OSSS/MC — A Mixed-Criticality Programming Model

EMC² WP2: Executable Application Models & Design Tools

Verkehr
Transportation

Philipp Ittershagen
pit@offis.de
OFFIS—Institute for Information Technology
R&D Division Transportation

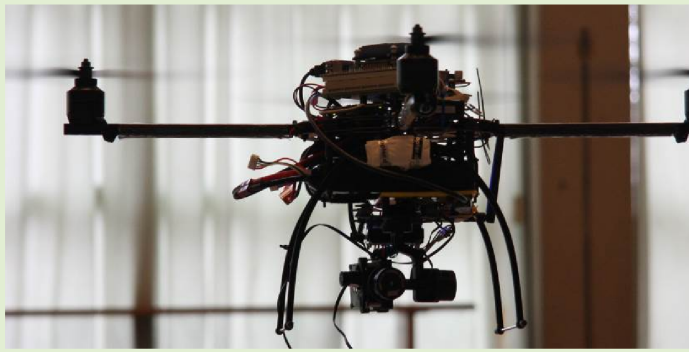
January 24, 2017
EMC² Workshop @HiPEAC'2017
Stockholm, Sweden

1 Mixed-Criticality Embedded Systems

Motivation

- **Mixed criticality**: combining applications/functions with different real-time requirements
  - **Safety**-critical: quadrocopter flight controller
  - **Performance**-critical: video-based object detection
- Goal: Meet **cost, size, and power consumption** constraints

Quadrocopter System

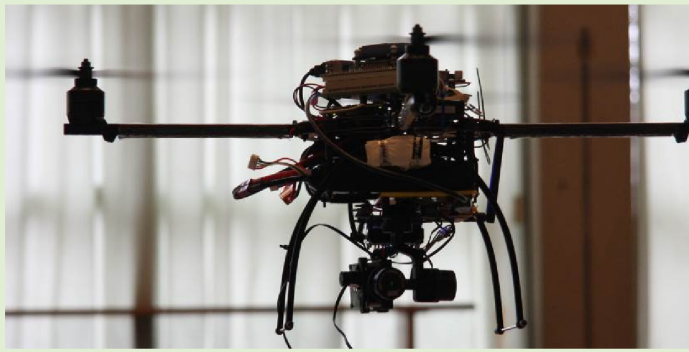


1 Mixed-Criticality Embedded Systems

Motivation

- **Mixed criticality**: combining applications/functions with different real-time requirements
  - **Safety**-critical: quadrocopter flight controller
  - **Performance**-critical: video-based object detection
- Goal: Meet **cost, size, and power consumption** constraints
- Possible solution: Integrate system on a single, **powerful MPSoC**

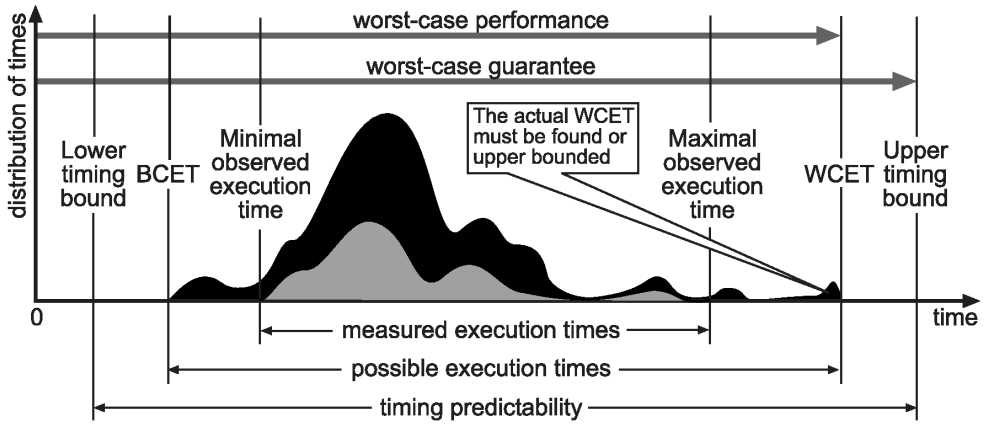
Quadrocopter System



2 Platform Complexity & Execution Time Analysis

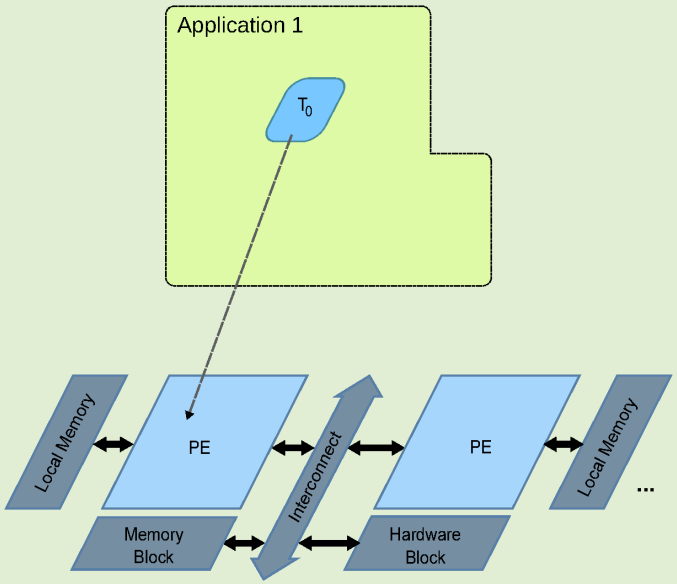
Motivation

- Platform complexity → WCET analysis uncertainty
- Increasing gap between **actual** and **upper-bounded worst-case** execution times



Source: R. Wilhelm[4]

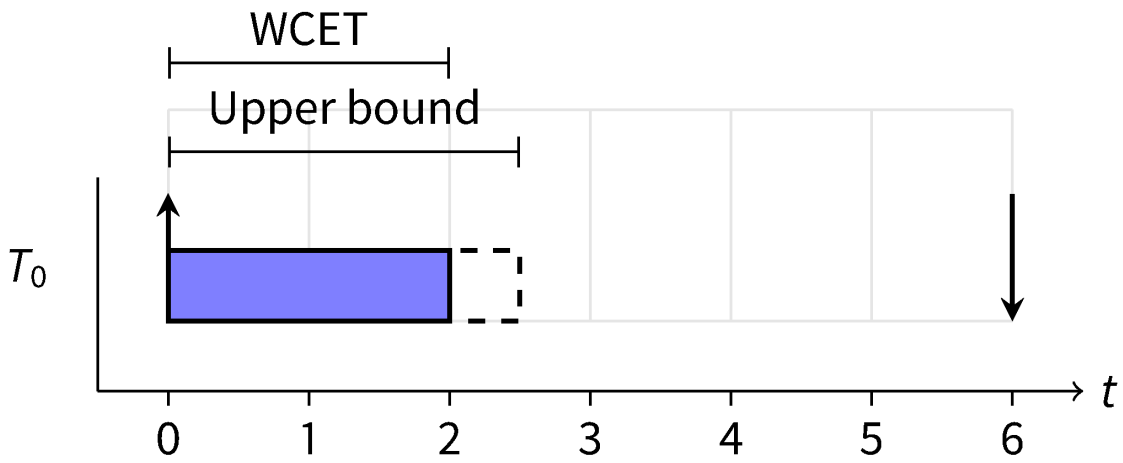
Example Task Mapping



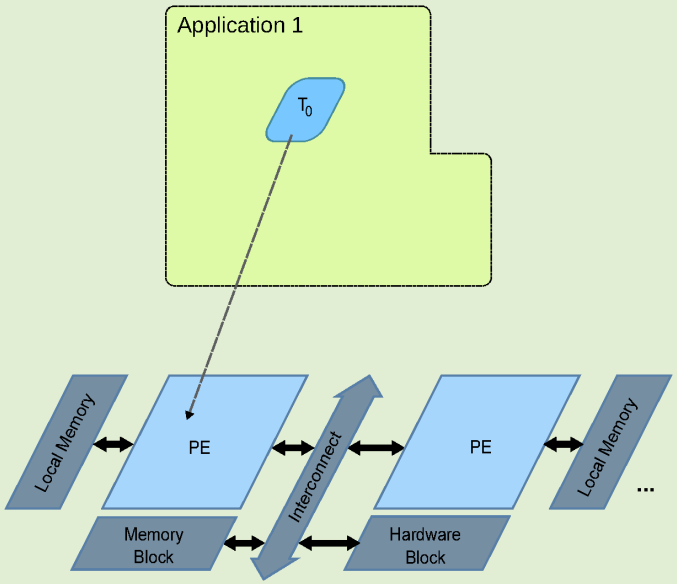
2 Platform Complexity & Execution Time Analysis

Motivation

- Platform complexity → WCET analysis uncertainty
- Increasing gap between **actual** and **upper-bounded worst-case** execution times



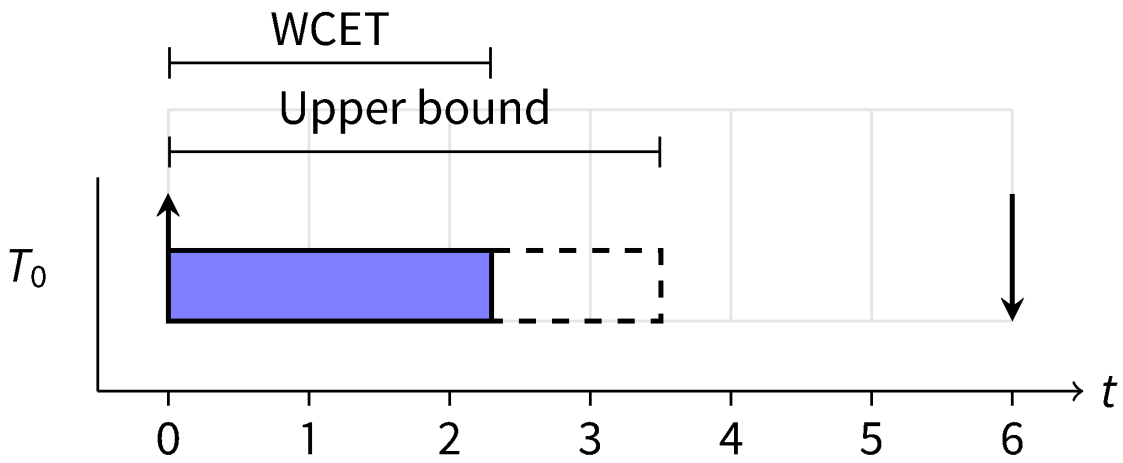
Example Task Mapping



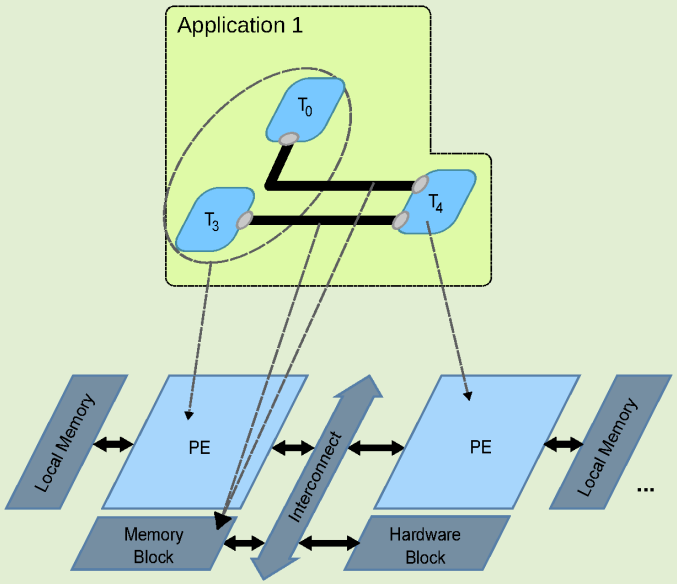
2 Platform Complexity & Execution Time Analysis

Motivation

- Platform complexity → WCET analysis uncertainty
- Increasing gap between **actual** and **upper-bounded worst-case** execution times



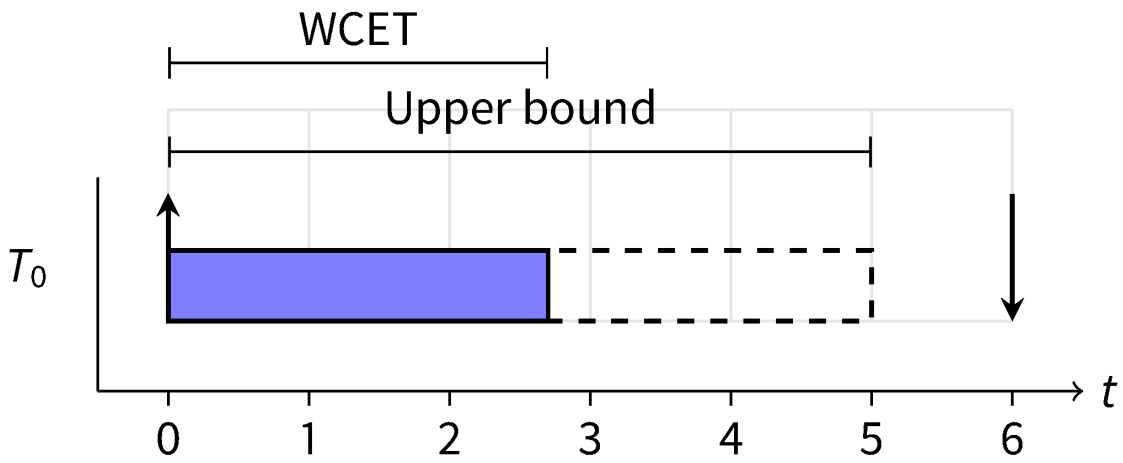
Example Application Mapping



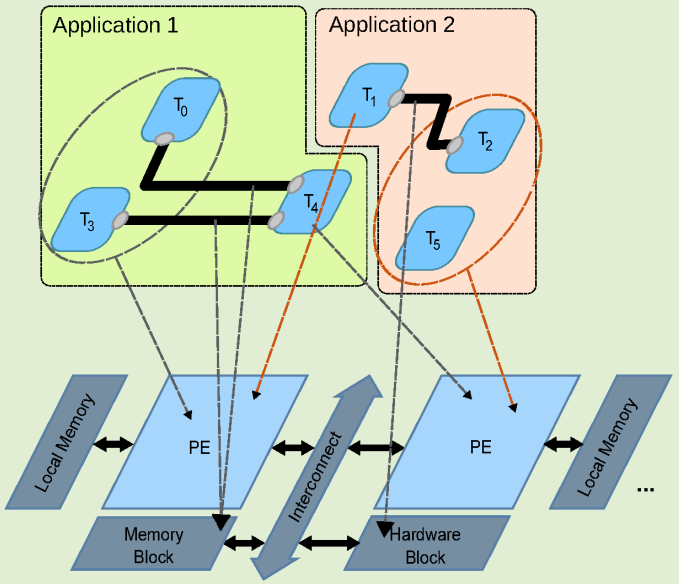
2 Platform Complexity & Execution Time Analysis

Motivation

- Platform complexity → WCET analysis uncertainty
- Increasing gap between **actual** and **upper-bounded worst-case** execution times



Example Application Mapping

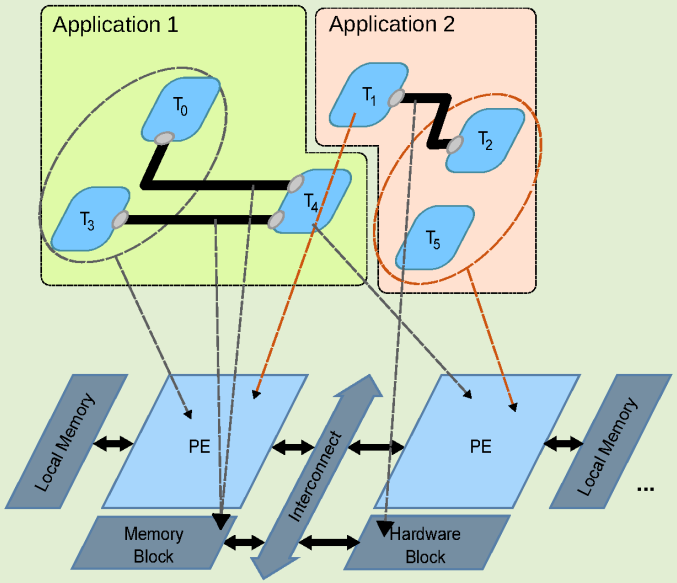


3 Application-Level Interferences

Motivation

- Prevent **spatial** interference
  - Spatial isolation of application contexts
  - Memory **protection** mechanisms

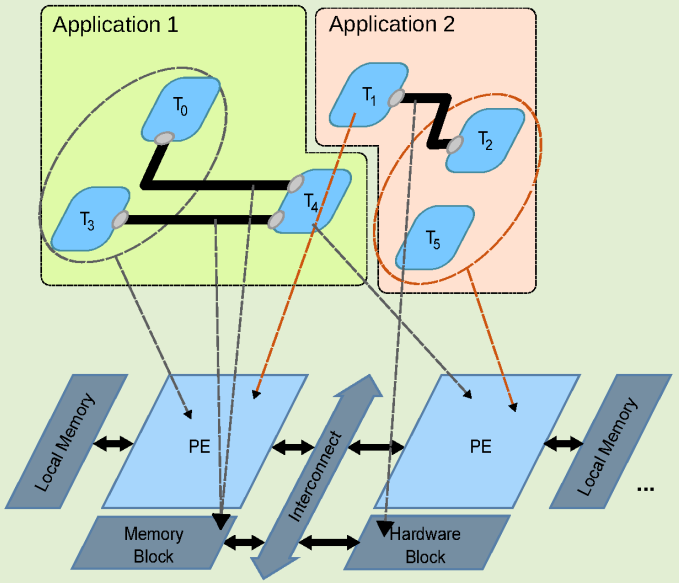
Applications on an MPSoC



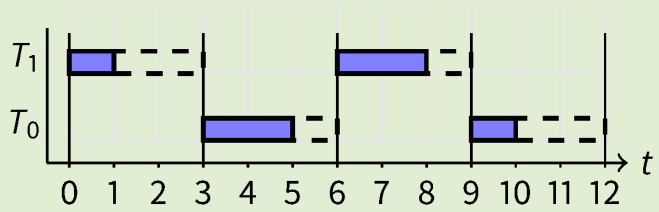
3 Application-Level Interferences
Motivation

- Prevent **spatial** interference
  - Spatial isolation of application contexts
  - Memory **protection** mechanisms
- Enforce **temporal** isolation
  - Fixed **time slots** at **design time**
  - Switch application context between slots

Applications on an MPSoC



Temporal Isolation

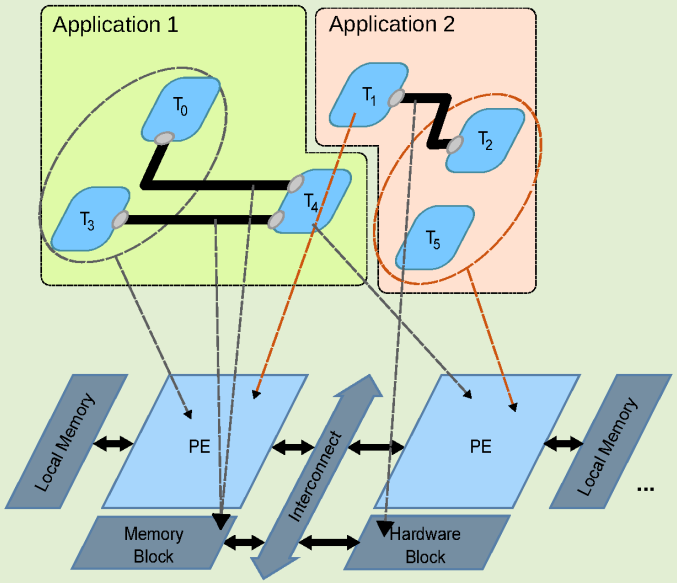


4 Execution Time Uncertainty & Temporal Isolation

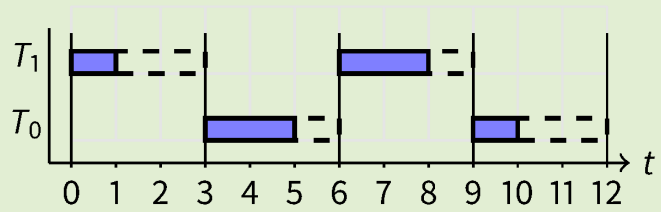
Motivation

- Increasing gap between **actual** and **upper-bounded** worst-case **execution time** on complex MPSoC's
- Static time-slice allocation leads to **reduced** average resource **utilisation**
- Mixed-Criticality: **different real-time requirements** of safety/performance-critical functions

Applications on an MPSoC



Temporal Isolation



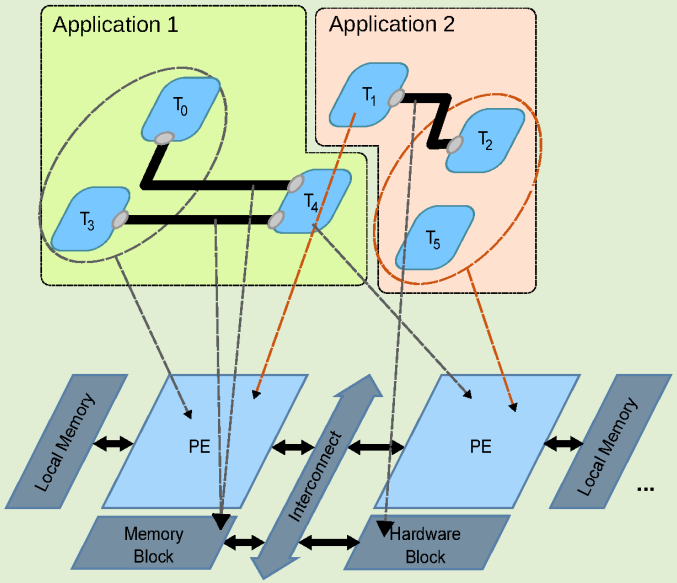
4 Execution Time Uncertainty & Temporal Isolation

Motivation

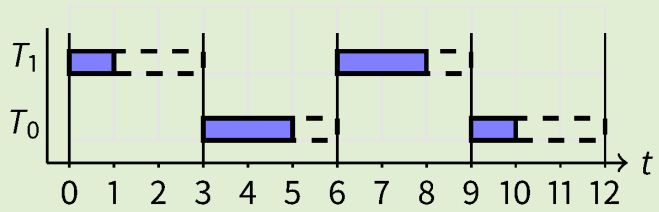
- Increasing gap between **actual** and **upper-bounded** worst-case **execution time** on complex MPSoC's
- Static time-slice allocation leads to **reduced** average resource **utilisation**
- Mixed-Criticality: **different real-time requirements** of safety/performance-critical functions

How can we allocate any unused resources to the performance-critical function and evaluate the impact at design time?

Applications on an MPSoC



Temporal Isolation



5 Combining Mixed-Criticality & Execution Time Uncertainties

Motivation

Define **utilisation profiles/scenarios** and let the performance-critical function **adapt its behaviour** accordingly.

5 Combining Mixed-Criticality & Execution Time Uncertainties

Motivation

Define **utilisation profiles/scenarios** and let the performance-critical function **adapt its behaviour** accordingly.

1. **Static:** safety-critical function behaves as **statically determined** (upper-bounded execution time)
   $\rightarrow$ fewer resources for performance-critical function

5 Combining Mixed-Criticality & Execution Time Uncertainties

Motivation

Define **utilisation profiles/scenarios** and let the performance-critical function **adapt its behaviour** accordingly.

1. **Static:** safety-critical function behaves as **statically determined** (upper-bounded execution time)
   $\rightarrow$ fewer resources for performance-critical function
2. **Dynamic:** safety-critical function uses **a fraction** of the upper-bounded execution time
   $\rightarrow$ more resources available for performance-critical function

5 Combining Mixed-Criticality & Execution Time Uncertainties

Motivation

Define **utilisation profiles/scenarios** and let the performance-critical function **adapt its behaviour** accordingly.

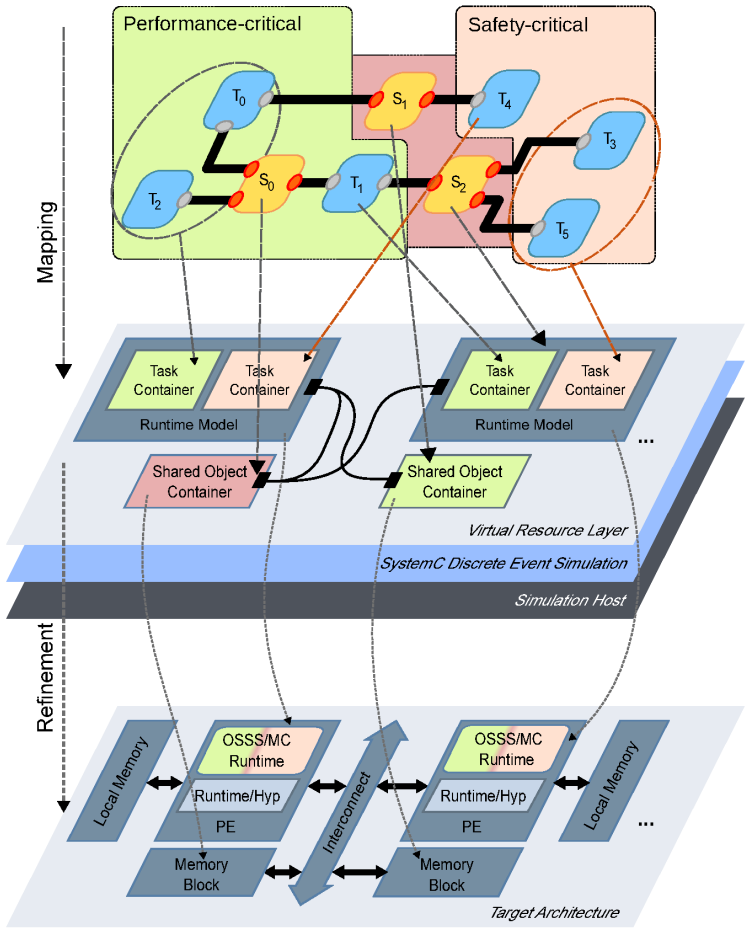
1. **Static:** safety-critical function behaves as **statically determined** (upper-bounded execution time)
   $\rightarrow$ fewer resources for performance-critical function
2. **Dynamic:** safety-critical function uses **a fraction** of the upper-bounded execution time
   $\rightarrow$ more resources available for performance-critical function

- **Determine** safety-critical resource utilisation by **execution time monitoring**
- Provide different performance-critical functional behaviours depending on the current **utilisation profile/scenario**

6 OSSS/MC Methodology Overview

OSSS/MC Methodology & Modelling Components

1. Mixed-Criticality **Programming Model**
   - Based on **SystemC/C++** methodology OSSS
   - Define functional and extra-functional **behaviour**
2. Mixed-Criticality **Resource Model**
   - Provide operational semantics (scheduling & resource access)
   - Explore mixed-critical application **behaviour** and **mapping** decisions
3. Mixed-criticality **Execution Environment**
   - Proof-of-Concept Implementation based on a **commercial MPSoC**
   - Hypervisor for **application-level** context switch
   - **Monitor** resource usage behaviour



6 OSSS/MC Methodology Overview

OSSS/MC Methodology & Modelling Components

1 Mixed-Criticality **Programming Model**

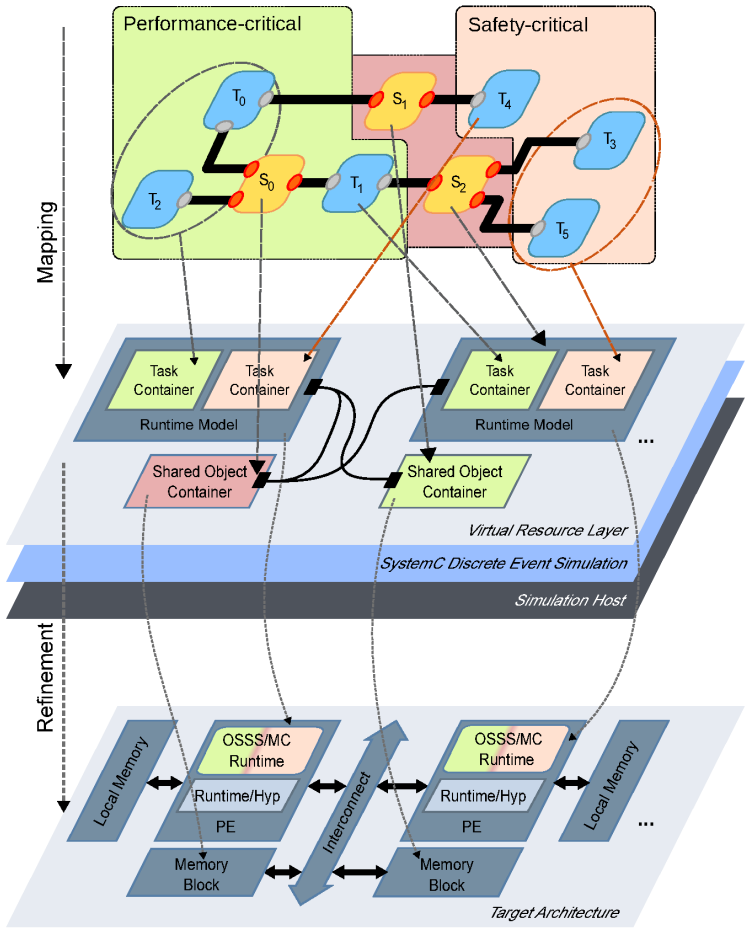
- Based on **SystemC/C++** methodology OSSS
- Define functional and extra-functional **behaviour**

2 Mixed-Criticality **Resource Model**

- Provide operational semantics (scheduling & resource access)
- Explore mixed-critical application **behaviour** and **mapping** decisions

3 Mixed-criticality **Execution Environment**

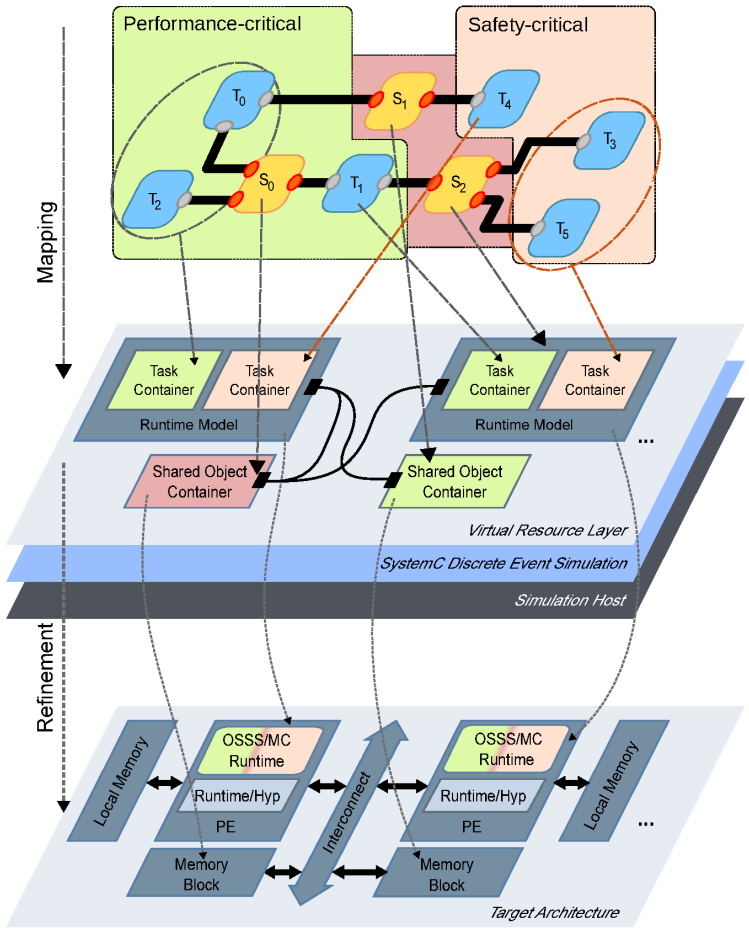
- Proof-of-Concept Implementation based on a **commercial MPSoC**
- Hypervisor for **application-level** context switch
- **Monitor** resource usage behaviour



6 OSSS/MC Methodology Overview

OSSS/MC Methodology & Modelling Components

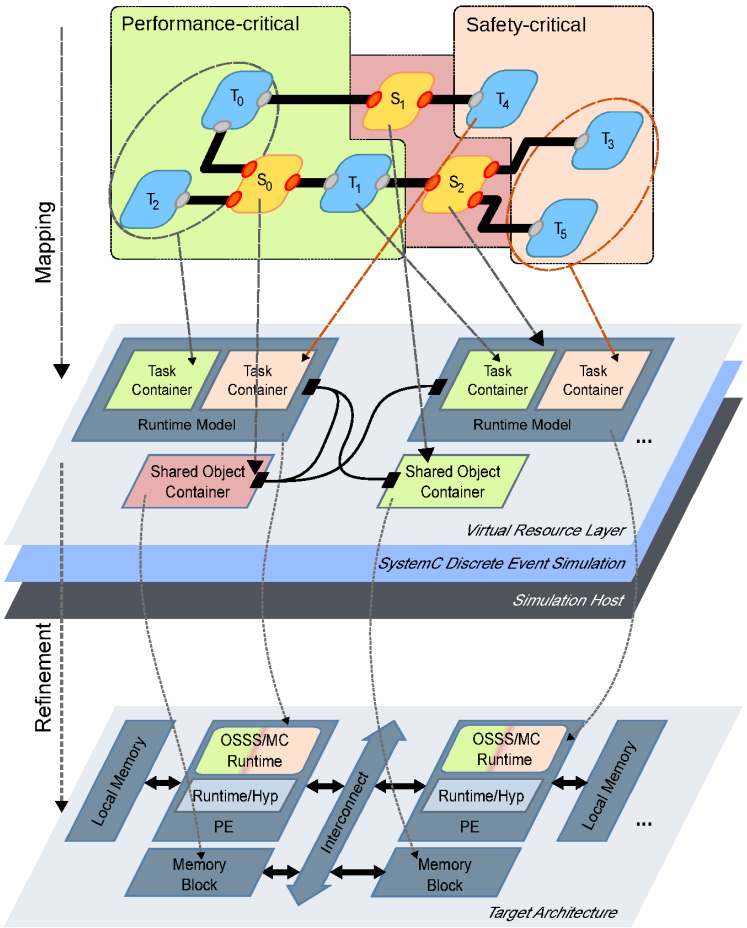
1. Mixed-Criticality **Programming Model**
   - Based on **SystemC/C++** methodology OSSS
   - Define functional and extra-functional **behaviour**
2. Mixed-Criticality **Resource Model**
   - Provide operational semantics (scheduling & resource access)
   - Explore mixed-critical application **behaviour** and **mapping** decisions
3. Mixed-criticality **Execution Environment**
   - Proof-of-Concept Implementation based on a **commercial MPSoC**
   - Hypervisor for **application-level** context switch
   - **Monitor** resource usage behaviour



7 OSSS/MC Scope

OSSS/MC Methodology & Modelling Components

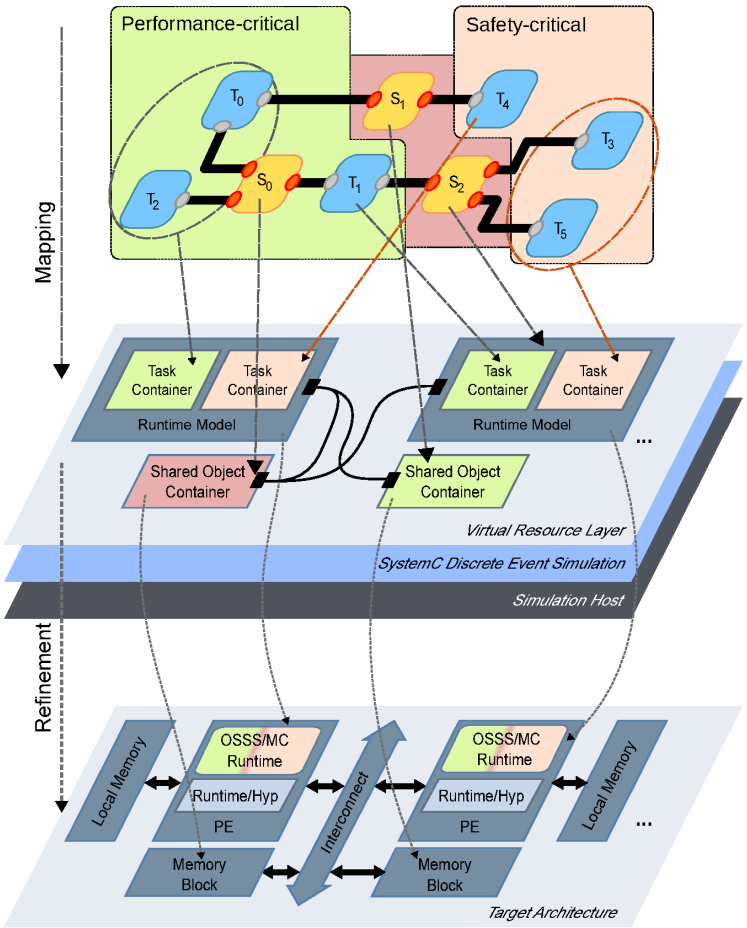
- Build upon external **analysis results**, e.g. EMC²
  WP2 WCRT analysis framework **art2kitekt**



7 OSSS/MC Scope

OSSS/MC Methodology & Modelling Components

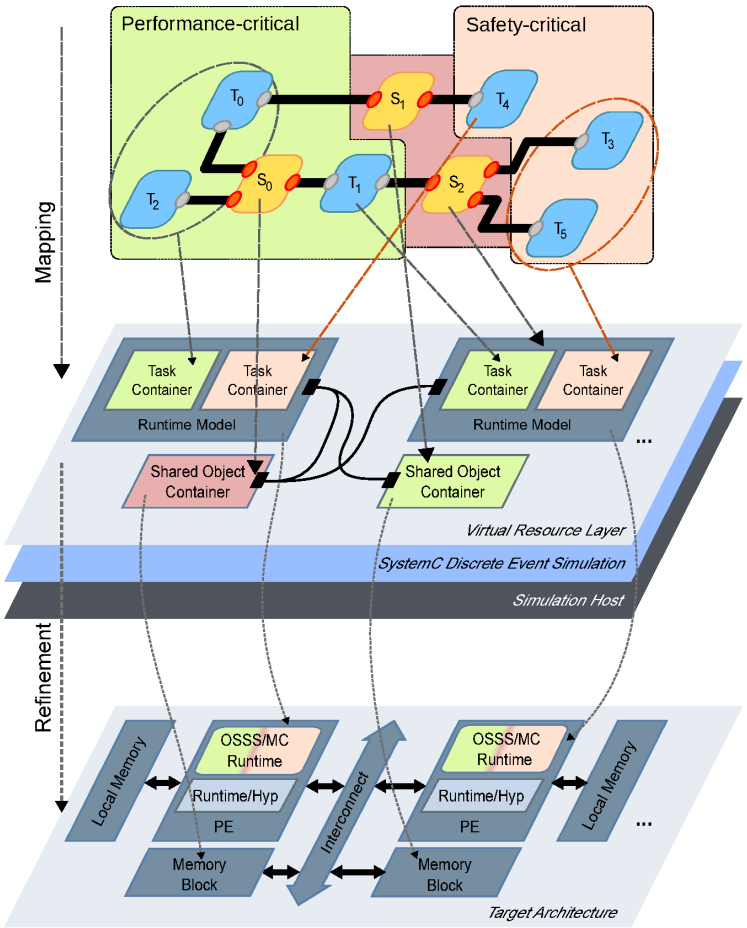
- Build upon external **analysis results**, e.g. EMC² WP2 WCRT analysis framework **art2kitekt**
- **Explore** integration choices for performance-critical use-cases
  - Integrate mixed-criticality scheduling policy
  - Evaluate and integrate functional behaviour



7 OSSS/MC Scope

OSSS/MC Methodology & Modelling Components

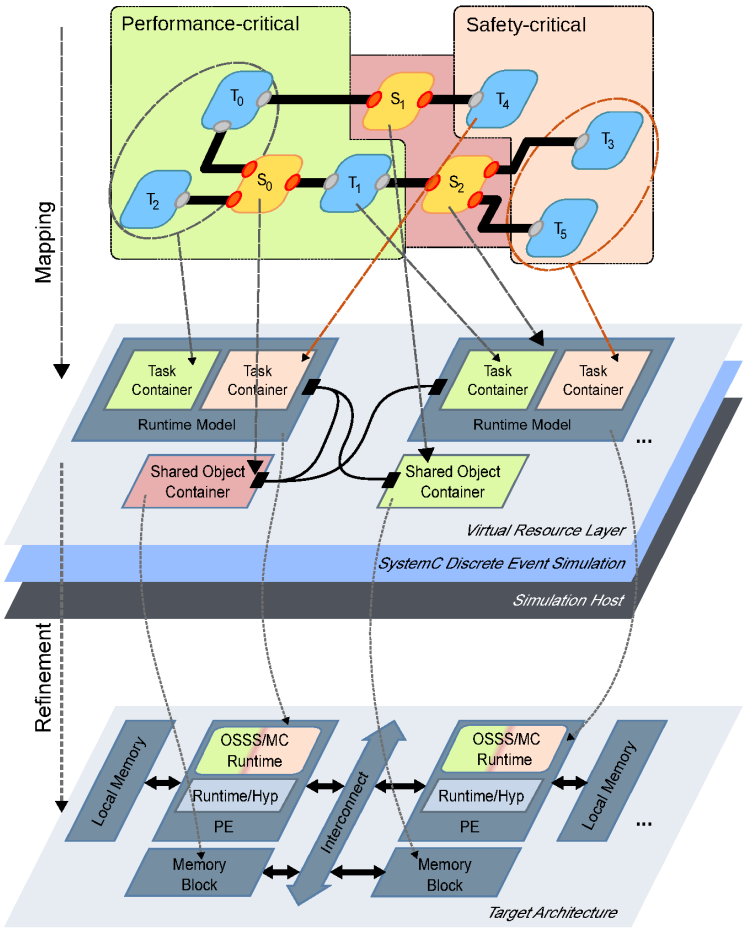
- Build upon external **analysis results**, e.g. EMC² WP2 WCRT analysis framework **art2kitekt**
- **Explore** integration choices for performance-critical use-cases
  - Integrate mixed-criticality scheduling policy
  - Evaluate and integrate functional behaviour
- Provide an **implementation candidate** on an MPSoC target
  - Hypervisor-based **temporal/spatial isolation**
  - Manage **dynamic slots**, **observe & contol** run-time behaviour



8 OSSS/MC Methodology Overview

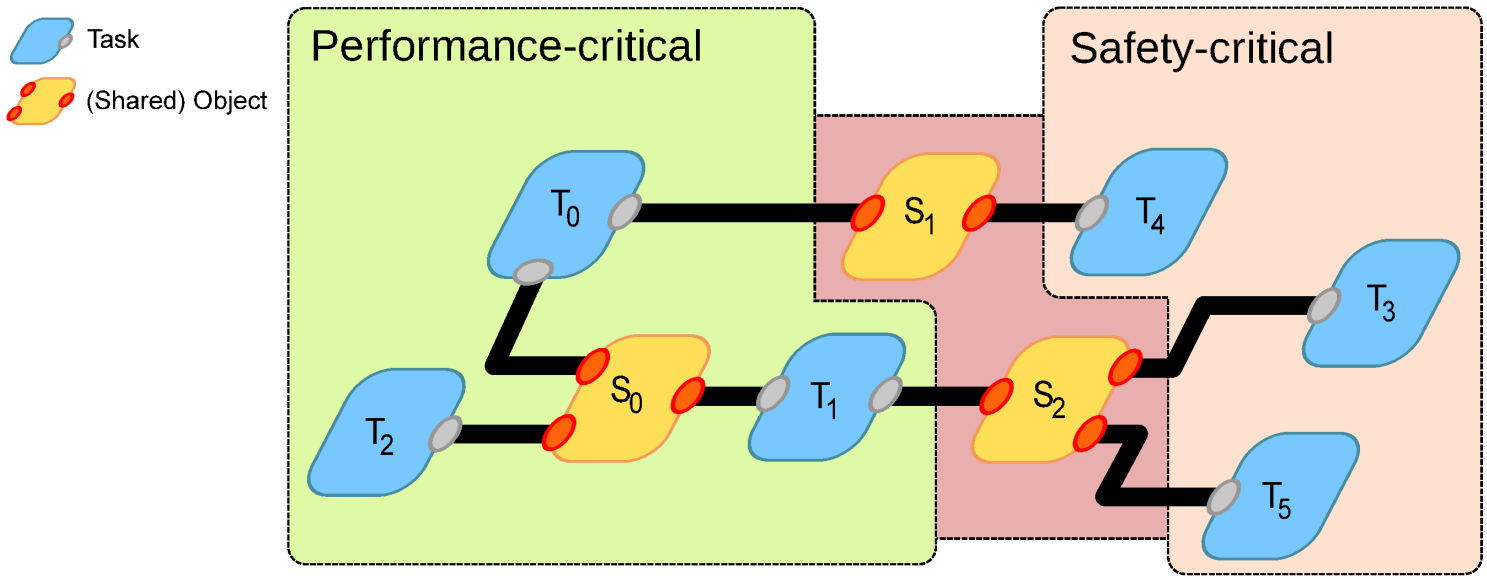
OSSS/MC Methodology & Modelling Components

1. Mixed-Criticality **Programming Model**
   - Based on **SystemC/C++** methodology OSSS
   - Define functional and extra-functional **behaviour**
2. Mixed-Criticality **Resource Model**
   - Provide operational semantics (scheduling & resource access)
   - Explore mixed-critical application **behaviour** and **mapping** decisions
3. Mixed-criticality **Execution Environment**
   - Proof-of-Concept Implementation based on a **commercial MPSoC**
   - Hypervisor for **application-level** context switch
   - **Monitor** resource usage behaviour



9 Programming Model Components

OSSS/MC Methodology & Modelling Components

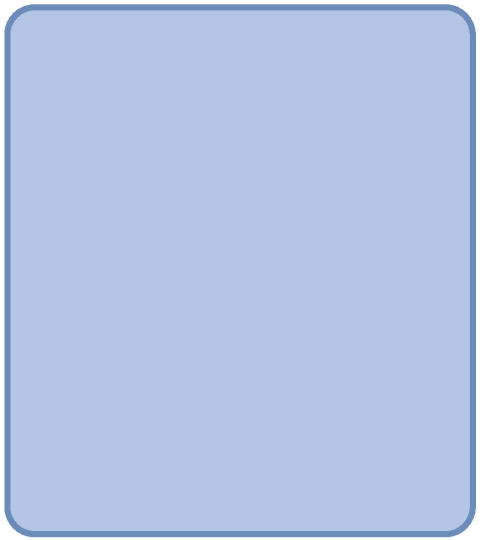


- **Tasks:** Capture behaviour and timing requirements for all utilisation profiles/scenarios
- **Shared Objects:** Communication between tasks and applications

10 Computation: Tasks

OSSS/MC Methodology & Modelling Components

```
OSSSMC_TASK(MySafetyTask) {

  OSSSMC_TASK_CTOR(MySafetyTask, criticality::safety,
                   1, /* Priority */
                   10_s, 10_s) /* Deadline & Period */ {}
  // ...
```

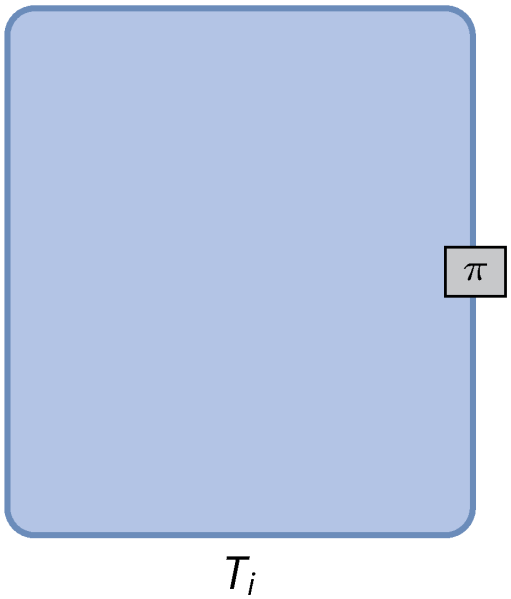


T_i

10 Computation: Tasks

OSSS/MC Methodology & Modelling Components

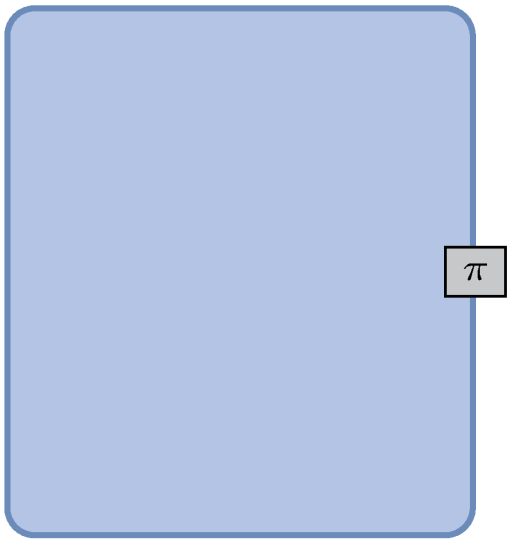
```
OSSSMC_TASK(MySafetyTask) {
  osssmc::port< fifo_put_if > obj;
  OSSSMC_TASK_CTOR(MySafetyTask, criticality::safety,
                   1, /* Priority */
                   10_s, 10_s) /* Deadline & Period */ {}
  // ...
```



10 Computation: Tasks

OSSS/MC Methodology & Modelling Components

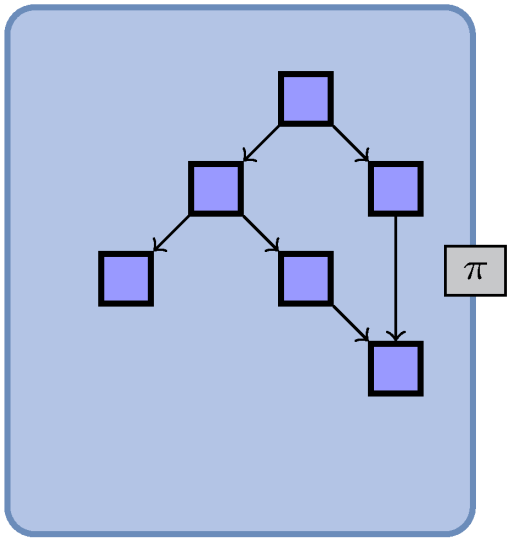
```
OSSSMC_TASK(MySafetyTask) {
  osssmc::port< fifo_put_if > obj;
  OSSSMC_TASK_CTOR(MySafetyTask, criticality::safety,
                   1, /* Priority */
                   10_s, 10_s) /* Deadline & Period */ {}
  OSSSMC_BEHAVIOUR(scenario) {
  // ...
```



10 Computation: Tasks

OSSS/MC Methodology & Modelling Components

```
OSSSMC_TASK(MySafetyTask) {
  osssmc::port< fifo_put_if > obj;
  OSSSMC_TASK_CTOR(MySafetyTask, criticality::safety,
                   1, /* Priority */
                   10_s, 10_s) /* Deadline & Period */ {}
  OSSSMC_BEHAVIOUR(scenario) {
    int val;
    val = behaviour();
    obj->put(val);
  }
};
```

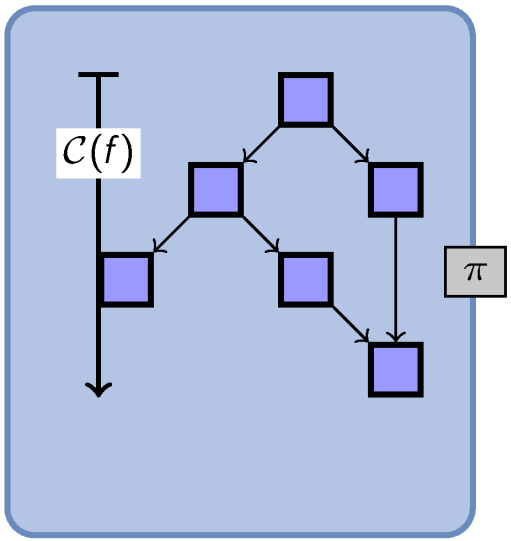


T_i

10 Computation: Tasks

OSSS/MC Methodology & Modelling Components

```
OSSSMC_TASK(MySafetyTask) {
  osssmc::port< fifo_put_if > obj;
  OSSSMC_TASK_CTOR(MySafetyTask, criticality::safety,
                   1, /* Priority */
                   10_s, 10_s) /* Deadline & Period */ {}
  OSSSMC_BEHAVIOUR(scenario) {
    int val;
    OSSSMC_CONSUME( distribution( 8_s ) ) {
      val = behaviour();
    }
    obj->put(val);
  }
};
```

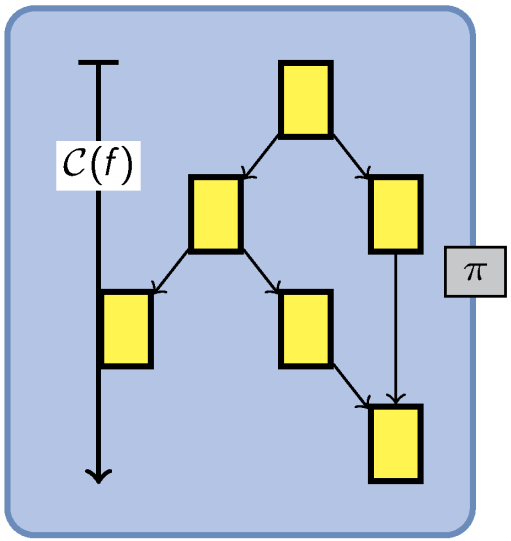


T_i

10 Computation: Tasks

OSSS/MC Methodology & Modelling Components

```
OSSSMC_TASK(MySafetyTask) {
  osssmc::port< fifo_put_if > obj;
  OSSSMC_TASK_CTOR(MySafetyTask, criticality::safety,
                   1, /* Priority */
                   10_s, 10_s) /* Deadline & Period */ {}
  OSSSMC_BEHAVIOUR(scenario) {
    int val;
    OSSSMC_CONSUME( distribution( 8_s ) ) {
      val = behaviour();
    }
    obj->put(val);
  }
};
```

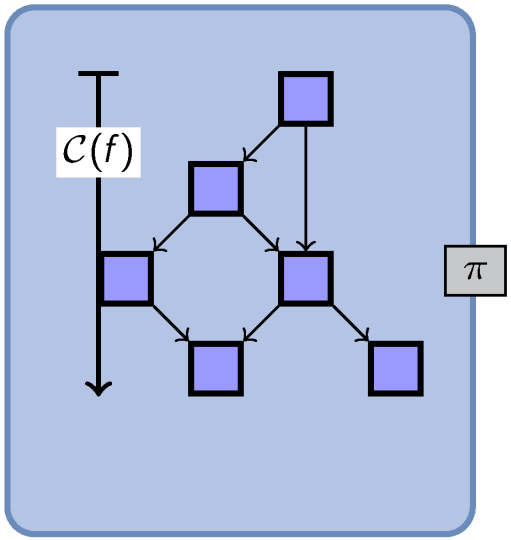


T_i

10 Computation: Tasks

OSSS/MC Methodology & Modelling Components

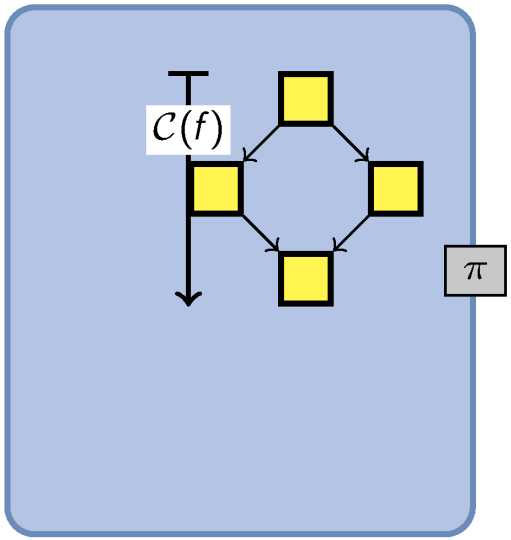
```
// ...
OSSSMC_BEHAVIOUR(scenario) {
  int val;
  if (scenario == scenario_type::dynamic)  {
    OSSSMC_CONSUME( 4_s ) {
      val = high_quality_behaviour();
    }
  }




  obj->put(val);
}
// ...
```



10 Computation: Tasks

OSSS/MC Methodology & Modelling Components

```
// ...
OSSSMC_BEHAVIOUR(scenario) {
  int val;
  if (scenario == scenario_type::dynamic)  {
    OSSSMC_CONSUME( 4_s ) {
      val = high_quality_behaviour();
    }
  } else {
    OSSSMC_CONSUME( 2_s ) {
      val = reduced_quality_behaviour();
    }
  }
  obj->put(val);
}
// ...
```



11 Communication: Shared Objects

OSSS/MC Methodology & Modelling Components

```
struct fifo_put_if {
    virtual void put(int val) = 0;
    virtual int get() = 0;
    virtual size_t num_elements() const = 0;
};

struct my_fifo : public fifo_put_if {
  size_t size_;
  std::queue<int> items_;

  // ...
};
```

- User-defined **data container**
- Implement communication interface
  - Blocking/non-blocking access

11 Communication: Shared Objects

OSSS/MC Methodology & Modelling Components

```
struct fifo_put_if {
    virtual void put(int val) = 0;
    virtual int get() = 0;
    virtual size_t num_elements() const = 0;
};

struct my_fifo : public fifo_put_if {
  size_t size_;
  std::queue<int> items_;

  // ...
};

// Task definition contains
osssmc::port< fifo_put_if > obj;
```

- User-defined **data container**
- Implement communication interface
  - Blocking/non-blocking access
  - Accessible using task ports

11 Communication: Shared Objects

OSSS/MC Methodology & Modelling Components

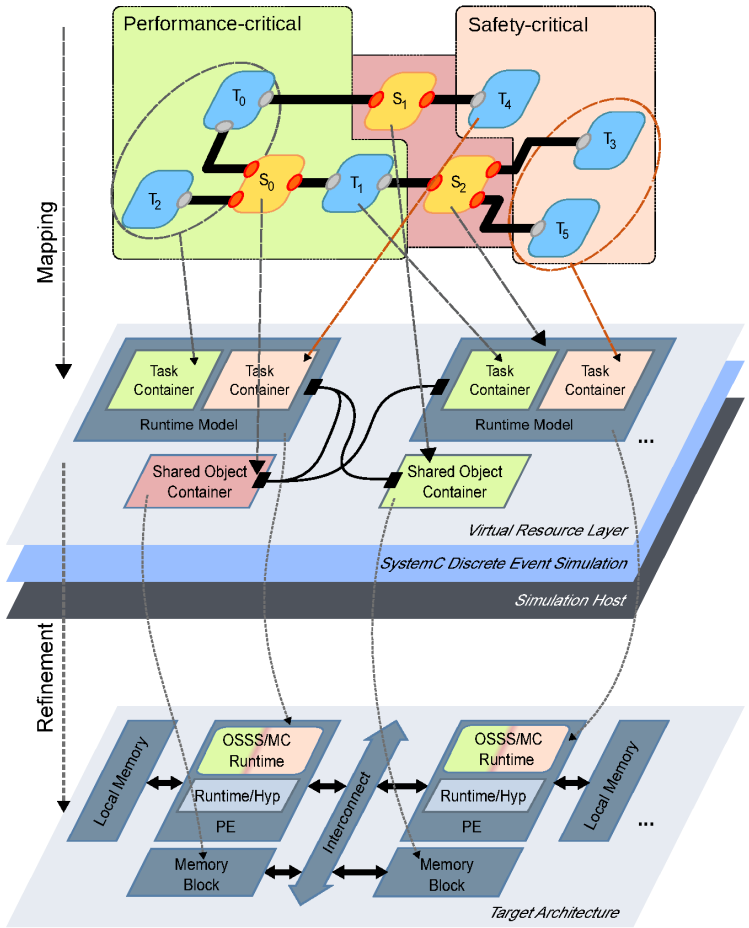
```
struct fifo_put_if {
    virtual void put(int val) = 0;
    virtual int get() = 0;
    virtual size_t num_elements() const = 0;
};

struct my_fifo : public fifo_put_if {
  size_t size_;
  std::queue<int> items_;

  // ...
};

// Task definition contains
osssmc::port< fifo_put_if > obj;
```

- User-defined **data container**
- Implement communication interface
  - Blocking/non-blocking access
  - Accessible using task ports
- **Cross-criticality** communication
  - **non-blocking**, no data **dependencies**
  - `const`: Prevent **spatial** interference
  - Mapping restrictions to ensure **temporal** isolation

12 OSSS/MC Methodology Overview

OSSS/MC Methodology & Modelling Components

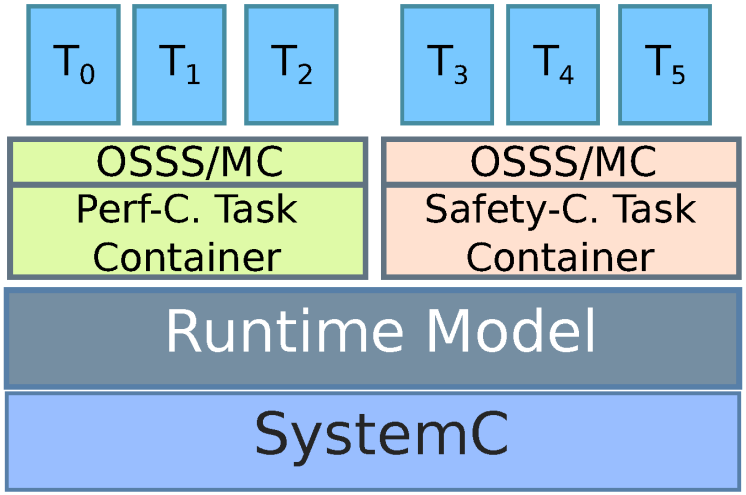
1. Mixed-Criticality **Programming Model**
   - Based on **SystemC/C++** methodology OSSS
   - Define functional and extra-functional **behaviour**
2. Mixed-Criticality **Resource Model**
   - Provide operational semantics (scheduling & resource access)
   - Explore mixed-critical application **behaviour** and **mapping** decisions
3. Mixed-criticality **Execution Environment**
   - Proof-of-Concept Implementation based on a **commercial MPSoC**
   - Hypervisor for **application-level** context switch
   - **Monitor** resource usage behaviour



13 Mapping the Programming Model to the Resource Model

OSSS/MC Methodology & Modelling Components

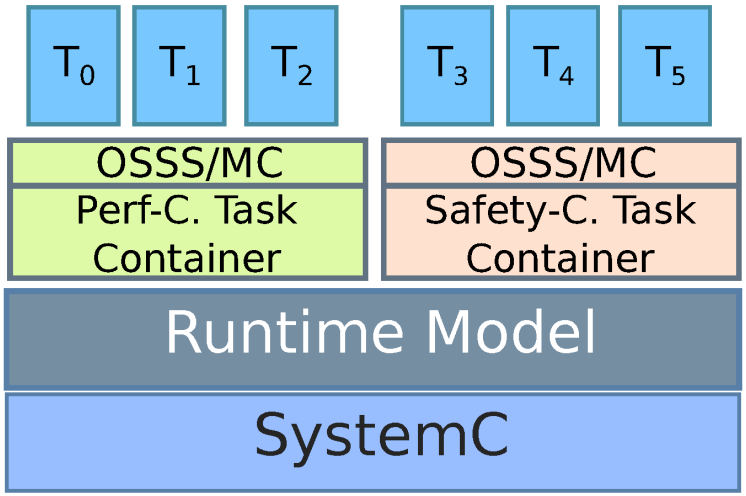
```
OSSSMC_PLATFORM(Top) {
  MyPerfTask t0, t1, t2;
  MySafetyTask t3, t4, t5;
  osssmc::shared_object<my_fifo> s0;
  // ...

  osssmc::runtime rt;
```



13 Mapping the Programming Model to the Resource Model

OSSS/MC Methodology & Modelling Components

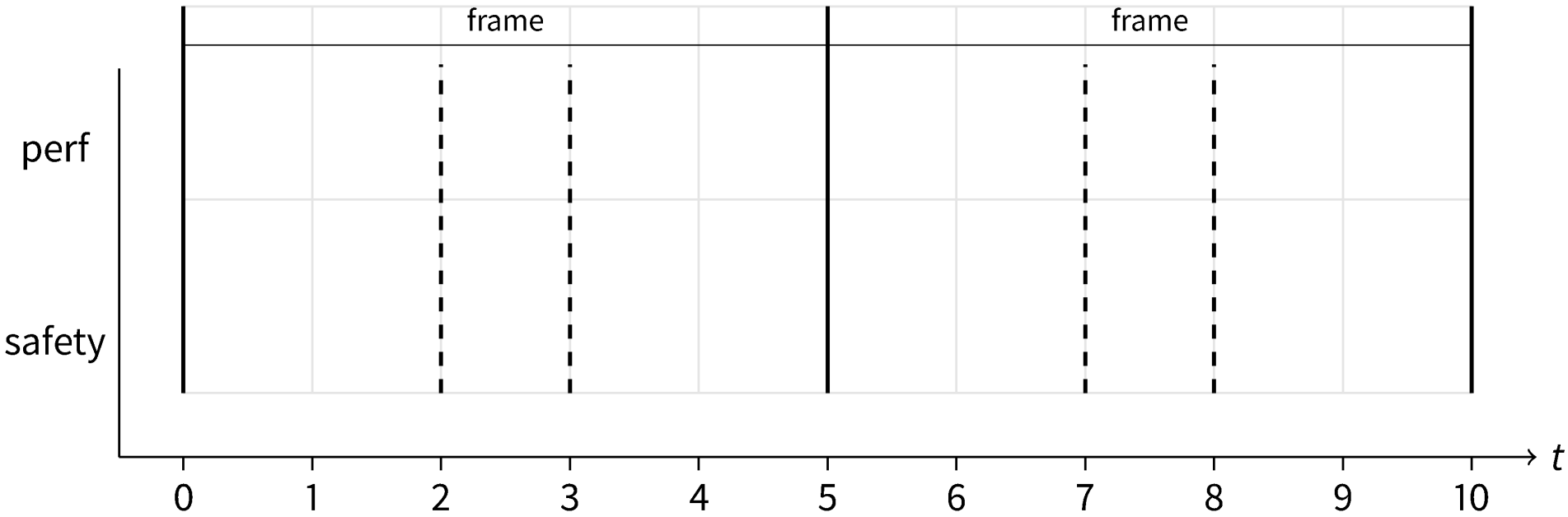
```
OSSSMC_PLATFORM(Top) {
  MyPerfTask t0, t1, t2;
  MySafetyTask t3, t4, t5;
  osssmc::shared_object<my_fifo> s0;
  // ...

  osssmc::runtime rt;

  SC_CTOR(Top) {
    t0.obj(s0);
    t1.obj(s1); // ...

    rt(t0);
    rt(t1); // ...

    rt.create_frame(5_s, {2_s, 3_s});
  }
};
```



14 Runtime Model Execution Semantics

OSSS/MC Methodology & Modelling Components

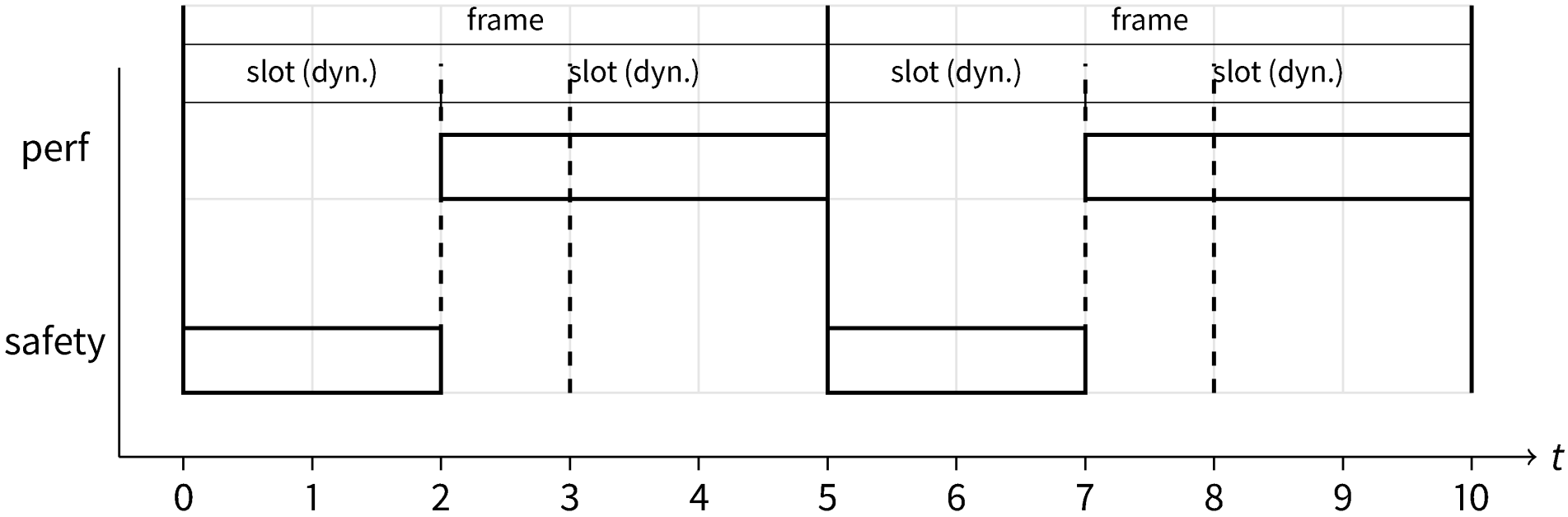
- `rt.create_frame(5_s, {2_s, 3_s});`
- **Flexible Time-Triggered Scheduling (FTTS)**, Giannopoulou et al. [2]



14 Runtime Model Execution Semantics

OSSS/MC Methodology & Modelling Components

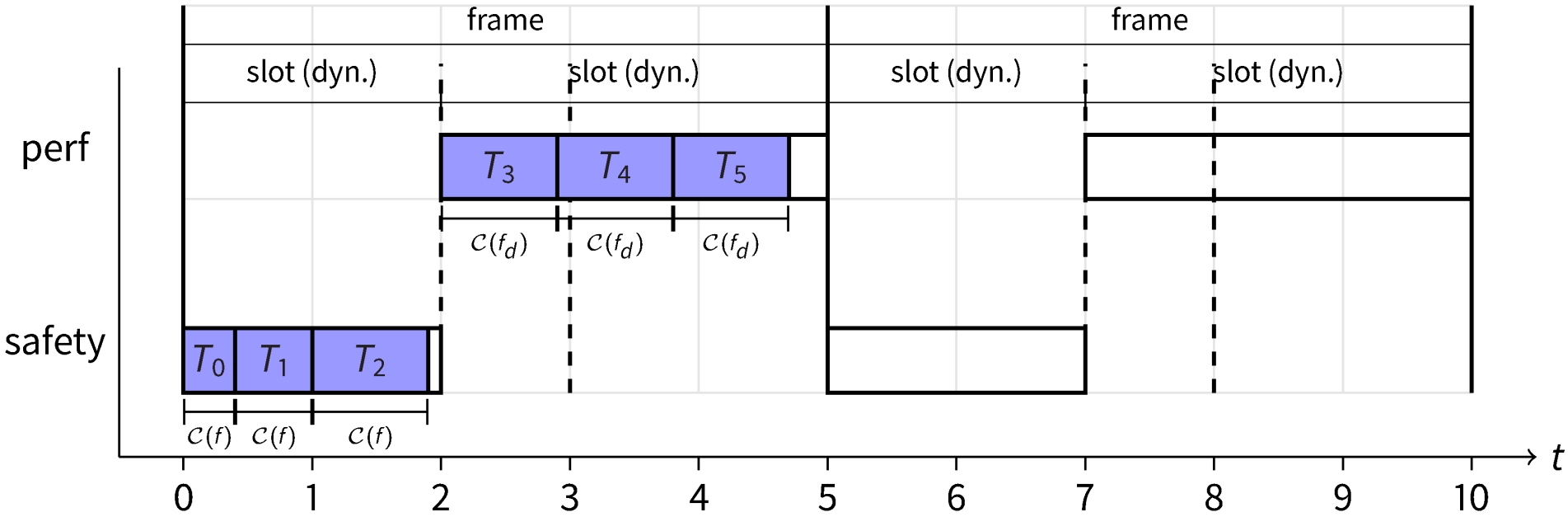
- `rt.create_frame(5_s, {2_s, 3_s});`
- **Flexible Time-Triggered Scheduling (FTTS)**, Giannopoulou et al. [2]



14 Runtime Model Execution Semantics

OSSS/MC Methodology & Modelling Components

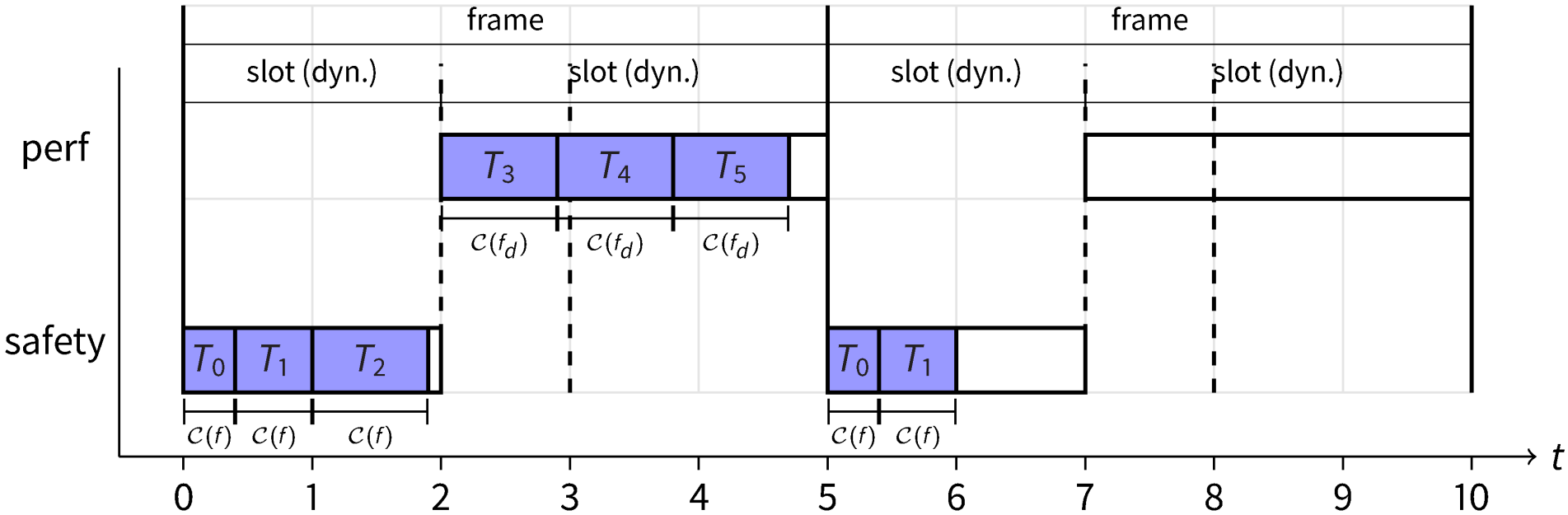
- `rt.create_frame(5_s, {2_s, 3_s});`
- **Flexible Time-Triggered Scheduling (FTTS)**, Giannopoulou et al. [2]



14 Runtime Model Execution Semantics

OSSS/MC Methodology & Modelling Components

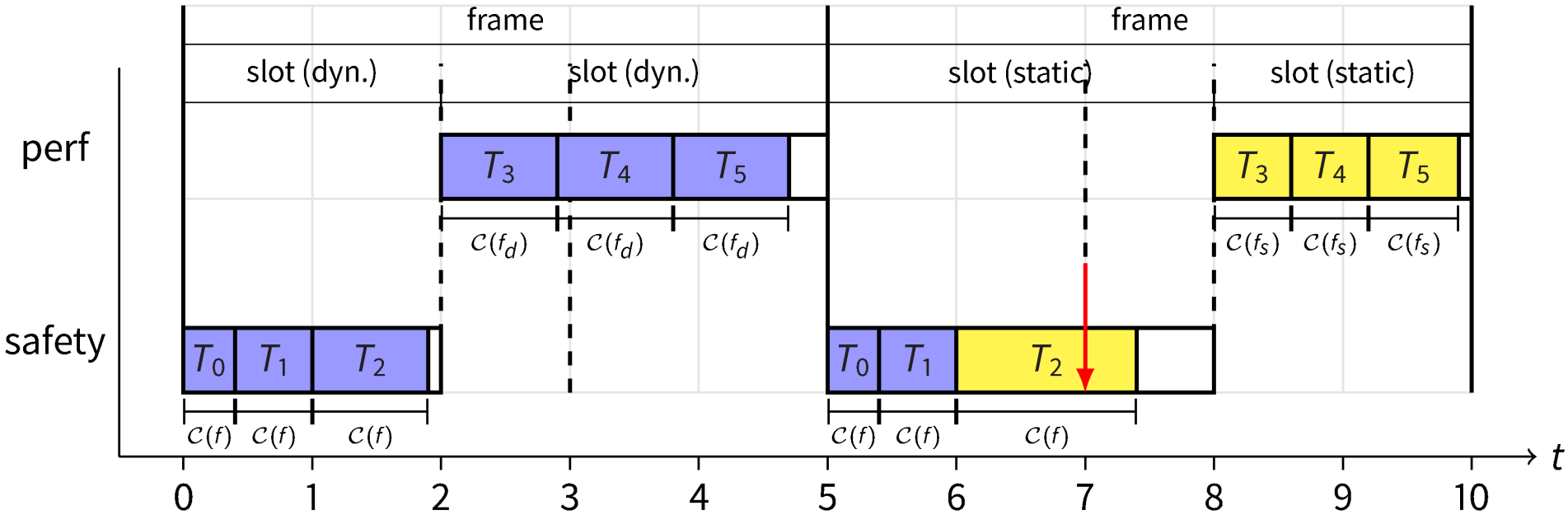
- `rt.create_frame(5_s, {2_s, 3_s});`
- **Flexible Time-Triggered Scheduling (FTTS)**, Giannopoulou et al. [2]



14 Runtime Model Execution Semantics

OSSS/MC Methodology & Modelling Components

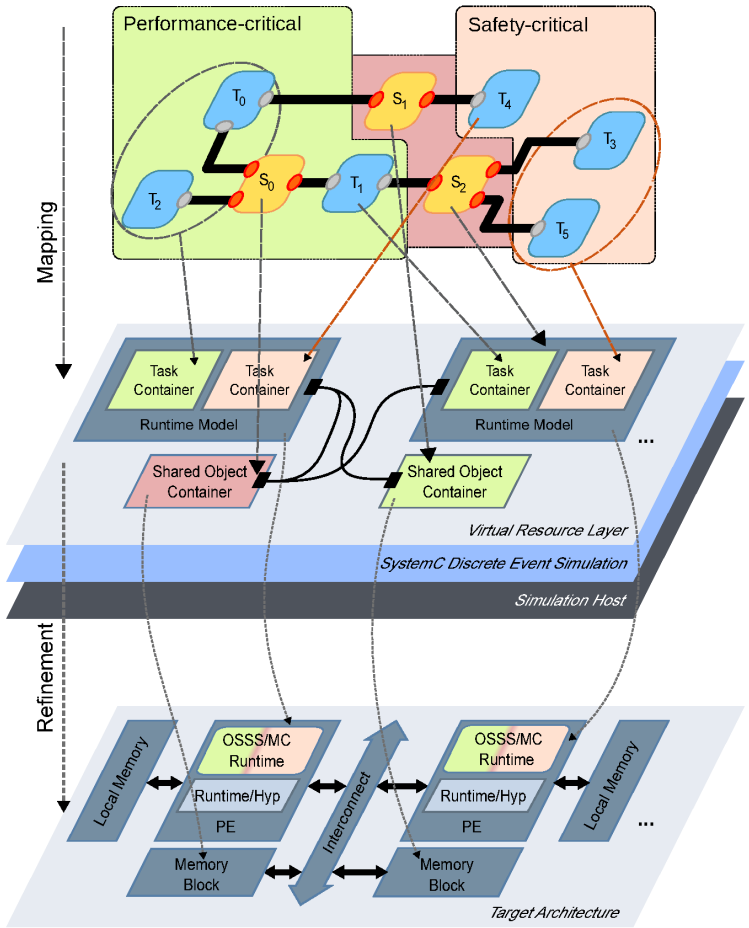
- `rt.create_frame(5_s, {2_s, 3_s});`
- **Flexible Time-Triggered Scheduling (FTTS)**, Giannopoulou et al. [2]



15 OSSS/MC Methodology Overview

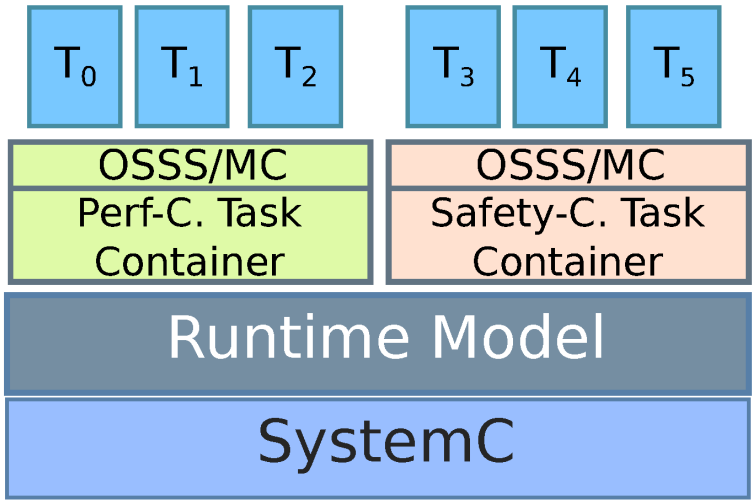
OSSS/MC Methodology & Modelling Components

1. Mixed-Criticality **Programming Model**
   - Based on **SystemC/C++** methodology OSSS
   - Define functional and extra-functional **behaviour**
2. Mixed-Criticality **Resource Model**
   - Provide operational semantics (scheduling & resource access)
   - Explore mixed-critical application **behaviour** and **mapping** decisions
3. Mixed-criticality **Execution Environment**
   - Proof-of-Concept Implementation based on a **commercial MPSoC**
   - Hypervisor for **application-level** context switch
   - **Monitor** resource usage behaviour



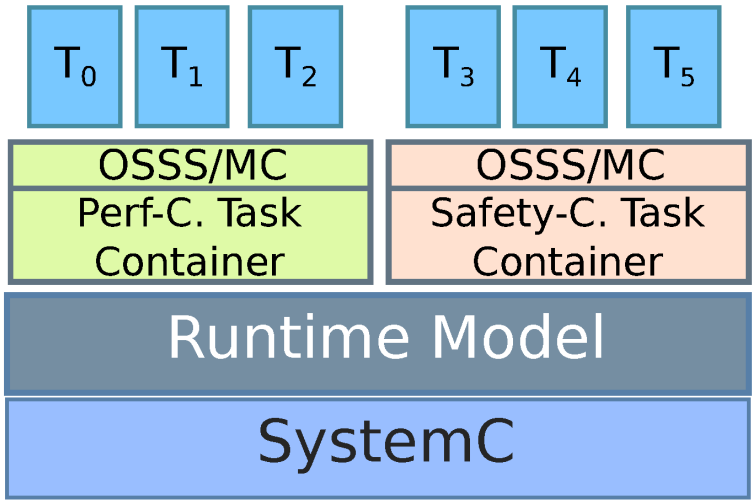
16 Resource Model Implementation

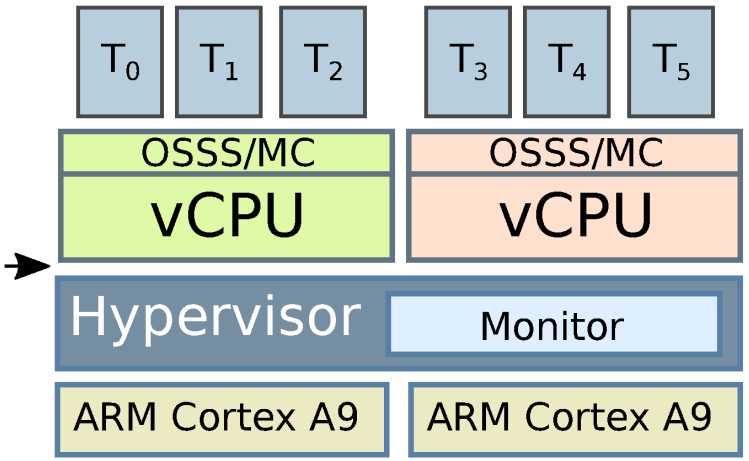
OSSS/MC Methodology & Modelling Components



16 Resource Model Implementation

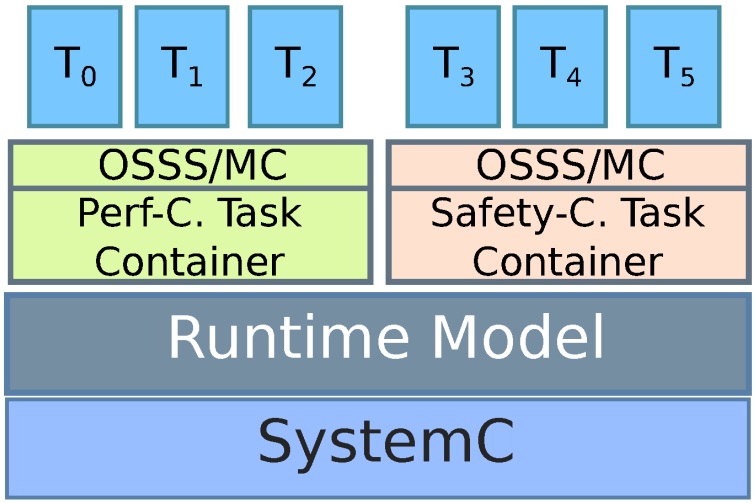
OSSS/MC Methodology & Modelling Components

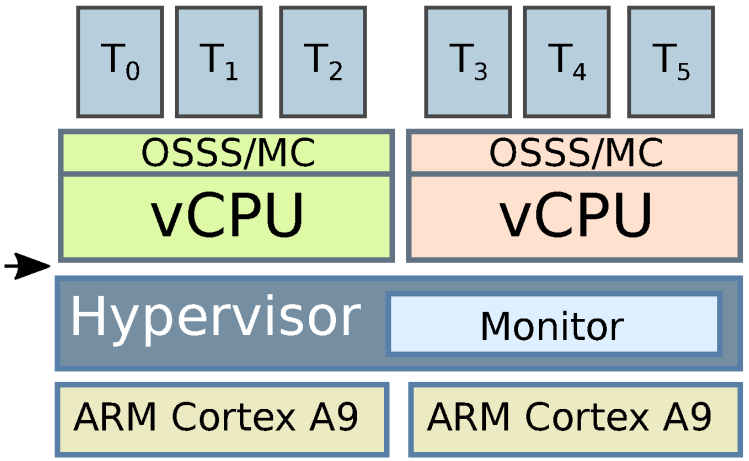




16 Resource Model Implementation

OSSS/MC Methodology & Modelling Components

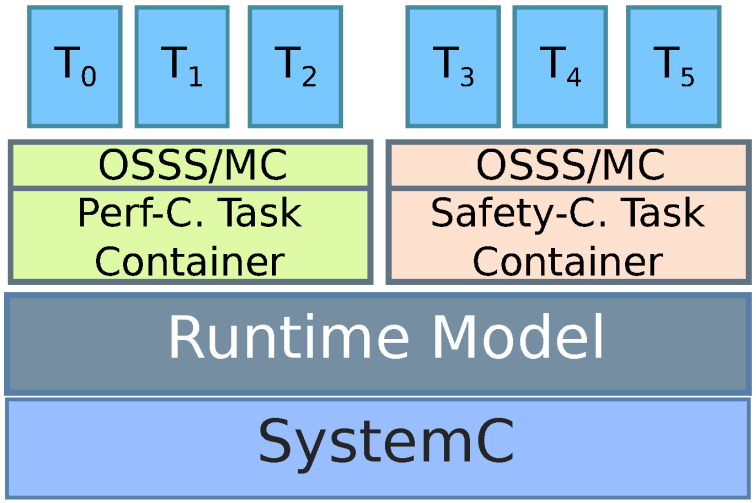


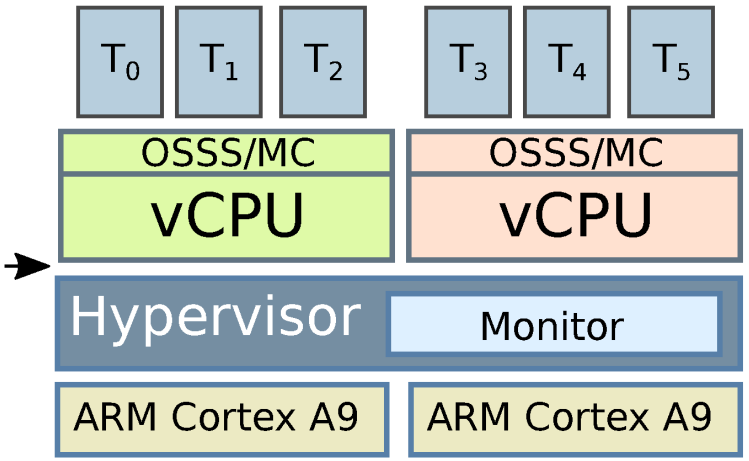


- **vCPUs & OSSS/MC runtime**
  - RTOS instance & OSSS/MC Layer
  - Control periodic task execution

16 Resource Model Implementation

OSSS/MC Methodology & Modelling Components

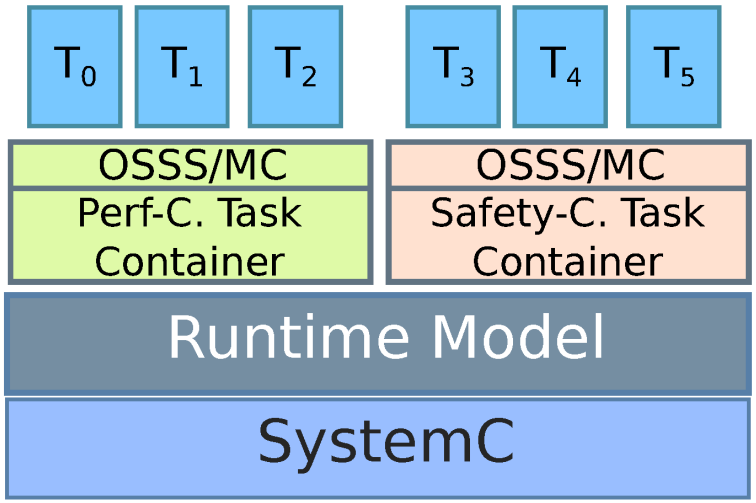




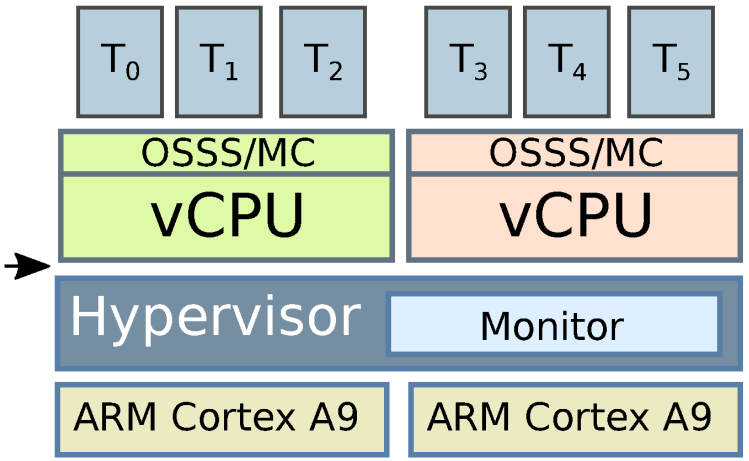
- **vCPUs & OSSS/MC runtime**
  - RTOS instance & OSSS/MC Layer
  - Control periodic task execution
- **Bare-metal hypervisor**
  - Application context switch
  - Manage **scenario** state, adjust slot length

16 Resource Model Implementation

OSSS/MC Methodology & Modelling Components



- **vCPUs & OSSS/MC runtime**
  - RTOS instance & OSSS/MC Layer
  - Control periodic task execution
- **Bare-metal hypervisor**
  - Application context switch
  - Manage **scenario** state, adjust slot length

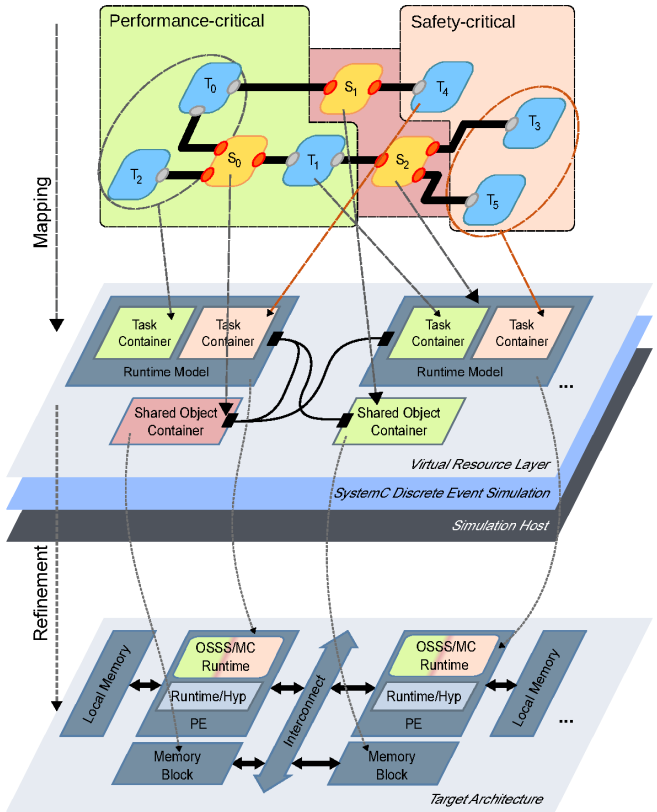


- **Monitor**
  - Virtual MMI/O device for task start/end events
  - Provide scenario state to OSSS/MC instances

17 Summary

Summary

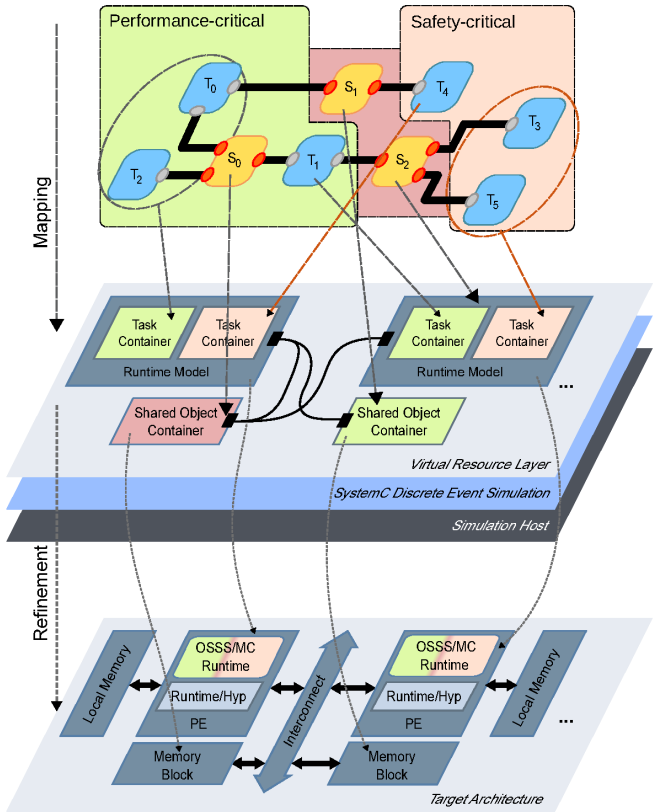
1. **Mixed-criticality** programming model
   - Dynamic system **scenarios/utilisation profiles**



17 Summary

Summary

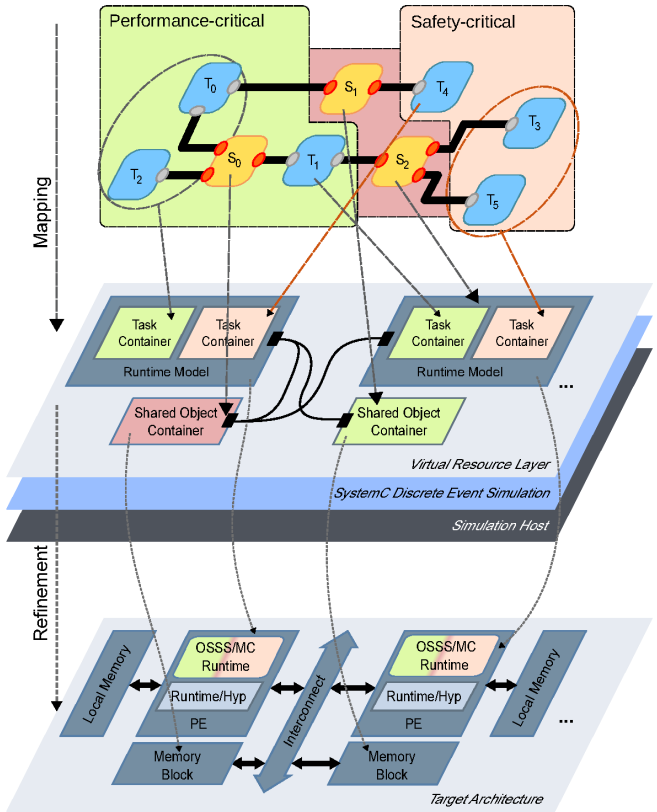
1. **Mixed-criticality** programming model
   - Dynamic system **scenarios/utilisation profiles**
2. Host-based mixed-criticality **resource model**
   - Explore functional application **behaviour** and **scheduling** decisions



17 Summary

Summary

1. **Mixed-criticality** programming model
   - Dynamic system **scenarios/utilisation profiles**
2. Host-based mixed-criticality **resource model**
   - Explore functional application **behaviour** and **scheduling** decisions
3. Mixed-criticality **execution environment**
   - Proof-of-concept implementation built on top of a **commercial MPSoC**
   - Provide isolated **execution slots**
   - **Monitor** resource usage behaviour

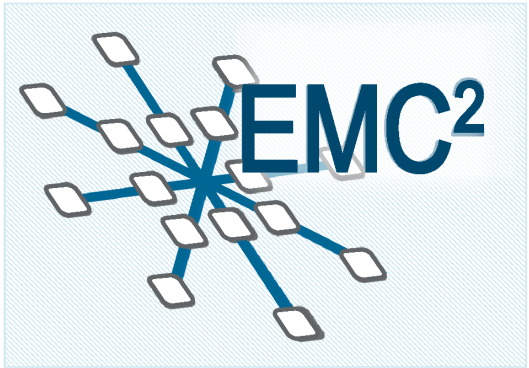


18 That's all, folks...

Summary

Thank you! Any questions?

Philipp Ittershagen, <pit@offis.de>

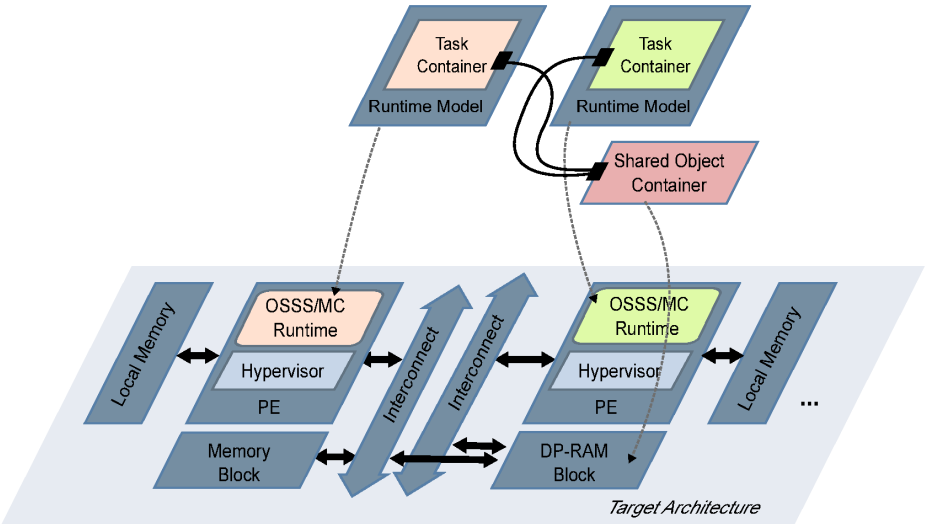


`https://www.artemis-emc2.eu/`

19 Application Model Mapping

Backup Slides

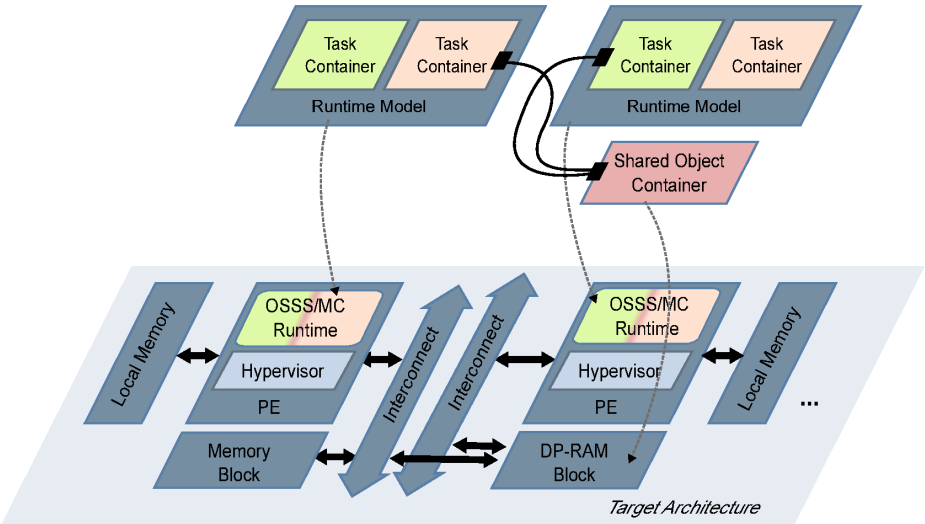
- 1:1 and 2:1 mapping of task containers to runtime models
- Communication across containers: **cross-criticality** only
  - Dual-ported RAM for **interference-free** cross-criticality communication across PEs
- No distribution of task containers across PEs
  - Currently no support for slot synchronisation
- Scenario state is local to each resource



19 Application Model Mapping

Backup Slides

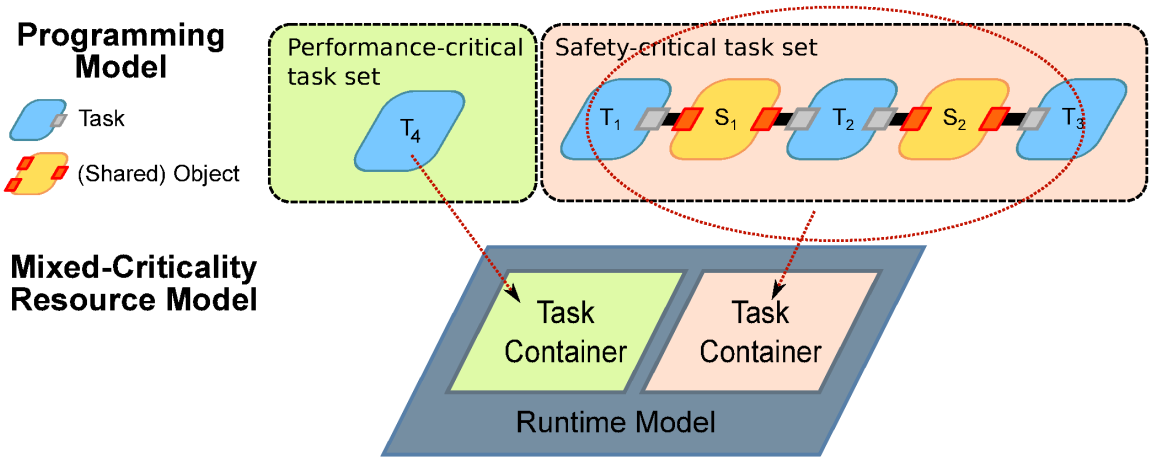
- 1:1 and 2:1 mapping of task containers to runtime models
- Communication across containers: **cross-criticality** only
  - Dual-ported RAM for **interference-free** cross-criticality communication across PEs
- No distribution of task containers across PEs
  - Currently no support for slot synchronisation
- Scenario state is local to each resource



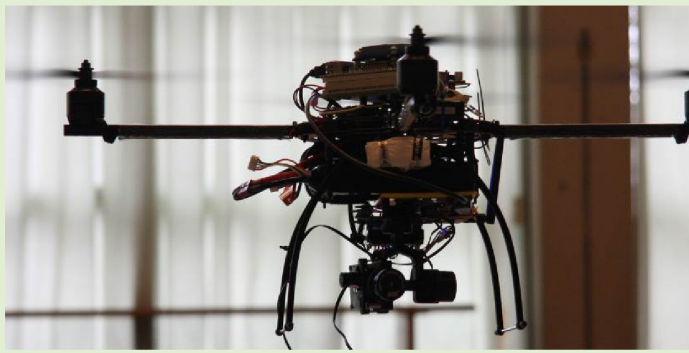
20 Evaluation: Quadrocopter Use-Case (Ittershagen et al. [3])

Evaluation

- Goal: Compare resource utilisation of **static** and **dynamic** time slot approach
- Safety-critical application: flight controller, **essential** for a stable flight
- Performance-critical application: video-based **object detection**, needs to run at **6 fps**.



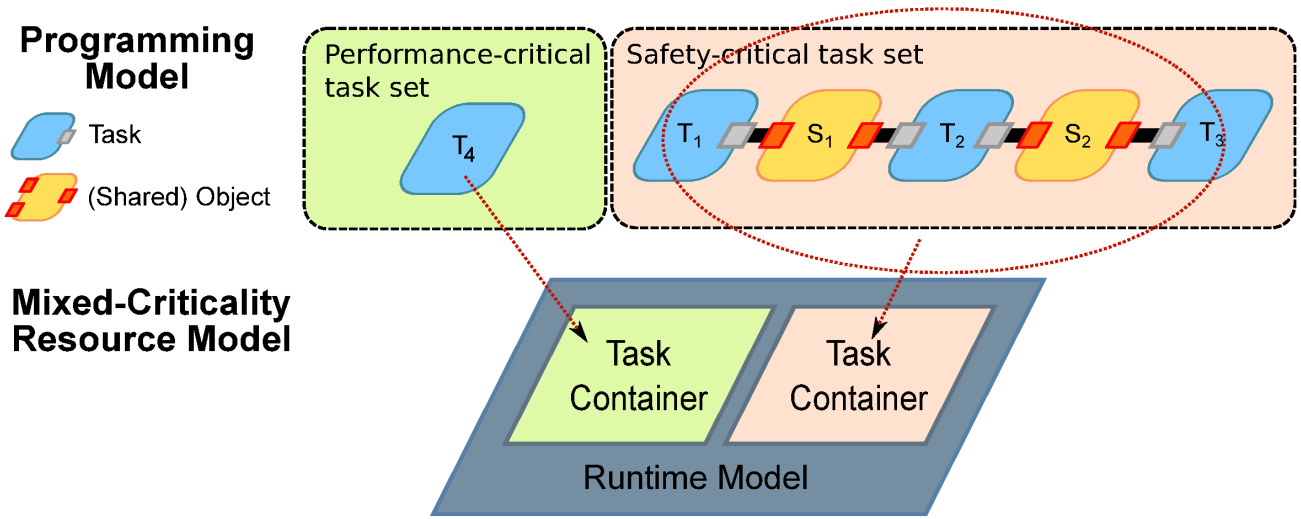
Quadrocopter System



21 Static Approach
Evaluation

Requirement: Video processing needs to achieve 6 fps (167 ms for each frame).

1. Statically analyse WCETs for safety-critical task set and determine utilisation: ≈ 61.50 % each frame.
2. Calibrate video processing:
   - 320x200 (58 ms ≈ 35 % each frame)

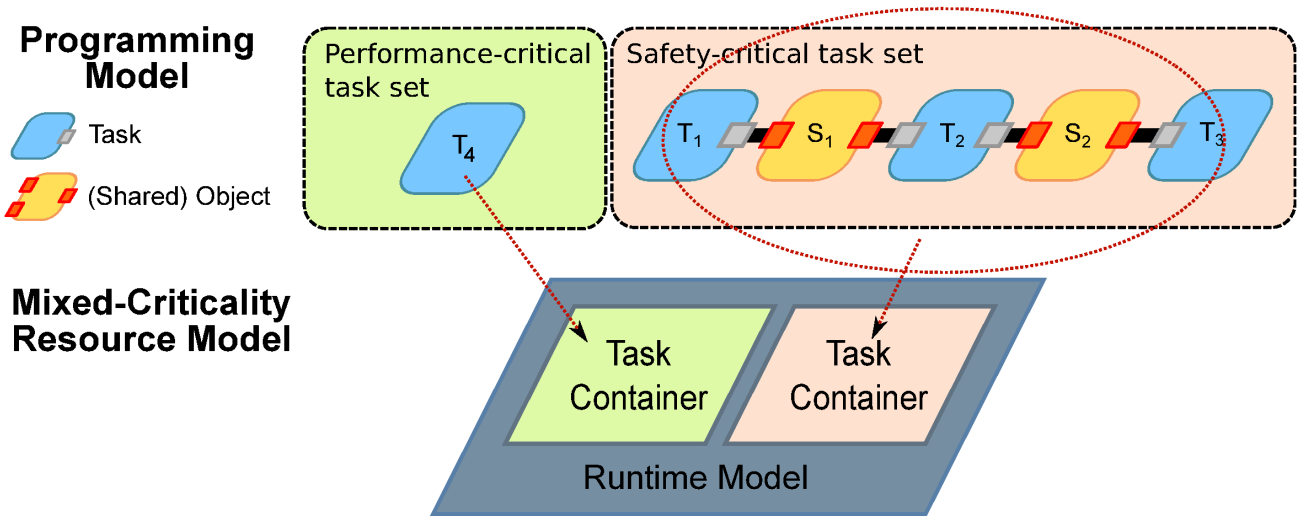


22 Mixed-Criticality Programming Model

Evaluation

Assumption: Observed safety-critical task execution time is normally distributed [1].

1. **Static** scenario: (previous slide)
2. **Dynamic** scenario: Assume safety-critical tasks finish within 90% of their WCET ≈ 43.4 % system utilisation.
3. Video processing for **dynamic scenario:**
   - 460x320 (93.3 ms ≈ 56 % each frame)



23 Evaluation Results

Evaluation

switching active?	static	dynamic	run-time utilisation (%)
no	30	0	65.54 (±0.16)
yes	13(±3)	17(±3)	86.70 (±0.14)

- Monitor safety-critical task computation time
- **Switch** scenario when detecting budget violation
- Dynamic scenario switching achieves better run-time utilisation
- Run-time switching overhead not included

24 Bibliography (I)

Evaluation

[1] Jaume Abella, Damien Hardy, Isabelle Puaut, Eduardo Quinones, and Francisco J Cazorla. On the comparison of deterministic and probabilistic wcet estimation techniques. In **26th Euromicro Conference on Real-Time Systems (ECRTS'14)**, pages 266–275. IEEE, 2014.

[2] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. Scheduling of mixed-criticality applications on resource-sharing multicore systems. In **Proceedings of the International Conference on Embedded Software (EMSOFT'2013)**, pages 1–15, Sept 2013. doi: 10.1109/EMSOFT.2013.6658595.

[3] Philipp Ittershagen, Kim Grüttner, and Wolfgang Nebel. Mixed-criticality system modelling with dynamic execution mode switching. In **Forum on Specification & Design Languages (FDL'2015)**. ECSI, 2015.

[4] Reinhard Wilhelm, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, Per Stenström, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, and Reinhold Heckmann. The worst-case execution-time problem—overview of methods and survey of tools. **ACM Transactions on Embedded Computing Systems**, 7(3):1–53, April 2008. ISSN 15399087. doi: 10.1145/1347375.1347389.