

INSTITUTE
OF COMMUNICATION,
INFORMATION
AND PERCEPTION
TECHNOLOGIES



Scuola Superiore
Sant'Anna

Multicore implementations of the Logical Execution Time paradigm

Alessandro Biondi, Marco Di Natale

Scuola Superiore Sant'Anna, Pisa
September, 2017



Summary

Motivation: Strong industrial push and the **WATERS17 challenge**

Subject: Implementation of the *logical execution time* (**LET**) model in **AUTOSAR**

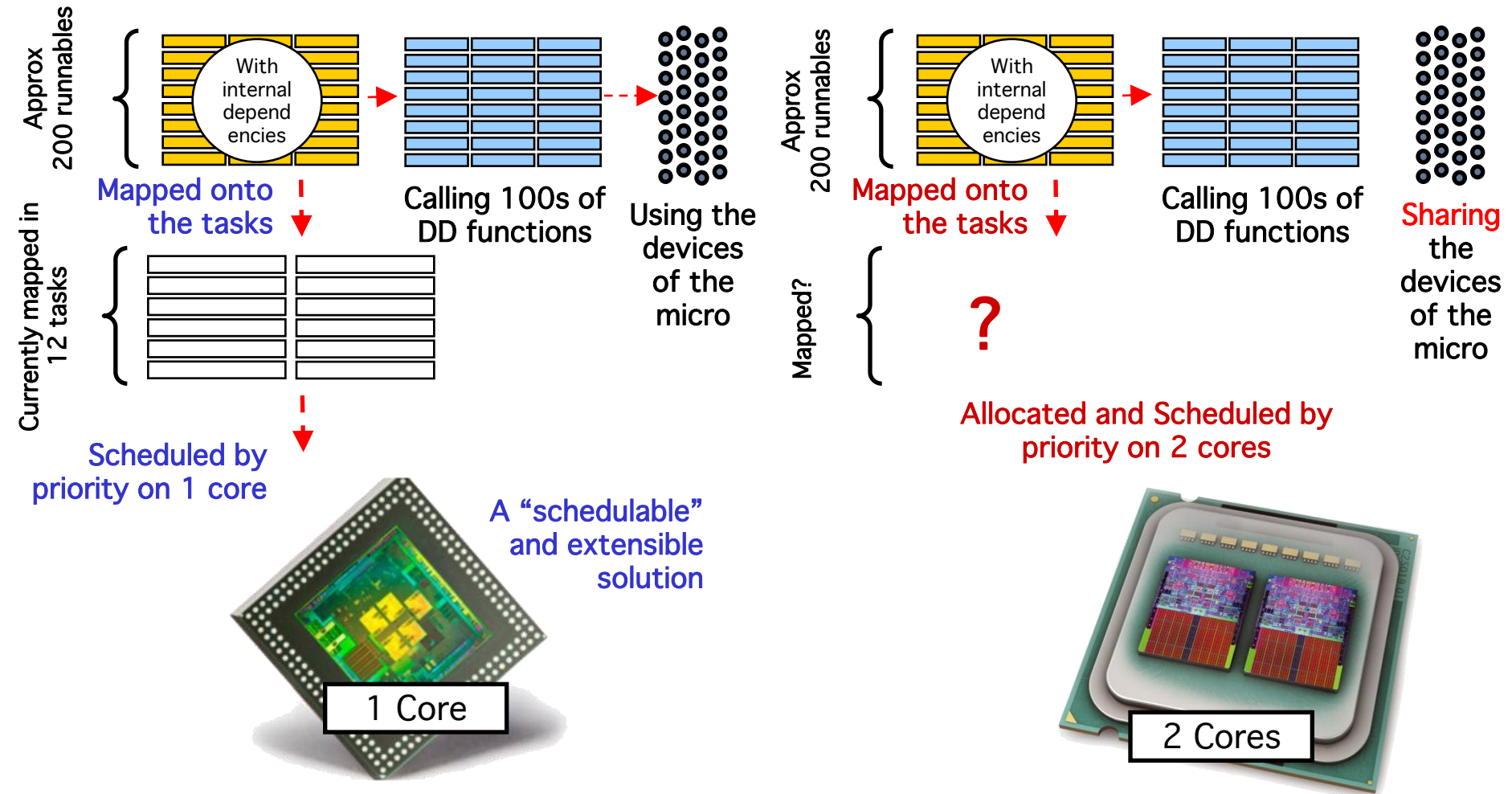
Analogies with the single-core equivalence model

response-time analysis is affected by **memory latencies** (access + contention) which is reduced by LET (see Waters)

Optimization algorithms for **label placement**

One instance of the problem


Porting engine control applications from single- to dual-core.



Examples of the problem

Consider the WATERS challenge 2017

<https://waters2017.inria.fr/challenge/>



The screenshot shows the WATERS 2017 website. At the top, it says "WATERS" in large blue letters, followed by "8th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems" and "27th June 2017, Dubrovnik, Croatia". Below this is a navigation menu with items: Home, Call for contributions, Organizers, Submission instructions, About WATERS, Industrial challenge, WATERS Community forum, and Program. The main content area is titled "Industrial challenge" and is in conjunction with ECRTS. It features sections for "Important dates" (Submission deadline: 14th April 2017, Acceptance notification: 8th May 2017, Early registration deadline: 12th May 2017, Final version deadline: 22th May 2017, Workshop: 27th June 2017), "previous editions" (WATERS 2016 to 2010), "About the WATERS industrial challenge" (The purpose of the WATERS industrial challenge, formerly called Formal Methods for Timing Verification (FMTV) challenge, is to share ideas, experiences and solutions to concrete timing verification problems issued from real industrial case studies. It also aims at promoting discussions, closer interactions, cross fertilization of ideas and synergies across the breadth of the real-time research community, as well as attracting industrial practitioners from different domains having a specific interest in timing verification.), and "The 2017 industrial challenge" (We are glad to announce an updated version of the 2016 industrial challenge proposed by Arne Hamann, Simon Kramer, Michael Pressler, Dakshina Dasari, Falk Wurst, and Dirk Ziegenbein from Robert Bosch GmbH. Compared to last year's challenge the following aspects have been added and changed: Detailed description of the hardware model with respect to communication costs (best- as well as worst-case models for read and write accesses to the global and local scratchpad memories); Explanation of two communication semantics and their role in the design of industrial embedded systems:).

Engine control application already deployed on a 4-core
2016 challenge asked for methods for timing analysis
2017 challenge asked for consideration of the LET paradigm and optimization of placement in memory of communication data

Data is provided as an AMALTHEA DSML model (in practice AUTOSAR)

Interesting placement of tasks on cores (2 cores for IO, one for rate dependent and very high rate, the other for all periodic tasks)

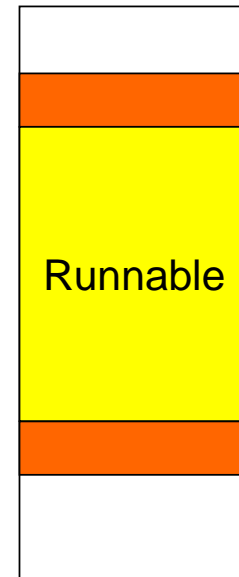
We provided our own solution (MILP / GA) with interesting outcome

AUTOSAR Implicit communication

Consider an `Rte_IWrite` API that sends `VariableDataPrototype` `D` via port `P` and an `Rte_IRead` API that reads `VariableDataPrototype` `E` via port `Q`.

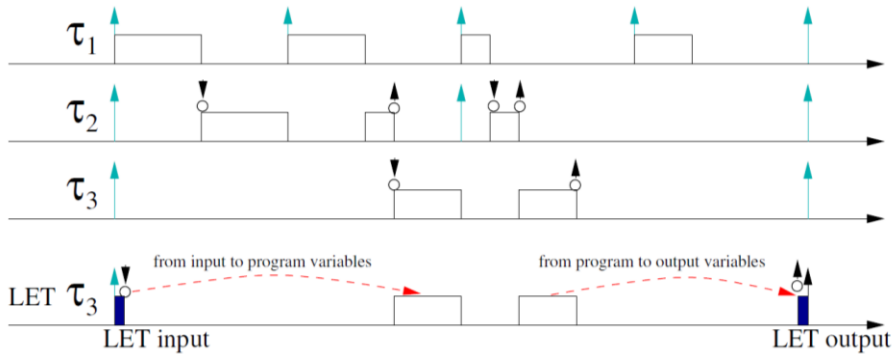
```
TASK (...)  
{  
    volatile <type> local_P_D;  
    volatile <type> local_Q_E;  
    /* ... */  
    local_P_D = global_P_D;  
    local_Q_E = global_Q_E;  
    Runnable();  
    global_P_D = local_P_D;  
}
```

Task



LET on multicores

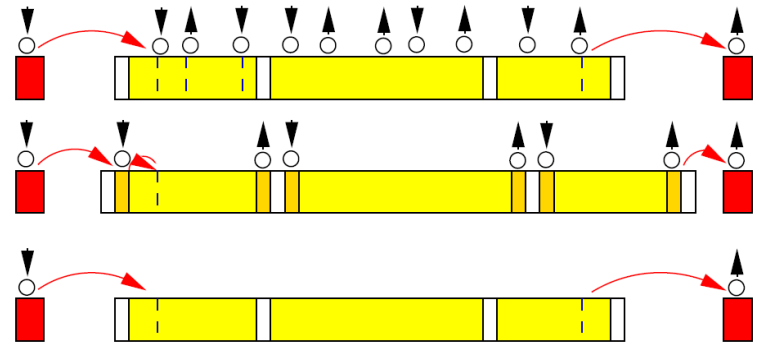
WATERS challenge 2017 ...



LET also brings similarity with the AUTOSAR RTE immediate communication model

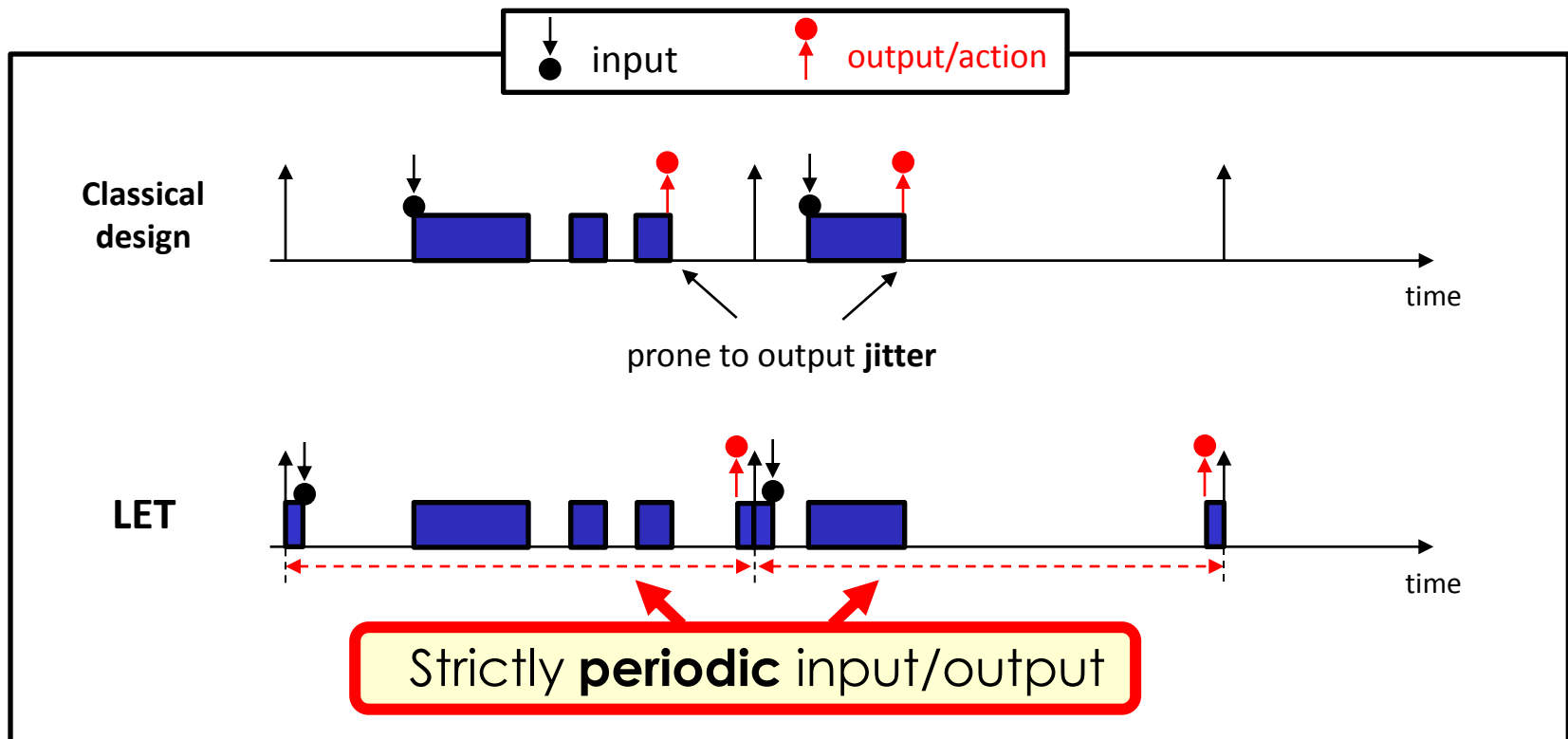
(Kirsch et al ?) Tasks input data at the beginning of their period and output is delayed until the end of the period (trade output jitter for delay)

Also improves determinism in the access to memory !



Logical Execution Time

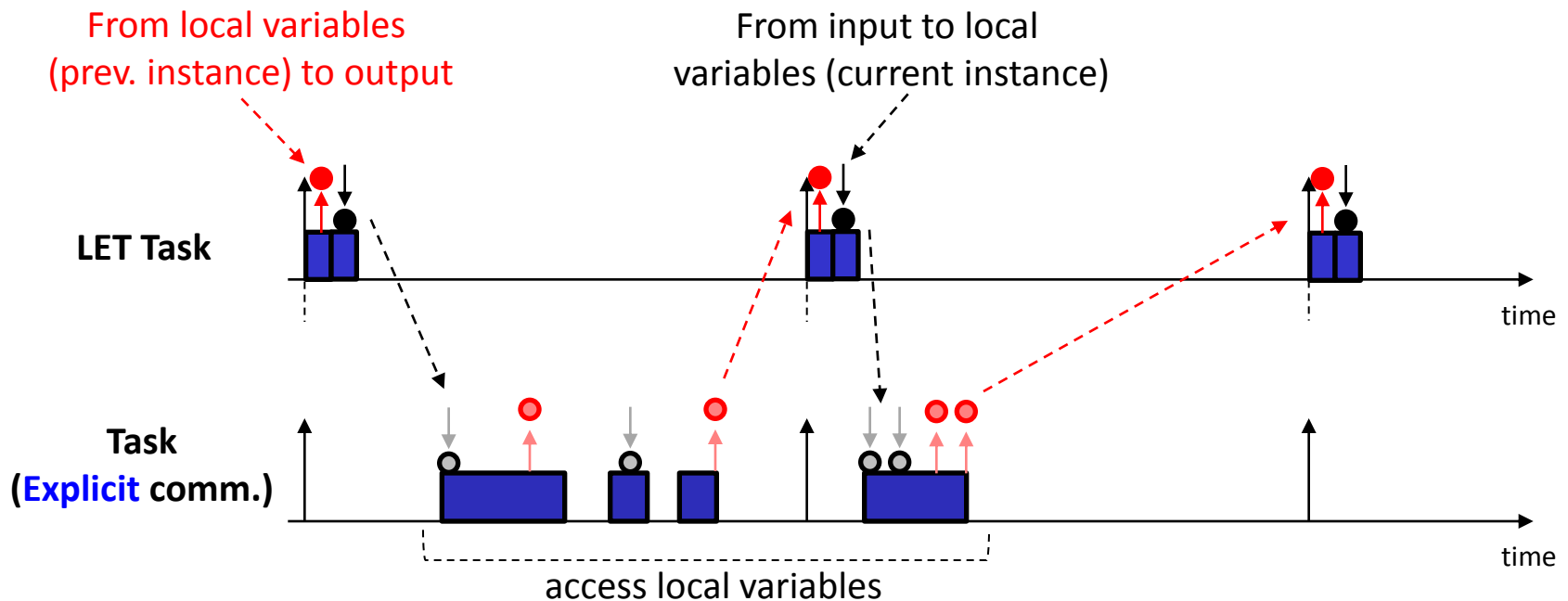
- Logical Execution Time (**LET**) eliminates output **jitter** and provide **time determinism** in the implementation of control algorithms



LET Implementation in AUTOSAR

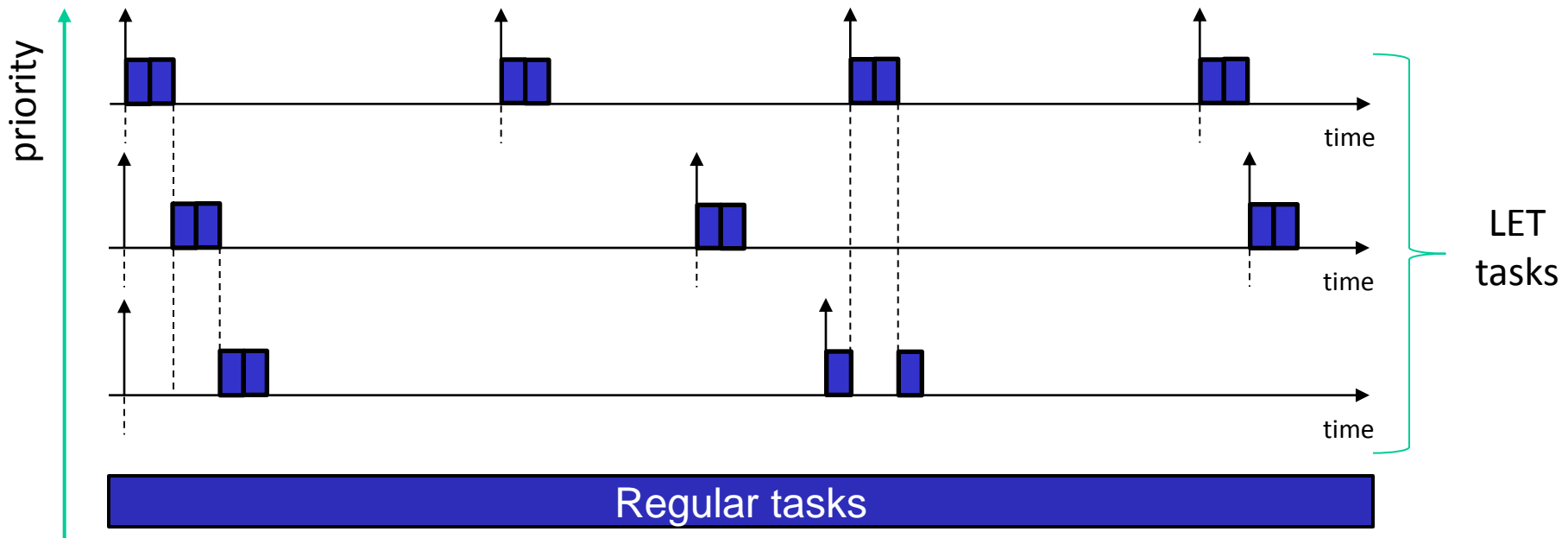
- LET can be implemented with **additional runnables** with *precedence constraints* w.r.t. the corresponding tasks
- Runnables of the same rate can be merged into a **LET task**
- Need for **duplicate variables** which may introduce additional *memory content* and *explicit communication*

This design can be **integrated** as part of the AUTOSAR **run-time environment (RTE)**



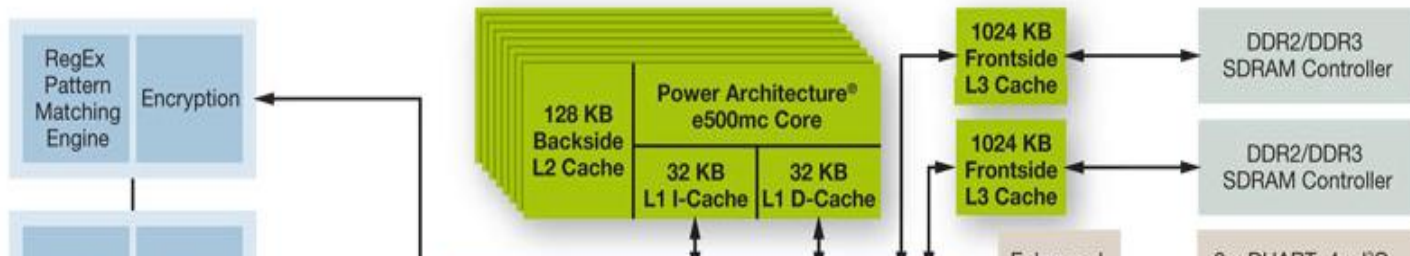
LET tasks

- LET tasks should be executed with the **highest priority** levels (possibly with relative *rate-monotonic* order)
- Limited **jitter** is anyway unavoidable
- **Precedence constraints** are automatically enforced by fixed-priority scheduling



Other considerations: Multicores

QorIQ™ P4080 Block Diagram



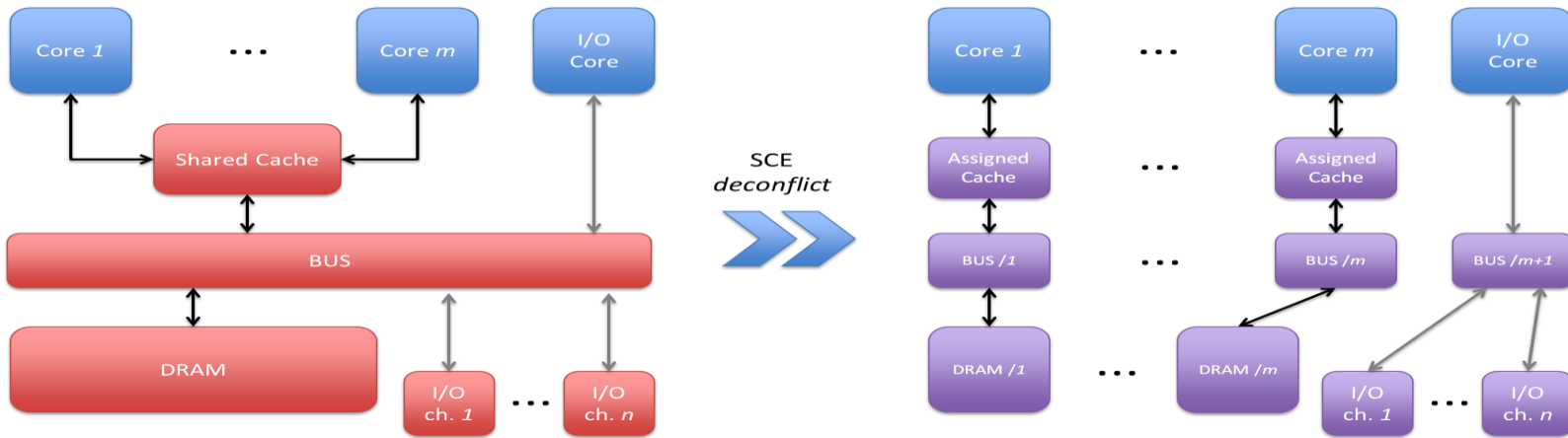
Multicore architectures were not designed or verified for compliance with safety critical systems

- Freescale P4080 (development partly driven by Bosch)
- In general
- Software executing in different cores of a multicore chip can severely interfere with each other due to shared hardware resources (e.g., cache, memory, I/O channels, etc.).
- Worst-Case Execution Time (WCET) of tasks directly depends on the number of active cores m .

By M. Caccamo – Univ. Of Illinois

Other considerations: Multicores

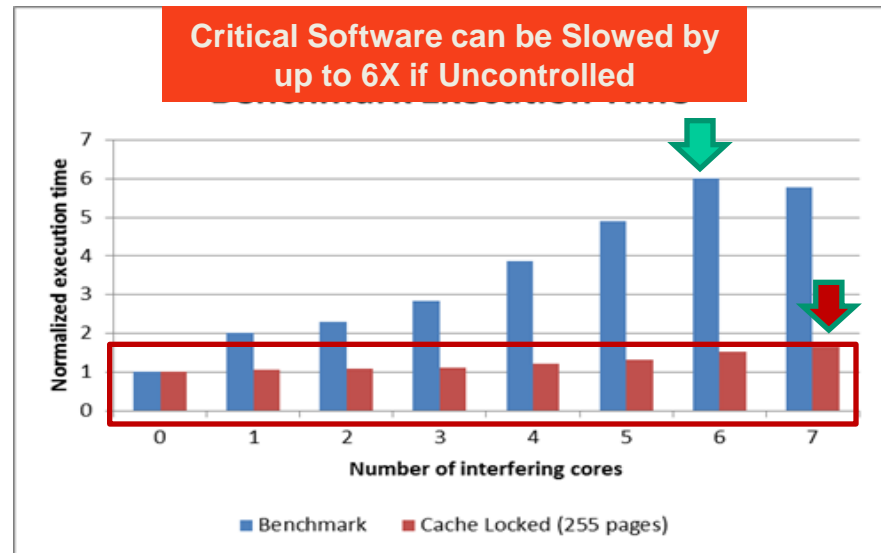
- The single-core equivalence projects at University of Illinois



Single Core Equivalence (SCE)

<http://rtsl-edge.cs.illinois.edu/SCE/>

Source: Lockheed Space Systems
HWIL Testbed



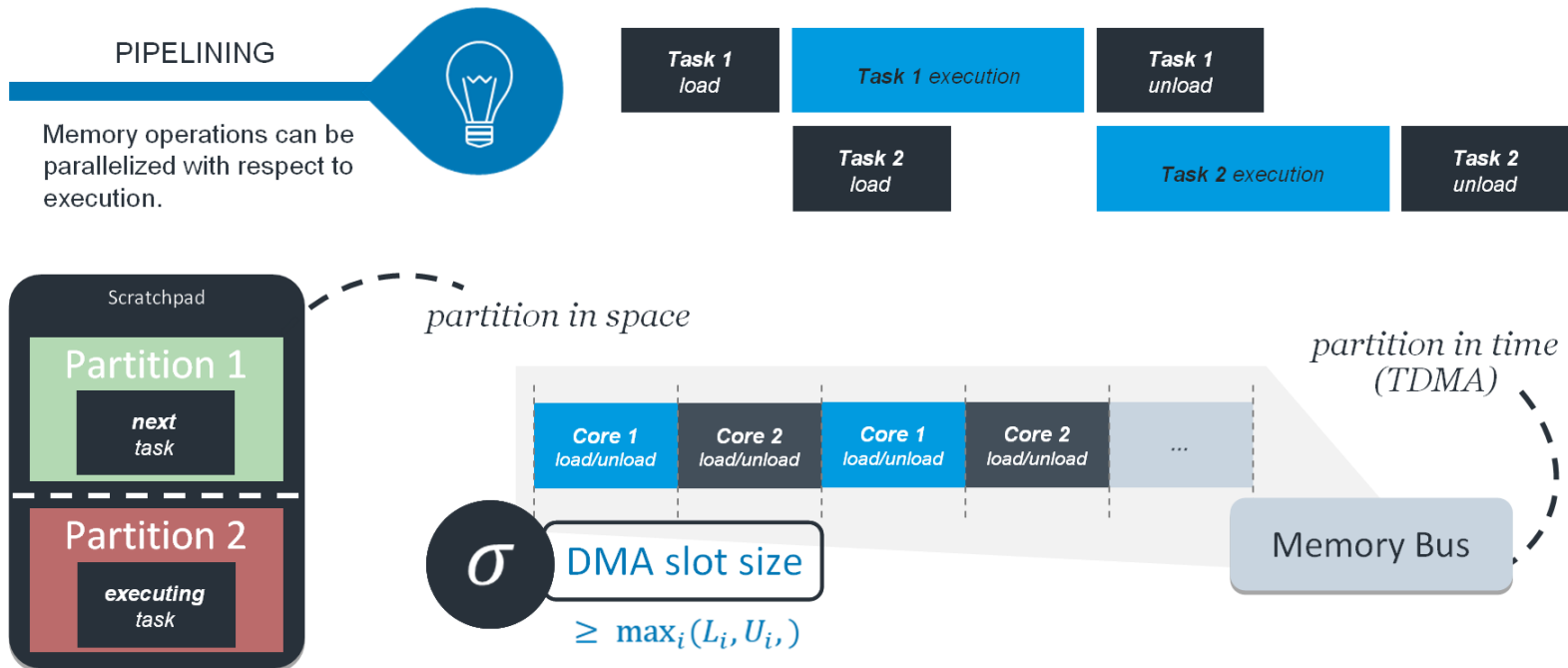
Issues with determinism of multicore platforms

Cache coloring (pinning), cache and local memory preloading

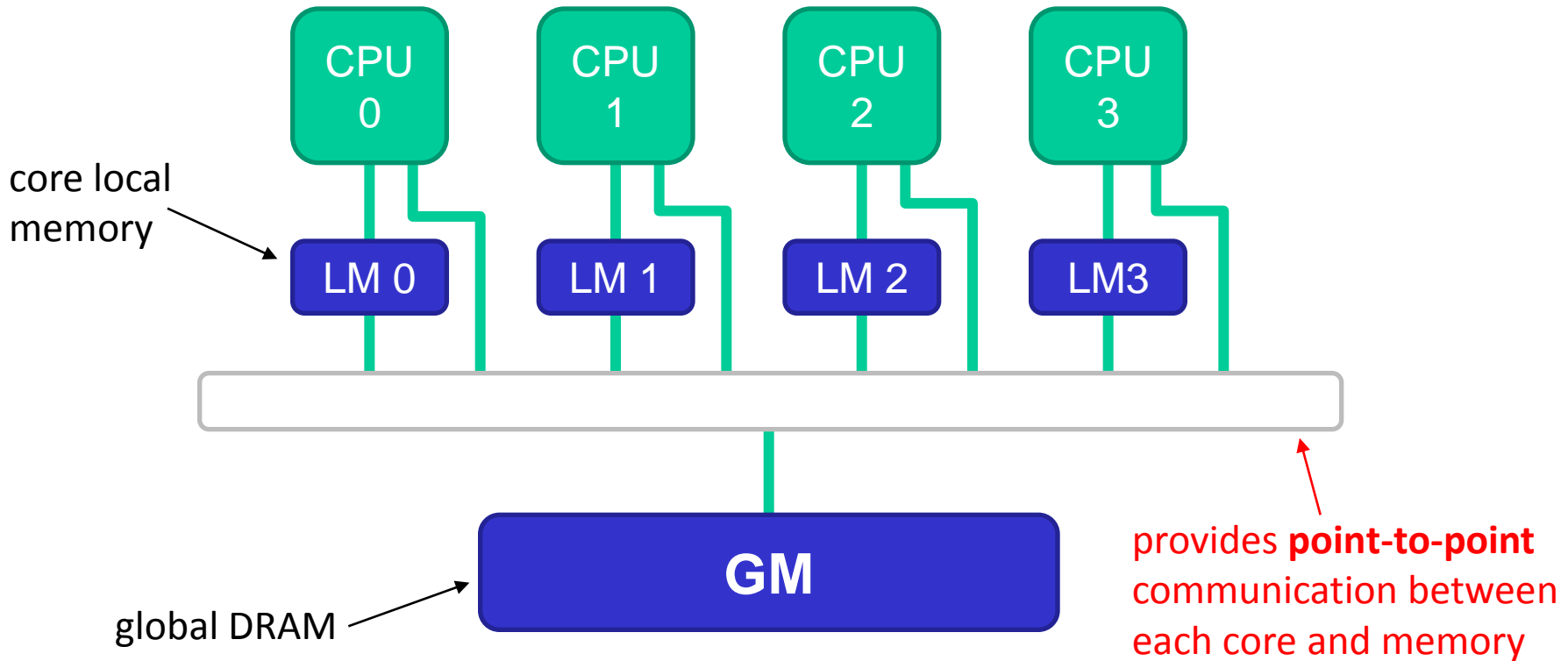


SPM-CENTRIC OS

Pipelining and memory bus scheduling



Back to the Waters challenge



MEMORY LATENCIES

Problem: How to **bound** memory access latencies to perform *response-time analysis*?

- **FIFO** arbitration is *starvation-free* and memory accesses are *non-interruptible*
- **FIFO invariant:** each memory access is **delayed** by *at most one* memory access per *remote processor*
- Naïve approach: *inflate* tasks' WCETs with a **coarse bound** for each memory access

$$\text{bound} = \langle \text{access cost} \rangle + (m - 1) \times \langle \text{contention cost} \rangle$$

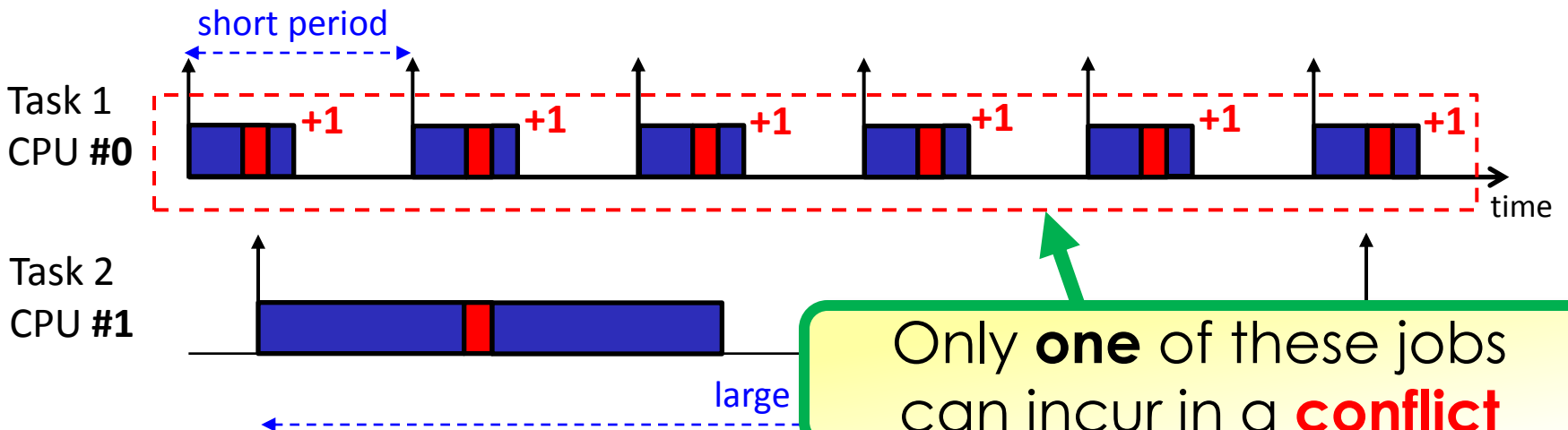
↑
number of CPUs

SOMETHING BETTER IS NEEDED

The **naïve approach** may result **very pessimistic**

- WCET inflation typically originates *multiple* sources of **pessimism** (e.g., see *Wieder and Brandenburg, RTSS13*)
- It may **completely hide** the **real problem** of **label placement**, as multiple solutions would result in the same latencies

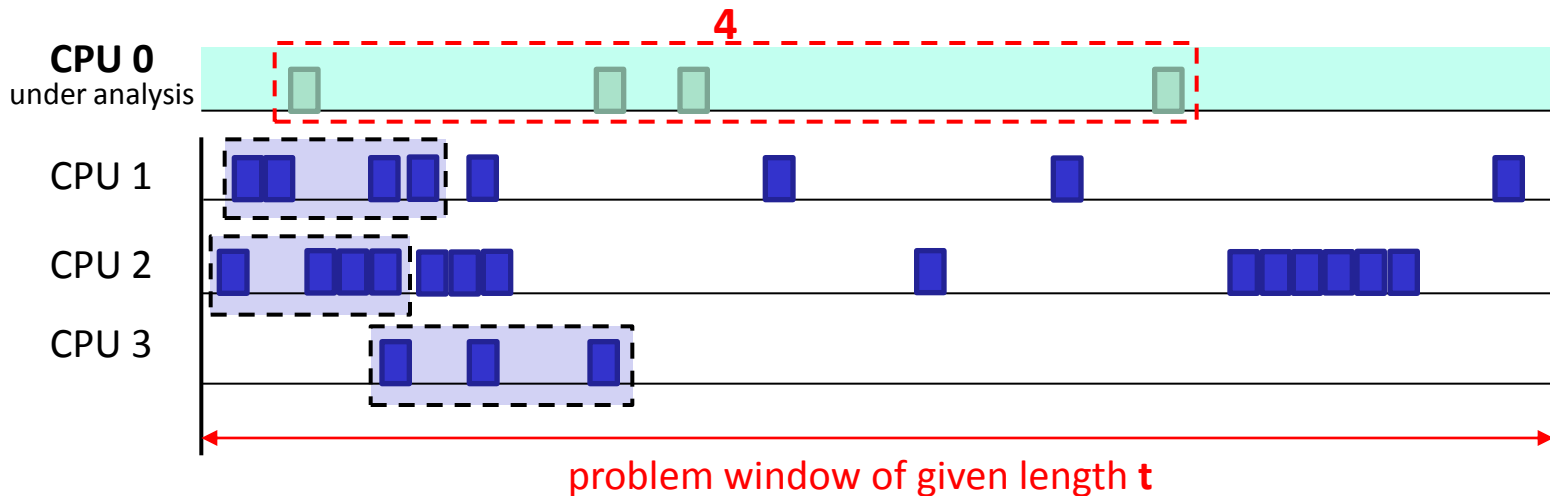
Example (source of pessimism)



PROPOSED APPROACH

- **Analysis design principles:**

- Do **not inflate** tasks' WCETs
- **Explicitly account** for memory contention at the stage of response-time analysis
- Do **not overcount** conflicting accesses



- Identification of all possible **memory accesses** that *may overlap* with the problem window
- Enforcement of the **FIFO invariant**

LABEL PLACEMENT

Problem: How to **optimize** the label placement?

- By leveraging the proposed response-time analysis, we formulated an **optimization problem** that aims at **placing the label** while **matching schedulability** constraints
- **Objective:** **minimize** the largest *normalized* response-time

$$\min \max \left\{ \frac{R_i}{D_i} \right\}$$

- The optimization problem has been addressed by
 - designing a **mixed-integer linear program (MILP)** formulation
 - developing a **genetic algorithm**

Achieved **+35%** w.r.t. the label placement provided in the **challenge** model

MILP FORMULATION

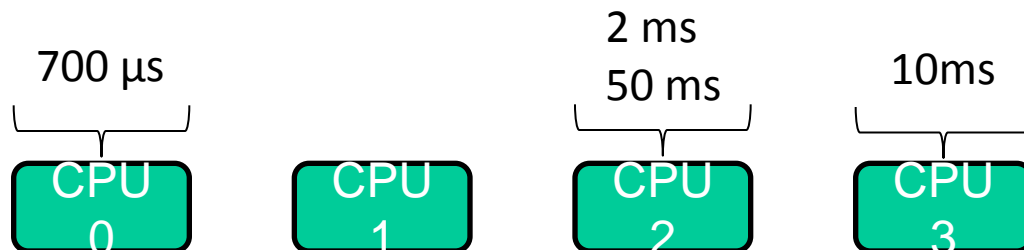
Guarantees **optimality** (w.r.t. the adopted analysis) and provides guaranteed **optimality gaps** of intermediate solutions

- Several **challenges** have been addressed to achieve a linear formulation of the problem
- Two key slight **approximations**:
 - Approximation of *high-priority Interference* (Park and Park, 2014), with **very low** empirical performance loss (<1%)
 - Resolution of *circular dependencies* introduced by the response-times in the memory contention terms (deadlines used as safe bounds on the response-times)

Communication	Objectives	Extensions
<ul style="list-style-type: none"> • Explicit • LET • Implicit 	<ul style="list-style-type: none"> • Minimize response-times • Minimize jitters • Maximize slacks • Minimize end-to-end latencies • ... 	<ul style="list-style-type: none"> • Deadlines on effect chains • Constraints on jitters • Constraints on response-times of runnables • ...

SETUP AND ASSUMPTIONS

- Use of **mean** execution times as **WCETs** (the system is definitively in **overload** when maximum execution times are used)
- Cost of **memory conflict**:
 - **1 cycle**, when *locally* generated (corresponding core)
 - **9 cycles**, when generated by a *remote* processor
- **Logical Execution Time**
 - Implemented **only** for the labels (and hence the corresponding runnables) involved in the **effect chains**
 - In total, **10 runnables** and just **7 labels**
 - Runnables of the same task managed by the same LET task, for a total of **5 LET tasks**



MEMORY OPTIMIZATION

MILP Formulation

- **IBM CPLEX** on 8-core Intel Xeon E5 @ 2.5Ghz
- The solver is able to **immediately** find a **feasible solution**
- **Optimal** solutions can be found in a **reasonable** amount of time
- **Provably very good** solutions can be found in **a few minutes**
- Impact of the approximations resulted very **marginal** (~1%)

Communication	Optimal	Optimality gap < 1%
Explicit	1h and 20 minutes	< 2 minutes
LET	1h and 50 minutes	< 7 minutes

Genetic Algorithm

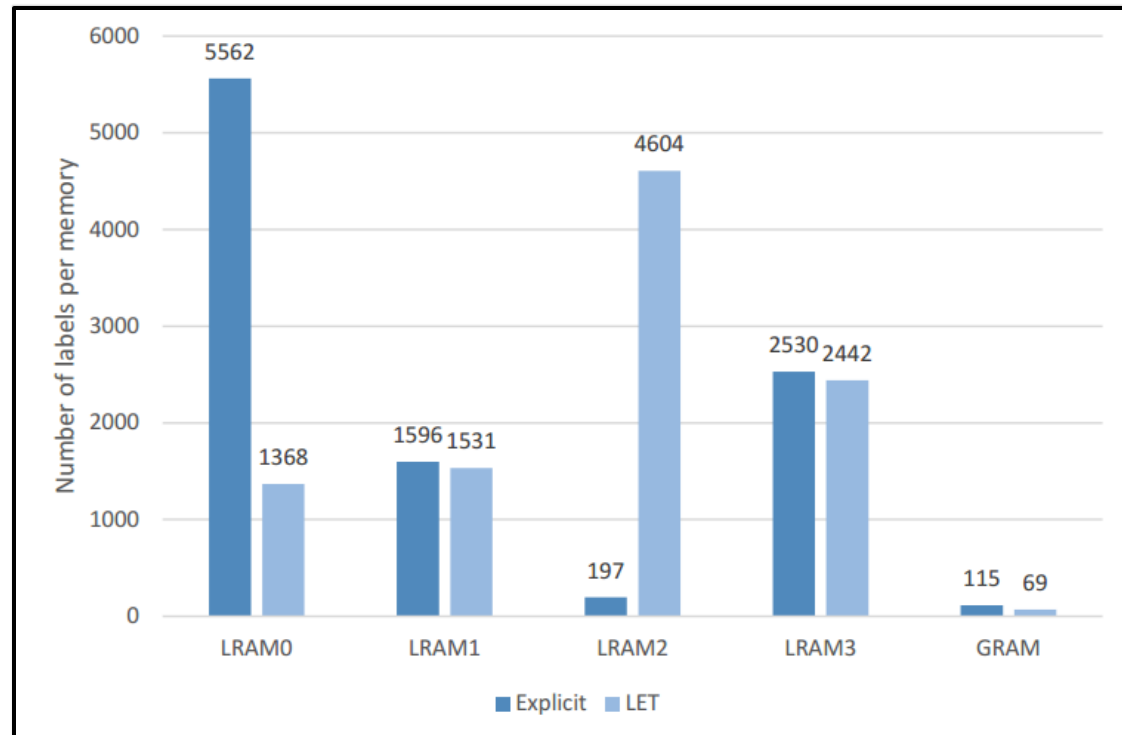
- Implemented in C++ and executed on Intel i7 @ 4Ghz
- Used as a **baseline for comparison**
- First feasible solution after ~2h and 45 mins
- Slight worse solution obtained with 20 runs of 40 hours each

OPTIMAL LABEL PLACEMENT WITH MILP

- Same quality of solution for both LET and explicit communication, but with completely **different placements**
- There are **several optimal solutions** that are **equivalent**
- **End-to-end latencies** have also been computed with the optimal label placement

Value of Objective Function

Explicit	0.849555
LET	0.849555



DISCUSSION

- Angle_sync task **overwhelms** all the timing constraints, leaving **little room** for observing interesting results
 - With **maximum** execution times, Angle_sync has an utilization of ~1.35 (without memory latencies)
 - Missing modeling of **speed-dependent behavior**?
 - The consideration of the **adaptive variable-rate (AVR)** task model would definitively improve the analysis precision
- **Local memories** are **quite large**: most labels can be fit into them without exploiting the global memory

The **key problem** is the placement of **runnables**

CONCLUSIONS AND FUTURE WORK

- **Contributions** (applicable in **general**):
 - Implementation of LET in AUTOSAR
 - New analysis for memory latencies
 - Optimization algorithms for label placement: MILP-based approach performs quite satisfactorily
- Several **limitations** in the challenge model:
 - Everything is **dominated** by the `Angle_sync`
 - Fixed placement of the **runnables**
 - Missing **deadlines** of the event chains

Need for a holistic **design methodology**:

- Joint placement of **runnables** and **labels**
- Selection of **communication mechanisms**
- Microcontroller configuration?, ...



The end

Thank you!