Hierarchical Scheduling for Components

Luca Abeni
luca.abeni@santannapisa.it

October 14, 2019

Component-Based Development

- Complex software systems built by “connecting” smaller components
- Component: described by an *interface*
  - Software Interface
  - Described using an IDL (Interface Definition Language), or similar technologies
  - Should fully describe the component, so that it can be used without knowing the internals
- Components often run on different physical nodes
- Or are isolated using a VM!
  - Can even use different OSs!

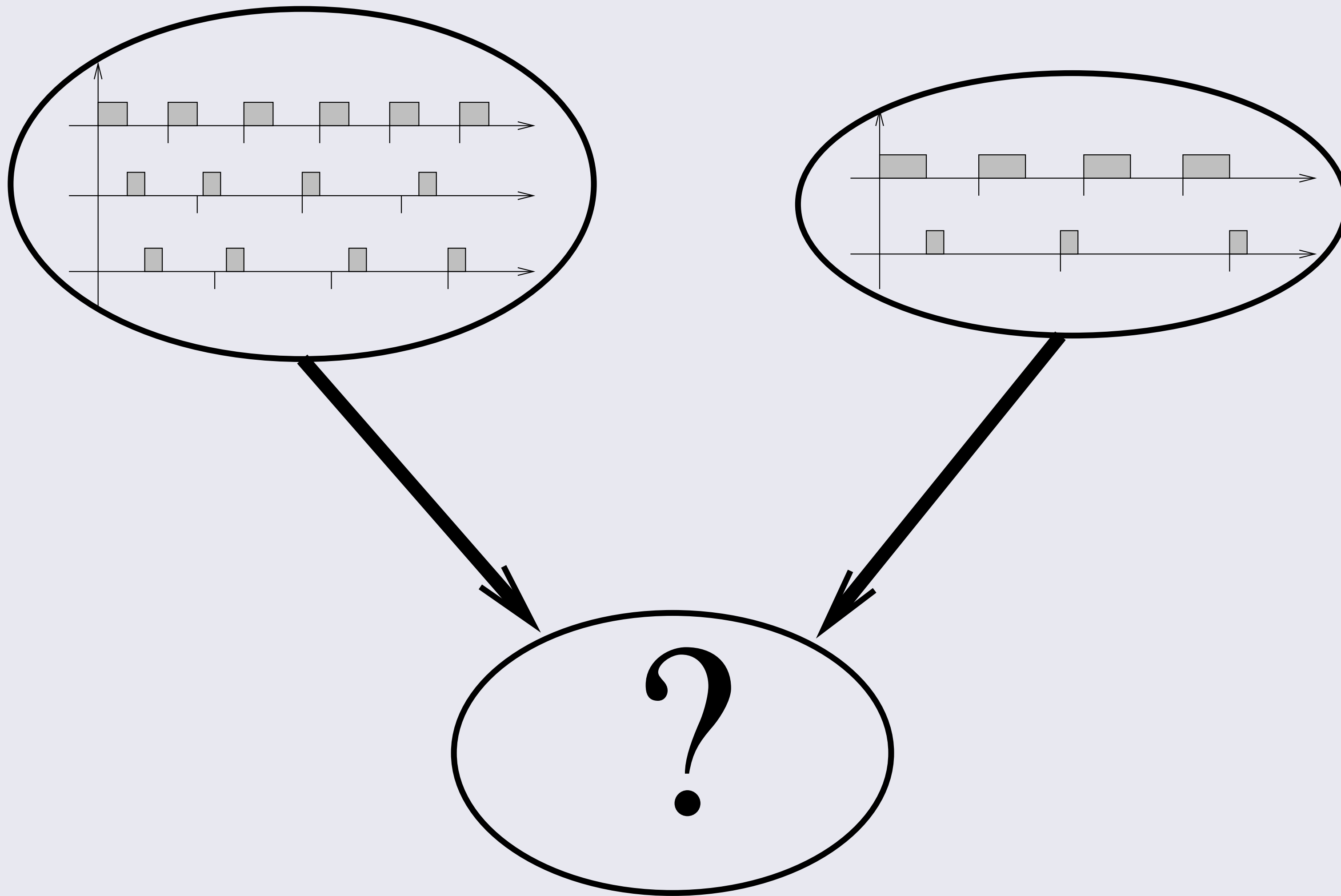
Non Functional Interfaces

- Component interfaces often tend to focus on functional aspects of the component
  - What about non-functional properties?
  - For example: temporal behaviour...
- Do the component interfaces consider the respect of timing constraints?
- In other words: can I build a real-time system using a component-based approach?

Component-Based Real-Time Systems

- Assume to have N components...
  - Each component is described by its interface...
  - ...But a description of its temporal behaviour is also somehow provided...
- Component are assumed to respect some temporal constraints
  - Some kind of “contract” provided by real-time guarantees
- Is it possible to combine the components so that the timing of the system is predictable?
  - Can real-time guarantees be combined? How?

Combining Real-Time Guarantees



- Schedulability analysis in each component...
- What about the resulting system?

Real-Time Components

- Single-thread/process components $\rightarrow$ combining them is easy...
  - Components can be modelled as real-time tasks
  - Simple (C, D, T) model, or something more complex...
- What about components composed by multiple execution flows?
  - Each component is composed by multiple real-time tasks...
  - Model for a component?
  - How to summarize its temporal requirements?
  - Scheduler in the component? How to model it?

Multi-Task Real-Time Components

- Component $\mathcal{C}^i$ composed by n^i tasks
- More complex component model... How to handle it?
  - We only know how to schedule single tasks...
  - And we need to somehow “summarise” the requirements of a compoment!

$$\mathcal{C}^i = \{(C_0^i, D_0^i, T_0^i), (C_1^i, D_1^i, T_1^i), \ldots, (C_{n^i}^i, D_{n^i}^i, T_{n^i}^i)\}$$

- So, 2 main issues:
  1. Describe the temporal requirements of a component in a simple way
  2. Schedule the components, and somehow “combine” their temporal guarantees

The "not so smart" Solution

- Each component is a set of real-time tasks:

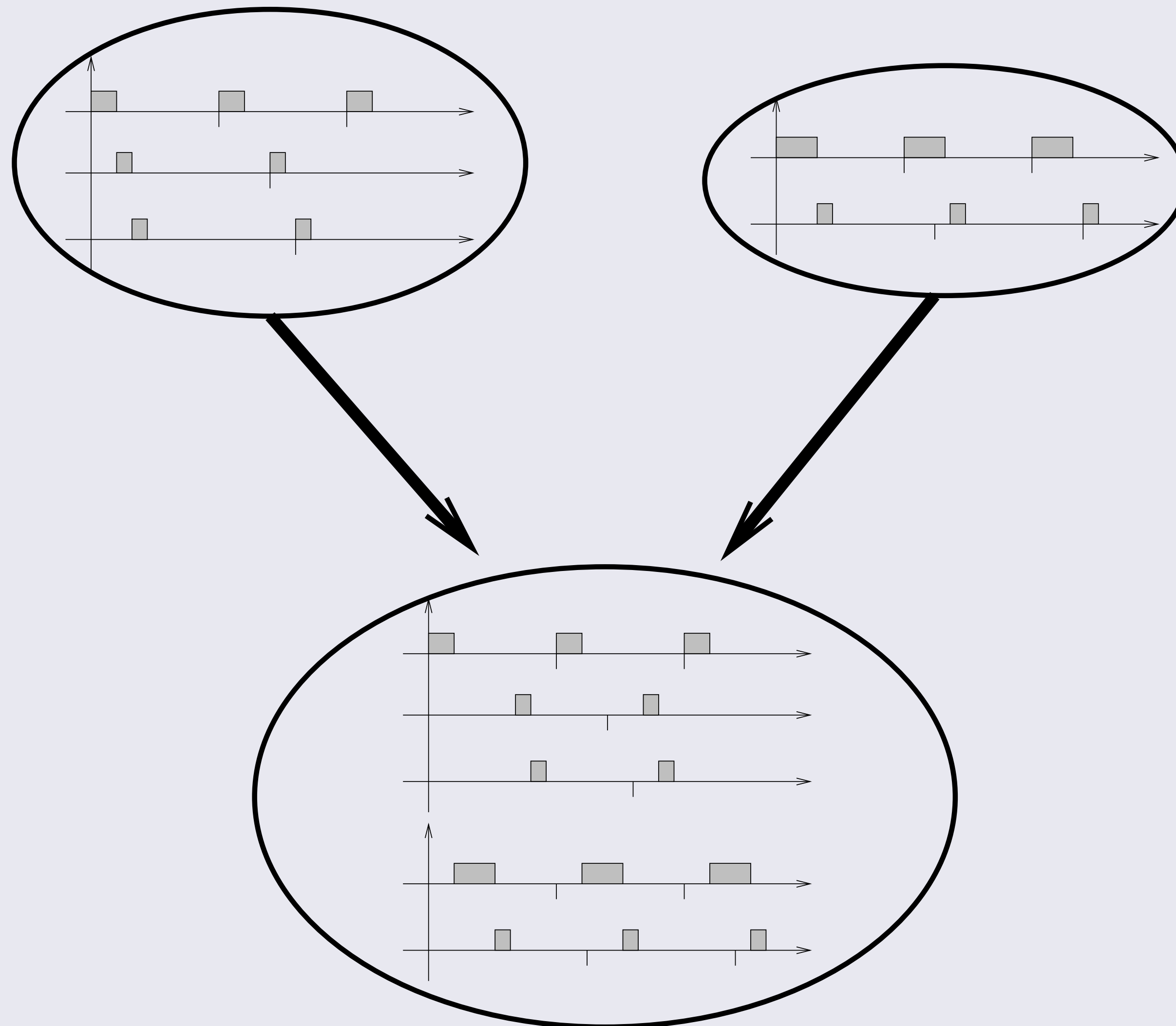
$$\mathcal{C}^i = \{(C^i_j, D^i_j, T^i_j)\}$$

- Build the "global taskset" composed by all the tasks from all the components

$$\Gamma = \bigcup_i \mathcal{C}^i$$

- ...And use some known real-time scheduler (RM, EDF, ...) on Γ!

Flattened Scheduling



- One single “flattened” scheduler seeing all the tasks

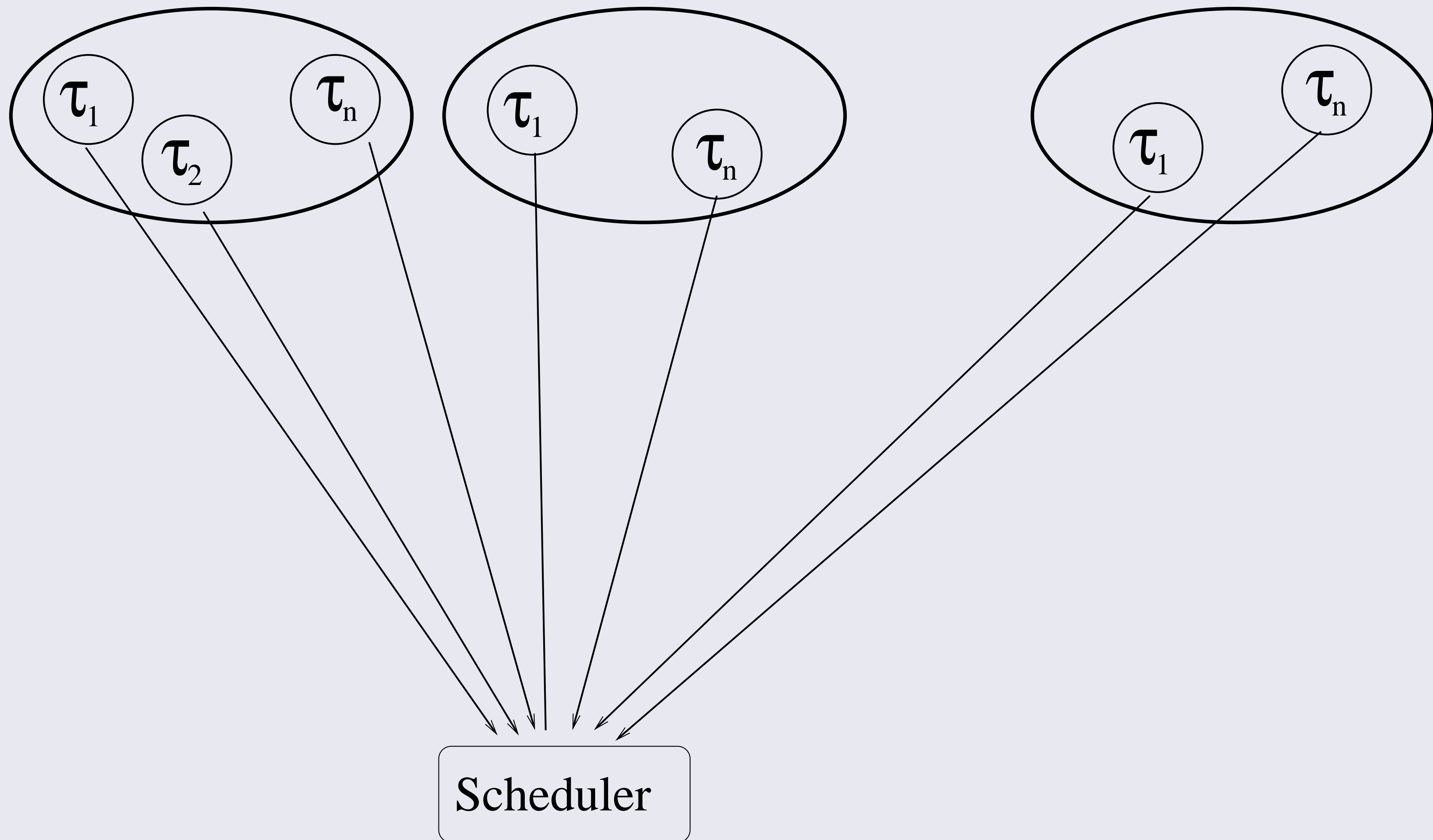
Why it is “not so smart”

- One single scheduler, that must “see” all the tasks of all the components
  - Internals of the components have to be exposed!
  - Components cannot have their own “local” schedulers
  - Misbehaving tasks in a component can affect other components
    - No isolation!!!
- Using fixed priorities might be “not so simple”
  - Think about RM: priorities in a component might depend on other components...
  - Components are not “inter-changeable” anymore!

Practical Issues

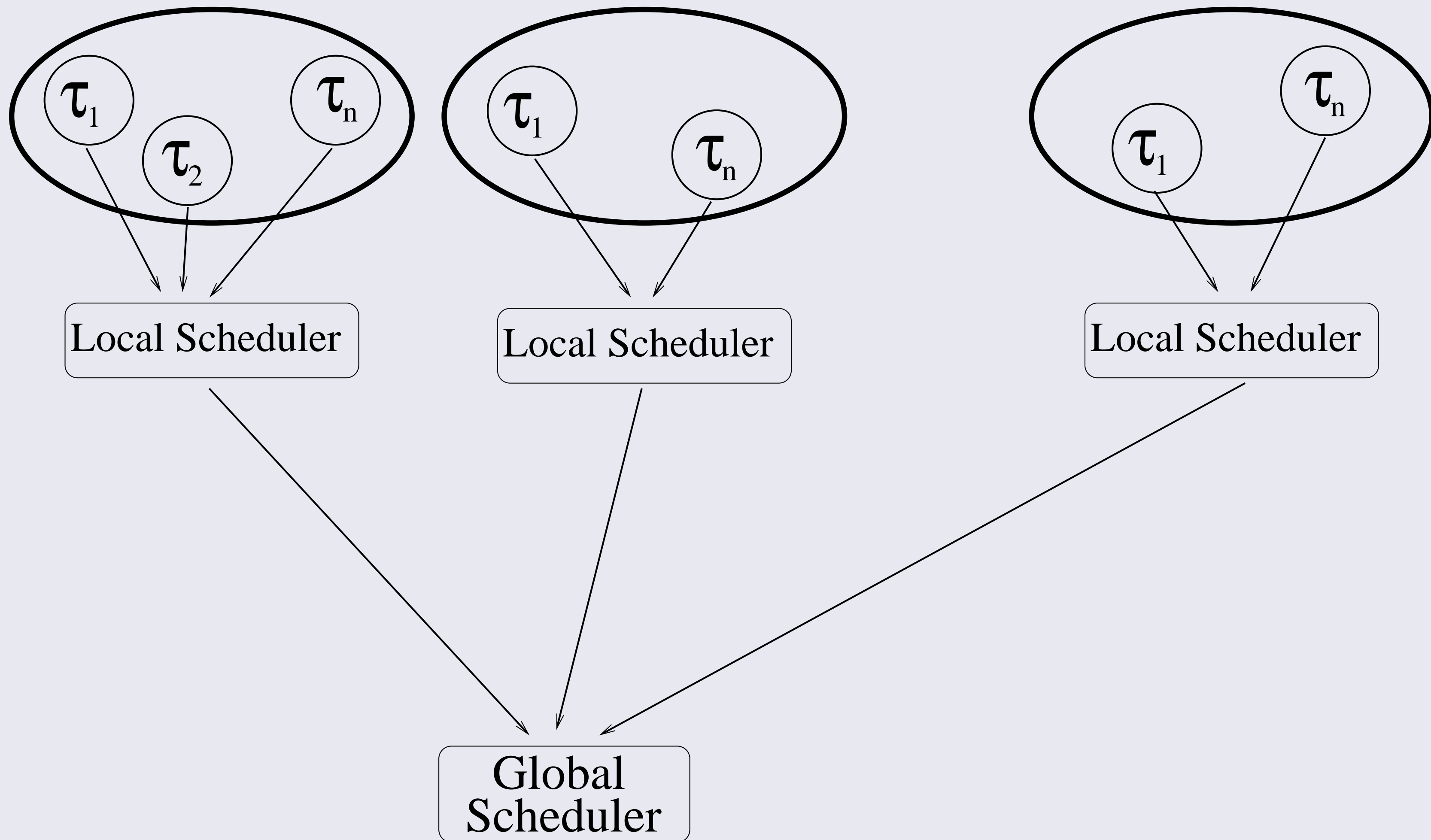
- Components can be isolated using VMs
  - The scheduler only sees a VM per component, but cannot see the tasks inside it
  - Para-virtualization (of the OS scheduler) could be used to address this issue, but it is not so simple...
  - ...And requires huge modifications to host, guest, and components!
- So, how to schedule VMs?
- Two-level hierarchical scheduling system
  - Host (global / root) scheduler, scheduling VMs
  - Each VM contains its (local / 2nd level) scheduler

From a 1-Level Scheduler...



- Scheduler assigns CPU to tasks “inside the components”

...To a 2-Levels Hierarchy



- Global Scheduler assigns CPU to components
- Local Schedulers assign CPU to single tasks

Hierarchical Scheduling

- The global scheduler does not see the components' tasks
- Components are free to define their own (fixed priorities, EDF, whatever) schedulers
  - No problems in assigning fixed priorities to tasks!
- Possible implementation: a VM per component
  - Global scheduler: host / hypervisor scheduler
  - Local scheduler: guest scheduler
- Problem: what to use as a global scheduler?
  - We must have a model for it
  - Must allow to compose the "local guarantees"
- Before going on, summary of RT definitions and concepts

Processes, Threads, and Tasks

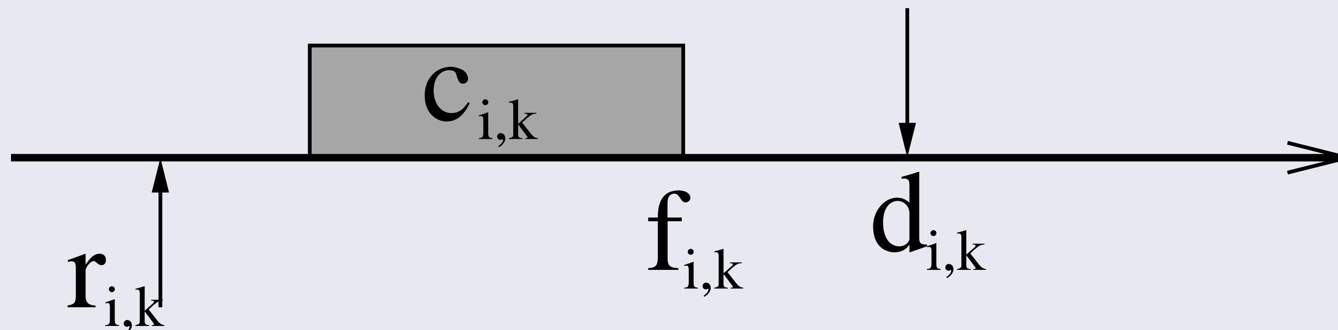
- Algorithm $\rightarrow$ logical procedure used to solve a problem
- Program $\rightarrow$ formal description of an algorithm, using a *programming language*
- Process $\rightarrow$ instance of a program (program in execution)
  - Program: static entity
  - Process: dynamic entity
- The term *task* is used to indicate a schedulable entity (either a process or a thread)
  - Thread $\rightarrow$ flow of execution
  - Process $\rightarrow$ flow of execution + private resources (address space, file table, etc...)

Real-Time Tasks

- A task can be seen as a sequence of actions . . .
- . . . and a deadline must be associated to each one of them!
  - Some kind of formal model is needed to identify these “actions” and associate deadlines to them

Mathematical Model of a Task - 1

- Real-Time task τ_i: stream of jobs (or instances) $J_{i,k}$
- Each job $J_{i,k} = (r_{i,k}, c_{i,k}, d_{i,k})$:
  - Arrives at time $r_{i,k}$ (activation time)
  - Executes for a time $c_{i,k}$
  - Finishes at time $f_{i,k}$
  - Should finish within an absolute deadline $d_{i,k}$



Mathematical Model of a Task - 2

- Job: abstraction used to associate deadlines (temporal constraints) to activities
  - $r_{i,k}$: time when job $J_{i,k}$ is *activated* (by an external event, a timer, an explicit activation, etc...)
  - $c_{i,k}$: computation time needed by job $J_{i,k}$ to complete
  - $d_{i,k}$: absolute time instant by which job $J_{i,k}$ must complete
    - job $J_{i,k}$ respects its deadline if $f_{i,k} \leq d_{i,k}$
- Response time of job $J_{i,k}$: $\rho_{i,k} = f_{i,k} - r_{i,k}$

Periodic / Sporadic Tasks

Periodic task $\tau_i = (C_i, D_i, T_i)$: stream of jobs $J_{i,k}$, with

$$\begin{aligned} r_{i,k+1} &= r_{i,k} + T_i \text{ (or, } \geq r_{i,k} + T_i) \\ d_{i,k} &= r_{i,k} + D_i \\ C_i &= \max_k \{c_{i,k}\} \end{aligned}$$

- T_i is the task *period* (or *minimum inter-arrival time*), D_i is the task *relative deadline*, C_i is the task worst-case execution time (WCET)
- R_i: worst-case response time $\rightarrow$ $R_i = max_k\{\rho_{i,k}\} = max_k\{f_{i,k} - r_{i,k}\}$
  - for the task to be correctly scheduled, it must be $R_i \leq D_i$

Definitions

- Algorithm $\rightarrow$ logical procedure used to solve a problem
- Program $\rightarrow$ formal description of an algorithm, using a *programming language*
- Process $\rightarrow$ instance of a program (program in execution)
  - Program: static entity
  - Process: dynamic entity
- The term *task* is used to indicate a schedulable entity (either a process or a thread)
  - Thread $\rightarrow$ flow of execution
  - Process $\rightarrow$ flow of execution + private resources (address space, file table, etc...)

Executing Concurrent Tasks

- Tasks do not run on bare hardware...
  - How can multiple tasks execute on one single CPU?
  - The OS kernel creates the illusion of having more CPUs, so that multiple tasks execute in parallel
    - Tasks have the illusion of executing concurrently
    - A dedicated CPU per task

Scheduling Concurrent Tasks

- Concurrency is implemented by multiplexing tasks on the same CPU...
  - Tasks are alternated on a real CPU...
  - ...And the *task scheduler* decides which task executes at a given instant in time
- Tasks are associated temporal constraints (deadlines)
  - The scheduler must allocate the CPU to tasks so that their deadlines are respected

Scheduler - 1

- Scheduler: generates a *schedule* from a set of tasks
  - Interesting definition: the scheduler is the thing that generates the schedule
- Let's be serious... Start from a mathematical model
  - First, consider UP systems (simpler definition)
    - A schedule $\sigma(t)$ is a function mapping time t into an executing task

$$\sigma : t \to \mathcal{T} \cup \tau_{idle}$$

      where $\mathcal{T}$ is the taskset and τ_{idle} is the *idle task*
  - For an SMP system (m CPUs), $\sigma(t)$ can be extended to map t in vectors $\tau \in (\mathcal{T} \cup \tau_{idle})^m$

Scheduler - 2

- Scheduler: implements $\sigma(t)$
  - The scheduler is responsible for selecting the task to execute at time t
- From an algorithmic point of view
  - Scheduling algorithm $\rightarrow$ Algorithm used to select for each time instant t a task to be executed on a CPU among the ready task
  - Given a task set $\mathcal{T}$, a scheduling algorithm $\mathcal{A}$ generates the schedule $\sigma_{\mathcal{A}}(t)$
- A task set is schedulable by an algorithm $\mathcal{A}$ if $\sigma_{\mathcal{A}}$ does not contain missed deadlines
- Schedulability test $\rightarrow$ check if $\mathcal{T}$ is schedulable by $\mathcal{A}$

Real-Time Guarantees in a Component

- First requirement: analyse the schedulability of a component independently from other components
  - This means that the root scheduler must provide some kind of temporal protection between components
- Various possibilities
  - Resource Reservations / server-based approach
  - Static time partitioning
  - ...
- In any case, the root scheduler must guarantee that each VM receives a minimum amount of resources in a time interval

Schedulability Analysis: the Basic Idea

- (Over?)Simplifying things a little bit...
- ...Suppose to know the amount of time needed by a component to respect its temporal constraints and the amount of time provided by the global scheduler
- A component is “schedulable” if

$$\text{demanded time} \leq \text{supplied time}$$

  - “demanded time”: amount of time (in a time interval) needed by a component
  - “supplied time”: amount of time (in a time interval) given by the global scheduler to a component
- Of course the devil is in the details

Demanded Time

- Amount of time needed by a component to respect its temporal constraints
  - Depends on the time interval we are considering
  - Depends on the component's local scheduler
    - EDF $\rightarrow dbf(t) = \sum_j \max\{0, \left\lfloor \frac{t+T_j-D_j}{T_j} \right\rfloor\} C_j$
    - RM: $\rightarrow$ workload $W(t) = C_i + \sum_{j<i} \left\lceil \frac{t}{T_i} \right\rceil C_j$
  - Note: $W(t)$ is very pessimistic, $dbf(t)$ is not
- This is the description of the temporal requirements of a component we were searching for...
- And what about the supplied time?

Supplied Time

- Description of the root scheduler temporal behaviour
- More formally:
  - Depends on the time interval t we are considering
  - Depends on the root scheduler $\mathcal{A}$
- Minimum amount of time given by $\mathcal{A}$ to a VM in a time interval of size s
  - Given all the time interval $(t_0, t_1) : t_1 - t_0 = s$...
  - ...Compute the size of the sub-interval in which $\sigma(t) = VM$...
  - ...And then find the minimum!

Supplied Time Bound Function

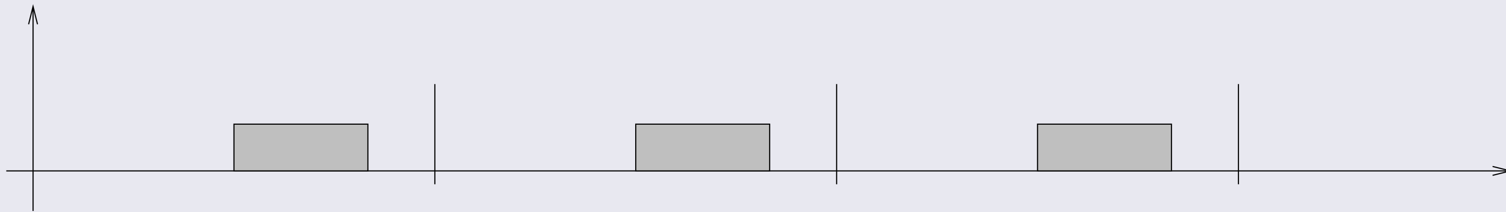
- Even more formally:
  - Define $s(t) = \begin{cases} 1 & \text{if } \alpha(t) = VM \\ 0 & \text{otherwise} \end{cases}$
  - Time for VM in $(t_0, t_0 + s)$: $\int_{t_0}^{t_0+s} s(t)dt$
  - Then, compute the minimum over t_0
- $sbf(t) = \min_{t_0} \int_{t_0}^{t_0+t} s(x)dx$

Example: Static Time Partitioning

- First (very simple) example of VM scheduling: static time partitioning
  - Static schedule describing when time is assigned to each VM
  - Pre-computed $\sigma(t)$
- Generally, periodic!
  - Otherwise, need to store an infinite schedule...
  - ...Might be problematic!
- Example: VM$_{\mathcal{A}}$ is scheduled in $(3, 4)$, $(9, 10)$, $(15, 16)$, ...
  - More formally: $s(t) = 1$ if $6k + 3 \leq t \leq 6k + 4$, $s(t) = 0$ otherwise

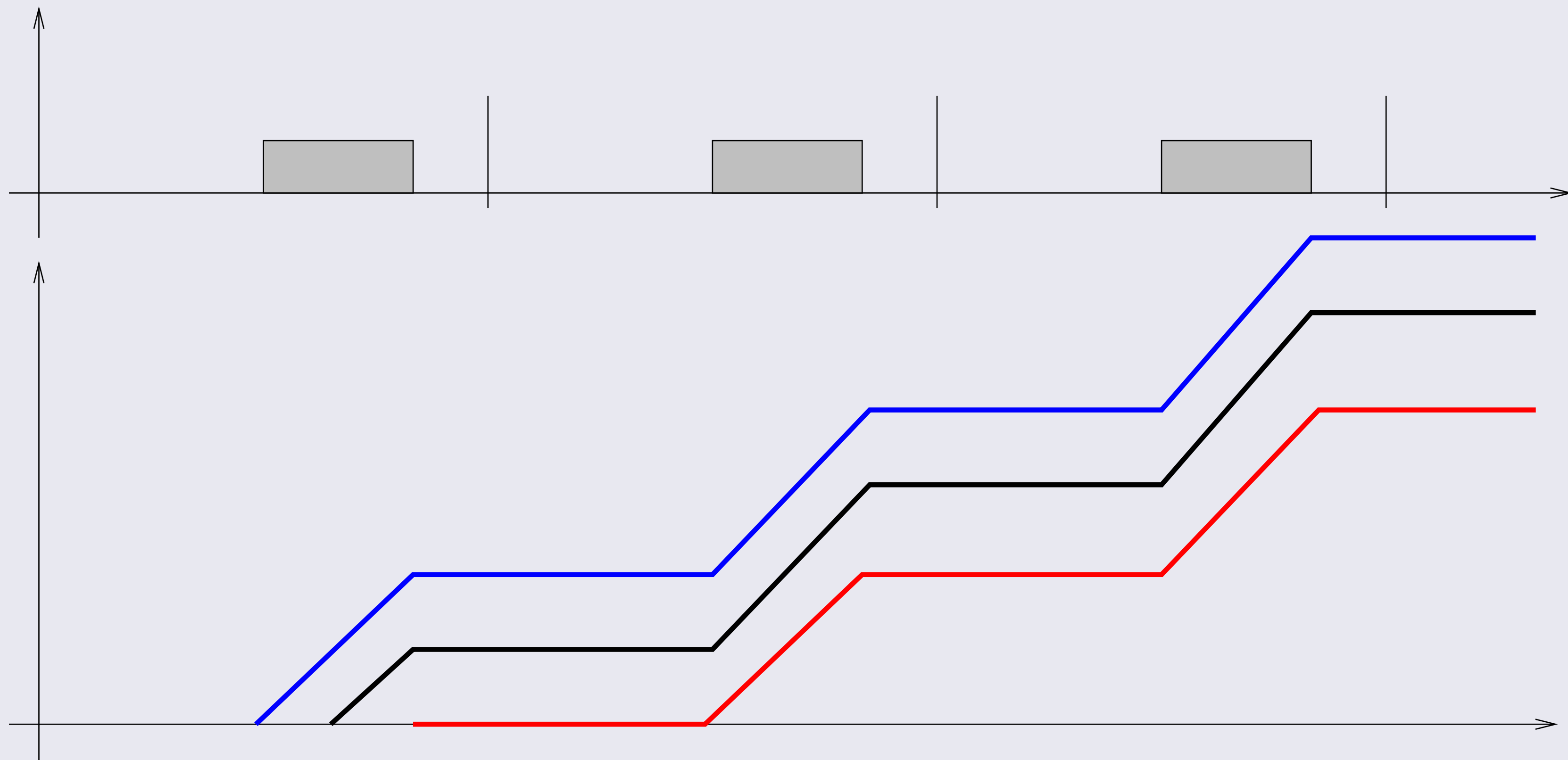
Example: Static Time Partitioning - 2

$$s(t) = \begin{cases} 1 & \text{if } 6k + 3 \le t \le 6k + 4 \\ 0 & \text{otherwise} \end{cases}$$



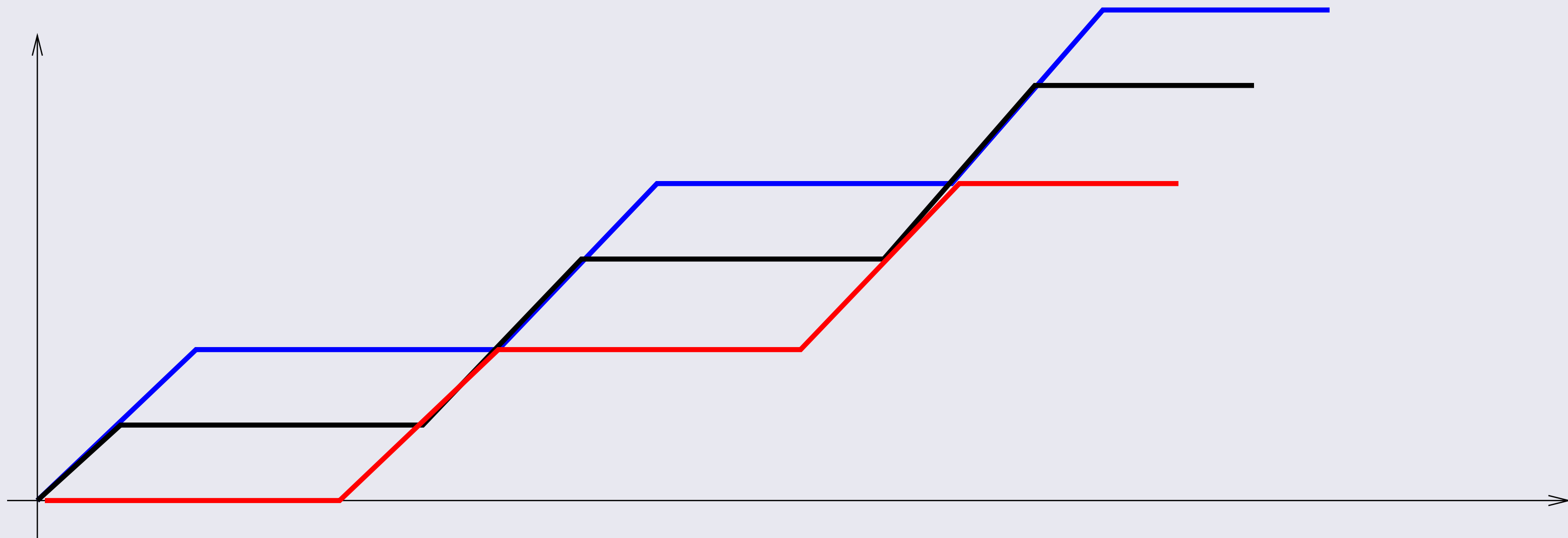
- What is the supply bound function $sbf(t)$ in this case?
- Let's try different supply functions compatibe with this schedule...
- ...And see what is the worst case!
  - Intervals of size t starting at different times...

Example: Static Time Partitioning - 3



- Different supply functions depending on when the considered interval begins
- Which one is the worst case (supply bound function)?

Example: Static Time Partitioning - 4



- Different supply functions depending on when the considered interval begins
- Which one is the worst case (supply bound function)?
  - The red one!

Example: Static Time Partitioning - 5