POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA,
INFORMAZIONE E BIOINGEGNERIA

EMBEDDED SYSTEMS
Prof. William Fornaciari

Real-time Systems
V2.0

William Fornaciari

<william.fornaciari@polimi.it>

Outline

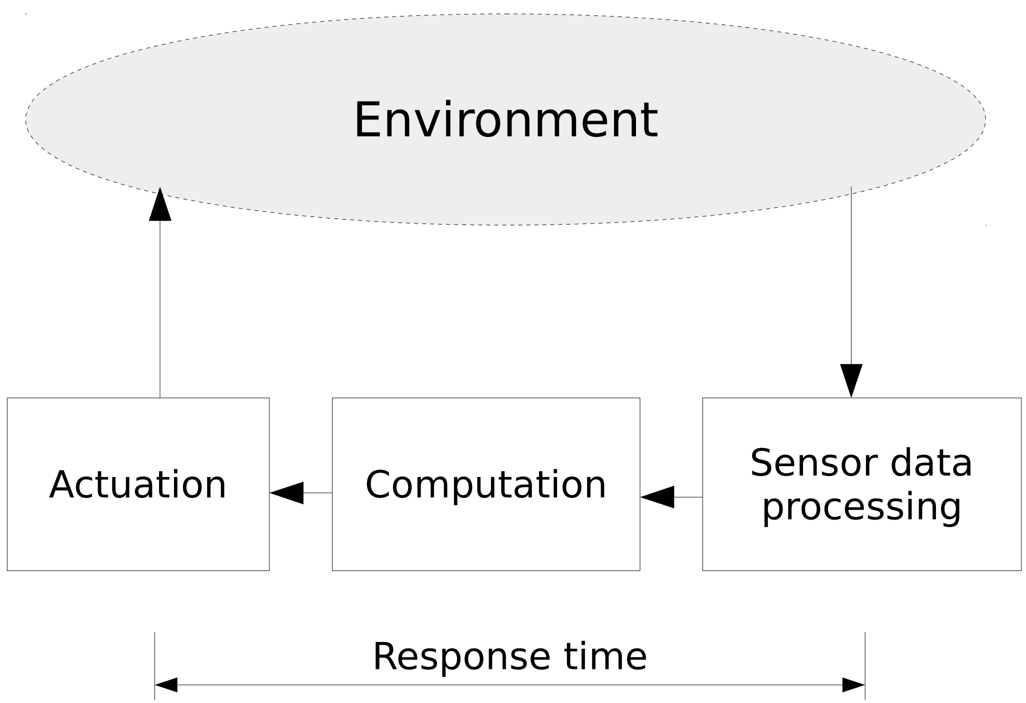
- Introduction to Real-time systems
- Predictability
- Real-time Operating Systems (RTOS)
- Real-time Scheduling

What is a real-time system?

- *Definition 1* – A real-time system is a system in which the correctness of its behavior depends on
  - The **logical correctness** of the tasks' output
  - The **time** within the output are produced
- *Definition 2* – A real time system is a system which has to respond to externally generated input stimuli within a finite and specified period
  - Reactive system with timing constraints
- A wide class of embedded systems is made by real-time systems

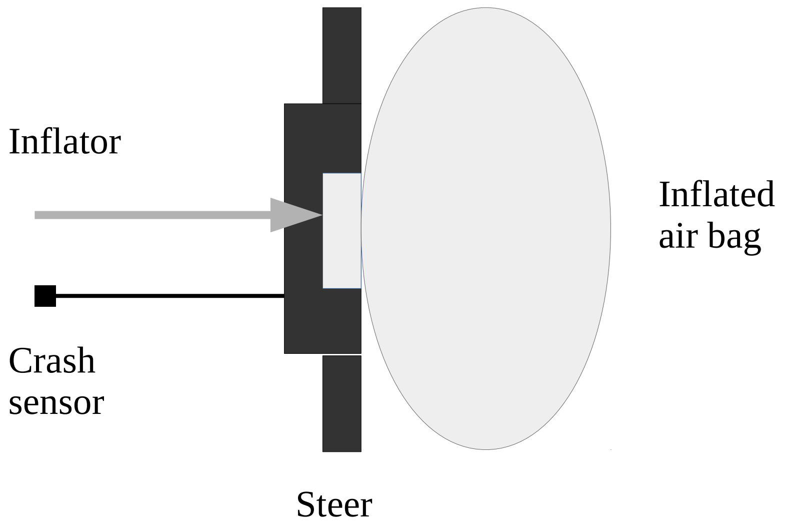
Structure

- Real-time systems are often designed to address dynamic control problems
  - Timing constraints in the control-loop



Introduction to Real-time Systems

Example: *Airbag*



- When a collision is detected by the crash sensor, the inflation of the air bag must be triggered
- The trigger must be within 10-20ms from the time instant in which the collision occurred (has been detected)

Classification

Timing constraints

- The execution of tasks in real-time systems is typically characterized by stringent timing constraints
  - i.e. tasks execution should complete within a **time deadline**

- Depending on the consequences of missing the deadline, we can distinguish among three criticality classes of real-time
  - ***Hard*** – missing the deadline may lead to catastrophic events on persons or on the system under control
  - ***Firm*** – missing the deadline does not cause system damages, but the output value is not valid
  - ***Soft*** – missing the deadline does not affect the validity of the output, but it results to be a performance degradation

Classification

Example: ***Autonomous Emergency Braking System***

- Detection of obstacles (pedestrian, stopped cars,...)
- Trigger the braking action
  - Consider the current car speed, distance of the obstacle...
  - How much time to trigger the brake?
- Consequences of a deadline miss?

Classification

Example: ***Video player application***

- What happens if the application processes a frame in 100ms (10 FPS) instead of 40ms (25 FPS) ?

Properties

It's all about time

- *Timeliness*
  - System must include time management mechanisms

- *Predictability*
  - Possibility of predicting a priori if the timing constraints can be guaranteed or not
  - The temporal behavior of the system must be analyzable at design-time

- *Determinism*
  - Provide timing guarantees on the execution of tasks at run-time, also in case of occurrence of external events to handle

Properties

Misconceptions

- "Real-time computing is equivalent to fast computing." (NO!)
- *Fast computing* objective is to minimize the average response time of a task set
- *Real-time computing* objective is to meet the individual timing requirement of each task!

Predictability

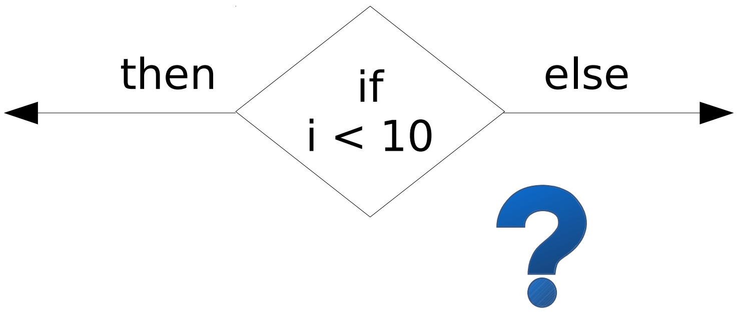
Predictability: potential enemies

- Processor
- Cache
- Direct Memory Access (DMA)
- Interrupts
- System calls
- Memory management
- Locking mechanisms
- Programming language

Predictability

Processor

- Modern processor architectures include hardware mechanisms to improve average performance
  - *Instruction / data Prefetching*
  - *Pipeline length and status*
  - *Speculation branch prediction*,
  - …

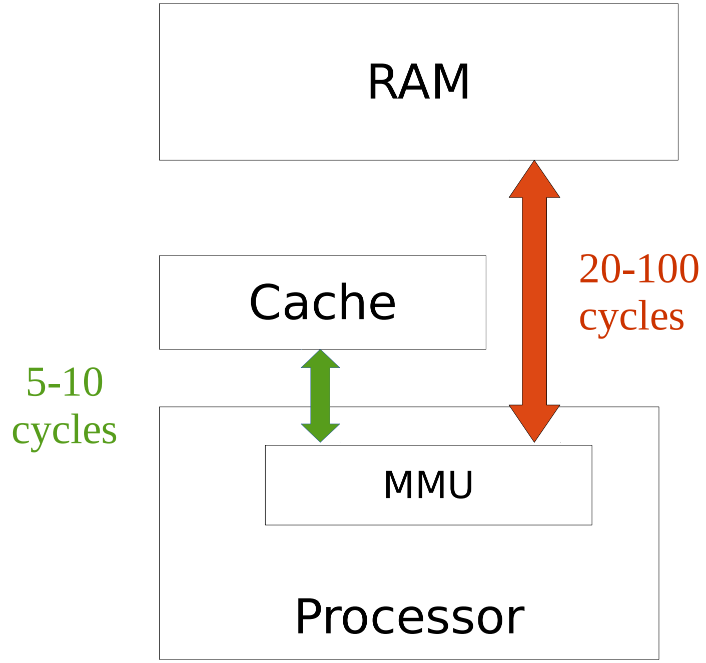


- **OK for average performance, but they are also source of non-determinism!**

Predictability

Cache

- Memory access instructions (LOAD/STORE) may require an unpredictable number of cycles depending on the occurrence of a cache *hit* or *miss*

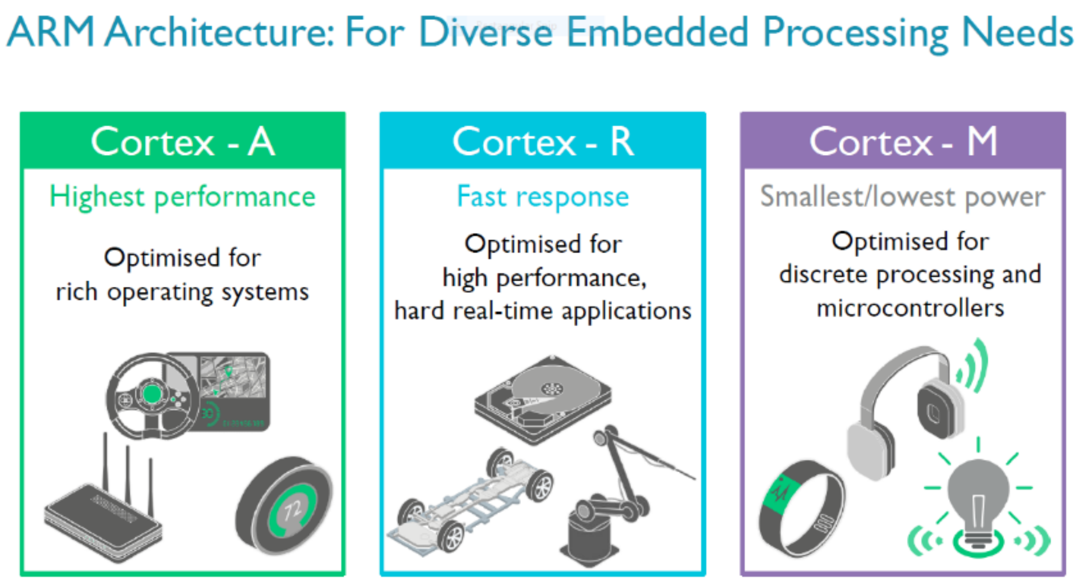


Predictability

Processors and Cache

- *Possible solution*
  - Micro-controllers have a much simple design, which provides more guarantees in terms of predictability
  - Adopting specific processors for real-time applications

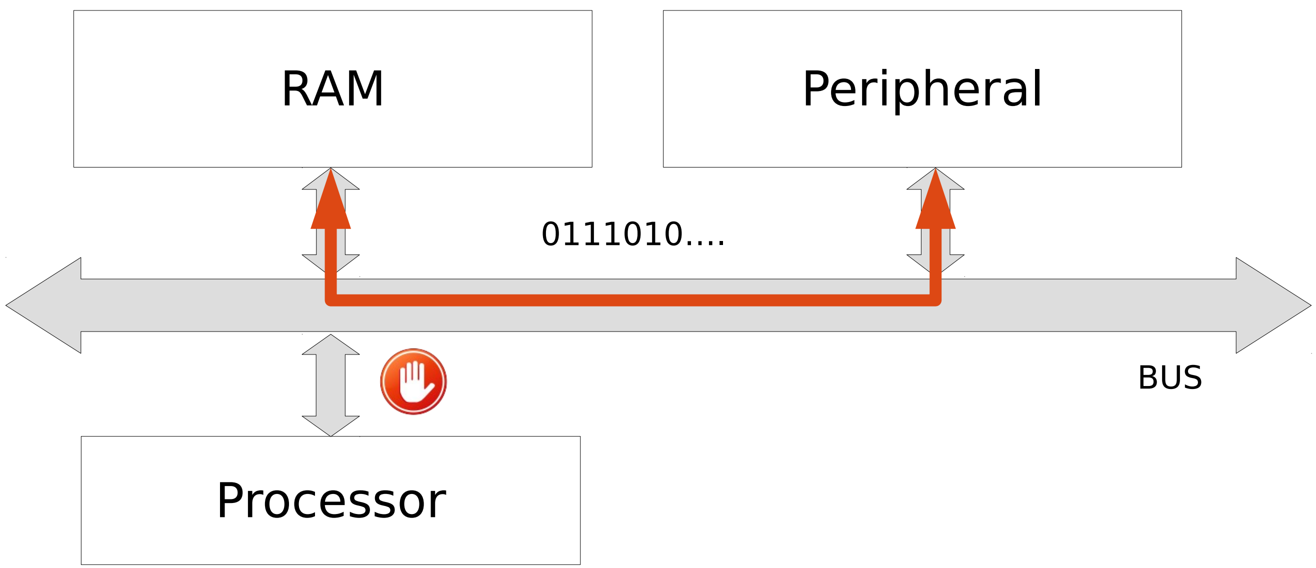
    e.g., ARM Cortex R family



Predictability

Direct Memory Access (DMA)

- DMA is used to transfer data between devices and main memory, without involving the CPU
  - Since the bus is shared, DMA could steal cycles to the CPU to perform data transfers (*cycle stealing*)



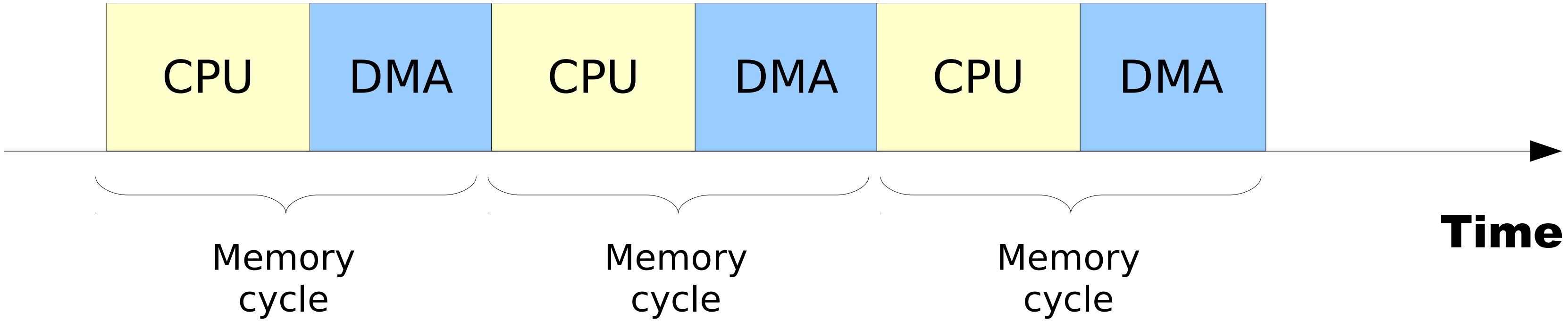
Predictability

Direct Memory Access (DMA)

- *Possible solution*
  - **Time-slice** method: split a memory cycle into separate time slots (for CPU and DMA), allowing predictive accesses to the shared bus

    More expensive but more predictive solution
    CPU and DMA do not conflict

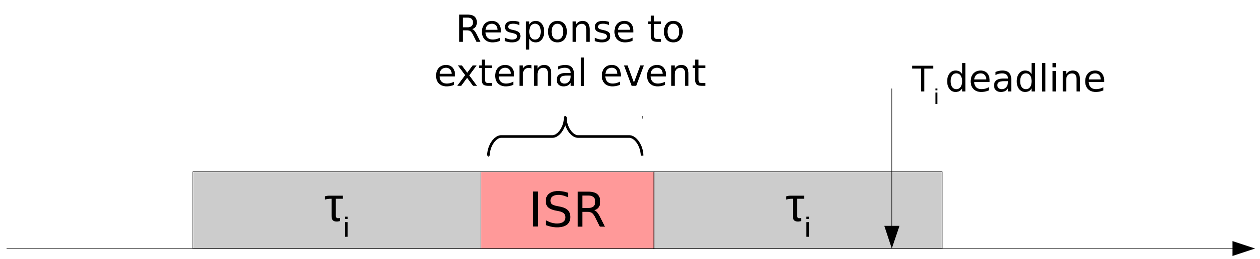
    Response time does not increase due to DMA



Predictability

Interrupts

- Serving an I/O request may delay the termination of the current task, with risk of missing the deadline



- *Possible solutions*
  - **(A) Disable interrupts (except the timer interrupt)**

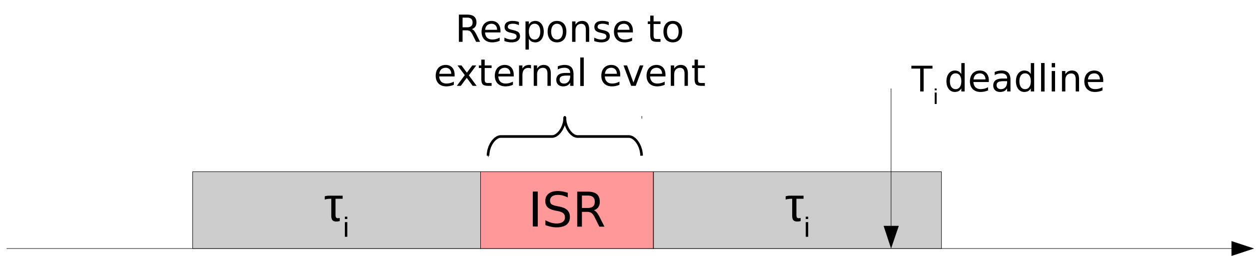
    Low processor efficiency on I/O operations

    Tasks must manage I/O operations by themselves via polling and direct access to peripheral registers

Predictability

Interrupts

- Serving an I/O request may delay the termination of the current task, with risk of missing the deadline



- *Possible solutions*
  - **(B) I/O requests are periodically handled by OS routines**

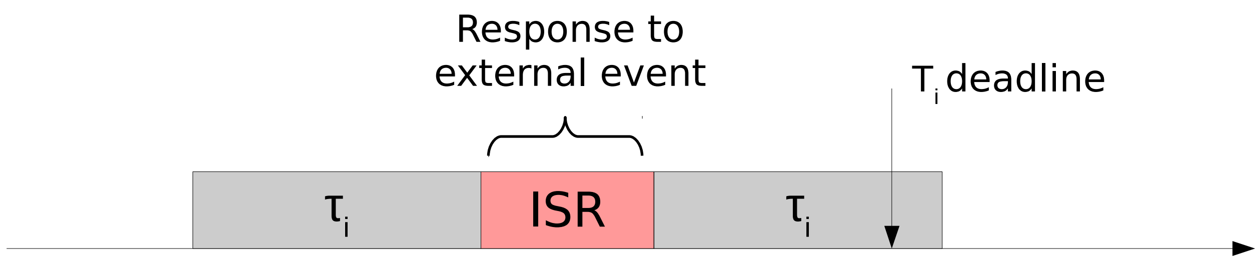
    Interrupts from devices are disabled but…

    Dedicated kernel routines periodically handle I/O operations

Predictability

Interrupts

- Serving an I/O request may delay the termination of the current task, with risk of missing the deadline



- *Possible solutions*
  - **(C) Minimal interrupt service routine**

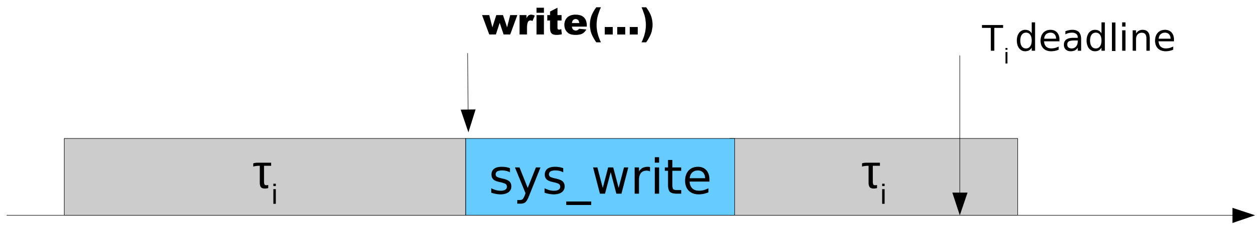
    The routines just set a flag and create a normal task for handling the request

    The I/O handler task is scheduled as any other task

Predictability

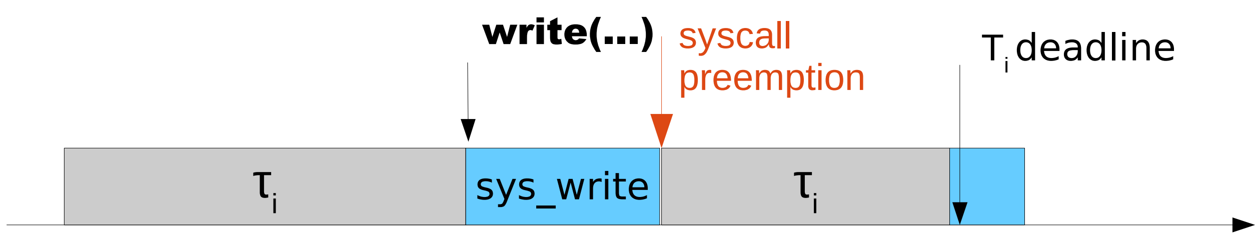
System calls

- Serving an I/O request may delay the termination of the current task, with risk of missing the deadline



- *Possible solution*
  - **Interruptible kernel routines**

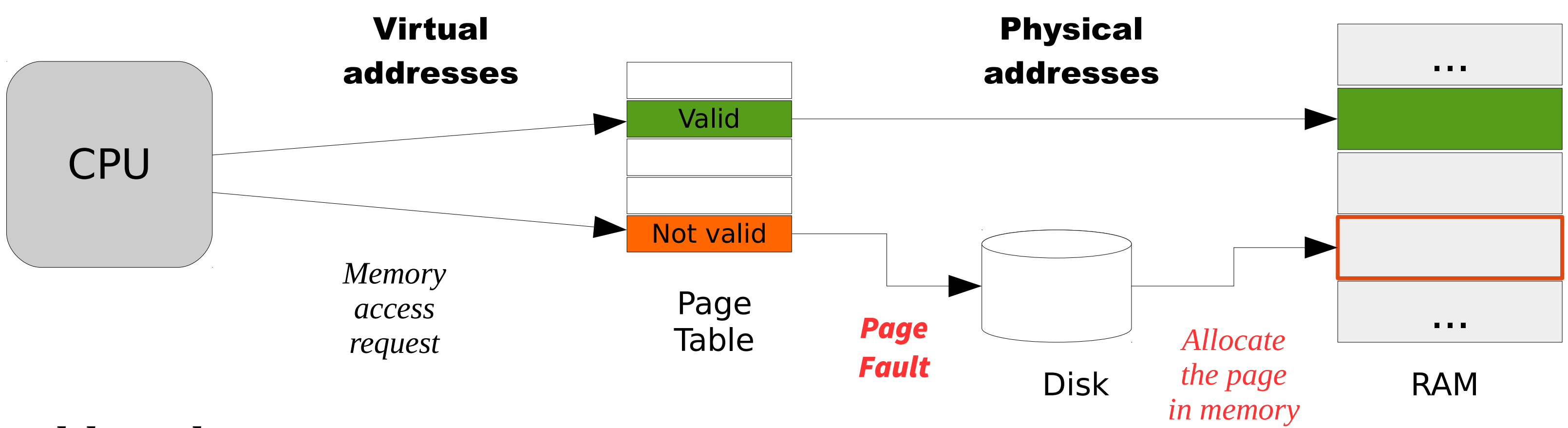
    A high priority task approaching the deadline should preempt also system calls



Predictability

Memory management

- On-demanding memory page loading leads to unpredictable delays
  - Page faults similar to cache misses with higher latencies

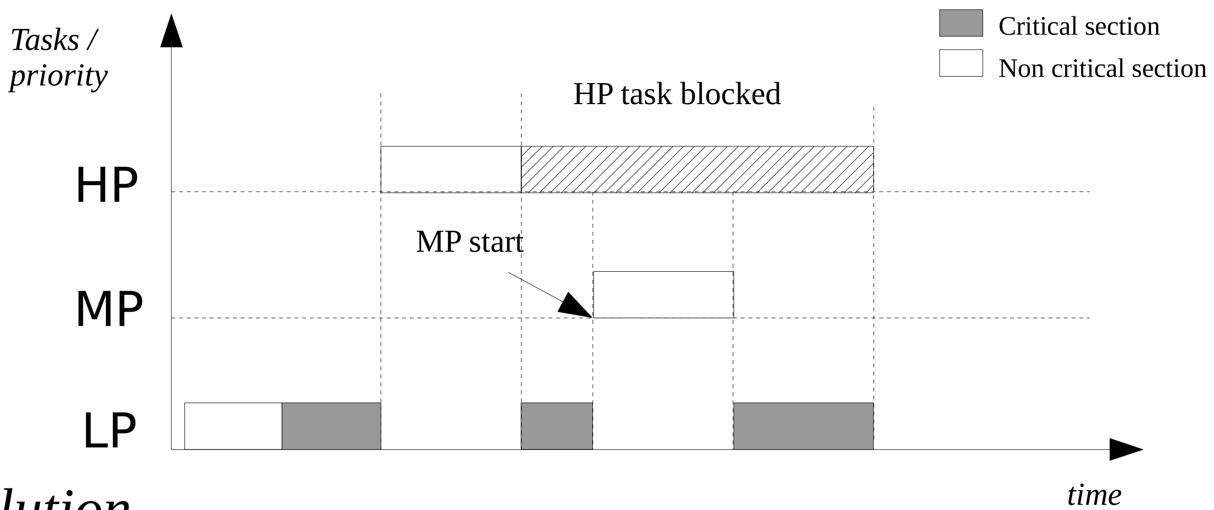


- *Possible solutions*
  - Fixed memory management schema
  - Use static allocation

Predictability

Locking mechanisms

- Consider the *Priority Inversion* problem (see slides on "Multi-tasking Synchronization...")
  - High-priority tasks blocked by low-priority tasks
  - Non deterministic delays due to locks



- *Possible solution*
  - Use locking data structure implementing suitable **Resource Access Protocols** (e.g., Priority Ceiling Protocol)

Predictability

Programming languages

- Most programming languages DO NOT provide...
  - Constructs to express timing constraints
  - Protocols for shared data accesses
  - ...
- Some constructs are source of non determinism
  - e.g. the *switch-case* prevents the performing of Worst-Case Execution Time (WCET) analysis

- *Possible solutions*
  - Adopting specific programming languages for real-time
  - Avoid dynamic data allocations
  - Avoid recursion
  - Maximum number of loops iterations must be known a priori

Real-time Operating Systems (RTOS)

Outline

- Overview
- Features
- Performance

Real-time Operating Systems (RTOS)

What is a RTOS?

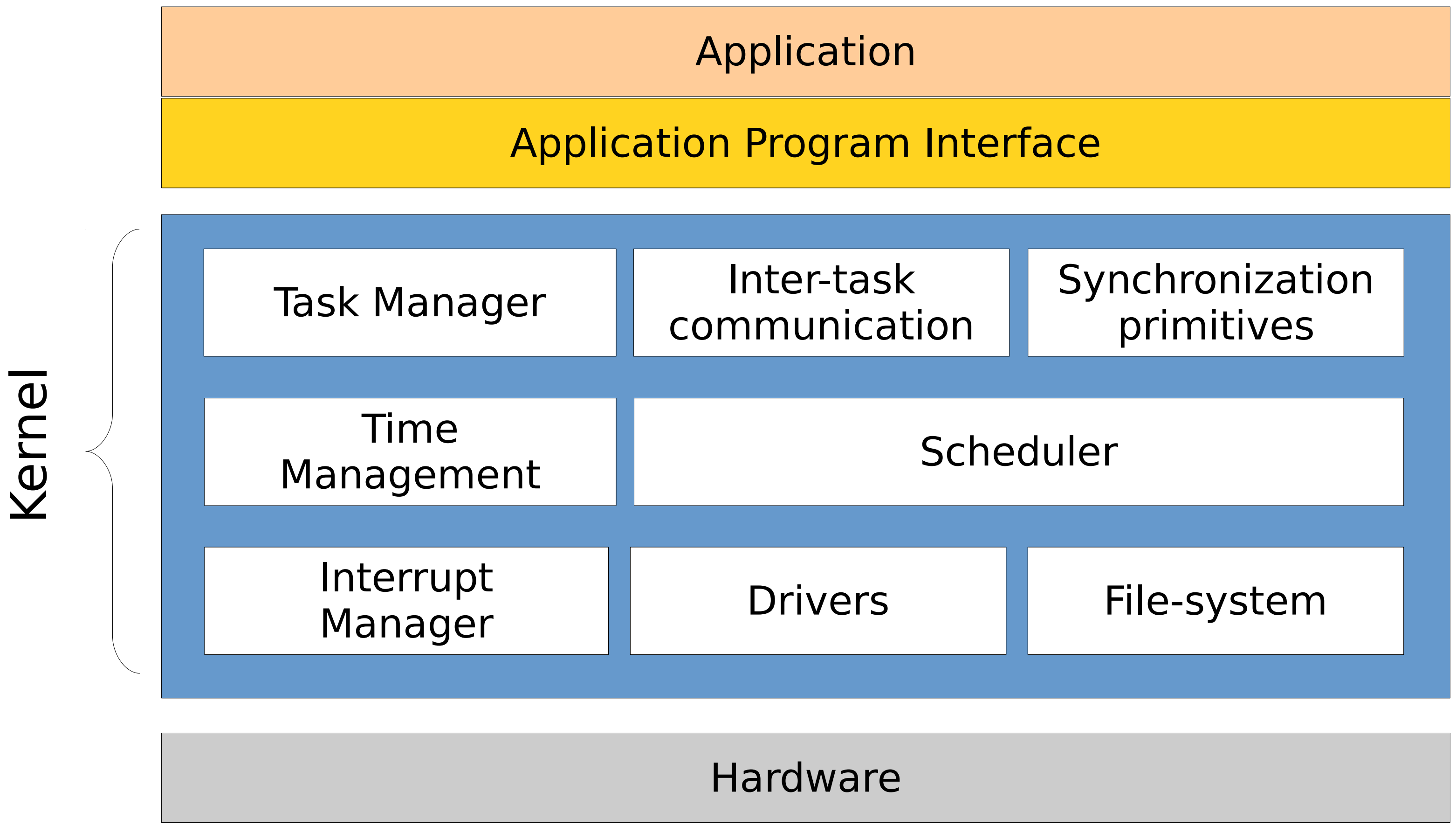
- Operating system specifically designed to address the problem of guaranteeing real-time requirements

Why to use a RTOS?

- Part of the complexity managed by the RTOS (e.g., access to peripheral registers, …)
- Faster development cycle w.r.t *bare-metal* programming
- Application timing constraints managed by real-time specific schedulers
- Part of sources of non determinism are managed by the RTOS (e.g, predictive ISR, deterministic memory management, resource access protocols, …)

Real-time Operating Systems (RTOS)

Overview



Real-time Operating Systems (RTOS)

Features

- Multi-tasking / Concurrency support
  - Task creation and execution support
  - Specific real-time scheduling policies
  - Effective inter-task synchronization and communication mechanisms

- Low-level user control
  - Fine-grained control over task priority assignment
  - Control over processes and pages that must reside in memory
  - Selection of scheduling policies
  - Selection of power management strategies (if any)
  - ...

Real-time Operating Systems (RTOS)

Features

- Reliability and robustness support
  - Manage system failures in such a way to preserve as much data and capability as possible
  - Multiple attempts to improve data consistency
  - For some real-time systems a minimal service level must be guaranteed in any case

Real-time Operating Systems (RTOS)

Typical characteristics

- Small memory footprint
  - RTOS usually requires a few KB of memory
- Predictive duration of system calls and kernel services
  - Preemptive kernel
- Responsiveness
  - Bounded *context-switch* and *interrupt management* latencies
- Determinism
  - External events must be handled such that no deadline misses occur

Real-time Operating Systems (RTOS)

Examples

- BeRTOS
- Contiki
- ChibiOS/RT
- ERIKA Enterprise
- FreeRTOS
- Mbed-rtos
- Miosix
- TinyOS
- VxWorks
- ...

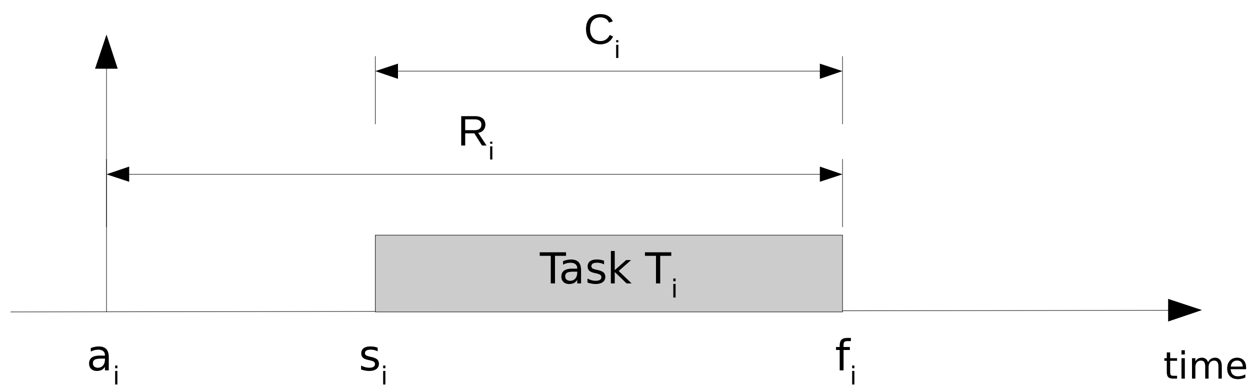
Real-time tasks

Outline

- Task parameters
- Task periodicity

Real-time tasks

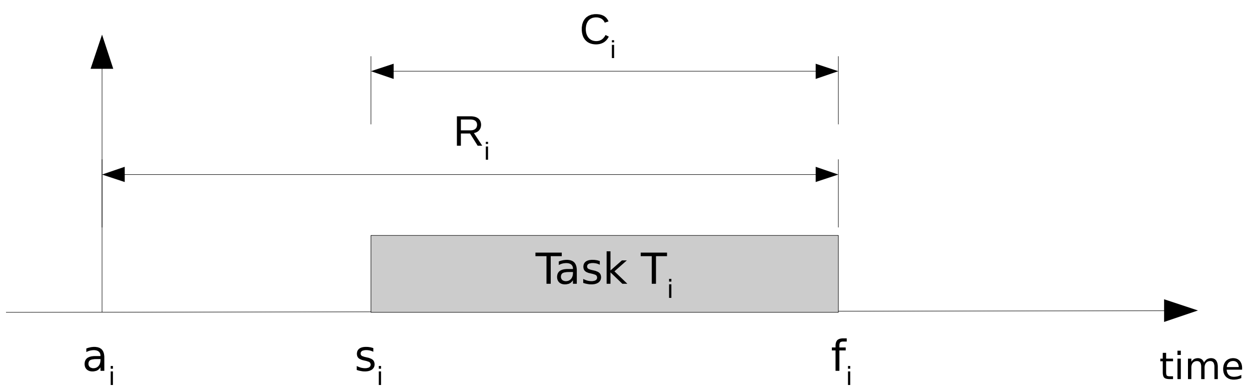
Real-time task parameters



- (See “Task scheduling” slides) …
- R_i : Response time – time between arrival time and the completion of the task
- $\boldsymbol{C_i}$: Computation time (or Burst time) – amount of time necessary to the processor to execute the task (without interruption)

Real-time tasks

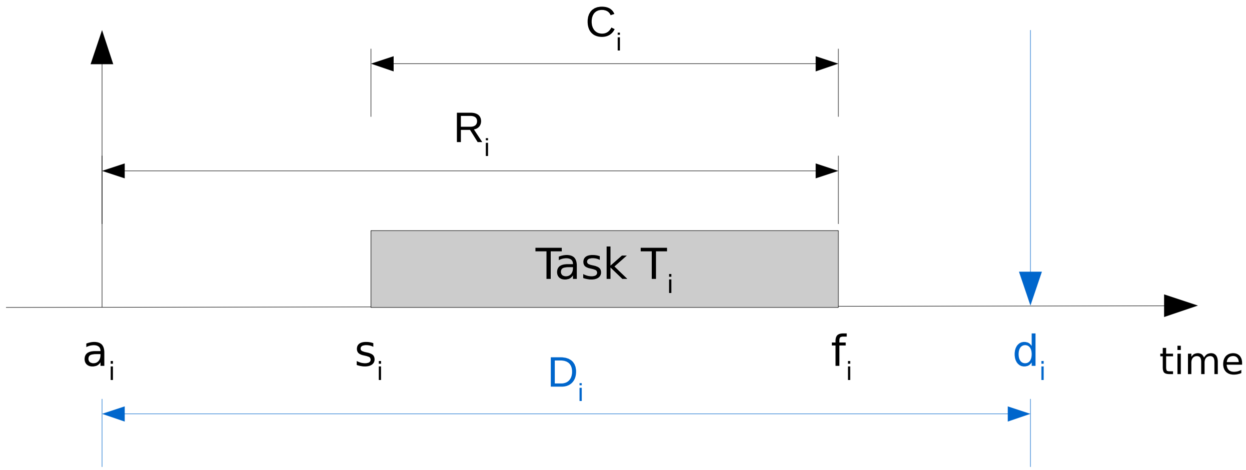
Real-time task parameters



- Real-time tasks are characterized by further parameters, related to their *timing constraints*

Real-time tasks

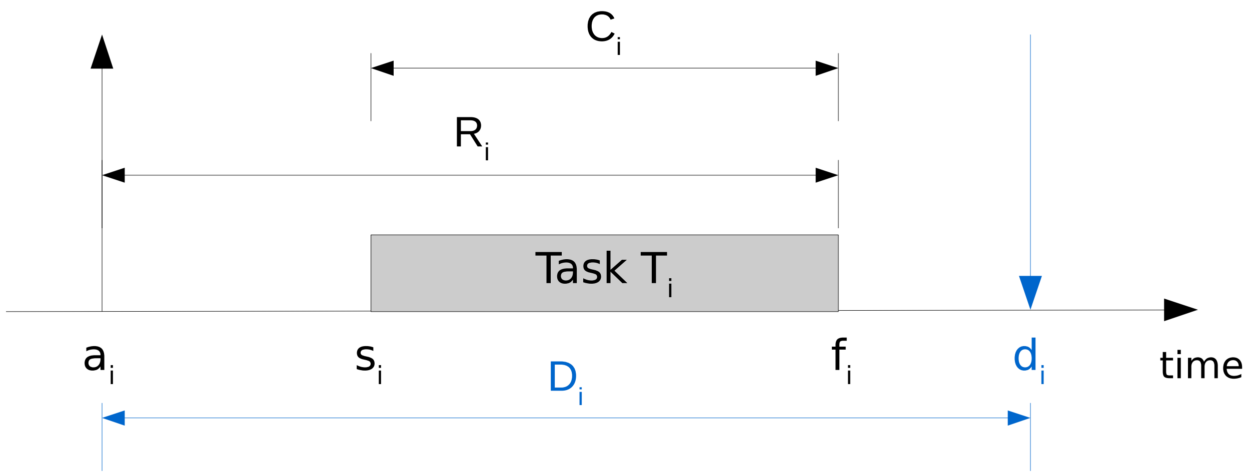
Real-time task parameters



- d_i : Absolute deadline – time before which the task should complete before violating the constraint
- D_i : Relative deadline – difference between absolute deadline and arrival time ($D_i = d_i - a_i$)

Real-time tasks

Real-time task parameters

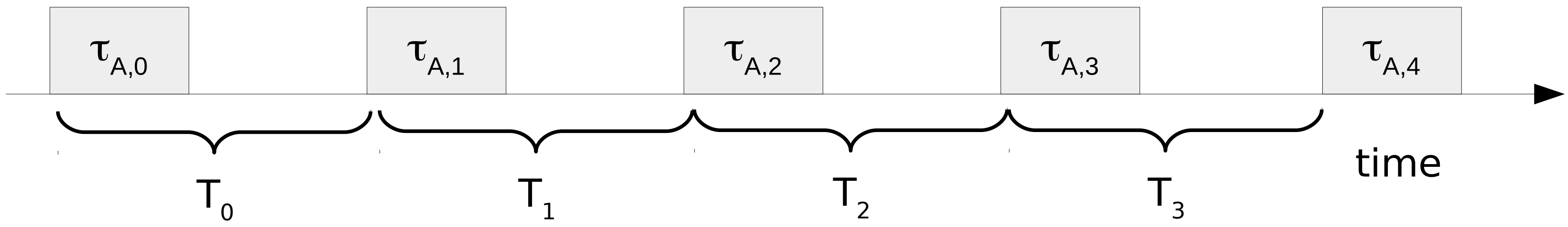


- ***L_i*** : Lateness – delay of a task completion w.r.t. its deadline
  ($L_i = f_i - d_i$)
- X_i : Laxity (or Slack time) – the maximum amount of time a task can be delayed to complete within its deadline
  ($X_i = d_i - a_i - C_i$)

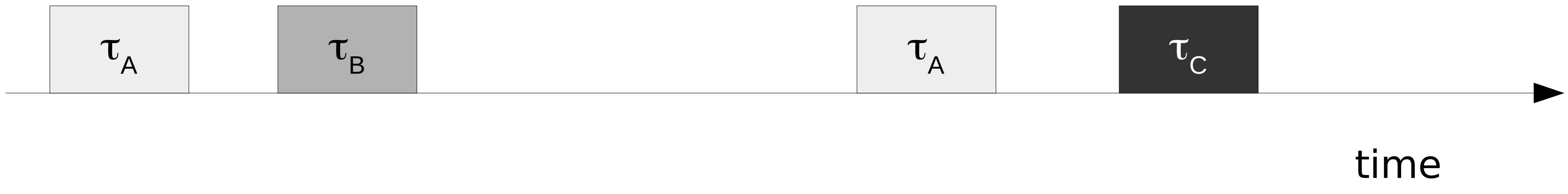
Real-time tasks

Task periodicity

- Periodic tasks



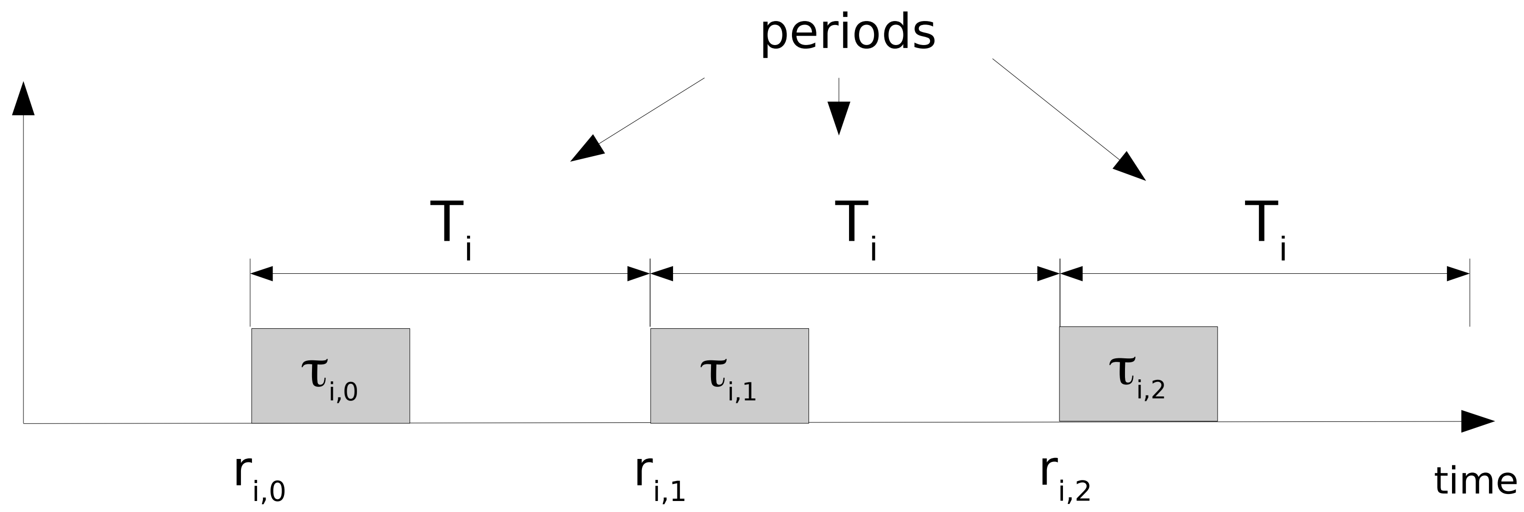
- Aperiodic tasks



Real-time tasks

Periodic tasks

- Most of the tasks of a control systems need to execute on a periodical time basis
  - Sensor data sampling
  - System monitoring
- Each periodic task τ_i is characterized by its period length $\mathbf{T_i}$

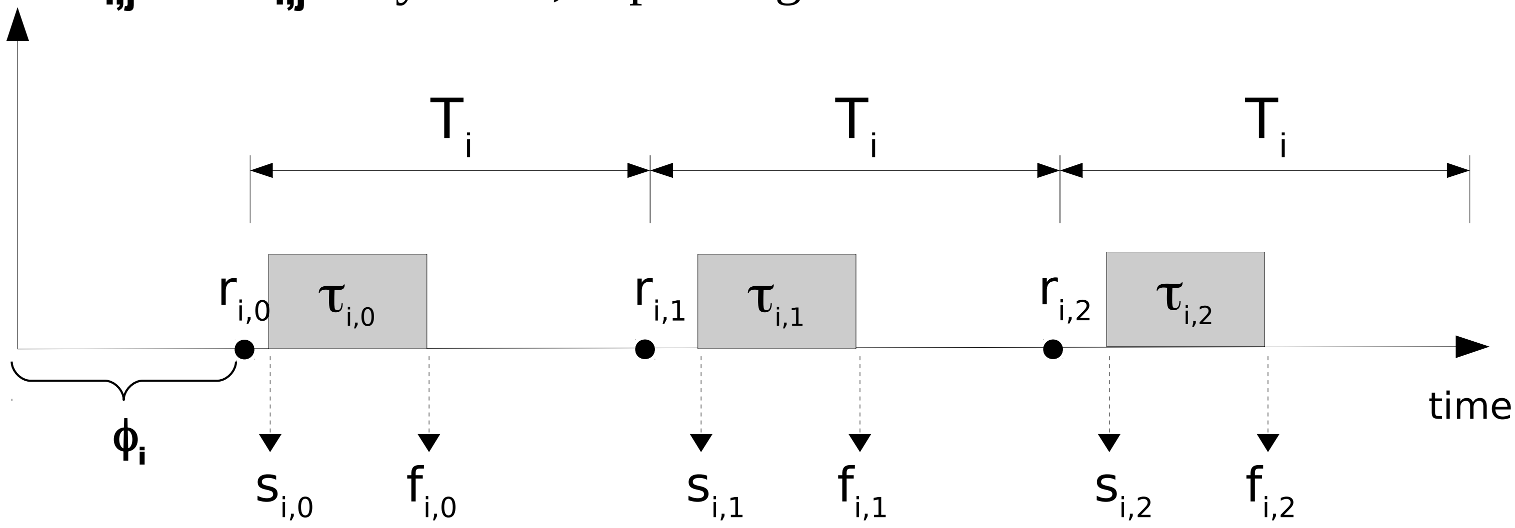


Real-time tasks

Periodic tasks

- A task τ_i is characterized by several execution instances $\tau_{i,j}$
  - $\mathbf{r_{i,j}}$ – the *arrival* (or *release*) time of the instance (when the task instance is ready to run)
  - ϕ_i – the release time of the first instance ($\phi_{i,j} = r_{i,0}$)
  - $\mathbf{s_{i,j}}$ and $\mathbf{f_{i,j}}$ are the *starting* and *finishing* times of the instances

$\mathbf{r_{i,j}}$ and $\mathbf{s_{i,j}}$ may differ, depending on the scheduler decisions

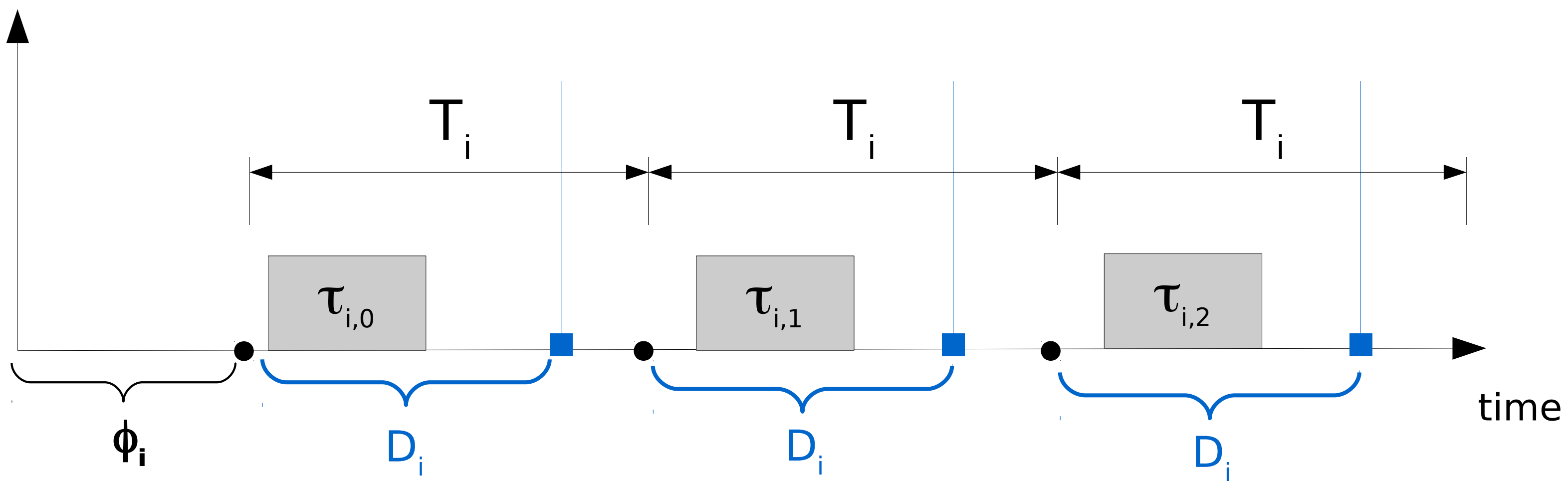


Real-time tasks

Periodic tasks

- A task τ_i is characterized by several execution instances $\tau_{i,j}$
  - **D_i** – relative deadline of the task within a period (the same for each instance)
  - **$d_{i,j}$** – absolute deadline (given a instance $\tau_{i,j}$):

$$d_{i,j}=\Phi_i+(j-1)T_i+D_i$$

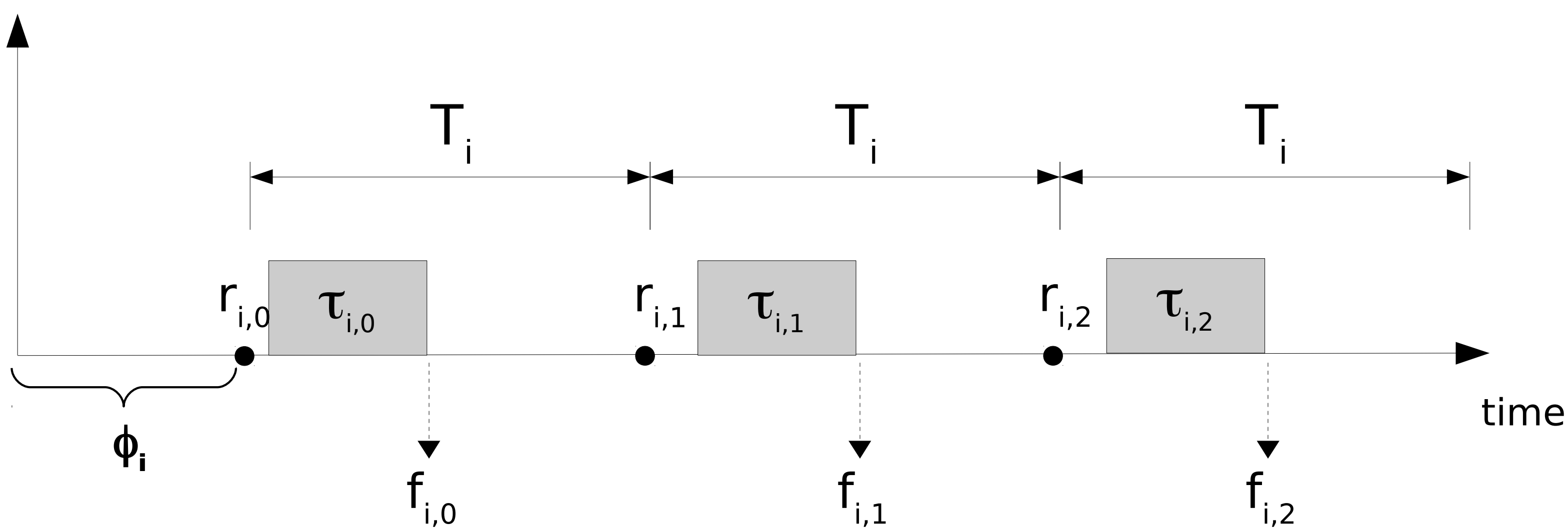


Real-time tasks

Periodic tasks

- A task instance $\tau_{i,j}$ is also called *job*
  - Job response time: $$R_{i,j} = f_{i,j} - r_{i,j}$$

- Task response time: $$R_i = \max_j R_{i,j}$$



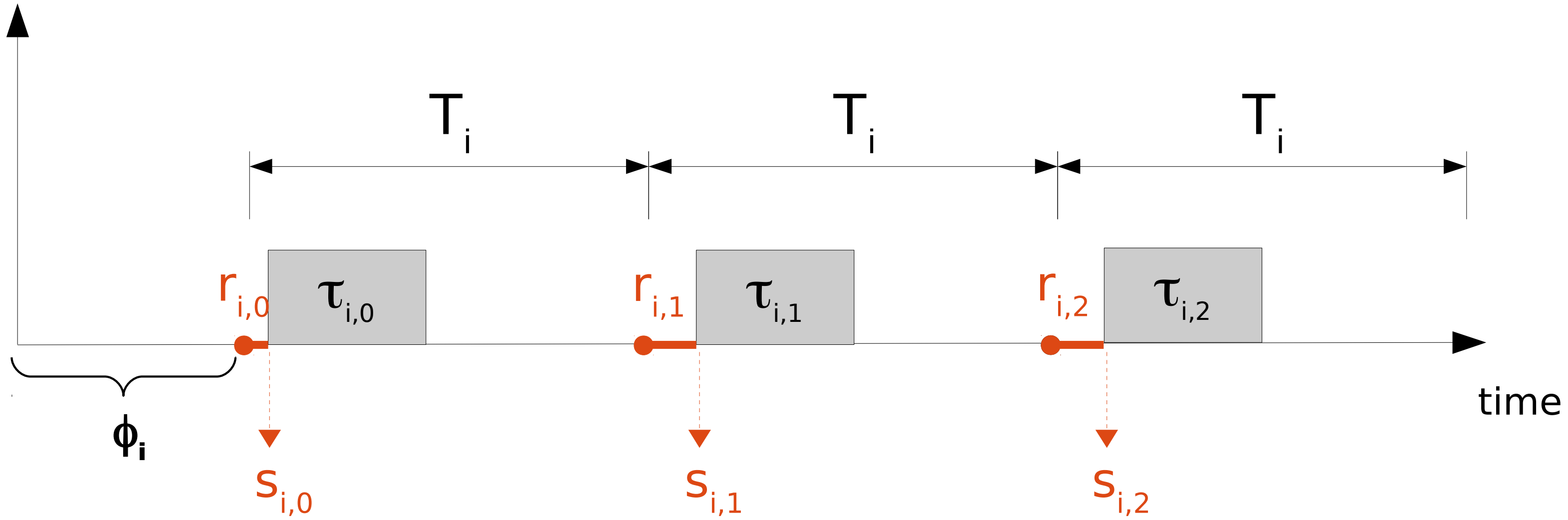
Periodic tasks

- *Relative start time jitter* – maximum deviation of start time between two consecutive instances/jobs

$$RRJ_i = \max_j \left| (s_{i,j} - r_{i,j}) - (s_{i,j-1} - r_{i,j-1}) \right|$$

- *Absolute start time jitter* – maximum deviation... among all instances

$$ARJ_i = \max_j (s_{i,j} - r_{i,j}) - \min_j (s_{i,j} - r_{i,j})$$



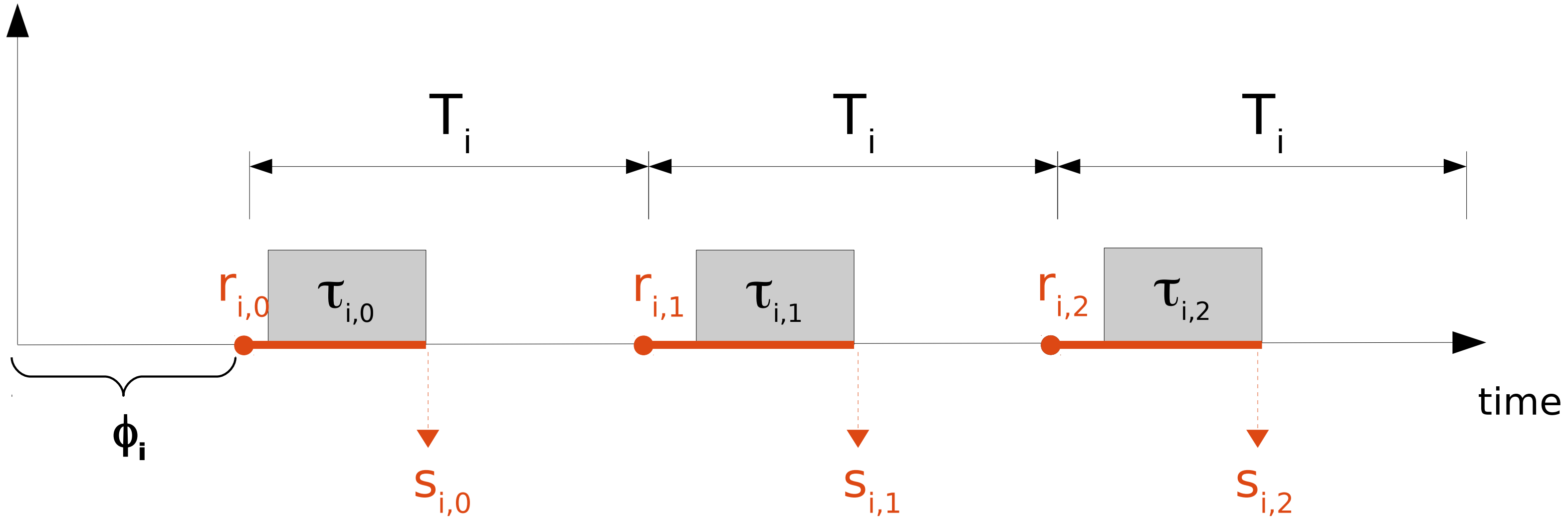
Periodic tasks

- *Relative finishing time jitter* – maximum deviation of finishing time between two consecutive instances/jobs

$$RFJ_i = \max_j \left| (f_{i,j} - r_{i,j}) - (f_{i,j-1} - r_{i,j-1}) \right|$$

- *Absolute finishing time jitter* – maximum deviation... among all instances

$$AFJ_i = \max_j (f_{i,j} - r_{i,j}) - \min_j (f_{i,j} - r_{i,j})$$



Real-time tasks

Periodic tasks

- When the system includes several concurrent periodic tasks with timing constraints, the scheduler provided by the operating system has to guarantee that...
  - Each task instance is activated (task is ready) at the proper rate
  - The task instance is completed within the deadline D_i

- This is what a real-time scheduler is intended for

Outline

- Definitions and Assumptions
- Algorithms

Real-time scheduling

Definitions

- Task scheduling plays a key role in real-time systems
- Real-time scheduling means that the algorithms must take into account the *timing constraints* of each task
  - A couple of definitions are worth to be mentioned...
- A schedule is *feasible*, if all the tasks of the given set can be completed without timing constraints violations
- A set of tasks is *schedulable* if there exists at least one algorithm capable of producing a feasible scheduling

Real-time scheduling

Assumptions

- [1] The instances of a periodic task τ_i are regularly activated at constant rate
- [2] All instances of a periodic task τ_i have the worst-case execution time C_i
- [3] All instances of a periodic task τ_i have the same deadline D_i equal to the period ($D_i = T_i$)
- [4] All the tasks in a set Γ are independent (no precedence relations and no resource constraints)
- [5] Tasks cannot suspend themselves (e.g., on I/O operations)
- [6] Release time coincides with the arrival of the task
- [7] Kernel overheads (e.g., context switch) are assumed to be zero

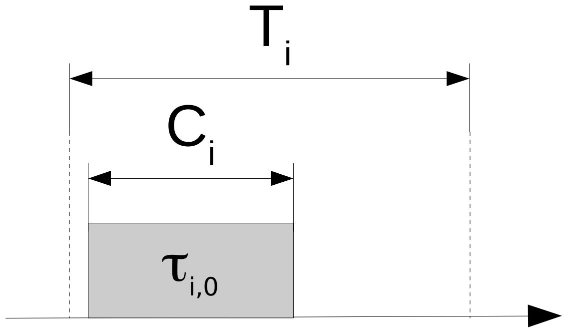
Real-time scheduling

Processor utilization factor

- The fraction of the processor time spent in the execution of the tasks from a set Γ

$$U = \sum_{i=0}^{n} \frac{C_i}{T_i}$$

- A measure of the computational load of the processor



Real-time scheduling

Processor utilization factor

- ***U*** is useful to check the feasibility of a schedule
- There exists an **upper bound value (*U_{ub}*)** above which a schedule is not feasible
- The upper bound depends on the task set Γ and the scheduling algorithm: $\boldsymbol{U_{ub}(\Gamma, A)}$
- Therefore, a task set Γ_i is schedulable with A iff

$$U_i \leq U_{ub}(\Gamma_i, A)$$

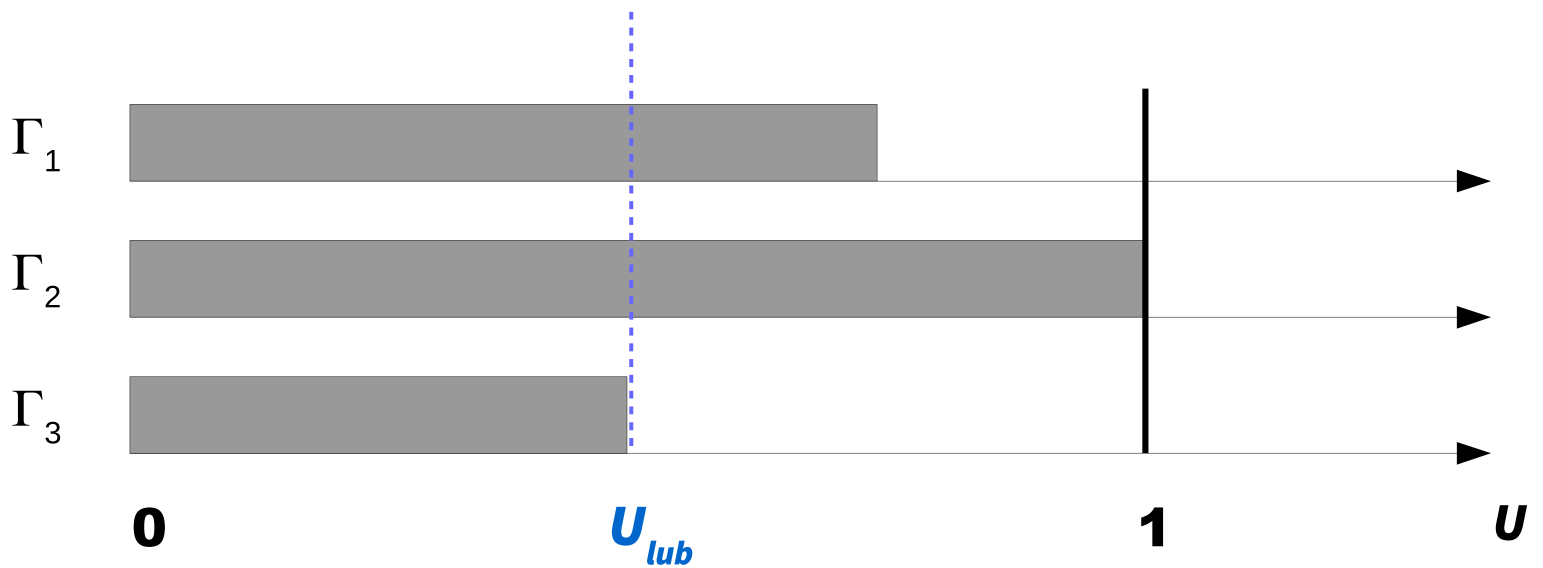
- Given the task set Γ the fully utilization of the processor is achieved for $\boldsymbol{U = U_{ub}(\Gamma, A)}$

Real-time scheduling

Processor utilization factor

- Given an algorithm A, we can also define the **least upper bound *$U_{lub}(A)$*** as the minimum utilization factor over all the task sets that fully utilize the processor

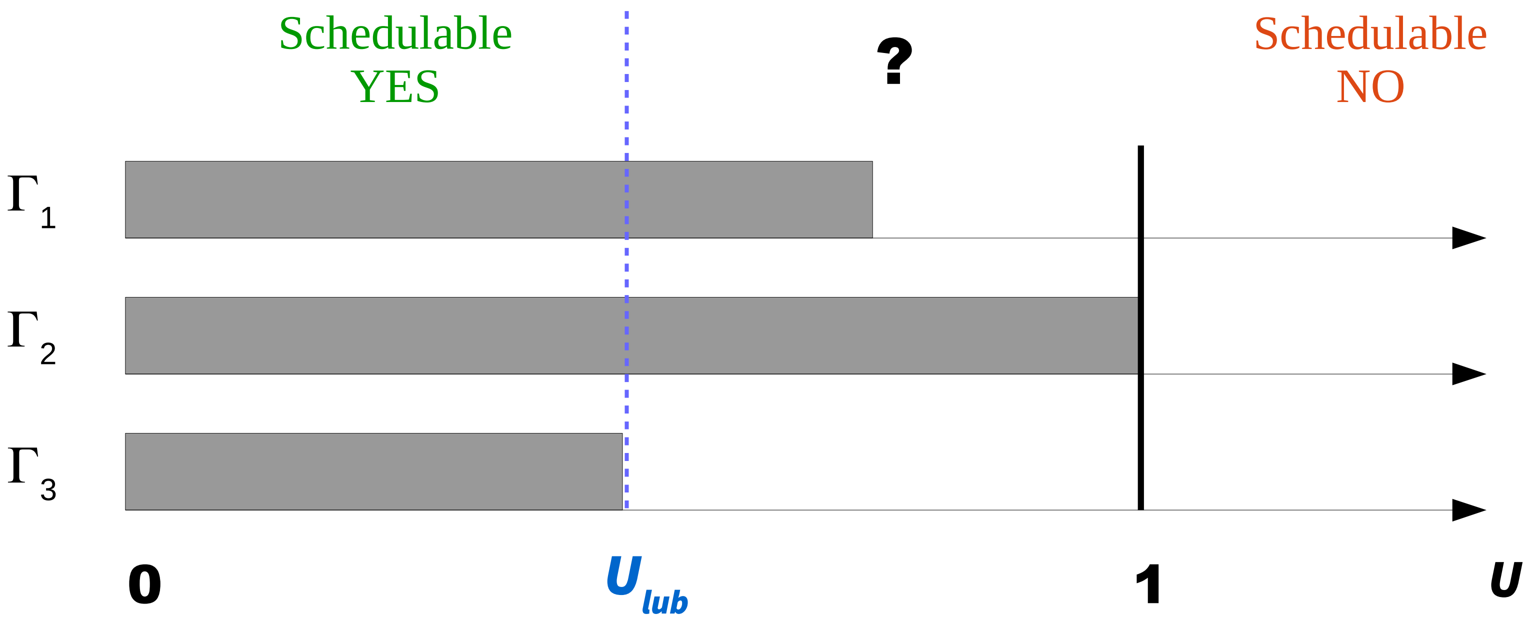
$$U_{lub}(A) = \min_{\Gamma} U_{ub}(\Gamma, A)$$



Real-time scheduling

Processor utilization factor

- For each task set Γ_i
  - If $U_i < U_{lub}$ $\Rightarrow \Gamma_i$ schedulable by the algorithm
  - If $U_{lub} < U_i \leq 1$ $\Rightarrow \Gamma_i$ schedulable if periods suitably related
  - If $U_i > 1$ $\Rightarrow \Gamma_i$ not schedulable



Real-time scheduling

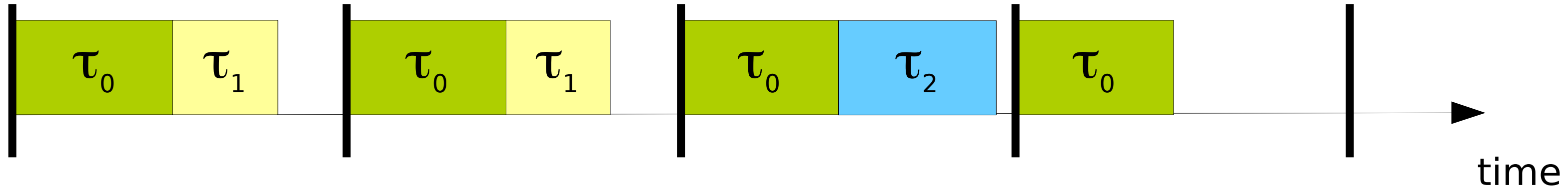
Scheduling algorithms

- Timeline Scheduling (TS)
- Rate Monotonic (RM)
- Earlieast Deadline First (EDF)
- Deadline Monotonic (DM)

Real-time scheduling

Timeline Scheduling

- Also known as *Cyclic Executive* scheduling
- Very used in periodic tasks scheduling in military and traffic control systems
- The requirement is to execute a task at a given rate
- The temporal axis is divided into equal length slots
  - One or more tasks allocated per slot
  - A time synchronizes the activation of the tasks at the beginning of the slot



Real-time scheduling

Timeline Scheduling

- *Example*

Task	Rate (Hz)	Period (ms)
A	40	25
B	20	50
C	10	100

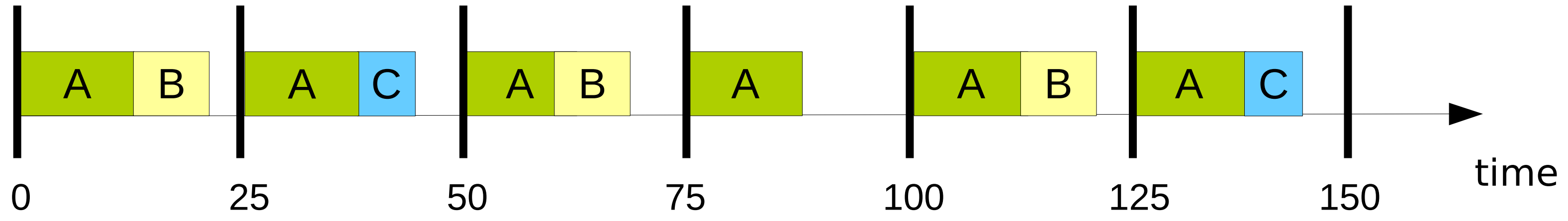
- The optimal slot length would be T_A = 25ms (the Greatest Common Divisor)
  - Task A will execute in every slot
  - Task B will execute every two slots
  - Task C will execute every four slots

Timeline Scheduling

- *Example*

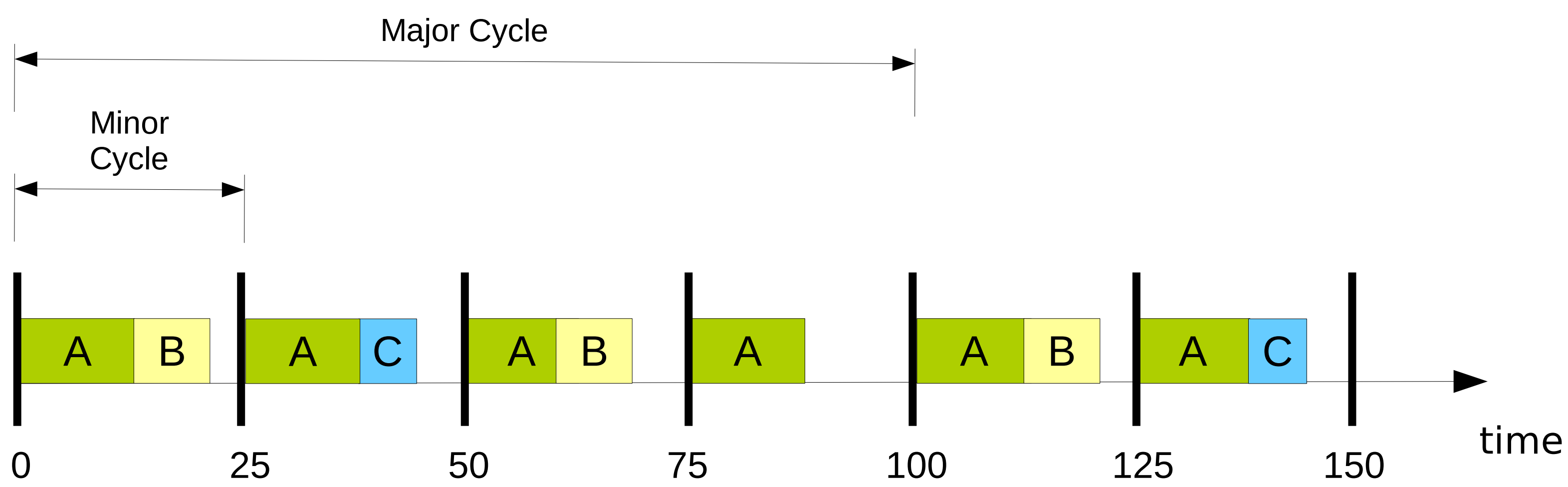
Task	Rate (Hz)	Period (ms)
A	40	25
B	20	50
C	10	100

- Possible solution:



Timeline Scheduling

- The duration of a time slot is also called minor cycle
  - *Greatest Common Divisor (GCD)* of the activation periods
- A major cycle instead is the time interval after which a schedule repeats itself (also called *hyper-period*)
  - *Least Common Multiple (lcm)* of the activation periods



Real-time scheduling

Timeline Scheduling

- Schedulability
  - Verify that the sum of the worst-execution times (WCET) of the tasks within each time slot is less than or equal to the Minor Cycle
  - Remember assumption [2]: $WCET_i = C_i$

$$\begin{Bmatrix} C_A + C_B \leq 25\,ms \\ C_A + C_C \leq 25\,ms \end{Bmatrix}$$

Real-time scheduling

Timeline Scheduling

- PROs
  - *Simple implementation*

    Timer interrupt set to the minor cycle length and a main program calling the tasks in the given order
  - *Very low overhead*

    The task execution order is not decided by a scheduler but by the program
  - *No jitter*

    Always the same sequence of task executions

    Start times and response times not subject to noticeable variations

Timeline Scheduling

- CONs
  - *Not robust*

    In overloaded conditions, a deadline miss causes a domino effect, breaking the schedule
  - *Sensitivity to application changes*

    Updates of computation time or activation rate may invalidate the current schedule
  - *Hard to handle aperiodic activities*

    May require a change in the task execution sequence

Real-time scheduling

Rate Monotonic

- Assign a priority to the task according to the **activation (or release) rate**
  - Higher rates (shorter periods) → higher priorities
- *Fixed-priority* assignment
  - Period length is constant periods
  - Priority assigned when task is ready and no longer changed
- *Preemptive* (intrinsically)
  - A new task preempts the currently running one if characterized by a shorter period

- Optimal algorithm among all the fixed-priority based scheduling algorithms
  - Liu and Layland (1973)

Rate Monotonic

- Schedulability
  - It can be demonstrated that for RM the least upper bound utilization factor converges to

$$U_{lub} = n\left(2^{1/n} - 1\right)$$

n = 2 : $U_{lub} = 0.828$

$$\lim_{n \to \infty} U_{lub}(n) = \ln(2) = 0.693$$

  - The task set is schedulable if

$$U_i \leq U_{lub}$$

Rate Monotonic

- Example

Task	Period T	WCET C
A	10	2
B	60	10
C	30	5

- Scheduler will be executed every minor cycle
  - Minor cycle = $\mathrm{GCD}(10, 60, 30) = 10$
- The schedule length is the given by the major cycle
  - Major cycle = $\mathrm{lcm}(10, 60, 30) = 60$

Rate Monotonic

- Example

Task	Period T	WCET C
A	10	2
B	60	10
C	30	5

- Schedulability
  - ➔ $U_{lub} = 3(2^{1/3} - 1) = 0.7798$
  - ➔ $U_i = (2/10) + (10/60) + (5/30) = 0.533$
  - ➔ $U_i \leq U_{lub}$ **OK!**

Rate Monotonic

- Example

Task	Period T	WCET C
A	10	2
B	60	10
C	30	5

- Reminding that the shorter the period, the higher the priority…
  - → $\mathbf{p_A > p_C > p_B}$

Rate Monotonic

- Example
  - Tasks are scheduled according to the priority: $\tau_A > \tau_C < \tau_B$
  - *t=10* : τ_B is preempted by τ_A
  - *t=12* : τ_A terminates and τ_B can be resumed to complete its execution

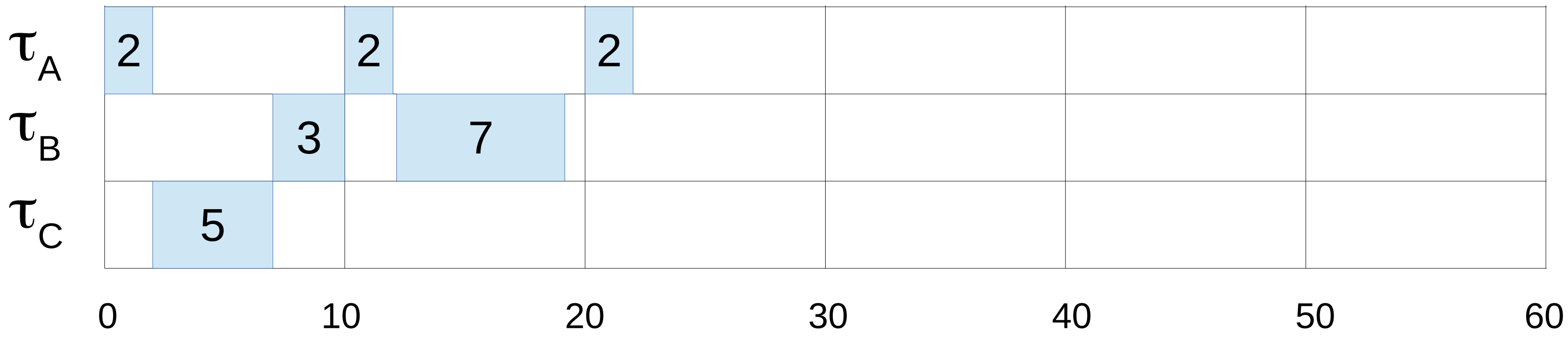
Task	Period T	WCET C
A	10	2
B	60	10
C	30	5

Real-time scheduling

Rate Monotonic

- Example
  - *t=20* : τ_A must be executed again
  - τ_B and τ_C has already completed the execution in the respective periods

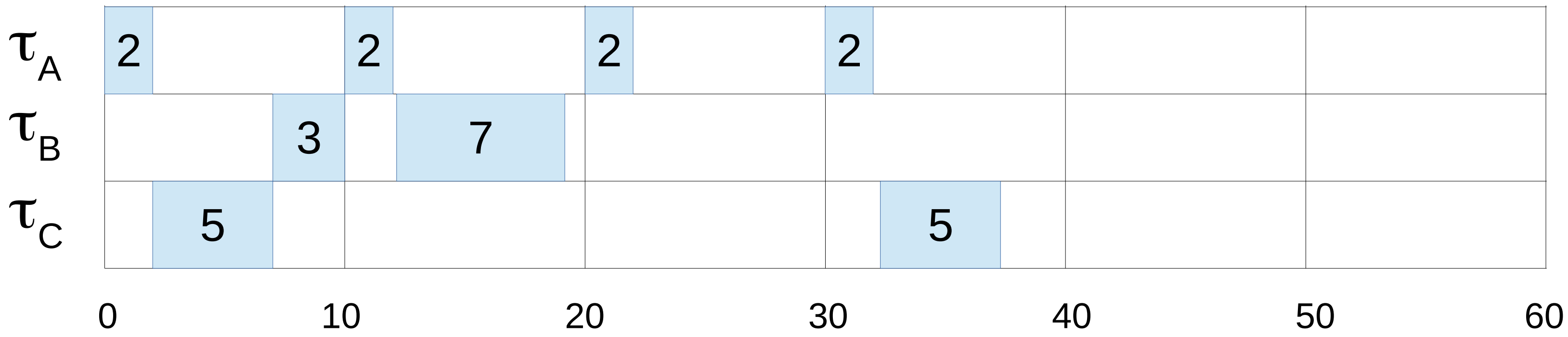
Task	Period T	WCET C
A	10	2
B	60	10
C	30	5



Rate Monotonic

- Example
  - *t=30* : τ_A must be executed again
  - *t=32* : τ_C must be executed again

Task	Period T	WCET C
A	10	2
B	60	10
C	30	5

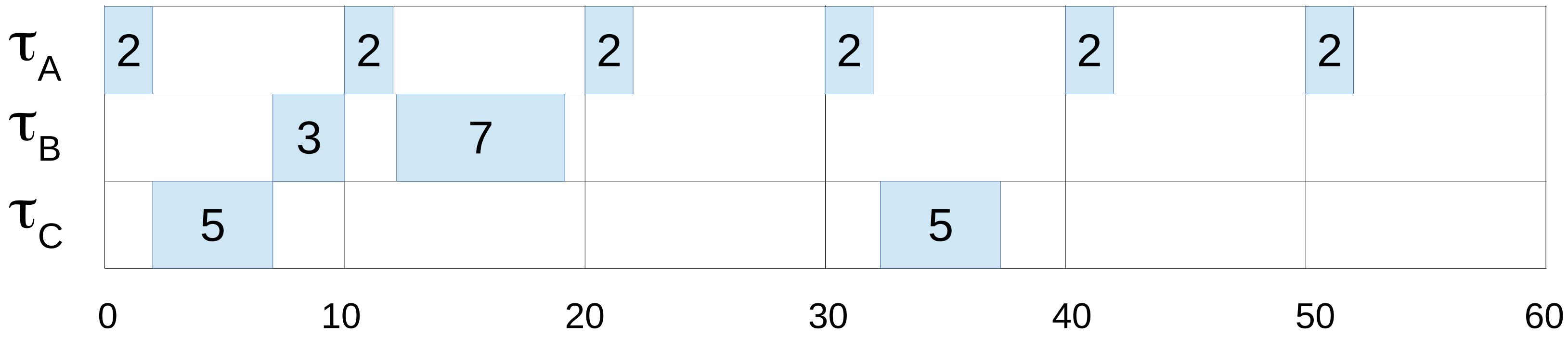


Real-time scheduling

Rate Monotonic

- Example
  - τ_A will be executed again every 10 time units
  - After 60 time units (mayor cycle) the schedule will repeat itself

Task	Period T	WCET C
A	10	2
B	60	10
C	30	5



Rate Monotonic

- Properties
  - Easy to implement algorithm
  - Lower overhead w.r.t EDF

    Fixed priorities → no need of updating
  - Stability is easier to achieve w.r.t other scheduler, in case of system problems(e.g. due to transient errors),
  - Actually exploited in industrial solutions
  - Versions with relaxed assumptions available (we will not shown them)

Earliest Deadline First (EDF)

- *Dynamic-priority* assignment scheduler
- Task to execute is selected according to the **absolute deadline**
  - Higher priorities to earlier deadline tasks

$$d_{i,j} = \Phi_i + (j-1)T_i + D_i$$

- *Preemptive*
  - Currently running task preempted when a periodic instance with earlier deadline is activated
- Agnostic w.r.t to the task periodicity
  - Both periodic or aperiodic tasks can be scheduled

Real-time scheduling

Earliest Deadline First (EDF)

- EDF is optimal (for single core processors)

Theorem

If a set of jobs is schedulable by an algorithm A,
then it is schedulable by EDF

Real-time scheduling

Earliest Deadline First (EDF)

- Schedulability
  - A set of periodic tasks is schedulable with EDF iff

$$U = \sum_{i=0}^{n} \frac{C_i}{T_i} \leqslant 1$$

i.e., the processor utilization is not greater than 100%

Real-time scheduling

Earliest Deadline First (EDF)

- Example
  - Release times and deadlines coincide (T = D)

Task	Relative deadline D	WCET C
A	4	1
B	6	2
C	12	4

- Schedulability

  $U = (1/4) + (2/6) + (4/12) = \mathbf{0.9167 < 1} \rightarrow$ **YES!**

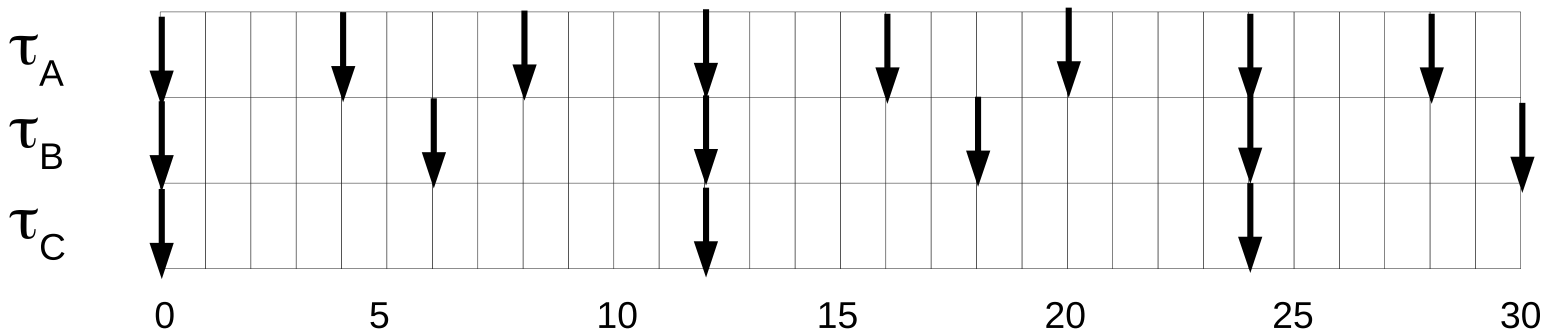
  - (no deadline misses expected)

Real-time scheduling

Earliest Deadline First (EDF)

- Example
  - Let's annotate release times and deadlines
  - $lcm(4,6,12)$ = **12**

Task	Relative deadline D	WCET C
A	4	1
B	6	2
C	12	4

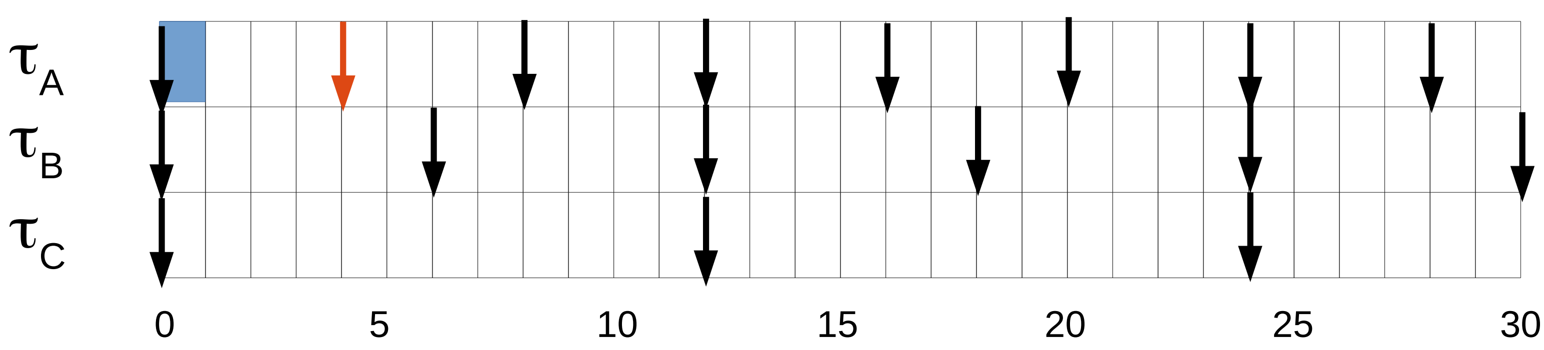


Real-time scheduling

Earliest Deadline First (EDF)

- Example
  - **t=0**: Closest deadline is $d_A \rightarrow$ schedule τ_A

Task	Relative deadline D	WCET C
A	4	1
B	6	2
C	12	4

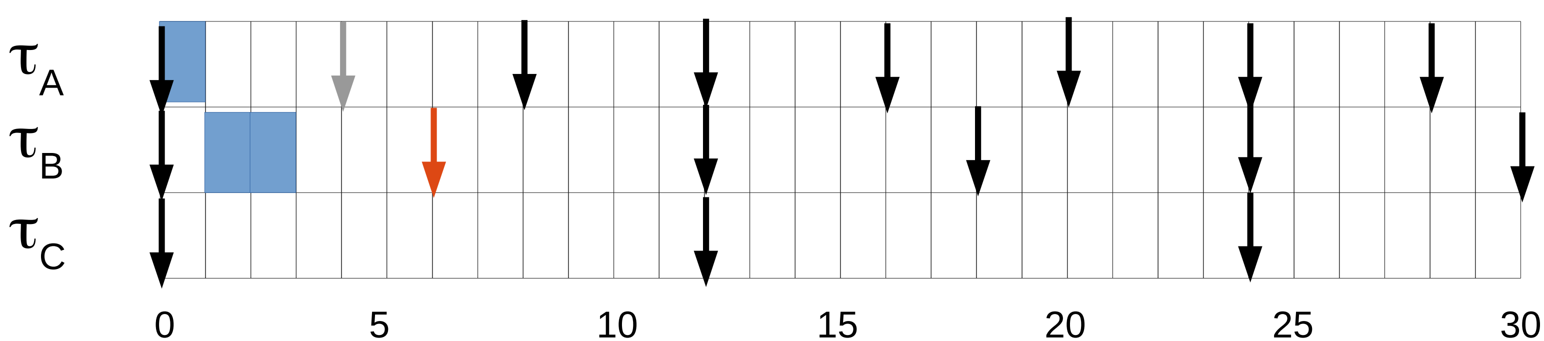


Real-time scheduling

Earliest Deadline First (EDF)

- Example
  - **t=1** : τ_A already executed
  - Closest deadline $d_B \rightarrow$ schedule τ_B

Task	Relative deadline D	WCET C
A	4	1
B	6	2
C	12	4

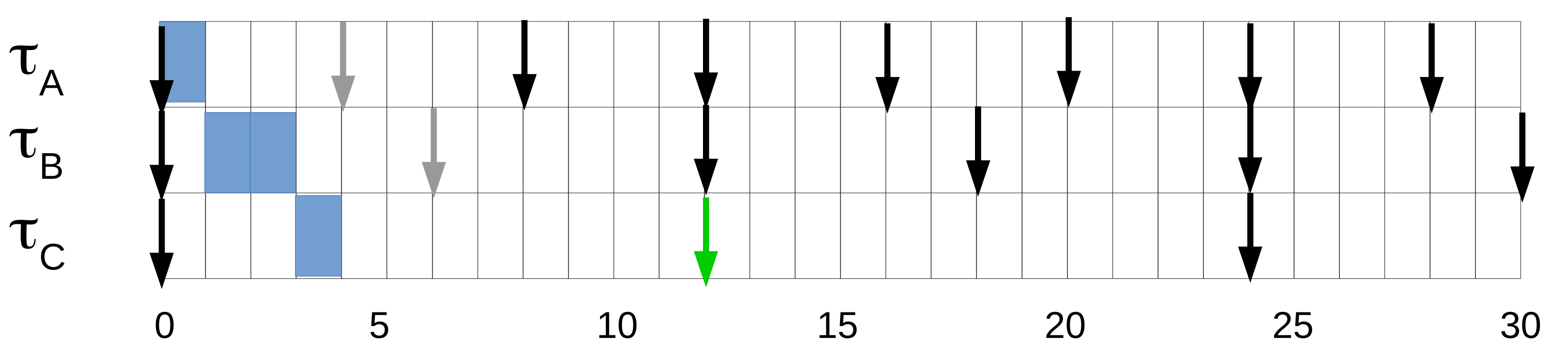


Real-time scheduling

Earliest Deadline First (EDF)

- Example
  - **t=3** :
    τ_C is the only ready task → Schedule τ_c

Task	Relative deadline D	WCET C
A	4	1
B	6	2
C	12	4

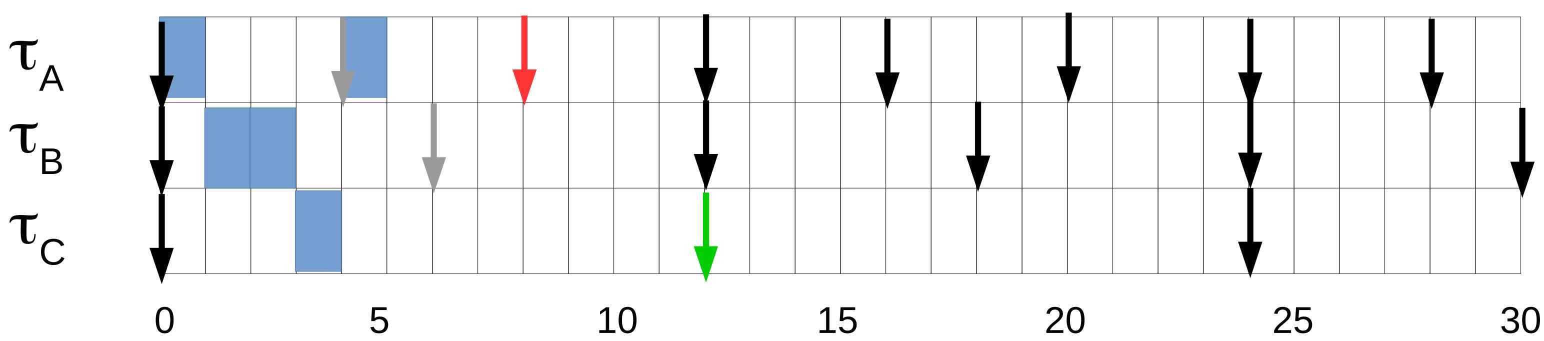


Real-time scheduling

Earliest Deadline First (EDF)

- Example
  - **t=4** : τ_C preempted by τ_A

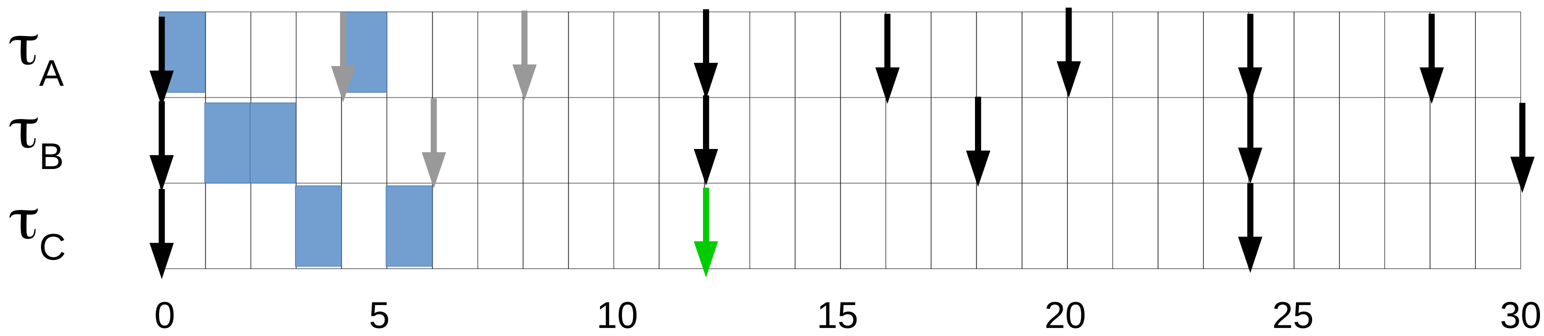
Task	Relative deadline D	WCET C
A	4	1
B	6	2
C	12	4



Earliest Deadline First (EDF)

- Example
  - **t=5** :
    τ_C is resumed as it is the only ready task

Task	Relative deadline D	WCET C
A	4	1
B	6	2
C	12	4

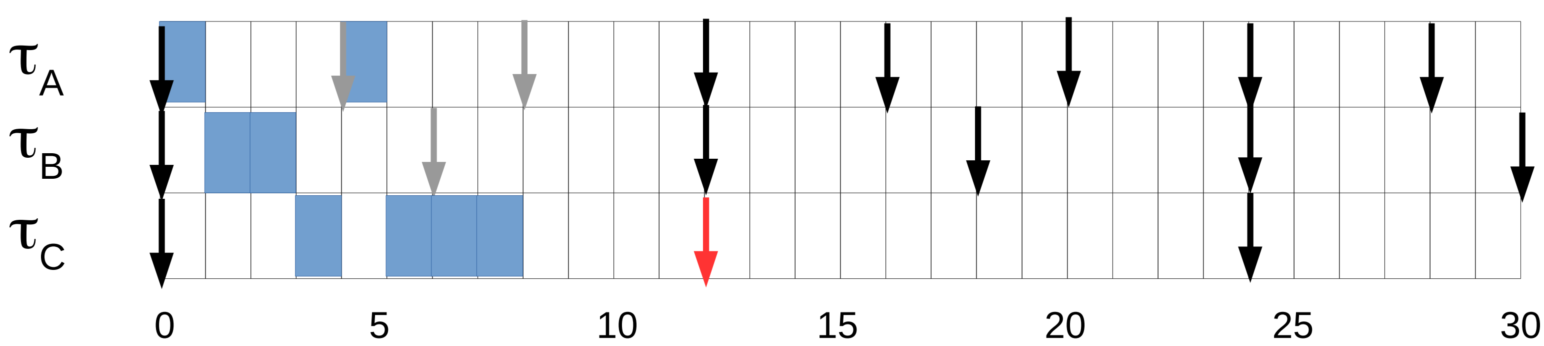


Real-time scheduling

Earliest Deadline First (EDF)

- Example
  - **t=6** :
    τ_C is kept in execution, since τ_B has the same distance from deadline

Task	Relative deadline D	WCET C
A	4	1
B	6	2
C	12	4

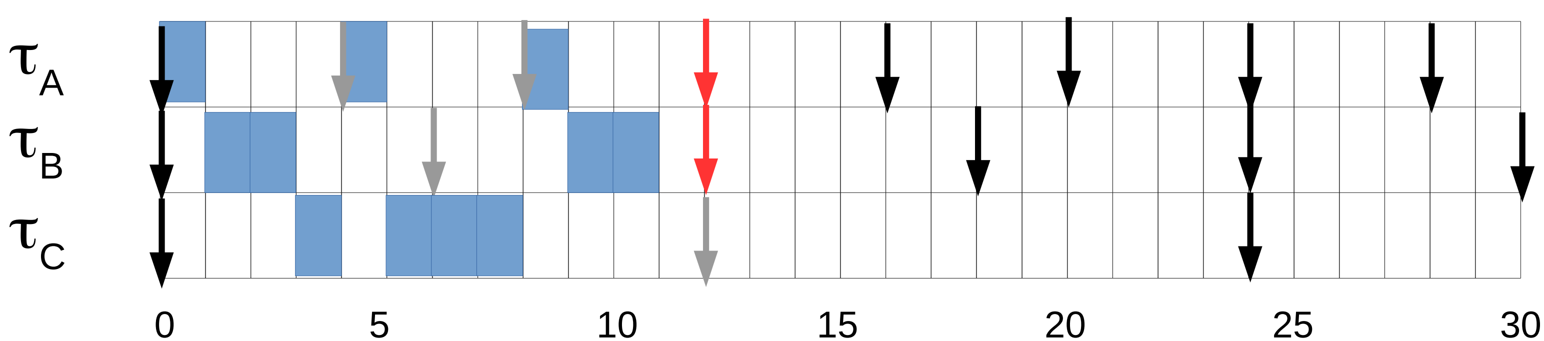


Real-time scheduling

Earliest Deadline First (EDF)

- Example
  - **t=8** :
    τ_A and τ_B are active and scheduled in sequence since $d_A = d_B$

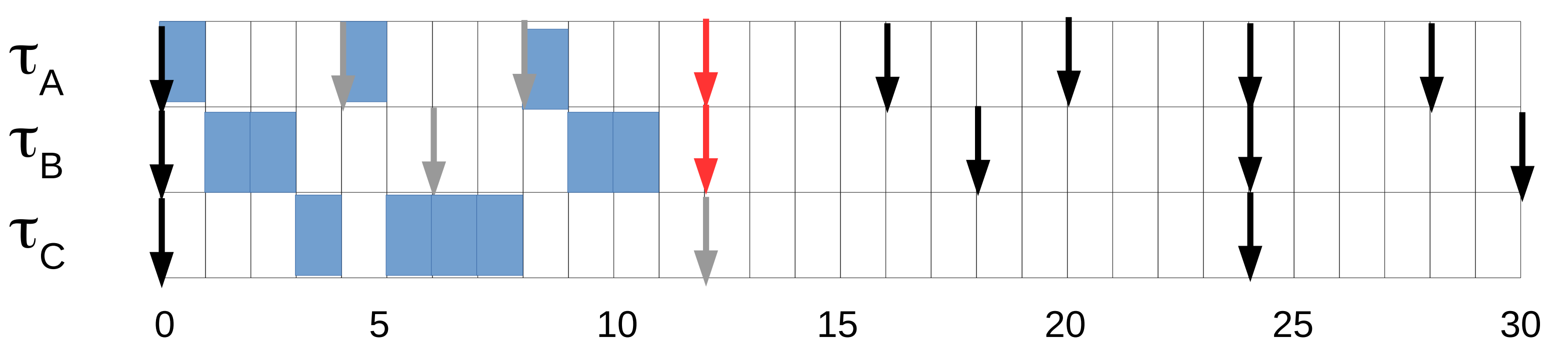
Task	Relative deadline D	WCET C
A	4	1
B	6	2
C	12	4



Earliest Deadline First (EDF)

- Example
  - **t=8** :
    τ_A and τ_B are active and scheduled in sequence since $d_A = d_B$

Task	Relative deadline D	WCET C
A	4	1
B	6	2
C	12	4

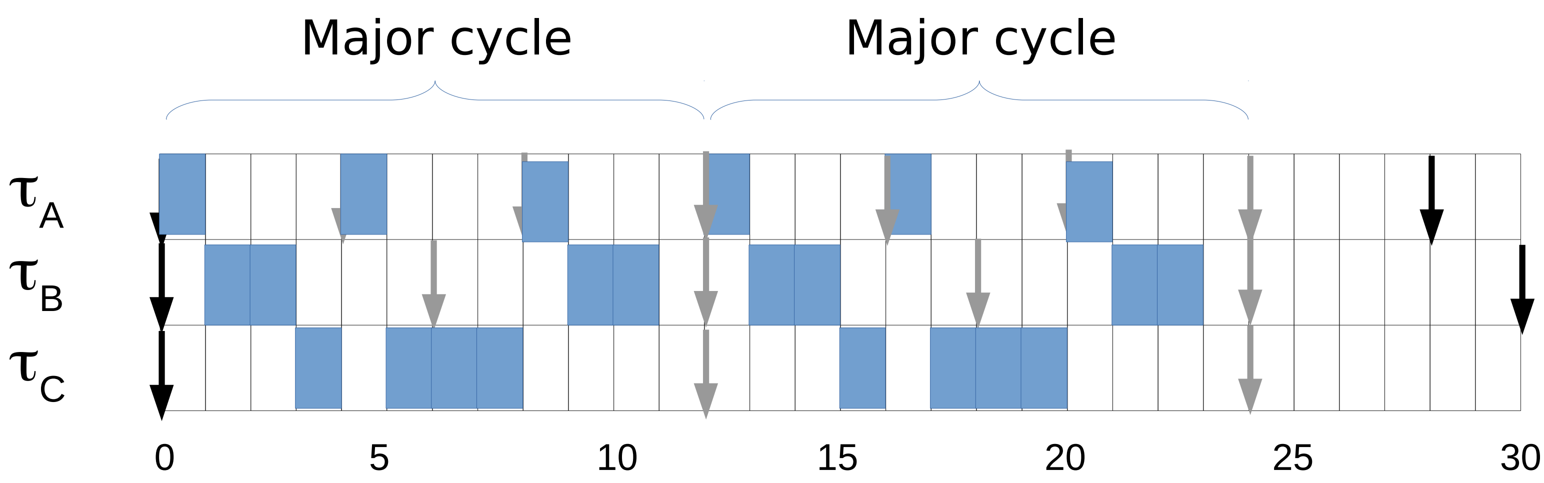


Real-time scheduling

Earliest Deadline First (EDF)

- Example
  - lcm(4,6,12) = 12
  - As expected, the schedule repeat itself after t=12

Task	Relative deadline D	WCET C
A	4	1
B	6	2
C	12	4



Real-time scheduling

Earliest Deadline First (EDF)

- With EDF we can achieve a higher processor utilization w.r.t. Rate Monotonic
- Less preemptions w.r.t Rate Monotonic, due to dynamic priorities
- But also… higher overhead w.r.t. Rate Monotonic
  - Task priority is dynamic → it may change online
- Hard and expensive to implement on micro-controller based embedded systems
  - Require very frequent invocations of the scheduler
  - Hard to predict overload conditions of the system
- Seldom adopted in industrial applications

Deadline Monotonic (DM)

- Proposed a generalization of Rate Monotonic by Audsley (1990)
  - Assumption [3] ($D_i = T_i$) is relaxed
- *Fixed priority* assignment
  - Relative deadline is constant
  - Priority inversely proportional to relative deadline Di
- *Preemptive*
  - Currently running task preempted when a new task with shorter relative deadline arrives
- DM is optimal
  - If a task set is schedulable by some fixed priority static algorithm, it is also schedulable by DM
  - If a task set is NOT schedulable by DM, it is not schedulable by any other fixed priority static algorithm

Real-time scheduling

Deadline Monotonic (DM)

- Schedulability
  - A “pessimistic” test derived from Rate Monotonic (processor utilization overestimated)

$$\sum_{i=0}^{n} \frac{C_i}{D_i} \leqslant n\left(2^{(1/n)} - 1\right)$$

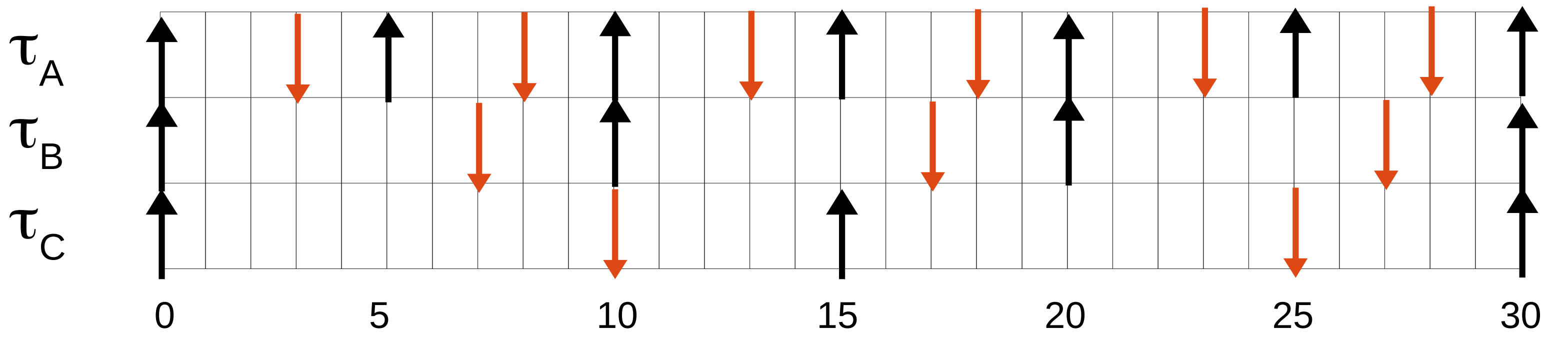
In DM must consider relative deadlines instead of activation periods

  - A more efficient (*sufficient and necessary*) test has been proposed by Audsley (1993) called Response Time Analysis

Deadline Monotonic (DM)

- Example
  - Assumption [3] relaxed
  - lcm(5,10,15) = 30
  - Let's annotate release times and deadlines

Task	Period T	Relative deadline D	WCET C
A	5	3	1
B	10	7	3
C	15	10	5

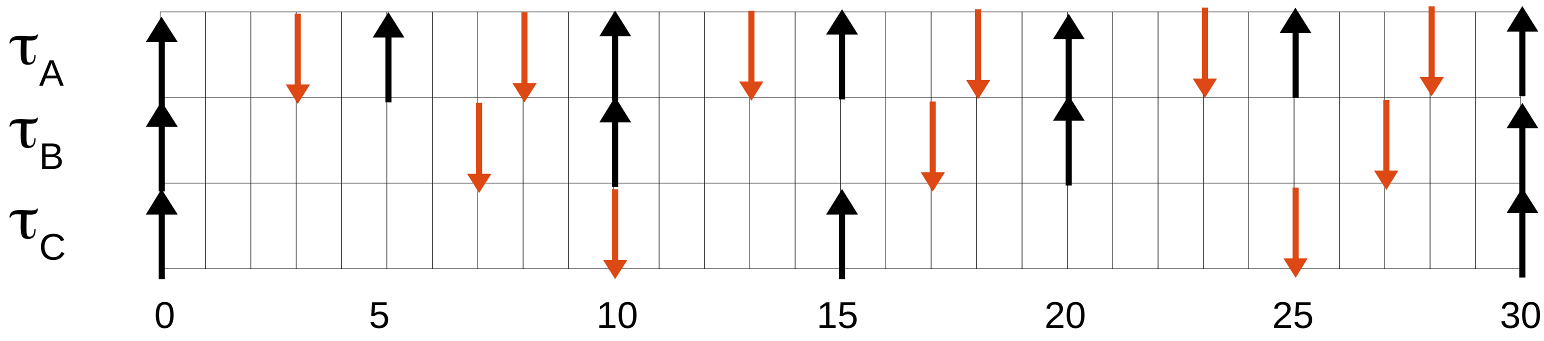


Deadline Monotonic (DM)

- Example
  - Fixed priorities assignment (based on D_i):

    $p_A > p_B > p_C$

Task	Period T	Relative deadline D	WCET C
A	5	3	1
B	10	7	3
C	15	10	5

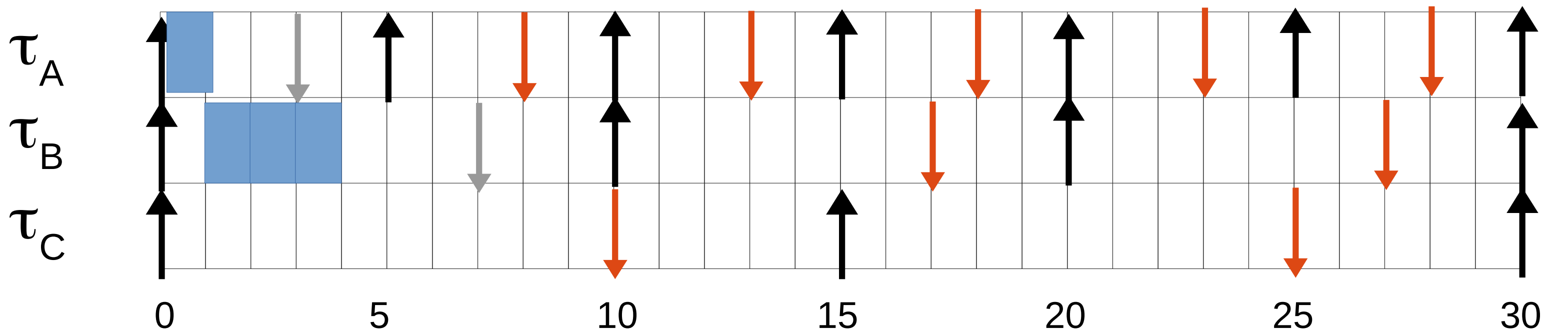


Deadline Monotonic (DM)

- Example
  - Priorities:

    $p_A > p_B > p_C$
  - t=0 : schedule τ_A
  - t=1 : schedule τ_B

Task	Period T	Relative deadline D	WCET C
A	5	3	1
B	10	7	3
C	15	10	5

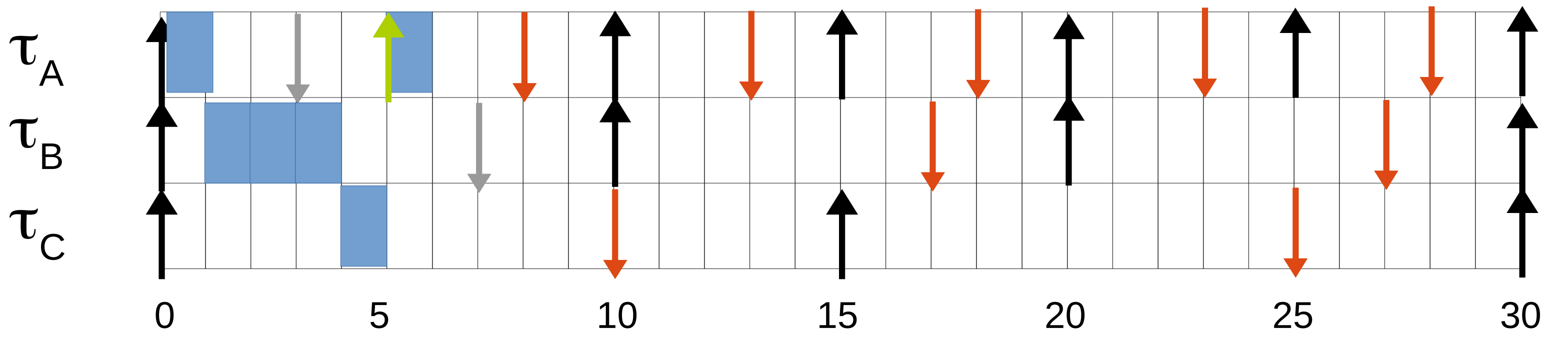


Deadline Monotonic (DM)

- Example
  - Priorities:

    $p_A > p_B > p_C$
  - t=4 : schedule τ_C
  - t=5 : τ_C is preempted by the activation of τ_A

Task	Period T	Relative deadline D	WCET C
A	5	3	1
B	10	7	3
C	15	10	5

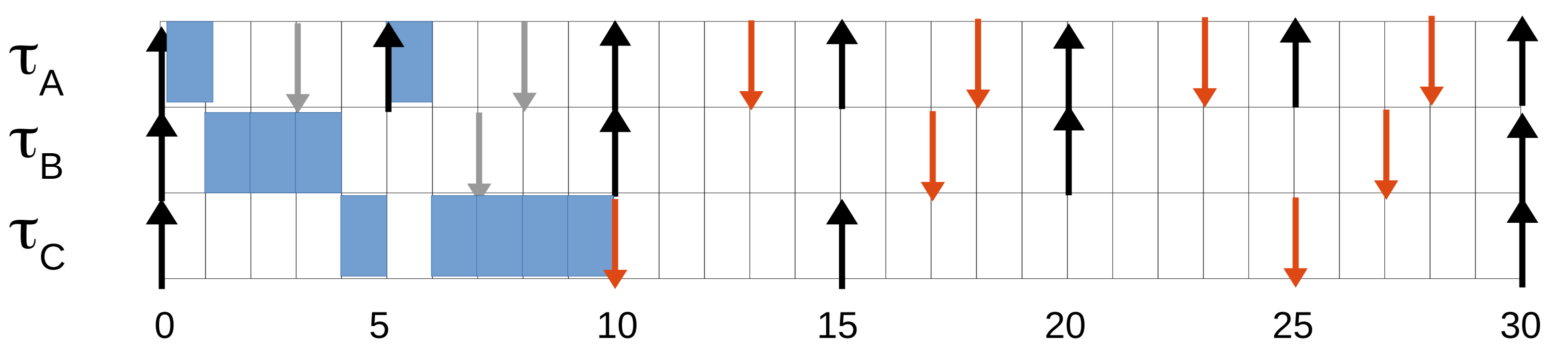


Deadline Monotonic (DM)

- Example
  - Priorities:

    $p_A > p_B > p_C$
  - $t=6$: schedule τ_C which terminates the instance execution

Task	Period T	Relative deadline D	WCET C
A	5	3	1
B	10	7	3
C	15	10	5

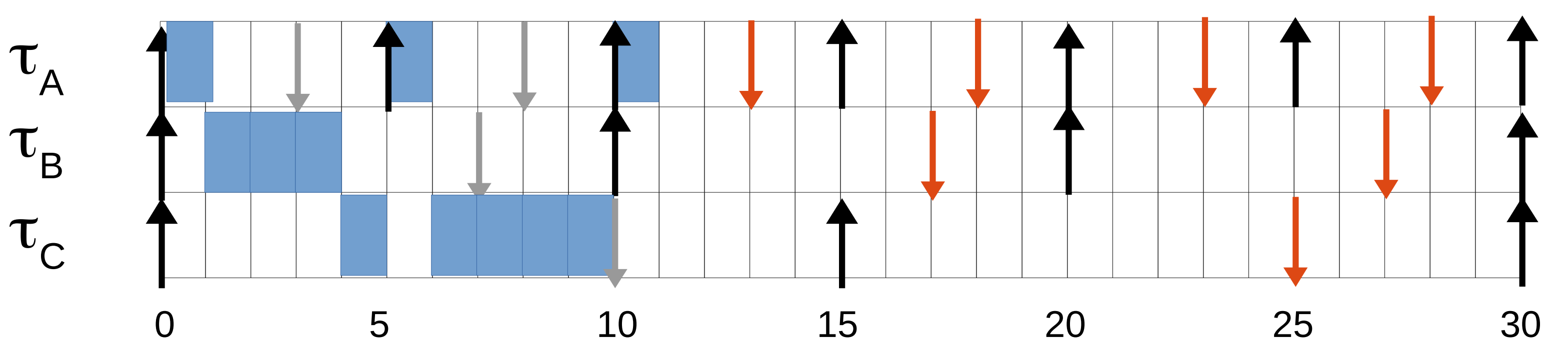


Deadline Monotonic (DM)

- Example
  - Priorities:

    $p_A > p_B > p_C$
  - t=10 : τ_A and τ_B activated
  - τ_A scheduled due to higher priority

Task	Period T	Relative deadline D	WCET C
A	5	3	1
B	10	7	3
C	15	10	5



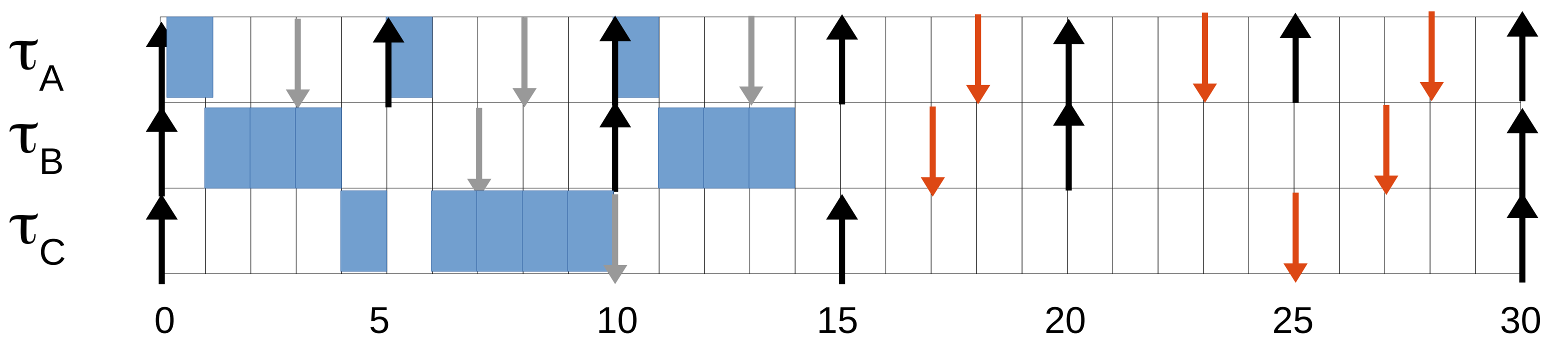
Deadline Monotonic (DM)

- Example
  - Priorities:

    $p_A > p_B > p_C$

  - $t=11$: schedule τ_B
  - $t=14$: no active instances → nothing to schedule

Task	Period T	Relative deadline D	WCET C
A	5	3	1
B	10	7	3
C	15	10	5



Real-time scheduling

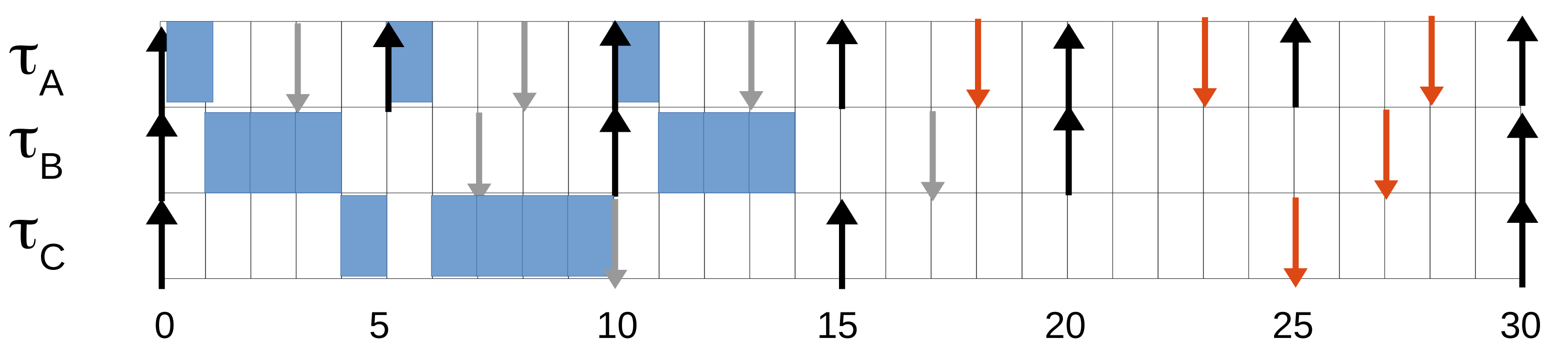
Deadline Monotonic (DM)

- Example
  - Priorities:

    $p_A > p_B > p_C$

  - t=14 : no active instances → nothing to schedule

Task	Period T	Relative deadline D	WCET C
A	5	3	1
B	10	7	3
C	15	10	5



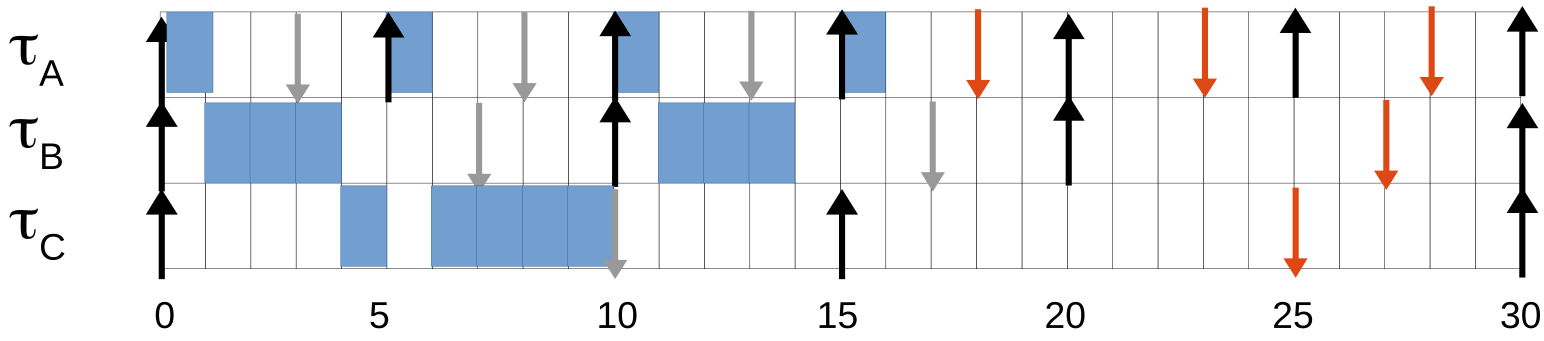
Deadline Monotonic (DM)

- Example
  - Priorities:

    $p_A > p_B > p_C$

  - $t=15$: τ_A and τ_C activated
  - τ_A scheduled due to higher priority

Task	Period T	Relative deadline D	WCET C
A	5	3	1
B	10	7	3
C	15	10	5



Real-time scheduling

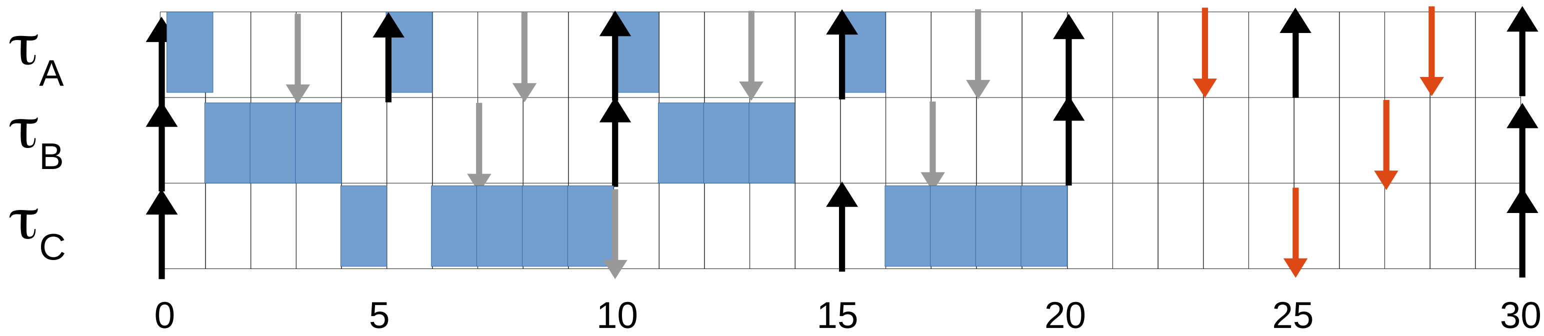
Deadline Monotonic (DM)

- Example
  - Priorities:

    $p_A > p_B > p_C$

  - $t=16$: τ_C scheduled since τ_B already executed

Task	Period T	Relative deadline D	WCET C
A	5	3	1
B	10	7	3
C	15	10	5



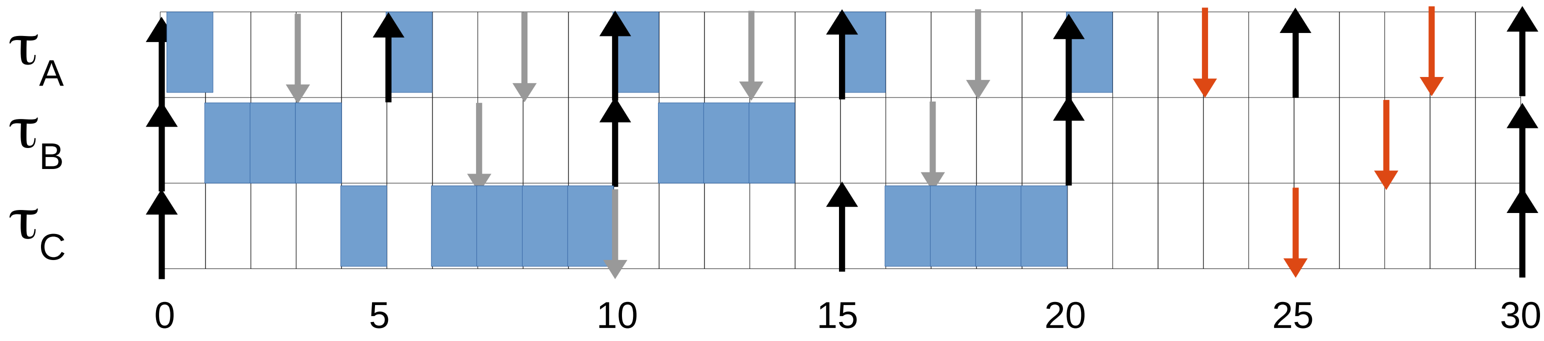
Deadline Monotonic (DM)

- Example
  - Priorities:

    $p_A > p_B > p_C$

  - $t=20$: τ_C preempted by τ_A

Task	Period T	Relative deadline D	WCET C
A	5	3	1
B	10	7	3
C	15	10	5



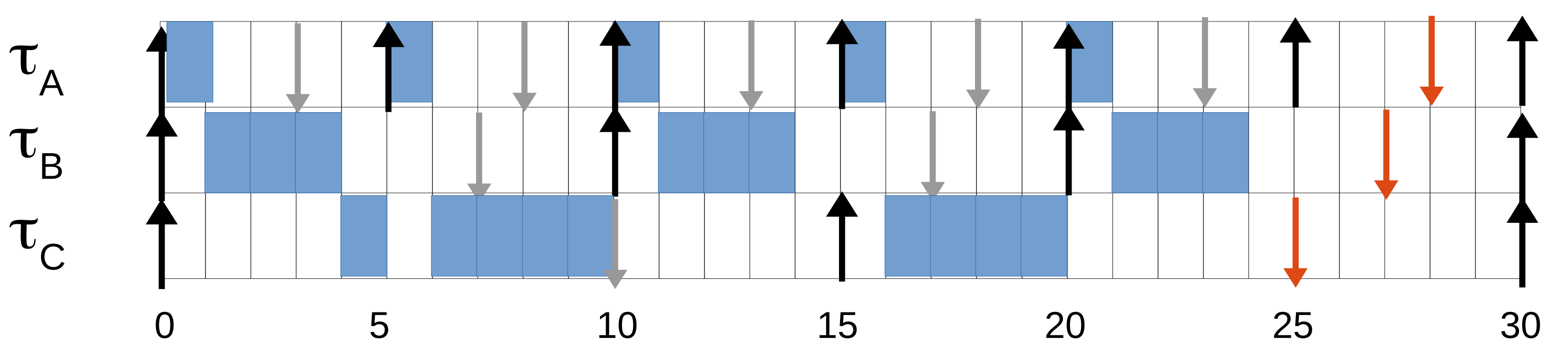
Deadline Monotonic (DM)

- Example
  - Priorities:

    $p_A > p_B > p_C$

  - t=21 : τ_B scheduled

Task	Period T	Relative deadline D	WCET C
A	5	3	1
B	10	7	3
C	15	10	5



Real-time scheduling

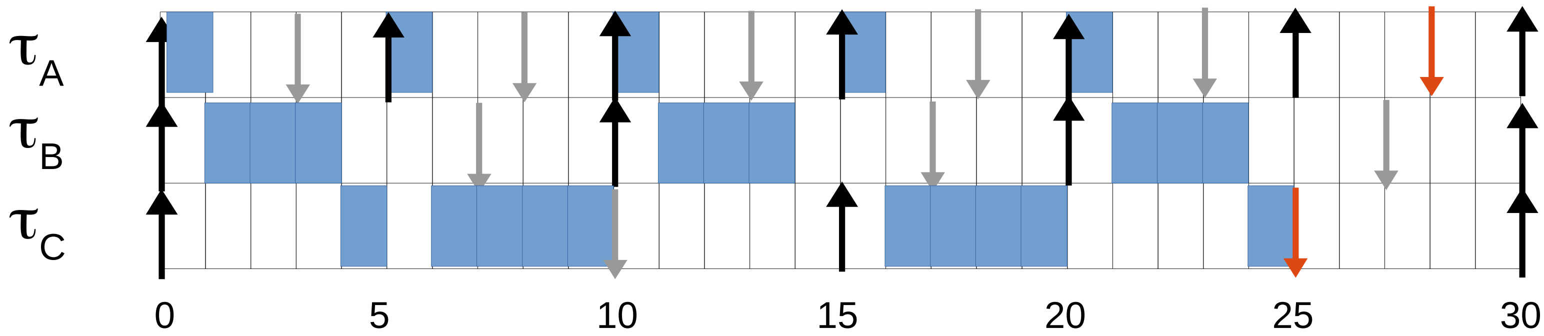
Deadline Monotonic (DM)

- Example
  - Priorities:

    $p_A > p_B > p_C$

  - t=24 : τ_C resumed

Task	Period T	Relative deadline D	WCET C
A	5	3	1
B	10	7	3
C	15	10	5



Real-time scheduling

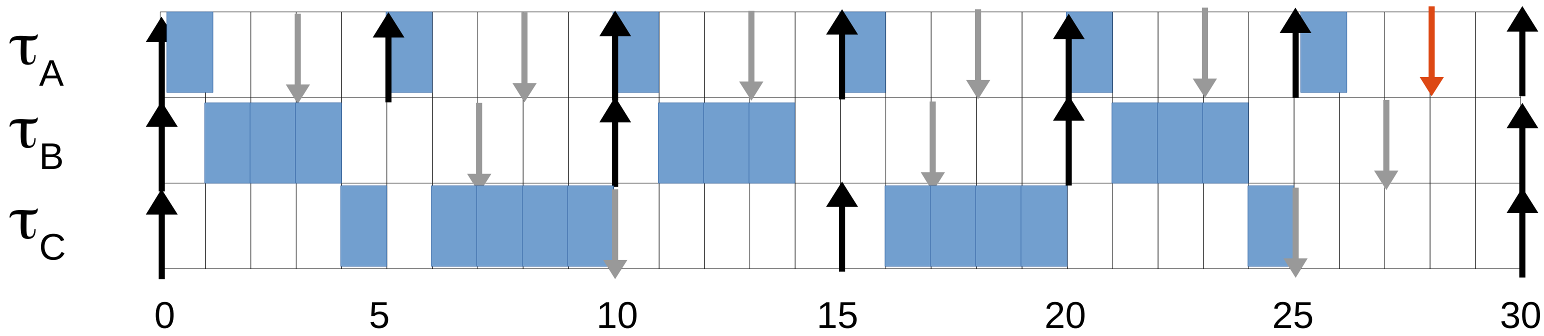
Deadline Monotonic (DM)

- Example
  - Priorities:

    $p_A > p_B > p_C$

  - t=25 : τ_A activated and scheduled
  - Mayor cycle completed

Task	Period T	Relative deadline D	WCET C
A	5	3	1
B	10	7	3
C	15	10	5



Real-time scheduling

Deadline Monotonic (DM)

- Observation
  - DM can schedule periodic tasks without the need of knowing the release times a priori (differently from Rate Monotonic)

    Priority does not depend on T_i

Questions?

Contacts

- William Fornaciari
  https://home.deib.polimi.it/fornacia
  william.fornaciari@polimi.it

- **HEAPLab**
  DEIB Politecnico di Milano
  Building 21, Floor 1
  Phone: 9613 / 9623

- **Slides credits**:
  - Giuseppe Massari <giuseppe.massari@polimi.it>