6 Baumstrukturen

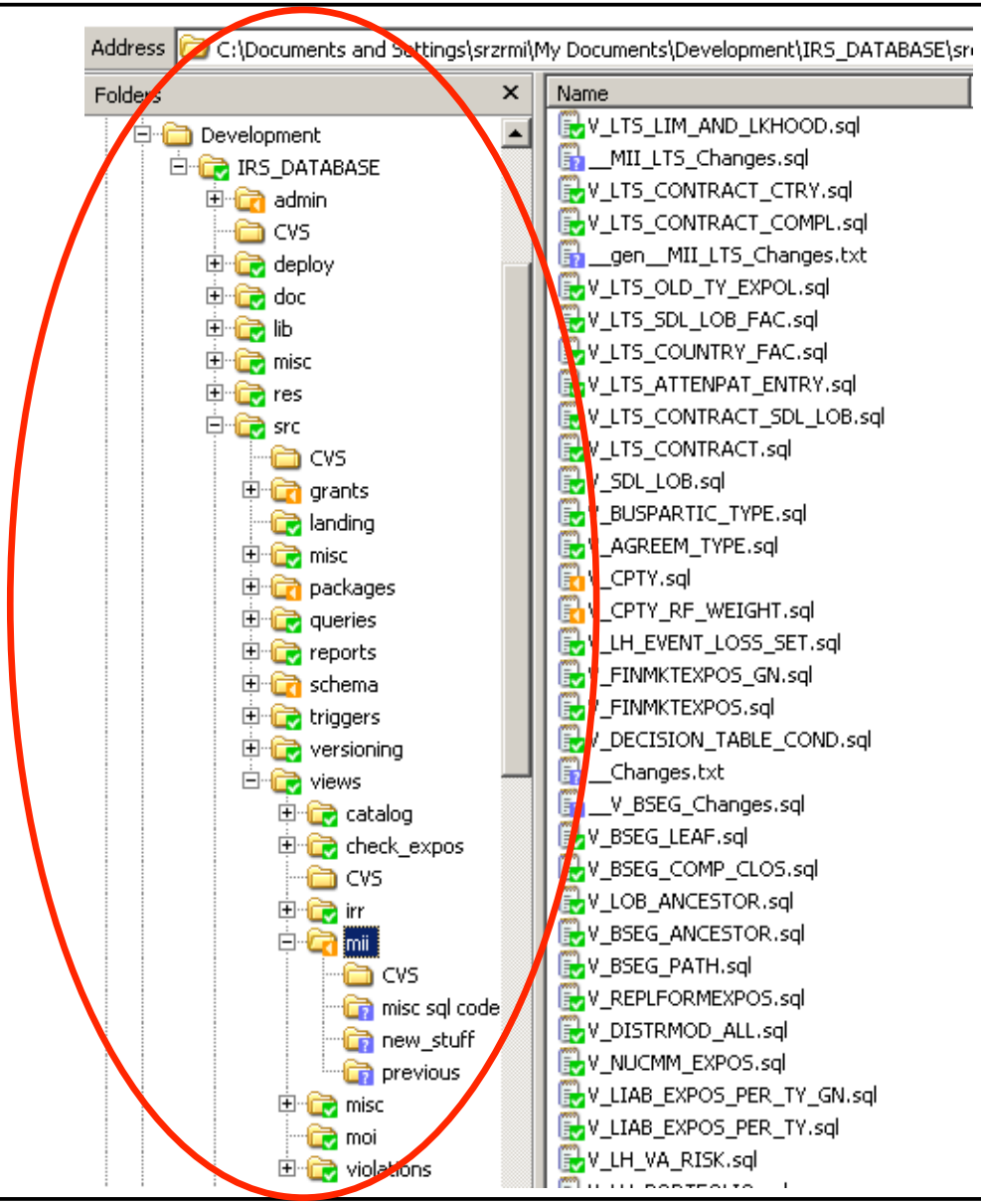
Formale Grundlagen der Informatik I

Herbstsemester 2011

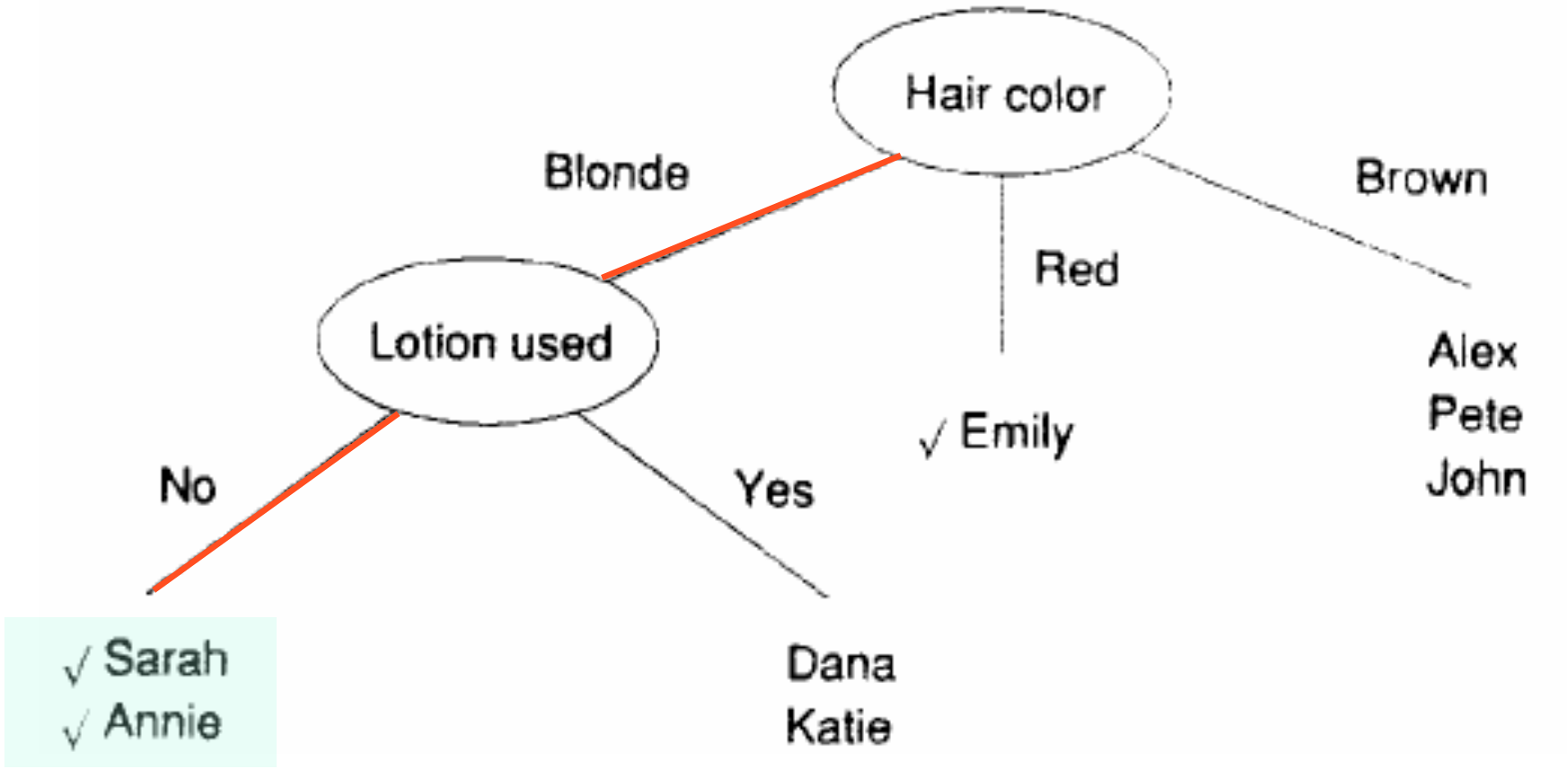
Robert Marti

Vorlesung teilweise basierend auf Unterlagen
von Prof. emer. Helmut Schauer

Beispiel: Hierarchisches File System



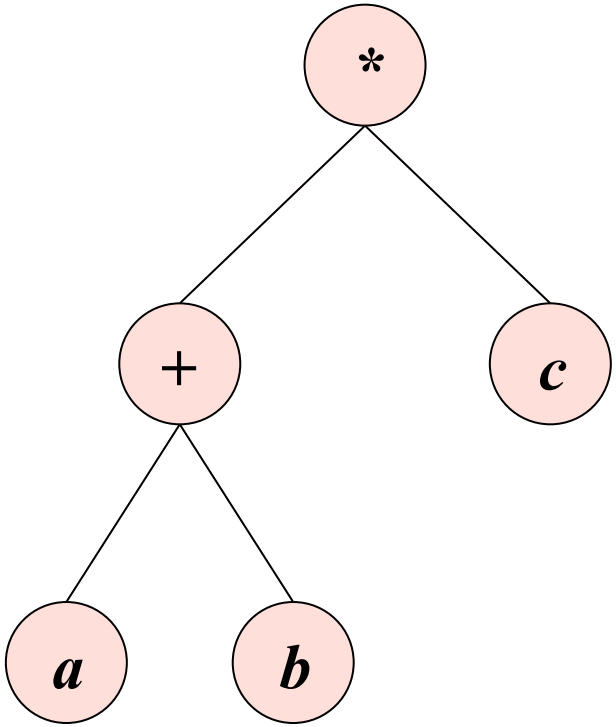
Beispiel: Entscheidungsbaum (Decision Tree)



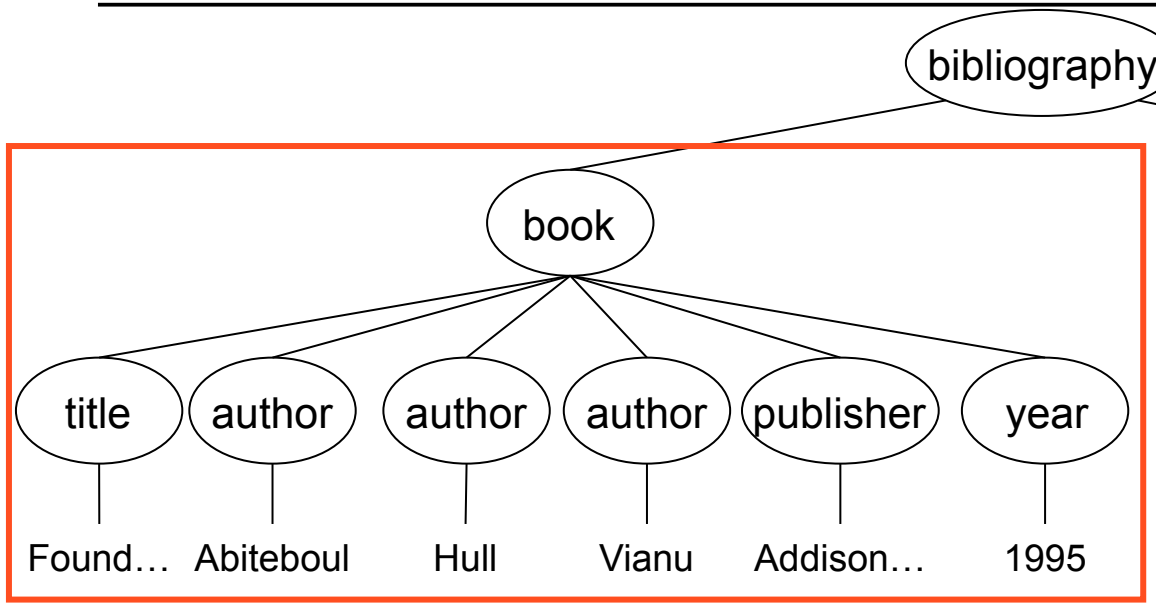
if *Hair_color = 'Blonde'* **and** *Lotion_used = 'No'* **then** *Sunburn = 'Yes'*

Beispiel: Operatorbaum

- Beispiel: $(a + b) \cdot c$ bzw. $(a + b) * c$
- Darstellung als Baum
  - Variablen sind Knoten
  - Operatoren sind ebenfalls Knoten, deren Unterbäume sind Operanden
- Traversieren des Baums
  - preorder: $* + a\ b\ c$
    in der Sprache Lisp: `(* (+ a b) c)`
  - inorder: $(a + b) * c$
  - postorder: $a\ b + c\ *$
    z.B. HP Calculators (UPN bzw. RPN), Sprachen Forth, Postscript



Beispiel: XML Dokument als Baum

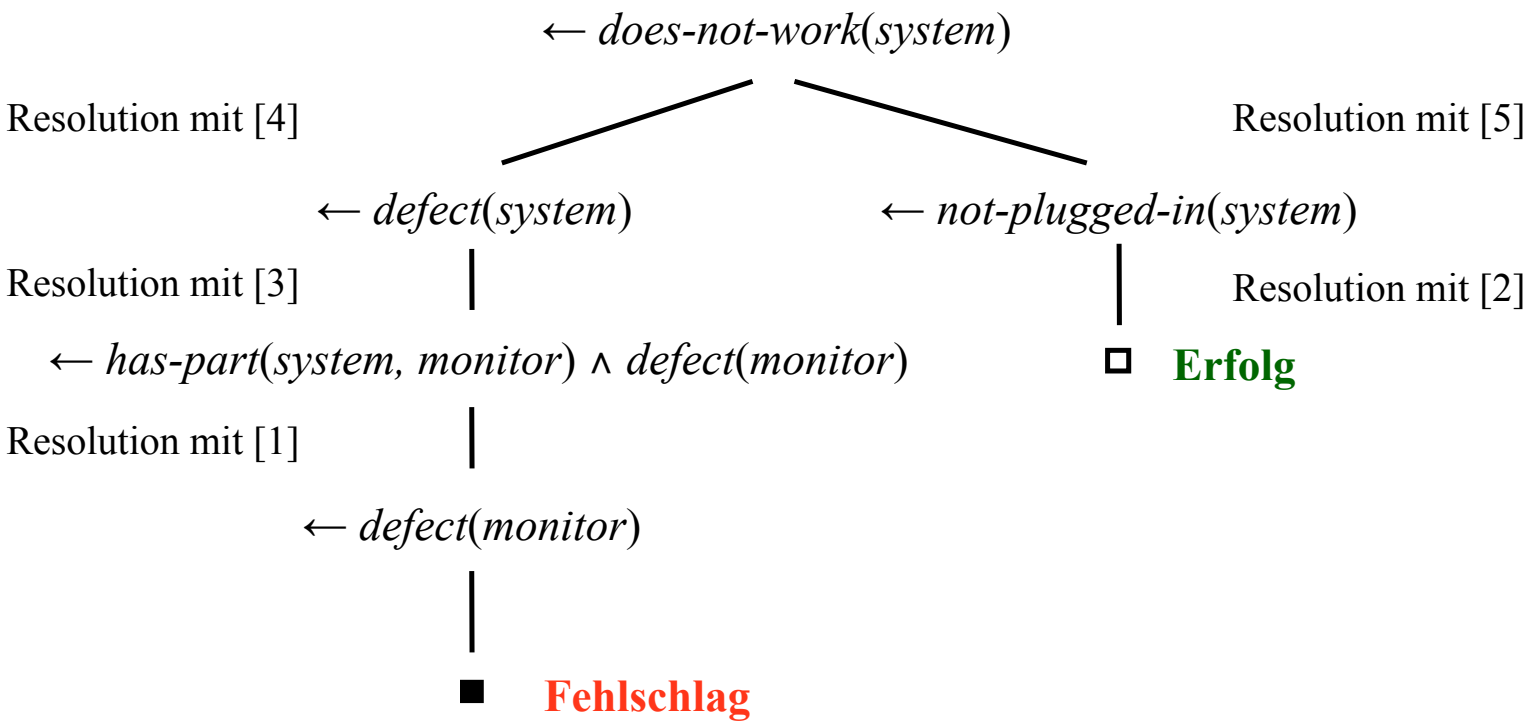



```
<bibliography>
  <book>
    <title>
       Foundations of Databases
     </title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>
       Addison Wesley
     </publisher>
    <year>1995</year>
  </book>
  <book>
    ...
   </book>
</bibliography>
```

Beispiel: Beweis-Baum (Proof Tree)

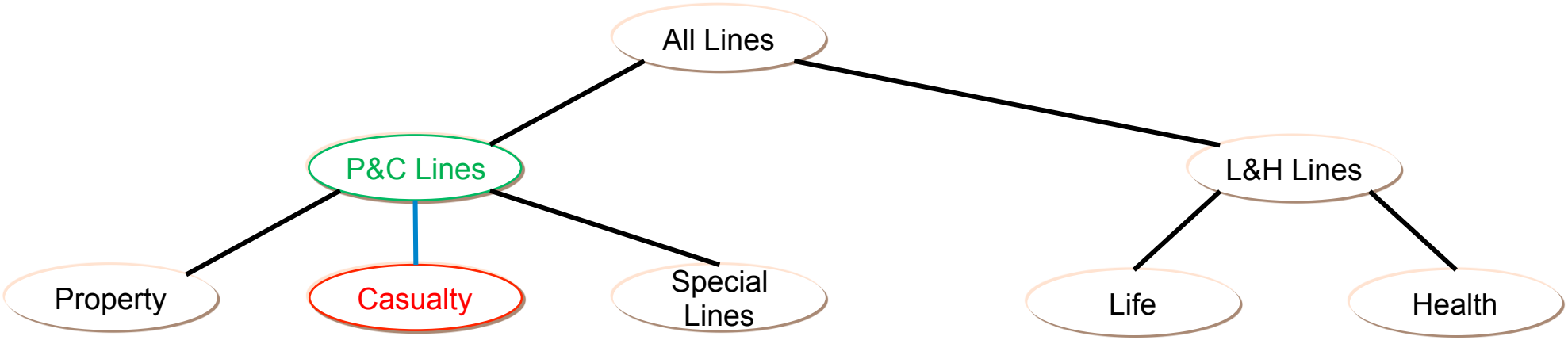
Δ := { *has-part*(*system*, *monitor*) , [1]
not-plugged-in(*system*) , [2]
defect(*system*) ← *has-part*(*system*, *monitor*) ∧ *defect*(*monitor*) , [3]
does-not-work(*system*) ← *defect*(*system*) , [4]
does-not-work(*system*) ← *not-plugged-in*(*system*) } [5]

goal := ← *does-not-work*(*system*)



Beispiel: Taxonomie von Geschäftsbegriffen

sog. Lines of Business im Versicherungsbereich



'Casualty' is a narrower term than 'P&C Lines'

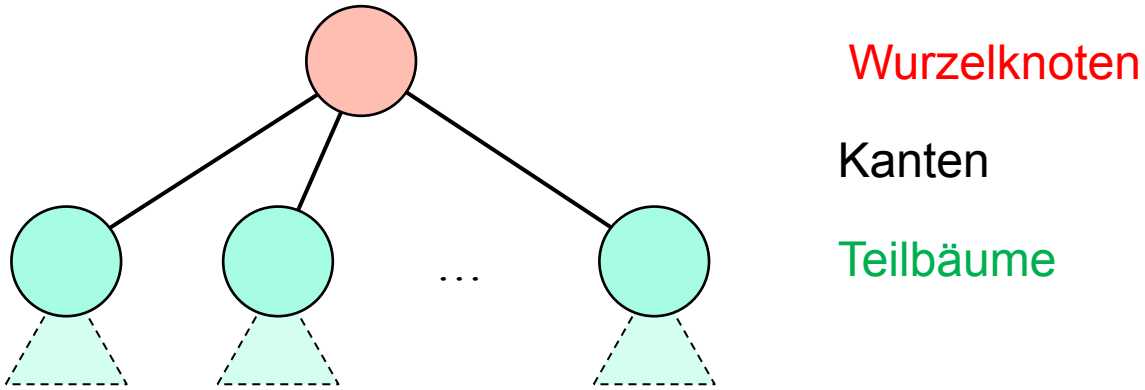
Definitionen (1)

Ein **Baum** (*tree*) ist

- entweder leer
- oder er besteht aus einer **Wurzel** (*root*) mit einer beliebigen Anzahl $n \geq 0$ von **Teilbäumen** (*subtrees*), von denen jeder wiederum ein Baum ist.

Die Wurzel des Baumes sowie die Wurzeln seiner Teilbäume heissen **Knoten** (*nodes* oder auch *vertices* [= Plural von *vertex*]) des Baumes.

Die Teilbäume sind mit der Wurzel durch **Kanten** (*edges*) verbunden.

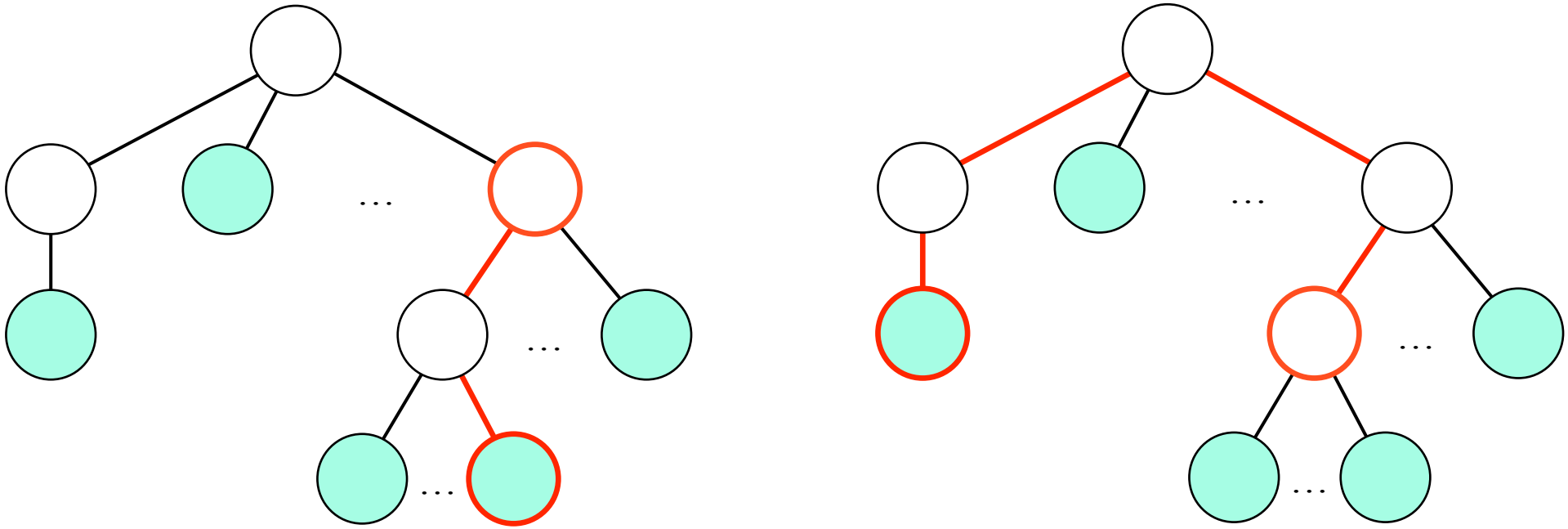


Definitionen (2)

Knoten ohne Teilbäume (bzw. Knoten, deren Teilbäume alle leer sind) heissen **Blätter** (*leaves*).

Eine Folge von Knoten die durch Kanten miteinander verbunden sind, heisst **Pfad** (*path*).

Zwischen zwei beliebigen Knoten eines Baumes gibt es genau einen Pfad.



Geordneter Baum

Knoten in einem Baum haben im allgemeinen

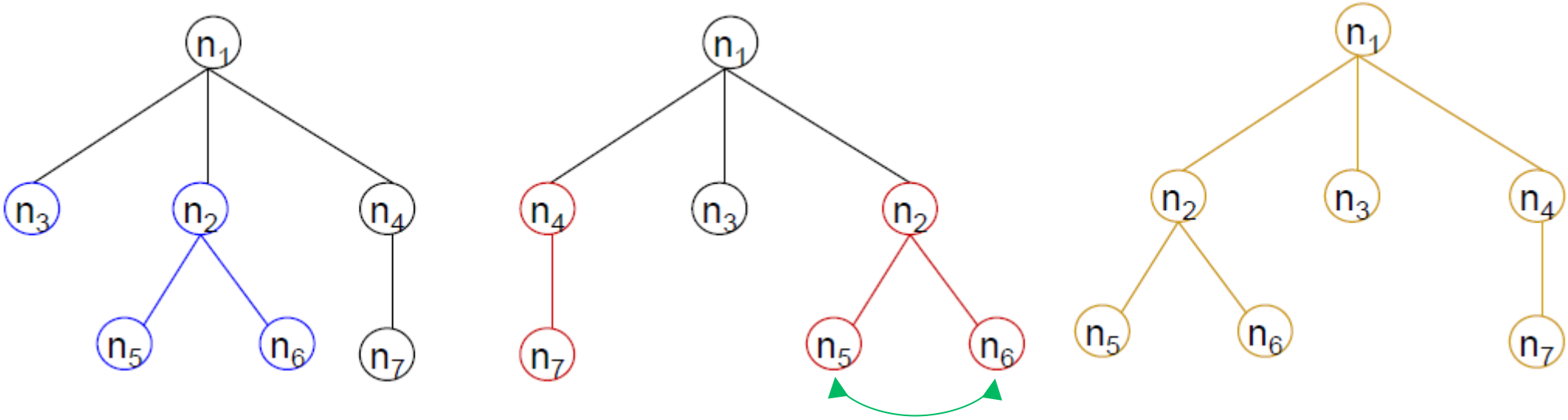
- eine Etikette (*label*) bzw. einen Schlüssel (*key*)
- und/oder einen Wert (*value*).

In einem **geordneten Baum** (*ordered tree*) ist die Reihenfolge der Teilbäume jedes Knotens relevant.

Dies kann eine explizite Ordnung sein oder aber eine Sortierreihenfolge, die vom Schlüssel oder vom Wert der Wurzelknoten der Teilbäume induziert wird.

Bäume, die sich nur durch die Reihenfolge ihrer Teilbäume unterscheiden, werden als zueinander **isomorph** bezeichnet.

Beispiel: Isomorphe Bäume

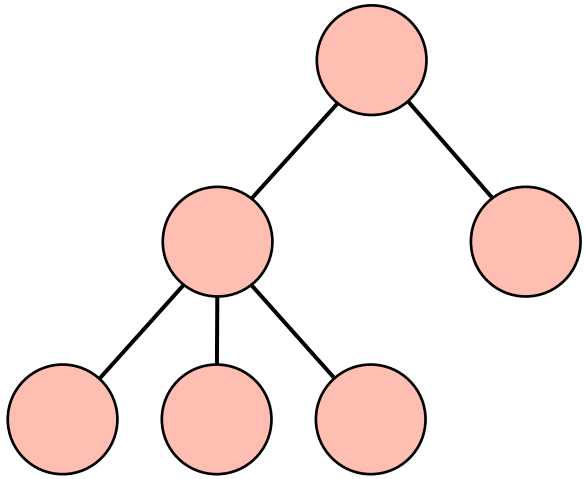


Ordnung eines Baums

Die **maximale Anzahl** von Teilbäumen eines Knotens entspricht der **Ordnung** (*degree*) eines Baumes.

Die (exakte oder mittlere) Anzahl von Teilbäumen eines Knotens wird auch als *fanout* bezeichnet.

Beispiel: Baum der Ordnung 3



Datenstruktur: $tree =$

$value$		
$subtree_1$	$subtree_2$	$subtree_3$

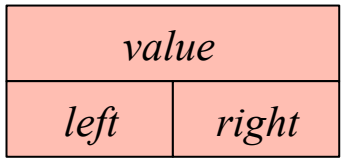
Binärer Baum

Ein **binärer Baum** (*binary tree*) ist ein Baum in dem jeder Knoten maximal zwei Teilbäume hat.

Binäre Bäume haben die Ordnung 2.

Ist der binäre Baum geordnet, so sprechen wir von linken und rechten Teilbäumen.

Datenstruktur: $tree =$



Perfekter binärer Baum

Ein binärer Baum heisst **perfekt** oder **voll** wenn für jeden Knoten gilt:

Der linke und der rechte Teilbaum dieses Knotens besteht aus gleich vielen Knoten.

In einem perfekten binären Baum ist jeder Knoten

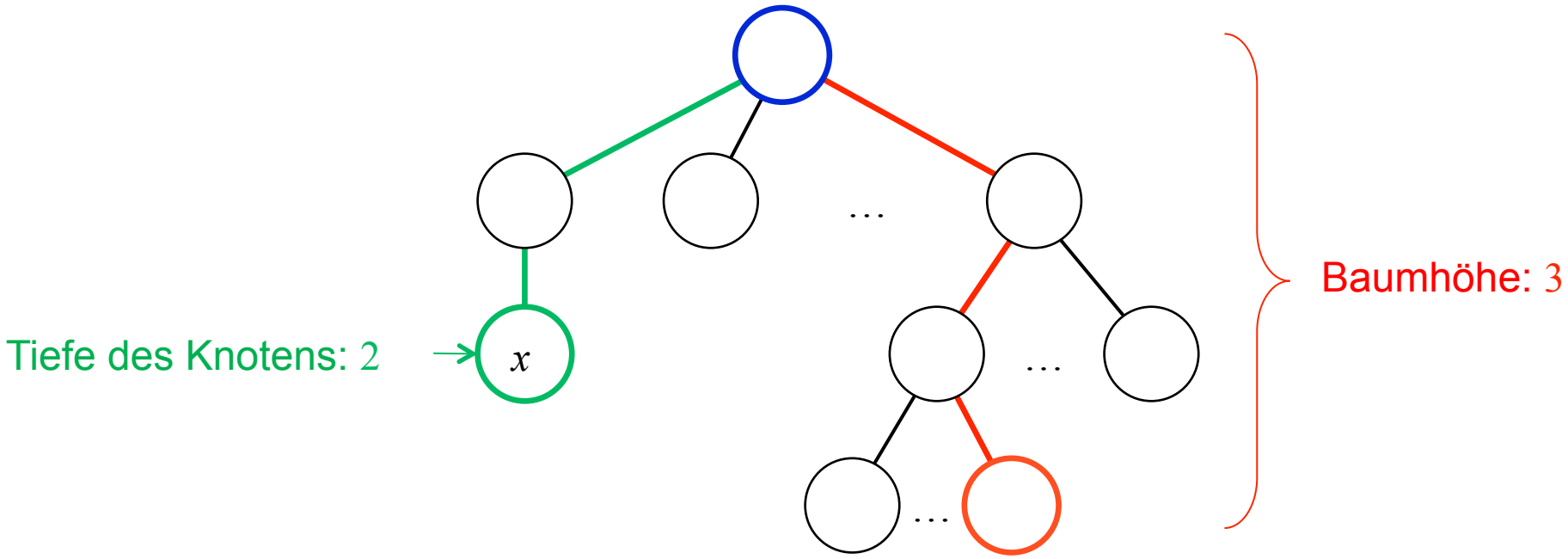
- entweder ein Blattknoten
- oder er hat genau zwei nicht leere Teilbäume.

Höhe eines Baums und Tiefe eines Knotens

Die **Höhe** (*height*) **eines Baumes** ist die maximale Pfadlänge von der Wurzel zu den Blättern des Baumes.

Bemerkung: Die Höhe wird bisweilen auch als Tiefe (*depth*) bezeichnet, siehe auch unten.

Die **Tiefe** (*depth*) [auch Ebene (*level*)] **eines Knotens** x ist die Länge des Pfades dieses Knotens zur Wurzel



Höhe und totale Anzahl von Knoten in perfekten Bäumen

Ein perfekter binärer Baum der Höhe h hat genau 2^h Blätter.

Für die totale Anzahl n von Knoten innerhalb eines solchen Baums gilt:

$$n = 1 + 2 + \ldots + 2^h = \sum_{k=0}^{h} 2^k = 2^{h+1} - 1$$

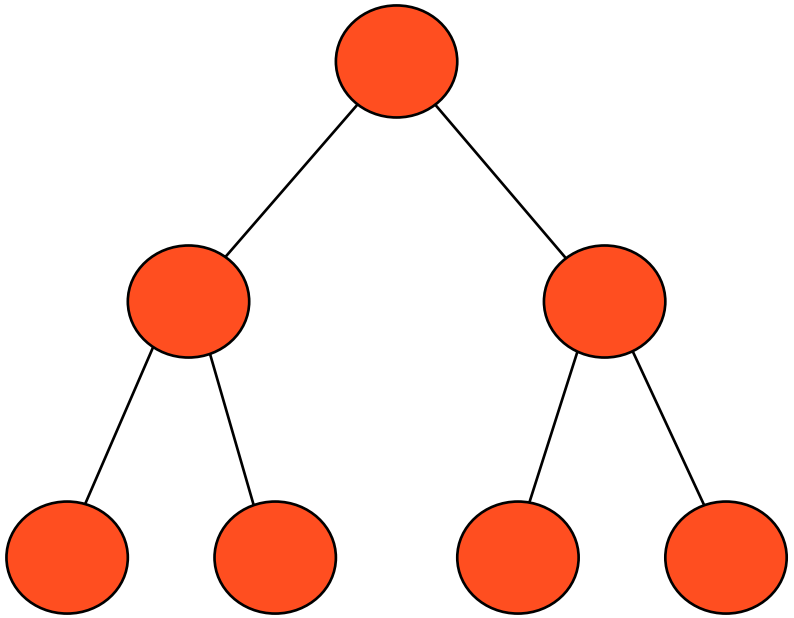
bzw.

$$h = \log_2(n+1) - 1$$

Generalisierungen: Für perfekte Bäume mit Höhe h und Ordnung f (*bzw. annäherungsweise für Bäume mit mittlerem fanout* f, $f \geq 2$) gilt:

- Die Anzahl Blätter ist genau f^h
- $n = 1 + f + \ldots + f^h = (f^{h+1} - 1) / (f - 1) \leq 2 \cdot (f^{h+1} - 1) / f \leq 2 \cdot f^h$
- $\log_f(n) \leq h \leq \log_f(n / 2)$

Beispiel: Perfekter binärer Baum



n = 7
h = 2

Traversieren eines Baums: "in-order"

Damit alle Knoten eines Baums bearbeitet werden können, muss der Baum **traversiert** werden (*tree traversal*).

Pseudocode für das "in-order" Traversieren eines Baums

```
procedure traverse(t: tree);
   begin
      if not empty(t) then
         traverse(t.left);        // rekursiver Aufruf: traversiere linken Teilbaum
         process(t.key);          // bearbeite den "Inhalt" des Wurzelknotens
                                  // (z.B. Ausgabe auf Bildschirm)
         traverse(r.right);       // rekursiver Aufruf: traversiere rechten Teilbaum
      end;
   end traverse;
```

Traversieren eines Baums: "pre-order"

Pre-Order: Verarbeite den "Inhalt" des Wurzelknotens zuerst.

Pseudocode für das "pre-order" Traversieren eines Baums

```
procedure preorder(t: tree);
   begin
      if not empty(t) then
         process(t.key);          // bearbeite den "Inhalt" des Wurzelknotens
                                  // (z.B. Ausgabe auf Bildschirm)
         preorder(t.left);        // rekursiver Aufruf: traversiere linken Teilbaum
         preorder(r.right);       // rekursiver Aufruf: traversiere rechten Teilbaum
      end;
   end preorder;
```

Traversieren eines Baums: "post-order"

Post-Order: Verarbeite den "Inhalt" des Wurzelknotens zuletzt.

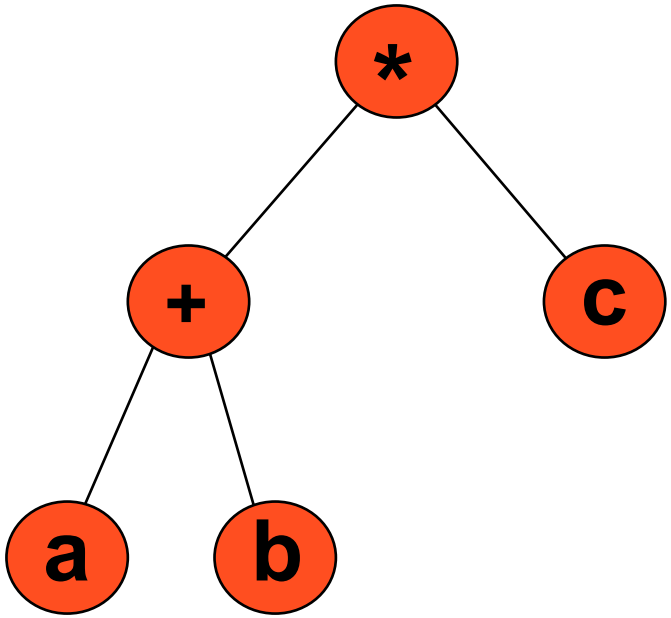
Pseudocode für das "post-order" Traversieren eines Baums

```
procedure postorder(t: tree);
   begin
      if not empty(t) then
         postorder(t.left);       // rekursiver Aufruf: traversiere linken Teilbaum
         postorder(r.right);      // rekursiver Aufruf: traversiere rechten Teilbaum
         process(t.key);          // bearbeite den "Inhalt" des Wurzelknotens
                                  // (z.B. Ausgabe auf Bildschirm)
      end;
   end postorder;
```

Beispiel: Binärer Baum (1)

Beispiel:

Arithmetischer Ausdruck (a+b)*c

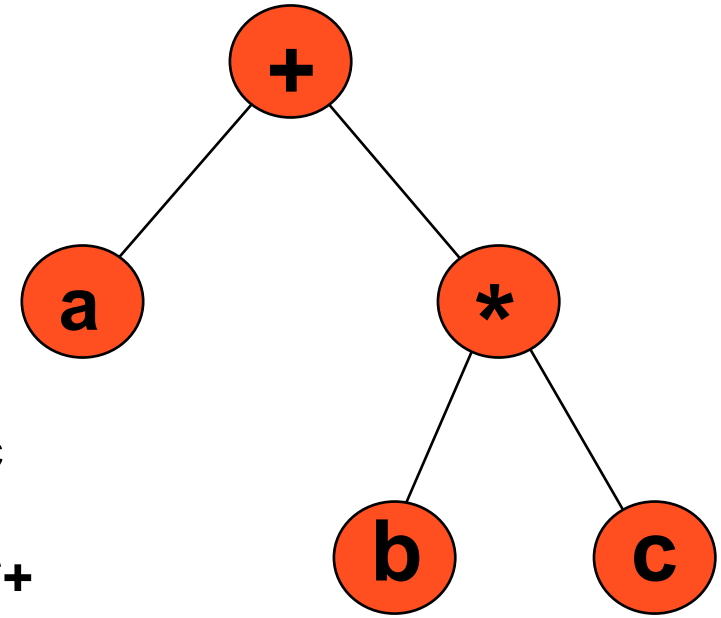


pre-order: *+abc
in-order: (a+b)*c
post-order: ab+c*

Beispiel: Binärer Baum (2)

Beispiel:

Arithmetischer Ausdruck a+b*c



pre-order: +a*bc
in-order: a+b*c
post-order: abc*+

Binärer Sortierbaum

Ein **binärer Sortierbaum** ist ein geordneter binärer attributierter Baum für den beim in-order Traversieren alle Attribute gemäss ihrer Ordnungsrelation durchlaufen werden.

Seien

- x ein Knoten eines binären Sortierbaums T
- $x.key$ das Attribut des Knotens x sowie
- $x.left$ und $x.right$ der linke respektive rechte Teilbaum des Knotens x,

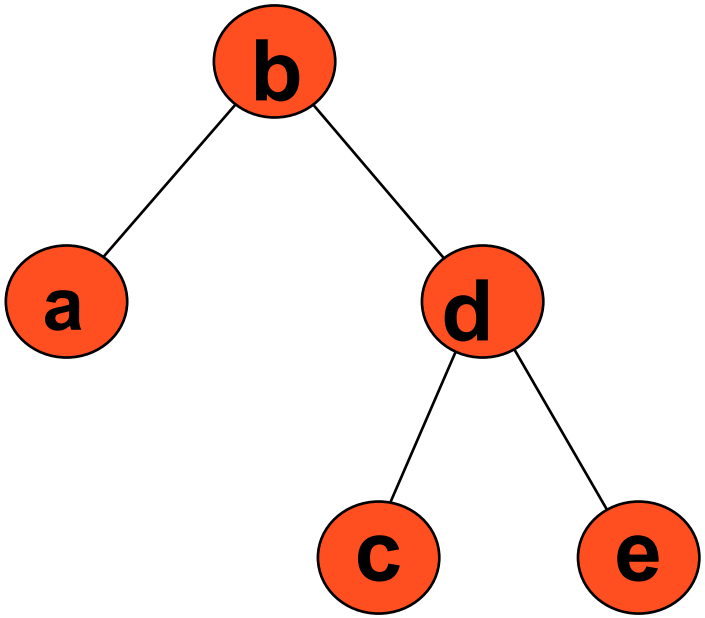
dann gilt (bei ansteigender Sortierteihenfolge) eines Sortierbaumes T:

$$\forall\, x:\ x \in T:$$
$$(\forall\, y:\ y \in x.left:\ y.key \leq x.key)\ \wedge\ (\forall\, y:\ y \in x.right:\ y.key \geq x.key)$$

Jeder Teilbaum eines Sortierbaums ist selbst ein Sortierbaum

Beispiel: Binärer Baum als Sortierbaum

Binärer Sortierbaum



in-order: abcde
pre-order: badce
post-order: acedb
level-order: badce (zuerst Level 0 (= Wurzel), dann Knoten auf Level 1, …)

Traversieren eines Baums: ”level-order"

Level-Order: Zuerst Level 0 (Wurzel), dann Level 1 Knoten, ...)

```
procedure traverseUntil(t: tree, maxlevel: integer, level: integer);
   begin
      if not empty(t) then
         if  level  = maxlevel  then
            process(t.key);
         else
            traverseUntil (t.left, maxlevel, level+1);
            traverseUntil (r.right, maxlevel, level+1);
      end;
   end traverseUntil;

procedure levelOrder(t: tree);          // Dies ist eine Implementation namens „iterative deepening“
   begin
      for  level  := 0  to  depth(t)  do
         traverseUntil(t, level, 0);
      end;
   end levelOrder;
```

Warum sind Sortierbäume wichtig?

Gegeben eine ungeordnete Liste mit n Datensätzen (*records*), z.B. Informationen über Personen.

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Um einen Datensatz mit einem bestimmten Namen zu finden, muss/müssen

- im besten Fall 1 Datensatz (*)
- im schlechtesten Fall n Datensätze

angeschaut werden, (im Mittel $^{n+1}/_{2}$)

Bei einem perfekten binären Baum sind es

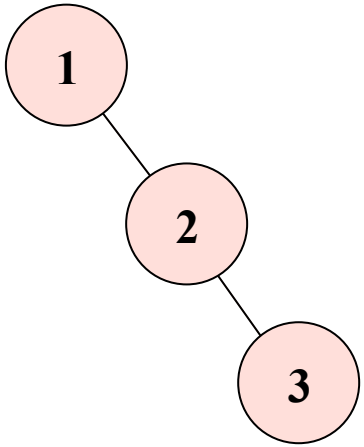
- im besten Fall 1 Datensatz (*)
- im schlechtesten Fall $h = \log_2(n+1) - 1$ Datensätze.
  (bei $n = 1'000'000$ sind dies noch 19 Datensätze)

(*) Dies gilt nur, wenn die Namen eindeutig sind!

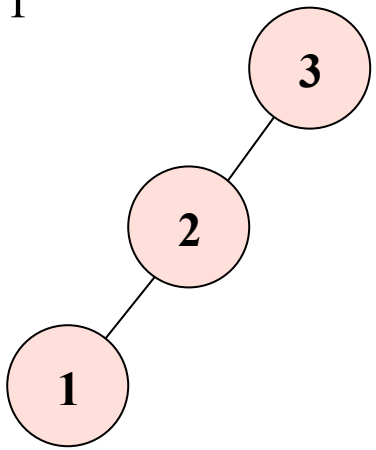
Problem: Sortierbäume sind nicht immer balanciert ...

Reihenfolge des Einfügens:

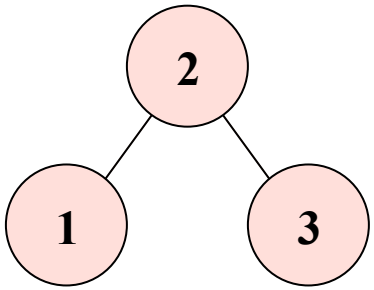
1, 2, 3



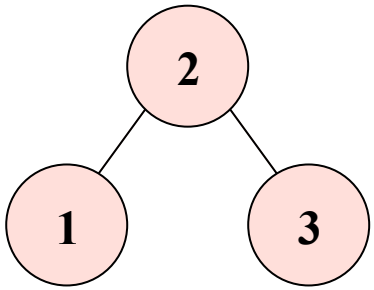
3, 2, 1



2, 1, 3



2, 3, 1



B Bäume

B Bäume und ihre Varianten (B+ Baum) wurden 1972 von Rudolf Bayer und Ed McCreight zur Indizierung grosser Datenmengen auf Disk erfunden.

B Bäume sind Mehrwegbäume, wobei **B** primär für **B**alanced steht
(und nebenbei vielleicht für **B**oeing und/oder Rudolf **B**ayer)

Eigenschaften eines B+ Baums

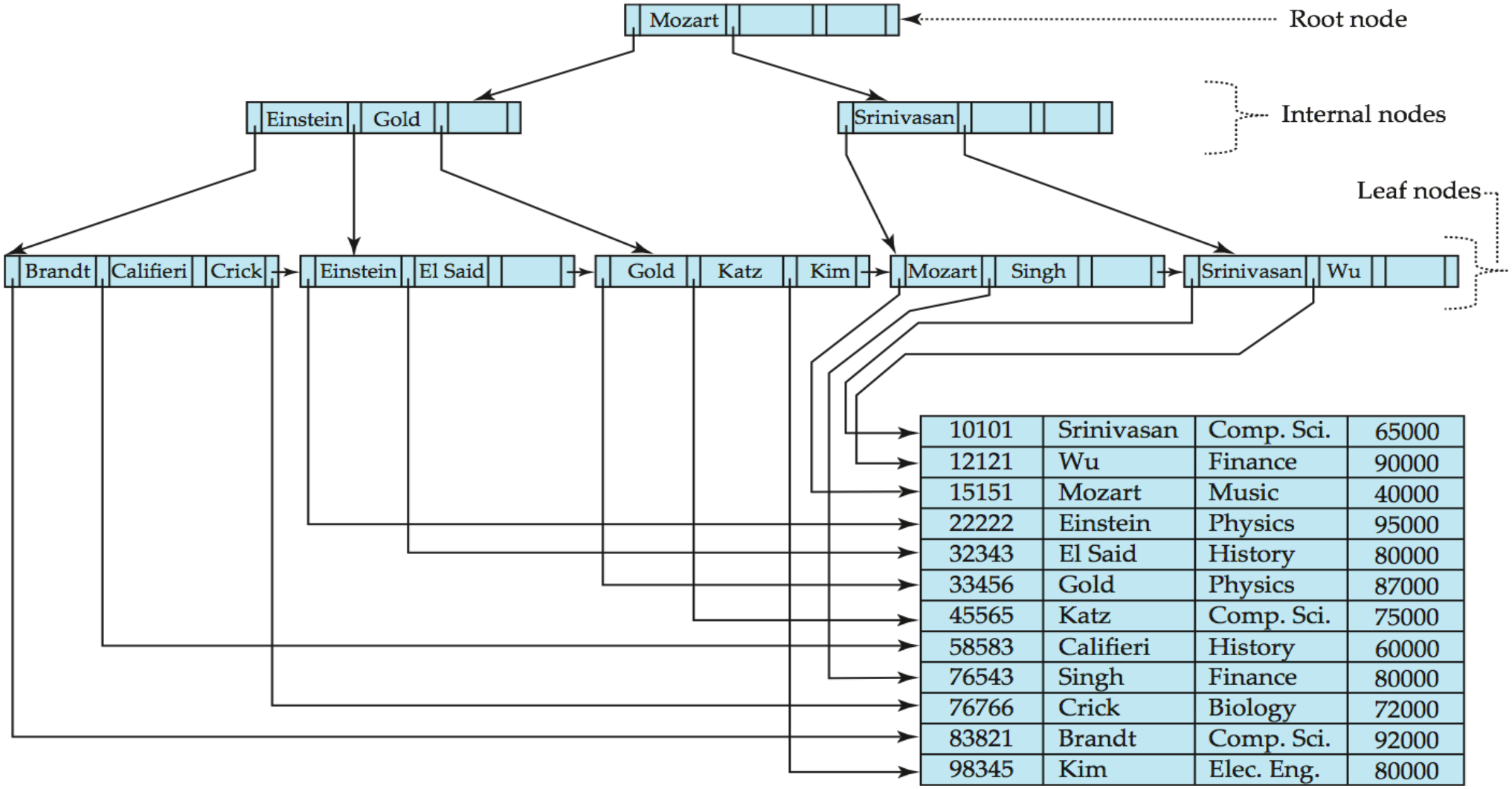
- Die minimale und maximale Pfadelänge von der Wurzel zu einem Blatt unterscheidet sich höchstens um 1.
- Jeder "innere Knoten" hat zwischen $\lceil n/2 \rceil$ und n Unterbäume.
- Ein Blattknoten hat zwischen $\lceil (n-1)/2 \rceil$ und $n-1$ Werte

Spezialfälle:

Wenn die Wurzel kein Blattknoten ist, dann hat sie mindestens 2 Unterbäume.

Wenn die Wurzel ein Blattknoten ist (und somt der einzige Knoten des Baums ist), kann sie zwischen 0 und (n–1) Werte haben.

Beispiel eines B$^+$-Baums



Fanout in B-Bäumen

- In Datenbanksystemen sind die Daten auf Disk in Blöcken von 8 kByte oder 16 kByte gespeichert.
- Der Fanout f berechnet sich aus der Blockgrösse S, dem Platzbedarf für einen Indexwert V sowie dem Platzbedarf für einen Pointer auf ein Tupel (sog. TID) P:

  $$f = \lfloor (S-P) / (V+P) \rfloor$$

  Beispiele für 8kByte Seiten (ohne Header-Info ≈ 8000 Byte, 8-byte TID)

  $\lfloor (8000-8) / (4+8) \rfloor = 666$ für 4-Byte INTEGER

  $\lfloor (8000-8) / (256+8) \rfloor = 30$ für 256-Byte String

- Worst-Case-Komplexität (Anzahl der Seitenzugriffe) für Suchen, Einfügen und Löschen in einem B-Baum mit insgesamt n Datensätzen: $O(log_f n)$.
- Mittlere Speicherplatzauslastung bei zufälligem Einfügen: $ln\ 2 \approx 0.69$.