Fallstudie: Online-Statistik

Ziel: Klasse / Objekt, welches Daten konsumiert und zu jeder Zeit Statistiken, z.B. Mittelwert, Varianz, Median (etc.) ausgeben kann

```
Statistics s = new Statistics(maxSize);
...

// Viele Datenpunkte einfügen
s.Put(data);


...
System.out.println(s.Mean());
System.out.println(s.Variance());
System.out.println(s.Median());
```

Minimal angestrebtes Interface

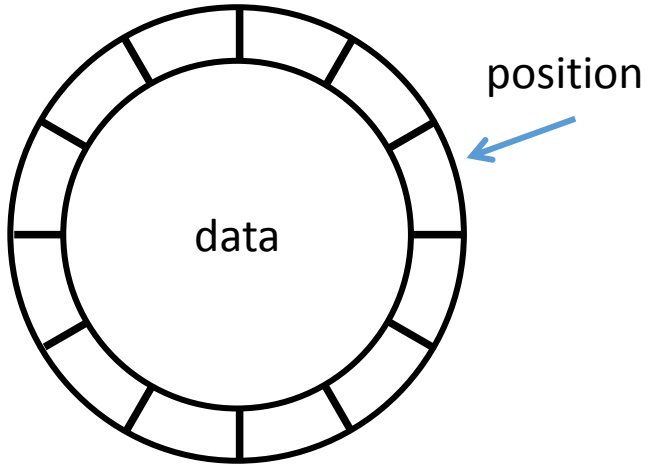
```
public class Statistics {

   // Konstruktor
   public Statistics()

   // Hinzufügen von Daten
   public void Put(double value)

   // Rückgabe des Mittelwertes
   public double Mean()

   // Rückgabe der geschätzten Varianz
   public double Variance()
}
```

Mögliche Implementationen?

- wachsendes Array
  (ähnlich wie in der Übung)
- **Zirkulärer Puffer**
- **Online-Algorithmus**

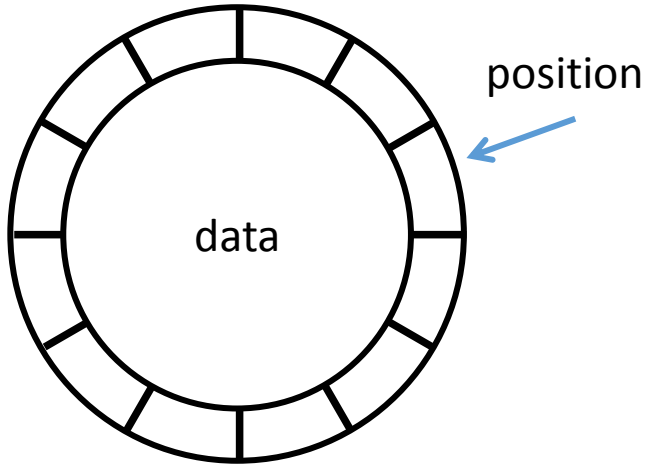
Zirkulärer Puffer

```
public class Statistics {
   private double data[];        // Zirkulärer Puffer
   private int n;                // Füllgrad
   private int pos;              // momentane Position

   // Konstruktor
   Statistics(int maxSize) {
      n = 0;
      pos = 0;
      data = new double[maxSize];
   }

   ...
}
```



Einfügen

```
public class Statistics {
   public Statistics() {...}

   // füge Wert in die den zirkulären Puffer ein
   public void Put(double value){
      data[pos] = value;
      if (n < data.length)
         n++;
      pos = (pos + 1) % data.length;
   }
...
}
```

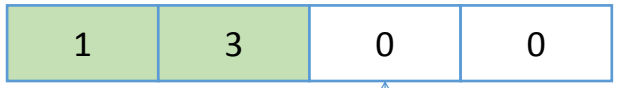


Szenario

```
Statistics s = new Statistics();
```

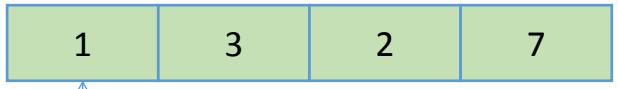
position n= 0

```
s.put(1);
s.put(3);
```



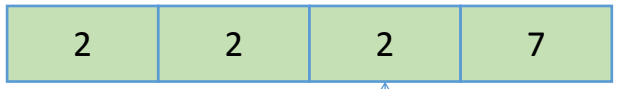
position n= 2

```
s.put(2);
s.put(7);
```



position n= 4

```
s.put(2);
s.put(2);
```



position n= 4

```
// füge Wert ein
public void Put(double value){
   data[pos] = value;
   if (n < data.length)
      n++;
   pos = (pos + 1) % data.length;
}
```

Mittelwert

```
public class Statistics {
   public Statistics() {...}
   public void Put(double value){...}

   public double Mean() {
      double sum = 0;
      for (int i=0; i<n; ++i)
         sum += data[i];
      return sum / n;
   }

   ...
}
```

Varianz

```
public class Statistics {
    public Statistics() {...}
    public void Put(double value){...}
    public double Mean() {...}

    public double Variance() {
        if (n > 1) {
            double ssq = 0;
            double mean = Mean();
            for (int i = 0; i < n; ++i)
                ssq += (data[i]-mean) * (data[i]-mean);
            return ssq / (n-1);
        }
        else
            return 0;
    }
}
```

2 * n Schritte

Frage

Ist der Algorithmus oben für sehr grosse Datensätze im Sinne eines online-Algorithmus geeignet?

- Antwort: nein, Online-Algorithmen halten nützliche Zwischenergebnisse und berechnen nicht jedes mal neu.

Effizienz

Obige Implementation ist nicht sehr effizient.

Wenn `Mean` oder `Variance` oft, z.B. nach jedem Eintrag eines Wertes, abgefragt wird, dann lohnt sich der *Provisional Means* Algorithmus

- Wenn $m_n = \frac{1}{n}\sum_{i=1}^{n} x_i$, dann $m_{n+1} = m_n + \frac{x_{n+1} - m_n}{n+1}$

- Für $s_n = \sum_{i=1}^{n}(x_i - m_n)^2$ analog: $s_{n+1} = s_n + (x_{n+1} - m_n) \cdot (x_{n+1} - m_{n+1})$

Damit fällt sogar die Notwendigkeit zum Speichern des Arrays weg

Provisional Means

```
public class Statistics {
   private int n = 0;
   private double mean = 0;
   private double ssq = 0;

   // Einfügen und Update
   public void Put(double value){
      n++;
      double oldMean = mean;
      mean = oldMean + (value - oldMean) / n;
      ssq = ssq + (value - oldMean) * (value - mean);
   }

   ...
}
```

Mittelwert und Varianz

```
public class Statistics {
   public void Put(double value){

   public double Mean(){
      return mean;
   }

   public double Variance() {
      if (n>1)
         return ssq / (n-1);
      else
         return 0;
   }
}
```

"1 Schritt"

"1 Schritt"

Fragen

Können wir das auch mit dem Median machen?

- Antwort: nein, deutlich komplizierter

Wenn $s(x,k)$ denn k-grössten Wert des Datensatzes x mit Länge n bezeichnet $(1 \leq k \leq n)$, so ist der Median definiert als

$$s\left(x, \frac{n+1}{2}\right), \quad \text{falls n ungerade und}$$

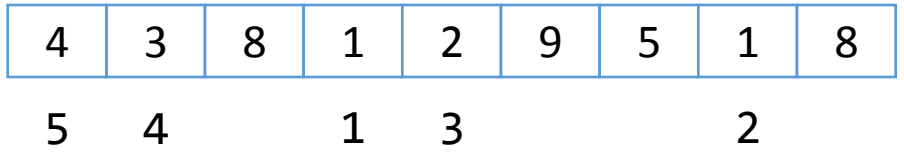
$$\frac{1}{2} \cdot \left(s\left(x, \frac{n}{2}\right) + s\left(x, \frac{n+1}{2}\right)\right), \quad \text{falls n gerade}$$

Was ist ein guter (On-line) Algorithmus für die Berechnung des Medians?

Median

Bestimmung des Medians ist ein *Auswahlproblem*

Naiv: Bestimmung des grössten / kleinsten Elements in n Schritten durch Traversieren aller Elemente.



Iterative Anwendung dieses Verfahrens unter Streichung des vorherigen Minimums liefert Median in

$\frac{n+1}{2} \cdot n$ Schritten (wenn n ungerade)

Bessere Ideen?

Median

Wir wissen schon aus Informatik I, dass *Mergesort* $n \log n$ Schritte (Präzisierung folgt!) benötigt.

Schneller als naive Methode:

- Sortieren des Arrays (Mergesort) und nachfolgend
- Auswahl des mittleren Elements (Array-Zugriff)

Nachteil

- Verändert die Daten oder benötigt Kopie der Daten

Vorteil

- Ist generisch: direkt für Auswahl des k-kleinsten Elements verwendbar

On-line Median?

Beobachtung: für zukünftige Updates des Medians sind immer alle Daten relevant.

- Es gibt keine Abkürzung wie beim Mittelwert.
- "Beweis": jeder Datenpunkt kann irgendwann zum Median werden.

Idee

- Daten werden grundsätzlich sortiert eingefügt. Geht das effizient?
- Wir kommen darauf später noch einmal zurück

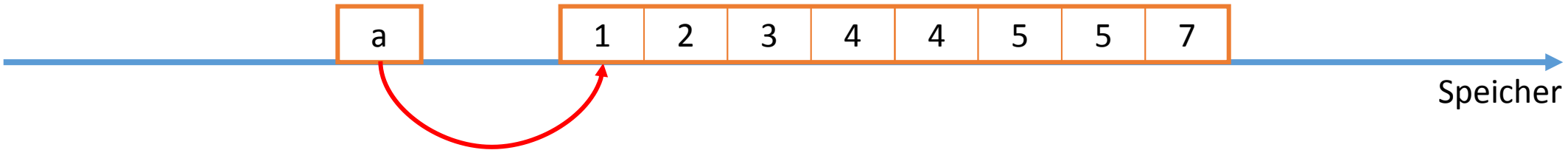
Verkettete Listen, Stapel

5. DYNAMISCHE DATENSTRUKTUREN

Container für eine Folge gleichartiger Daten

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

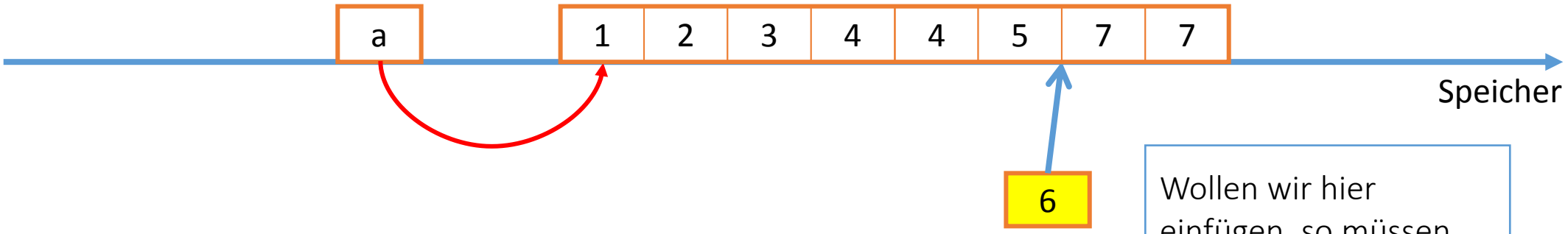
```
int a[] = new int[8];
```



Einmal alloziert ist die Länge fixiert

Probleme mit Arrays

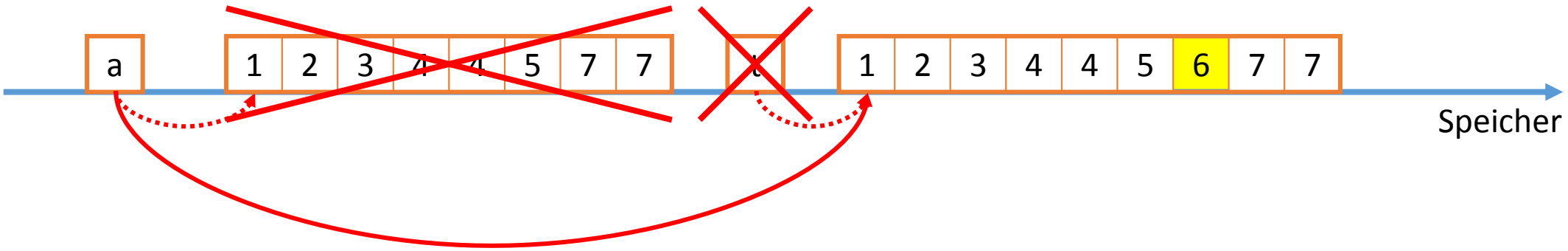
Einfügen oder Löschen von Elementen "in der Mitte" ist aufwändig



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

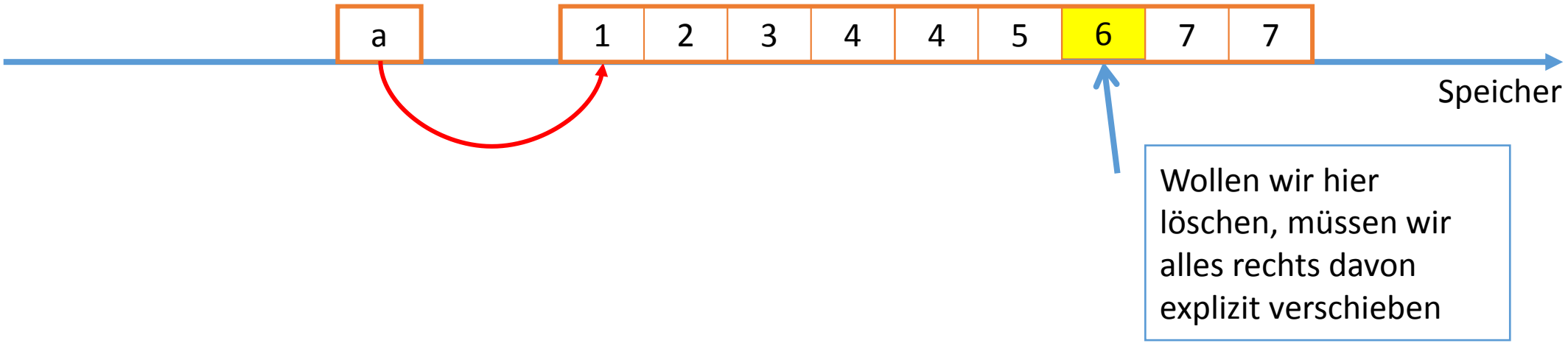
Probleme mit Arrays

Beispiel: Einfügen unter Neuallokation des Array.



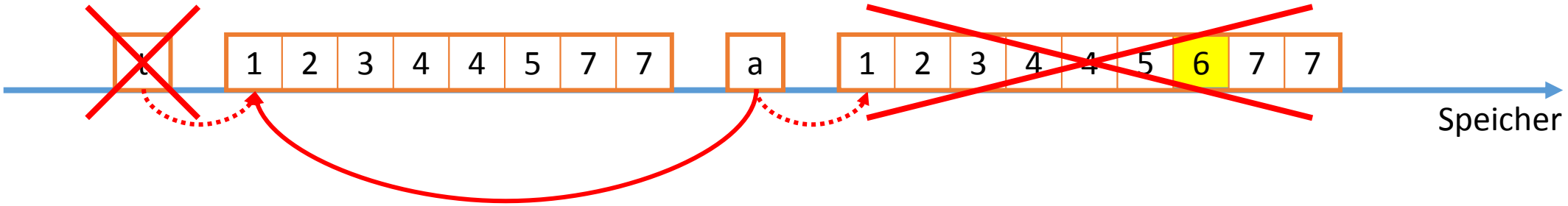
Probleme mit Arrays

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



Probleme mit Arrays

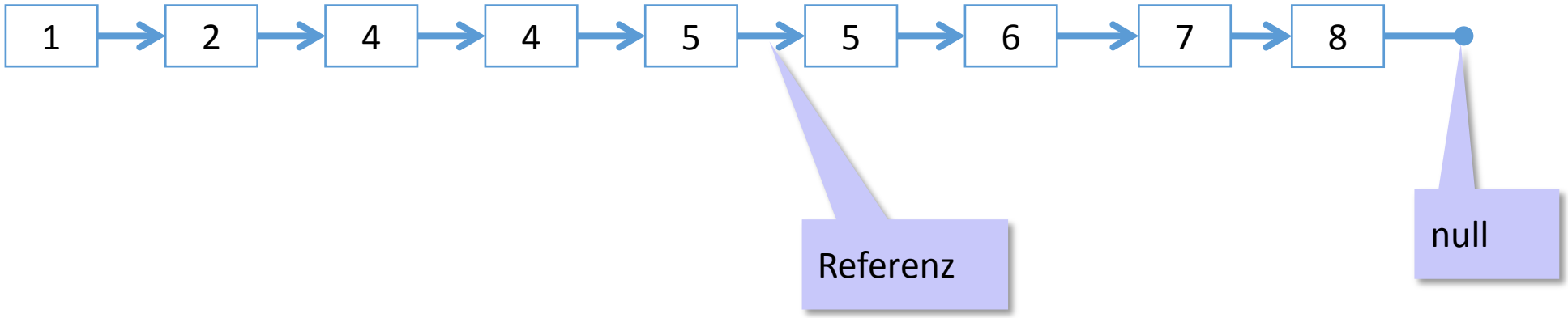
Beispiel: Löschen mit Neuallokation des Arrays



Lösung: (Verkettete) Listen

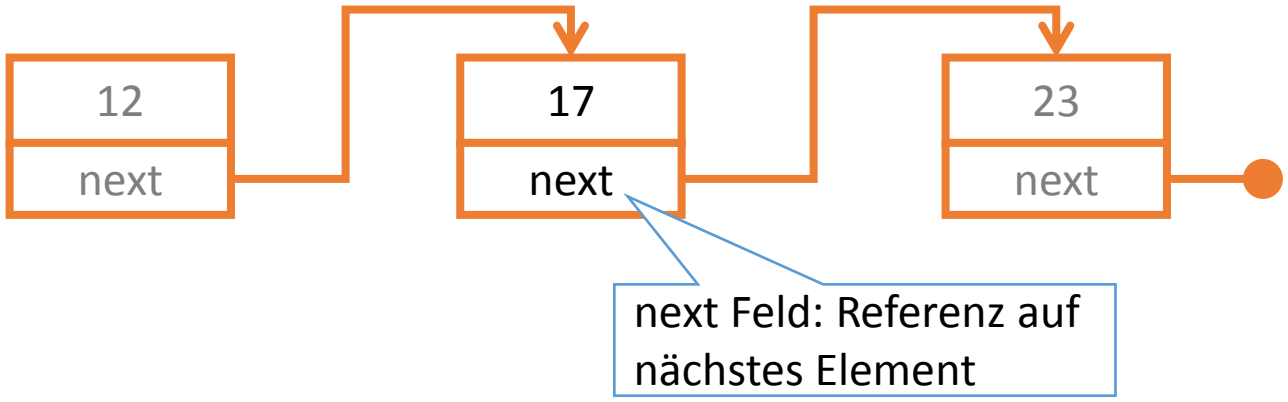
Container für eine Folge von Daten des gleichen Typs

Kein zusammenhängender Speicherbereich, kein wahlfreier Zugriff



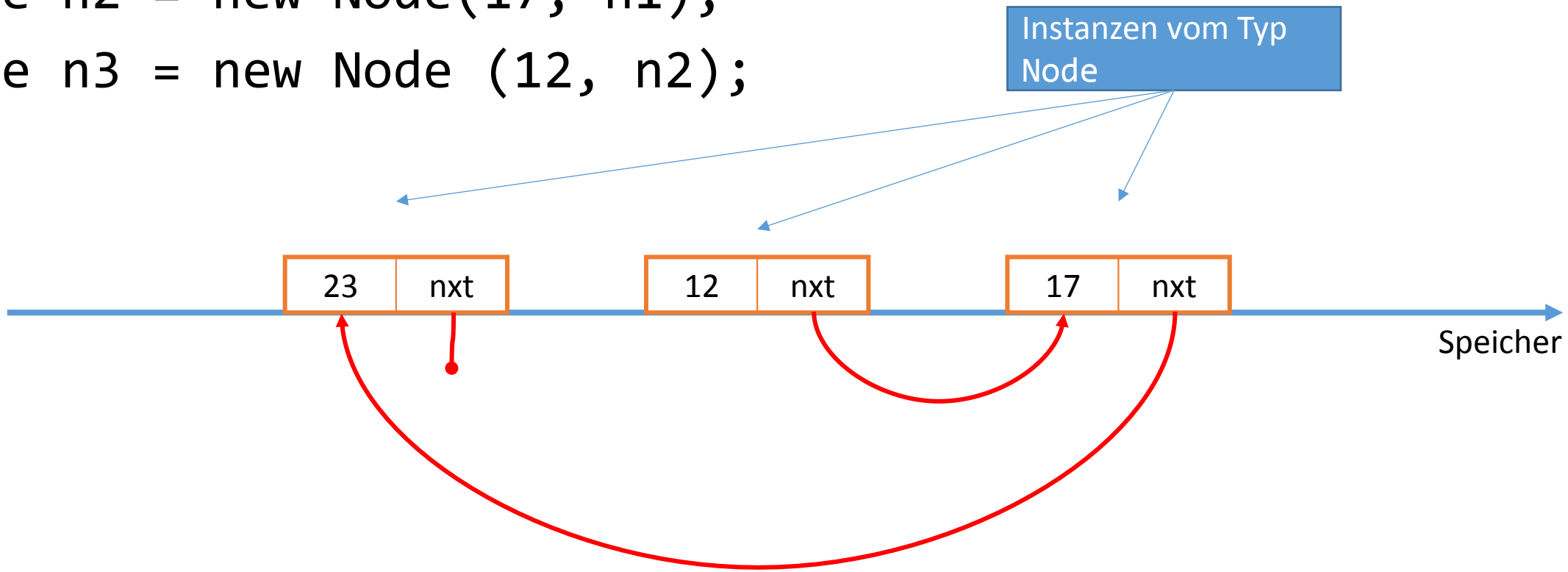
(Einfach) verkettete Liste

```
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 (LIFO – Last In First Out)

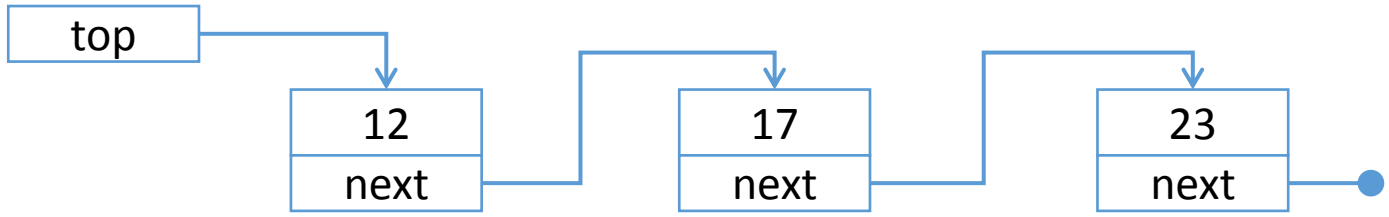
Der Stapel ist das einfachste Beispiel einer Datenstruktur, welche man gut mit einer verketteten Liste implementieren kann.

Funktionalität:

Knoten vorne einfügen: **Push**

Knoten vorne herausnehmen: **Pop**

Erstes Element durch "top" repräsentiert

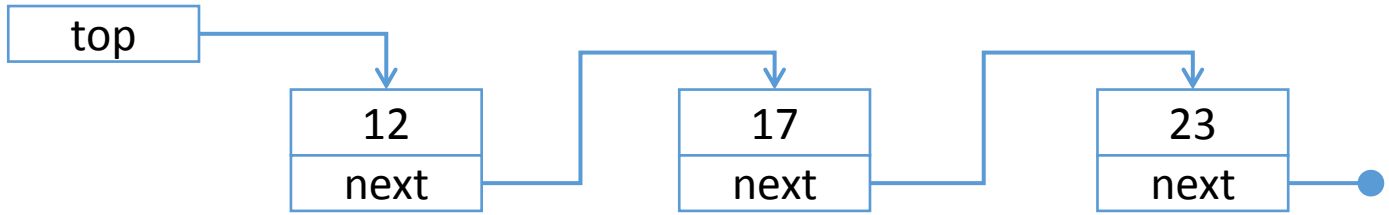


Stapel: Interface

```
public class Stack {

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

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

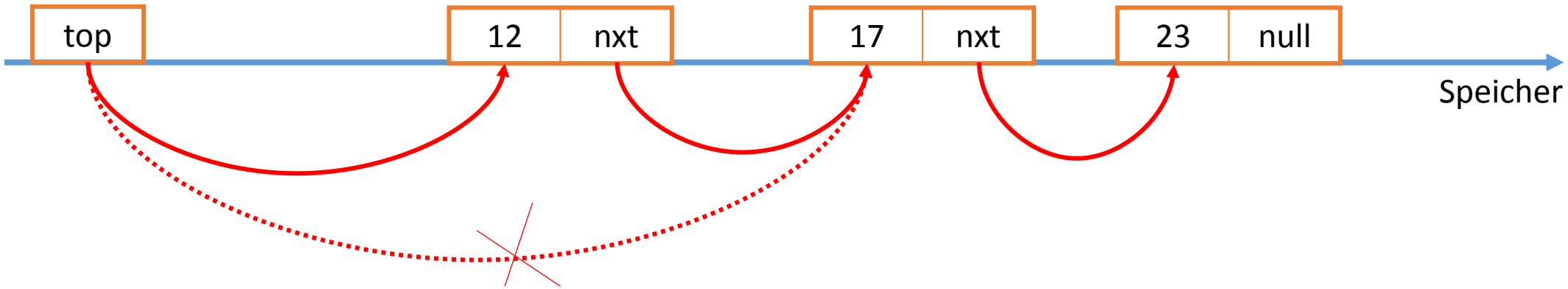
Stapel mit verketteter Liste

```
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
        }
}
```