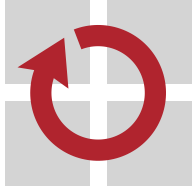
Übung zu Betriebssysteme

Interruptsynchronisation

22. & 24. November 2017

Andreas Ziegler
Bernhard Heinloth

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Motivation

Interrupts verändern (potenziell) den Zustand des Systems

Motivation

Interrupts verändern (potenziell) den Zustand des Systems

```
main(){
    consume();
}
```

```
IRQ_handler(){
    produce();
}
```

Keine Synchronisation

E_0 (Anwendung)

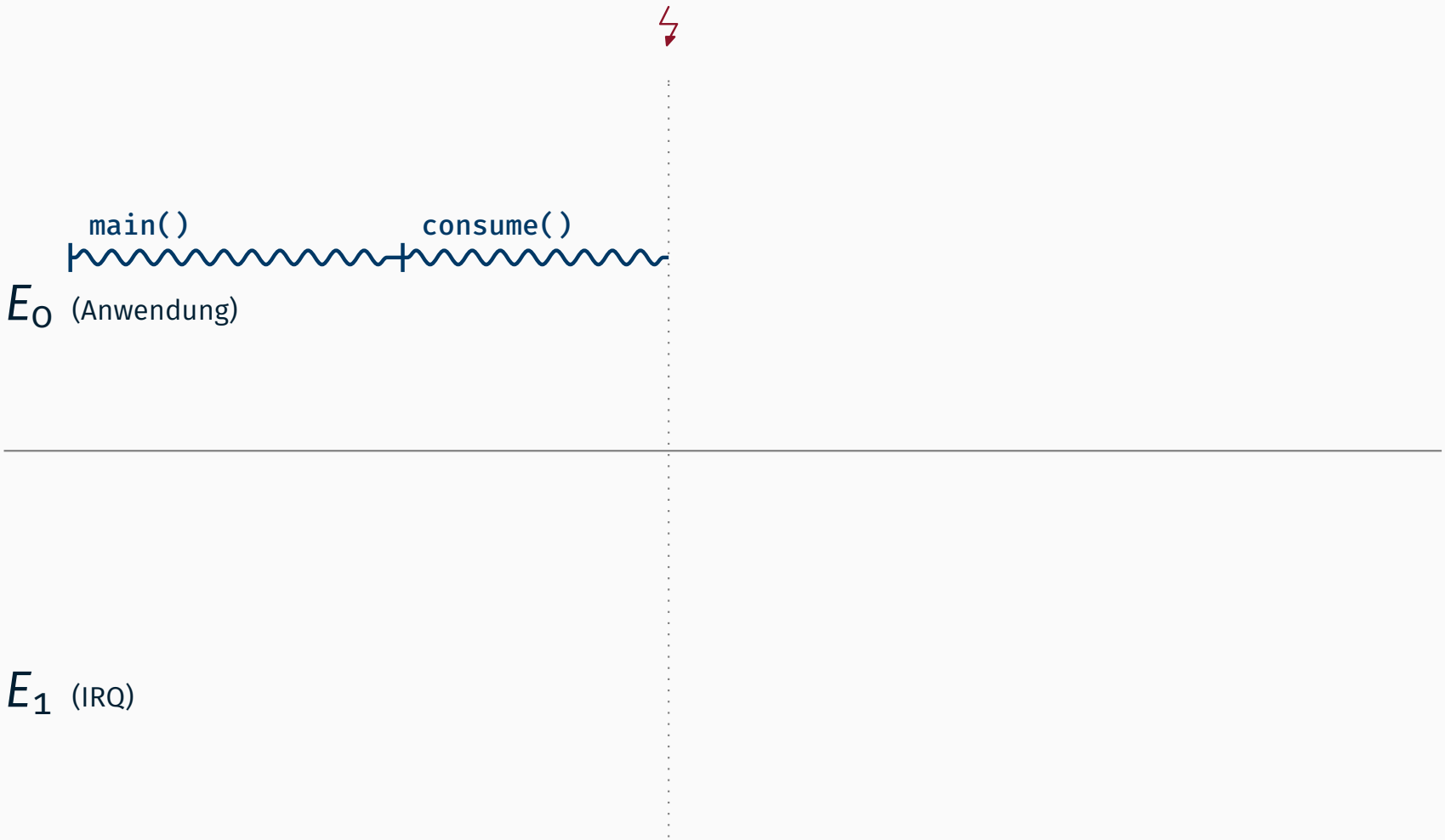
E_1 (IRQ)

Keine Synchronisation

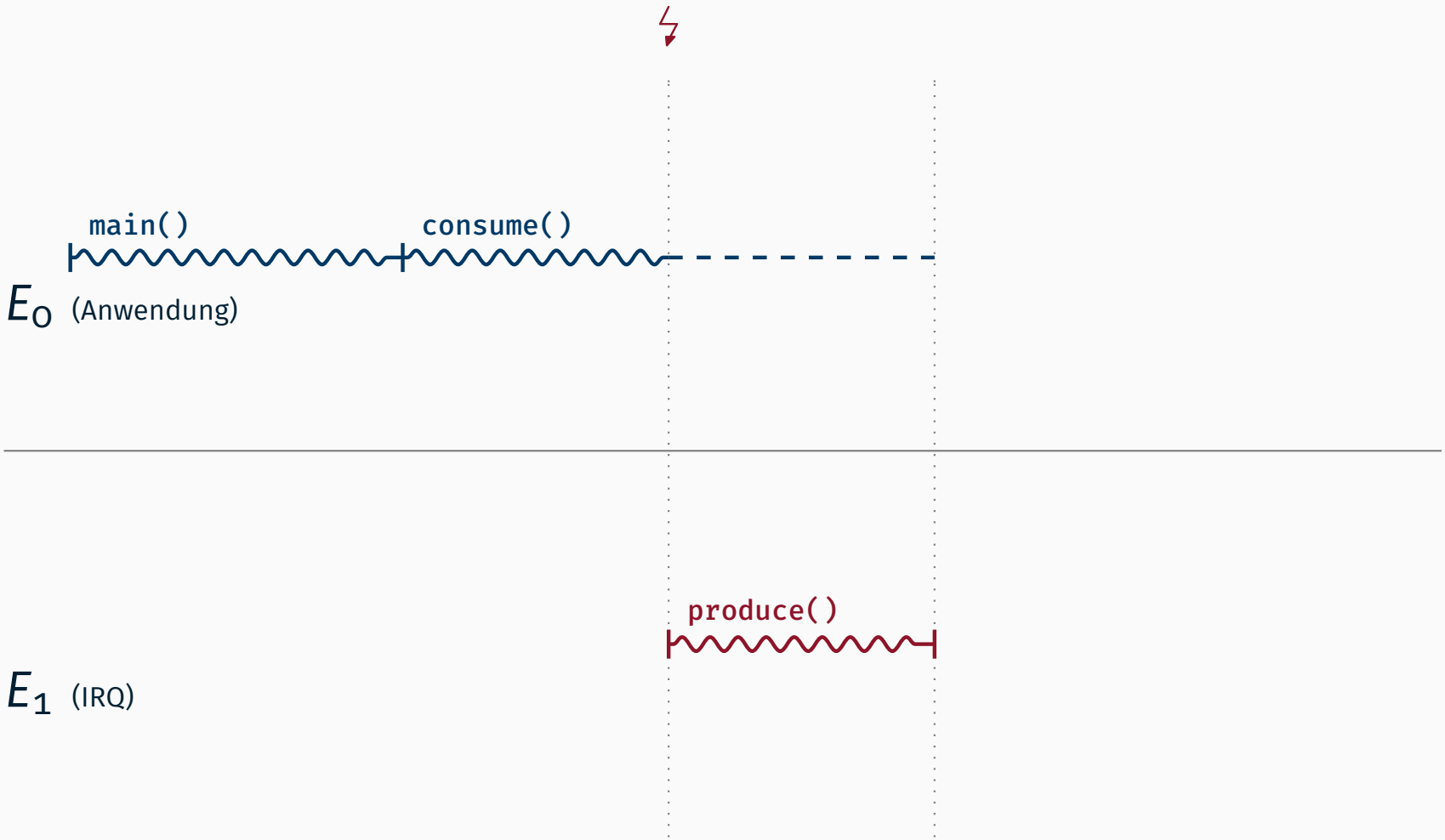
E_0 (Anwendung)

E_1 (IRQ)

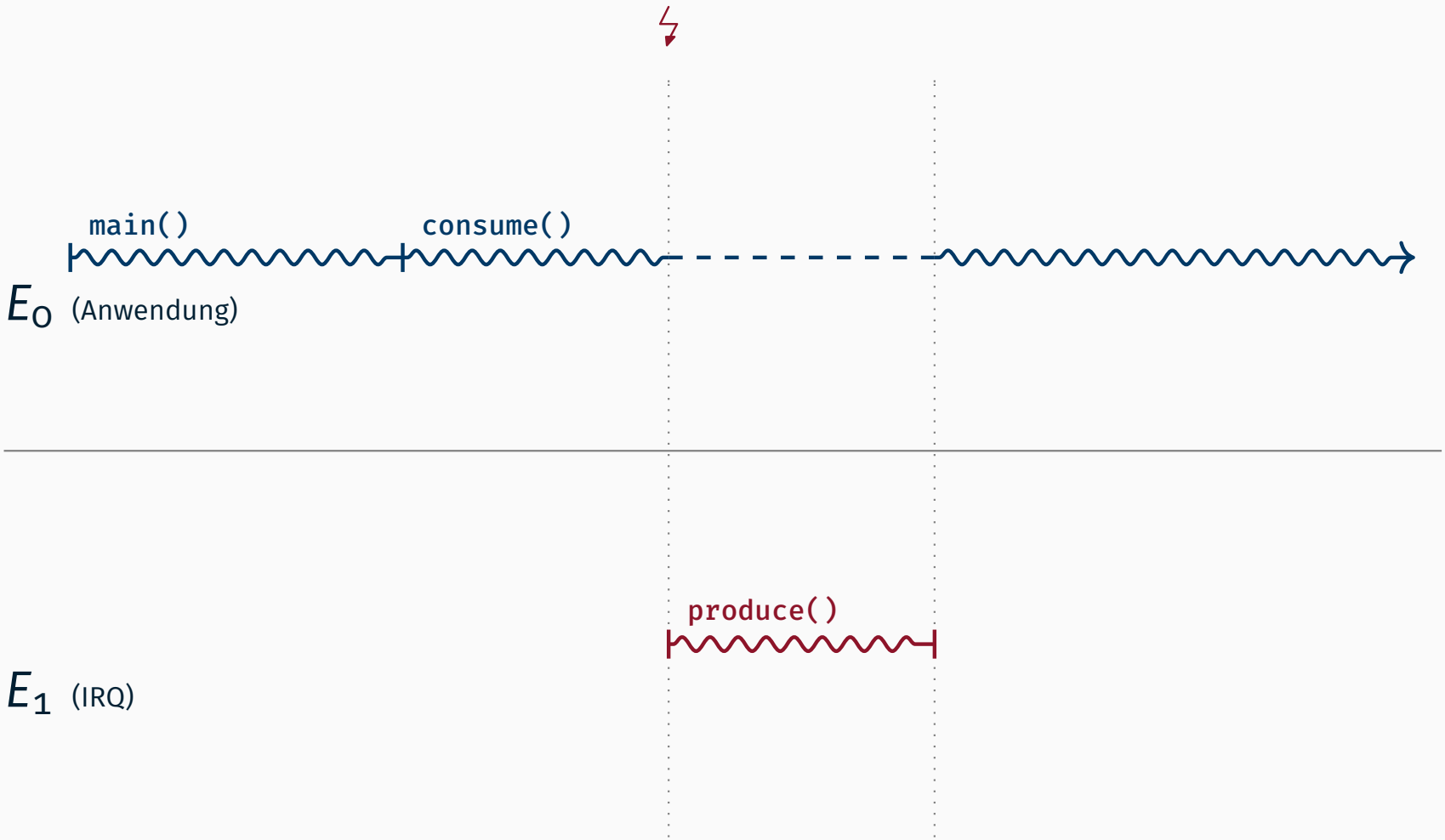
Keine Synchronisation



Keine Synchronisation



Keine Synchronisation



Keine Synchronisation

Was ist mit

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

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

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

?

Keine Synchronisation

Was ist mit

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

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

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

? Hier ist ein **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;
}
```

bewährtes Hausmittel: Mutex

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;
}
```

weiche Synchronisation

Optimistischer Ansatz

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

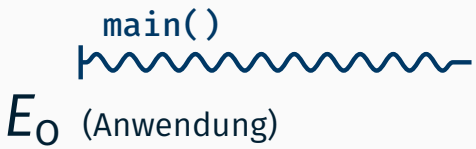
weiche Synchronisation

Optimistischer Ansatz

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

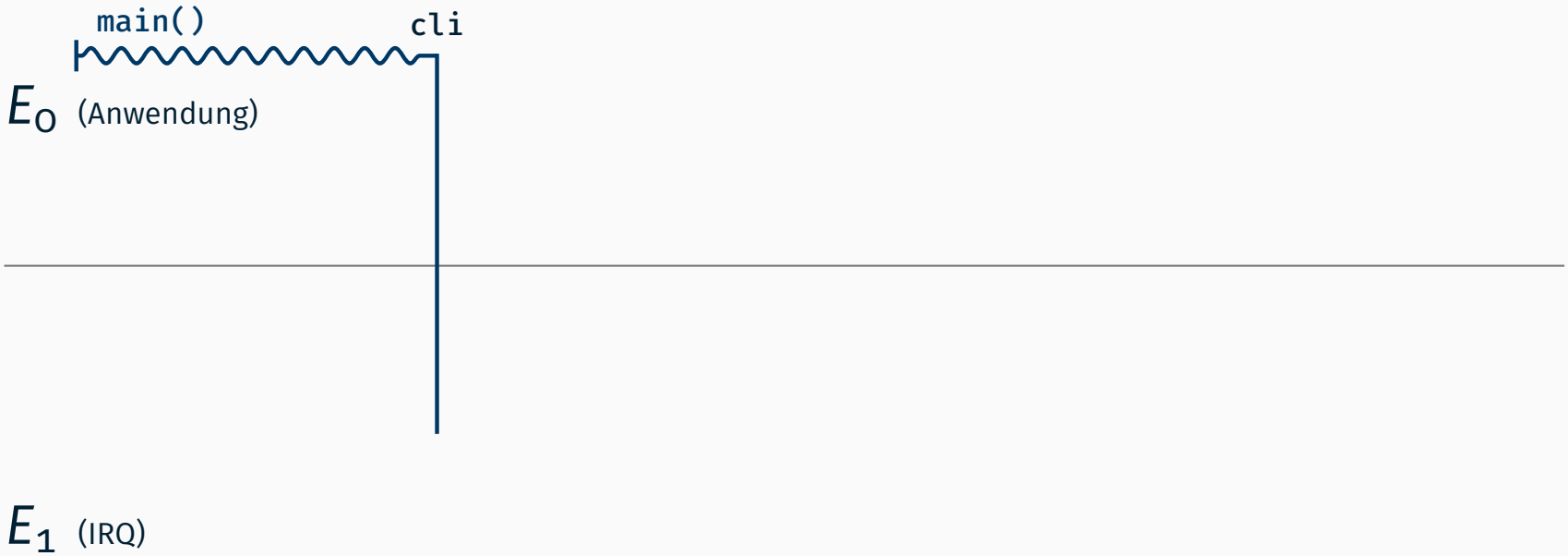
Betriebssystemarchitekt sollte Treiber- und Anwendungsentwicklern entgegenkommen
$\rightarrow$ für Erfolg des Betriebsystems entscheidend

harte Synchronisation

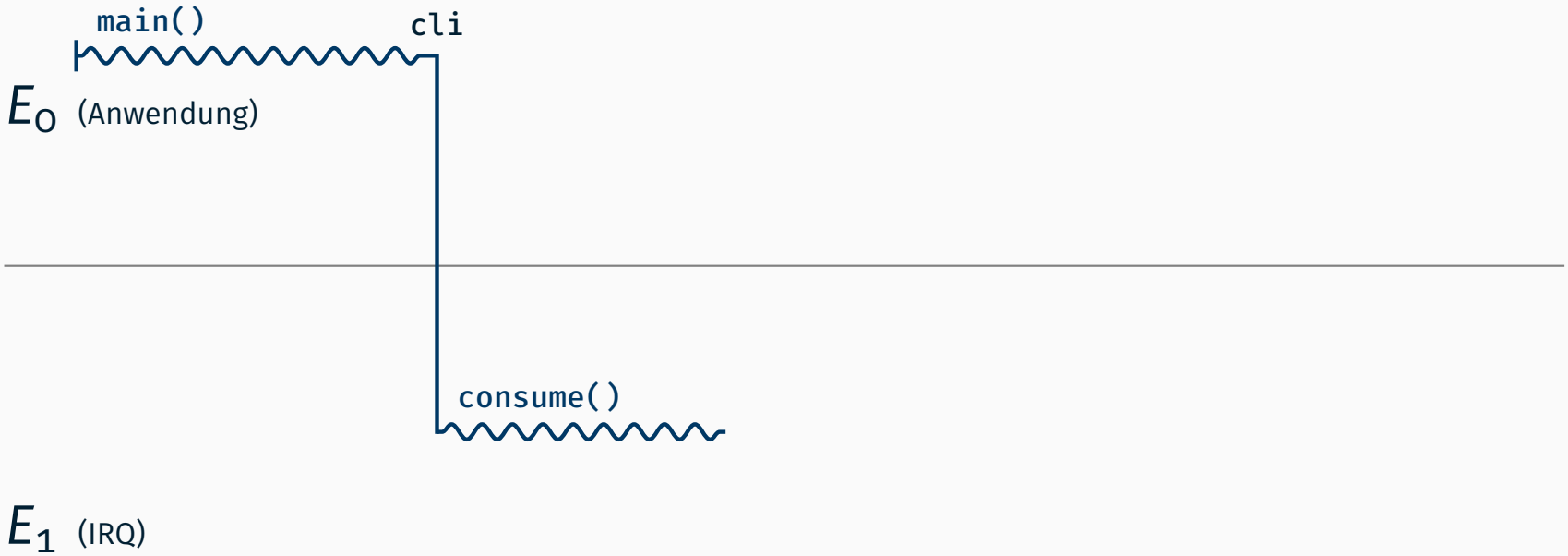


E_1 (IRQ)

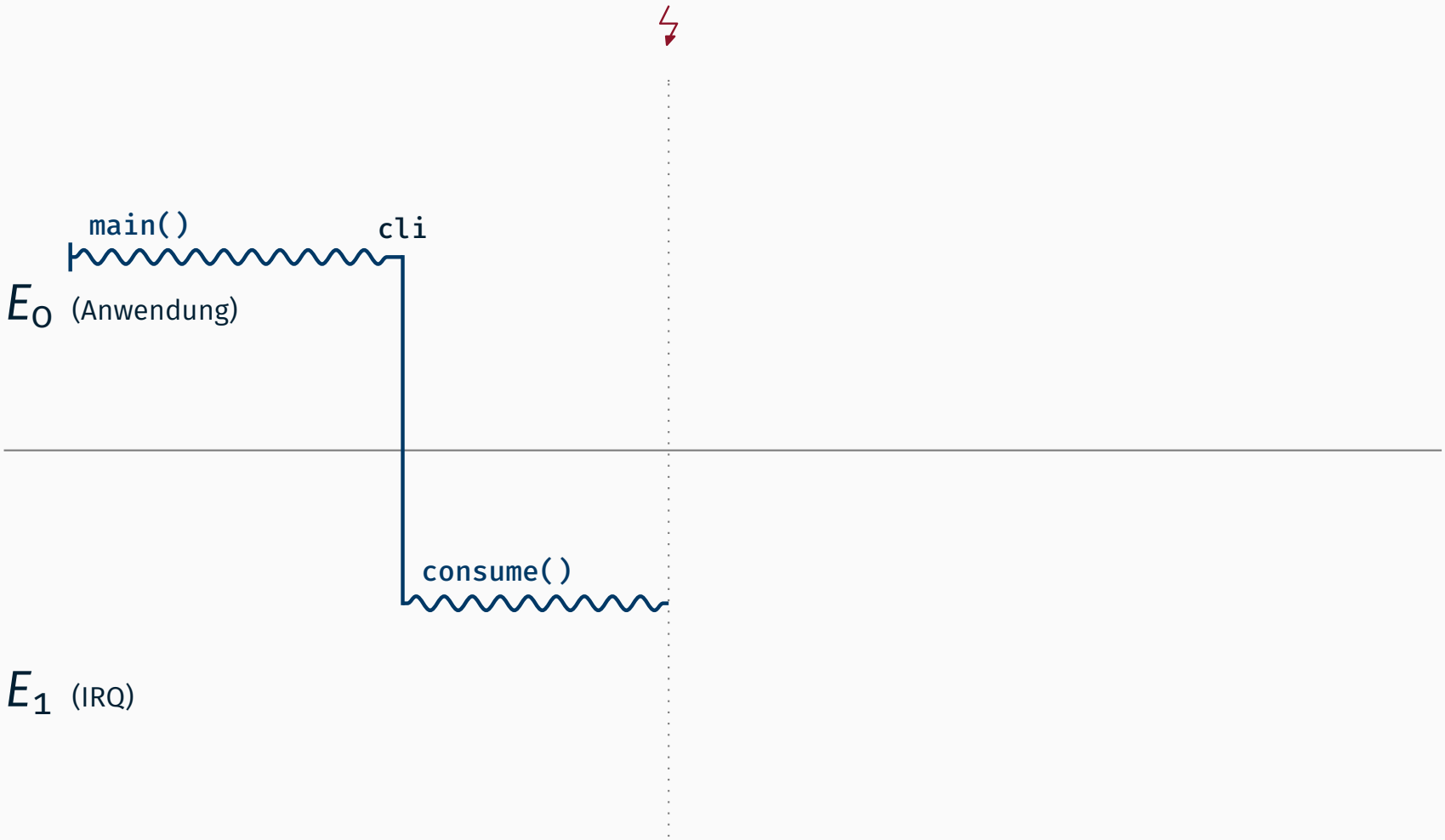
harte Synchronisation



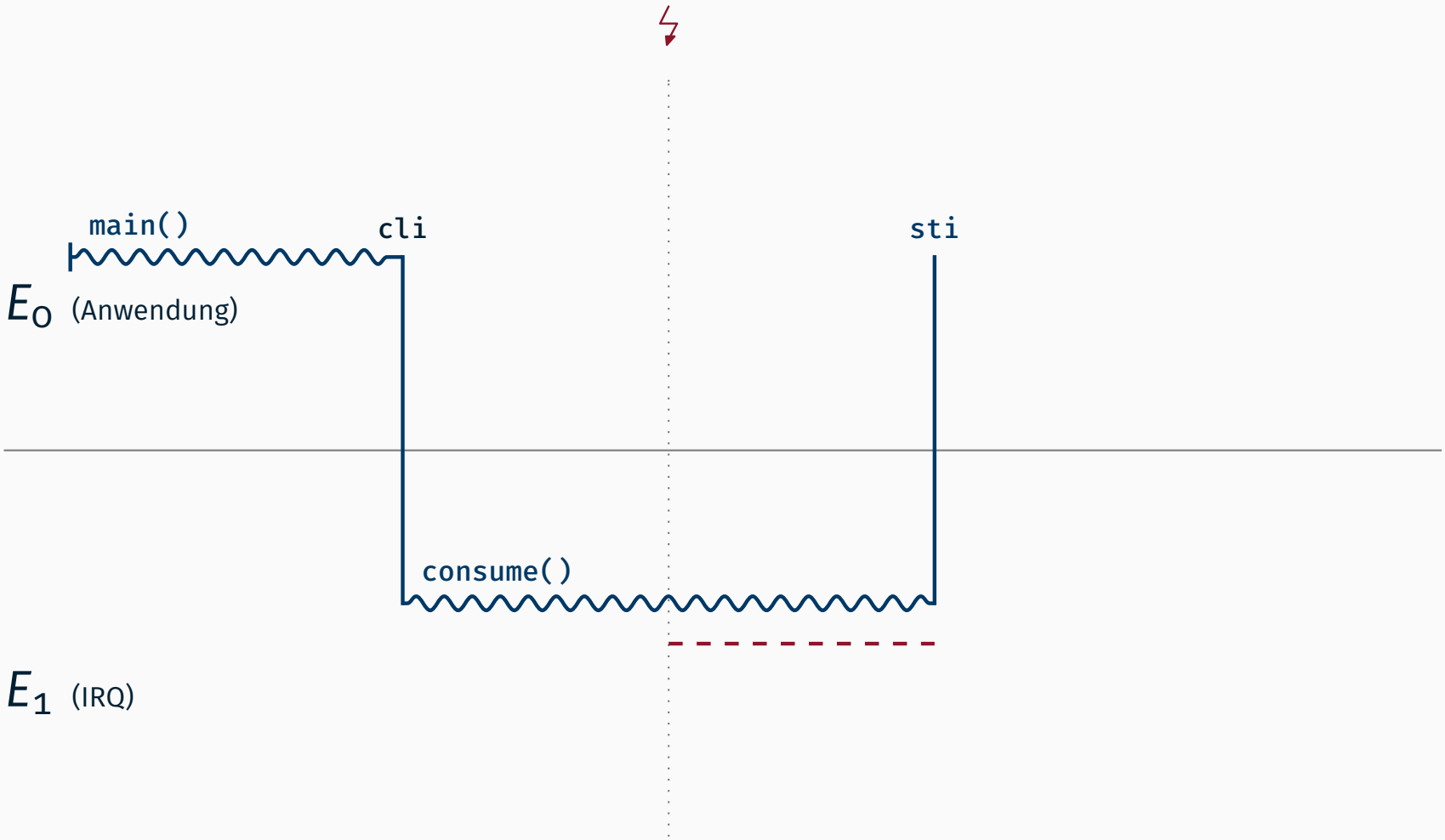
harte Synchronisation



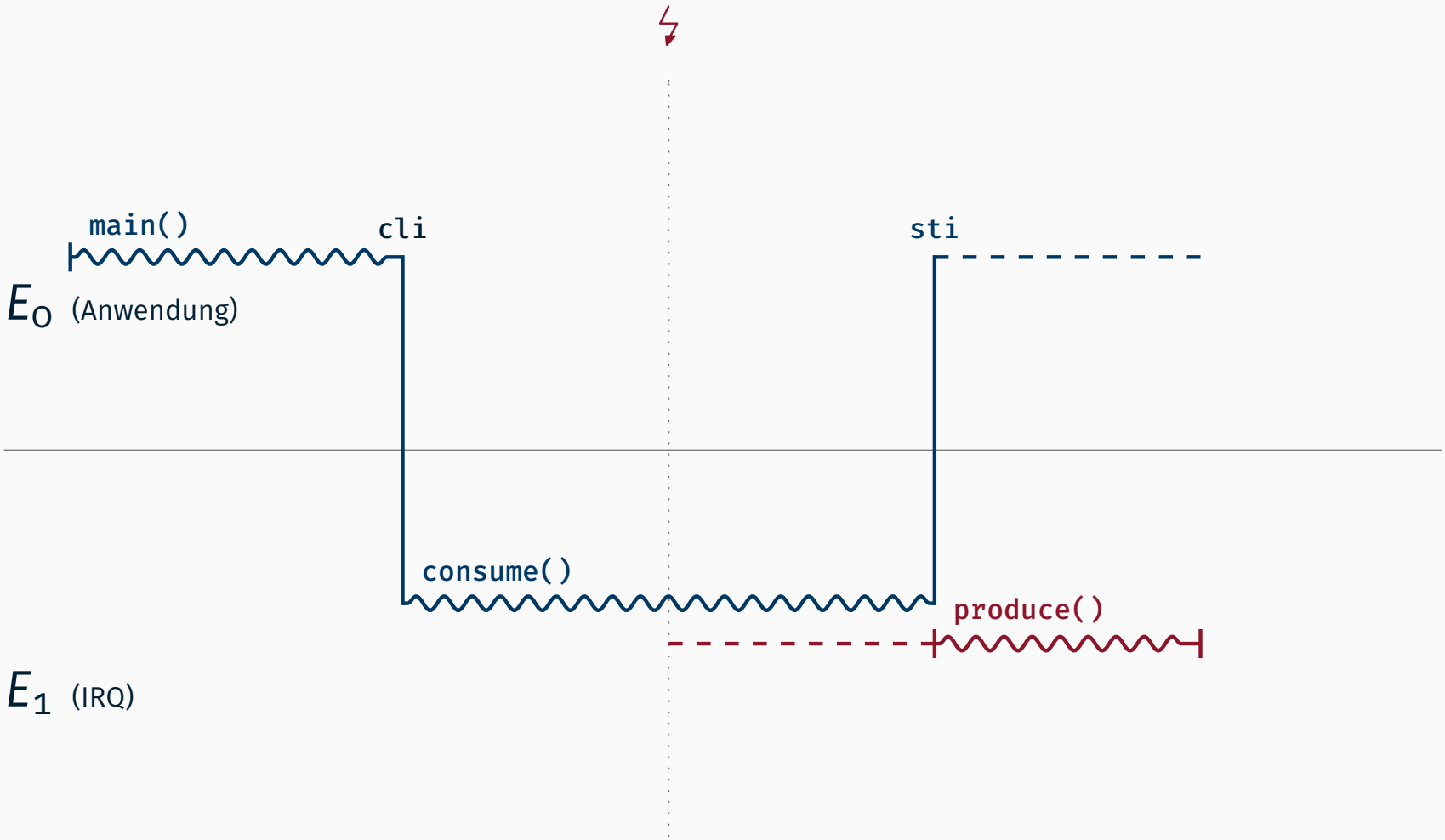
harte Synchronisation



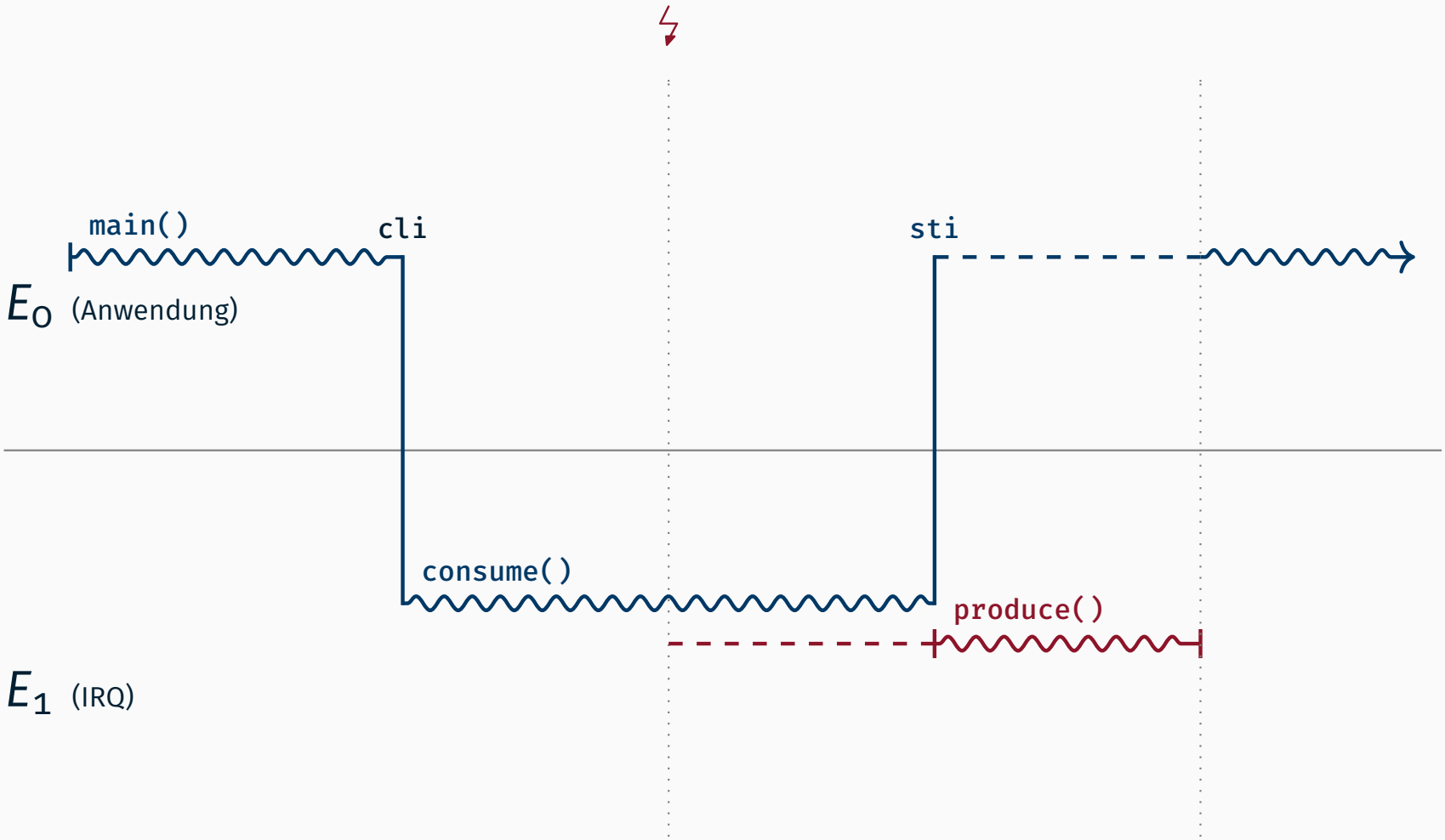
harte Synchronisation



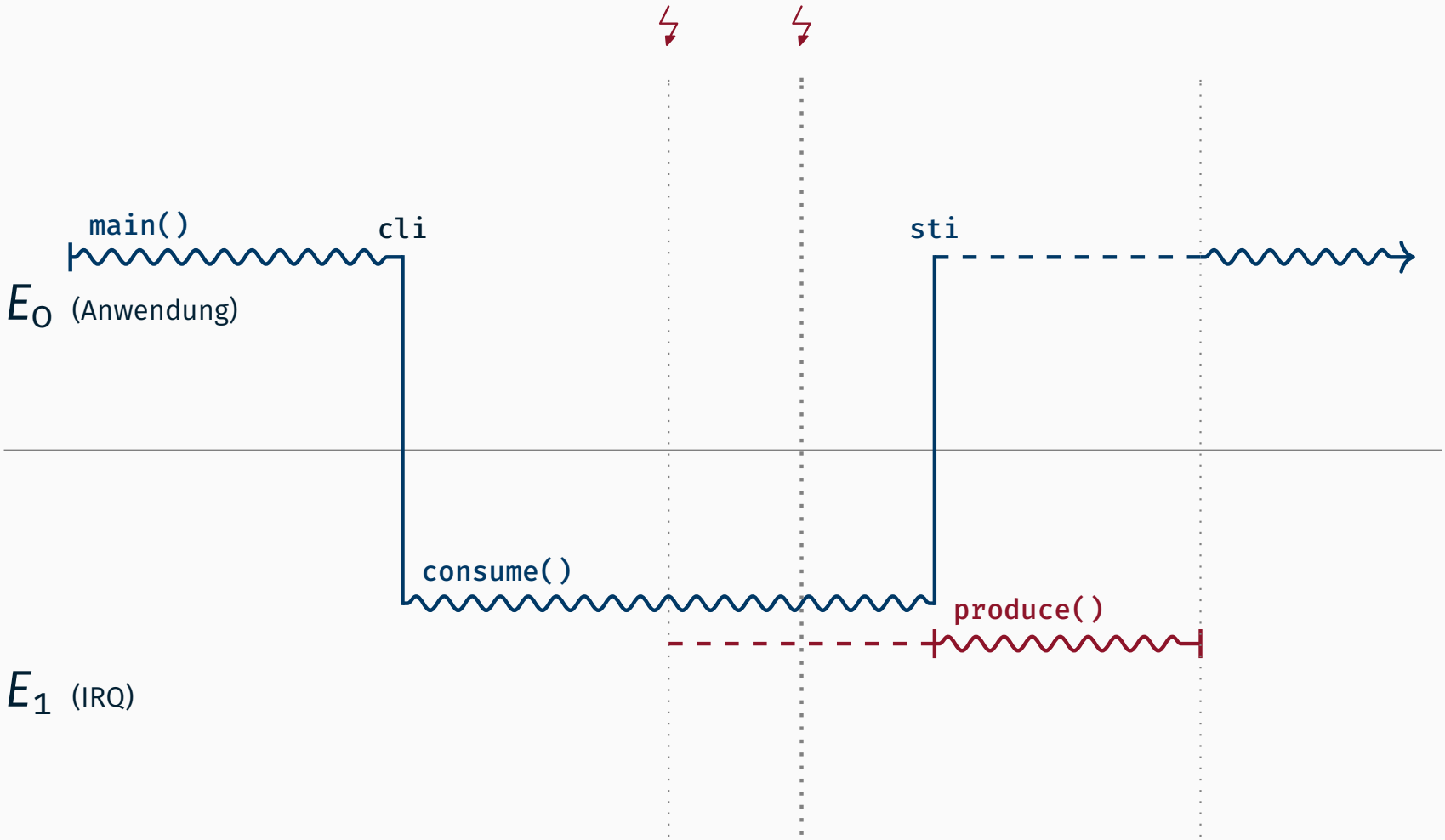
harte Synchronisation



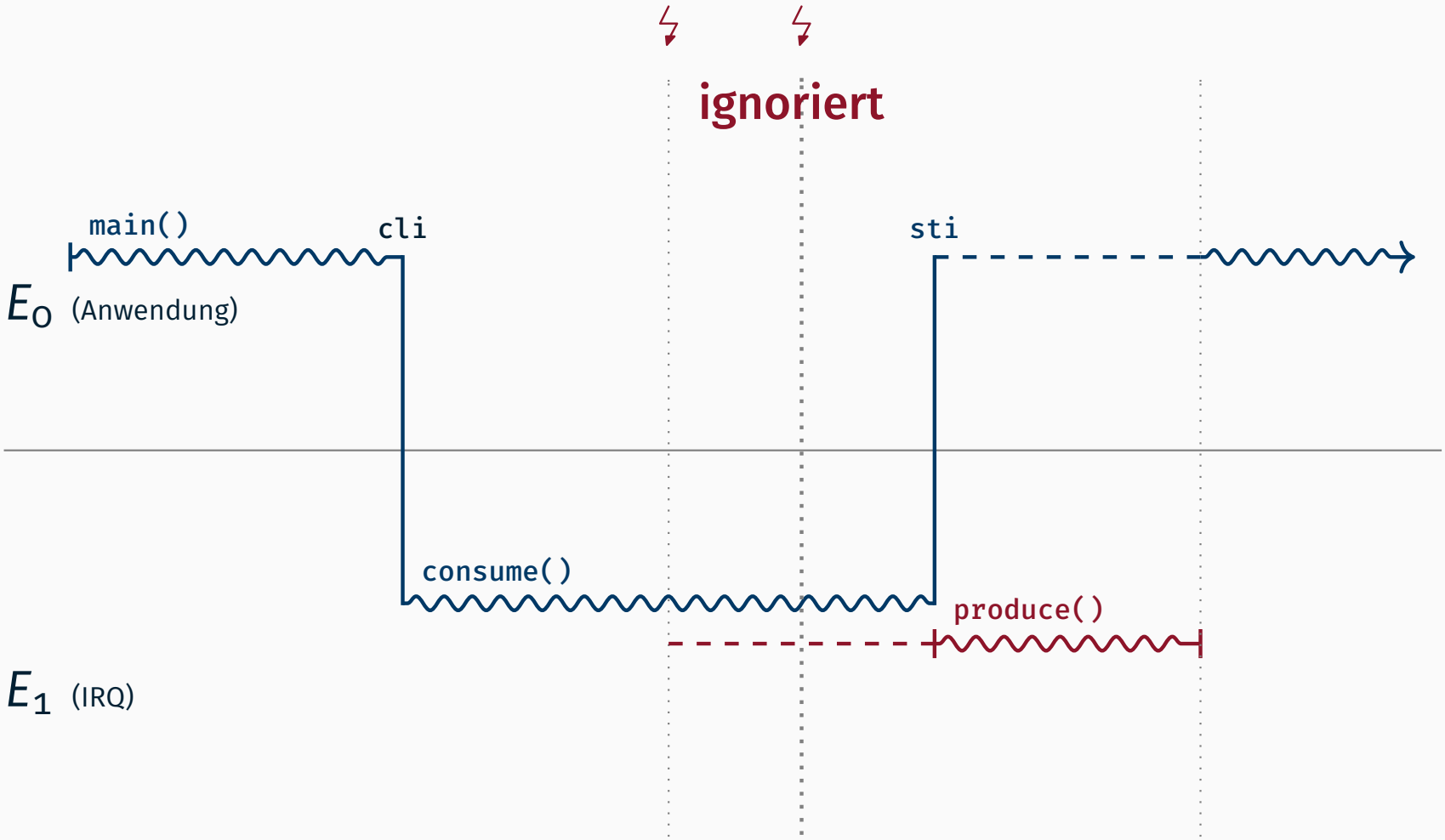
harte Synchronisation



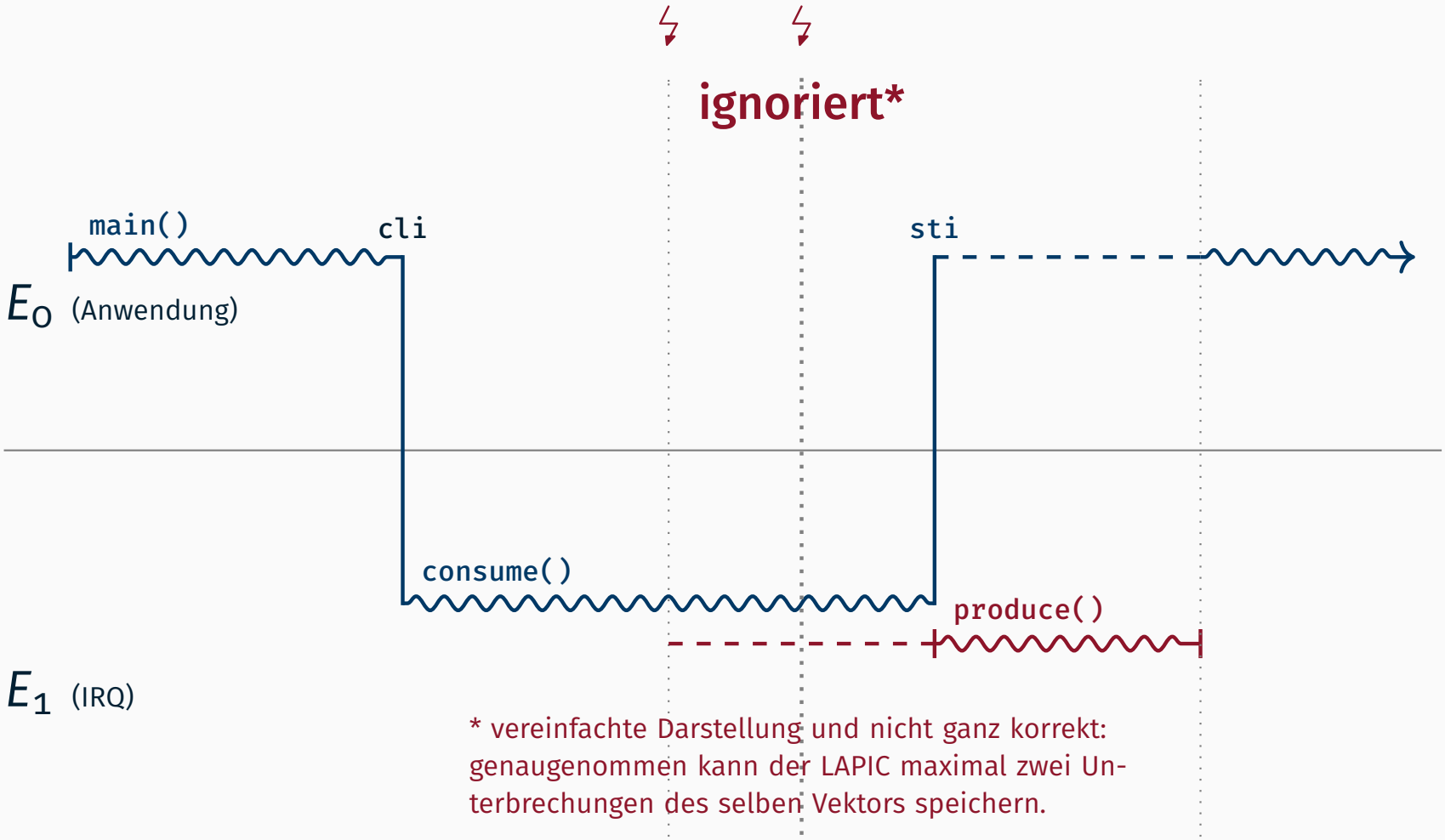
harte Synchronisation



harte Synchronisation



harte Synchronisation



* vereinfachte Darstellung und nicht ganz korrekt: genaugenommen kann der LAPIC maximal zwei Unterbrechungen des selben Vektors speichern.

harte Synchronisation

Pessimistischer Ansatz

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

Prolog/Epilog-Modell

Idee

Aufspalten der IRQ-Behandlung in

Prolog erledigt das Nötigste auf E_1

Epilog läuft auf neuer Ebene $\frac{1}{2}$ und übernimmt Synchronisation $\rightarrow E_1$ / IRQs wieder frei

Idee

Aufspalten der IRQ-Behandlung in

Prolog erledigt das Nötigste auf E_1

Epilog läuft auf neuer Ebene $\frac{1}{2}$ und übernimmt Synchronisation $\rightarrow E_1$ / IRQs wieder frei

Operationen

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

bei harter Synchronisation

Idee

Aufspalten der IRQ-Behandlung in

Prolog erledigt das Nötigste auf E_1

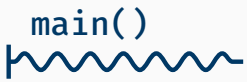
Epilog läuft auf neuer Ebene $\frac{1}{2}$ und übernimmt Synchronisation $\rightarrow E_1$ / IRQs wieder frei

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

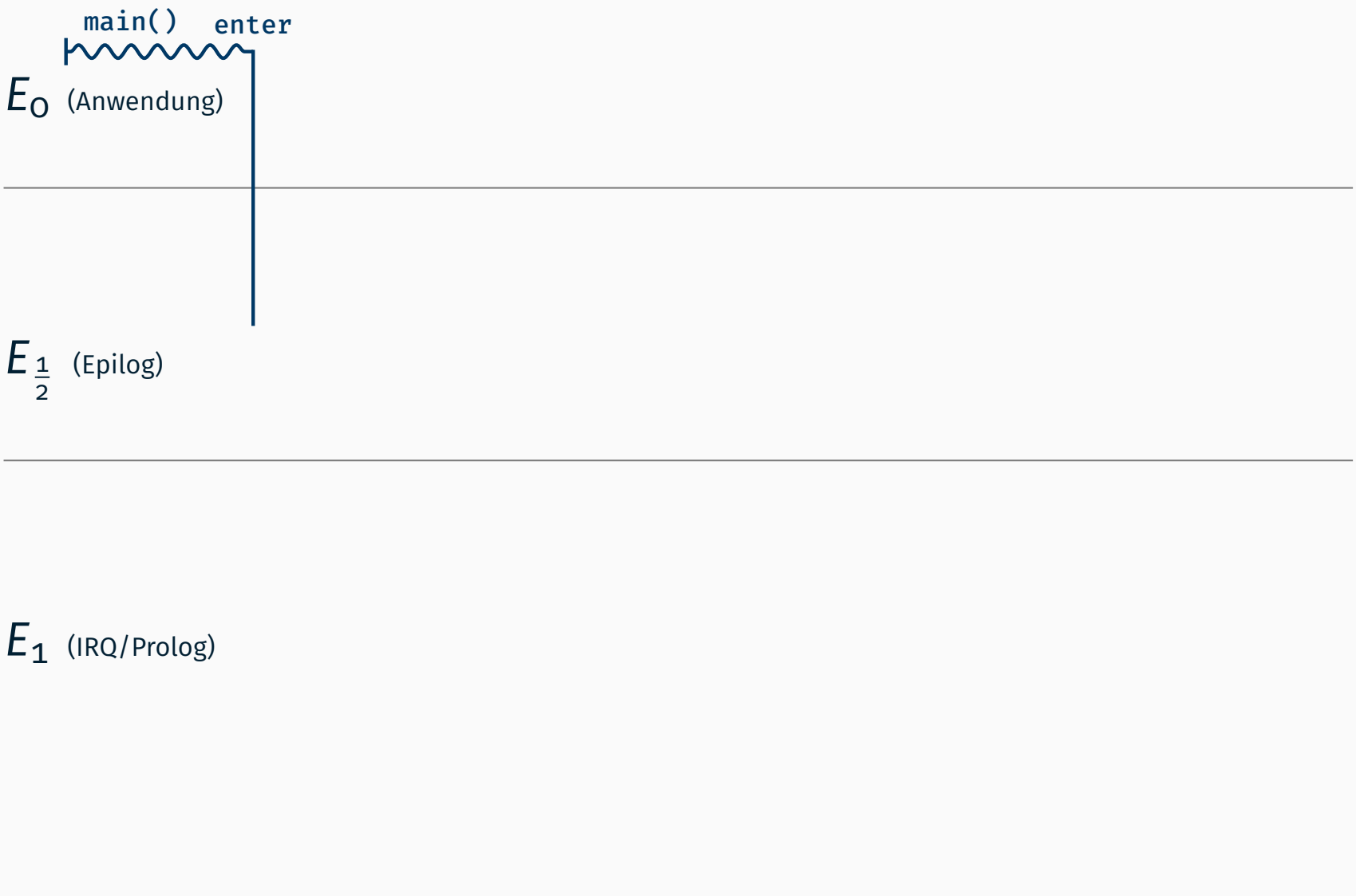


E_0 (Anwendung)

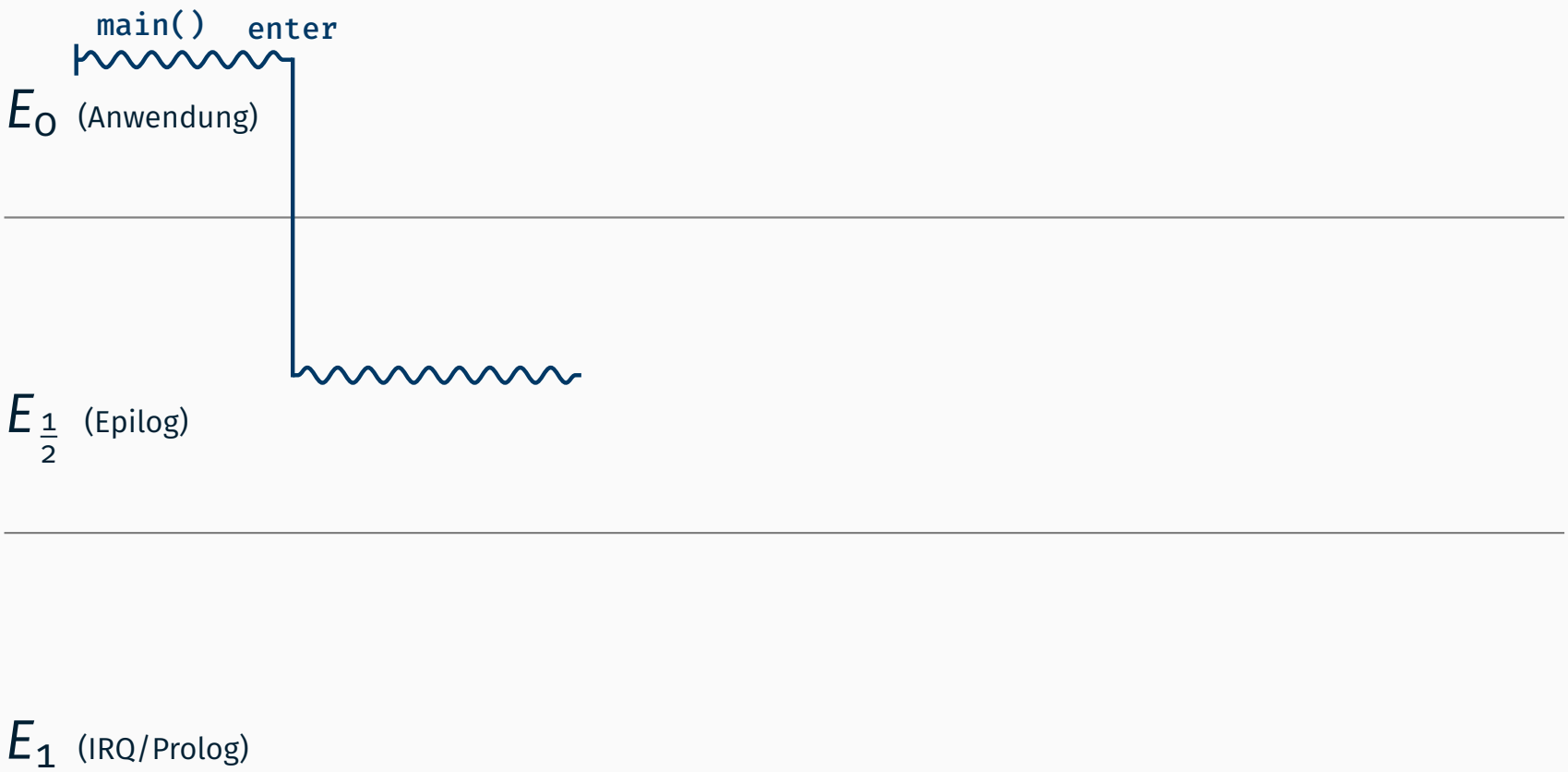
$E_{\frac{1}{2}}$ (Epilog)

E_1 (IRQ/Prolog)

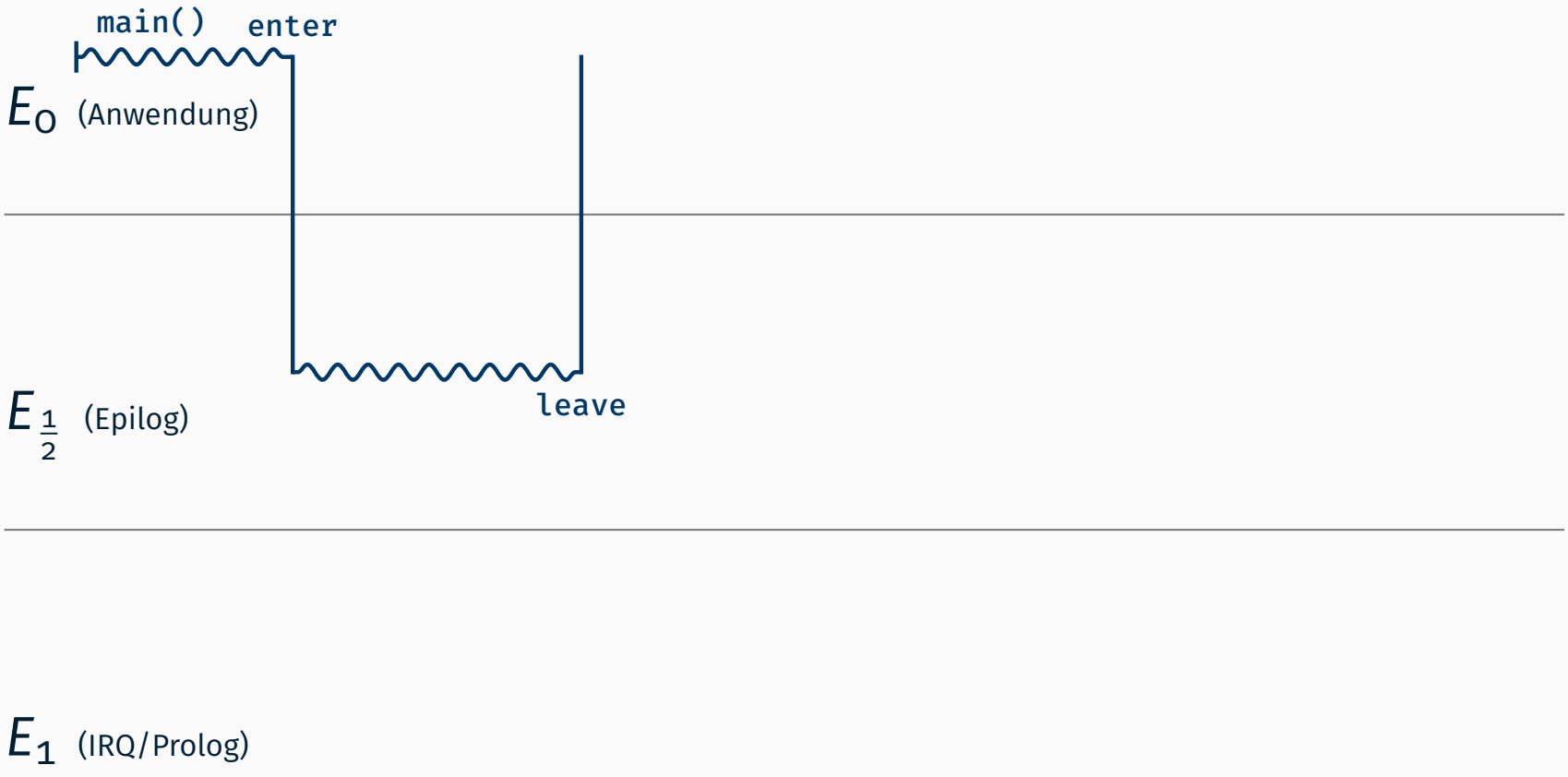
Prolog/Epilog-Modell



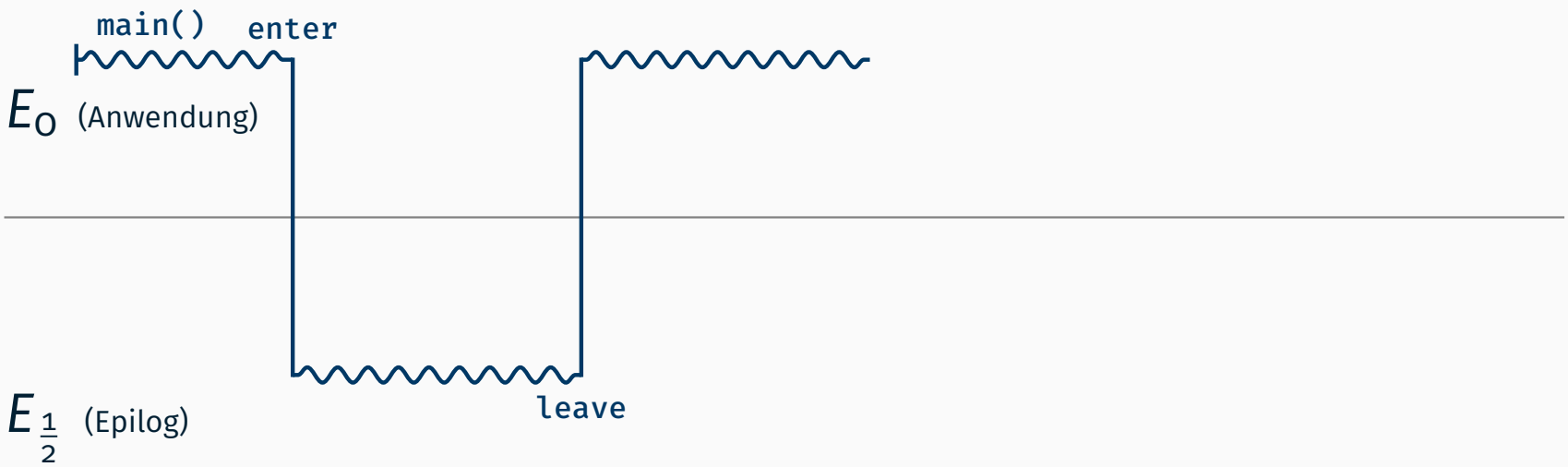
Prolog/Epilog-Modell



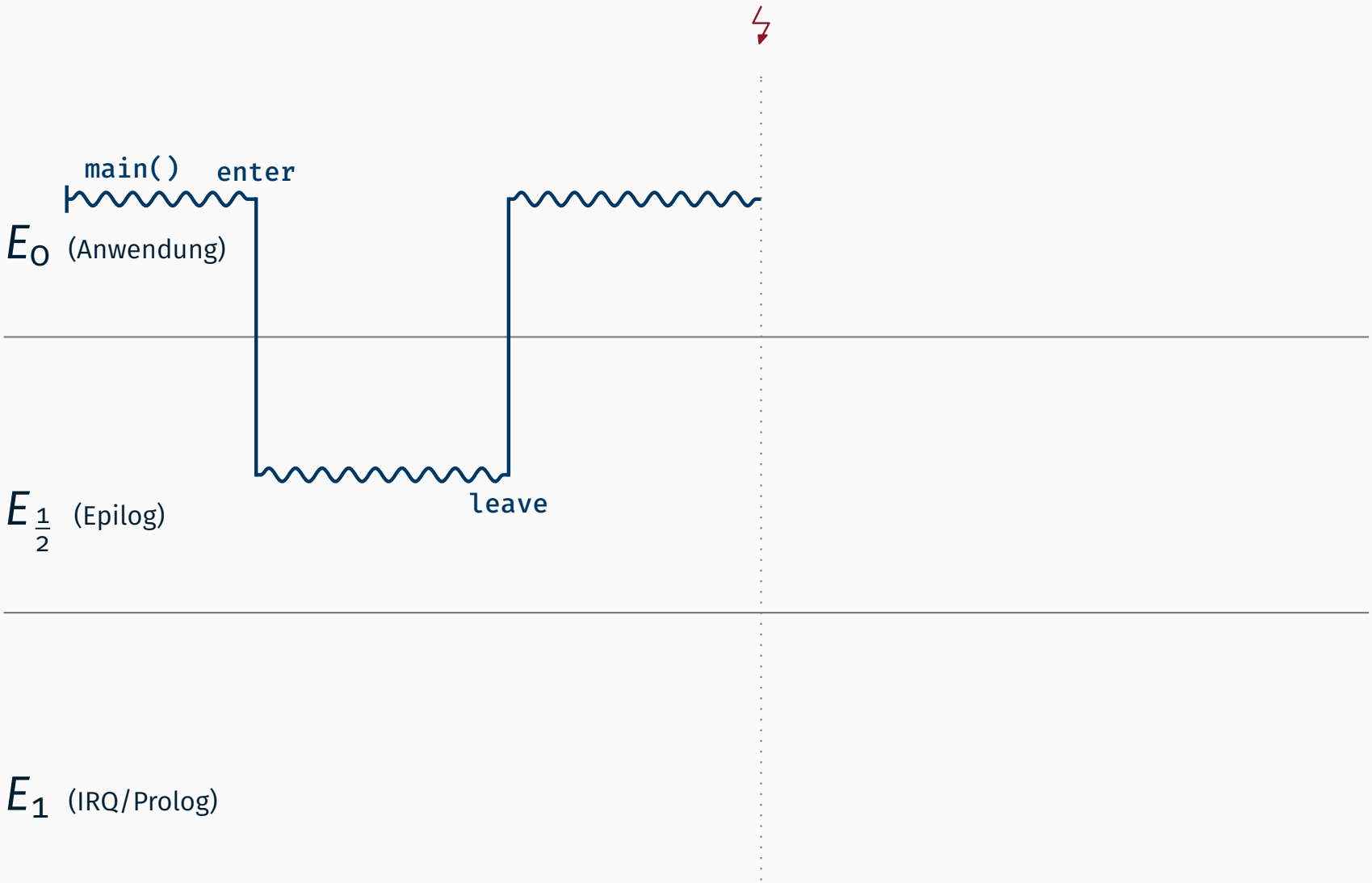
Prolog/Epilog-Modell



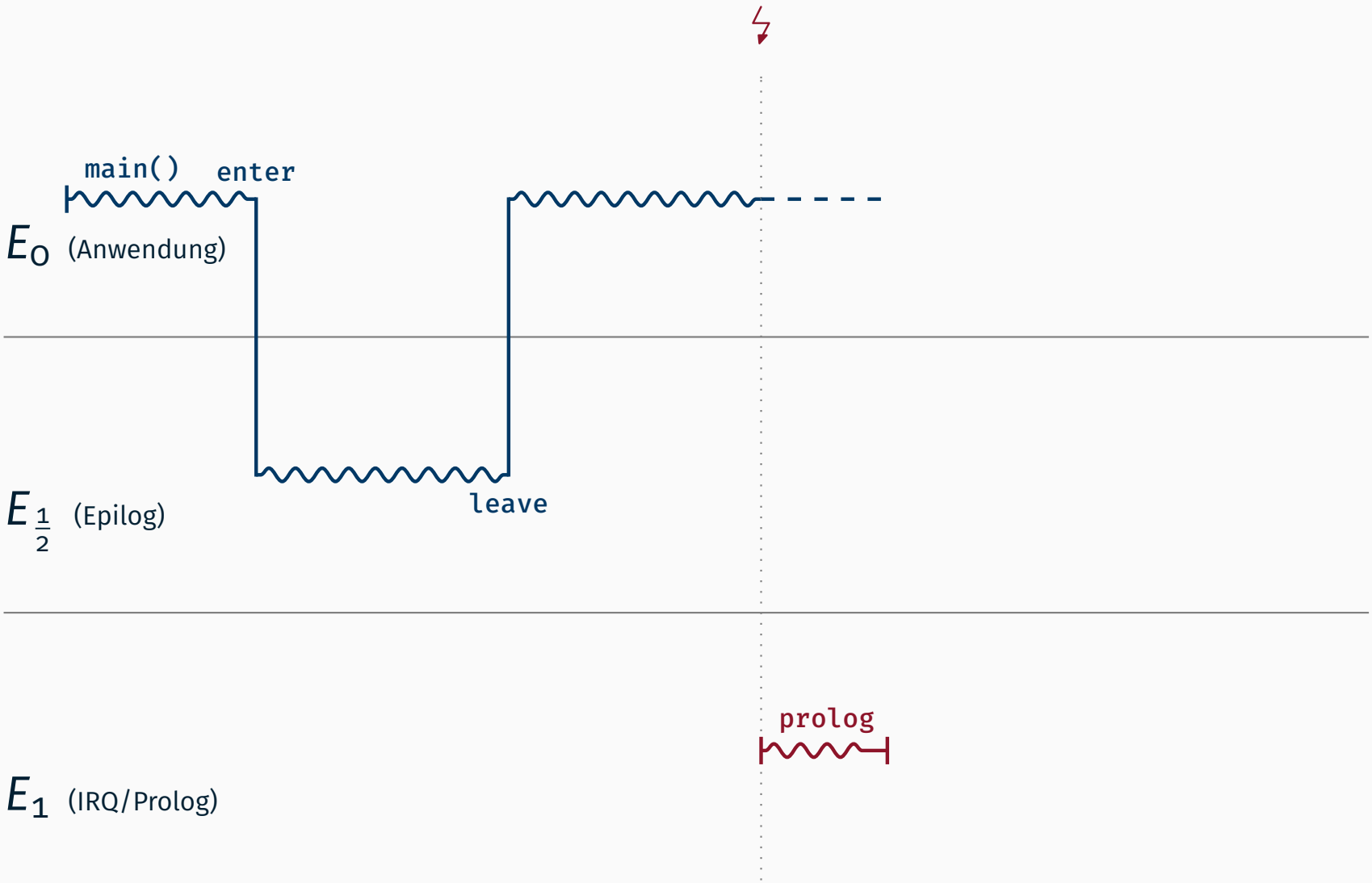
Prolog/Epilog-Modell



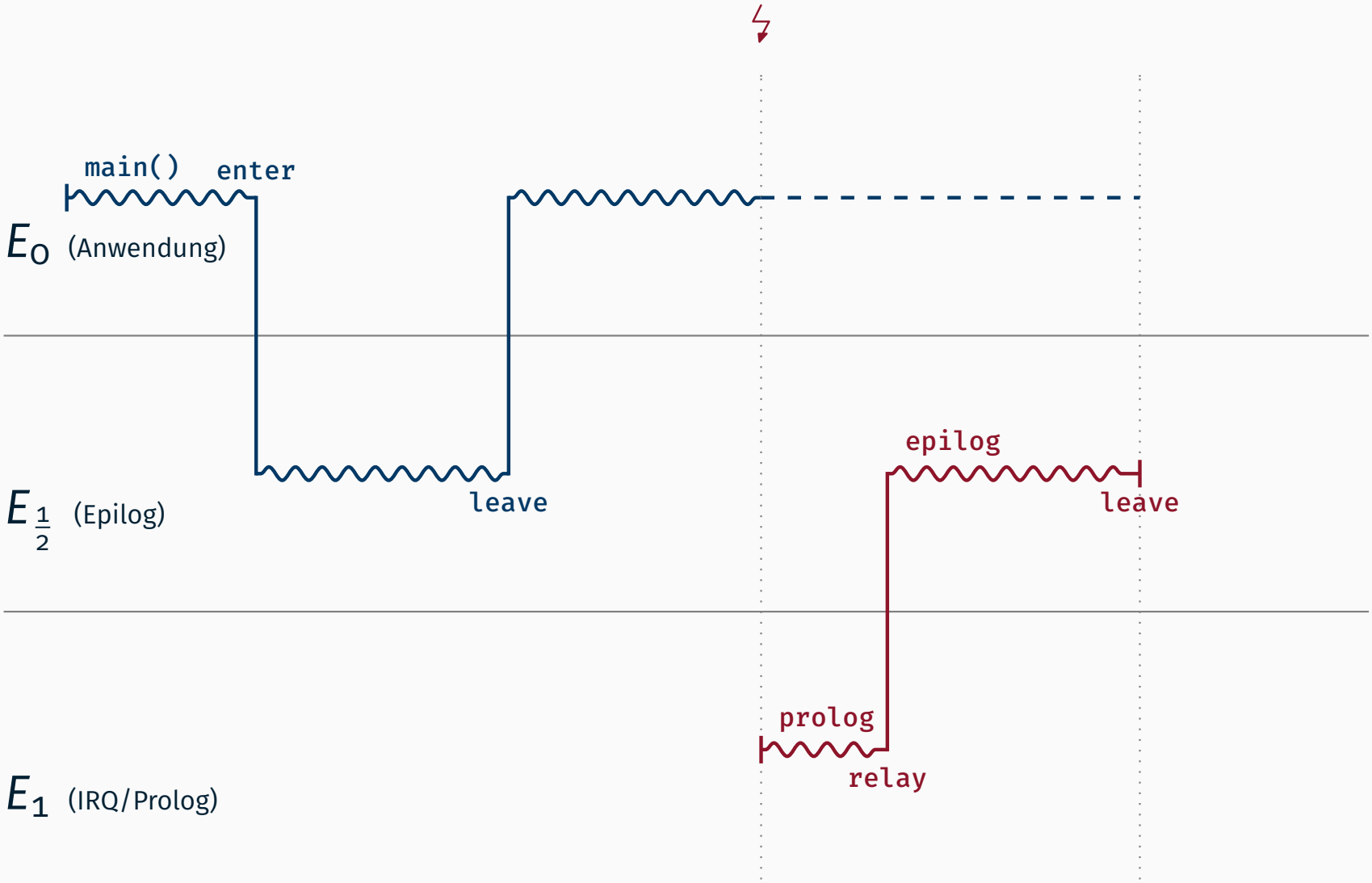
Prolog/Epilog-Modell



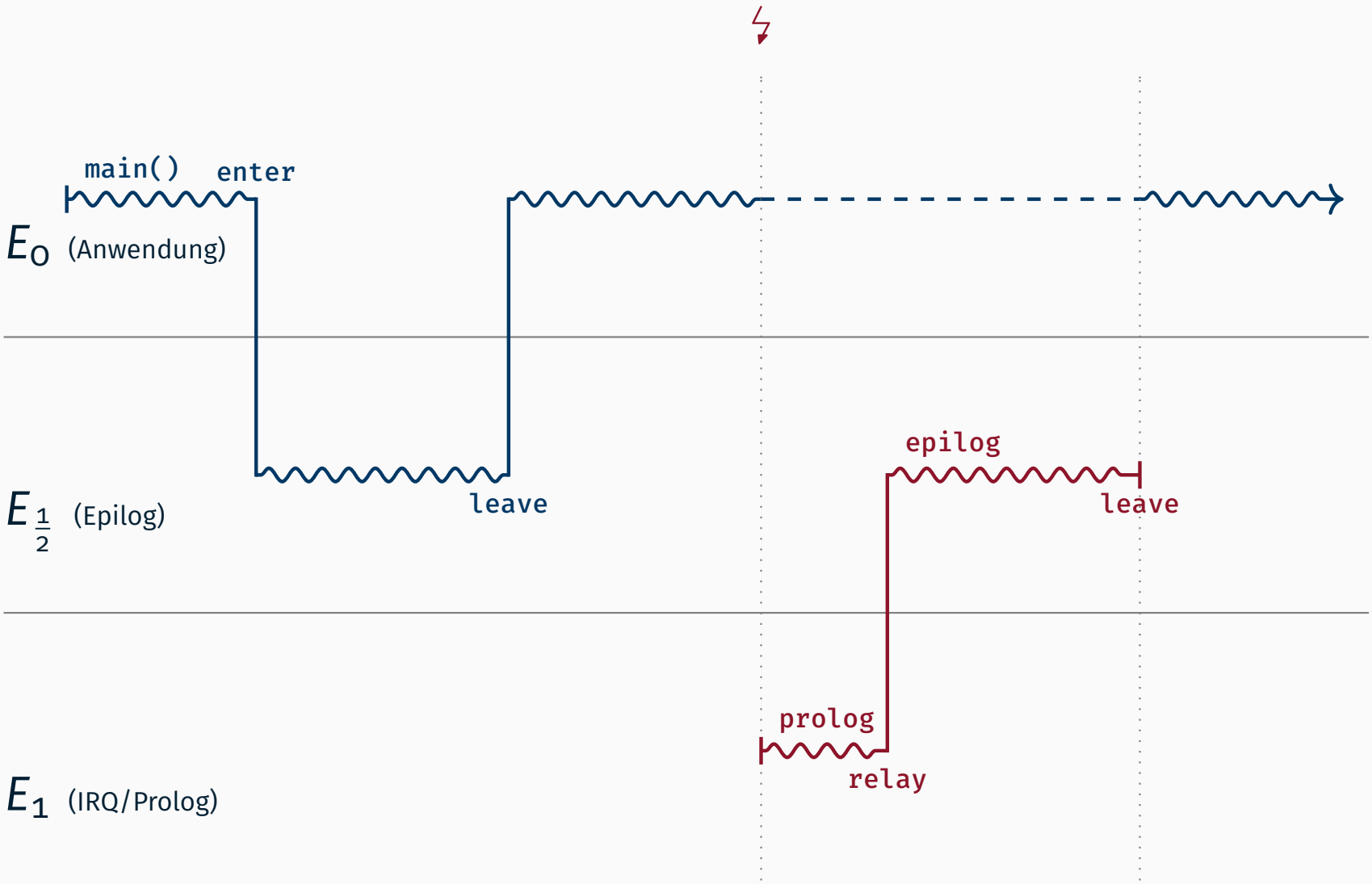
Prolog/Epilog-Modell



Prolog/Epilog-Modell



Prolog/Epilog-Modell



Prolog/Epilog-Modell

Kombinierter Ansatz

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

Prolog/Epilog-Modell

Kombinierter Ansatz

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

$\rightarrow$ **guter Kompromiss**

Umsetzung

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

```
epilog(){

    produce();

}
```

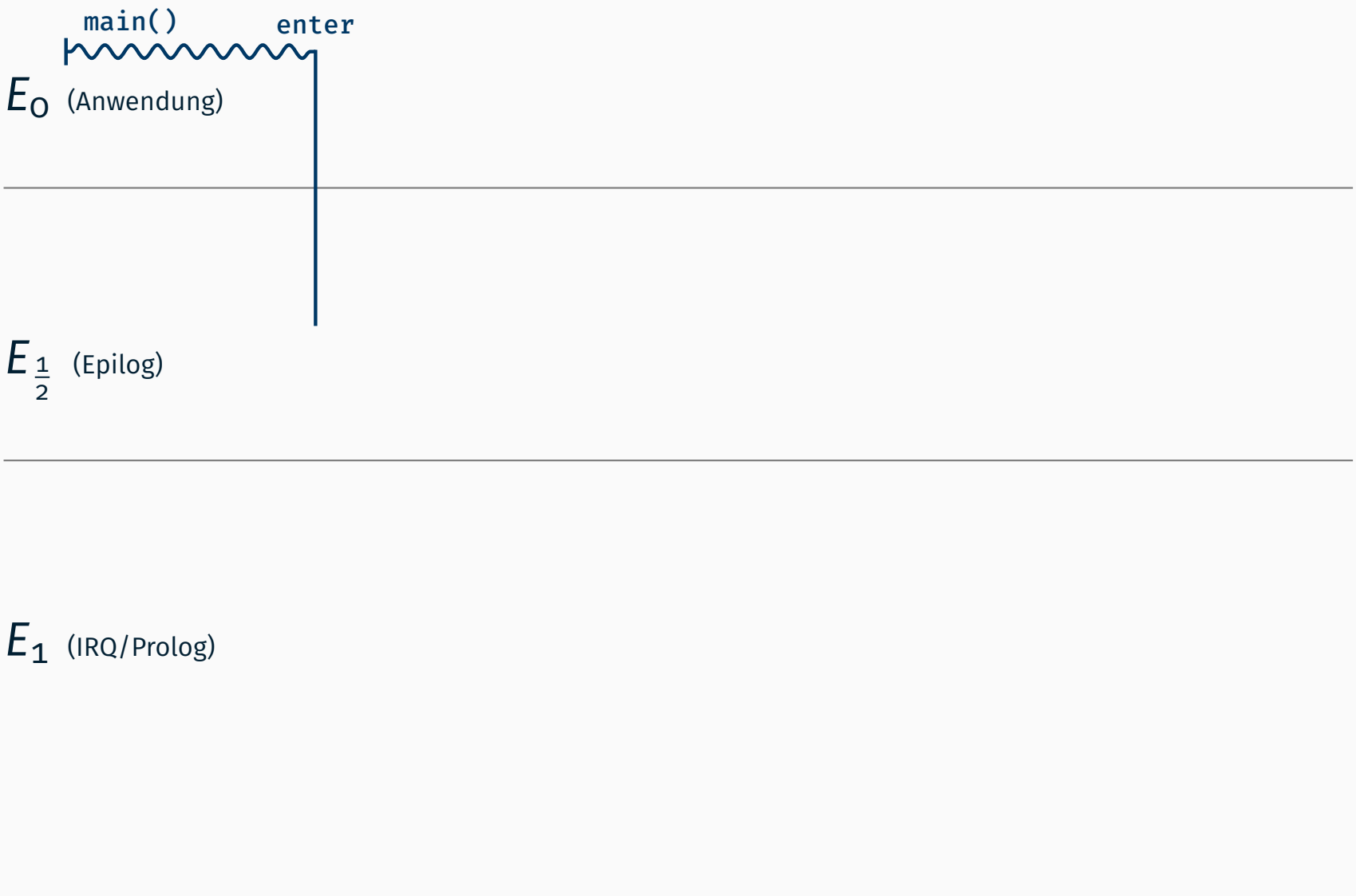
Umsetzung

E_0 (Anwendung)

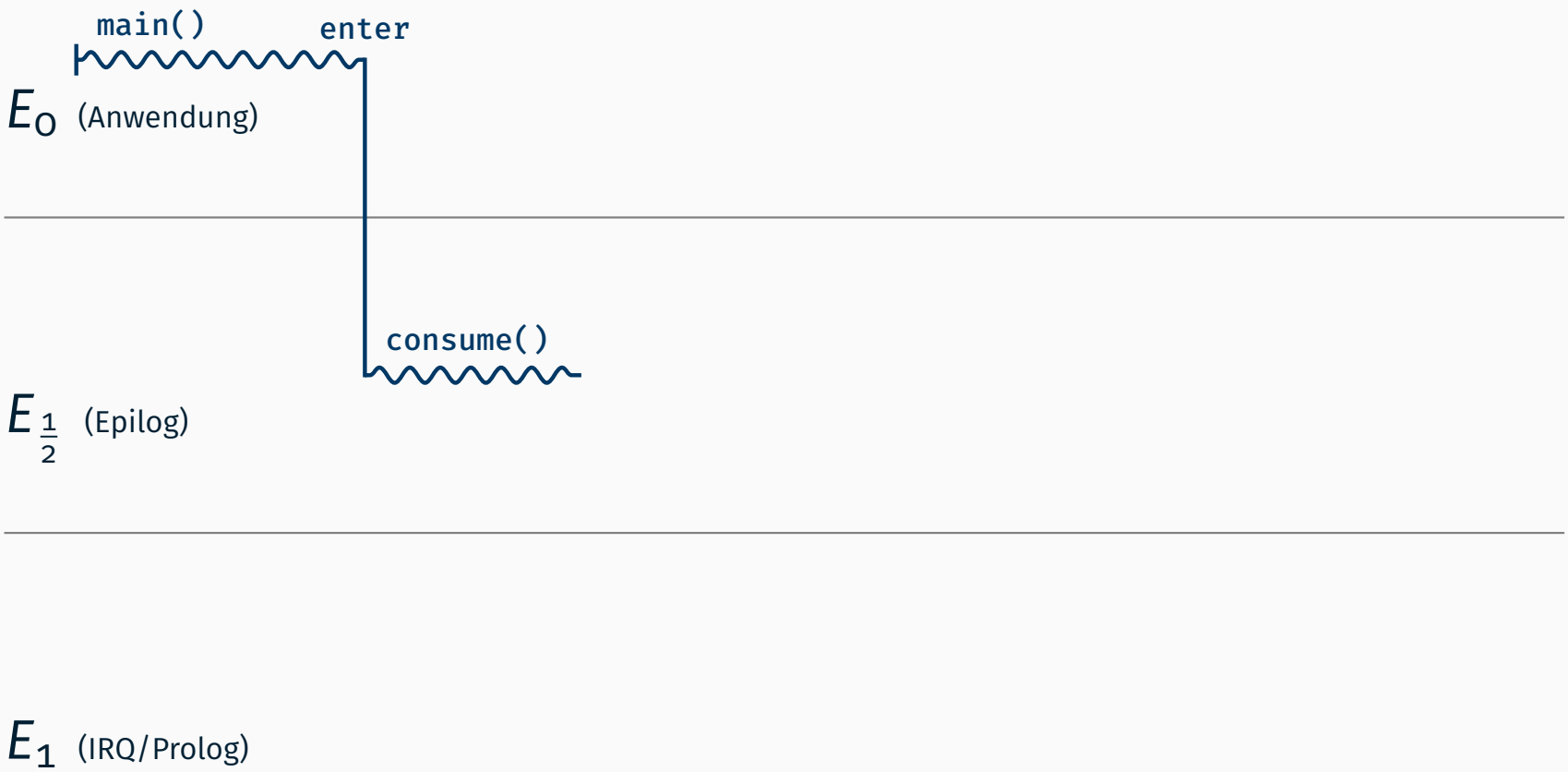
$E_{\frac{1}{2}}$ (Epilog)

E_1 (IRQ/Prolog)

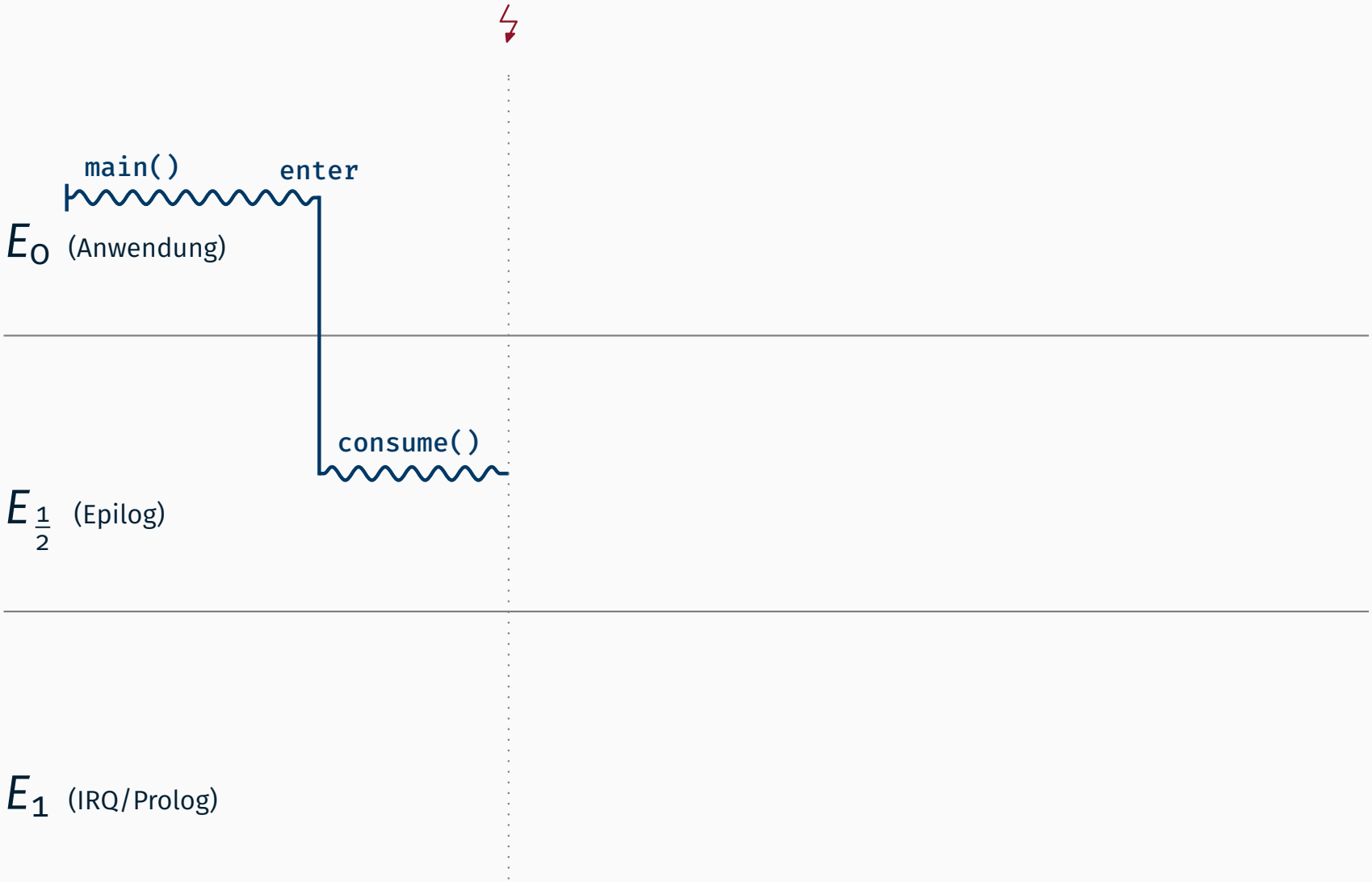
Umsetzung



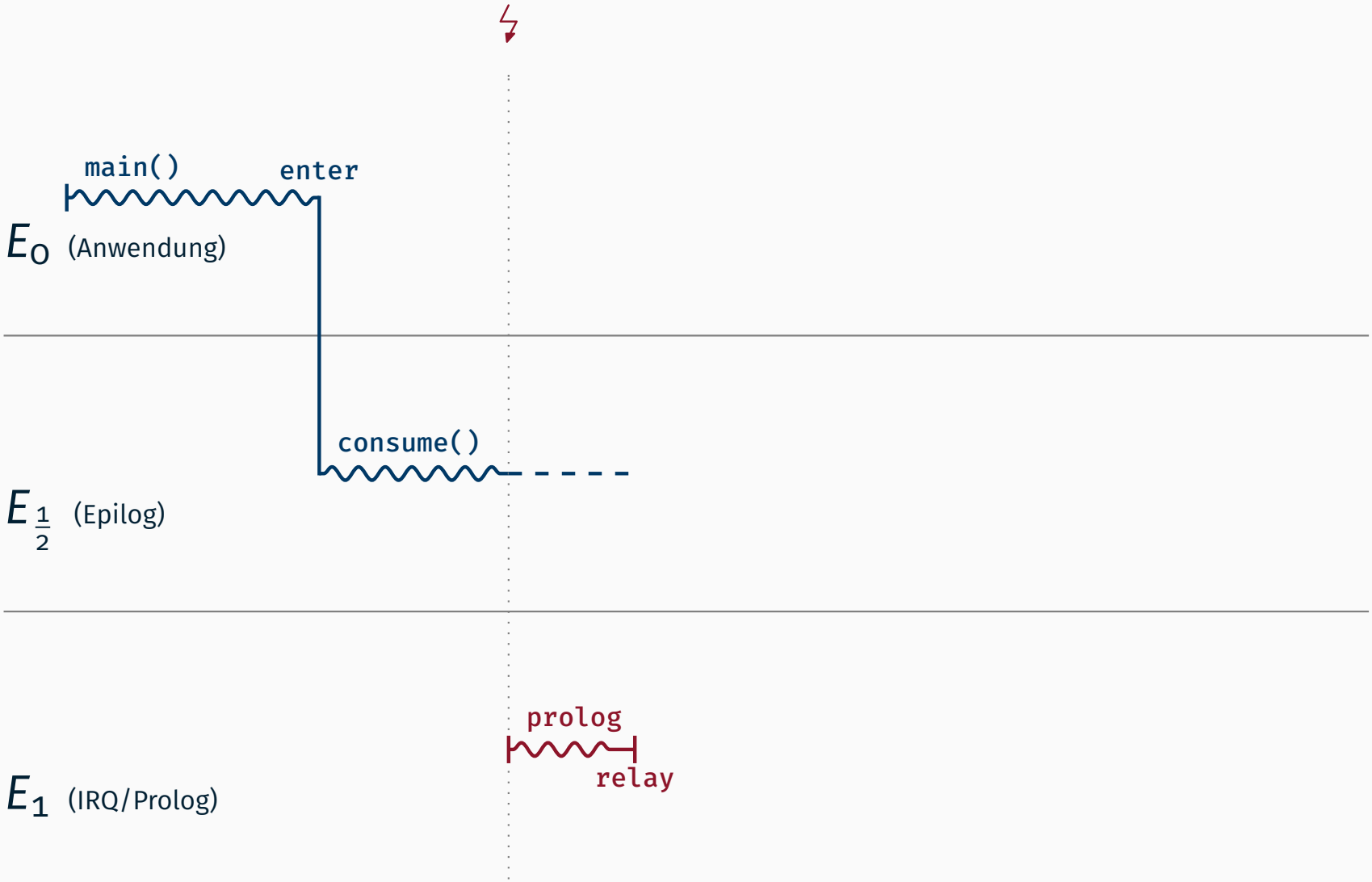
Umsetzung


main()
enter
E_0 (Anwendung)
consume()
$E_{\frac{1}{2}}$ (Epilog)
E_1 (IRQ/Prolog)

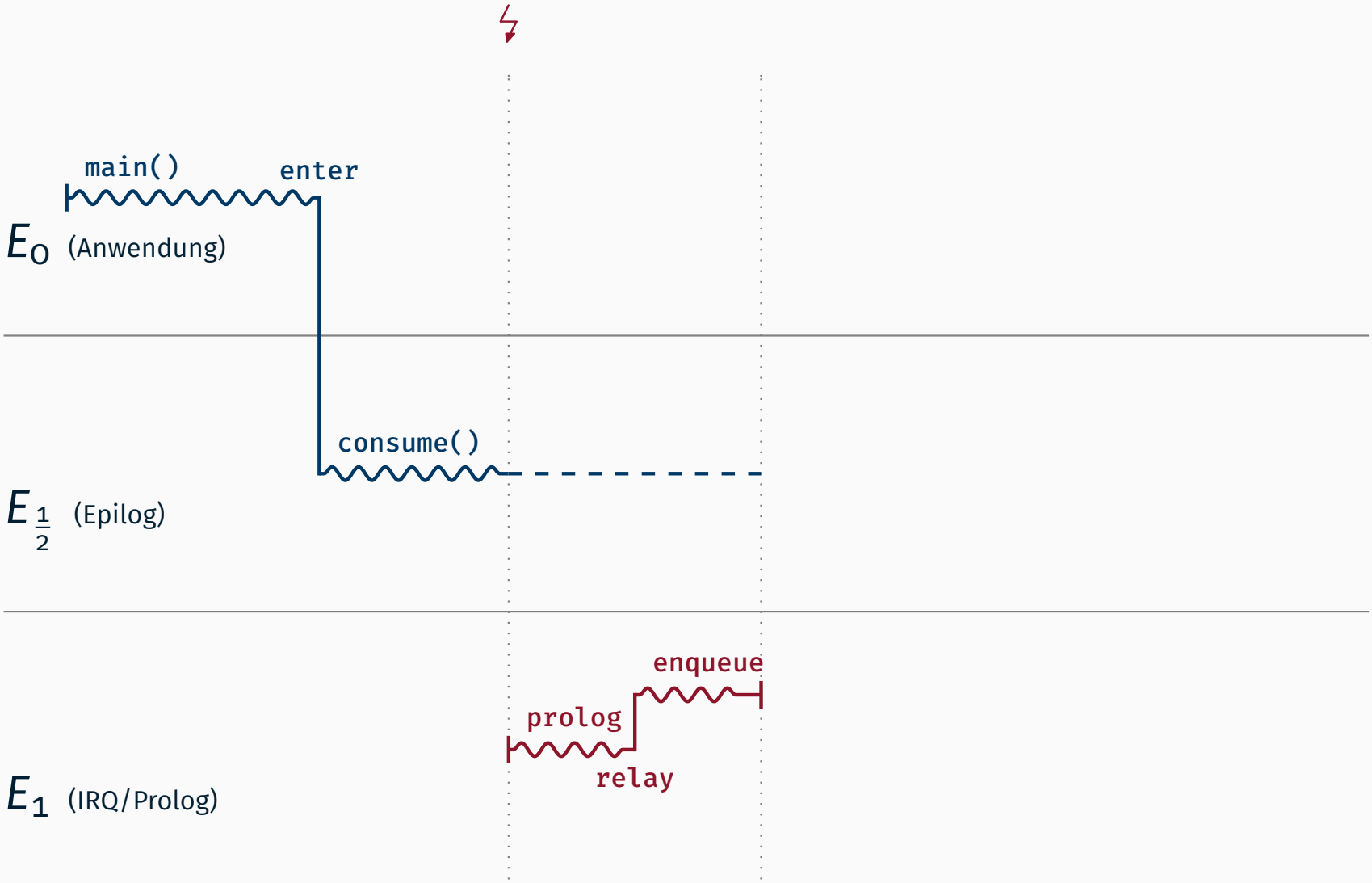
Umsetzung


main()
enter
E_0 (Anwendung)
consume()
$E_{\frac{1}{2}}$ (Epilog)
E_1 (IRQ/Prolog)

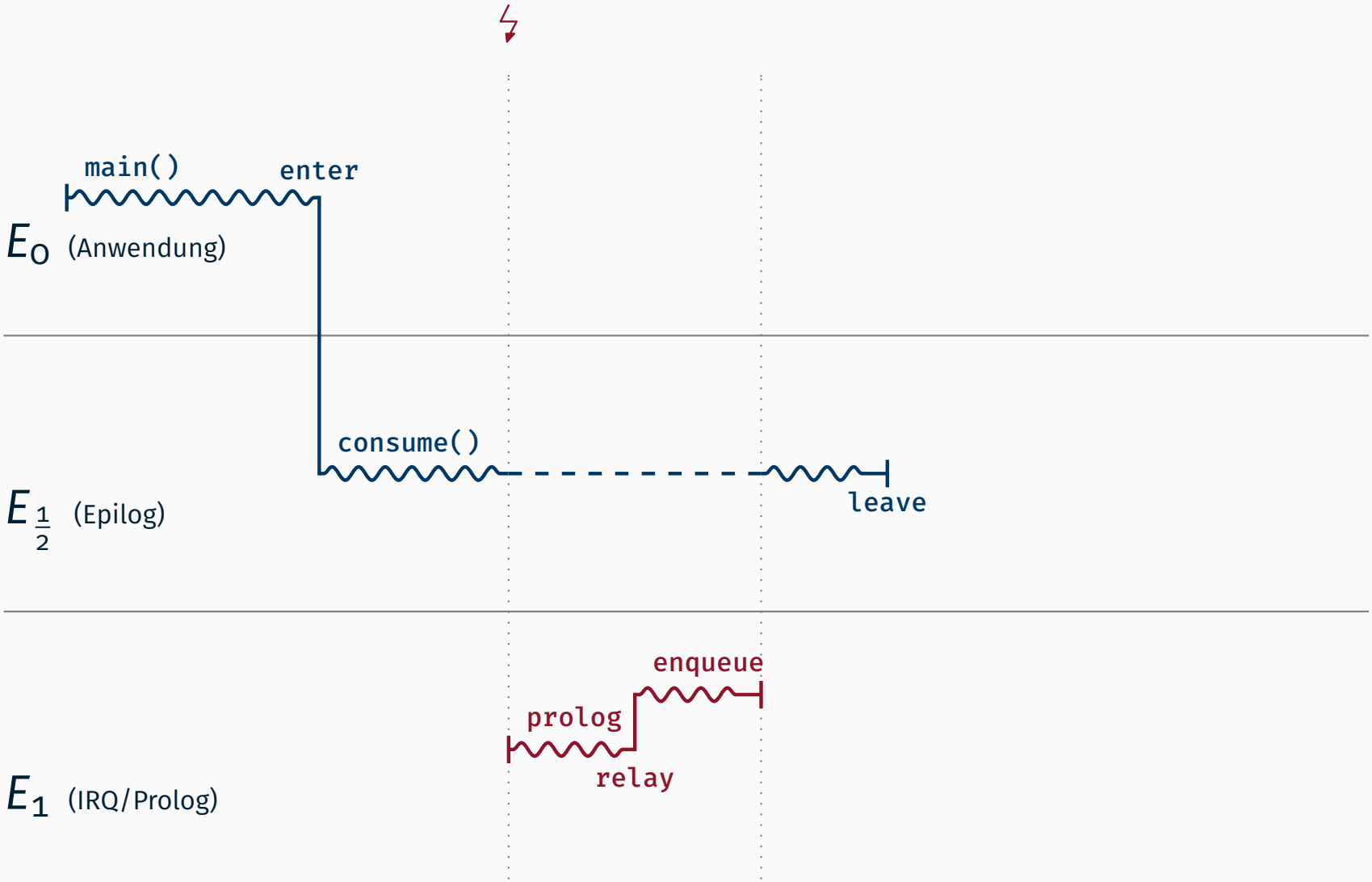
Umsetzung


main()
enter
E_0 (Anwendung)
consume()
$E_{\frac{1}{2}}$ (Epilog)
prolog
relay
E_1 (IRQ/Prolog)

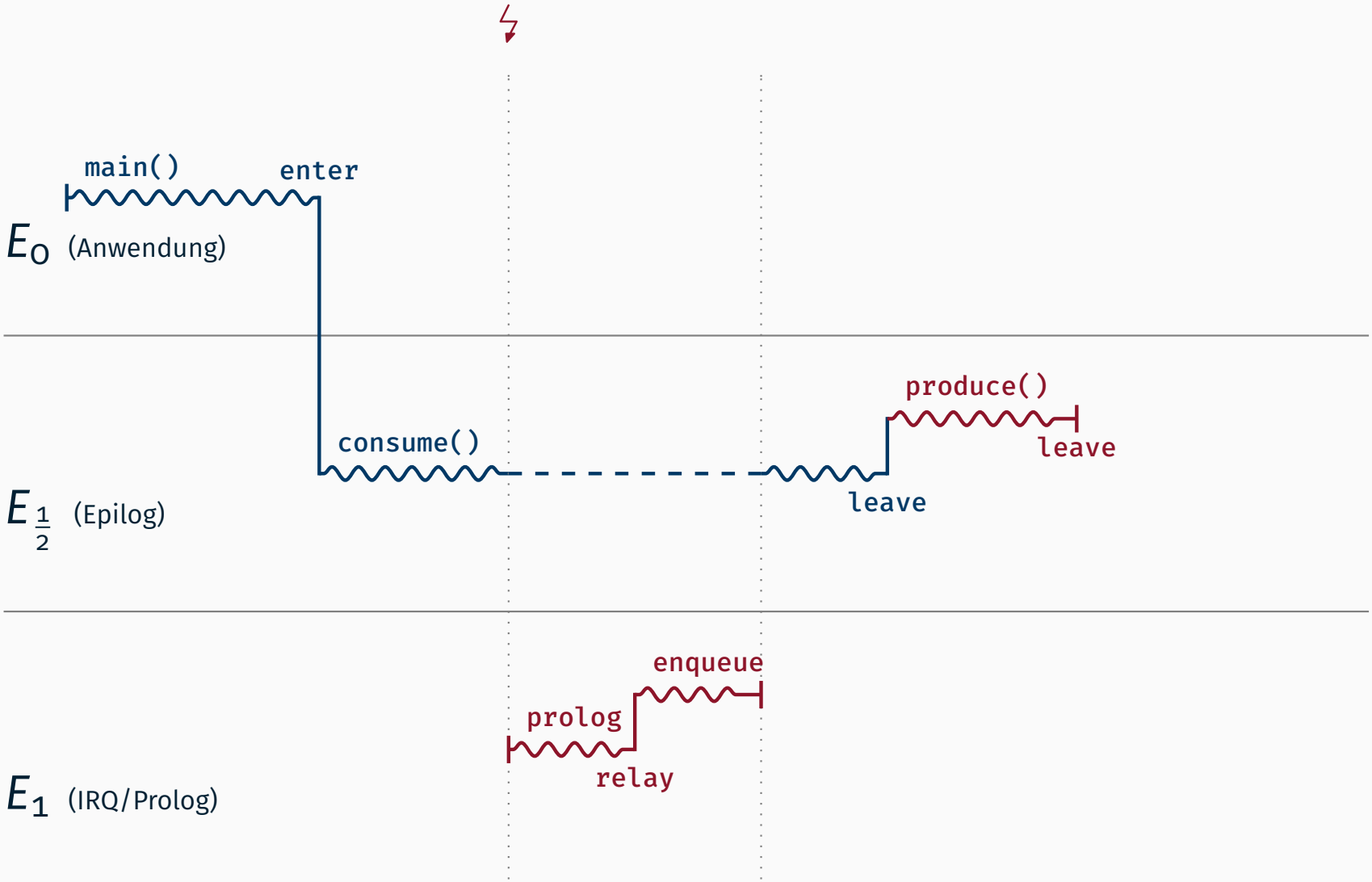
Umsetzung


main()
enter
E_0 (Anwendung)
consume()
$E_{\frac{1}{2}}$ (Epilog)
enqueue
prolog
relay
E_1 (IRQ/Prolog)

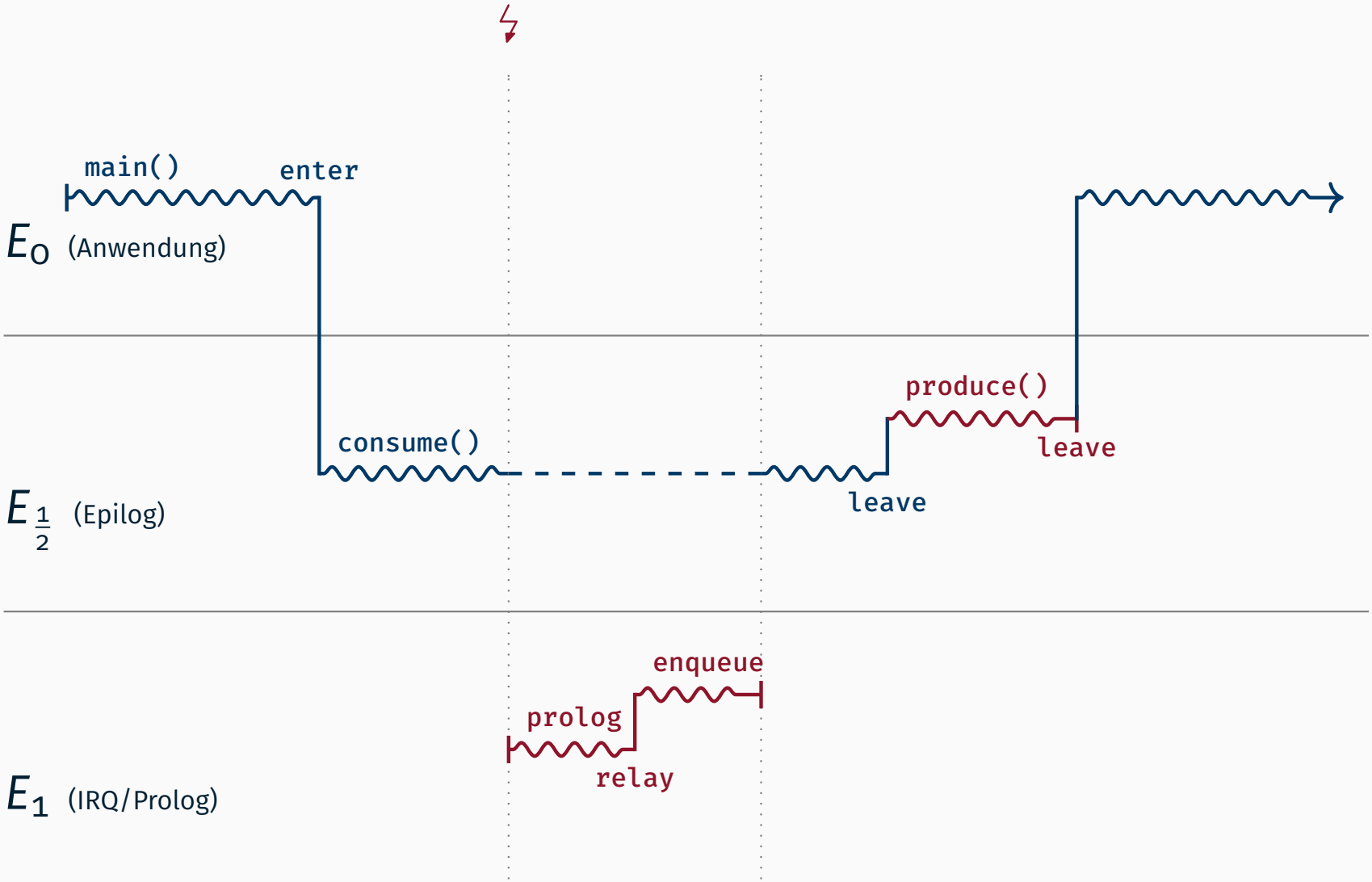
Umsetzung


main()
enter
E_0 (Anwendung)
consume()
leave
$E_{\frac{1}{2}}$ (Epilog)
enqueue
prolog
relay
E_1 (IRQ/Prolog)

Umsetzung


main()
enter
E_0 (Anwendung)
consume()
produce()
leave
leave
$E_{\frac{1}{2}}$ (Epilog)
enqueue
prolog
relay
E_1 (IRQ/Prolog)

Umsetzung


main()
enter
E_0 (Anwendung)
consume()
produce()
leave
leave
$E_{\frac{1}{2}}$ (Epilog)
enqueue
prolog
relay
E_1 (IRQ/Prolog)

Aufgabe

Wechsel von harter Synchronisation zu Prolog/Epilog-Modell

Wechsel von harter Synchronisation zu Prolog/Epilog-Modell

Was wird gebraucht?

Wechsel von harter Synchronisation zu Prolog/Epilog-Modell

Was wird gebraucht?

`List`/`Queue` zum Einreihen der Epiloge

`Guard` mit `enter()`, `leave()` und `relay()` für Prioritätsebenen

Wechsel von harter Synchronisation zu Prolog/Epilog-Modell

Was wird gebraucht?

`List`/`Queue` zum Einreihen der Epiloge

`Guard` mit `enter()`, `leave()` und `relay()` für Prioritätsebenen

Was muss angepasst werden?

Wechsel von harter Synchronisation zu Prolog/Epilog-Modell

Was wird gebraucht?

`List`/`Queue` zum Einreihen der Epiloge

`Guard` mit `enter()`, `leave()` und `relay()` für Prioritätsebenen

Was muss angepasst werden?

- Keyboard
- Application
- Guardian

Wechsel von harter Synchronisation zu Prolog/Epilog-Modell

Was wird gebraucht?

`List`/`Queue` zum Einreihen der Epiloge

`Guard` mit `enter()`, `leave()` und `relay()` für Prioritätsebenen

Was muss angepasst werden?

- Keyboard
- Application
- Guardian
- *alles was hart synchronisiert*

Aufgabenstellung

- Warteschlange gegeben

Aufgabenstellung

- Warteschlange gegeben, Implementierung ist allerdings nicht unterbrechungstransparent
  - benötigt Schutz vor Unterbrechungen im `Guard` oder
  - durch Modifikation der `Queue` selbst

Aufgabenstellung

- Warteschlange gegeben, Implementierung ist allerdings nicht unterbrechungstransparent
  - benötigt Schutz vor Unterbrechungen im `Guard` oder
  - durch Modifikation der `Queue` selbst
- `Gate` muss geändert werden (`trigger()` zu `prolog()` und `epilog()`)

Aufgabenstellung

- Warteschlange gegeben, Implementierung ist allerdings nicht unterbrechungstransparent
  - benötigt Schutz vor Unterbrechungen im `Guard` oder
  - durch Modifikation der `Queue` selbst
- `Gate` muss geändert werden (`trigger()` zu `prolog()` und `epilog()`)
- jede `Gate`-Instanz darf nur einmal in der Epilogwarteschlange vorkommen

MPStuBS

Prolog/Epilog-Modell auf Mehrkernprozessoren

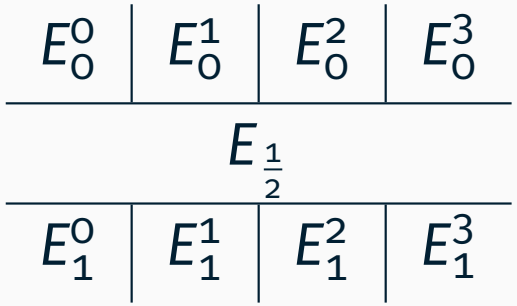
- jeder Kern hat eine eigene Epilogwarteschlange
  (damit die Epiloge auf dem selben Kern wie deren zugehörige Prologe ausgeführt werden)

Prolog/Epilog-Modell auf Mehrkernprozessoren

- jeder Kern hat eine eigene Epilogwarteschlange
  (damit die Epiloge auf dem selben Kern wie deren zugehörige Prologe ausgeführt werden)
- Zu jedem Zeitpunkt darf maximal ein Kern Epiloge ausführen

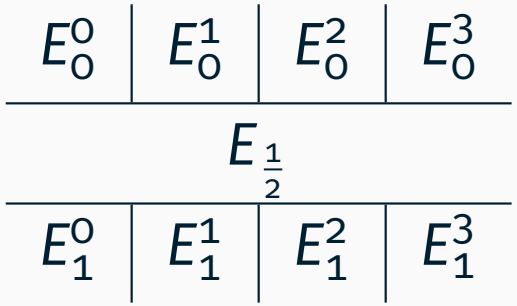
Prolog/Epilog-Modell auf Mehrkernprozessoren

- jeder Kern hat eine eigene Epilogwarteschlange
  (damit die Epiloge auf dem selben Kern wie deren zugehörige Prologe ausgeführt werden)
- Zu jedem Zeitpunkt darf maximal ein Kern Epiloge ausführen



Prolog/Epilog-Modell auf Mehrkernprozessoren

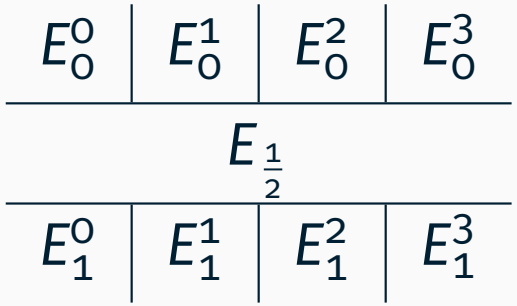
- jeder Kern hat eine eigene Epilogwarteschlange
  (damit die Epiloge auf dem selben Kern wie deren zugehörige Prologe ausgeführt werden)
- Zu jedem Zeitpunkt darf maximal ein Kern Epiloge ausführen



  $\Rightarrow$ 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)
- Zu jedem Zeitpunkt darf maximal ein Kern Epiloge ausführen



  $\Rightarrow$ Verwendung eines **big kernel lock** (BKL)
- korrekte Sperrreihenfolge ist extrem wichtig!

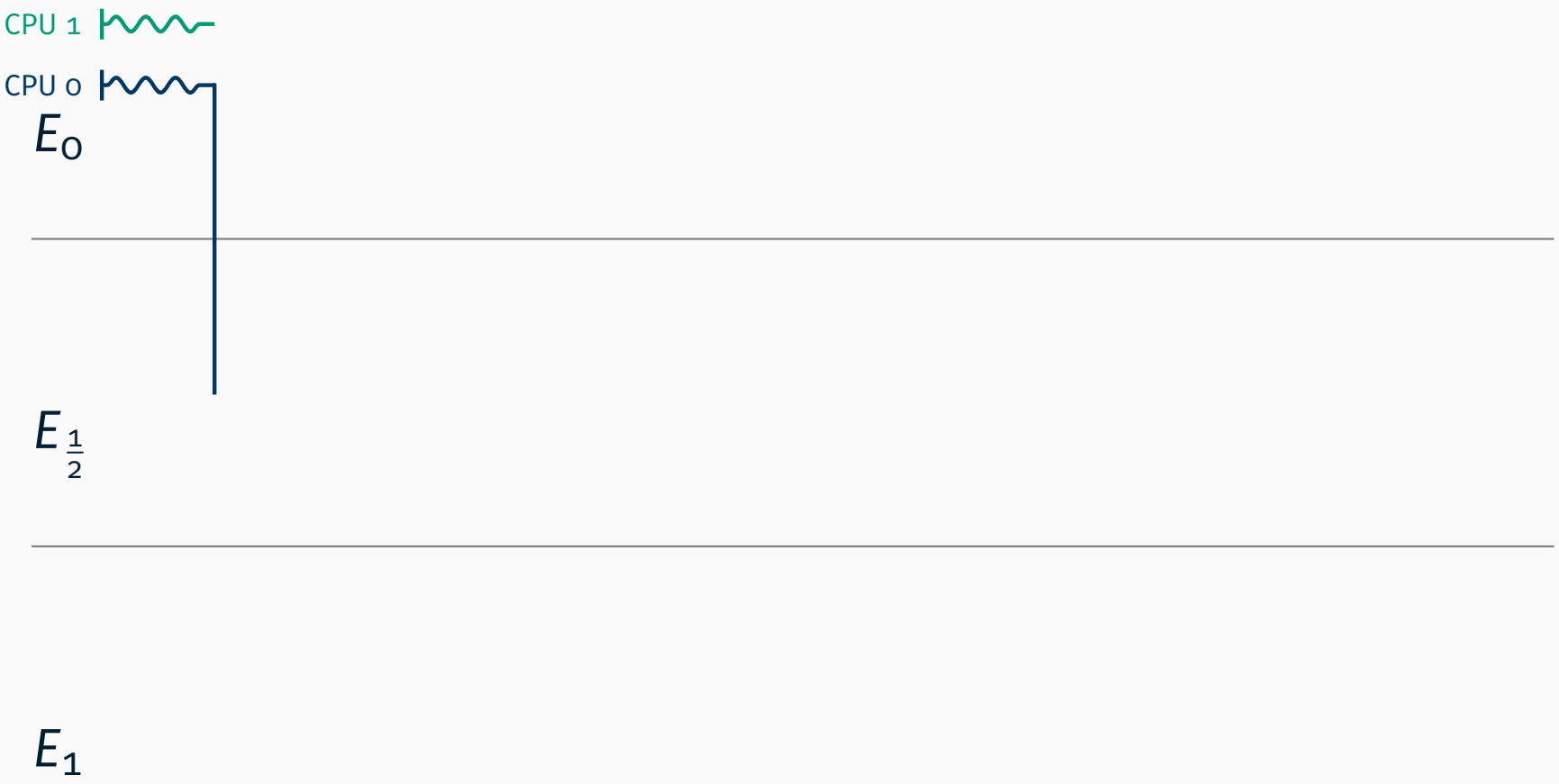
Beispiel für Mehrkernprozessoren

E_0

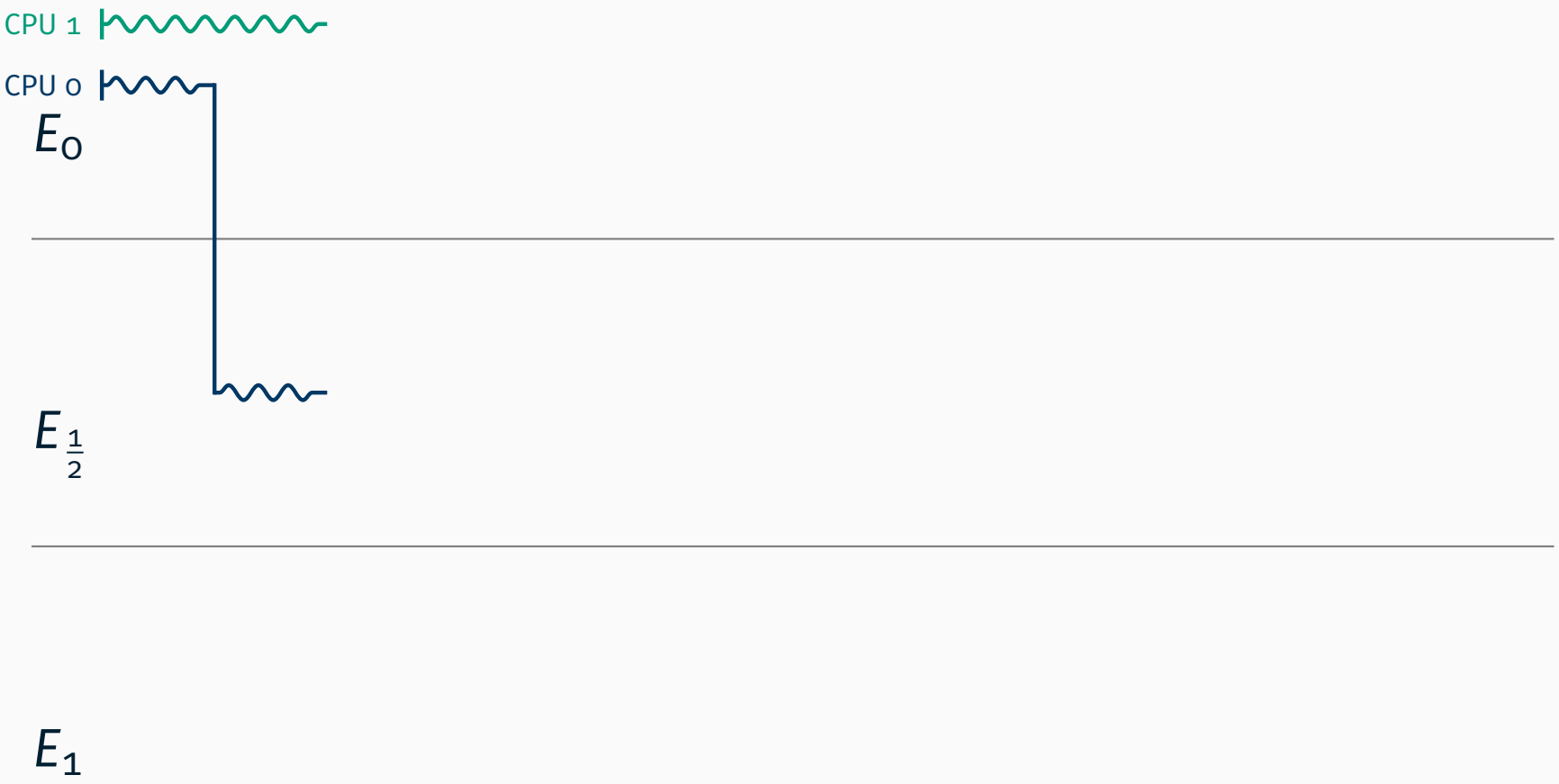
$E_{\frac{1}{2}}$

E_1

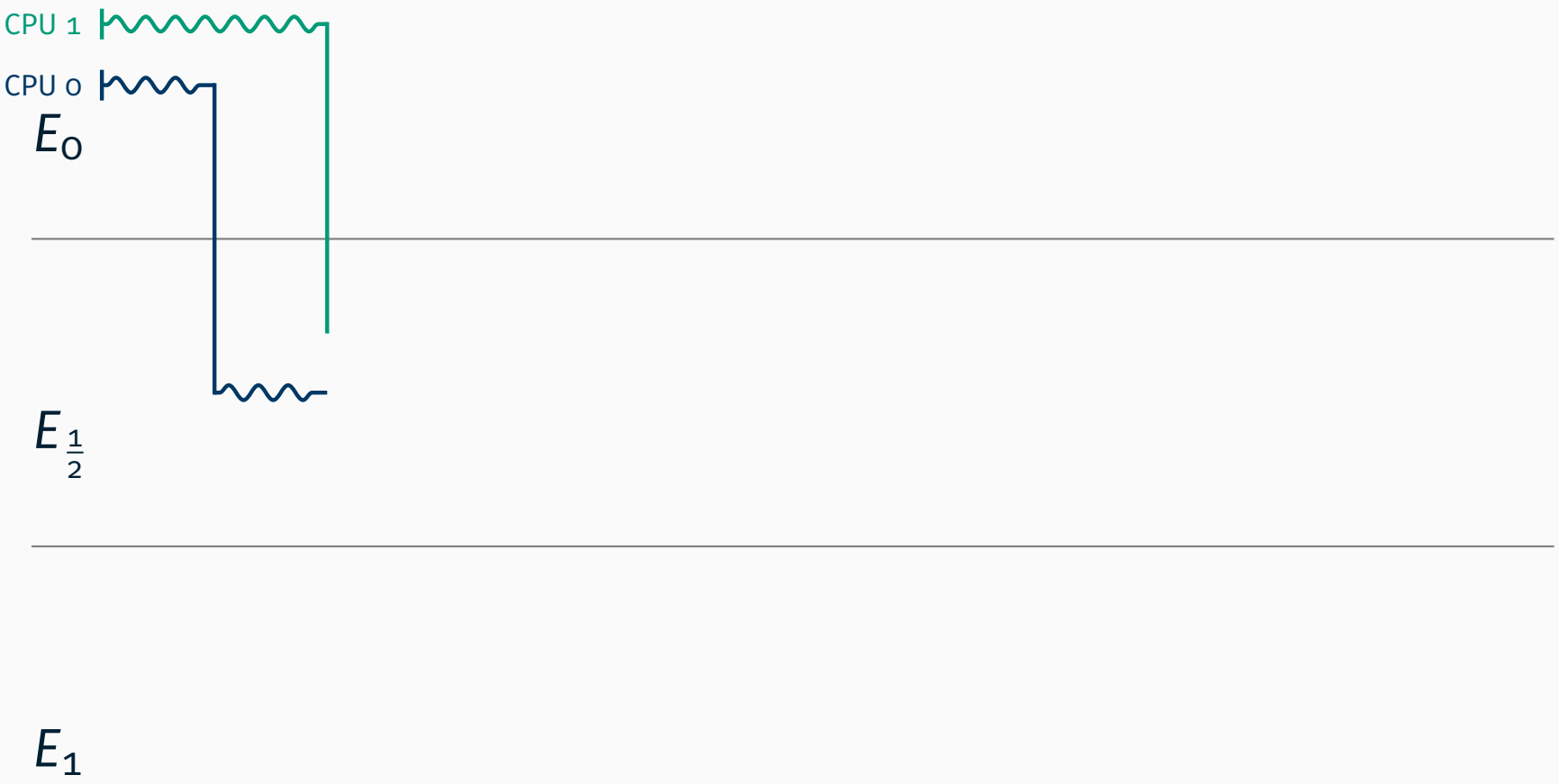
Beispiel für Mehrkernprozessoren



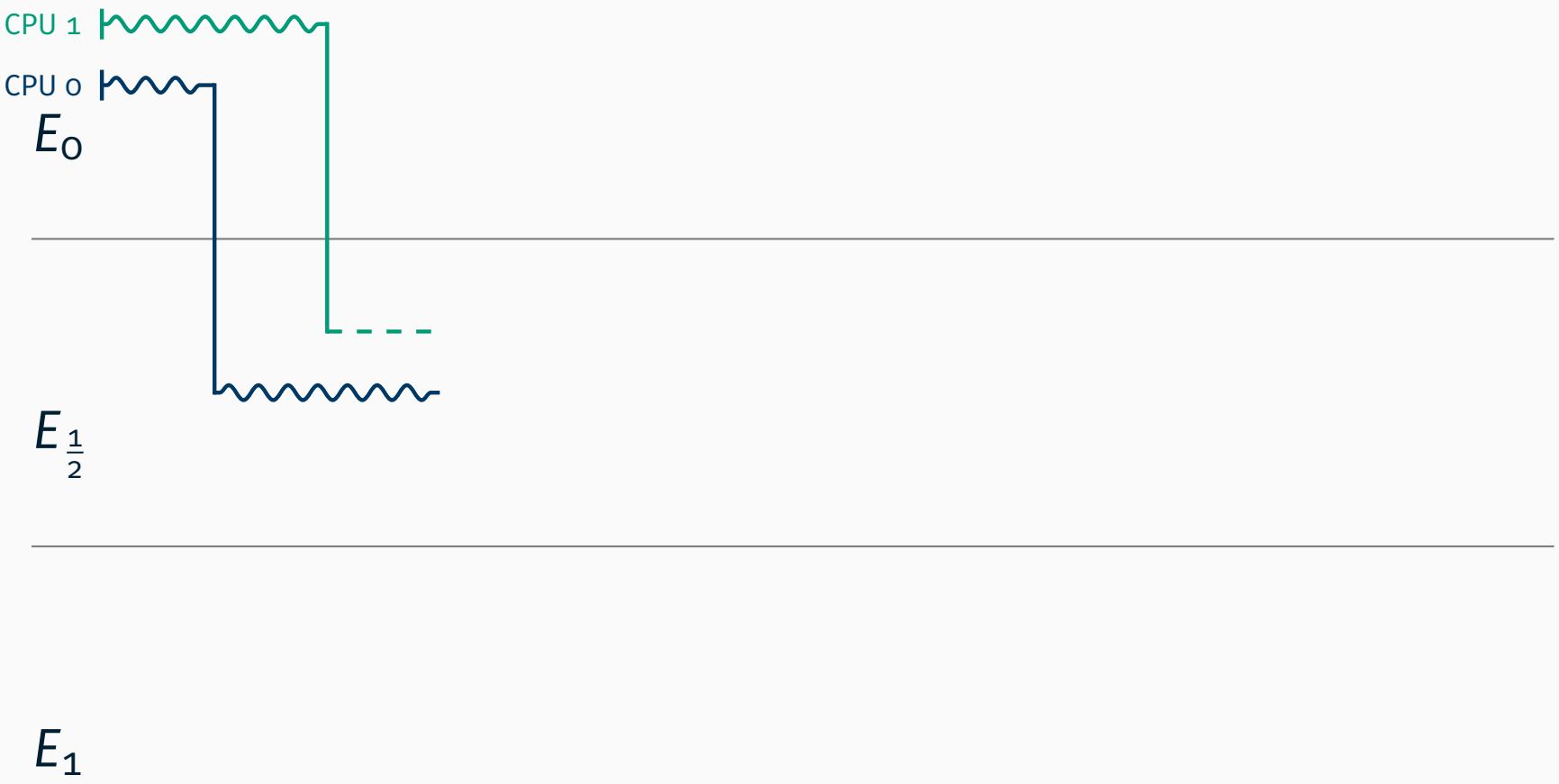
Beispiel für Mehrkernprozessoren



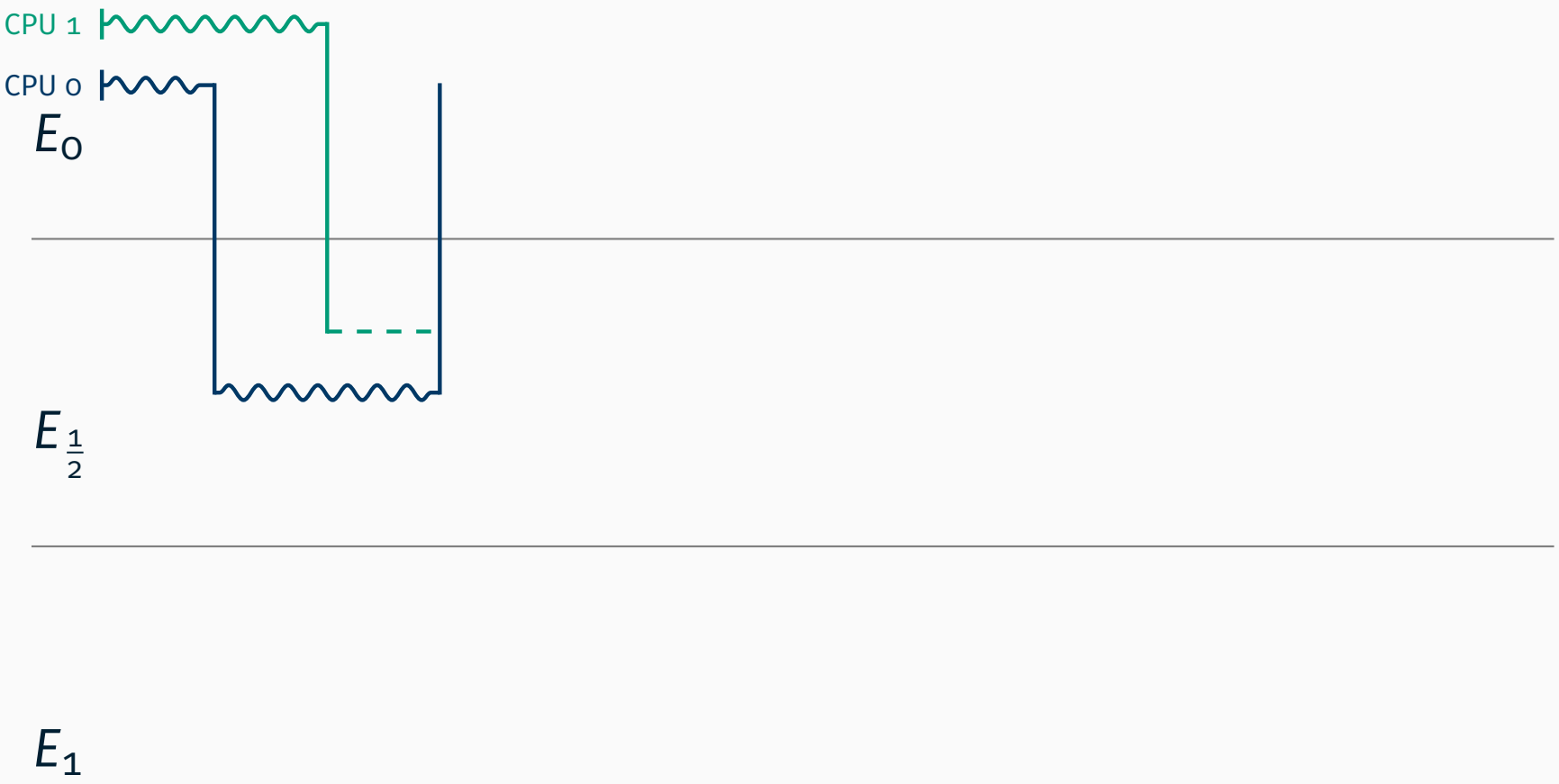
Beispiel für Mehrkernprozessoren



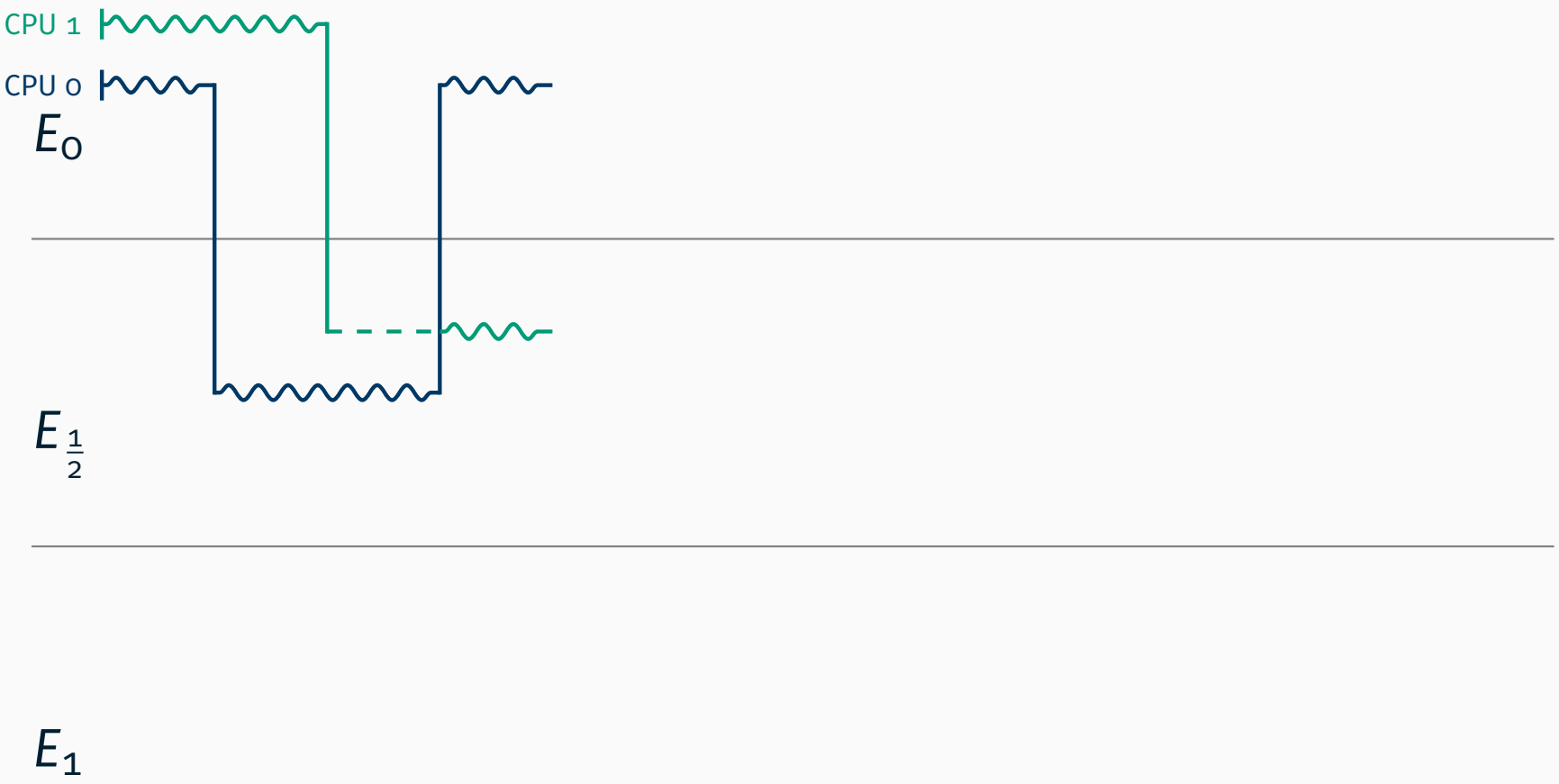
Beispiel für Mehrkernprozessoren



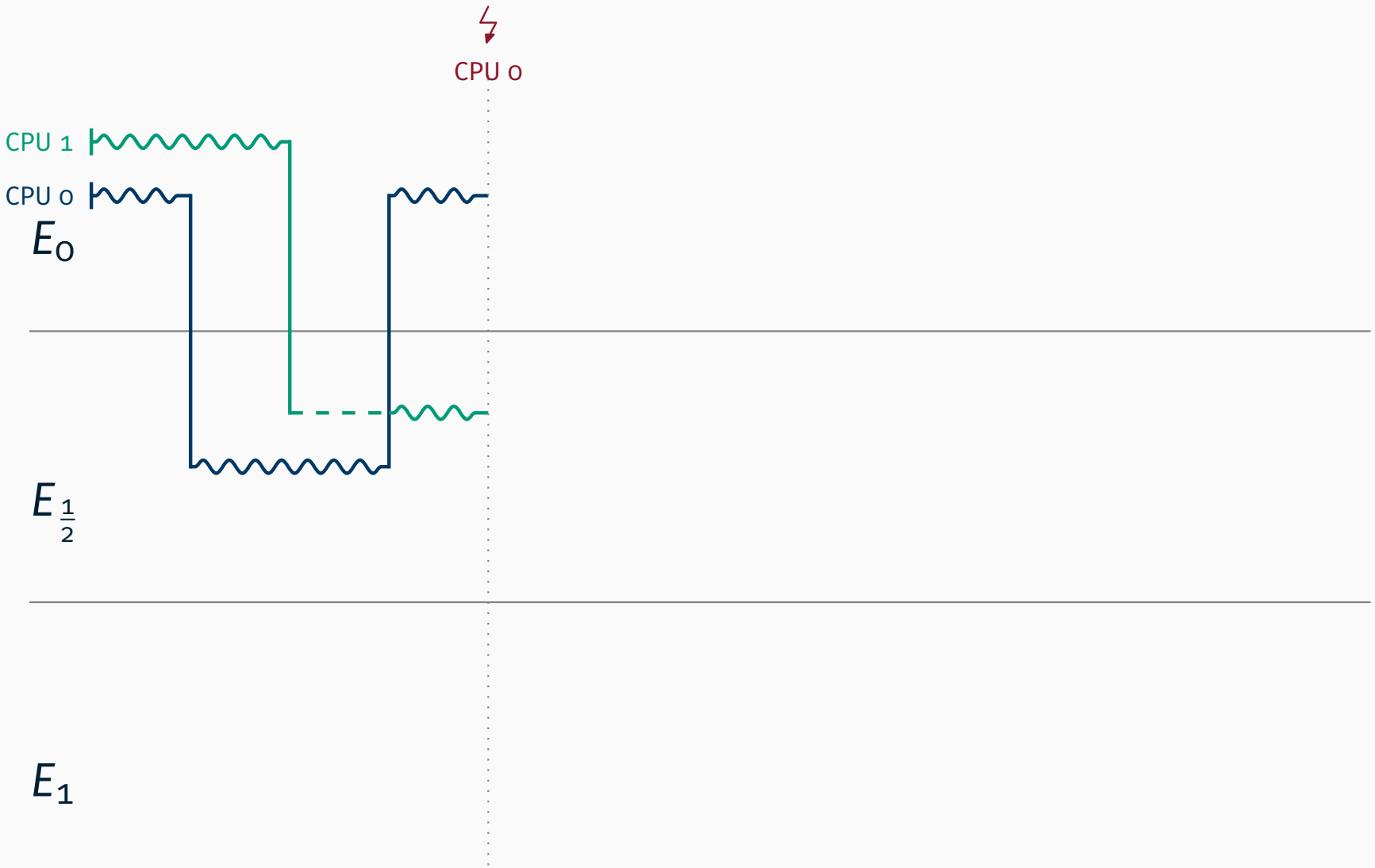
Beispiel für Mehrkernprozessoren



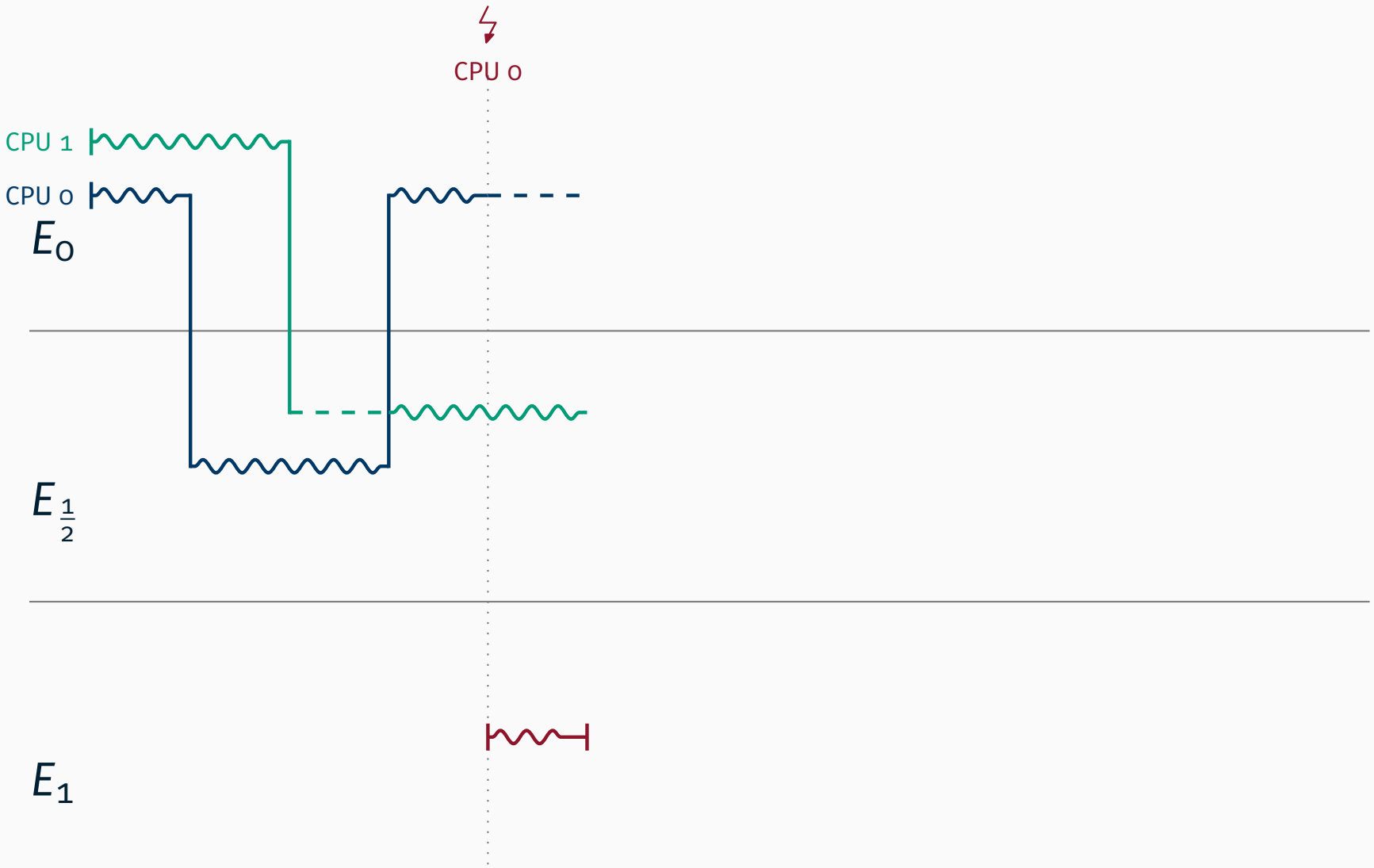
Beispiel für Mehrkernprozessoren



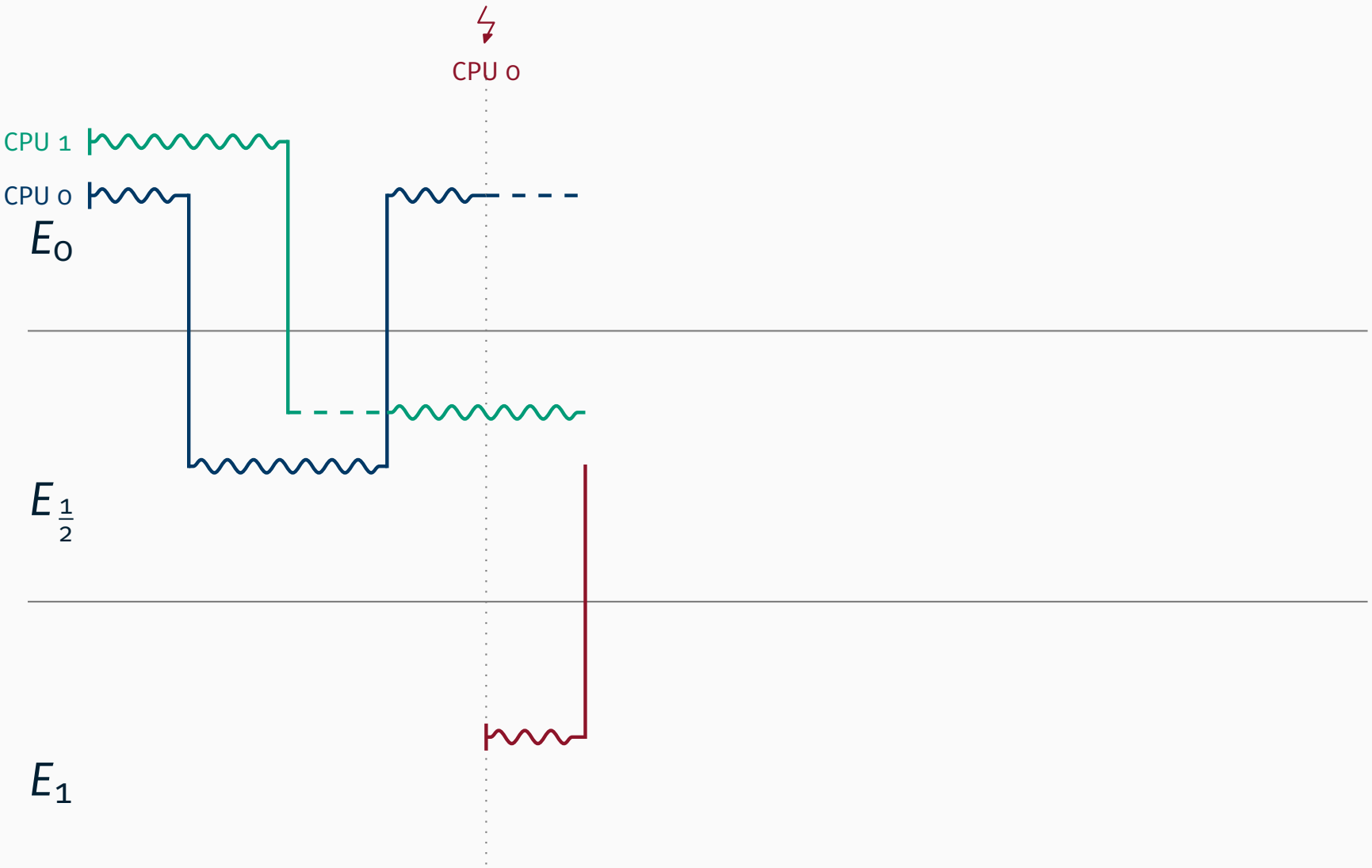
Beispiel für Mehrkernprozessoren



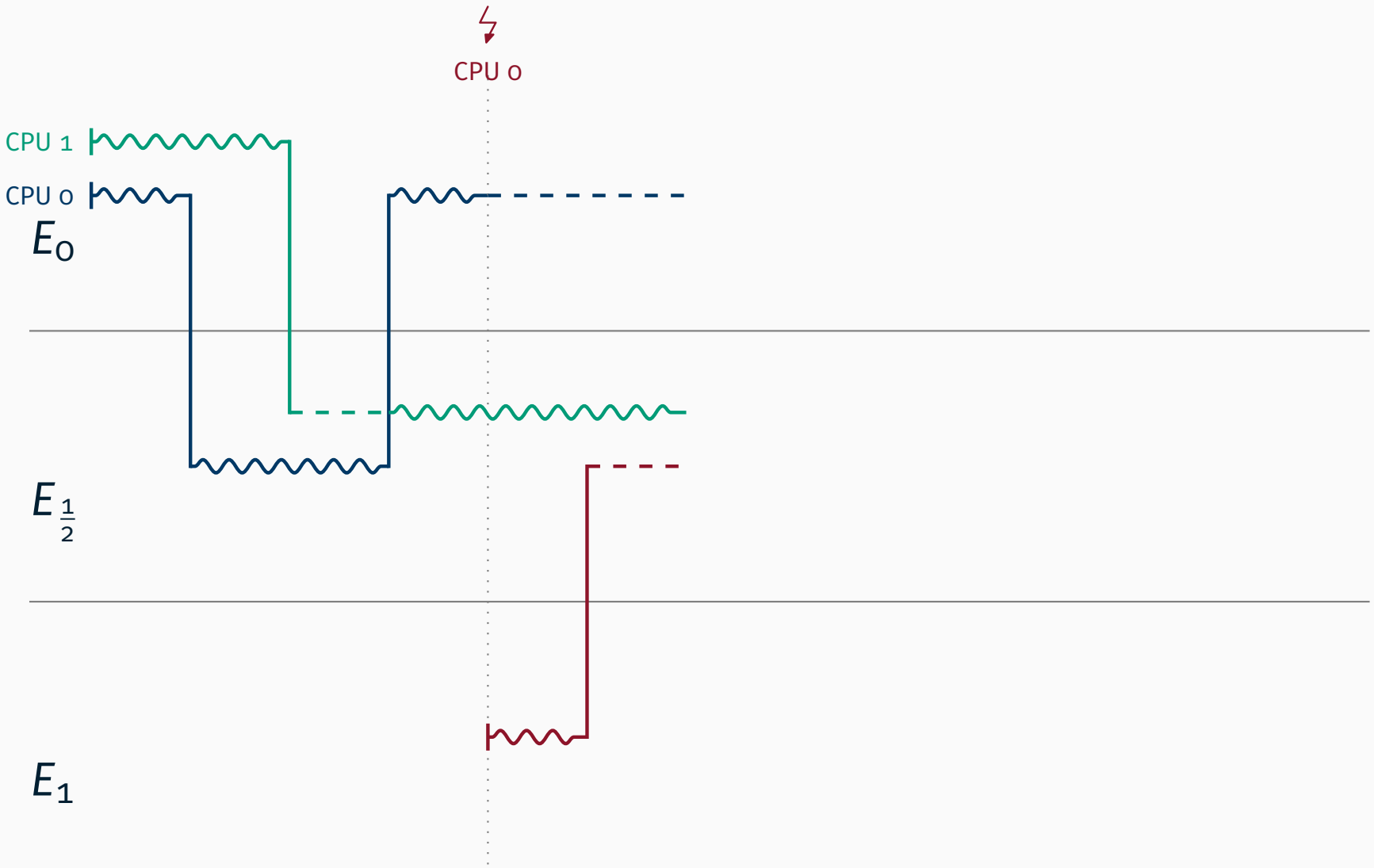
Beispiel für Mehrkernprozessoren



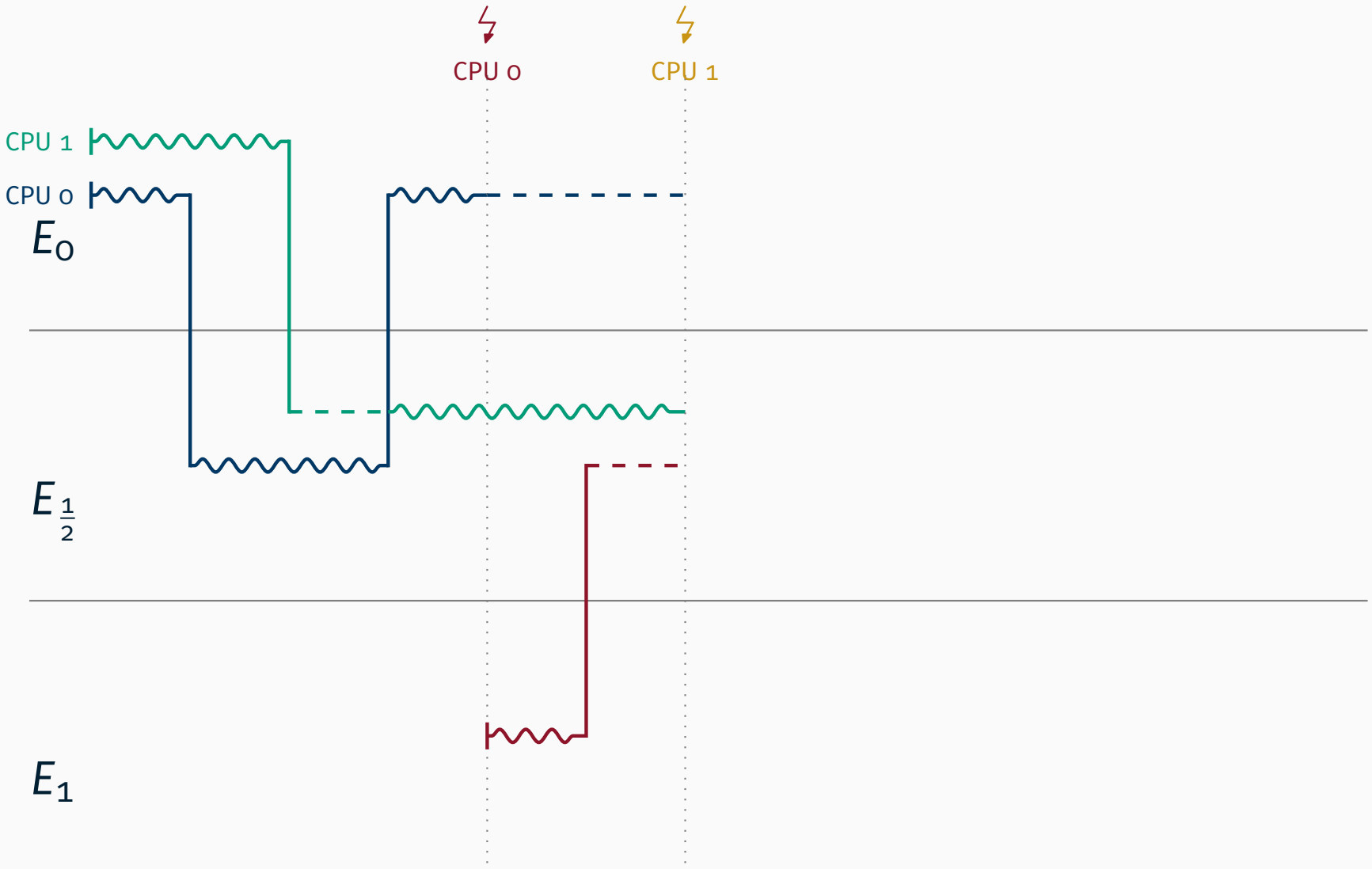
Beispiel für Mehrkernprozessoren



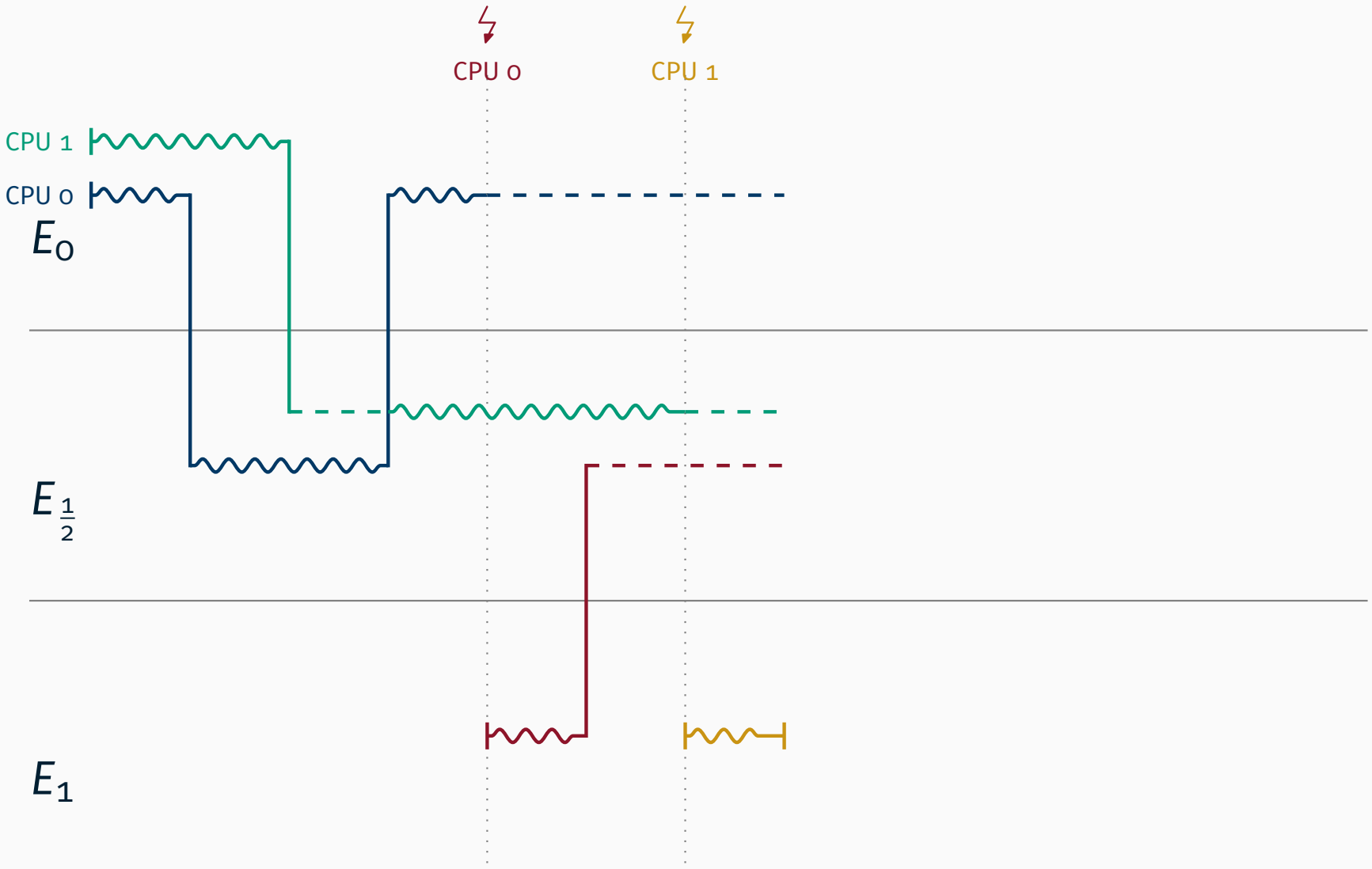
Beispiel für Mehrkernprozessoren



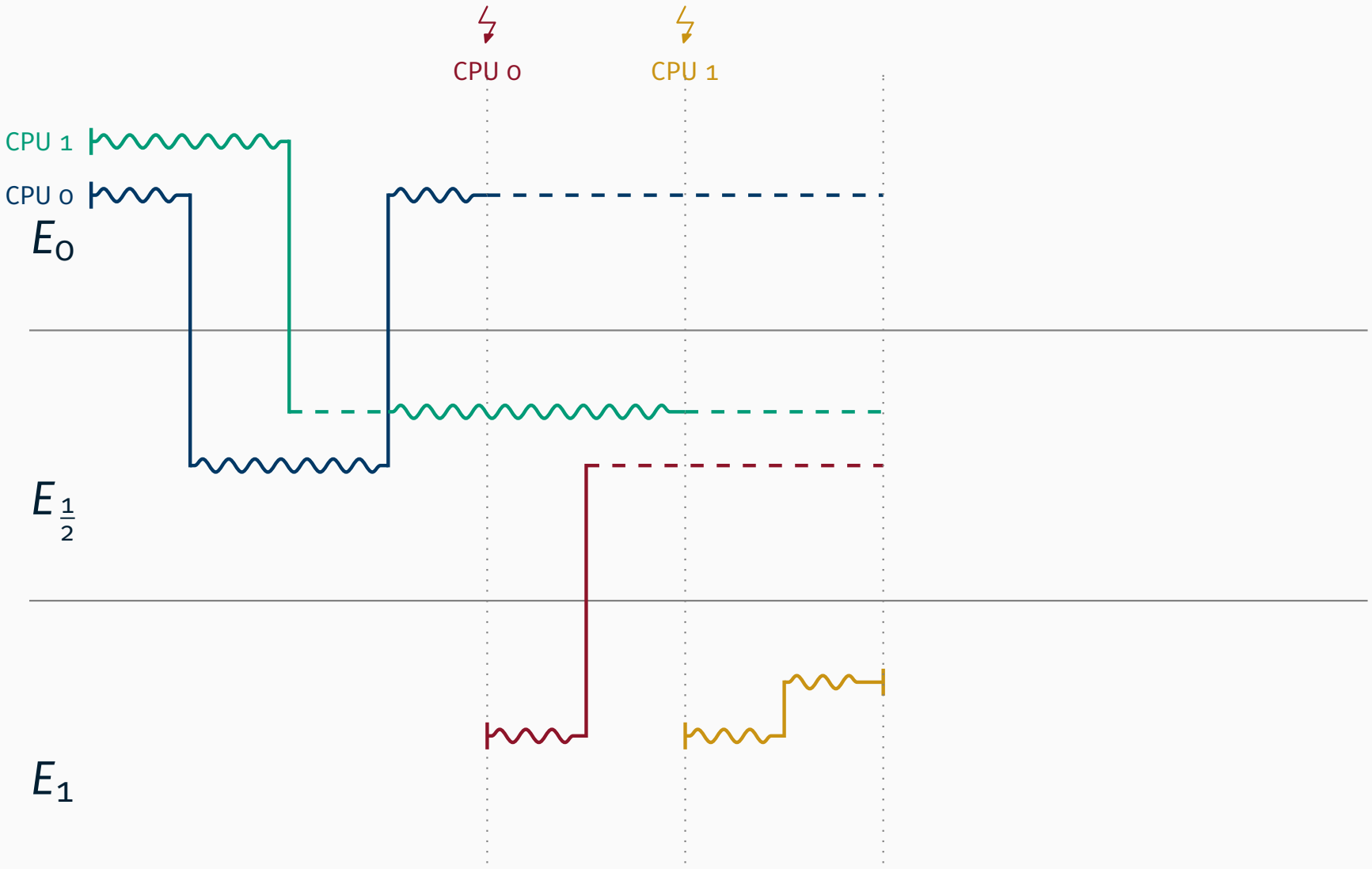
Beispiel für Mehrkernprozessoren



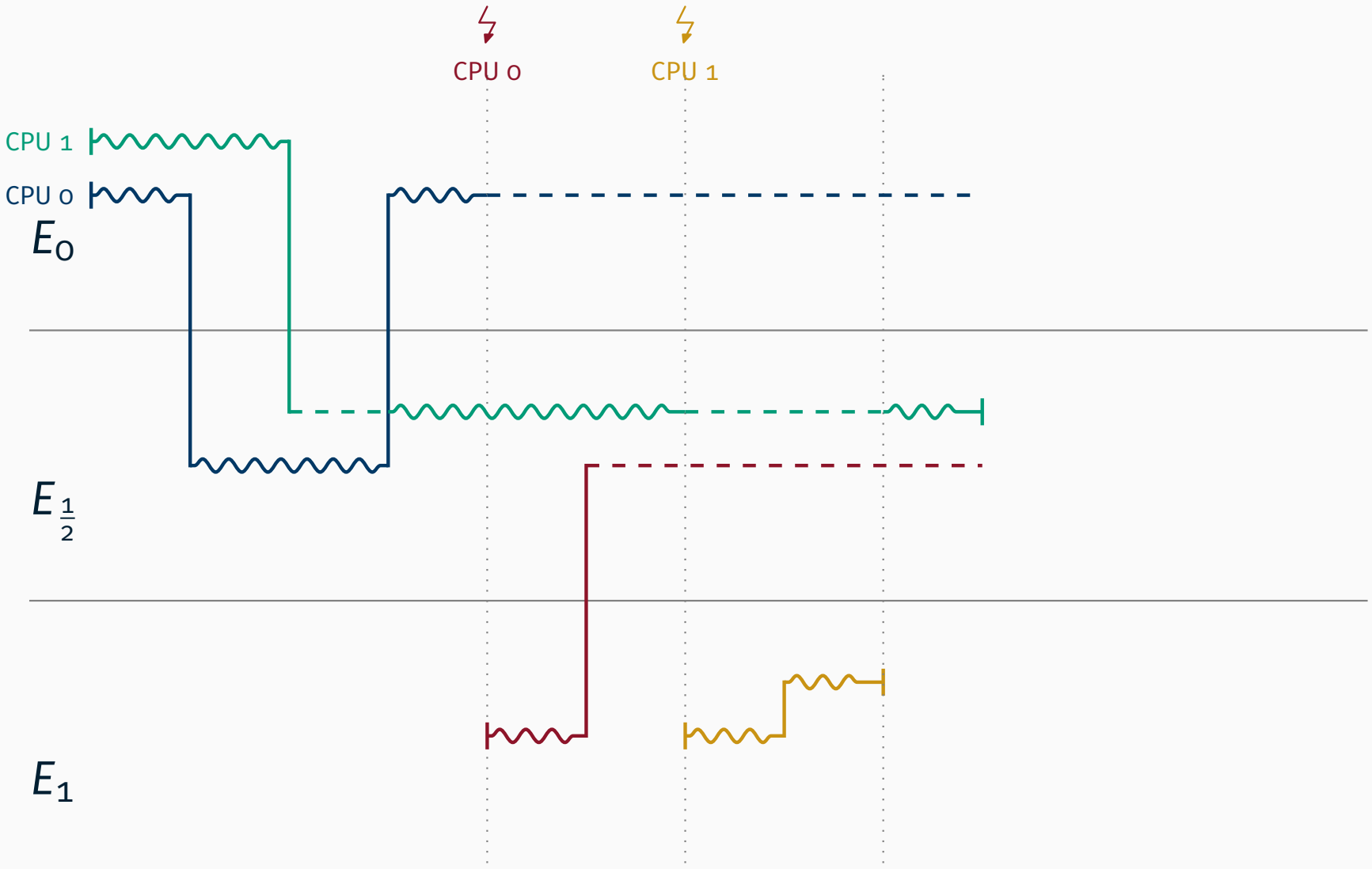
Beispiel für Mehrkernprozessoren



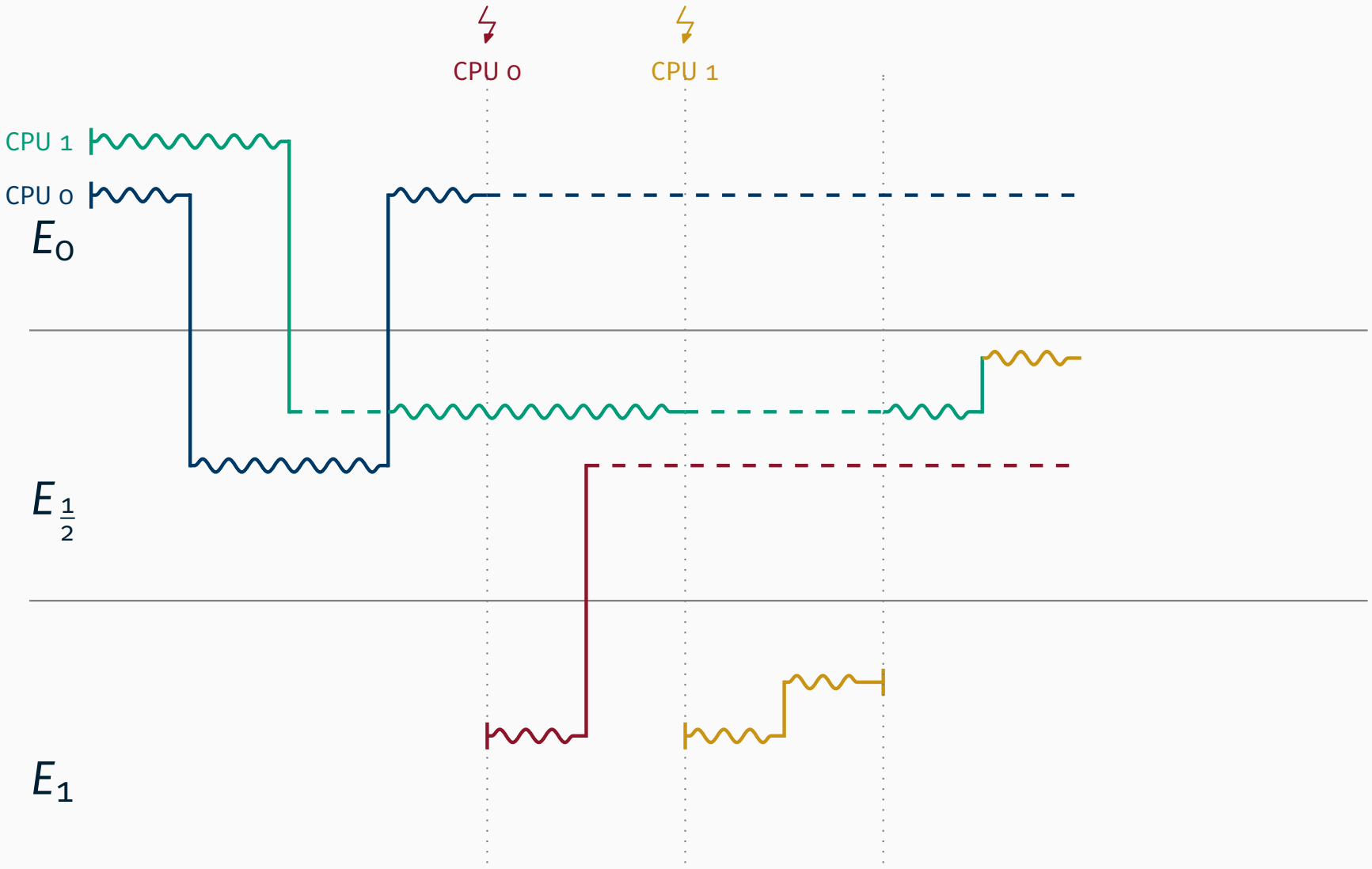
Beispiel für Mehrkernprozessoren



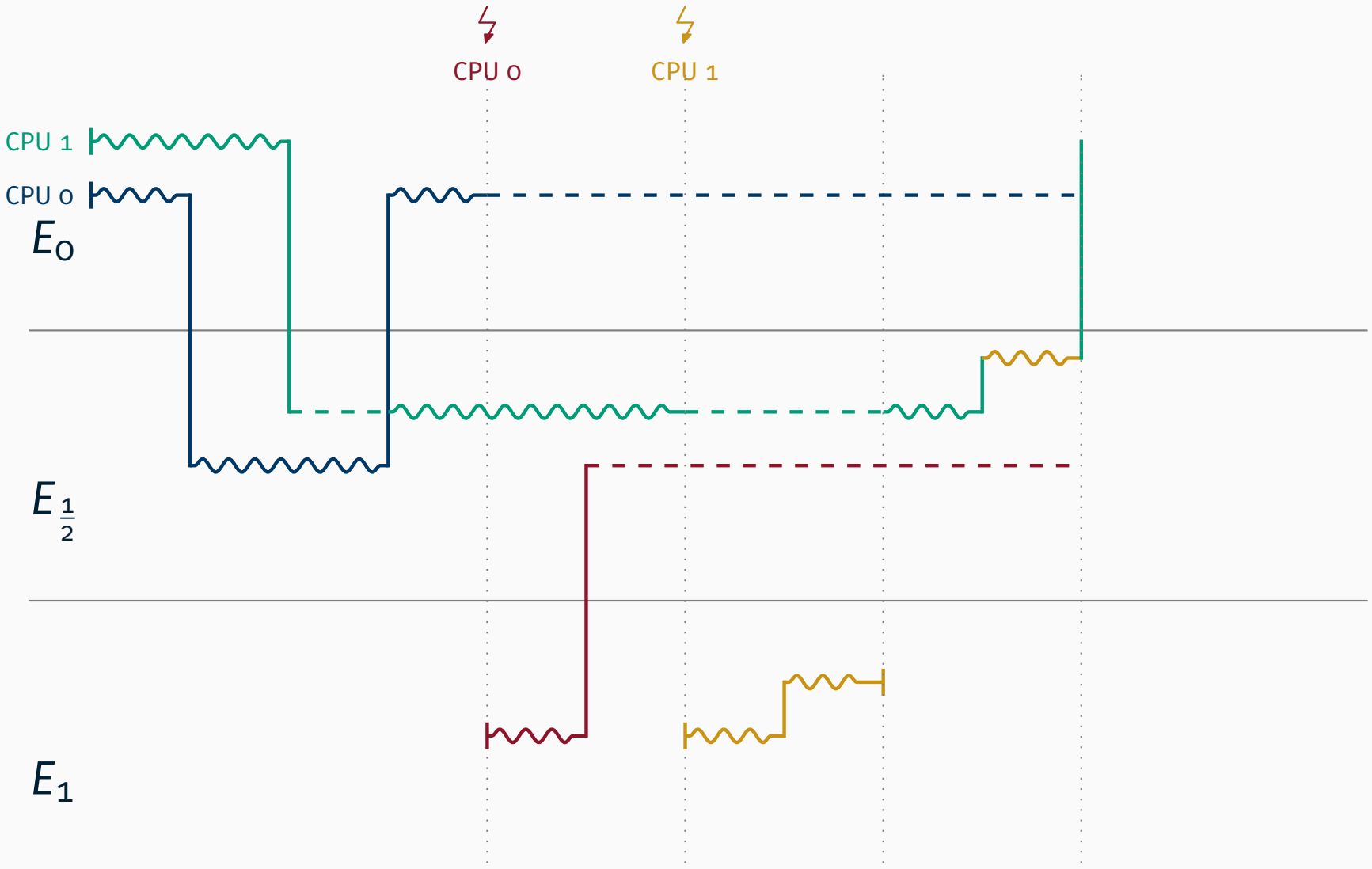
Beispiel für Mehrkernprozessoren



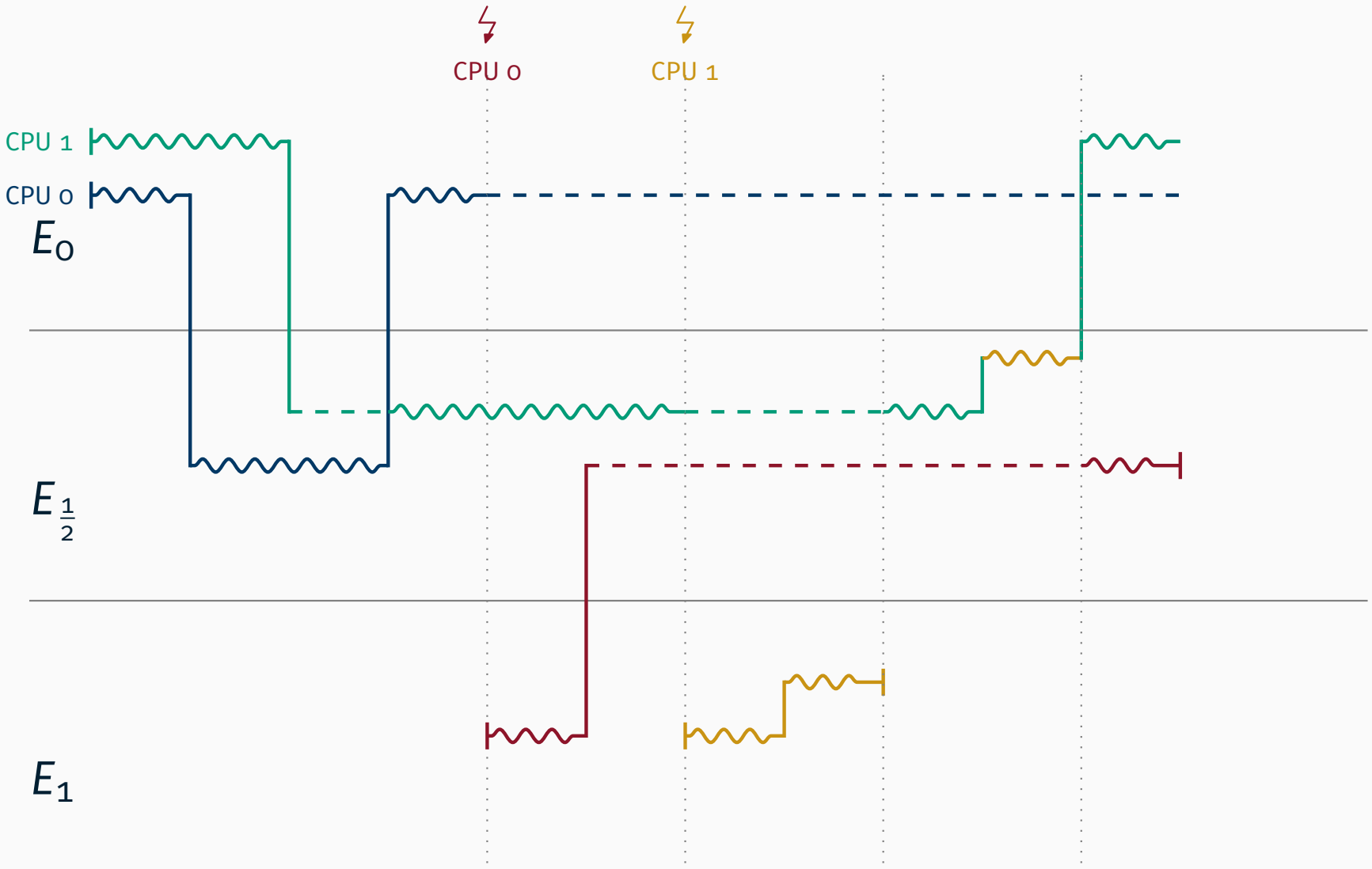
Beispiel für Mehrkernprozessoren



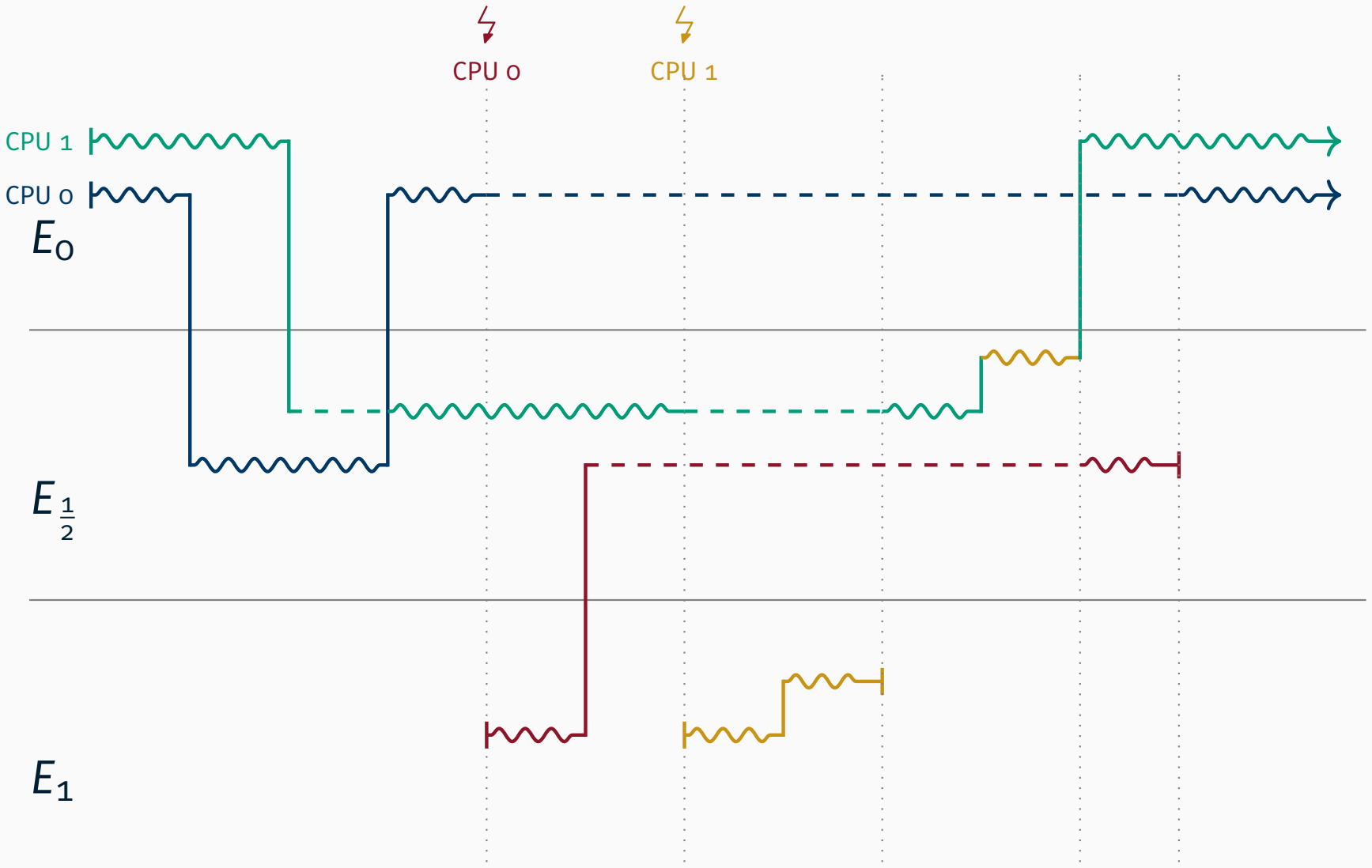
Beispiel für Mehrkernprozessoren



Beispiel für Mehrkernprozessoren



Beispiel für Mehrkernprozessoren



Fragen?

Nächste Woche (29. November & 1. Dezember)
Abgabe von Aufgabe 2 im Huber-CIP