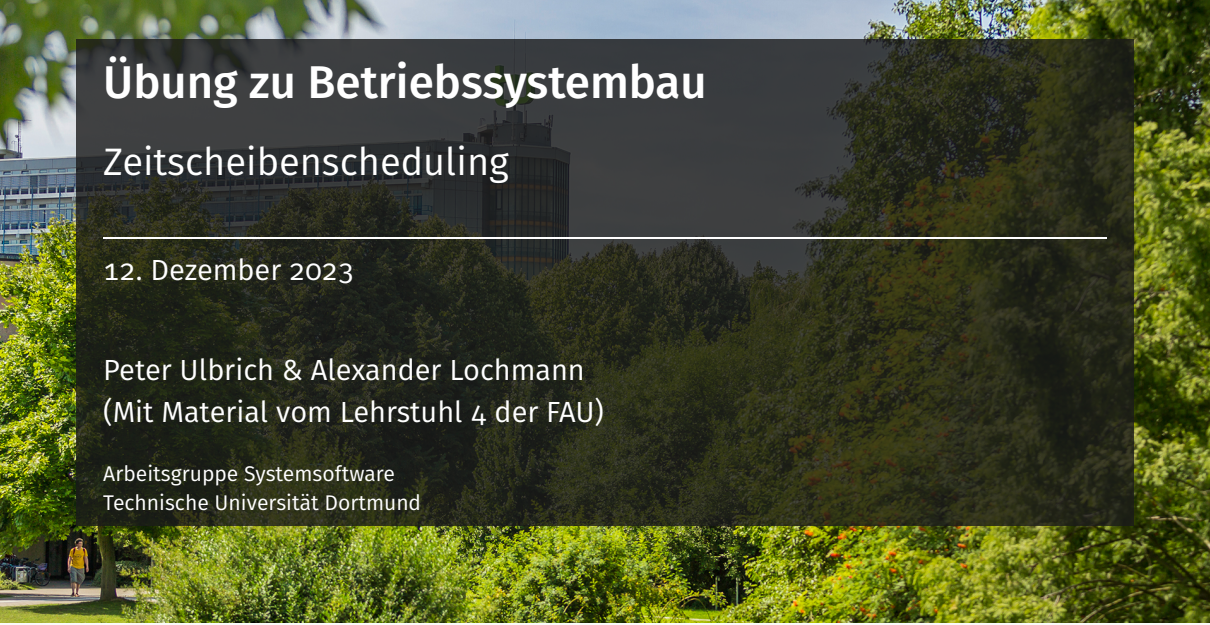


Übung zu Betriebssystembau

Zeitscheibenscheduling

12. Dezember 2023

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

Arbeitsgruppe Systemsoftware
Technische Universität Dortmund

Motivation

Kooperative Ablaufplanung

```
01 int i = 0;
02 while (true){
03     {
04         Guarded _;
05         kout << i++ << endl;
06     }
07     scheduler.resume();
08 }
```

Unterbrechende Ablaufplanung

```
01 int i = 0;
02 while (true){
03     {
04         Guarded _;
05         kout << i++ << endl;
06     }
07
08 }
```

Unterbrechende Ablaufplanung

```
01 int i = 0;
02 while (true){
03     {
04         Guarded _;
05         kout << i++ << endl;
06     }
07 ——————————————————————————— ϟ Scheduler Interrupt
08 }
```

Unterbrechende Ablaufplanung

```
01 int i = 0;
02 while (true){
03     {
04         Guarded _;
05         kout << i++ << endl;
06     }
07 ——————————————————————————— ϟ Scheduler Interrupt
08 }
```

Aufgabe: **Präemptives Scheduling mittels Timer.**

Zeitgeber

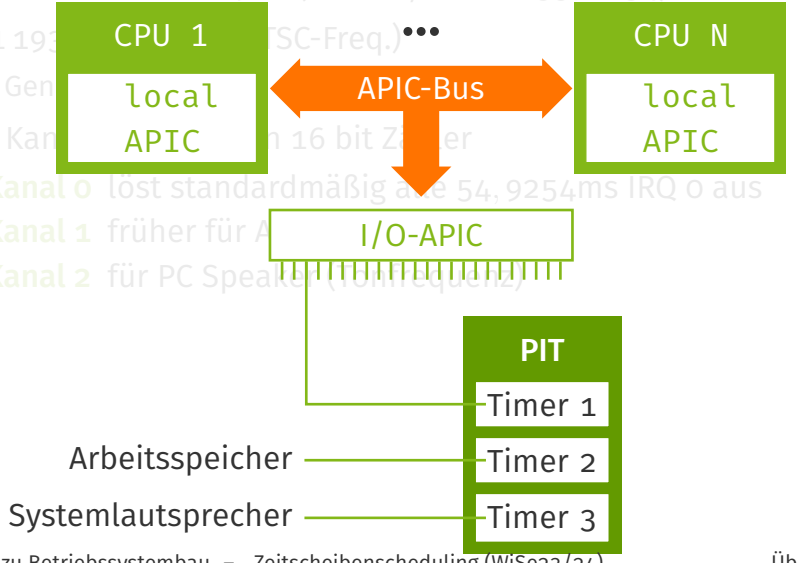
Programmable Interval Timer (PIT)

- Standardtimer seit 1981 (IBM-PC, Intel 8253/8254)
- ca. 1 193 182 Hz ($= \frac{1}{3}$ NTSC-Freq.)
  - → Genauigkeit: 838 ns
- drei Kanäle mit je einen 16 bit Zähler
  - **Kanal 0** löst standardmäßig alle $54,9254$ms IRQ 0 aus
  - **Kanal 1** früher für Arbeitsspeicher
  - **Kanal 2** für PC Speaker (Tonfrequenz)

Programmable Interval Timer (PIT)

- Standardtimer seit 1981 (IBM-PC, Intel 8253/8254)
- ca. 1 193 182 Hz (= $\frac{1}{3}$ NTSC-Freq.)
  - → Genauigkeit
- drei Kanäle mit je einen 16 bit Zähler
  - **Kanal 0** löst standardmäßig alle 54,9254ms IRQ 0 aus
  - **Kanal 1** früher für Arbeitsspeicher
  - **Kanal 2** für PC Speaker (Tonfrequenz)

Programmable Interval Timer (PIT)



Programmable Interval Timer (PIT)

- Standardtimer seit 1981 (IBM-PC, Intel 8253/8254)
- ca. 1 193 182 Hz ($= \frac{1}{3}$ NTSC-Freq.)
  - ➔ Genauigkeit: 838 ns
- drei Kanäle mit je einen 16 bit Zähler
  - **Kanal 0** löst standardmäßig alle $54,9254$ms IRQ 0 aus
  - **Kanal 1** früher für Arbeitsspeicher
  - **Kanal 2** für PC Speaker (Tonfrequenz)
- via PIC bzw. I/O APIC ➔ (relativ) langsam

Programmable Interval Timer (PIT)

- Standardtimer seit 1981 (IBM-PC, Intel 8253/8254)
- ca. 1 193 182 Hz ($= \frac{1}{3}$ NTSC-Freq.)
  - → Genauigkeit: 838 ns
- drei Kanäle mit je einen 16 bit Zähler
  - **Kanal 0** löst standardmäßig alle 54,9254ms IRQ 0 aus
  - **Kanal 1** früher für Arbeitsspeicher
  - **Kanal 2** für PC Speaker (Tonfrequenz)
- via PIC bzw. I/O APIC → (relativ) langsam
- *ausreichend für frühere BSB-Varianten,* ***geht aber besser.***

→ `machine/pit.h`

Real Time Clock (RTC)

- seit 1984 (IBM-PC/AT)
- 32 768 Hz (= 2^{15} Hz, Verwendung in Uhren)
  - Standardmäßig Interrupts bei 1 024 Hz (fast 1 ms)
  - 12 weitere Möglichkeiten von 2 bis 8 192 Hz durch Vorteiler
  - IRQ 8
- für Zeit & Datum
- Betrieb im ausgeschalteten Zustand mittels Batterie

Time Stamp Counter (TSC)

- seit 1993 (Pentium)
- 64 bit, auslesbar über Assemblerinstruktion `rdtsc`
- Taktfrequenz wie CPU
  - ursprünglich Erhöhung mit jedem Clock-Signal
  - unterschiedliche Takte abhängig vom Stromsparmodus
  - bei neueren Versionen: konstante Rate entsprechend nominaler Geschwindigkeit
- kann keinen Interrupt auslösen

→ `machine/tsc.h`

ACPI Power Management Timer

- seit es ACPI-Mainboards gibt (1996)
- 3 579 545 Hz (= NTSC-Freq.)
- ein 24 oder 32 bit Zähler
  - besser als alte (nicht konstante) TSC
  - Zugriff über I/O Port
- kann auch keinen Interrupt auslösen

High Precision Event Timer (HPET)

- von Intel und Microsoft 2005 als PIT- & RTC-Ersatz veröffentlicht
- $\geq$ 10 MHz
  - ➔ Genauigkeit: 100 ns oder besser
- ein 64 bit Zähler
  - min. drei 32 oder 64 bit breite Vergleichseinrichtungen
  - konfigurierbarer Interrupt bei Gleichheit

LAPIC Timer

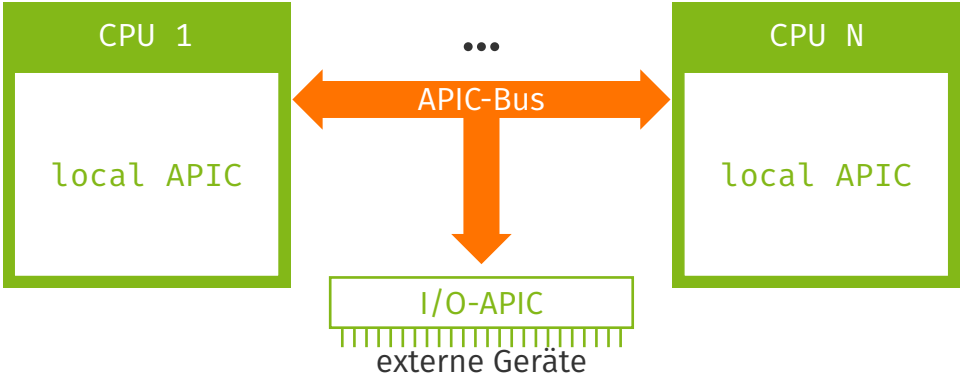
- $\geq$ 100 MHz
  - ➔ Genauigkeit: 10 ns oder besser
- 32 bit Zähler
- verwendet Busfrequenz
  - abhängig vom System
  - aber unabhängig von Stromsparmodus
  - Interrupt geht nur an den entsprechenden Kern
  - 8 Möglichkeiten (bis $\frac{1}{128}$ Busfrequenz) durch Vorteiler

LAPIC Timer

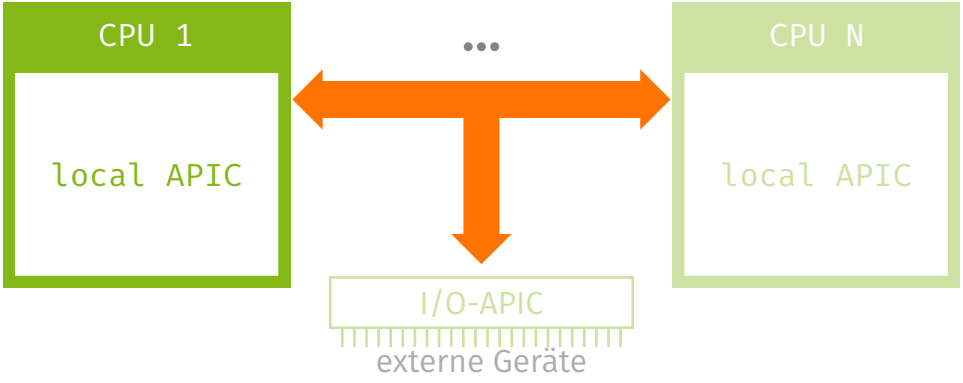
- $\geq$ 100 MHz
  - ➔ Genauigkeit: 10 ns oder besser
- 32 bit Zähler
- verwendet Busfrequenz
  - abhängig vom System
  - aber unabhängig von Stromsparmodus
  - Interrupt geht nur an den entsprechenden Kern
  - 8 Möglichkeiten (bis $\frac{1}{128}$ Busfrequenz) durch Vorteiler

Perfekt für unsere Bedürfnisse

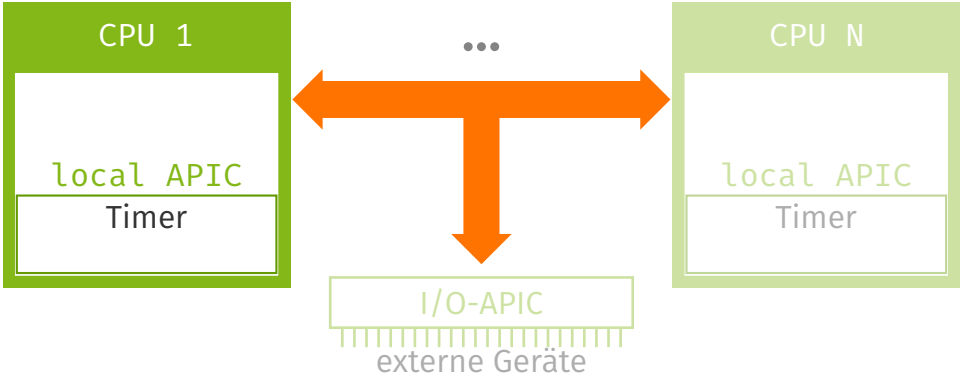
Funktionsweise des LAPIC Timers



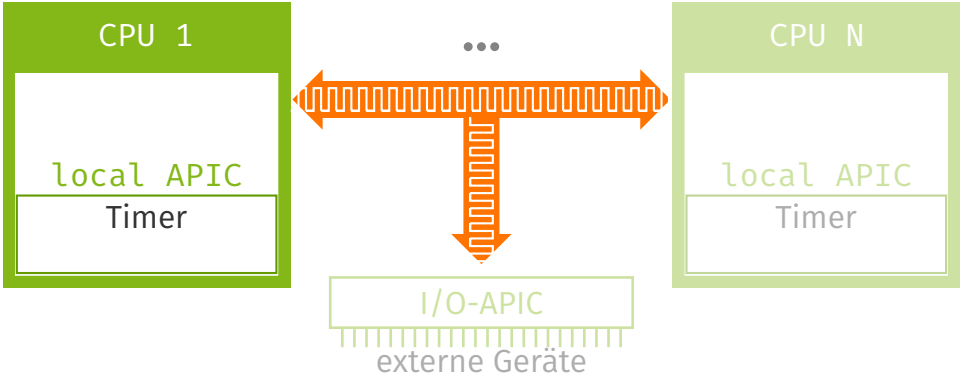
Funktionsweise des LAPIC Timers



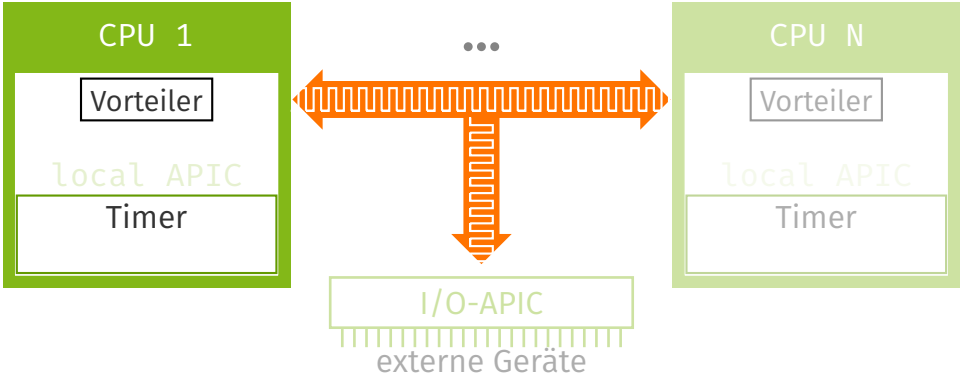
Funktionsweise des LAPIC Timers



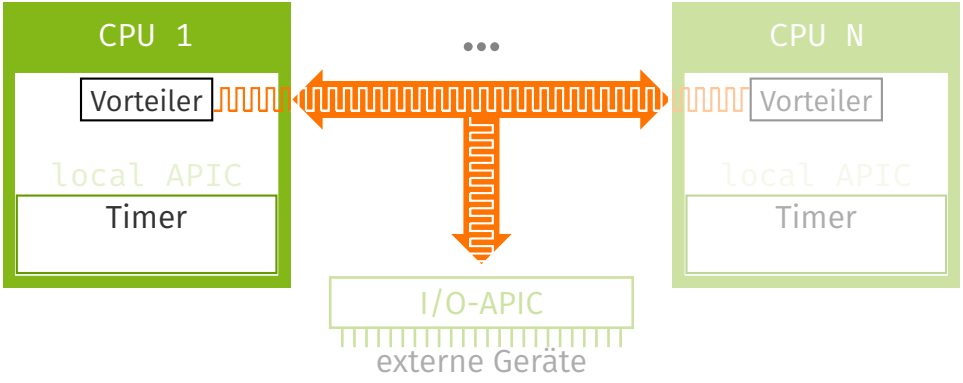
Funktionsweise des LAPIC Timers



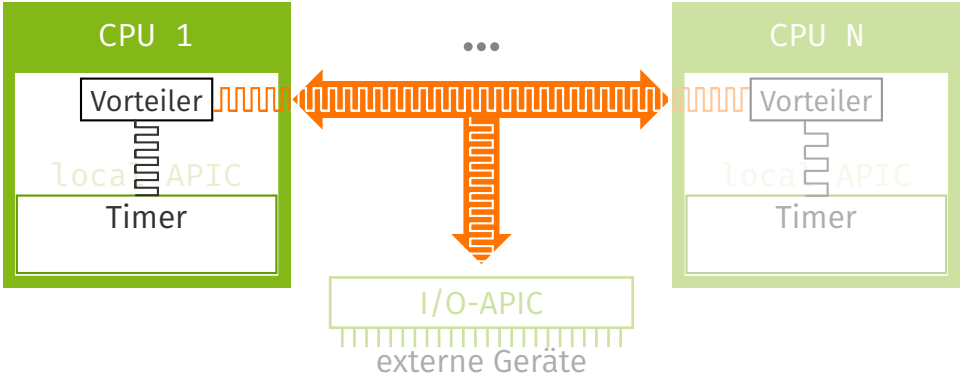
Funktionsweise des LAPIC Timers



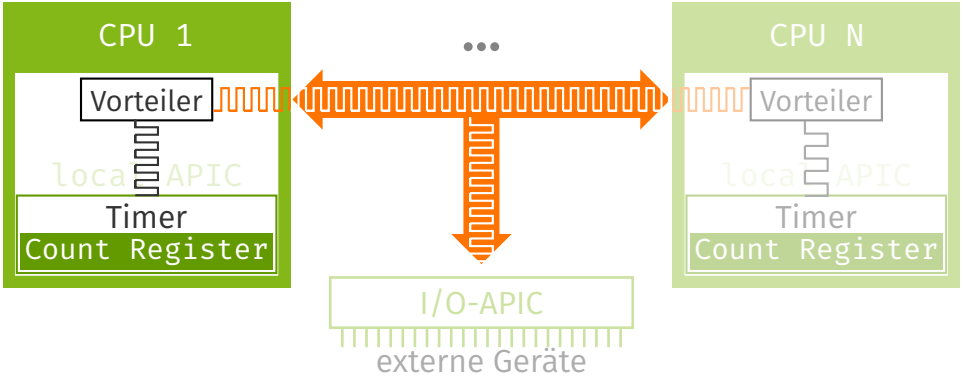
Funktionsweise des LAPIC Timers



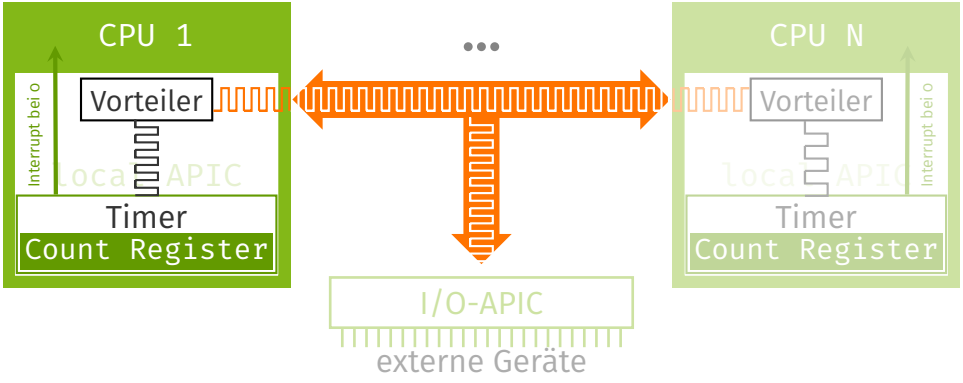
Funktionsweise des LAPIC Timers



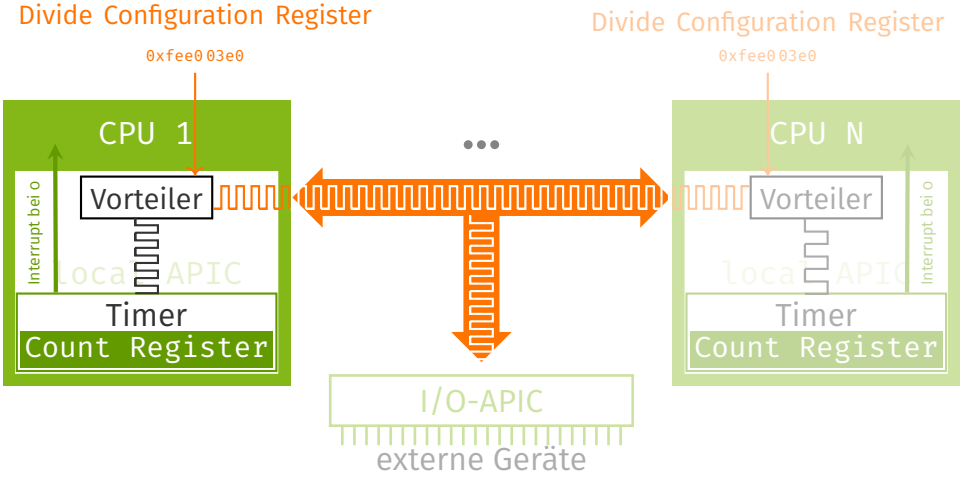
Funktionsweise des LAPIC Timers



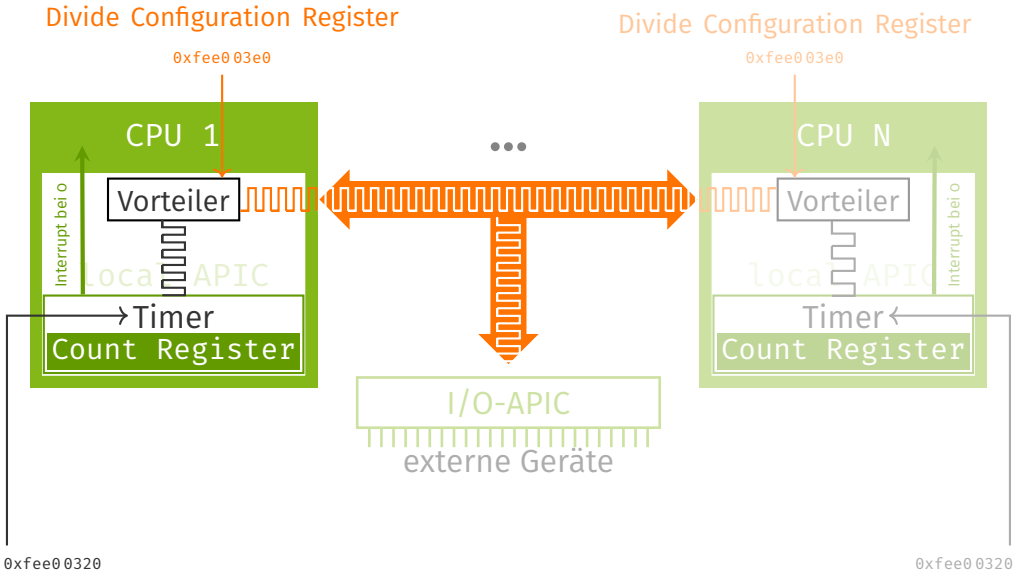
Funktionsweise des LAPIC Timers



Funktionsweise des LAPIC Timers



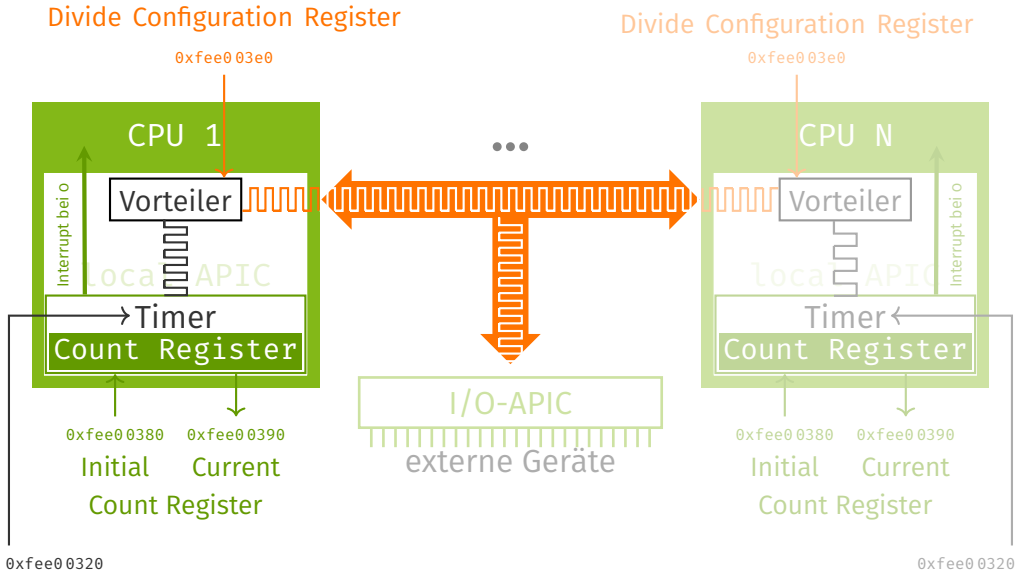
Funktionsweise des LAPIC Timers



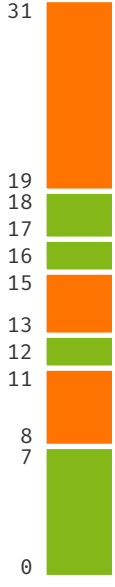
Timer Control Register

Timer Control Register

Funktionsweise des LAPIC Timers



Aufbau des Timer Control Register Eintrags



Betriebsmodus: einmalig (`0`) oder periodisch (`1`)
Interrupt-Mask: Interrupt aktiv (`0`) oder inaktiv (`1`)

Zustellstatus Interrupt Nachricht noch unterwegs? **(RO)**

Interrupt Vektor: Nummer in der Vektortabelle (32-255)

Zusammenfassung LAPIC Timer

- jede CPU hat einen eigenen 32bit Timer
- Änderung am Initial Count Register startet den Timer
- zu Beginn wird der initiale Startzählwert aus dem Initial Count Register in das Current Count Register kopiert
- welches im $\frac{Bustakt}{Vorteiler}$ dekrementiert wird
- bei 0 wird – sofern aktiviert – ein Interrupt ausgelöst
- je nach Betriebsmodus wird gestoppt oder wieder neu begonnen

Umsetzung

Watch

windup stellt das Unterbrechungsintervall ein
(z.B. alle 1000 Mikrosekunden)

activate setzt den Timer und aktiviert Interrupts

prologue fordert Epilog an
(und kann zu Testzwecken eine Ausgabe tätigen)

epilogue wechselt die Anwendung mittels `scheduler.resume()`

LAPIC-Timer einstellen

1. LAPIC-Timer kalibrieren

LAPIC-Timer einstellen

1. LAPIC-Timer kalibrieren

- in der Funktion `LAPIC::Timer::ticks`
  (Funktion gibt die Anzahl der Ticks in einer **Milli**sekunde zurück)

LAPIC-Timer einstellen

1. LAPIC-Timer kalibrieren

- in der Funktion `LAPIC::Timer::ticks`
  (Funktion gibt die Anzahl der Ticks in einer **Milli**sekunde zurück)
- benötigt zur Konfiguration die Funktion `LAPIC::Timer::set`, Aufruf mit
  - maximalem Wert für den LAPIC-Zähler,
  - als `ONE_SHOT` und
  - ohne Unterbrechungen

LAPIC-Timer einstellen

1. LAPIC-Timer kalibrieren

- in der Funktion `LAPIC::Timer::ticks`
  (Funktion gibt die Anzahl der Ticks in einer **Milli**sekunde zurück)
- benötigt zur Konfiguration die Funktion `LAPIC::Timer::set`, Aufruf mit
  - maximalem Wert für den LAPIC-Zähler,
  - als `ONE_SHOT` und
  - ohne Unterbrechungen
- unter Verwendung des PIT
  - Wartezeit von mehreren MS einstellen: `PIT::set`
  - mittels `PIT::waitForTimeout` warten
  - Start- und Endwert des LAPIC-Zählerregisters merken

LAPIC-Timer einstellen

1. LAPIC-Timer kalibrieren

- in der Funktion `LAPIC::Timer::ticks`
  (Funktion gibt die Anzahl der Ticks in einer **Milli**sekunde zurück)
- benötigt zur Konfiguration die Funktion `LAPIC::Timer::set`, Aufruf mit
  - maximalem Wert für den LAPIC-Zähler,
  - als `ONE_SHOT` und
  - ohne Unterbrechungen
- unter Verwendung des PIT
  - Wartezeit von mehreren MS einstellen: `PIT::set`
  - mittels `PIT::waitForTimeout` warten
  - Start- und Endwert des LAPIC-Zählerregisters merken
- Hilfsstrukturen in `lapic_timer.cc` & `lapic_registers.h`

LAPIC-Timer einstellen

2. Initialen Wert und Vorteiler korrekt setzen

LAPIC-Timer einstellen

2. Initialen Wert und Vorteiler korrekt setzen

- Interrupt soll alle *n* Mikrosekunden ausgelöst werden

LAPIC-Timer einstellen

2. Initialen Wert und Vorteiler korrekt setzen

- Interrupt soll alle n Mikrosekunden ausgelöst werden
- Startwertzähler $\mathit{initial} = \frac{n \cdot \texttt{LAPIC::Timer::ticks()}}{\mathit{Vorteiler} \cdot 1000}$ berechnen, dabei gilt $\mathit{Vorteiler} = 2^x$ mit $x \in \{0, \ldots, 7\}$

LAPIC-Timer einstellen

2. Initialen Wert und Vorteiler korrekt setzen

- Interrupt soll alle n Mikrosekunden ausgelöst werden
- Startwertzähler $\mathit{initial} = \frac{n \cdot \texttt{LAPIC::Timer::ticks()}}{\mathit{Vorteiler} \cdot 1000}$ berechnen, dabei gilt $\mathit{Vorteiler} = 2^x$ mit $x \in \{0, \ldots, 7\}$
- Möglichst kleiner Vorteiler, aber kein Überlauf (32 bit!)

LAPIC-Timer einstellen

2. Initialen Wert und Vorteiler korrekt setzen

- Interrupt soll alle n Mikrosekunden ausgelöst werden
- Startwertzähler $\mathit{initial} = \frac{n \cdot \texttt{LAPIC::Timer::ticks()}}{\mathit{Vorteiler} \cdot 1000}$ berechnen, dabei gilt $\mathit{Vorteiler} = 2^x$ mit $x \in \{0, \ldots, 7\}$
- Möglichst kleiner Vorteiler, aber kein Überlauf (32 bit!)
- **Beispiel:** `watch.windup(5000000);`

n	$5\,000\,000\ \mu s = 5\ s$
`LAPIC::Timer::ticks`	$1\,000\,000\ ms^{-1}$

LAPIC-Timer einstellen

2. Initialen Wert und Vorteiler korrekt setzen

- Interrupt soll alle n Mikrosekunden ausgelöst werden
- Startwertzähler $\mathit{initial} = \frac{n \cdot \texttt{LAPIC::Timer::ticks()}}{\mathit{Vorteiler} \cdot 1000}$ berechnen, dabei gilt $\mathit{Vorteiler} = 2^x$ mit $x \in \{0, \ldots, 7\}$
- Möglichst kleiner Vorteiler, aber kein Überlauf (32 bit!)
- **Beispiel:** `watch.windup(5000000);`

n	$5\,000\,000\ \mu s = 5\ s$
`LAPIC::Timer::ticks`	$1\,000\,000\ ms^{-1}$
$\mathit{Vorteiler}$	$1 = 2^0$

LAPIC-Timer einstellen

2. Initialen Wert und Vorteiler korrekt setzen

- Interrupt soll alle n Mikrosekunden ausgelöst werden
- Startwertzähler $initial = \frac{n \cdot \texttt{LAPIC::Timer::ticks()}}{Vorteiler \cdot 1000}$ berechnen, dabei gilt $Vorteiler = 2^x$ mit $x \in \{0, \ldots, 7\}$
- Möglichst kleiner Vorteiler, aber kein Überlauf (32 bit!)
- **Beispiel:** `watch.windup(5000000);`

n	5 000 000 µs = 5 s
`LAPIC::Timer::ticks`	1 000 000 ms^{-1}
$Vorteiler$	$1 = 2^0$
$initial$	5 000 000 000

LAPIC-Timer einstellen

2. Initialen Wert und Vorteiler korrekt setzen

- Interrupt soll alle n Mikrosekunden ausgelöst werden
- Startwertzähler $initial = \frac{n \cdot \texttt{LAPIC::Timer::ticks()}}{Vorteiler \cdot 1000}$ berechnen, dabei gilt $Vorteiler = 2^x$ mit $x \in \{0, \ldots, 7\}$
- Möglichst kleiner Vorteiler, aber kein Überlauf (32 bit!)
- **Beispiel:** `watch.windup(5000000);`

n	$5\,000\,000\ \mu s = 5\ s$
`LAPIC::Timer::ticks`	$1\,000\,000\ ms^{-1}$
$Vorteiler$	$1 = 2^0$
$initial$	$5\,000\,000\,000$

LAPIC-Timer einstellen

2. Initialen Wert und Vorteiler korrekt setzen

- Interrupt soll alle n Mikrosekunden ausgelöst werden
- Startwertzähler $initial = \frac{n \cdot \texttt{LAPIC::Timer::ticks()}}{Vorteiler \cdot 1000}$ berechnen, dabei gilt $Vorteiler = 2^x$ mit $x \in \{0, \ldots, 7\}$
- Möglichst kleiner Vorteiler, aber kein Überlauf (32 bit!)
- **Beispiel:** `watch.windup(5000000);`

n	$5\,000\,000\ \mu s = 5\ s$
`LAPIC::Timer::ticks`	$1\,000\,000\ ms^{-1}$
$Vorteiler$	$2 = 2^1$
$initial$	

LAPIC-Timer einstellen

2. Initialen Wert und Vorteiler korrekt setzen

- Interrupt soll alle n Mikrosekunden ausgelöst werden
- Startwertzähler $initial = \frac{n \cdot \texttt{LAPIC::Timer::ticks()}}{Vorteiler \cdot 1000}$ berechnen, dabei gilt $Vorteiler = 2^x$ mit $x \in \{0, \ldots, 7\}$
- Möglichst kleiner Vorteiler, aber kein Überlauf (32 bit!)
- **Beispiel:** `watch.windup(5000000);`

n	5 000 000 µs = 5 s
`LAPIC::Timer::ticks`	1 000 000 ms^{-1}
$Vorteiler$	$2 = 2^1$
$initial$	2 500 000 000

LAPIC-Timer einstellen

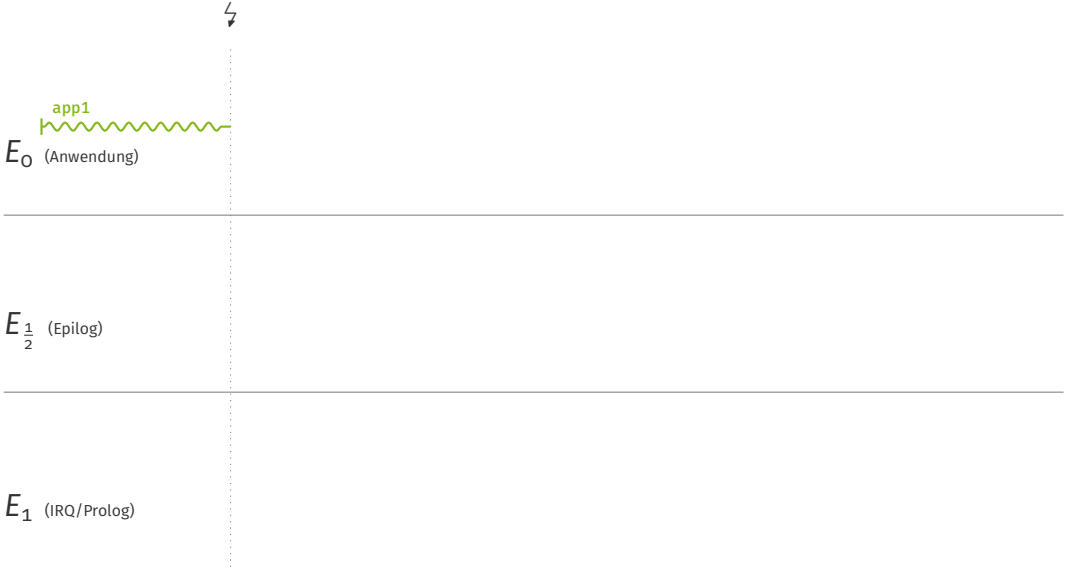
2. Initialen Wert und Vorteiler korrekt setzen

- Interrupt soll alle n Mikrosekunden ausgelöst werden
- Startwertzähler $initial = \frac{n \cdot \texttt{LAPIC::Timer::ticks()}}{Vorteiler \cdot 1000}$ berechnen, dabei gilt $Vorteiler = 2^x$ mit $x \in \{0, \ldots, 7\}$
- Möglichst kleiner Vorteiler, aber kein Überlauf (32 bit!)
- **Beispiel:** `watch.windup(5000000);`

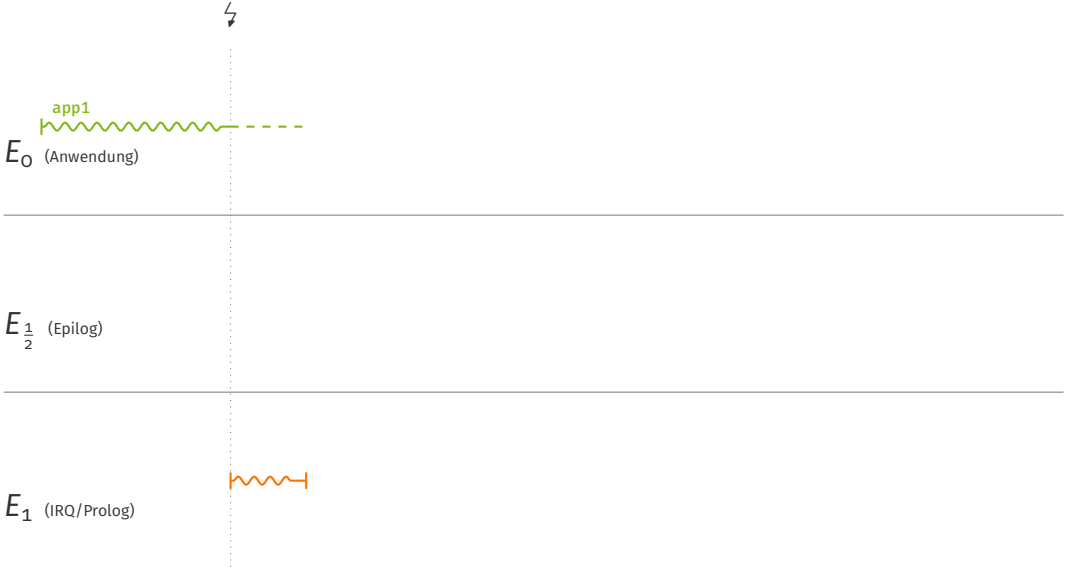
n	5 000 000 µs = 5 s
`LAPIC::Timer::ticks`	1 000 000 ms^{-1}
$Vorteiler$	$2 = 2^1$
$initial$	2 500 000 000 ✓

Ablaufbeispiel (Standardfall)

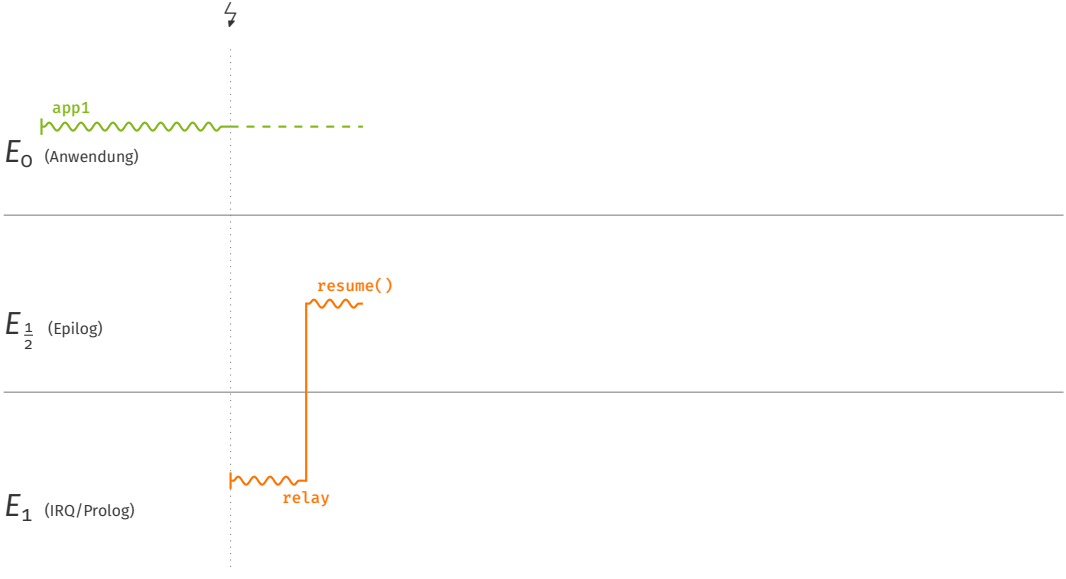
Ablaufbeispiel (Standardfall)



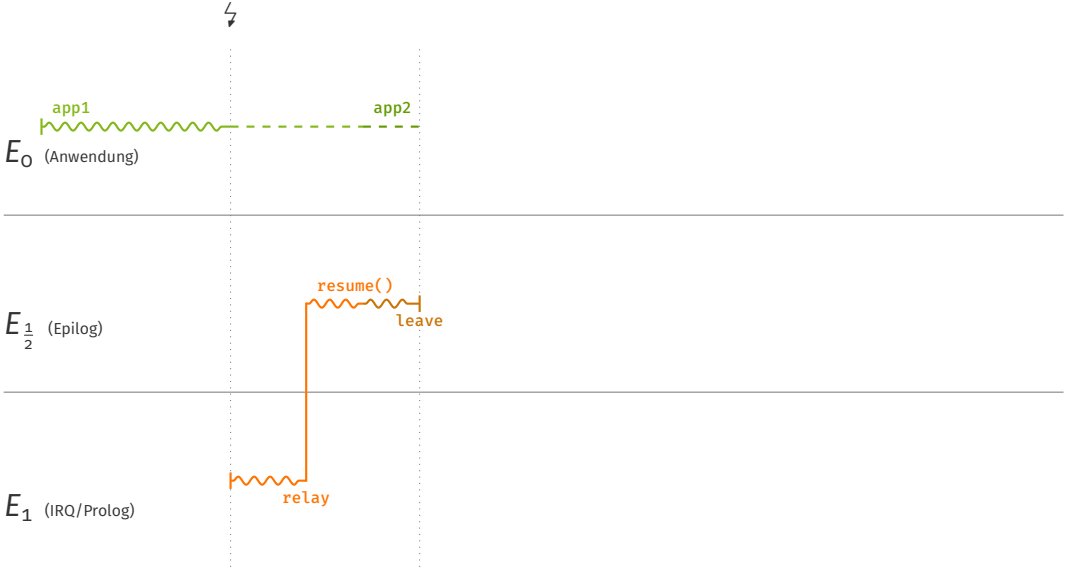
Ablaufbeispiel (Standardfall)



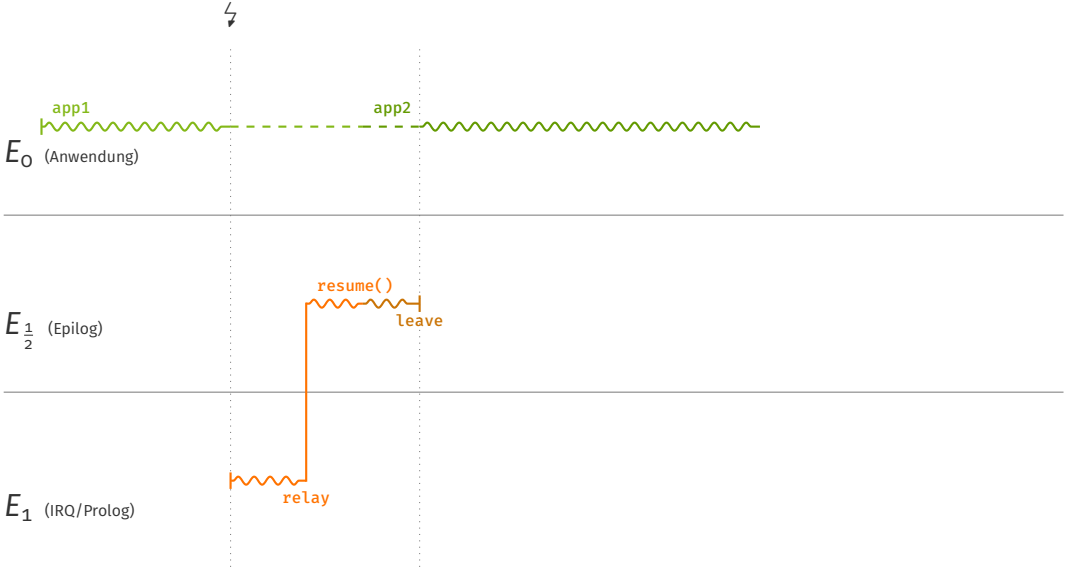
Ablaufbeispiel (Standardfall)



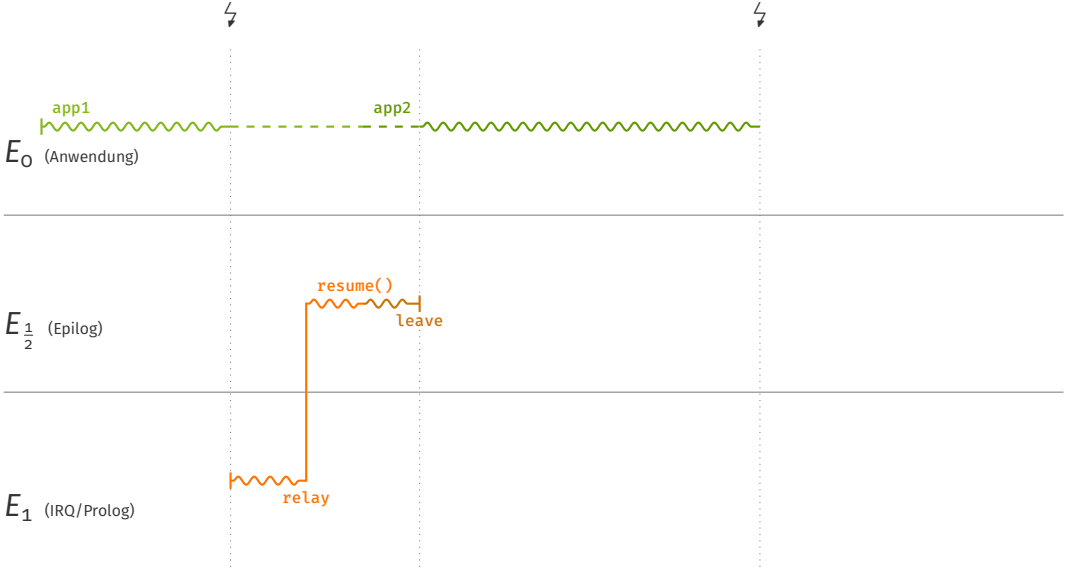
Ablaufbeispiel (Standardfall)



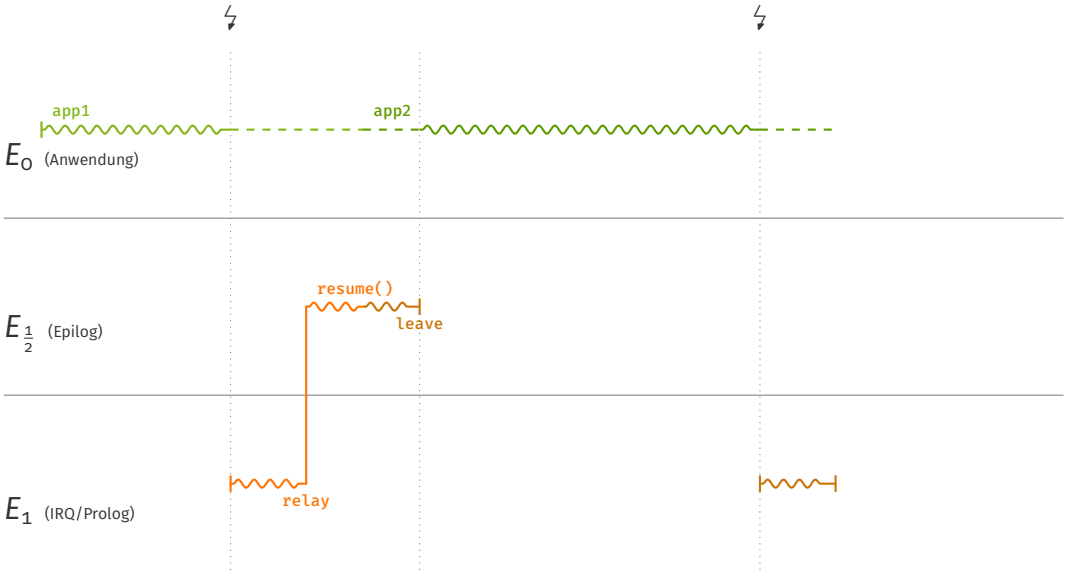
Ablaufbeispiel (Standardfall)



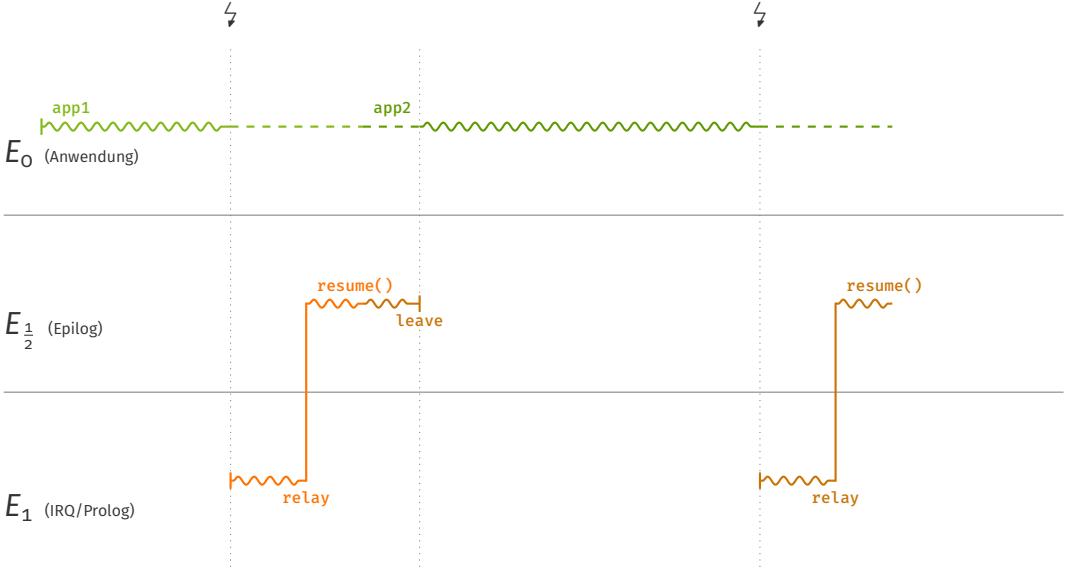
Ablaufbeispiel (Standardfall)



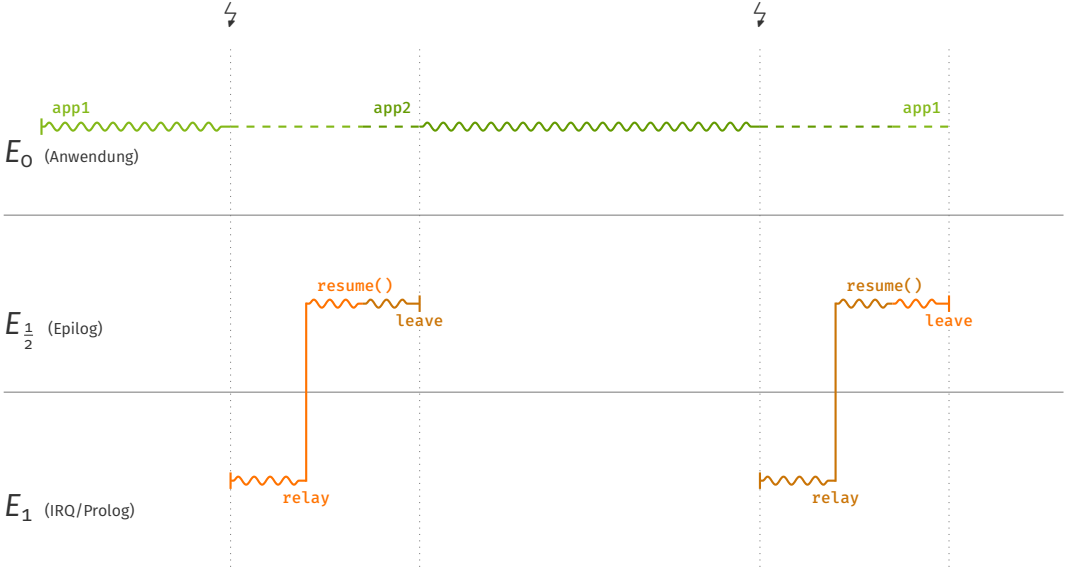
Ablaufbeispiel (Standardfall)



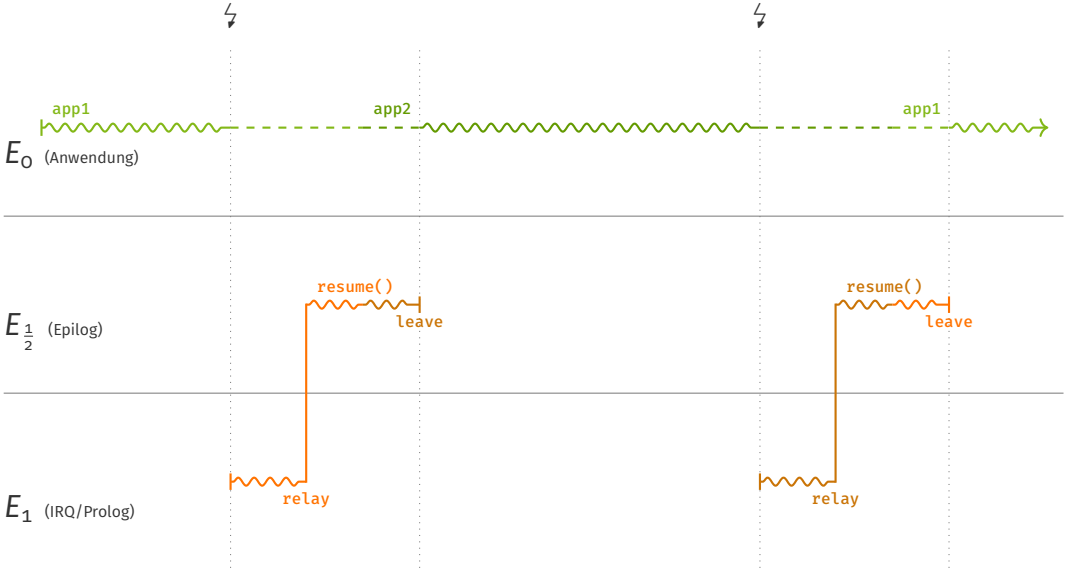
Ablaufbeispiel (Standardfall)



Ablaufbeispiel (Standardfall)



Ablaufbeispiel (Standardfall)



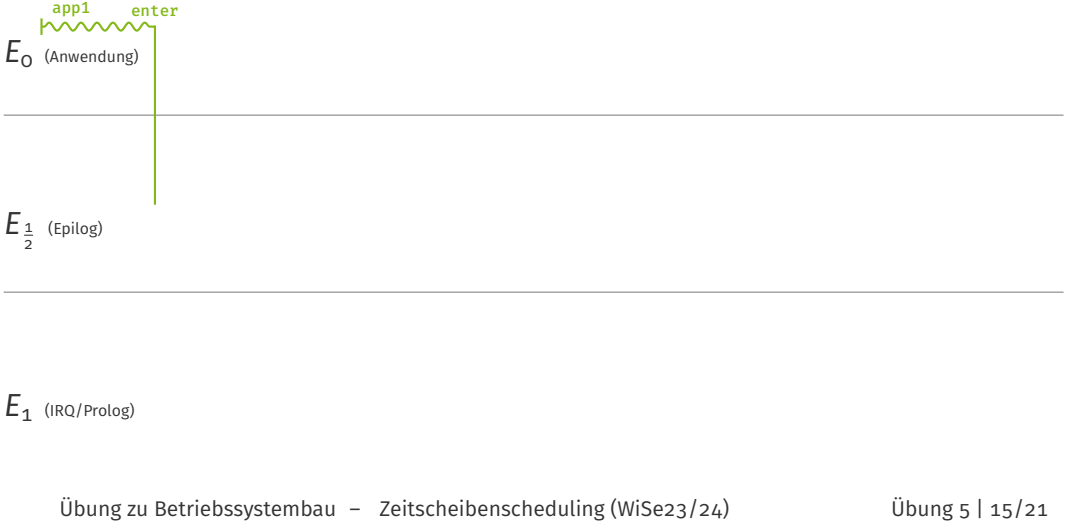
Ablaufbeispiel bei Faden in Systemebene

E_0 (Anwendung)

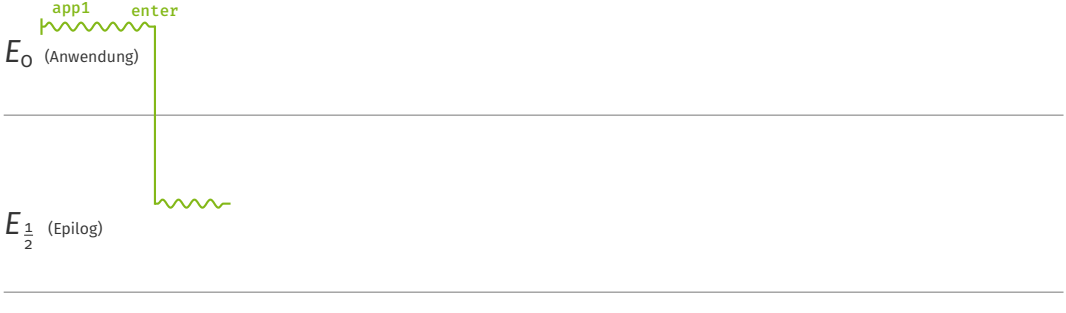
$E_{\frac{1}{2}}$ (Epilog)

E_1 (IRQ/Prolog)

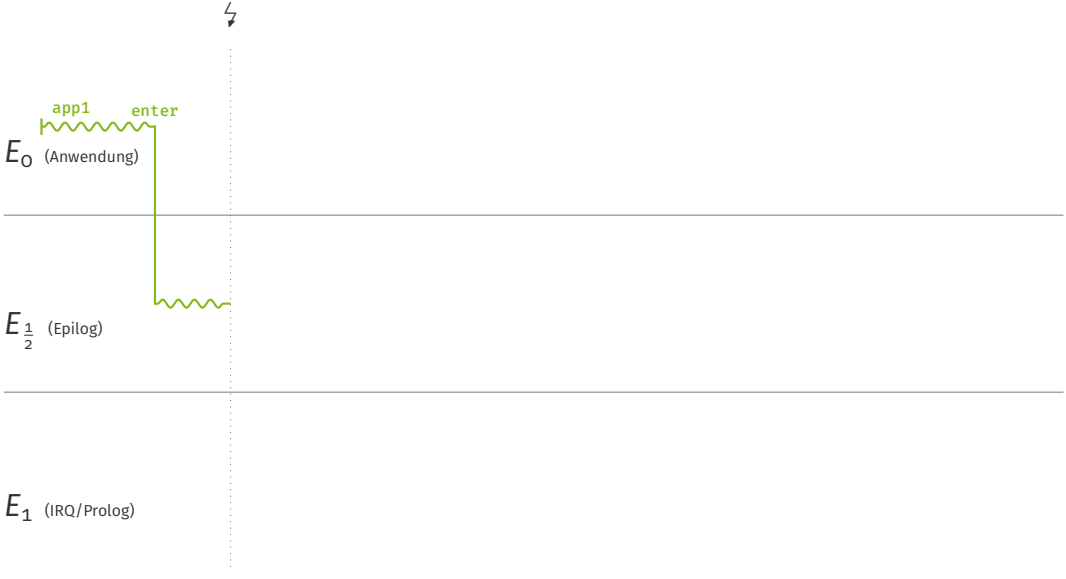
Ablaufbeispiel bei Faden in Systemebene



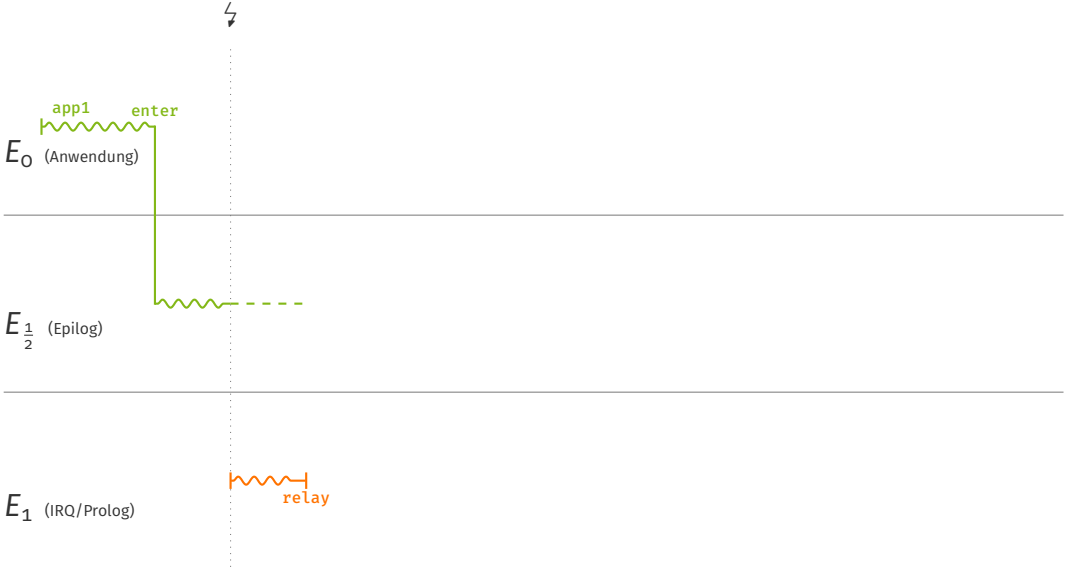
Ablaufbeispiel bei Faden in Systemebene



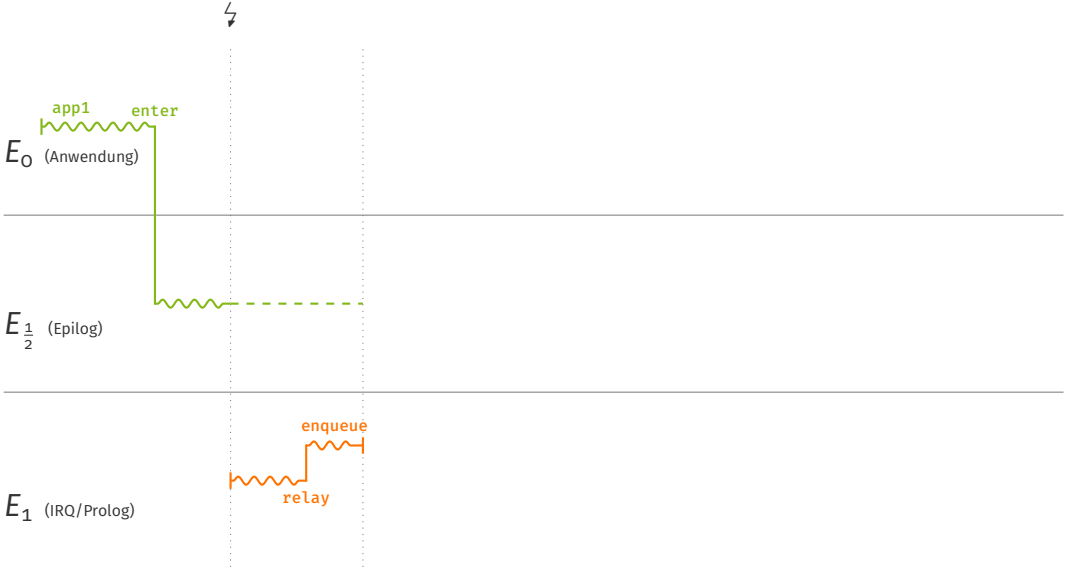
Ablaufbeispiel bei Faden in Systemebene



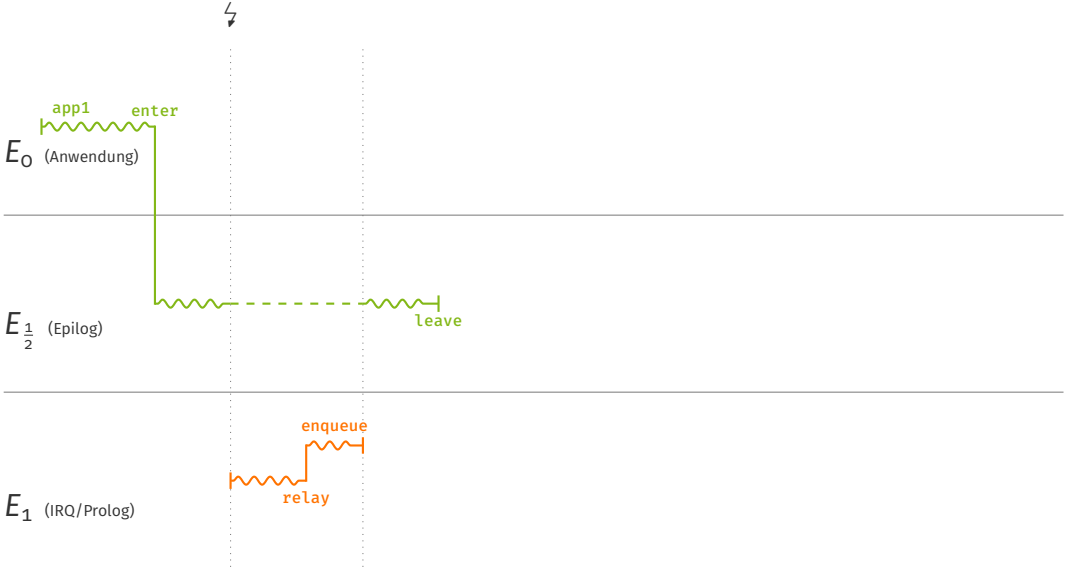
Ablaufbeispiel bei Faden in Systemebene



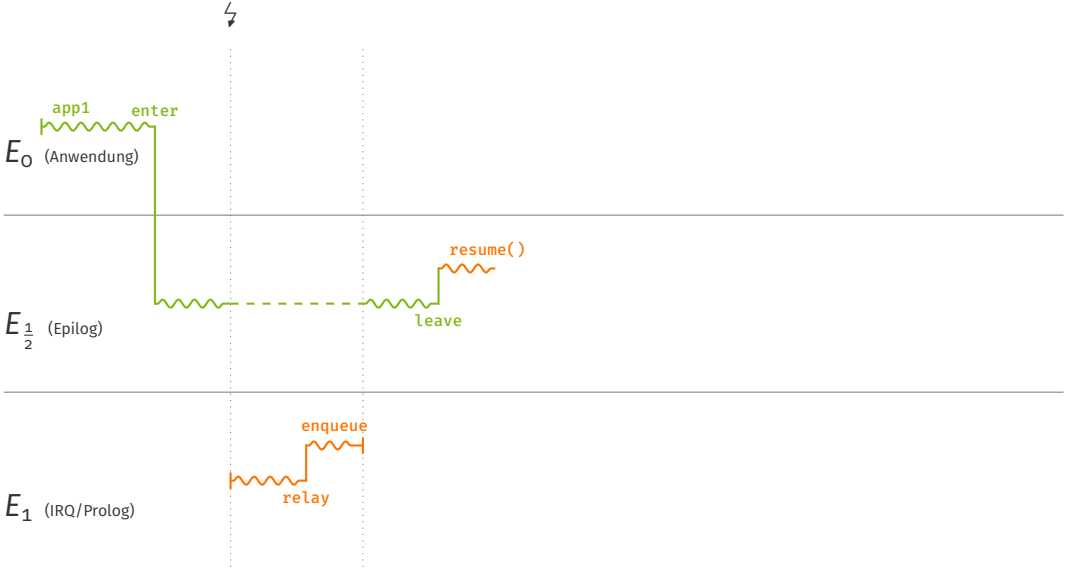
Ablaufbeispiel bei Faden in Systemebene



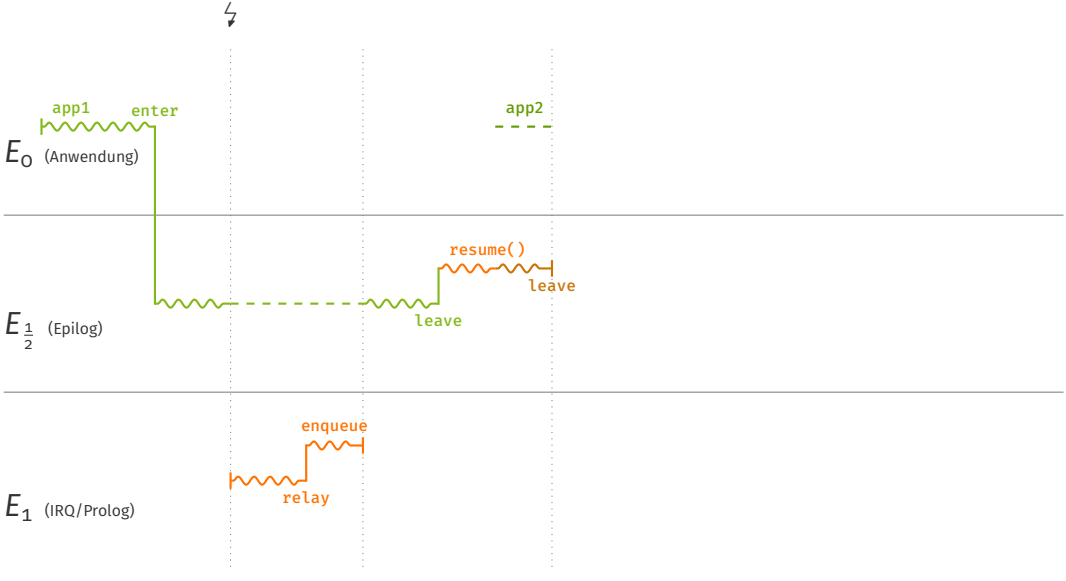
Ablaufbeispiel bei Faden in Systemebene



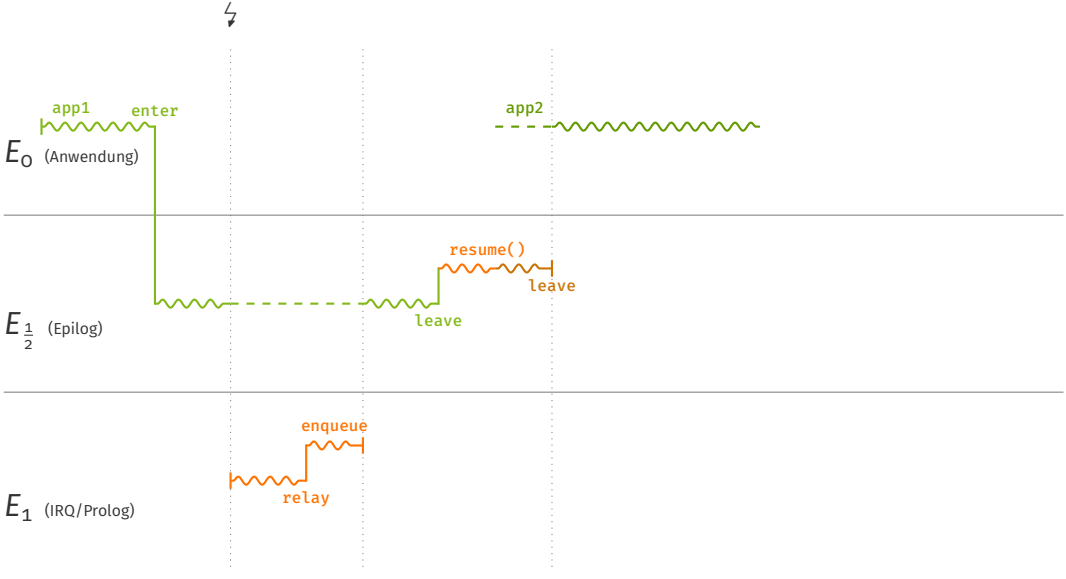
Ablaufbeispiel bei Faden in Systemebene



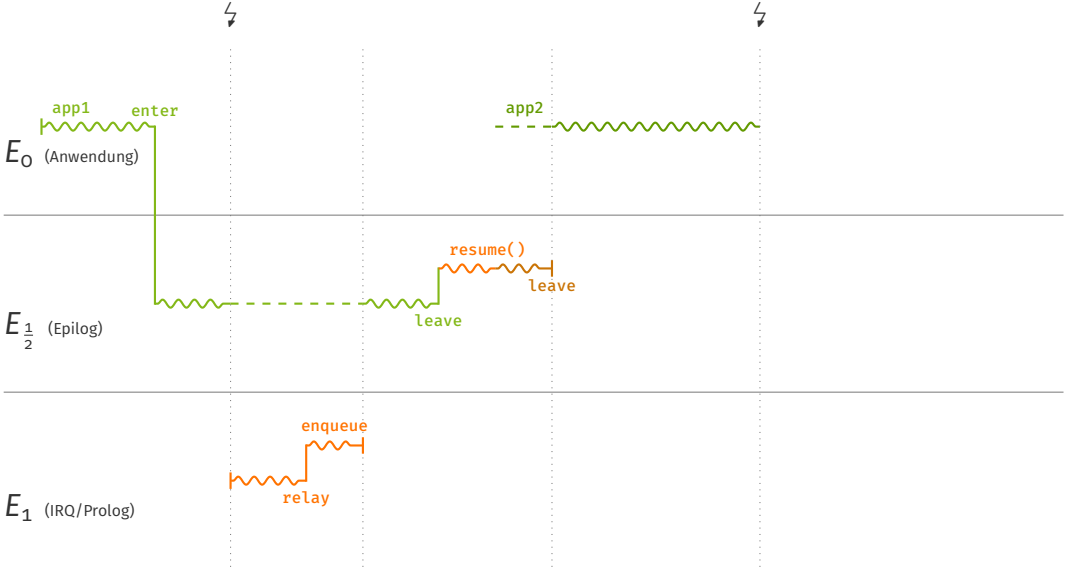
Ablaufbeispiel bei Faden in Systemebene



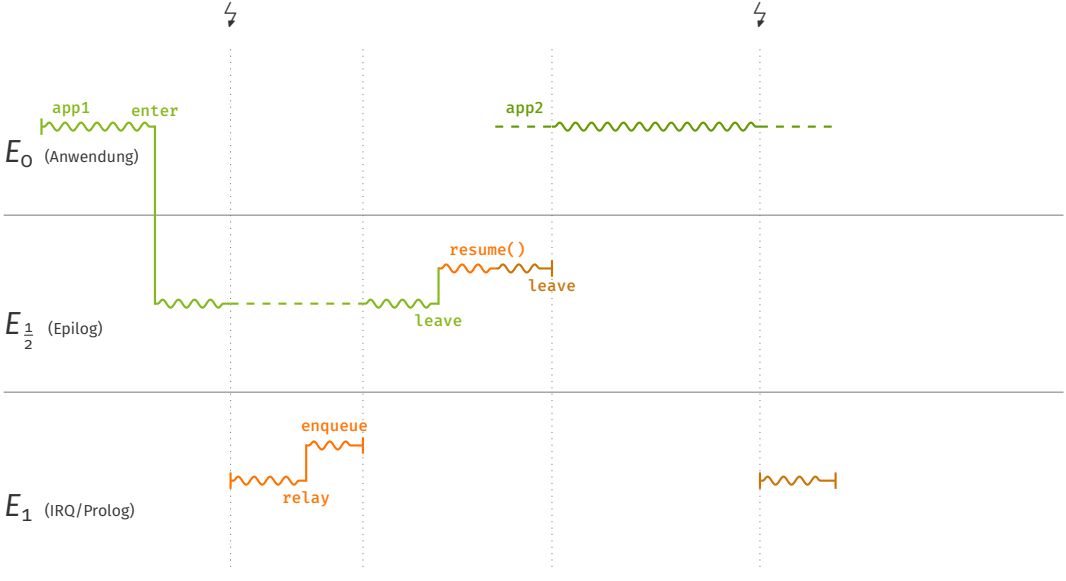
Ablaufbeispiel bei Faden in Systemebene



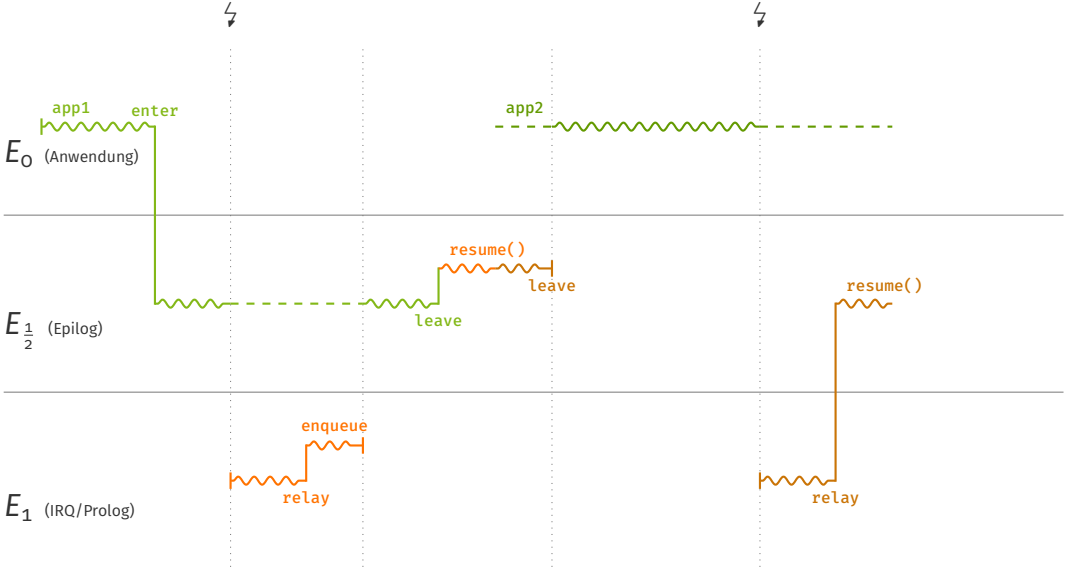
Ablaufbeispiel bei Faden in Systemebene



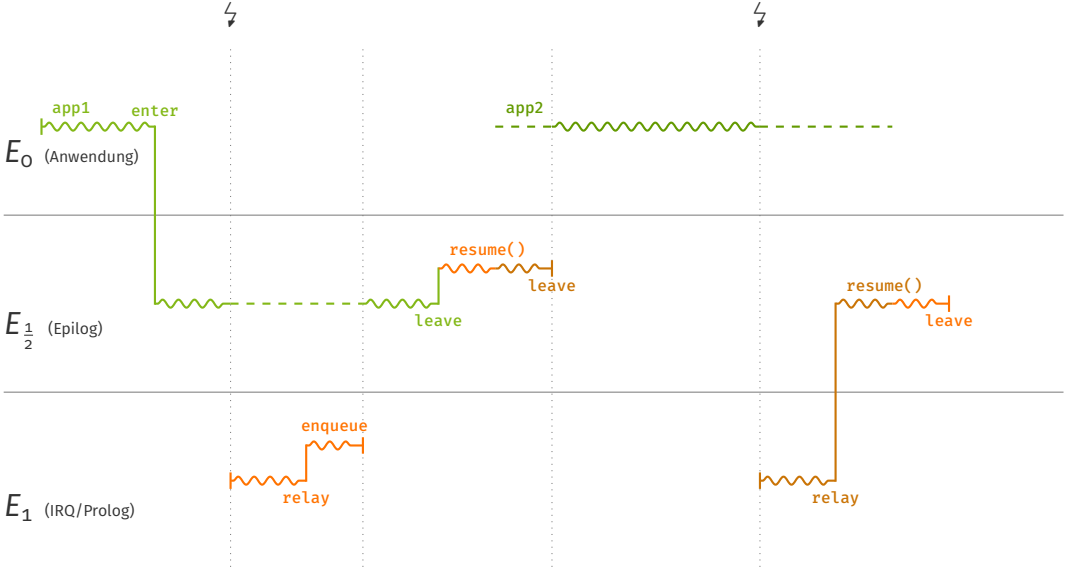
Ablaufbeispiel bei Faden in Systemebene



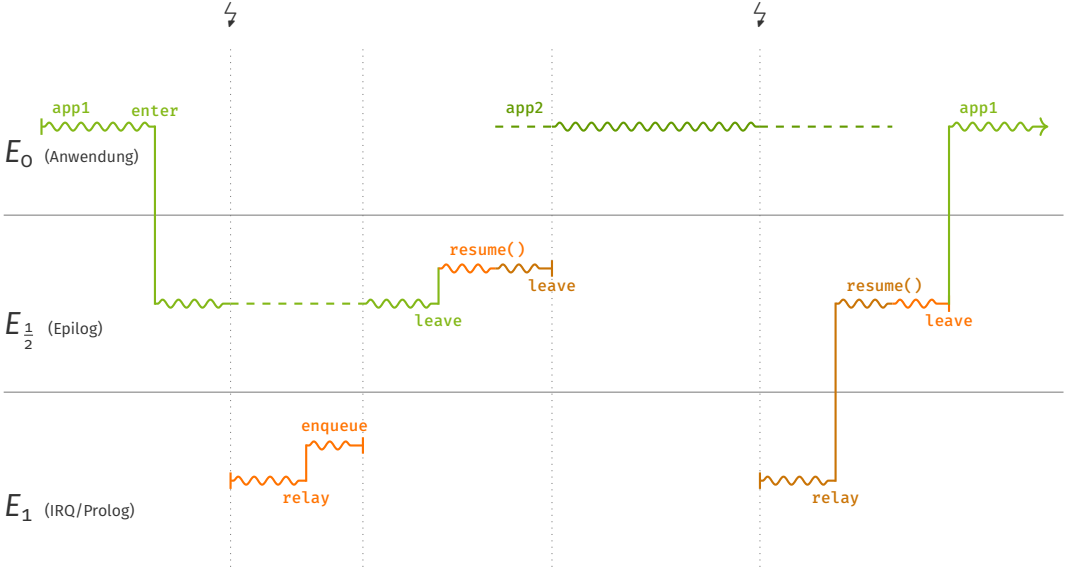
Ablaufbeispiel bei Faden in Systemebene



Ablaufbeispiel bei Faden in Systemebene



Ablaufbeispiel bei Faden in Systemebene



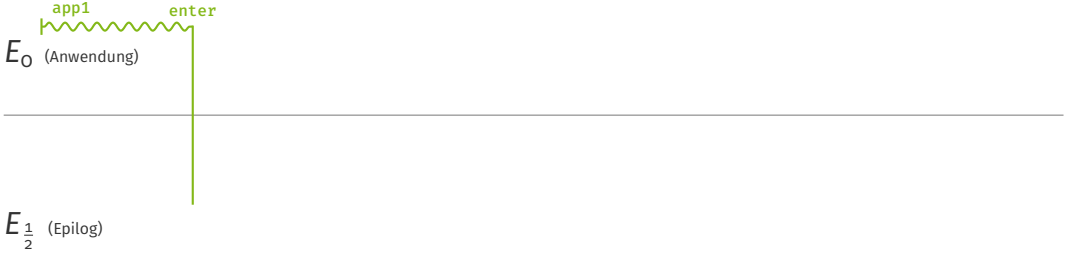
Ablaufbeispiel mit neuem Thread

E_0 (Anwendung)

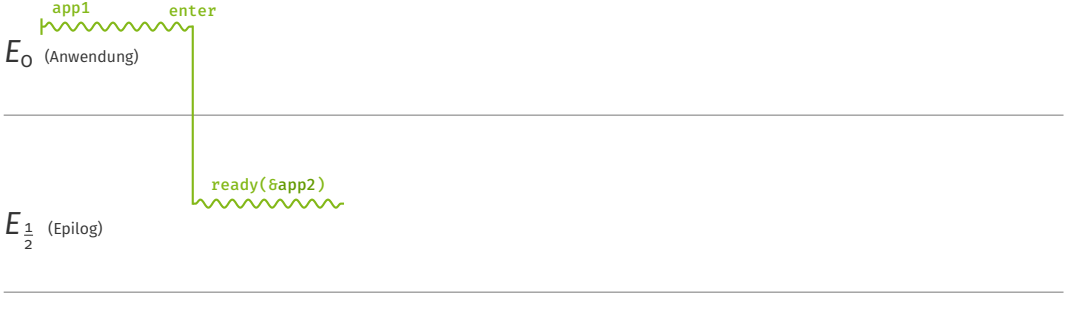
$E_{\frac{1}{2}}$ (Epilog)

E_1 (IRQ/Prolog)

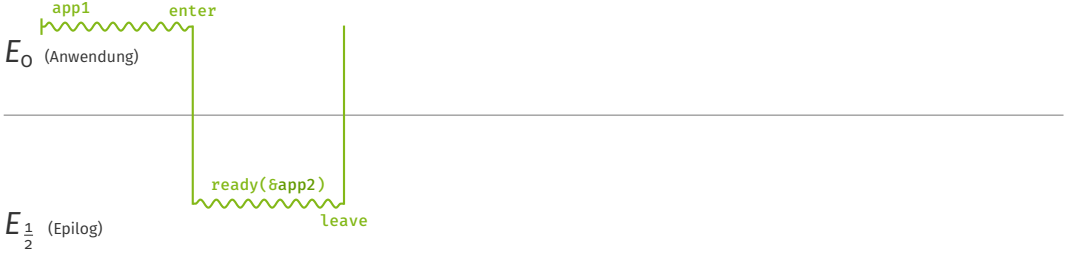
Ablaufbeispiel mit neuem Thread



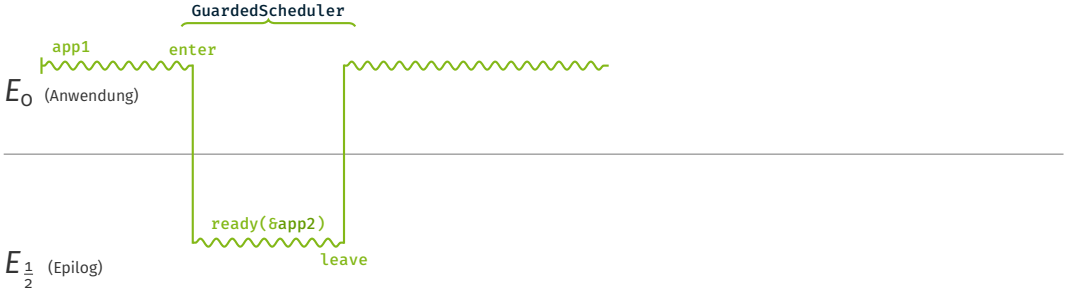
Ablaufbeispiel mit neuem Thread



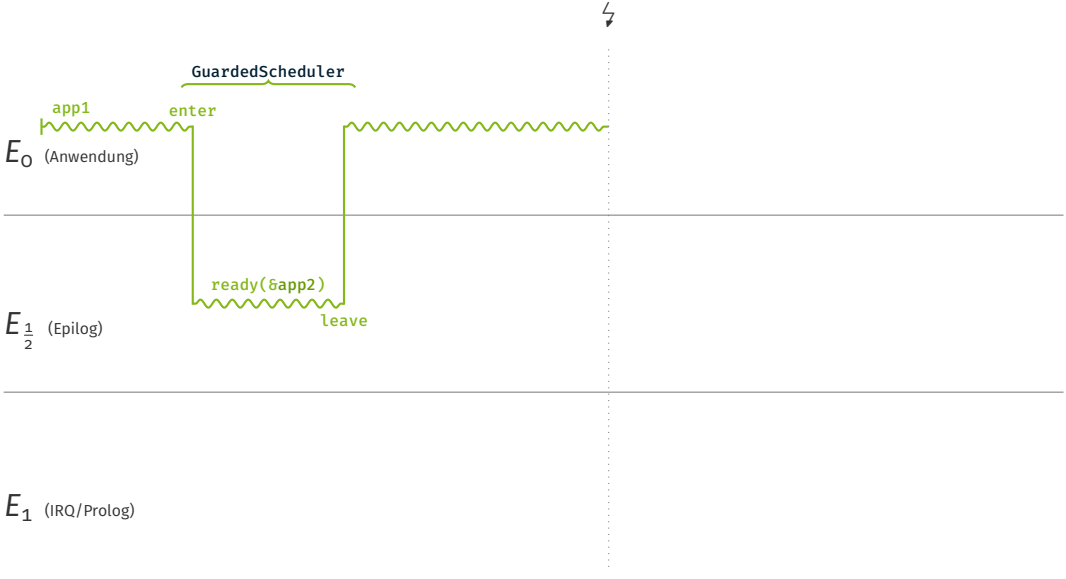
Ablaufbeispiel mit neuem Thread



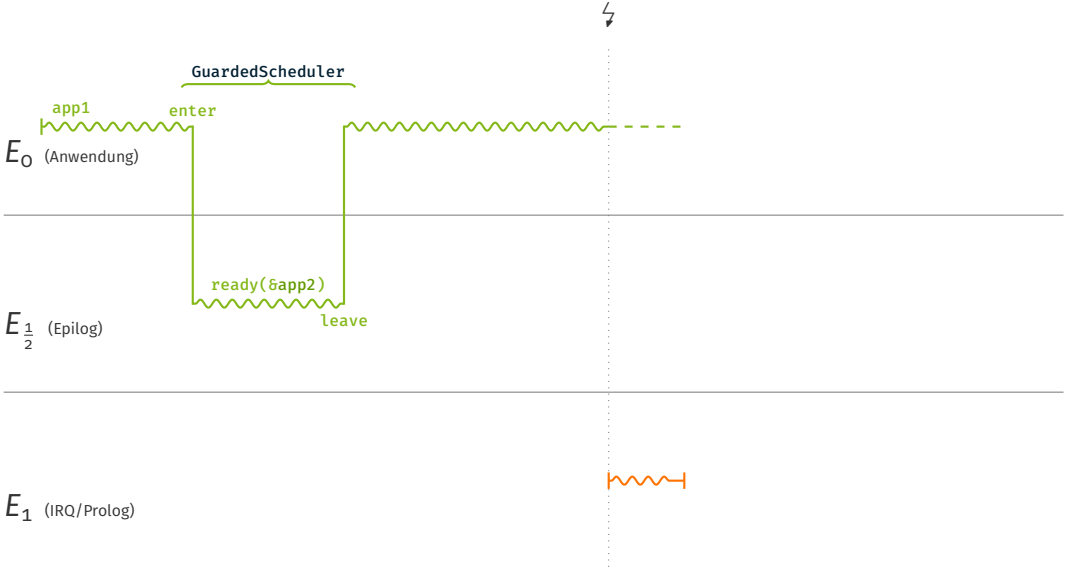
Ablaufbeispiel mit neuem Thread



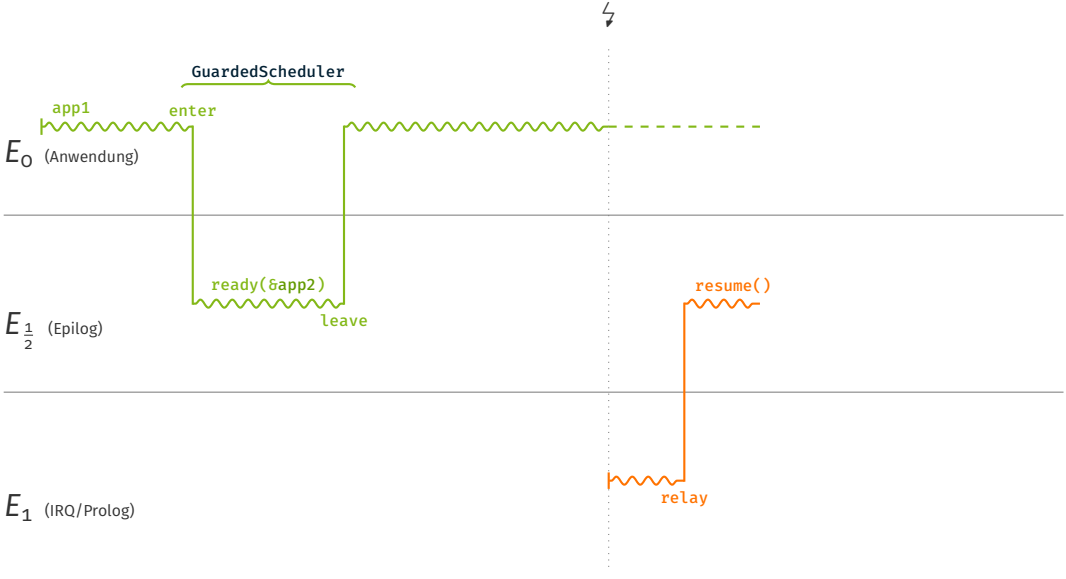
Ablaufbeispiel mit neuem Thread



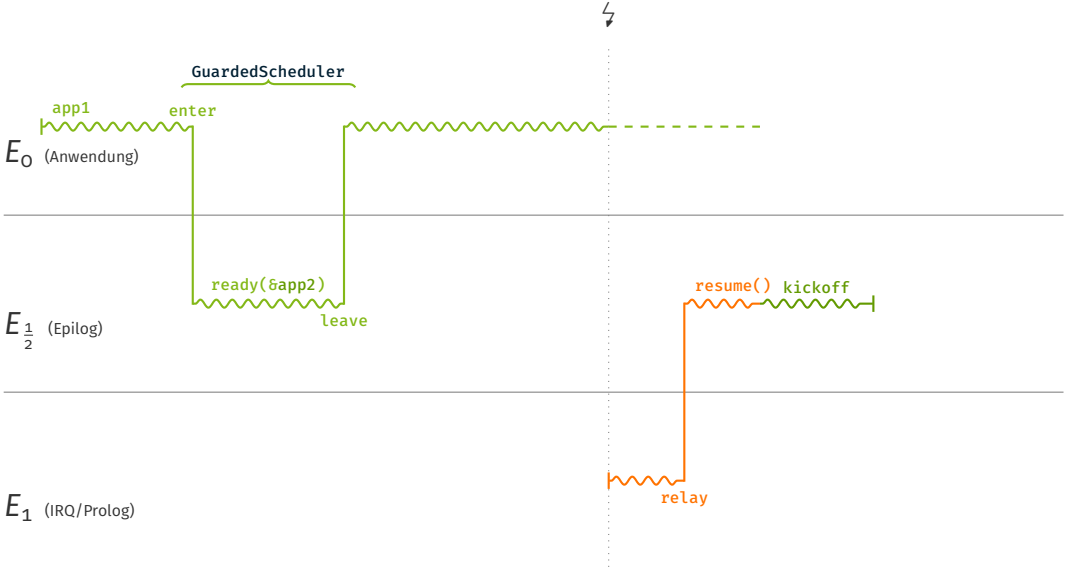
Ablaufbeispiel mit neuem Thread



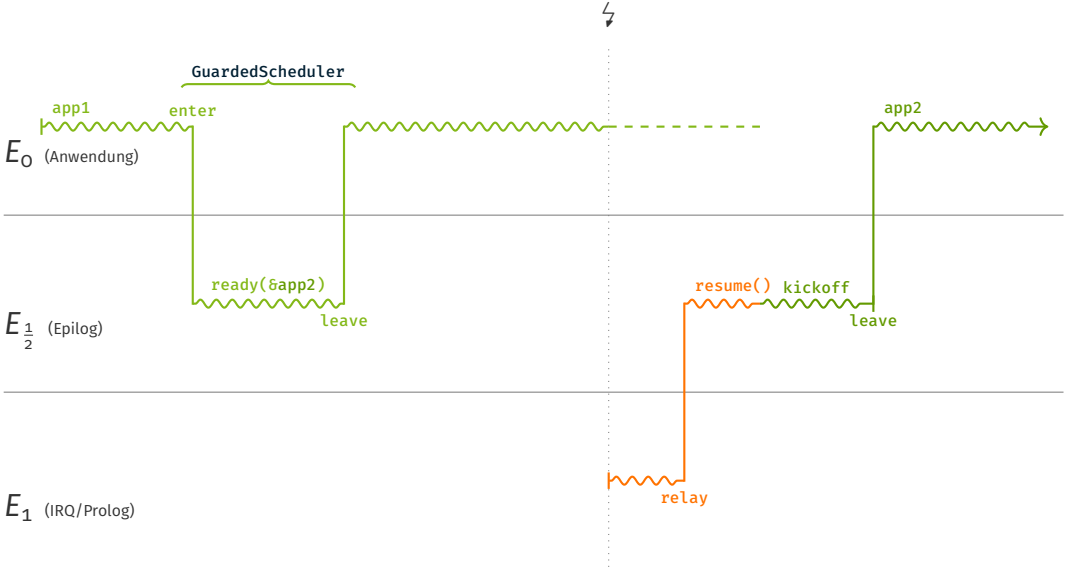
Ablaufbeispiel mit neuem Thread



Ablaufbeispiel mit neuem Thread



Ablaufbeispiel mit neuem Thread



Besonderheiten

Anwendungsfaden beenden

Präemptives Beenden mittels `Scheduler::kill(Thread&)`

Anwendungsfaden beenden

Präemptives Beenden mittels `Scheduler::kill(Thread&)`

OOStuBS *keine Änderung*

Anwendungsfaden beenden

Präemptives Beenden mittels `Scheduler::kill(Thread&)`

OOStuBS *keine Änderung:* aus der Ready-Liste entfernen
bzw. Kill-Flag setzen und bei `resume` prüfen

Anwendungsfaden beenden

Präemptives Beenden mittels `Scheduler::kill(Thread&)`

OOStuBS *keine Änderung:* aus der Ready-Liste entfernen
bzw. Kill-Flag setzen und bei `resume` prüfen

MPStuBS der Anwendungsfaden kann gerade auf einer anderen CPU laufen

Anwendungsfaden beenden

Präemptives Beenden mittels `Scheduler::kill(Thread&)`

OOStuBS *keine Änderung:* aus der Ready-Liste entfernen
bzw. Kill-Flag setzen und bei `resume` prüfen

MPStuBS der Anwendungsfaden kann gerade auf einer anderen CPU laufen

- Kill-Flag setzen (wie gehabt)

Anwendungsfaden beenden

Präemptives Beenden mittels `Scheduler::kill(Thread&)`

OOStuBS *keine Änderung:* aus der Ready-Liste entfernen
bzw. Kill-Flag setzen und bei `resume` prüfen

MPStuBS der Anwendungsfaden kann gerade auf einer anderen CPU laufen

- Kill-Flag setzen (wie gehabt)
- falls er nicht in der Ready-Liste ist, läuft er wohl gerade auf einer anderen CPU

Anwendungsfaden beenden

Präemptives Beenden mittels `Scheduler::kill(Thread&)`

OOStuBS *keine Änderung:* aus der Ready-Liste entfernen
bzw. Kill-Flag setzen und bei `resume` prüfen

MPStuBS der Anwendungsfaden kann gerade auf einer anderen CPU laufen

- Kill-Flag setzen (wie gehabt)
- falls er nicht in der Ready-Liste ist, läuft er wohl gerade auf einer anderen CPU
- diese andere CPU muss benachrichtigt werden

Anwendungsfaden beenden

Präemptives Beenden mittels `Scheduler::kill(Thread&)`

OOStuBS *keine Änderung:* aus der Ready-Liste entfernen
bzw. Kill-Flag setzen und bei `resume` prüfen

MPStuBS der Anwendungsfaden kann gerade auf einer anderen CPU laufen

- Kill-Flag setzen (wie gehabt)
- falls er nicht in der Ready-Liste ist, läuft er wohl gerade auf einer anderen CPU
- diese andere CPU muss benachrichtigt werden
  → Inter Processor Interrupt (IPI)

Anwendungsfaden beenden

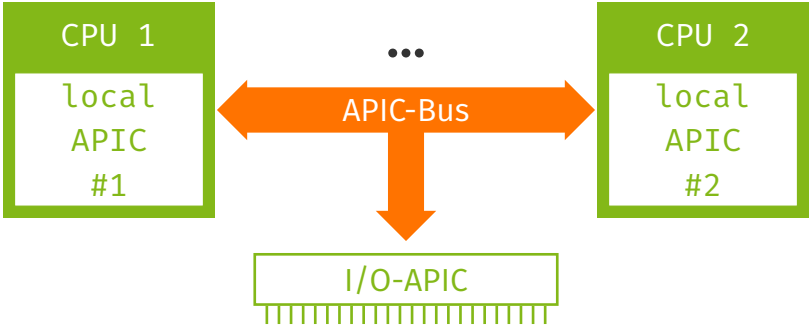
Präemptives Beenden mittels `Scheduler::kill(Thread&)`

OOStuBS *keine Änderung:* aus der Ready-Liste entfernen
bzw. Kill-Flag setzen und bei `resume` prüfen

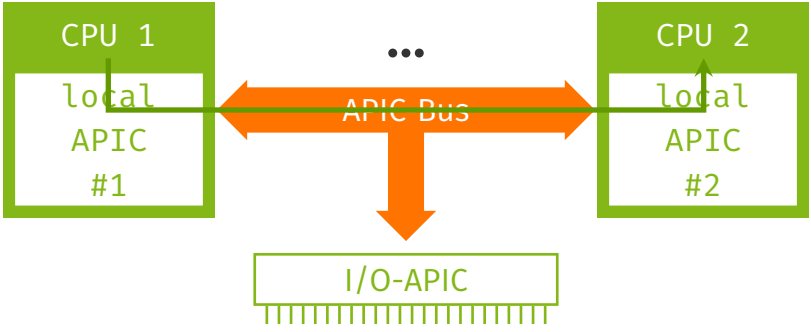
MPStuBS der Anwendungsfaden kann gerade auf einer anderen CPU laufen

- Kill-Flag setzen (wie gehabt)
- falls er nicht in der Ready-Liste ist, läuft er wohl gerade auf einer anderen CPU
- diese andere CPU muss benachrichtigt werden
  → Inter Processor Interrupt (IPI)
- die angesprochene CPU muss dann das Kill-Flag des aktuellen Prozesses prüfen

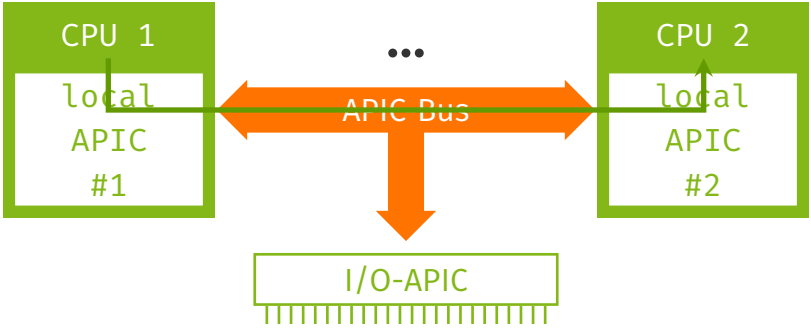
Inter Processor Interrupt



Inter Processor Interrupt

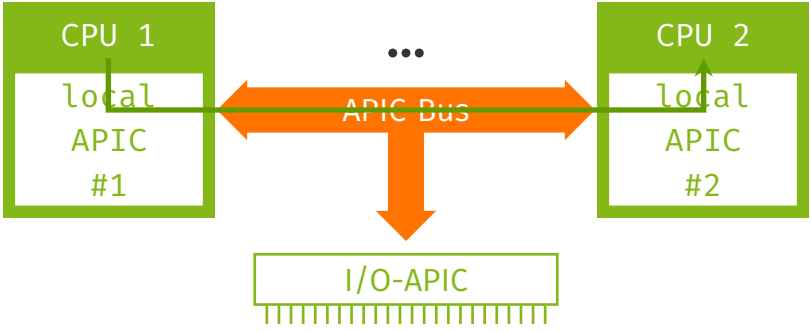


Inter Processor Interrupt




```
LAPIC::IPI::send(destination, vector);
```

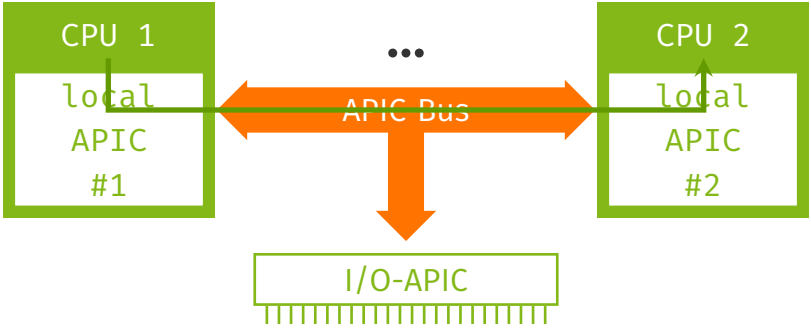
Inter Processor Interrupt




```
LAPIC::IPI::send(destination, vector);
```

Interrupt

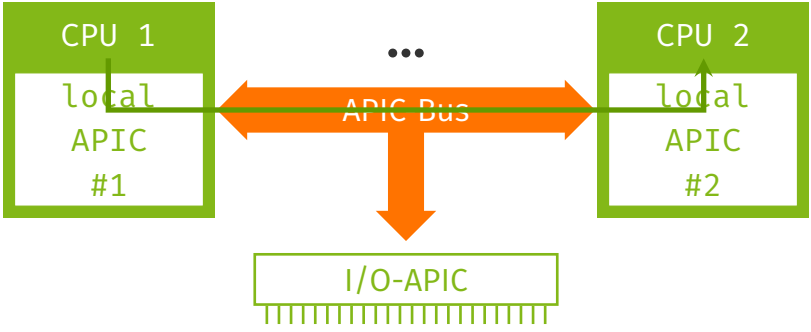
Inter Processor Interrupt




```
LAPIC::IPI::send(destination, vector);
```

Empfänger — Interrupt

Inter Processor Interrupt




```
destination = APIC::getLAPICID(cpu);

LAPIC::IPI::send(destination, vector);
```

Empfänger — Interrupt

Der Teufel steckt im Detail

Was kann hier schon schief gehen?

```
01 lock[Core::getID()] = true;
```

Der Teufel steckt im Detail

Was kann hier schon schief gehen?

```
01 lock[Core::getID()] = true;
```

Viel - diese Zeile wird nicht atomar ausgeführt:

```
01 ; Array lock an Adresse 0x2000
02 call <Core::getID()>
03 mov [rax+0x2000], 0x1
```

Der Teufel steckt im Detail

Was kann hier schon schief gehen?

```
01 lock[Core::getID()] = true;
```

Viel - diese Zeile wird nicht atomar ausgeführt:

```
01 ; Array lock an Adresse 0x2000
02 call <Core::getID()>
03 mov [rax+0x2000], 0x1
```

ϟ Scheduler Interrupt

Der Teufel steckt im Detail

Was kann hier schon schief gehen?

```
01 lock[Core::getID()] = true;
```

Viel - diese Zeile wird nicht atomar ausgeführt:

```
01 ; Array lock an Adresse 0x2000
02 call <Core::getID()>
03 mov [rax+0x2000], 0x1
```

↯ Scheduler Interrupt

Was passiert nun, wenn der Anwendungsfaden anschließend auf einer anderen CPU eingeplant wird?

Fragen über Fragen

- Kann nun eine fehlerhafte Anwendung unser Betriebssystem blockieren?

Fragen über Fragen

- Kann nun eine fehlerhafte Anwendung unser Betriebssystem blockieren?
- Unter welchen Umständen wäre es effizienter, den Timer abzuschalten (Stichwort *tickless*)?

Fragen über Fragen

- Kann nun eine fehlerhafte Anwendung unser Betriebssystem blockieren?
- Unter welchen Umständen wäre es effizienter, den Timer abzuschalten (Stichwort *tickless*)?

Gibt es noch Fragen?