Informatik II

Vorlesung am D-BAUG der ETH Zürich

Vorlesung 6, 4.4.2016

Dynamische Datenstrukturen: verkettete Listen, Stapel

Stepwise Refinement: Programmiertechnik zum Lösen komplexer Probleme

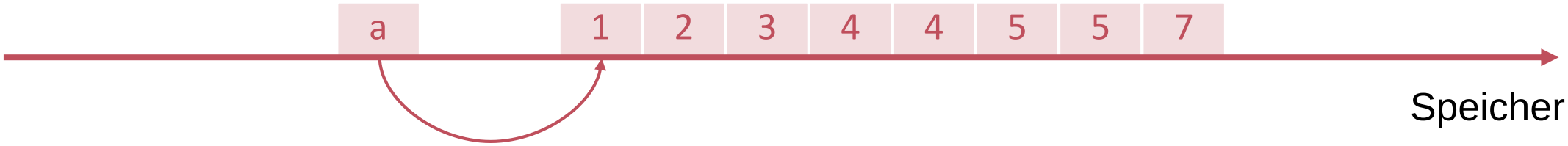
Verkettete Listen, Stapel

DYNAMISCHE DATENSTRUKTUREN

Container für eine Folge gleichartiger Daten

Arrays: zusammenhängender Speicherbereich, wahlfreier Zugriff (auf i-tes Element)

```
int a[] = new int[8];
```



Einmal alloziert ist die Länge fixiert

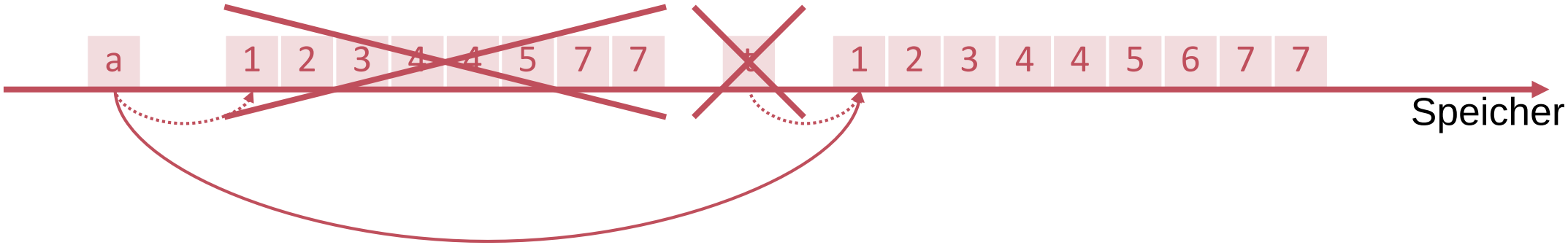
Probleme mit Arrays

Einfügen oder Löschen von Elementen "in der Mitte" ist aufwändig

Wollen wir hier einfügen, so müssen wir alles rechts davon explizit verschieben und ggfs. vorher Platz schaffen / kopieren

Probleme mit Arrays

Beispiel: Einfügen unter Neuallokation des Array.



Probleme mit Arrays

Einfügen oder **Löschen** von Elementen "in der Mitte" ist aufwändig

Probleme mit Arrays

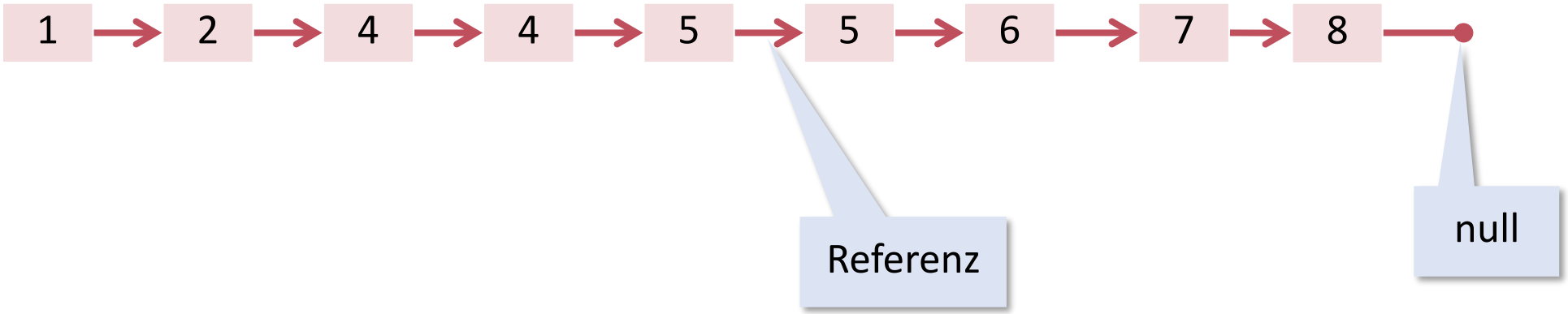
Einfügen oder **Löschen** von Elementen "in der Mitte" ist aufwändig

Verkettete Listen

Lösung: (Verkettete) Listen

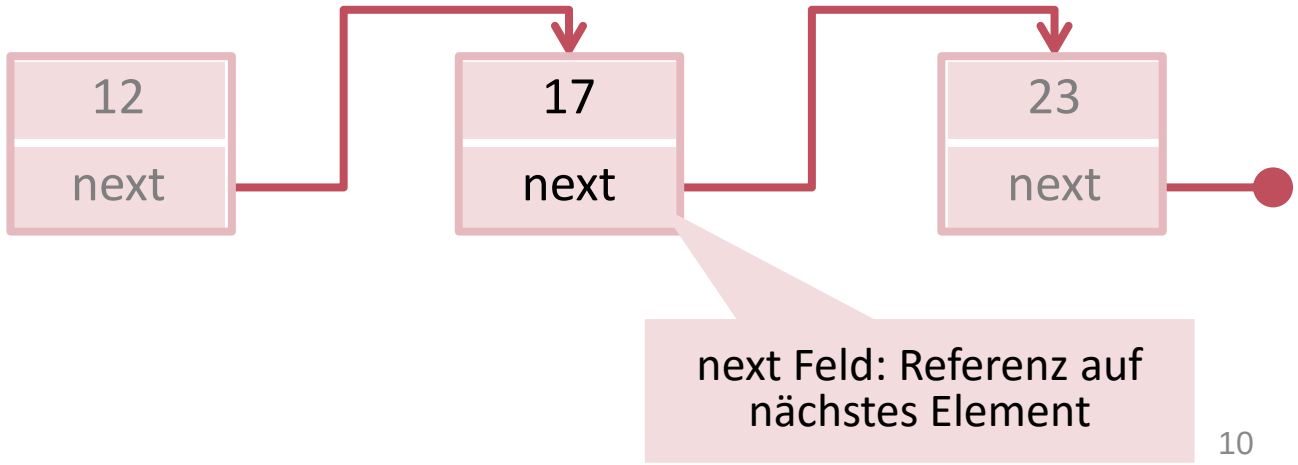
Container für eine Folge von Daten des gleichen Typs

Kein zusammenhängender Speicherbereich, kein wahlfreier Zugriff



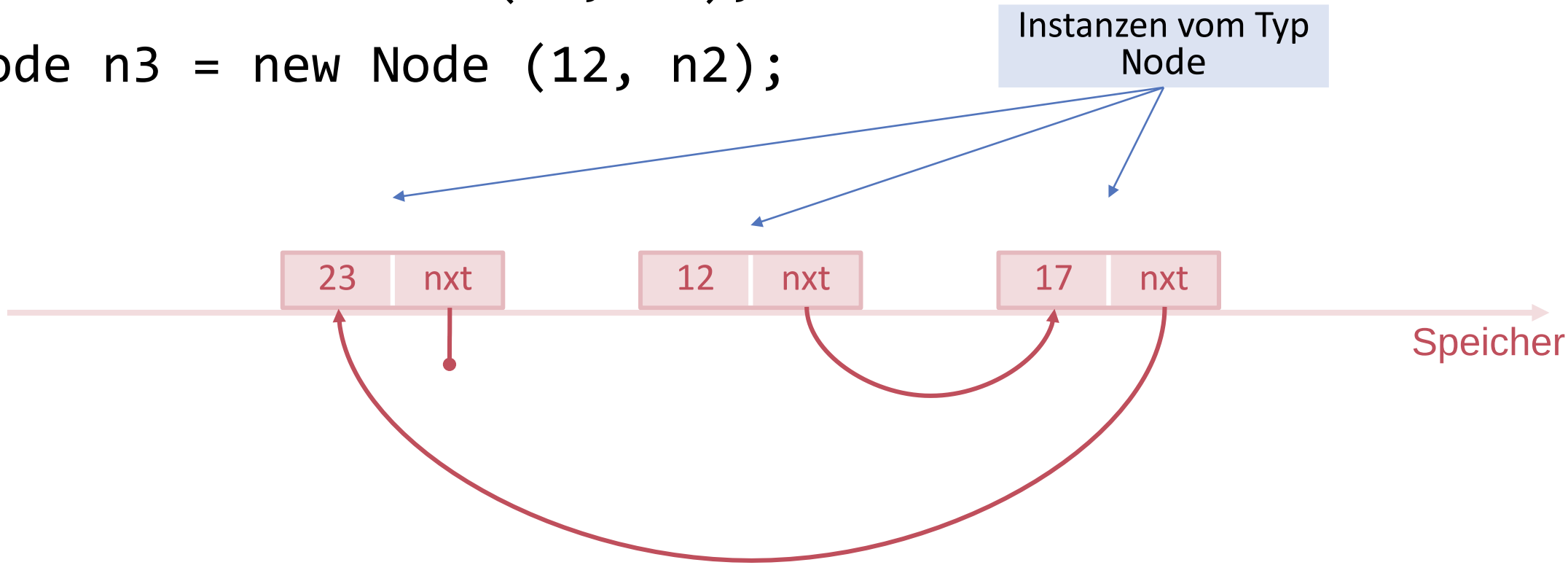
(Einfach) verkettete Liste

```
class Node
{
   double value;  // "Nutzlast" des Knoten
   Node next;     // Verkettung: nächster Knoten

   Node (double v, Node nxt) // Konstruktor
   {
      value= v;
      next = nxt;
   }
}
```

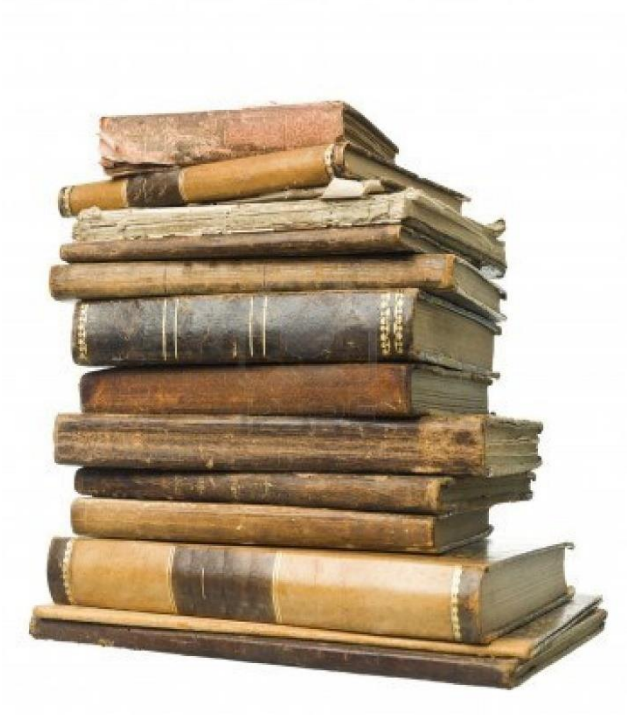


Im Speicher

```
Node n1 = new Node(23, null);
Node n2 = new Node(17, n1);
Node n3 = new Node (12, n2);
```



Stapel



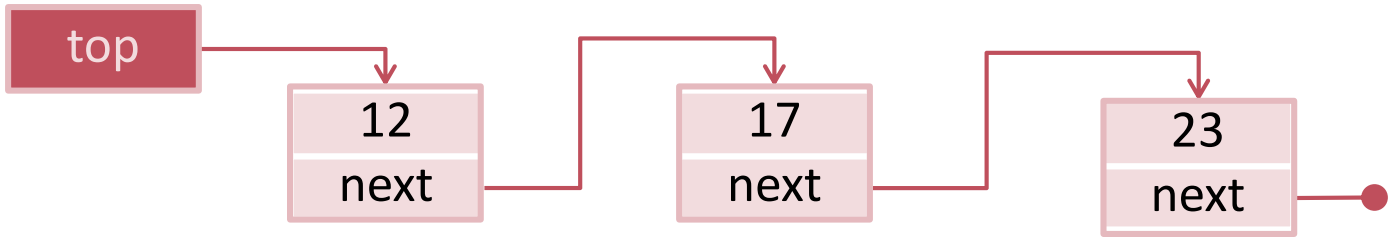
Stapel (LIFO – Last In First Out)

Der Stapel ist das einfachste Beispiel einer Datenstruktur, welche man gut mit einer verketteten Liste implementieren kann.

Funktionalität:

- Knoten vorne einfügen: **Push**
- Knoten vorne herausnehmen: **Pop**

Erstes Element durch "top" repräsentiert

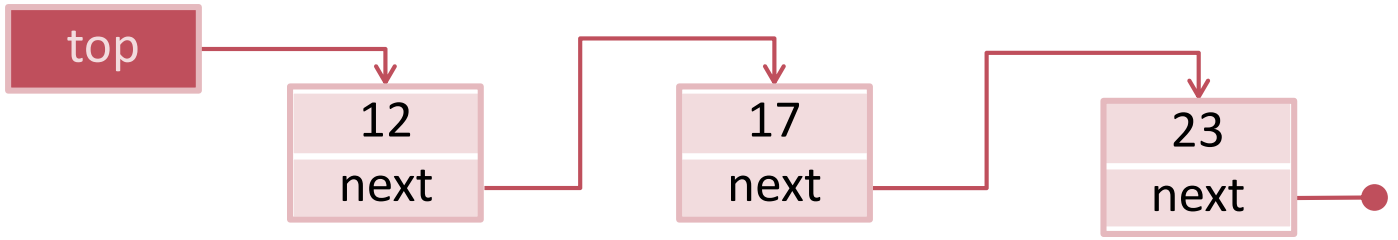


Stapel: Interface

```
public class Stack {

      // pre: beliebiger Fliesskommawert val
      // post: val liegt oben auf dem Stapel
      public void Push(double val)

      // pre: nichtleerer Stapel
      // post: oberstes Element t des Stapels entfernt
      //       Rückgabe des zu t gehörigen Wertes
      public double Pop()
}
```

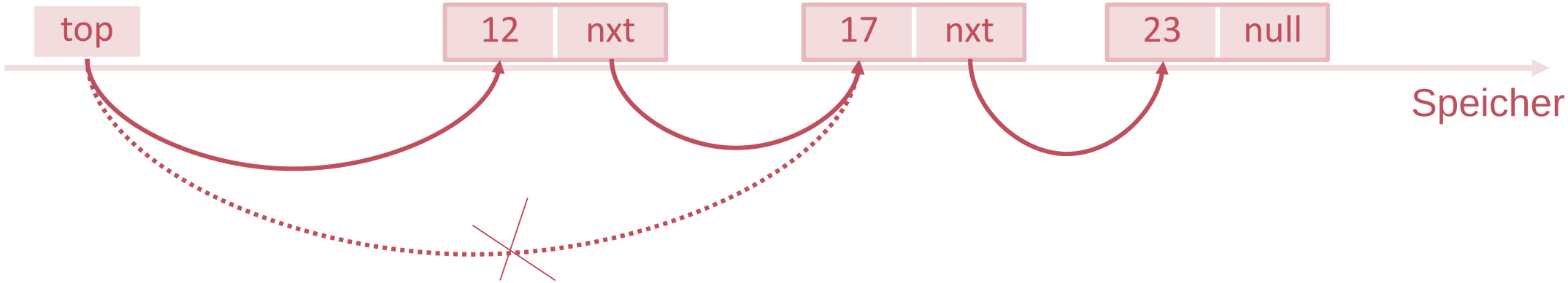
Stapel mit verketteter Liste

```
class Node {...} // wie oben


public class Stack {
      private Node top = null;
      public void Push(double val) { ... }
      public double Pop() { ... }
}
```



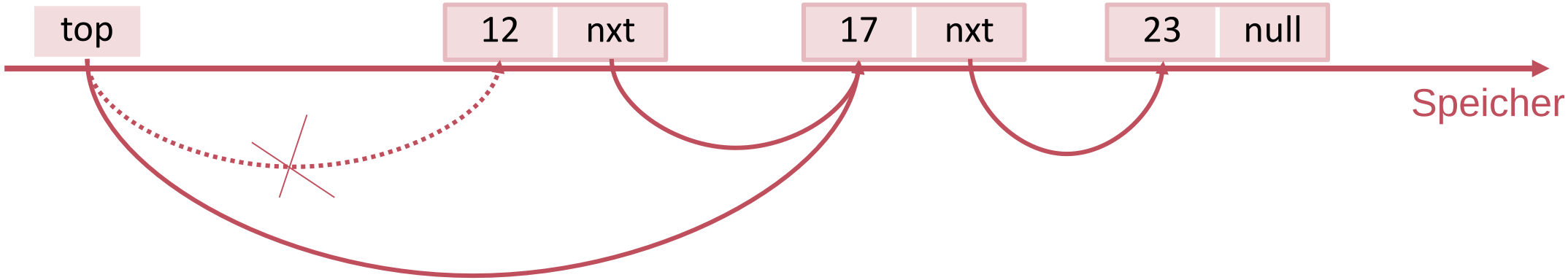
Push

```
public class Stack {
        ...
        public void Push(double val) {
                Node next = new Node(val, top);
                top = next;
        }
}
```



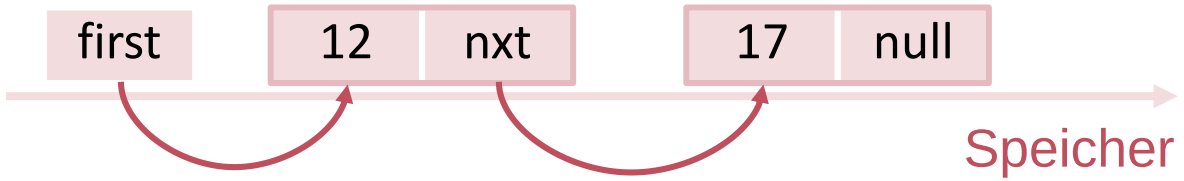
Pop

```
public class Stack {
        ...
        public double Pop() {
                double value = top.value;
                top = top.next;
                return value
        }
}
```



Liste traversieren

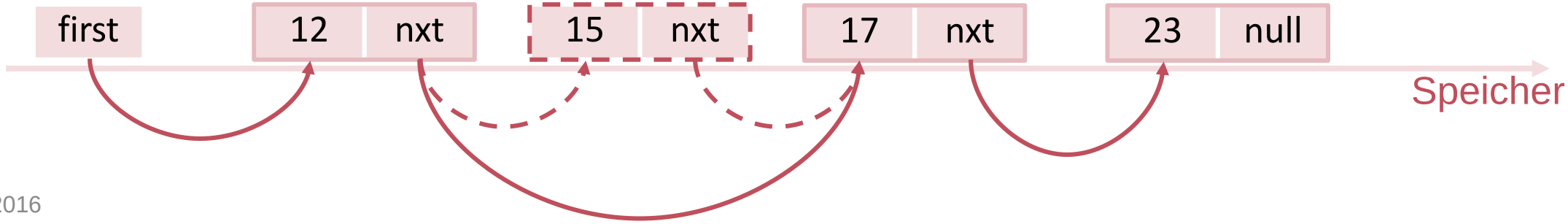
```
class Node {
   int value;
   Node next;

   Node (int v; Node n){
      value = v;
      next = n;
   }
}
```




```
void outputI(Node n) {
   while (n != null){
      System.out.println(n.value);
      n = n.next;
   }
}

void outputR(Node n) {
   if (n != null){
      outputR(n.next);
      System.out.println(n.value);
   }
}
```

Sortierte Liste

```
class SortedList {
      Node first = null;

      // pre: verkettete Liste, erstes Element first
      // post: value in Liste aufsteigend sortiert eingefügt
      public void Insert (int value){..?..}
}
```



Invarianten für Knoten in einer sortierten Liste

Für jeden Knoten n gilt entweder

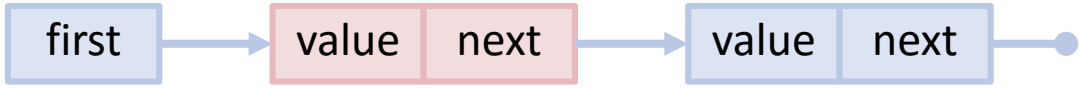
n == null

oder

n.next == null

oder

n.next != null
&& n.next.value >= n.value



Fälle : Einfügen von x

1. Fall: leere Liste

2. Fall: nichtleere Liste

a. v <= n.value für alle Knoten n
b. v > n.value für alle Knoten n
c. Es gibt einen Knoten n und Nachfolger m so dass
   x > n.value und x <= m.value

Stepwise Refinement

Einfache Programmiertechnik zum Lösen komplexer Probleme

Top-Down Ansatz

Program Development by Stepwise Refinement, *Niklaus Wirth*
Communications of the ACM, Vol. 14, No. 4, April 1971, pp. 221-227

Stepwise Refinement

Formulierung einer groben Lösung mit Hilfe von

- Kommentaren
- fiktiven Funktionen

Wiederholte Verfeinerung

- Kommentare → Programmtext
- Fiktive Funktionen → Funktionsdefinitionen

Beispiel: Einfügen in sortierte Liste

Live Coding in der Vorlesung.

Auf separaten Slides als Präsentation.

Ergebnis: Einfügen

```
public void Insert (int value){
    if (first == null || value <= first.value)
        first = new Node(value, first);
    else {
        Node prev = first;
        Node v = prev.next;
        while (v != null && v.value > value){ // suche Nachfolger und Vorgänger
            prev = v;
            v = v.next;
        }
        prev.next = new Node(value, v); // Einfügen
    }
}
```