

# Übung zu Betriebssystembau

## Prolog/Epilog-Modell

---

15. November 2022

Peter Ulbrich & Alexander Lochmann  
(Mit Material vom Lehrstuhl 4 der FAU)

Arbeitsgruppe Systemsoftware  
Technische Universität Dortmund


# Motivation

---

**Interrupts verändern (potenziell) den Zustand des Systems**



# Ohne Synchronisation

`main()`  
  
 $E_0$  (Anwendung)

---

$E_1$  (IRQ)

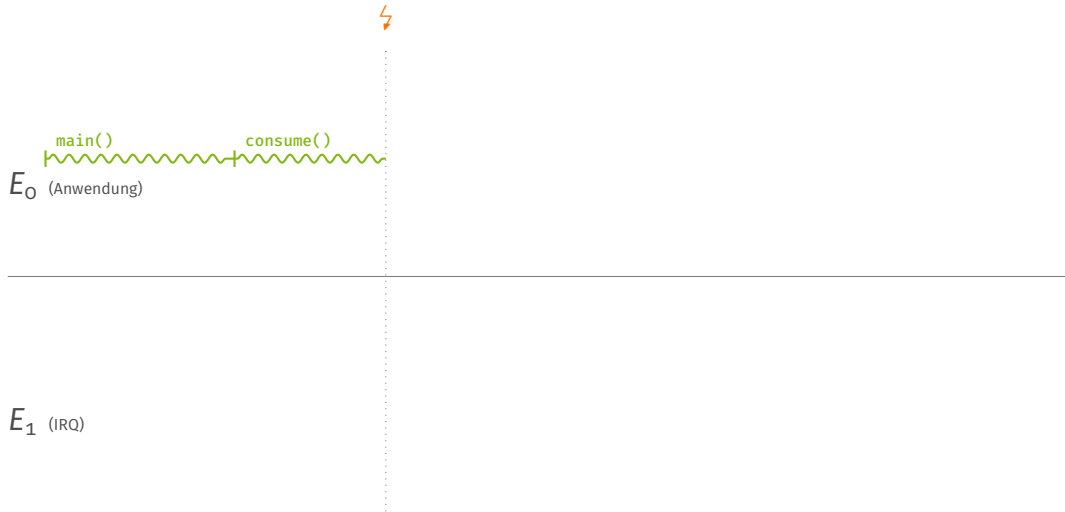
# Ohne Synchronisation



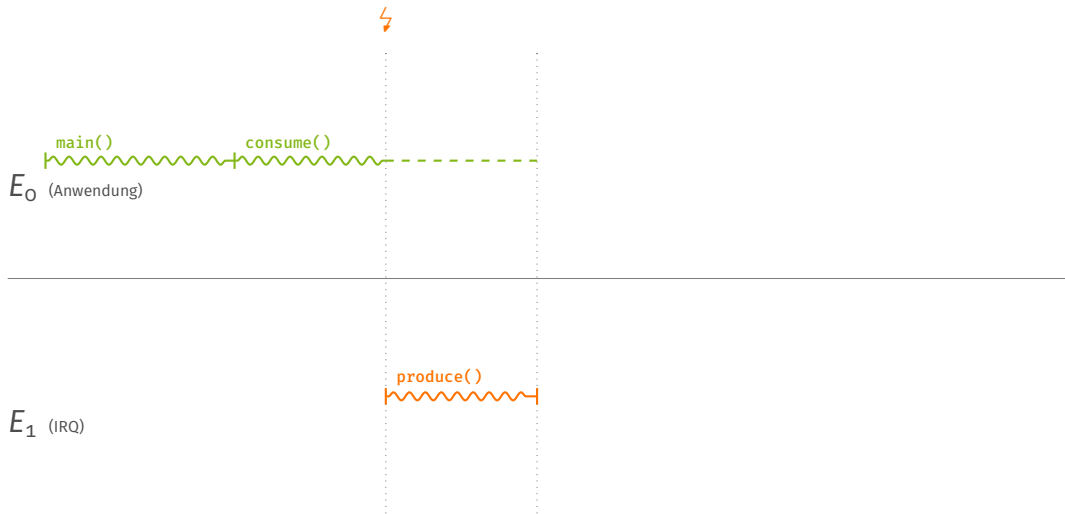
---

$E_1$  (IRQ)

# Ohne Synchronisation

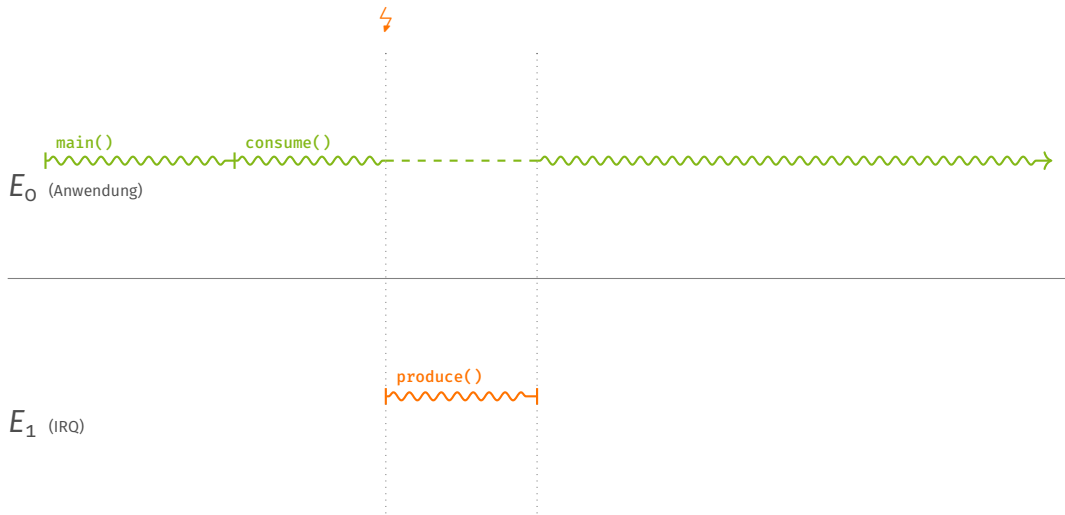


# Ohne Synchronisation





# Ohne Synchronisation



# Ohne Synchronisation

```
int buf[SIZE];
int pos = 0;

void produce(int data) {
    if (pos < SIZE)
        buf[pos++] = data;
}

int consume() {
    return pos > 0 ? buf[--pos] : -1;
}
```

**Lost Update möglich!**

## Bewährtes Hausmittel: Mutex

```
void produce(int data) {
    mutex.lock();
    if (pos < SIZE)
        buf[pos++] = data;
    mutex.unlock();
}

int consume() {
    mutex.lock();
    int r = pos > 0 ? buf[--pos] : -1;
    mutex.unlock();
    return r;
}
```

**Verklemmt sich!**

# Weiche Synchronisation

```
void produce(int data) {
    if (pos < SIZE)
        buf[pos++] = data;
}

int consume() {
    int x, r = -1;
    if (pos > 0)
        do {
            x = pos;
            r = buf[x];
        } while(!CAS(&pos, x, x-1));
    return r;
}
```

**Funktioniert!**

## Optimistischer Ansatz

- + keine Interruptsperre
- + kann sehr effizient sein
- kein generischer Ansatz
- kann sehr kompliziert werden
- fehleranfällig




## Optimistischer Ansatz

- + keine Interruptsperre
- + kann sehr effizient sein
- kein generischer Ansatz
- kann sehr kompliziert werden
- fehleranfällig

Betriebssystemarchitekt sollte Treiber- und Anwendungsentwicklern entgegenkommen → für Erfolg des Betriebssystems entscheidend

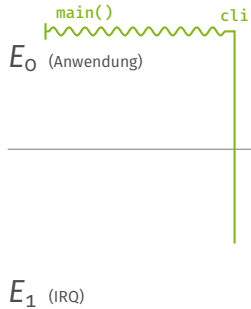
# Harte Synchronisation

`main()`  
  
 $E_0$  (Anwendung)

---

$E_1$  (IRQ)

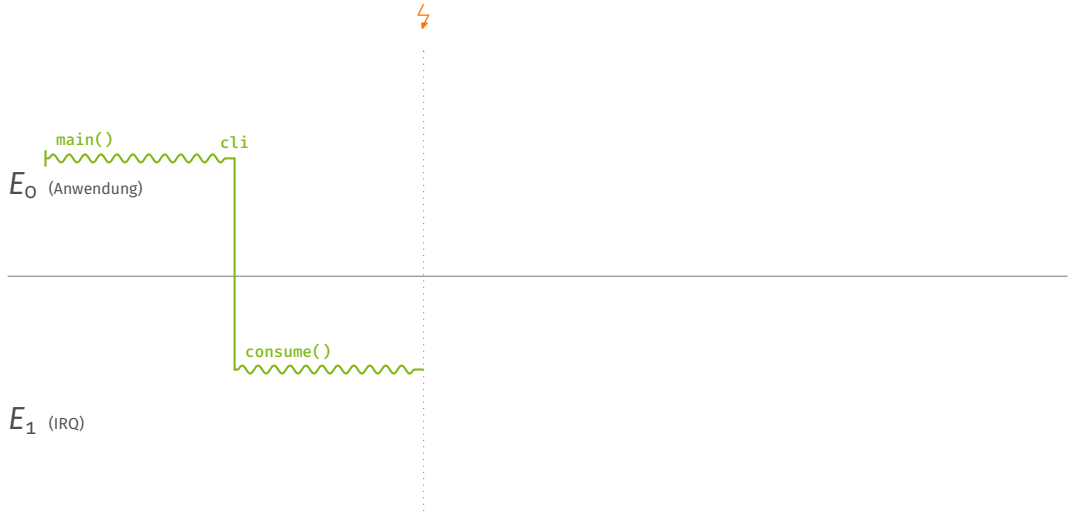
# Harte Synchronisation



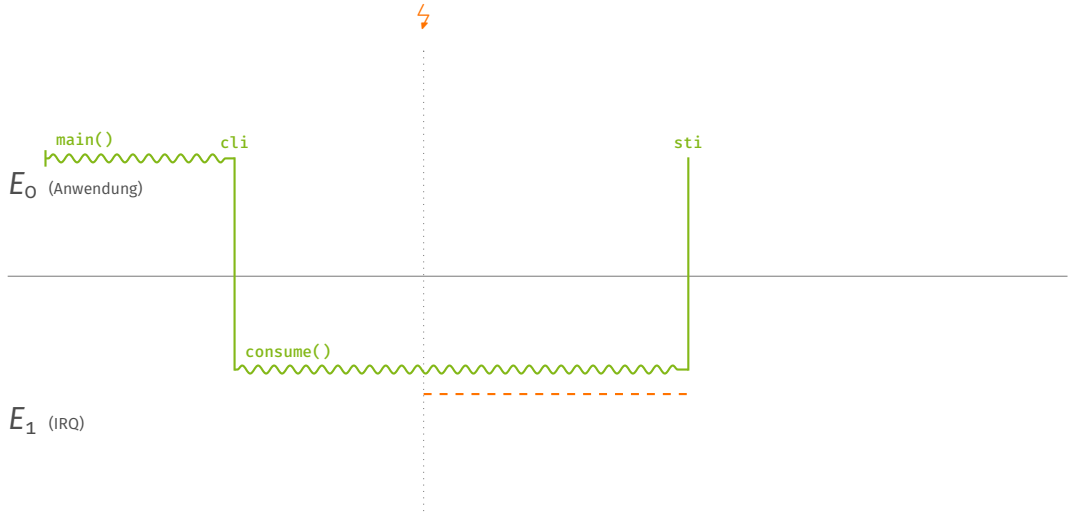
# Harte Synchronisation



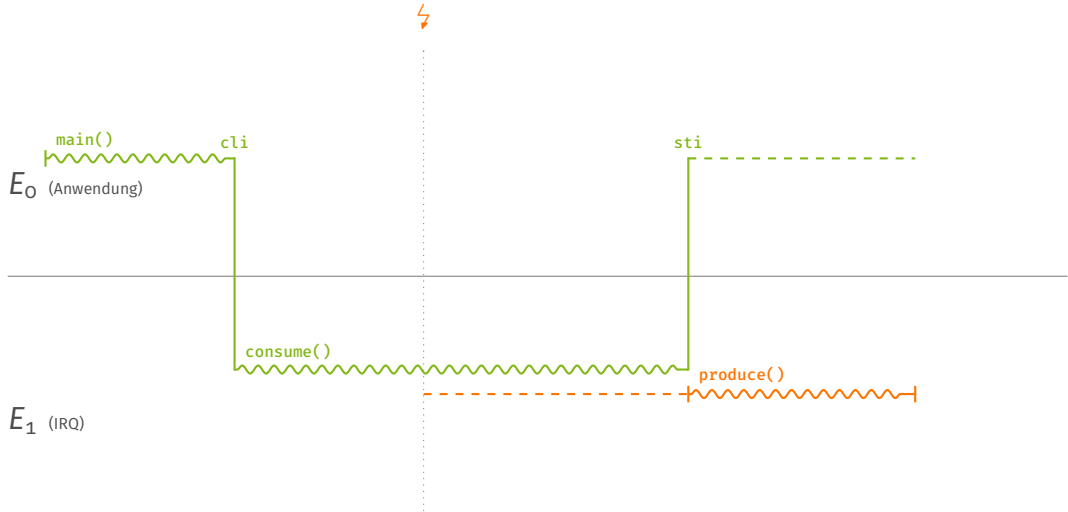
# Harte Synchronisation



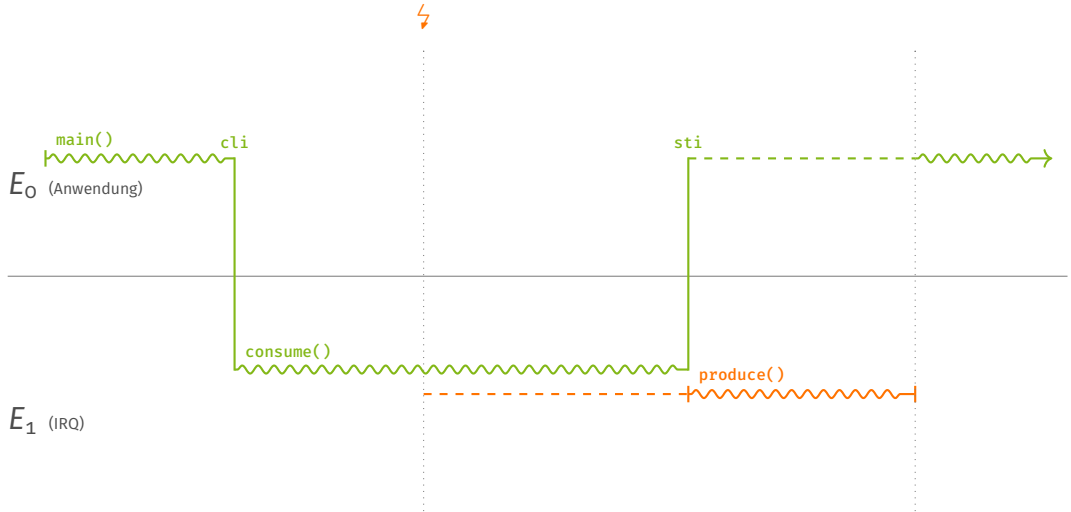
# Harte Synchronisation



# Harte Synchronisation

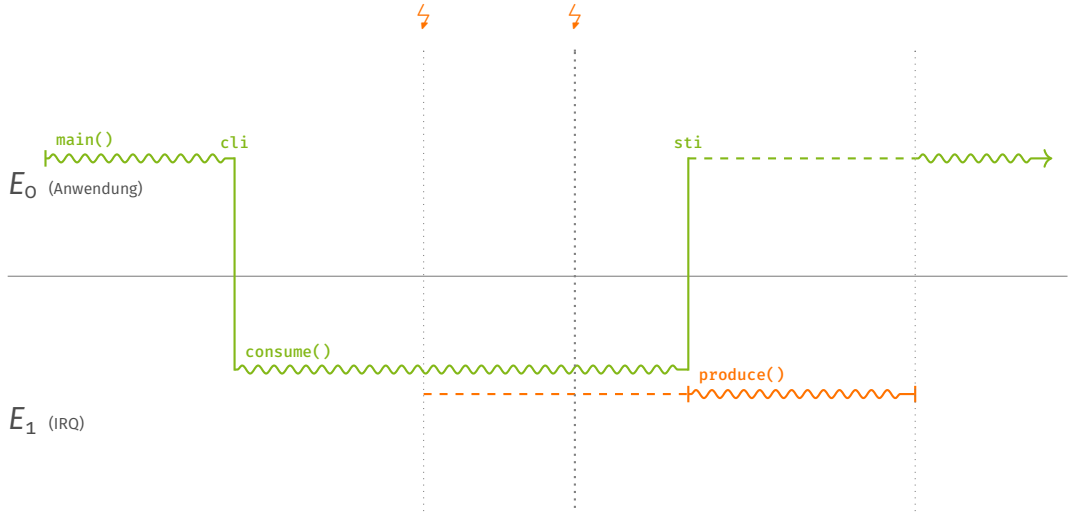


# Harte Synchronisation

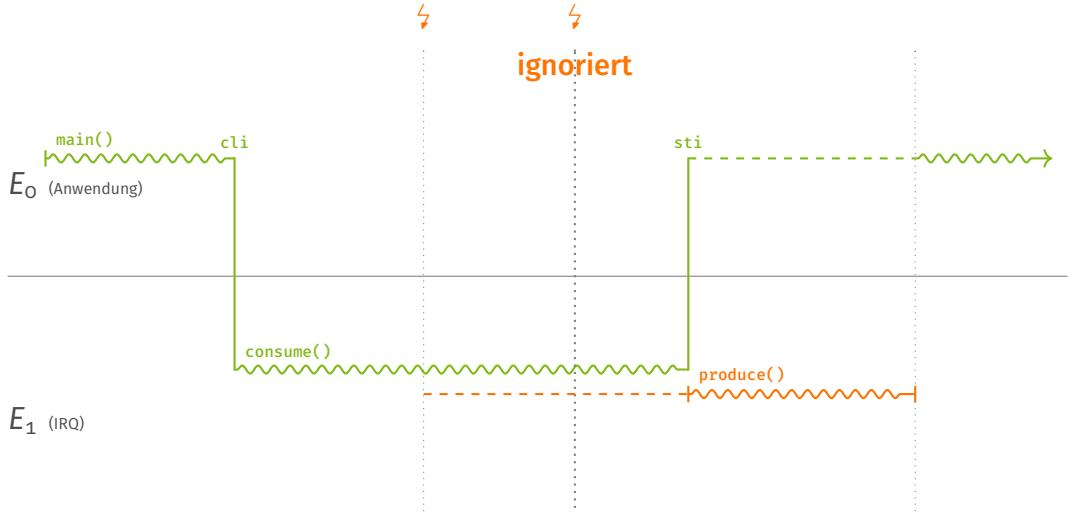




# Harte Synchronisation



# Harte Synchronisation



## Pessimistischer Ansatz

- + einfach
- + funktioniert immer
- Verzögerung von IRQs
  - hohe Latenz, ggf. Verlust von Interrupts
- blockiert pauschal alle IRQs

# Prolog/Epilog-Modell

---

## Aufspalten der IRQ-Behandlung in

**Prolog** erledigt das Nötigste auf  $E_1$

## Aufspalten der IRQ-Behandlung in

**Prolog** erledigt das Nötigste auf  $E_1$

**Epilog** läuft auf neuer Ebene  $\frac{1}{2}$  und übernimmt Synchronisation  
somit Ebene 1 / IRQs wieder frei

## Aufspalten der IRQ-Behandlung in

**Prolog** erledigt das Nötigste auf  $E_1$

**Epilog** läuft auf neuer Ebene  $\frac{1}{2}$  und übernimmt Synchronisation  
somit Ebene 1 / IRQs wieder frei

## Operationen

- höhere Ebene betreten: **cli**
- höhere Ebene verlassen: **sti**
- niedrigere Ebene unterbrechen: **IRQ-Leitung**

bei **harter Synchronisation**

## Aufspalten der IRQ-Behandlung in

**Prolog** erledigt das Nötigste auf  $E_1$

**Epilog** läuft auf neuer Ebene  $\frac{1}{2}$  und übernimmt Synchronisation  
somit Ebene 1 / IRQs wieder frei

## Operationen

- höhere Ebene betreten: **cli**, **enter**
- höhere Ebene verlassen: **sti**, **leave**
- niedrigere Ebene unterbrechen: **IRQ-Leitung**

bei **harter Synchronisation** und **Prolog/Epilog-Modell**



## Aufspalten der IRQ-Behandlung in

**Prolog** erledigt das Nötigste auf  $E_1$

**Epilog** läuft auf neuer Ebene  $\frac{1}{2}$  und übernimmt Synchronisation  
somit Ebene 1 / IRQs wieder frei

## Operationen

- höhere Ebene betreten: **cli**, **enter**
- höhere Ebene verlassen: **sti**, **leave**
- niedrigere Ebene unterbrechen: **IRQ-Leitung**, **relay**

bei **harter Synchronisation** und **Prolog/Epilog-Modell**

# Prolog/Epilog-Modell

`main()`  
  
 $E_0$  (Anwendung)

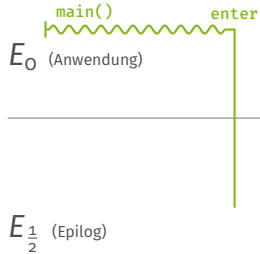
---

$E_{\frac{1}{2}}$  (Epilog)

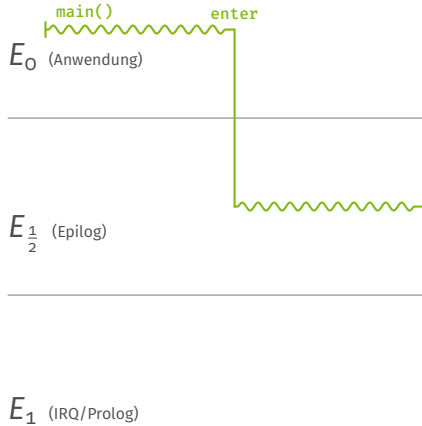
---

$E_1$  (IRQ/Prolog)

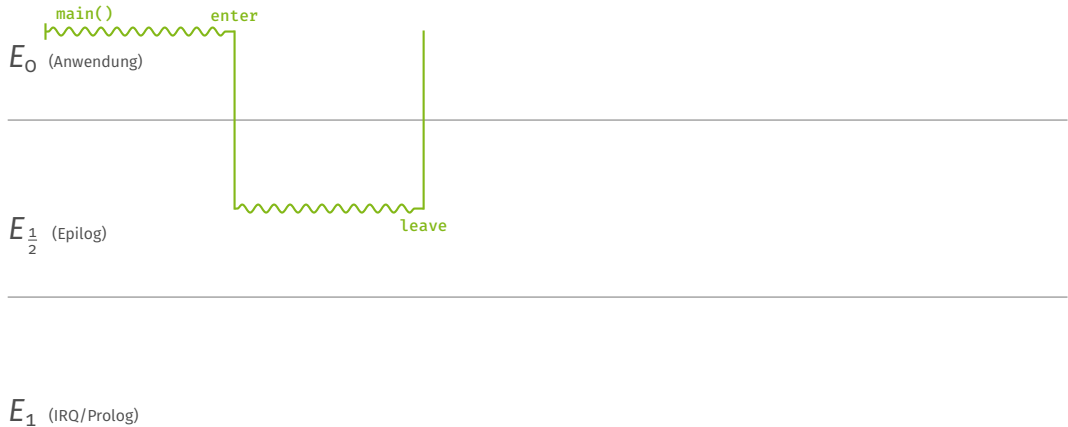
# Prolog/Epilog-Modell



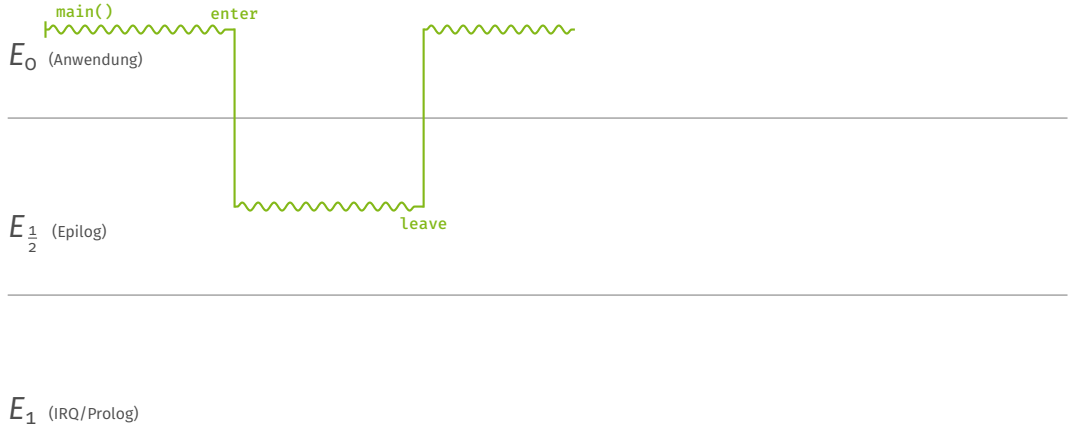
# Prolog/Epilog-Modell



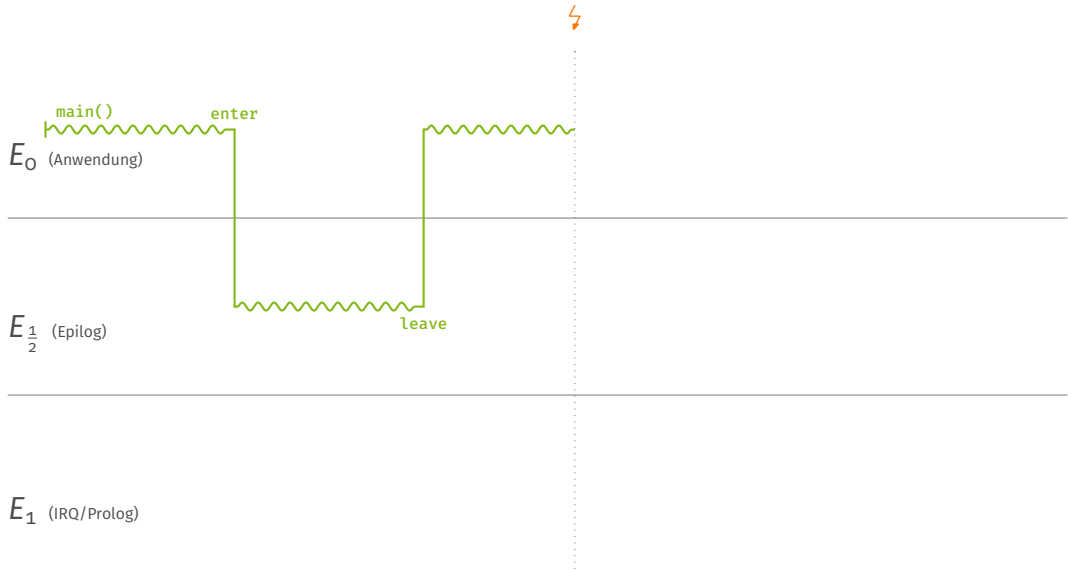
# Prolog/Epilog-Modell



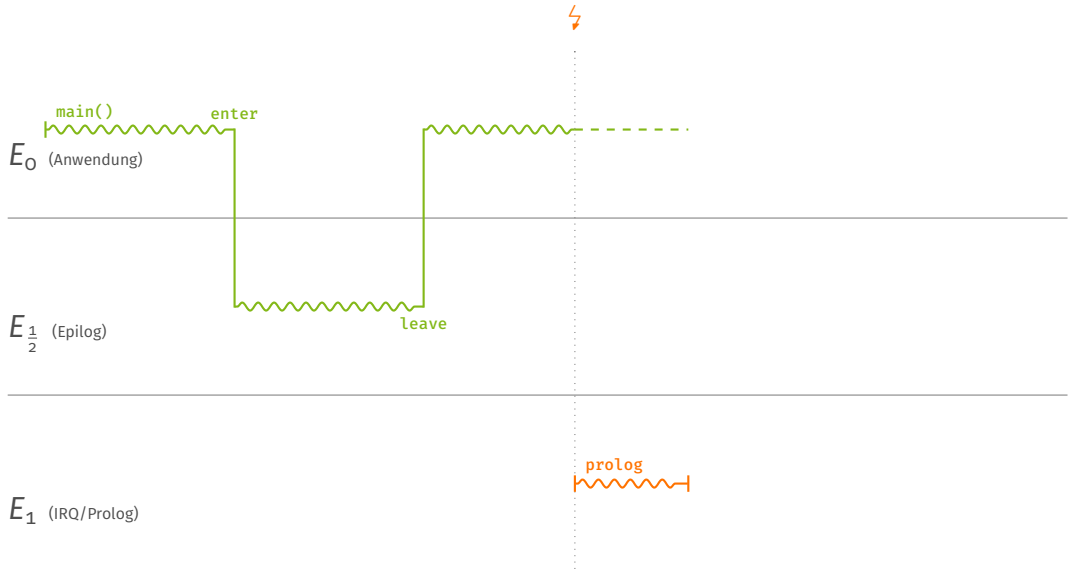
# Prolog/Epilog-Modell



# Prolog/Epilog-Modell

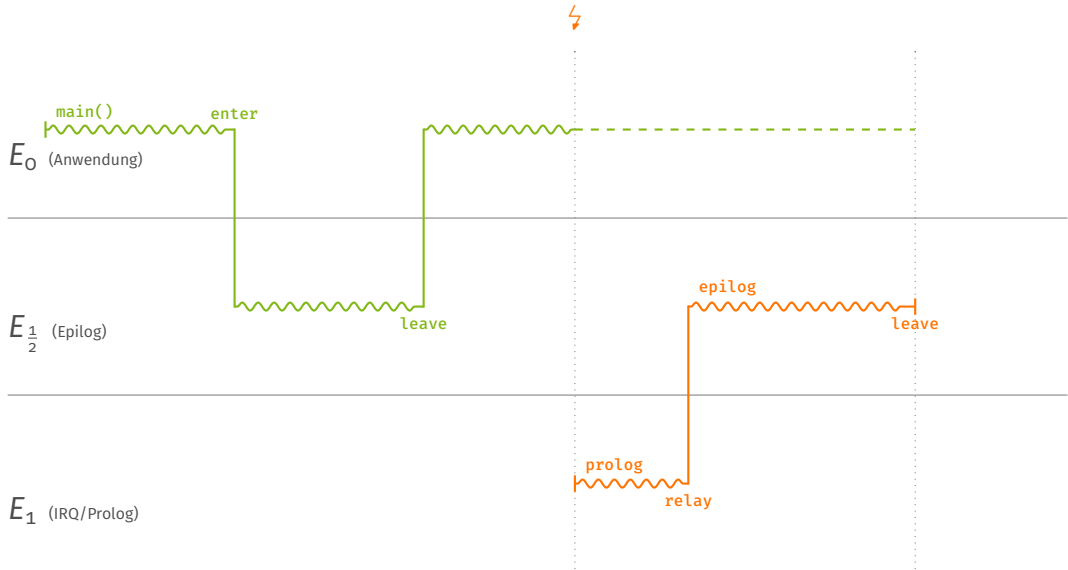


# Prolog/Epilog-Modell

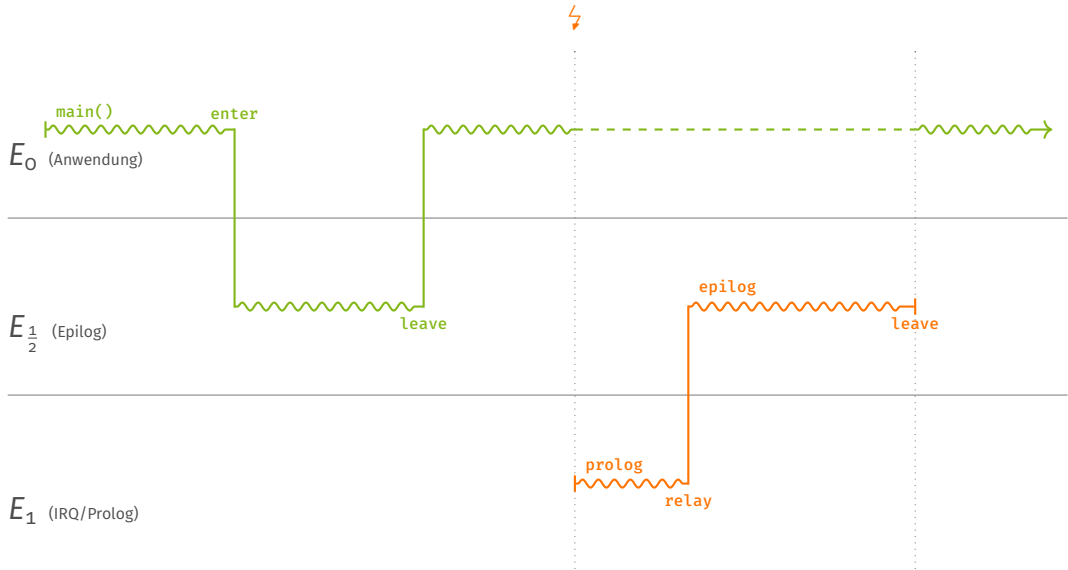




# Prolog/Epilog-Modell



# Prolog/Epilog-Modell



## Kombinierter Ansatz

- + einfaches Programmiermodell  
(für Anwendungsentwickler)
- + geringer Interruptverlust
- Ebene  $\frac{1}{2}$  ist zusätzlicher Overhead
- etwas mehr Arbeit für den Betriebssystemarchitekten

## Kombinierter Ansatz


- + einfaches Programmiermodell  
(für Anwendungsentwickler)
- + geringer Interruptverlust
- Ebene  $\frac{1}{2}$  ist zusätzlicher Overhead
- etwas mehr Arbeit für den Betriebssystemarchitekten

→ **guter Kompromiss**

```
main(){  
  while(1){  
    enter();  
    consume();  
    leave();  
  }  
}
```

```
epilog(){  
  // ...  
  produce();  
  // ...  
}
```

# Umsetzung

`main()`  
  
 $E_0$  (Anwendung)

---

$E_{\frac{1}{2}}$  (Epilog)

---

$E_1$  (IRQ/Prolog)

# Umsetzung

`main()`  
`enter`  
 $E_0$  (Anwendung)



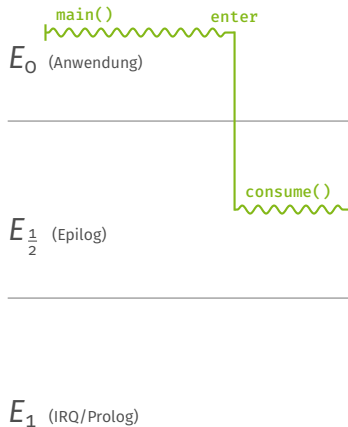
---

$E_{\frac{1}{2}}$  (Epilog)

---

$E_1$  (IRQ/Prolog)

# Umsetzung

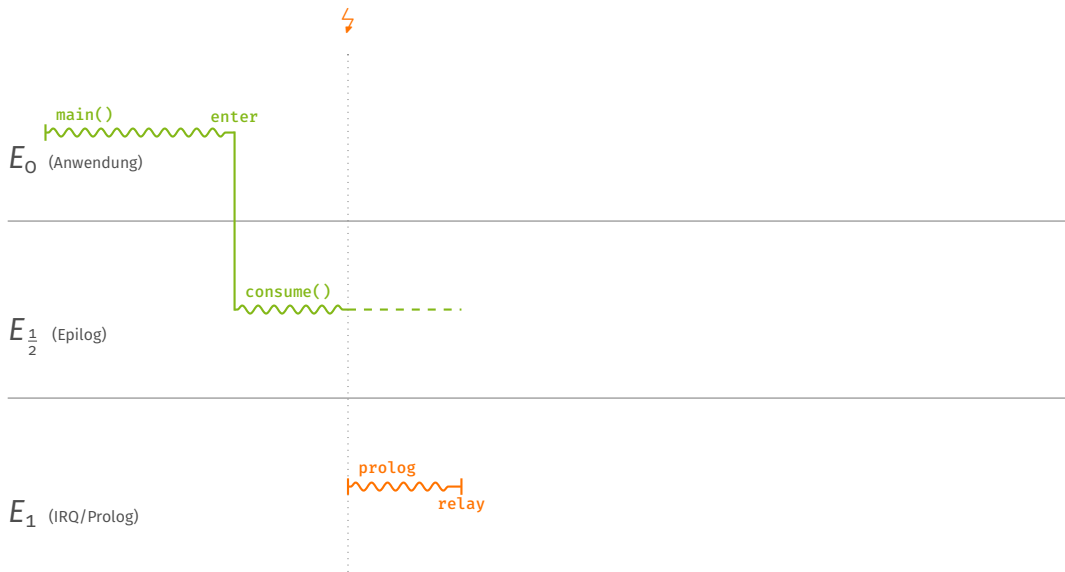




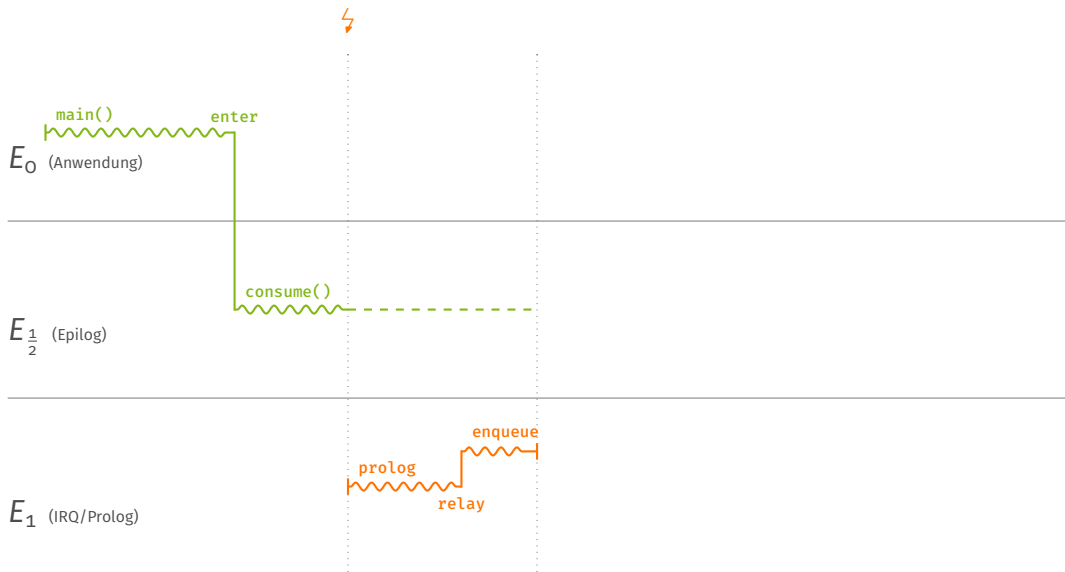
# Umsetzung



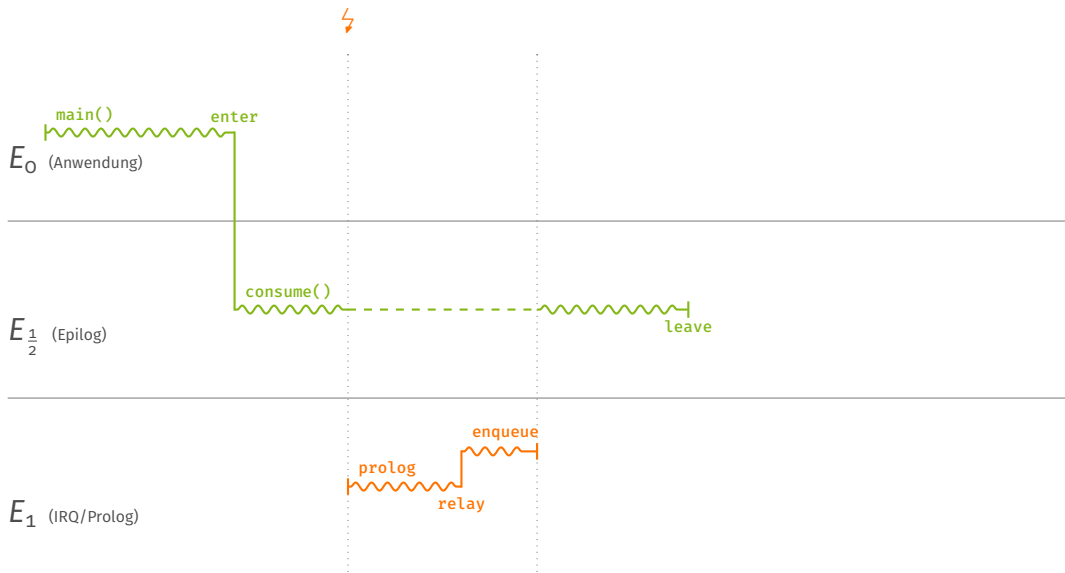
# Umsetzung



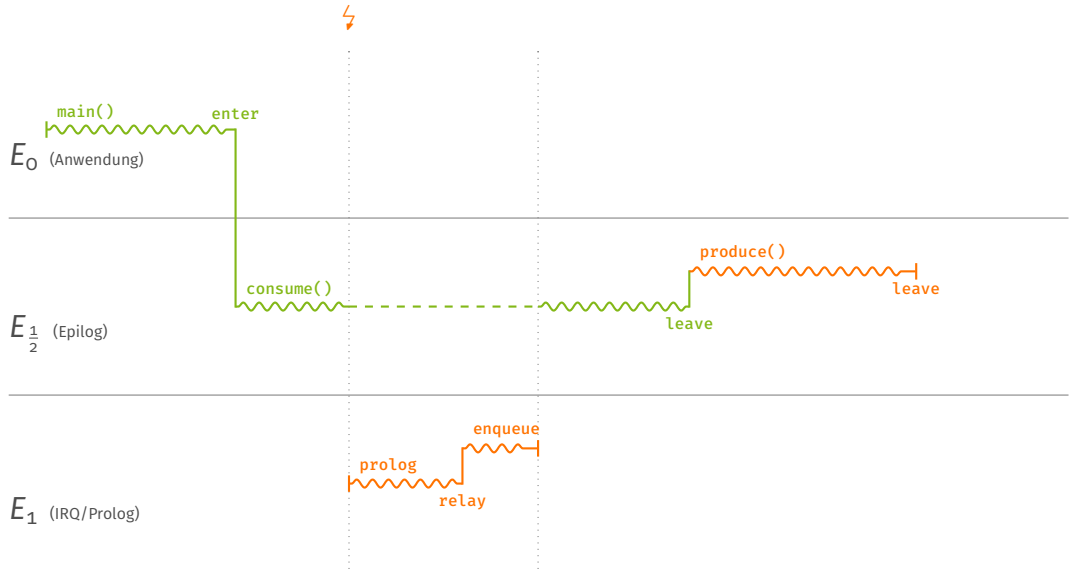
# Umsetzung



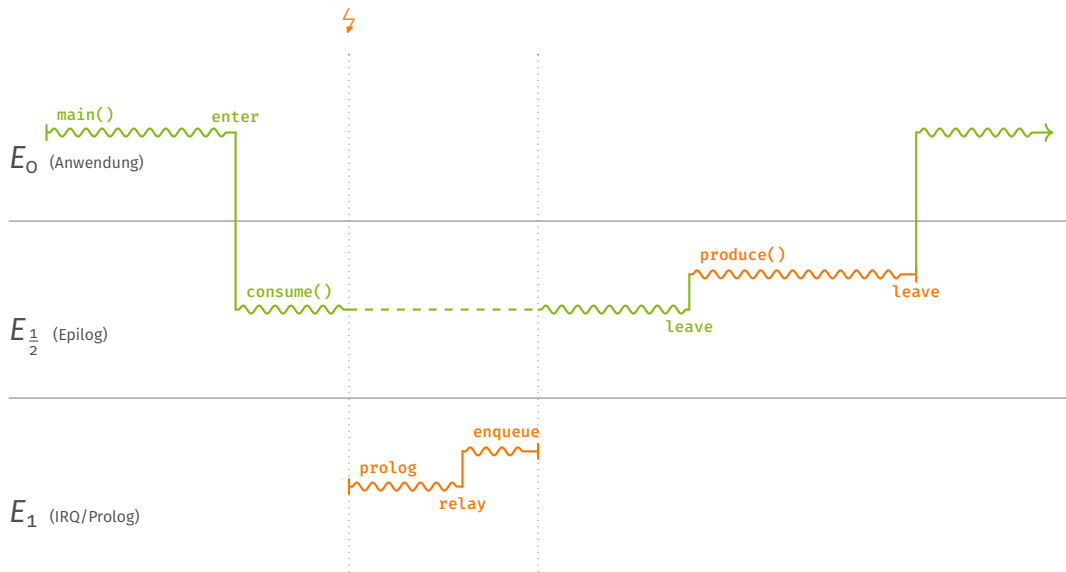
# Umsetzung



# Umsetzung



# Umsetzung



# Implementierung

---

# Wechsel von harter Synchronisation zu Prolog/Epilog-Modell

Was wird gebraucht?



# Wechsel von harter Synchronisation zu Prolog/Epilog-Modell

## Was wird gebraucht?

- **Guard** mit `enter()`, `leave()` und `relay()` für Prioritätsebenen
- Epilogwarteschlange **GateQueue** zum Einreihen der Epiloge

# Wechsel von harter Synchronisation zu Prolog/Epilog-Modell

## Was wird gebraucht?

- **Guard** mit **enter()**, **leave()** und **relay()** für Prioritätsebenen
- Epilogwarteschlange **GateQueue** zum Einreihen der Epiloge

## Was muss angepasst werden?

# Wechsel von harter Synchronisation zu Prolog/Epilog-Modell

## Was wird gebraucht?

- **Guard** mit **enter()**, **leave()** und **relay()** für Prioritätsebenen
- Epilogwarteschlange **GateQueue** zum Einreihen der Epiloge

## Was muss angepasst werden?

- Unterbrechungsbehandlung (**interrupt\_handler**) und **Gate** (von **trigger()** zu **prolog()** und **epilog()**)
- alle Treiber (**Keyboard**)
- die Anwendung (**Application**)

# Wechsel von harter Synchronisation zu Prolog/Epilog-Modell

## Was wird gebraucht?

- **Guard** mit **enter()**, **leave()** und **relay()** für Prioritätsebenen
- Epilogwarteschlange **GateQueue** zum Einreihen der Epiloge

## Was muss angepasst werden?

- Unterbrechungsbehandlung (**interrupt\_handler**) und **Gate** (von **trigger()** zu **prolog()** und **epilog()**)
- alle Treiber (**Keyboard**)
- die Anwendung (**Application**)
- *alles was hart synchronisiert*

## **Besonderheiten in MPStuBS**

---

# Prolog/Epilog-Modell auf Mehrkernprozessoren

- Jeder Kern hat eine **eigene** Epilogwarteschlange  
(damit die Epiloge auf dem selben Kern wie deren zugehörige Prologe ausgeführt werden)

# Prolog/Epilog-Modell auf Mehrkernprozessoren

- Jeder Kern hat eine **eigene** Epilogwarteschlange  
(damit die Epiloge auf dem selben Kern wie deren zugehörige Prologe ausgeführt werden)
- Eine **Gate**-Instanz darf **nicht mehrfach in einer** Epilogwarteschlangen vorkommen

# Prolog/Epilog-Modell auf Mehrkernprozessoren

- Jeder Kern hat eine **eigene** Epilogwarteschlange  
(damit die Epiloge auf dem selben Kern wie deren zugehörige Prologe ausgeführt werden)
- Eine **Gate**-Instanz darf **nicht mehrfach in einer** Epilogwarteschlangen vorkommen
- Eine **Gate**-Instanz kann aber **gleichzeitig in unterschiedlichen** Epilogwarteschlangen vorkommen



# Prolog/Epilog-Modell auf Mehrkernprozessoren

- Jeder Kern hat eine **eigene** Epilogwarteschlange  
(damit die Epiloge auf dem selben Kern wie deren zugehörige Prologe ausgeführt werden)
- Eine **Gate**-Instanz darf **nicht mehrfach in einer** Epilogwarteschlangen vorkommen
- Eine **Gate**-Instanz kann aber **gleichzeitig in unterschiedlichen** Epilogwarteschlangen vorkommen
- Zu jedem Zeitpunkt darf **maximal ein Kern** Epiloge ausführen

# Prolog/Epilog-Modell auf Mehrkernprozessoren

- Jeder Kern hat eine **eigene** Epilogwarteschlange  
(damit die Epiloge auf dem selben Kern wie deren zugehörige Prologe ausgeführt werden)
- Eine **Gate**-Instanz darf **nicht mehrfach in einer** Epilogwarteschlangen vorkommen
- Eine **Gate**-Instanz kann aber **gleichzeitig in unterschiedlichen** Epilogwarteschlangen vorkommen
- Zu jedem Zeitpunkt darf **maximal ein Kern** Epiloge ausführen  
→ Verwendung eines **big kernel lock** (BKL)

# Prolog/Epilog-Modell auf Mehrkernprozessoren

- Jeder Kern hat eine **eigene** Epilogwarteschlange  
(damit die Epiloge auf dem selben Kern wie deren zugehörige Prologe ausgeführt werden)
- Eine **Gate**-Instanz darf **nicht mehrfach in einer** Epilogwarteschlangen vorkommen
- Eine **Gate**-Instanz kann aber **gleichzeitig in unterschiedlichen** Epilogwarteschlangen vorkommen
- Zu jedem Zeitpunkt darf **maximal ein Kern** Epiloge ausführen  
→ Verwendung eines **big kernel lock** (BKL) via Spinlock

# Prolog/Epilog-Modell auf Mehrkernprozessoren

- Jeder Kern hat eine **eigene** Epilogwarteschlange  
(damit die Epiloge auf dem selben Kern wie deren zugehörige Prologe ausgeführt werden)
- Eine **Gate**-Instanz darf **nicht mehrfach in einer** Epilogwarteschlangen vorkommen
- Eine **Gate**-Instanz kann aber **gleichzeitig in unterschiedlichen** Epilogwarteschlangen vorkommen
- Zu jedem Zeitpunkt darf **maximal ein Kern** Epiloge ausführen  
→ Verwendung eines **big kernel lock** (BKL) via Spinlock
- **Korrekte Sperrreihenfolge ist extrem wichtig!**

# Beispiel für Mehrkernprozessoren

CPU 1 

CPU 0 

$E_0$

---

$E_{\frac{1}{2}}$

---

$E_1$

# Beispiel für Mehrkernprozessoren

CPU 1 

CPU 0 

$E_0$

$E_{\frac{1}{2}}$

$E_1$

# Beispiel für Mehrkernprozessoren

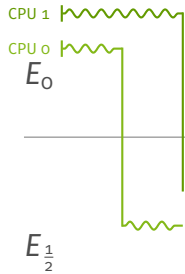
CPU 1 

CPU 0   
 $E_0$

$E_{\frac{1}{2}}$

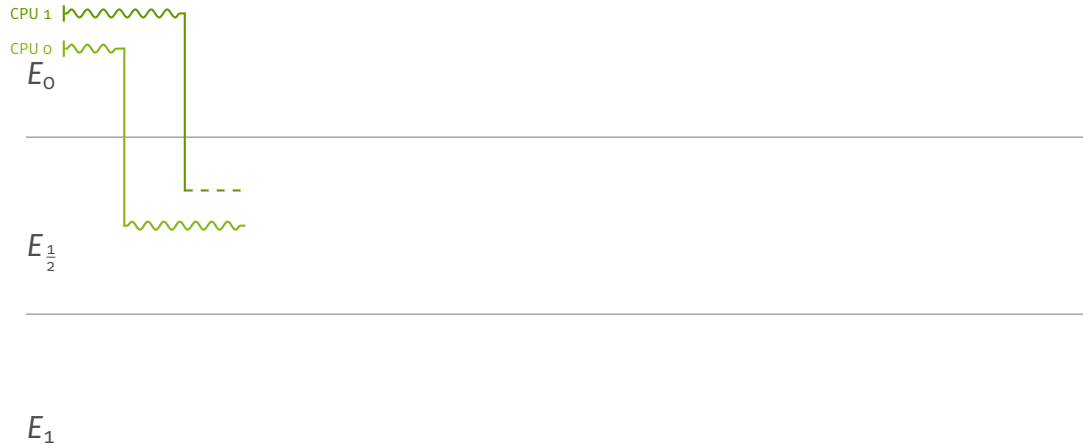
$E_1$

# Beispiel für Mehrkernprozessoren

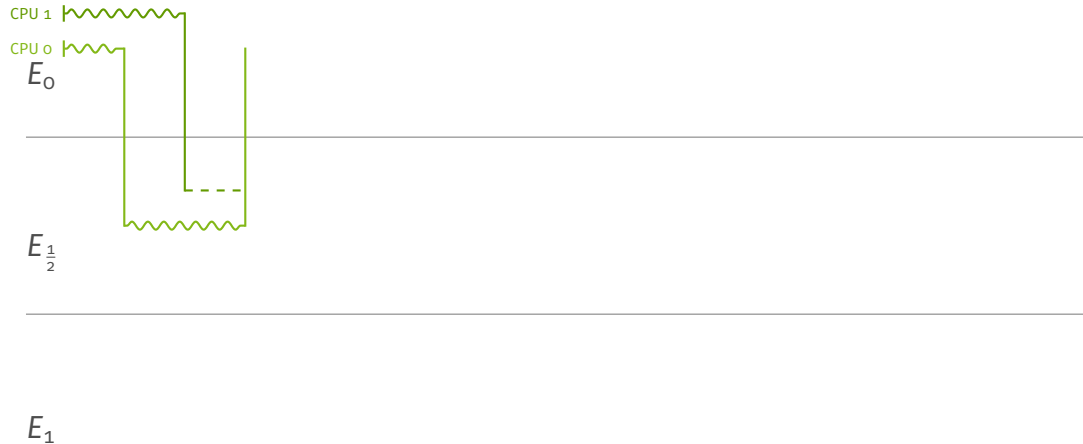




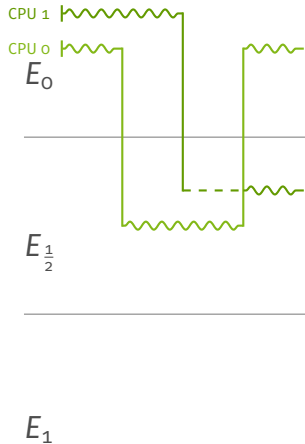
# Beispiel für Mehrkernprozessoren



# Beispiel für Mehrkernprozessoren



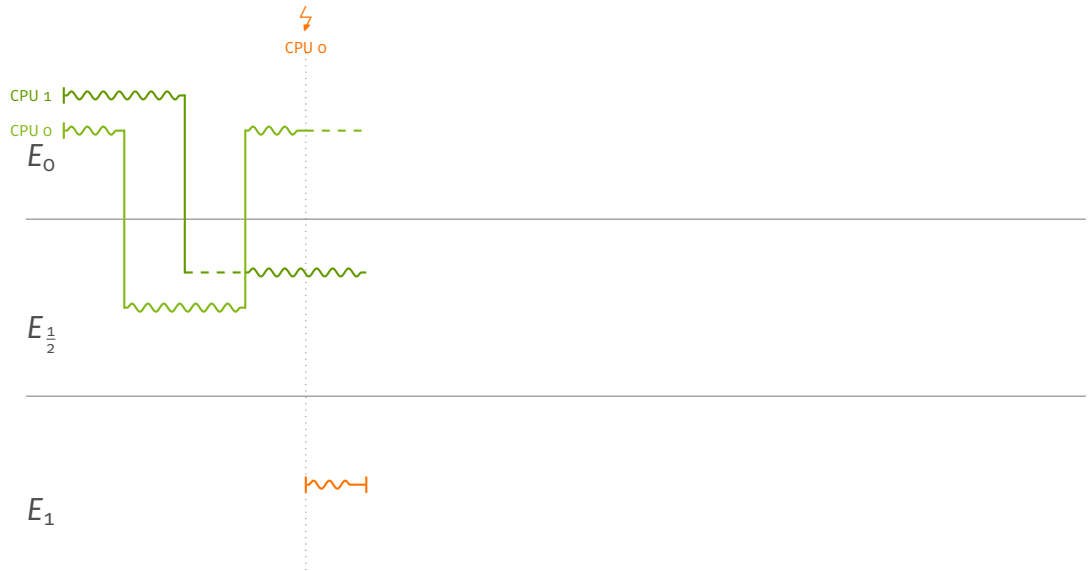
# Beispiel für Mehrkernprozessoren



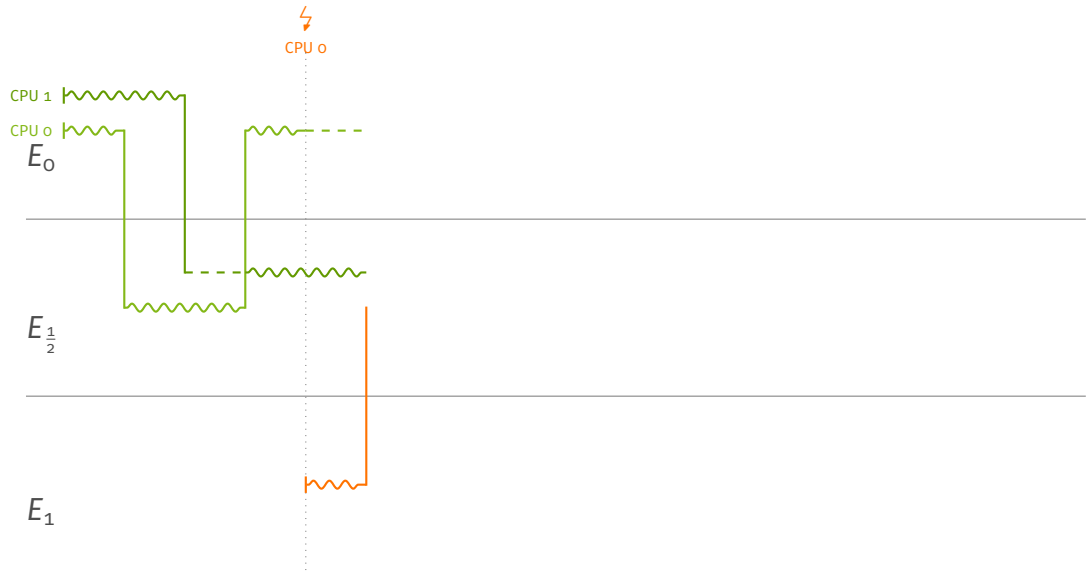
# Beispiel für Mehrkernprozessoren



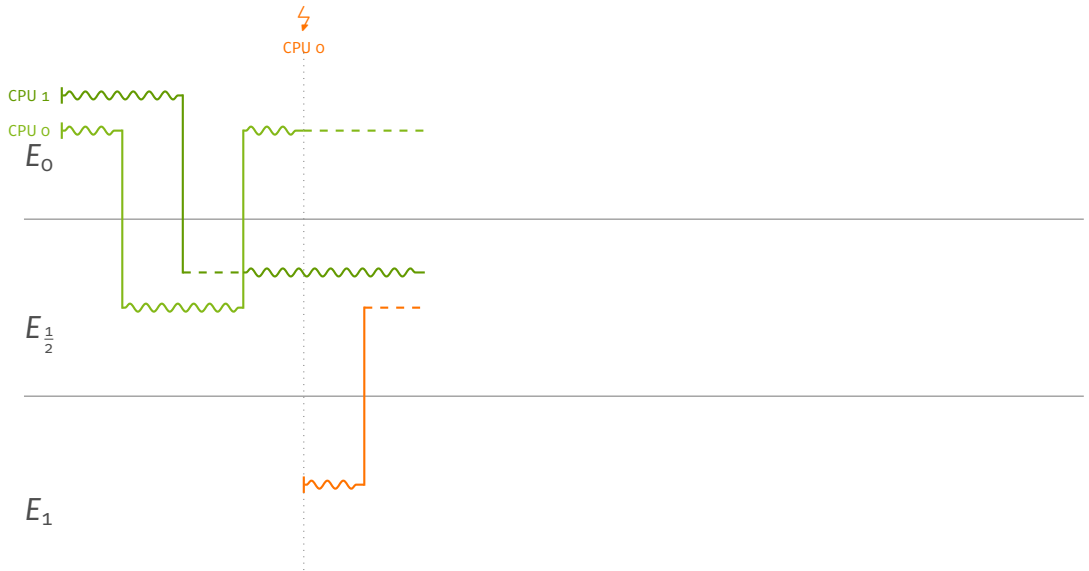
# Beispiel für Mehrkernprozessoren



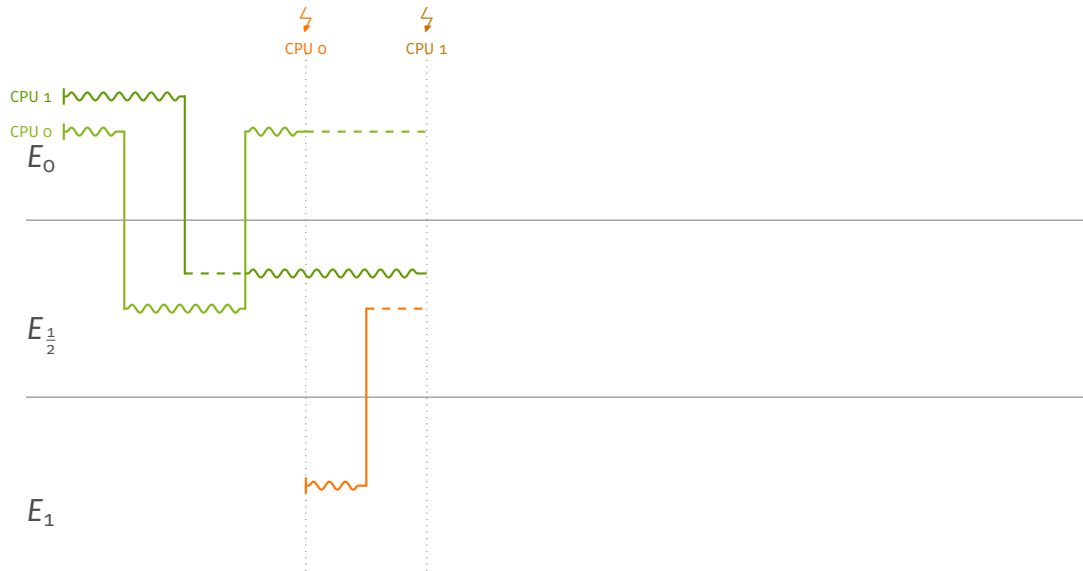
# Beispiel für Mehrkernprozessoren



# Beispiel für Mehrkernprozessoren

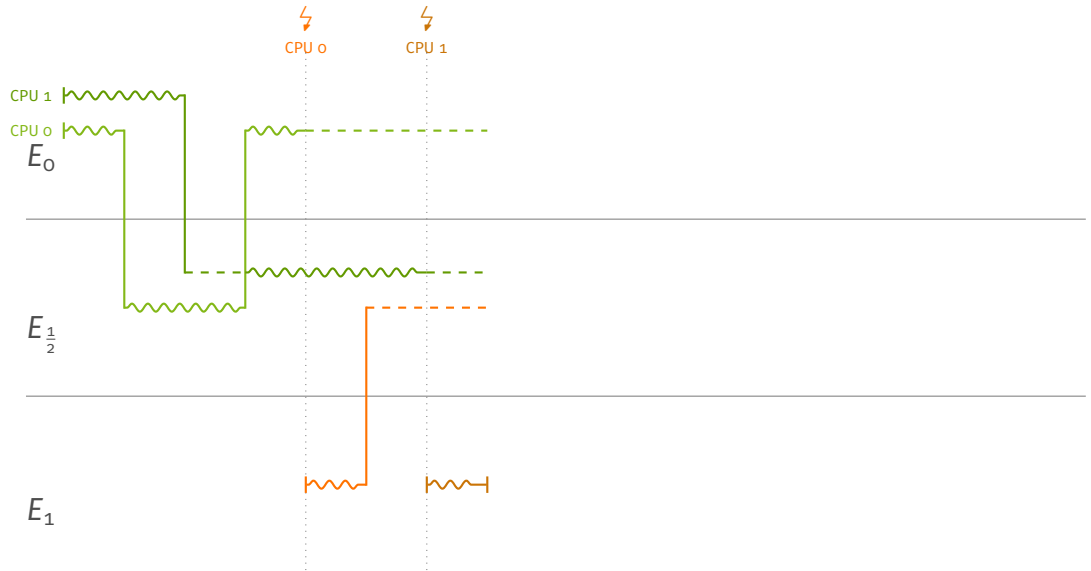


# Beispiel für Mehrkernprozessoren

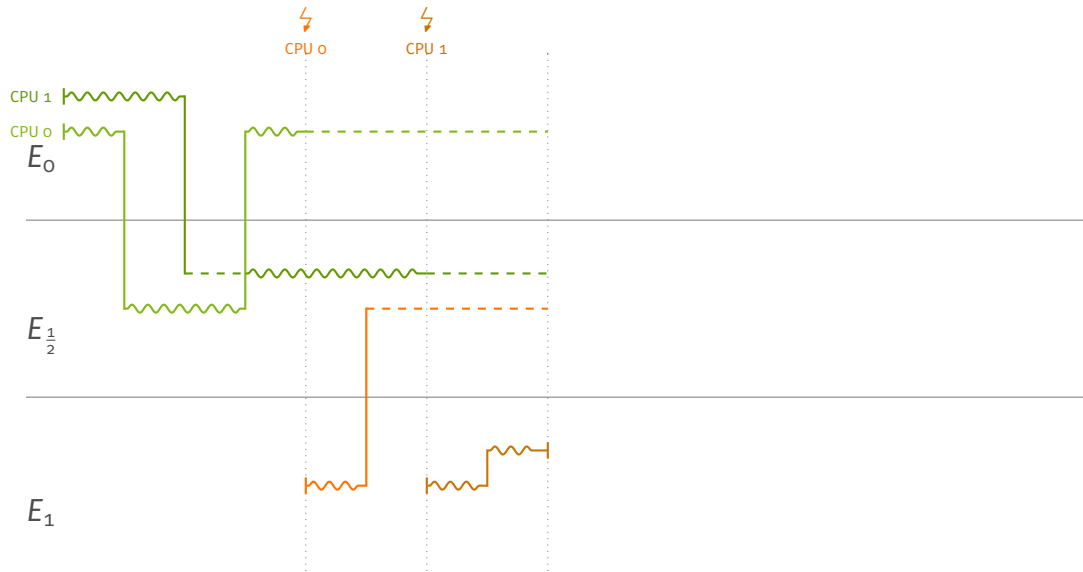




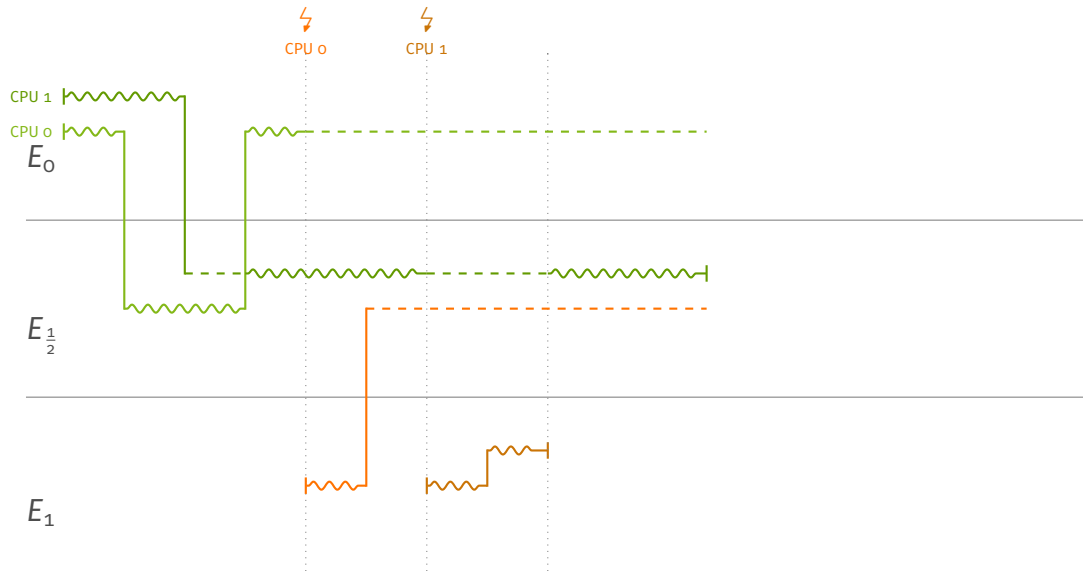
# Beispiel für Mehrkernprozessoren



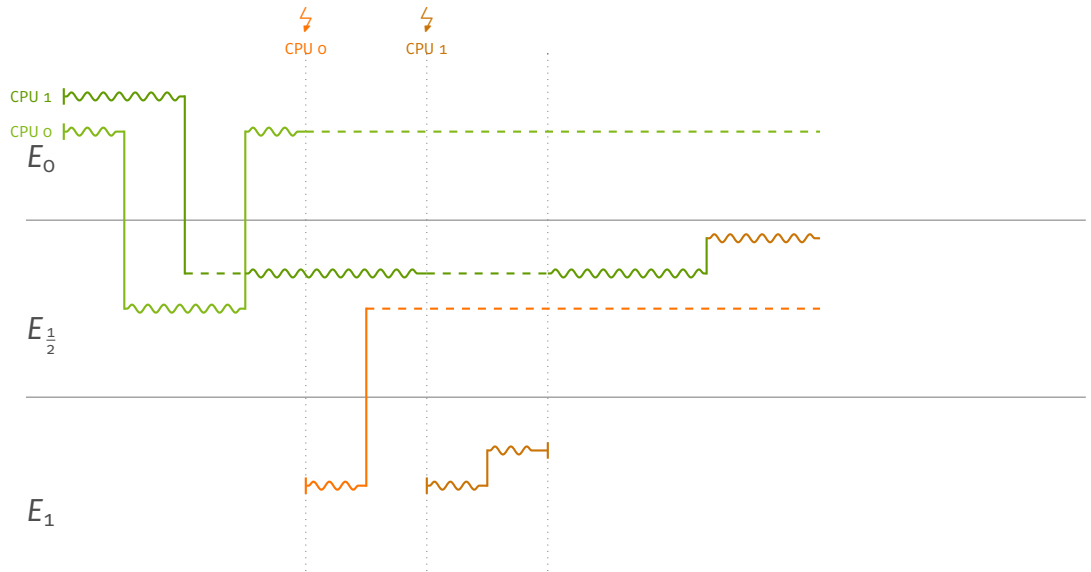
# Beispiel für Mehrkernprozessoren



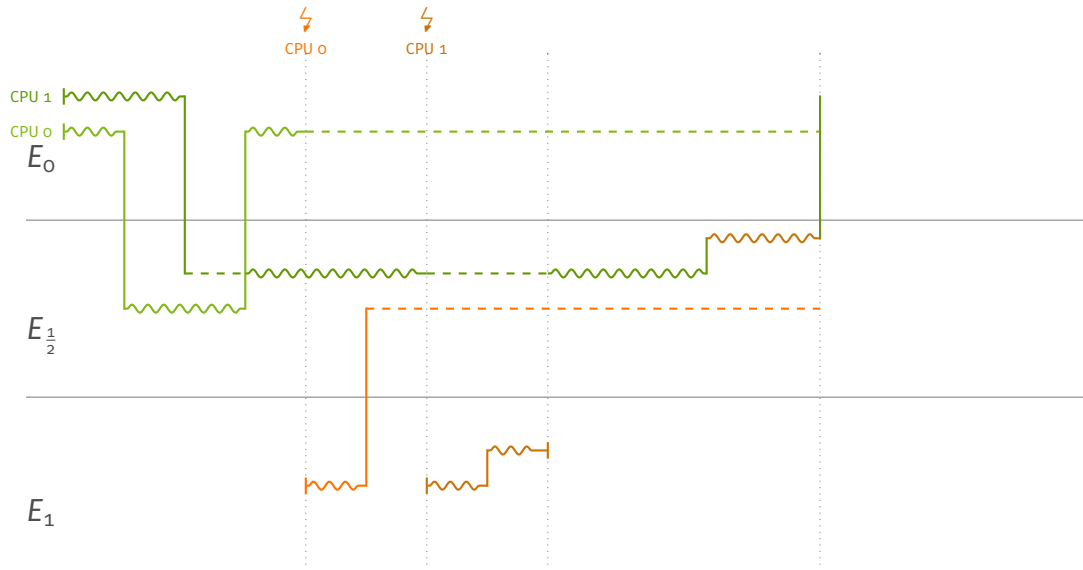
# Beispiel für Mehrkernprozessoren



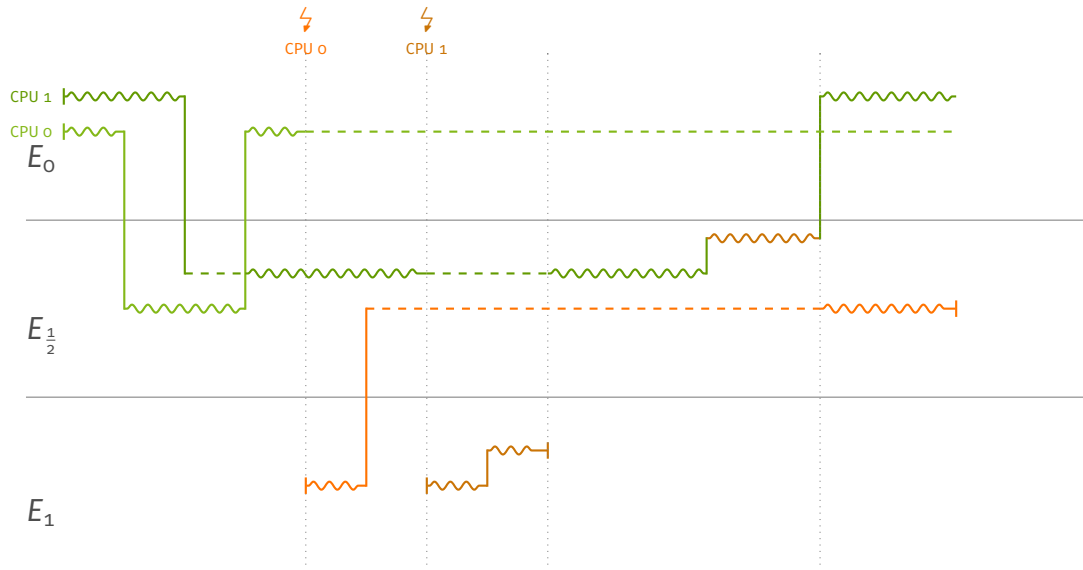
# Beispiel für Mehrkernprozessoren



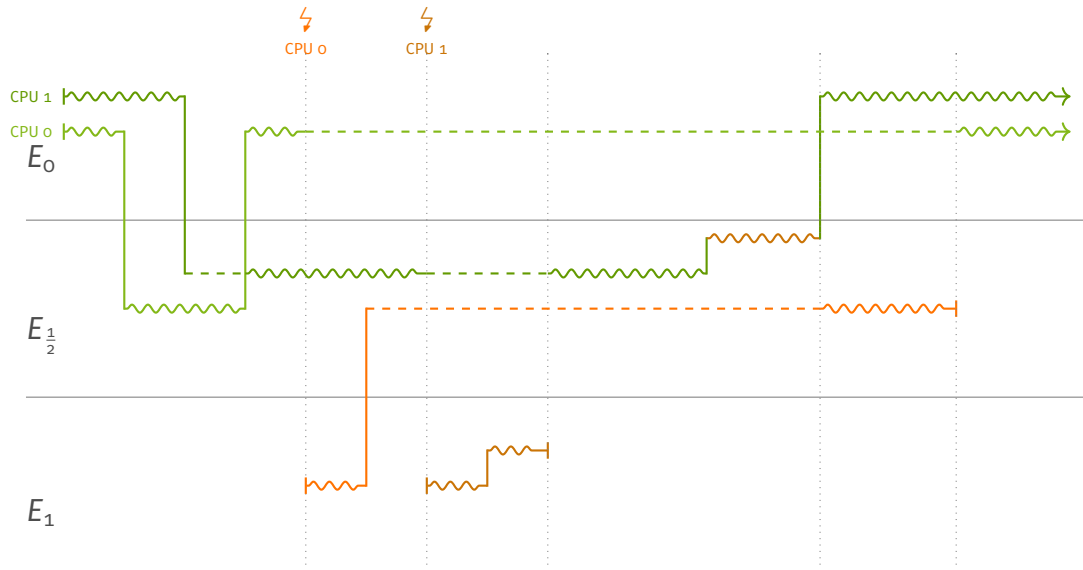
# Beispiel für Mehrkernprozessoren



# Beispiel für Mehrkernprozessoren



# Beispiel für Mehrkernprozessoren



# Fragen?

---

Nächste Woche (22. November) ist das Assembler-Seminar