

Kap. 4.3: AVL-Bäume
Kap. 4.4: B-Bäume

Professor Dr. Petra Mutzel

Lehrstuhl für Algorithm Engineering, LS11

12. VO

16. Mai 2006

Motivation

„Warum soll ich heute hier bleiben?“

Müssen Sie nicht: DEMO

„Was gibt es heute Besonderes?“

Schönes Java-Applet!

Überblick

- Wiederholung AVL-Bäume

- Java-Applet AVL-Bäume

- Einführung von B-Bäumen

- Eigenschaften von B-Bäumen

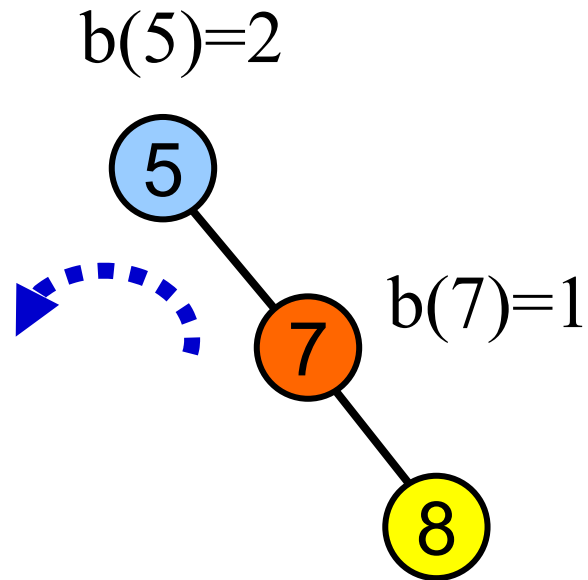
Definitionen für AVL-Bäume

- **Höhe eines Knotens:** Länge eines längsten Pfades von v zu einem Nachkommen von v
- **Höhe eines Baumes:** Höhe seiner Wurzel
- **Höhe eines leeren Baumes:** -1
- **Balance eines Knotens:** $\text{bal}(v) = h_2 - h_1$, wobei h_1 und h_2 die Höhe des linken bzw. rechten Unterbaumes von v
- **v heißt balanciert:** wenn $\text{bal}(v) \in \{-1, 0, +1\}$, sonst heißt v unbalanciert
- **AVL-Baum:** Binärer Suchbaum, bei dem alle Knoten balanciert sind

Implementierung der Operationen Insert und Delete

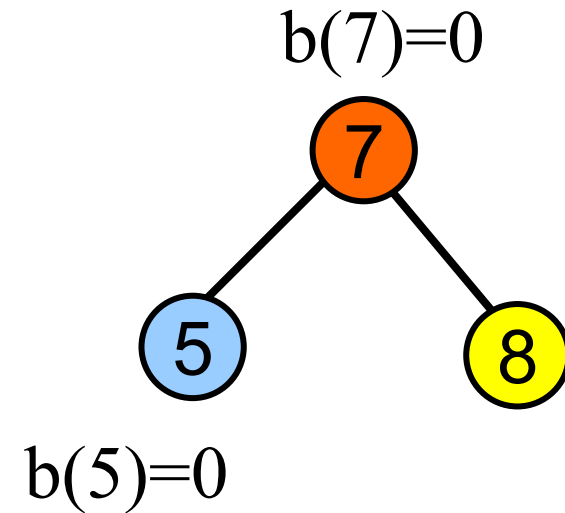
- **Idee:**
- zunächst wie bei den natürlichen binären Suchbäumen.
- Falls Baum nicht mehr balanciert ist, dann wissen wir, dass ein Knoten u auf dem Suchpfad existiert mit $\text{bal}(u) \in \{-2, +2\}$
- Wir rebalancieren den Baum nun an dieser Stelle, so dass er danach wieder balanciert ist.

Rotationen (1)



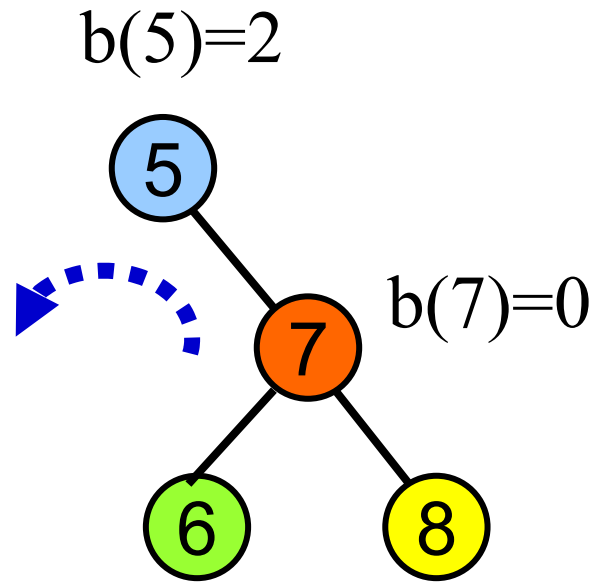
5 ist unbalanciert

Rotation
nach
links
an 5



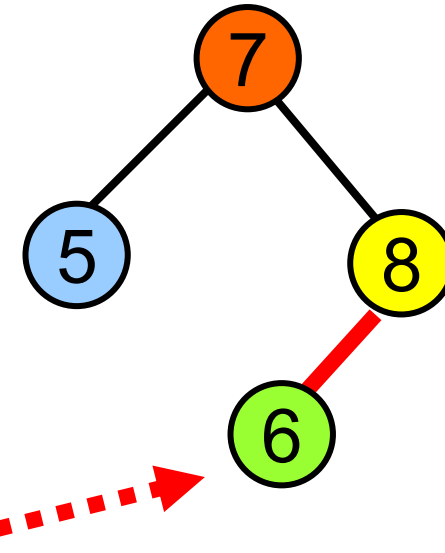
5, 7, 8 sind balanciert

Rotationen (2)



5 ist unbalanciert

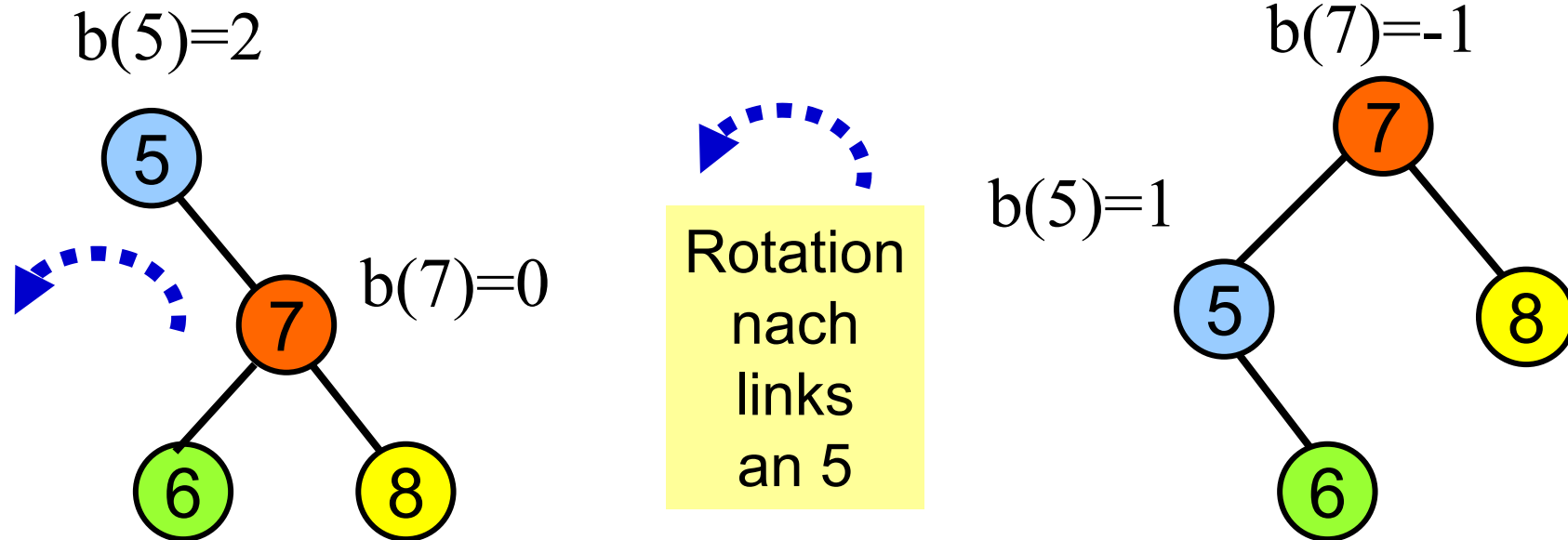
Rotation
nach
links
an 5



falsch: die 6 muss mitwandern

5, 7, 8 sind balanciert

Rotationen (3)



so korrekt! Warum eigentlich?

5 ist unbalanciert

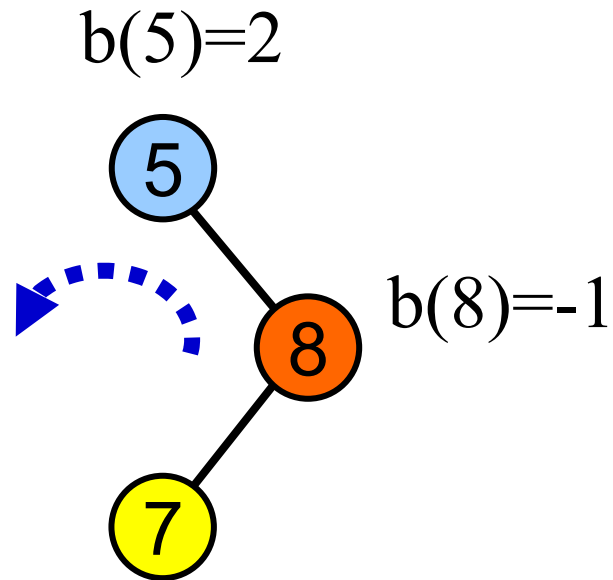
5, 7, 8 sind balanciert

Die Suchbaumeigenschaft bleibt bei einer Rotation erhalten.

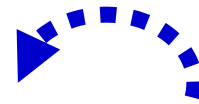
Nach einer Rotation entsteht wieder ein binärer Suchbaum

Achtung: dies ist nicht so wenn gleiche Schlüssel enthalten sind

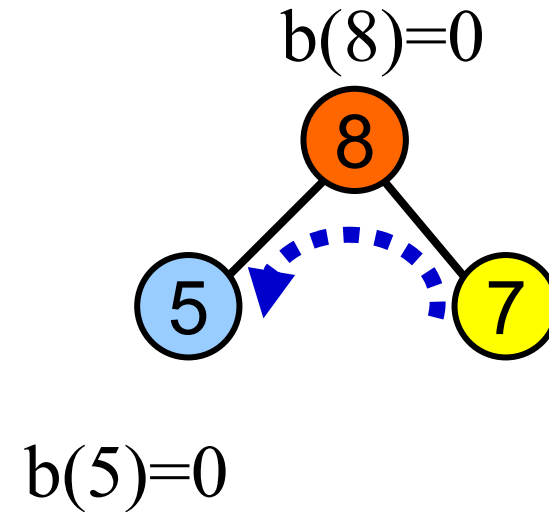
Rotationen (4)



5 ist unbalanciert

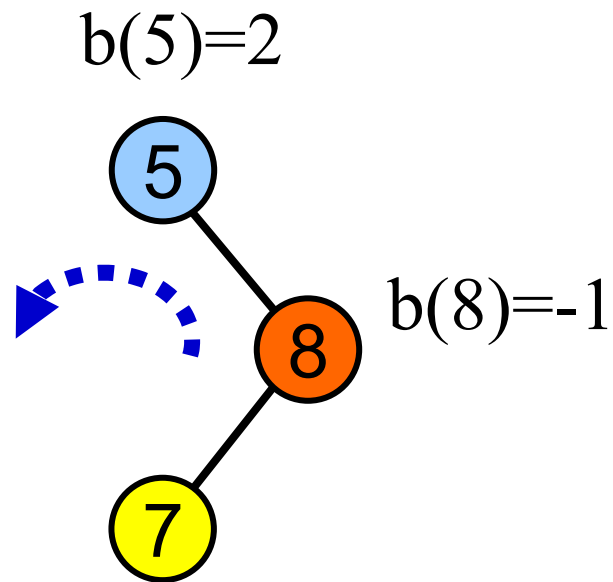


Rotation
nach
links
an 5

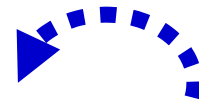


falsch: Das war keine
Rotation!!!

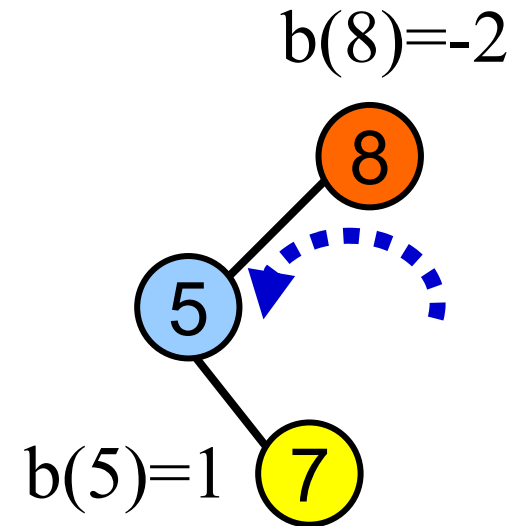
Rotationen (5)



5 ist unbalanciert

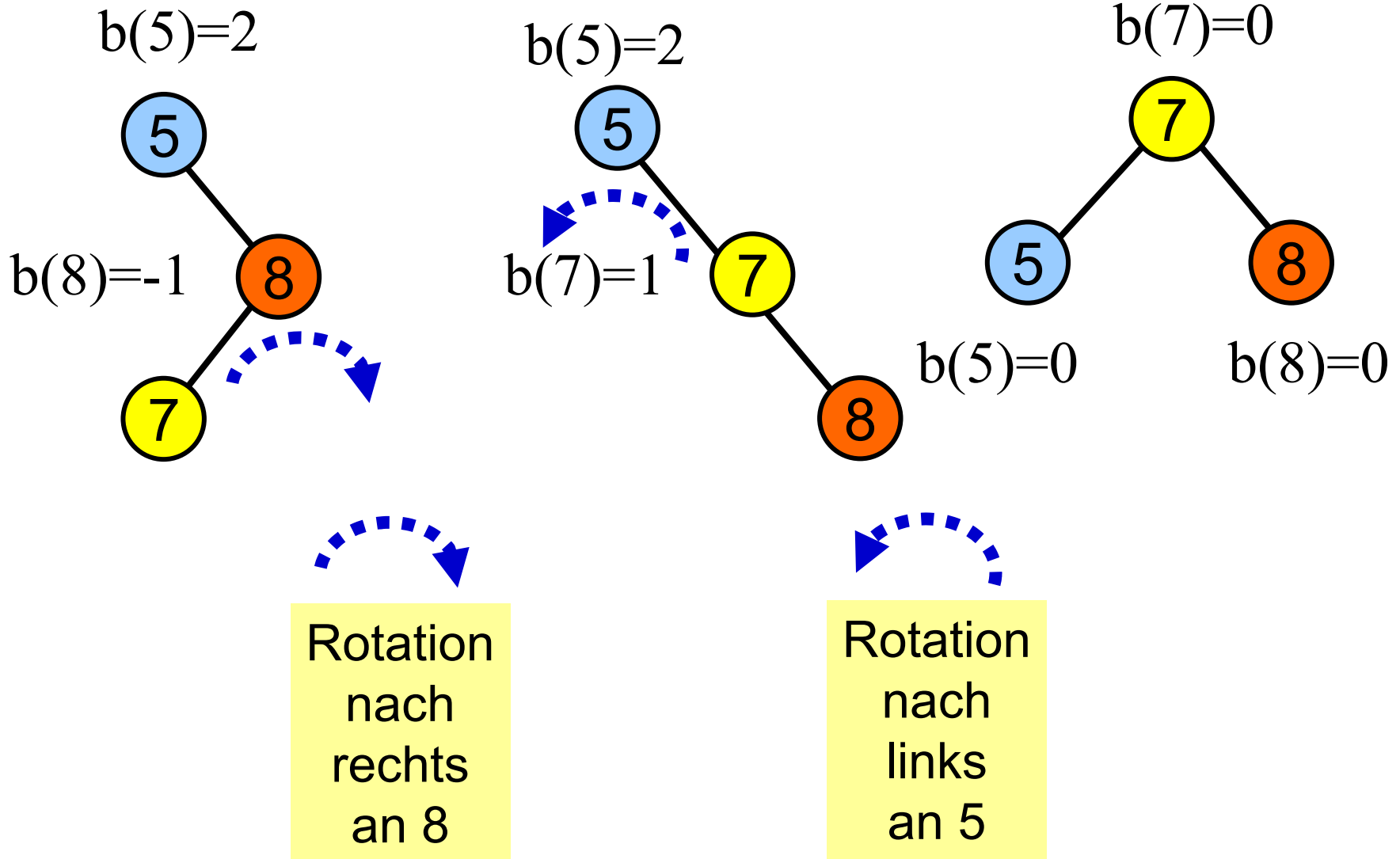


Rotation
nach
links
an 5



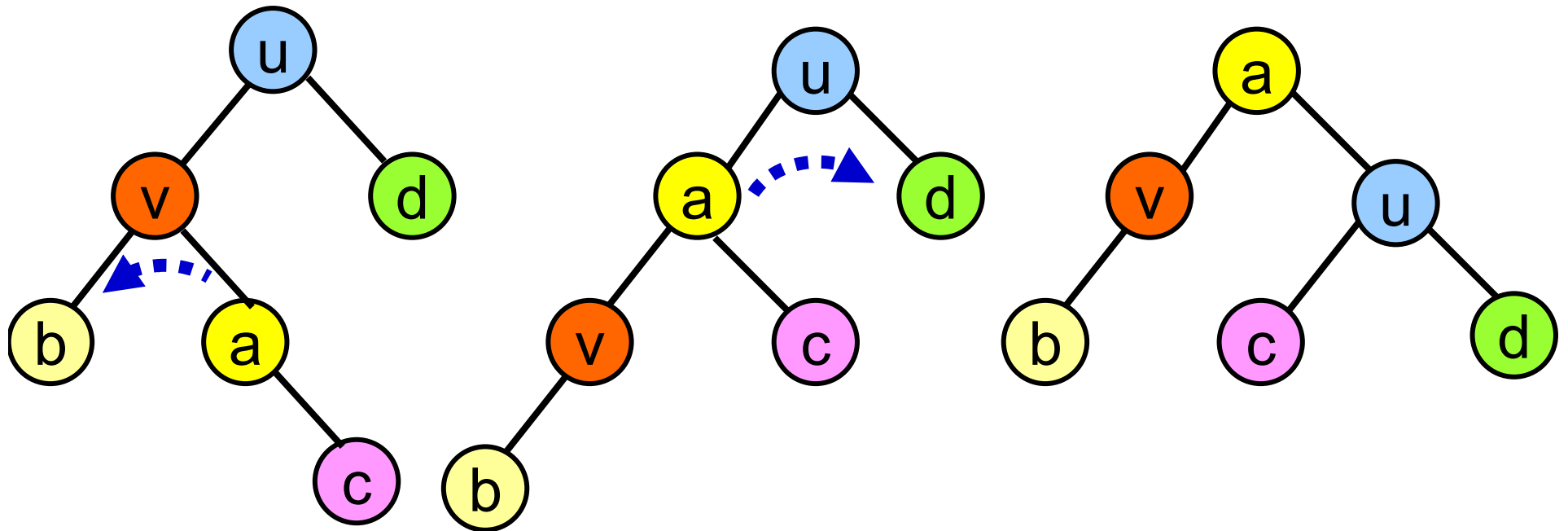
Diese Rotation hat
nichts genützt!

Rotationen (6)



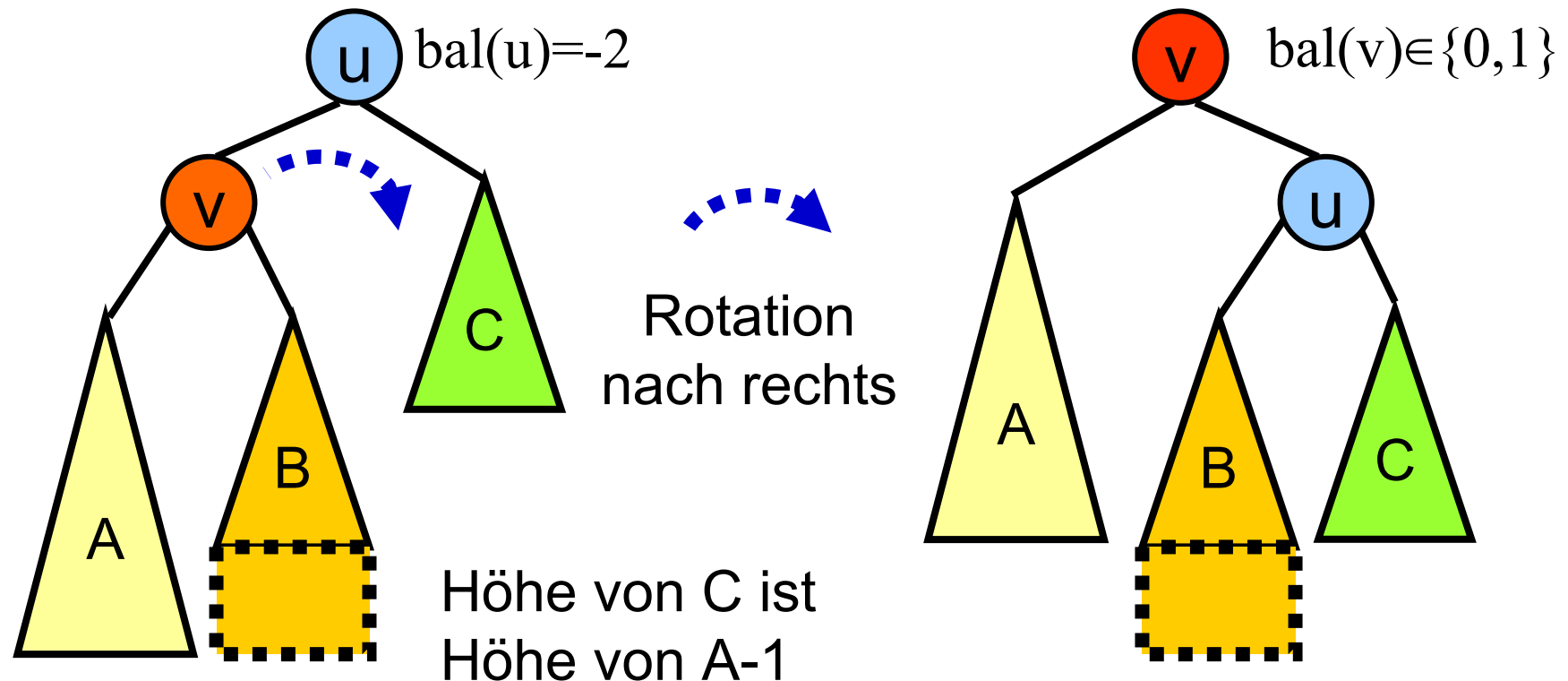
Doppelrotation Rechts-Links notwendig!

Doppelrotationen



1. Fall: $\text{bal}(u) = -2$

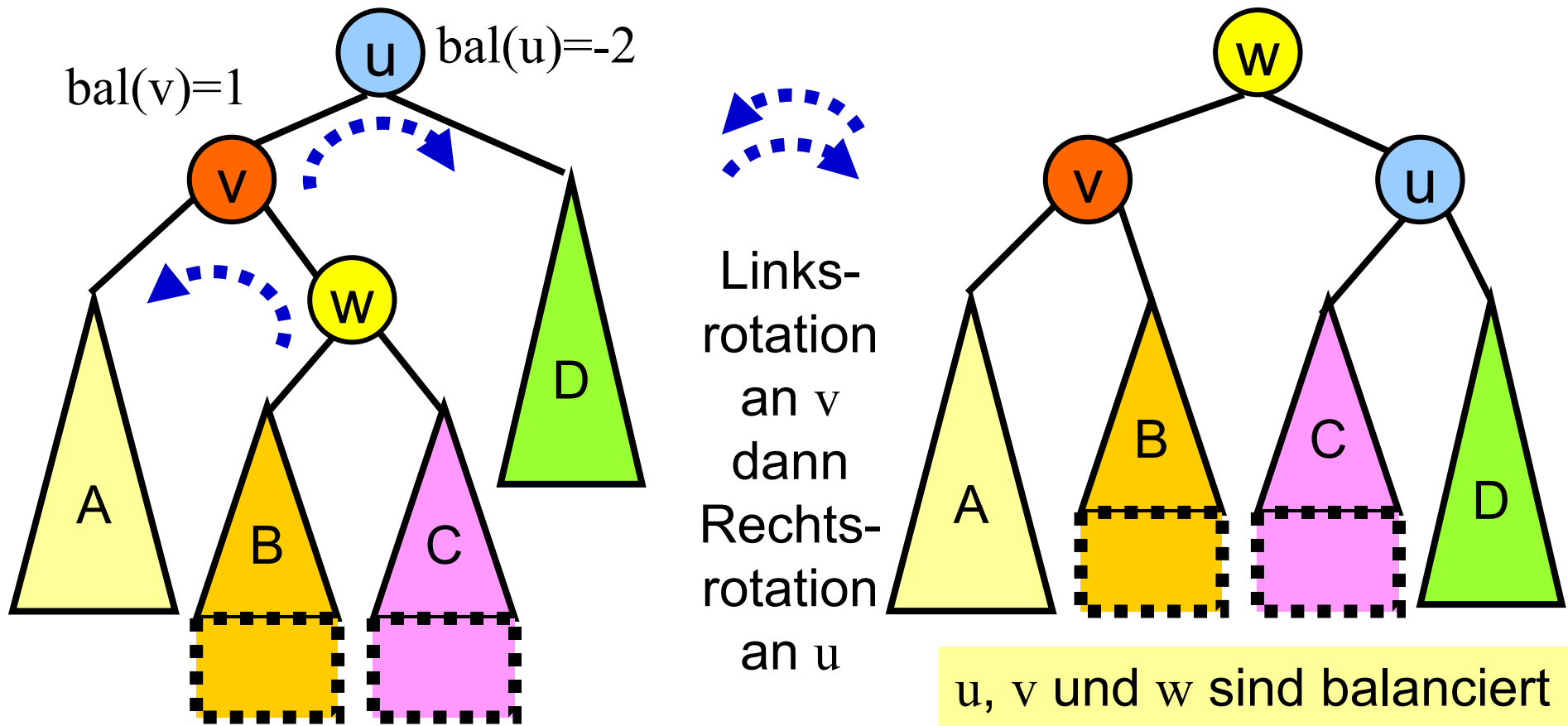
- Sei v das linke Kind von u (existiert!)
- **Fall 1.1: $\text{bal}(v) \in \{-1, 0\}$:**



- Suchbaumeigenschaft bleibt erhalten; u und v sind balanciert
- Für die Knoten unterhalb hat sich nichts geändert.

1. Fall: $\text{bal}(u) = -2$ ff.

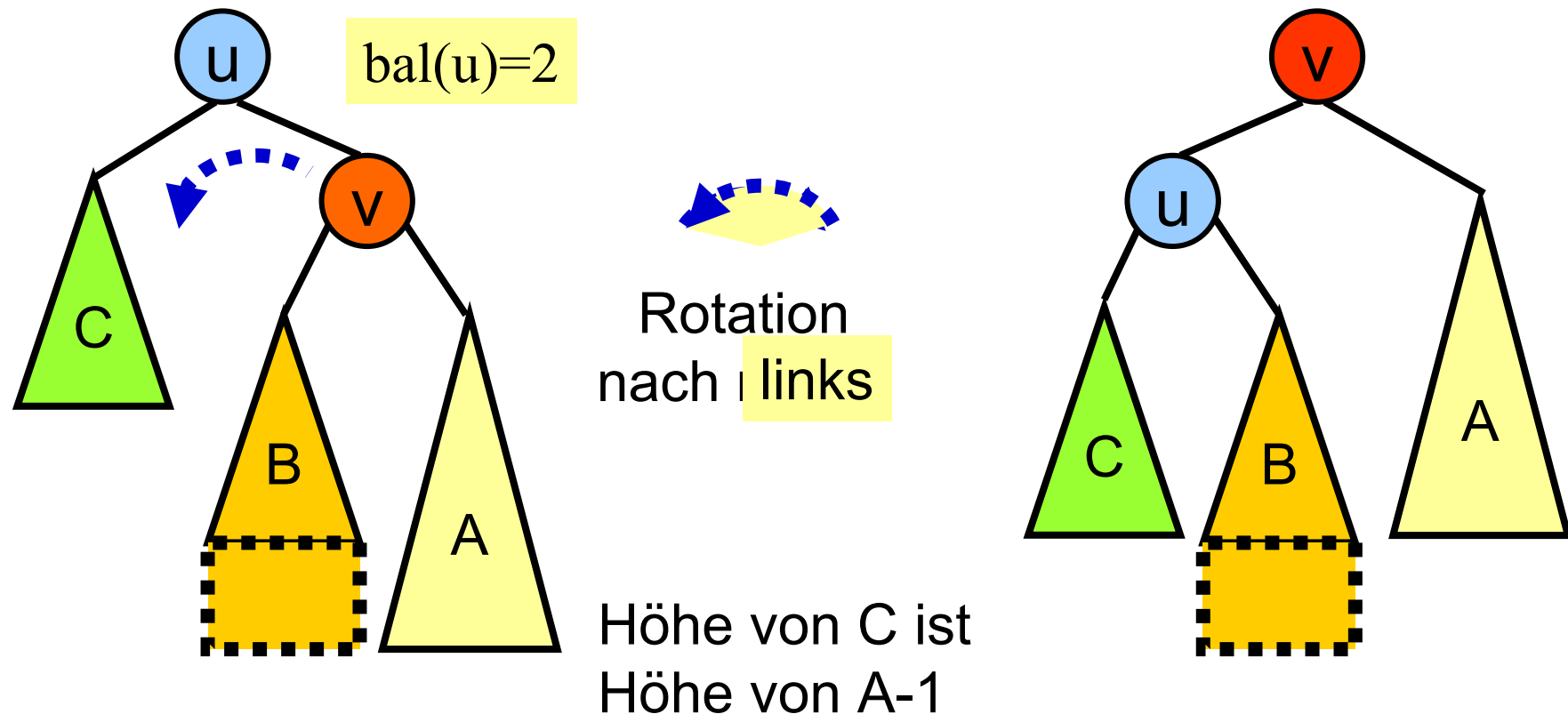
- Sei v das linke Kind von u (existiert!)
- **Fall 1.2: $\text{bal}(v) = +1$:**



Höhe von (B oder C) und von D ist Höhe von A

2. Fall: $\text{bal}(u)=+2$

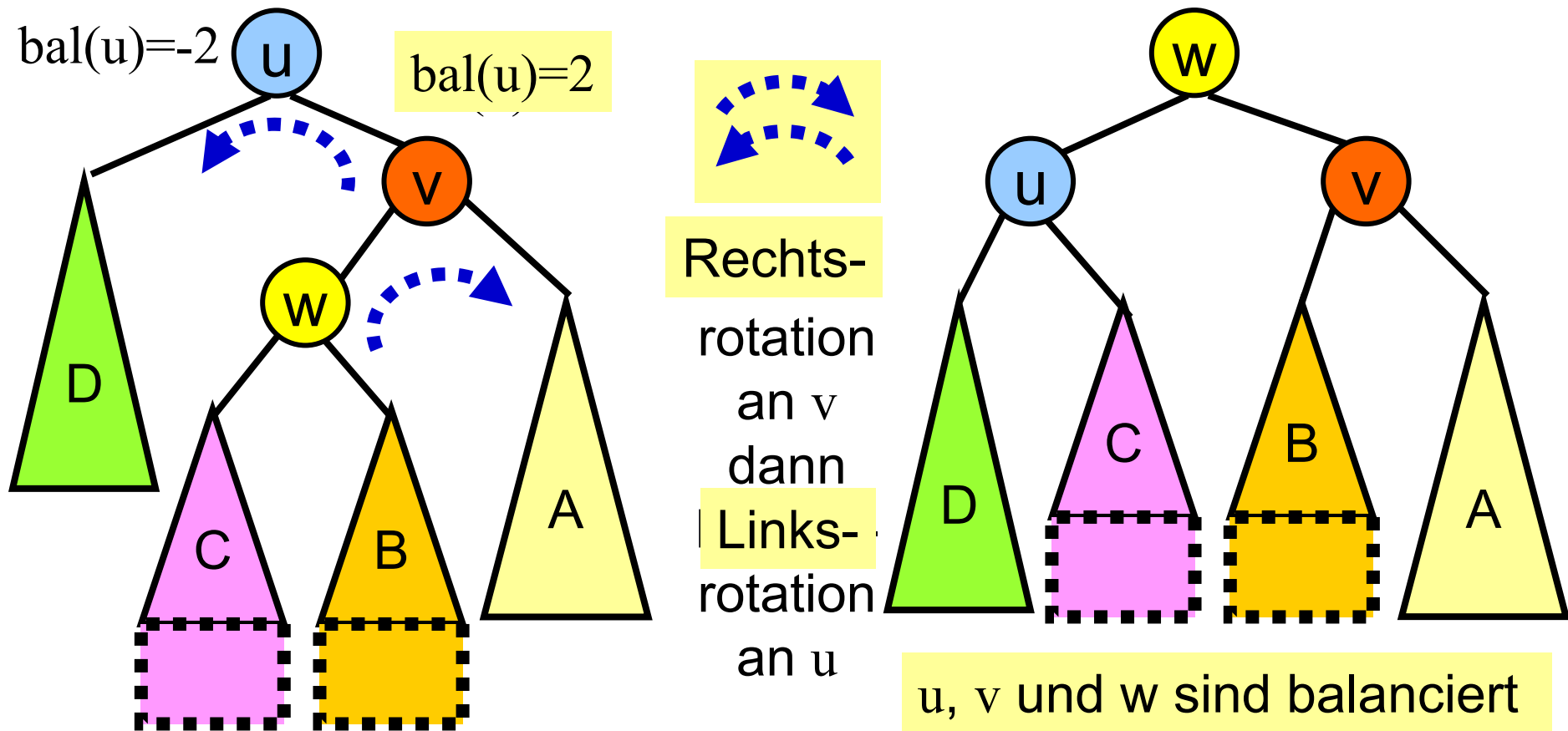
- Sei v das rechte Kind von u (existiert!)
- **Fall 1.1:** $\text{bal}(v) \in \{0, 1\}$: Rotation nach rechts an u



- Inorder-Reihenfolge bleibt erhalten und u und v sind balanciert
- Für die Knoten unterhalb hat sich nichts geändert.

2. Fall: $\text{bal}(u) = +2$ ff.

- Sei v das rechte Kind von u (existiert!)
- **Fall 1.2:** $\text{bal}(v) = -1$; Rechts-Links rotation an u



Höhe von (B oder C) und von D ist Höhe von A

Rebalancierung ff.

- Bestimme nun den nächsten unbalancierten Knoten maximaler Tiefe. Diese Tiefe ist nun kleiner als vorher.
- Wiederhole die Rebalancierung (AVL-Ersetzung) solange, bis alle Knoten balanciert sind.

- Das Verfahren konvergiert nach $O(h(T))$ Iterationen zu einem gültigen AVL-Baum.

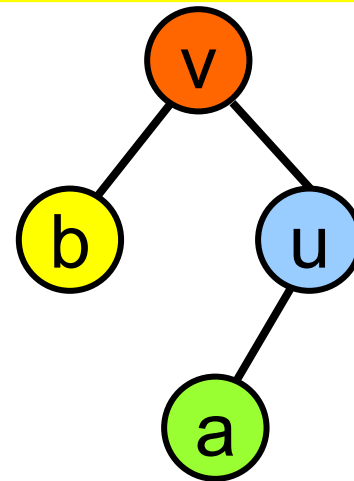
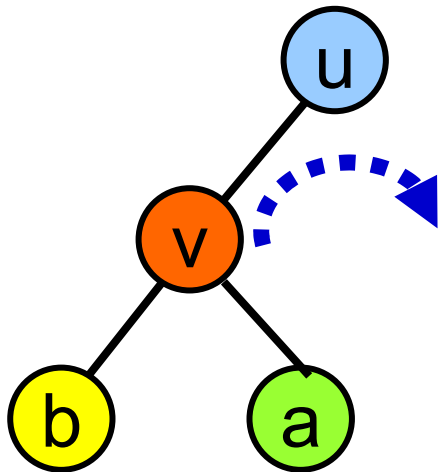
Rebalancierung ff.

- Die Insert-Operation unterscheidet sich nur insofern von der Delete-Operation, dass hier ein einziger Rebalancierungsschritt genügt.
-
- Bei der Delete-Operation hingegen kann es sein, dass mehrere Rebalancierungsschritte notwendig sind.

Implementierungen:

RotateRight(u)

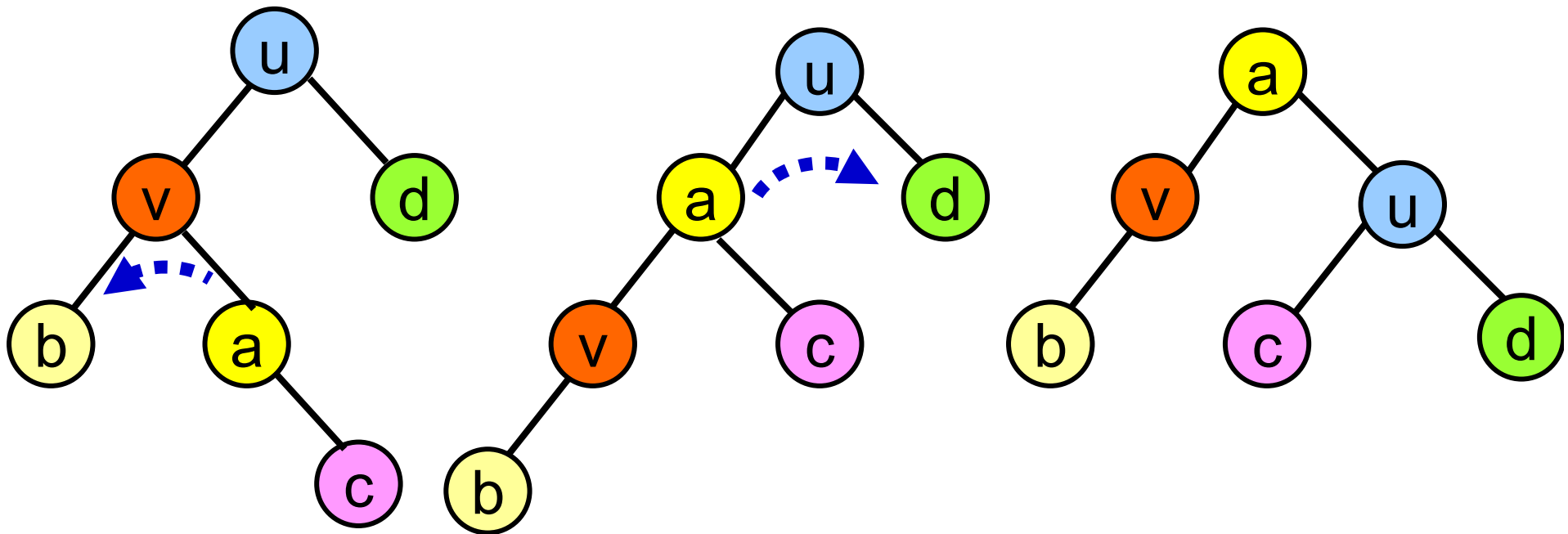
- (1) $v := u.\text{left}$ //Bestimme v
- (2) $u.\text{left} := v.\text{right}$
- (3) $v.\text{right} := u$
- (4) $u.\text{height} := \max(h(u.\text{left}), h(u.\text{right})) + 1$
- (5) $v.\text{height} := \max(h(v.\text{left}), h(u)) + 1$
- (6) $\text{return}(v)$ //neue Wurzel



RotateLeftRight(u)

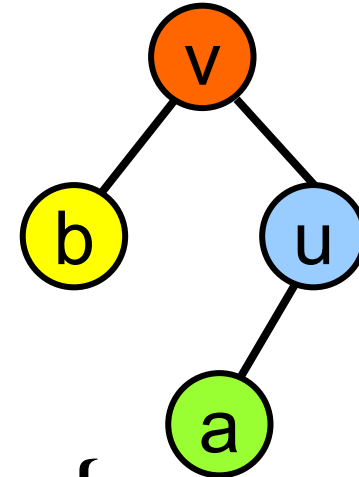
- (1) `u.left:=RotateToLeft(u.leftson)`
- (2) `return(RotateToRight(u))`

Beispiel:



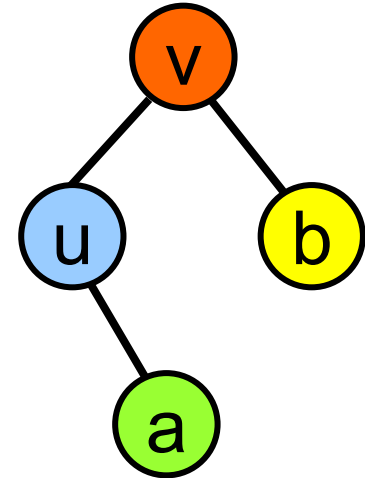
INSERTAVL(p,q)

- (1) **If** $p == \text{NULL}$ **then** INSERT(p,q)
- (2) **else**
- (3) **If** $q.\text{key} < p.\text{key}$ **then** {
- (4) INSERTAVL(p.left,q)
- (5) **if** $h(p.\text{right}) - h(p.\text{left}) == -2$ **then** {
- (6) **if** $h(p.\text{left}.\text{left}) > h(p.\text{left}.\text{right})$ **then**
- (7) $p = \text{RotateToRight}(p)$
- (8) **else** $p = \text{DoubleRotateLeftRight}(p)$
- (9) } }



INSERTAVL(p,q)

- (1) Else {
- (2) If $q.key > p.key$ then
- (3) INSERTAVL(p.right,q)
- (4) if $h(p.right) - h(p.left) == 2$ then {
- (5) if $h(p.right.right) > h(p.right.left)$ then
- (6) $p = \text{RotateToLeft}(p)$
- (7) else $p = \text{DoubleRotateRightLeft}(p)$
- (8) } }
- (9) $p.height := \max(h(p.left), h(p.right)) + 1$



Analyse

- Rotationen und Doppelrotationen können in konstanter Zeit ausgeführt werden.
- Eine Rebalancierung wird höchstens einmal an jedem Knoten auf dem Pfad vom eingefügten oder gelöschten Knoten zur Wurzel durchgeführt.

- Insert und Delete-Operationen sind in einem AVL-Baum in Zeit $O(\log n)$ möglich.

- AVL-Bäume unterstützen die Operationen Suchen, Insert, Delete, Minimum, Maximum, Successor, Predecessor, in Zeit $O(\log n)$.

Java-Applets

- Es gibt zu AVL-Bäumen sehr schöne Java-Applets, die die Doppelrotationen sehr schön darstellen, z.B. die Studiosseiten von Math^e(Prism)^a:
- <http://www.matheprisma.uni-wuppertal.de/Module/BinSuch/index.html>



Kap. 4.4: B-Bäume

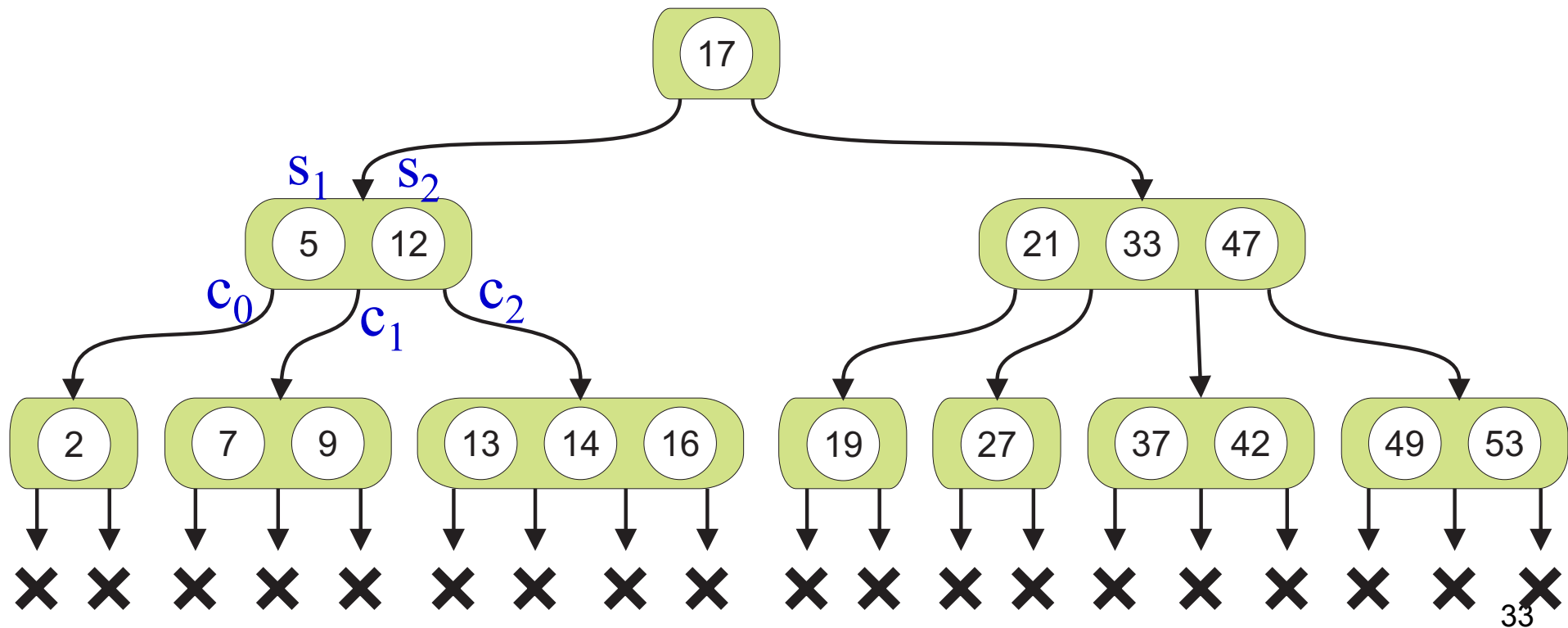
B-Bäume

- Einführung von Rudolf Bayer und Eduard M. McCreight 1972
- Datenstruktur zur Verwaltung von Indizes für das relationale Datenmodell von Edgar F. Codd 1972
- → Entwicklung des ersten SQL-Datenbanksystems System R bei IBM

- B-Bäume sind ausgeglichene Mehrwegbäume
- **Idee:** jeder Knoten eines B-Baums der Ordnung m besitzt zwischen $\lceil m/2 \rceil$ und m Kinder.

B-Bäume

- Ein Knoten mit $k+1$ Zeigern c_0, c_1, \dots, c_k auf die Unterbäume besitzt k Schlüssel s_1, \dots, s_k
- Diese sind in sortierter Reihenfolge gespeichert und trennen die jeweiligen Unterbäume.



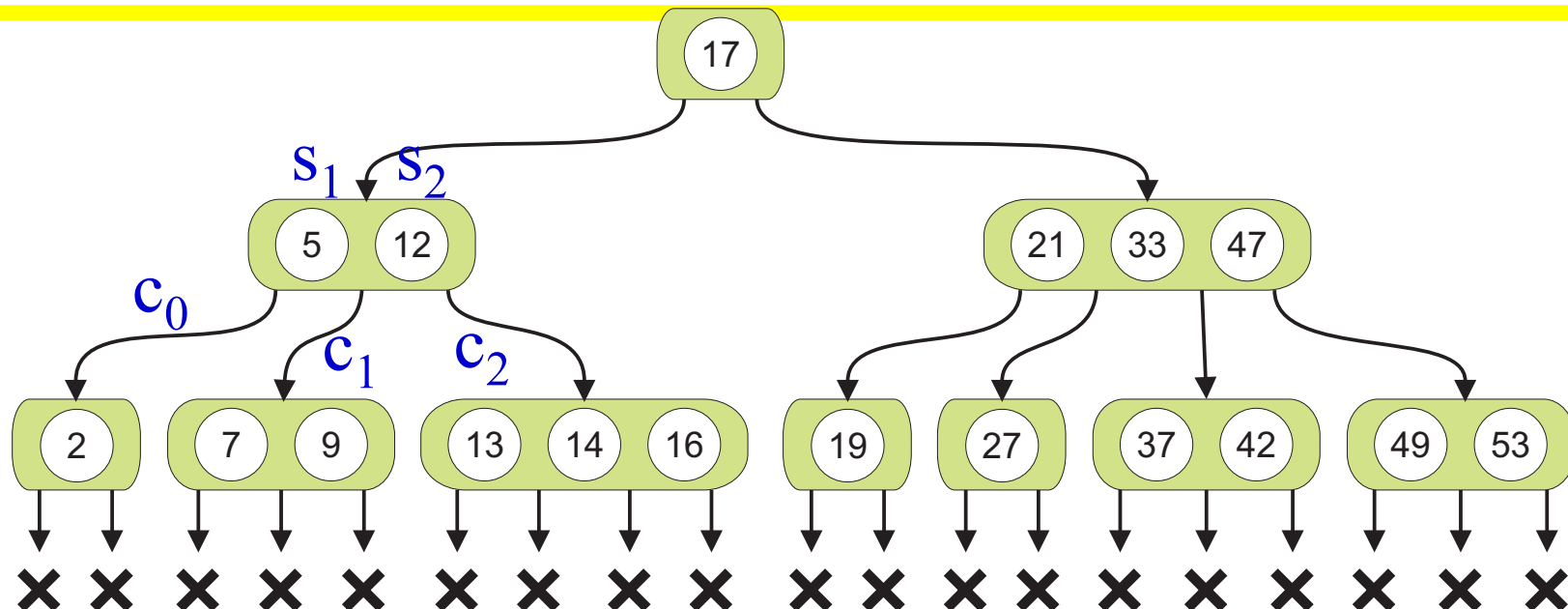
Definition B-Bäume

- Ein **B-Baum der Ordnung $m > 2$** ist ein Baum mit folgenden Eigenschaften (1)-(6):

- (1) Kein Schlüsselwert kommt mehrfach vor
- (2) Für jeden Knoten α mit $k+1$ Zeigern c_0, c_1, \dots, c_k auf Unterbäume gilt:
 - a) α hat k Schlüssel s_1, \dots, s_k , wobei gilt: $s_1 < \dots < s_k$
 - b) Für jeden Schlüssel s im Teilbaum mit Wurzel c_i gilt: $s_i < s < s_{i+1}$, wobei $s_0 = -\infty$ und $s_{k+1} = +\infty$
 - c) Ein Schlüssel s_i wird als Trennschlüssel der Teilbäume mit Wurzel c_{i-1} und c_i bezeichnet.

Definition B-Bäume ff

- (3) Der Baum ist leer, oder die Wurzel hat mindestens 2 Zeiger auf Kinder.
- (4) Jeder Knoten mit Ausnahme der Wurzel enthält mindestens $\lceil m/2 \rceil$ Zeiger.
- (5) Jeder Knoten hat höchstens m Zeiger.
- (6) Alle Blätter haben die gleiche Tiefe.



Existenz von B-Bäumen

Theorem: Für jede beliebige Menge von Schlüsseln und jedes beliebige $m > 2$ existiert ein gültiger B-Baum der Ordnung m .

Beweisidee: Besteht die Schlüsselmenge aus weniger als m Schlüsseln, dann besteht der B-Baum einfach aus einem einzigen Wurzelknoten, der alle Schlüssel enthält.
Sonst: benutze folgendes Lemma.

Größe von B-Bäumen

Lemma: Die Größe eines B-Baums ist linear in n , der Anzahl der Schlüssel.

Beweis:

- Jeder Schlüssel kommt im Baum vor, deshalb: Größe ist in $\Omega(n)$.
- Sei t die Anzahl der Knoten im Baum. Da jeder Knoten mindestens einen Schlüssel enthält, gilt $t = O(n)$.
- Da in jedem Knoten mit k Schlüsseln gilt, dass er $k+1$ Zeiger enthält, haben wir insgesamt $n+t = O(n)$ viele Zeiger.
- Also gezeigt: $\Omega(n)$ und $O(n)$.

Minimale Höhe von B-Bäumen

Lemma: Die minimale Höhe eines B-Baums mit n Schlüsseln ist $\lceil \log_m(n+1) \rceil - 1$

Beweis: Ein B-Baum hat die kleinste Höhe, wenn alle Knoten $m-1$ Schlüssel enthalten.

Bei Höhe h des Baums enthält er dann m^h Blätter.

Ein B-Baum mit $l+1$ Blättern hat l Schlüssel in inneren Knoten.

$$\text{Insgesamt: } n = m^h(m-1) + (m^h-1) = m^{h+1} - 1$$

Schlüssel in Blättern

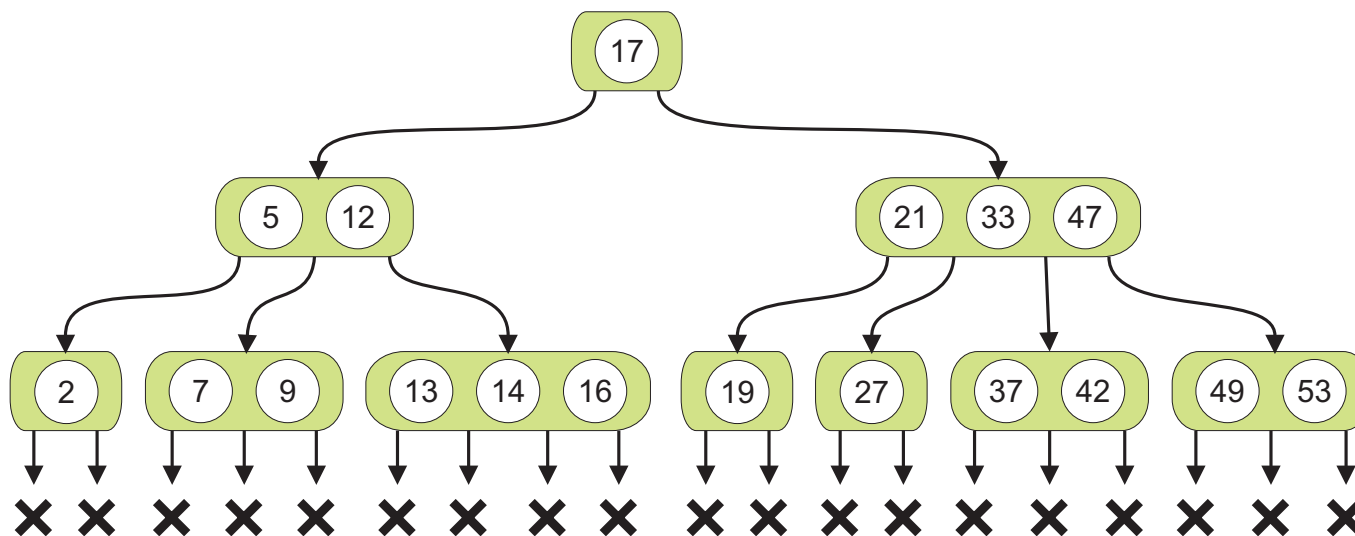
Schlüssel in inneren Knoten

Maximale Höhe von B-Bäumen

Lemma: Die maximale Höhe eines B-Baums mit n Schlüsseln ist $\lfloor \log_{\lceil m/2 \rceil} ((n+1)/2) \rfloor$

Beweis: Ein B-Baum hat die größte Höhe, wenn die Wurzel nur einen einzigen Schlüssel enthält und die beiden Teilbäume und deren Teilbäume (rekursiv) jeweils nur $\lceil m/2 \rceil - 1$ Schlüssel.

Insgesamt bei Höhe h : $n = 1 + 2(\lceil m/2 \rceil^h - 1) = 2 \lceil m/2 \rceil^h - 1$



Datenstruktur eines B-Baums

```
struct BTreeNode  
  var int k           // Anzahl der Schlüssel  
  var KeyType key[1..k]  
  var DataType data[1..k]  
  var BTreeNode child[0..k]  
end struct
```

Interne Repräsentation eines B-Baums:

```
var BTreeNode root
```

Implementierung von $SEARCH(r,s)$ in B-Bäumen

Im Wesentlichen wie im Binärbaum:

1. Vergleiche s mit allen Schlüsseln in der Wurzel (des Teilbaums)
2. Falls gefunden: STOP!
3. Sonst: Folge dem Zeiger, der sich zwischen den beiden im Wurzelknoten benachbarten Schlüsseln befindet. Gehe zu 1.
4. Ausgabe: nicht gefunden!

SEARCH(p,x)

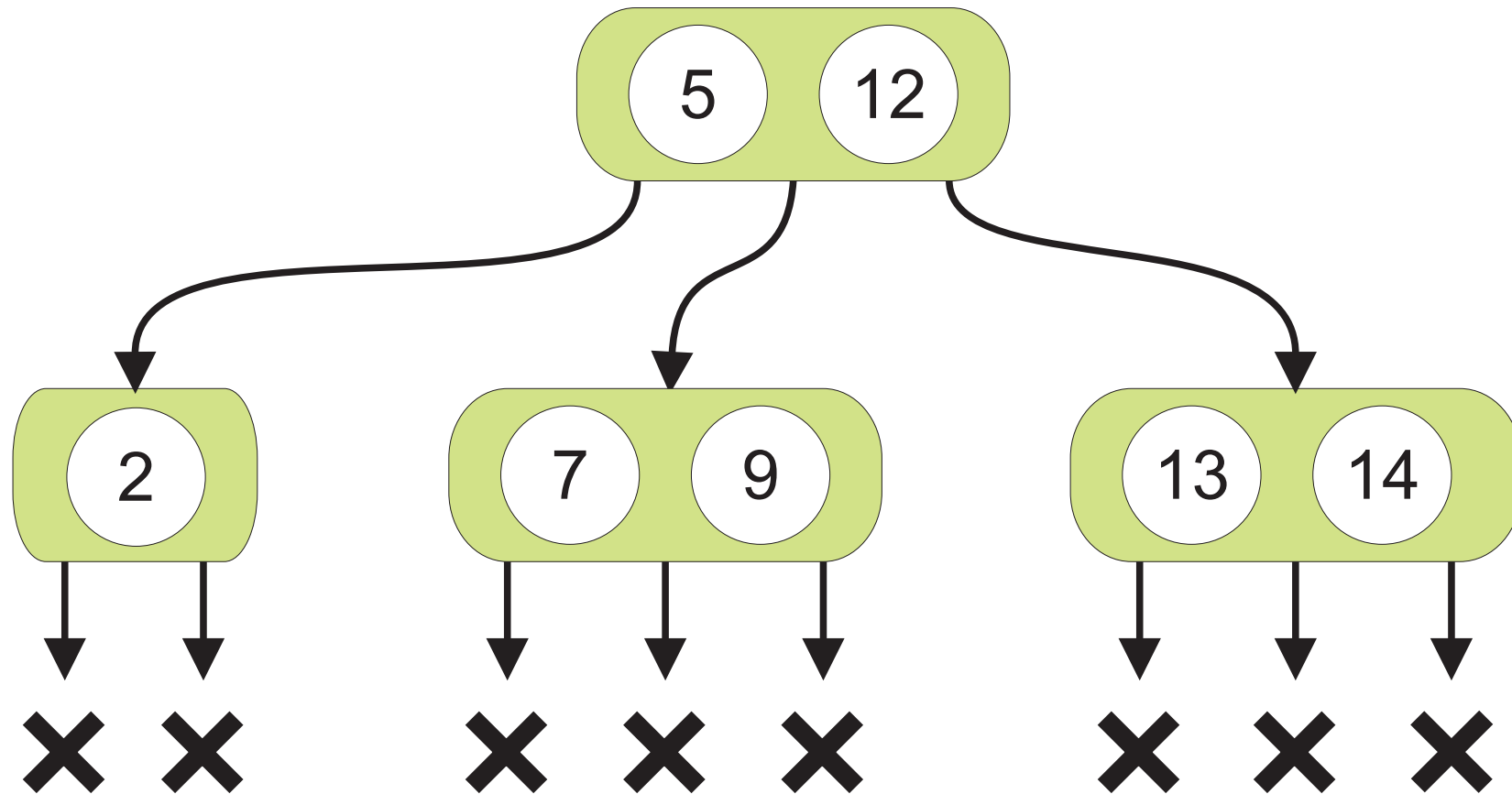
Suche in Baum mit Wurzel p den Schlüssel x

```
(1) if p==NULL then return NULL
(2) else {
(3)   var int i:=1
(4)     while i≤p.k and x>p.key do
(5)       i:=i+1
(6)     if i≤p.k and x==p.key[i] then
(7)       return p.data[i]
(8)     else return SEARCH(p.child[i-1],x)
(9) }
```

Analyse von $SEARCH(p,x)$

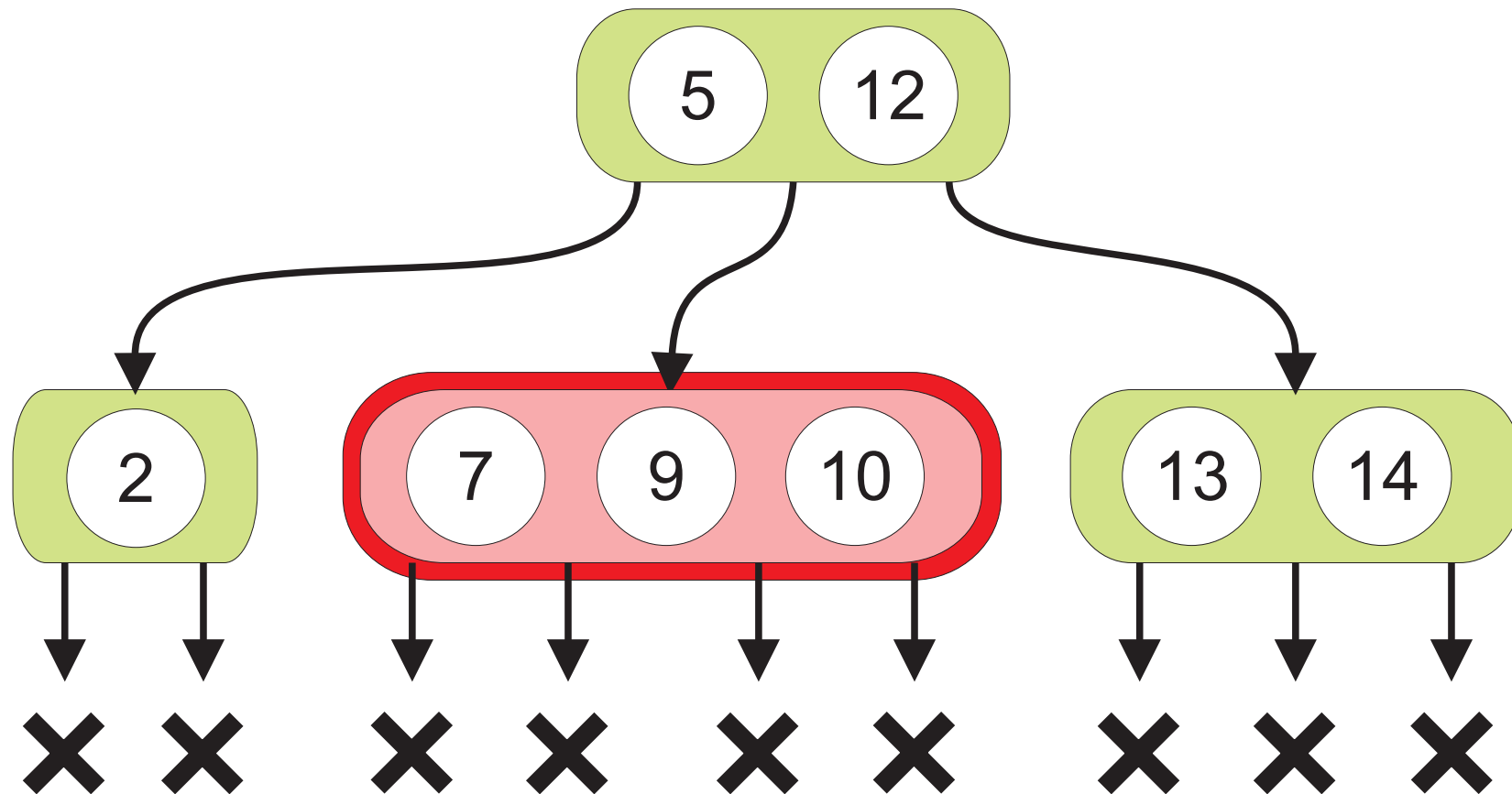
- In Implementierung: lineare Suche
- besser: binäre Suche
- Aber asymptotisch: beide Male Suche innerhalb eines Knotens konstant (wegen m konstant)
- Insgesamt: Laufzeit: $O(\text{Höhe von } B)$

Einfügen eines Schlüssels in einen B-Baum



- Baum vor dem Einfügen von **10**, **m=3**

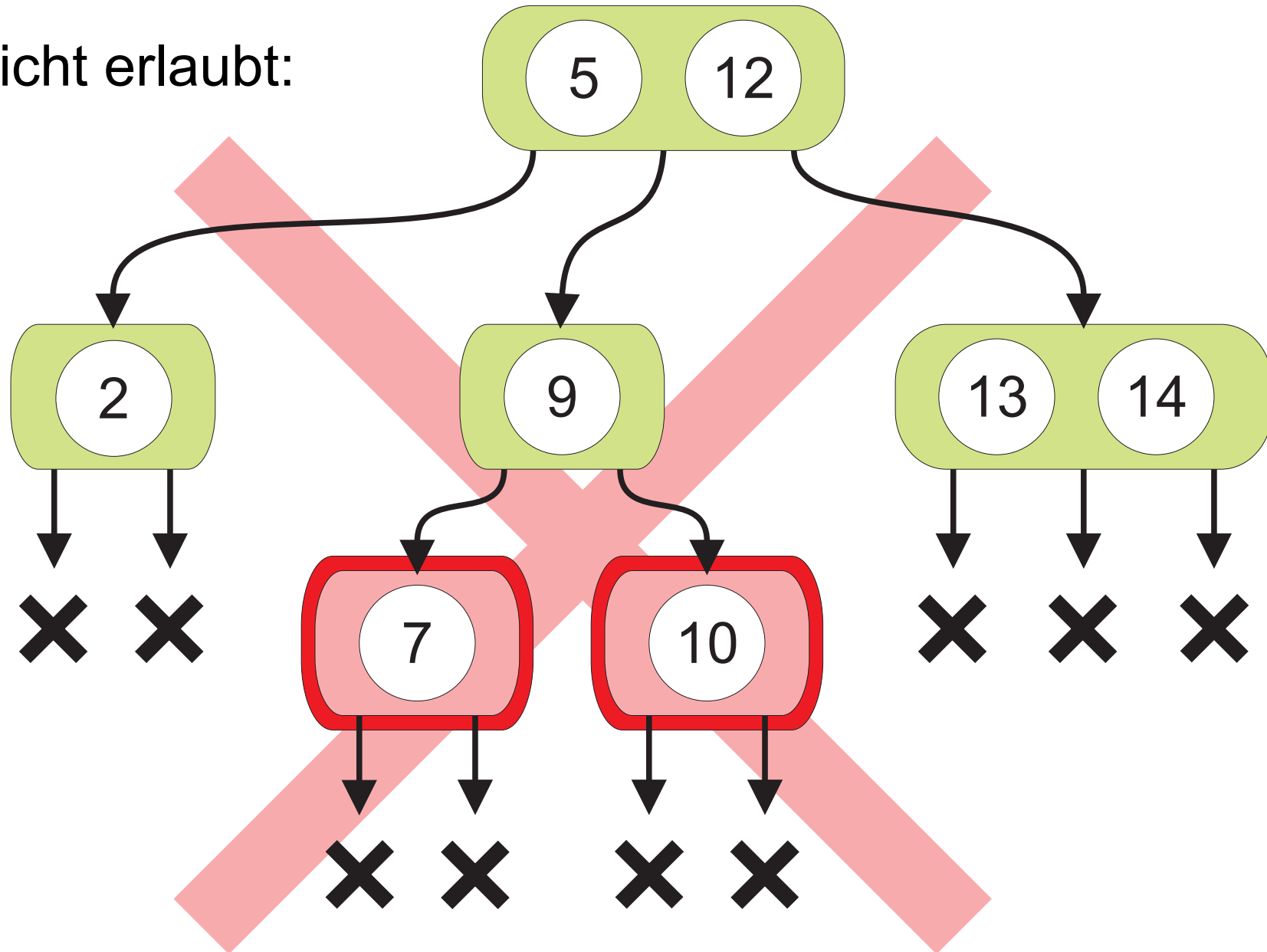
Einfügen in B-Baum



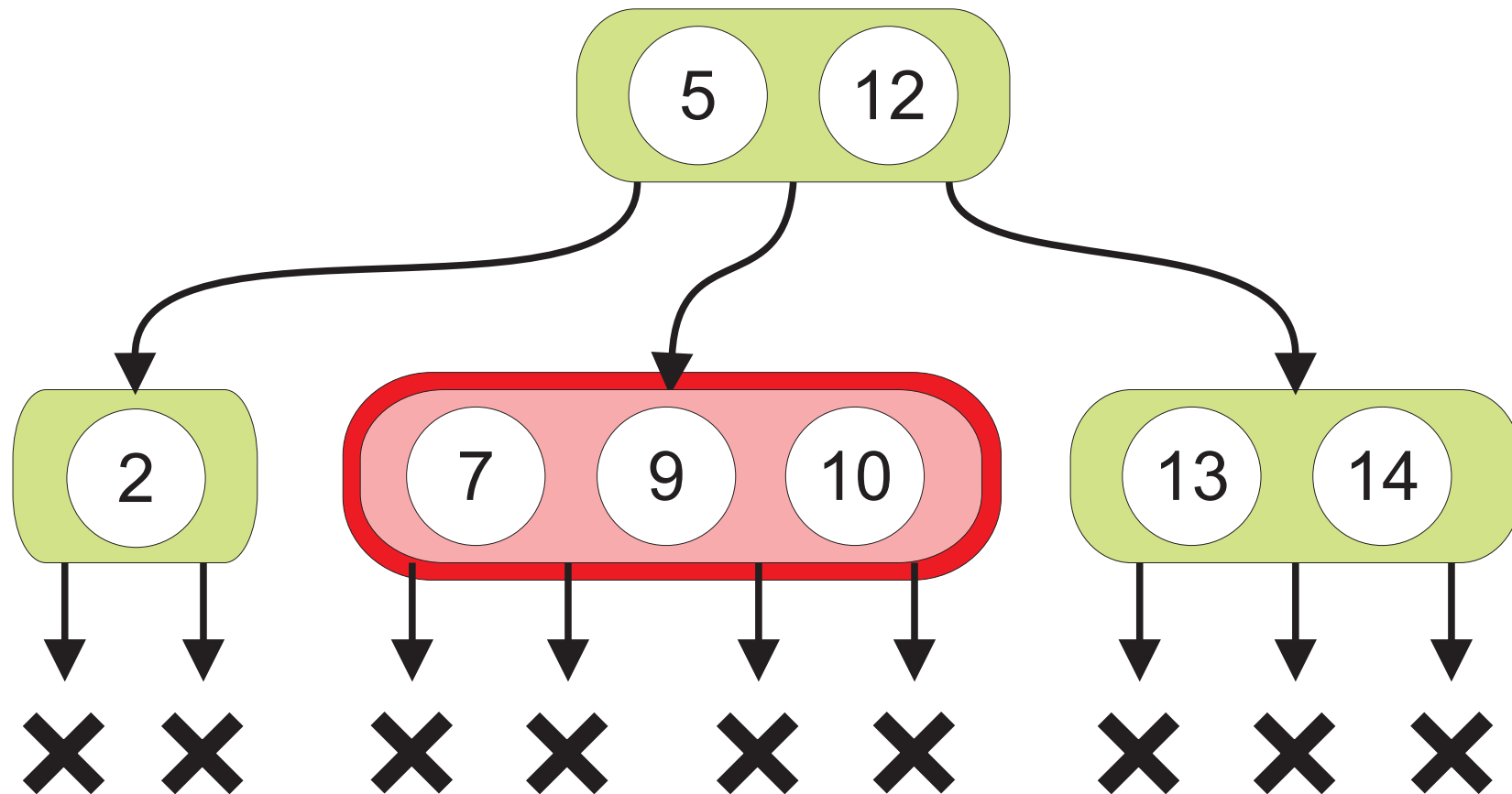
- Blatt wird zu groß: hat nun 3 Schlüssel!

Einfügen in B-Baum

nicht erlaubt:



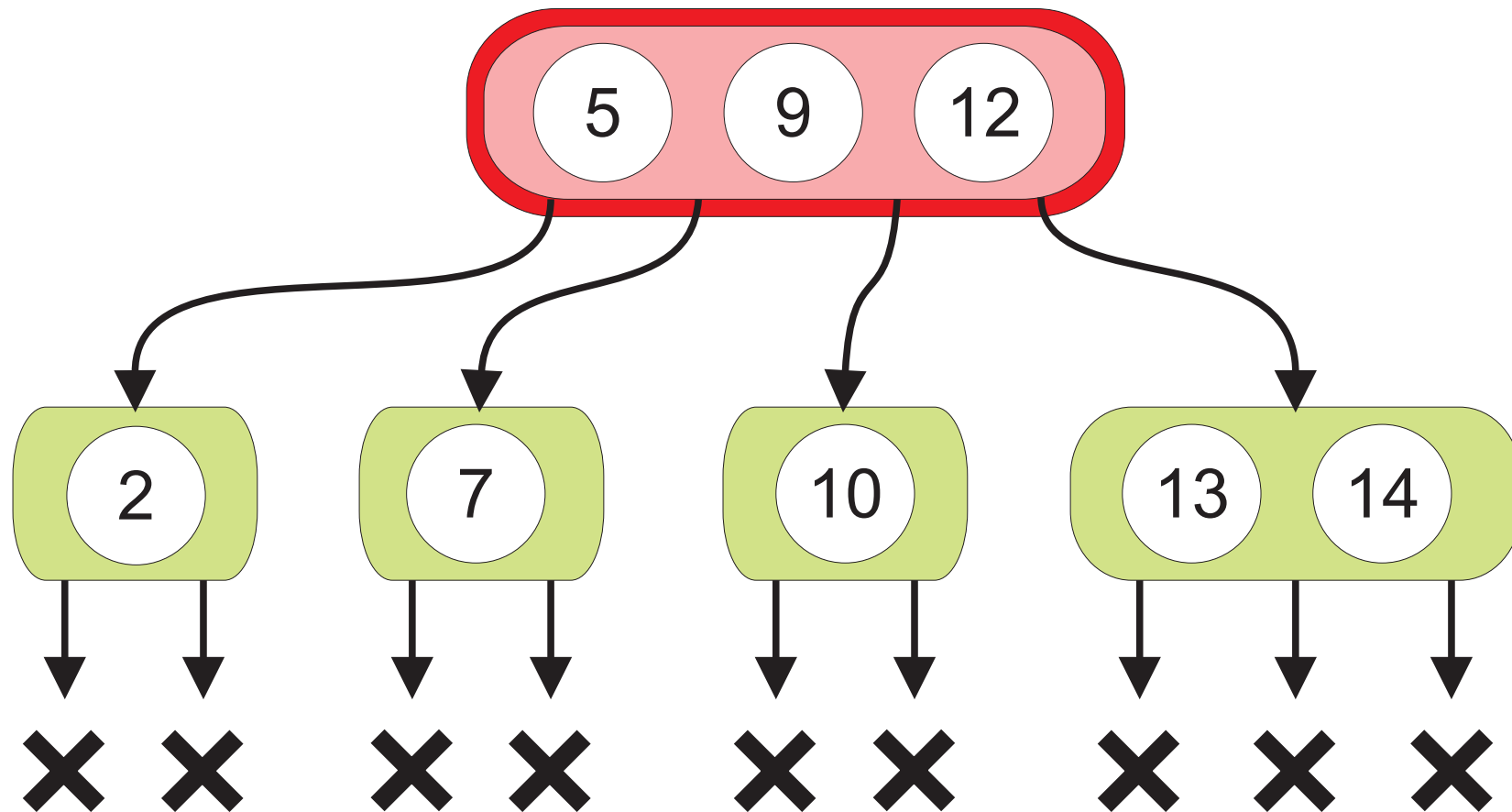
Einfügen in B-Baum



Blatt wird zu groß: hat nun 3 Schlüssel!

Einfügen in B-Baum

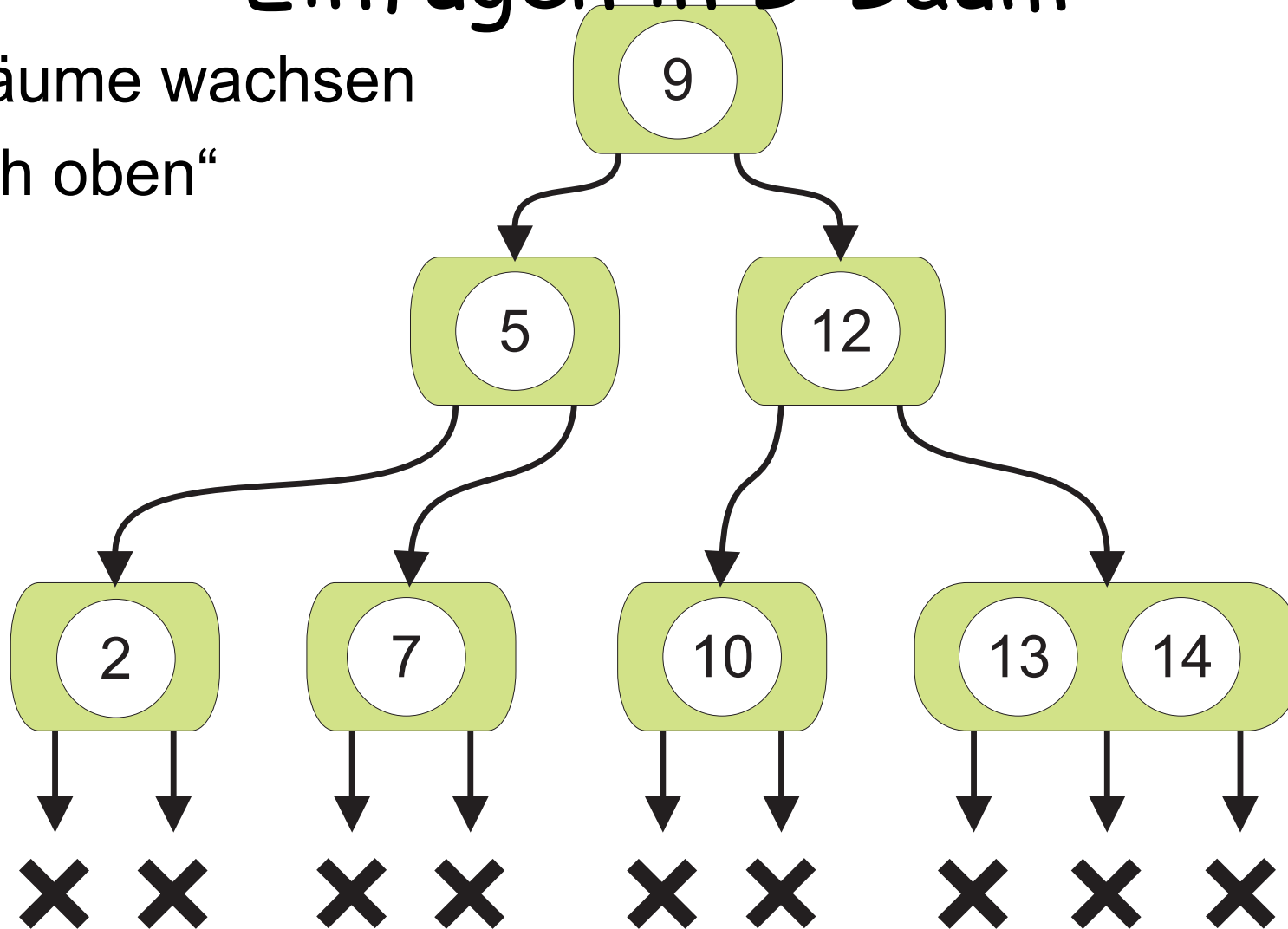
Elternknoten wird zu groß:



Aufspalten: der mittlere Schlüssel wandert in den Elternknoten

Einfügen in B-Baum

B-Bäume wachsen
„nach oben“



Aufspalten der Wurzel → neue Wurzel