Kapitel 6
Suchen

- ... auf verketteten Strukturen:
  - Suchbäume
  - ausgeglichene Bäume
  - B-Bäume
- ... auf sequentiellen Strukturen:
  - Hashing

Suchen
Grundbegriffe

- **Suchen** = Tätigkeit, in einem vorgegebenen Datenbestand alle Objekte zu ermitteln, die eine best. Bedingung, das **Suchkriterium**, erfüllen und ggf. einen Verweis auf diese Objekte abzuliefern
- Suche erfolglos ⇒ meist neues Objekt in den Datenbestand einfügen
- **Schlüssel**=Merkmal, das Objekt eindeutig identifiziert

Suchen
sequentiell und verkettet

- Bekannt:
  - **sequentielles Suchen** auf allgemeinen Datenbeständen
  - **binäres Suchen** auf sortierten Datenbeständen
- **Sequentielle** Datentypen: bleiben über längeren Zeitraum unverändert; kaum Einfüge- oder Ausfügeoperationen
- praktische Anwendungsfälle: meist **verkettete** Speicherverfahren, bei denen Einfügeoperationen effizient möglich sind

Suchen
Suchbäume

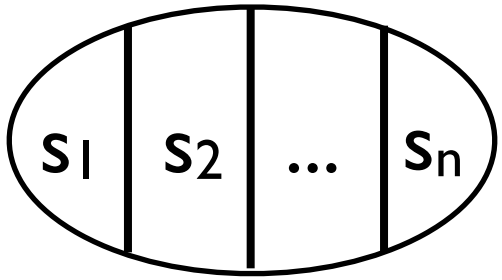
- Datenstruktur auf der Basis von binären Bäumen, in der man Objekte effizient finden, einfügen und löschen kann
- Voraussetzung: linear geordnete Menge M von **Schlüsseln**

Suchen

Suchbäume - Struktur

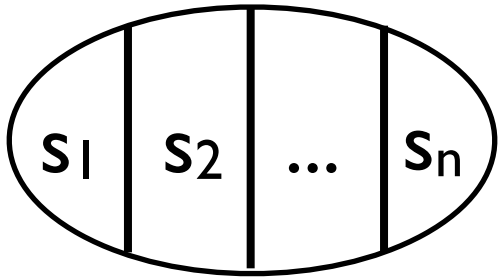
Suchen

Suchbäume - Struktur



Suchen

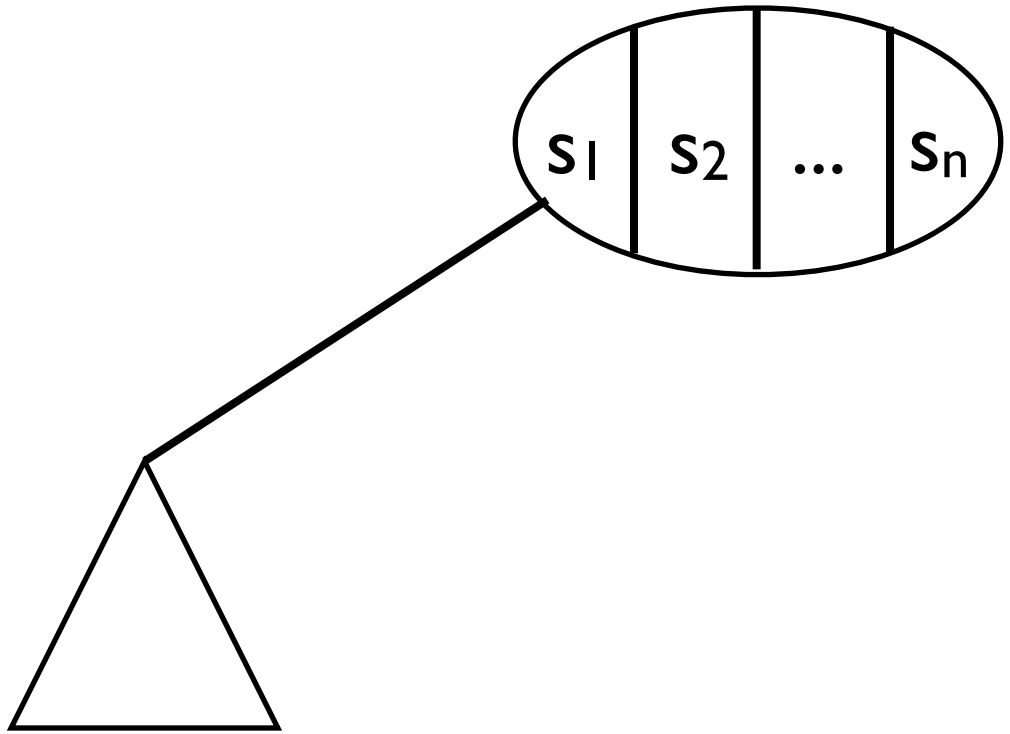
Suchbäume - Struktur



Schlüssel $s_1,\ldots,s_n$

Suchen

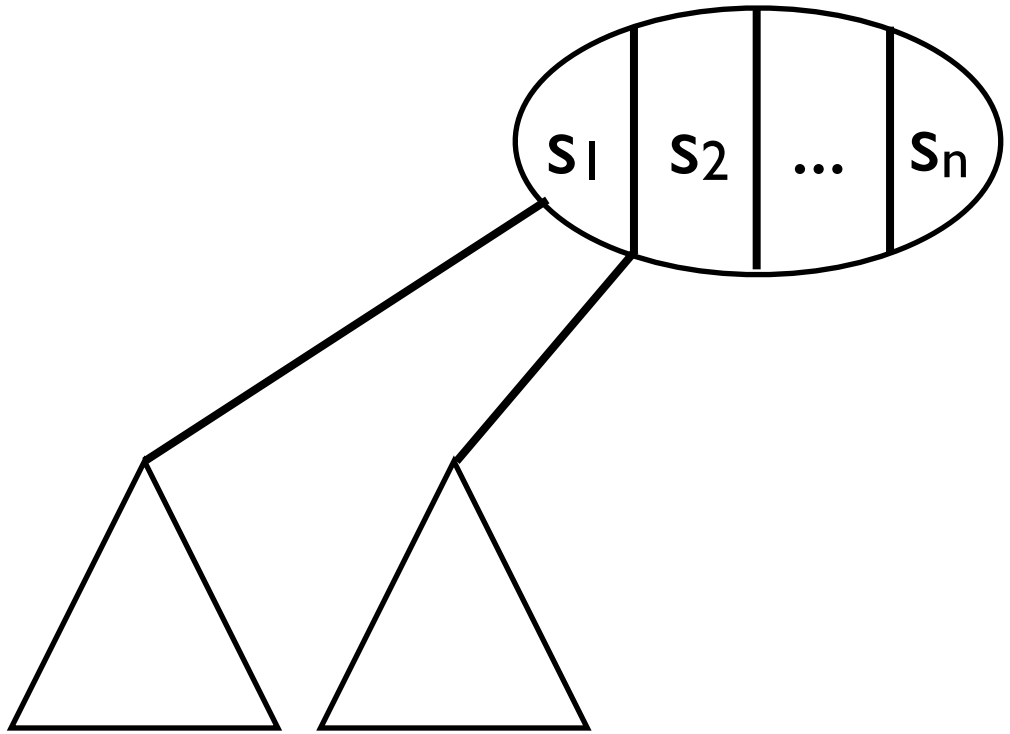
Suchbäume - Struktur



Schlüssel $s_1,\ldots,s_n$

Suchen

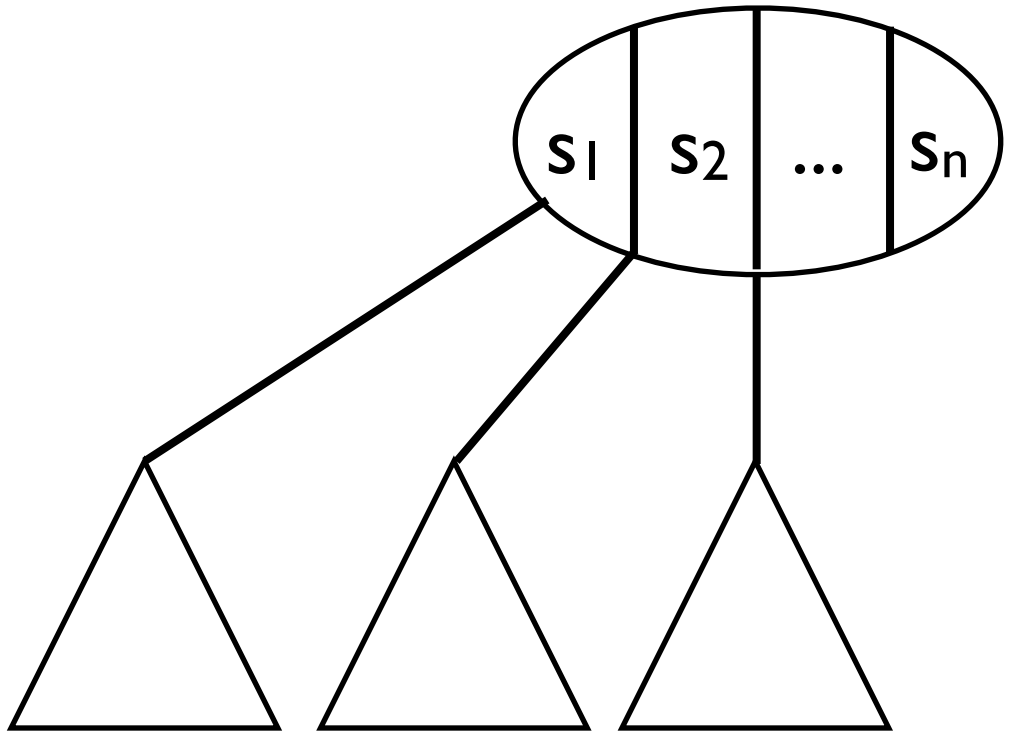
Suchbäume - Struktur



Schlüssel $s_1,\ldots,s_n$

Suchen

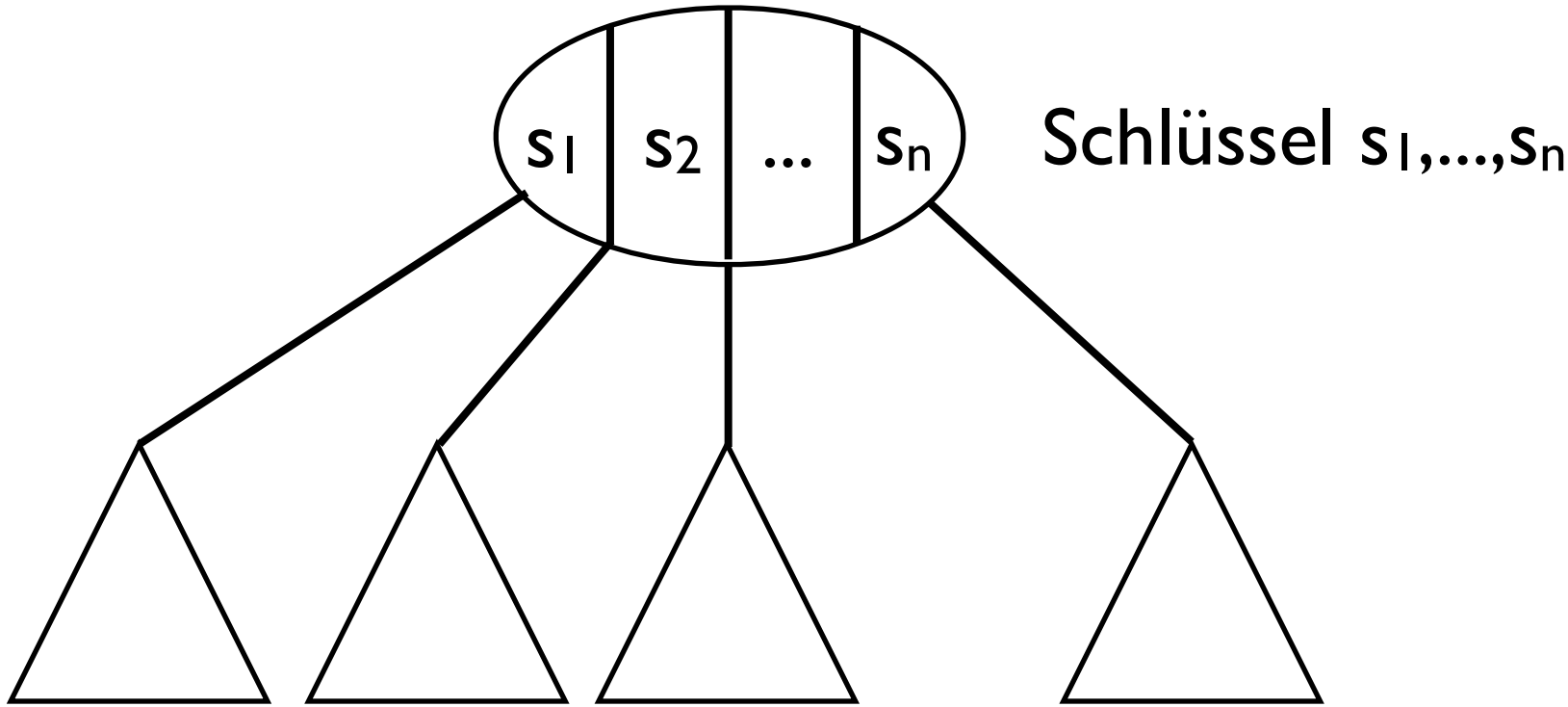
Suchbäume - Struktur



Schlüssel $s_1,\ldots,s_n$

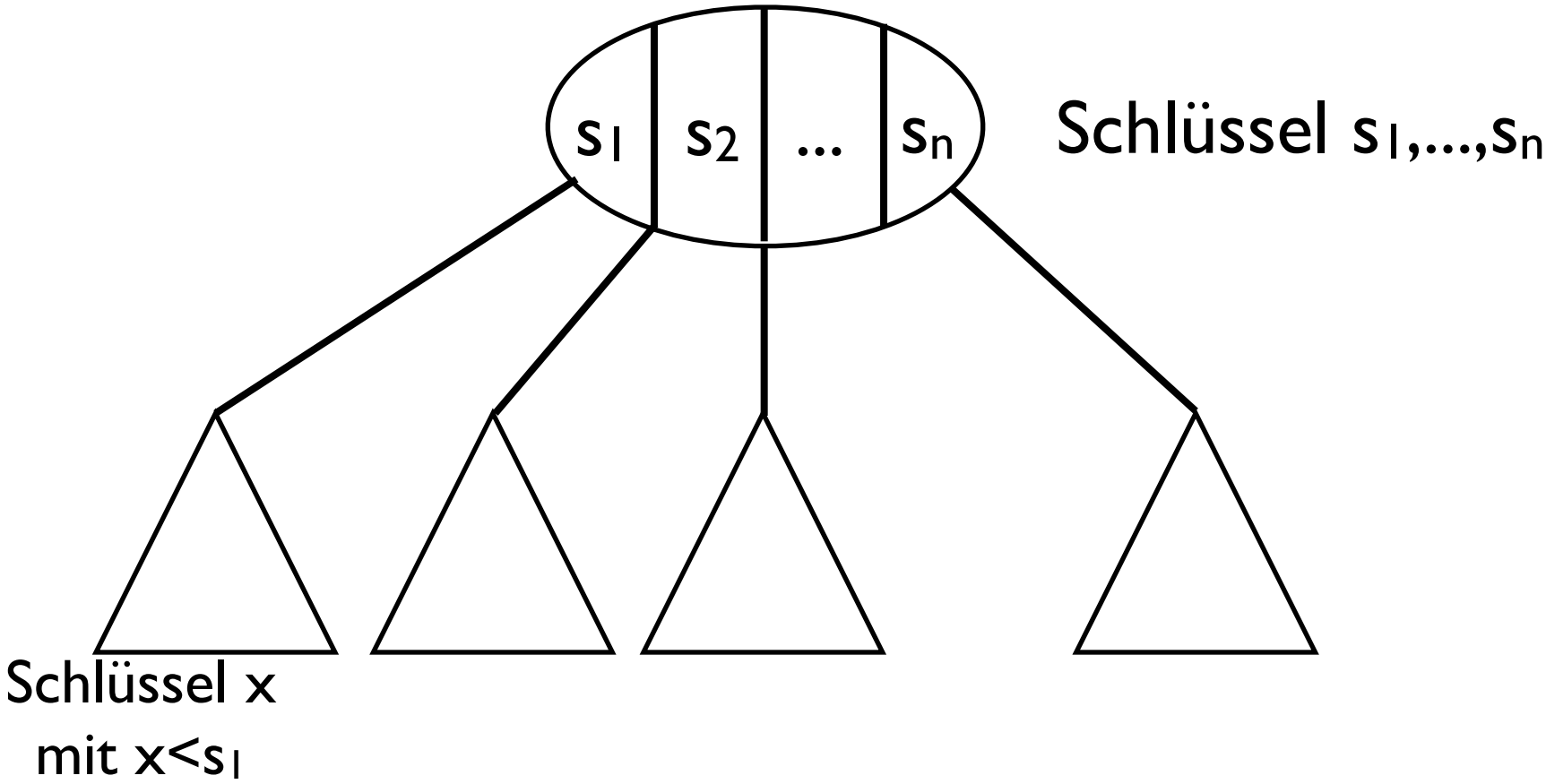
Suchen

Suchbäume - Struktur



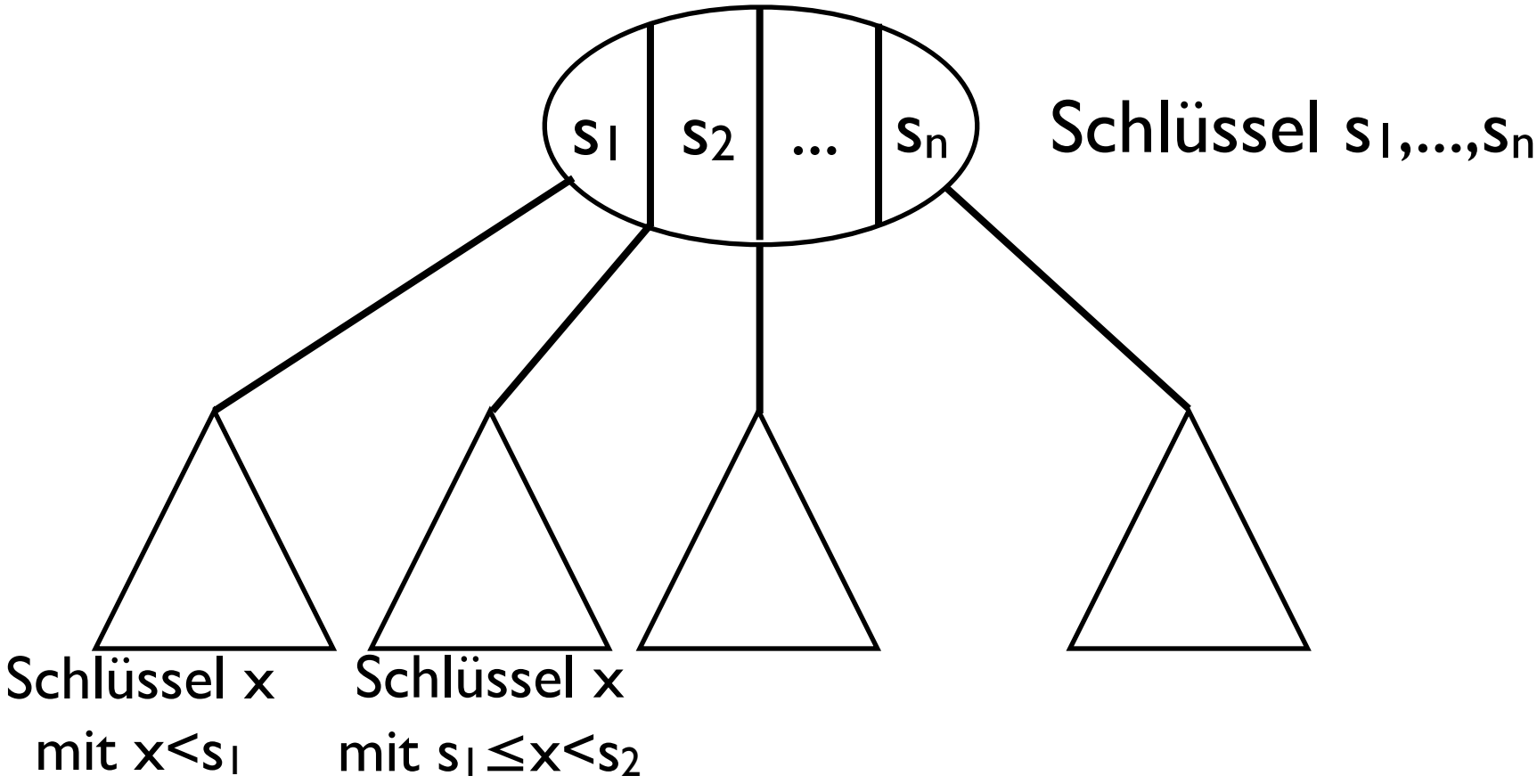
Suchen

Suchbäume - Struktur



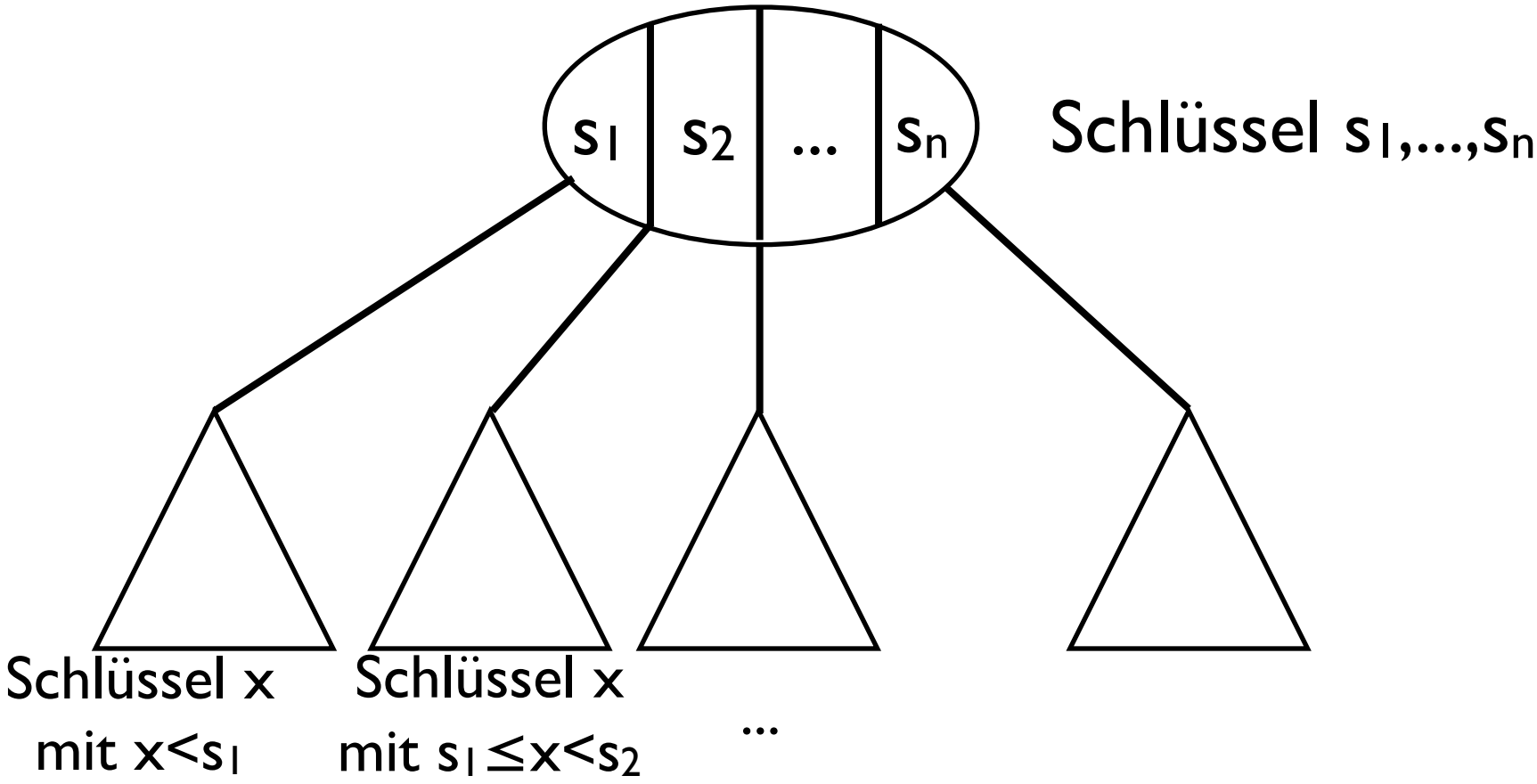
Suchen

Suchbäume - Struktur



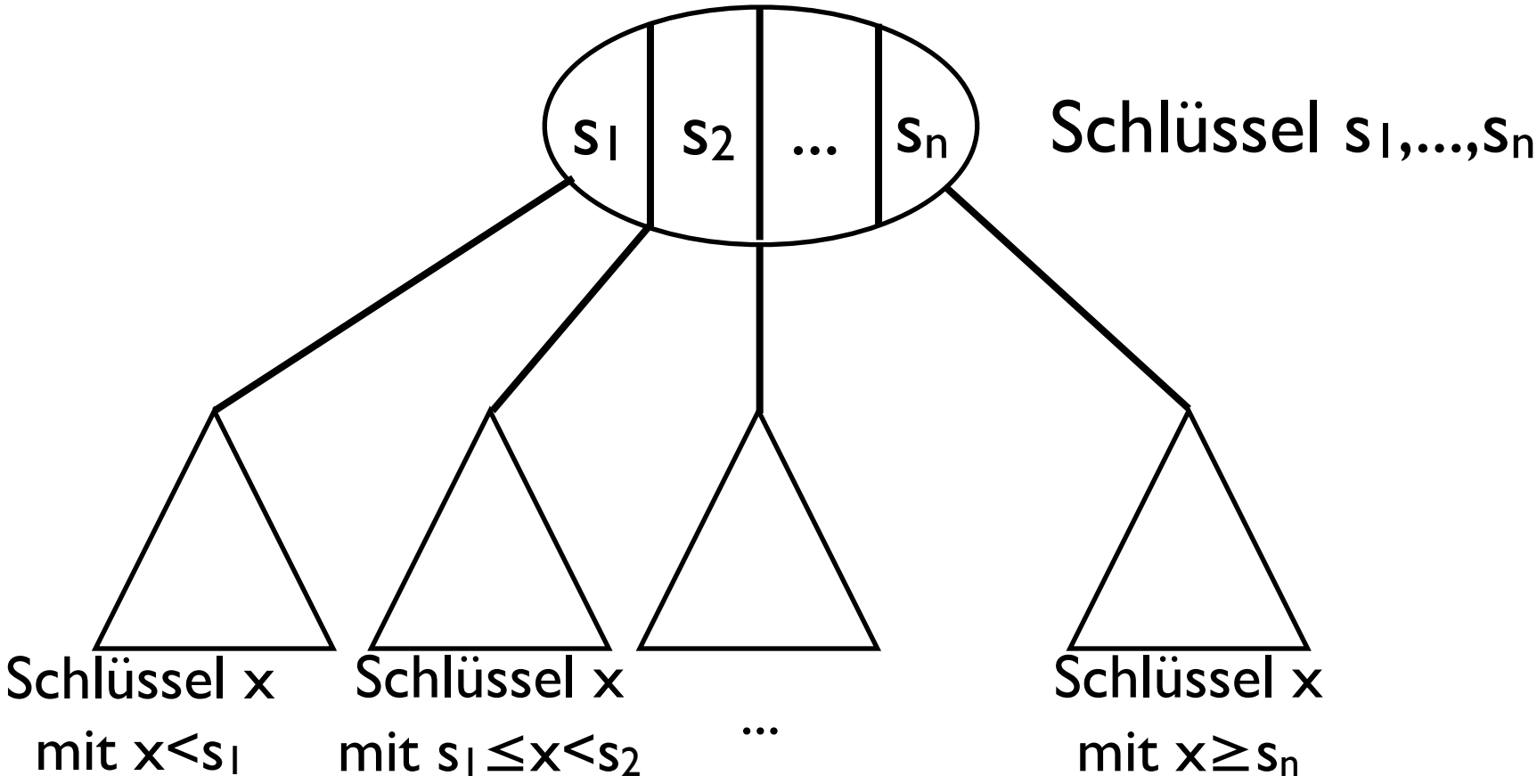
Suchen

Suchbäume - Struktur



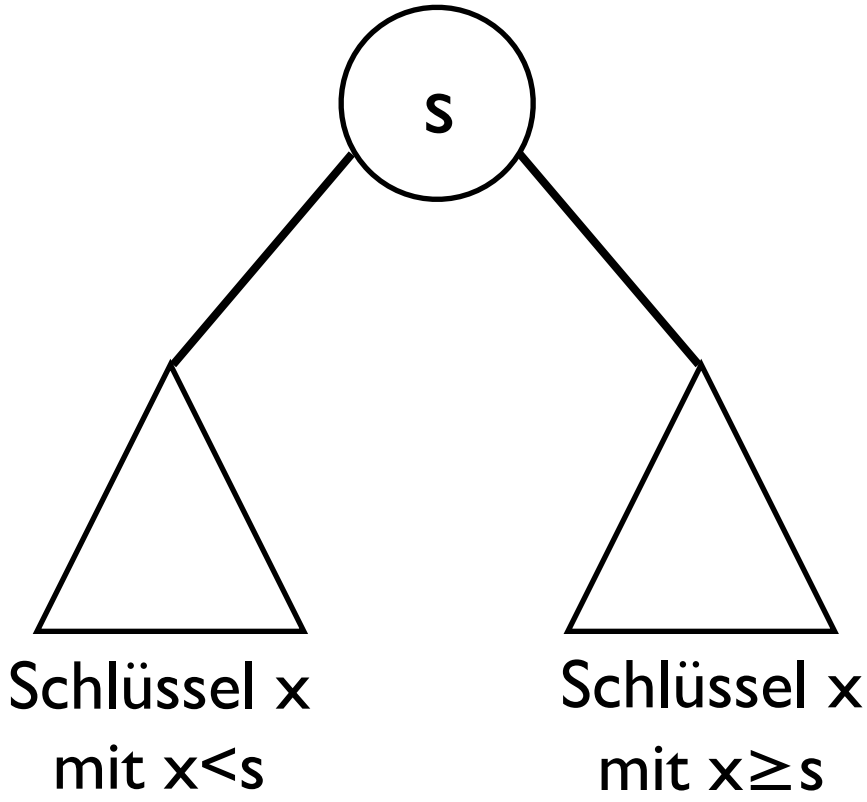
Suchen

Suchbäume - Struktur



Suchen
binäre Suchbäume

- **Internspeicher**: binäre Suchbäume
- je Knoten **ein** Schlüssel s
- Knoten im linken Teil-baum mit Schlüsseln ≤s
- Knoten im rechten Teilbaum mit Schlüsseln >s
- **Externspeicher**: Suchbäume mit n=128, 256 oder mehr Schlüsseln (**B-Bäume**)



Suchen
binäre Suchbäume - Definition

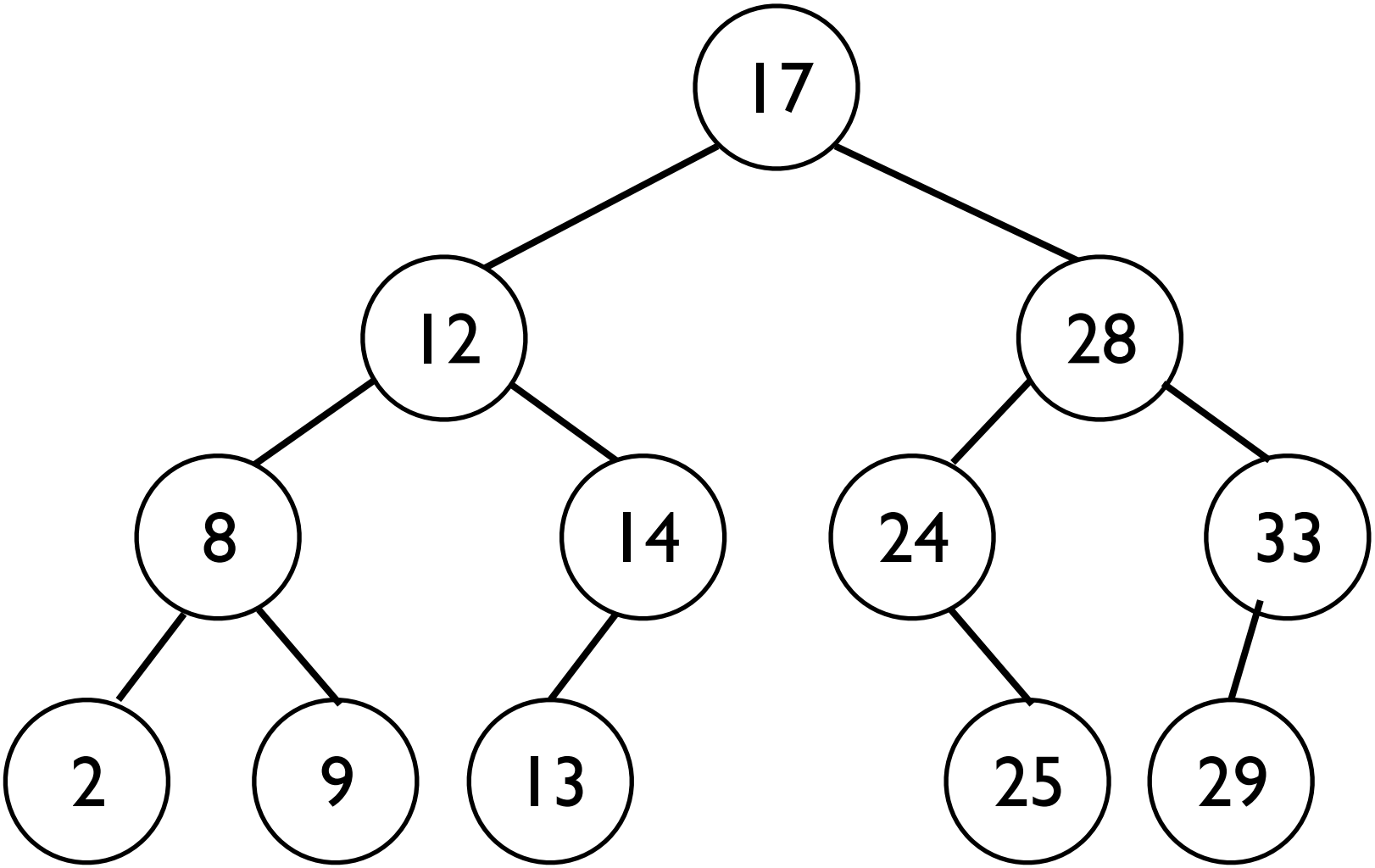
Definition

Sei M eine linear geordnete Menge. Ein binärer Suchbaum B=(K,A,S) für M ist ein geordneter, binärer Baum B=(K,A) mit einer Abb. S: K→M, so daß für jeden Knoten k∈K gilt:

1. S(u)≤S(k) für alle Knoten u im linken Teilbaum von k
2. S(u)>S(k) für alle Knoten u im rechten Teilbaum von k.

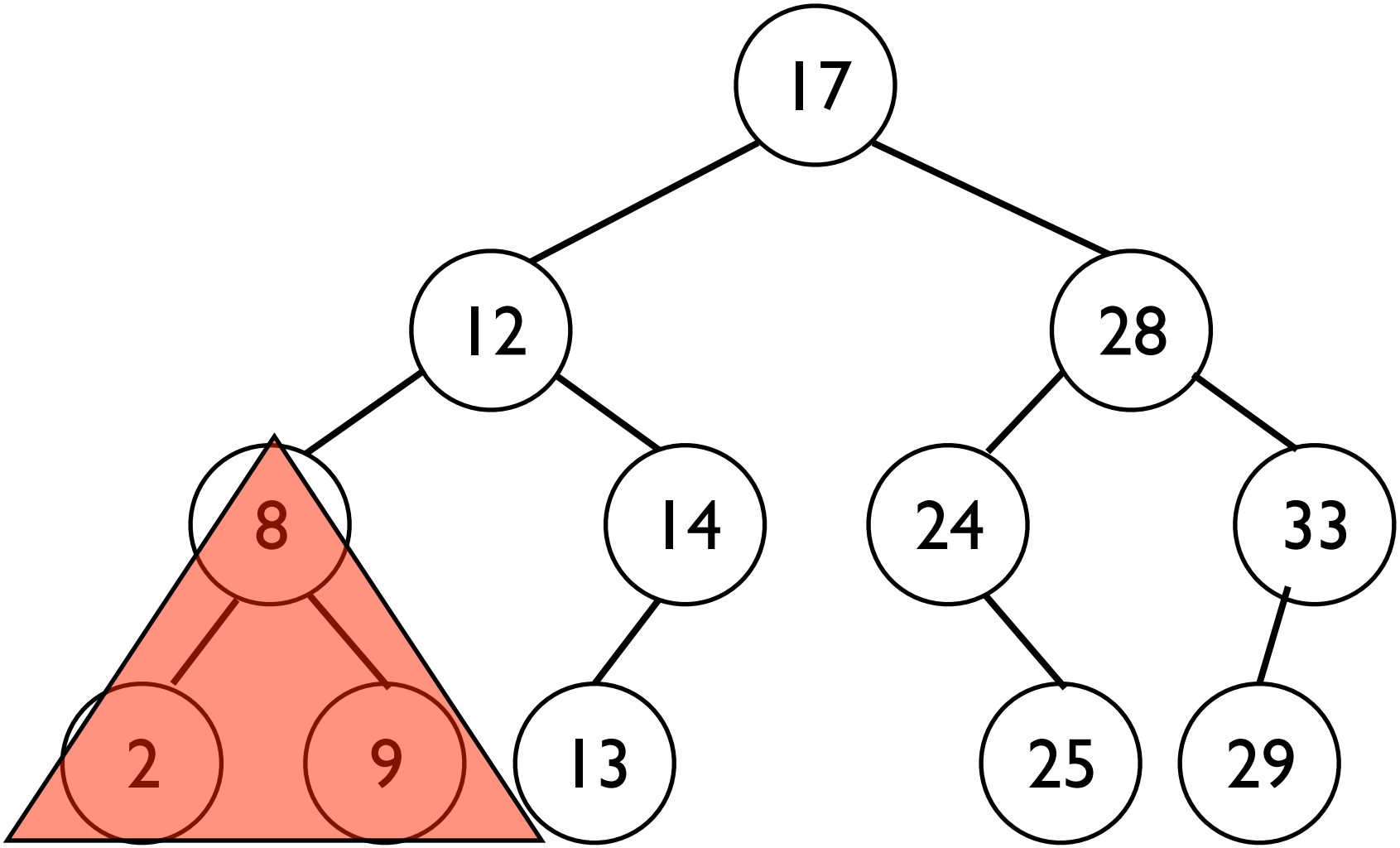
Suchen

binäre Suchbäume - Beispiel



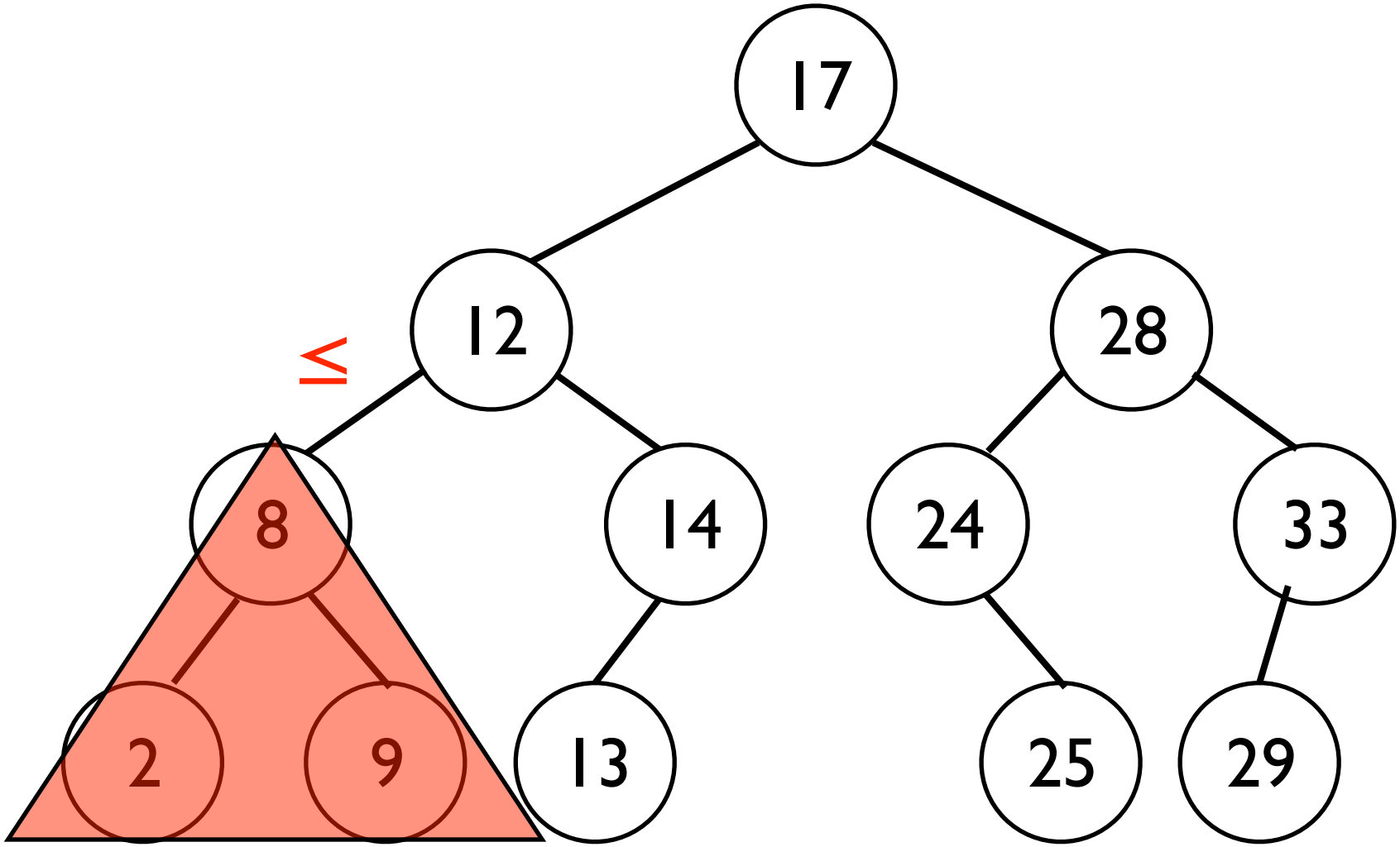
Suchen

binäre Suchbäume - Beispiel



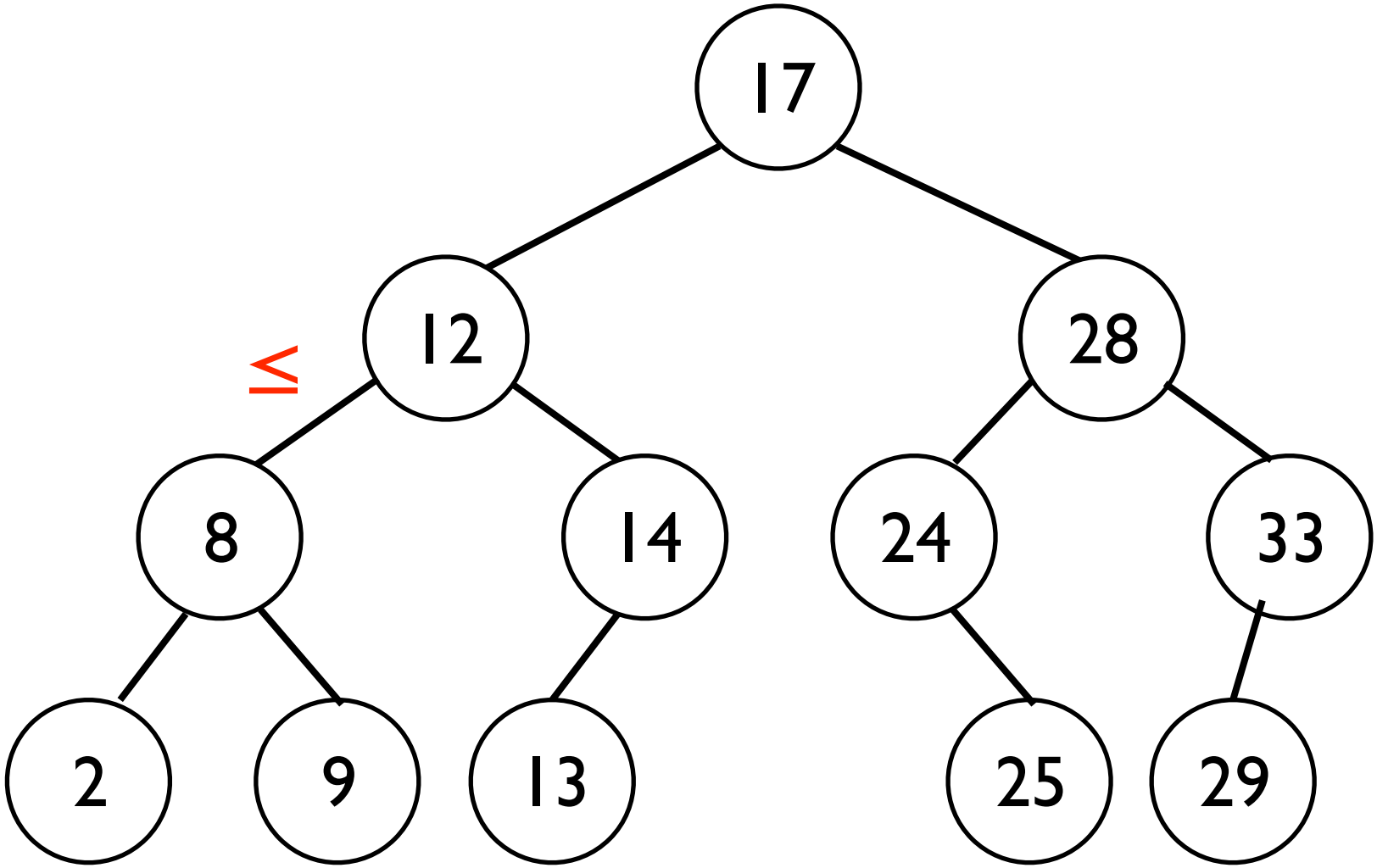
Suchen

binäre Suchbäume - Beispiel



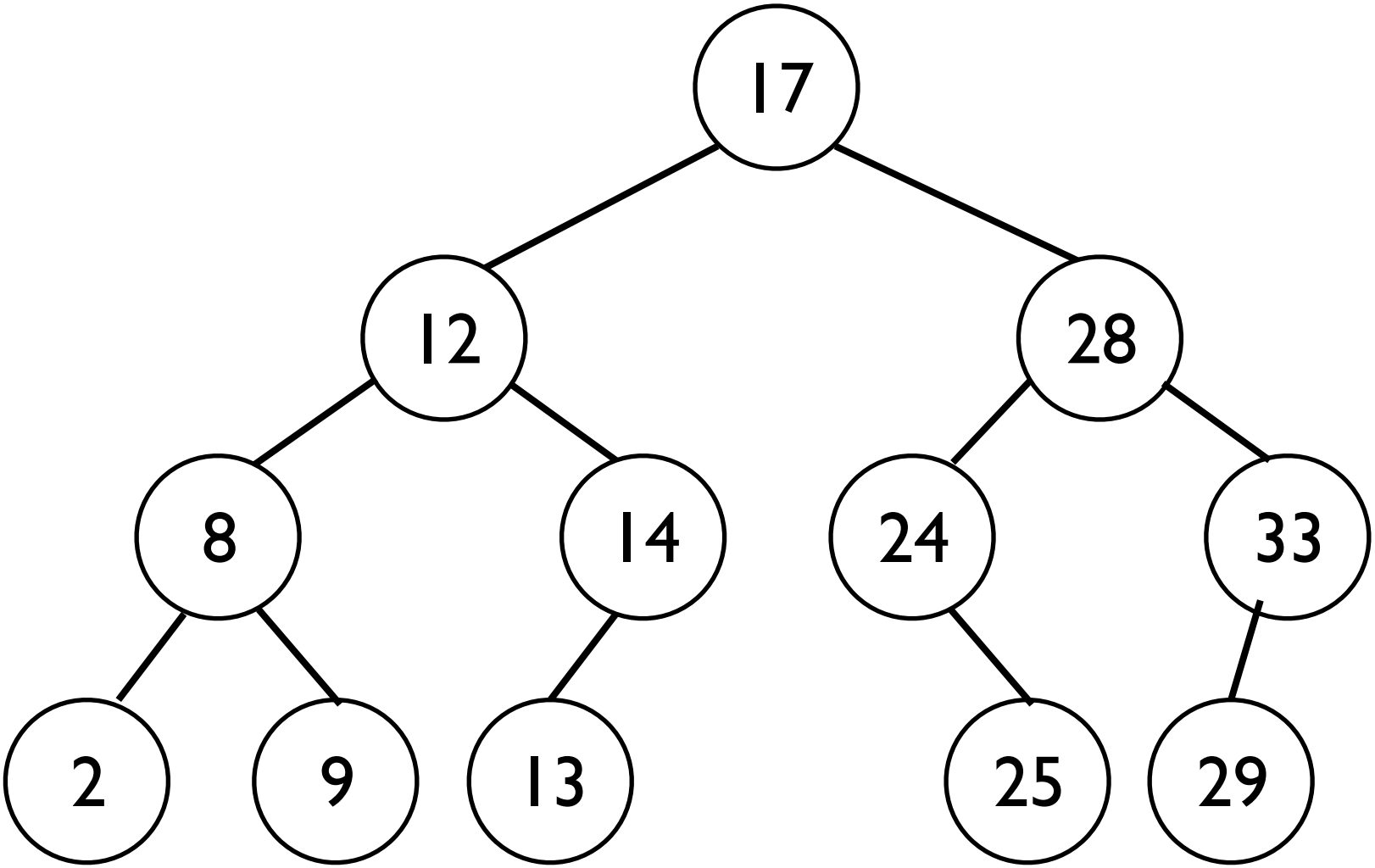
Suchen

binäre Suchbäume - Beispiel



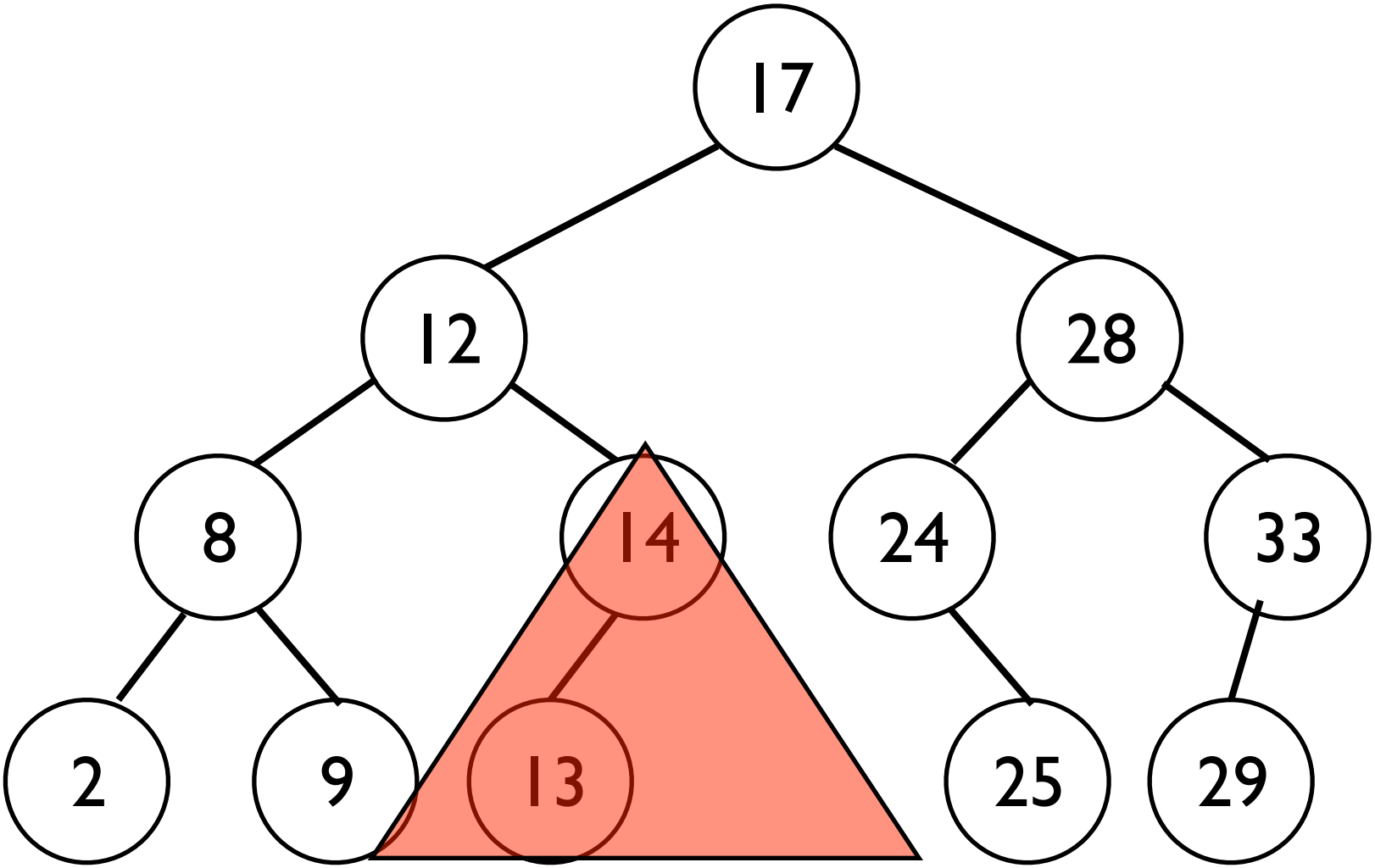
Suchen

binäre Suchbäume - Beispiel



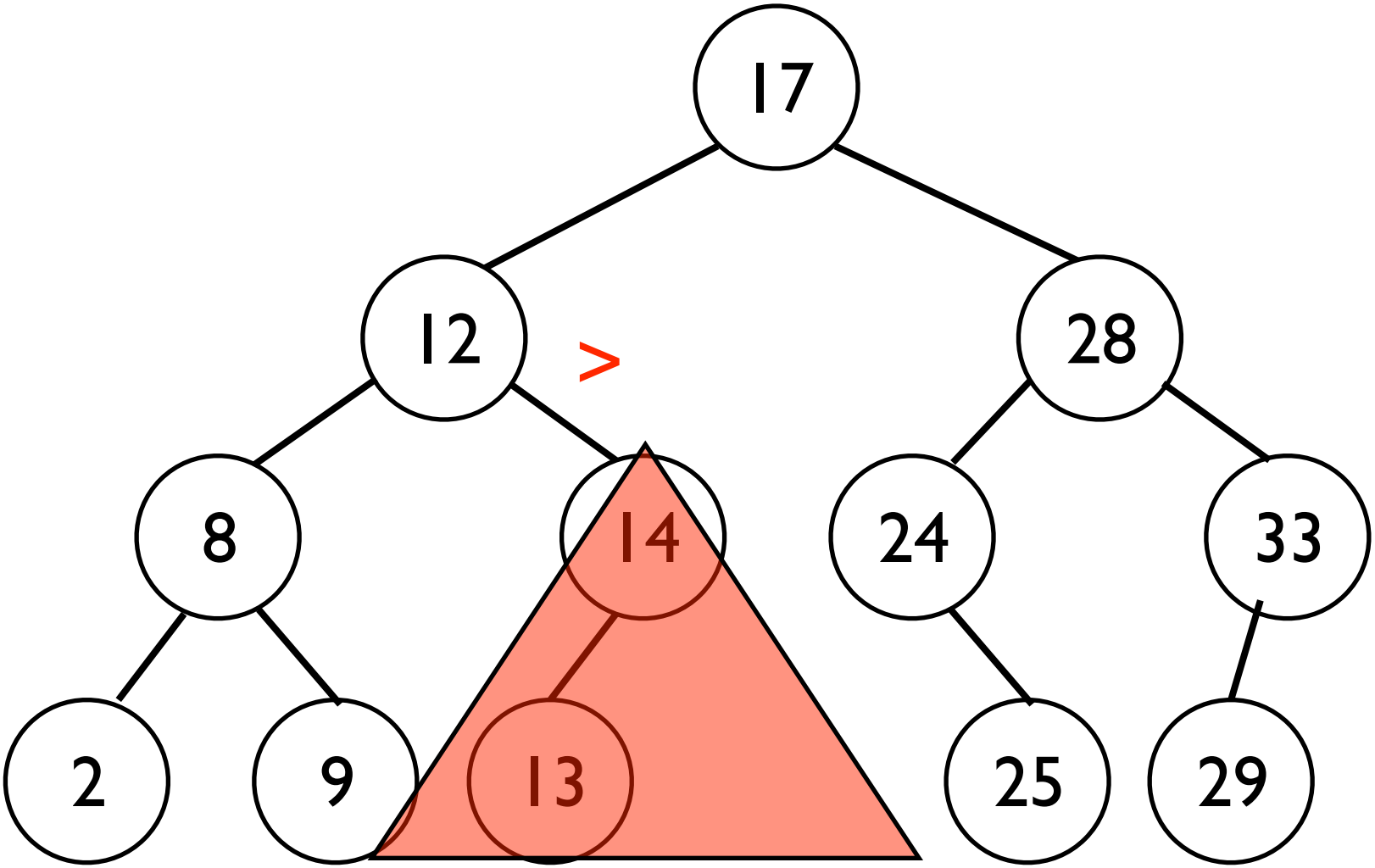
Suchen

binäre Suchbäume - Beispiel



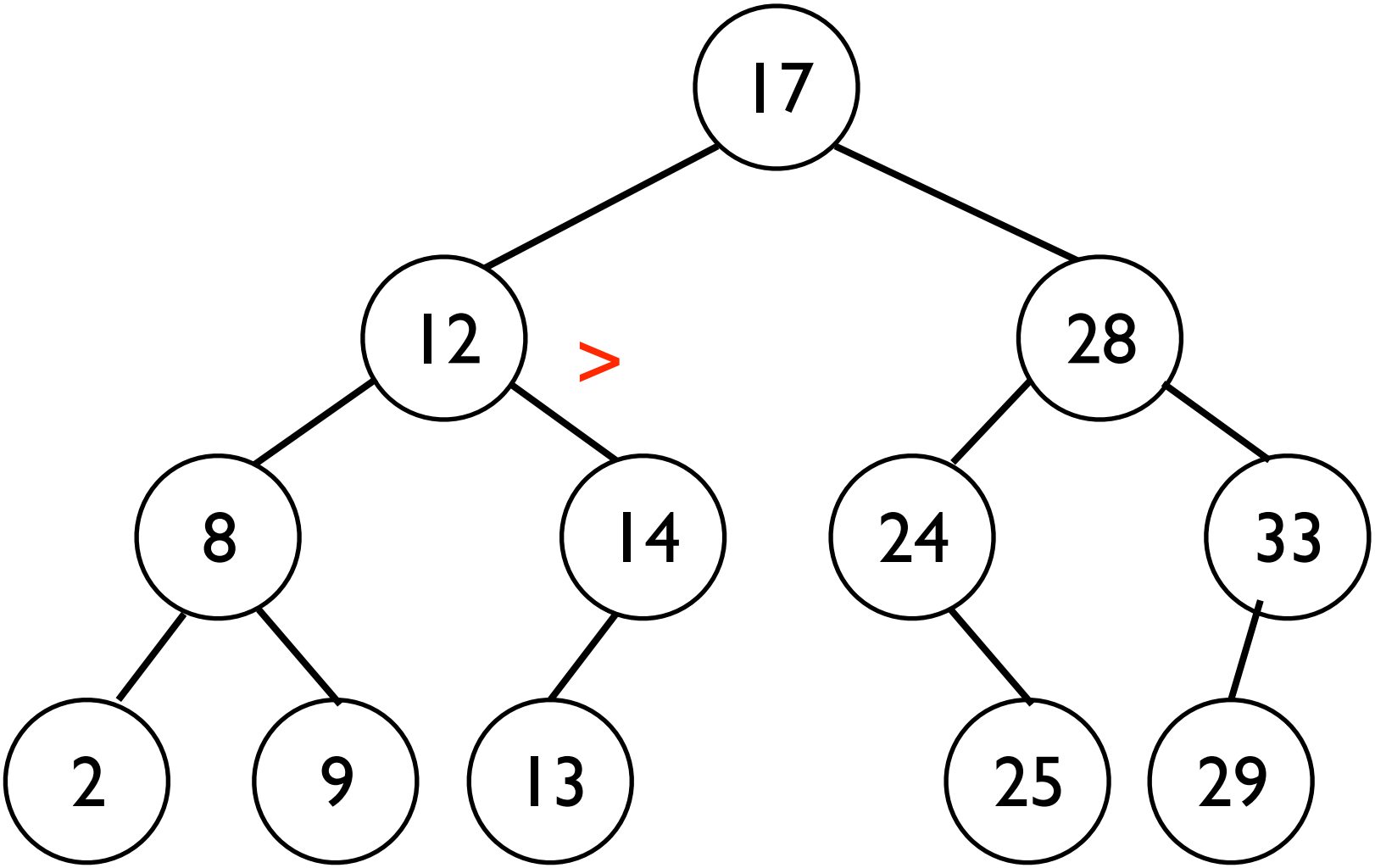
Suchen

binäre Suchbäume - Beispiel



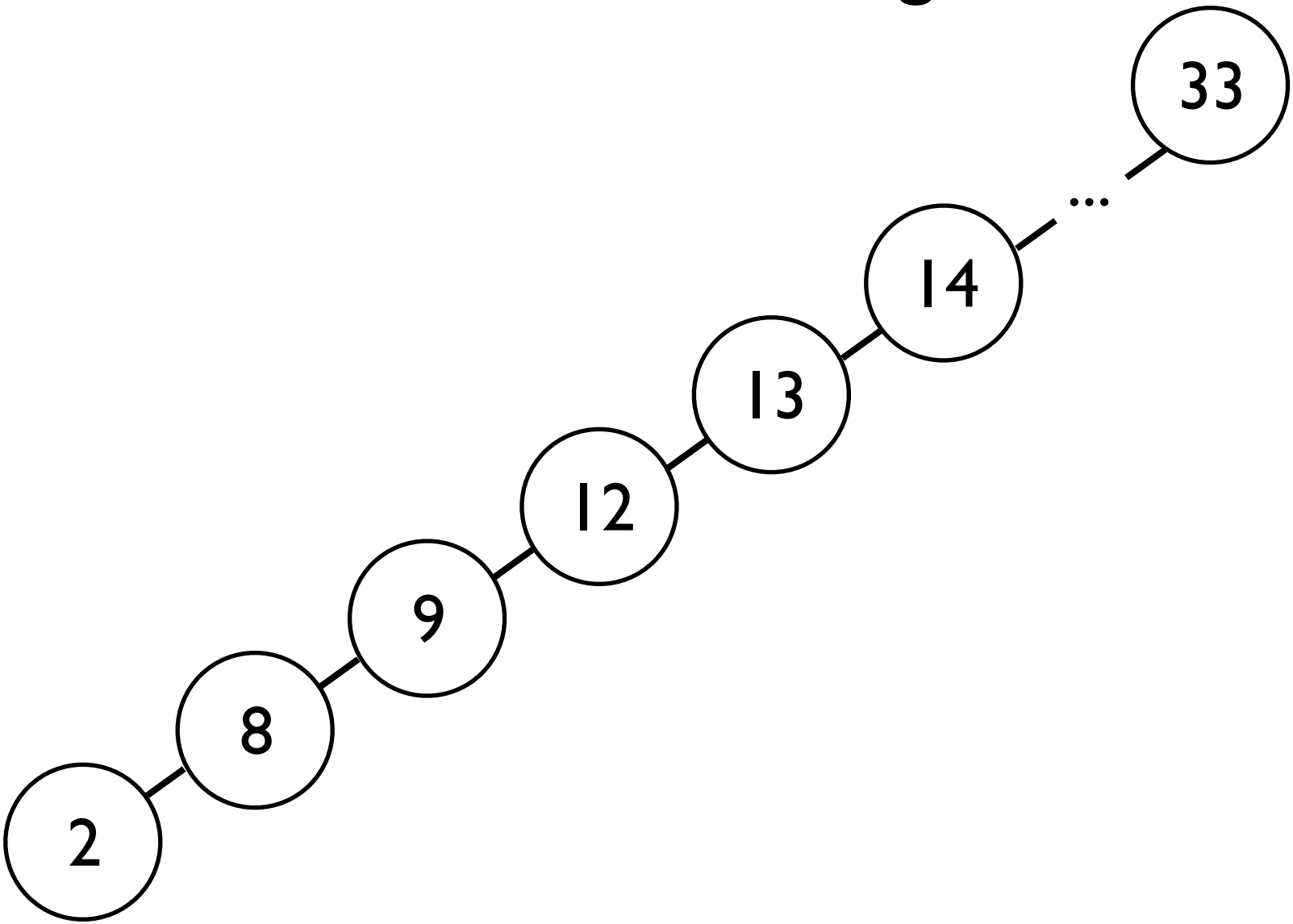
Suchen

binäre Suchbäume - Beispiel



Suchen

binärer Suchbäume - degeneriert



Suchen
binäre Suchbäume - Operation Suchen

Suchen nach Schlüssel s

- Beginn an der Wurzel s_0
- Vergleiche s mit s_0
- $s \leq s_0 \Rightarrow$ *rekursiv* weitersuchen im linken

  Teilbaum
- $s > s_0 \Rightarrow$ *rekursiv* weitersuchen im rechten

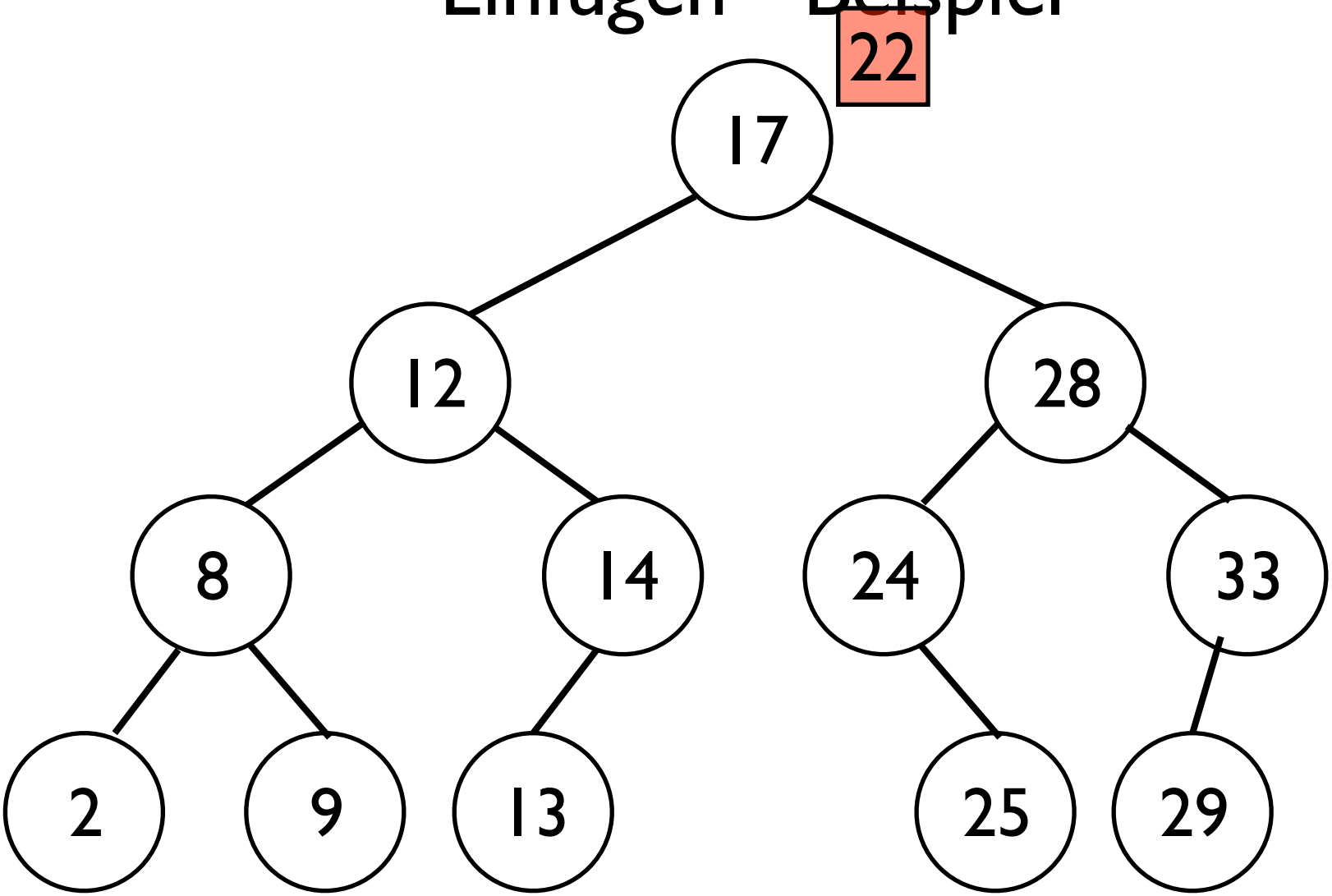
  Teilbaum
- Suche erfolglos, falls Sohn, zu dem man verzweigen soll, nicht existiert

Suchen
binäre Suchbäume - Einfügen

- Neue Knoten s werden grundsätzlich als Blatt eingefügt.
- Vorgehensweise:
  1. Suche zunächst s
  2. dabei gelangt man an ein Blatt, zu dem der Nachfolger nicht existiert
  3. füge s als diesen Nachfolger ein

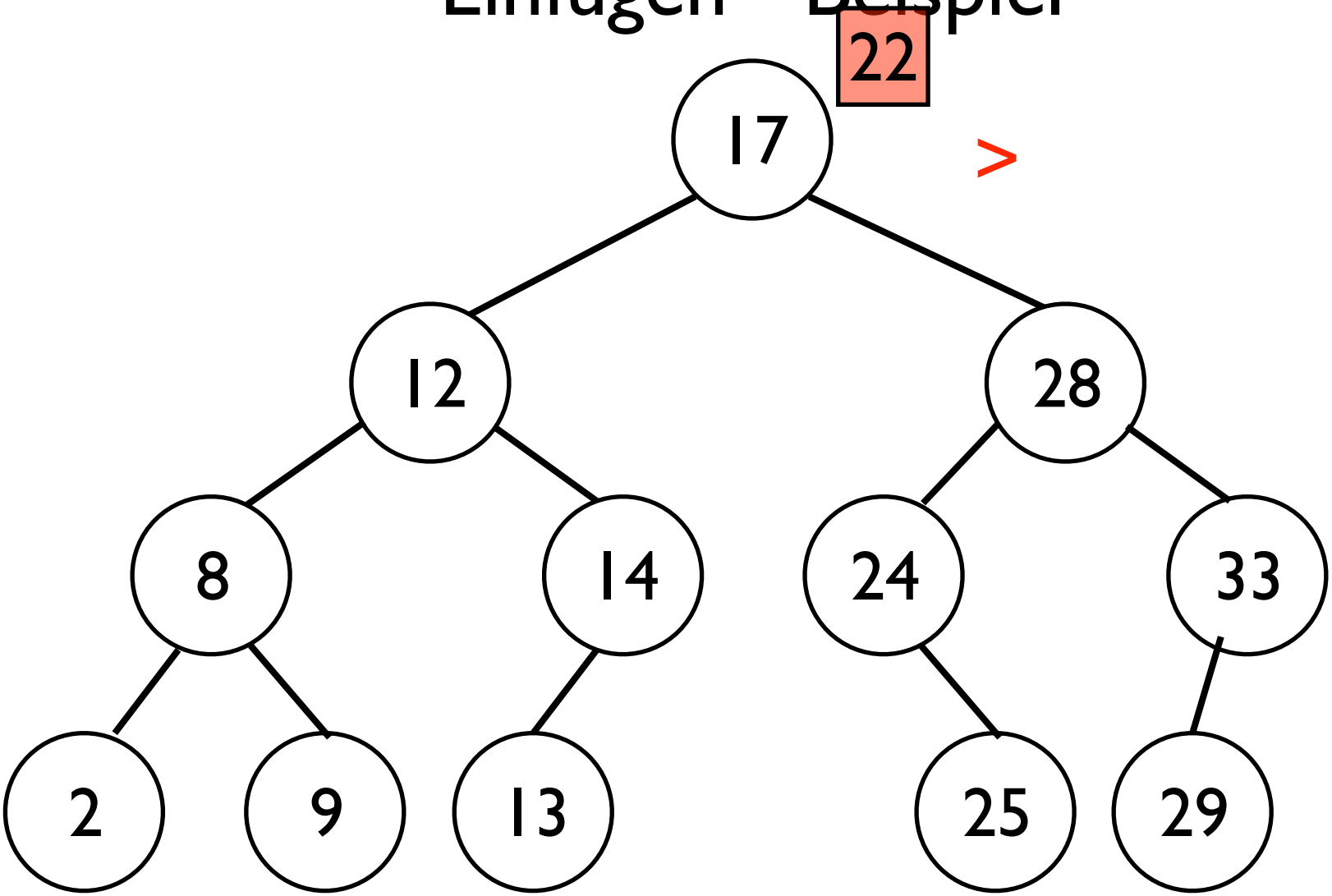
Suchen

Einfügen - Beispiel



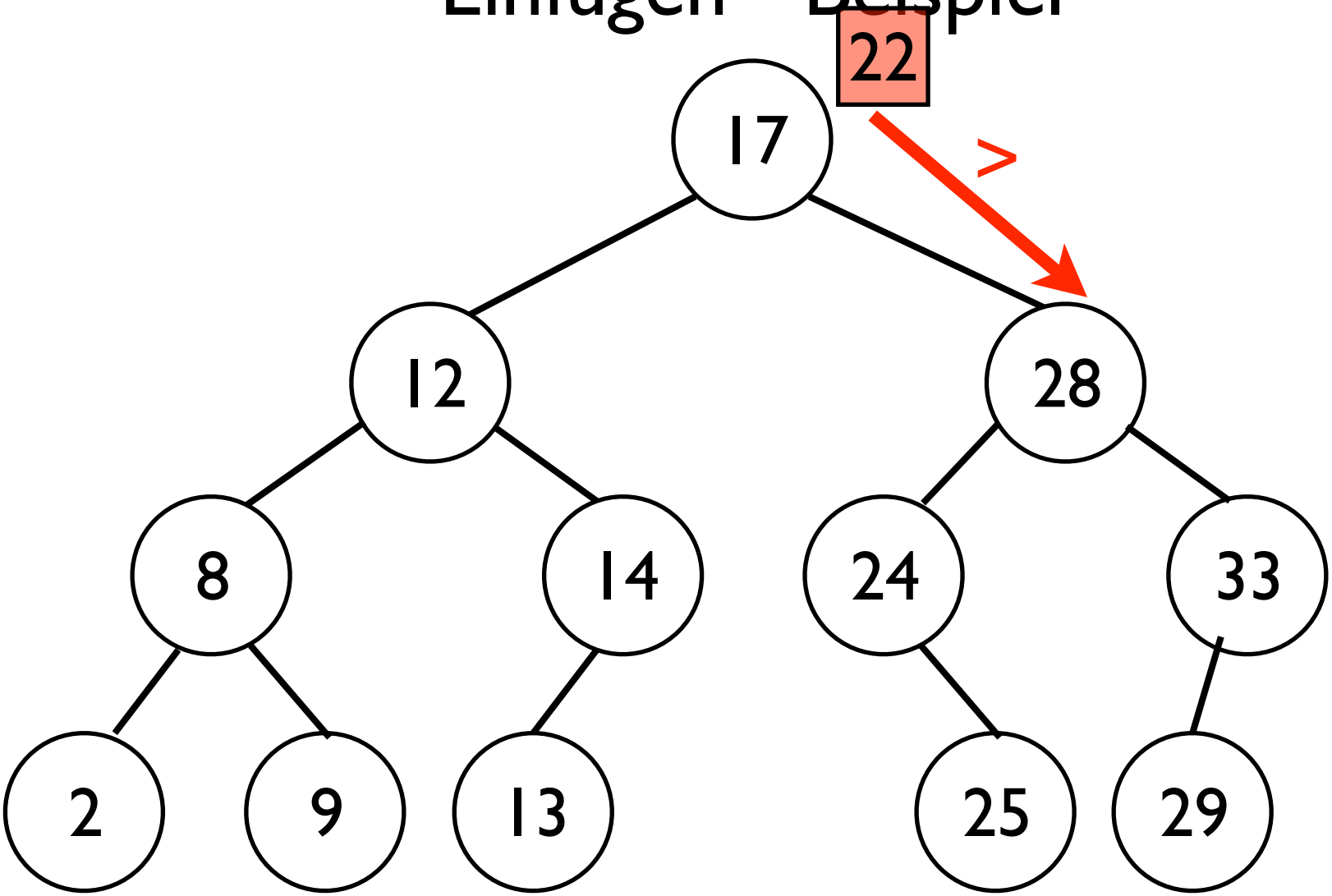
Suchen

Einfügen - Beispiel



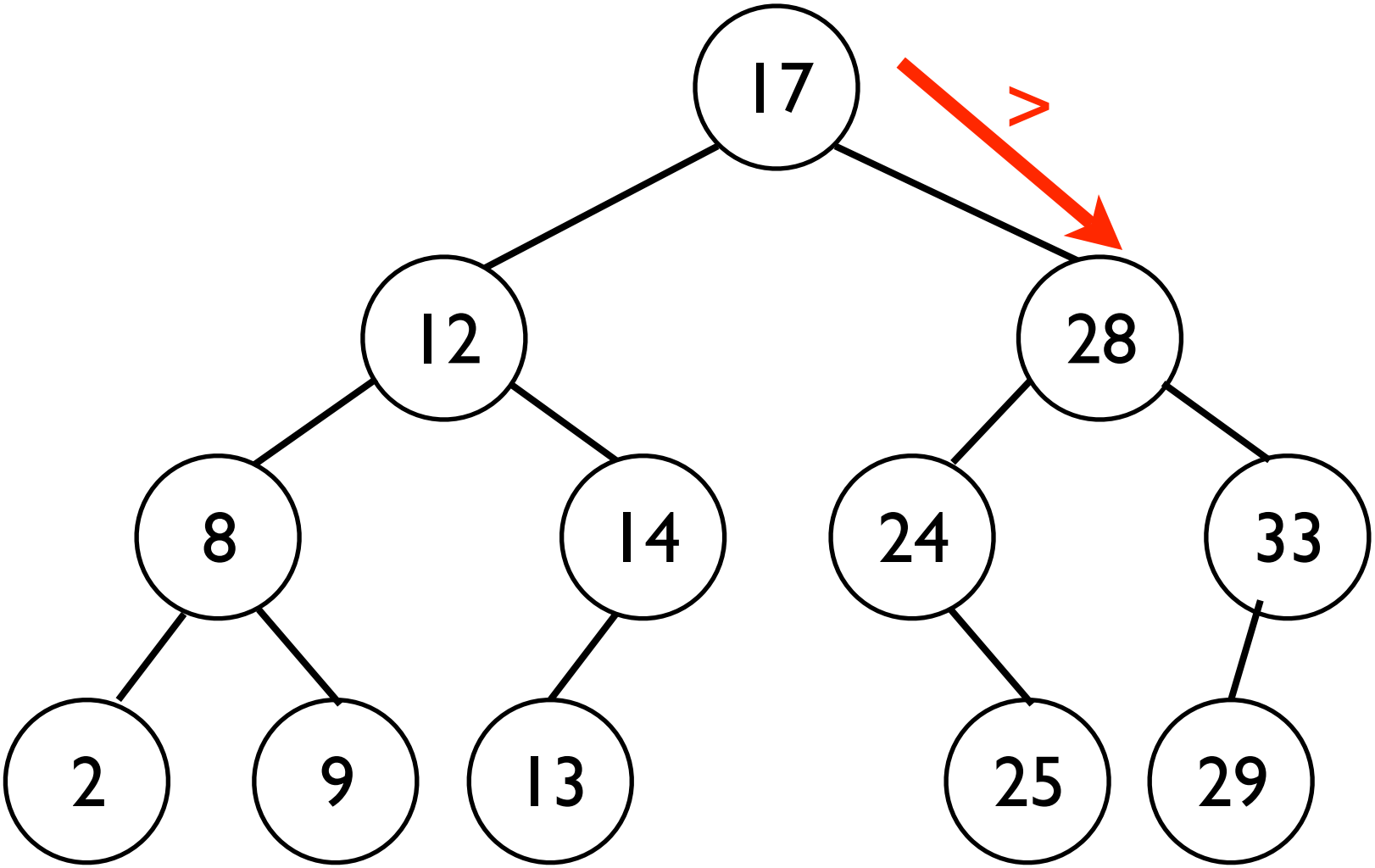
Suchen

Einfügen - Beispiel



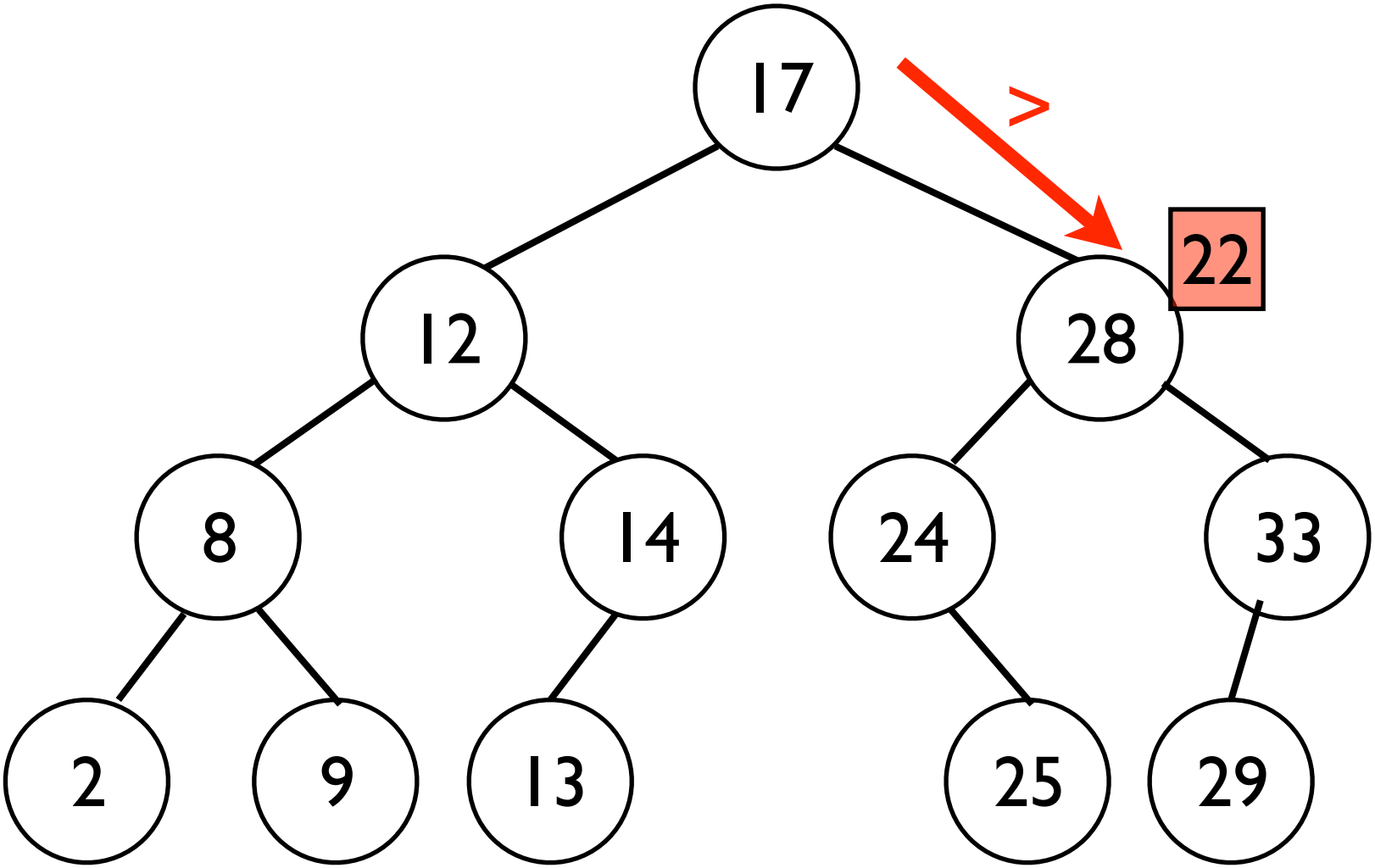
Suchen

Einfügen - Beispiel



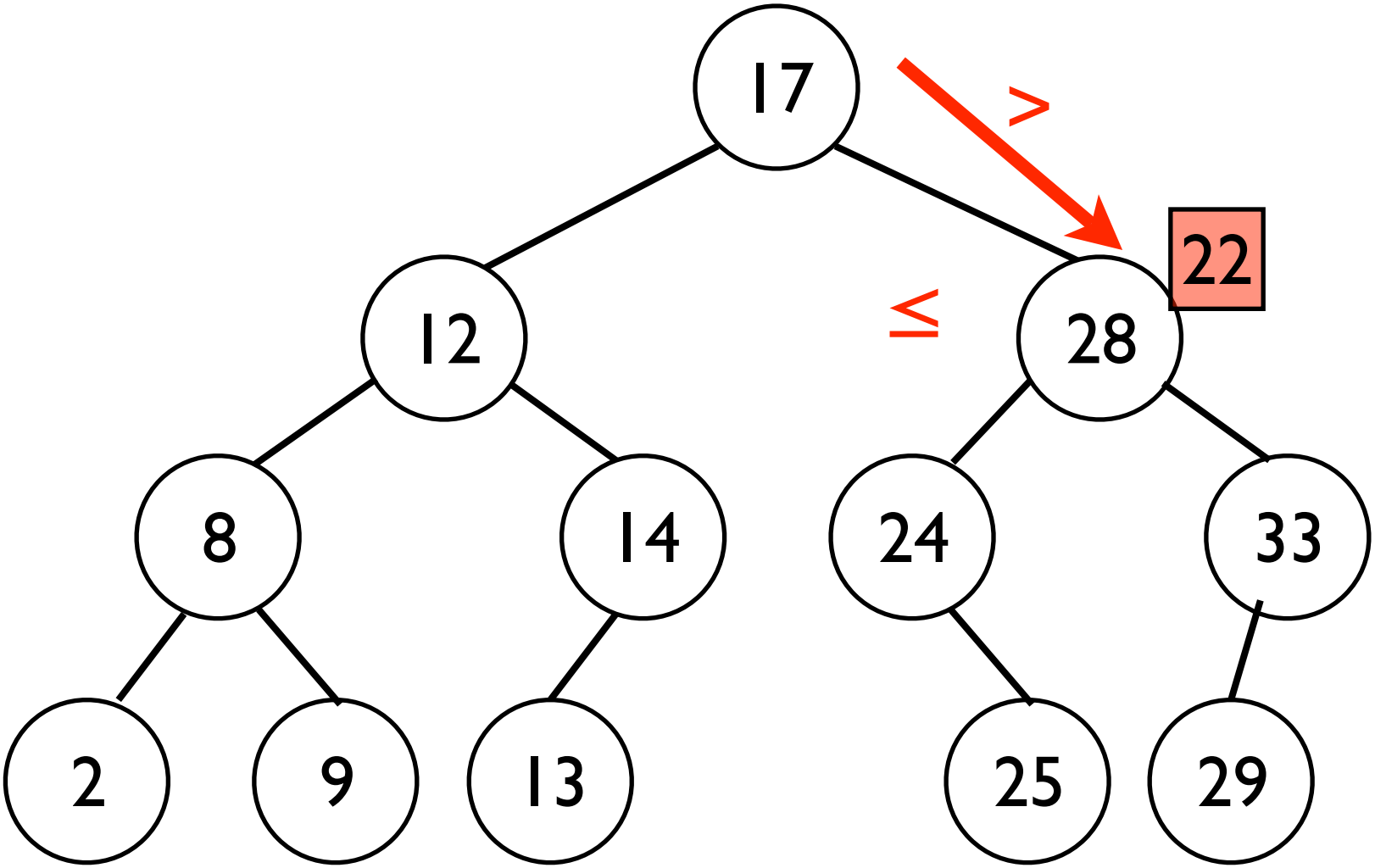
Suchen

Einfügen - Beispiel



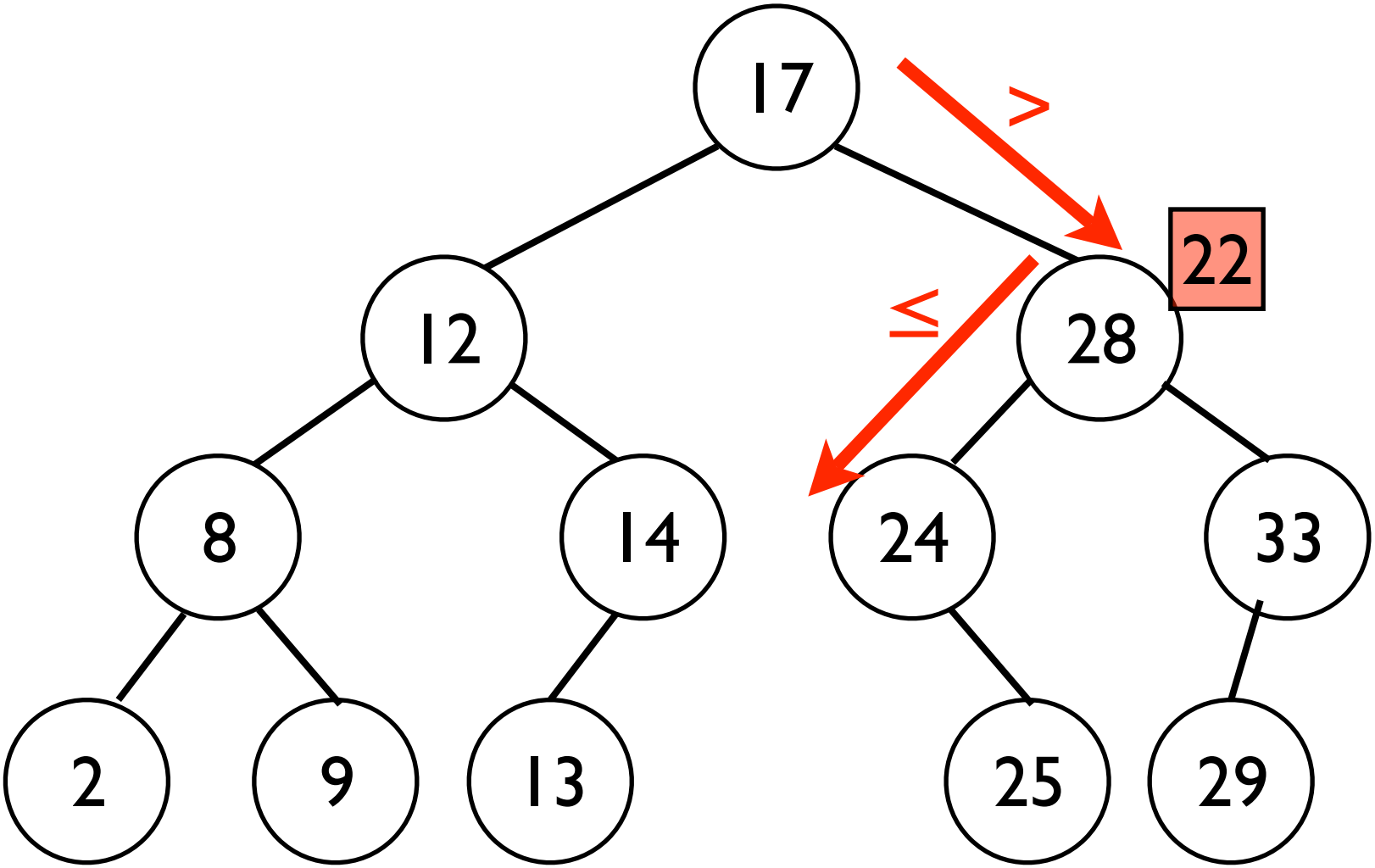
Suchen

Einfügen - Beispiel



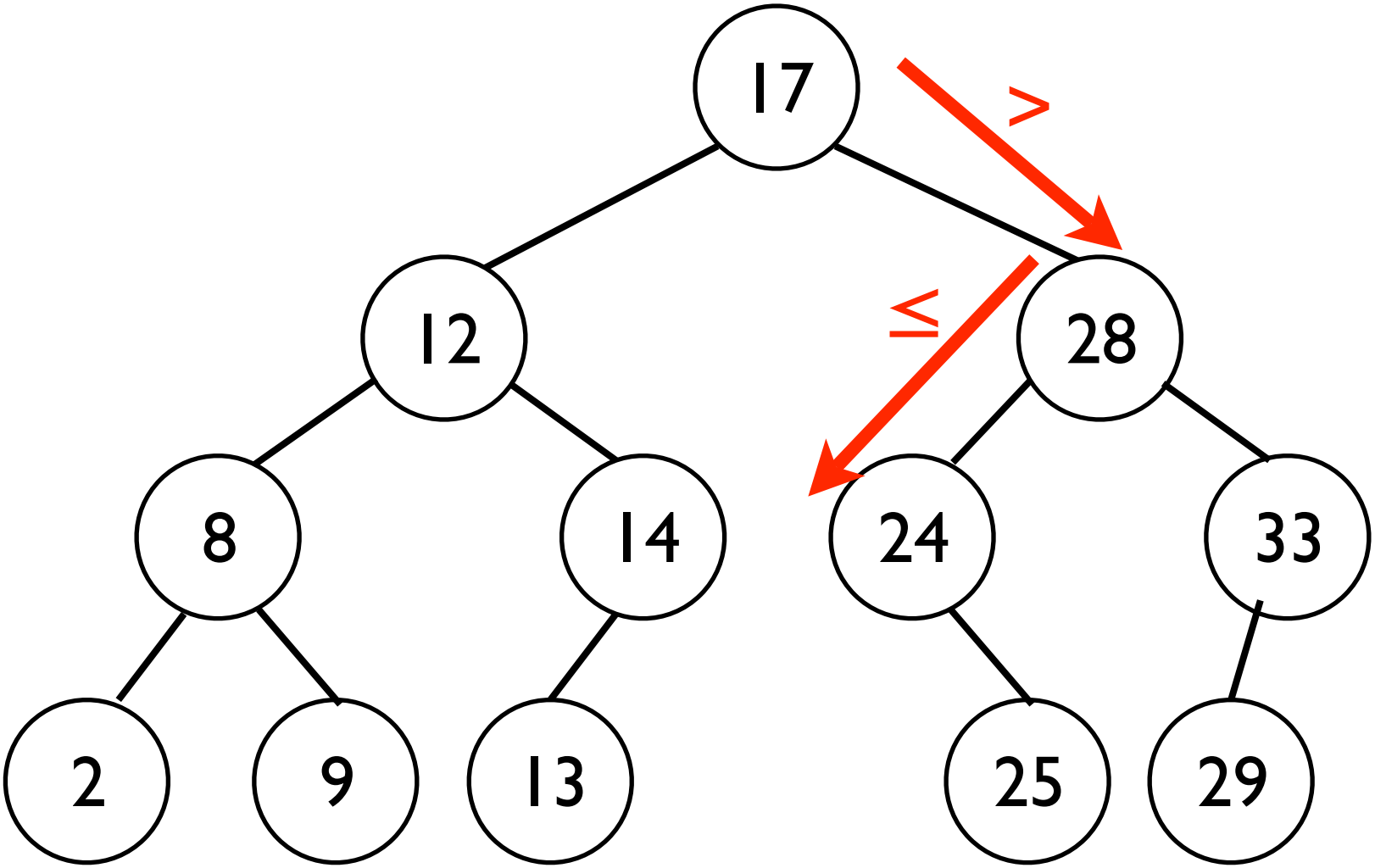
Suchen

Einfügen - Beispiel



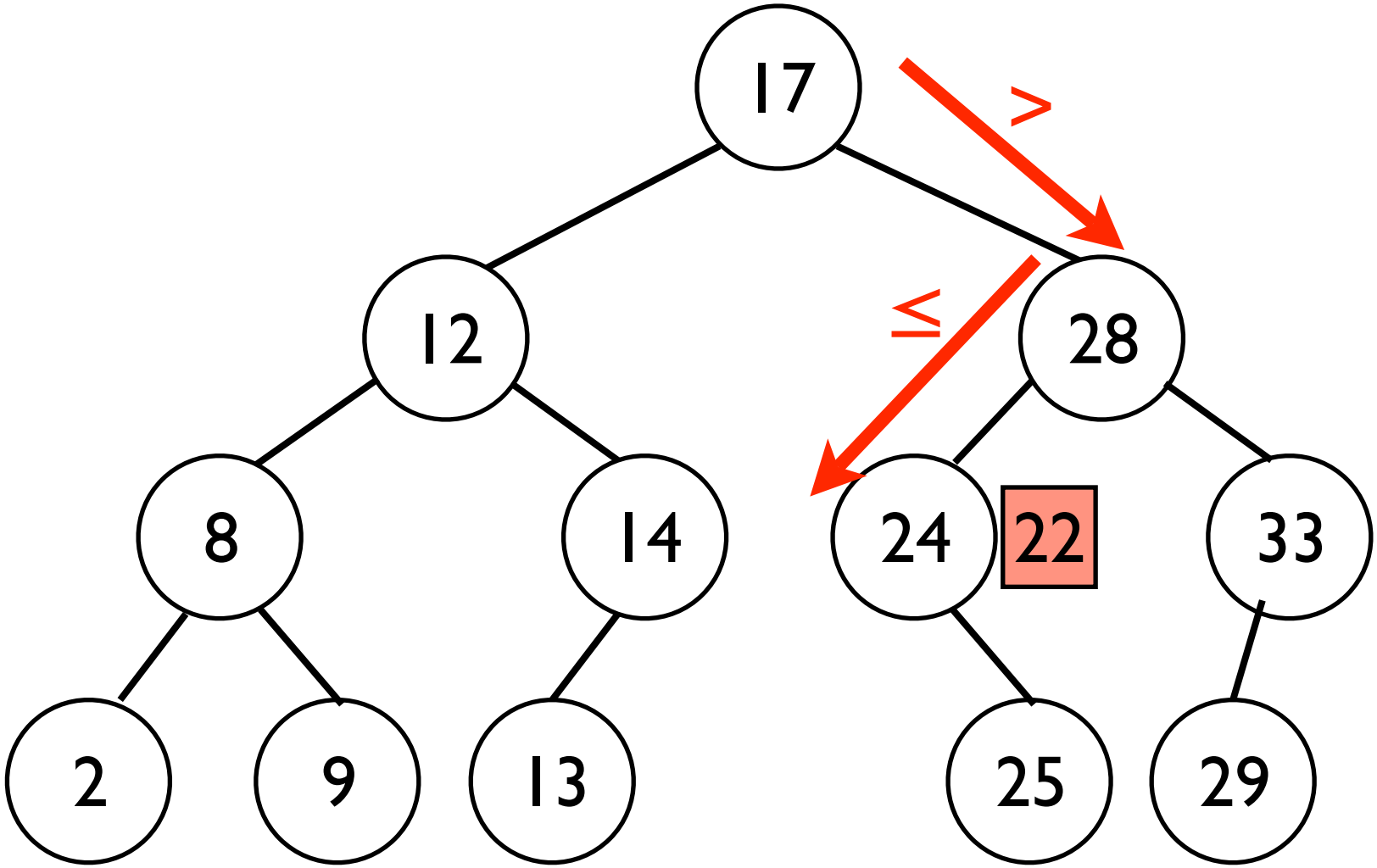
Suchen

Einfügen - Beispiel



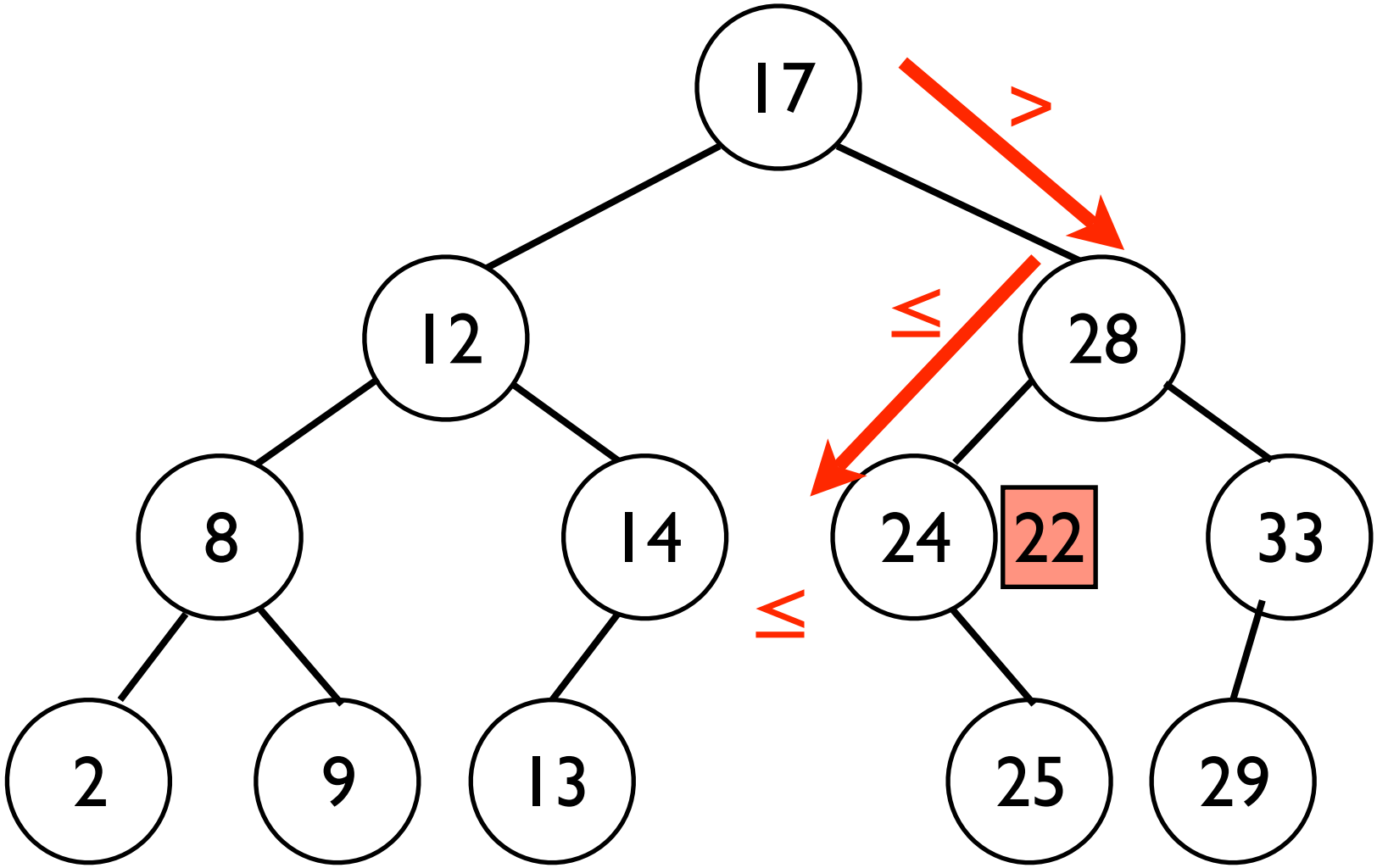
Suchen

Einfügen - Beispiel



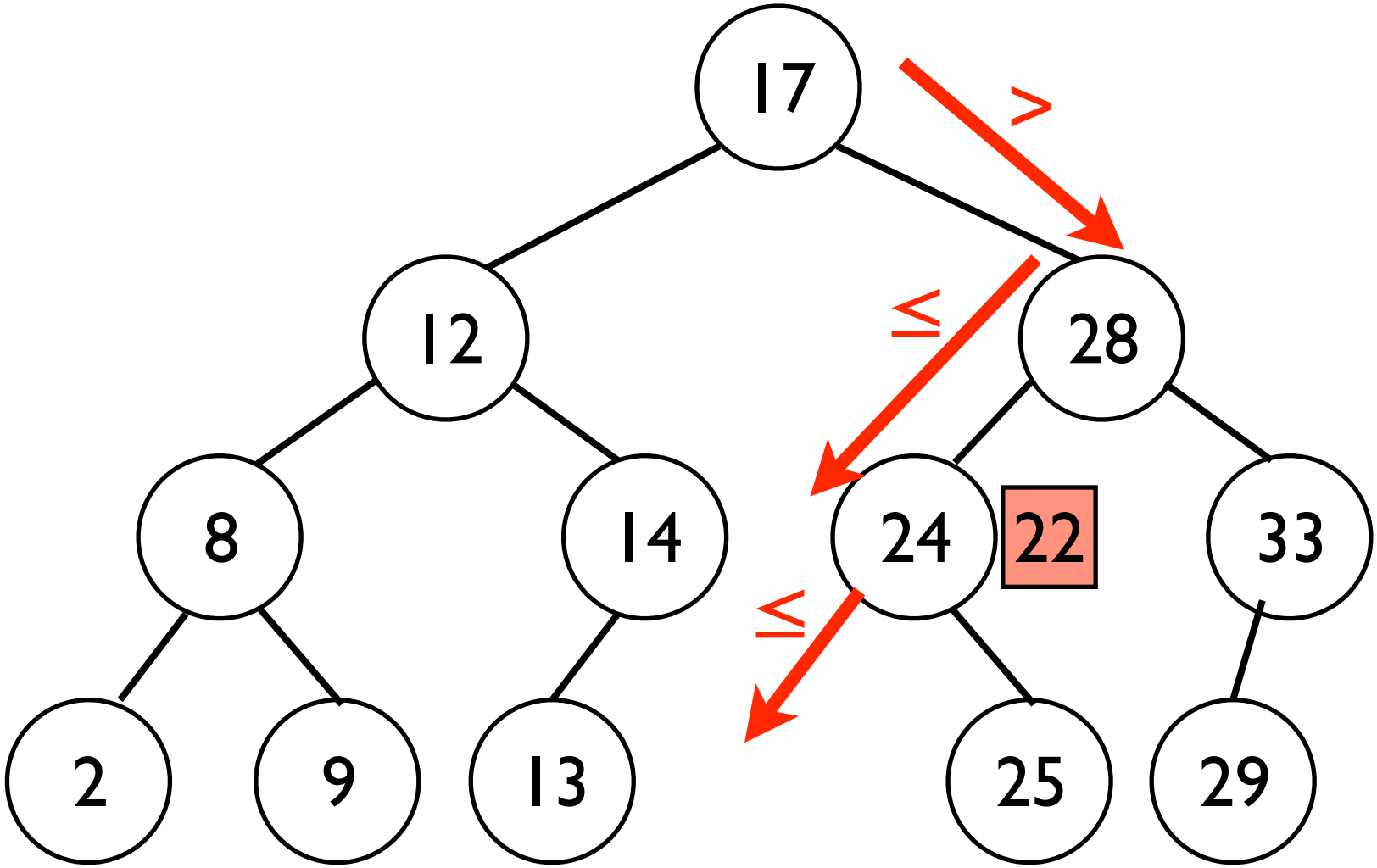
Suchen

Einfügen - Beispiel



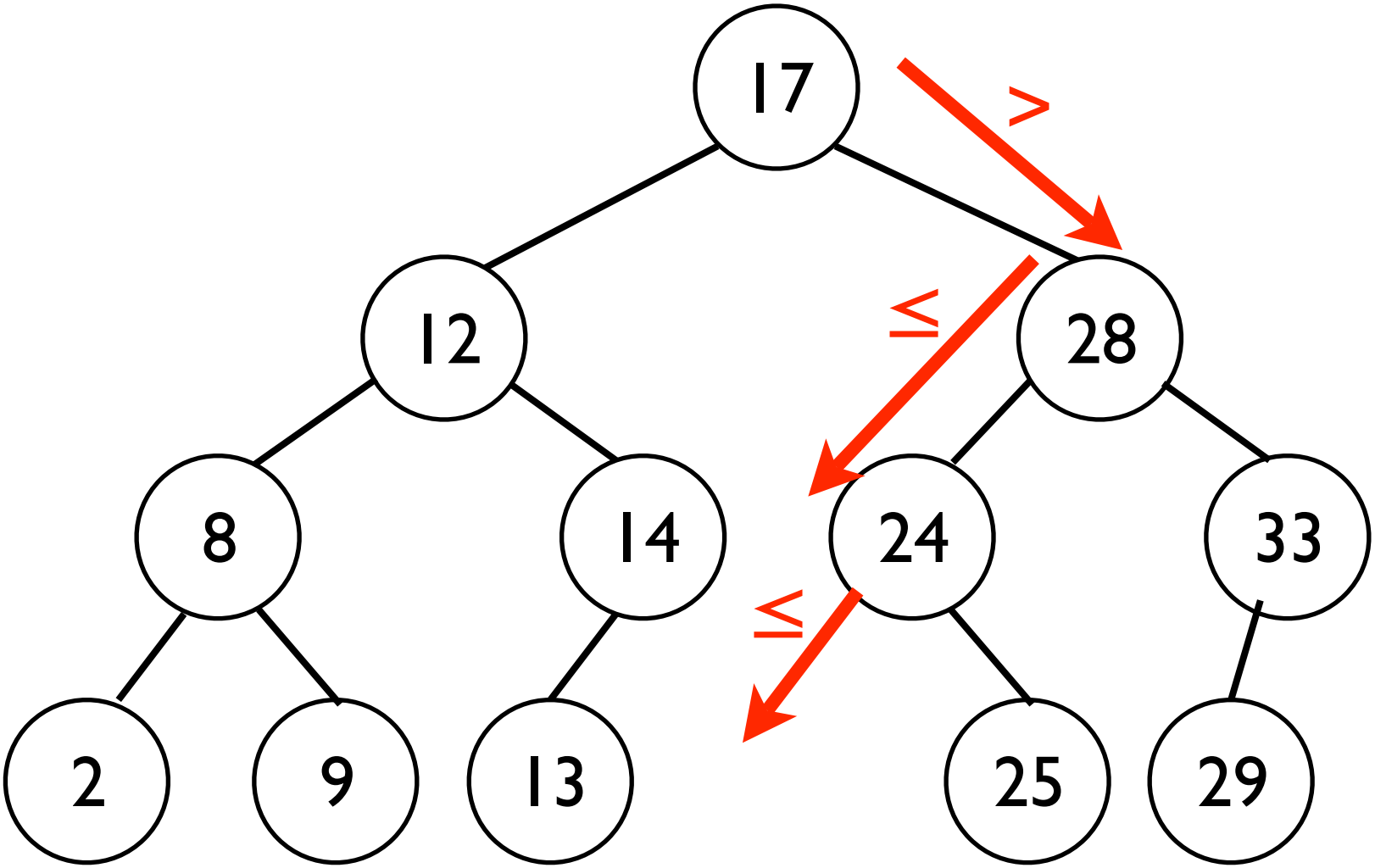
Suchen

Einfügen - Beispiel



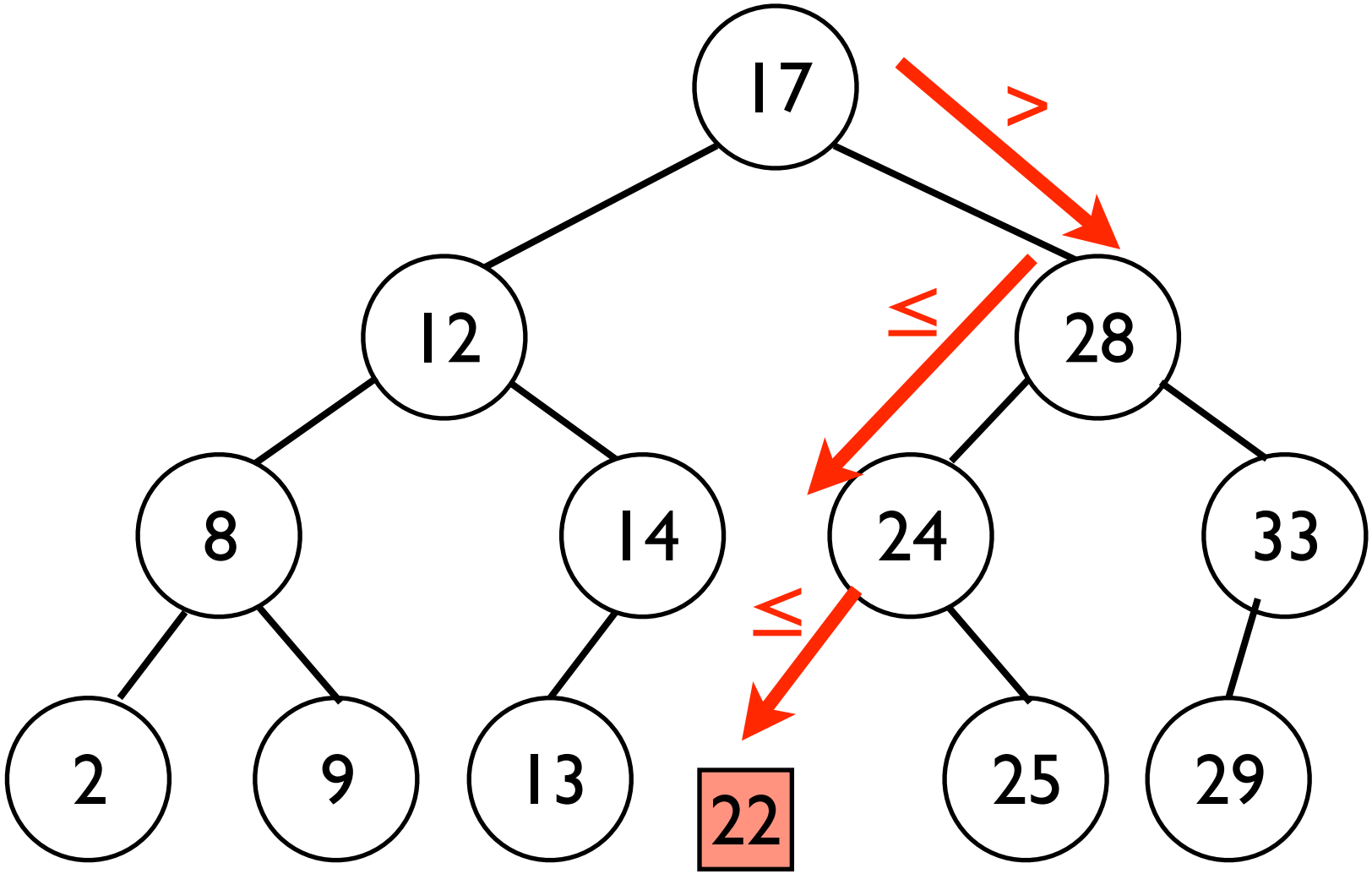
Suchen

Einfügen - Beispiel



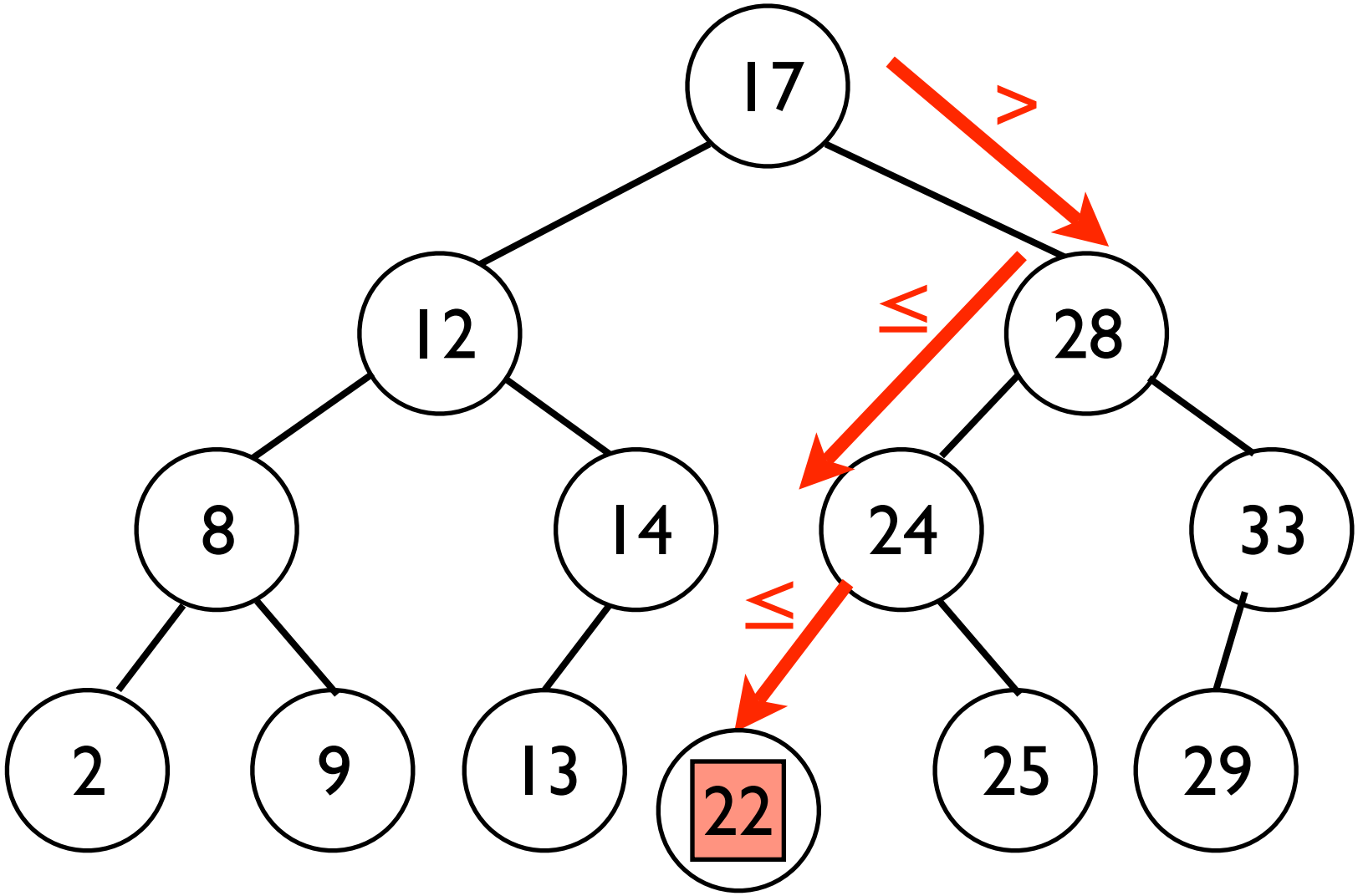
Suchen

Einfügen - Beispiel

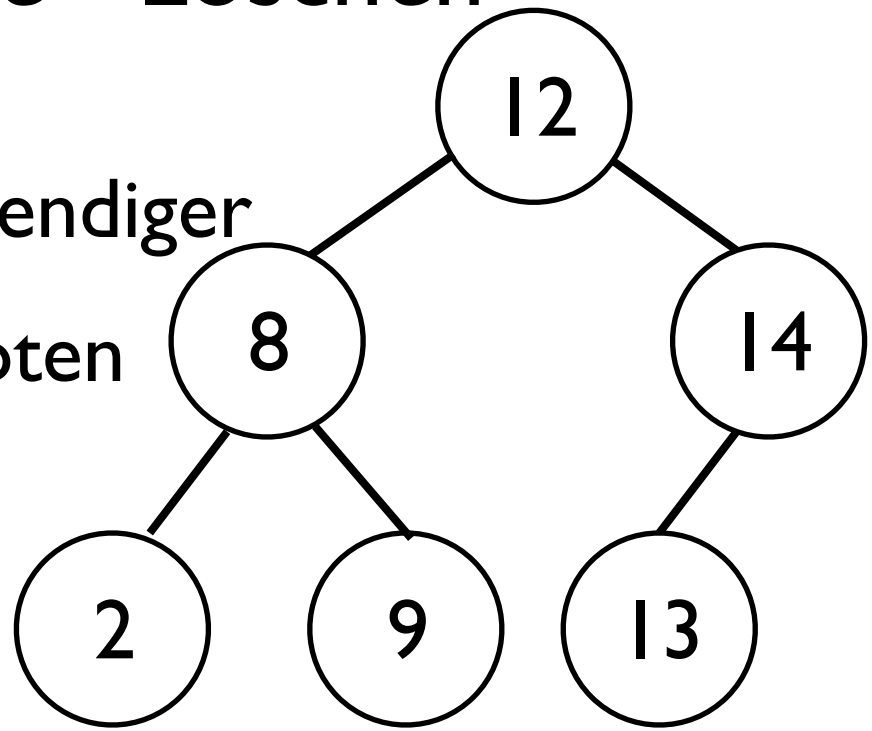


Suchen

Einfügen - Beispiel



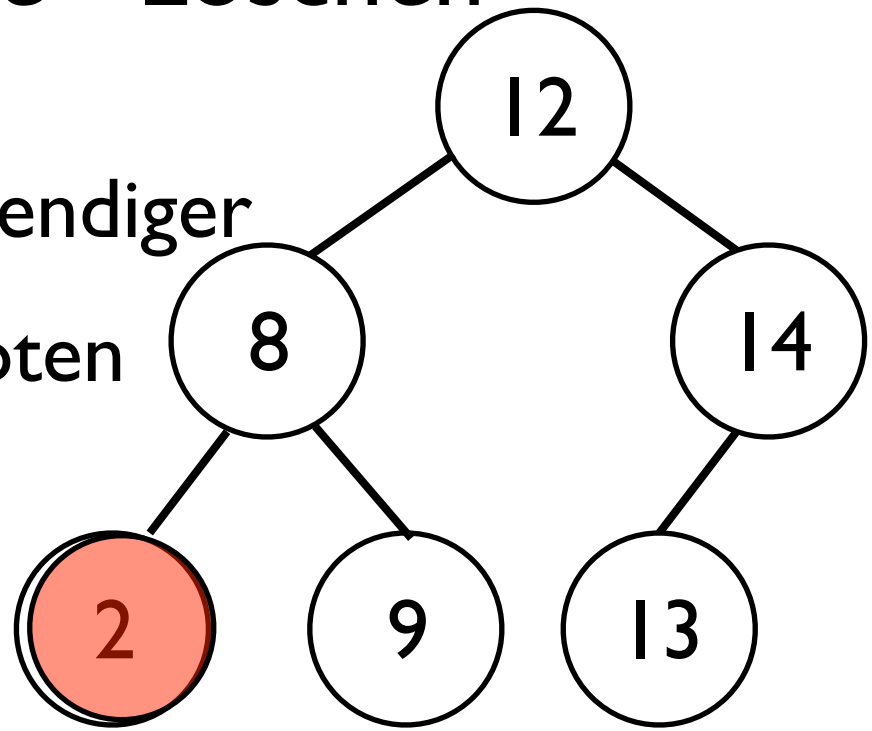
Suchen
binäre Suchbäume - Löschen



- Bis auf Sonderfälle aufwendiger
- k sei zu löschender Knoten
- Sonderfälle:

1. k ist Blatt. Lösche k
2. k hat keinen rechten Sohn: Lösche k; neuer Sohn des Vaters von k wird der linke Sohn von k
3. k hat keinen linken Sohn: analog.

Suchen
binäre Suchbäume - Löschen

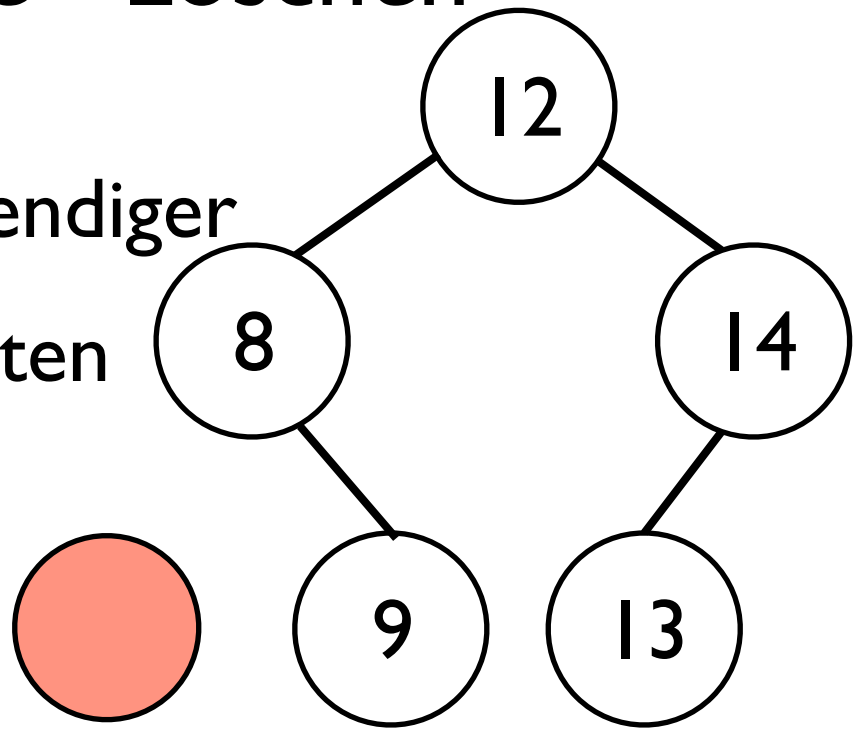


- Bis auf Sonderfälle aufwendiger
- k sei zu löschender Knoten
- Sonderfälle:

1. k ist Blatt. Lösche k
2. k hat keinen rechten Sohn: Lösche k; neuer Sohn des Vaters von k wird der linke Sohn von k
3. k hat keinen linken Sohn: analog.

Suchen

binäre Suchbäume - Löschen

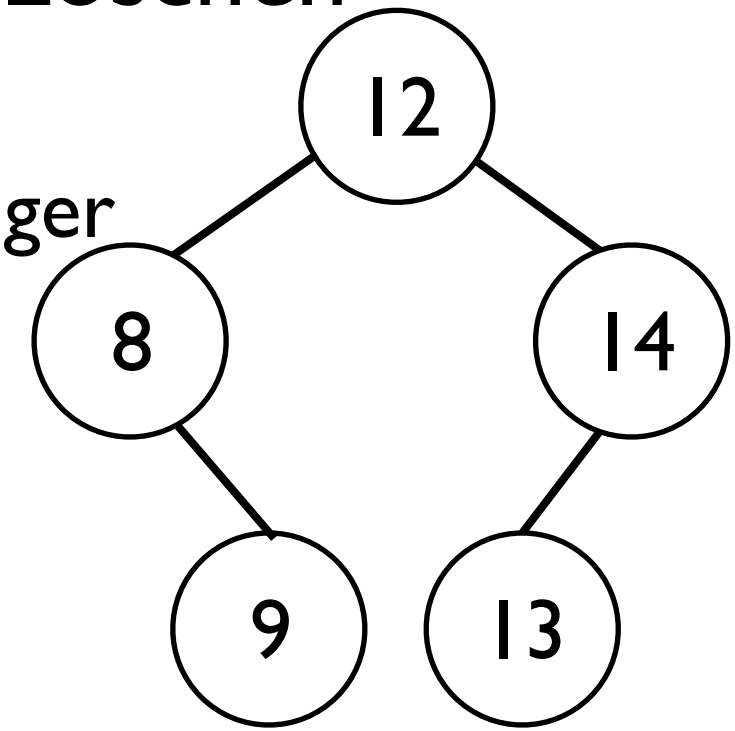


- Bis auf Sonderfälle aufwendiger
- k sei zu löschender Knoten
- Sonderfälle:

1. k ist Blatt. Lösche k
2. k hat keinen rechten Sohn: Lösche k; neuer Sohn des Vaters von k wird der linke Sohn von k
3. k hat keinen linken Sohn: analog.

Suchen

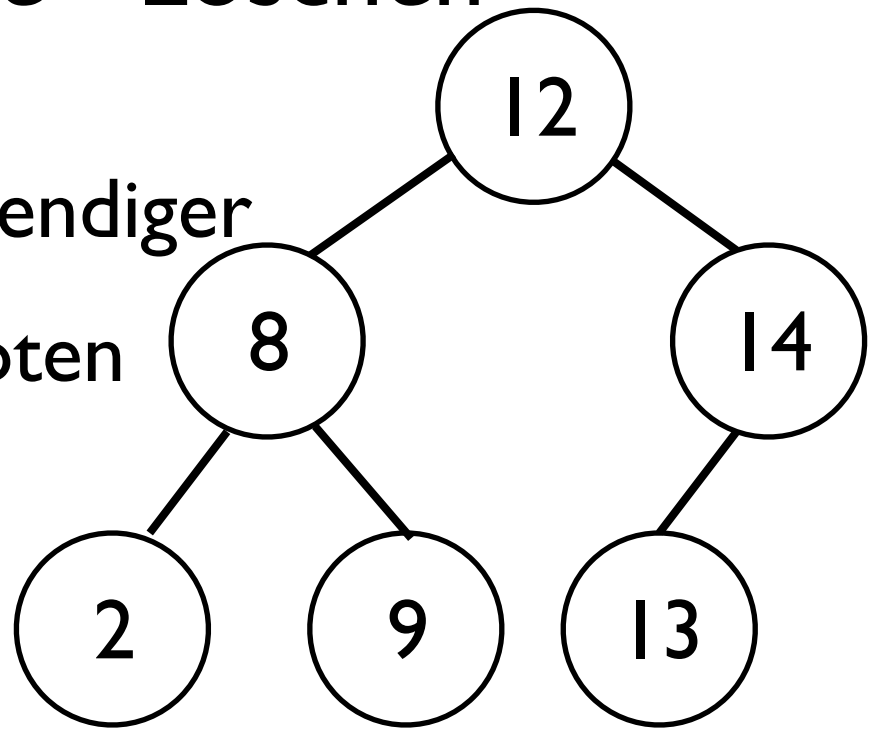
binäre Suchbäume - Löschen



- Bis auf Sonderfälle aufwendiger
- k sei zu löschender Knoten
- Sonderfälle:

1. k ist Blatt. Lösche k
2. k hat keinen rechten Sohn: Lösche k; neuer Sohn des Vaters von k wird der linke Sohn von k
3. k hat keinen linken Sohn: analog.

Suchen
binäre Suchbäume - Löschen

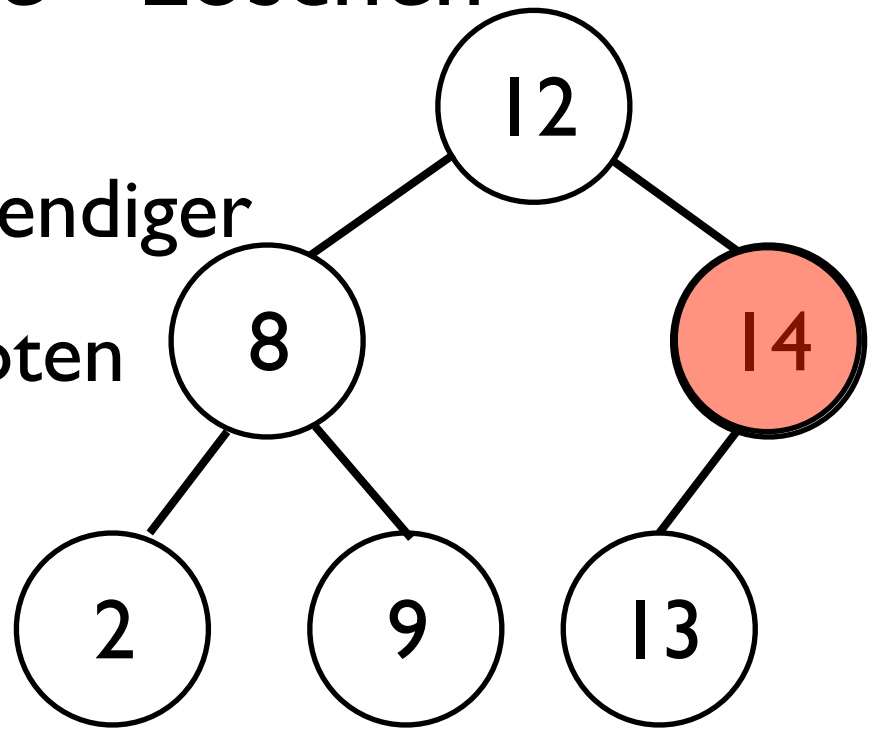


- Bis auf Sonderfälle aufwendiger
- k sei zu löschender Knoten
- Sonderfälle:

1. k ist Blatt. Lösche k
2. k hat keinen rechten Sohn: Lösche k; neuer Sohn des Vaters von k wird der linke Sohn von k
3. k hat keinen linken Sohn: analog.

Suchen

binäre Suchbäume - Löschen

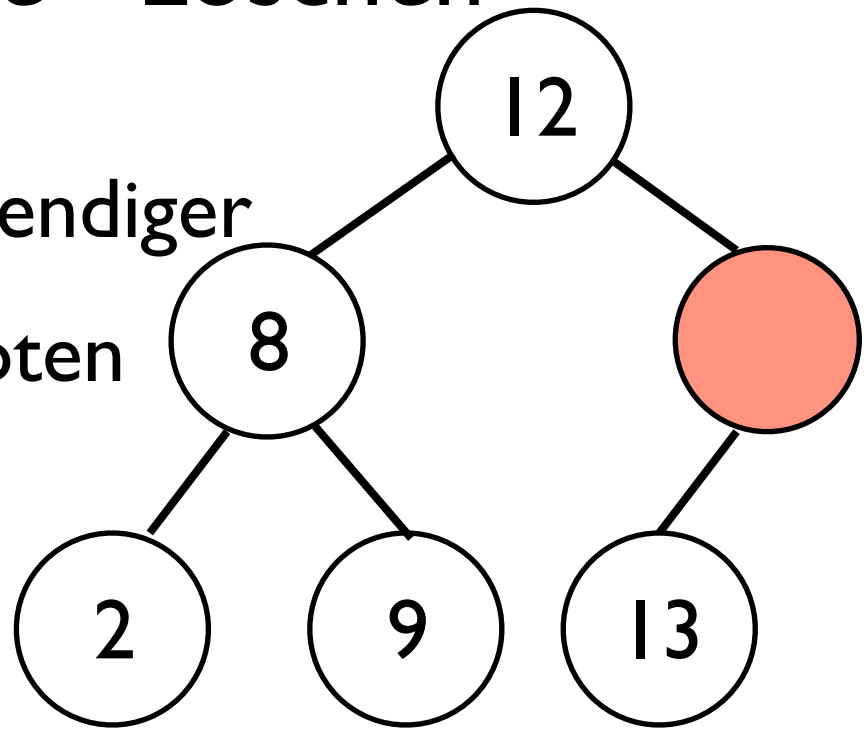


- Bis auf Sonderfälle aufwendiger
- k sei zu löschender Knoten
- Sonderfälle:

1. k ist Blatt. Lösche k
2. k hat keinen rechten Sohn: Lösche k; neuer Sohn des Vaters von k wird der linke Sohn von k
3. k hat keinen linken Sohn: analog.

Suchen

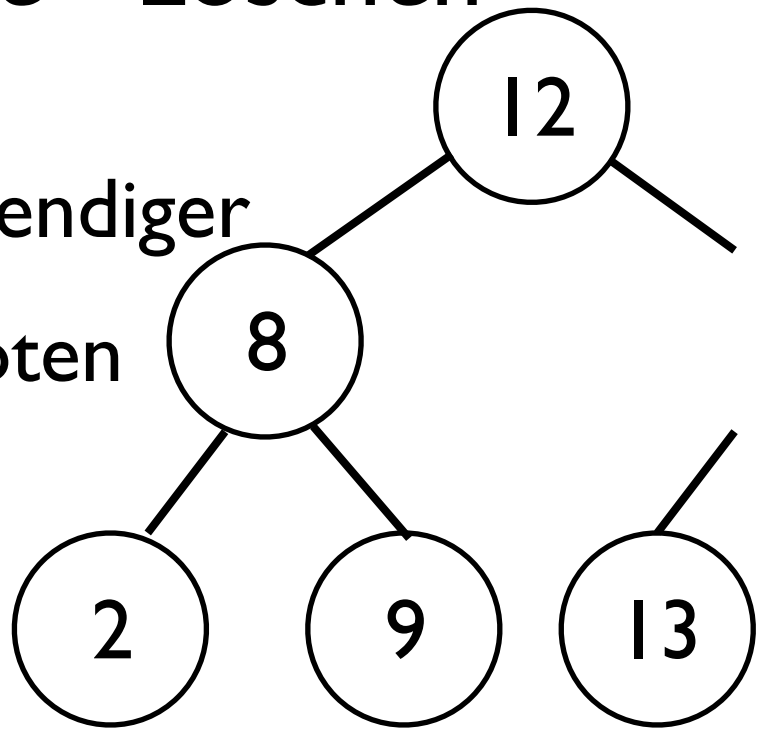
binäre Suchbäume - Löschen



- Bis auf Sonderfälle aufwendiger
- k sei zu löschender Knoten
- Sonderfälle:

1. k ist Blatt. Lösche k
2. k hat keinen rechten Sohn: Lösche k; neuer Sohn des Vaters von k wird der linke Sohn von k
3. k hat keinen linken Sohn: analog.

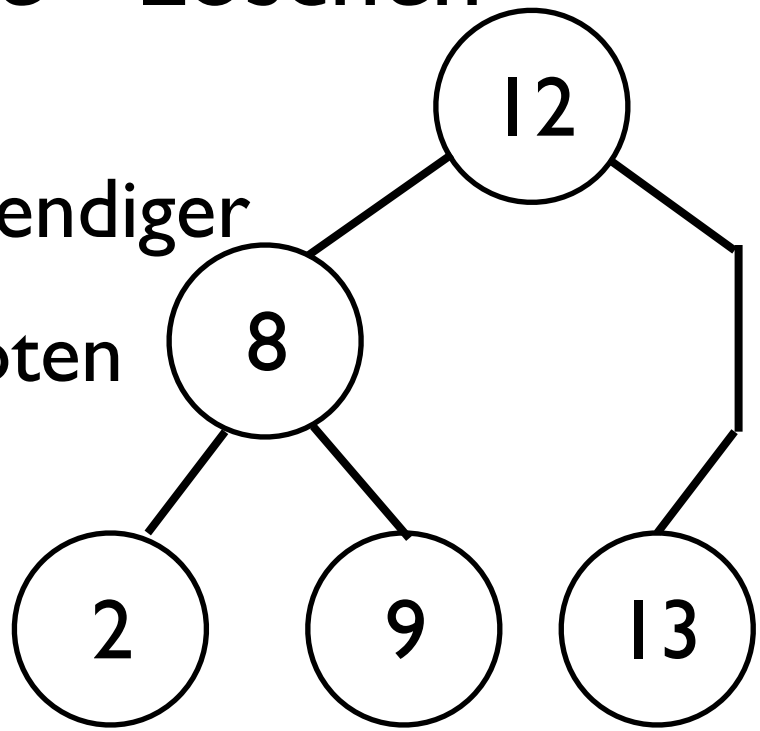
Suchen
binäre Suchbäume - Löschen



- Bis auf Sonderfälle aufwendiger
- k sei zu löschender Knoten
- Sonderfälle:

1. k ist Blatt. Lösche k
2. k hat keinen rechten Sohn: Lösche k; neuer Sohn des Vaters von k wird der linke Sohn von k
3. k hat keinen linken Sohn: analog.

Suchen
binäre Suchbäume - Löschen



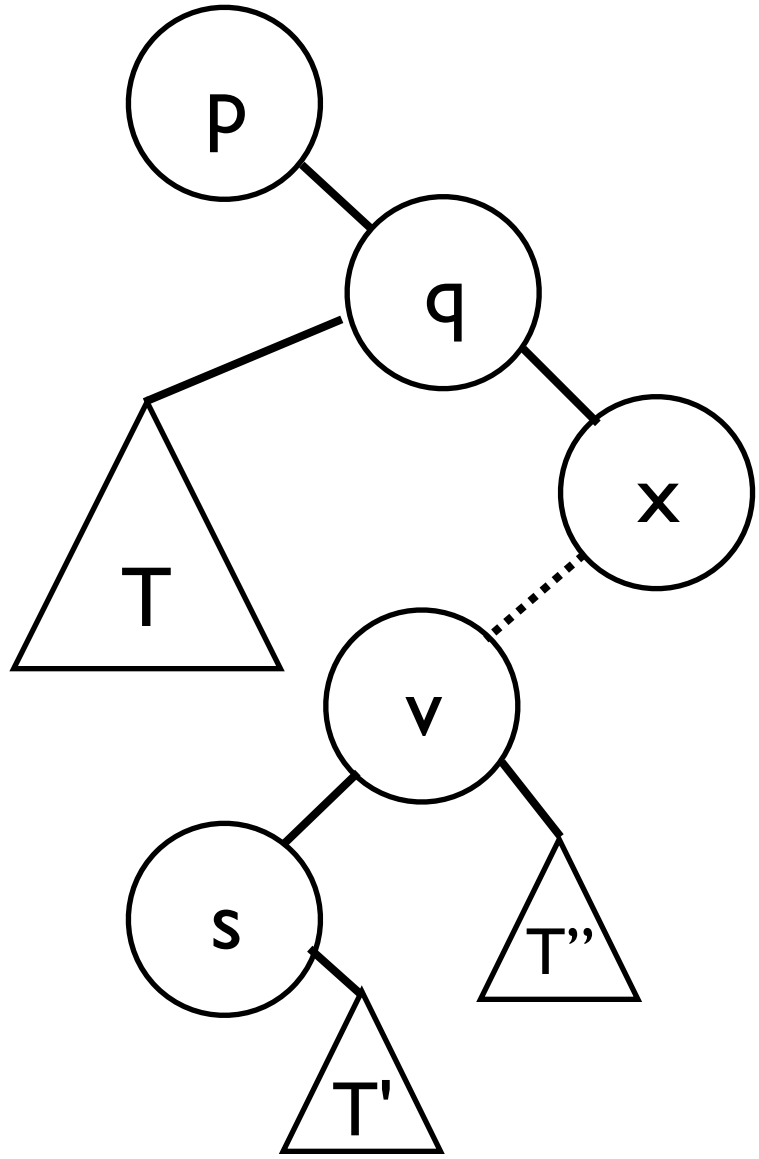
- Bis auf Sonderfälle aufwendiger
- k sei zu löschender Knoten
- Sonderfälle:

1. k ist Blatt. Lösche k
2. k hat keinen rechten Sohn: Lösche k; neuer Sohn des Vaters von k wird der linke Sohn von k
3. k hat keinen linken Sohn: analog.

Suchen
binäre Suchbäume - Löschen

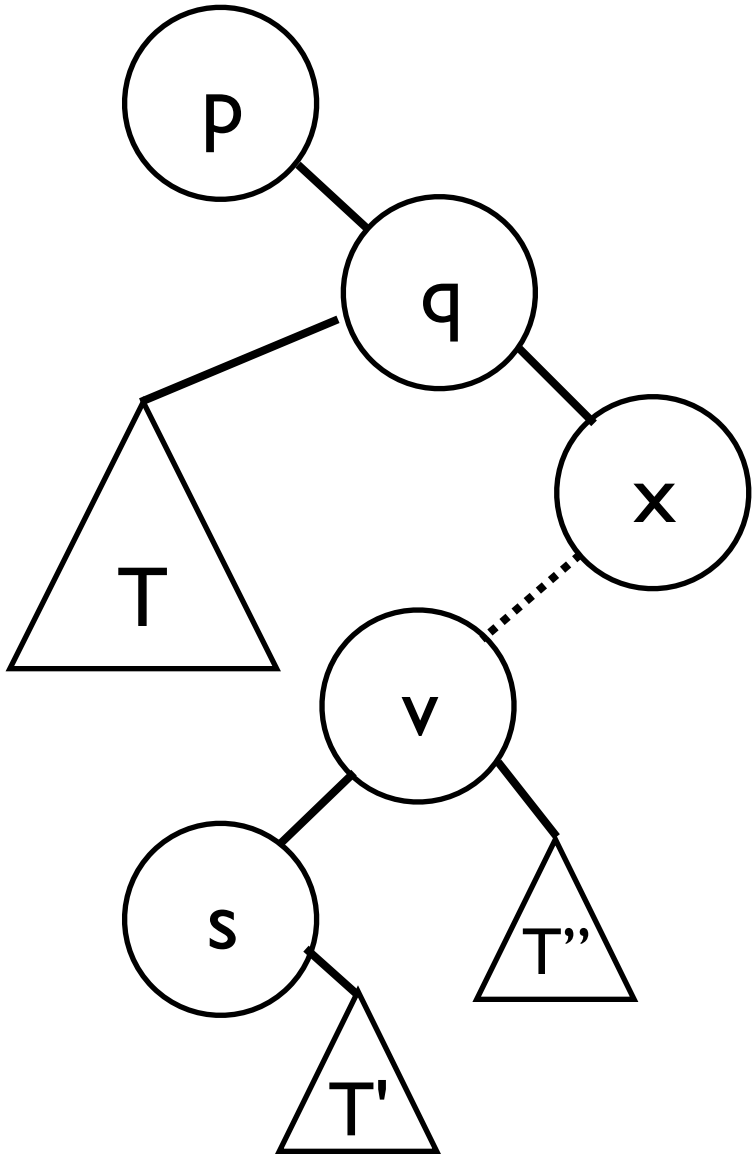
Allgemein:

- Suche in der inorder-Reihenfolge im rechten Teilbaum von k den ersten Knoten, der keinen linken Sohn besitzt, und füge ihn an die Stelle des zu entfernenden Knotens ein.



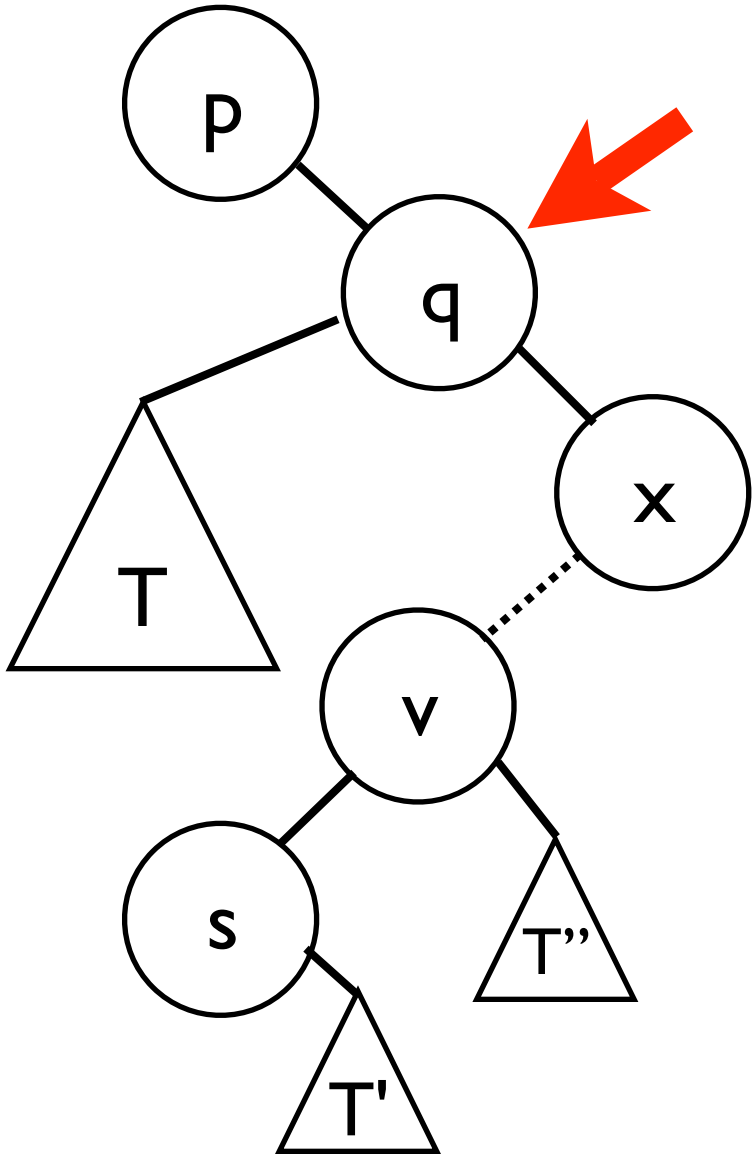
Suchen

binäre Suchbäume - Löschen



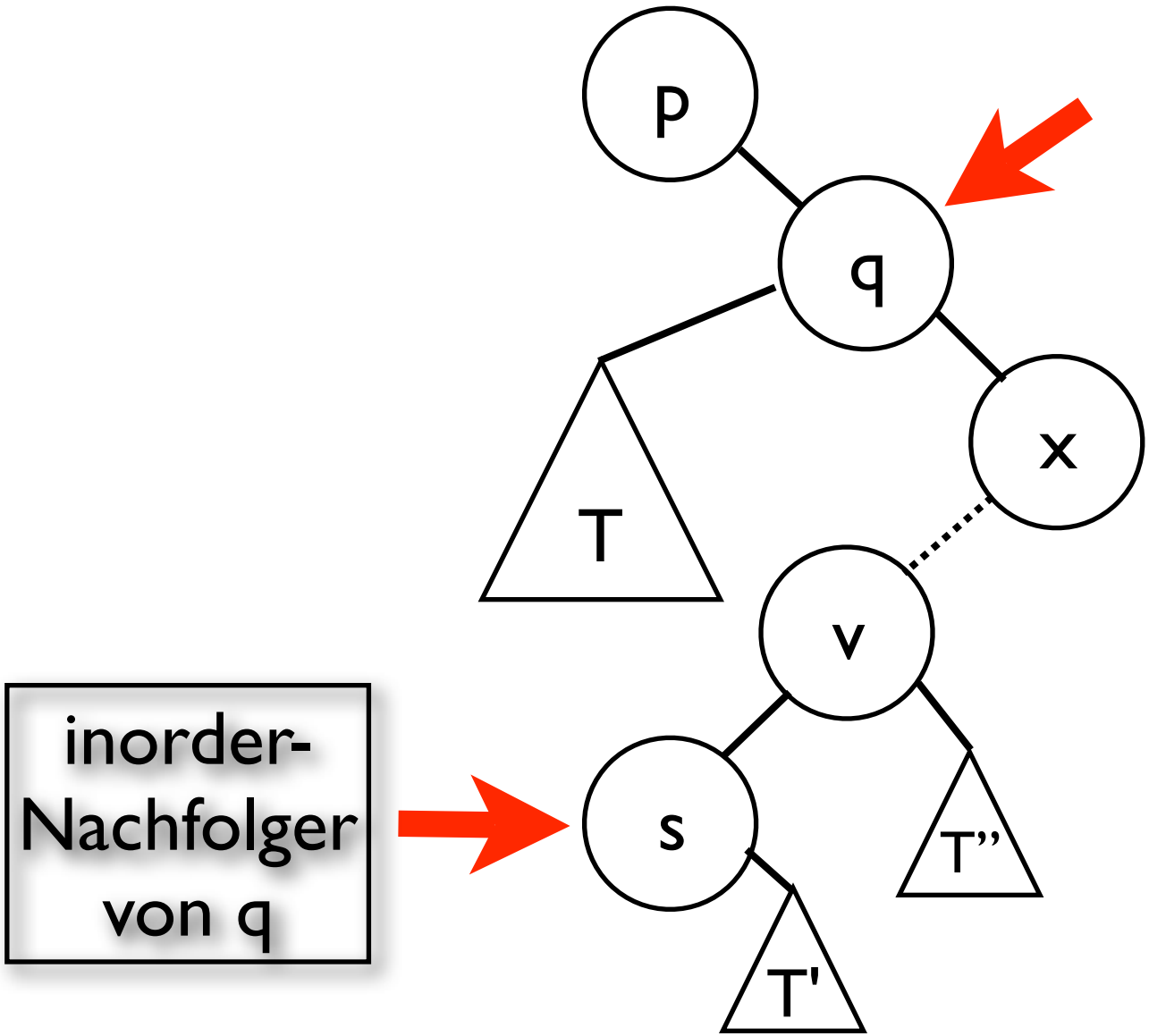
Suchen

binäre Suchbäume - Löschen



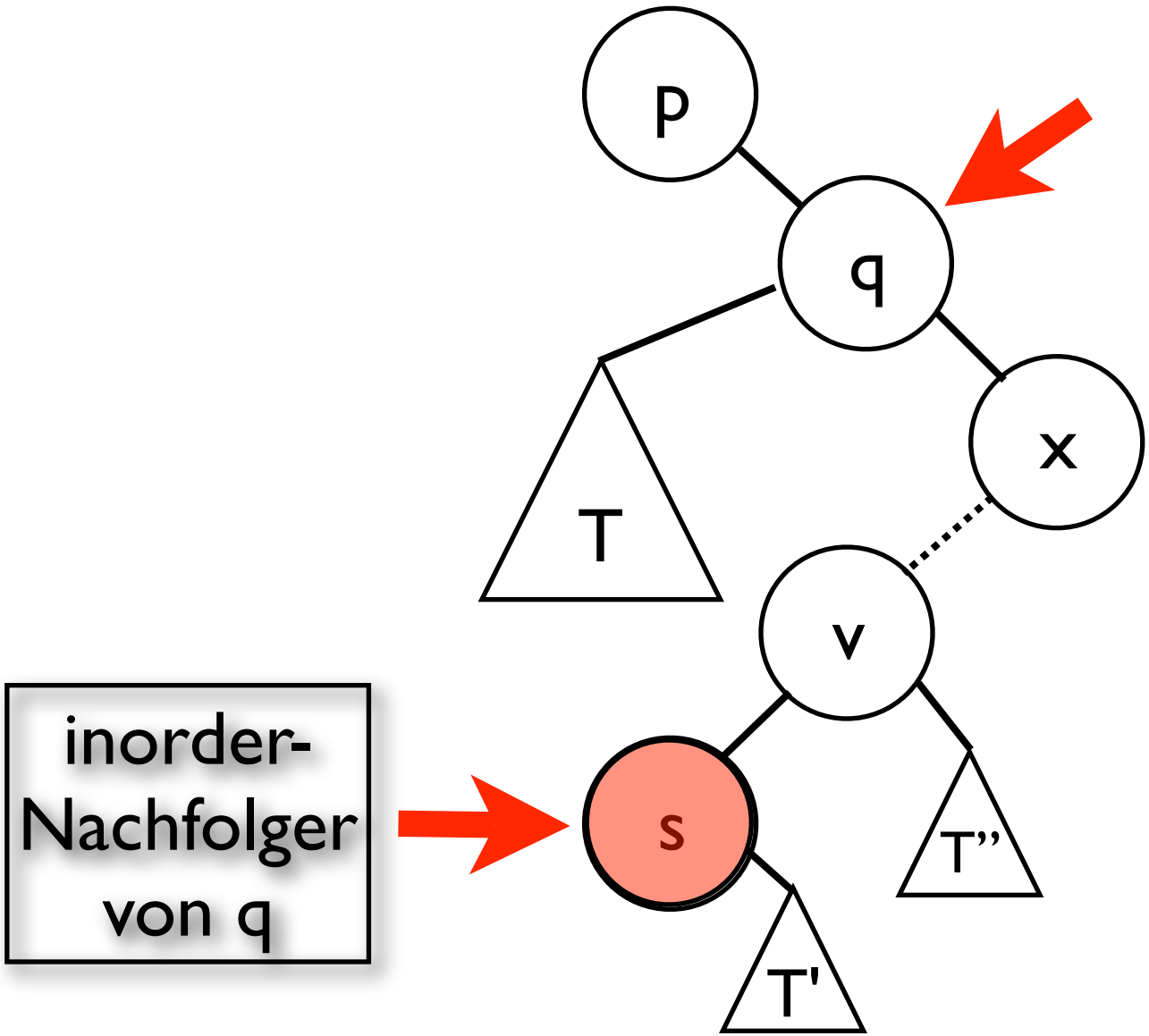
Suchen

binäre Suchbäume - Löschen



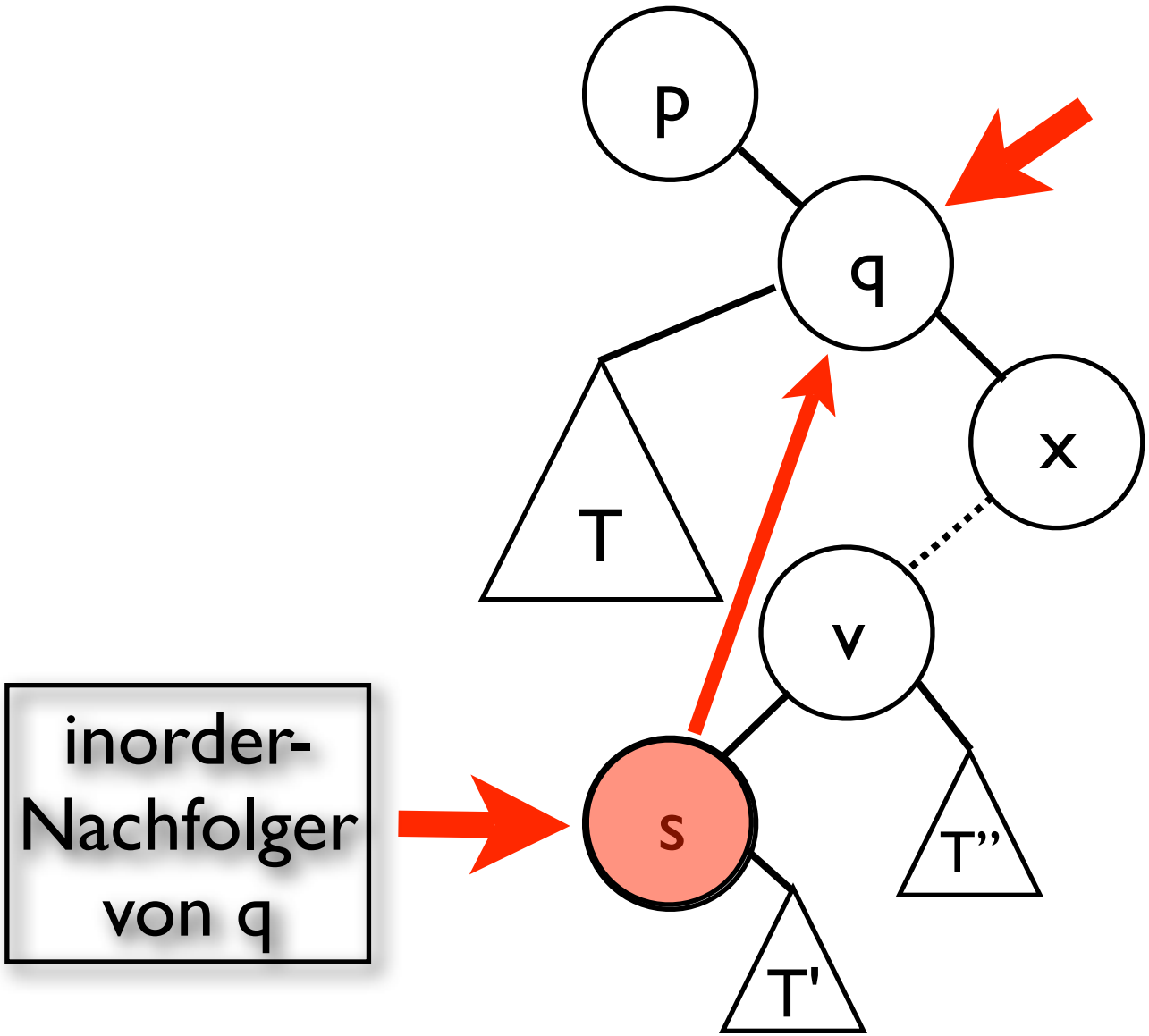
Suchen

binäre Suchbäume - Löschen



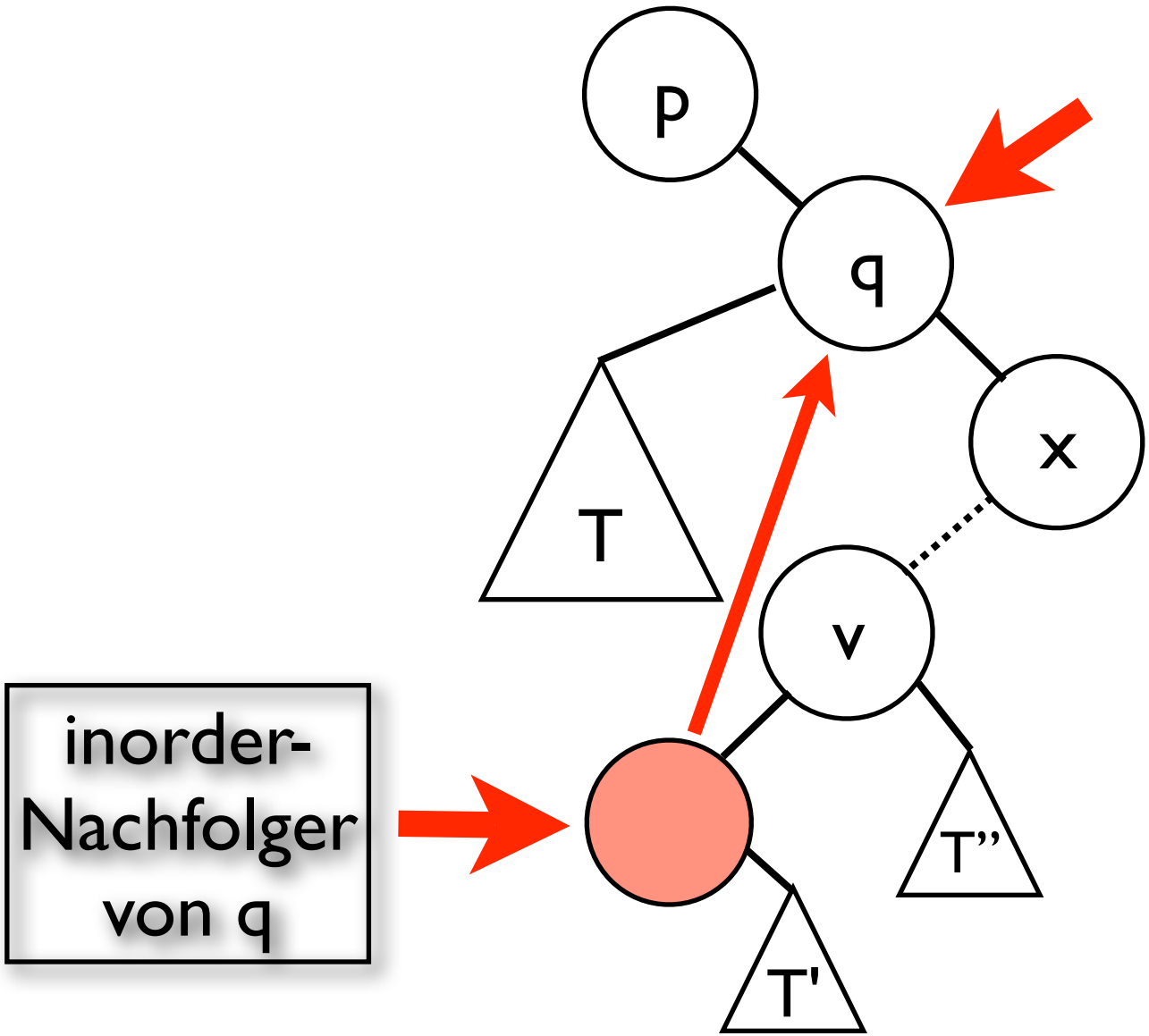
Suchen

binäre Suchbäume - Löschen



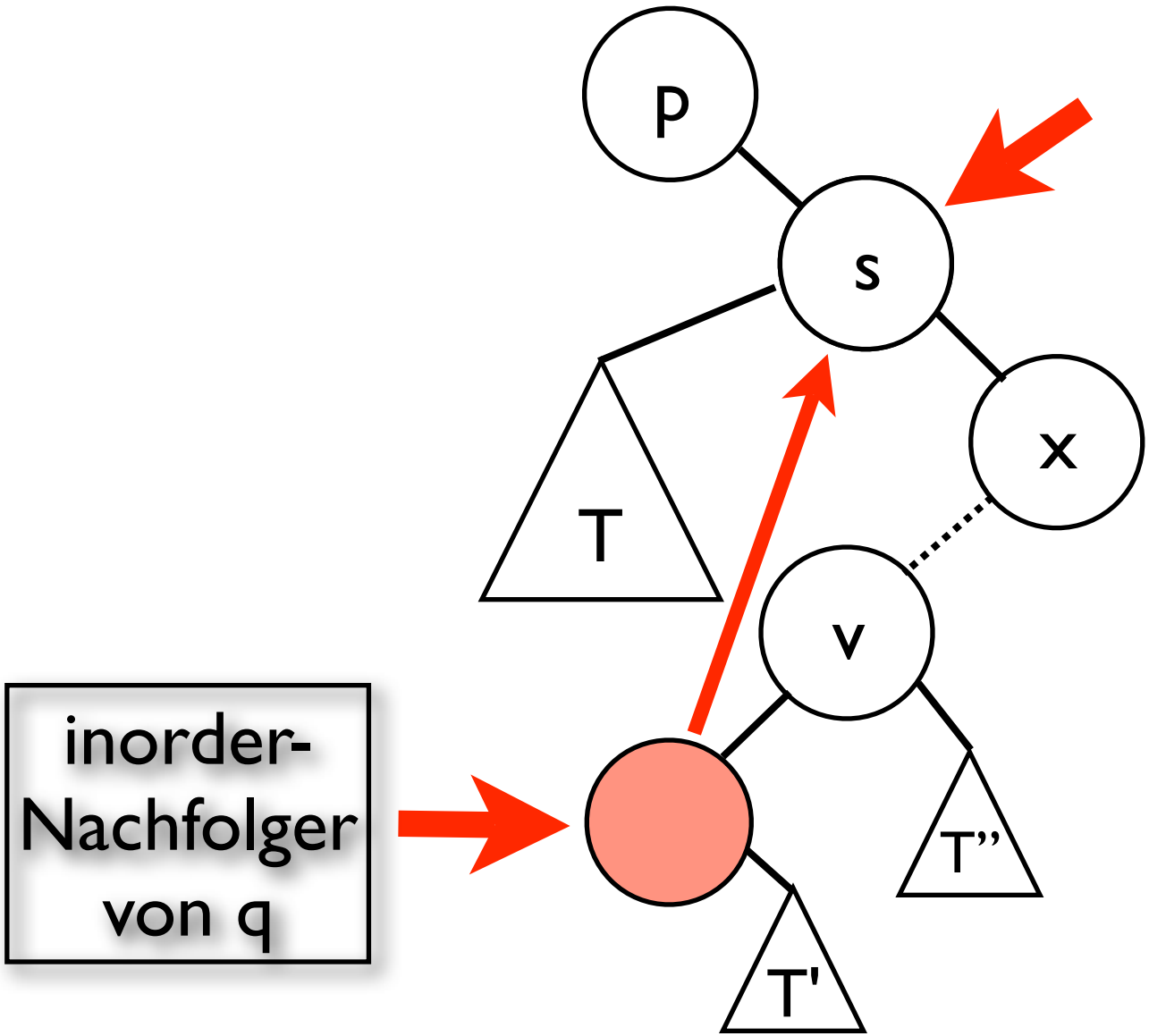
Suchen

binäre Suchbäume - Löschen



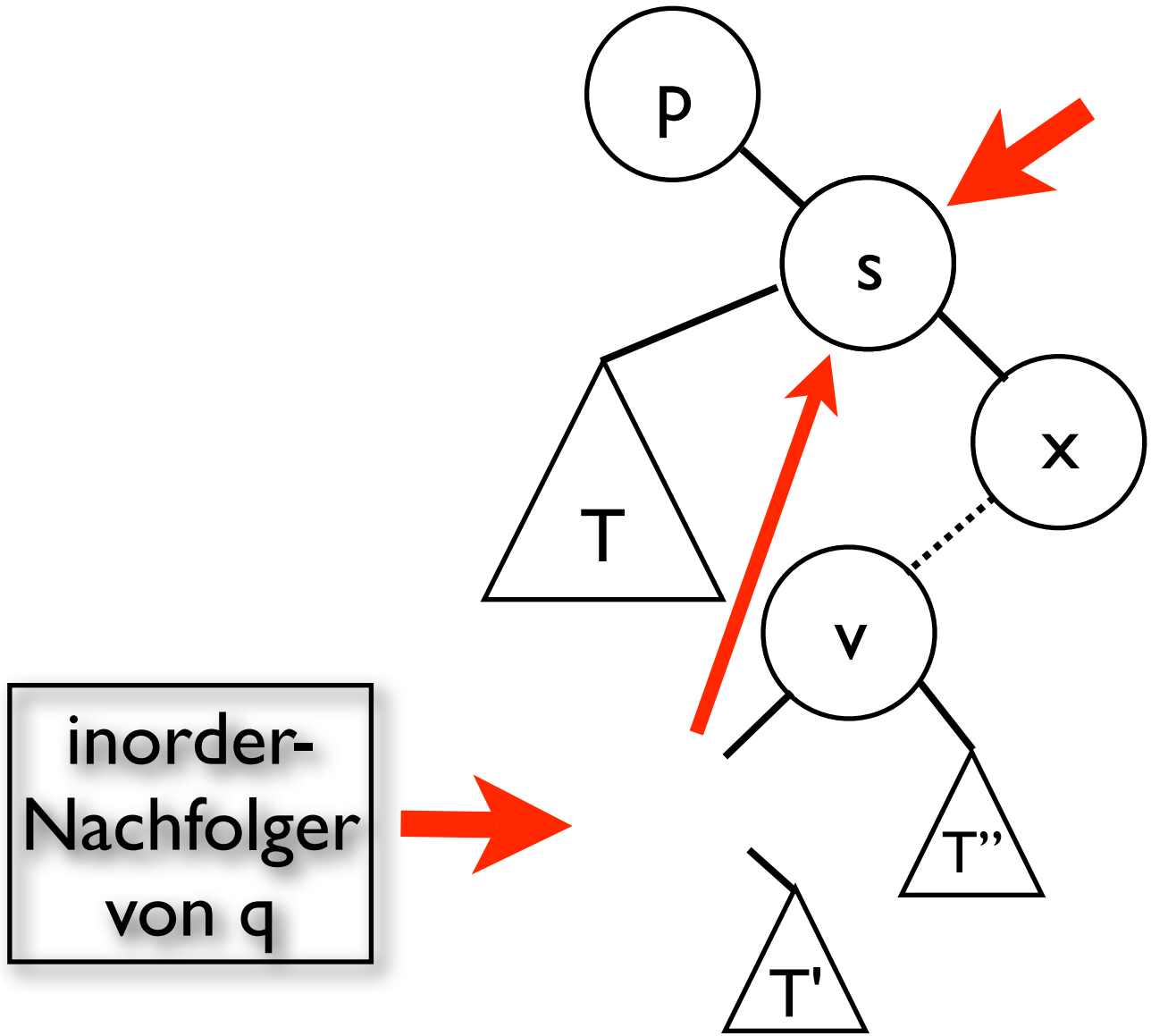
Suchen

binäre Suchbäume - Löschen



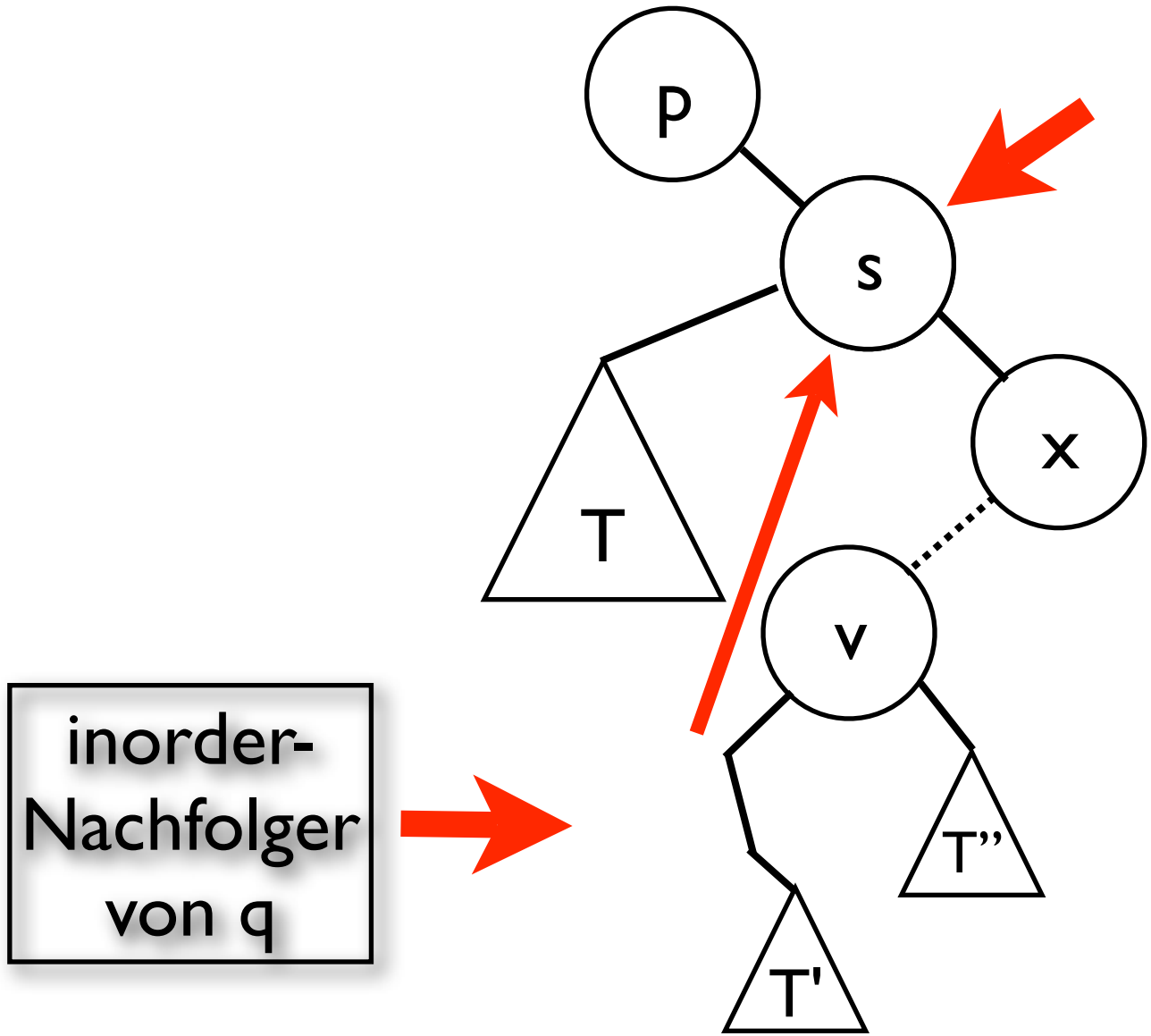
Suchen

binäre Suchbäume - Löschen



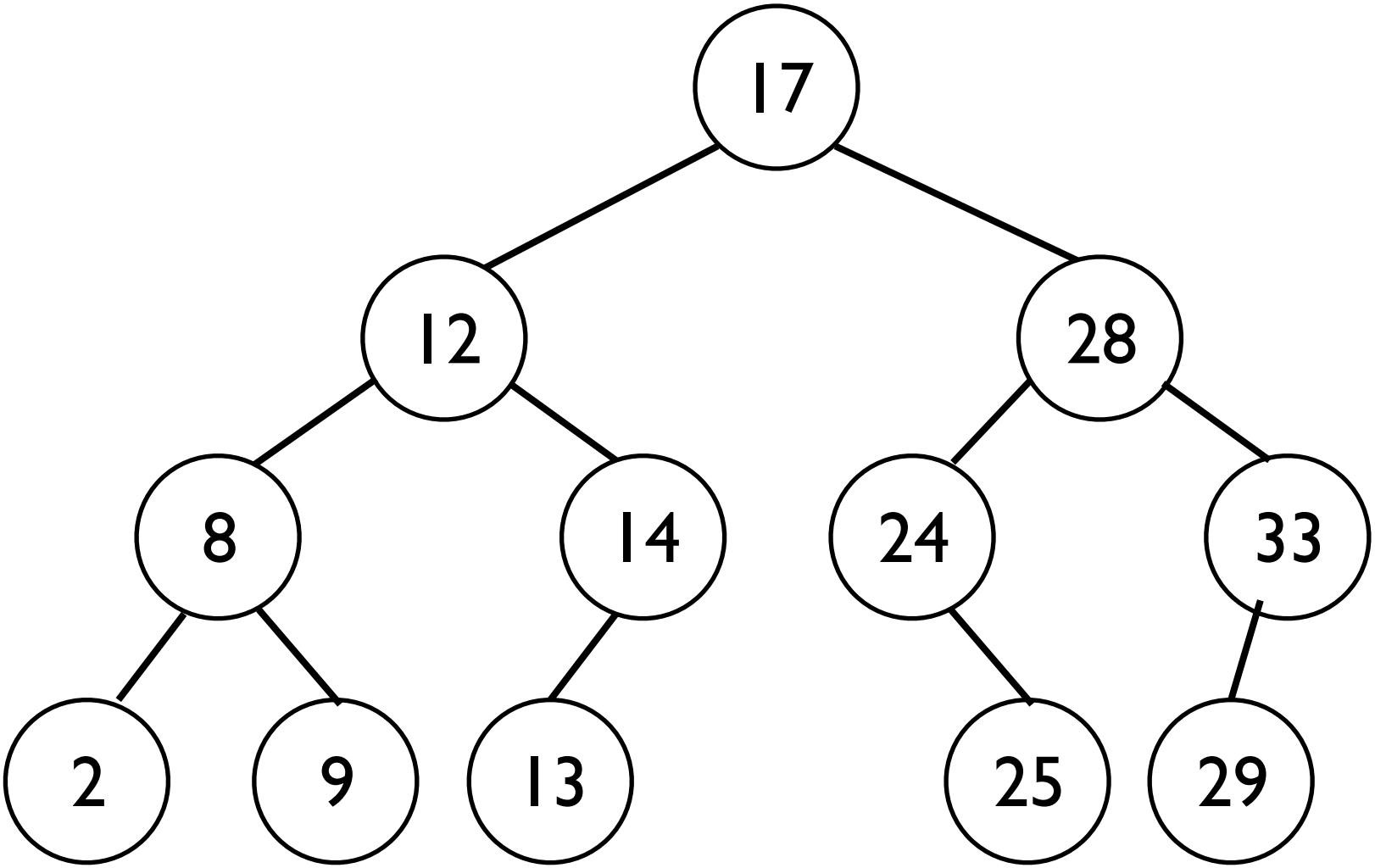
Suchen

binäre Suchbäume - Löschen



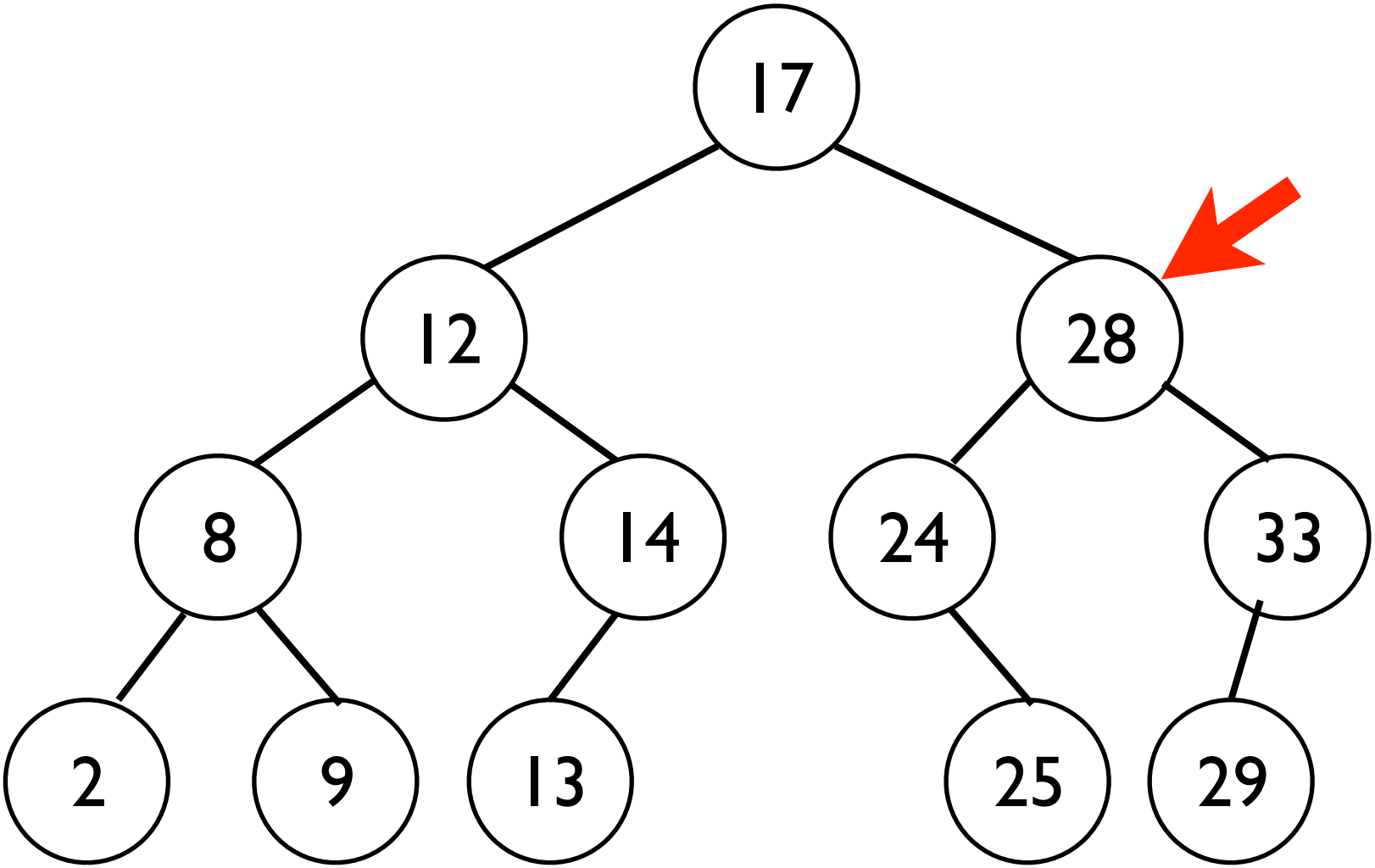
Suchen

Löschen - Beispiel



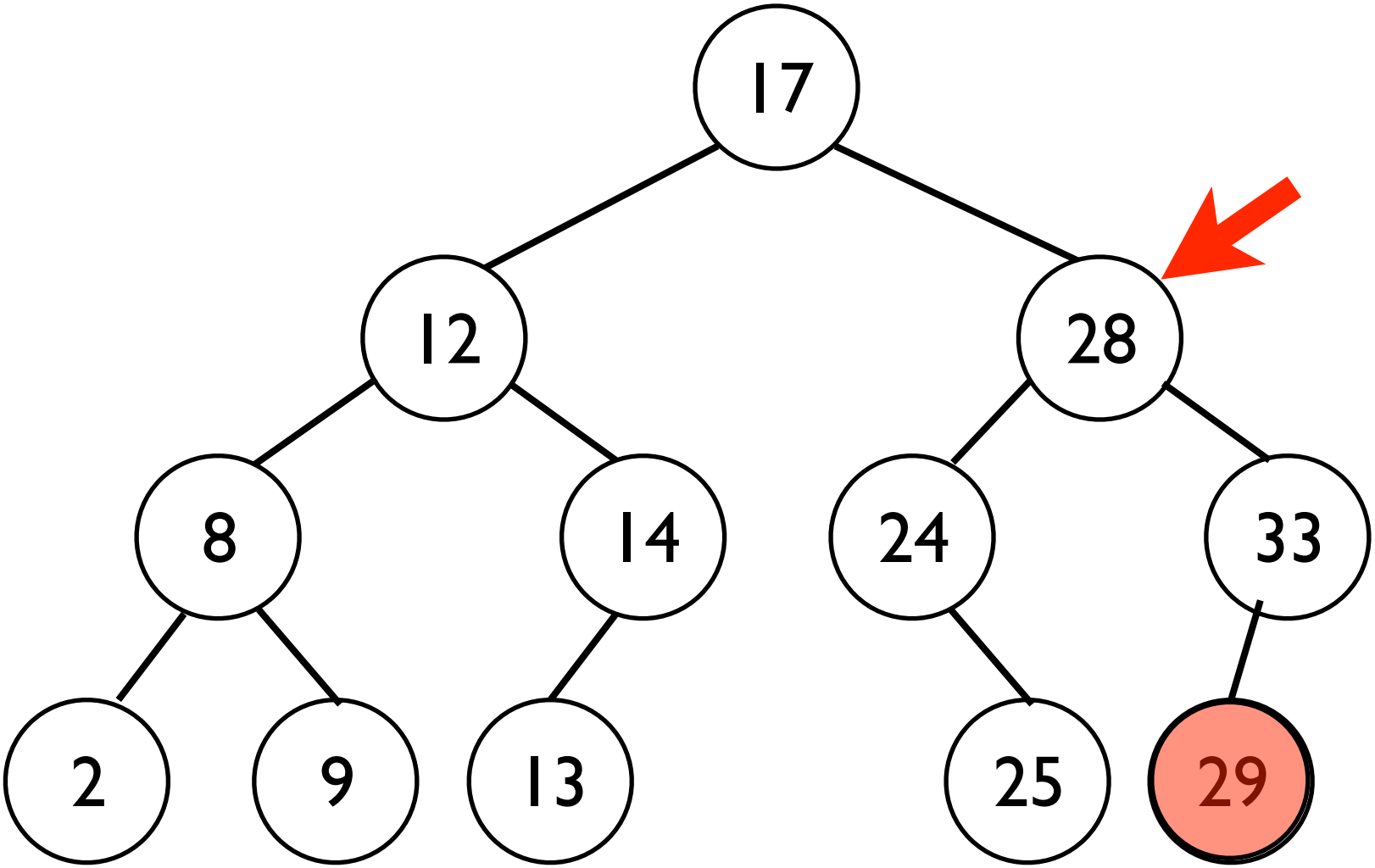
Suchen

Löschen - Beispiel



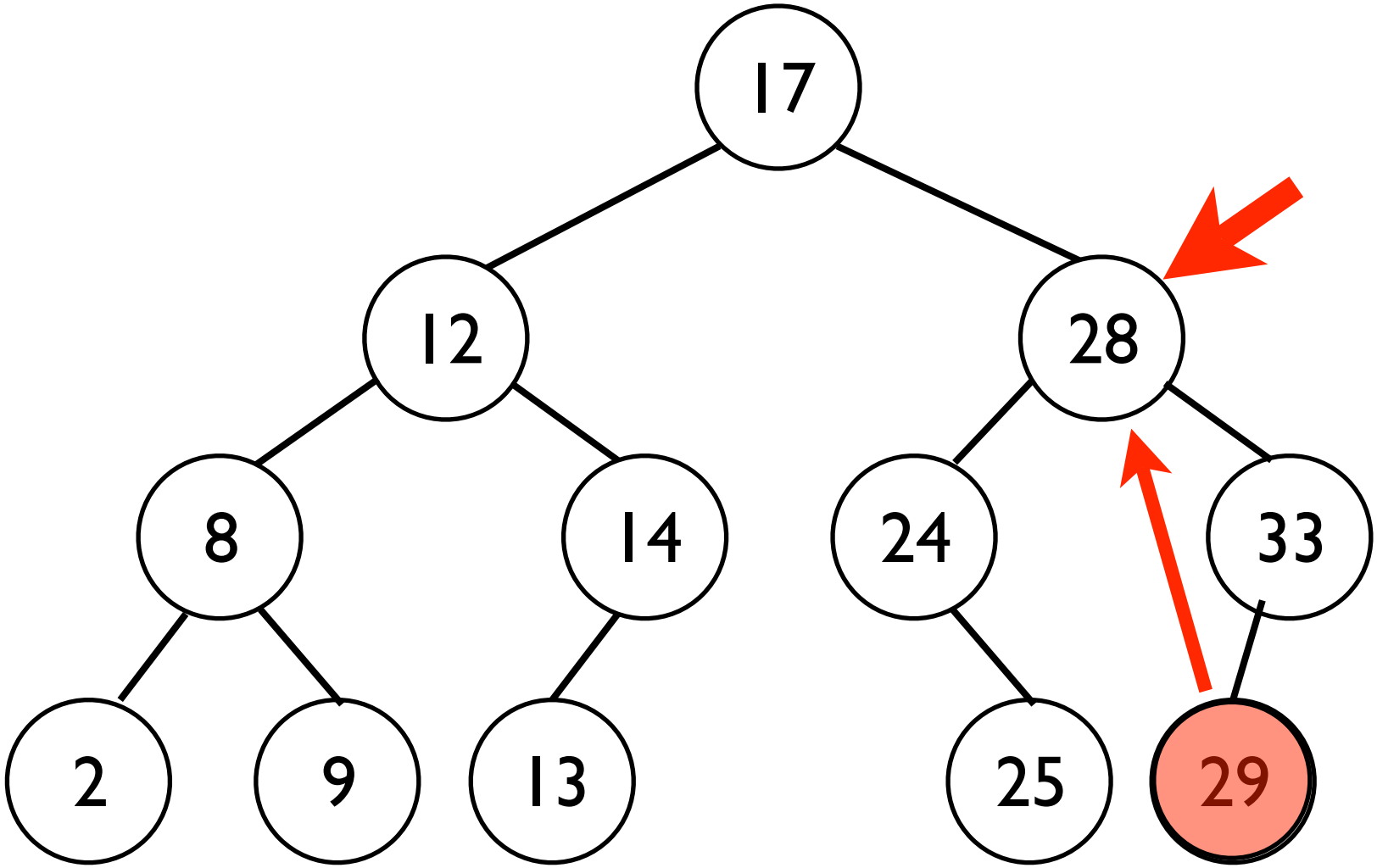
Suchen

Löschen - Beispiel



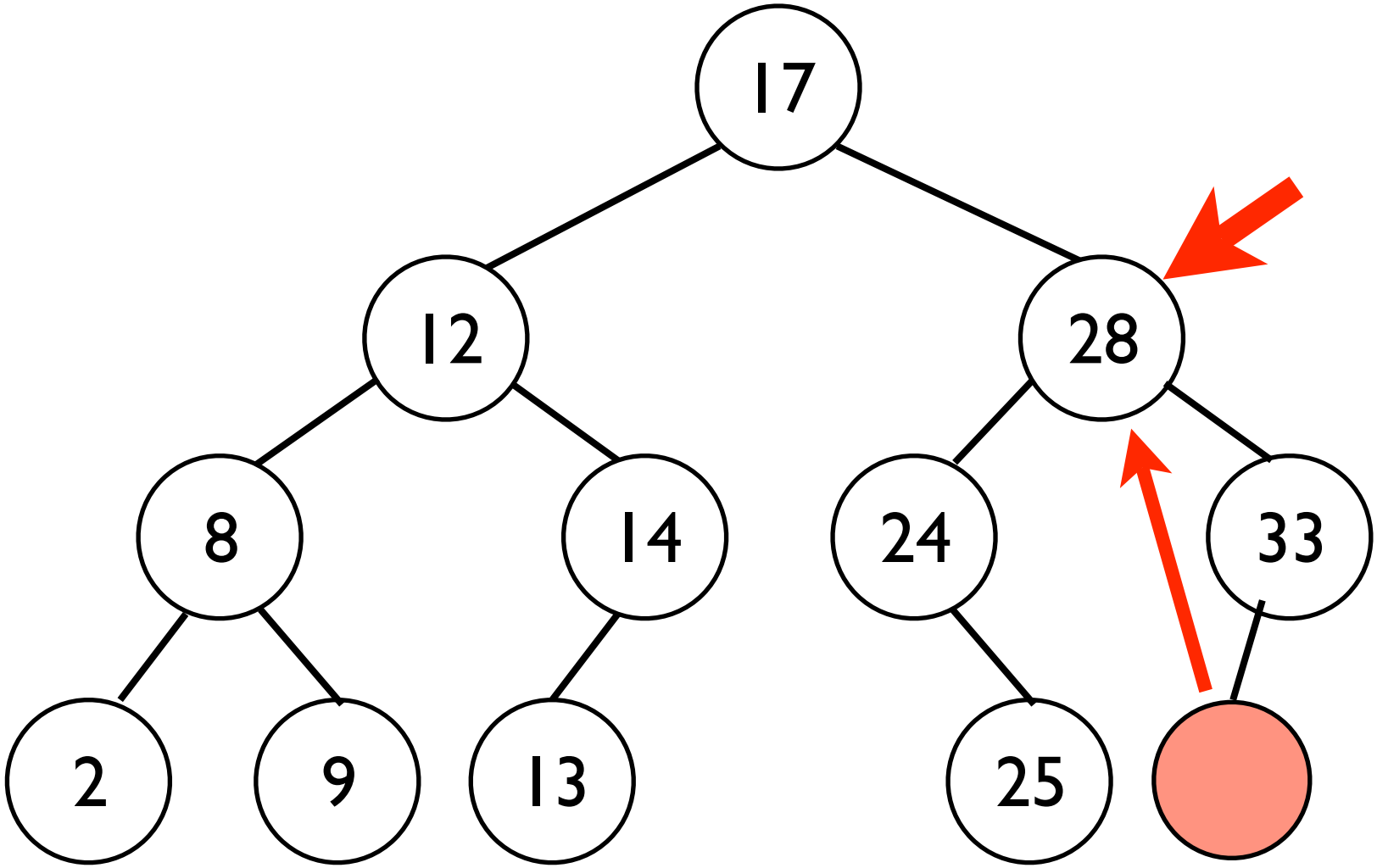
Suchen

Löschen - Beispiel



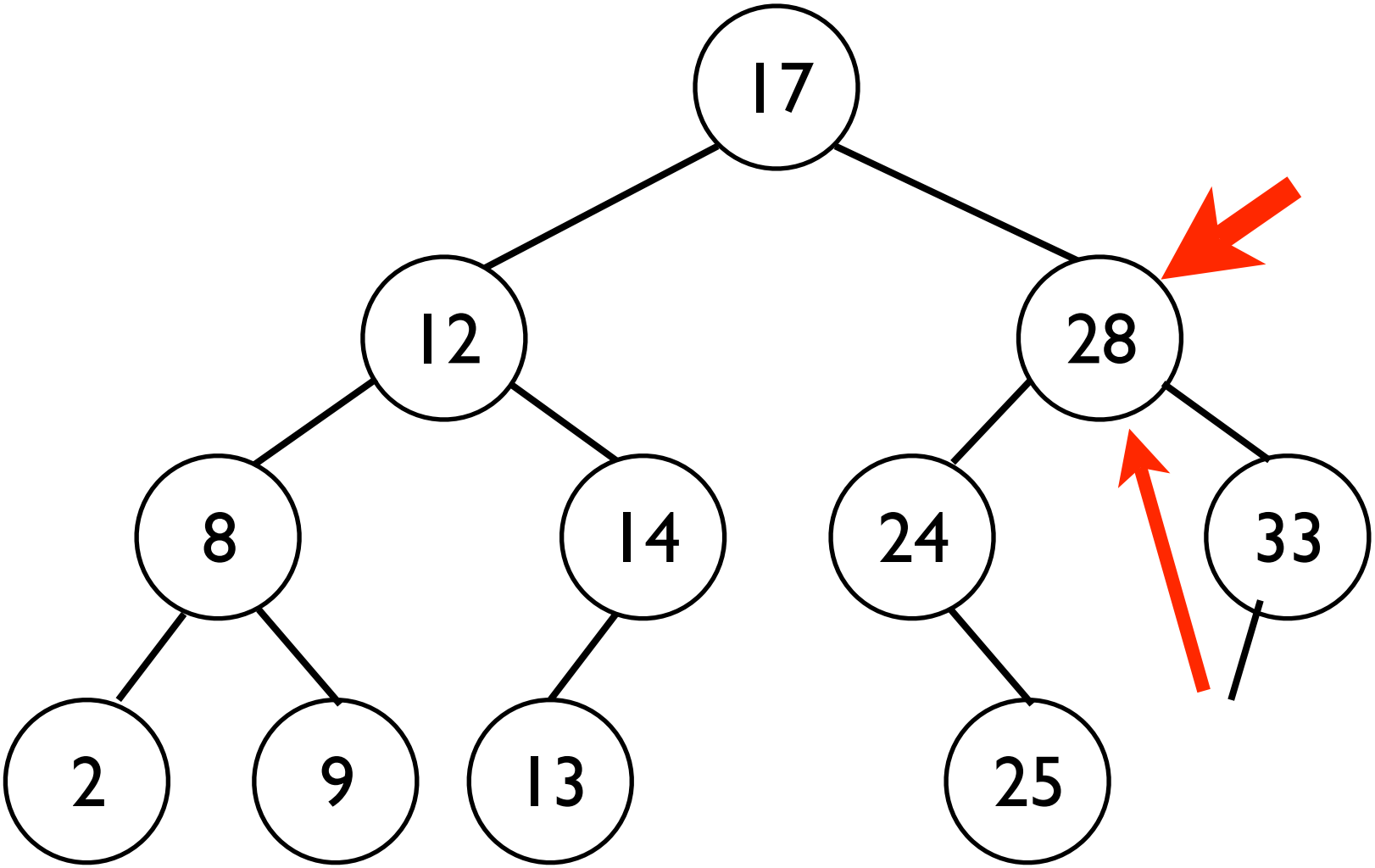
Suchen

Löschen - Beispiel



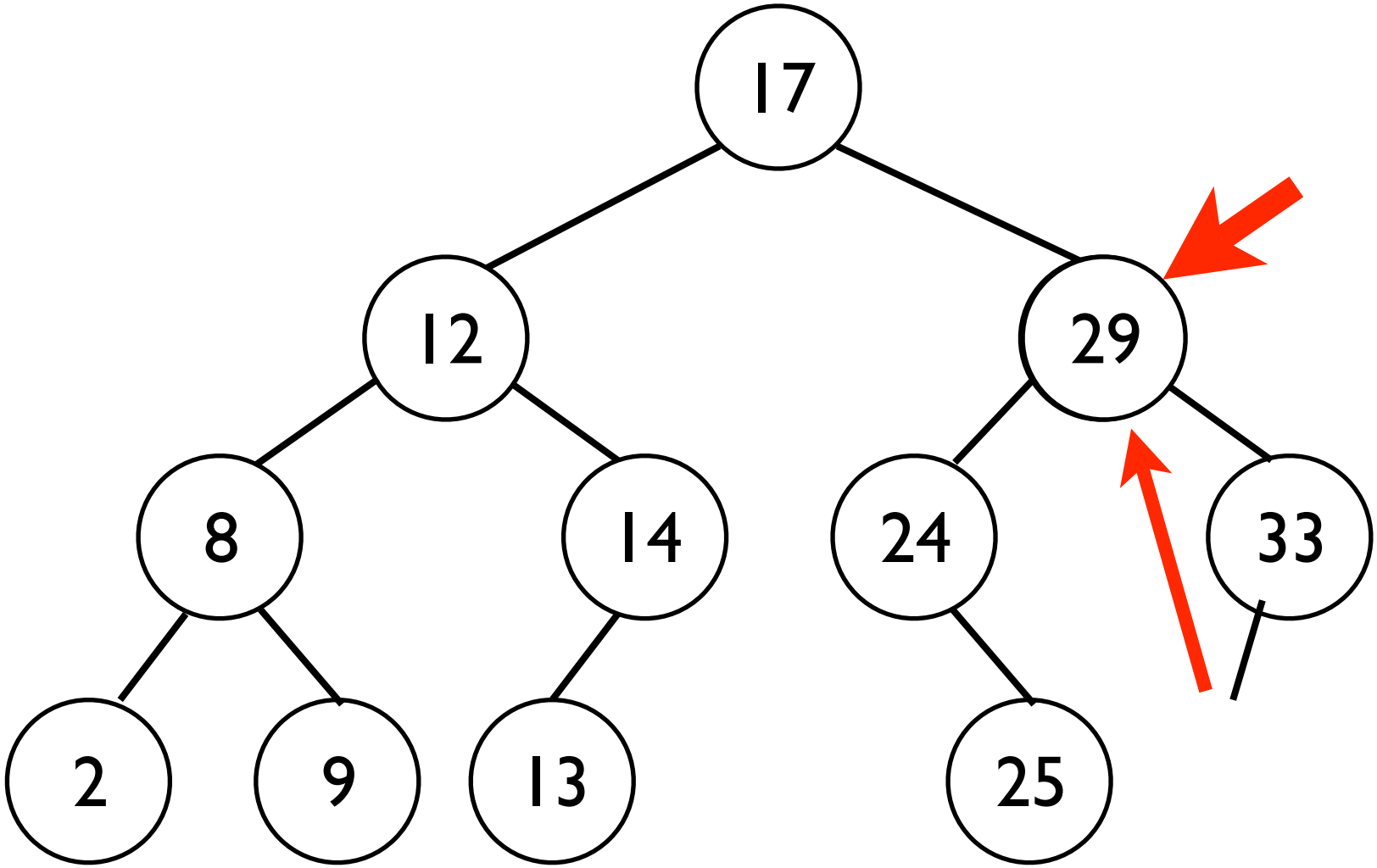
Suchen

Löschen - Beispiel



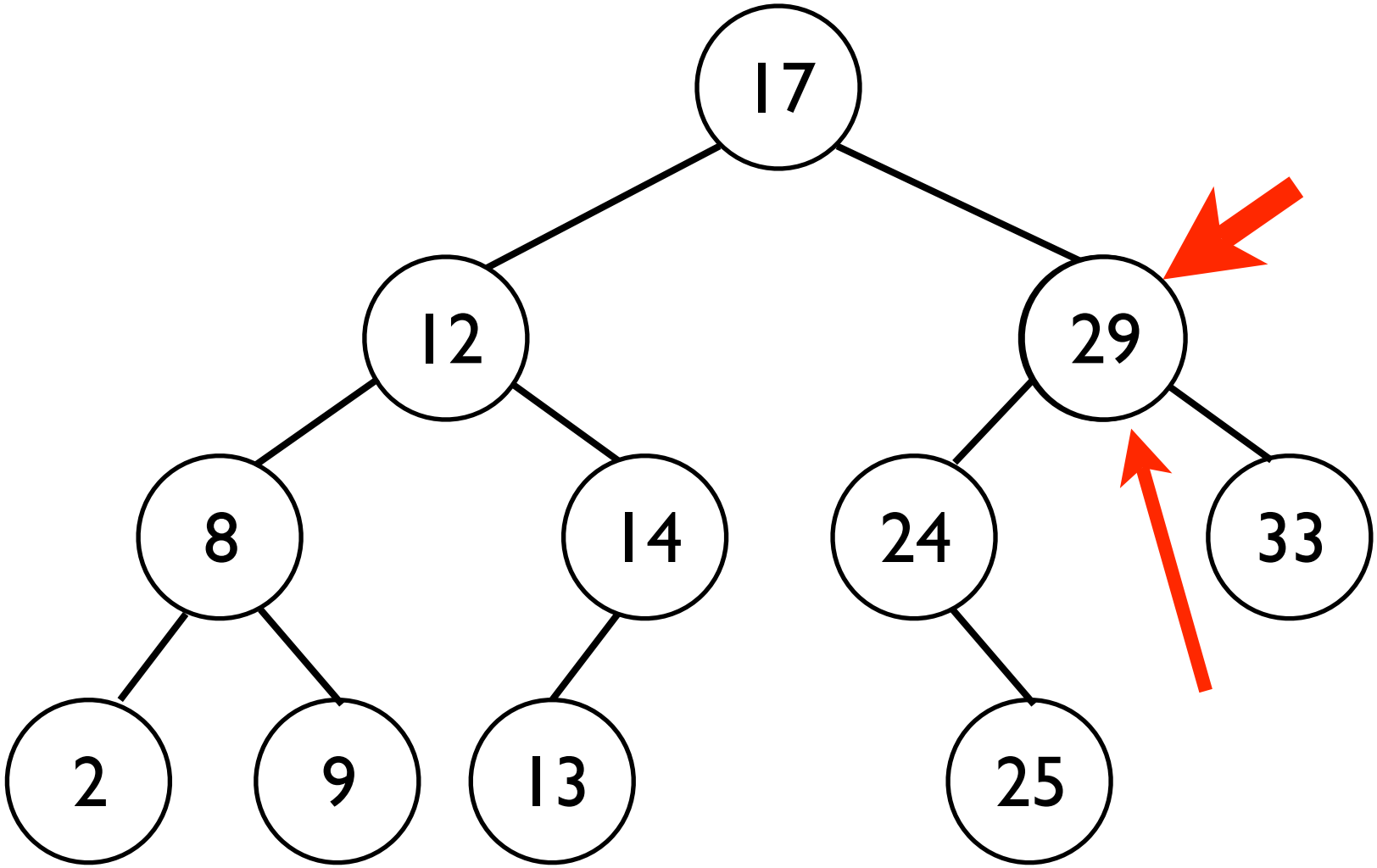
Suchen

Löschen - Beispiel



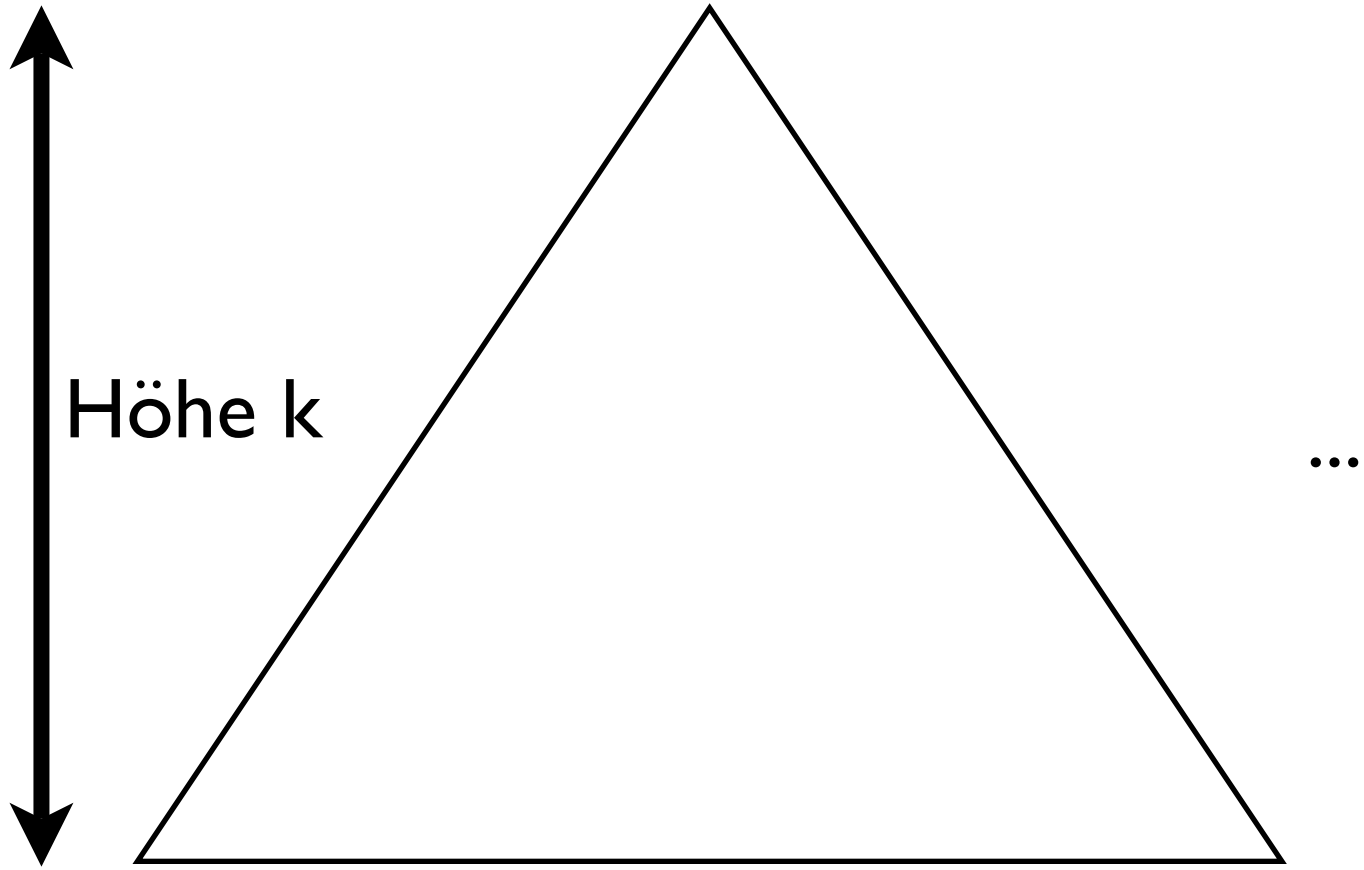
Suchen

Löschen - Beispiel



Suchen

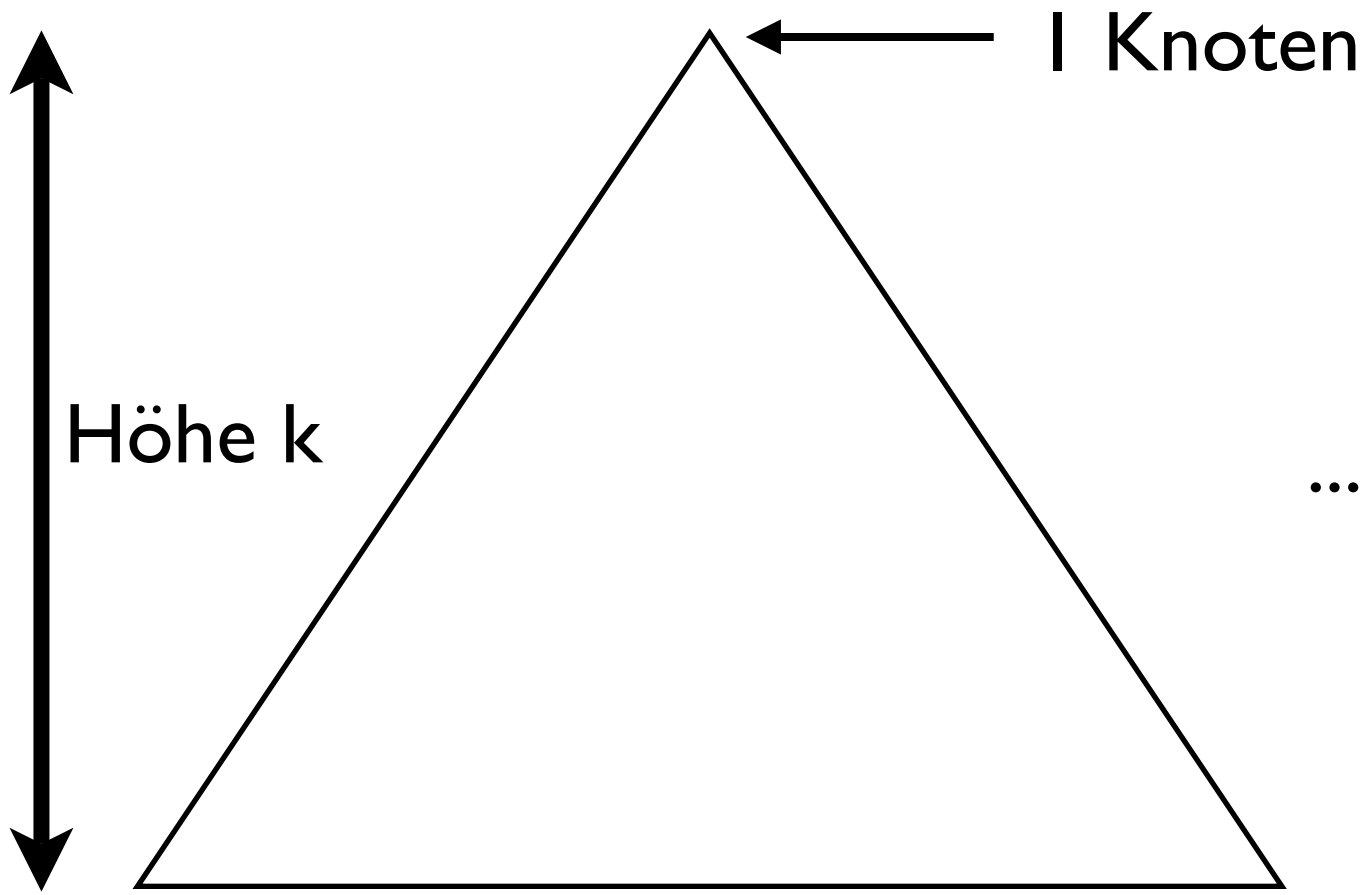
binäre Suchbäume - Laufzeitanalyse



bester Fall: Baum vollständig ausgeglichen

Suchen

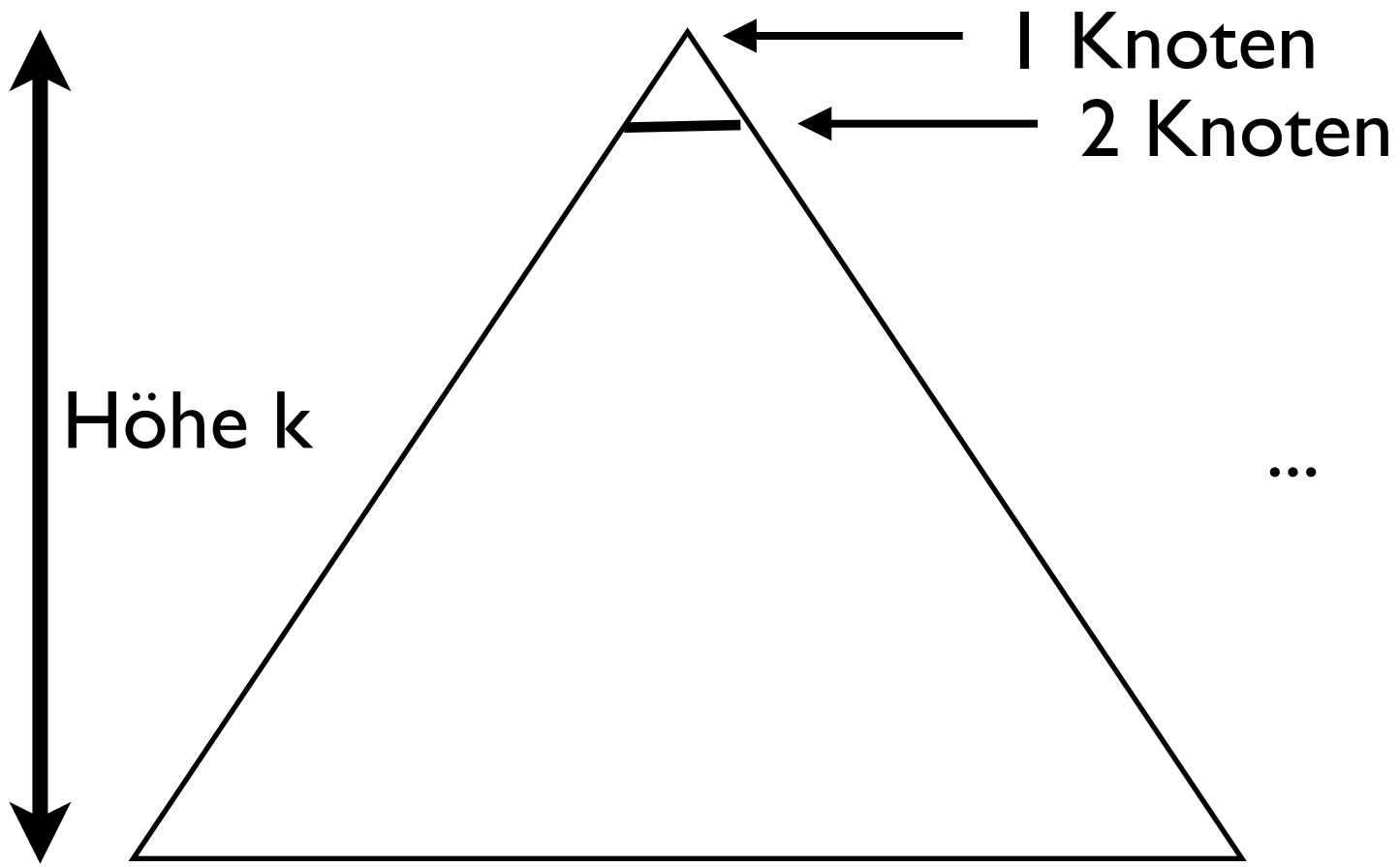
binäre Suchbäume - Laufzeitanalyse



bester Fall: Baum vollständig ausgeglichen

Suchen

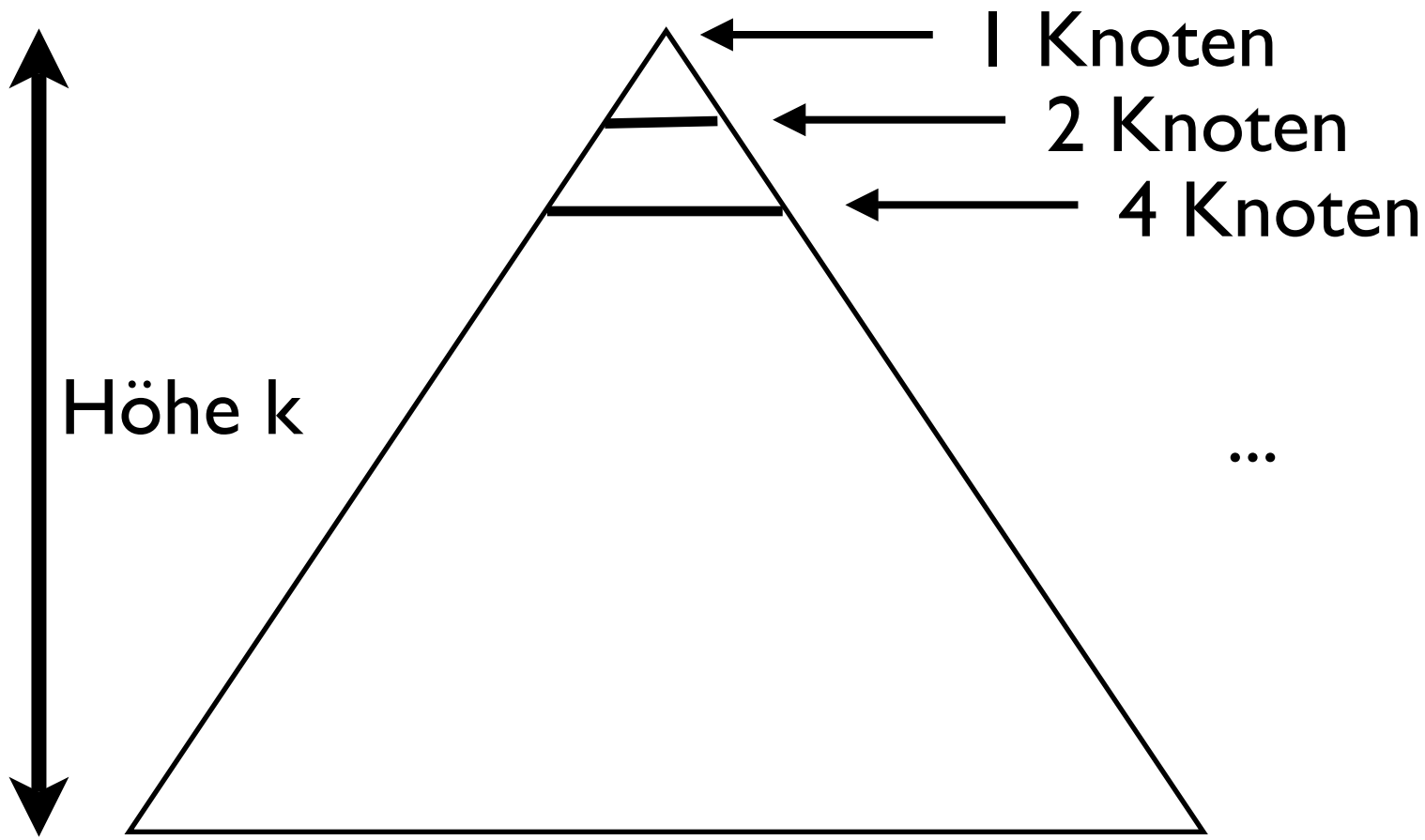
binäre Suchbäume - Laufzeitanalyse



bester Fall: Baum vollständig ausgeglichen

Suchen

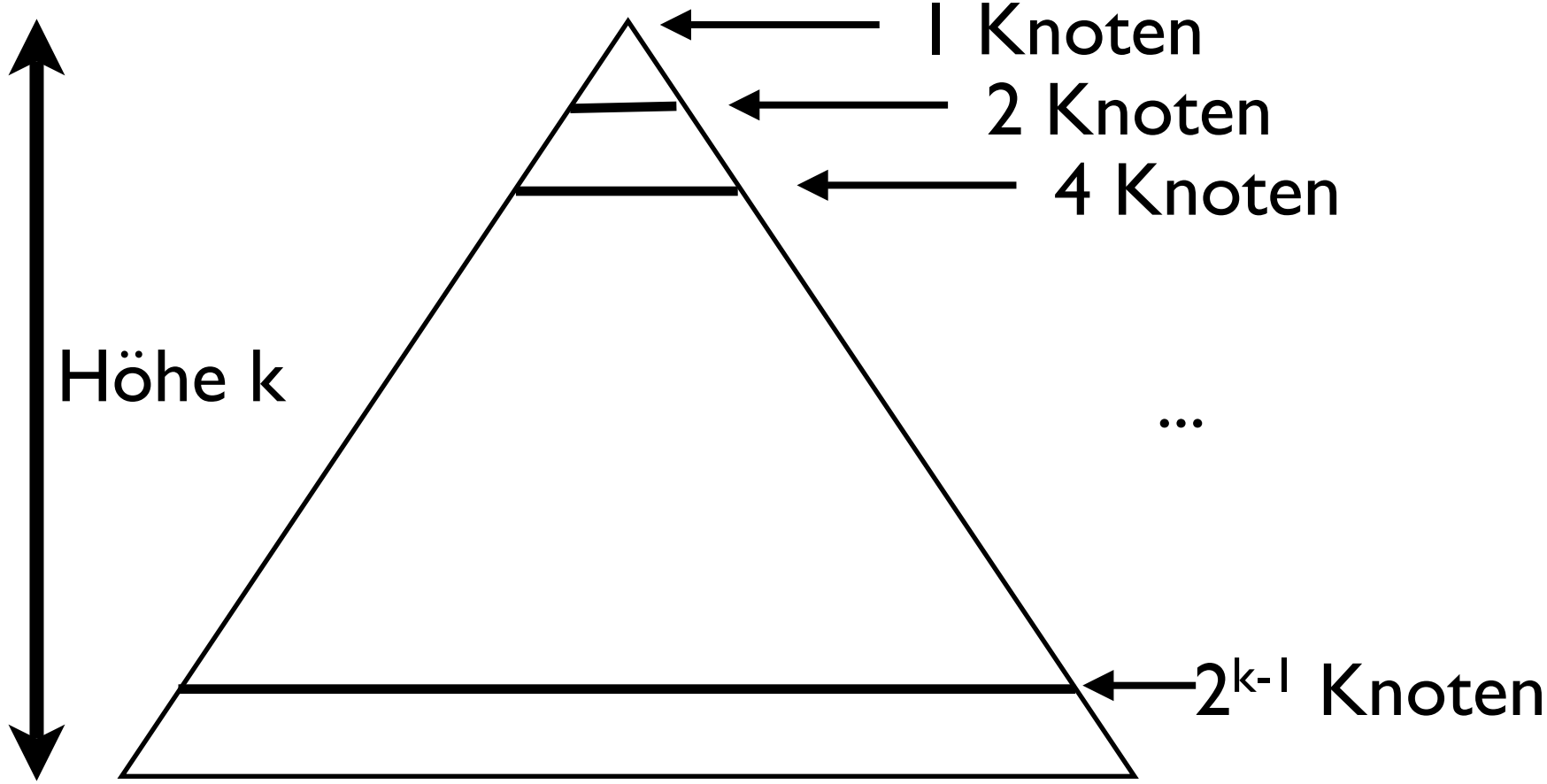
binäre Suchbäume - Laufzeitanalyse



bester Fall: Baum vollständig ausgeglichen

Suchen

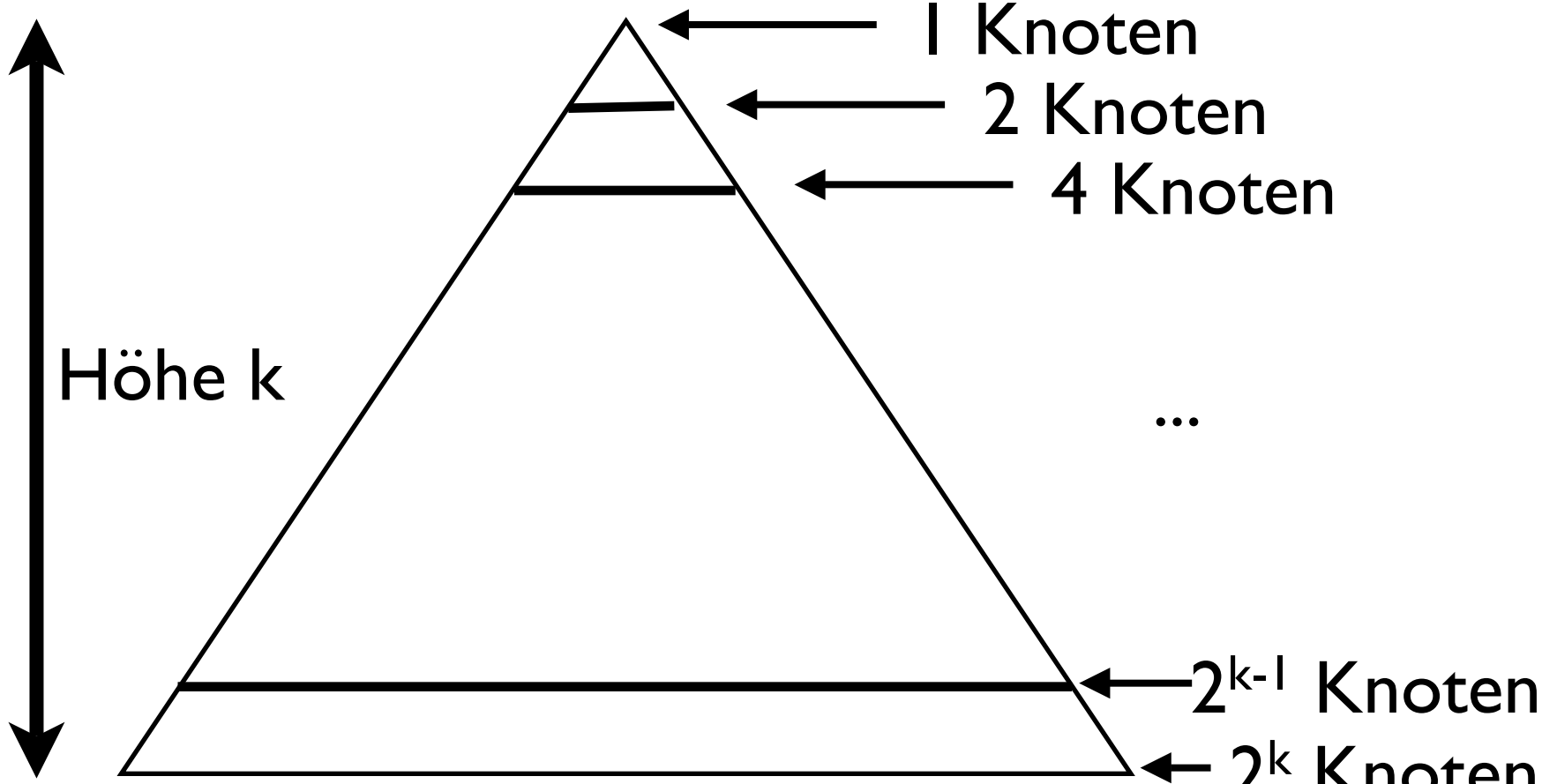
binäre Suchbäume - Laufzeitanalyse



bester Fall: Baum vollständig ausgeglichen

Suchen

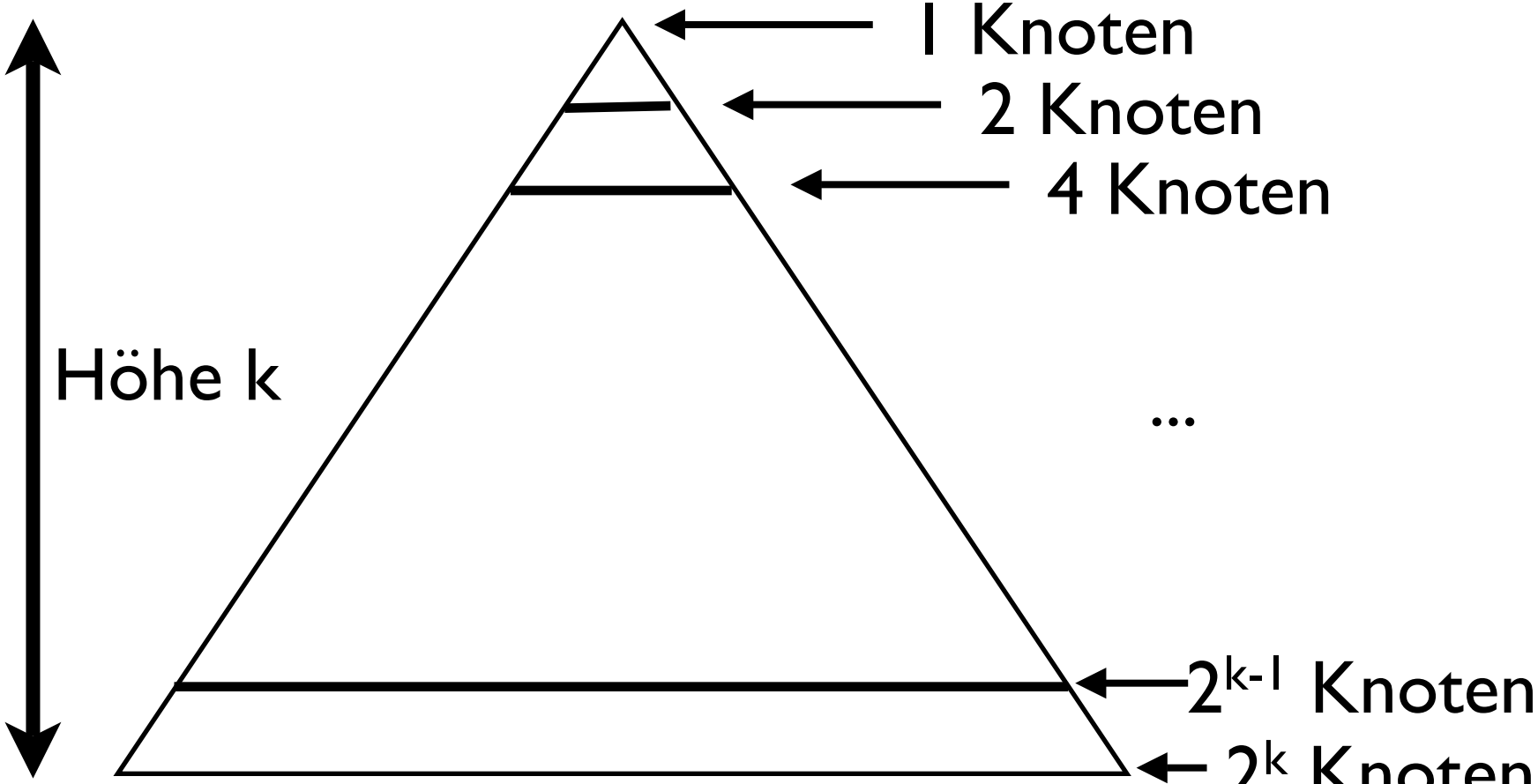
binäre Suchbäume - Laufzeitanalyse



bester Fall: Baum vollständig ausgeglichen

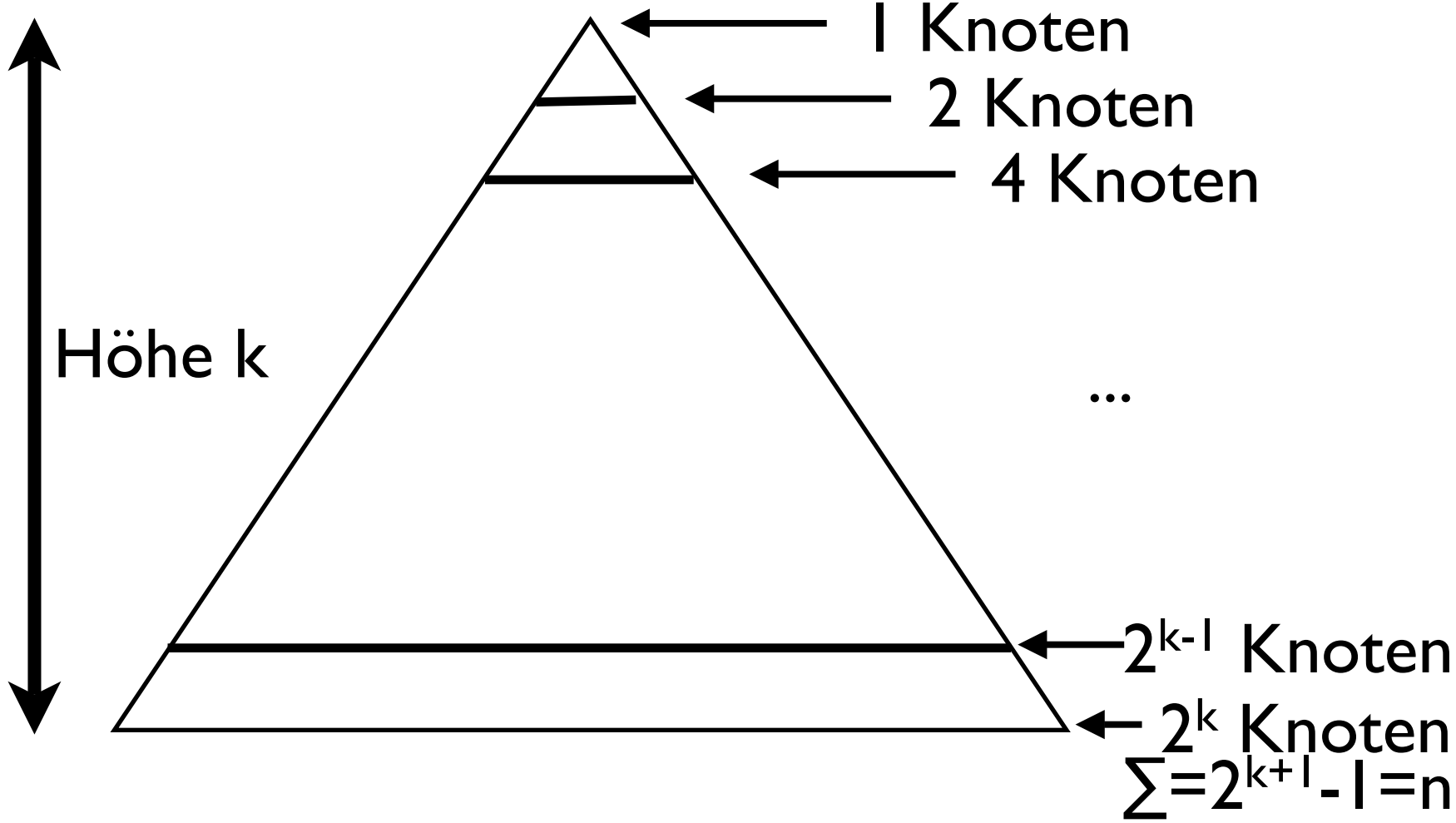
Suchen

binäre Suchbäume - Laufzeitanalyse



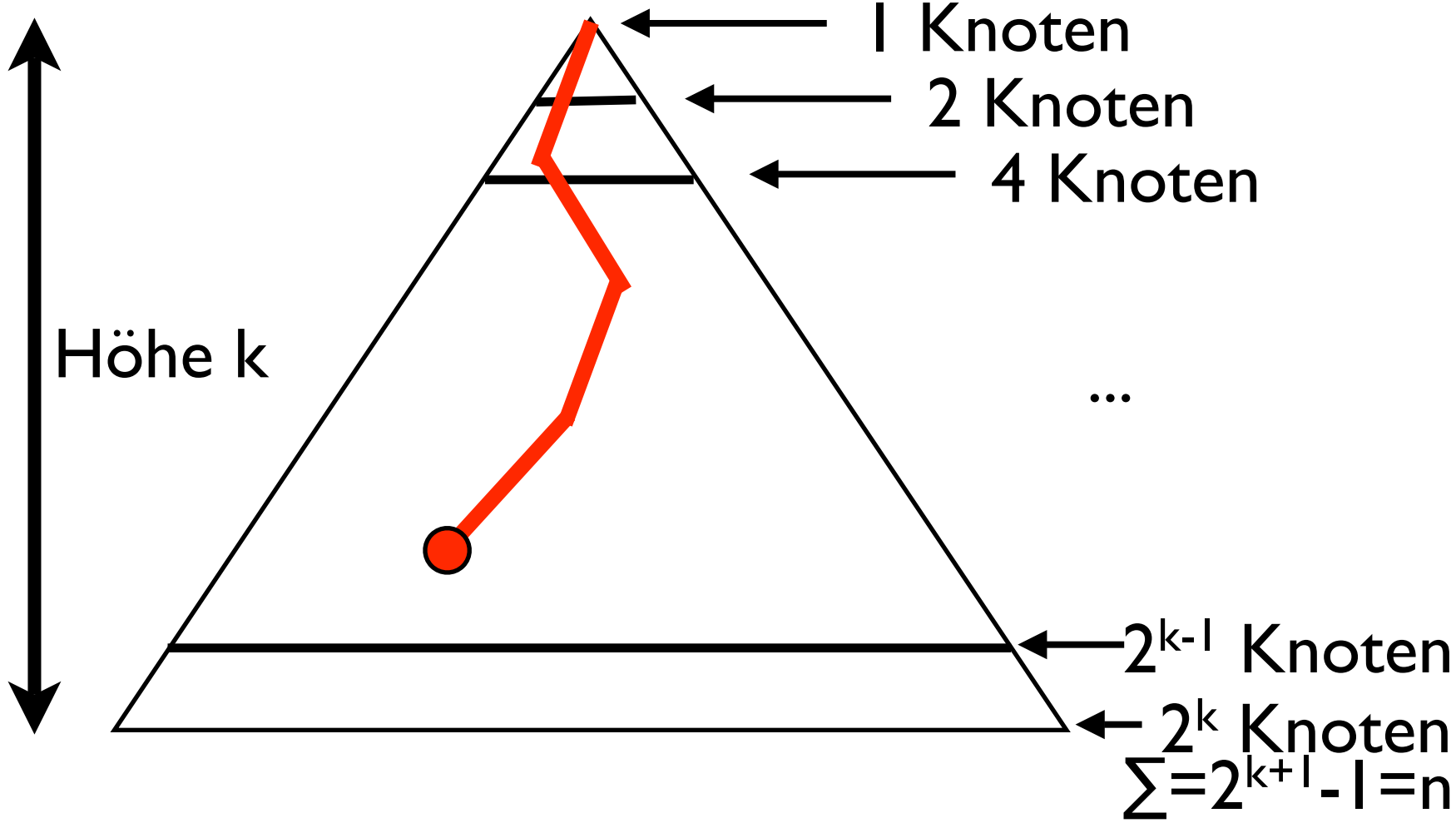
Suchen

binäre Suchbäume - Laufzeitanalyse



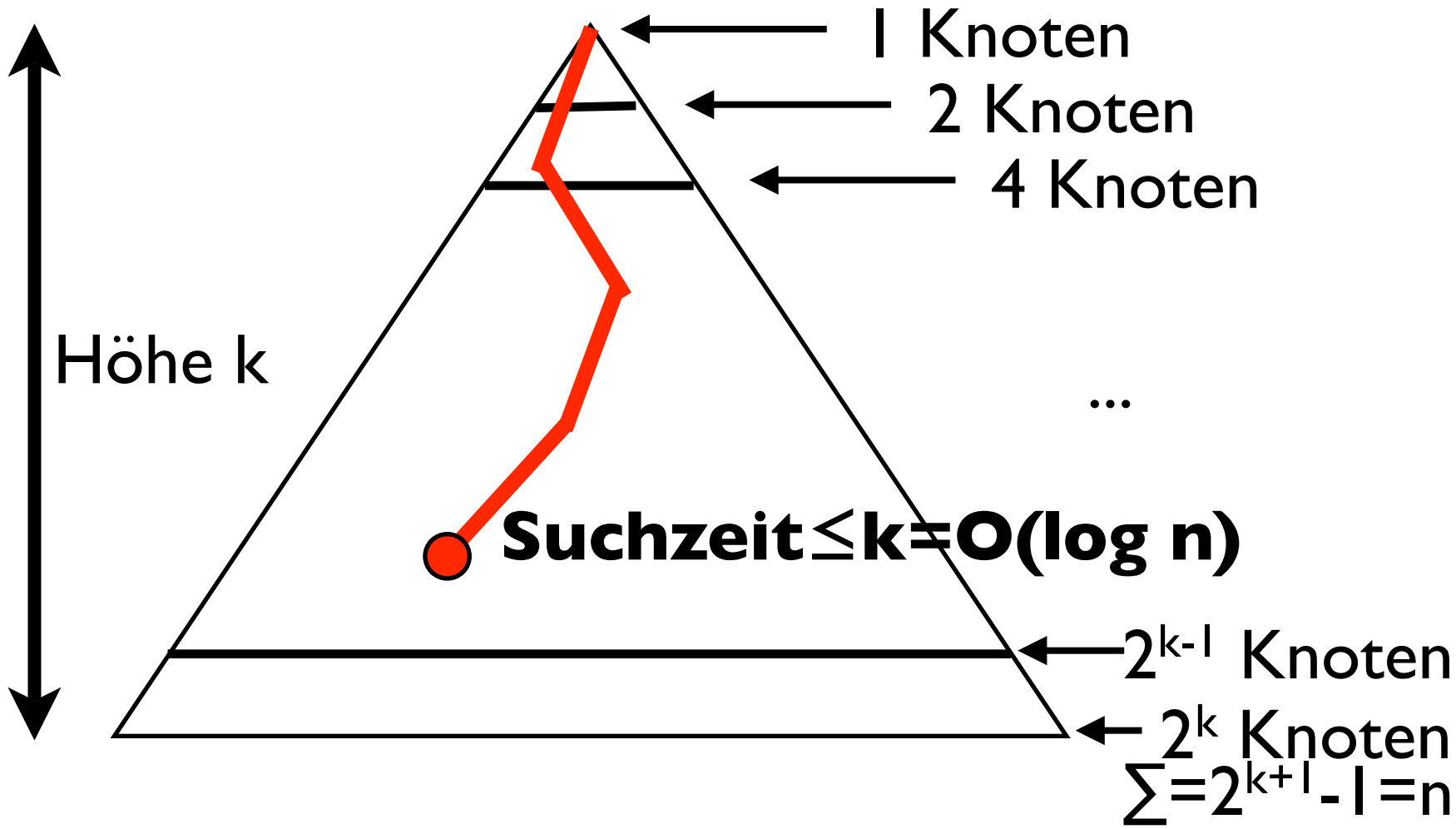
Suchen

binäre Suchbäume - Laufzeitanalyse



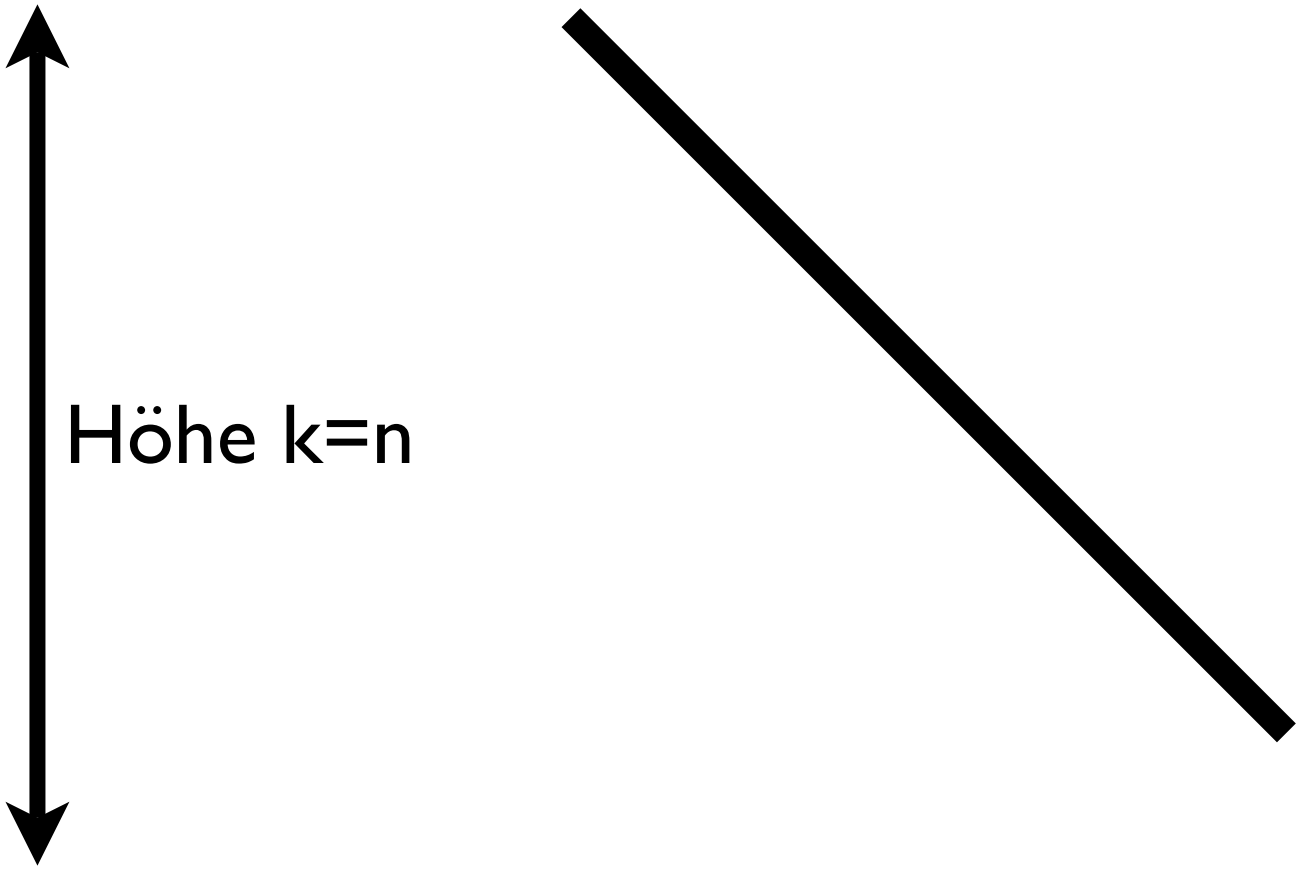
Suchen

binäre Suchbäume - Laufzeitanalyse



Suchen

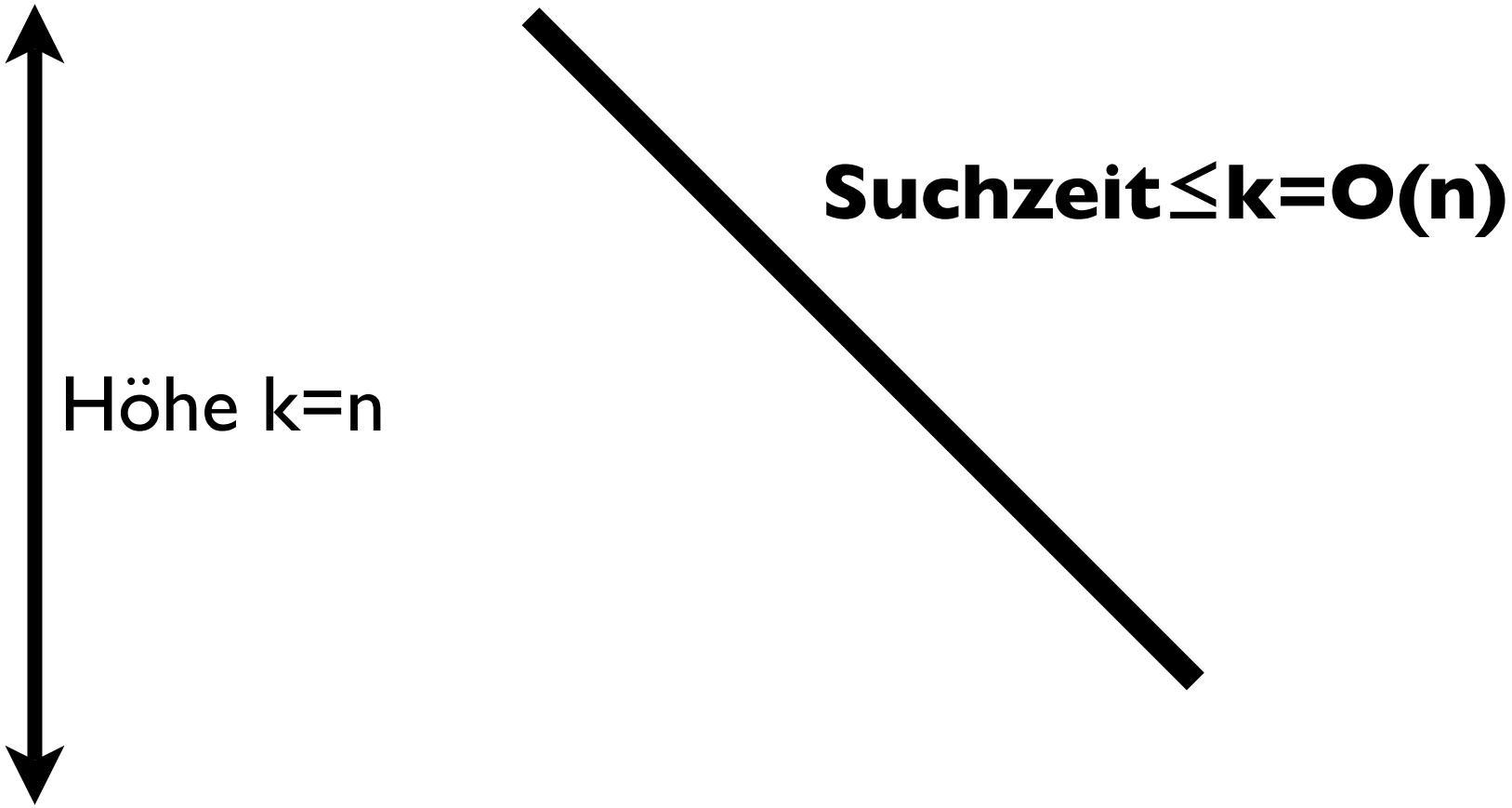
binäre Suchbäume - Laufzeitanalyse



schlimmster Fall: Baum degeneriert

Suchen

binäre Suchbäume - Laufzeitanalyse



schlimmster Fall: Baum degeneriert

Suchen
binäre Suchbäume - Analyse

- Suchen, Einfügen, Löschen: O(log n) bis O(n)
- Speicherbedarf: O(1) bei nicht-rekursiver Implementierung, sonst O(log n) bis O(n)
- Glückssache: degenerierte Bäume sind selten. Man kann zeigen:
  - sie entstehen nicht zufällig
  - Werden Schlüssel zufällig eingegeben, sind im Mittel 2 ln n Vergleiche nötig
- Dennoch: Wie verhindert man degenerierte Bäume? Lösung: AVL-Bäume

Suchen
ausgeglichene Suchbäume

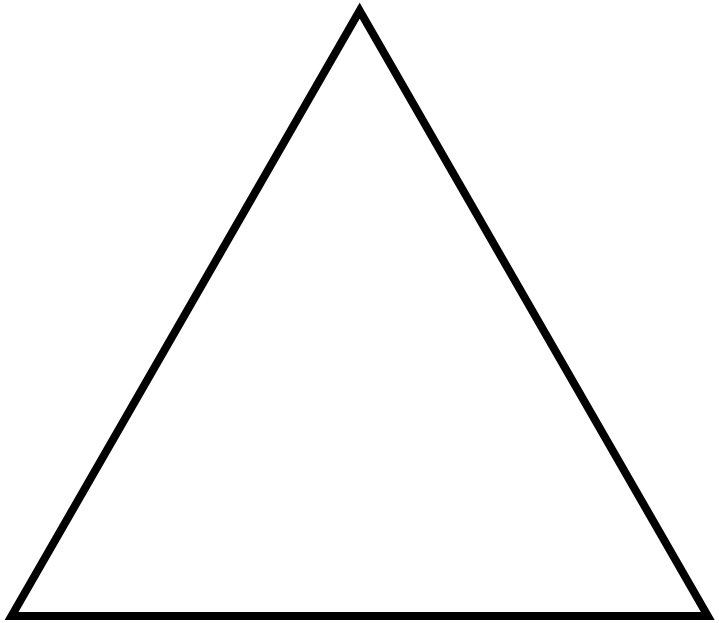
- Situation: worst-case Laufzeit bei Suchbäumen ↔ Länge des längsten Weges von Wurzel zu Blatt
- Idee: rücke alle Knoten möglichst nah zur Wurzel
- Problem: Bei dynamischer Änderung des Baumes kann die Eigenschaft verloren gehen, der Baum wieder mehr und mehr degenerieren

Suchen
ausgeglichene Suchbäume

- Naiver Zugang:
  - gleiche Suchbaum nach jeder Veränderung wieder vollständig aus. Konsequenz: erheblicher Effizienzverlust.
- Besser:
  - statt vollständiger, reicht “gewisse” Ausgeglichenheit
  - Übergang zu “fast” ausgeglichenen Bäumen

Suchen

ausgeglichene Suchbäume

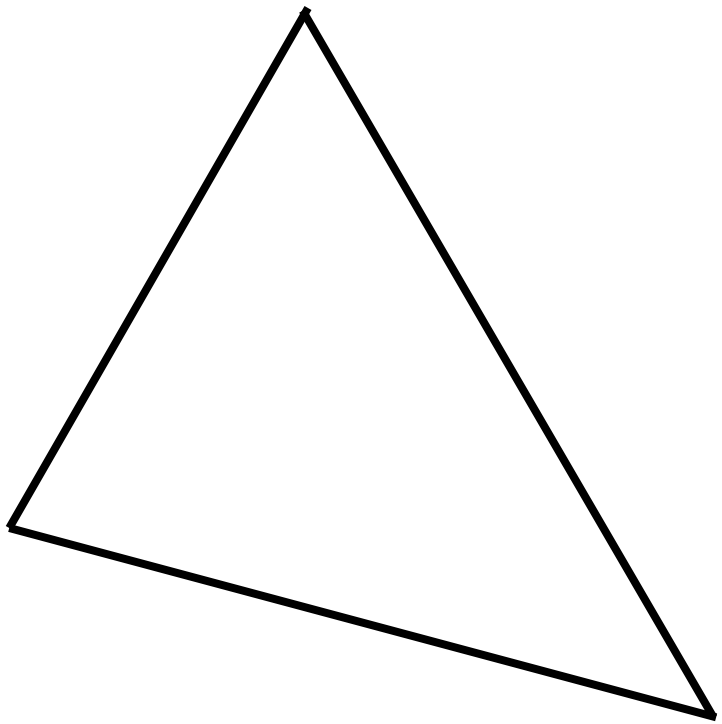


Suchen

ausgeglichene Suchbäume

Suchen

ausgeglichene Suchbäume

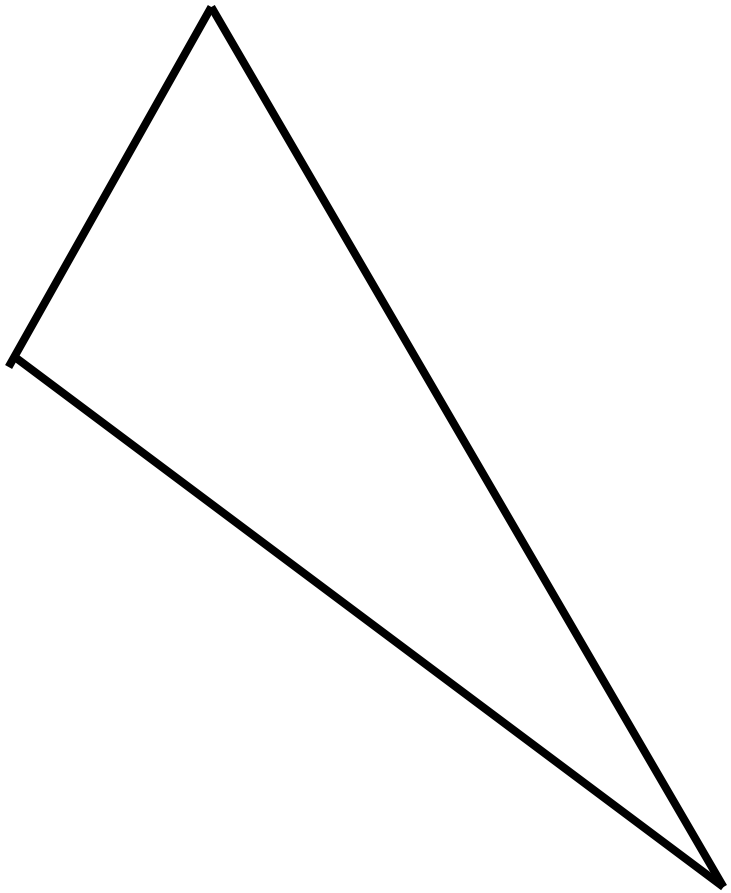


Suchen

ausgeglichene Suchbäume

Suchen

ausgeglichene Suchbäume

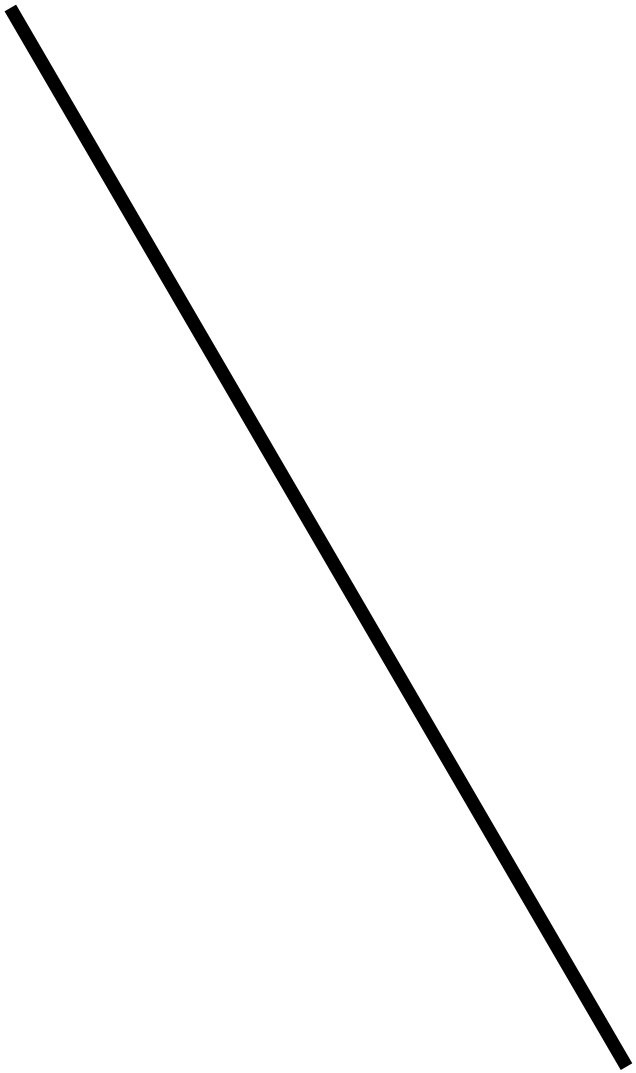


Suchen

ausgeglichene Suchbäume

Suchen

ausgeglichene Suchbäume



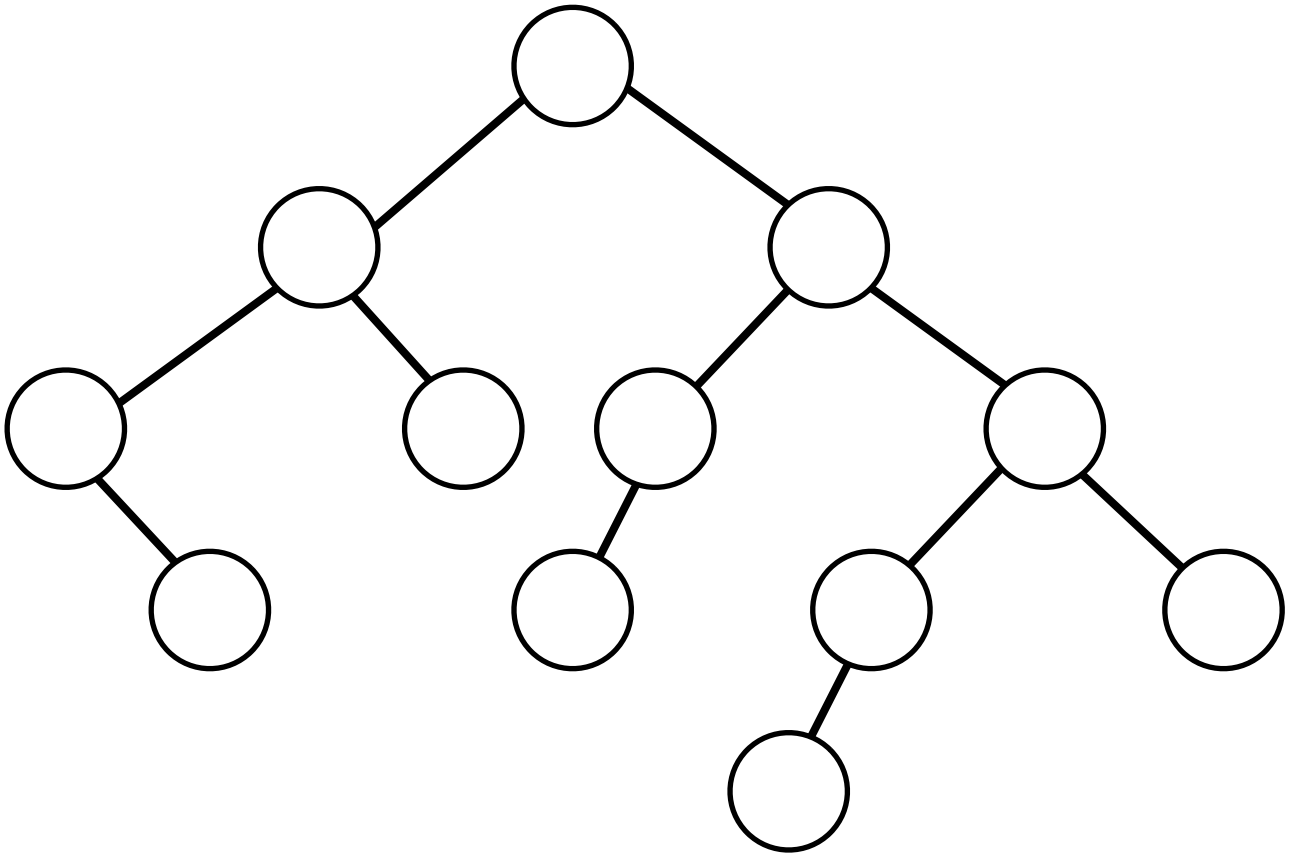
Suchen
ausgeglichene Suchbäume - AVL-Bäume

Definition

Ein binärer Baum heißt **ausgeglichener Baum** oder **AVL-Baum** (Adelson-Velskij und Landis), falls sich für jeden Knoten s die Höhen der beiden Teilbäume um höchstens 1 unterscheiden.

Suchen

AVL-Bäume - Beispiel

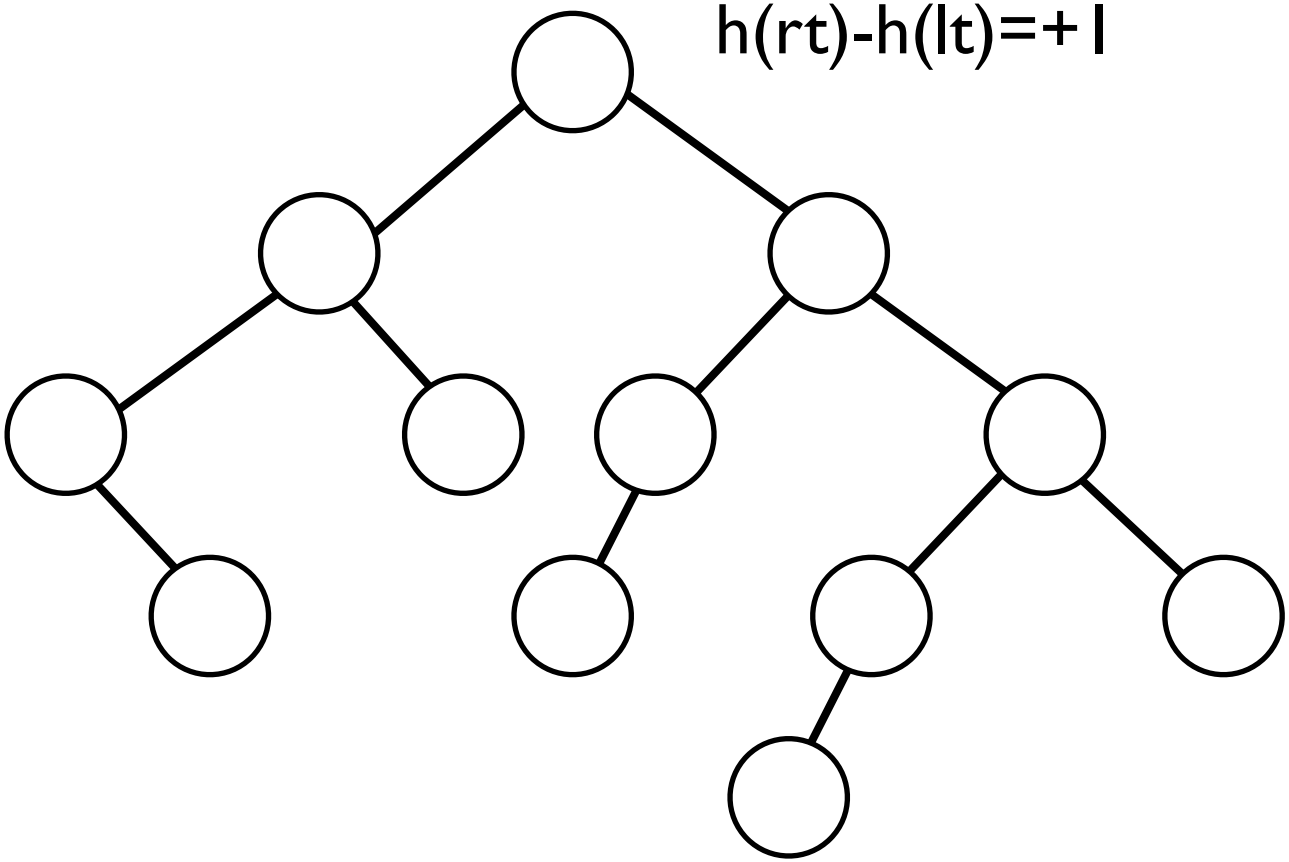


Suchen

AVL-Bäume - Beispiel

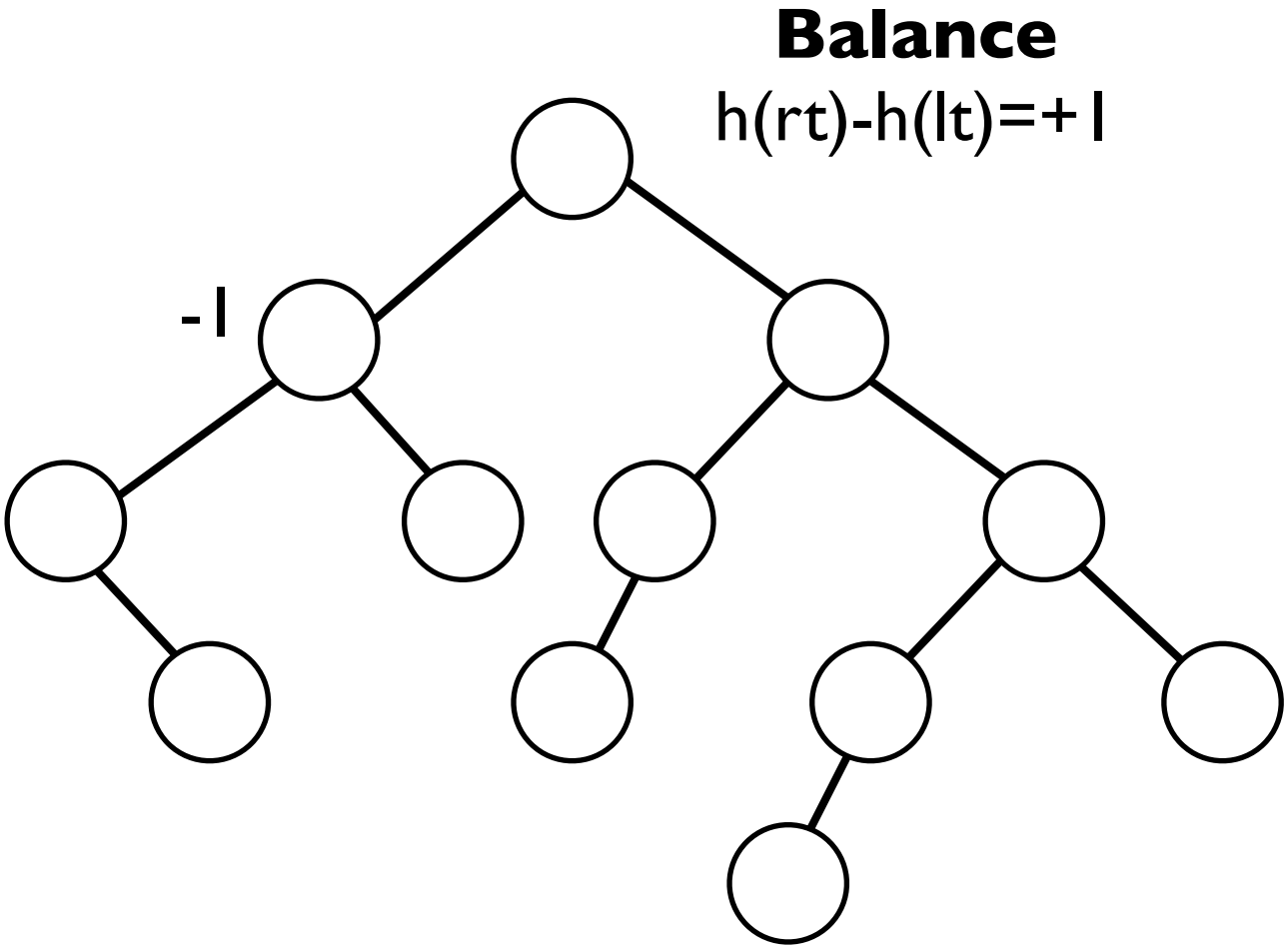
Balance

h(rt)-h(lt)=+1



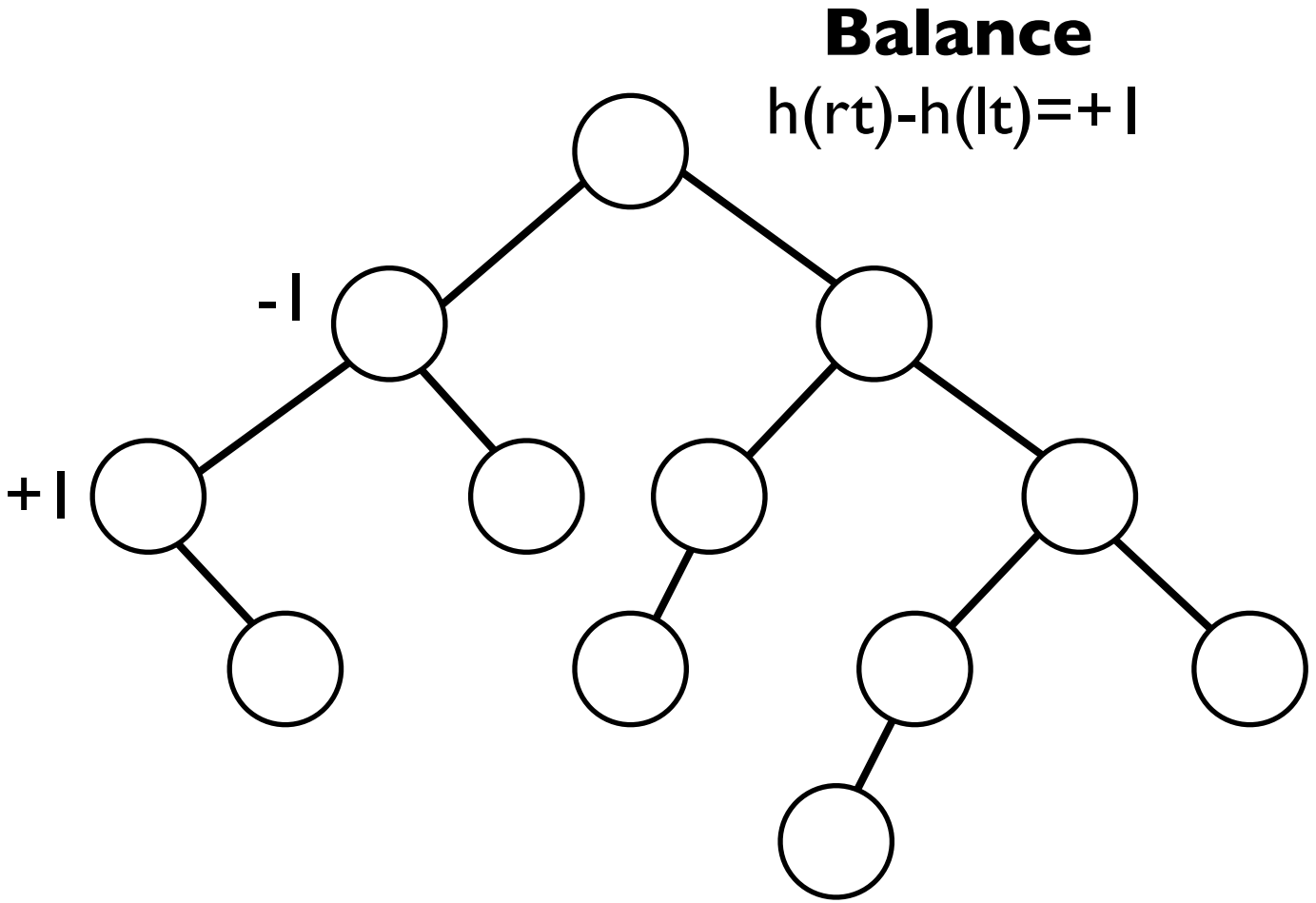
Suchen

AVL-Bäume - Beispiel



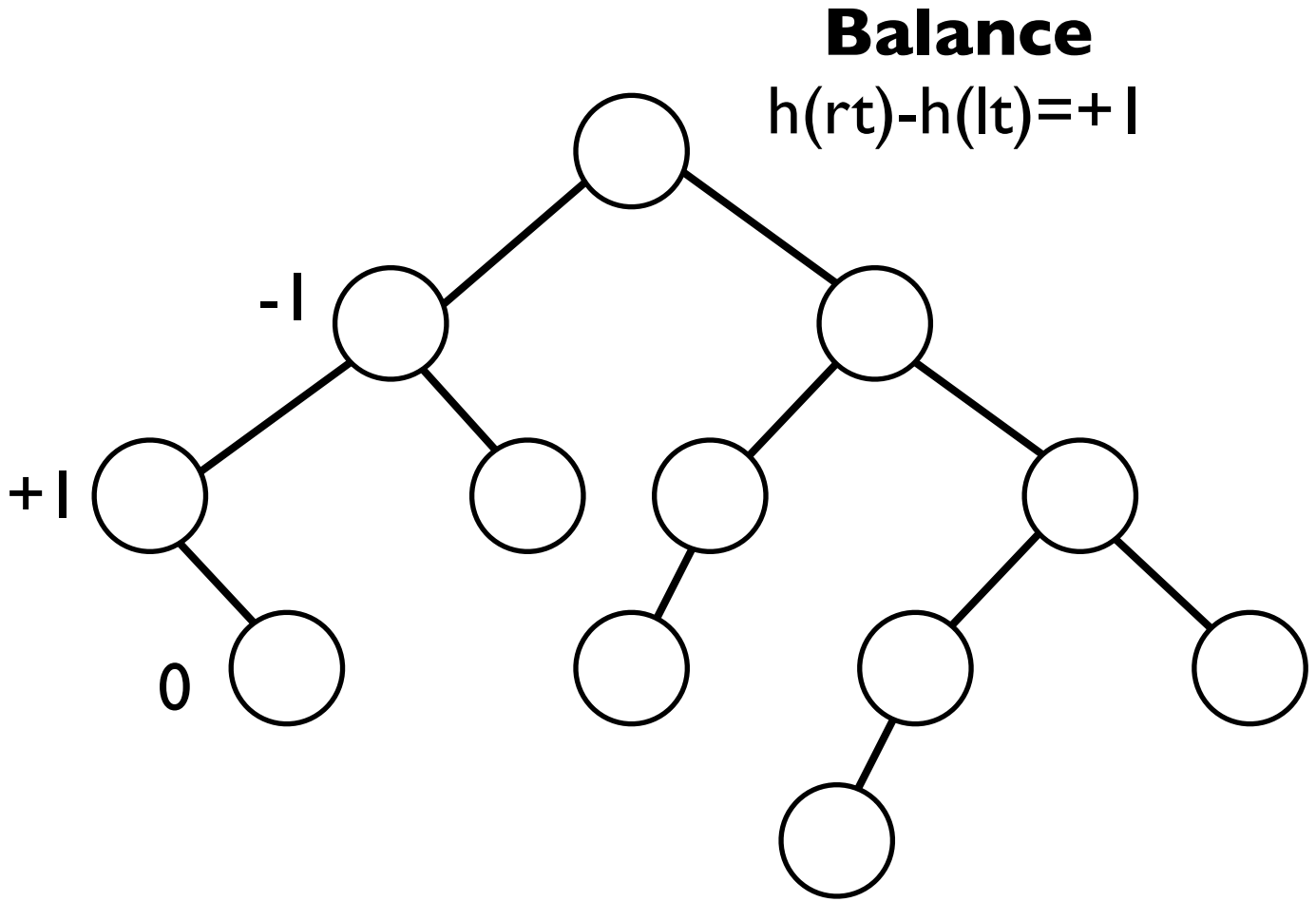
Suchen

AVL-Bäume - Beispiel



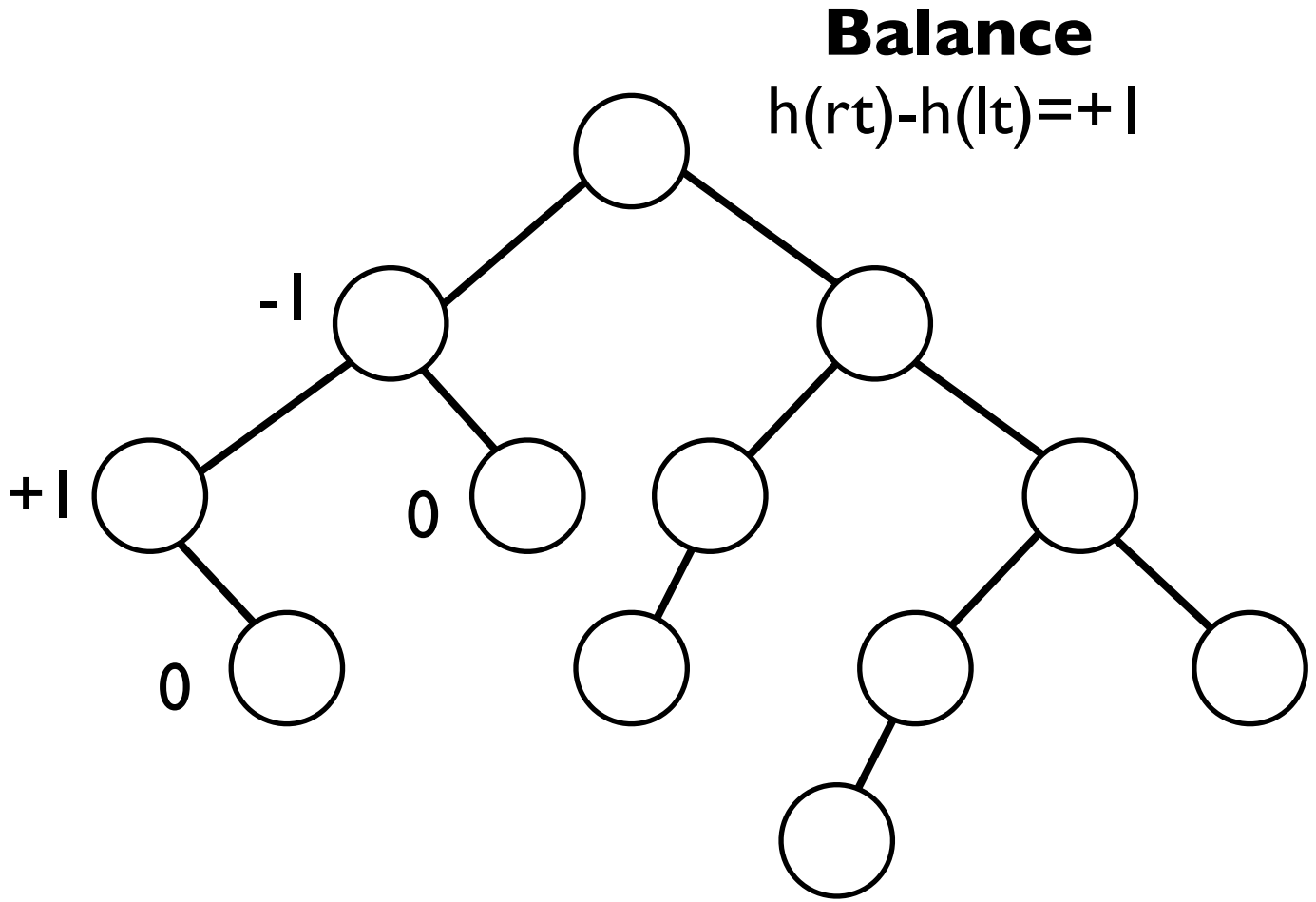
Suchen

AVL-Bäume - Beispiel



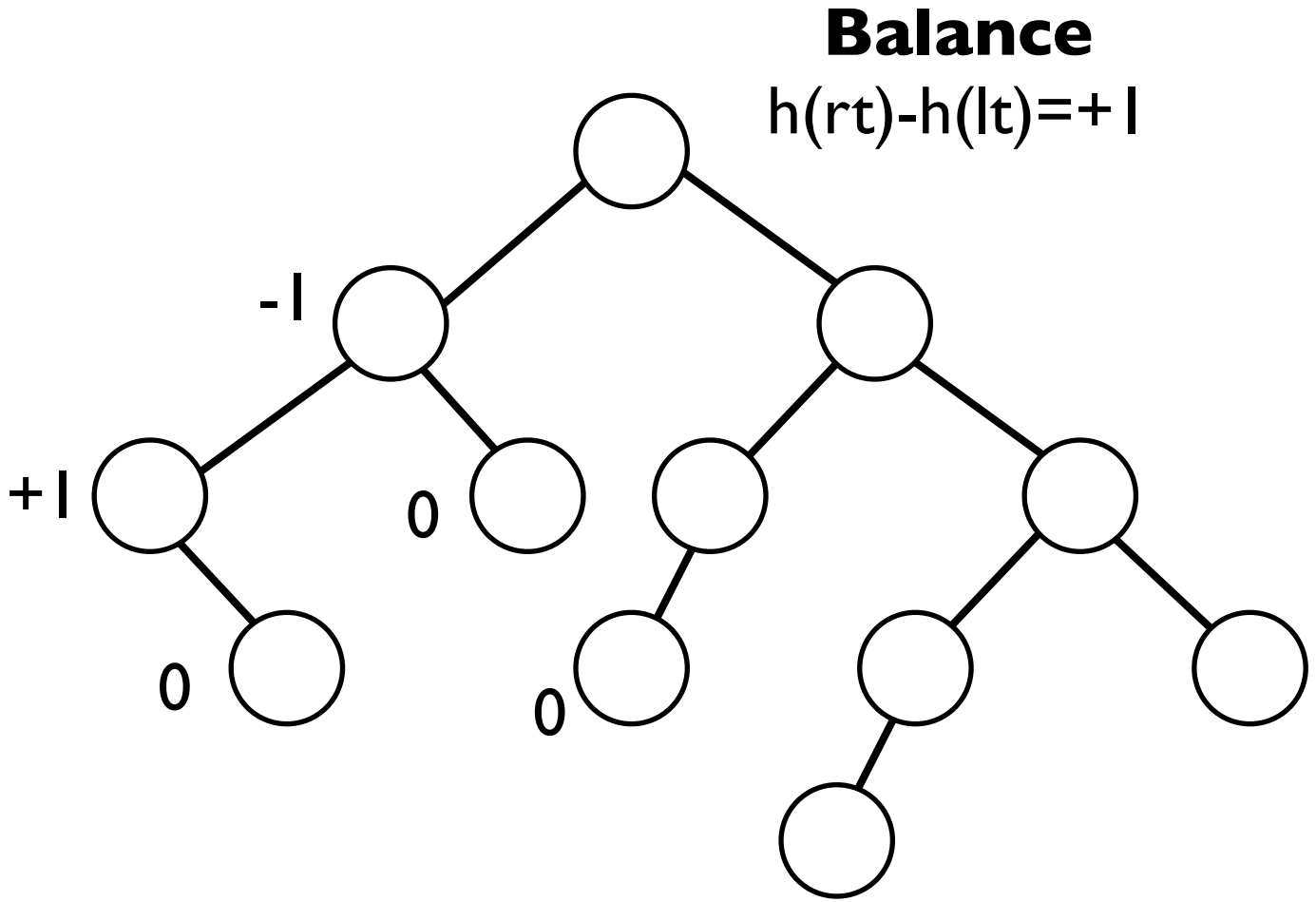
Suchen

AVL-Bäume - Beispiel



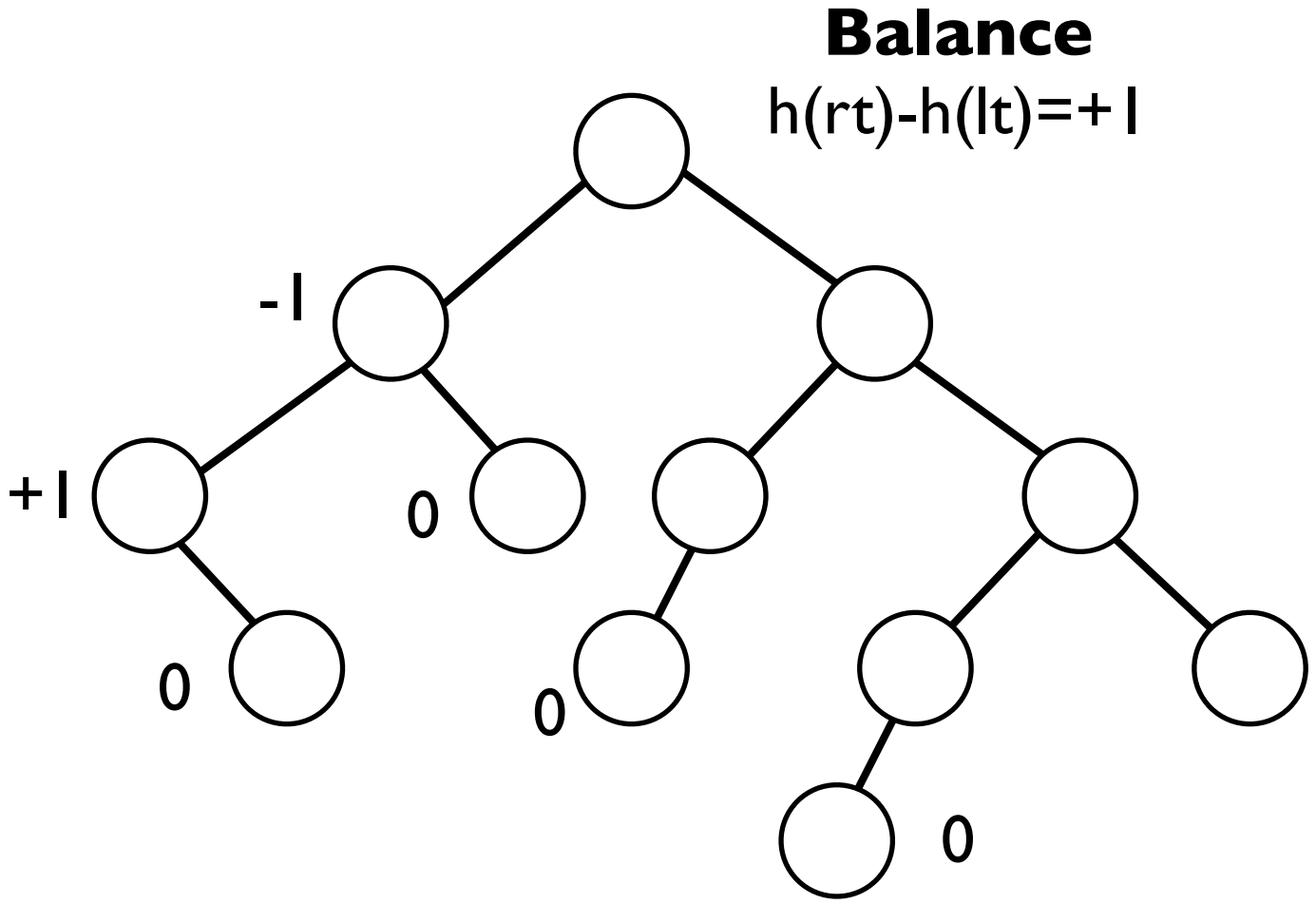
Suchen

AVL-Bäume - Beispiel



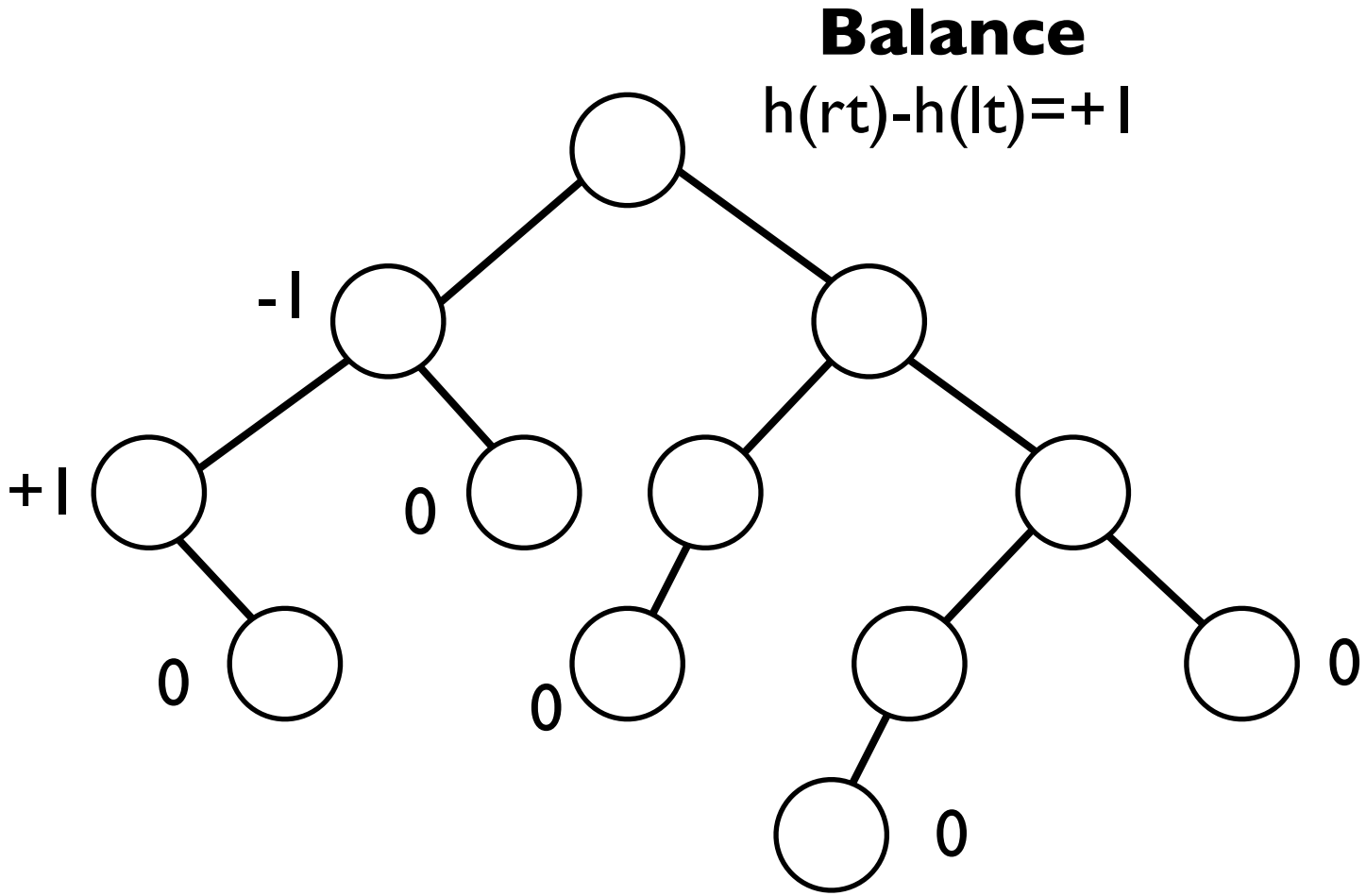
Suchen

AVL-Bäume - Beispiel



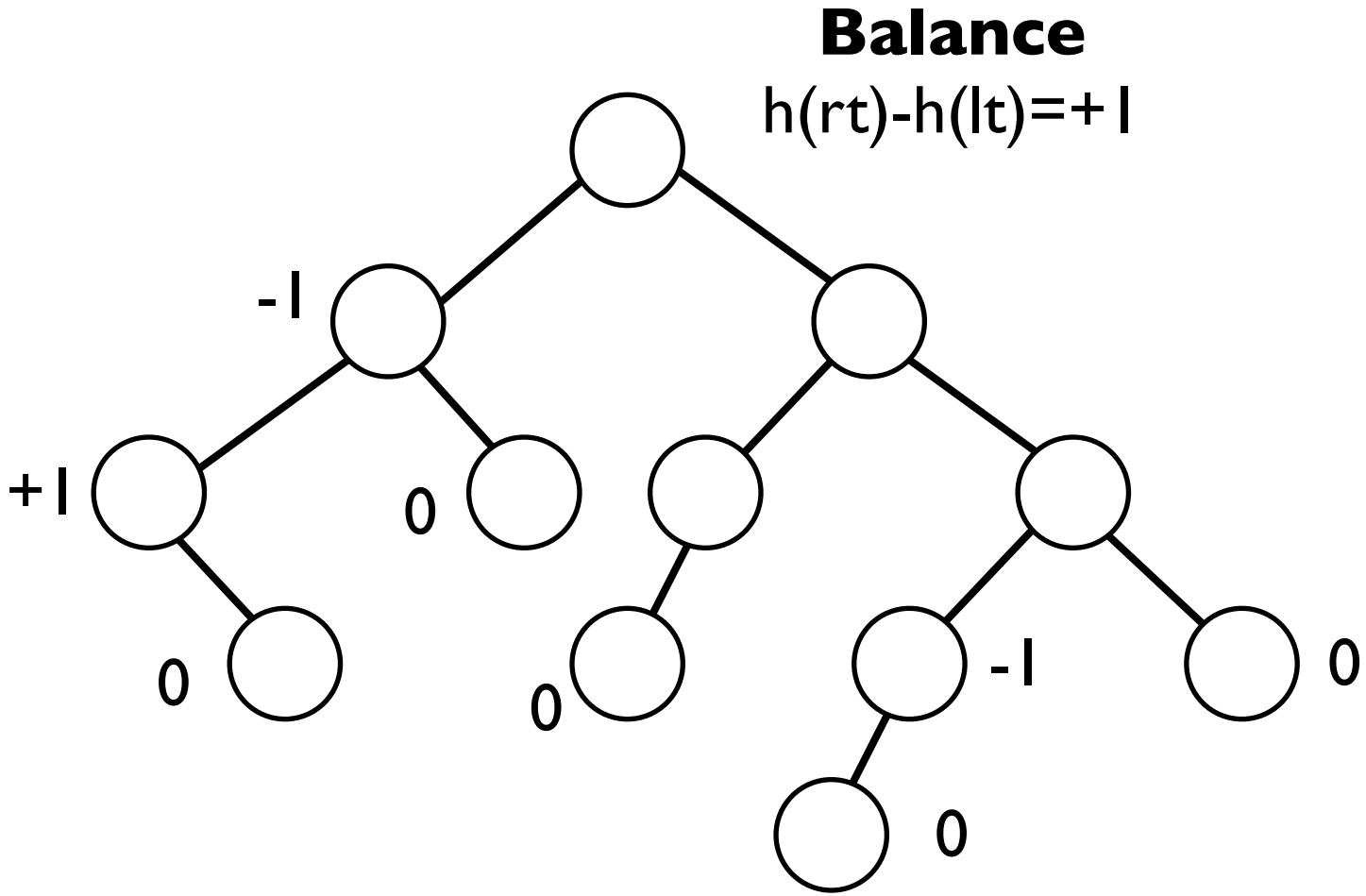
Suchen

AVL-Bäume - Beispiel



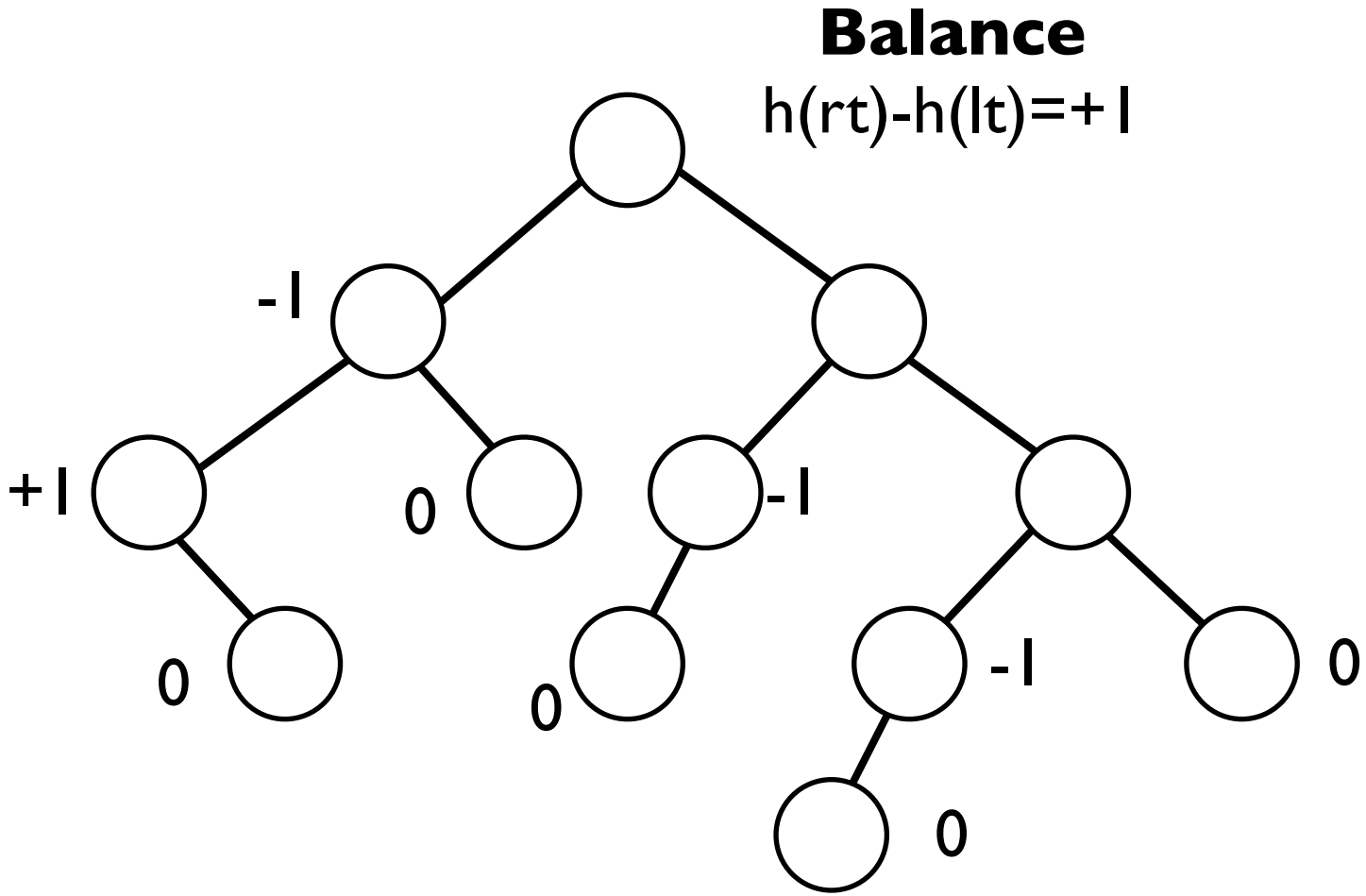
Suchen

AVL-Bäume - Beispiel



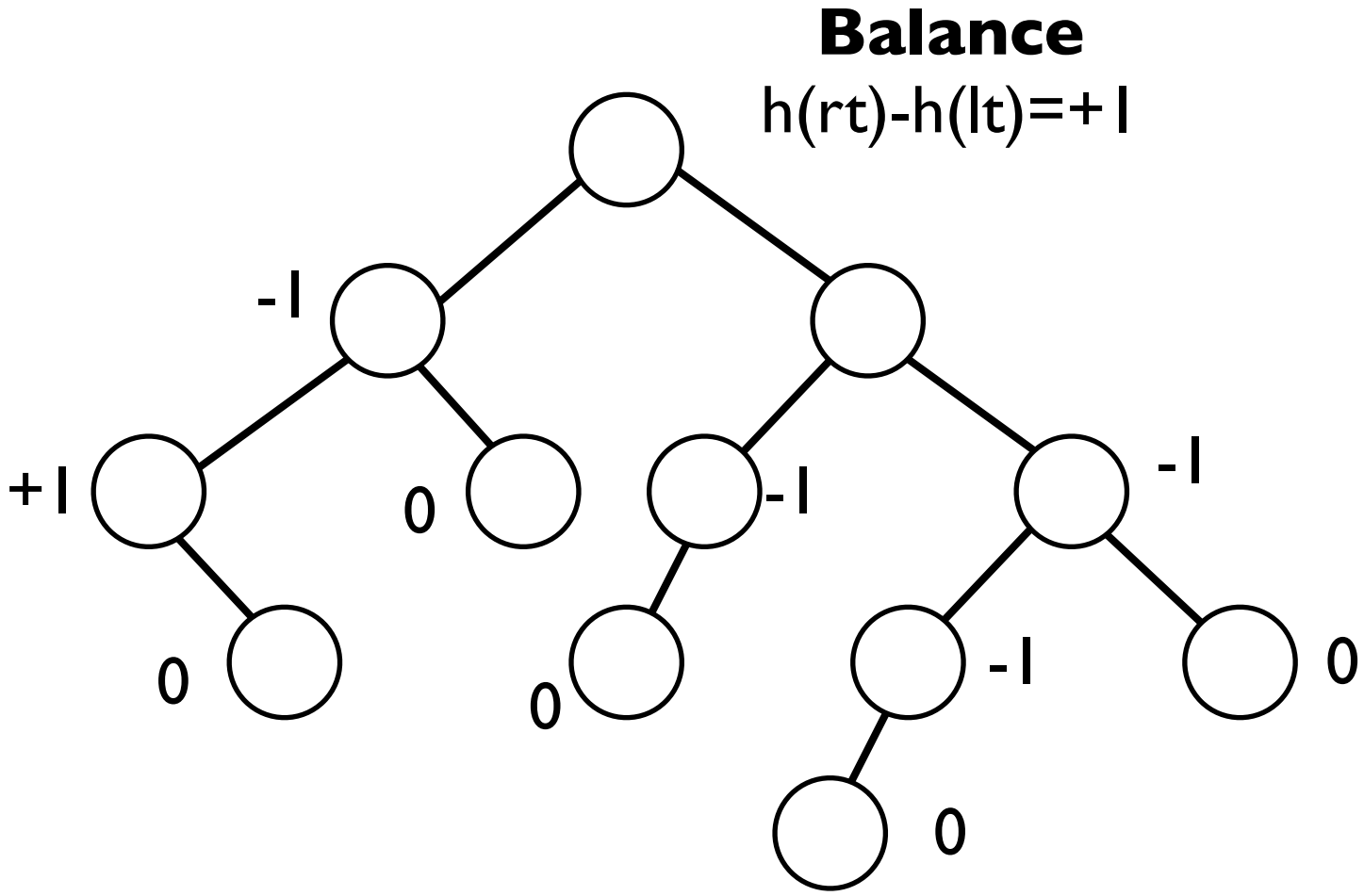
Suchen

AVL-Bäume - Beispiel



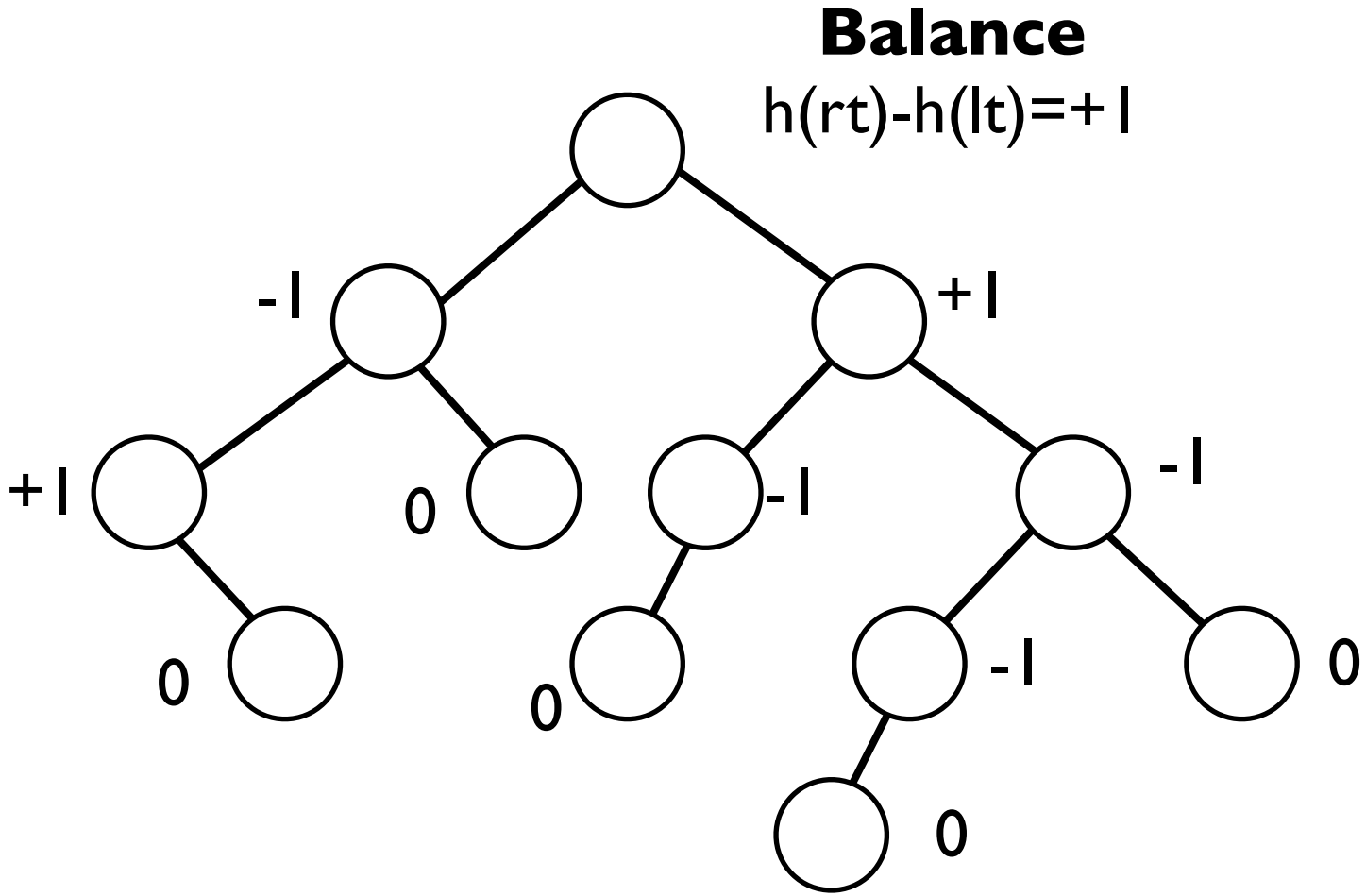
Suchen

AVL-Bäume - Beispiel



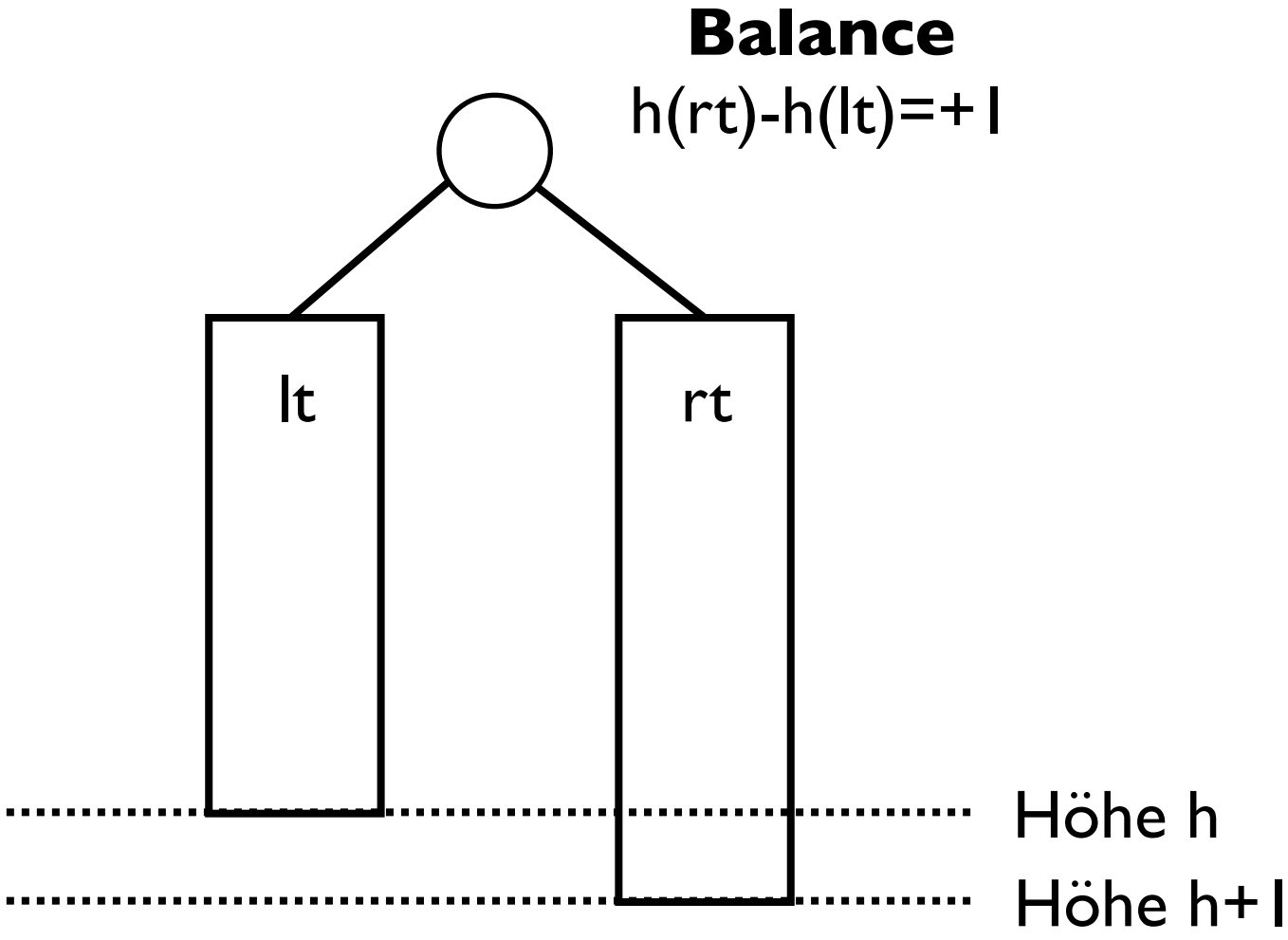
Suchen

AVL-Bäume - Beispiel



Suchen

AVL-Bäume - schematisch

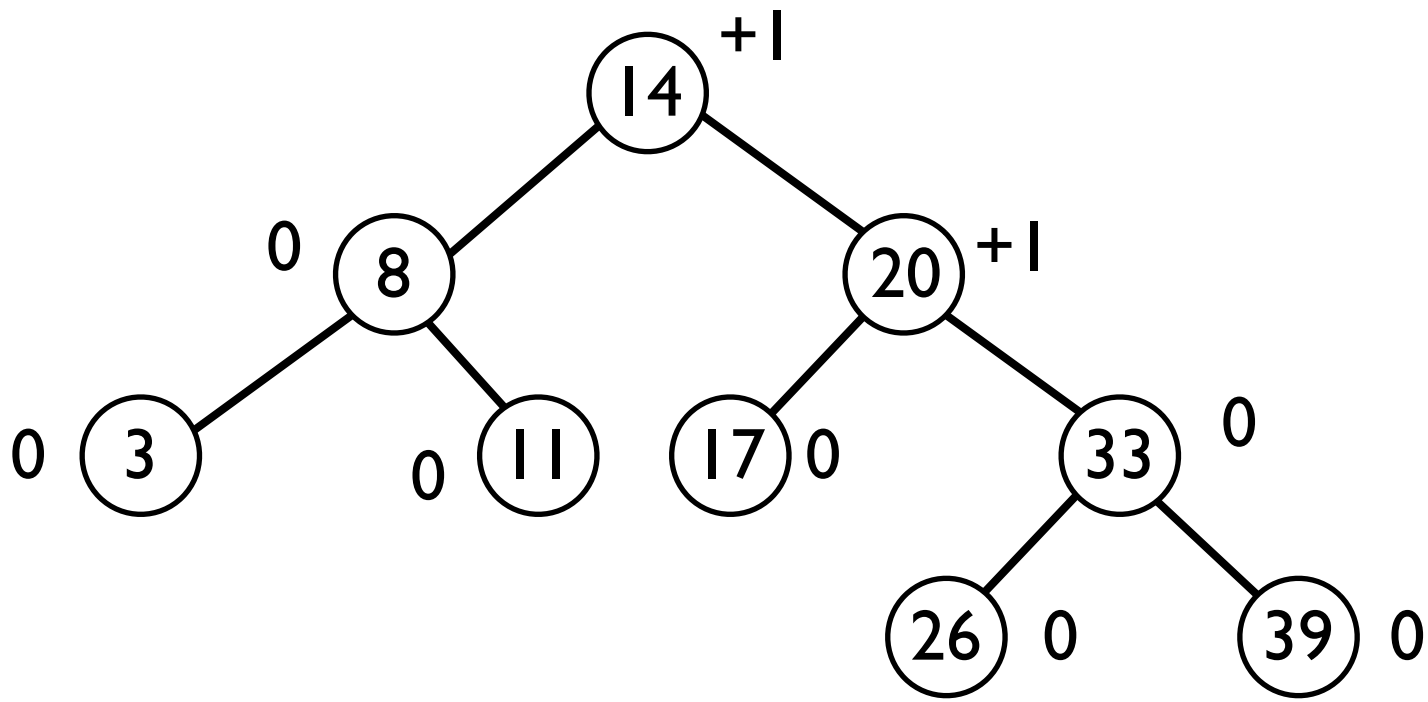


Suchen
AVL-Bäume - Operationen

- Suchen: genau wie bisher
- Einfügen:
  - genau wie bisher
  - anschließend Baum ausbalancieren und AVL-Eigenschaft wieder herstellen

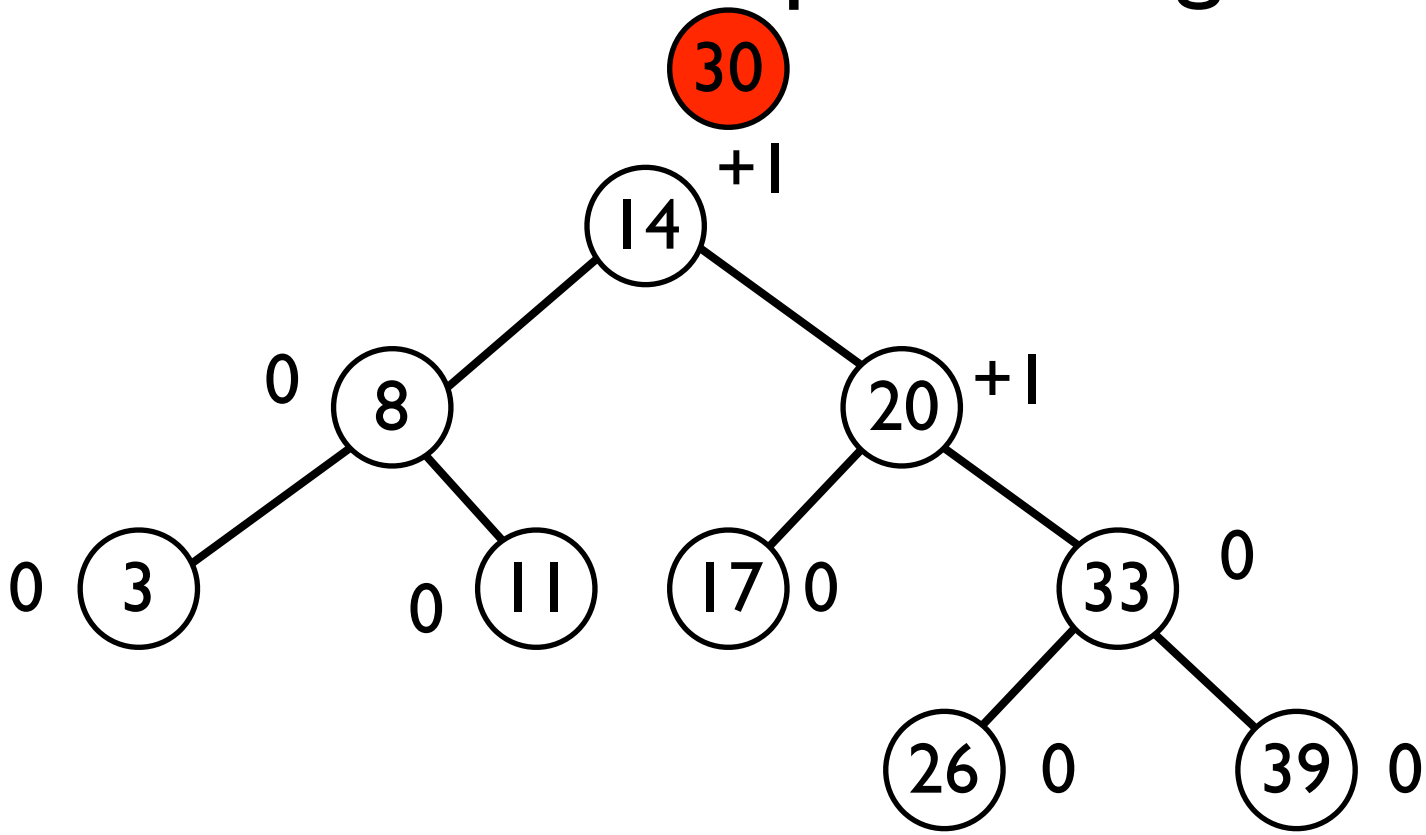
Suchen

AVL-Bäume - Beispiel: Einfügen



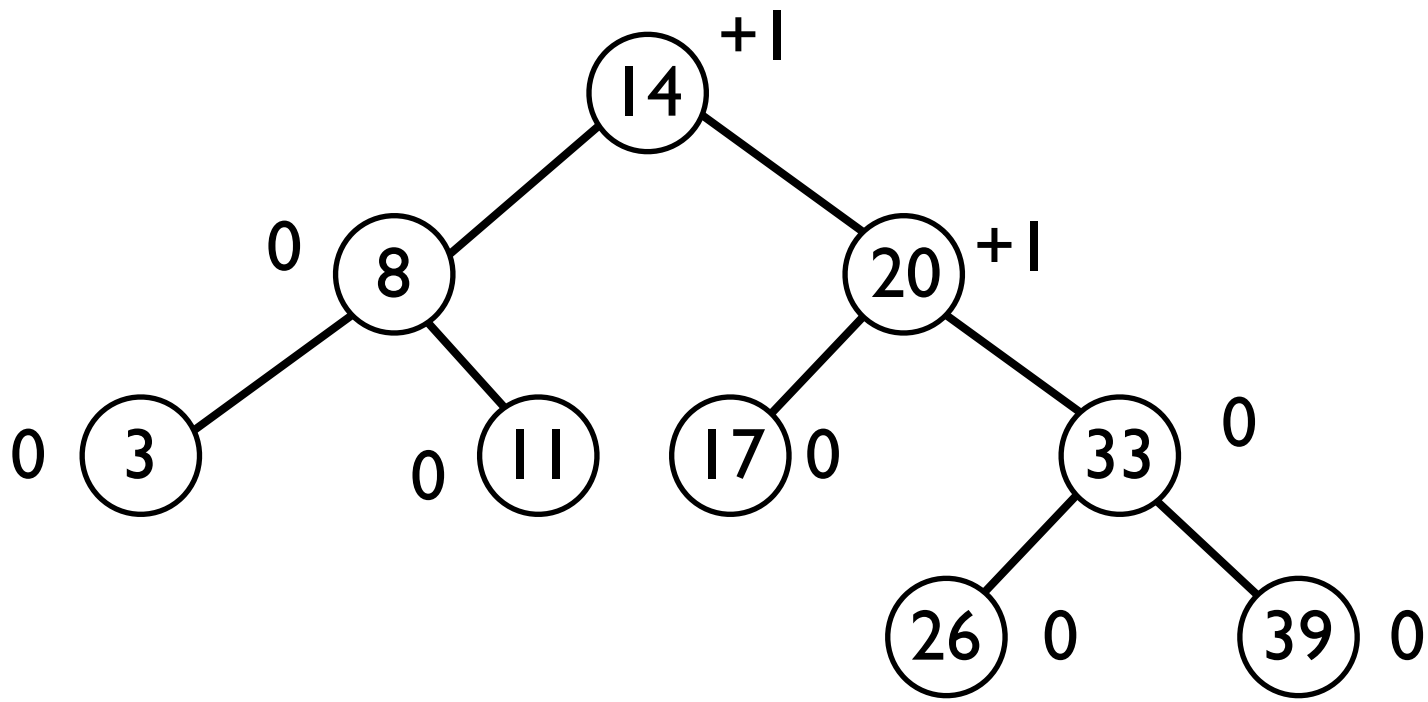
Suchen

AVL-Bäume - Beispiel: Einfügen



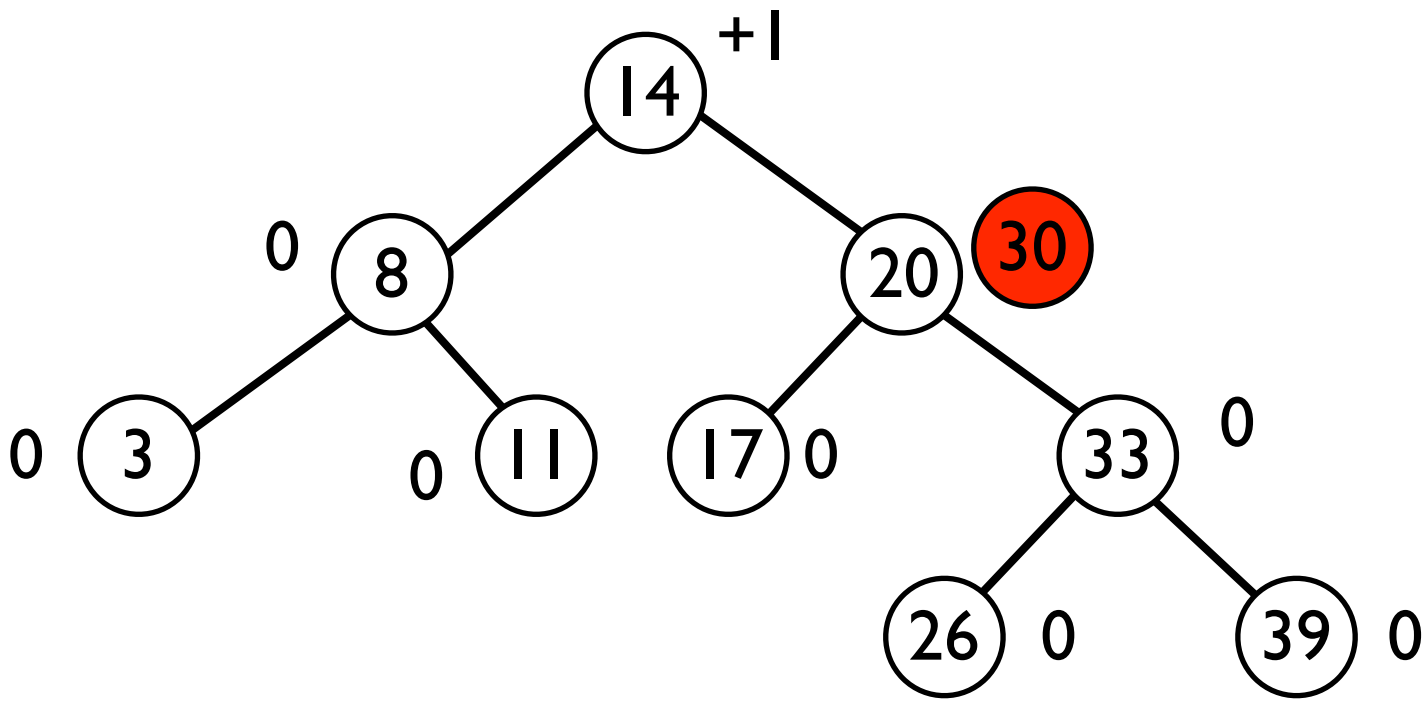
Suchen

AVL-Bäume - Beispiel: Einfügen



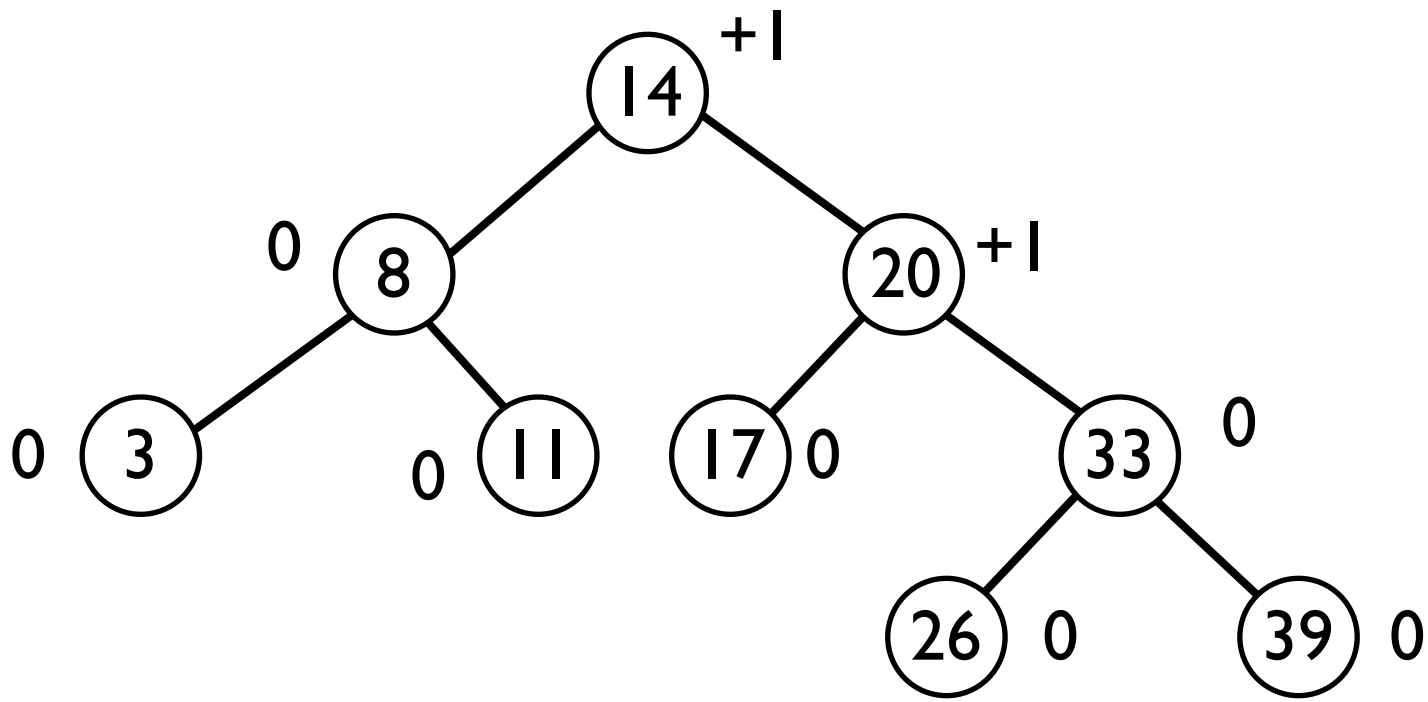
Suchen

AVL-Bäume - Beispiel: Einfügen



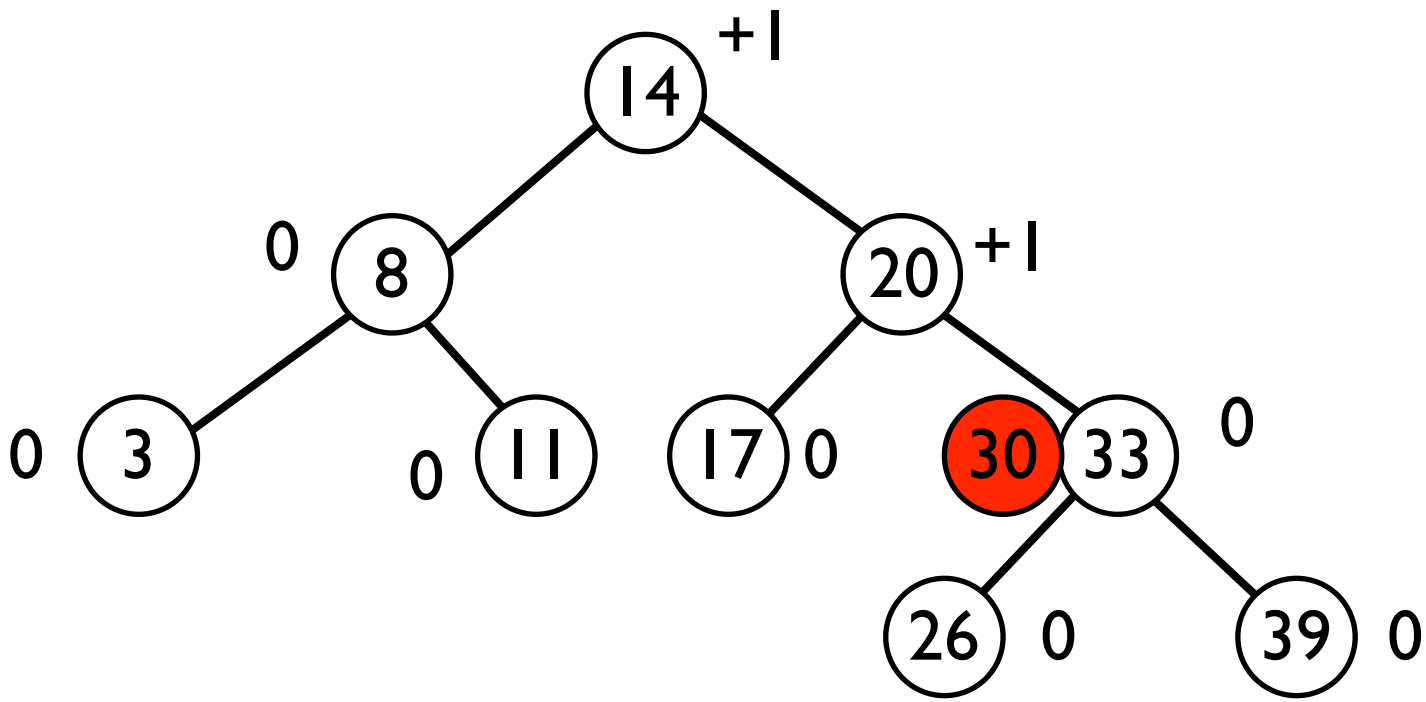
Suchen

AVL-Bäume - Beispiel: Einfügen



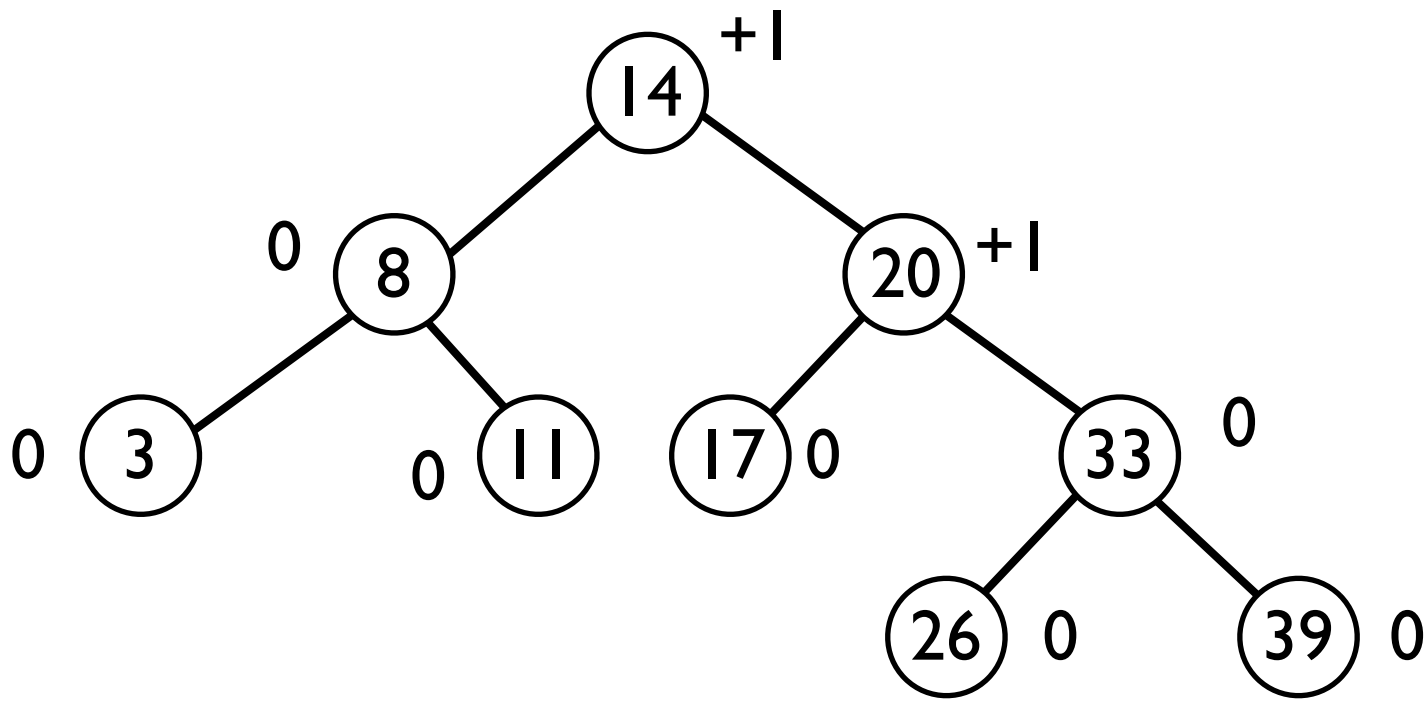
Suchen

AVL-Bäume - Beispiel: Einfügen



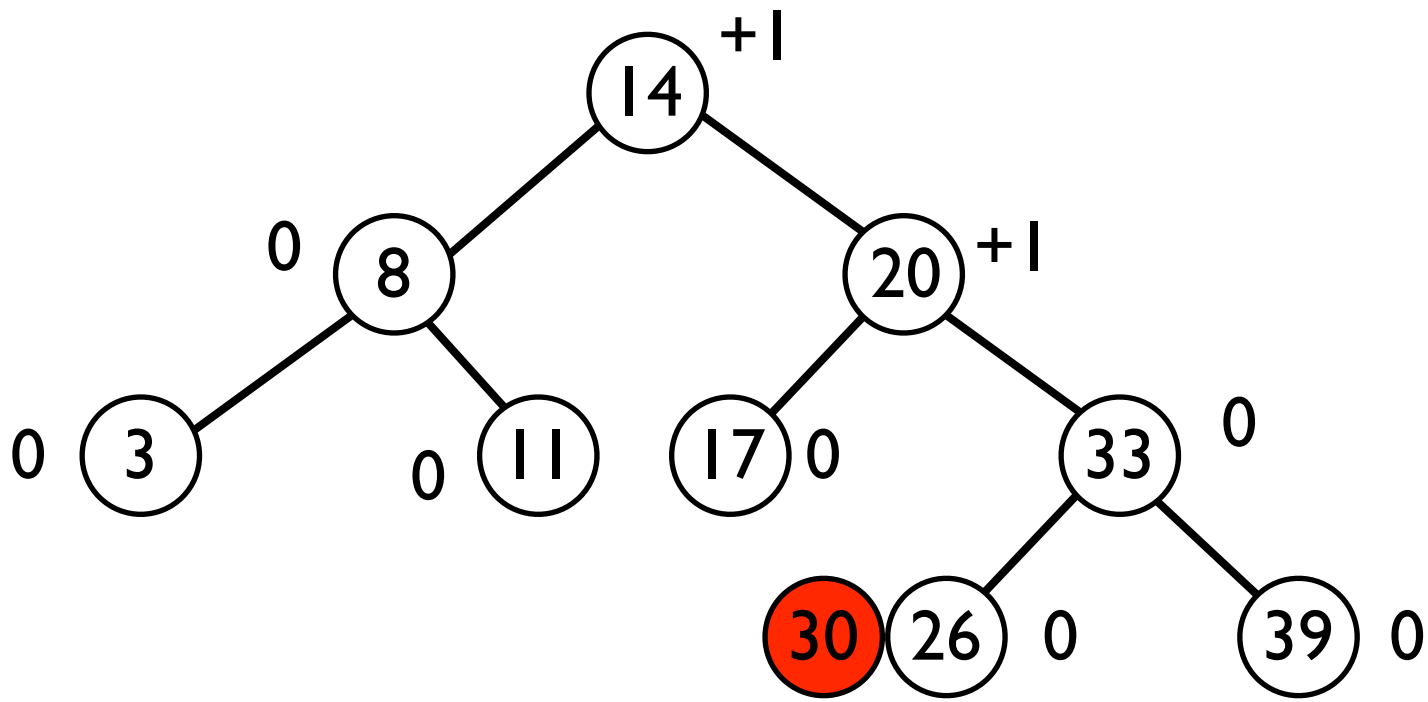
Suchen

AVL-Bäume - Beispiel: Einfügen



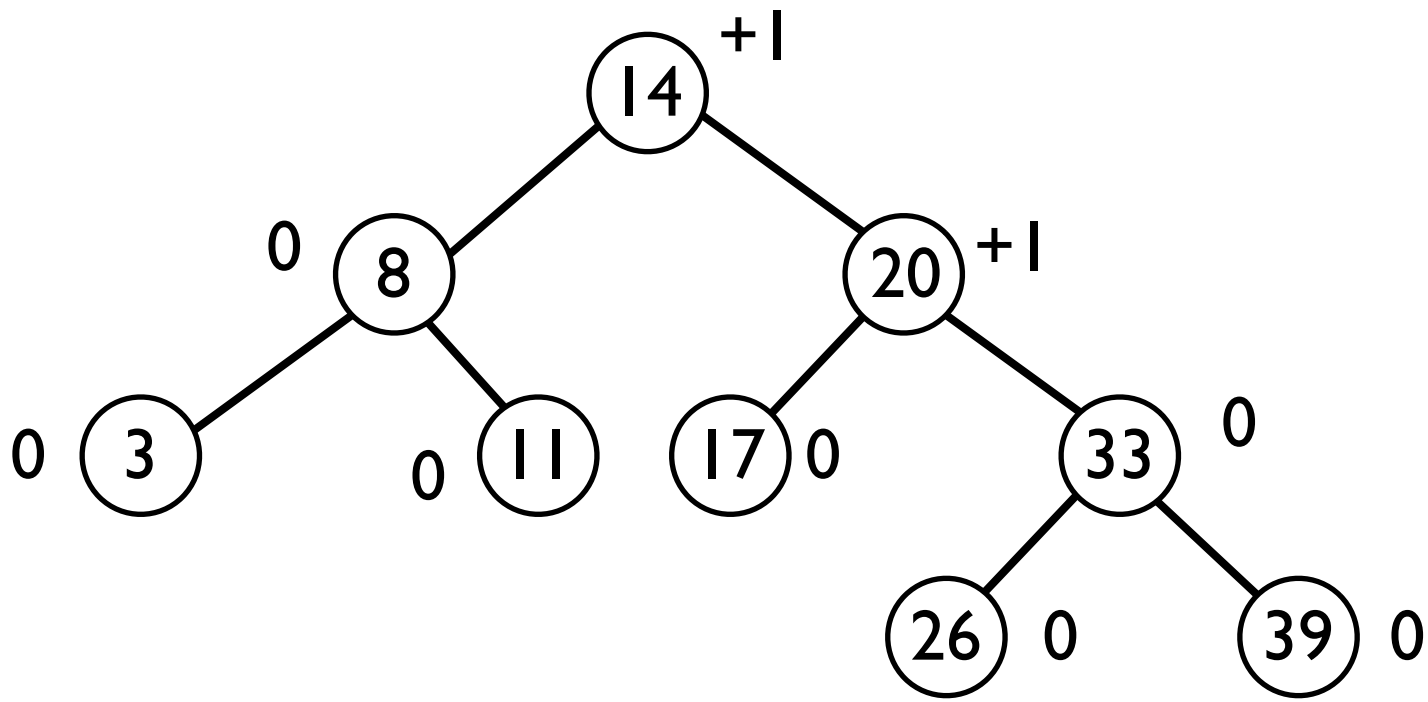
Suchen

AVL-Bäume - Beispiel: Einfügen



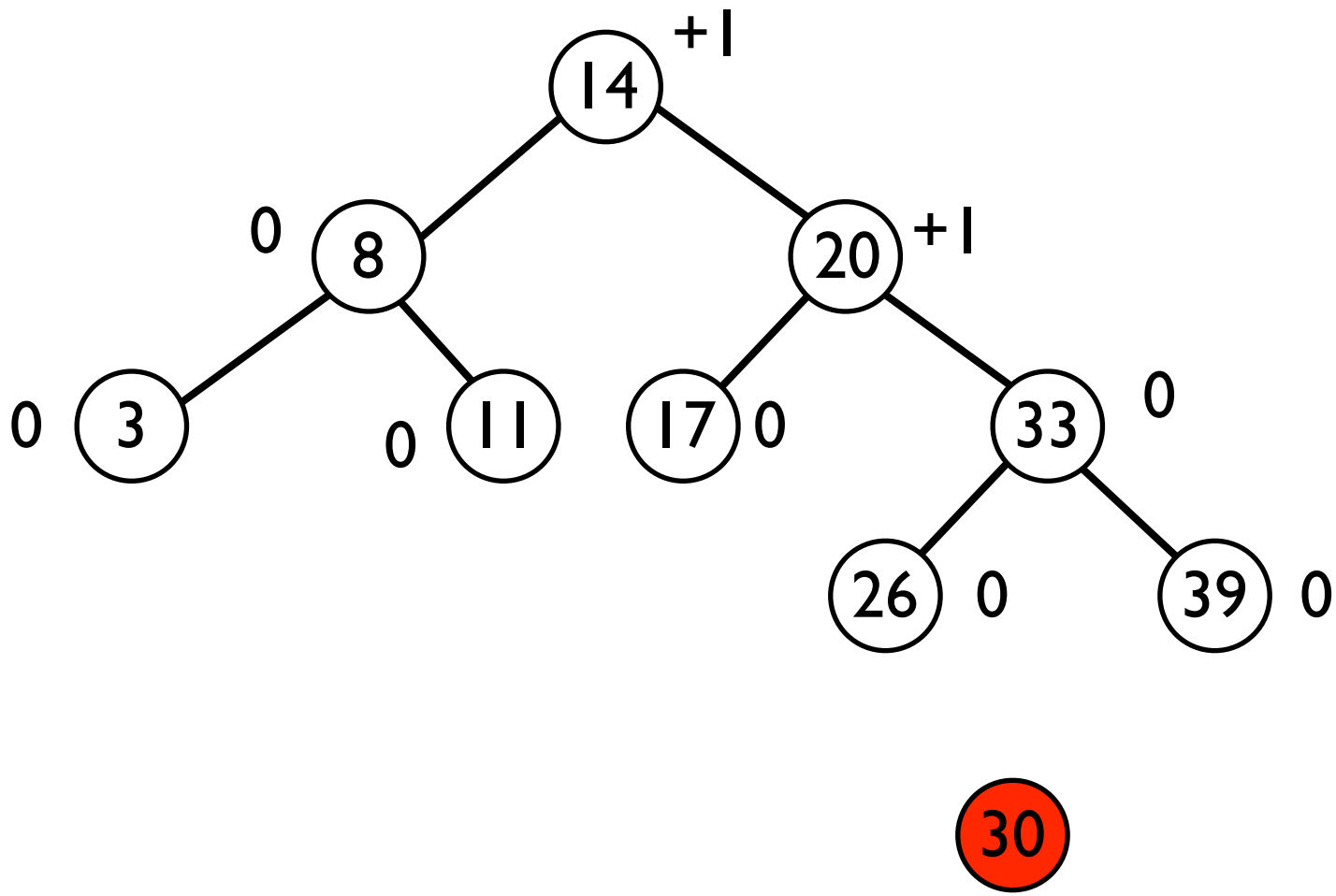
Suchen

AVL-Bäume - Beispiel: Einfügen



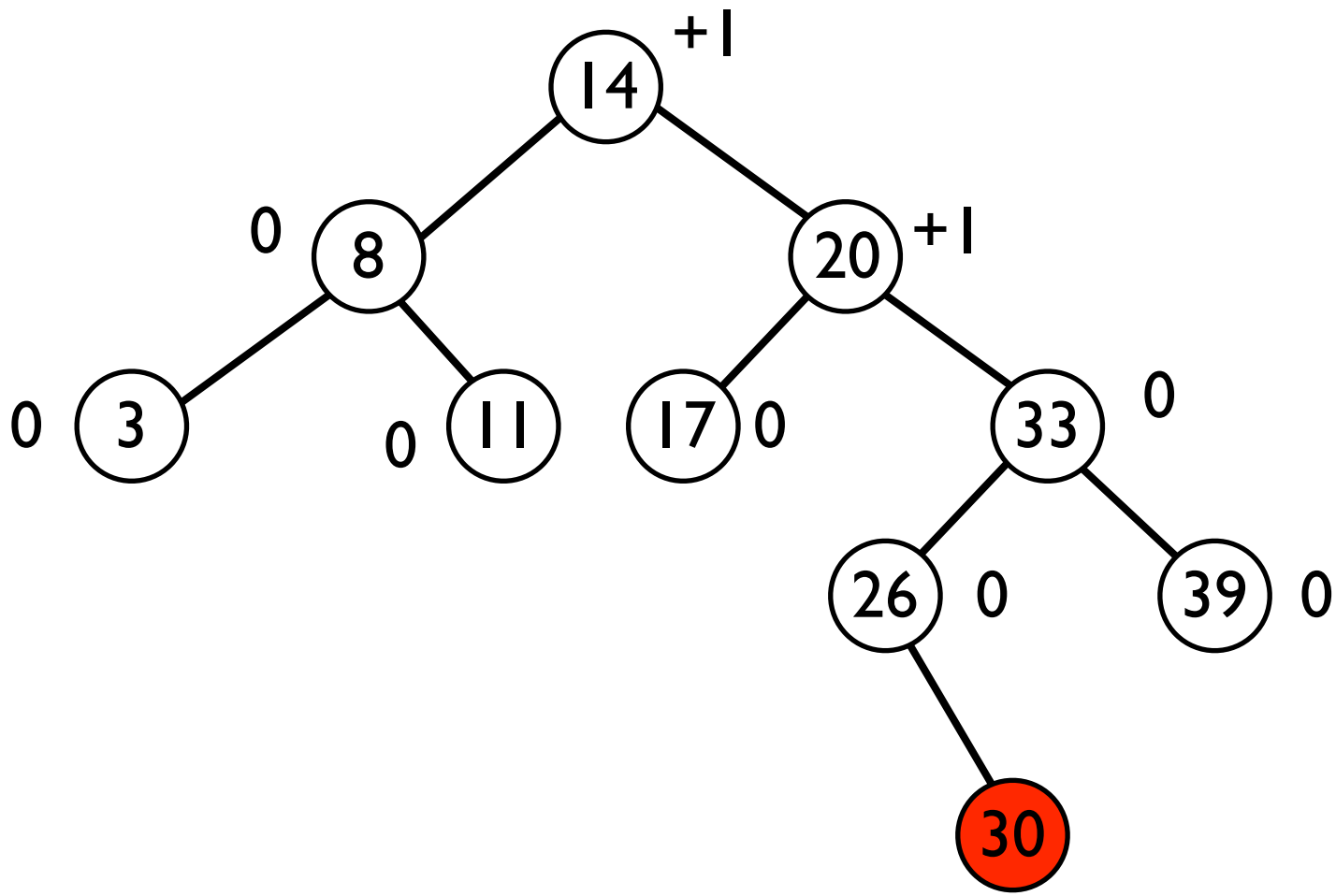
Suchen

AVL-Bäume - Beispiel: Einfügen



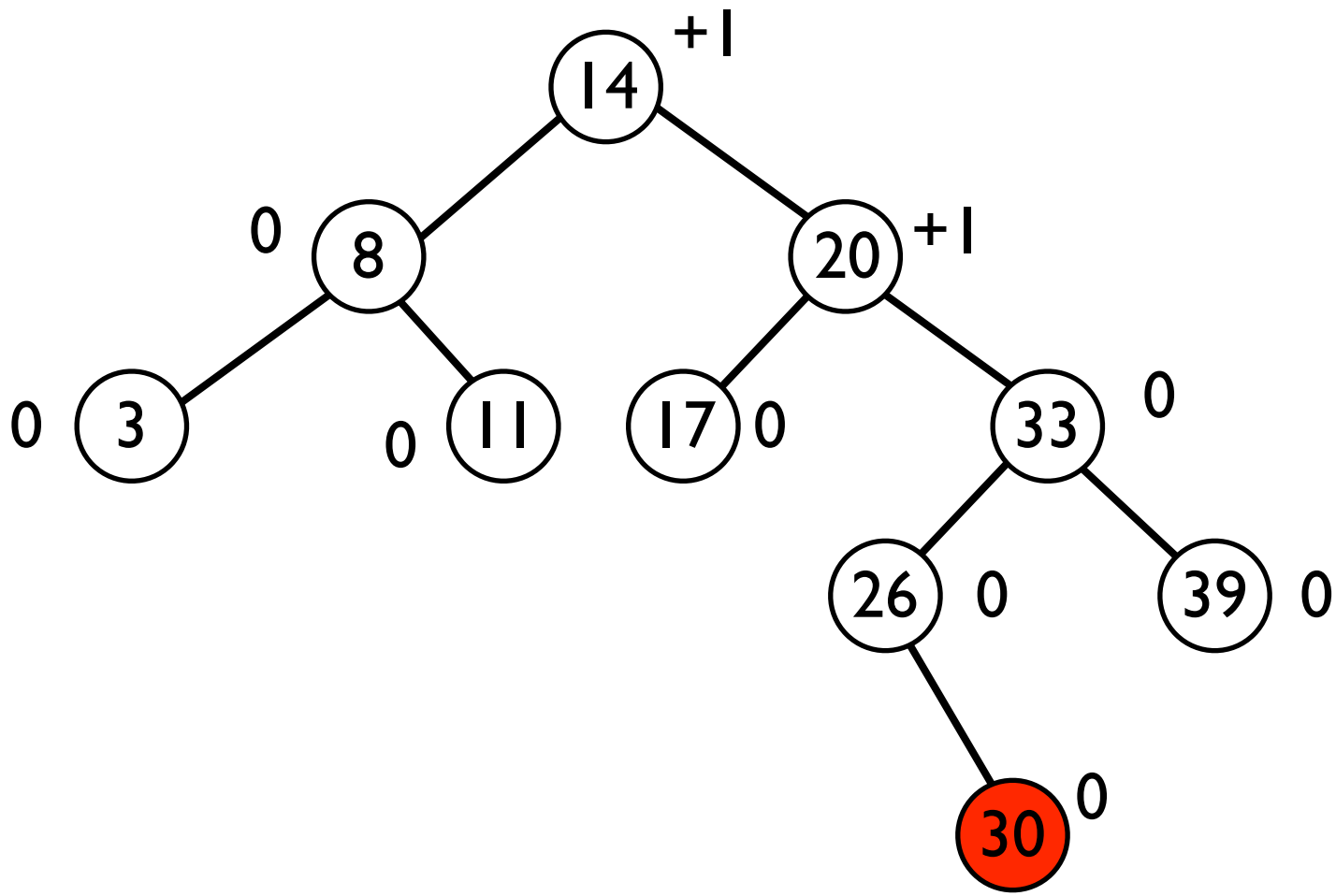
Suchen

AVL-Bäume - Beispiel: Einfügen



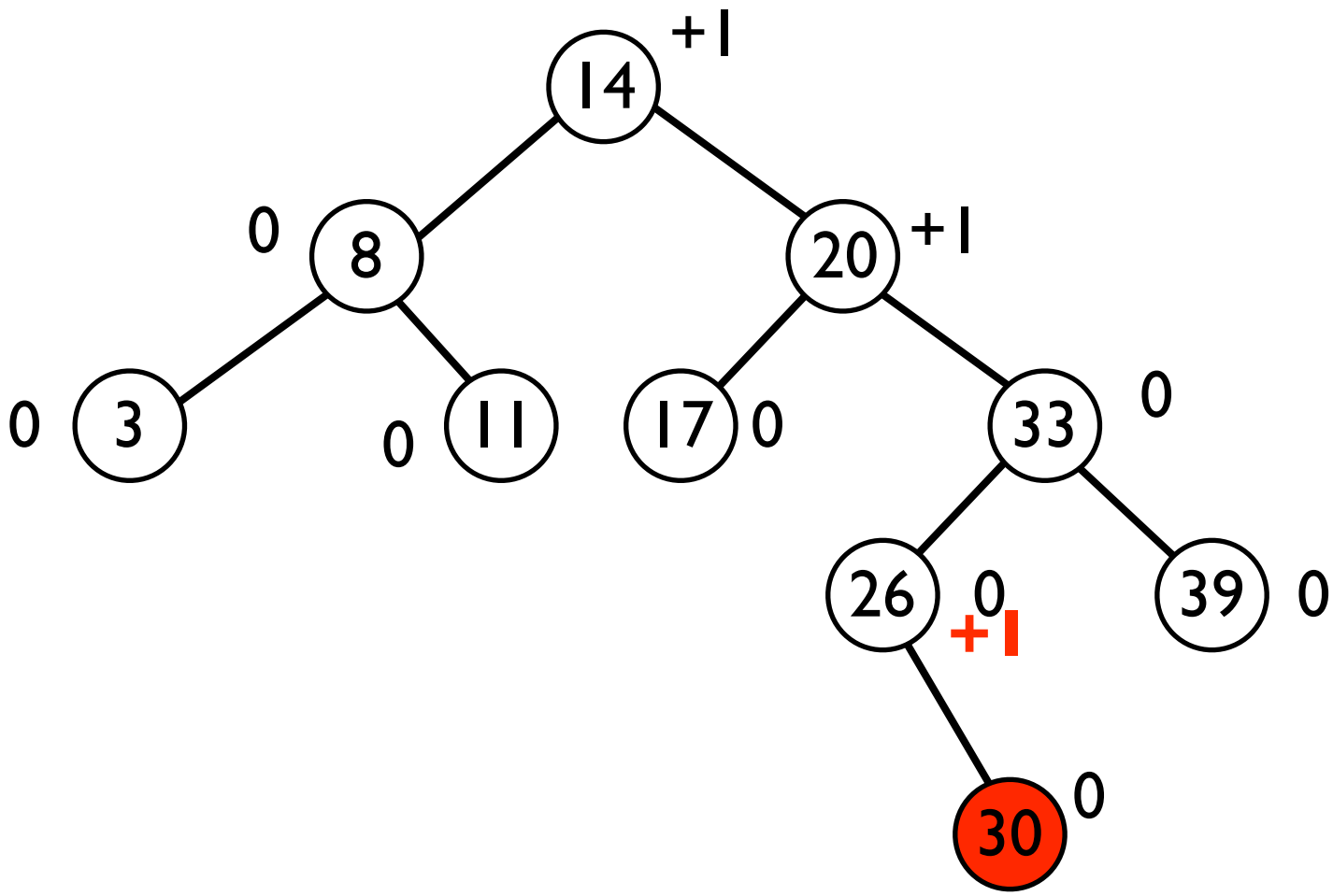
Suchen

AVL-Bäume - Beispiel: Einfügen



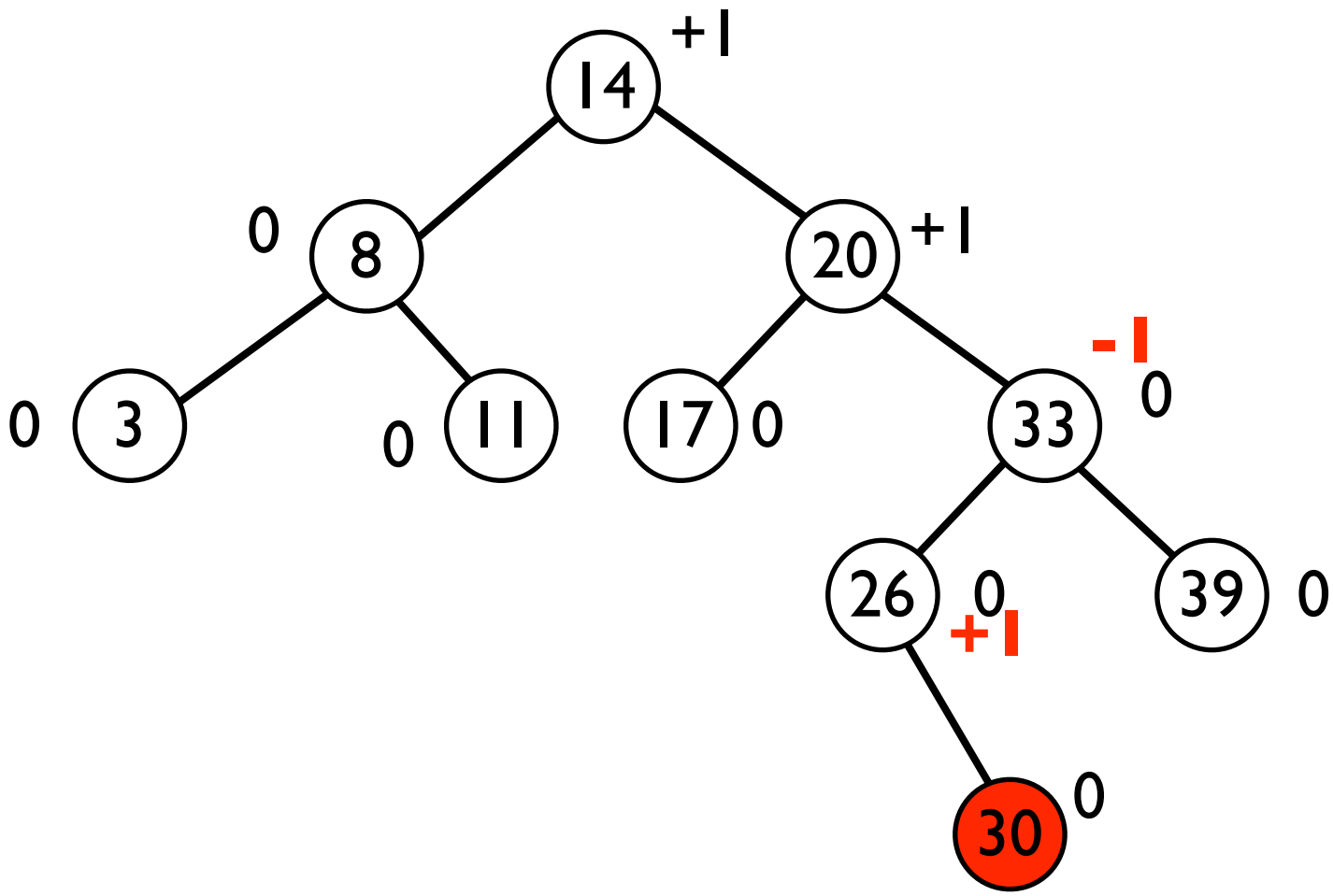
Suchen

AVL-Bäume - Beispiel: Einfügen



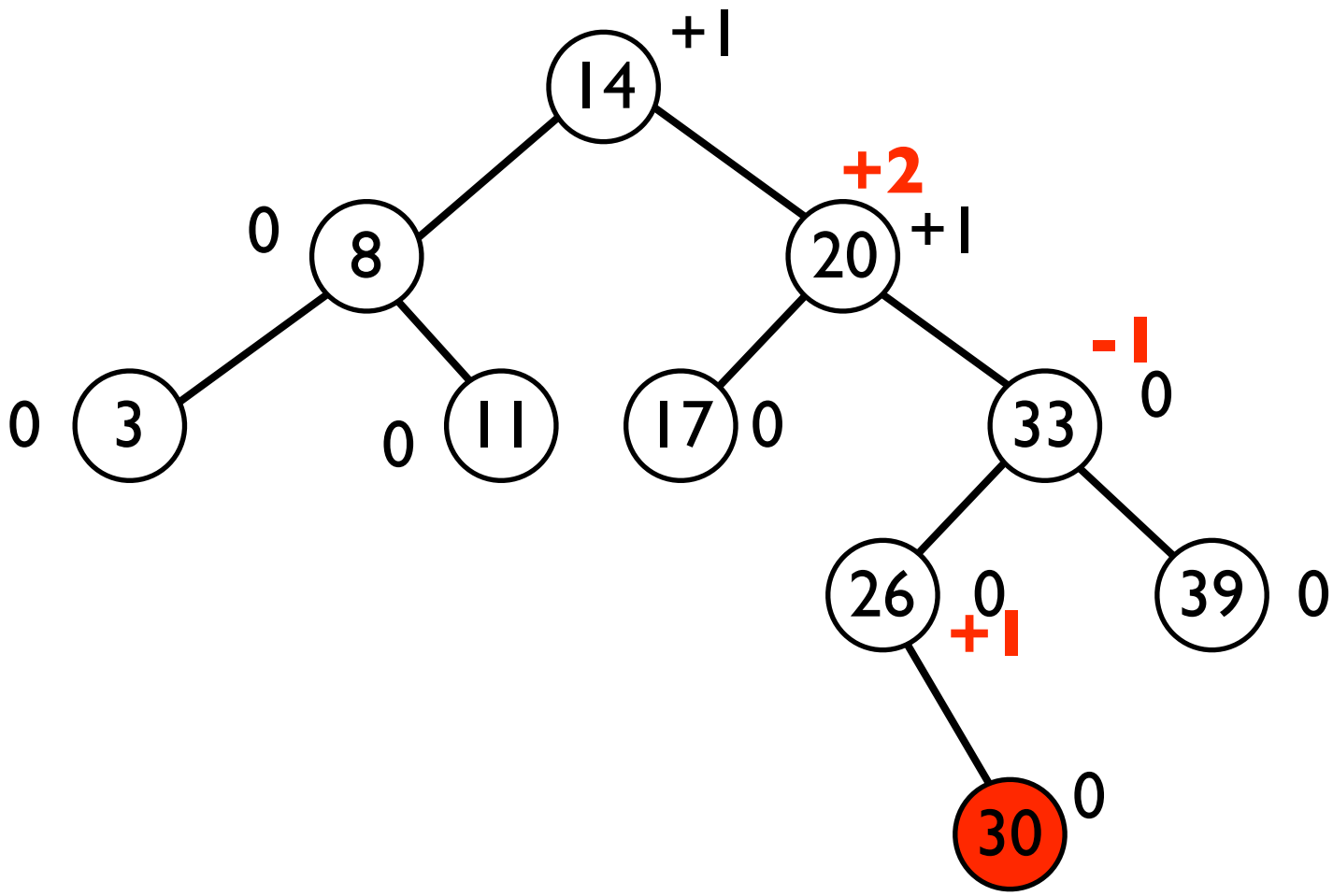
Suchen

AVL-Bäume - Beispiel: Einfügen



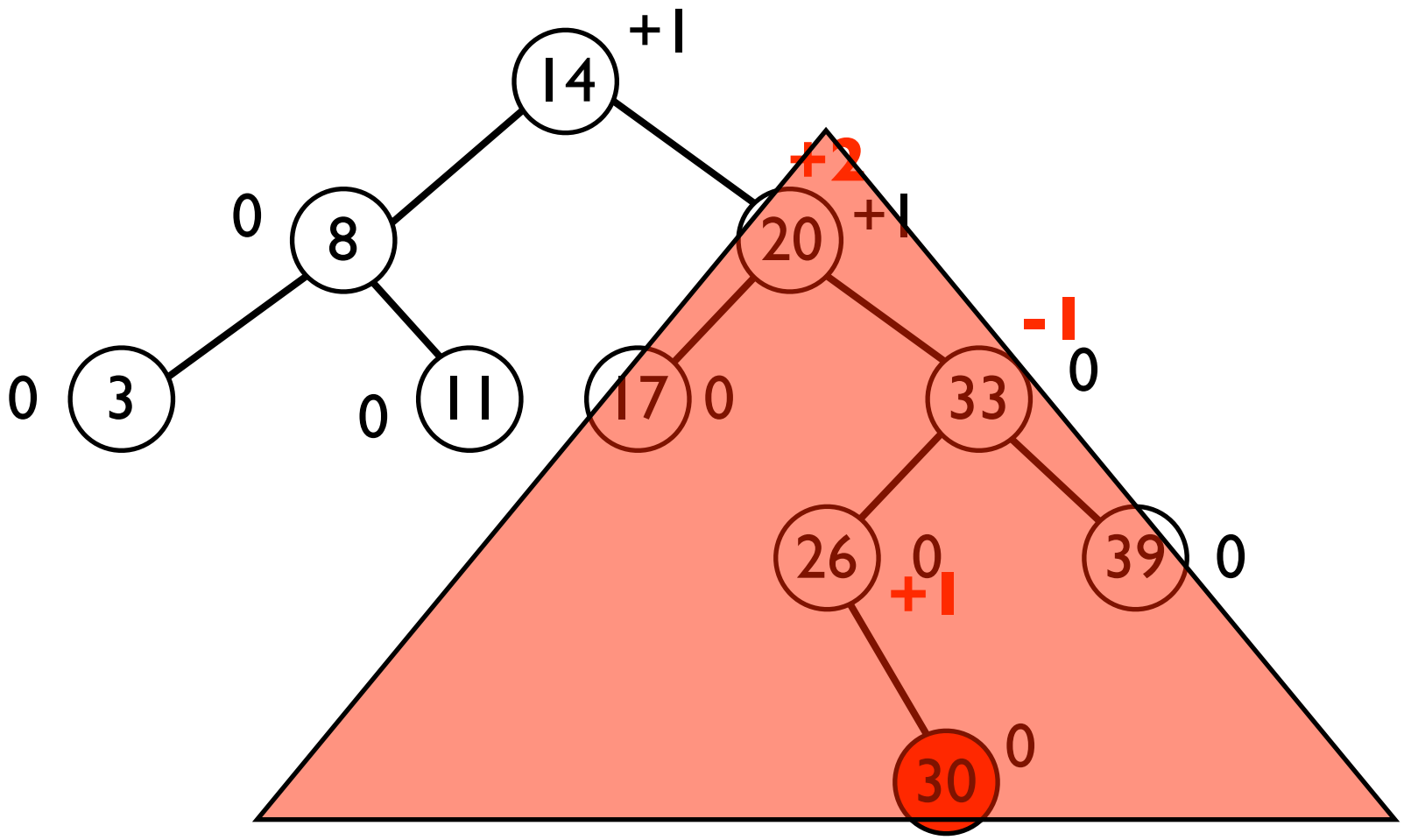
Suchen

AVL-Bäume - Beispiel: Einfügen



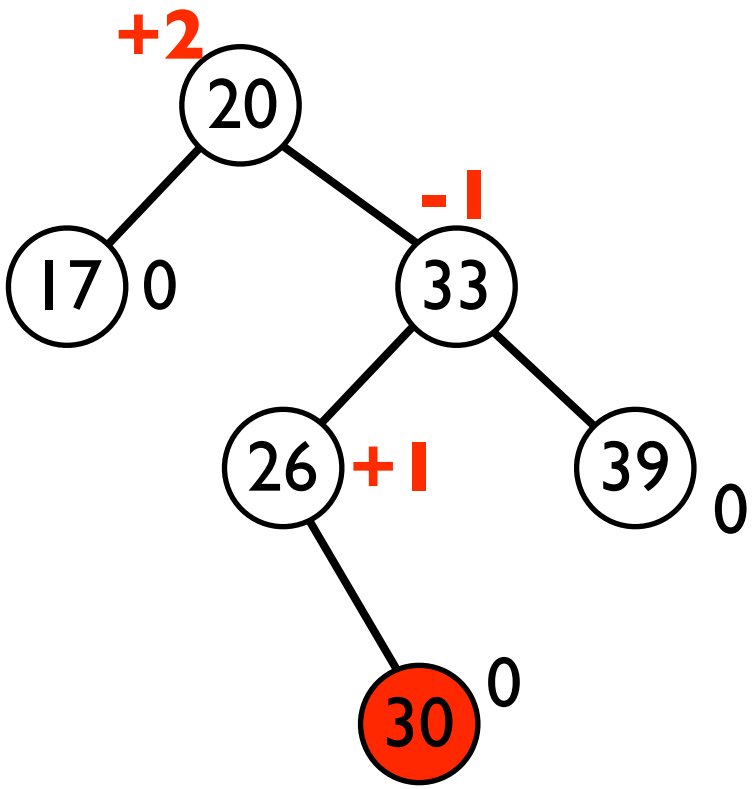
Suchen

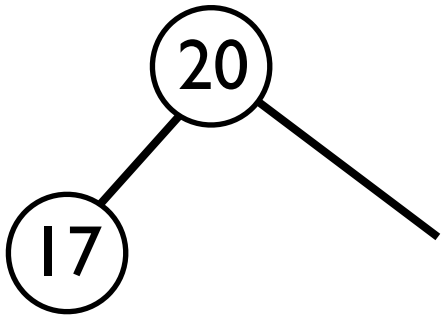
AVL-Bäume - Beispiel: Einfügen



Suchen

Ausbalancieren - Beispiel: Einfügen

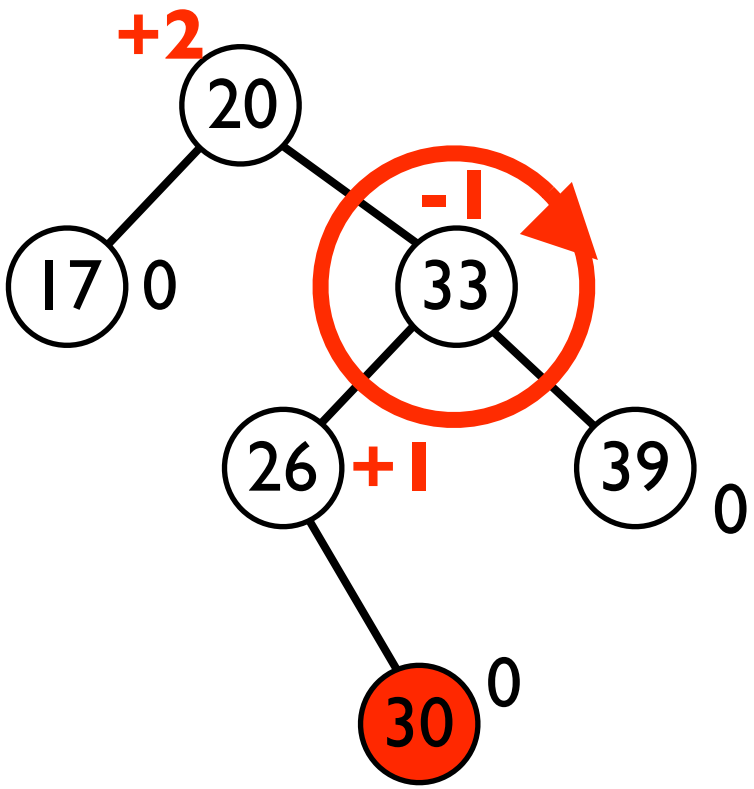


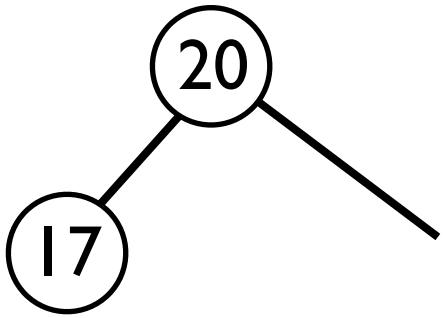


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

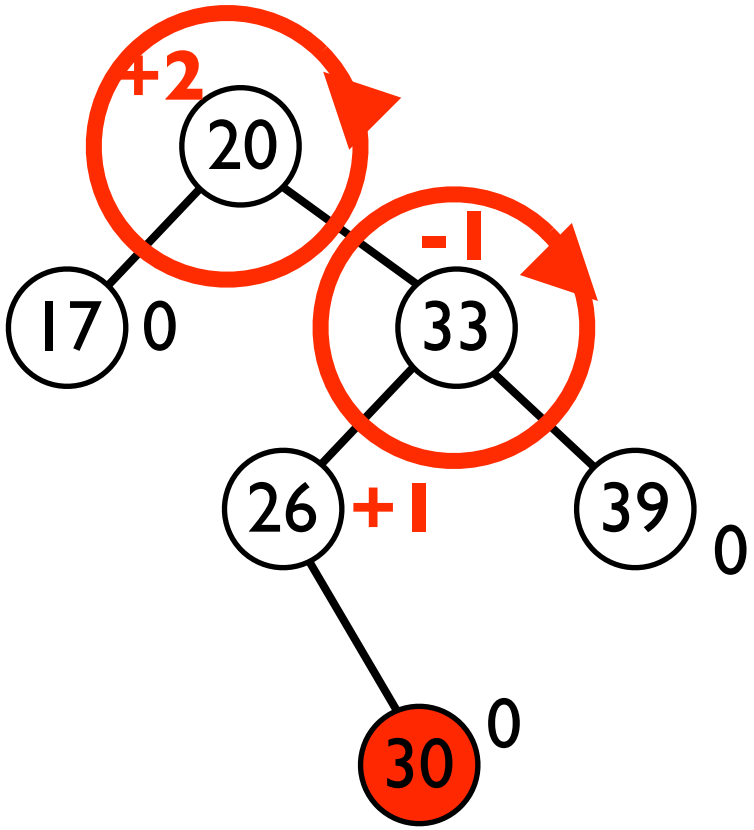


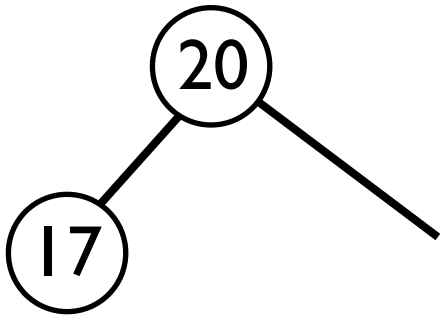


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

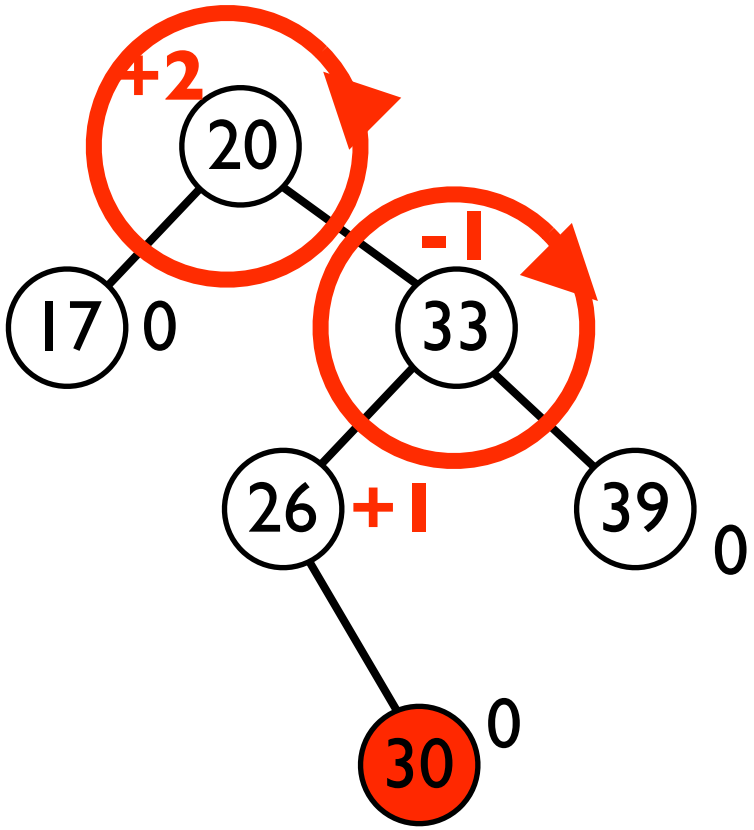


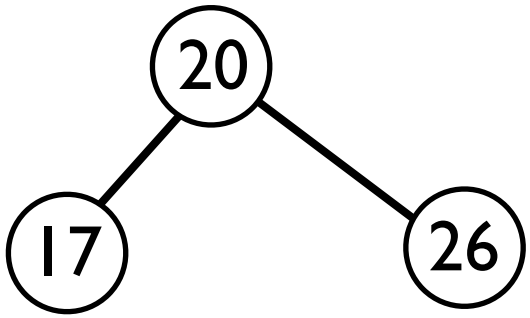


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

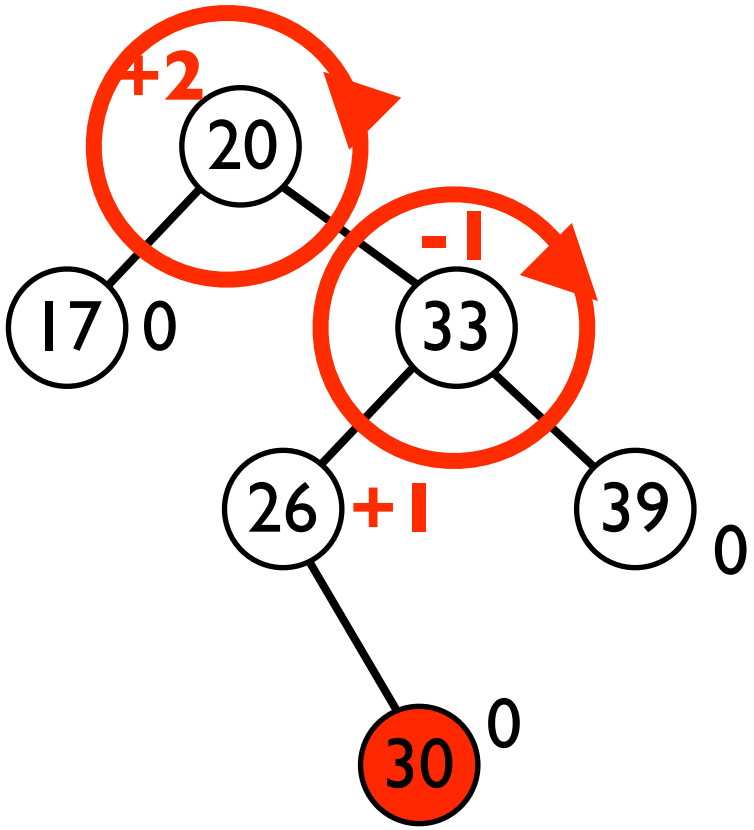


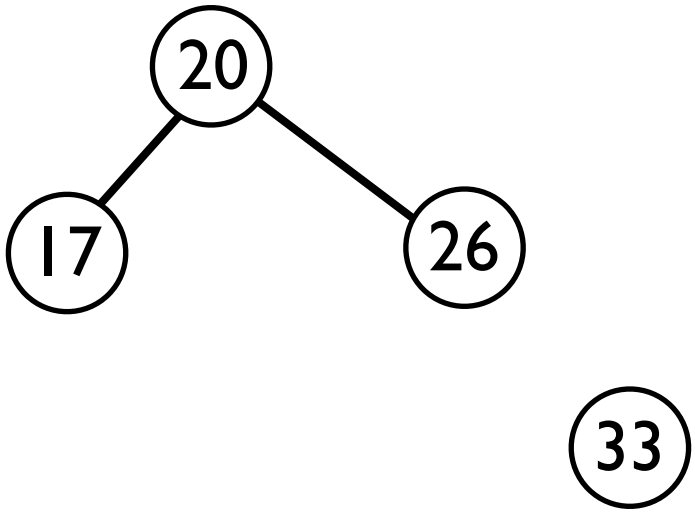


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

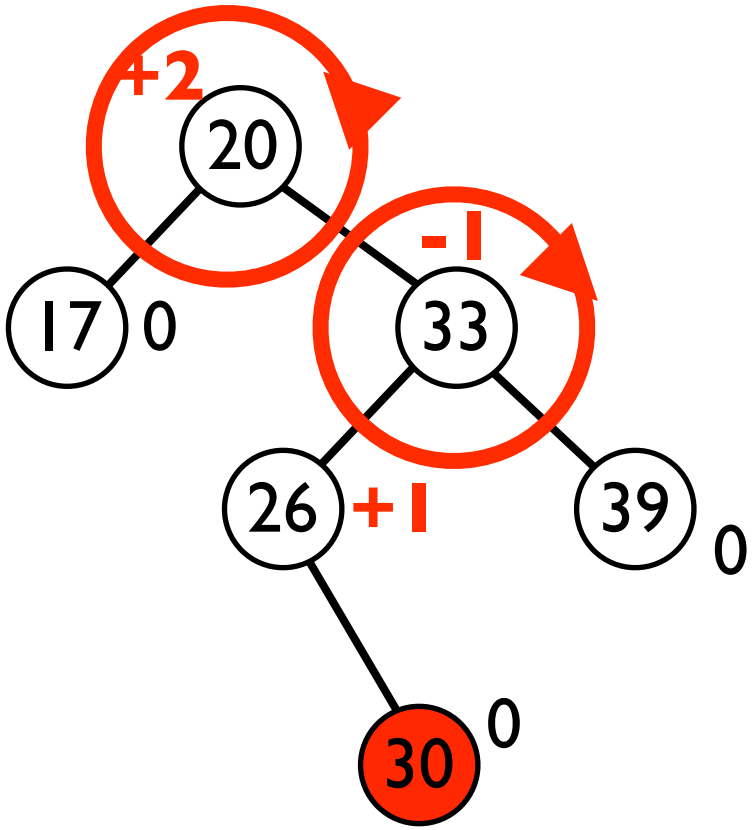


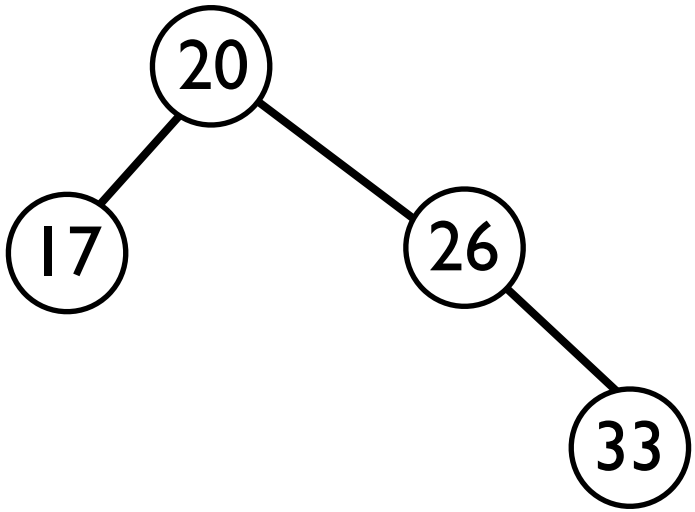


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

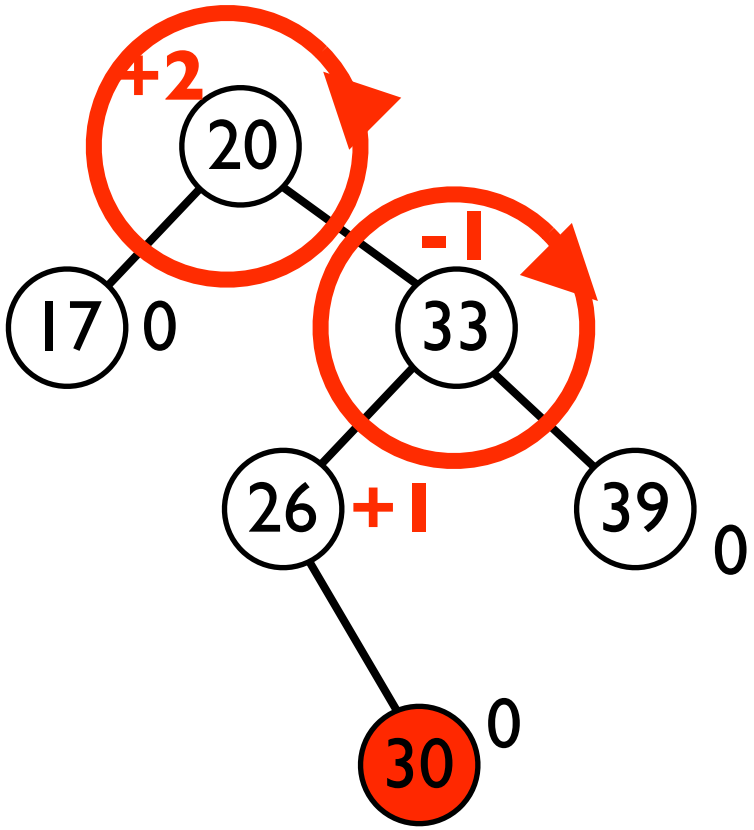


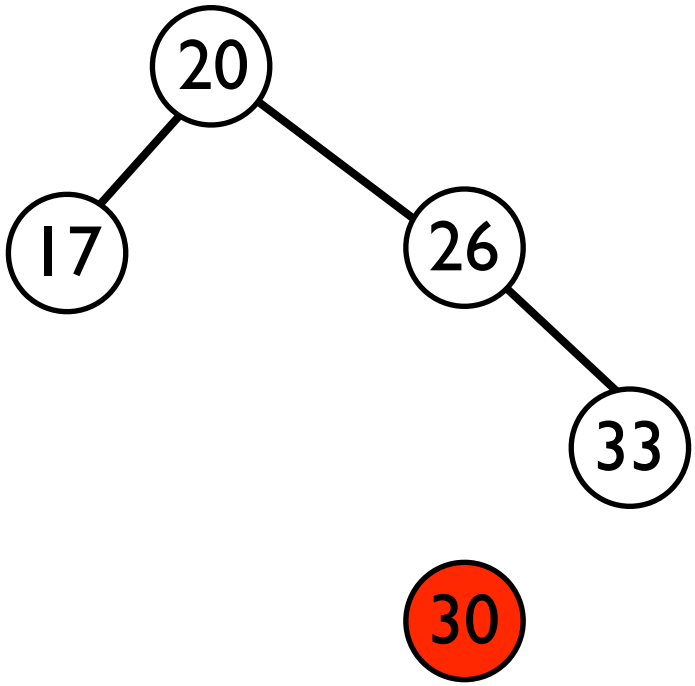


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

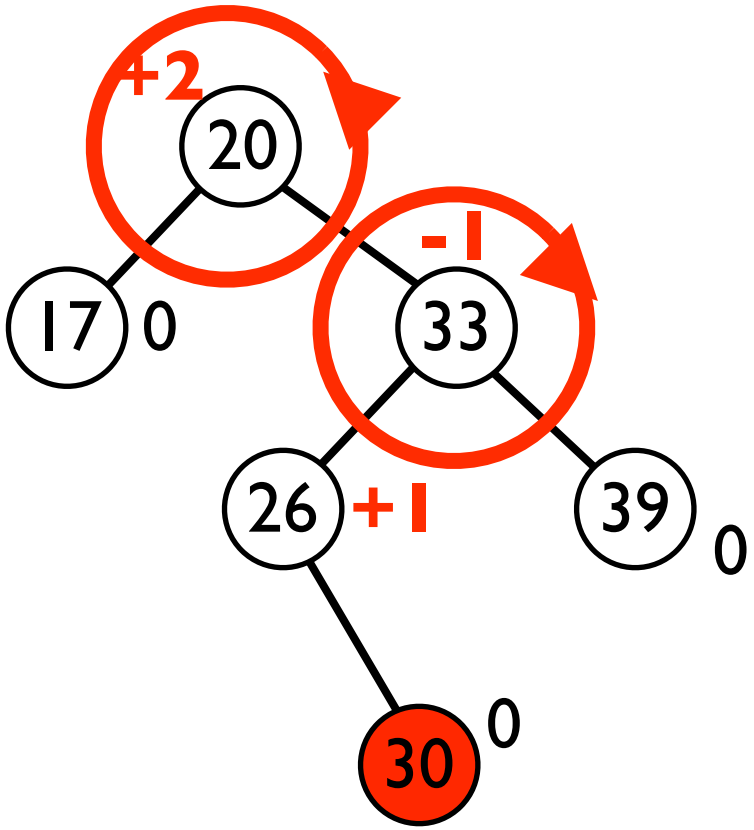


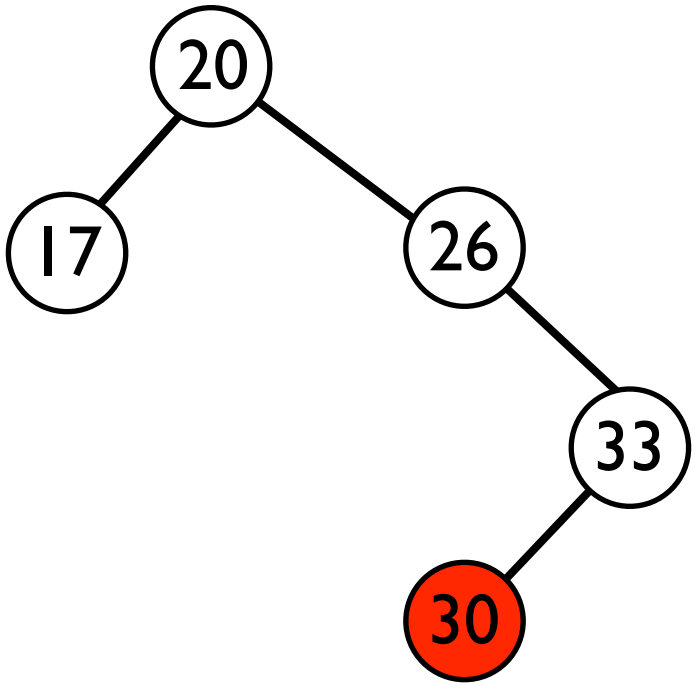


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

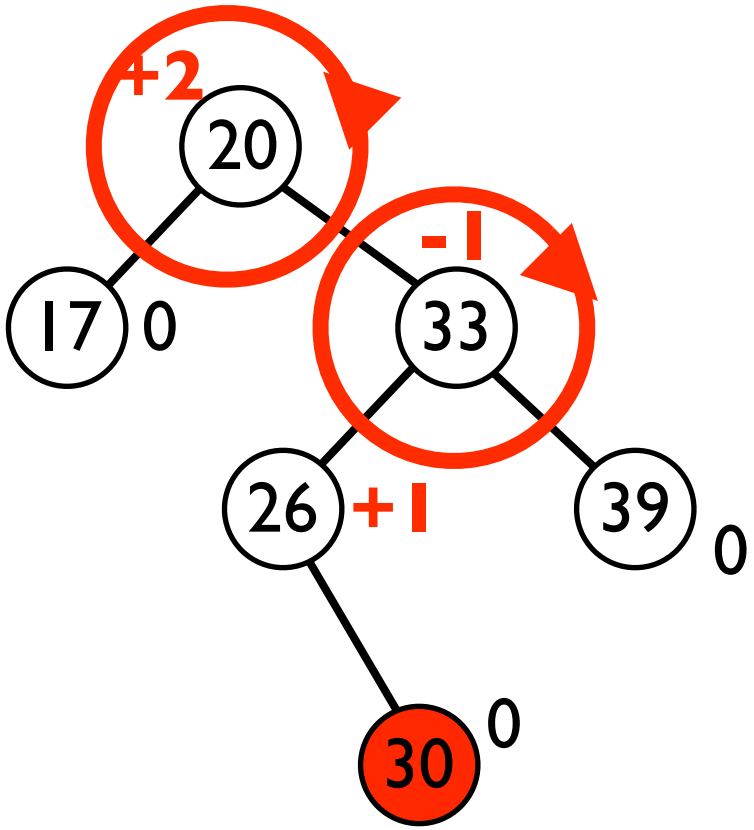


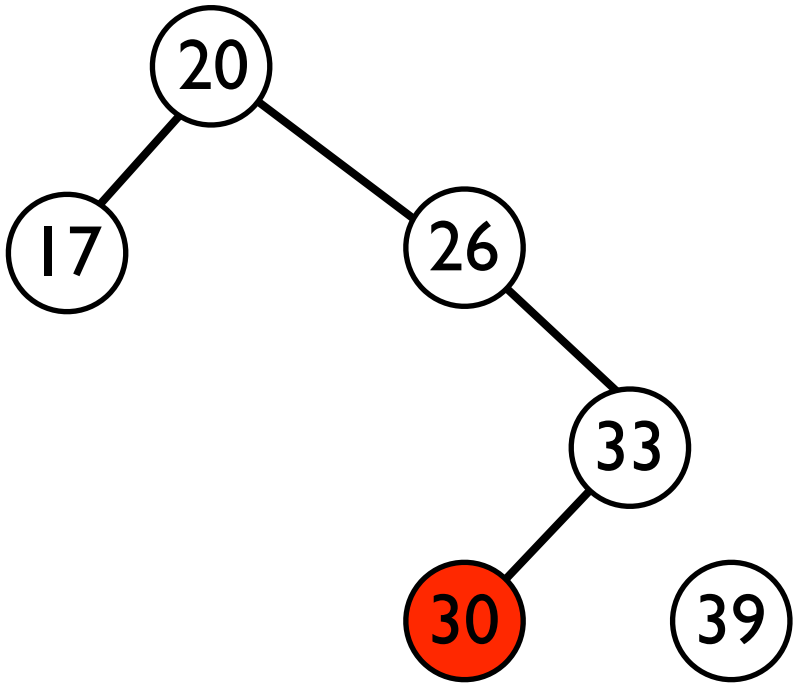


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

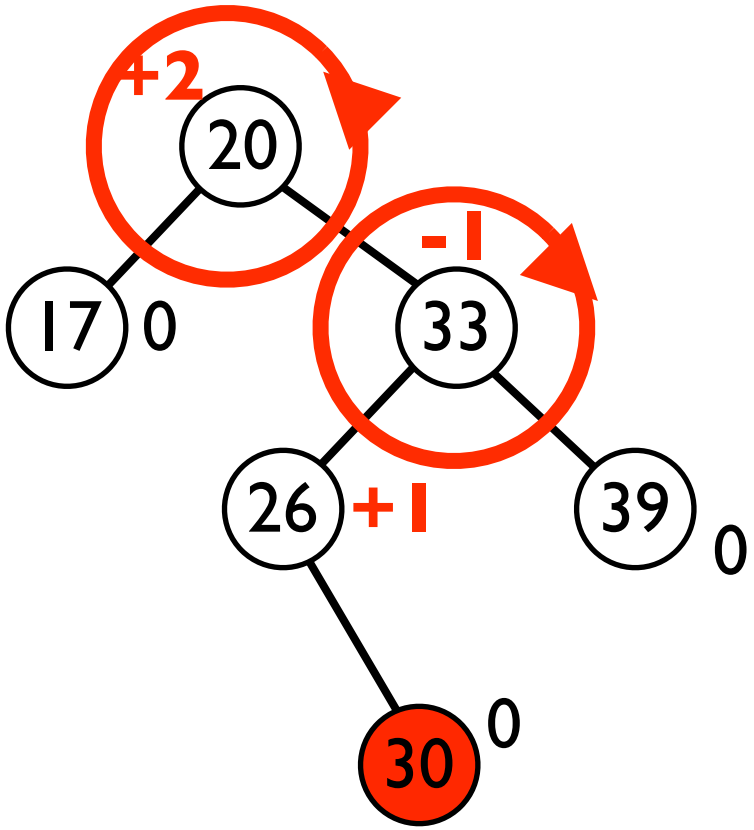


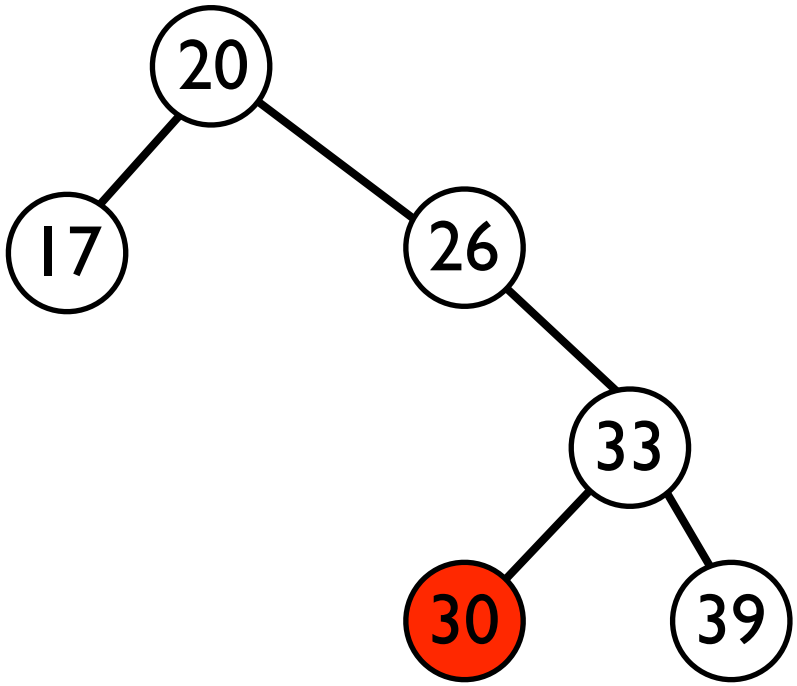


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

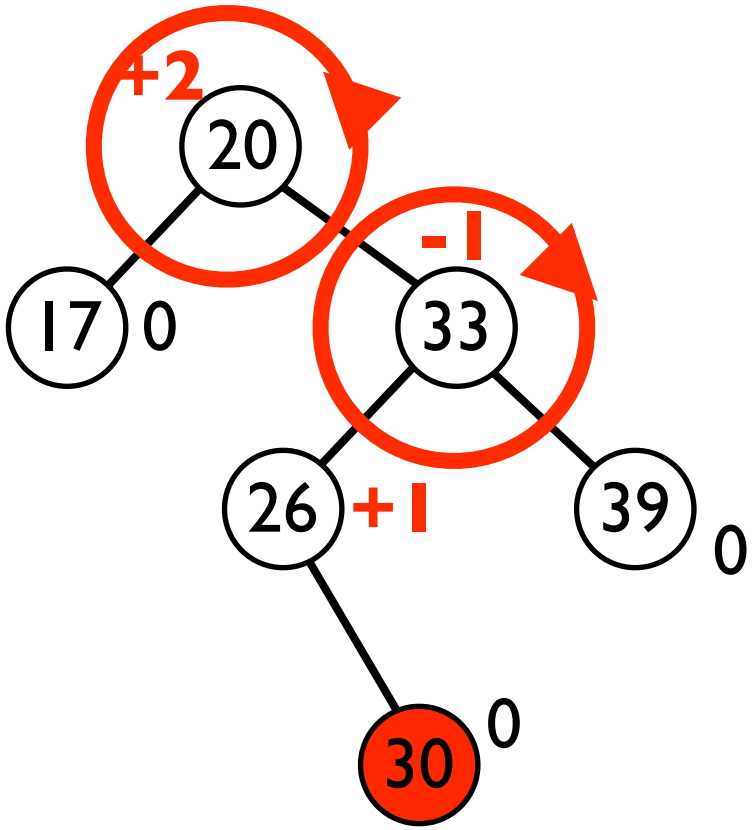


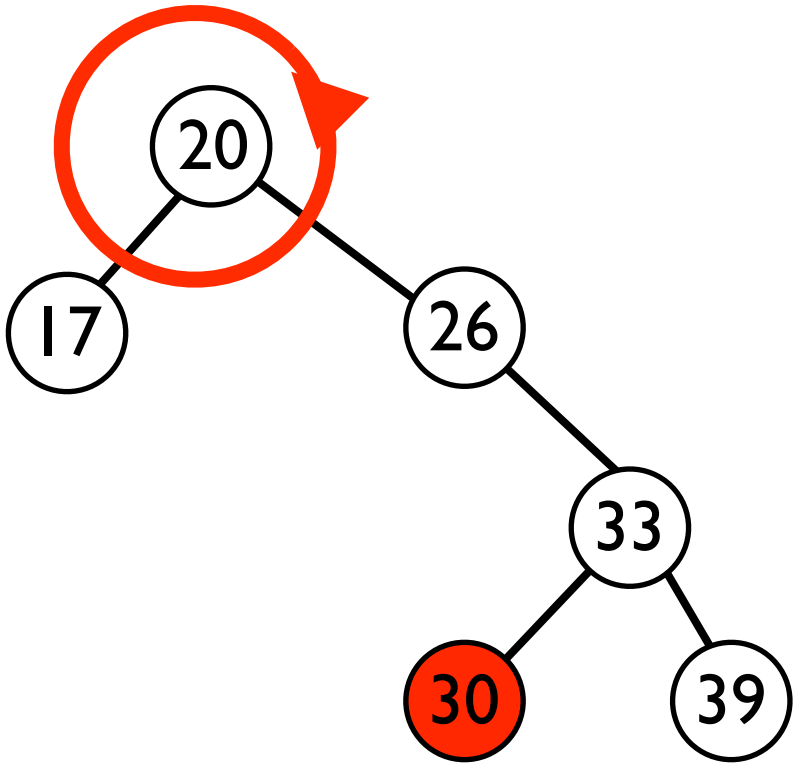


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

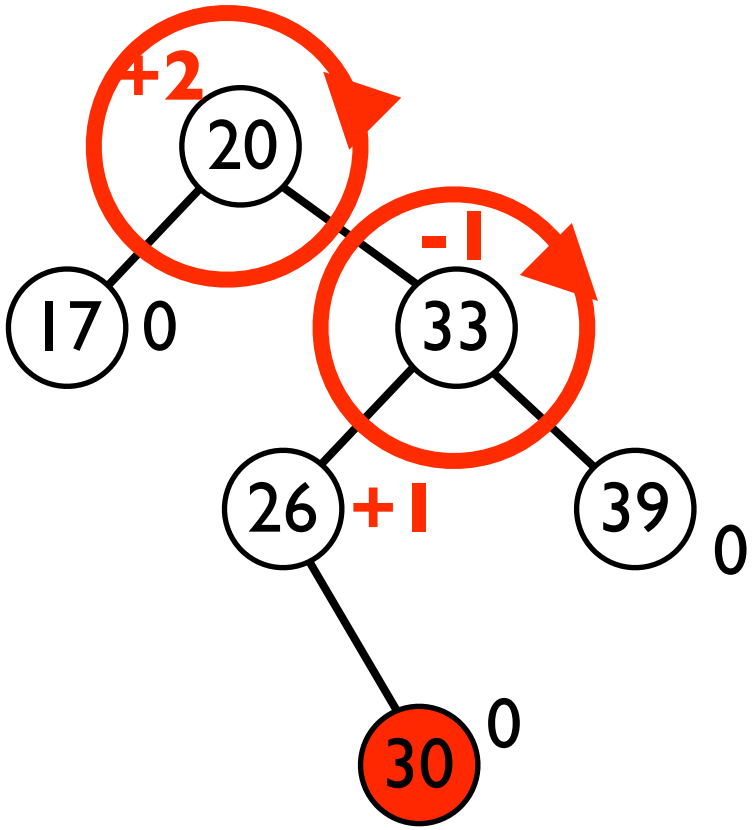


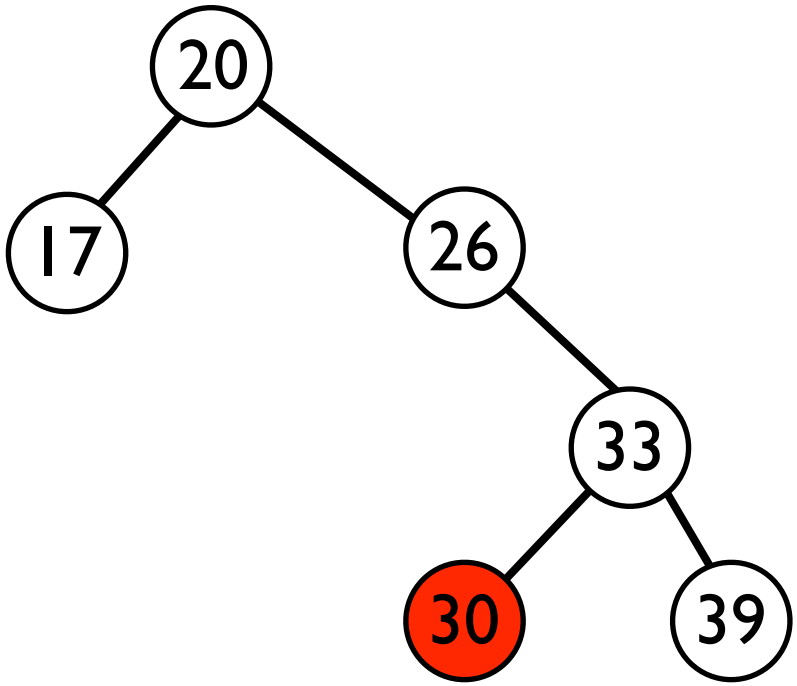


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

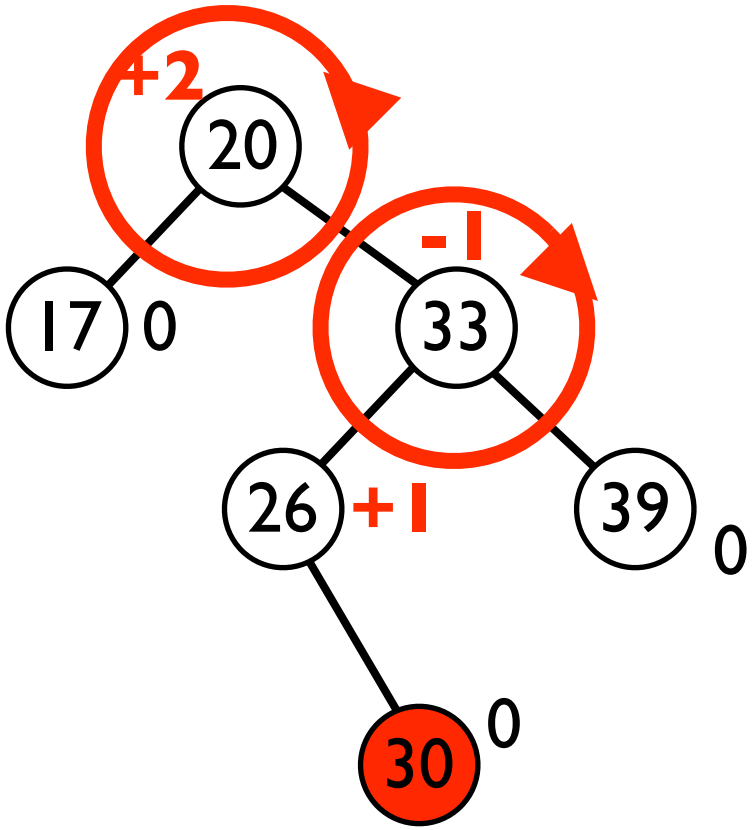


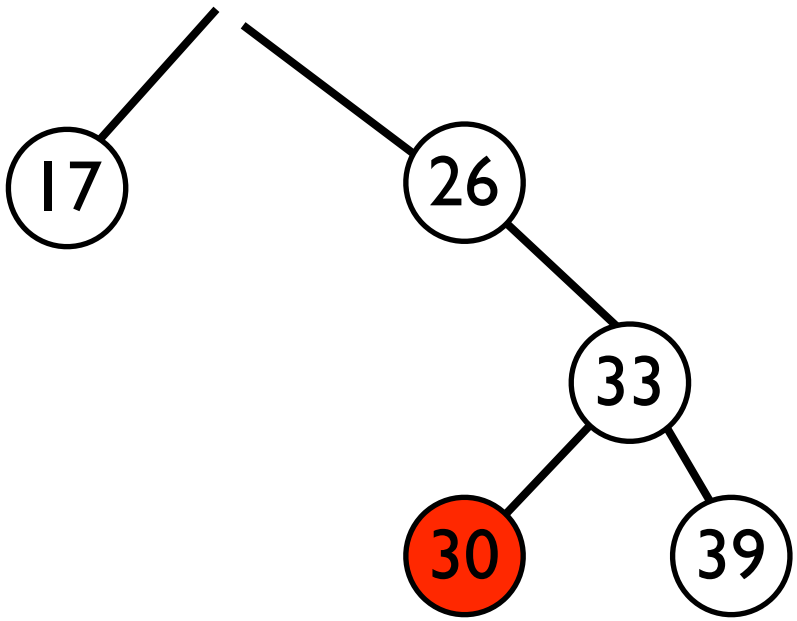


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

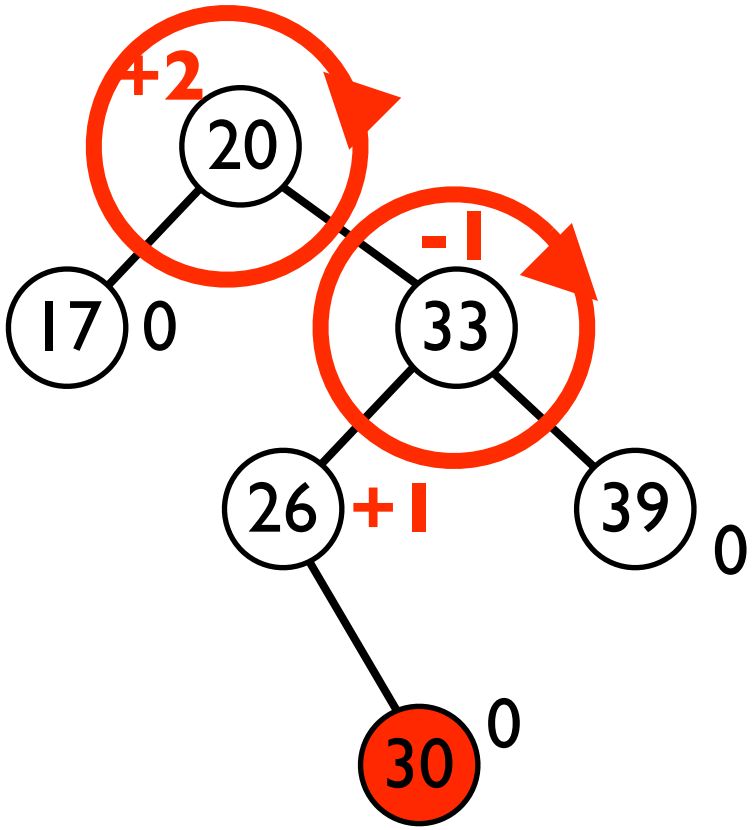


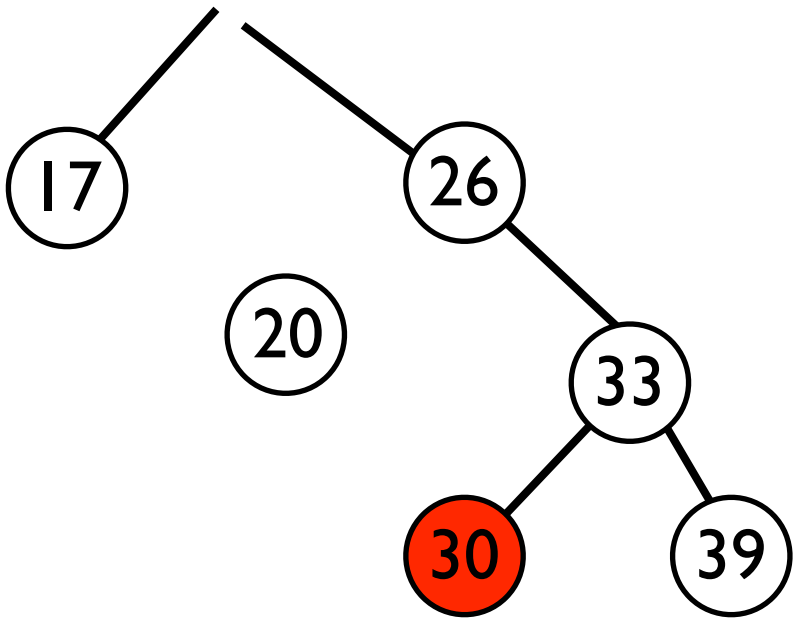


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

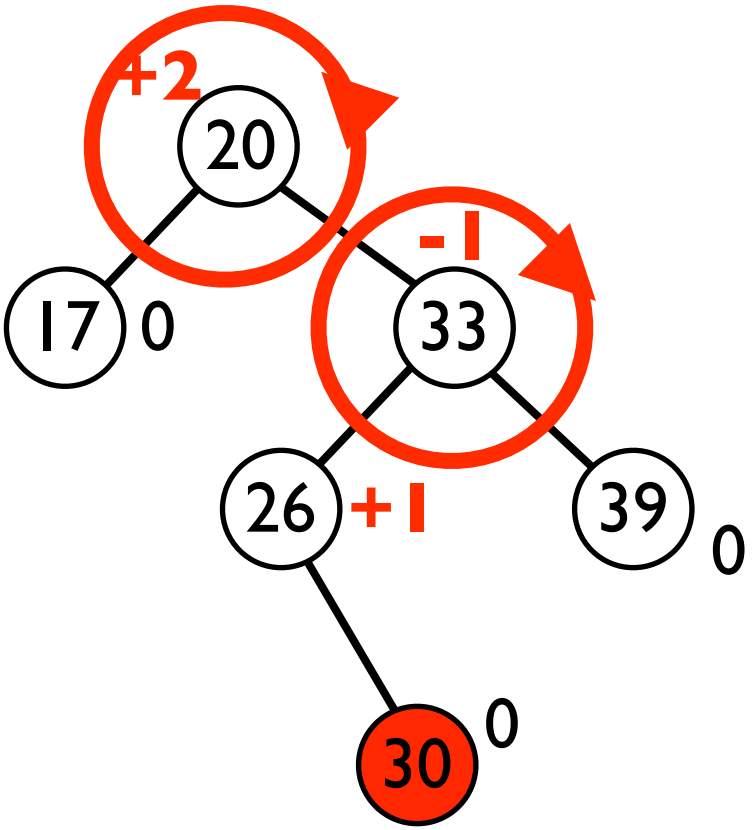


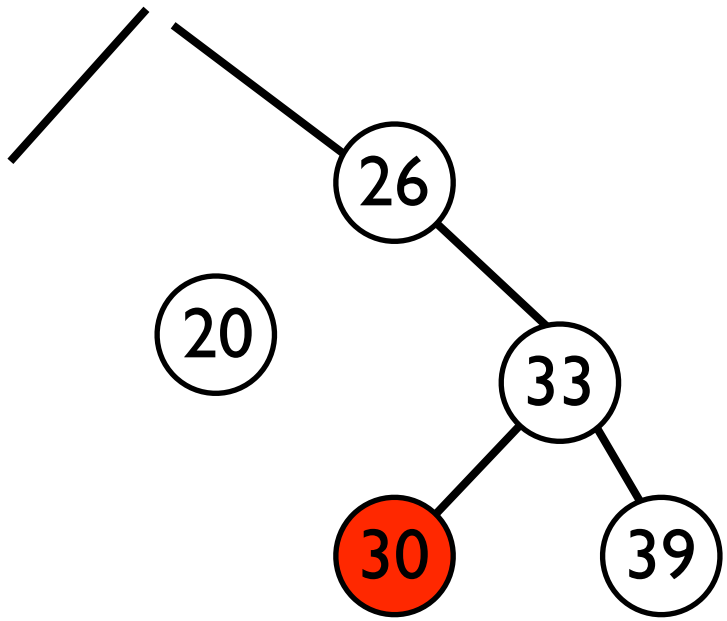


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

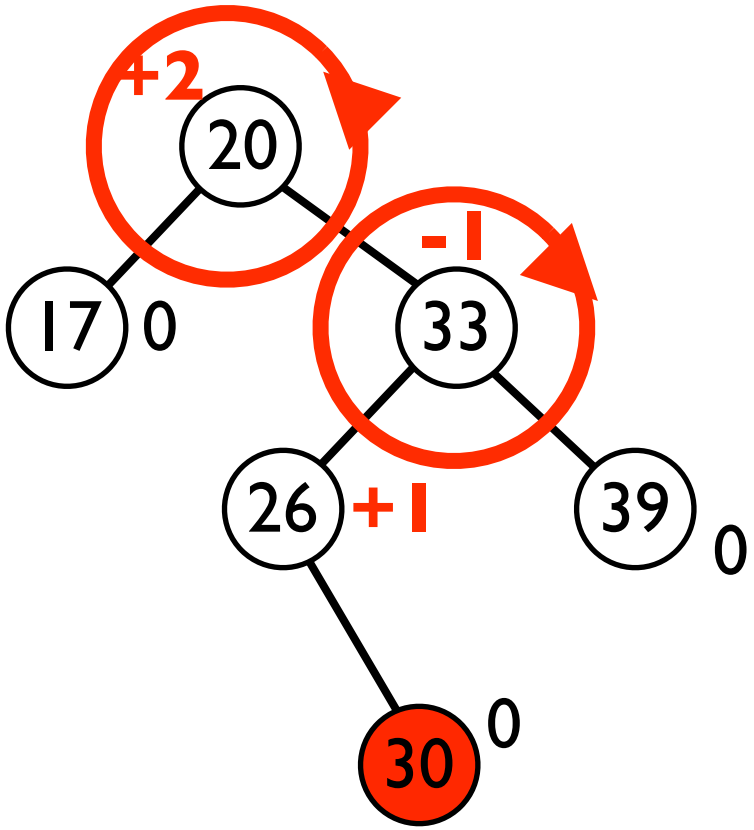


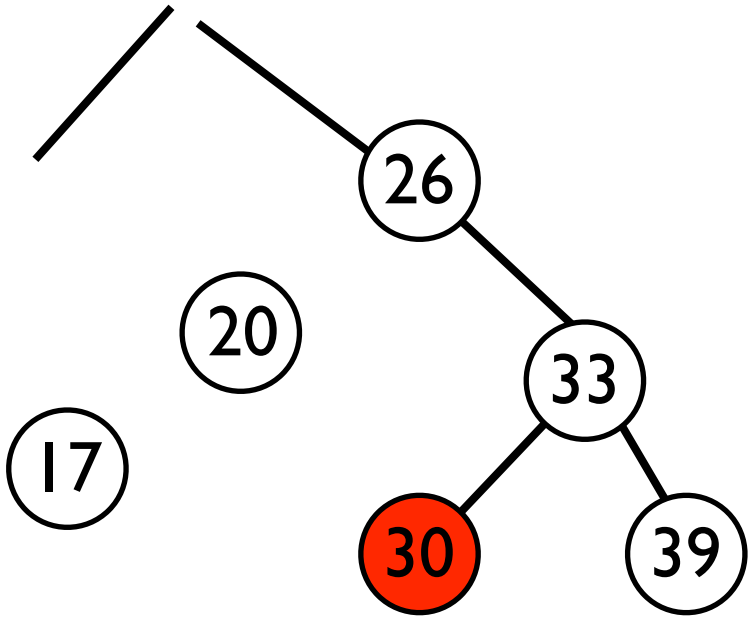


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

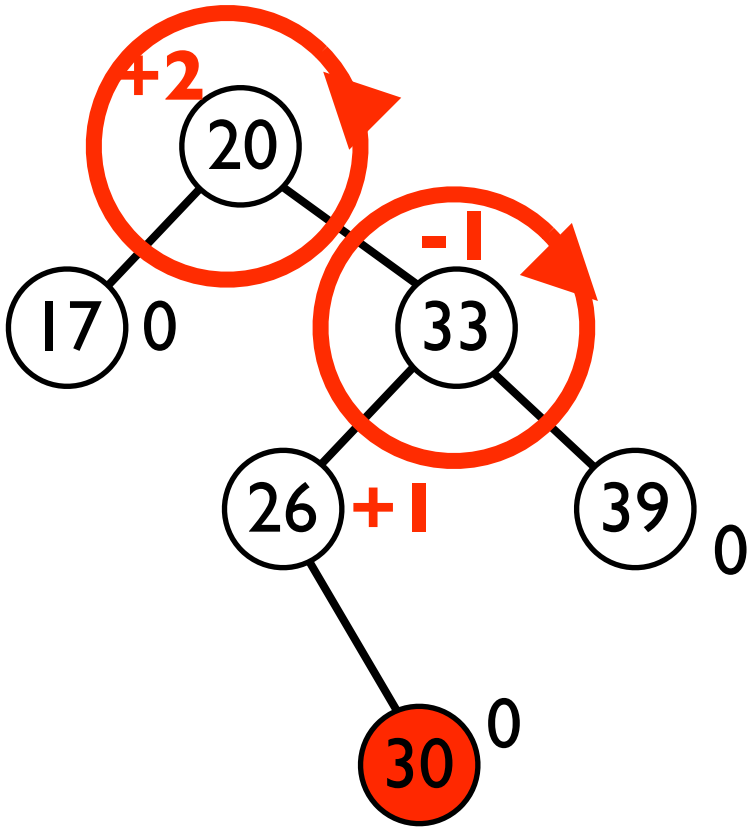


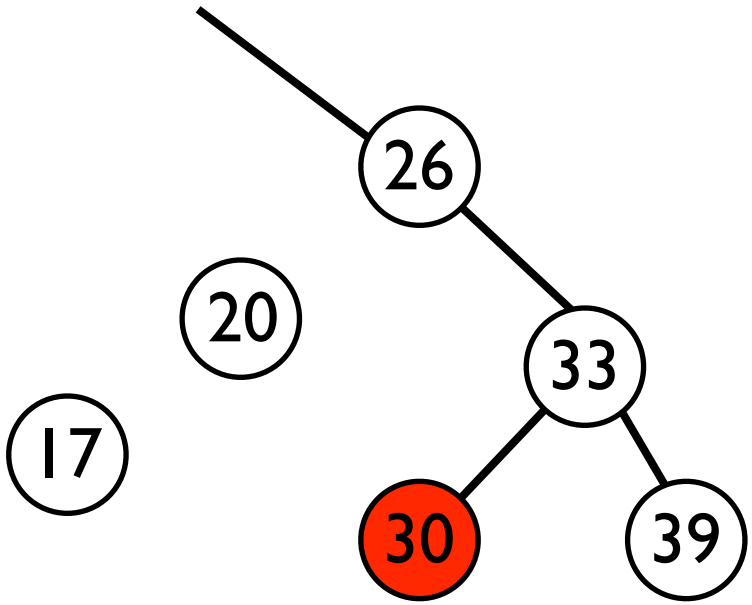


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

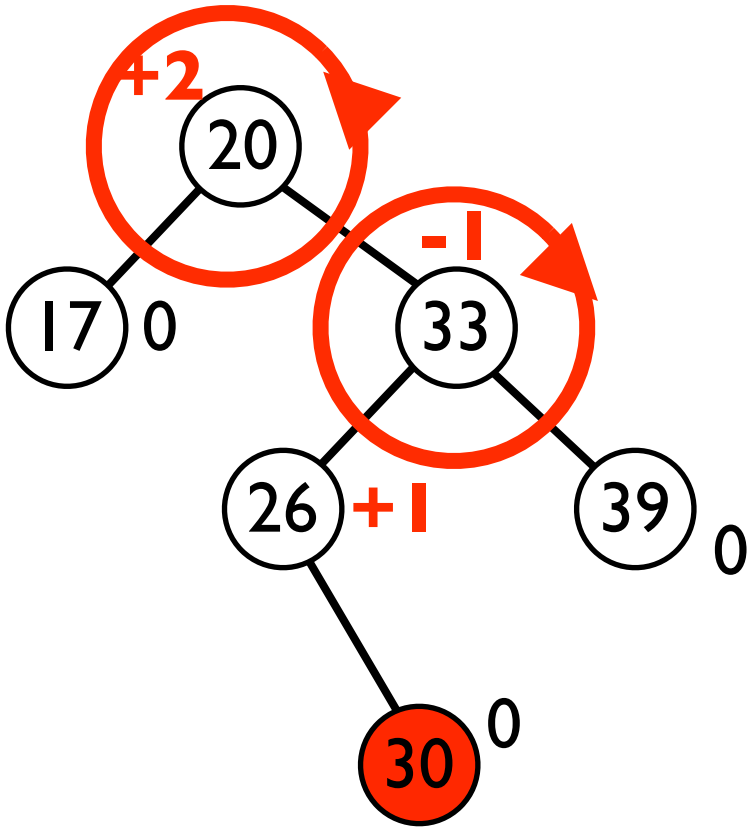


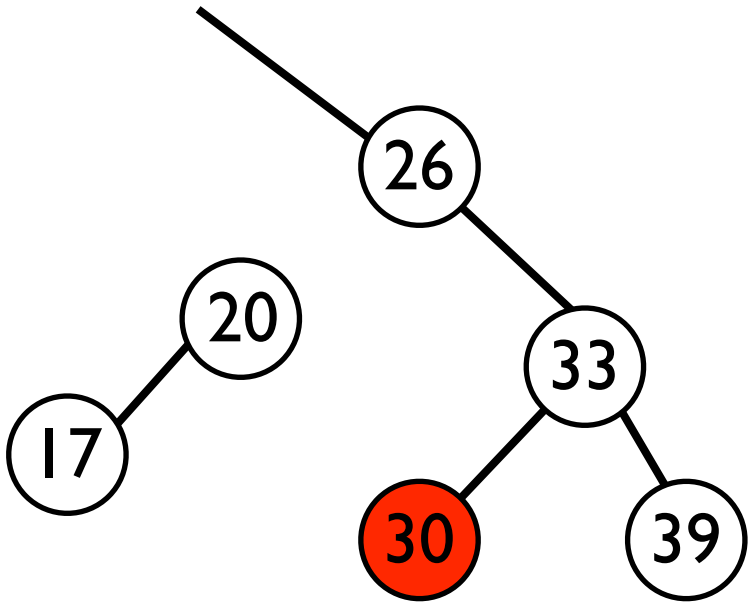


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

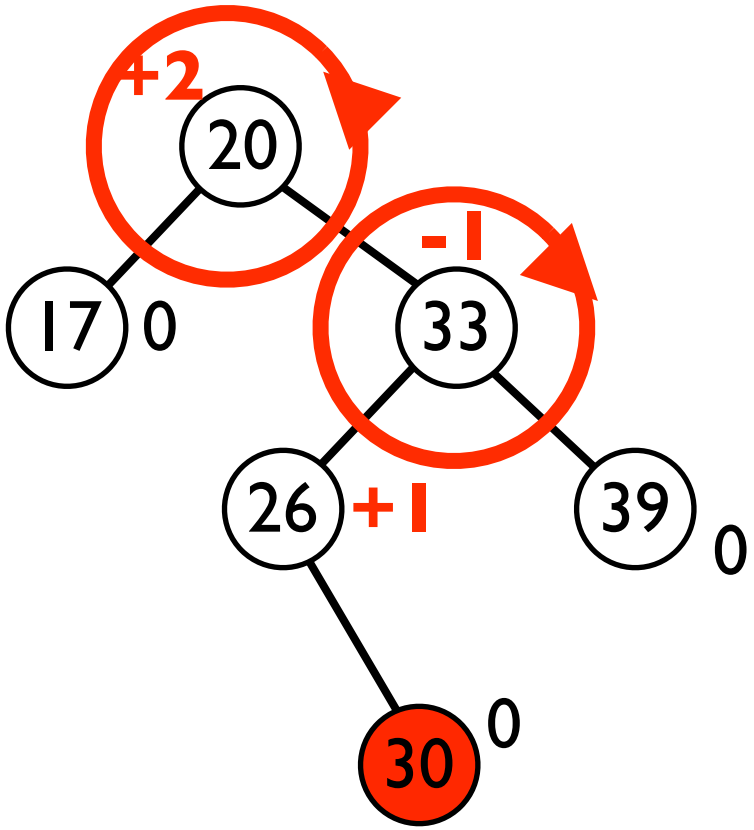


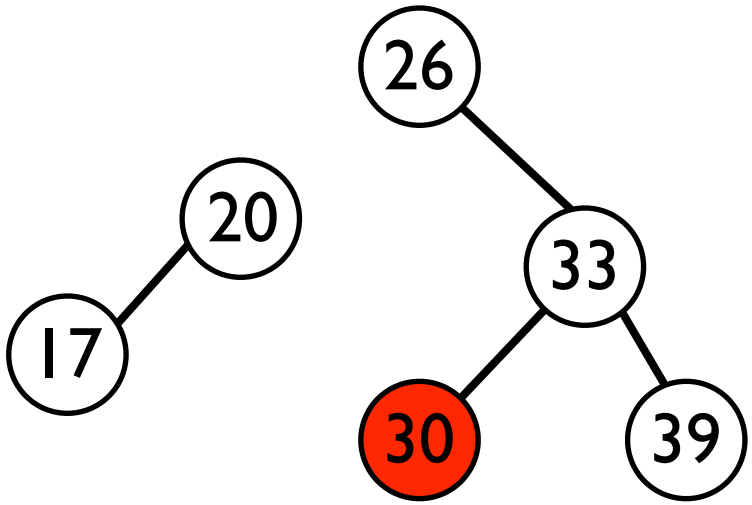


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

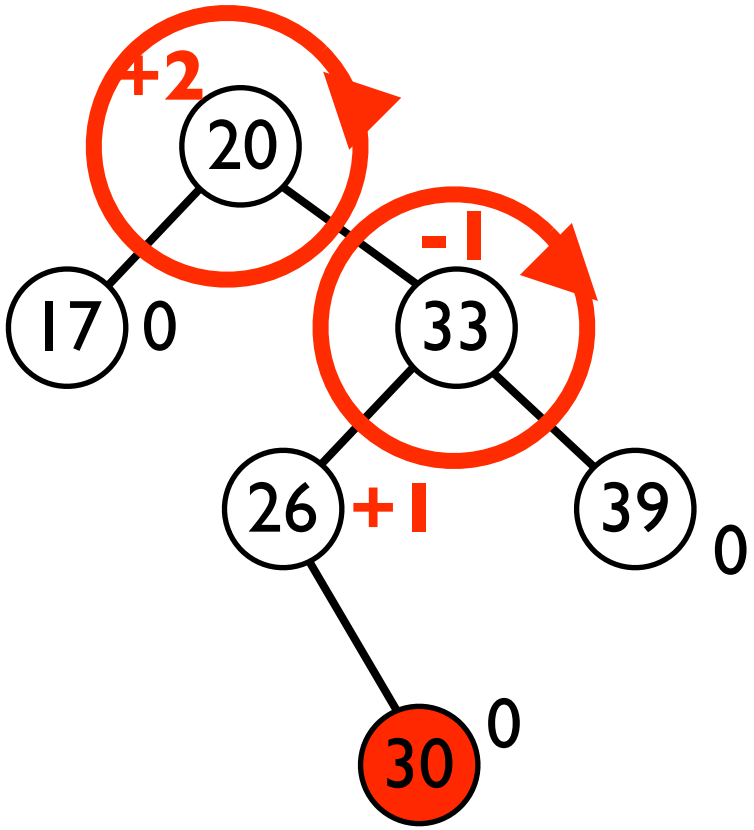


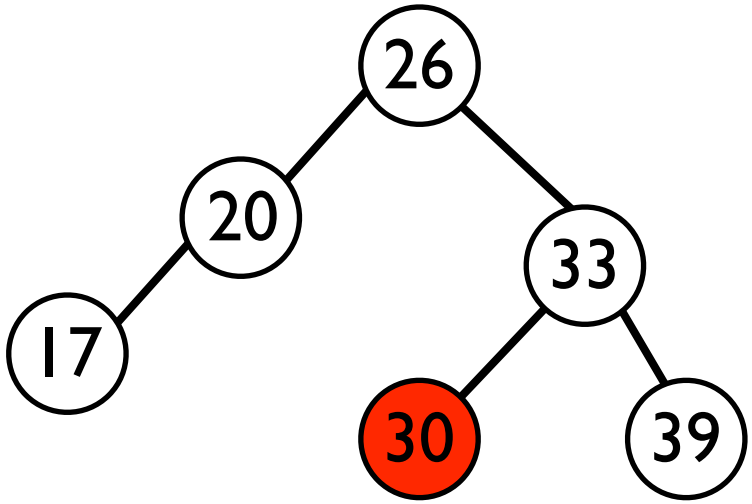


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

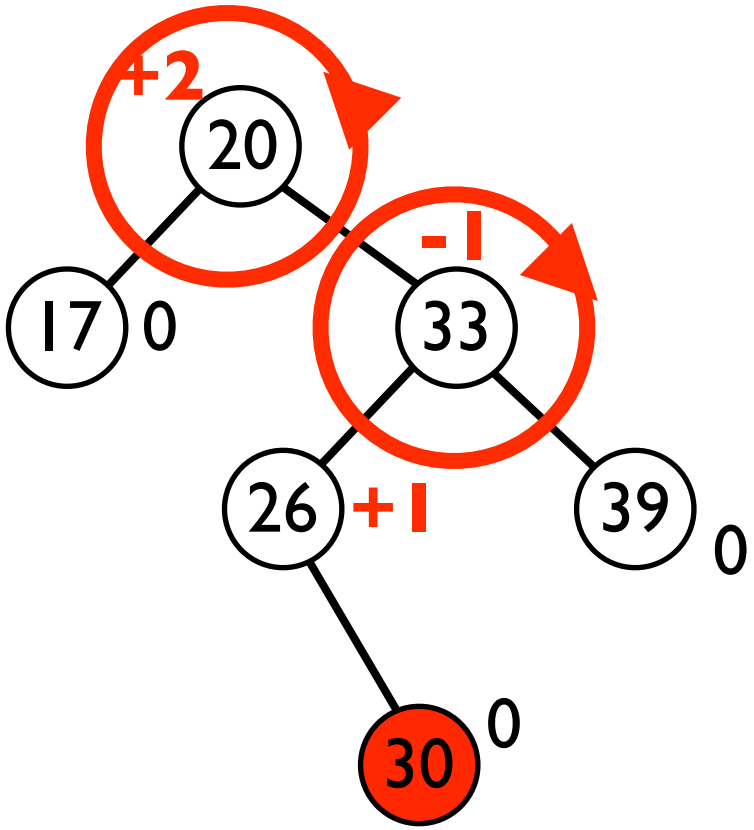


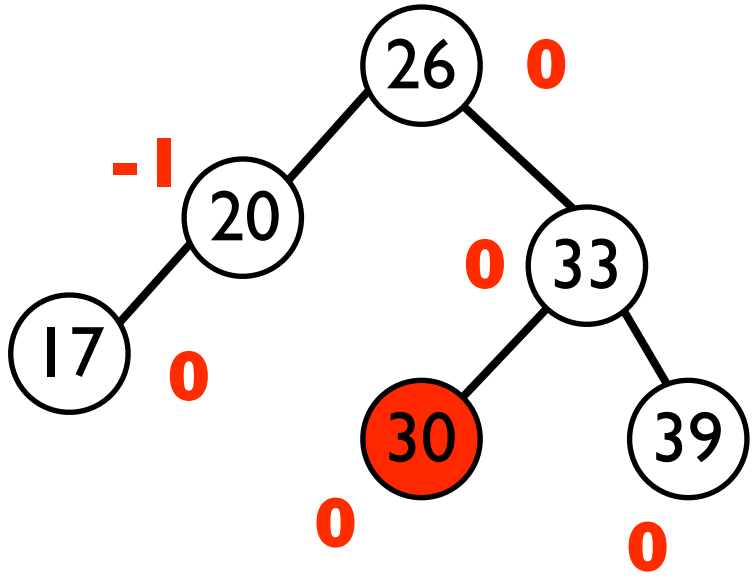


Rechtsrotation (R)
Linksrotation (L)

Suchen

Ausbalancieren - Beispiel: Einfügen

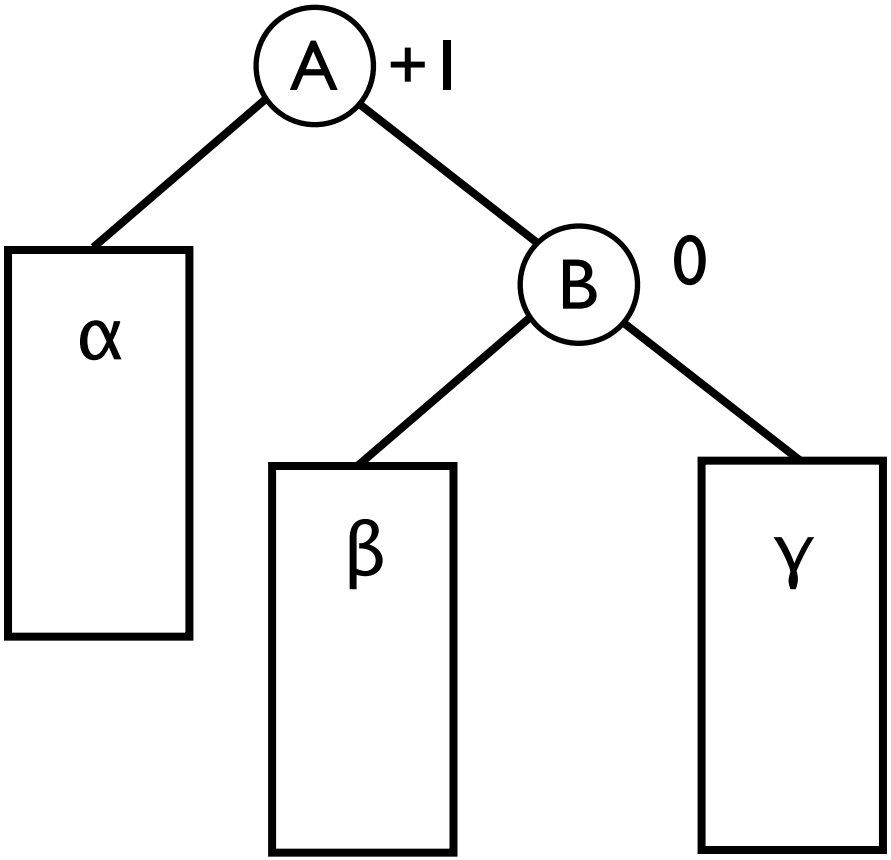




Rechtsrotation (R)
Linksrotation (L)

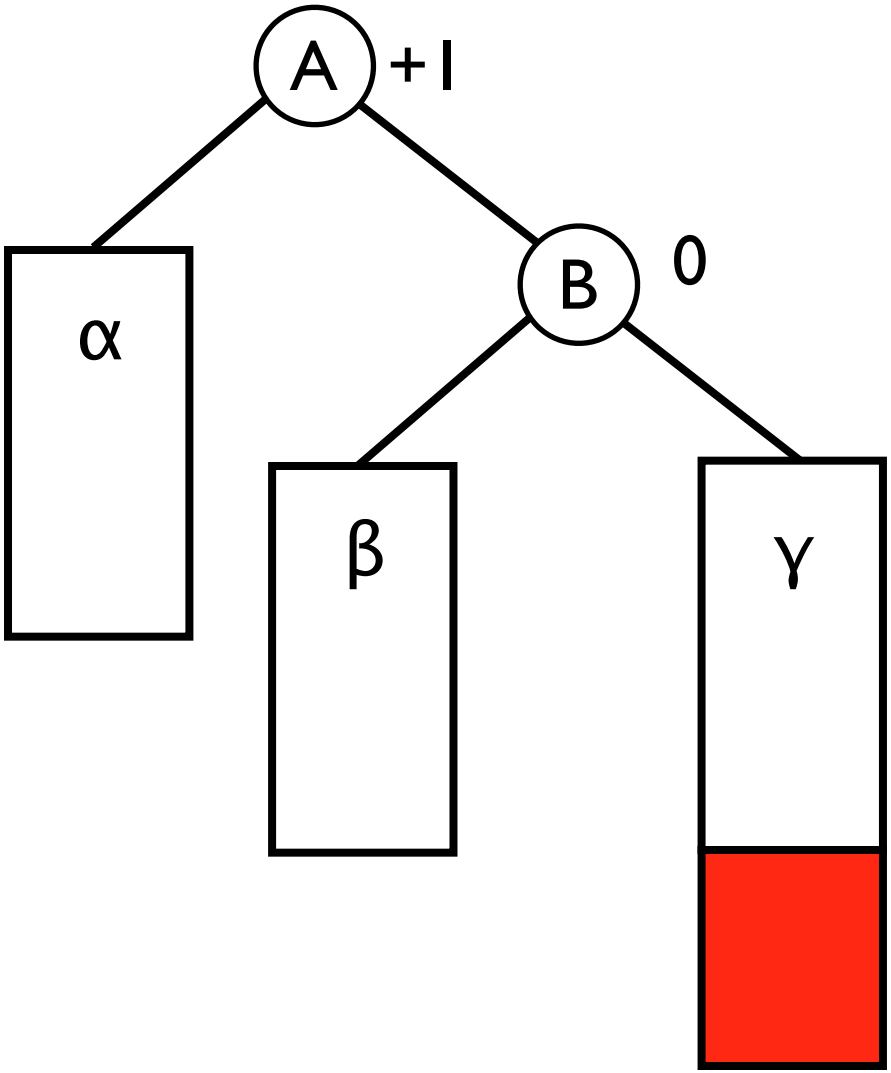
Suchen

Ausbalancieren - systematisch



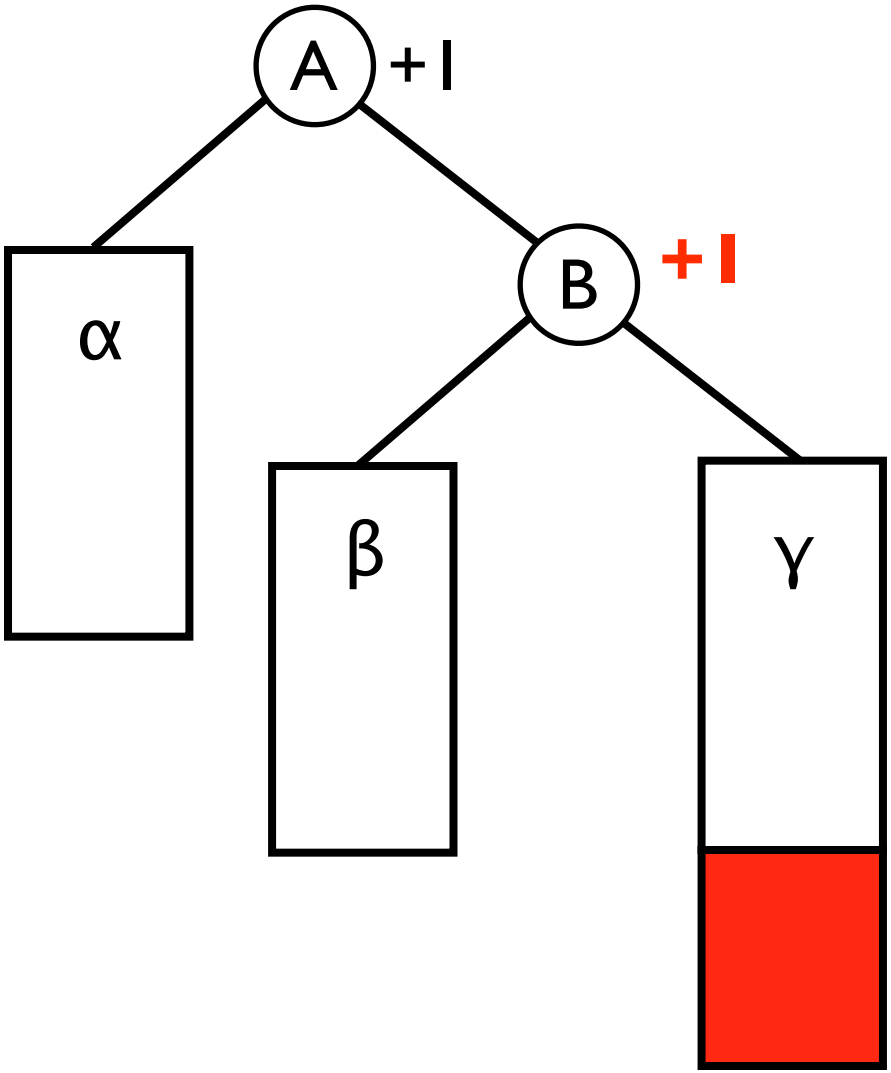
Suchen

Ausbalancieren - systematisch



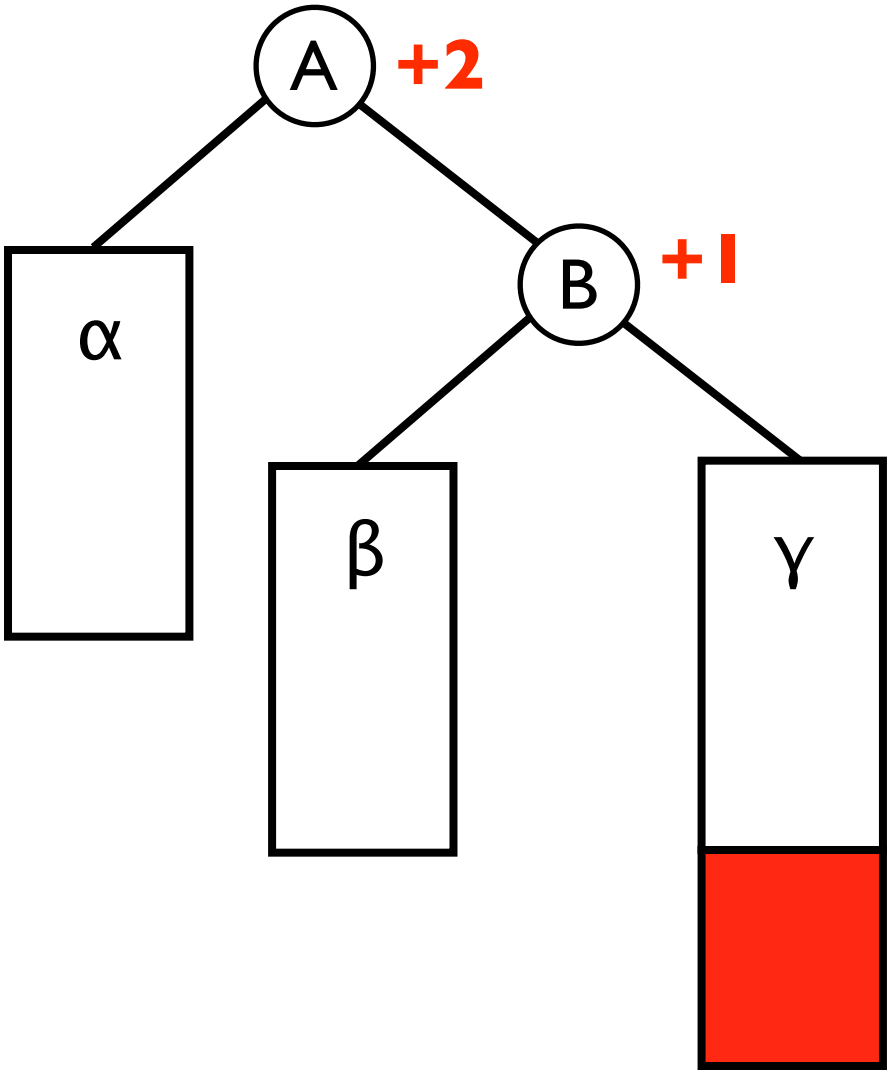
Suchen

Ausbalancieren - systematisch



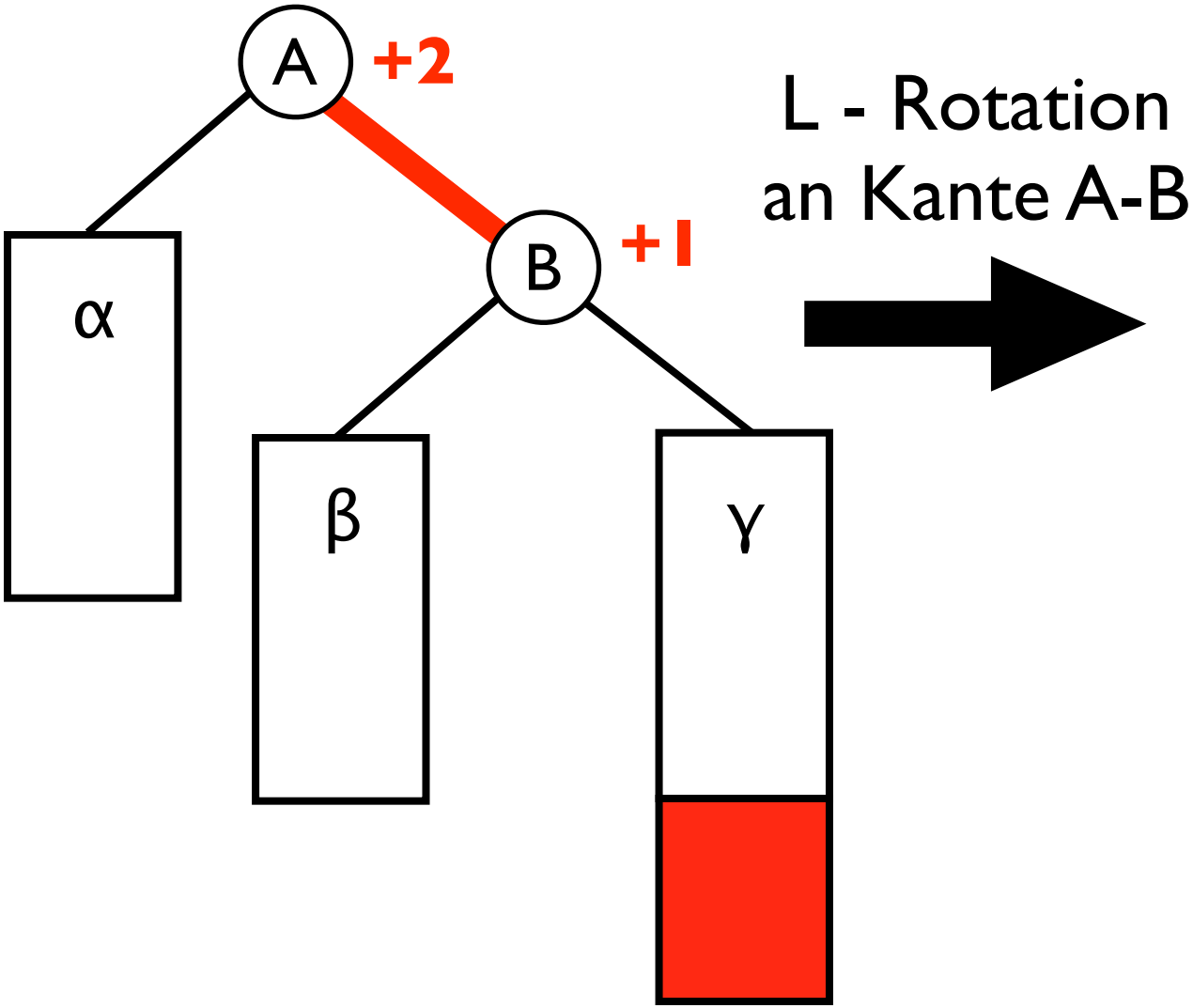
Suchen

Ausbalancieren - systematisch



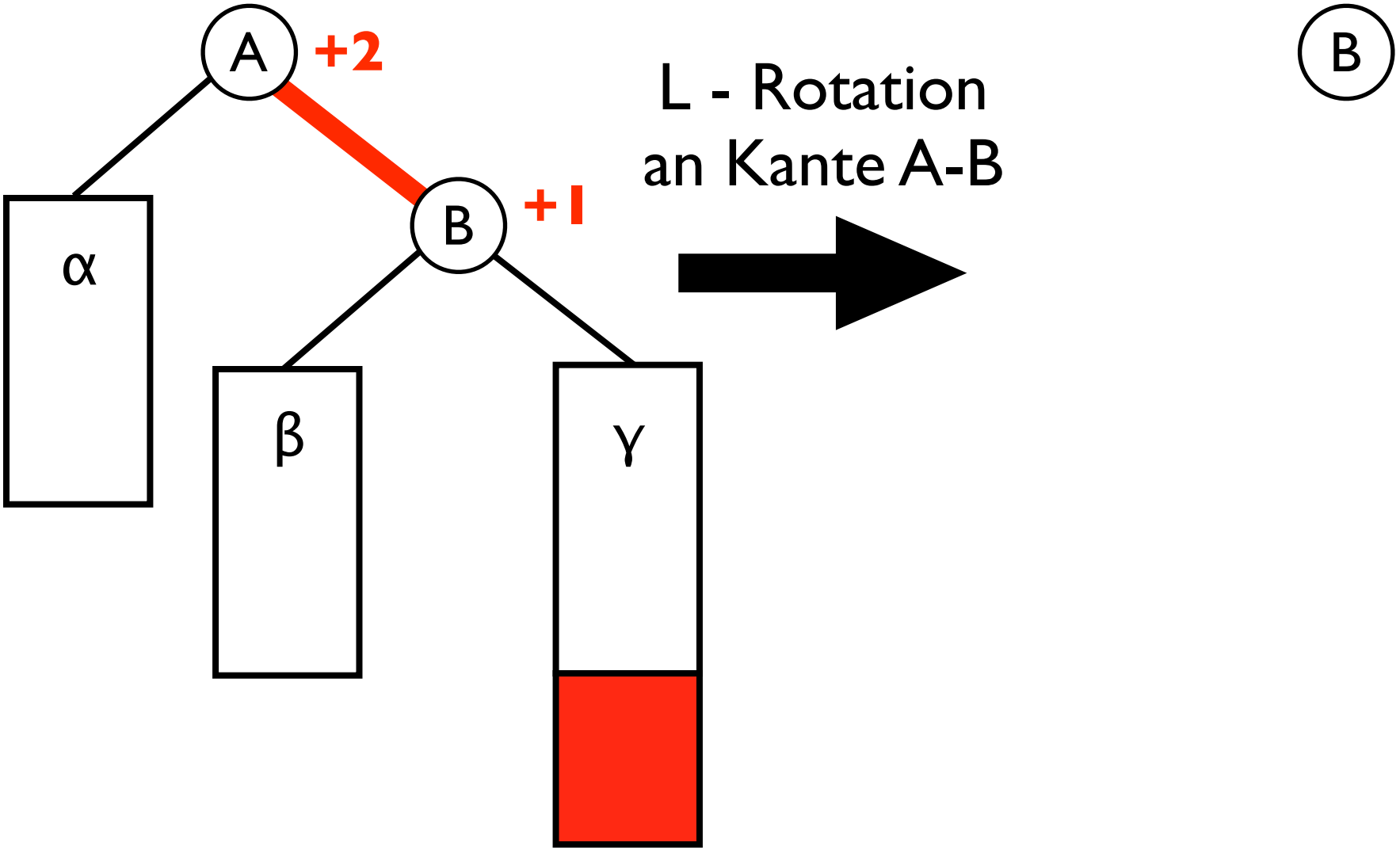
Suchen

Ausbalancieren - systematisch



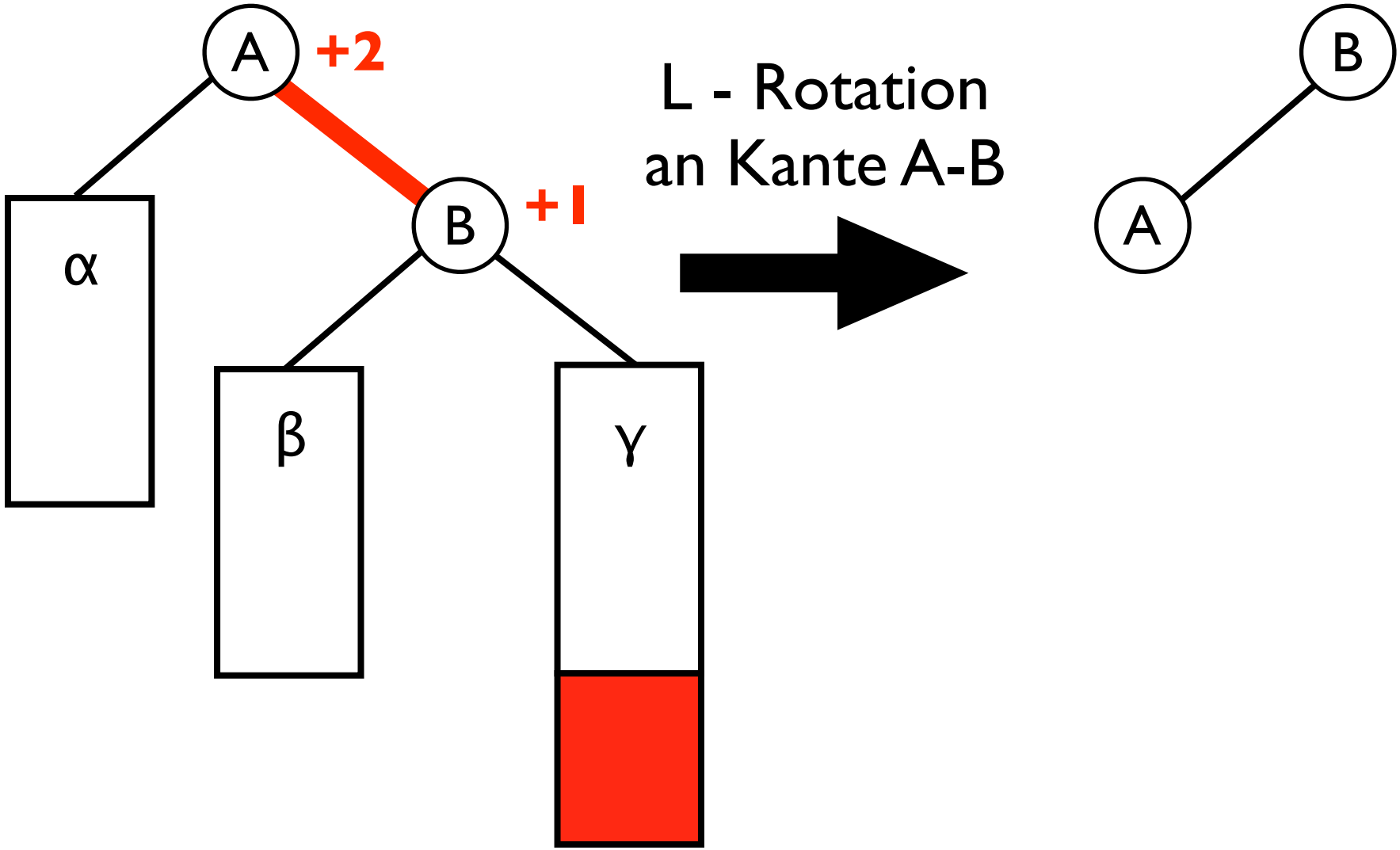
Suchen

Ausbalancieren - systematisch



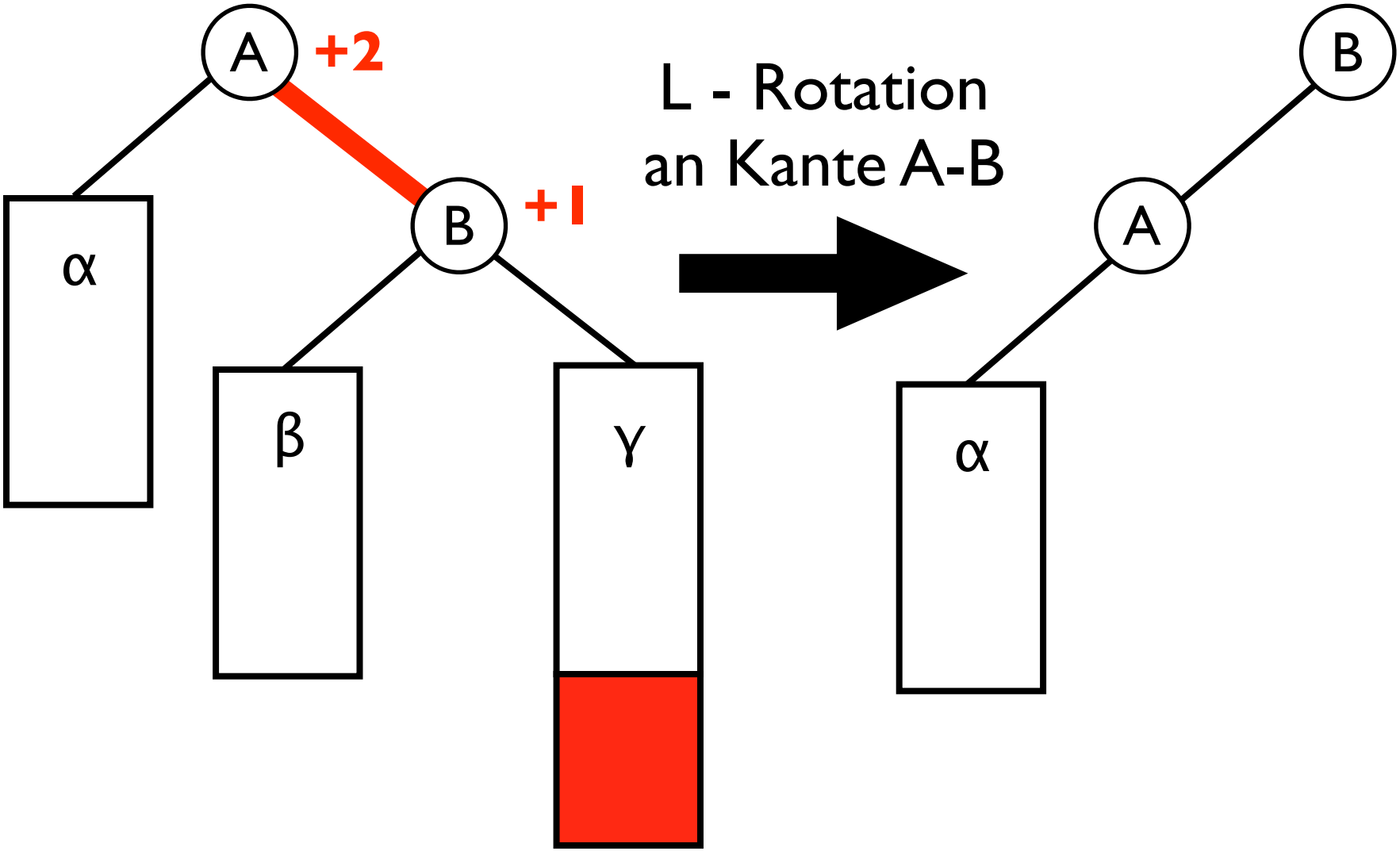
Suchen

Ausbalancieren - systematisch



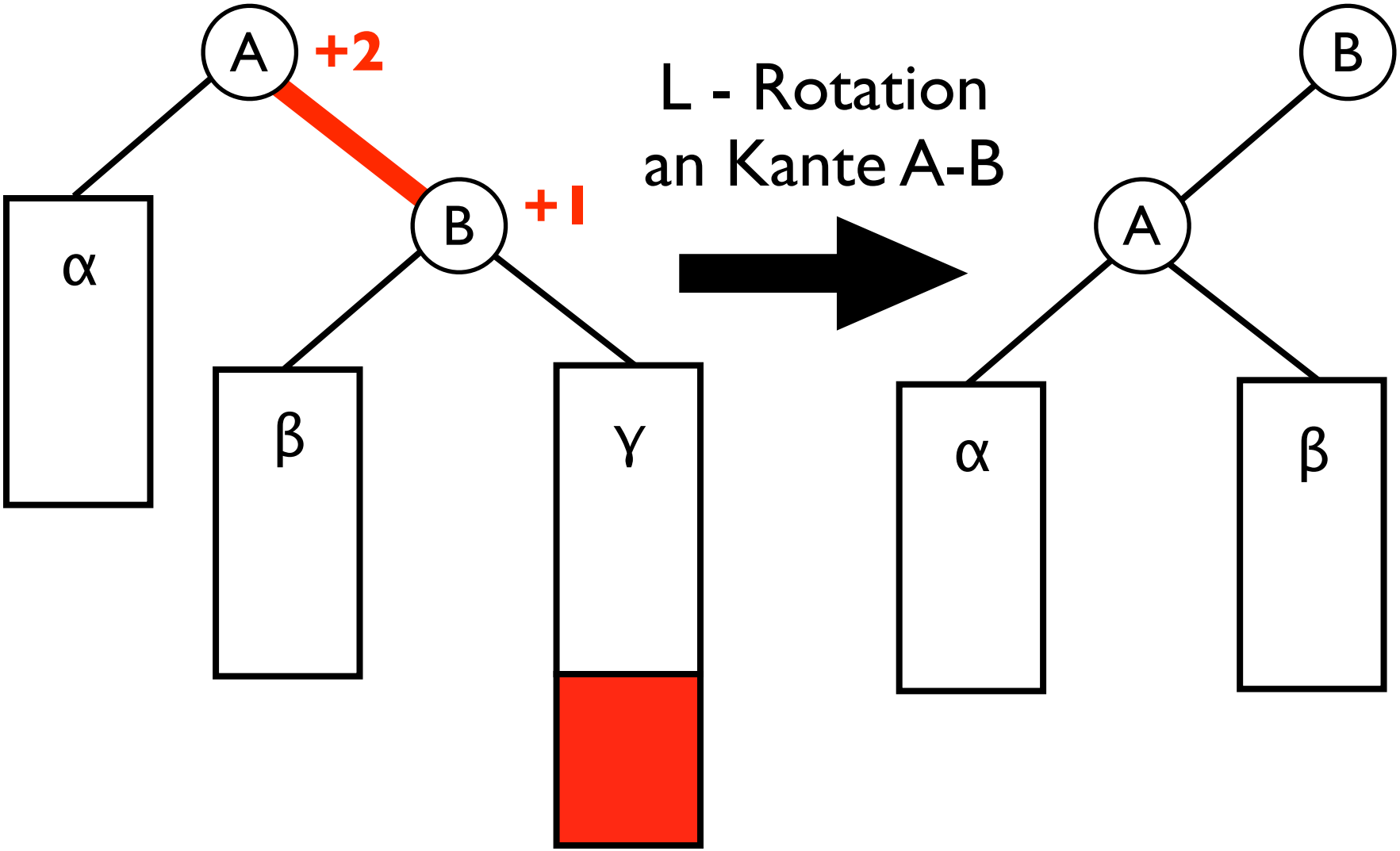
Suchen

Ausbalancieren - systematisch



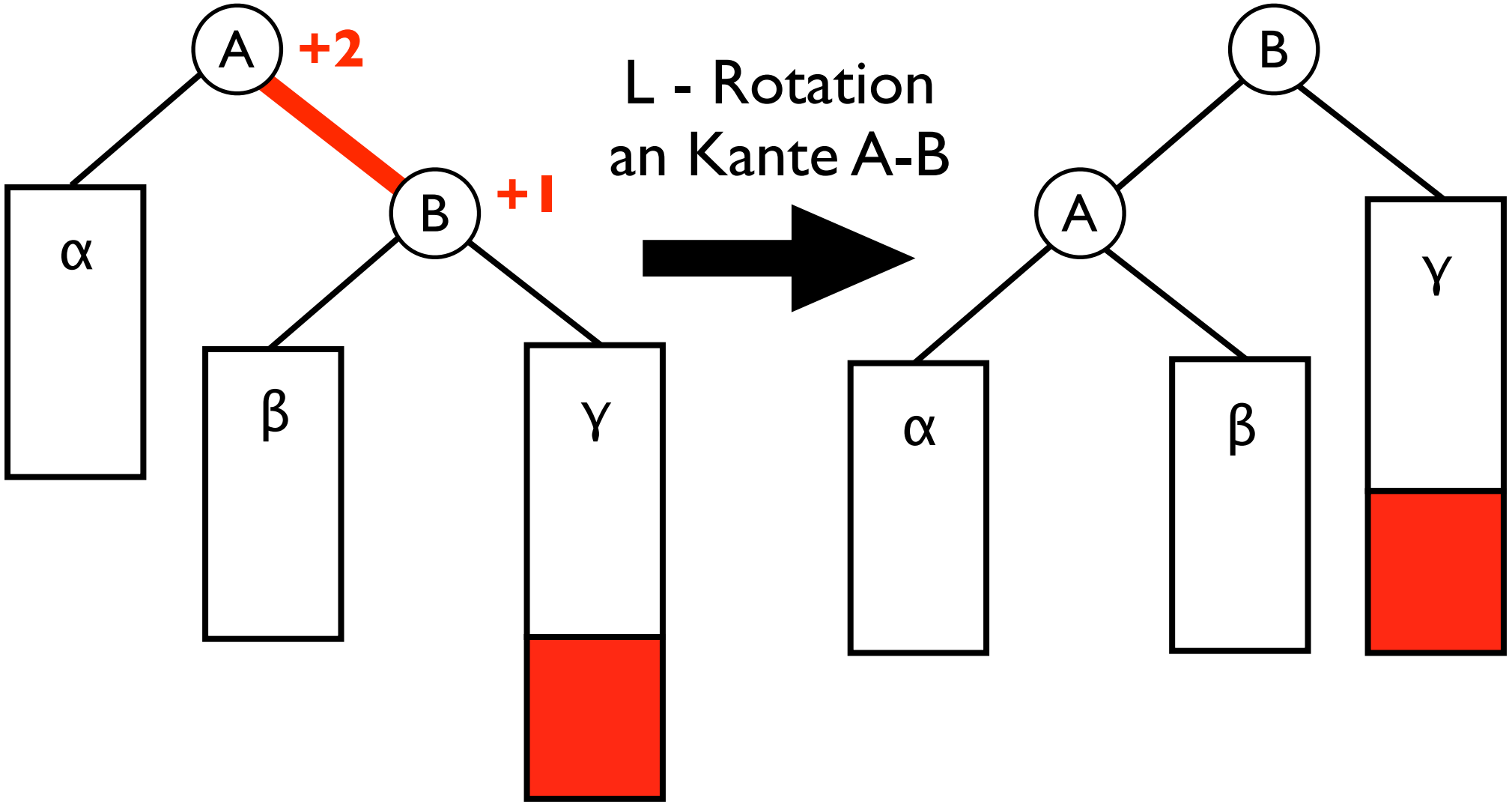
Suchen

Ausbalancieren - systematisch



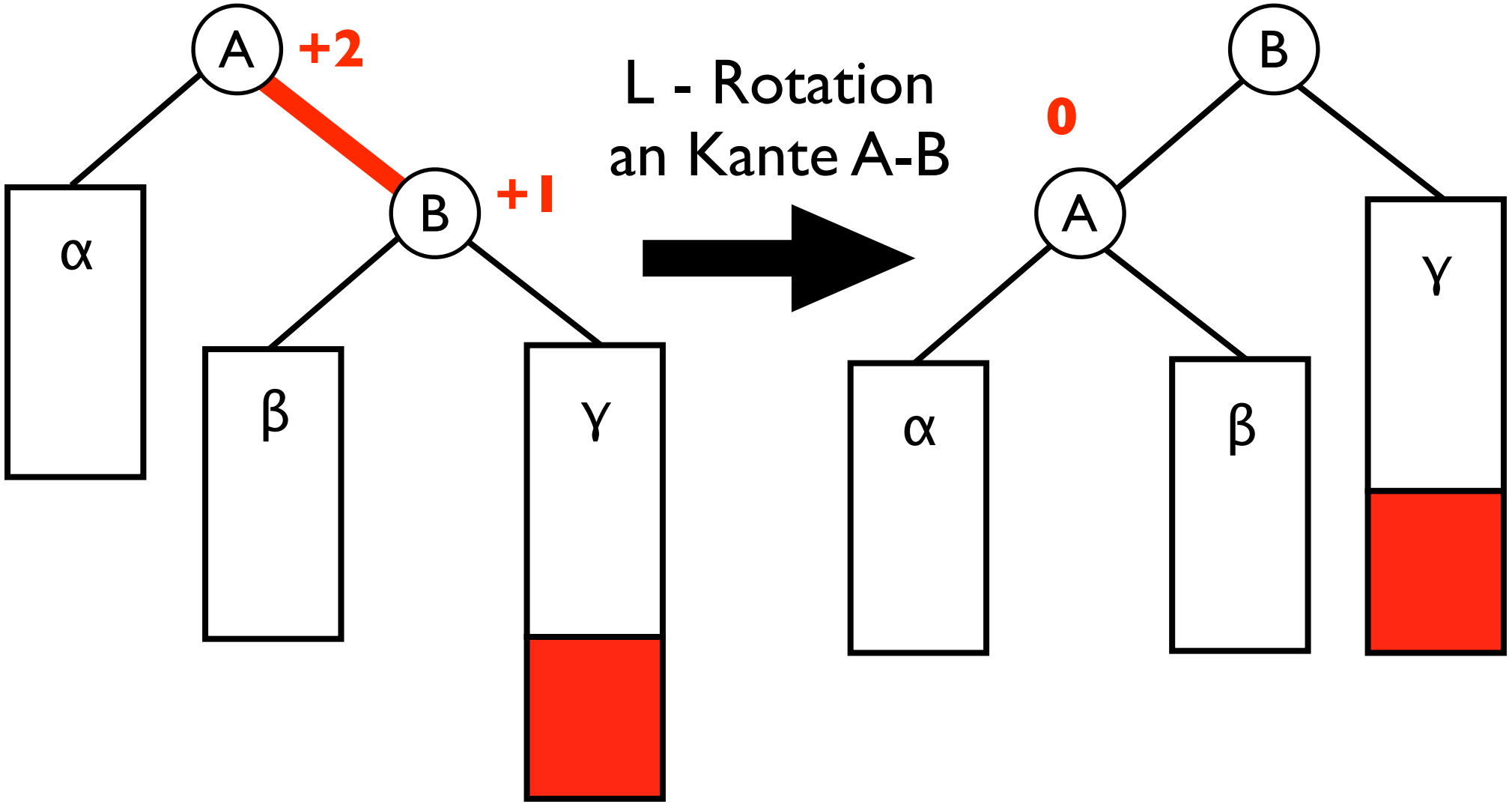
Suchen

Ausbalancieren - systematisch



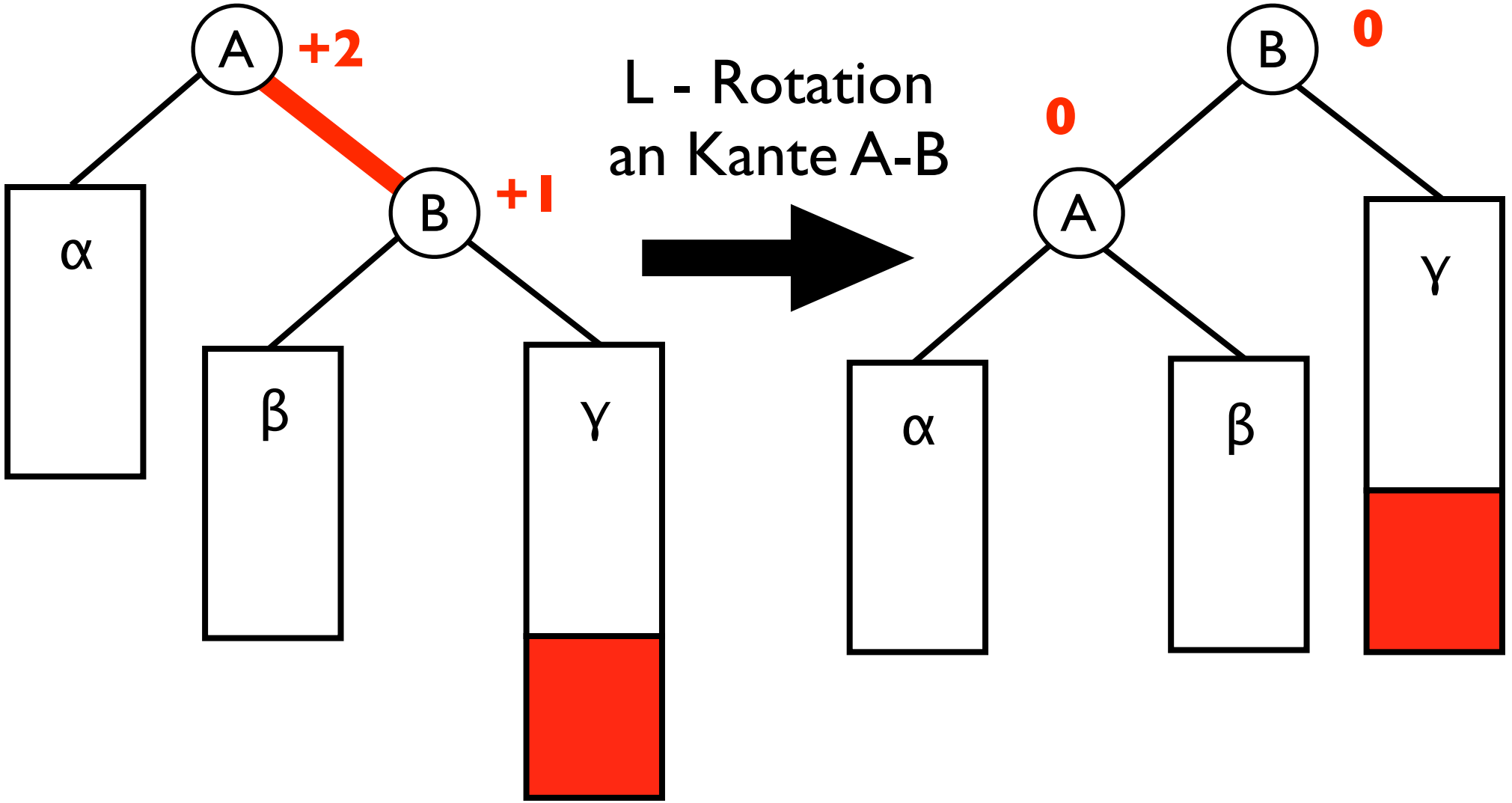
Suchen

Ausbalancieren - systematisch



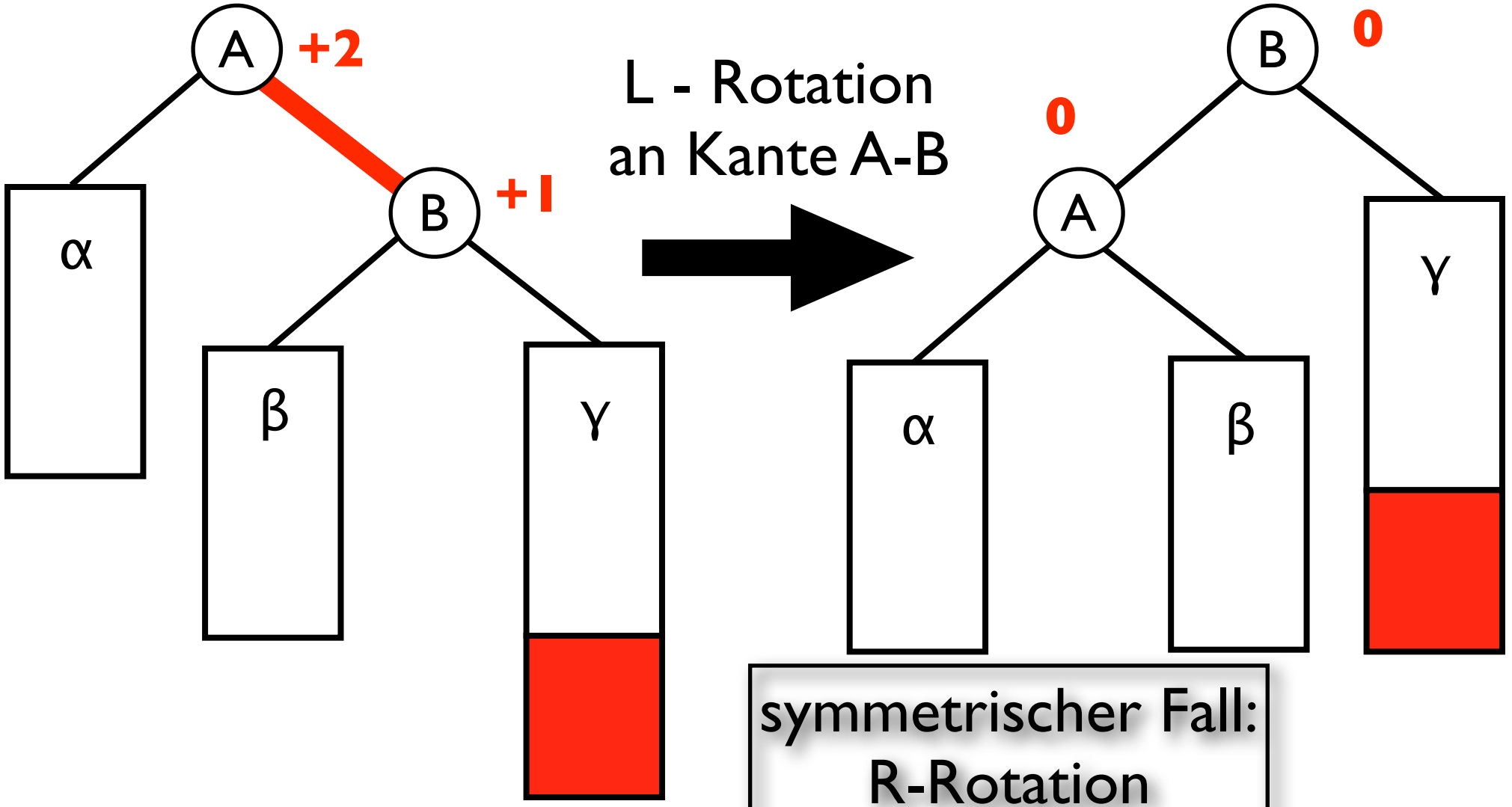
Suchen

Ausbalancieren - systematisch



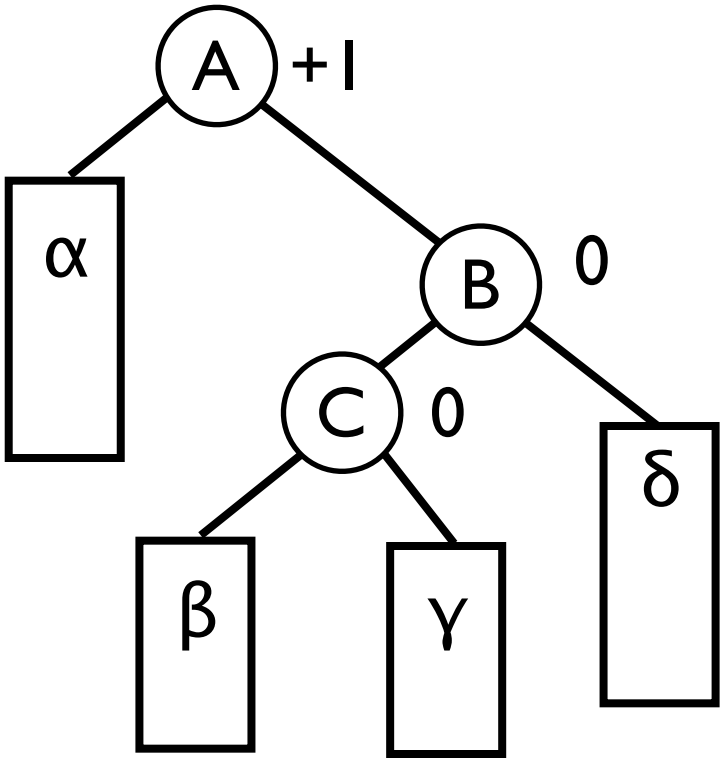
Suchen

Ausbalancieren - systematisch



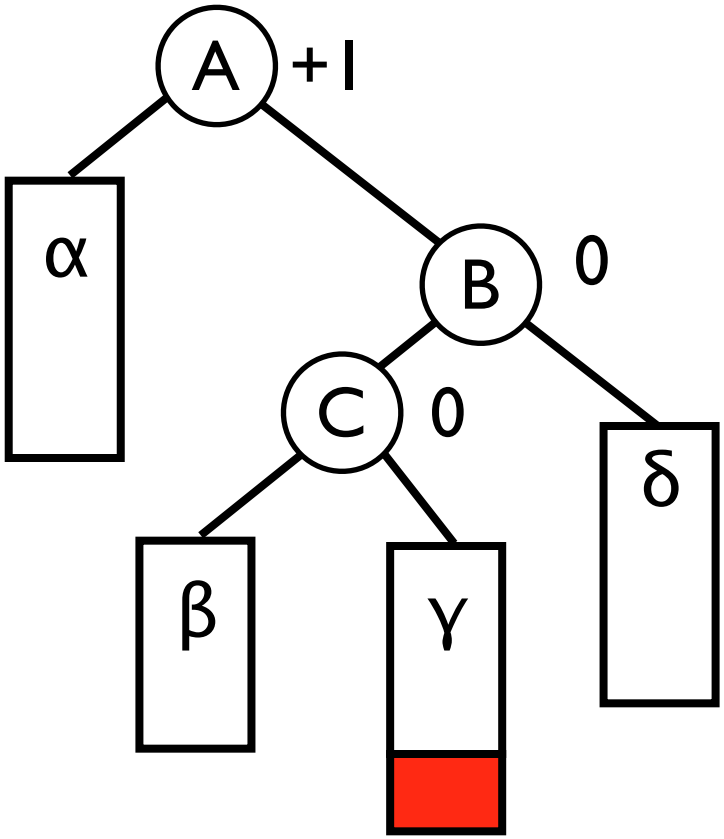
Suchen

Ausbalancieren - systematisch



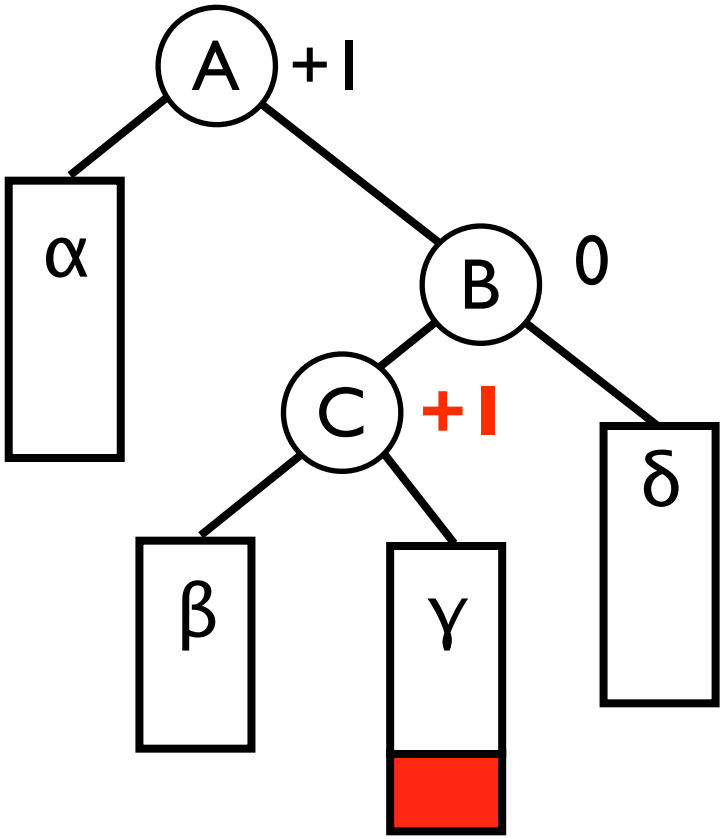
Suchen

Ausbalancieren - systematisch



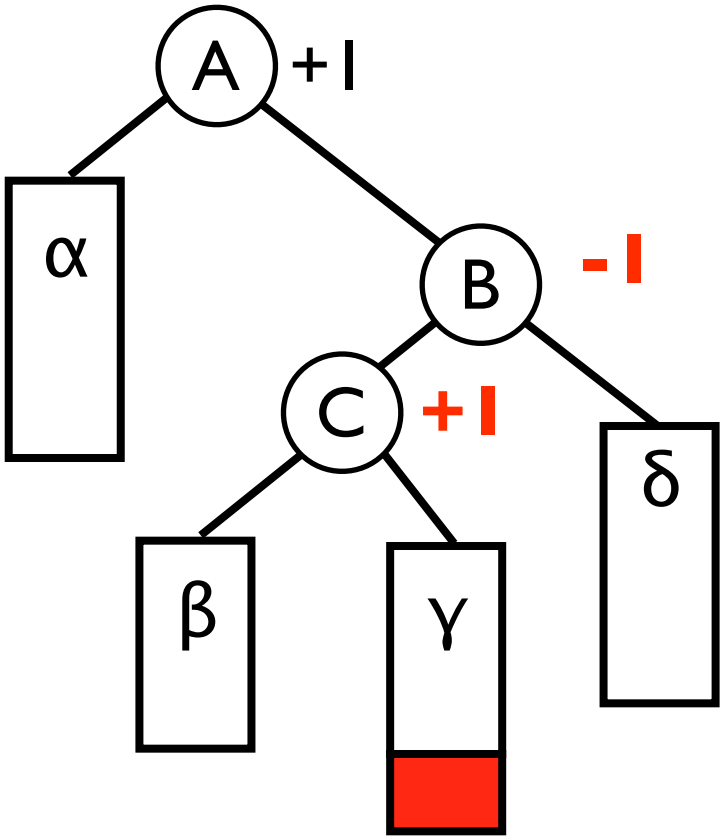
Suchen

Ausbalancieren - systematisch



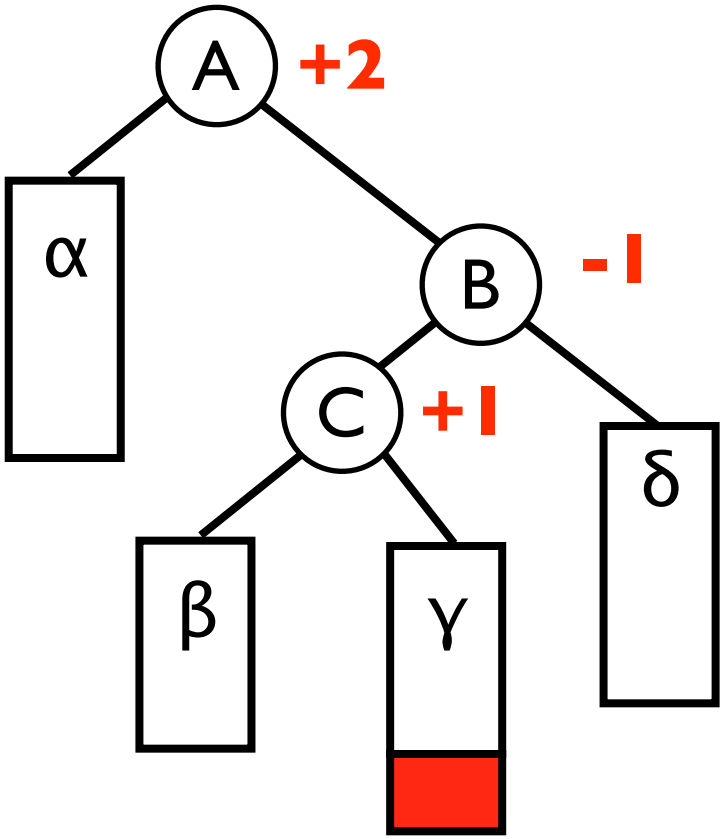
Suchen

Ausbalancieren - systematisch



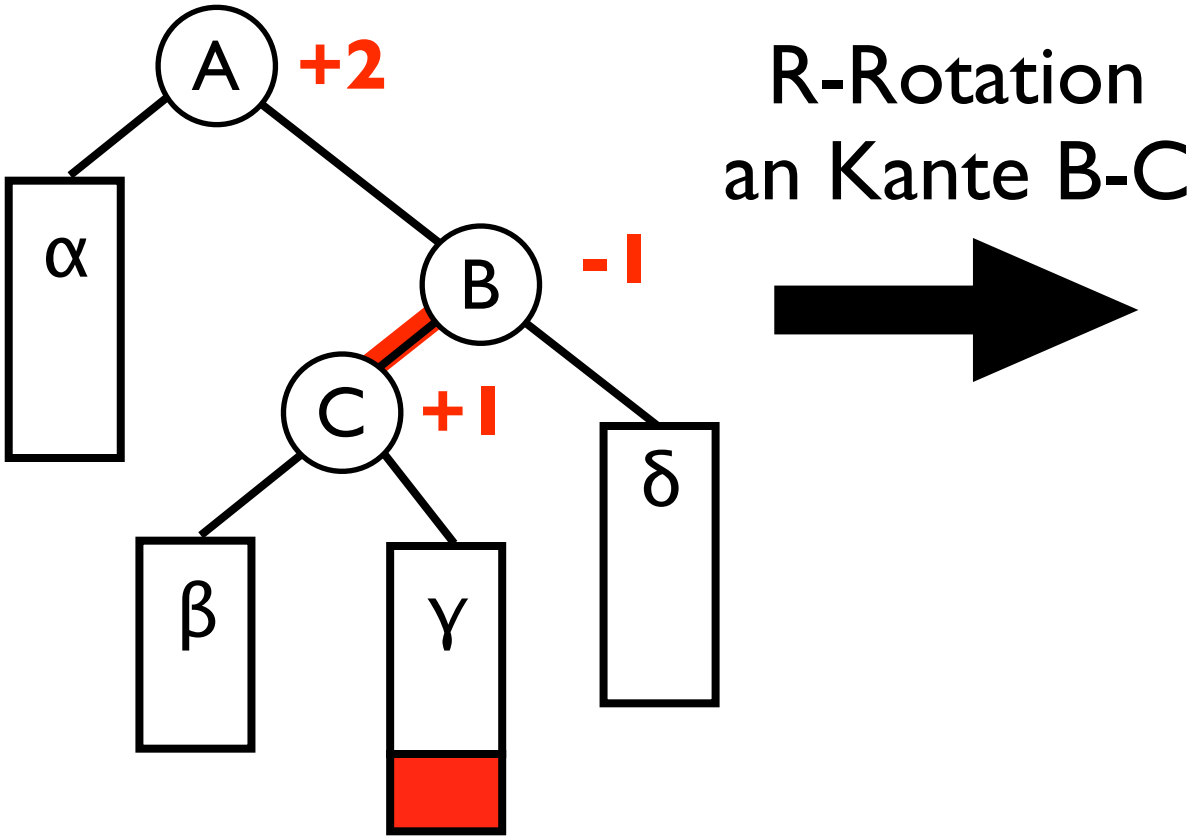
Suchen

Ausbalancieren - systematisch



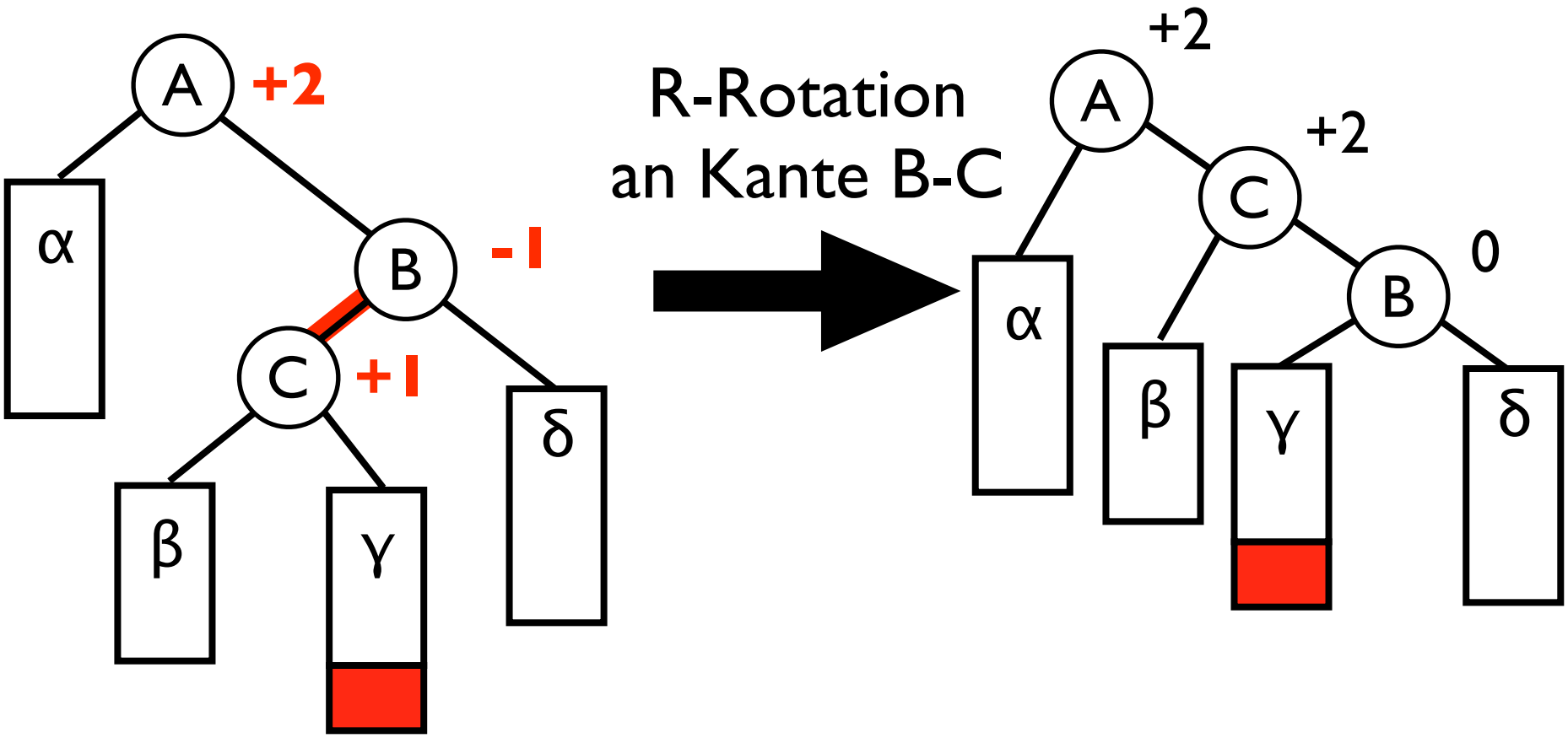
Suchen

Ausbalancieren - systematisch



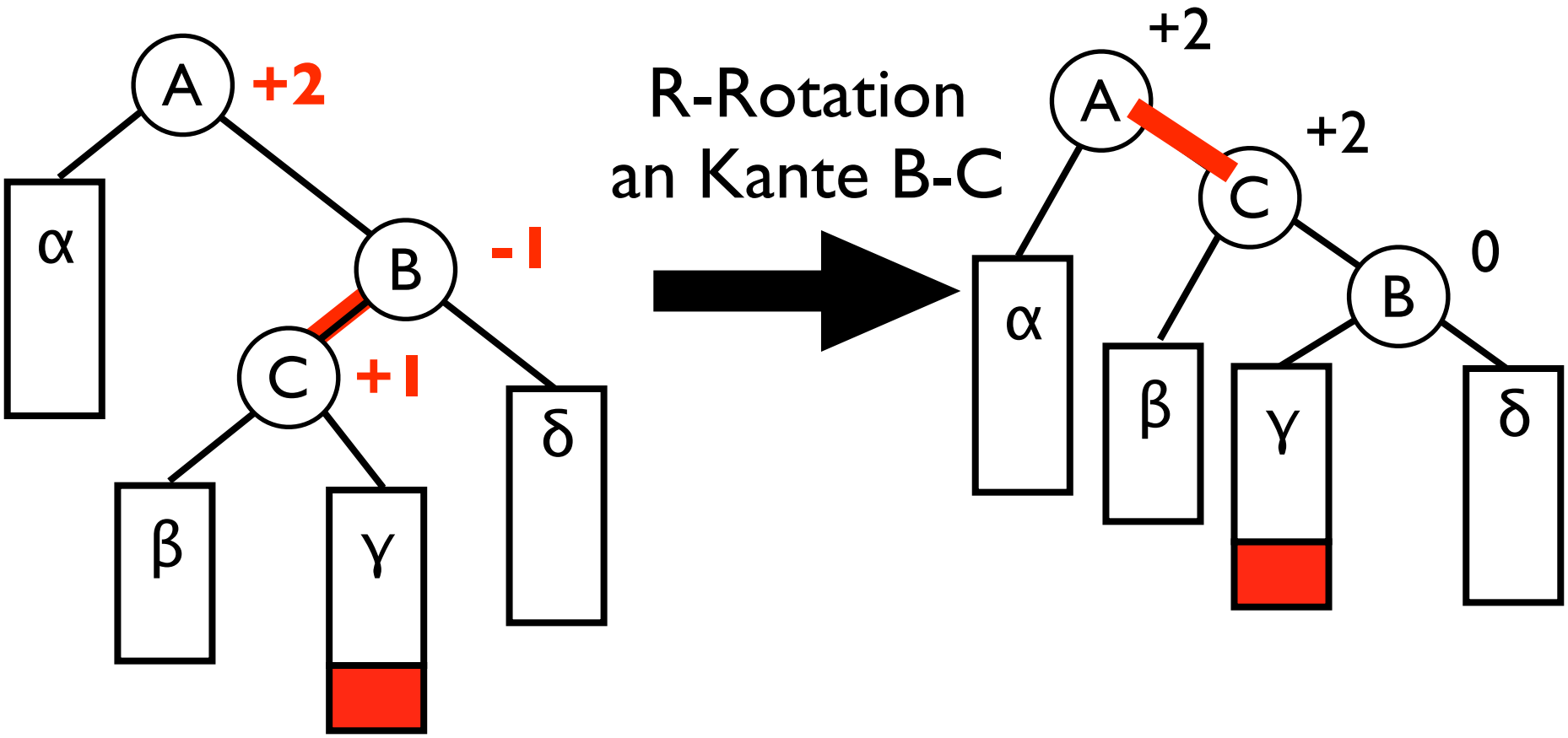
Suchen

Ausbalancieren - systematisch



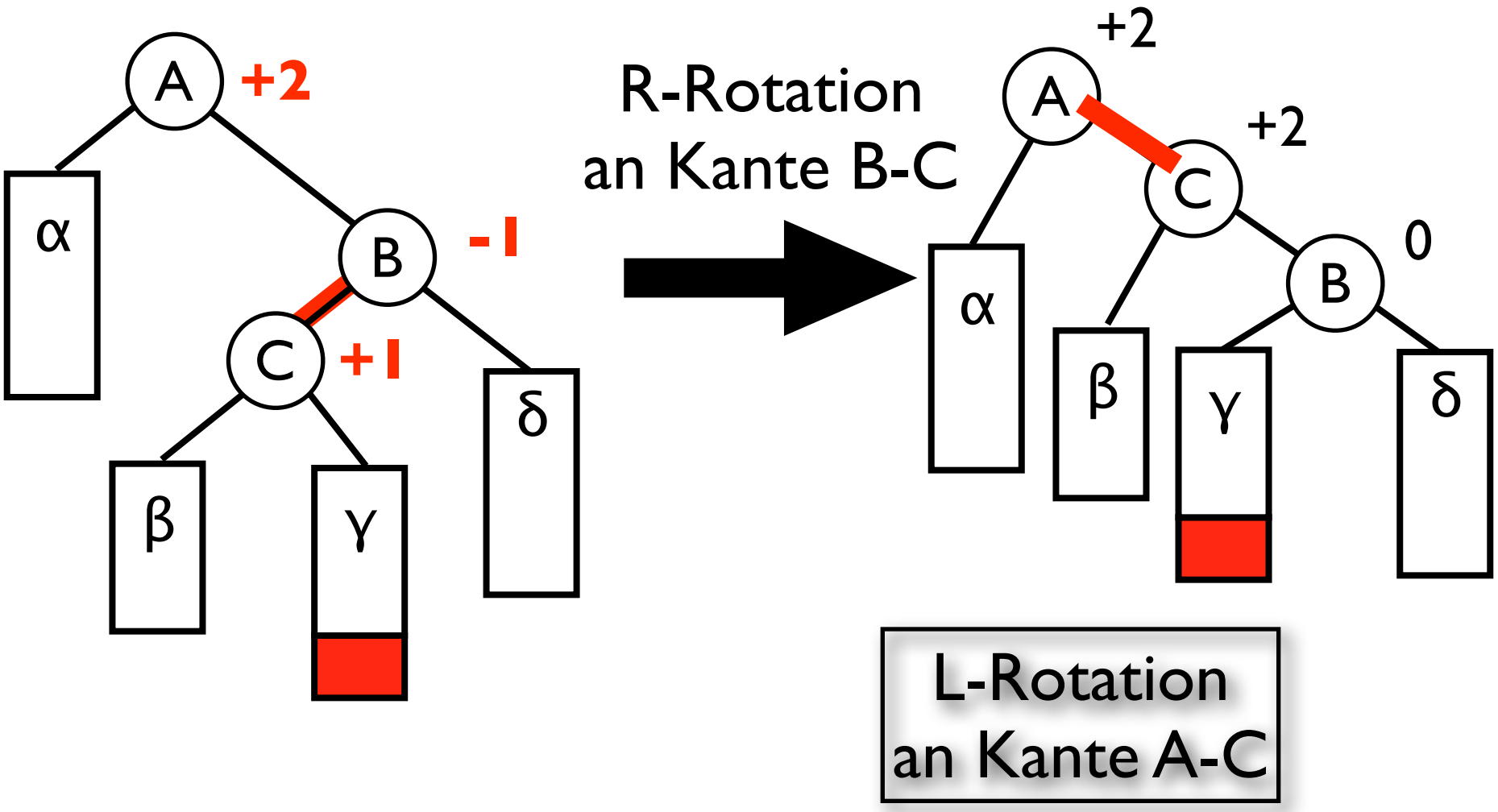
Suchen

Ausbalancieren - systematisch



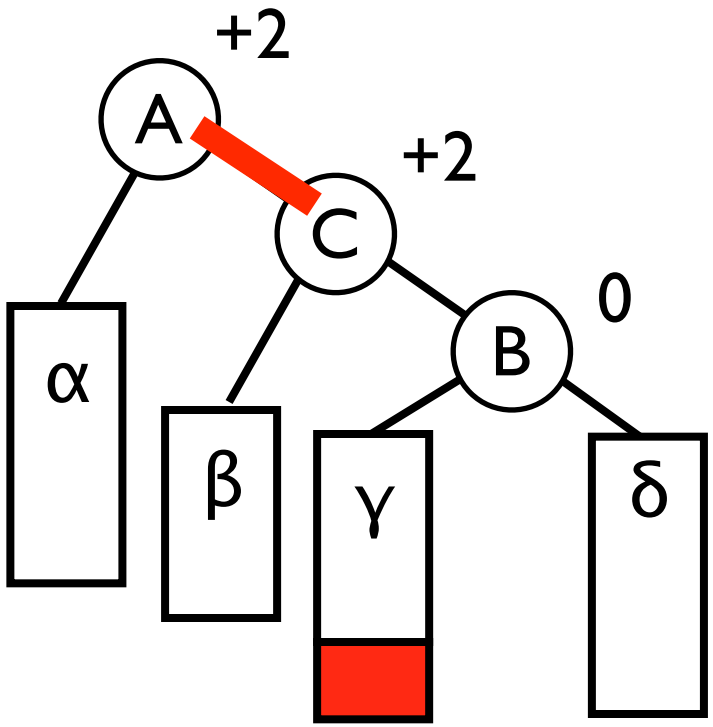
Suchen

Ausbalancieren - systematisch



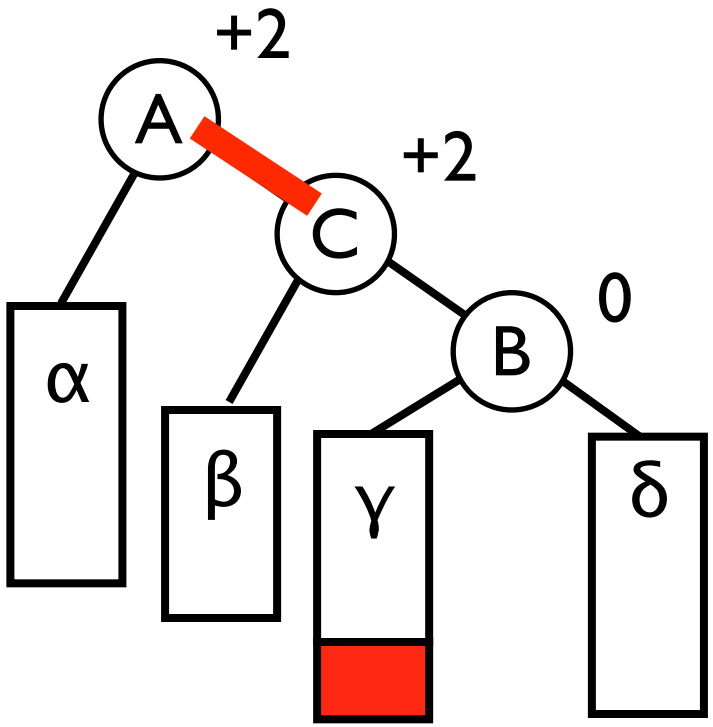
Suchen

Ausbalancieren - systematisch

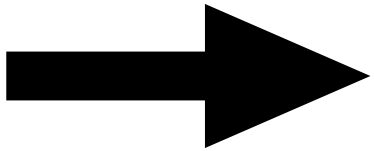


Suchen

Ausbalancieren - systematisch

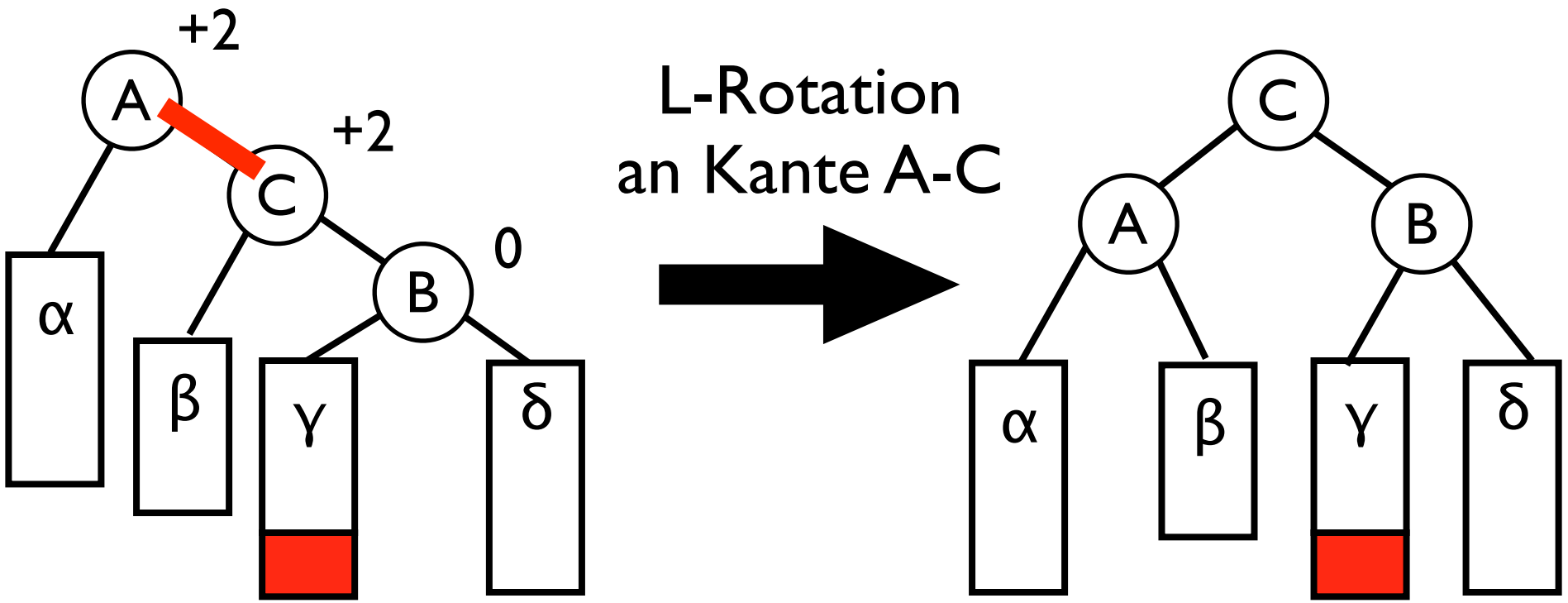


L-Rotation
an Kante A-C



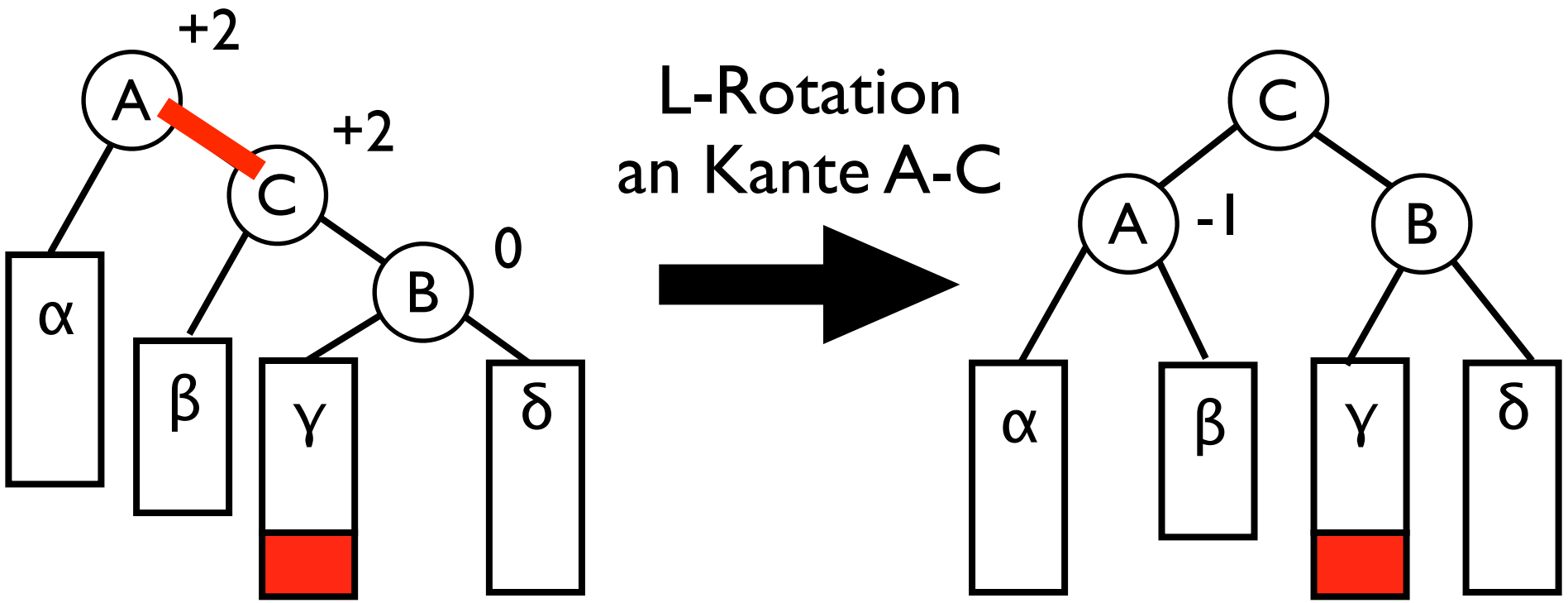
Suchen

Ausbalancieren - systematisch



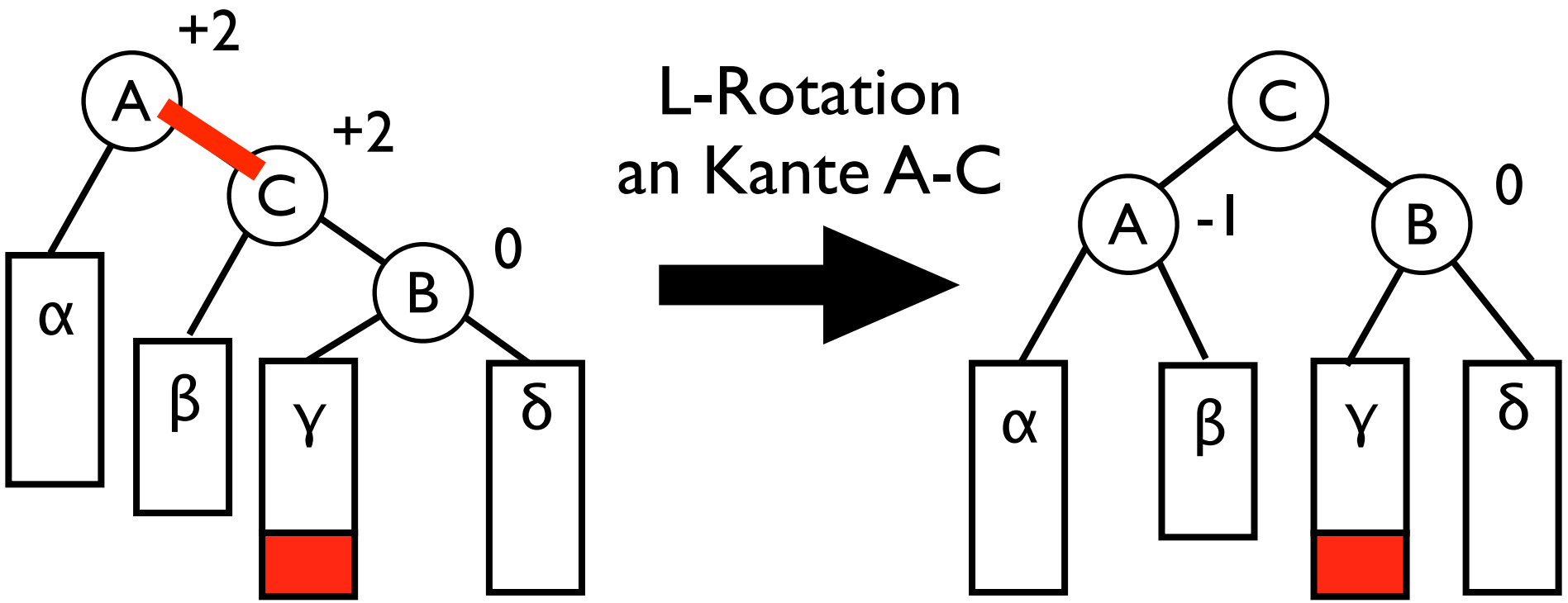
Suchen

Ausbalancieren - systematisch



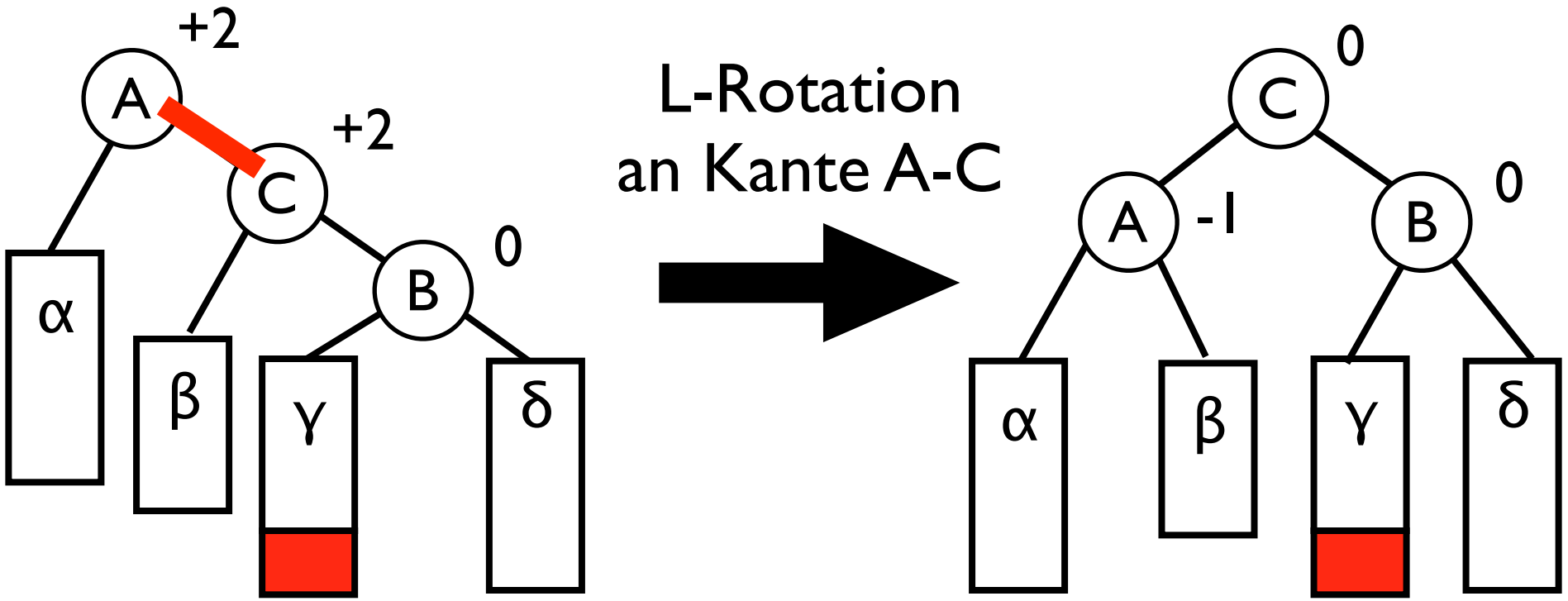
Suchen

Ausbalancieren - systematisch



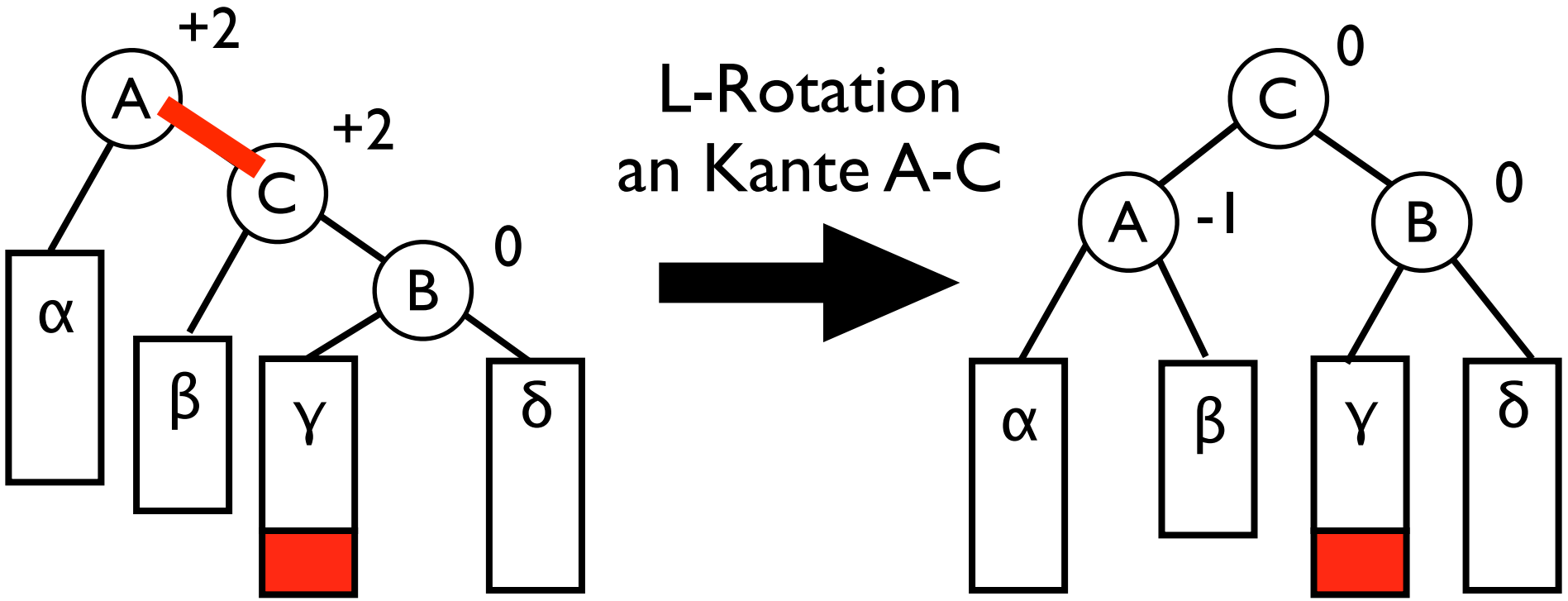
Suchen

Ausbalancieren - systematisch



Suchen

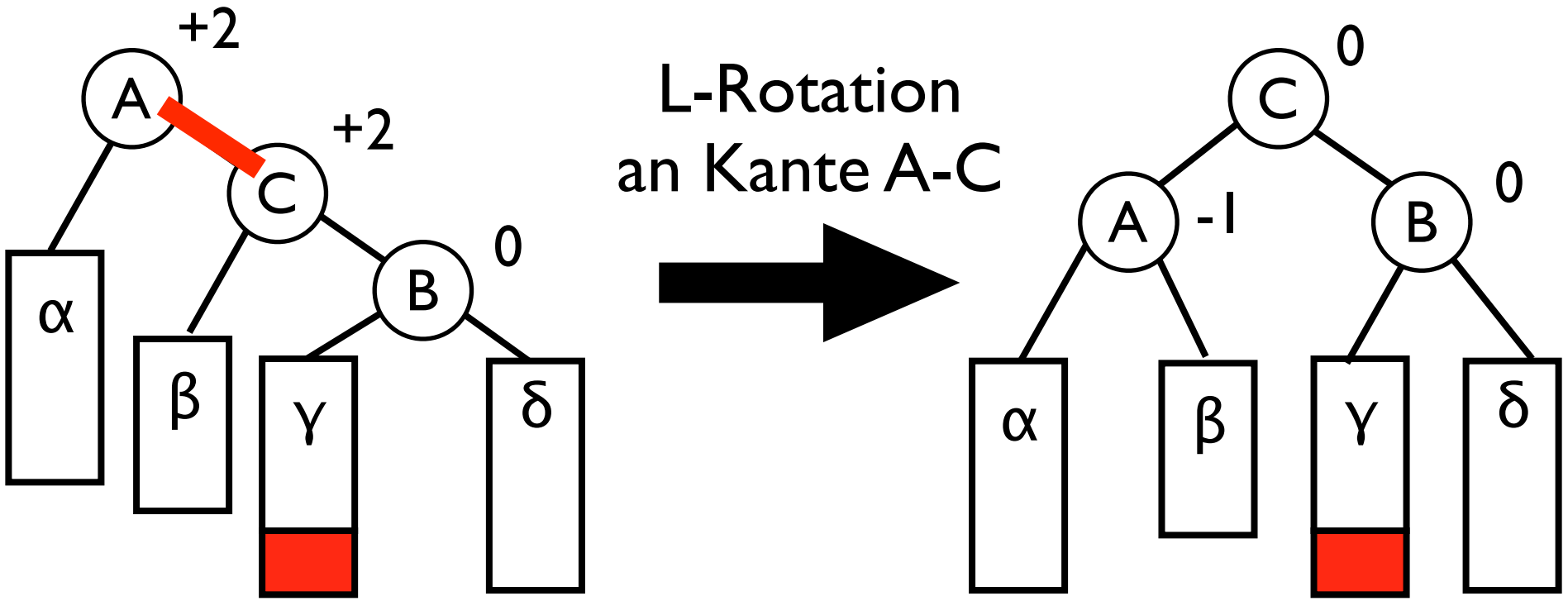
Ausbalancieren - systematisch



Insgesamt: RL-Rotation

Suchen

Ausbalancieren - systematisch



Insgesamt: RL-Rotation

symmetrischer Fall:
LR-Rotation

Suchen
AVL-Bäume - Implem.: Einfügen

- Trage Schlüssel in Baum ein
- Verfolge Weg zurück und verändere Balance
  - bis Balance von ±1 auf 0 abgeändert (=fertig)
- oder
  - Balance auf ±2 abgeändert wird (= rotieren)

- Beobachtung: max. **eine** Balance-Operation nötig

Suchen
AVL-Bäume - Implem.: Löschen

- Lösche Schlüssel im Baum
- Verfolge Weg zurück und verändere Balancen + Ausgleichsoperationen

- Beobachtung: max. an **jedem** Knoten vom Blatt zur Wurzel **eine** Balance-Operation nötig: O(log n) Ausgleichsoperationen mit jeweils **konstanter** (!) Zeit

- Zeige noch: AVL-Bäume haben logarithmische Höhe

Suchen
AVL-Bäume - Höhe

Satz:

AVL-Bäume mit n Knoten haben eine Höhe von O(log n).

Suchen
AVL-Bäume - Höhe

Beweis:

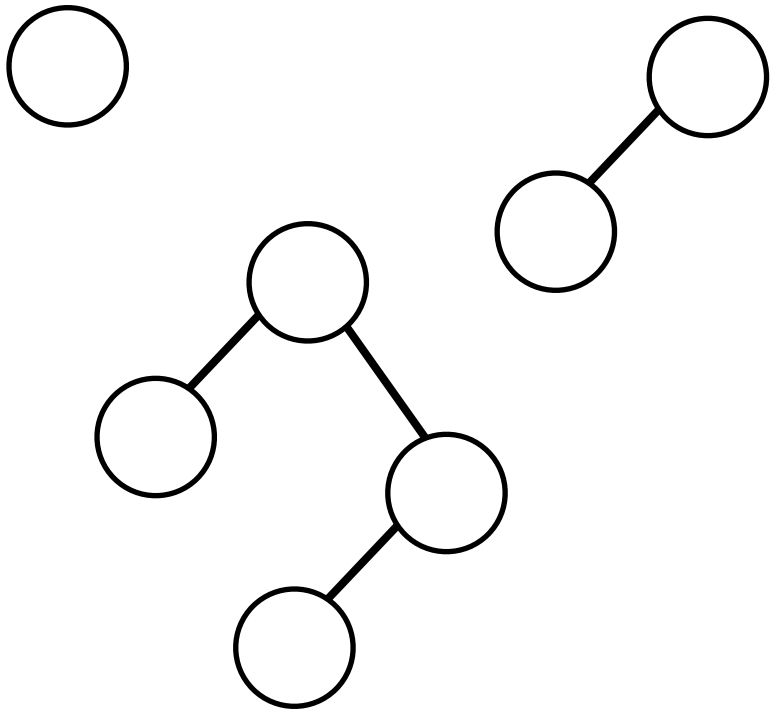
Wieviele Knoten F(h) enthält ein AVL-Baum der Höhe h mindestens?

Höhe 1: 1 Knoten

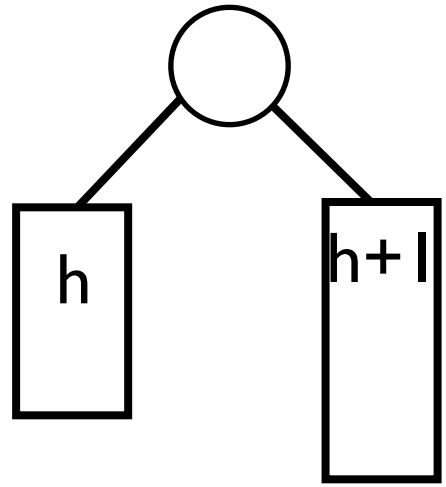
Höhe 2: 2 Knoten

Höhe 3: 4 Knoten

Höhe 4: 7 Knoten



Suchen
AVL-Bäume - Höhe



Höhe h+2:

$F(h+2)=F(h)+F(h+1)+1 \geq F(h)+F(h+1)$,

$F(0)=0, F(1)=1$

Fibonacci-Funktion

Näherung: $F(h) \approx 0{,}72 \cdot 1{,}6^h$

Anzahl der Knoten wächst exponentiell mit h.

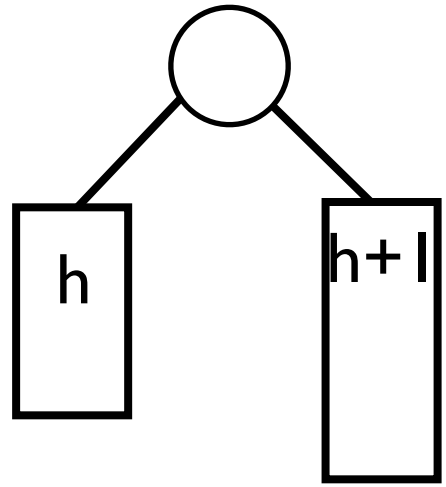
Anz. Knoten $n \geq F(h) \approx 0{,}72 \cdot 1{,}6^h = 1{,}15 \cdot 1{,}6^{h-1}$

$\Rightarrow h \leq (\log_2 n - \log_2 1{,}15)/(\log_2 1{,}6)+1$

$\Rightarrow h \leq 1{,}47 \cdot \log_2 n + 1 = O(\log_2 n)$

Ergebnis: ca. 47 % Verlust an Höhe gegenüber einem optimal ausgeglichenen Baum

Suchen
AVL-Bäume - Höhe



Höhe h+2:

$F(h+2)=F(h)+F(h+1)+1 \geq F(h)+F(h+1)$,

$F(0)=0, F(1)=1$

Fibonacci-Funktion

Näherung: $F(h) \approx 0{,}72 \cdot 1{,}6^h$

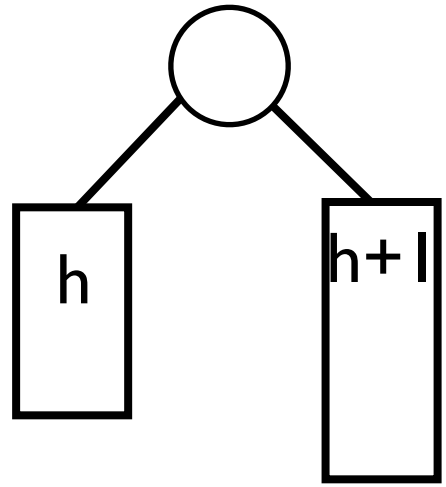
F (h) ist die $\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+1}$ nächstgel. Zahl

$\Rightarrow h \leq (\log_2 n - \log_2 1{,}19)/(\log_2 1{,}6) + 1$

$\Rightarrow h \leq 1{,}47 \cdot \log_2 n + 1 = O(\log_2 n)$

Ergebnis: ca. 47 % Verlust an Höhe gegenüber einem optimal ausgeglichenen Baum

Suchen
AVL-Bäume - Höhe



Höhe h+2:

$F(h+2)=F(h)+F(h+1)+1 \geq F(h)+F(h+1)$,

$F(0)=0, F(1)=1$

Fibonacci-Funktion

Näherung: $F(h) \approx 0{,}72 \cdot 1{,}6^h$

Anzahl der Knoten wächst exponentiell mit h.

Anz. Knoten $n \geq F(h) \approx 0{,}72 \cdot 1{,}6^h = 1{,}15 \cdot 1{,}6^{h-1}$

$\Rightarrow h \leq (\log_2 n - \log_2 1{,}15)/(\log_2 1{,}6)+1$

$\Rightarrow h \leq 1{,}47 \cdot \log_2 n + 1 = O(\log_2 n)$

Ergebnis: ca. 47 % Verlust an Höhe gegenüber einem optimal ausgeglichenen Baum

Suchen
AVL-Bäume - Laufzeit

Folgerung:

Suchen, Einfügen, Löschen in AVL-Bäumen erfordern im schlimmsten Fall O(log n) Zeit.

Suchen
AVL-Bäume - Schlußbeispiel

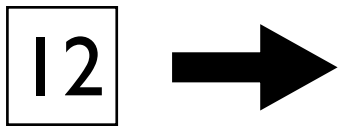
- Aufbau eines AVL-Baums mit der Zahlenfolge 12, 17, 13, 6, 10, 9, 5, 4, 15

Suchen
AVL-Bäume - Schlußbeispiel

- Aufbau eines AVL-Baums mit der Zahlenfolge 12, 17, 13, 6, 10, 9, 5, 4, 15

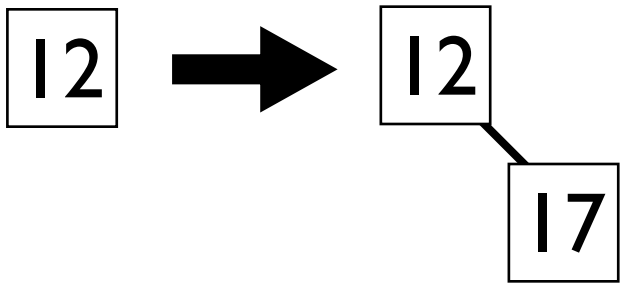
Suchen
AVL-Bäume - Schlußbeispiel

- Aufbau eines AVL-Baums mit der Zahlenfolge 12, 17, 13, 6, 10, 9, 5, 4, 15



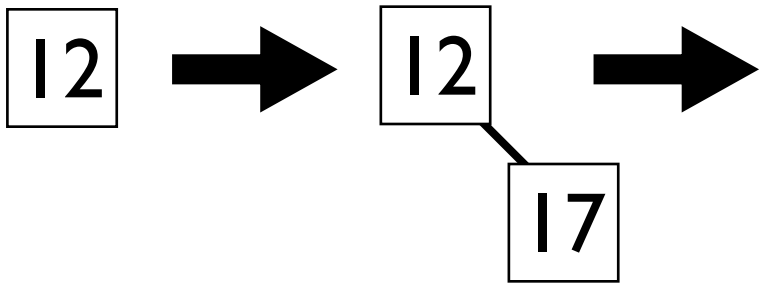
Suchen
AVL-Bäume - Schlußbeispiel

- Aufbau eines AVL-Baums mit der Zahlenfolge 12, 17, 13, 6, 10, 9, 5, 4, 15



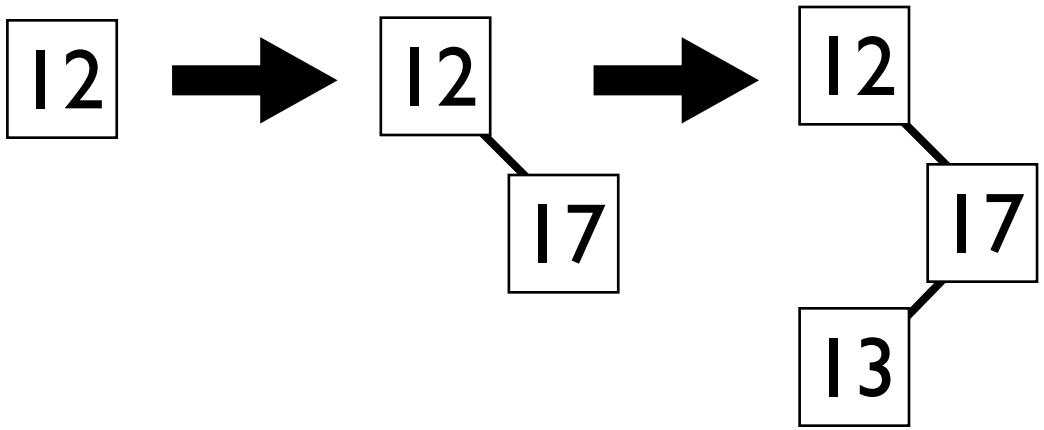
Suchen
AVL-Bäume - Schlußbeispiel

- Aufbau eines AVL-Baums mit der Zahlenfolge 12, 17, 13, 6, 10, 9, 5, 4, 15



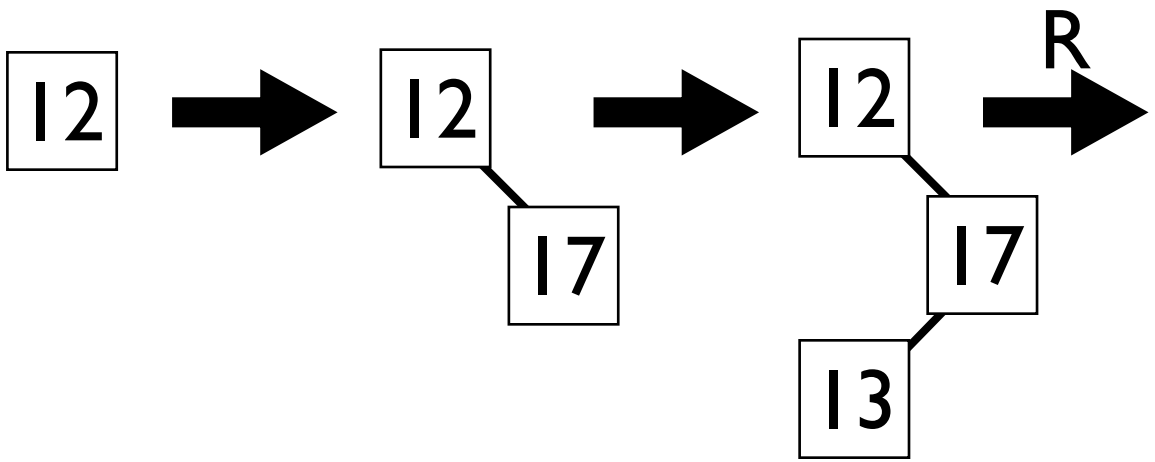
Suchen
AVL-Bäume - Schlußbeispiel

- Aufbau eines AVL-Baums mit der Zahlenfolge 12, 17, 13, 6, 10, 9, 5, 4, 15



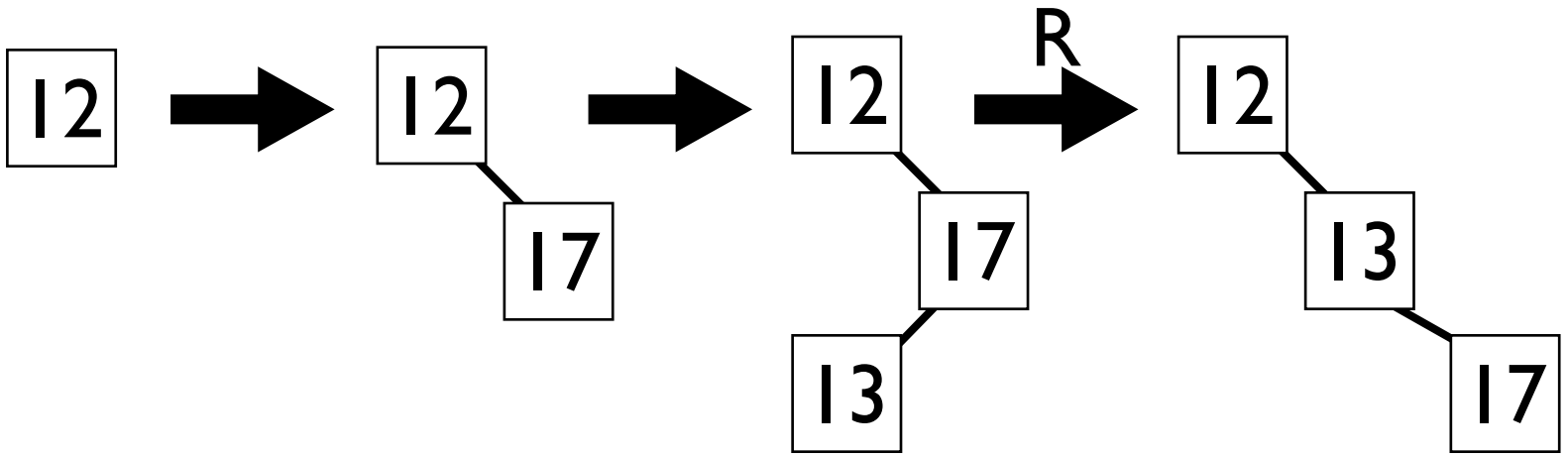
Suchen
AVL-Bäume - Schlußbeispiel

- Aufbau eines AVL-Baums mit der Zahlenfolge 12, 17, 13, 6, 10, 9, 5, 4, 15



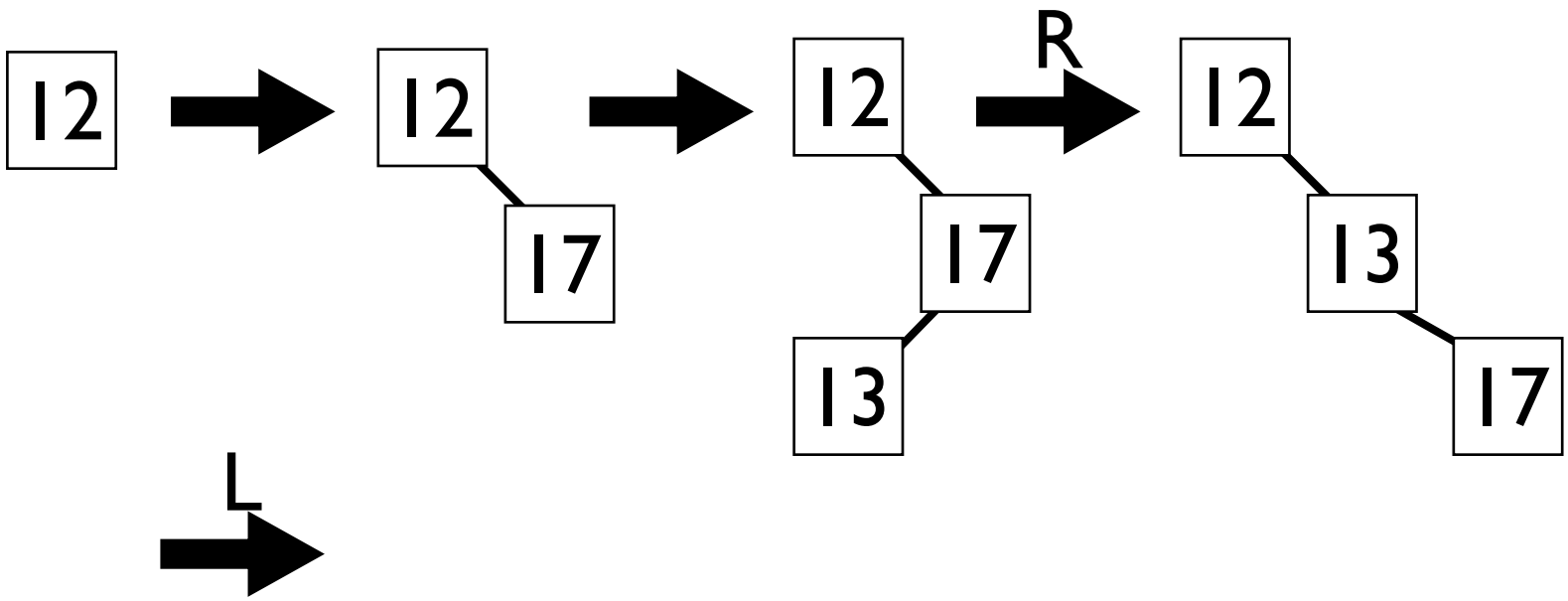
Suchen
AVL-Bäume - Schlußbeispiel

- Aufbau eines AVL-Baums mit der Zahlenfolge 12, 17, 13, 6, 10, 9, 5, 4, 15



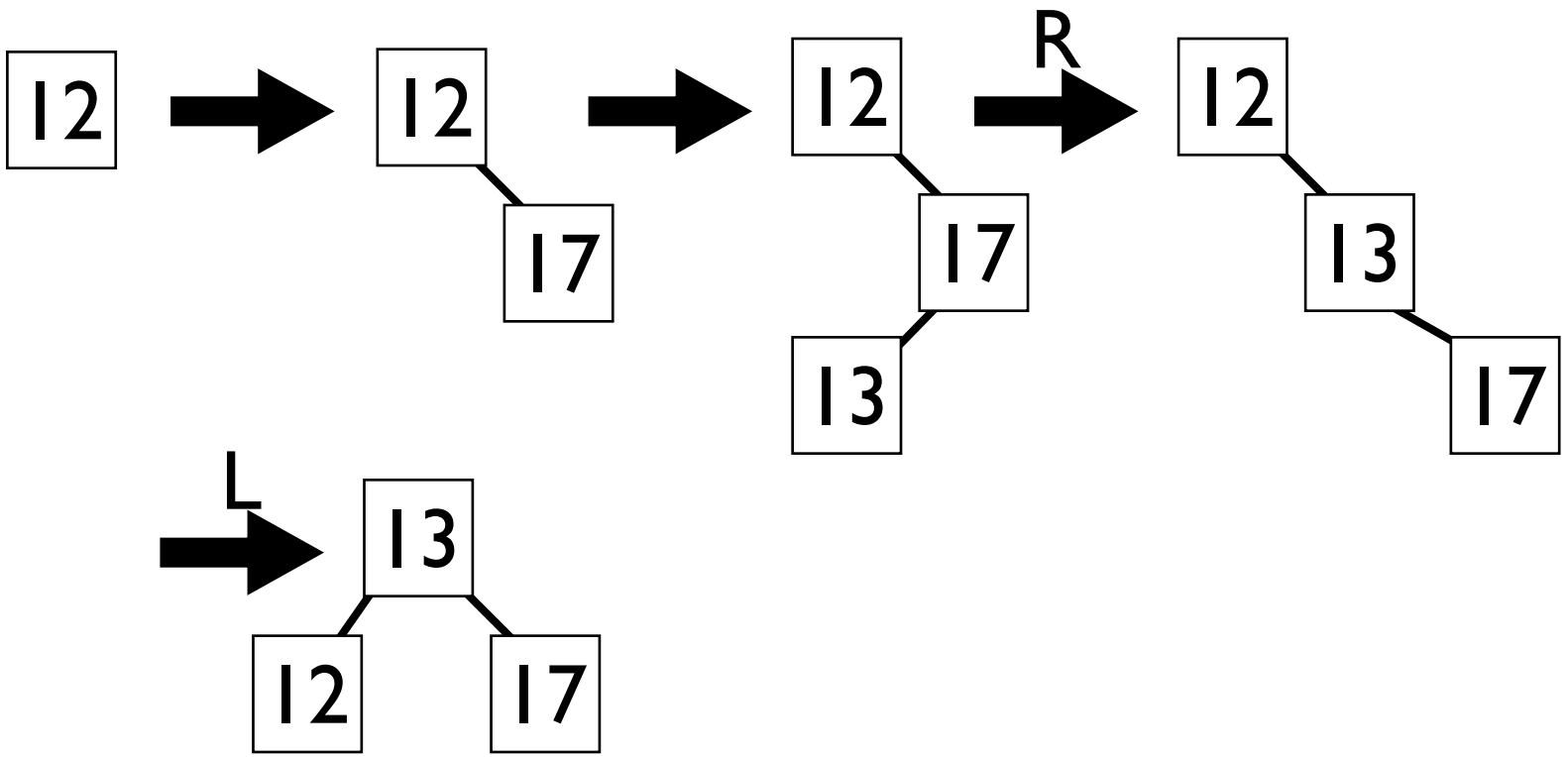
Suchen
AVL-Bäume - Schlußbeispiel

- Aufbau eines AVL-Baums mit der Zahlenfolge 12, 17, 13, 6, 10, 9, 5, 4, 15



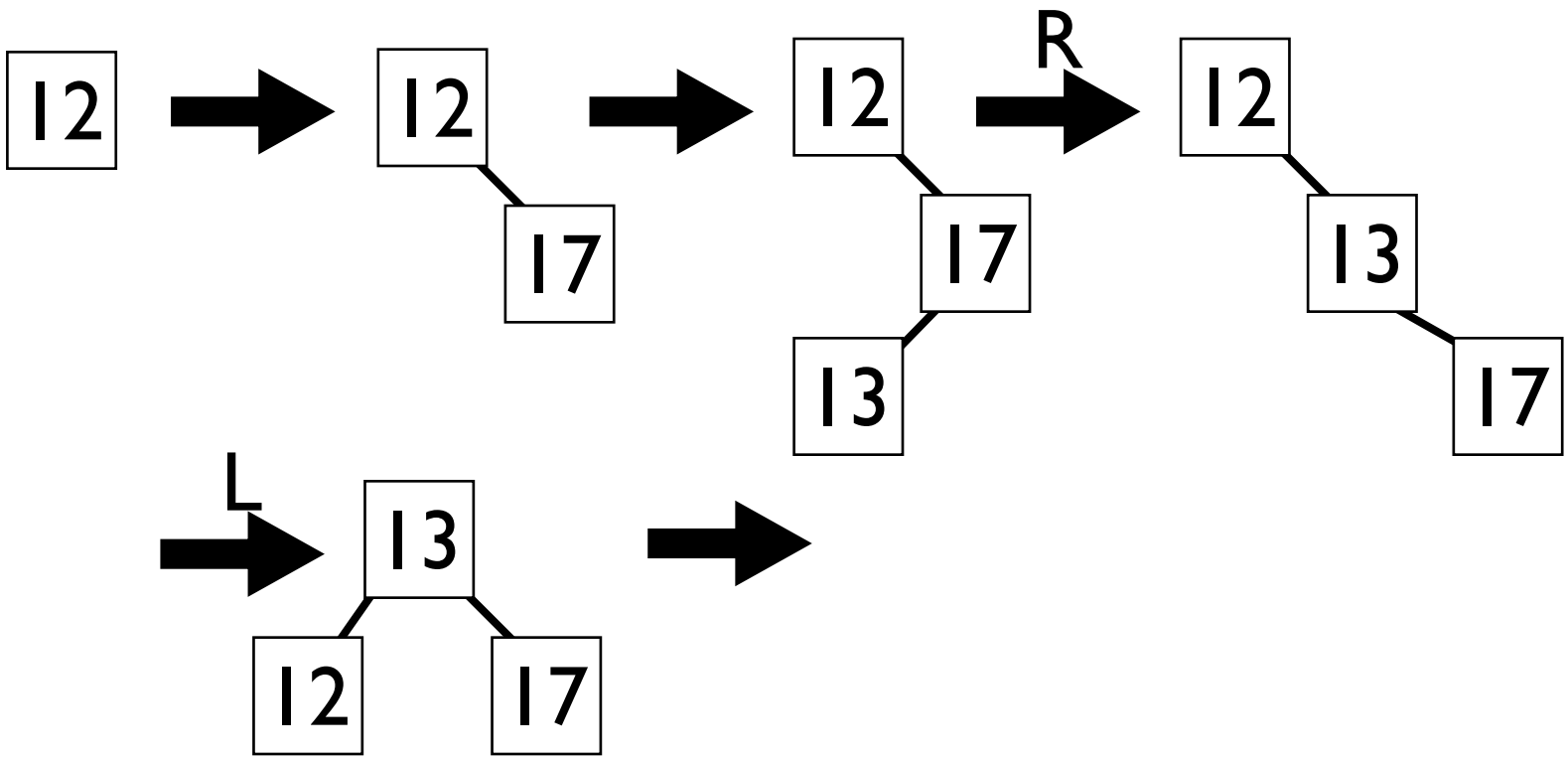
Suchen
AVL-Bäume - Schlußbeispiel

- Aufbau eines AVL-Baums mit der Zahlenfolge 12, 17, 13, 6, 10, 9, 5, 4, 15



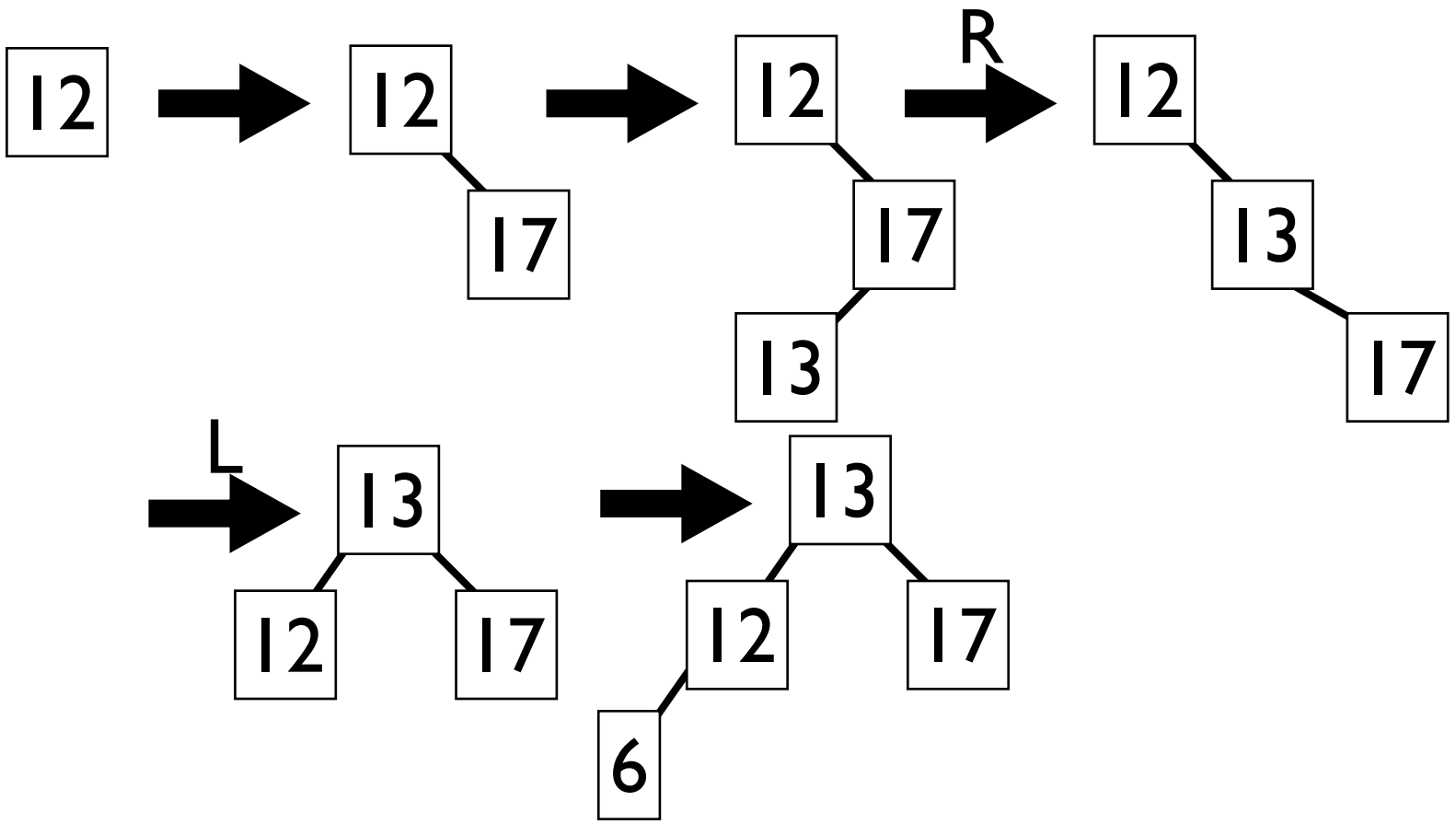
Suchen
AVL-Bäume - Schlußbeispiel

- Aufbau eines AVL-Baums mit der Zahlenfolge 12, 17, 13, 6, 10, 9, 5, 4, 15



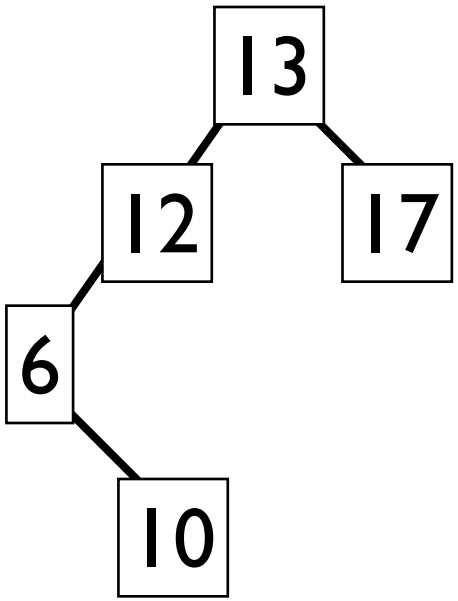
Suchen
AVL-Bäume - Schlußbeispiel

- Aufbau eines AVL-Baums mit der Zahlenfolge 12, 17, 13, 6, 10, 9, 5, 4, 15



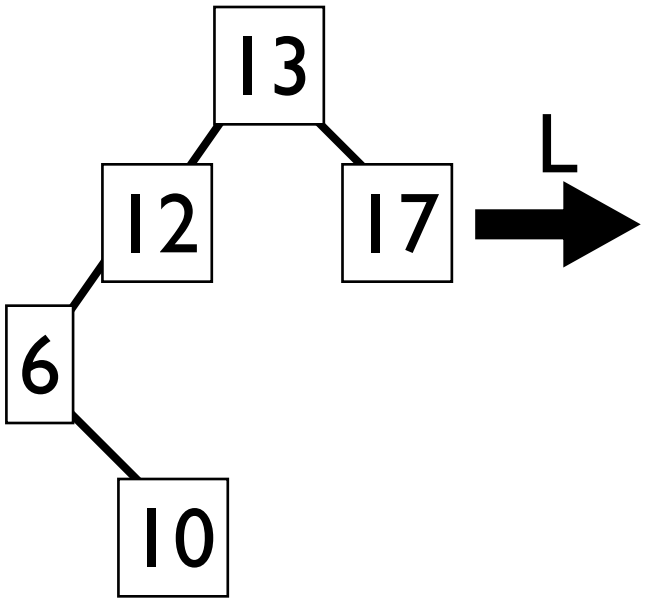
Suchen

AVL-Bäume - Schlußbeispiel



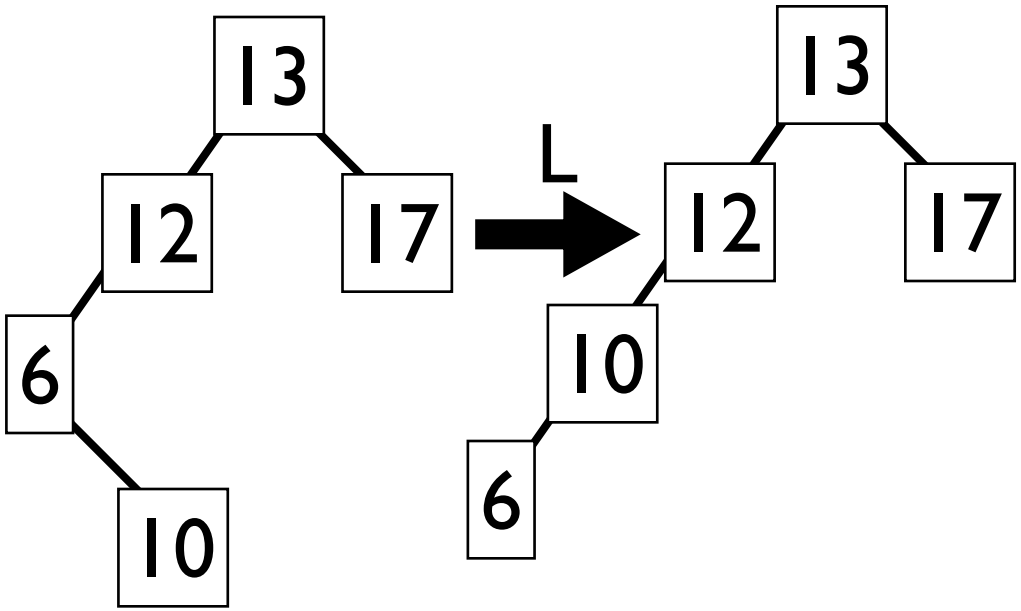
Suchen

AVL-Bäume - Schlußbeispiel



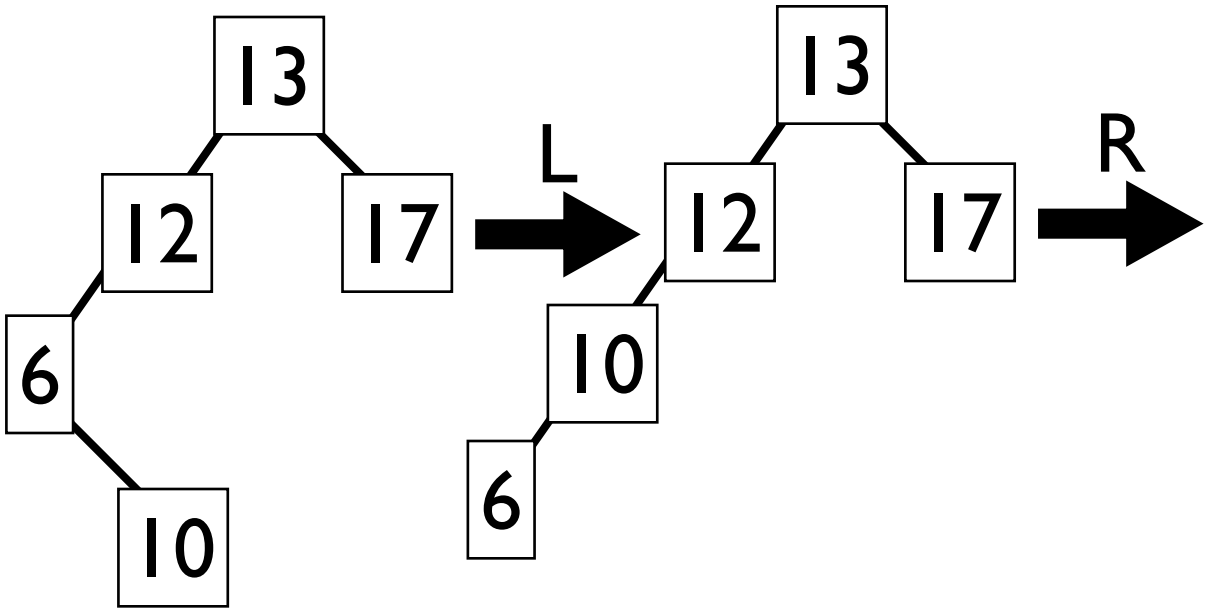
Suchen

AVL-Bäume - Schlußbeispiel



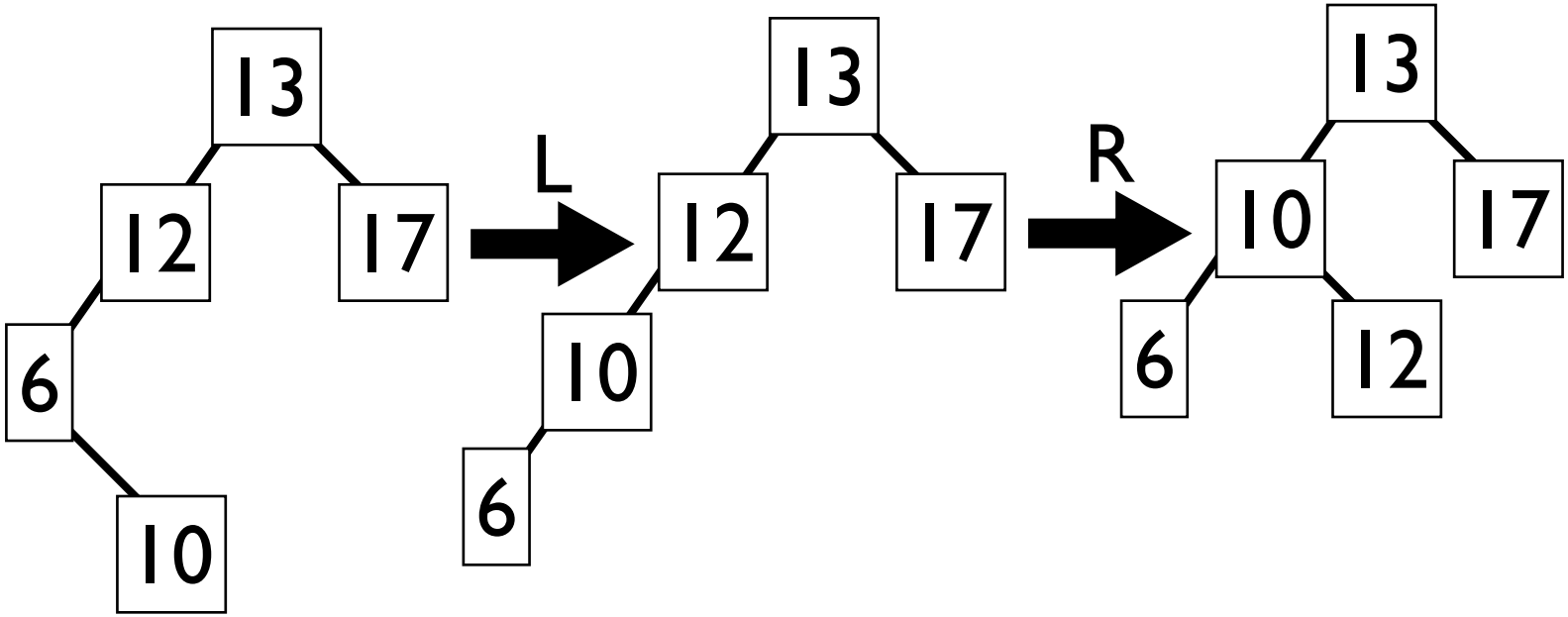
Suchen

AVL-Bäume - Schlußbeispiel



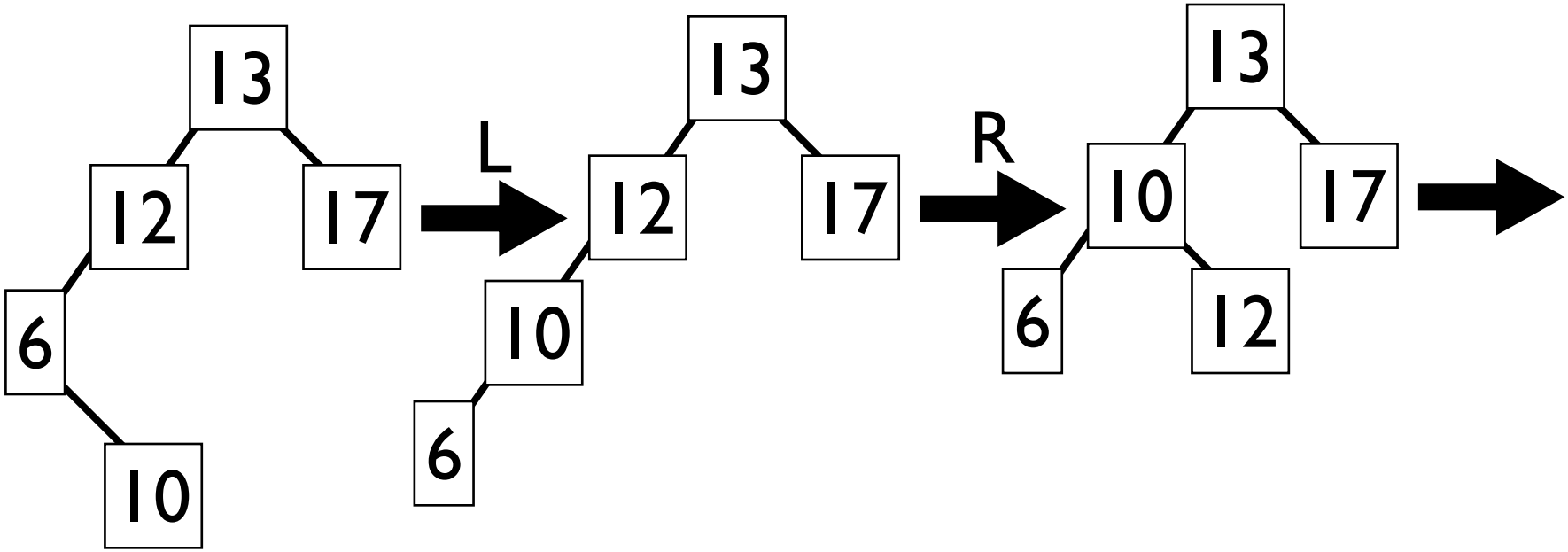
Suchen

AVL-Bäume - Schlußbeispiel



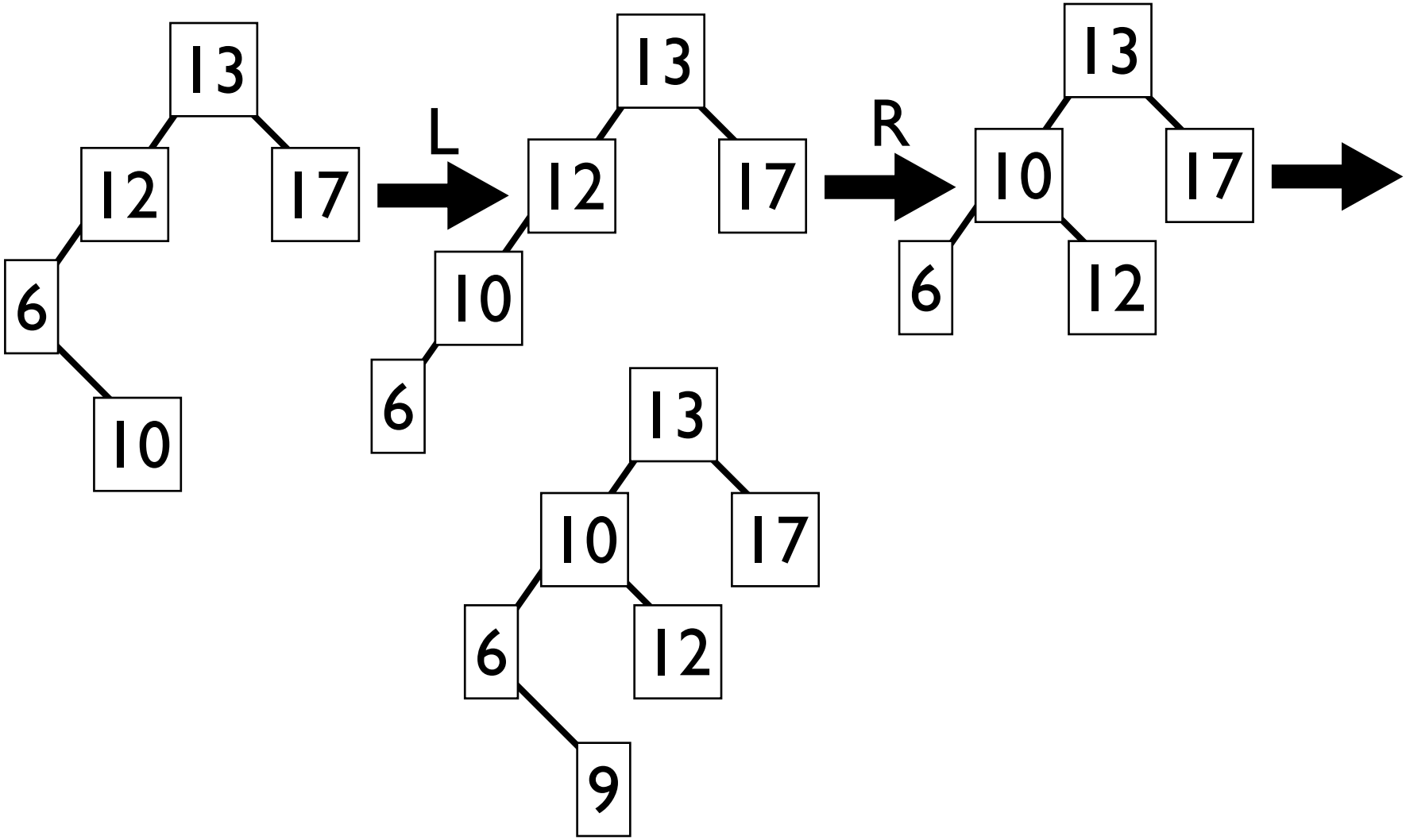
Suchen

AVL-Bäume - Schlußbeispiel



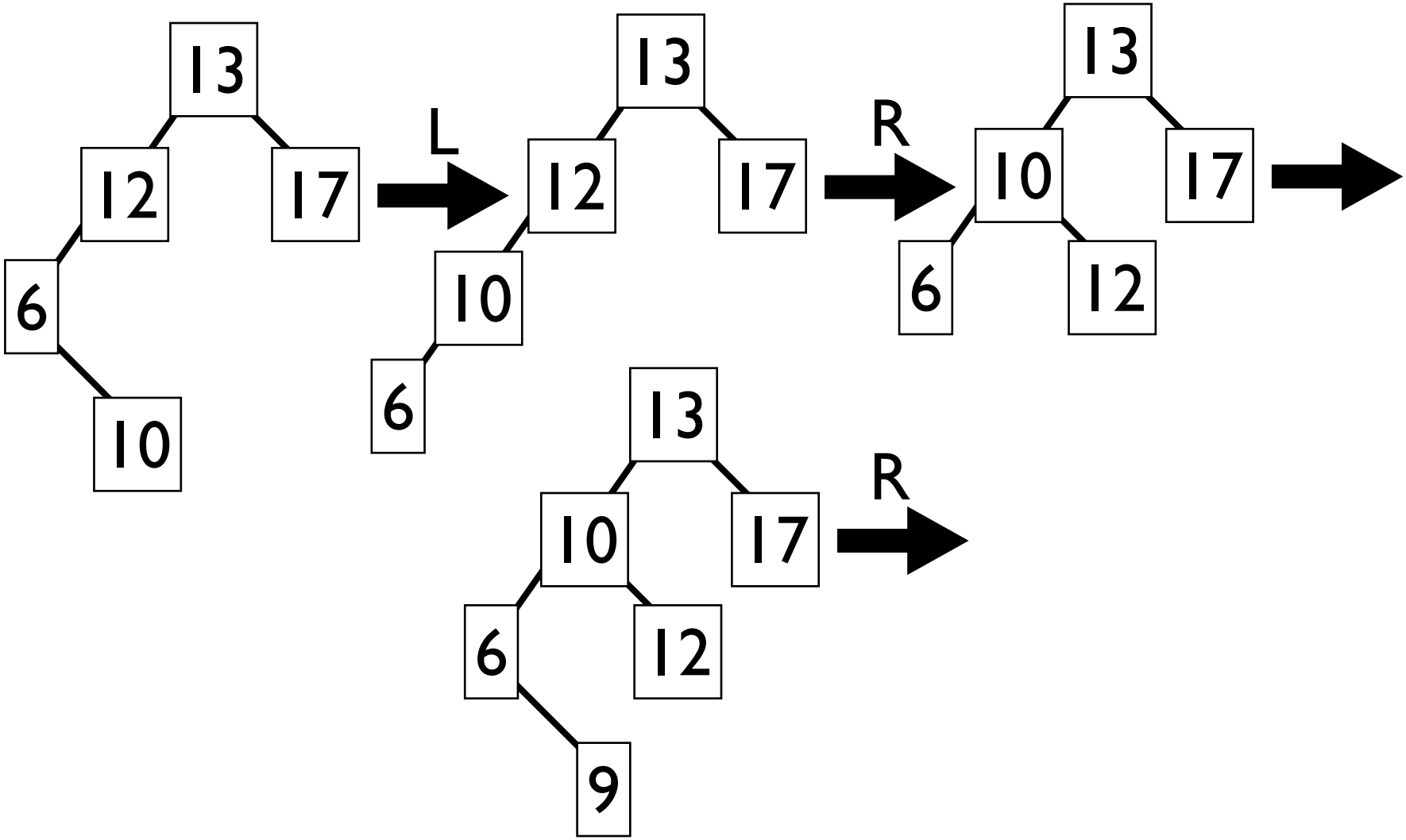
Suchen

AVL-Bäume - Schlußbeispiel



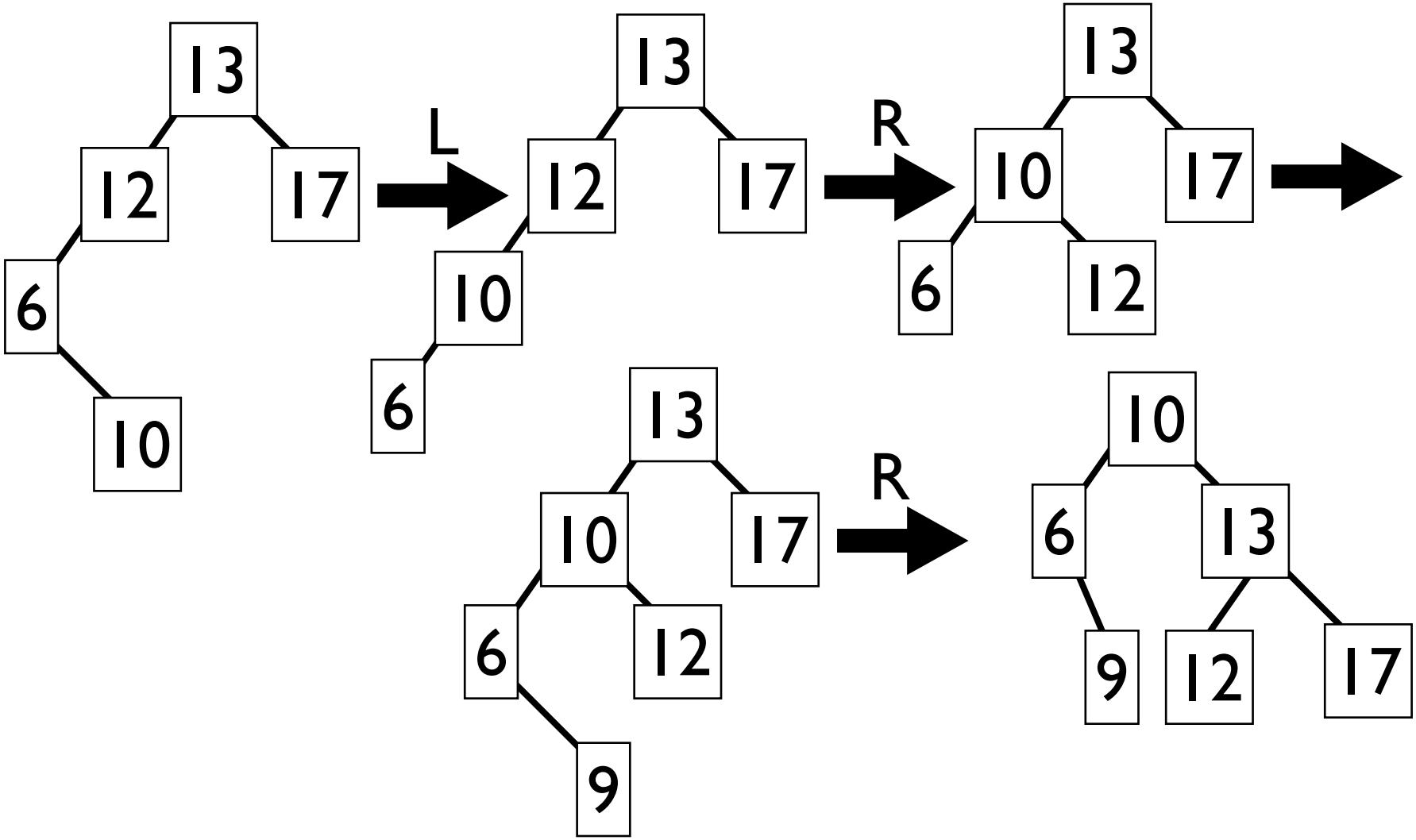
Suchen

AVL-Bäume - Schlußbeispiel



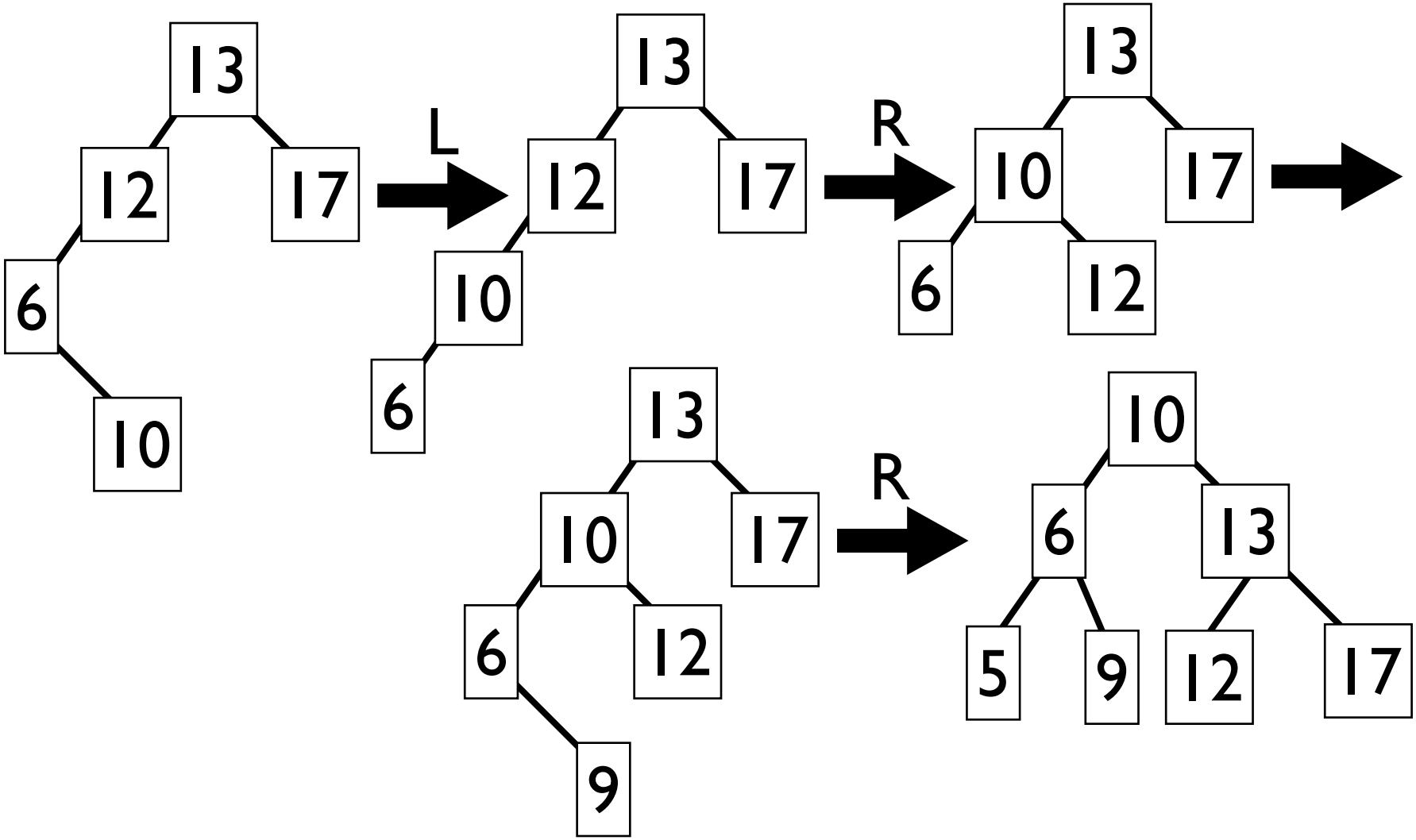
Suchen

AVL-Bäume - Schlußbeispiel



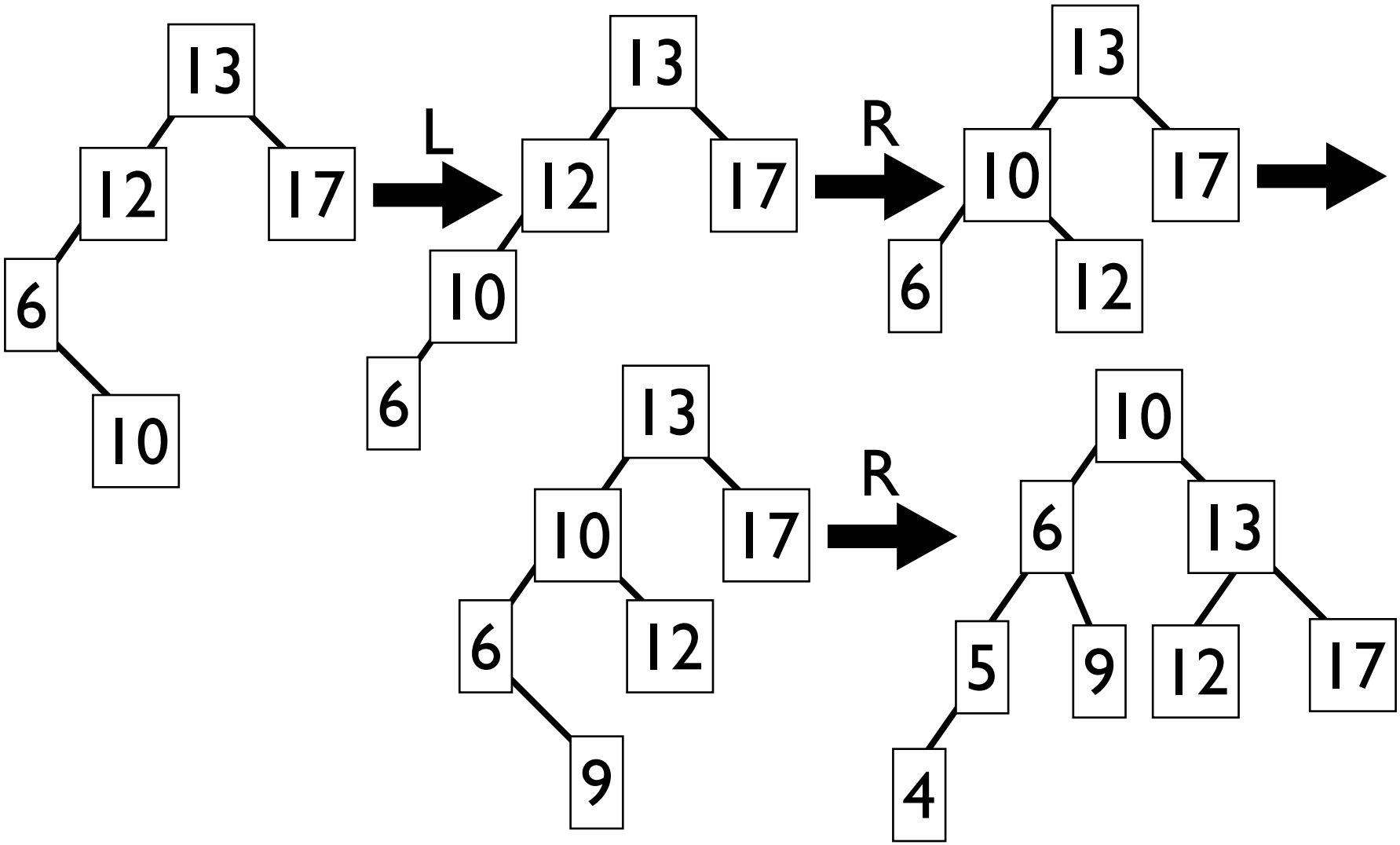
Suchen

AVL-Bäume - Schlußbeispiel



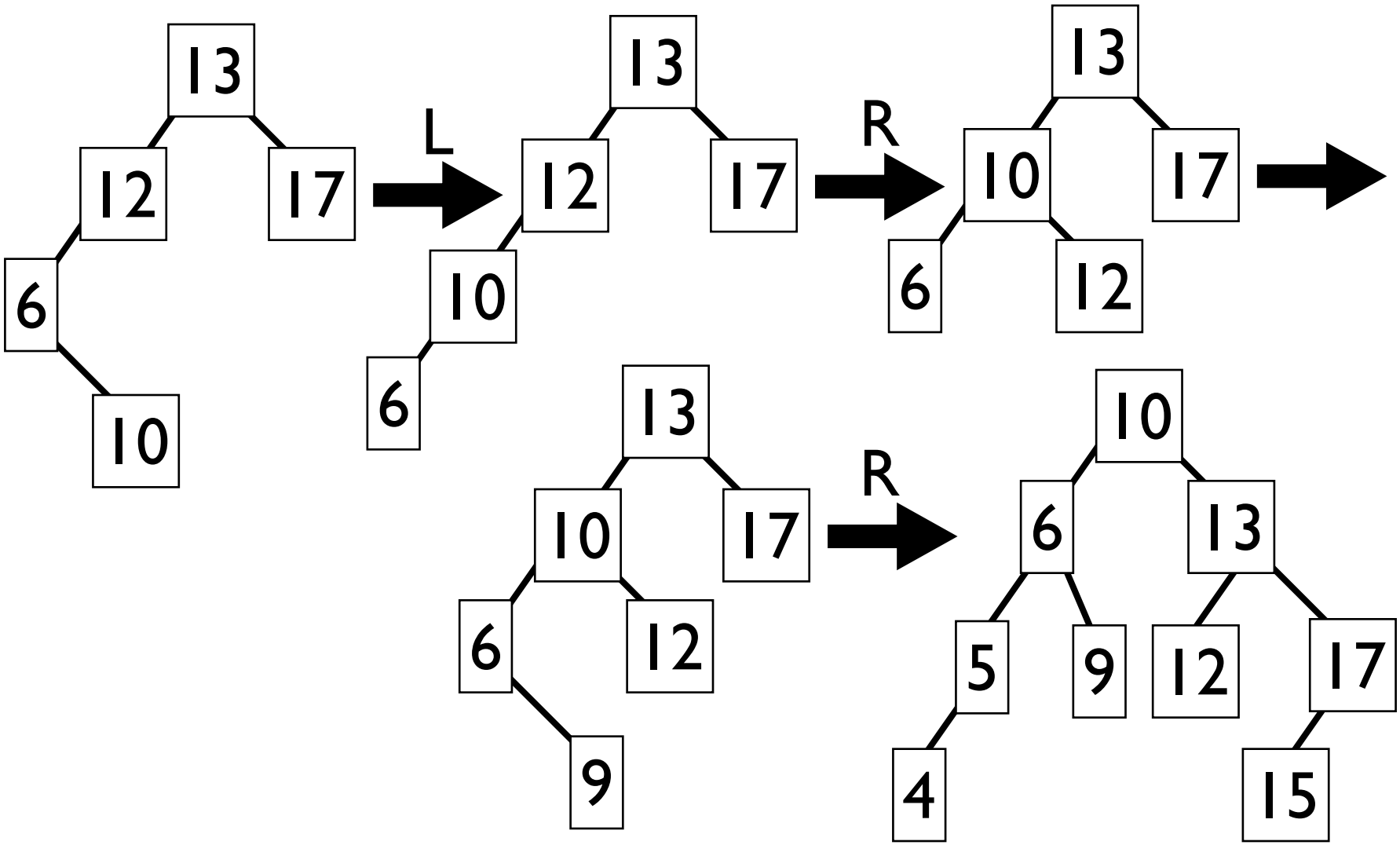
Suchen

AVL-Bäume - Schlußbeispiel



Suchen

AVL-Bäume - Schlußbeispiel



Suchen
B-Bäume - Motivation

- Bisher nur Suchalgorithmen für Hauptspeicher
- Nun Suche auf externen Speichern (Platten)
- Baumstrukturen → Plattenspeicher
  (Hauptspeicheradressen → Plattenspeicheradressen)
- Problem: Zugriffszeit auf einen Knoten über einen Zeiger ist nicht mehr vernachlässigbar (nicht mehr nur eine Zeiteinheit)

Suchen
B-Bäume - Motivation

- Konsequenzen:
  - Reduktion der Plattenzugriffe auf Mindestmaß und Nutzung der geblockten Übertragung und Abspeicherung
  - mit jedem Plattenzugriff Übertragung eines vollständigen Teilbaums
- Idee: Zusammenfassung von Teilbäumen zu einem Superknoten und Realisierung einer Balanciertheit

Suchen
B-Bäume - Definition

Definition

Ein Baum heißt **B-Baum der Ordnung m**, falls gilt:

a) Jeder Knoten enthält $\leq 2m$ Schlüssel.

b) Jeder Knoten (mit Ausnahme der Wurzel) enthält $\geq m$ Schlüssel.

c) Ein Knoten mit k Schlüsseln hat genau k+1 Söhne oder keinen Sohn.

d) Alle Knoten, die keine Söhne haben, befinden sich auf gleichem Niveau.

e) *Suchbaumeigenschaft*: ...

Suchen
B-Bäume - Definition

Definition (Forts.)

e) *Suchbaumeigenschaft*: Sind $s_1,..., s_k$ mit $m \leq k \leq 2m$ die Schlüssel eines Knotens x, dann gilt:

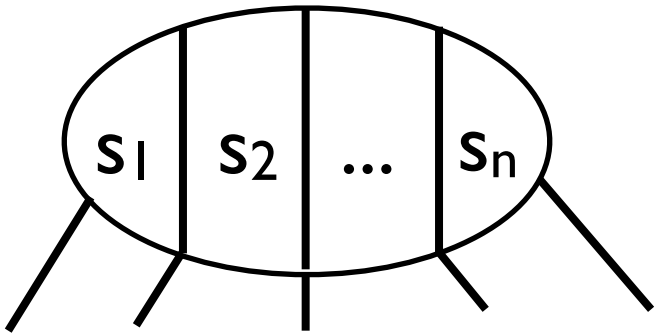
- alle Schlüssel des ersten Sohnes von x sind $\leq s_1$,
- alle Schlüssel des (k+1)-ten Sohnes sind $> s_k$ und
- alle Schlüssel des i-ten Sohnes, $1<i<k+1$, sind $> s_{i-1}$ und $\leq s_i$.

Suchen
B-Bäume - Definition

Definition (Forts.)

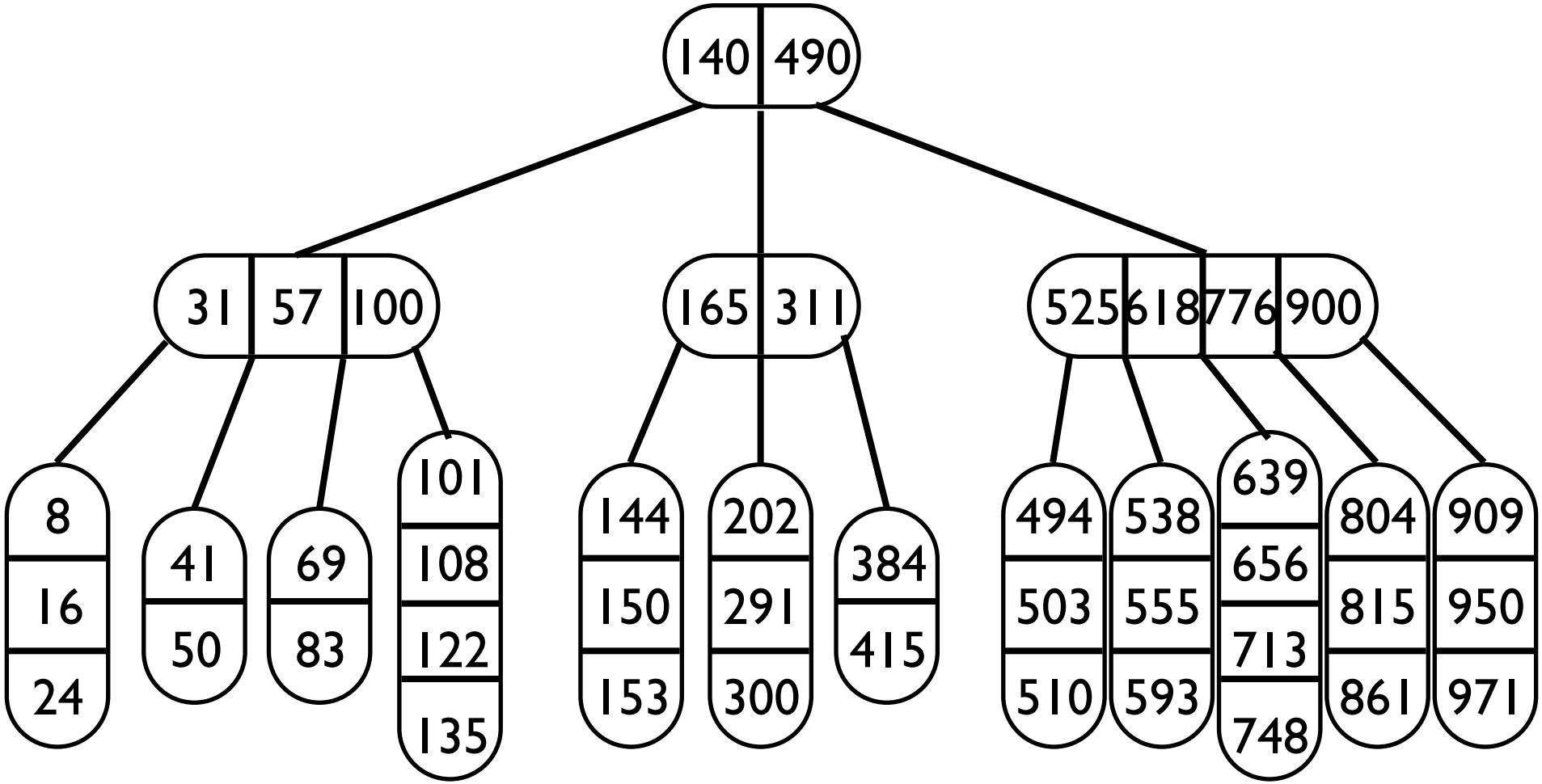
e) *Suchbaumeigenschaft*: Sind $s_1,..., s_k$ mit $m \leq k \leq 2m$ die Schlüssel eines Knotens x, dann gilt:

- alle Schlüssel des ersten Sohnes von x sind $\leq s_1$,
- alle Schlüssel des (k+1)-ten Sohnes sind $> s_k$ und
- alle Schlüssel des i-ten Sohnes, $1<i<k+1$, sind $> s_{i-1}$ und $\leq s_i$.



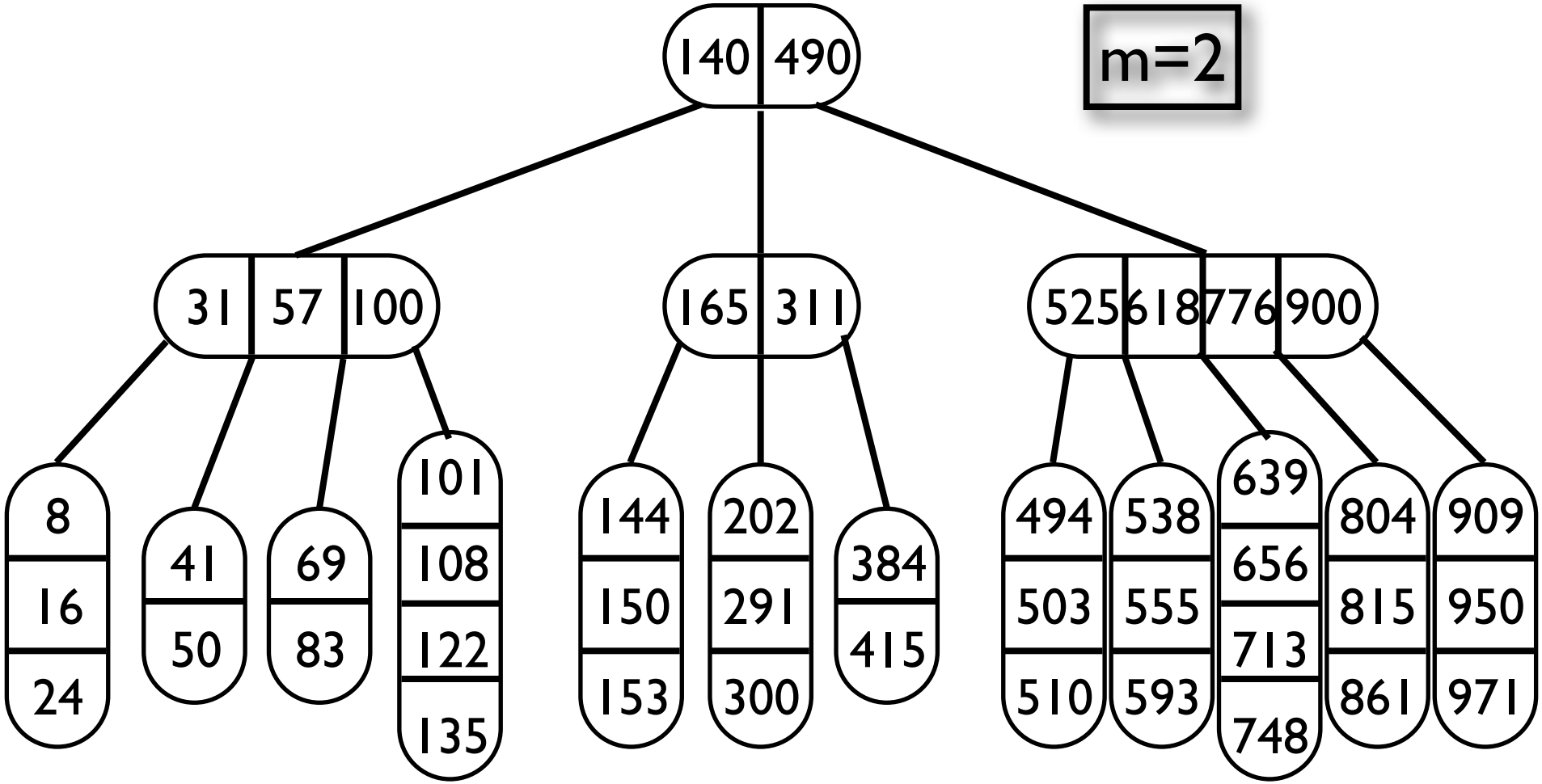
Suchen

B-Bäume - Beispiel



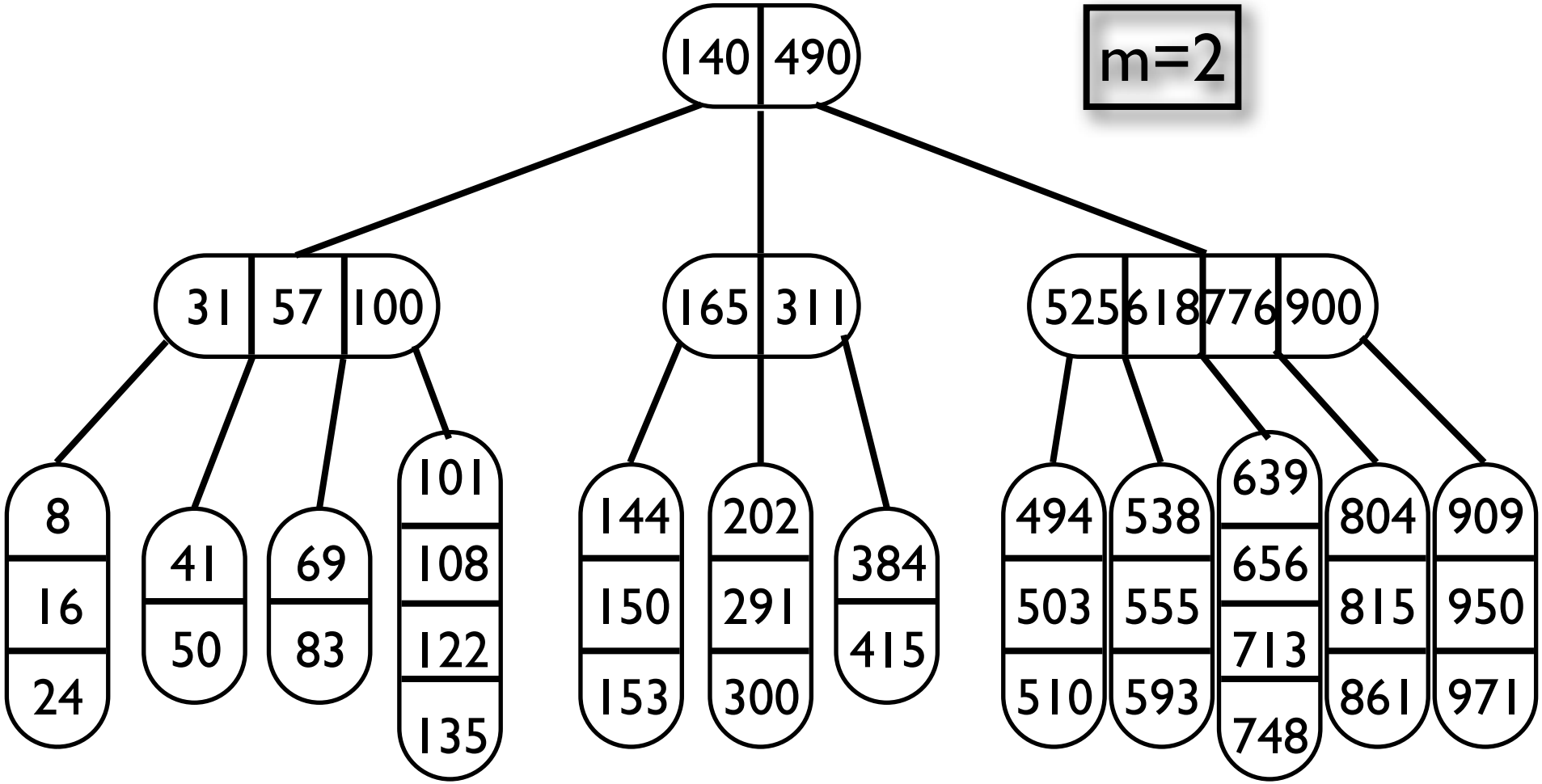
Suchen

B-Bäume - Beispiel



Suchen

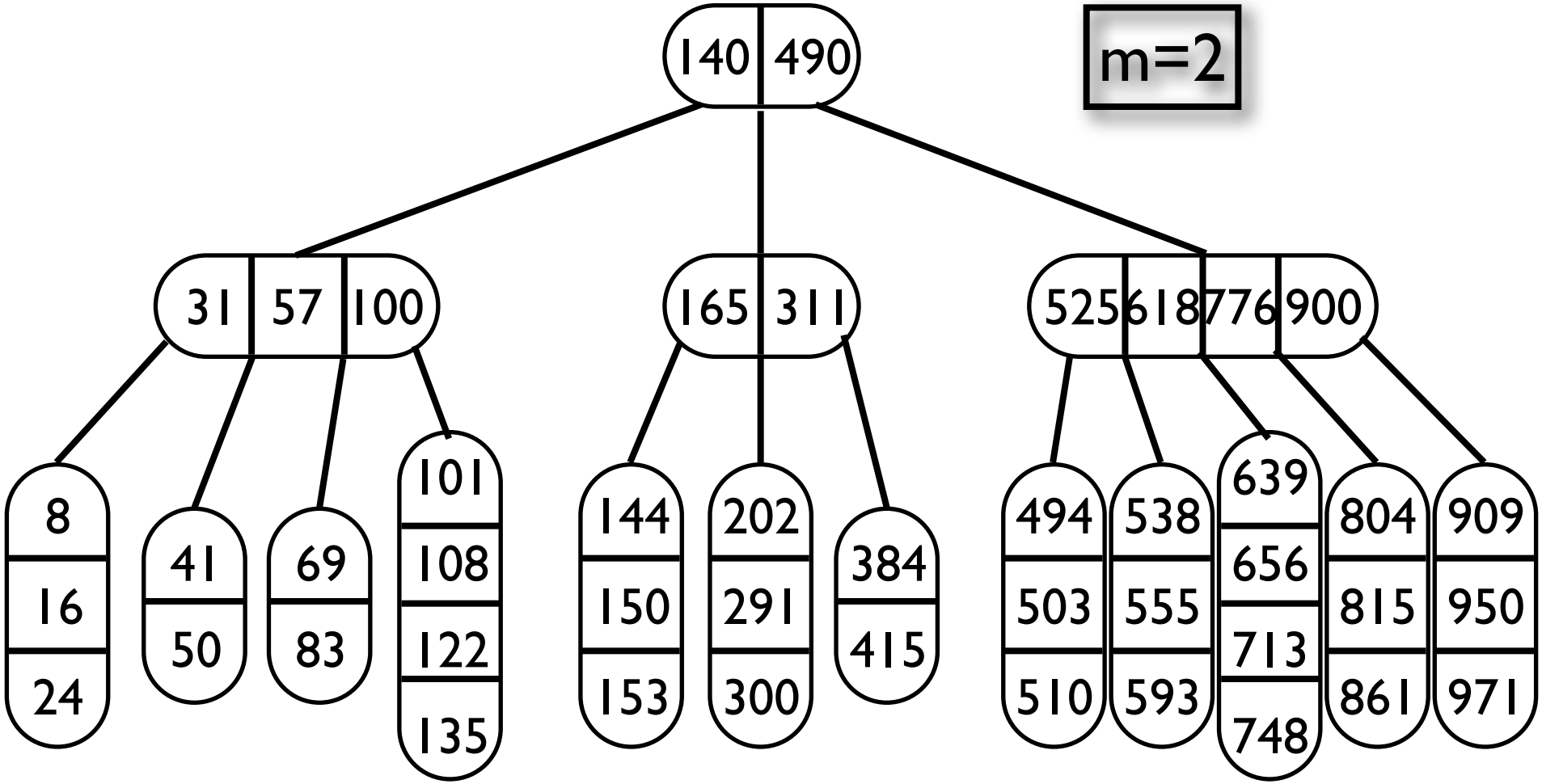
B-Bäume - Beispiel



Jeder Knoten enthält $\leq 2m$ Schlüssel.

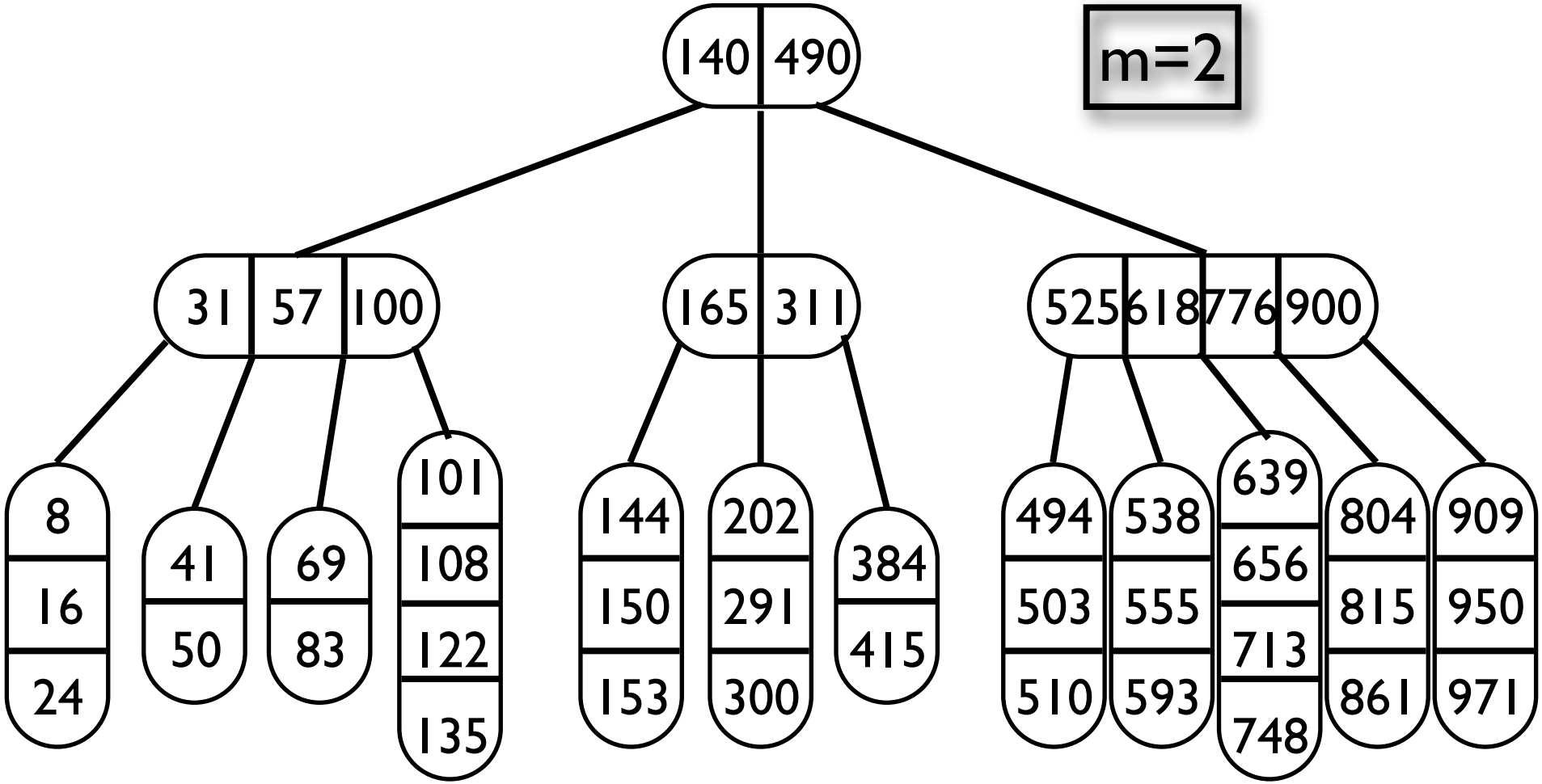
Suchen

B-Bäume - Beispiel



Suchen

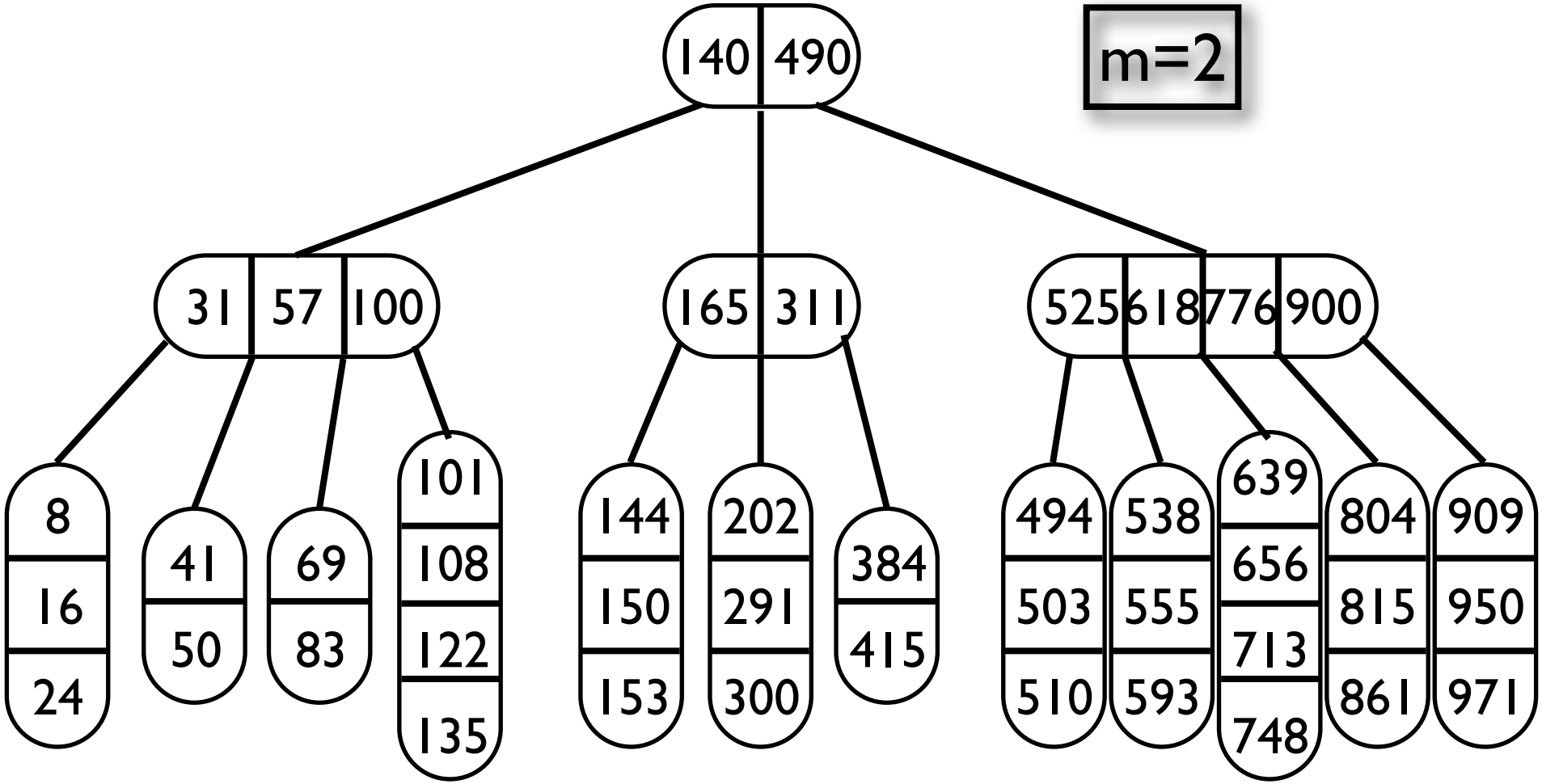
B-Bäume - Beispiel



Jeder Knoten (ohne Wurzel) enthält $\geq m$ Schlüssel.

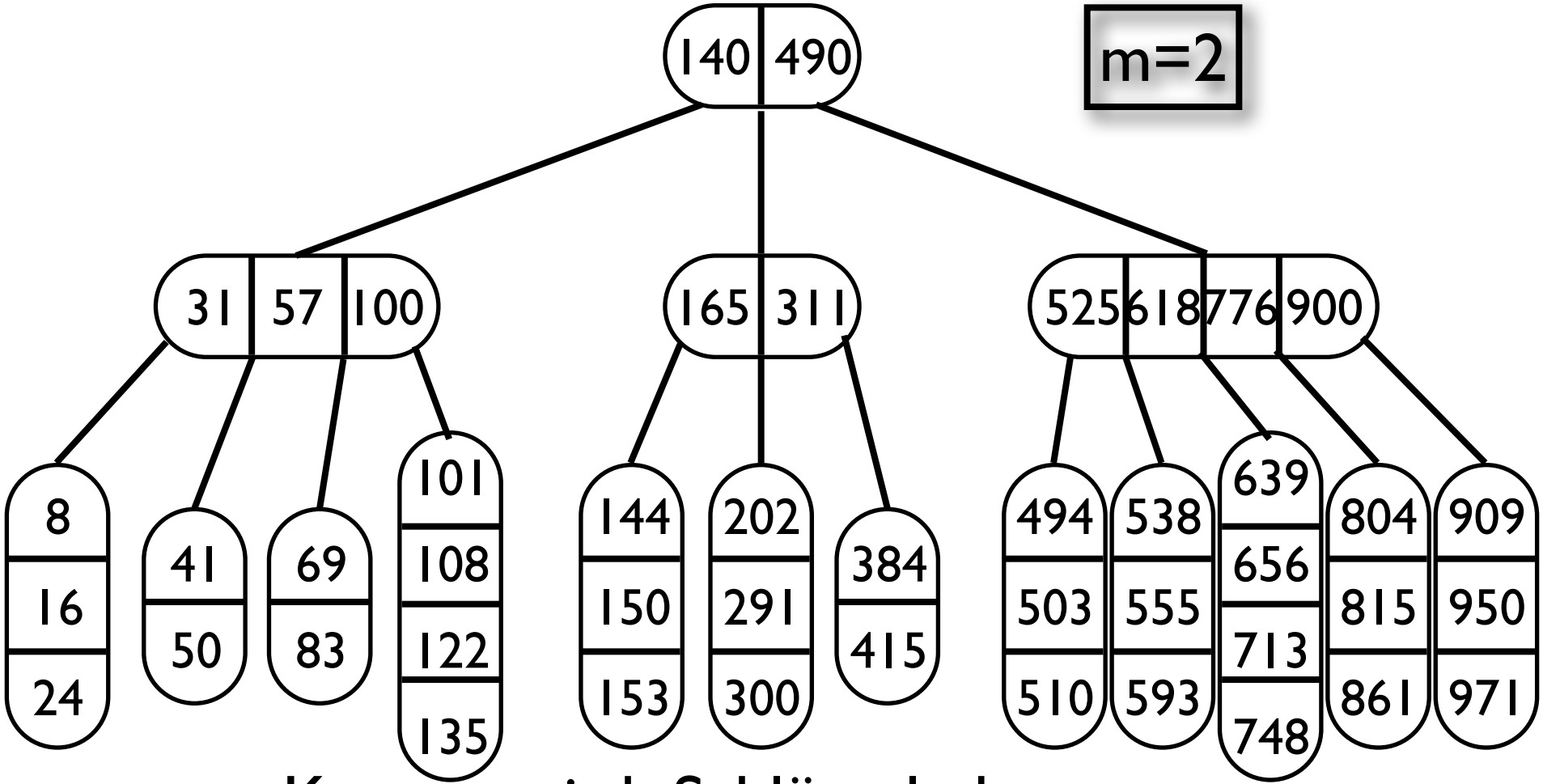
Suchen

B-Bäume - Beispiel



Suchen

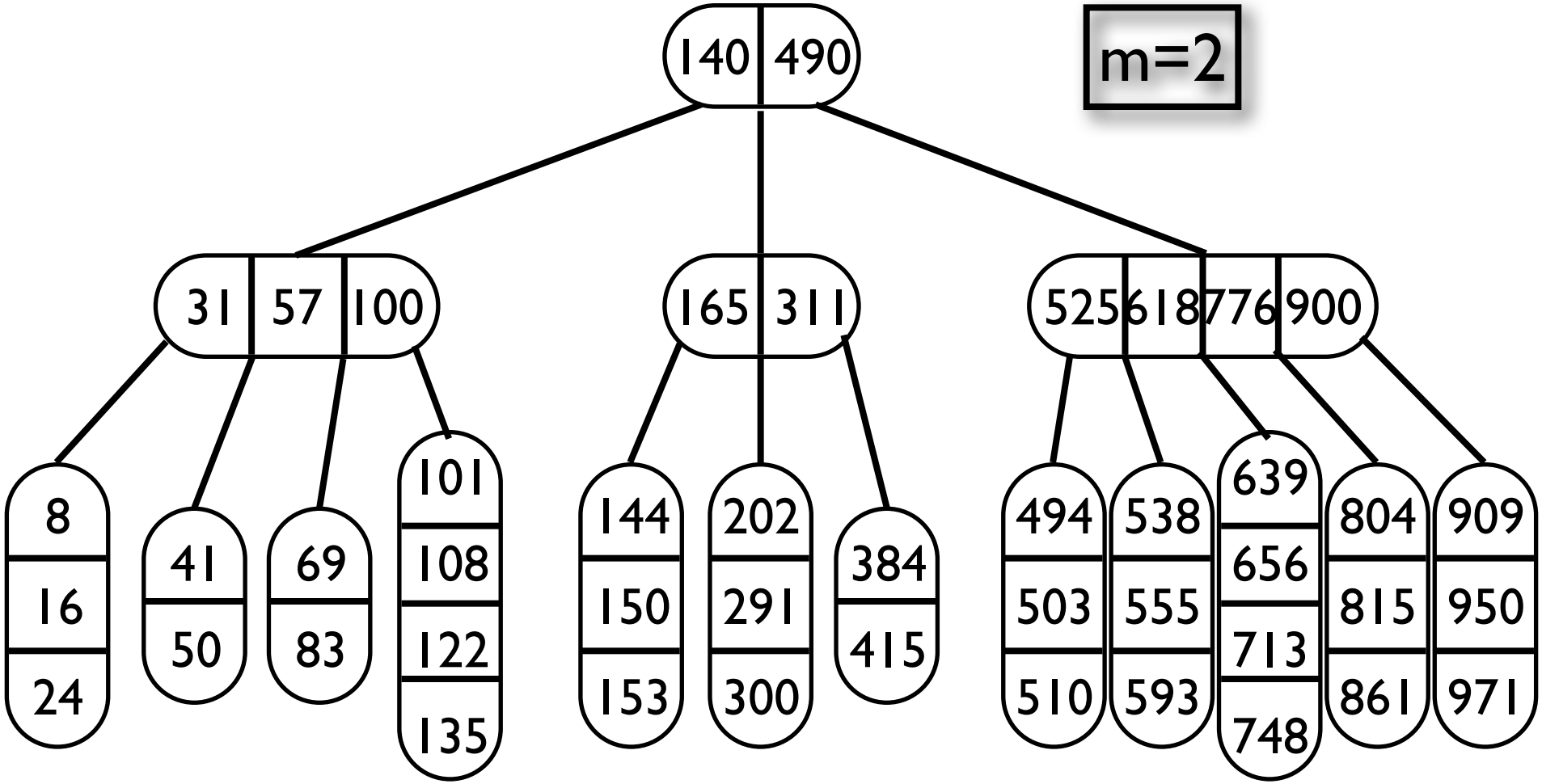
B-Bäume - Beispiel



Knoten mit k Schlüsseln hat genau
k+1 Söhne oder keinen

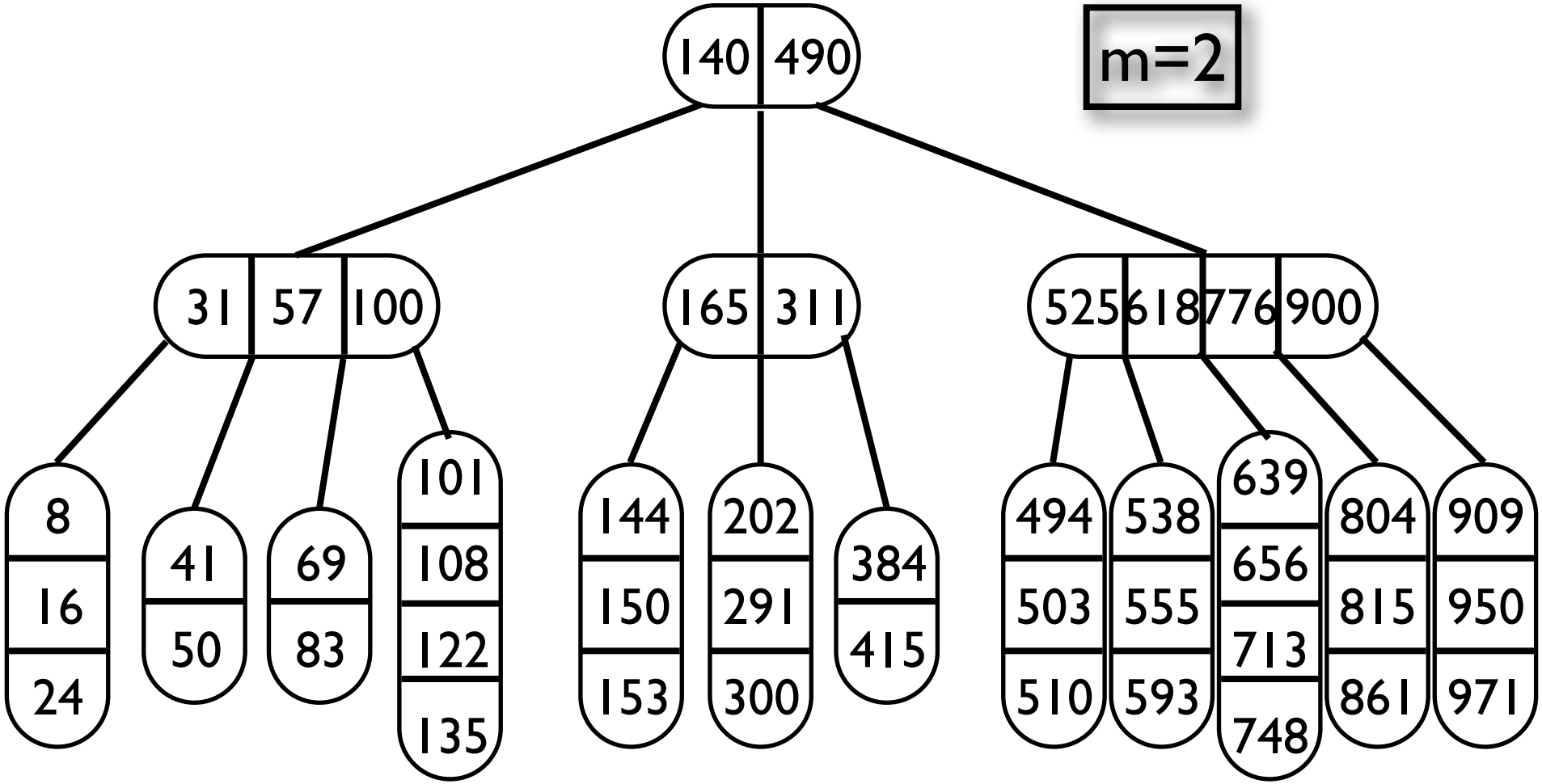
Suchen

B-Bäume - Beispiel



Suchen

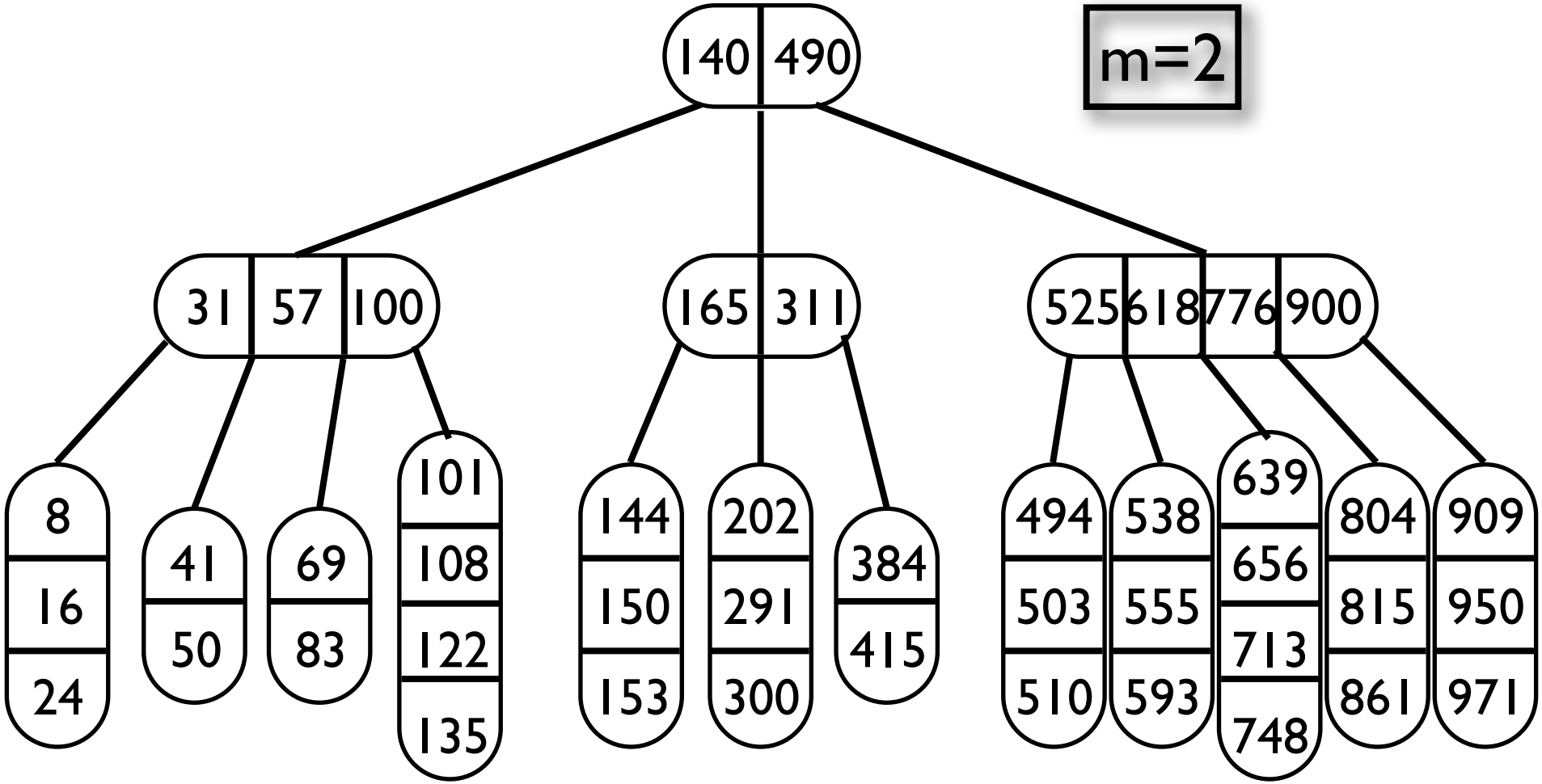
B-Bäume - Beispiel



Knoten ohne Söhne gleiches Niveau

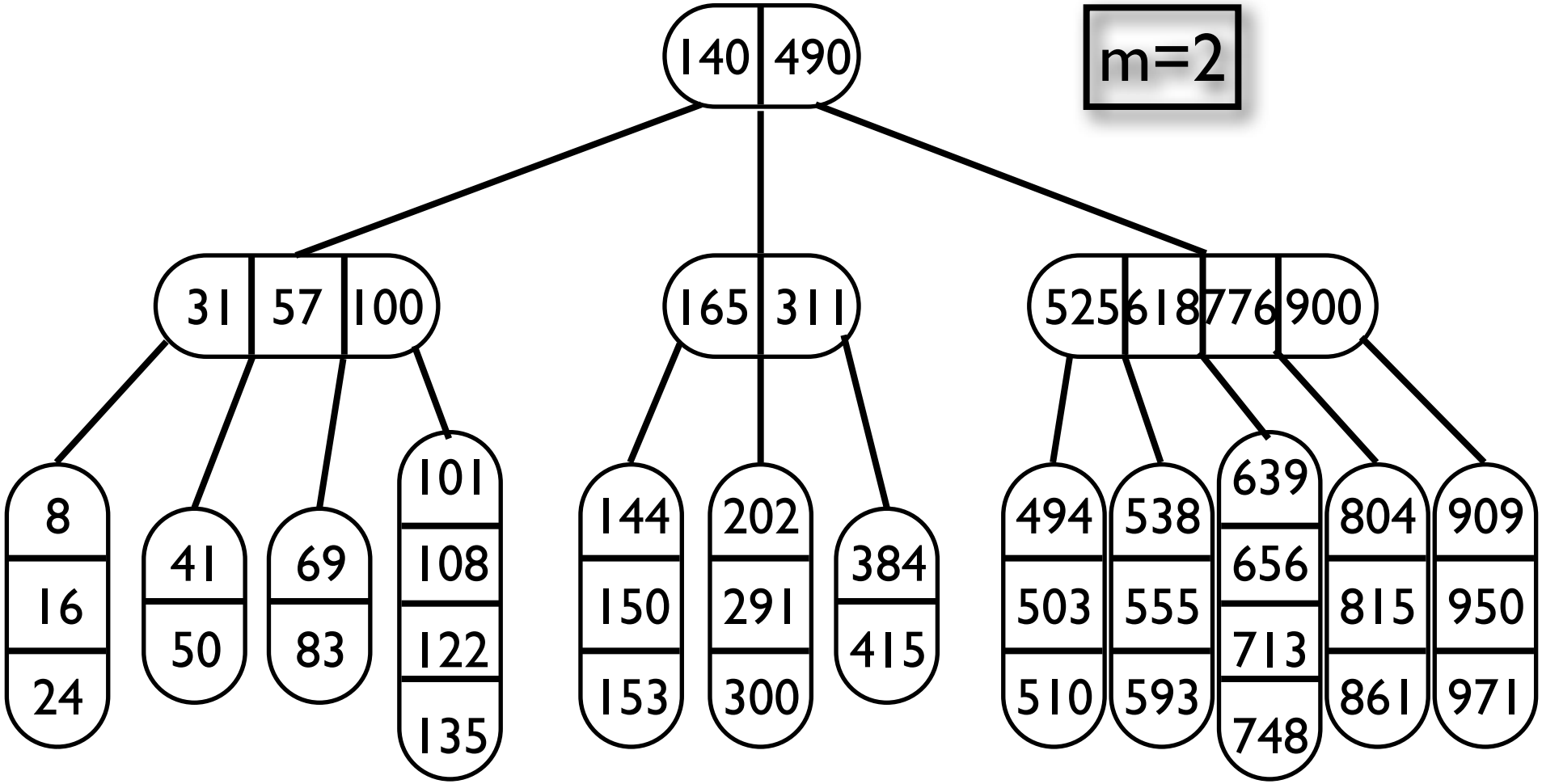
Suchen

B-Bäume - Beispiel



Suchen

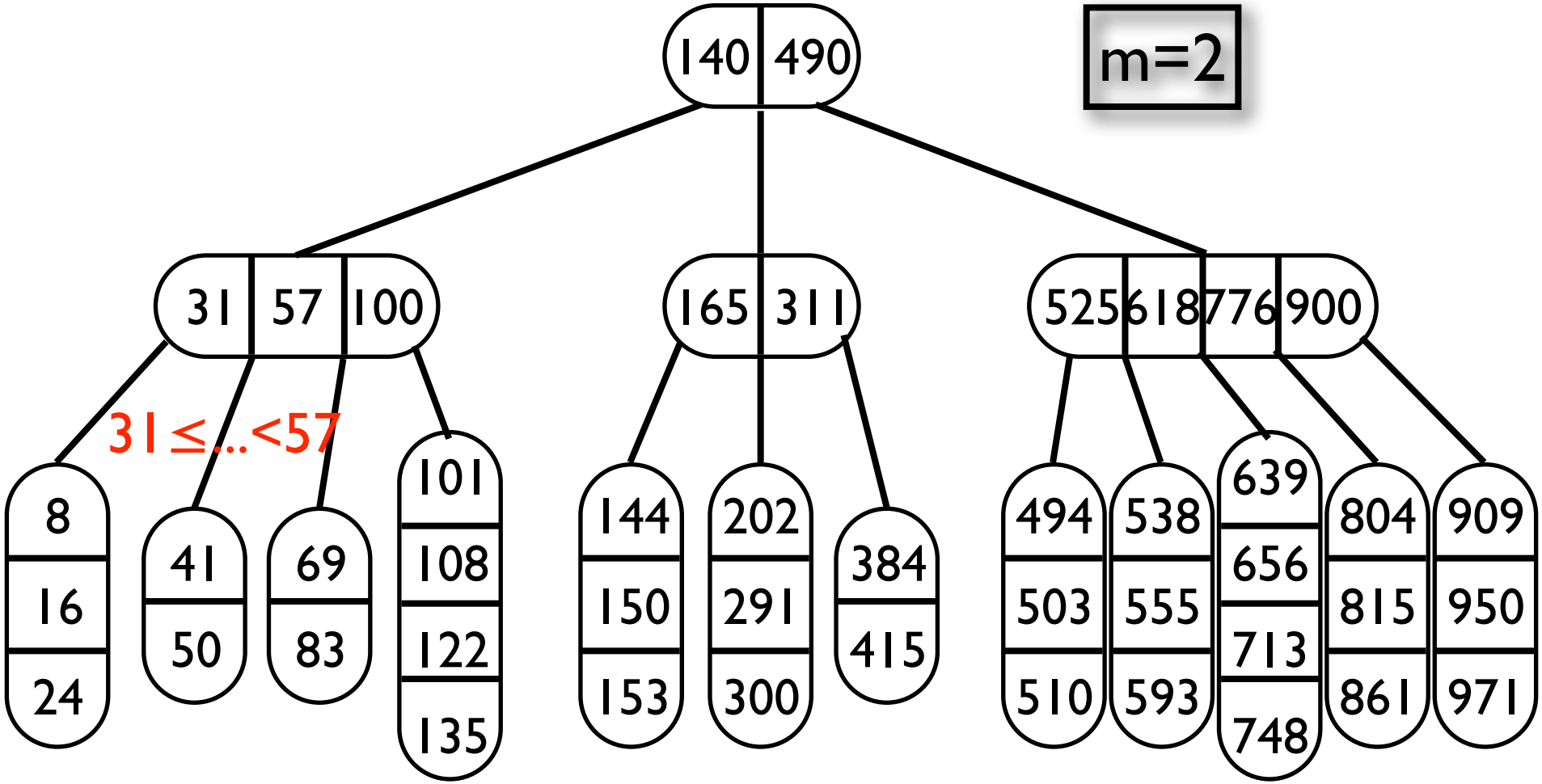
B-Bäume - Beispiel



Suchbaumeigenschaft

Suchen

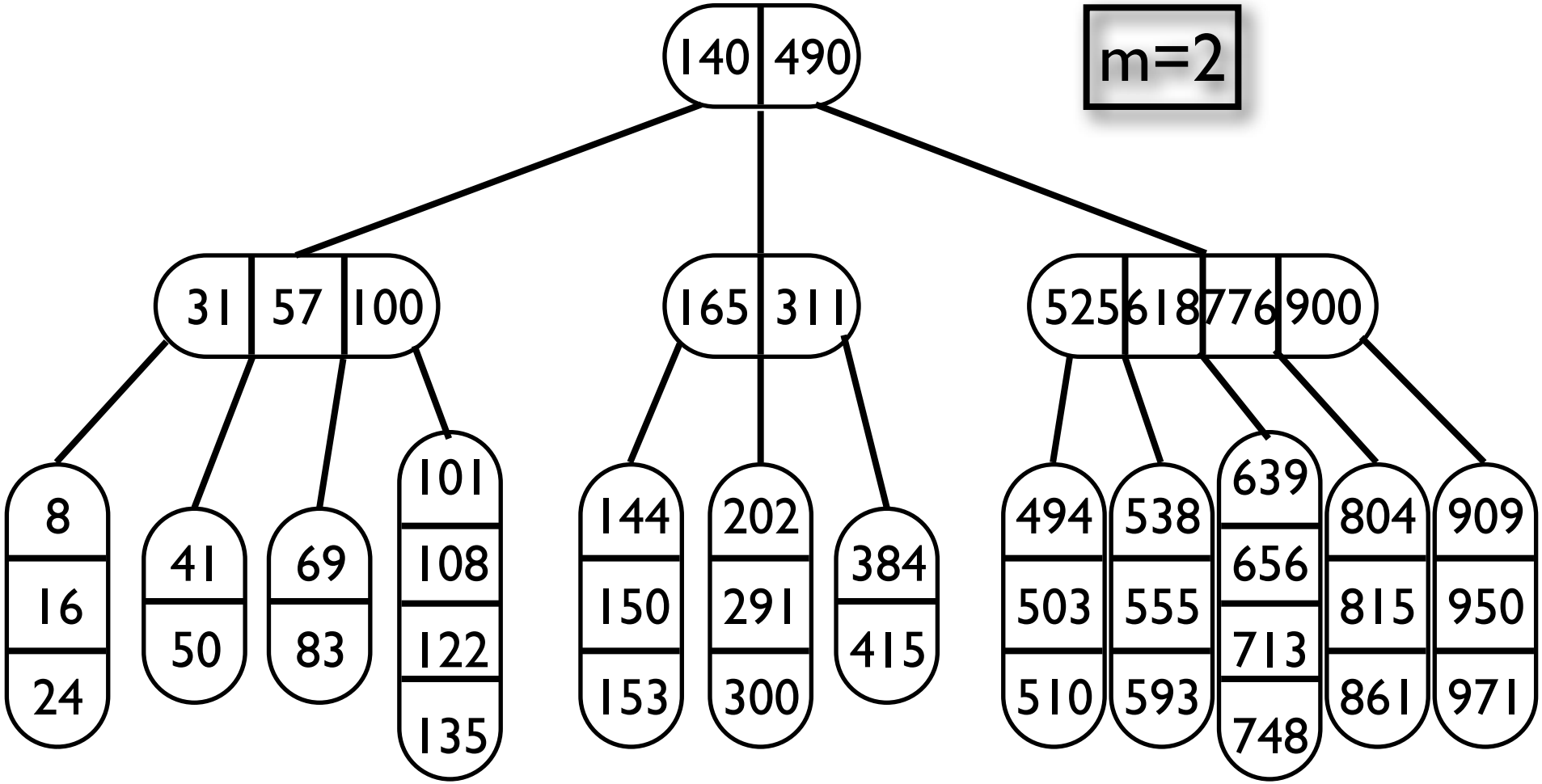
B-Bäume - Beispiel



Suchbaumeigenschaft

Suchen

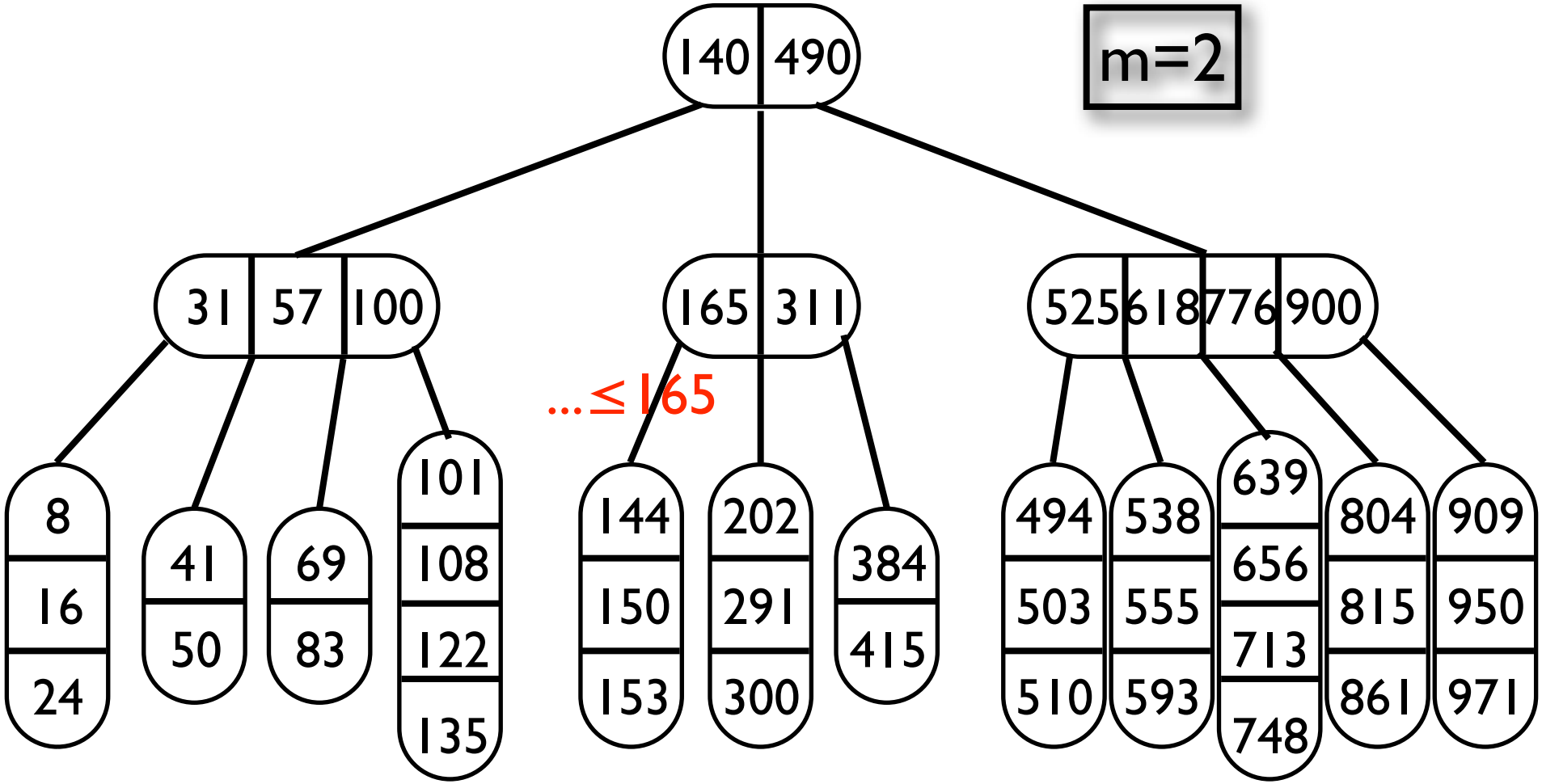
B-Bäume - Beispiel



Suchbaumeigenschaft

Suchen

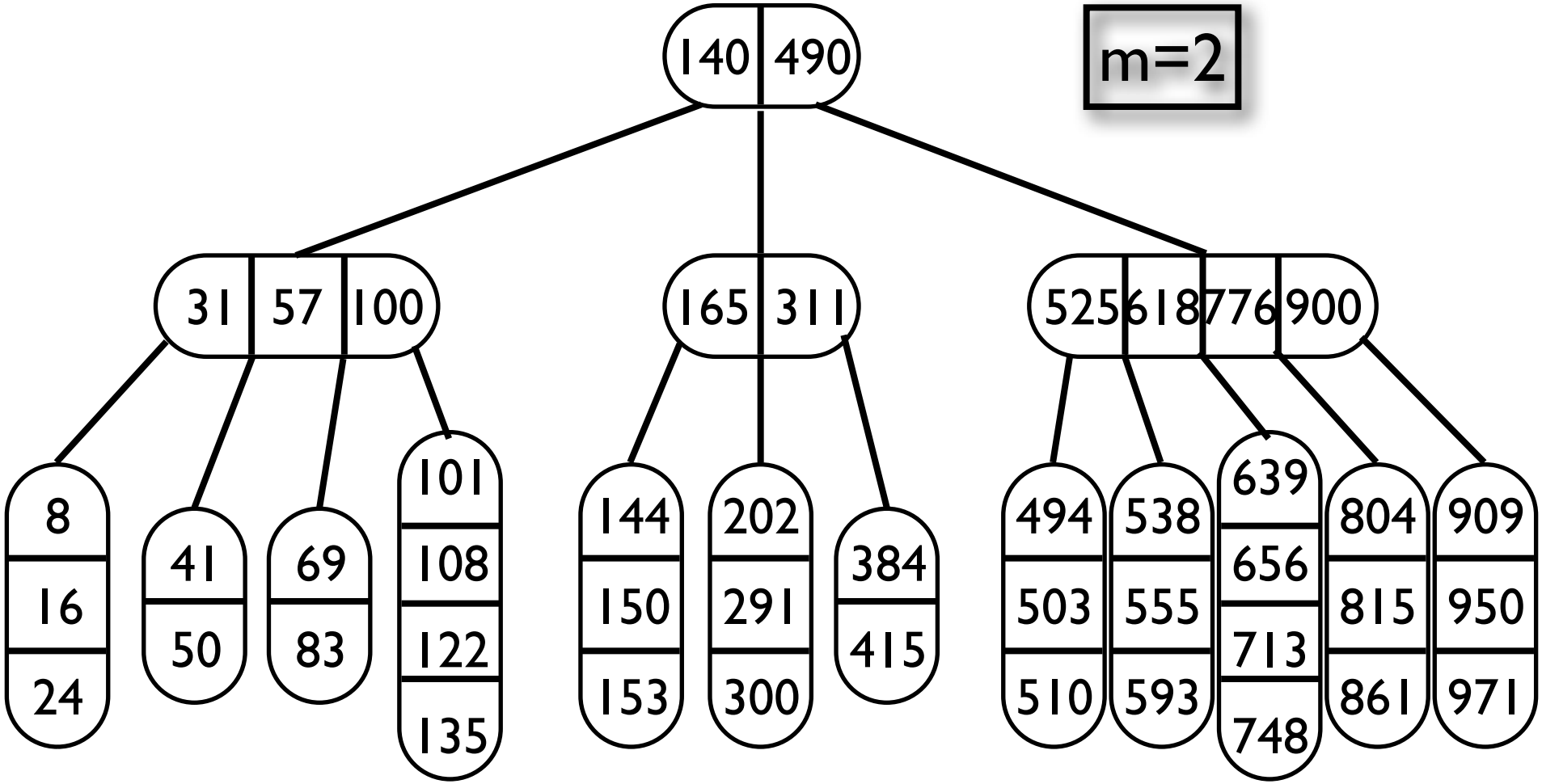
B-Bäume - Beispiel



Suchbaumeigenschaft

Suchen

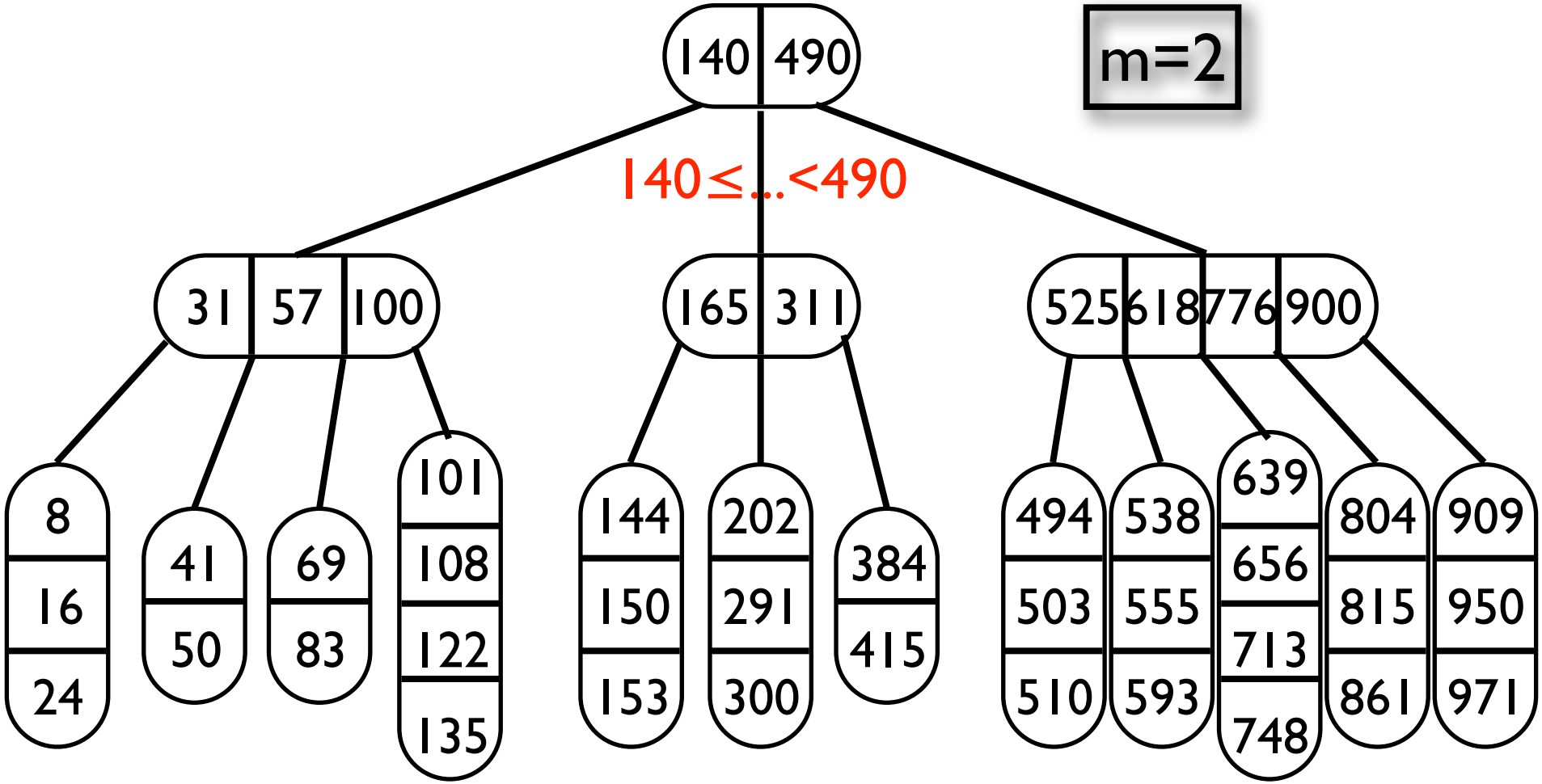
B-Bäume - Beispiel



Suchbaumeigenschaft

Suchen

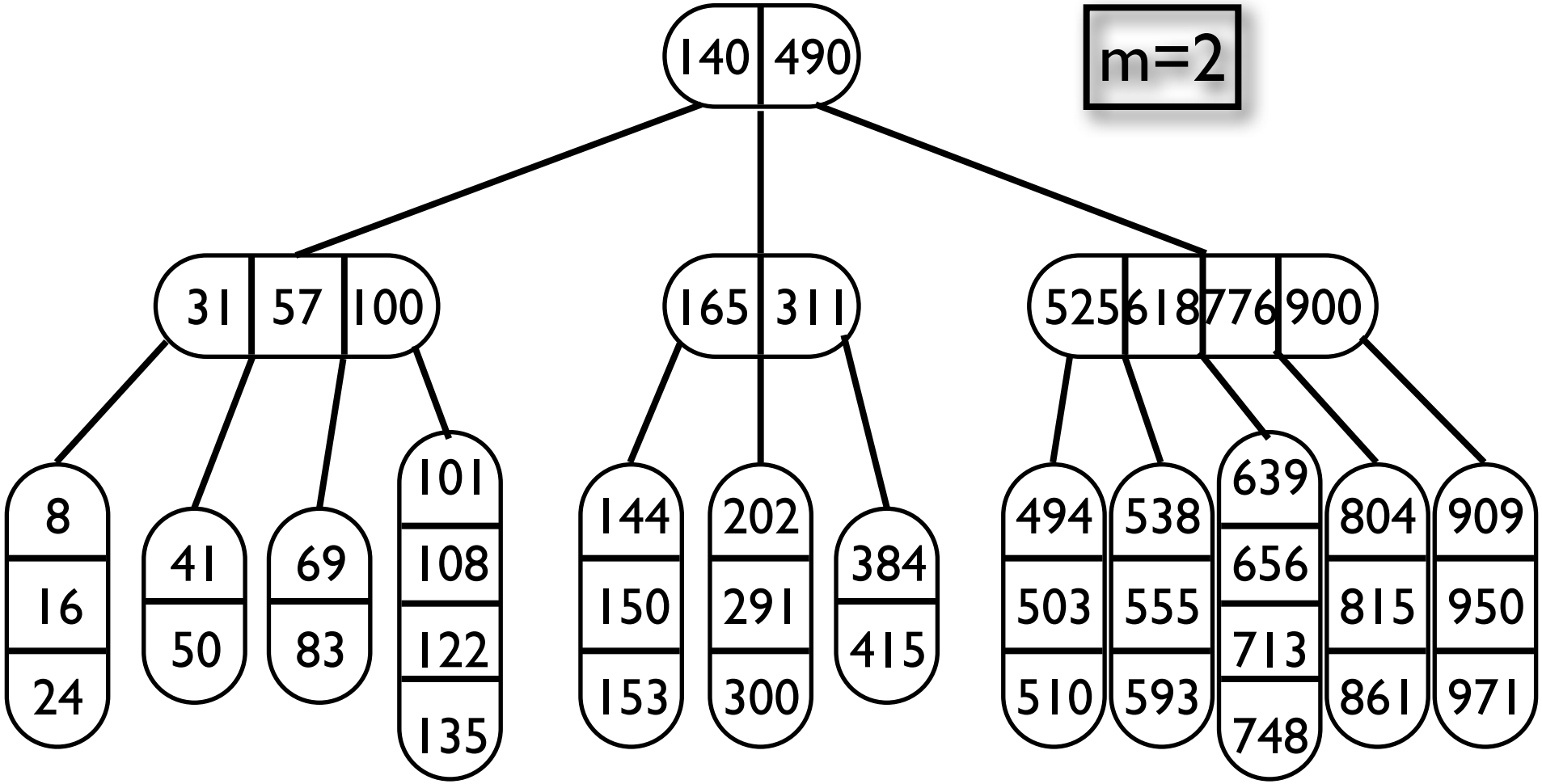
B-Bäume - Beispiel



Suchbaumeigenschaft

Suchen

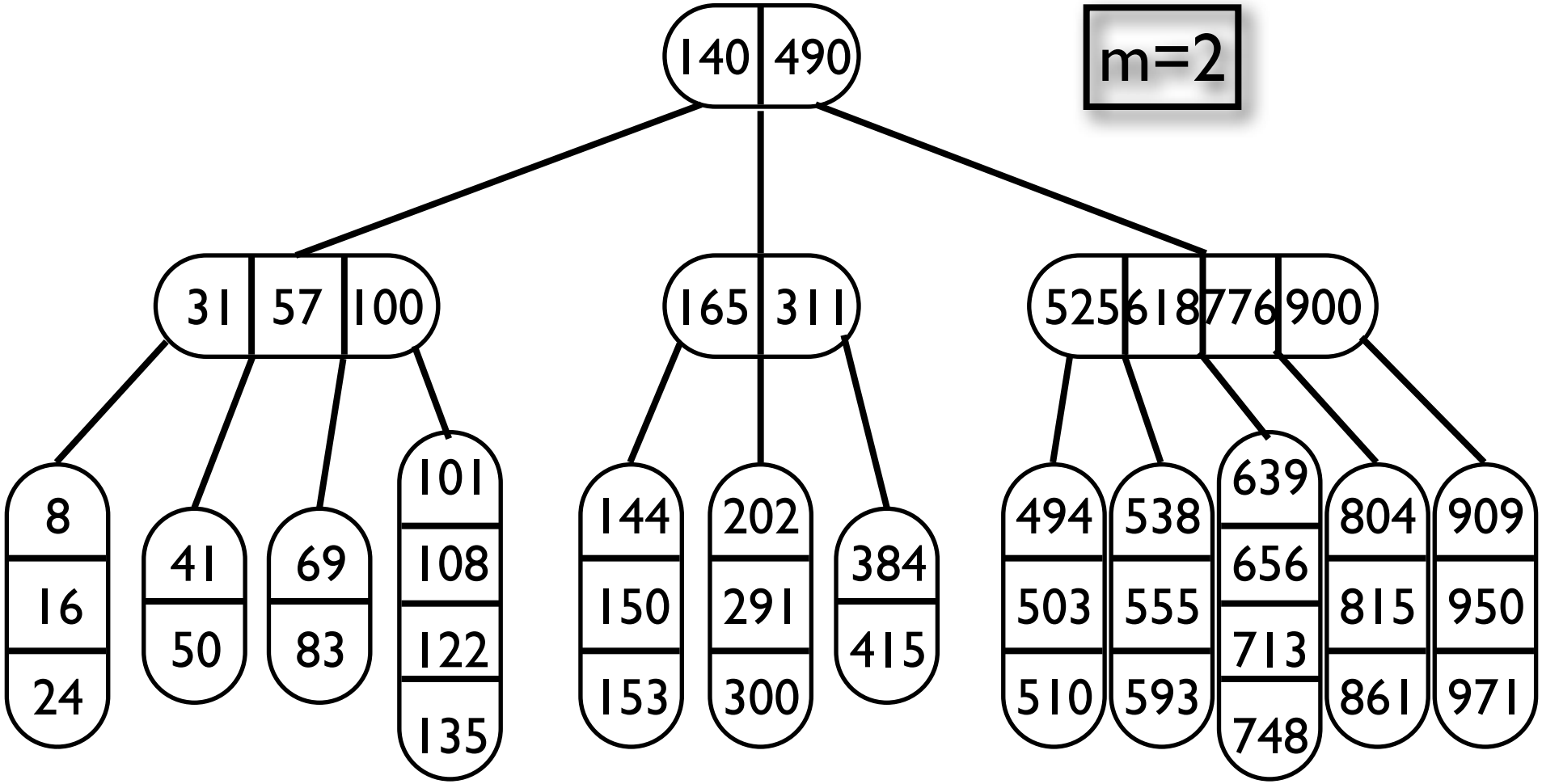
B-Bäume - Beispiel



Suchbaumeigenschaft

Suchen

B-Bäume - Beispiel

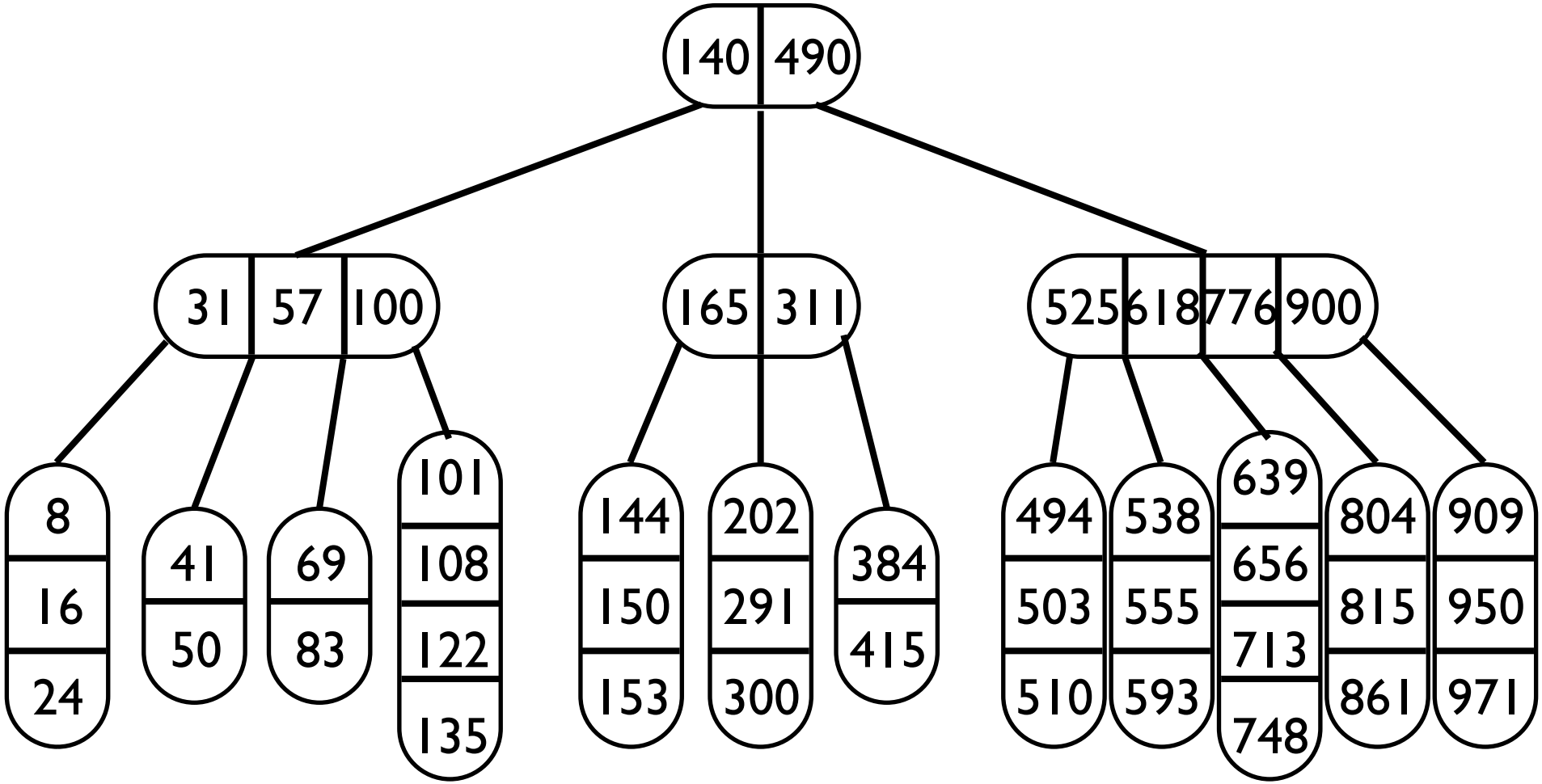


Suchen
B-Bäume - Operationen

- *Suchen*: klar
- *Einfügen*
  - grundsätzlich an den Blättern
  - Falls Überlauf, Knoten teilen, mittleren Schlüssel zum Vater reichen usw.
- *Löschen*: aufwendiger
  - Prinzip: (Algorithmus s. Buch von Güting)
    - Anreichern von Knoten um Schlüssel aus dem Vater
    - Verschmelzen von Knoten

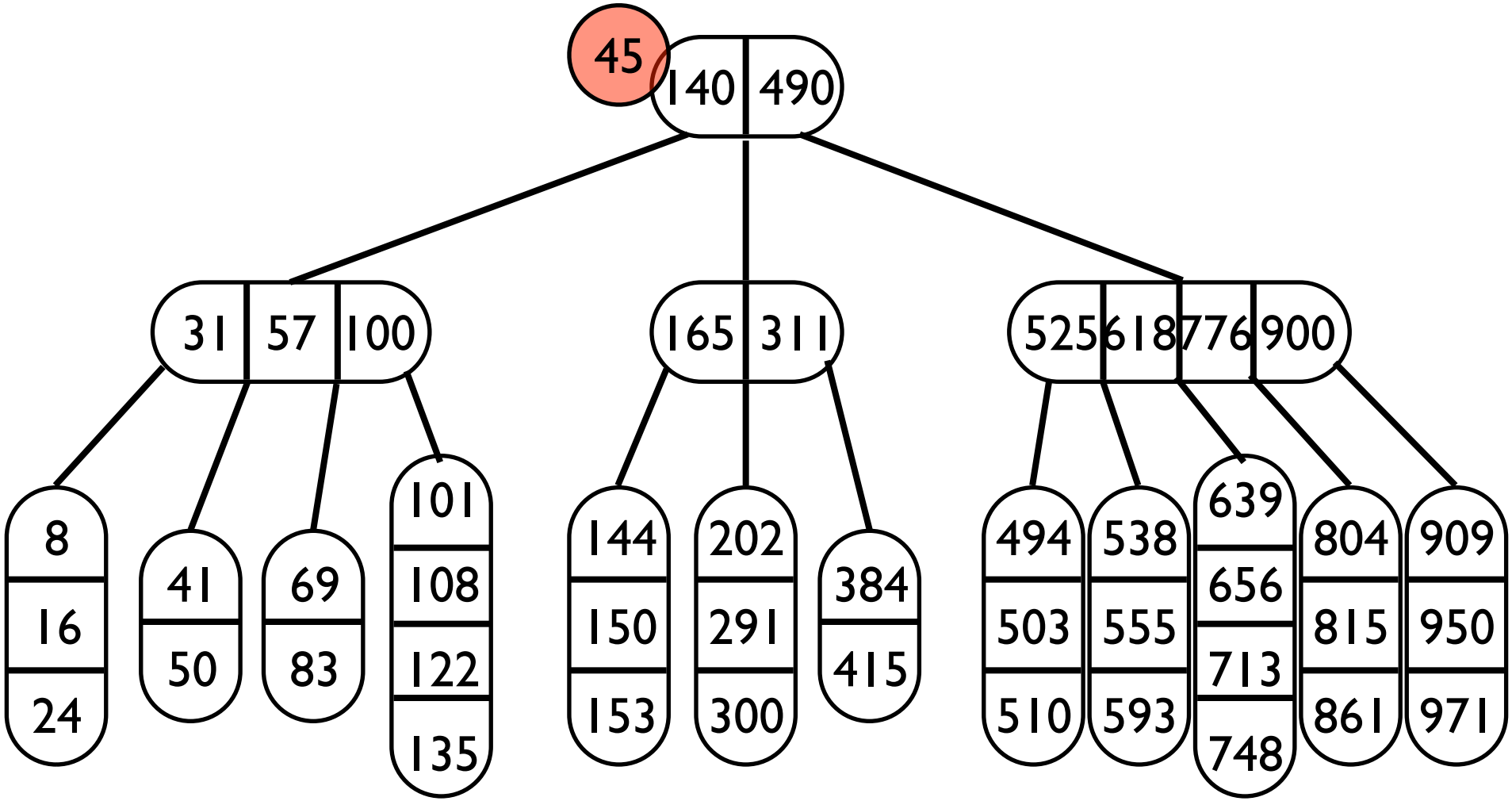
Suchen

B-Bäume - Beispiel: Einfügen 45



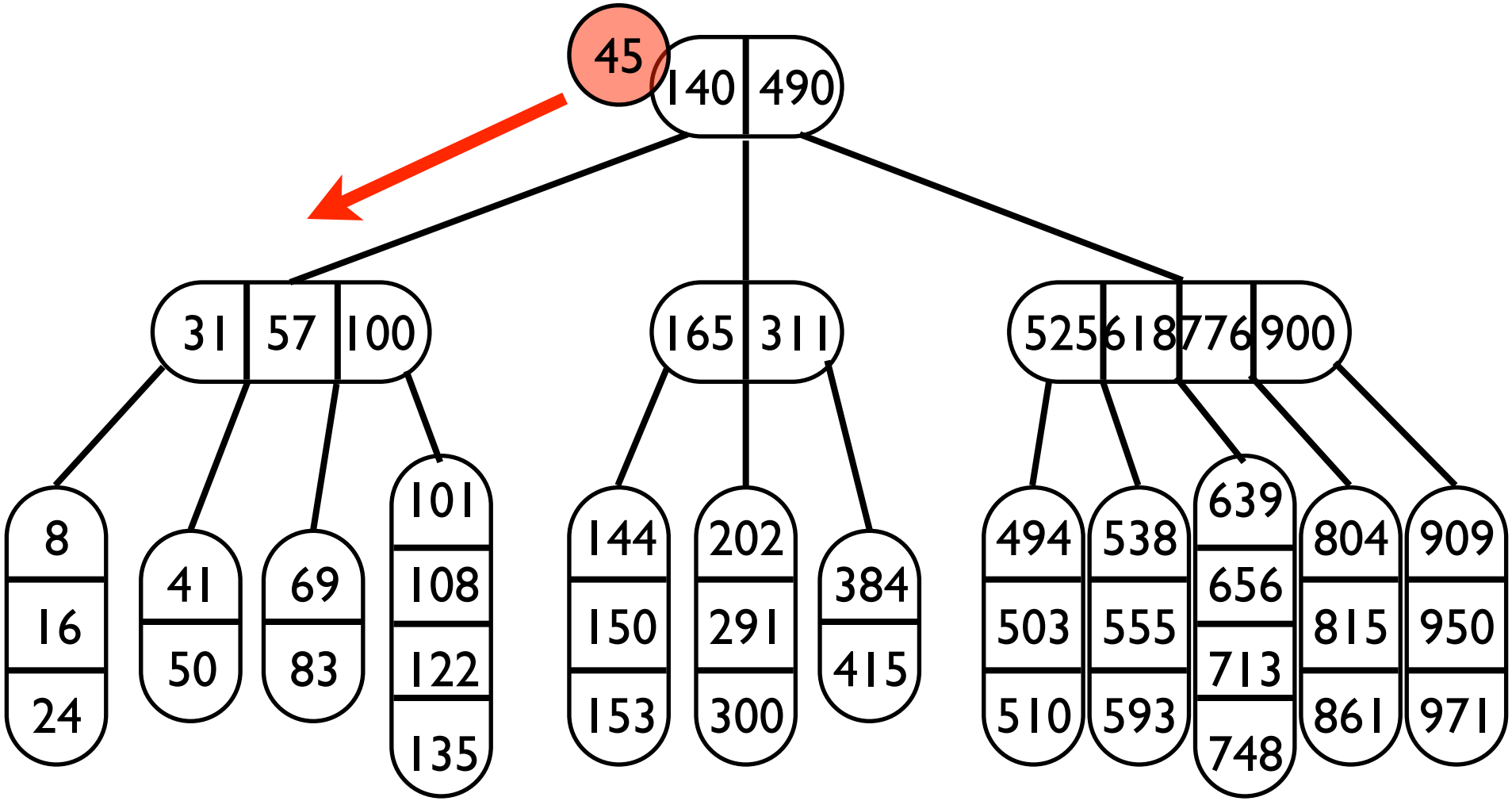
Suchen

B-Bäume - Beispiel: Einfügen 45



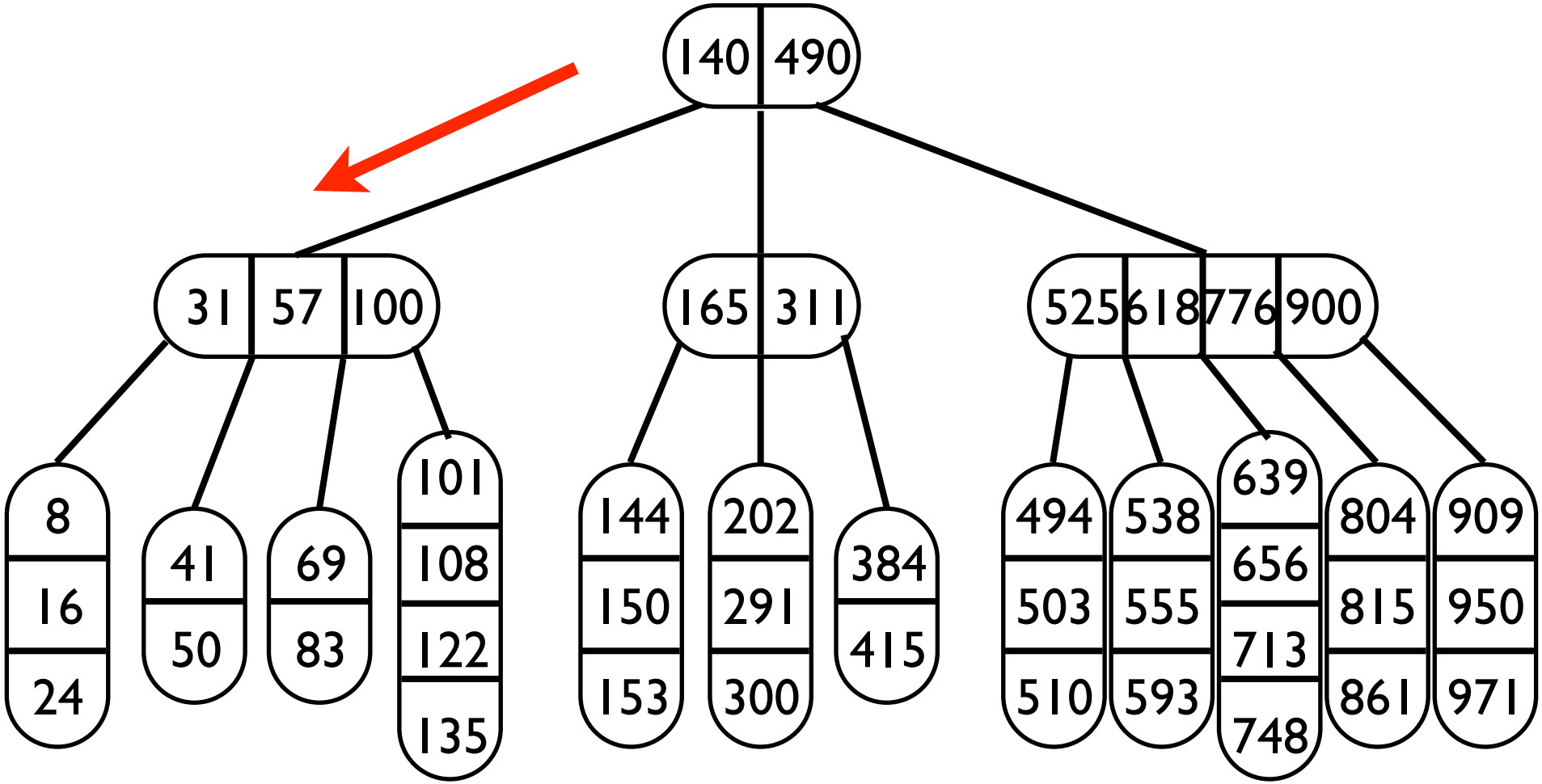
Suchen

B-Bäume - Beispiel: Einfügen 45



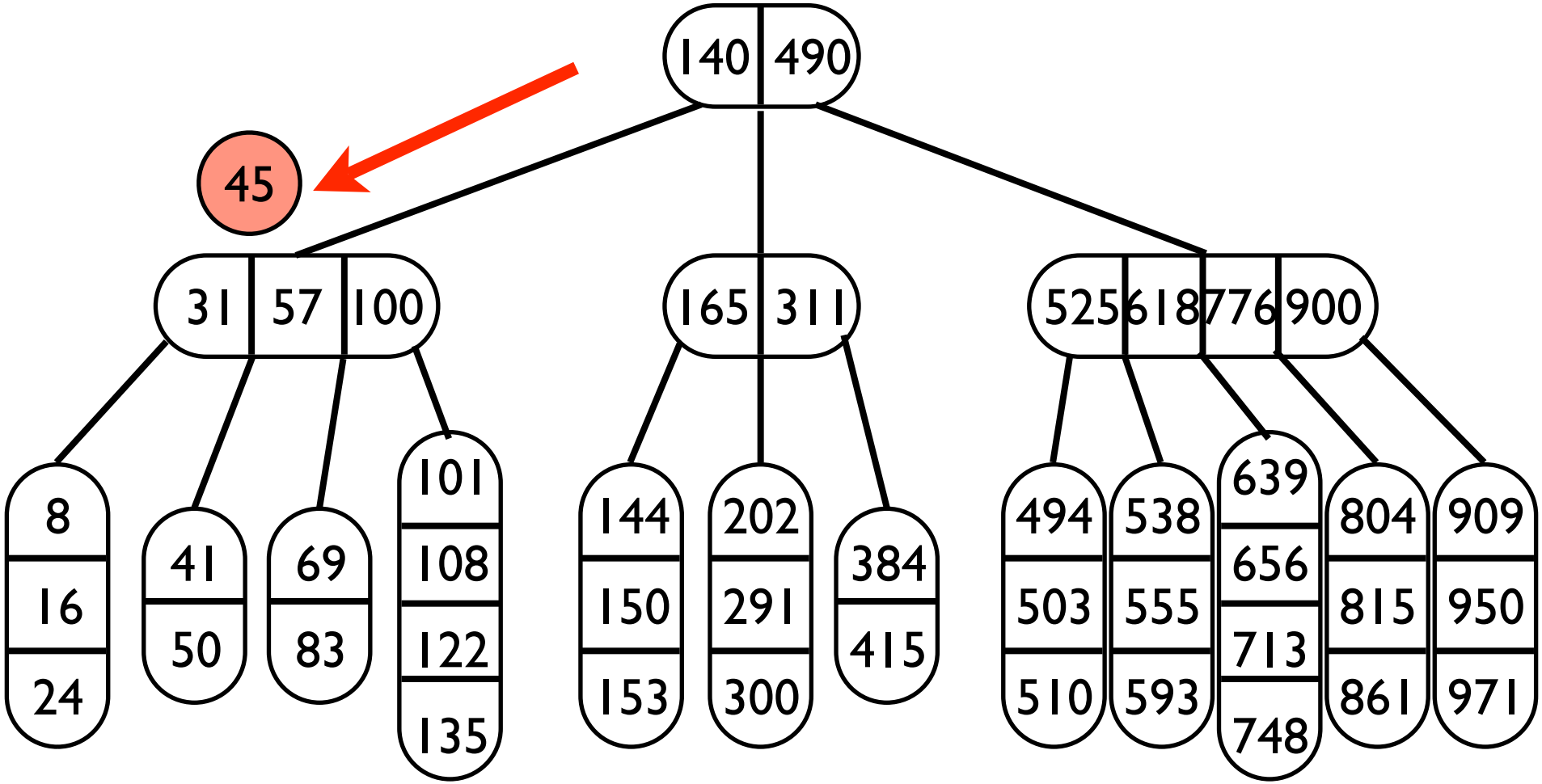
Suchen

B-Bäume - Beispiel: Einfügen 45



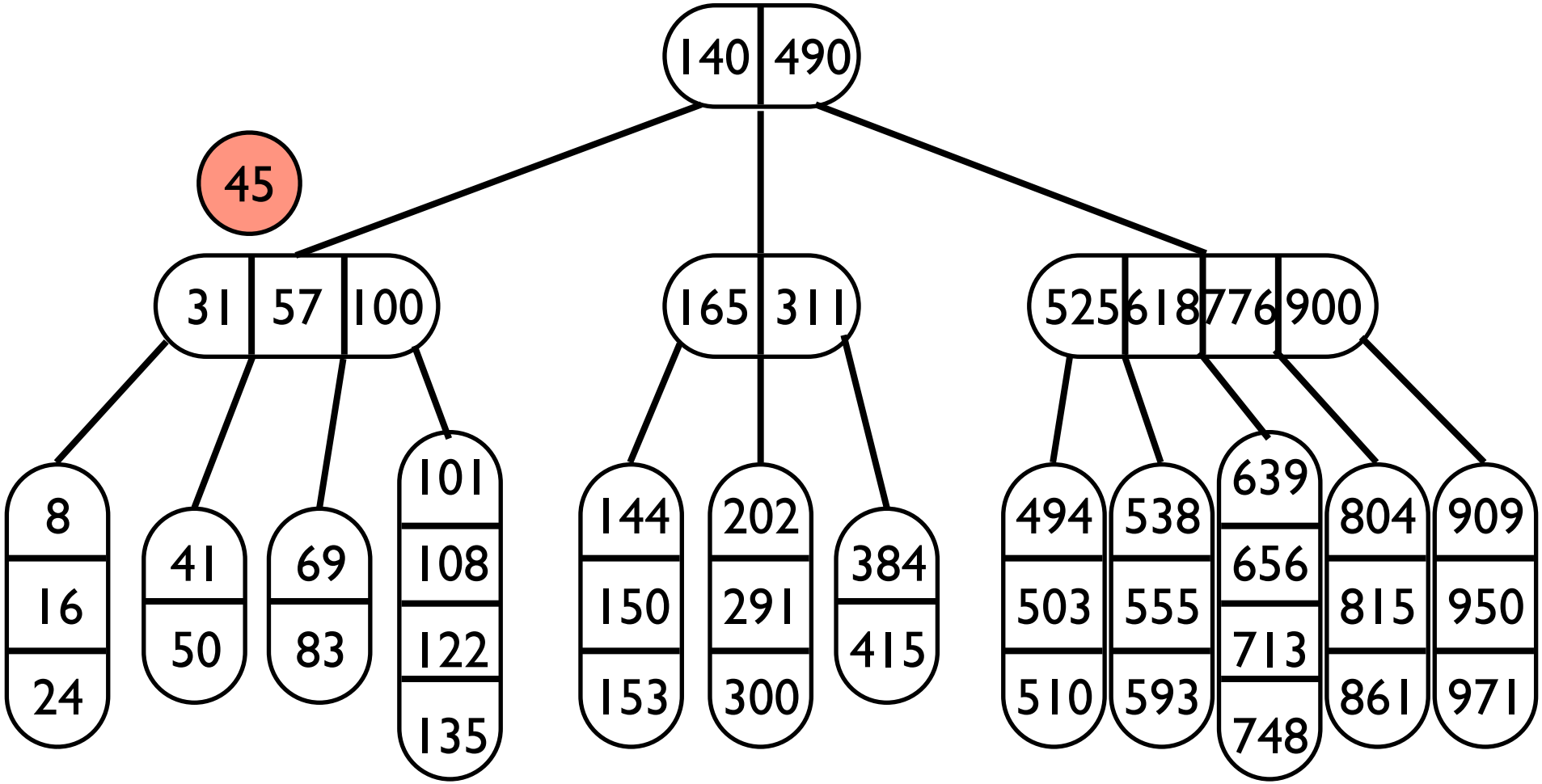
Suchen

B-Bäume - Beispiel: Einfügen 45



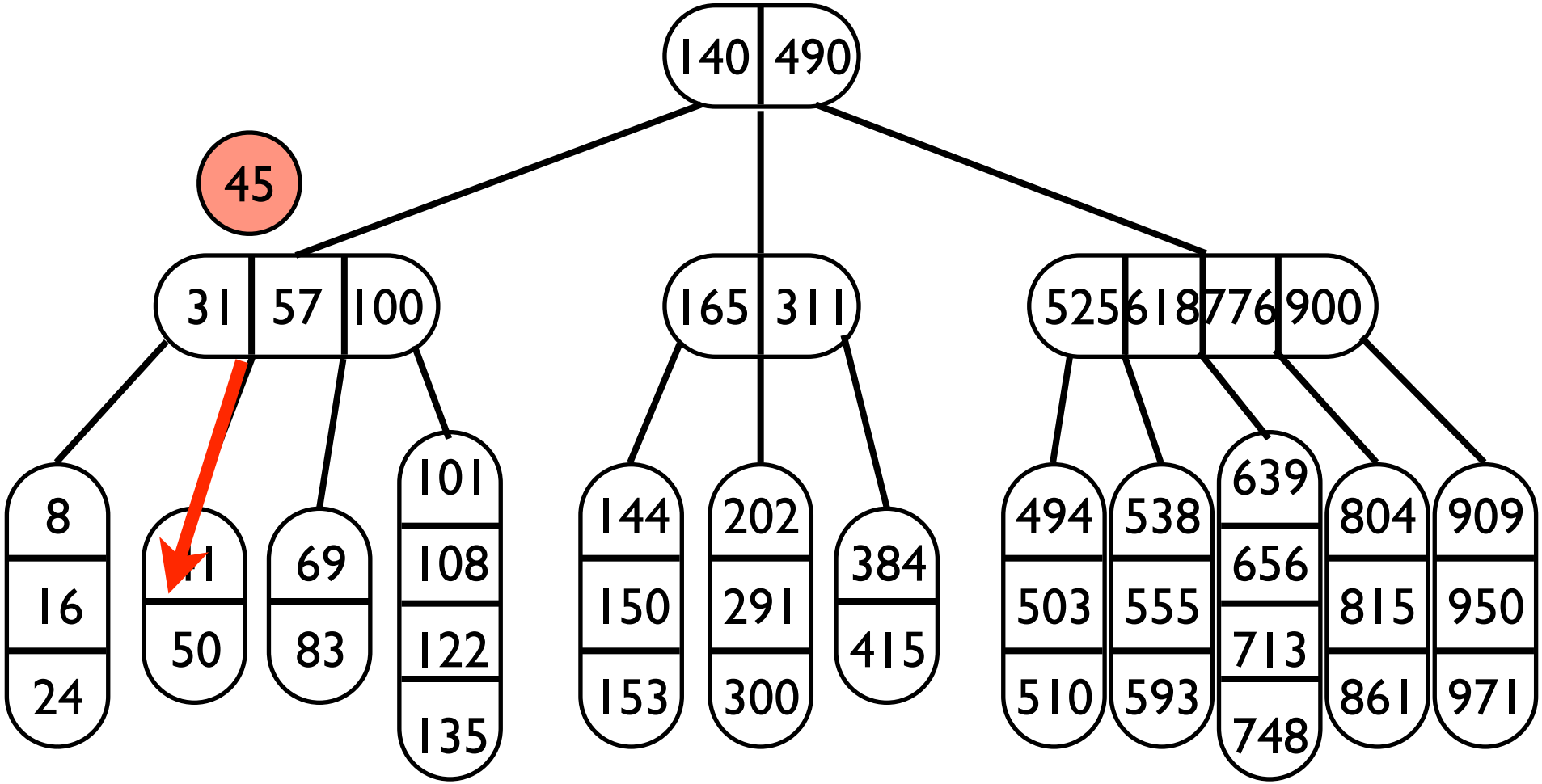
Suchen

B-Bäume - Beispiel: Einfügen 45



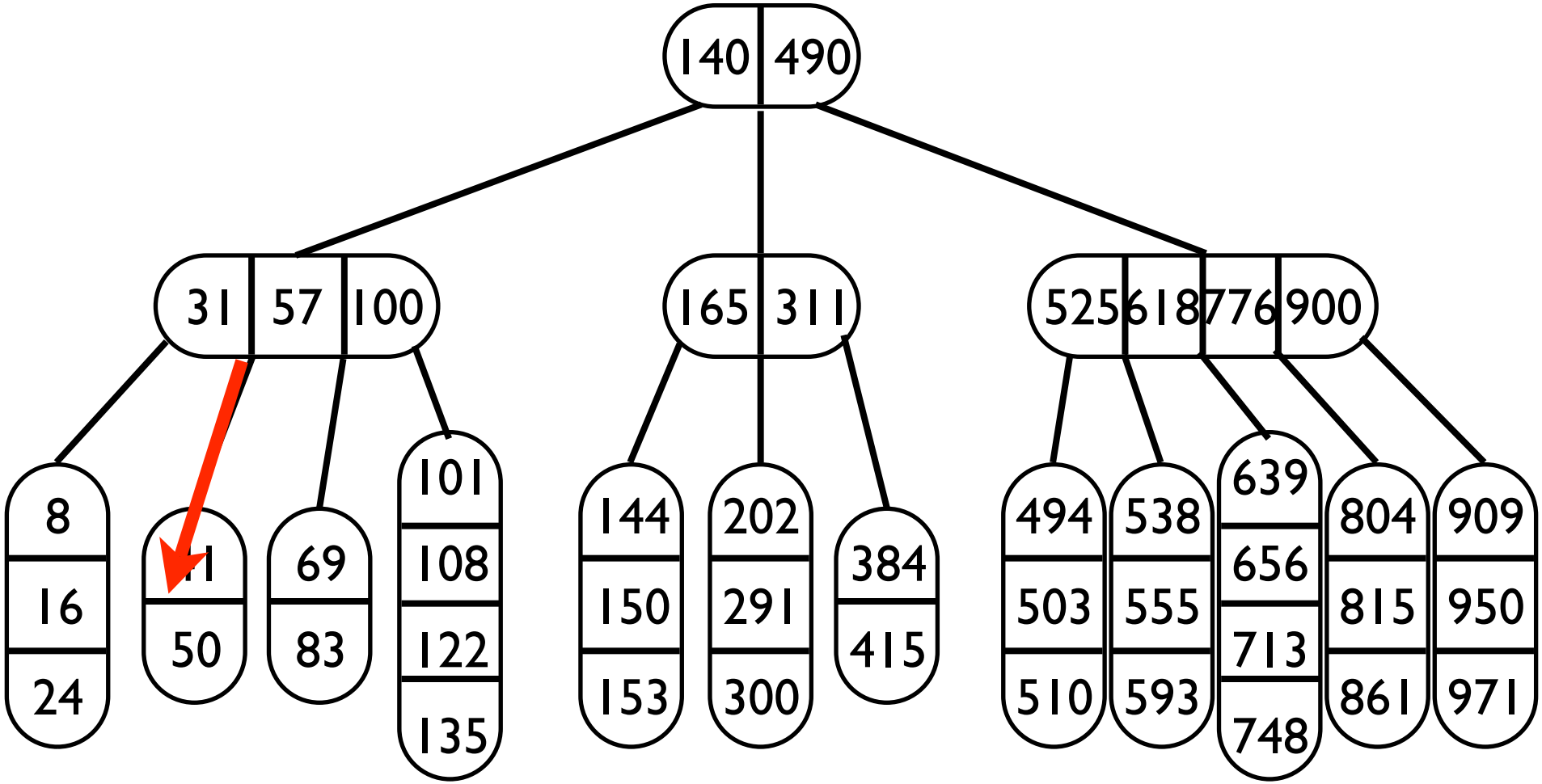
Suchen

B-Bäume - Beispiel: Einfügen 45



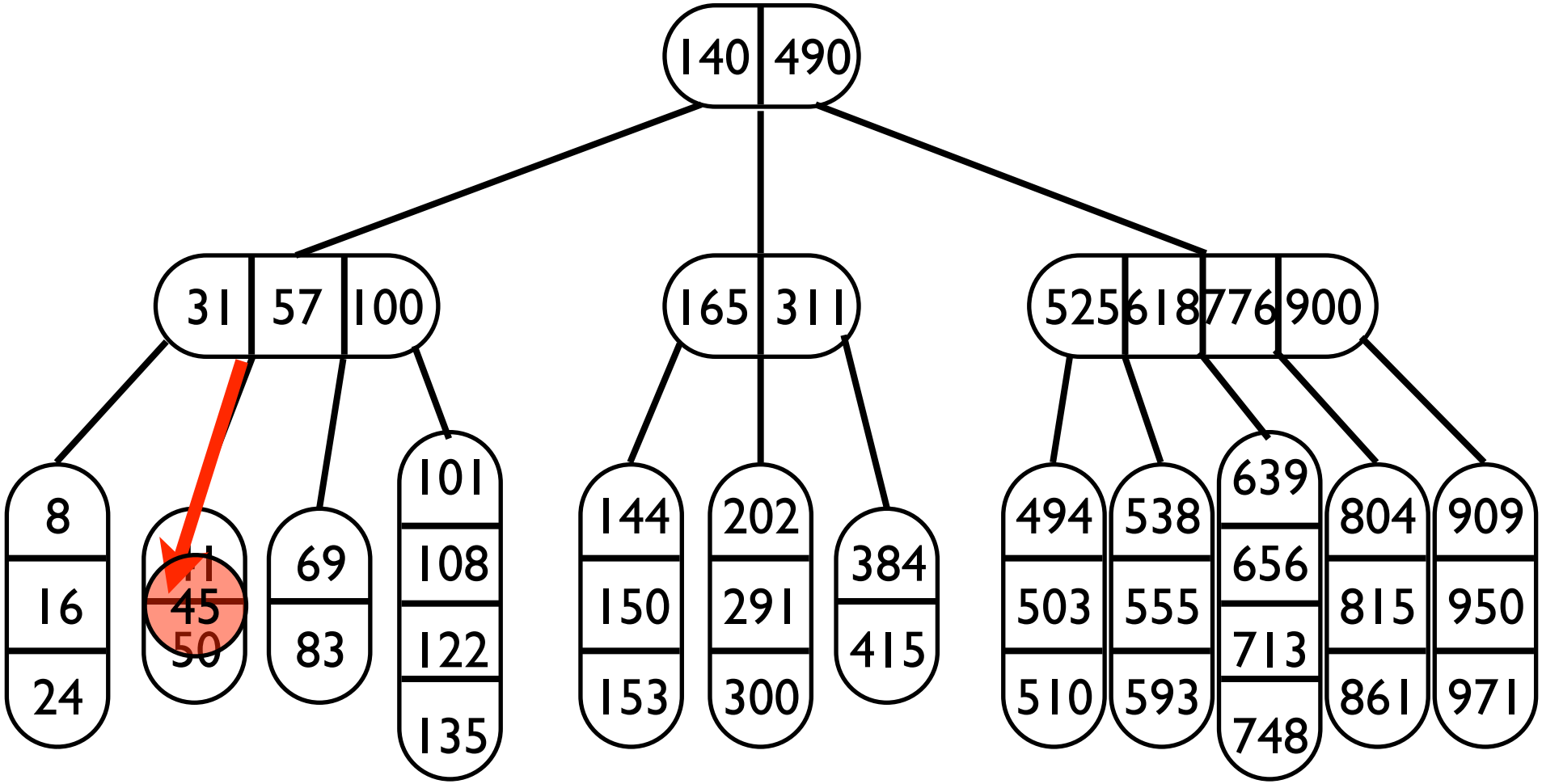
Suchen

B-Bäume - Beispiel: Einfügen 45



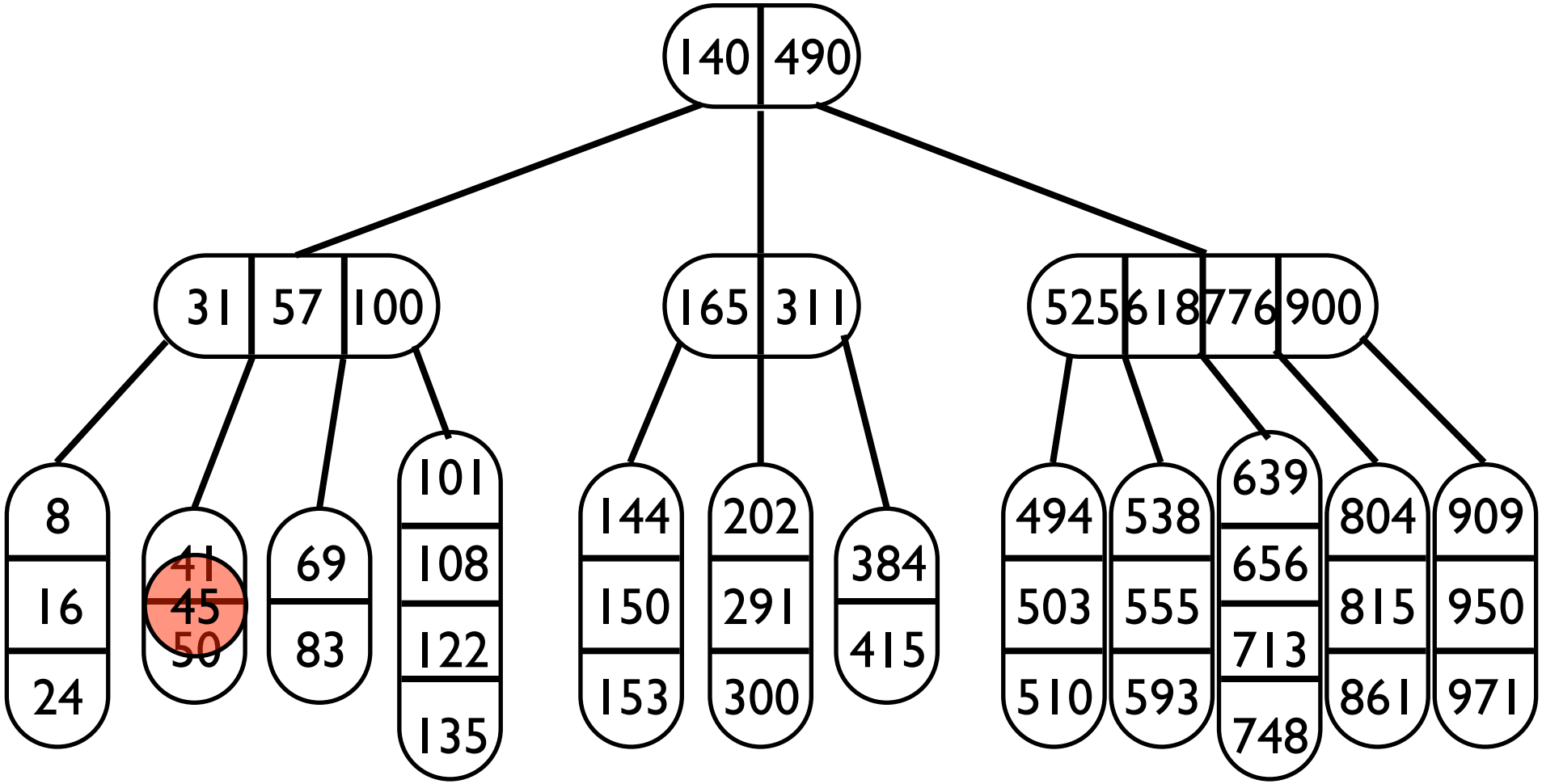
Suchen

B-Bäume - Beispiel: Einfügen 45



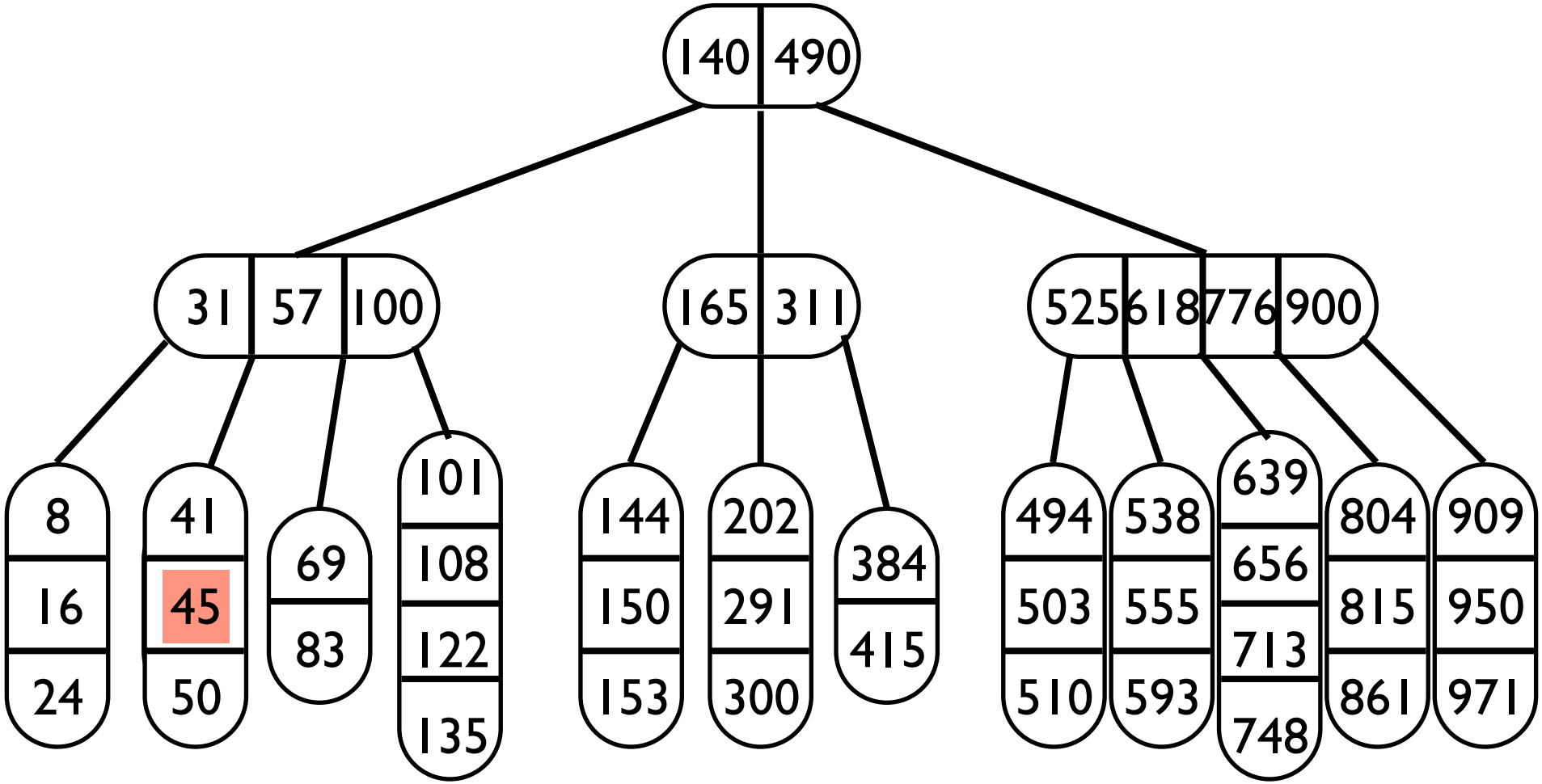
Suchen

B-Bäume - Beispiel: Einfügen 45



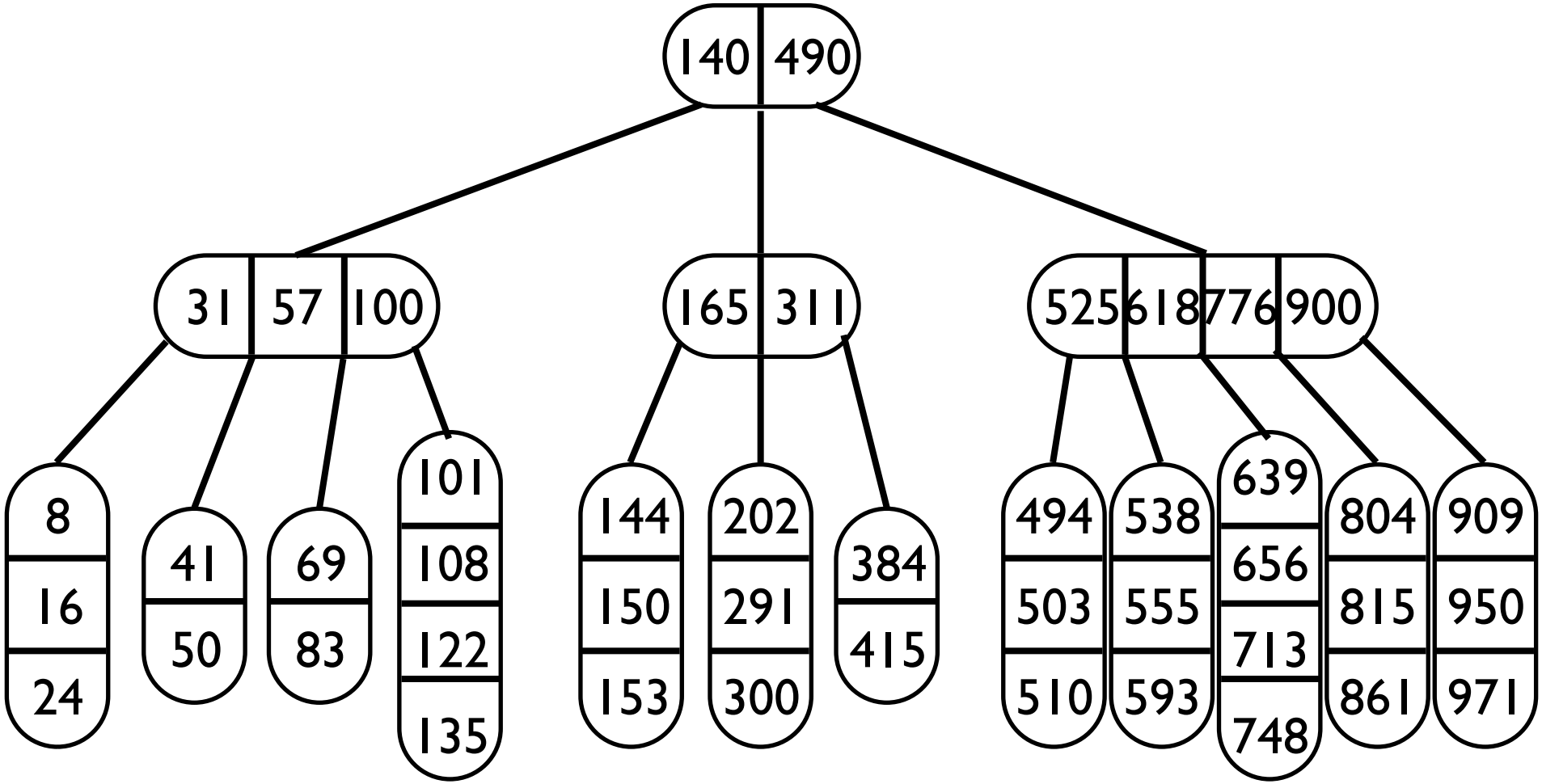
Suchen

B-Bäume - Beispiel: Einfügen 45



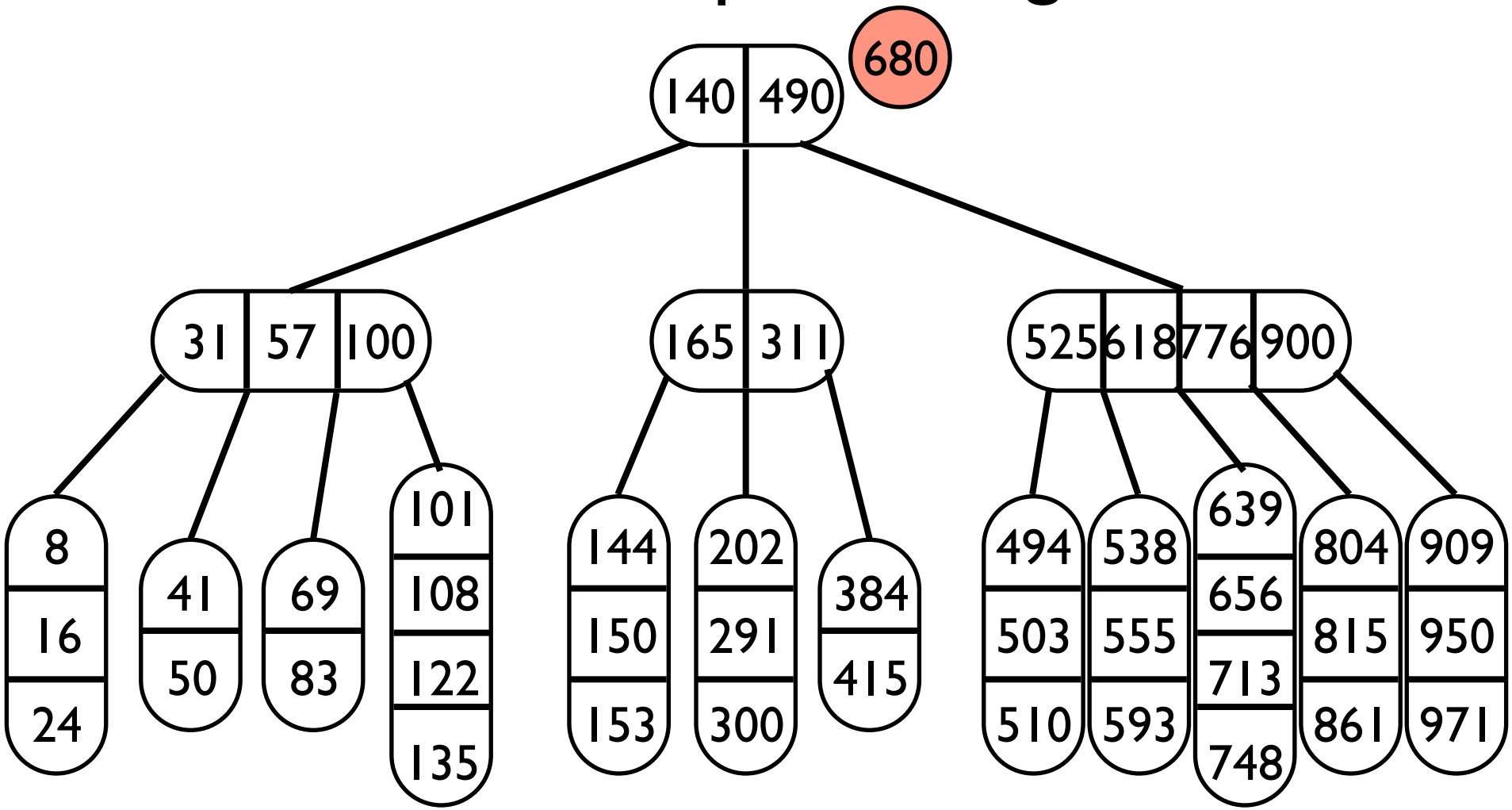
Suchen

B-Bäume - Beispiel: Einfügen 680



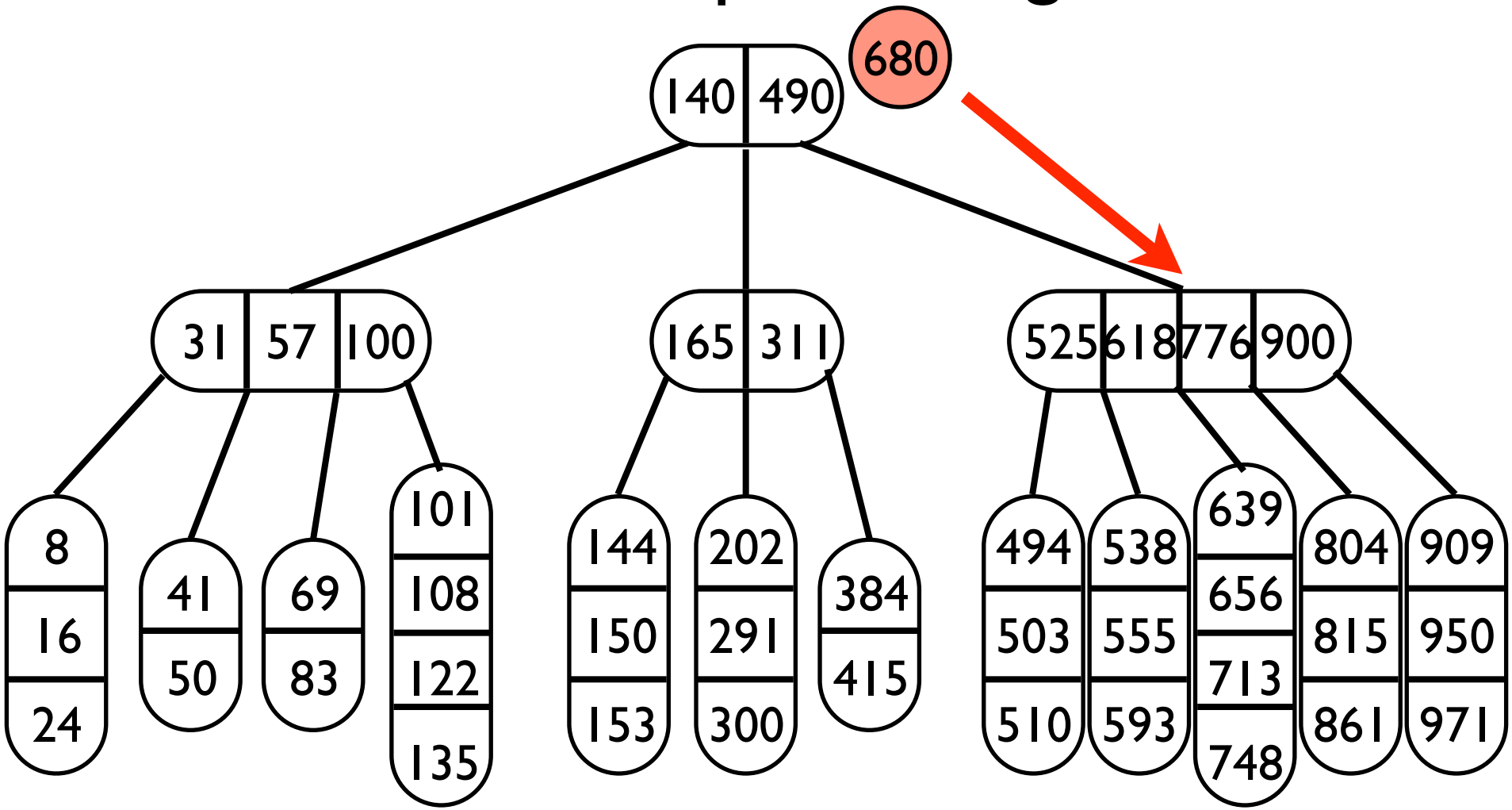
Suchen

B-Bäume - Beispiel: Einfügen 680



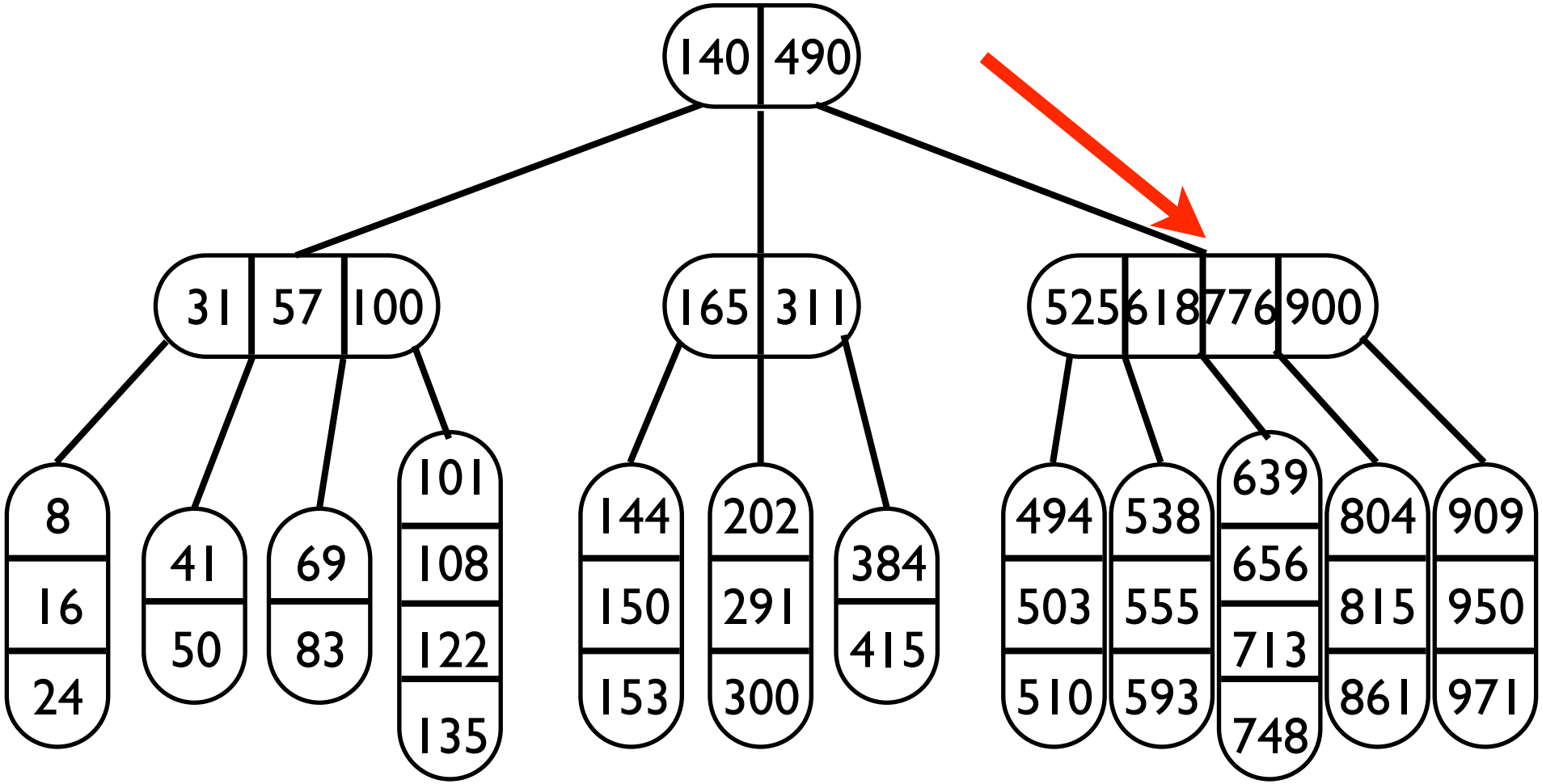
Suchen

B-Bäume - Beispiel: Einfügen 680



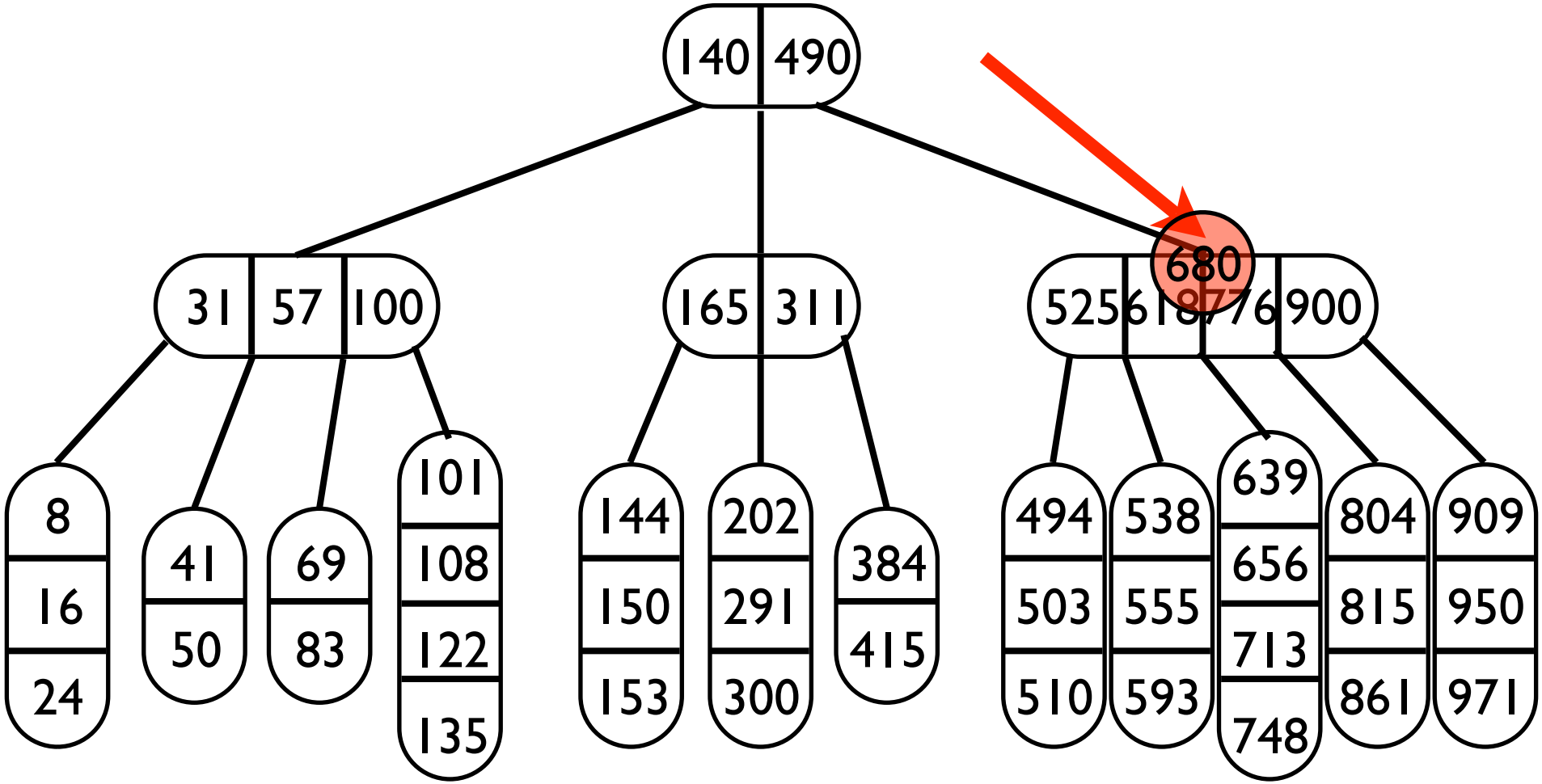
Suchen

B-Bäume - Beispiel: Einfügen 680



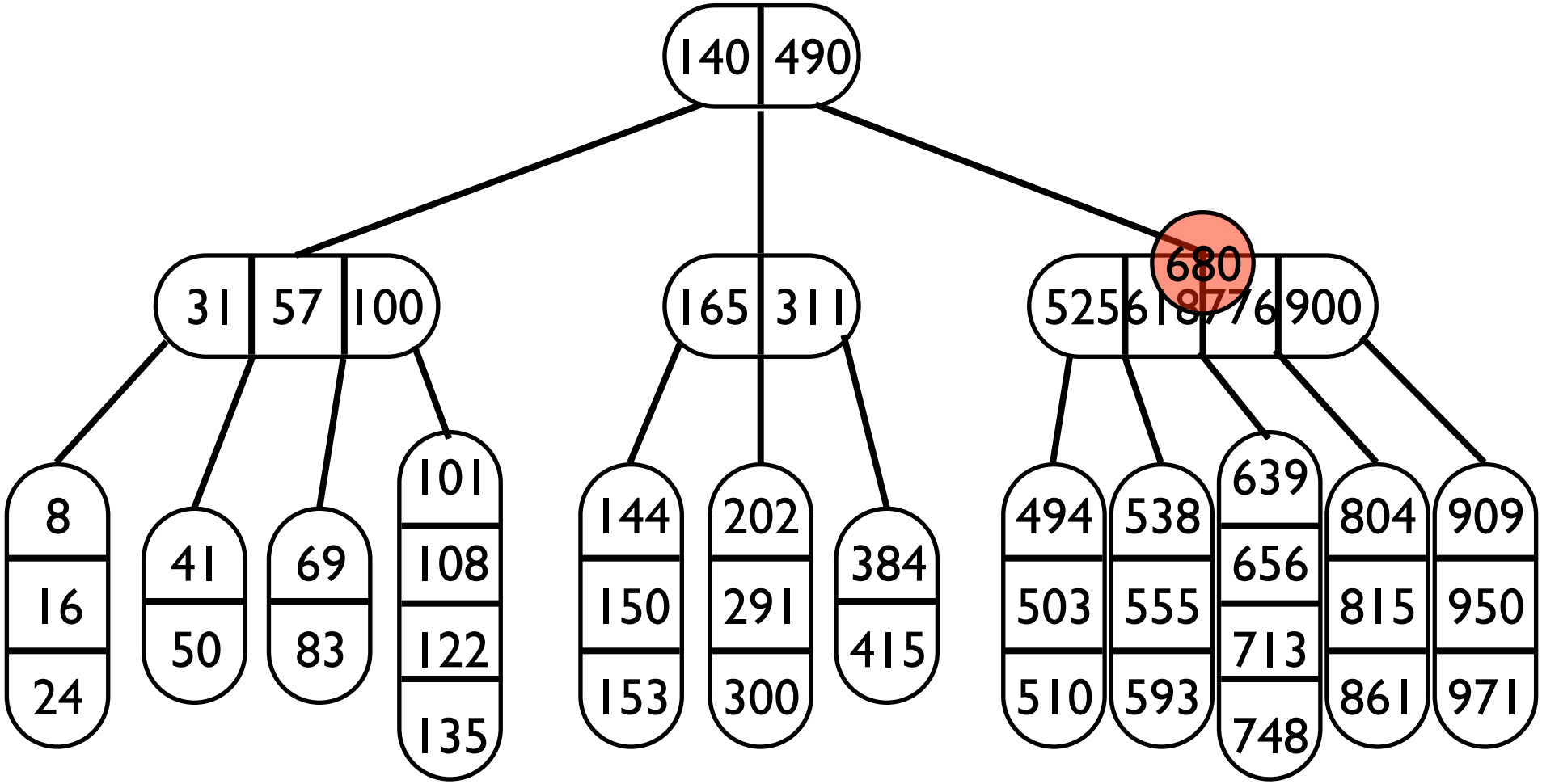
Suchen

B-Bäume - Beispiel: Einfügen 680



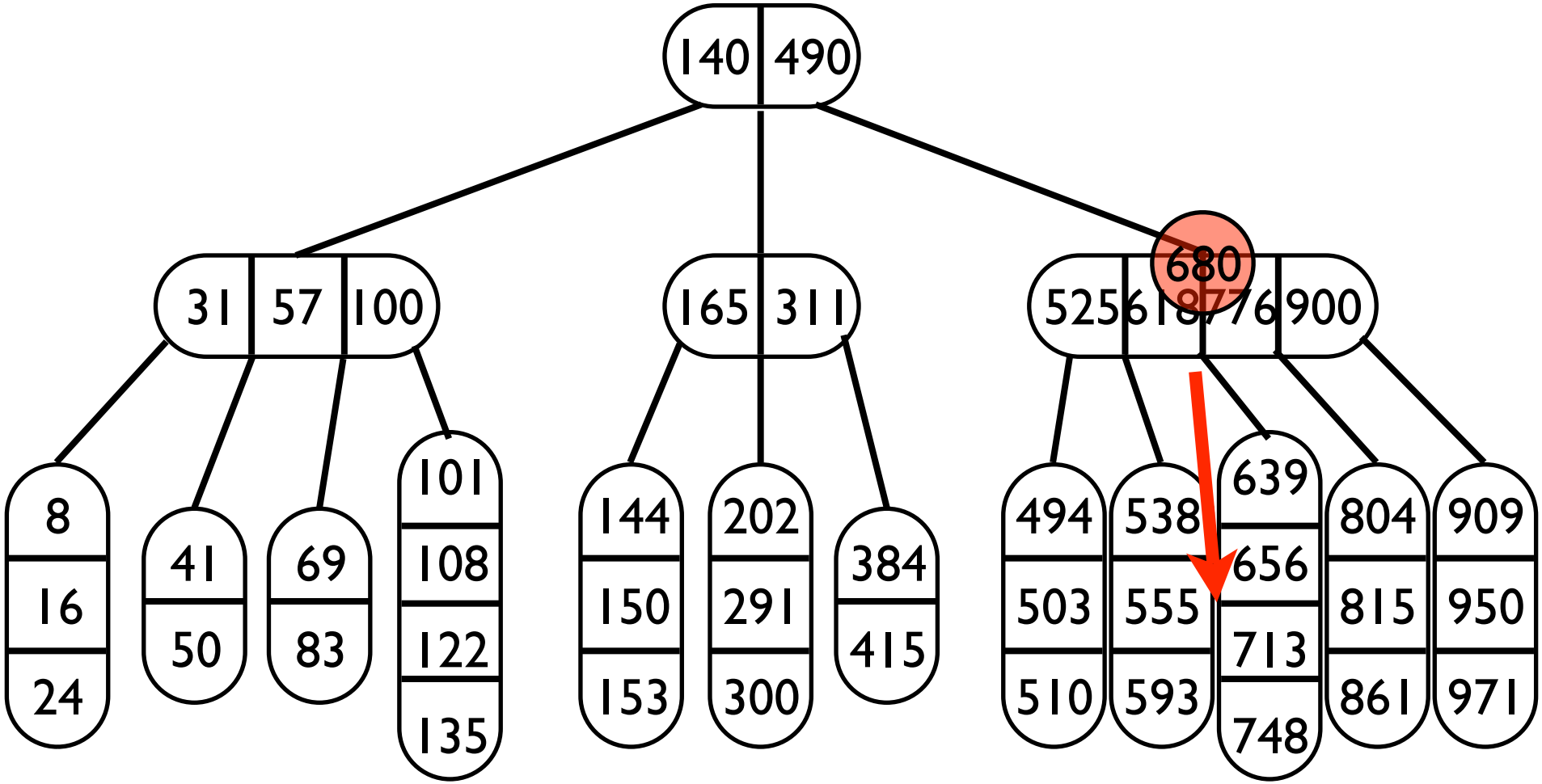
Suchen

B-Bäume - Beispiel: Einfügen 680



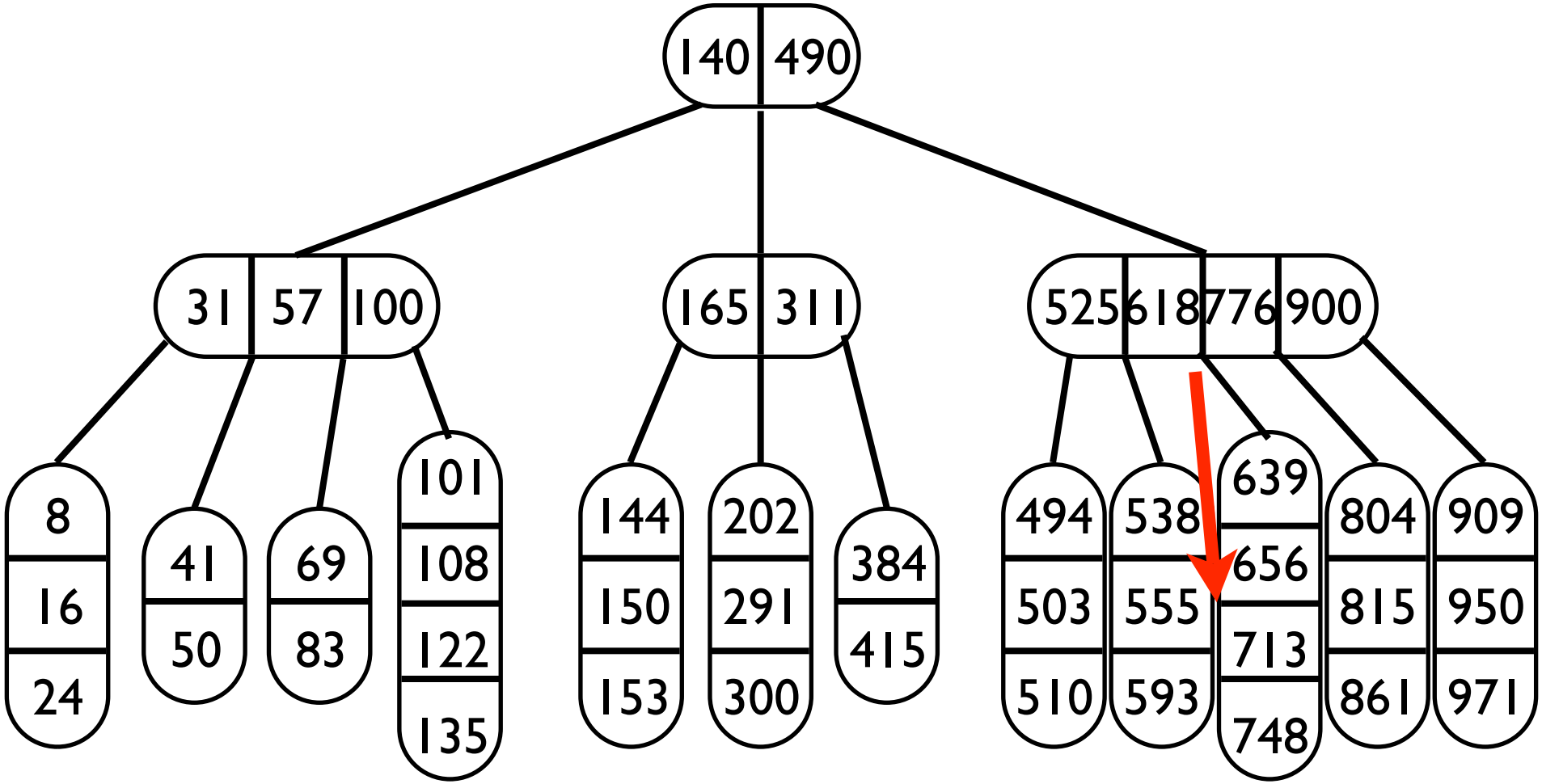
Suchen

B-Bäume - Beispiel: Einfügen 680



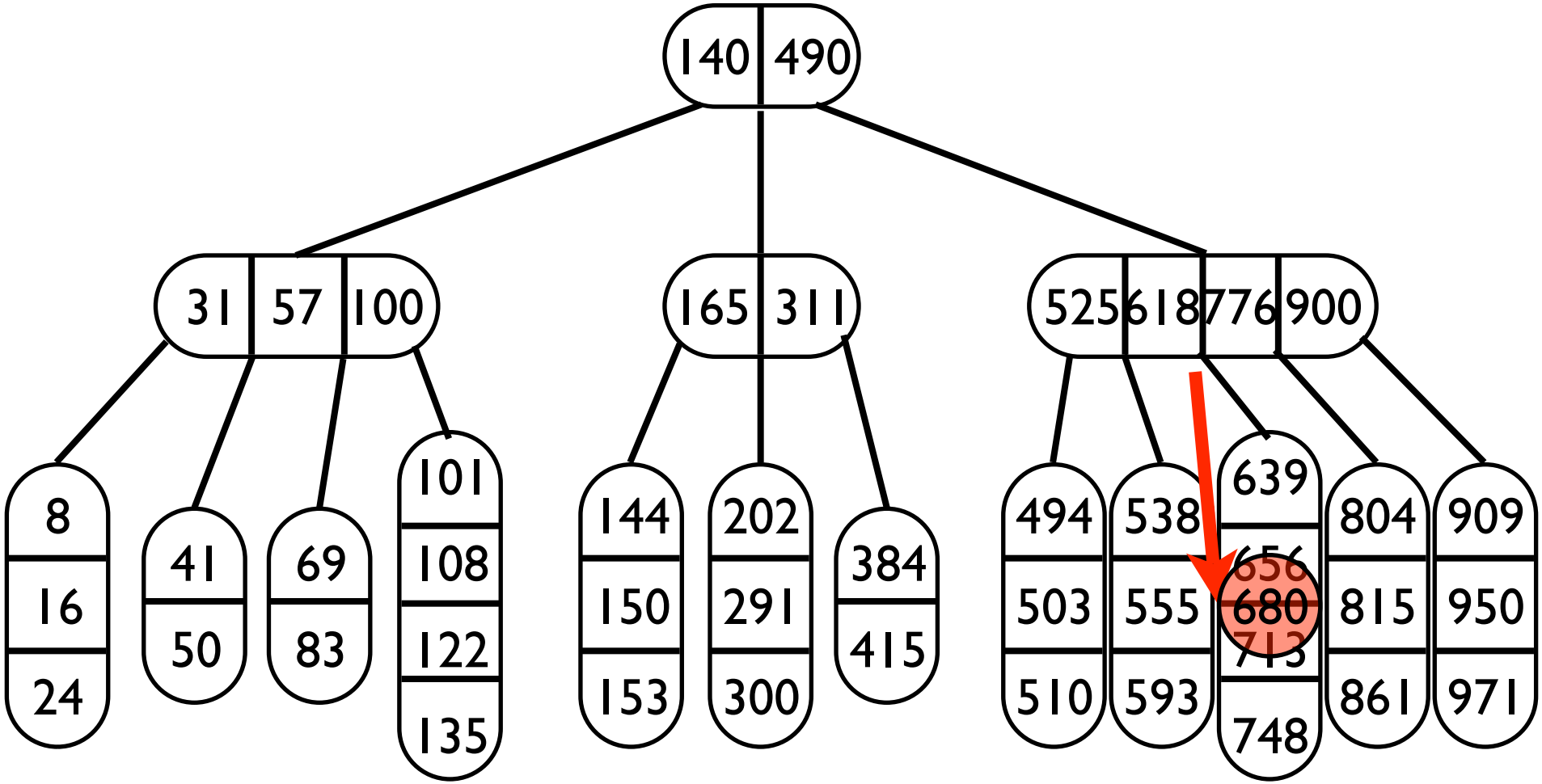
Suchen

B-Bäume - Beispiel: Einfügen 680



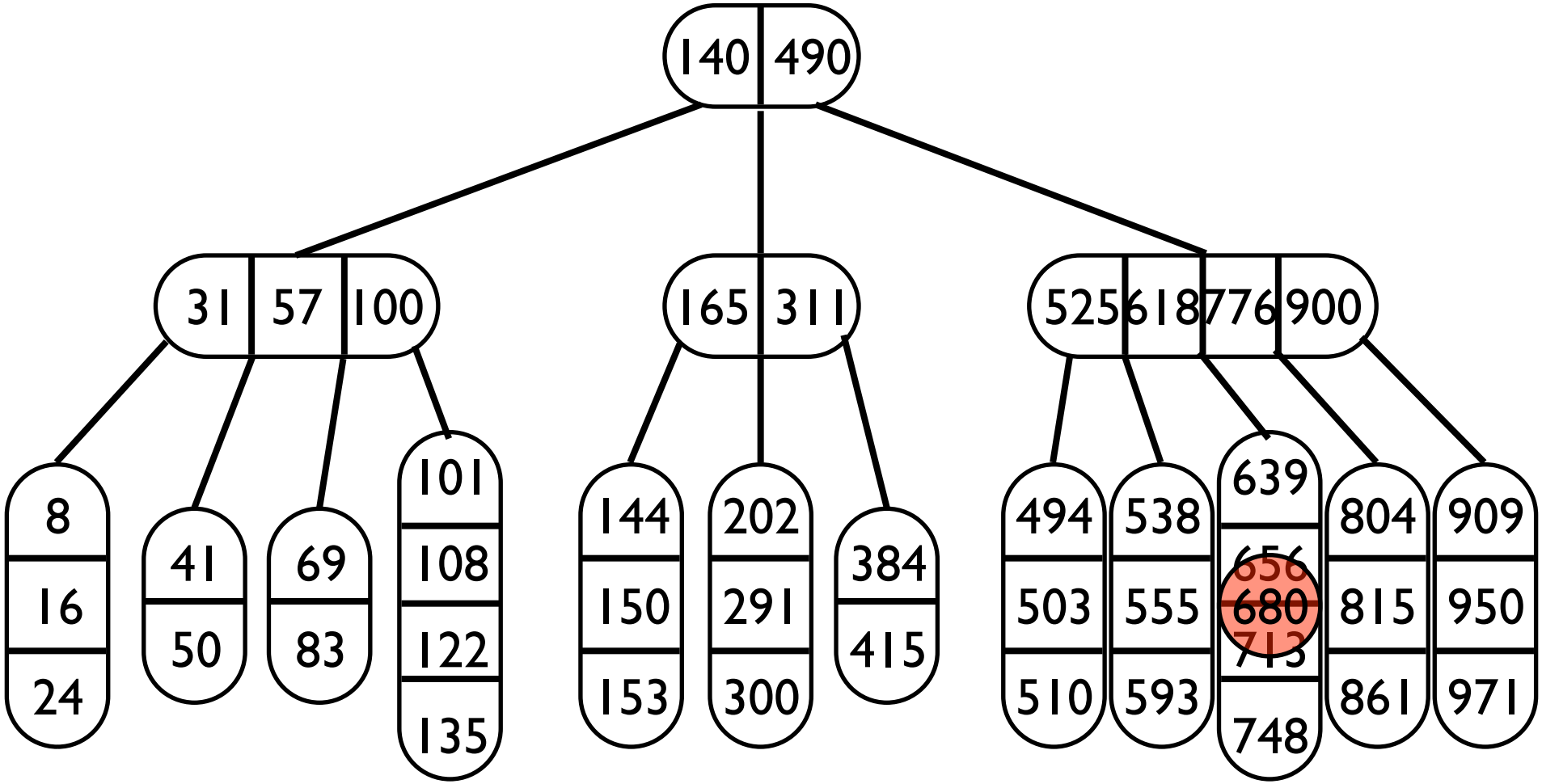
Suchen

B-Bäume - Beispiel: Einfügen 680



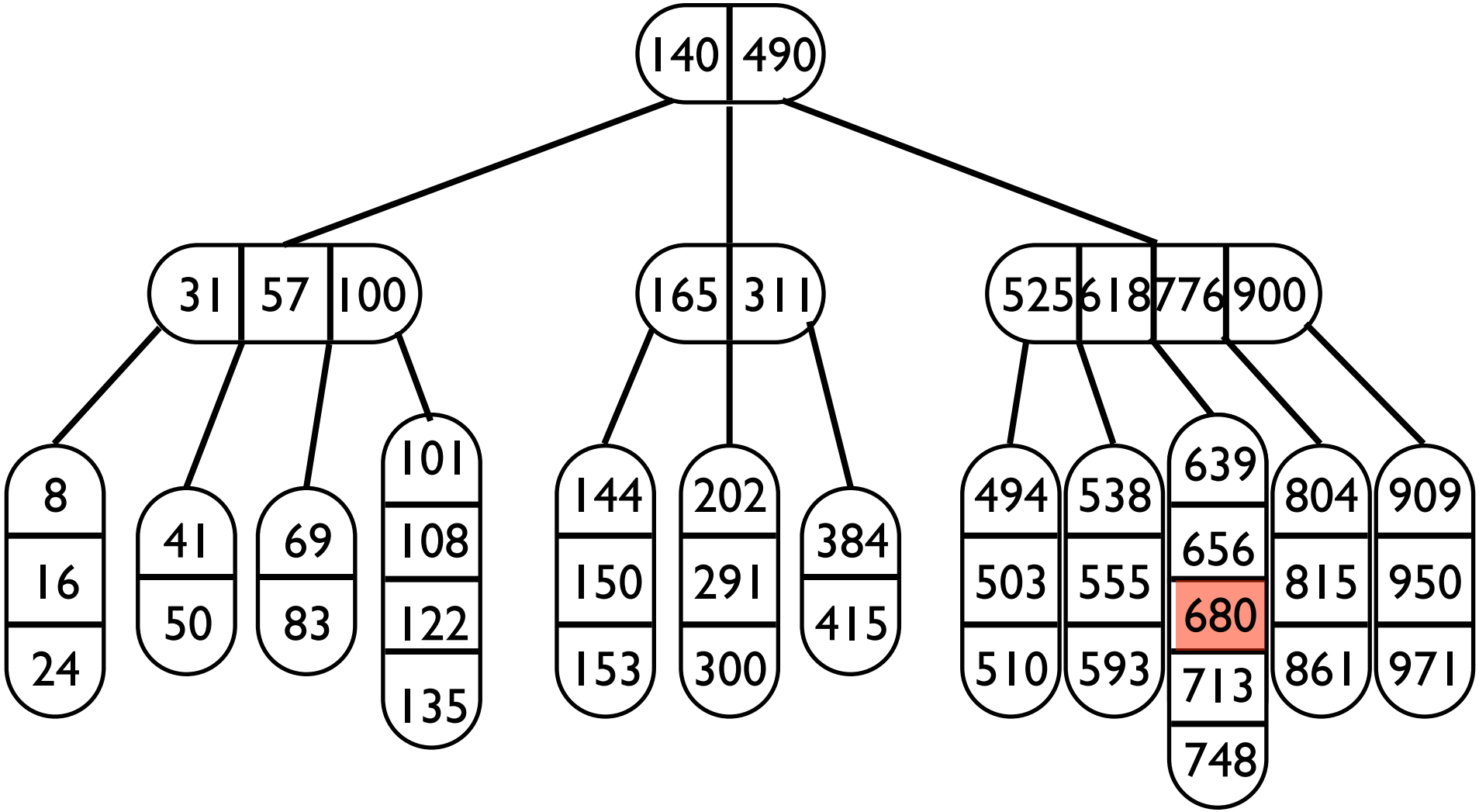
Suchen

B-Bäume - Beispiel: Einfügen 680



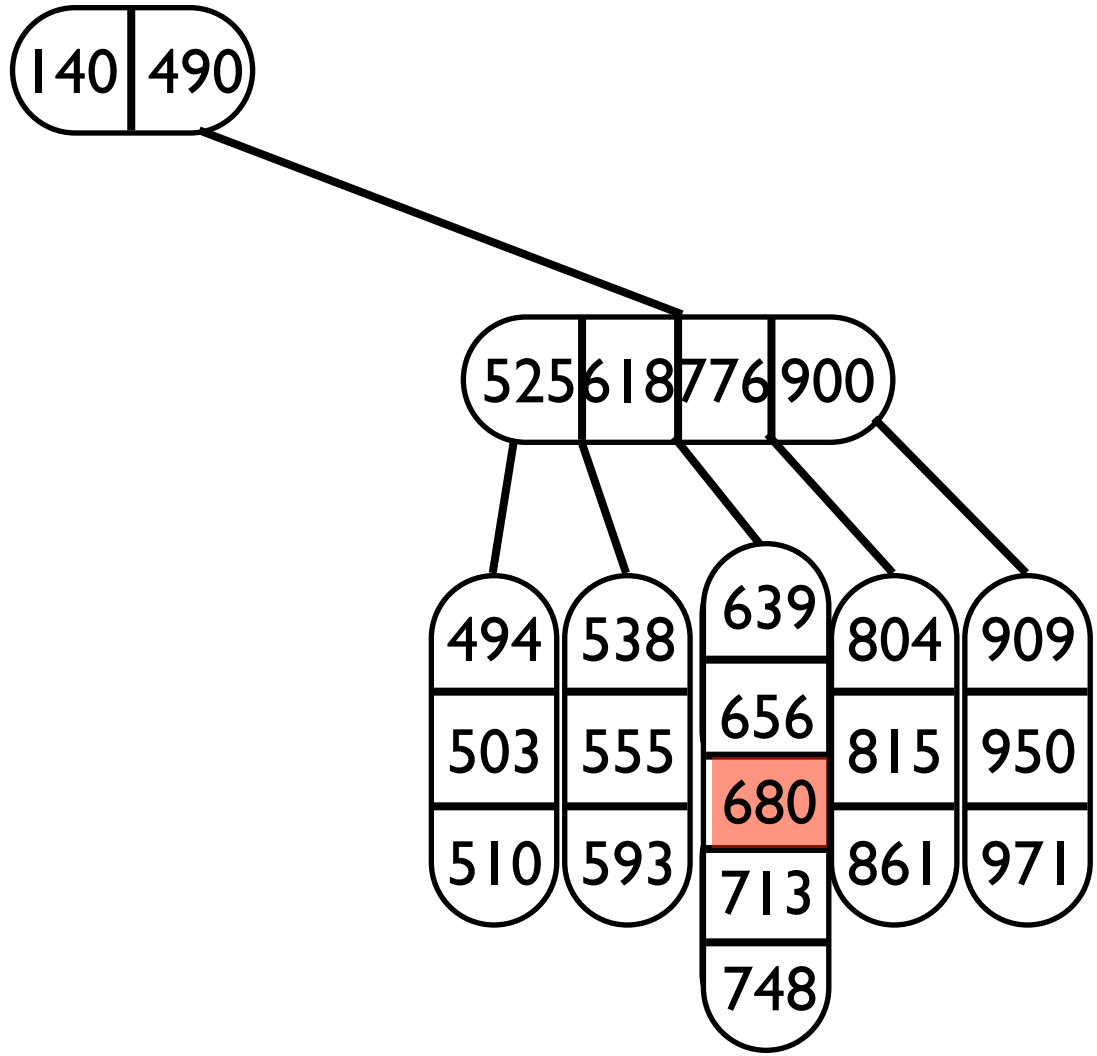
Suchen

B-Bäume - Beispiel: Einfügen 680



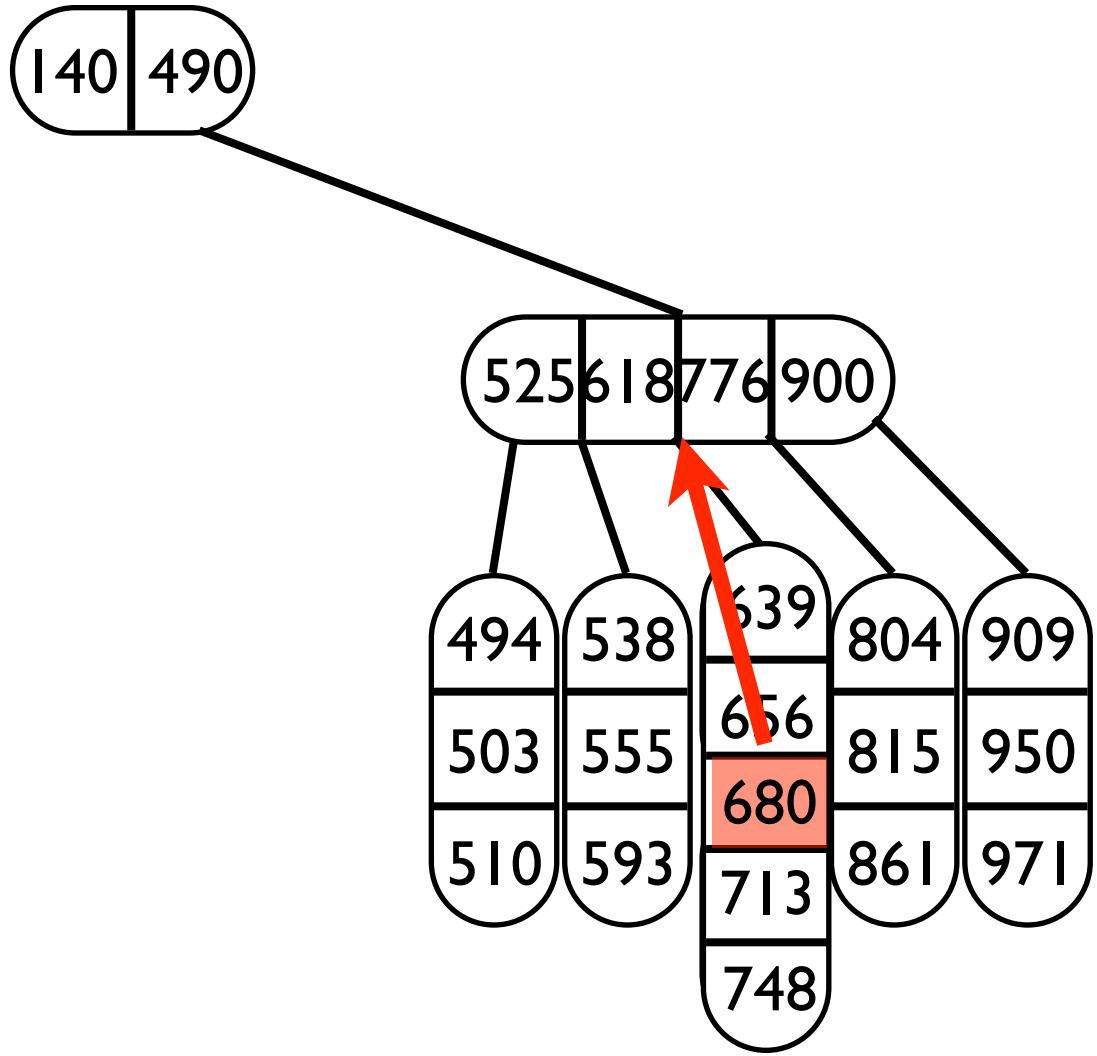
Suchen

B-Bäume - Beispiel: Einfügen 680



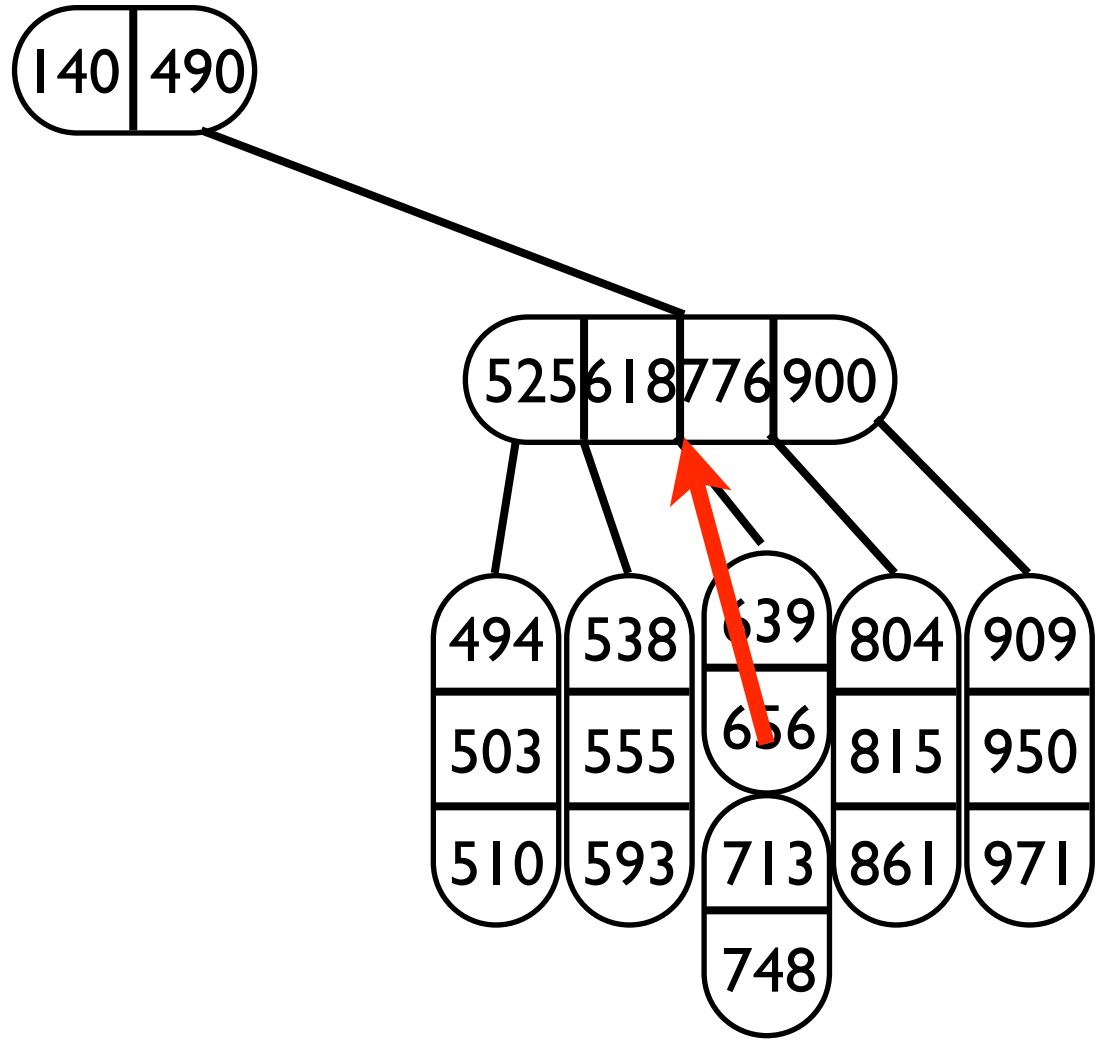
Suchen

B-Bäume - Beispiel: Einfügen 680



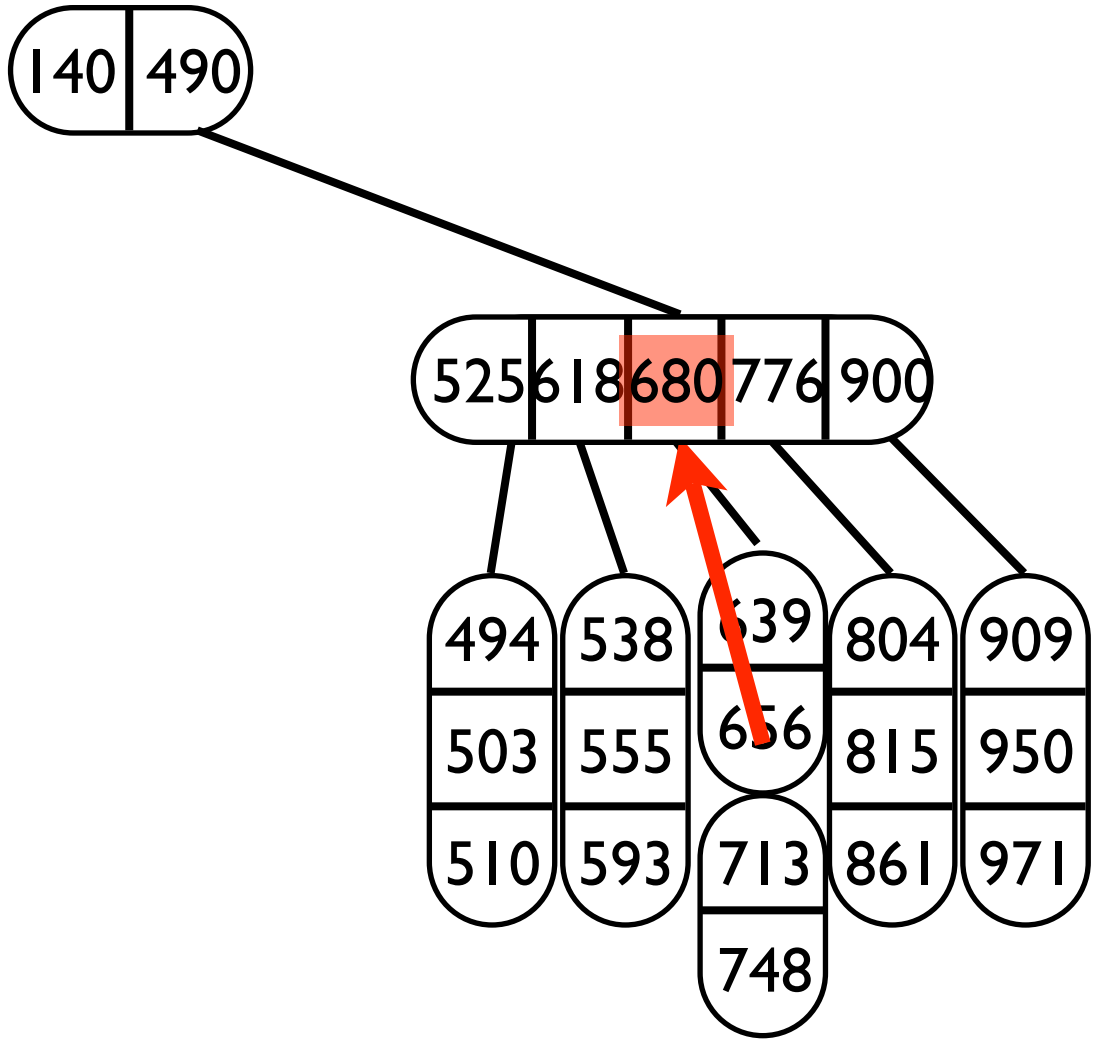
Suchen

B-Bäume - Beispiel: Einfügen 680



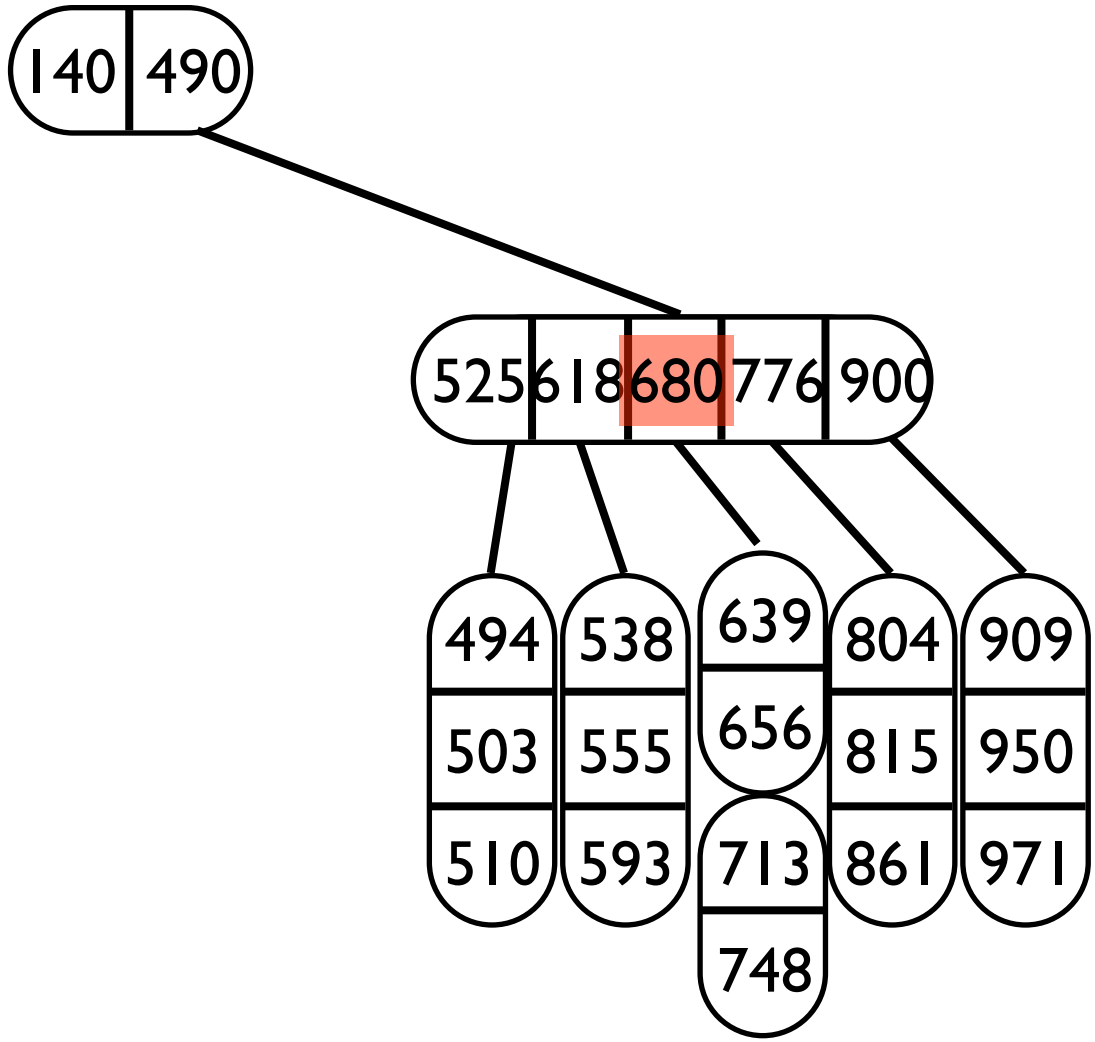
Suchen

B-Bäume - Beispiel: Einfügen 680



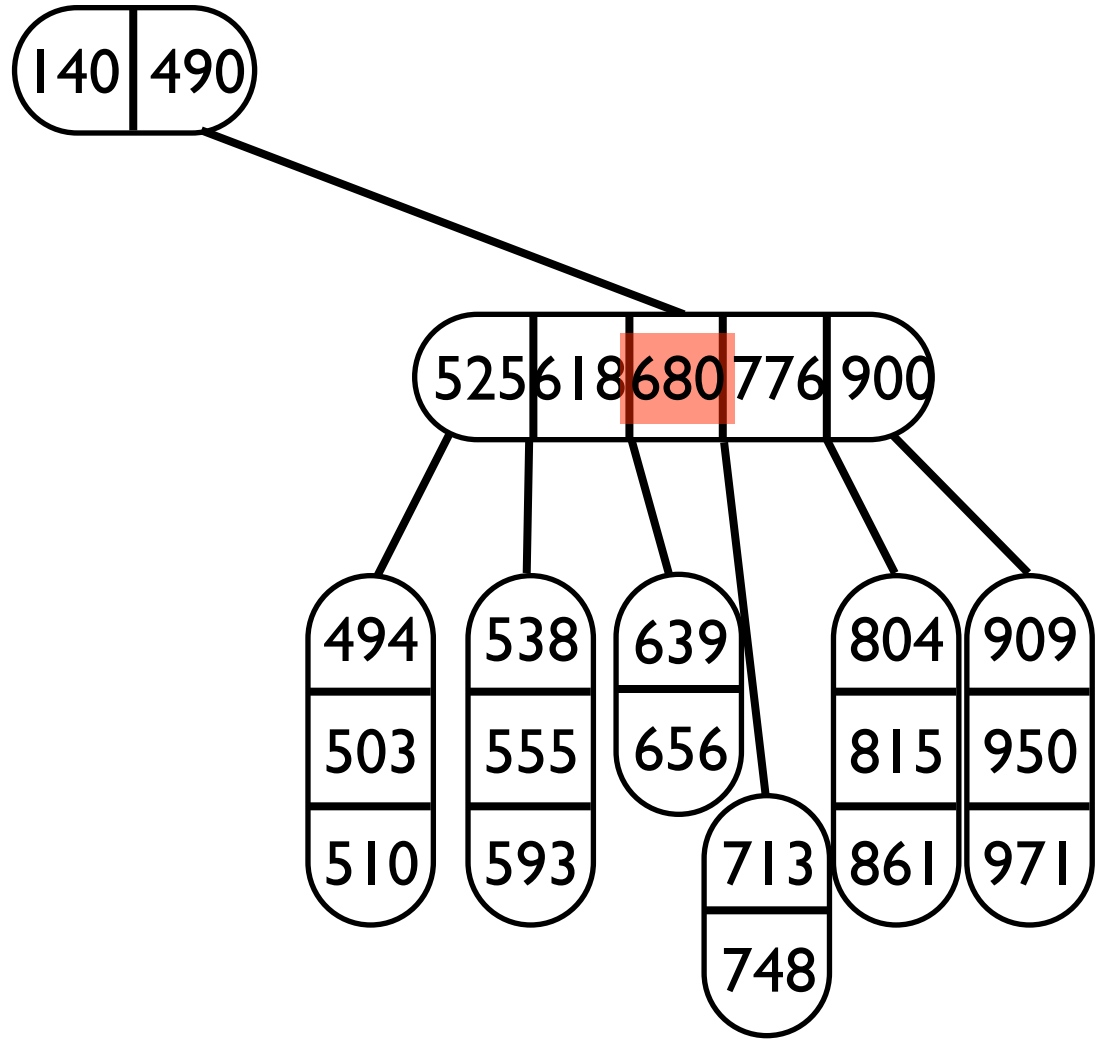
Suchen

B-Bäume - Beispiel: Einfügen 680



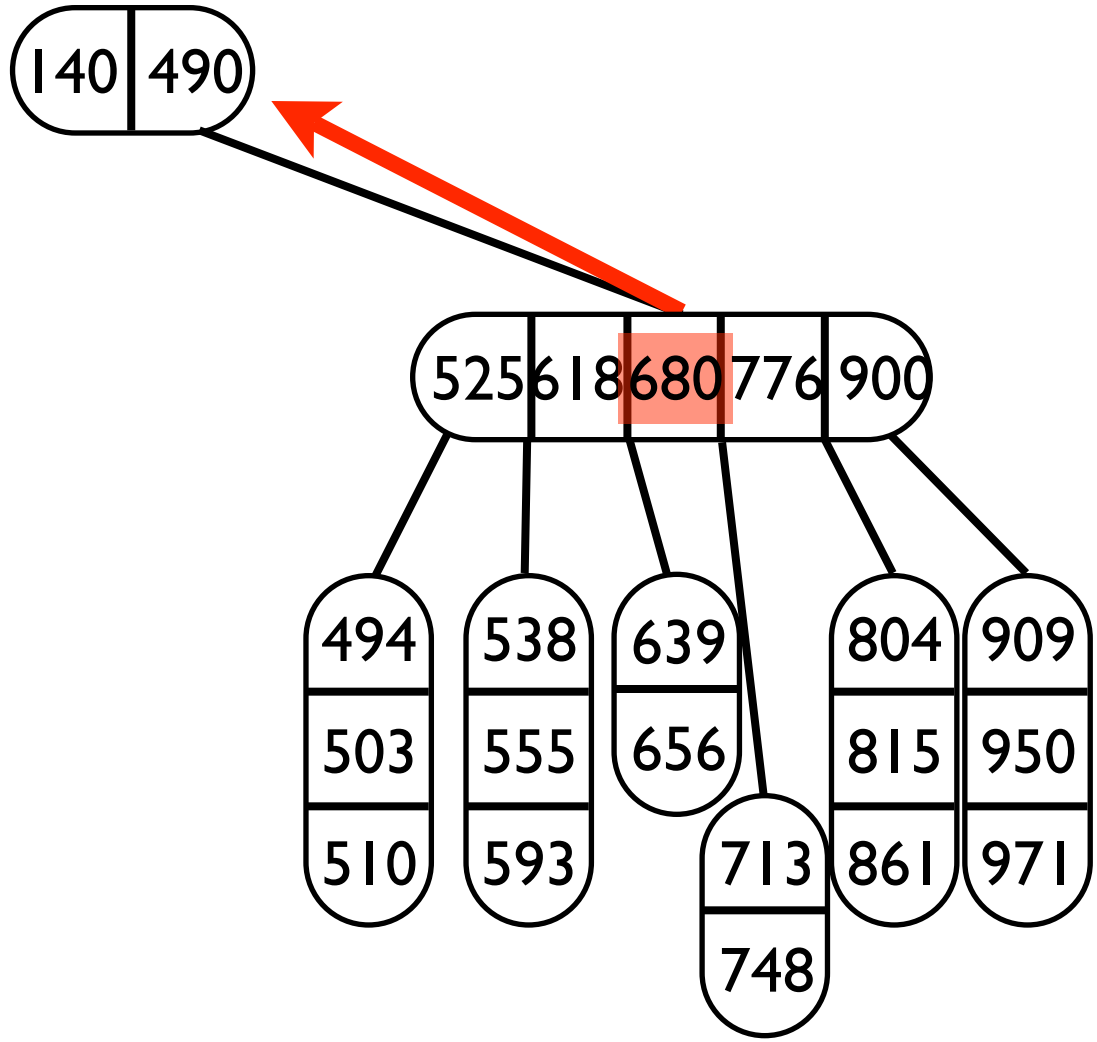
Suchen

B-Bäume - Beispiel: Einfügen 680



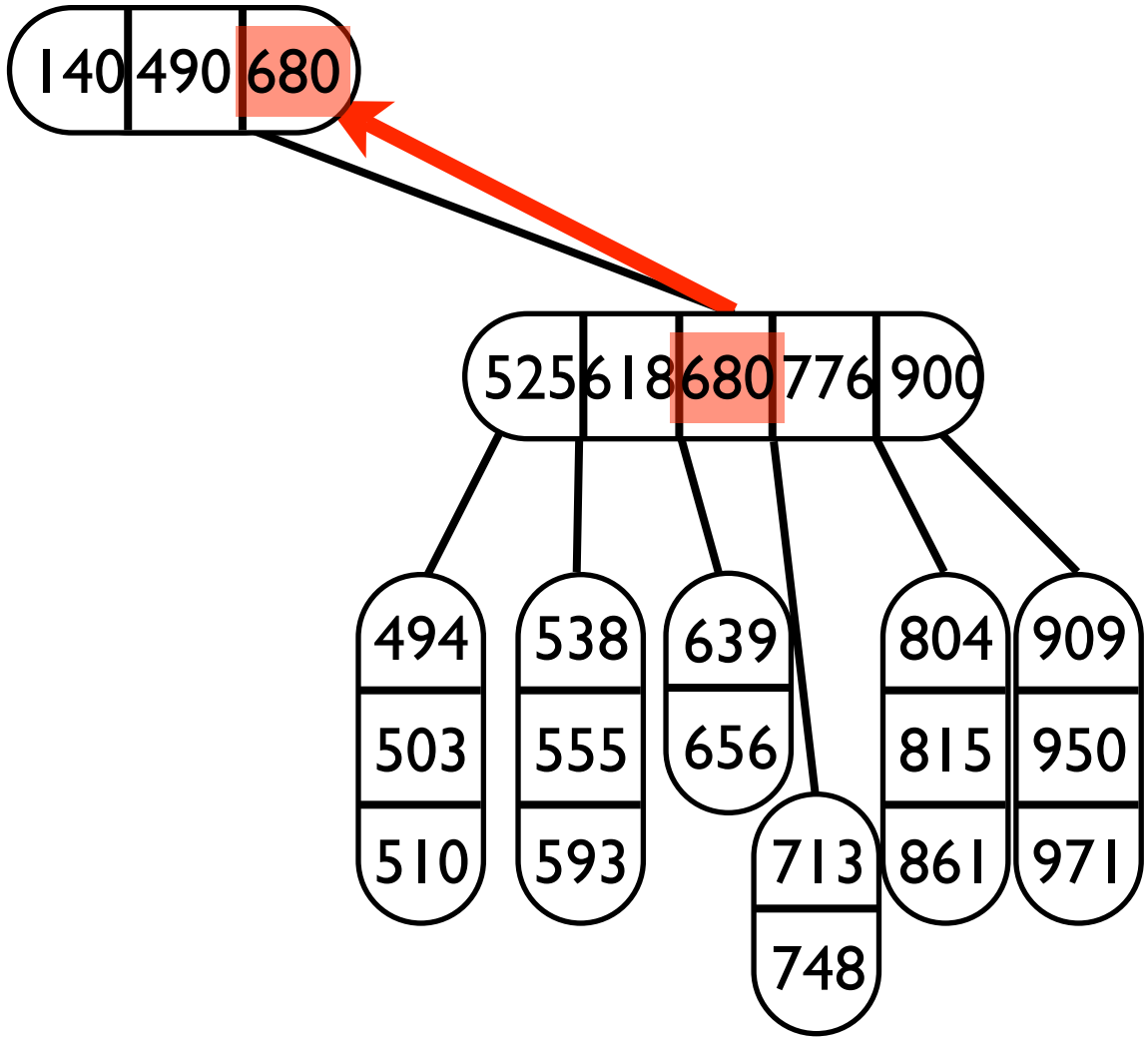
Suchen

B-Bäume - Beispiel: Einfügen 680



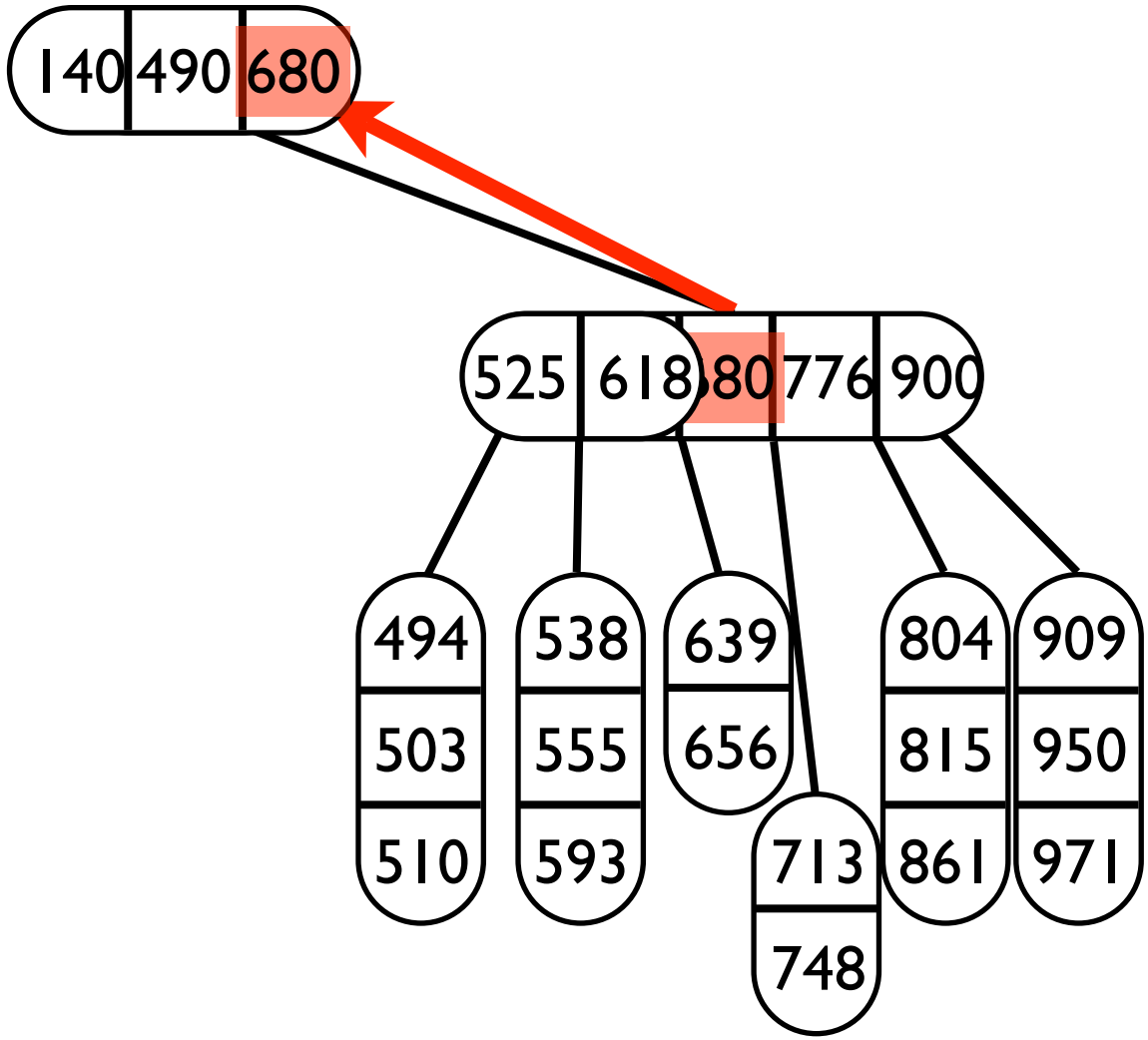
Suchen

B-Bäume - Beispiel: Einfügen 680



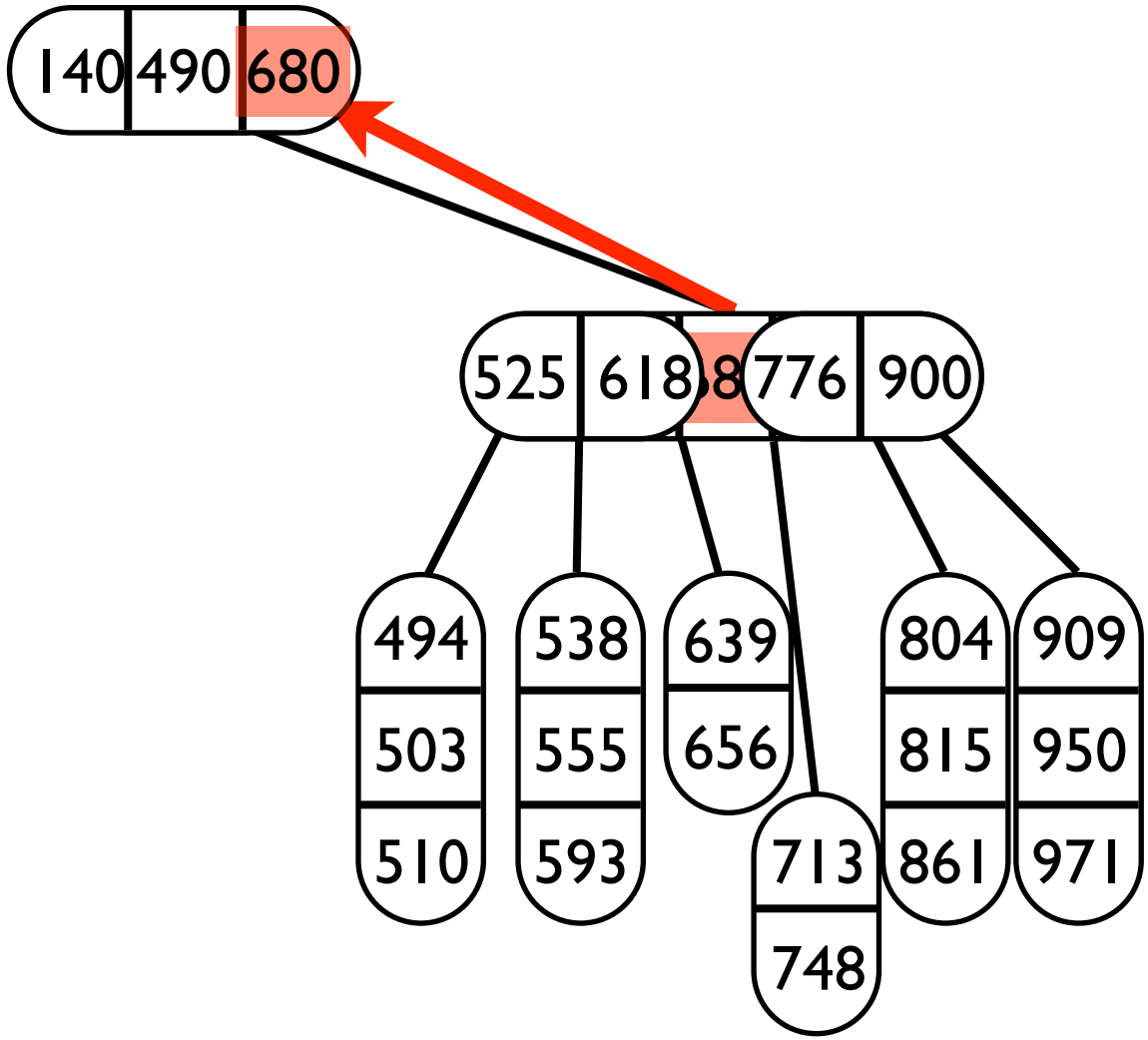
Suchen

B-Bäume - Beispiel: Einfügen 680



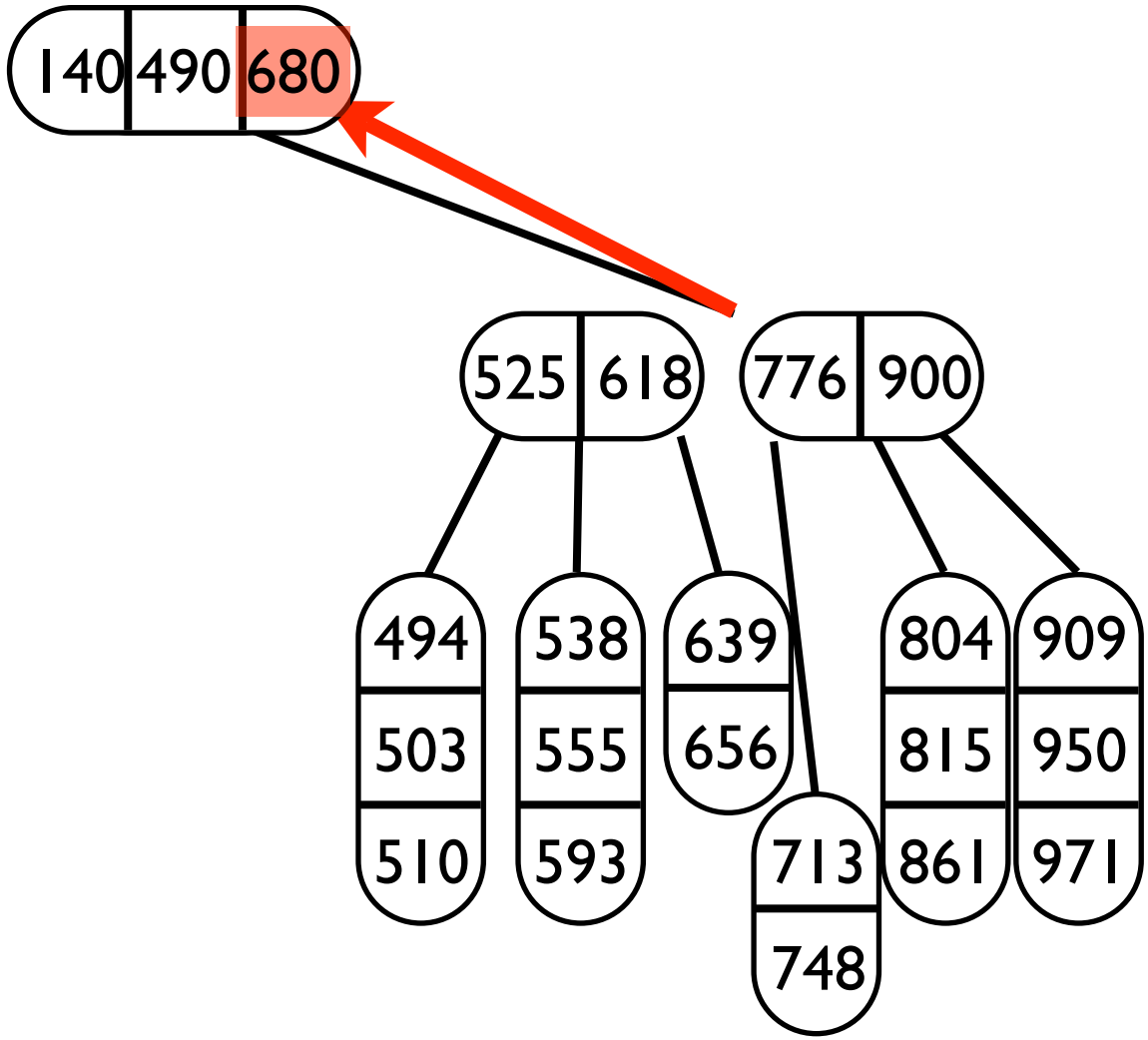
Suchen

B-Bäume - Beispiel: Einfügen 680



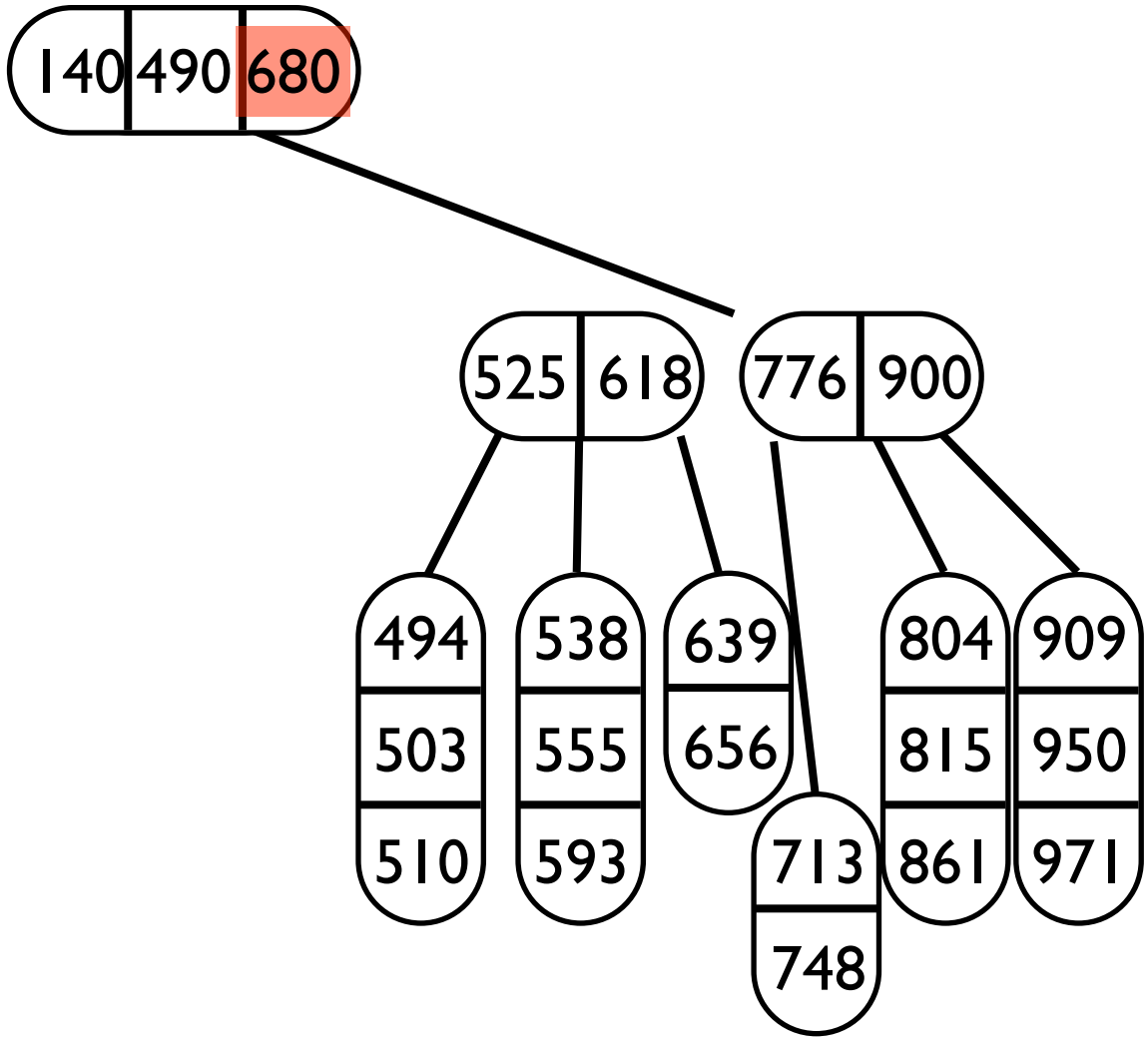
Suchen

B-Bäume - Beispiel: Einfügen 680



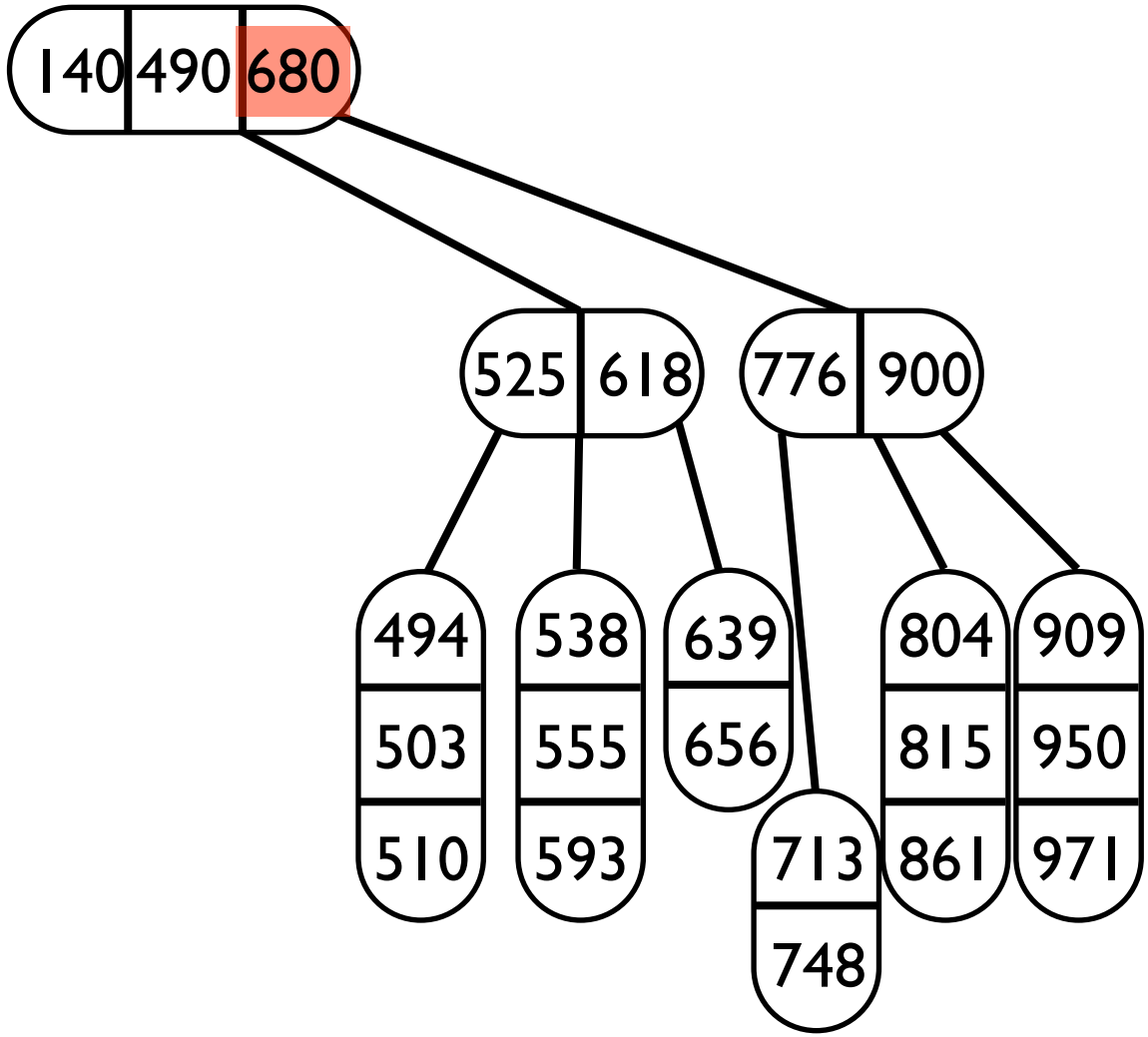
Suchen

B-Bäume - Beispiel: Einfügen 680



Suchen

B-Bäume - Beispiel: Einfügen 680



Suchen

B-Bäume - Laufzeitanalyse Suchen

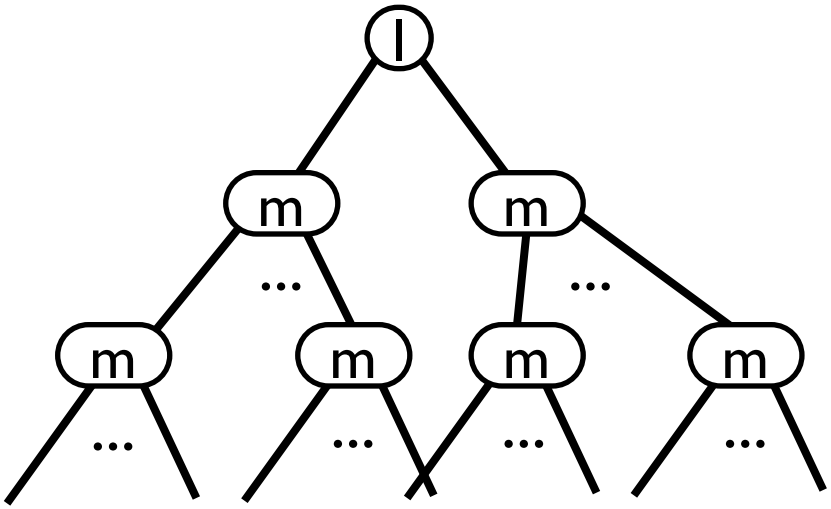
Gegeben: B-Baum der Ordnung m mit n Knoten.

Suchen: Auf wieviele Plattenblöcke (Superknoten) muß im worst-case zugegriffen werden?

Wie hoch sind B-Bäume der Ordnung m mit n Knoten im schlimmsten Fall?

Suchen

B-Bäume - Laufzeitanalyse Suchen

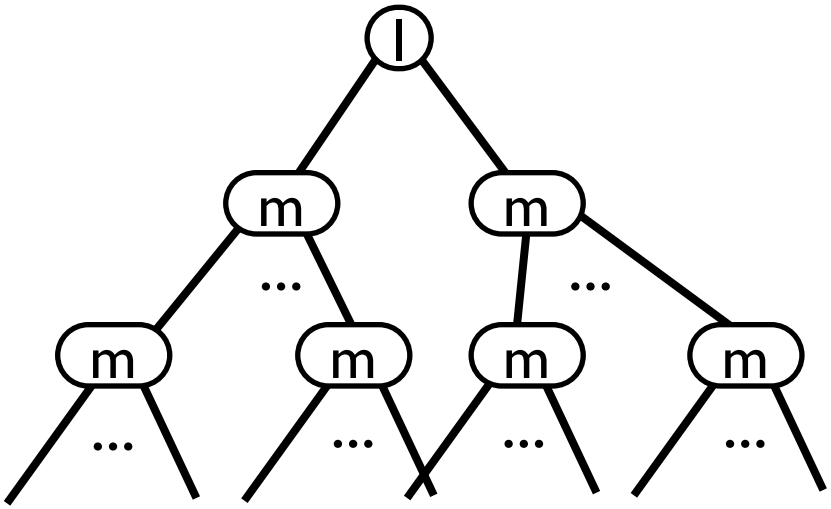


je m+1 Söhne!

Suchen

B-Bäume - Laufzeitanalyse Suchen

Schlimmster Fall:



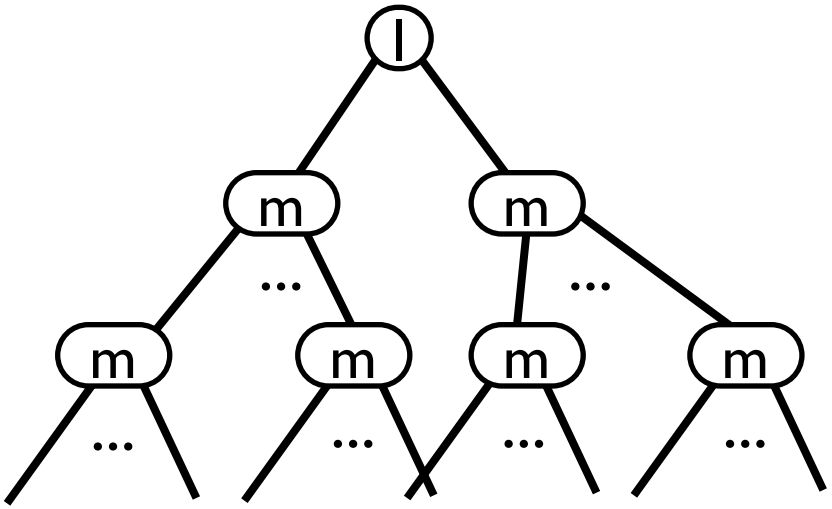
je m+1 Söhne!

Suchen

B-Bäume - Laufzeitanalyse Suchen

Schlimmster Fall:

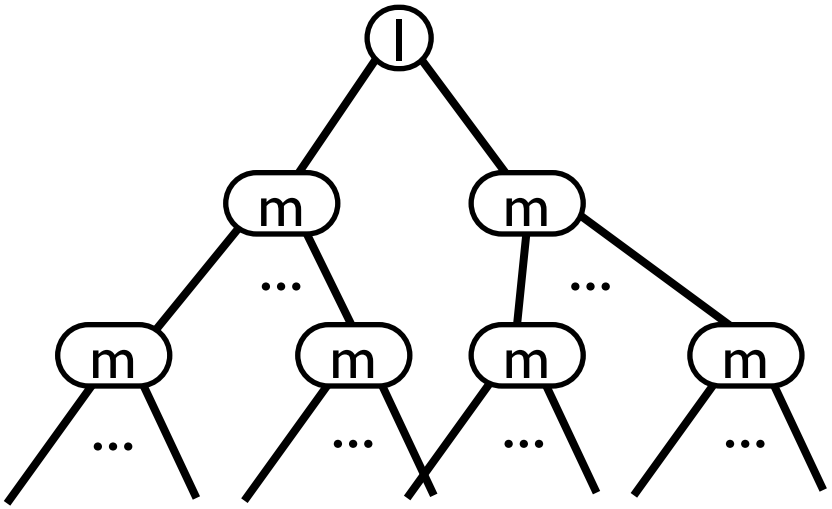
Baum so "dünn" besetzt wie möglich



je m+1 Söhne!

Suchen

B-Bäume - Laufzeitanalyse Suchen



je m+1 Söhne!

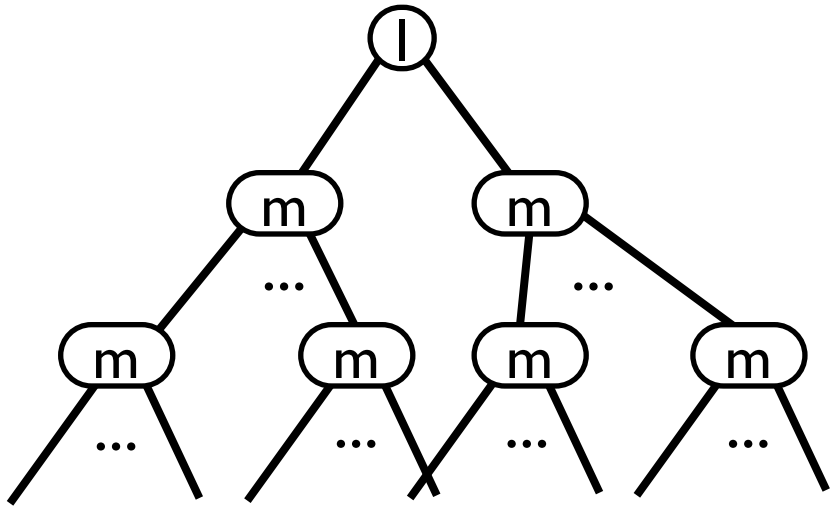
Schlimmster Fall:

Baum so "dünn" besetzt wie möglich

Niveau 0: ≥ 1 Schlüssel

Suchen

B-Bäume - Laufzeitanalyse Suchen



je m+1 Söhne!

Schlimmster Fall:

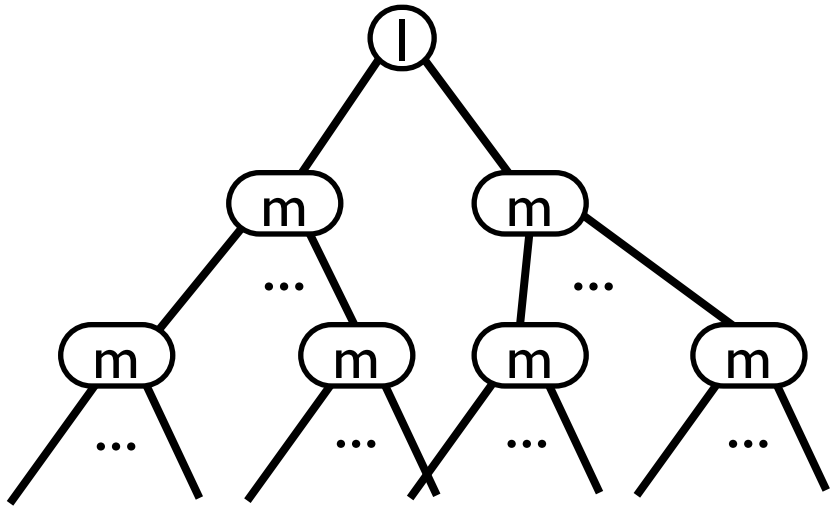
Baum so "dünn" besetzt wie möglich

Niveau 0: ≥ 1 Schlüssel

Niveau 1: $\geq 2m$ Schlüssel

Suchen

B-Bäume - Laufzeitanalyse Suchen



je m+1 Söhne!

Schlimmster Fall:

Baum so "dünn" besetzt wie möglich

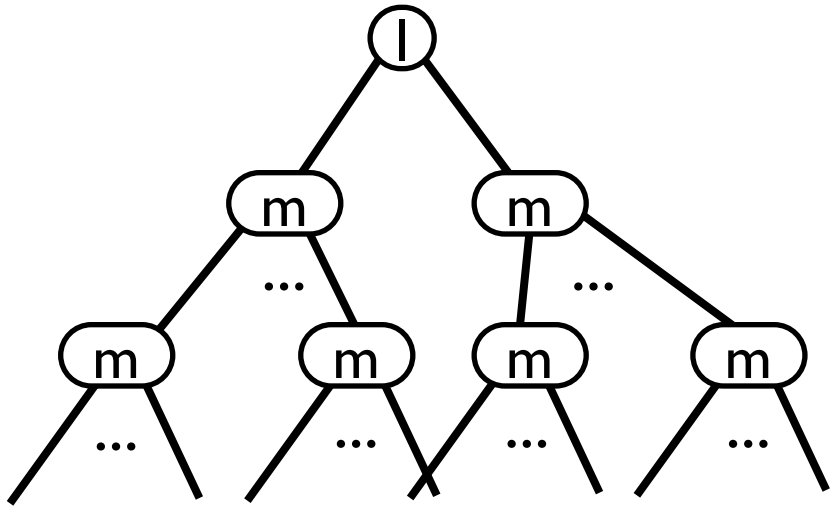
Niveau 0: ≥ 1 Schlüssel

Niveau 1: $\geq 2m$ Schlüssel

Niveau 2: $\geq 2m(m+1)$ Schlüssel

Suchen

B-Bäume - Laufzeitanalyse Suchen



je m+1 Söhne!

Schlimmster Fall:

Baum so "dünn" besetzt wie möglich

Niveau 0: ≥ 1 Schlüssel

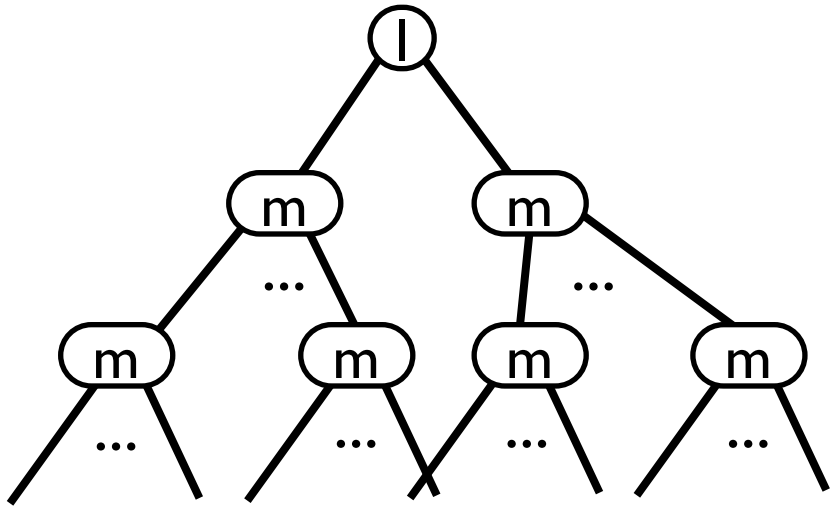
Niveau 1: $\geq 2m$ Schlüssel

Niveau 2: $\geq 2m(m+1)$ Schlüssel

Niveau 3: $\geq 2m(m+1)^2$ Schlüssel

Suchen

B-Bäume - Laufzeitanalyse Suchen



je m+1 Söhne!

Schlimmster Fall:

Baum so "dünn" besetzt wie möglich

Niveau 0: ≥ 1 Schlüssel

Niveau 1: $\geq 2m$ Schlüssel

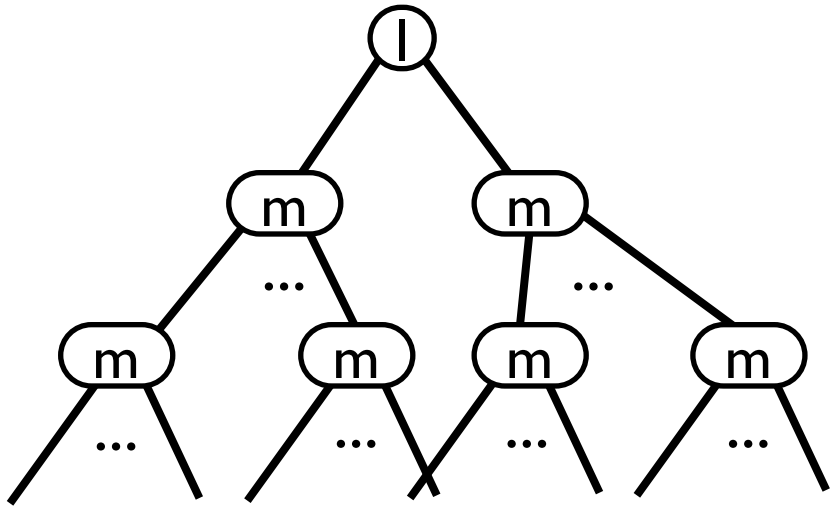
Niveau 2: $\geq 2m(m+1)$ Schlüssel

Niveau 3: $\geq 2m(m+1)^2$ Schlüssel

Niveau h+1: $\geq 2m(m+1)^h$ Schlüssel

Suchen

B-Bäume - Laufzeitanalyse Suchen



je m+1 Söhne!

Schlimmster Fall:

Baum so "dünn" besetzt wie möglich

Niveau 0: ≥ 1 Schlüssel

Niveau 1: $\geq 2m$ Schlüssel

Niveau 2: $\geq 2m(m+1)$ Schlüssel

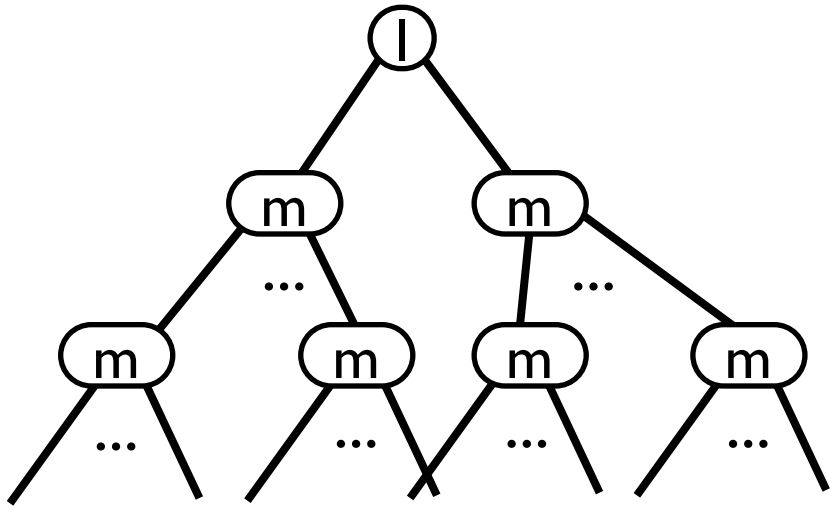
Niveau 3: $\geq 2m(m+1)^2$ Schlüssel

Niveau h+1: $\geq 2m(m+1)^h$

~~Schlüssel~~

Suchen

B-Bäume - Laufzeitanalyse Suchen



je m+1 Söhne!

Schlimmster Fall:

Baum so "dünn" besetzt wie möglich

Niveau 0: ≥ 1 Schlüssel

Niveau 1: $\geq 2m$ Schlüssel

Niveau 2: $\geq 2m(m+1)$ Schlüssel

Niveau 3: $\geq 2m(m+1)^2$ Schlüssel

Niveau h+1: $\geq 2m(m+1)^h$

~~Schlüssel~~

$1+2m\Sigma_{j=0}^{h}(m+1)^j$ Schlüssel

Suchen

B-Bäume - Laufzeitanalyse Suchen

$$\ldots = 1+2m\sum_{j=0}^{h}(m+1)^j = 1+2m\,\frac{(m+1)^{h+1}-1}{(m+1)-1}$$

$$= 1+2(m+1)^{h+1}-1 = 2(m+1)^{h+1} \text{ Schlüssel}$$

Sind nun n Schlüssel im B-Baum gespeichert, so gilt:

$$2(m+1)^{h+1} \leq n$$

$$h \leq \log_{m+1}(n/2)-1 = O(\log_{m+1} n)$$

Folglich ist die Suchzeit wieder logarithmisch.

Praxis: m=200 bis 2000 (hardwareabhängig)

Suchen
B-Bäume - Laufzeitanalyse Aufspalten

Aufspalten: ... ist zeitraubend. Wie oft kommen Aufspaltungen von Knoten vor?

Seien:

s: mittlere Anzahl der Aufspaltungen pro Einfügung

t: Gesamtzahl der Aufspaltungen bei Einfügung der n Schlüssel $k_1, ..., k_n$ in leeren B-Baum

Offenbar: s=t/n (Mittelwert)

Suchen
B-Bäume - Laufzeitanalyse Aufspalten

Nach t Aufspaltungen gibt es in dem Baum anschließend offenbar $p=t+1$ Knoten.

Wieviele Schlüssel n_p befinden sich in einem Baum mit p Knoten mindestens:

$$n_p = 1 + (p-1) \cdot m \leq n$$

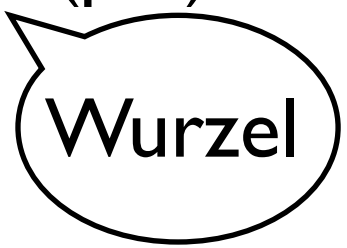
Suchen

B-Bäume - Laufzeitanalyse Aufspalten

Nach t Aufspaltungen gibt es in dem Baum anschließend offenbar $p=t+1$ Knoten.

Wieviele Schlüssel n_p befinden sich in einem Baum mit p Knoten mindestens:

$$n_p = 1 + (p-1) \cdot m \leq n$$



Suchen
B-Bäume - Laufzeitanalyse Aufspalten

Nach t Aufspaltungen gibt es in dem Baum anschließend offenbar $p=t+1$ Knoten.

Wieviele Schlüssel n_p befinden sich in einem Baum mit p Knoten mindestens:

$$n_p = 1 + (p-1) \cdot m \leq n$$

Suchen

B-Bäume - Laufzeitanalyse Aufspalten

Nach t Aufspaltungen gibt es in dem Baum anschließend offenbar p=t+1 Knoten.

Wieviele Schlüssel n_p befind[illegible] in einem Baum mit p Knoten minde[illegible]

übrige Knoten

$$n_p = 1 + (p-1) \cdot m \leq n$$

Suchen
B-Bäume - Laufzeitanalyse Aufspalten

Nach t Aufspaltungen gibt es in dem Baum anschließend offenbar $p=t+1$ Knoten.

Wieviele Schlüssel n_p befinden sich in einem Baum mit p Knoten mindestens:

$$n_p = 1 + (p-1) \cdot m \leq n$$

Suchen

B-Bäume - Laufzeitanalyse Aufspalten

Nach t Aufspaltungen gibt es in dem Baum anschließend offenbar p=t+1 Knoten.

Wieviele Schlüssel n_p befinden sich in einem Baum mit p Knoten mindestens?

m
Ordnung

$$n_p = 1 + (p-1) \cdot m \leq n$$

Suchen
B-Bäume - Laufzeitanalyse Aufspalten

Nach t Aufspaltungen gibt es in dem Baum anschließend offenbar $p=t+1$ Knoten.

Wieviele Schlüssel n_p befinden sich in einem Baum mit p Knoten mindestens:

$$n_p = 1 + (p-1) \cdot m \leq n$$

Suchen

B-Bäume - Laufzeitanalyse Aufspalten

Zusammenstellung:

$$s = \frac{t}{n}$$

$$p = t + 1$$

$$n_p = 1 + (p-1) \cdot m \leq n \Rightarrow p \leq 1 + \frac{n-1}{m}$$

$$\Rightarrow s = \frac{t}{n} = \frac{p-1}{n} \leq \frac{1}{n} \cdot \frac{n-1}{m} < \frac{1}{m}$$

$\Rightarrow$ Aufspaltungen von Knoten relativ selten

(bei $200 \leq m \leq 2000$)

Suchen
B-Bäume - Laufzeitanalyse Ergebnis

Satz

Suchen, Einfügen, Löschen in B-Bäumen der Ordnung m ist implementierbar mit $O(\log_m n)$ Plattenzugriffen.

Suchen
B-Bäume - Schlußbeispiel

- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17

Suchen
B-Bäume - Schlußbeispiel

- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17

Suchen
B-Bäume - Schlußbeispiel

- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17

50	102

Suchen
B-Bäume - Schlußbeispiel

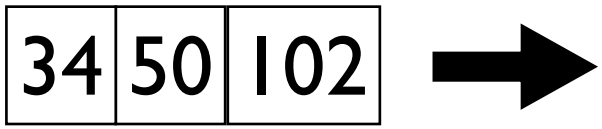
- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17

34	50	102

Suchen

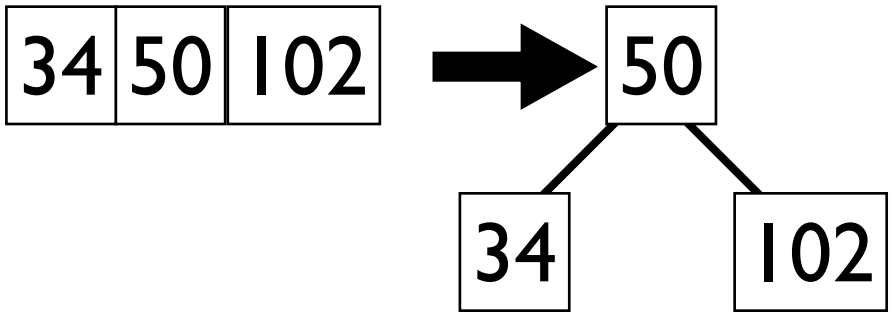
B-Bäume - Schlußbeispiel

- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17



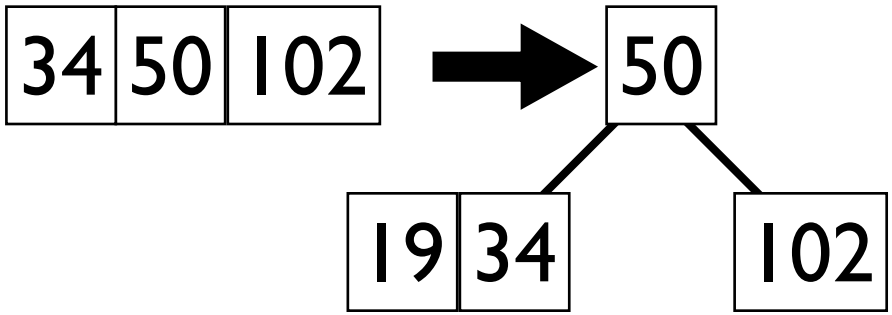
Suchen
B-Bäume - Schlußbeispiel

- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17



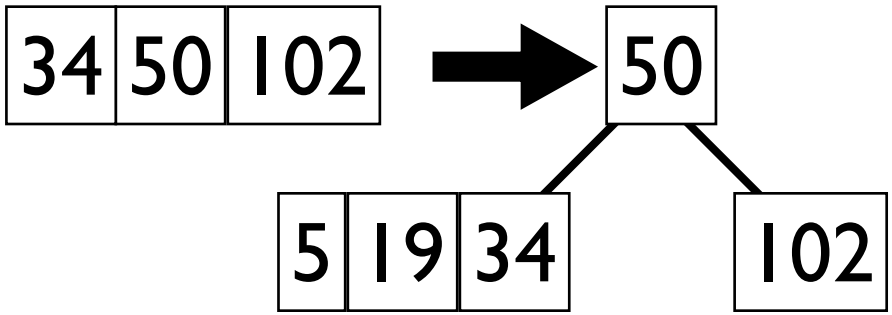
Suchen
B-Bäume - Schlußbeispiel

- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17



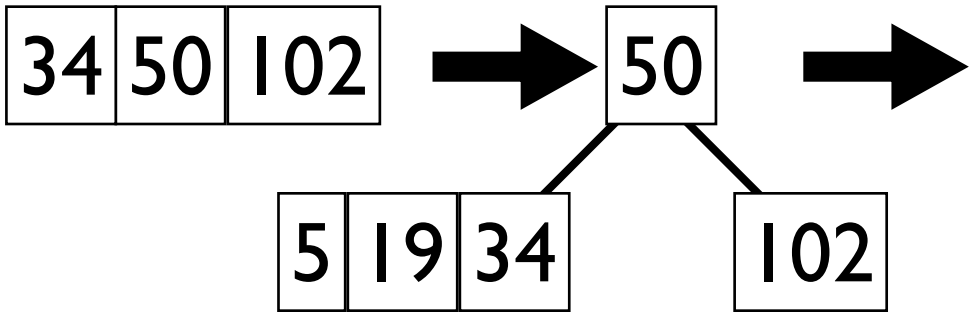
Suchen
B-Bäume - Schlußbeispiel

- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17



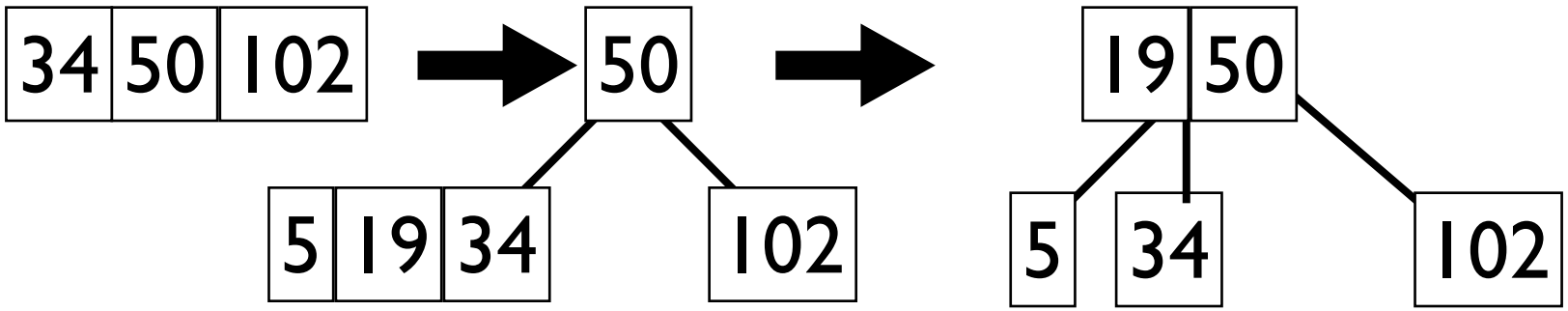
Suchen
B-Bäume - Schlußbeispiel

- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17



Suchen
B-Bäume - Schlußbeispiel

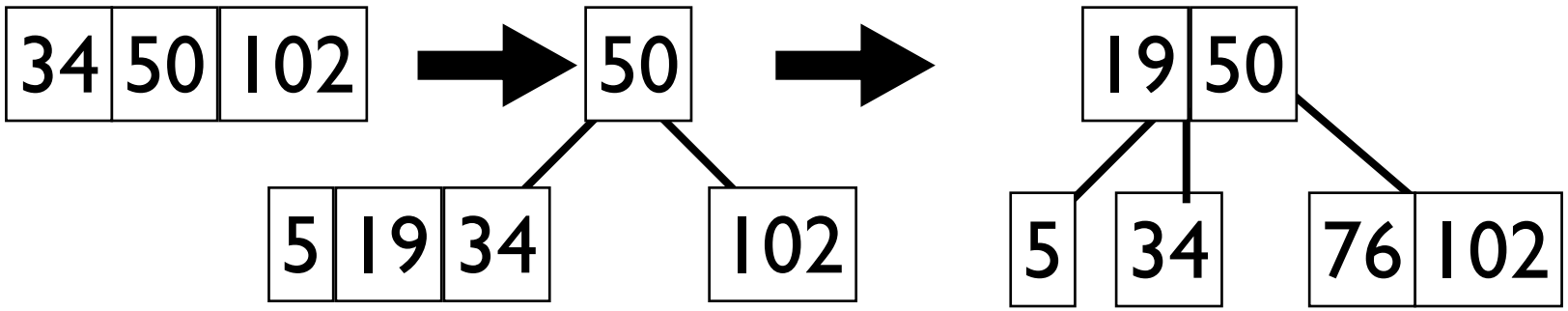
- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17



Suchen

B-Bäume - Schlußbeispiel

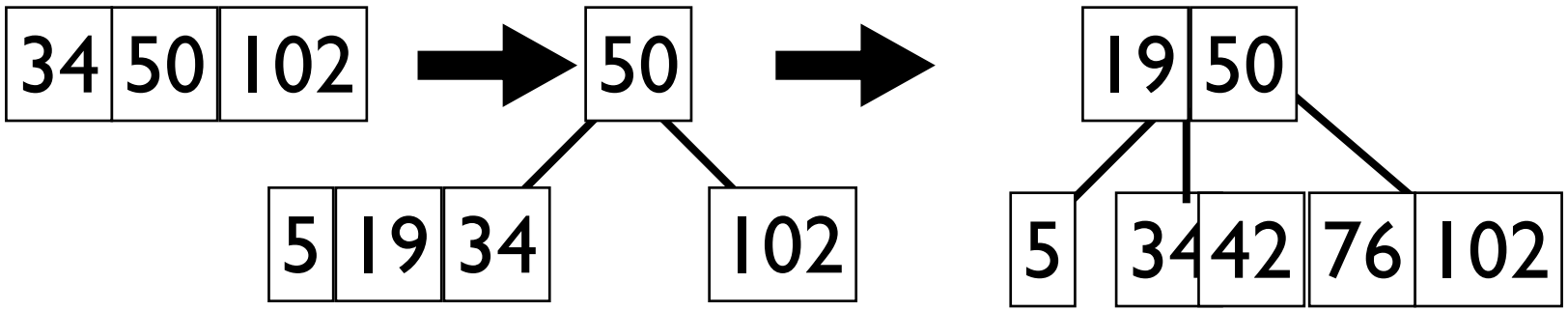
- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17



Suchen

B-Bäume - Schlußbeispiel

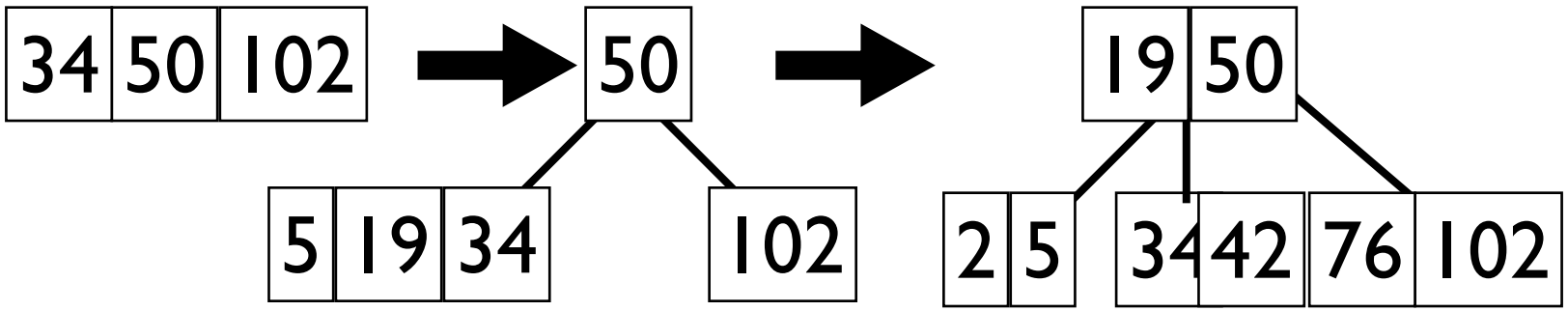
- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17



Suchen

B-Bäume - Schlußbeispiel

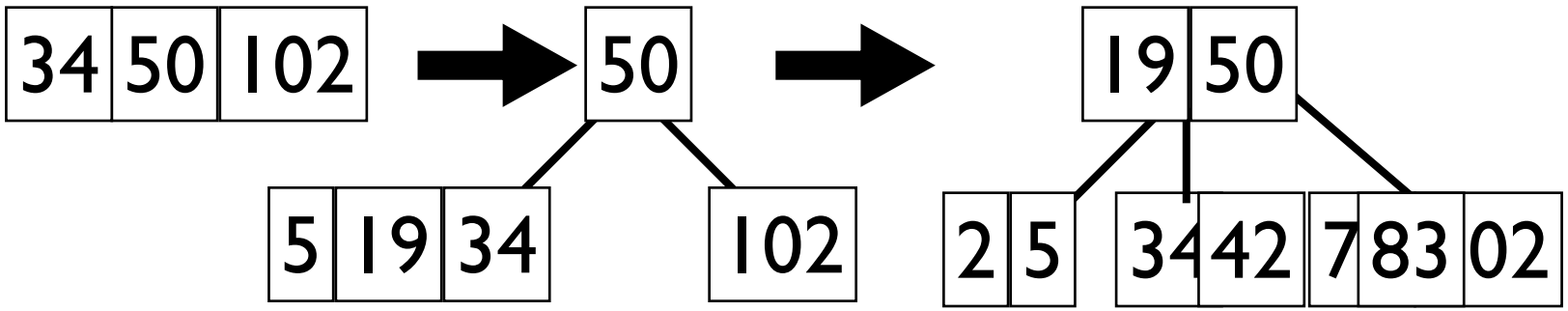
- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17



Suchen

B-Bäume - Schlußbeispiel

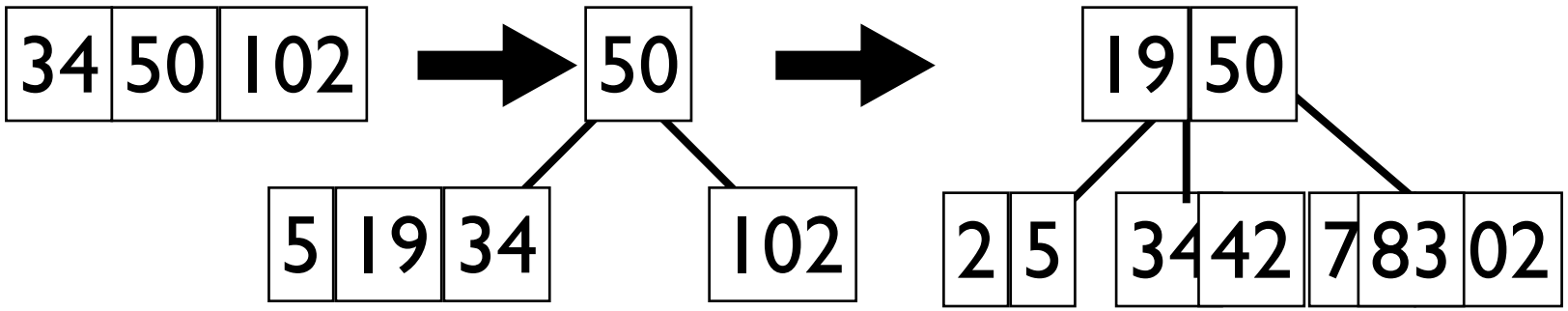
- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17

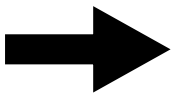


Suchen

B-Bäume - Schlußbeispiel

- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17

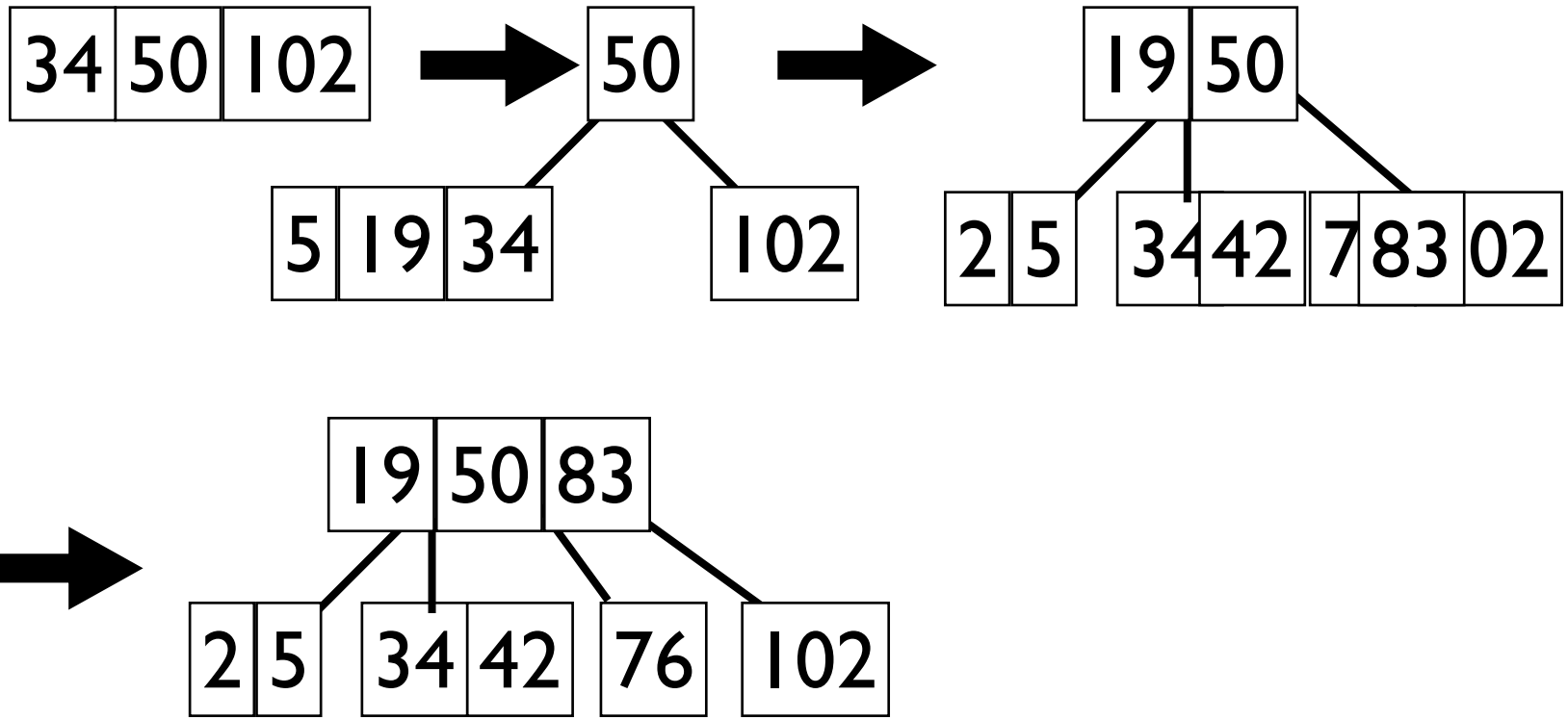




Suchen

B-Bäume - Schlußbeispiel

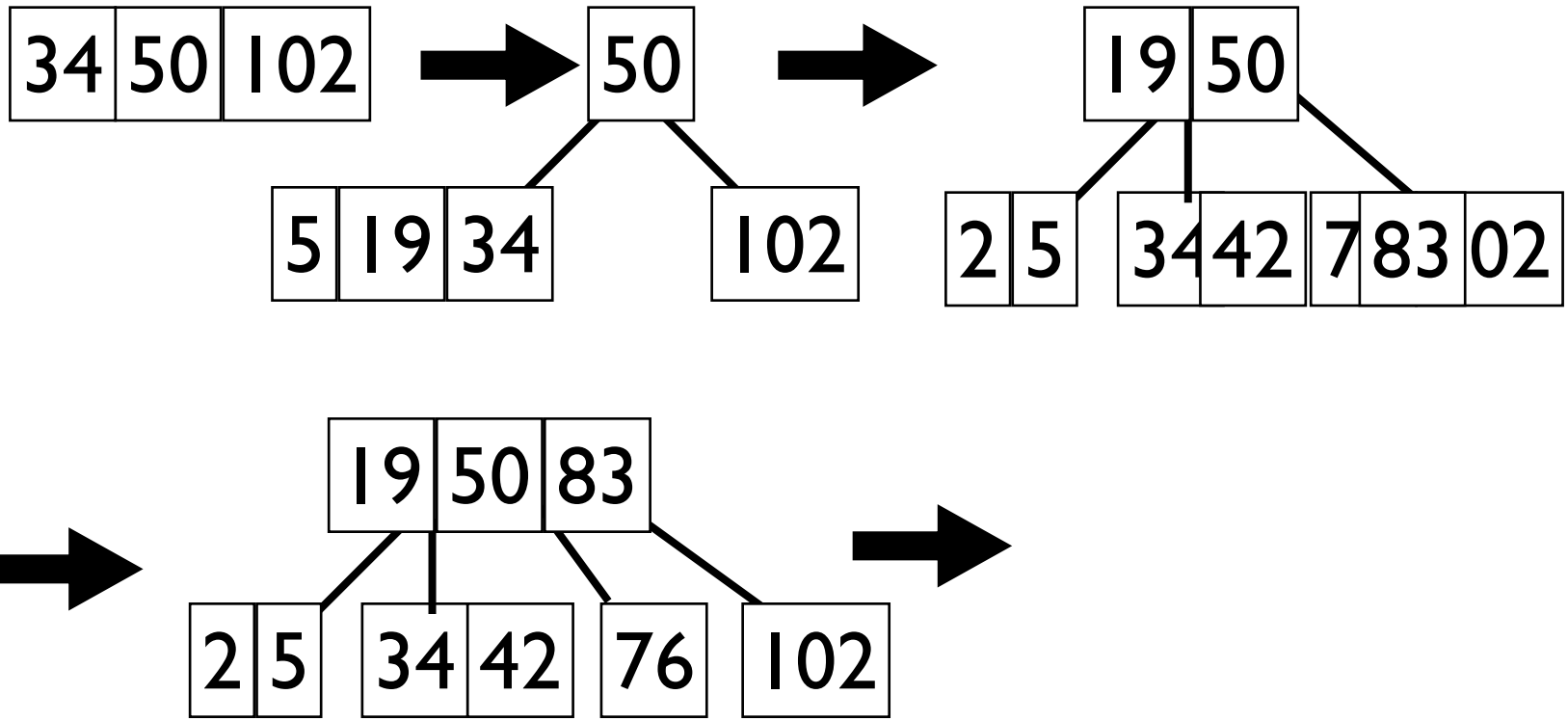
- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17



Suchen

B-Bäume - Schlußbeispiel

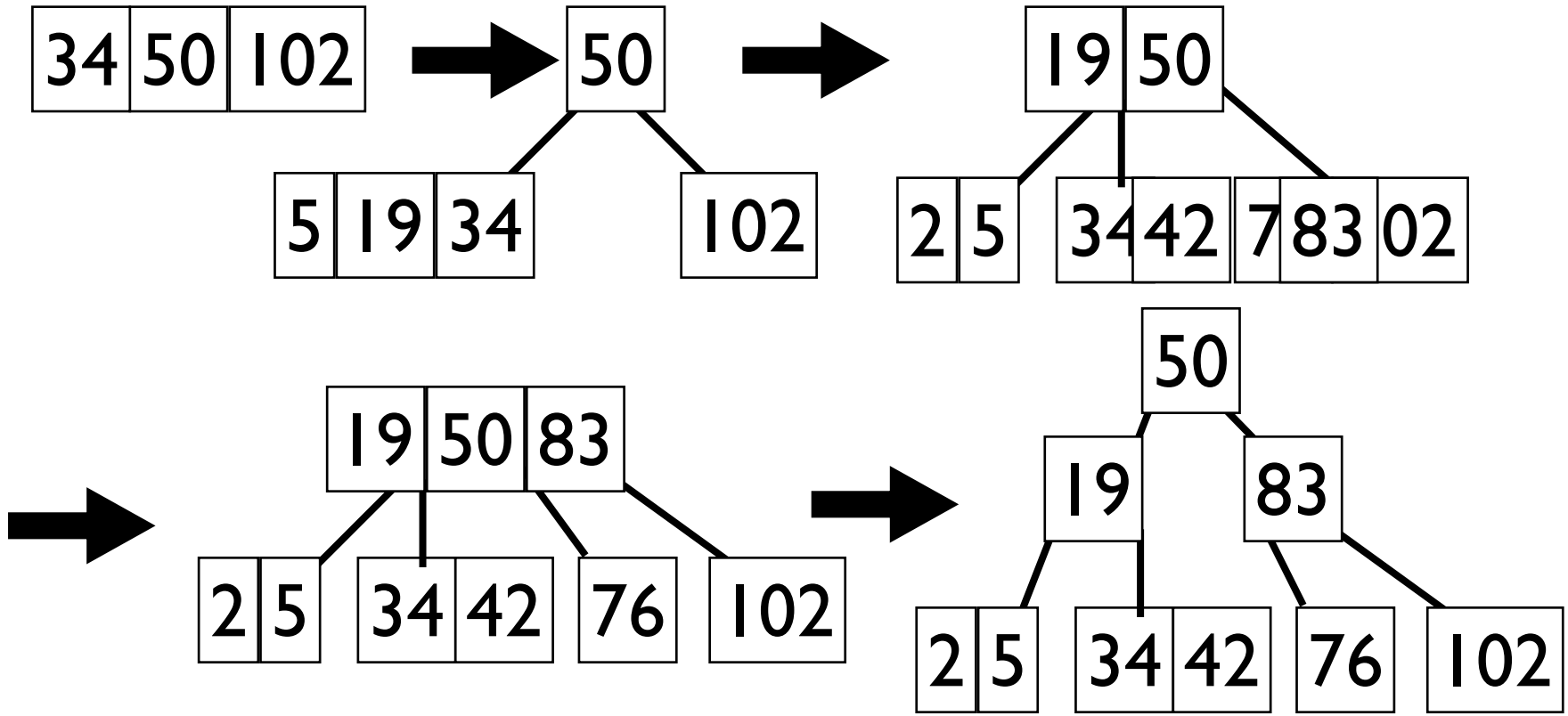
- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17



Suchen

B-Bäume - Schlußbeispiel

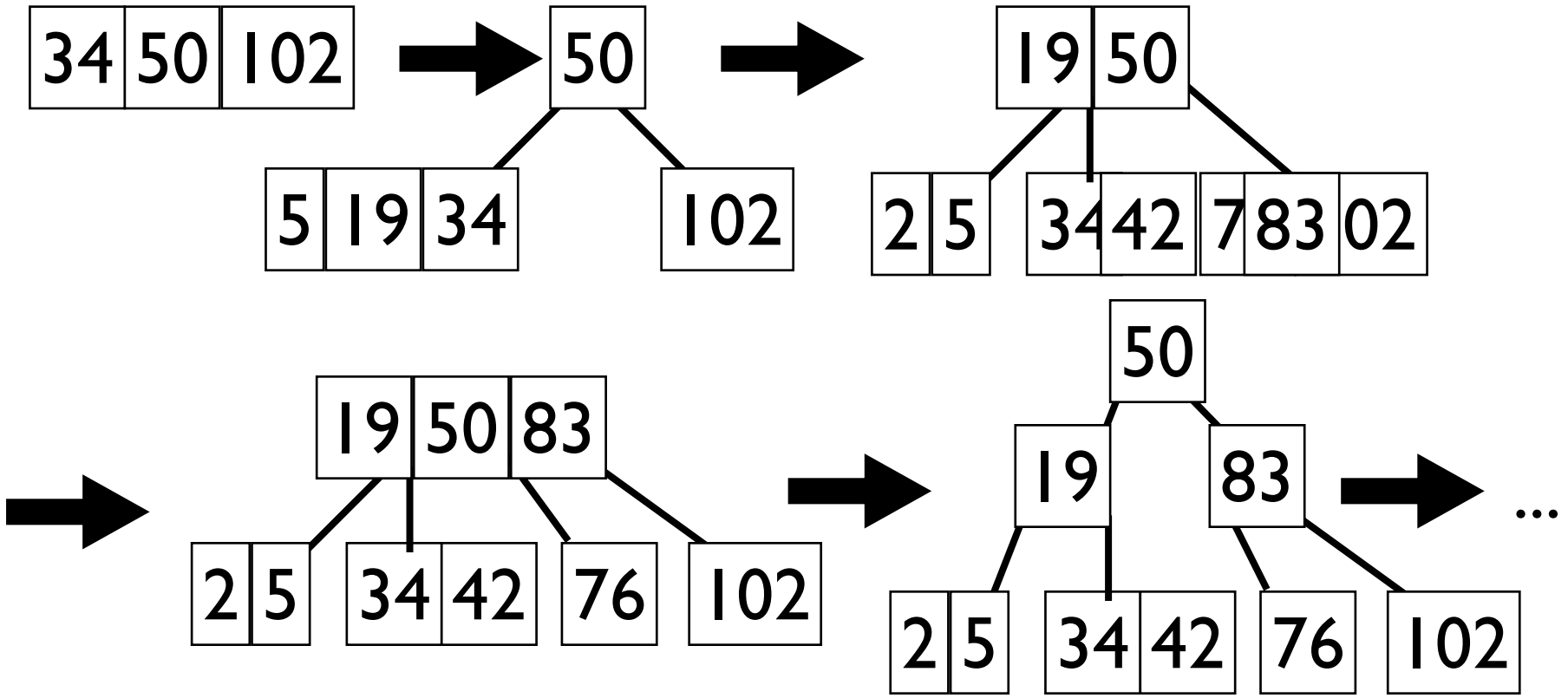
- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17



Suchen

B-Bäume - Schlußbeispiel

- Aufbau eines B-Baums (m=1) mit der Folge 50,102,34,19,5,76,42,2,83,59,70,12,17

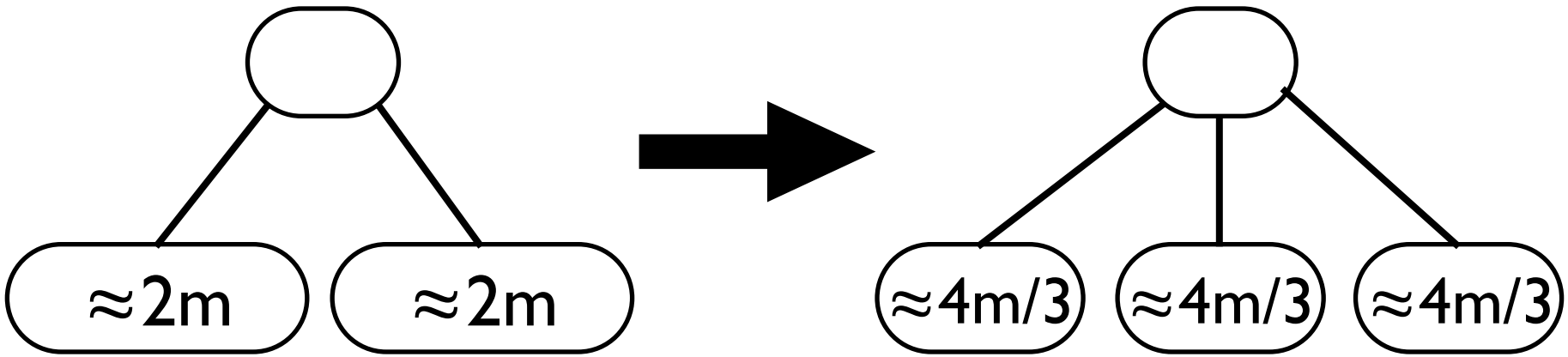


Suchen
B-Bäume - Optimierung

- Aufspalten eines Knotens → geringe Speicherausnutzung
- besser: Umverteilung der Schlüssel auf Bruderknoten (wie beim Löschen)
- erst wenn Bruderknoten voll ist, wird aufgespalten (sog. **B*-Bäume**).
- Garantie: Ausnutzung von mind. 2/3 der Kapazität jedes Knotens (Ausnahme: Wurzel)

Suchen

B-Bäume - B*-Bäume



Suchen
Hashing - Motivation

- Bisher:
  - Gegeben Schlüssel s
  - systematische Suche nach s im Datenbestand
- Idee:
  - Errechne aus s die Position, an der sich das zugehörige Objekt im Speicher befindet

Suchen
Hashing - Motivation

- Genauer:
  - K: Menge von Schlüsseln
  - A: Menge von Adressen, unter denen Objekte abgespeichert sind
- Eine Funktion

$$h: K \rightarrow A$$

  heißt **Hash-Funktion**
- Suche nach Objekt mit Schlüssel k → berechne h(k)
- Ziel: Beschreibe h mit möglichst einfachen arithmetischen Operationen

Suchen
Hashing - Beispiel

- Schlüsselmenge K_1={Januar, ..., Dezember}
- Adreßraum A={0, ..., 16}
- f: {A, ..., Z}→{1, ..., 26} ordne jedem Buchstaben seine Position im Alphabet zu
- Definiere: $h_1: K_1 \rightarrow A$ durch

h_1(k)=(f(1. Buchst. von k)+f(2. Buchst. von k)) mod 17

Suchen
Hashing - Beispiel

- Schlüsselmenge K_1={Januar, ..., Dezember}
- Adreßraum A={0, ..., 16}
- f: {A, ..., Z}→{1, ..., 26} ordne jedem Buchstaben seine Position im Alphabet zu
- Definiere: h_1: K_1 → A durch

$h_1(k)$=(f(1. Buchst. von k)+f(2. Buchst. von k)) mod 17

Monat	J	F	M	A	M	J	J	A	S	O	N	D
h_1	11	11	14	0	14	14	14	5	7	9	12	9

Suchen
Hashing - Beispiel

- Schlüsselmenge K_1={Januar, ..., Dezember}
- Adreßraum A={0, ..., 16}
- f: {A, ..., Z}→{1, ..., 26} ordne jedem Buchstaben seine Position im Alphabet zu
- Definiere: h_1: K_1 → A durch

$h_1(k)=(f(1.\text{ Buchst. von } k)+f(2.\text{ Buchst. von } k)) \bmod 17$

Monat	J	F	M	A	M	J	J	A	S	O	N	D
h_1	11	11	14	0	14	14	14	5	7	9	12	9

Kollisionen

Suchen
Hashing - Beispiel

- Schlüsselmenge K_1={Januar, ..., Dezember}
- Adreßraum A={0, ..., 16}
- f: {A, ..., Z}→{1, ..., 26} ordne jedem Buchstaben seine Position im Alphabet zu
- Definiere: h_1: K_1 → A durch

$h_1(k)$=(f(1. Buchst. von k)+f(2. Buchst. von k)) mod 17

Monat	J	F	M	A	M	J	J	A	S	O	N	D
h_1	11	11	14	0	14	14	14	5	7	9	12	9

Suchen
Hashing - Beispiel

- Schlüsselmenge K_1={Januar, ..., Dezember}
- Adreßraum A={0, ..., 16}
- f: {A, ..., Z}→{1, ..., 26} ordne jedem Buchstaben seine Position im Alphabet zu
- Definiere: h_1: K_1 → A durch

$h_1(k)=(f(1.\text{ Buchst. von } k)+f(2.\text{ Buchst. von } k)) \bmod 17$

Monat	J	F	M	A	M	J	J	A	S	O	N	D
h_1	11	11	14	0	14	14	14	5	7	9	12	9

Kollisionen

Suchen
Hashing - Beispiel

- 2. Versuch:
  - Definiere: $h_2: K_1 \rightarrow A$ durch

$h_2(k)=(f(2.\text{ Buchst. von } k)+f(3.\text{ Buchst. von } k)) \bmod 17$

  - immer noch **1 Kollision**

- 3. Versuch:
  - Definiere: $h_3: K_1 \rightarrow A$ durch

$h_3(k)=(2\cdot f(1.\text{ Buchst.})+2\cdot f(2.\text{ Buchst.})+f(3.\text{ Buchst.})) \bmod 17$

  - **kollisionsfrei**

Suchen
Hashing - weiteres Vorgehen

1. Bestimme Hash-Funktion h, die einfach zu berechnen ist und K möglichst gleichmäßig auf A abbildet.
2. Festlegung einer Strategie für unvermeidbare Kollisionen (man weiß ja vorher nicht, welche Schlüssel im Laufe der Lebenszeit eingefügt werden)

Suchen
Hashing - zufälliges h?

- Man könnte einfach eine zufällige Hash-Funktion h wählen und hoffen, daß Kollisionen dann sehr selten sind
- **Problem**: Schon bei kleinem K und großem A scheitert Wahl einer *zufälligen* Funktion

$$h: K \rightarrow A$$

Suchen
Hashing - Geburtstagsparadoxon

Geburtstagsparadoxon - Partywette

Wahrscheinlichkeit, daß von 23 Personen zwei am selben Tag Geburtstag haben, ist $\geq 1/2$.

- Genauer: $|K|=23$, $|A|=365$
- $f: K \rightarrow A$ zufällig gewählt
- Dann Wahrscheinlichkeit $\leq \mathbf{1/2}$, daß

$$f(k) \neq f(k') \text{ für alle } k,k' \in K, k \neq k'$$

Suchen
Hashing - Kollisionen

- Geburtstagsparadoxon: $|K|=23 << |A|=365$
- in der Praxis: $|K| >> |A|$, z.B.
  - K={Nachnamen mit $\leq$10 Buchstaben} in einer Personaldatei
  - A={1,...,1000} Personalnummern für max. 1000 Mitarbeiter
- h: $K \rightarrow A$ üblicherweise nicht umkehrbar eindeutig (**sehr viele** Kollisionen = Normalfall)

Suchen
Hashing - gute Hash-Funktion

- In der Praxis bewährt: (|A|=p Primzahl)

$$h(k)=k \bmod p$$

- k ggf. vorher numerisch machen

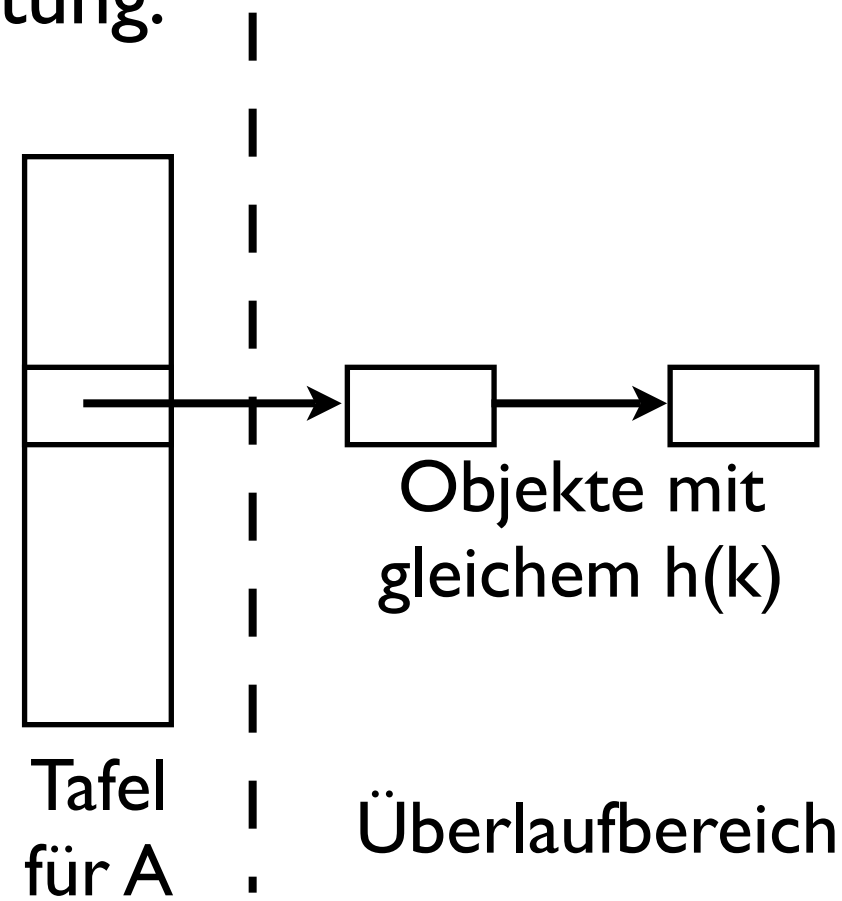
Suchen
Hashing - Kollisionsstrategien

- **offenes** Hashing: Überlaufbereich neben dem Adreßraum
- **geschlossenes** Hashing: nur Adreßraum darf genutzt werden

Suchen

Hashing - offenes Hashing

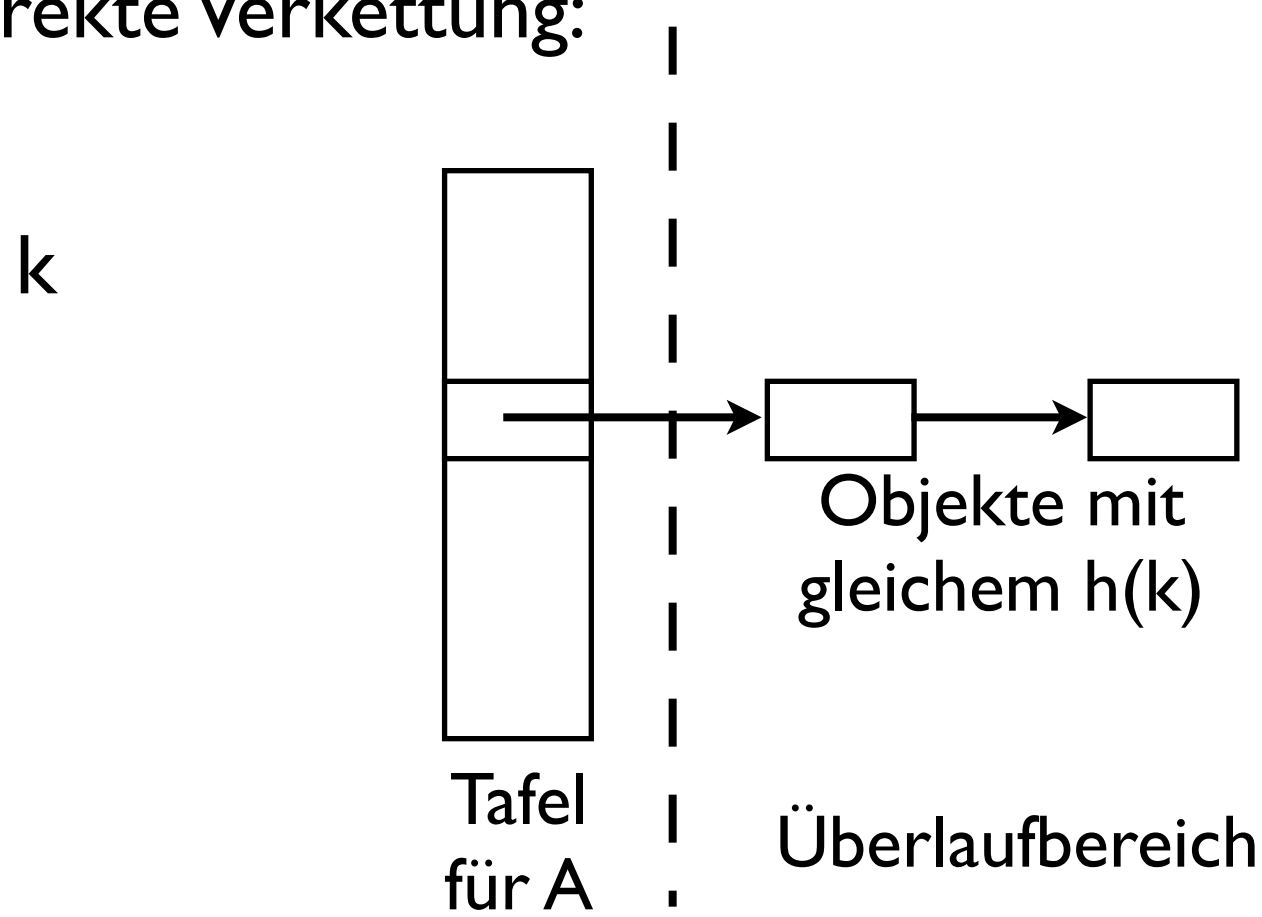
direkte Verkettung:



Suchen

Hashing - offenes Hashing

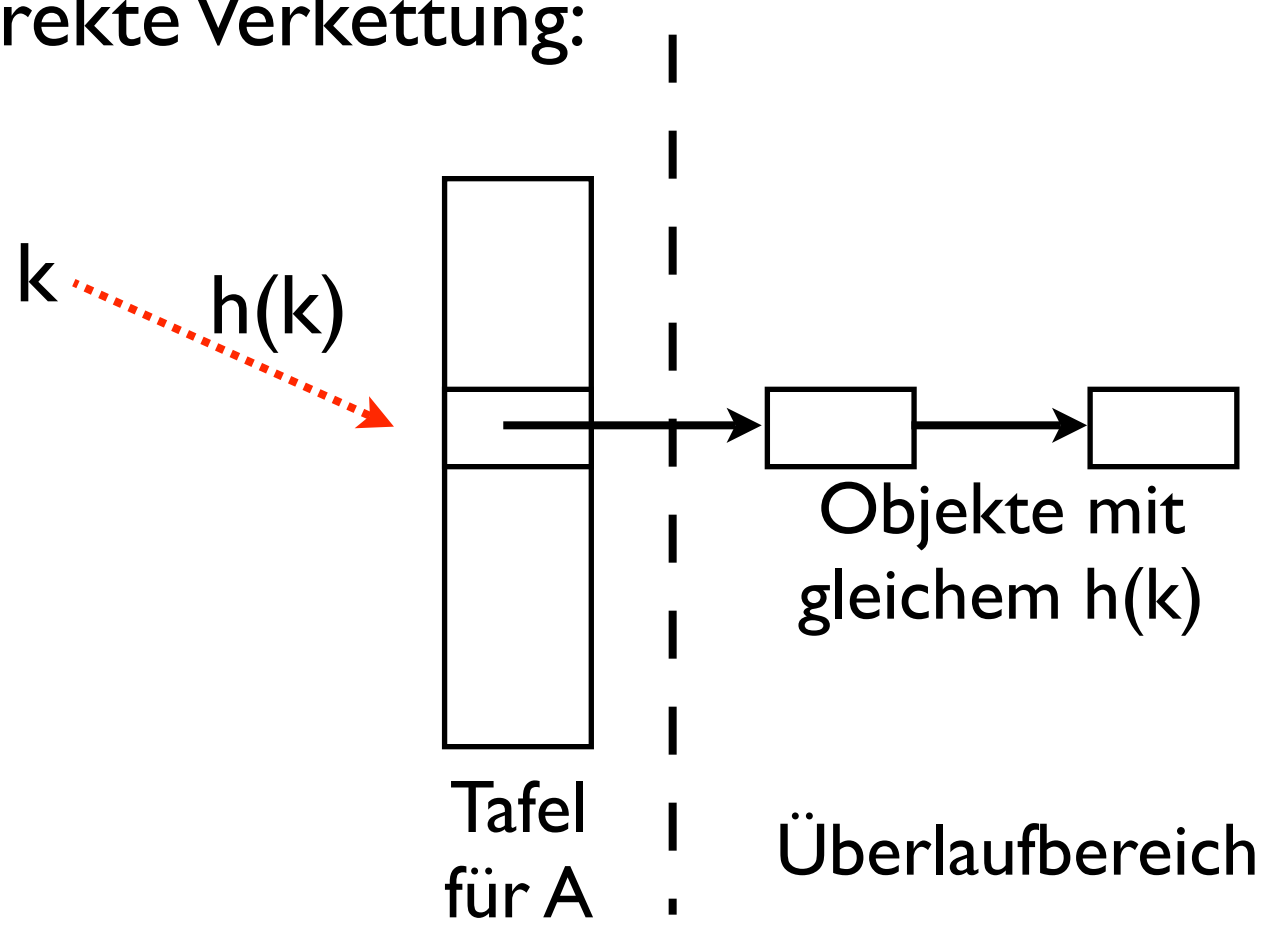
direkte Verkettung:



Suchen

Hashing - offenes Hashing

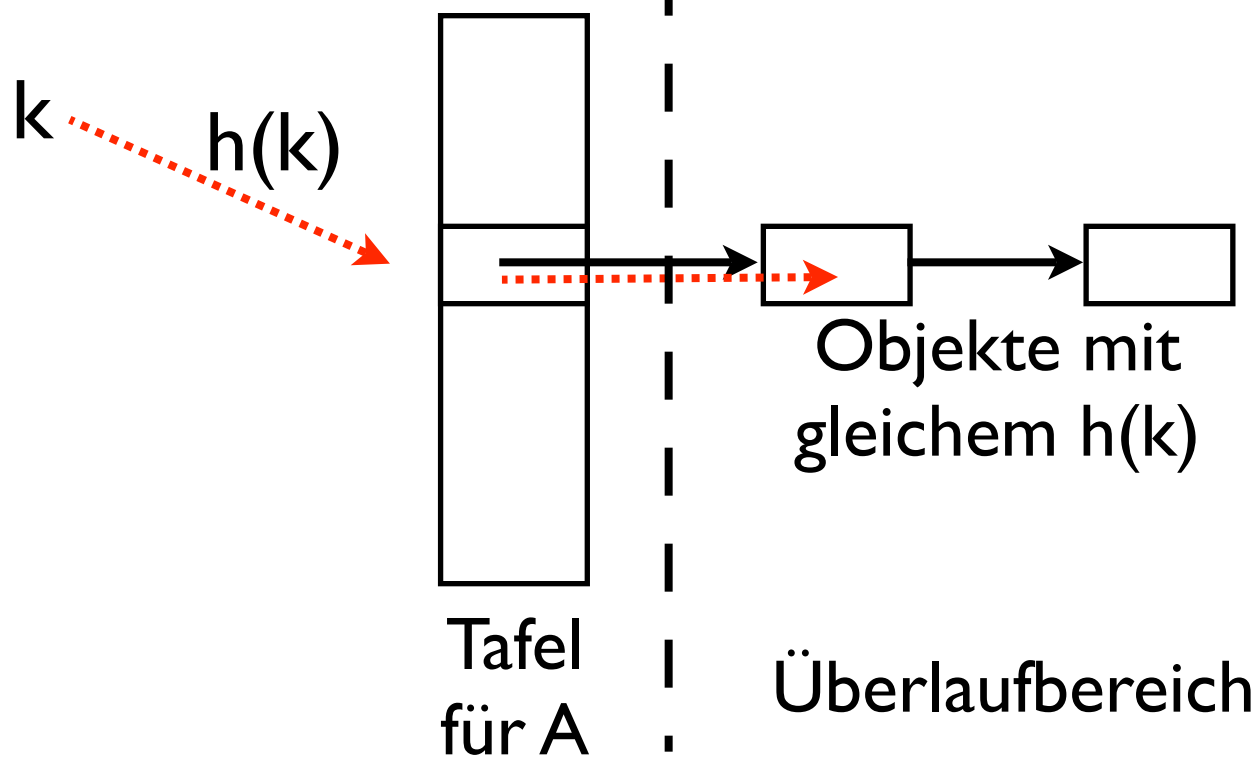
direkte Verkettung:



Suchen

Hashing - offenes Hashing

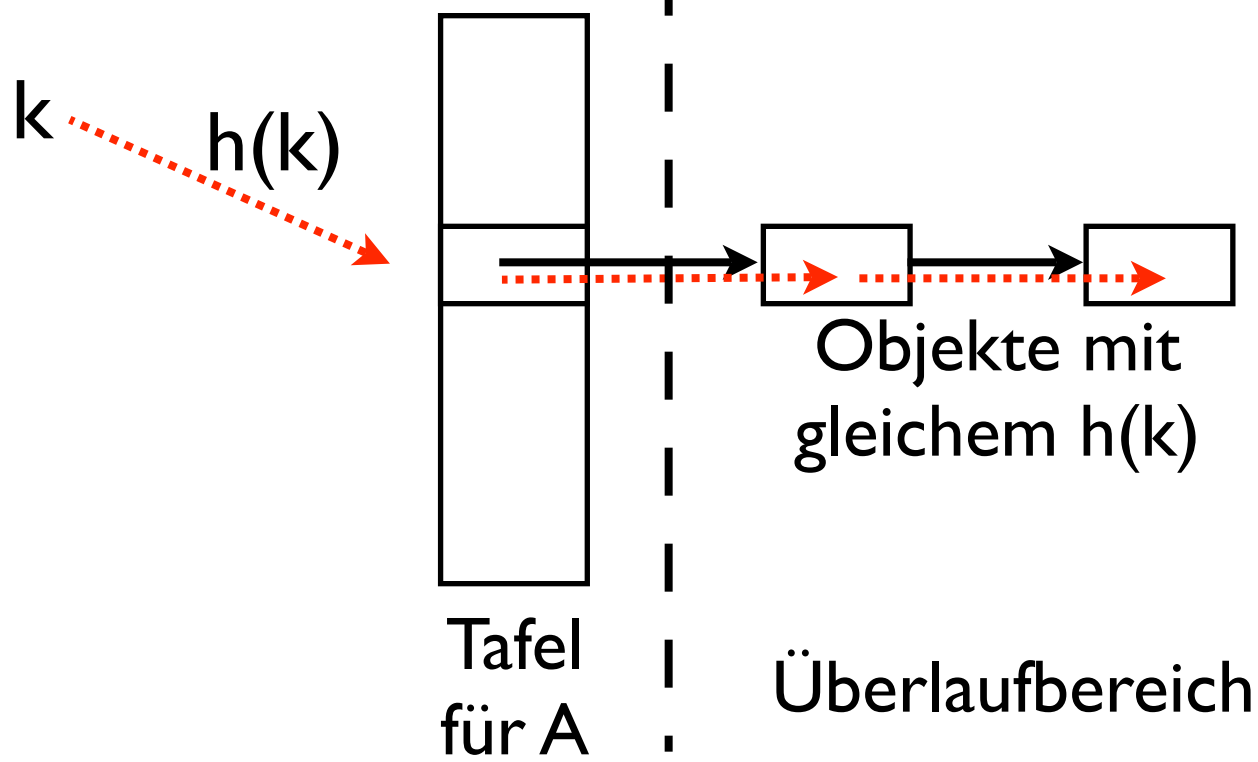
direkte Verkettung:



Suchen

Hashing - offenes Hashing

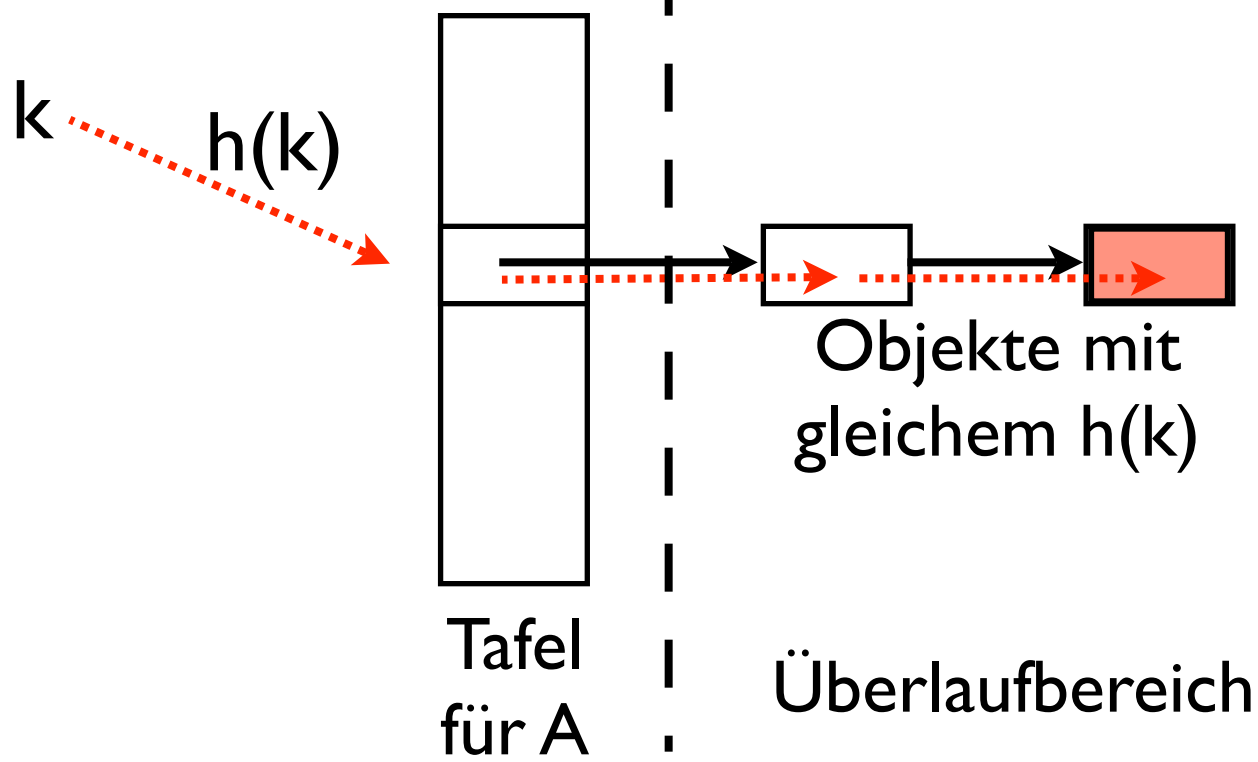
direkte Verkettung:



Suchen

Hashing - offenes Hashing

direkte Verkettung:



Suchen
offenes Hashing - Implementierung

Datenstruktur

```
type Objekt = record
                  info: Basistyp
                  next: ↑Objekt
              end
type Tafel = array [A] of ↑Objekt
var T: Tafel
```

Suchen nach Objekt x mit Schlüssel k

```
h:=h(k); p:=T[h];
"sequentielle Suche nach x"
```

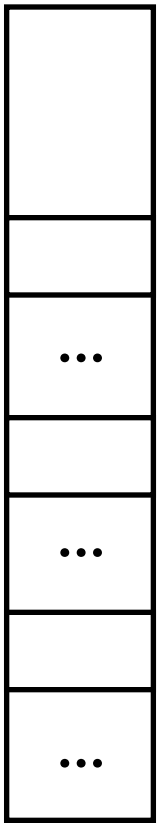
Suchen
offenes Hashing - Komplexität

- Laufzeit:
  - gut bei kurzen Listen
  - |K|/|A| Zeit, da je Liste im Mittel |K|/|A| Einträge
- Speicher:
  - |K| Objekte
  - zusätzl. |K|+|A| Zeiger

Suchen

Hashing - geschlossenes Hashing

offene Adressierung: nur Adreßraum darf verwendet werden; bei Kollision suche freien Platz

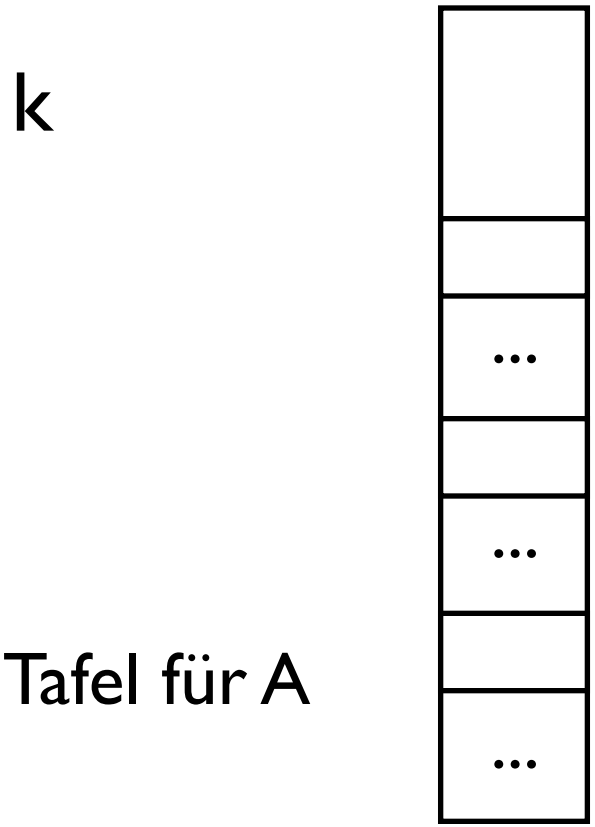


Tafel für A

Suchen

Hashing - geschlossenes Hashing

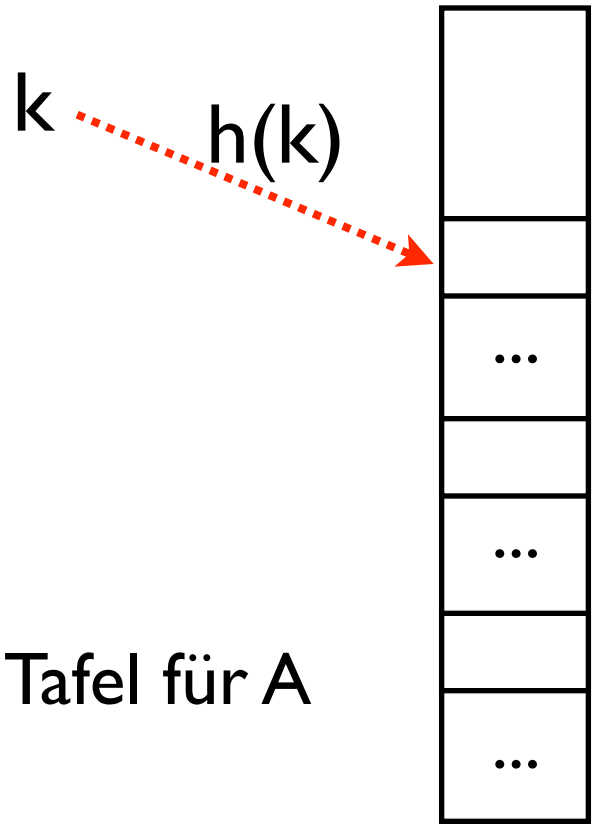
offene Adressierung: nur Adreßraum darf verwendet werden; bei Kollision suche freien Platz



Suchen

Hashing - geschlossenes Hashing

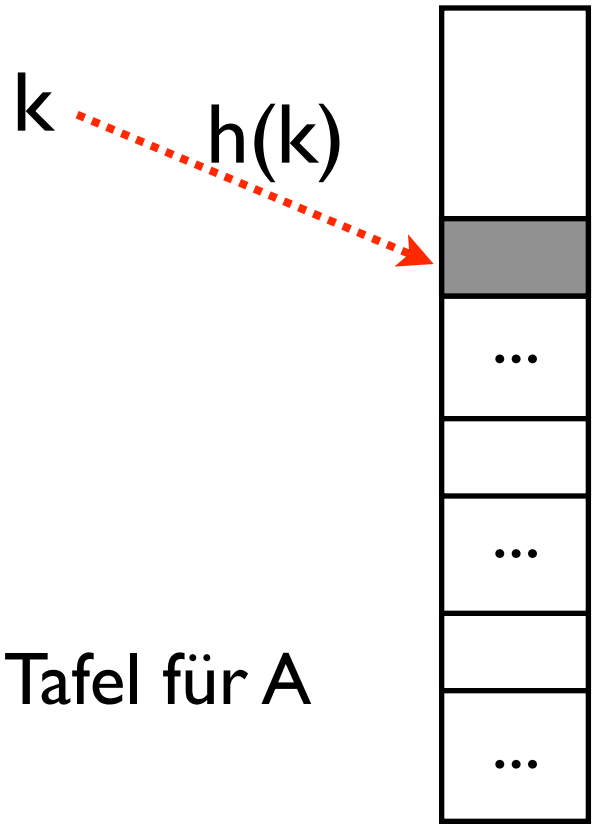
offene Adressierung: nur Adreßraum darf verwendet werden; bei Kollision suche freien Platz



Suchen

Hashing - geschlossenes Hashing

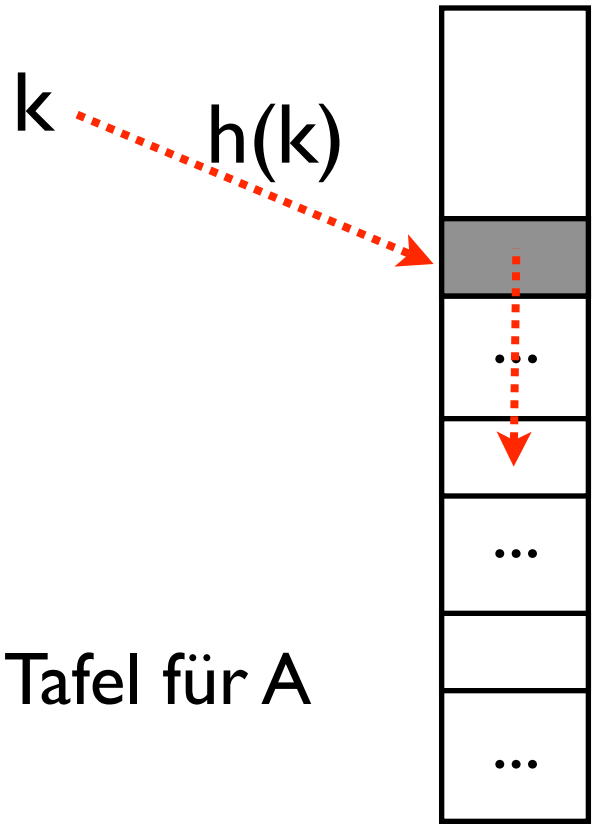
offene Adressierung: nur Adreßraum darf verwendet werden; bei Kollision suche freien Platz



Suchen

Hashing - geschlossenes Hashing

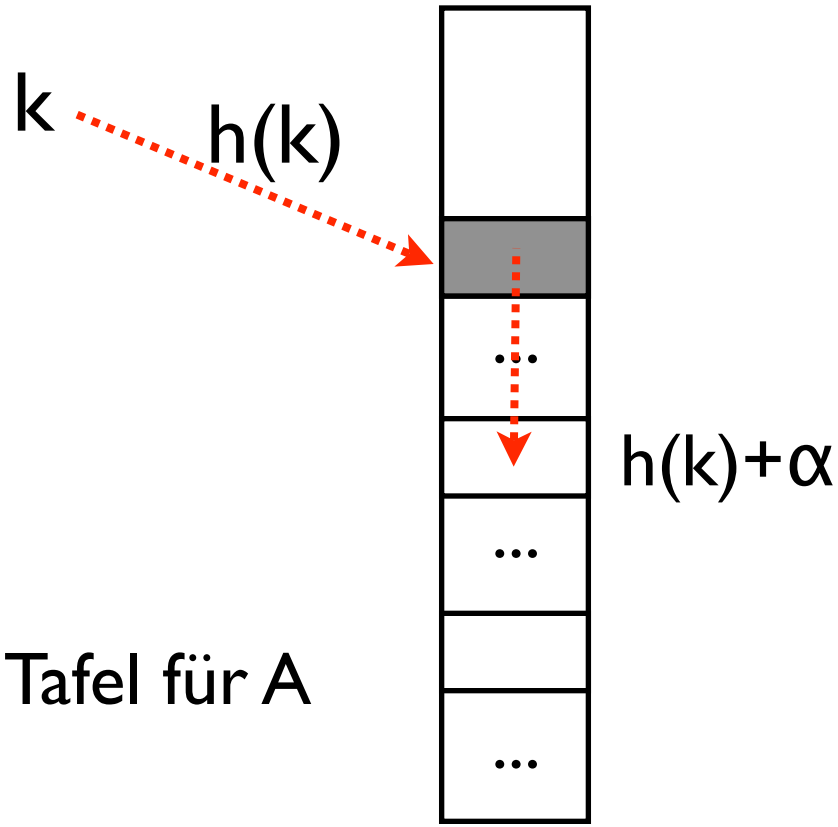
offene Adressierung: nur Adreßraum darf verwendet werden; bei Kollision suche freien Platz



Suchen

Hashing - geschlossenes Hashing

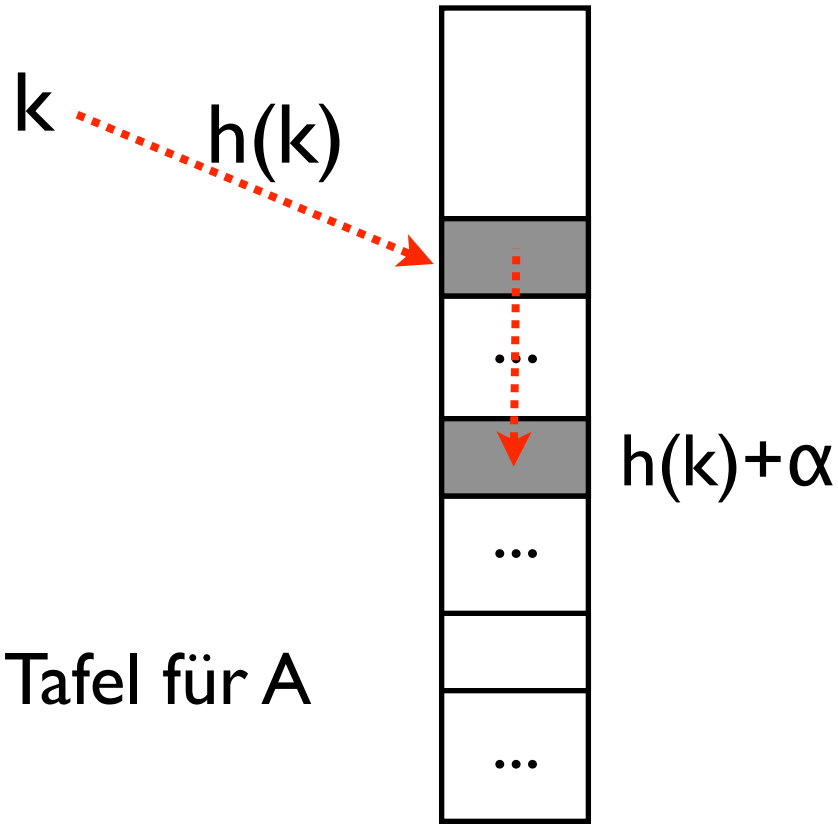
offene Adressierung: nur Adreßraum darf verwendet werden; bei Kollision suche freien Platz



Suchen

Hashing - geschlossenes Hashing

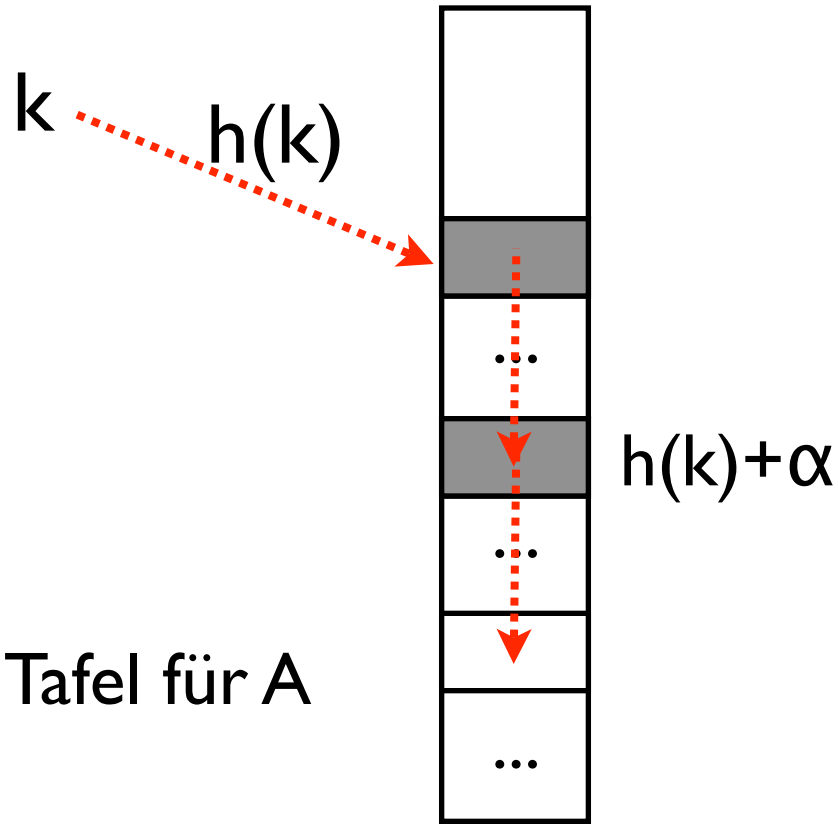
offene Adressierung: nur Adreßraum darf verwendet werden; bei Kollision suche freien Platz



Suchen

Hashing - geschlossenes Hashing

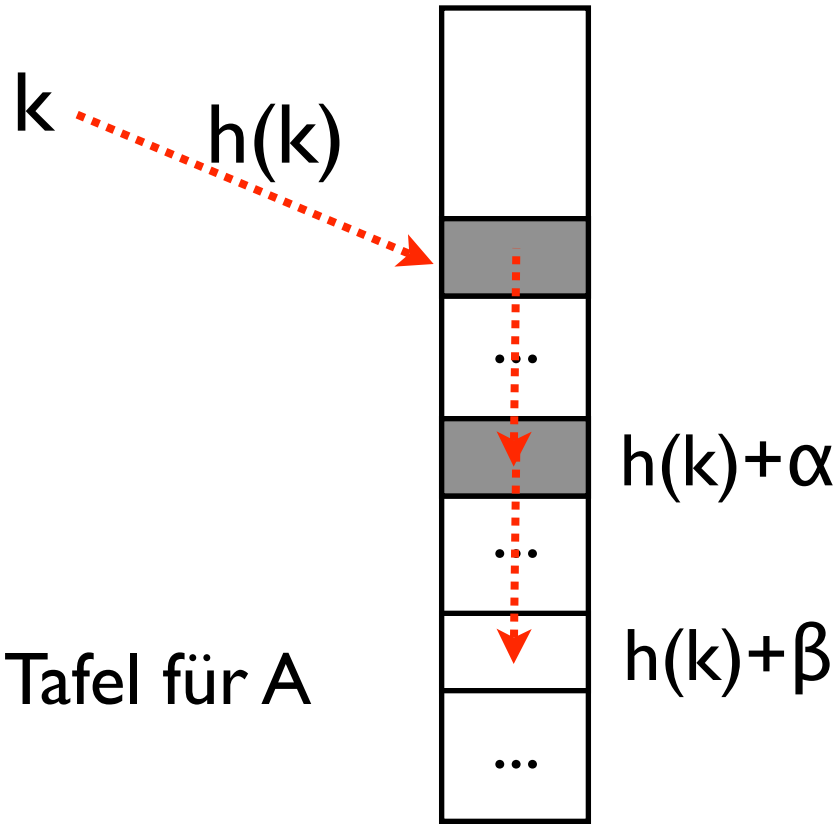
offene Adressierung: nur Adreßraum darf verwendet werden; bei Kollision suche freien Platz



Suchen

Hashing - geschlossenes Hashing

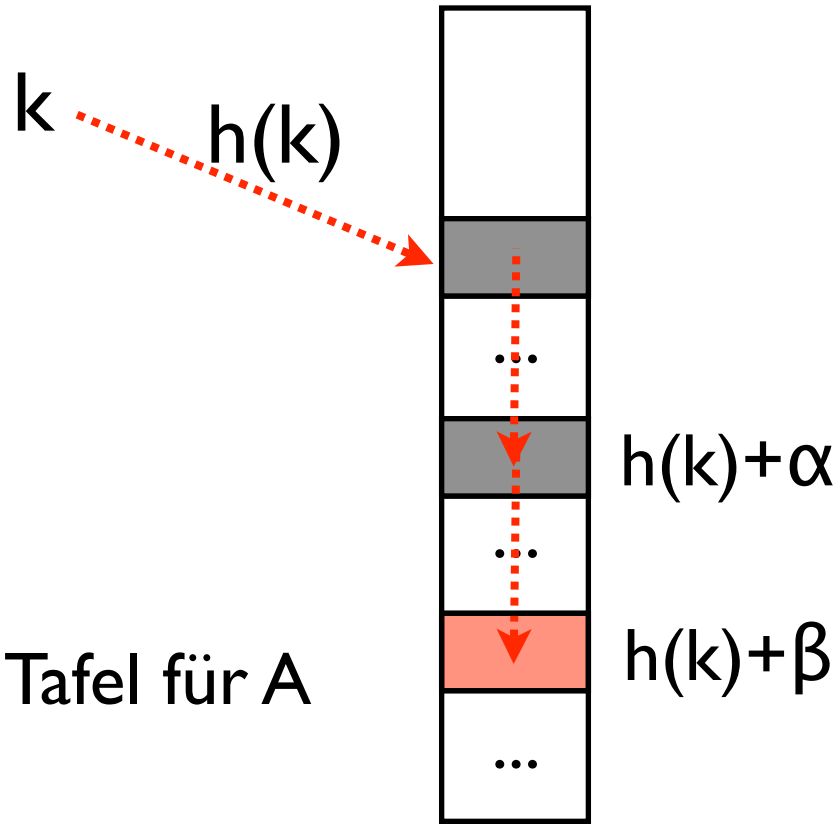
offene Adressierung: nur Adreßraum darf verwendet werden; bei Kollision suche freien Platz



Suchen

Hashing - geschlossenes Hashing

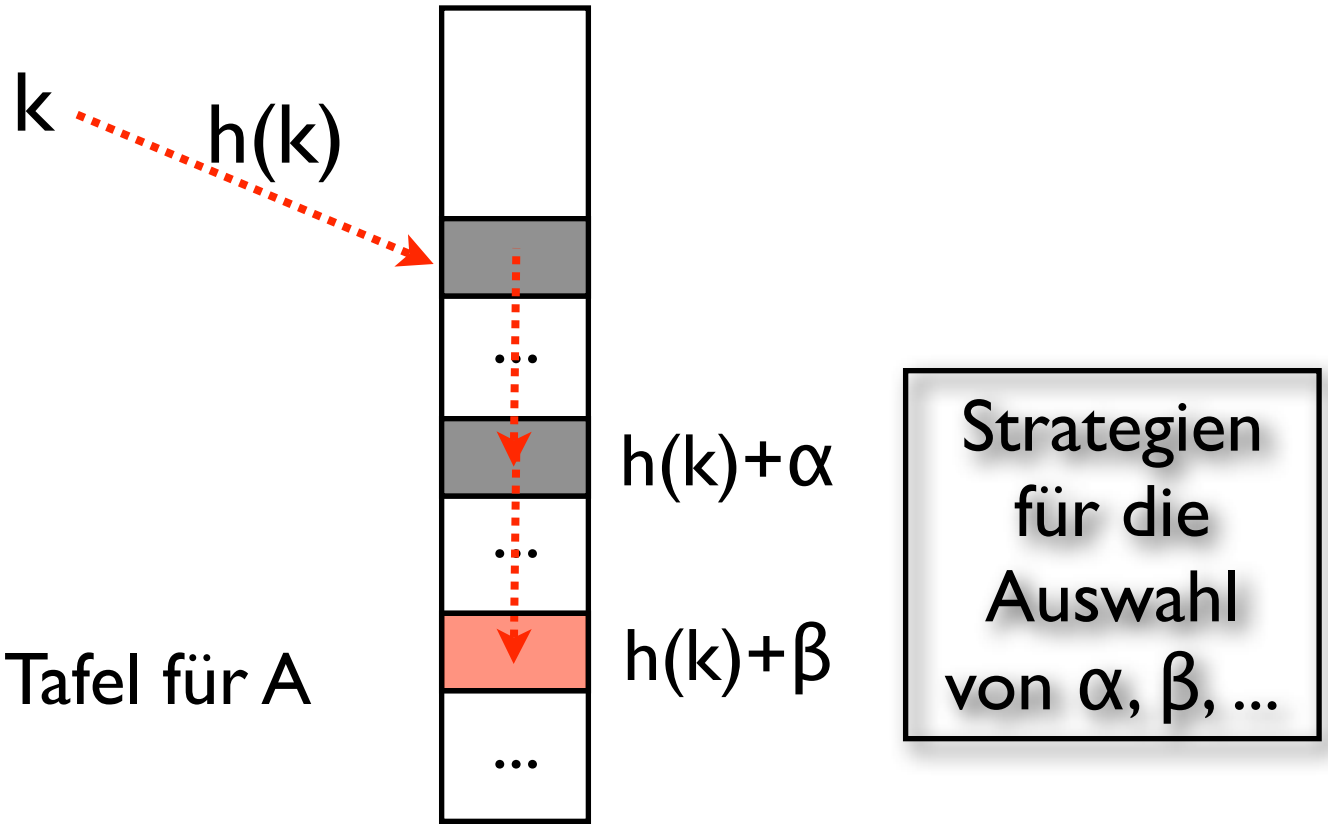
offene Adressierung: nur Adreßraum darf verwendet werden; bei Kollision suche freien Platz



Suchen

Hashing - geschlossenes Hashing

offene Adressierung: nur Adreßraum darf verwendet werden; bei Kollision suche freien Platz



Suchen
Hashing - lineare Verschiebung

Strategie

Wähle Konstante c teilerfremd zu p.

Falls k nicht an Pos. h(k), suche k nacheinander an den Stellen (jeweils mod |A|)

$$h(k)+c,\ h(k)+2c,\ \ldots,\ h(k)+ic,\ \ldots$$

Dto. beim Einfügen.

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
												Ja					

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
												Ja					

Fe

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
												Ja					

Fe ⟶

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
												Ja			Fe		

Fe ⟶

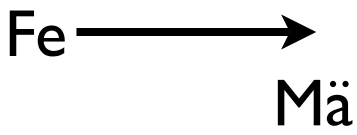
Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
												Ja			Fe		



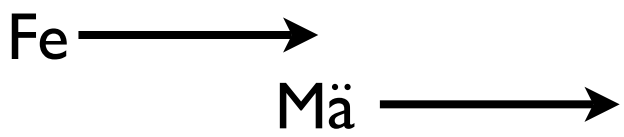
Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
												Ja			Fe		



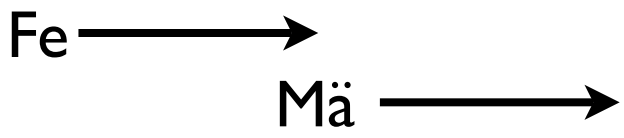
Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Mä											Ja			Fe		



Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Mä											Ja			Fe		

Ap

Fe →

Mä →

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Mä											Ja			Fe		

Ap →

Fe →

Mä →

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Mä			Ap								Ja			Fe		

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

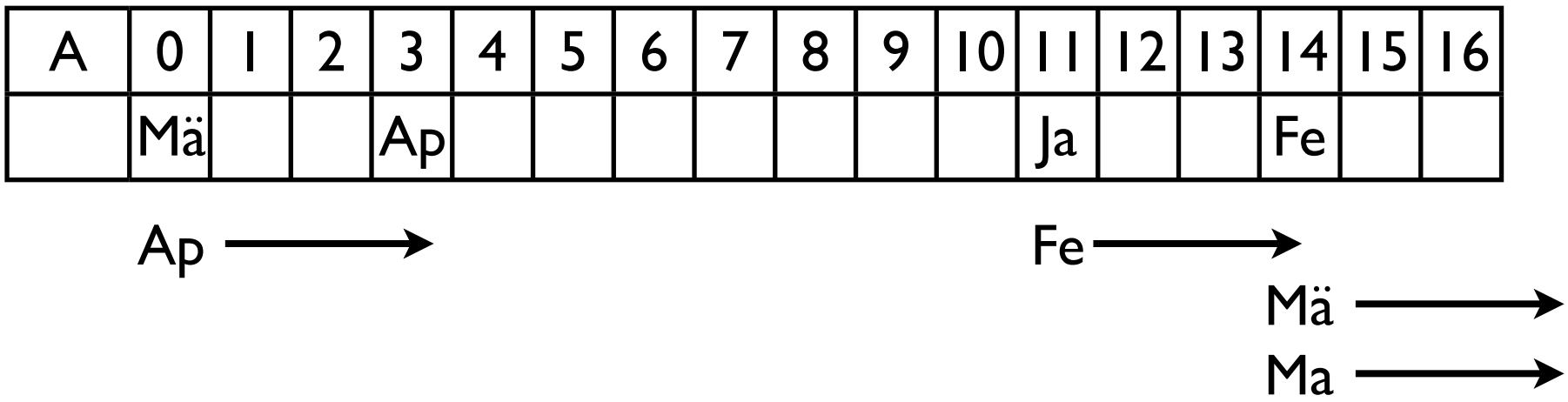
A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Mä			Ap								Ja			Fe		

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17



Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Mä			Ap								Ja			Fe		

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Mä			Ap								Ja			Fe		

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Mä			Ap								Ja			Fe		

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Mä			Ap			Ma					Ja			Fe		

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Mä			Ap			Ma			Jn		Ja			Fe		

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Mä			Ap			Ma			Jn		Ja	Jl		Fe		

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Mä			Ap		Au	Ma			Jn		Ja	Jl		Fe		

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Mä			Ap		Au	Ma	Se		Jn		Ja	Jl		Fe		

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Mä			Ap		Au	Ma	Se		Jn		Ja	Jl		Fe	Ok	

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Mä	No		Ap		Au	Ma	Se		Jn		Ja	Jl		Fe	Ok	

Suchen

Hashing - lineare Verschiebung - Beispiel

- Einfügen: Sei c=3 (lineare Verschiebung) und

h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Mä	No		Ap	De	Au	Ma	Se		Jn		Ja	Jl		Fe	Ok	

Suchen

Hashing - quadratische Verschiebung

Strategie

Falls k nicht an Pos. h(k), suche k nacheinander an den Stellen (jeweils mod |A|)

$$h(k)+1, h(k)+4, ..., h(k)+i^2, ...$$

Dto. beim Einfügen.

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
												Ja					

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
												Ja					

Fe

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
												Ja					

Fe→

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
												Ja	Fe				

Fe→

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
												Ja	Fe		Mä		

Fe→

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap											Ja	Fe		Mä		

Fe→

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap											Ja	Fe		Mä		

Fe→ Ma

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap											Ja	Fe		Mä		

Fe→ Ma→

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap											Ja	Fe		Mä	Ma	

Fe→ Ma→

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap											Ja	Fe		Mä	Ma	

Fe → Ma →

Jn

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap											Ja	Fe		Mä	Ma	

Fe→ Ma→

Jn→

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap											Ja	Fe		Mä	Ma	

Fe → Ma →

Jn →

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap											Ja	Fe		Mä	Ma	

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap	Jn										Ja	Fe		Mä	Ma	

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap	Jn										Ja	Fe		Mä	Ma	

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap	Jn										Ja	Fe		Mä	Ma	

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap	Jn										Ja	Fe		Mä	Ma	

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap	Jn										Ja	Fe		Mä	Ma	

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap	Jn										Ja	Fe		Mä	Ma	

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap	Jn					Jl					Ja	Fe		Mä	Ma	

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap	Jn				Au	Jl					Ja	Fe		Mä	Ma	

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap	Jn				Au	Jl	Se				Ja	Fe		Mä	Ma	

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap	Jn				Au	Jl	Se		Ok		Ja	Fe		Mä	Ma	

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap	Jn				Au	Jl	Se		Ok		Ja	Fe		Mä	Ma	

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap	Jn				Au	Jl	Se		Ok		Ja	Fe		Mä	Ma	

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap	Jn				Au	Jl	Se		Ok		Ja	Fe	No	Mä	Ma	

Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap	Jn				Au	Jl	Se		Ok		Ja	Fe	No	Mä	Ma	

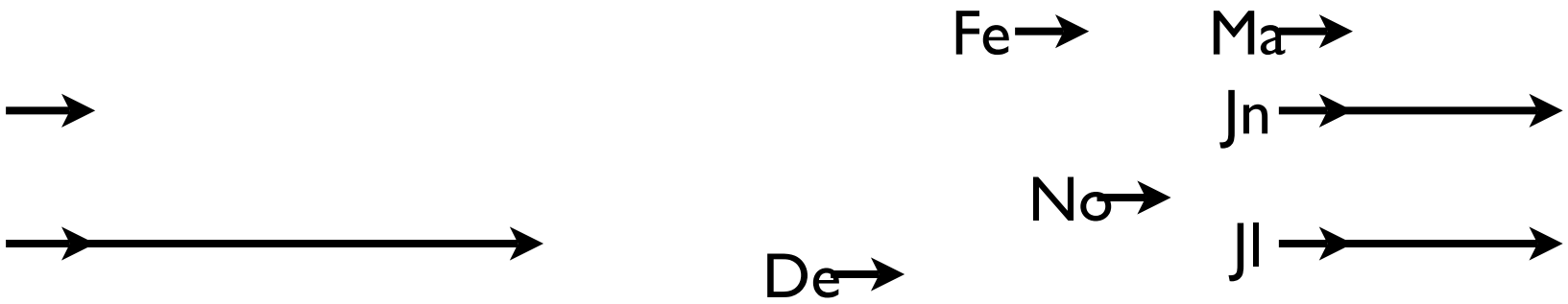
Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

  h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap	Jn				Au	Jl	Se		Ok		Ja	Fe	No	Mä	Ma	



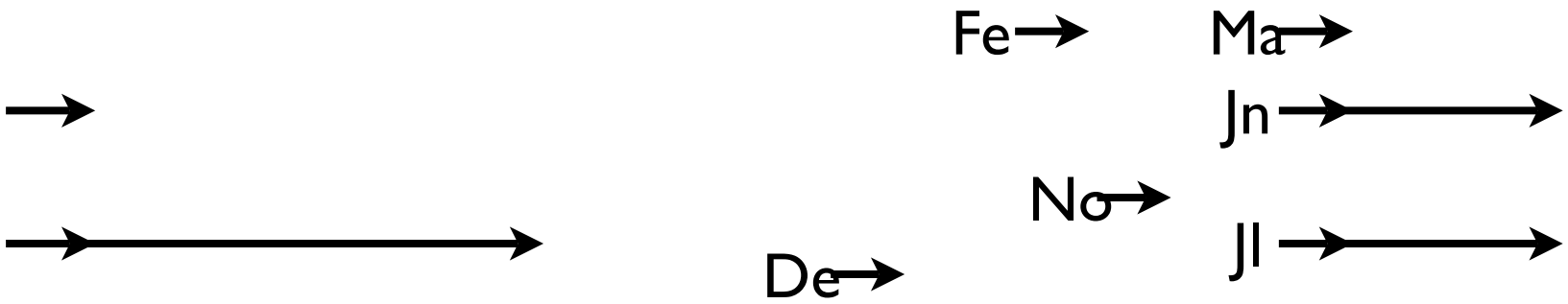
Suchen

Hashing - quadr. Verschiebung - Beispiel

- Einfügen: Sei

h(k)=f(1. Buchst. von k)+ f(2. Buchst. von k)) mod 17

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Ap	Jn				Au	Jl	Se		Ok	De	Ja	Fe	No	Mä	Ma	



Suchen
Hashing - Problem

- Beobachte: es können lange Ketten belegter Zellen entstehen
- bei ziemlich voller Hash-Tabelle sehr häufig
- im schlimmsten Fall nähert man sich dem sequentiellen Suchen mit O(n)
- Analyse der Wahrscheinlichkeit für lange Suchketten nötig

Suchen
Hashing - Analyse

Analyse des geschlossenen Hashings

- Annahme: Ignoriere zunächst mögliche Kettenbildung (→uniformes/ideales Hashing)

Frage: Kosten des Einfügens

- |A|=m Größe der Tabelle
- n Objekte sind bereits eingetragen
- Wie groß ist die Wahrscheinlichkeit, daß beim Einfügen des (n+1)-ten Elements x genau r Zellen getestet werden müssen?

Suchen
Hashing - Analyse

- Wahrscheinlichkeit, beim Einfügen des (n+1)-ten Elements x genau r Zellen testen zu müssen?
- x bestimmt eine Folge von Zellen $h_0(x), h_1(x), \ldots$, die nacheinander getestet werden
- es werden genau r Tests vorgenommen, wenn die ersten r-1 Zellen $h_0(x), \ldots, h_{r-2}(x)$ belegt und die r-te Zelle $h_{r-1}(x)$ frei ist (oder das gesuchte Element enthält)
- Sei P_r die Wahrscheinlichkeit, daß genau r Tests vorgenommen werden

Suchen
Hashing - Analyse

- Sei P_r die Wahrscheinlichkeit, daß genau r Tests vorgenommen werden:

$$P_r = \frac{\#\text{Möglichkeiten, bei denen die ersten r-1 Zellen belegt und r-te Zelle frei}}{\#\text{Möglichkeiten, n besetzte und n-m freie Zellen zu haben}}$$

- Nenner offenbar: $\binom{m}{n}$
- Zähler: Durch $h_0(x),...,h_{r-1}(x)$ sind r Zellen als belegt oder frei bereits festgelegt. Die übrigen n-(r-1) Zellen können bel. auf die noch verbleibenden m-r Zellen verteilt werden: $\binom{m-r}{n-(r-1)}$

Suchen
Hashing - Analyse

$$P_r = \frac{\binom{m-r}{n-r+1}}{\binom{m}{n}}$$

- Erwartungswert:

$$\sum_{r=1}^{m} r \cdot P_r = \sum_{r=1}^{m} r \cdot \frac{\binom{\ }{\ }}{\binom{\ }{\ }} = \ldots$$

... <Buch von Güting> ... $= \frac{m+1}{m-n+1}$

Suchen
Hashing - Analyse

- erwartete Kosten des Einfügens des (n+1)-ten Elements

  zugleich

- erwartete Kosten des erfolglosen Suchens in einer Hash-Tabelle der Größe m mit n Einträgen:

$$\frac{m+1}{m-n+1}$$

- Bilde Mittelwert über alle n eingefügten Elemente in eine anfangs leere Hash-Tabelle

Suchen
Hashing - Analyse

- Erwartete Kosten für Einfügen und erfolgloses Suchen gemittelt über alle n eingefügten Elemente in eine anfangs leere Hash-Tabelle:

$$\frac{1}{n} \sum_{k=1}^{n} \frac{m+1}{m-(k-1)+1} = \frac{m+1}{n} \sum_{k=1}^{n} \frac{1}{m-k+2}$$

Suchen

Hashing - Einschub harmon. Zahl

- **Harmonische Zahl**:

$$H_n = \sum_{k=1}^{n} (1/k)$$

- Es gilt:

$$\frac{\lfloor \log n \rfloor + 1}{2} < H_n \leq \lfloor \log n \rfloor + 1$$

$$\ln n < H_n < \ln n + 1, \; n > 1$$

Suchen
Hashing - Analyse

- Erwartete Kosten für Einfügen und erfolgloses Suchen gemittelt über alle n eingefügten Elemente in eine anfangs leere Hash-Tabelle:

$$\frac{1}{n} \sum_{k=1}^{n} \frac{m+1}{m-(k-1)+1} = \frac{m+1}{n} \sum_{k=1}^{n} \frac{1}{m-k+2}$$

$$= \frac{m+1}{n} (H_{m+1} - H_{m-n+1}) \approx \frac{m+1}{n} \ln \frac{m+1}{m-n+1}$$

Suchen
Hashing - Analyse

$$\frac{m+1}{n} \ln \frac{m+1}{m-n+1}$$

- Wählt man als Auslastungsgrad für die Hash-Tabelle $\alpha := n/m$, so ist

$$\frac{m+1}{m-n+1} \approx \frac{1}{1-\alpha}$$

und

$$\frac{m+1}{n} \ln \frac{m+1}{m-n+1} \approx \frac{1}{\alpha} \ln \frac{1}{1-\alpha}$$

Suchen
Hashing - Analyse

Auslastung α	Kosten für Einfügen/ erfolgslose Suche	Kosten für Entfernen/ erfolgreiche Suche
50%	2	1,38
80%	5	2,01
95%	20	3,15

Suchen
Hashing - Analyse

- Bezieht man die Effekte ein, die durch Kettenbildung verursacht werden, so verschlechtern sich die Werte dramatisch:
  - Lineare Verschiebung:
    - erfolgreiche Suche: $LIN_n \approx \frac{1}{2}\left(1+\frac{1}{1-\alpha}\right)$
    - erfolglose Suche: $LIN_n' \approx \frac{1}{2}\left(1+\frac{1}{(1-\alpha)^2}\right)$
  - Quadratische Verschiebung deutlich besser: s. Buch von Ottmann/Widmayer

Suchen
Hashing - Analyse

Auslastung α	LIN_n	LIN_n'
50%	1,5	2,5
80%	3	13
90%	5,5	50,5
95%	10,5	200,5

Suchen
Hashing - Analyse

Auslastung α	LIN_n	LIN_n'
50%	1,5	2,5
80%	3	13
90%	5,5	50,5
95%	10,5	200,5

Suchen

Hashing - Analyse

Auslastung α	LIN_n	LIN_n'
50%	1,5	2,5
80%	3	13
90%	5,5	50,5
95%	10,5	200,5

Effizienzgrenze

Suchen

Hashing - Analyse

Auslastung α	LIN_n	LIN_n'
50%	1,5	2,5
80%	3	13
90%	5,5	50,5
95%	10,5	200,5

Effizienzgrenze

Hash-Tabellen sollten nicht zu mehr als 80% gefüllt werden.