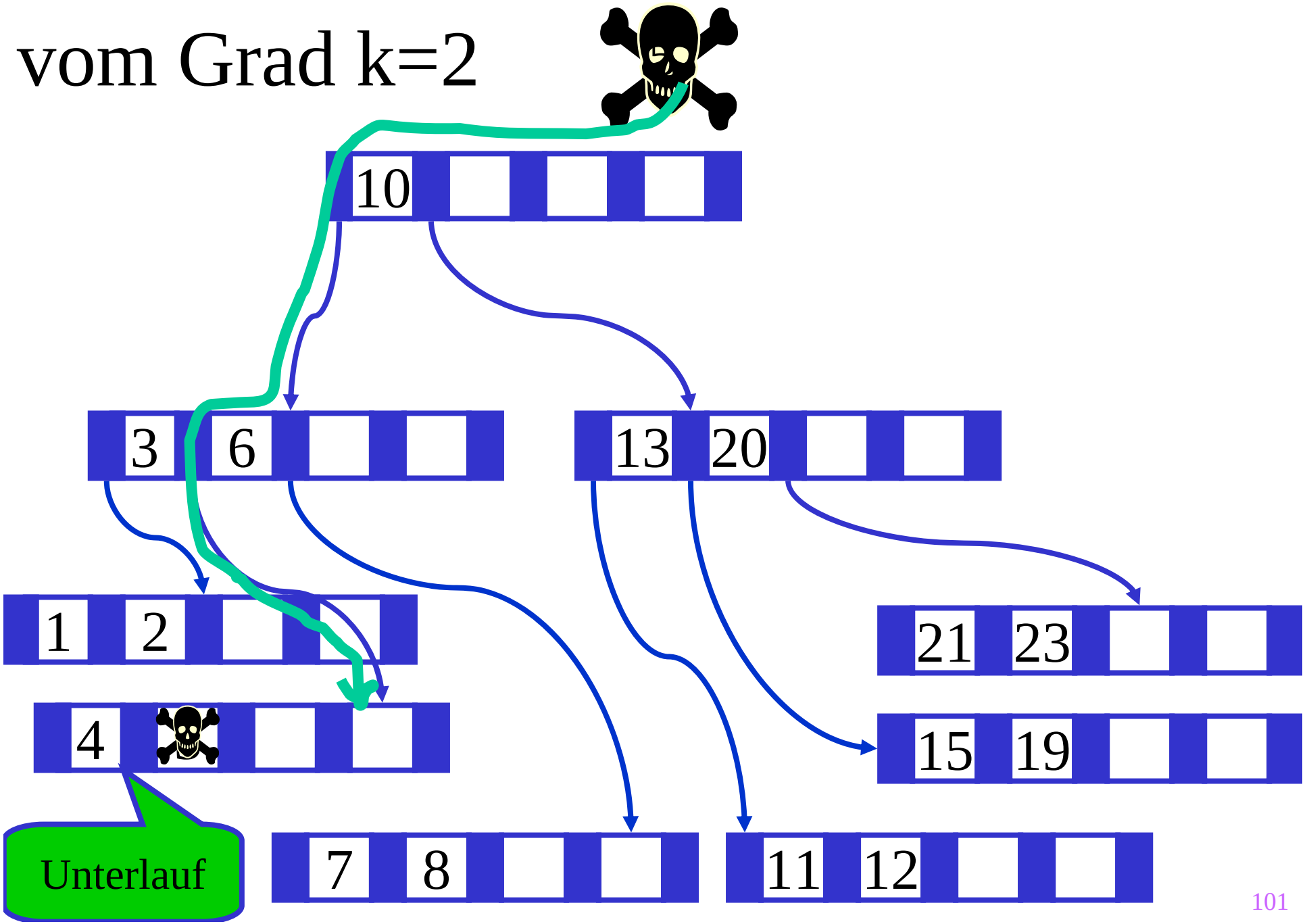
Sukzessiver Aufbau eines B-Baums vom Grad k=2



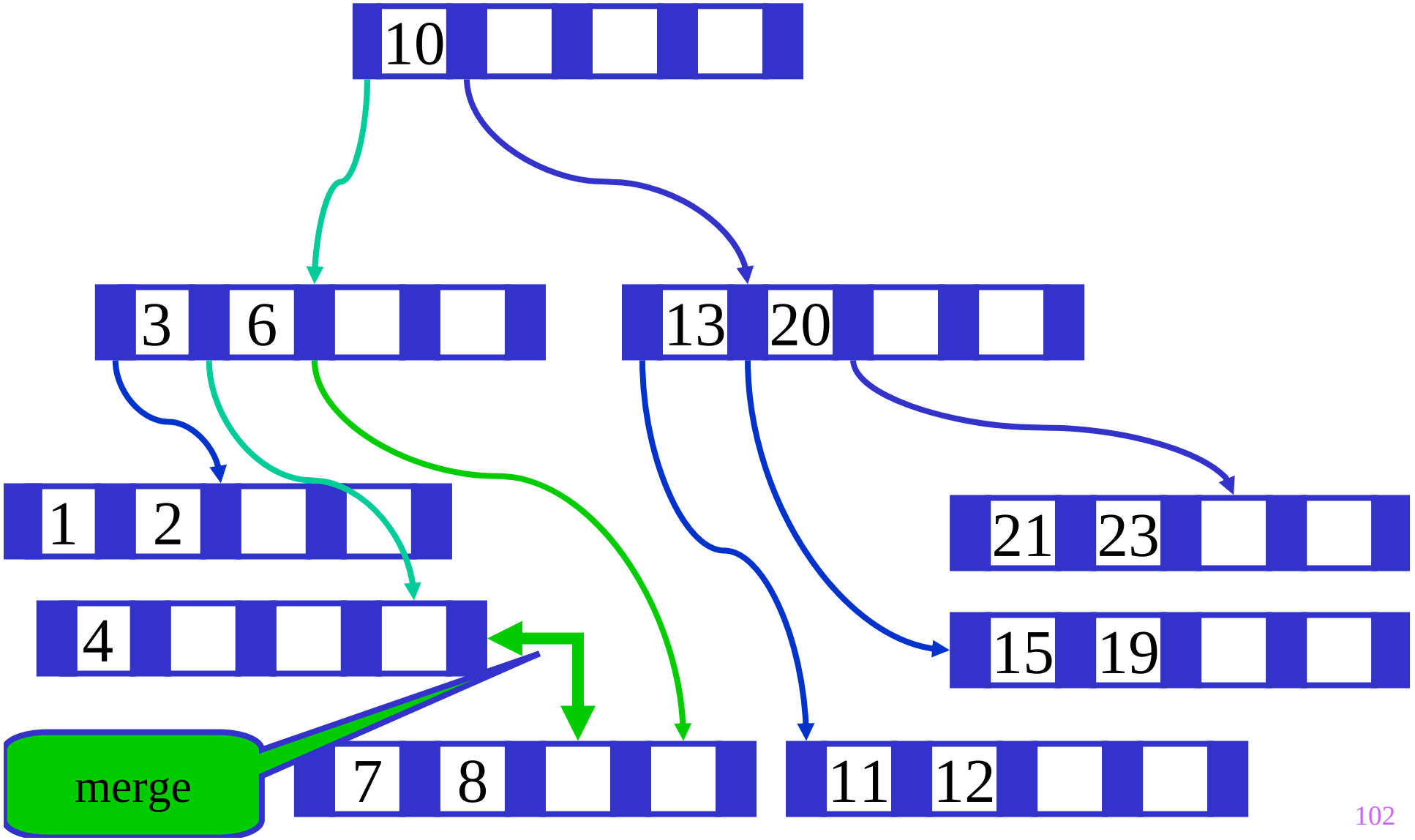
Sukzessiver Aufbau eines B-Baums vom Grad k=2



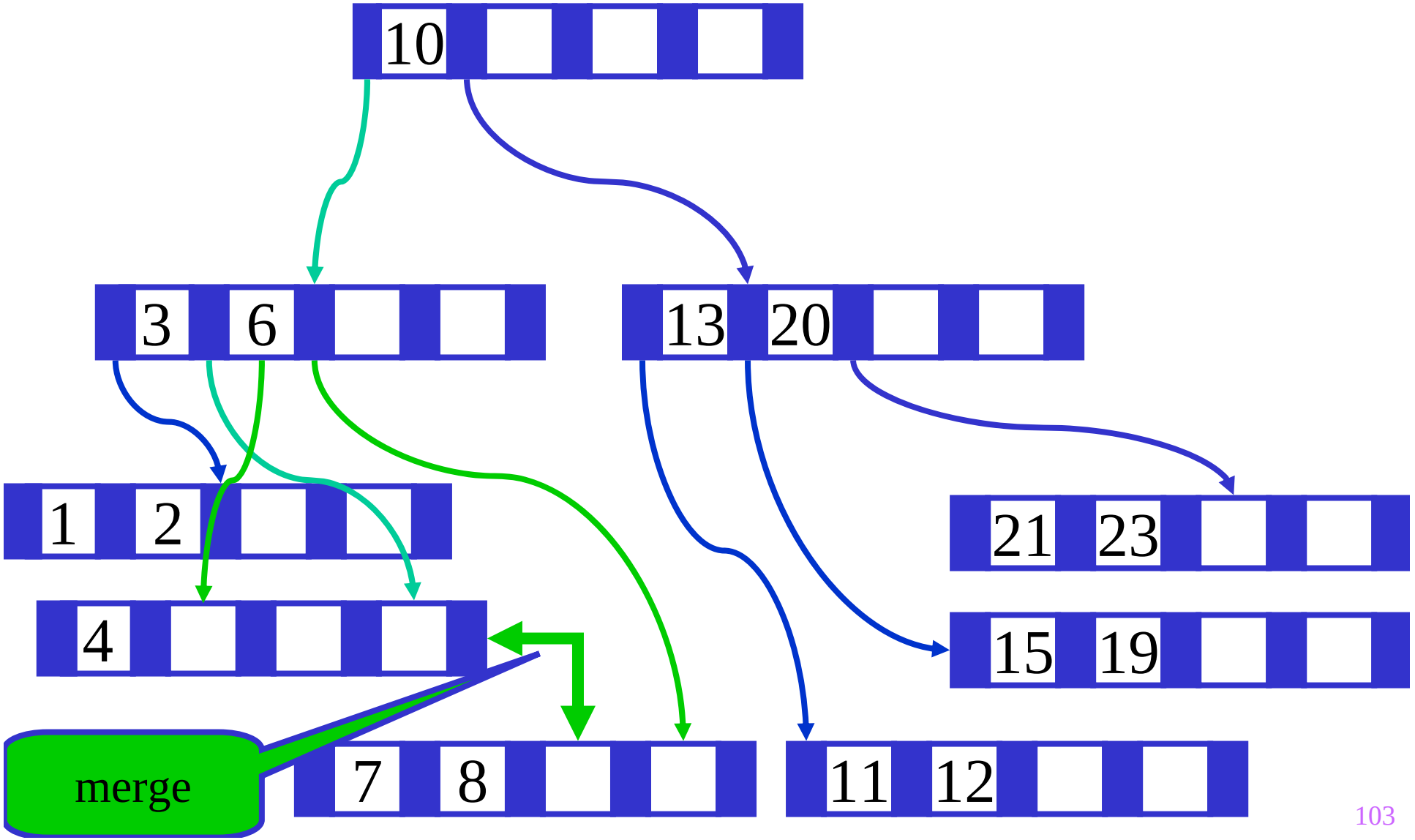
Sukzessiver Aufbau eines B-Baums vom Grad k=2



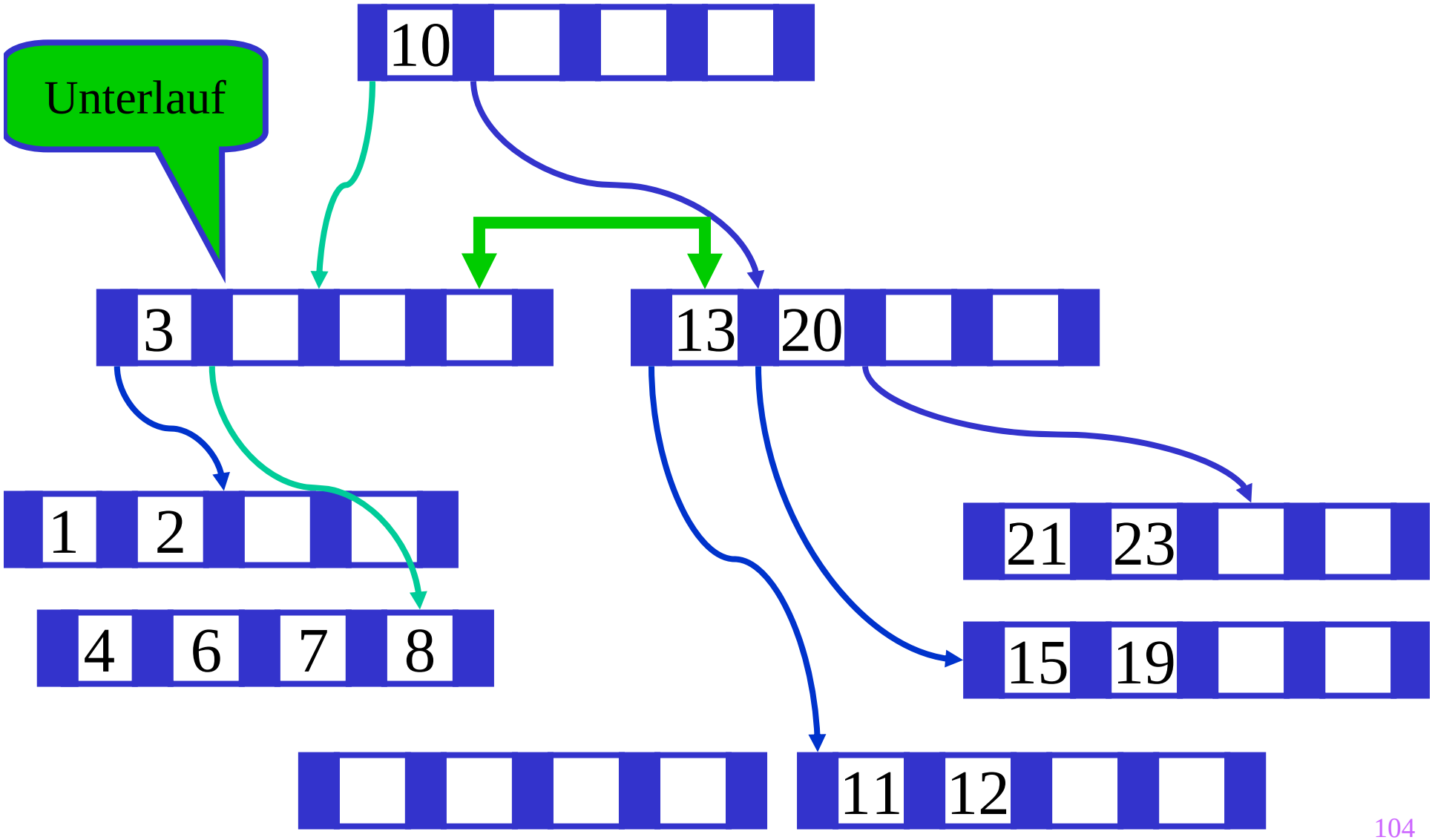
Sukzessiver Aufbau eines B-Baums vom Grad k=2



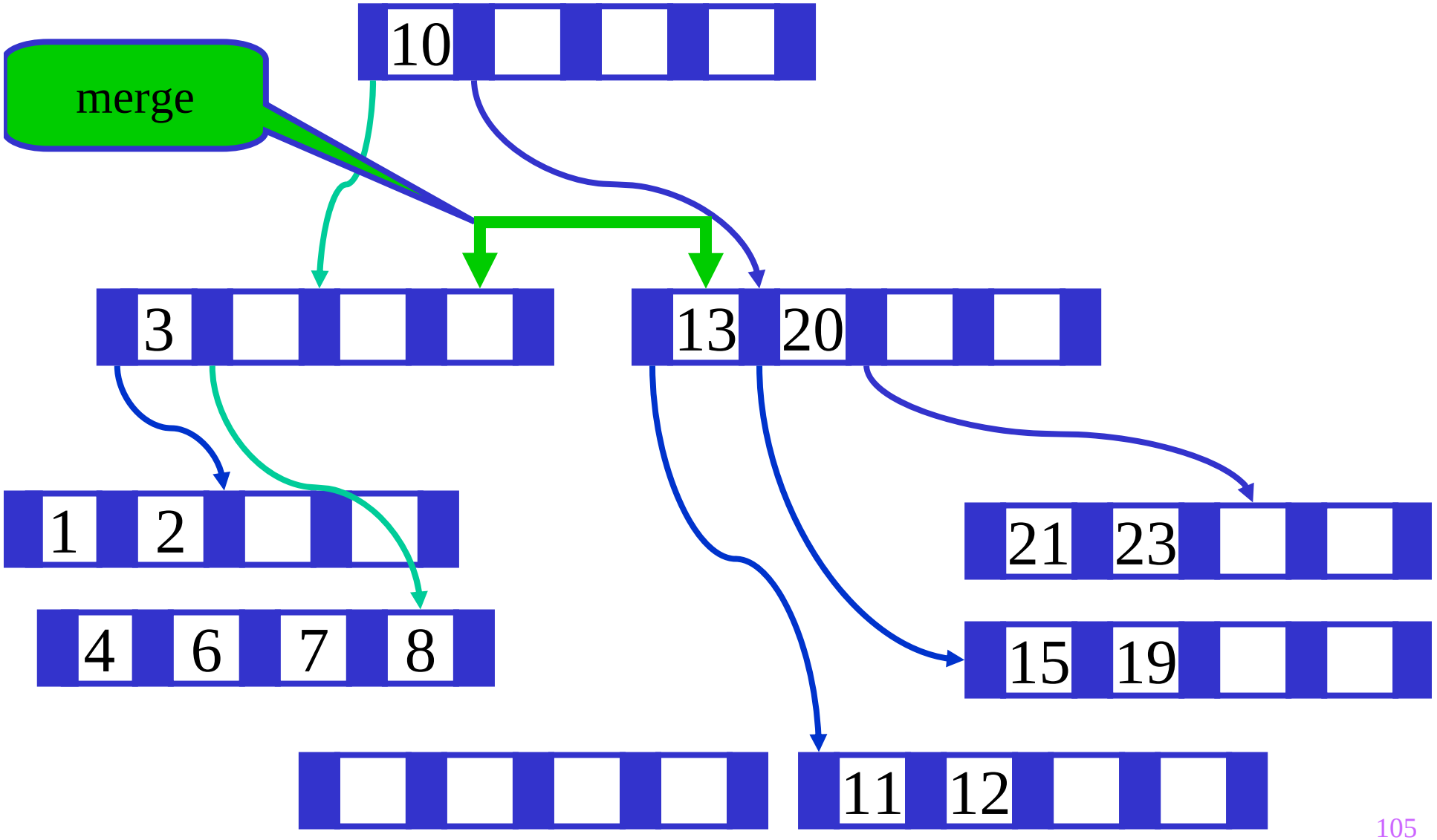
Sukzessiver Aufbau eines B-Baums vom Grad k=2



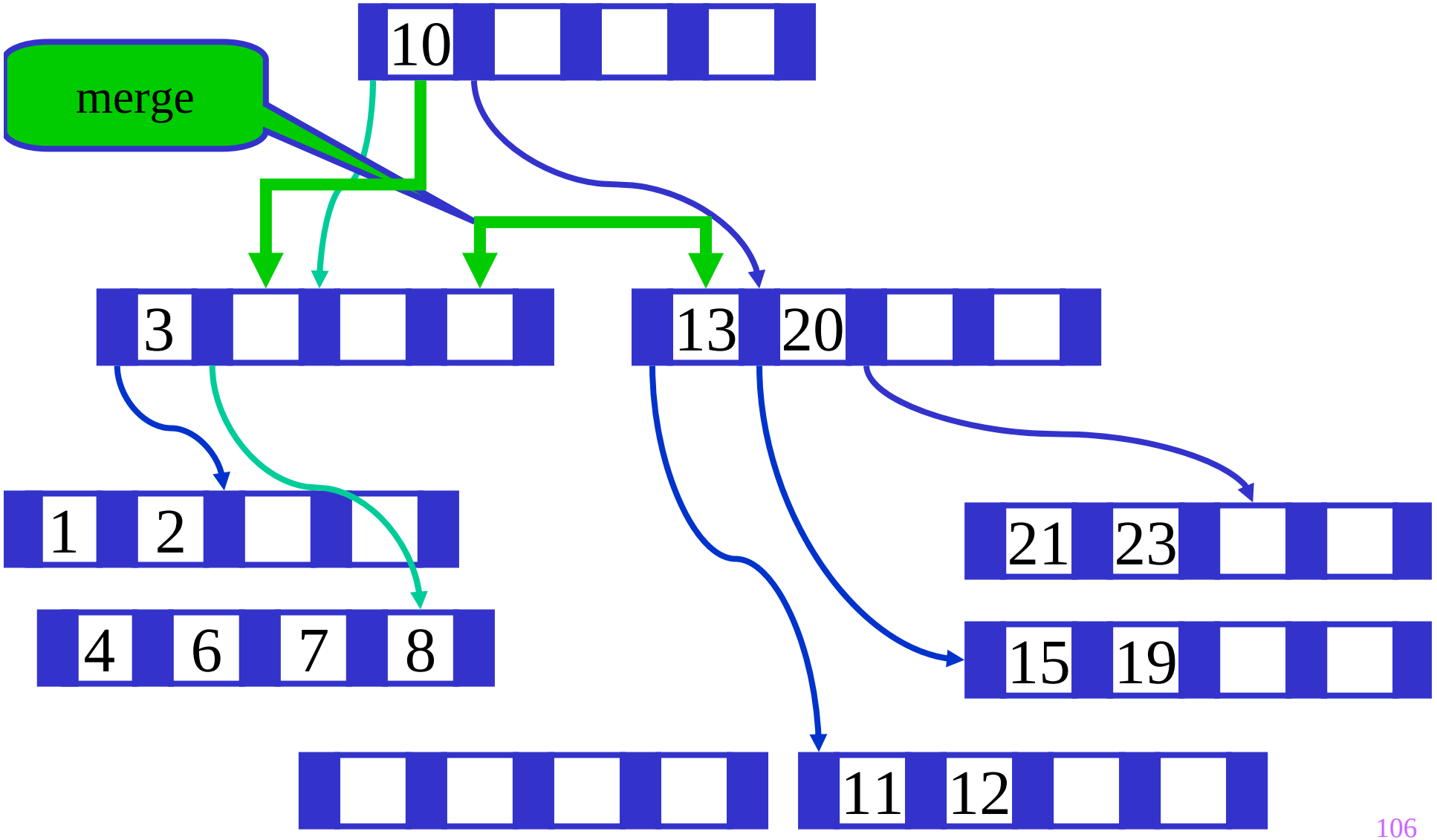
Sukzessiver Aufbau eines B-Baums vom Grad k=2



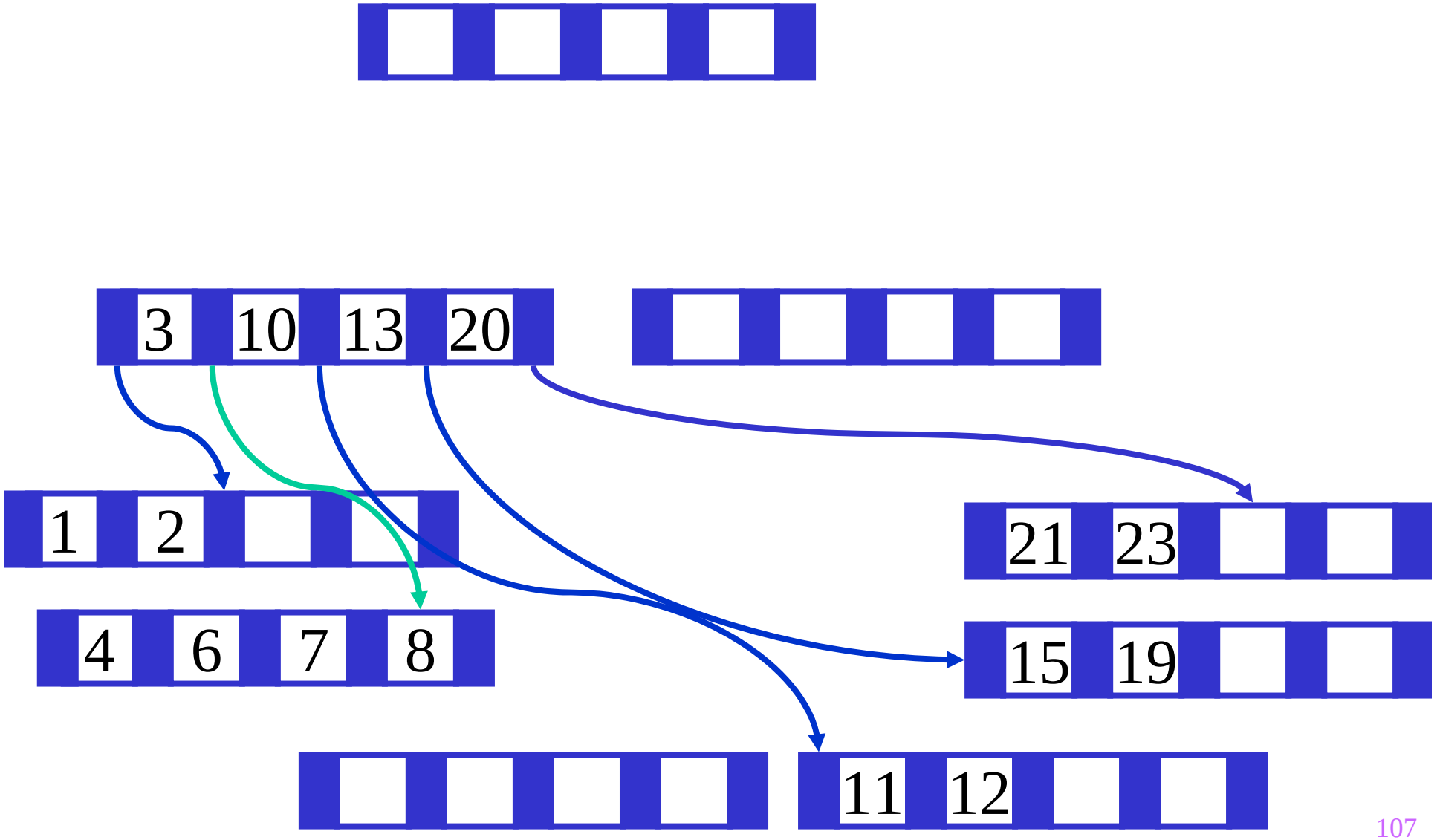
Sukzessiver Aufbau eines B-Baums vom Grad k=2



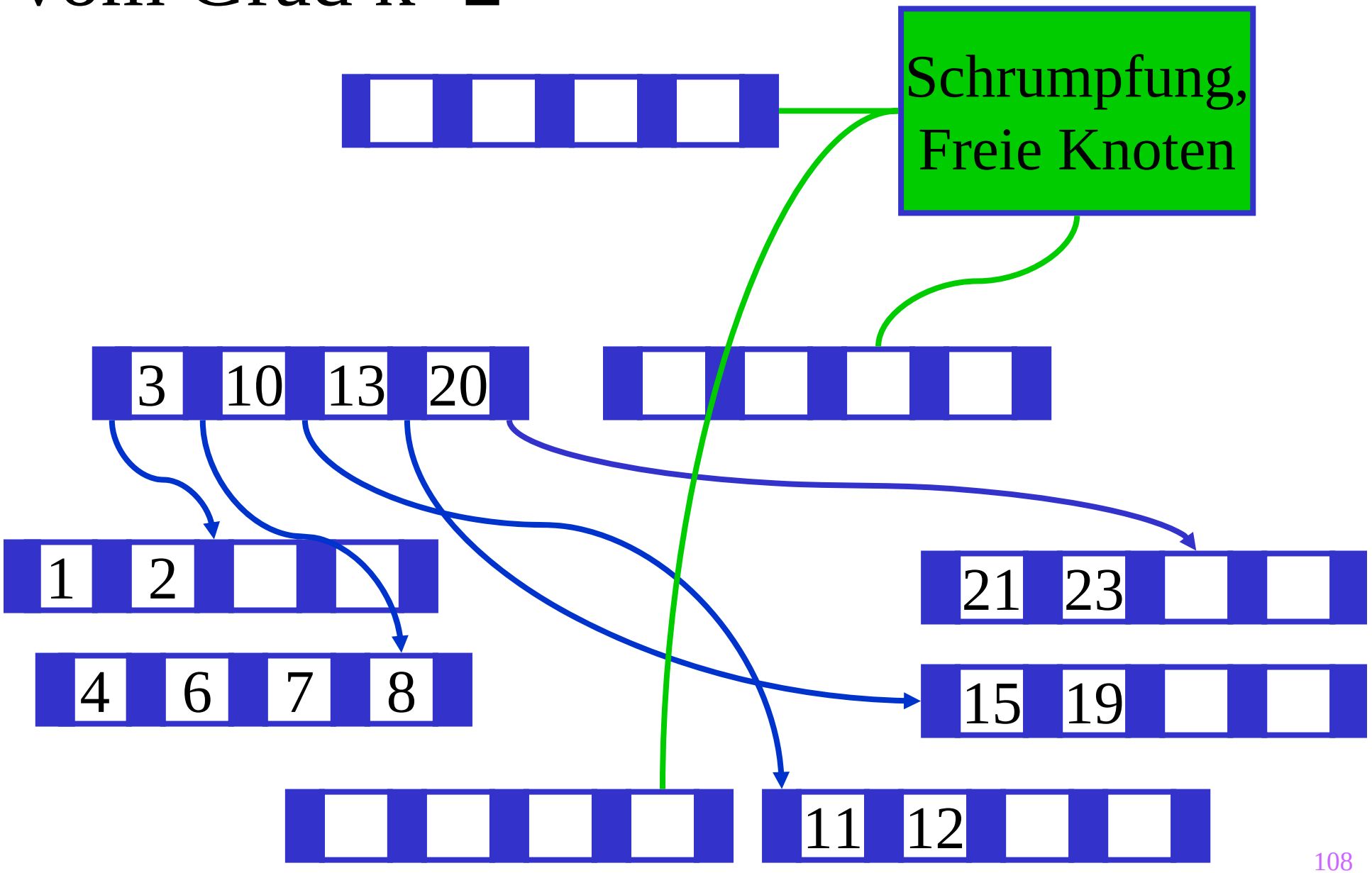
Sukzessiver Aufbau eines B-Baums vom Grad k=2



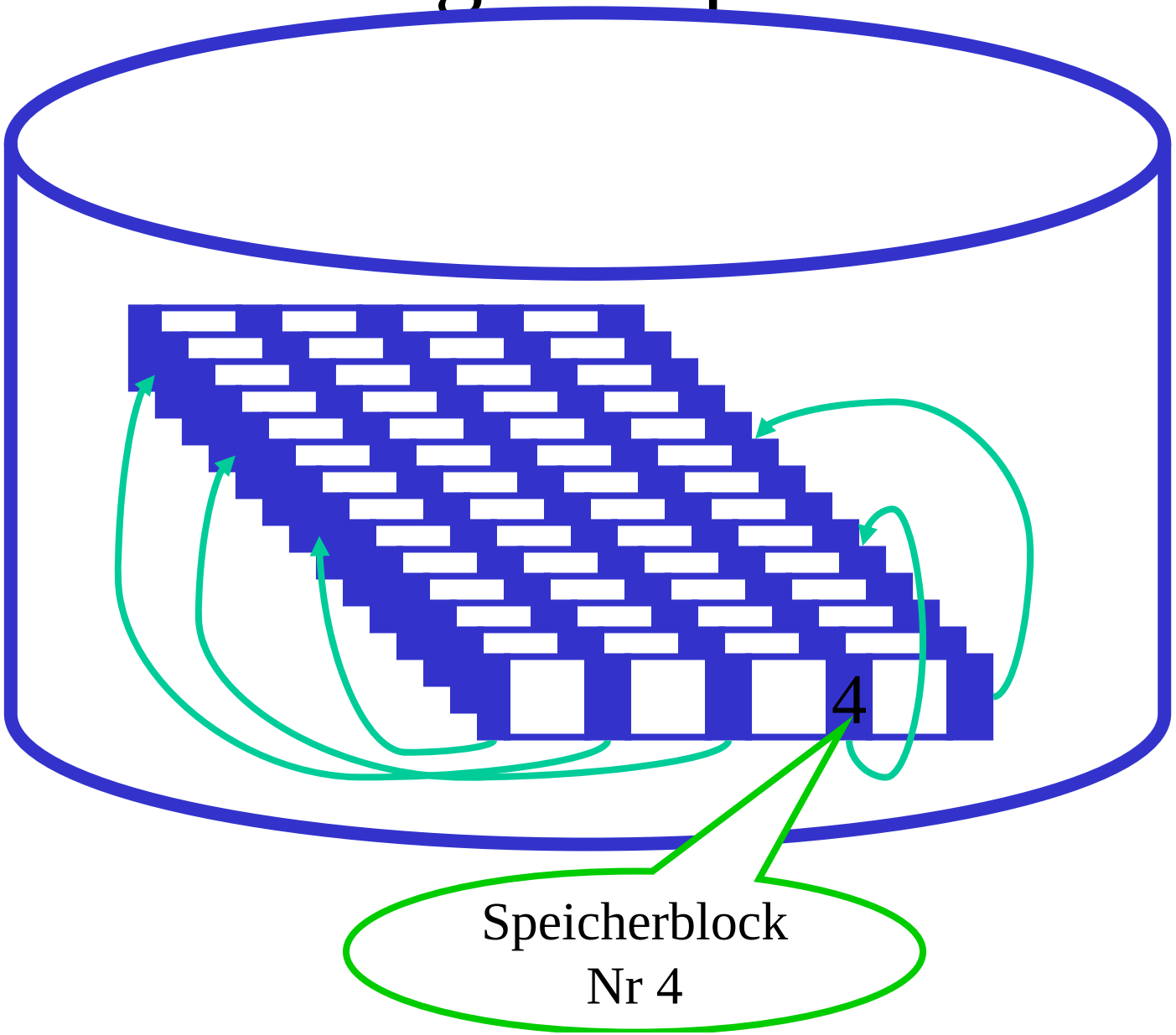
Sukzessiver Aufbau eines B-Baums vom Grad k=2



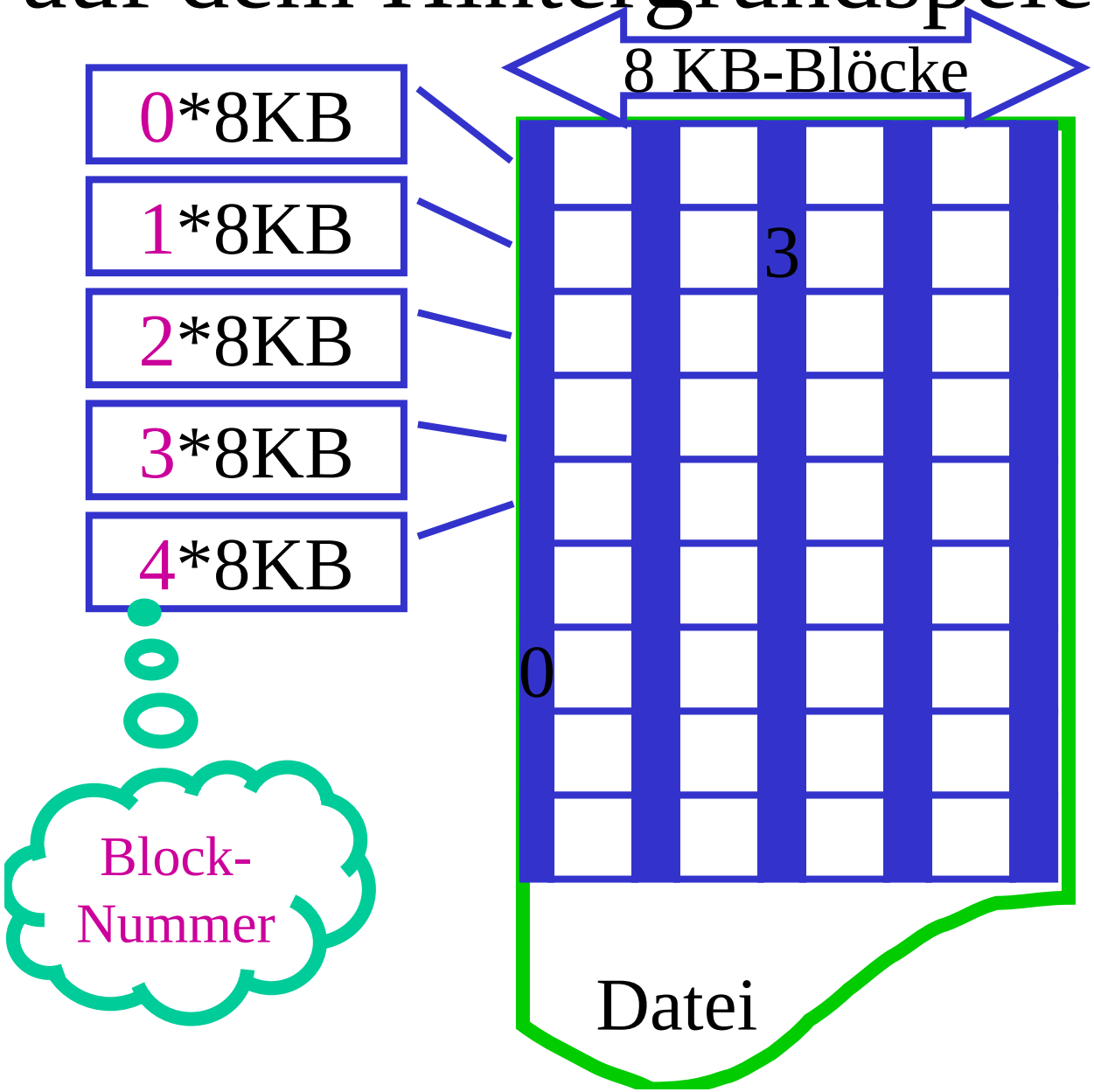
Sukzessiver Aufbau eines B-Baums vom Grad k=2



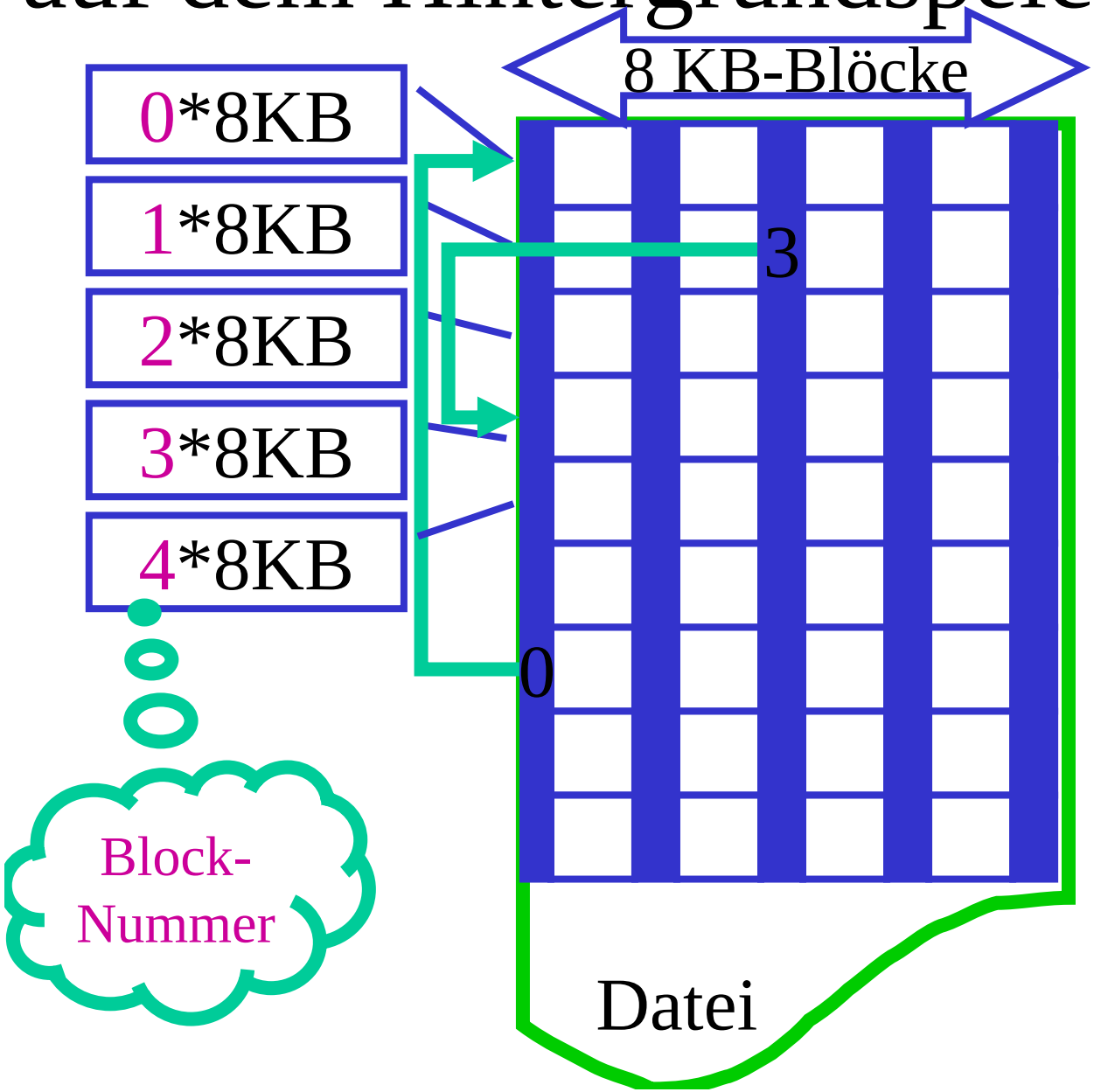
Speicherstruktur eines B-Baums auf dem Hintergrundspeicher



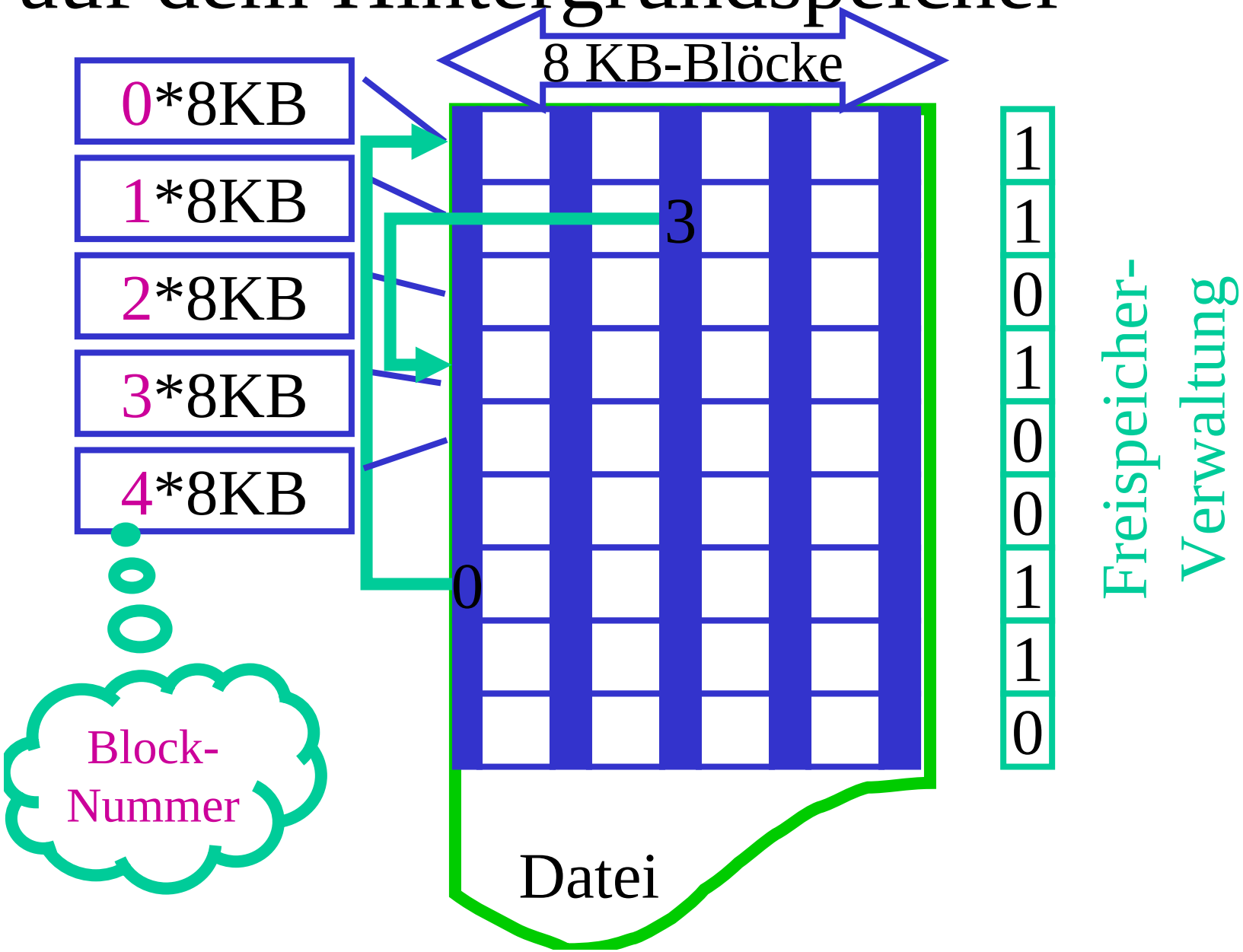
Speicherstruktur eines B-Baums auf dem Hintergrundspeicher



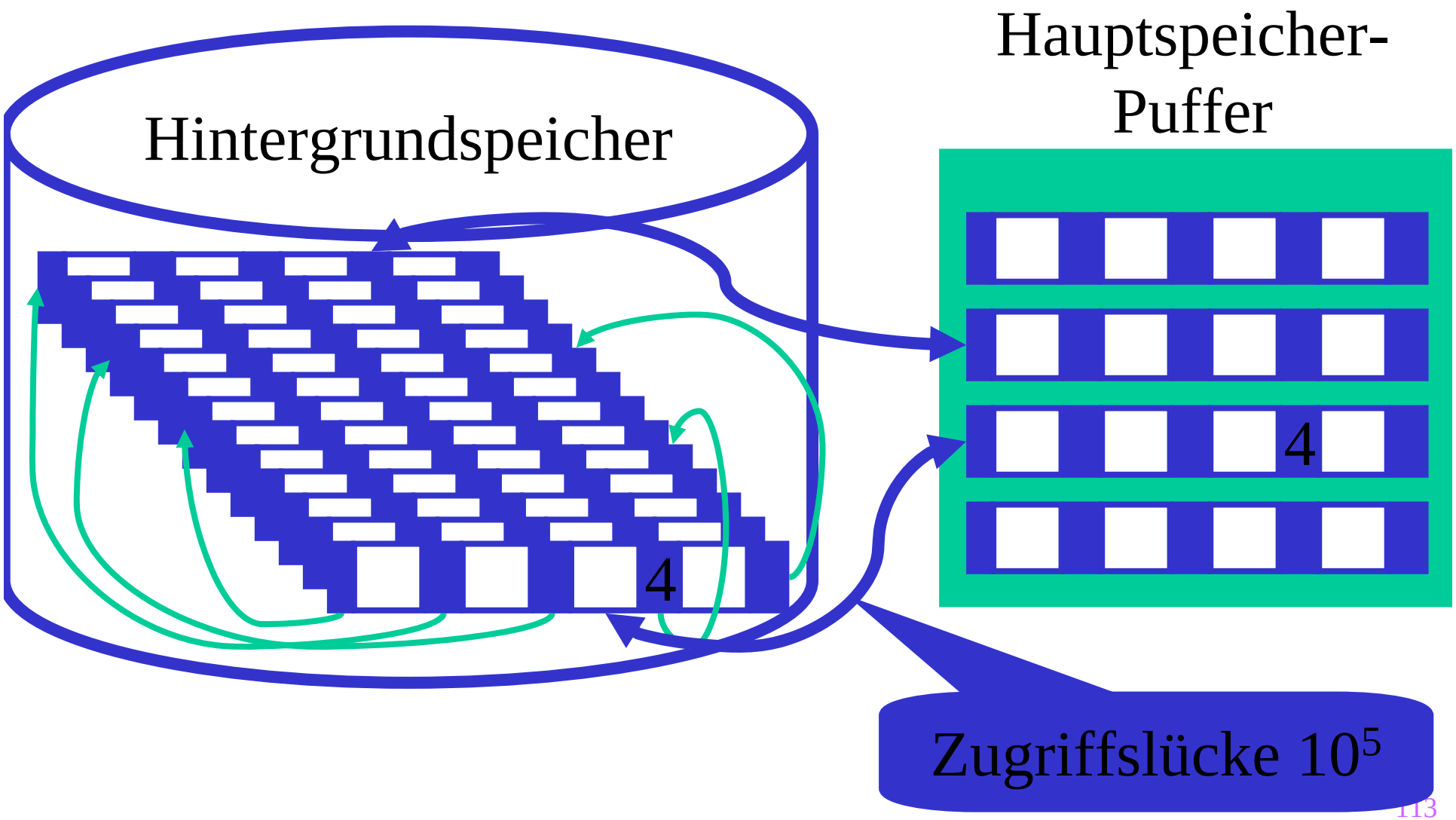
Speicherstruktur eines B-Baums auf dem Hintergrundspeicher



Speicherstruktur eines B-Baums auf dem Hintergrundspeicher



Zusammenspiel: Hintergrundspeicher -- Hauptspeicher

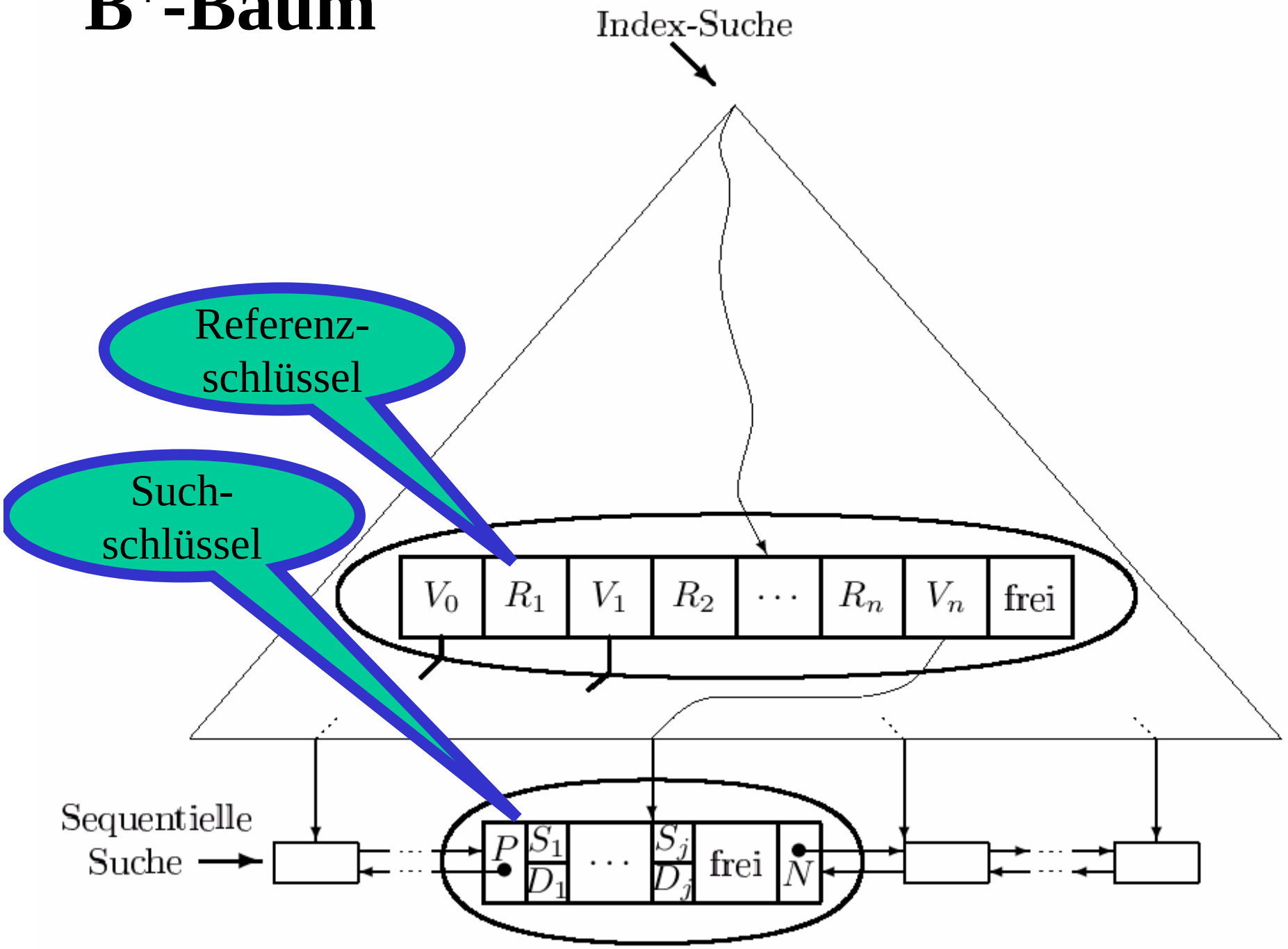


B^+-Bäume

- Je größer der Verzweigungsgrad des Baumes ist, desto flacher ist er.
- Dadurch verringert sich die Suchtiefe.
- Bei B^+-Bäumen werden daher (im Gegensatz zu B-Bäumen) die Inhalte in die Blätter verlagert.

- In der Literatur heissen die B^+-Bäume auch B^*-Bäume.

B$^+$-Baum

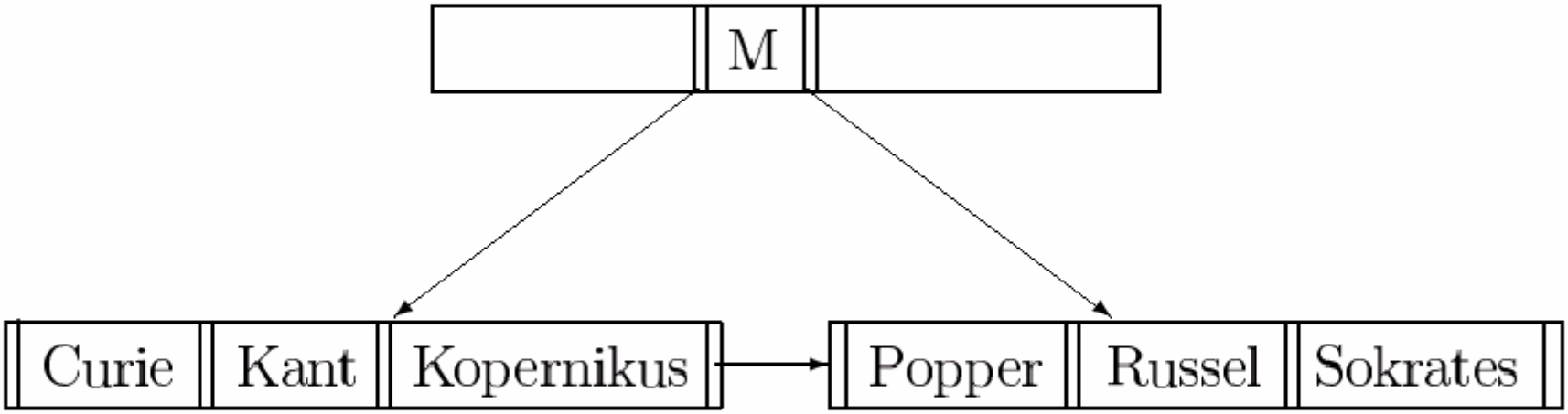

Index-Suche
Referenz-schlüssel
Such-schlüssel
V_0 R_1 V_1 R_2 $\cdots$ R_n V_n frei
Sequentielle Suche
P S_1 D_1 $\cdots$ S_j D_j frei N

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.

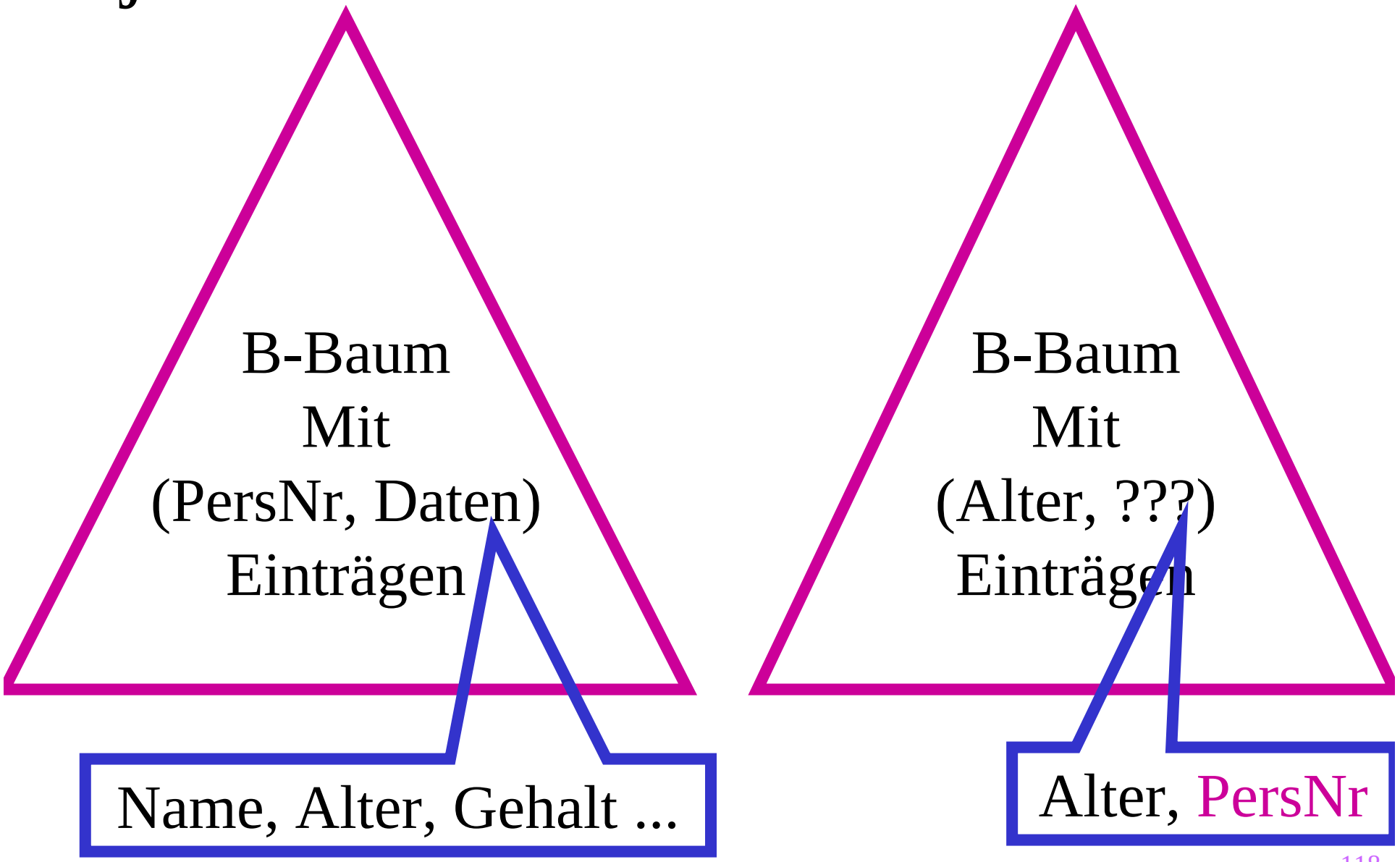
Präfix-B$^+$-Bäume

- Es werden nur *Referenz*schlüssel benötigt.

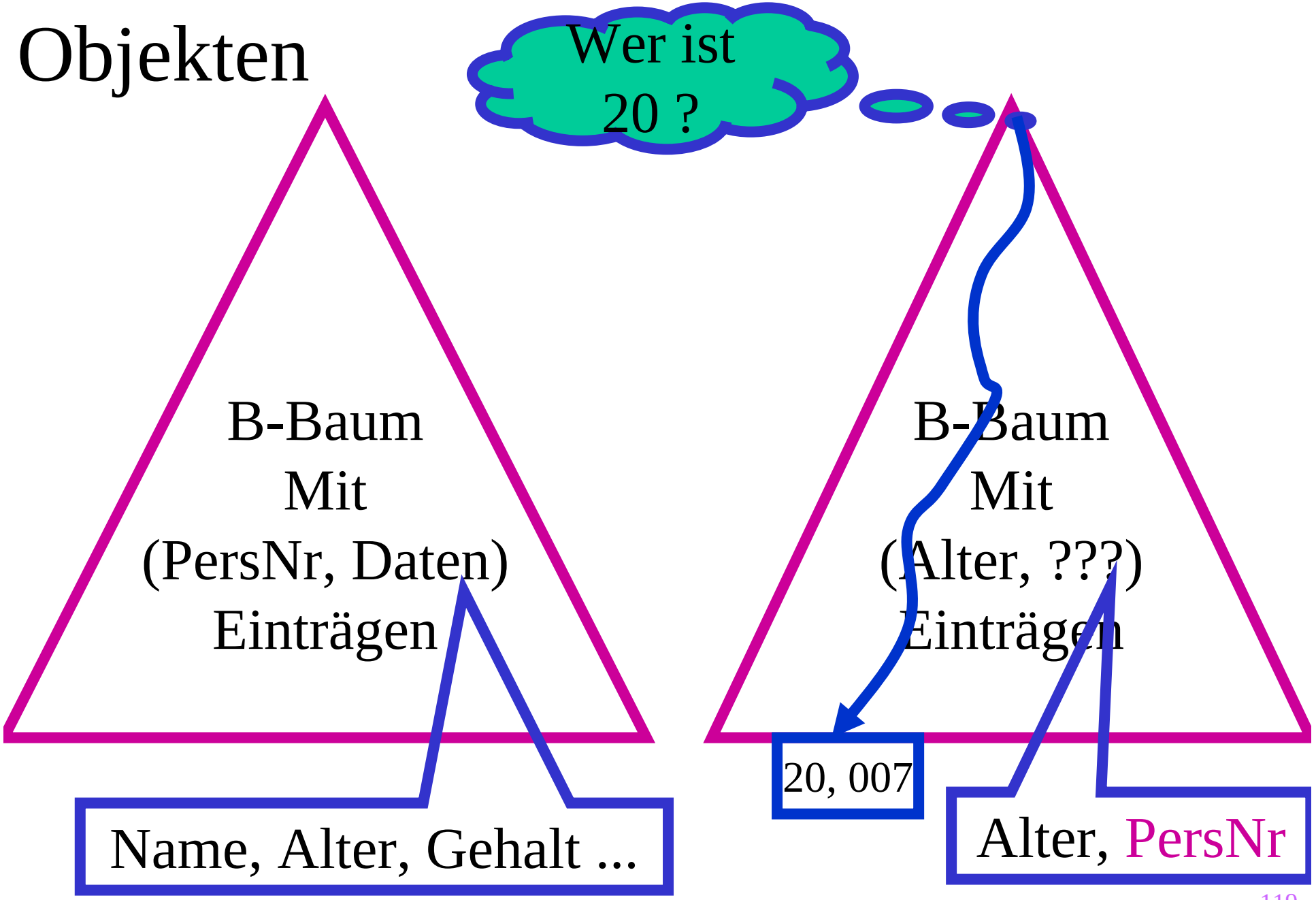


- beliebiger Referenzschlüssel R mit
  Kopernikus $\leq R <$ Popper

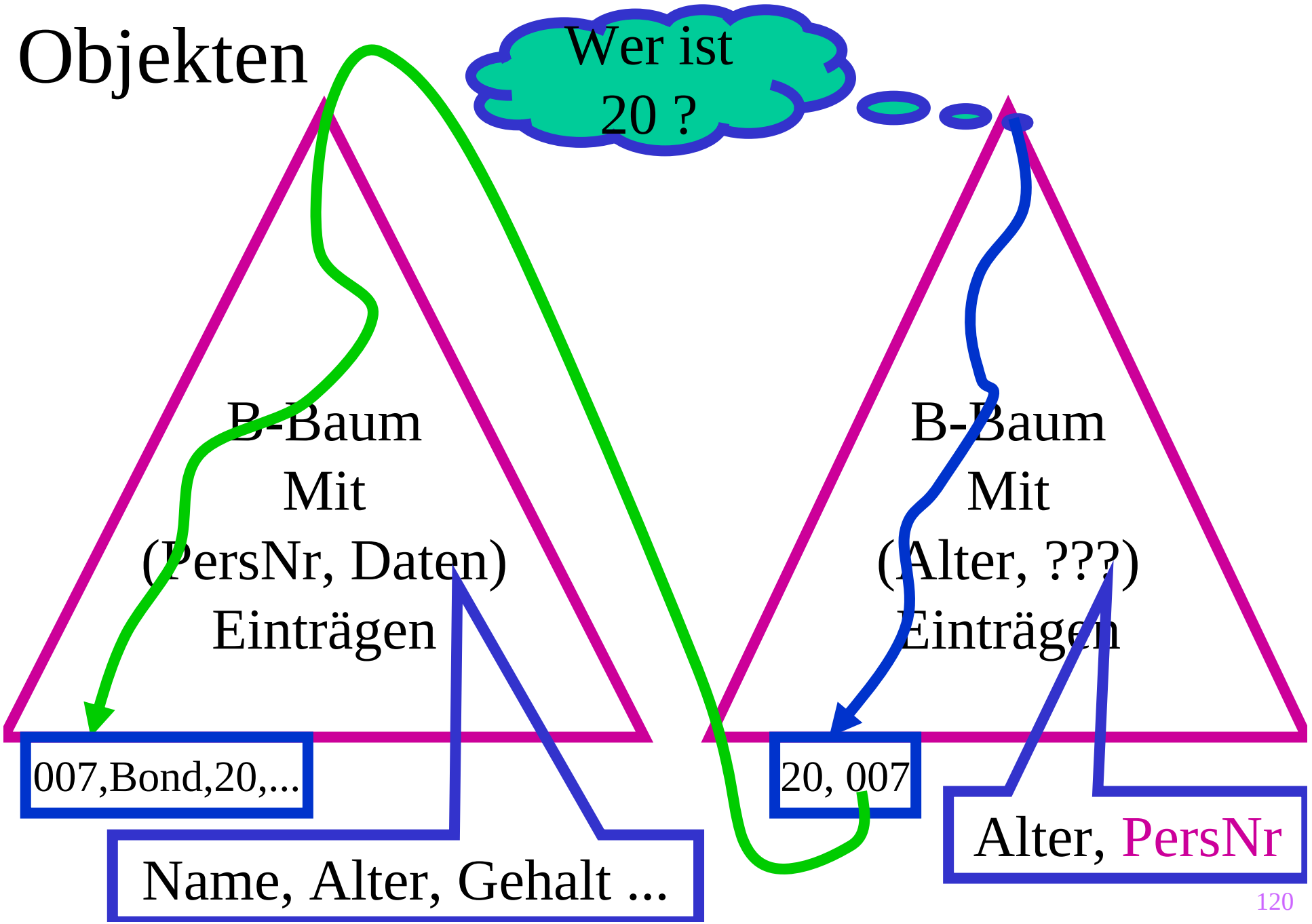
Mehrere Indexe auf denselben Objekten



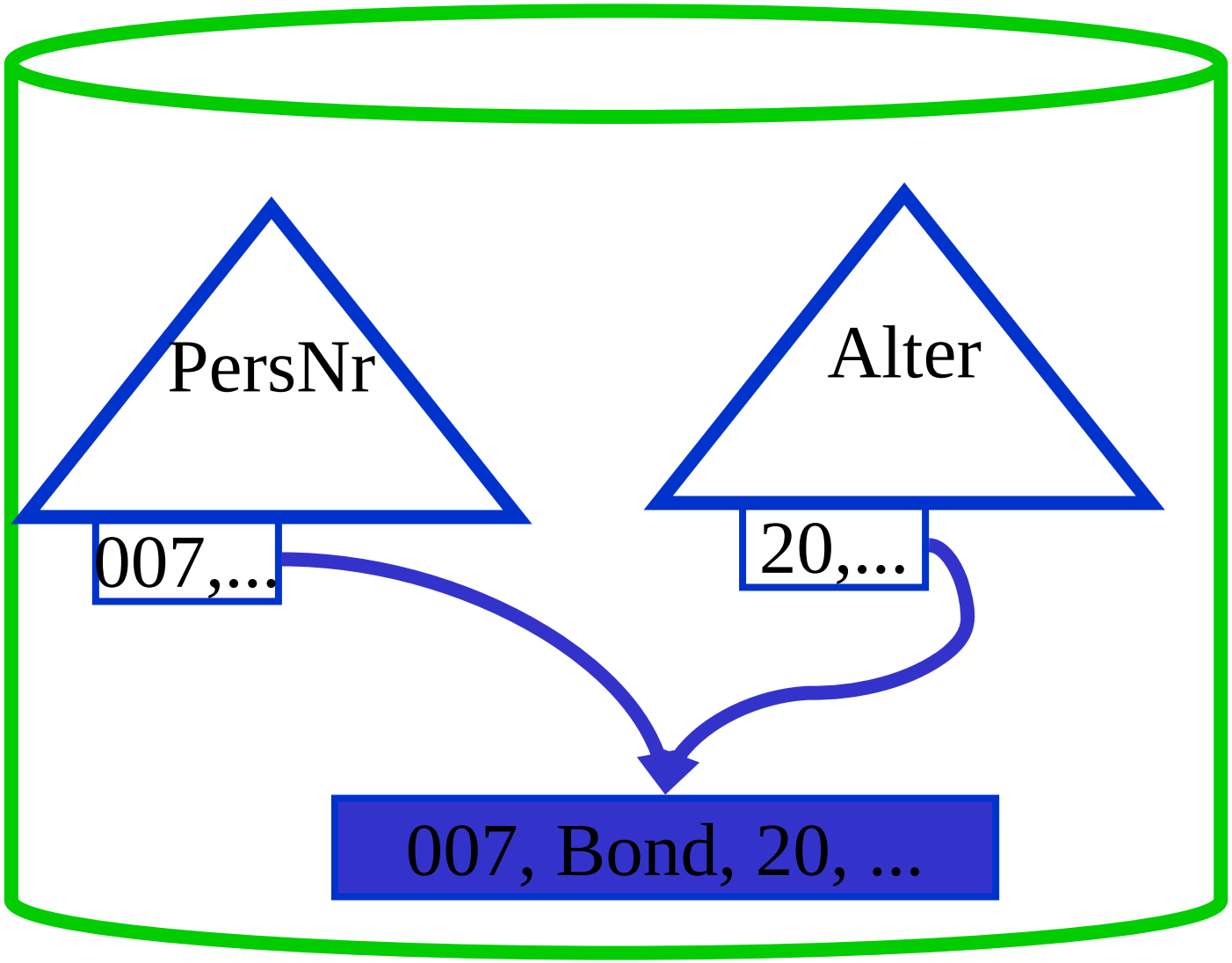
Mehrere Indexe auf denselben Objekten



Mehrere Indexe auf denselben Objekten



Eine andere Möglichkeit: Referenzierung über Speicheradressen



Hashing

- B-Bäume haben den Vorteil, dass sie Bereichsanfragen und Sortierungen unterstützen.
- Aufwand: O(log n)

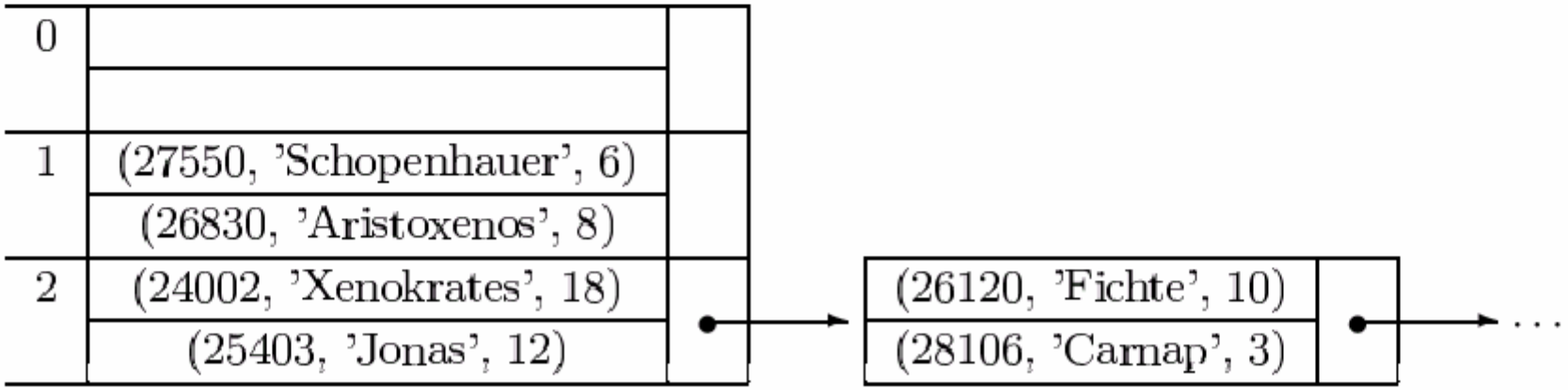
- Hashing ist vom Aufwand O(1) für das Suchen einzelner Tupel, unterstützt aber keine Sortierung.

Statisches Hashing

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

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

- Kollisionsbehandlung durch Überlaufseiten



$\Longrightarrow$ ineffizient bei nicht vorhersehbarer Datenmenge

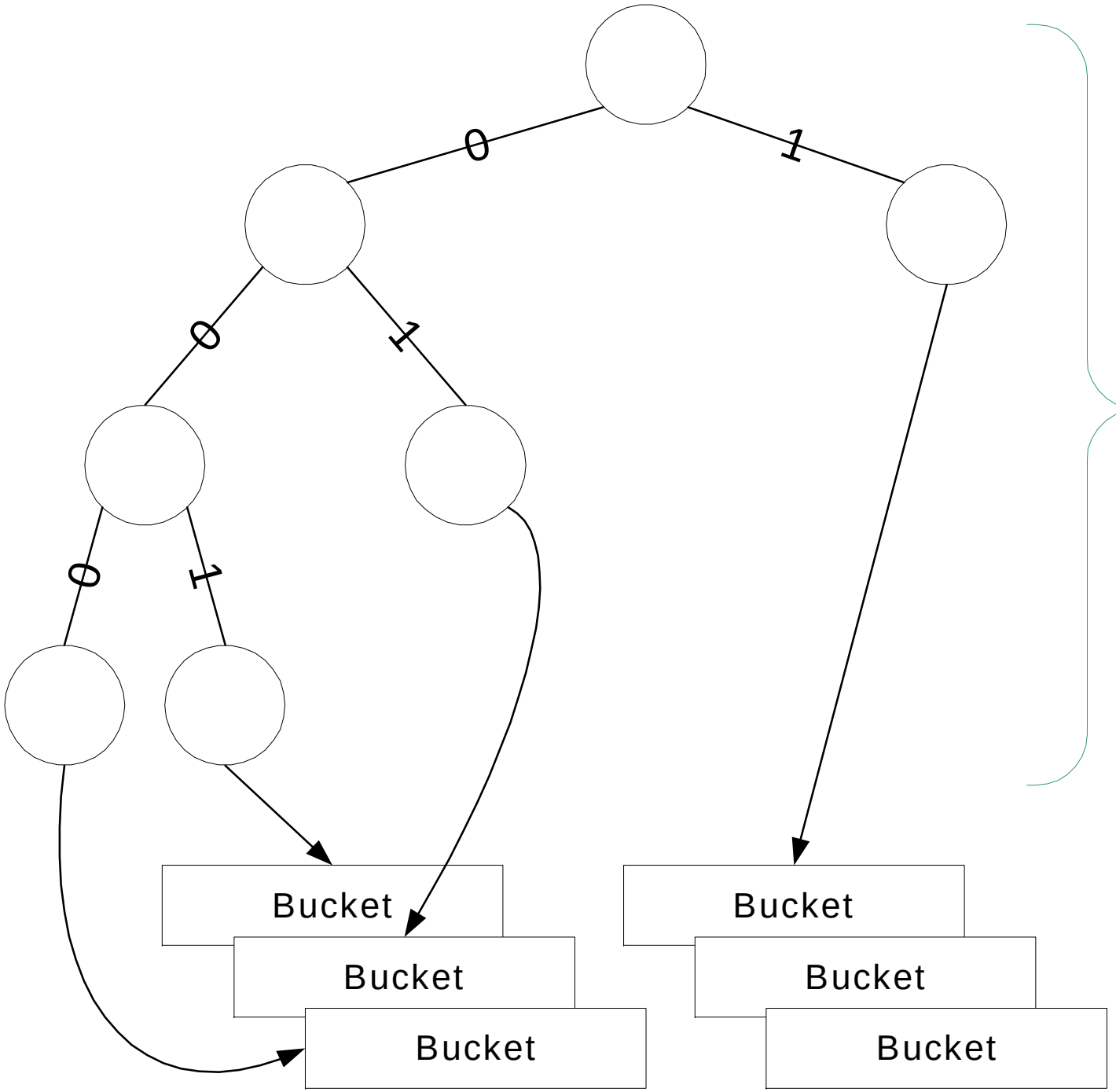
Statisches Hashing

- Einfacher Ansatz:
  - Hashfunktion h(x) = x mod N, wobei N die Größe der Tabelle in Seiten ist.
  - h(x) gibt die Seite an, auf der der Eintrag zu Schlüssel x zu finden ist.

- Probleme:
  - 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



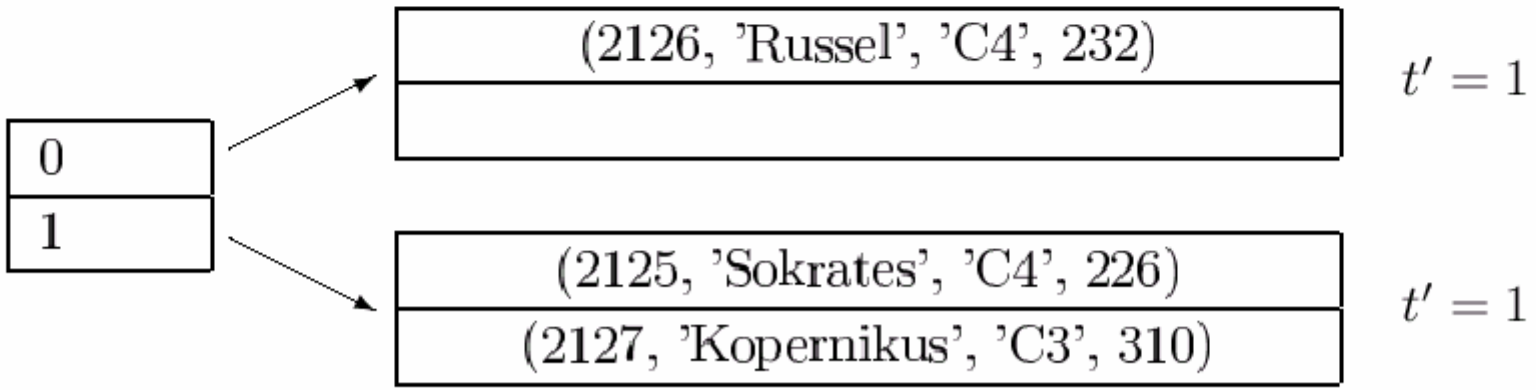
binärer Trie, Entschei-dungs-baum Directory

Hashfunktion für erweiterbares Hashing

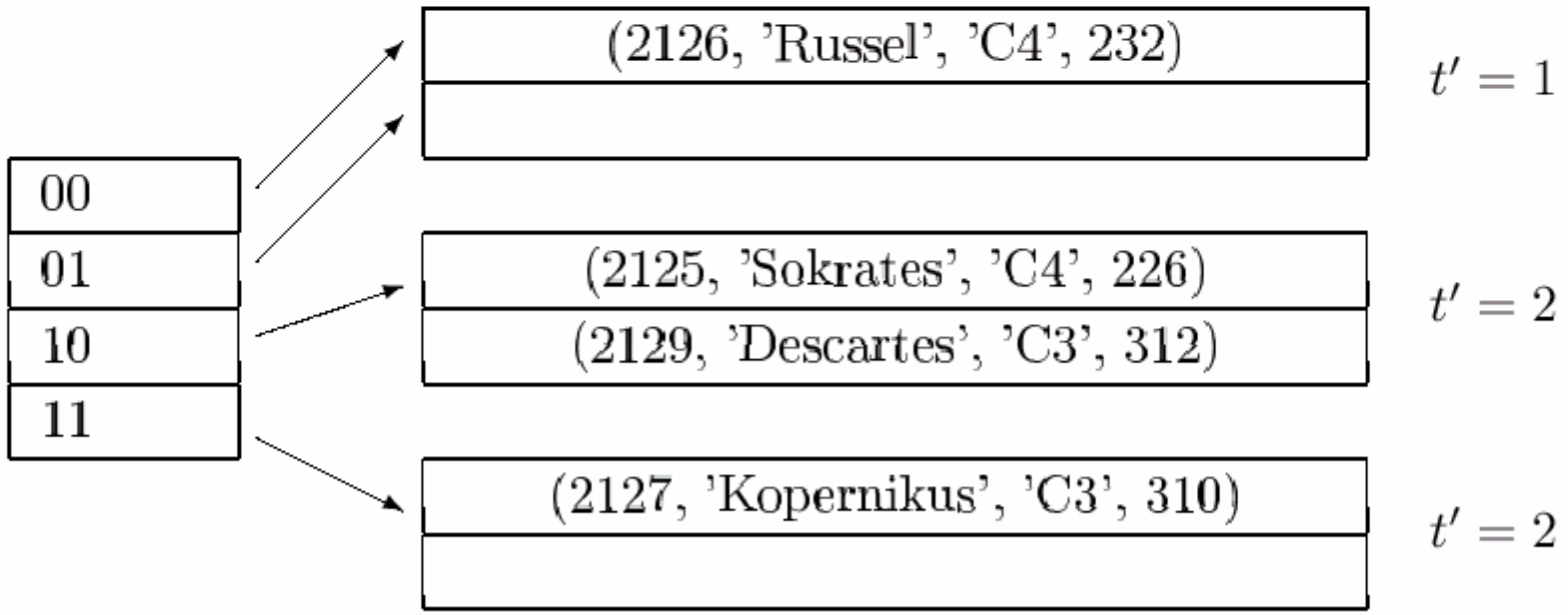
- h: Schlüsselmenge $\rightarrow$ {0,1}*
- Der Bitstring muss lang genug sein, um alle Objekte auf ihre Buckets abbilden zu können
- Anfangs wird nur ein (kurzer) Präfix des Hashwertes (Bitstrings) benötigt
- Wenn die Hashtabelle wächst wird aber sukzessive ein längerer Präfix benötigt
- Beispiel-Hashfunktion: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

Demonstration des erweiterbaren Hashings

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

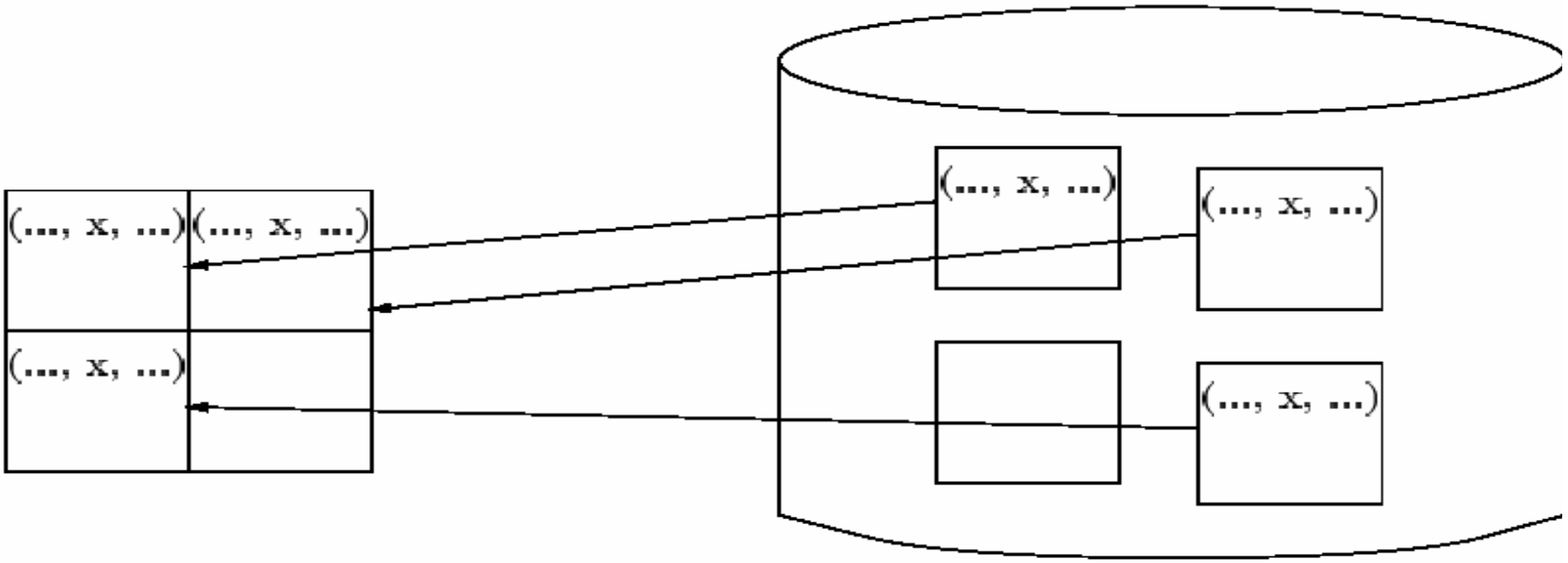


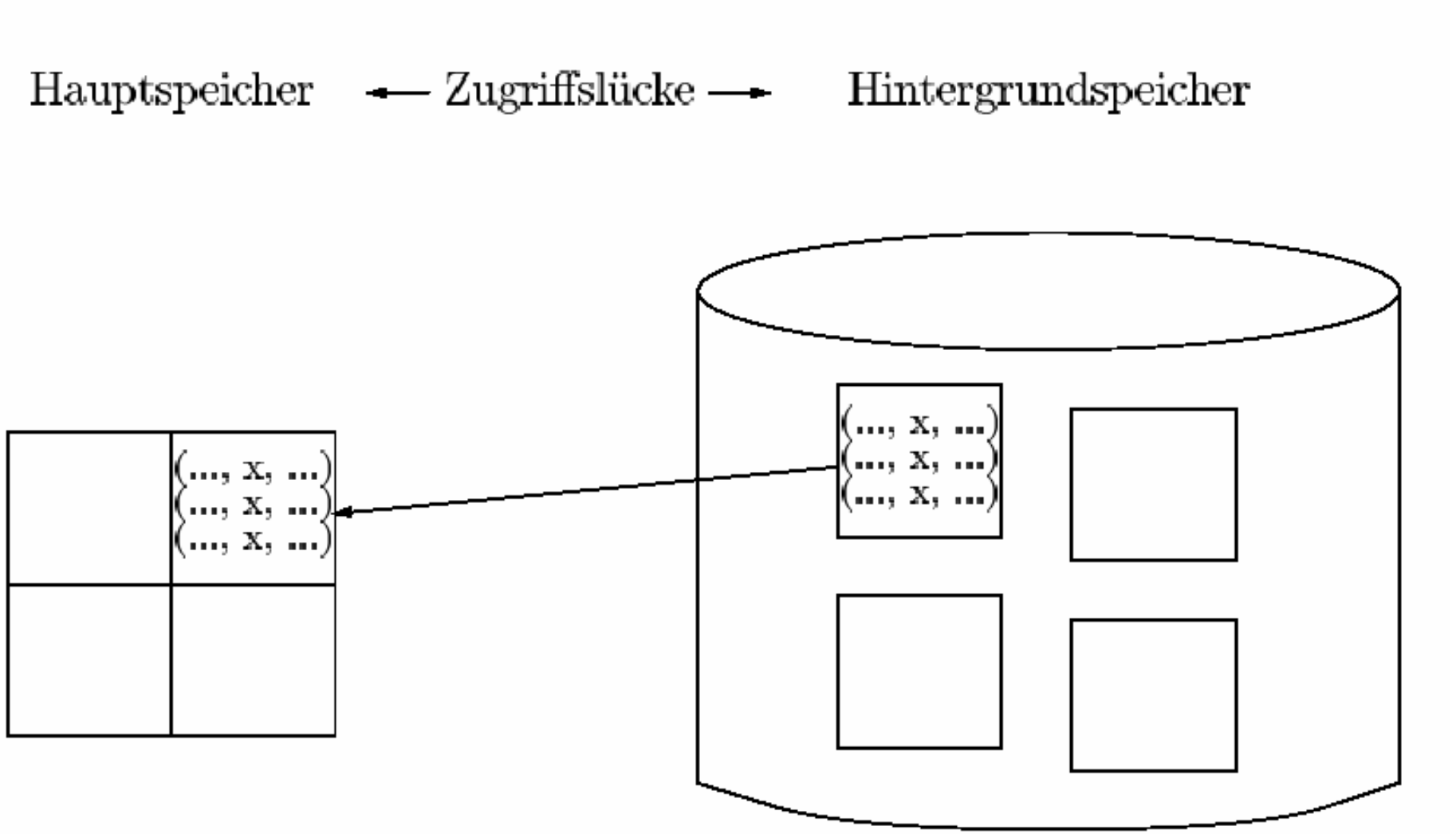
x	$h(x)$	
	d	p
2125	10	1100100001
2126	01	1100100001
2127	11	1100100001
2129	10	0010100001



Objektballung / Clustering logisch verwandter Daten

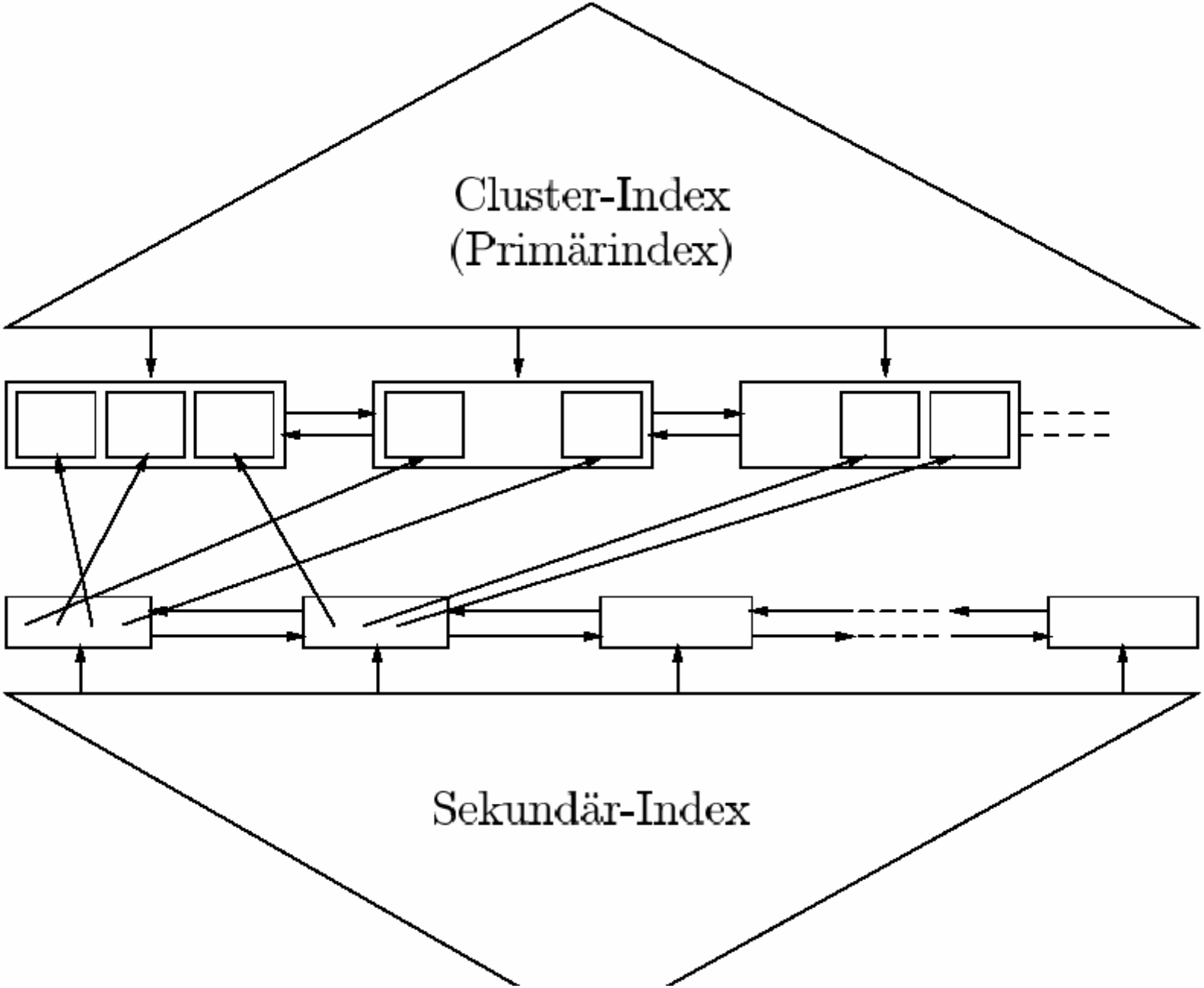
select $*$
from R
where $A = x$;




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
```