

# Übung zu Betriebssystembau

## Ereignisbearbeitung und Synchronisation

07. Januar 2025

---

**Alexander Krause**

Arbeitsgruppe Systemsoftware  
Technische Universität Dortmund

(Mit Material vom Lehrstuhl 4 der FAU)

Wie sieht es mit gegenseitigem Ausschluss auf Fadenebene in STUBS aus?



Wie sieht es mit gegenseitigem Ausschluss auf Fadenebene in STUBS aus?

Wir haben doch bereits ein `spinlock` implementiert...



# Mutex mit aktivem Warten

```
mutex.lock()  
// code  
mutex.unlock()
```



## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()
```



# Mutex mit aktivem Warten

**Beispiel:** Drei Threads auf einer CPU

**App1**

```
mutex.lock()  
// code  
mutex.unlock()
```

**App2**

```
mutex.lock()  
// code  
mutex.unlock()
```

**App3**

```
mutex.lock()  
// code  
mutex.unlock()
```

active



ready list

App1 App2 App3



# Mutex mit aktivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()
```

active

App1

ready list

App2 App3



# Mutex mit aktivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()
```

active

App1

ready list

App2 App3

CPU-Zeit →



# Mutex mit aktivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()
```

active

App1

ready list

App2 App3

CPU-Zeit →



# Mutex mit aktivem Warten

Beispiel: Drei Threads auf einer CPU

App1

```
mutex.lock()
```

```
// code
```

```
mutex.unlock() ⚡
```

App2

```
mutex.lock()
```

```
// code
```

```
mutex.unlock()
```

App3

```
mutex.lock()
```

```
// code
```

```
mutex.unlock()
```

active

App1

ready list

App2 App3

CPU-Zeit →



# Mutex mit aktivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()
```

active



ready list



CPU-Zeit →



# Mutex mit aktivem Warten

**Beispiel:** Drei Threads auf einer CPU

**App1**

```
mutex.lock()  
// code  
mutex.unlock()
```

**App2**

```
mutex.lock()  
// code  
mutex.unlock()
```

**App3**

```
mutex.lock()  
// code  
mutex.unlock()
```

active

App2

ready list

App3 App1

CPU-Zeit →



# Mutex mit aktivem Warten

**Beispiel:** Drei Threads auf einer CPU

**App1**

```
mutex.lock()  
// code  
mutex.unlock()
```

**App2**

```
mutex.lock()  
// code  
mutex.unlock()
```

**App3**

```
mutex.lock()  
// code  
mutex.unlock()
```

active

App2

ready list

App3 App1

CPU-Zeit →



# Mutex mit aktivem Warten

**Beispiel:** Drei Threads auf einer CPU

**App1**

```
mutex.lock()  
// code  
mutex.unlock()
```

**App2**

```
mutex.lock()  
// code  
mutex.unlock()
```

**App3**

```
mutex.lock()  
// code  
mutex.unlock()
```

active

App2

ready list

App3 App1

CPU-Zeit →



# Mutex mit aktivem Warten

**Beispiel:** Drei Threads auf einer CPU

**App1**

```
mutex.lock()  
// code  
mutex.unlock()
```

**App2**

```
mutex.lock() ⚡  
// code  
mutex.unlock()
```

**App3**

```
mutex.lock()  
// code  
mutex.unlock()
```

active

App2

ready list

App3 App1

CPU-Zeit →



# Mutex mit aktivem Warten

Beispiel: Drei Threads auf einer CPU

App1

```
mutex.lock()  
// code  
mutex.unlock()
```

App2

```
mutex.lock()  
// code  
mutex.unlock()
```

App3

```
mutex.lock()  
// code  
mutex.unlock()
```

active



ready list



CPU-Zeit →



# Mutex mit aktivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()
```

active

App3

ready list

App1 App2

CPU-Zeit →



# Mutex mit aktivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()
```

active

App3

ready list

App1 App2

CPU-Zeit →



# Mutex mit aktivem Warten

**Beispiel:** Drei Threads auf einer CPU

**App1**

```
mutex.lock()  
// code  
mutex.unlock()
```

**App2**

```
mutex.lock()  
// code  
mutex.unlock()
```

**App3**

```
mutex.lock()  
// code  
mutex.unlock()
```

active

App3

ready list

App1 App2

CPU-Zeit →



# Mutex mit aktivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()
```

### App3

```
mutex.lock() ⚡  
// code  
mutex.unlock()
```

active

App3

ready list

App1 App2

CPU-Zeit →



# Mutex mit aktivem Warten

**Beispiel:** Drei Threads auf einer CPU

**App1**

```
mutex.lock()  
// code  
mutex.unlock()
```

**App2**

```
mutex.lock()  
// code  
mutex.unlock()
```

**App3**

```
mutex.lock()  
// code  
mutex.unlock()
```

active



ready list



CPU-Zeit →



# Mutex mit aktivem Warten

**Beispiel:** Drei Threads auf einer CPU

**App1**

```
mutex.lock()  
// code  
mutex.unlock()
```

**App2**

```
mutex.lock()  
// code  
mutex.unlock()
```

**App3**

```
mutex.lock()  
// code  
mutex.unlock()
```

active

App1

ready list

App2 App3

CPU-Zeit →



# Mutex mit aktivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()
```

active

App1

ready list

App2 App3

CPU-Zeit →



# Mutex mit aktivem Warten

Beispiel: Drei Threads auf einer CPU

App1

```
mutex.lock()  
// code  
mutex.unlock()
```

App2

```
mutex.lock()  
// code  
mutex.unlock()
```

App3

```
mutex.lock()  
// code  
mutex.unlock()
```

active

App1

ready list

App2 App3

CPU-Zeit →



Verschwendung von CPU-Zeit

# Mutex mit harter Synchronisation

- Analog zur Interruptsperre mit `cli`
- **Ziel:** Kein (präemptives) Scheduling



# Mutex mit harter Synchronisation

- Analog zur Interruptsperre mit `cli`
- **Ziel:** Kein (präemptives) Scheduling
- Realisierbar durch
  - Multitasking (temporär) deaktivieren
  - Erweiterung des Schedulers
  - Wechsel auf Epilogebe



- Analog zur Interruptsperre mit `cli`
- **Ziel:** Kein (präemptives) Scheduling
- Realisierbar durch
  - Multitasking (temporär) deaktivieren
  - Erweiterung des Schedulers
  - Wechsel auf Epilogebe
- **Vorteile:**
  - konsistent
  - (relativ) einfach zu implementieren



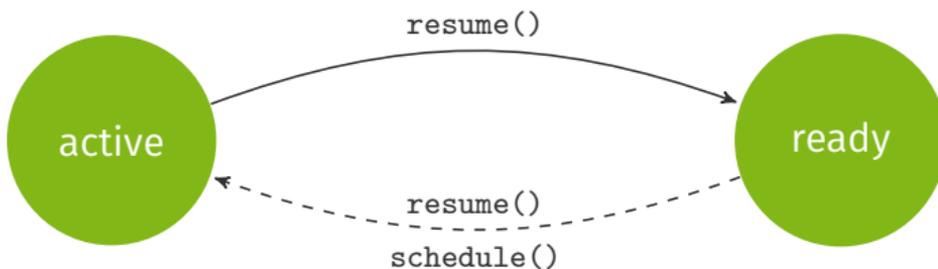
- Analog zur Interruptsperre mit `cli`
- **Ziel:** Kein (präemptives) Scheduling
- Realisierbar durch
  - Multitasking (temporär) deaktivieren
  - Erweiterung des Schedulers
  - Wechsel auf Epilogebe
- **Vorteile:**
  - konsistent
  - (relativ) einfach zu implementieren
- **Nachteile:**
  - Breitbandwirkung
  - Prioritätsverletzung
  - Prophylaktisch



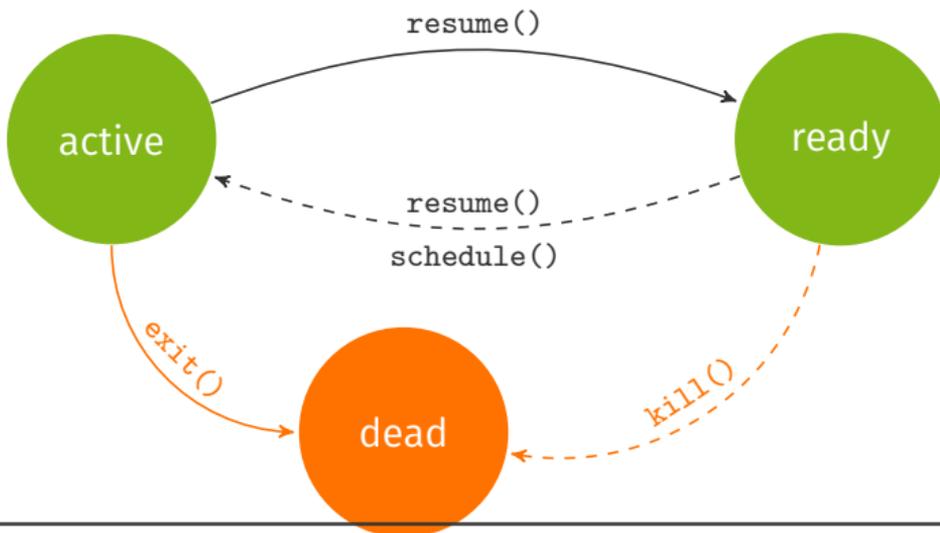
**Ansatz:** Fäden, die den kritischen Abschnitt nicht betreten können, werden blockiert (d.h. von der CPU-Zuteilung ausgeschlossen)



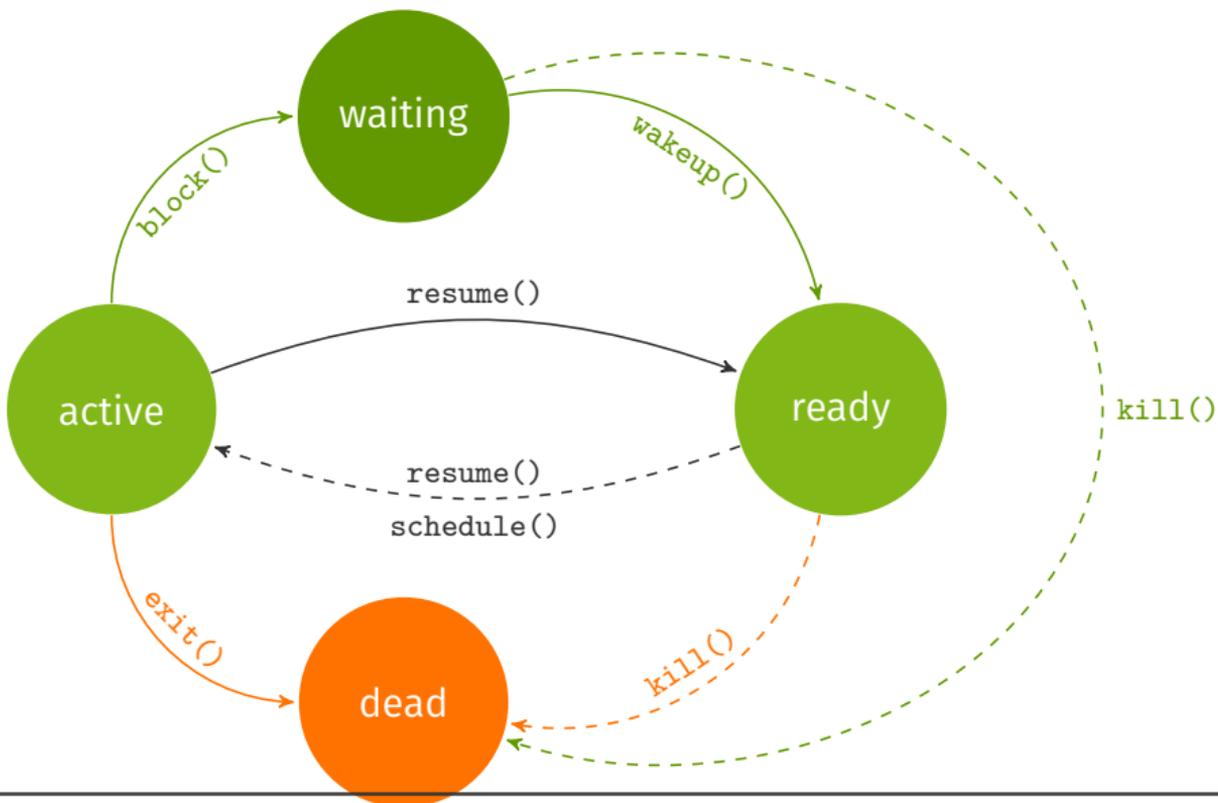
**Ansatz:** Fäden, die den kritischen Abschnitt nicht betreten können, werden blockiert (d.h. von der CPU-Zuteilung ausgeschlossen)



**Ansatz:** Fäden, die den kritischen Abschnitt nicht betreten können, werden blockiert (d.h. von der CPU-Zuteilung ausgeschlossen)



# Idee: Passives Warten





waiting

Einführung eines **waiting rooms**  
(Liste mit wartenden Threads)



# Mutex mit passivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active



ready list

App1 App2 App3

waiting room



# Mutex mit passivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active

App1

ready list

App2 App3

waiting room



# Mutex mit passivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active

App1

ready list

App2 App3

waiting room

CPU-Zeit →



# Mutex mit passivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active

App1

ready list

App2 App3

waiting room

CPU-Zeit →



# Mutex mit passivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock() ⚡  
// mehr code
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active

App1

ready list

App2 App3

waiting room

CPU-Zeit →



# Mutex mit passivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active



ready list



waiting room



CPU-Zeit →



# Mutex mit passivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active

App2

ready list

App3 App1

waiting room

CPU-Zeit →



# Mutex mit passivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active

App2

ready list

App3 App1

waiting room

CPU-Zeit →



# Mutex mit passivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active



ready list



waiting room



CPU-Zeit →



# Mutex mit passivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active

App3

ready list

App1

waiting room

App2

CPU-Zeit →



# Mutex mit passivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active

App3

ready list

App1

waiting room

App2

CPU-Zeit →



# Mutex mit passivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active



ready list



waiting room



CPU-Zeit →



# Mutex mit passivem Warten

Beispiel: Drei Threads auf einer CPU

**App1**

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

**App2**

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

**App3**

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active

App1

ready list



waiting room

App2 App3

CPU-Zeit →



# Mutex mit passivem Warten

**Beispiel:** Drei Threads auf einer CPU

**App1**

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

**App2**

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

**App3**

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active

App1

ready list

waiting room

App2 App3

CPU-Zeit →



# Mutex mit passivem Warten

Beispiel: Drei Threads auf einer CPU

App1

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active

App1

ready list

App2

waiting room

App3

CPU-Zeit →



# Mutex mit passivem Warten

Beispiel: Drei Threads auf einer CPU

**App1**

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

**App2**

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

**App3**

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active

App1

ready list

App2

waiting room

App3

CPU-Zeit →



# Mutex mit passivem Warten

Beispiel: Drei Threads auf einer CPU

App1

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```



active

App1

ready list

App2

waiting room

App3

CPU-Zeit →



# Mutex mit passivem Warten

## Beispiel: Drei Threads auf einer CPU

### App1

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

### App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active



ready list



waiting room



CPU-Zeit →



# Mutex mit passivem Warten

Beispiel: Drei Threads auf einer CPU

App1

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active

App2

ready list

App1

waiting room

App3

CPU-Zeit →



# Mutex mit passivem Warten

Beispiel: Drei Threads auf einer CPU

App1

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active

App2

ready list

App1

waiting room

App3

CPU-Zeit →



# Mutex mit passivem Warten

**Beispiel:** Drei Threads auf einer CPU

**App1**

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

**App2**

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

**App3**

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active

App2

ready list

App1

waiting room

App3

CPU-Zeit →



# Mutex mit passivem Warten

Beispiel: Drei Threads auf einer CPU

App1

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

App2

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

App3

```
mutex.lock()  
// code  
mutex.unlock()  
// mehr code
```

active

App2

ready list

App1 App3

waiting room

CPU-Zeit →



## sem·a·phore

1. any apparatus for signaling, as by an arrangement of lights, flags, and mechanical arms on railroads
2. a system for signaling by the use of two flags, one held in each hand: the letters of the alphabet are represented by the various positions of the arms
3. any system of signaling by semaphore

*nach V. E. Neufeld, Webster's New World Dictionary, Simon & Schuster Inc., third college edition, 1988*



**init()** Zähler  $c$  mit positivem Wert initialisieren

**P()** von *Prolaag* (versuchen zu verringern)  
bzw. *Passeering*<sup>†</sup> (passieren)

$c > 0$  dekrementieren

$c = 0$  warten

**V()** von *Verhoog* (erhöhen) bzw. *Vrijgave*<sup>†</sup> (freigeben)

- nächsten wartenden Faden aufwecken oder
- Zähler  $c$  erhöhen

<sup>†</sup> nach Edsger W. Dijkstra, Over de sequentialiteit van procesbeschrijvingen, ca. 1962



```
1 Semaphore mutex(1);
2
3 void func() {
4
5     mutex.p(); // lock
6
7     // critical section
8
9     mutex.v(); // unlock
10
11 }
```



```
1 Semaphore mutex(1);
2
3 void func() {
4
5     mutex.p(); // lock
6
7     // critical section
8
9     mutex.v(); // unlock
10
11 }
```

```
1 Semaphore empty(size);
2 Semaphore full(0);
3
4 void producer(){
5     empty.p();
6     // produce
7     full.v();
8 }
9
10 void consumer(){
11     full.p();
12     // consume
13     empty.v();
14 }
```



```
1 App::action(){  
2     foo();  
3     sleep(13);  
4     bar();  
5 }
```

- ähnlich der Funktion `sleep(3)`
- jedoch mit Wartezeit in Millisekunden (statt Sekunden)
- analog zu Wartezimmer den Thread aus Ready-Liste des Schedulers nehmen



# Schlafen legen

```
1 App::action(){  
2     foo();  
3     sleep(13);  
4     bar();  
5 }
```

- ähnlich der Funktion `sleep(3)`
- jedoch mit Wartezeit in Millisekunden (statt Sekunden)
- analog zu Wartezimmer den Thread aus Ready-Liste des Schedulers nehmen  
`waiting room (13ms)`

App



# Schlafen legen

```
1 App::action(){
2     foo();
3     sleep(13);
4     bar();
5 }
```

- ähnlich der Funktion `sleep(3)`
- jedoch mit Wartezeit in Millisekunden (statt Sekunden)
- analog zu Wartezimmer den Thread aus Ready-Liste des Schedulers nehmen



*(alternative Darstellung)*



# Schlafen legen

```
1 App::action(){  
2     foo();  
3     sleep(13);  
4     bar();  
5 }
```

- ähnlich der Funktion `sleep(3)`
- jedoch mit Wartezeit in Millisekunden (statt Sekunden)
- analog zu Wartezimmer den Thread aus Ready-Liste des Schedulers nehmen



- mit jedem Tick die Wartezeit dekrementieren



# Schlafen legen

```
1 App::action(){  
2     foo();  
3     sleep(13);  
4     bar();  
5 }
```

- ähnlich der Funktion `sleep(3)`
- jedoch mit Wartezeit in Millisekunden (statt Sekunden)
- analog zu Wartezimmer den Thread aus Ready-Liste des Schedulers nehmen



- mit jedem Tick die Wartezeit dekrementieren •



# Schlafen legen

```
1 App::action(){  
2     foo();  
3     sleep(13);  
4     bar();  
5 }
```

- ähnlich der Funktion `sleep(3)`
- jedoch mit Wartezeit in Millisekunden (statt Sekunden)
- analog zu Wartezimmer den Thread aus Ready-Liste des Schedulers nehmen



- mit jedem Tick die Wartezeit dekrementieren ●●



# Schlafen legen

```
1 App::action(){  
2     foo();  
3     sleep(13);  
4     bar();  
5 }
```

- ähnlich der Funktion `sleep(3)`
- jedoch mit Wartezeit in Millisekunden (statt Sekunden)
- analog zu Wartezimmer den Thread aus Ready-Liste des Schedulers nehmen



- mit jedem Tick die Wartezeit dekrementieren ●●●



# Schlafen legen

```
1 App::action(){
2     foo();
3     sleep(13);
4     bar();
5 }
```

- ähnlich der Funktion `sleep(3)`
- jedoch mit Wartezeit in Millisekunden (statt Sekunden)
- analog zu Wartezimmer den Thread aus Ready-Liste des Schedulers nehmen



- mit jedem Tick die Wartezeit dekrementieren



```
1 App::action(){  
2     foo();  
3     sleep(13);  
4     bar();  
5 }
```

- ähnlich der Funktion `sleep(3)`
- jedoch mit Wartezeit in Millisekunden (statt Sekunden)
- analog zu Wartezimmer den Thread aus Ready-Liste des Schedulers nehmen



- mit jedem Tick die Wartezeit dekrementieren
- nach Ablauf der Wartezeit Thread wieder im Scheduler einreihen



**Beispiel:** Drei Anwendungen legen sich nacheinander schlafen



**Beispiel:** Drei Anwendungen legen sich nacheinander schlafen

1. Thread **foo** für 666ms



**Beispiel:** Drei Anwendungen legen sich nacheinander schlafen

1. Thread **foo** für 666ms



**Beispiel:** Drei Anwendungen legen sich nacheinander schlafen

1. Thread **foo** für 666ms
2. Thread **bar** für 23ms



**Beispiel:** Drei Anwendungen legen sich nacheinander schlafen

1. Thread **foo** für 666ms
2. Thread **bar** für 23ms



**Beispiel:** Drei Anwendungen legen sich nacheinander schlafen

1. Thread **foo** für 666ms
2. Thread **bar** für 23ms
3. Thread **baz** für 42ms



**Beispiel:** Drei Anwendungen legen sich nacheinander schlafen

1. Thread **foo** für 666ms
2. Thread **bar** für 23ms
3. Thread **baz** für 42ms



**Beispiel:** Drei Anwendungen legen sich nacheinander schlafen

1. Thread **foo** für 666ms
2. Thread **bar** für 23ms
3. Thread **baz** für 42ms

Verwaltung mittels verketteter Liste



**Beispiel:** Drei Anwendungen legen sich nacheinander schlafen

1. Thread **foo** für 666ms
2. Thread **bar** für 23ms
3. Thread **baz** für 42ms

Verwaltung mittels verketteter Liste



**Beispiel:** Drei Anwendungen legen sich nacheinander schlafen

1. Thread **foo** für 666ms
2. Thread **bar** für 23ms
3. Thread **baz** für 42ms

Verwaltung mittels verketteter Liste hat den Nachteil, dass bei jedem Tick die gesamte Liste durchlaufen werden muss



**Beispiel:** Drei Anwendungen legen sich nacheinander schlafen

1. Thread **foo** für 666ms
2. Thread **bar** für 23ms
3. Thread **baz** für 42ms

Verwaltung mittels verketteter Liste hat den Nachteil, dass bei jedem Tick die gesamte Liste durchlaufen werden muss •



**Beispiel:** Drei Anwendungen legen sich nacheinander schlafen

1. Thread **foo** für 666ms
2. Thread **bar** für 23ms
3. Thread **baz** für 42ms

Verwaltung mittels verketteter Liste hat den Nachteil, dass bei jedem Tick die gesamte Liste durchlaufen werden muss ●●



**Beispiel:** Drei Anwendungen legen sich nacheinander schlafen

1. Thread **foo** für 666ms
2. Thread **bar** für 23ms
3. Thread **baz** für 42ms

Verwaltung mittels verketteter Liste hat den Nachteil, dass bei jedem Tick die gesamte Liste durchlaufen werden muss ●●●



**Beispiel:** Drei Anwendungen legen sich nacheinander schlafen

1. Thread **foo** für 666ms
2. Thread **bar** für 23ms
3. Thread **baz** für 42ms

Verwaltung mittels verketteter Liste hat den Nachteil, dass bei jedem Tick die gesamte Liste durchlaufen werden muss

( $\mathcal{O}(n)$ , und das  $1000\times$  pro Sekunde in der Epilogebe)



**Beispiel:** Drei Anwendungen legen sich nacheinander schlafen

1. Thread **foo** für 666ms
2. Thread **bar** für 23ms
3. Thread **baz** für 42ms

Verwaltung mittels verketteter Liste hat den Nachteil, dass bei jedem Tick die gesamte Liste durchlaufen werden muss

( $\mathcal{O}(n)$ , und das  $1000\times$  pro Sekunde in der Epilogebe)



*Das muss besser gehen!*



- Einführung einer absoluten Zeit



- Einführung einer absoluten Zeit

**Beispiel:** absolute Zeit  $T = 1334\text{ms}$



- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert

**Beispiel:** absolute Zeit  $T = 1334\text{ms}$



- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert •

**Beispiel:** absolute Zeit  $T = 1335\text{ms}$



- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert ●●

**Beispiel:** absolute Zeit  $T = 1336\text{ms}$



- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert ●●●

**Beispiel:** absolute Zeit  $T = 1337\text{ms}$



- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert
- Berechnung der Endzeit beim Einfügen neuer Threads

**Beispiel:** absolute Zeit  $T = 1337\text{ms}$



- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert
- Berechnung der Endzeit beim Einfügen neuer Threads

**Beispiel:** absolute Zeit  $T = 1337\text{ms}$

1. Thread **foo**:  $666\text{ms} + 1337\text{ms}$



- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert
- Berechnung der Endzeit beim Einfügen neuer Threads

**Beispiel:** absolute Zeit  $T = 1337\text{ms}$

1. Thread **foo**:  $666\text{ms} + 1337\text{ms}$



- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert
- Berechnung der Endzeit beim Einfügen neuer Threads

**Beispiel:** absolute Zeit  $T = 1337\text{ms}$

1. Thread **foo**:  $666\text{ms} + 1337\text{ms}$
2. Thread **bar**:  $23\text{ms} + 1337\text{ms}$



- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert
- Berechnung der Endzeit beim Einfügen neuer Threads

**Beispiel:** absolute Zeit  $T = 1337\text{ms}$

1. Thread **foo**:  $666\text{ms} + 1337\text{ms}$
2. Thread **bar**:  $23\text{ms} + 1337\text{ms}$



- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert
- Berechnung der Endzeit beim Einfügen neuer Threads
- Einordnen in einer Vorrangwarteschlange ( $\mathcal{O}(n)$ )

**Beispiel:** absolute Zeit  $T = 1337\text{ms}$

1. Thread **foo**:  $666\text{ms} + 1337\text{ms}$
2. Thread **bar**:  $23\text{ms} + 1337\text{ms}$



- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert
- Berechnung der Endzeit beim Einfügen neuer Threads
- Einordnen in einer Vorrangwarteschlange ( $\mathcal{O}(n)$ )

**Beispiel:** absolute Zeit  $T = 1337\text{ms}$

1. Thread **foo**:  $666\text{ms} + 1337\text{ms}$
2. Thread **bar**:  $23\text{ms} + 1337\text{ms}$



- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert
- Berechnung der Endzeit beim Einfügen neuer Threads
- Einordnen in einer Vorrangwarteschlange ( $\mathcal{O}(n)$ )

**Beispiel:** absolute Zeit  $T = 1337\text{ms}$

1. Thread **foo**:  $666\text{ms} + 1337\text{ms}$
2. Thread **bar**:  $23\text{ms} + 1337\text{ms}$
3. Thread **baz**:  $42\text{ms} + 1337\text{ms}$



- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert
- Berechnung der Endzeit beim Einfügen neuer Threads
- Einordnen in einer Vorrangwarteschlange ( $\mathcal{O}(n)$ )

**Beispiel:** absolute Zeit  $T = 1337\text{ms}$

1. Thread **foo**:  $666\text{ms} + 1337\text{ms}$
2. Thread **bar**:  $23\text{ms} + 1337\text{ms}$
3. Thread **baz**:  $42\text{ms} + 1337\text{ms}$



- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert
- Berechnung der Endzeit beim Einfügen neuer Threads
- Einordnen in einer Vorrangwarteschlange ( $\mathcal{O}(n)$ )

**Beispiel:** absolute Zeit  $T = 1337\text{ms}$

1. Thread **foo**:  $666\text{ms} + 1337\text{ms}$
2. Thread **bar**:  $23\text{ms} + 1337\text{ms}$
3. Thread **baz**:  $42\text{ms} + 1337\text{ms}$



## Alternative Variante

- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert
- Berechnung der Endzeit beim Einfügen neuer Threads
- Einordnen in einer Vorrangwarteschlange ( $\mathcal{O}(n)$ )
- Wenn die aktuelle Zeit  $T$  dem ersten Element entspricht: Thread wieder dem Scheduler übergeben ( $\mathcal{O}(1)$ )

**Beispiel:** absolute Zeit  $T = 1337\text{ms}$

1. Thread **foo**:  $666\text{ms} + 1337\text{ms}$
2. Thread **bar**:  $23\text{ms} + 1337\text{ms}$
3. Thread **baz**:  $42\text{ms} + 1337\text{ms}$



- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert
- Berechnung der Endzeit beim Einfügen neuer Threads
- Einordnen in einer Vorrangwarteschlange ( $\mathcal{O}(n)$ )
- Wenn die aktuelle Zeit  $T$  dem ersten Element entspricht: Thread wieder dem Scheduler übergeben ( $\mathcal{O}(1)$ )

**Beispiel:** absolute Zeit  $T = 1360\text{ms}$

1. Thread **foo**:  $666\text{ms} + 1337\text{ms}$
2. Thread **bar**:  $23\text{ms} + 1337\text{ms}$
3. Thread **baz**:  $42\text{ms} + 1337\text{ms}$



- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert
- Berechnung der Endzeit beim Einfügen neuer Threads
- Einordnen in einer Vorrangwarteschlange ( $\mathcal{O}(n)$ )
- Wenn die aktuelle Zeit  $T$  dem ersten Element entspricht: Thread wieder dem Scheduler übergeben ( $\mathcal{O}(1)$ )

### Nachteile

- Absolute Zeit ist ein neuer Zustand
- Bei 32bit Überlauf *möglich* (nach 49.7 Tagen)



- Einführung einer absoluten Zeit
  - wird mit jedem Tick inkrementiert
- Berechnung der Endzeit beim Einfügen neuer Threads
- Einordnen in einer Vorrangwarteschlange ( $\mathcal{O}(n)$ )
- Wenn die aktuelle Zeit  $T$  dem ersten Element entspricht: Thread wieder dem Scheduler übergeben ( $\mathcal{O}(1)$ )

### Nachteile

- Absolute Zeit ist ein neuer Zustand
- Bei 32bit Überlauf *möglich* (nach 49.7 Tagen)

*Geht das nicht effizienter (ohne solche Probleme)?*



## Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit



## Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert (negative Zeitdifferenzen sind nicht erlaubt!)



## Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert

### Beispiel:

1. Thread **foo**: 666ms



## Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert

### Beispiel:

1. Thread **foo**: 666ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$

## Beispiel:

1. Thread **foo**: 666ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$

## Beispiel:

1. Thread **foo**:  $666\text{ms} - t_0 = 666\text{ms}$



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$

## Beispiel:

1. Thread **foo**: 666ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$

## Beispiel:

1. Thread **foo**: 666ms
2. Thread **bar**: 23ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$

## Beispiel:

1. Thread **foo**: 666ms
2. Thread **bar**: 23ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )

## Beispiel:

1. Thread **foo**: 666ms
2. Thread **bar**: 23ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )

## Beispiel:

1. Thread **foo**: 666ms
2. Thread **bar**: 23ms -  $t_0 = 23\text{ms}$



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )

## Beispiel:

1. Thread **foo**: 666ms
2. Thread **bar**: 23ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
  - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )

## Beispiel:

1. Thread **foo**: 666ms
2. Thread **bar**: 23ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
  - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )

## Beispiel:

1. Thread **foo**:  $666\text{ms} - t_{\text{bar}} = 643\text{ms}$
2. Thread **bar**:  $23\text{ms}$



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
  - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )

## Beispiel:

1. Thread **foo**: 643ms
2. Thread **bar**: 23ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
  - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )

## Beispiel:

1. Thread **foo**: 643ms
2. Thread **bar**: 23ms
3. Thread **baz**: 42ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
  - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )

## Beispiel:

1. Thread **foo**: 643ms
2. Thread **bar**: 23ms
3. Thread **baz**: 42ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
  - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )

## Beispiel:

1. Thread **foo**: 643ms
2. Thread **bar**: 23ms
3. Thread **baz**: 42ms -  $t_0 = 42\text{ms}$



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
  - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )

## Beispiel:

1. Thread **foo**: 643ms
2. Thread **bar**: 23ms
3. Thread **baz**:  $42\text{ms} - t_0 = 42\text{ms} > t_{\text{bar}}$



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
  - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )

## Beispiel:

1. Thread **foo**: 643ms
2. Thread **bar**: 23ms
3. Thread **baz**:  $42\text{ms} - t_0 - t_{\text{bar}} = 19\text{ms}$



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
  - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )

## Beispiel:

1. Thread **foo**: 643ms
2. Thread **bar**: 23ms
3. Thread **baz**: 19ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
    - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
    - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
  - Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
    - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )
- ⇒ keine Vorrangwarteschlange!

## Beispiel:

1. Thread **foo**: 643ms
2. Thread **bar**: 23ms
3. Thread **baz**: 19ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
    - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
    - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
  - Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
    - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )
- ⇒ keine Vorrangwarteschlange!

## Beispiel:

1. Thread **foo**:  $643\text{ms} - t_{\text{baz}} = 624\text{ms}$
2. Thread **bar**:  $23\text{ms}$
3. Thread **baz**:  $19\text{ms}$



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
    - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
    - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
  - Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
    - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )
- ⇒ keine Vorrangwarteschlange!

## Beispiel:

1. Thread **foo**: 624ms
2. Thread **bar**: 23ms
3. Thread **baz**: 19ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
  - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )
  - ⇒ keine Vorrangwarteschlange!
- Erstes Element wird mit jedem Tick dekrementiert

## Beispiel:

1. Thread **foo**: 624ms
2. Thread **bar**: 23ms
3. Thread **baz**: 19ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
  - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )

⇒ keine Vorrangwarteschlange!
- Erstes Element wird mit jedem Tick dekrementiert ●

## Beispiel:

1. Thread **foo**: 624ms
2. Thread **bar**: 23ms
3. Thread **baz**: 19ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
  - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )

⇒ keine Vorrangwarteschlange!
- Erstes Element wird mit jedem Tick dekrementiert ●●

## Beispiel:

1. Thread **foo**: 624ms
2. Thread **bar**: 23ms
3. Thread **baz**: 19ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
  - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )

⇒ keine Vorrangwarteschlange!
- Erstes Element wird mit jedem Tick dekrementiert ●●●

## Beispiel:

1. Thread **foo**: 624ms
2. Thread **bar**: 23ms
3. Thread **baz**: 19ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
  - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )

⇒ keine Vorrangwarteschlange!
- Erstes Element wird mit jedem Tick dekrementiert und bei 0 dem Scheduler übergeben

## Beispiel:

1. Thread **foo**: 624ms
2. Thread **bar**: 23ms
3. Thread **baz**: 19ms



# Effiziente Variante (Optional!)

- Verwendung der relativen Delta-Zeit
  - Es wird nur die Zeitdifferenz zum Vorgänger gespeichert
  - Vorgänger des ersten Elements hat Zeit  $t_0 = 0\text{ms}$
- Neue Threads nach Schlafdauer einordnen ( $\mathcal{O}(n)$ )
  - Nachfolgendes Element muss angepasst werden ( $\mathcal{O}(1)$ )

⇒ keine Vorrangwarteschlange!
- Erstes Element wird mit jedem Tick dekrementiert und bei 0 dem Scheduler übergeben ( $\mathcal{O}(1)$ )

## Beispiel:

1. Thread **foo**: 624ms
2. Thread **bar**: 23ms
3. Thread **baz**: 19ms



- Implementierung von Semaphoren mit passivem Warten
  - Verwendung in der neuen `getKey()` Funktion
- Zeitgesteuertes Schlafen der Threads
- Leerlauf des Prozessor (falls keine Threads vorhanden)
- Kapselung in Systemaufrufchnittstellen (`syscall`)  
welche sich um den Wechsel in die Epilogebebene kümmern



- jede Semaphore ist gleichzeitig ein Wartezimmer



- jede Semaphore ist gleichzeitig ein Wartezimmer
- und jeder Wecker (`be11`) ebenfalls
  - auch wenn bei unseren Weckern nur je ein Faden darin weilt
  - einfache Implementierung, da wir beim Wecker die neuen Ablaufplanmethoden für die Semaphoren verwenden können.  
*Wirkt aber auf den ersten Blick halt komisch.*



- jede Semaphore ist gleichzeitig ein Wartezimmer
- und jeder Wecker (`be11`) ebenfalls
  - auch wenn bei unseren Weckern nur je ein Faden darin weilt
  - einfache Implementierung, da wir beim Wecker die neuen Ablaufplanmethoden für die Semaphoren verwenden können.  
*Wirkt aber auf den ersten Blick halt komisch.*
- Umsetzung durch Ableitung von `waitingroom`



- jede Semaphore ist gleichzeitig ein Wartezimmer
- und jeder Wecker (`bell`) ebenfalls
  - auch wenn bei unseren Weckern nur je ein Faden darin weilt
  - einfache Implementierung, da wir beim Wecker die neuen Ablaufplanmethoden für die Semaphoren verwenden können.  
*Wirkt aber auf den ersten Blick halt komisch.*
- Umsetzung durch Ableitung von `waitingroom`
- Alle aktiven Wecker werden selbst wieder in einer verketteten Liste (vom `bellringer`) verwaltet



Der `bellringer` prüft regelmäßig die Wecker

- unter Verwendung des LAPIC Timers
- welcher mit `windup(1000)` auf Millisekundentakt gestellt wird
- es reicht, wenn eine CPU das übernimmt



**Problem:** zu wenig Threads bereit



**Problem:** zu wenig Threads bereit

**Lösung:** je ein IdleThread pro CPU

```
1 void IdleThread::action() {  
2     while (true) {  
3         if (!Scheduler::isEmpty())  
4             GuardedScheduler::resume();  
5     }  
6 }
```



**Problem:** zu wenig Threads bereit

**Lösung:** je ein IdleThread pro CPU

```
1 void IdleThread::action() {  
2     while (true) {  
3         if (!Scheduler::isEmpty())  
4             GuardedScheduler::resume();  
5     }  
6 }
```

CPU fungiert effektiv als Heizkörper



**Problem:** zu wenig Threads bereit

**Lösung:** je ein IdleThread pro CPU

```
1 void IdleThread::action() {  
2     while (true) {  
3         if (!Scheduler::isEmpty())  
4             GuardedScheduler::resume();  
5     }  
6 }
```

CPU fungiert effektiv als Heizkörper, besser wäre jedoch ein Schlafzustand  
`Core::idle()` hält CPU bis zum nächsten Interrupt an



`Core::idle()` hält CPU bis zum nächsten Interrupt an



Core::idle() hält CPU bis zum nächsten Interrupt an

```
1 void IdleThread::action() {  
2     while (true) {  
3         if (Scheduler::isEmpty())  
4             Core::idle();  
5         else  
6             GuardedScheduler::resume();  
7     }  
8 }
```



Core::idle() hält CPU bis zum nächsten Interrupt an

```
1 void IdleThread::action() {
2     while (true) {
3         if (Scheduler::isEmpty()) Thread ready
4             Core::idle();
5         else
6             GuardedScheduler::resume();
7     }
8 }
```



Core::idle() hält CPU bis zum nächsten Interrupt an

```
1 void IdleThread::action() {
2     while (true) {
3         if (Scheduler::isEmpty()) Thread ready
4             Core::idle();
5         else
6             GuardedScheduler::resume();
7     }
8 }
```

Durch Aufwachen eines wartenden Threads (oder Neueinplanung bei MPStuBS) kann ein **Lost-Wakeup** passieren!



Core::idle() hält CPU bis zum nächsten Interrupt an  
(mittels atomaren `sti` und `hlt`)

```
1 void IdleThread::action() {
2     while (true) {
3         if (Scheduler::isEmpty()) Thread ready
4             Core::idle();
5         else
6             GuardedScheduler::resume();
7     }
8 }
```

Durch Aufwachen eines wartenden Threads (oder Neueinplanung bei MPSTuBS) kann ein **Lost-Wakeup** passieren!



Core::idle() hält CPU bis zum nächsten Interrupt an  
(mittels atomaren `sti` und `hlt`)

```
1 void IdleThread::action() {
2     while (true) {
3         Core::Interrupt::disable();
4         if (Scheduler::isEmpty())
5             Core::idle();
6         else {
7             Core::Interrupt::enable();
8             GuardedScheduler::resume();
9         }
10    }
11 }
```



## Leerlauf mittels `Core::idle()`

```
1 namespace Core {
2     inline void idle() {
3         asm volatile("sti\n\t"
4                     "hlt\n\t"
5                     ::: "memory");
6     }
7 }
```



**Problem:** Sobald ein Thread bereit ist, soll eine CPU im Leerlauf sofort mit der Abarbeitung beginnen



**Problem:** Sobald ein Thread bereit ist, soll eine CPU im Leerlauf sofort mit der Abarbeitung beginnen

**Lösung:** Aufwecken der CPU mittels IPI



**Weitere Fragen?**

---

**Fast geschafft - dies ist die letzte *Pflicht*aufgabe!**