

# Teil 8: Dynamische Strukturen

## ■ Gliederung

Datenabstraktion

Stack

Queue

Verkettete Liste

Binärer Suchbaum

## ■ Motivation

```
typedef struct
{
    char name[20];
    char vorname[20];
    unsigned int ma_nummer;
    float gehalt;
} MITARBEITER;
```

**Problem:** Wie kann man eine Beliebige Anzahl von Strukturen des Typs `MITARBEITER` effizient verwalten?

**Bedingungen:**

- Anzahl der Strukturen variabel
- Speicher effizient verwenden
- schneller Zugriff

## Statisches Array

```
MITARBEITER ma_datenbank[200];
```

<code>char name[20]</code>	<code>char name[20]</code>	<code>char name[20]</code>	<code>char name[20]</code>
<code>char vorname[20]</code>	<code>char vorname[20]</code>	<code>char vorname[20]</code>	<code>char vorname[20]</code>
<code>unsigned int ma_nummer;</code>	<code>unsigned int ma_nummer;</code>	<code>unsigned int ma_nummer;</code>	<code>unsigned int ma_nummer;</code>
<code>float gehalt;</code>	<code>float gehalt;</code>	<code>float gehalt;</code>	<code>float gehalt;</code>

...

## Statisches Array

```
MITARBEITER ma_datenbank[200];
```

char name[20]	<del>char name[20]</del>	char name[20]	char name[20]
char vorname[20]	<del>char vorname[20]</del>	char vorname[20]	char vorname[20]
unsigned int ma_nummer;	<del>unsigned int ma_nummer;</del>	unsigned int ma_nummer;	unsigned int ma_nummer;
float gehalt;	<del>float gehalt;</del>	float gehalt;	float gehalt;

...

## Statisches Array

```
MITARBEITER ma_datenbank[200];
```

char name[20]		char name[20]	char name[20]
char vorname[20]		char vorname[20]	char vorname[20]
unsigned int ma_nummer;		unsigned int ma_nummer;	unsigned int ma_nummer;
float gehalt;		float gehalt;	float gehalt;

...

## Statisches Array

```
MITARBEITER ma_datenbank[200];
```

char name[20]		char name[20]	char name[20]
char vorname[20]		char vorname[20]	char vorname[20]
unsigned int ma_nummer;		unsigned int ma_nummer;	unsigned int ma_nummer;
float gehalt;		float gehalt;	float gehalt;

...

## Zeiger-Array

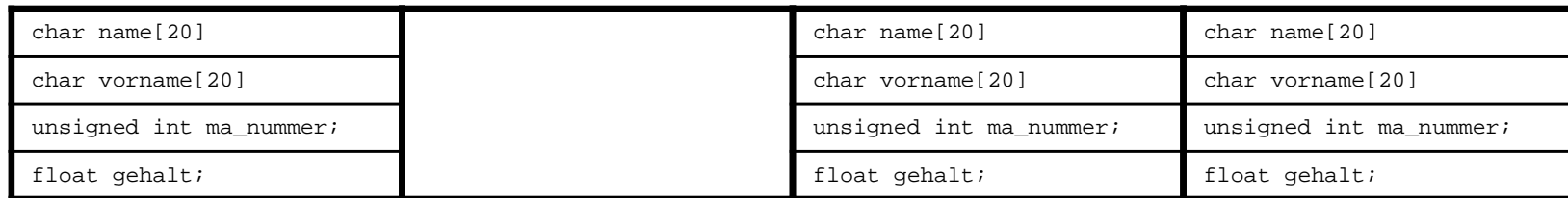
```
MITARBEITER *ma_datenbank[200];
```



...

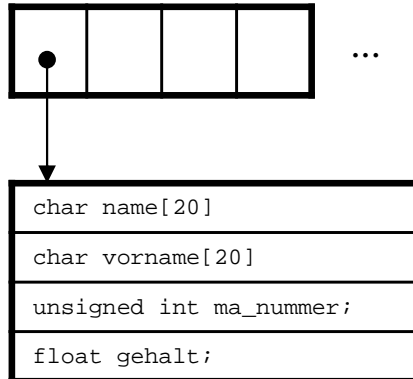
## Statisches Array

```
MITARBEITER ma_datenbank[200];
```



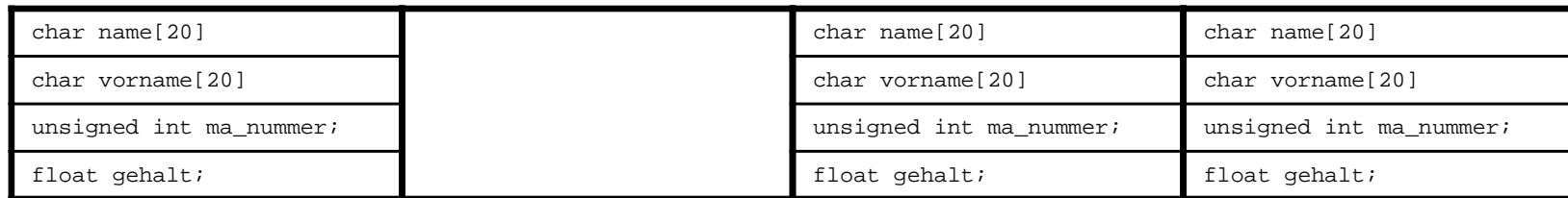
## Zeiger-Array

```
MITARBEITER *ma_datenbank[200];
```



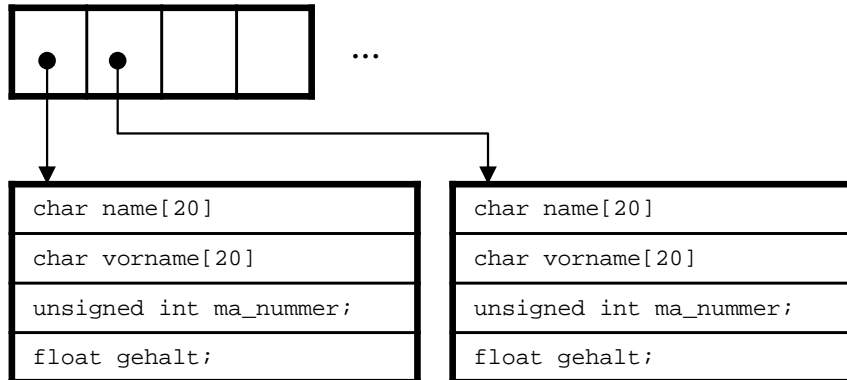
## Statisches Array

```
MITARBEITER ma_datenbank[200];
```



## Zeiger-Array

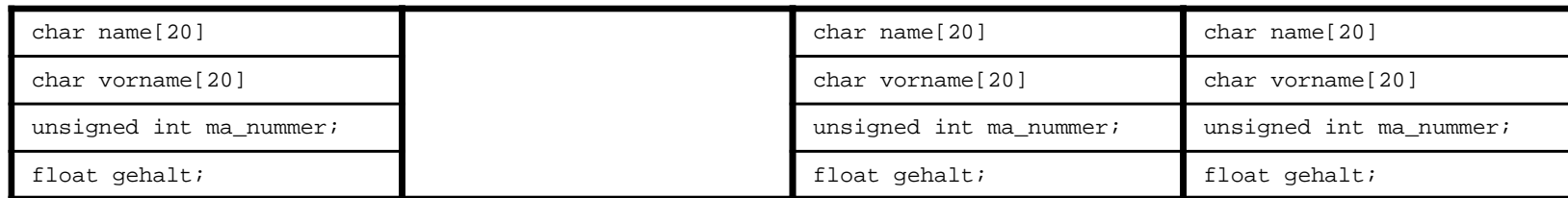
```
MITARBEITER *ma_datenbank[200];
```





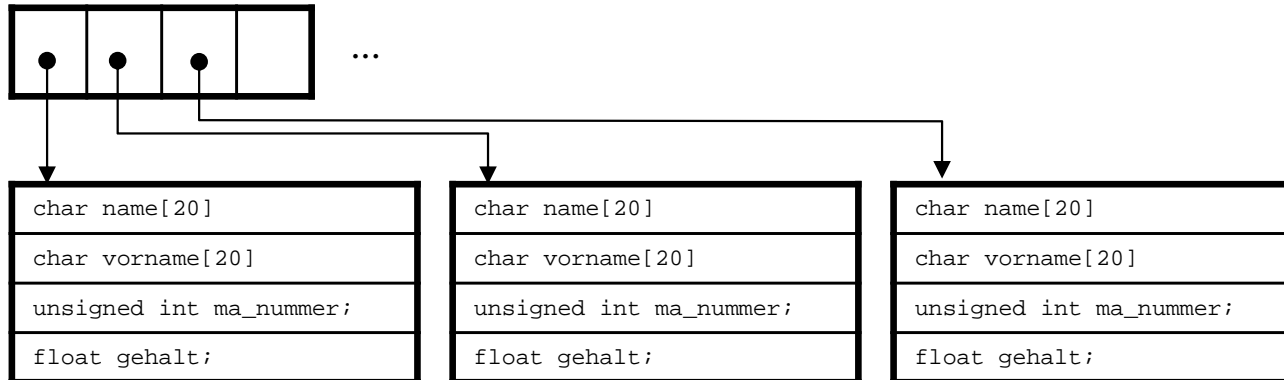
## Statisches Array

```
MITARBEITER ma_datenbank[200];
```



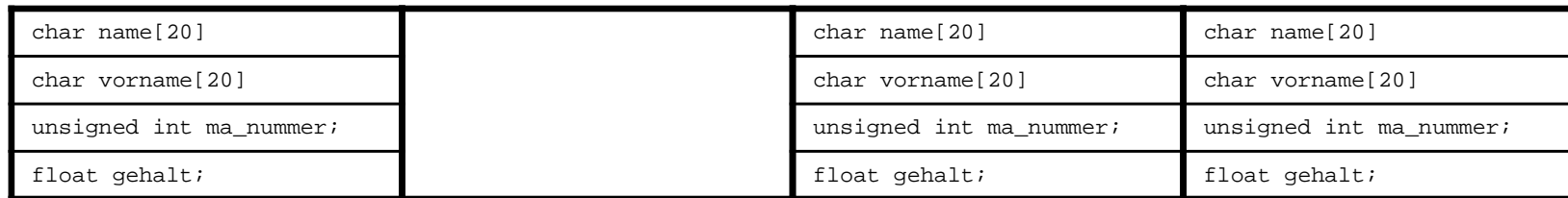
## Zeiger-Array

```
MITARBEITER *ma_datenbank[200];
```



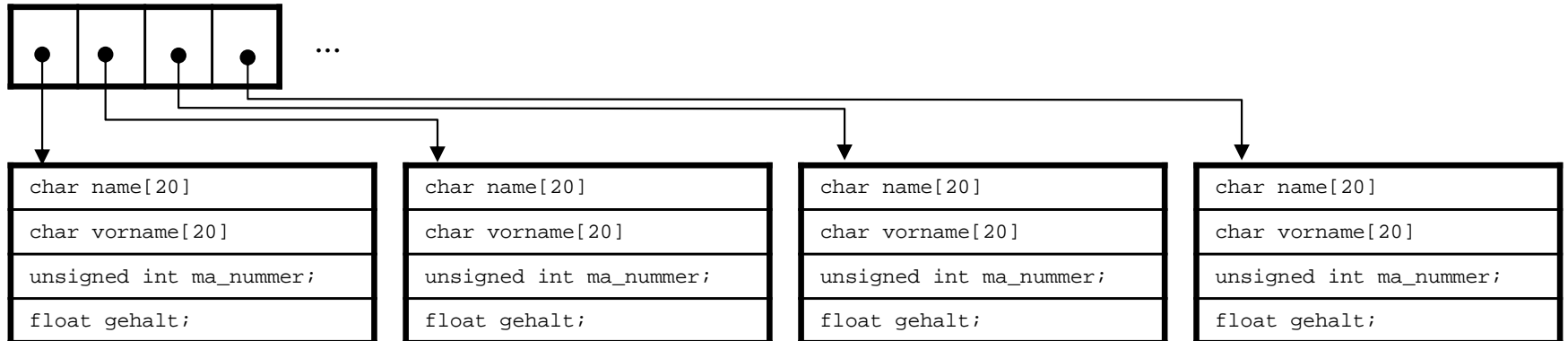
## Statisches Array

```
MITARBEITER ma_datenbank[200];
```



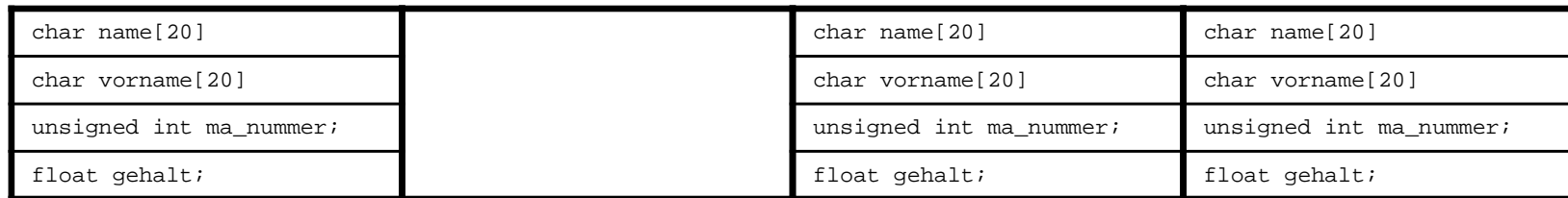
## Zeiger-Array

```
MITARBEITER *ma_datenbank[200];
```



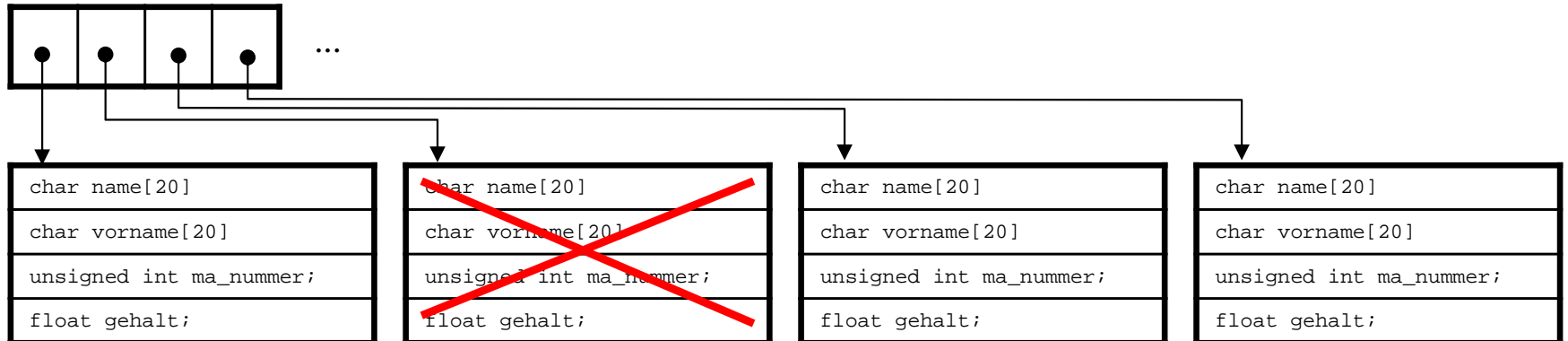
## Statisches Array

```
MITARBEITER ma_datenbank[200];
```



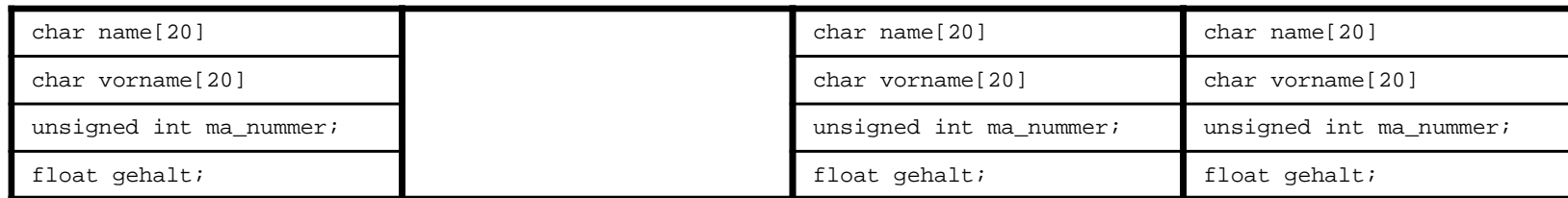
## Zeiger-Array

```
MITARBEITER *ma_datenbank[200];
```



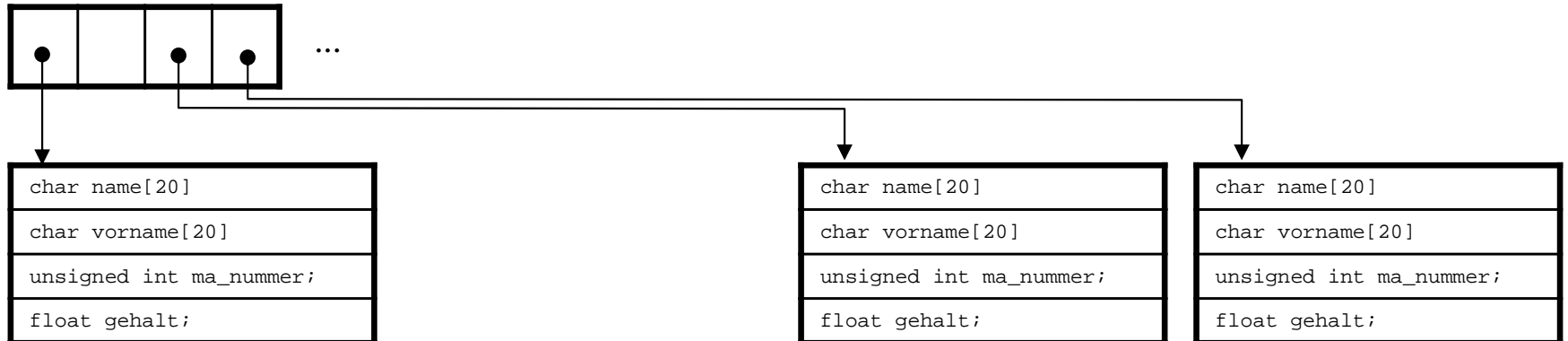
## Statisches Array

```
MITARBEITER ma_datenbank[200];
```



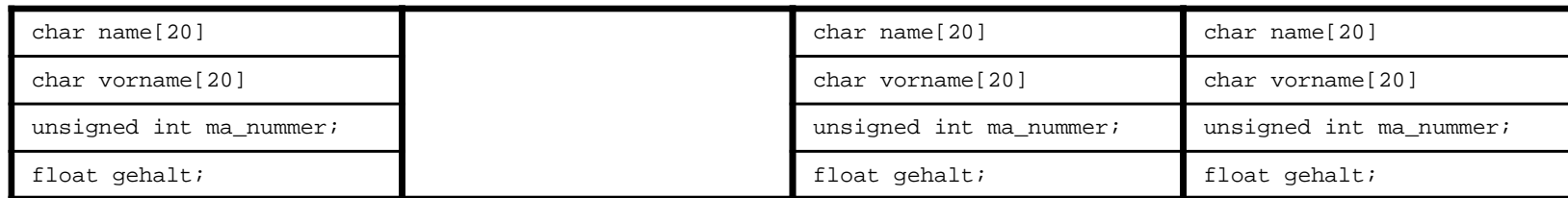
## Zeiger-Array

```
MITARBEITER *ma_datenbank[200];
```



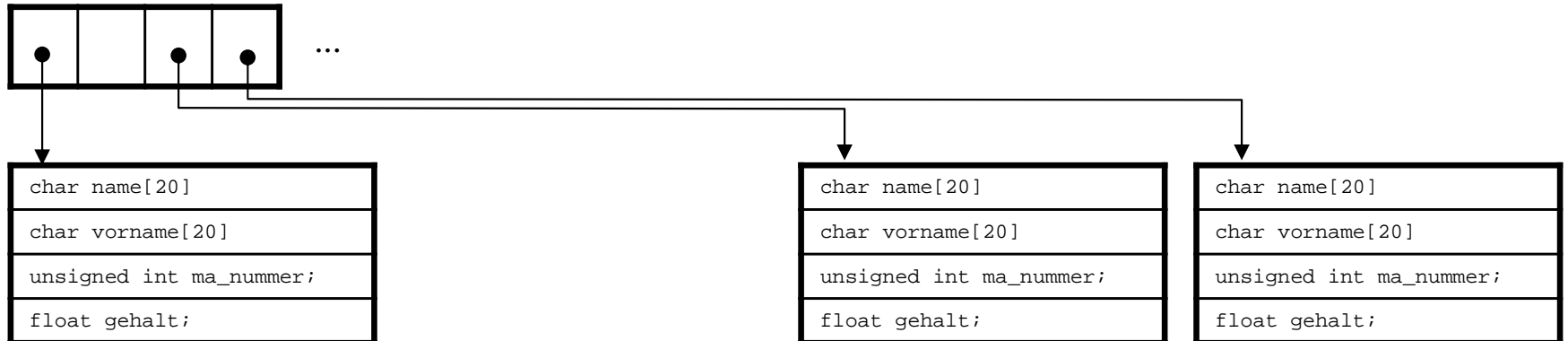
## Statisches Array

```
MITARBEITER ma_datenbank[200];
```



## Zeiger-Array

```
MITARBEITER *ma_datenbank[200];
```

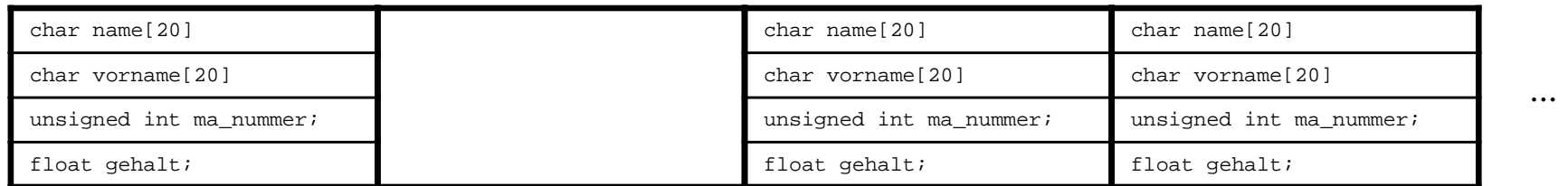


## Verkettete Liste

```
MITARBEITER *ma_liste;
```

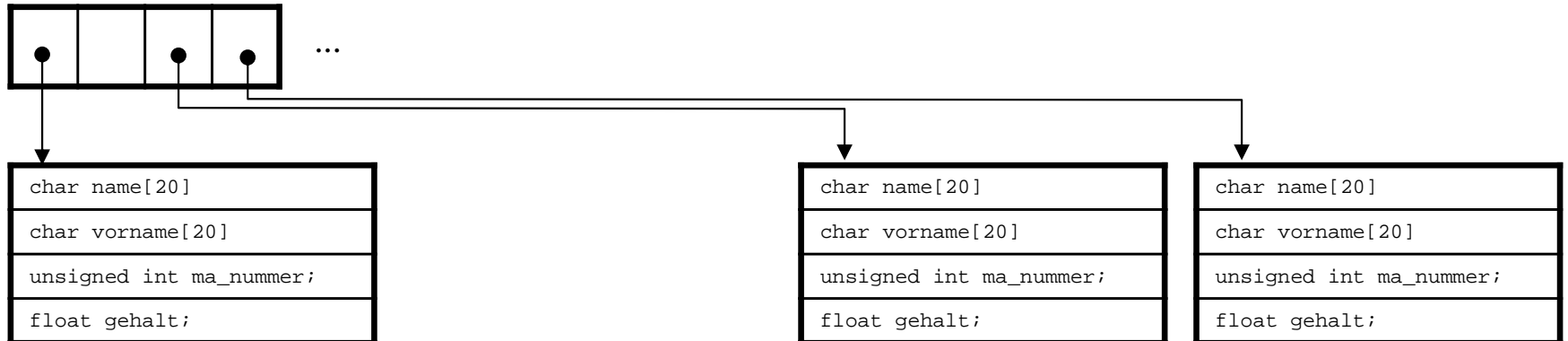
## Statisches Array

```
MITARBEITER ma_datenbank[200];
```



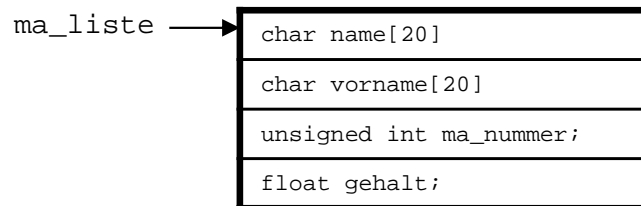
## Zeiger-Array

```
MITARBEITER *ma_datenbank[200];
```



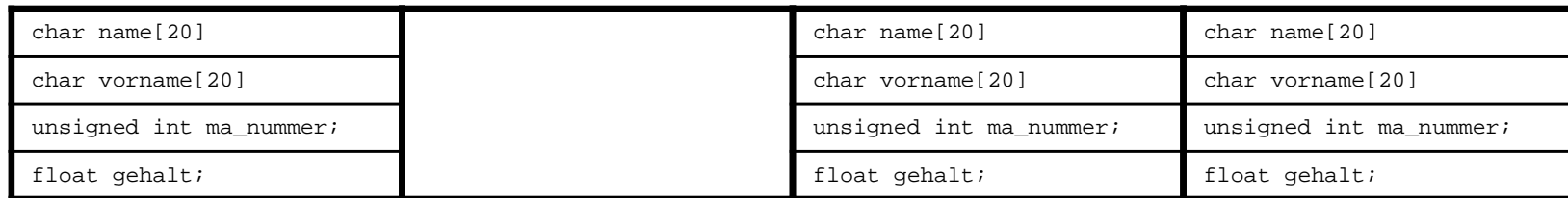
## Verkettete Liste

```
MITARBEITER *ma_liste;
```



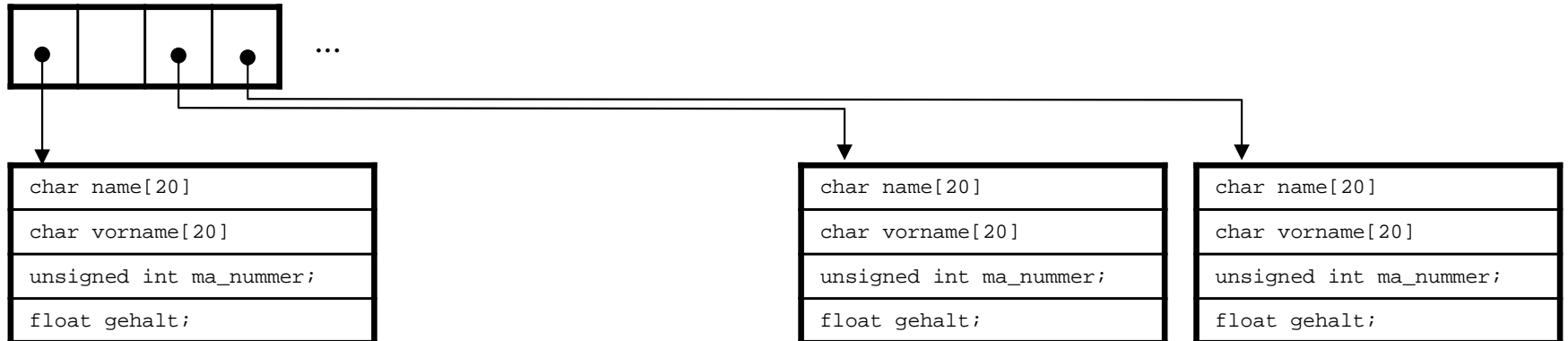
## Statisches Array

```
MITARBEITER ma_datenbank[200];
```



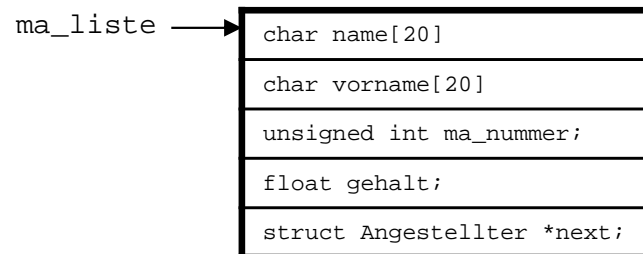
## Zeiger-Array

```
MITARBEITER *ma_datenbank[200];
```



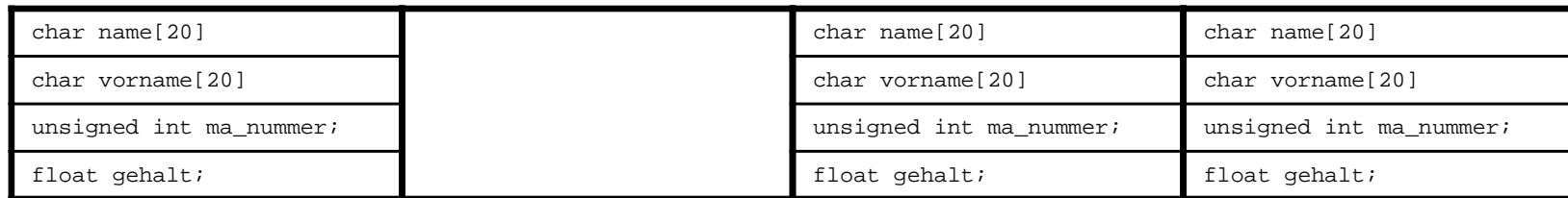
## Verkettete Liste

```
MITARBEITER *ma_liste;
```



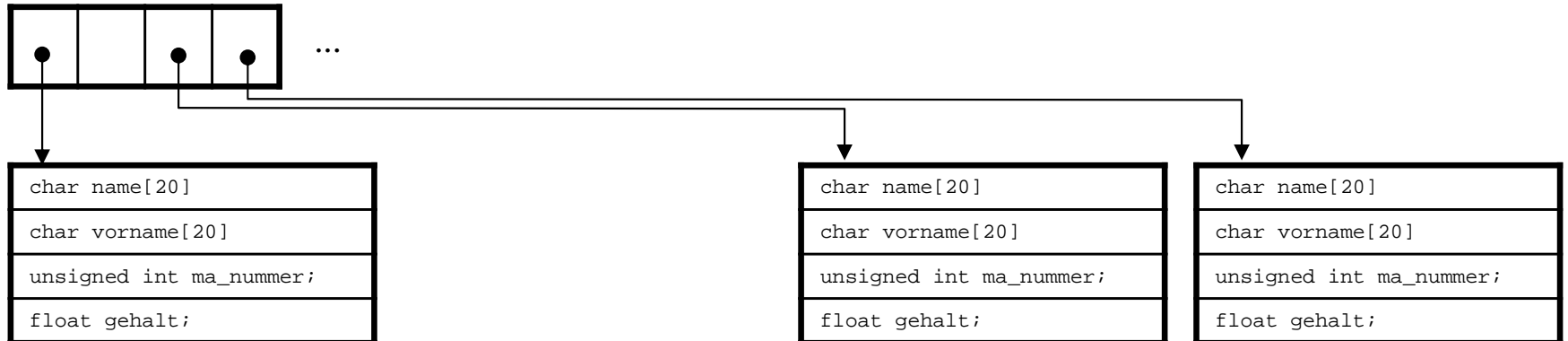
## Statisches Array

```
MITARBEITER ma_datenbank[200];
```



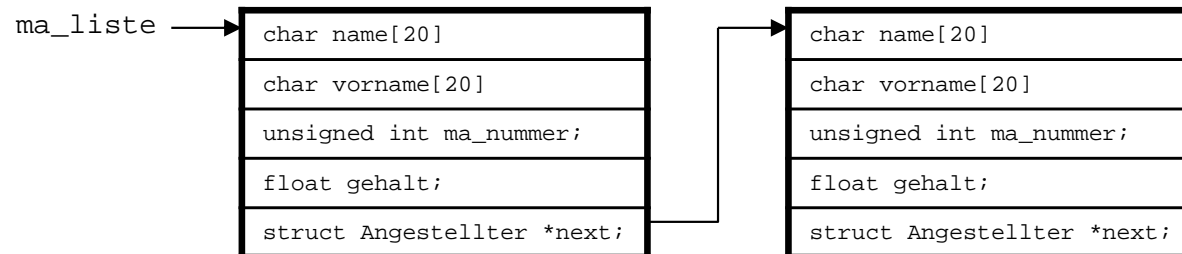
## Zeiger-Array

```
MITARBEITER *ma_datenbank[200];
```



## Verkettete Liste

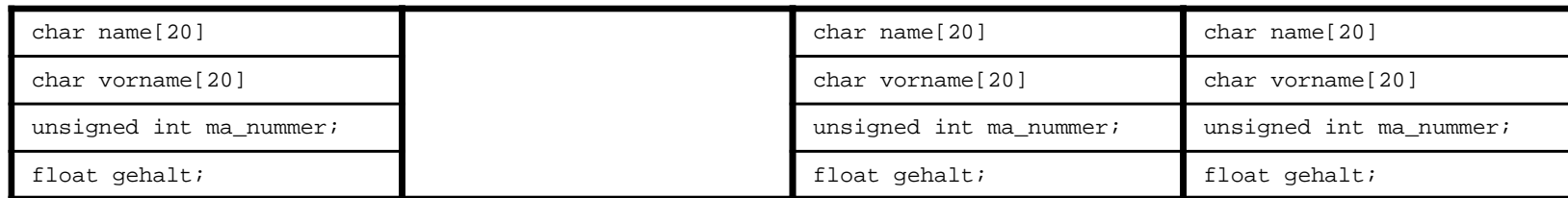
```
MITARBEITER *ma_liste;
```





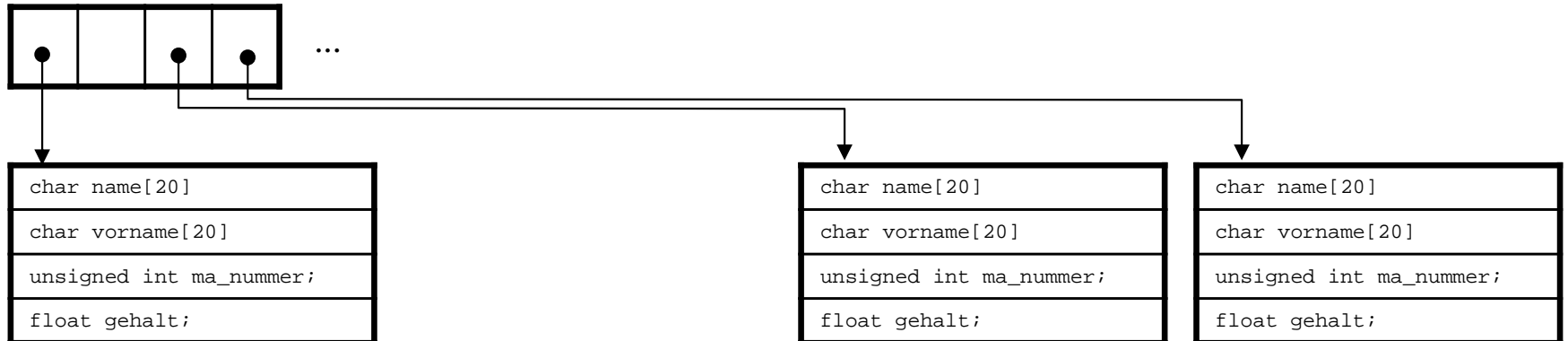
## Statisches Array

```
MITARBEITER ma_datenbank[200];
```



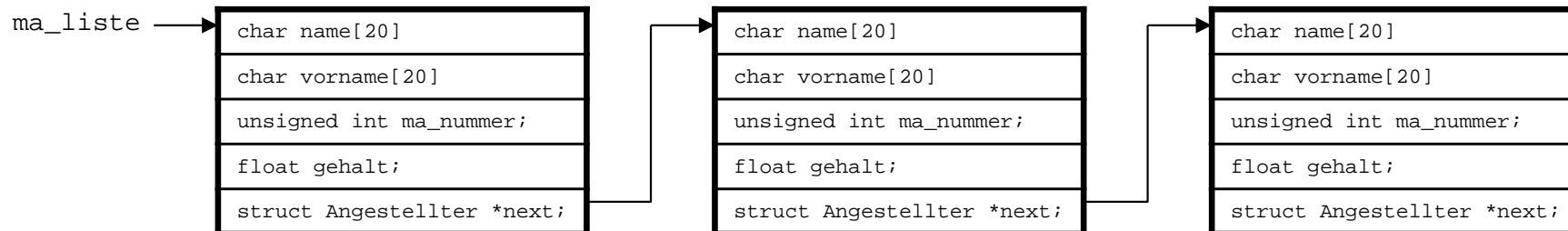
## Zeiger-Array

```
MITARBEITER *ma_datenbank[200];
```



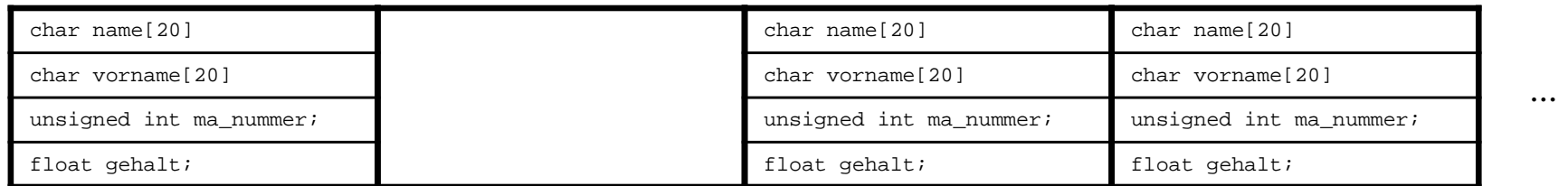
## Verkettete Liste

```
MITARBEITER *ma_liste;
```



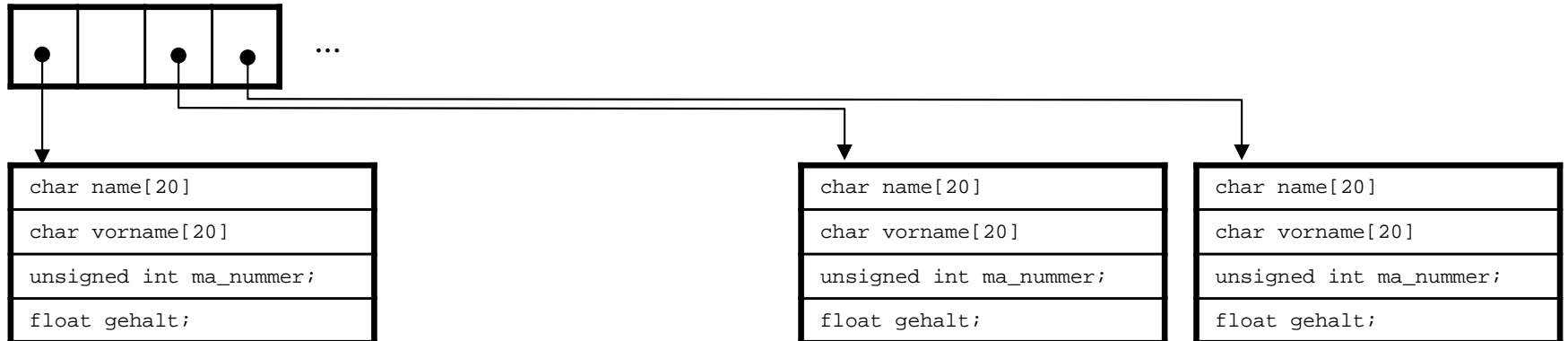
## Statisches Array

```
MITARBEITER ma_datenbank[200];
```



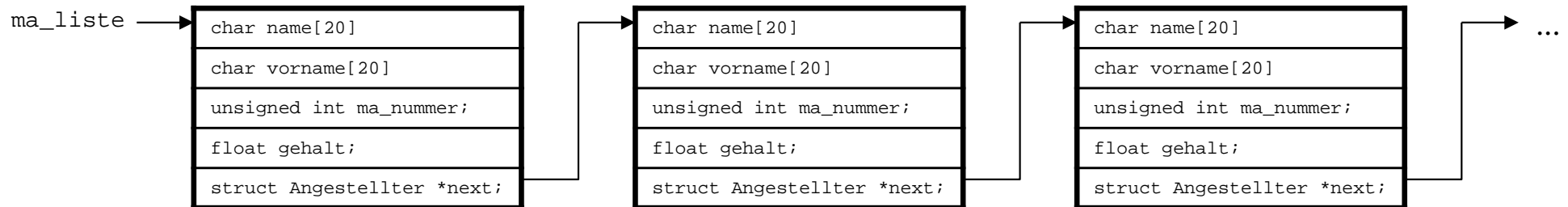
## Zeiger-Array

```
MITARBEITER *ma_datenbank[200];
```



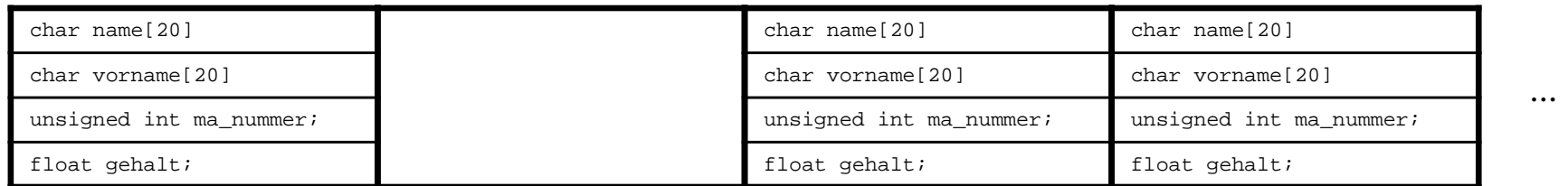
## Verkettete Liste

```
MITARBEITER *ma_liste;
```



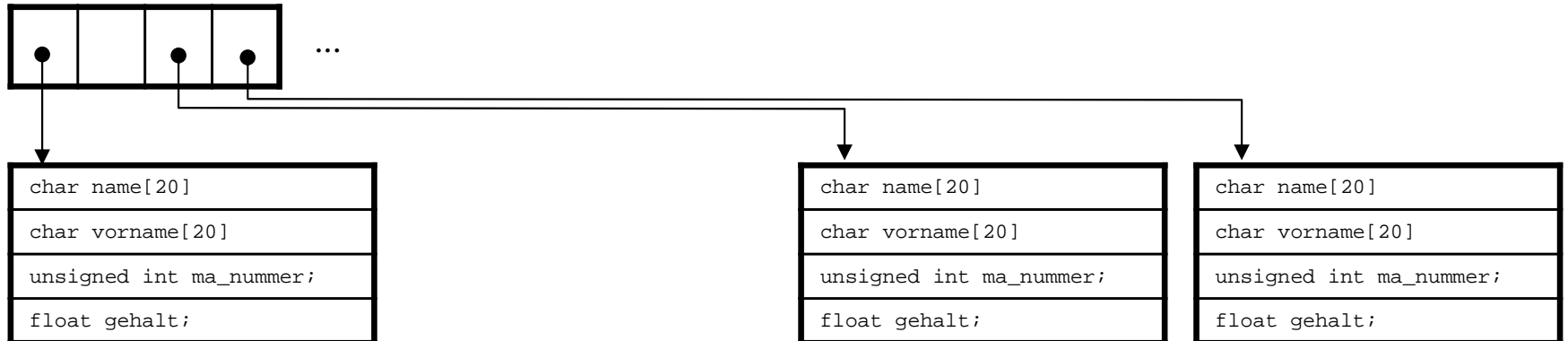
## Statisches Array

```
MITARBEITER ma_datenbank[200];
```



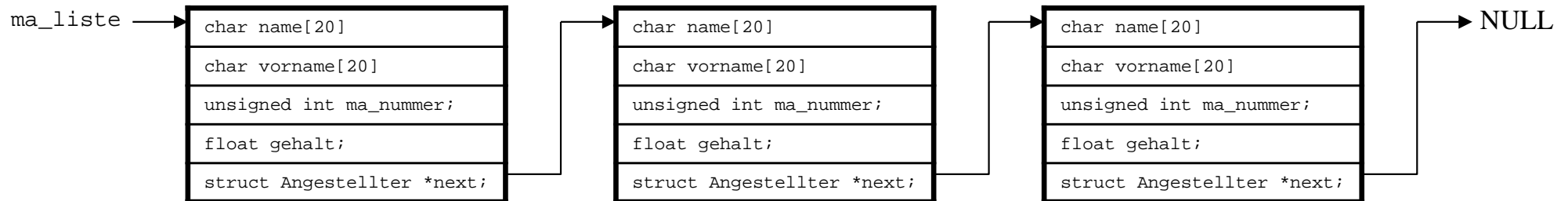
## Zeiger-Array

```
MITARBEITER *ma_datenbank[200];
```



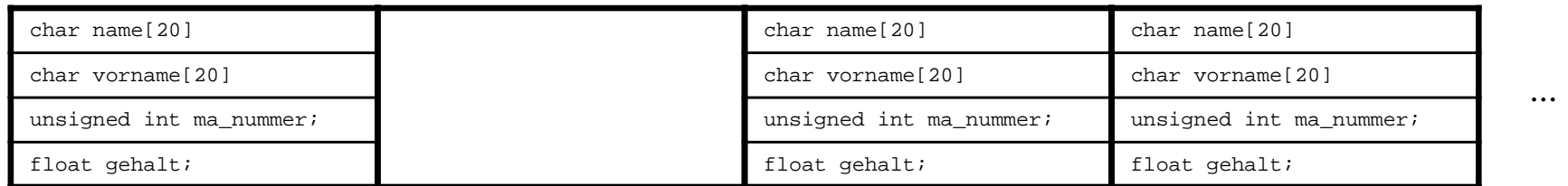
## Verkettete Liste

```
MITARBEITER *ma_liste;
```



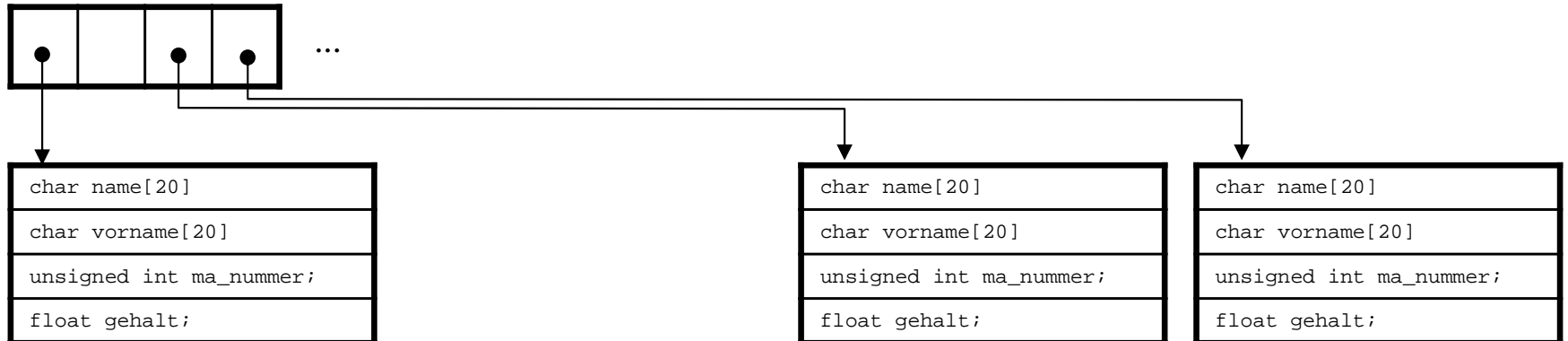
## Statisches Array

```
MITARBEITER ma_datenbank[200];
```



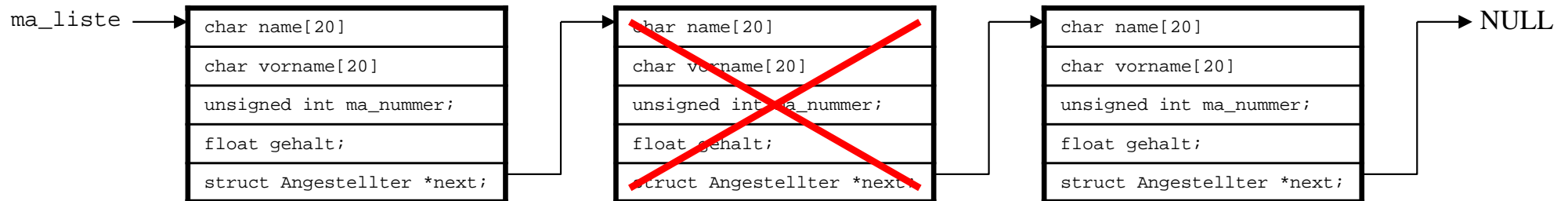
## Zeiger-Array

```
MITARBEITER *ma_datenbank[200];
```



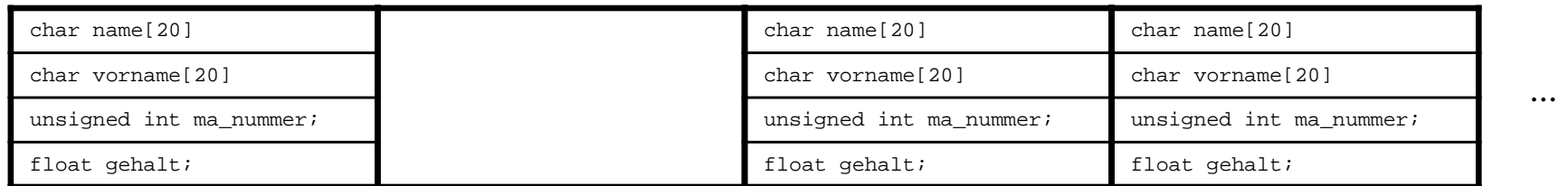
## Verkettete Liste

```
MITARBEITER *ma_liste;
```



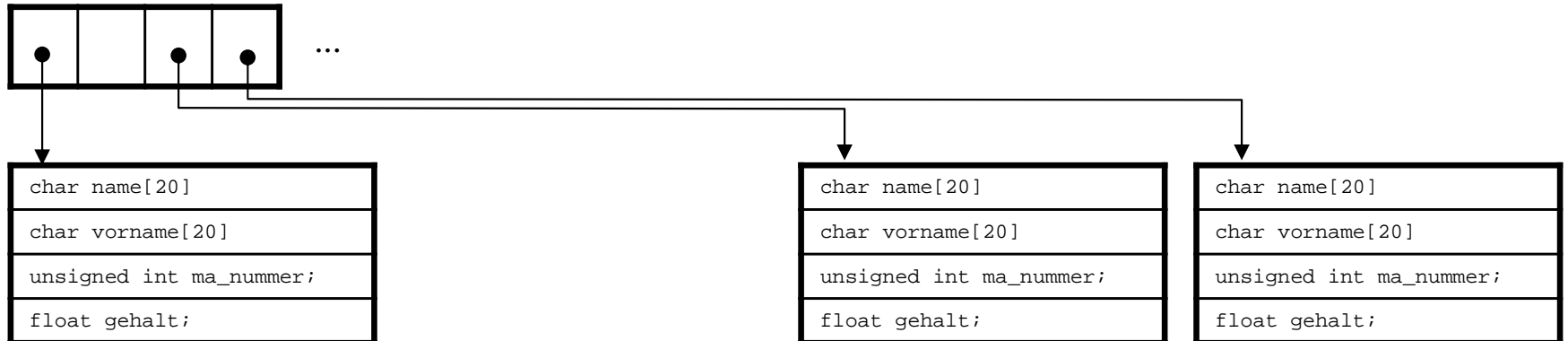
## Statisches Array

```
MITARBEITER ma_datenbank[200];
```



## Zeiger-Array

```
MITARBEITER *ma_datenbank[200];
```



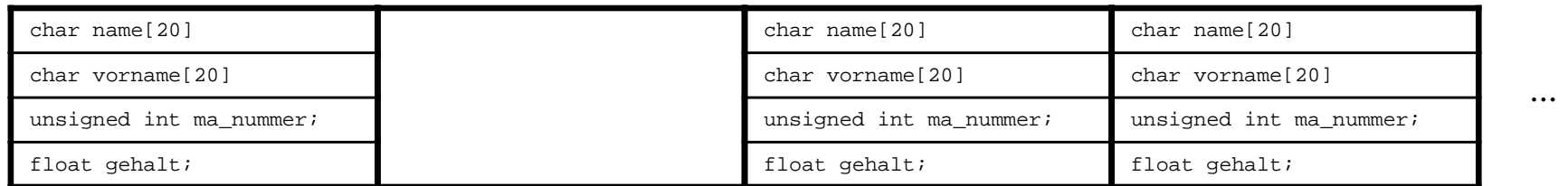
## Verkettete Liste

```
MITARBEITER *ma_liste;
```



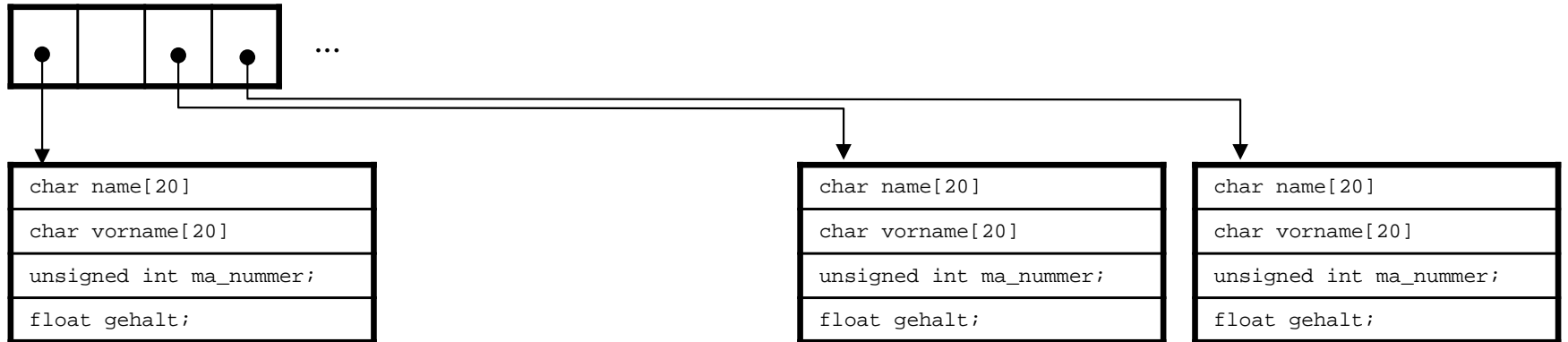
## Statisches Array

```
MITARBEITER ma_datenbank[200];
```



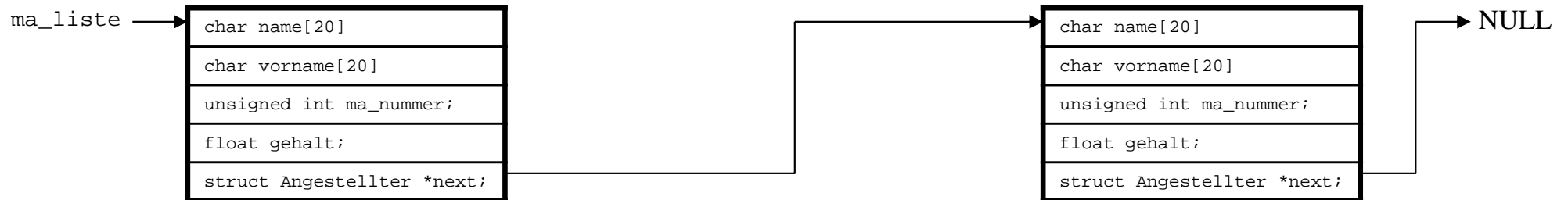
## Zeiger-Array

```
MITARBEITER *ma_datenbank[200];
```



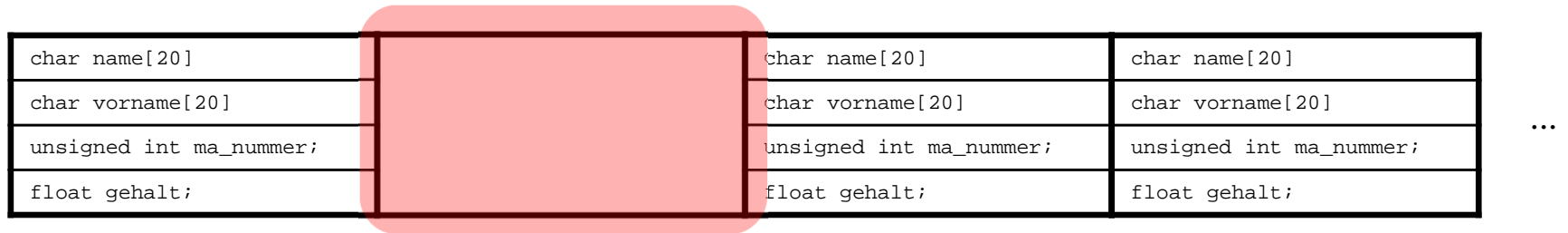
## Verkettete Liste

```
MITARBEITER *ma_liste;
```



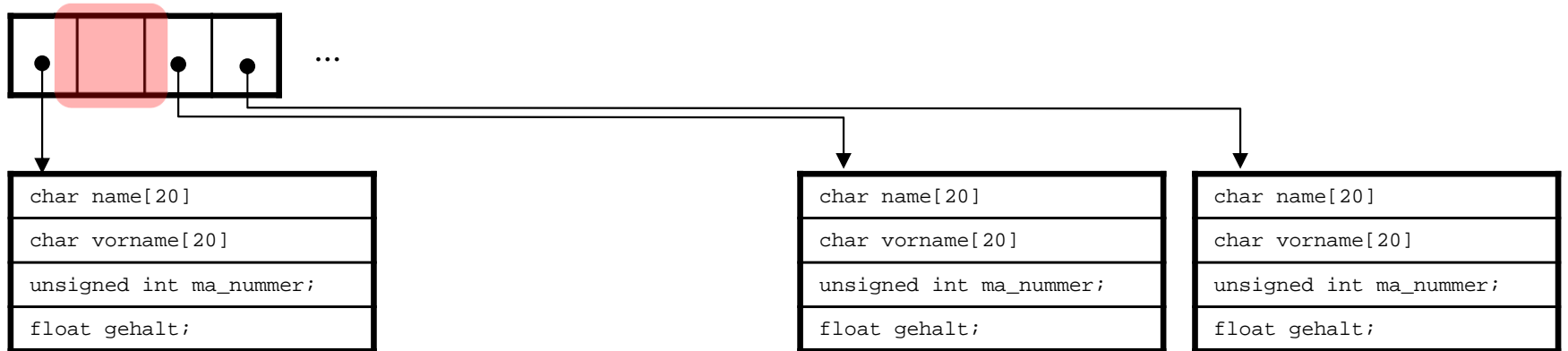
## Statisches Array

```
MITARBEITER ma_datenbank[200];
```



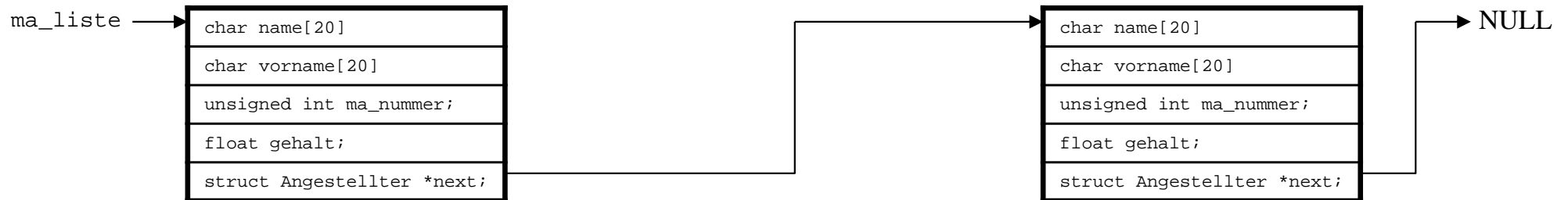
## Zeiger-Array

```
MITARBEITER *ma_datenbank[200];
```

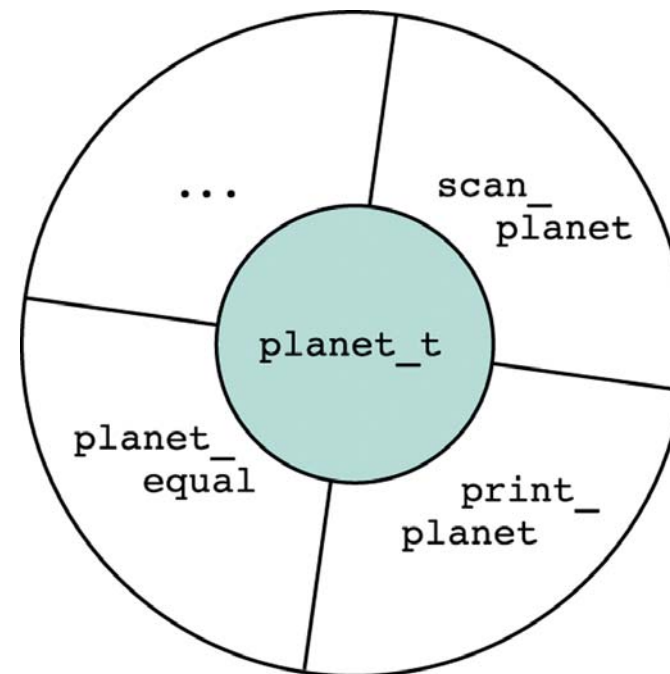


## Verkettete Liste

```
MITARBEITER *ma_liste;
```



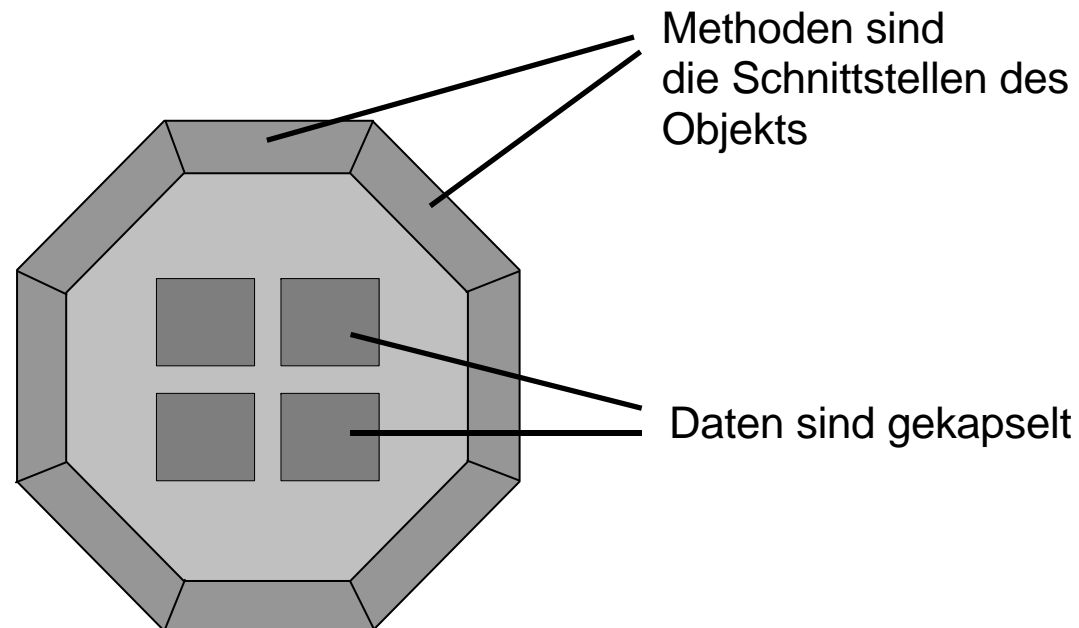
# Datenabstraktion



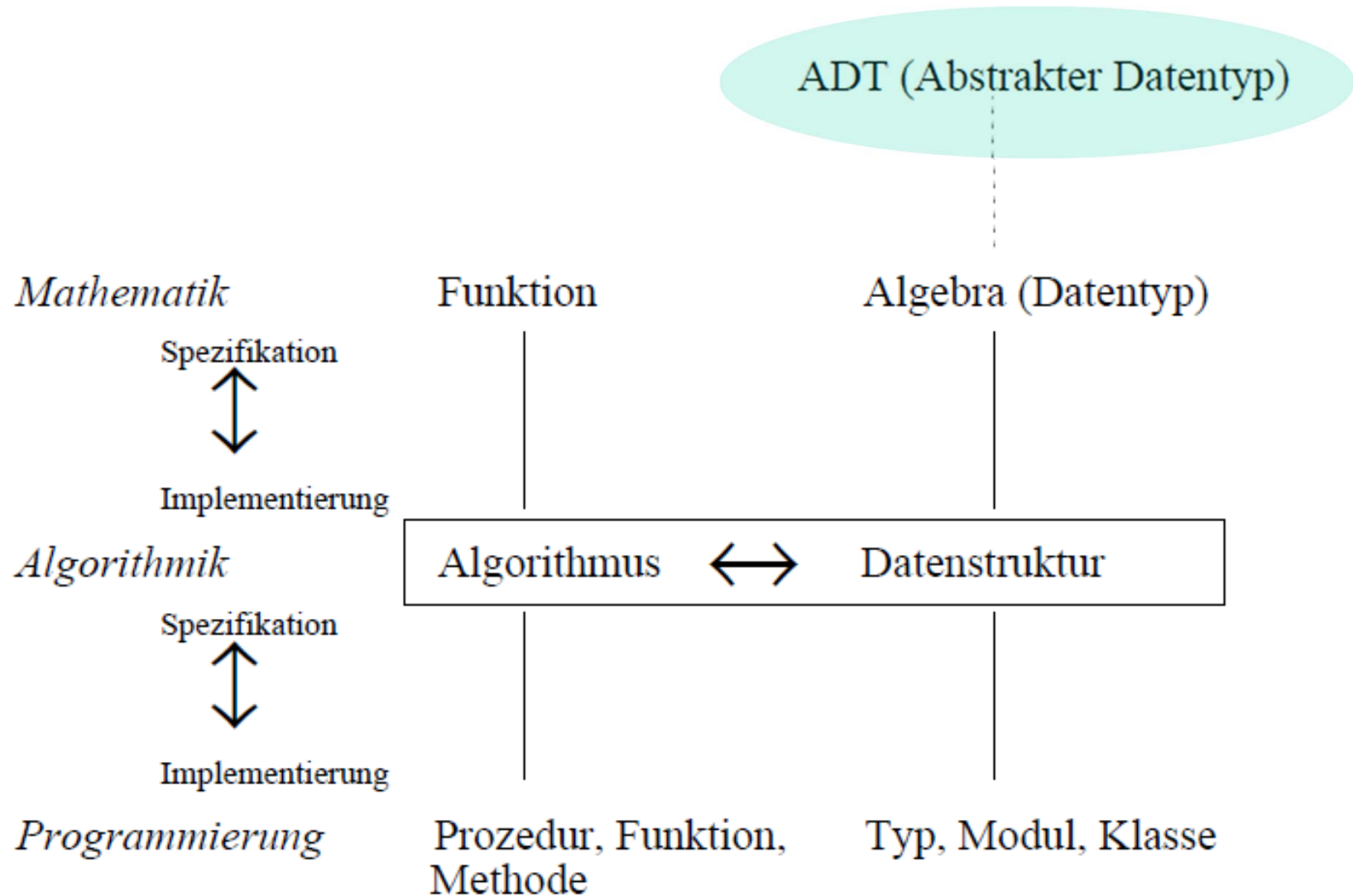


## ■ Datenabstraktion

- Im Gegensatz zu Datentypen und Datenstrukturen, die eher physikalisch definiert werden, gibt man bei einem **Abstrakten Datentyp** nur die Operationen an, die mit ihm ausgeführt werden können sollen.
- **Abstrakter Datentyp (ADT)**
  - Spezifikation durch seine (öffentlich bekannten) Operationen
  - Interne Darstellung und Implementierung bleiben verborgen



## ■ Abstraktionsebenen

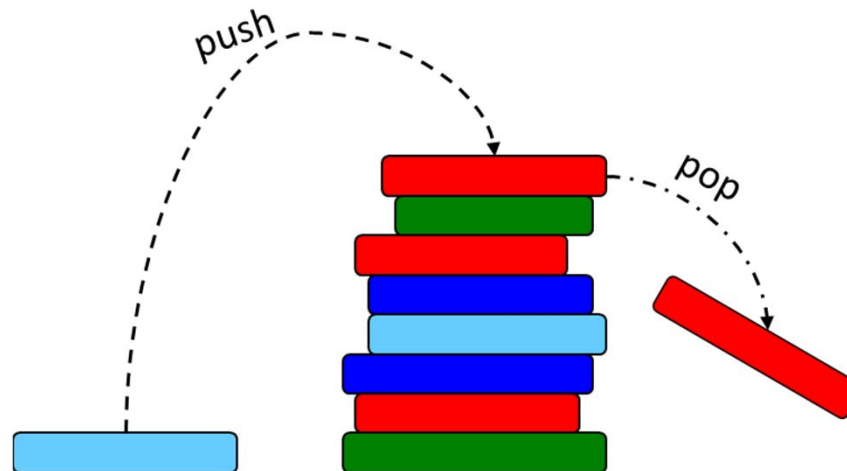


# Stack



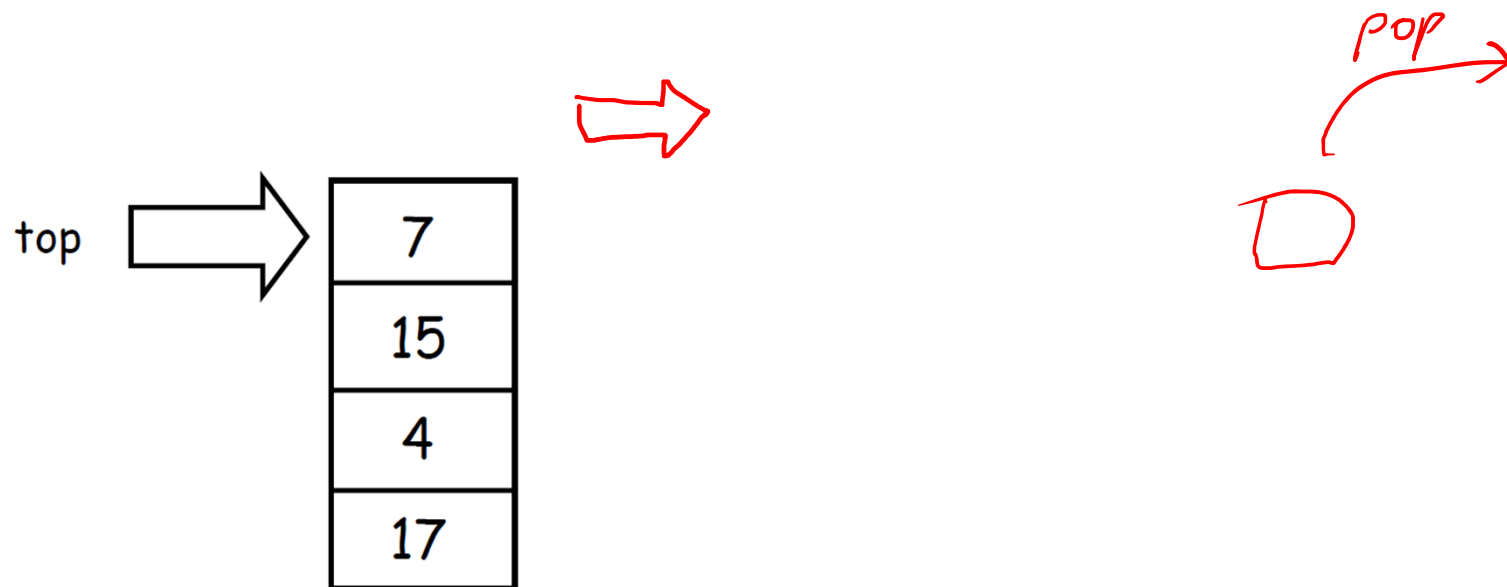
## ■ Stack (Stapel, Keller)

- Daten-Pufferung nach dem **LIFO-Prinzip**
- Anwendung: Parameterablage bei Funktionsaufruf, Syntaxprüfung, etc.
- Operationen:    `push()`    fügt ein Element ein  
                  `pop()`     entfernt ein Element

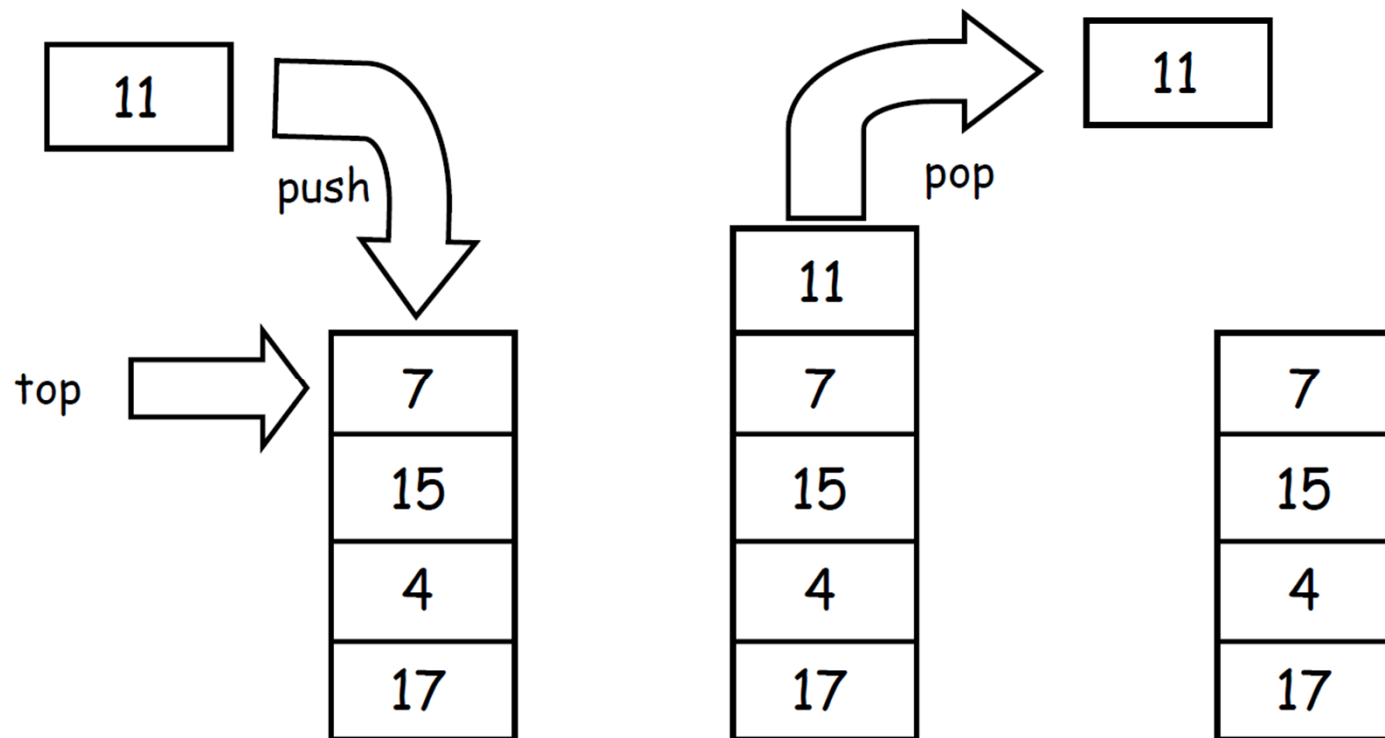


ggf. weitere: `init()`, `top()`, `empty()`

## ■ Stack (Stapel, Keller)



## ■ Stack (Stapel, Keller)



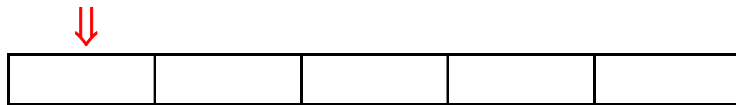
## ■ Stack: Beispiel-Implementierung

```
int stack[5], index = 0;
```

```
push(10);           ← 10  
push(14);          ← 14  
pop();             ⇒ 14  
pop();             ⇒ 10
```

## ■ Stack: Beispiel-Implementierung *für den Datentyp int*

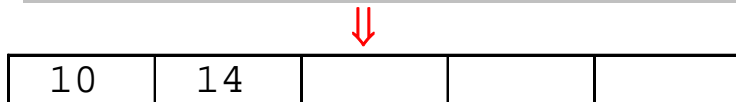
```
int stack[5], index = 0;
```



```
push(10);       $\Leftarrow 10$ 
```



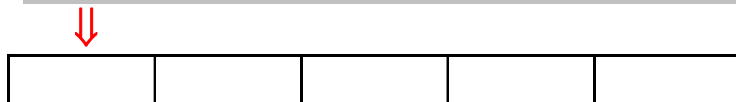
```
push(14);       $\Leftarrow 14$ 
```



```
pop();       $\Rightarrow 14$ 
```

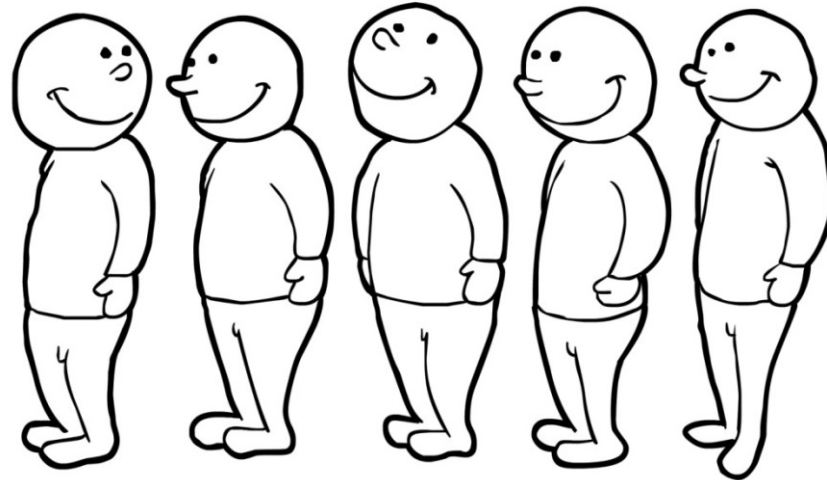


```
pop();       $\Rightarrow 10$ 
```



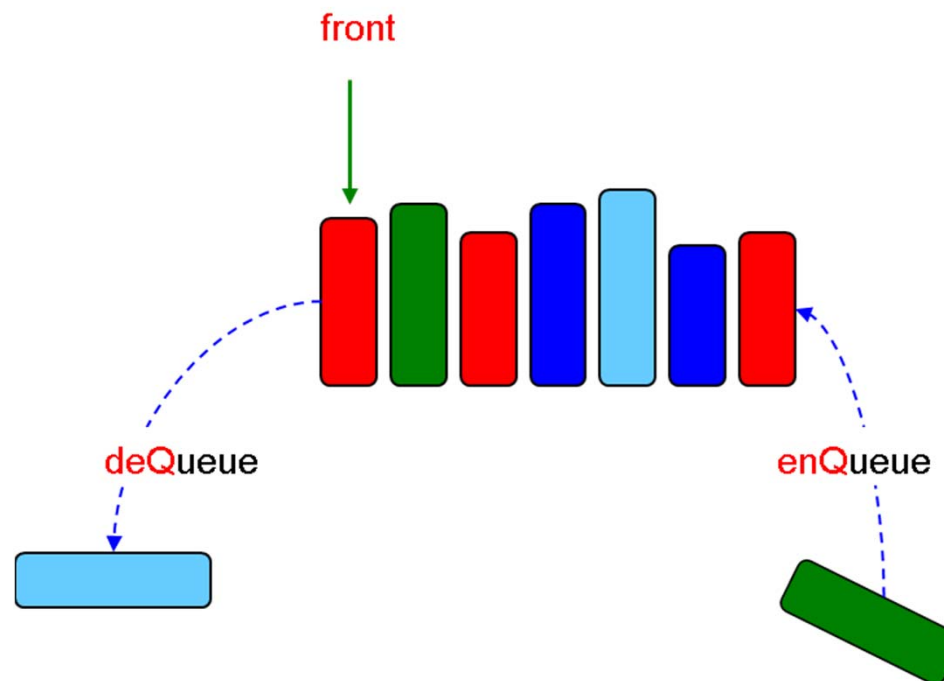


# Queue

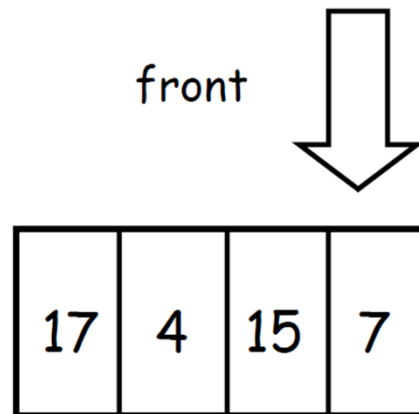


## ■ Queue (Reihe, Schlange)

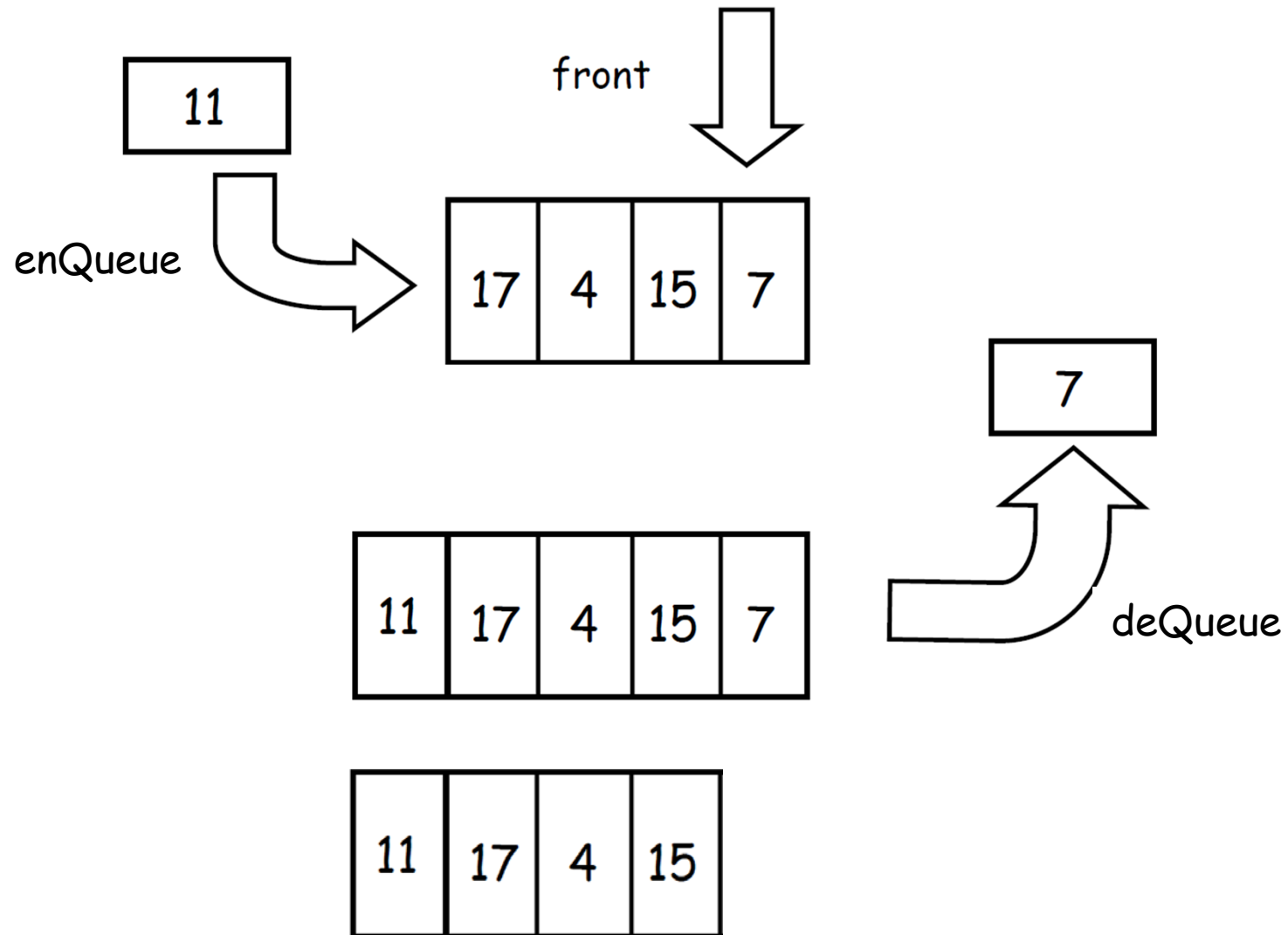
- Daten-Pufferung nach dem **FIFO-Prinzip**
- Bsp.: Ein-/Ausgabepuffer, Druckerwarteschlange
- Operationen: `enqueue()` fügt ein Element ein  
`dequeue()` entfernt ein Element



## ■ Queue (Reihe, Schlange)



## ■ Queue (Reihe, Schlange)

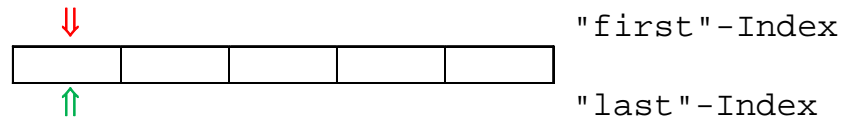


## ■ Queue: Beispiel-Implementierung

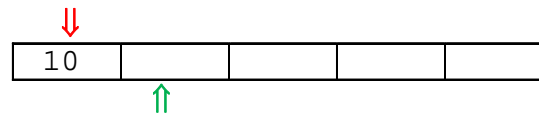
```
int queue[5], first = 0, last = 0;  
enqueue(10);           ← 10  
enqueue(14);          ← 14  
enqueue(7);           ← 7  
dequeue();            ⇒ 10  
dequeue();            ⇒ 14
```

## Queue: Beispiel-Implementierung

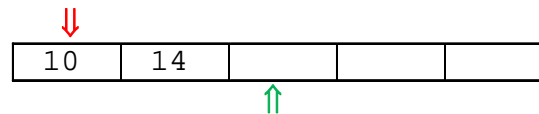
```
int queue[5], first = 0, last = 0;
```



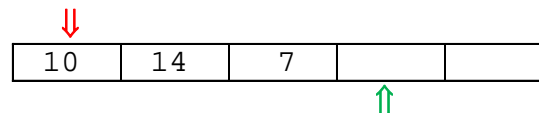
```
enqueue(10); ← 10
```



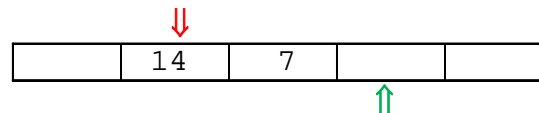
```
enqueue(14); ← 14
```



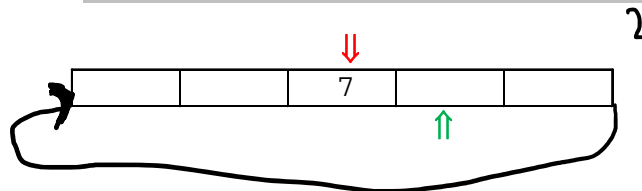
```
enqueue(7); ← 7
```



```
dequeue(); ⇒ 10
```



```
dequeue(); ⇒ 14
```




## Verkettete Liste



## ■ Verkettete Liste

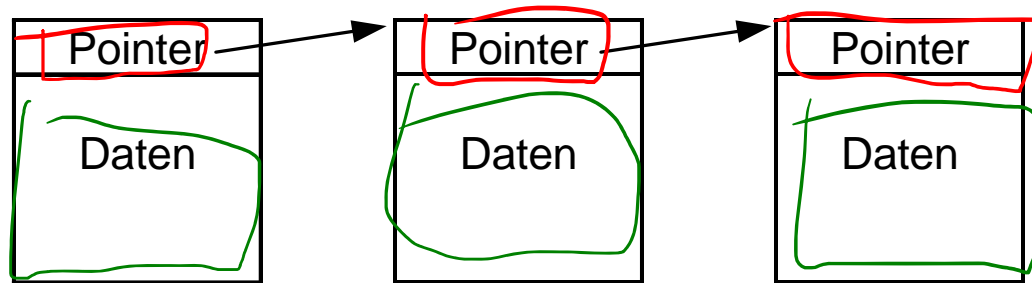
- dynamische Struktur
- gleiche Listenelemente, beinhalten Verweis auf das nächste Element
- Datentyp eines Listenelements

```
struct element  
{  
    float f;   
    struct element *next;  
};
```

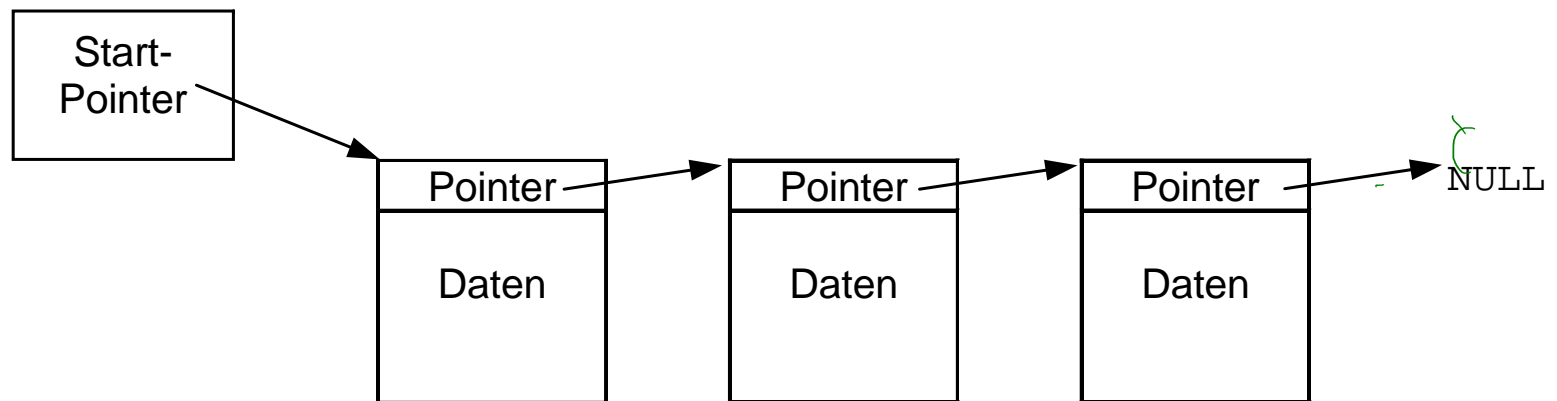
- Operationen:
  - Liste initialisieren
  - Neues Listenelement einfügen
  - Listenelement löschen
  - Suchen eines bestimmten Listenelements



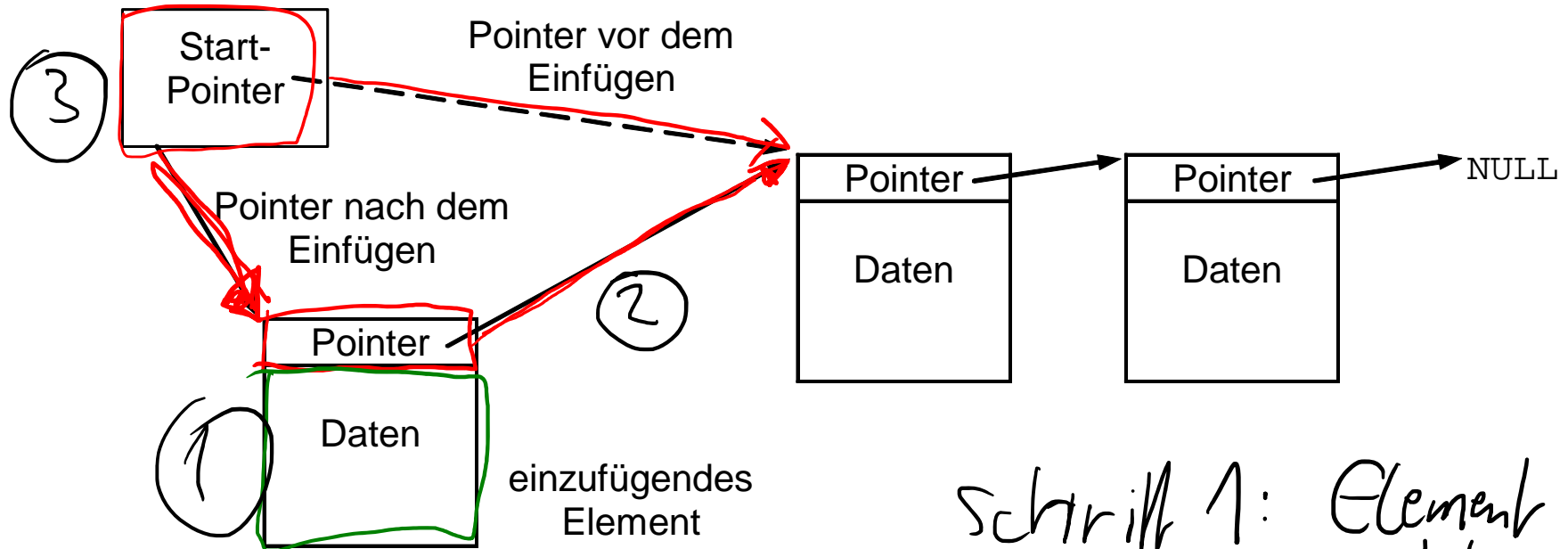
## ■ Einfach verkettete Liste



- mit Start-Pointer und Ende



## ■ Implementierung: Neues Listenelement vorn einfügen



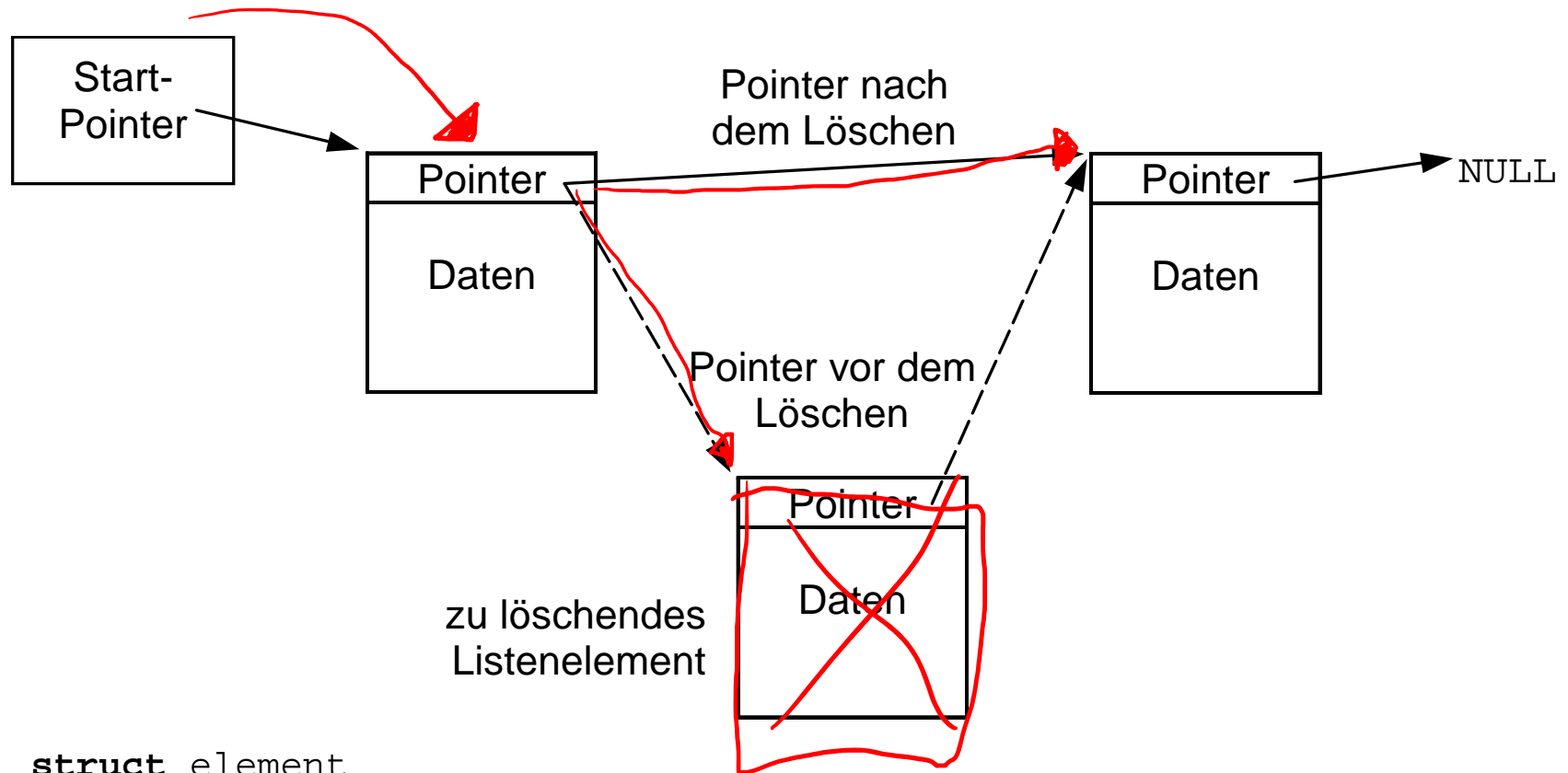
```

struct element
{
    float f;
    struct element *next;
};

struct element *start, *temp;
    
```

Schritt 1: Element erstellen/ anlegen  
 Schritt 2: Pointer auf erstes Element setzen  
 Schritt 3: Start-Pointer auf neues Element setzen

## ■ Implementierung: Listenelement löschen



```

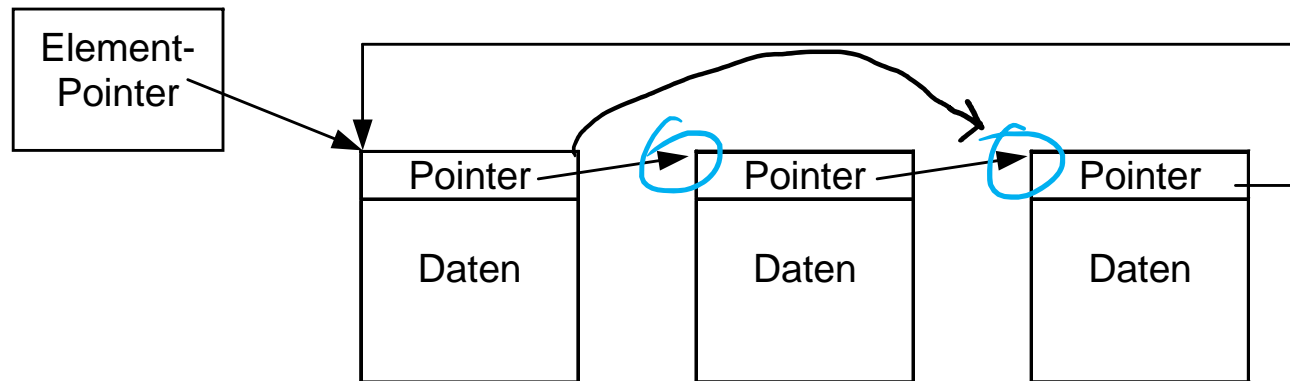
struct element
{
    float f;
    struct element *next;
};
    
```

```

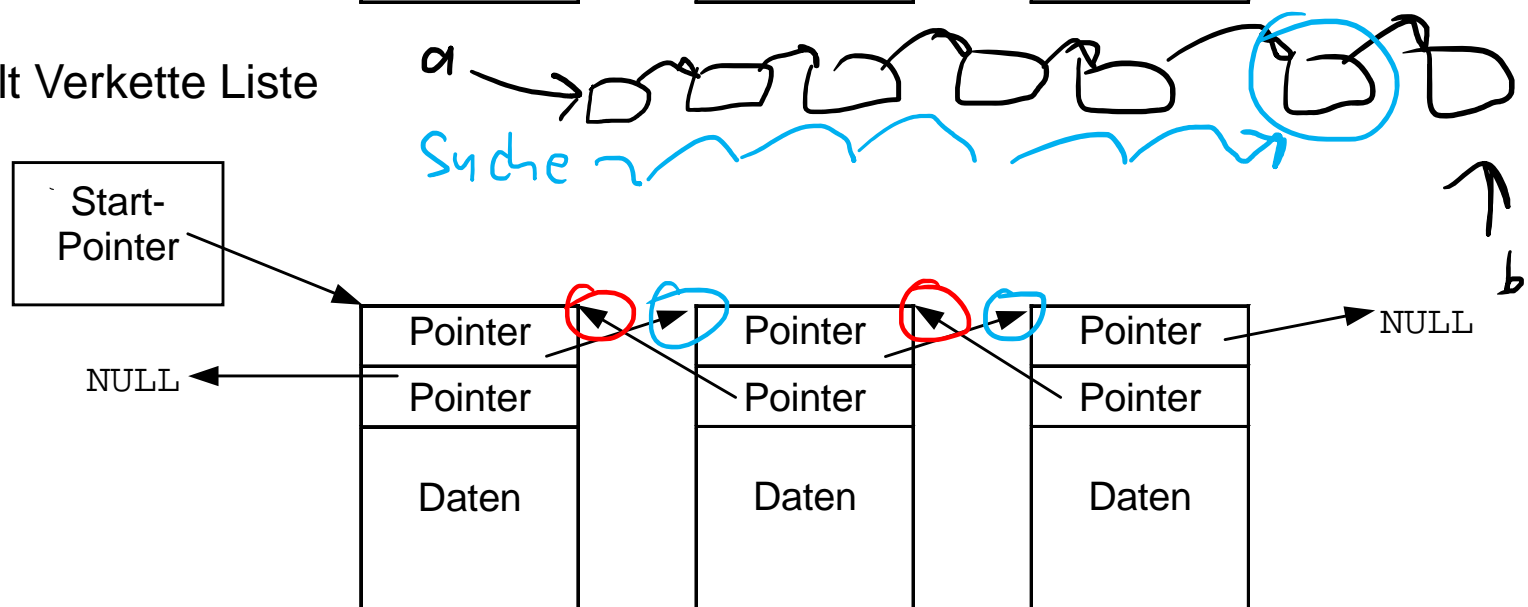
struct element *start, *temp; *aktuell, *ende;
    
```

## ■ Varianten verketteter Listen

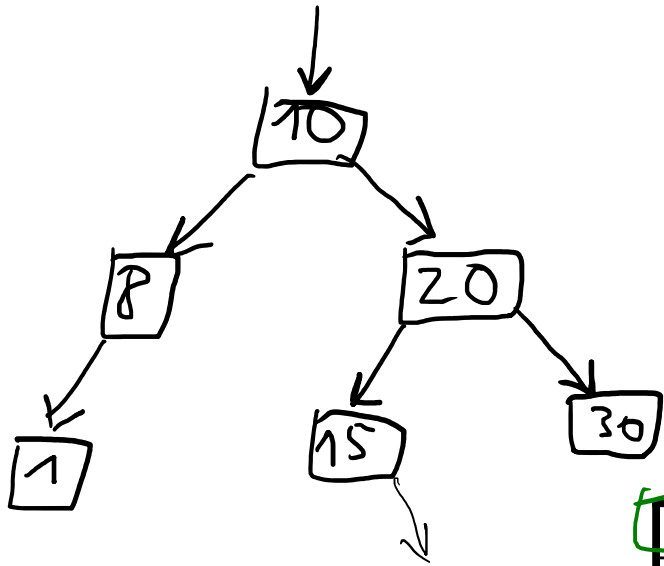
- Ringpuffer



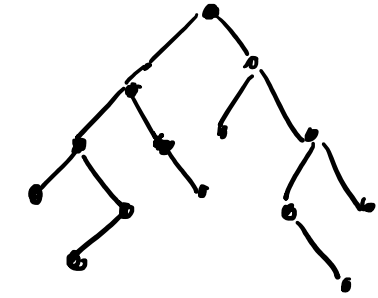
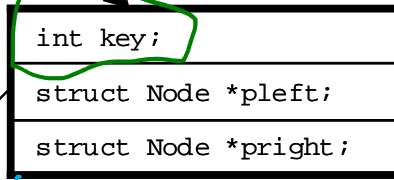
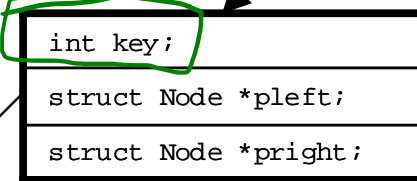
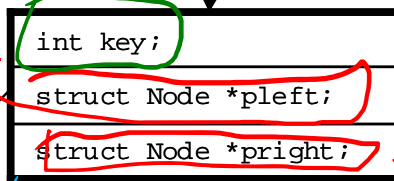
- Doppelt Verkettete Liste



# Binärer Suchbaum



struct Node \*proot;



NULL

NULL

NULL

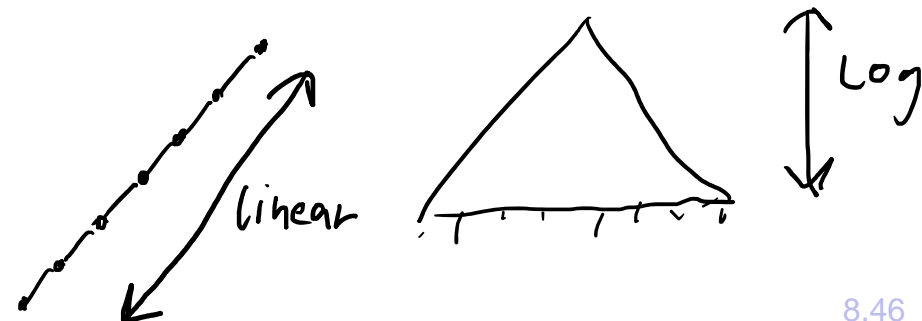
NULL

## ■ Effizienz: Binärbaum vs. Verkettete Liste

Effizienz beim Suchen von Elementen:

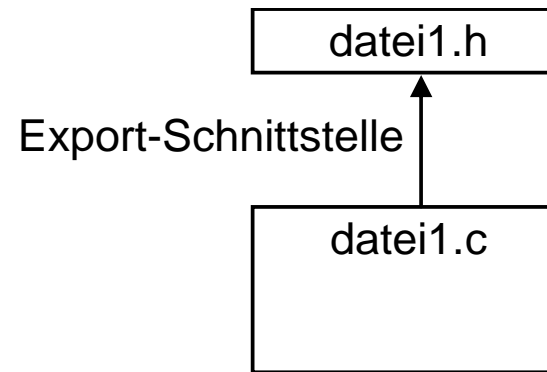
∅ Anzahl der Vergleiche: verkettete Liste ↔ ausgeglichener Binärbaum

Anzahl der Elemente	# Vergleiche Liste	# Vergleiche Baum
1	1	2
3	2	4
15	8	8
63	32	12
255	128	16
511	256	18
...	...	...
1048575	524288	40



## ■ Implementierung eines ADT

- Allgemeines Schema



- Am Beispiel des ADT "Stack"

