Timed Functional Simulation and Interference Analysis of Mixed-Criticality Applications

Philipp A. Hartmann, Philipp Ittershagen, Kim Grüttner

Hardware/Software Design Methodology Group, R&D Division Transportation
OFFIS - Institute for Information Technology, Oldenburg, Germany
{philipp.hartmann, philipp.ittershagen, kim.gruettner}@offis.de

Motivation

Systems are increasingly complex, both in terms of
- **application** features and complexity
- target **architectures**, and their constraints

Both **hardware and software architectures** have to be considered already in early design phases.

Design exploration
- Software deployment and task mapping
- Number of processing elements (PEs) & cores
- Task priorities, periods, deadlines, scheduling policy, resource access protocols, …

Interference analysis
- Data dependent task workload model with computation times, shared memory read and write accesses
- Explicit communication and synchronisation between Tasks
- Physical communication medium transport delays and arbitration

System validation and refinement requires **fast simulation** of early system models.

Application-Level Parallelism

Always **simulate task until shared resource access**
- even **across periods**
- **collect local trace** according to task FSM

Concurrent accesses are arbitrated.

Honour **guard conditions** in evaluation as well
→ **ignore** earlier **access**, **iff blocked** by guard.
- **update blocking times** after decoupled access

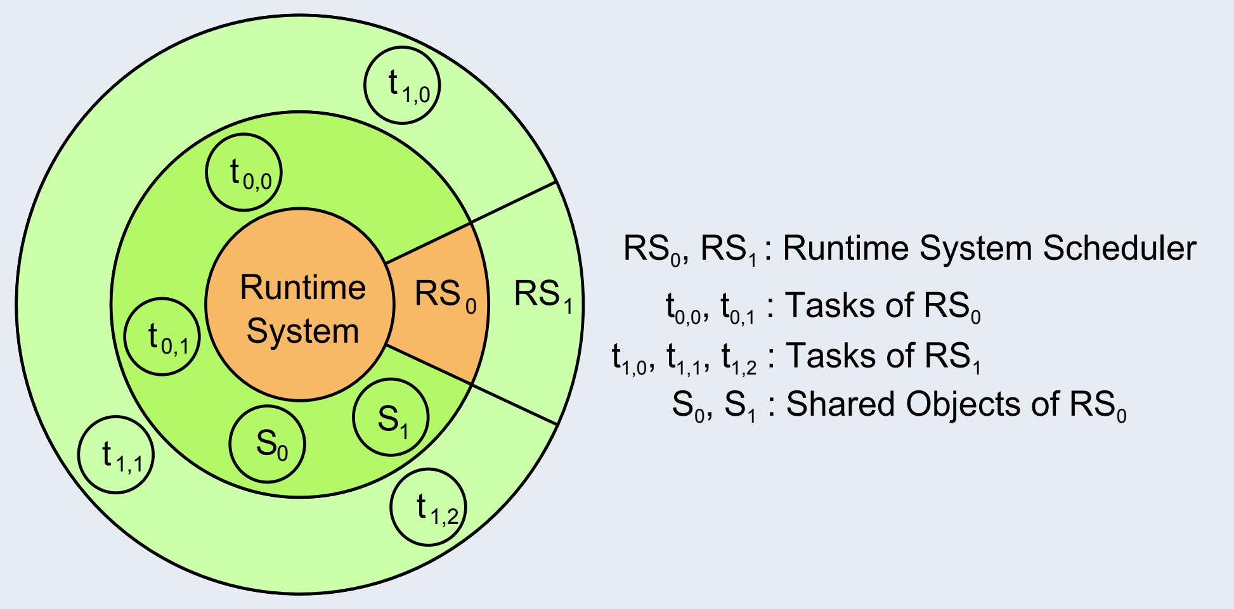
Mixed Criticality Model

Finite set of Applications A_i with
- criticality level L_i
- with set of Tasks T_i
- with set of Shared Objects S_i

Each Task t_j in T_i is defined by (P_j, D_j, C_j, SI_j, L_j) with
- period (minimum arrival time) P
- deadline D
- workload and memory access graph C
- ports to Shared Object Interfaces SI in S_i.I
- criticality level L

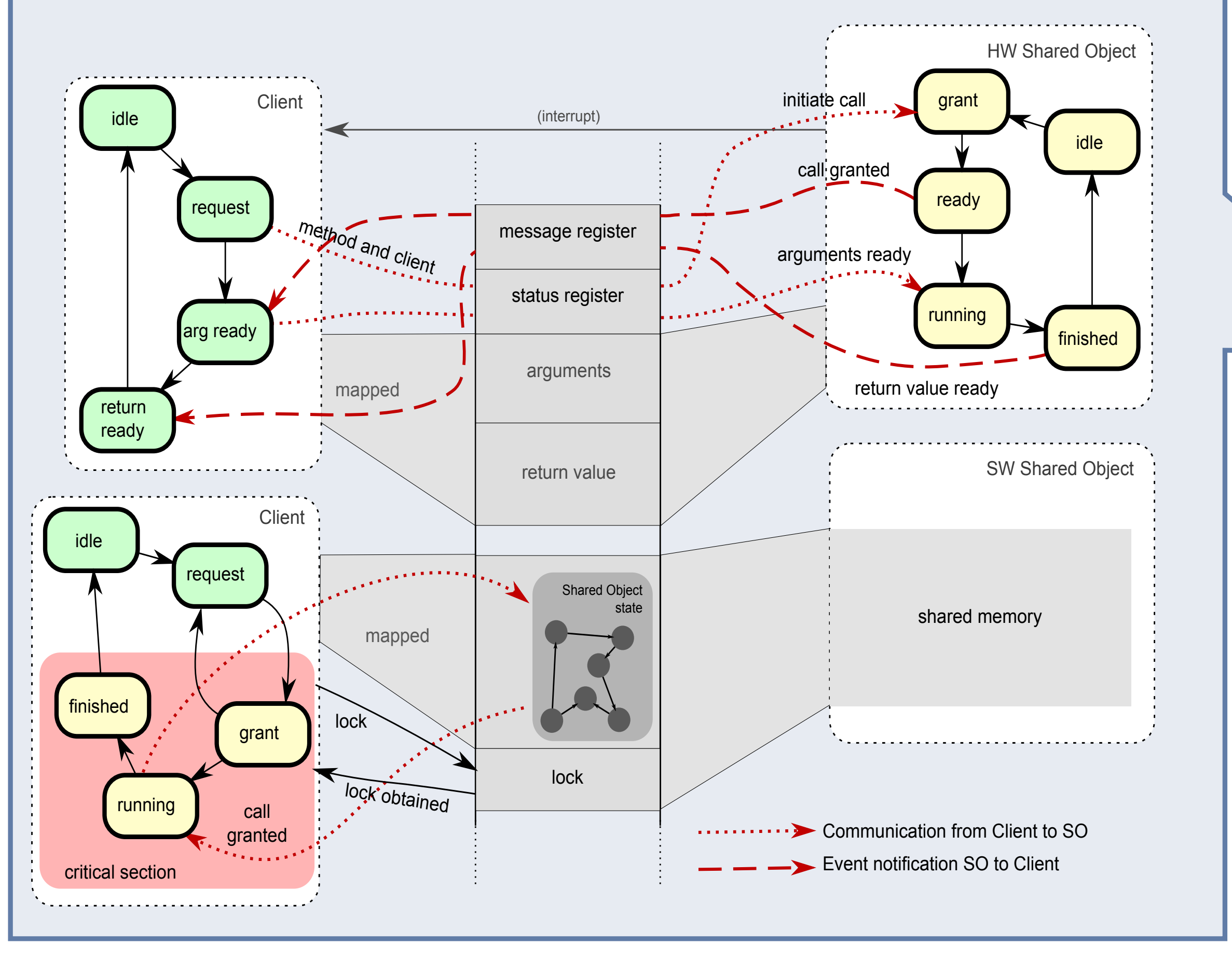
Each Shared Object S_i consistst of
- a set of Interfaces I_i with methods m_j in i_k in Ii (let M_i be the union of all methods in I_i)
- a set of side effect free Guards G_i
- a set of guarded methods GM_i in M_i x G_i implementing all interfaces methods M_i
- a shared resource access arbitration policy

Hierarchical Scheduling

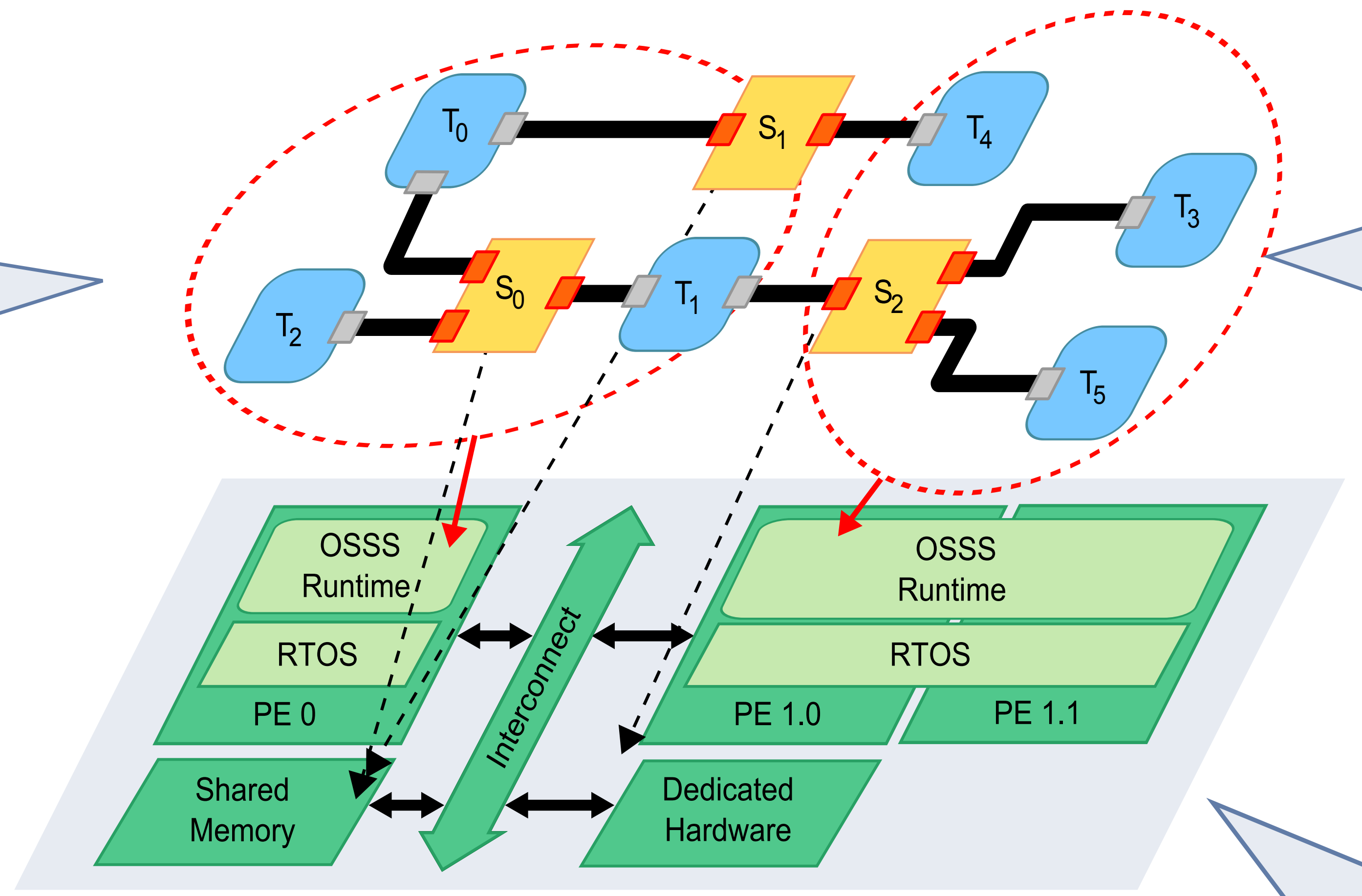


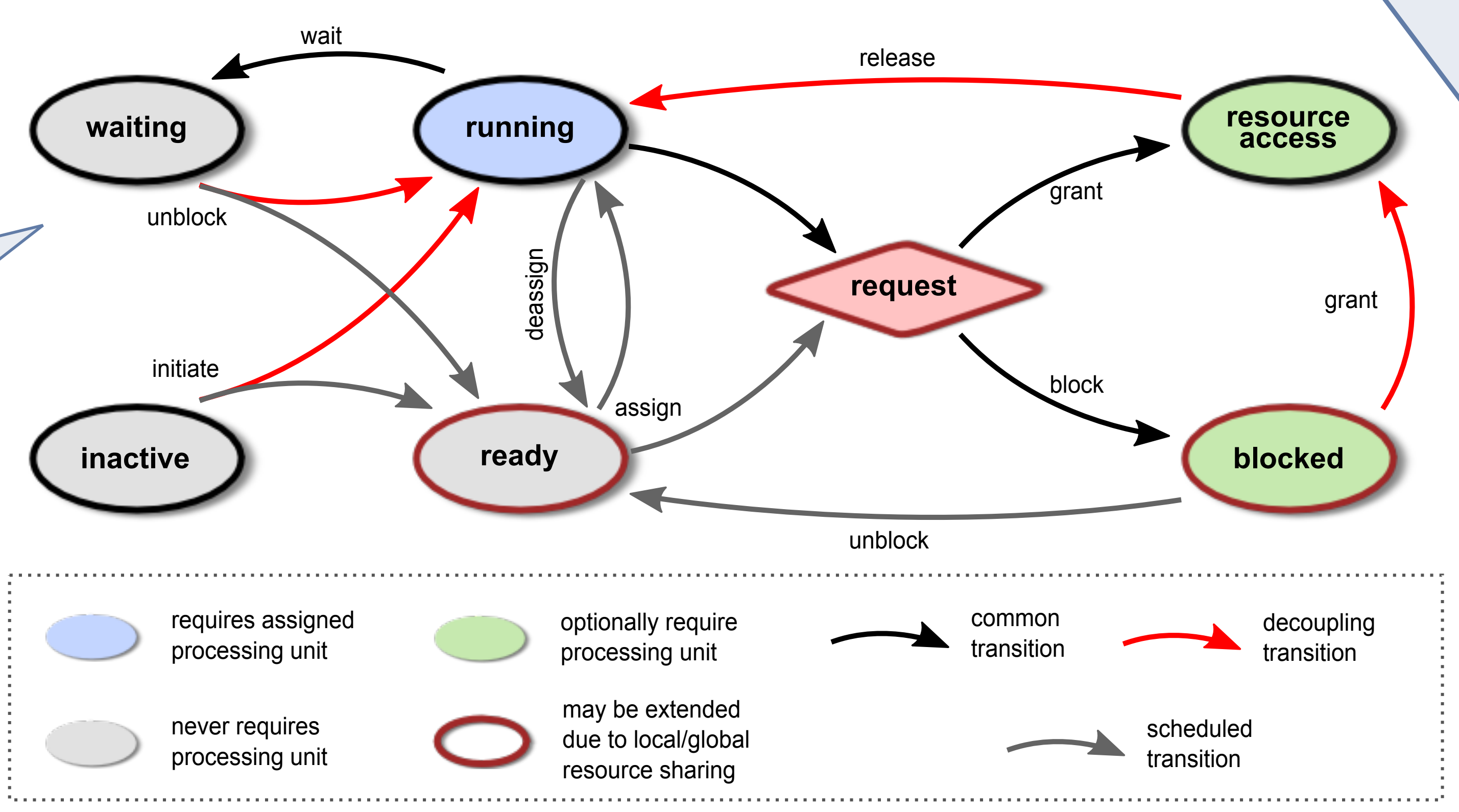
RS_0, RS_1 : Runtime System Scheduler
$t_{0,0}$, $t_{0,1}$: Tasks of RS_0
$t_{1,0}$, $t_{1,1}$, $t_{1,2}$: Tasks of RS_1
S_0, S_1 : Shared Objects of RS_0

HW & SW Shared Object Implementation



Modelling and Refinement for Mixed-Criticality Applications





Application Layer

OSSS (*Oldenburg System Synthesis Subset*) is a **C++ and SystemC-based** simulation environment.

Modelling starts on Application Layer, with an **executable, functional specification**.

System consists of **Tasks (T)** communicating via **Shared Objects (S)**.

Shared Objects enable high-level, method-based, communication via user-defined transactions.

Task dependencies can be expressed explicitly and implicitly through Shared Object synchronisation.

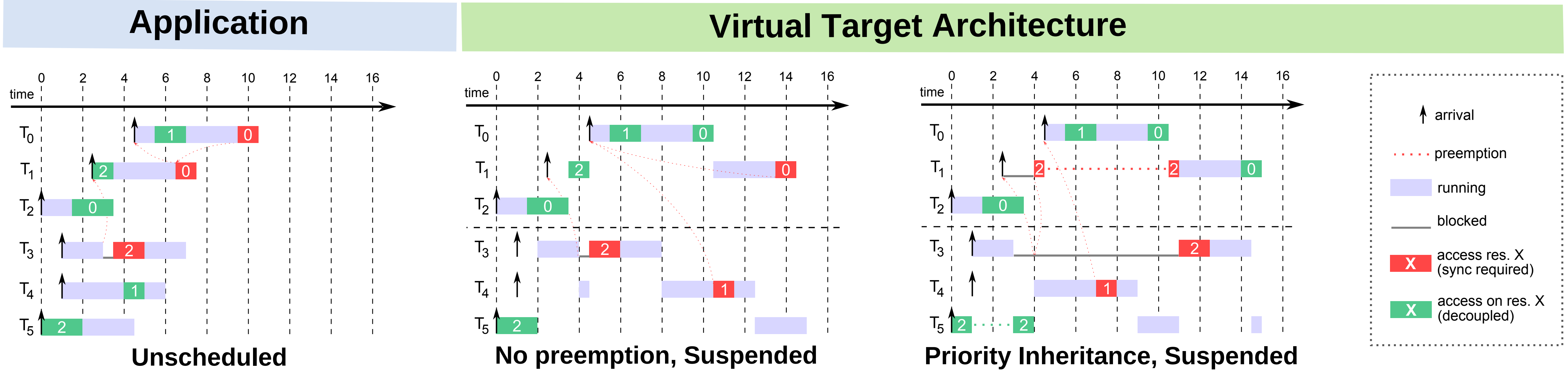
Virtual Target Architecture (VTA) Layer

During **refinement**, components of Application Layer are mapped to Virtual Target Architecture:
- Tasks onto runtime sytems
- Shared Objects to dedicated hardware blocks or shared memory

Scheduling of associated tasks
- **Task state** management ensures, that only one task per PE is *running* at a given time.
- Activation of **periodic tasks**, deadline observation.
- Time synchronisation and modelling of **preemption**.

Order of accesses
- to single shared resource have to stay the same!
- to different resources may differ, without impairing functional correctness.

Local Guarantee
- Accessing task is not preempted/delayed by other local tasks on the same processing unit.

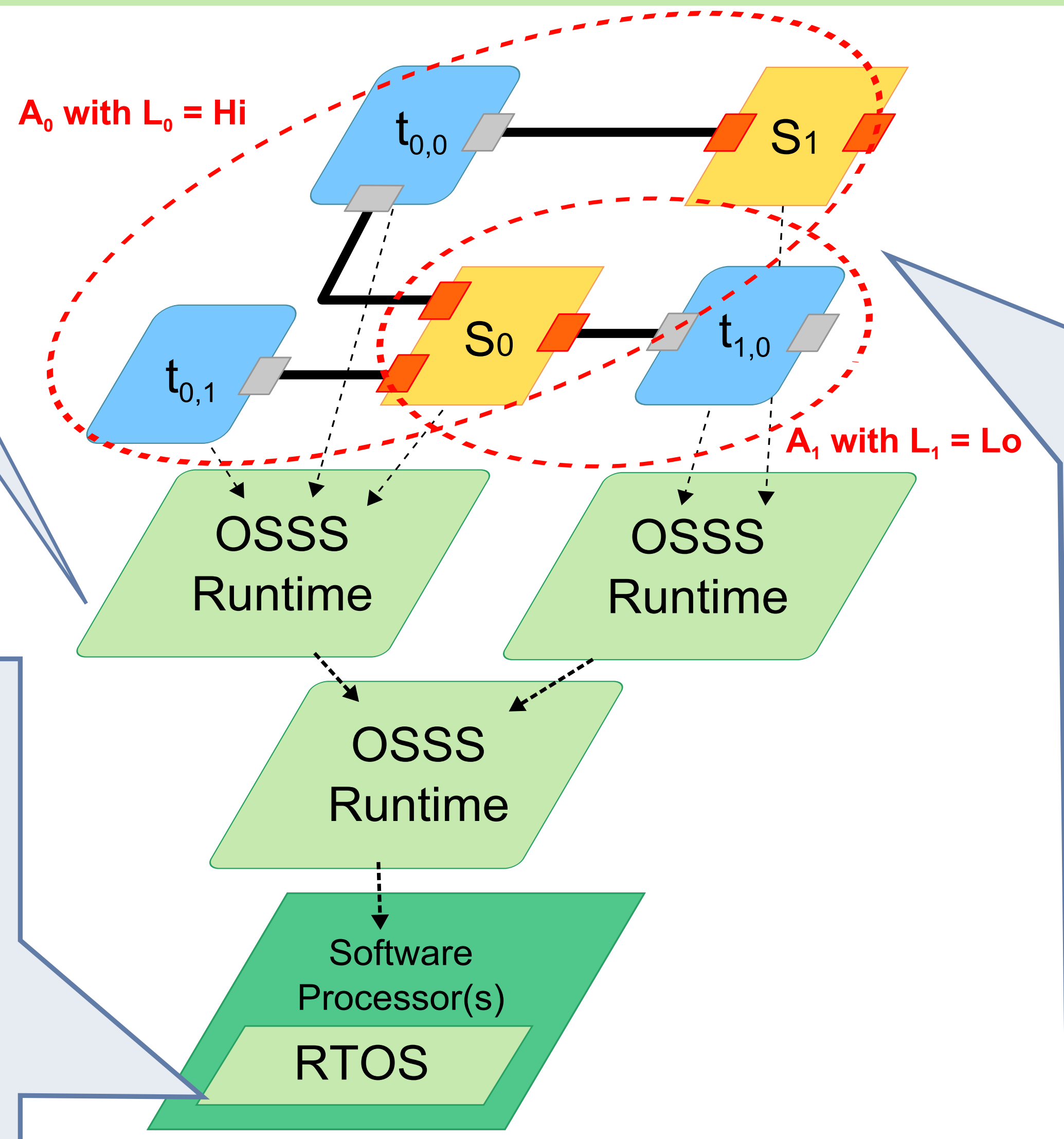


Unscheduled

No preemption, Suspended

Priority Inheritance, Suspended

Goal: Timed Functional Simulation and Interference Analysis

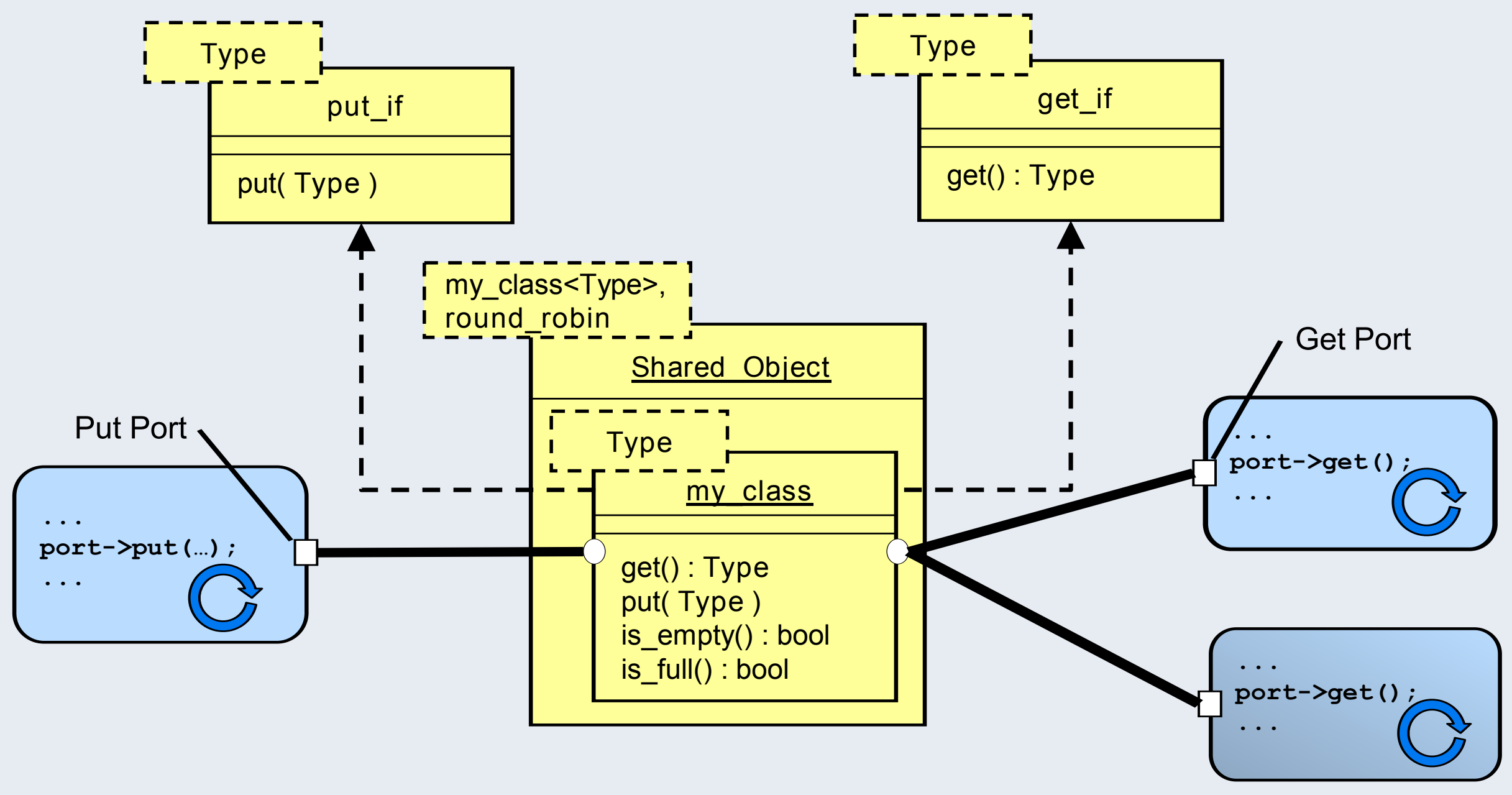


Explicit communication via Shared Objects

Task **communication/synchronisation** modelled **explicitly** via Shared Objects.
- User-defined, **method-based interfaces**.
- User-defined **scheduler** and **resource access protocol.**

Mutual exclusive access ensures consistency.

Guard conditions
- or *guarded methods*
- Boolean conditions, depending on object's state
- Can block a caller, until condition holds



References and further readings

[1] Philipp A. Hartmann, Philipp Reinkemeier, Henning Kleen and Wolfgang Nebel: **Efficient modelling and simulation of embedded software multi-tasking using SystemC and OSSS.** *Forum on Specification, Verification and Design Languages (FDL), 2008*

[2] Philipp A. Hartmann, Kim Grüttner, Achim Rettberg and Ina Podolski: **Distributed Resource-Aware Scheduling for Multi-Core Architectures with SystemC.** *7th IFIP Conference on Distributed and Parallel Embedded Systems (DIPES), 2010*

[3] Philipp A. Hartmann, Kim Grüttner, Philipp Ittershagen and Achim Rettberg: **A Framework for Generic HW/SW Communication using Remote Method Invocation.** *The 2011 Electronic System Level Synthesis Conference (ESLsyn), 2011*

[4] Philipp Reinkemeier, Philipp Ittershagen, Ingo Stierand, Philipp A. Hartmann, Stefan Henkler and Kim Grüttner: **Seamless Segregation for Multi-Core Systems.** *OFFIS Technical Report, 2013*

[5] Philipp Ittershagen, Philipp A. Hartmann, Kim Grüttner and Achim Rettberg: **Hierarchical Real-Time Scheduling in the Multi-Core Era - An Overview.** *The Fourth IEEE Workshop on Self-Organizing Real-Time Systems (SORT) 2013*