Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Übung Datenbanksysteme II

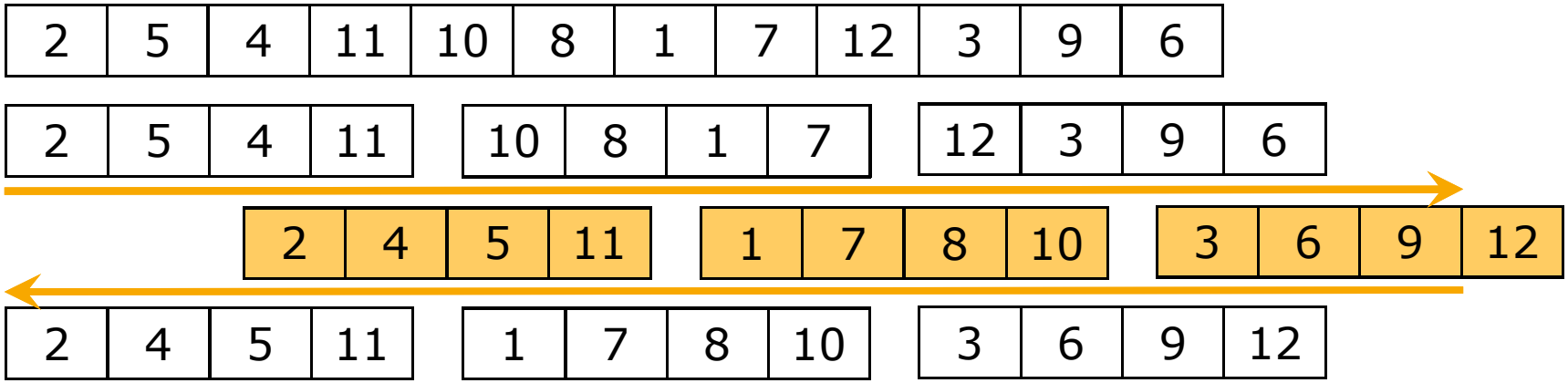
Index-strukturen

Leon Bornemann

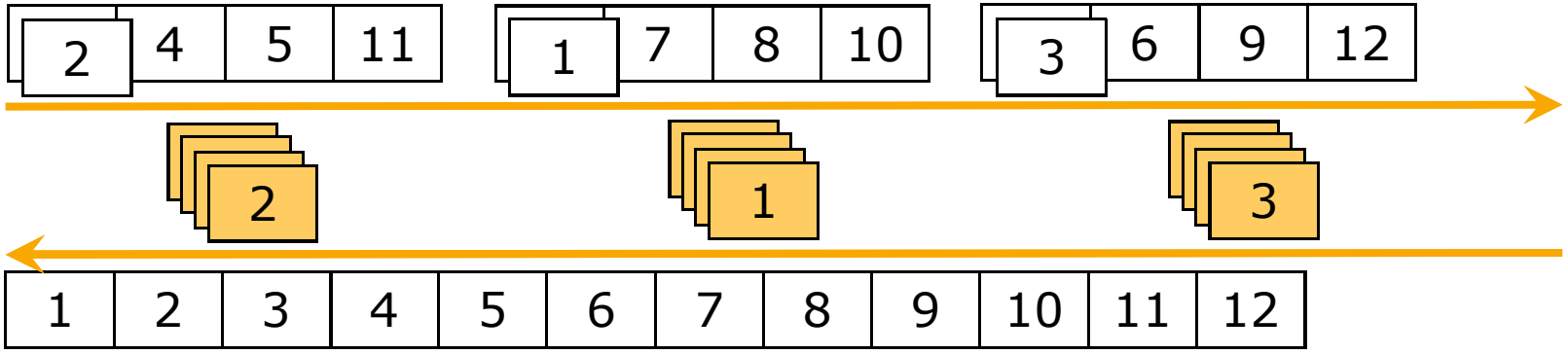
Folien basierend auf Maximilian Jenders, Thorsten Papenbrock

Rückblick: TPMMS

- Phase 1:



- Phase 2:



Hausaufgabe 1 – TPMMS Phase 1

- Phase 1 grundlegender Algorithmus:
  - Solange noch unsortierte „Teillisten" existieren:
    - Hauptspeicher füllen
    - Sortieren (vernachlässigbar)
    - Schreiben sortierter Listen

$$t_{phase1} = 1t_{readFullData} + t_{writeFullData}$$

$$t_{phase1} = 2 \cdot t_{readFullData}$$

- → jeden Block einmal lesen und schreiben
- Optimale Verteilung → Lesen ganzer Spuren am Stück
  - Unter anderem benötigt: Wie viele Spuren nehmen die Daten ein?

Bsp: 12 Blöcke, Platz für 5
Variante 1: 5,5,2
Variante 2: 4,4,4

- Zwei Varianten des Lesens:
  - Hauptspeicher jedes Mal komplett füllen
  - Alle Daten über alle Füllungen gleichmäßig aufteilen

Hausaufgabe 1 – TPMMS Phase 1

(vernachlässigbar)

$$t_{readFullData} = t_{fillMemory} \cdot numFillings + t_{rot} \cdot (numFillings - 1)$$

$$t_{fillMemory} = t_{seek} + numTracksPerFilling \cdot t_{Umdrehung} + t_{trackChanges}$$

$$t_{trackChanges} = 1.002 \cdot (\lceil numTracksPerFilling \rceil - 1) ms$$

$$numTracksPerFilling = \frac{numBlocksPerFilling}{BlocksPerTrack}$$

$$numFillings = \lceil \frac{numBlocksTotal}{numBlocksPerFilling} \rceil$$

Als Kommazahl verwenden!

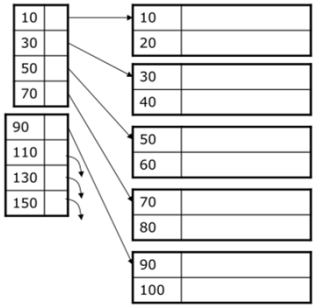
Hausaufgabe 1 – TPMMS Phase 2

- Wesentlich einfacher:
  - Jeden Block einmal lesen und schreiben
  - Diesmal durchschnittliche Latenzzeit pro Block
    - Wir wissen nicht wo wir als nächstes Lesen müssen
    - Kein Prefetching
    - Ort der sortierten Listen unklar

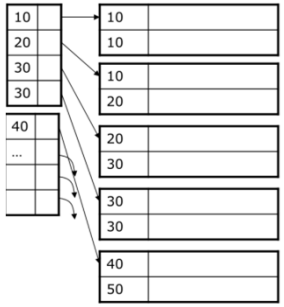
Quiz: Zuordnung

Indexstrukturen

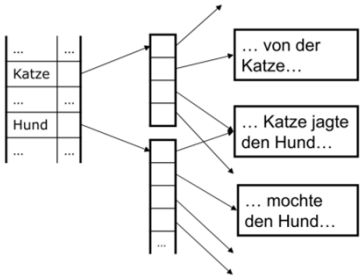
b.



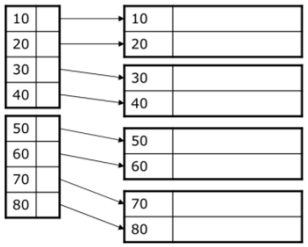
d.



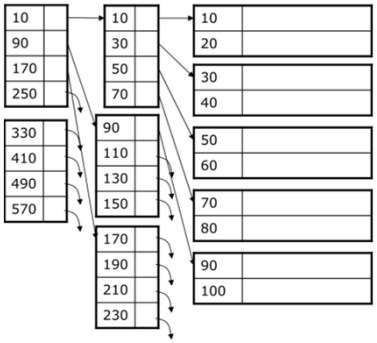
f.



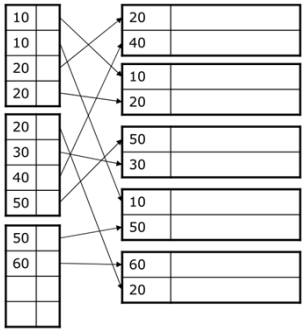
a.



c.



e.



a. dichtbesetzter Index
b. dünnbesetzter Index
c. mehrstufiger Index
d. nicht-eindeutige Attribute
e. Sekundärindex
f. invertierter Index

Indexe

- Es ist für jedes Datenfile möglich, zwei separate *dünnbesetzte* Level-1-Indexe auf unterschiedlichen Attributen anzulegen.

  Falsch Bei zwei Schlüsseln kann nur nach einem Schlüssel sortiert werden. Beide dünnbesetzte Level-1-Indexe benötigten daher eine Sortierung nach ihrem Schlüssel-Attribut. Das ist gleichzeitig nicht möglich!

- Es ist für jedes Datenfile möglich, zwei separate *dichtbesetzte* Level-1-Indexe auf unterschiedlichen Attributen anzulegen.

  Wahr Dichtbesetzte Level-1-Indexe benötigen keine Sortierung. Es können daher mehrere dichtbesetzte Indexe auf unterschiedlichen Schlüsseln angelegt werden!

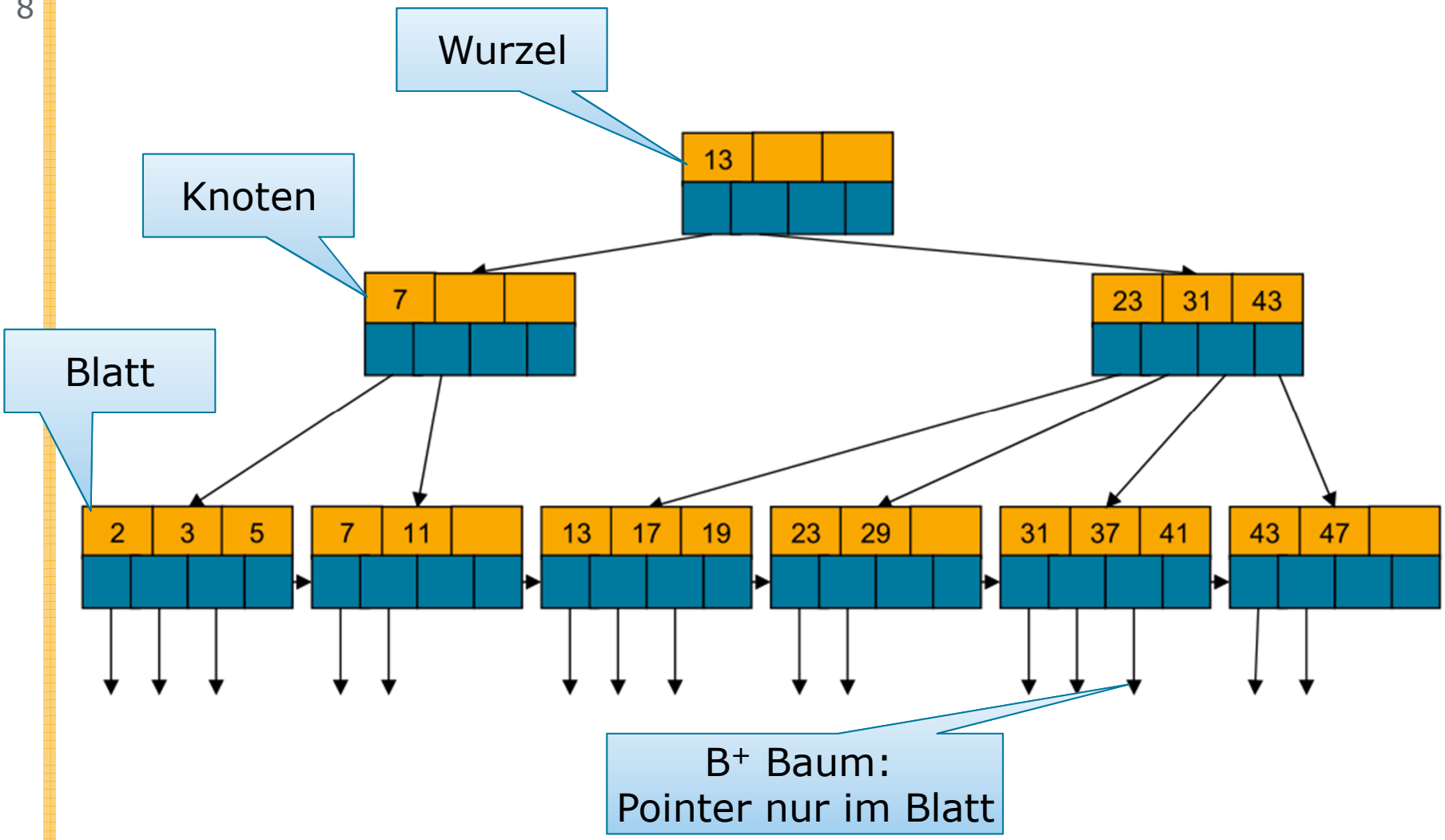
- Es ist für jedes Datenfile möglich, einen *dünnbesetzten* Level-1-Index mit einem *dichtbesetzten* Level-2-Index anzulegen. Beide Indexe sind sinnvoll.

  Falsch Ein dichtbesetzter Level-2-Index auf einen Level-1-Index ist sinnlos, da er alle Werte noch einmal indexiert, dabei zusätzlich Speicher belegt und keinen Mehrwert schafft.

- Es ist für jedes Datenfile möglich, einen *dichtbesetzten* Level-1-Index mit einem *dünnbesetzten* Level-2-Index anzulegen. Beide Indexe sind sinnvoll.

  Wahr Ein dünnbesetzter Level-2-Index auf einem dichtbesetzten Level-1-Index ist sinnvoll, da der dünnbesetzte Index nur die ersten Elemente der Blöcke indexiert und so letztendlich Speicher und Zugriffszeit spart.

B+-Bäume: Aufbau



B+-Bäume: Eigenschaften

- Perfekt balanciert
- Parameter n
  - n Suchschlüssel pro Knoten
  - n+1 Pointer pro Knoten
  - ➔ Jeder Suchschlüssel S hat einen „linken“ und einen „rechten“ Pointer.
- Wurzel
  - Mindestens zwei Pointer
- Knoten
  - Mindestens $\lceil (n+1)/2 \rceil$ Pointer
  - Suchschlüssel S ist kleinster Schlüssel im rechten Teilbaum
- Blätter
  - Letzter Pointer zeigt auf nächstes Blatt
  - Zusätzlich mindestens $\lfloor (n+1)/2 \rfloor$ Pointer

Quiz: Richtig oder Falsch?
B⁺-Bäume

- B⁺-Bäume können Overflow-Buckets benötigen.

  Falsch Statt Overflow-Buckets zu generieren nutzen B⁺-Baume *split*-Operationen um den Index beim Überlauf eines Knotens oder Blattes zu erweitern.

- Ein Block eines B⁺-Baums speichert genau n Pointer und n+1 Suchschlüssel.

  Falsch Ein Block (= Knoten bzw. Blatt) speichert *bis zu* **n Suchschlüssel** und *bis zu* **n+1 Pointer**.

- Der durch Löschen eines Schlüssels entstehende B⁺-Baum ist nicht eindeutig bestimmt.

  Wahr Beim Geschwister-Klauen und beim Merge ist nicht eindeutig bestimmt, mit welchem anderen Knoten die Operation durchgeführt wird!

- B⁺-Bäume können nur als Primärindexe verwendet werden.

  Falsch B⁺-Baume können verschiedene Index-Rollen übernehmen. Dabei können sie auch eine Sortierung des indexierten Attributes nutzen, benötigen diese aber nicht zwangsläufig. B ⁺-Baume sind daher nicht nur als Primärindex einsetzbar.

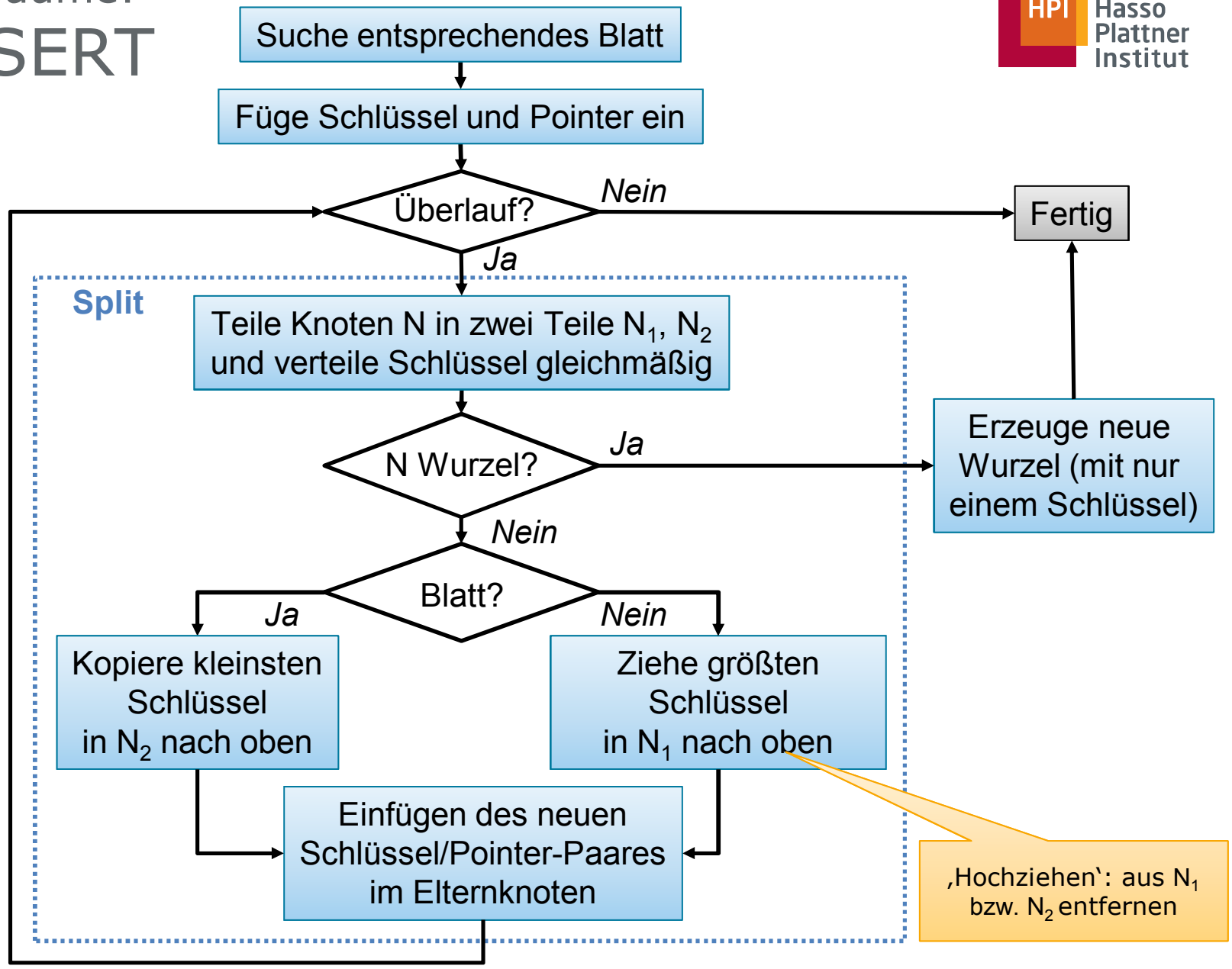
- B⁺-Bäume können auf eindeutigen und nicht-eindeutigen Attributen angelegt werden.

  Wahr Ein B+-Baum ist ein mehrstufiger Index, der sowohl auf eindeutigen als auch auf nicht-eindeutigen Attributen aufgebaut werden kann. Bei doppelten Attributwerten können mehrere Schlüssel-Pointer-Paare eingefügt werden mit NULL-Werten in den Knoten, falls im Kind keine neuen Werte vorkommen.

- B⁺-Bäume sind im Allgemeinen für Einzelanfragen effizienter als Hashtabellen.

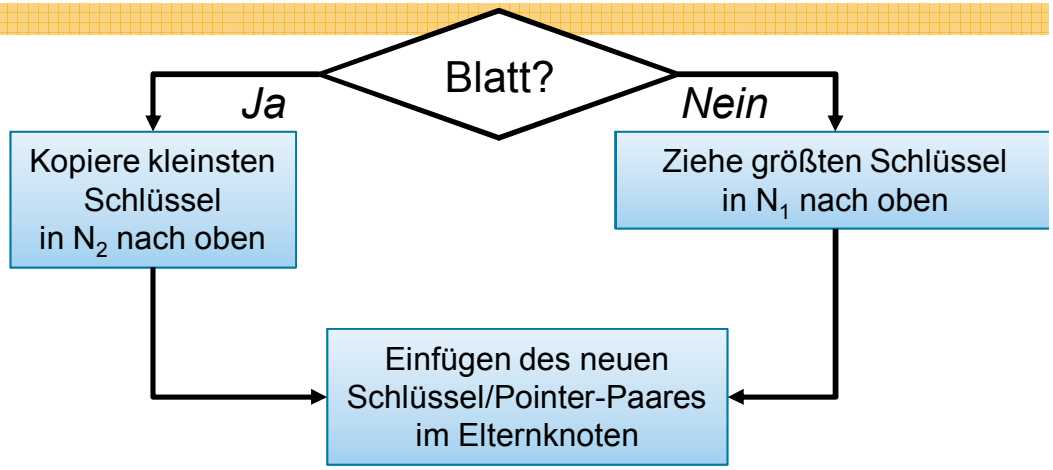
  Falsch B+-Bäume müsen für Einzelanfragen bis maximal über die gesamten Höhe des Baums traversieren.

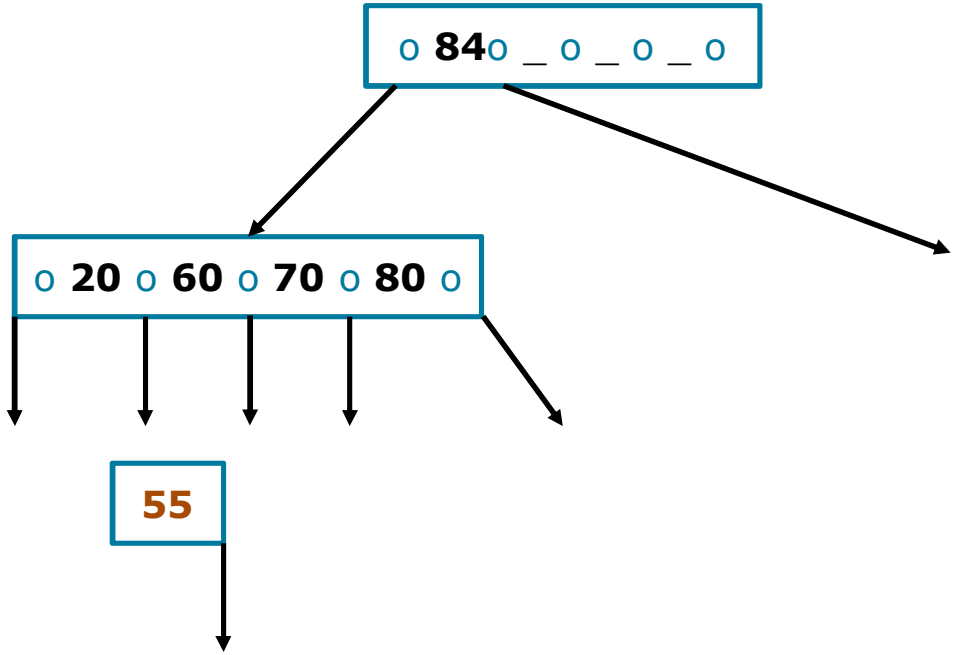
B+-Bäume: INSERT



B+-Bäume: INSERT

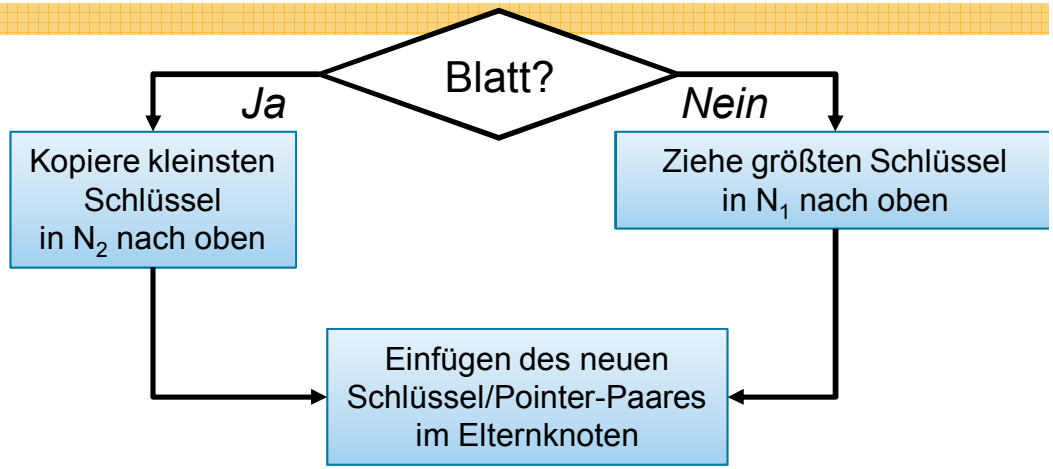
- Warum funktioniert das?
- Überlauf bei n+1 Schlüsseln (n+2 Pointer)
- Beispiel: n=4

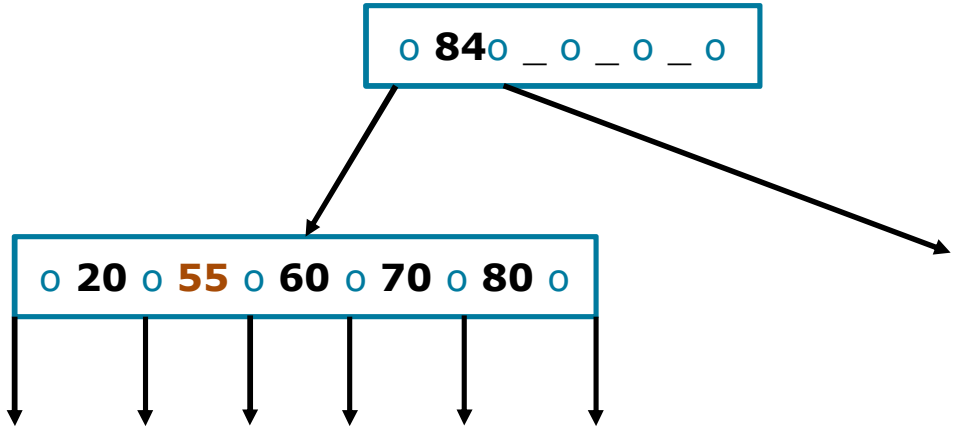




B+-Bäume: INSERT

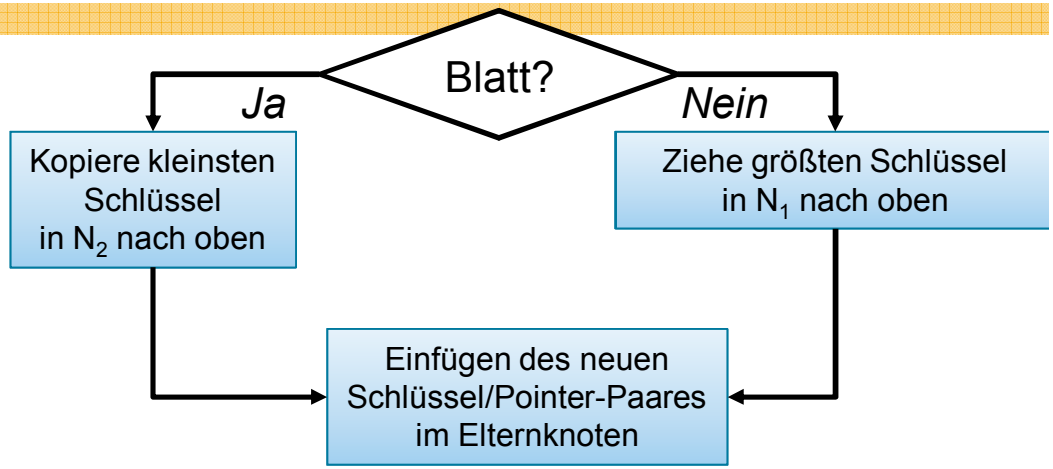
- Warum funktioniert das?
- Überlauf bei n+1 Schlüsseln (n+2 Pointer)
- Beispiel: n=4

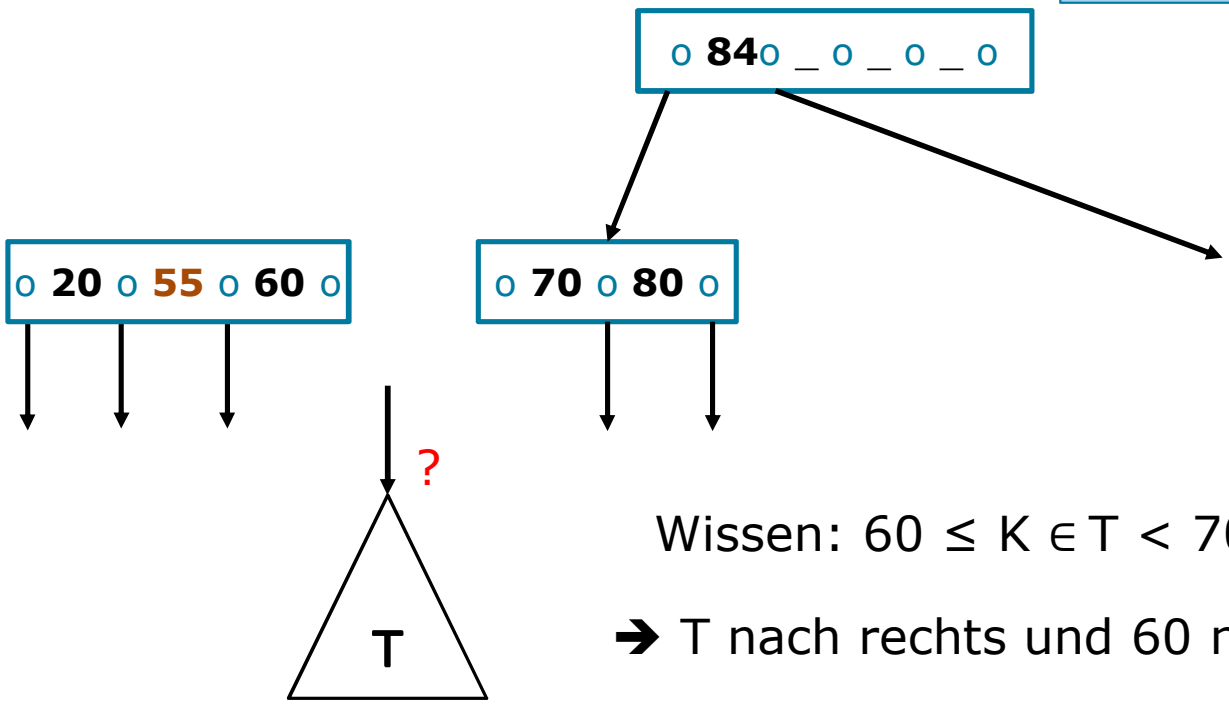




B+-Bäume: INSERT

- Warum funktioniert das?
- Überlauf bei n+1 Schlüsseln (n+2 Pointer)
- Beispiel: n=4



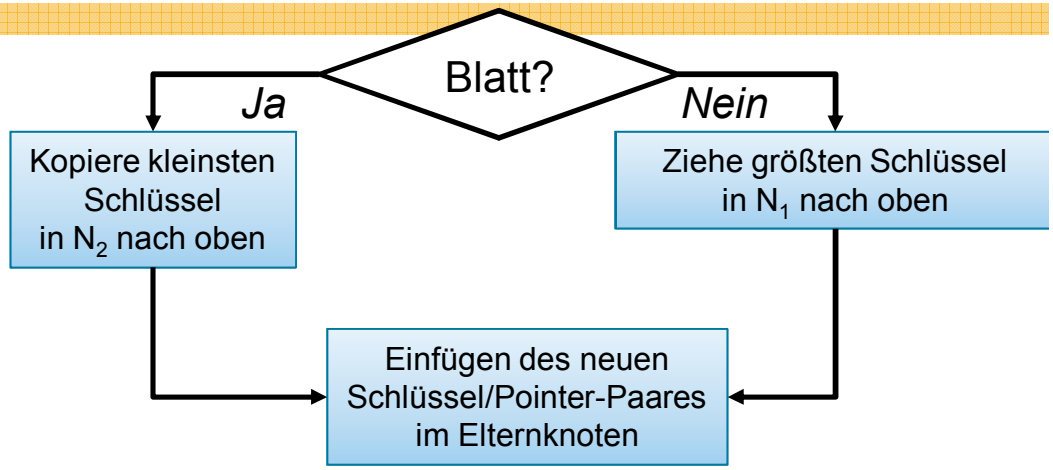


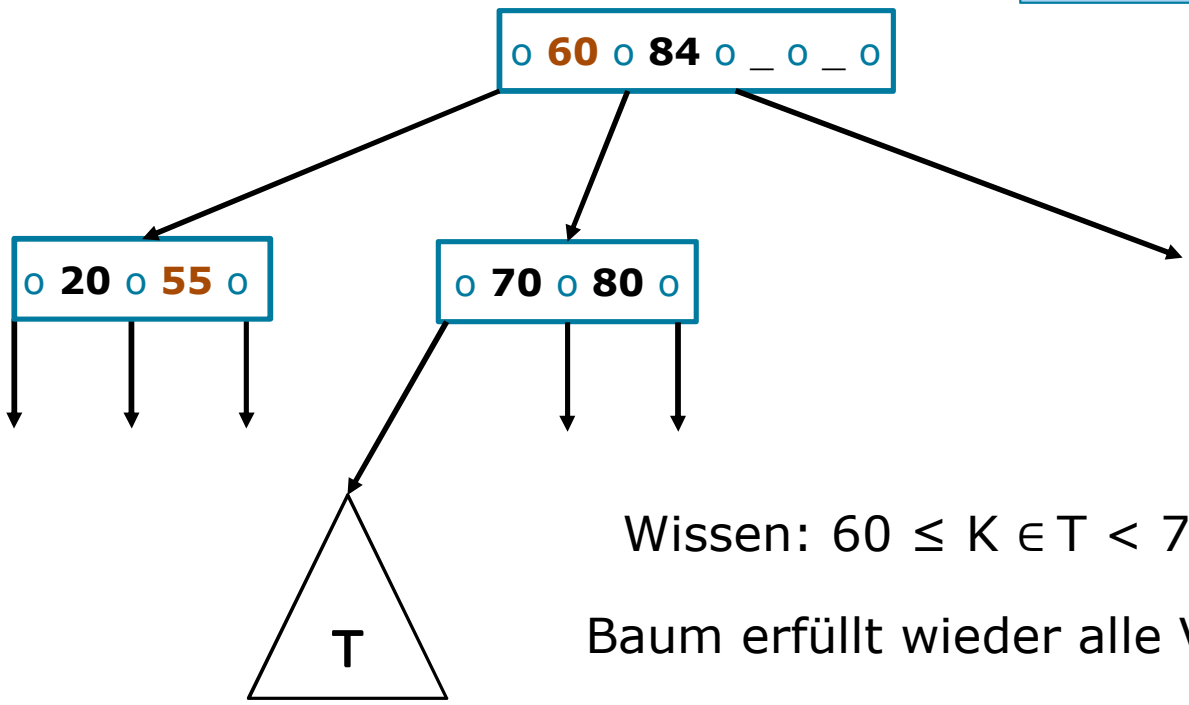
Wissen: $60 \leq K \in T < 70$ und $60 \in T$

➔ T nach rechts und 60 nach oben

B+-Bäume: INSERT

- Warum funktioniert das?
- Überlauf bei n+1 Schlüsseln (n+2 Pointer)
- Beispiel: n=4



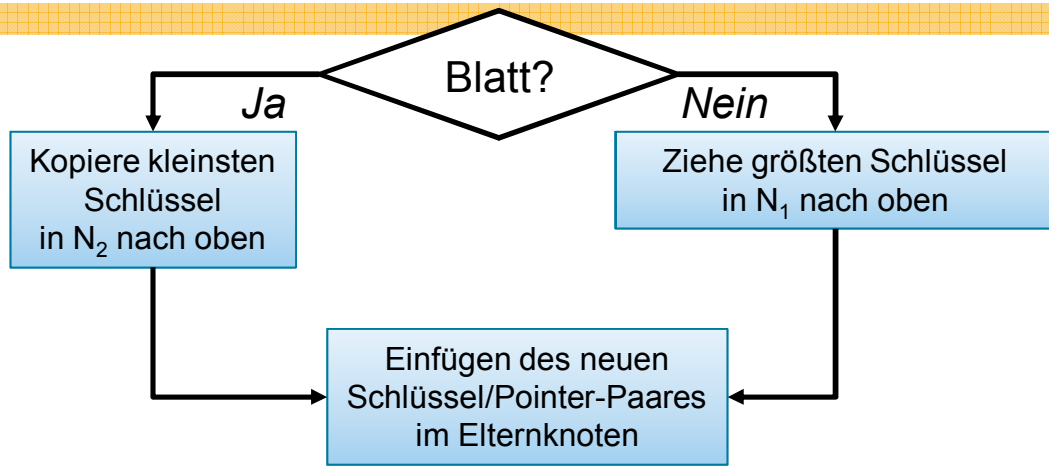


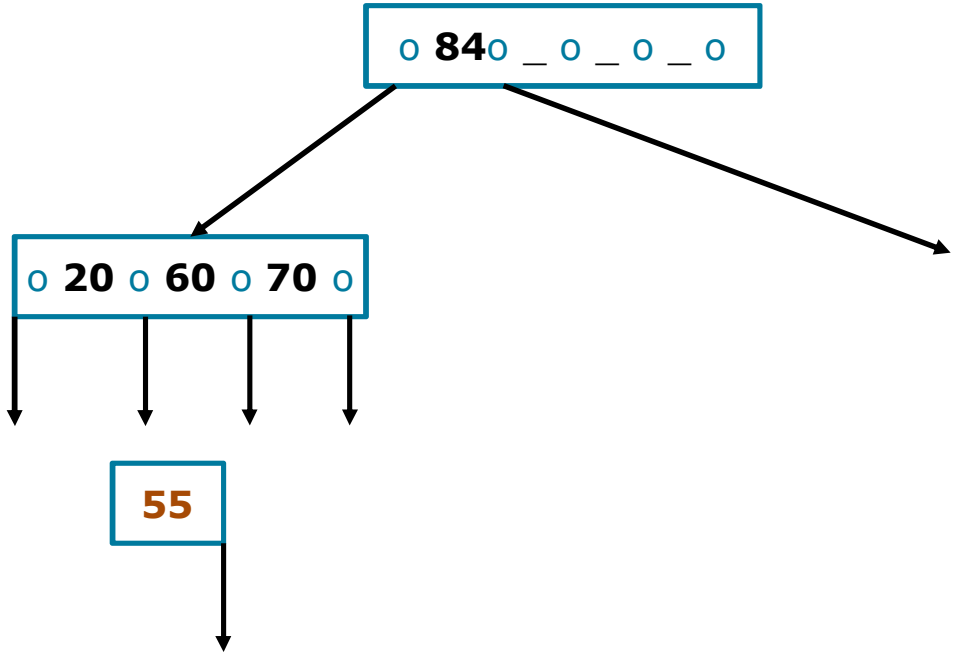
Wissen: $60 \leq K \in T < 70$ und $60 \in T$

Baum erfüllt wieder alle Vorgaben

B+-Bäume: INSERT

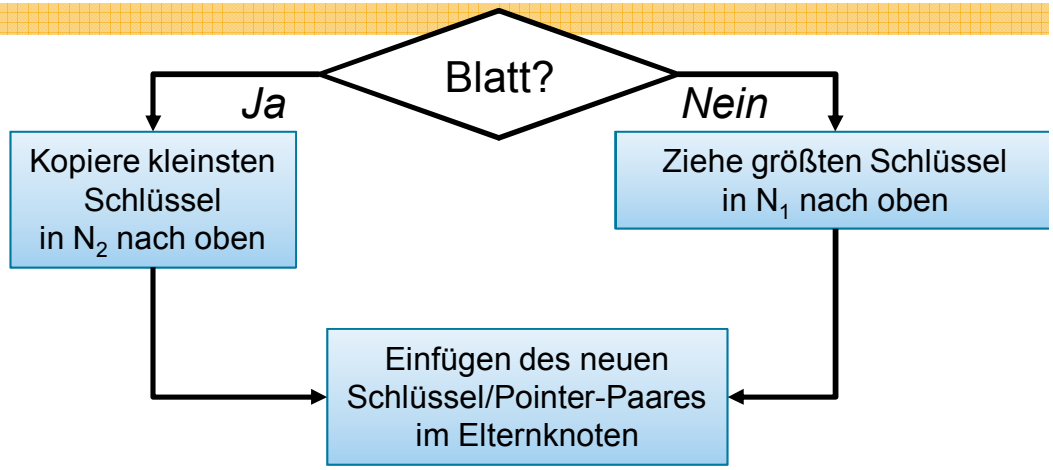
- Warum funktioniert das?
- Überlauf bei n+1 Schlüsseln (n+2 Pointer)
- Beispiel: n=3

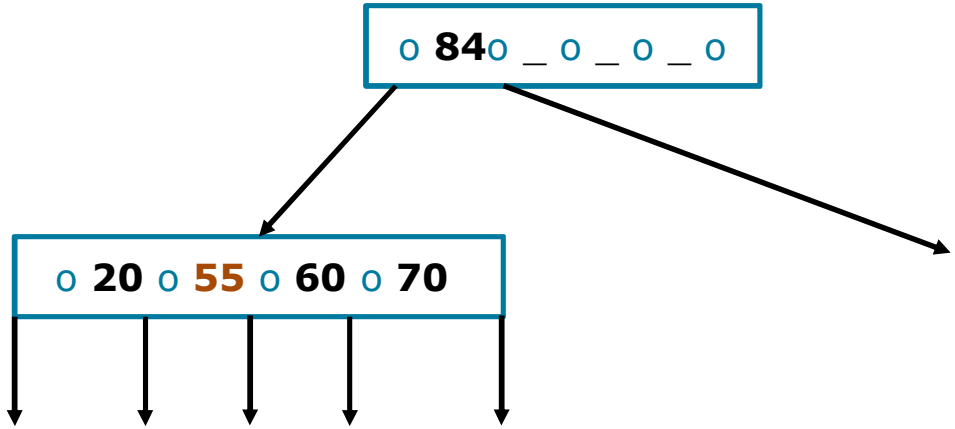




B+-Bäume: INSERT

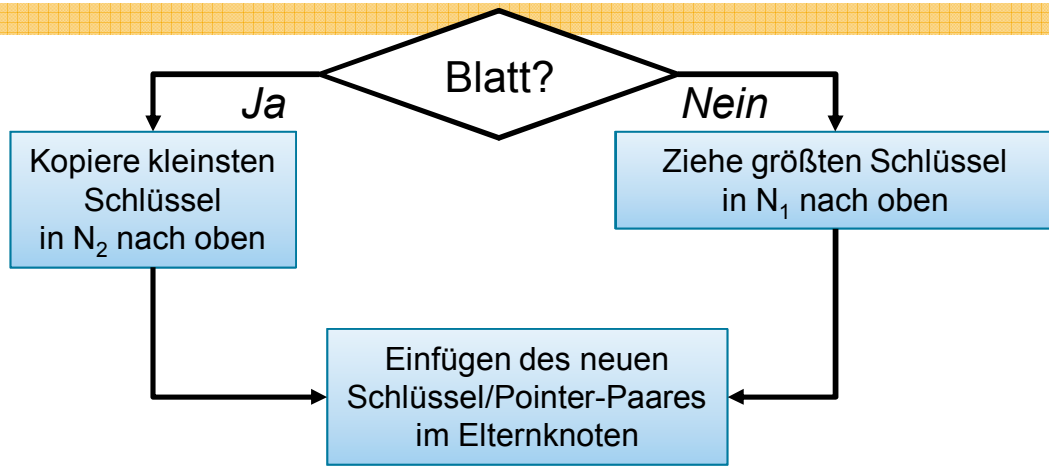
- Warum funktioniert das?
- Überlauf bei n+1 Schlüsseln (n+2 Pointer)
- Beispiel: n=3

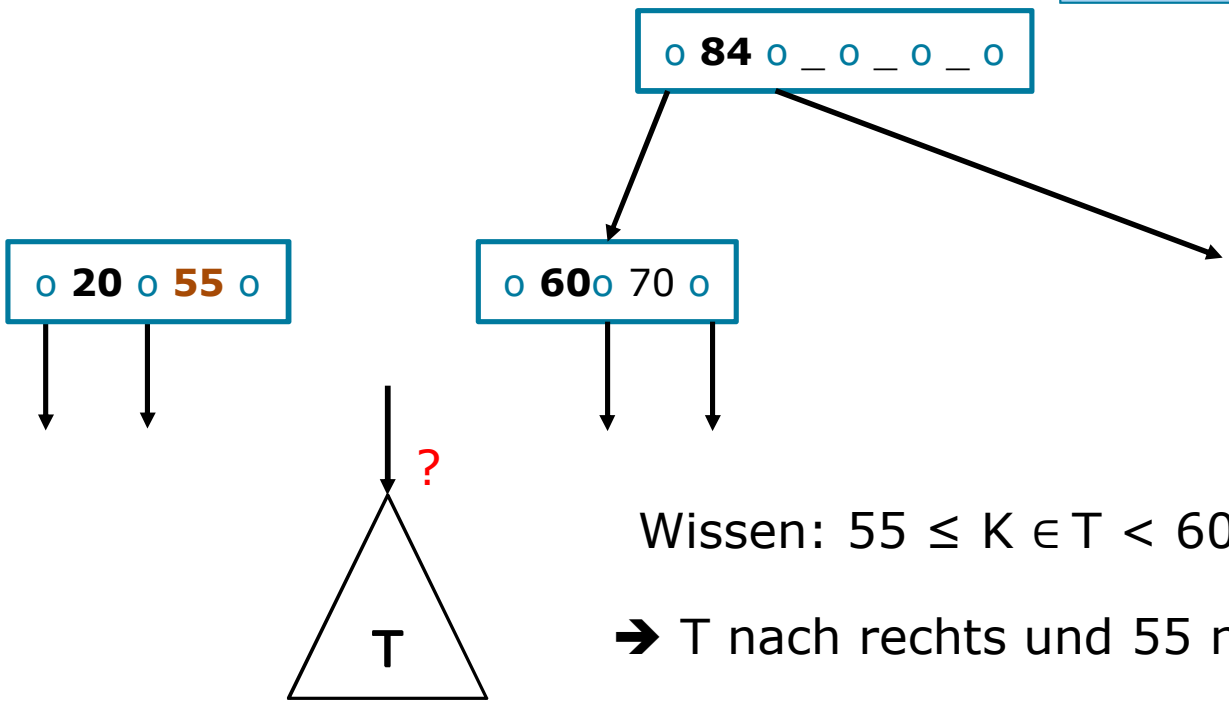




B+-Bäume: INSERT

- Warum funktioniert das?
- Überlauf bei n+1 Schlüsseln (n+2 Pointer)
- Beispiel: n=4



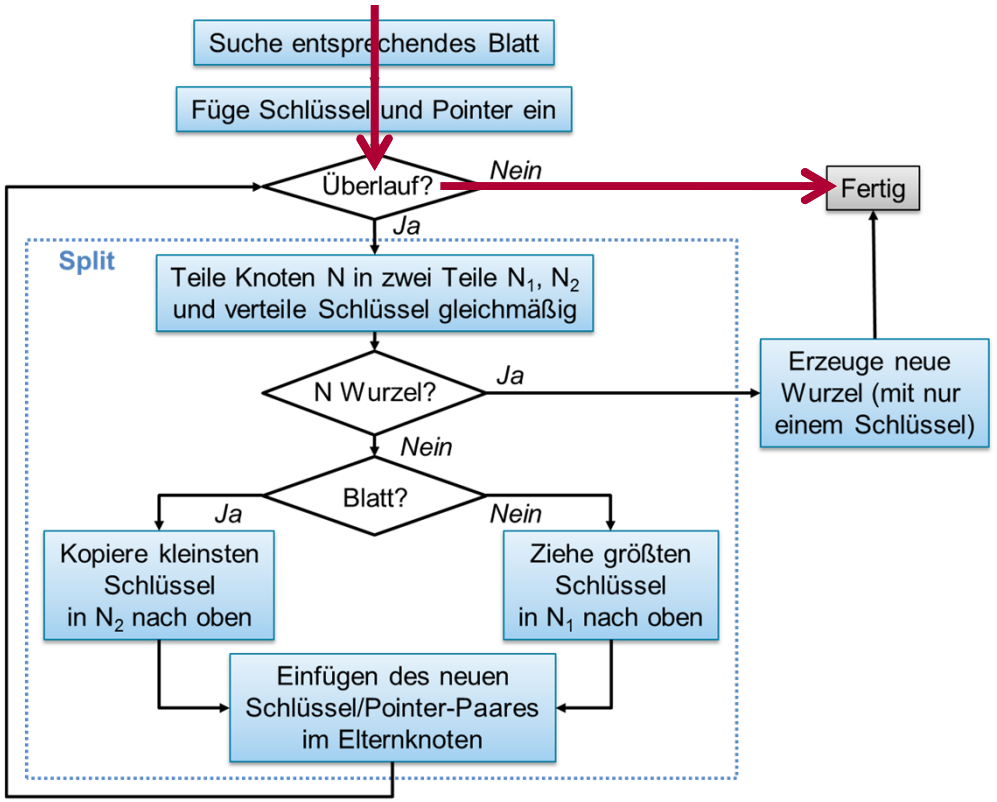


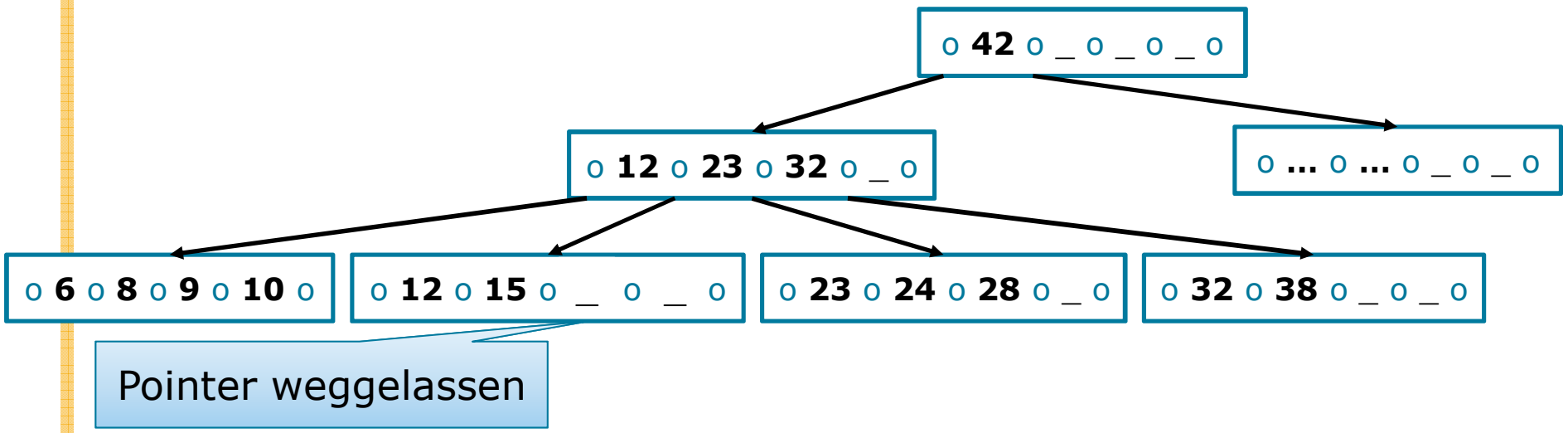
Wissen: $55 \leq K \in T < 60$ und $55 \in T$

➔ T nach rechts und 55 nach oben

B+-Bäume: INSERT

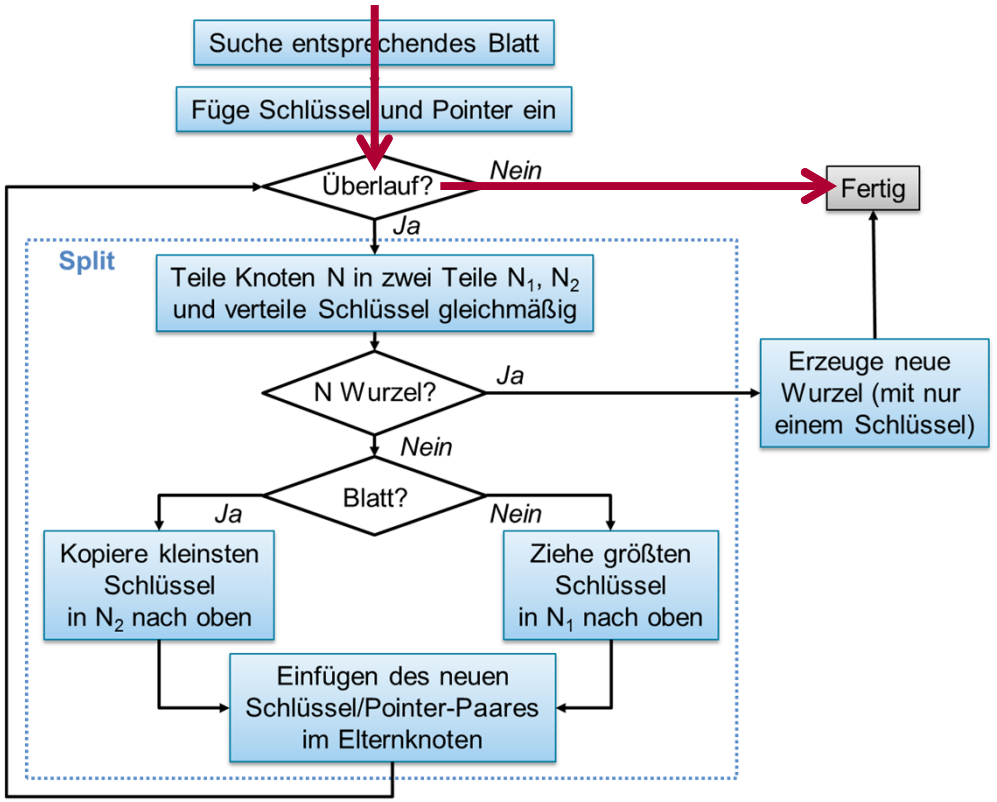
- INSERT(17)

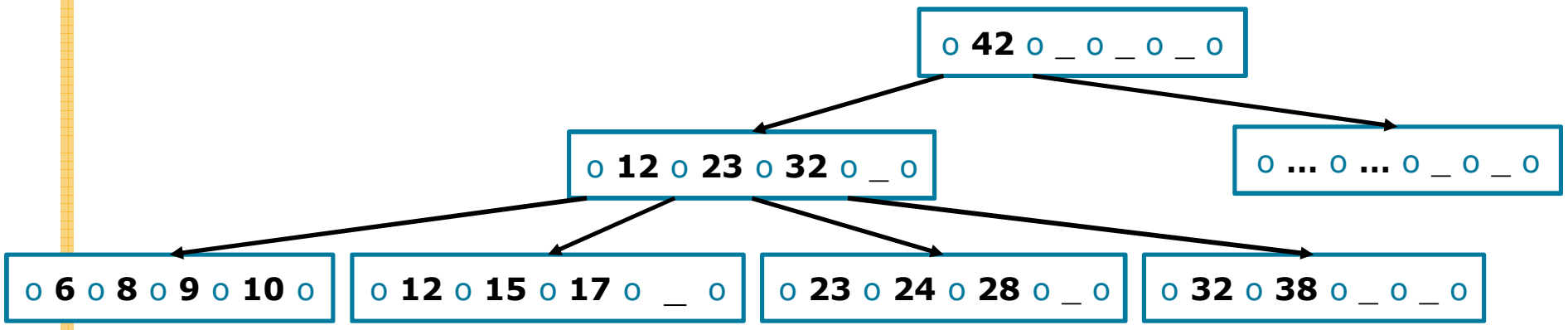




B+-Bäume: INSERT

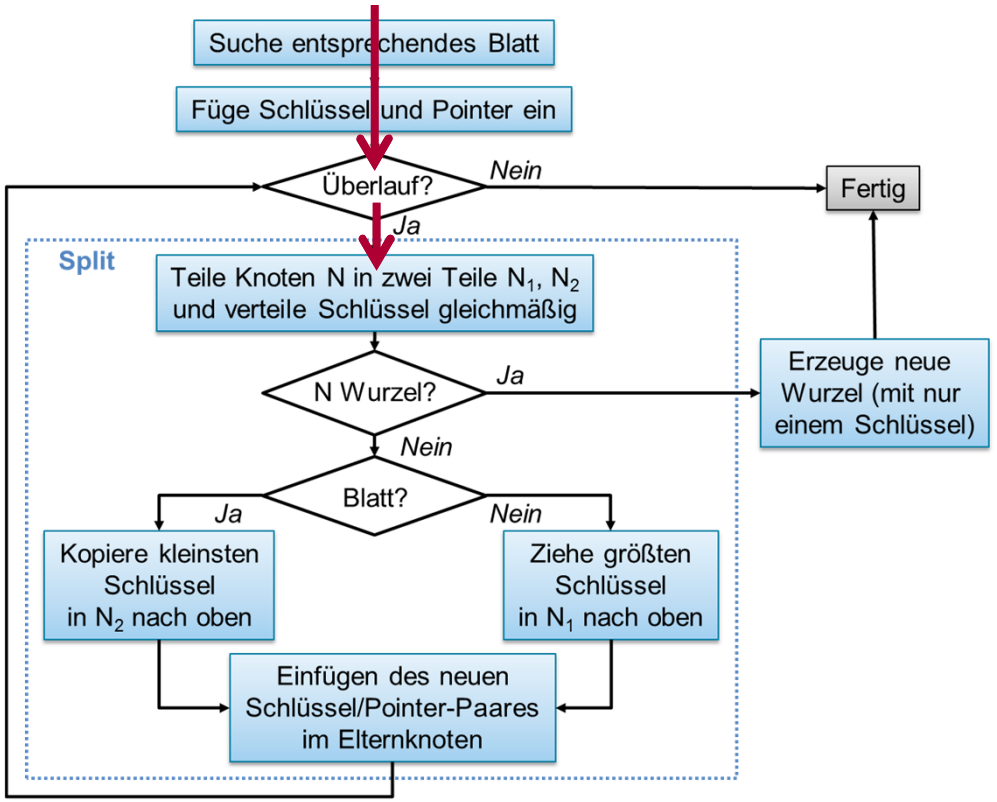
- INSERT(17)
- INSERT(18)

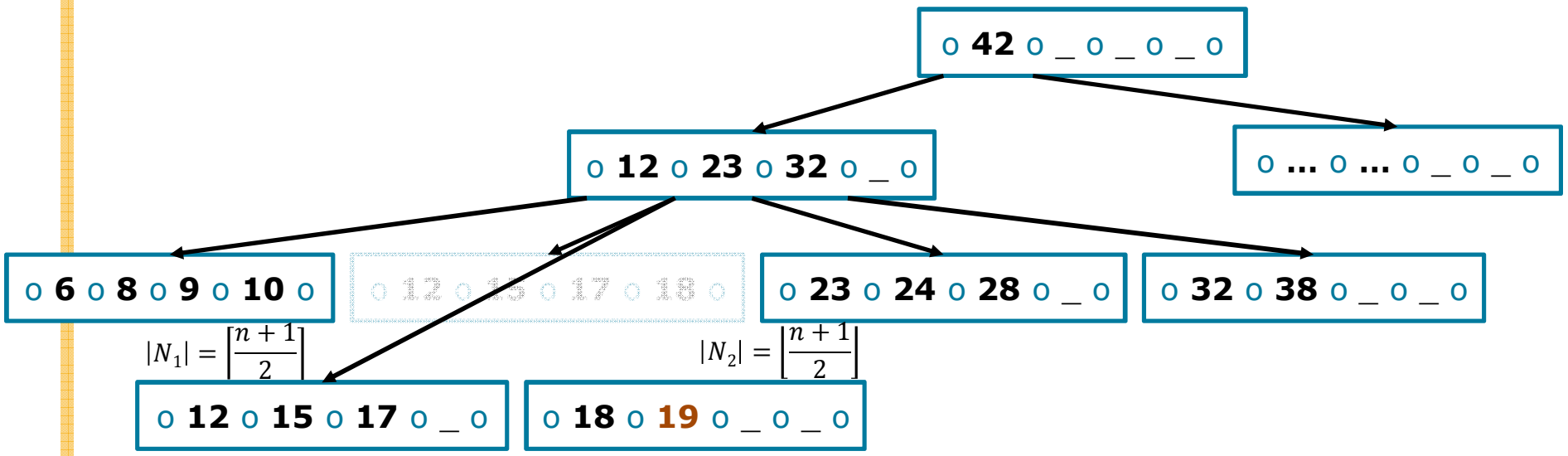




B+-Bäume: INSERT

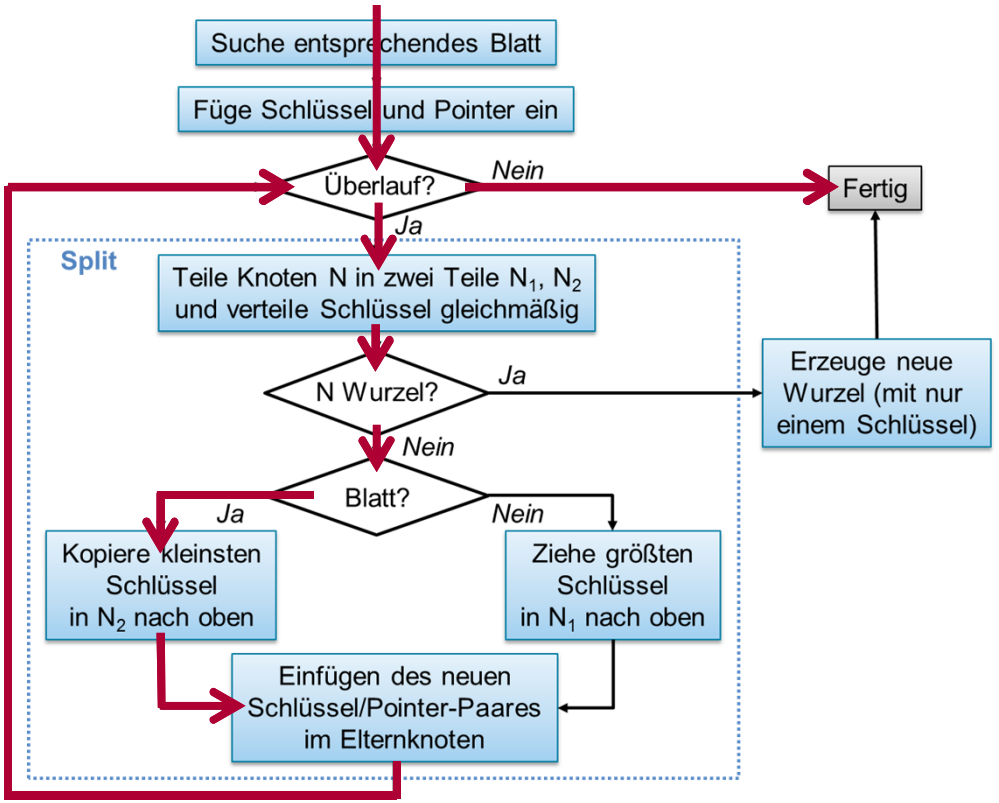
- INSERT(17)
- INSERT(18)
- INSERT(19)

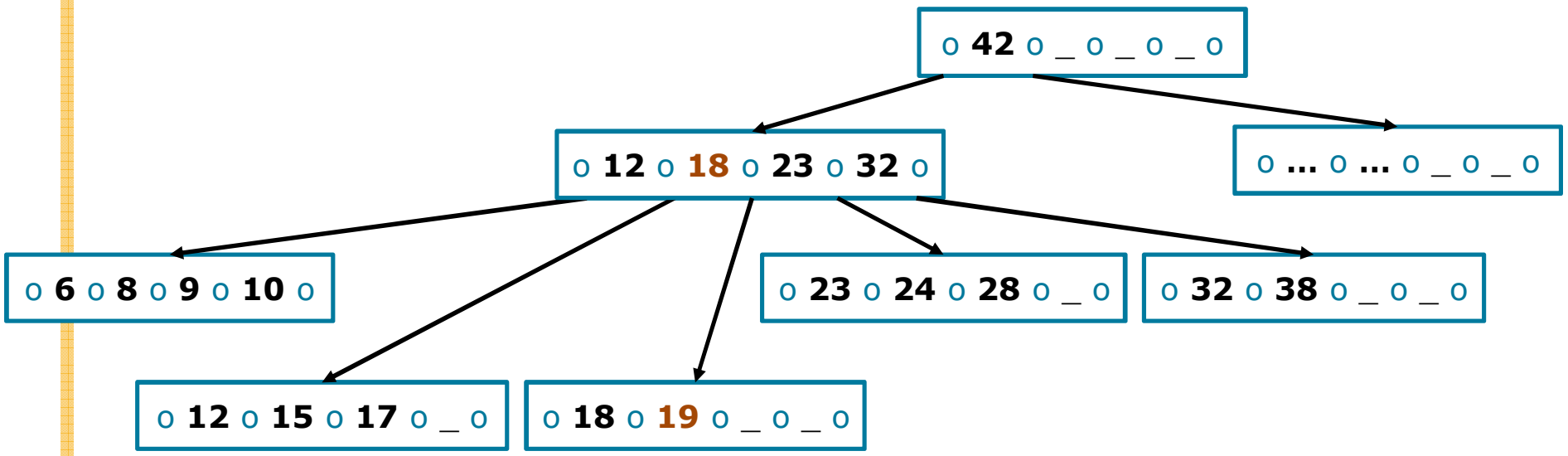




B+-Bäume: INSERT

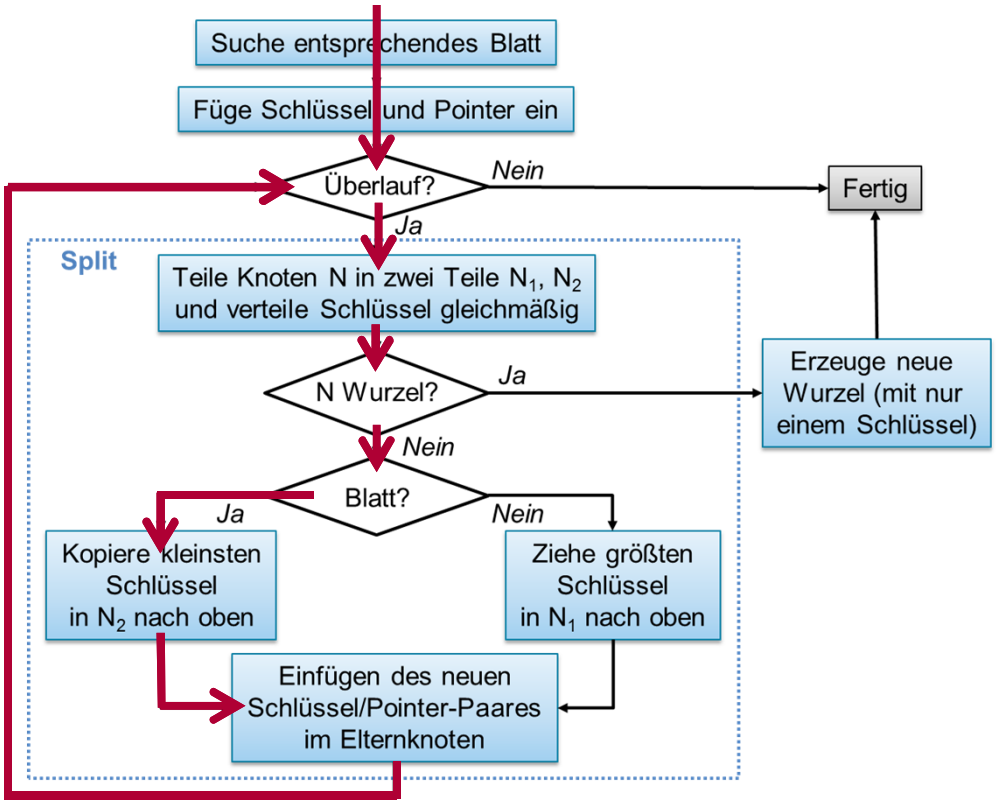
- INSERT(17)
- INSERT(18)
- INSERT(19)

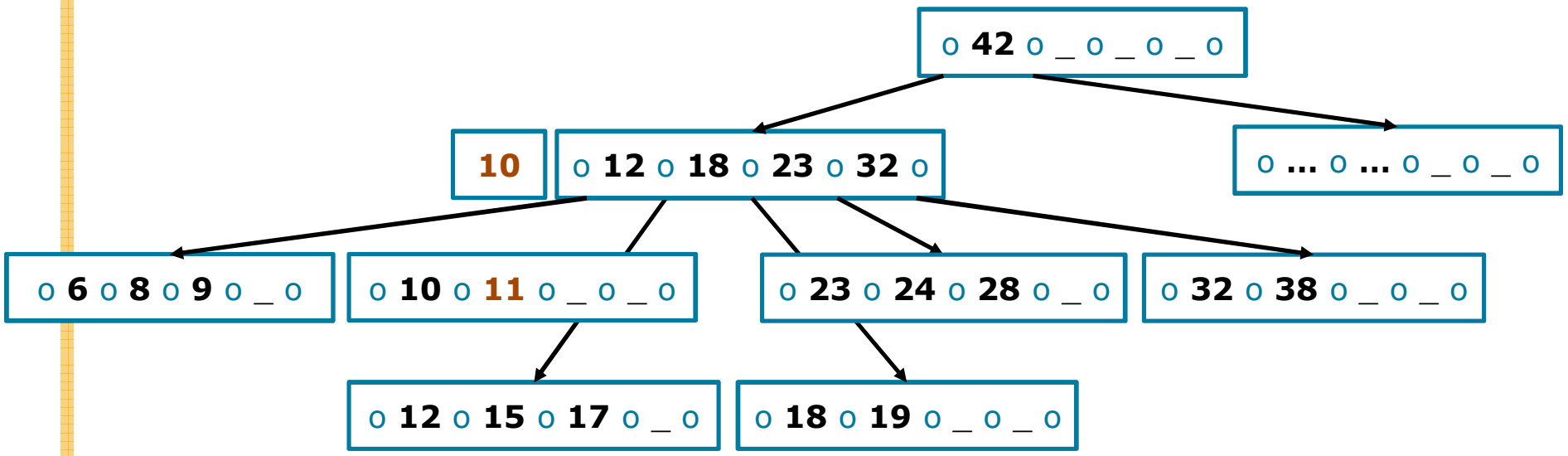




B+-Bäume: INSERT

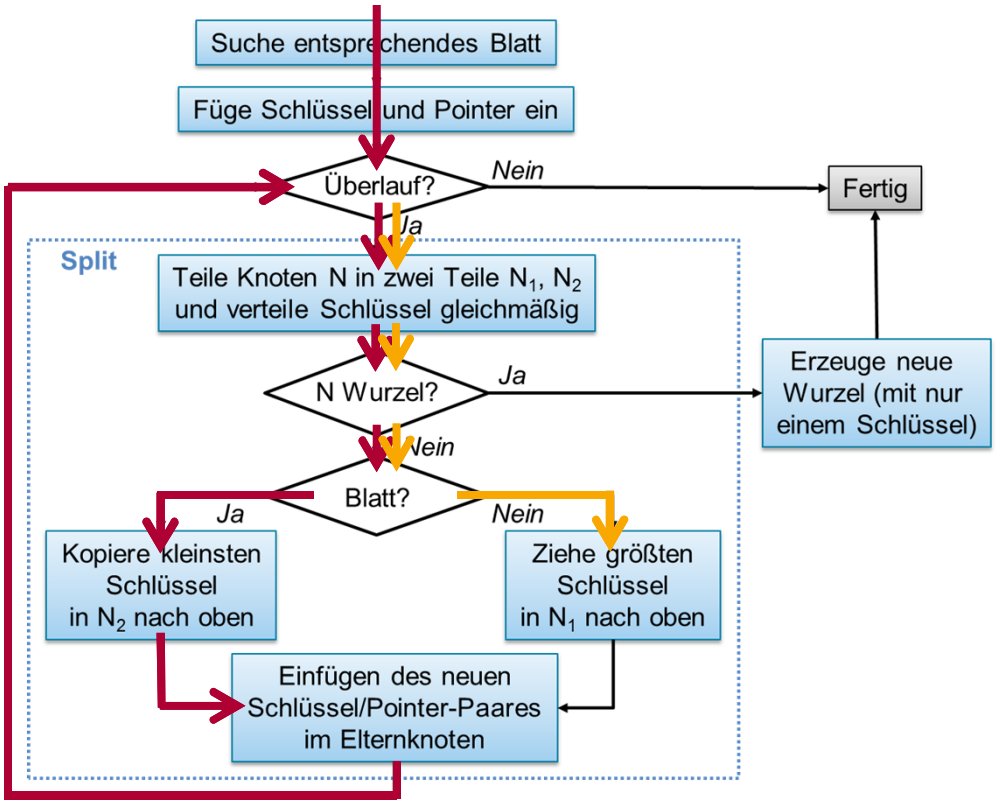
- INSERT(17)
- INSERT(18)
- INSERT(19)
- INSERT(11)

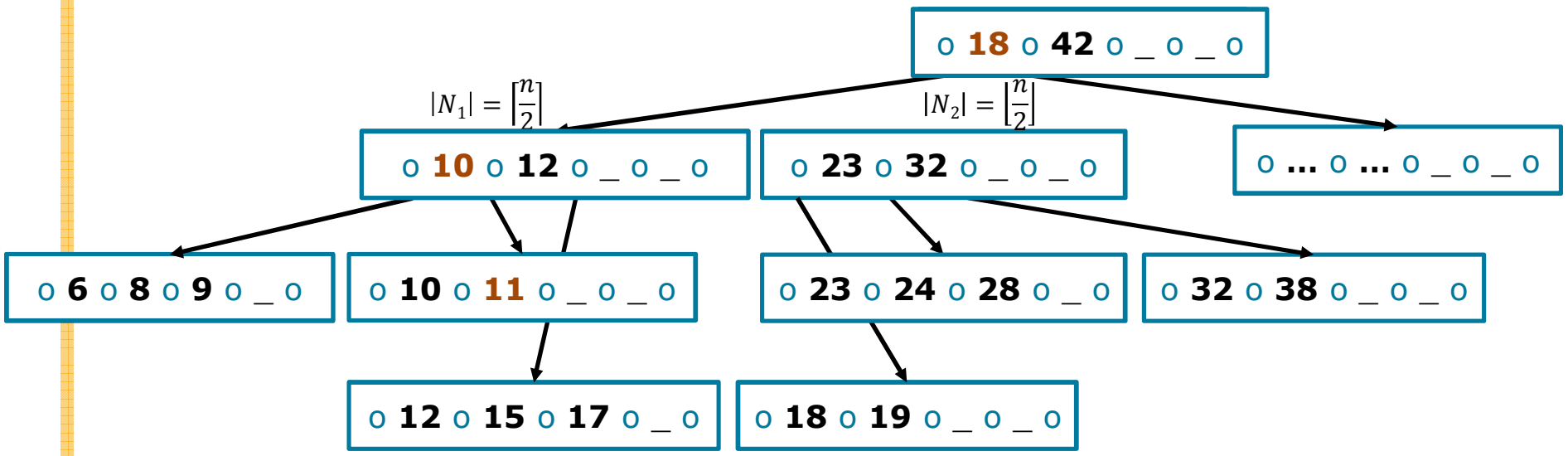




B+-Bäume: INSERT

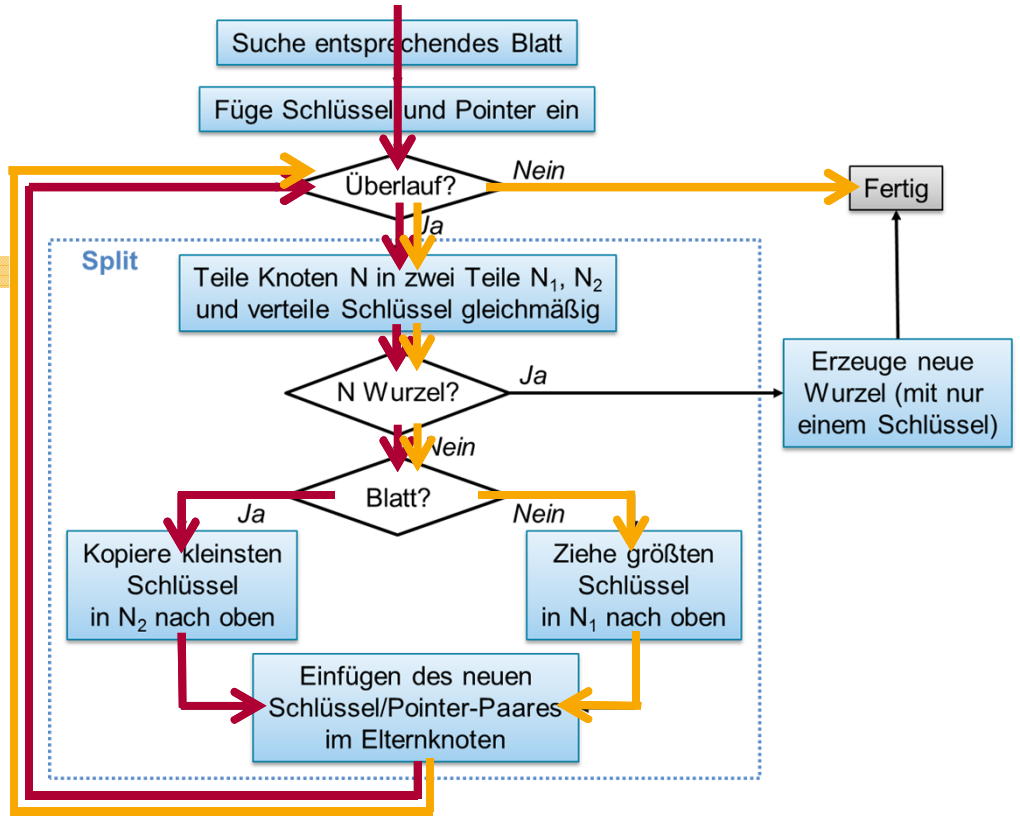
- INSERT(17)
- INSERT(18)
- INSERT(19)
- INSERT(11)

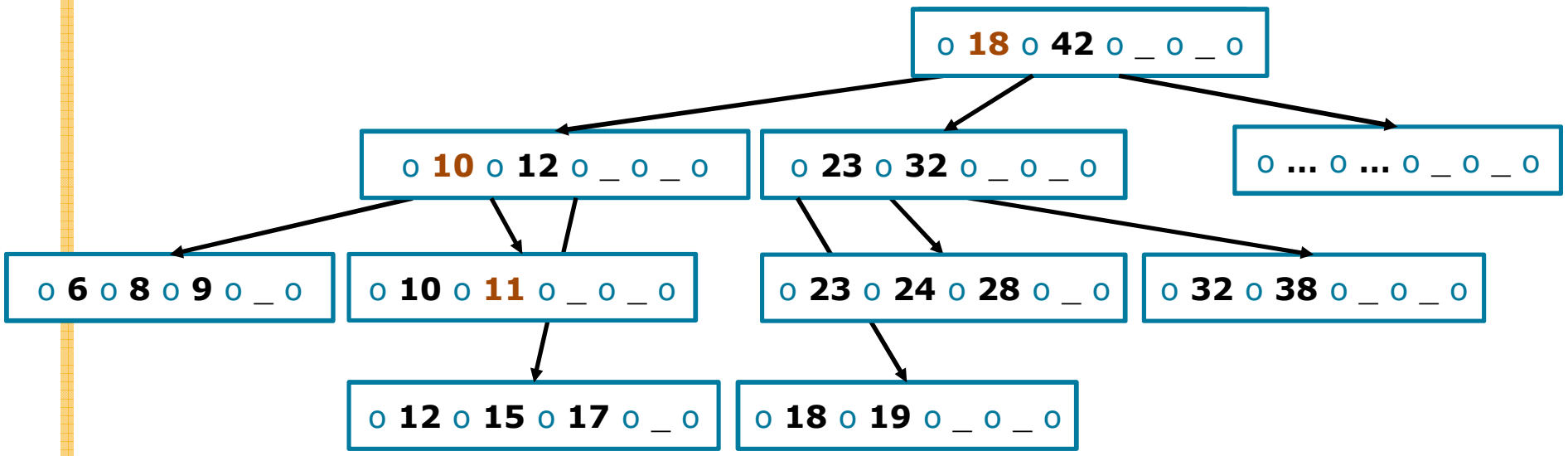




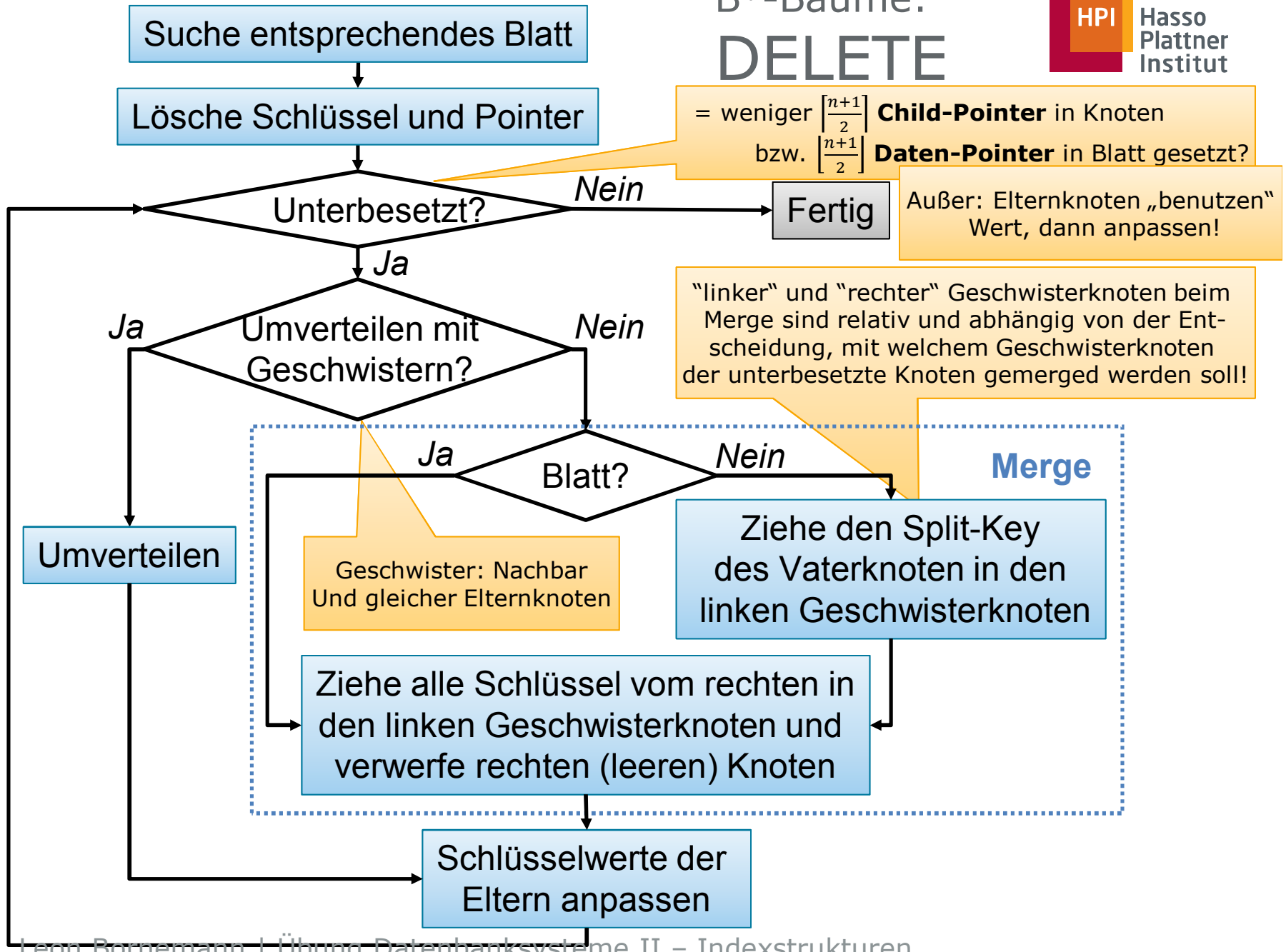
B+-Bäume: INSERT

- INSERT(17)
- INSERT(18)
- INSERT(19)
- INSERT(11)



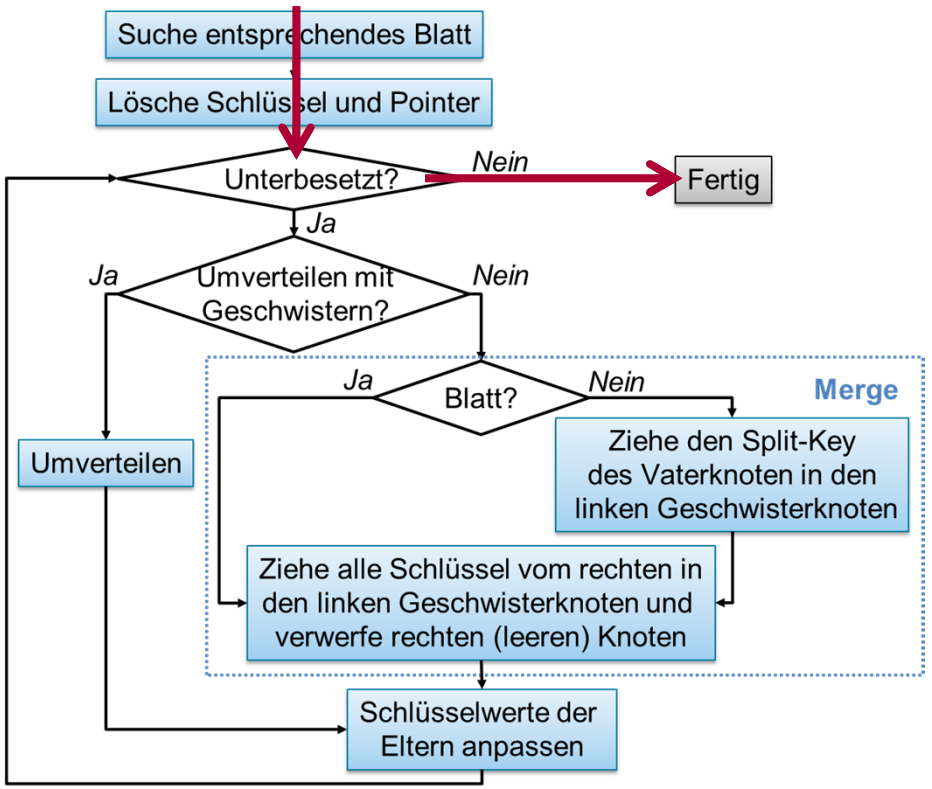


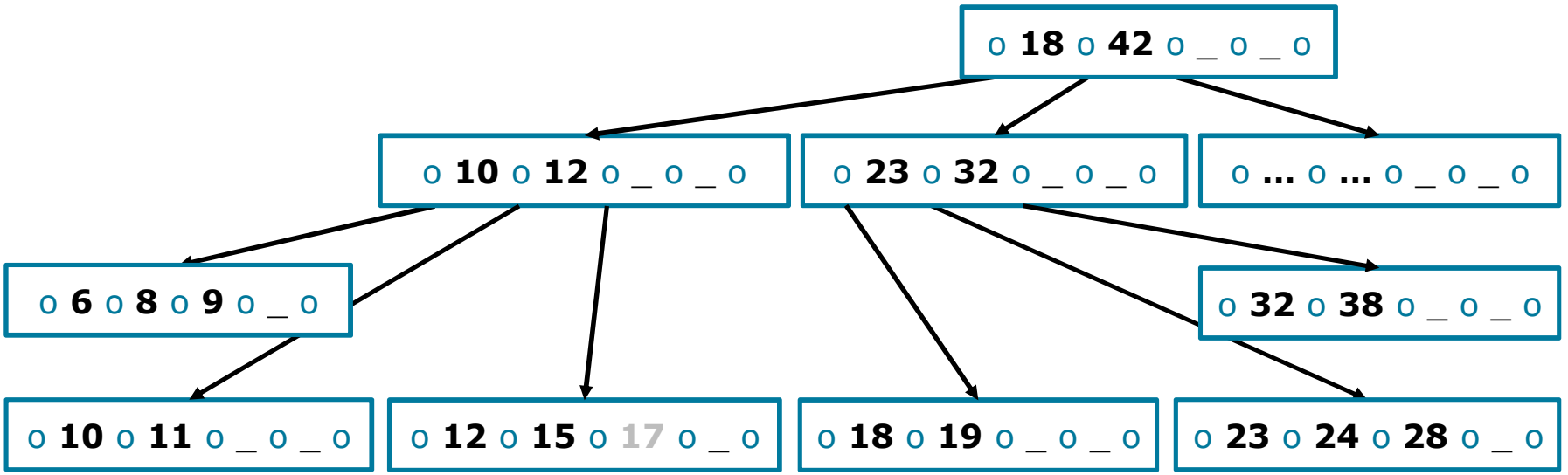
B⁺-Bäume: DELETE



B+-Bäume: DELETE

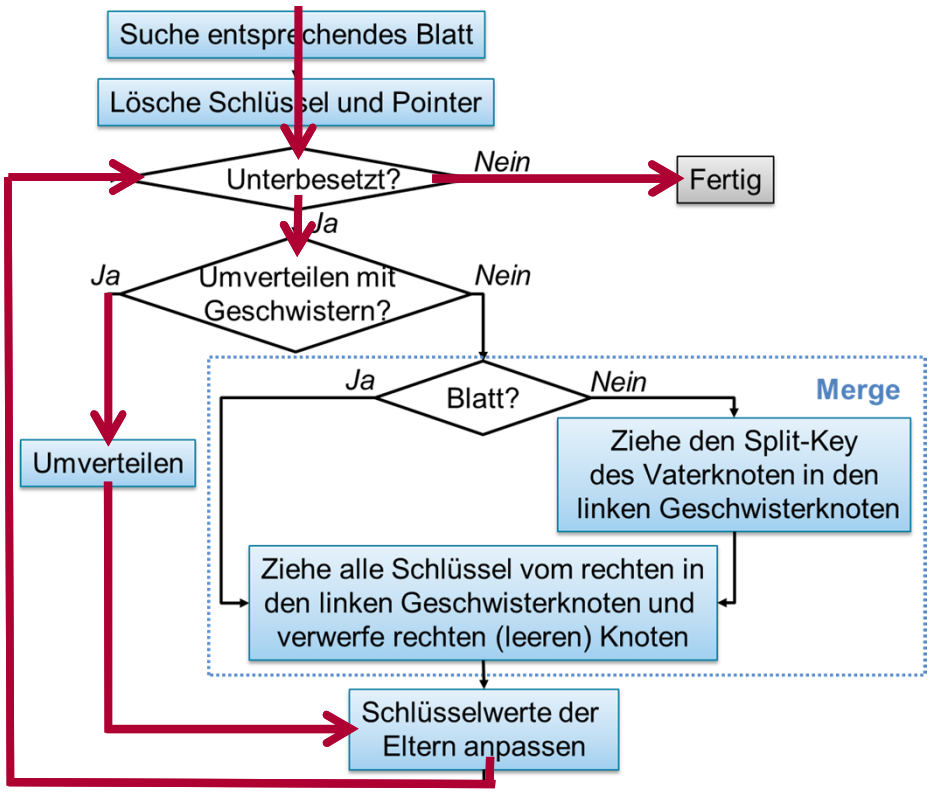
- DELETE(17)



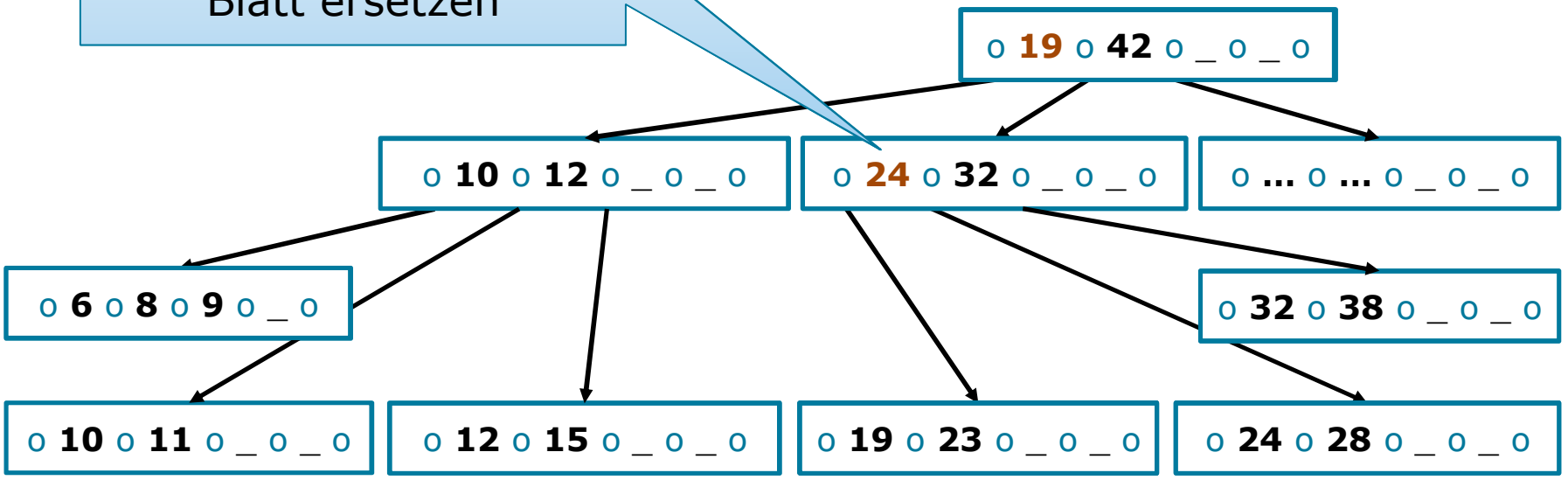


B+-Bäume: DELETE

- DELETE(17)
- DELETE(18)

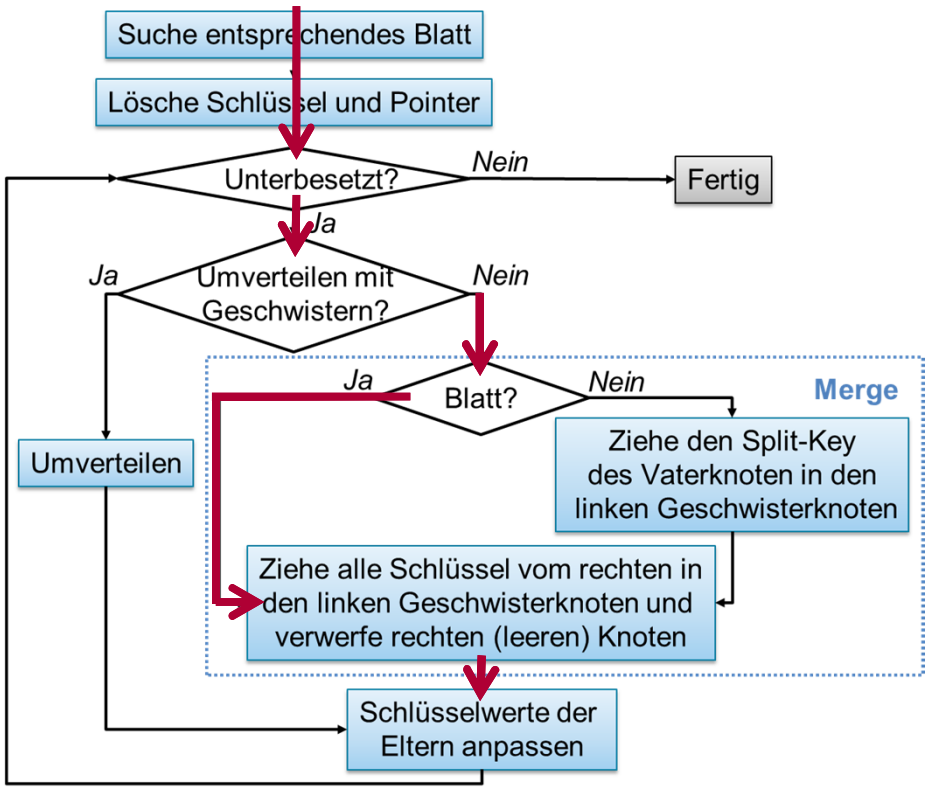


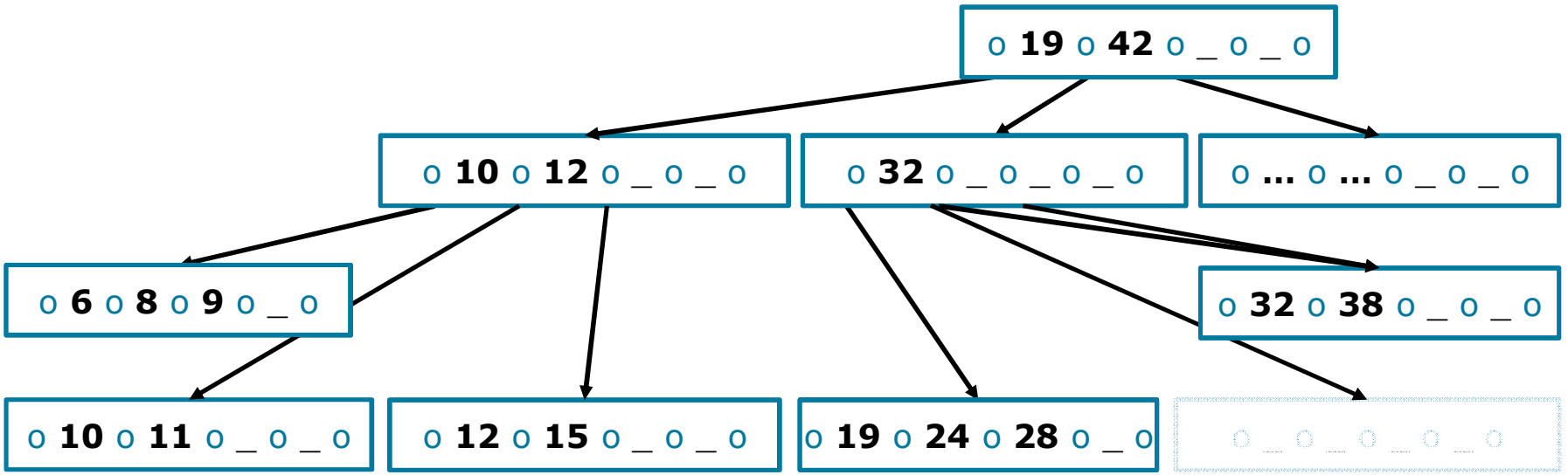
Ursprünglichen Schlüssel durch neuen kleinsten Schlüssel im rechten Blatt ersetzen



B+-Bäume: DELETE

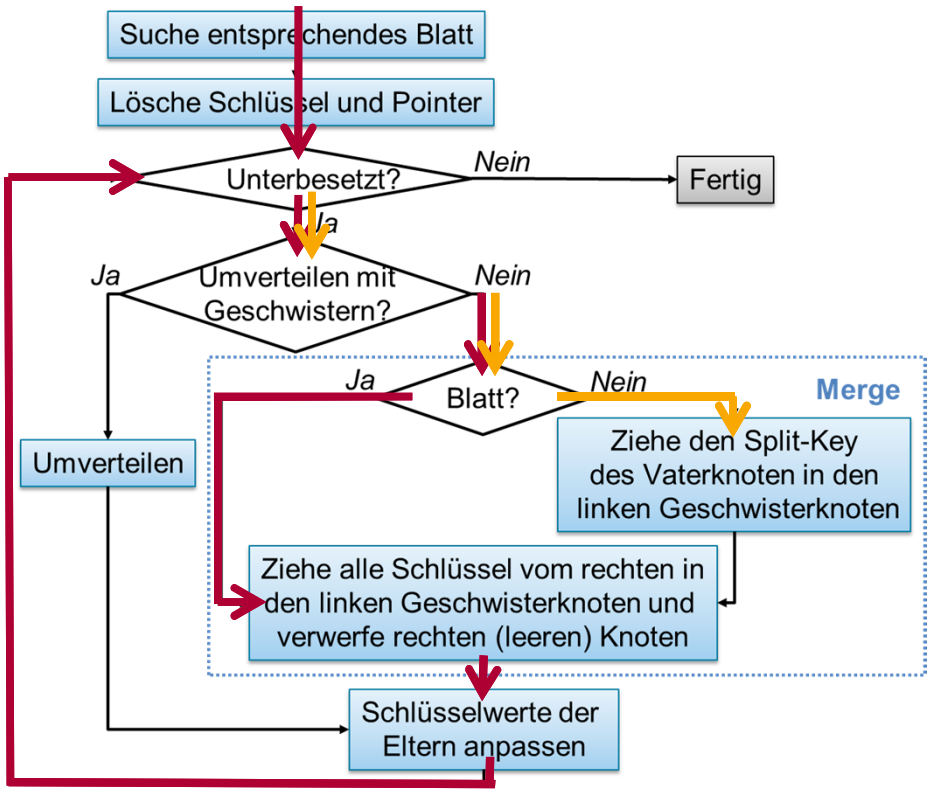
- DELETE(17)
- DELETE(18)
- DELETE(23)

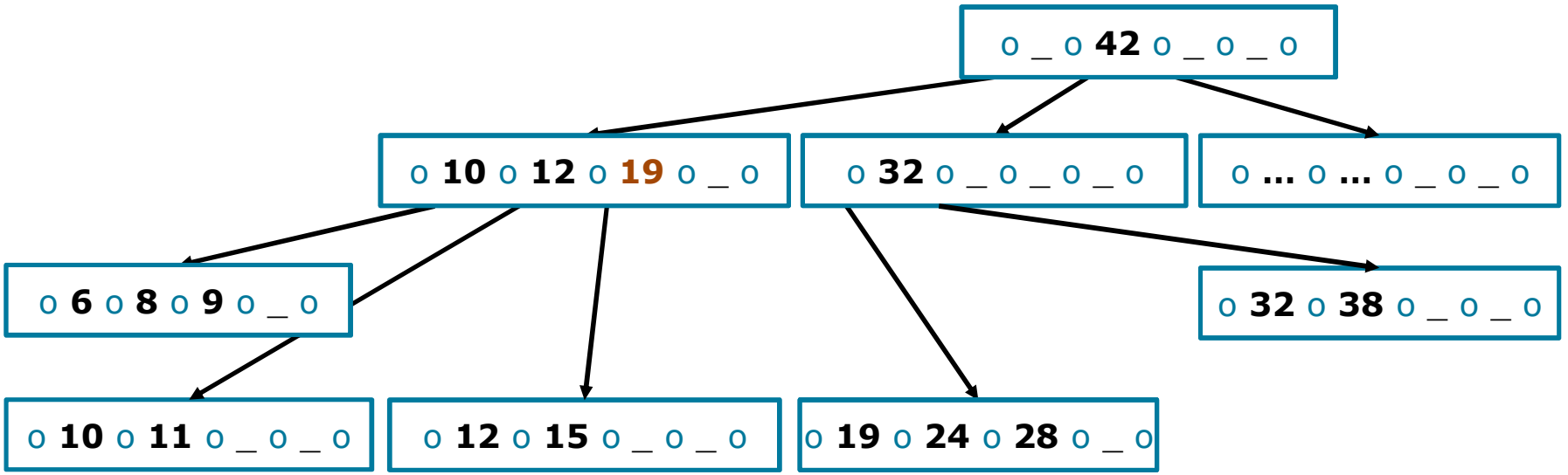




B+-Bäume: DELETE

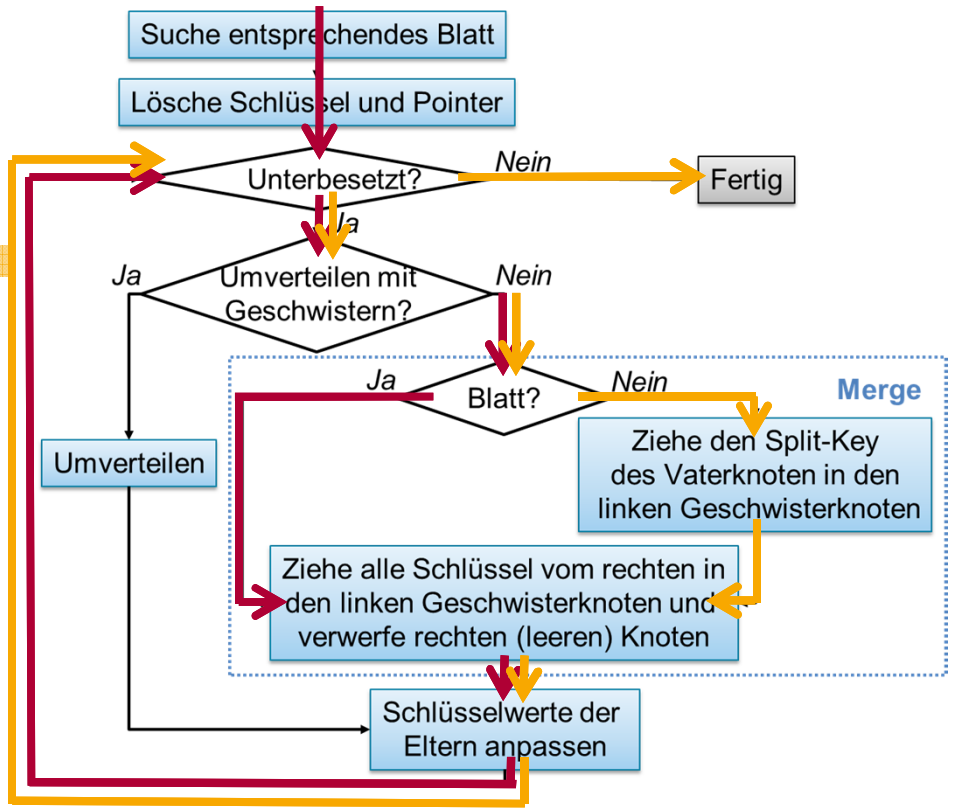
- DELETE(17)
- DELETE(18)
- DELETE(23)

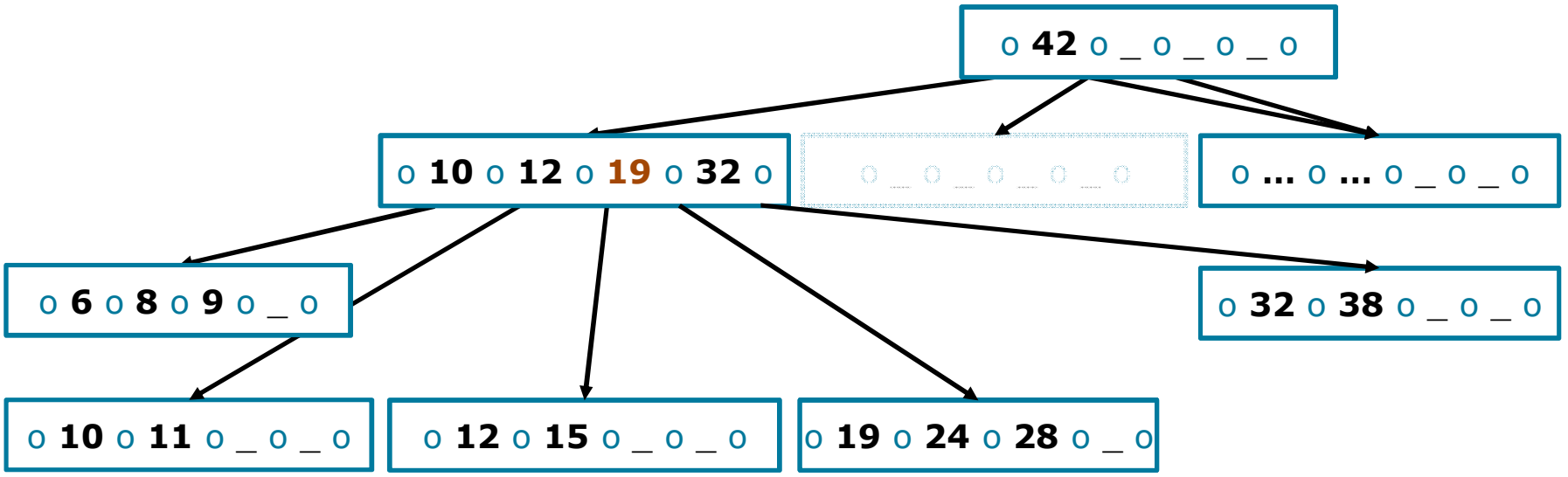




B+-Bäume: DELETE

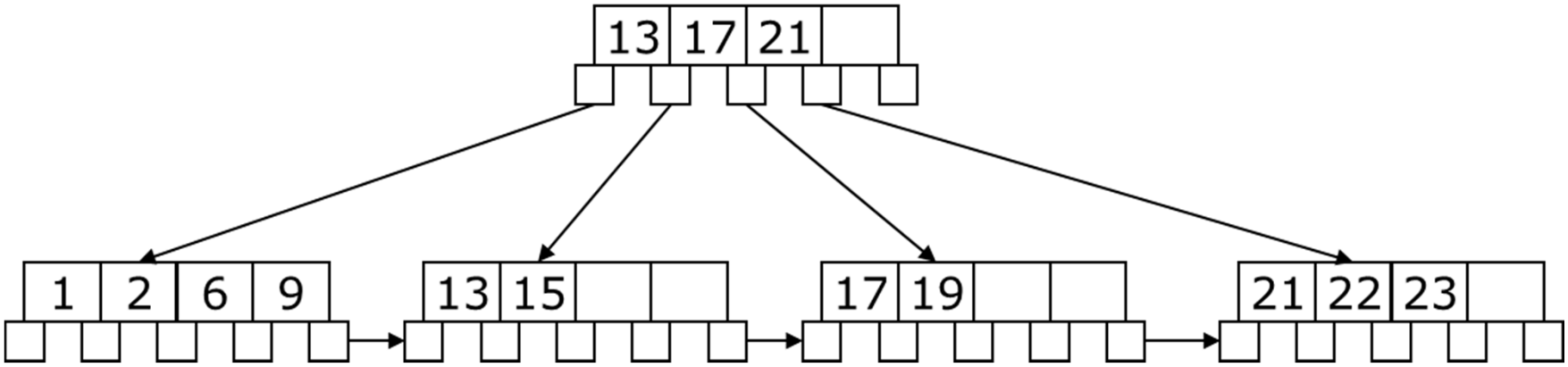
- DELETE(17)
- DELETE(18)
- DELETE(23)





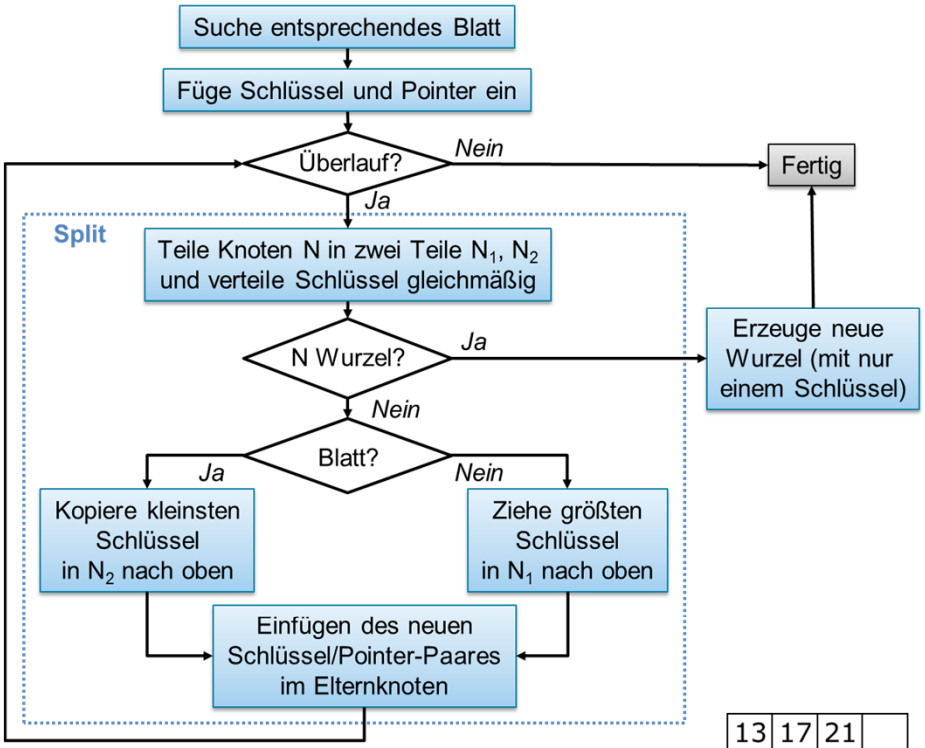
Übungsaufgabe

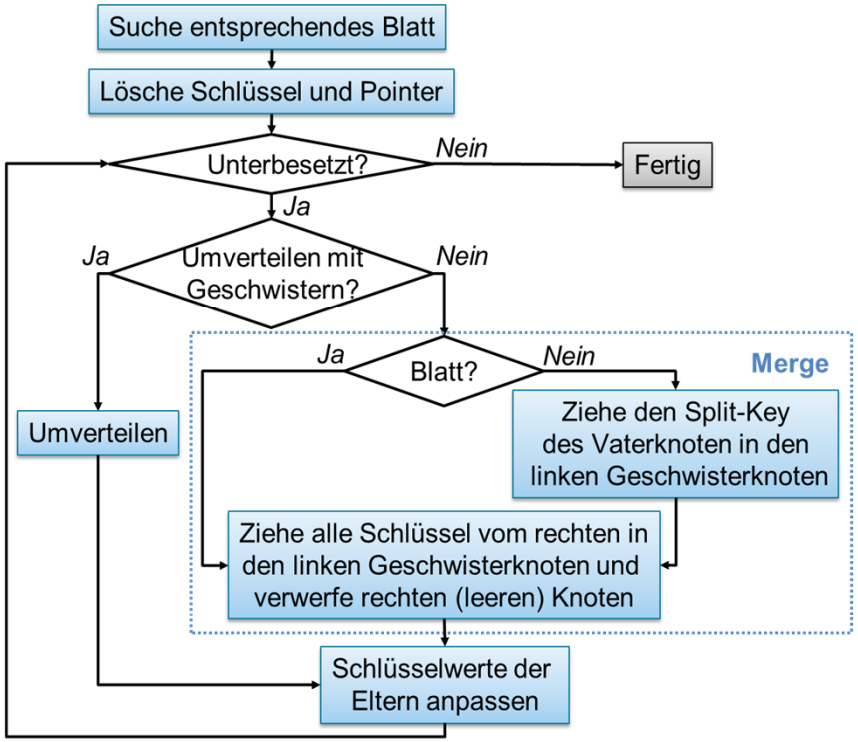
- Gegeben: B+-Baum, entstanden durch Einfügen von
  9, 22, 19, 21, 13, 17, 1, 6, 23, 15

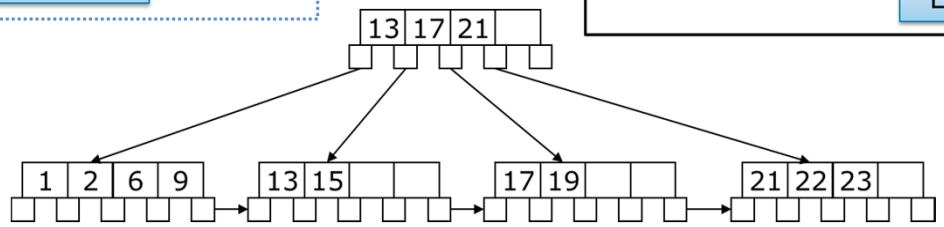


1. Füge nacheinander ein: 14, 29, 4, 24
2. Lösche nacheinander: 6, 13, 15

B+-Bäume: Übungsaufgabe – Übersicht



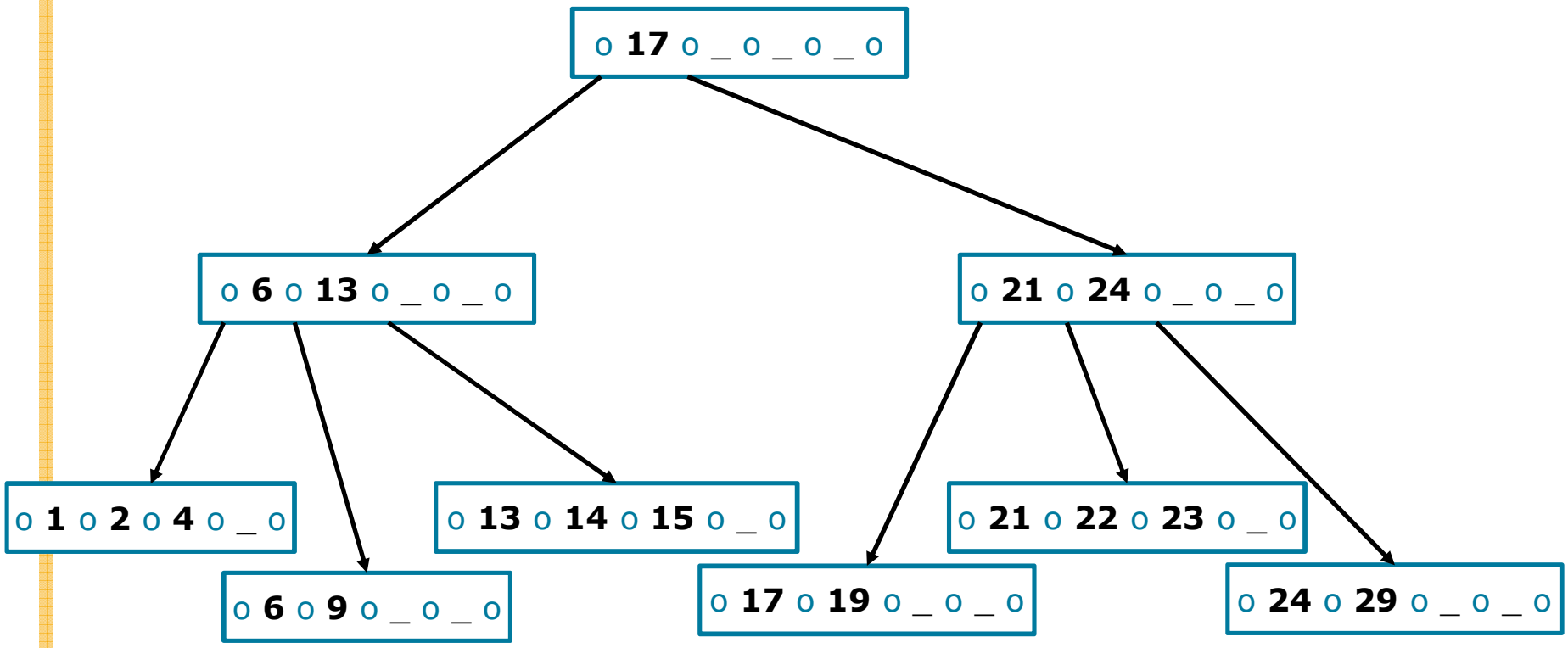




1. Füge nacheinander ein: 14, 29, 4, 24
2. Lösche danach nacheinander: 6, 13, 15

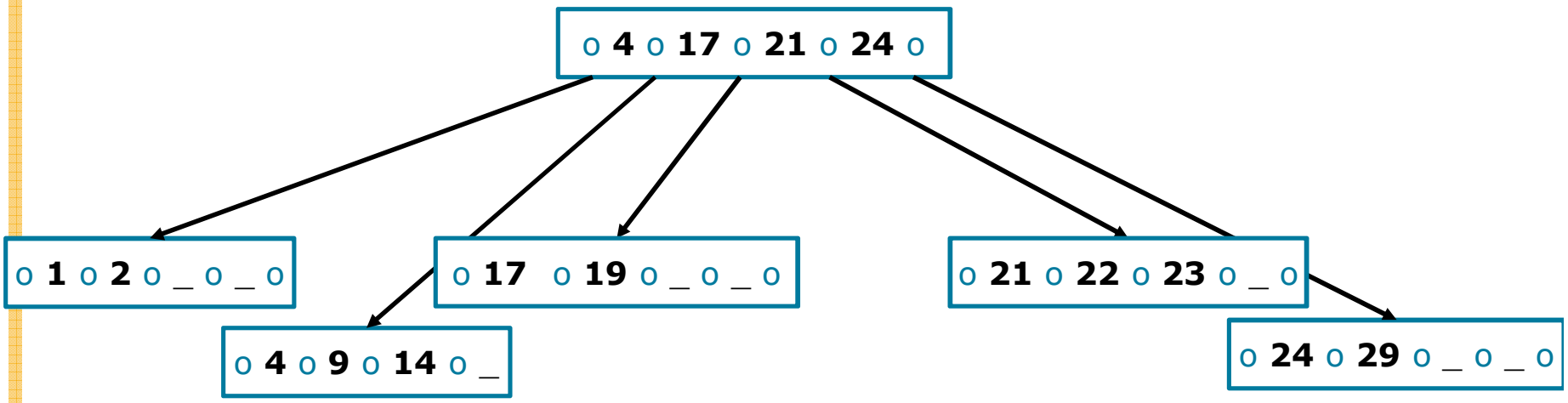
B+-Bäume:

Übungsaufgabe – Lösung 1



1. Füge nacheinander ein: 14, 29, 4, 24
2. Lösche danach nacheinander: 6, 13, 15

Übungsaufgabe – Lösung 2



1. Füge nacheinander ein: 14, 29, 4, 24
2. Lösche danach nacheinander: 6, 13, 15