FACHHOCHSCHULE **TRIER**
Hochschule für Technik, Wirtschaft und Gestaltung
University of Applied Sciences
Informatik

SimTrOS: A Heterogenous Abstraction Level Simulator for Multicore Synchronization in Real-Time Systems

Jörn Schneider / Michael Bohn / Christian Eltges

Trier University of Applied Sciences

SPONSORED BY THE
Federal Ministry of Education and Research

Another Simulator?

Another?

SimTrOS: A Heterogenous Abstraction Level Simulator for Multicore Synchronization in Real-Time Systems *

Jörn Schneider, Michael Bohn, Christian Eltges
Dept. of Computer Science
Trier University of Applied Sciences
Trier, Germany
{j.schneider, m.bohn, c.eltges}@fh-trier.de

***Abstract*—To provide a common ground for the comparison of real-time multicore synchronization protocols we developed a framework that supports heterogenous levels of abstraction for simulated functionality and simulated timing. Our intention is to make the simulator available to the real-time research**

that the simulator core itself can be used for any timing evaluation of multicore real-time systems and moreover, that the novel idea of heterogenous abstraction levels that lies at the heart of its design can also be a key to fast

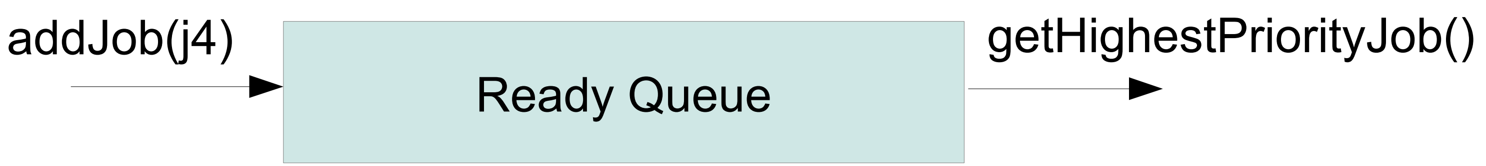
Context

- Initial purpose of the simulator
  - Evaluation of Multicore-Resource-Protocols
  - MPCP, FMLP, MSRP, ...

- At least two different aspects
  - Functional behaviour
    - Global critical sections
  - Timing behaviour
    - Which protocol performs best for a specific scenario?
      - Our scenario: AUTOSAR

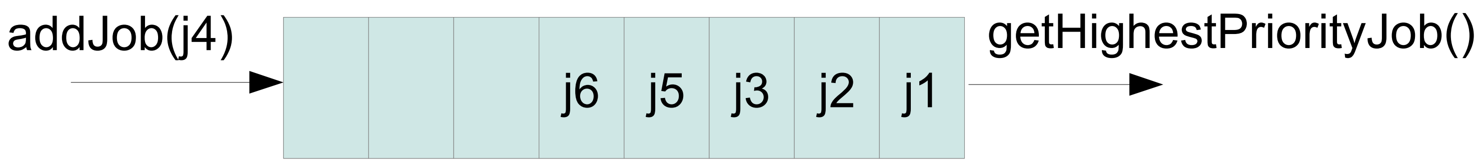
Teaser

- Heterogenous Abstraction Levels at work:



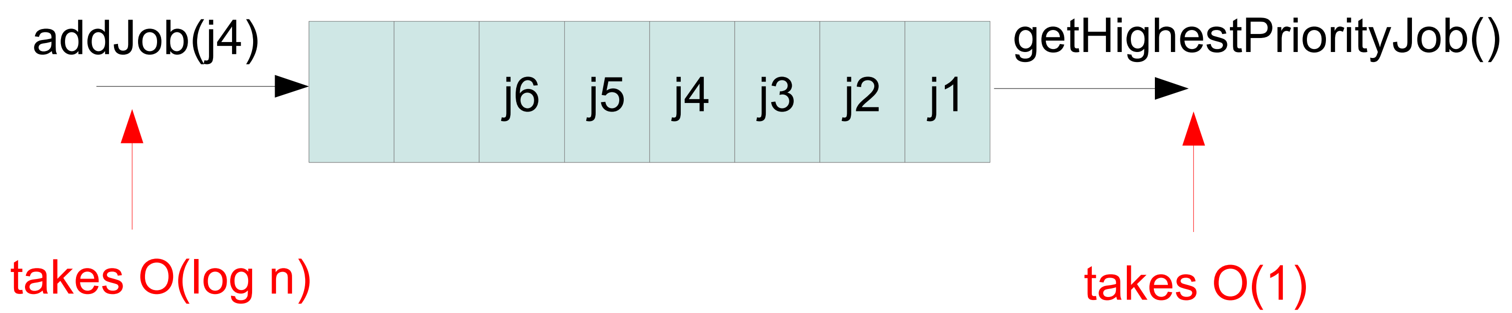
Teaser

- Heterogenous Abstraction Levels at work:
- Implementation 1: Sorted job list



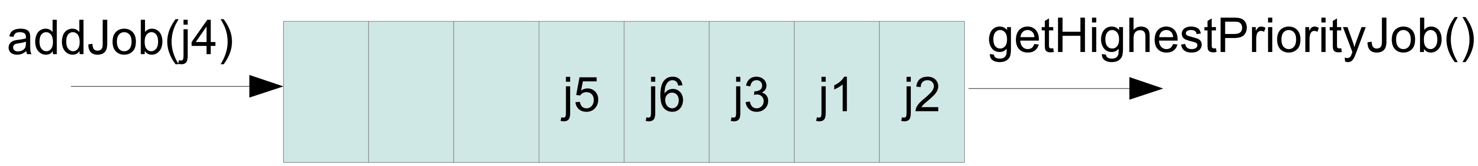
Teaser

- Heterogenous Abstraction Levels at work:
- Implementation 1: Sorted job list



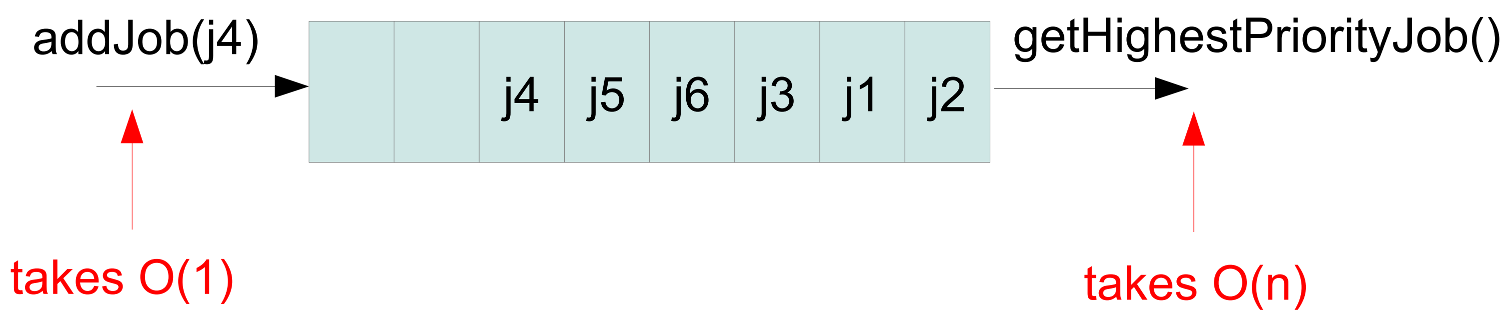
Teaser

- Heterogenous Abstraction Levels at work:
- Implementation 2: Job set



Teaser

- Heterogenous Abstraction Levels at work:
- Implementation 2: Job set



Teaser

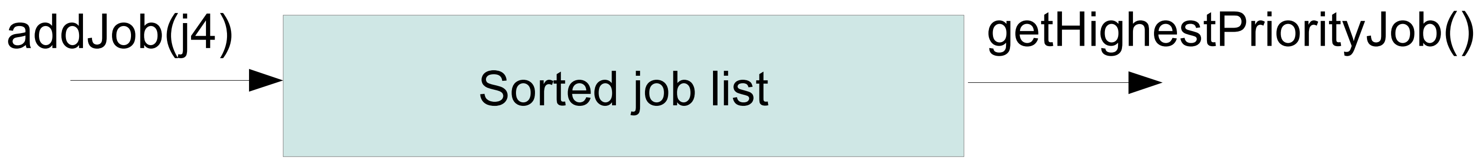
- Heterogenous Abstraction Levels at work:
- Which implementation to choose?
  - Implementation 1: Sorted job list
  - Implementation 2: Job set

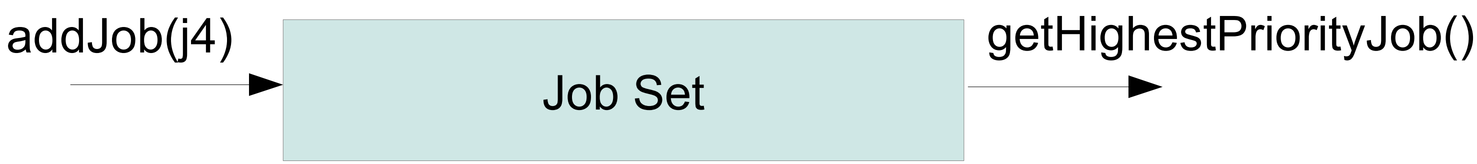
- Simulate both implementations
  - Means: implement both variants?

Teaser

- Heterogenous Abstraction Levels at work:
- Which implementation to choose?
  - Implementation 1: Sorted job list
  - Implementation 2: Job set
- Simulate both implementations
  - ~~Means: Implement both variants?~~

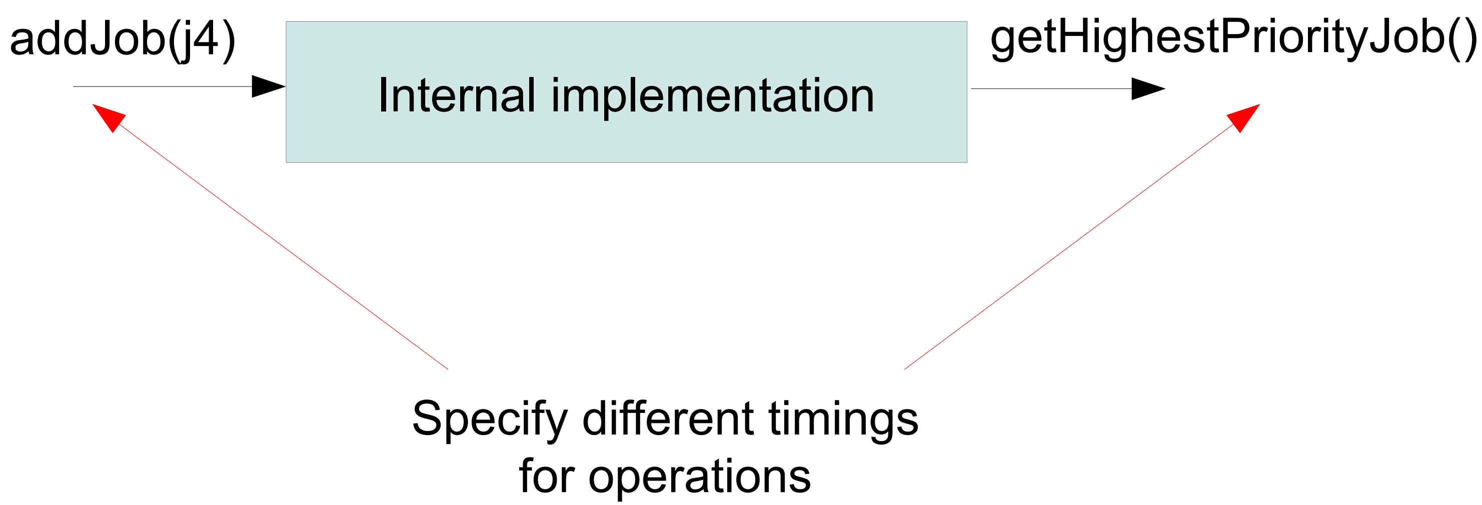
Teaser



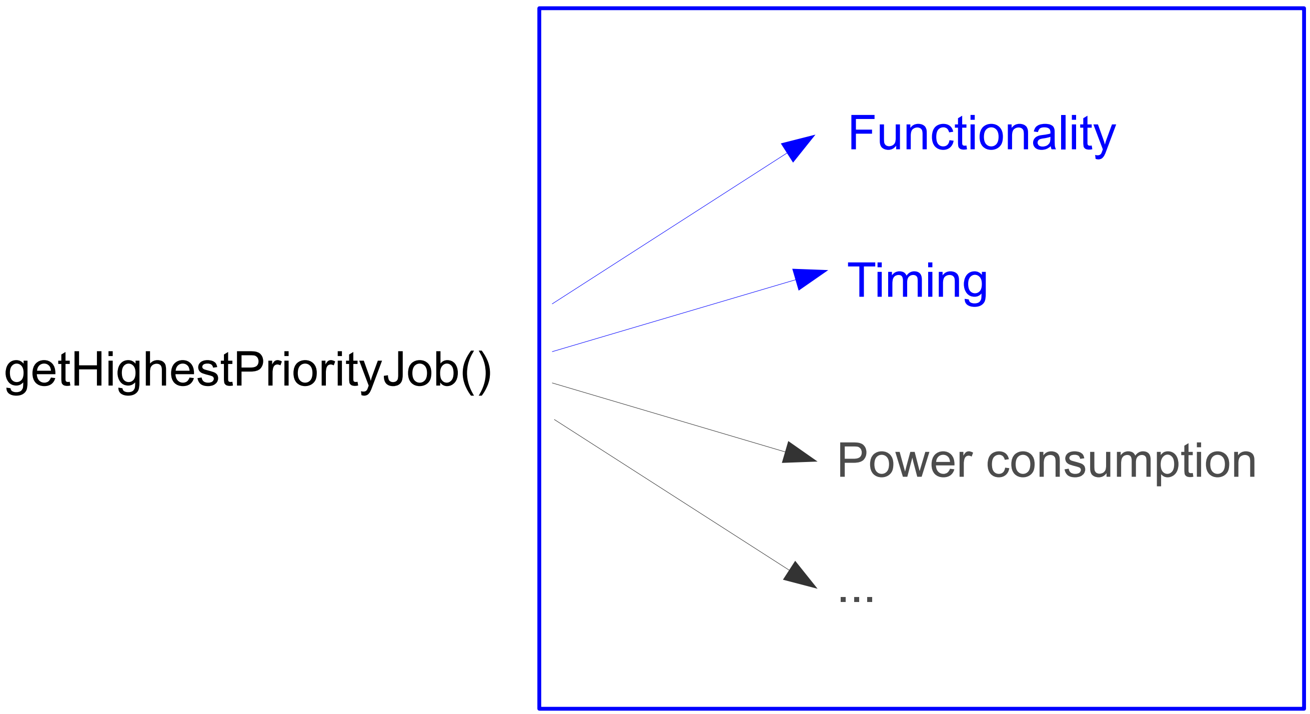


Functionality the same – only timing differs!

Teaser



addJob(j4)
Internal implementation
getHighestPriorityJob()
Specify different timings
for operations

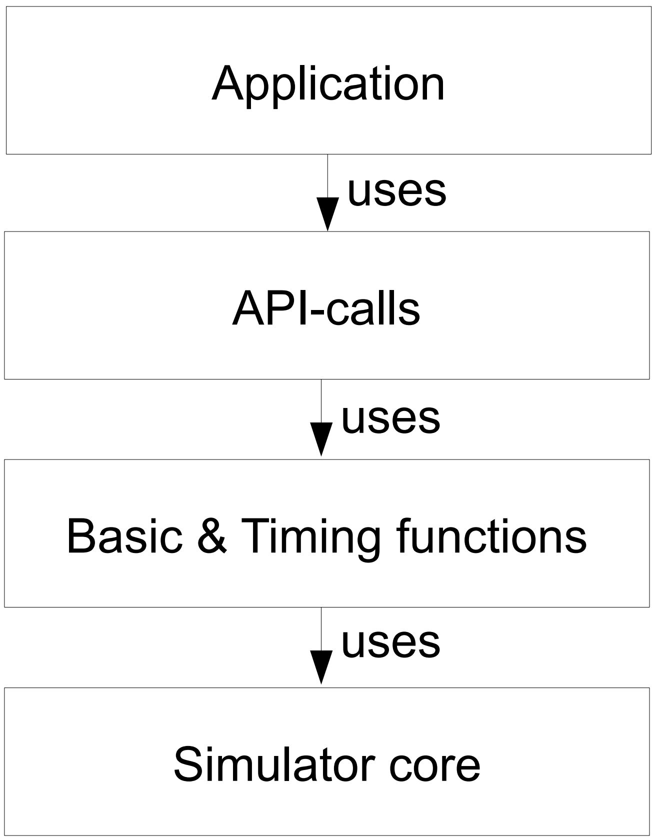


Heterogenous abstraction levels

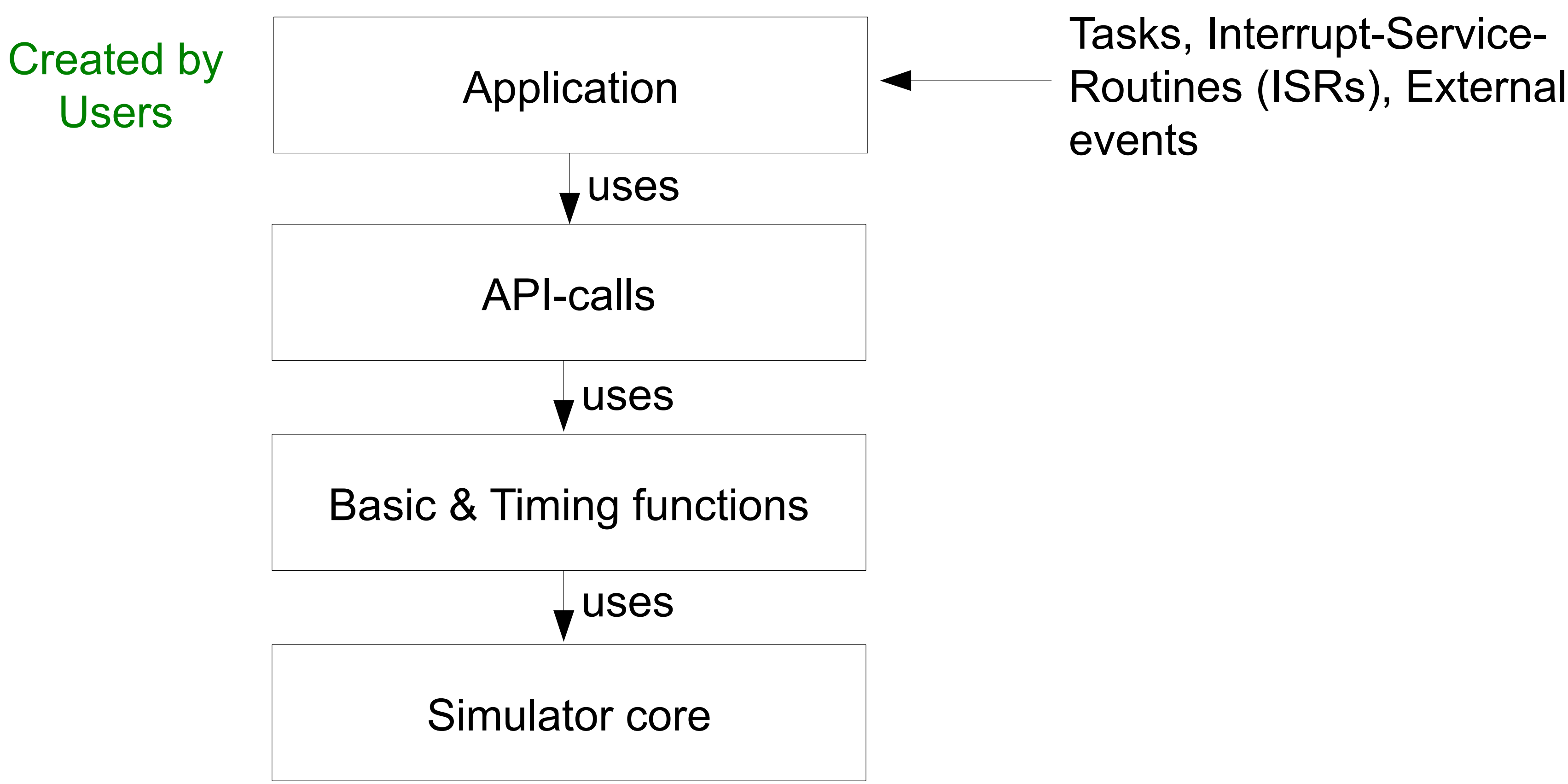
Rest of the talk

- Usage of the simulator
- Internals of the simulator
- Conclusion

Layered architecture



Layered architecture



Defining applications

- Task definition

```
task_i = autosarTask {
    taskPeriod = 100,
    taskPhase = 0,
    taskPriority = 1,
    taskName  = ”task i”,
    taskCore = 0,
    taskProgram = do {
            osGetResource ”R1”;
            time 33;
            osReleaseResource ”R1”;
            time 5;
            osTerminateTask;
    }
}
```

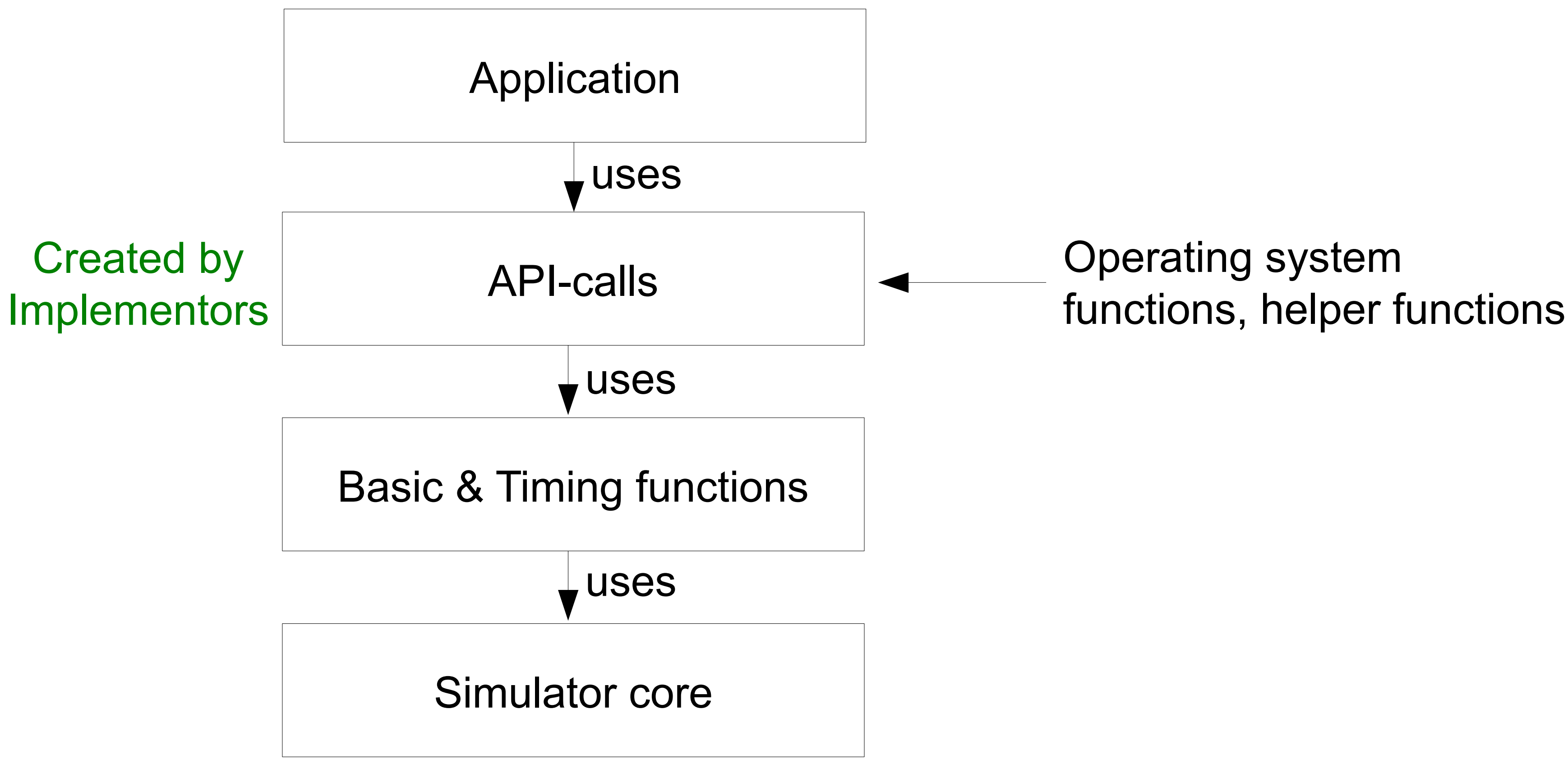
API-calls

Defining applications

- Event definition

```
event_j = event {
    eventPeriod = Infinity,
    eventPhase = 70,
    eventName = "event_j",
    eventEffect = startISR 1 ( do {
                                osActivateTask task3
                          })      interrupt on core 1
    }
```

Layered architecture



Defining API-calls

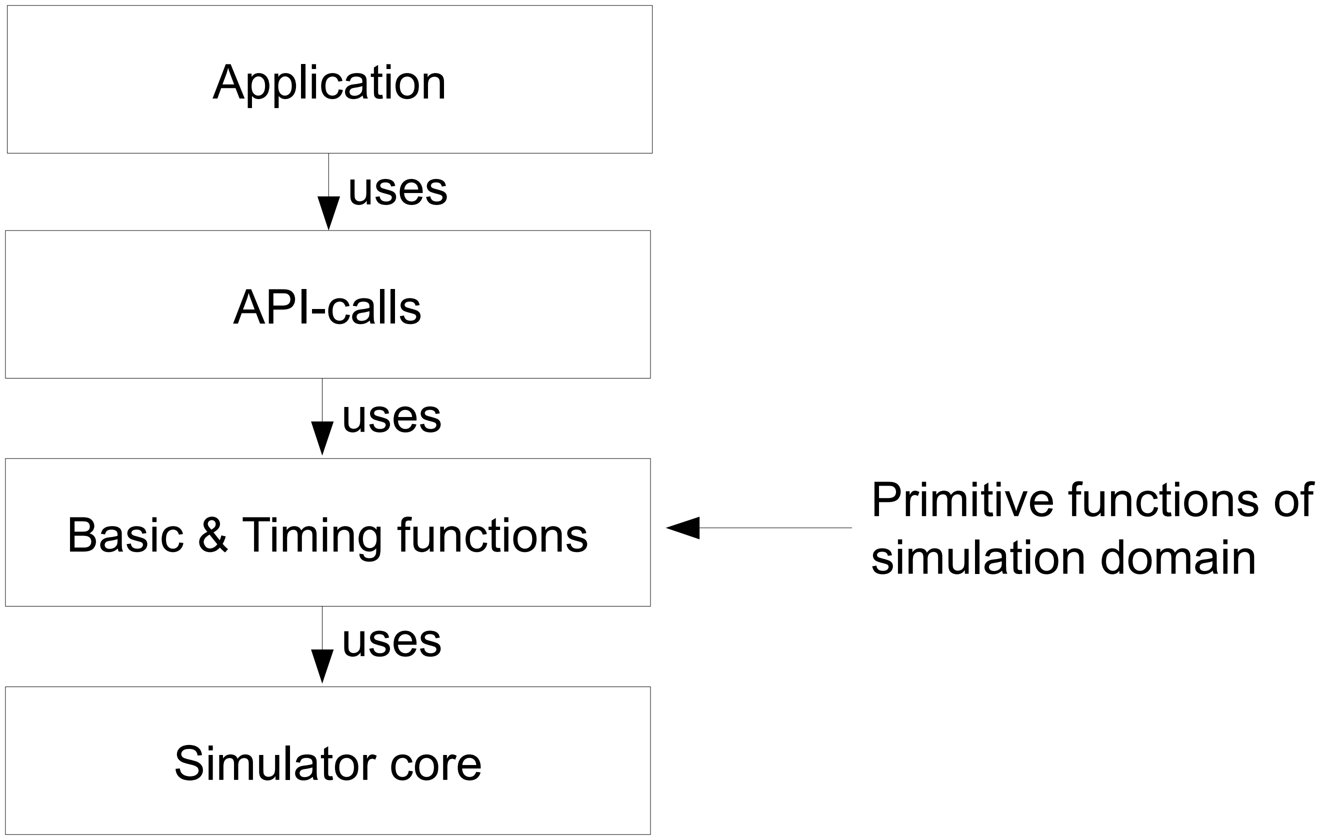
- API-call examples

```
osTerminateTask = do {
    setJobVar “state” Suspended;
    schedule;
}

schedule = do {
   j <- getHighestPriorityJob;
   setRunningJob j;
}
```

Basic function calls

Layered architecture



Defining basic functions

getHighestPriorityJob

Functionality (effect)

getHighestPrioJobImpl
→ pure Haskell

Timing

linearTime s = 10 + 5   length (getL
readyQueue s)

constTime s = 10
→ pure Haskell

Defining basic functions

getHighestPriorityJob

getHighestPriorityJob =
  makeBasicFunction
    linearTime getHighestPrioJobImpl

Functionality

getHighestPrioJobImpl
→ pure Haskell

Timing

linearTime s = 10 + 5   length (getL
                                readyQueue s)

constTime s = 10
→ pure Haskell

Simulating a system

FACHHOCHSCHULE TRIER
Hochschule für Technik, Wirtschaft und Gestaltung
University of Applied Sciences
Informatik

- Compile with GHC (Glorious Glasgow Haskell Compiler)
  Application code + API-calls + Basic functions + Simulator core
- Run executable
  - Interactive (step-wise)
  - Non-Interactive
    - End of simulation
    - Time limit
  - Writes XML-Logfile during simulation

Simulator core

- The core of the simulator
  - Discrete event simulation engine
  - “Hops” from event to event
  - Skips time where nothing happens

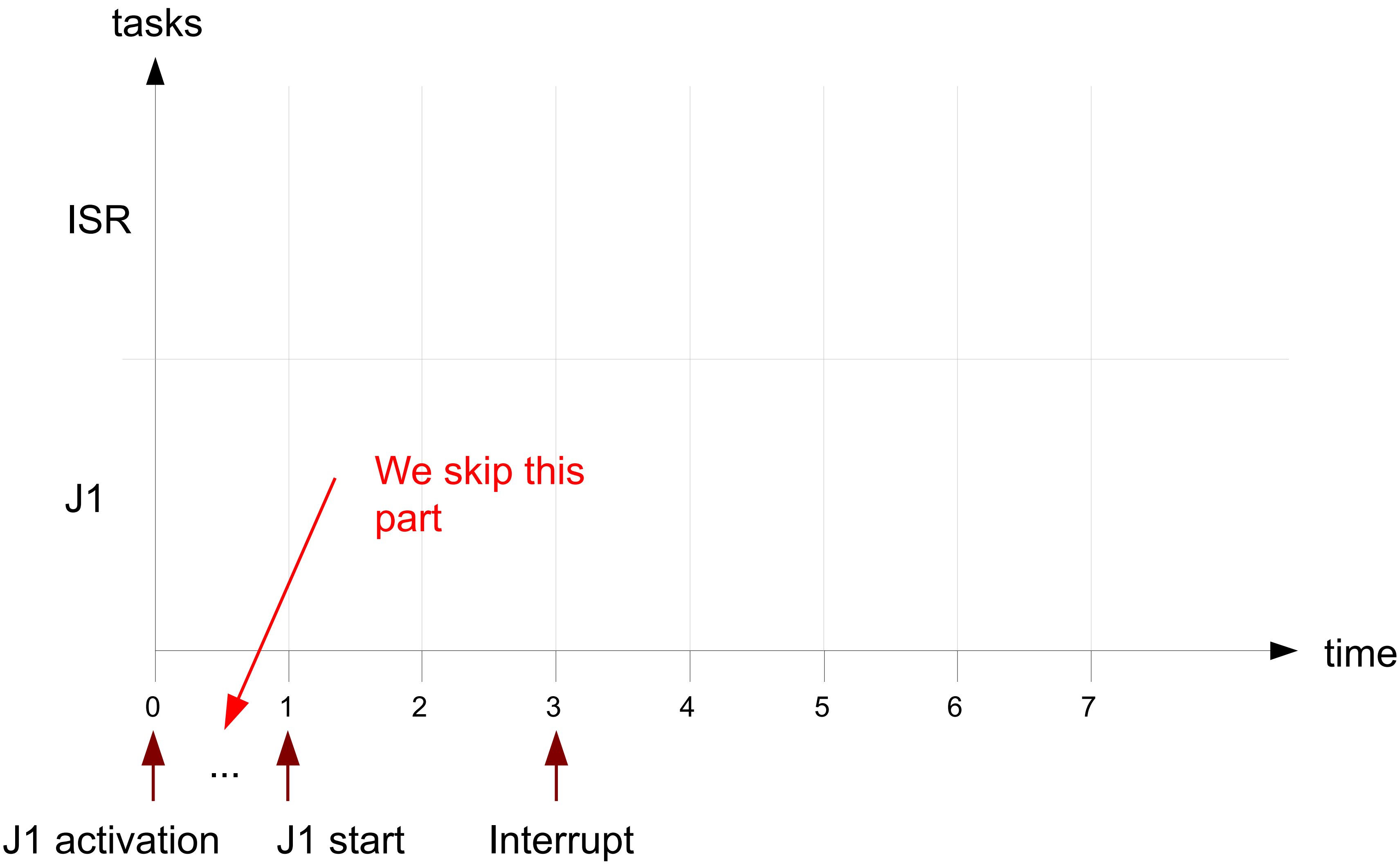
Simulator core

- Single-core example

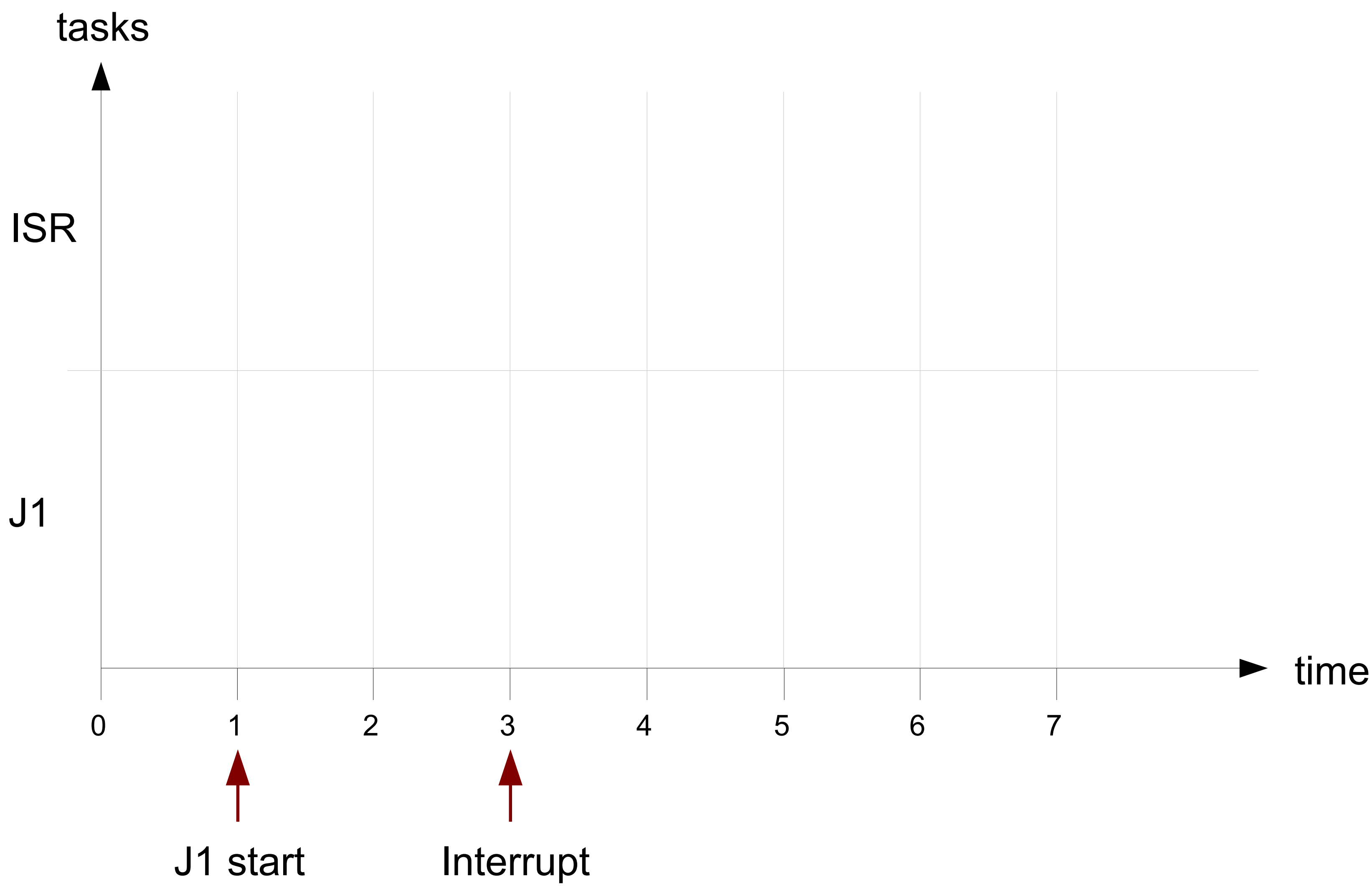
```
task_j1 = autosarTask {
    taskPeriod = Infinity,
    taskPhase = 0,
    taskPriority = 1,
    taskName  = ”J1”,
    taskCore = 0,
    taskProgram = do {
            time 4;
    }
}
```

```
event_isr = event {
    eventPeriod = Infinity,
    eventPhase = 3,
    eventName = “ISR”,
    eventEffect = startISR 0 ( do {
                        time 1;
                        rfi ;
                   })
}
```

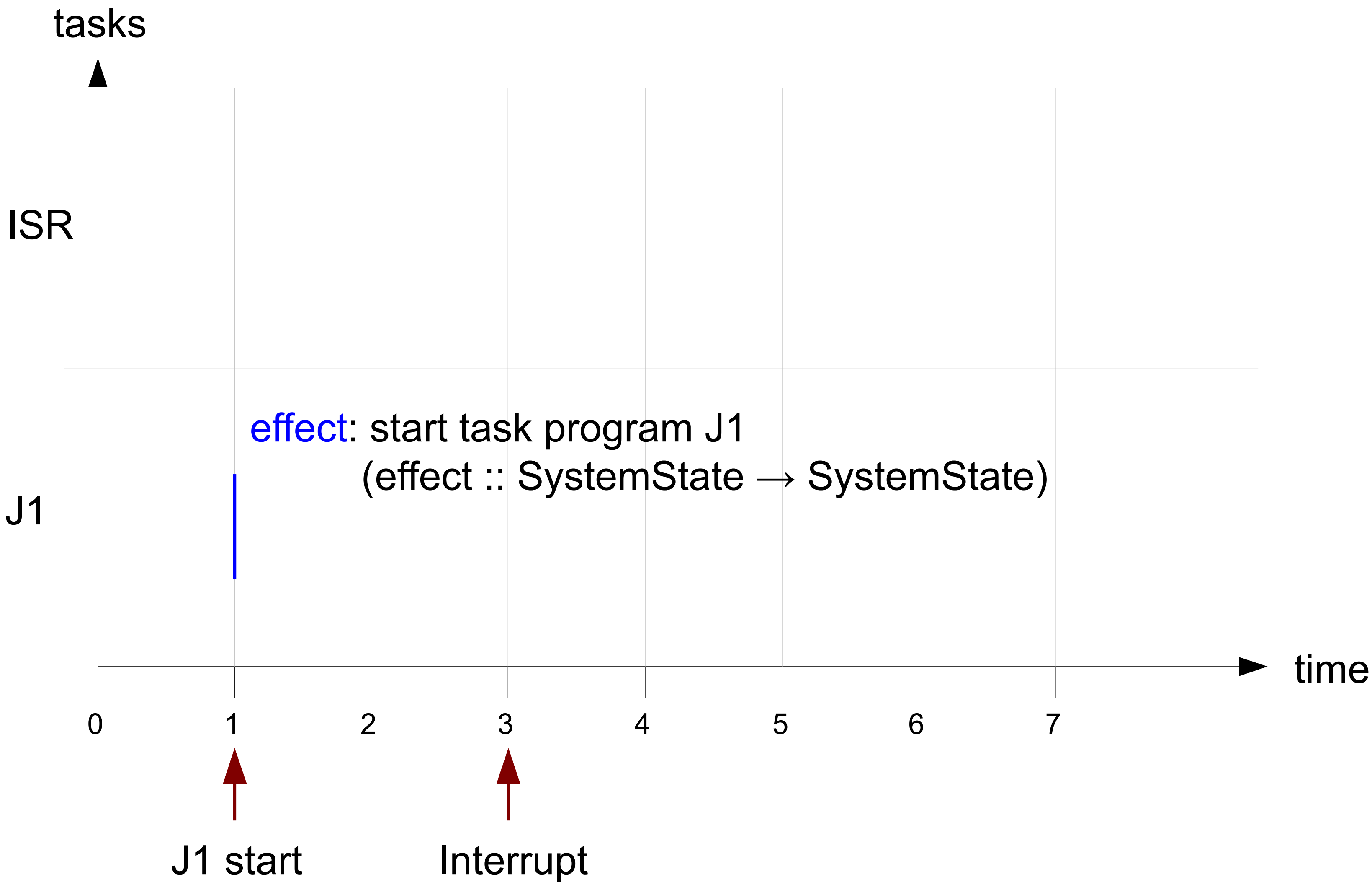
Simulator core



Simulator core

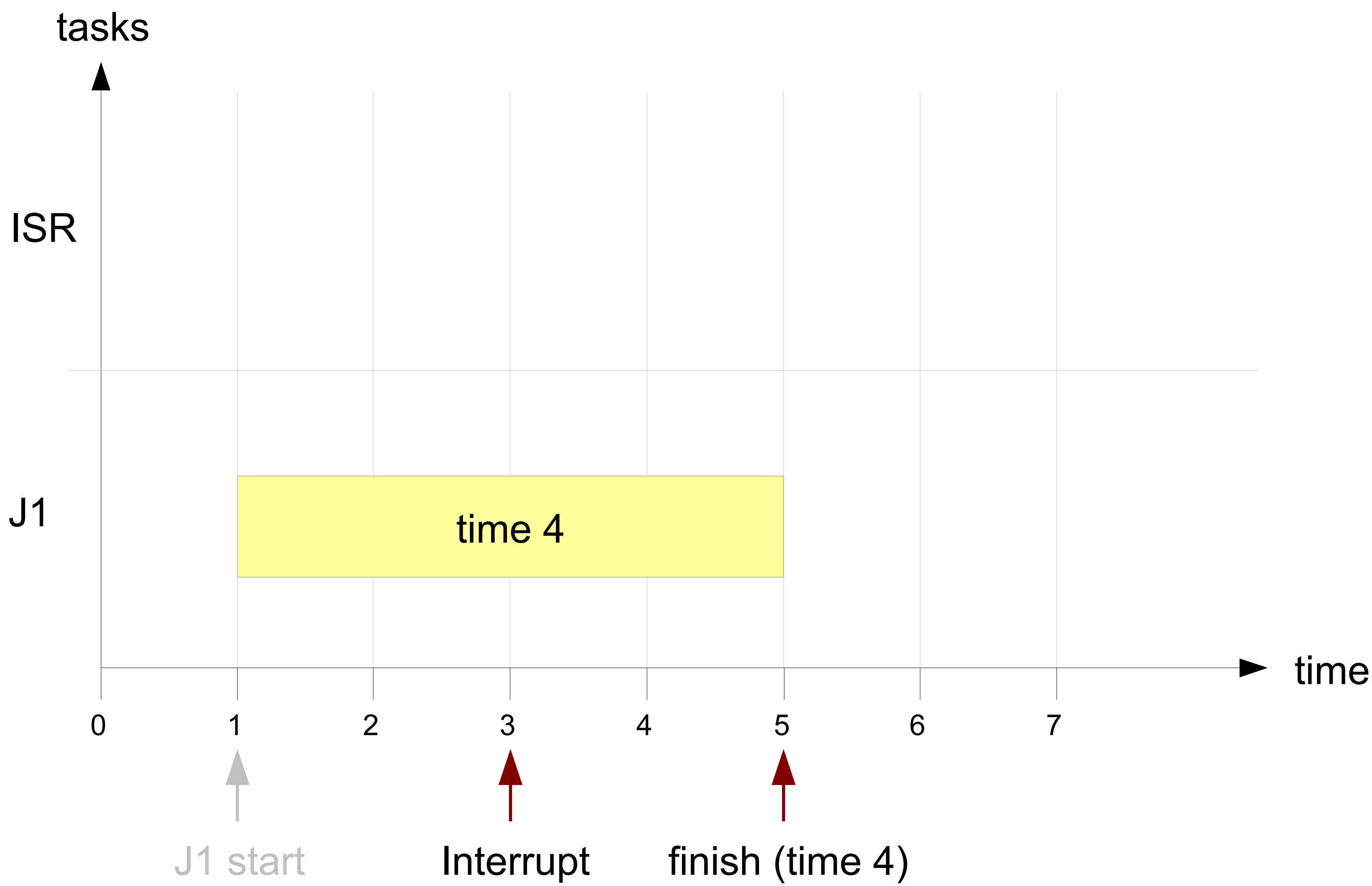


Simulator core

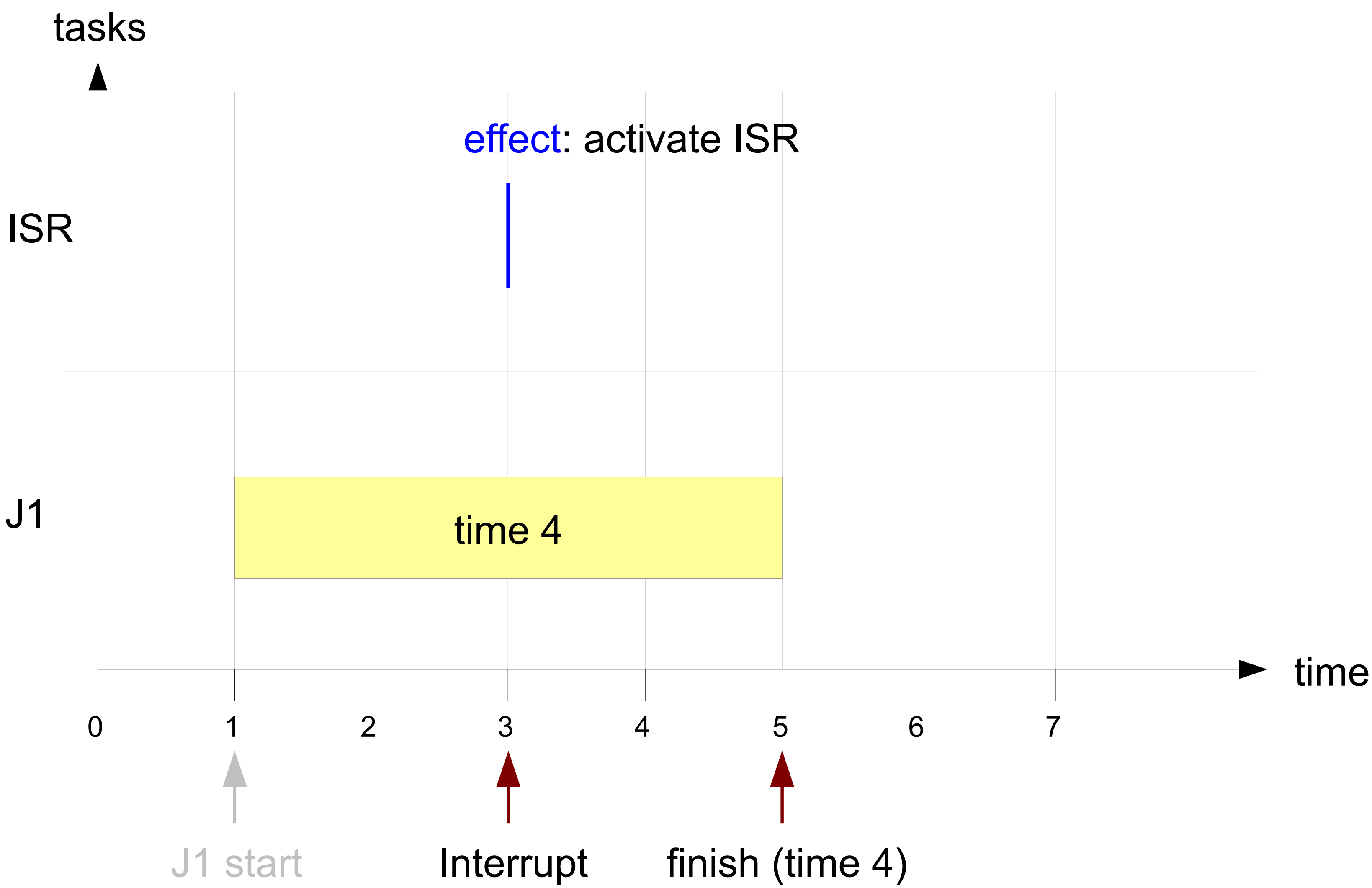


Simulator core

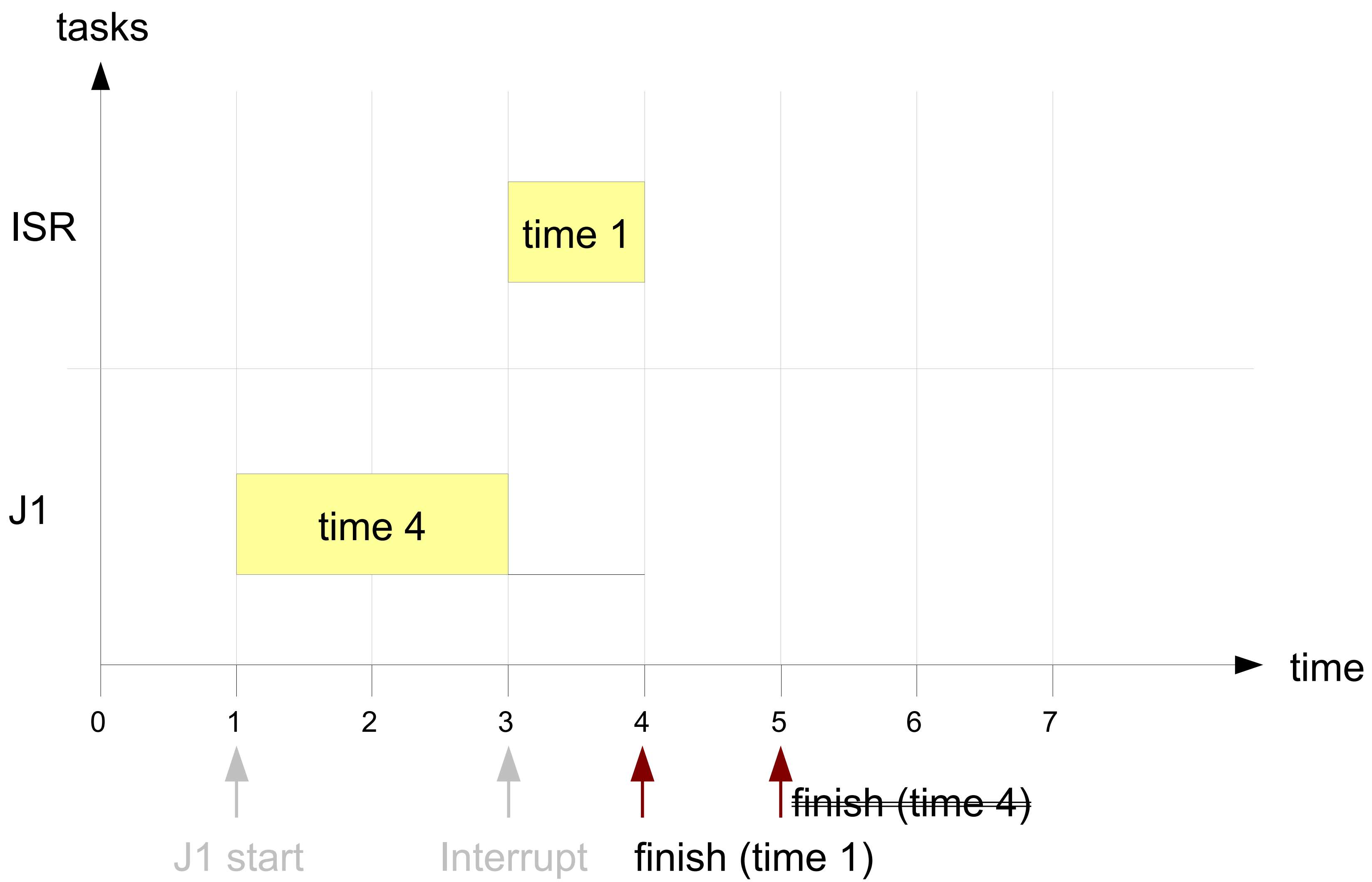
FACHHOCHSCHULE TRIER
Hochschule für Technik, Wirtschaft und Gestaltung
University of Applied Sciences
Informatik



Simulator core

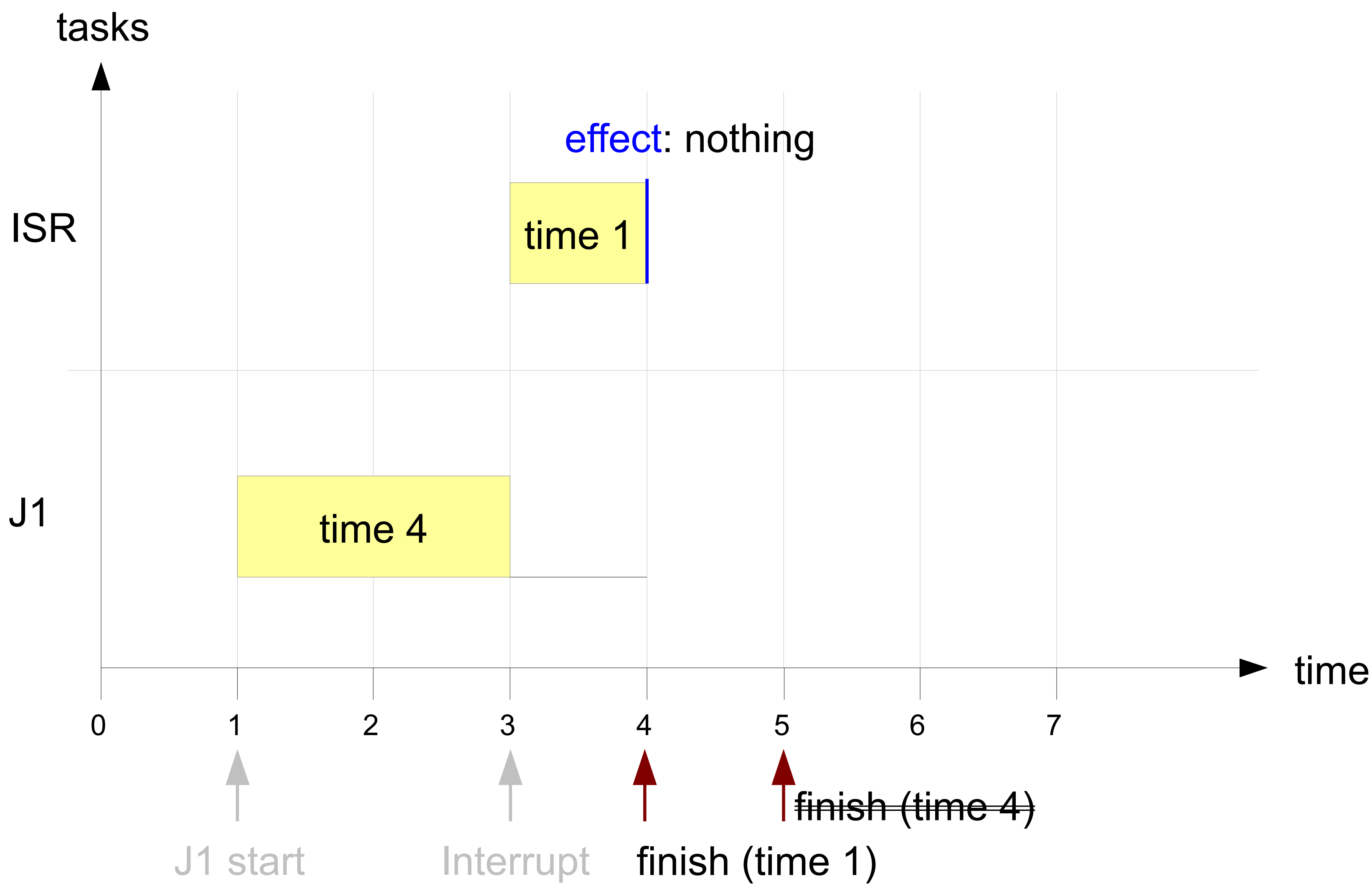


Simulator core

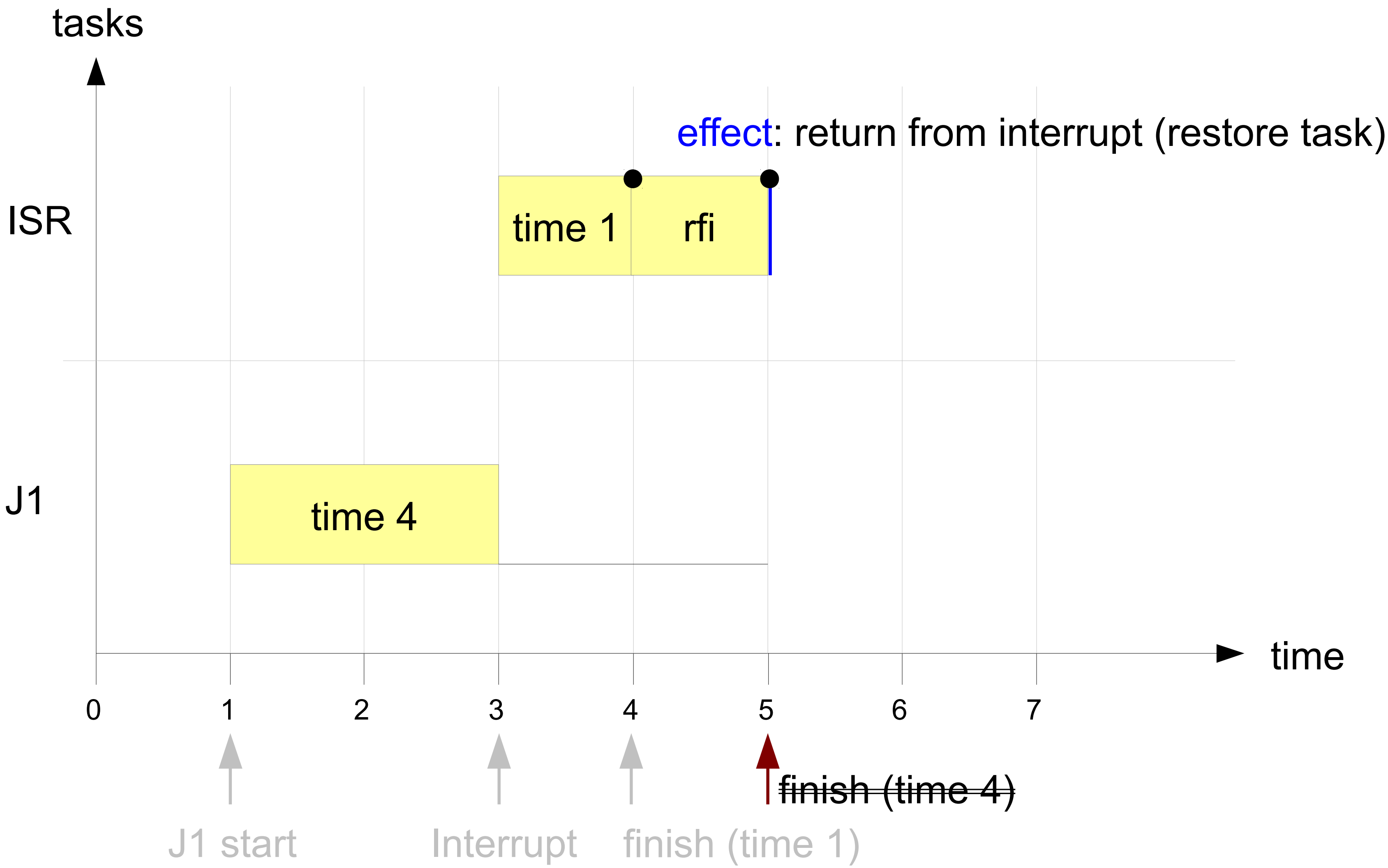


Simulator core

FACHHOCHSCHULE TRIER
Hochschule für Technik, Wirtschaft und Gestaltung
University of Applied Sciences
Informatik

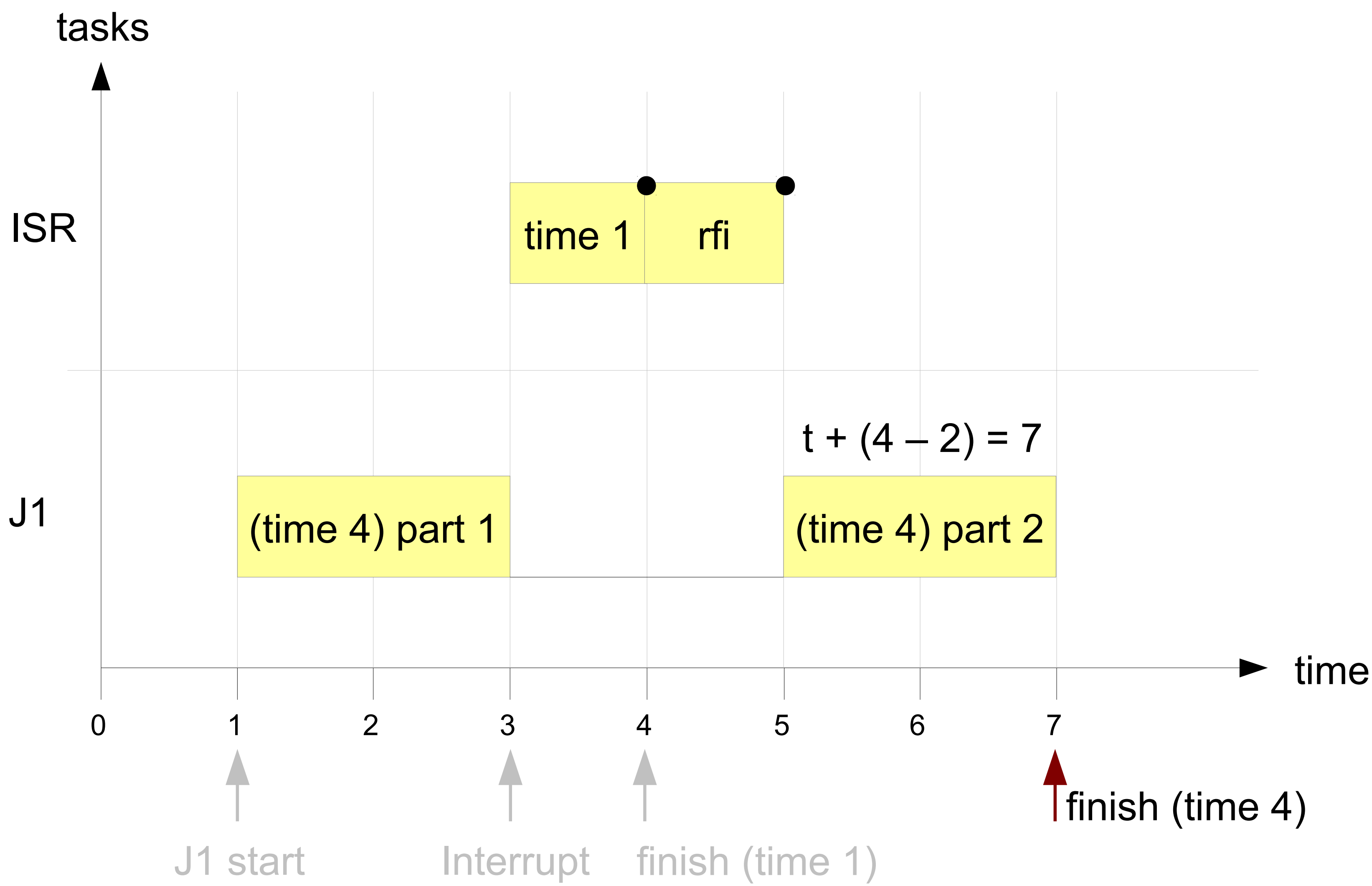


Simulator core

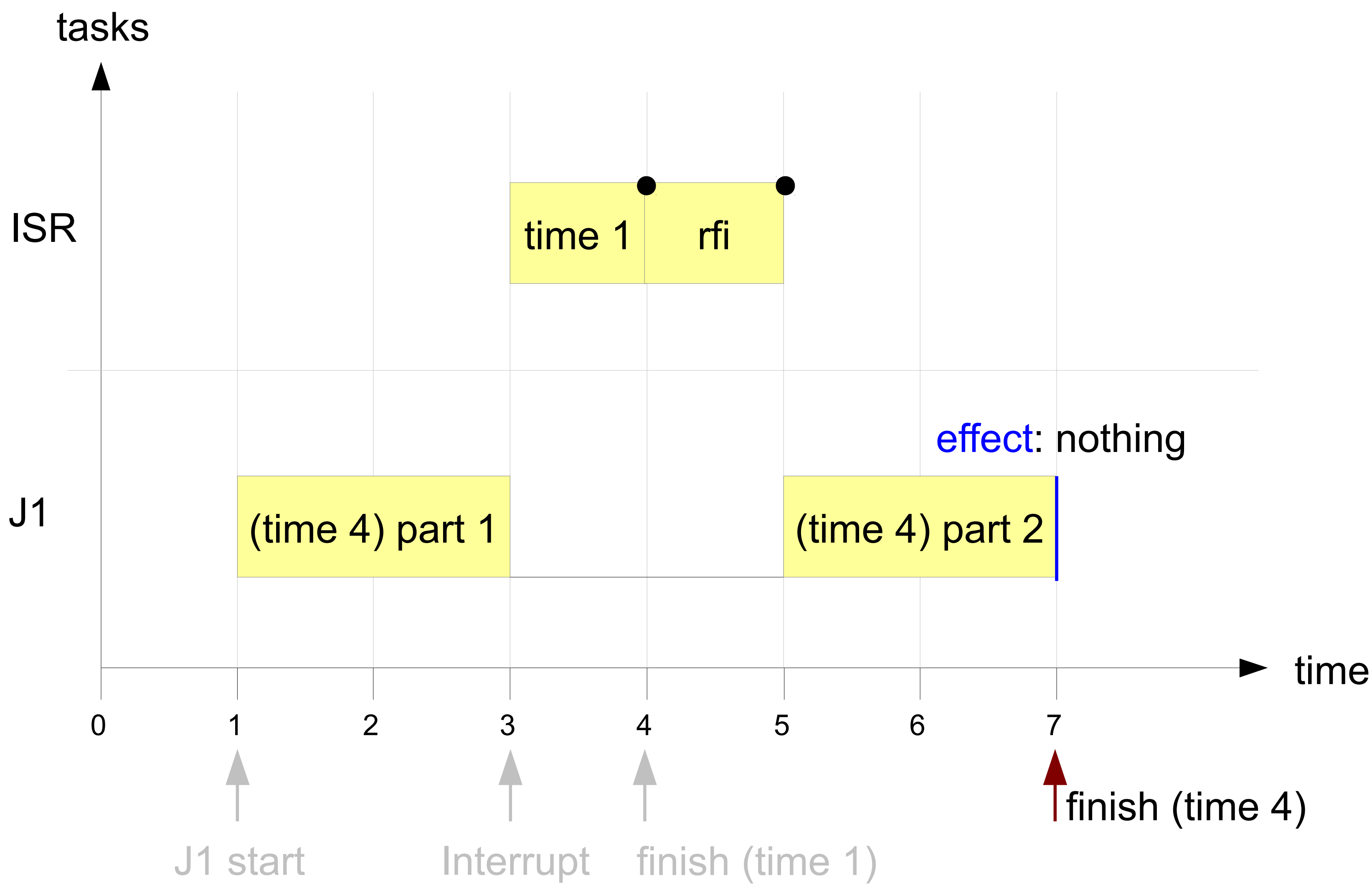


Simulator core

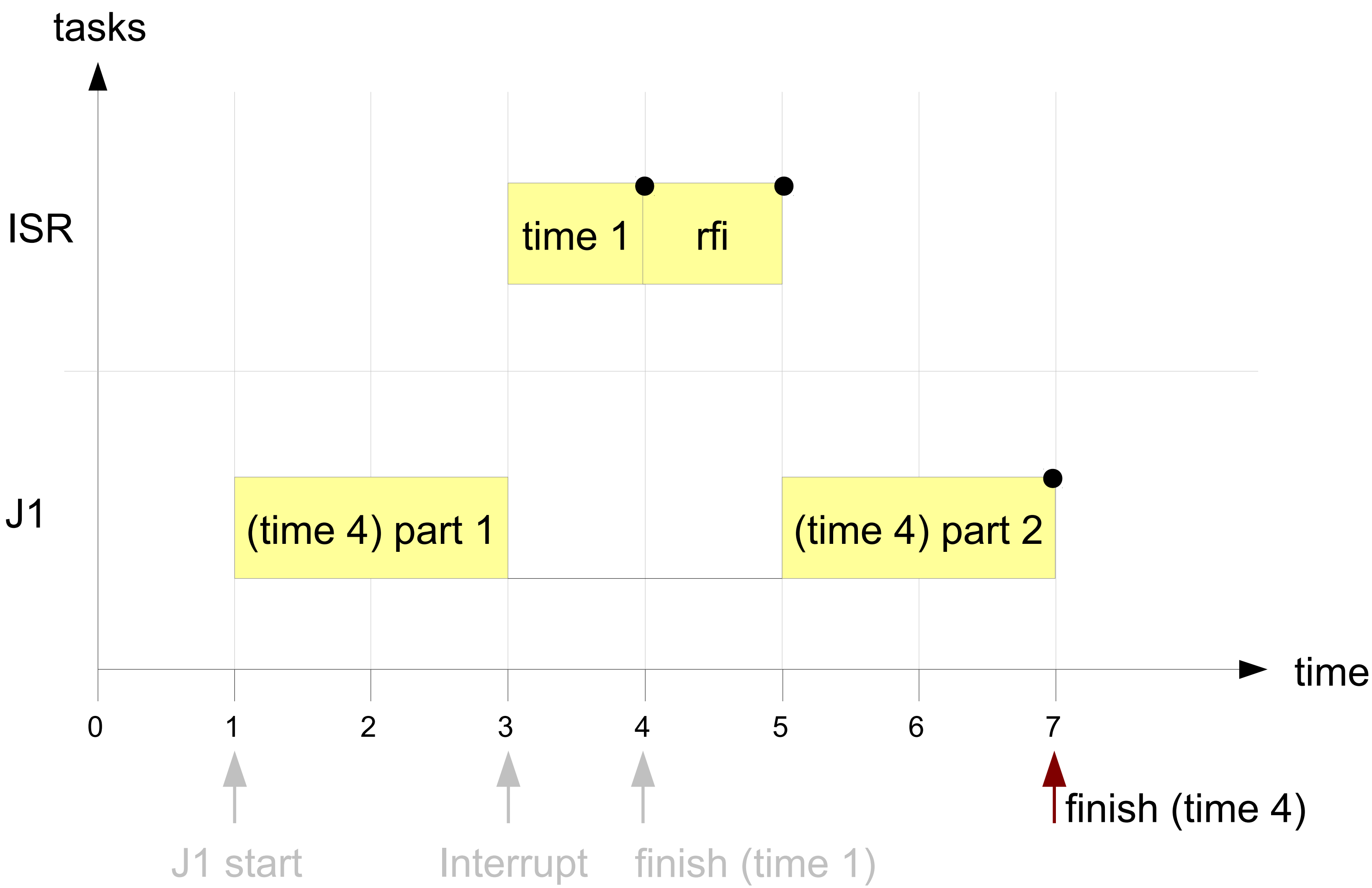
FACHHOCHSCHULE TRIER
Hochschule für Technik, Wirtschaft und Gestaltung
University of Applied Sciences
Informatik



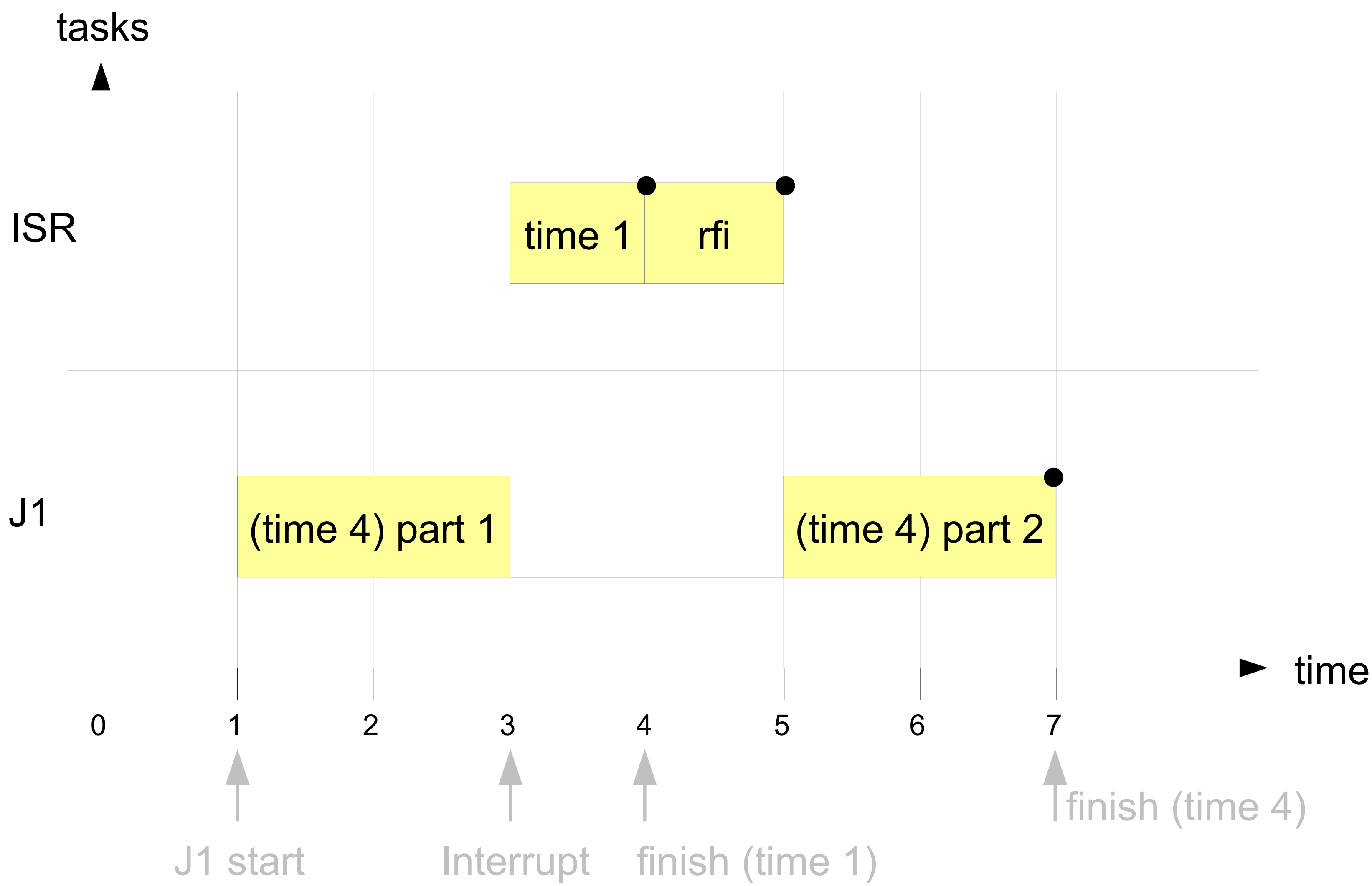
Simulator core



Simulator core



Simulator core



Simulator core

- E = nearest external event
- B_i = finishing time of executing basic function on core i

- Single-core
  - next event: *min(E, B_0)*

- Generalising to Multi-core (n cores)
  - next event: min(E,B_0, …, B_{n-1})

Simulator core

- What's with non-determinism?
  - External event effect and basic function finishes at same time
    - External event effect occurs before basic function effect
  - Two basic functions finish at same time (only multicore)
    - => User-supplied decision function called

Conclusion

FACHHOCHSCHULE TRIER
Hochschule für Technik, Wirtschaft und Gestaltung
University of Applied Sciences
Informatik

- Key feature of SimTrOS
  - Separation between timing and functionality
  - Evaluate implementations that differ on timing behaviour only, without touching functional implementation
- Simulator will be available as open source:
  - Timeframe: this year
- We hope to see contributions by the community
  - Task sets
  - Operating system implementations
  - ...

Last slide...

Questions?

More information:

m.bohn@fh-trier.de
www.fh-trier.de/go/simtros