Scheduling in Real-Time Operation System

Đánh giá môn học

- Điểm bài tập: Chọn 1 trong những bài báo trong đây để báo cáo, sử dụng nhóm đã lập.
  - Nêu vấn đề
  - Nghiên cứu liên quan
  - Giải pháp
  - Kết luận
- Điểm GK: Lý thuyết, làm bài tại lớp
- Báo cáo cuối kì:

Đánh giá môn học

- Điểm Cuối kì (project): Làm sản phẩm ứng dụng RTOS và viết báo cáo
- Mục tiêu: RTOS hoặc FREERTOS, ứng dụng cho y tế, giáo dục .v.v, tham khảo các bài báo đã publish ở link này.
- Yêu cầu:
- Tên đề tài khác với những bài báo trong link
- Khuyến khích apply machine learning hoặc tiny machine learning. hoặc ứng dụng RTOS trong y tế.
- Mỗi nhóm tối đa 4 thành viên, 2 nhóm 5 thành viên.
- Có thể chọn ESP32, ARM, Rasperry, phải sử dụng RTOS, đồng thời lưu ý yêu cầu tên khác với báo báo đã publish.

Outline

- Real-time systems

- Real-time scheduling algorithms
  - Fixed-priority algorithm (RM)
  - Dynamic-priority algorithm (EDF)

Introduction

- ❑ Why do we need scheduling?
  - There are always more tasks than processors.
  - Multiple tasks run concurrently on uniprocessor system.
- ❑ Scheduling policy: the criterion to assign the CPU time to concurrent tasks
- ❑ Scheduling algorithm: the set of rules that determines the order in which tasks are executed

→ *What is the main difference between scheduling in RTOS and GPOS?*

Scheduling algorithms

- A scheduling algorithm is a scheme that selects what job to run next.
  - Can be preemptive or non-preemptive.
  - Dynamic or static priorities
  - Etc.

In general, a RTS will use some scheduling algorithm to meet its deadlines.

Real-Time Systems

- Definition
  - Systems whose correctness depends on their temporal aspects as well as their functional aspects

- Performance measure
  - Timeliness on timing constraints (deadlines)
  - Speed/average case performance are less significant.

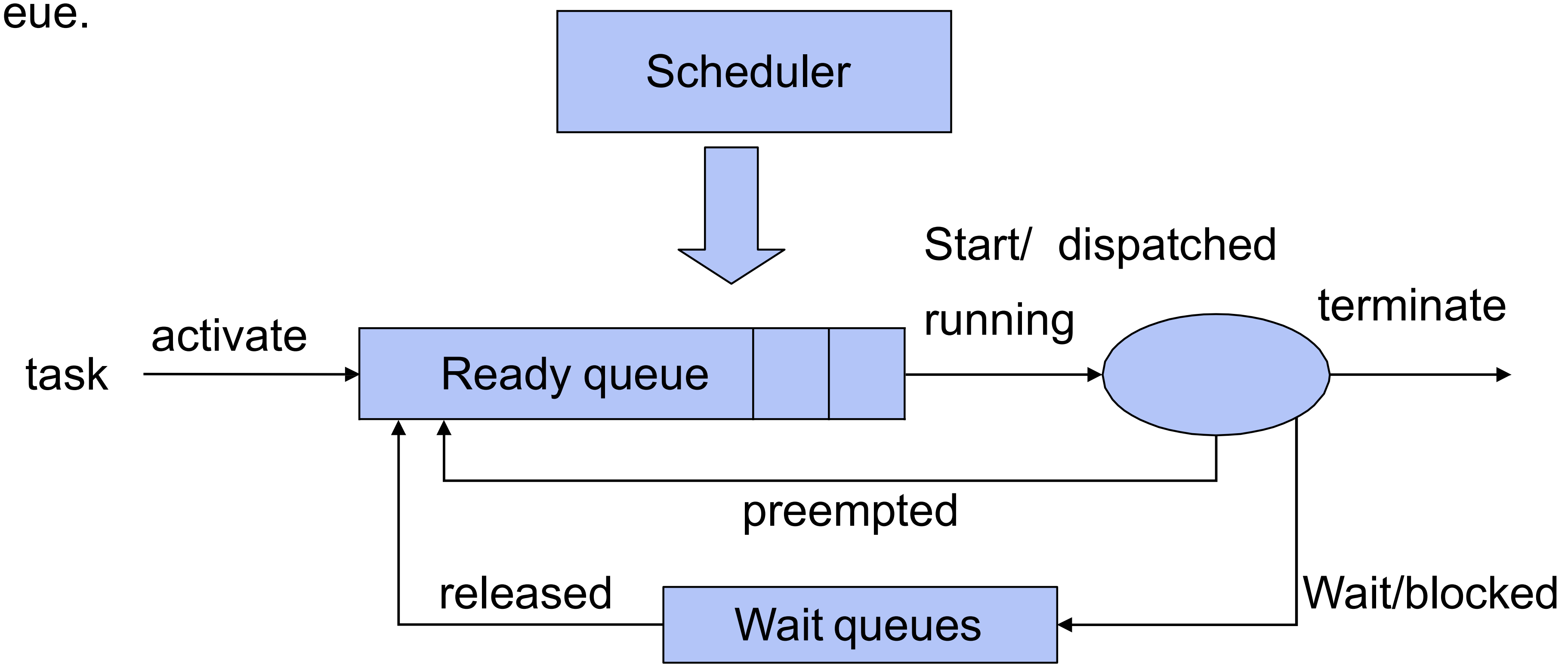
- Key property
  - Predictability on timing constraints

Terms and definitions

- Release time (or ready time): This is the time instant at which a task(process) is ready or eligible for execution
- Schedule Time: This is the time instant when a task gets its chance to execute
- Completion time: This is the time instant when task completes its execution
- Deadline: This is the instant of time by which the execution of task should be completed
- Runtime: The time taken without interruption to complete the task, after the task is released

An illustration of scheduling

- All activated tasks enters “ready queue” at first.
- The scheduler selects one task in the Ready queue according to the tasks’ priorities allocated based on the scheduling algorithm.
- The selected task is dispatched and becomes in “running” state.
- After the selected task is completed, it is removed from the Ready queue.

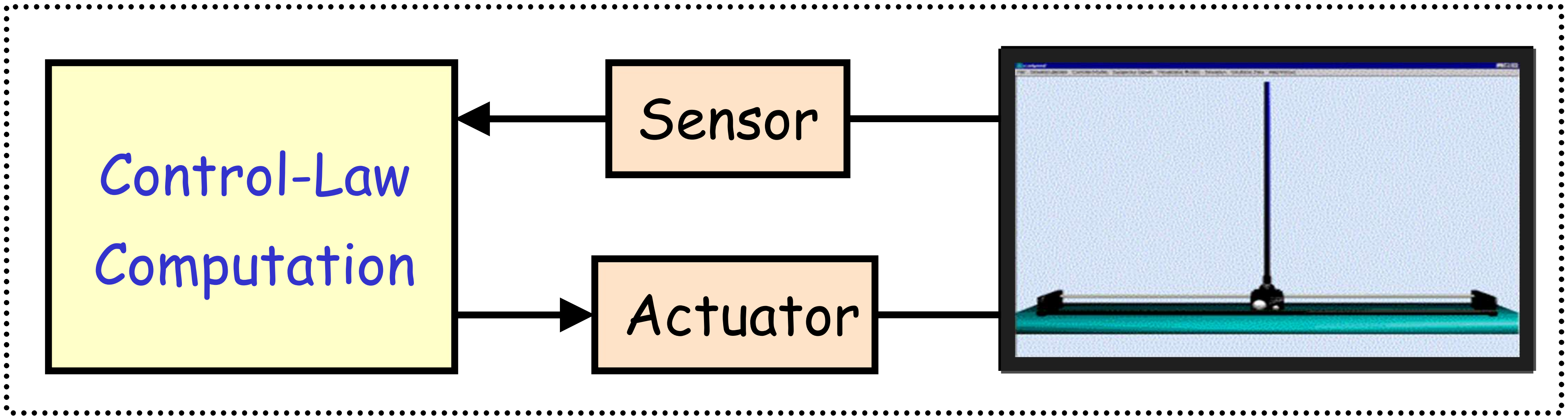


Real-Time System Example

- Digital control systems
  - periodically performs the following job:

    senses the system status and

    actuates the system according to its current status



Real-Time System Example

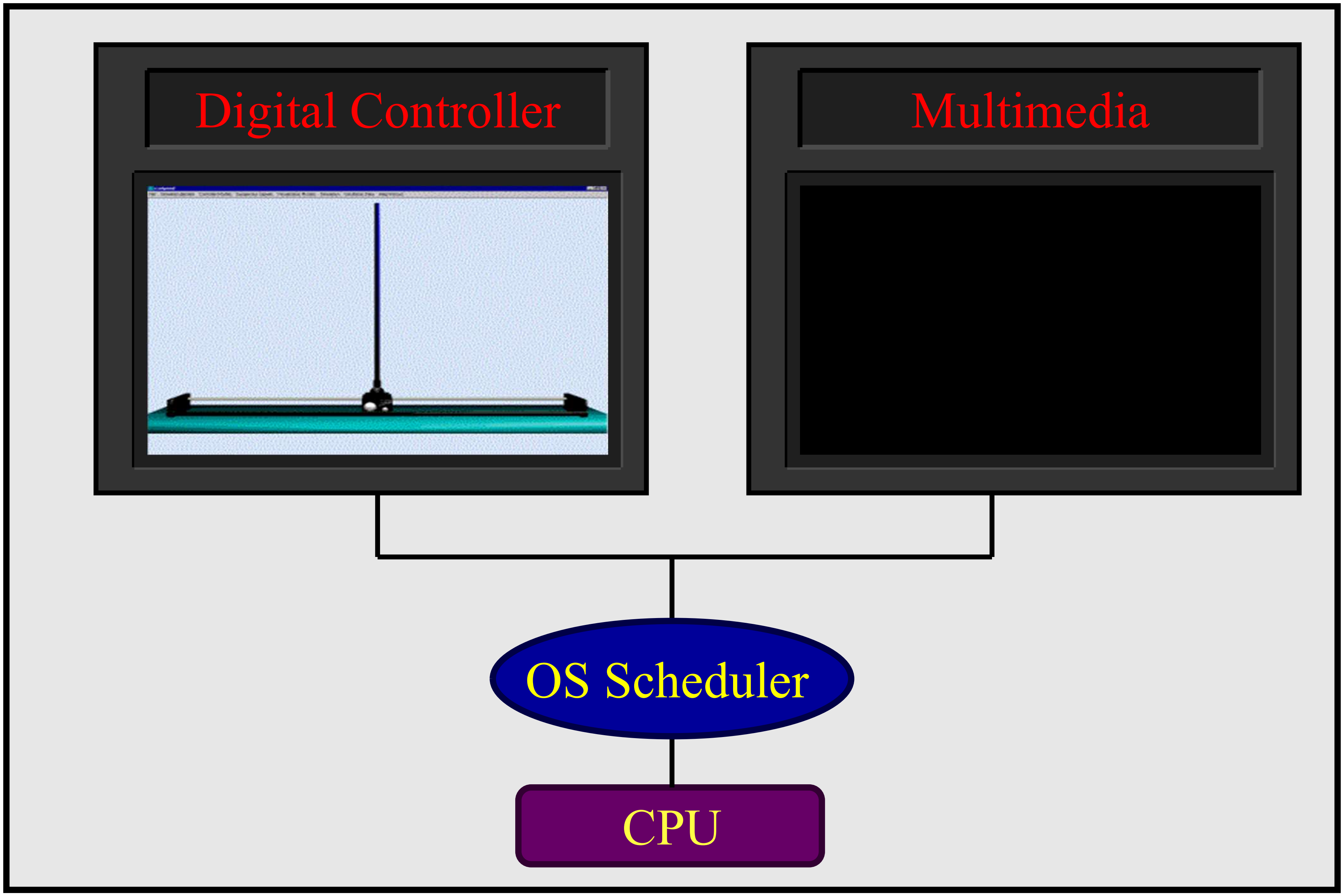
- Multimedia applications
  - periodically performs the following job:

    reads, decompresses, and displays video and audio streams

Fundamental Real-Time Issue

- To specify the timing constraints of real-time systems
- To achieve predictability on satisfying their timing constraints, possibly, with the existence of other real-time systems

Scheduling Framework Example



Preemption

- The running task can be interrupted at any point, so that a more important task that arrives can immediately gain the processor.
- The to-be-preempted task is interrupted and inserted to the ready queue, while CPU is assigned to the most important ready task which just arrived.
- Why preemption is needed in real-time systems?

Exception handling of a task

Treating with different criticalities of tasks, permits to anticipate the execution of the most critical activities

Efficient scheduling to improve system responsiveness

Notation of scheduling (1)

- $J = \{J_1,\ldots,J_n\}$ A set of tasks
- $\sigma:\mathbf{R}^+\to\mathbf{N}$ A schedule

A function mapping from time to task to assign task to CPU

If $\sigma(t)=i$ for $\forall t\in[t_1,t_2)$, task J_i is executed during time duration $[t_1,t_2)$.

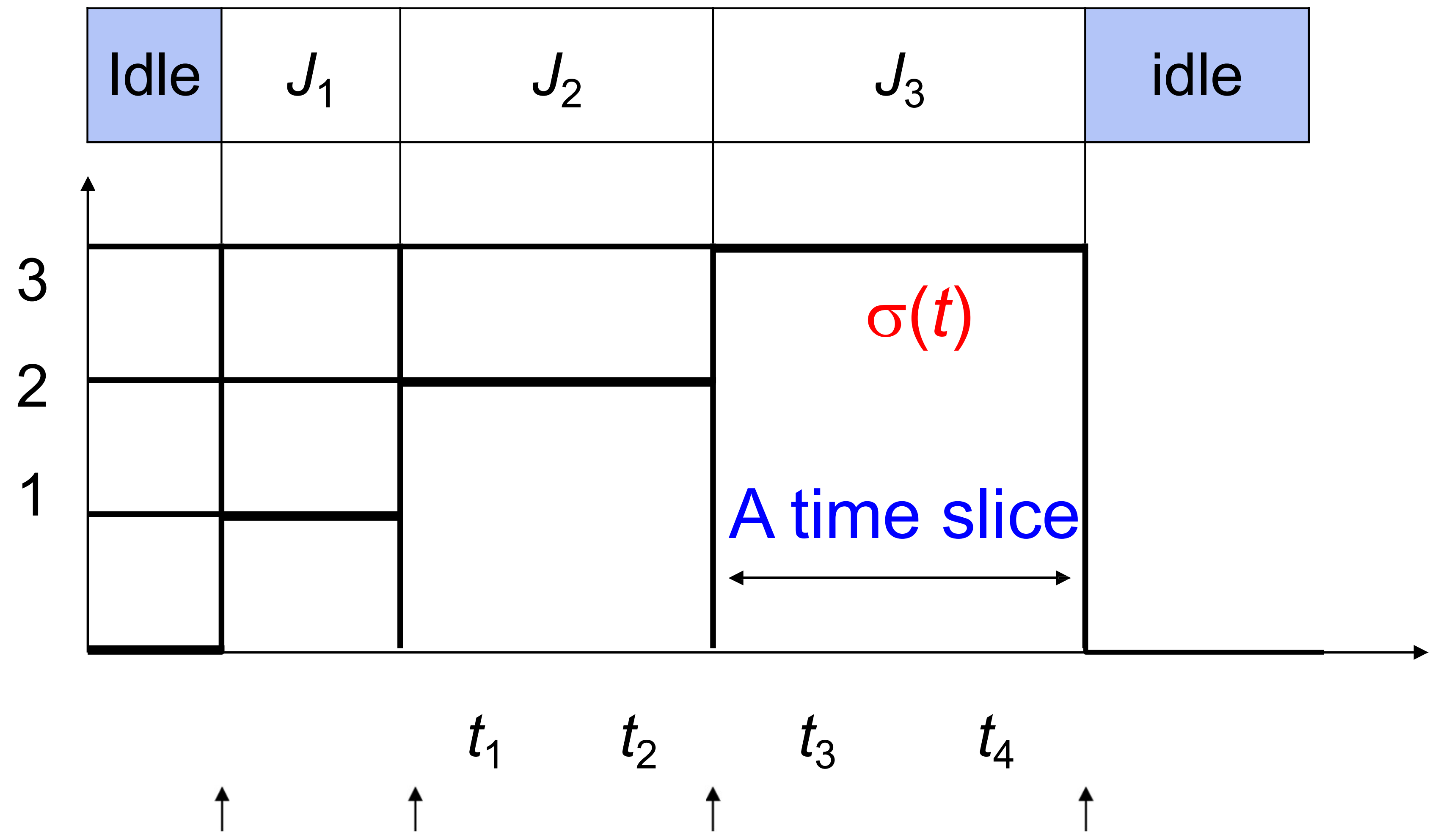
If $\sigma(t)=0$, the CPU is *idle*.

- Simple translation

CPU time is divided in to time slices $[t_1,t_2)$

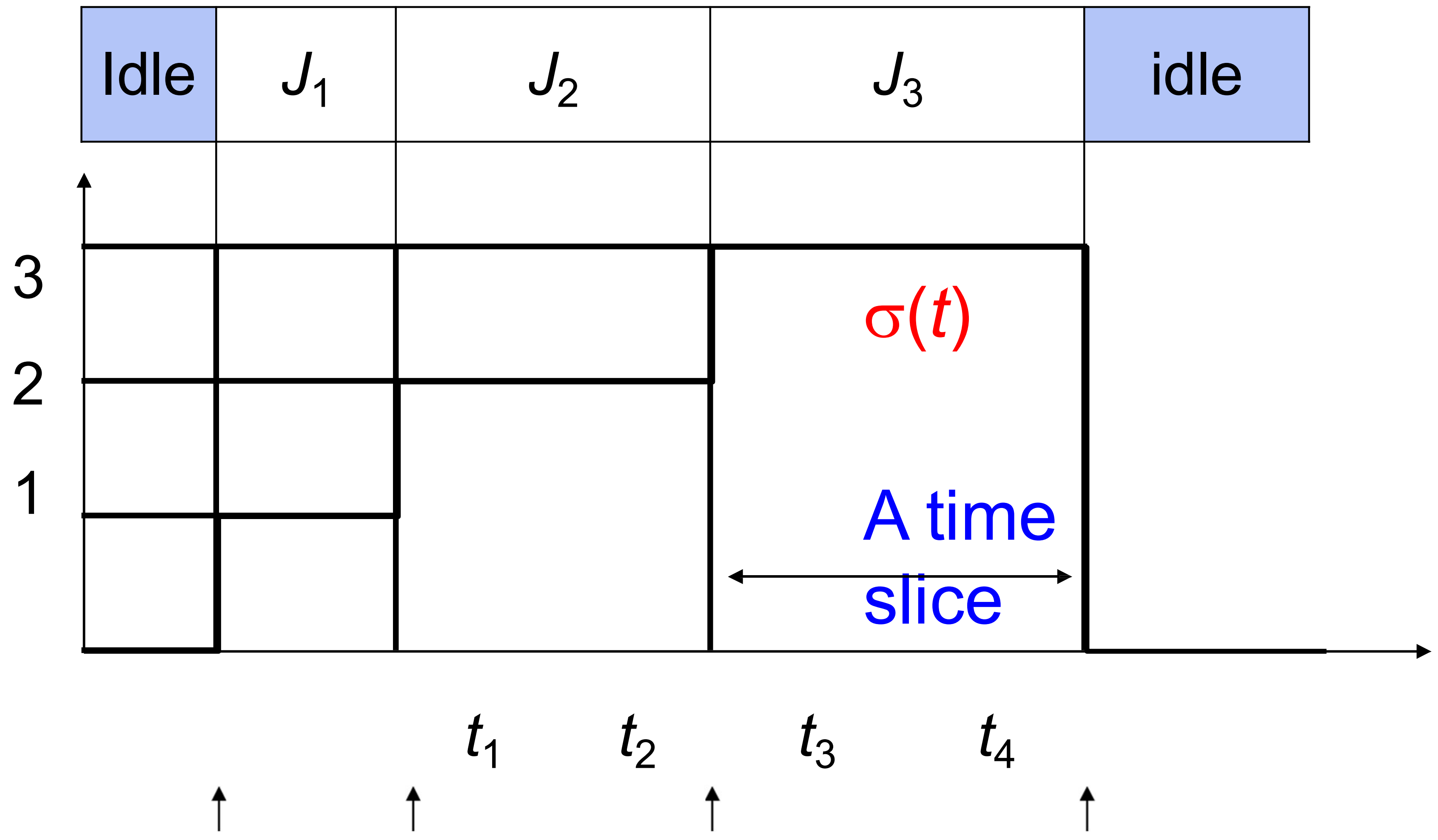
During a time slice $\sigma(t)$=const, representing the task that is executed

Notation of scheduling (2)



Context switching are performed at these times

Notation of scheduling (2)



Context switching are performed at these times

Notation of scheduling (3)

❑ Preemptive schedule

- A schedule in which the running task can be arbitrarily suspended at any time, to assign the CPU to another task

❑ Feasible schedule

- A schedule that all tasks can be completed according to a set of specified constraints

❑ Schedulable set of tasks

- A set of tasks that has at least one feasible schedule by some scheduling algorithm

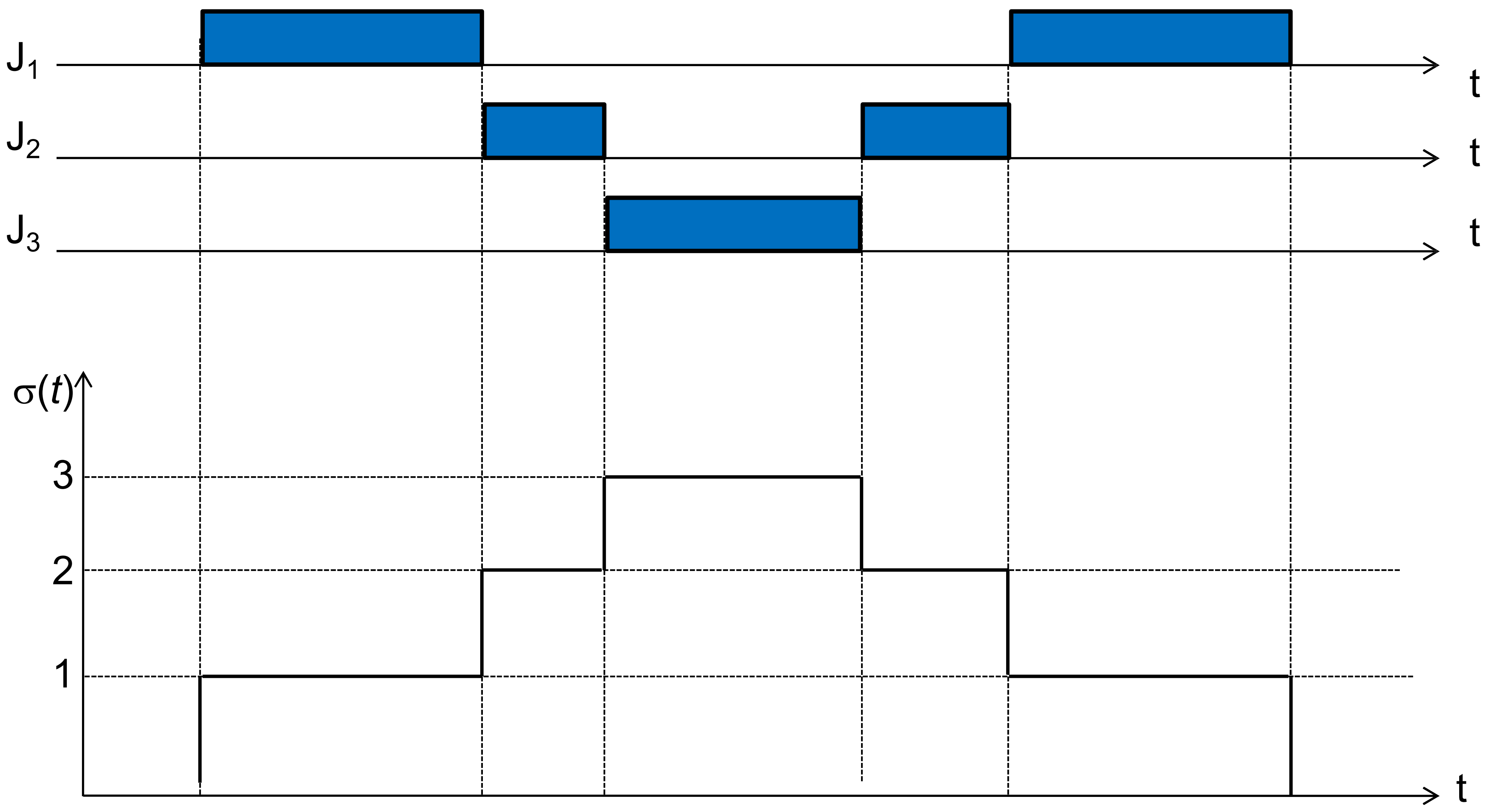
Notation of scheduling (3)

- Preemptive schedule
  - A schedule in which the running task can be arbitrarily suspended at any time, to assign the CPU to another task
- Feasible schedule
  - A schedule that all tasks can be completed according to
a set of specified constraints
- Schedulable set of tasks
  - A set of tasks that has at least one feasible schedule by some scheduling algorithm

Notation of scheduling (3)

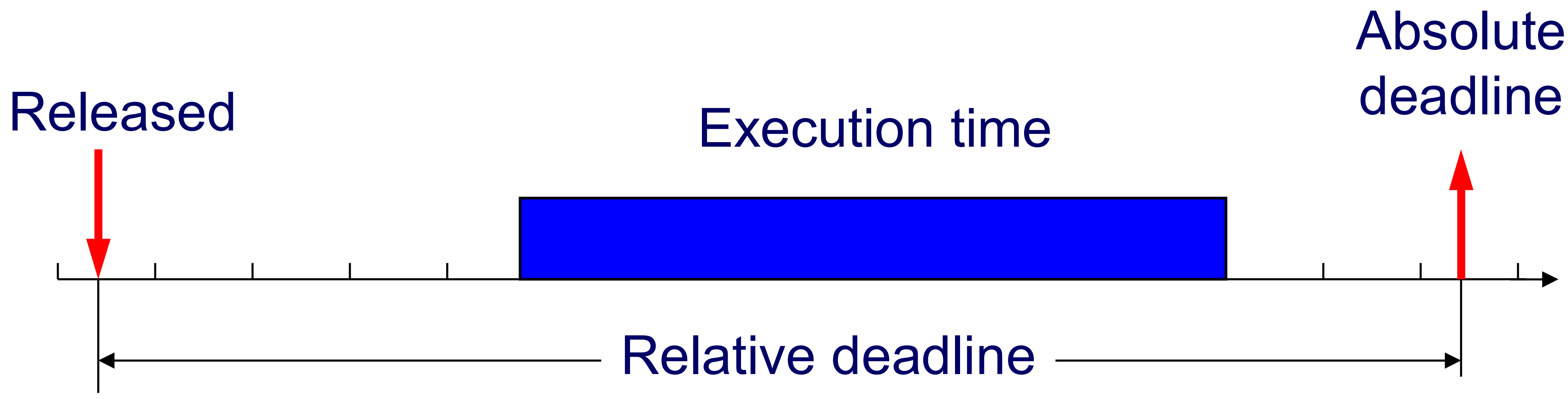
- Preemptive schedule
  - A schedule in which the running task can be arbitrarily suspended at any time, to assign the CPU to another task
- Feasible schedule
  - A schedule that all tasks can be completed according to
a set of specified constraints
- Schedulable set of tasks
  - A set of tasks that has at least one feasible schedule by some scheduling algorithm

Example of preemptive schedule



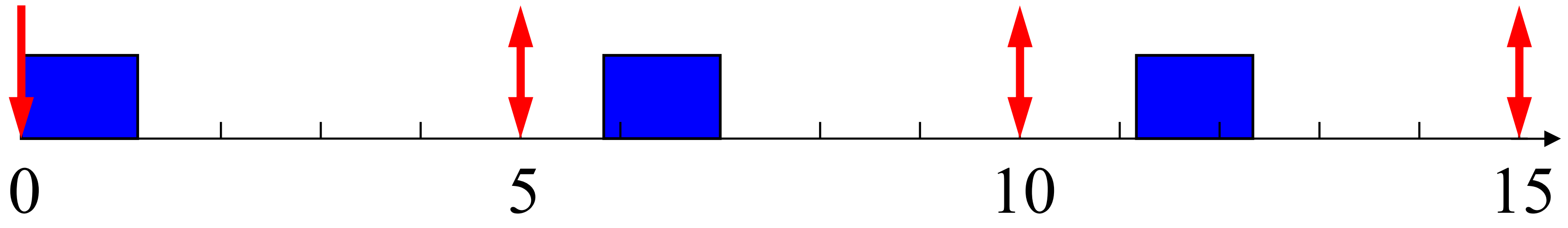
Real-Time Workload

- Job (unit of work)
  - a computation, a file read, a message transmission, etc
- Attributes
  - Resources required to make progress
  - Timing parameters

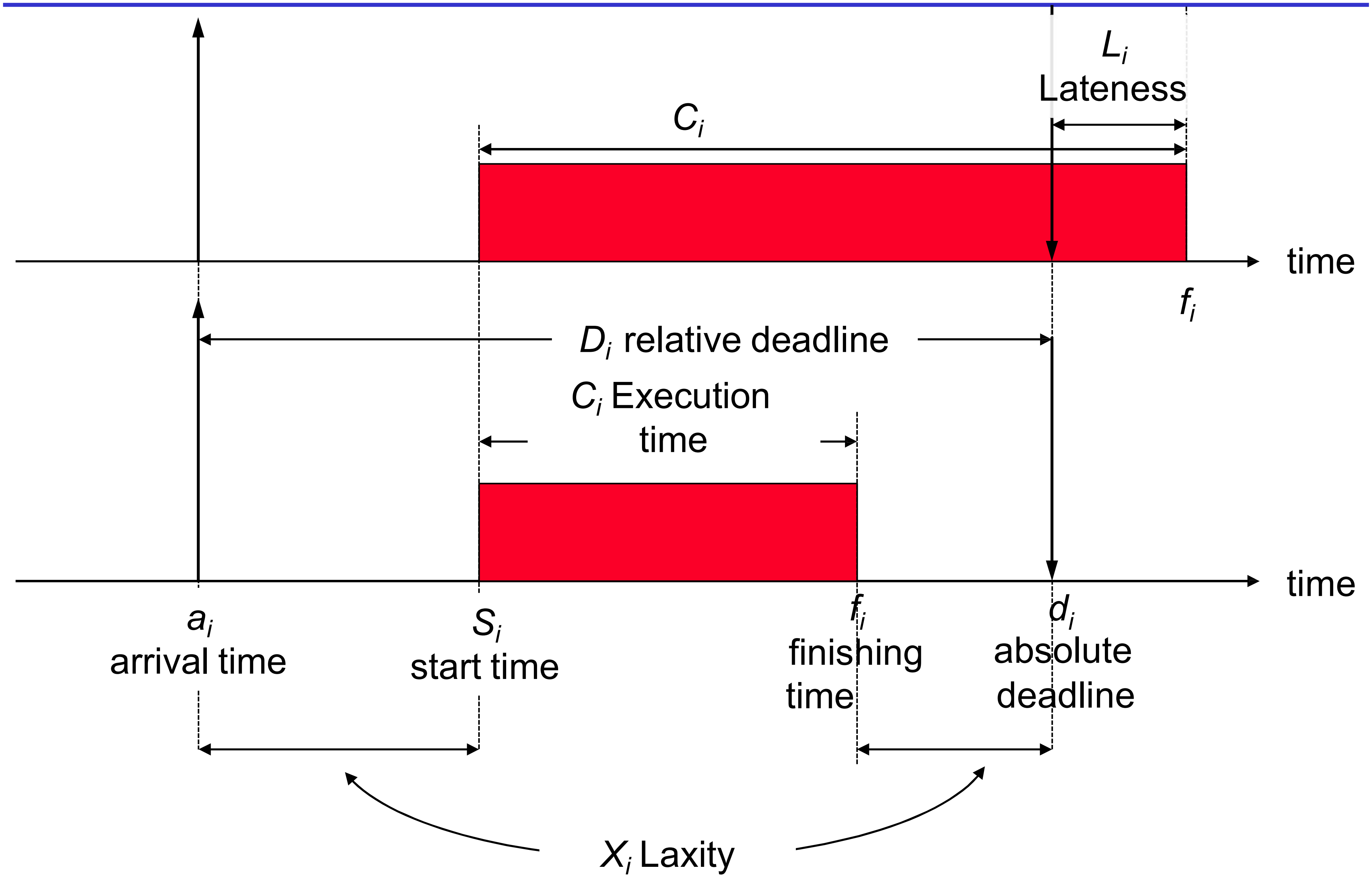


Real-Time Task

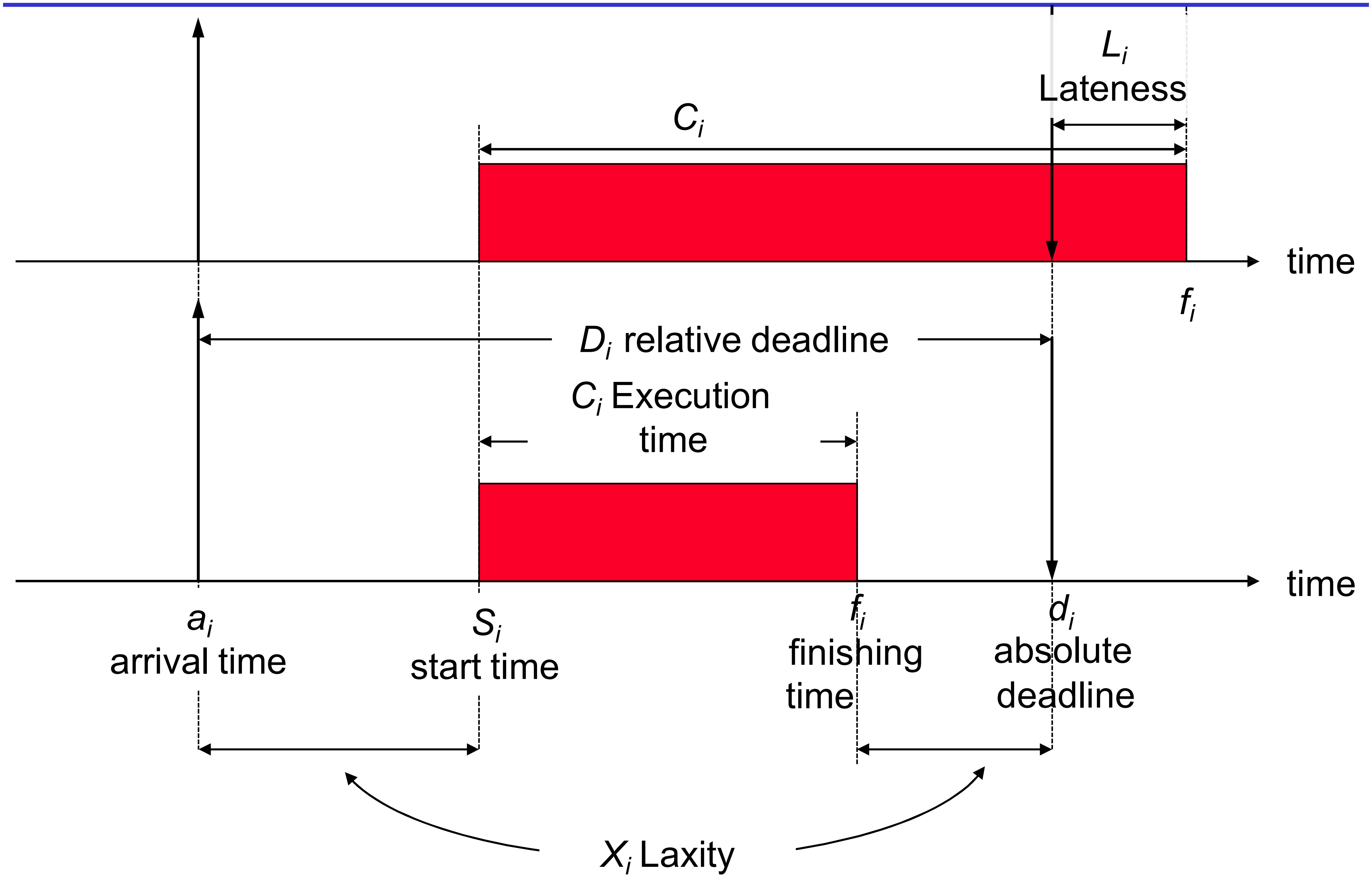
- Task : a sequence of similar jobs
  - Periodic task (*p*,*e*)
    - Its jobs repeat regularly
    - Period *p* = inter-release time ($0 < p$)
    - Execution time *e* = maximum execution time ($0 < e < p$)
    - Utilization U = e/p



Task parameters(1)



Task parameters(1)



Task parameters(2)

- Other task parameters
- Criticality: Hard or Soft
- Value v_i: relative importance of task with respect to the other tasks
- Lateness: the delay of a task completion with respect to its deadline $L_i = f_i - d_i$
- Tardiness or *Exceeding time*: $E_i = \max(0, L_i)$ is the time a task stays active after its deadline.
- Laxity or *Slack time* $X_i = d_i - a_i - C_i$ is the maximum time a task can be delayed on its activation to complete within its deadline

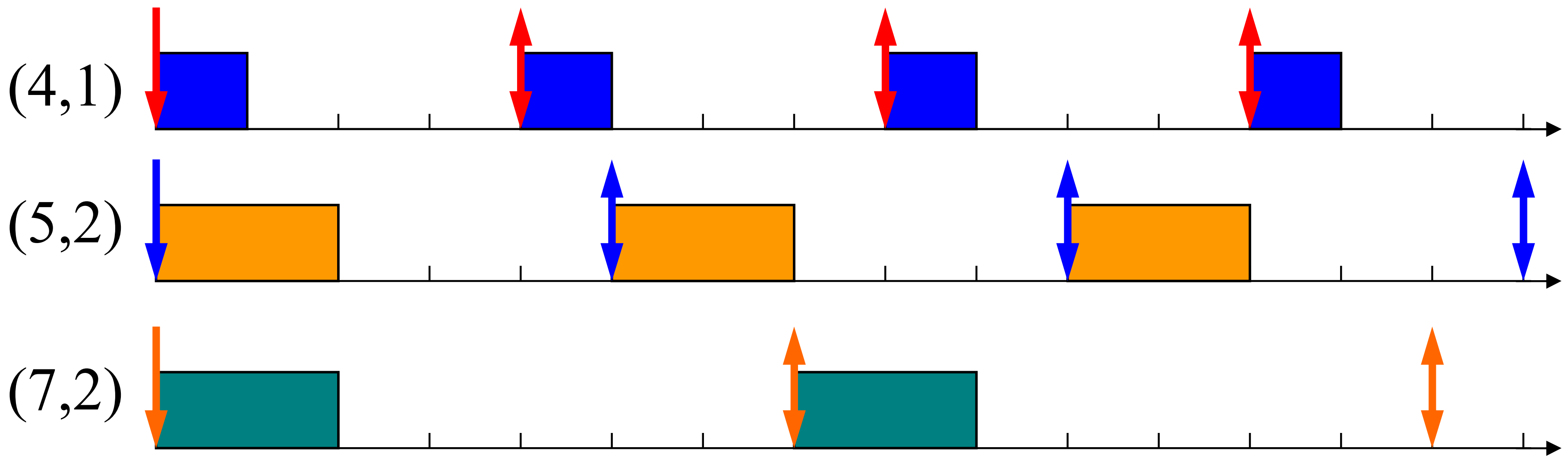
Deadlines: Hard vs. Soft

- Hard deadline
  - Disastrous or very serious consequences may occur if the deadline is missed
  - Validation is essential : can all the deadlines be met, even under worst-case scenario?
  - Deterministic guarantees

- Soft deadline
  - Ideally, the deadline should be met for maximum performance. The performance degrades in case of deadline misses.
  - Best effort approaches / statistical guarantees

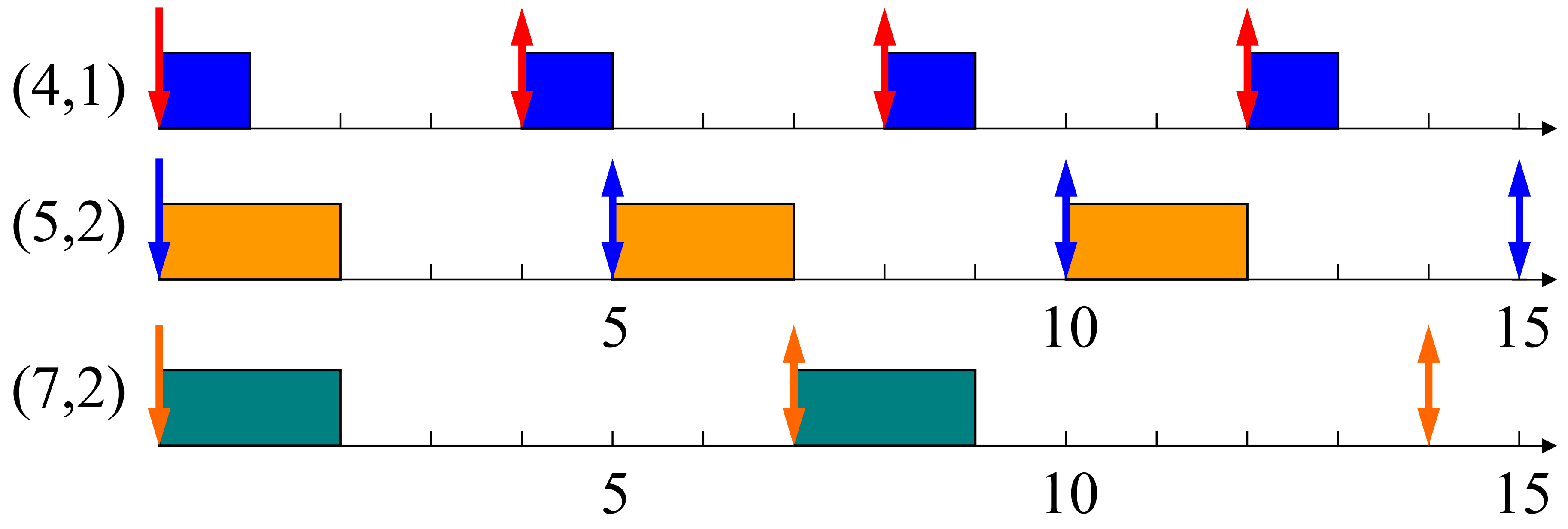
Schedulability

- Property indicating whether a real-time system (a set of real-time tasks) can meet their deadlines



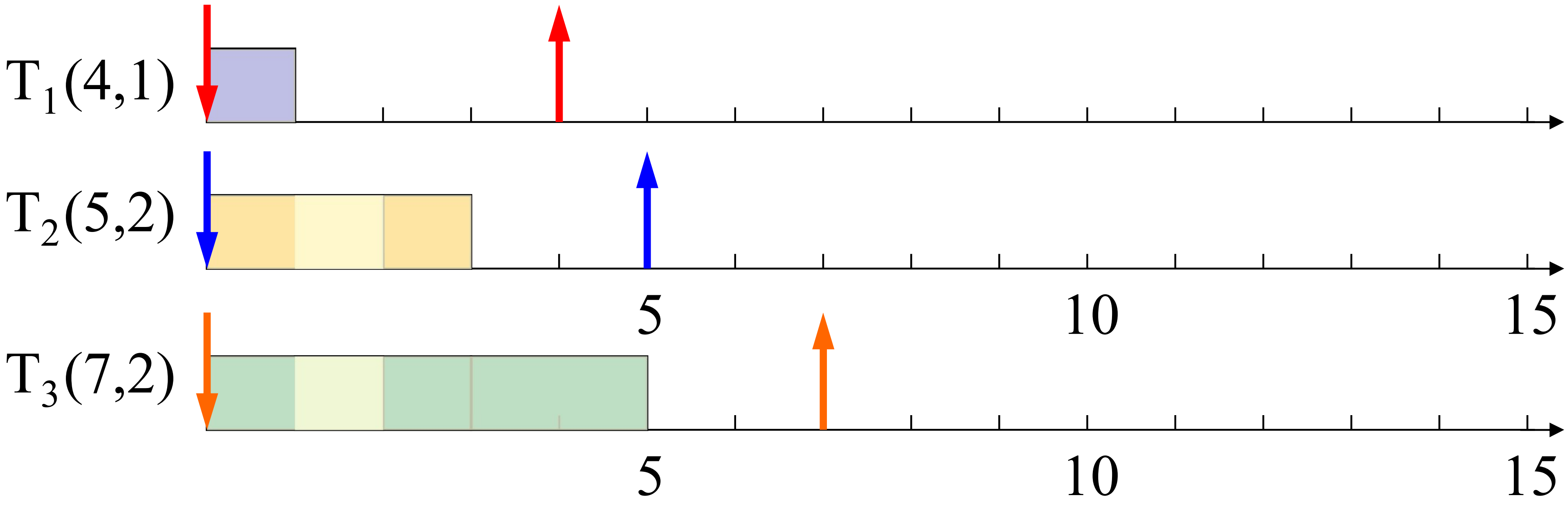
Real-Time Scheduling

- Determines the order of real-time task executions
- Static-priority scheduling
- Dynamic-priority scheduling



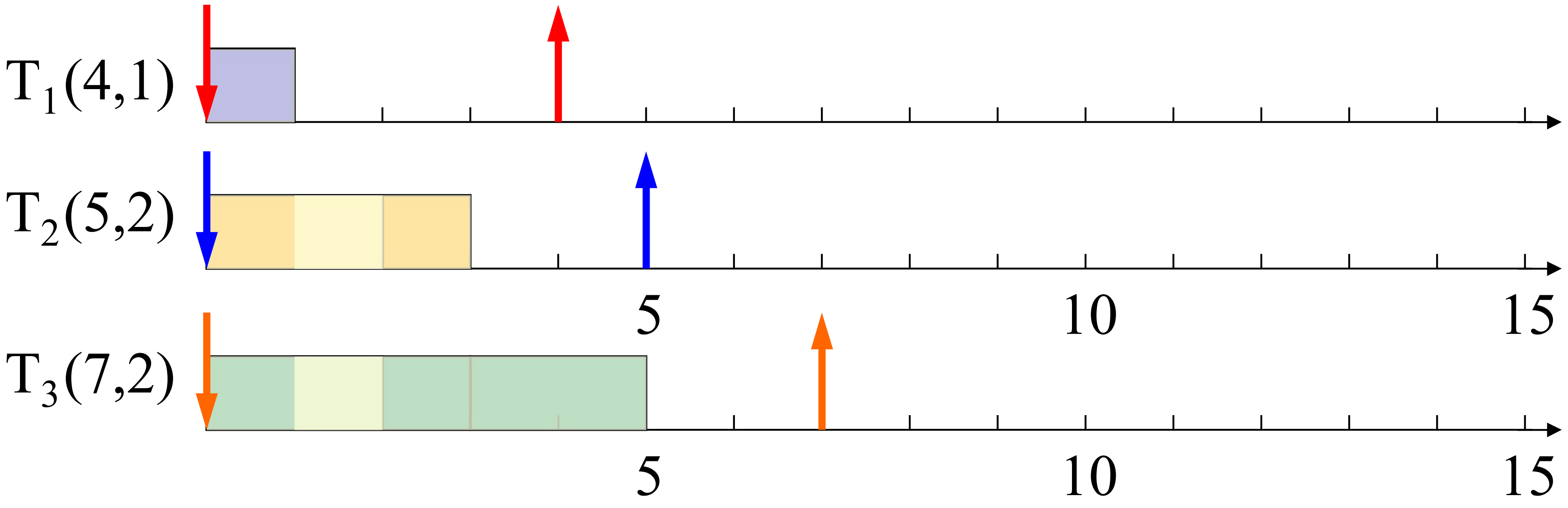
RM (Rate Monotonic)

- Optimal static-priority scheduling
- It assigns priority according to period
- A task with a shorter period has a higher priority
- Executes a job with the shortest period



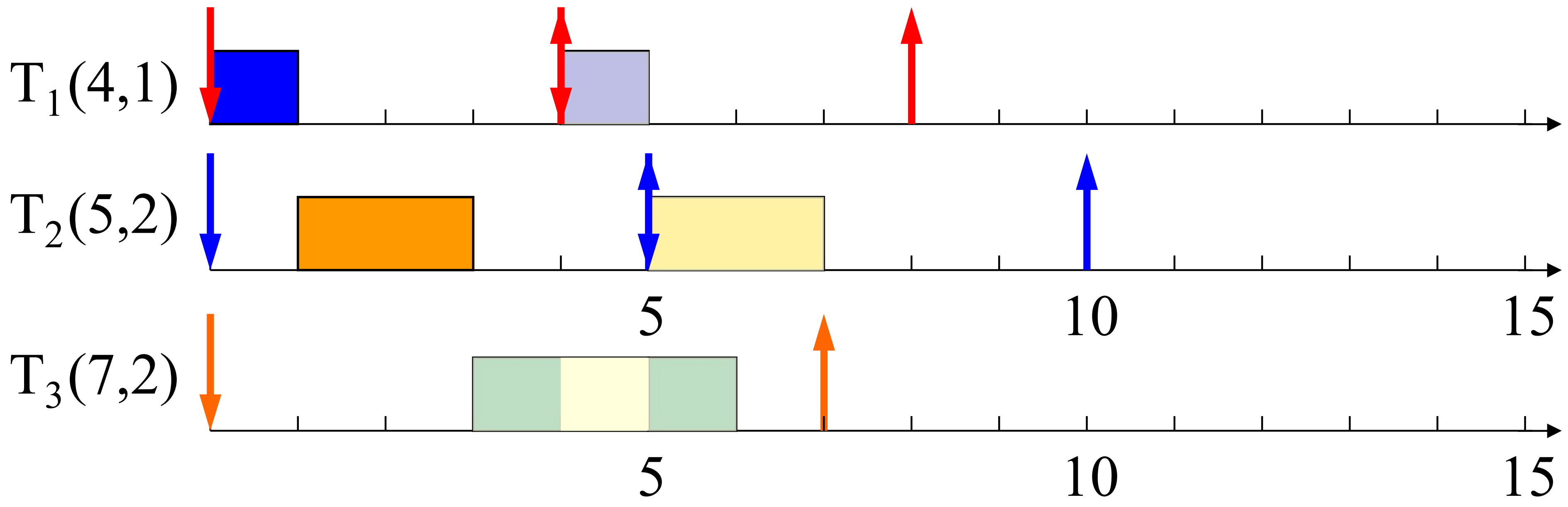
RM (Rate Monotonic)

- Optimal static-priority scheduling
- It assigns priority according to period
- A task with a shorter period has a higher priority
- Executes a job with the shortest period



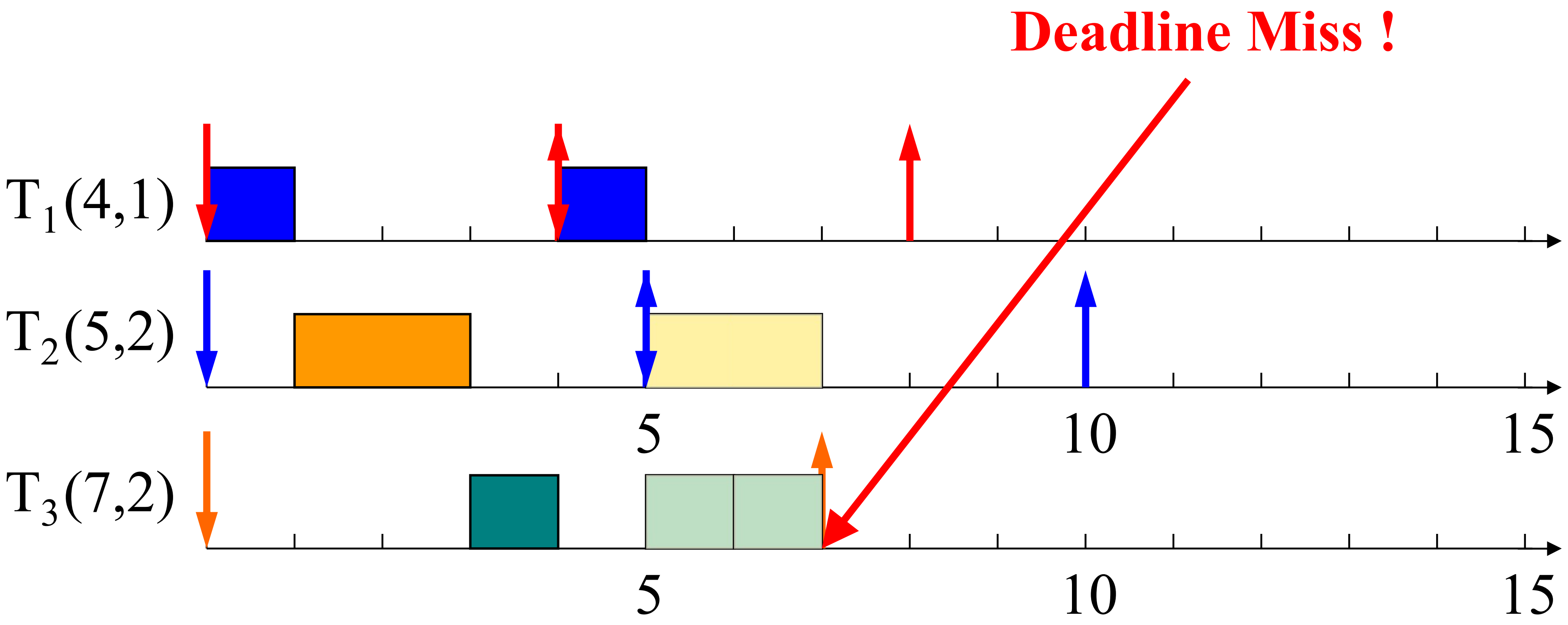
RM (Rate Monotonic)

- Executes a job with the shortest period



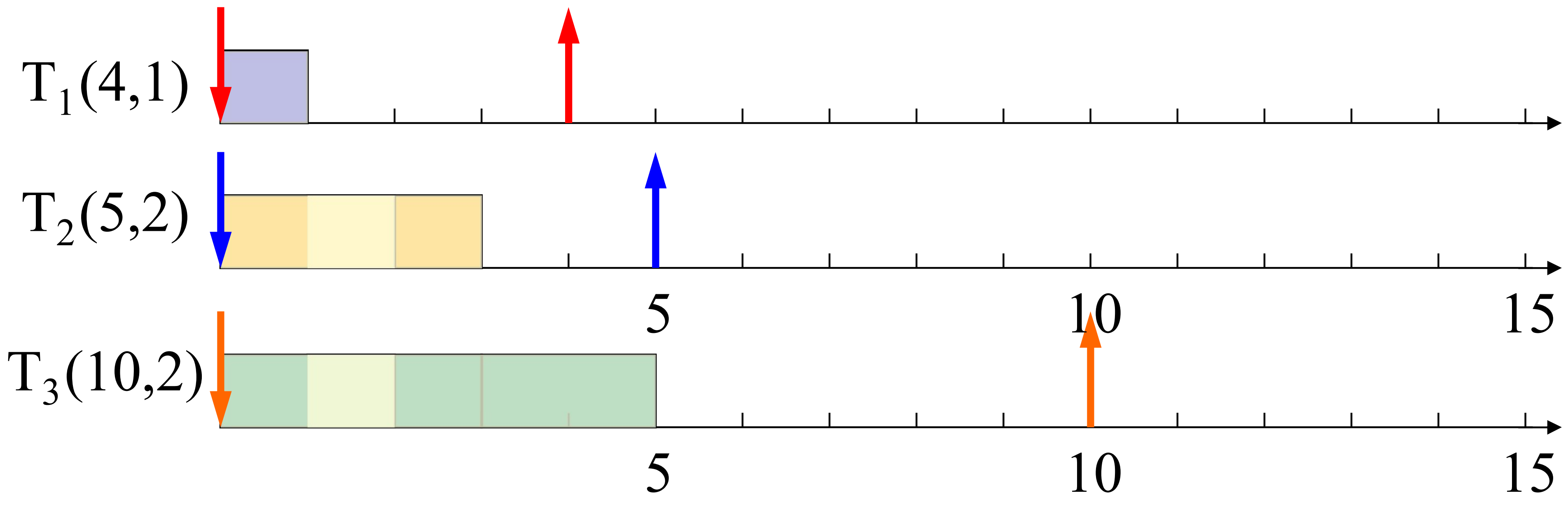
RM (Rate Monotonic)

- Executes a job with the shortest period



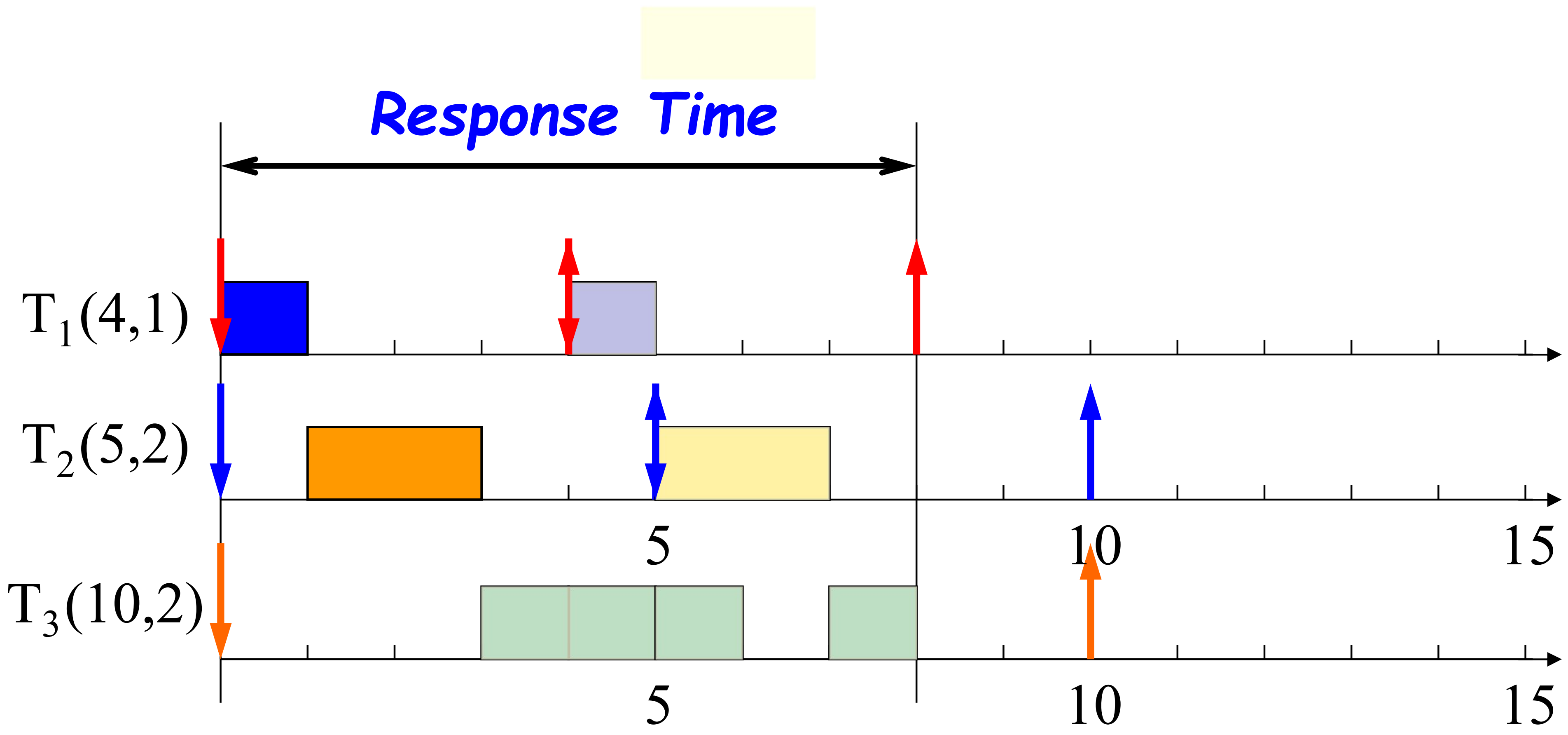
Response Time

- Response time
  - Duration from released time to finish time



Response Time

- Response time
  - Duration from released time to finish time

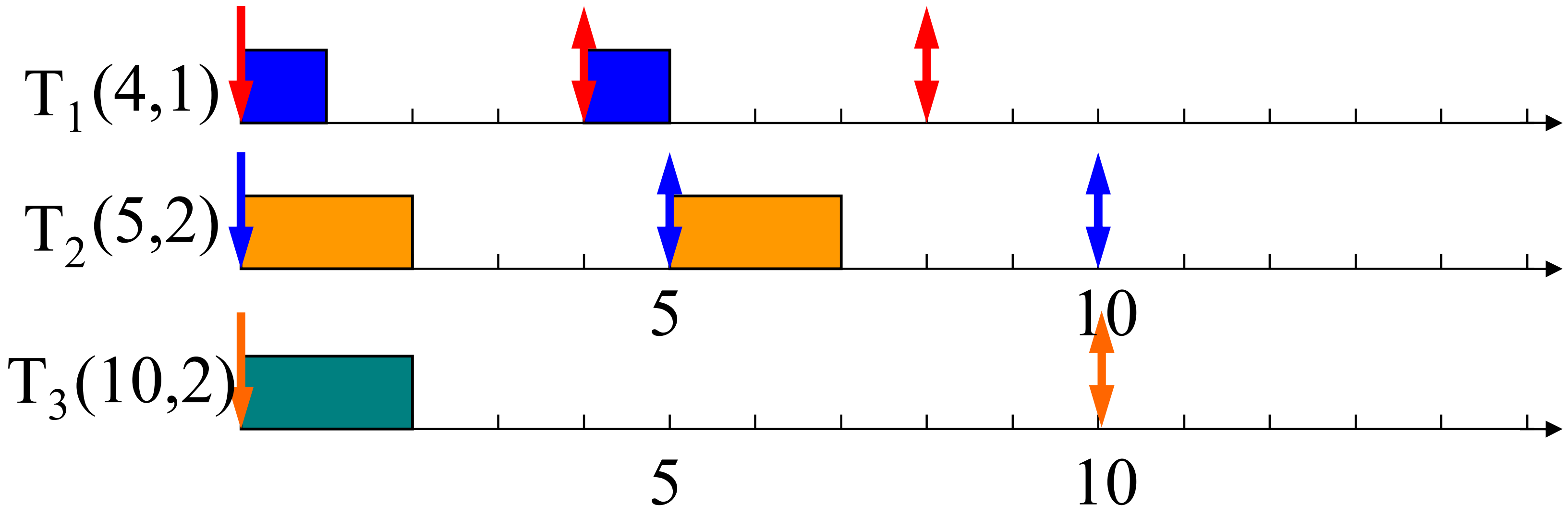


Response Time

- Response Time (r_i) [Audsley et al., 1993]

$$r_i = e_i + \sum_{T_k \in HP(T_i)} \left\lceil \frac{r_i}{p_k} \right\rceil \cdot e_k$$

- HP(T_i) : a set of higher-priority tasks than T_i

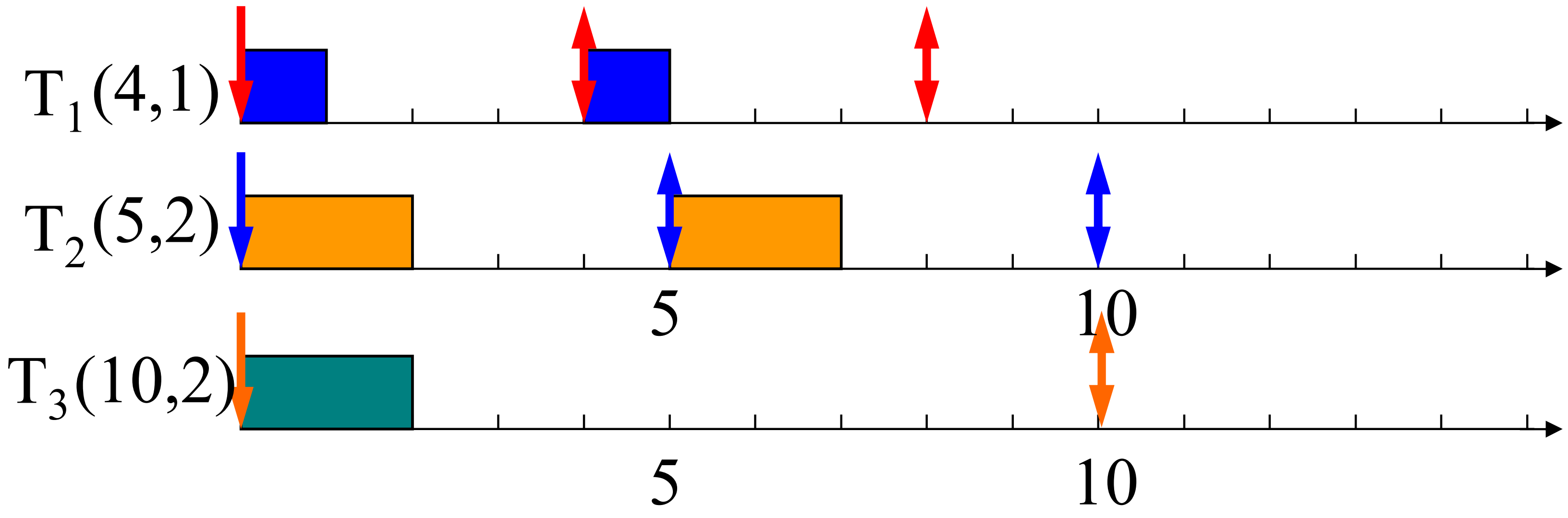


Response Time

- Response Time (r_i) [Audsley et al., 1993]

$$r_i = e_i + \sum_{T_k \in HP(T_i)} \left\lceil \frac{r_i}{p_k} \right\rceil \cdot e_k$$

- HP(T_i) : a set of higher-priority tasks than T_i



RM - Schedulability Analysis

- Real-time system is schedulable under RM

  if and only if $r_i \le p_i$ for all task $T_i(p_i,e_i)$

Joseph & Pandya,
“Finding response times in a real-time system”,
The Computer Journal, 1986.

RM – Utilization Bound

- Real-time system is schedulable under RM if

$$\sum U_i \leq n\,(2^{1/n}-1)$$

Liu & Layland,
"Scheduling algorithms for multi-programming in a hard-real-time environment", *Journal of ACM*, 1973.

RM – Utilization Bound

- Real-time system is schedulable under RM if

$$\sum U_i \le n\,(2^{1/n}-1)$$

Liu & Layland,
"Scheduling algorithms for multi-programming in a hard-real-time environment", *Journal of ACM*, 1973.

RM – Utilization Bound

- Real-time system is schedulable under RM if

$$\sum U_i \leq n\,(2^{1/n}-1)$$

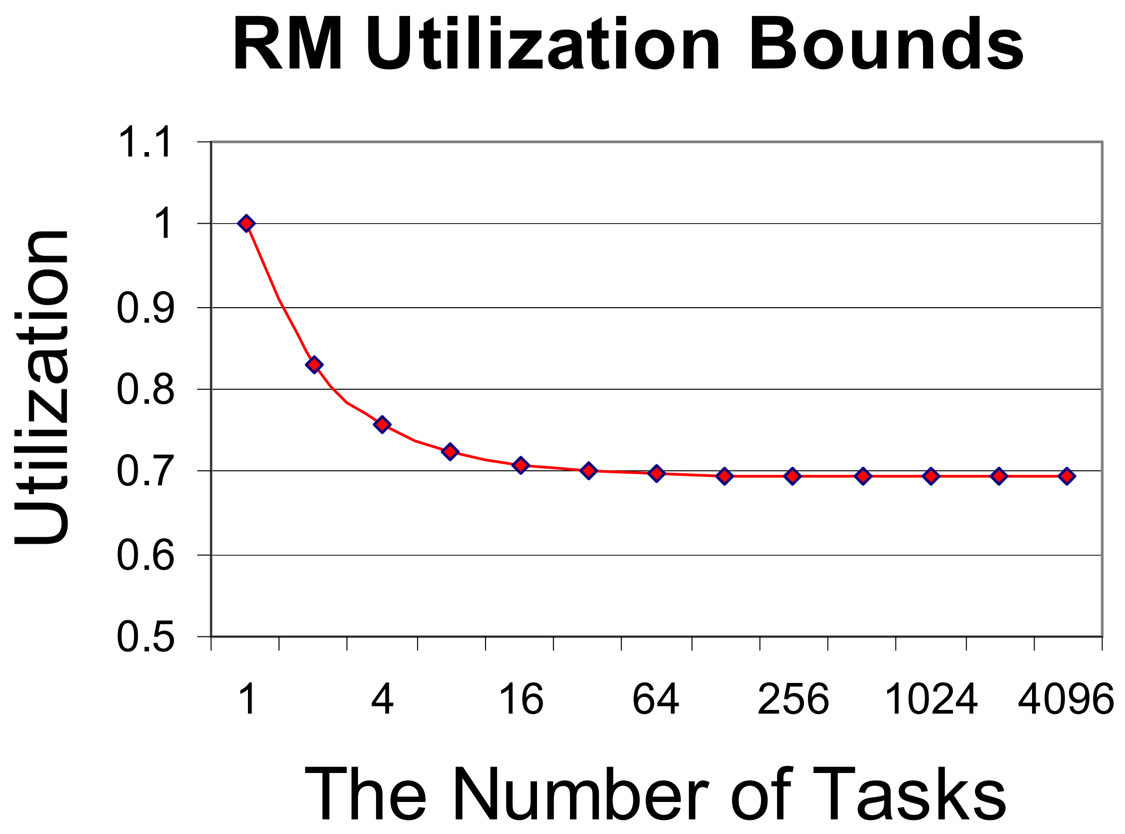
- Example: $T_1(4,1)$, $T_2(5,1)$, $T_3(10,1)$,

$$\sum U_i = 1/4 + 1/5 + 1/10$$
$$= 0.55$$
$$3\,(2^{1/3}-1) \approx 0.78$$

Thus, $\{T_1, T_2, T_3\}$ is schedulable under RM.

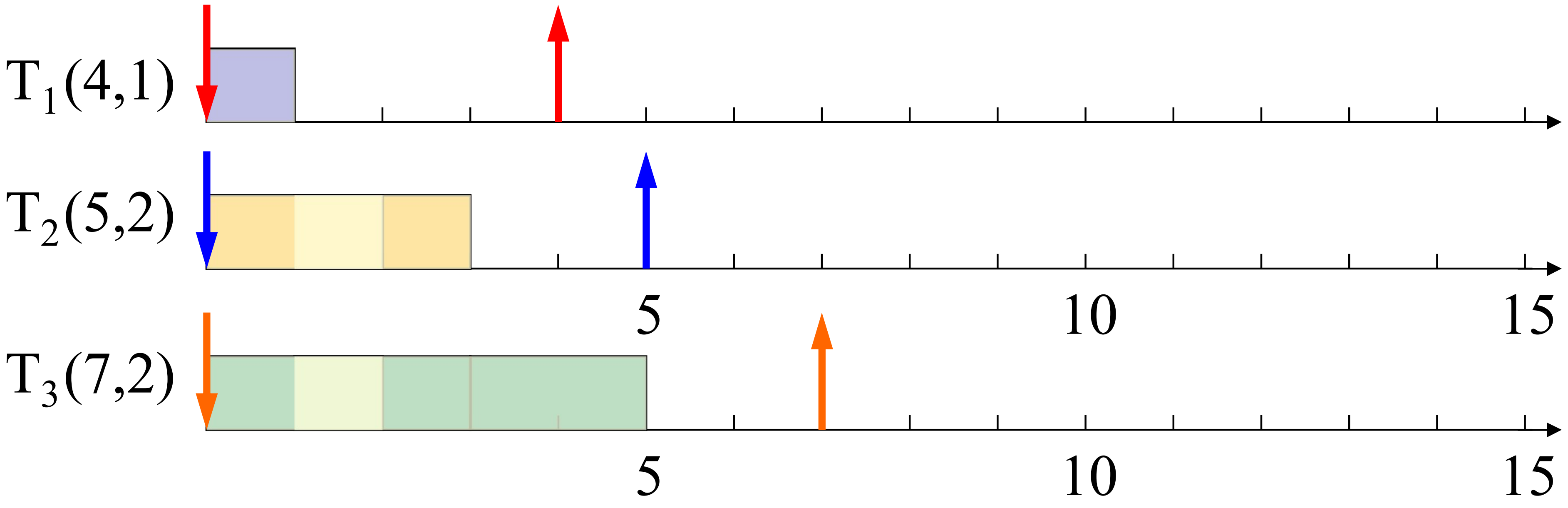
RM – Utilization Bound

- Real-time system is schedulable under RM if
  $\sum U_i \leq n\,(2^{1/n}-1)$



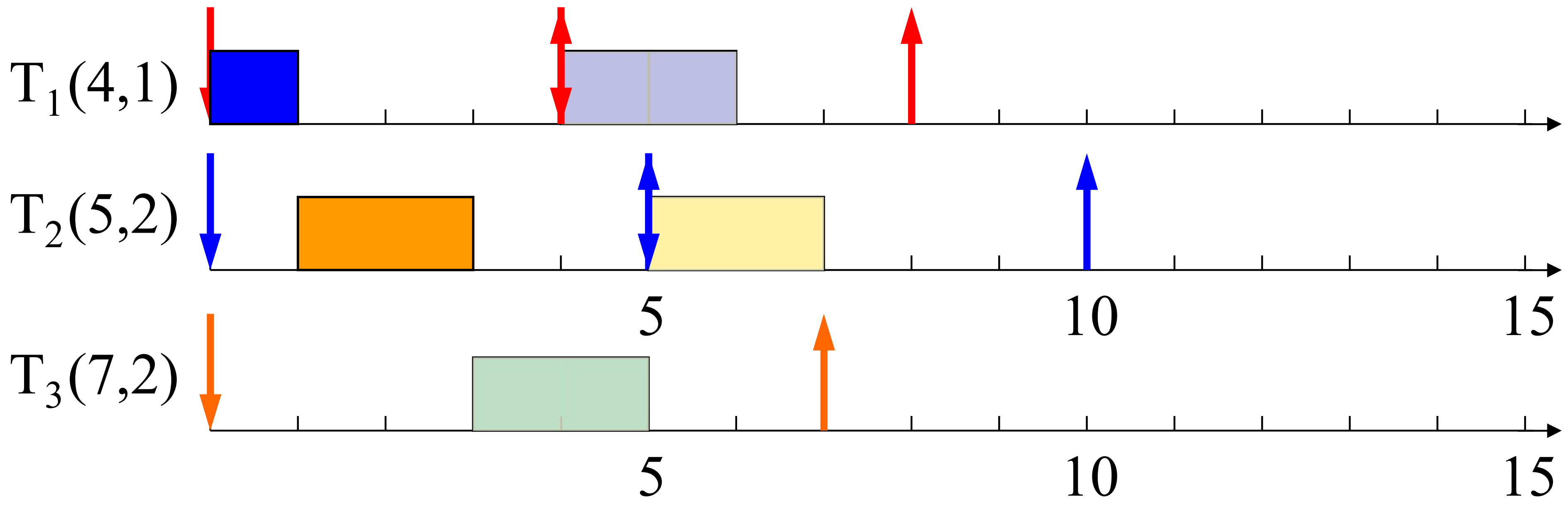
EDF (Earliest Deadline First)

- Optimal dynamic priority scheduling
- A task with a shorter deadline has a higher priority
- Executes a job with the earliest deadline



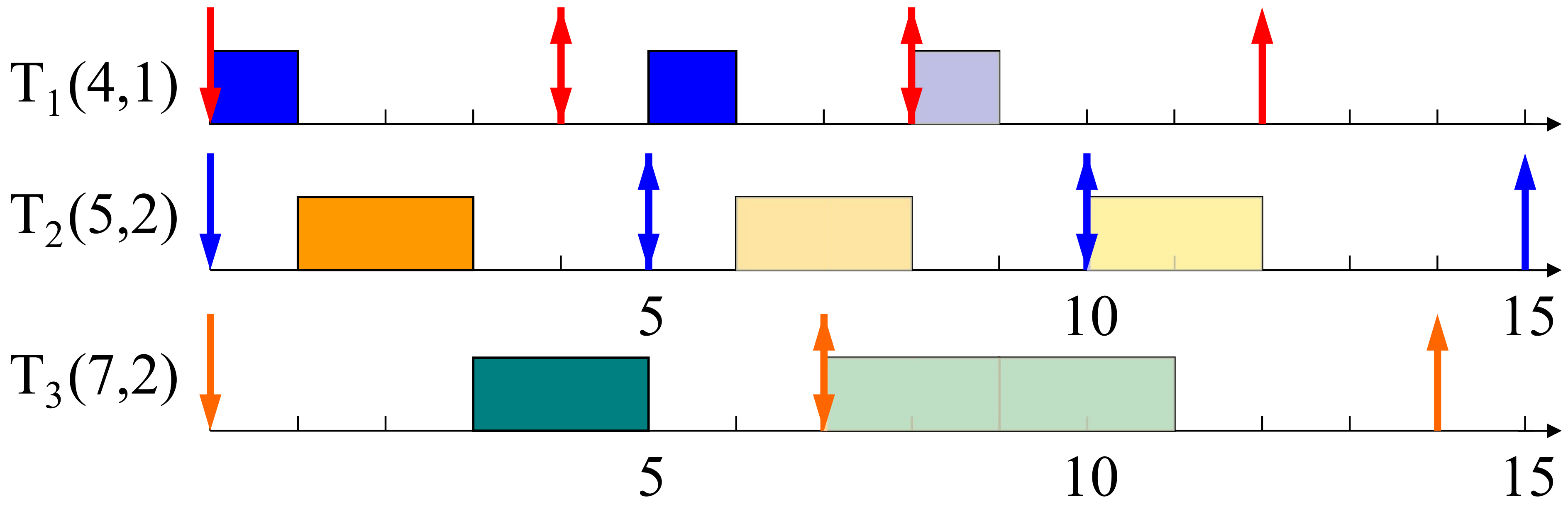
EDF (Earliest Deadline First)

- Executes a job with the earliest deadline



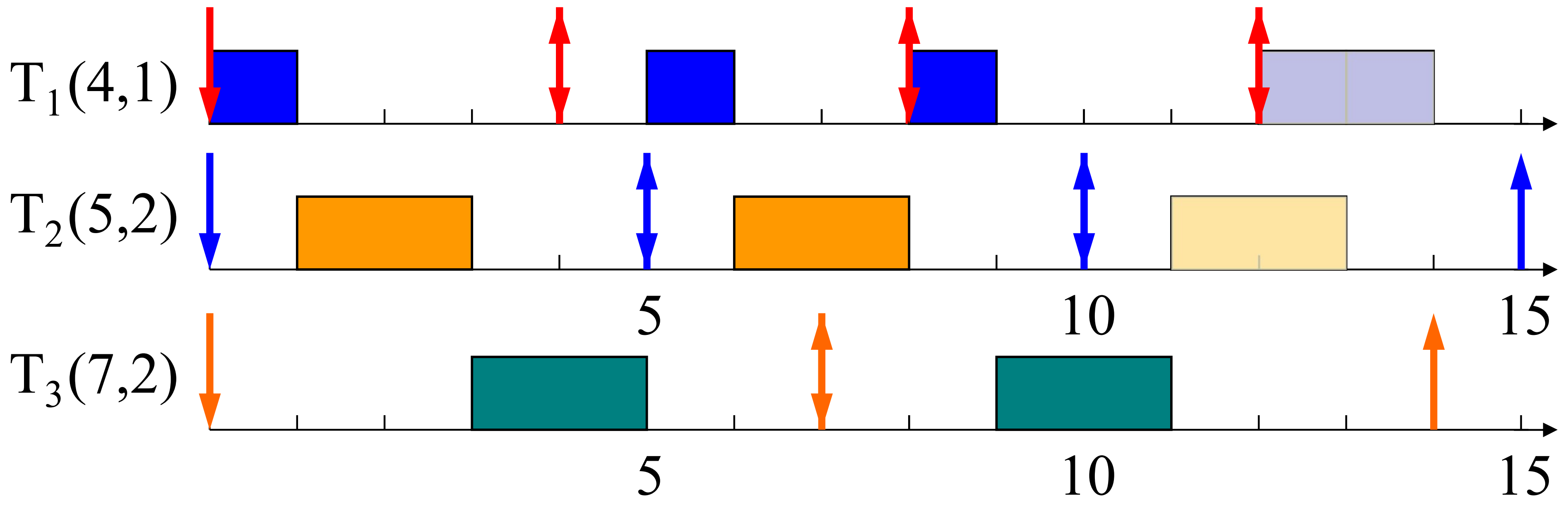
EDF (Earliest Deadline First)

- Executes a job with the earliest deadline



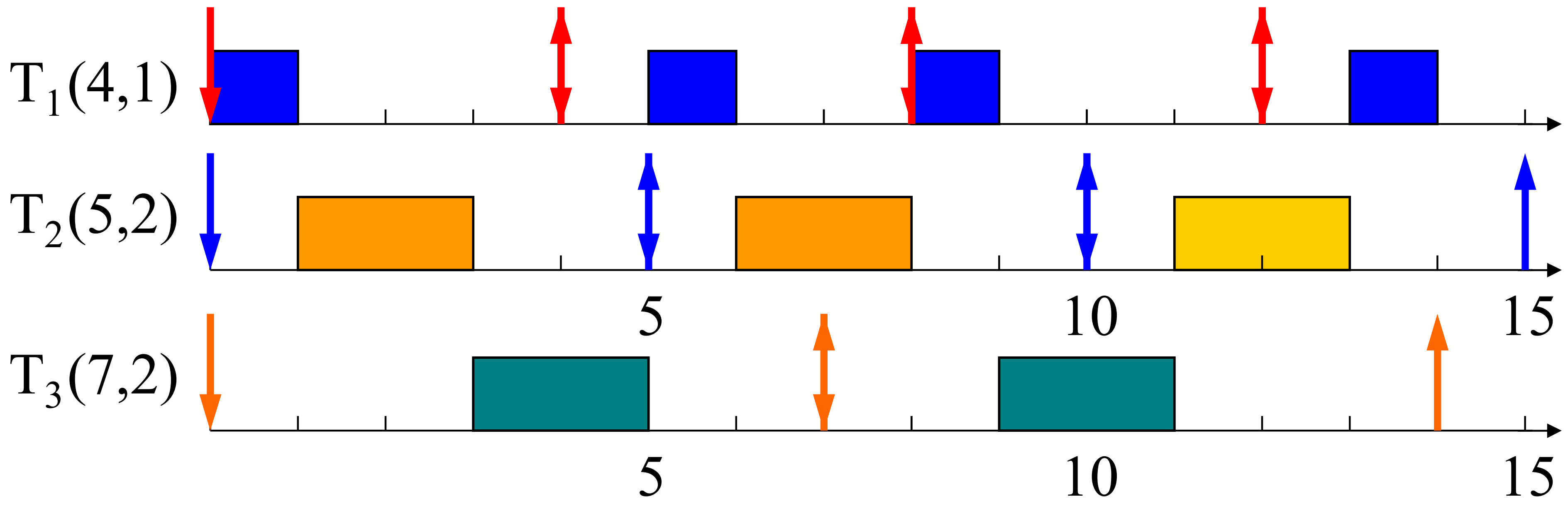
EDF (Earliest Deadline First)

- Executes a job with the earliest deadline



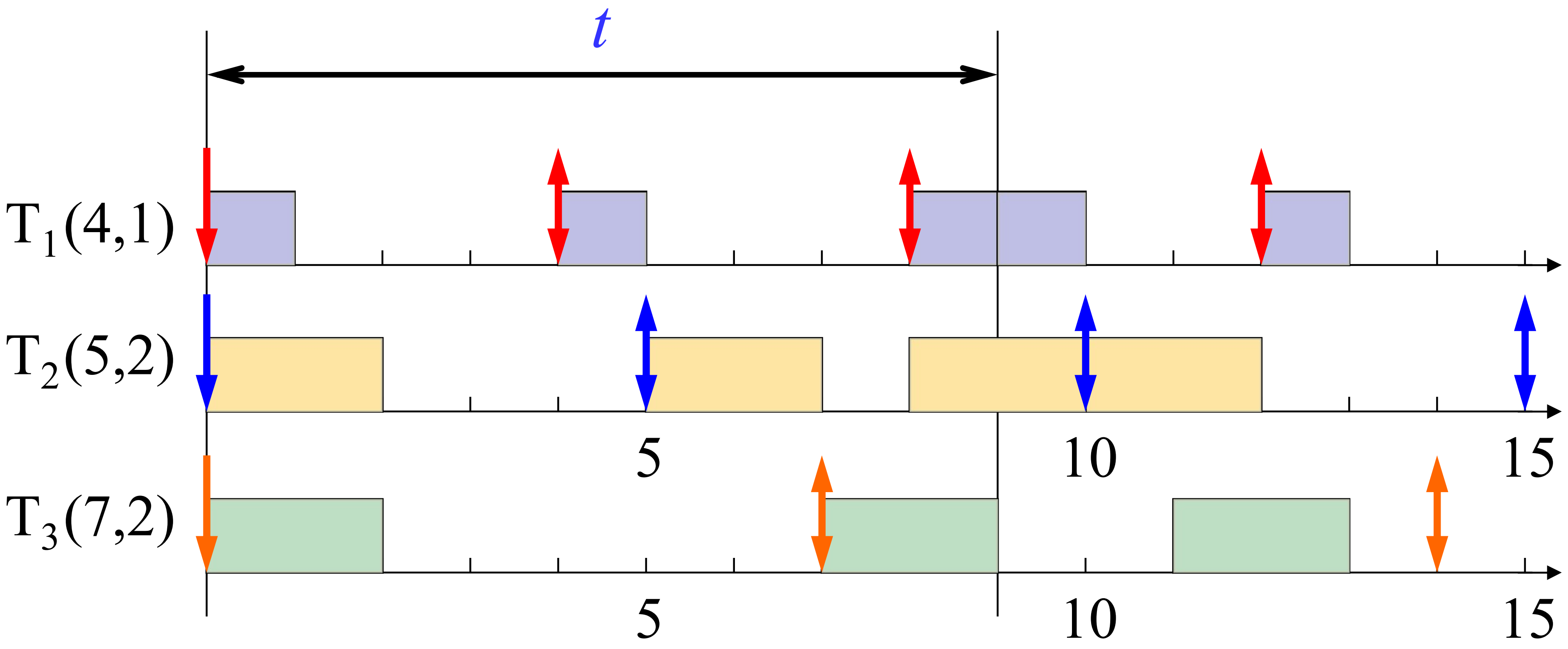
EDF (Earliest Deadline First)

- Optimal scheduling algorithm
  - if there is a schedule for a set of real-time tasks, EDF can schedule it.



Processor Demand Bound

- Demand Bound Function : *dbf(t)*
  - the maximum processor demand by workload over any interval of length t



EDF - Schedulability Analysis

- Real-time system is schedulable under EDF
  if and only if **dbf(t) ≤ t** for all interval t

  Baruah et al.
  "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor", Journal of Real-Time Systems, 1990.

- Demand Bound Function : dbf(t)
  - the maximum processor demand by workload over any interval of length t

EDF – Utilization Bound

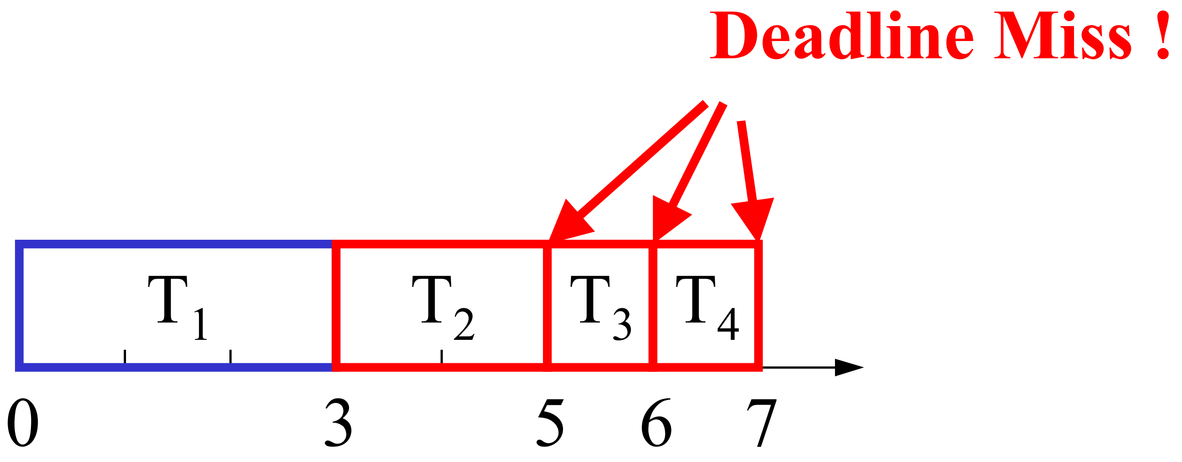
- Real-time system is schedulable under EDF if and only if

$$\sum U_i \leq 1$$

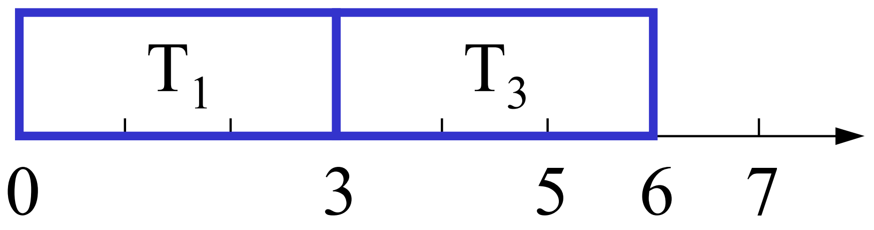
Liu & Layland,
"Scheduling algorithms for multi-programming in a hard-real-time environment", *Journal of ACM*, 1973.

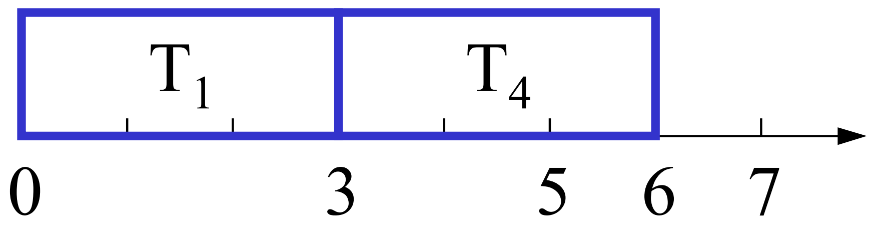
EDF – Overload Conditions

- Domino effect during overload conditions
  - Example: $T_1(4,3)$, $T_2(5,3)$, $T_3(6,3)$, $T_4(7,3)$



Better schedules :





Two common scheduling schemes

- **Rate monotonic (RM)**
  - Static priority scheme
  - Preemption required
  - Simple to implement
  - Nice properties
- **Earliest deadline first (EDF)**
  - Dynamic priority scheme
  - Preemption required
  - Harder to implement
  - Very nice properties

Sharing resources

- Need some kind of a lock on a resource.
  - If a high priority task finds a resource is locked, it goes to sleep until the resource is available.
  - Task is woken up when resource is freed by lower priority task.
  - Sounds reasonable, but leads to problems.

- More formally stated on next slide.

Priority Inversion

- In a preemptive priority based real-time system, sometimes tasks may need to access resources that cannot be shared.
  - The method of ensuring exclusive access is to guard the critical sections with binary semaphores.
  - When a task seeks to enter a critical section, it checks if the corresponding semaphore is locked.
  - If it is not, the task locks the semaphore and enters the critical section.
  - When a task exits the critical section, it unlocks the corresponding semaphore.
- This could cause a high priority task to be waiting on a lower priority one.
  - Even worse, a medium priority task might be running and cause the high priority task to not meet its deadline!

<u>**Example**</u>: Priority inversion

- Low priority task "C" locks resource "Z".
- High priority task "A" preempts "C" then requests resource "Z"
  - Deadlock, but solvable by having "A" sleep until resource is unlocked.
- But if medium priority "B" were to run, it would preempt C, thus effectively making C <u>and</u> A run with a lower priority than B.
  - Thus priority *inversion.*

Solving Priority inversion

- Priority Inheritance
  - When a high priority task sleeps because it is waiting on a lower priority task, have it boost the priority of the blocking task to its own priority.