# Balancierte Binär- und B-Bäume

- Sie kennen die Kriterien um die Ausgeglichenheit von Binär-Bäumen zu bestimmen.
- Sie können AVL Bäume implementieren.
- Sie wissen, was B-Bäume und Rot-Schwarz-Bäume sind und wie man sie einsetzt.

Basiert auf Material von:

Kurt Bleisch
Stephan Neuhaus
Karl Rege
Marcela Ruiz
Jürgen Spielberger

Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

# Suchen und Tiefe

# Suchen im binären Suchbaum

Suche x im Baum B:

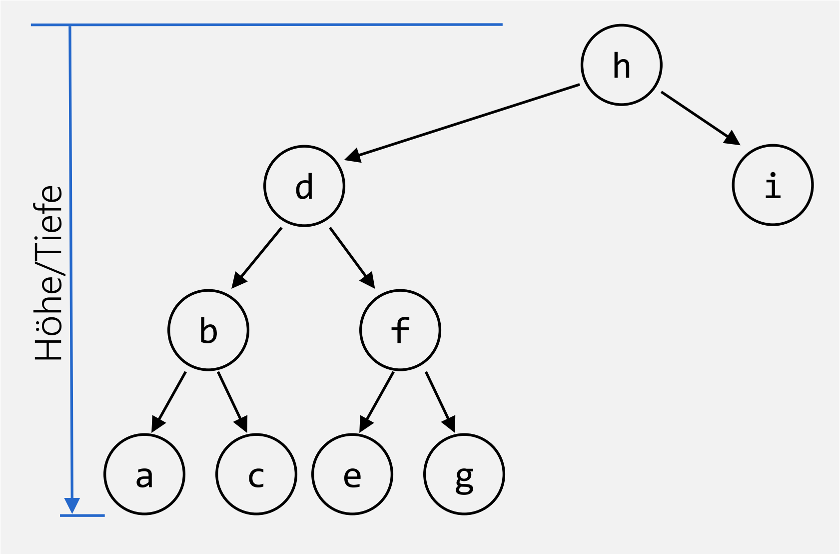
- Wenn x == Wurzelelement gilt, haben wir x gefunden.
- Wenn x > Wurzelelement  gilt, wird die Suche im rechten Teilbaum von B fortgesetzt, sonst im linken Teilbaum.

```
public Object search(TreeNode<T> node, T  x) {
   if ((node == null) return node;
   else if (x.compareTo(node.element) == 0)
      return node;
   else if (x.compareTo(node.element) <= 0)
      return search(node.left,x);
   else
      return search(node.right,x);
}
```

- Bei einem vollständigen Binärbaum müssen lediglich ~ $\text{Log}_2$ Schritte durchgeführt werden, bis Element gefunden wird.
- Entspricht Aufwand für binäres Suchen.
- sehr effizient Bsp.:
  1'000 -> 10 Schritte
  1'000'000 -> 20 Schritte

# Zugriffszeiten und Tiefe

- Die Zugriffszeit (Such- und Einfügezeit) von Elementen ist proportional zur Höhe/Tiefe des Baumes.
- Ziel: bei gegebener Anzahl Elemente ein Baum mit möglichst geringer Tiefe.
- Probleme:
  - neue Knoten können nur unten angehängt werden
  - einzufügende Elemente sind meist nicht à priori bekannt
  - bei «unglücklicher» Reihenfolge entstehen sehr ungleichmässige, d.h. «unbalancierte» Bäume



# Balancieren

# Übung

Zeichnen sie alle möglichen sortierten Binärbäume der Knoten mit den Werten A,B,C auf

# Übung

Zeichnen Sie den sortieren Binärbaum auf, der beim Einfügen der Zeichenkette THEQUICKBROWN entsteht (Anfang des Satzes: «the quick brown fox jumps over the lazy dog»). Fügen Sie die Buchstaben in dieser Reihenfolge hinzu.

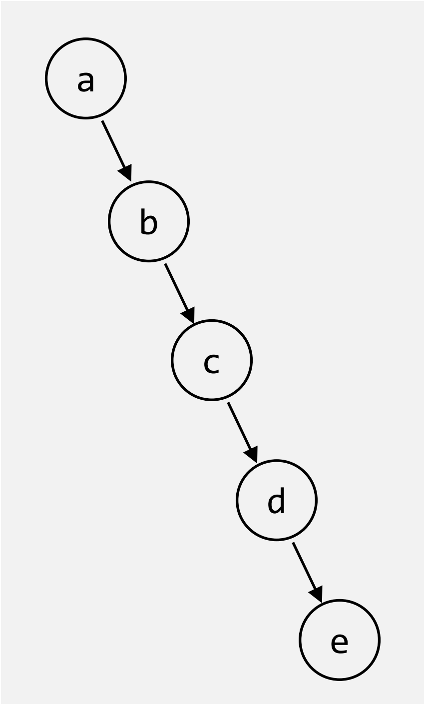
Englischsprachiges Pangram – ein Satz, der alle Buchstaben des englischen Alphabets enthält.

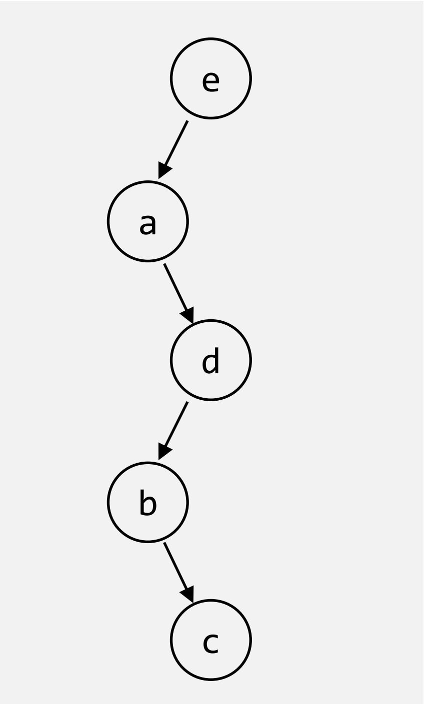
# Übung

Erstellen Sie einen optimal balancierten Baum mit den Buchstaben der Zeichenkette THEQUICKBROWN. Können Sie einen Algorithmus herleiten (für die korrekte Reihenfolge zum Einfügen)?

# Balanciertheit von Suchbäumen

Wenn man Daten in beliebiger Reihenfolge in einen Binärbaum 'naiv' (wie bisher 'einfach so') einfügt, werden die beiden Teilbäume vermutlich unterschiedlich schwer und unterschiedlich tief sein.



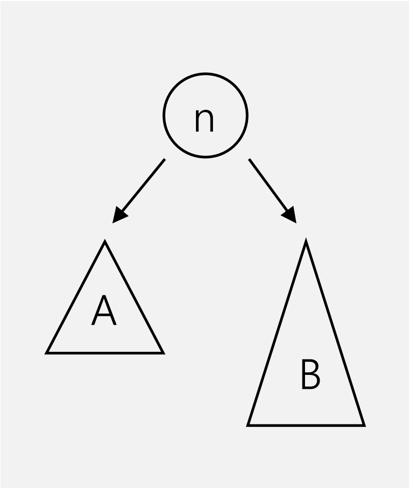

- Möglicher schlimmster Fall: Die Daten werden in sortierter Reihenfolge eingefügt (linker Baum). Der Baum degeneriert zur Liste.
- Zeitlicher Aufwand zum Suchen: O(n)
- Idee: wir fordern dass der Baum 'vollständig ausgeglichen' wird...

# Balanciertheit von Suchbäumen

Ein vollständiger Baum ist immer balanciert (siehe vorherige Lektion).

**Vollständig balancierter Baum:**

Ist ein Baum, bei dem, abgesehen von der untersten Ebenen, alle Ebenen vollständig (mit Knoten) besetzt sind.

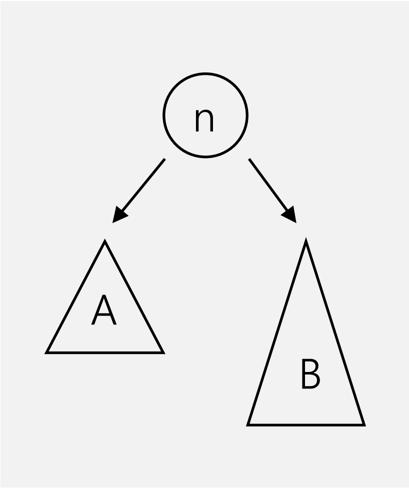



- Tiefe beträgt: $\log_2(n+1)$
- Man findet auch den Begriff: vollständig ausgeglichener Baum.
- Dies ist die ideale Tiefe für binäre Suchbäume, der zeitliche Aufwand zum Suchen ist optimal: $O(\log_2(n))$.
- Aber, beim Einfügen, Löschen und Ändern muss der Baum möglicherweise vollständig reorganisiert werden: O(n) – es gibt bessere Lösungen.
- Idee: Kompromisslösung, wir schwächen die Bedingung ab…

# Balanciertheit von Suchbäumen

**Balancierter Baum:**

Ist ein Baum, der eine maximale Höhe von $c_1 \cdot \log(n) + c_2$ garantiert.



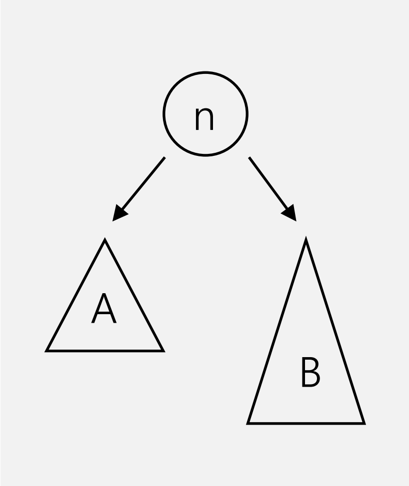

- Maximale Höhe: c1·log(n) + c2, c1 und c2 sind Konstanten.
- Es sind **unterschiedliche Regeln** möglich. Die Regeln können sich z.B. auf Höhen, Gewicht oder Struktur der Bäume beziehen.
- Es gibt **verschiedene Baumarten** und Algorithmen, die dieses Kriterium der maximalen Höhe erfüllen.

02 01

# Balanciertheit von Suchbäumen

**AVL-Baum:**

Der AVL-Baum ist ein balancierter Baum, bei dem für jeden Knoten gilt, dass sich die Höhe der beiden Teilbäume um höchstens eins unterscheidet.



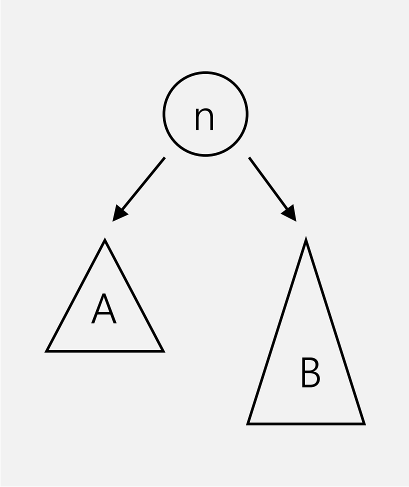

- Maximale Tiefe: $c_1 \cdot \log_2(n+2) + c_2$, $c_1 \approx 1.44$, $c_2 \approx -0.33$
- Etwa 44% höher als ein vollständig ausgeglichener Baum.
- Benannt nach den russischen Mathematikern G.M. Adelson-Velskii und E. M. Landis, entwickelt 1962.
- Beim Einfügen und Löschen sorgt man dafür, dass die AVL-Ausgleichbedingung erhalten bleibt.

Wir verzichten auf eine Herleitung ;-)

# Balanciertheit von Suchbäumen

**AVL-Baum:**

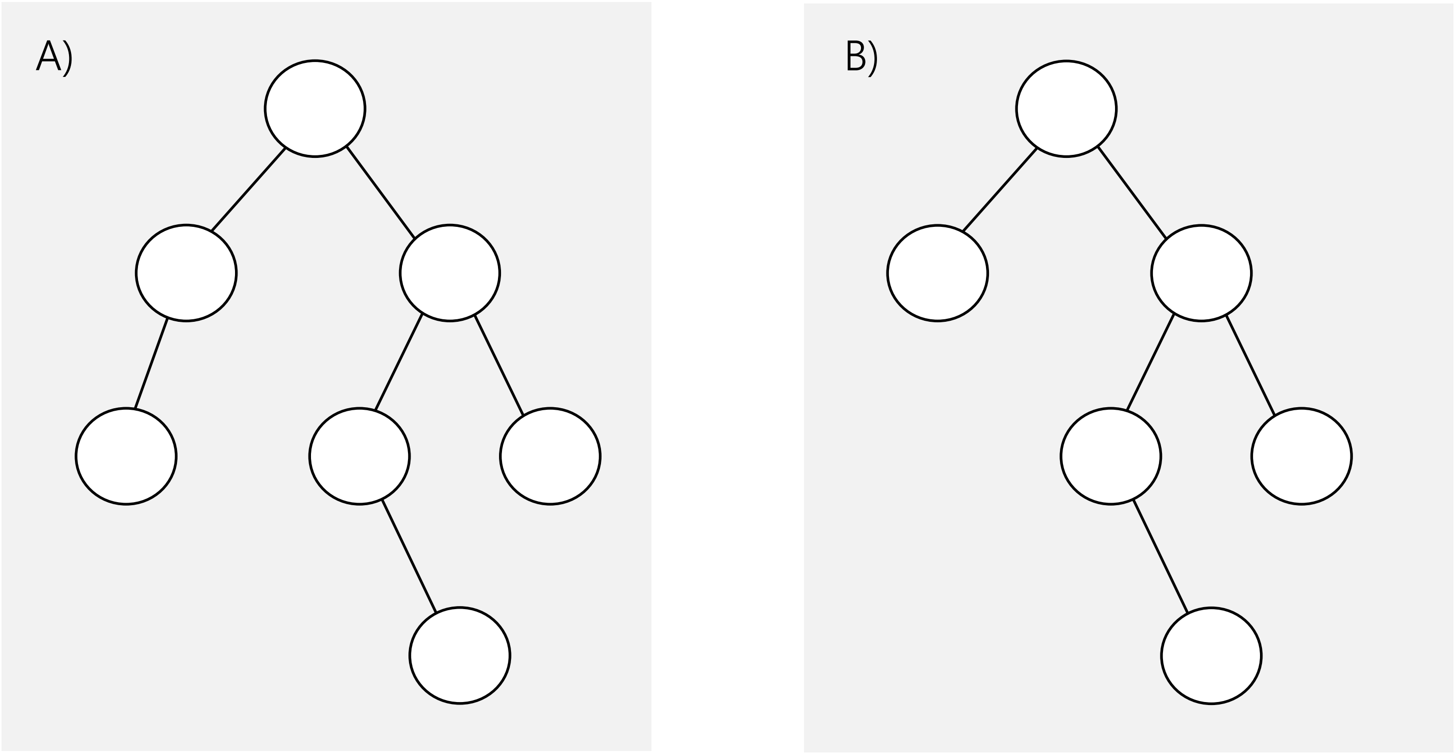
Der AVL-Baum ist ein balancierter Baum, bei dem für jeden Knoten gilt, dass sich die Höhe der beiden Teilbäume um höchstens eins unterscheidet.




- Vorteile:
  - einfacher zu realisieren
  - Degenerierung zu einer Liste ist nicht möglich
  - Suchoperationen sind schnell: O(log n)
- Nachteile:
  - zusätzlicher Aufwand bei der Programmierung, Einfügen und Löschen sind komplizierter

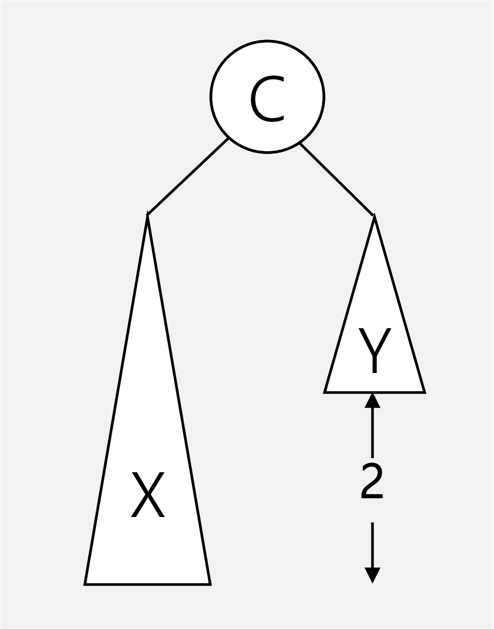
# AVL-Baum: Übung

Erfüllt einer der beiden Bäume das Kriterium der AVL-Ausgeglichenheit? Wenn ja, welcher?



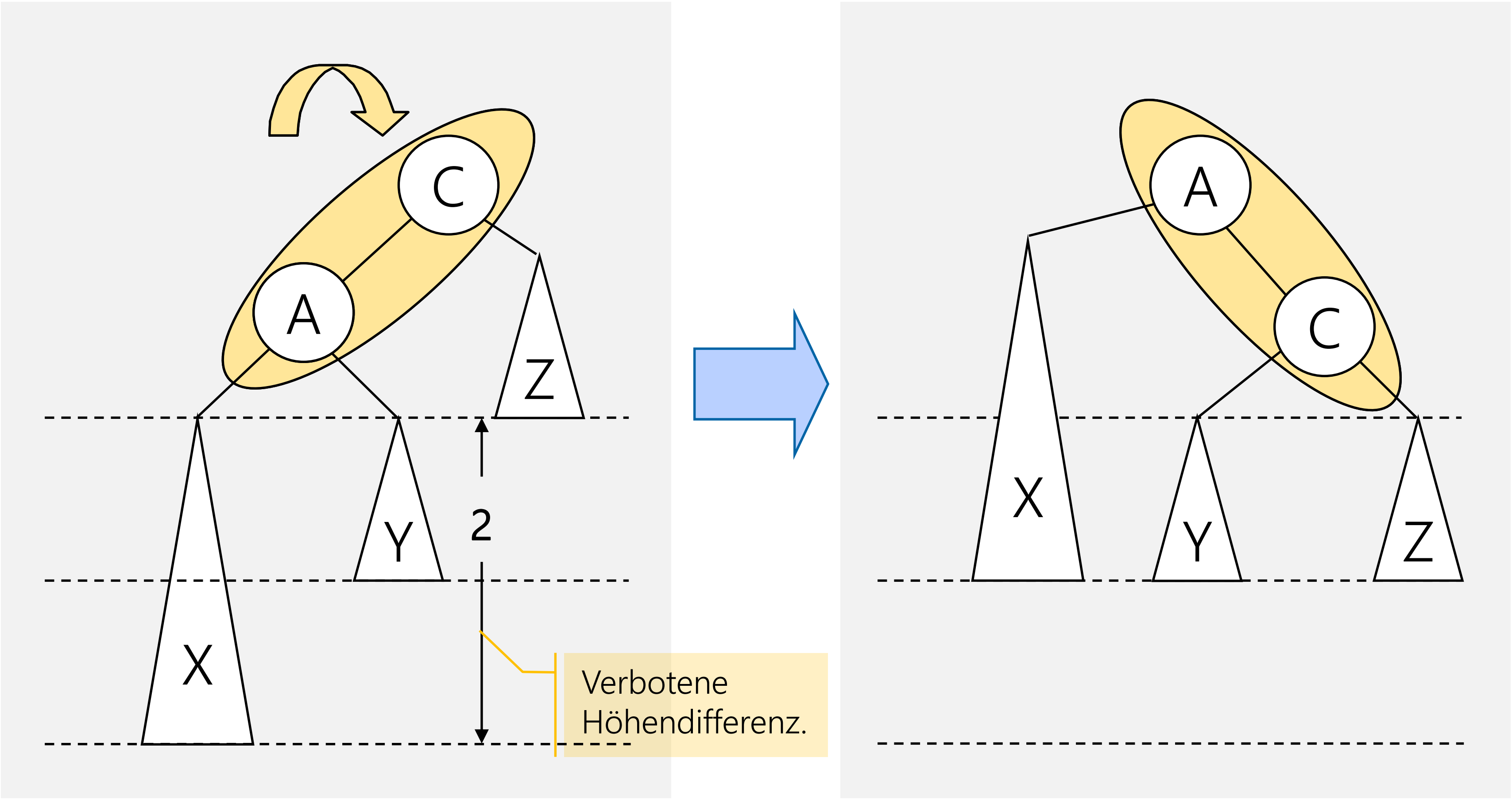
# AVL-Baum: Operationen

- Suchen und Traversieren unverändert (ist und bleibt ein Binärbaum)
- Bei allen Einfüge- und Löschoperationen wird sichergestellt, dass die AVL-Ausgleichsbedingung erhalten bleibt
  - Es muss Buch geführt werden, wie tief die darunter gelegenen Teilbäume sind (pro Knoten eine Zahl).
  - Wird die Differenz zwischen linkem und rechten Teilbaum grösser 1 muss etwas unternommen werden.
- Zum Wiederherstellen der Ausgleichsbedingung werden sogenannte Rotationen eingesetzt.

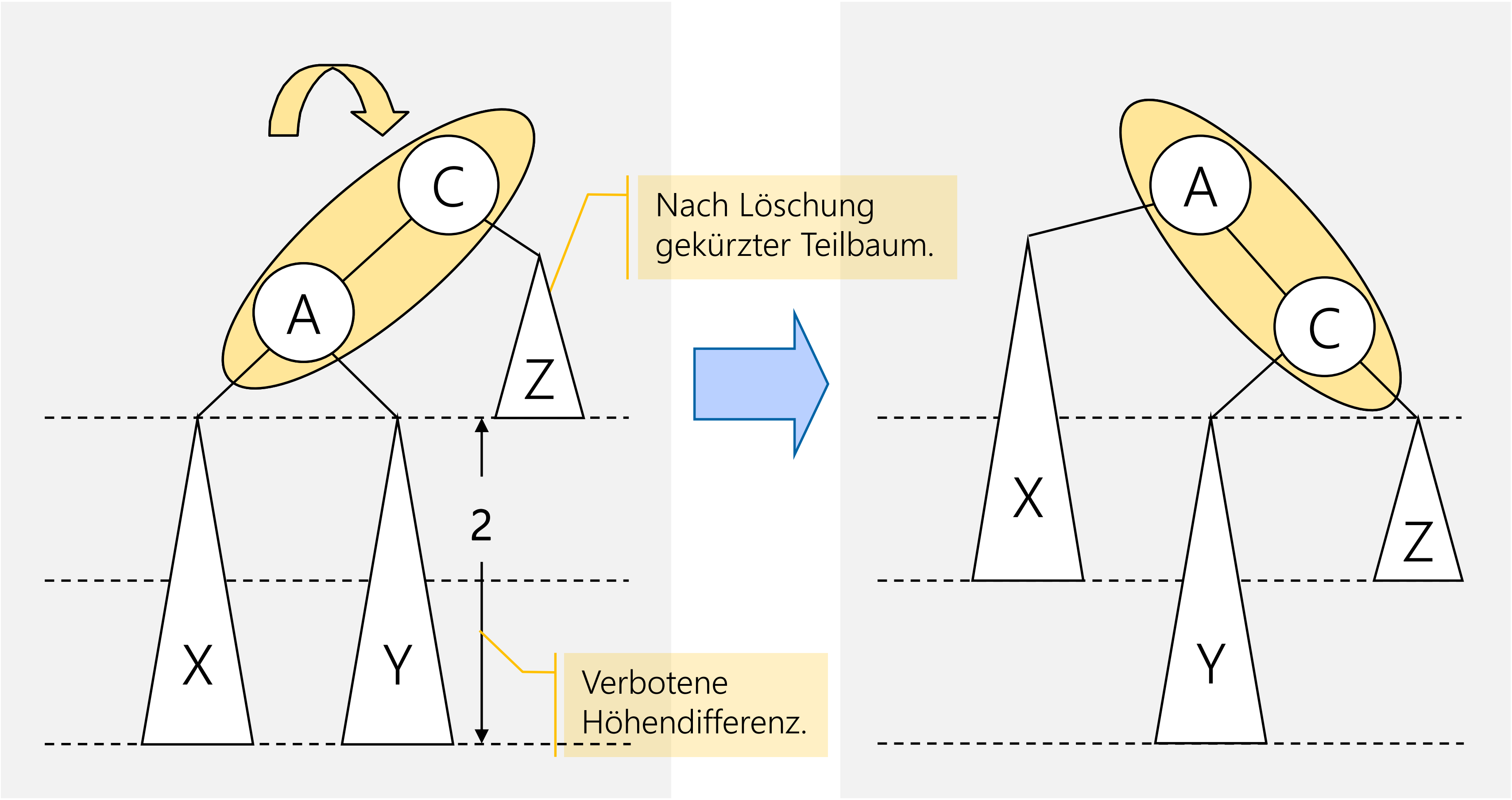


# AVL-Baum: Einzelrotation Rechts

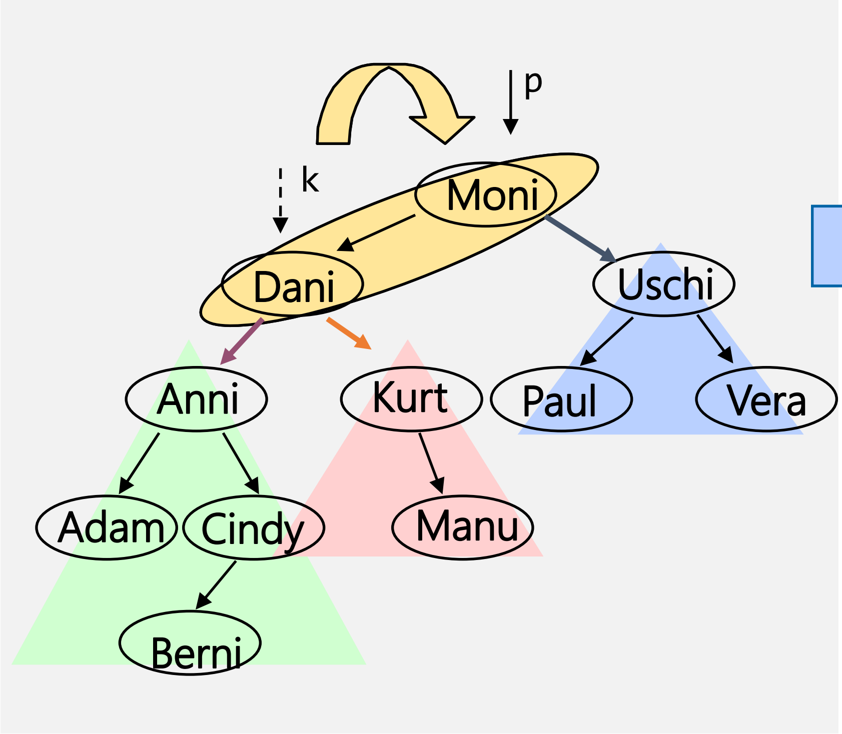


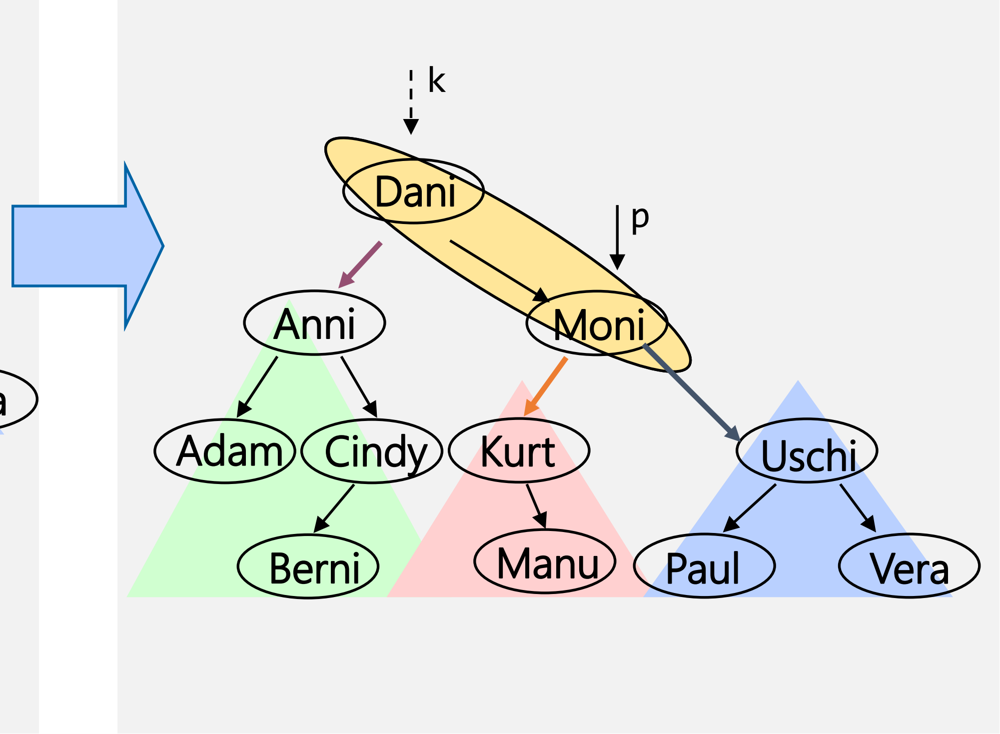


# AVL-Baum: Einzelrotation Rechts



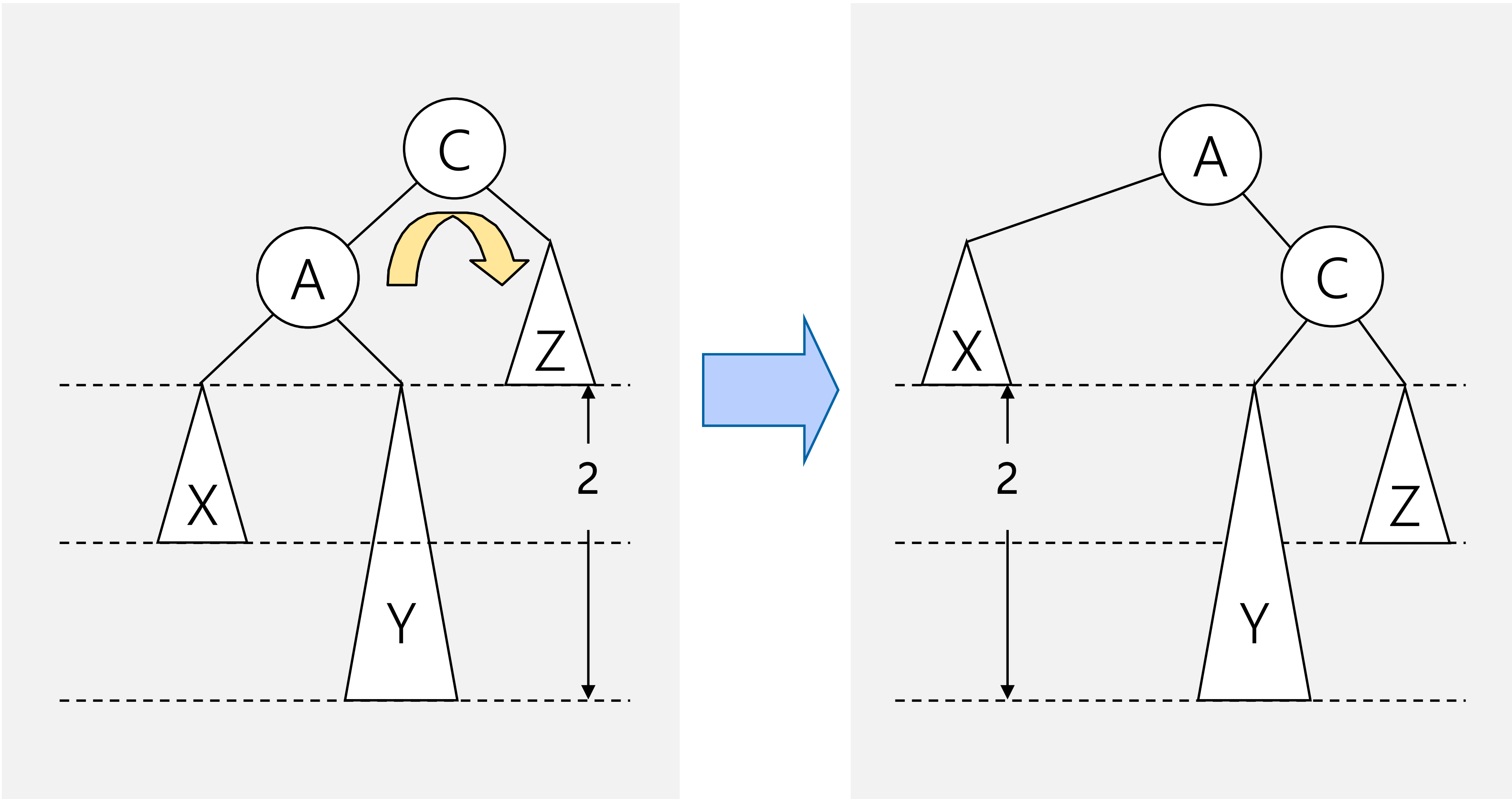
# AVL-Baum: Einzelrotation Rechts



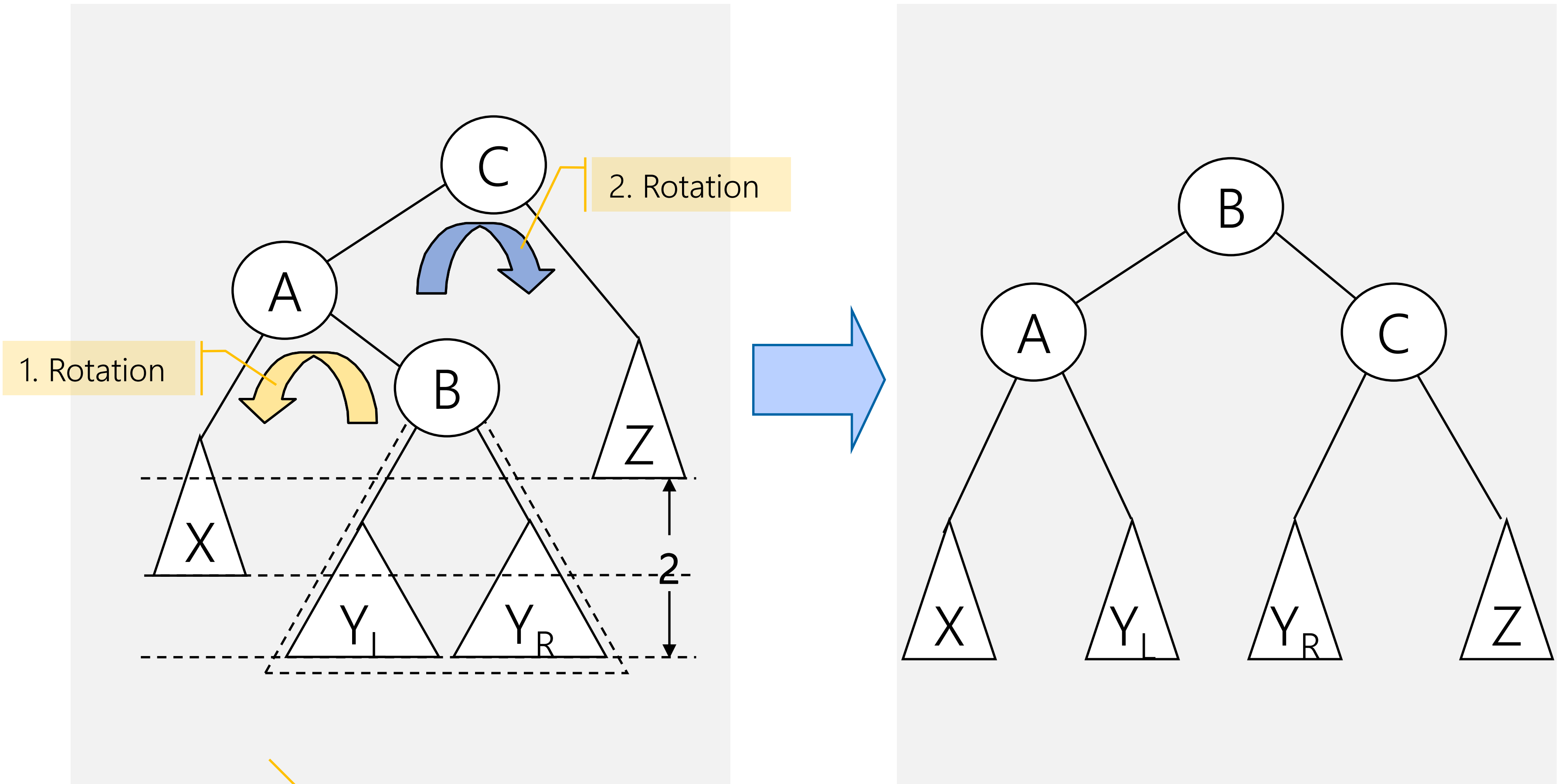

```
Node rotateR(Node p) {
    Node k = p.l;
    p.l = k.r;
    k.r = p;
    return k;
}
```

# AVL-Baum: Problem Einzelrotation




Kann nicht mit Einzelrotation balanciert werden.

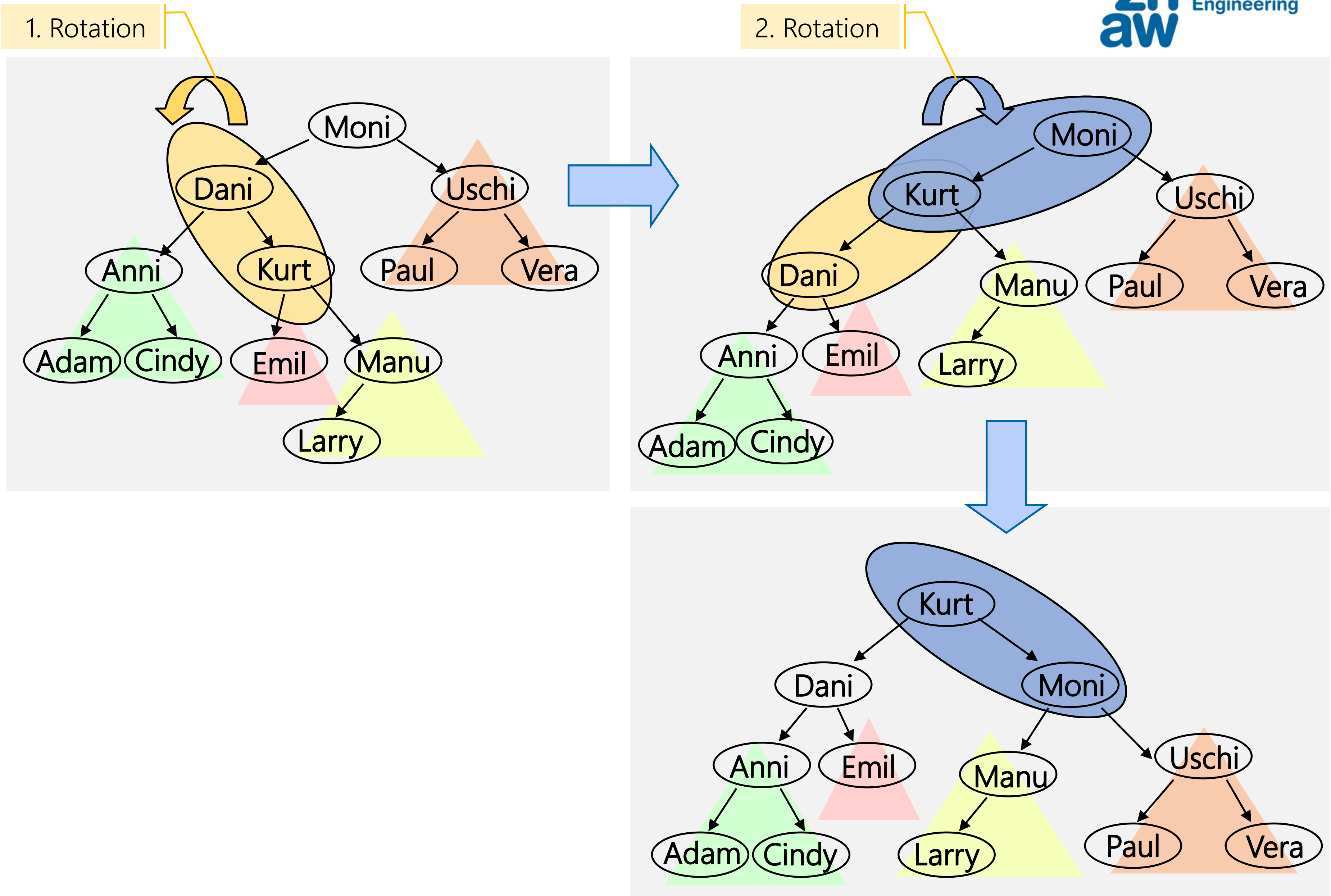
# AVL-Baum: Doppelrotation




Mit der ersten Rotation wird dieselbe Ausgangssituation wie bei der Einzelrotation hergestellt.

# AVL-Baum: Doppelrotation, Beispiel RotateLR





# AVL-Baum: Implementation

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering

- Knoten

```
class TreeNode<T extends Comparable<T>> {
    List<T> values;
    TreeNode left, right;
    int height;

    TreeNode(T element){
        this.values = new LinkedList<>();
        this.values.add(value);
        this.height = 1;      }

    TreeNode(T value, TreeNode left, TreeNode right){
        this(value); this.left = left; this.right = right;
    }

    T getValue(){return values.get(0);}
}
```

So können Doubletten gespeichert werden.

Höhe des Teilbaums.

Gibt immer «nur» das erste Element (der Doubletten) zurück.

- AVL Baum

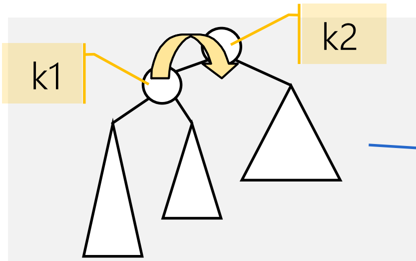
```
public class AVLSearchTree<T extends Comparable<T>> implements Tree<T> {
    private TreeNode root;

    /**
    * Return the height of node t, or 0, if null.
    */
    private static int height(TreeNode t) {
        return t == null ? 0 : t.height;
    }
}
```

# AVL-Baum: Implementation

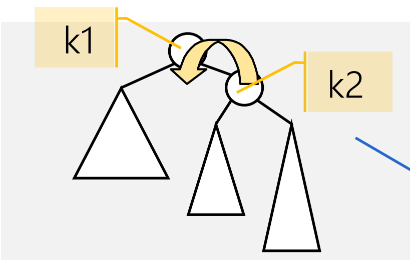
Zürcher Hochschule für Angewandte Wissenschaften

zh aw School of Engineering

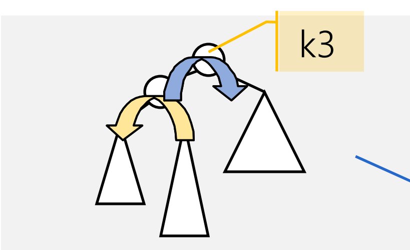
## Rotation Methoden




```
private static Node rotateR(Node k2) {
    Node k1 = k2.left;
    k2.left = k1.right;
    k1.right = k2;
    k2.height = Math.max(height(k2.left), height(k2.right)) + 1;
    k1.height = Math.max(height(k1.left), k2.height) + 1;
    return k1;
}
```

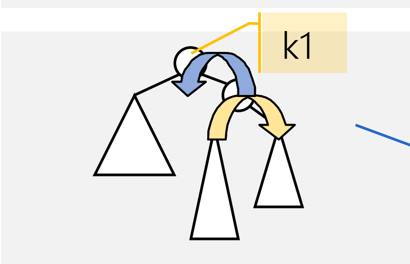



```
private static Node rotateL(Node k1) {
    Node k2 = k1.right;
    k1.right = k2.left;
    k2.left = k1;
    k1.height = Math.max(height(k1.left), height(k1.right)) + 1;
    k2.height = Math.max(height(k2.right), k1.height) + 1;
    return k2;
}
```




```
private static Node rotateLR(Node k3) {
    k3.left = rotateL(k3.left);
    return rotateR(k3);
}
```

Linksrotation linker Teilbaum.




```
private static Node rotateRL(Node k1) {
    k1.right = rotateR(k1.right);
    return rotateL(k1);
}
```

Rechtsrotation rechter Teilbaum.

# AVL-Baum: Implementation

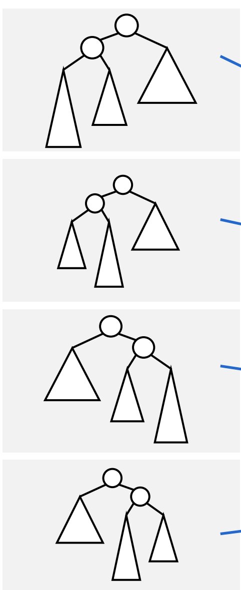
## Einfügen:

```java
private TreeNode insertAt(TreeNode p, T element) {
    if (p == null) {
        p new TreeNode<T>(element);
        return p;
    } else {
        int c = element.compareTo((T) p.getValue());
        if (c == 0) {
            p.values.add(element);
        } else if (c < 0) {
            p.left = insertAt(p.left, element);
        } else if (c > 0) {
            p.right = insertAt(p.right, element);
        }
    }
    return balance(p);
}
```

Gleiche Werte dürfen nicht doppelt eingefügt werden.

Wird bei jedem «Rück-sprung» ausgeführt.

## Balancieren:



```java
private TreeNode<T> balance(TreeNode<T> p) {
    if (p == null) return null;
    if (height(p.left) - height(p.right) == 2) {
        if (height(p.left.left) >= height(p.left.right)) {
            p = rotateR(p);
        } else {
            p = rotateLR(p);
        }
    } else if (height(p.right) - height(p.left) == 2) {
        if (height(p.right.right) >= height(p.right.left)) {
            p = rotateL(p);
        } else {
            p = rotateRL(p);
        }
    }
    p.height = Math.max(height(p.left), height(p.right)) + 1;
    return p;
}
```

Passt die Höhe wieder an.

# Zusammenfassung Binär-Bäume

- Binär-Bäume sind Bäume mit 2 Nachfolgern.
- Sortierte Binär-Bäume (auch Suchbäume genannt) erfüllen zusätzlich das Kriterium $K_{links}$ <= aktueller Knoten und $K_{rechts}$ > aktueller Knoten.
- Bei balancierten Bäumen werden Duplikate im Knoten «gezählt».
- Einfüge-/Lösch und Suchoperationen sind einfach und effizient: wachsen mit dem Log der Anzahl Knoten im Baum.
- Die meisten Operationen können einfach rekursiv programmiert werden.
- Zur Verhinderung, dass degenerierte Fälle entstehen, können Ausgleichs-operationen (Rotationen) angewandt werden.
- Die Bedingung, dass der Höhenunterschied zwischen linkem und rechtem Teilbaum maximal 1 ist, wird als AVL Ausgeglichenheitsbedingung bezeichnet.
- Diese führen aber dazu, dass Mutationen (etwas) aufwendiger werden.

Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

# B-Bäume

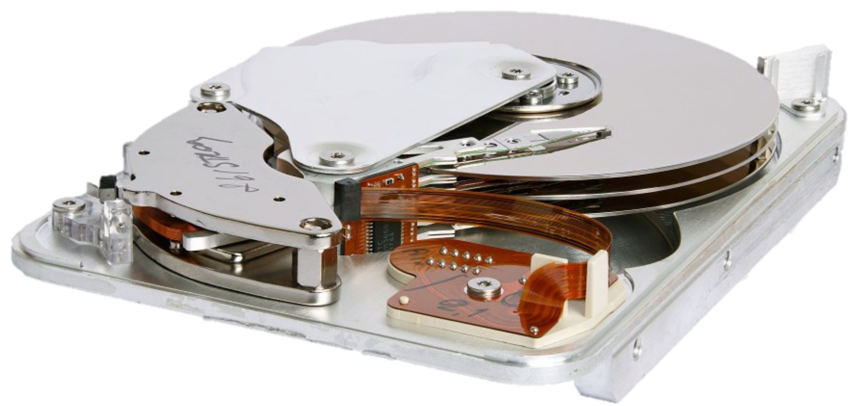
Zürcher Hochschule
für Angewandte Wissenschaften
zh aw School of Engineering

# B-Baum

- Binär-Bäume eignen sich gut für Strukturen im Hauptspeicher.
- Schlecht wenn Daten blockweise gelesen werden (Filesystem / HD und SSD):
  - Viele Random-Zugriffe auf verschiedene Blöcke (z.B. 4KB für eine Page).
  - Harddisk zusätzlich «teuer» da der Kopf physisch positioniert werden muss.
  - Lesezeiten: HD ca. ~10ms, SSD ca. ~0.1ms, Hauptspeicher Zugriff ~10 ns.

Direkter Vergleich ist aber schwierig.

- Idee: Baum so aufbauen, dass die blockweisen Zugriffe auf Disk- oder Speicher-Blöcke minimiert werden.
  - Möglichst breiter Baum, damit geringe Tiefe und wenig Zugriffe.
  - Ein Knoten im Baum wird so gross gewählt, dass er ein mehrfaches einer Disk-Seite (Page) ist, z.B. 1024 Bytes.
  - Alle Knoten sind gleich gross.
  - Der B-Baum ist ein balancierter Baum



# B-Baum

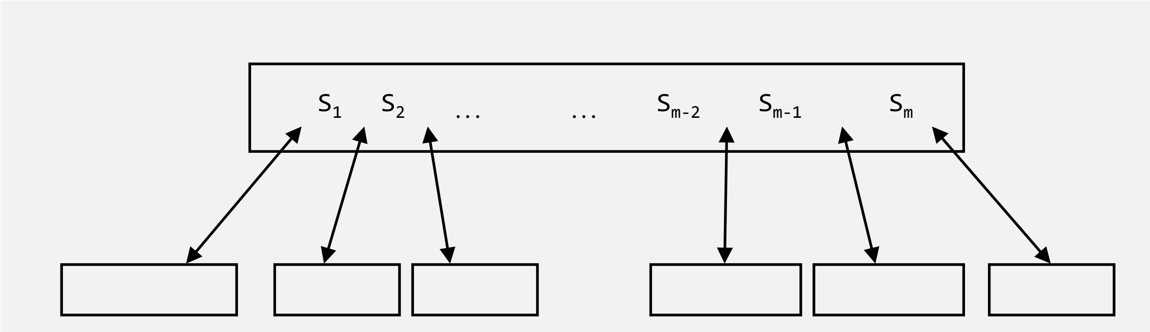
**B-Baum:**

Ein B-Baum ist ein vollständig balancierter Baum.

In der Ordnung n (n = max. Anzahl Kinder) enthält jeder Knoten, ausser der Wurzel, mindestens $\lfloor(n-1)/2\rfloor$ und höchstens n-1 Schlüssel.

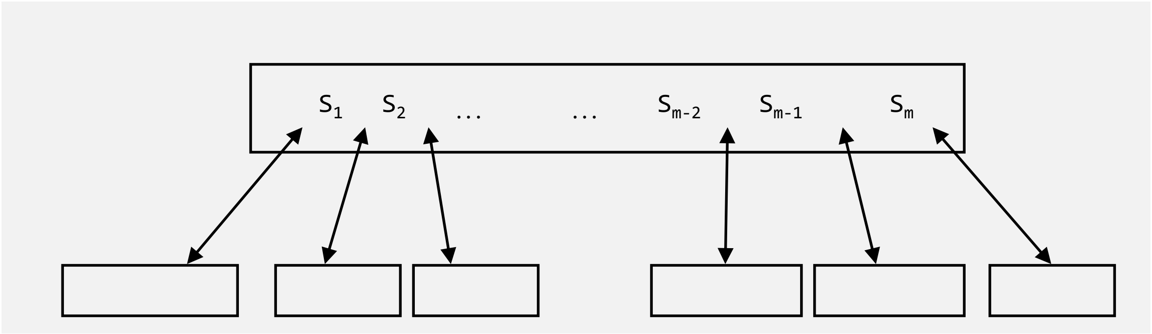
Alle Blätter haben die gleiche Tiefe.

- «B» steht für den Erfinder Rudolf Bayer (*1939), oder für Boeing (dort hat er sie entwickelt), oder für balanciert (Bayer selbst weiss auch nicht, woher der Name kam).
- Die Wurzel hat 1 bis n-1 Schlüssel.
- Tiefe des Baumes ≈ $\log_{\text{AnzahlVerweise}}$(Anzahl Elemente)



# B-Baum: Schlüssel und Verweise

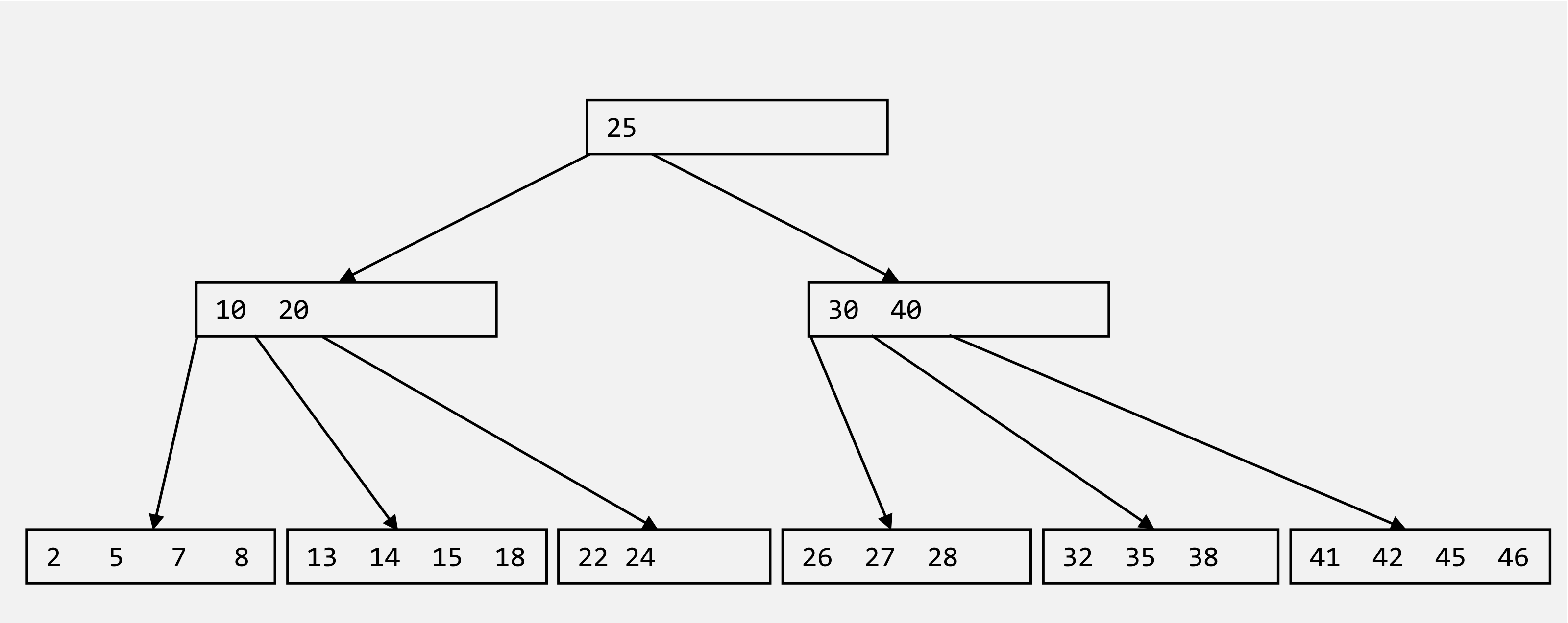
- Für die Schlüssel und Verweise gilt:
  - innerhalb eines Knotens sind alle Schlüssel sortiert
  - alle Schlüssel im i–1-ten Nachfolgerknoten sind kleiner
  - alle Schlüssel im i-ten Nachfolgerknoten sind grösser oder gleich dem Schlüssel $s_i$
- Die inneren Knoten enthalten Schlüssel und Verweise (auf Nachfolgeknoten und Informationen).
- Sind die Informationen nur in den Blättern gespeichert, spricht man vom $B^+$-Baum.



# B-Baum: Anwendungen

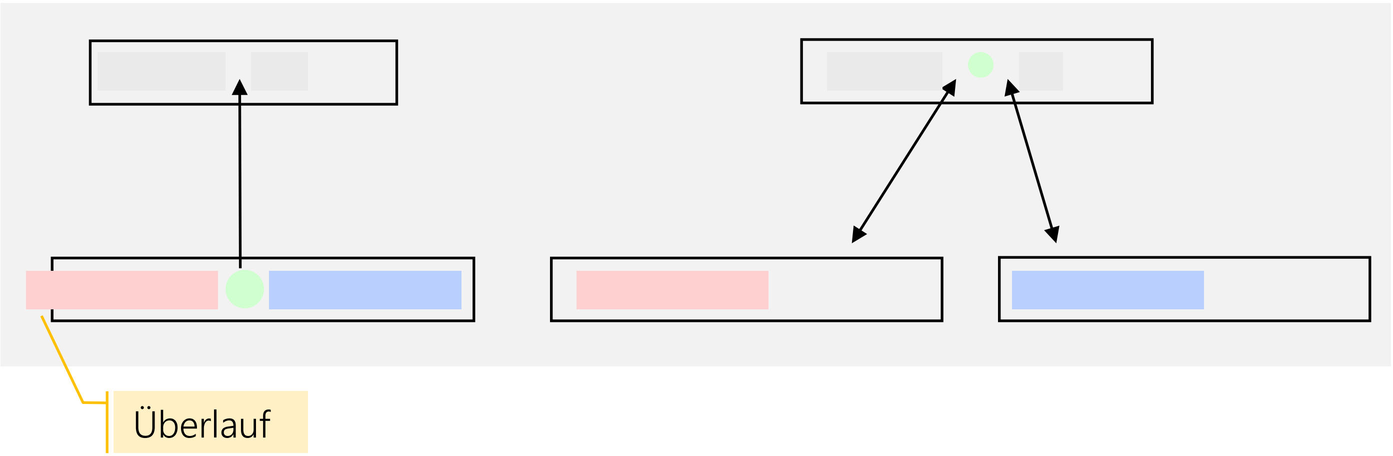
- Indexe in Datenbankensystemen: mit wenigen Diskzugriffen kann ein bestimmter Datensatz gefunden werden.
  - $O(\log_n(N))$; n = Anz. Schlüssel pro Knoten, N = Anz. Datensätze.
- Dateisysteme: Organisation der Daten auf Disk, z.B. Directorys im Windows NTFS-Filesystem ($B^+$-Baum).

# B-Baum: Beispiel mit Ordnung 5

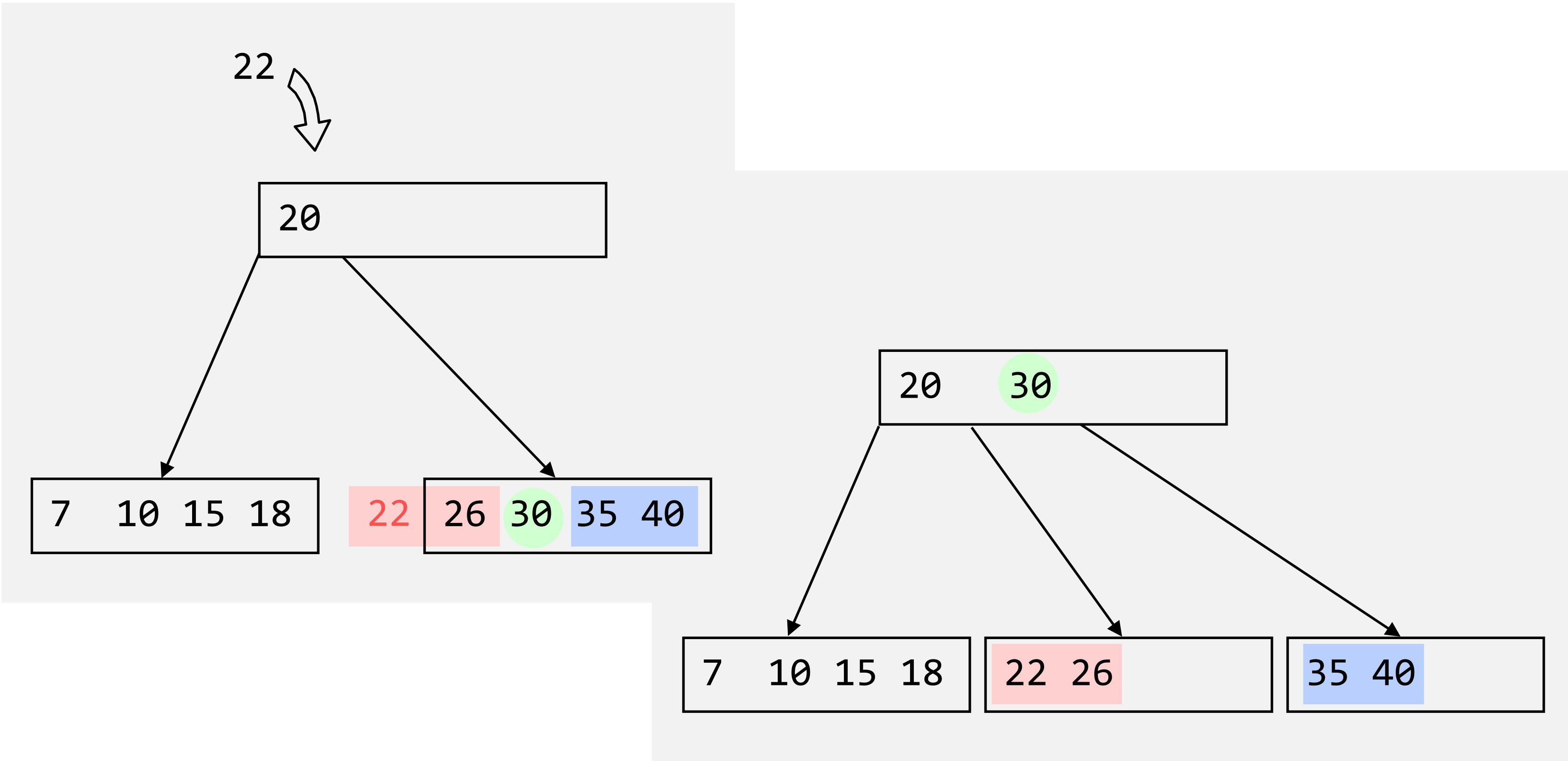


# B-Baum: Einfügen

- Eingefügt wird immer in den Blättern (solange Platz).
- Ist dieser bereits voll gibt es einen Überlauf:
  - Aufteilen in zwei Knoten und «heraufziehen» des mittleren Elements in den Vaterknoten.
  - Falls der Vaterknoten überläuft: Vorgang wiederholen.



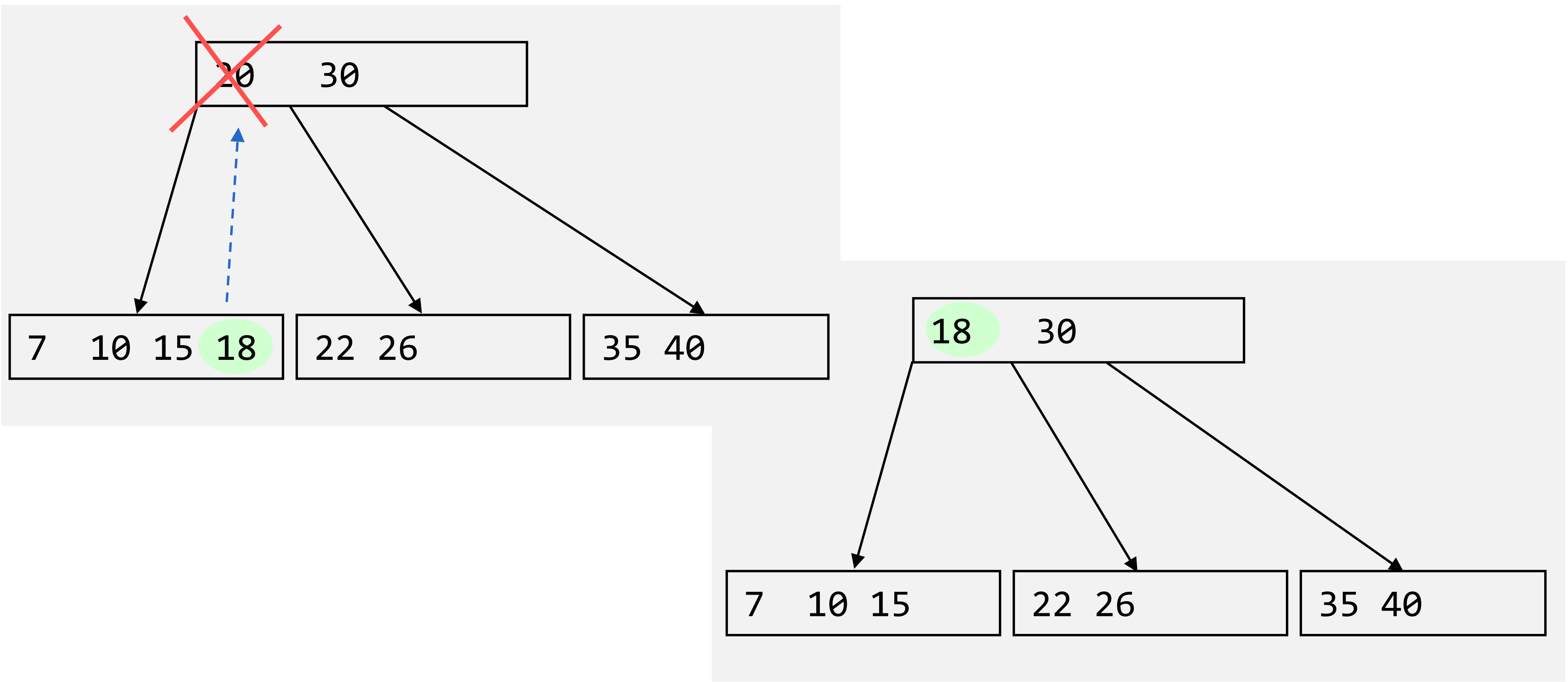
# B-Baum: Einfügeoperation mit Aufteilen




Natürlich kann dabei der Vater-Knoten ebenfalls überlaufen. In Extremfällen kann dies bis zur Wurzel propagieren. Dann ändert sich die Höhe des Baumes → B-Bäume wachsen von den Blättern zur Wurzel.

# B-Baum: Löschen

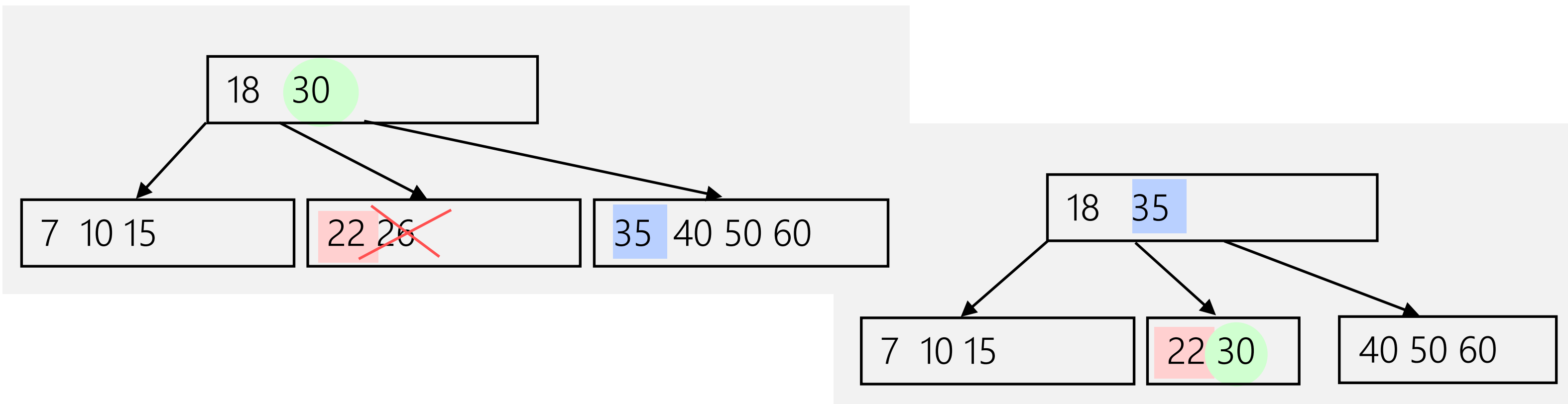
- Zu löschendes Element ist in einem Blatt: Eintrag löschen (Unterlauf?)
- Zu löschendes Element ist einem inneren Knoten:
  Gleich verfahren wie bei Binärbaum: Ersatzwert in Blättern suchen (Unterlauf?).



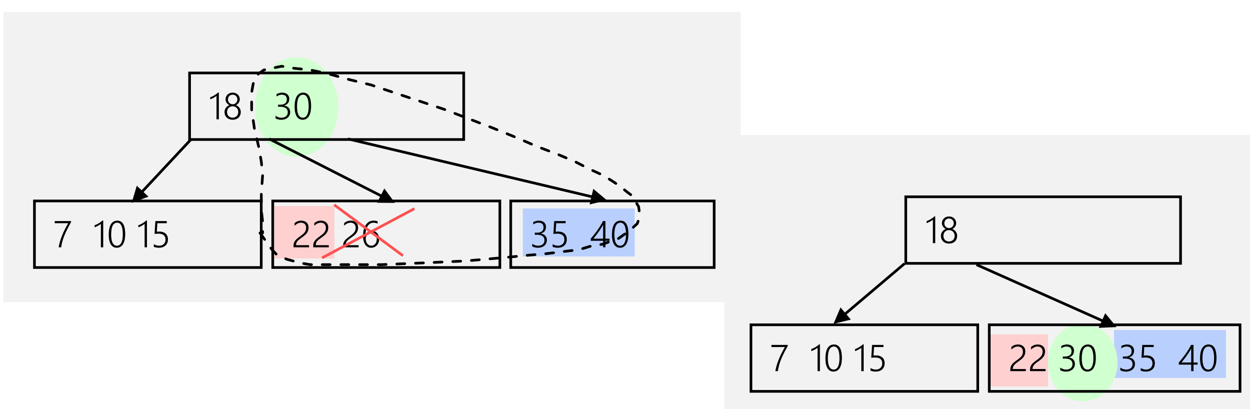
# B-Baum: Löschen mit Unterlauf

Unterlauf: Ein Blatt enthält weniger als n/2 Schlüssel.

- «Ausleihen» bei einem Nachbarknoten.




- Oder (besser) verschmelzen: Zwei benachbarte Knoten werden zu einem zusammengefasst.



# B-Baum: Suchen

1. Den Wurzelblock lesen.
2. Gegebenen Schlüssel S auf dem gelesenen Block suchen.
3. Wenn gefunden → referenzierte Daten lesen → fertig
4. Ansonsten i finden, sodass: $S_i <= S < S_{i+1}$
5. Block des i-ten Nachfolgeknoten einlesen, Schritte 2 bis 5 wiederholen.

- Tiefe des Baumes ≈ $\log_{\text{AnzahlVerweise}}(\text{Anzahl Elemente})$
- Anzahl Zugriffe: proportional zur Tiefe des Baumes

Annahme: mehrere hundert Schlüssel und Verweise pro Block → Tiefe des Baumes selten grösser als 5 bis 6.

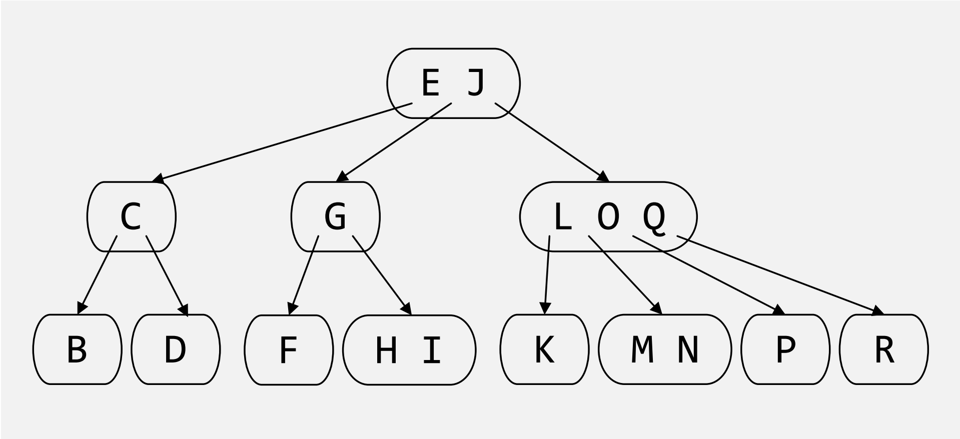
# 2-3-4-Baum und Rot-Schwarz-Bäume

# B-Baum: 2-3-4-Baum

**2-3-4-Baum:**

Ein 2-3-4-Baum ist ein Spezialfall des B-Baums, in dem jeder Knoten zwei, drei oder maximal vier Kinder besitzt (= Ordnung 4).

- 2-3-4-Bäume sind eine gute Alternative zu AVL-Bäumen.
- Aufgrund der geringen Anzahl von Schlüsseln in den Knoten, eigenen sie sich eher für Datenstrukturen im Memory, als für die block-orientierten Speicher (z.B. Disk).

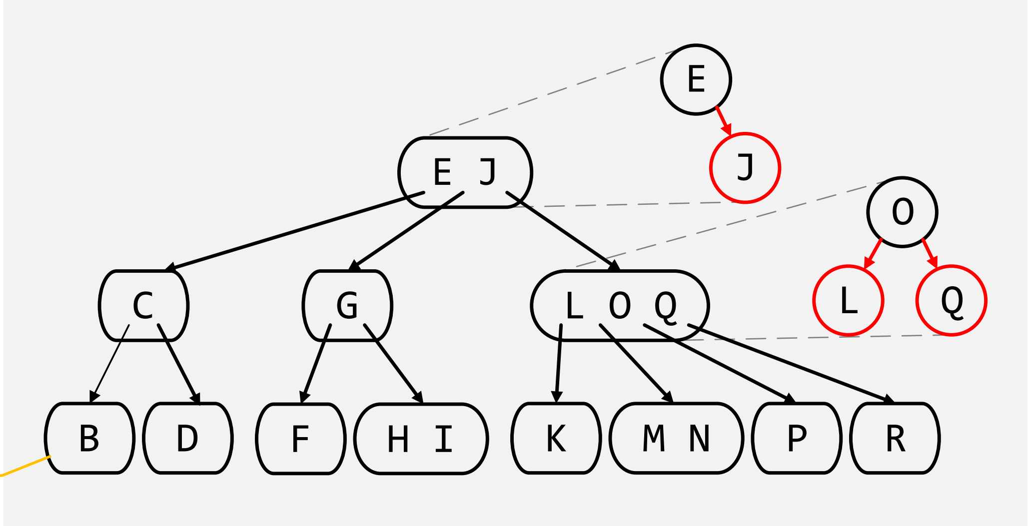


# 2-3-4-Baum: Rot-Schwarz-Baum

**Rot-Schwarz-Baum:**

Ein Rot-Schwarz-Baum ist ein Spezialfall des 2-3-4-Baums, bei welchem die Knoten mit 2 oder mehr Schlüsseln durch Binärbäume implementiert werden.

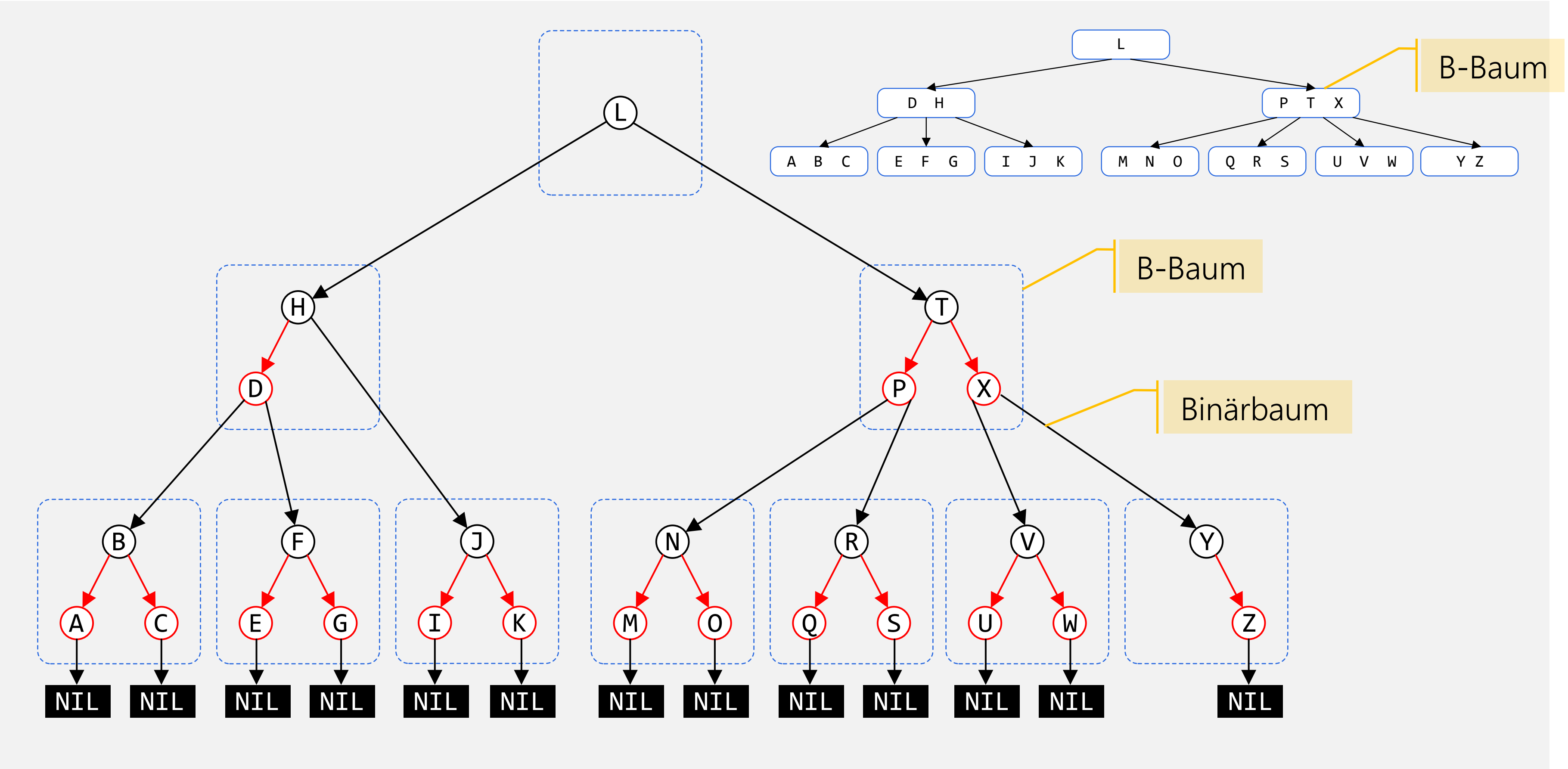
- Durch «Färben» der Kanten wird 2-3-4-Baum als Binärbaum implementiert.
- Schwarze Knoten werden strikt blanciert (gemäss B-Baum), rote dienen als «Schlupf», der Anteil des Schlupfs ist beschrankt.
- 1972 von Rudolf Bayer beschrieben.
- Maximale Höhe: $2 \cdot \text{Log}_2(n+2) - 2$
- Aufwand Suchen: $O(\log(n))$




Wie kann ich diesen 2-3-4-Baum (B-Baum der Ordnung 4) als Binärbaum darstellen?

# Rot-Schwarz-Baum: Eigenschaften

- Vorteil: Einfachheit von Binärbäumen und Ausgeglichenheit von B-Bäumen
- Weniger gut balanciert als AVL Baum, aber Einfüge- und Lösch-Operationen sind schneller.



# Rot-Schwarz-Baum: Bedingungen

Ausgleichverfahren so dass Rot-Schwarz-Bedingungen erhalten bleiben:

1. Jeder Knoten im Baum ist entweder rot oder schwarz.
2. Die Wurzel des Baums ist schwarz.
3. Null-Zeiger (früher NIL statt Null) für fehlendes Kind, betrachten wir als externe Knoten mit der Farbe schwarz.
4. Kein roter Knoten hat ein rotes Kind.
5. Jeder Pfad, von einem gegebenen Knoten zu seinen Blattknoten, enthält die gleiche Anzahl schwarzer Knoten (Schwarzhöhe/Schwarztiefe). Für die Tiefe zählt man nur die Schwarzen Knoten.

# Zusammenfassung

- Zugriffszeit im binären Suchbaum
- Balancieren von Bäumen:
  - Vollständig Balanciert: Abgesehen von der untersten Ebenen, alle Ebenen vollständig mit Knoten besetzt
  - Balanciert: Maximale Höhe von c·log(n) garantiert
  - AVL-Balanciert: Tiefe der Teilbäume je Knoten unterscheidet sich nur um +/-1
- B-Baum
  - Optimiert für Massenspeicherzugriff.
  - Es gibt mindestens n/2 Unterbäume (n wird bewusst an die Grösse der Speicherseite angepasst). Dadurch wird der Baum weniger hoch ⇒ weniger Plattenzugriffe notwendig.
- 2-3-4 Baum
  - B-Baum mit max. 4 Nachfolgern, B-Baum der Ordnung 4
- Rot-Schwarz-Baum:
  - Durch «Färben» der Kanten kann 2-3-4-Baum als Binärbaum implementiert werden: java.util.TreeMap
  - Rot-Schwarz-Bäume sind 2-3-4-Bäume als Binärbäume

Kontrollfragen Lektion 6 nicht vergessen – heute mit dem Rumpelstilzchen