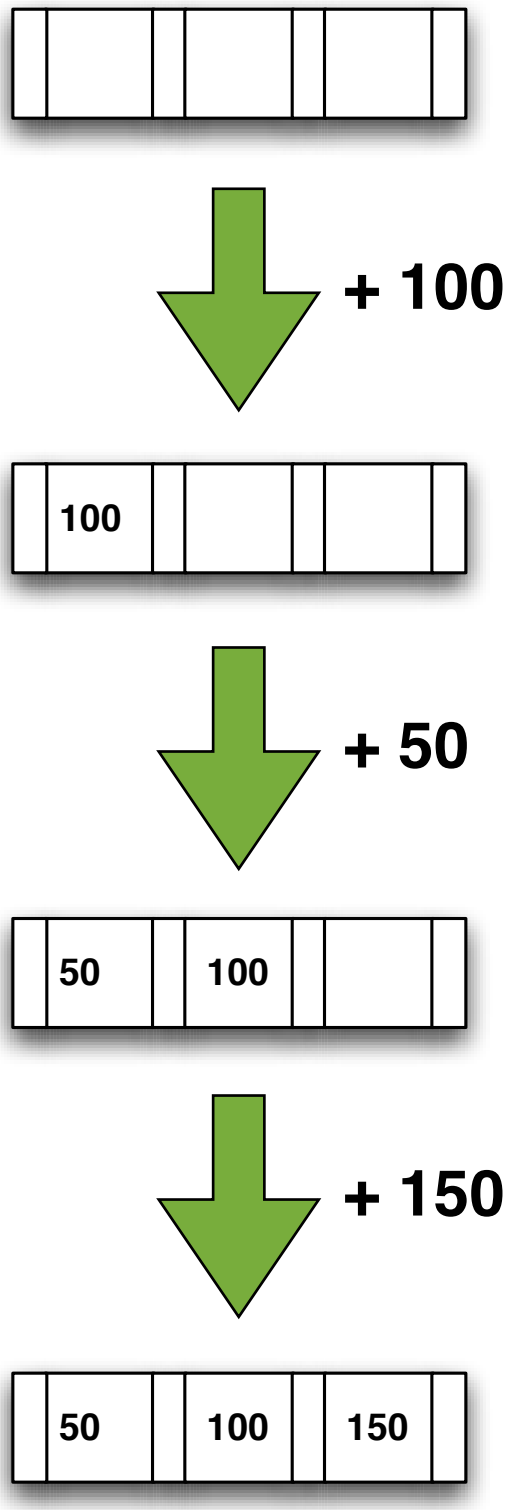
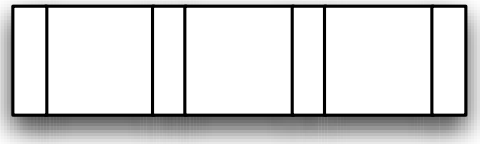
Aufbau eines "B-Baums"
der Ordnung 3, Teil 1

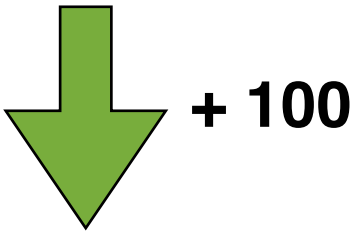


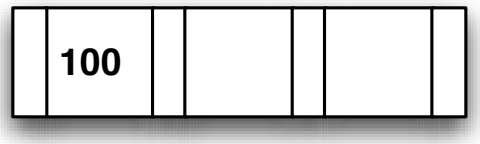
Leerer B-Baum der Ordnung 3.
Insgesamt Platz für 3 Werte.

Aufbau eines "B-Baums"
der Ordnung 3, Teil 1

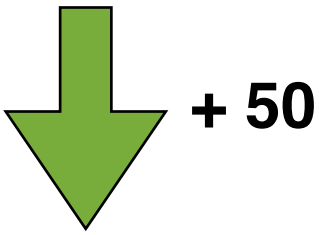


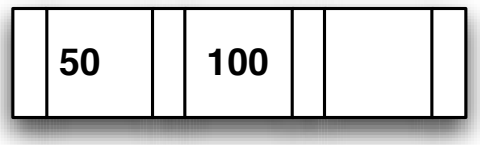
Leerer B-Baum der Ordnung 3.
Insgesamt Platz für 3 Werte.



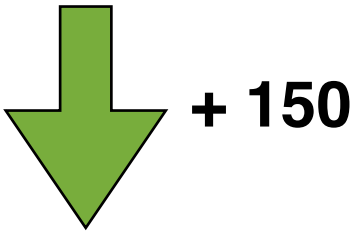


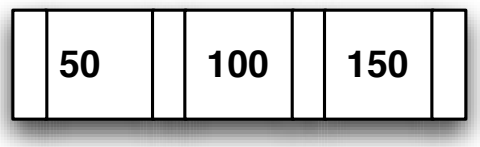
B-Baum nach Einfügen der 100.
Noch Platz für 2 weitere Werte.





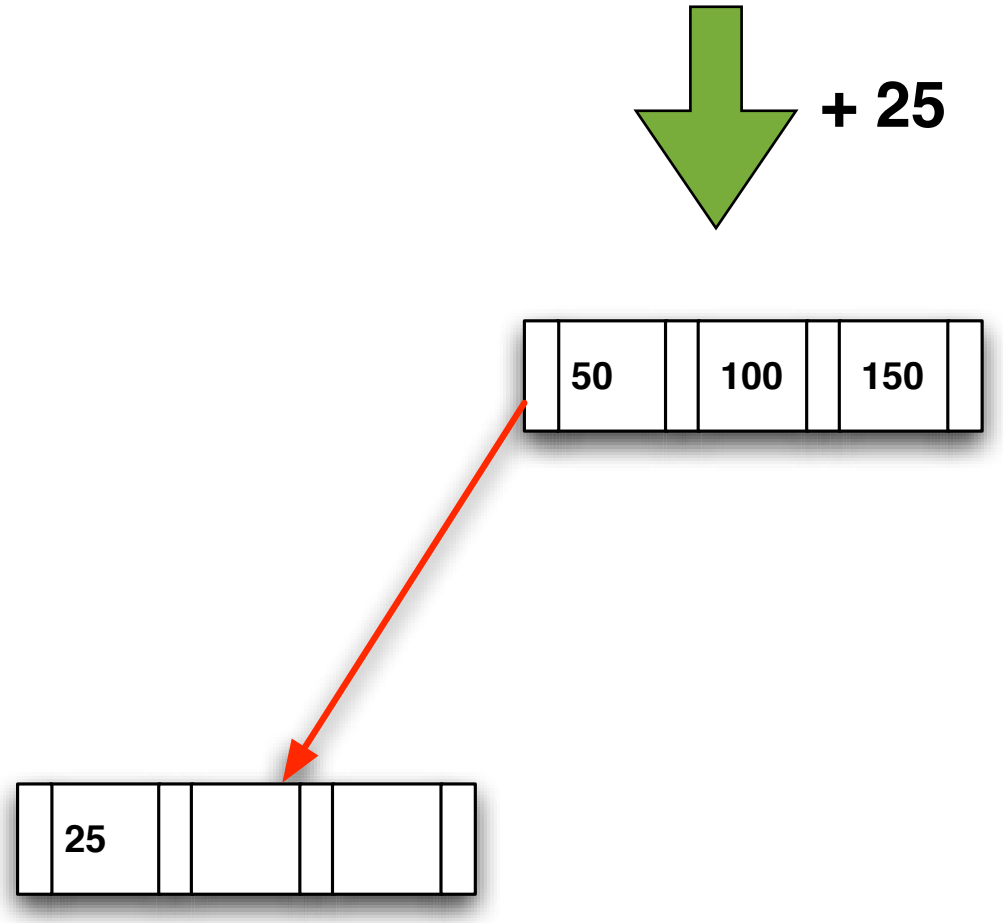
B-Baum nach Einfügen der 50.
Noch Platz für 1 weiteren Wert.



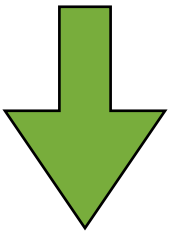


B-Baum nach Einfügen der 150.
Der Knoten ist voll.

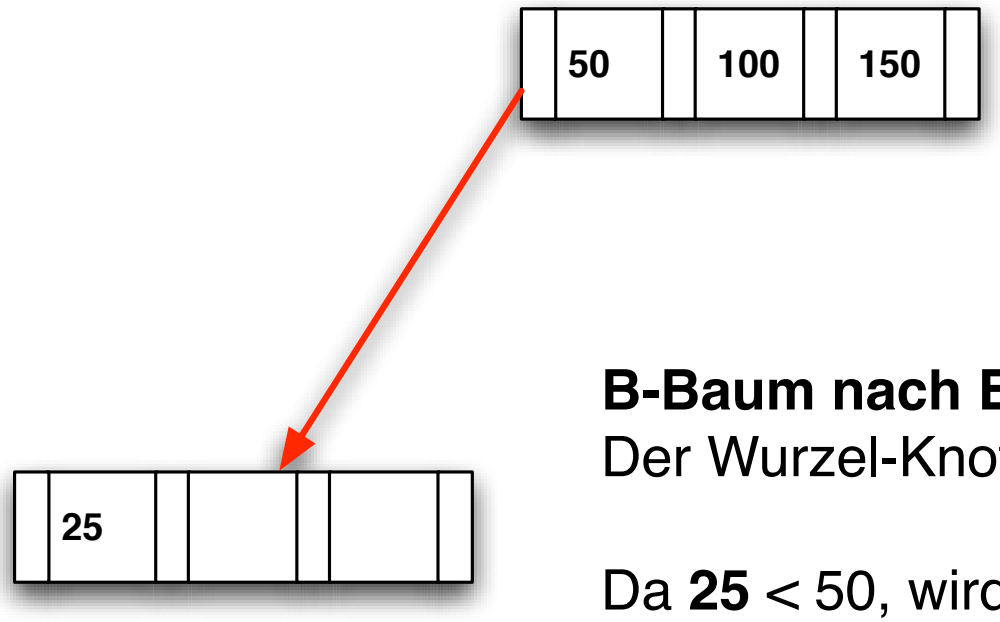
Aufbau eines "B-Baums"
der Ordnung 3, Teil 2



Aufbau eines "B-Baums"
der Ordnung 3, Teil 2



+ 25

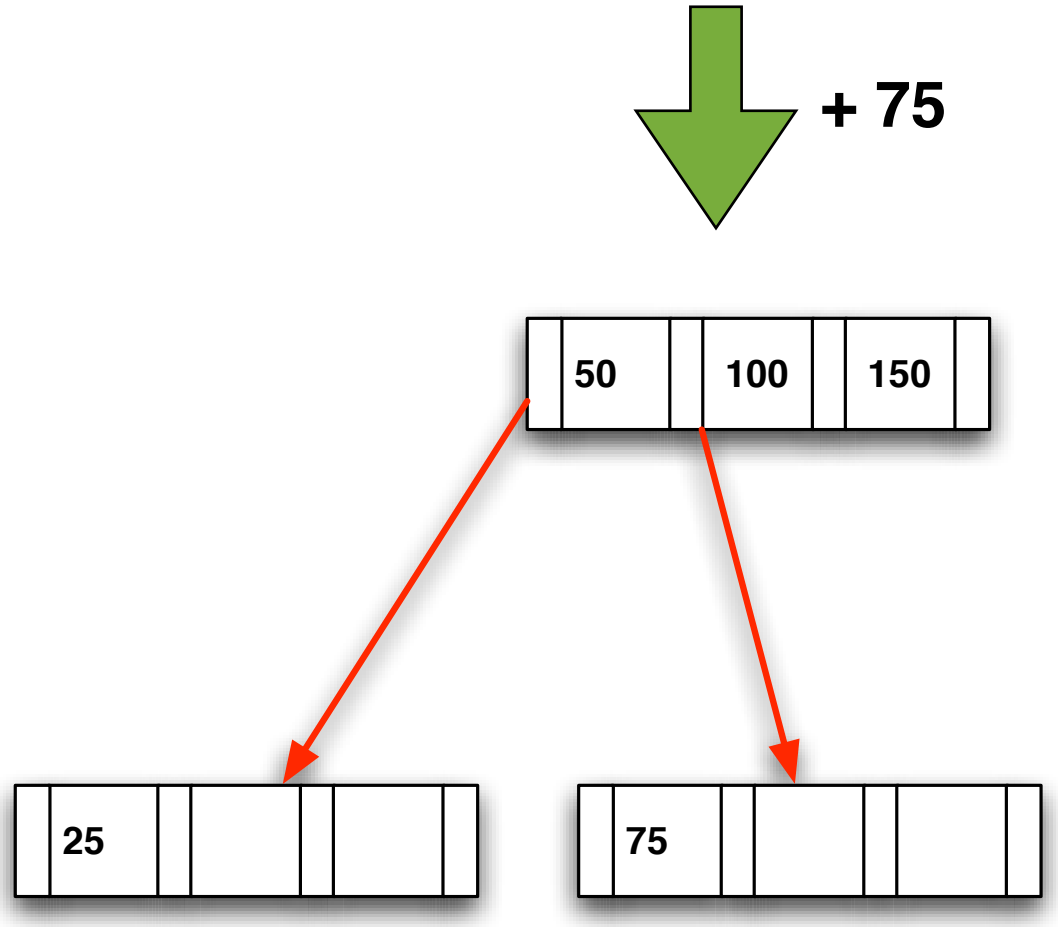


B-Baum nach Einfügen der 25.
Der Wurzel-Knoten ist voll besetzt.

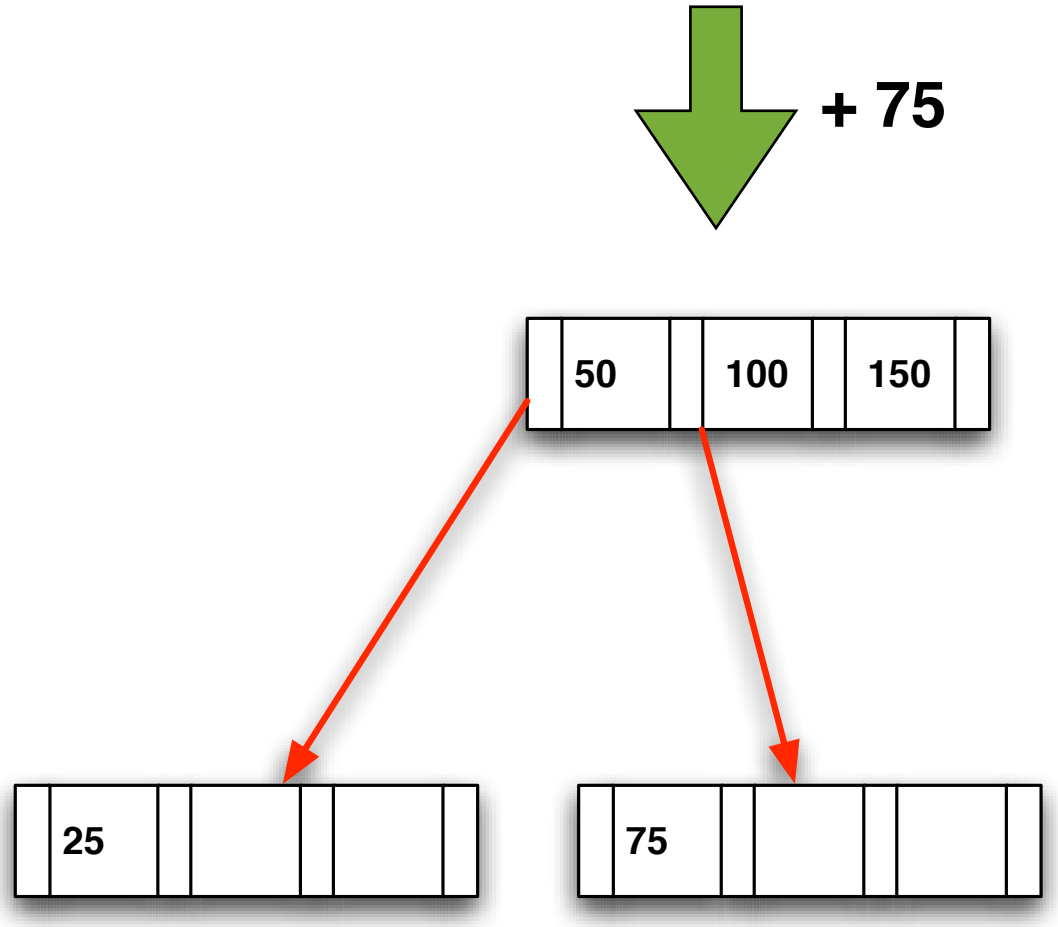
Da **25** < 50, wird links von der 50 weitergesucht.

Dort ist noch kein Knoten,
also wird ein neuer Knoten
erzeugt und mit der 25 belegt.

Aufbau eines "B-Baums"
der Ordnung 3, Teil 3



Aufbau eines "B-Baums"
der Ordnung 3, Teil 3

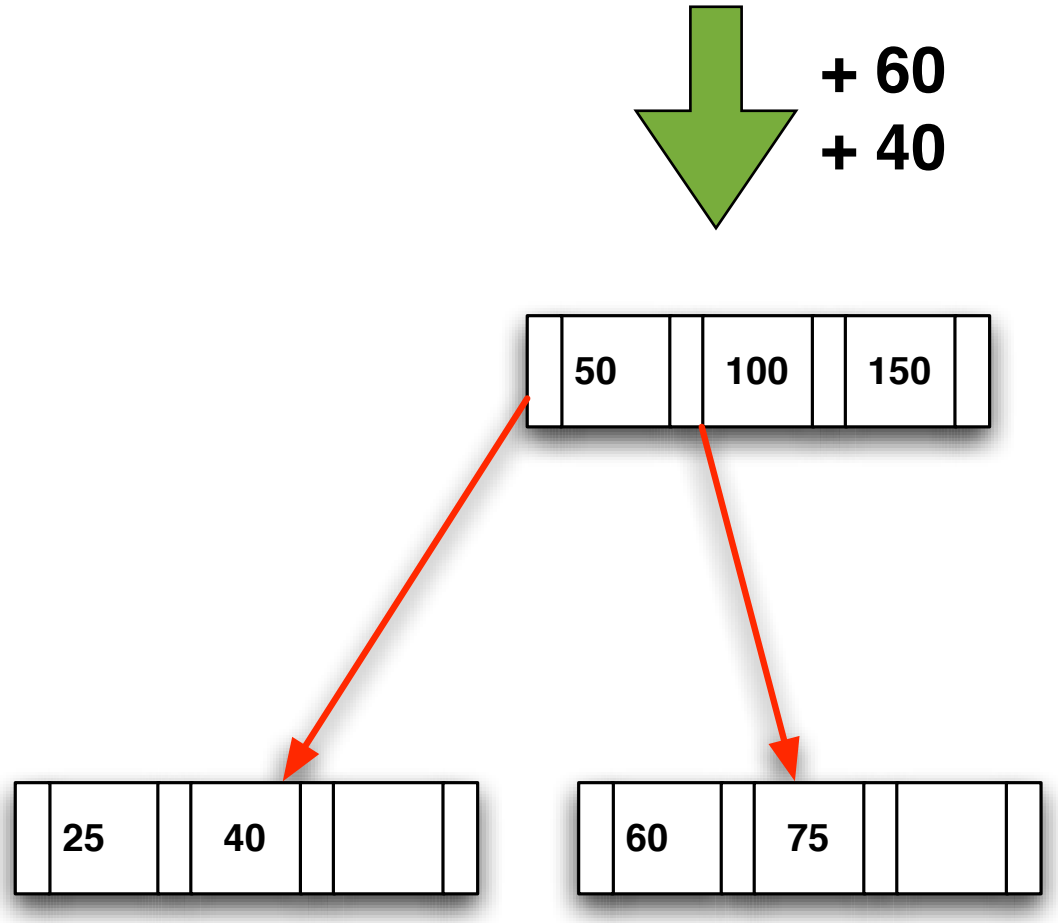


B-Baum nach Einfügen der 75.
Der Wurzel-Knoten ist voll besetzt.

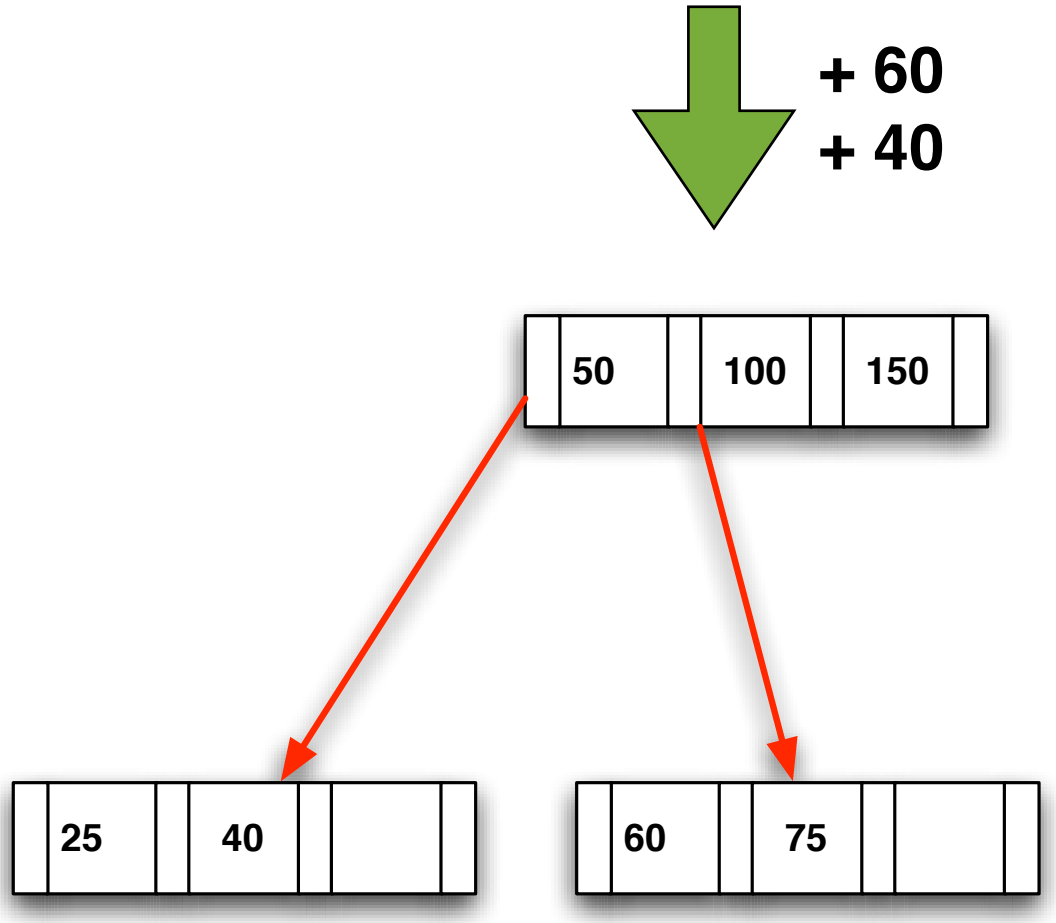
Da 50 < **75** < 100, wird rechts von der 50 weitergesucht.

Dort ist noch kein Knoten, also wird ein neuer Knoten erzeugt und mit der 75 belegt.

Aufbau eines "B-Baums"
der Ordnung 3, Teil 4



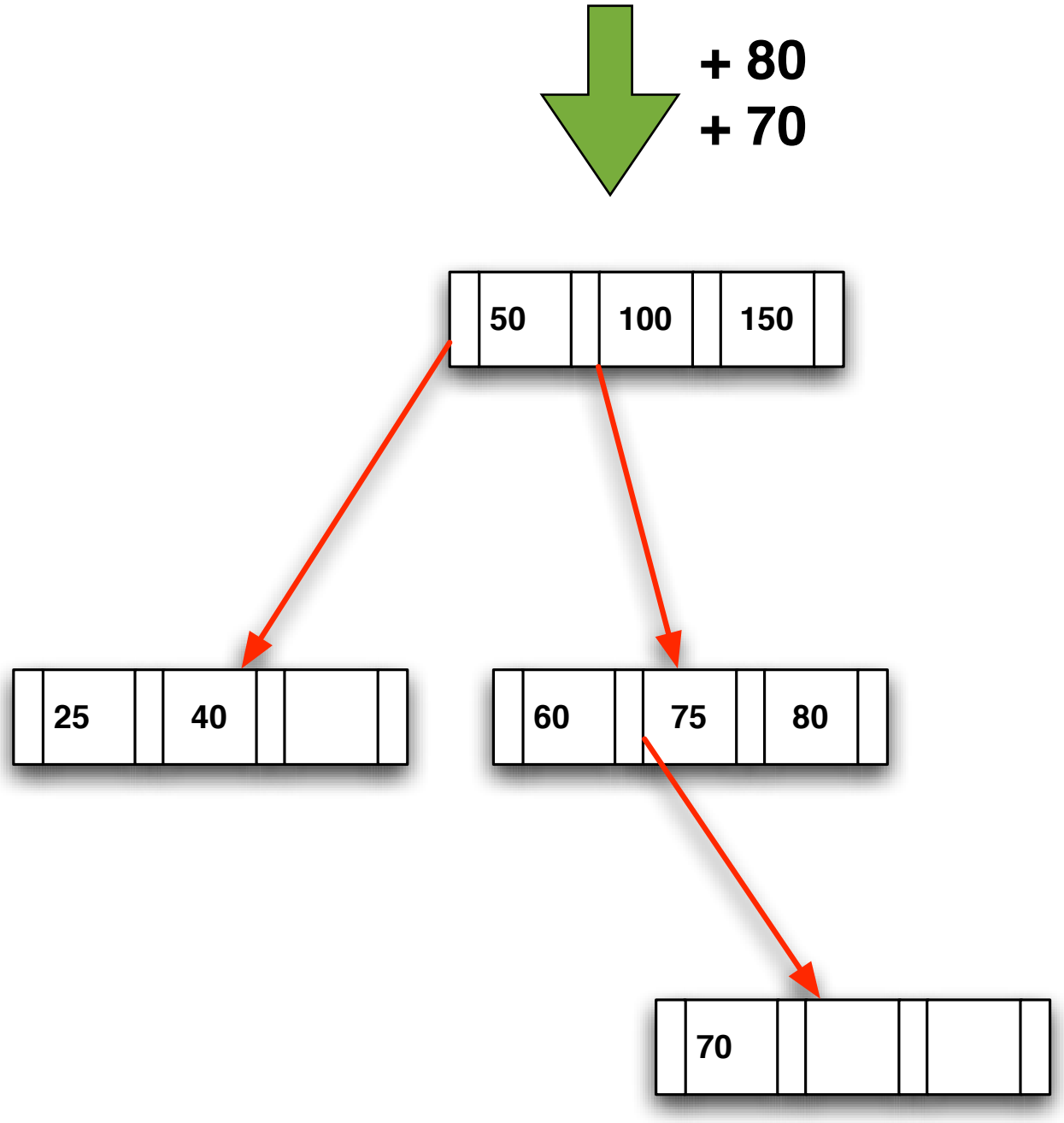
Aufbau eines "B-Baums"
der Ordnung 3, Teil 4



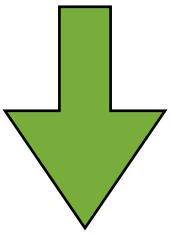
B-Baum nach Einfügen der 60 und der 40.
Da 50 < **60** < 100, wird rechts von der 50 weitergesucht. Dort ist bereits ein Knoten, also wird die 60 in den Tochterknoten eingefügt.

Da **40** < 50, wird links von der 50 weitergesucht. Dort ist bereits ein Knoten, also wird die 40 in den Tochterknoten eingefügt.

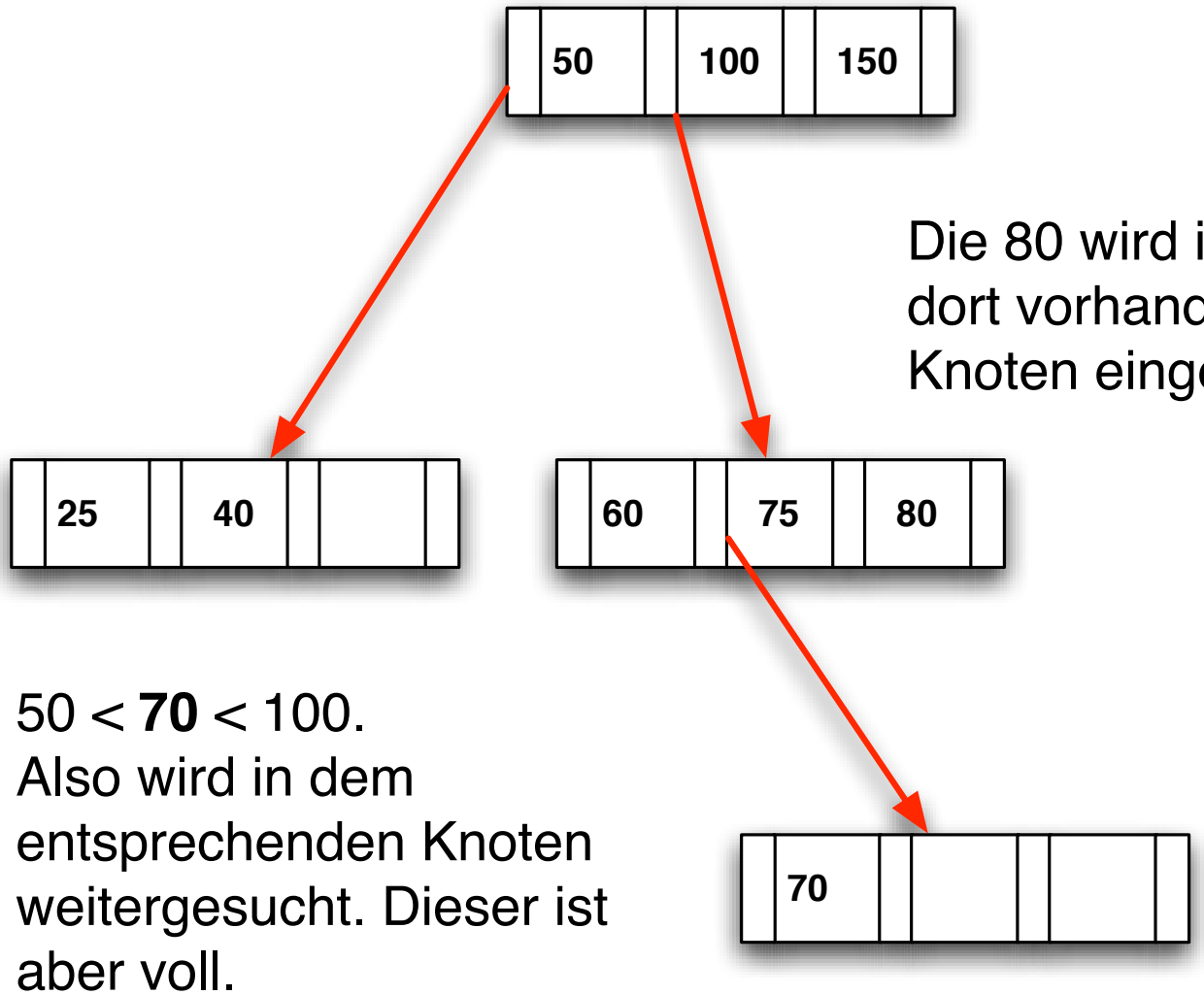
Aufbau eines "B-Baums"
der Ordnung 3, Teil 5



Aufbau eines "B-Baums"
der Ordnung 3, Teil 5



+ 80
+ 70

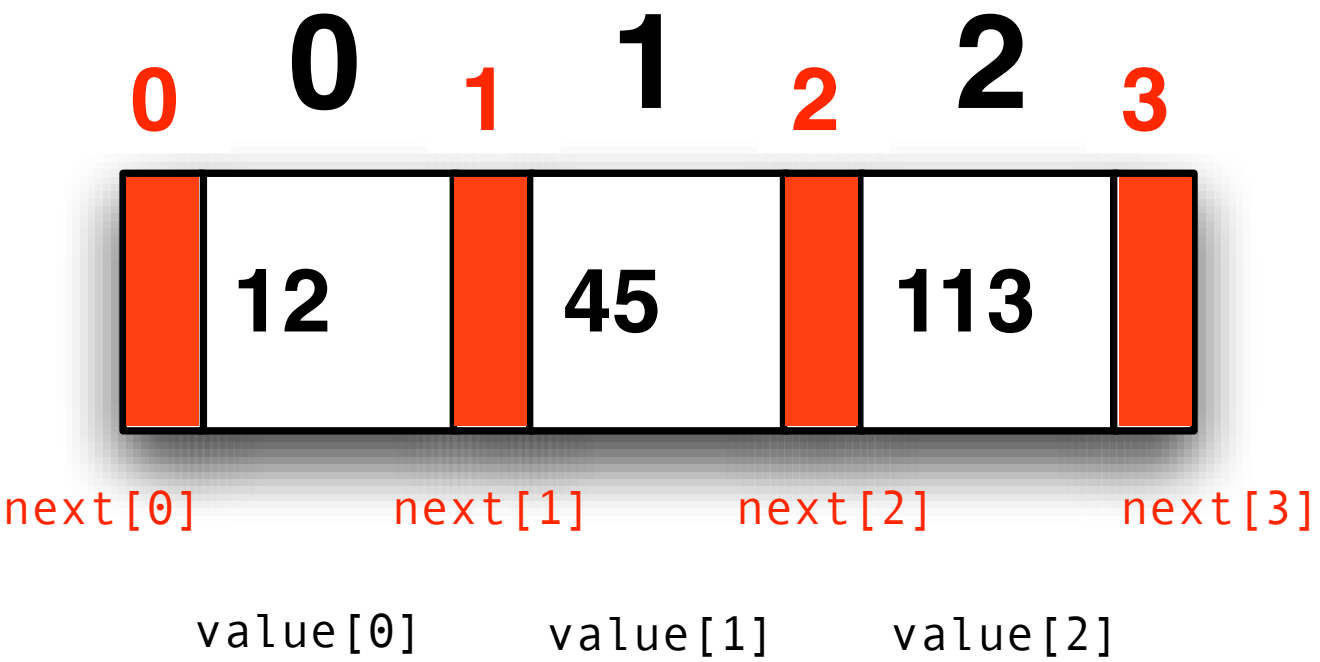


Die 80 wird in den dort vorhandenen Knoten eingefügt.

50 < **70** < 100.
Also wird in dem entsprechenden Knoten weitergesucht. Dieser ist aber voll.

60 < **70** < 75.
Der next-Zeiger zeigt auf null, also wird ein neuer Knoten angelegt.

Implementation eines B-Baum-Elementes, Teil 1

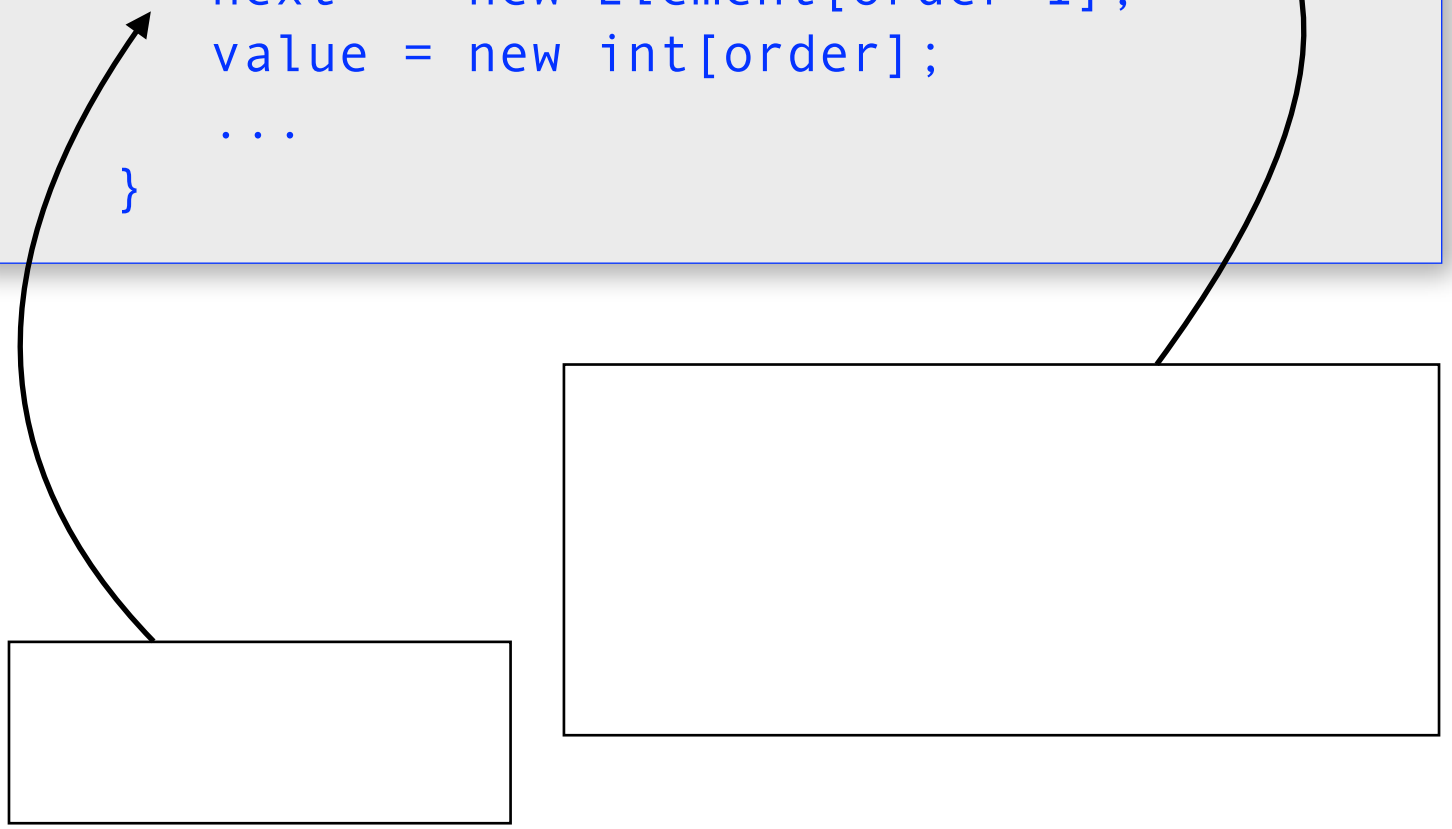



```
public class Element
{
   private Element[] next; // Länge 4
   private int[] value;    // Länge 3
   private int count;
   public int order;
   ...
```

Implementation eines B-Baum-Elementes, Teil 2

```
public class Element
{
   ...

   public Element(int n, int x)
   {
      order = n;
      next  = new Element[order+1];
      value = new int[order];
      ...
   }
```



Implementation eines B-Baum-Elementes, Teil 2

```
public class Element
{
   ...

   public Element(int n, int x)
   {
      order = n;
      next  = new Element[order+1];
      value = new int[order];
      ...
   }
```

Initialisierung der beiden Arrays.

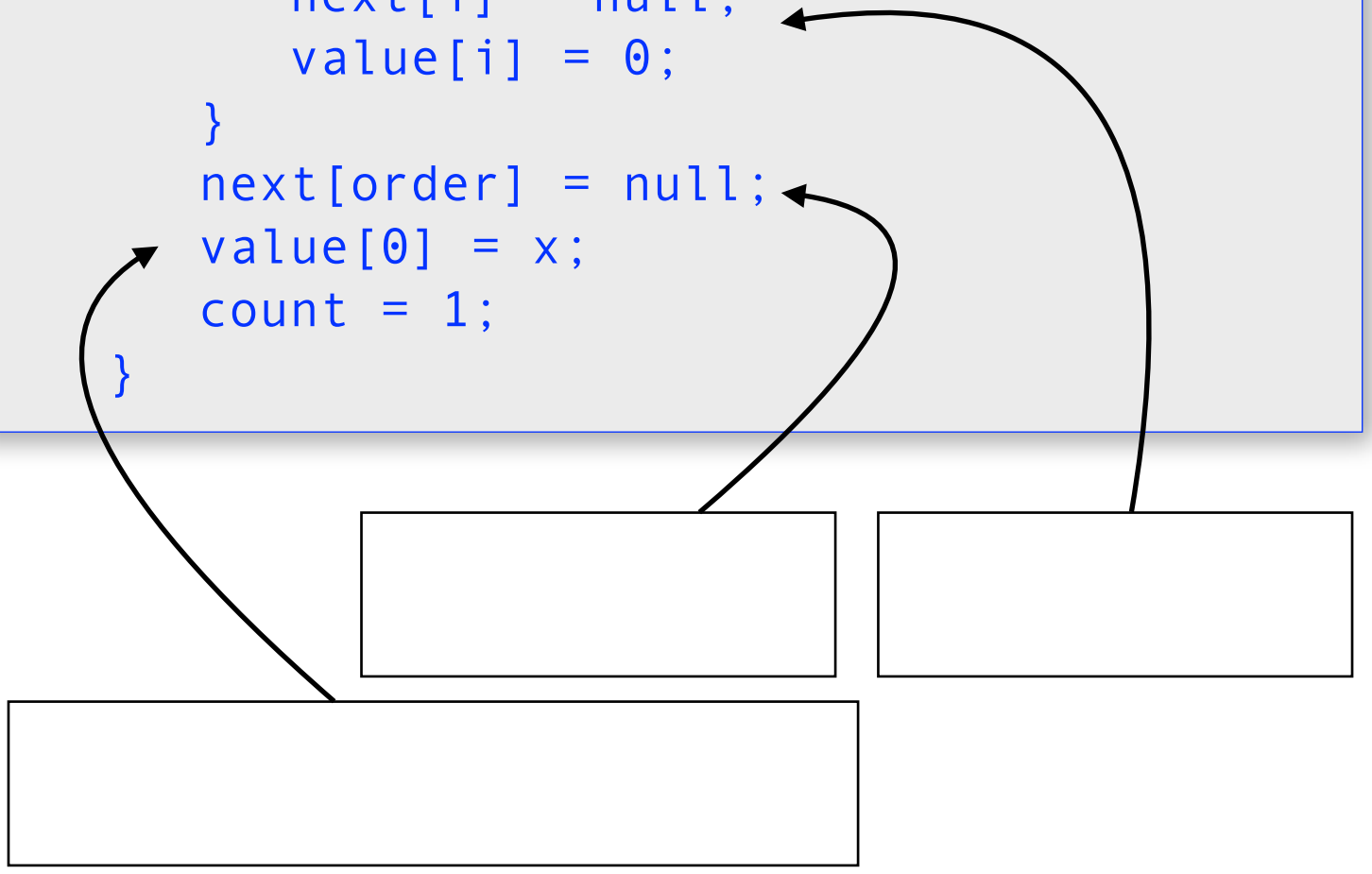
Parameter **x** = der erste Wert, der gespeichert werden soll.

Parameter **n** = die Ordnung des Baumes.

Implementation eines B-Baum-Elementes, Teil 3

```
public class Element
{
    ...

    public Element(int n, int x)
    {
        ...
        for (int i=0; i<order; i++)
        {
            next[i] = null;
            value[i] = 0;
        }
        next[order] = null;
        value[0] = x;
        count = 1;
    }
```



Implementation eines B-Baum-Elementes, Teil 3

```
public class Element
{
    ...

    public Element(int n, int x)
    {
        ...
        for (int i=0; i<order; i++)
        {
            next[i] = null;
            value[i] = 0;
        }
        next[order] = null;
        value[0] = x;
        count = 1;
    }
```

Letzten Zeiger initialisieren.

Beide Arrays mit Nullwerten füllen.

Ersten Wert in den Knoten schreiben, Zähler auf 1 setzen.

Einfügen in einen Knoten, Teil 1

Fall 1:

Der Knoten hat noch freie Plätze (count < order):

1a) Füge neuen Wert hinten an

1b) Sortiere Knotenwerte

Implementation von Fall 1, Schritte a und b:

```
if (count < order)
{
   value[count++] = x;
   sort();
}
```

Einfügen in einen Knoten, Teil 2

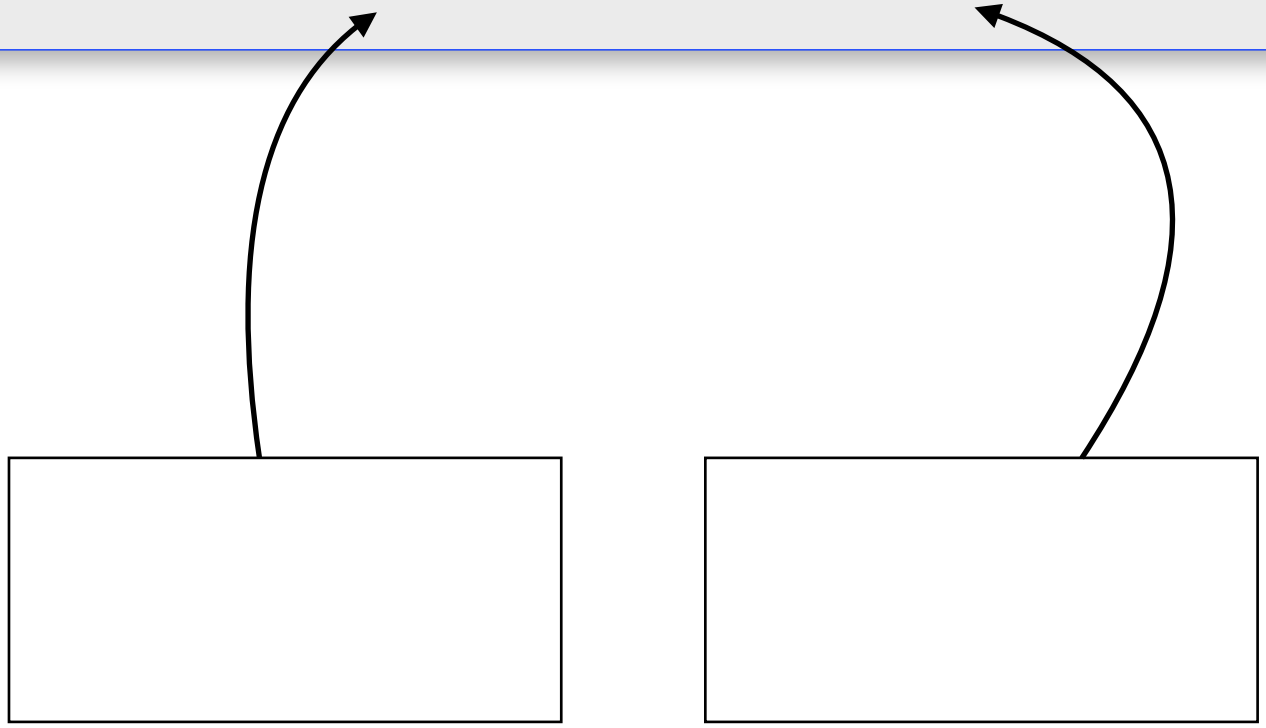
Fall 2:

Der Knoten ist voll (count == order):

2a) Suche passende Stelle zum Einfügen

Implementation von Fall 2a) :

```
int i=0;
while ((i<count) && (x>value[i])) i++;
```



Einfügen in einen Knoten, Teil 2

Fall 2:

Der Knoten ist voll (count == order):

2a) Suche passende Stelle zum Einfügen

Implementation von Fall 2a) :

```
int i=0;
while ((i<count) && (x>value[i])) i++;
```

Aufpassen, dass der Array beim Suchen nicht verlassen wird.

Suche nach der passenden Stelle zum Einfügen.

Einfügen in einen Knoten, Teil 3

Fall 2:

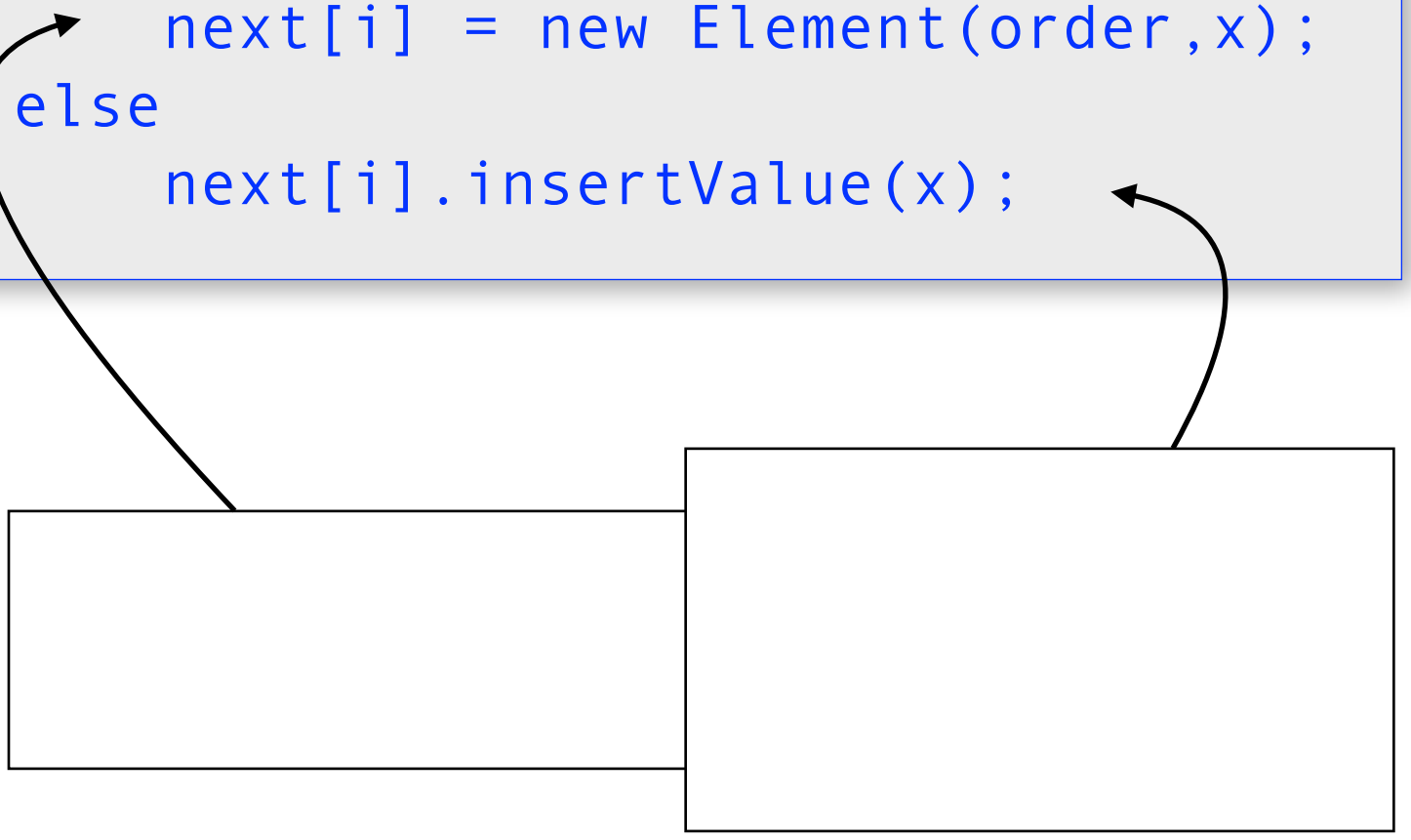
Der Knoten ist voll (count == order):

2a) Suche passende Stelle zum Einfügen

2b) Knoten erzeugen / in vorhandenen Knoten einfügen.

Implementation von Fall 2b) :

```
if (next[i] == null)
    next[i] = new Element(order,x);
else
    next[i].insertValue(x);
```



Einfügen in einen Knoten, Teil 3

Fall 2:

Der Knoten ist voll (count == order):

2a) Suche passende Stelle zum Einfügen

2b) Knoten erzeugen / in vorhandenen Knoten einfügen.

Implementation von Fall 2b) :

```
if (next[i] == null)
    next[i] = new Element(order,x);
else
    next[i].insertValue(x);
```

Ein Nachfolgerknoten ist nicht vorhanden. Also wird ein neuer Knoten erzeugt.

Ein Nachfolgerknoten ist vorhanden. Also wird der Einfüge-Algorithmus für diesen Knoten aufgerufen (im Prinzip eine Rekursion!)

Einfügen in einen Knoten, Teil 4

Zusammenfassung: Element.insertValue()

```
public void insertValue(int x)
{
   if (count < order)
   {
      value[count++] = x;
      sort();
   }
   else
   {
      int i=0;
      while ((i<count) && (x>value[i]))
         i++;

      if (next[i] == null)
         next[i] = new Element(order,x);
      else
         next[i].insertValue(x);
   }
}
```

Anzeigen eines Knotens

```
public void show(int level)
{
   for (int i=0; i<=count; i++)
   {
      if (next[i] != null)
         next[i].show(level+1);
      if (i<count)
         System.out.println
             (level+" / "+value[i]);
   }
}
```

Ein einfacher inorder-Mechanismus, der auch das Niveau des jeweiligen Knotens ausgibt (level).