ISE SRA

Leibniz Universität Hannover

Anwendungs- und Hardwaregewahrer Betriebssystementwurf

Gerion Entrup, Christian Dietrich, Daniel Lohmann

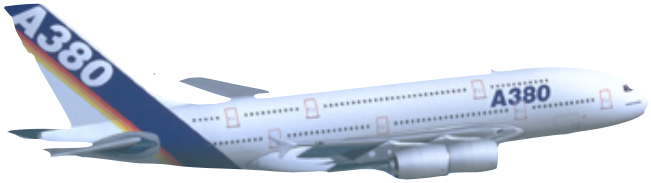
{entrup,dietrich,lohmann}@sra.uni-hannover.de

Leibniz Universität Hannover

7. February 2018

gefördert durch die DFG

Diversität in Hard- und Software



Diversität in Hard- und Software

Möglicher Weg: Spezialisierung

Wie können wir Betriebssysteme effizient spezialisieren?

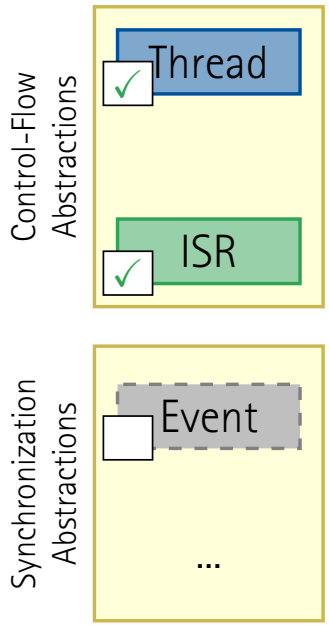
Spezialisierung kann man verschieden weit treiben

```
ISR(I1) {
   isr()
   if (cond)
     ActivateTask(T);
   iret();
}

TASK(T) {
   kickoff()
   computation();
   TerminateTask();
}
```

Spezialisierung kann man verschieden weit treiben

```
ISR(I1) {
   isr()
   if (cond)
     ActivateTask(T);
   iret();
}

TASK(T) {
   kickoff()
   computation();
   TerminateTask();
}
```

Bis zur Ebene der Abstraktionen

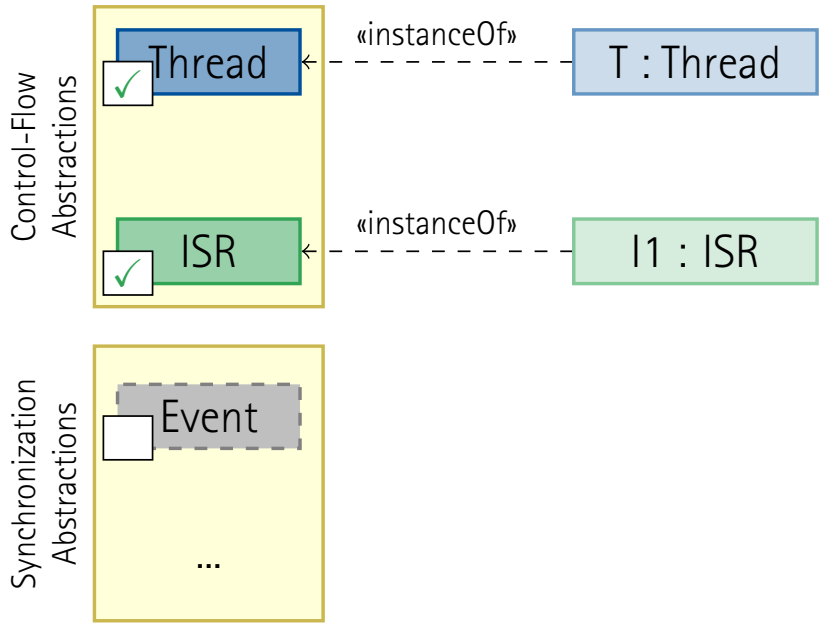


z. B. Linux, eCos, FreeRTOS

Spezialisierung kann man verschieden weit treiben

```
ISR(I1) {
   isr()
   if (cond)
     ActivateTask(T);
   iret();
}

TASK(T) {
   kickoff()
   computation();
   TerminateTask();
}
```

Bis zur Ebene der Instanzen

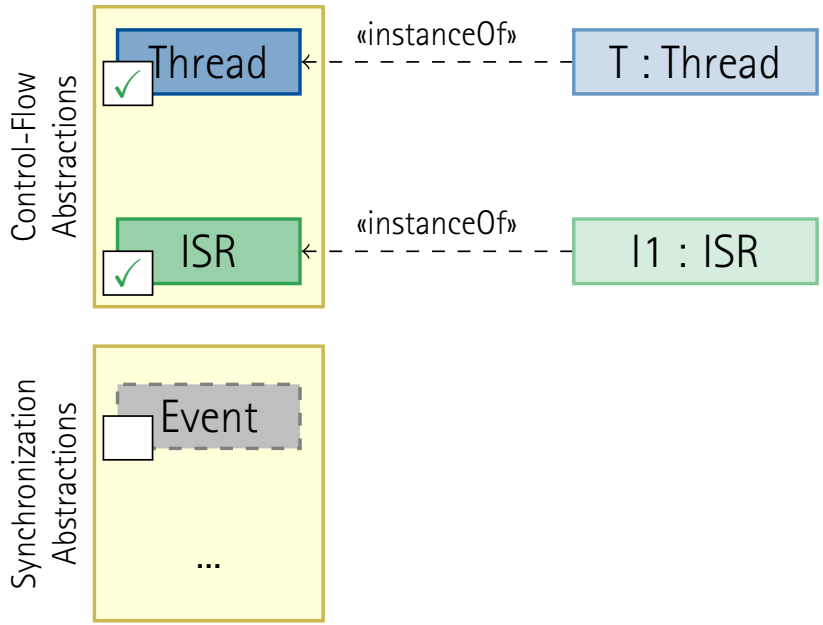


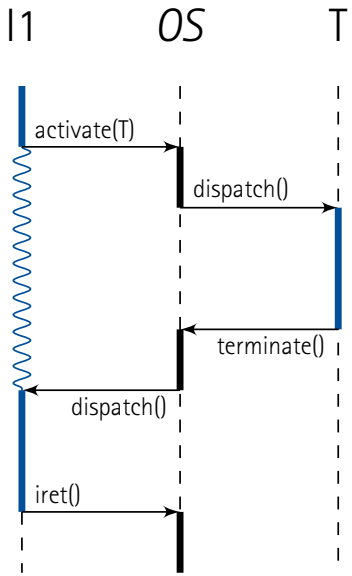
z. B. OSEK/AUTOSAR

Spezialisierung kann man verschieden weit treiben

```
ISR(I1) {
   isr()
   if (cond)
     ActivateTask(T);
   iret();
}

TASK(T) {
   kickoff()
   computation();
   TerminateTask();
}
```

Bis zur Ebene der Interaktionen

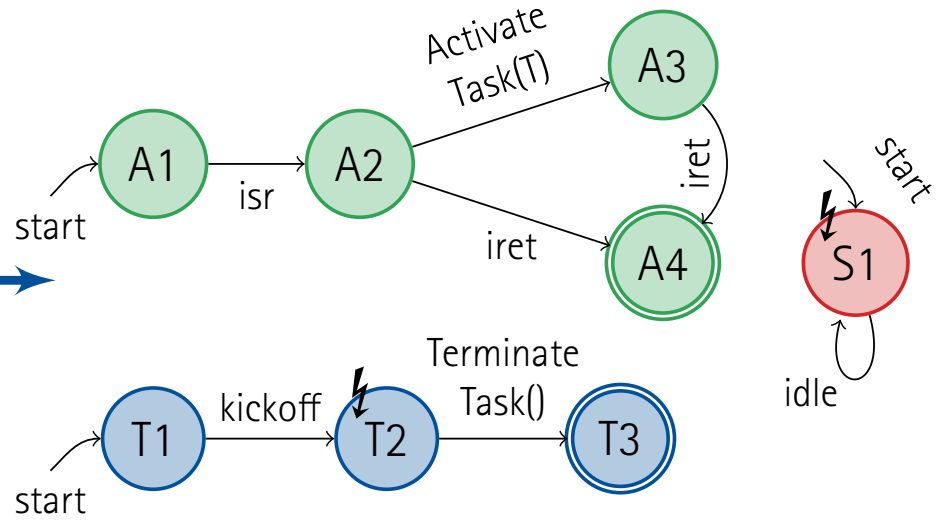




z. B. *d*OSEK

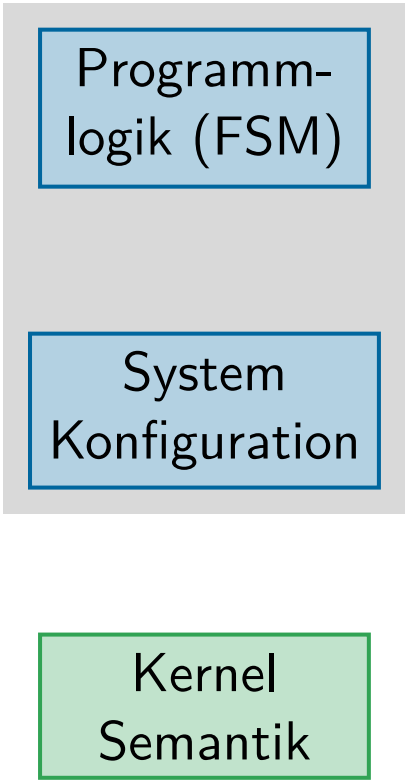
Erzeugung der Applikationsautomaten

```
ISR(I1) {
   isr()
   if (cond)
     ActivateTask(T);
   iret();
}

TASK(T) {
   kickoff()
   computation();
   TerminateTask();
}
```

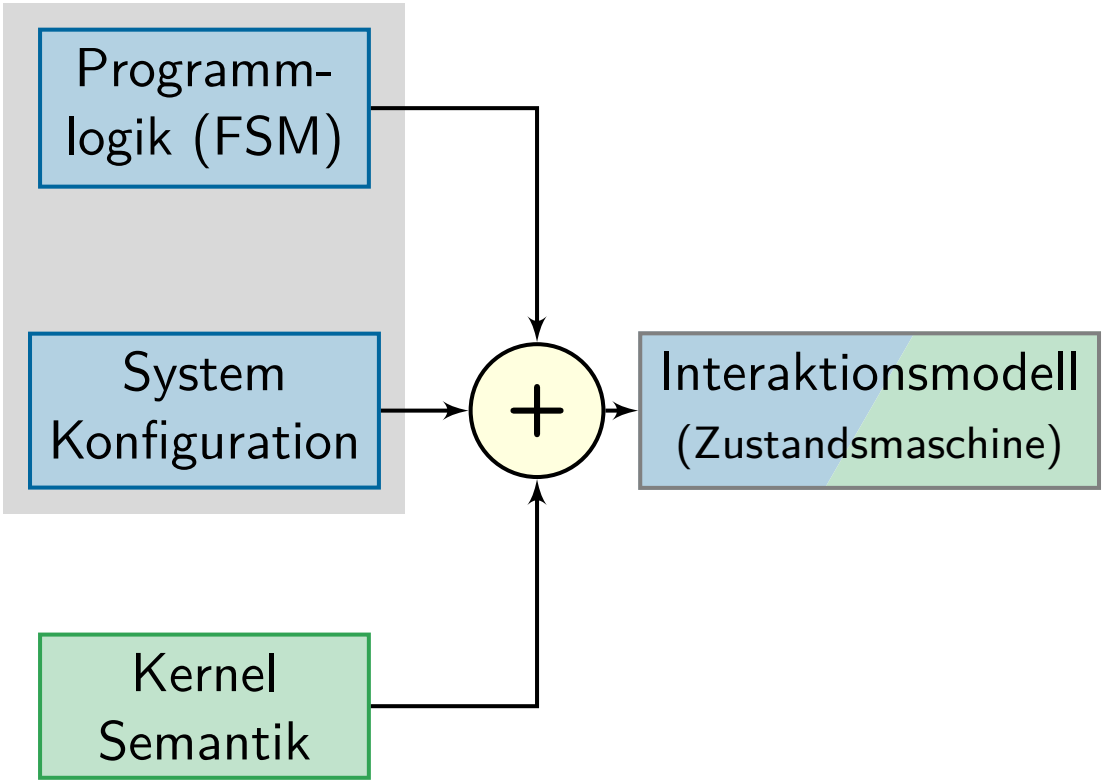


- Statisch konfigurierbares, ereignesgesteuertes Echtzeitsystem
- Konzentration auf Interaktionen mit dem RTOS
- Zustandsübergänge „generieren“ Syscalls an das Betriebssystem

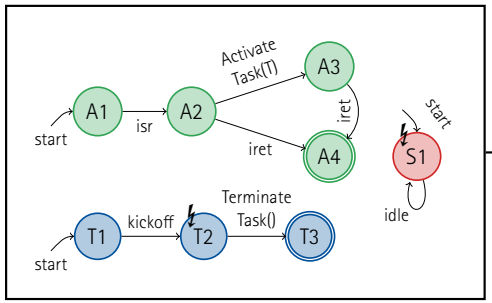
Von der Anwendung zum Modell



Von der Anwendung zum Modell

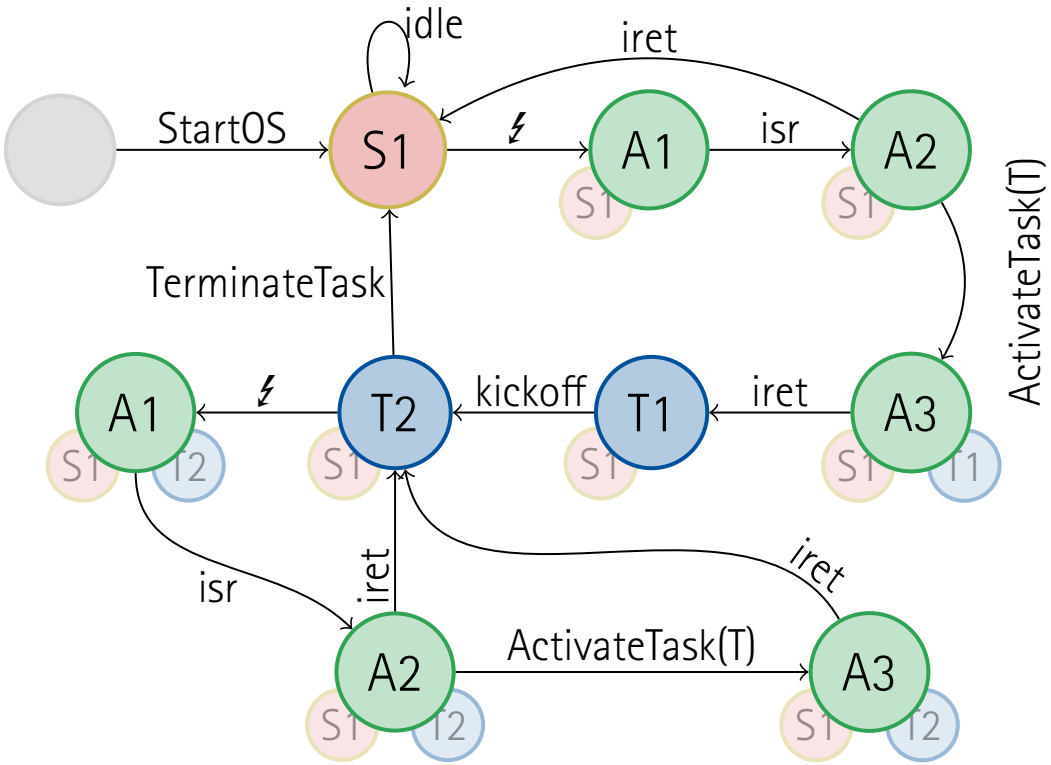


Die System State Machine




```
SYSTEM {
  TASK T {
    PRIO = 4;
    ACTIVATION = 1;
  }
  ISR I1 {
    DEVICE = 35;
  }
}
```

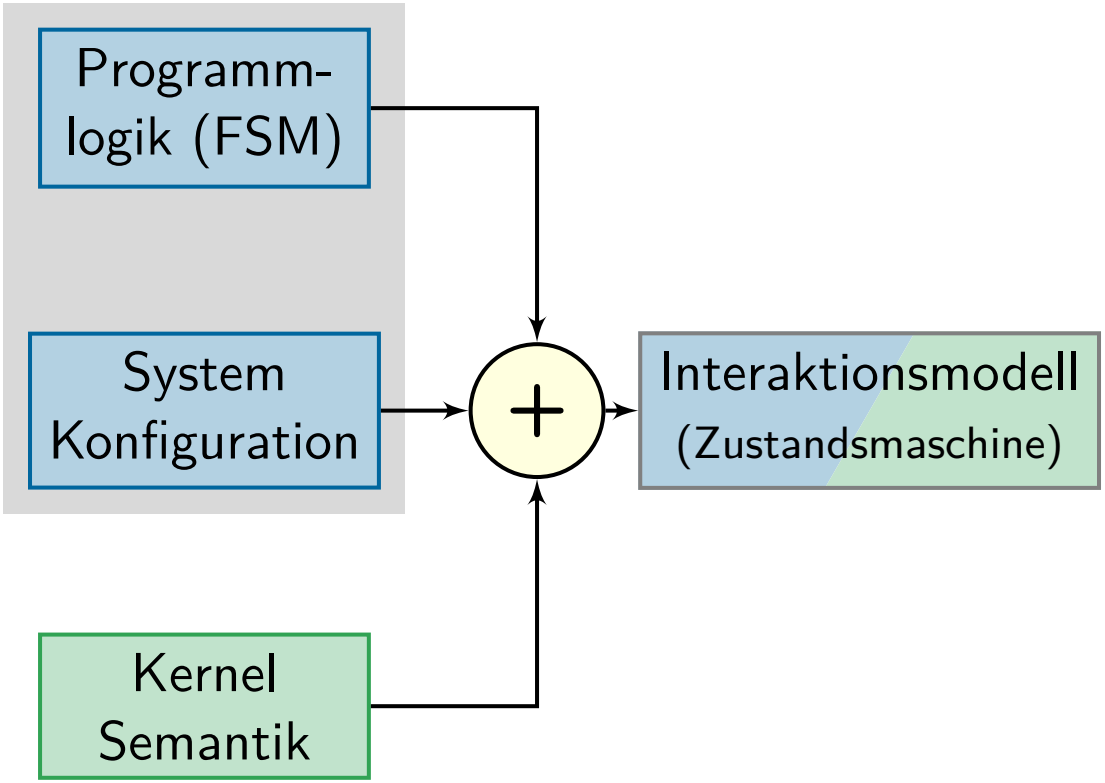
System State Enumeration



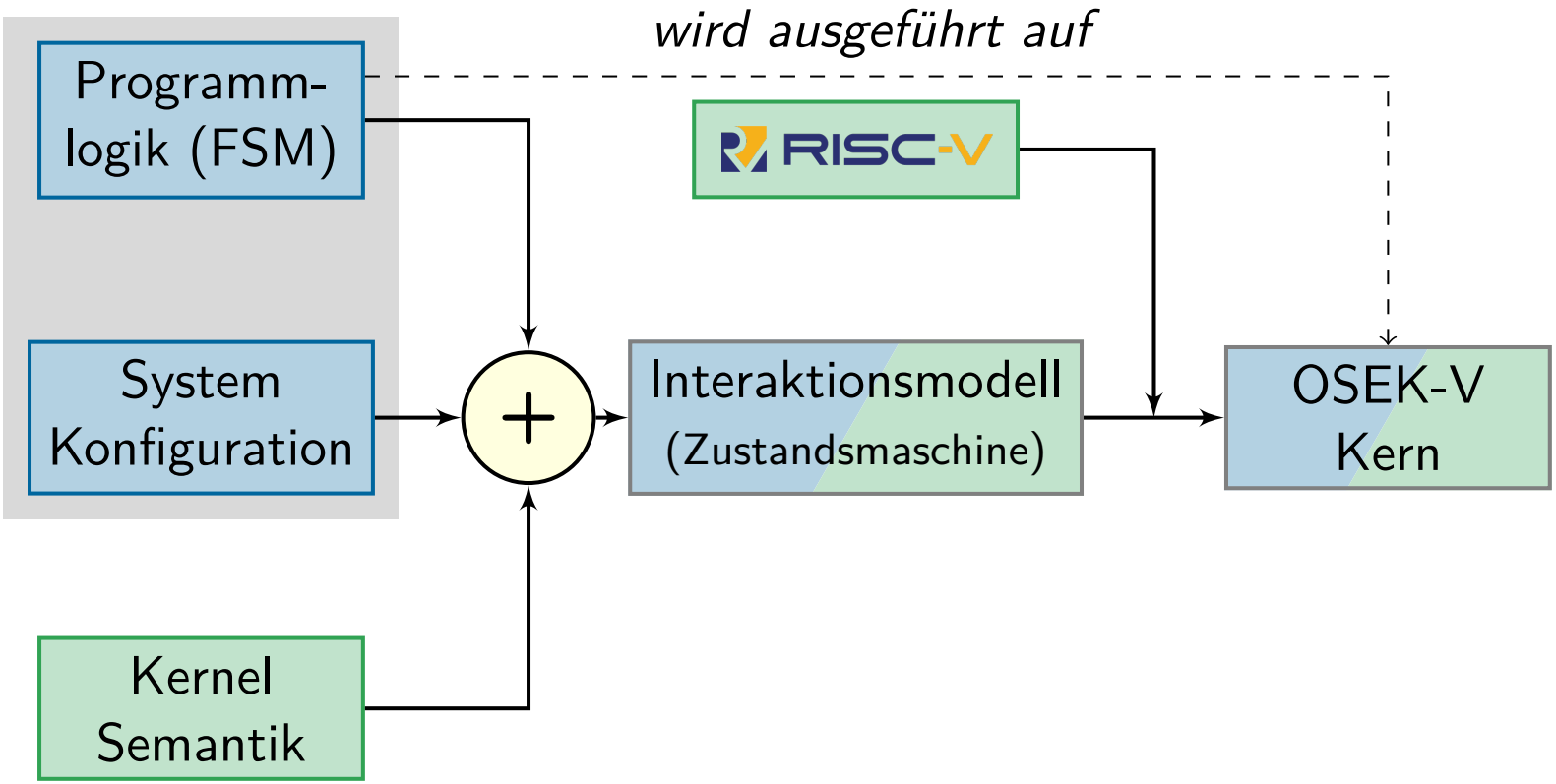
(LCTES'15, TECS'17)

- Kombination von Anwendungslogik, Systemkonfiguration und Kernel Semantik
- In jedem Zustand ist genau ein Kontrollfluss aktiv.

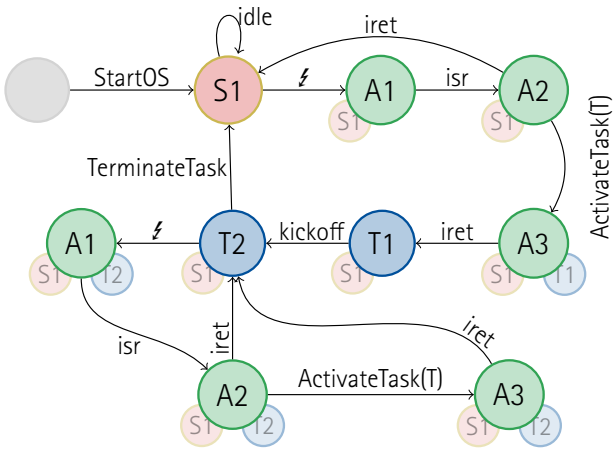
Integration in die Hardware

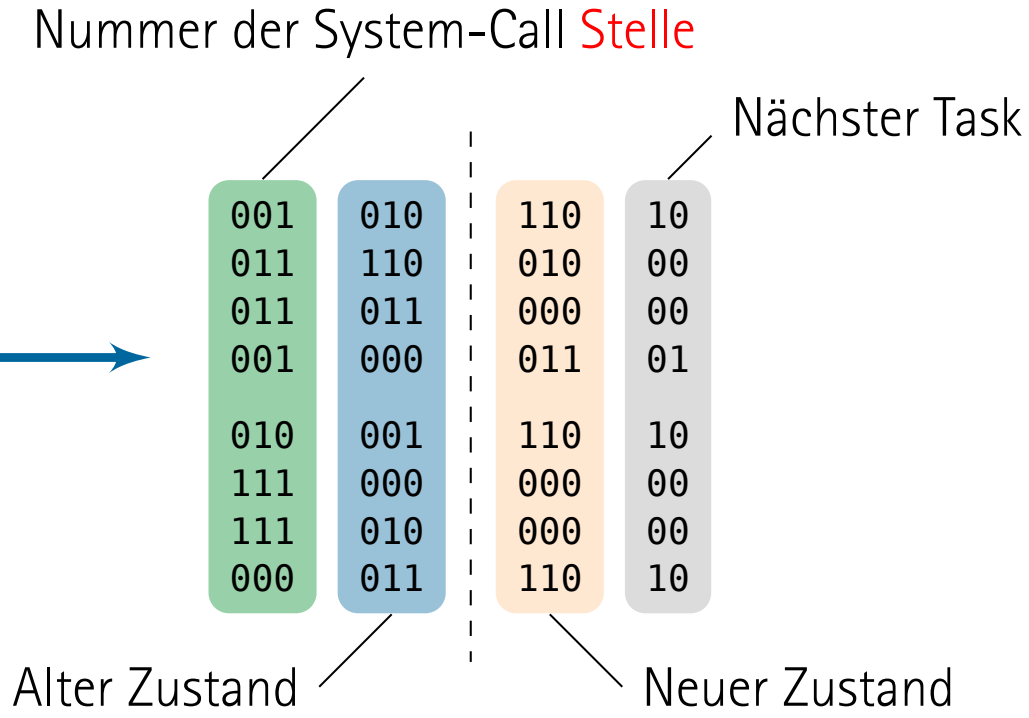


Integration in die Hardware



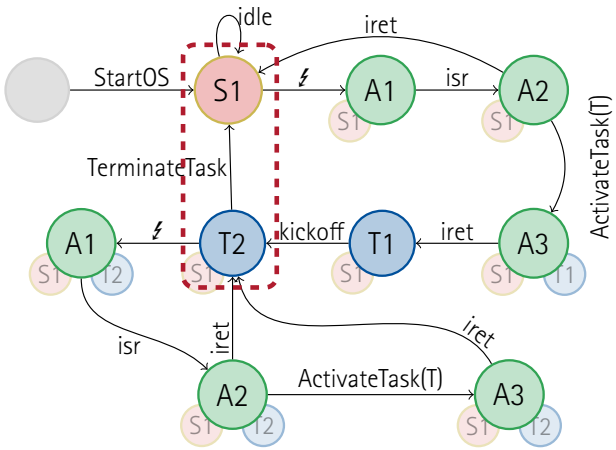
Implementierung des Automaten

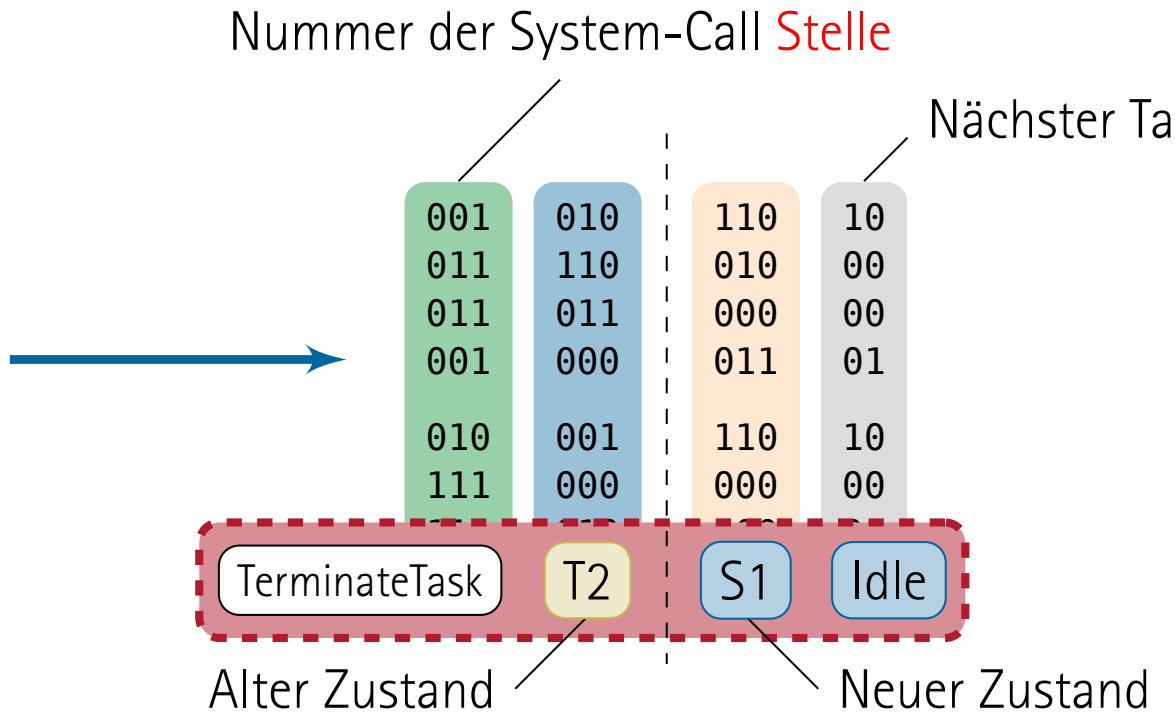




- Automat kann in Tabelle codiert und minimiert werden
- Effizient in Hardware implementierbar

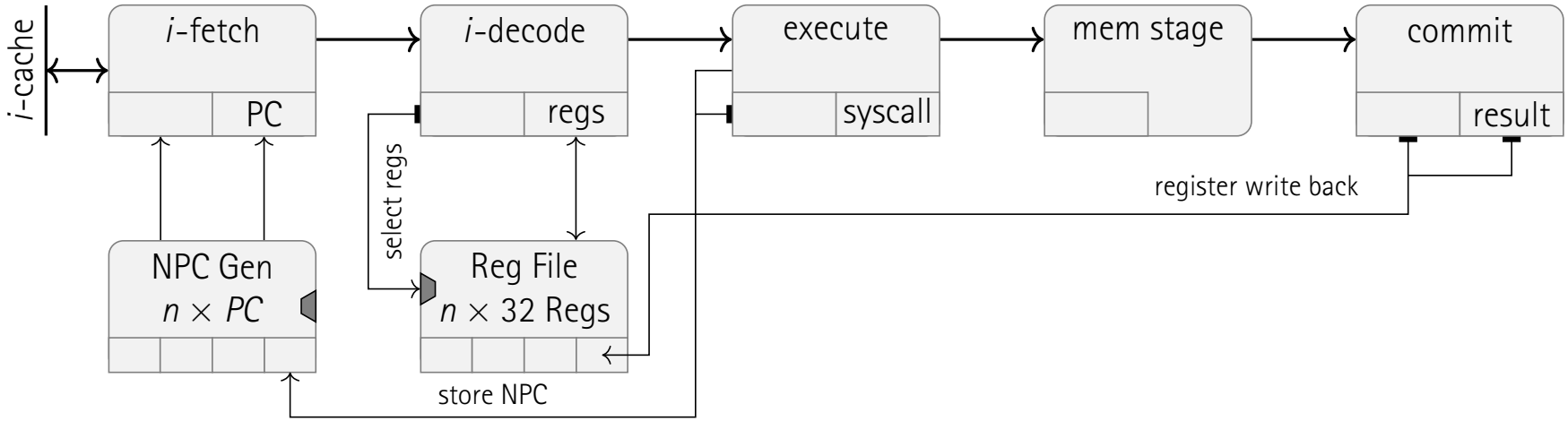
Implementierung des Automaten





- Automat kann in Tabelle codiert und minimiert werden
- Effizient in Hardware implementierbar

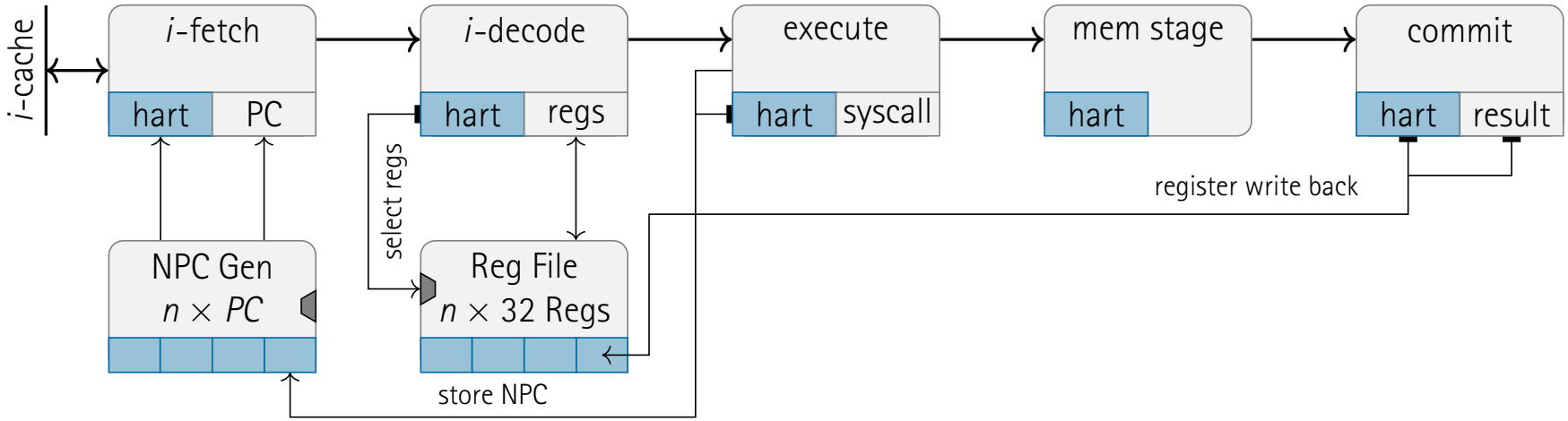
Die OSEK-V Pipeline



RISC-V and the Rocket Core

- Freie ISA für Forschung (seit etwa 2015)
- Rocket ist eine 5-stufige Pipeline

Die OSEK-V Pipeline



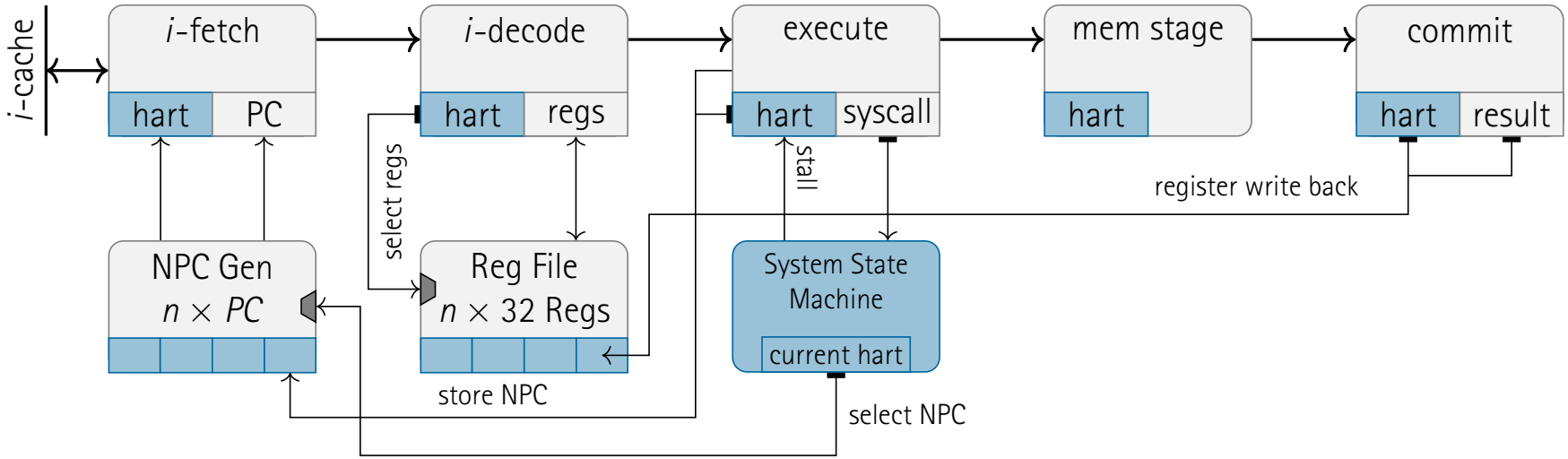
RISC-V and the Rocket Core

- Freie ISA für Forschung (seit etwa 2015)
- Rocket ist eine 5-stufige Pipeline

Erweitert um

- Hardware Threads werden zur Eigenschaft der Pipeline

Die OSEK-V Pipeline



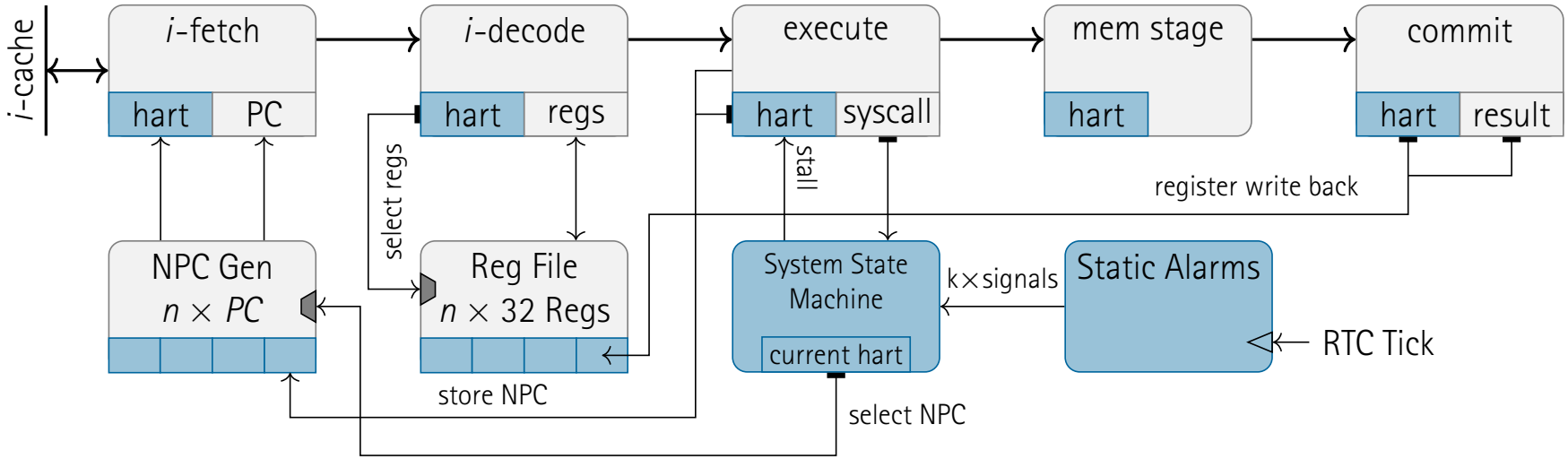
RISC-V and the Rocket Core

- Freie ISA für Forschung (seit etwa 2015)
- Rocket ist eine 5-stufige Pipeline

Erweitert um

- Hardware Threads werden zur Eigenschaft der Pipeline
- Signal-State-Machine implementiert RTOS-Semantik

Die OSEK-V Pipeline



RISC-V and the Rocket Core

- Freie ISA für Forschung (seit etwa 2015)
- Rocket ist eine 5-stufige Pipeline

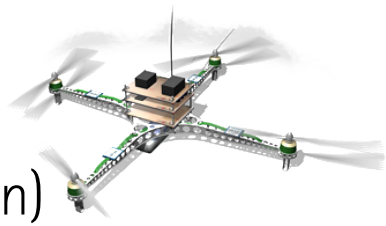
Erweitert um

- Hardware Threads werden zur Eigenschaft der Pipeline
- Signal-State-Machine implementiert RTOS-Semantik
- Optionale Komponente für statische Alarme

Evaluation: Quadrokopter Flugkontroller

*i4*Copter

- Realistisches, sicherheitskritisches Echtzeitsystem
- 11 Threads, 3 Timer, 1 ISR, 53 Stellen mit Systemcalls
- Nur Interaktionen verwendet (keine Code für Berechnungen)



*d*OSEK

- Framework für OSEK System Analyse und Kernel Generator
- Erstellung des Interaktions-Modells für gegebene Anwendungen

OSEK-V

- CPU Pipeline mit Funktionalität des OSEK Kernel
- Erzeugt Verilog Code für den Zynq-7020 FPGA (ZedBoard)
- Erzeugt einen zylusgeneuen C++ Simulator

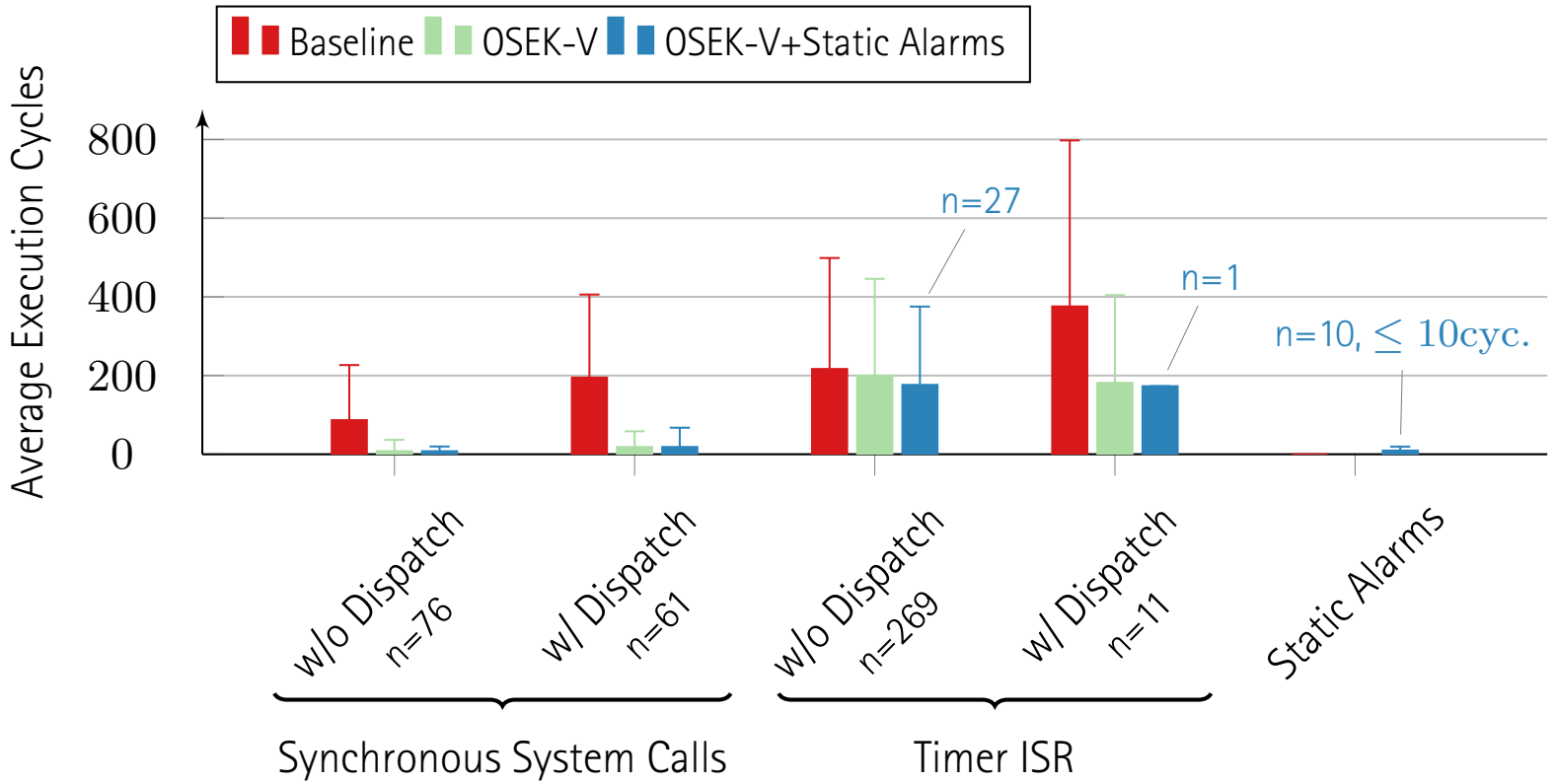
FPGA-Synthese für das ZedBoard (Zynq-7020)

- Erzeugt Zustandsmaschine in 75 s (96 % zum Codieren der Zustände)
  - Vor der Minimierung: 4834 Zustände, 7479 Übergänge
  - Nach der Minimierung: 701 Zustände, 1246 Übergänge
  - Minimierte Wahrheitstabelle: 781 Reihen/Regeln

- Synthetisiert den Rocket in weniger als 10 Minuten mit der Xilinx Toolchain
  - Memory LUTs für die Register (96 %)
  - Logic LUTs für die Zustandsmaschine (76 %)

	Baseline	OSEK-V	+ Static Alarms
Kernel Text Segment (bytes)	14 386	8669	8393
Kernel Data Segment (bytes)	1908	410	354
Lookup Tables (LUT)	29 460	32 041	32 341
Lookup Tables for Memory (Mem-LUT)	1033	2016	2016
Flip-Flops	14 208	14 129	14 196

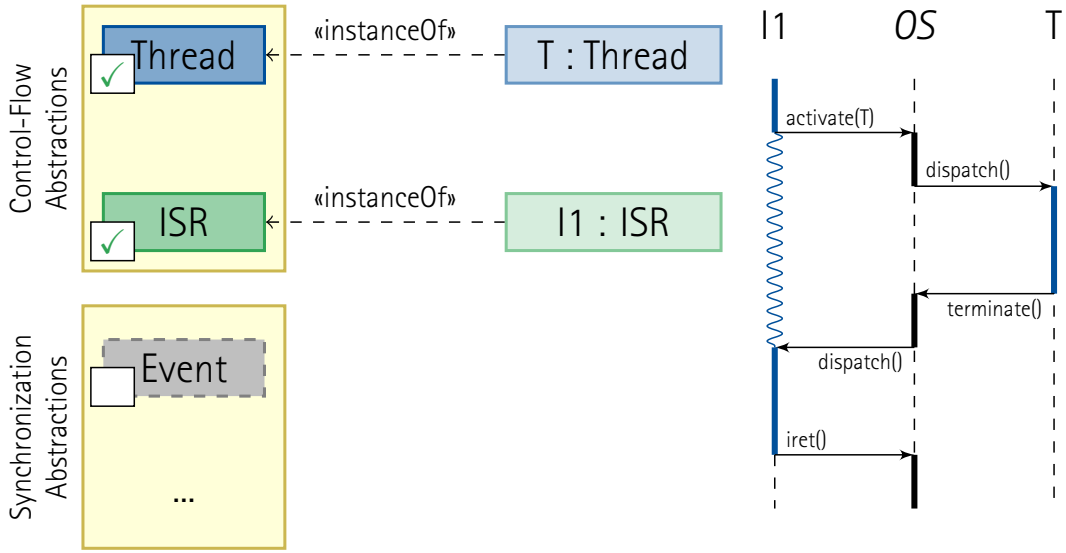
Simulation in Cycle-Accurate Simulator



- Synchrone Syscalls in $\leq$ 25 % der Zeit
- Interrupts ohne Dispatch in $\leq$ 88 % der Zeit
- Interrupts mit Dispatch in $\leq$ 50 % der Zeit
- Reduktion der Sperrzeit der Interrupts durch von 195 auf 41 Zyklen

Forschungsprojekt AHA

Abstraktionen	Instanzen	Interaktionen

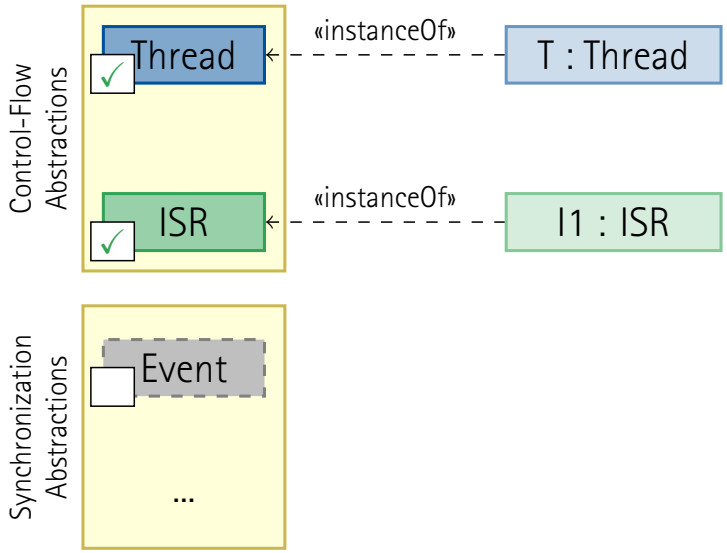


Software

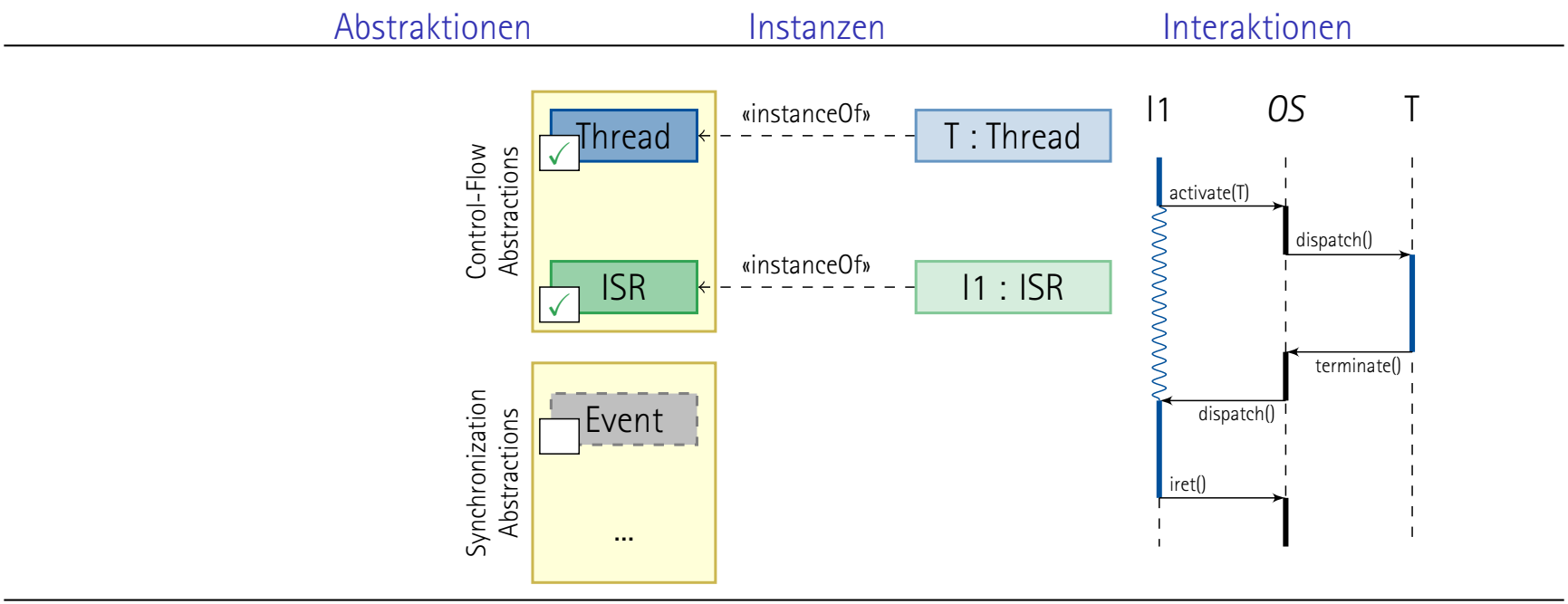
Hardware

OSEK-V

Forschungsprojekt AHA

	Abstraktionen	Instanzen	Interaktionen
			
Software	Linux, eCos, FreeRTOS, ...	OSEK/AUTOSAR	*Synthesis (dynamisch)* dOSEK
Hardware	Atalanta HW-RTOS (μITRON)	*Sloth (COTS)* *FlexPRET*	OSEK-V

Forschungsprojekt AHA



- Welche Tiefe der Spezialisierung ist für welche Anwendung sinnvoll?
- Wie kann man sinnvoll vergleichen? (⇒ Automatisierung)

⇒ Forschungsprojekt **AHA** (Automatisierte Hardware-Abstraktion)

gefördert durch die DFG