Real-Time and Embedded Systems

Luís Miguel Pinho

Real-Time Systems

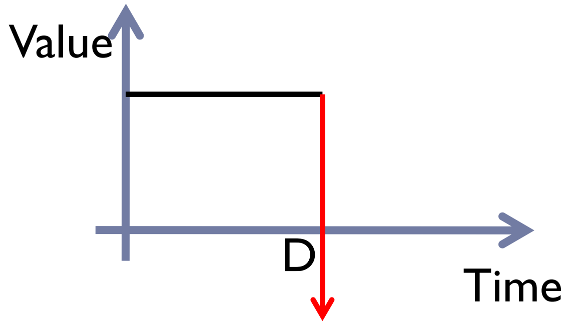
- Used to refer to systems where the timing requirements of the applications are as important (or more) as other functional requirements
  - The system is said to be correct, if it provides the correct functional result, at the correct time instant
- Examples:
  - Airplane control systems
  - Medical equipment
  - Automotive software
  - Industrial control
  - Telecommunications (routers, …)
  - Consumer electronics (mobile phones, set-top-boxes, …)
- Usually embedded, and often critical
  - Interacts with its environment
  - Highly critical when human lifes at stake

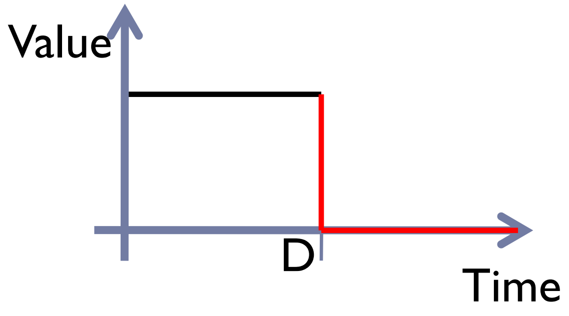
Real-Time Systems

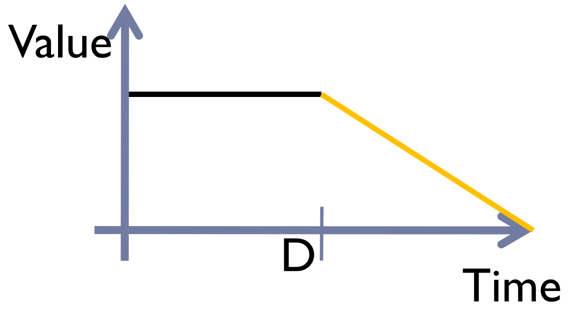
- Distinguishing feature of real-time systems: handling time is fundamental
  - Results need to be provided within specific time intervals (deadlines)
    - Deadlines are requirements (imposed) from applications and systems
  - Real-time is many times an important factor in cyber-physical systems
    - Correction of the computing system, an already important aspect is even more challenged due to the time dimension
  - If deadlines are not met, the system may fail (hard deadlines), which can lead to severe consequences
    - Vehicle not stopping before collision
    - X-ray machine causing too much radiation
    - Trade order after stock price change
    - …

Real-Time Systems

- One of the most important concepts is that of the Deadline (D)
  - It is the time instant where the result should be delivered
    - Lack of delivery has different meanings depending of the criticality of the activity







Hard Real-Time
-The Value is infinetively negative after the deadline
- Critical to the system correct behavior

Firm Real-Time
-The Value is zero after the deadline
- Not critical to the system correct behavior

Soft Real-Time
-The Value decreases after the deadline
- Not critical to the system correct behavior

Real-Time Systems

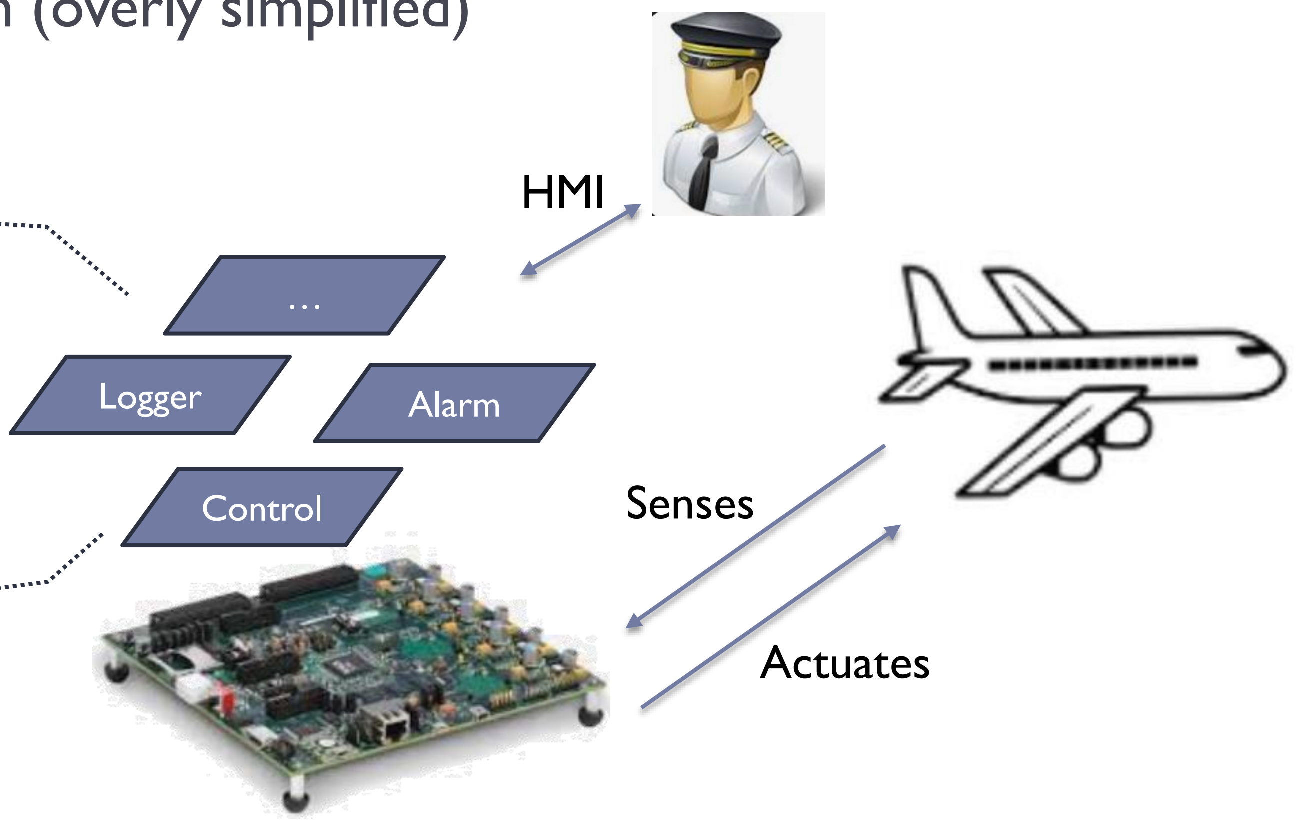
- Real-Time Systems are classified
  - Soft Real-Time – Soft (or firm) time constraints only, such as multimedia systems
  - Hard Real-Time – At least one hard time constraint, such as automotive control, avionics, nuclear (safety/mission-critical systems)
- Hard real-time systems are usually safety-critical, which means that
  - Strong worst-case assumptions need to be considered
    - The system must be able to provide the required level of functionality in the worst-case scenario (which may be overly too pessimistic)
  - Dependability requirements are also fundamental
    - e.g. $< 10^{-9}$ failures/hour, which means one every 114 thousand years
  - These cannot be validated by testing only
    - Out of scope of RTAES

Real-Time Scheduling

- An example
  - An airplane control system (overly simplified)

Temporal requirements of control software are imposed by the dynamics of the plane

The different activities execute concurrently, sharing the resources of the computing platform



Real-Time Systems

- Activities
  - Sense the environment and acquire data
    - Observe the system state, building a "digital shadow" of the system
    - Data with temporal validity depending on the dynamics of the controlled system
    - Eventually need for periodic sampling, or interrupt-driven behavior
      - Taking into consideration also the variations (jitter) of the sampling
  - Perform control algorithms
    - Control the evolution of the system
    - Control laws dictated by the dynamics of the system
    - Necessary to take into consideration the execution time of the computing activities
  - Actuate
    - Interface with actuators
    - Periodically or event-driven

Real-Time Systems

- In order to provide a correct (time and value) result, the system needs to:
  - Provide a functional correct algorithm
    - Functional properties still hold

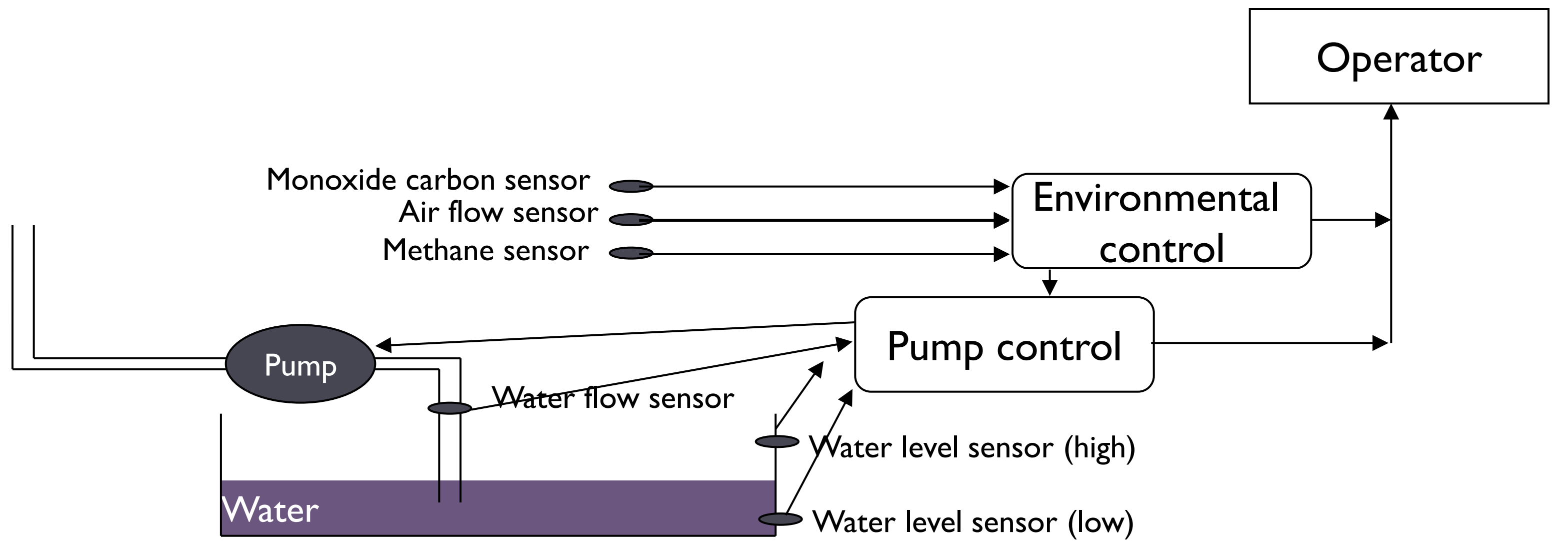
  - Guarantee timing requirements of applications
    - Using specific scheduling techniques
    - Applying specific schedulability analysis
    - Control resource consumption of the different concurrent activities
      - Isolate critical and non-critical activities
    - For critical systems, using worst-case assumptions, not average

Non-functional properties

This is the subject of the course

Real-Time Systems

- Slightly more complete example (of a simpler system)
  - Adapted from Burns & Wellings, "Real-Time Systems and Programming Languages", Addision-Wesley, 2001
  - Mine pump control system



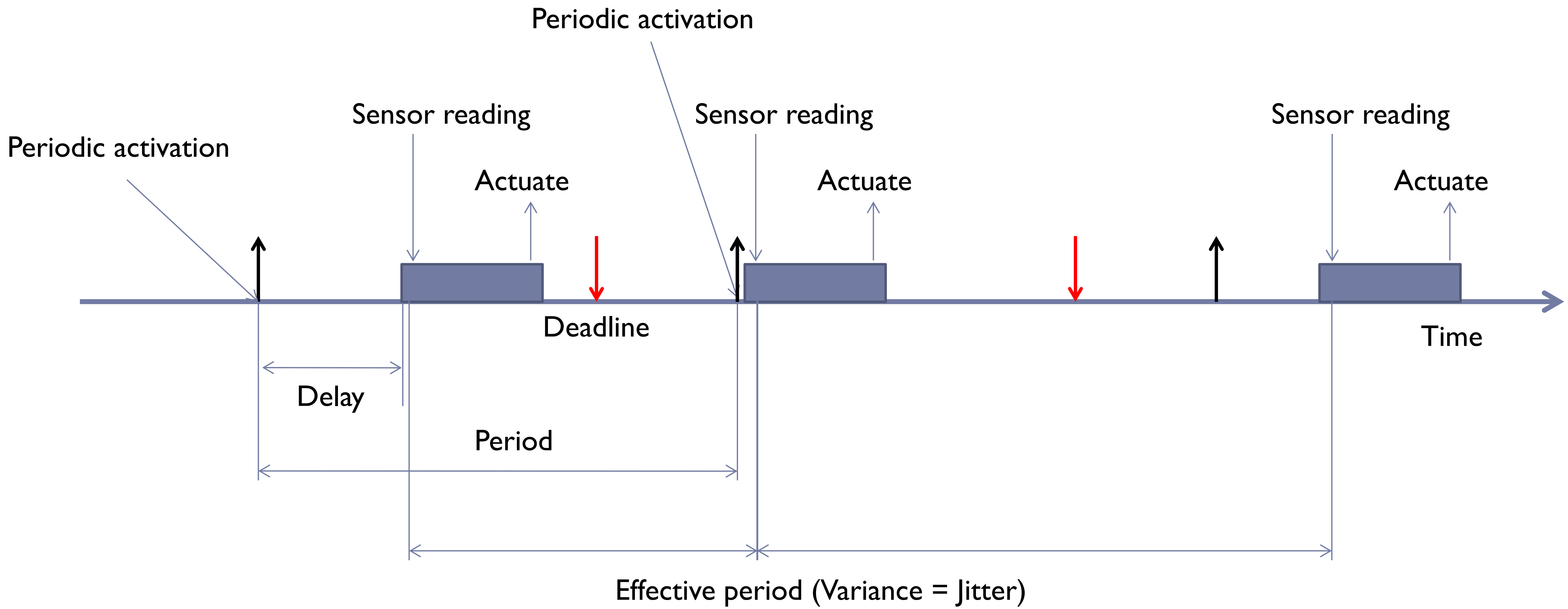
Real-Time Systems

- Functional requirements
  - Turn on the pump when the water level reaches a maximum limit.
  - When the water level reaches the minimum level, the pump should be turned off.
  - There is a sensor that detects the water flow from the pump to the surface, controlling the pump's operation.
  - If the methane level in the mine exceeds a certain threshold, the pump should be turned off.
  - The methane level must be monitored so that when it exceeds the maximum allowable level, the pump is turned off, and the mine is evacuated.
  - If the carbon monoxide level exceeds the maximum level, evacuate the mine.
  - In the absence of airflow, evacuate the mine.
  - The sensors for water flow, carbon monoxide, methane, and airflow are read through polling.
  - The maximum and minimum water level sensors operate through interruption.
  - The operator should be periodically informed of the system's status.
  - In case of any issues, the operator should be promptly notified.

Real-Time Systems

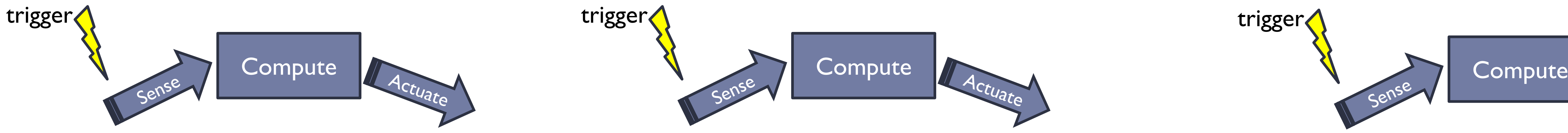
- Timing requirements
  - The carbon monoxide sensor value must be read every 10 milliseconds (ms) with a deadline of 6 ms
  - The methane sensor value must be read every 8 ms, with a deadline of 3 ms
  - Airflow should be checked every 10 ms, with a 10 ms deadline
  - The water flow must be checked every 100 milliseconds with a deadline of 4 ms
  - The system state must be made available to the operator every second (there is no deadline)
  - The water level sensors operate every 600 ms at most (the deadline is 20 ms)
  - A problem can happen at most every 1 second, and when it happens, the system must act in less than 20 ms

Real-Time Systems



Real-Time Computing Model

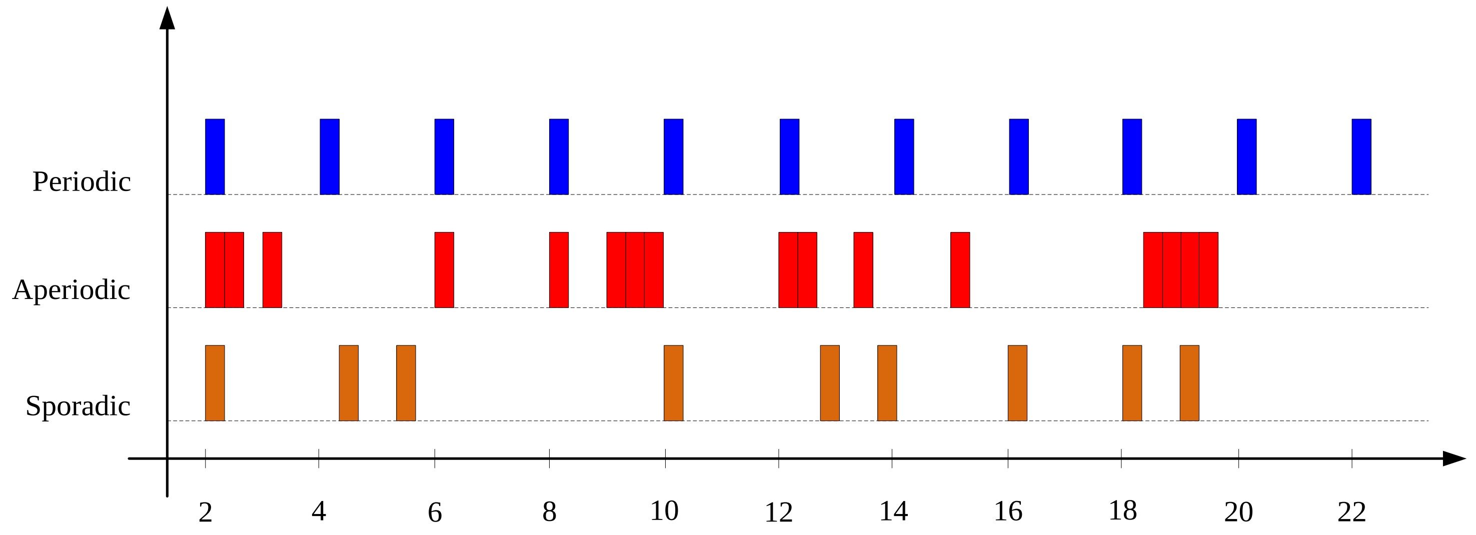
- The real-time computing model is a recurrent reactive model
  - An infinite sequence of recurring activities (named **tasks**), processing inputs and computing the respective outputs
  - The simplest is a sense-compute-actuate



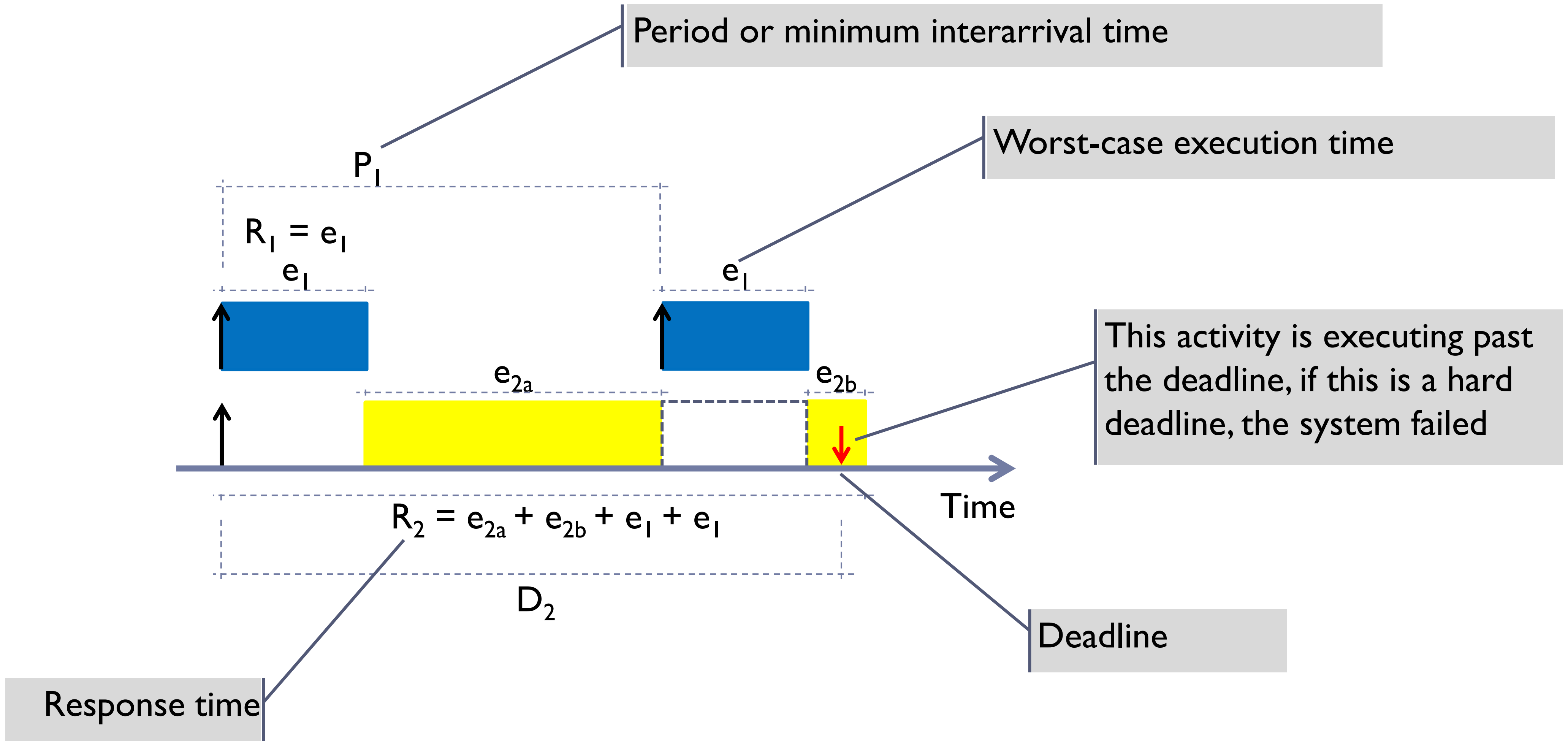
  - But more complex can exist (e.g a DAG)

Real-Time Computing Model

- Interaction of multiple concurrent activities (which may also communicate/synchronize), competing for the same hardware resources
  - These activities (tasks) can be periodic or non-periodic (i.e., event-driven)
    - Periodic activities repeat regularly after a fixed time interval
    - Sporadic activities have a maximum frequency of repetition
    - Aperiodic activities have no constraints



Real-Time Computing Model


Period or minimum interarrival time
Worst-case execution time
This activity is executing past the deadline, if this is a hard deadline, the system failed
Deadline
Response time
P_1
$R_1 = e_1$
e_1
e_1
e_{2a}
e_{2b}
Time
$R_2 = e_{2a} + e_{2b} + e_1 + e_1$
D_2

Real-Time Computing Model

- Examples of Periodic Task

Ada Implementation

```
task body Task_A is
   Next_Time : Time;
   Task_A_Period : Time_Span := To_Time_Span (0.003);
begin
   Next_Time := Clock;
   loop
      - -  Do something

      Next_Time := Next_Time + Task_A_Period;
      delay until Next_Time;
   end loop;
end Task_A;
```

C on FreeRTOS Implementation

```
void vPeriodicTask( void * pvParameters ) {
   TickType_t xLastWakeTime;
   const TickType_t xDelay3ms = pdMS_TO_TICKS( 3 );
   xLastWakeTime = xTaskGetTickCount();

   for( ;; )   {
    // Do something

       vTaskDelayUntil( &xLastWakeTime, xDelay3ms );
   }
}
```

Real-Time Computing Model

- Examples of Sporadic/Aperiodic Task

Ada Implementation

```
task body Task_A is
begin
   loop
      IO.Get;
      - -  Do something

   end loop;
end Task_A;
```

C on FreeRTOS Implementation

```
void vAPeriodicTask( void * pvParameters ) {
   for( ;; )   {
     xSemaphoreTake( xBinarySem, portMAX_DELAY );
     // Do something

   }
}
```

- Examples with C and POSIX OS in the practical labs

Real-Time Computing Model

- Assume a simple model
  - The application is assumed to consist of a fixed set of tasks
  - All tasks are periodic, with known periods
    - Each period there is a new **job** of the task
  - The tasks are completely independent of each other (for now)
  - All system's overheads, context-switching times and so on are ignored (i.e, assumed to have zero cost)
  - All tasks have a deadline equal to their period (that is, each one must complete before it is next released)
    - We will also see simple cases of constrained deadlines ($D < T$), but not address arbitrary deadlines ($D > T$)
  - All tasks have a fixed worst-case execution time
- Very restrictive model
  - More accurate models exist but not in the context of this course

Real-Time Computing Model

- On the *a priori* assumptions
  - Context switches and overheads are usually incorporated in the execution time of tasks
  - Caches, pipelines are usually ignored
    - In Critical systems they are switched off
    - In other systems, measurable "worst-case" values are used
  - Worst-case execution time of tasks
    - In critical systems it is calculated by static analysis/inspecting object code
      - Loops and conditions are always with their worst-case
      - More and more difficult to statically analyze complex processors (pipelines, caches, buses, …)
    - In other systems measurable "worst-case" values are used
    - In soft real-time, average values may be used
  - Task periods
    - Tasks may be not periodic in nature: in this case (sporadic or aperiodic), the worst-cast inter-arrival time is used

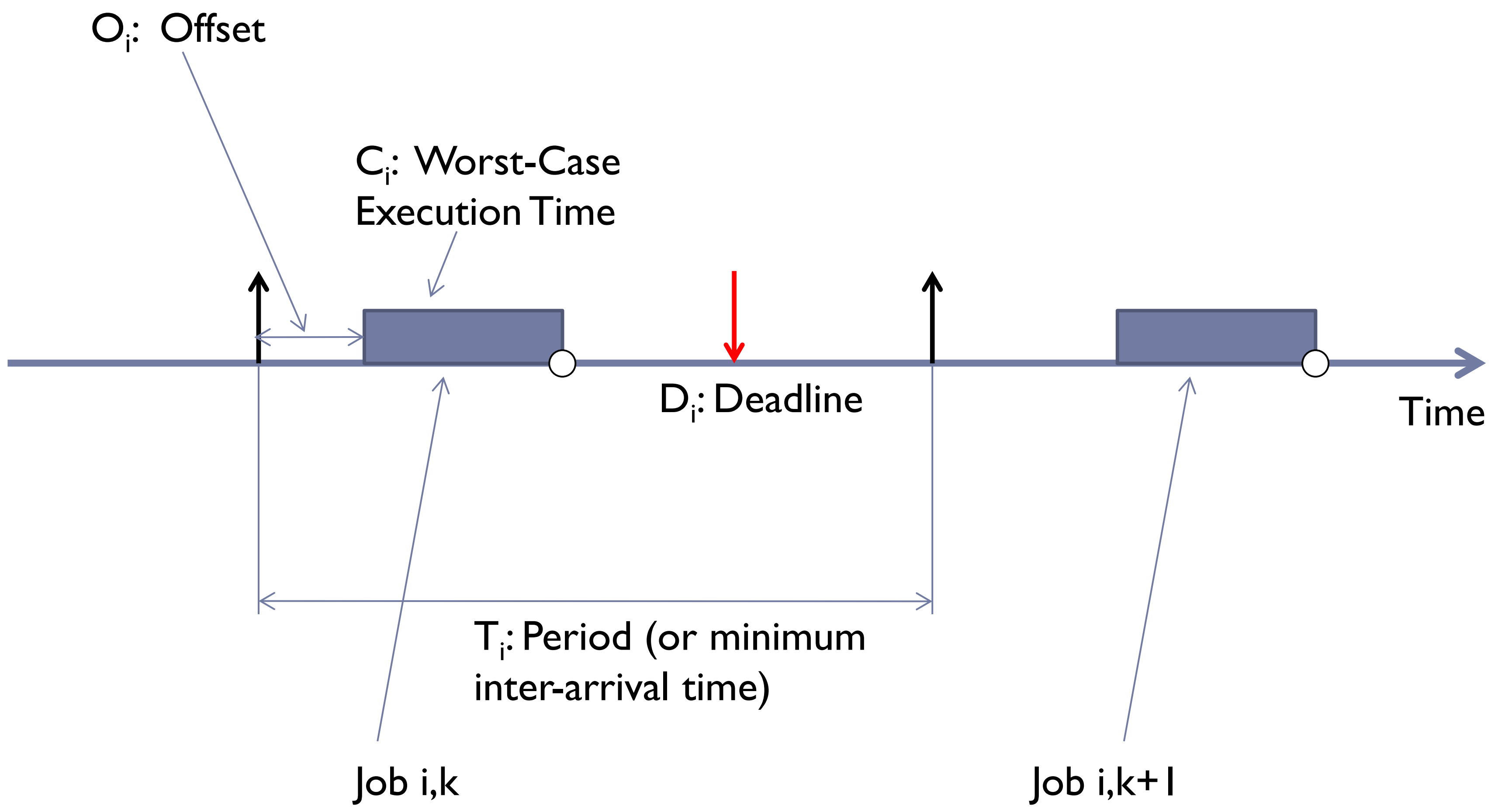
Real-Time Computing Model

- An example
  - An airplane control system has the following tasks:
    - A periodic task responsible for controling the pitch, roll and yaw of the airplane, which has to execute every 60 ms. It takes 20 ms to execute and must output the results in 40 ms
    - A sporadic alarm task, that when occurring must be handled in 20 ms, taking 5 ms to execute.
    - A periodic logger task, which writes the log of events to the black-box, with a period of 100 ms and an execution time of 50 ms

	C (ms)	T (ms)	D (ms)
Control	20	60	40
Alarm	5	-	20
Logger	50	100	-

Real-Time Computing Model

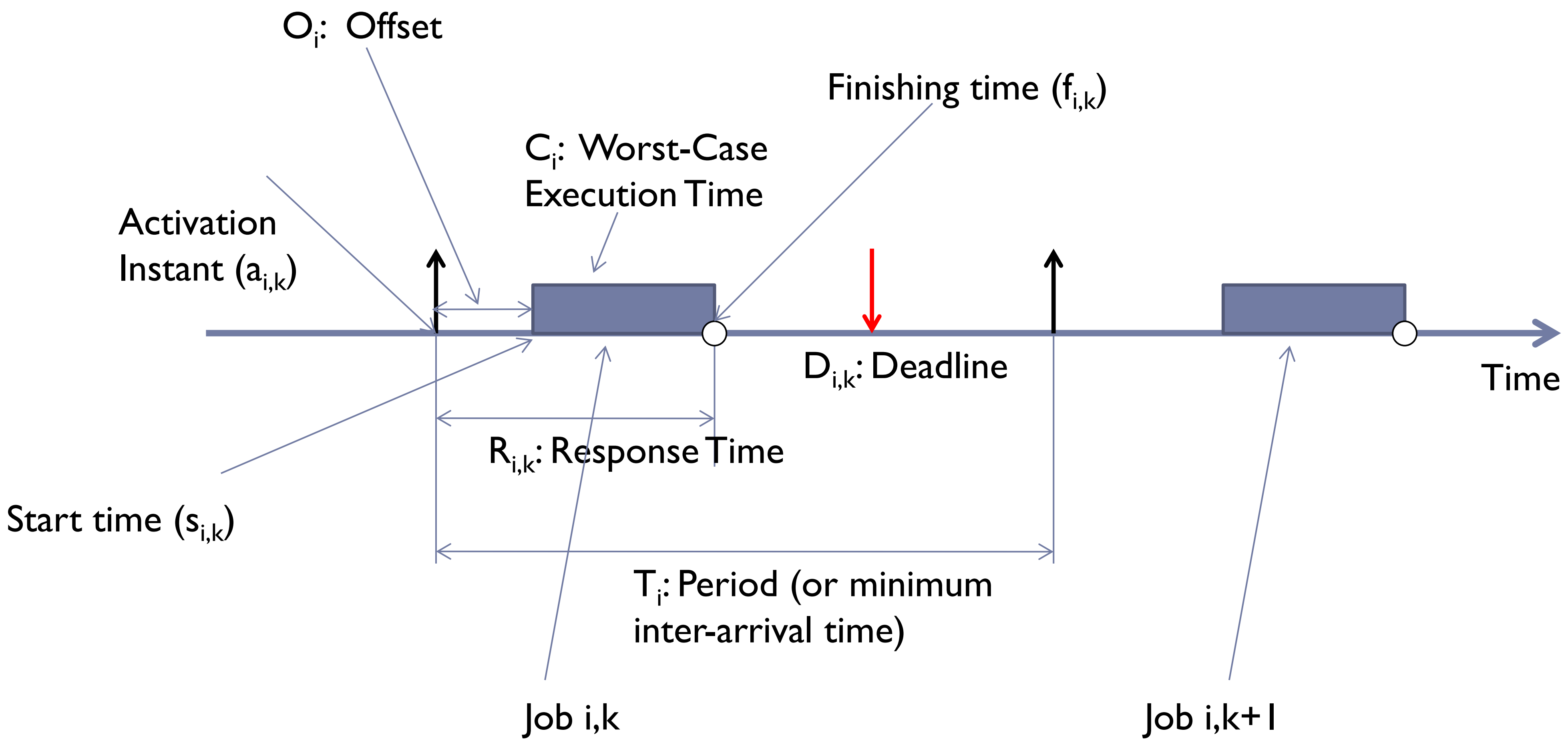
- A simple model of a periodic task (t_i)



O_i: Offset
C_i: Worst-Case Execution Time
D_i: Deadline
Time
T_i: Period (or minimum inter-arrival time)
Job i,k
Job i,k+1

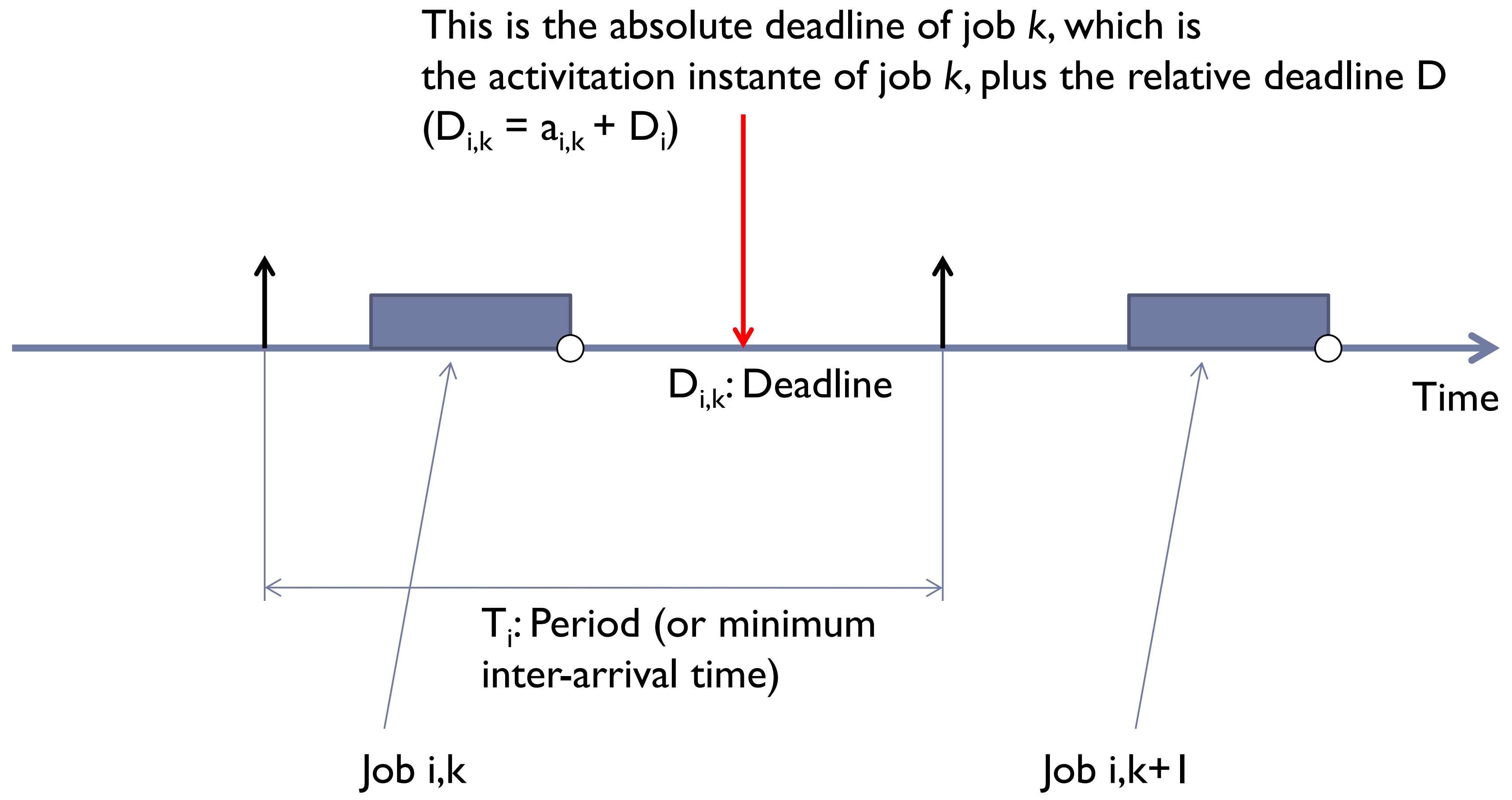
Real-Time Computing Model

- A simple model of a periodic task (t_i)


O_i: Offset
Finishing time ($f_{i,k}$)
C_i: Worst-Case Execution Time
Activation Instant ($a_{i,k}$)
$D_{i,k}$: Deadline
Time
$R_{i,k}$: Response Time
Start time ($s_{i,k}$)
T_i: Period (or minimum inter-arrival time)
Job i,k
Job i,k+1

Real-Time Computing Model

- A simple model of a periodic task (t_i)



This is the absolute deadline of job *k*, which is the activitation instante of job *k*, plus the relative deadline D ($D_{i,k} = a_{i,k} + D_i$)
$D_{i,k}$: Deadline
Time
T_i: Period (or minimum inter-arrival time)
Job i,k
Job i,k+1

Real-Time Computing Model

- A simple model of a periodic task *i* (t_i)
  - C_i: Worst-Case Execution Time
  - T_i: Period (or minimum inter-arrival time)
  - D_i: Relative Deadline
  - O_i: Offset (phase shift)
  - R_i: Worst-case Response Time
  - Activation instant of a job ($a_{i,k}$)
  - Start time of a job ($s_{i,k}$)
  - Finishing time of a job ($f_{i,k}$)
  - $R_{i,k}$: Response Time of a job
  - $D_{i,k}$: Absolute Deadline of a job

Real-Time Computing Model

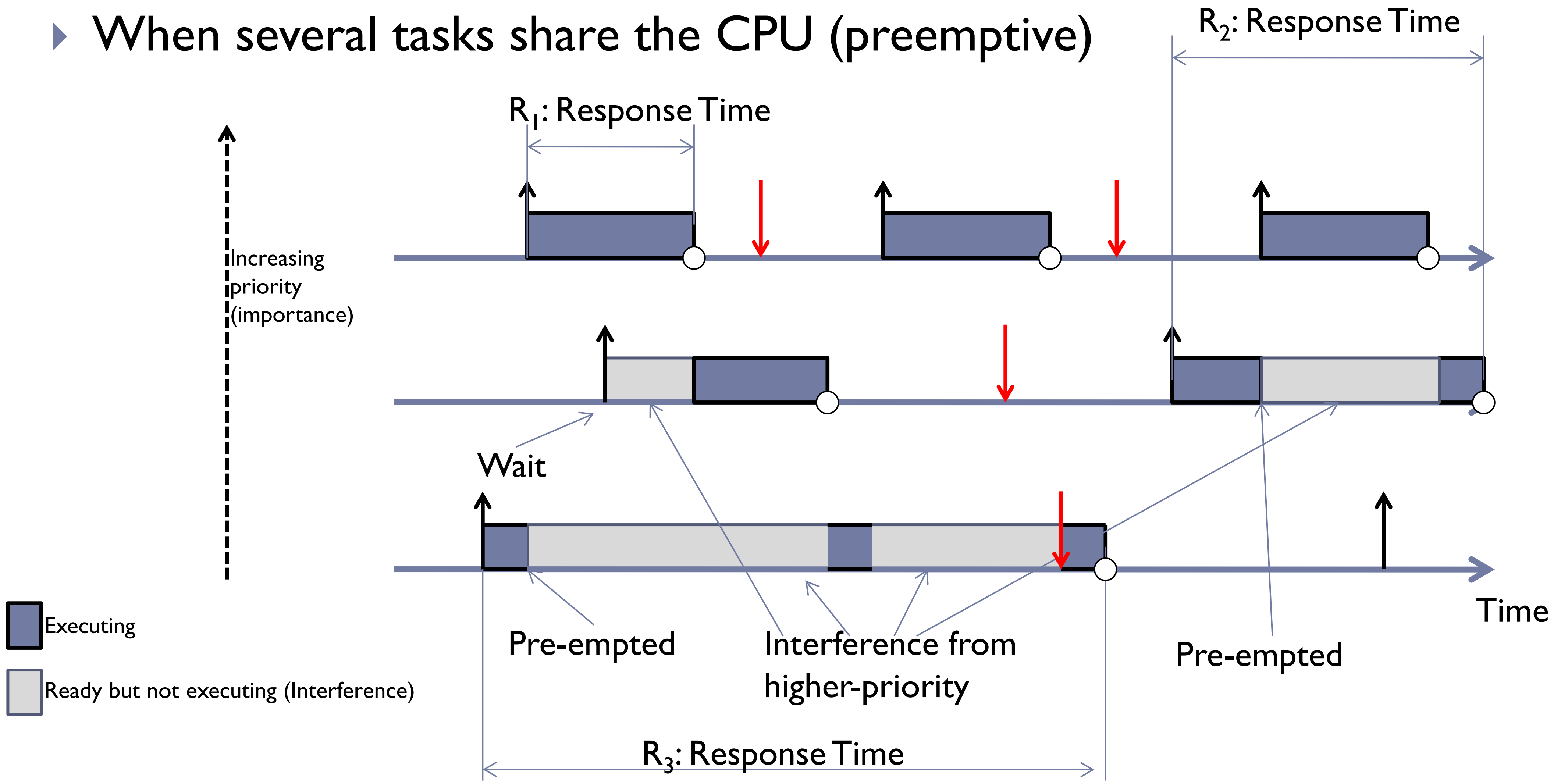
- When several tasks share the CPU
  - System can be preemptive
    - A task executing can be suspended (preempted) and replaced in the CPU by another task (e.g. due to an interrupt, or a higher priority task becoming ready)
  - Non-preemptive
    - A task executed until the end of its current job, with no suspension
  - Limited preemptive
    - Tasks can only be preempted at specific points of its execution
- Access to shared resources may change the preemptibility of a task, even on a preemptive system
  - E.g. a task cannot be suspended as its holding a lock

Real-Time Computing Model

- Preemption is important in real-time scheduling
  - It allows to allocate the processor to "important" ready tasks immediately
  - High priority tasks are not blocked by low priority tasks
- However, non-preemptive reduces context-switch overhead
  - Making execution times smaller and easier to calculate (more predictable)
  - Simplifies the access to shared resources (no semaphores needed)
  - Deadlock prevention is easy
  - Reduces stack size (and tasks can share the same stack, since no more than one task can be in execution)

Real-Time Computing Model

- When several tasks share the CPU (preemptive)

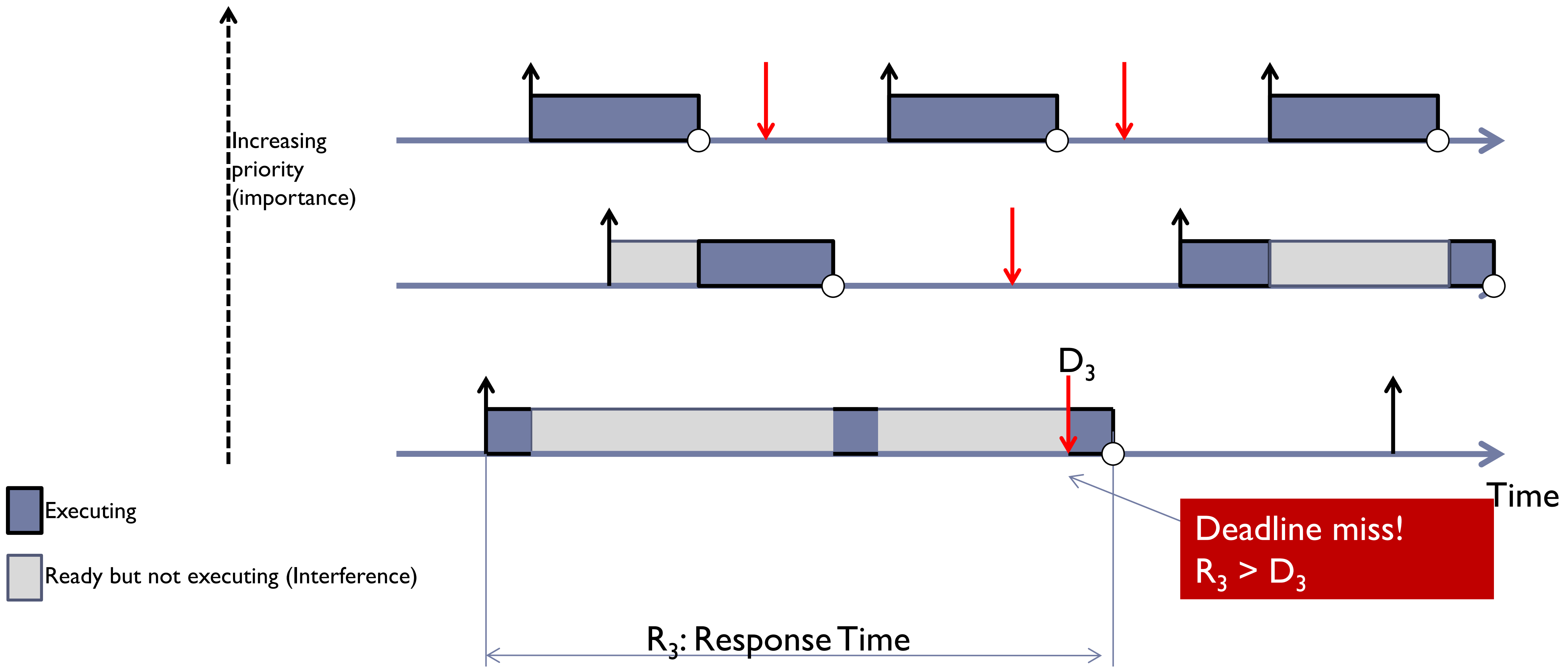


R_2: Response Time
R_1: Response Time
Increasing priority (importance)
Wait
Executing
Ready but not executing (Interference)
Pre-empted
Interference from higher-priority
Pre-empted
Time
R_3: Response Time

$$R_i = \max_{k=1 \ldots L} \{ r_{i,k} \}$$

Real-Time Computing Model

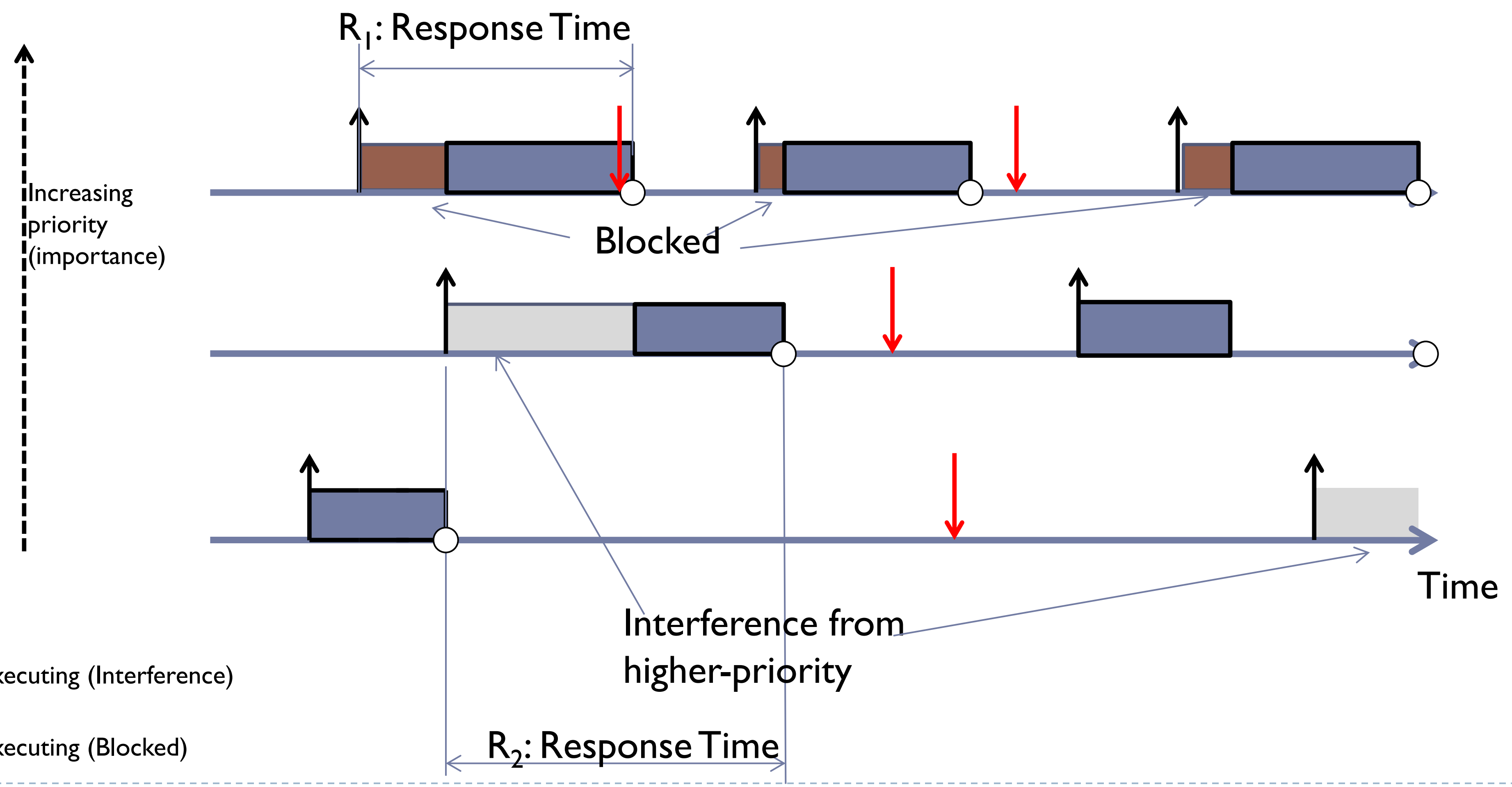
- When several tasks share the CPU (preemptive)



Increasing priority (importance)
D_3
Time
Deadline miss!
$R_3 > D_3$
R_3: Response Time
Executing
Ready but not executing (Interference)

Real-Time Computing Model

- When several tasks share the CPU (non-preemptive)


R_1: Response Time
Increasing priority (importance)
Blocked
Time
Interference from higher-priority
R_2: Response Time

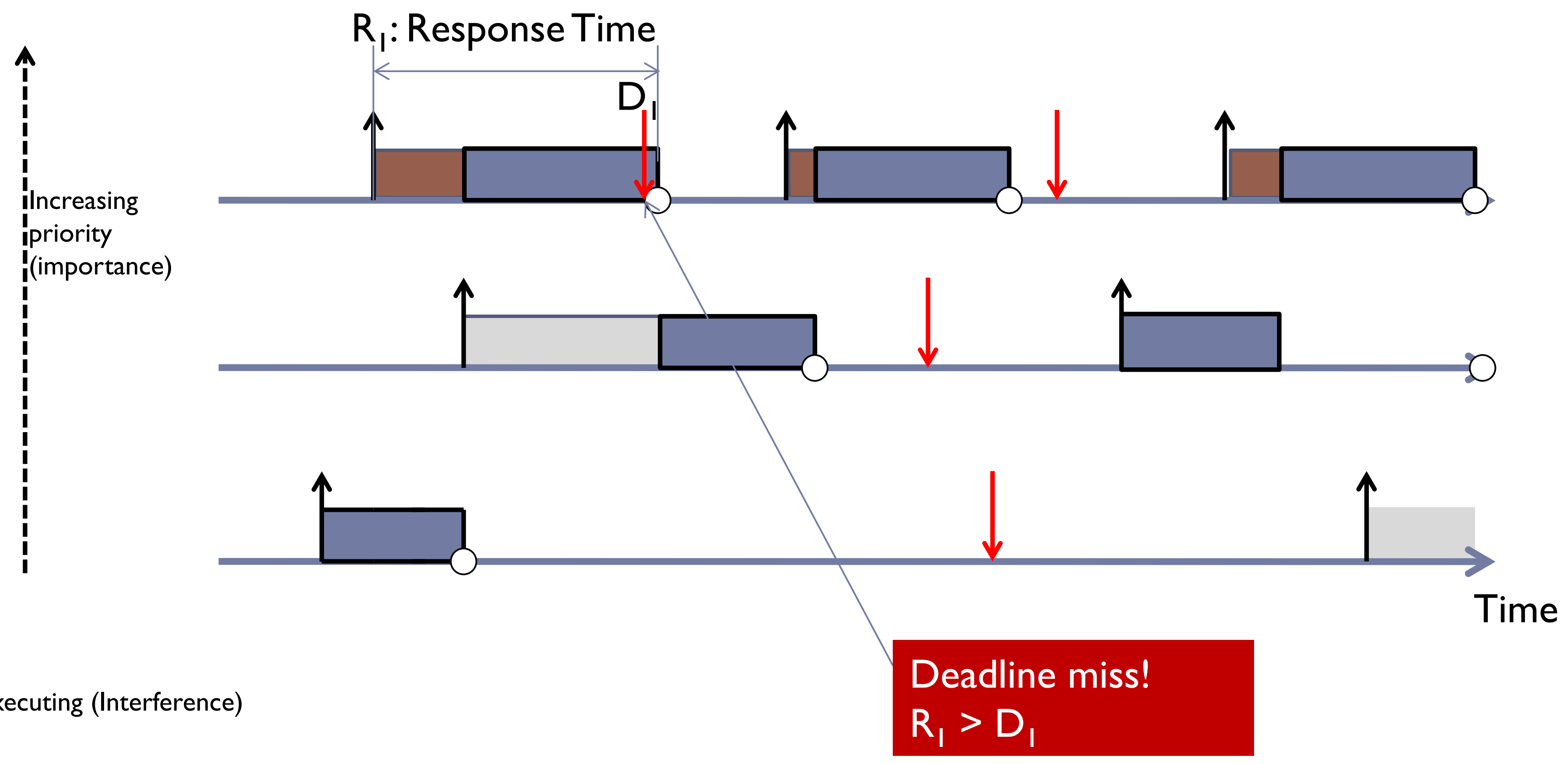
🟦 Executing

⬜ Ready but not executing (Interference)

🟫 Ready but not executing (Blocked)

Real-Time Computing Model

- When several tasks share the CPU (non-preemptive)



🟦 Executing

⬜ Ready but not executing (Interference)

🟫 Ready but not executing (Blocked)

Real-Time Computing Model

- Task Activation Models
  - Time-triggered: activation is triggered by the progression of time (periodic tasks)
    - Well-defined scenario, appropriate for control
    - Easy to analyze worst-case
    - May induce underutilization
  - Event-triggered: activation is triggered by asynchronous events, e.g. interrupt (sporadic/aperiodic task)
    - Less predictable scenario, appropriate for alarms
    - Difficult to define worst-case situation
    - Necessary to provide some control of event rate bounds ("bandwidth")

Real-Time Computing Model

- A simple model of a set of periodic tasks
  - $\tau_i = (C_i , O_i , T_i , D_i)$

	C (ms)	T (ms)	D (ms)	Act Model
Control (τ_1)	20	60	40	TT
Alarm (τ_2)	5	-	20	ET
Logger (τ_3)	50	100	-	TT

  - $\tau_1 = (20, 0, 60, 40)$
  - $\tau_2 = (5, 0, -, 20) = (5, 0, \mathbf{70}, 20)$ e.g. alarm hysteresis of 70 ms
  - $\tau_3 = (50, 0, 100, -) = (5, 0, 100, \mathbf{100})$ e.g. deadline equals period

Real-time Scheduling

- Probably the most developed topic in the real-time systems community
  - Scheduling: how to manage the interaction of multiple concurrent activities (which may also communicate/synchronize), competing for the same hardware resources
  - These are fundamental to guarantee the correct behaviour of the application
- An important and complementary area of activity is related to the techniques for analysing the timing behaviour
  - Timing analysis – the objective is to compute bounds (or probabilistic profiles) on the worst-case execution time (WCET) of activities
  - Schedulability analyses – the objective is to check analytically, at design or run time, whether all the deadlines will be met
  - Usually necessary to give guarantees of correct time behaviour before deployment
    - Offline analysis – will always meet (hard) deadlines (under a certain confidence)

Real-Time Scheduling

- In general, a scheduling scheme provides two features:
  - An algorithm for ordering the use of system resources (in particular the CPUs)
  - A means of predicting the worst-case behavior of the system when the scheduling algorithm is applied
    - The prediction can then be used to confirm the temporal requirements of the application
  - A taskset is schedulable if there is at least one schedule that allows to guarantee the temporal requirements of the application
- In Real-Time Systems it is important to derive the worst-case conditions
  - To validate if, in the worst-case, the system will fail a deadline
- The goal is to provide
  - Determinism: same inputs will provide same outputs
  - Predictability: "determinism" within a known time interval

Real-Time Scheduling

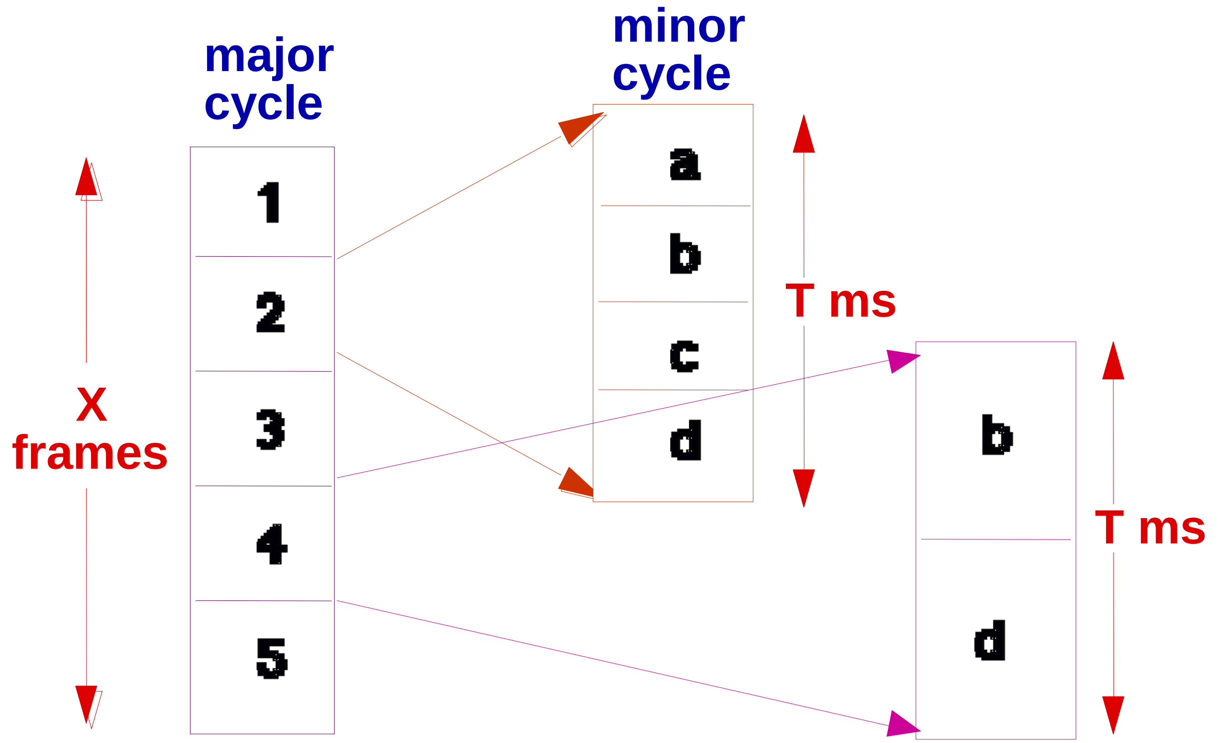
- The scheduling techniques for real-time systems range
  - I. Static approaches where the schedule is built *a priori* (off-line)
  - II. Hybrid approaches where all characteristics are known *a priori* but the schedule is created in run-time (on-line)
  - III. Fully on-line approaches, which only provide best effort guarantees
  - IV. Complex approaches, integrating some of the above
    - But where the critical applications are scheduled with static approaches
    - And isolation is guaranteed
- We will briefly refer to (I) and (III) and mainly talk of (II)
  - Preemptive and non-preemptive
  - Work-conserving

Real-Time Scheduling

- Cyclic Executive
  - In the simplest solution, a hard real-time systems can be implemented with a cyclic executive
    - Tasks are designed as a set of functions
    - Functions are mapped onto a set of minor cycles that constitute the complete schedule (or major cycle)
  - The difficulty of incorporating tasks with long periods; the major cycle time is the maximum period that can be accommodated without secondary schedules
    - Non-periodic tasks are difficult to incorporate
  - The cyclic executive is difficult to construct and difficult to maintain
    - it is a NP-hard problem
  - Any "task" with a sizable computation time will need to be split into a fixed number of fixed sized functions
    - this may cut across the structure of the code from a software engineering perspective, and hence may be error-prone

Real-Time Scheduling

- Cyclic Executive
  - Manually constructed Time Frame or Schedule
    - Completely deterministic execution
    - Schedule repeatedly executed every X * T ms



Real-Time Scheduling

Cyclic Executive

▨ Task 1 (C = 1 , T = 4)

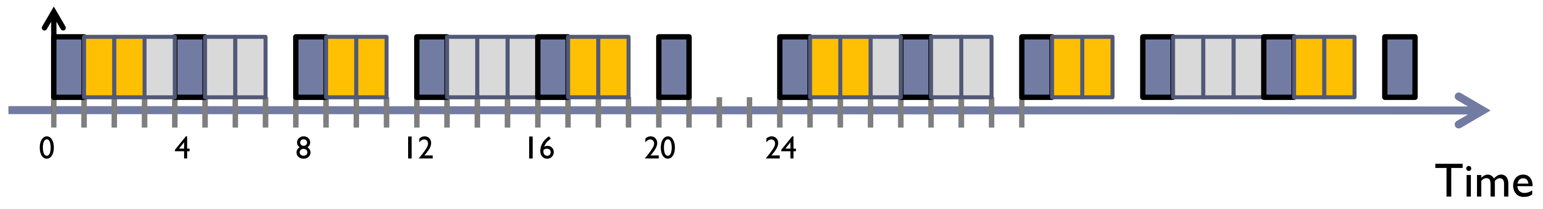
🟨 Task 2 (C = 2 ,T = 8)

□ Task 3 (C = 3 ,T = 12)

What are the minor and major cycle?

- Minor cycle time is the HCF [highest common factor] of periods
- Major cycle time is the LCM [least common multiple] of periods

In this case, periods are 4,8,12 => HCF = 4; LCM = 24



CPU utilization is: $$\sum U_i = \sum \frac{C_i}{T_i} = \frac{1}{4} + \frac{2}{8} + \frac{3}{12} = 75\%$$

Real-Time Scheduling

- Cyclic Executive
  - Advantages
    - Simple to implement
    - Low runtime overhead
    - Fully time-triggered
    - Allows to control jitter, precedences
  - Disadvantages
    - Low robustness - small change in execution time can complete break the schedule
    - Requires periods to be harmonic, otherwise major cycle can be too large
    - Becomes difficult to construct for more than a few tasks

Real-Time Scheduling

- More flexible schemes
  - These are based in determining tasks’ characteristics offline (pre-run-time) but allowing the actual execution of tasks to be determined online (during execution)
  - There are pre-emptive and non-pre-emptive versions of these
    - In a preemptive scheme, if necessary there will be an immediate switch to the new task
    - With non-preemption, the currently executing task will be allowed to complete before the other executes
  - Most important
    - Hybrid schemes: Fixed-Priority Scheduling
    - Dynamic schemes: Earliest-Deadline First