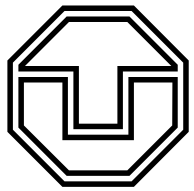
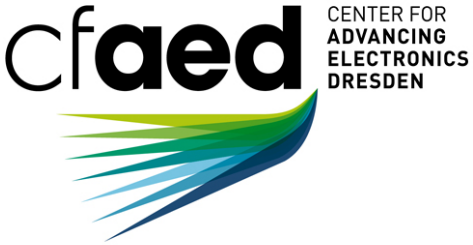
Networked Embedded Systems WS 2016/17

Lecture 2: Real-time Scheduling

Marco Zimmerling



TECHNISCHE UNIVERSITÄT DRESDEN



Goal of Today's Lecture

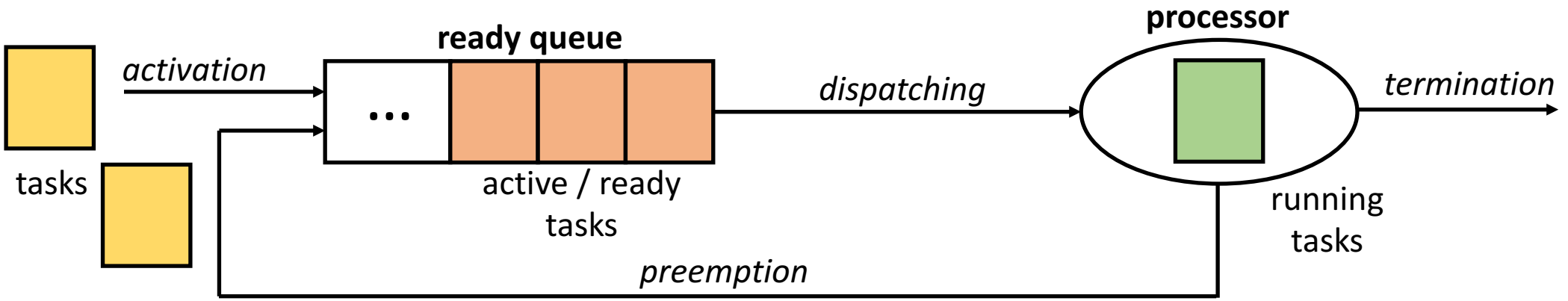
- Introduction to scheduling of compute tasks on a single processor
  - Tasks need to finish before specified deadlines
  - Preemptive real-time scheduling algorithms for periodic and aperiodic tasks
- Scheduling is everywhere
  - Processes in operating system kernel (*e.g.*, Completely Fair Scheduler in Linux)
  - Large compute jobs in a datacenter (*e.g.*, Apache Mesos)
  - Packet flows in a middlebox (*e.g.*, providing filtering, firewalling, VPN, etc.)
  - …
- Other related topics include load balancing and queuing

Literature on Real-time Scheduling

- Recommended:
  - Giorgio C. Buttazzo, *Hard Real-time Computing Systems: Predictable Scheduling Algorithms and Applications*, Third Edition, Springer-Verlag, 2011 (freely available online)
- Further reading:
  - Lui Sha, Tarek F. Abdelzaher, Karl-Erik Årzén, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Loheczky, and Aloysius Mok, *Real-Time Scheduling Theory: A Historical Perspective*, Real-Time Systems, vol. 28, no. 2-3, pp. 101-155, 2004
  - Giorgio C. Buttazzo, *Rate Monotonic vs. EDF: Judgment Day*, Real-Time Systems, vol. 29, no. 1, pp. 5-26, 2005

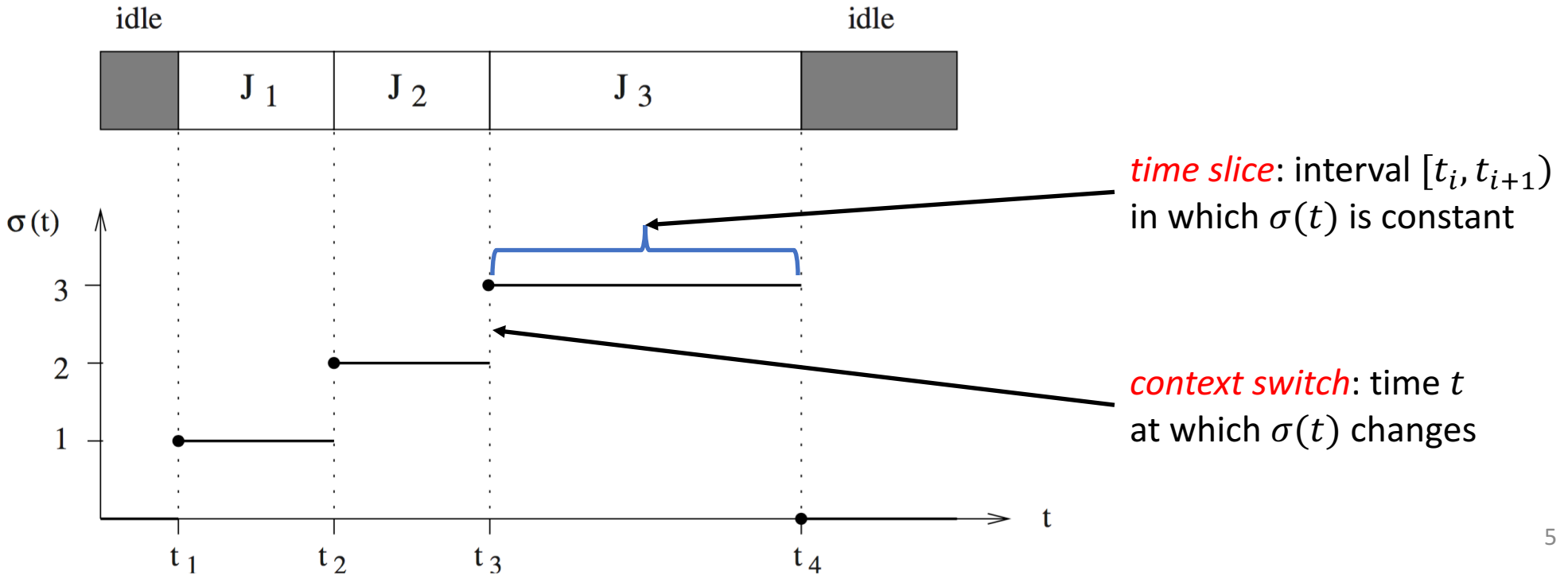
Basic Scheduling Concepts

- A *task* is a computation that is executed by the processor in a sequential fashion.
- A *scheduling algorithm* determines the order in which tasks that can overlap in time are executed on the processor.
- The operation of suspending the running task and inserting it into the ready queue is called *preemption*.



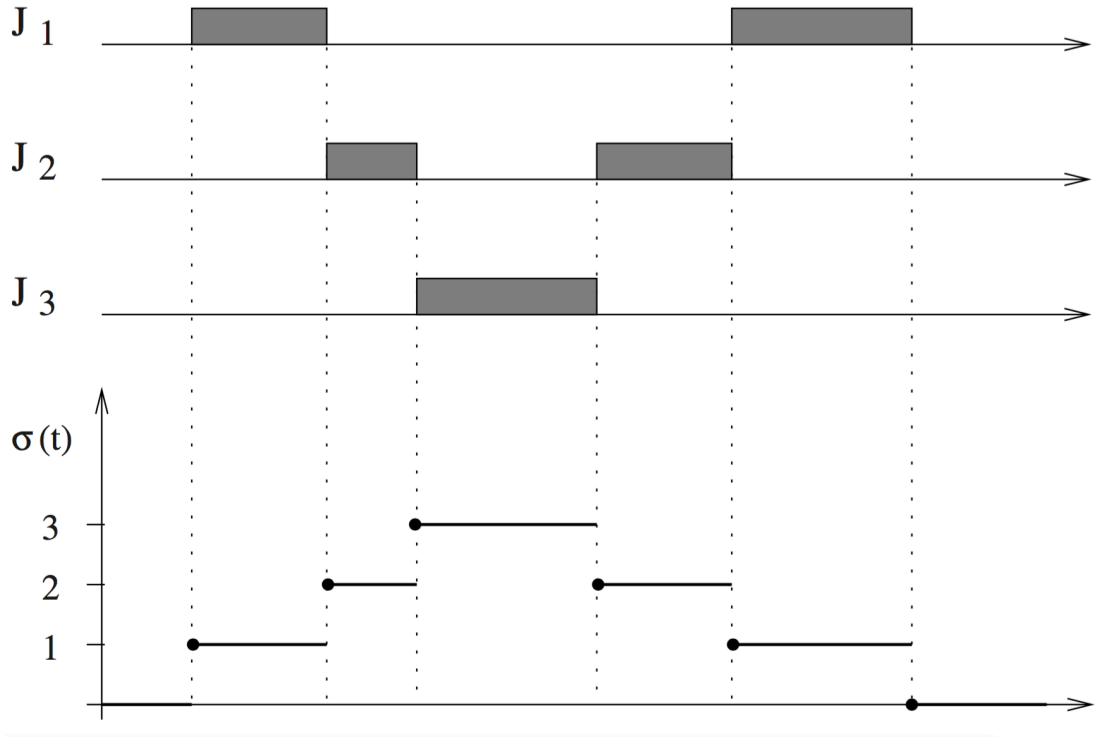
Definition of Schedule

- Given a set of tasks, $J = \{J_1, \ldots, J_n\}$, a *schedule* is an assignment of tasks to the processor, so that each task is executed until completion.
- Formally, a schedule is an integer step function $\sigma : \mathbb{R}^+ \rightarrow \mathbb{N}$, where
  - $\sigma(t) = k$ means that the processor *executes* task J_k at time t, and
  - $\sigma(t) = 0$ means that the processor is *idle* at time t.



Important Attributes

- A *preemptive* schedule is a schedule in which the running task can be arbitrarily suspended at any time to assign the processor to another task.
- A schedule is said to be *feasible* if all tasks can be completed according to a set of specified constraints.
- A set of tasks is said to be *schedulable* if there exists at least one scheduling algorithm that can produce a feasible schedule.

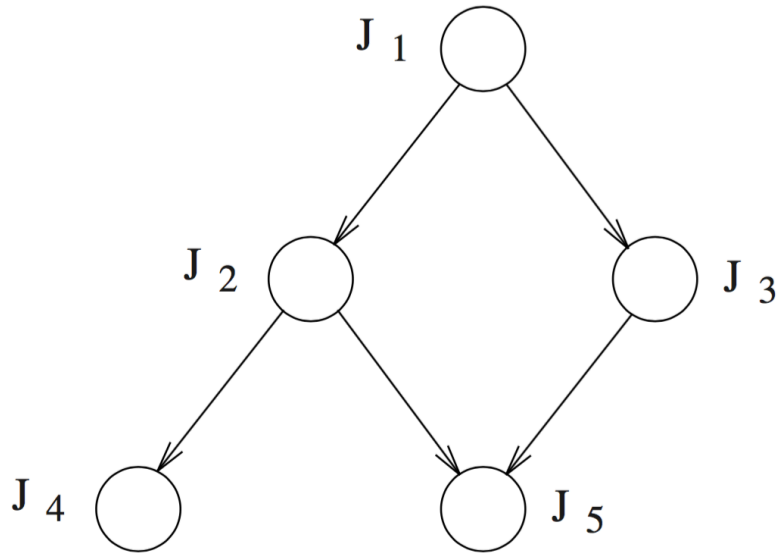


Types of Task Constraints

- *Timing constraints*: tasks need to complete before given deadlines

in this course

- *Precedence constraints*: tasks need to execute in a given order



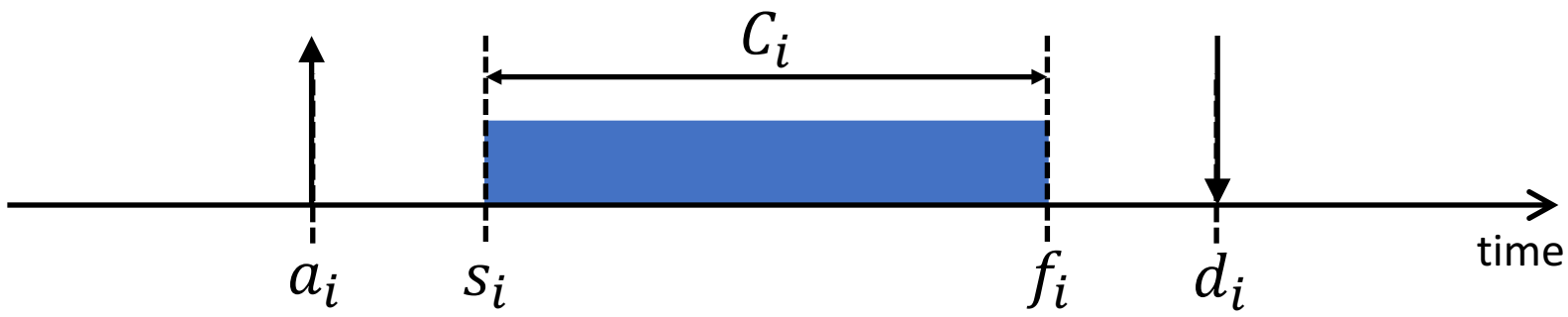
- *Resource constraints*: tasks need to execute on given resources (*e.g.*, data structure, piece of a program, memory area, peripheral device)

Tasks with Timing Constraints

- A typical timing constraint on a task is a *deadline*, representing the time before which the task should complete its execution.
- A task is called a *real-time task* if it is subject to a specified deadline.
- Depending on the consequences of a missed deadline, real-time tasks can be classified into two main categories:
  - A real-time task is said to be hard if missing its deadline may cause catastrophic consequences on the system under control (*e.g.*, sensing, actuation, and control tasks in a cyber-physical system).
  - A real-time task is said to be soft if missing its deadline does not cause any serious damage and has still some utility for the system; that is, a deadline miss is considered a performance issue, not an issue of correct behavior (*e.g.*, tasks related to user-system interactions on a smartphone).

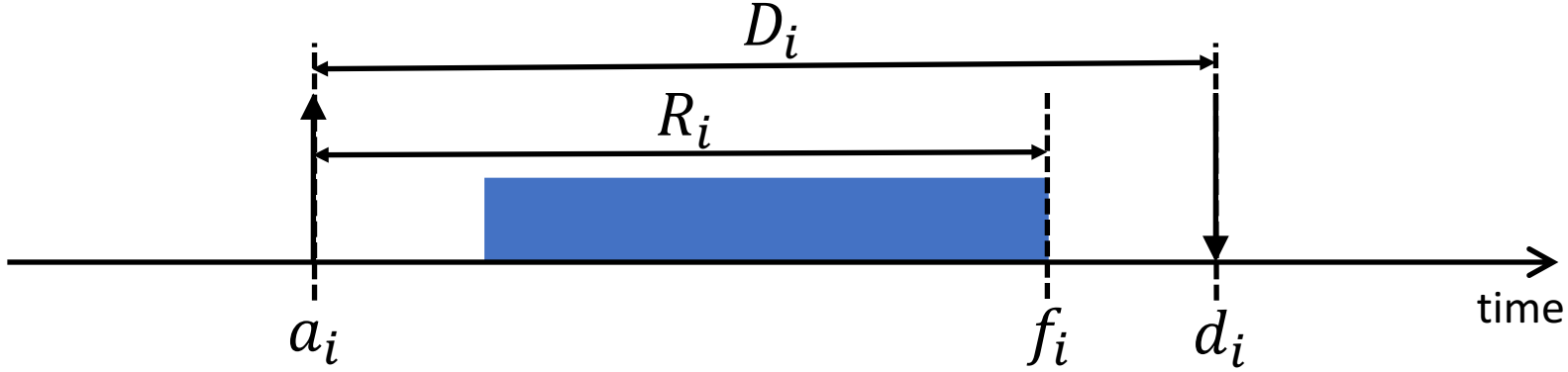
Parameters of Real-time Tasks (1)

- *Arrival time* a_i (or *release time* r_i) is the time at which a task becomes ready for execution.
- *Start time* s_i is the time at which a task starts its execution.
- *Execution time* C_i is the time needed by the processor to execute a task without interruption.
- *Finishing time* f_i is the time at which a task finishes its execution.
- *Absolute deadline* d_i is the time by which a task should be completed.

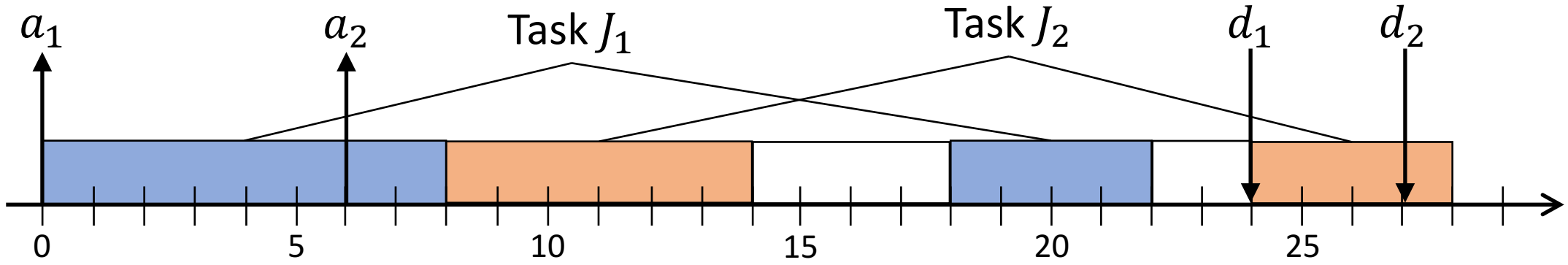


Parameters of Real-time Tasks (2)

- *Relative deadline* D_i is the difference between the absolute deadline and the arrival time of a task, that is, $D_i = d_i - a_i$.
- *Response time* R_i is the difference between the finishing time and the arrival time of a task, that is, $R_i = f_i - a_i$.
- *Lateness* $L_i = f_i - d_i$ represents the delay of a task completion with respect to its deadline (*i.e.*, $L_i \leq 0$ if task completes by its deadline).
- *Tardiness* $E_i = \max(0, L_i)$ is the time a task stays active after its deadline.
- *Laxity* $X_i = d_i - a_i - C_i$ is the maximum time a task can be delayed on its execution in order to complete within its deadline.



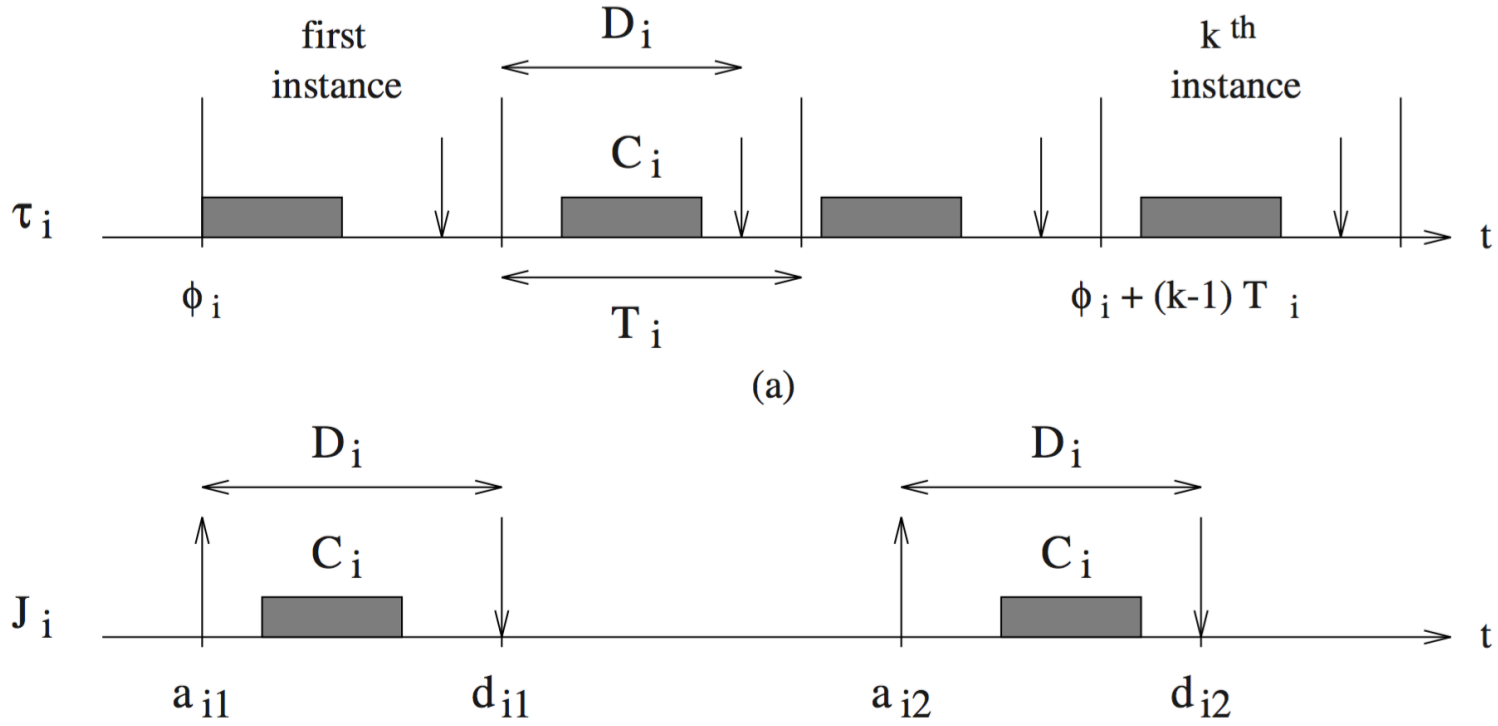
Example: Parameters of Real-time Tasks



- Execution times: $C_1 = 12, C_2 = 10$
- Start times: $s_1 = 0, s_2 = 8$
- Finishing times: $f_1 = 22, f_2 = 28$
- Lateness: $L_1 = -2, L_2 = 1$
- Tardiness: $E_1 = 0, E_2 = 1$
- Laxity: $X_1 = 12, X_2 = 11$

Periodic and Aperiodic Tasks

- A *periodic task* τ_i consists of an infinite sequence of identical activities, called instances or jobs, that are regularly activated with a constant *period* T_i. The arrival time of the first instance is called *phase* Φ_i.
- We use τ_i to denote a periodic task and J_i to denote an aperiodic task.



Classification of Scheduling Algorithms (1)

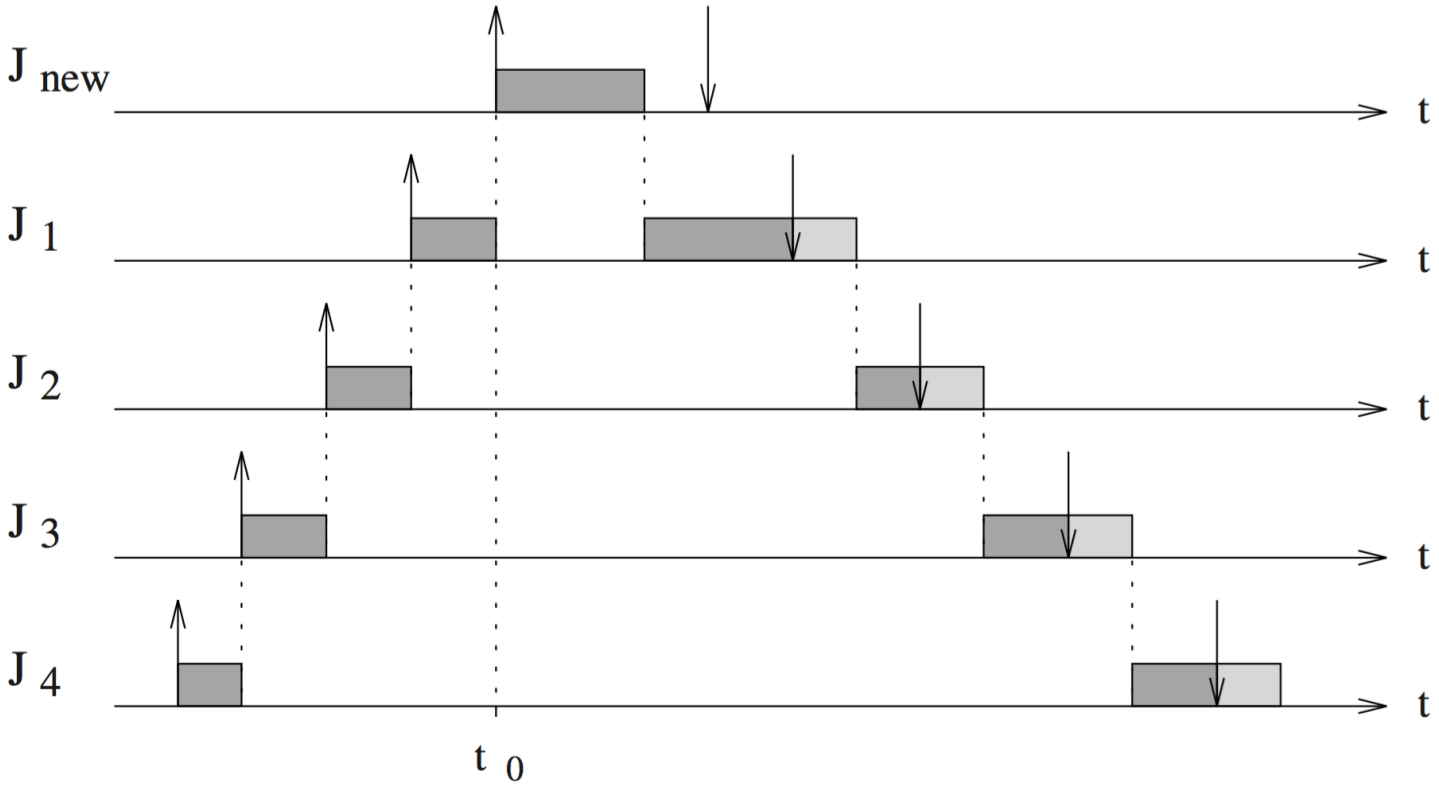
- Using a *preemptive* algorithm, the running task can be interrupted at any time to assign the processor to another active task.
- Using a *non-preemptive* algorithm, a task, once started, is executed by the processor until completion.
- *Static* algorithms are those in which scheduling decisions are based on fixed parameters that are assigned to tasks before their activation.
- *Dynamic* algorithms are those in which scheduling decisions are based on dynamic parameters that may change during system operation.

Classification of Scheduling Algorithms (2)

- An algorithm is used *offline* if it is executed on the entire task set before any task activation. The generated schedule can be stored in a table and then executed at runtime by a dispatcher.
- An algorithm is used *online* if scheduling decisions are taken at runtime every time a new task enters the system or when a running task terminates.
- An algorithm is said to be *optimal* it it minimizes a given cost function defined over the task set.
- An algorithm is said to be *heuristic* if it tends toward the optimal schedule, but does not guarantee finding it.

Schedulability Analysis

- In hard real-time applications, feasibility of the schedule should be guaranteed in advance (*i.e.*, before task execution).
  - Can be checked offline if task set is fixed and known a priori.
  - Must be checked online if tasks can be created at runtime (*acceptance test*).



Domino effect: if task J_{new} were accepted at time t_0, all other (previously schedulable) tasks would miss their deadline.

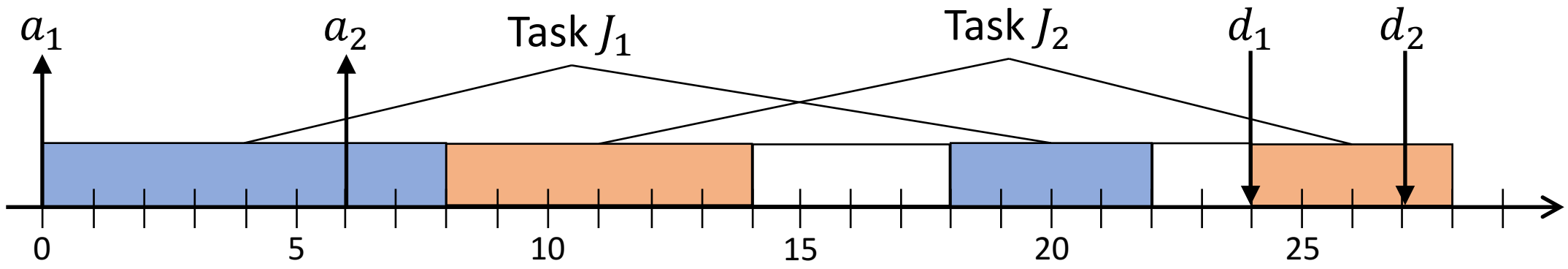
Example Cost Functions

- Average response time: $\overline{t_r} = \frac{1}{n}\sum_{i=1}^{n}(f_i - a_i)$
- Total completion time: $t_c = \max_i(f_i) - \min_i(a_i)$
- Weighted sum of finishing times: $t_w = \sum_{i=1}^{n} w_i f_i$
- Maximum lateness: $L_{max} = \max_i(f_i - d_i)$
- Maximum number of late tasks: $N_{late} = \sum_{i=1}^{n} miss(f_i)$

where $miss(f_i) = \begin{cases} 0 \text{ if } f_i \leq d_i \\ 1 \text{ otherwise} \end{cases}$

possibly useful for hard real-time systems, but … (see slide after the next)

Example: Cost Functions

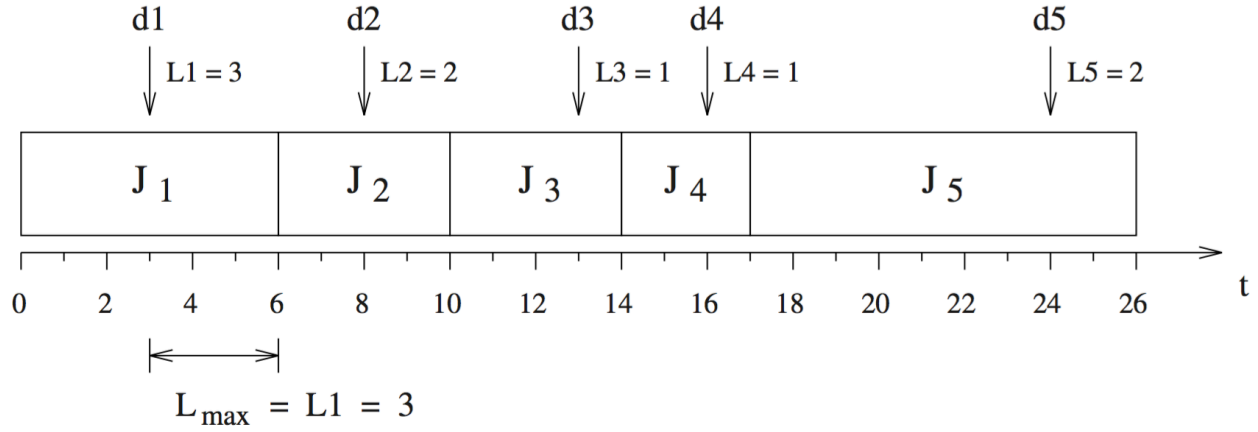


- Average response time: $\overline{t_r} = \frac{1}{n}\sum_{i=1}^{n}(f_i - a_i) = \frac{1}{2}(22 + 20) = 21$
- Total completion time: $t_c = \max_i(f_i) - \min_i(a_i) = 28 - 0 = 28$
- Weighted sum of finishing times: $t_w = \sum_{i=1}^{n} w_i f_i = 2{\times}22 + 1{\times}28 = 72$
- Maximum lateness: $L_{max} = \max_i(f_i - d_i) = \max_2(-2,1) = 1$
- Maximum number of late tasks: $N_{late} = \sum_{i=1}^{n} miss(f_i) = 0 + 1 = 1$

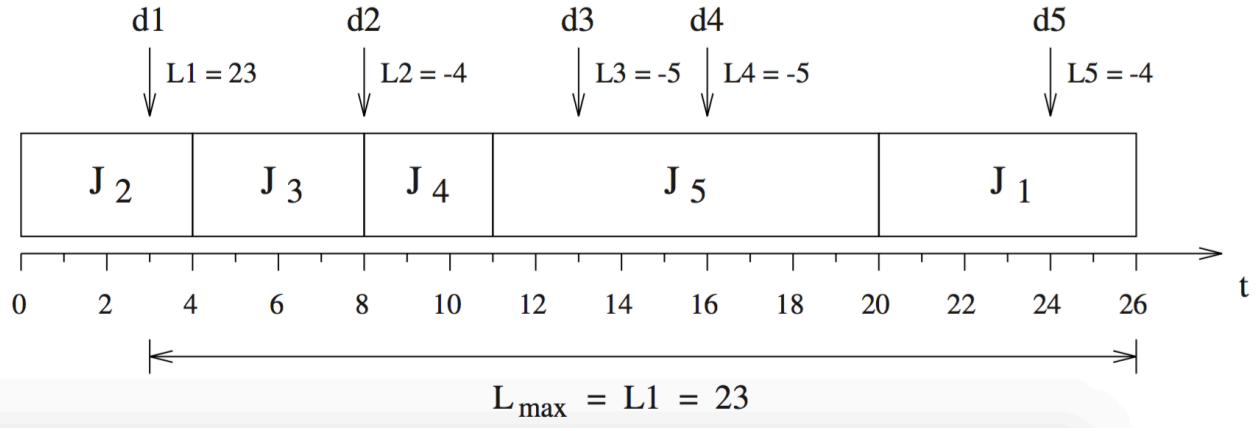
where $\quad miss(f_i) = \begin{cases} 0 \text{ if } f_i \le d_i \\ 1 \text{ otherwise} \end{cases}$

Example: Maximum Lateness vs. Deadline Misses

(a)

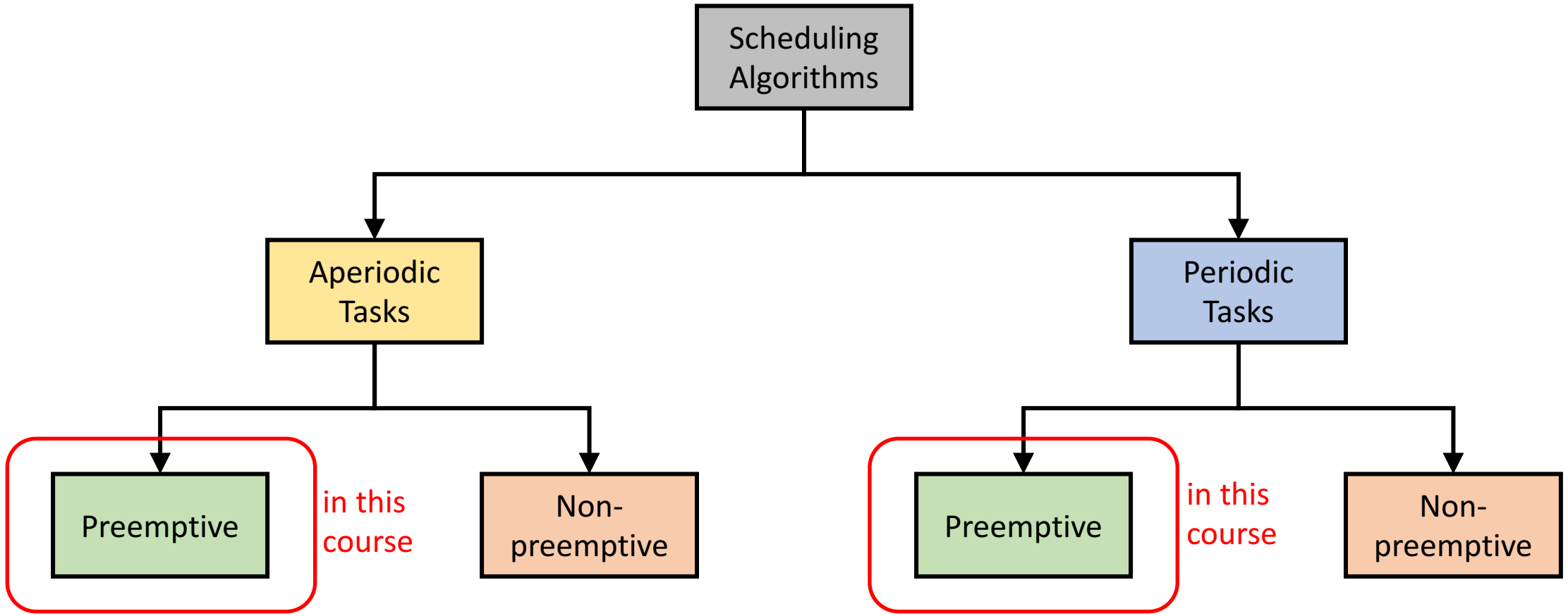


(b)



- Schedule in (a) minimizes maximum lateness, but all tasks miss deadline.
- Schedule in (b) has higher maximum lateness, but only one deadline miss.

Overview of Scheduling Algorithms

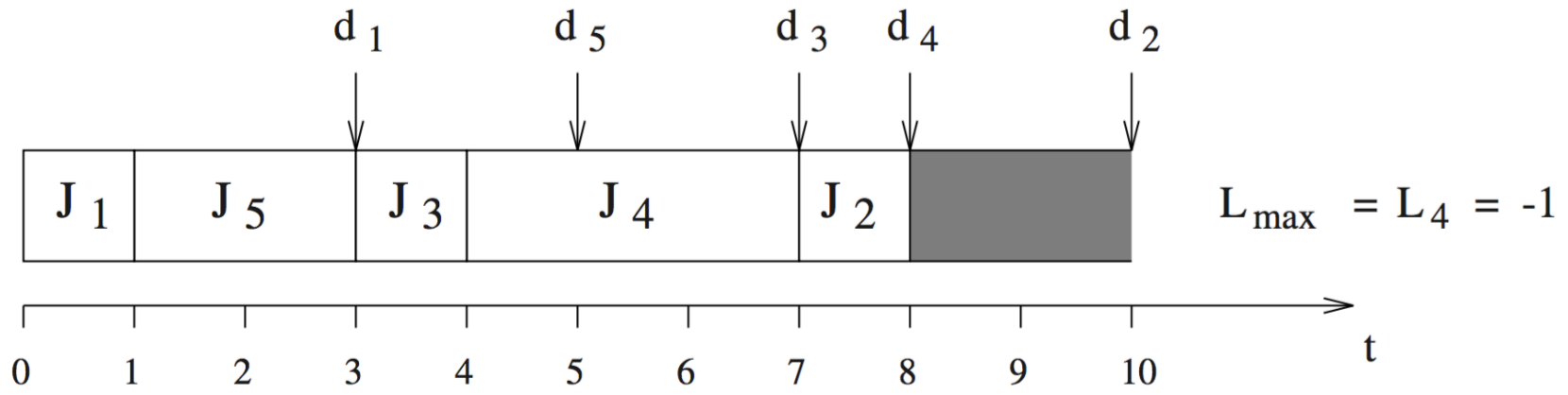


Earliest Deadline Due (EDD)

- Preemptive scheduling of aperiodic tasks J_i with *equal arrival times*
  - We assume all tasks arrive at time $t = 0$ (*i.e.*, $a_i = 0$ for all tasks J_i).
  - Thus, each task J_i is characterized by its execution time C_i and relative deadline D_i.
  - Note: Preemption is not an issue if all tasks arrive at the same time!
- *Jackson's rule*: Given a set of n real-time tasks, any algorithm that executes the tasks in order of non-decreasing deadline is optimal with respect to minimizing the maximum lateness of the task set.

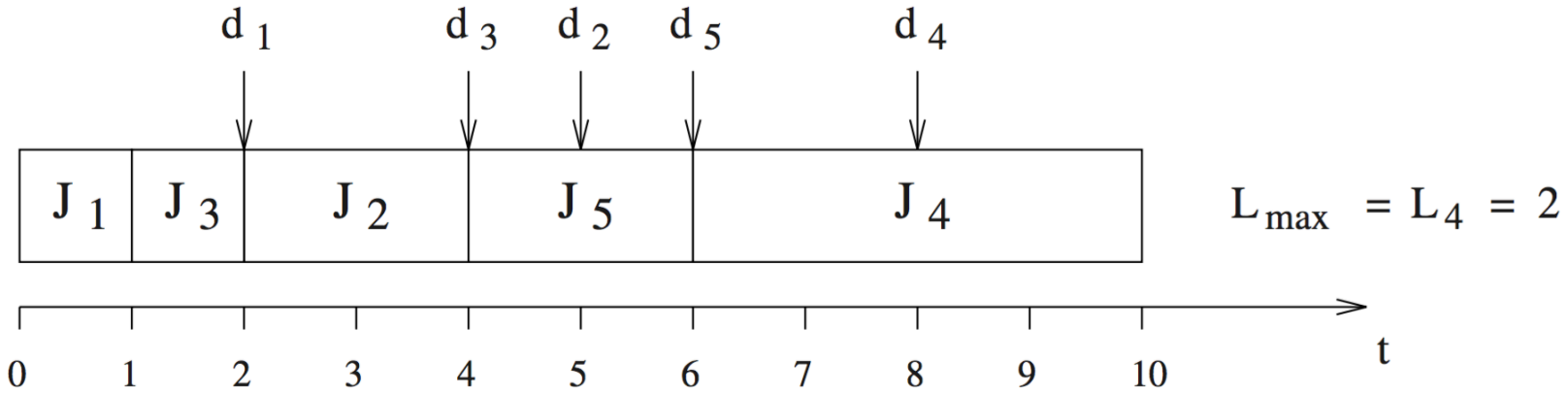
Example: Feasible Schedule Produced by EDD

	J_1	J_2	J_3	J_4	J_5
C_i	1	1	1	3	2
d_i	3	10	7	8	5



Example: Infeasible Schedule Produced by EDD

	J_1	J_2	J_3	J_4	J_5
C_i	1	2	1	4	2
d_i	2	5	4	8	6



EDD Schedulability Test

- To guarantee that scheduling a set of tasks using EDD produces a feasible schedule, we need to show that in the worst case all tasks can complete before their deadlines, that is, $f_i \leq d_i$ for all tasks J_i.
- If tasks $J_1, J_2, \ldots, J_n$ are ordered by increasing deadline, we have

$$f_i = \sum_{k=1}^{i} C_k$$

- Thus, the EDD schedulability test can be performed (offline) by verifying for each task J_i

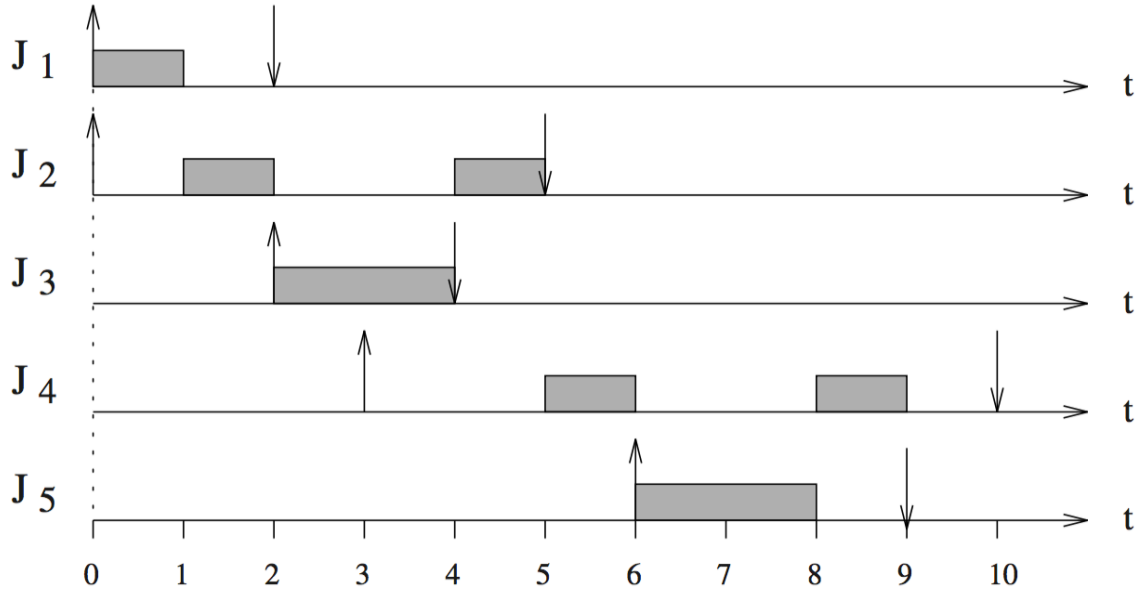
$$\sum_{k=1}^{i} C_k \leq d_i$$

Earliest Deadline First (EDF)

- Preemptive scheduling of aperiodic tasks J_i with *arbitrary arrival times*
  - Tasks $J_i(C_i, D_i)$ arrive dynamically, so preemption is an important factor.
- *Horn's rule*: Given a set of n real-time tasks with arbitrary arrival times, any algorithm that at any point in time executes the task with the earliest absolute deadline among all ready tasks is optimal with respect to minimizing the maximum lateness.

Example: Feasible Schedule Produced by EDF

	J_1	J_2	J_3	J_4	J_5
a_i	0	0	2	3	6
C_i	1	2	2	2	2
d_i	2	5	4	10	9



EDF Schedulability Test: Approach

- Similar to EDD, but the test must be done *online* whenever a new task J_{new} enters the system. Thus, assuming the current set of tasks J is schedulable, we need to check if $J' = J \cup J_{new}$ is also schedulable.
- If tasks $J_1, J_2, \ldots, J_n$ are ordered by increasing deadline, the worst-case finishing time of task J_i at time t is given by

$$f_i = t + \sum_{k=1}^{i} c_k(t)$$

- Here, $c_k(t)$ is the *remaining worst-case execution* time of task J_k. It is initially equal to C_k, but may have a lower value at time t when J_{new} arrives since task J_k (and others) may have been partially executed.
- Thus, the EDF schedulability test performed online at time t amounts to verifying for each task $J_i \in J'$

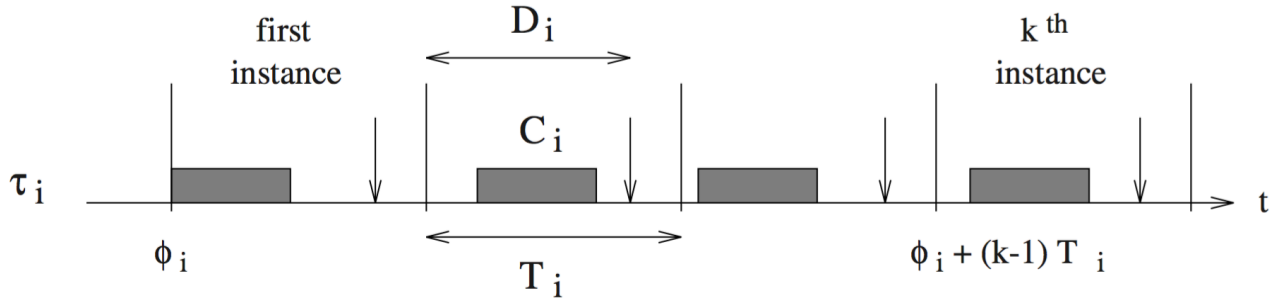
$$t + \sum_{k=1}^{i} c_k(t) \leq d_i$$

EDF Schedulability Test: Algorithm

```
edf_schedulability_test(J, J_new){
    J' = J ∪ {J_new}; /* ordered by incr. deadline */
    f_0 = get_current_time();
    for each J_i ∈ J' {
        f_i = f_i-1 + c_i(t);
        if (f_i > d_i){
            return NOT_SCHEDULABLE;
        }
    }
    return SCHEDULABLE;
}
```

Rate-monotonic (RM) Scheduling

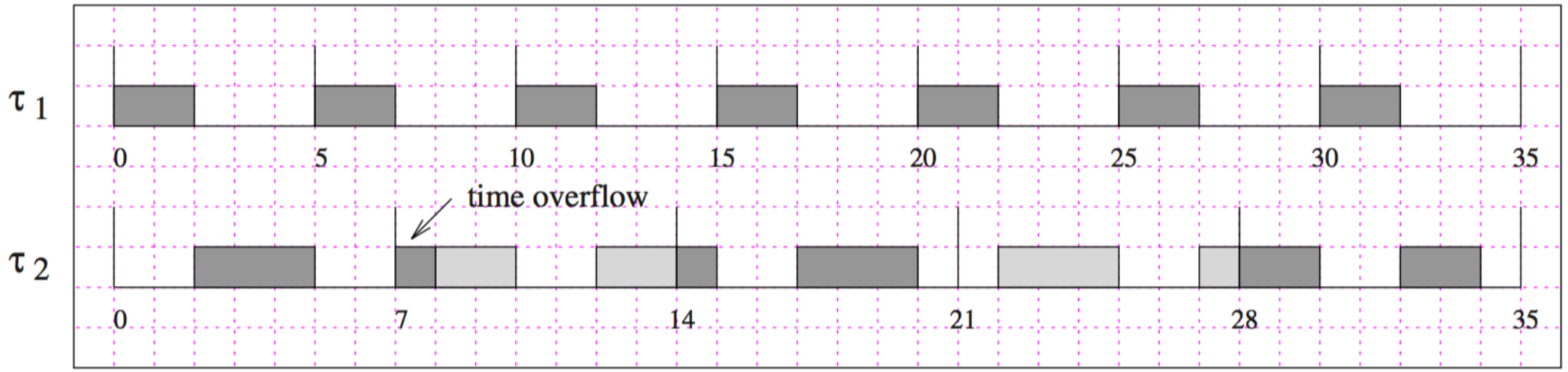
- Preemptive scheduling of periodic tasks τ_i with
  - phase Φ_i, period T_i, relative deadline D_i, and (worst-case) execution time C_i.



  - Thus, release times are given by $r_{i,k} = \Phi_i + (k-1)T_i$ and absolute deadlines are given by $d_{i,k} = r_{i,k} + D_i$. If $D_i = T_i$ for all tasks, we have $d_{i,k} = \Phi_i + kT_i$.
- *Algorithm*: Given a set of n periodic real-time tasks with $D_i = T_i$, assign a *fixed* priority to each task, such that tasks with higher request rates (*i.e.*, with shorter periods) have higher priorities. The currently executing task is preempted by a task with higher (fixed) priority.

Example: Infeasible Schedule Produced by RM

- Two tasks τ_1 and τ_2 with
  - Phases $\Phi_1 = \Phi_2 = 0$
  - Periods $T_1 = 5$ and $T_2 = 7$, thus task τ_1 has higher priority than task τ_2
  - Worst-case execution times $C_1 = 2$ and $C_2 = 4$



RM Schedulability Test

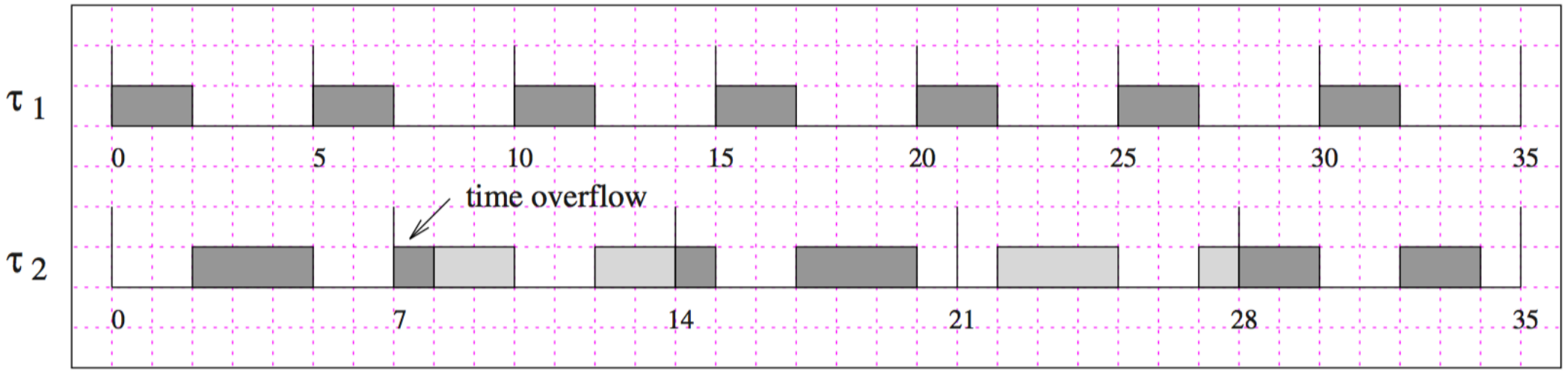
- A set of n periodic real-time tasks is schedulable using RM if

$$\sum_{i=1}^{n} C_i/T_i \leq n\left(2^{1/n} - 1\right)$$

- This condition is *sufficient but no necessary*. That is, if the above condition holds for a given task set, then this task set is definitely schedulable using RM, but if the above condition does not hold, then this task set may or may not be schedulable using RM.
- The term $\sum_{i=1}^{n} C_i/T_i$ is called the *processor utilization* U of a set of n periodic real-time tasks. It denotes the fraction of time the processor spends executing the task set (*i.e.*, the computational load).

RM Example Revisited: Processor Utilization

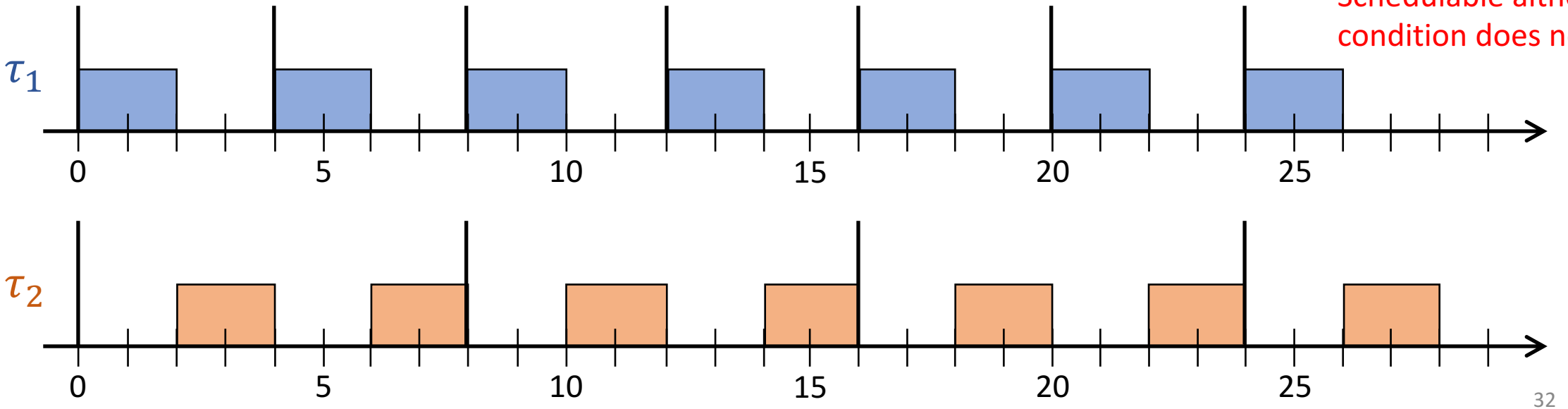
- Two tasks τ_1 and τ_2 with
  - Phases $\Phi_1 = \Phi_2 = 0$
  - Periods $T_1 = 5$ and $T_2 = 7$, thus task τ_1 has higher priority than task τ_2
  - Worst-case execution times $C_1 = 2$ and $C_2 = 4$
  - *Processor utilization* $U = \sum_{i=1}^{n} C_i/T_i = \frac{2}{5} + \frac{4}{7} \approx 0.97 > 2\left(2^{\frac{1}{2}} - 1\right) \approx 0.83$



RM Example Revisited: Sufficiency

- Two tasks τ_1 and τ_2 with
  - Phases $\Phi_1 = \Phi_2 = 0$
  - Periods $T_1 = 4$ and $T_2 = 8$, thus task τ_1 has higher priority than task τ_2
  - Worst-case execution times $C_1 = 2$ and $C_2 = 4$
  - Processor utilization $U = \sum_{i=1}^{n} C_i/T_i = \frac{2}{4} + \frac{4}{8} = 1 > 2\left(2^{\frac{1}{2}} - 1\right) \approx 0.83$

Schedulable although condition does not hold



Earliest Deadline First (EDF)

- Preemptive scheduling of periodic tasks τ_i
- *Algorithm*: A *dynamic* priority is assigned to each task, such that tasks with earlier deadlines have higher priorities. The currently executing task is preempted whenever a task with earlier deadline becomes active.
- Using EDF, priorities are assigned *dynamically*, because the absolute deadline $d_{i,k}$ of a periodic task τ_i depends on the current kth instance

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

- Using RM, the priorities are *fixed*, because the periods T_i are constant.

EDF Schedulability Test

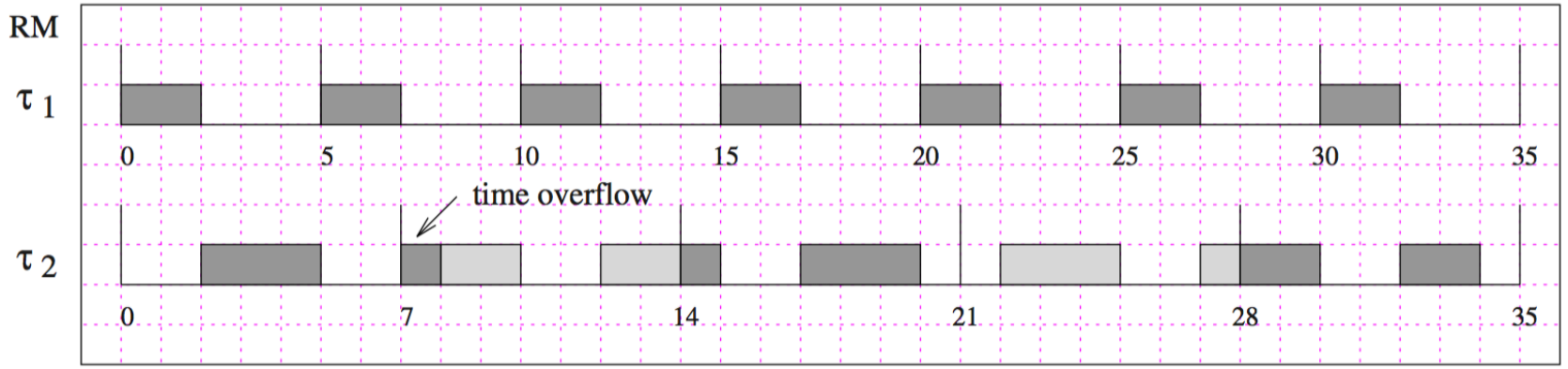
- A set of n periodic real-time tasks, where $D_i = T_i$ for all tasks τ_i, is schedulable using EDF if and only if

$$\sum_{i=1}^{n} C_i/T_i \leq 1$$

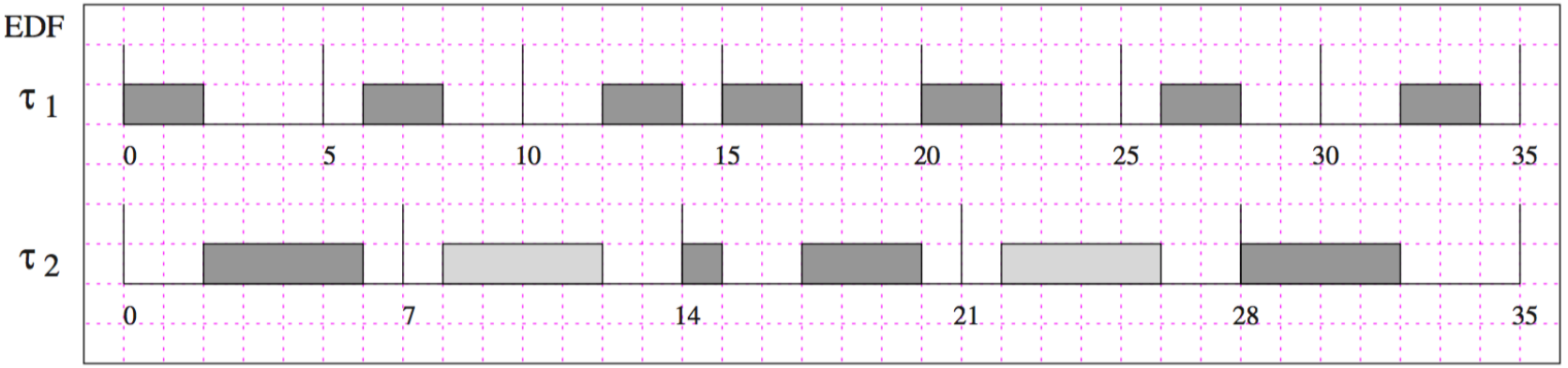
- This condition is both *necessary and sufficient*.

Example: EDF vs. RM

- Two tasks τ_1 and τ_2 with
  - Phases $\Phi_1 = \Phi_2 = 0$
  - Periods $T_1 = 5$ and $T_2 = 7$
  - Worst-case execution times $C_1 = 2$ and $C_2 = 4$



$U \approx 0.97 > 0.83$



$U \approx 0.97 < 1$

No deadline miss, fewer preemptions due to dynamic priority assignment

A Note on Optimality

- *RM is optimal* among all fixed-priority assignments in the sense that no other fixed-priority algorithm can schedule a set of periodic real-time tasks that cannot be scheduled by RM.
- *EDF is optimal* in the sense that no other algorithm can schedule a set of periodic real-time tasks that cannot be scheduled by EDF.

Summary of Today's Lecture

- Correct behavior of a real-time system depends not only on the output of a computation but also on the time at which the output is produced.
- Real-time scheduling algorithms aim to meet application-specific timing (and possibly other types of) constraints on tasks.
- In this course: different algorithms and corresponding schedulability tests for scheduling real-time tasks on a single processor
  - Preemptive + aperiodic tasks: EDD, EDF
  - Preemptive + periodic tasks: RM (fixed priorities), EDF (dynamic priorities)
- Despite their long history of 40+ years, real-time scheduling concepts are highly relevant today in CPS, IoT, datacenters, cloud computing, etc.