

# MAS: Betriebssysteme

## Grundlegende Datenstrukturen für Betriebssysteme (Exkurs)

T. Pospíšek

# Zielsetzung für diese Vorlesung

---

- Datenstrukturen, die man in Betriebssystemen häufig vorfindet, verstehen

# Wichtige Datenstrukturen für Betriebssysteme

---

- Zeiger und Referenztypen
- Sequenzen bzw. diverse Listen
- Stapelspeicher (Kellerspeicher, Stack)
- Warteschlangen (Queues)
- Hashtabellen

# Zeiger und Referenztypen

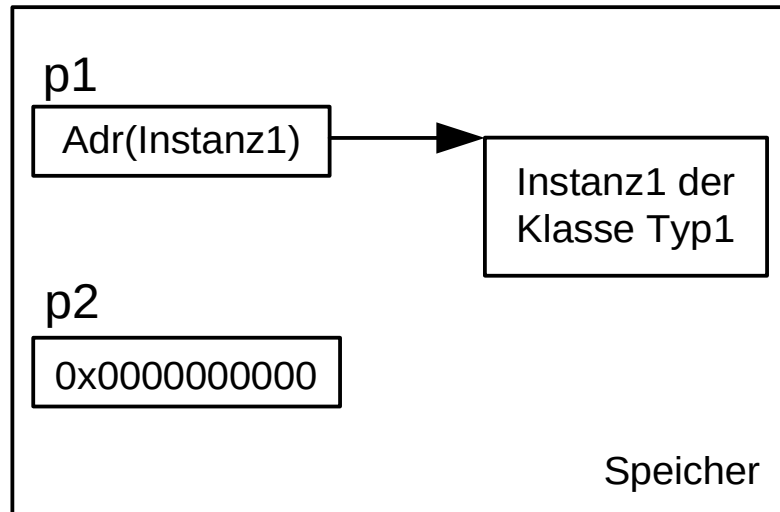
```
class MyType { ...} // Klasse MyType wird definiert.
```

```
MyType p1; // Referenz p1 wird erzeugt
```

```
MyType p2; // Referenz p2 wird erzeugt
```

```
...
```

```
p1 = new MyType(); // Standardkonstruktor wird durchlaufen
```



**p1** enthält  
Speicheradresse  
einer konkreten  
Instanz (Instanz1)

**p2** zeigt auf keine  
konkrete Instanz

# Sequenzen und diverse Listen

## ■ Was ist eine Liste?

- Allgemein ist eine Sequenz  $\langle a_1, \dots, a_n \rangle$  eine Folge aus  $n$  Elementen des gleichen Grundtyps
- Grundtypen sind beliebige Datentypen
- Mögliche Beschreibung als Abstrakter Datentyp (ADT)
- Verschiedene Implementierungsvarianten
- Z.B. als einfaches Array, Adressierung über Index
- Adresse eines Elements  $i$   
= Anfangsadresse des Arrays +  
( $i$  \* Länge des Elements)

Index	
	1333
1	1444
2	1555
3	132
4	1400
5	2
6	15
7	4

# Sequenzen und diverse Listen

## ■ Beispiel einer formalen Beschreibung

**algebra** MyList  
**sorts** list, elem, bool  
**ops**    empty    :     $\square$  list  
          first : list  $\square$  elem  
          rest : list  $\square$  list  
          append : list x elem  $\square$  list  
          concat : list x list  $\square$  list  
          isempty : list  $\square$  bool  
**sets**    list    = {  $\langle a_1, \dots, a_n \rangle \mid n \geq 0, a_i \in \text{elem} \}$   
**functions**  
          empty                = 0  
          first ( $a_1 \dots a_n$ )        =  $a_1$ , falls  $n > 0$ ; sonst undefiniert  
          rest ( $a_1 \dots a_n$ )        =  $a_2 \dots a_n$ , falls  $n > 0$ ; sonst undefiniert  
          append ( $a_1 \dots a_n, x$ )    =  $x a_1 \dots a_n$   
          concat ( $a_1 \dots a_n, b_1, \dots b_m$ ) =  $a_1 \dots a_n * b_1, \dots b_m$   
          isempty ( $a_1 \dots a_n$ )        =  $(n = 0)$   
**end** MyList

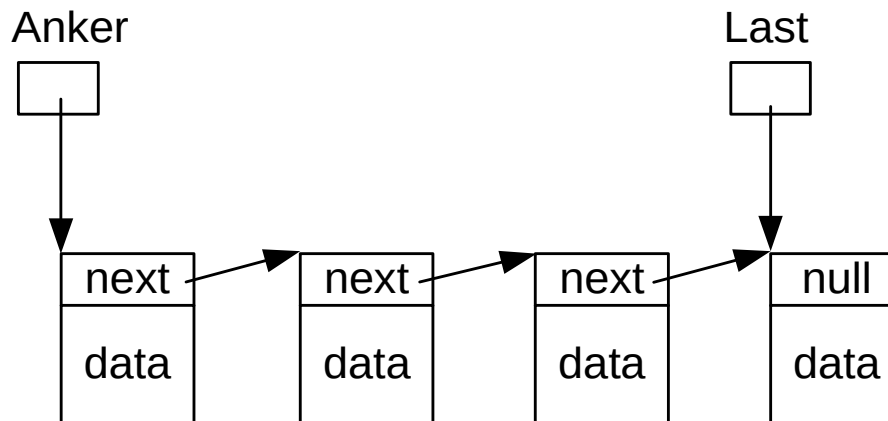
Verwendete Typen

Operationen

# Sequenzen und diverse Listen

## ■ Implementierung als einfach verkettete lineare Liste

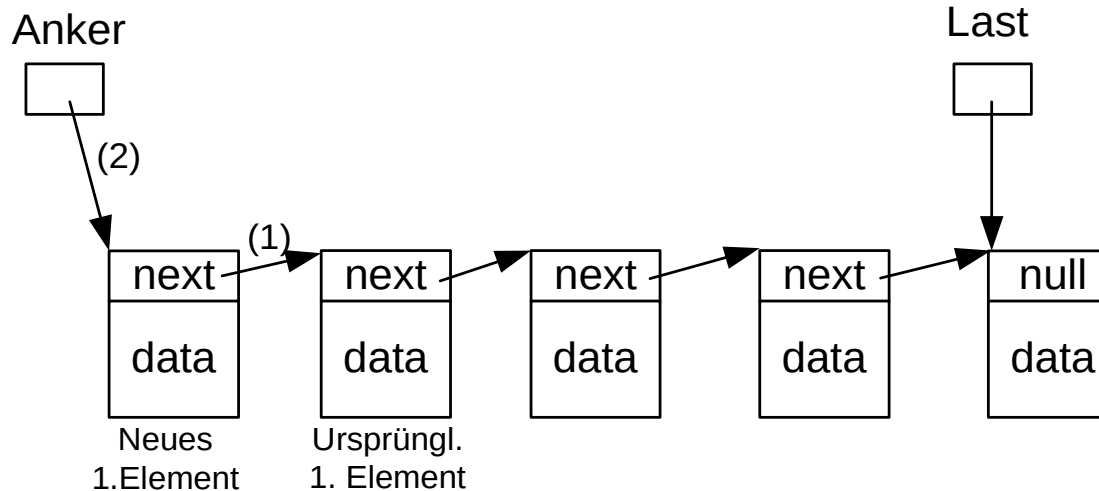
- Anker zeigt auf erstes Element
- Last zeigt auf letztes Element



# Sequenzen und diverse Listen

## ■ Implementierung als einfach verkettete lineare Liste

- Ergänzen eines Elements am Anfang der Liste
- 2 Operationen (1) und (2) erforderlich

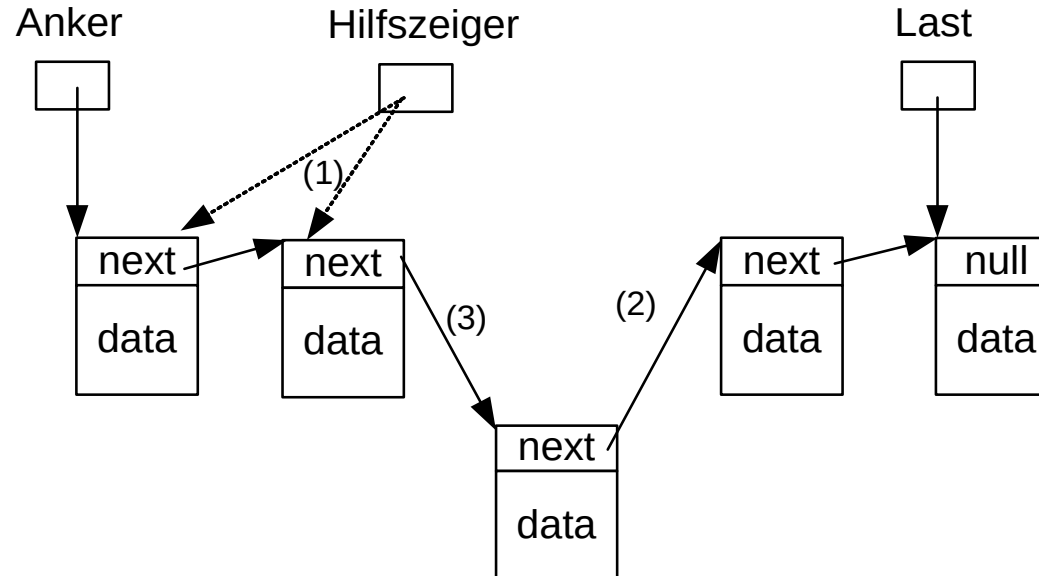




# Sequenzen und diverse Listen

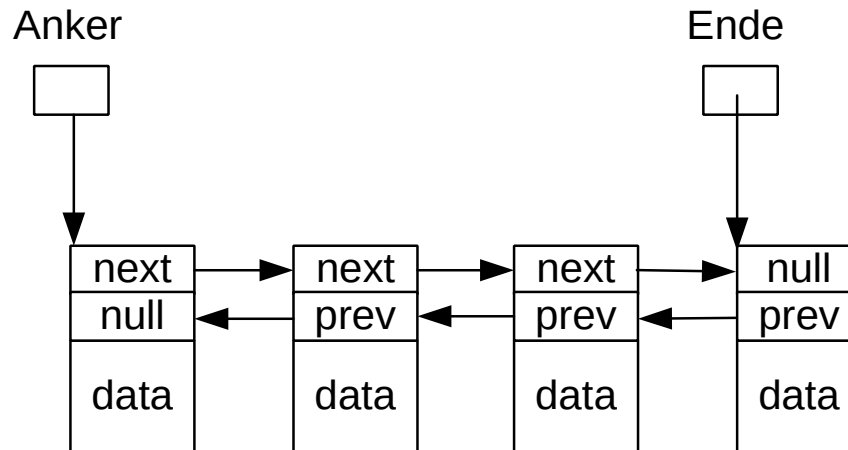
## ■ Implementierung als einfach verkettete lineare Liste

- Einfügen eines Elements
- 3 Operationen (1), (2) und (3) erforderlich



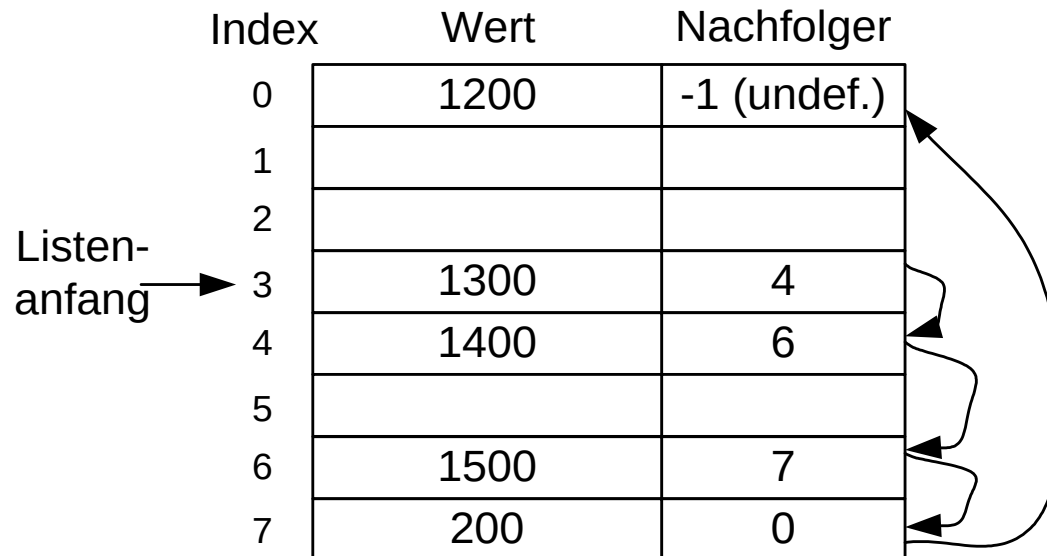
# Sequenzen und diverse Listen

- **Implementierung als doppelt verkettete lineare Liste**
  - Vorteile?



# Sequenzen und diverse Listen

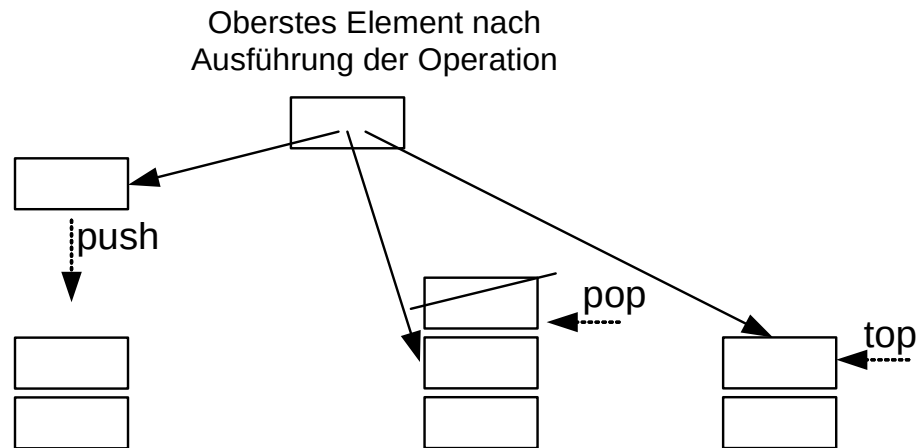
- **Implementierung einer einfach verketteten Liste in einem Array**
  - Vorteile?



# Stack

## ■ Kellerspeicher = Stapelspeicher = Stack

- LIFO-Prinzip
- Dient dazu, eine beliebige Anzahl von meist gleich großen „Objekten“ zu speichern
- Entnahme in umgekehrter Reihenfolge
- Operationen: push, pop, top



# Queue

## ■ Warteschlange = Queue

- FIFO-Prinzip
- Spezielle Listen, bei denen Elemente nur vorne eingefügt und hinten, also am anderen Ende entnommen werden können

Operationen: *enqueue*, *dequeue*

**algebra** MyQueue

**sorts** queue, elem, bool

**ops**

empty :  $\square$  queue // Anlegen einer Queue

front : queue  $\square$  elem // Erstes Element

enqueue : queue x elem  $\square$  queue // Einfügen eines Elements

dequeue : queue  $\square$  queue // Abholen eines Elements

isempty : queue  $\square$  bool // Abfrage, ob Queue leer ist

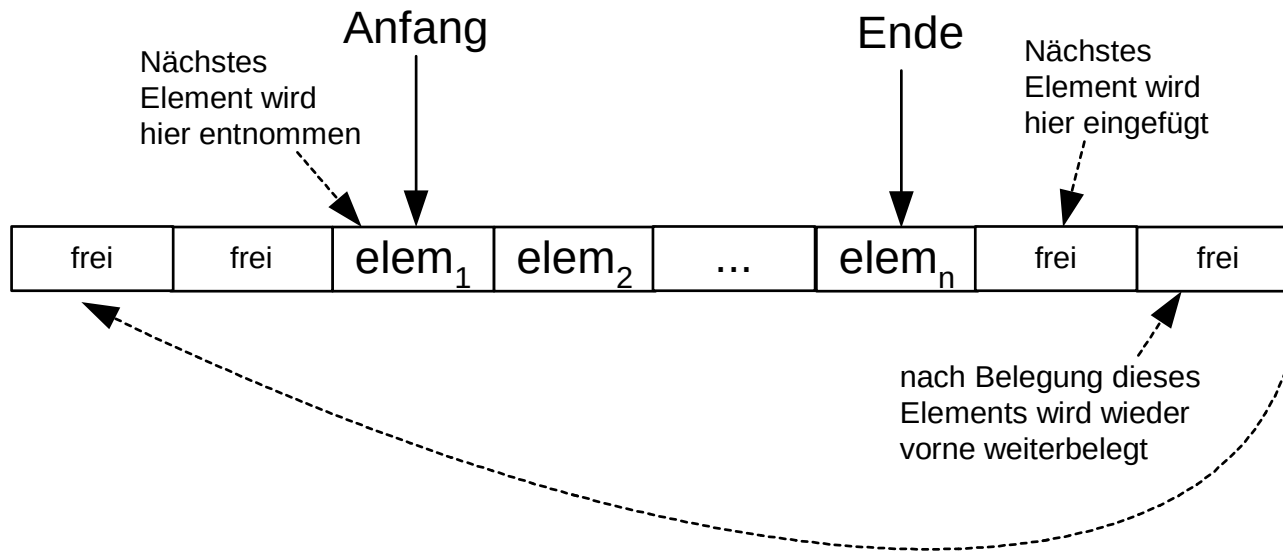
...

**end** MyQueue

# Queue

## ■ Implementierung einer Queue in einem Array

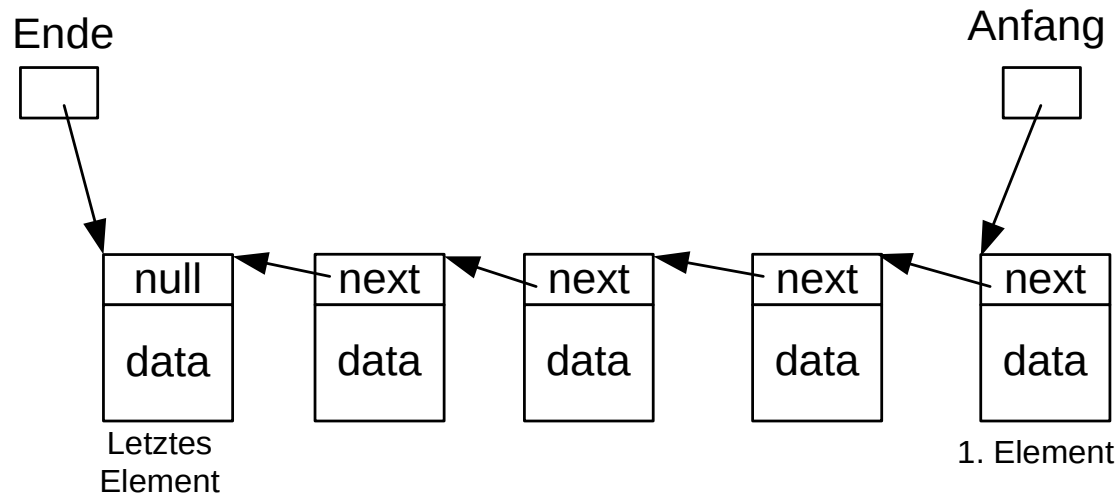
- Zyklische Queue
- Pufferüberlauf?



# Queue

## ■ Implementierung einer Queue als verkettete Liste

- Anfangs- und Endzeiger
- Spezielle Form: Vorrang- bzw. Prioritätswarteschlange



# Hashtabellen

## ■ Hashtabellen

- Spezielle **Indexstruktur** zum schnellen Auffinden von Elementen in **großen Datenmengen**

## ■ Hashverfahren

- **Hashverfahren** zur Auffindung der gesuchten Elemente notwendig  
□ Hashfunktion

## ■ Hashfunktion

- **Hashfunktion** ermittelt aus einem Element-Schlüssel einen Hash-Wert
- **Hashwert** stellt den Index des gesuchten Elements innerhalb der Tabelle dar
- Ideal: Die Hashfunktion liefert genau einen Hash-Wert
- Aber Hashfunktionen sind in der Regel **nicht ein-eindeutig**  
□ Kollisionen möglich

## ■ Abbildung

$h: U \rightarrow \{0, 1, \dots, m-1\}$ , wobei  $U$  die Menge aller möglichen Schlüssel ist



# Hashtabellen: Beispiel

- Hashfunktion: Modulo-Funktion
  - Beispiel: mod 6

