Geordnete Binärbäume

Gilbert Beyer und Annabelle Klarl

Zentralübung zur Vorlesung Einführung in die Informatik

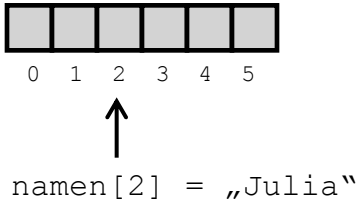
http://www.pst.ifi.lmu.de/Lehre/wise-11-12/infoeinf

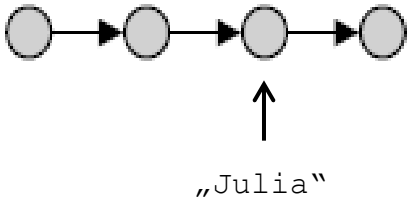
Inhalte der heutigen Vorlesung:

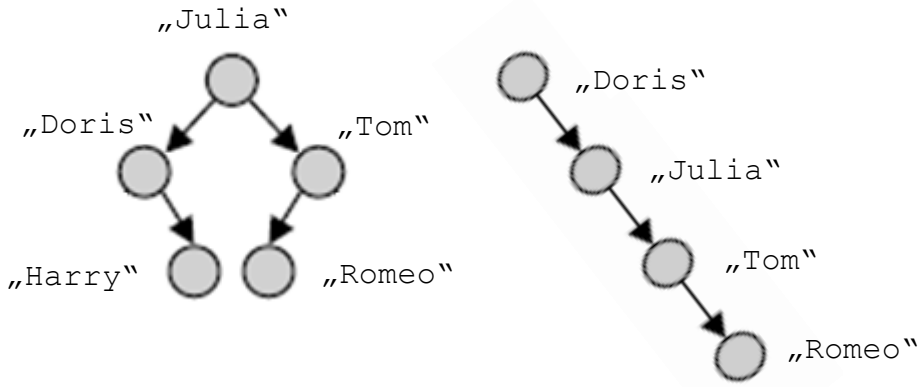
- Bäume
- Terminologie, Anwendungen
- Binärbäume in Java
- Klassen für Binärbäume
- Konstruktion eines Baums
- Operationen auf Binärbäumen
- Der Baum im Heap
- Durchlaufen eines Baums
- Baumdurchlauf mit Iteratoren

Geordnete Binärbäume

- **Motivation**
  - Eine Datenstruktur für ein Namensverzeichnis soll implementiert werden
  - Die Datenstruktur soll die Operationen `find`, `insert` und `print` anbieten
  - Mit der Zeit wird die Menge an Namen wachsen (>1000)
- **Beispiele**





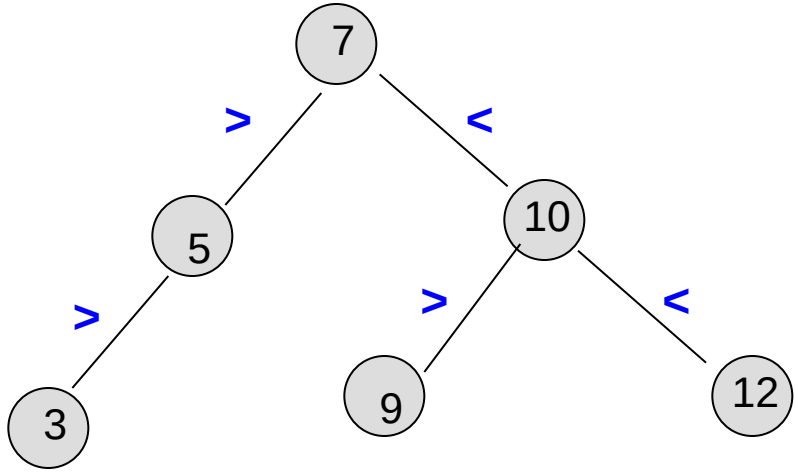


1. Array
2. Liste
3. Geordnete Binärbäume
   (balanciert und zu einer Liste entartet)

Geordnete Binärbäume

- Ein Binärbaum b heißt **geordnet**, 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

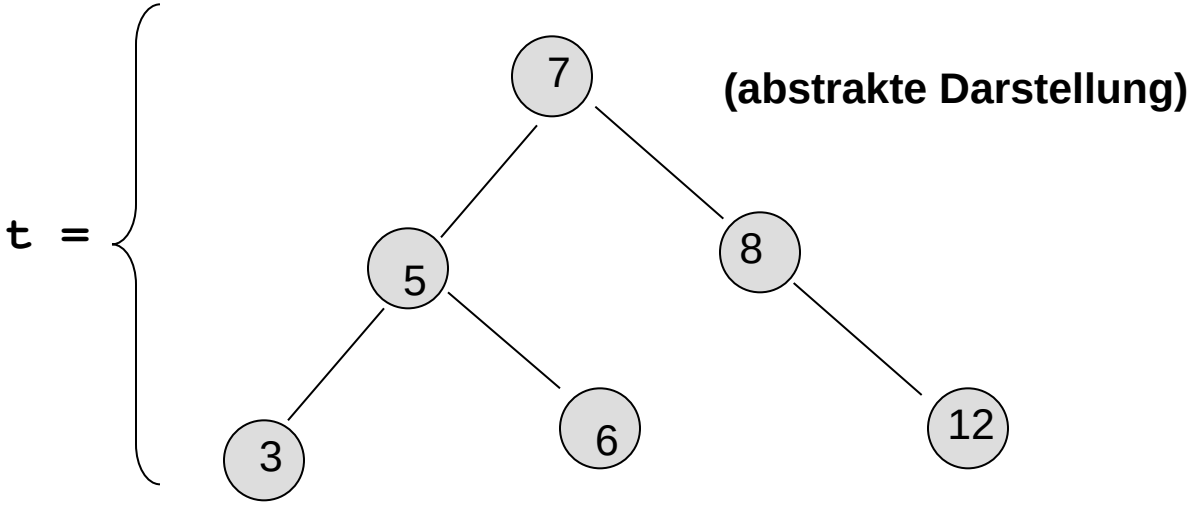


Geordnete Binärbäume

- **Beispiel: Geordnet** sind:

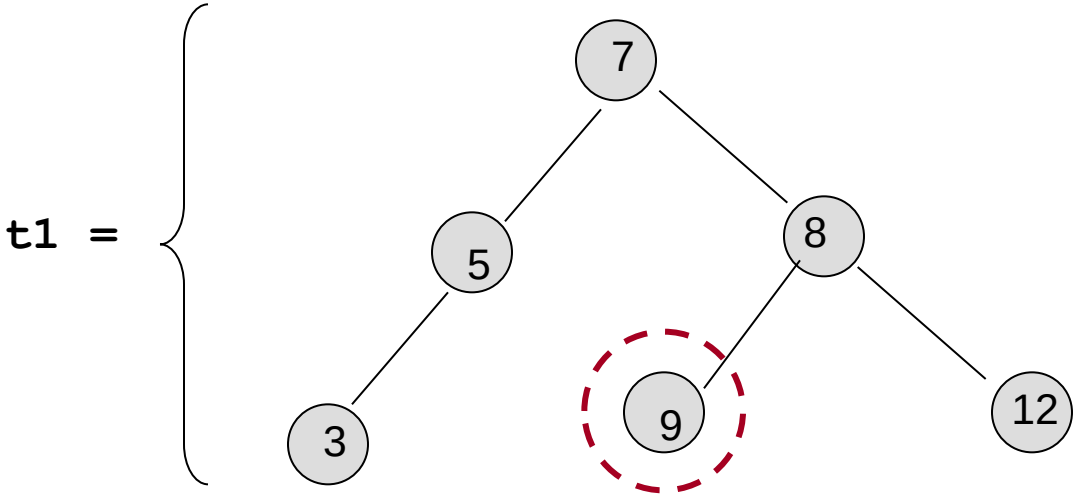
  Der leere Baum und

  der Baum t:



- **Nicht geordnet** ist

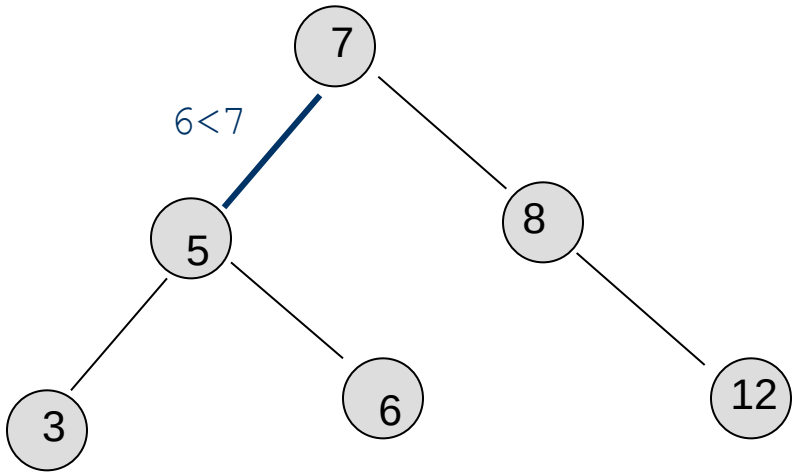
  der Baum t1:



Suche im geordneten Binärbaum

Prinzipieller Ablauf der Berechnung von t.find(6):

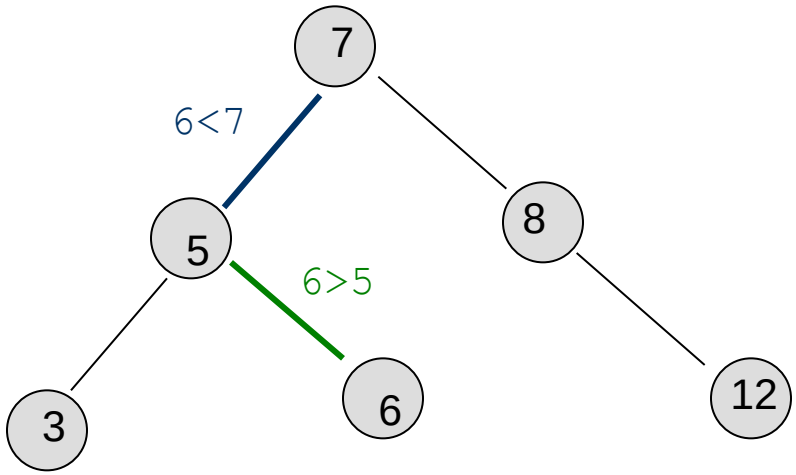
1. Vergleiche `6` mit dem Wert der Wurzel;
2. Da `6<7`, gehe zum linken Kindknoten;



Suche im geordneten Binärbaum

Prinzipieller Ablauf der Berechnung von t.find(6):

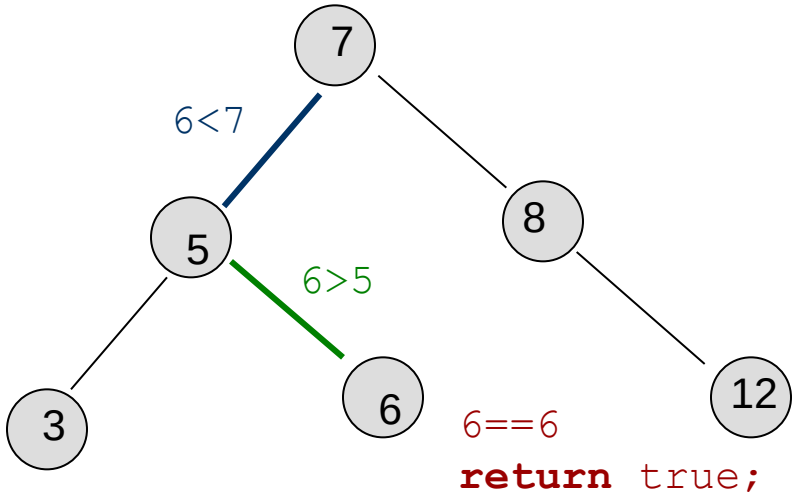
1. Vergleiche `6` mit dem Wert der Wurzel;
2. Da `6<7`, gehe zum linken Kindknoten;
3. Vergleiche `6` mit dem Wert dieses Knotens;
4. Da `6>5`, gehe zum rechten Kindknoten dieses Knotens;



Suche im geordneten Binärbaum

Prinzipieller Ablauf der Berechnung von t.find(6):

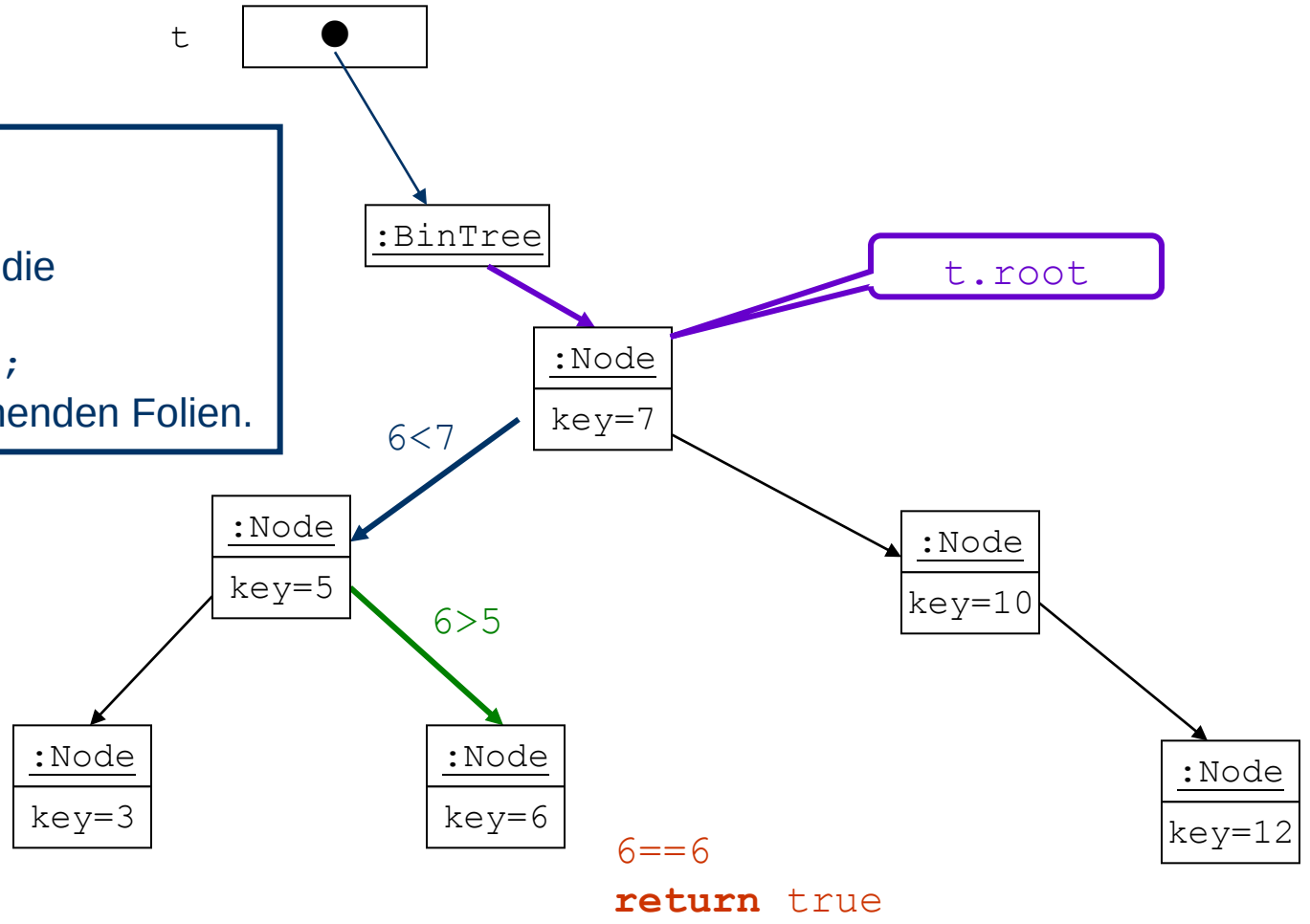
1. Vergleiche `6` mit dem Wert der Wurzel;
2. Da `6<7`, gehe zum linken Kindknoten;
3. Vergleiche `6` mit dem Wert dieses Knotens;
4. Da `6>5`, gehe zum rechten Kindknoten dieses Knotens;
5. Vergleiche `6` mit dem Wert dieses Knotens;
6. Da `6==6`, gebe Ergebnis true zurück.



Suche im geordneten Binärbaum (Implementierung)

`t.find(6):`

1. Suche zunächst in BinTree.
2. Wenn t nicht leer, delegiere die Aufgabe an Node durch Aufruf von `root.find(6);`
3. Verfahre wie auf den vorgehenden Folien.



```
public class BinTree
{ . . .
    public boolean find(int key)
    { if (root == null) return false;
      else return root.find(key);
    }
  . . .
}
class Node
{ . . .
     boolean find(int key)
     {  Node current = this;
        while(current.key != key)          // solange nicht gefunden,
        { if (key < current.key)           // gehe nach links?
            current = current.left;
          else                              // sonst gehe nach rechts
            current = current.right;
          if(current == null) return false; //nicht gefunden!
        }
        return true;                      //gefunden; gib true zurück
     }
```

Gibt `true` zurück, wenn `key` im Baum; sonst wird `false` zurückgegeben

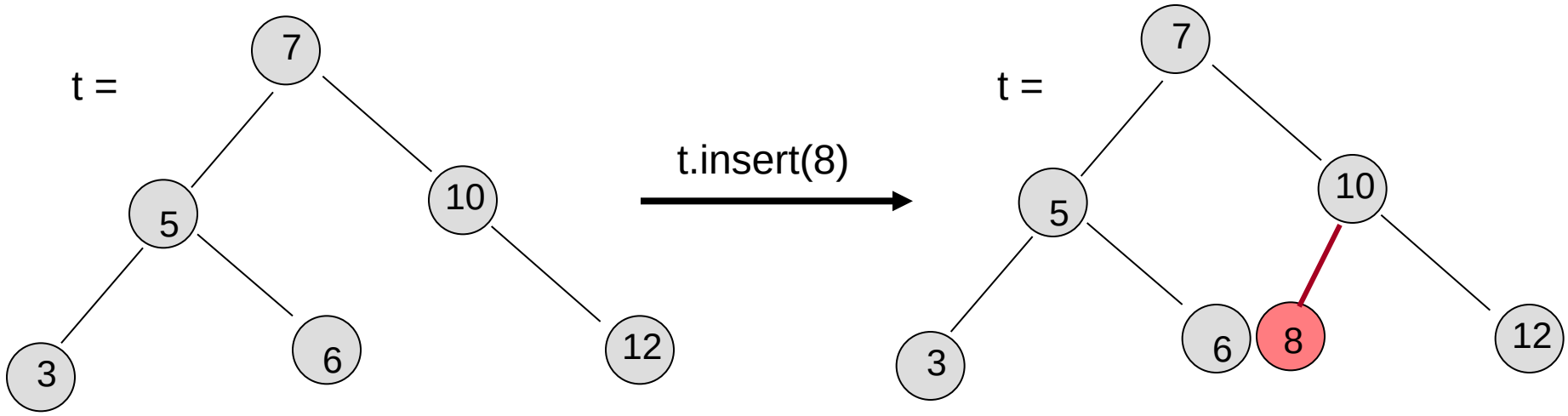
```
public class BinTree
{ . . .
    public Object findValue(int key)
    { if (root == null) return null;
      else return root.findValue(key);
    }
  . . .
}
class Node
{ . . .
     Object findValue(int key)
     {  Node current = this;
        while(current.key != key)          // solange nicht gefunden,
        { if (key < current.key)           // gehe nach links?
            current = current.left;
          else                              // sonst gehe nach rechts
            current = current.right;
          if(current == null) return null; //nicht gefunden!
        }
        return current.value;          //gefunden; gib true zurück
     }
```

Gibt `value` zurück, wenn `key` im Baum; sonst wird `null` zurückgegeben

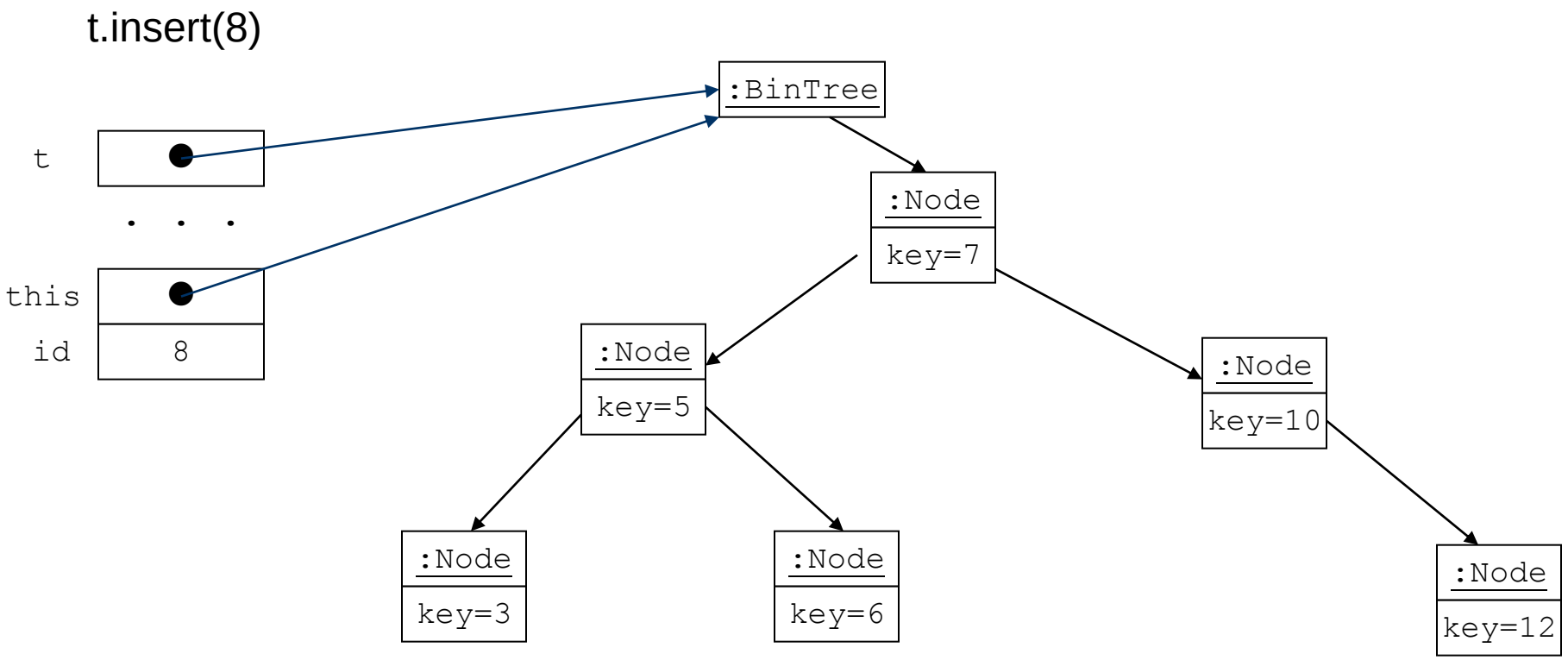
Einfügen in geordneten Binärbaum

- Beim Einfügen in einen geordneten Binärbaum wird rekursiv die “richtige” Stelle gesucht, so dass wieder eine geordneter Binärbaum entsteht.
- Beispiel: t.insert(8) ergibt:

(Zur Vereinfachung der Darstellung wird hier nur ein Schlüssel und kein Wert eingefügt.)

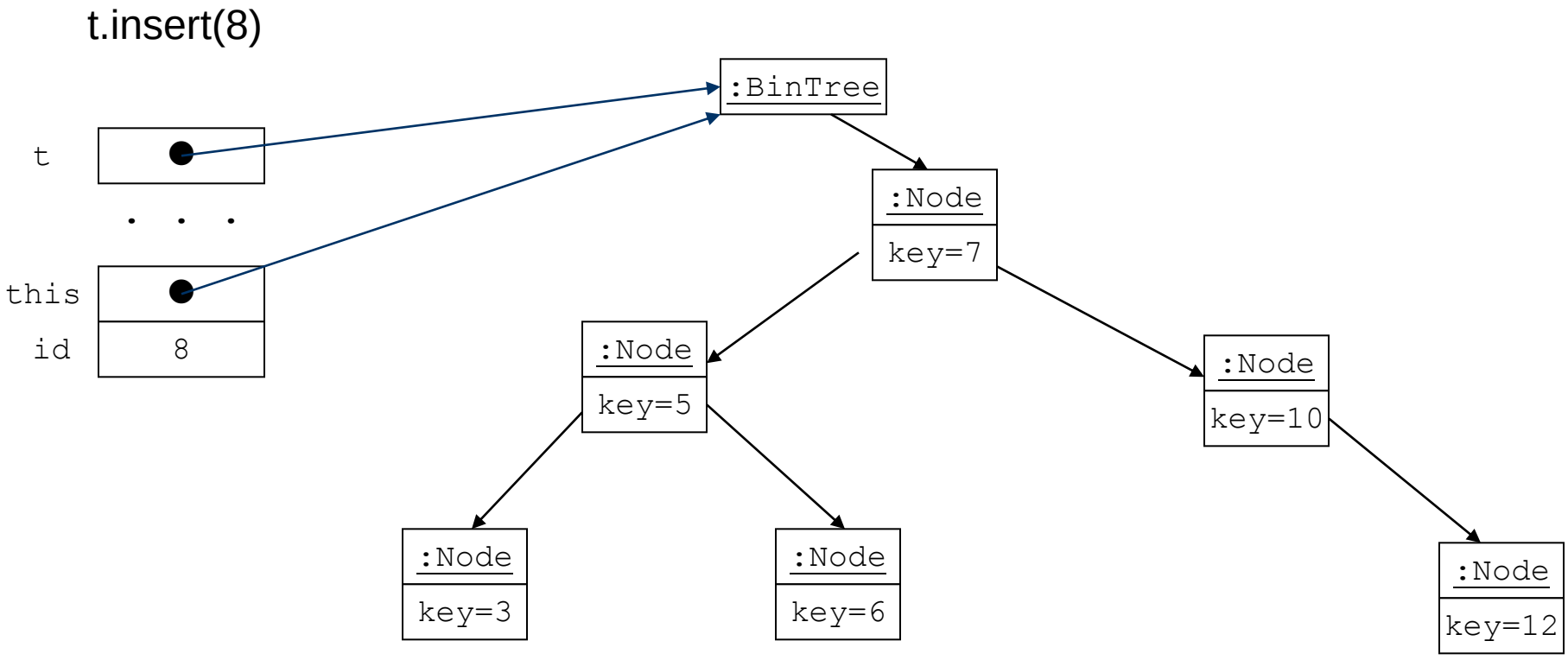


Einfügen in geordneten Binärbaum (Implementierung)



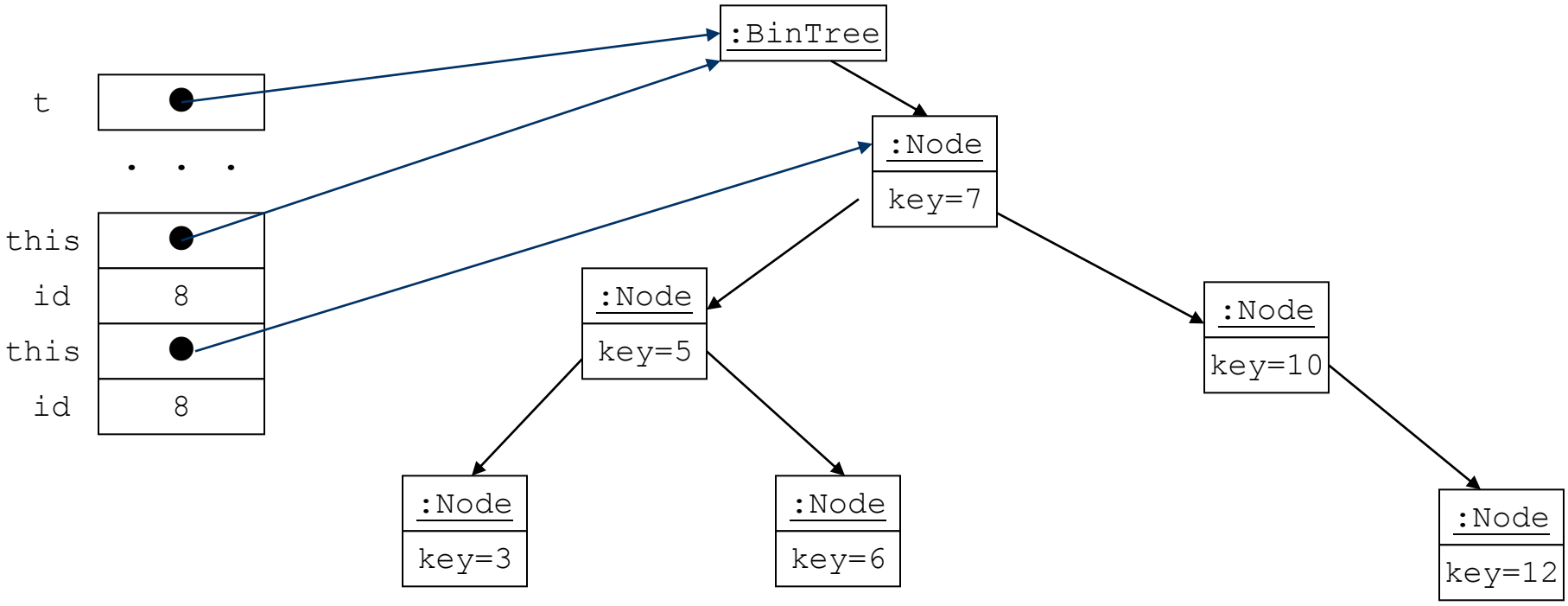
Aufruf von `t.insert(id)`:

Einfügen in geordneten Binärbaum (Implementierung)



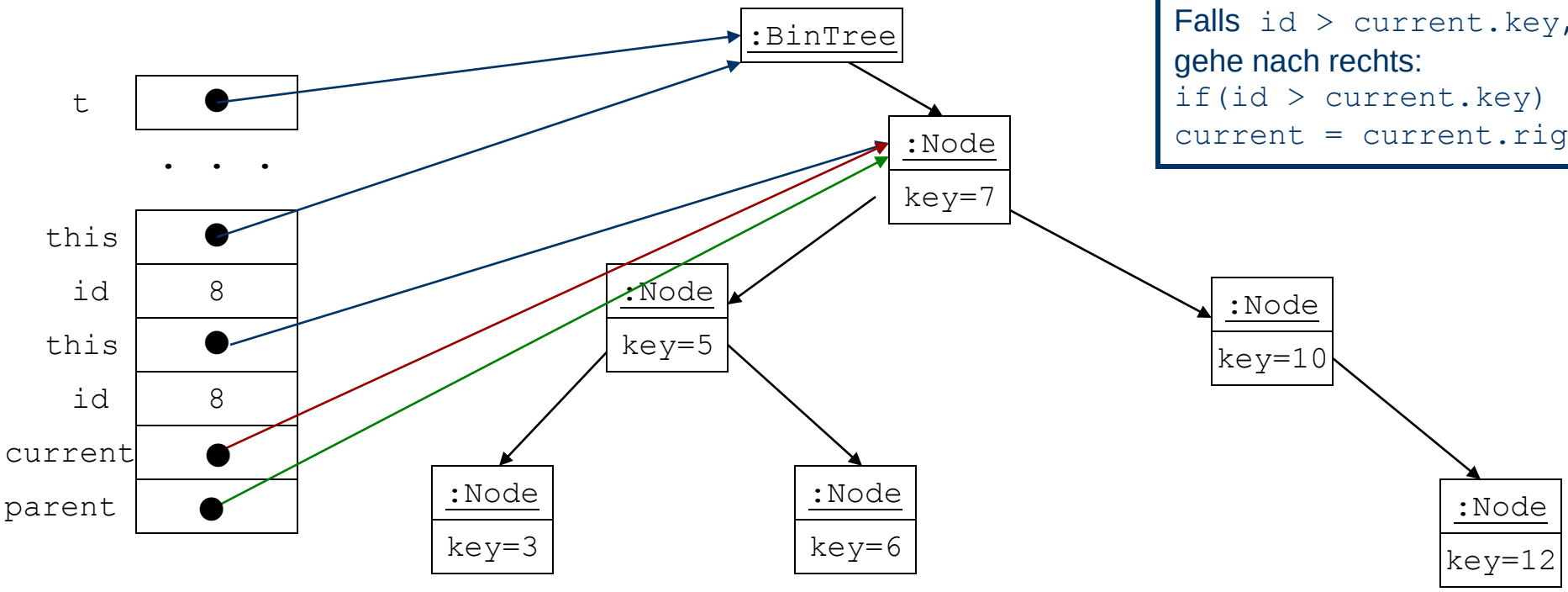
Delegieren der Aufgabe durch Aufruf von `root.insert(id)`:

root.insertKey(8):



Delegieren der Aufgabe durch Aufruf von `root.insert(id)`:
Durchlauf durch das Node-Geflecht mit zwei Hilfsvariablen `current` und `parent`

root.insert(8):

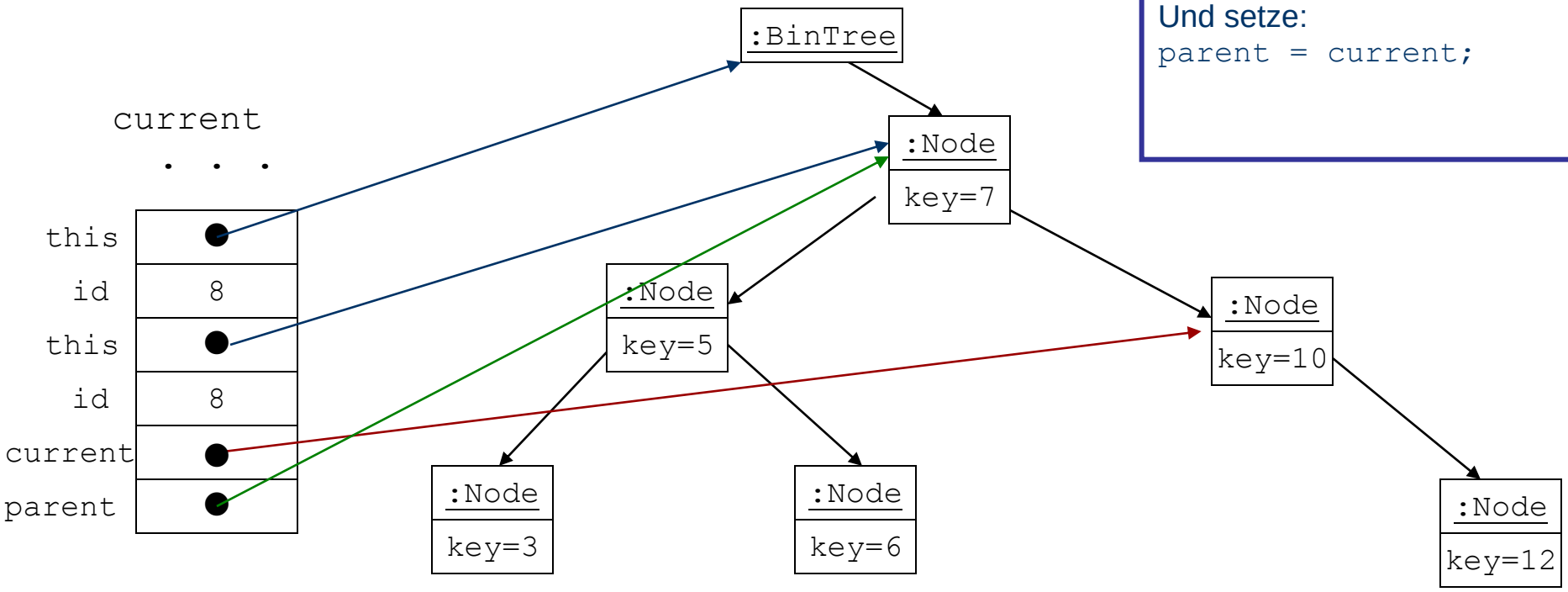


Falls `id > current.key`,
gehe nach rechts:

```
if(id > current.key)
current = current.right;
```

Delegieren der Aufgabe durch
Aufruf von `root.insert(id)`:
Durchlauf durch das Node-Geflecht mit
zwei Hilfsvariablen `current` und `parent`

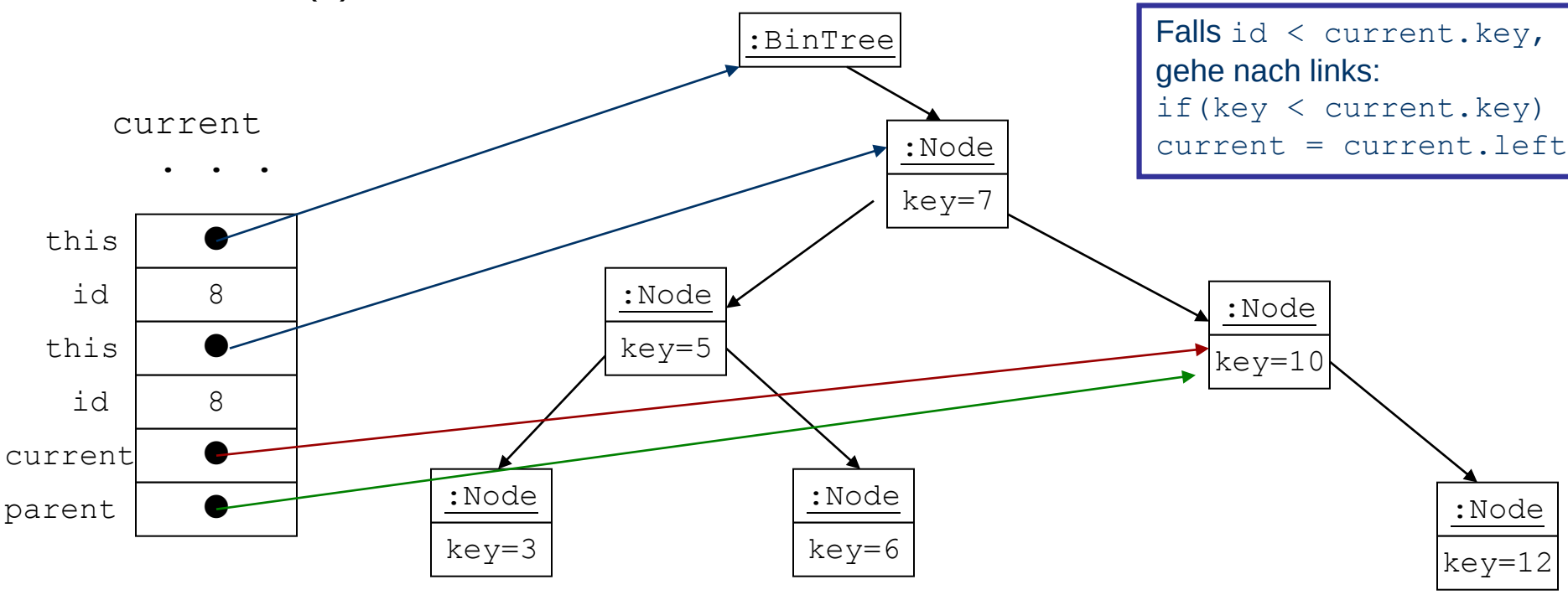
root.insert(8):



Und setze:

```
parent = current;
```

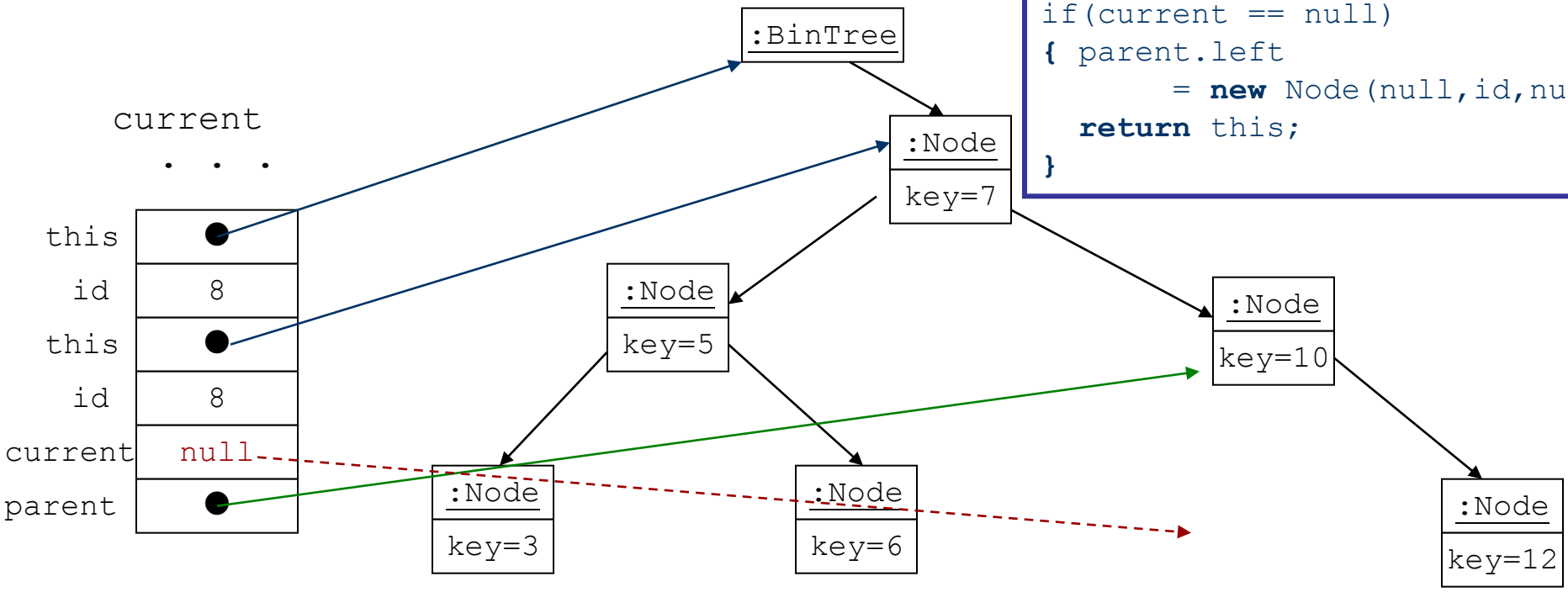
root.insert(8):



Falls `id < current.key`,
gehe nach links:

```
if(key < current.key)
current = current.left;
```

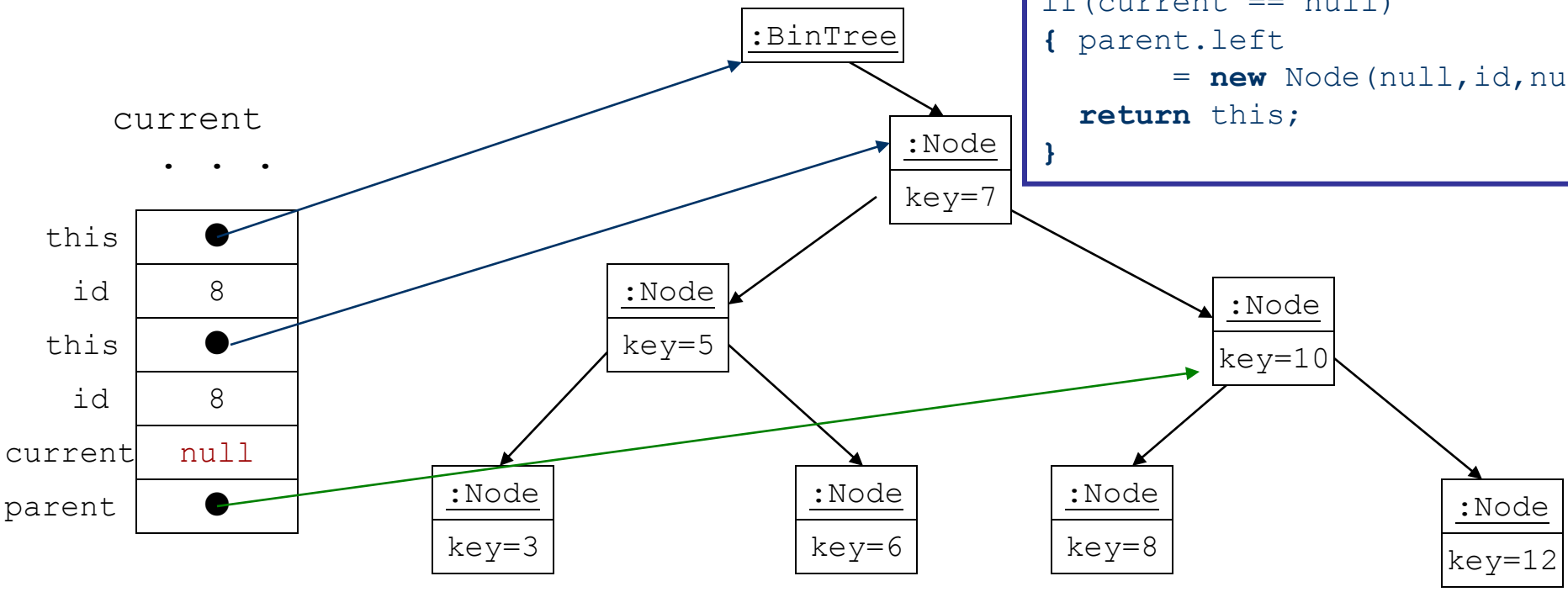
root.insert(8):



Wenn `current= null`, füge neuen Knoten ein:

```
if(current == null)
{ parent.left
        = new Node(null,id,null);
  return this;
}
```

root.insert(8):



Wenn `current= null`, füge neuen Knoten ein:

```
if(current == null)
{ parent.left
        = new Node(null,id,null);
  return this;
}
```

Einfügen in geordneten Binärbaum (Implementierung)

Fügt einen neuen Knoten mit Schlüssel `id` an der richtigen Stelle im geordneten Baum ein

```
public void insert(int id, Object o)
{   if(root==null)                    // falls kein Knoten im root
       root = new Node(null, id, o, null,);    // neuer Knoten
    else  root = root.insert(id, o);
}
```

wobei `insert` in **`class`** `Node` folgendermaßen definiert wird:

```
Node insert(int id, Object o)
{    Node current = this;        // starte bei this
     Node parent;
     while(true)                 // terminiert intern
     {   parent = current;
         if(id < current.key)    // gehe nach links?
         { current = current.left;
           if(current == null)  // am Ende füge links ein
           {   parent.left = new Node(null, id, o, null);
               return this;
           }
         }  // end if go left
         else                    // falls id > current.key, gehe nach rechts
         {  current = current.right;
            if(current == null)  // am Ende füge rechts ein
            {   parent.right = new Node(null, id, o, null);
                return this;
            }
         }  // end else go right
     }  // end while
}
```

Fügt einen neuen Knoten passend ein
Achtung: `id` darf nicht im Baum vorkommen!

Zusammenfassung

- Binäre Bäume können in Java implementiert werden als Verallgemeinerung der einfach verketteten Listen mit zwei Nachfolgerverweisen
- Eine Operation auf binären Bäume mit Knoten (z.B. `find`, `insert`) wird dann definiert durch Delegation der Operation an die Knotenklasse
- Im gezeigten Beispiel gibt die Klasse `BinTree` die Verantwortung für einen Teil ihrer Funktionalität an die Hilfsklasse `Node` ab
- Als weiteres Beispiel kann eine Methode `print` definiert werden, welche Informationen über den Knotenwert in der Konsole ausgibt