# A Framework for the Optimization of the WCET of Programs on Multi-Core Processors

Max John, Michael Jacobs

Department of Computer Science
Saarland University

October 8, 2014

# Setting



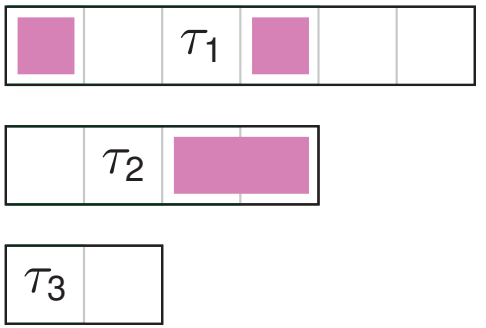
- Timing-critical applications
  - Embedded systems
  - Strict deadlines, e.g. in automotive applications
  - Need of tight WCET bounds
- Multi-core processor with shared bus
  - Exploit task parallelism
  - However: cores interfere
- Usage of a TDMA bus
  - Cores no longer interfere
- Use static task scheduling

## Challenge

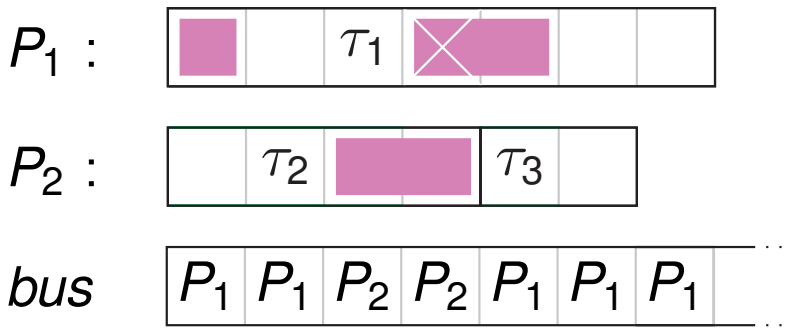
- Find static system schedule and bus schedule
- However: Optimal schedule hard to obtain
- Approximation framework needed

# System Model

- Given input
  - Number of processor cores
  - Set of tasks, each described by
    - ★ Length
    - ★ Bus accesses




- Variable parameters
  - System schedule
    - ★ Assigns tasks to cores
    - ★ Task order per core
  - Bus schedule
- Example schedule:




$\Rightarrow$ overall WCET: 7 time units

# Contributions

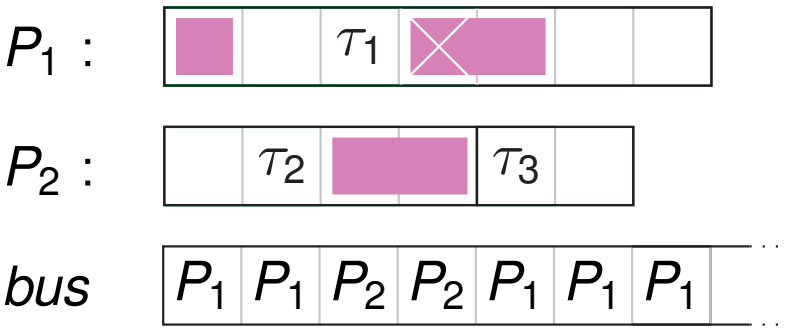


- A simple system model
  - One behavior per task
- An optimization framework
  - Goal: reduce the overall WCET
  - How: by constructing
    - ★ system schedule
    - ★ bus schedule
  - Modularity
    - ★ plug in different heuristics
  - Efficient implementation
    - ★ based on our system model
- Steps towards reality
  - Real-world programs have multiple behaviors
  - Soundly over-approximate them by a single one

# Future work



- Develop effective heuristics
  - Motivation: avoid access overlaps




- Experiments
  - Extract traces from real-world programs
  - Evaluate effectiveness of heuristics
  - Soundly combine several traces to one
    - ★ Determine degree of over-approximation

# Statically Resolving Computed Calls via DWARF Debug Information

Florian Haupenthal

October 8, 2014

# Motivation

## Allowing virtual functions in safety-critical embedded systems

```
class A {
  public:
    virtual void function() {
      // general implementation
    }
};

class B {
  public:
    void function() {
      // special implementation
    }
};
```

# Setting

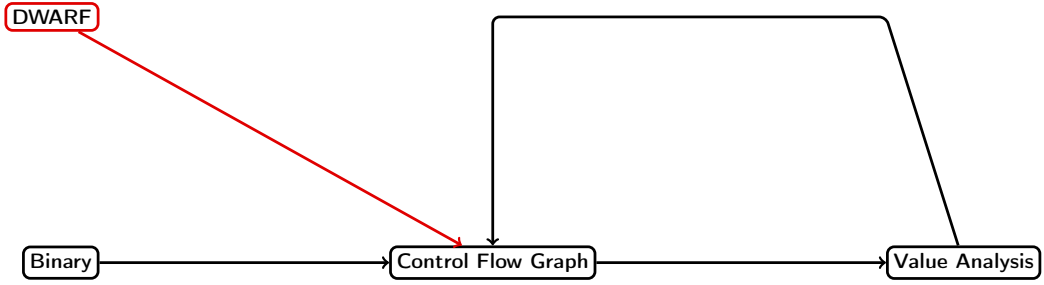
## Issues for the analysis

```
int main(int argc, char** argv) {
  A a;
  B b;

  A* aOrB;

  if (argc == 23) {
    aOrB = &a;
  } else {
    aOrB = &b;
  }

  aOrB->function();
}
```

```
...
lwz     r9, +0(r9)
lwz     r0, +0(r9)
mtspr   ctr, r0
lwz     r3, +8(r31)
bctrl
...
```

# Basic idea

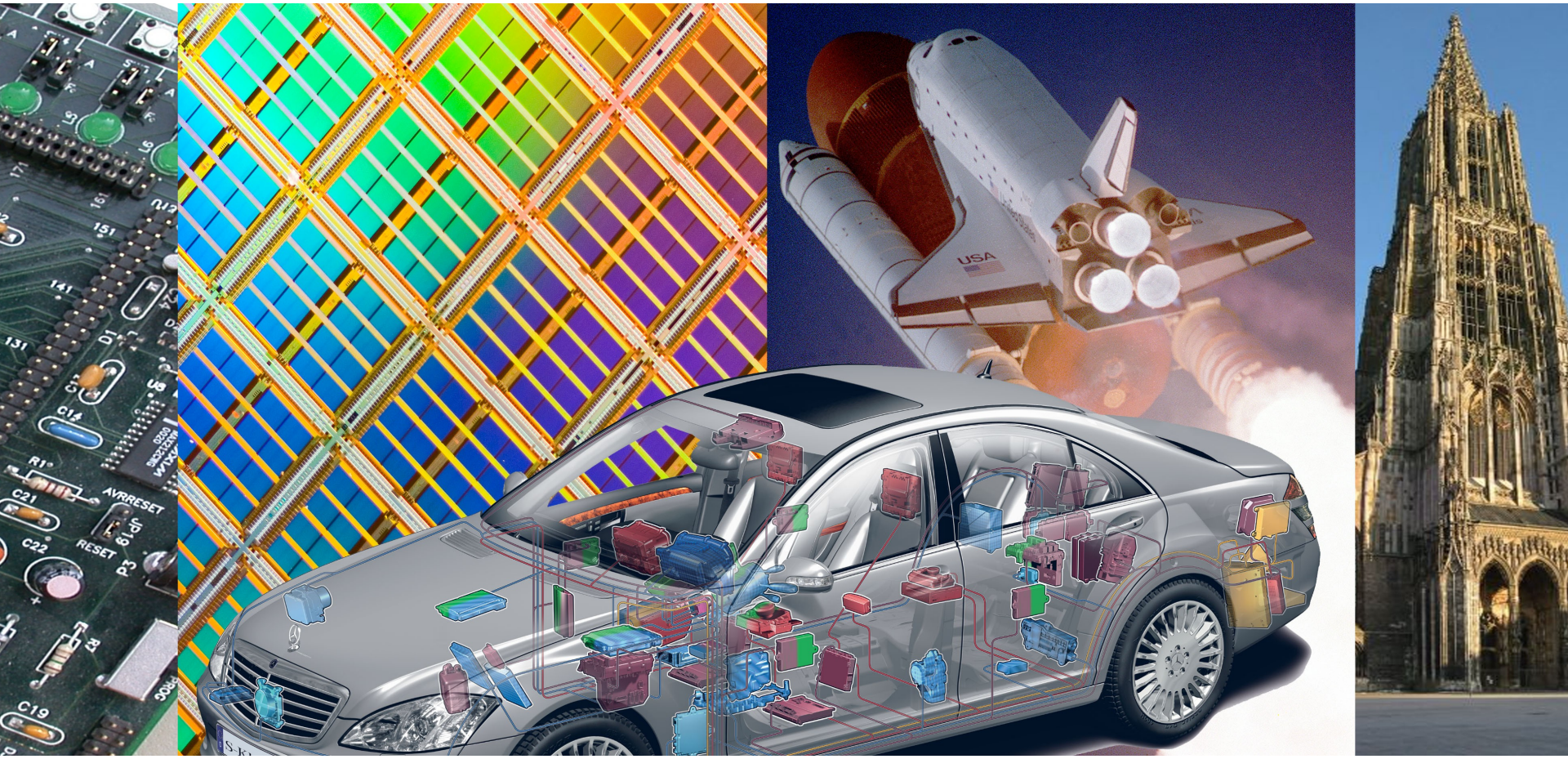
## Adding an additional information source to the analysis



# Suggestions for discussions at the poster

- A most likely not too expensive approach
- Evaluation still incomplete
- Works platform and compiler independent

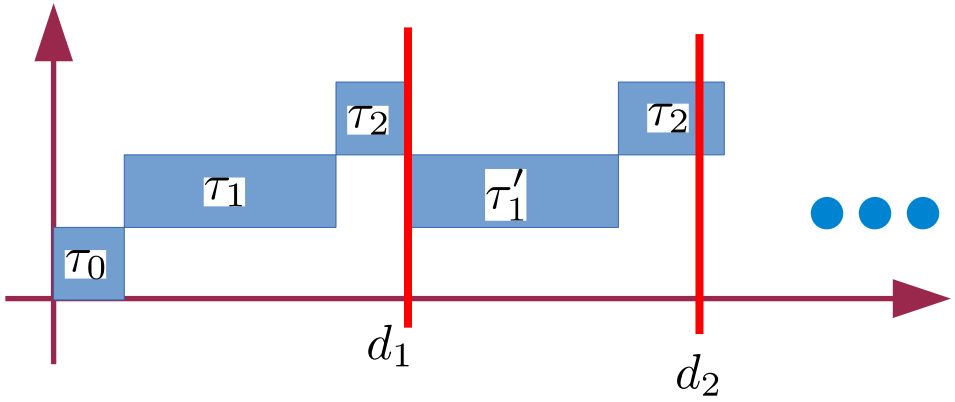
ulm university universität ulm




Arno Luppold | Heiko Falk
2014/10/08

# Schedulability-Oriented WCET-Optimization of Hard Real-Time Multitasking Systems

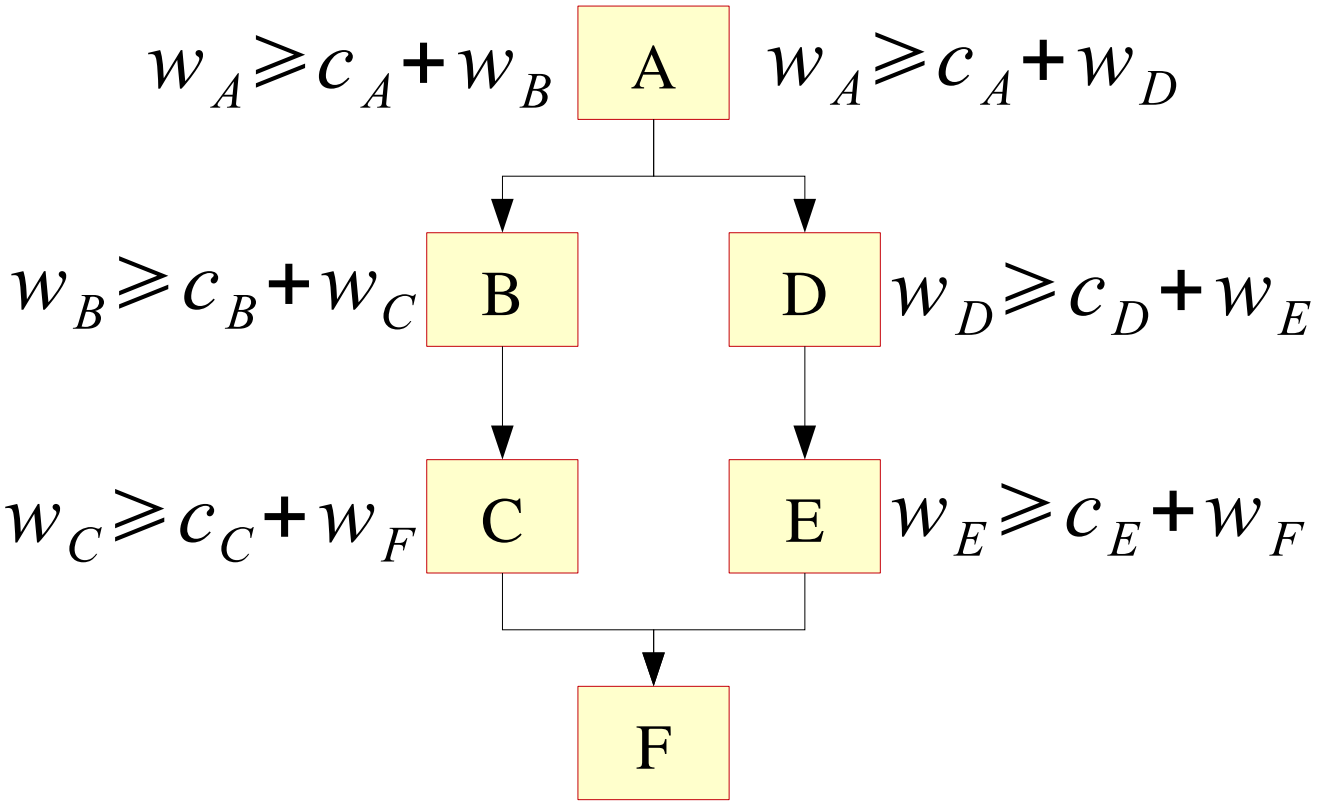
# Establish schedulability of a not schedulable system




- Increase deadline d
- Reduce WCRT r by decreasing WCET c
  - Remove functionality
  - Increase CPU capabilities
  - Compiler optimizations

# Integer-Linear Program (ILP) Based Singletasking Optimizations

$$min(w_A)$$




$$w_F \geqslant c_F$$

*V. Suhendra et al., "WCET Centric Data Allocation to Scratchpad Memory" in RTSS, 2005, pp. 223–232.*

# Extension for Periodic Multitasking Systems

Fixed Priorities:

$$r_i = c_i + \sum_{j=0}^{i-1} \left\lceil \frac{r_i}{T_j} \right\rceil \cdot c_j$$

Dynamic Priorities:

$$u = \sum_i \frac{c_i}{T_i}$$

*C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Journal of the ACM (JACM), 20(1):46–61, 1973.*

- Integration of the classic approaches on WCRT analysis into the singletasking ILP formulation to allow for scheduling-oriented optimizations

# Conclusion and Future Work

- Response-Time Analysis must be considered to effectively optimize hard real-time multitasking systems.
- Existing ILP based optimizations can be seamlessly used for multitasking systems using our approach.
- We demonstrated the approach using an ILP based instruction scratch-pad optimization.
- In the future we want to extend our approach to event-triggered systems.

# Accounting for Cache Related Pre-emption Delays in Hierarchal Scheduling with Local EDF Scheduler

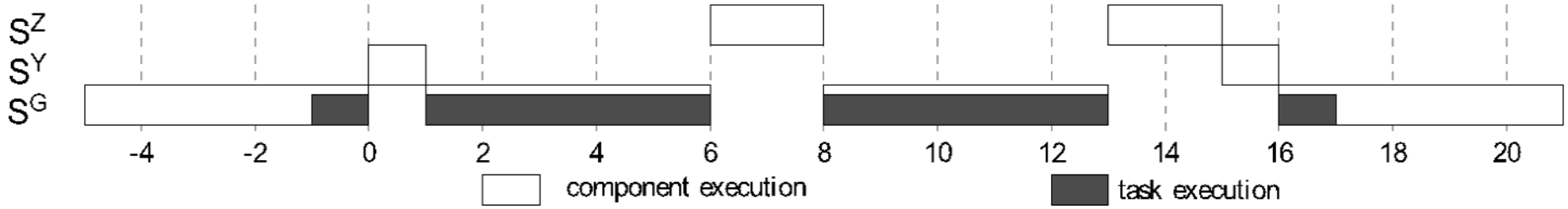
**Will Lunniss**[1] Sebastian Altmeyer[2] Robert I. Davis[1]
[1]Department of Computer Science, University of York, UK
{wl510, rob.davis}@york.ac.uk
[2]Computer Systems Architecture Group, University of Amsterdam, Netherlands
altmeyer@uva.nl

# Introduction

- Hierarchal Scheduling
  - Run multiple components on a single processor
  - Components should be isolated and not interfere with each other
  - Components are scheduled using a global scheduler
    - Assume non-pre-emptive
  - Tasks within a component are scheduled using a local scheduler
    - Assume pre-emptive EDF
- Cache Related Pre-emption Delays (CRPD)
  - Caused by the need to re-load blocks into cache that have been evicted by a pre-empting task
  - Tasks in other components could evict cache blocks, causing ‘inter-component’ CRPD

# Accounting for CRPD in Hierarchal Scheduling

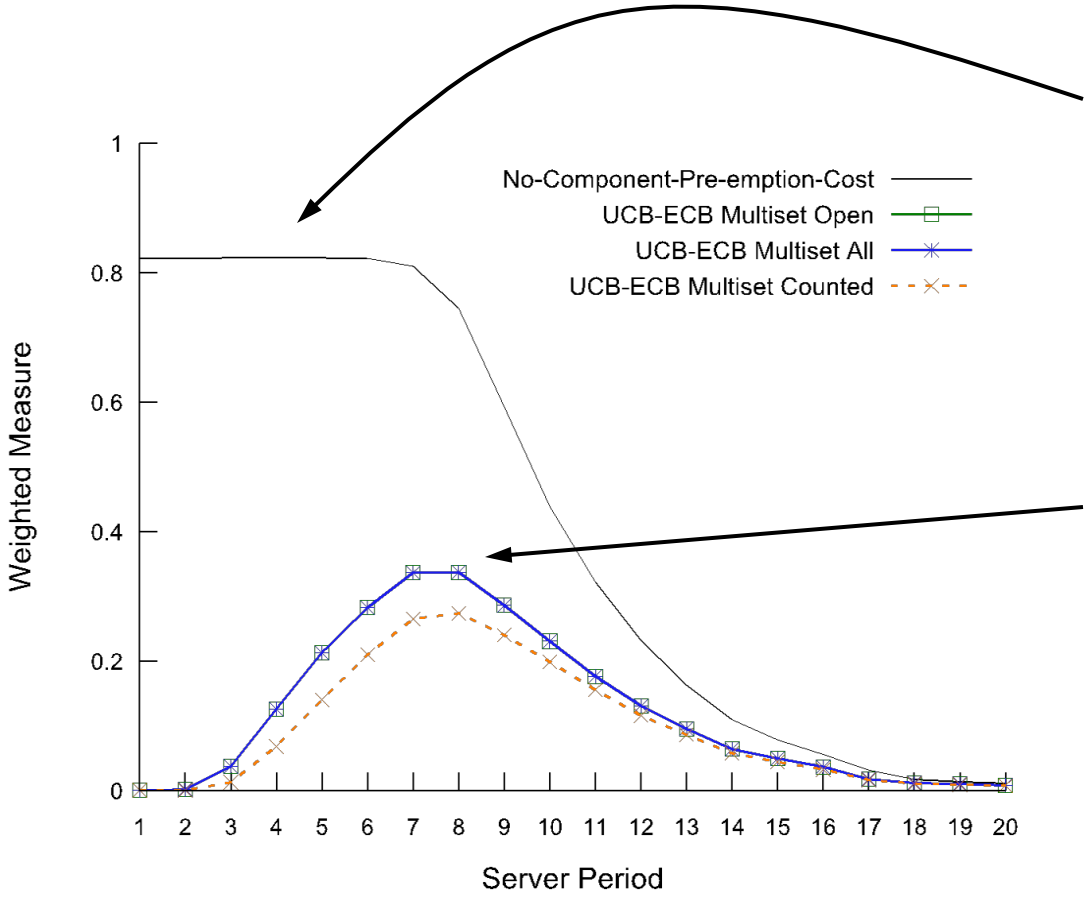
- CRPD due to tasks in the same component
  - Lunniss *et al.* [1] *Combined Multiset* approach
- Shared access to the processor
  - Shin and Lee [2] *Supply bound function*
- CRPD due to tasks in other components
  - Bound the number of times a component can be both *suspended* and *resumed* in an interval of length *t*




  - Calculate the set of blocks that if evicted by the tasks in the other components may need to be reloaded

[1] Lunniss, W., Altmeyer, S., Maiza, C., and Davis, R. I. Integrating Cache Related Pre-emption Delay Analysis into EDF Scheduling. In *Proceedings 19th IEEE Conference on Real-Time and Embedded Technology and Applications (RTAS)* (2013), 75-84.
[2] Shin, I. and Lee, I. Periodic Resource Model for Compositional Real- Time Guarantees. In *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS)* (2003), 2-13.

# Results




Small server periods maximise schedulability when component CRPD is not considered

In fact it is important to balance the server period

# Conclusions/Future Work

- New analysis for bounding inter-component CRPD with a local EDF scheduler
- Based on approaches for bounding inter-component CRPD with a local FP scheduler
  - "Accounting for Cache Related Pre-emption Delays in Hierarchical Scheduling"
  - Presentation tomorrow during Session 6 (13:50)
- Showed that inter-component CRPD must be carefully considered when selecting the server period
- Analysis uses the tasks' deadlines to bound inter-component CRPD which can be pessimistic
  - Aim to investigate other implementations to improve precision

# Alignment of Memory Transfers of a Time-Predictable Stack Cache

**Sahar Abbaspour**
Embedded Systems Engineering Sect.
Technical University of Denmark

**Florian Brandner**
Unité d'Informatique et d'Ing. des Systèmes
ENSTA-ParisTech



This work is partially supported by the EC project T-CREST.

# Introduction

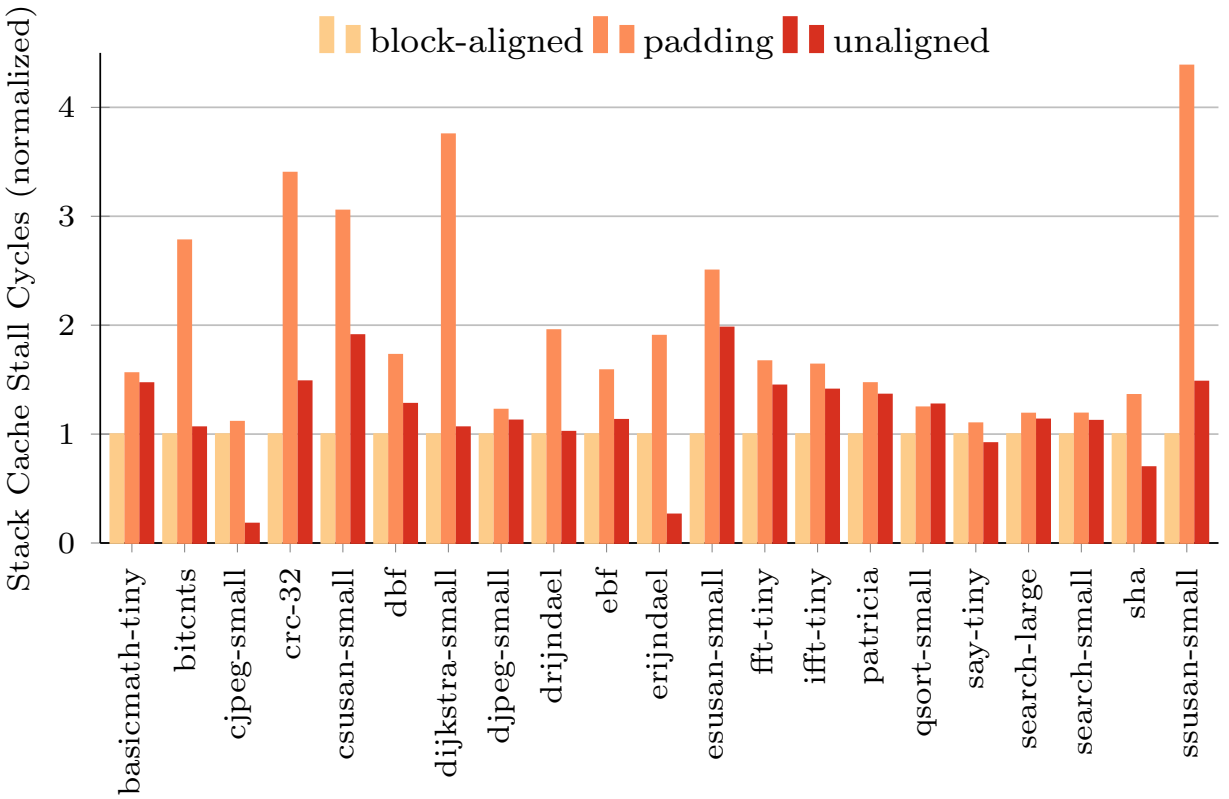
- Stack Cache
  - Specialized cache dedicated to stack data
  - Window following the logical stack of function calls
    - Reserve: Stack frames are allocated upon entering a function: potential spilling
    - Free: Stack frames are freed immediately before returning from a function
    - Ensure: Compiler ensures a valid stack cache state: potential filling
- Spilling/filling causes unaligned memory transfers
  - Increases analysis complexity
  - Cuases redundant transfers
    $\Longrightarrow$ **Transfers should be aligned to memory's burst size**

# Block-Aligned Stack Cache

- Hardware extension to align memory transfers
  - Stack cache organized in burst-sized blocks
    - Reserve one block as **alignment buffer**
  - Spilling/filling of whole blocks only
    - Alignment buffer rules over/underflows out
    - Improved utilization of bandwidth
    - Very low hardware overhead
  - Simple WCET analysis (alignment is guaranteed)
- Experiments
  - Compare against padding and unaligned transfers
  - Impact on runtime/analysis overhead

# Experimental Results

Mibench benchmarks compiled using LLVM compiler, running on the Patmos processor




Number of stall cycles normalized to our block-aligned stack cache extension

# Conclusion

- Padding is a simple solution to the alignment problem
  - Wastes space in stack cache
  - Increases spilling and filling
  - . . . up to a factor of 4
- Block-aligned stack cache
  - Reasonable trade-off with moderate hardware overhead
  - Average performance comparable to unaligned transfers
  - Simple analysis

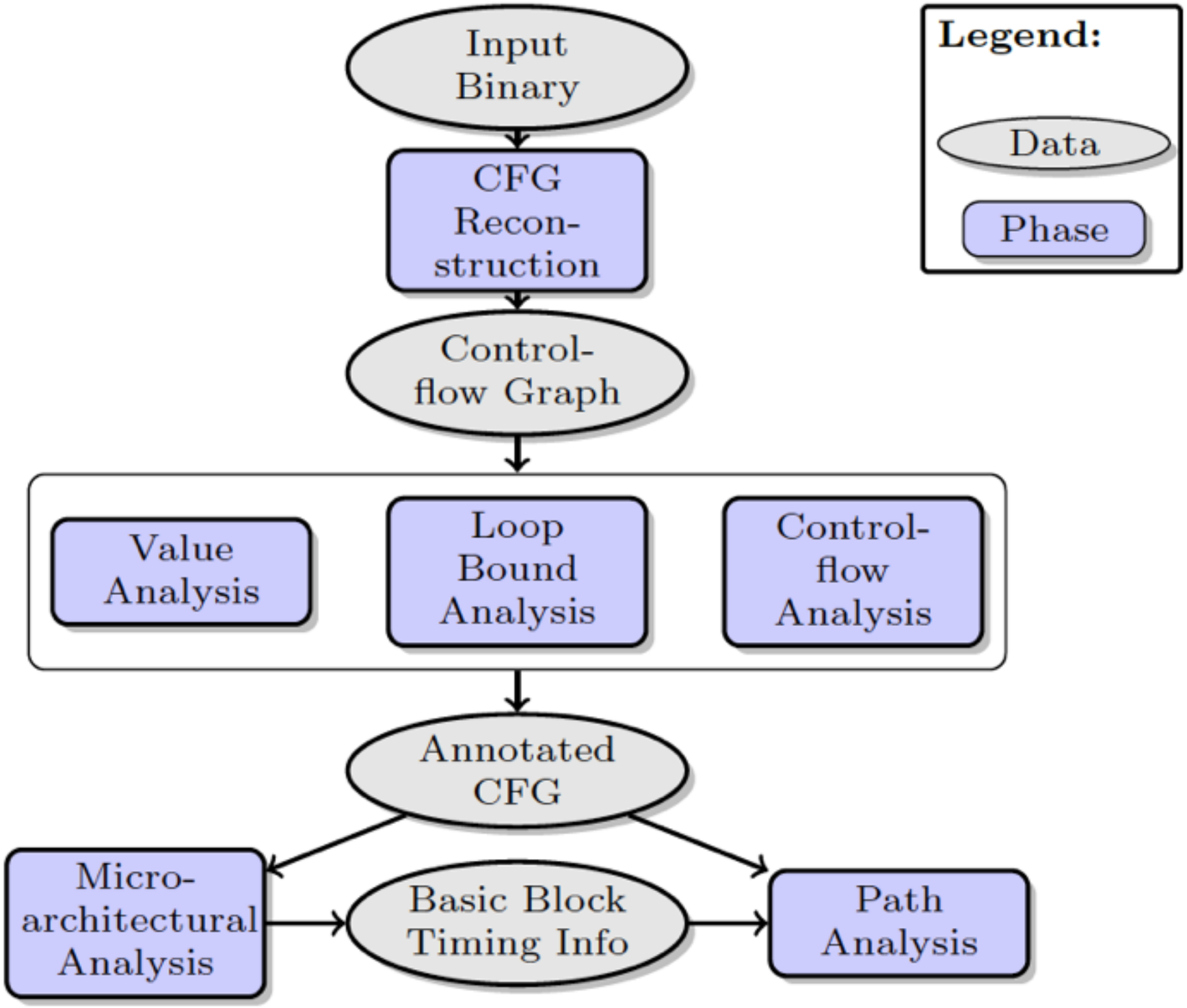
# The WCET Analysis using Counters
## *- A Preliminary Assessment -*

Remy Boutonnet, **Mihail Asavoae**

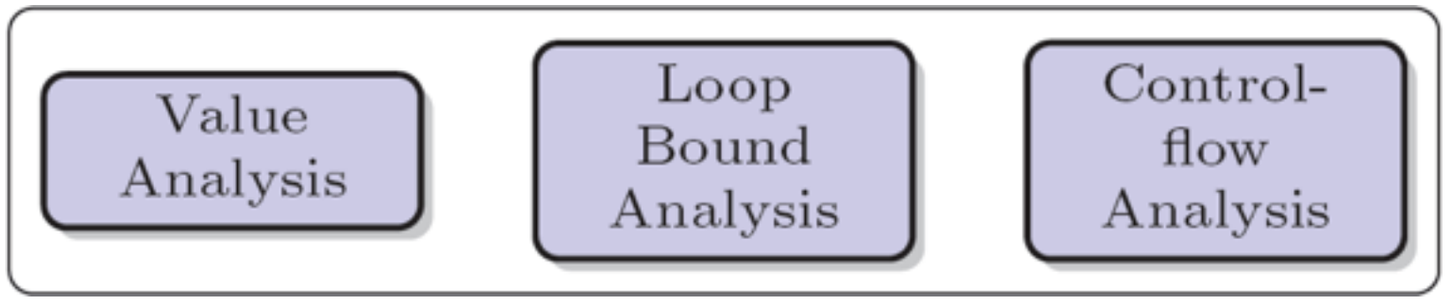
VERIMAG / UJF

**JRWRTC 2014 — 08 OCT 2014**

# Typical Workflow for the WCET Analysis

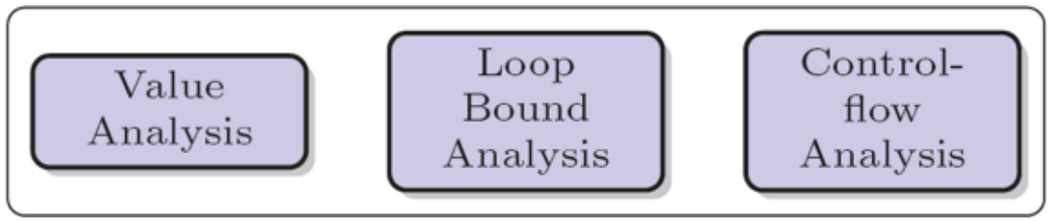


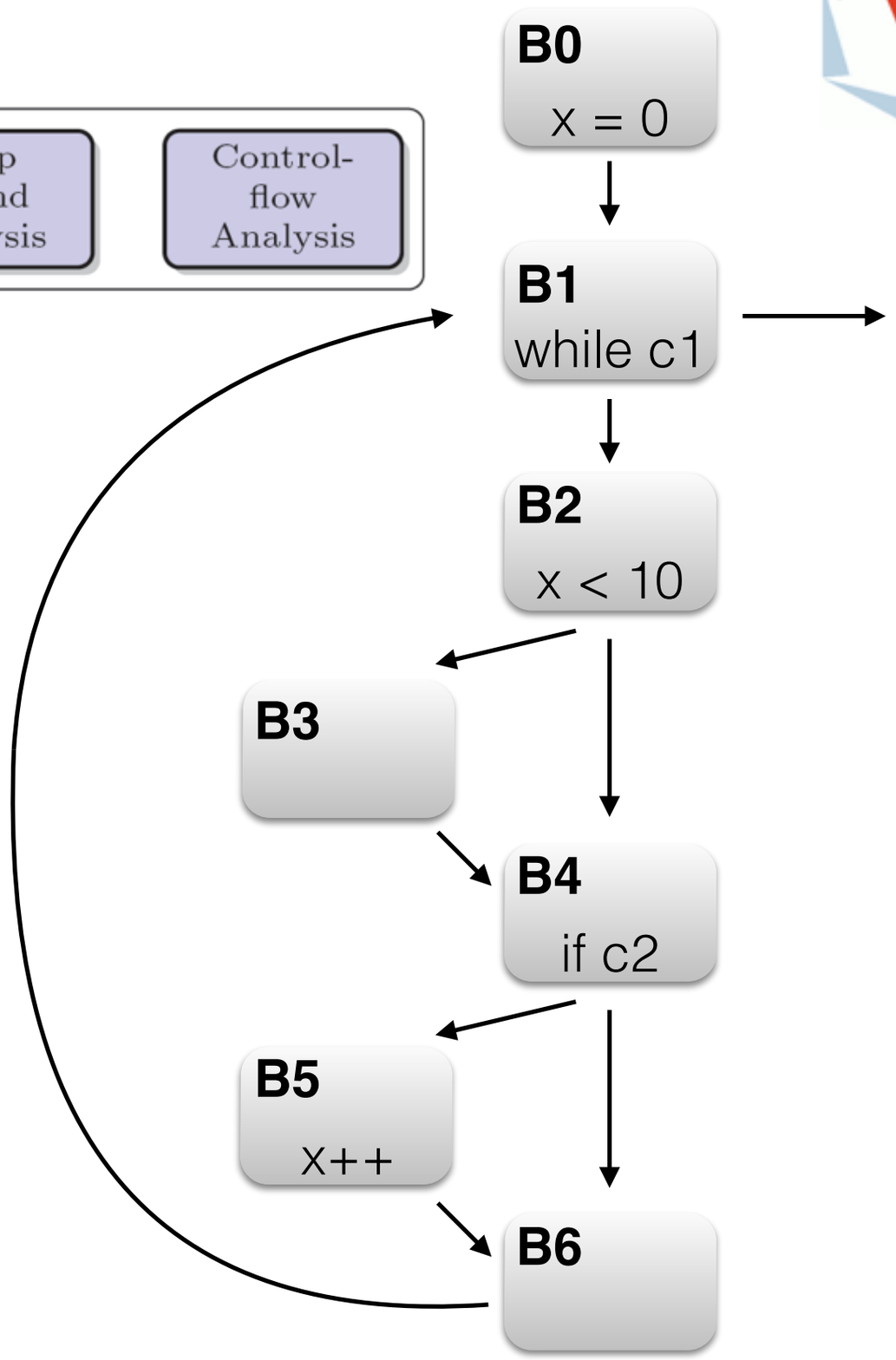
# Typical Workflow for the WCET Analysis



# Infeasible paths (I)

```
x = 0;
while (c1) {

    if (x < 10) {
        ...                    A
    }

    if (c2) {
        ...
        x++;                   B
    }
}
```

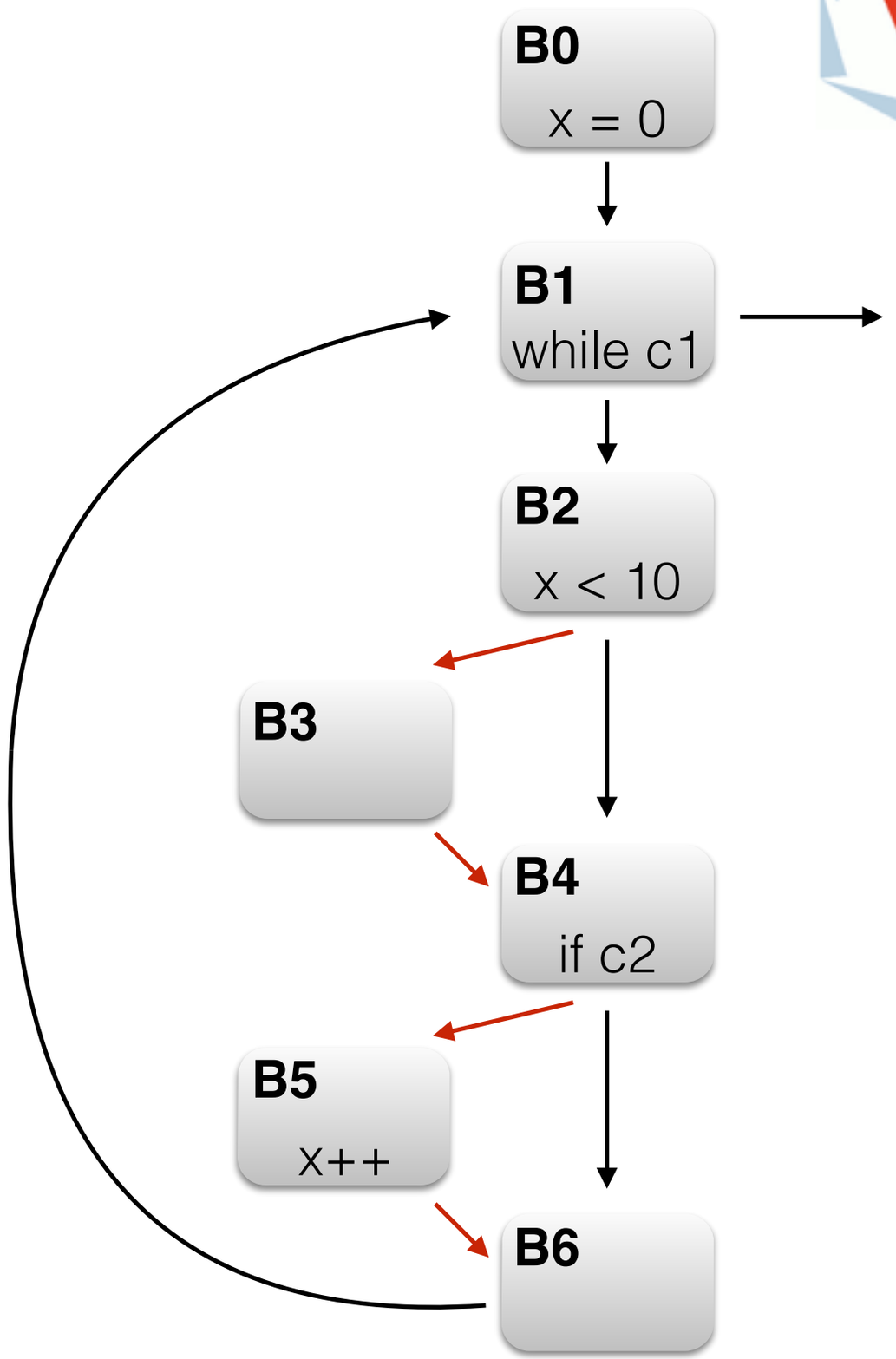



**Property : At most 10 iterations of the loop to execute A & B**

# Infeasible paths (II)

```
x = 0; alpha,beta,gamma=0;
while (c1) {
      alpha ++;
      if (x < 10) {
            ...
            beta ++;          A
      }

      if (c2) {
            ...
            x++;              B
            gamma++
      }
}
```




**Property : At most 10 iterations of the loop to execute A & B**

# Invariant Generation

```
x = 0; alpha,beta,gamma=0;
while (c1) {
    alpha ++;
    if (x < 10) {
        ...
        beta ++;          A
    }
    if (c2) {
        ...
        x++;              B
        gamma++
    }
}
```

gamma >= 0
**10 - beta - gamma+ alpha >= 0**
alpha - gamma >= 0
-beta + alpha >= 0

# Question (I)

```
x = 0; alpha,beta,gamma=0;
while (c1) {
    alpha ++;
    if (x < 10) {
        ...
        beta ++;          A
    }
    if (c2) {
        ...
        x++;              B
        gamma++
    }
}
```

gamma >= 0
**10 - beta - gamma+ alpha >= 0**
alpha - gamma >= 0
-beta + alpha >= 0

how to get them?

## Question (II)

```
x = 0; alpha,beta,gamma=0;
while (c1) {
    alpha ++;
    if (x < 10) {
        ...
        beta ++;          A
    }

    if (c2) {
        ...
        x++;              B
        gamma++
    }
}
```

gamma >= 0
**10 - beta - gamma+ alpha >= 0**
alpha - gamma >= 0
-beta + alpha >= 0

how to get them?

how many counters…?

## Question (…)

```
x = 0; alpha,beta,gamma=0;
while (c1) {
    alpha ++;
    if (x < 10) {
        …
        beta ++;          A
    }

    if (c2) {
        …
        x++;              B
        gamma++
    }
}
```

gamma >= 0
**10 - beta - gamma+ alpha >= 0**
alpha - gamma >= 0
-beta + alpha >= 0

how to get them?

how many counters…?

all invariants are useful?

is the method scalable?

what kind of app?

## Ask

```
x = 0; alpha,beta,gamma=0;
while (c1) {
    alpha ++;
    if (x < 10) {
        ...
        beta ++;          A
    }

    if (c2) {
        ...
        x++;              B
        gamma++
    }
}
```

gamma >= 0
**10 - beta - gamma+ alpha >= 0**
alpha - gamma >= 0
-beta + alpha >= 0

how to [illegible]et them?

how ma[illegible]unters…?

a[illegible] **POSTER** [illegible]ul?

is the[illegible]lable?

[illegible]hat kind of ap[illegible]?

# THANK YOU

# Adaptation of RUN to Mixed-Criticality Systems


Romain GRATIA — IRT SystemX

Thomas ROBERT — Institut Mines-Telecom

Laurent PAUTET — Institut Mines-Telecom

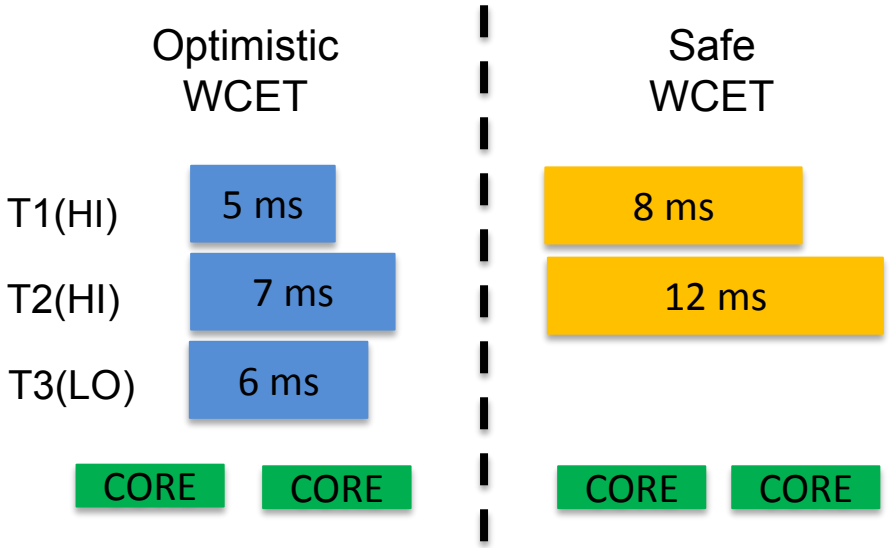
# Introduction and Motivation

**Sharing a multi-core platform between applications of different criticality levels**

**Compute Worst Case Execution Times (WCET) far greater than Average Execution Times in order to be safe.**
**Is it always relevant ?**

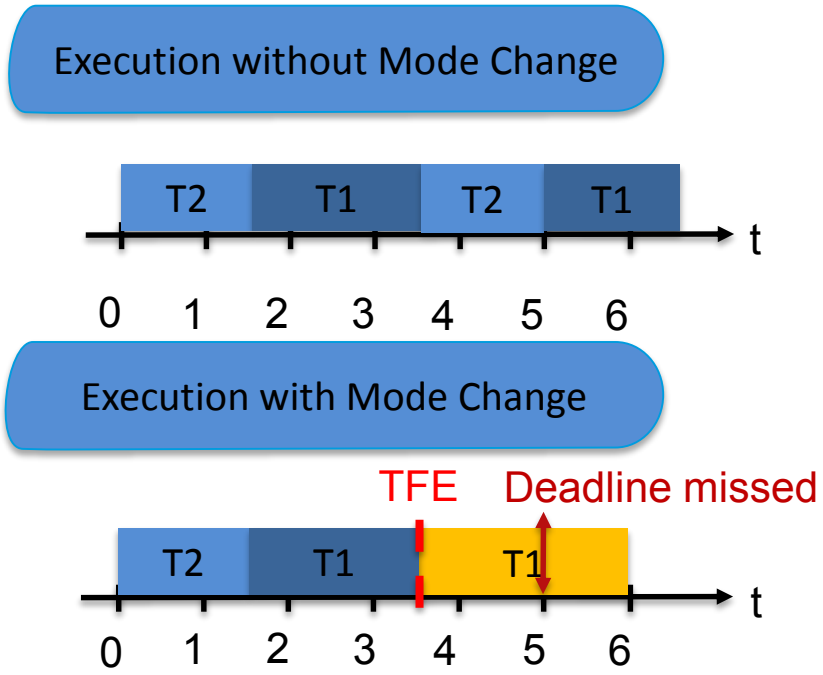
**The higher the criticality level, the safer the WCET must be**

**Scheduler has to properly handle the switching from Optimistic to Safe modes**

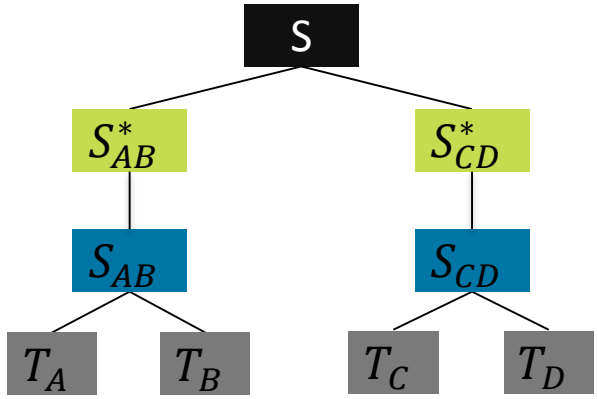


# Mode change & the RUN algorithm

- **Mode Change**
  - Optimistic & Safe modes called respectively LO-mode & HI-mode
  - Mode change triggered by a Timing Failure Event (TFE)

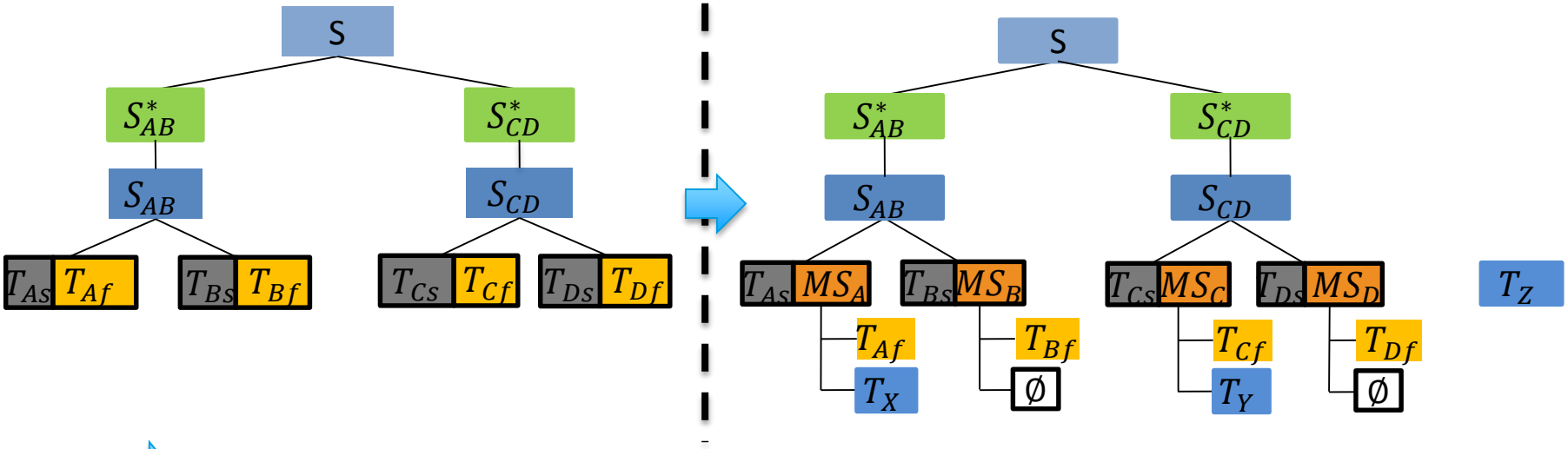



- **RUN**
  - Optimal global multi-core scheduling algorithm
  - Fewer preemptions and migrations than other optimal global scheduling algorithms
  - Two-steps algorithm
  - Based on a hierarchy of Primal $S$ and Dual servers $S^*$



# Adaptation of RUN to Mixed-Criticality systems

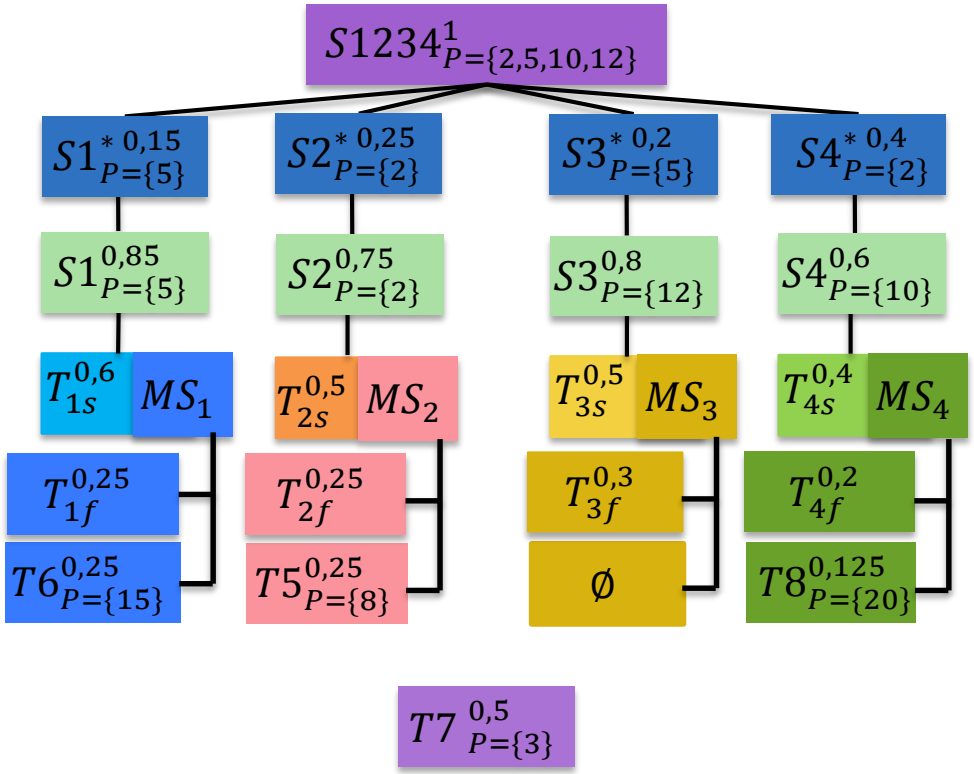
- Compute RUN schedule for HI-mode
- Split HI tasks in LO-mode part & remaining part up to HI-mode
- Define remaining parts as Modal Servers
- Allocate as many as possible LO tasks to Modal Servers
- Compute a RUN schedule for remaining LO tasks independently




**Objective: reduce the required number of processors for the scheduling of a task set compared to non-modified RUN**

# Example

| Task name | Period | Criti-cality level | Utiliza-tion (LO-mode) | Utiliza-tion (HI-mode) |
|---|---|---|---|---|
| T1 | 5 | Hi | 0,6 | 0,85 |
| T2 | 2 | Hi | 0,5 | 0,75 |
| T3 | 12 | Hi | 0,5 | 0,8 |
| T4 | 10 | Hi | 0,4 | 0,6 |
| T5 | 8 | Low | 0,25 | 0,25 |
| T6 | 15 | Low | 0,25 | 0,25 |
| T7 | 3 | Low | 0,5 | 0,5 |
| T8 | 20 | Low | 0,125 | 0,125 |




| Task set / Mode | Ceiling utilization |
|---|---|
| LO+HI / HI-mode | 5 |
| Our RUN Adaption | 4 |

# Study of Temporal Constraints for Data Management in Wireless Sensor Networks

Abderrahmen Belfkih, Bruno Sadeg, Claude Duvallet, Laurent Amanton

{Abderrahmen.Belfkih,Bruno.Sadeg,Claude.Duvallet,
Laurent.Amanton}@univ-lehavre.fr

Le Havre University

8th Junior Researcher Workshop on Real-Time Computing

# Introduction and motivation

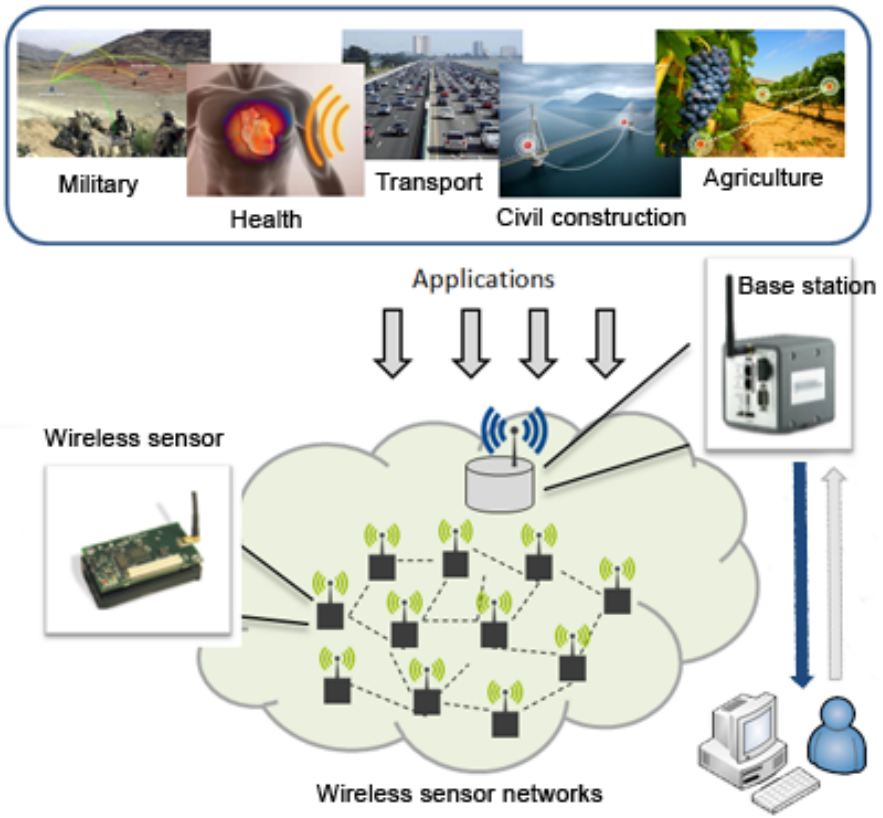



Figure: Wireless Sensor Networks

## Introduction

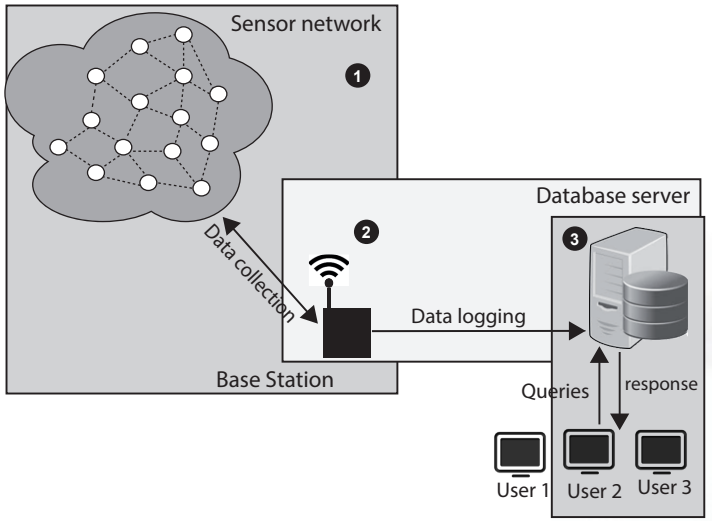
1. WSN are deployed without considering the data deadlines.
2. Many WSN applications require a strict deadline for data delivery.
3. Researchers are interested in data processing techniques to increase the network lifetime.

## Motivation

1. We study temporal constraints and data arrival times from sensors to users.
2. We test two technologies: abstract database and periodic data collection.
3. We identify the factors which enhance the respect of temporal constraints.

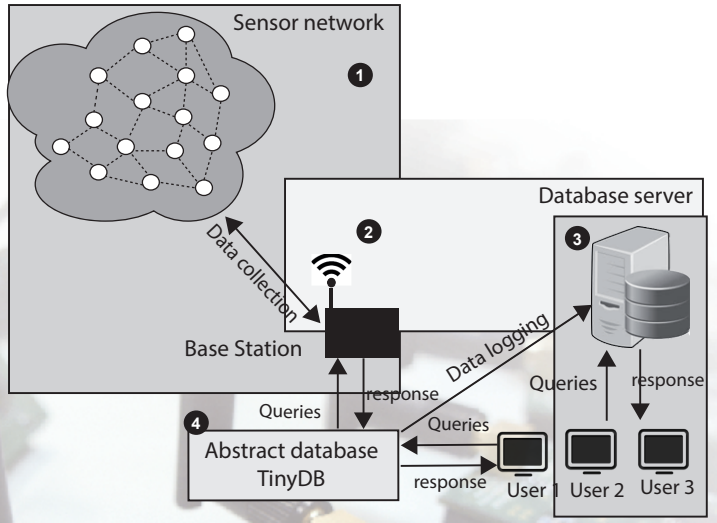
# Model and constraints

## Data collection with remote database




1. Sensor nodes send periodically data to the base station.
2. The base station inserts sensor data into the remote database.
3. Users can connect to the database to get information about WSN.

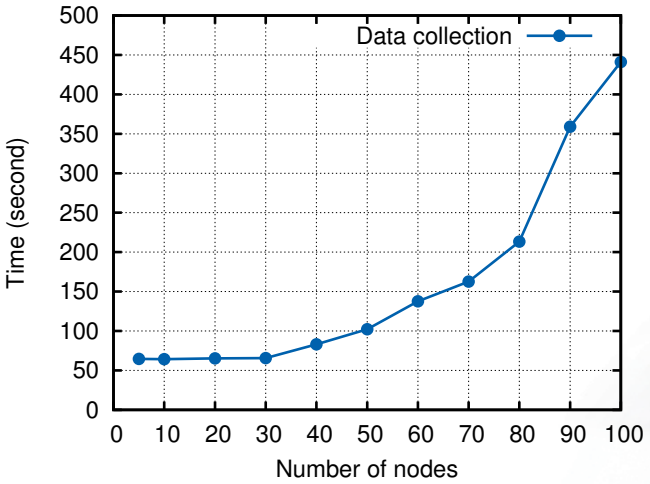
## Query processing with TinyDB



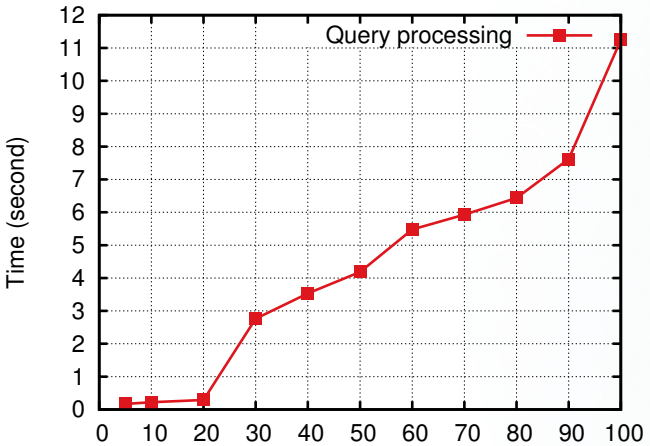

1. User sends SQL-like query to the base station via the the abstract database.
2. The base station broadcast these queries over the network.
3. The base station sends received data to the user via the abstract database.

# Experimental results

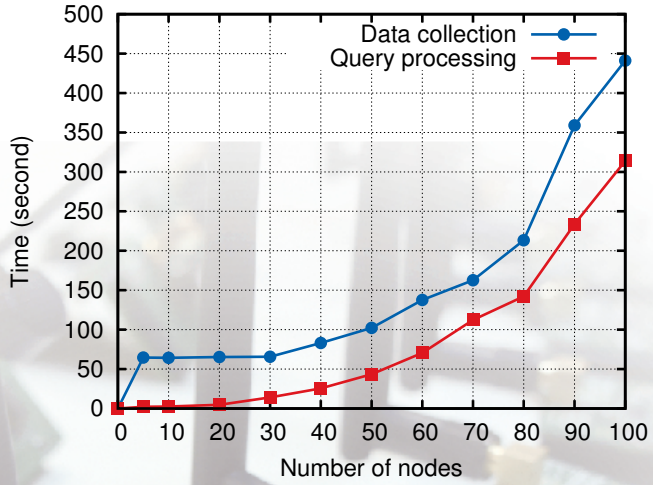
## Data collection Convergence time




## Query processing convergence time (TinyDB)




## Completed Cycle Time for data collection & Query processing (TinyDB)




## Impact of network topologies time (sec)

| Topologies | Data collection | Query processing |
|---|---|---|
| Star | 62.312 | 1.832 |
| Mesh | 56.002 | 1.362 |
| Grid | 54.121 | 2.163 |
| Tree | 53.515 | 2.143 |

## Impact of choosing the database time(sec)

| Query type | PostgreSQL | MySQL | SQLite |
|---|---|---|---|
| Insert queries | 9.397 | 48.626 | 72.788 |
| Select queries | 0.992 | 0.690 | 0.225 |

# Conclusion

- The convergence time has an impact on the process of data collection.
- The network topology and the routing protocol, together may play an important role on data collection time.
- The timing-response advantage of using a TinyDB approach compared to accessing the data stored in an external database.
- The abstract database approach has shown performances for data collected time and for network convergence time than the data collection approach.
- The network topology and the routing protocol with the right choice of approach can improves the temporal constraints in WSN.

FROM RESEARCH TO INDUSTRY

cea tech

le cnam

ens cachan

# An Approach for Verifying Concurrent C Programs

Amira METHNI (CEA/CNAM)
Matthieu LEMERRE (CEA) & Belgacem BEN HEDIA (CEA)
Serge HADDAD (ENS Cachan) & Kamel BARKAOUI (CNAM)

www.cea.fr

**October 8th, 2014**

leti & list

# Introduction

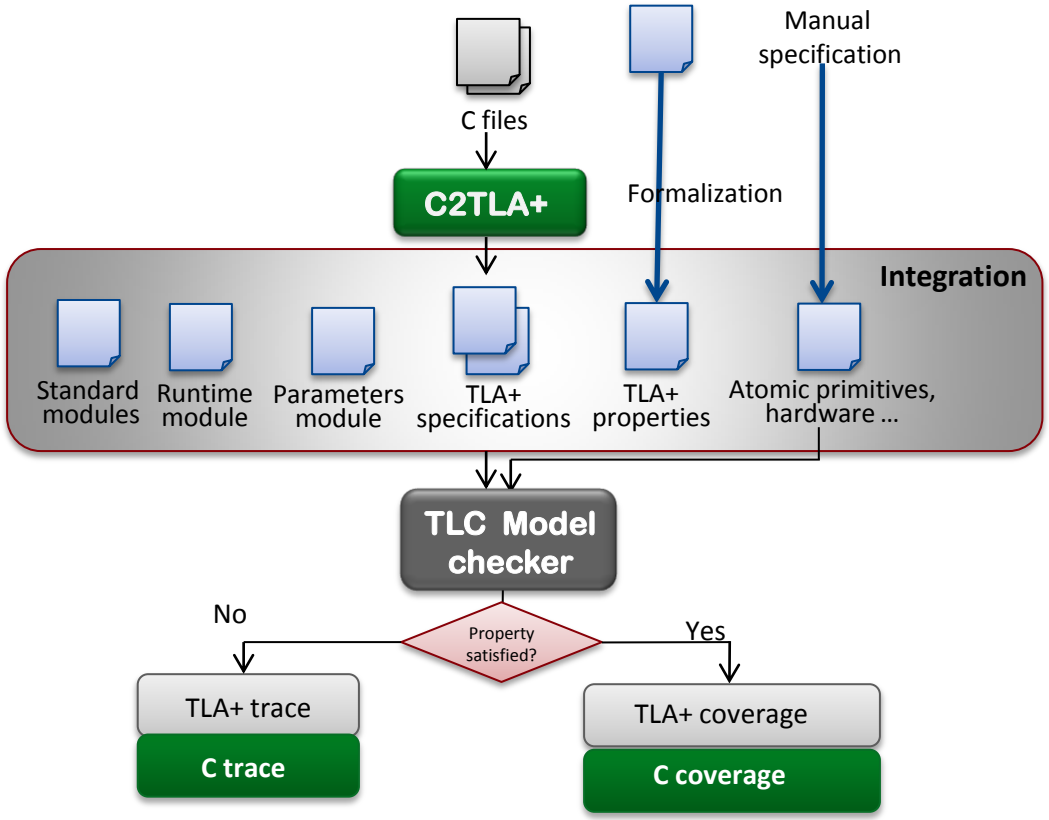
- Context and problemtaic
  - C is a low level language
  - Concurrency  is hard to verify
  - Verifying C code is challenging
  - Manual inspection is error-prone and costly
- Objectives
  - Method and tools adapted to this type of engineering
  - Approach for design assistance and formal verification

# Related work

| | | |
|---|---|---|
| SLAM [Ball, T & al], BLAST [Henzinger, T & al] | - CEGAR (ConterExample-Guided Abstraction Refinement)<br>- Model checking/proofs/static analysis | - Limited support for concurrent properties<br>- Only safety properties |
| Modex [Holzmann, G.J. & al] | - Model extraction<br>- Modeling language: Promela | - No support for pointers<br>- Well suited or specifying communication protocols |
| PlusCal [Lamport, L.] | - High level language<br>- TLA logic | - No support for pointers data structure and function calls |

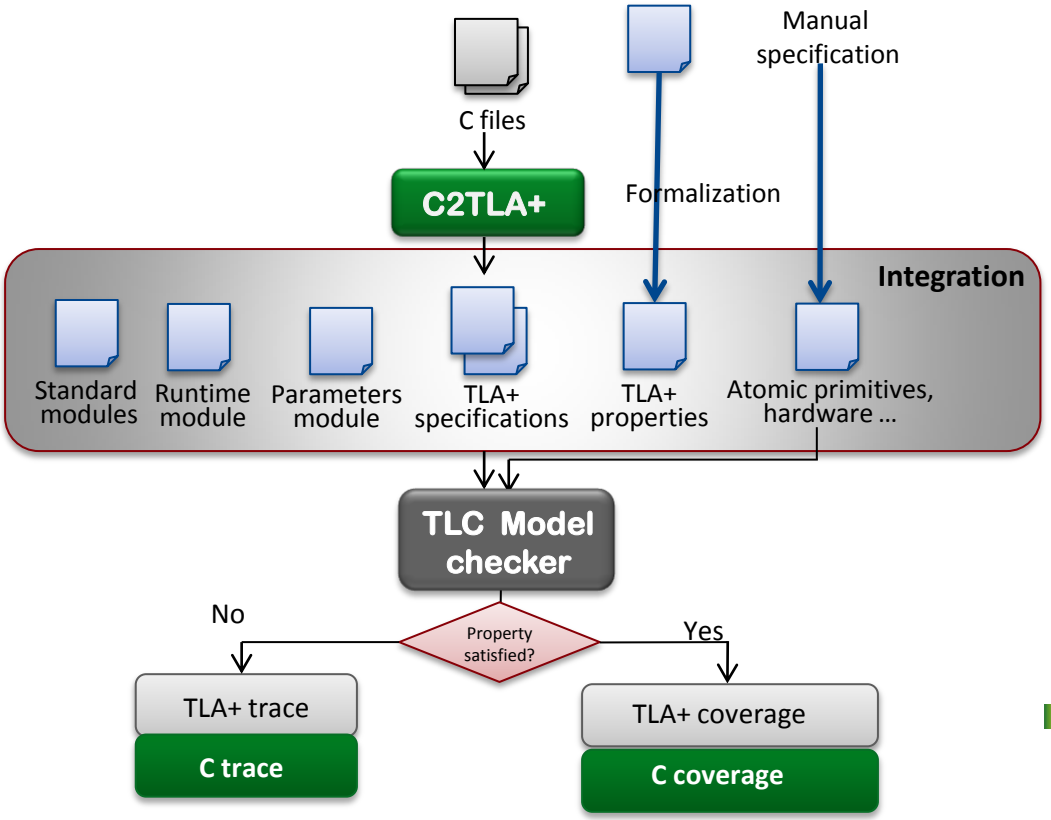
[Ball, T & al] : *The SLAM project: Debugging System Software via Static Analysis*. SIGPLAN Not, 2002
[Henzinger, T & al] : *Software Verification with BLAST*. Springer, 2003, 235-239
[Holzmann, G.J. & al] : *Automating Software Feature Verification.* Bell Labs Technical Journal, 2000, pp72-87
[Lamport, L.] : The PlusCal Algorithm Language. In ICTAC, 2009, pp36-60

# General approach




- **TLA+**
  - Its semantics is suited to express a programming language
  - Safety and liveness properties
  - Structural concepts: Refinement of specifications
  - Supporting tools : TLC model-checker , TLAPS prover.
- **Supported features**
  - Data types: int, struct, enum
  - Pointers, pointer arithmetic, array indexing
  - All kinds of control flow statement
  - Recursion
  - Concurrency

# General approach




- **Integration**
  - Manually specified modules
    - To provide concurrency primitives or hardware that can not be expressed in C
    - To define properties
  - Abstract modules
    - Relate states of the abstract specification with states of the concrete specification
    - Property preservation through refinement
    - Substitute a concrete C specification with an equivalent simpler one.
- **Using TLC to verify properties**
  - Safety (Invariants)
  - Liveness (program termination)
- **Getting the C trace and C coverage**

# Conclusion

## Conclusion

- Approach for specification and verification of C code
  - Automatic translation (C2TLA+) based on a set of rules.
  - Integrate generated modules with other manually specified specifications and abstract specifications.
  - Verifying a set of properties (safety and liveness).

## Future work

- To further study the refinement between two C programs
- To apply the approach on a concrete case study (PharOS*)
- To benefit from dependencies analysis of shared variables in order to generate an optimized TLA+ code.
- To use TLAPS and C2TLA+ in order to prove that a (translated) specification implements an abstract one or to prove properties on the specification.

* Lemerre, M. et al. *Method and Tools for Mixed-Criticality Real-Time Applications within PharOS.In* Proceedings of AMICS 2011

**CISTER** - Research Center in
Real-Time & Embedded Computing Systems

# Resource Sharing Under a Server-based Semi-partitioned Scheduling Approach

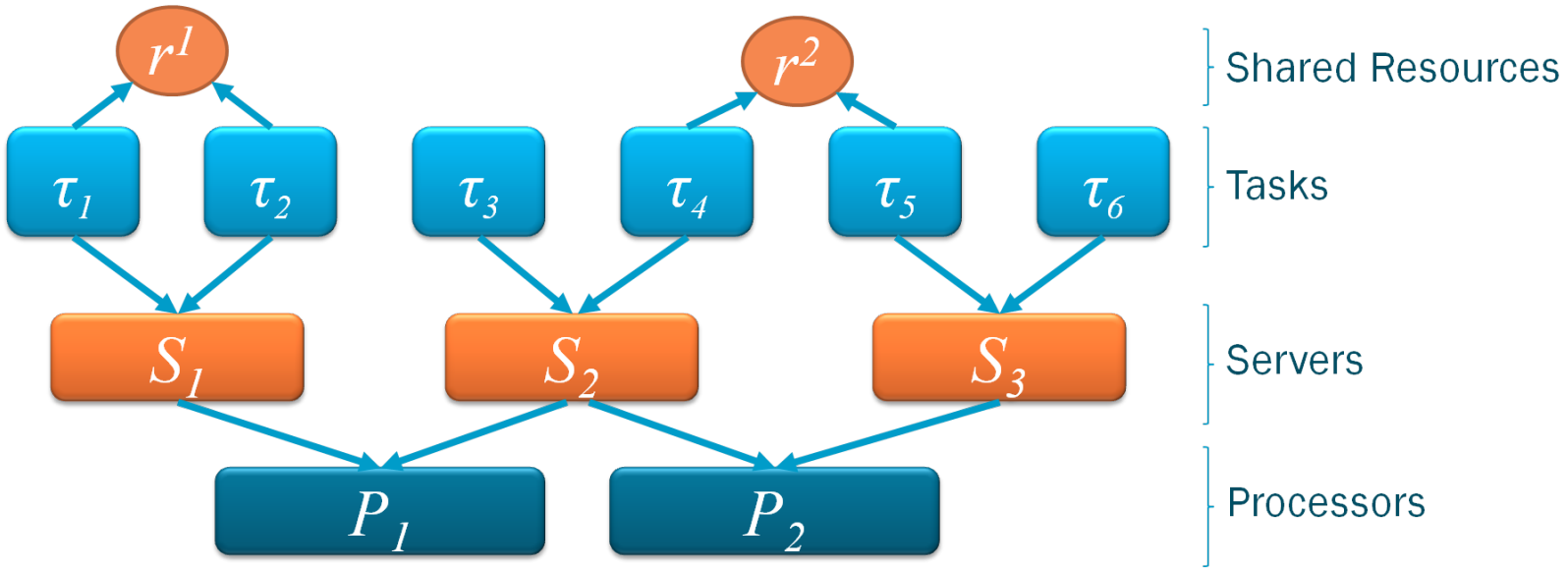
Alexandre Esper

Eduardo Tovar

# Context and Goal

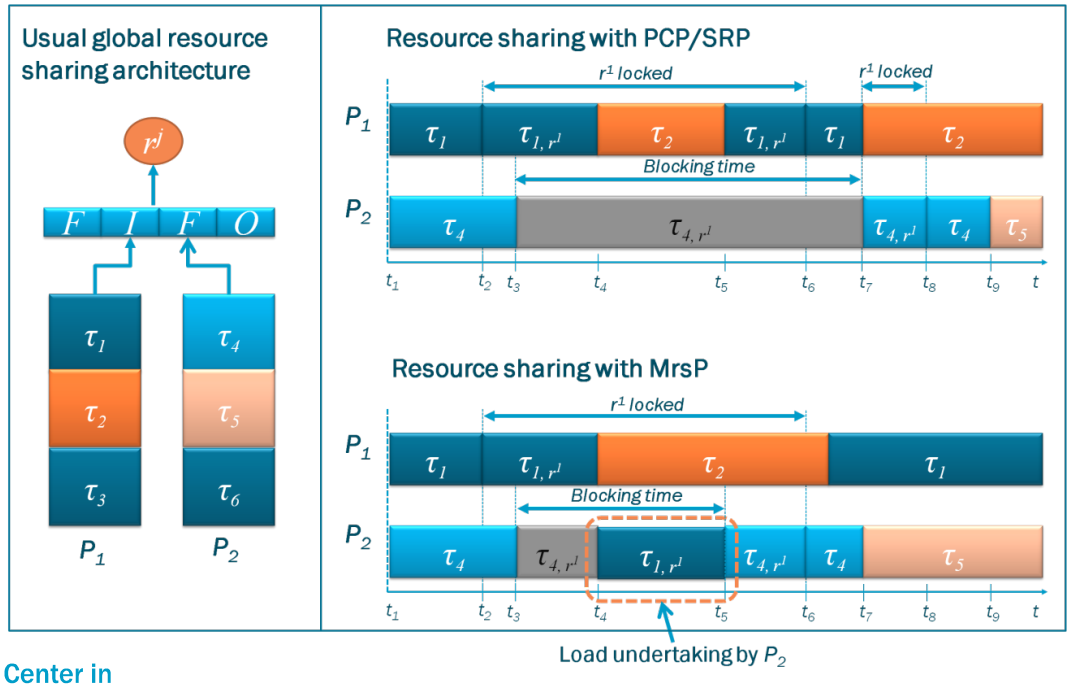
Goal:

- Adapt **MrsP** *resource sharing protocol* to work with servers through *bandwidth inheritance*
- Adapt **NPS-F** schedulability test to introduce adapted version of MrsP



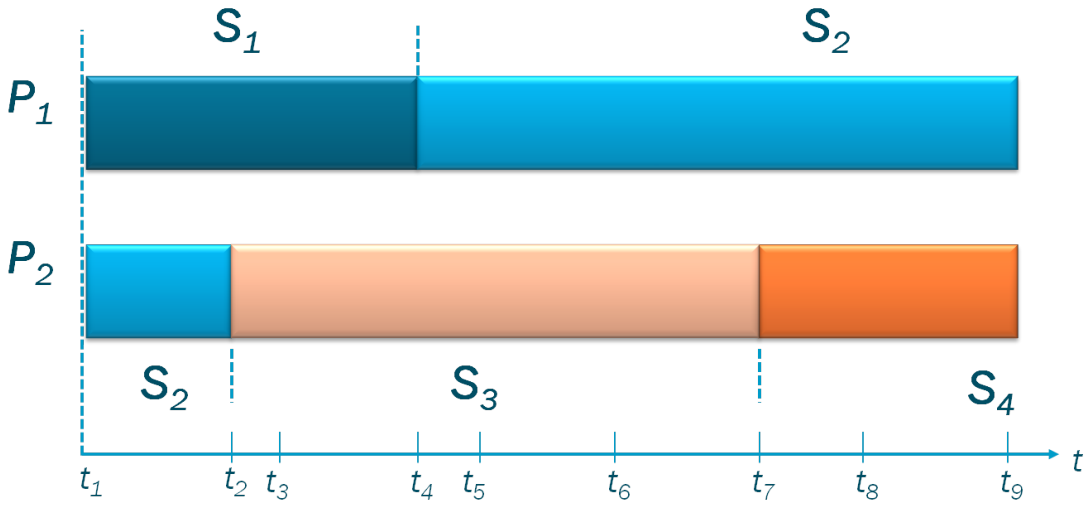
# MrsP

- Generalization of PCP/SRP Response Time Analysis to **multicore**
- Defined for **fully partitioned** systems where tasks are scheduled using **fixed priorities**
- Only one task per processor accessing a resource at any time
- Blocked tasks can undertake load of tasks holding the resource that has been preempted



# NPS-F

- Semi-partitioned scheduling algorithm
- Server-based approach
- Does not consider shared resources
- Servers serve one or more tasks **using EDF**



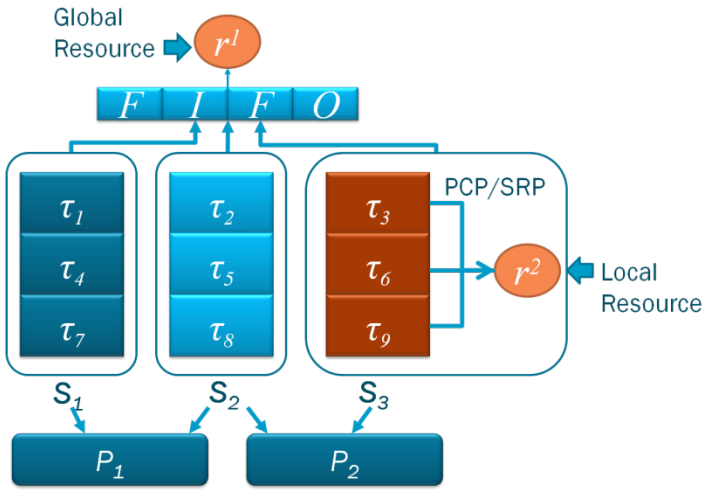
# Contribution

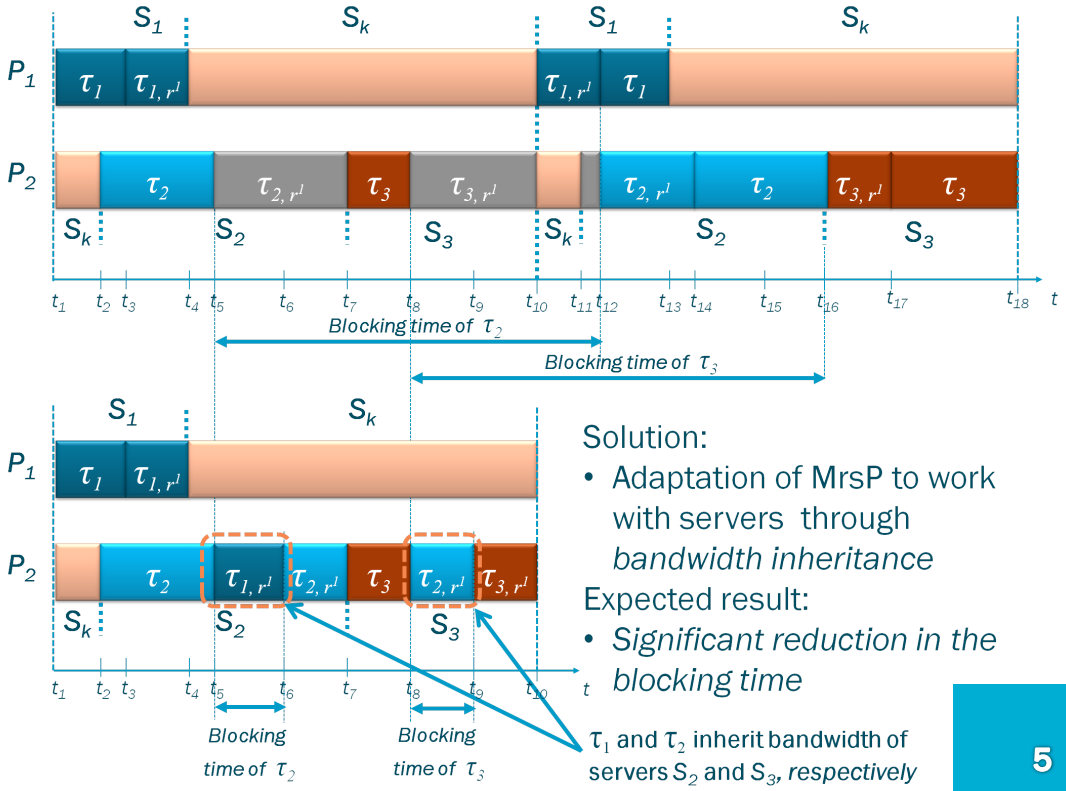
Goal:

- Account for **shared resources** in NPS-F by adapting MrsP

Challenges:

- MrsP is defined for fixed priority while NPS-F uses EDF
- MrsP is defined for fully partitioned while NPS-F uses servers




Solution:

- Adaptation of MrsP to work with servers through *bandwidth inheritance*

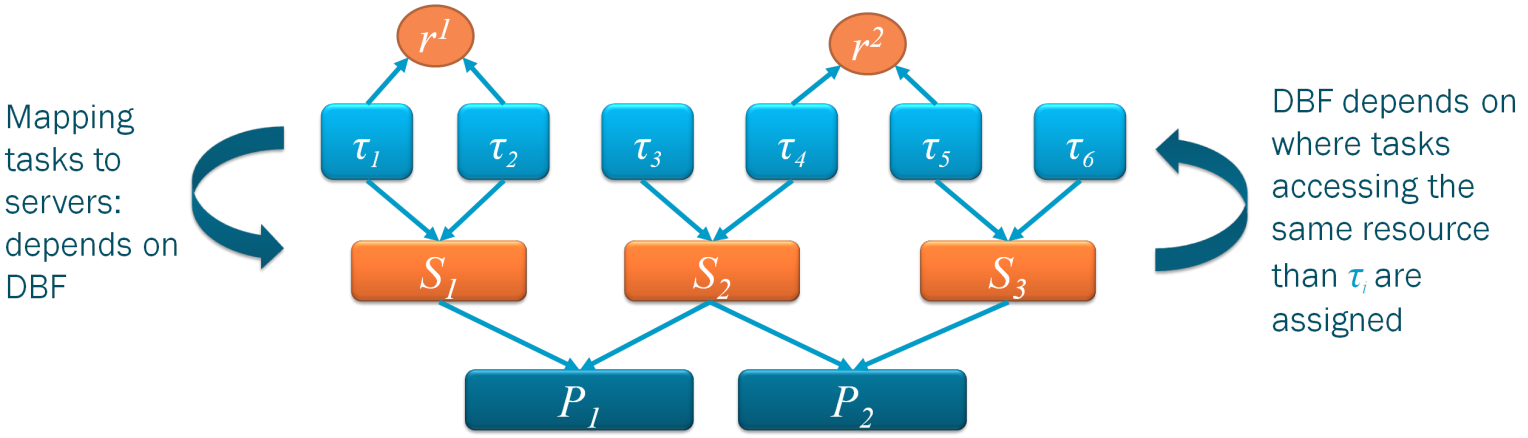
Expected result:

- *Significant reduction in the blocking time*

$\tau_1$ and $\tau_2$ inherit bandwidth of servers $S_2$ and $S_3$, *respectively*

# Future Work

1. **Prove the correctness** of the schedulability test equations provided
2. **Define** approach for mapping of the tasks to the servers:
   - Challenge → circular dependencies with the schedulability test provided
3. **Extend the approach** to any server based scheduling algorithm for multicore architectures (e.g., RUN/SPRINT)



FROM RESEARCH TO INDUSTRY

cea tech

# Externalisation of Time-Triggered communication system in BIP high level models

**H. GUESMI[1, 3], B. BEN HEDIA[1], S. BLIUDZE[2], S. BENSALEM[3]**

[1]CEA, LIST, Embedded Real Time Systems Laboratory, France
[2] RISD, Ecole polytechnique de LAUSANNE, Switzerland
[3] Verimag, Université Joseph Fourrier, France

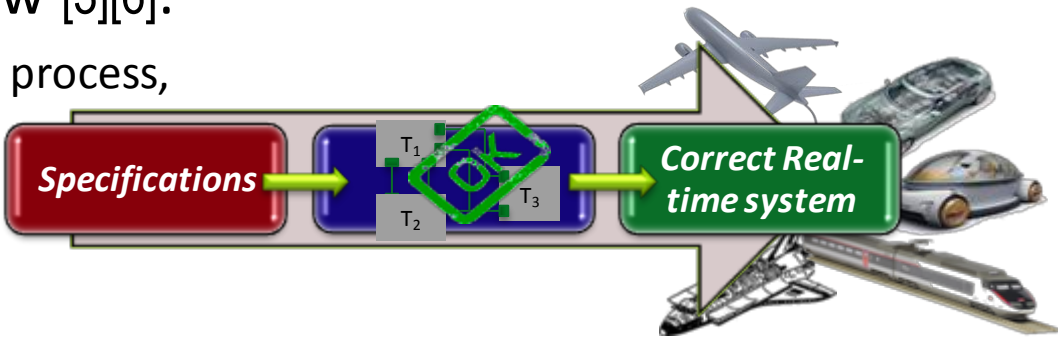
www.cea.fr

leti & list

# Introduction & motivation

- Embedded critical real time systems:
  - Increasing complexity
  - Methods using **a posteriori verification** to ensure correctness
  - → At best a major factor in the development cost and, and at worst, simply impossible.

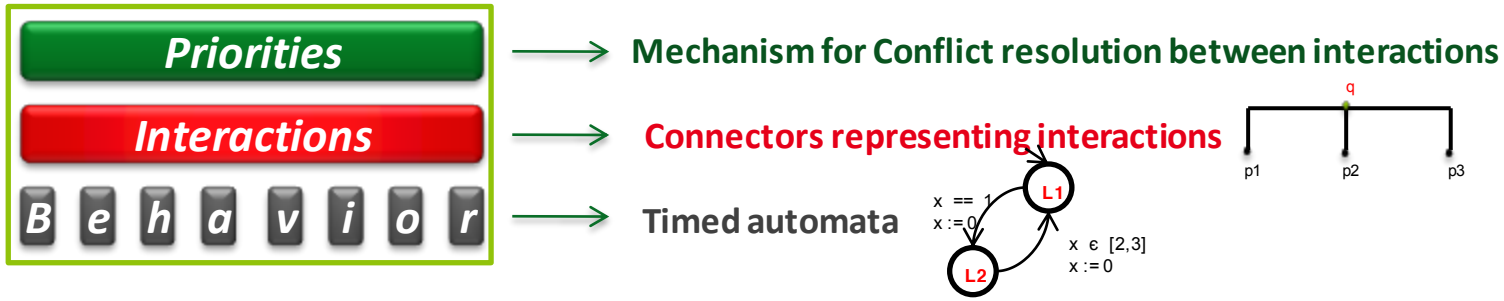
- Rigorous system design flow [3][6]:
  - Formal accountable & iterative process,
  - Component-based process,
  - Correctness-by-construction.




- **The challenge** = Apply the Rigorous Design Flow to the Time-Triggered domain
  - Correctness-by-construction
  - Predictability & determinism

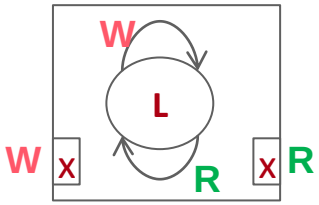
# Concepts

- BIP Framework [1][2]:
  - Structure of a real-time BIP model:



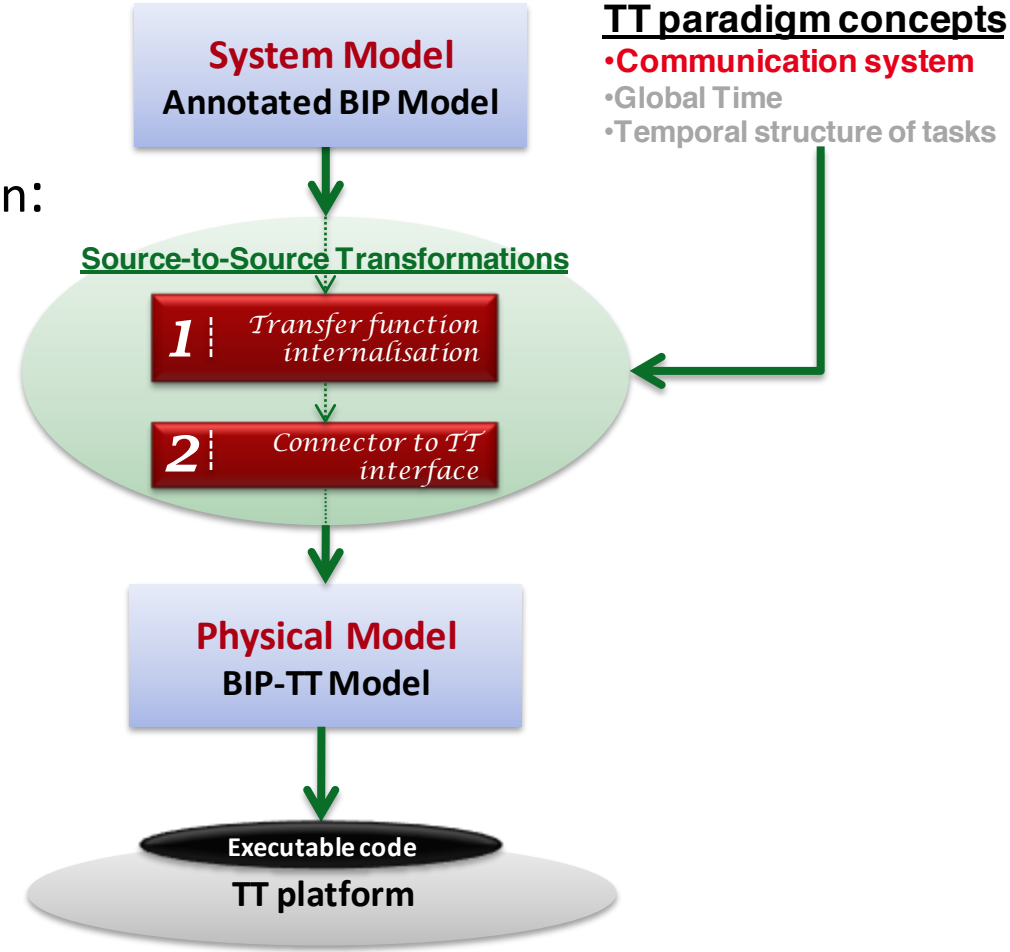

  - BIP tool chain: parser, code generators, verification and validation tools..

- Time-Triggered paradigm [4][5]:
  - **Global synchronized time:** periodic clock synchronisation
  - **Temporal control structure of tasks:** predefined start and termination instants.
  - **Time-Triggered interface:** data-sharing boundary between two communicating subsystems



# Approach

- 1/ Transfer function internalisation:
  - Both automata and connectors are modified
  - The global behavior of the model remains intact.
- 2/ Connector to TT interface:
  - Each connector is transformed into TT interface component and 2 connectors.



# Conclusion

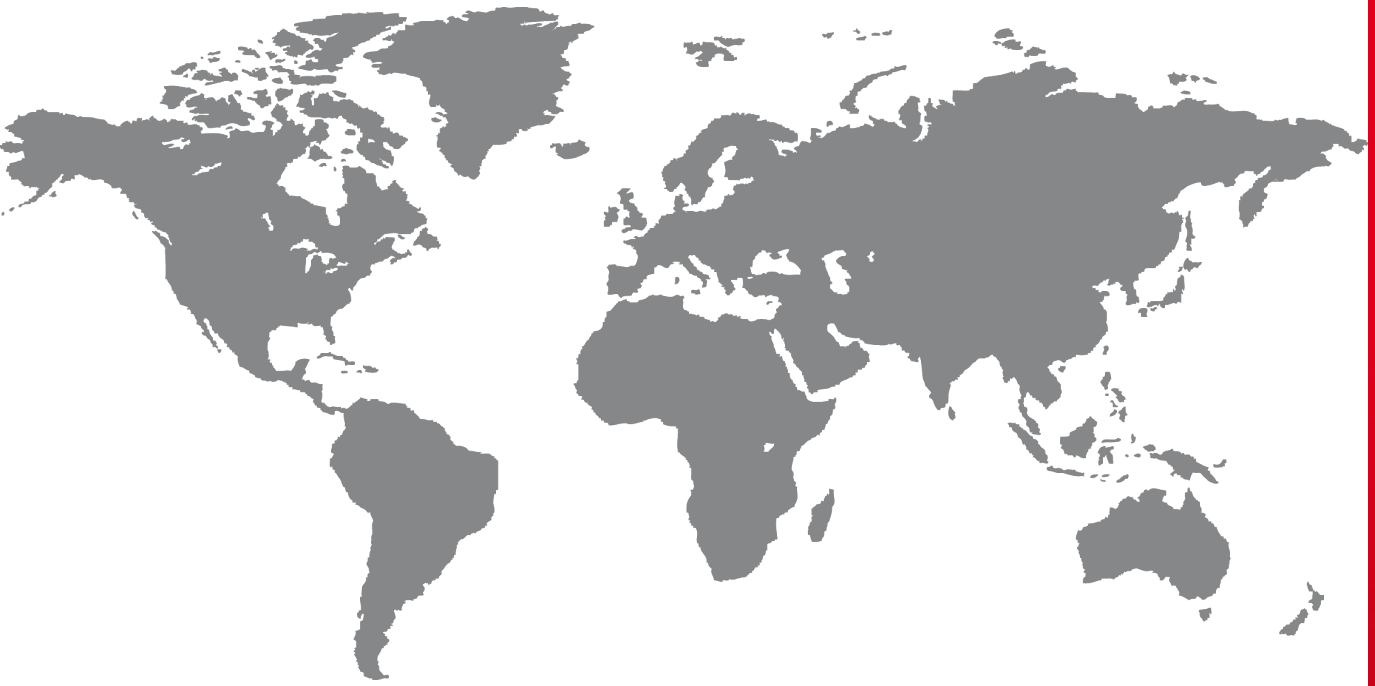
- A 2-step transformation process:
  - Simplify the connector transfer functions by modifying components automata,
  - Modify connector with simple transfer functions into TT interfaces.

  → We avoid adding new components that integrate communication specificities into the system .

  → We avoid the question: “These new components belong to which task?”

- Work in progress
  - Study the alternative approach, based on adding a communication component per connector,
  - Define a trade-off if possible,
  - Integrate other TT concepts & prove the correctness of transformations.

[1] BIP2 Documentation, July 2012.
[2] T. Abdellatif. Rigourous Implementation of real-time Systems. PhD thesis, UJF, 2012.
[3] P. Bourgos. Rigorous design flow for program-ming manycore platforms.
[4] H. Kopetz. The time-triggered approach to real-time system design. Predictably Dependable Computing Systems. Springer, 1995.
[5] H. Kopetz. The time-triggered model of computation. In Real-Time Systems Symposium, 1998. Proceedings
.,The 19th IEEE, pages 168–177. IEEE, 1998.
[6] J. Sifakis. Rigorous system design. Foundations and Trends R in Electronic Design Automation, 6(EPFL-ARTICLE-185999):293–362, 2012.



# Thank you!

leti
Centre de Grenoble
17 rue des Martyrs
38054 Grenoble Cedex

list
Centre de Saclay
Nano-Innov PC 172
91191 Gif sur Yvette Cedex

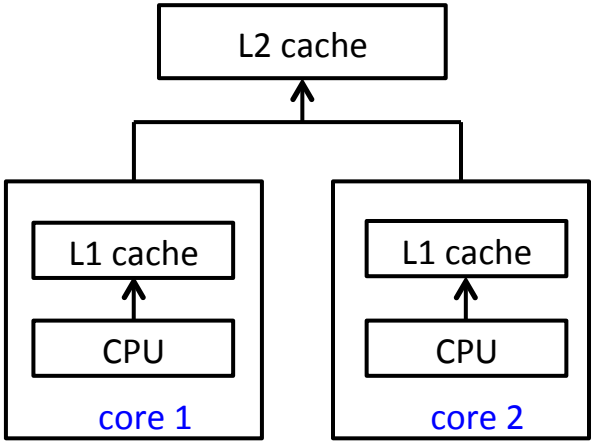
# Towards Exploiting Limited Preemptive Scheduling for Partitioned Multicore Systems

Abhilash Thekkilakattil, Radu Dobrin and Sasikumar Punnekkat

MÄLARDALEN UNIVERSITY
SWEDEN

Vetenskapsrådet

# Introduction and Motivation

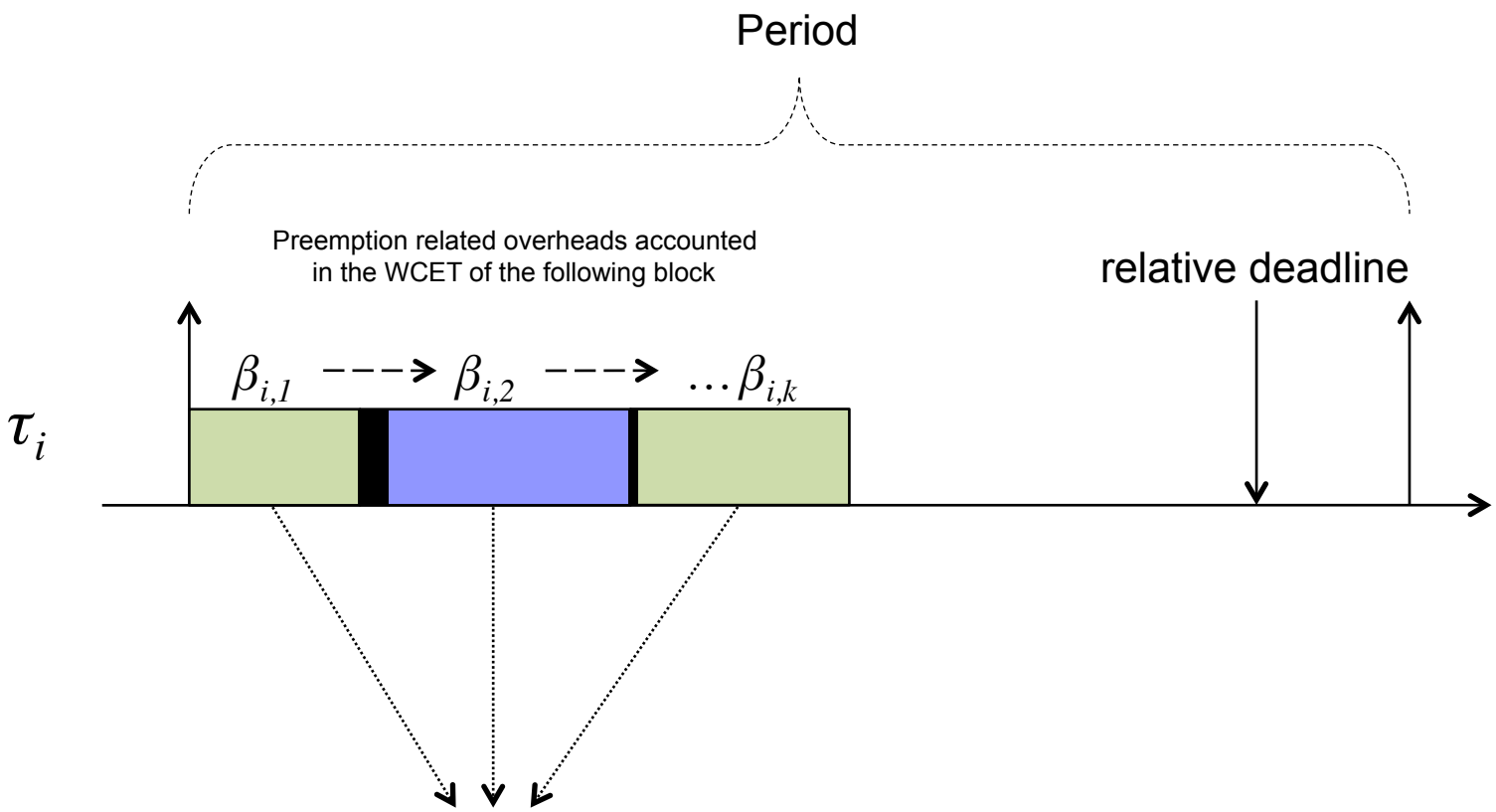



- Shared hardware in multicore systems
  - Caches
  - Buses … etc

- Increases analysis pessimism in multicore schedulability analysis
  - Difficulties in bounding the Worst Case Execution Time due to resource contention

Solution: enforce temporal separation

# System Model



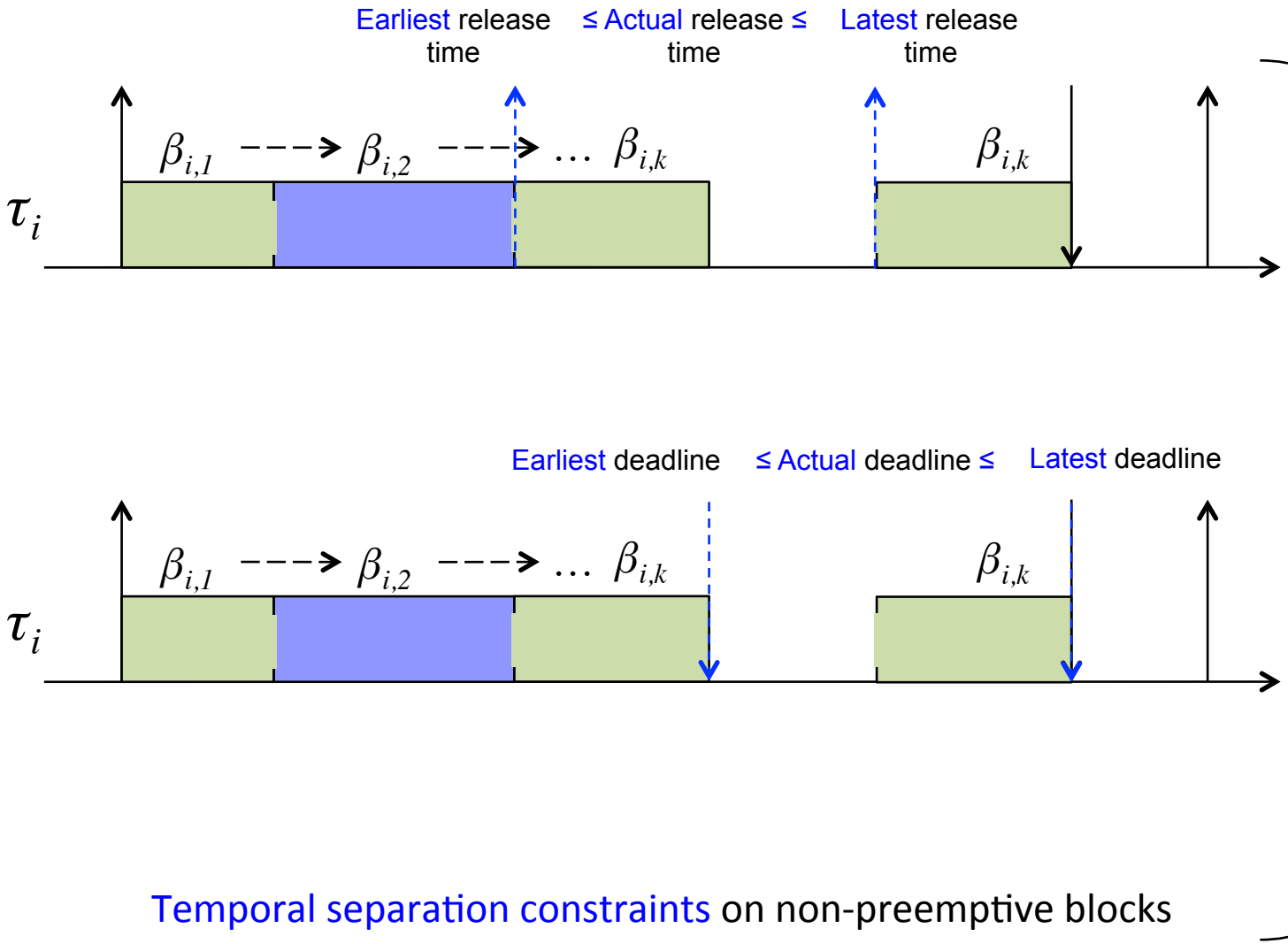

Non-preemptive blocks:

- Optimal preemption points
- Regions of code accessing shared resource e.g., cache

regions of code that do not access L2 cache

regions of code that access L2 cache

# Feasibility Window Derivation



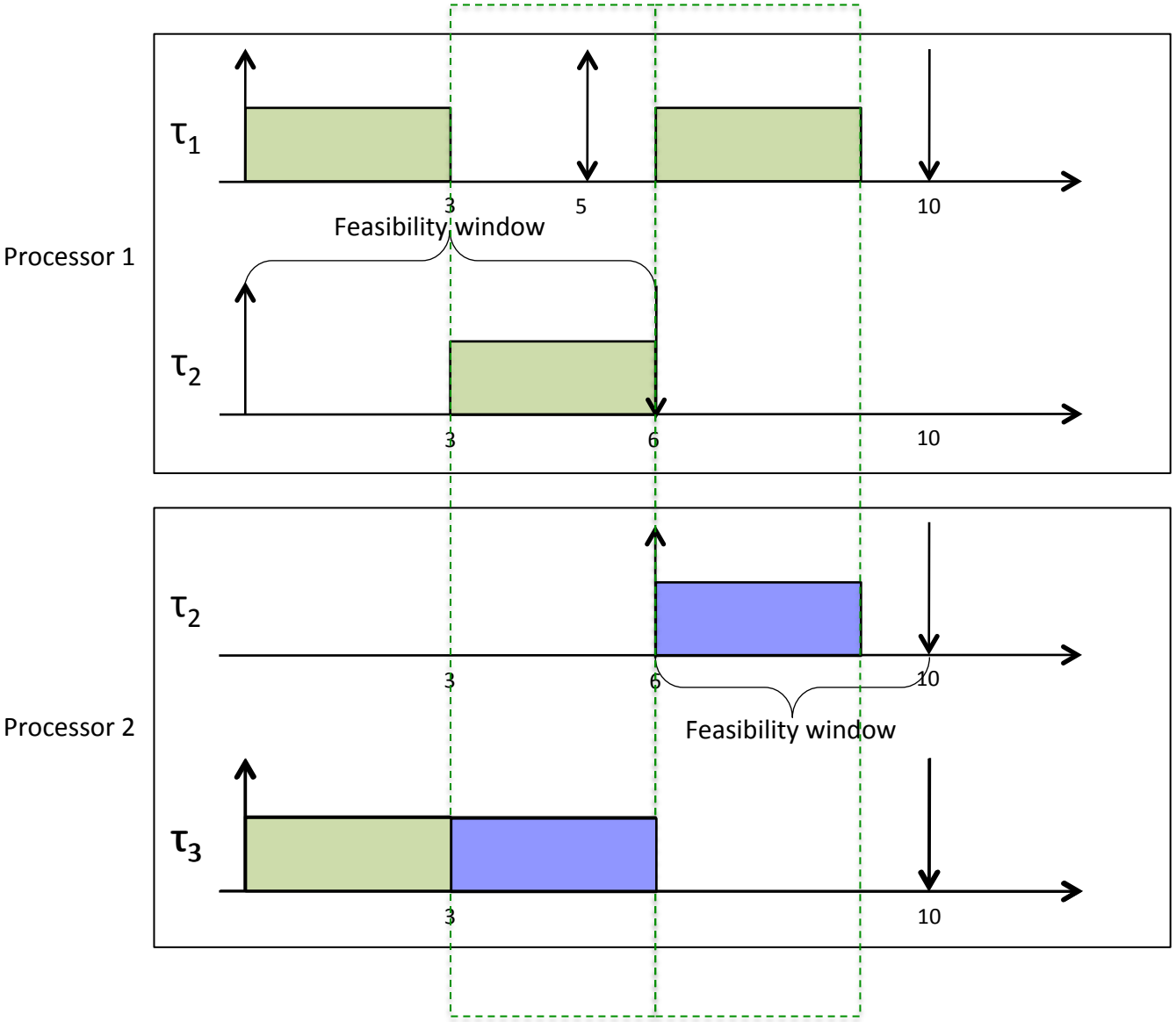

Temporal separation constraints on non-preemptive blocks

Mathematical Optimization

Minimize number of cores

Release time, Deadline and a processor per block

# Example




| Task | $\beta_{i,k}$ | $C_i$ | $T_i$ |
|---|---|---|---|
| $\tau_1$ | $\beta_{1,1}=3$ | 3 | 5 |
| $\tau_2$ | $\beta_{2,1}=3$<br>$\beta_{2,2}=3$ | 6 | 10 |
| $\tau_3$ | $\beta_{3,1}=3$<br>$\beta_{3,2}=3$ | 6 | 10 |

$U_{tot} = 1.8$

regions of code that do not access L2 cache

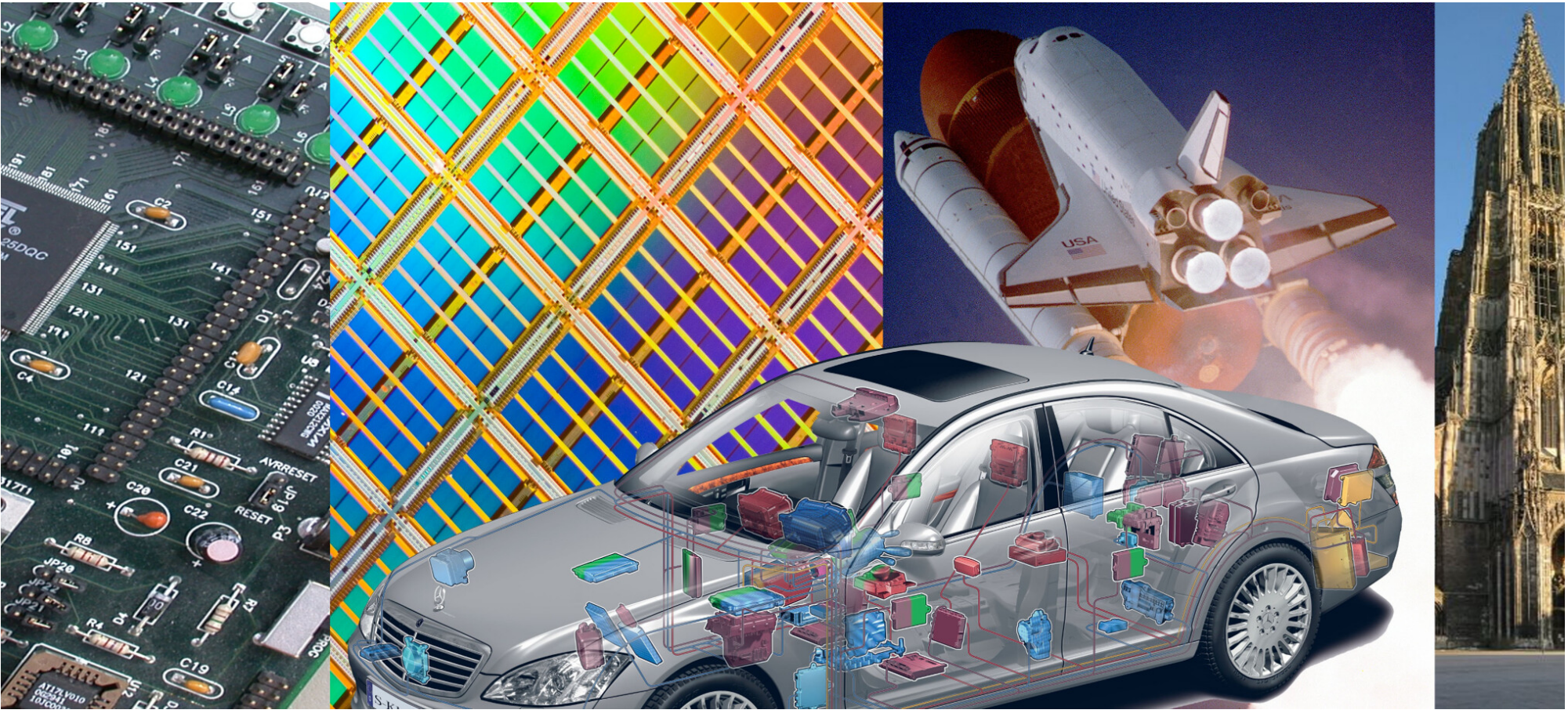
regions of code that access L2 cache

# Summary and Future Work

- Limited preemptive scheduling for efficient utilization of multicore platforms
  - Input: Task parameters
  - Output: Pseudo release times and deadlines for non-preemptive blocks, and a processor that guarantees temporal separation

- Implement and evaluate the approach
  - Exact solution on minimum number of cores Vs. heuristics (that may use few more cores)

ulm university universität ulm



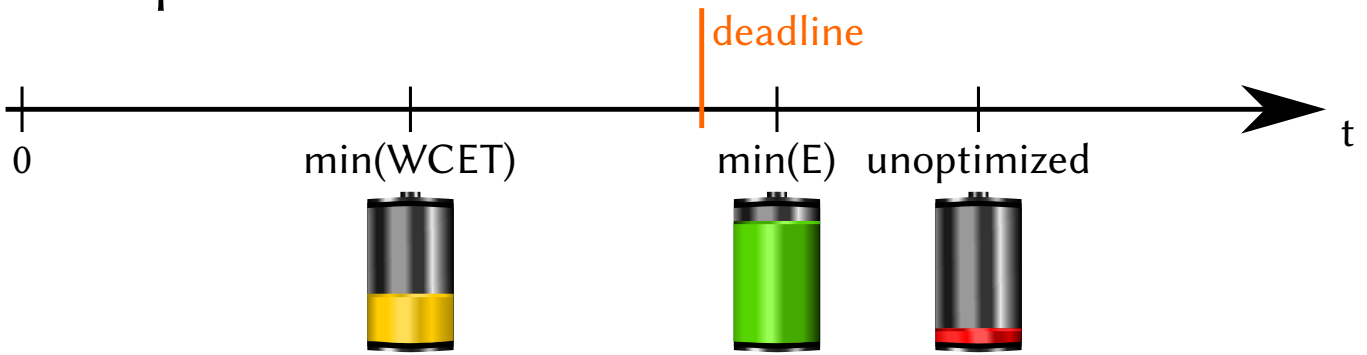

# Multi-Criteria Optimization of Hard Real-Time Systems

Nicolas Roeser, Arno Luppold and Heiko Falk
JRWRTC, 2014-09-09

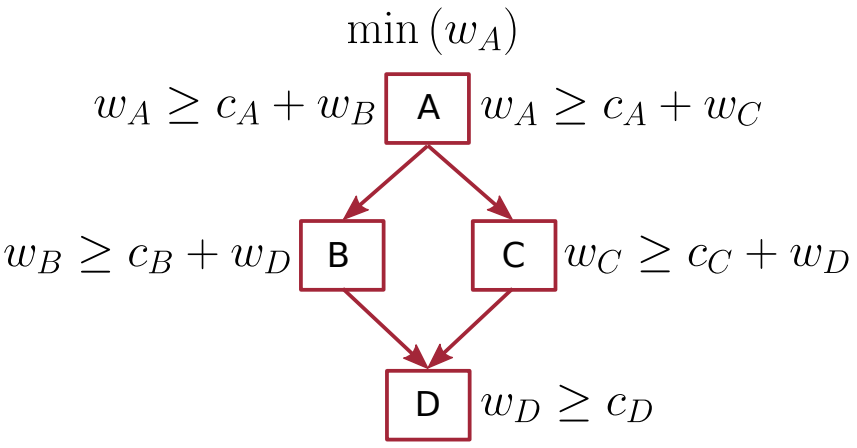
# Optimization of Software for Embedded Systems

- WCET $\overset{!}{<}$ deadline
- other constraints,
  e.g. low energy consumption ($\Rightarrow$ longer battery life)

Possible compilation results:



# ILP Model




V. Suhendra et al. *WCET Centric Data Allocation to Scratchpad Memory.* RTSS 2005.

Do not *minimize* the WCET, but constrain it:

$$w^*_{\texttt{main}} \leq D$$

Add function specialization (size ↑), and program SPM allocation (WCET ↓, energy consumption ↓).

Further constraints, like for energy:

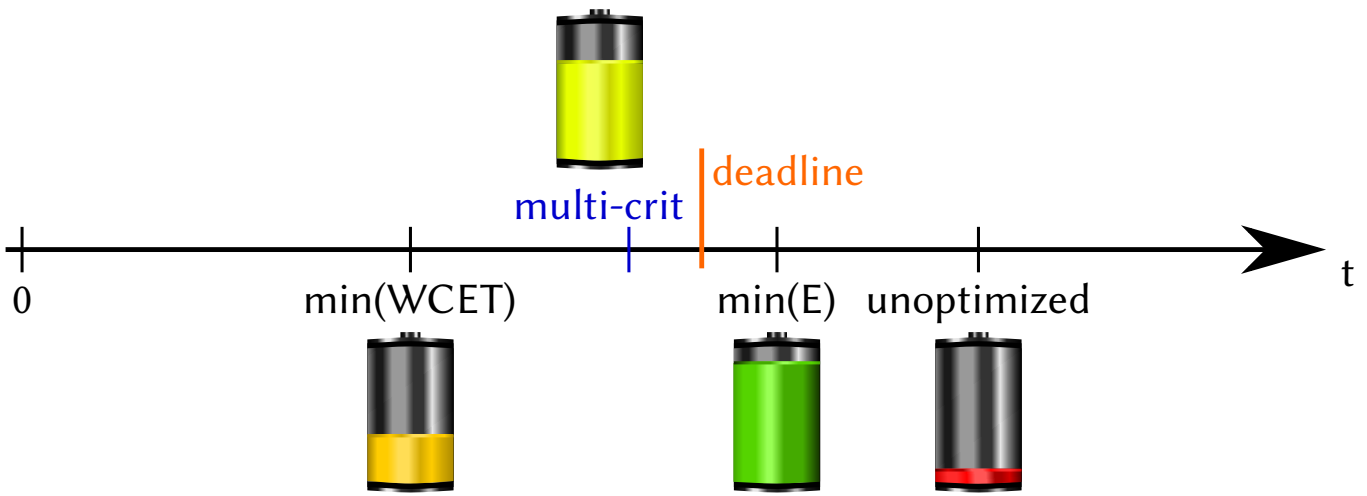
$$e^*_f \geq N_{f,f} \cdot E_f + \sum_{g \in \mathcal{F}} N_{f,g} \cdot \left( e^*_{g_0} \cdot p_g + e^*_g \cdot (1 - p_g) \right)$$

# Optimization Results

Solving the ILP problem sets binary decision variables for the combined optimizations:

- specialize the function?
- put the function into SPM?

Multi-criteria optimization results:



# Conclusion

- ILP-based multi-criteria optimization for hard real-time systems $\Rightarrow$ relaxed timing with energy and/or memory savings,
- optimum solution;
- other/further constraints and other optimizations can be used as long as they can be described with ILP formulae.

Future Work:

- generic multi-criteria optimization framework in compiler,
- ability to handle multi-tasking systems.