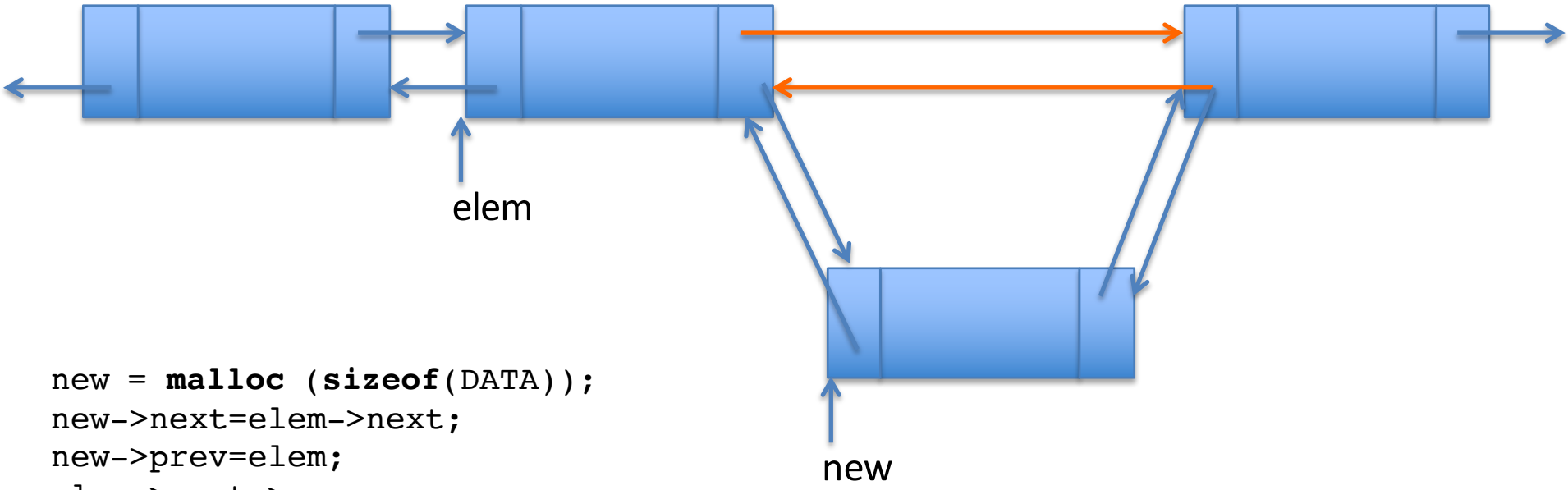
Doppelt verkettete Listen

- Vorteil:
  - schnelleres Navigieren durch Liste

```
typedef struct data {
    char name[MAX_LEN];
    char vorname[MAX_LEN];
    struct data *next;
    struct data *prev;
}DATA;
```

Doppelt verkettete Listen

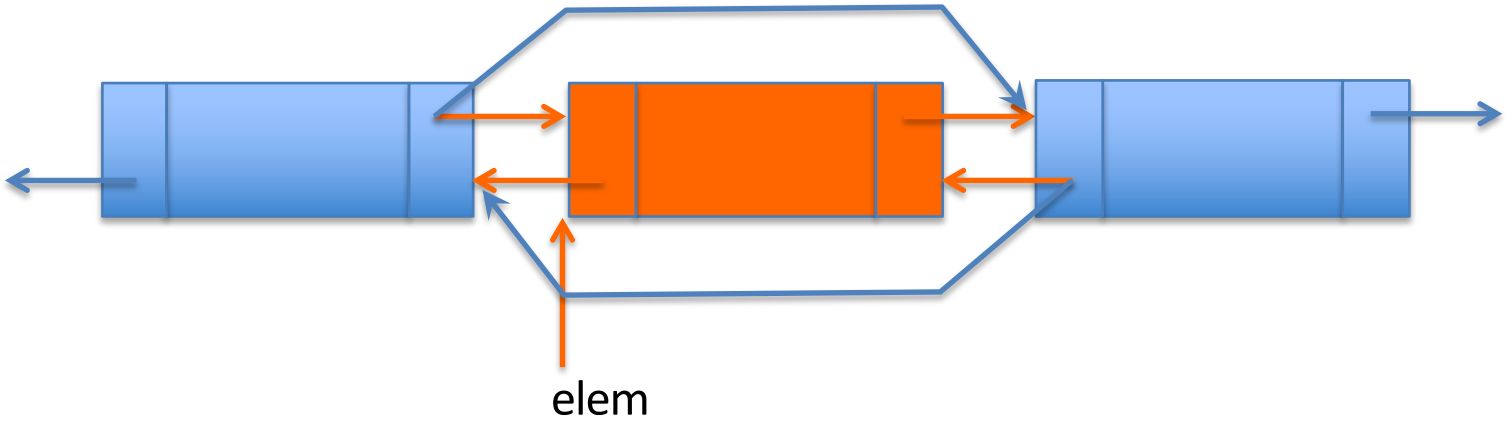
- Einfügen von Elementen




```
new = malloc (sizeof(DATA));
new->next=elem->next;
new->prev=elem;
elem->next->prev = new;
elem->next = new;
```

Doppelt verkettete Listen

- Löschen von Elementen



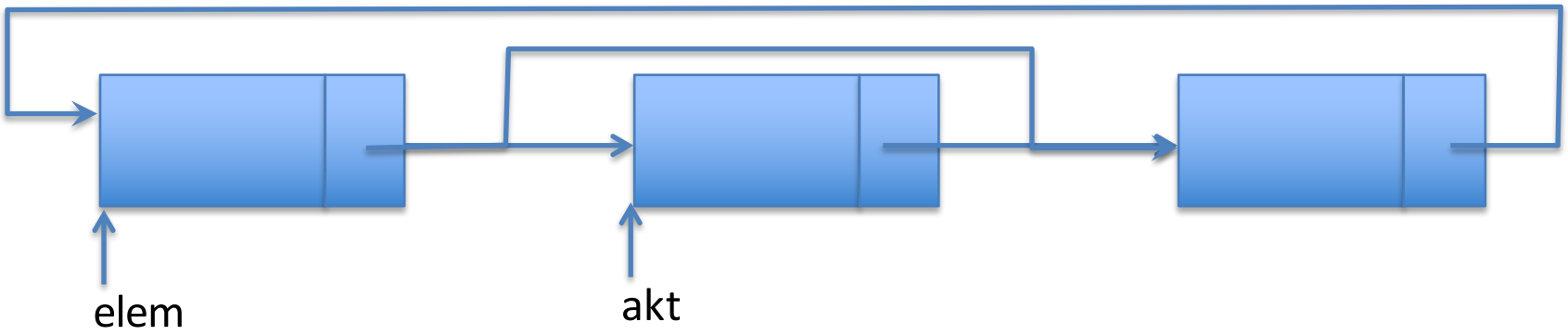

```
elem->prev->next=elem->next;
elem->next->prev = elem->prev;
free (elem);
```

Ringstruktur

- Spezialfall einer verketteten Liste
- Letztes Element ist mit ersten verbunden
- gibt keinen Startknoten
- ein Knoten muss immer bekannt sein
- Achtung bei erstem/letztem Knoten

Ringstruktur

- Element löschen (einfach verkettet)

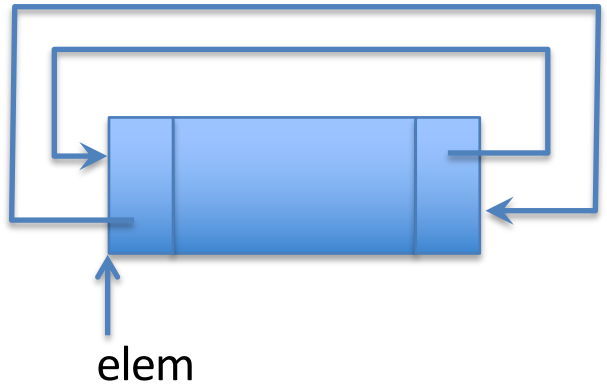



```
akt=elem->next;
elem->next = akt->next; // oder: elem->next->next;
free(akt);
```

Ringstruktur

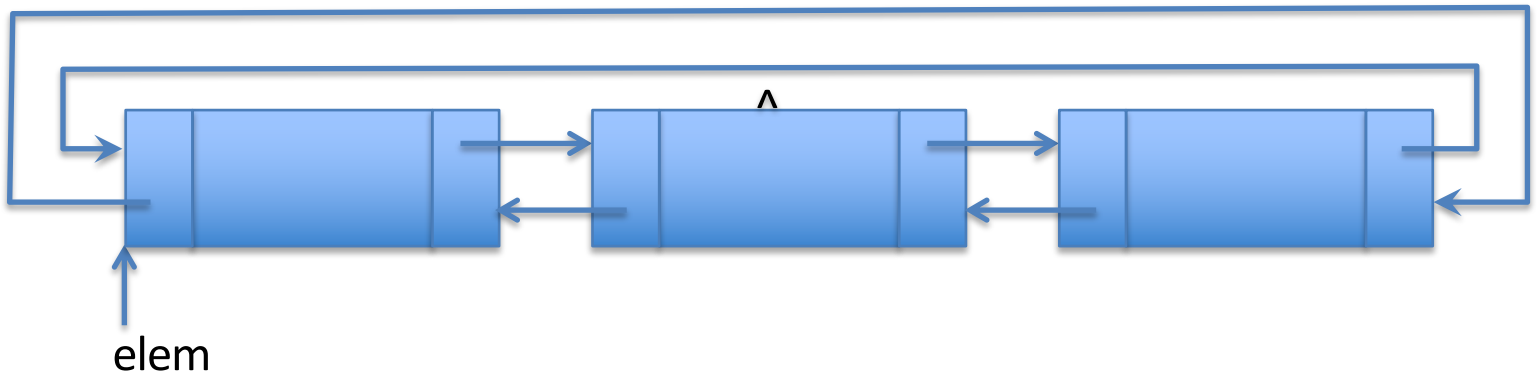
- Element erzeugen (doppelt verkettet)

```
elem=malloc(sizeof(DATA);
elem->next = elem;
elem->pref = elem;
```



Ringstruktur

- Element einfügen (doppelt verkettet)




```
akt=malloc(sizeof(DATA));
akt->next=elem->next;
akt->prev=elem;
elem->next->prev  = akt;
elem->next  = akt;
```

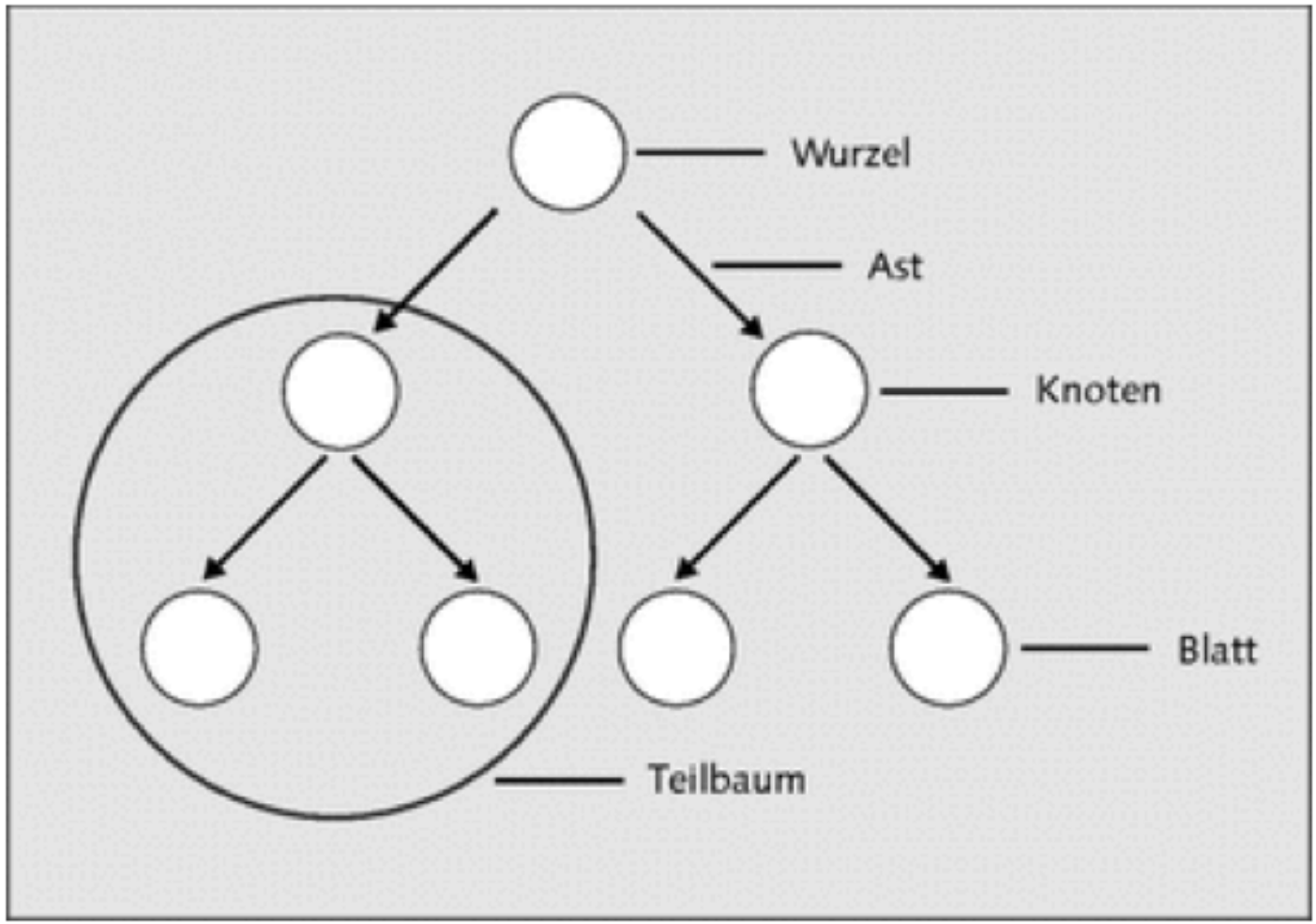
Bäume

- Bäume (Trees)
  - sind hierarchis
  - Name kommt v
  - Baum, der auf
  - Elemente des B

Bäume

- Wurzel (root)
  - Einziger Knoten, der keinen Vorgänger besitzt
- Ast, Kante (edge)
  - Knoten sind mit Ast verbunden
- Blatt (Leaf)
  - Knoten ohne Nachfolger
- Teilbaum
  - Knoten mit linken + rechten Nachfolger

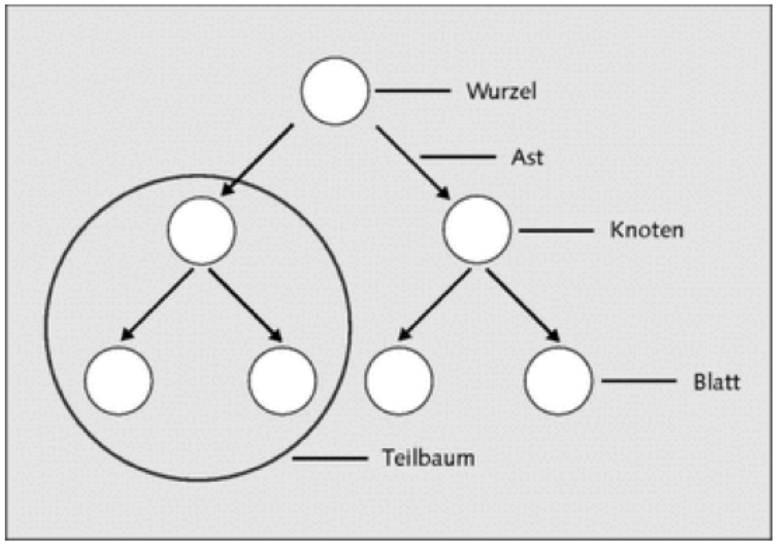
Bäume



Bäume

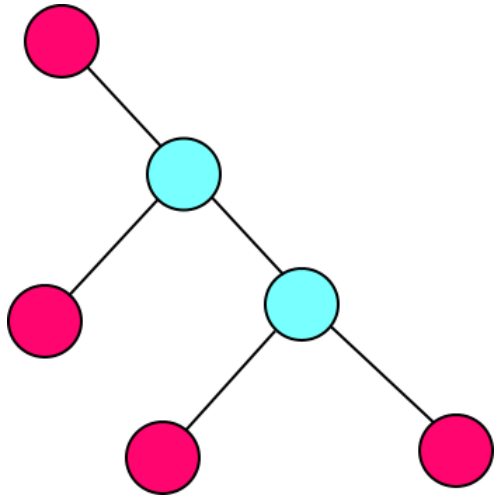
- Tiefe:
  - Anzahl der Äste von der Wurzel bis zu Blatt
- Grad:
  - Anzahl an Ästen

    eines Knoten



Bäume

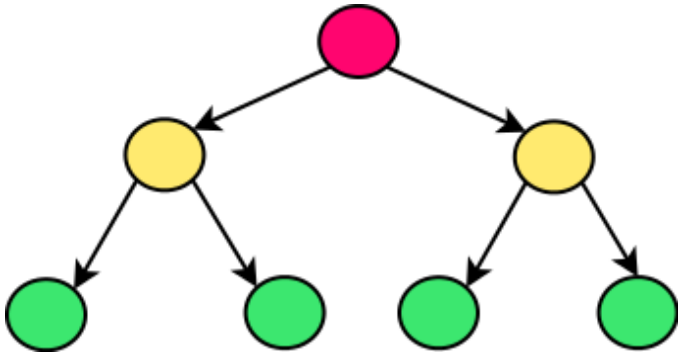
- Ungerichteter Baum:
  - zusammenhängende Knoten
  - Kreisfrei
  - Knoten mit Grad 1 => Blätter
  - Alle anderen Knoten:
    - Innere Knoten



ungerichteter Baumes mit zwei inneren Knoten (blau) und vier Blättern (rot)

Bäume

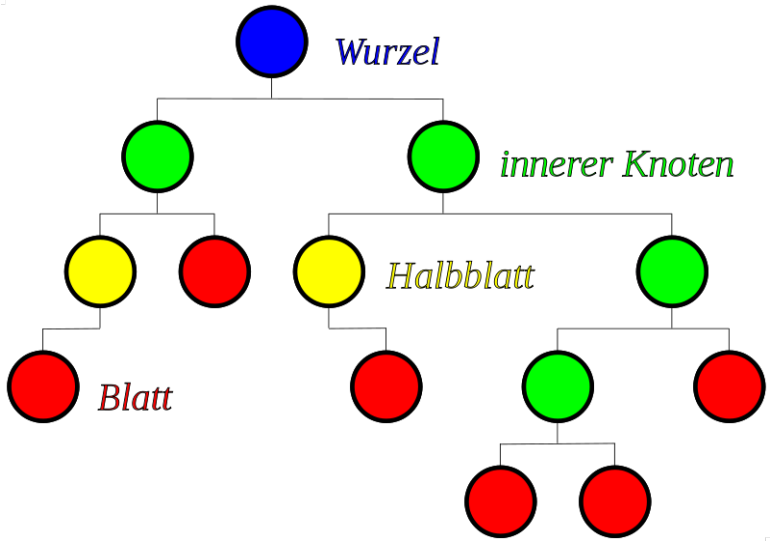
- Gerichteter Baum
  - genau eine Wurzel
  - kreisfrei
  - Wurzel
    - Knoten mit Eingangsgrad 0
  - Blätter
    - Knoten mit Ausgangsgrad 0



Wurzel (rot), zwei inneren Knoten (gelb), vier Blättern (grün)

Bäume

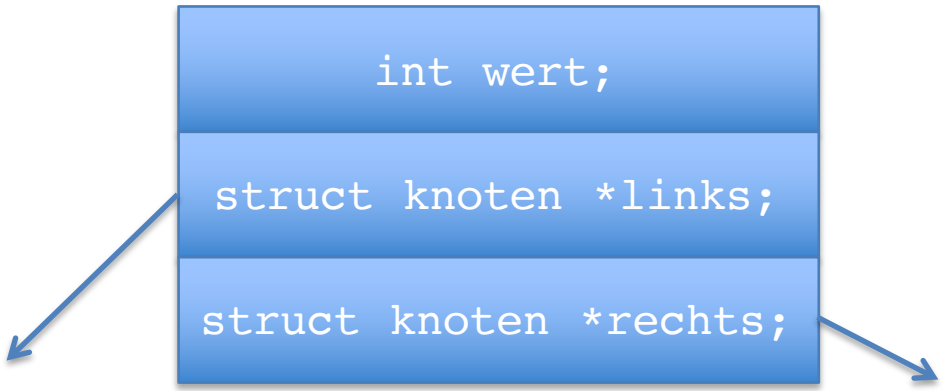
- Binärbaum
  - Spezialform eines Baumes
  - Jeder Knoten hat maximal 2 Kindknoten
  - Bsp. Ahnentafel



Bäume

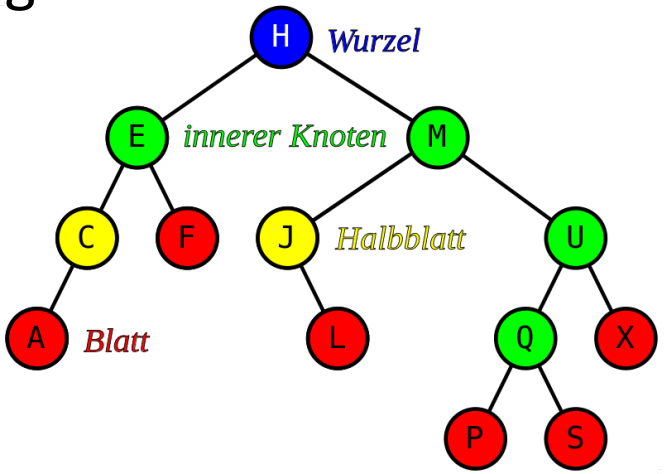
- Datenstruktur

```
struct knoten {
    int wert;
    struct knoten *links;
    struct knoten *rechts;
};
```



Bäume

- Binärer Suchbaum
  - Ermöglicht sehr effizientes Suchen: O(h)
  - Schlüssel in Knoten
    - Schlüssel des linken Teilbaums „kleiner“
    - Schlüssel des rechten Teilbaums „größer“



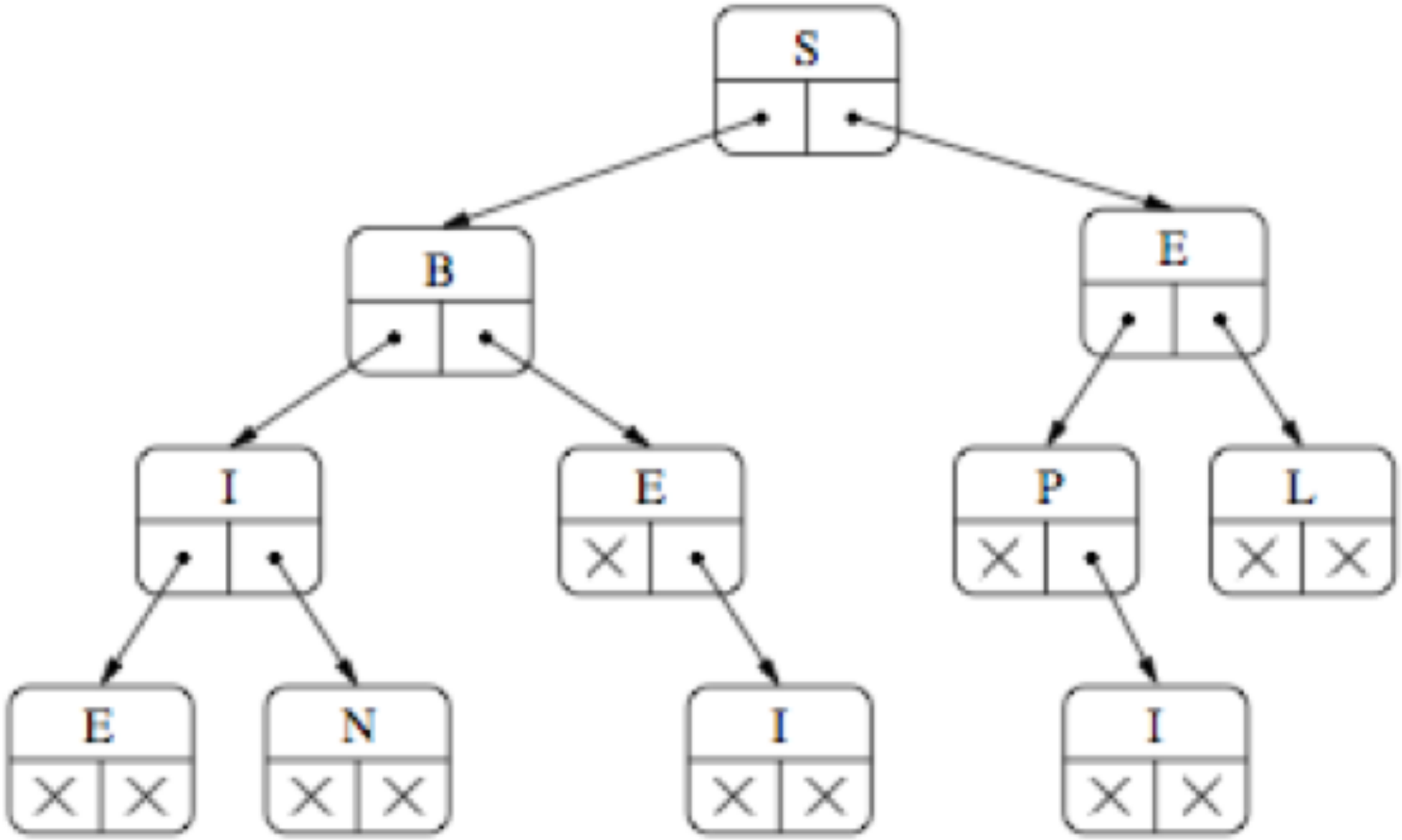
Bäume

- Einfügen in binärer Suchbaum

Bäume

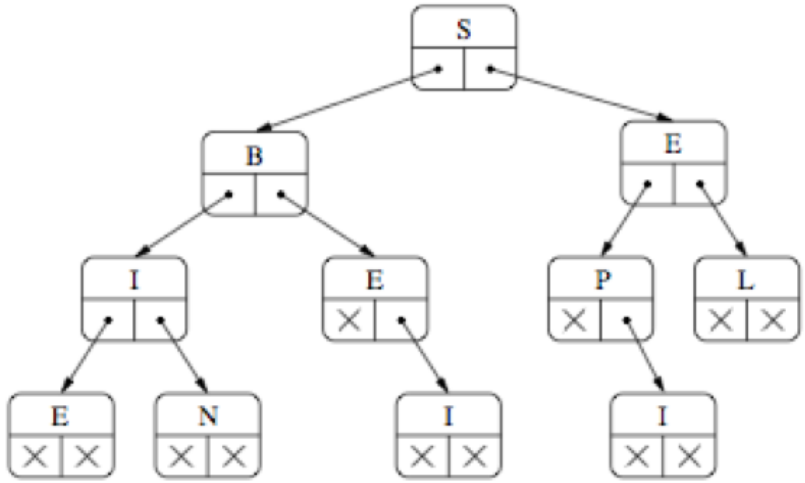
- Durchlaufen von Binären Suchbäumen
  - Präfix: N – L – R
    - Zuerst Knoten (N)
    - dann Linker Teilbaum (L)
    - dann rechter TB(R)
  - Infix: L – N – R
  - Postfix: L – R - N

Bäume



Bäume

- Präfix: N – L – R



S, {Unterbaum B}, {Unterbaum E1} =

S, B, {Unterbaum I}, {Unterbaum E2}, {Unterbaum E1} =

S, B, I, E, N, {Unterbaum E2}, {Unterbaum E1} =

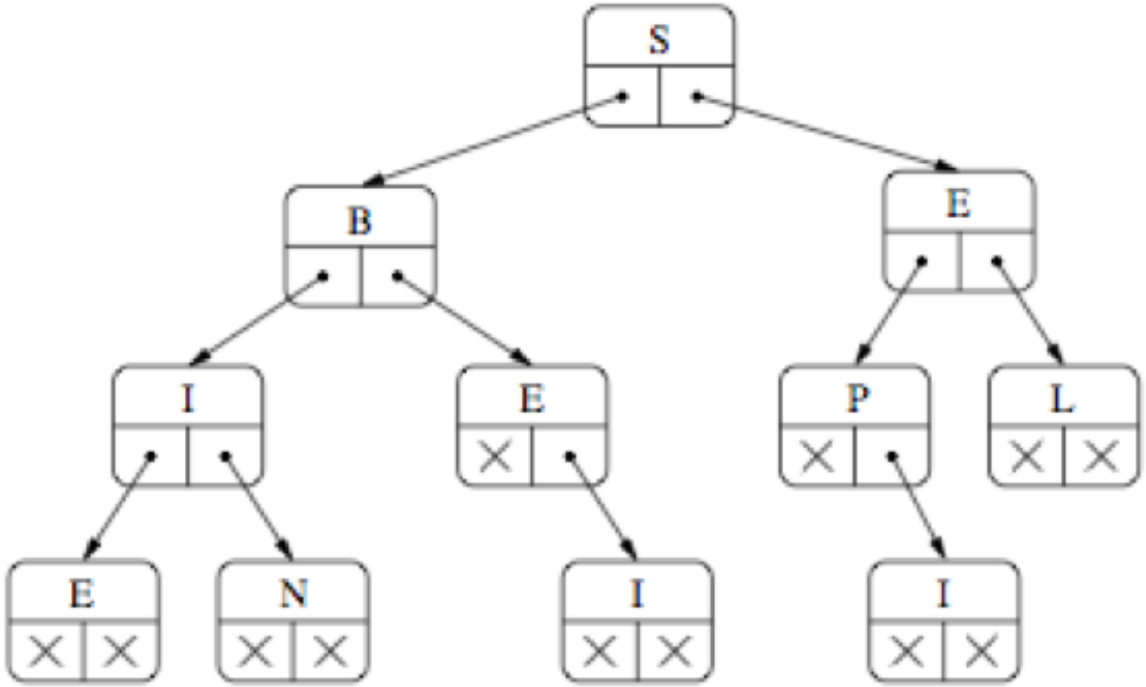
S, B, I, E, N, E, I, {Unterbaum E1} =

S, B, I, E, N, E, I, E, {Unterbaum P}, {Unterbaum L}=

'S'-'B'-'I'-'E'-'N'-'E'-'I'-'E'-'P'-'I'-'L'

Bäume

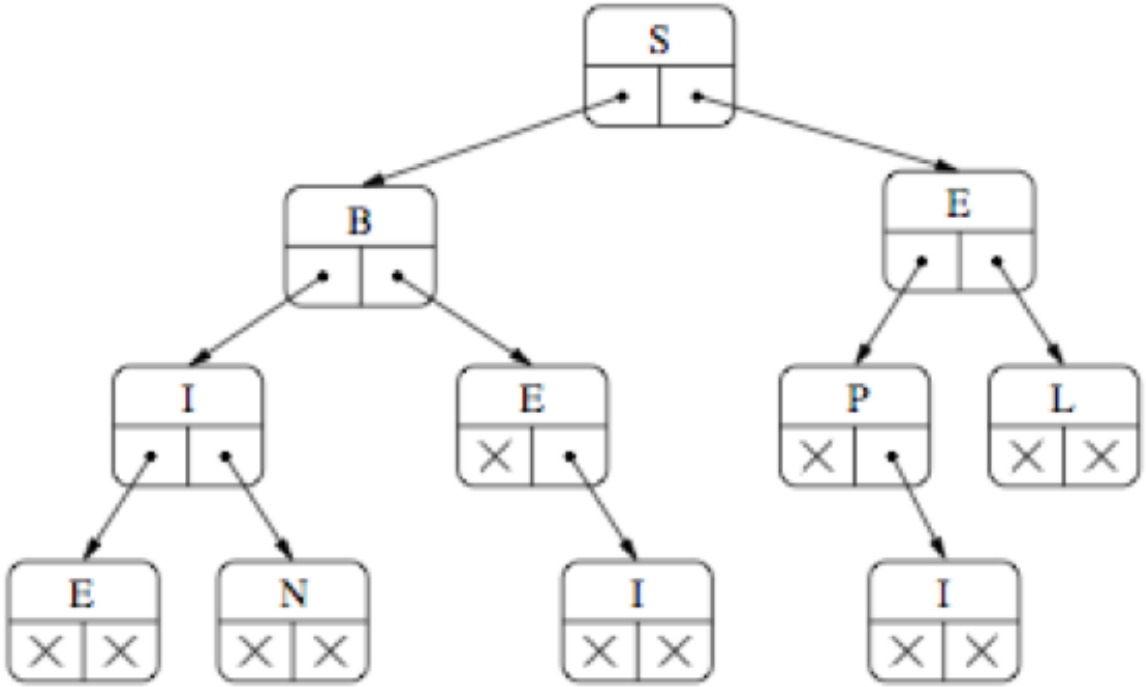
- Infix: L – N - R



‘E’-‘I’-‘N’-‘B’-‘E’-‘I’-‘S’-‘P’-‘I’-‘E’-‘L’

Bäume

- Postfix: L - R - N



‘E’-‘N’-‘I’-‘I’-‘E’-‘B’-‘I’-‘P’-‘L’-‘E’-‘S’

Bäume

- Entscheidungsbäume
  - Darstellung von Entscheidungsregeln
  - Visualisiert hierarchische, aufeinander folgende Entscheidungen
  - Ermöglicht automatische Klassifizierung
  - Beispiele?

Bäume

- Klassifizierung mit Entscheidungsbäumen
  - vom Wurzelknoten abwärts
  - bei jedem Knoten Attribut abfragen und Entscheidung über Weg treffen
  - wird solange wiederholt, bis man Blatt erreicht
  - Blatt = Klassifizierung
  - EIN Baum = Regeln zur Beantwortung EINER Frage

Bäume

- Bsp. Entscheidungsbäume
  - Frage: „Trägt Apfelbaum Früchte?“