# Component-based Analysis of Real-Time Systems

Giuseppe Lipari

Scuola Superiore Sant’Anna – Pisa, Italy and
LSV, ENS Cachan, France

Séminaire MeFoSyLoMa
LIP6

# Outline

# Outline

1. A 10 minutes introduction to Real-Time scheduling
2. Component-based Real-Time Systems
3. Time partitioning
4. Analysis
   - Single processors
   - Distributed systems
   - Multicore
   - Formal methods
5. From theory to practice
6. Conclusions and open problems

# Real-Time Systems

- Most real-time systems are concurrent
  - need to handle many events with different temporal characteristics
- Periodic events
  - In control systems, periodic sampling, computation of the control algorithm, actuation
  - Different events may have different periods
- Aperiodic events
  - May be triggered by the external environment
  - Examples: a sensor triggers an interrupt, a packet arrives from the network
- Different events are handled by different tasks that run concurrently
- Constraints: each task instance must complete before a certain instant (deadline)
- Scheduling problem: how to interleave tasks executions so that each task instance meets its deadline

# Task model

A task can be:

- *periodic*: has a regular structure, consisting of an infinite cycle, in which it executes a computation and then suspends itself waiting for the next periodic activation. An example of pthread library code for a periodic task is the following:

```
void * PeriodicTask(void *arg)
{
    <initialization>;
    <start periodic timer, period = T>;
    while (cond) {
        <read sensors>;
        <update outputs>;
        <update state variables>;
        <wait next activation>;
    }
}
```

# Periodic tasks

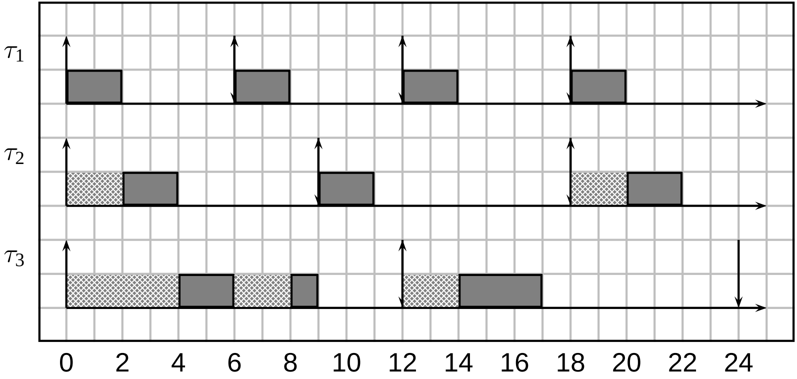
A periodic task $\tau_i = (C_i, D_i, T_i)$ is a infinite sequence of jobs $J_{i,k} = \{a_{i,k}, c_{i,k}, d_{i,k}\}$, where:

$$
\begin{aligned}
a_{i,0} &= 0 \\
a_{i,k} &= a_{i,k-1} + T_i \quad \forall k > 0 \\
d_{i,k} &= a_{i,k} + D_i \quad \forall k \geq 0 \\
C_i &= \max_k \{c_{i,k}\}
\end{aligned}
$$

- $T_i$ is the task's period;
- $D_i$ is the task's relative deadline;
- $C_i$ is the task's worst-case execution time (WCET);
- $R_i$ is the worst-case response time
- for the task to be schedulable, it must be $R_i \leq D_i$.

# Example of schedule

- Fixed priority: the active task with the highest priority is executed on the processor.



# Difference with Classical Scheduling problems

- In classical scheduling problems (i.e. Job-shop)
  - Tasks are one-shot (not periodic)
  - No timing constraints
  - Goal is to minimise completion time (the *make-span* problem), or some *cost function*.
  - Resources can be complex (different machines, precedence constraints, etc.)
  - The general form is often only solvable by Mixed-Integer Linear Programming.
- In real-time scheduling
  - Tasks are periodic or sporadic
  - emphasis on time constraints
  - resources are simple (single processors, uniform multiprocessors)
  - many problems can be solved in polynomial time

# Scheduling, schedulability, feasibility

- Scheduling algorithm
  - An on-line or off-line algorithm $\mathcal{A}$ that, given a task set $\mathcal{T}$ decides which tasks are executed at each instant on each processor (the *schedule* $\sigma(\mathcal{A}, \mathcal{T}, t)$)

# Scheduling, schedulability, feasibility

- Scheduling algorithm
  - An on-line or off-line algorithm $\mathcal{A}$ that, given a task set $\mathcal{T}$ decides which tasks are executed at each instant on each processor (the *schedule* $\sigma(\mathcal{A}, \mathcal{T}, t)$)
- Schedulable task set
  - A task set $\mathcal{T}$ is schedulable by algorithm $\mathcal{A}$ iff all jobs complete before their deadlines in the schedule $\sigma(\mathcal{A}, \mathcal{T}, t)$

# Scheduling, schedulability, feasibility

- Scheduling algorithm
  - An on-line or off-line algorithm $\mathcal{A}$ that, given a task set $\mathcal{T}$ decides which tasks are executed at each instant on each processor (the *schedule* $\sigma(\mathcal{A}, \mathcal{T}, t)$)
- Schedulable task set
  - A task set $\mathcal{T}$ is schedulable by algorithm $\mathcal{A}$ iff all jobs complete before their deadlines in the schedule $\sigma(\mathcal{A}, \mathcal{T}, t)$
- Schedulability test
  - Given a scheduling algorithm $\mathcal{A}$, and a set of tasks $\mathcal{T}$, decide if $\mathcal{A}$ will produce a feasible schedule (i.e. a schedule in which all jobs)

# Scheduling, schedulability, feasibility

- Scheduling algorithm
  - An on-line or off-line algorithm $\mathcal{A}$ that, given a task set $\mathcal{T}$ decides which tasks are executed at each instant on each processor (the *schedule* $\sigma(\mathcal{A}, \mathcal{T}, t)$)
- Schedulable task set
  - A task set $\mathcal{T}$ is schedulable by algorithm $\mathcal{A}$ iff all jobs complete before their deadlines in the schedule $\sigma(\mathcal{A}, \mathcal{T}, t)$
- Schedulability test
  - Given a scheduling algorithm $\mathcal{A}$, and a set of tasks $\mathcal{T}$, decide if $\mathcal{A}$ will produce a feasible schedule (i.e. a schedule in which all jobs)
- Feasibility problem
  - Given a set of tasks $\mathcal{T}$, decide if it exists a scheduling algorithm $\mathcal{A}$ that produces a feasible schedule on $\mathcal{T}$.

# Schedulability test

- One key objective of *real-time analysis* is to be able to know in advance if the task set is schedulable by a certain scheduling algorithm
- Generate and check the schedule (hint: it is a periodic function)
  - Pro: in this case, feasibility can be reduced to a classical MILP problem
  - Cons: NP-Hard

# Schedulability test

- One key objective of *real-time analysis* is to be able to know in advance if the task set is schedulable by a certain scheduling algorithm
- Generate and check the schedule (hint: it is a periodic function)
  - Pro: in this case, feasibility can be reduced to a classical MILP problem
  - Cons: NP-Hard
- Worst-case approach: try to identify *worst-case scenario*
  - Pro: feasibility in polynomial (or pseudo-polynomial) complexity
  - Cons: not quite easy to identify the worst-case
  - Cons: often, only sufficient conditions

# Schedulability tests

## Theorem (Liu and Layland, 1973)

*Consider $n$ periodic (or sporadic) tasks with relative deadline equal to periods, whose priorities are assigned in Rate Monotonic order. Then,*

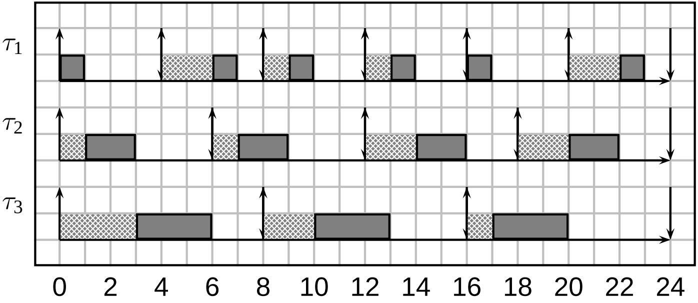
$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq U_{lub} = n(2^{1/n} - 1)$$

- $U_{lub}$ is a decreasing function of $n$;
- For large $n$: $U_{lub} \to 0.69$

| **n** | **U**$_{lub}$ | **n** | **U**$_{lub}$ |
|---|---|---|---|
| 2 | 0.828 | 7 | 0.728 |
| 3 | 0.779 | 8 | 0.724 |
| 4 | 0.756 | 9 | 0.720 |
| 5 | 0.743 | 10 | 0.717 |
| 6 | 0.734 | 11 | ... |

# Dynamic priority scheduling

- The most important dynamic priority algorithm is Earliest Deadline First (EDF)
  - The priority of a job (instance) is inversely proportional to its absolute deadline;
- Example with $U = \frac{23}{24}$



# Schedulability tests

**Theorem (Optimality, Dertouzos '73)**

*If a set of jobs $\mathcal{J}$ is schedulable by an algorithm $\mathcal{A}$, then it is schedulable by EDF.*

**Theorem (Liu & Layland '71)**

*Given a task set of periodic or sporadic tasks, with relative deadlines equal to periods, the task set is schedulable by EDF if and only if*

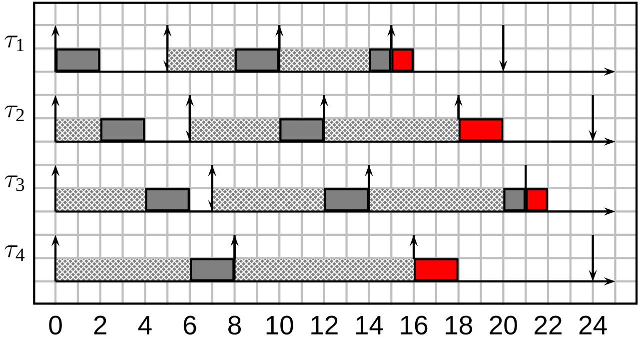
$$U = \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$$

# A key problem

- Scheduling experts start from a task model where the computation times of the tasks are given
- However, estimating WCET can be extremely difficult
  - Compute all possible paths in the code (not so difficult)
  - Under all possible values of input vectors (much more difficult), and state variables (very difficult!)
  - For each path, take the assembly code and compute number of cycles
- Last step requires a precise model of the hardware platform
  - A model of the hardware instruction pipeline
  - A model of the cache memory
  - a model of other unpredictabilities (like out-of-order execution)
- If it is not done right, large overestimation of WCET, or (even worse!) underestimation

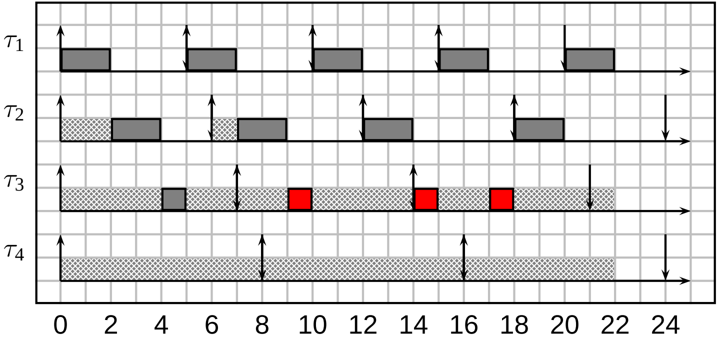
# Domino effect

- In case of overhead ($U > 1$), in EDF we have the *domino effect*: it means that all tasks miss their deadlines.
- An example of domino effect is the following;



# Domino effect: considerations

- FP is more predictable: only lower priority tasks miss their deadlines! In the previous example, if we use FP:




- However, it may happen that some task never executes in case of high overload
- EDF is more *fair* (all tasks are treated in the same way).

# Current research trends

- Many different task models have been proposed
  - With precedence constraints, varying computation time, probabilistic, shared resources, soft real-time, etc.
- After many years, single processor problem is (*almost*) a closed area of investigation
- Multi-processor scheduling: one or two orders of magnitude more difficult problem, still open
- Distributed system: general problem still very difficult, but lot of research has been done

In this talk:

- Component-based analysis of Real-Time Systems

# Outline

# Modern Real-Time Systems

Modern real-time applications can be very complex

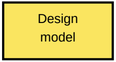
- Automotive software (high-end car model)
  - Millions of lines of (low level) code
  - up to 80 distributed nodes
  - up to 5 different networks
- At the same time they are safety critical
  - A single bug may compromise human life

Problems:

- How to analyse, certify and validate the code?
- How to manage complexity?

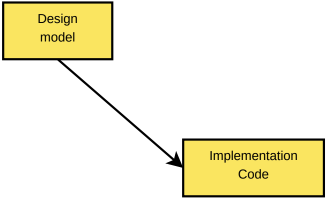
# Off-line and on-line

- Off-line:



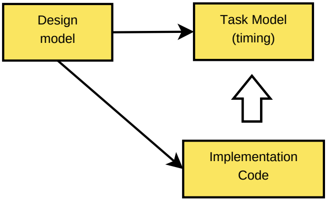
# Off-line and on-line

- Off-line:
  - Write code,



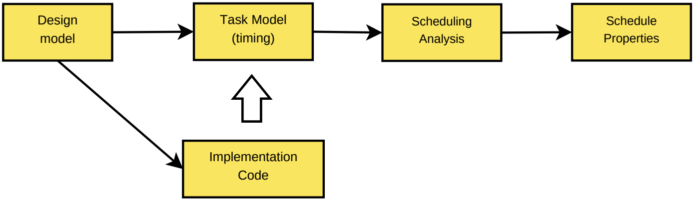
# Off-line and on-line

- Off-line:
    - Write code,
    - estimate WCET,



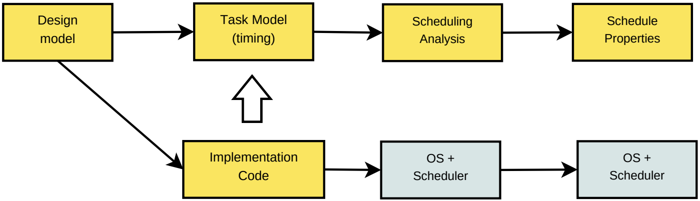
# Off-line and on-line

- Off-line:
  - Write code,
  - estimate WCET,
  - perform analysis



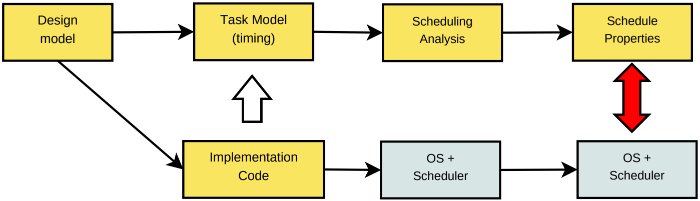
# Off-line and on-line

- Off-line:
  - Write code,
  - estimate WCET,
  - perform analysis
- On-line:
  - Execute task on the OS (by the scheduler)



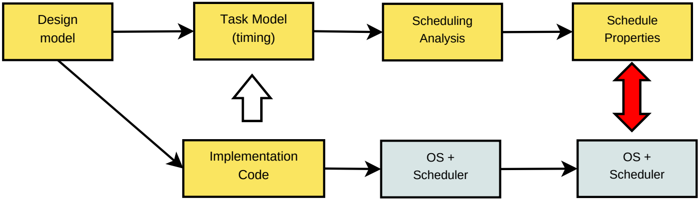
# Off-line and on-line

- Off-line:
  - Write code,
  - estimate WCET,
  - perform analysis
- On-line:
  - Execute task on the OS (by the scheduler)
- If some WCET is underestimated, anything can happen



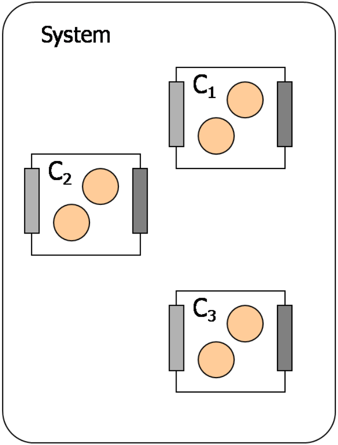
# Off-line and on-line

- Off-line:
  - Write code,
  - estimate WCET,
  - perform analysis
- On-line:
  - Execute task on the OS (by the scheduler)
- If some WCET is underestimated, anything can happen
- The more complex is the system, the more difficult is to keep analysis and execution in sync



# Component-based design

- Design the overall architecture
  - as a set of smaller interacting components
- Component design and implementation
  - in modern applications, some component is implemented by third parties
  - some component could be reused from previous projects
- When components are completed, do integration and analysis



# Advantages

Simplify the design of complex distributed systems

- system as hierarchy of components

Independent design and implementation of sub-systems

- separation between interface and implementation

Re-use of existing and well-tested components

- to reduce development cost

Dynamic and on-line (re-)configuration

- substitute or upgrade a component, possibly on-line

# Component-based analysis

- A component-based methodology should include a component-based analysis

# Component-based analysis

- A component-based methodology should include a component-based analysis
- Analysis is first done at each component level
  - This is the “local” analysis
  - The result is a (functional and non-functional) characterisation of the properties of the component
  - For example: resource requirements of the component over time

# Component-based analysis

- A component-based methodology should include a component-based analysis
- Analysis is first done at each component level
  - This is the “local” analysis
  - The result is a (functional and non-functional) characterisation of the properties of the component
  - For example: resource requirements of the component over time
- Then, at the global level
  - Component are integrated in the final system

# Component-based analysis

- A component-based methodology should include a component-based analysis
- Analysis is first done at each component level
  - This is the “local” analysis
  - The result is a (functional and non-functional) characterisation of the properties of the component
  - For example: resource requirements of the component over time
- Then, at the global level
  - Component are integrated in the final system
  - Each component is represents by its interface, including functional and non-functional properties (e.g., resource requirements)
  - Therefore, in global analysis we can ignore the internal details of all components

# Component-based analysis

- A component-based methodology should include a component-based analysis
- Analysis is first done at each component level
  - This is the “local” analysis
  - The result is a (functional and non-functional) characterisation of the properties of the component
  - For example: resource requirements of the component over time
- Then, at the global level
  - Component are integrated in the final system
  - Each component is represents by its interface, including functional and non-functional properties (e.g., resource requirements)
  - Therefore, in global analysis we can ignore the internal details of all components

# Component-based analysis

- A component-based methodology should include a component-based analysis
- Analysis is first done at each component level
  - This is the “local” analysis
  - The result is a (functional and non-functional) characterisation of the properties of the component
  - For example: resource requirements of the component over time
- Then, at the global level
  - Component are integrated in the final system
  - Each component is represents by its interface, including functional and non-functional properties (e.g., resource requirements)
  - Therefore, in global analysis we can ignore the internal details of all components
- Pro: simplification

# Component-based analysis

- A component-based methodology should include a component-based analysis
- Analysis is first done at each component level
  - This is the “local” analysis
  - The result is a (functional and non-functional) characterisation of the properties of the component
  - For example: resource requirements of the component over time
- Then, at the global level
  - Component are integrated in the final system
  - Each component is represents by its interface, including functional and non-functional properties (e.g., resource requirements)
  - Therefore, in global analysis we can ignore the internal details of all components
- Pro: simplification
- Cons: we lose optimality, we may waste resources

# Run-time support

- Analysis is necessary, but not sufficient to implement a component-based system
- We also need Run-Time Support
  - The concept of component should be supported by at the Operating Systems (or at the Middle-ware) level
  - Component must be “isolated” from each other to avoid cross-talk effects not caught at analysis time
- OS should enforce isolation
  - Memory isolation (to avoid memory corruption by a bugged component)
  - Temporal isolation (to avoid that a component uses more resources than expected)

# Outline

# Traditional approach to scheduling fails

Summary of objectives

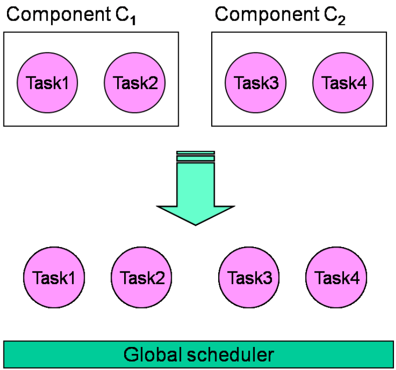
- Objective 1: independent component analysis
- Objective 2: system analysis using (light) abstractions

Now we see why it is impossible to achieve these objectives with a single flat scheduler

- (hint: complexity is high)

# Example

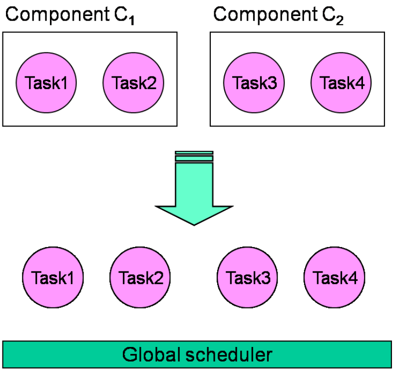
- Designer can assign *local priorities* (no global knowledge)
- At integration phase need to assign priorities relative to each other
- Example: two components, two tasks each




- $p_1 > p_2$ and $p_3 > p_4$

# Example

- Designer can assign *local priorities* (no global knowledge)
- At integration phase need to assign priorities relative to each other
- Example: two components, two tasks each




- $p_1 > p_2$ and $p_3 > p_4$
- Possible priority ordering:
  1. $p_1 > p_2 > p_3 > p_4$
  2. $p_1 > p_3 > p_2 > p_4$
  3. $p_1 > p_3 > p_4 > p_2$
  4. $p_3 > p_4 > p_1 > p_2$
  5. $p_3 > p_1 > p_4 > p_2$
  6. $p_3 > p_1 > p_2 > p_4$

# Temporal isolation

- Higher priority does not always mean higher importance
  - Priority is a scheduling artifact
  - For example, it could be used to maximise the probability of being schedulable
- Without "temporal isolation",
  - A task that executes more than expected may cause a deadline miss to lower priority tasks (that may belong to other components)
- In a flat system, everything interacts with everything else

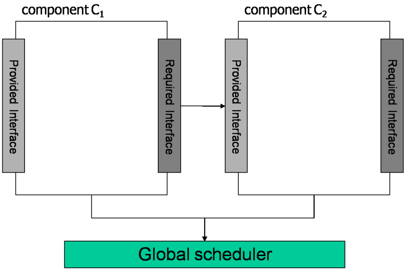
# Two-levels of scheduling

Summary of objectives

- Objective 1: independent component analysis
- Objective 2: system analysis using (light) abstractions

Our solution: two levels of scheduling

- A *global scheduler* selects the components to execute, regardless of their internal structure



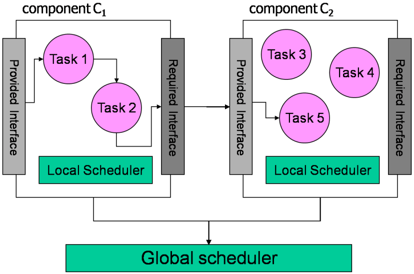
# Two-levels of scheduling

Summary of objectives

- Objective 1: independent component analysis
- Objective 2: system analysis using (light) abstractions

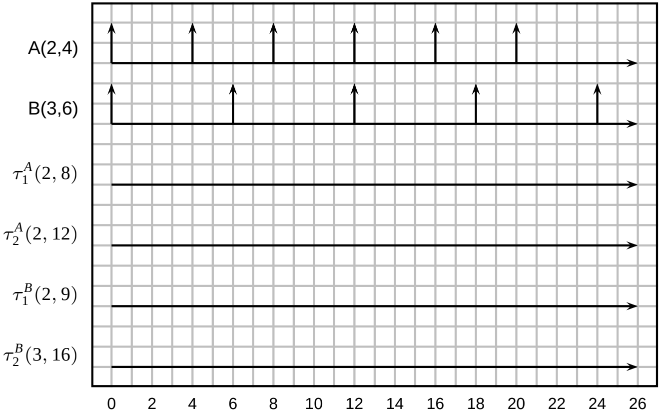
Our solution: two levels of scheduling

- A *global scheduler* selects the components to execute, regardless of their internal structure
- When a component is selected by the global scheduler, a *local scheduler* decides which of the tasks is executing



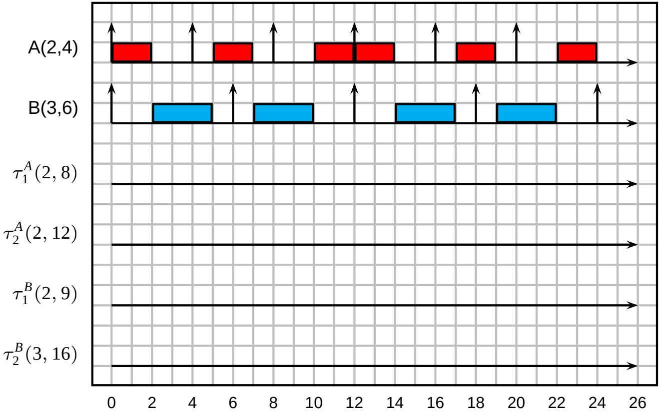
# Global and local scheduler

- The global scheduler *partitions* the resource and allocates it to the components
- the local scheduler assign the resource to the component threads



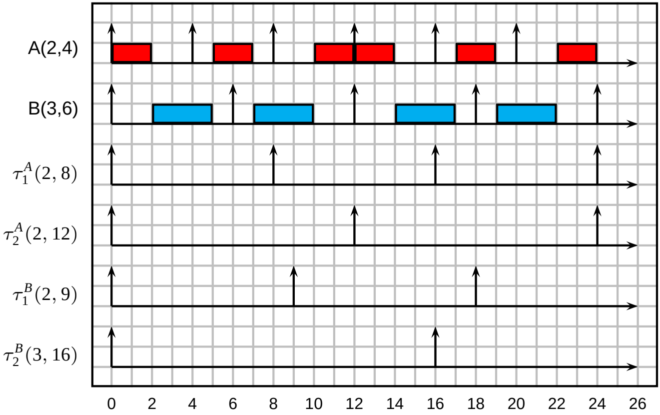
# Global and local scheduler

- The global scheduler *partitions* the resource and allocates it to the components
- the local scheduler assign the resource to the component threads



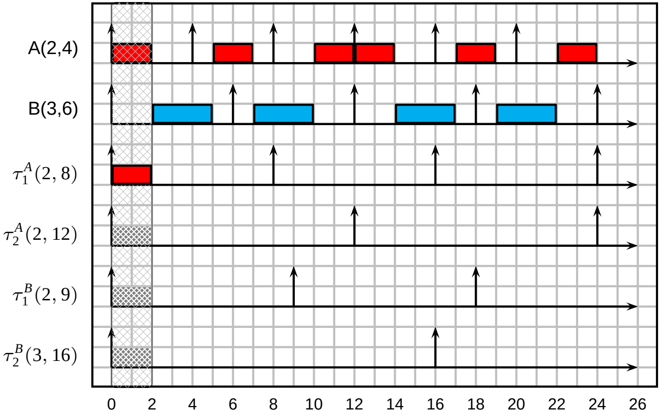
# Global and local scheduler

- The global scheduler *partitions* the resource and allocates it to the components
- the local scheduler assign the resource to the component threads



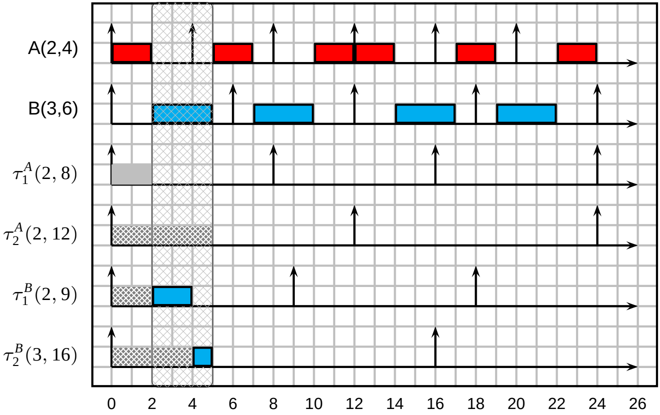
# Global and local scheduler

- The global scheduler *partitions* the resource and allocates it to the components
- the local scheduler assign the resource to the component threads



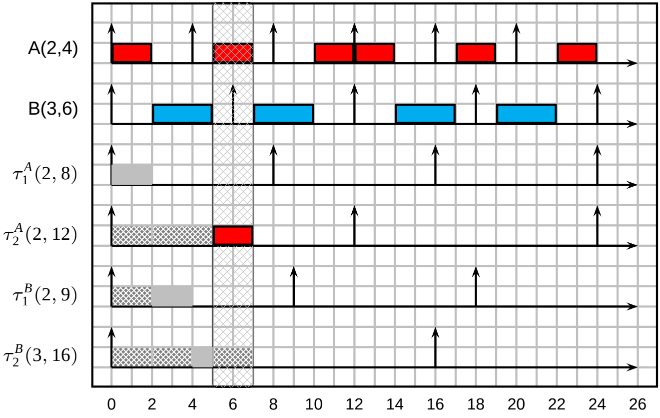
# Global and local scheduler

- The global scheduler *partitions* the resource and allocates it to the components
- the local scheduler assign the resource to the component threads



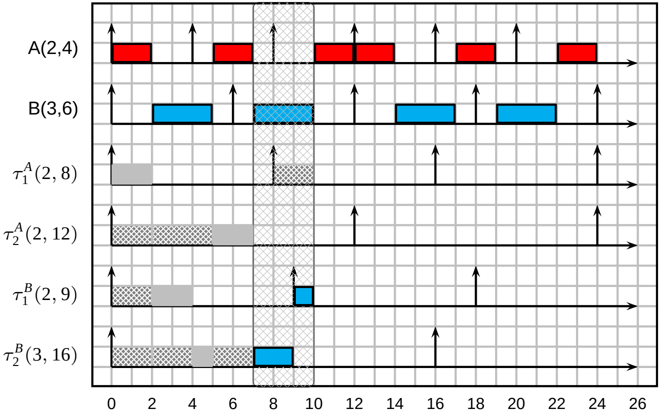
# Global and local scheduler

- The global scheduler *partitions* the resource and allocates it to the components
- the local scheduler assign the resource to the component threads



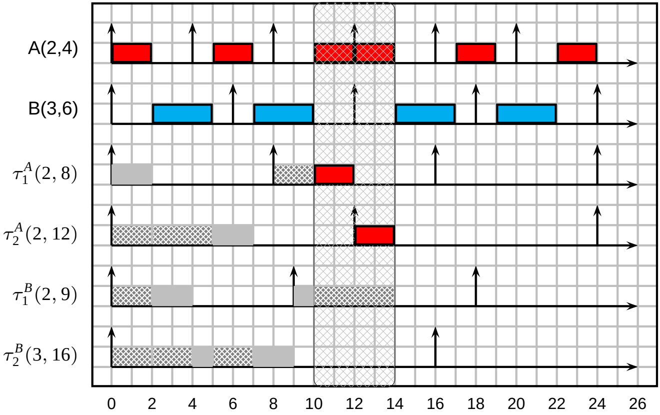
# Global and local scheduler

- The global scheduler *partitions* the resource and allocates it to the components
- the local scheduler assign the resource to the component threads



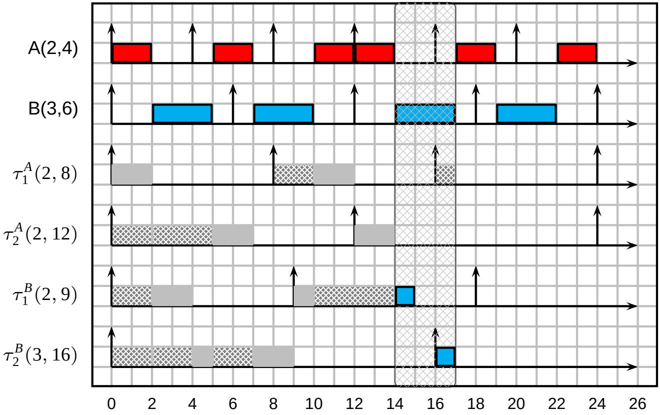
# Global and local scheduler

- The global scheduler *partitions* the resource and allocates it to the components
- the local scheduler assign the resource to the component threads



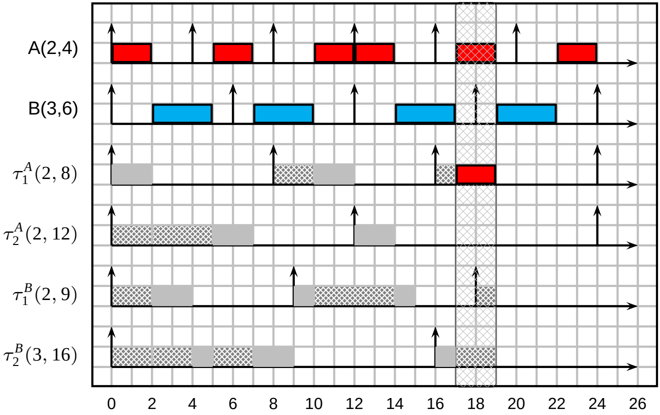
# Global and local scheduler

- The global scheduler *partitions* the resource and allocates it to the components
- the local scheduler assign the resource to the component threads



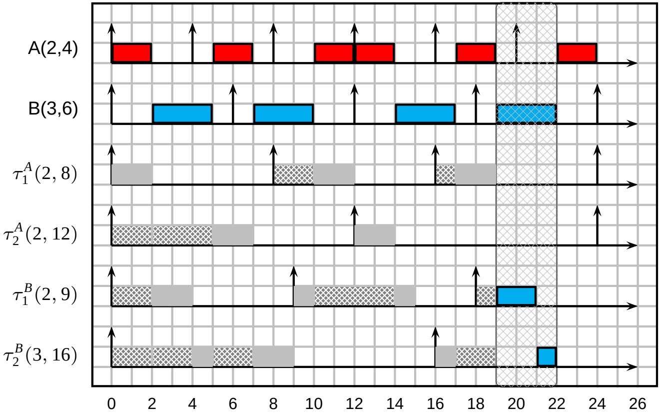
# Global and local scheduler

- The global scheduler *partitions* the resource and allocates it to the components
- the local scheduler assign the resource to the component threads



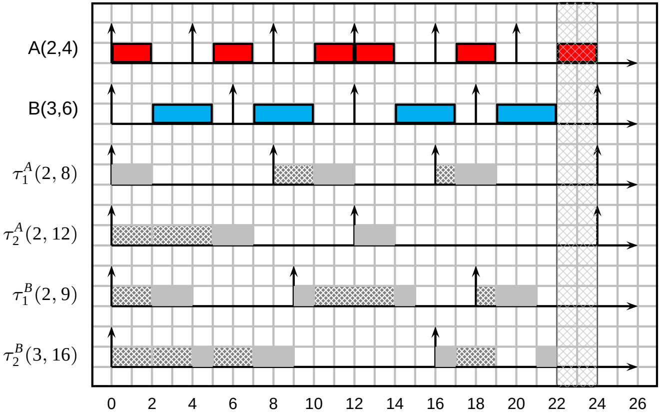
# Global and local scheduler

- The global scheduler *partitions* the resource and allocates it to the components
- the local scheduler assign the resource to the component threads



# Global and local scheduler

- The global scheduler *partitions* the resource and allocates it to the components
- the local scheduler assign the resource to the component threads



# Outline

# Outline

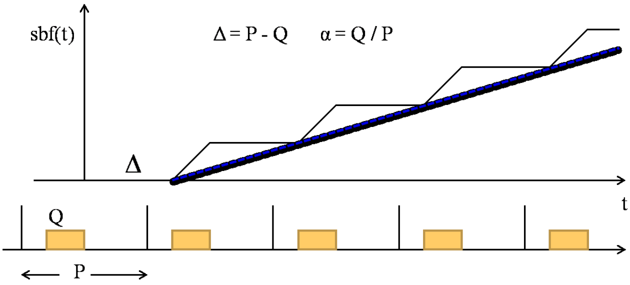
# Chronology of papers

- Problem: given a component (set of periodic threads and a local scheduler) on a time partition, how to test its schedulability?
  - Deng and Liu, (1997) [DL97]
  - The BSS algorithm, by Lipari, Buttazzo, Baruah, Carpenter (1998–2000) [LB00, LCB00, LBA98]
  - Time partitions, Feng and Mok, (2001 – 2002) [MF01, FM02]
  - Temporal interfaces, Shin and Lee, (2003) [SL03]
- Inverse problem: find a partition that makes the component schedulable
  - Lipari and Bini (2003, 2005), [LB03, LB05]
  - Almeida and Pedreiras (2004) [AP04]

# Supply-bound function

The supply bound function (sbf$(t)$)

- it is the minimum amount of resource that the global scheduler provides to one component in an interval of length $t$
- It depends on how the resource is partitioned by the global scheduler




- $\Delta$ is the maximum delay (interval with no resource)
- $\alpha$ is the provided bandwidth

# Static and dynamic partitions

Static partitions:

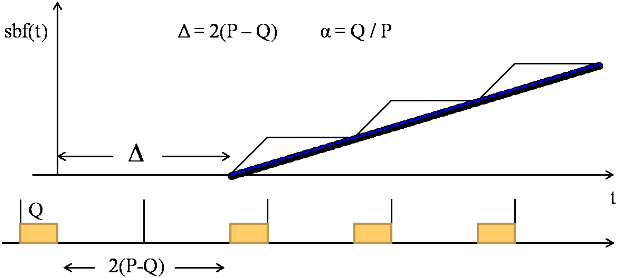
- The global scheduler uses TDM
- Advantages: reduces delay, improves determinism
- Disadvantages: rigid and unflexible

Dynamic Partitions:

- The global scheduler uses a Resource Reservation Algorithm (e.g. CBS, or similar)
- Advantages: can reclaim unused bandwidth, can adapt dynamically, useful for open systems
- Disadvantages: may have a larger delay

# Dynamic partitions

- The sbf$(t)$ for CBS is as follows:



# Schedulability conditions

- After characterising the sbf$(t)$, it is possible to test schedulability using the following properties
- Fixed Priority Local scheduler:
  - Lehoczky test: for every task, it must exist a point $t$ where the required computation time does not exceed $t$:

$$\exists t \in \mathcal{P}_i \quad \sum_{j=1}^{i} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t$$

# Schedulability conditions

- After characterising the sbf$(t)$, it is possible to test schedulability using the following properties
- Fixed Priority Local scheduler:
  - Lehoczky test: for every task, it must exist a point $t$ where the required computation time does not exceed $t$:

$$\exists t \in \mathcal{P}_i \quad \sum_{j=1}^{i} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t$$

  - Lehoczky test for partitions: for every task, it must exist a point $t$ where the required computation time does not exceed sbf$(t)$

$$\exists t \in \mathcal{P}_i \quad \sum_{j=1}^{i} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq \text{sbf}(t)$$

- Earliest Deadline First Local scheduler:
  - Demand Bound Function test: For any interval of length $t$ the demand bound function does not exceed $t$:

$$\forall t \leq \text{dline}(\mathcal{T}) \quad \sum_{j=1}^{i} \left( \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i \leq t$$

# Schedulability conditions

- Earliest Deadline First Local scheduler:
  - Demand Bound Function test: For any interval of length $t$ the demand bound function does not exceed $t$:

$$\forall t \leq \text{dline}(\mathcal{T}) \quad \sum_{j=1}^{i} \left( \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i \leq t$$

  - DBF test partitions: For interval of length $t$ the demand bound function does not exceed sbf(t):

$$\forall t \leq \text{dline}(\mathcal{T}) \quad \sum_{j=1}^{i} \left( \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i \leq \text{sbf}(t)$$

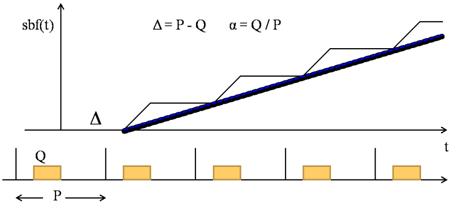
# Time granularity

- Parameter $\Delta$ can have a large impact on the schedulability of the component
  - $\Delta$ represents the maximum period with no resource
  - Clearly, $\Delta$ should be less that the smaller task deadline in the component

# Time granularity

- Parameter $\Delta$ can have a large impact on the schedulability of the component
  - $\Delta$ represents the maximum period with no resource
  - Clearly, $\Delta$ should be less that the smaller task deadline in the component

- However, $\Delta$ is also related to $(P - Q)$
  - A smaller $\Delta$ means more frequent context switches between components, and higher overhead



# Feasibility area

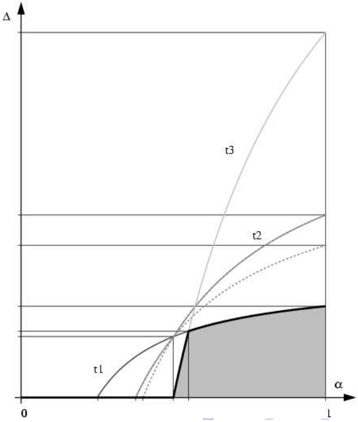
- Reverse problem: given a component (set of periodic tasks), find a partition (sbf$(t)$) such that the component is schedulable
- Lipari and Bini, 2003 and 2005, solved the problem for fixed priority (for EDF is very similar)

- Write Lehoczky's equations with $\alpha, \Delta$ unknowns

# Feasibility area

- Reverse problem: given a component (set of periodic tasks), find a partition (sbf$(t)$) such that the component is schedulable
- Lipari and Bini, 2003 and 2005, solved the problem for fixed priority (for EDF is very similar)

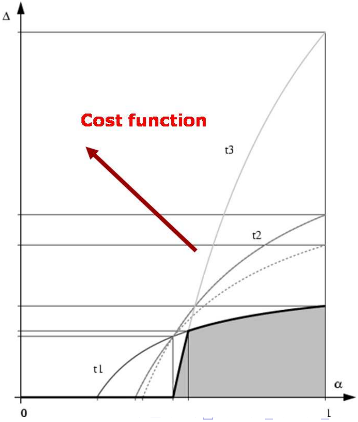
- Write Lehoczky's equations with $\alpha, \Delta$ unknowns
- Find all possible pairs $\alpha, \Delta$ that make the component feasible



# Feasibility area

- Reverse problem: given a component (set of periodic tasks), find a partition (sbf$(t)$) such that the component is schedulable
- Lipari and Bini, 2003 and 2005, solved the problem for fixed priority (for EDF is very similar)

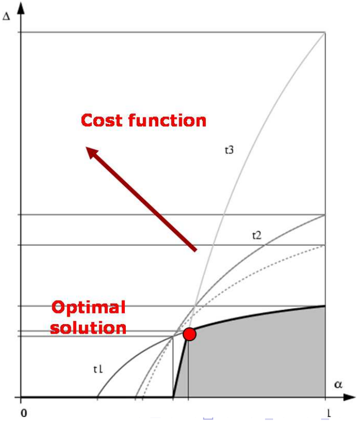
- Write Lehoczky's equations with $\alpha, \Delta$ unknowns
- Find all possible pairs $\alpha, \Delta$ that make the component feasible
- Select a cost function (e.g. minimise overhead)



# Feasibility area

- Reverse problem: given a component (set of periodic tasks), find a partition (sbf($t$)) such that the component is schedulable
- Lipari and Bini, 2003 and 2005, solved the problem for fixed priority (for EDF is very similar)

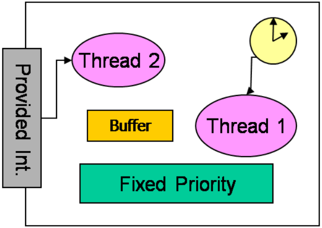
- Write Lehoczky's equations with $\alpha, \Delta$ unknowns
- Find all possible pairs $\alpha, \Delta$ that make the component feasible
- Select a cost function (e.g. minimise overhead)
- Find optimal solution



# Outline

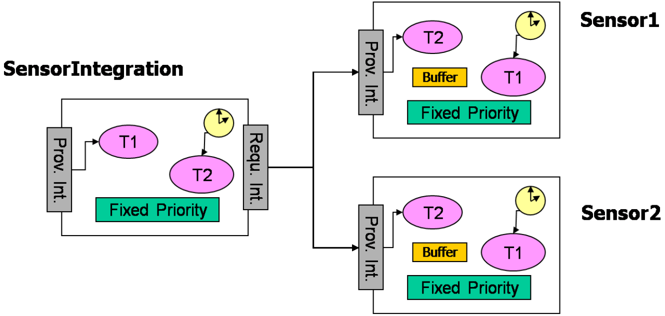
# Message passing

- In the previous model, components may interact through shared memory
  - However, there may be other important isolation requirements (memory protection, fault-confinement) that forbid the use of shared memory in user space
  - Therefore, it is important to also consider message-passing systems
- Let's get back to the definition of component:



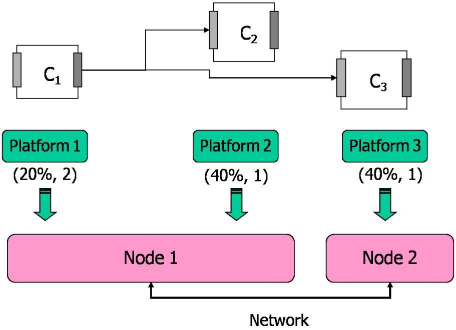

```
SensorReading {
provided:
    double read();
required:
implementation:
    Thread T1 : periodic (15msec),
         priority = 1;
    Thread T2 : implements read(),
         priority = 2;
    Scheduler : FixedPriority;
}
```

# Component interaction




- Component `SensorIntegration` performs a integrations of the two stereoscopic images for reconstructing a 3D model
- Therefore, it uses two instance of component `SensorReadings`, and uses its interface (`read()`), through a Remote Procedure Call (RPC)

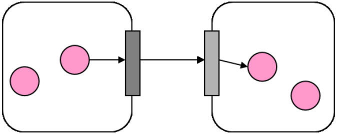
# Mapping




- We prepare three virtual platforms
  - A virtual platform models a temporal partition on one physical processor
- Then, allocate virtual platforms on physical processors

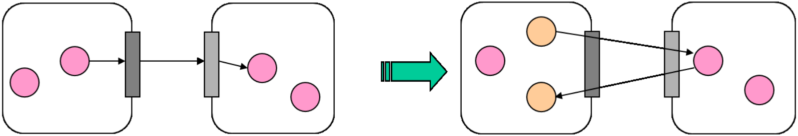
# Analysis

- Lorente, Lipari and Bini (2006), [LLB06]
- Model of the Remote Procedure call
  - We use holistic analysis, therefore, we use the same underlying model
  - A *transaction* is a sequence of *stages*, each stage is part of a task *stages*



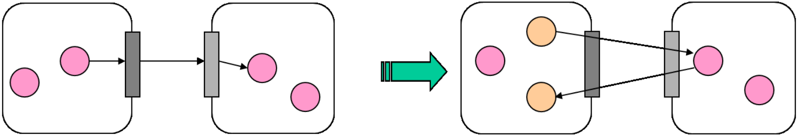
# Analysis

- Lorente, Lipari and Bini (2006), [LLB06]
- Model of the Remote Procedure call
  - We use holistic analysis, therefore, we use the same underlying model
  - A *transaction* is a sequence of *stages*, each stage is part of a task *stages*



# Analysis

- Lorente, Lipari and Bini (2006), [LLB06]
- Model of the Remote Procedure call
  - We use holistic analysis, therefore, we use the same underlying model
  - A *transaction* is a sequence of *stages*, each stage is part of a task *stages*



- Each *virtual platform* is considered as a separate node in a distributed system
  - two components allocated on the same physical node will communicate with very small delay
  - A transaction models the flow of execution through the distributed system

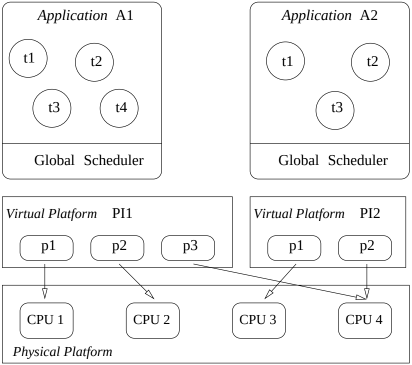
# Analysis

- Holistic Analysis for components
  - Fix platform parameters $(\alpha_i, \Delta_i)$ for every component
  - Perform holistic analysis (fixed priority)
  - As a result, obtain the response times of the tasks
  - If schedulable, then we can stop
  - otherwise, change $(\alpha_i, \Delta_i)$, and start over
- The methodology can be very time-consuming
- Open Problems:
  - how to derive platform parameters?
  - how to change them so to make the system schedulable?

# Outline

# Multicore platforms

- Lipari and Bini (2010) [LB10]
- Each component is scheduled by a Virtual Platform

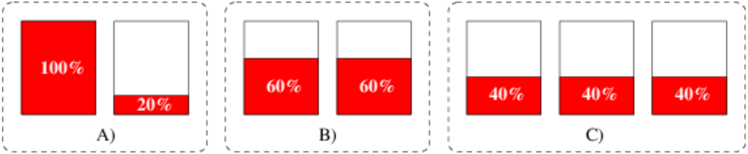



- Virtual platform is modelled by a set of virtual processors $\{\pi_1, \ldots, \pi_m\}$
- Each virtual processor is statically assigned to a physical processor
- More than one virtual processor may be allocated on the same physical processor

# Basic idea

Suppose that we need 120% bandwidth for our component

Various possibilities:

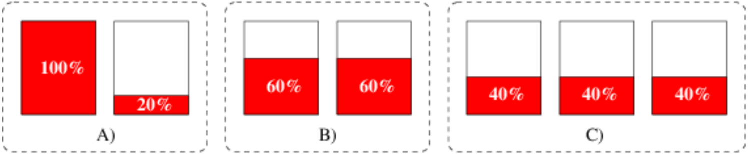



Which one is better?

# Basic idea

Suppose that we need 120% bandwidth for our component

Various possibilities:




Which one is better?

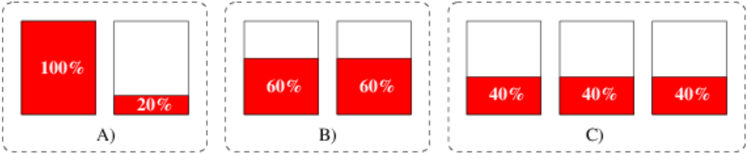
From the component point of view, platform **A)** is better

- *in general* it is easier to schedule tasks on such a platform

# Basic idea

Suppose that we need 120% bandwidth for our component

Various possibilities:




Which one is better?

From the component point of view, platform **A)** is better

- *in general* it is easier to schedule tasks on such a platform

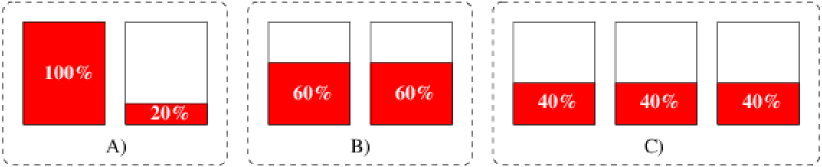
From the system point of view, it is difficult to say which platform is better

- Which one fits better on an existing physical platform?
- The goal should be to use the least number of physical processors
- Platform **C)** is a better candidate in most cases (smaller pieces)

# Basic idea

We want to take advantage of this trade-off to add flexibility

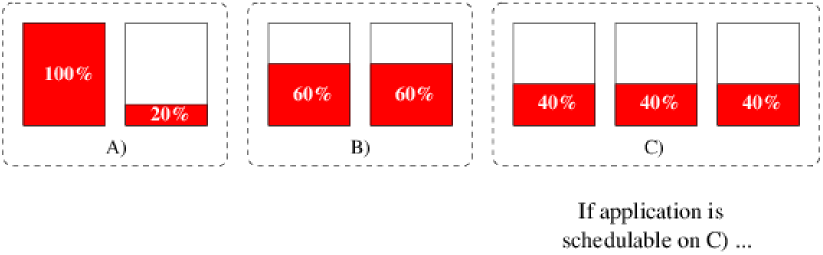
**1) Component schedulability.** We want to propose an interface, and the corresponding schedulability test, such that:



# Basic idea

We want to take advantage of this trade-off to add flexibility

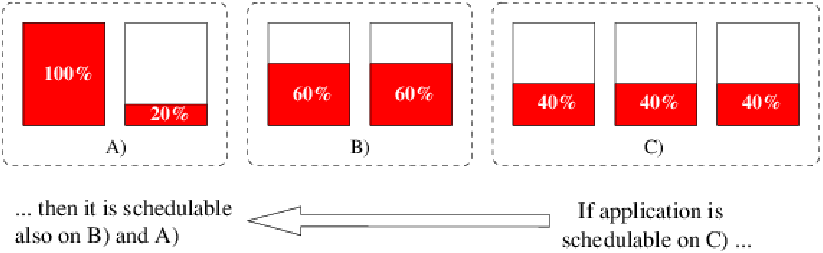
**1) Component schedulability.** We want to propose an interface, and the corresponding schedulability test, such that:



# Basic idea

We want to take advantage of this trade-off to add flexibility

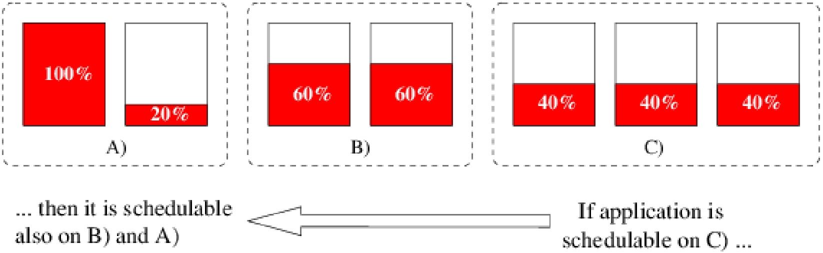
**1) Component schedulability.** We want to propose an interface, and the corresponding schedulability test, such that:



# Basic idea

We want to take advantage of this trade-off to add flexibility

**1) Component schedulability.** We want to propose an interface, and the corresponding schedulability test, such that:




**2) Platform instantiation and allocation.** We want to derive a run-time allocation algorithm that, starting from **C)**, derives the "best" platform, and allocates it

# Bounded Delay Multipartition Interface

- We propose the Bounded Delay Multipartition (BDM) Interface model
  - A BDM interface is characterised by a $\mathcal{I} = (m, \Delta, [\beta_1, \ldots, \beta_m])$
  - $m$ is the maximum number of virtual processors
  - $\Delta$ is the worst-case delay (i.e. the longest interval without service)
  - $\beta_k$ is the cumulative service utilisation with $k$ processors
  - We impose $0 \leq \beta_k - \beta_{k-1} \leq 1$, and $\beta_k - \beta_{k-1} \geq \beta_{k+1} - \beta_k$
- Which platforms are compliant with this interface model?
  - All platforms whose virtual processors have bandwidth $\alpha_i$ such that:

$$\sum_{i=0}^{k} \alpha_i \geq \beta_k$$

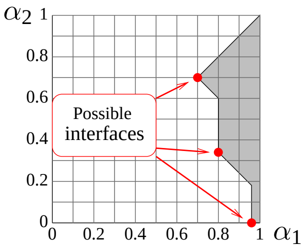
# Schedulability Analysis

- We use the work by Bini, Baruah, Bertogna

$$\bigwedge_{i=1,\dots,n} \bigvee_{k=1\dots,m} \sum_{j=1}^{k} \alpha_j (D_i - \Delta)_0 \geq kC_i + W_i$$

Where $W_i$ is the interference of the task

- Example with three tasks on 2 virtual processors:




| $i$ | $C_i$ | $T_i$ | $D_i$ | $W_i$ |
|---|---|---|---|---|
| 1 | 1 | 6 | 6 | 0 |
| 2 | 15 | 27 | 27 | 5 |
| 3 | 9 | 52 | 52 | 39 |

| $\beta_1 = \alpha_1$ | $\beta_2$ | $\alpha_2 = \beta_2 - \beta_1$ | $c(\Pi)$ |
|---|---|---|---|
| 0.7 | 1.4 | 0.7 | 0 |
| 0.8 | 1.14 | 0.34 | 0.46 |
| 0.96 | 0.96 | 0 | 0.96 |

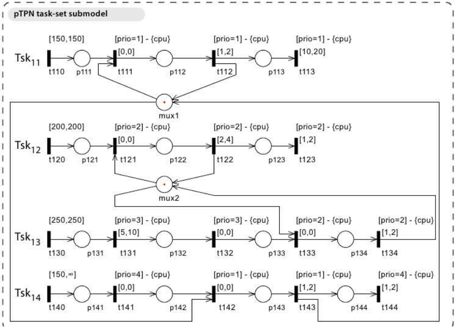
# Outline

# Component-based analysis and formal methods

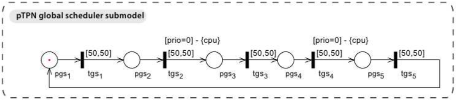
- Carnevali, Pizzuti, Vicario, Lipari (2010)
- The idea is to combine component based analysis with Preemptive Timed Petri Nets (pTPN)
  - The global scheduler is TDM (similar to ARINC 653)
  - The local scheduler can be FP or EDF
  - Threads can be modelled as periodic, sporadic or aperiodic tasks, and can share local resources through mutex
  - Components are independent

# pTPN model

Model of one component (application)




Model of the scheduler



Execution time:

| Model | # Classes | RAM | Time |
|---|---|---|---|
| model of $A_1$ | 32084 | $\sim$ 300 MB | $\sim$ 20 sec |
| model of $A_2$ | 183981 | $\sim$ 300 MB | $\sim$ 83 sec |
| model of $A_3$ | 26147 | $\sim$ 300 MB | $\sim$ 15 sec |
| flat model | $> 10^6$ | $>$ 4 GB (out of memory) | $>$ 13 min |

- The total running time is orders of magnitude less for the component-based model than for the flat model
- The state space is manageable, therefore it is possible to easily analyse large systems (more than 10 tasks)
- Work in progress:
  - Extend to more general global schedulers
  - Extend to interacting components

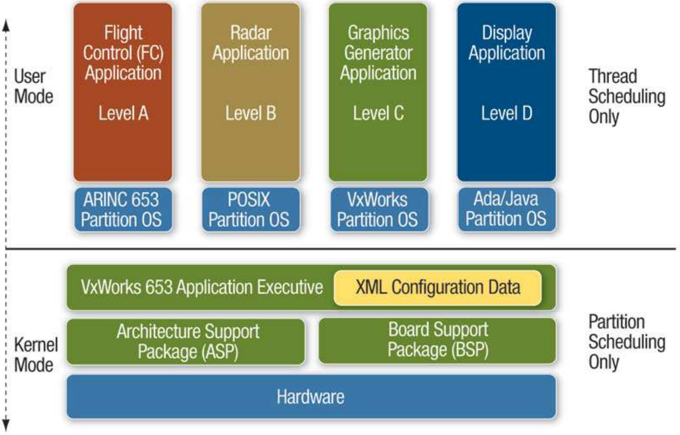
# Outline

# CBD in industry

- Industry is (mildly) pushing for a component-based technology for real-time embedded systems
- Cost increases more than linearly with complexity
  - Need to contain the development cost without compromising safety
  - Need to integrate components from different providers in the same ECU
  - Need to re-use existing components
  - Need to reduce testing effort and improve its quality and effectiveness

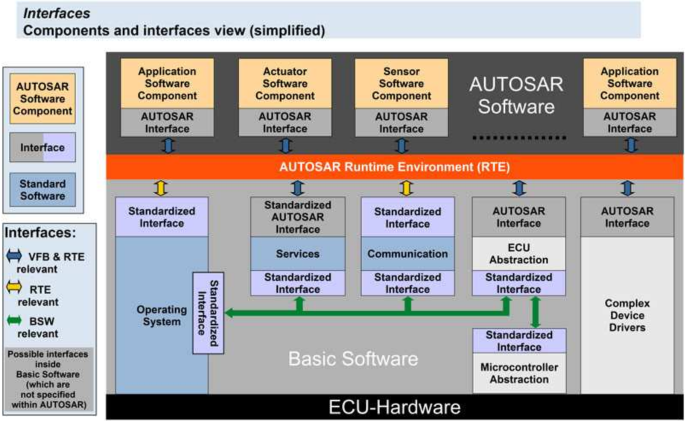
# Avionics

- ARINC 653 is a specification for space and time partitioning in avionic software
- The global scheduler is a simple TDM



# CBD in automotive

- AUTOSAR defines space and memory isolation among components/applications

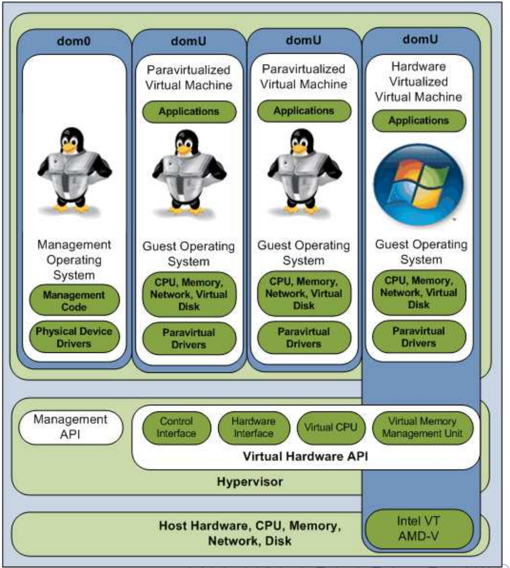


# Operating Systems

- Many EU projects on this topic:
  - FIRST: integrations of different schedulers through hierarchies (Shark, MARTE OS)
  - FRESCOR: An API for contract-based scheduling (MARTE OS, Linux, RTLinux)
  - ACTORS: Resource reservations and control, also on multicore (Linux)
  - IRMOS: Virtualisation for supporting Service Oriented real-time and multimedia systems (Linux)
- SSSA work in Linux:
  - SCHED_DEADLINE patch: provides EDF+CBS in Linux, will be extended to hierarchical systems
  - IRMOS scheduler: provides soft real-time EDF+CBS with group scheduling (via Cgroups)
- Other commercial kernels provide means to implement Resource Reservations and group scheduling, but no direct API for component based-RT.

# Virtual Machines

- Virtualisation can be considered as a way of providing temporal partitions
- Many VM Hypervisors provide global schedulers that partition the time line (e.g. Xen)
- Experiments with KVM in the IRMOS project



# Design methodologies

- Existing approaches for embedded RT systems
  - HRT-Hood
  - UML-RT (from OMG profile for SPT)
  - UML-MARTE (OMG profile for embedded systems)

- UML Marte enable schedulability analysis, but . . .
  - does not address the “component” issues very well
  - schedulability analysis is only done at the integration phase
  - no hierarchy of schedulers
  - heavily dependent on underlying scheduling mechanisms

# Outline

# Open problems

- *Component-based design and analysis of Real-Time Systems* already has more than ten years of research behind
  - Increasingly complex models, from the Liu&Layland model to multicore systems, and interacting components
- However, it has not yet been widely adopted in the industrial practice
  - Exception: avionics
- What is still missing?

# Open problems

- *Component-based design and analysis of Real-Time Systems* already has more than ten years of research behind
  - Increasingly complex models, from the Liu&Layland model to multicore systems, and interacting components
- However, it has not yet been widely adopted in the industrial practice
  - Exception: avionics
- What is still missing?
- Run-Time Support
  - A proper API to support components, time partitions and local schedulers
- Analysis
  - Component interaction using message passing
  - Formal methods to deal with components
- Scheduling
  - Scheduling on multicores and distributed systems

# Thank you

Scuola Superiore Sant'Anna

THANK YOU!

ReTiS Lab

# Bibliography I


Luis Almeida and Paulo Pedreiras.
Scheduling within temporal partitions: response-time analysis and server design.
In *Proceedings of the* $4^{th}$ *ACM International Conference on Embedded Software*, pages 95–103, Pisa, Italy, September 2004.

M. Bertogna, N. Fisher, and S. Baruah.
Resource-sharing servers for open environments.
5(3):202–219, 2009.

R.I. Davis and A. Burns.
Resource sharing in hierarchical fixed priority pre-emptive systems.
In *Real-Time Systems Symposium, 2006. RTSS '06. 27th IEEE International*, pages 257 –270, dec. 2006.

Zhong Deng and Jane win-shih Liu.
Scheduling real-time applications in Open environment.
In *Proceedings of the* $18^{th}$ *IEEE Real-Time Systems Symposium*, pages 308–319, San Francisco, CA, U.S.A., December 1997.

# Bibliography II


Xiang Feng and Aloysius K. Mok.
A model of hierarchical real-time virtual resources.
In *Proceedings of the $23^{rd}$ IEEE Real-Time Systems Symposium*, pages 26–35, Austin, TX, U.S.A., December 2002.

Giuseppe Lipari and Sanjoy K. Baruah.
Efficient scheduling of multi-task applications in open systems.
In *Proceedings of the $6^{th}$ Real-Time Systems and Applications Symposium,*, pages 166–175, Washington, DC U.S.A., June 2000.

Giuseppe Lipari and Enrico Bini.
Resource partitioning among real-time applications.
In *Proceedings of the $15^{th}$ Euromicro Conference on Real-Time Systems*, pages 151–158, Porto, Portugal, July 2003.

Giuseppe Lipari and Enrico Bini.
A methodology for designing hierarchical scheduling systems.
*Journal Embedded Computing*, 1(2):257–269, 2005.

# Bibliography III


G. Lipari and E. Bini.
A framework for hierarchical scheduling on multiprocessors: From application requirements to run-time allocation.
In *Proc. IEEE 31st Real-Time Systems Symp. (RTSS)*, pages 249–258, 2010.

G. Lipari, G. Buttazzo, and L. Abeni.
A bandwidth reservation algorithm for multi application systems.
In *Proceedings of the 5th International Conference of Real-Time Computing Systems and Applications*, pages 303–310, Hiroshima, Japan, October 1998. IEEE.

G. Lipari, J. Carpenter, and S. Baruah.
A framework for achieving inter-application isolation in multiprogrammed, hard real-time environments.
*Real-Time Systems Symposium, 2000. Proceedings. The 21st IEEE*, pages 217–226, 2000.

# Bibliography IV


G. Lipari, G. Lamastra, and L. Abeni.
Task synchronization in reservation-based real-time systems.
*IEEE Transactions on Computers*, 53(12):1591–1601, 2004.

José L. Lorente, Giuseppe Lipari, and Enrico Bini.
A hierarchical scheduling model for component-based real-time systems.
In *Proceedings of the* $20^{th}$ *International Parallel and Distributed Processing Symposium*, Rhodes Island, Greece, April 2006.

Aloysius K. Mok and Xiang Feng.
Towards compositionality in real-time resource partitioning based on regularity bounds.
In *Proceedings of the* $22^{nd}$ *IEEE Real-Time Systems Symposium*, pages 129–138, London, United Kingdom, 2001.

Insik Shin and Insup Lee.
Periodic resource model for compositional real-time guarantees.
In *Proceedings of the* $24^{th}$ *Real-Time Systems Symposium*, pages 2–13, Cancun, Mexico, December 2003.