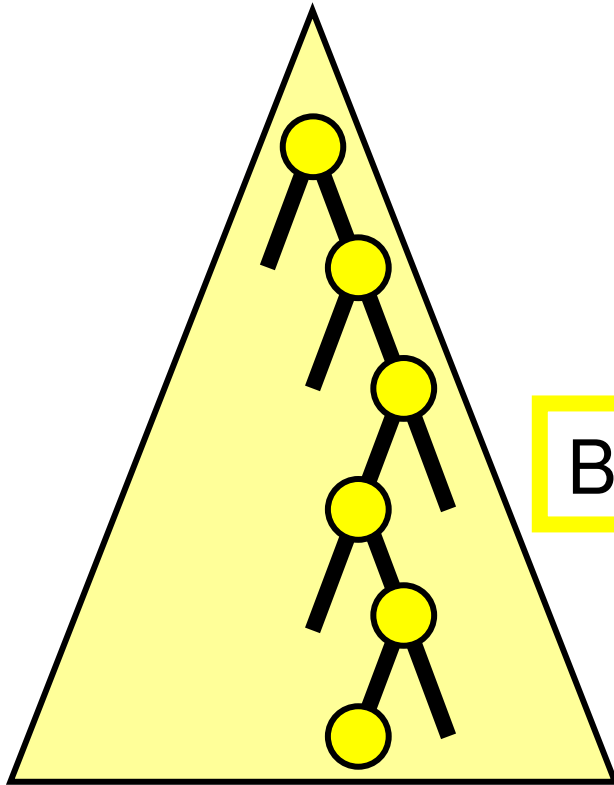


Kap. 4.4: B-Bäume

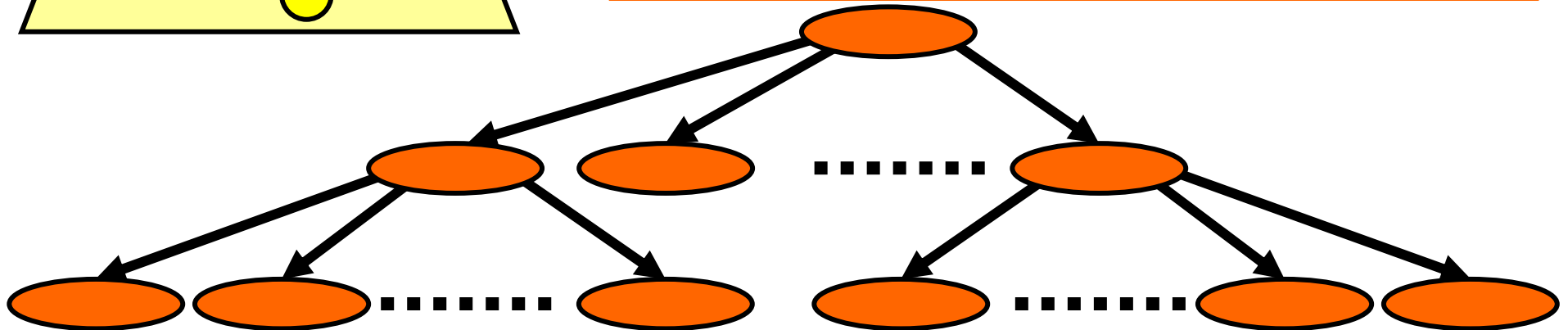
Große Bäume -> Extern -> Wie?



1 Mio Daten: Tiefe=
(binärer Baum) 20
(B-Baum d. Ord. 100) 3

Binärer Baum: $n = 2^h$

B-Baum der Ordnung m : $n = m^h$

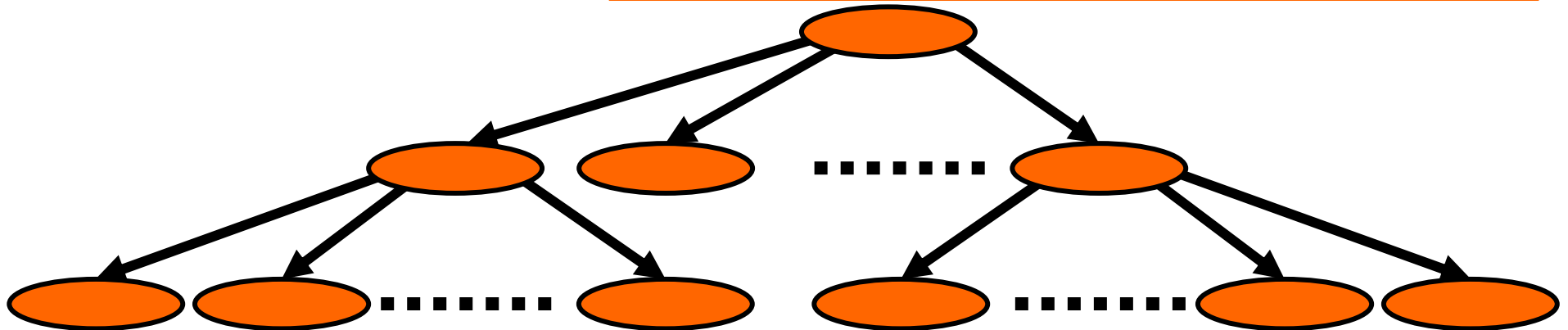


Große Bäume -> Extern -> Wie?

wähle m so, dass ein Knoten gerade noch in einen Speicherblock passt!

1 Mio Daten: Tiefe=
(binärer Baum) 20
(B-Baum d. Ord. 100) 3

B-Baum der Ordnung m : $n = m^h$



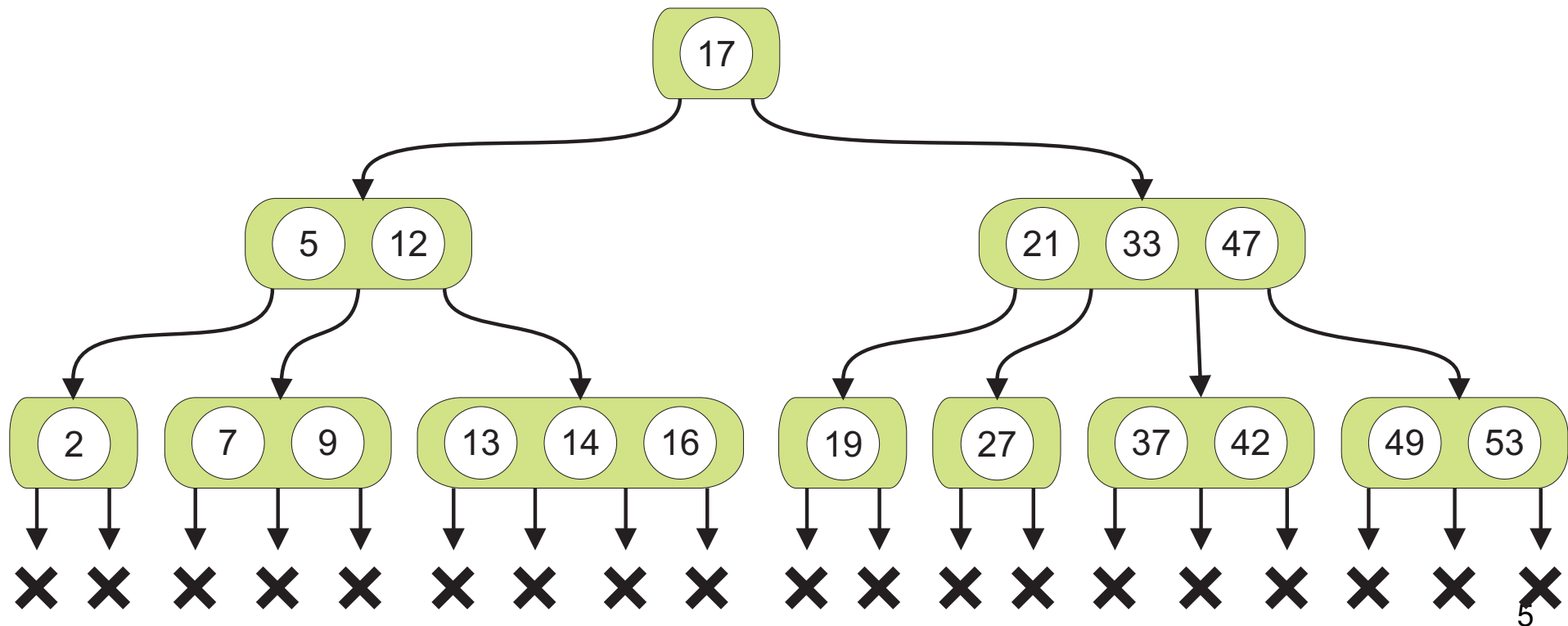
B-Bäume

- 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.



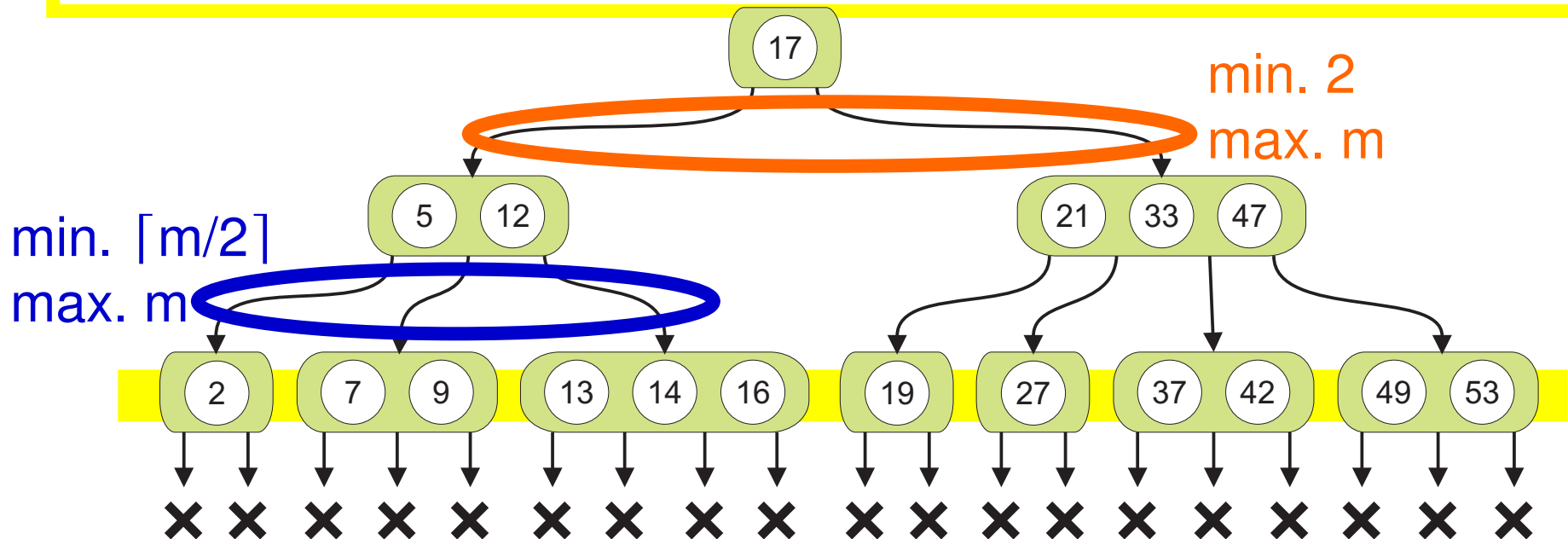
Definition B-Bäume

- 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.



Spezielle B-Bäume

- 2-3 Baum = B-Baum der Ordnung 3
[Hopcroft '70]
- 2-3-4 Baum = B-Baum der Ordnung 4
- Rot-Schwarz-Baum = 2-3-4 Baum in dem große Knoten durch binäre Teilbäume simuliert werden

Existenz von B-Bäumen

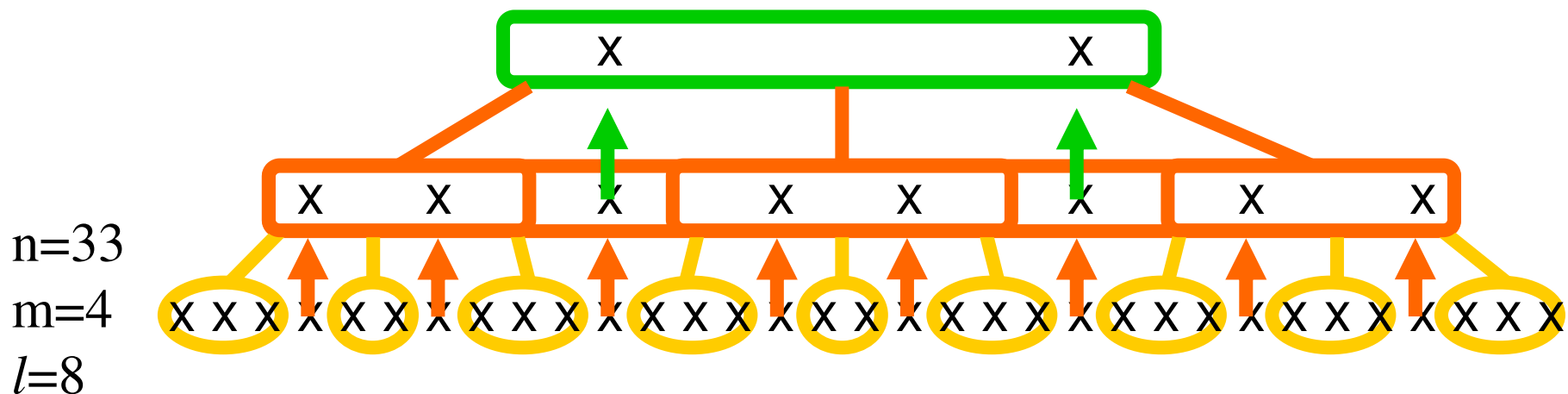
Theorem: Für jede beliebige Menge von Schlüsseln und jedes beliebige $m > 2$ existiert ein gültiger B-Baum der Ordnung m .

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.

Beweisidee der Existenz von B-Bäumen

Lemma: Gegeben sei eine Menge M aus n Schlüsseln und $m \geq 3$ mit $n \geq m$. Wir setzen $l := \lfloor n/m \rfloor$.

Wir können l Trennschlüssel aus M so auswählen, dass die übrigen Schlüssel in $l+1$ Teilmengen S_0, \dots, S_l zerfallen, die jeweils eine gültige Befüllung eines Knotens in einem B-Baum der Ordnung m darstellen.



Durchschnittliche Größe einer Teilmenge: $(n+1)/(l+1)-1 = 2,77$

Beweisidee der Existenz von B-Bäumen

Lemma: Gegeben sei eine Menge M aus n Schlüsseln und $m \geq 3$ mit $n \geq m$. Wir setzen $l := \lfloor n/m \rfloor$.

Wir können l Trennschlüssel aus M so auswählen, dass die übrigen Schlüssel in $l+1$ Teilmengen S_0, \dots, S_l zerfallen, die jeweils eine gültige Befüllung eines Knotens in einem B-Baum der Ordnung m darstellen.

Weiter mit Beweis zu Theorem: Diese $l+1$ Teilmengen S_i bilden die Blätter unseres B-Baums. Wiederhole diese Aufteilung rekursiv für die Trennschlüssel.

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.

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 $\lceil \log_{\lceil m/2 \rceil} ((n+1)/2) \rceil$

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

SEARCH(p,x)

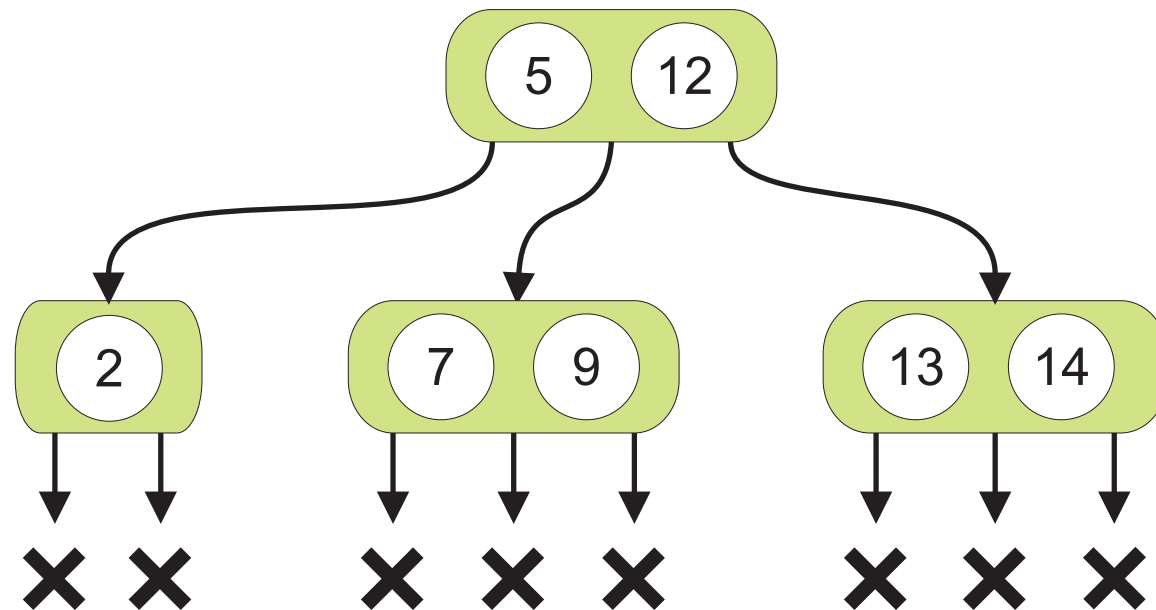
Suche in Baum mit Wurzel p den Schlüssel x

```
(1) if p==NULL then return NULL
(2) else {
(3)   var int i:=1
(4)   while i≤p.k and x>p.key[i] 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 $SEARCH(p,x)$

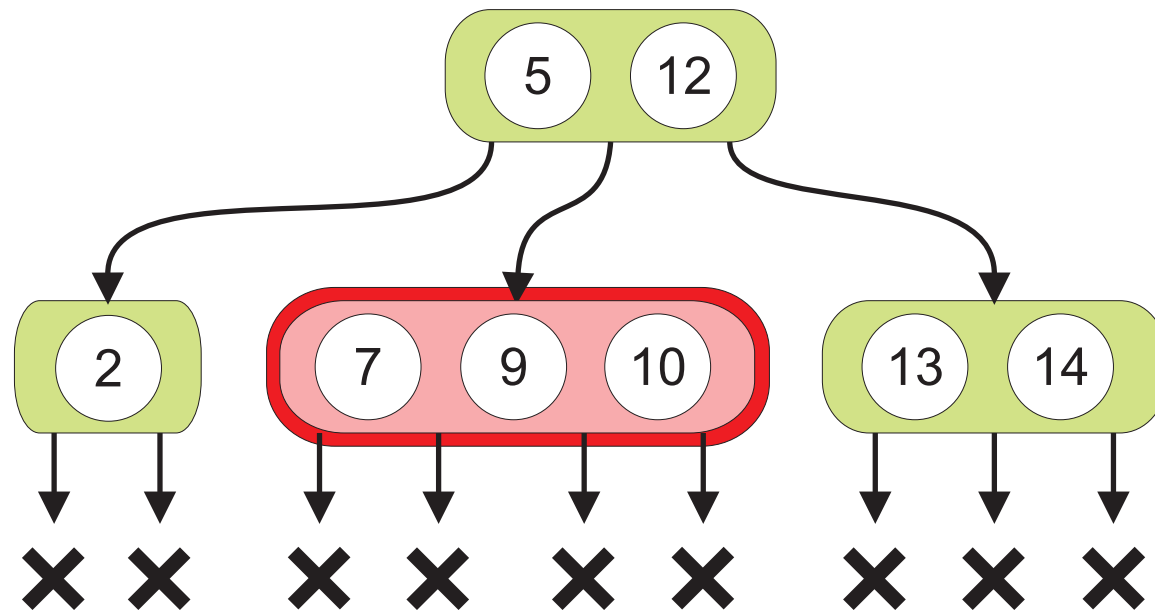
- In Implementierung: lineare Suche
- besser: binäre Suche
- Aber asymptotisch: beide Male Suche innerhalb eines Knotens konstant (wegen m konstant)
- Insgesamt: Laufzeit: $O(\text{Höhe des B-Baums})$

Einfügen eines Schlüssels in einen B-Baum



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

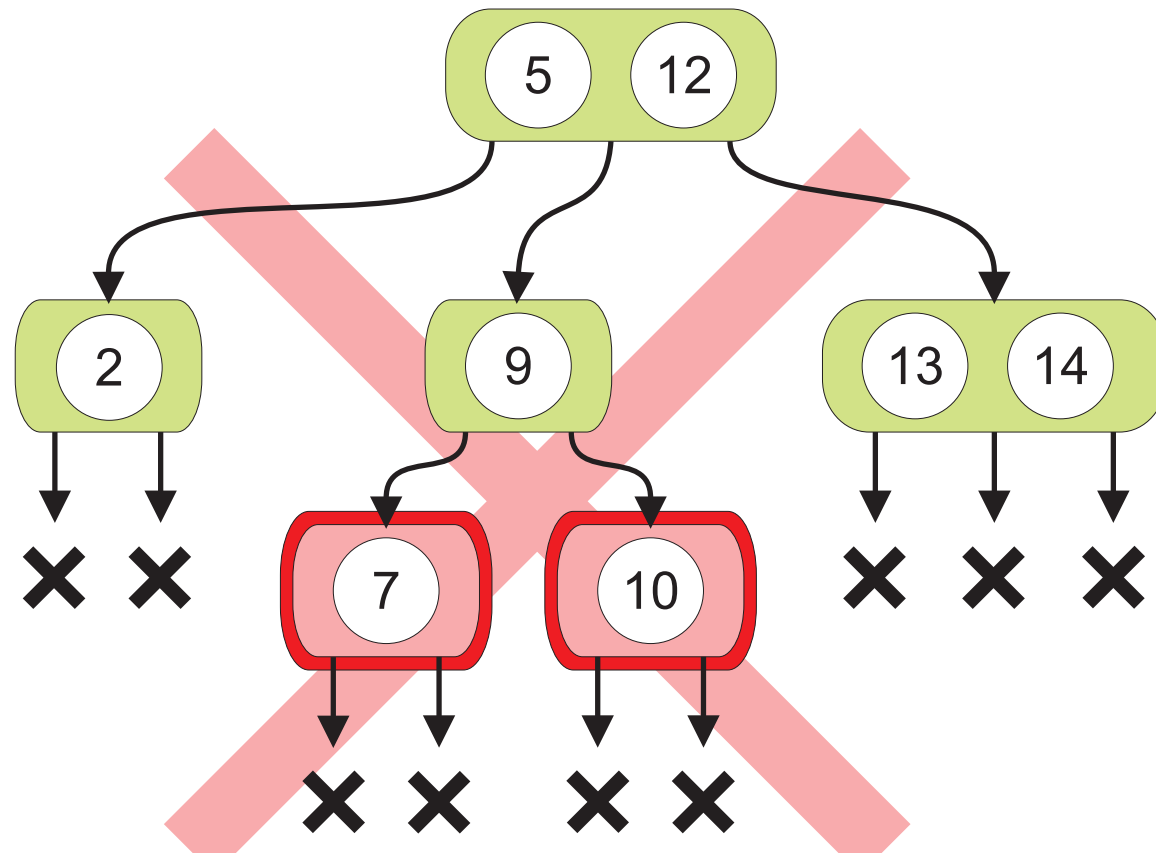
Einfügen in B-Baum



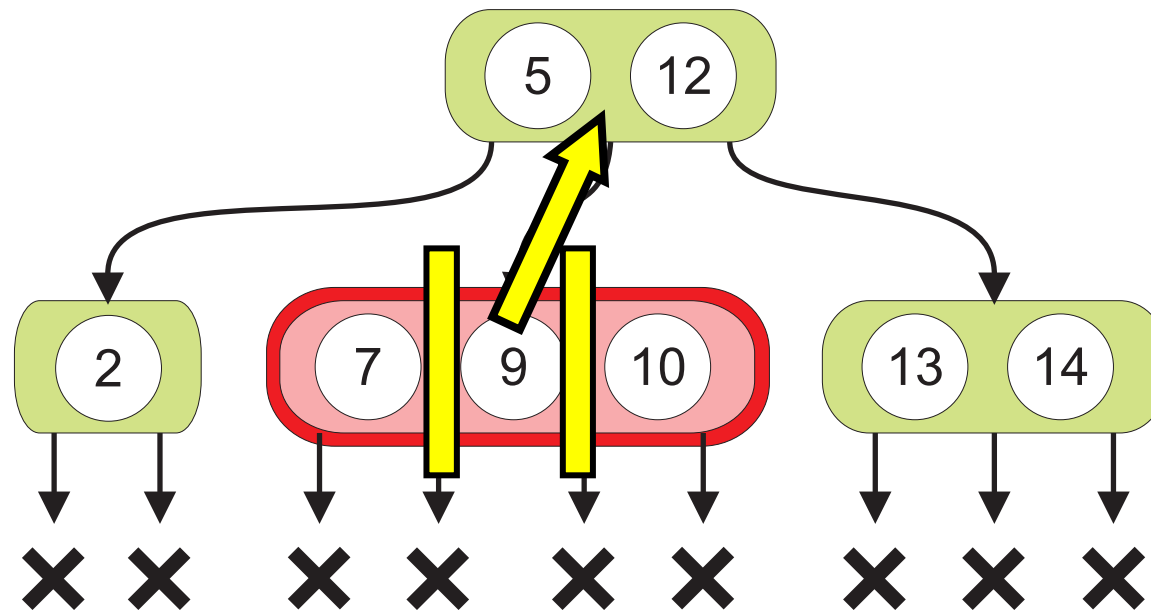
Blatt wird zu groß: hat nun 3 Schlüssel!

Einfügen in B-Baum

nicht erlaubt:

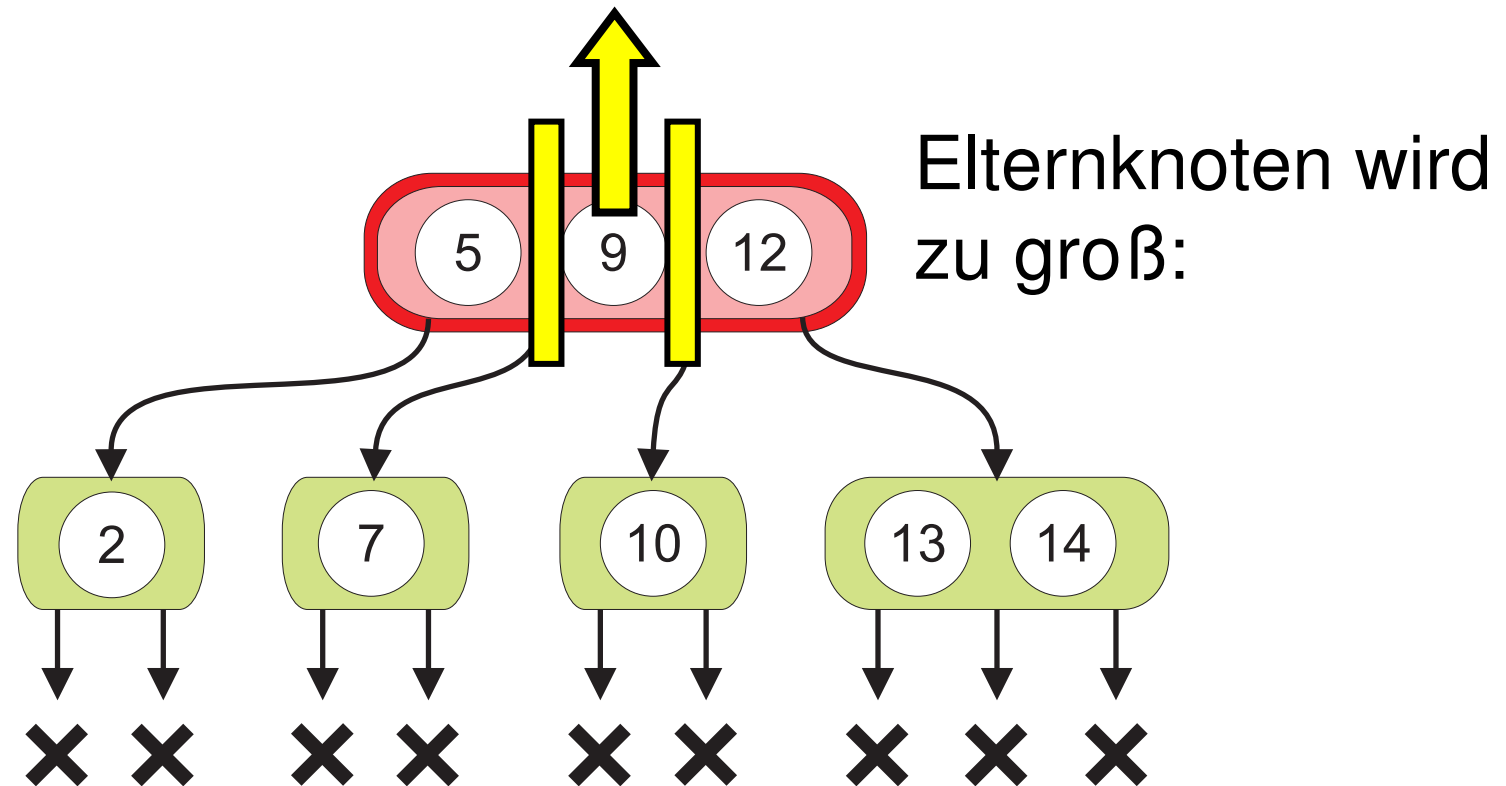


Einfügen in B-Baum



Blatt wird zu groß: hat nun 3 Schlüssel!

Einfügen in B-Baum

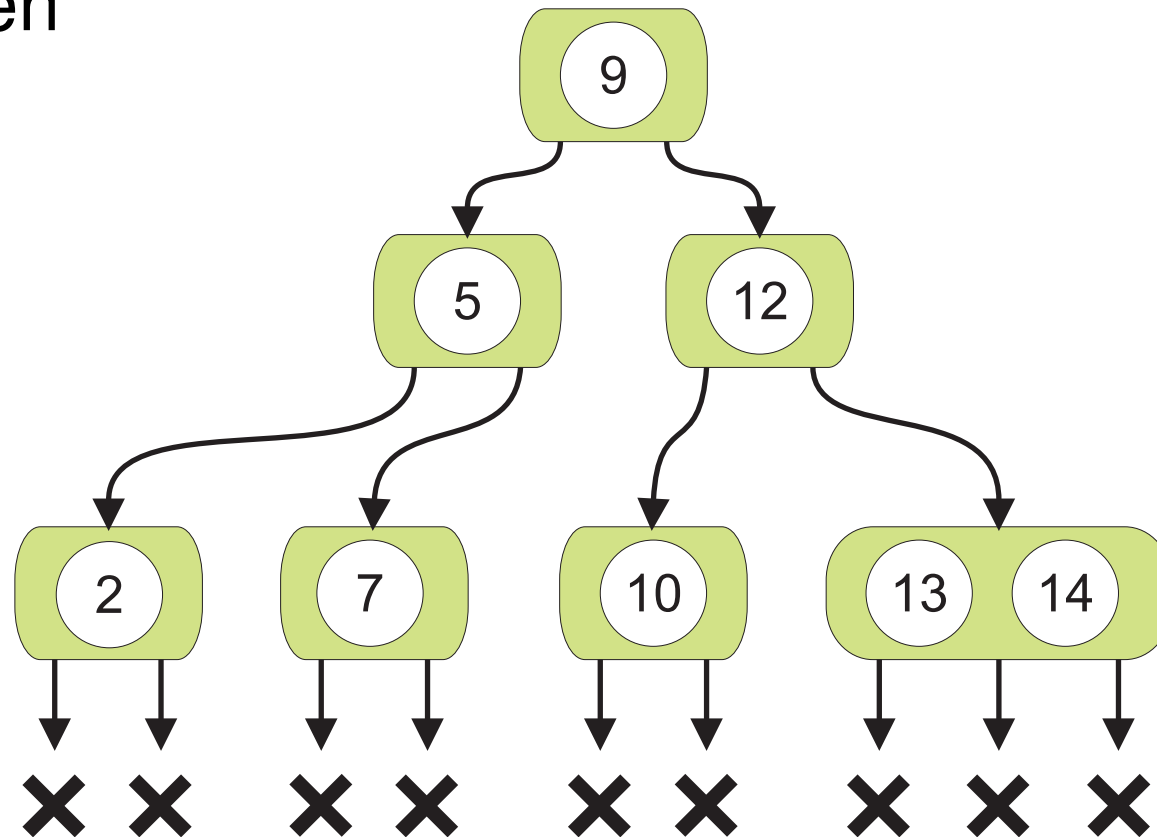


Aufspalten: der mittlere Schlüssel wandert in den Elternknoten

Einfügen in B-Baum

B-Bäume wachsen

„nach oben“



Aufspalten der Wurzel → neue Wurzel

Implementierung von INSERT(r,s) in B-Bäumen

1. Einfügeposition suchen: SEARCH(r,s)
2. Einfügen in Blatt
3. Wiederhole
 - Falls dieser Knoten zu viele Schlüssel enthält, dann **Aufspalten**: der mittlere Schlüssel wird zum Elterknoten verschoben.
bis Knoten nicht mehr zu groß.
4. Falls wir die Wurzel aufspalten mußten, setze:
neue Wurzel.

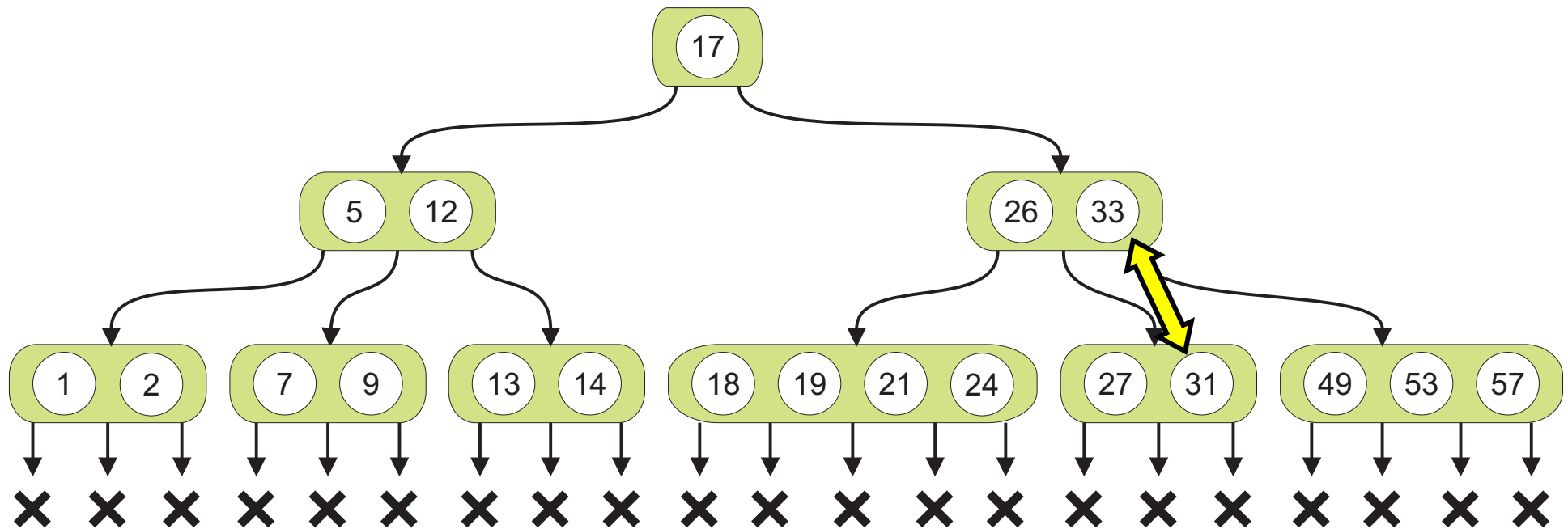
Analyse von INSERT(r,s)

Laufzeit: $O(\text{Höhe des Baums}) = \Theta(\log n)$

Entfernen in B-Bäumen

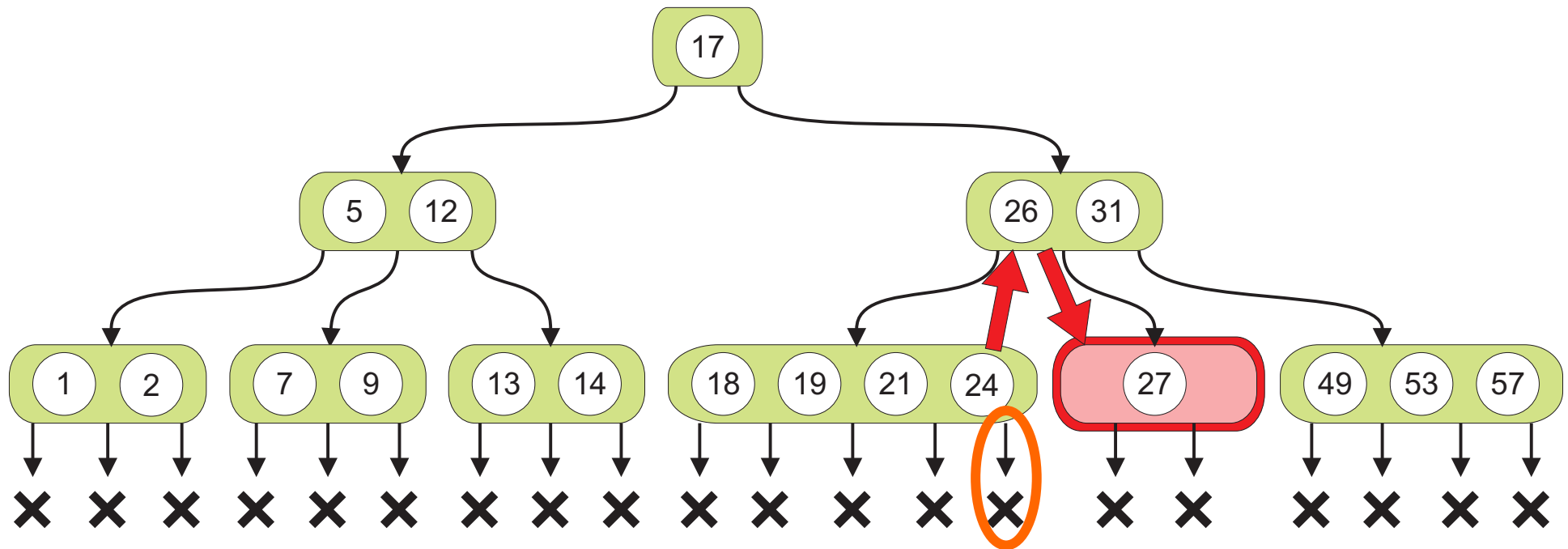
B-Baum der Ordnung 5

Entfernen von 33



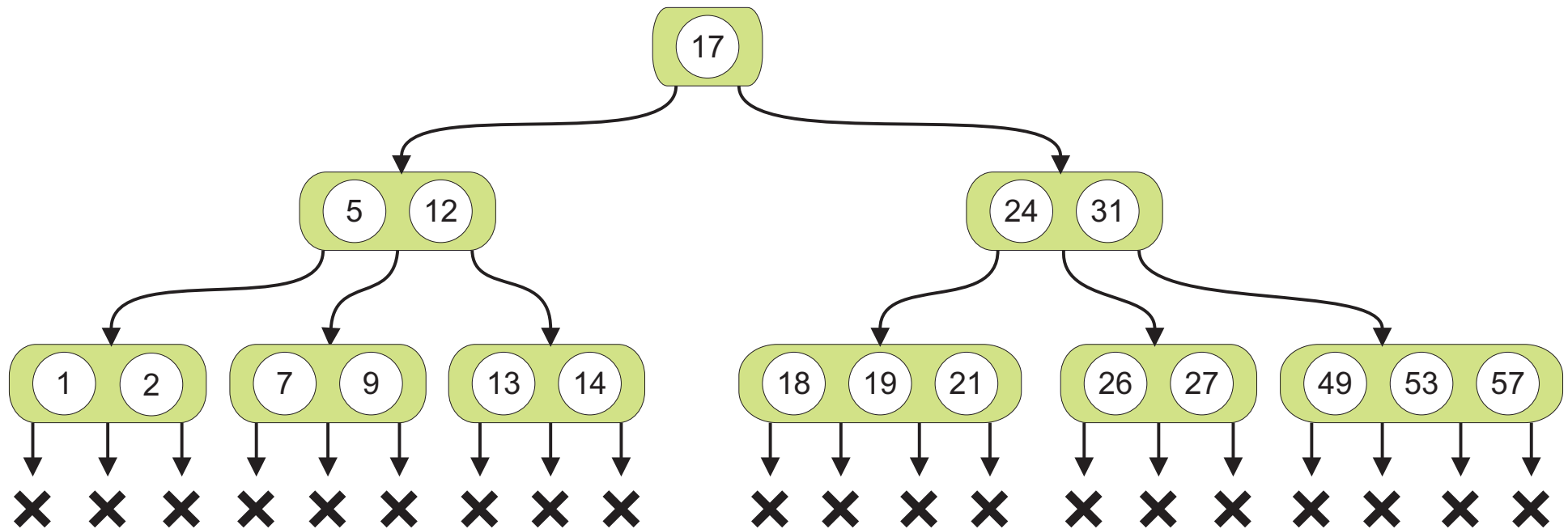
Entfernen in B-Bäumen

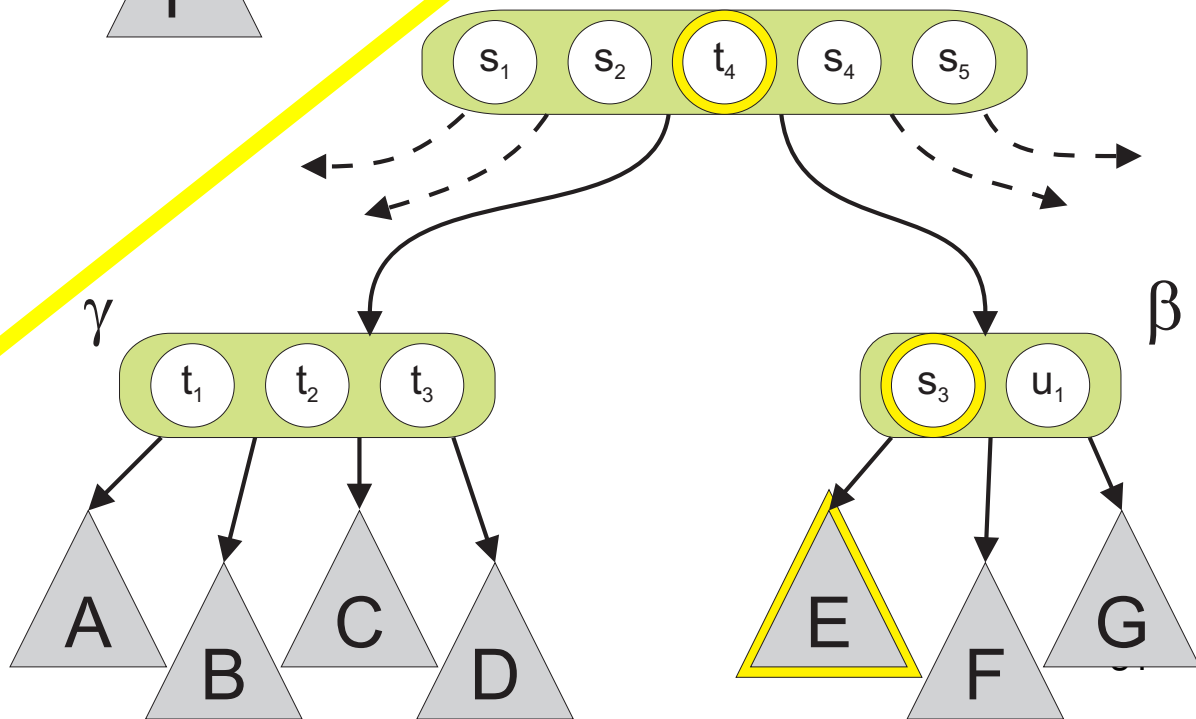
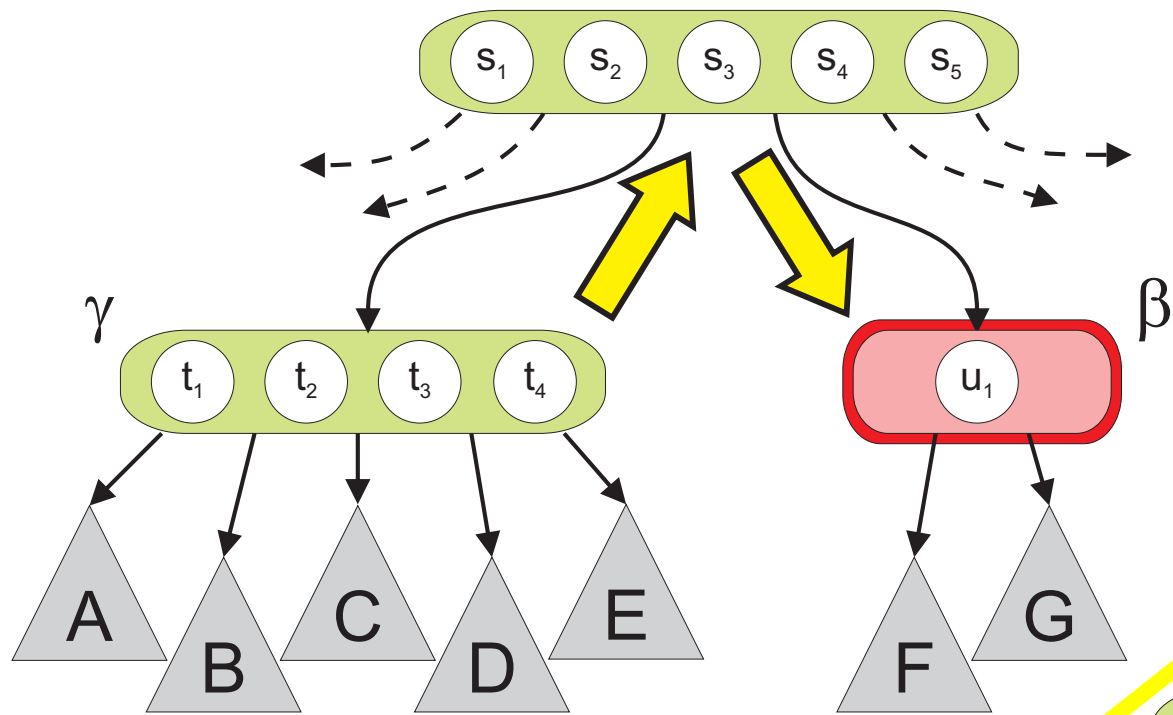
- Entfernen von 33



Entfernen in B-Bäumen

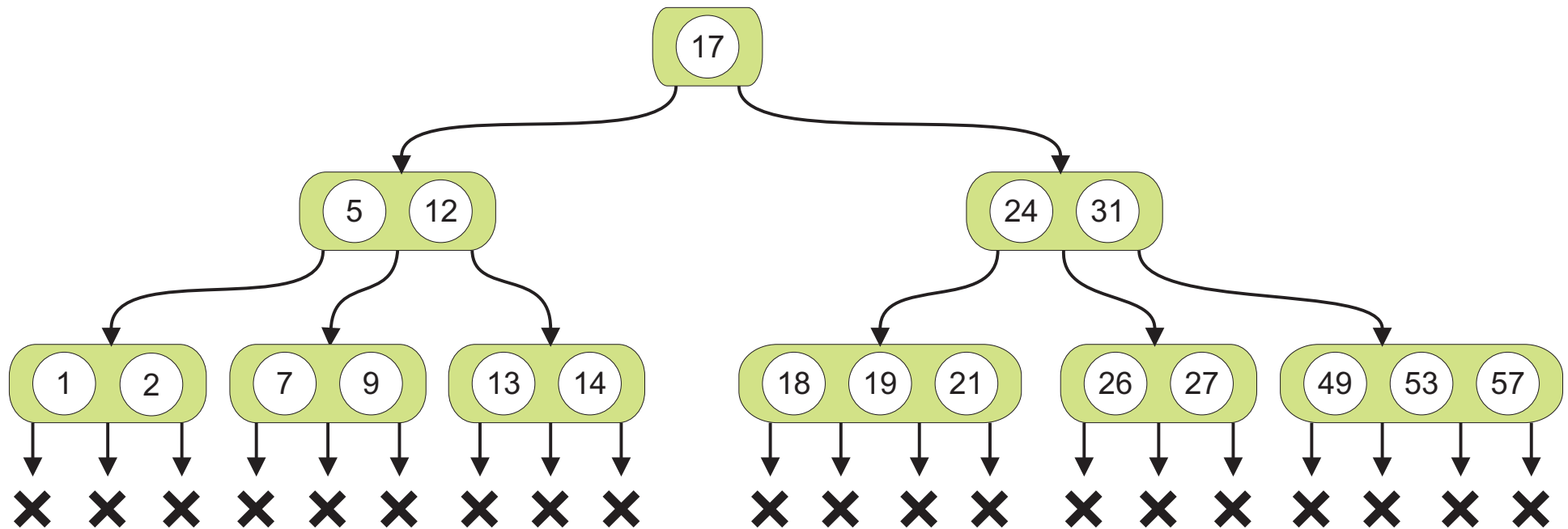
- B-Baum nach der Rotation





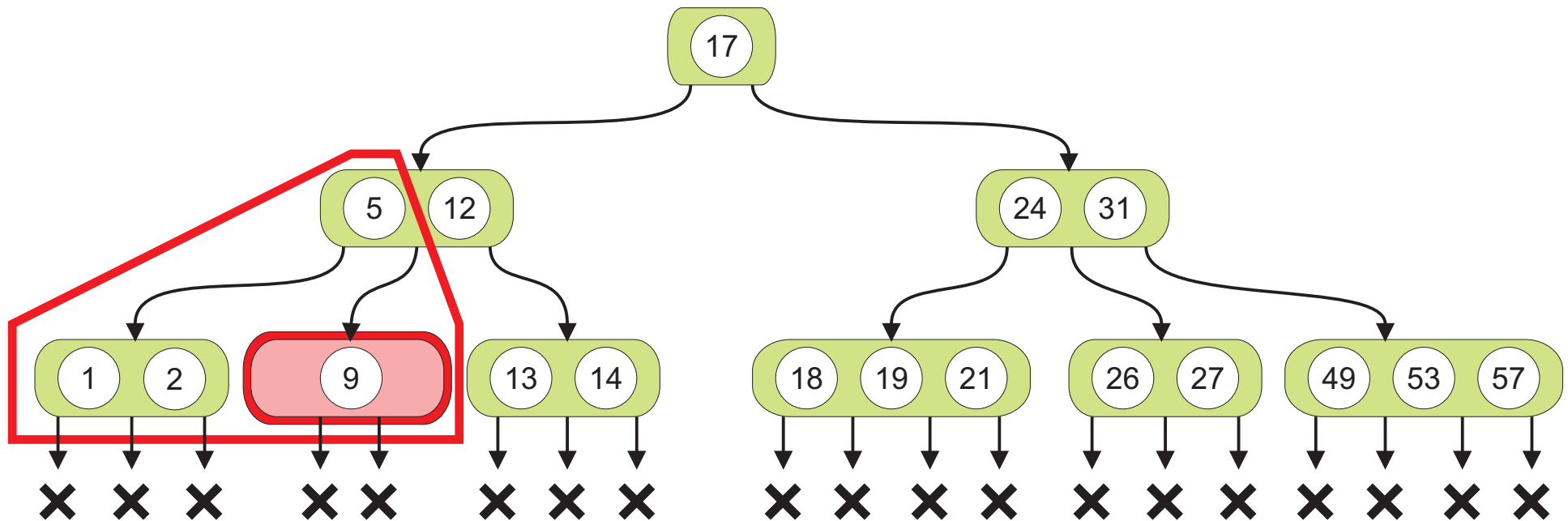
Entfernen in B-Bäumen

- Entfernen von 7



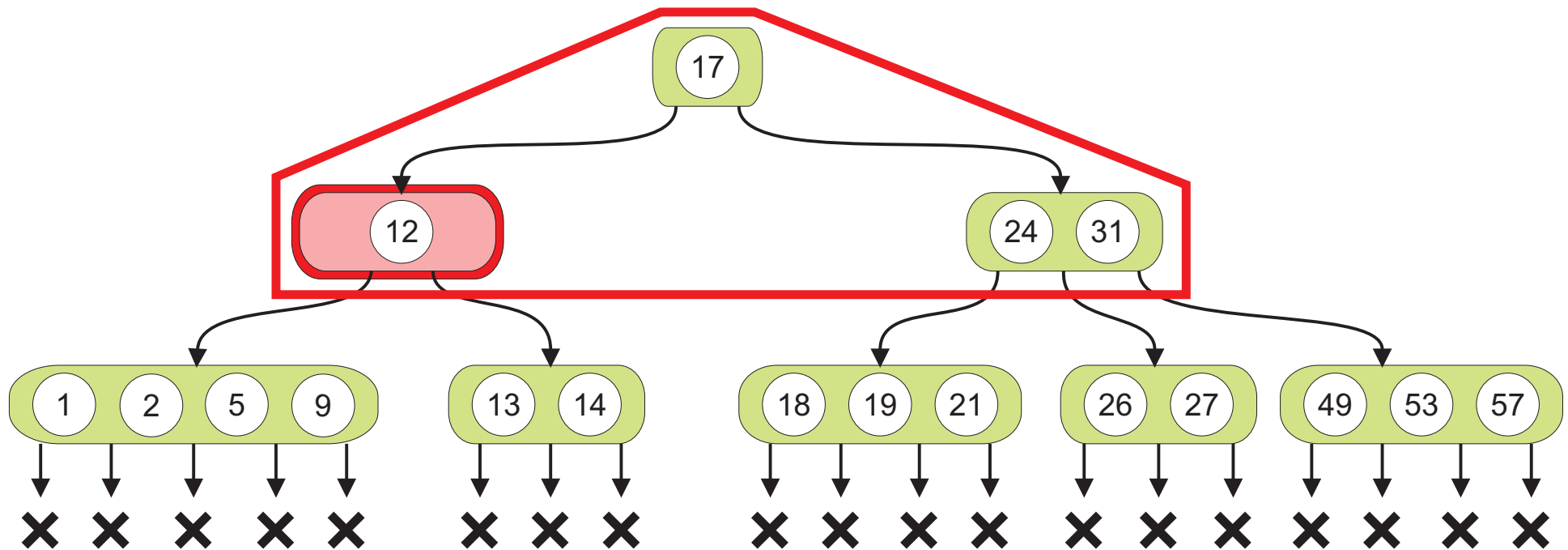
Entfernen in B-Bäumen

- Entfernen von 7



Entfernen in B-Bäumen

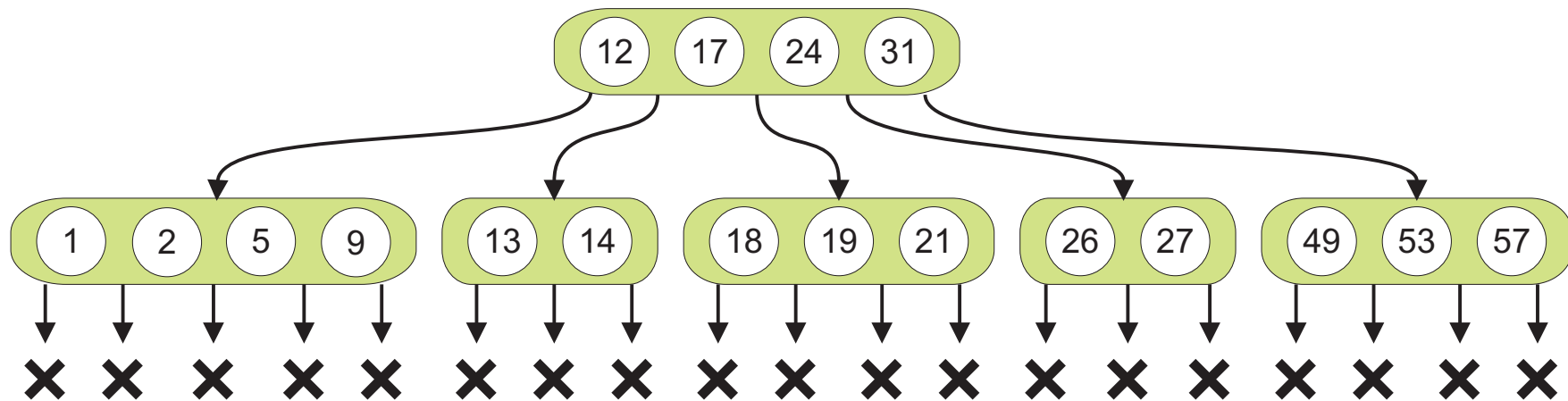
- Entfernen von 7



Entfernen in B-Bäumen

- Entfernen von 7

B-Bäume schrumpfen
„von oben“



Analyse von Delete(r,s)

Laufzeit: $O(\text{Höhe des Baums}) = \Theta(\log n)$