

Welcome to

# OSPERT 2016

12<sup>th</sup> workshop on Operating Systems Platforms for Embedded Real-Time Applications  
July 5<sup>th</sup>, 2016, Toulouse France

in conjunction with



the 28<sup>th</sup> Euromicro Conference on Real-Time Systems  
July 6–8, 2016, Toulouse, France

Robert Kaiser, Marcus Völp

# Numbers for the Crunchers

**OSP**ERT 2016,

solicited diverse types of contributions, including regular and short papers as well as experimental studies.

- regular workshop papers: up to six pages
- short papers: up to three pages, intended for reports on work in progress, project status reports, and replication studies

Received 15 submissions

- 10 regular papers
- 3 short papers
- 2 experimental studies

Accepted 9 papers

- 8 regular papers and one short paper
- at least 3 independent reviews per paper

# Many thanks to the program committee!

Andrea Bastoni, SYSGO AG

Michael Engel, multicores.org

Paolo Gai, Evidence Srl

Shinya Honda, Nagoya University

Adam Lackorzynski, Kernkonzept / TU Dresden

Daniel Lohmann, FAU Erlangen-Nuernberg

Chanik Park, Pohang University of Science and Technology

Pavel Pisa, Czech Technical University Prague

Linh Thi Xuan Phan, University of Pennsylvania

Richard West, Boston University

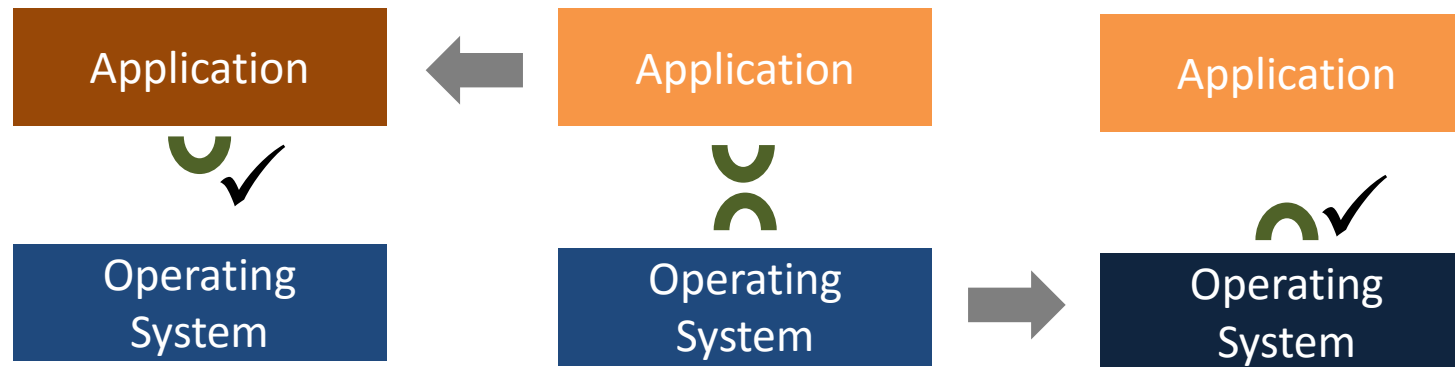


# Open Research Challenges in Real-Time Operating-Systems

# Discussion on Open Research Challenges in Real-Time Operating Systems

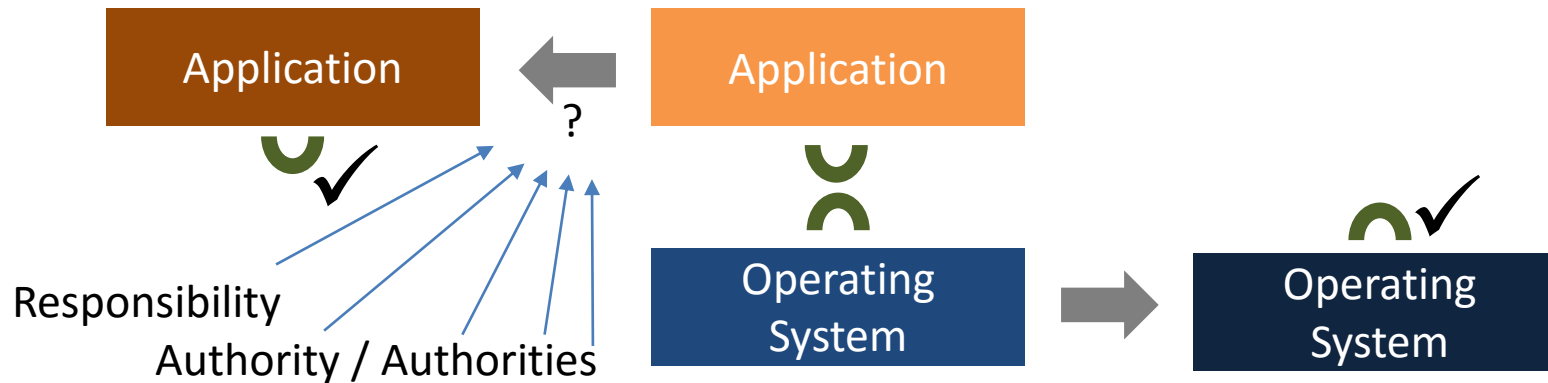
- Automating re-verification through OS – application contracts *Johannes Schlatow*
- Security and Dependency issues when upgrading real-time systems *Mohammad Hamad*
- *How should SMP RTOS ( $\mu$ )kernels look like?*
- Real-time systems under attack; time plane attacks *Marcus Völp*

# Challenge 1: OS – Application Contracts, *Johannes Schlatow*



For the integration of real-time applications, several model-based methods have been established that allow verifying the correct (real-time) behaviour of the system. These models are typically based on certain assumptions on the RTOS behaviour, such as scheduling policy, context switch overhead, or temporal/spatial isolation. It is obvious that the quality (and validity) of those models highly depend on the accuracy of these assumptions. This not only requires an in-depth understanding of the OS but may also conflict with OS upgrades that slightly change its functional or non-functional behaviour and hence necessitate re-certification or re-verification of the entire system. As those upgrades are undoubtedly necessary for long-term maintenance of embedded real-time systems, we think it'd be worth investigating methods that formalise the guarantees provided by the OS, e.g. in terms of a contract interface between OS and applications. Such an interface would not only reduce the (re-)verification effort but also allow for an automated approach to this.

# Challenge 2a: Upgrade authority, *Mohammad Hamad*



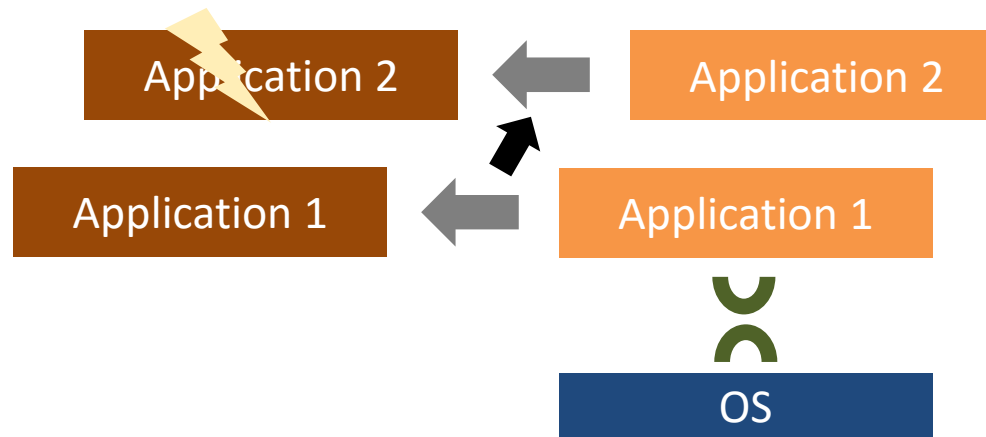
## Authority:

Not only applications require upgrading, but OS itself need to be also upgraded. In this manner, we need to answer the big question: "Who is authorized to upgrade the OS?"

One answer could be the manufacturer of the platform! But what will happen if this manufacturer does not exist anymore? Who should take his responsibility?

Similarly, we could ask this question regarding the upgrade of an application. Some application could require the authorization from various trust authorities. In this case, how could we handle the transfer of authority?

# Challenge 2b: Upgrade dependencies, *Mohammad Hamad*



Dependency:

Most of the applications depend on/collaborate with other application or services to handle its work (e.g. lamp and light switch). Upgrading one application could require upgrading of other services or applications, which could cause the domino effect of upgrading.

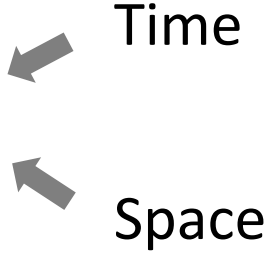
The failure of upgrading one component (for some reasons ) could leave the system nonfunctional (e.g. updated server uses a new version of a communication protocol while a client still uses an old version). The downgrade of the server, in this case, to retrieve the functionality could open a security vulnerability!

The OS should guarantee a mechanism to handle the conflict of a sequential upgrading of the different components with the constraint of RTOS.

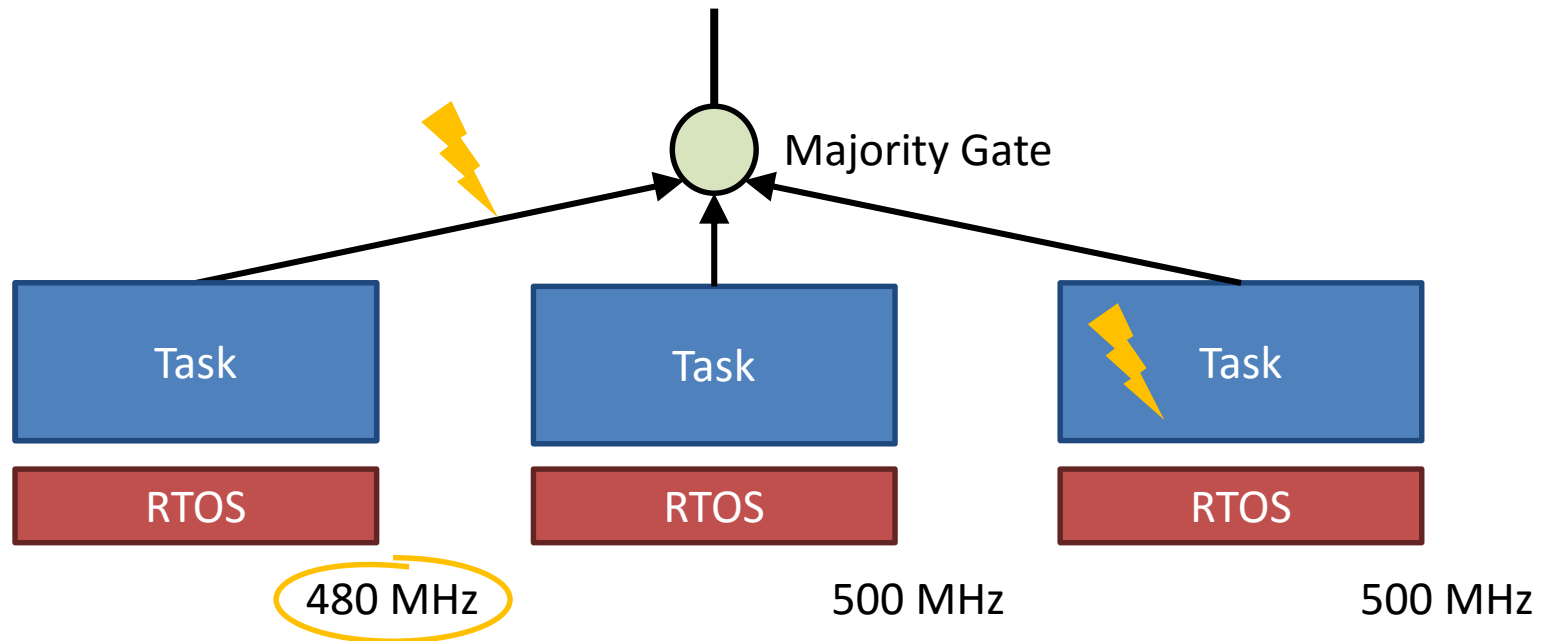


# Challenge 3:

## How should SMP RTOS ( $\mu$ )kernels look like

- Isolation
    - Memory
    - Devices
    - NoC
  - Task Container
    - Address space
    - Virtual machine
    - ???
  - Task Interaction
    - IPC
    - asynchronous
  - Scheduling
    - whole system
    - subsystem
    - both? / other?
- Time
- Space
- 

# Challenge 3: Time plane attacks, *Marcus Völp*



- How can we survive attacks to real-time / cyber-physical systems?
- How can we preserve timeliness while under attack?
- How do attack countermeasure and real-time systems interplay?
- What mechanisms do we need in the RTOS / hardware / elsewhere to create dependable real-time systems that are robust against attacks?

# Closing Remarks

# OSPERT 2016 Collections

## Adam:

History and some future of L4.Fiasco

=> How to survive 5 years (and hopefully longer) with your microkernel company

## Hendrik:

- DSL for expressing Hardware configurations

=> Generate OS from DSL?

## Mohammad:

- Communication security policies in  $\mu$ K-based ECUs

=> Proxy versus distributed capabilities (c.f., Amoeba)

## Benjamin:

- “A clever trick to role back critical section state in RT locking protocols” [bbb]

=> How much do we rely on the (partial) correctness of low critical tasks?

## Nathan:

- GPU-Lite: or the desperate task of convincing NVIDIA to run more than one kernel at a time.

=> Is NVIDIA's scheduler simply broken or should they just hire a few of us?

# OSPERT 2016 Collections

## Johannes: Open Problems

- Contracts to allow separate verification and re-verification
- => Related work in Model-Checking community e.g.,  
[http://dx.doi.org/10.1007/978-3-662-49665-7\\_17](http://dx.doi.org/10.1007/978-3-662-49665-7_17)

## Mohammad:

- Long term responsibility for software
- => We can't simply stockpile programmers for the next 20 years
- Partial reconfiguration
- => Probably less severe once long term responsibility is solved:
- compatibility layers to non-upgradeable components;
  - no partial upgrades

## All:

- The next generation RTOS interface

## Brief mentioning:

- Time plane attacks

## Papers

### Adam:

- HPC needs predictability as in real-time (even if they don't know it)
- Split applications help again

### José:

- How to survive in space
- => Hardware is not always the solution

# OSPERT 2016 Collections

## Geet:

- (Re-)configuration to tailor our code base to what we actually need.  
=> It's open source ;)

## Alessandro:

- dynamic partial reconfiguration  
=> real-time applications to bring along their own hardware  
=> no wear leveling needed; SRAM based FPGAs  
=> a lot of challenges remain (a lot of papers for future iterations of OSPERT ??)

## Reinder:

- What if we actually implement FSLM?

# Really the last things

Many thanks to the authors and the PC for an interesting day!

OSPERT'17 deadline likely end of April, 2017. Submit early, submit often!

*Consider submitting replication studies or other experimental studies. Get early feedback on your work-in-progress RTAS submissions.*

Feedback and suggestions for next year?