System Software for Resource Arbitration on Future Many-* Architectures

2nd Workshop on Resource Arbitration for Dynamic Runtimes (RADR)

2020-05-18

Florian Schmaus[1] Sebastian Maier[1] Tobias Langer[1]
Jonas Rabenstein[1] Timo Hönig[1] Lars Bauer[2] Jörg Henkel[2]
Wolfgang Schröder-Preikschat[1]

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
[2] Karlsruhe Institute of Technology (KIT)

Chair in Distributed Systems and Operating Systems

DFG

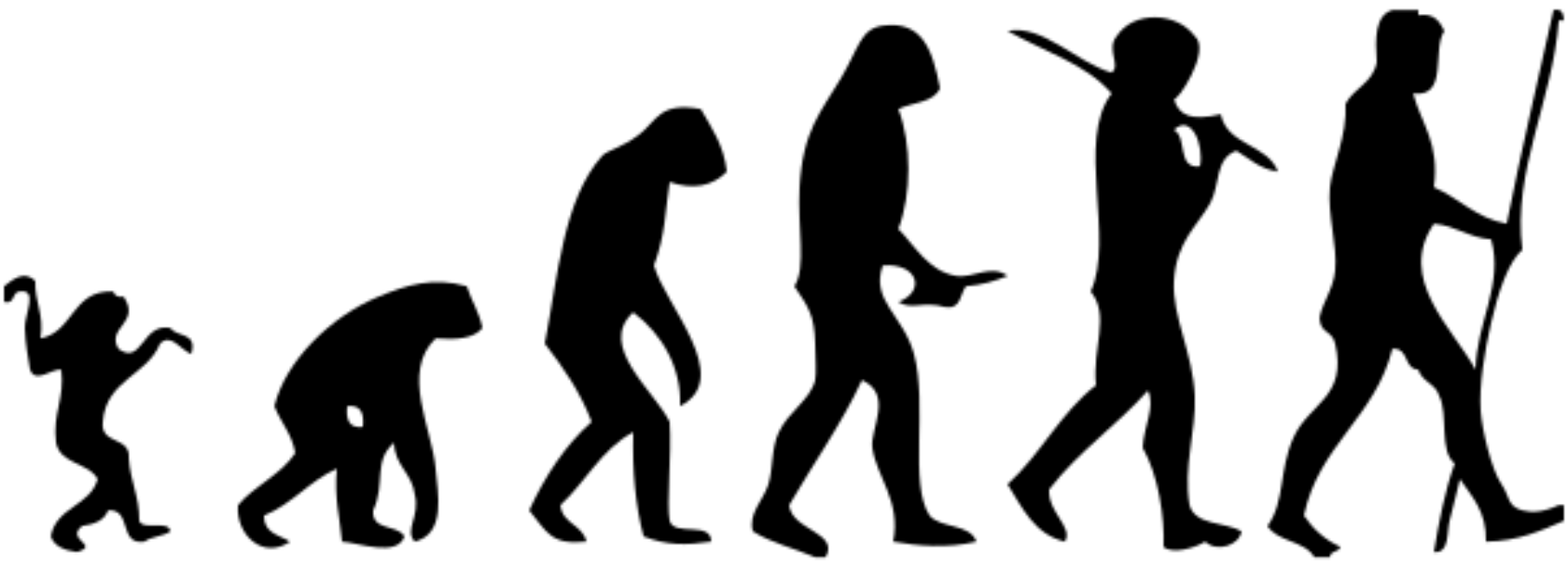
FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

Supported by German Research Council (DFG): CRC/TRR 89 (InvasIC, Project C1)

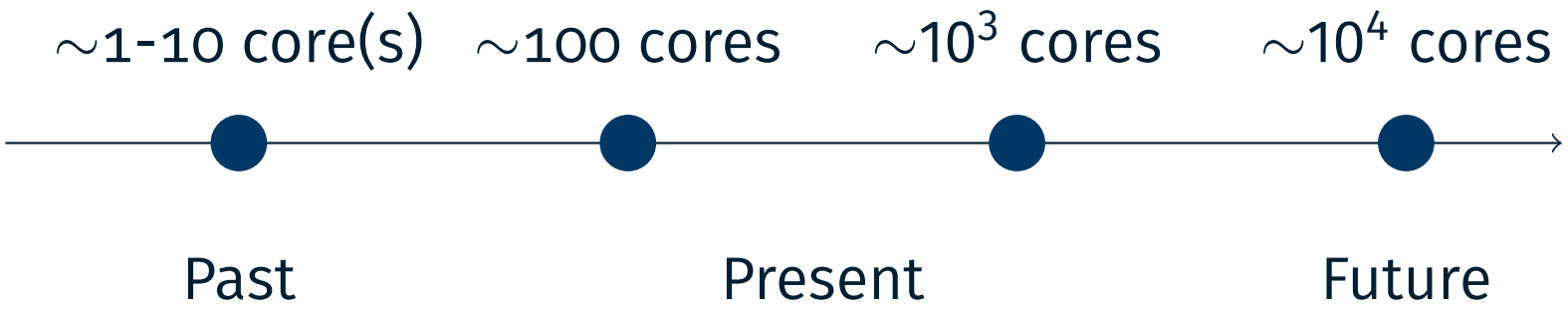
Computing Evolution

As every ecosystem, the computing one is subject to evolution. This includes:

- Application Development
- Computing Requirements
- Computer Architecture



Computer Architecture



Many-Core Era

- Distances increase, incl. computation to data
- In- and near-memory computing

Computer Architecture (cont.)

Global Cache-Coherency

Pure Shared-Memory System (e.g., Numascale)

Tile-Based Architecture

Hybrid Shared-Memory and Message-Passing

Trend towards Heterogenity

- Accelerators (Tensor PU, ...)
- big.LITTLE
- Mixing basic and extended instruction set cores
- Dynamically reconfigurable cores / processing elements

New Challenges

Satisfy *intrinsic* constraints/requirements — Application C2

- Energy/Power
- Dependability
- Heterogenity
- ...

Counter *extrinsic* uncertainties — Application C3

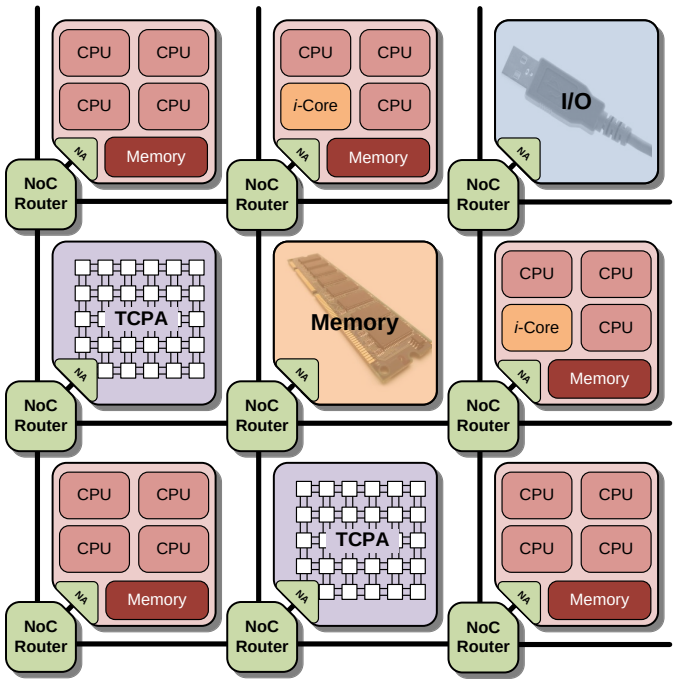
- React in real-time to unpredictable input sizes
- Monitor and enforce non-functional properties like latency, throughput, ...

Many-* Architecture — Hardware C1

- Concurrency and parallelism become necessary
- Where to perform the computation? **Data Locality**
- Heterogenity: Which and how many * to use?

Invasive Computing

Invasive Computing (InvasIC)



- Fudamental research of future many-core systems
  - 3 phases á 4 years and ~9M €
- Tackles the challenges on every layer of the technology stack
  - Hardware Architecture
  - Operating System
  - Runtime Environment
  - Compiler
- Collaboration between layers is decisive
- HW/SW Co-Design

Find out more about at invasic.de

Invasive Run-Time Support System (*i*RTSS)

System Software Needs to Adapt

Application

Hardware

System Software Needs to Adapt

Application

- Satisfy *intrinsic* constraints/requirements (C2)
- Counter *extrinsic* uncertainties (C3)

Hardware

- Many-∗ Architecture (C1)

System Software Needs to Adapt

Application

- Satisfy *intrinsic* constraints/requirements (C2)
- Counter *extrinsic* uncertainties (C3)

System Software

Adjust Architecture and Design?

Hardware

- Many-∗ Architecture (C1)

System Software Needs to Adapt

Application

- Satisfy *intrinsic* constraints/requirements (C2)
- Counter *extrinsic* uncertainties (C3)

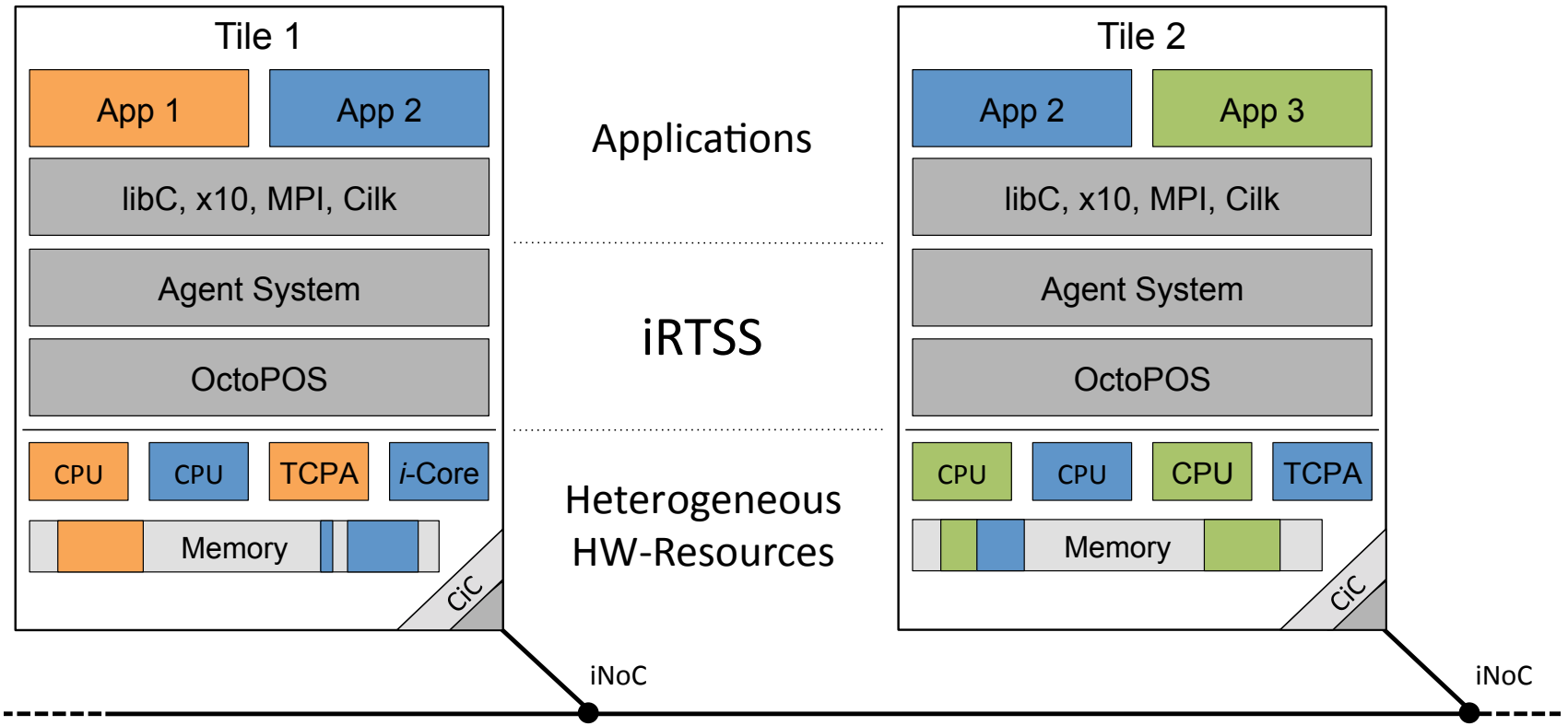
System Software

- Allow variable and dynamic application demand
- Ensure user constraints
- Support plentiful and heterogenous hardware resources
- Provide inter-tile communication primitives

Hardware

- Many-∗ Architecture (C1)

Invasive Run-Time Support System (*i*RTSS)



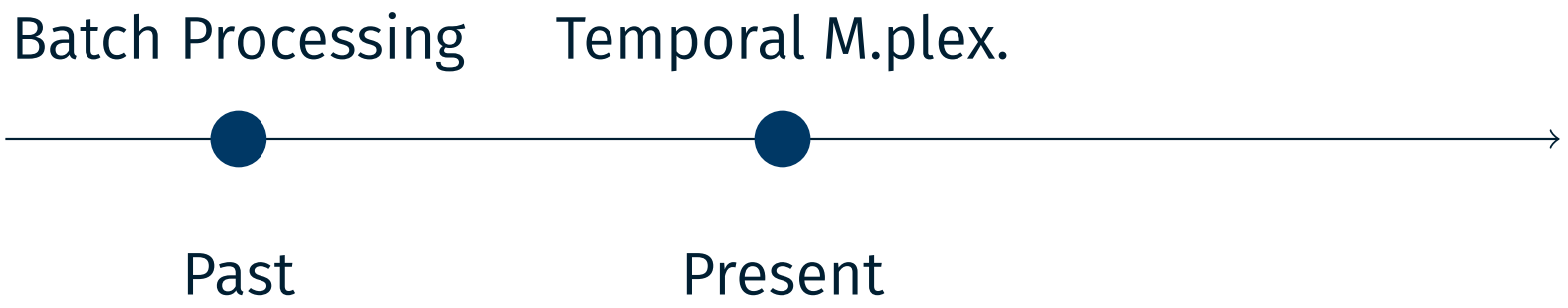
- One instance per tile
- Distributed state

Batch Processing

Past

Batch Processing

Sharing Resources

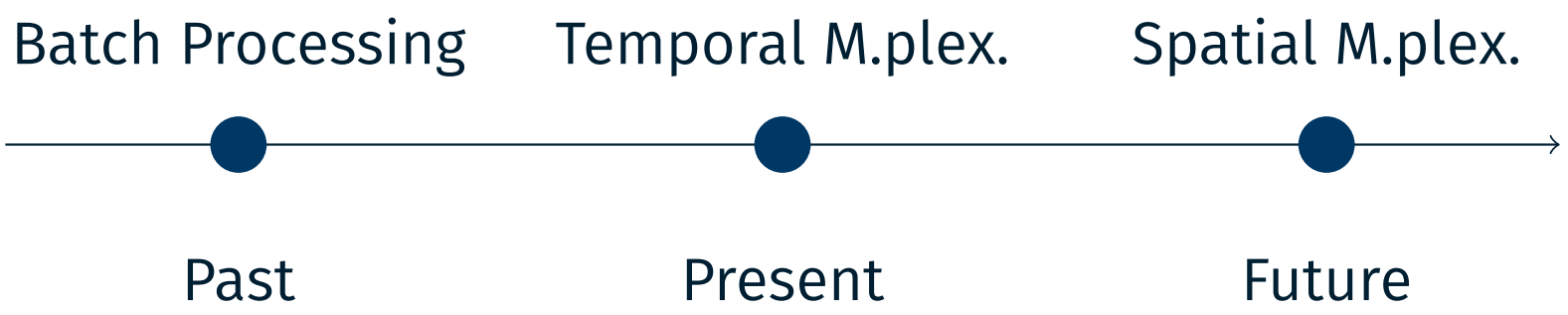


Temporal Multiplexing

- $N_{\text{Cores}} < N_{\text{Apps}}$
- Only solution if you have more applications than cores

$\Rightarrow$ Source of interference ☹

Sharing Resources

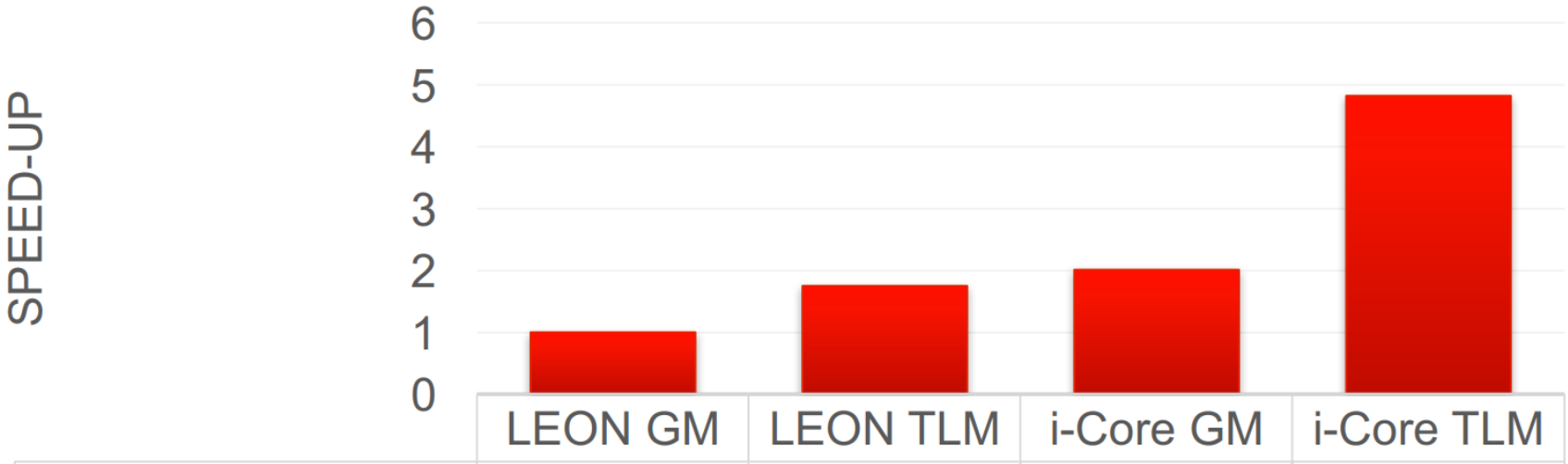


Spatial Multiplexing

- Ratio N_{Cores} to N_{Apps} will flip in the future
- Inherent part of OctoPOS's execution model
- Granting exclusive access to cores/resources
- Reduces side-channels

$\Rightarrow$ **Minimized** interference ☺

Hardware/Software Co-Design Example: *i*Core



	LEON GM	LEON TLM	i-Core GM	i-Core TLM
Speedup	1	1,75	2,01	4,82
Execution time [ms]	2049	1169	1017	425

X10 Tsunami simulation on the Invasic FPGA prototype using the *i*Core

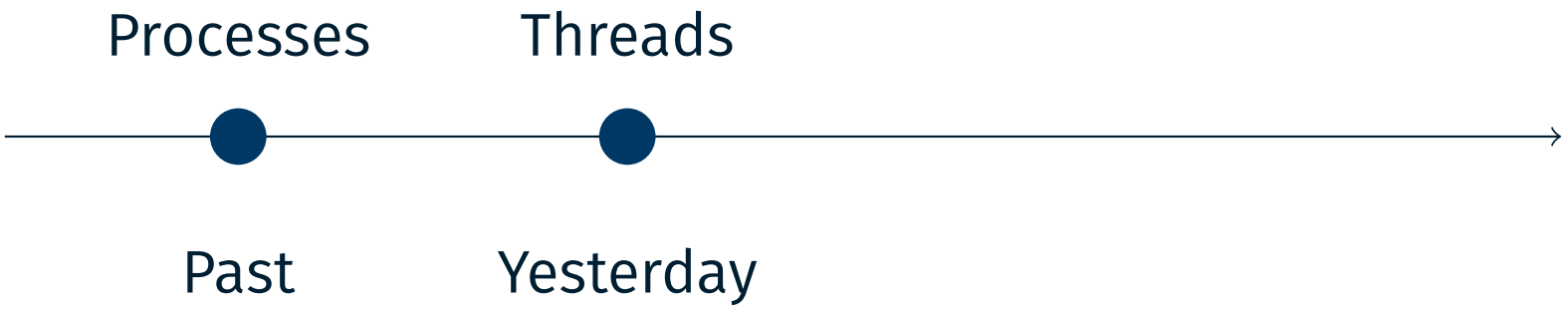
Alexander Pöppl, Marvin Damschen, Florian Schmaus, et al. “Shallow Water Waves on a Deep Technology Stack: Accelerating a Finite Volume Tsunami Model Using Reconfigurable Hardware in Invasive Computing”. In: *Euro-Par 2017: Parallel Processing Workshops*. Ed. by Dora B. Heras, Luc Bougé, Gabriele Mencagli, et al. Cham: Springer International Publishing, 2018, pp. 676–687. ISBN: 978-3-319-75178-8

Execution Model

Heavy-weight Processes

```
fork();
```

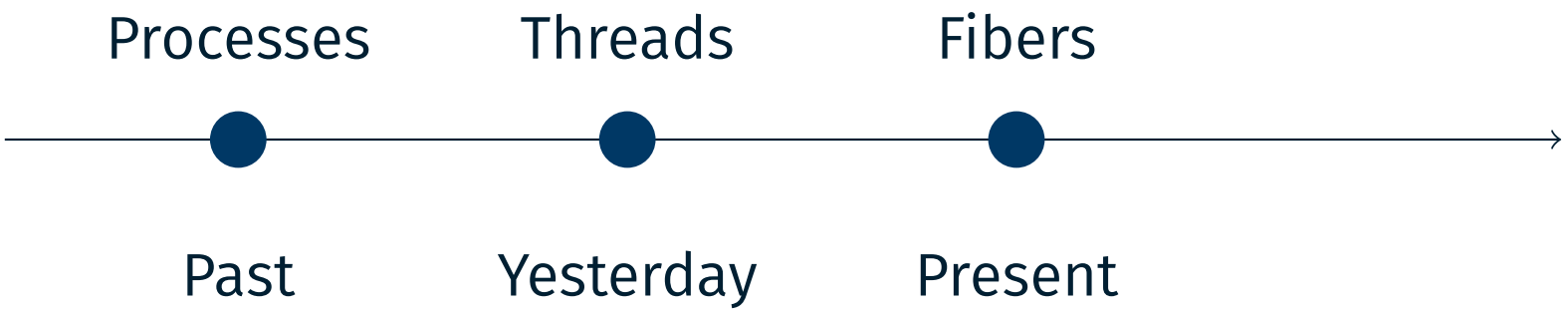
Execution Model



Light-weight Processes

```
pthread_create();
```

Execution Model

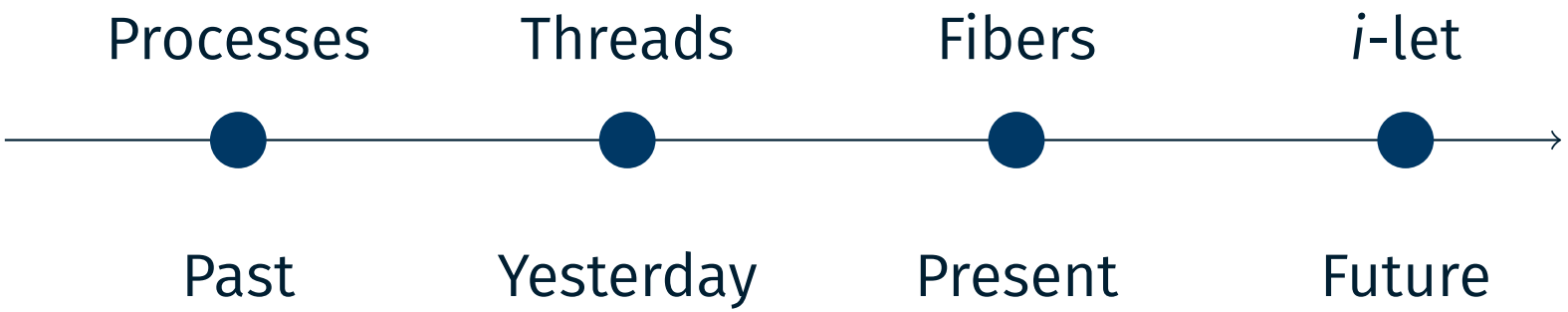


Feather-weight Processes

Concurrency Platforms: Go, Cilk, ...

```
for i := 0; i < 10; i++ {
  go f(i)
}
```

Execution Model

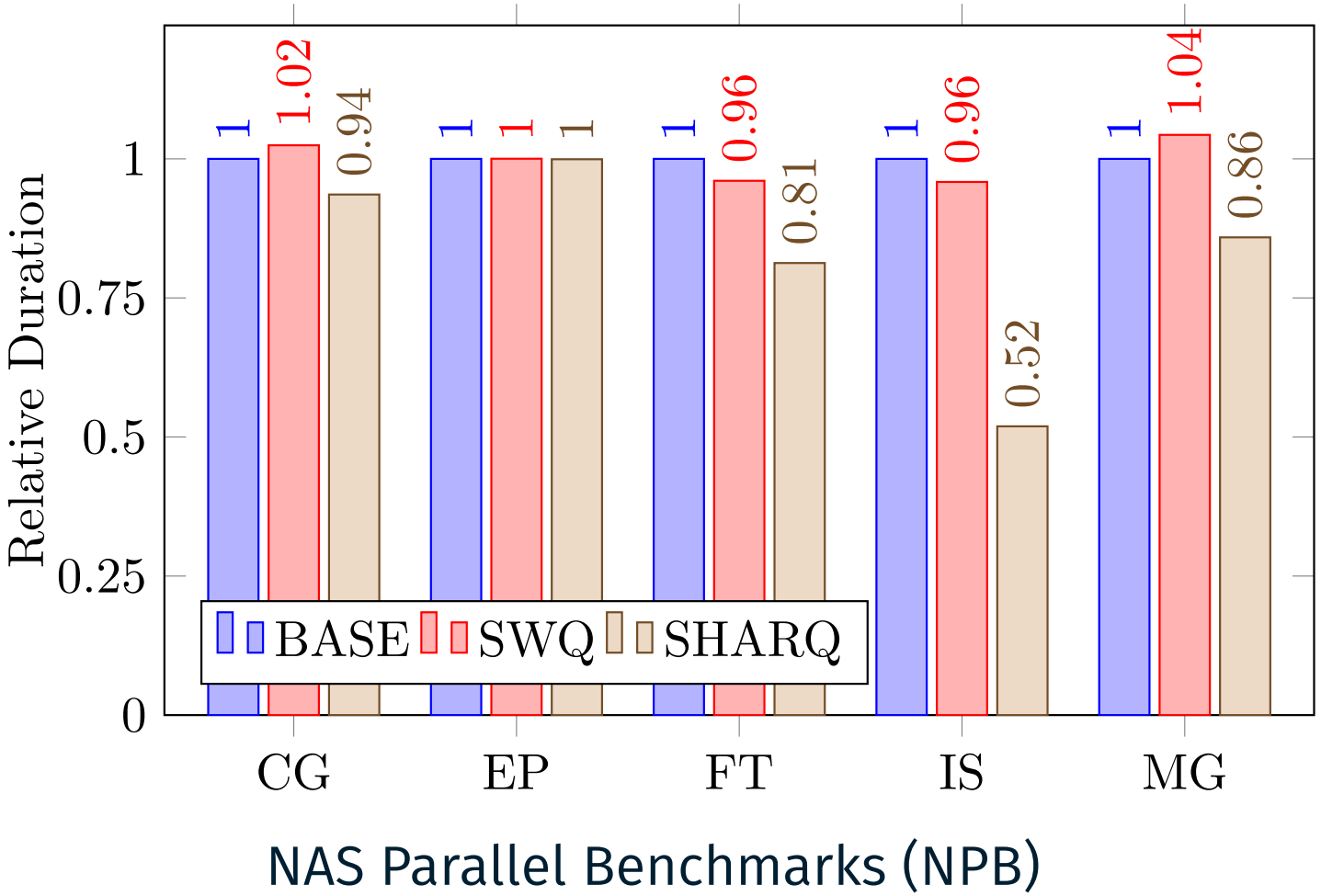


i-lets — OctoPOS's control-flow abstraction

- Run-to-completion
- Lazy context allocation
- Small footprint: 4 machine words
- **Every layer is *i*-let aware** (App, Runtime, OS, HW)

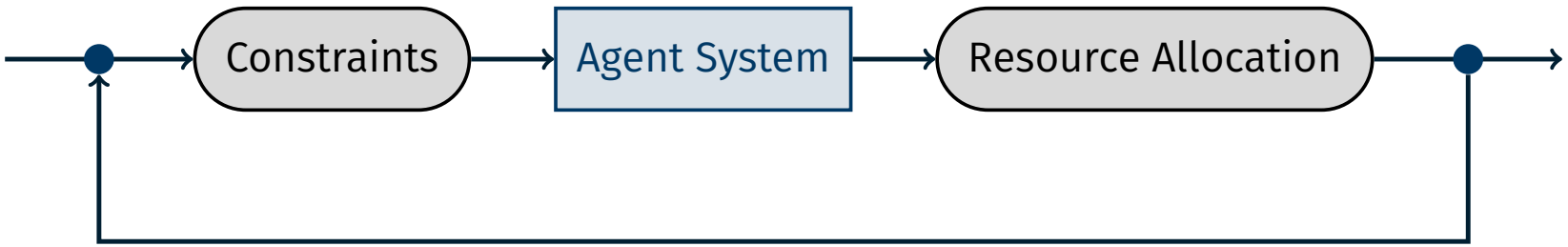
$\Rightarrow$ Allows massive µ-parallelism ☺

Hardware/Software Co-Design Example: SHARQ



Sven Rheindt, Sebastian Maier, Florian Schmaus, et al. "SHARQ: Software-Defined Hardware-Managed Queues for Tile-Based Manycore Architectures". In: *Proceedings of the 19th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. 2019

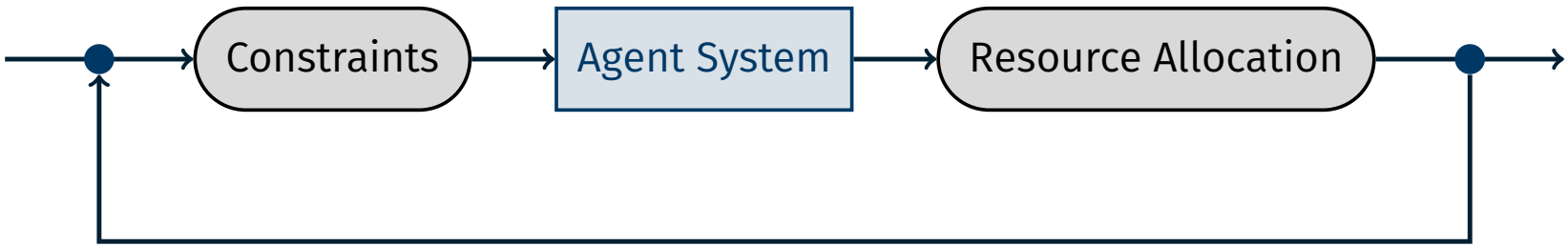
Dynamic Resource Management via Agent System



1. Gather Application Constraints

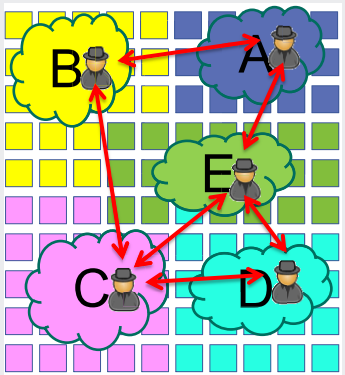
- Scalability curve(s)
- Desired accelerators
- Communication properties

Dynamic Resource Management via Agent System

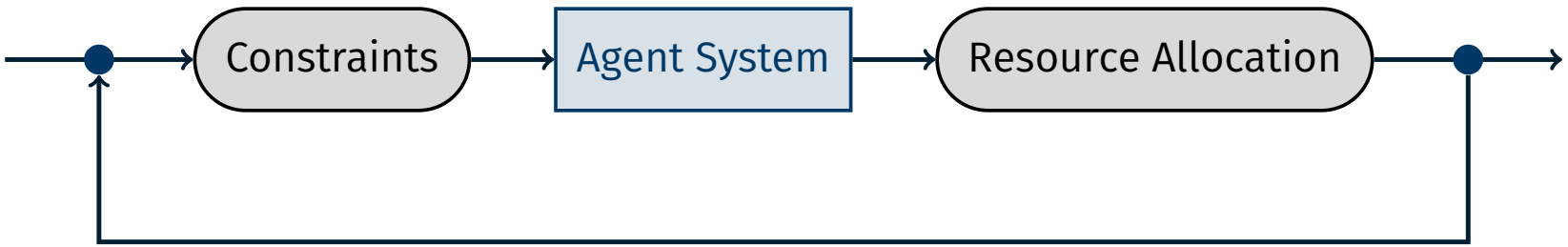


2. Agent System

- One agent per application
- Optmize resource assignment for all applications
- Distributed Constraint Optimization Problem (DCOP)
- Maximum Gain Message (MGM) (any-time algorithm)



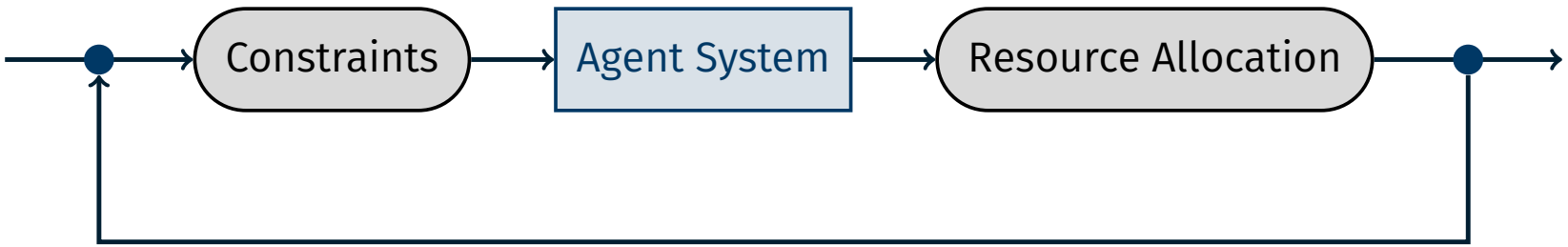
Dynamic Resource Management via Agent System



3. Resource Allocation

- Position/Amount
- Resources now available to application

Dynamic Resource Management via Agent System



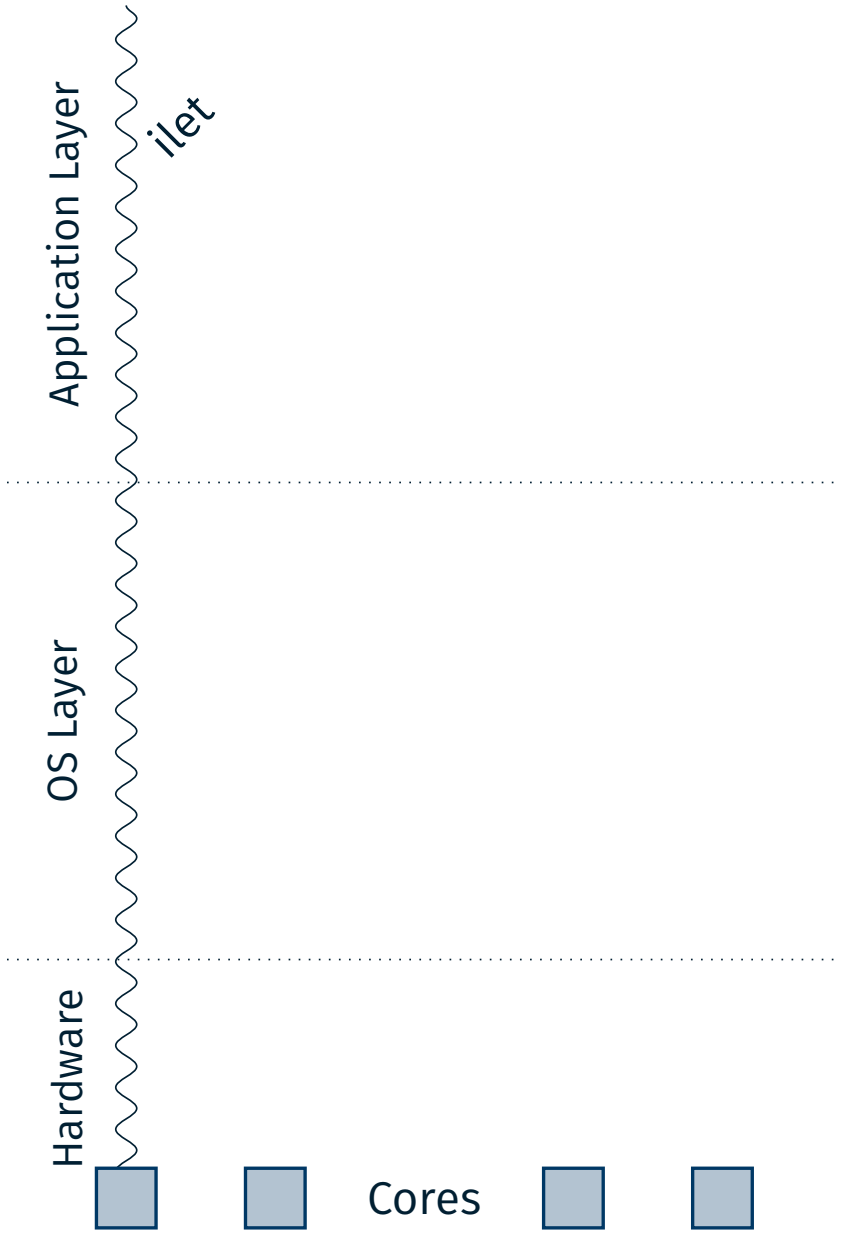
3. Resource Allocation

- Position/Amount
- Resources now available to application

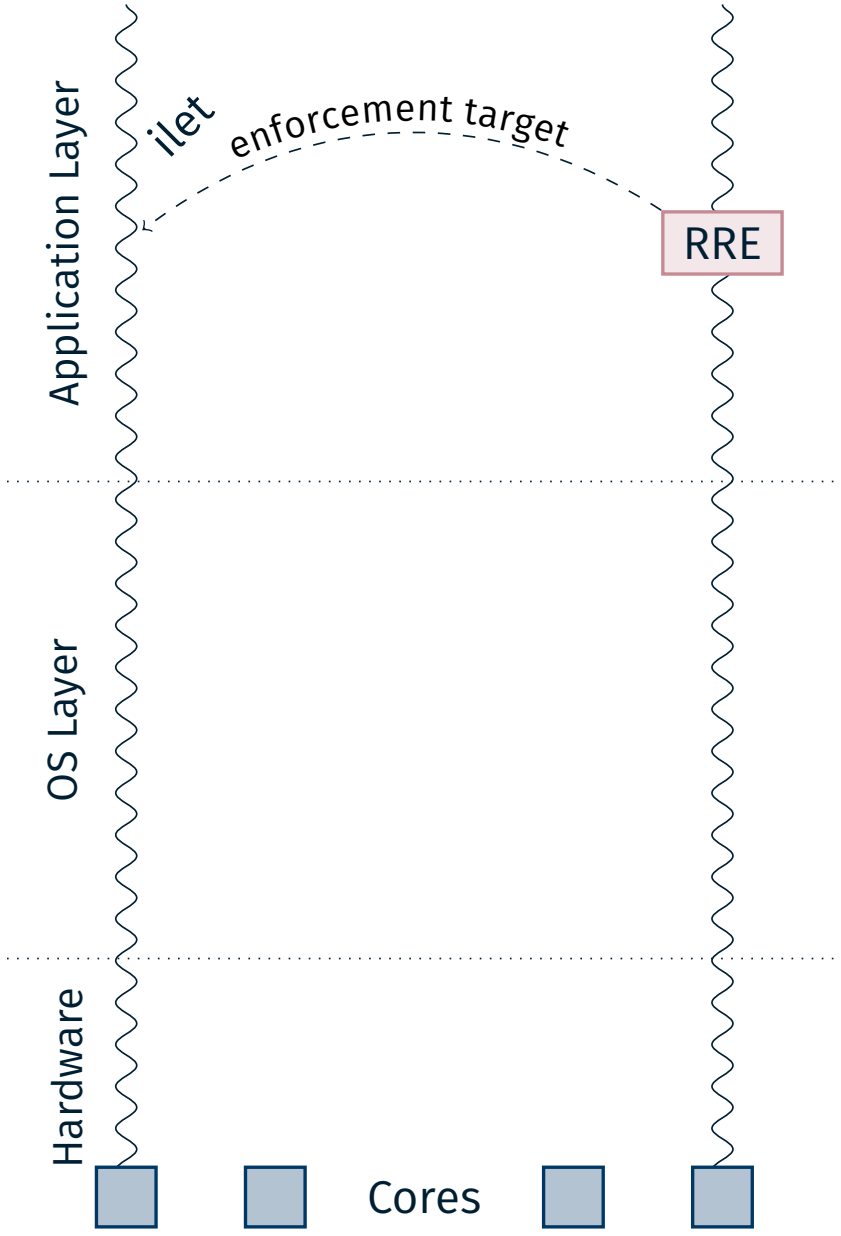
4. Repeat (on demand)

Application transitions into another phase

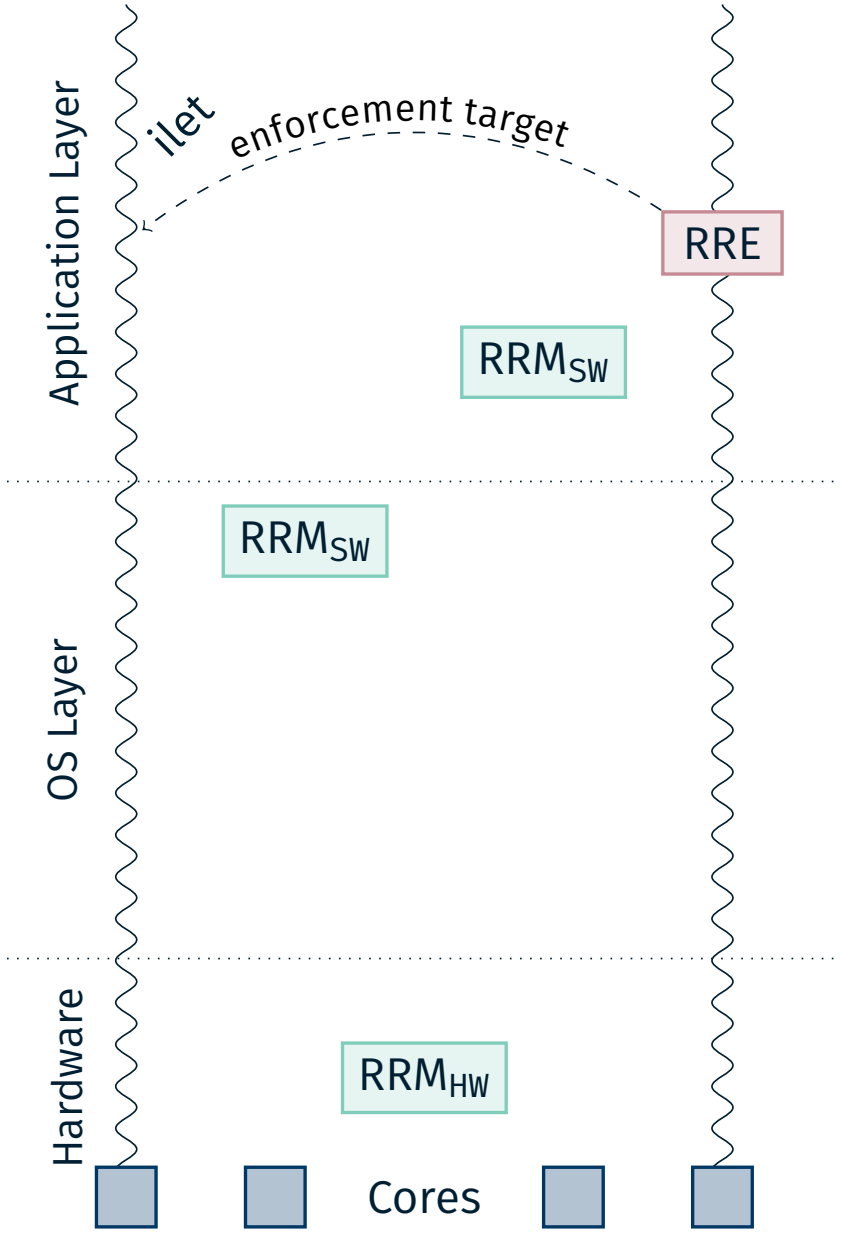
Runtime Requirement-Monitoring and Enforcement (RRM/RRE)



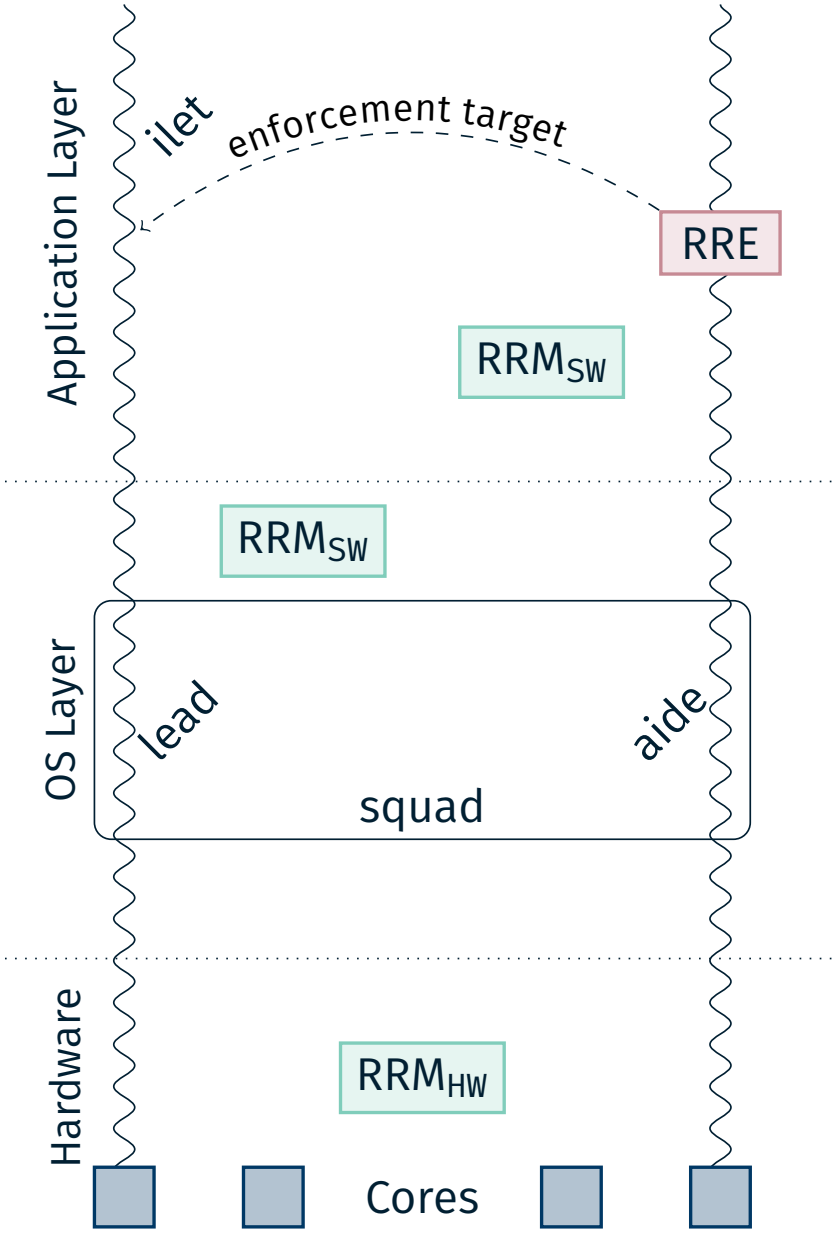
Runtime Requirement-Monitoring and Enforcement (RRM/RRE)



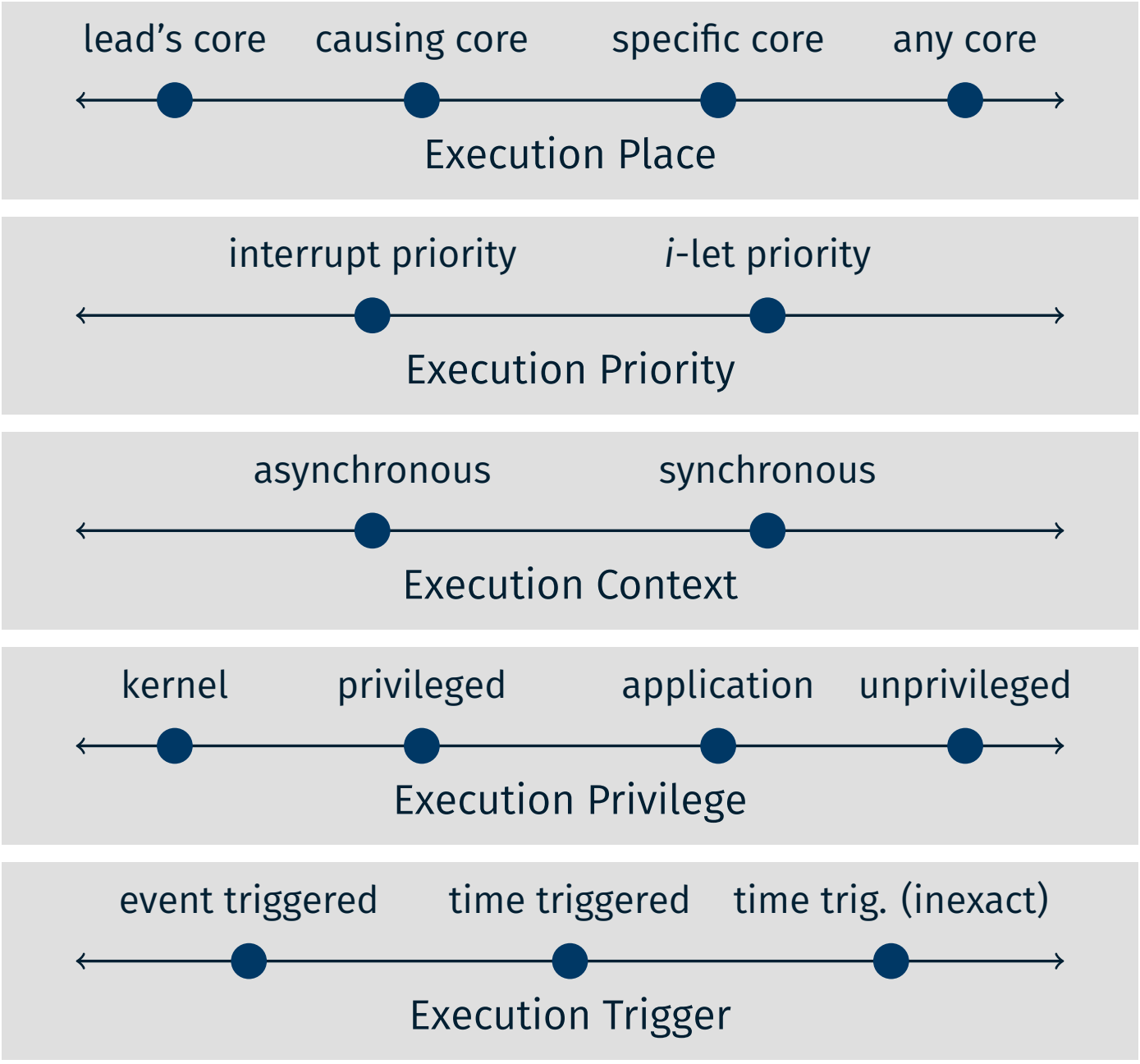
Runtime Requirement-Monitoring and Enforcement (RRM/RRE)



Runtime Requirement-Monitoring and Enforcement (RRM/RRE)



Runtime Requirement-Monitoring and Enforcement (RRM/RRE)



Conclusion

Satisfy *intrinsic* constraints/requirements Application C2

Counter *extrinsic* uncertainties Application C3

Many-* Architecture Hardware C1

Conclusion

Satisfy *intrinsic* constraints/requirements Application C2

- Agent System
- Explicit enumeration of constraints

✔

Counter *extrinsic* uncertainties Application C3

Many-⋆ Architecture Hardware C1

Conclusion

Satisfy *intrinsic* constraints/requirements Application C2 ✔

- Agent System
- Explicit enumeration of constraints

Counter *extrinsic* uncertainties Application C3 ✔

- RRM/RRE (Squads)

Many-∗ Architecture Hardware C1

Conclusion

Satisfy *intrinsic* constraints/requirements Application C2 ✔

- Agent System
- Explicit enumeration of constraints

Counter *extrinsic* uncertainties Application C3 ✔

- RRM/RRE (Squads)

Many-* Architecture Hardware C1 ✔

- Invasive Run-Time Support System (*i*RTSS)

Thank you for your attention.

Thank you for your attention.

Questions?