Energy Saving in Real-Time Embedded Systems

Mario Bambagini

ReTiS Lab, TeCIP Institute
Scuola Superiore Sant'Anna, Pisa, Italy

September 26, 2014

Outline

1. Introduction
   - Problem formulation
   - Contribution
2. System model
   - Application
   - Hardware
   - Real-Time Operating Systems
3. Energy-Aware Scheduling on single core systems
   - Motivational example
   - System model
   - Algorithm
   - Experimental results
4. Conclusions
   - Presented work
   - PhD results

Presentation

Mario Bambagini

Current:
SW engineer @ Magneti Marelli

Apr 2014 — Sept 2014:
Research fellow @ Unimore

Oct 2010 — today:
PhD student @ SSSUP

Sept 2004 — May 2010:
BSC and MSC @ Unipi

Problem formulation

Meaning of "Energy Saving in Real-Time Embedded Systems"

Problem formulation

Meaning of "Energy Saving in Real-Time Embedded Systems"

Embedded Systems

Specific purpose systems with limited resources (such as computational capabilities, memory, energy suppliers)

Problem formulation

Meaning of "Energy Saving in Real-Time Embedded Systems"

Embedded Systems

Specific purpose systems with limited resources (such as computational capabilities, memory, energy suppliers)

Real-Time Scheduling

Correctness depends also on when data is produced (not only on values). The scheduler must select next job to execute carefully

Problem formulation

Meaning of "Energy Saving in Real-Time Embedded Systems"

Embedded Systems

Specific purpose systems with limited resources (such as computational capabilities, memory, energy suppliers)

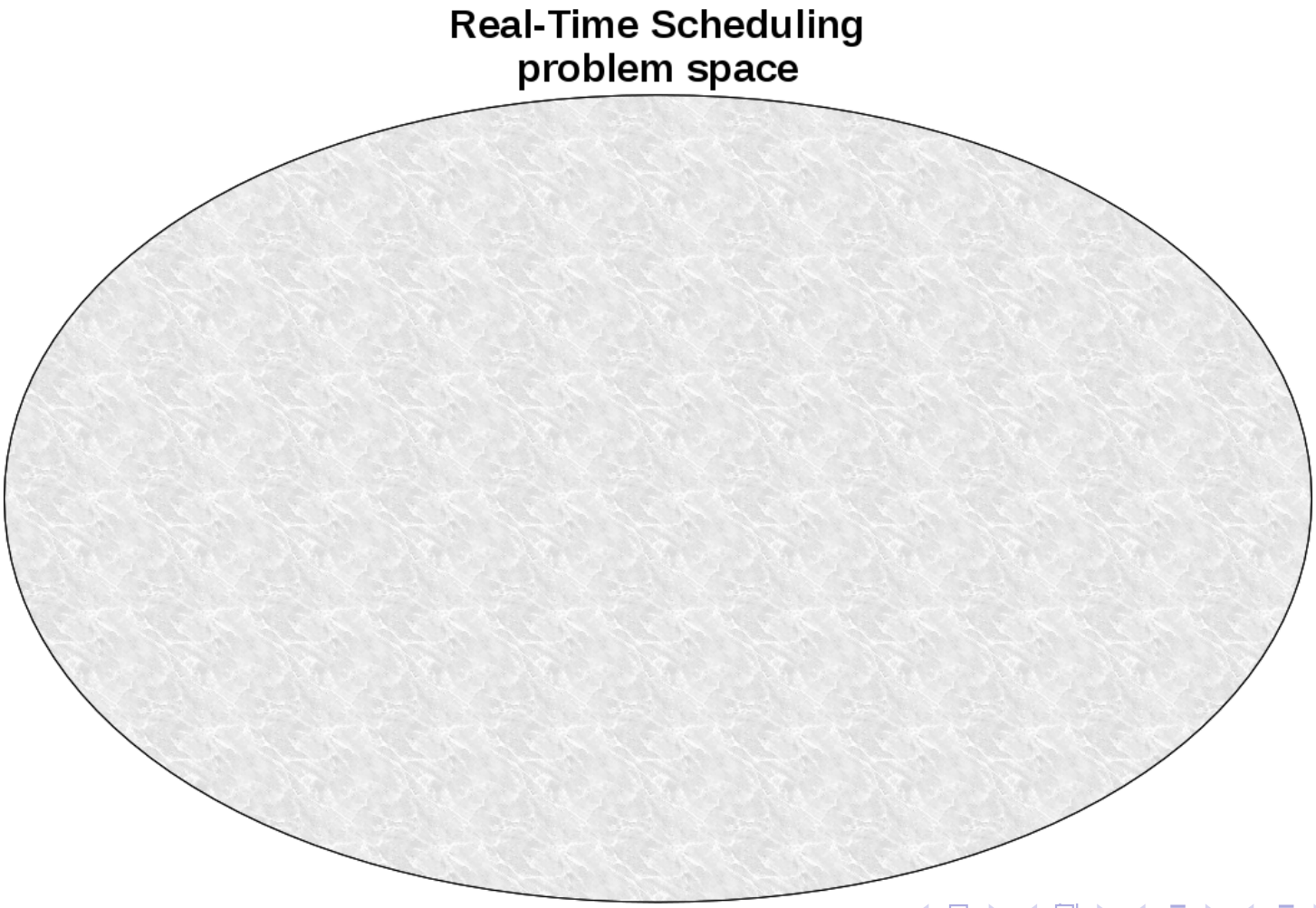
Real-Time Scheduling

Correctness depends also on when data is produced (not only on values). The scheduler must select next job to execute carefully

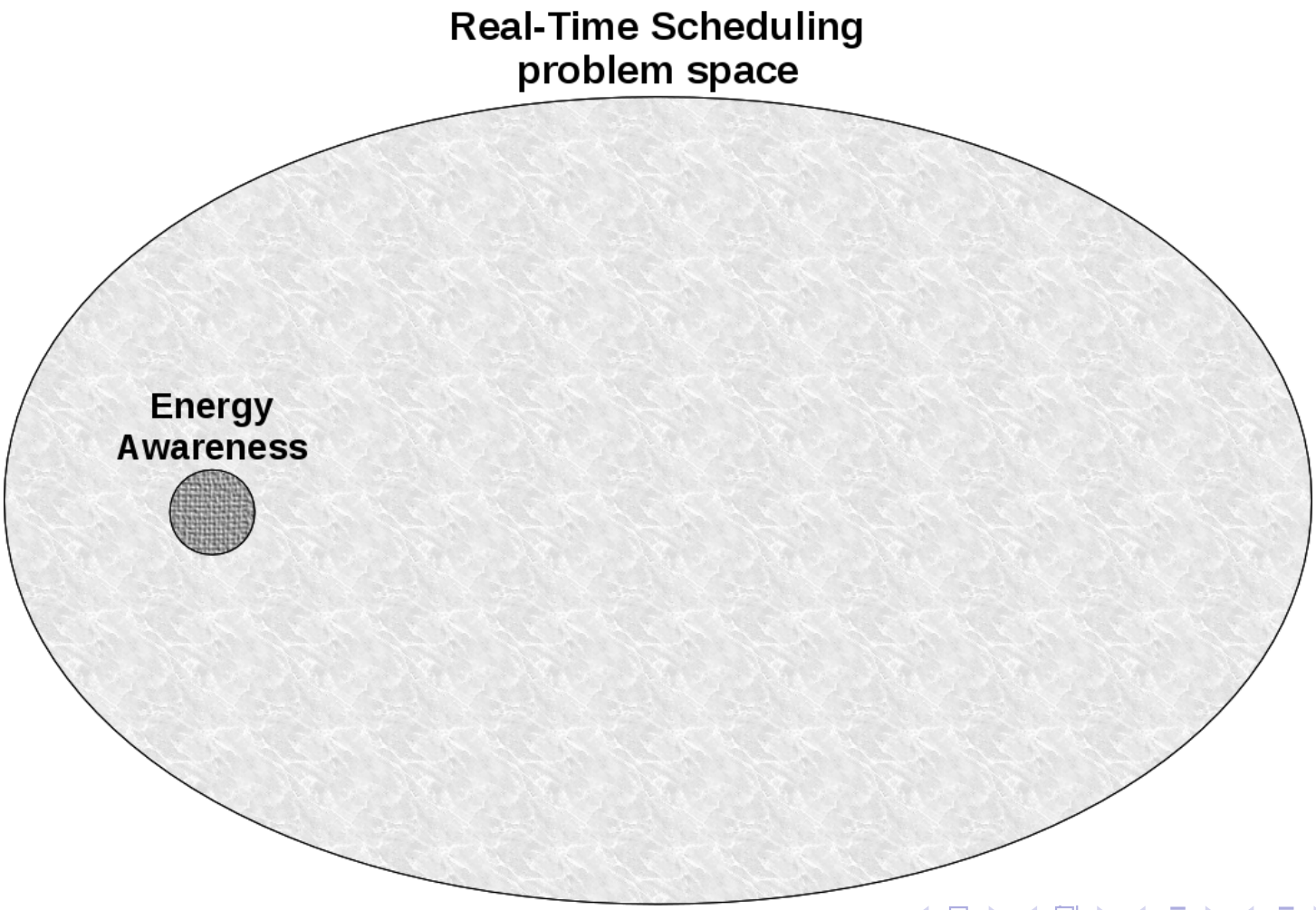
Energy Saving

Besides the correct execution (w.r.t. values and time), the overall energy consumption has to be minimized

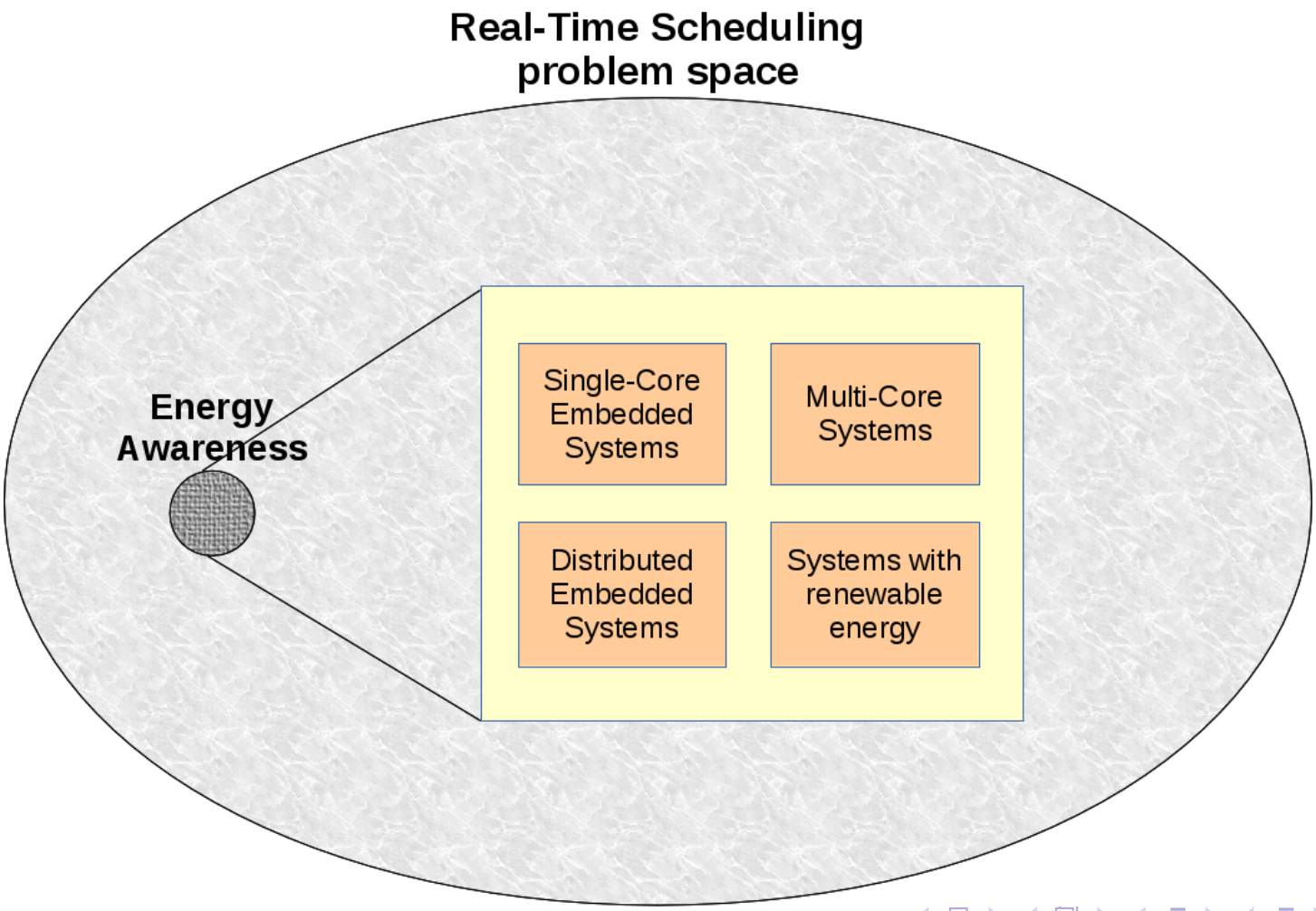
Contribution



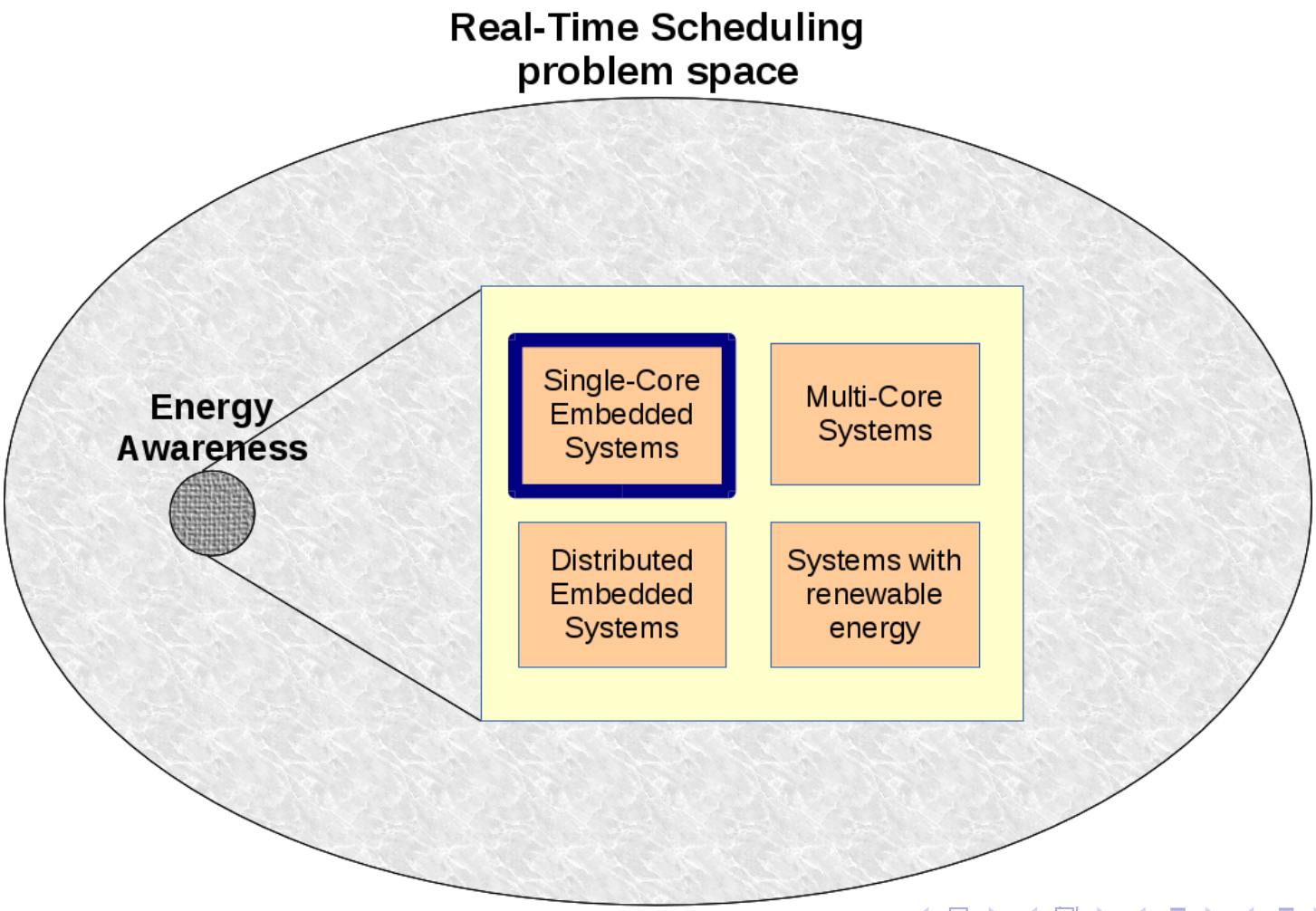
Contribution



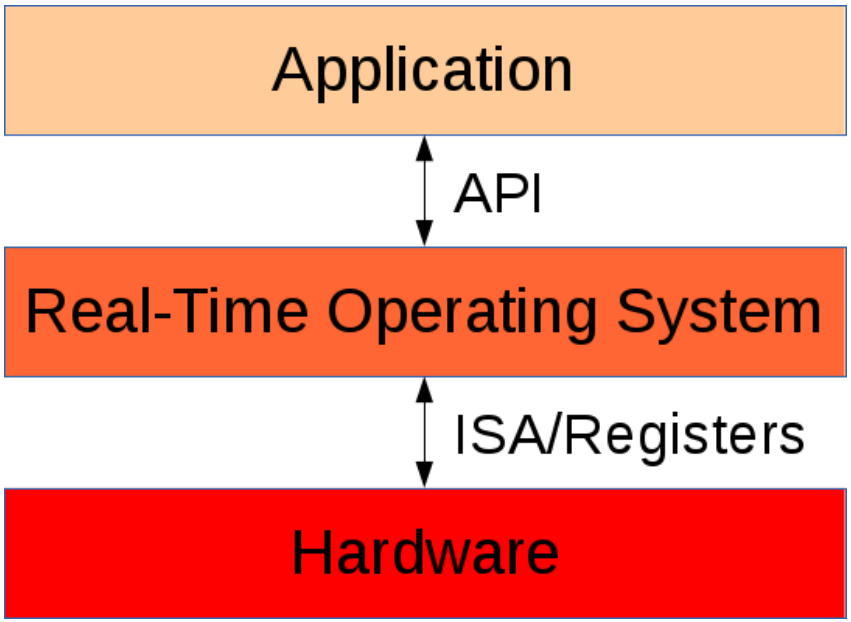
Contribution



Contribution



Embedded systems overview



Application:

- Control algorithm
- User Interface

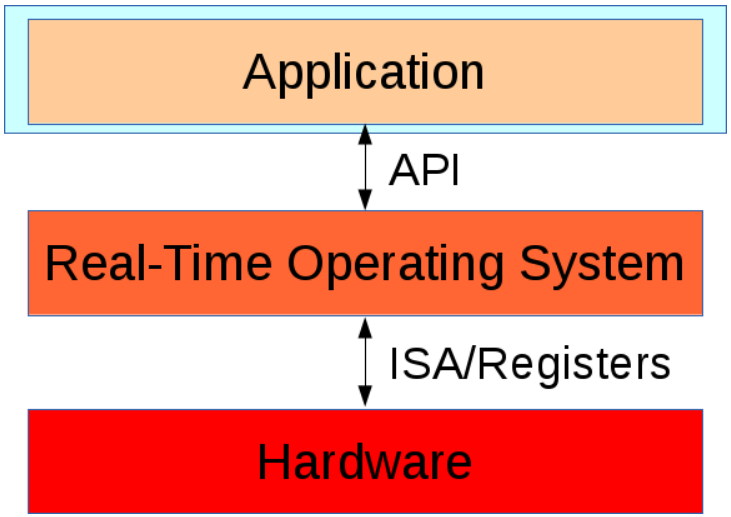
Operating systems:

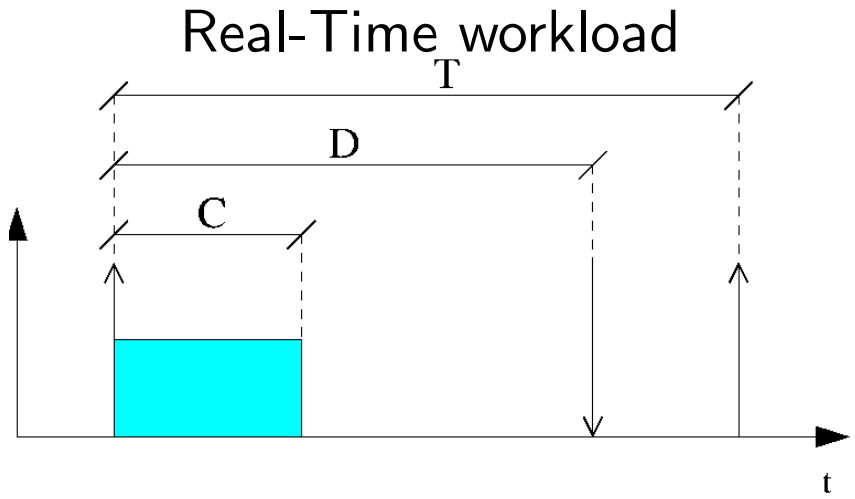
- Linux / RTOS / Bare metal
- Drivers
- Communication protocols

Hardware

- MCUs
- Transceivers
- Sensors / Actuators

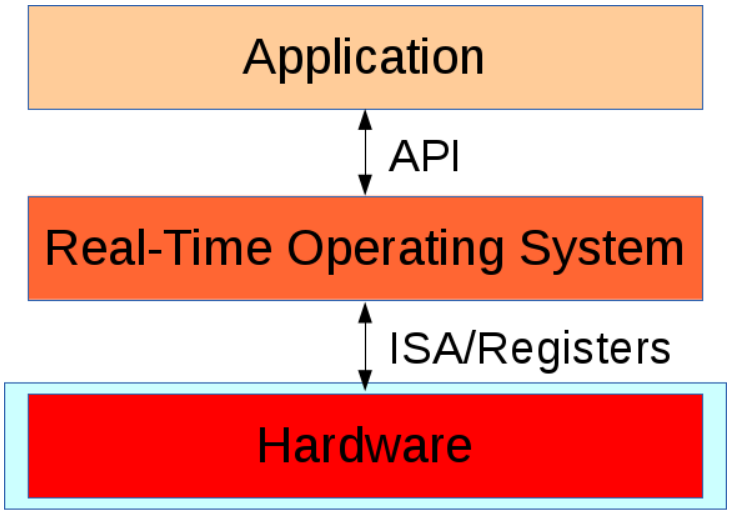
Application



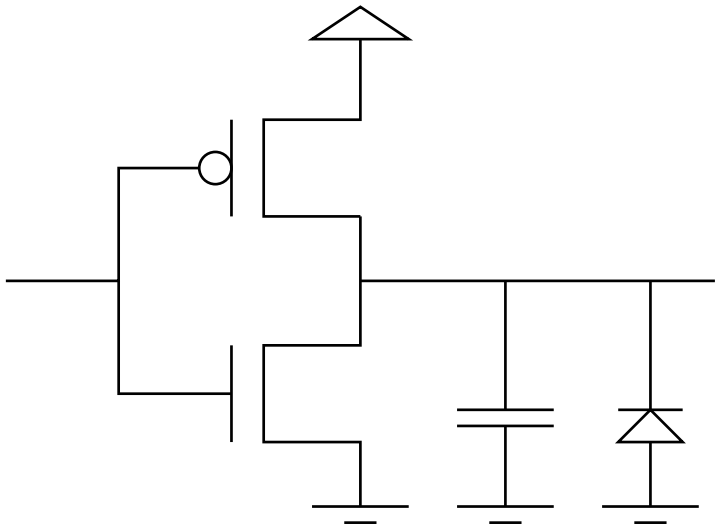


- T: minimum inter-arrival time (aka period)
- D: deadline
- C: Worst-Case Execution Time (WCET)

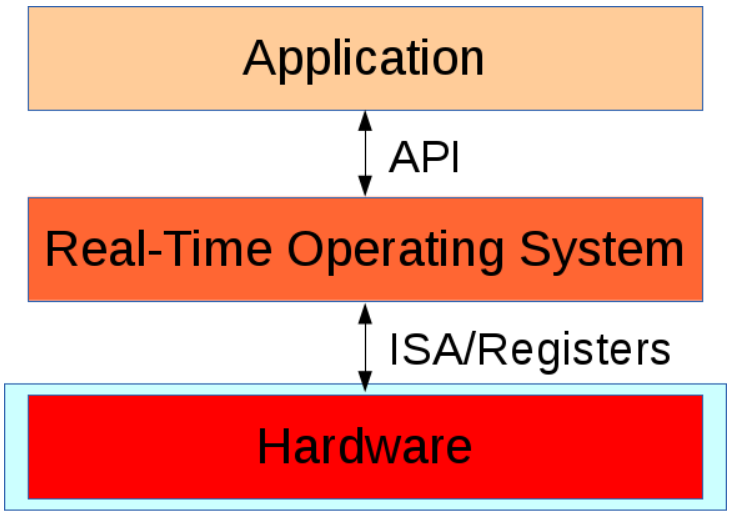
Hardware



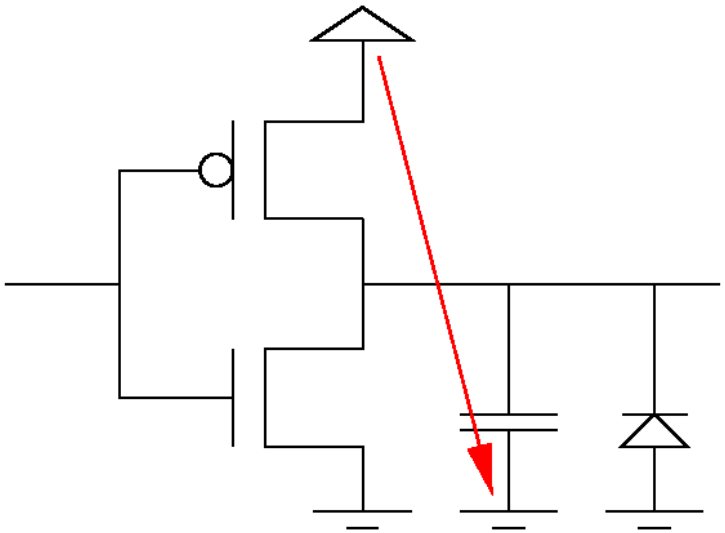
CMOS Technology



Hardware

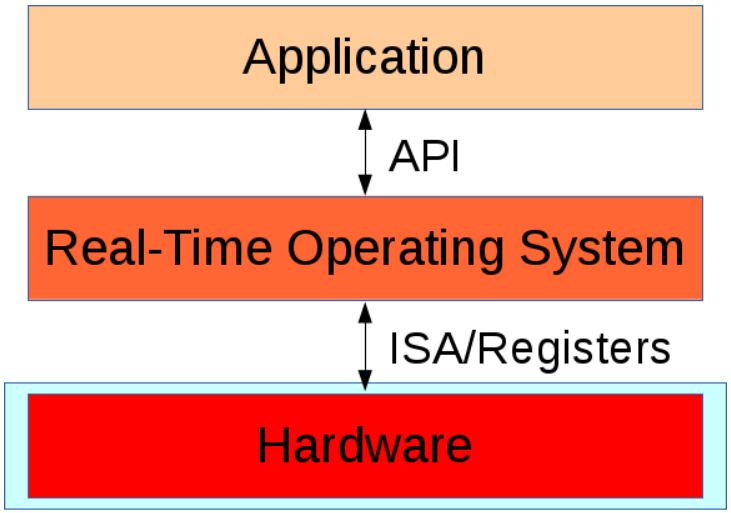


CMOS Technology

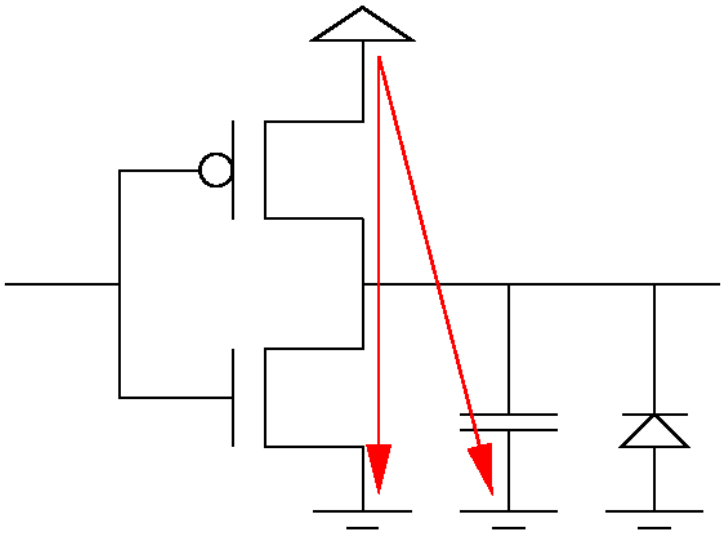


$$P_{gate} = \alpha \times C_L \times V^2 \times f +$$

Hardware

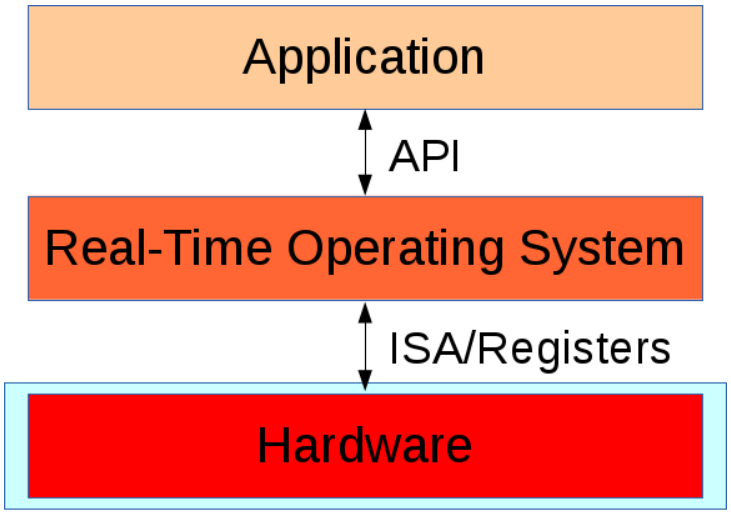


CMOS Technology

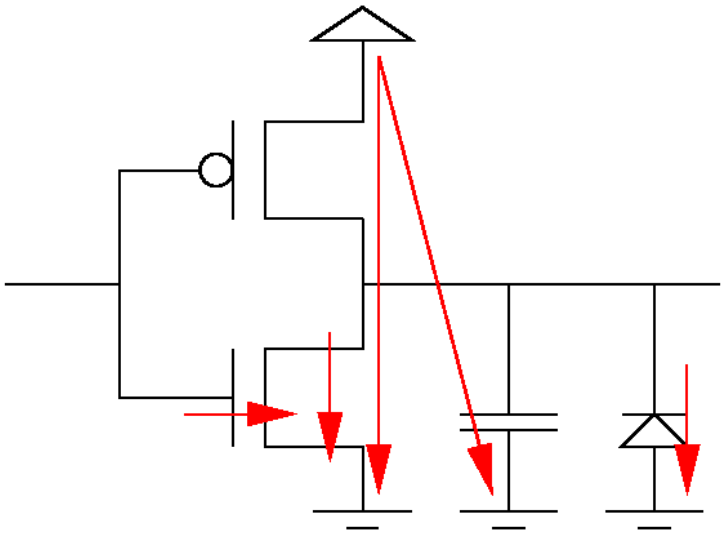


$$P_{gate} = \alpha \times C_L \times V^2 \times f + \\ + \alpha \times V \times I_{short}$$

Hardware



CMOS Technology

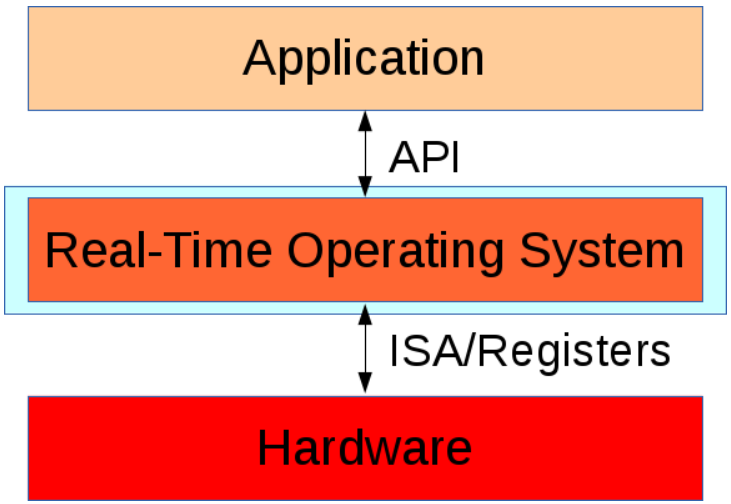


$$P_{gate} = \alpha \times C_L \times V^2 \times f + \\ + \alpha \times V \times I_{short} + V \times I_{leak}$$

Variables: Voltage (V), Frequency (f)

$$P(f) = k_3 f^3 + k_2 f^2 + k_1 f + k_0$$

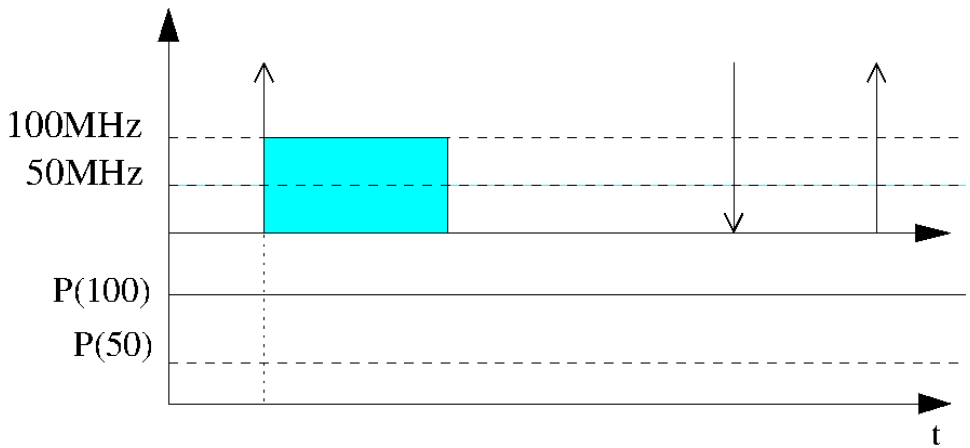
Real-Time Operating Systems



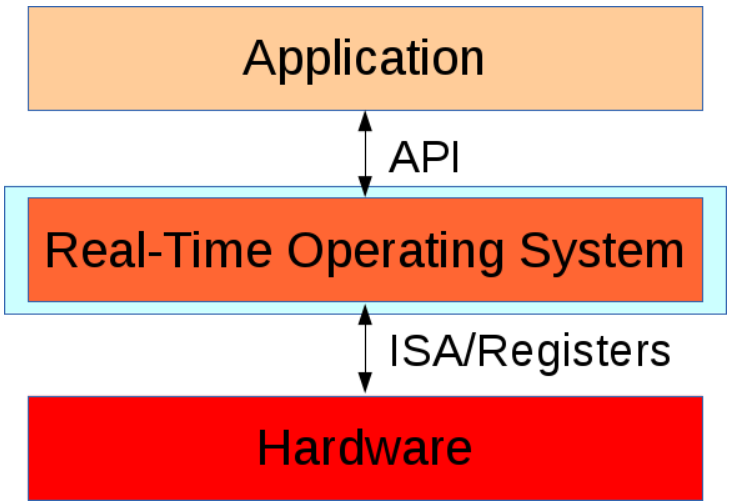
Real-Time Operating Systems

Scheduling techniques:

- DVFS: Dynamic Voltage and Frequency Scaling
- DPM: Dynamic Power Management



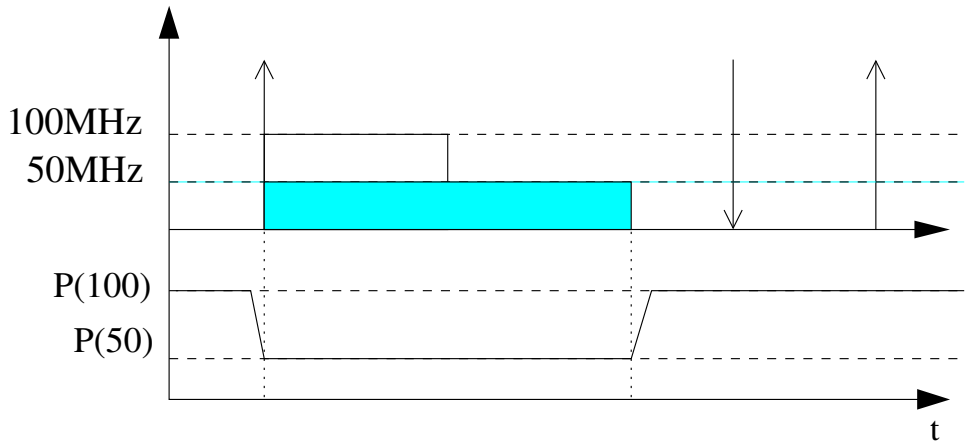
Real-Time Operating Systems



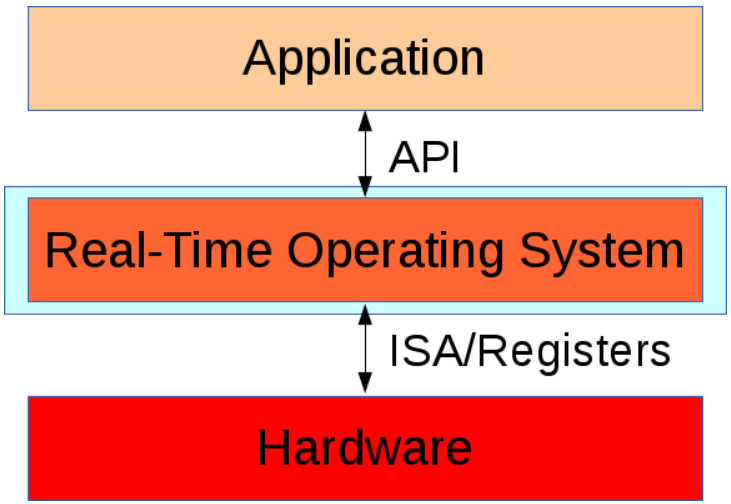
Real-Time Operating Systems

Scheduling techniques:

- **DVFS**: Dynamic Voltage and Frequency Scaling
- DPM: Dynamic Power Management



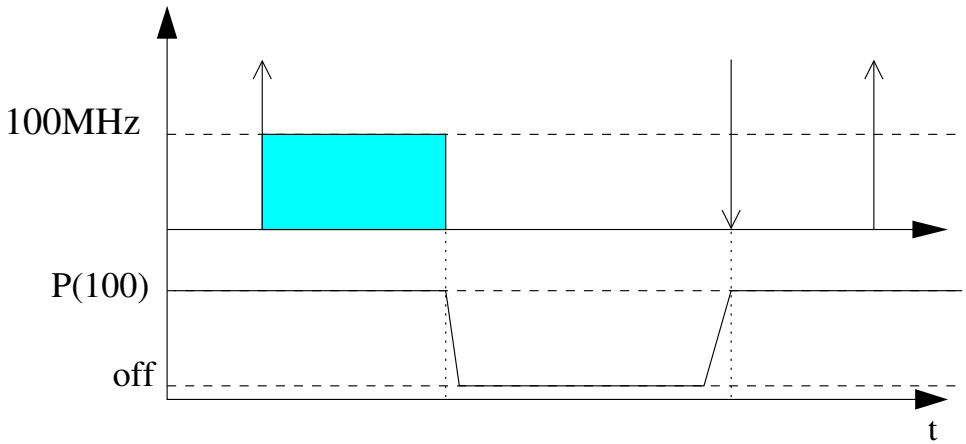
Real-Time Operating Systems



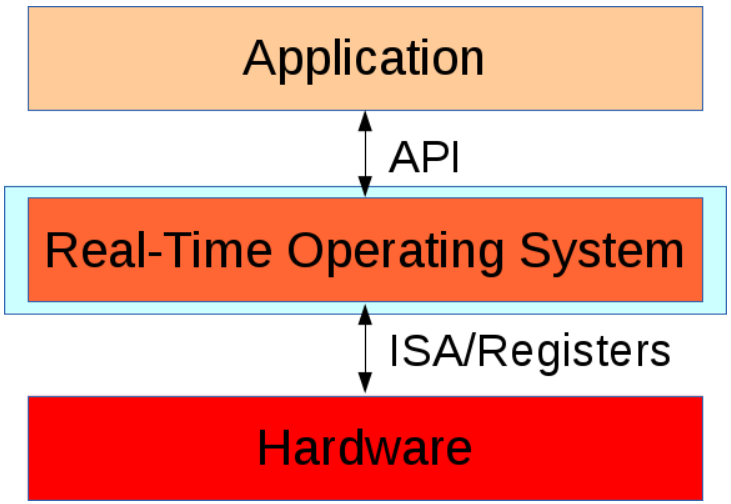
Real-Time Operating Systems

Scheduling techniques:

- DVFS: Dynamic Voltage and Frequency Scaling
- **DPM**: Dynamic Power Management



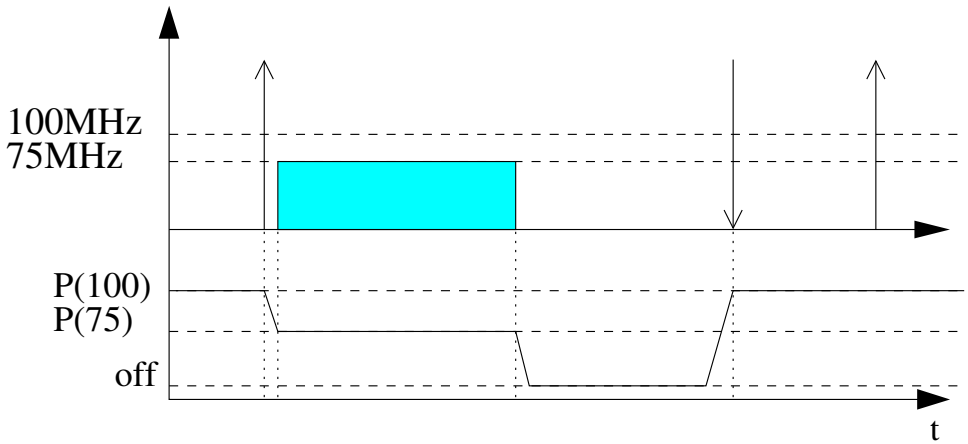
Real-Time Operating Systems



Real-Time Operating Systems

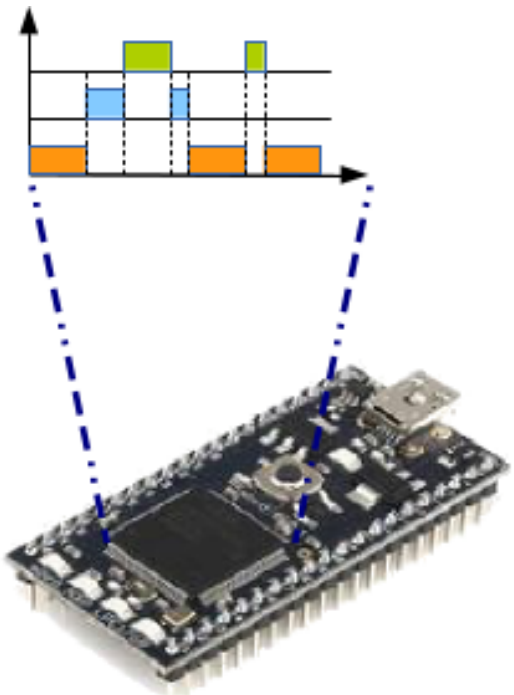
Scheduling techniques:

- **DVFS**: Dynamic Voltage and Frequency Scaling
- **DPM**: Dynamic Power Management



Introduction

This work presents an energy saving scheduling algorithm for periodic real-time tasks on single core fixed-priority systems



Main contributions:

- mix of DVFS and DPM
- platform aware
- preemption aware
- limited preemptive task model
- low online complexity

Paper: "An Energy-Aware Algorithm Exploiting Limited Preemptive Scheduling under Fixed Priorities"

Why the limited preemptive model?

Non-preemptive systems:

- simple RTOSs
- no race condition
- no preemption overhead

Fully-preemptive systems:

- more complex RTOSs
- race conditions
- preemption overhead

Why the limited preemptive model?

Non-preemptive systems:

- simple RTOSs
- no race condition
- no preemption overhead
- CPUs are poorly exploited

Fully-preemptive systems:

- more complex RTOSs
- race conditions
- preemption overhead
- CPUs are better exploited

Which is the best?

Why the limited preemptive model?

Non-preemptive systems:

- simple RTOSs
- no race condition
- no preemption overhead
- CPUs are poorly exploited

Fully-preemptive systems:

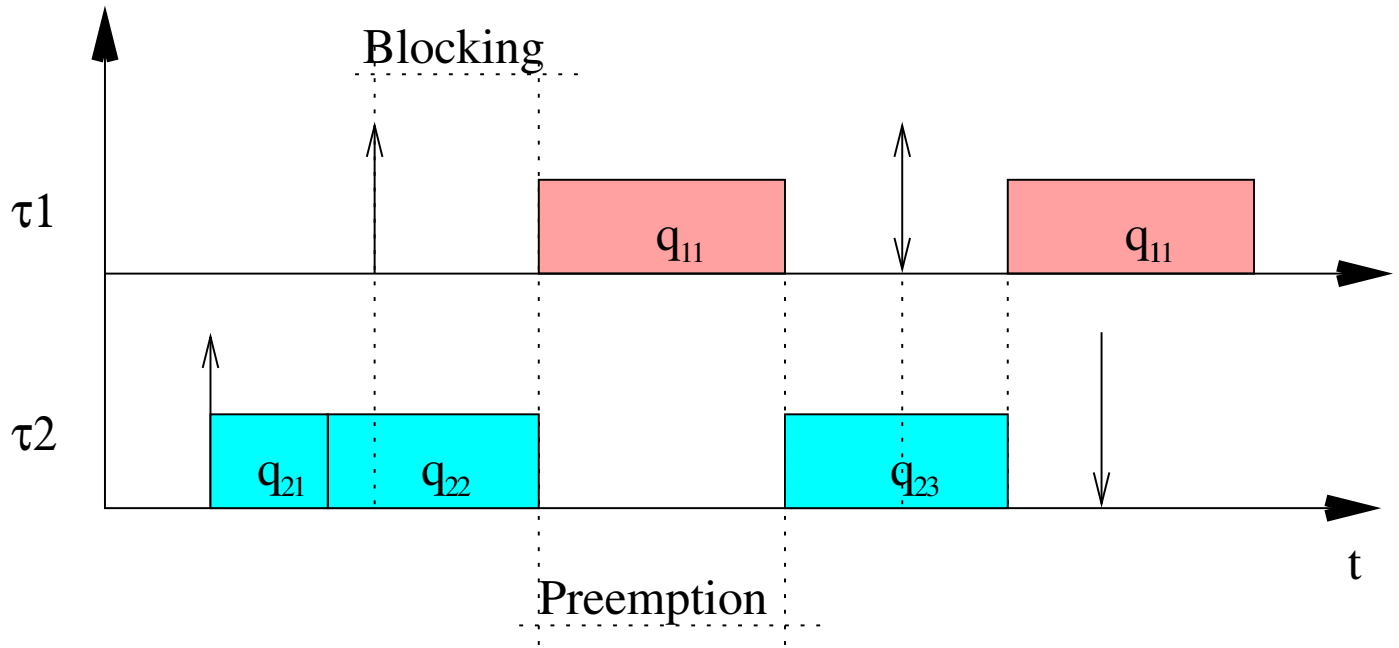
- more complex RTOSs
- race conditions
- preemption overhead
- CPUs are better exploited

Which is the best?

No winner in the embedded system domain

Why the limited preemptive model?

The Limited Preemptive task model: a fair trade-off



$q_{i,j}(s)$: length of the j-th non preemptive region of τ_i
Preemption Point Placement: Bertogna, Buttazzo and Yao

Motivational example

Consider a system with $\xi = 0$, two speeds ($s_1 = 0.5$ and $s_2 = 1$) and a task set Γ, scheduled using RM:

- τ_1: high priority, $C_1^{NP} = 30$ (at $s = 1$) and $T_1 = D_1 = 80$
- τ_2: low priority, $C_2^{NP} = 25$ (at $s = 1$) and $T_2 = D_2 = 200$

Motivational example

Consider a system with $\xi = 0$, two speeds ($s_1 = 0.5$ and $s_2 = 1$) and a task set Γ, scheduled using RM:

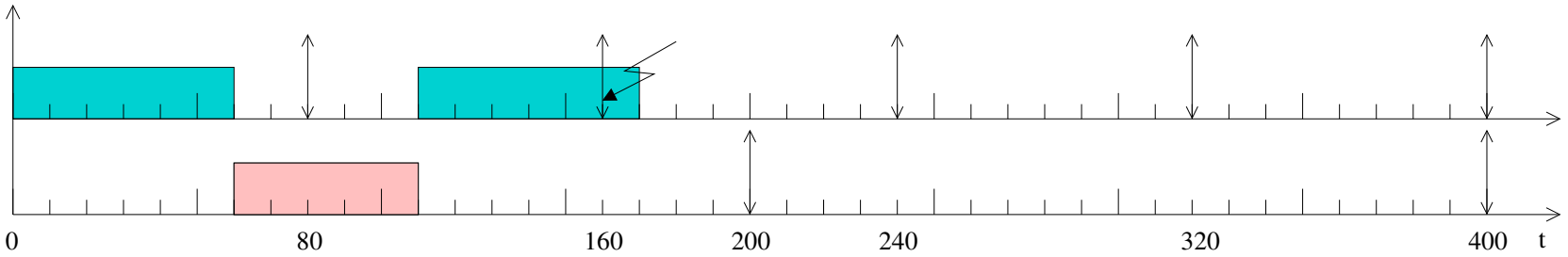
- τ_1: high priority, $C_1^{NP} = 30$ (at $s = 1$) and $T_1 = D_1 = 80$
- τ_2: low priority, $C_2^{NP} = 25$ (at $s = 1$) and $T_2 = D_2 = 200$

At $s = s_2 = 1$ ($U = 0.5$), the task set is feasible with the models:

- Non-Preemptive
- Fully-Preemptive
- Limited-Preemptive

Motivational example

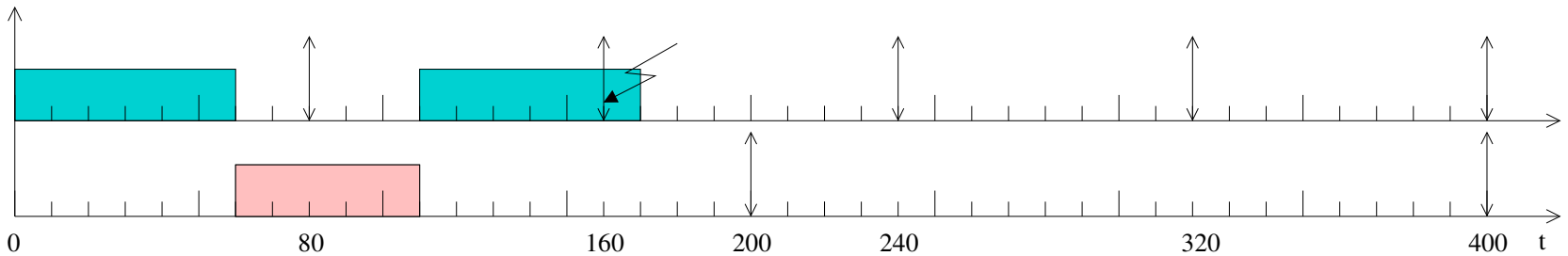
At $s = s_1 = 0.5$ ($\tau_1 : 60/80, \tau_2 : 50/200,\ U = 1$):



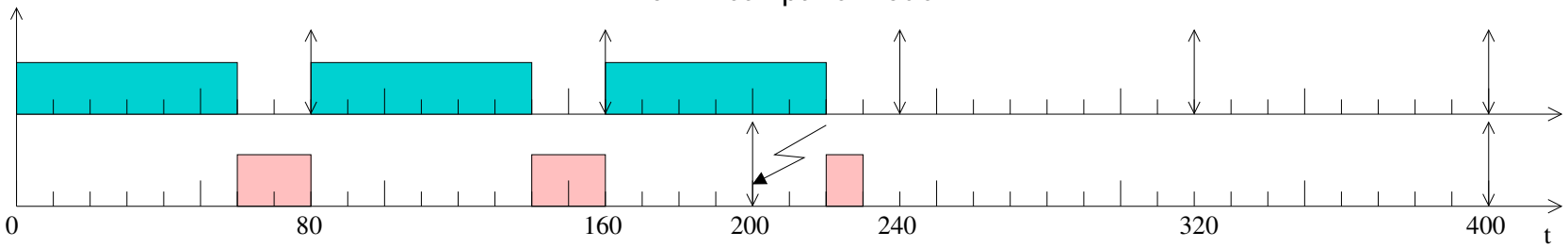
Non-Preemptive model

Motivational example

At $s = s_1 = 0.5$ ($\tau_1 : 60/80, \tau_2 : 50/200,\ U = 1$):



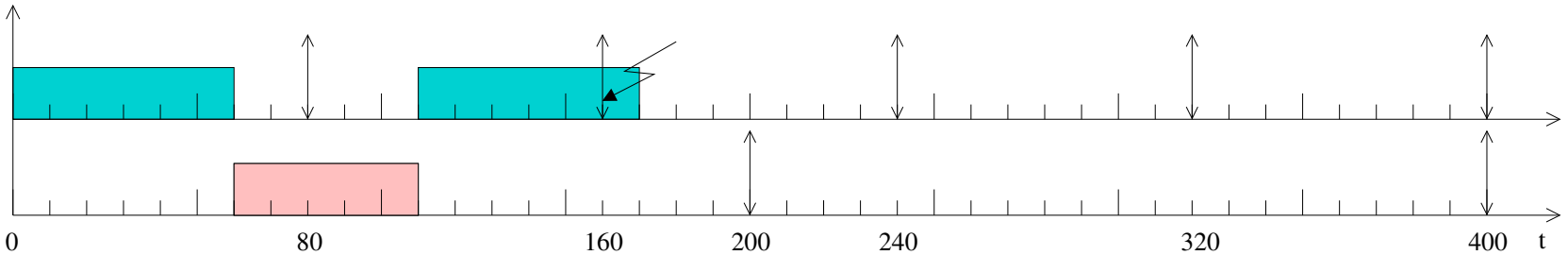
Non-Preemptive model



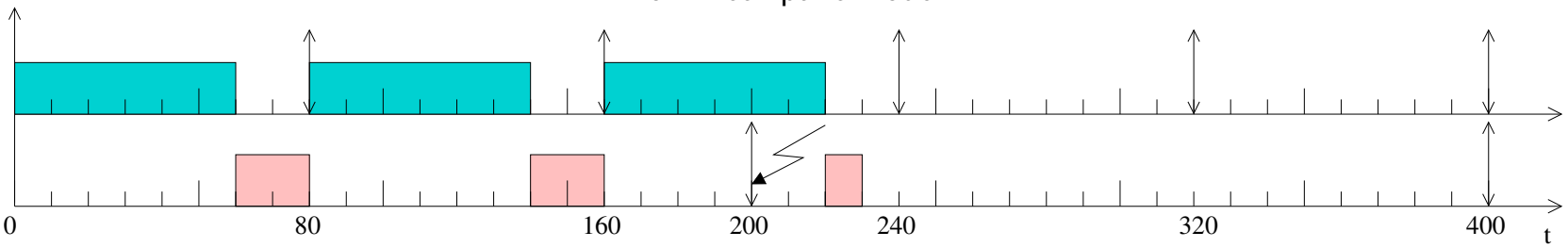
Fully-Preemptive model

Motivational example

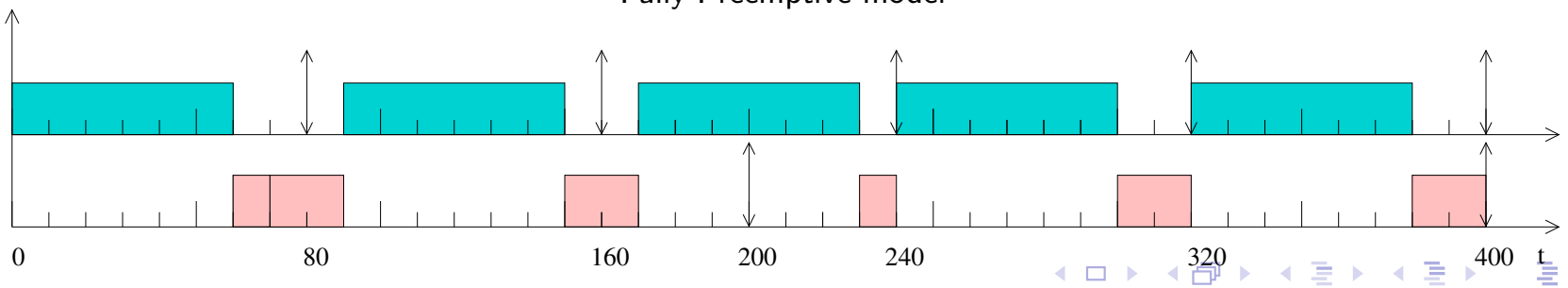
At $s = s_1 = 0.5$ ($\tau_1 : 60/80, \tau_2 : 50/200,\ U = 1$):



Non-Preemptive model



Fully-Preemptive model



Motivational example

So, why is it worthy to use the limited preemptive model?

1. as shown, higher number of feasible task sets
2. preemptions bounded up to non-preemptive-region number
3. longer blocking tolerance
4. simpler management of mutual exclusion

Motivational example

So, why is it worthy to use the limited preemptive model?

1. as shown, higher number of feasible task sets
2. preemptions bounded up to non-preemptive-region number
3. longer blocking tolerance
4. simpler management of mutual exclusion

What can such a model do for reducing the dissipation?

1. task sets are feasible at lower speeds than in the other models
2. lower preemption overhead (not a naive note)
3. longer consecutive intervals in deep sleep

System model

Set of m different speeds ($s = f/f_{max}$), sorted in ascending order

Power consumption: $P(s) = K_3 s^3 + K_2 s^2 + K_1 s + K_0$

The critical speed s^* minimizes the energy for executing a job in the active state

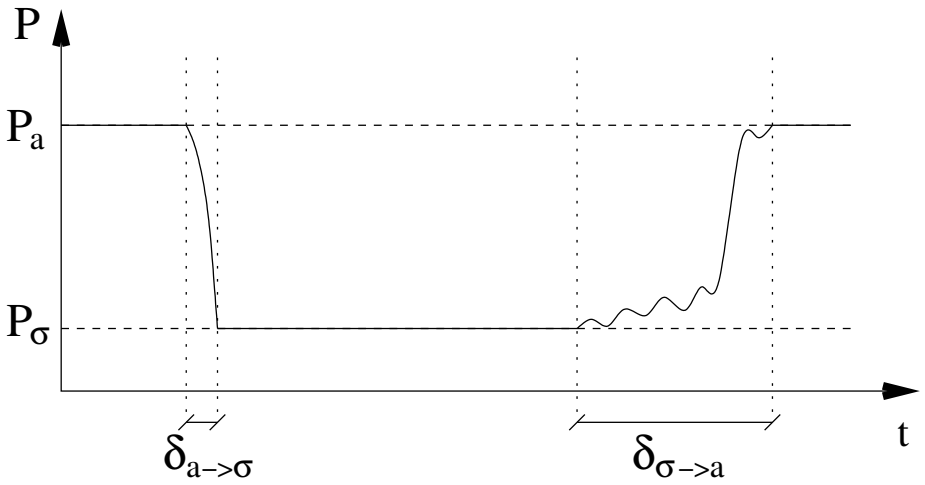
Trade-off between power consumption and execution time:

- higher speeds: shorter execution, high power $\Rightarrow$ higher energy
- lower speeds: longer execution, lower power $\Rightarrow$ higher energy

Power model

Additional energy saving feature: low-power states

- low power consumption
- no execution
- no negligible time overhead to handle a complete transition



Break-even time: $\delta = \delta_{a \to \sigma} + \delta_{\sigma \to a}$

System model

Preemption cost, ξ: constant value due to context switch, pipeline invalidation and cache-related preemption delay

n periodic fixed-priority tasks, $\tau_1, ...\tau_n$ (descending priority order)

Non preemptive WCET: $C_i^{NP}(s) = \alpha C_i^{NP} + (1 - \alpha) C_i^{NP}/s$

Algorithm

The algorithm is divided in two stages:

- offline DVFS step selects the most energy-effective speed
- online DPM step prolongs the time spent in low power state

Algorithm - Offline DVFS step

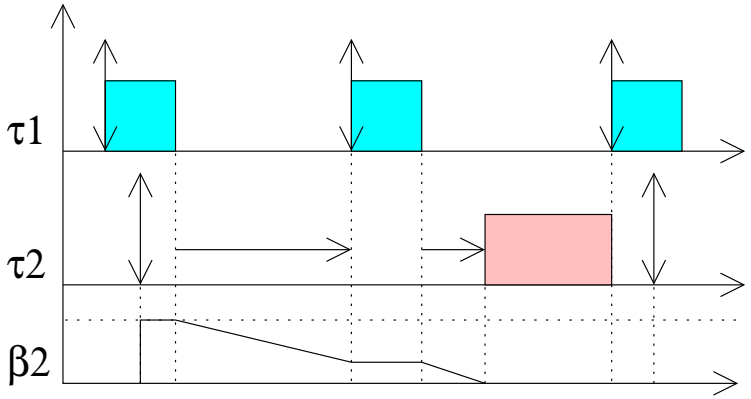
For each task, the PPP algorithm finds the maximum chunk length as a function of higher priority tasks' blocking tolerance

The offline phase finds the minimum s in $[s^*, s_m]$ which guarantees the task set feasibility, also considering the preemption overhead

Complexity: pseudo polynomial

Algorithm - Online DPM step

The PPP algorithm also returns the blocking tolerances, β_i



This step runs at each idle. The CPU is put in sleep (if possible) until first job arrival plus $\beta_{min} = \min_{\tau_i} \beta_i$, using only a timer

The algorithm accounts for both static and dynamic slacks

Complexity: O(1)

Algorithm

Consider a systems with four speeds:

- $s_1 = 0.3$, $s_2 = 0.6$, $s_3 = 0.7$ and $s_4 = 1.0$

Two tasks, τ_1 and τ_2:

- $T_1 = 60$, $T_2 = 150$,
- $C_1^{NP}(s_4) = 18$, $C_2^{NP}(s_4) = 42$
- $\alpha_1 = 0.0$, $\alpha_2 = 0.0 \rightarrow C_i^{NP}(s) = C_i^{NP}/s$

Although the analysis uses preemption cost, here it is assumed null

Algorithm

Assuming $P(s) = 0.9s^3 + 0.1$, $s^* = 0.4$

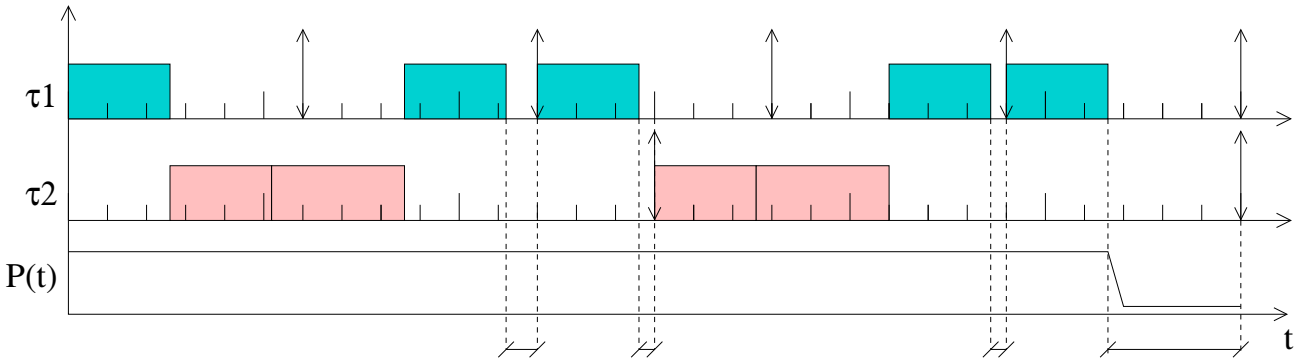
Speed	$C_1^{NP}(s)$	$C_2^{NP}(s)$	β_{min}	$U(s)$	Action
$s_1 = 0.3$	-	-	-	-	Discarded, $s_1 < s^*$
$s_2 = 0.6$	30	70	< 0	$0.9\bar{6}$	Not feasible
$s_3 = 0.7$	26	60	34	$0.8\bar{3}$	Feasible, then exit

τ_1 is executed non-preemptively

τ_2 is divided in two chunks lasting 26 and 34

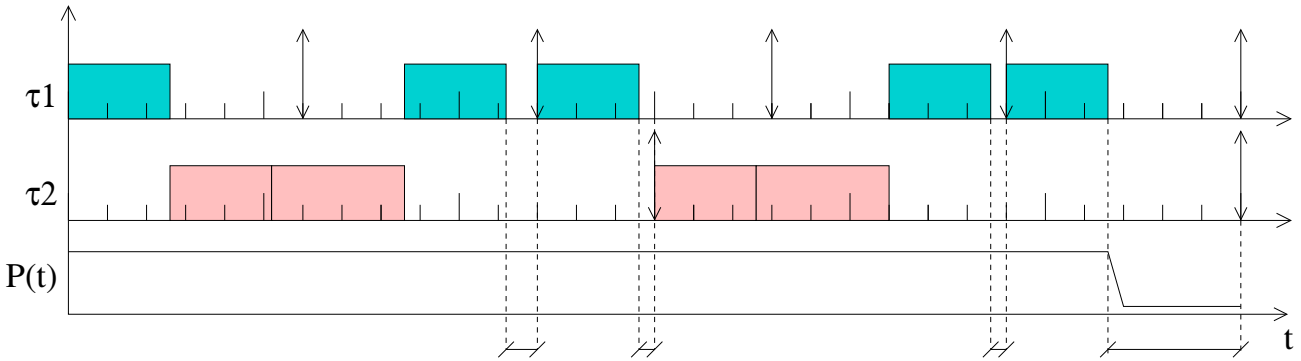
Algorithm

Execution without the online step: fragmented idle times

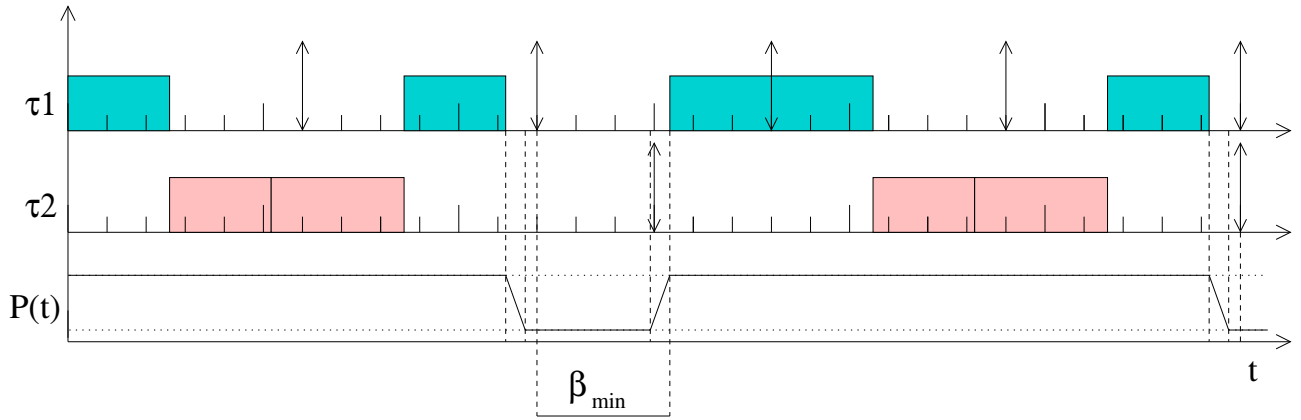


Algorithm

Execution without the online step: fragmented idle times



Execution with the online step



Algorithm

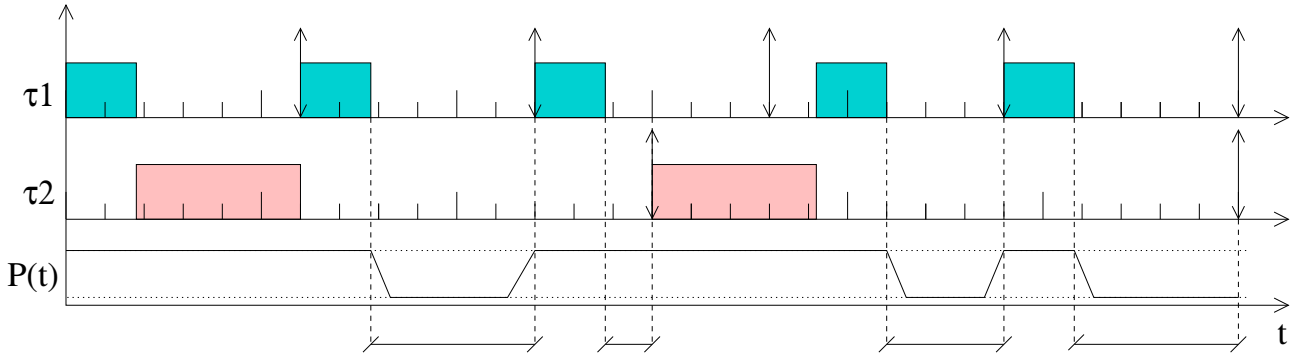
Assuming $P(s) = 0.278s + 0.722$, $s^* = 1.0$

Speed	$C_1^{NP}(s)$	$C_2^{NP}(s)$	β_{min}	$U(s)$	Action
$s_1 = 0.3$	-	-	-	-	Discarded, $s_1 < s^*$
$s_2 = 0.6$	-	-	-	-	Discarded, $s_2 < s^*$
$s_3 = 0.7$	-	-	-	-	Discarded, $s_3 < s^*$
$s_4 = 1.0$	18	42	42	0.58	Feasible, then exit

Both τ_1 and τ_2 execute non-preemptively

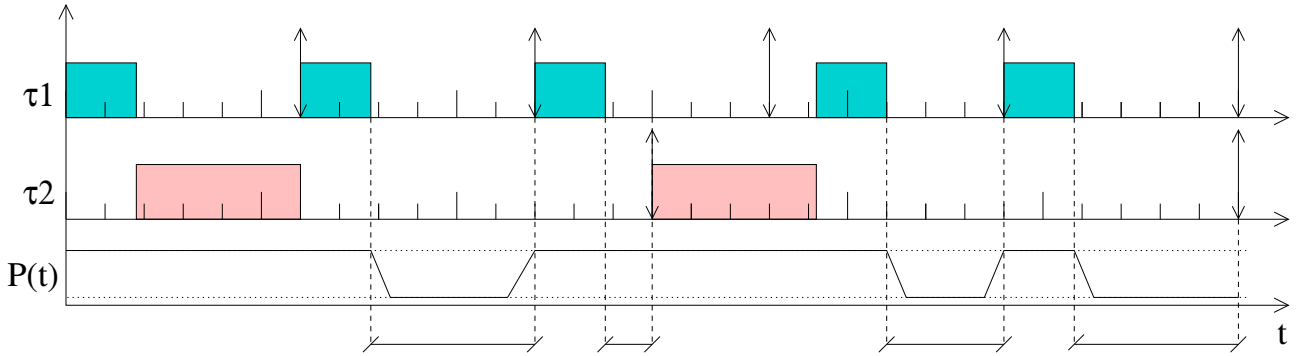
Algorithm

Execution without the online step: fragmented idle times

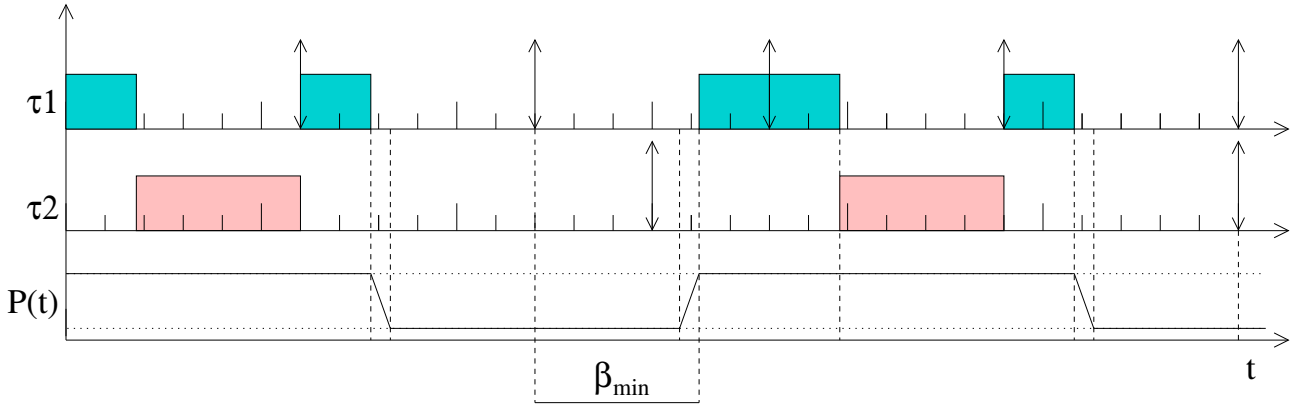


Algorithm

Execution without the online step: fragmented idle times



Execution with the online step



Simulations

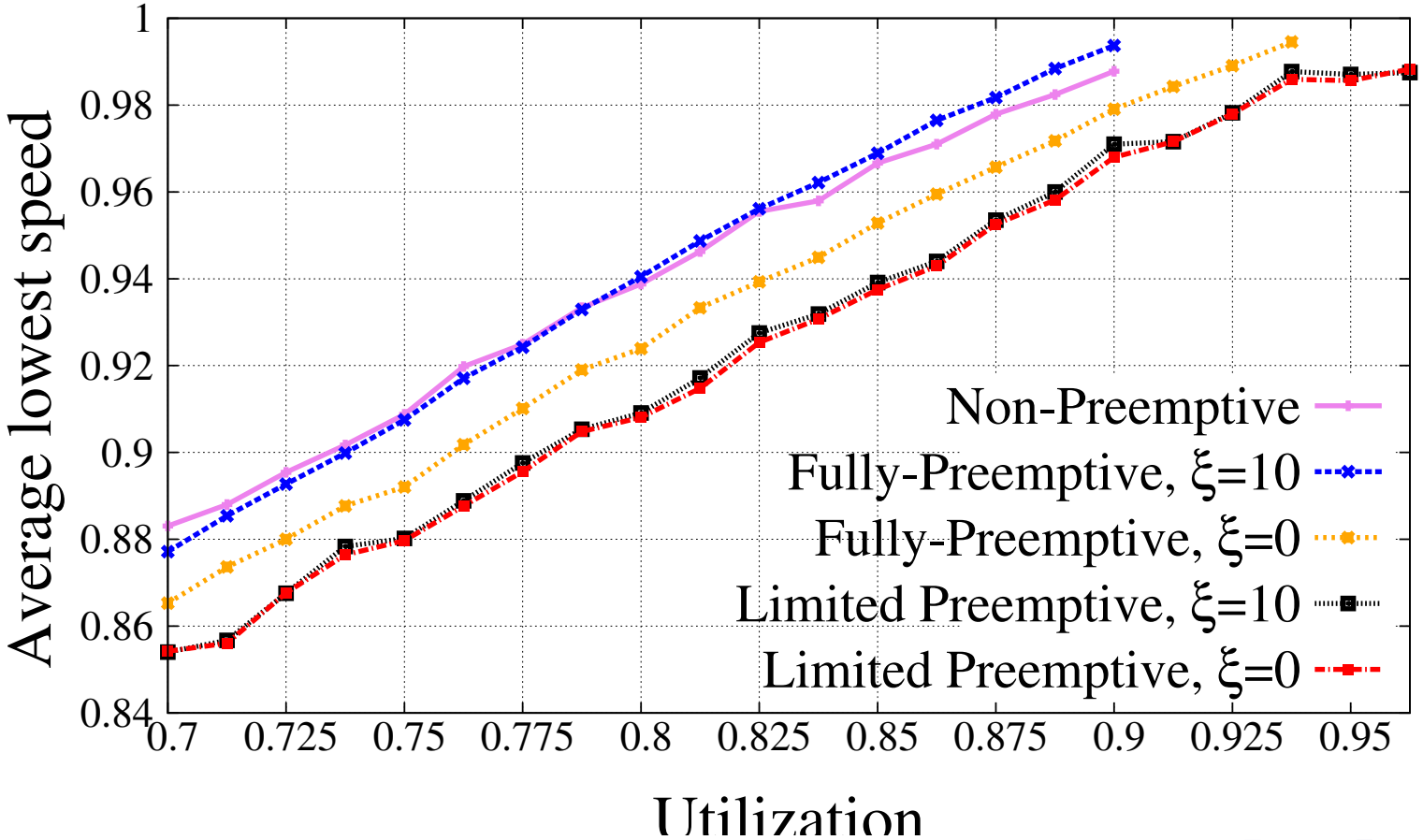
Experimental results were obtained through simulations

Simulation parameters:

- DVFS-sensitive architecture (theoretical model):
  - $P^{(1)}(s) = 0.9s^3 + 0.1$ and $P_\delta^{(1)} = 0.05$
- DPM-sensitive architecture (ARM Cortex M3):
  - $P^{(2)}(s) = 0.278s + 0.722$ and $P_\delta^{(2)} = 0.4$
- 19 discrete speeds in $[0.1, 1.0]$ with step 0.05
- 10 periodic tasks with $C_i^{NP}(1.0) \in [100, 500]$ and $\alpha_i = 0.2$
- task generation algorithm: UUniFast (Bini et al.)

Simulations

Average slowest speed analysis w/o preemption overhead



Simulations

The following tests uses VOSS (Chen and Kuo), the actual state of art for fixed priority systems using Rate Monotonic

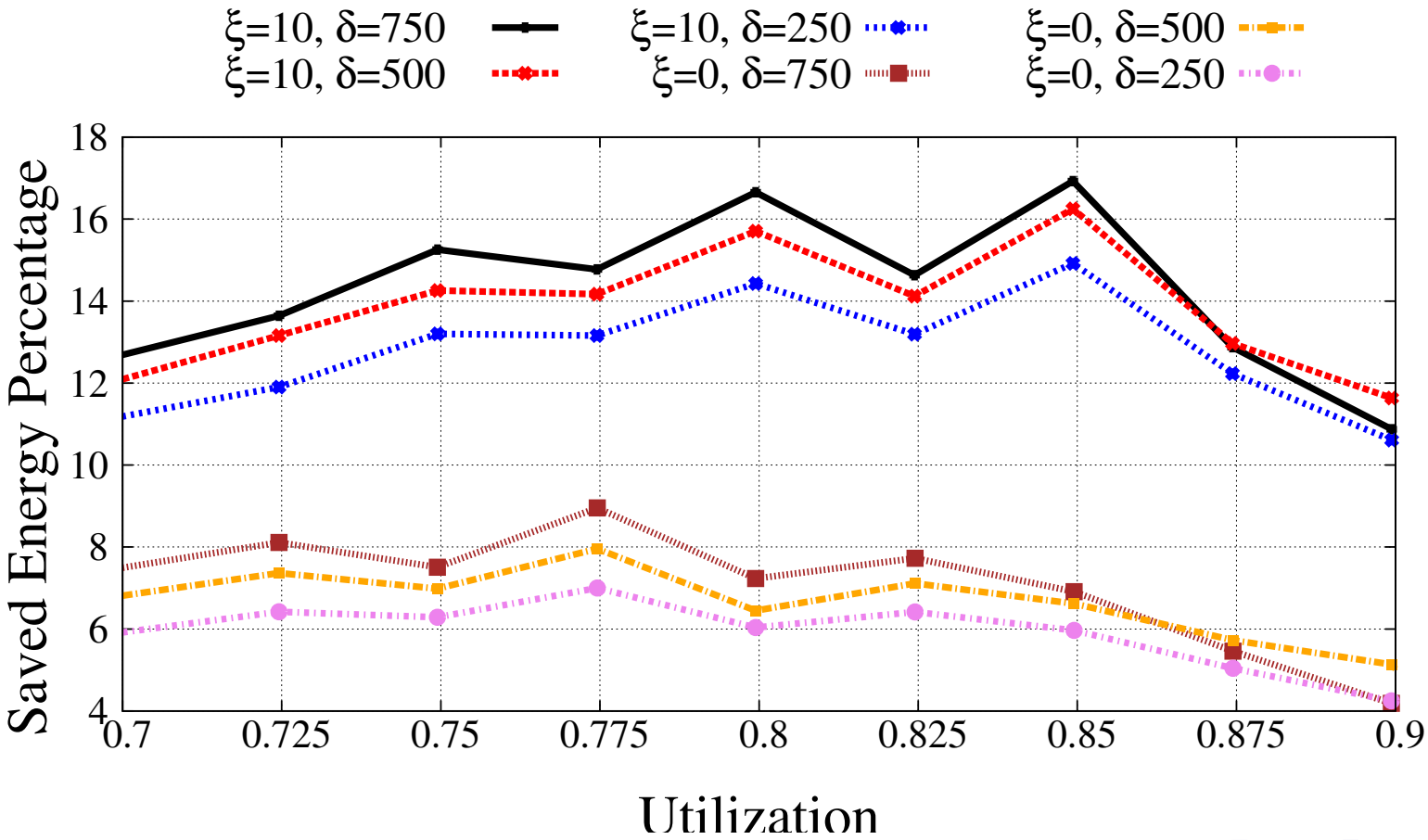
Offline, VOSS computes the slowest possible speed using the more precise RTA (including ξ) instead of Liu and Layland's bound

Online, at every idle, VOSS postpones each task arrival (by the first abs deadline) by its blocking tolerance and the estimated idle time

VOSS complexity: $O(n \cdot \log n)$ at each idle time

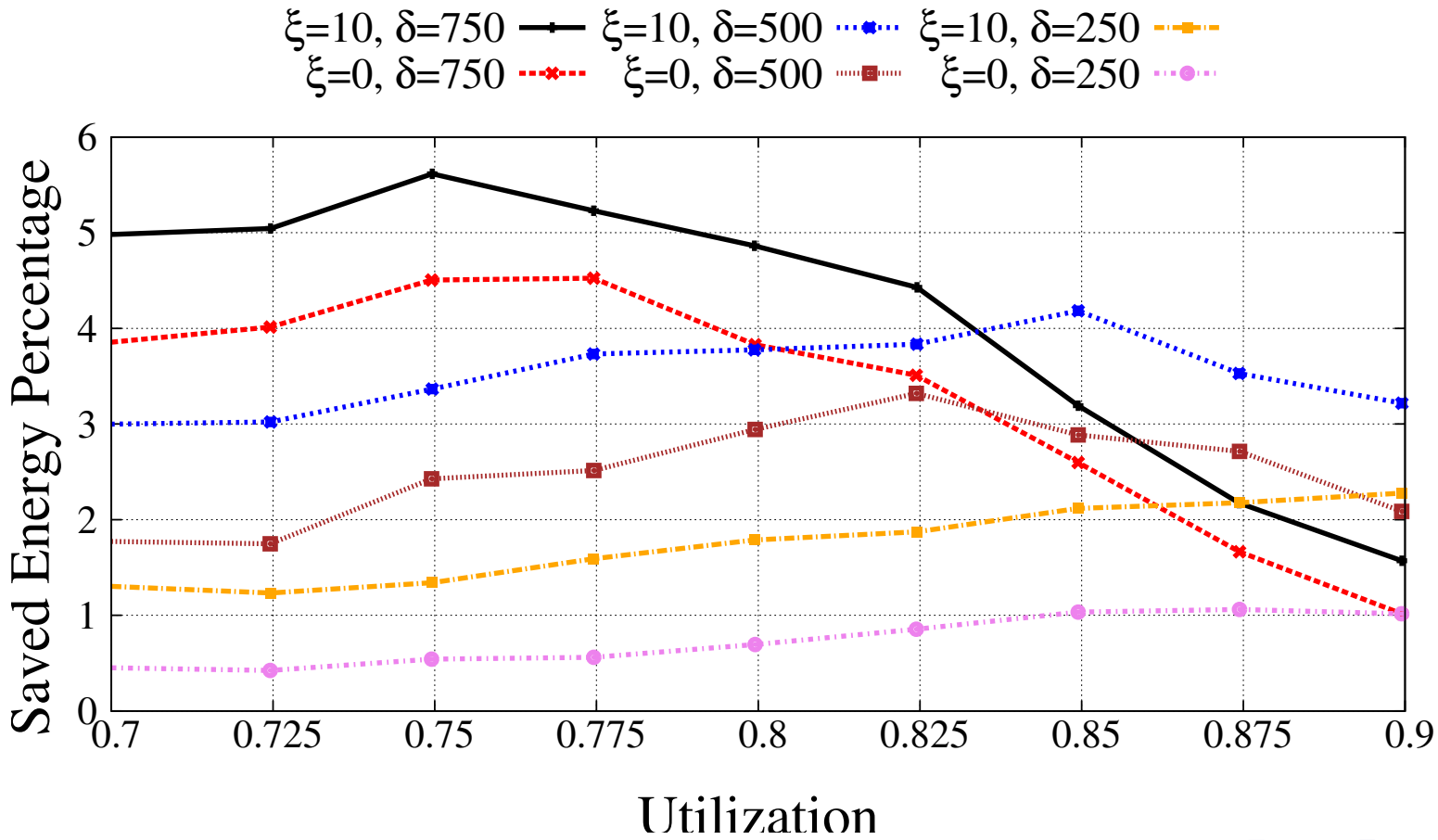
Simulations

Comparisons with VOSS on DVFS-sensitive architecture



Simulations

Comparisons with VOSS on DPM-sensitive architecture



Presented work

The proposed algorithm integrates DVFS and DPM techniques to further optimize the schedule

It has been the first study which had exploited the limited preemptive task model on fixed priority systems to save energy

It outperforms VOSS with $O(1)$ complexity rather than $O(n \cdot \log n)$

More generally...

During the PhD program, the energy-aware real-time scheduling problem has been considered for:

- single-core embedded systems and multi-core systems
- distributed embedded systems
- embedded systems with harvesting devices

We aimed at providing solutions with low complexity while considering the state-of-art hardware

For future generations: BIG.little cores and high-integrated chips with shared memory

thank you

m.bambagini@sssup.it

Bibliography

14 M. Bambagini and H. Aydin, "Periodic Charging Scheme for Fixed-Priority Real-Time Systems with Renewable Energy";
13 M. Bambagini, M. Bertogna and G. Buttazzo, "On the Effectiveness of Energy-Aware Real-Time Scheduling Algorithms on Single-Core Platforms";
12 M. Bambagini, J. Lelli, G. Buttazzo and G. Lipari, "On the Energy-Aware Partitioning of Real-Time Tasks on Homogeneous Multi-Processor Systems";
11 M. Bambagini, M. Bertogna, M. Marinoni and G. Buttazzo, "An Energy-Aware Algorithm Exploiting Limited Preemptive Scheduling under Fixed Priorities";
10 M. Bambagini, G. Buttazzo, and M. Bertogna, "Energy-Aware Scheduling for Tasks with Mixed Energy Requirements";
9 M. Bambagini and M. Di Natale, "Code Generation Framework for Distributed Real-Time Embedded Systems";
8 M. Bambagini, M. Bertogna, M. Marinoni and G. Buttazzo, "On the Impact of Runtime Overhead on Energy-Aware Scheduling"
7 F. Prosperi, M. Bambagini, G. Buttazzo, M. Marinoni and G. Franchino, "An Energy-Aware Algorithm for Tasks and Bandwidth Co-Allocation under Real-Time Constraints";
6 M. Bambagini, G. Buttazzo, and M. Bertogna, "Energy Saving Exploiting the Limited Preemption Task Model";
5 M. Bambagini, G. Buttazzo, and S. Hendseth, "Exploiting Uni-Processor Schedulabilty Analysis for Partitioned Task Allocation on Multi-Processors with Precedence Constraints";
4 M. Di Natale, M. Bambagini, M. Morelli, A. Passaro, D. Di Stefano and G. Arturi, "Enabling Model-Based Development of Distributed Embedded Systems on Open Source and Free Tools";
3 J. Lelli, M. Bambagini and G. Lipari, "Deadline scheduling: can your mobile device last longer?";
2 M. Bambagini, F. Prosperi, M. Marinoni and G. Buttazzo, "Energy Management for Tiny Real-Time Kernels";
1 M. Marinoni, M. Bambagini, F. Prosperi, F. Esposito, G. Franchino, L. Santinelli and G. Buttazzo, "Platform-aware Bandwidth-oriented Energy Management Algorithm for Real-Time Embedded Systems";