IIF
INSTITUT FÜR
INFORMATIK
FREIBURG

Vorlesung Informatik 2
Algorithmen und Datenstrukturen

(22 - AVL-Bäume: Entfernen)

Prof. Dr. Susanne Albers

Definition von AVL-Bäumen

Definition: Ein binärer Suchbaum heißt AVL-Baum oder höhenbalanziert, wenn für jeden Knoten v gilt, dass sich die Höhe des rechten Teilbaumes $h(T_r)$ von v und die Höhe des linken Teilbaumes $h(T_l)$ von v um maximal 1 unterscheiden.

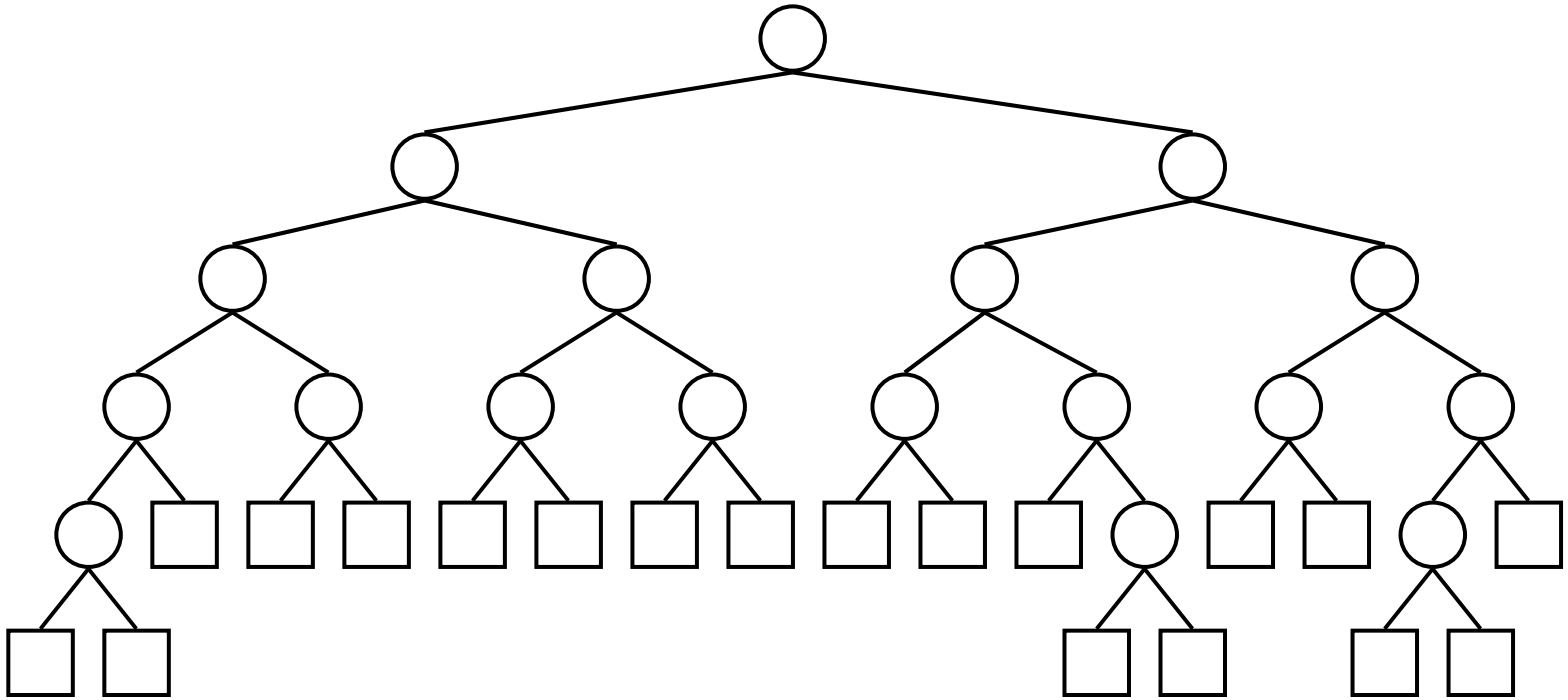
Balancegrad:

$$bal(v) = h(T_r) - h(T_l) \in \{-1, 0, +1\}$$

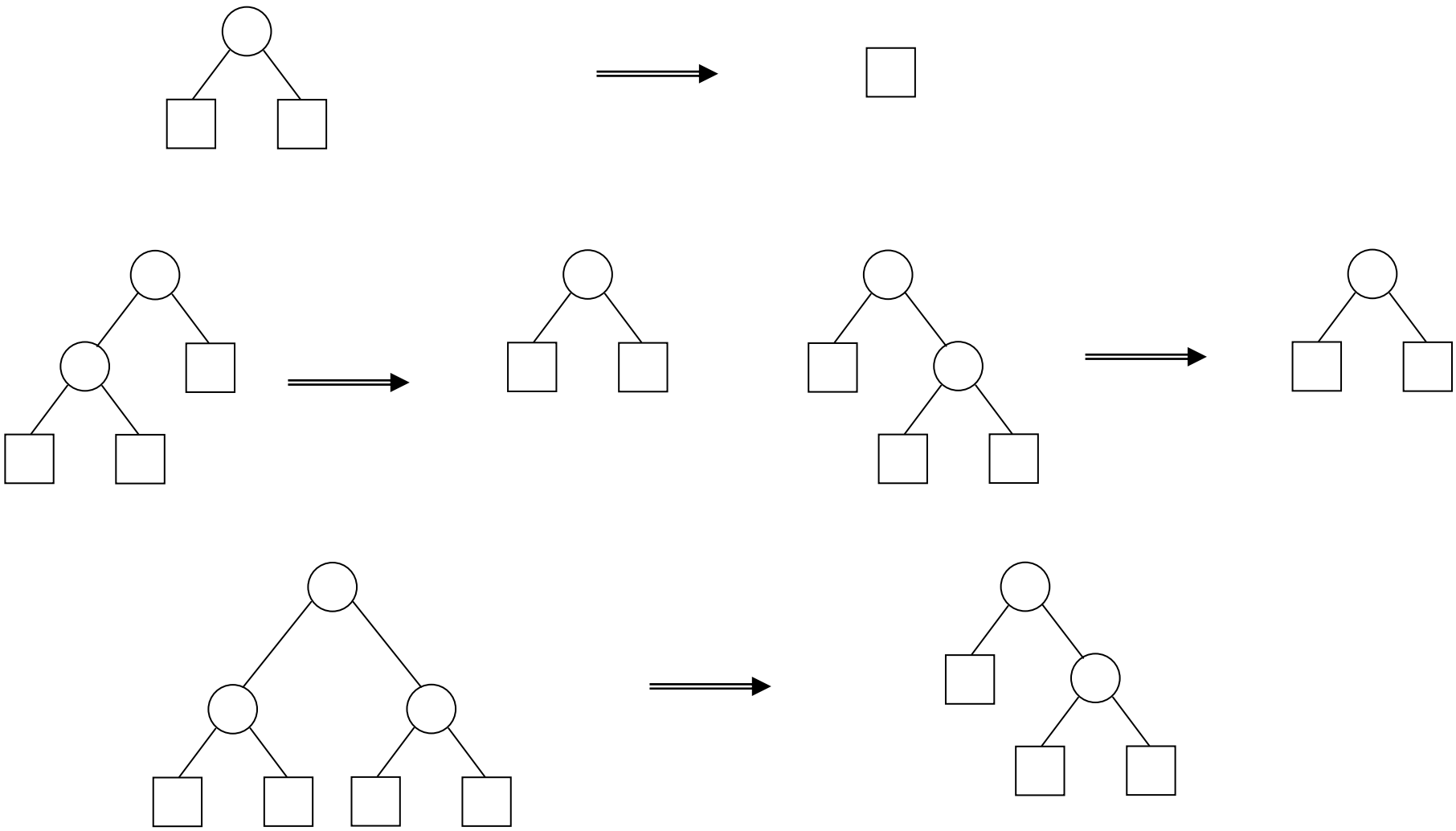
Entfernen aus einem AVL-Baum

- Wie gehen ähnlich vor wie bei Suchbäumen:
  1. Suche nach dem zu entfernenden Schlüssel.
  2. Falls der Schlüssel nicht enthalten ist, sind wir fertig.
  3. Andernfalls unterscheiden wir drei Fälle:
     (a) Der zu löschende Knoten hat keine inneren Knoten als Nachfolger.
     (b) Der zu löschende Knoten hat genau einen inneren Knoten als Nachfolger.
     (c) Der zu löschende Knoten hat zwei innere Knoten als Nachfolger.
- Nach dem Löschen eines Knotens kann ggf. die AVL-Baum-Eigenschaft verletzt sein (wie beim Einfügen).
- Dies muss entsprechend behandelt werden.

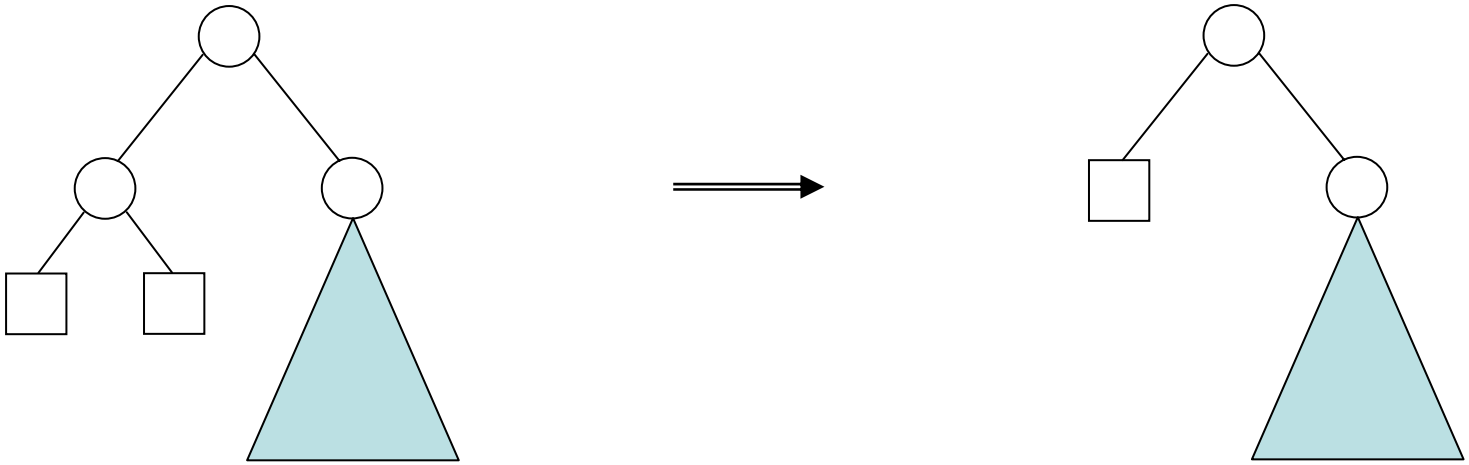
Beispiel



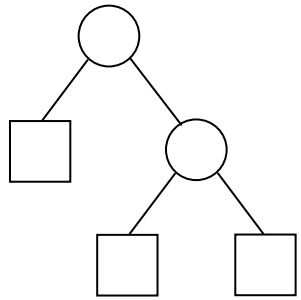
Zu löschender Knoten hat nur Blätter als Nachfolger



Der zu löschende Knoten hat nur Blätter als Nachfolger

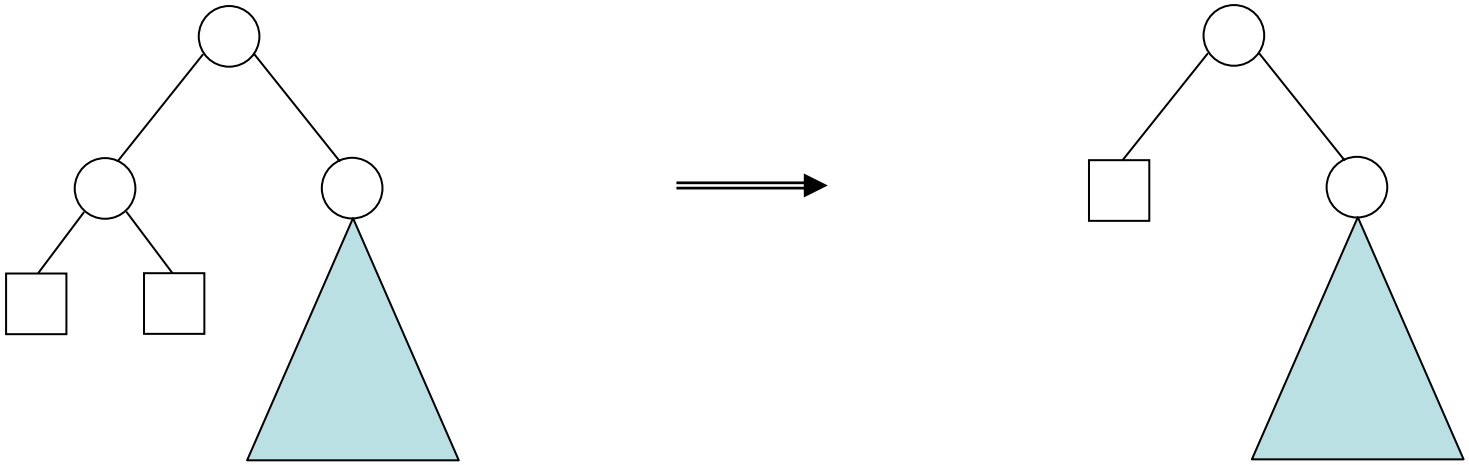


Höhe $\in \{1, 2\}$

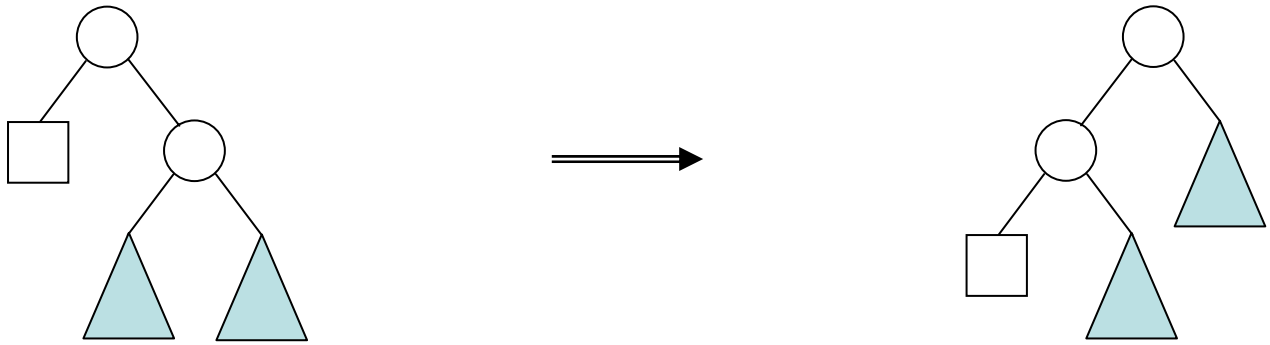


Fall1: Höhe = 1: Fertig!

Der zu löschende Knoten hat nur Blätter als Nachfolger

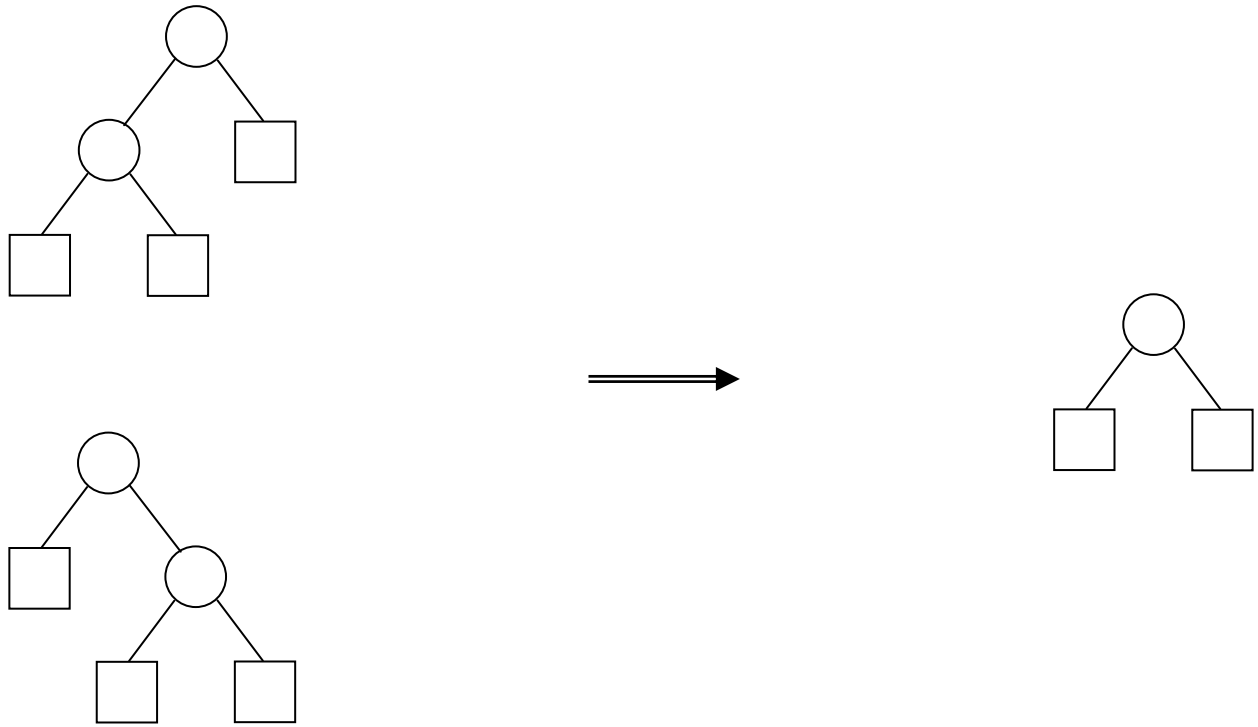


Fall2: Höhe = 2



Achtung: Höhe kann um 1 gesunken sein!

Zu löschender Knoten hat einen inneren Knoten als Nachfolger



Zu löschender Knoten hat genau 2 innere Knoten als Nachfolger

- Wir gehen zunächst so vor, wie bei Suchbäumen:

  1. Wir ersetzen den Inhalt des zu löschenden Knotens p durch den seines symmetrischen Nachfolgers q.

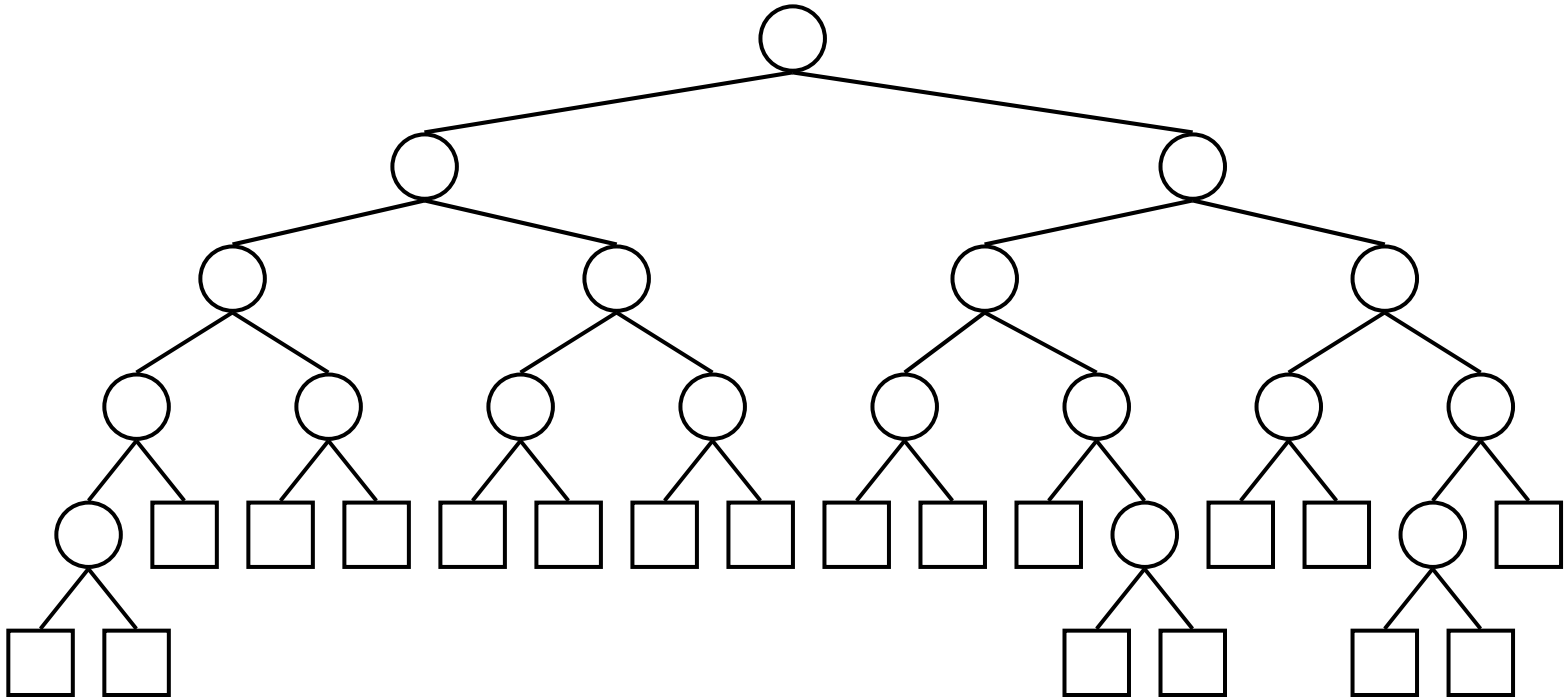
  2. Danach löschen wir den Knoten *q*.

- Da q höchstens einen inneren Knoten als Nachfolger (den rechten) haben kann, treffen für *q* die Fälle 1 und 2 zu.

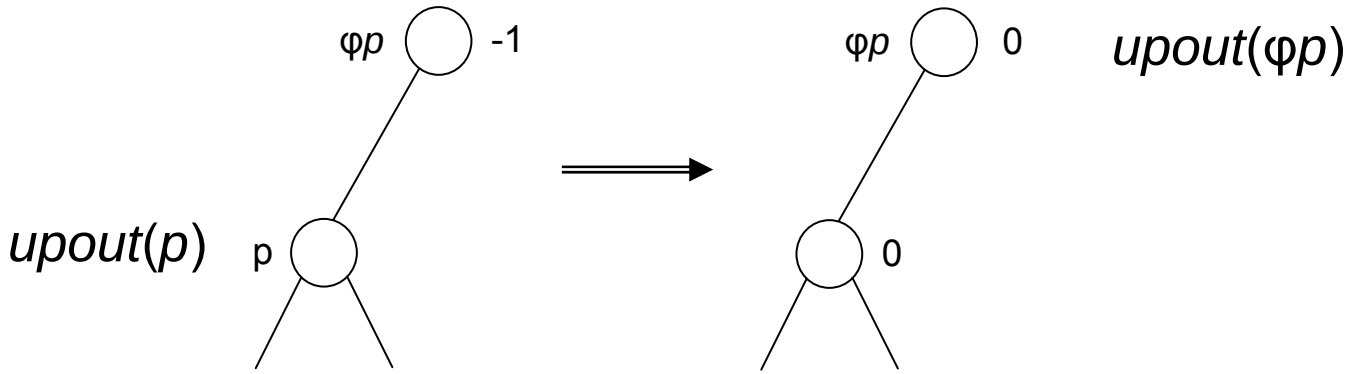
Die Methode *upout*

- Die Methode *upout* funktioniert ähnlich wie die Methode *upin*.
- Sie wird entlang des Suchpfads rekursiv aufgerufen und adjustiert die Balancegrade durch Rotationen und Doppelrotationen.
- Wenn *upout* für einen Knoten *p* aufgerufen wird, gilt (s.o.):
  1. $bal(p) = 0$
  2. Die Höhe des Teilbaums mit Wurzel *p* ist um 1 gefallen.
- *upout* wird nun so lange rekursiv aufgerufen, wie diese beiden Bedingungen gelten (Invariante).
- Wiederum unterscheiden wir 2 Fälle, abhängig davon, ob *p* linker oder rechter Nachfolger seines Vorgängers φp ist.
- Da beide Fälle symmetrisch sind, behandeln wir im Folgenden nur den Fall, dass *p* linker Nachfolger von φp ist.

Beispiel

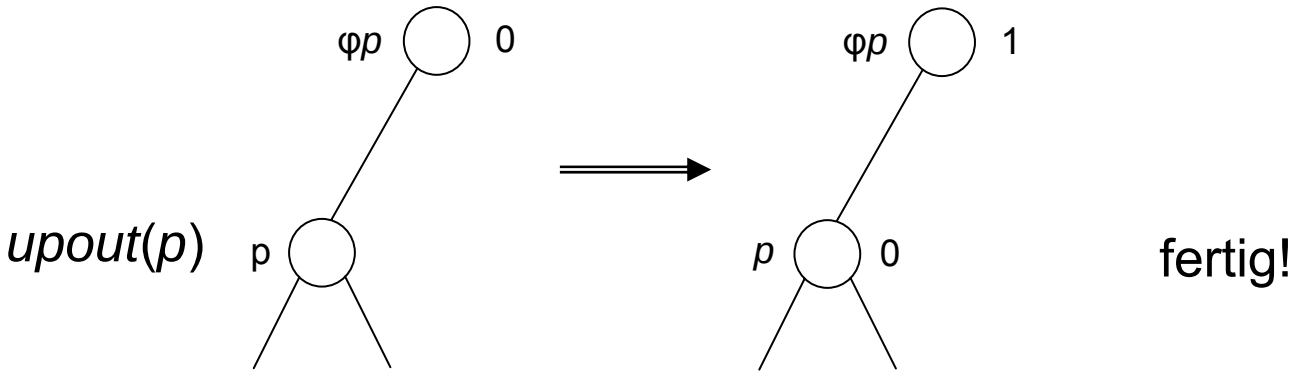


Fall 1.1: *p* ist linker Nachfolger von φ*p* und *bal*(φ*p*) = -1



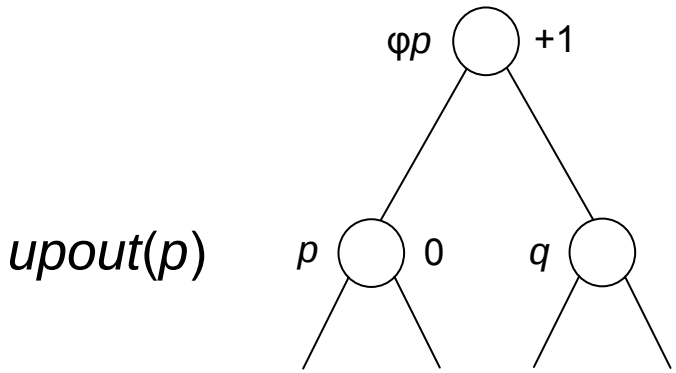
- Da die Höhe des Teilbaums mit Wurzel *p* um 1 gesunken ist, ändert sich die Balance von φ*p* zu 0.
- Damit ist aber die Höhe des Teilbaums mit Wurzel φ*p* um 1 gesunken und wir müssen *upout*(φ*p*) aufrufen (die Invariante gilt jetzt für φ*p*!).

Fall 1.2: *p* ist linker Nachfolger von φ*p* und *bal*(φ*p*) = 0



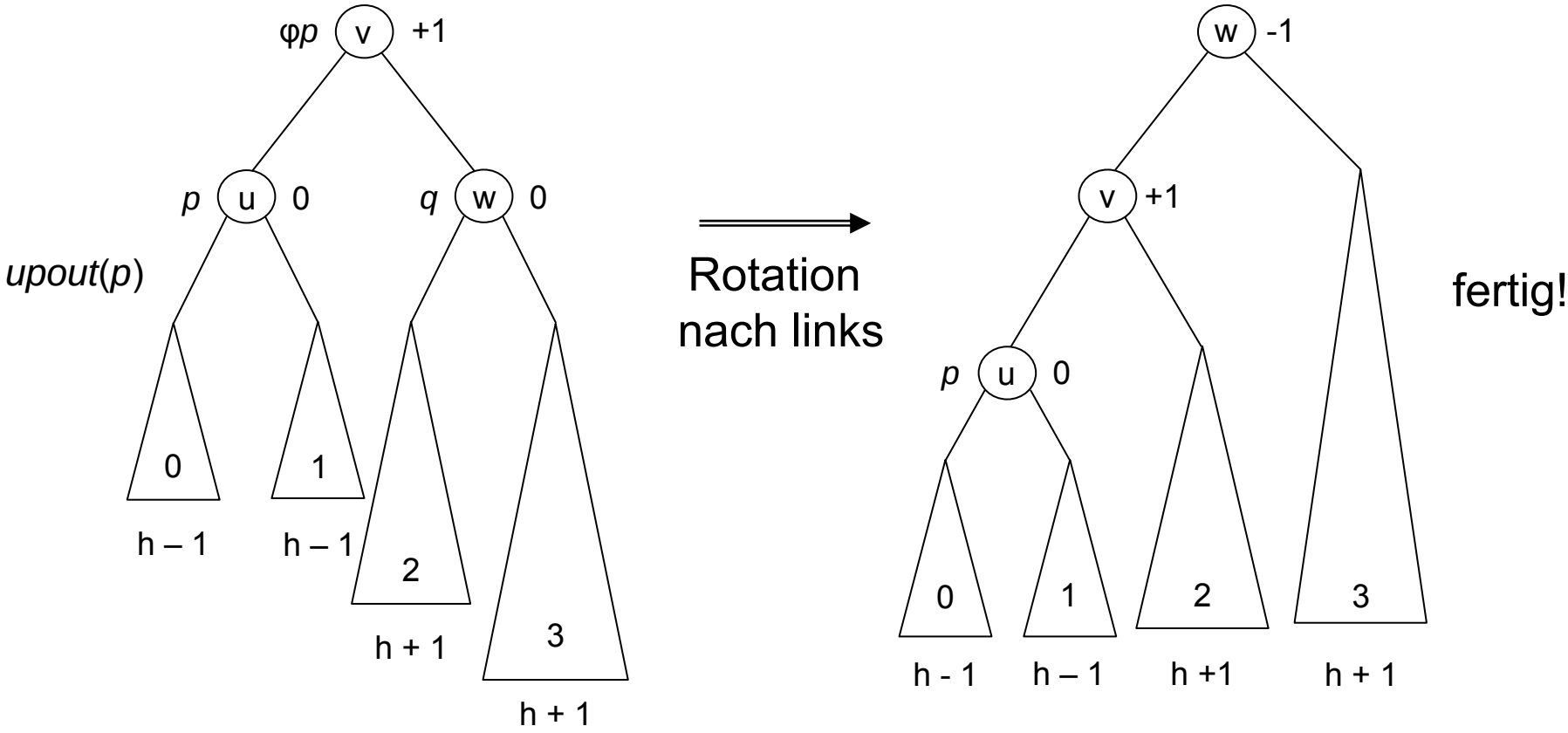
- Da sich die Höhe des Teilbaums mit Wurzel *p* um 1 verringert hat, ändert sich die Balance von φ*p* zu 1.
- Anschließend sind wir fertig, weil sich die Höhe des Teilbaums mit Wurzel φp nicht geändert hat.

Fall 1.3: *p* ist linker Nachfolger von φ*p* und *bal*(φ*p*) = +1

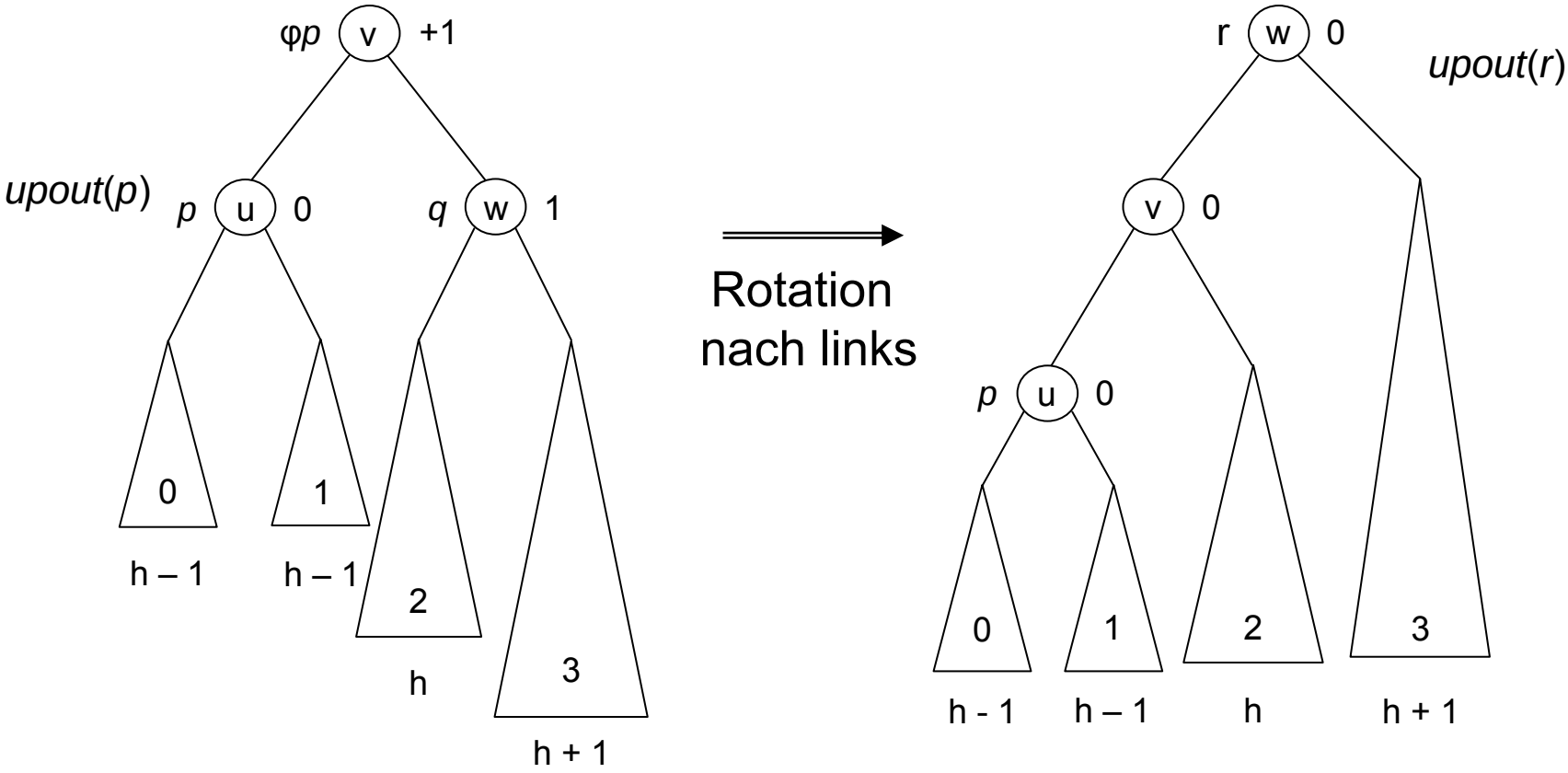


- Der rechte Teilbaum von φ*p* war also vor der Löschung bereits um 1 größer als der linke.
- Somit ist jetzt in dem Teilbaum mit Wurzel φ*p* die AVL-Baum-Eigenschaft verletzt.
- Wir unterscheiden drei Fälle entsprechend dem Balancegrad von *q*.

Fall 1.3.1: *bal*(*q*) = 0

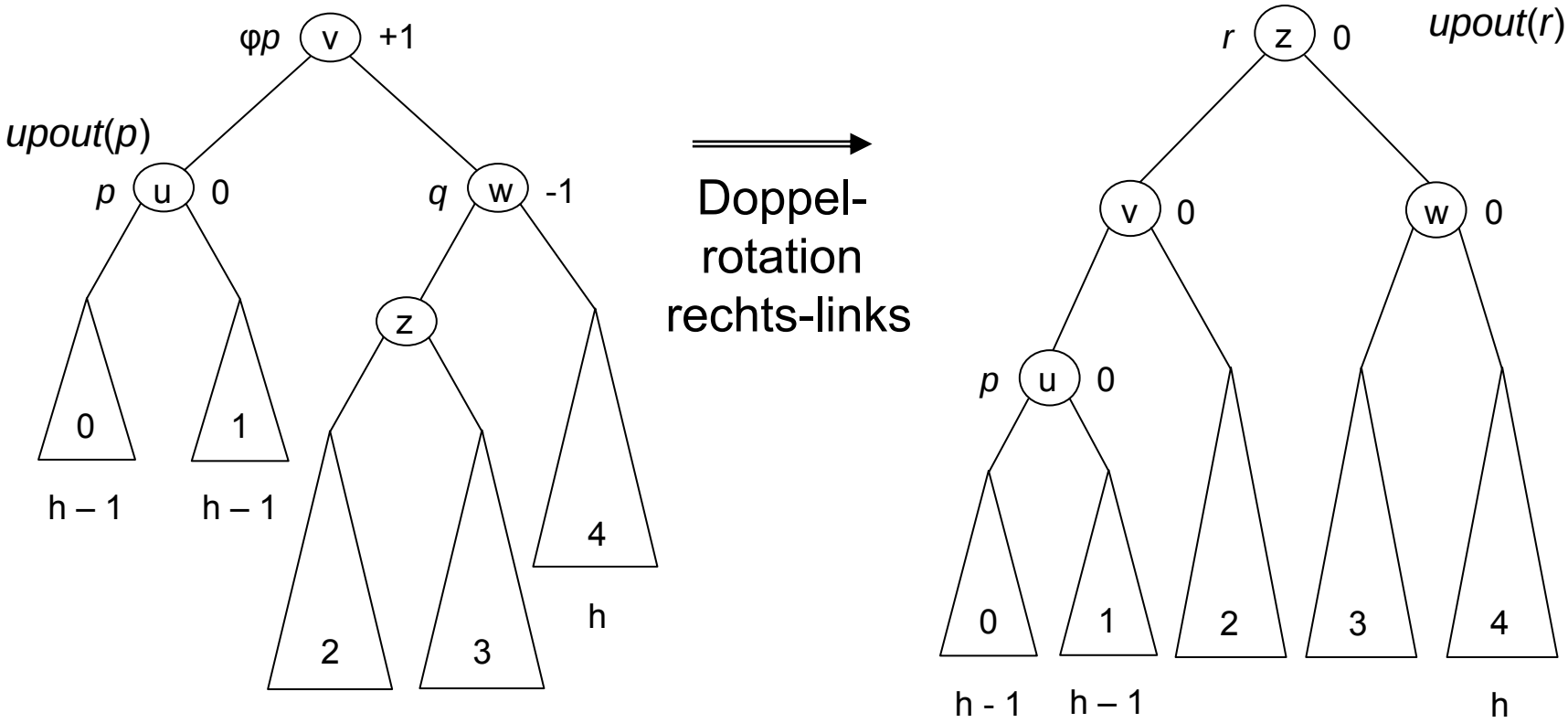


Fall 1.3.2: *bal*(*q*) = +1



- Erneut hat sich die Höhe des Teilbaums um 1 verringert, wobei *bal*(*r*) = 0 (Invariante).
- Wir rufen also *upout*(*r*) auf.

Fall 1.3.3: *bal*(*q*) = -1



- Wegen *bal*(*q*) = -1 muss einer der Bäume 2 oder 3 die Höhe *h* besitzen.
- Deswegen ist auch in diesem Fall die Höhe des gesamten Teilbaums um 1 gefallen, wobei gleichzeitig *bal*(*r*) = 0 gilt (Invariante).
- Wir rufen also wieder *upout*(*r*) auf.

Beobachtungen

- Anders als beim Einfügen, kann es beim Löschen vorkommen, dass auch nach einer Doppelrotation die Methode *upout* rekursiv aufgerufen werden muss.
- Daher reicht i.Allg. eine einzelne Rotation oder Doppelrotation nicht aus, um den Baum wieder auszugleichen.
- Man kann Beispiele konstruieren, in denen an allen Knoten entlang des Suchpfads Rotationen oder Doppelrotationen ausgeführt werden müssen.
- Wegen $h \leq 1.44 \ldots \log_2(n) + 1$ gilt aber, dass das Entfernen eines Schlüssels aus einem AVL-Baum mit n Schlüsseln in höchstens $O(\log n)$ Schritten ausführbar ist.
- AVL-Bäume sind eine *worst-case-effiziente* Datenstruktur für das Suchen, Einfügen und Löschen von Schlüsseln.