Balancierte Bäume

Martin Wirsing

in Zusammenarbeit mit
Moritz Hammer und Axel Rauschmayer

http://www.pst.ifi.lmu.de/lehre/SS06/infoII/

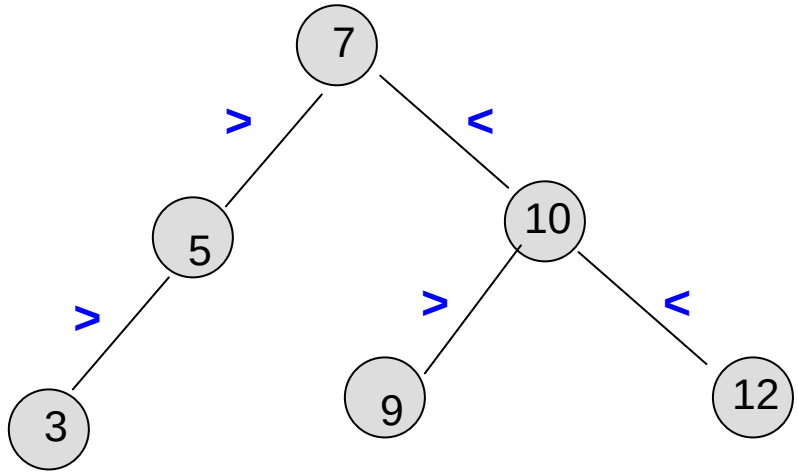
Ziele

- AVL-Bäume als einen wichtigen Vertreter balancierter Bäume kennen lernen
- Realisierung der Mengen-Operationen „einfügen“, „löschen“ und der „enthalten sein“-Abfrage verstehen

Wiederholung: Suchbäume (Geordnete Binärbäume)

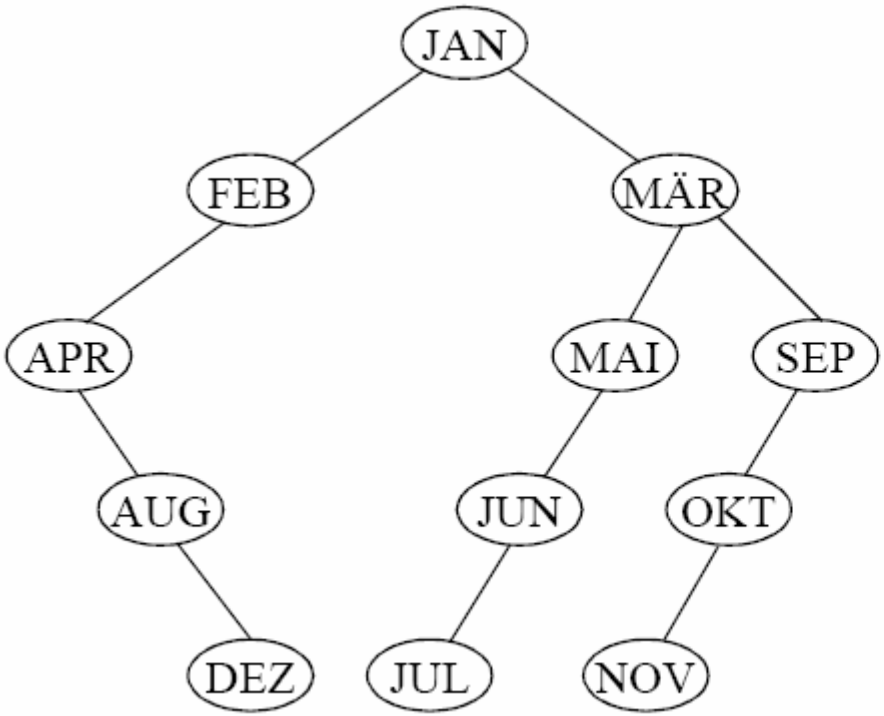
- Ein Binärbaum b heißt **Suchbaum**, wenn
  - b **leer** ist oder wenn
  - Folgendes **für alle nichtleeren Teilbäume t** von b gilt:

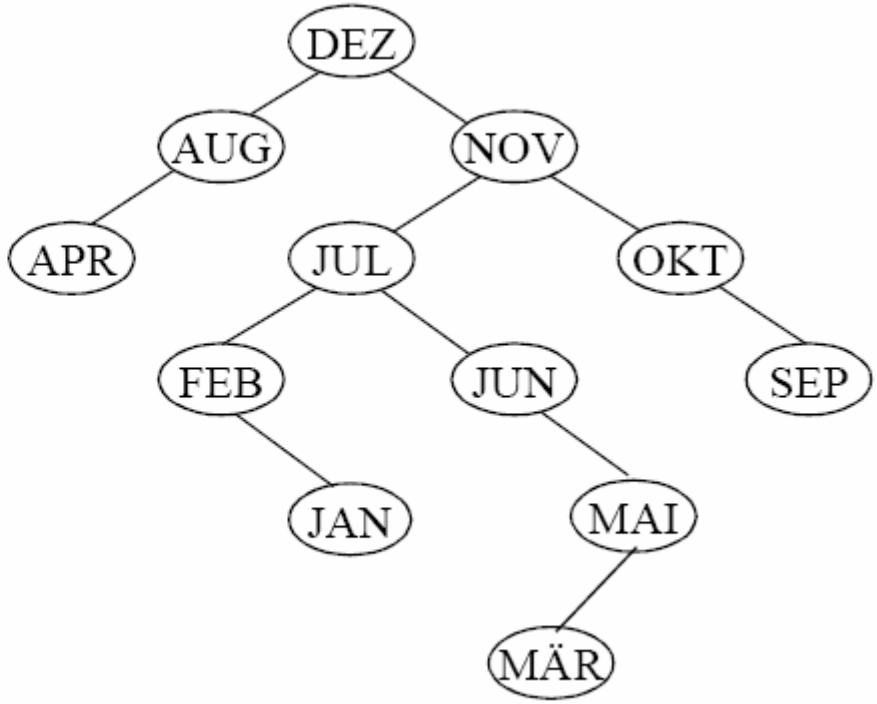
    Der Schlüssel von t ist
    - **größer** (oder gleich) **als alle Schlüssel des linken Teilbaums** von t und
    - **kleiner** (oder gleich) **als alle Schlüssel des rechten Teilbaums** von t



Beispiele: Suchbäume

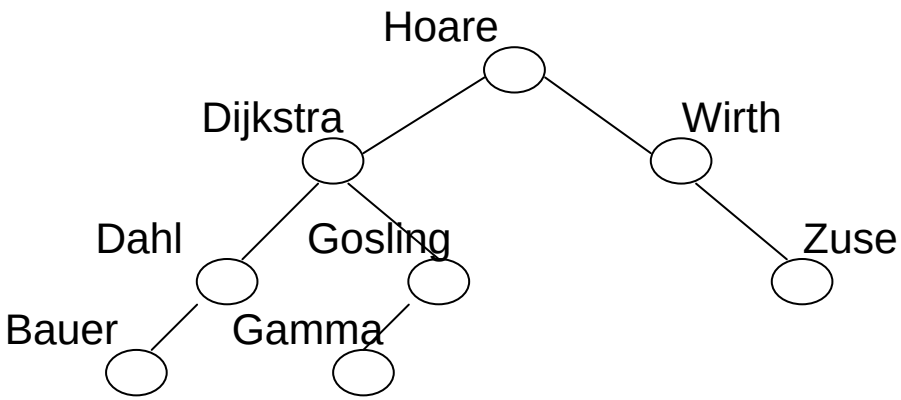
- **Beispiel:** Zwei verschiedene binäre Suchbäume über den Monatsnamen:

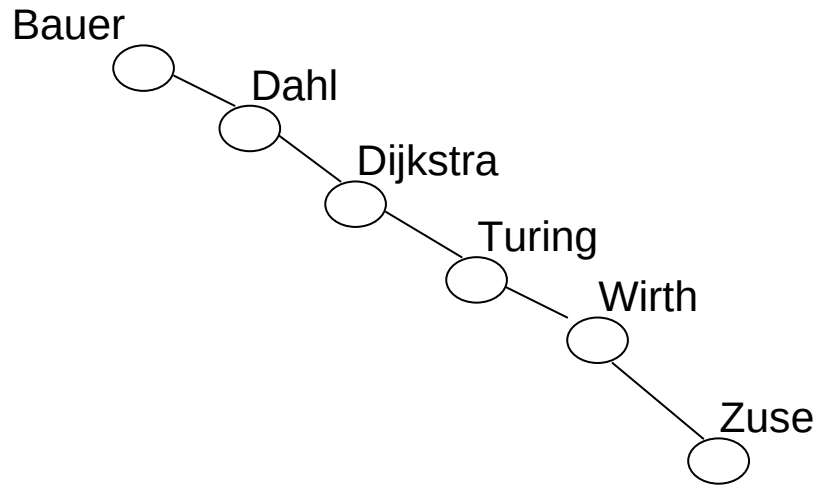




- **Nach welchen Kriterien (Vergleichsoperationen) sind diese Bäume geordnet?**

Beispiele: Suchbäume

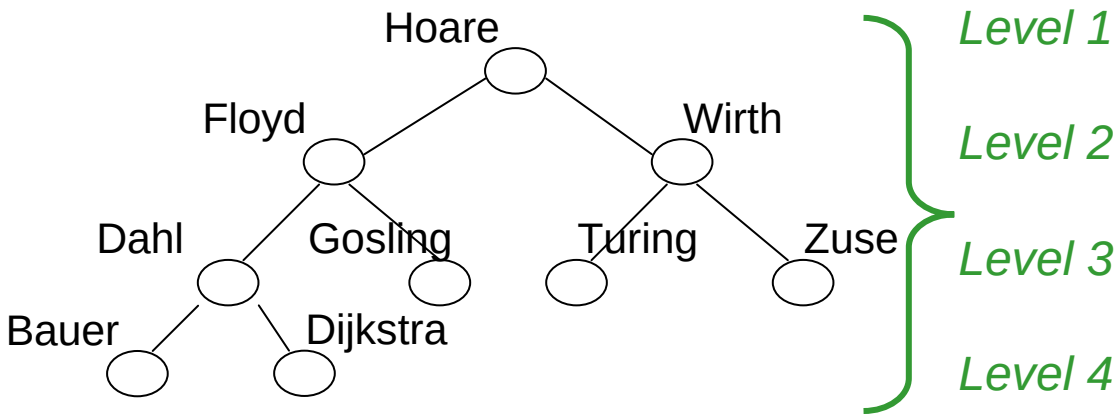




Einfügereihenfolge: Hoare, Dijkstra, Gosling, Wirth, Zuse, Gamma, Dahl, Bauer

Einfügereihenfolge: Bauer, Dahl, Dijkstra, Turing, Wirth, Zuse

Beispiele: Suchbäume



Ein **vollständig ausgeglichener Baum**

Einfügereihenfolge: Hoare, Floyd, Dahl, Dijkstra, Wirth, Zuse, Turing, Bauer, Gosling

Vollständig ausgeglichene binäre Suchbäume

- Ein **vollständig ausgeglichener binärer Suchbaum** ist ein
  - binärer Suchbaum, bei dem

    - abgesehen von der untersten Schicht -

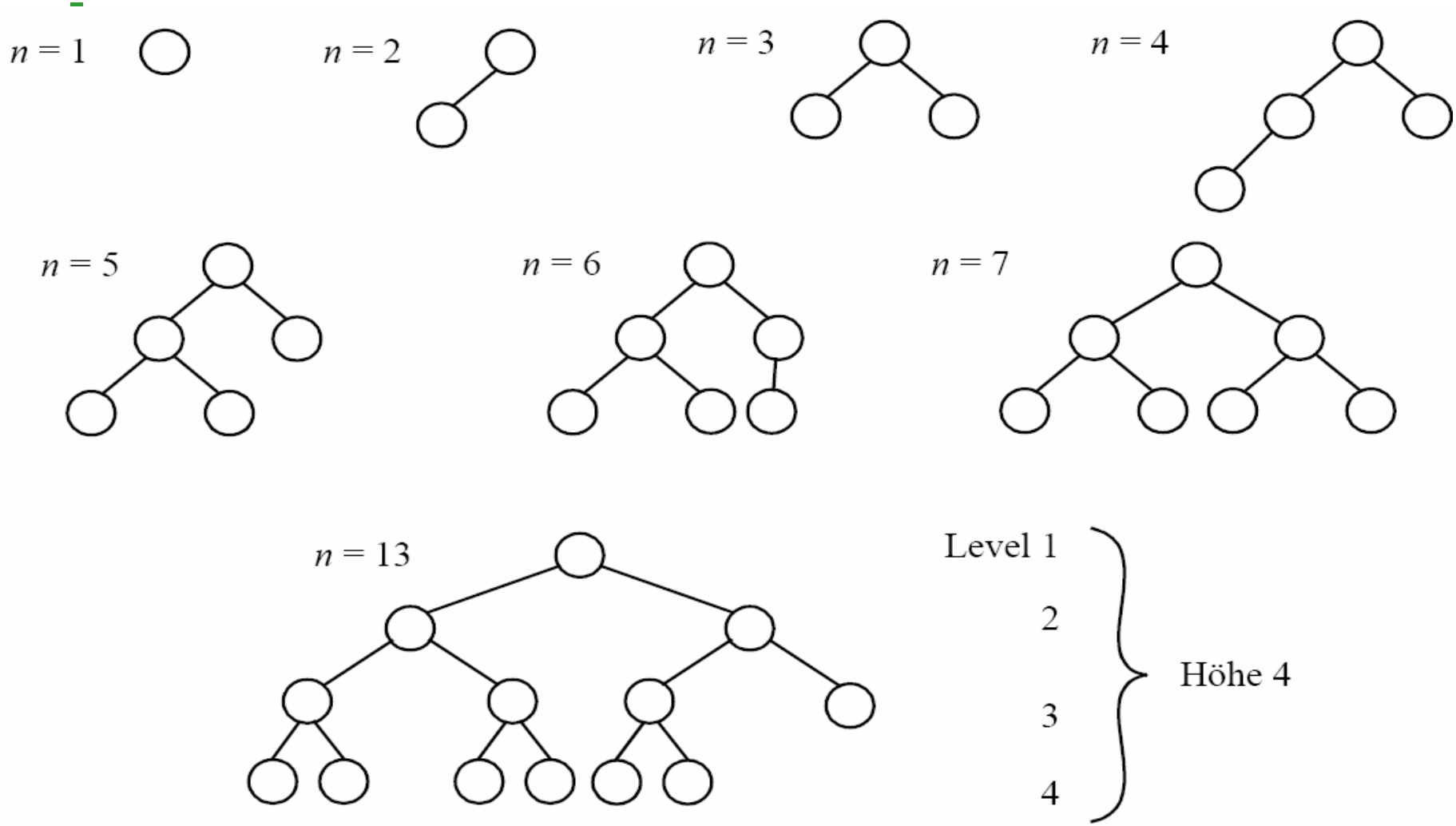
    alle Levels vollständig besetzt sind.
- Die Höhe eines vollständig ausgeglichenen Suchbaums beträgt $\log_2(n+1)$

Dies ist die minimale Höhe für alle Suchbäume:

➔ Optimaler Zeitaufwand für Suche

Vollständig ausgeglichene binäre Suchbäume

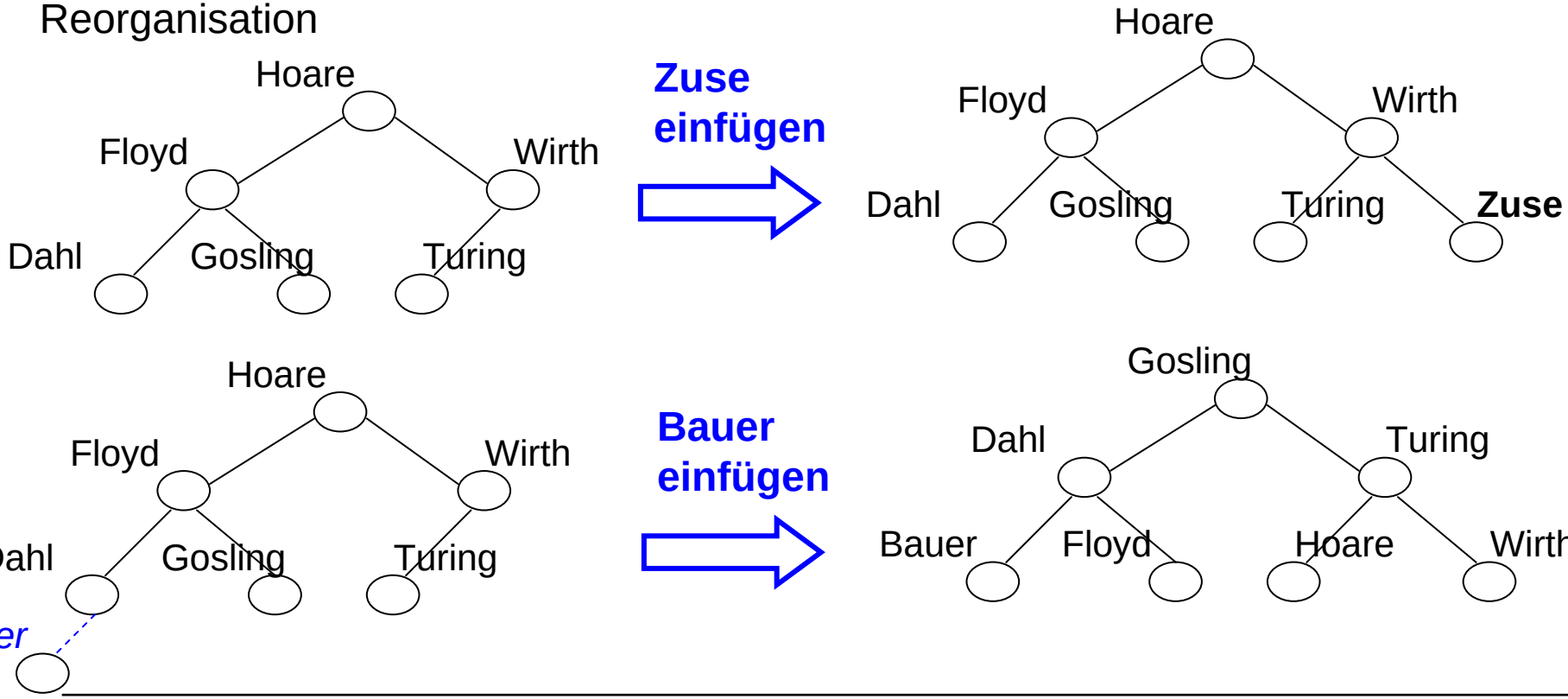
- **Beispiele:** Vollständig ausgeglichene binäre Suchbäume der Höhen 1-4



Vollständig ausgeglichene binäre Suchbäume

Problem: Beim Einfügen eines Elements muss ein vollständiger Suchbaum möglicherweise vollständig reorganisiert werden.

Beispiel: Einfügen von Zuse unproblematisch; Einfügen von Bauer führt zu vollständiger Reorganisation



Vollständig ausgeglichene binäre Suchbäume

Problem: Beim Einfügen eines Elements muss ein vollständiger Suchbaum möglicherweise vollständig reorganisiert werden.

➔ Einfügezeit im schlechtesten Fall: *O(n)*

➔ Kompromißlösung nötig mit:

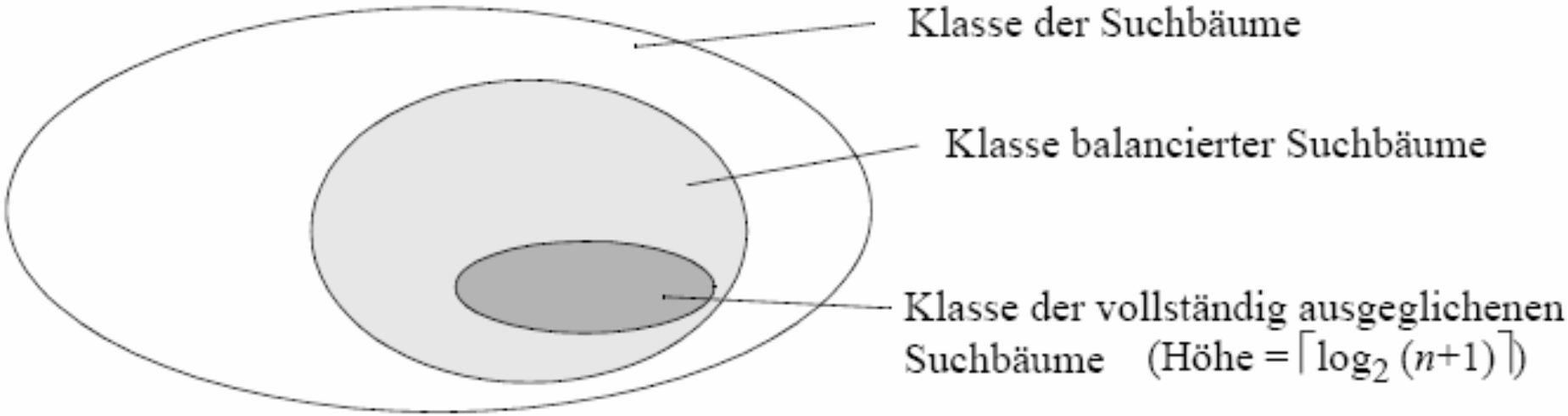
- Höhe des Baumes ist im schlechtesten Fall *O(log(n)) .*
- Reorganisation bleibt auf einen Suchpfad beschränkt und ist damit im schlechtesten Fall in Zeit *O(log(n))* ausführbar.

Balancierte Suchbäume

Eine Klasse von Suchbäumen heißt **balanciert**, falls:

- $h_{max} = O(n \log(n))$
- die Operationen *Suchen*, *Einfügen* und *Entfernen* sind auf einen Pfad von der Wurzel zu einem Blatt beschränkt und

benötigen damit im schlechtesten Fall $O(n \log(n))$ Zeit.



AVL-Bäume

- Ein binärer Suchbaum heißt **AVL-Baum**, falls für den linken Teilbaum *T1* und den rechten Teilbaum *T2* der Wurzel gilt:
  - $|h(T2) - h(T1)| <= 1$
  - *T1* und *T2* sind ihrerseits AVL-Bäume.
- Der Wert $|h(T1) - h(T2)|$ wird als **Balancefaktor** (BF) eines Knotens bezeichnet.
  Er kann in einem AVL-Baum nur die Werte -1, 0 oder 1 (dargestellt durch -, = und +) annehmen.
- Jeder AVL-Baum ist ein binärer Suchbaum.

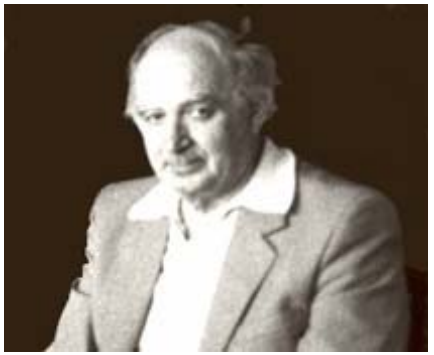
AVL-Bäume

- Ein binärer Suchbaum heißt **AVL-Baum**, falls für die beiden Teilbäume *T1* und *T2* der Wurzel gilt:
  - $|h(T1) - h(T2)| <= 1$
  - *T1* und *T2* sind ihrerseits AVL-Bäume.
- Der Wert $|h(T1) - h(T2)|$ wird als **Balancefaktor** (BF) eines Knotens bezeichnet. Er kann in einem AVL-Baum nur die Werte -1, 0 oder 1 (dargestellt durch -, = und +) annehmen.
- Jeder AVL-Baum ist ein binärer Suchbaum.
- Strukturverletzungen durch Einfügen oder Entfernen von Schlüsseln erfordern **Rebalancierungsoperationen.**
- Die **minimale Höhe** eines AVL-Baumes mit n Schlüsseln ist $\log_2(n+1)$.
- Die **maximale Höhe** eines AVL-Baumes mit *n* Schlüsseln ist *O(log n).*
- Im **Durchschnitt** ist ein AVL-Baum **ca. 44% höher** als ein vollständig ausgeglichener Baum.

AVL-Bäume: Entstehung

George Adelson-Velsky *1922 , jetzt in Israel Forscher am ITEP (Moskau Inst. für Theor. u. Exper. Physik) Schachcomputer Pionier Gewann 1967 3:1 im Computerschach gegen John McCarthy

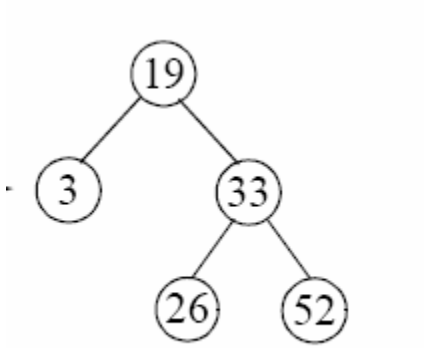
- AVL Bäume sind benannt nach den russischen Mathematikern G.M. Adelson-Velskii und E. M. Landis, die diese schwächere Definition von Ausgeglichenheit eines Baums 1962 aufstellten.
- G. M. Adelson-Velskii und E. M. Landis, *Doklady Nauk SSSR* 146, S. 263-266, 1962; Englische Übersetzung in *Soviet Math* 3, S. 1259-1263

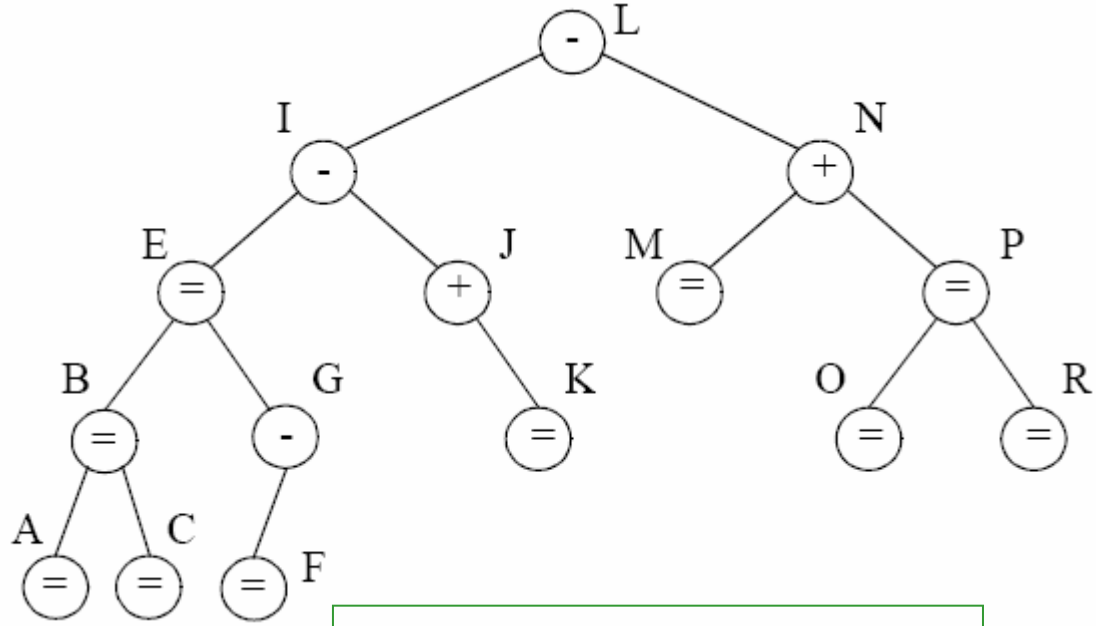


Evgenii Mikhailovich Landis 1921-1997 Diss. 1953, ITEP und Professor der Moskauer Staatsuni.

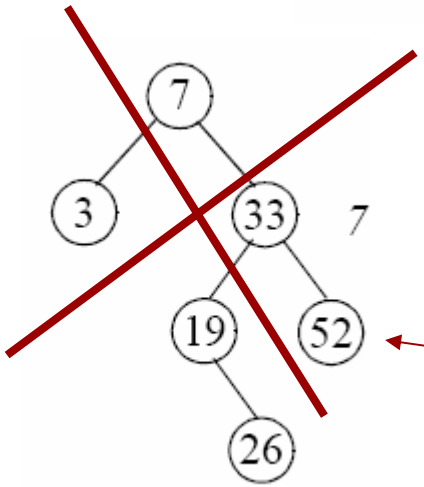
AVL-Bäume

- **Beispiele**:





Mit Angabe der Balancefaktoren



KEIN AVL-Baum

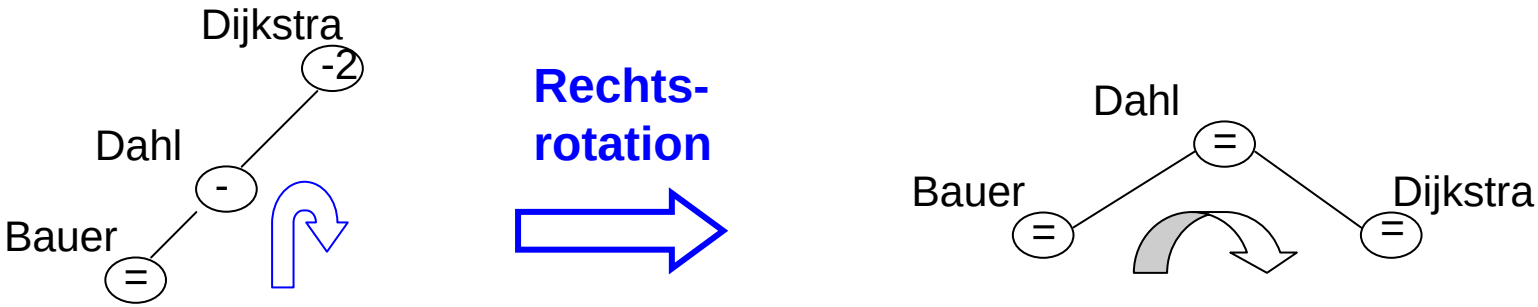
Einfügen in AVL-Baum

Einfügen von Dahl:
Neuberechnung des Balancierungsfaktors, AVL Kriterium erfüllt

Einfügen in AVL-Baum

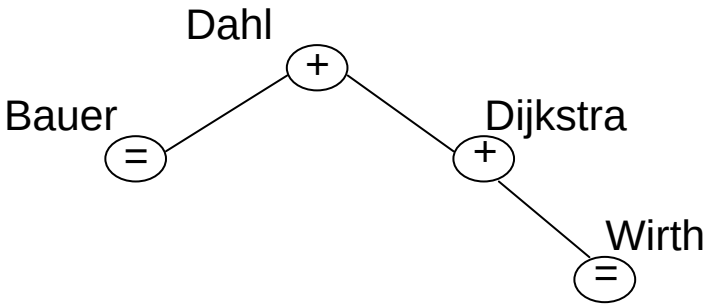
Einfügen von Dahl:
Neuberechnung des Balancierungsfaktors, AVL Kriterium erfüllt

Einfügen von Bauer: Verletzung des AVL Kriteriums



Nach Rechtsrotation: AVL Kriterium wieder erfüllt

Einfügen in AVL-Baum

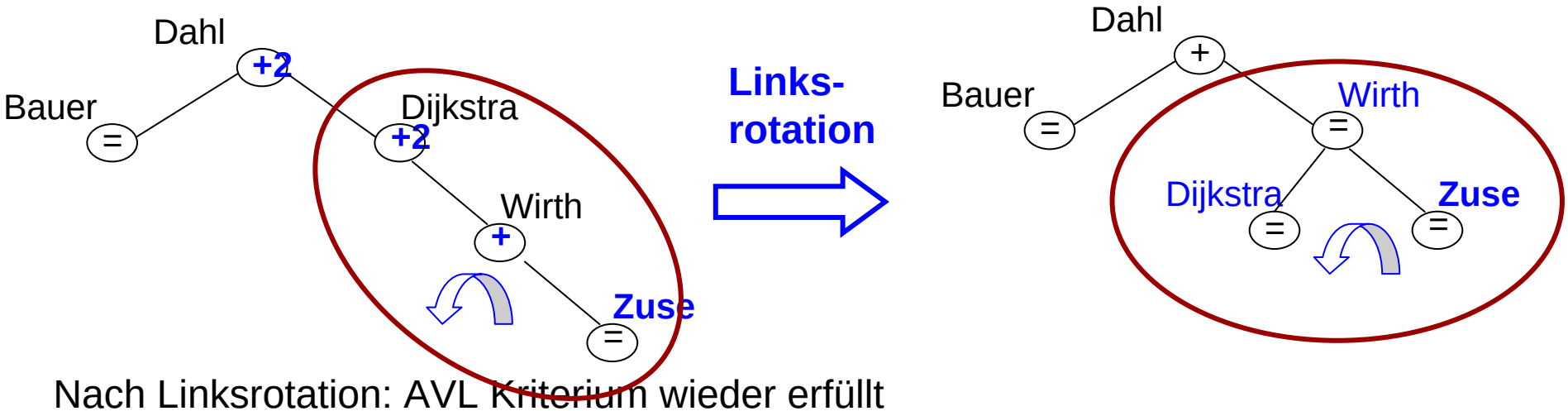


Einfügen von Wirth:

Nach Neuberechnung des Balancierungsfaktors ist

AVL-Kriterium weiter erfüllt

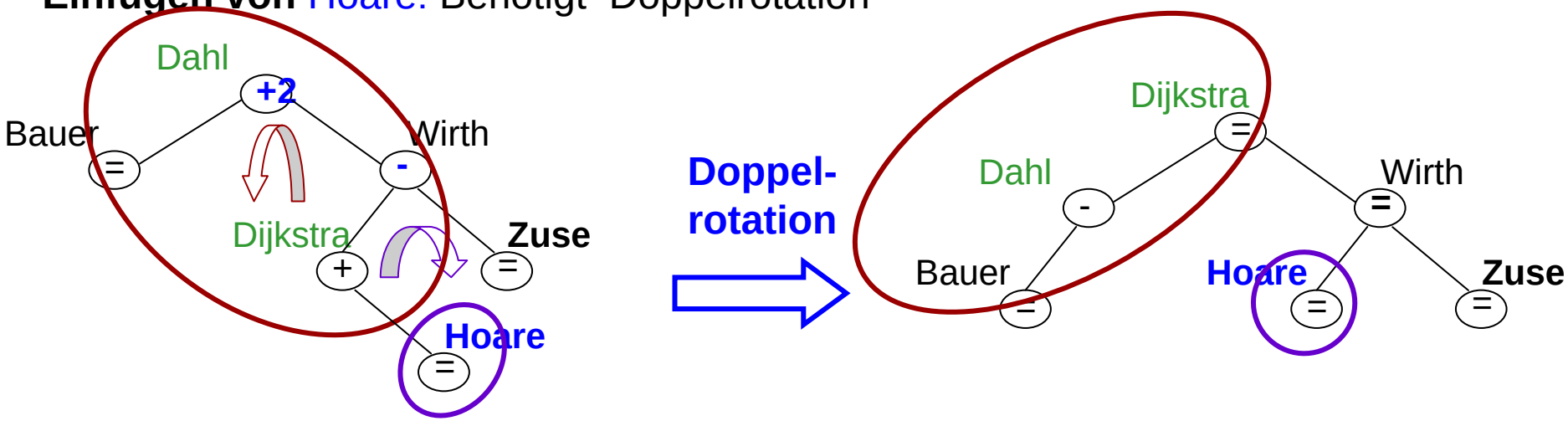
Einfügen von Zuse:



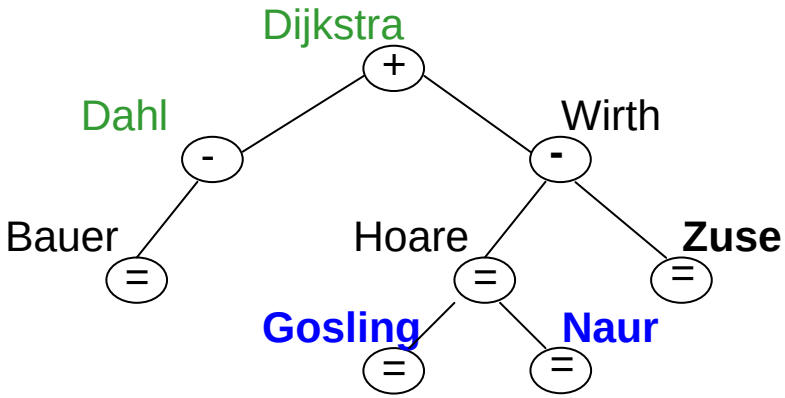
Nach Linksrotation: AVL Kriterium wieder erfüllt

Einfügen in AVL-Baum

Einfügen von Hoare: Benötigt Doppelrotation

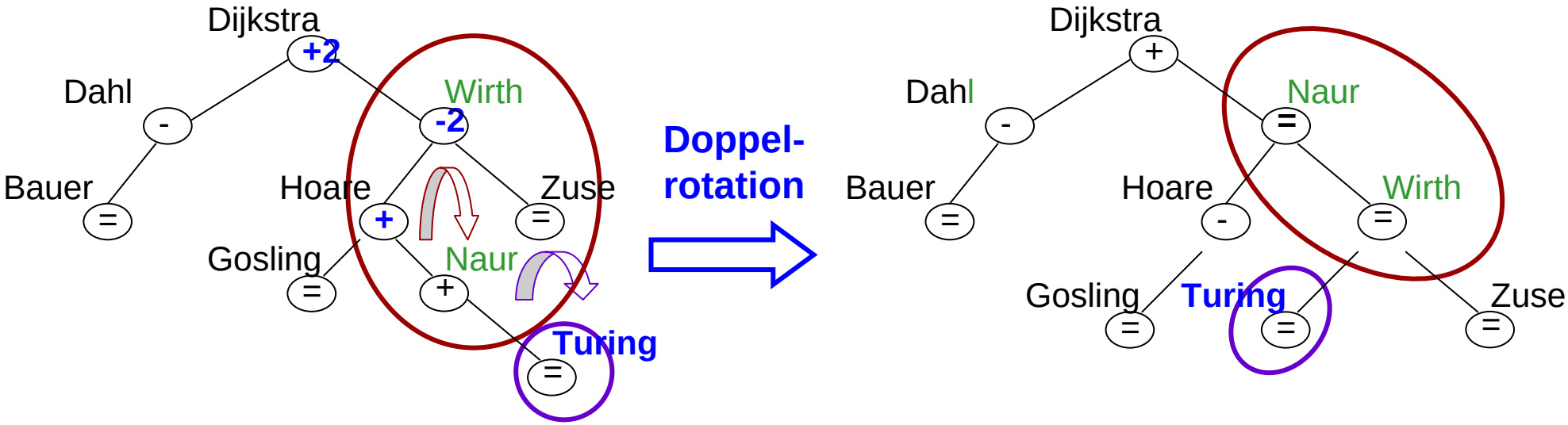


Einfügen von Gosling und Naur: ohne Probleme …



Einfügen in AVL-Baum

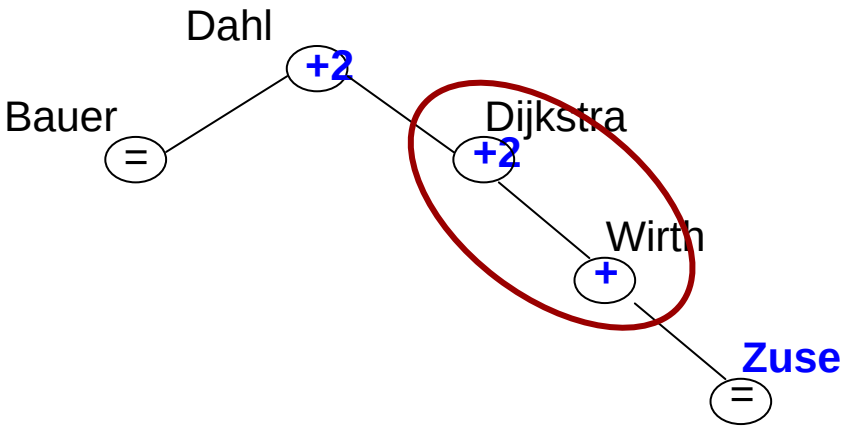
Einfügen von Turing: Noch einmal Doppelrotation

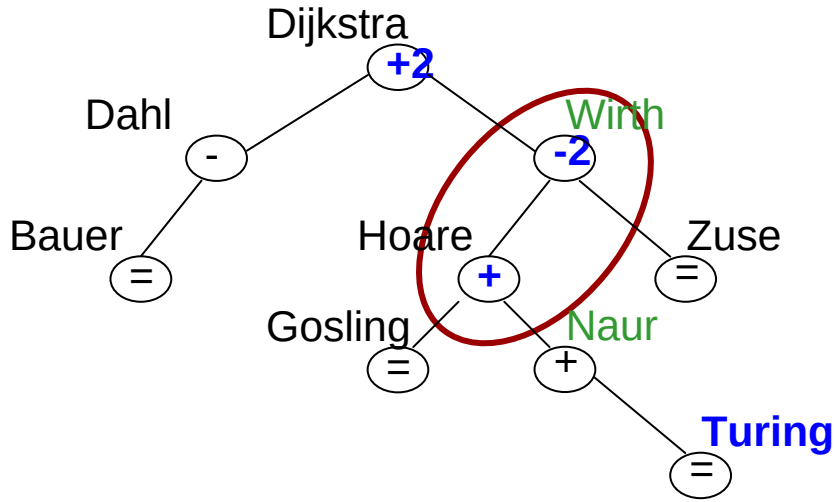


- Doppelrotation stellt AVL Kriterium wieder her
- Sind die vorgestellten Rotationen ausreichend?

Anwendungsstelle der Rotation

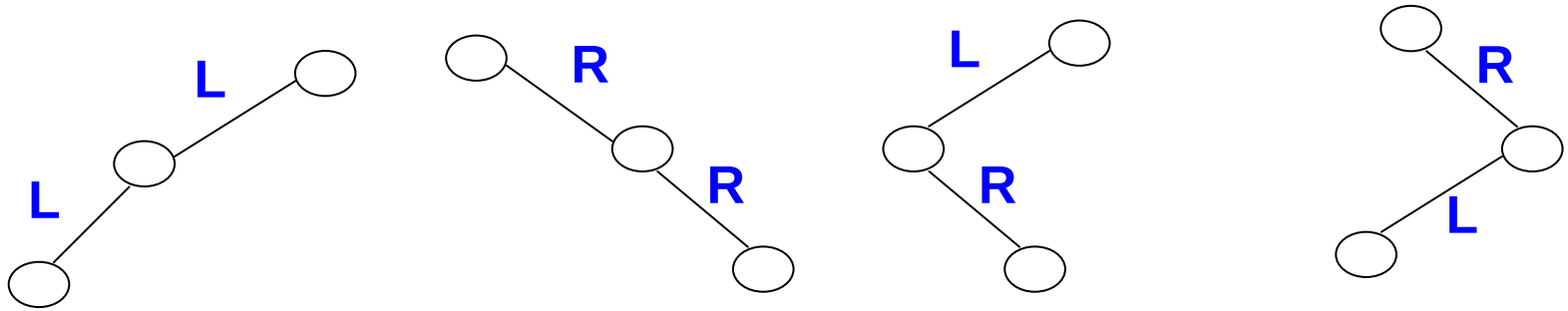
- Veränderungen der Balancierungsfaktoren geschehen ausschließlich auf dem **Pfad von der Wurzel zur Einfügeposition**
- Ausgangspunkt der Rotation ist immer der „**tiefste“ Elternknoten mit BF = ±2** (dieser Knoten hatte vorher BF = ±1)
- Der (auf dem Pfad) **darunter liegende Knoten** hat **BF = ±1**





Rotationstypen

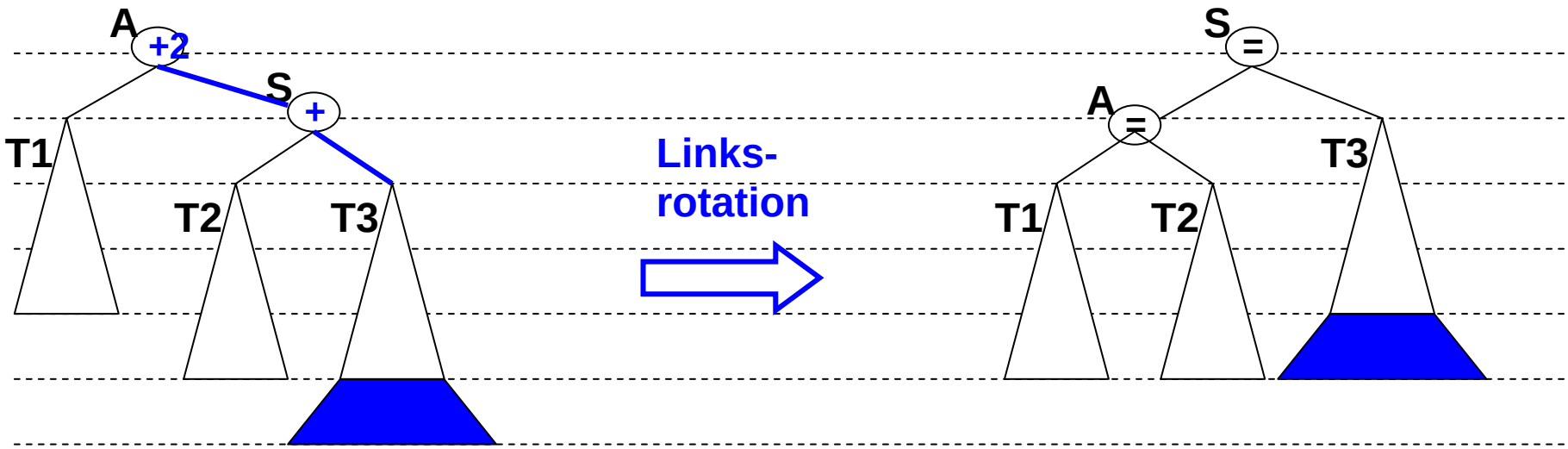
- Betrachte ausgehend vom tiefsten Knoten mit BF = 2 den Pfad zur Einfügeposition:
  - **RR: Rechts-Rechts** Linksrotation
  - **LL: Links-Links** Rechtsrotation
  - **RL: Rechts-Links** Doppelrotation „rechts“
  - **LR: Links-Rechts** Doppelrotation „links“



- Rotation ist immer eindeutig bestimmt
- Jetzt genauere Betrachtungen der einzelnen Typen

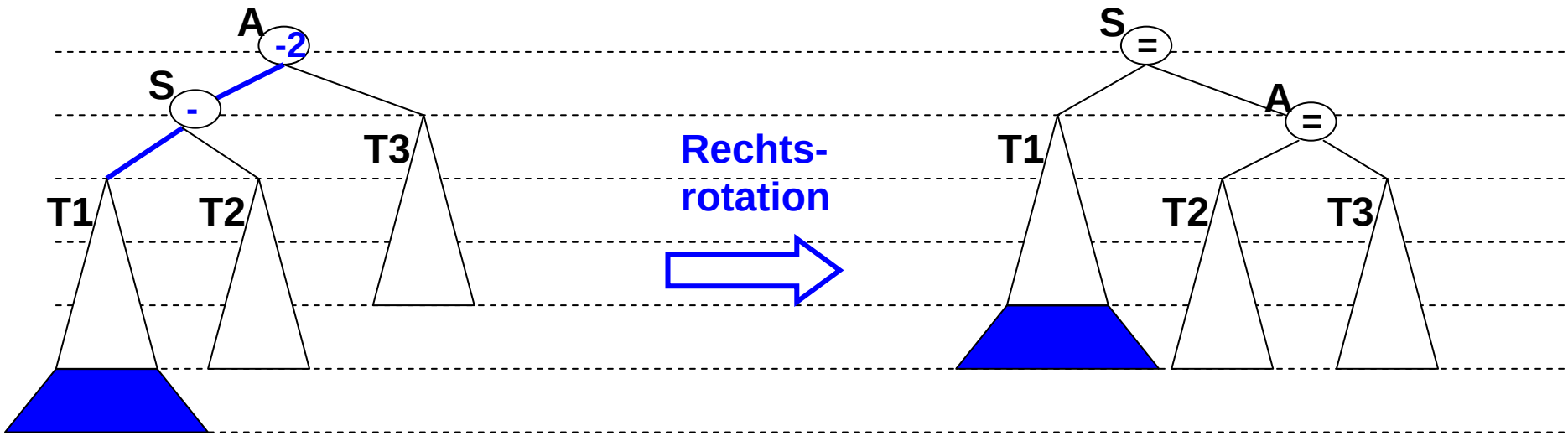
Typ RR: Linksrotation

- Wir bezeichnen den „tiefsten“ Knoten mit Strukturverletzung mit A, dessen Kind mit S und den Enkelknoten mit B
- Bei der Linksrotation hat S den BF „+“ und A den BF „+2“



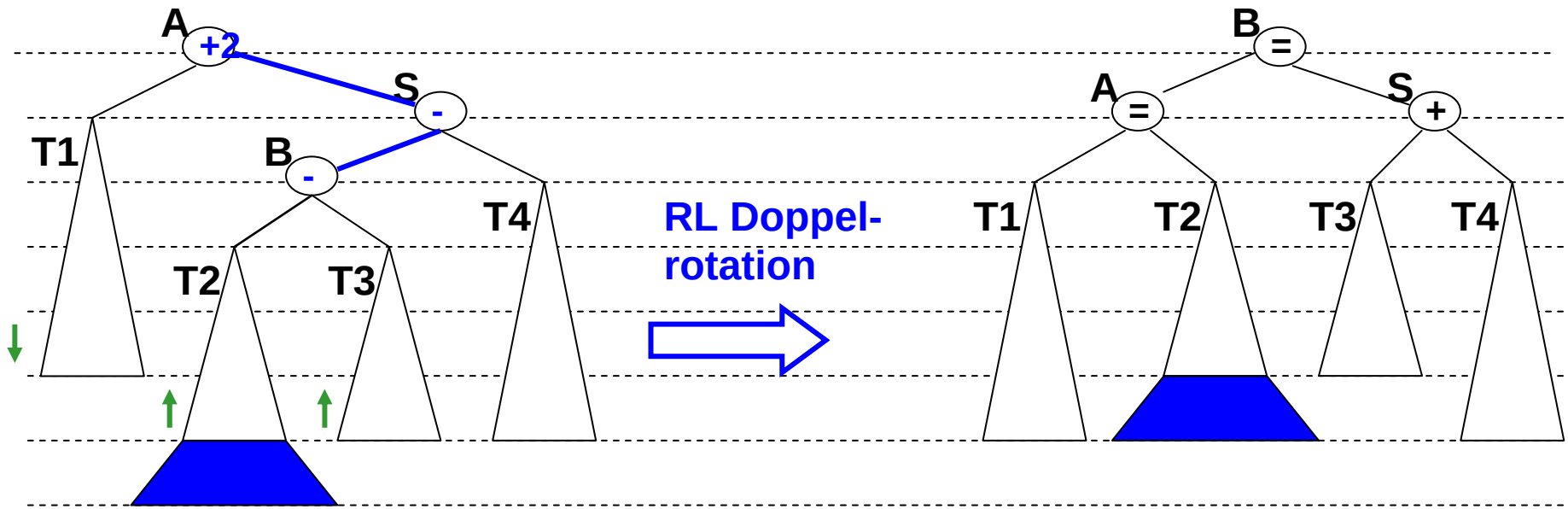
Typ LL: Rechtsrotation

- Bei der Rechtsrotation hat S den BF „-“ und A den BF „-2“



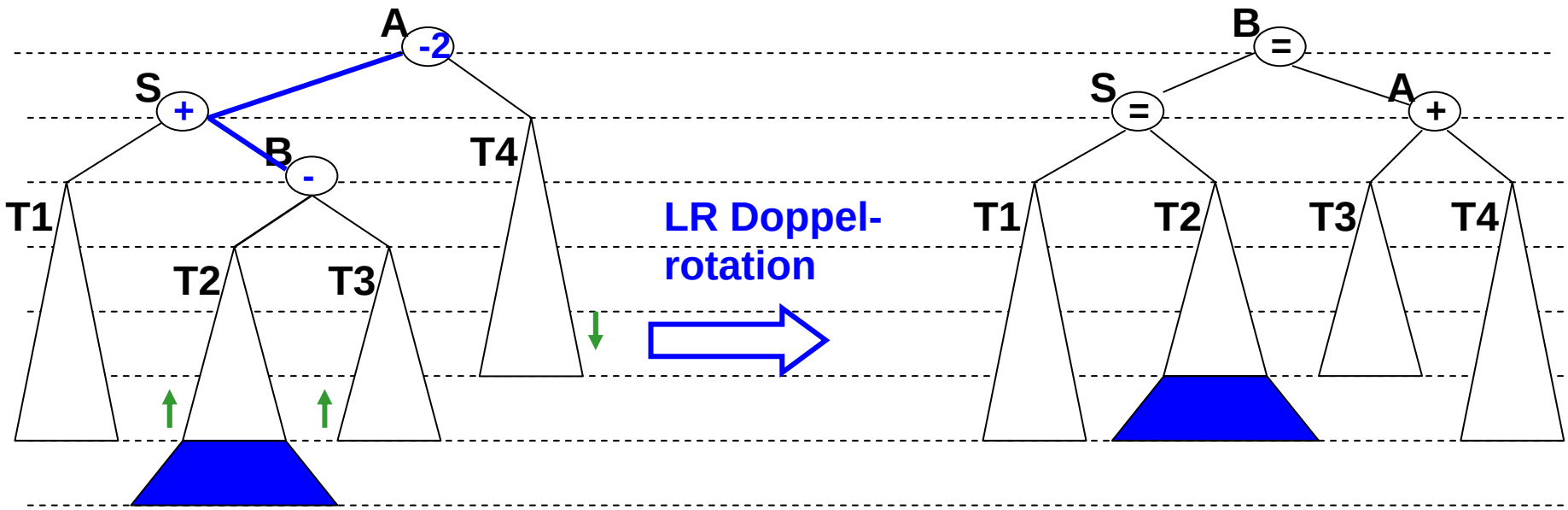
Typ RL: Doppelrotation

- Bei der RL-Doppelrotation hat A den BF „+2“, S den BF „-2“, B den BF „+“ oder „-“ .
- Wir wählen „-“ für den BF von B.



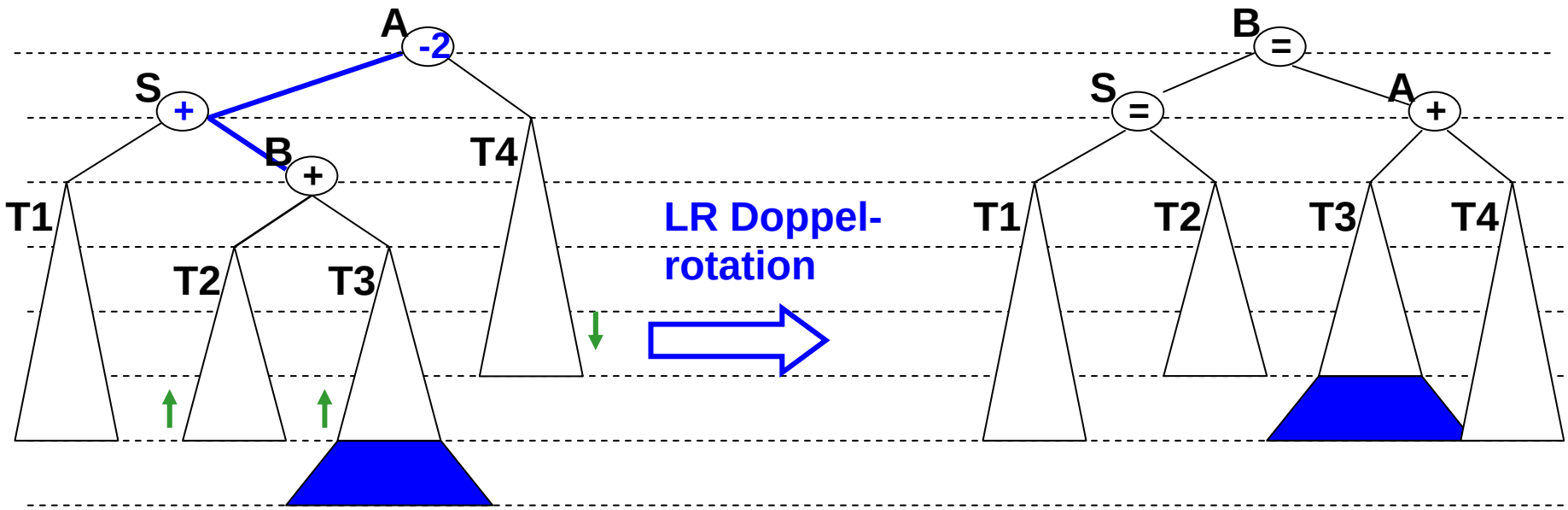
Typ LR: Doppelrotation

- Bei der LR-Doppelrotation hat A den BF „-2“, S den BF „+“, B den BF „+“ oder „-“ .
- Wir wählen „-“ für den BF von B.



Typ LR: Doppelrotation

- Variante der LR-Doppelrotation mit Balancefactor „+“ für B

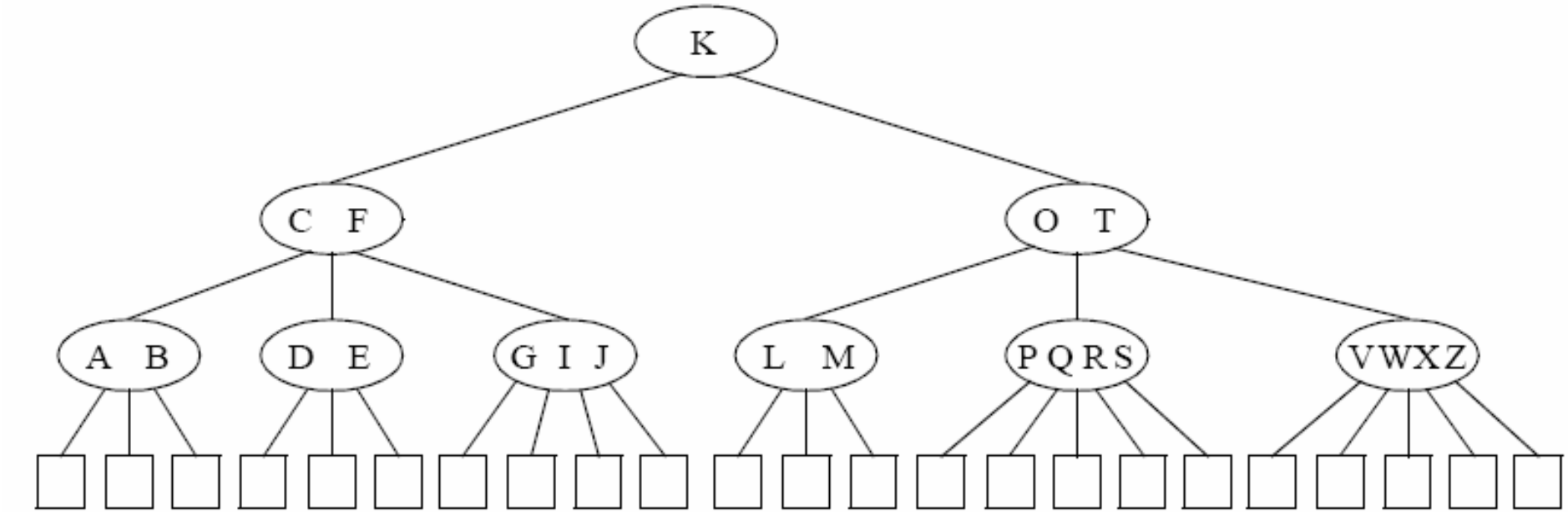


Löschen

- Das Löschen erfolgt wie bei Suchbäumen und kann (wie das Einfügen) zu Strukturverletzungen führen, die durch Rotationen ausgeglichen werden.
- Beim Löschen genügt nicht immer eine einzige Rotation oder Doppelrotation beseitigt. Im schlechtesten Fall muss auf dem Suchpfad bottom-up vom zu entfernenden Schlüssel bis zur Wurzel auf jedem Level eine Rotation bzw. Doppelrotation durchgeführt werden.

Weitere effiziente Baumstrukturen: (a,b)-Bäume

- ***Interner Suchbaum***: Information steht in den Knoten.
- ***Externer Suchbaum***: Information steht nur in den Blättern; Knoten tragen Verwaltungsinformation.



Weitere effiziente Baumstrukturen: (a,b)-Bäume

- Ein **(a,b)-Baum**, wobei b >= 2a - 1, ist ein **externer Suchbaum**, dessen interne Knoten außer der Wurzel einen **Rang (Kinderzahl) zwischen a und b** (einschließlich) haben.
- Die Wurzel hat mindestens 2 und höchstens b Kinder.
- Jeder interne Knoten vom Rang N enthält N-1 aufsteigend geordnete Zahlen $k_1, \ldots, k_{N-1}$ als Verwaltungsinformation:
  - Beim **Suchen** vergleicht man den Schlüssel k mit diesen ki.

    Es gilt: ist k überhaupt unterhalb des Knotens zu finden, so unterhalb des i-ten Kindes, falls $k_{i-1} <= k < k_i$.
  - Beim **Einfügen** und Entfernen von Einträgen muss man all diesen Bedingungen Rechnung tragen.

Weitere effiziente Baumstrukturen: B-Bäume

- Ein **B-Baum** ist ein (a, b)-Baum mit b = 2a - 1.
- Entwickelt 1972 von **Rudolf Bayer**, em. Prof. TU München, und Edward McCreight; führte zur 1. relationalen SQL-Datenbank **R** von IBM
- In den Anwendungen ist **a relativ groß**, z.B. 512. Dadurch reduziert sich die Höhe enorm gegenüber einem Binärbaum.
- Das ist sinnvoll, wenn so viele Daten zu verwalten sind, dass die Knoten nicht im Hauptspeicher Platz finden, sondern auf Festplatten gespeichert werden. Ein Plattenzugriff dauert sehr lange (bis zu 10000 Mal solange wie ein Hauptspeicherzugriff), kann aber gleich eine ganze “Seite” auslesen, also z.B. einen ganzen Knoten eines B-Baumes.
- Interessant ist auch der Spezialfall a = 2, b = 3; d.h. die Knoten haben 2 oder 3 Kinder. Solche Bäume kann man wiederum als Binärbäume codieren und erhält so die ***Rot-Schwarz-Bäume*** [siehe Vorlesung Effiziente Algorithmen].

Zusammenfassung

- Balancierte Bäume sind besondere binäre Suchbäume, die der zusätzlichen Invariante *Höhe* = O(log(*Knotenzahl*)) genügen. Dadurch laufen die Operationen Suchen, Einfügen, Löschen in logarithmischer Zeit.
- Vollständig ausgeglichen Suchbäume sind 1-balancierte Suchbäume. Beim Einfügen eines Elements muss ein vollständiger Suchbaum möglicherweise vollständig reorganisiert werden (Zeitkomplexität O(*Knotenzahl*)).
- AVL-Bäume sind 1-balancierte Bäume, bei denen die Operationen Suchen, Einfügen, Löschen logarithmische Zeitkompexität besitzen.
- Weitere effiziente Baumstrukturen sind (a,b)-Bäume mit den Spezialfällen der B-Bäume und Rot-Schwarz-Bäume.