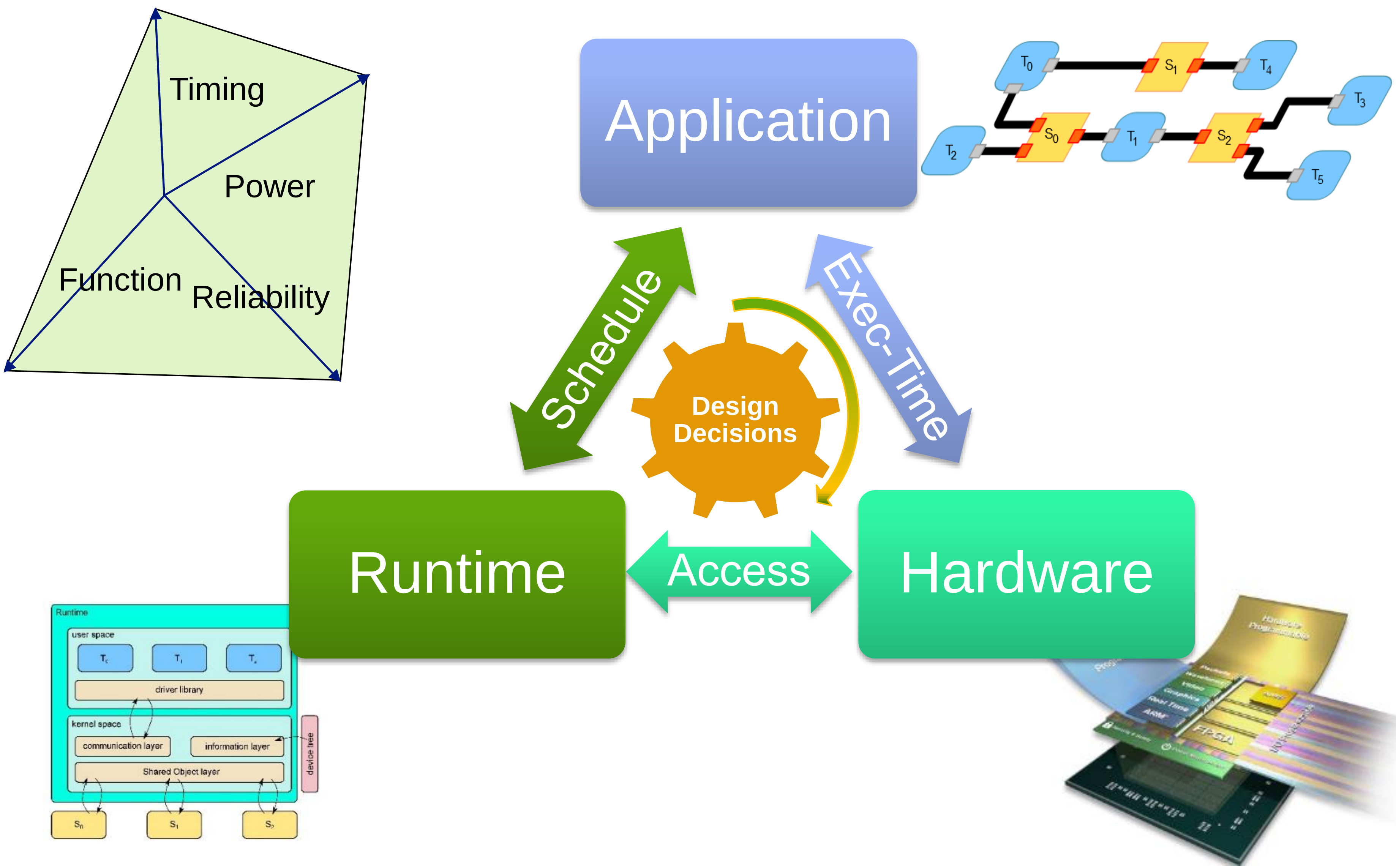
Verkehr
Transportation

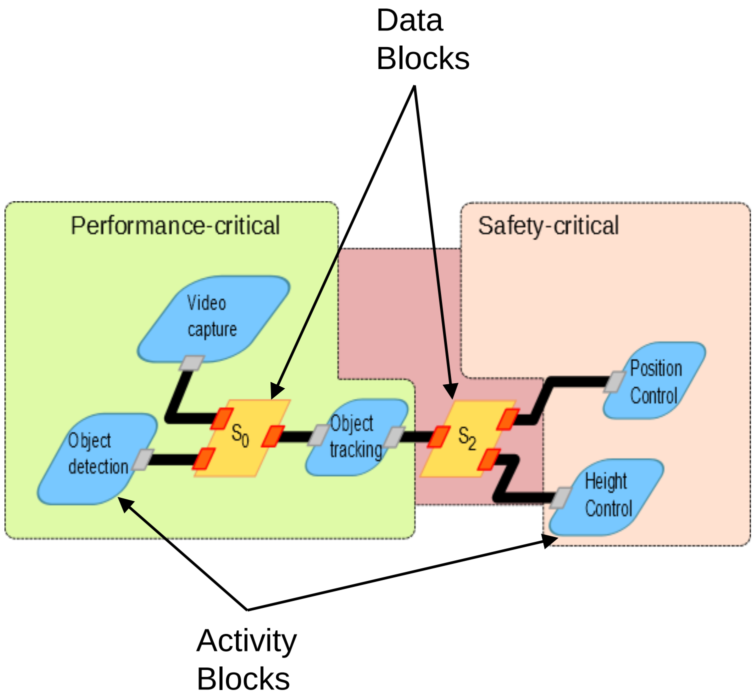
Platform-aware Modelling for Mixed-Critical Applications

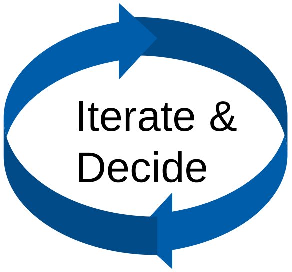
Frank Oppenheimer, Philipp Ittershagen, Sören Schreiner

2 Motivation: Cyber Physical Systems Decision Triangle



3 Clear steps & strong abstractions – MC parallel Application





Application Layer

- \+ Function and parallelism
- \+ Executable (SystemC)
- \+ Criticalities
- \+ Extra-functional assumptions and specifications
- \- Full target abstraction

Task Definition

Task $Ti \in \tau$,
$T_i = \left(\vec{T_i}, D_i, \vec{C_i}, \pi_i, L_i\right)$

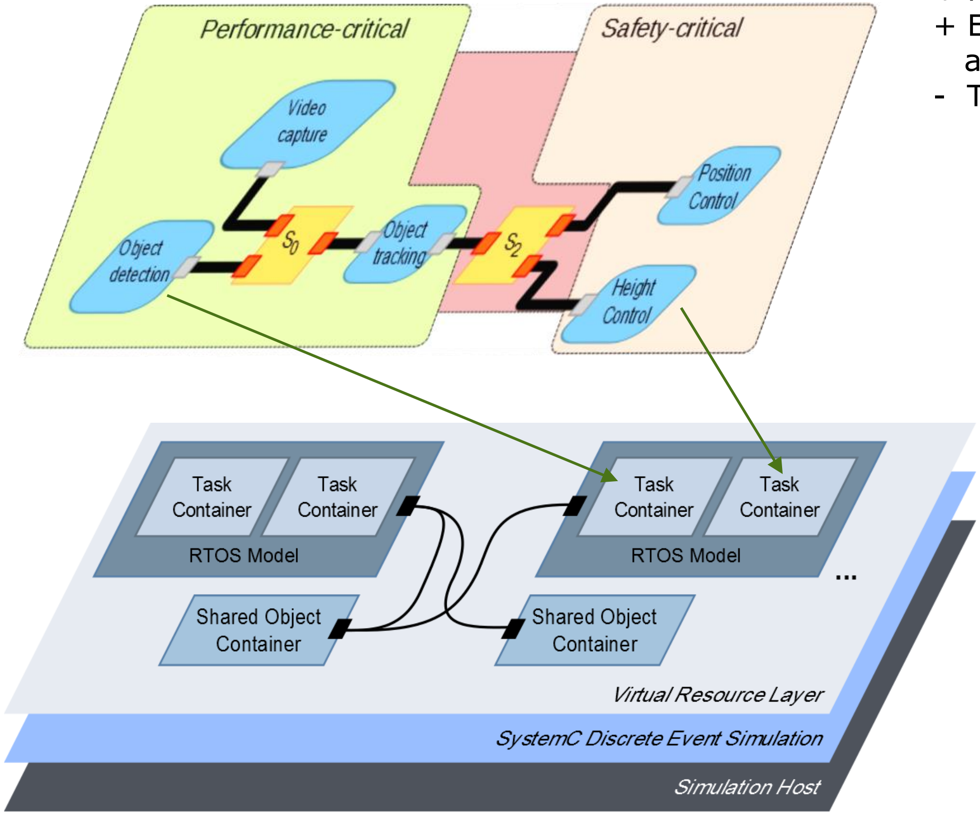
- a **vector** of periods $\vec{T_i}$ (minimum arrival interval)
- D_i: deadline
- $\vec{C_i}$: **vector of computation times** (one for each criticality level)
- π_i: **ports** for connecting to communication objects
- L_i: **criticality** level (e.g. *LO*, *HI*)

Communication Object

Communication Object
$S = (\Sigma, \Sigma', L, M, I, \Phi)$

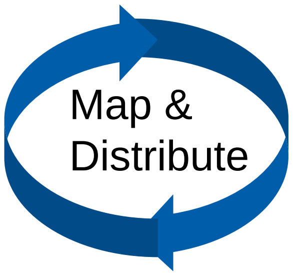
- $\Sigma_{0,1}$: **inner states** (containing abstract data types),
- L: current criticality level (e.g. *LO*, *HI*)
- $M \subseteq \Sigma \times \Sigma$: a set of **methods** or **services** (e.g. *read*(), *write*())
- $I \subseteq \mathcal{P}(M)$: Interfaces for grouping methods
- Φ: **resource arbitration policy**

4 Clear steps & strong abstractions - Application to Runtime Mode



Application Layer

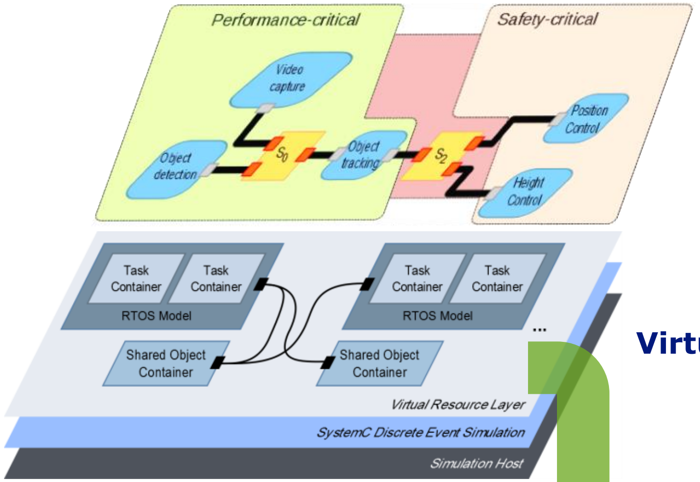
+ Function and parallelism
+ Extra-functional assumptions and specifications
- Target abstraction



Virtual Resource Layer

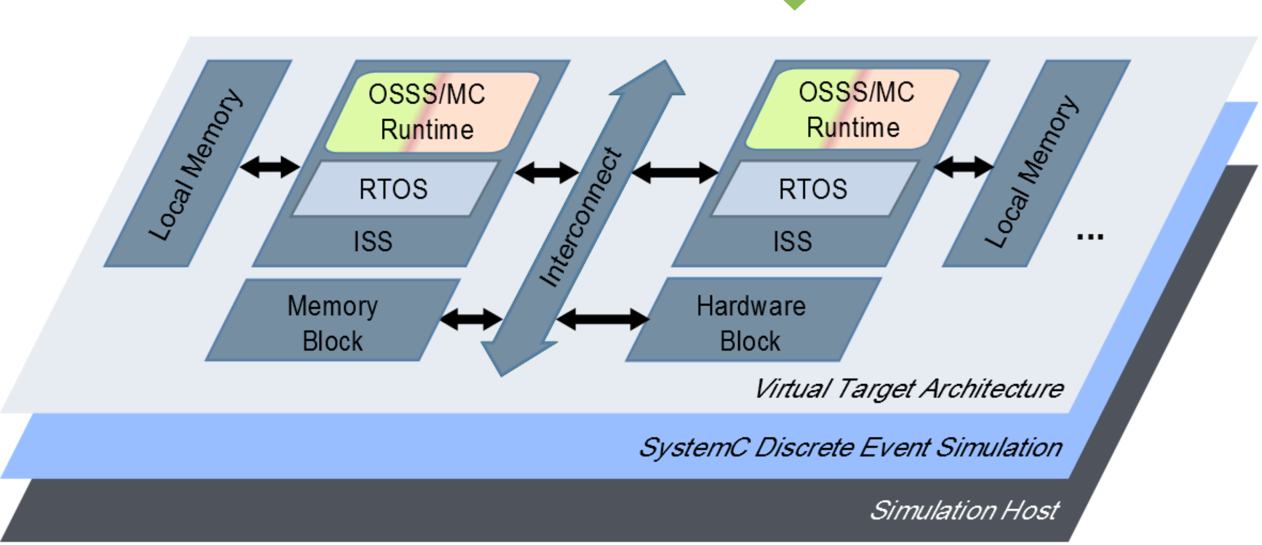
+ Hierarchical runtime system
+ Contention on shared resources
+ Scheduling effects

5 Clear steps & strong abstractions – Runtime to Target Modell



Application Layer

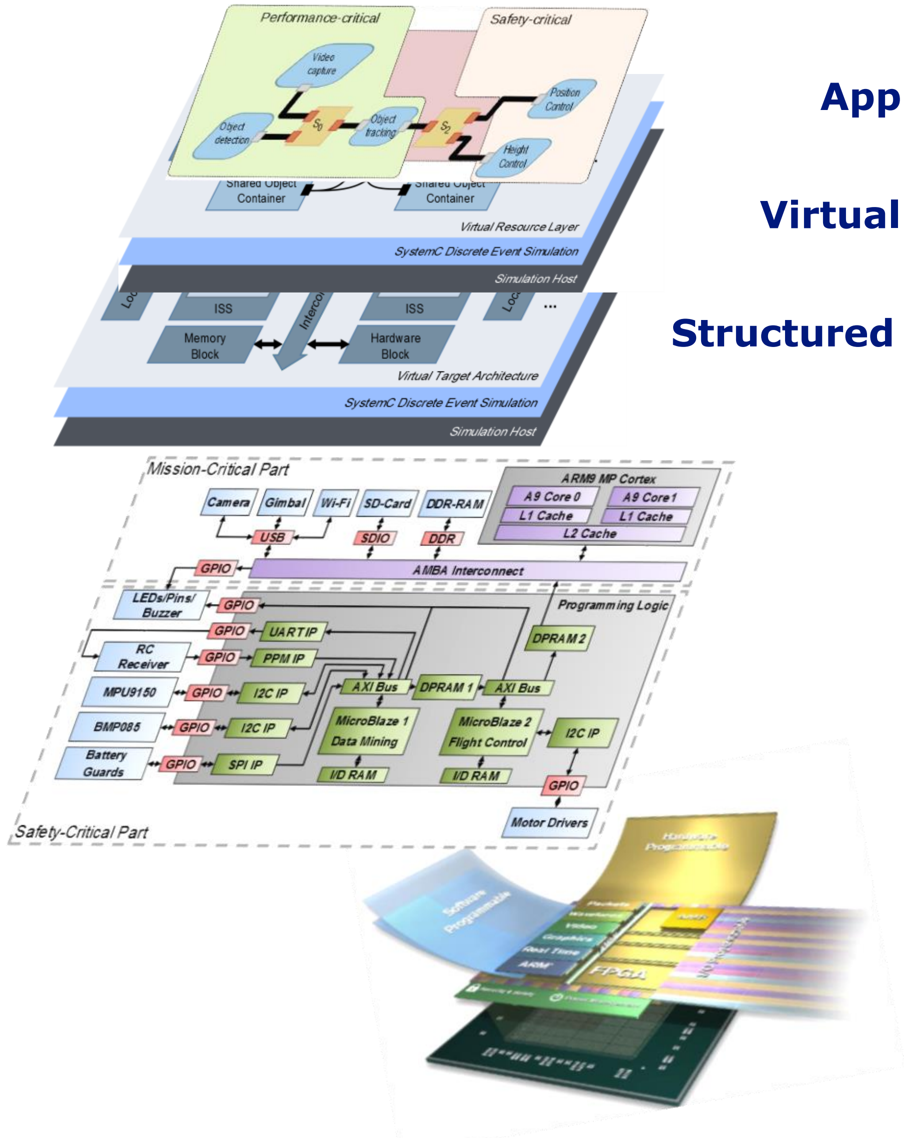
Virtual Resource Layer



Structured Target Architecture

+ Structure and topology
+ Mapping and sharing
+ Access paths (contention)
- Technology abstraction

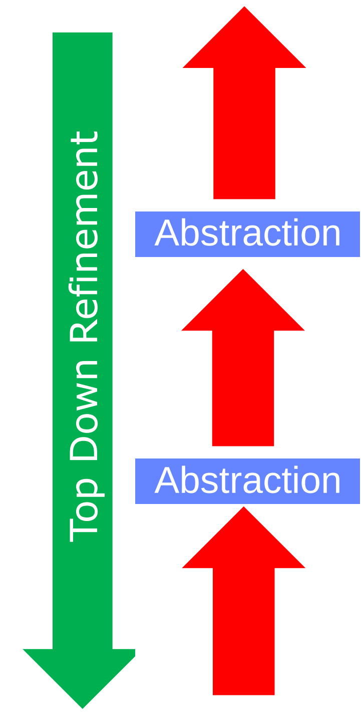
6 Clear steps & strong abstractions – TA to Physical Architecture



Application Layer

Virtual Resource Layer

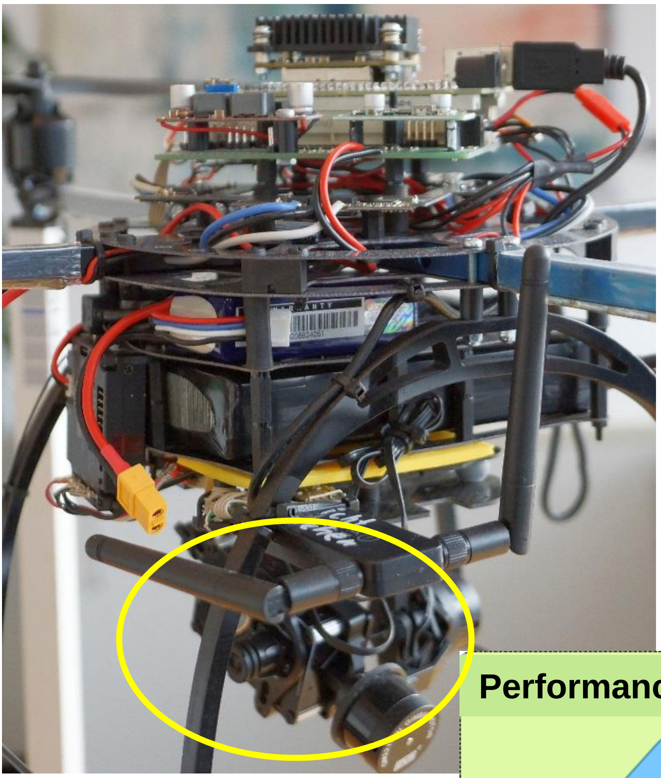
Structured Target Architecture

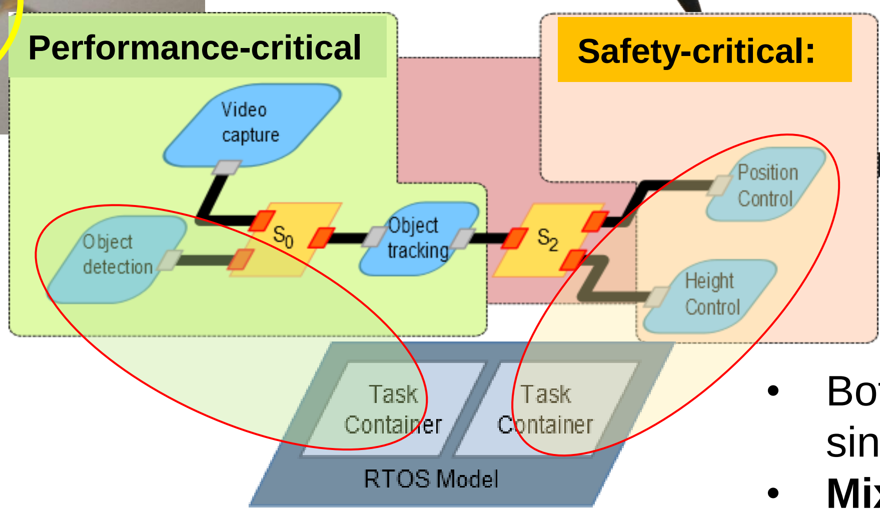


Physical Target Architecture

+ Implementation artefacts
+ Busses, Drivers, Arbitration
+ Caches, Bitwidth, Interupts, …

7 Evaluation : Quadrocopter – a mixed-criticality use case





- video-based object tracking,
- **needs** to run at 6 fps

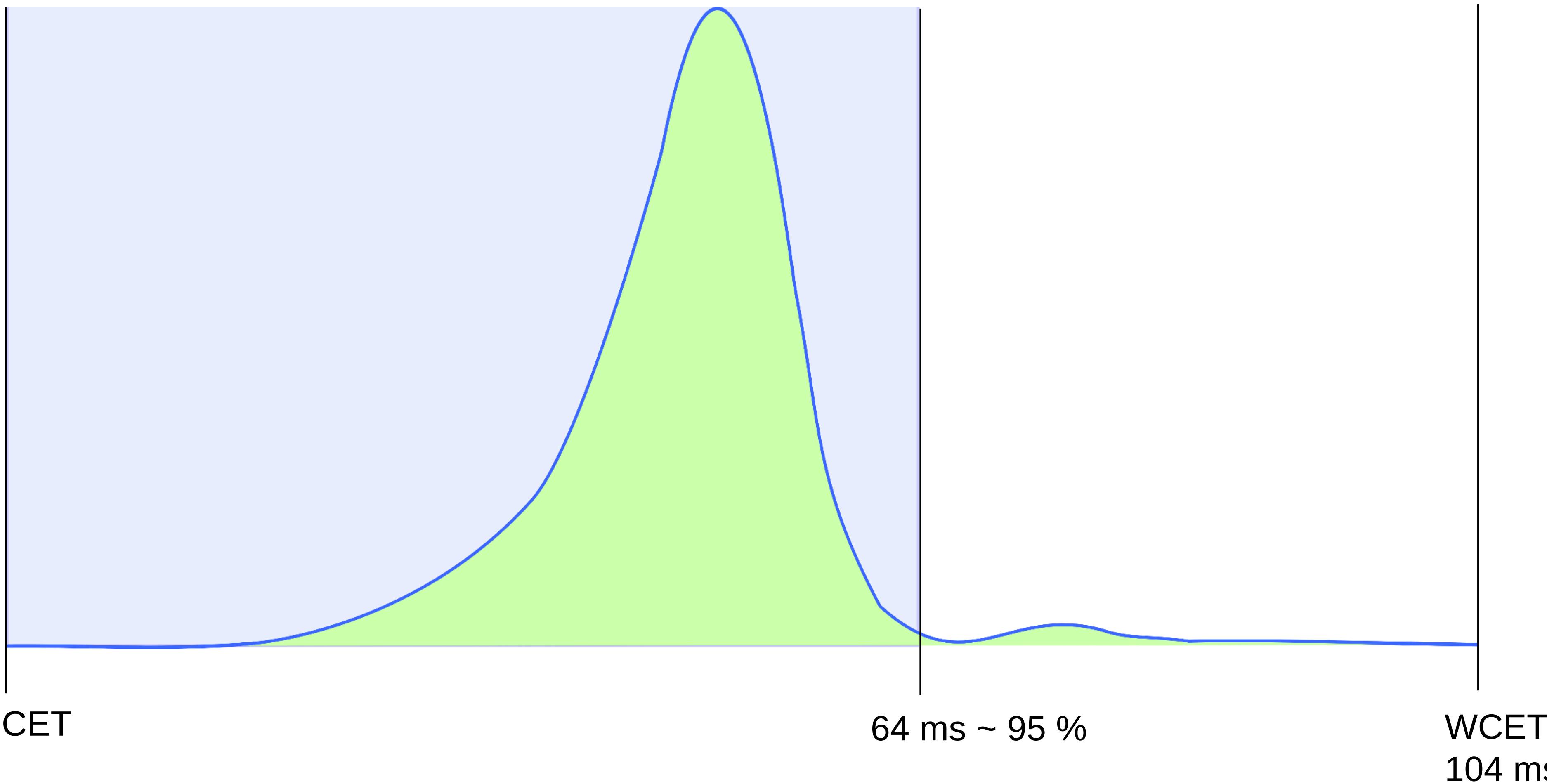
- flight control
- **essential** for stable flight control

- Both applications sharing a single core/RTOS
- **Mixed-Criticality**

8 Evaluation : Quadrocopter use case - static approach

- **Assumption**: Flight control and object tracking mapped to the same CPU core
- **Requirement**: Video processing needs to achieve 6 fps (167 ms per frame)
- **Statically** analyse WCETs
  - Safety-critical task set and determine utilization: 104 ms ≈ 62% each frame.
  - Leave slack for video processing: 300x200 px. (58 ms or ≈ 35% each frame)
- **Analysed (static) total system utilisation: 97%**

9 Why use dynamic criticalities? Distribution of execution times !

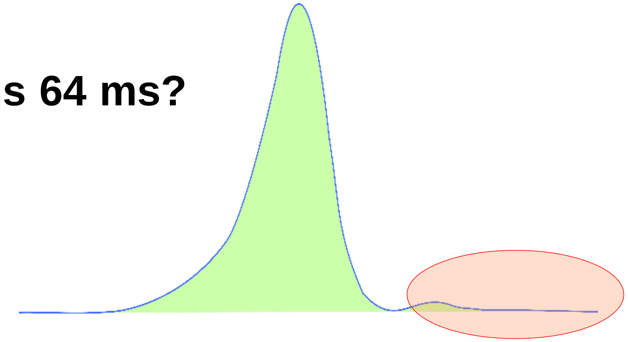


Observation: Flight control: 95 % of all executions are lower that 64 ms, leaving 103 ms (167 – 64) for better object tracking.

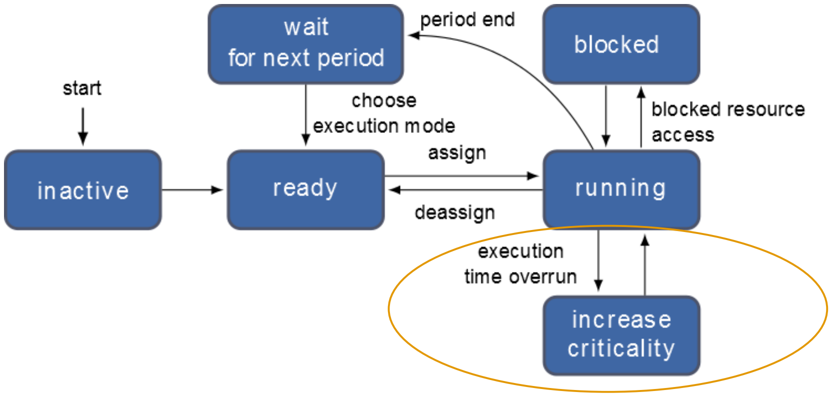
10 Quadrocpoter with dynamic mixed-criticality

- **Define two criticality levels for camera based object detection**
  - DEGraded (high criticality): Video frame of 300x200 px. using 58.1 ms (measured)
  - Full Quality(low criticality): Video frame of 460x320 px. using 93.3 ms (measured)

- **But in those critical 5%, when flight control overruns 64 ms?**

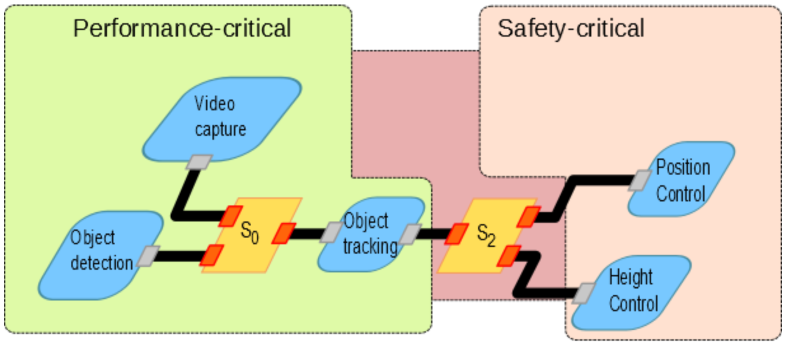


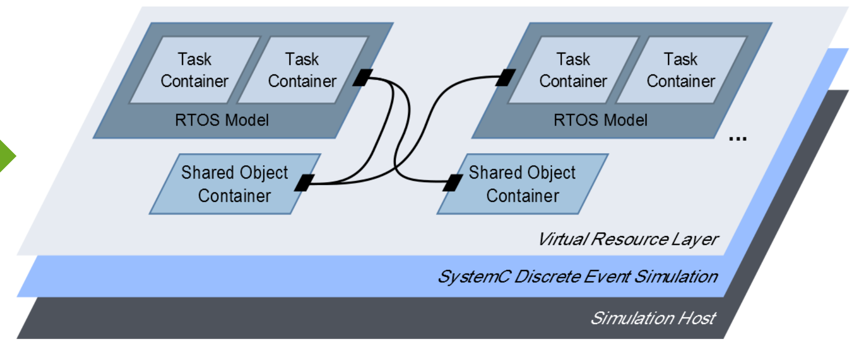
- **Criticality switches based on run-time monitoring**:
  - At overrun increase criticality
  - At underrun decrease criticality



11 Evaluation Setting

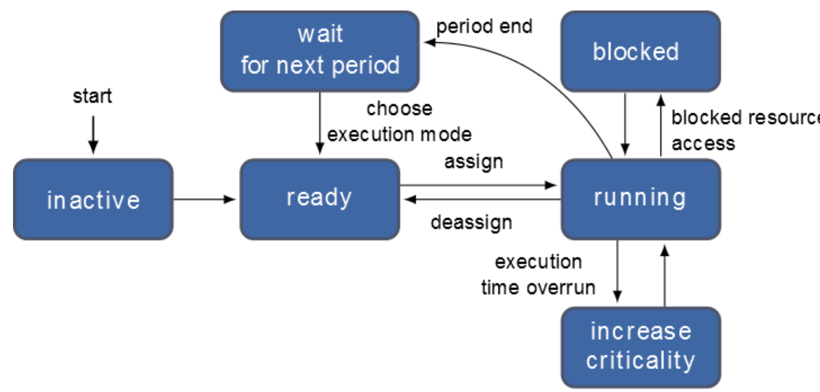
Application Layer Model + Virtual Resource Layer



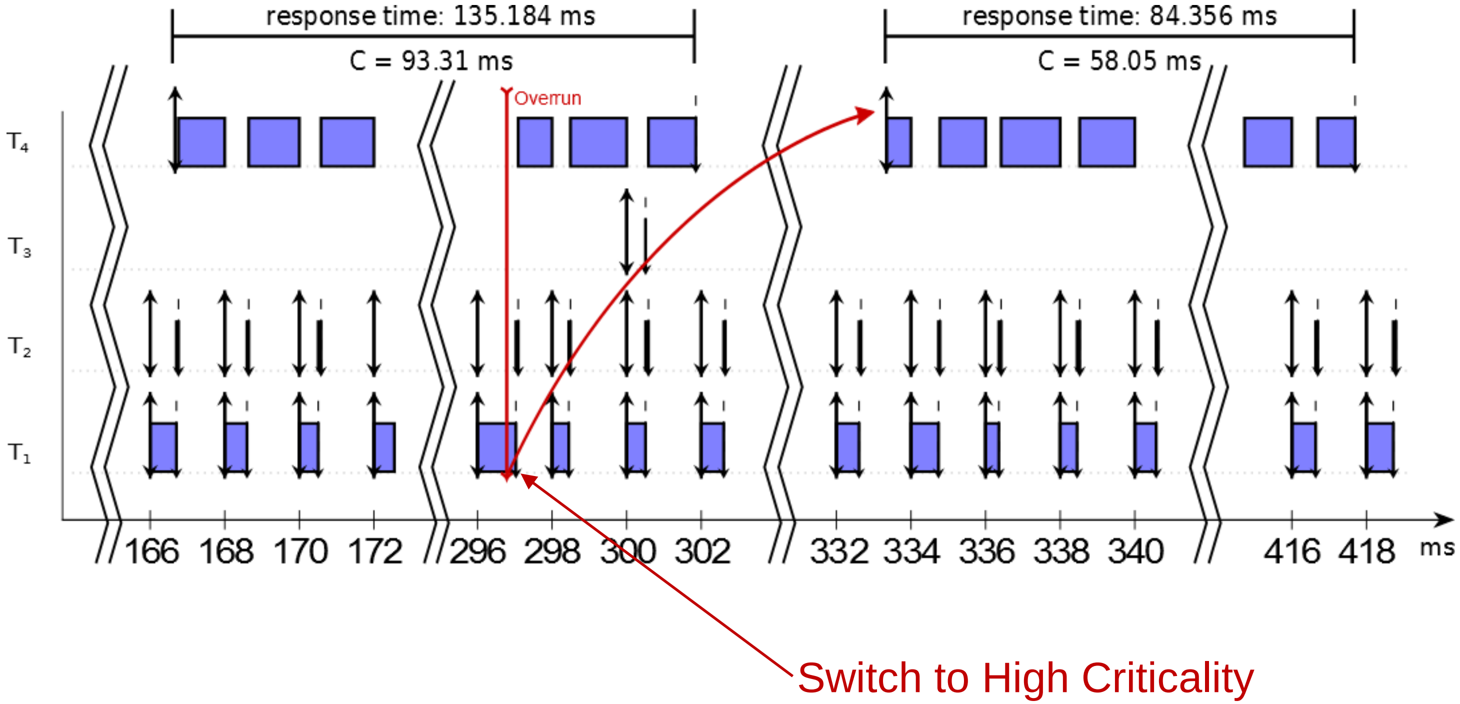



```
// Task properties
set_period(  _MS(167));
set_eet(DEG, _MS(104));
set_eet(FuQ, _MS(64));
set_criticality(FuQ);
set_priority(1);
...

void flight_control(){
// Gauss randomized execution time
OSSSMC_EET(gauss(_MS(104)){
// compute next value
}
```



12 Results



Criticality Policy	# Degraded	# Full Quality	System utilisation [%]
Static	30	0	65.54 (±0.16)
Dynamic	13 (±3)	17(±3)	86.70 (±0.14)

▸13 Conclusion

- **Application modelling for parallel platforms**
  - Platform aware modelling of computation, communication and data
  - Real-time and scheduling at application level
- **Early assessment of crucial design decisions**
  - For choosing the most efficient target platform
  - For managing trade-offs
- **Stack of abstractions to validate and verify**
  - Functional behaviour and QoS
  - Segregation for mixed-critical integration
  - Meeting extra-functional properties (e.g. time/power)
- **Evaluation example dynamic Criticalities switches**
  - Allowing for a better QoS
  - Guaranteeing critical deadlines