

Übung zu Betriebssystembau

Prolog/Epilog-Modell

12. November 2024

Alexander Krause

Arbeitsgruppe Systemsoftware
Technische Universität Dortmund

(Mit Material vom Lehrstuhl 4 der FAU)


Interrupts verändern (potenziell) den Zustand des Systems

Interrupts verändern (potenziell) den Zustand des Systems

```
01 main(){  
02     while(1){  
03         // ...  
04         consume();  
05         // ...  
06     }  
07 }
```

```
01 interrupt_handler(){  
02  
03  
04     produce();  
05  
06  
07 }
```

Ohne Synchronisation

`main()`

 E_0 (Anwendung)

E_1 (IRQ)



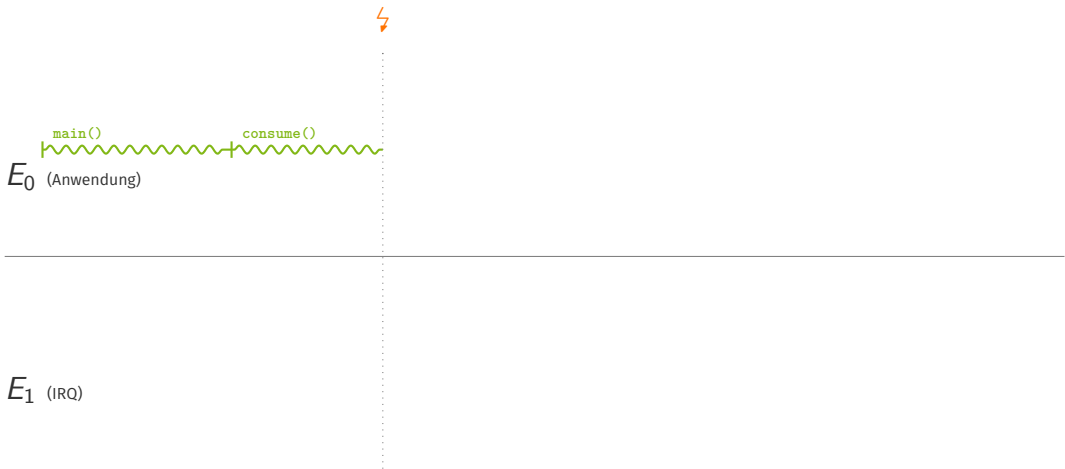
Ohne Synchronisation



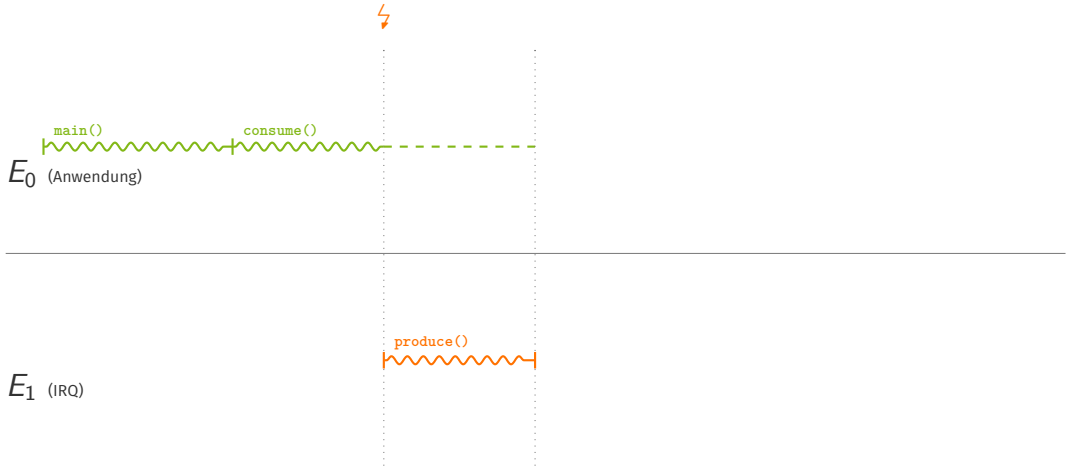
E_1 (IRQ)



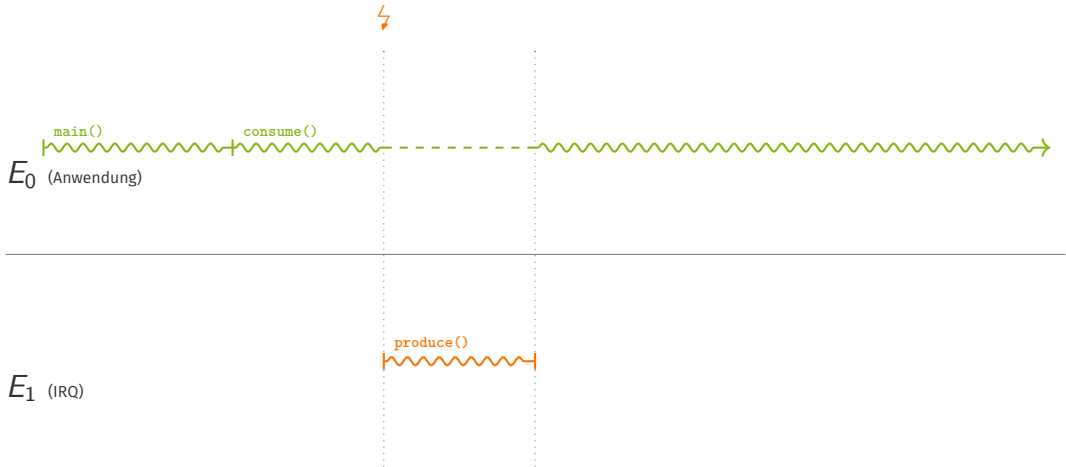
Ohne Synchronisation



Ohne Synchronisation



Ohne Synchronisation



Ohne Synchronisation

```
01 int buf[SIZE];
02 int pos = 0;
03
04 void produce(int data) {
05     if (pos < SIZE)
06         buf[pos++] = data;
07 }
08
09 int consume() {
10     return pos > 0 ? buf[--pos] : -1;
11 }
```



Lost Update möglich!



Bewährtes Hausmittel: Mutex

```
01 void produce(int data) {
02     mutex.lock();
03     if (pos < SIZE)
04         buf[pos++] = data;
05     mutex.unlock();
06 }
07
08 int consume() {
09     mutex.lock();
10     int r = pos > 0 ? buf[--pos] : -1;
11     mutex.unlock();
12     return r;
13 }
```



Verklemmt sich!



Weiche Synchronisation

```
01 void produce(int data) {
02     if (pos < SIZE)
03         buf[pos++] = data;
04 }
05
06 int consume() {
07     int x, r = -1;
08     if (pos > 0)
09         do {
10             x = pos;
11             r = buf[x];
12         } while(!CAS(&pos, x, x-1));
13     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)



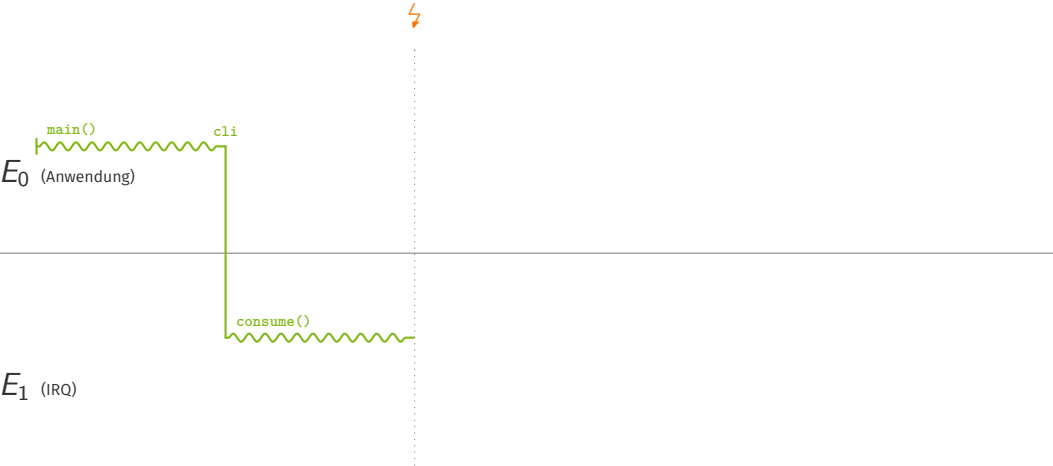
Harte Synchronisation



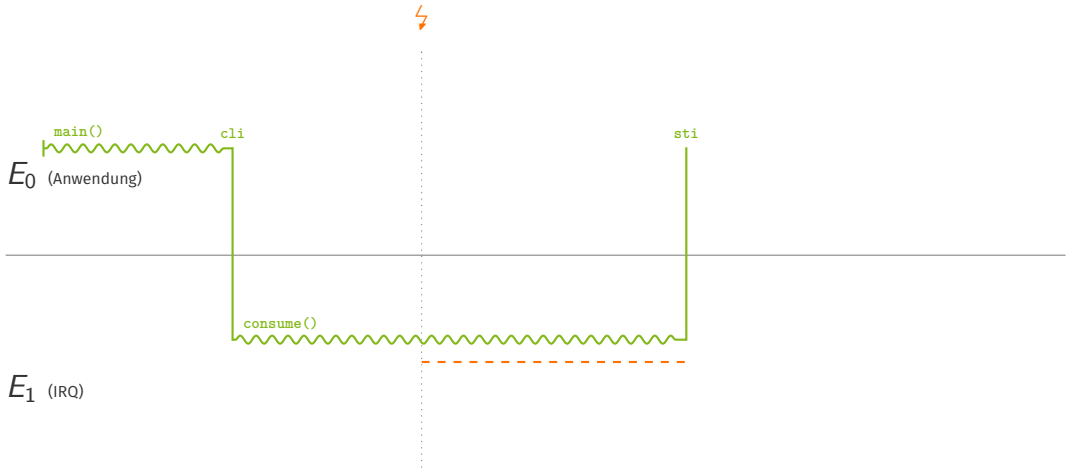
Harte Synchronisation



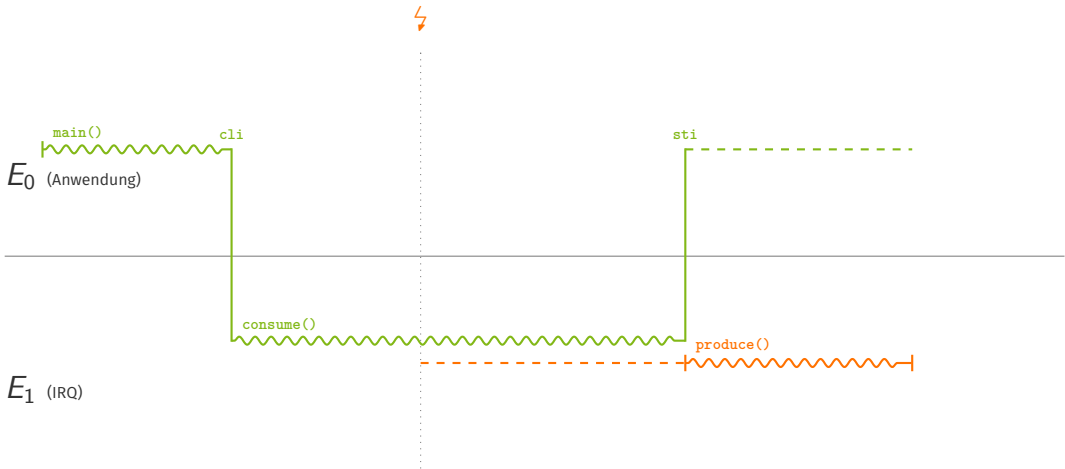
Harte Synchronisation



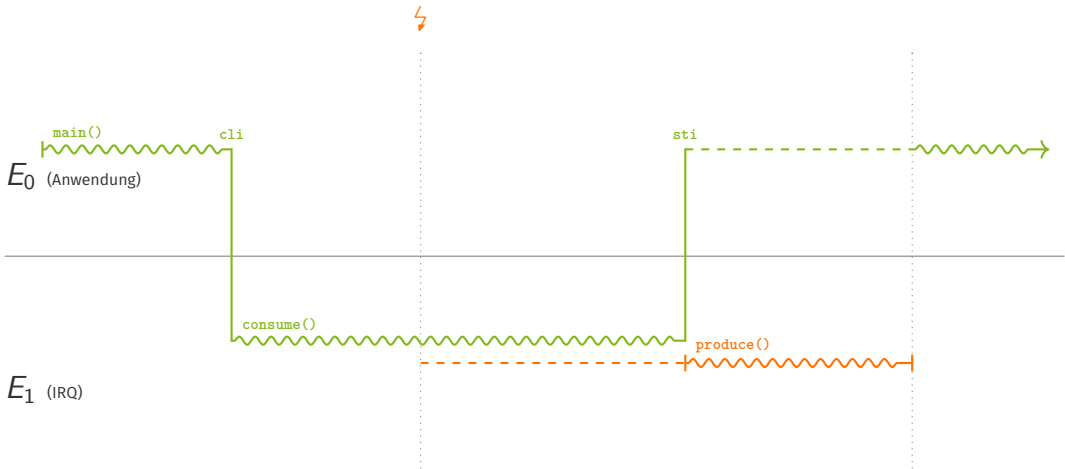
Harte Synchronisation



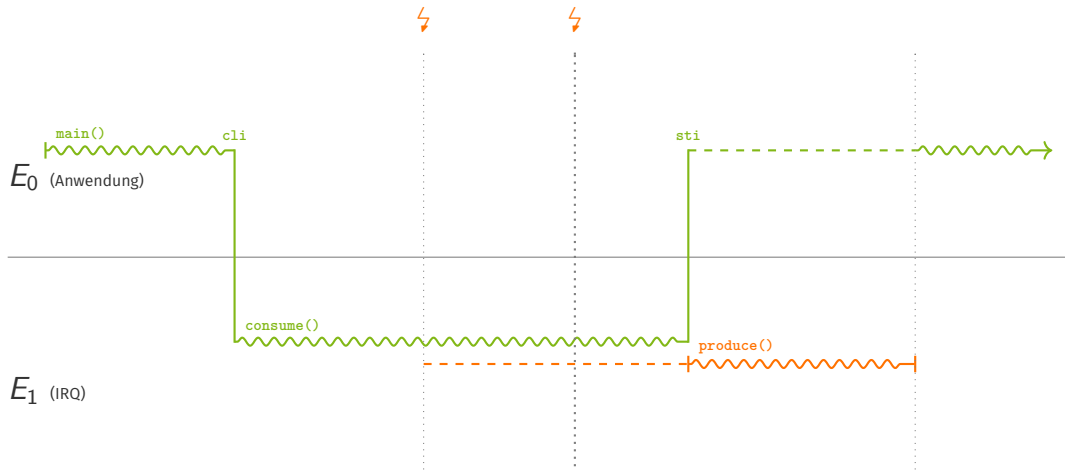
Harte Synchronisation



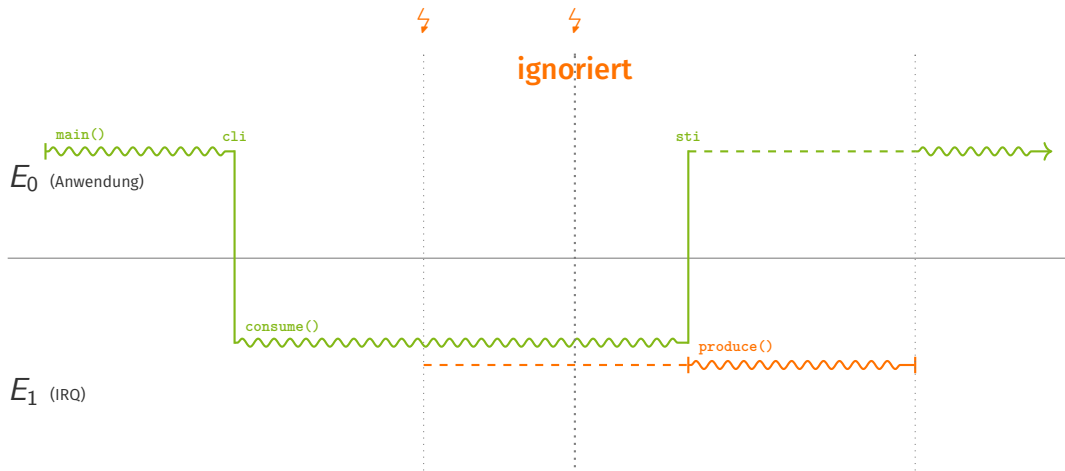
Harte Synchronisation



Harte Synchronisation



Harte Synchronisation



Pessimistischer Ansatz

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



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

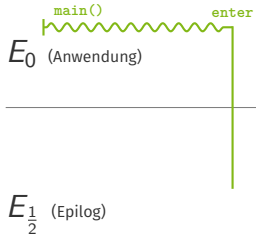
`main()`

 E_0 (Anwendung)

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

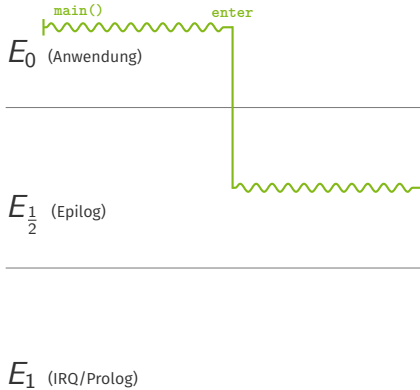
E_1 (IRQ/Prolog)



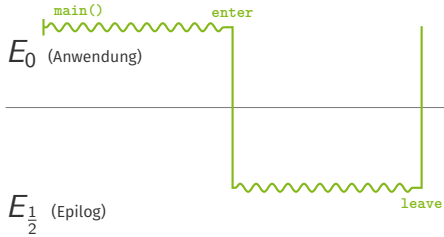
Prolog/Epilog-Modell



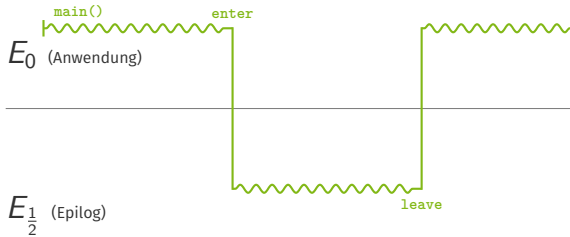
Prolog/Epilog-Modell



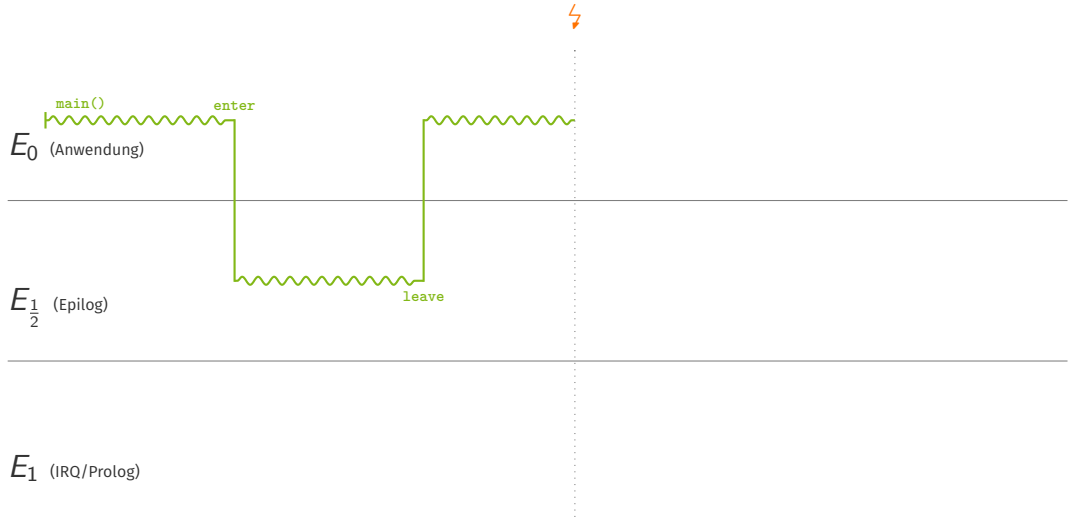
Prolog/Epilog-Modell



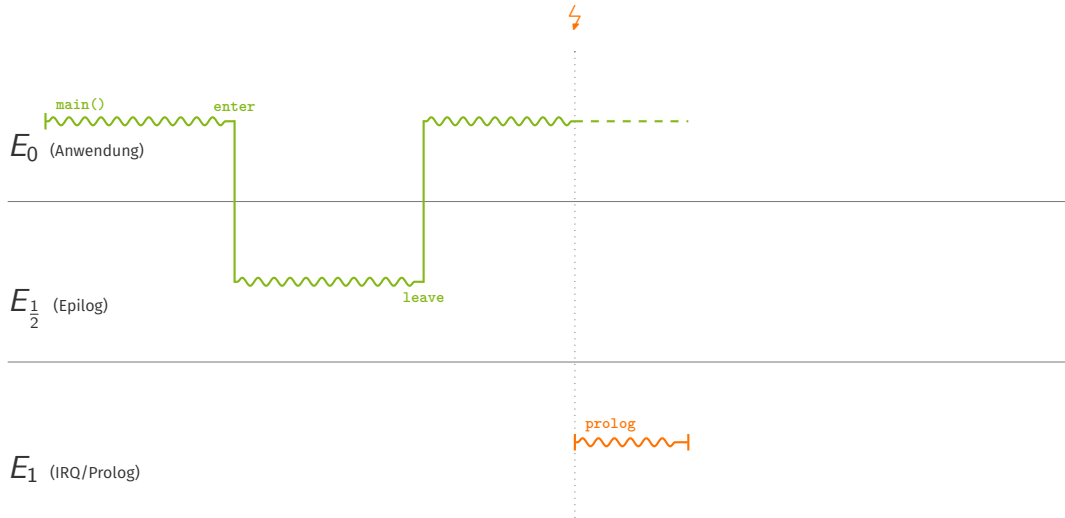
Prolog/Epilog-Modell



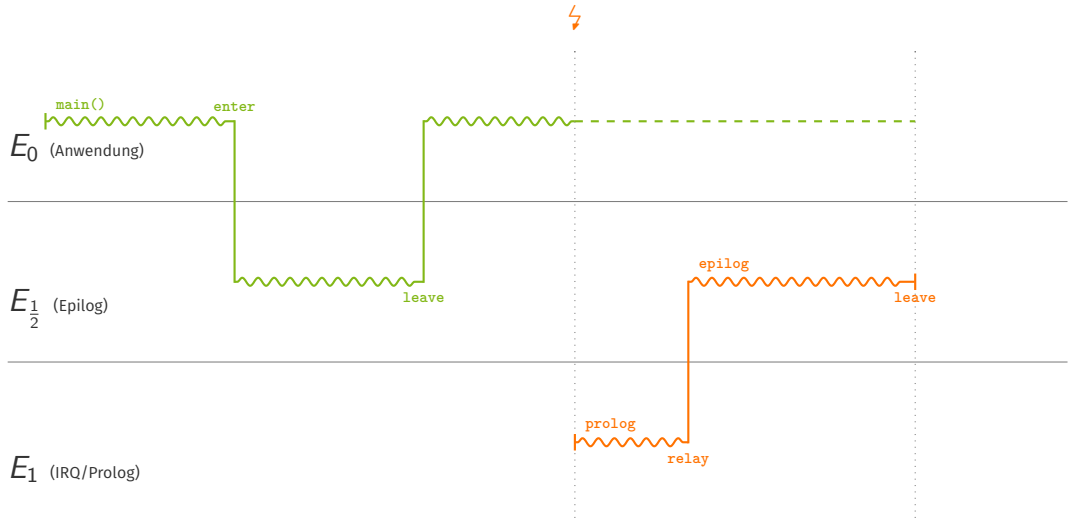
Prolog/Epilog-Modell



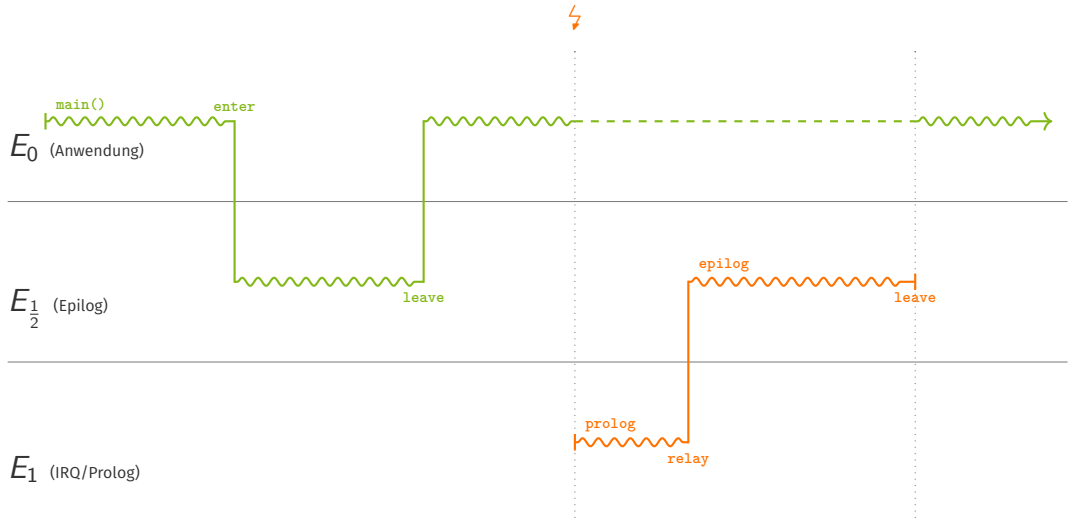
Prolog/Epilog-Modell



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**



```
01 main(){
02     while(1){
03         enter();
04         consume();
05         leave();
06     }
07 }
```

```
01 epilog(){
02
03     // ...
04     produce();
05     // ...
06
07 }
```



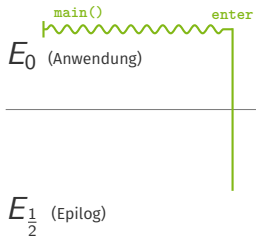
`main()`

 E_0 (Anwendung)

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

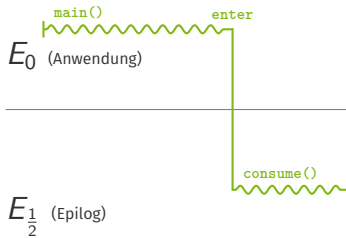
E_1 (IRQ/Prolog)



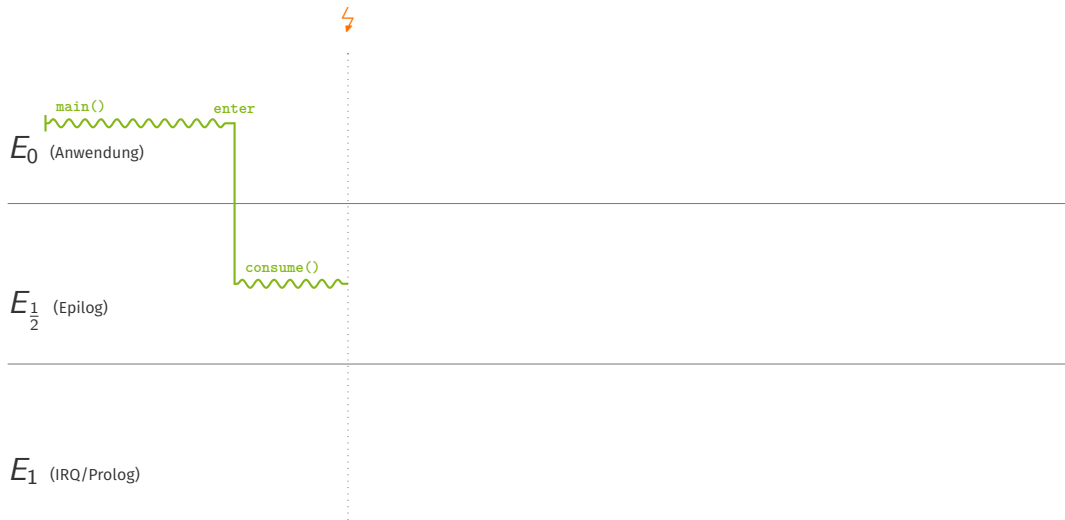
Umsetzung



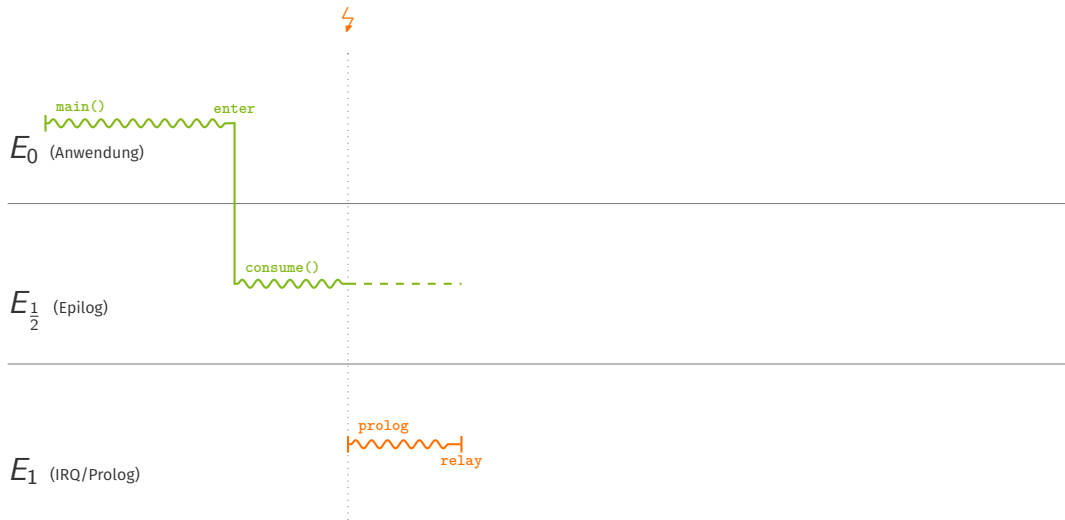
Umsetzung



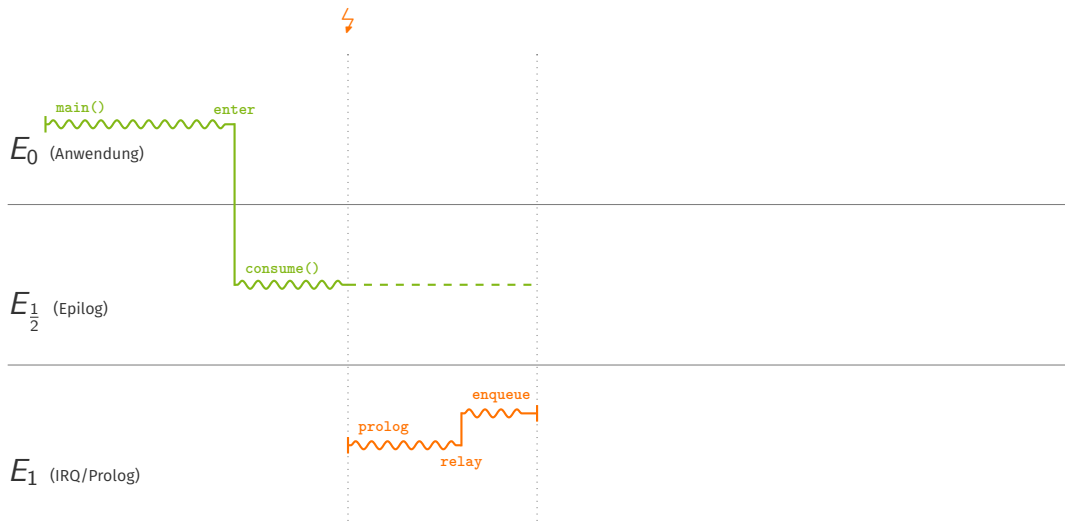
Umsetzung



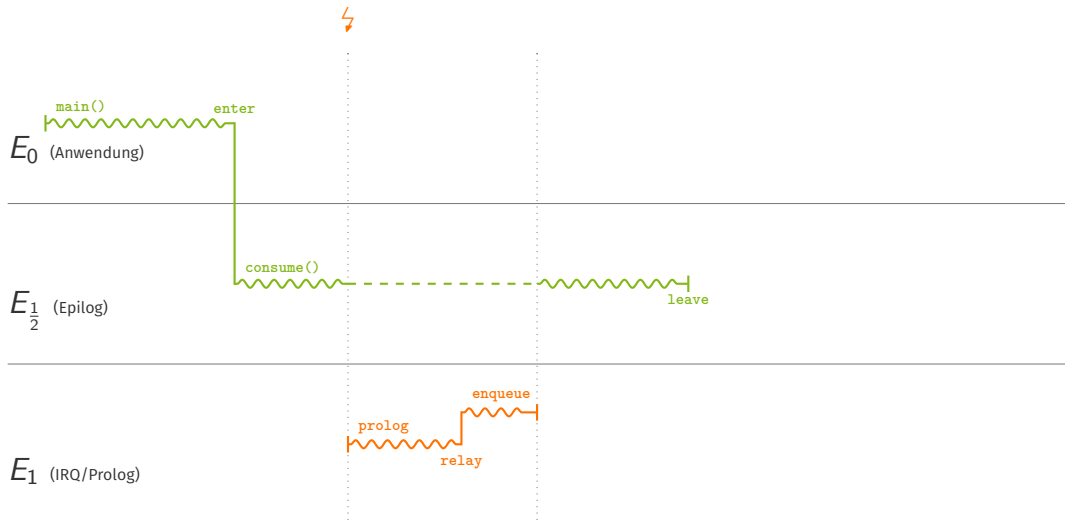
Umsetzung



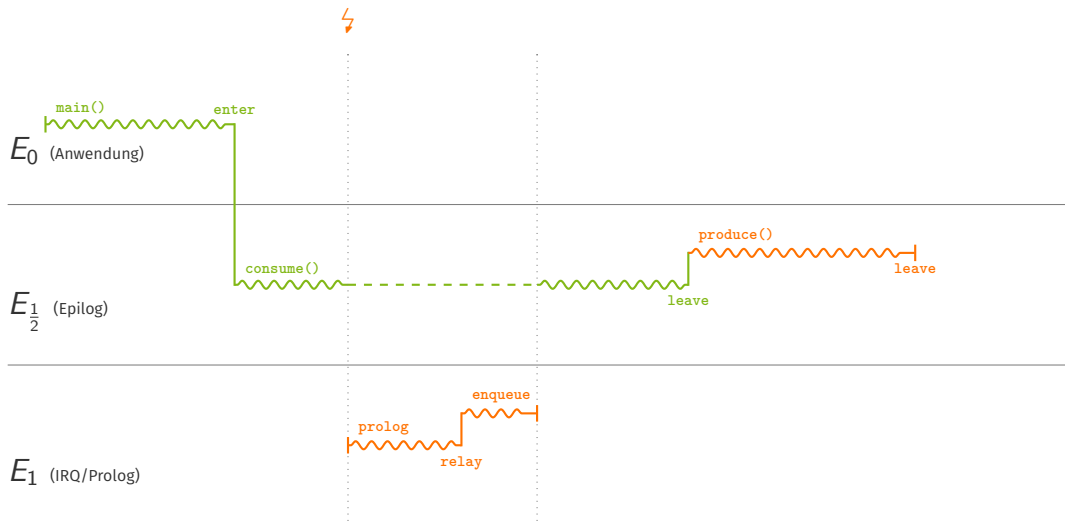
Umsetzung



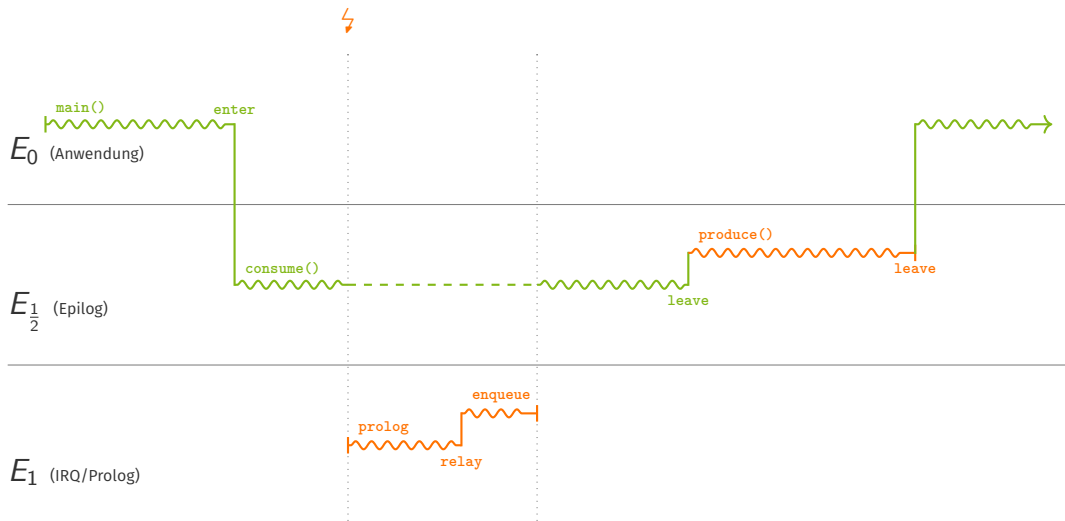
Umsetzung



Umsetzung



Umsetzung



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*



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 

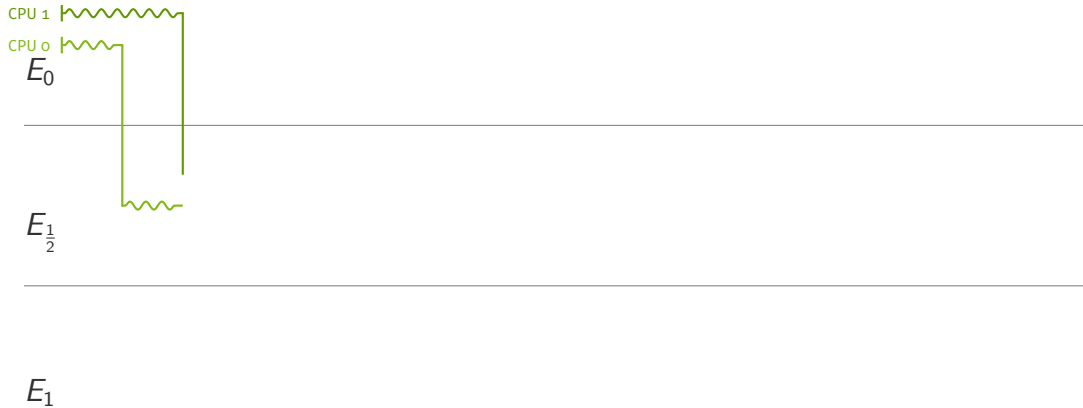
E_0

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

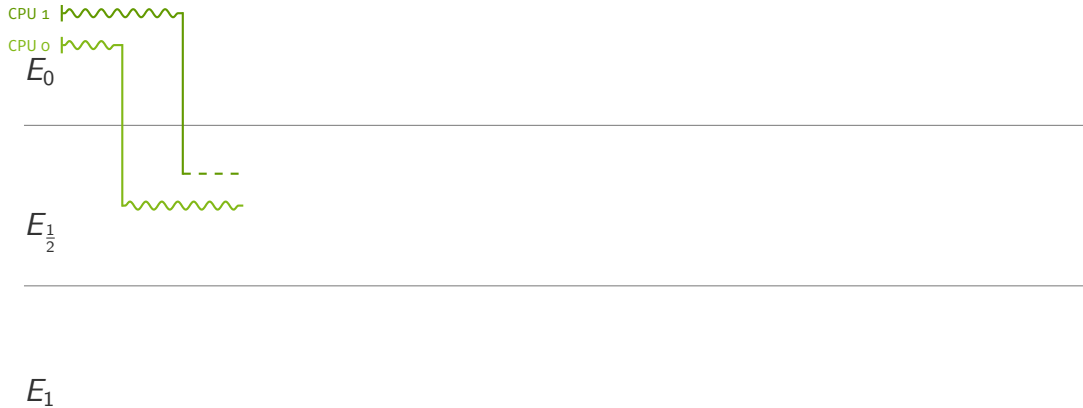
E_1



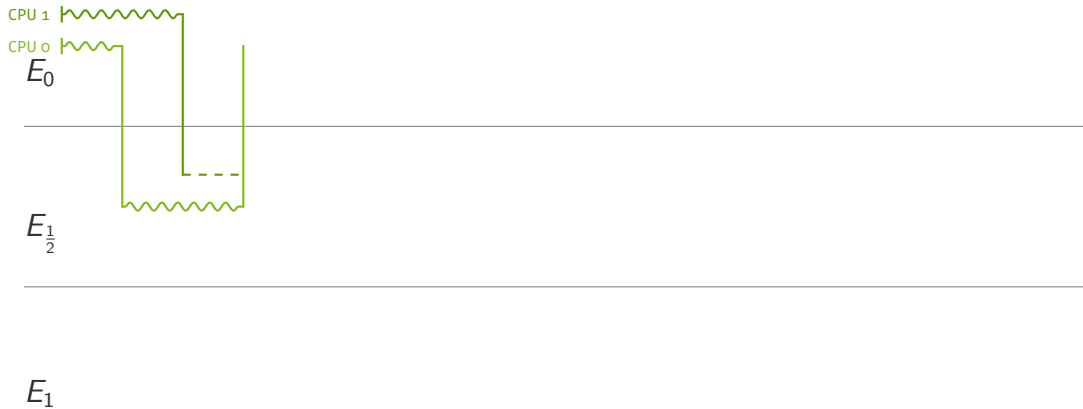
Beispiel für Mehrkernprozessoren



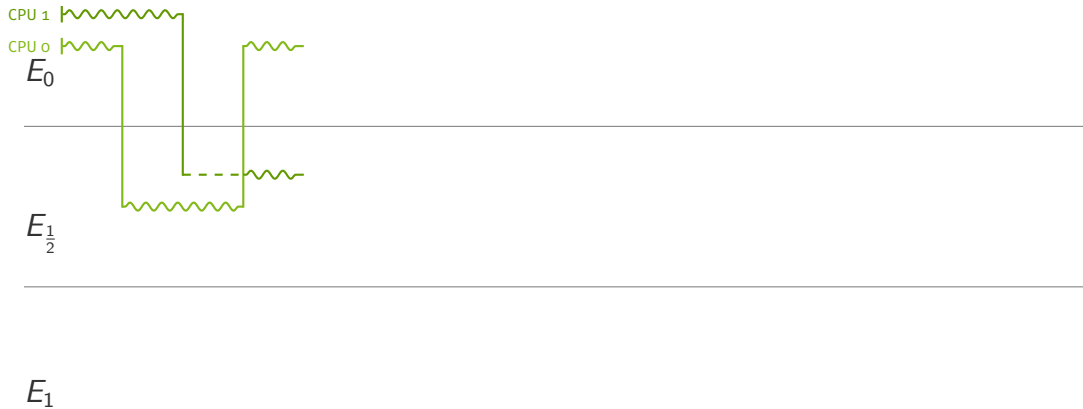
Beispiel für Mehrkernprozessoren



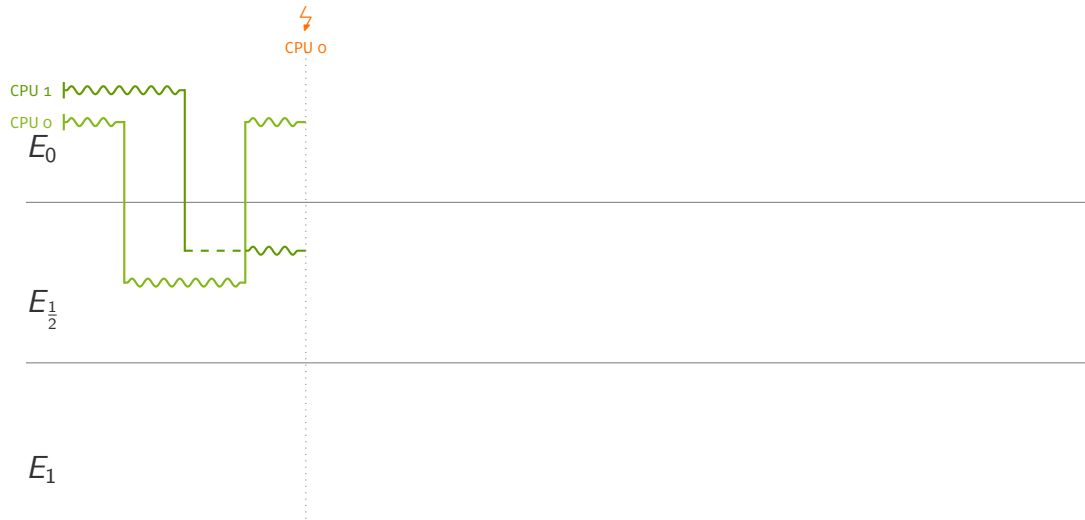
Beispiel für Mehrkernprozessoren



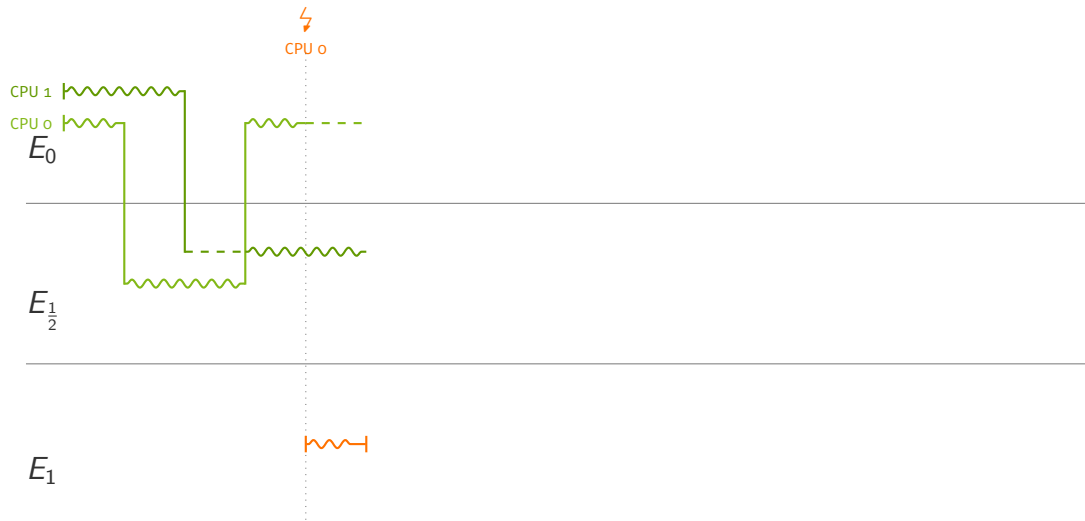
Beispiel für Mehrkernprozessoren



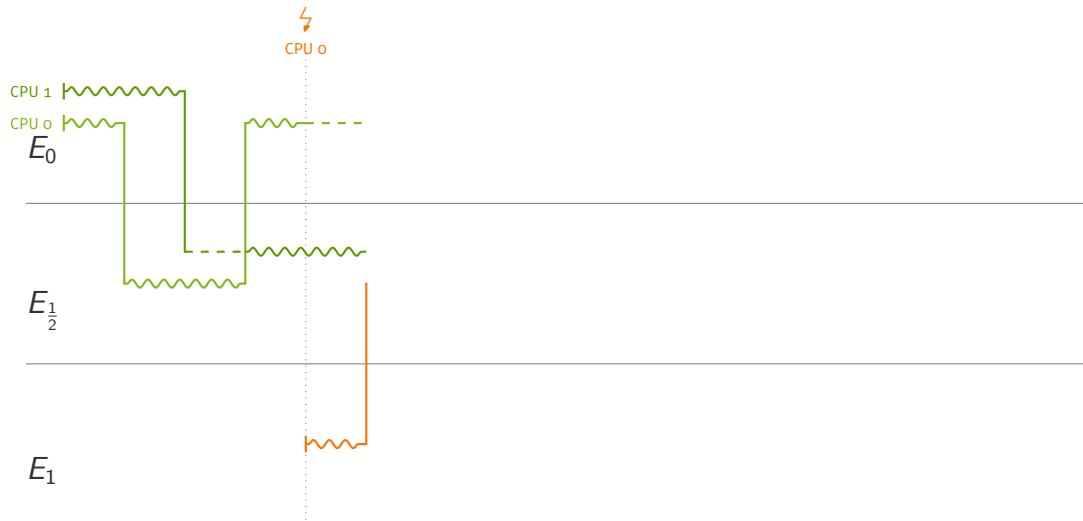
Beispiel für Mehrkernprozessoren



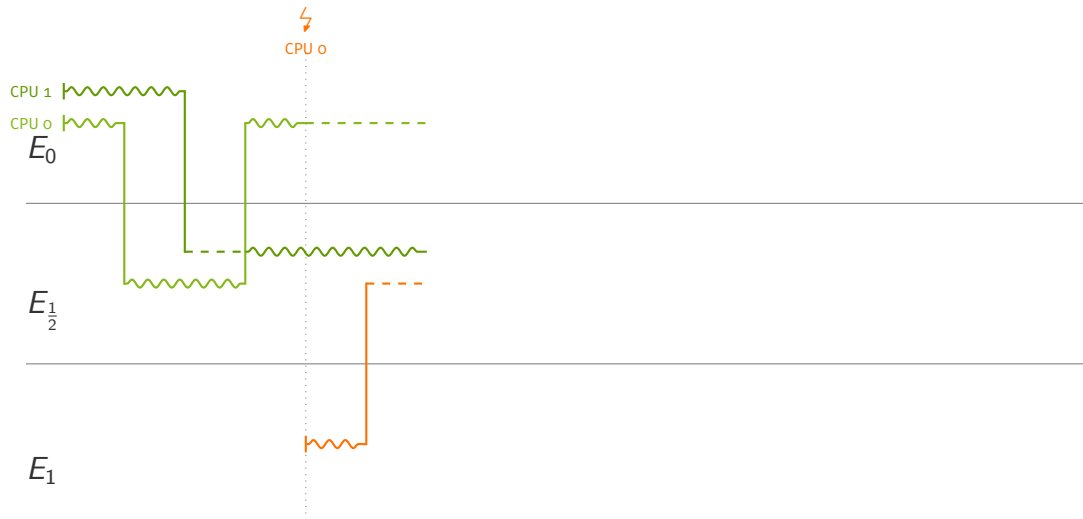
Beispiel für Mehrkernprozessoren



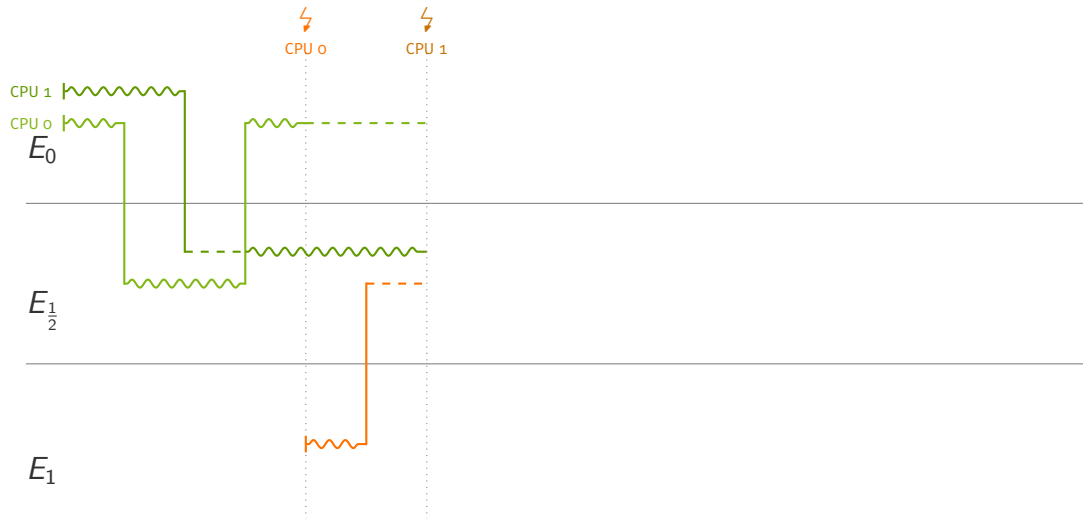
Beispiel für Mehrkernprozessoren



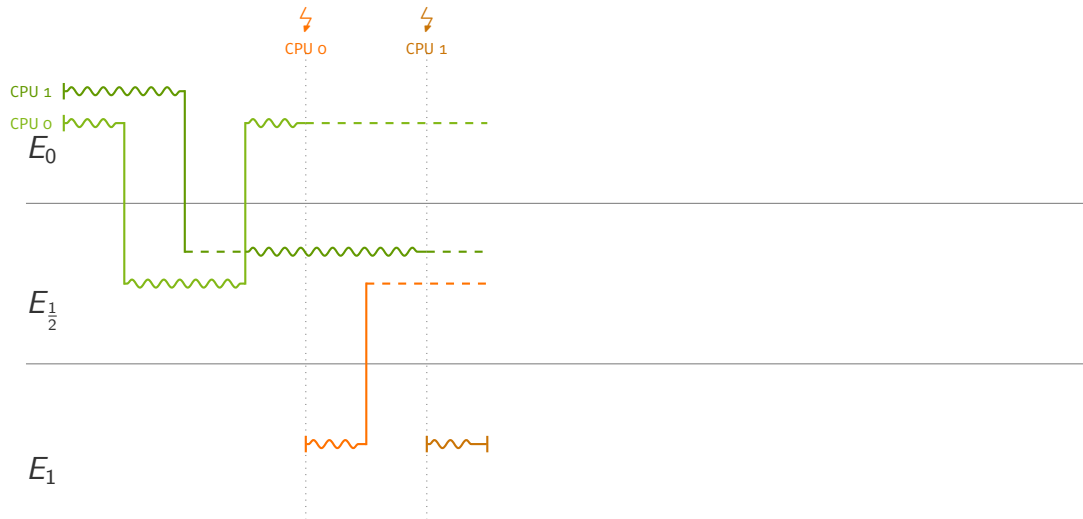
Beispiel für Mehrkernprozessoren



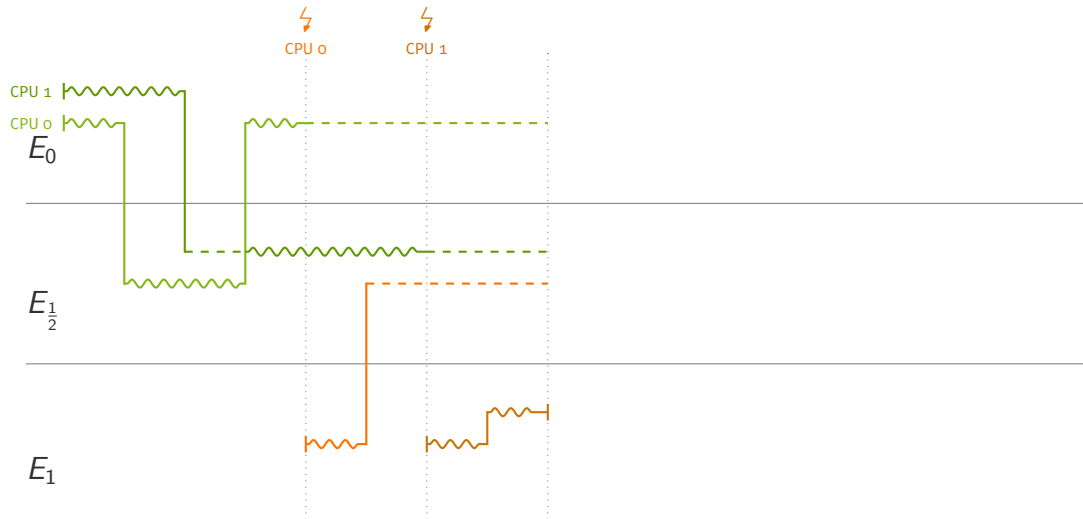
Beispiel für Mehrkernprozessoren



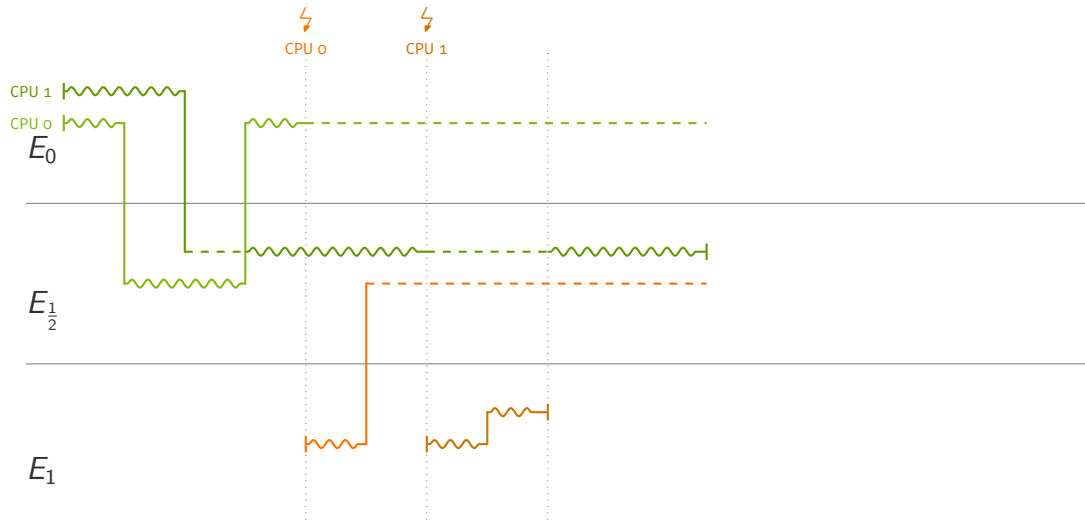
Beispiel für Mehrkernprozessoren



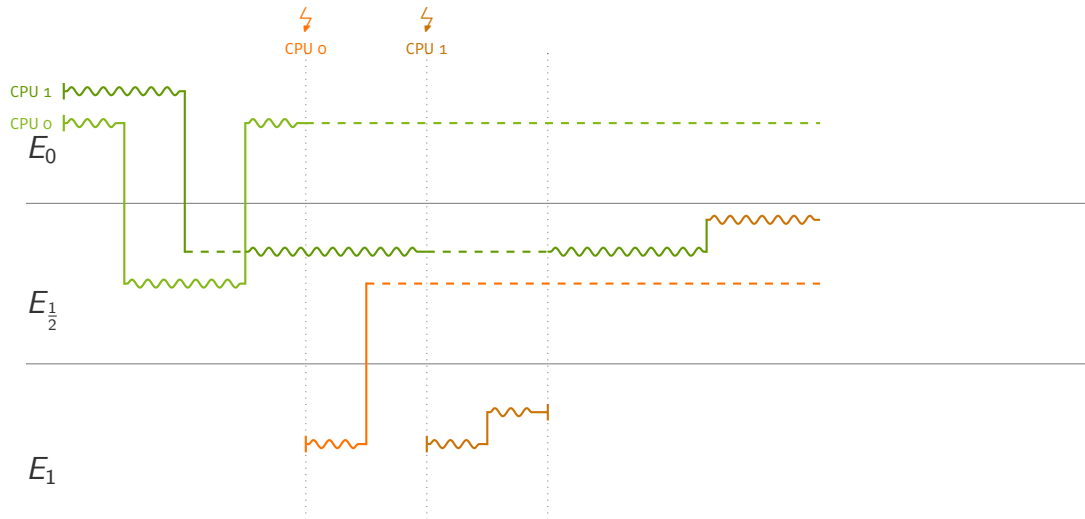
Beispiel für Mehrkernprozessoren



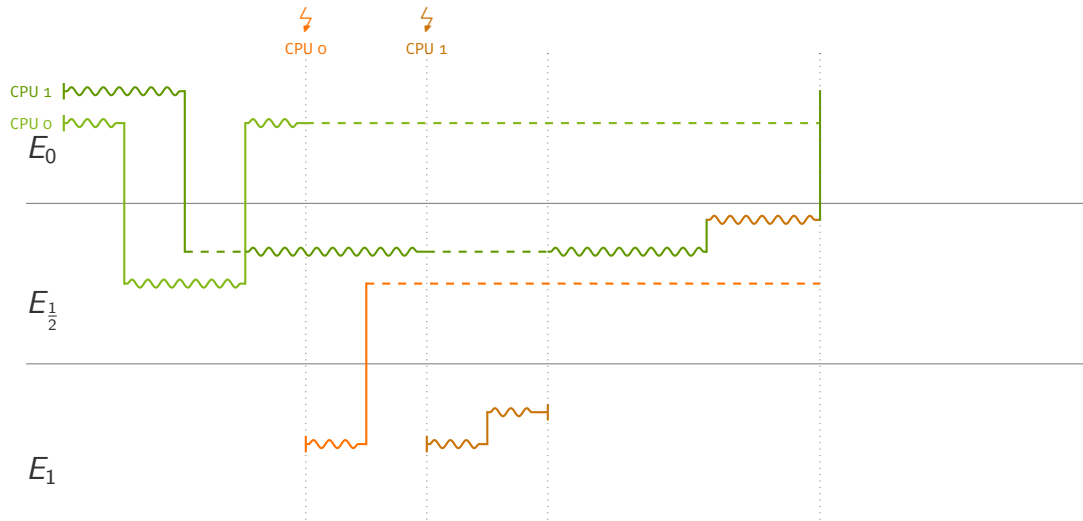
Beispiel für Mehrkernprozessoren



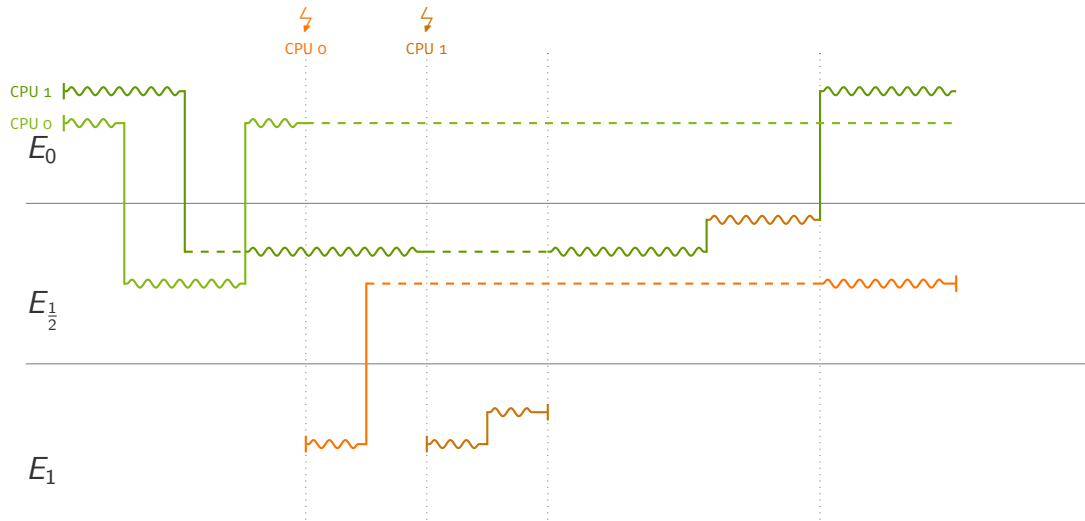
Beispiel für Mehrkernprozessoren



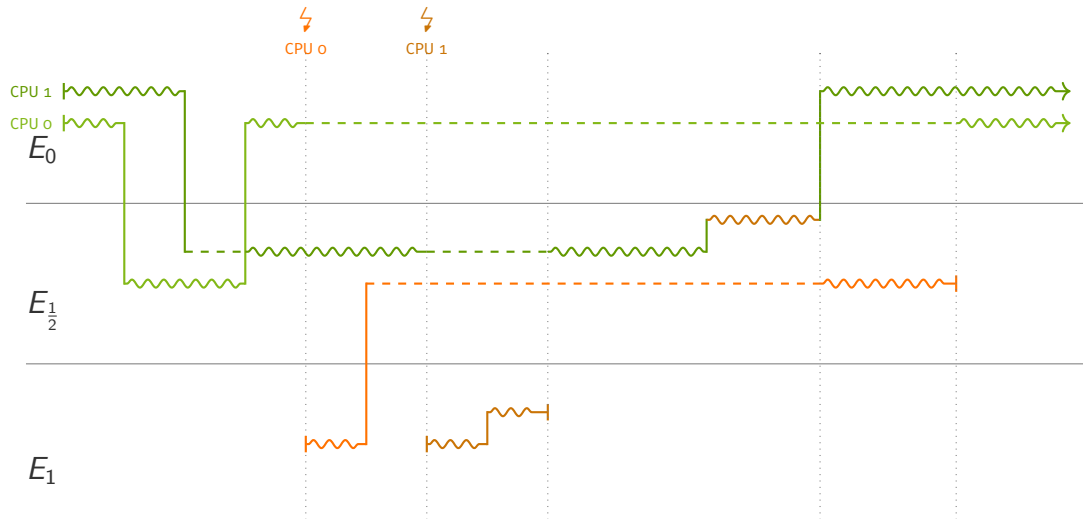
Beispiel für Mehrkernprozessoren



Beispiel für Mehrkernprozessoren



Beispiel für Mehrkernprozessoren



Fragen?

Abgabe der Aufgabe bis Mittwoch, den 27. November