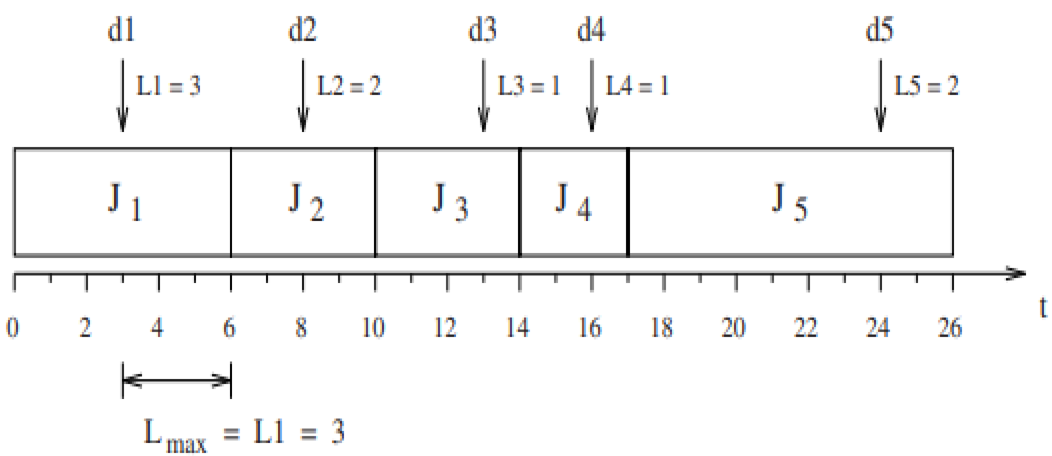
Royaume du Maroc
Ministère de l'Education Nationale, de la Formation Professionnelle
de l'Enseignement Supérieur et de la Recherche Scientifique

Université Chouaib Doukkali,
Ecole Nationale des Sciences Appliquées d'El Jadida (ENSA)
2020-2021

An Introduction To Real-Time Systems and Computing



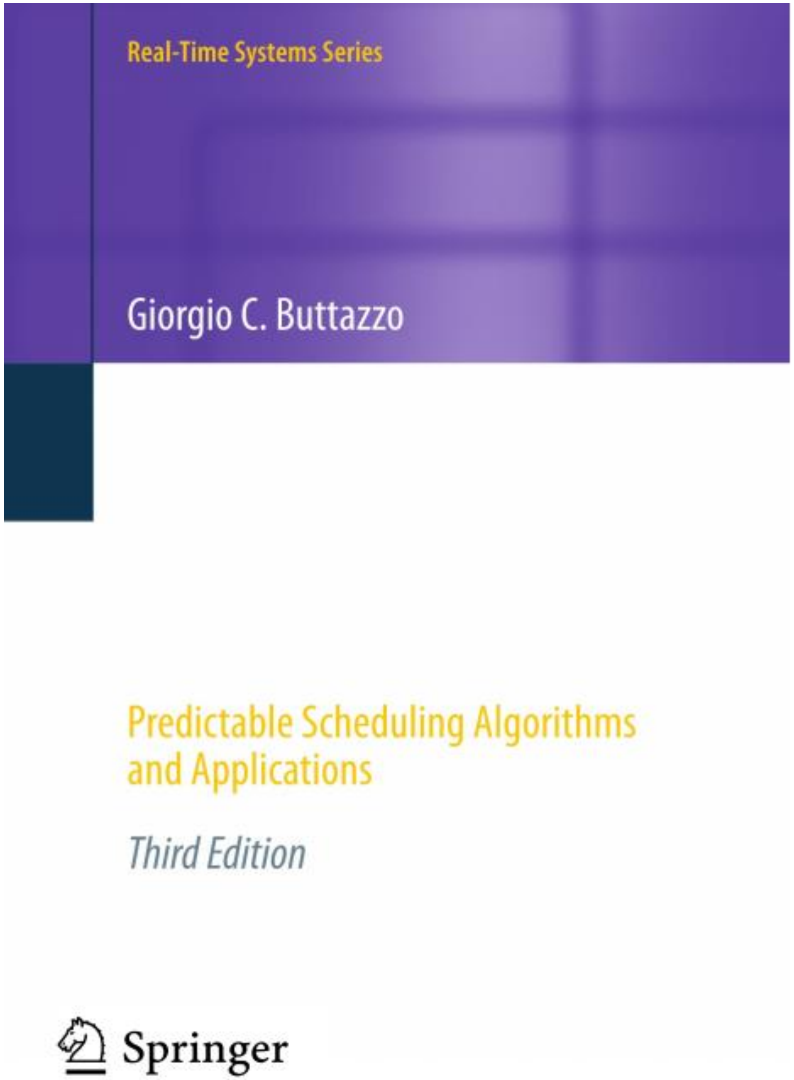
Semester 4

2ITE/Grade 2/Class 5

Dr.-Ing. Fouad KHARROUBI

Scheduling Algorithms

The content of these slides is based on this book:



Giorgio C. Buttazzo

Hard Real-Time Computing Systems

Predictable Scheduling Algorithms
and Applications

Third Edition

Springer

BASIC CONCEPTS

Introduction

Types of task constraints

Definition of scheduling problems

Scheduling anomalies

PERIODIC TASK SCHEDULING

Introduction

Rate Monotonic scheduling

Earliest Deadline First

Deadline Monotonic

EDF with constrained deadlines

Comparison between RM and EDF

APERIODIC TASK SCHEDULING

Introduction

Jackson's algorithm

Horn's algorithm

Non-preemptive scheduling

Scheduling with precedence constraints

Summary

Basic Concepts: Introduction

Basic Concepts : Introduction

Let's define some basic concepts that will be used throughout this presentation:

•**The process:** A process is a computation that is executed by the CPU in a sequential fashion. In this text, the term process is used as synonym of task and thread.

• **scheduling policy**: When a single processor has to execute a set of concurrent tasks – that is, tasks that can **overlap** in time – the **CPU** has to be assigned to the various tasks according to a **predefined criterion**, called a scheduling policy.

• **scheduling algorithm:** The set of rules that, at any time, determines the order in which tasks are executed is called a scheduling algorithm.

• **dispatching** : The specific operation of **allocating the CPU to a task selected by the scheduling algorithm** is referred as dispatching.

Basic Concepts : Introduction

- **execution**: a task that could potentially execute on the CPU can be either **in execution** (if it has been selected by the **scheduling algorithm**) or **waiting for the CPU** (if another task is **executing**).

- **Active task:** A task that can potentially execute on the processor, **independently** on its **actual availability**, is called an **active task**.

- **Ready task**: A task **waiting for the processor is** called a **ready task**.

- **Running task**: the task **in execution** is called a **running task**.

- **Ready queue**: **All ready tasks waiting for the processor are kept in a queue**, called ready queue. Operating systems that handle different types of tasks may have more than one ready queue.

Basic Concepts : Introduction

• **preemption**: In many operating systems that allow **dynamic task activation**, the running task can be **interrupted at any point**, so that a **more important task** that arrives in the system can **immediately gain the processor** and does not need to wait in the ready queue. **In this case, the running task is interrupted and inserted in the ready queue**, while the CPU is assigned to the most important ready task that just arrived. The operation of suspending the running task and inserting it into the ready queue is called **preemption**

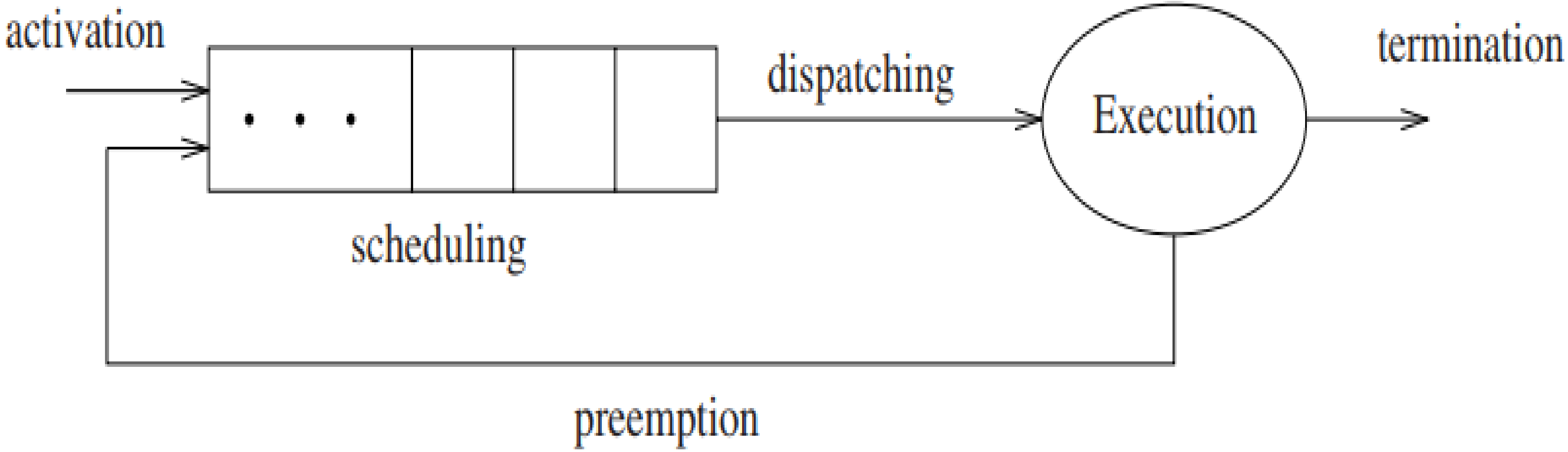


Figure . Queue of ready tasks waiting for execution.

Basic Concepts : Introduction

- **preemption**: In dynamic realtime systems, preemption is important **for three reasons:**

•Tasks performing exception handling may need to preempt existing tasks so that responses to exceptions may be issued in a timely fashion.

•When tasks have different levels of criticality (expressing task importance), preemption permits executing the most critical tasks, as soon as they arrive.

•Preemptive scheduling typically allows higher efficiency, in the sense that it allows executing a real-time task sets with higher processor utilization.

On the other hand, preemption **destroys** program locality and introduces a runtime overhead that inflates the execution time of tasks. As a consequence, limiting preemptions in real-time schedules can have beneficial effects in terms of schedulability.

Basic Concepts : Introduction

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. More formally, a schedule can be defined as a function $\sigma : \mathbf{R}^+ \rightarrow \mathbf{N}$ such that $\forall t \in \mathbf{R}^+$, $\exists t_1, t_2$ such that $t \in [t_1, t_2)$ and $\forall t' \in [t_1, t_2)$ $\sigma(t) = \sigma(t')$. In other words, $\sigma(t)$ is an integer step function and $\sigma(t) = k$, with $k > 0$, means that task J_k is executing at time t, while $\sigma(t) = 0$ means that the CPU is idle. Figure 2.2 shows an example of schedule obtained by executing three tasks: J_1, J_2, J_3.

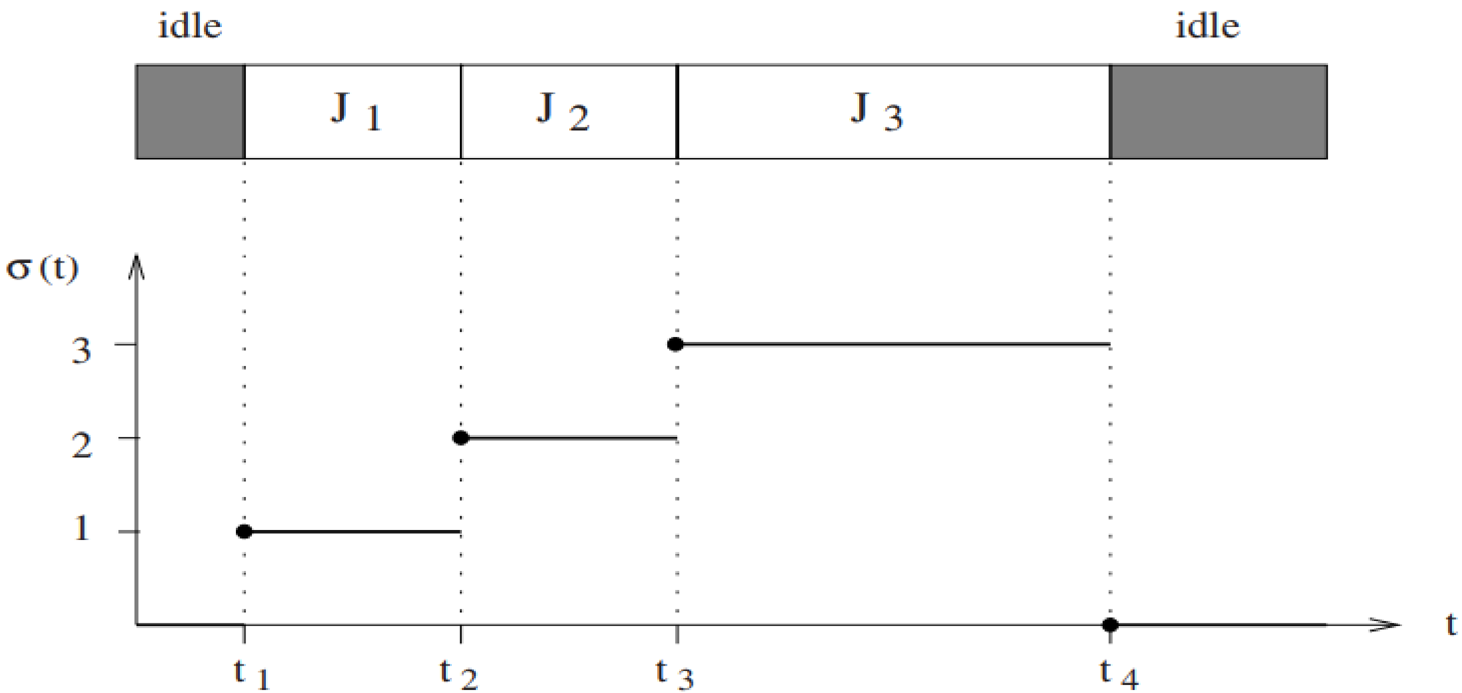


Figure . Schedule obtained by executing three tasks J_1, J_2, and J_3.

Basic Concepts: Types of Task Constraints

Basic Concepts : Types of Task Constraints

Typical **constraints** that can be specified on real-time tasks are of three classes:

- **Timing constraints**
- **Precedence relations**
- **Resource Constraints**

Basic Concepts : Types of Task Constraints

- ## Timing constraints

Real-time systems are characterized by computational activities with stringent timing constraints that must be met in order to achieve the desired behavior. A typical timing constraint on a task is the *deadline*, which represents the time before which a process should complete its execution without causing any damage to the system. If a deadline is specified with respect to the task arrival time, it is called a *relative deadline*, whereas if it is specified with respect to time zero, it is called an *absolute deadline*. Depending on the consequences of a missed deadline, real-time tasks are usually distinguished in three categories:

- **Hard**: A real-time task is said to be *hard* if missing its deadline may cause catastrophic consequences on the system under control.
- **Firm**: A real-time task is said to be *firm* if missing its deadline does not cause any damage to the system, but the output has no value.
- **Soft**: A real-time task is said to be *soft* if missing its deadline has still some utility for the system, although causing a performance degradation.

Basic Concepts : Types of Task Constraints

- **Timing constraints**

In general, a real-time task τ_i can be characterized by the following parameters:

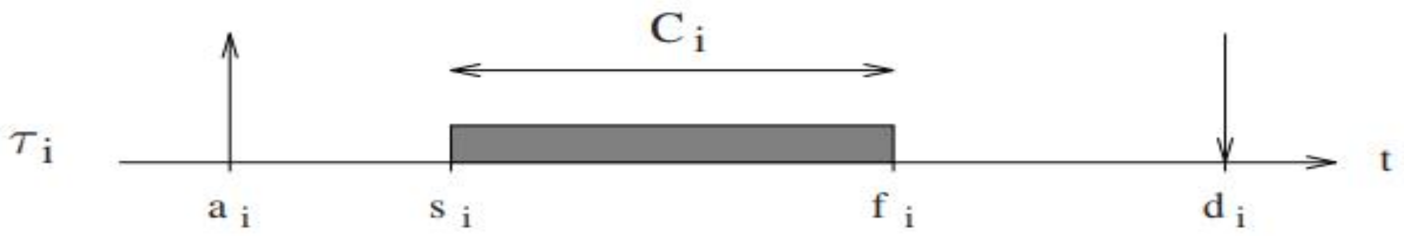


Figure. Typical parameters of a real-time task.

- **Arrival time** a_i is the time at which a task becomes ready for execution; it is also referred as *request time* or *release time* and indicated by r_i;
- **Computation time** C_i is the time necessary to the processor for executing the task without interruption;
- **Absolute Deadline** d_i is the time before which a task should be completed to avoid damage to the system;
- **Relative Deadline** D_i is the difference between the absolute deadline and the request time: $D_i = d_i - r_i$;
- **Start time** s_i is the time at which a task starts its execution;
- **Finishing time** f_i is the time at which a task finishes its execution;
- **Response time** R_i is the difference between the finishing time and the request time: $R_i = f_i - r_i$;

Basic Concepts : Types of Task Constraints

- **Timing constraints**

In general, a real-time task τ_i can be characterized by the following parameters:

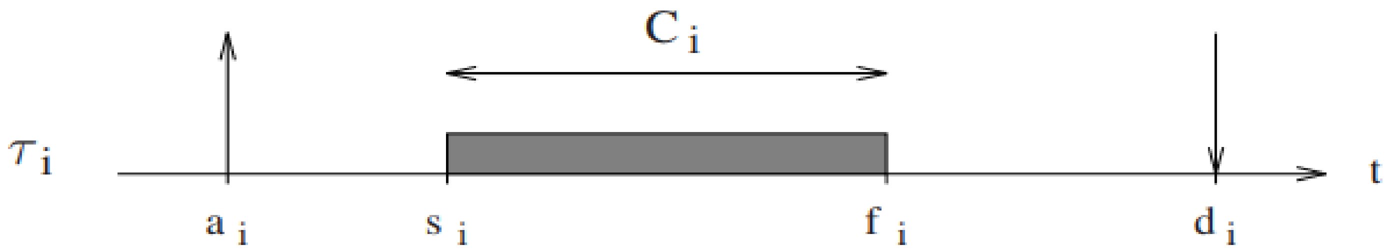


Figure . Typical parameters of a real-time task.

- **Criticality** is a parameter related to the consequences of missing the deadline (typically, it can be hard, firm, or soft);
- **Value** v_i represents the relative importance of the task with respect to the other tasks in the system;
- **Lateness** L_i: $L_i = f_i - d_i$ represents the delay of a task completion with respect to its deadline; note that if a task completes before the deadline, its lateness is negative;
- **Tardiness** or *Exceeding time* E_i: $E_i = max(0, L_i)$ is the time a task stays active after its deadline;
- **Laxity** or *Slack time* X_i: $X_i = d_i - a_i - C_i$ is the maximum time a task can be delayed on its activation to complete within its deadline.

Basic Concepts : Types of Task Constraints

- **Timing constraints**

Another timing characteristic that can be specified on a real-time task concerns the regularity of its activation. In particular, tasks can be defined as *periodic* or *aperiodic*. Periodic tasks consist of an infinite sequence of identical activities, called *instances* or *jobs*, that are regularly activated at a constant rate. For the sake of clarity, from now on, a periodic task will be denoted by τ_i, whereas an aperiodic job by J_i. The generic k^{th} job of a periodic task τ_i will be denoted by $\tau_{i,k}$.

The activation time of the first periodic instance ($\tau_{i,1}$) is called *phase*. If ϕ_i is the phase of task τ_i, the activation time of the k^{th} instance is given by $\phi_i + (k-1)T_i$, where T_i is the activation *period* of the task. In many practical cases, a periodic process can be completely characterized by its phase ϕ_i, its computation time C_i, its period T_i, and its relative deadline D_i.

Aperiodic tasks also consist of an infinite sequence of identical jobs (or instances); however, their activations are not regularly interleaved. An aperiodic task where consecutive jobs are separated by a minimum inter-arrival time is called a *sporadic task*.

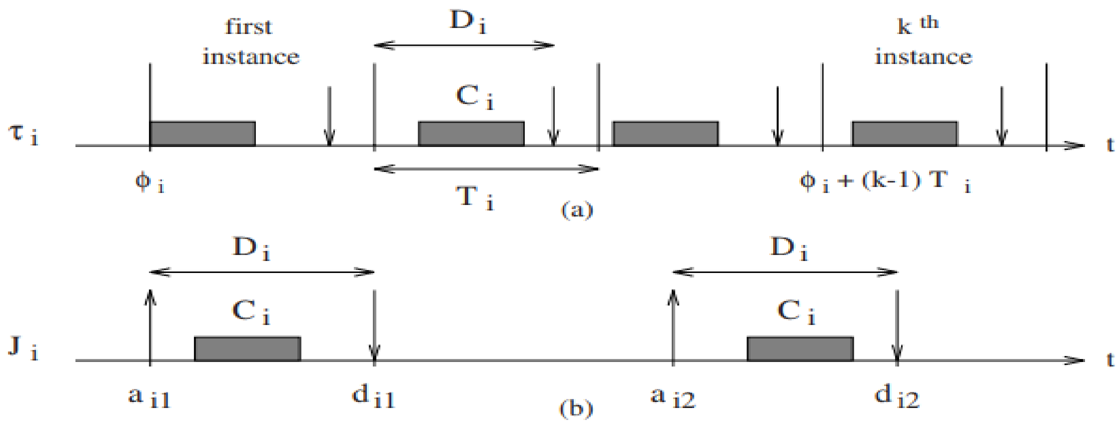
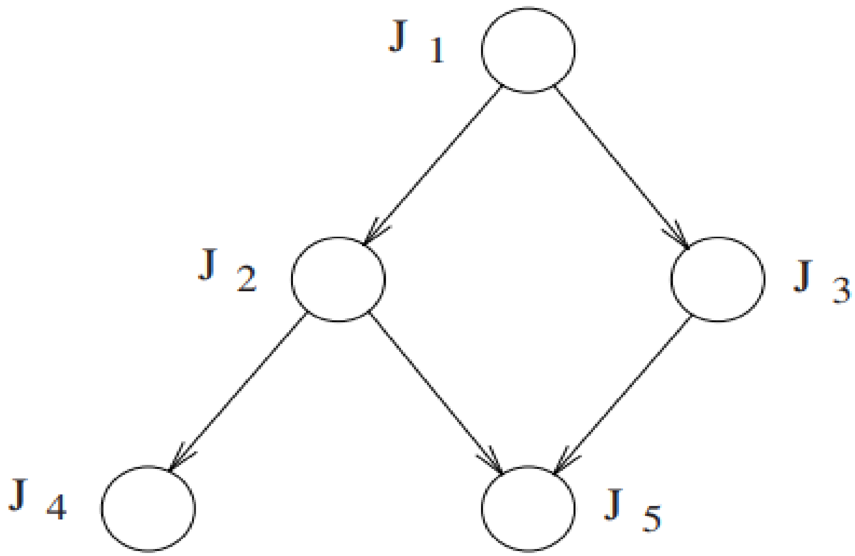


Figure. Sequence of instances for a periodic task (a) and an aperiodic job (b).

Basic Concepts : Types of Task Constraints

- **Precedence constraints**

In certain applications, computational activities cannot be executed in arbitrary order but have to respect some precedence relations defined at the design stage. Such precedence relations are usually described through a directed acyclic graph G, where tasks are represented by nodes and precedence relations by arrows. A precedence graph G induces a partial order on the task set.



$J_1 \prec J_2$

$J_1 \rightarrow J_2$

$J_1 \prec J_4$

$J_1 \not\rightarrow J_4$

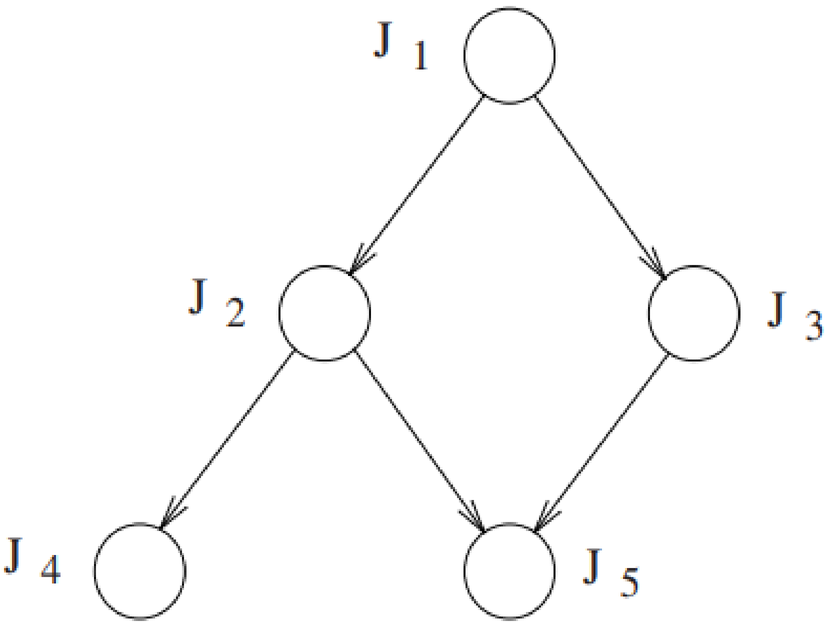
Figure . Precedence relations among five tasks.

- The notation $J_a \prec J_b$ specifies that task J_a is a *predecessor* of task J_b, meaning that G contains a directed path from node J_a to node J_b.

- The notation $J_a \rightarrow J_b$ specifies that task J_a is an *immediate predecessor* of J_b, meaning that G contains an arc directed from node J_a to node J_b.

Basic Concepts : Types of Task Constraints

- **Precedence constraints**



$$J_1 \prec J_2$$
$$J_1 \rightarrow J_2$$
$$J_1 \prec J_4$$
$$J_1 \not\rightarrow J_4$$

Figure . Precedence relations among five tasks.

the Figure illustrates a directed acyclic graph that describes the precedence constraints among five tasks. From the graph structure we observe that task J_1 is the only one that can start executing since it does not have predecessors. Tasks with no predecessors are called *beginning tasks*. As J_1 is completed, either J_2 or J_3 can start. Task J_4 can start only when J_2 is completed, whereas J_5 must wait for the completion of J_2 and J_3. Tasks with no successors, as J_4 and J_5, are called *ending tasks*.

Basic Concepts : Types of Task Constraints

- **Precedence constraints**

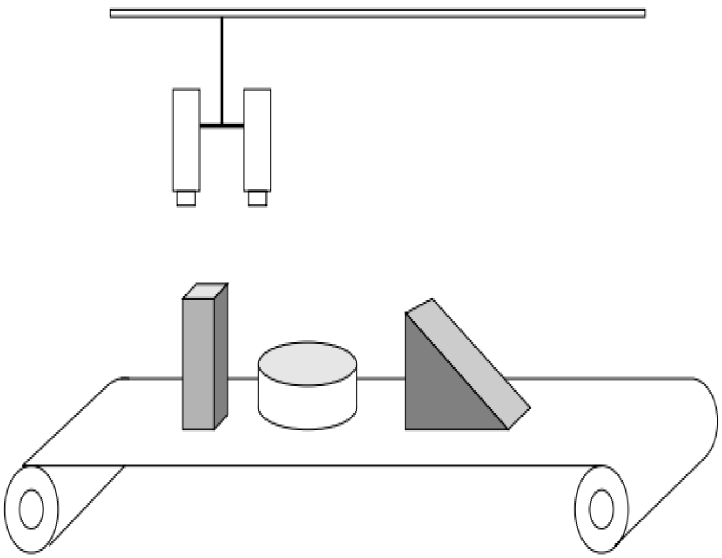


Figure . Industrial application that requires a visual recognition of objects on a conveyor belt.

Can you draw the Precedence task graph associated with this industrial application?

- Two tasks (one for each camera) dedicated to image acquisition, whose objective is to transfer the image from the camera to the processor memory (they are identified by *acq1* and *acq2*);
- Two tasks (one for each camera) dedicated to low-level image processing (typical operations performed at this level include digital filtering for noise reduction and edge detection; we identify these tasks as *edge1* and *edge2*);
- A task for extracting two-dimensional features from the object contours (it is referred as *shape*);
- A task for computing the pixel disparities from the two images (it is referred as *disp*);
- A task for determining the object height from the results achieved by the *disp* task (it is referred as *H*);
- A task performing the final recognition (this task integrates the geometrical features of the object contour with the height information and tries to match these data with those stored in the data base; it is referred as *rec*).

Basic Concepts : Types of Task Constraints

- **Precedence constraints**

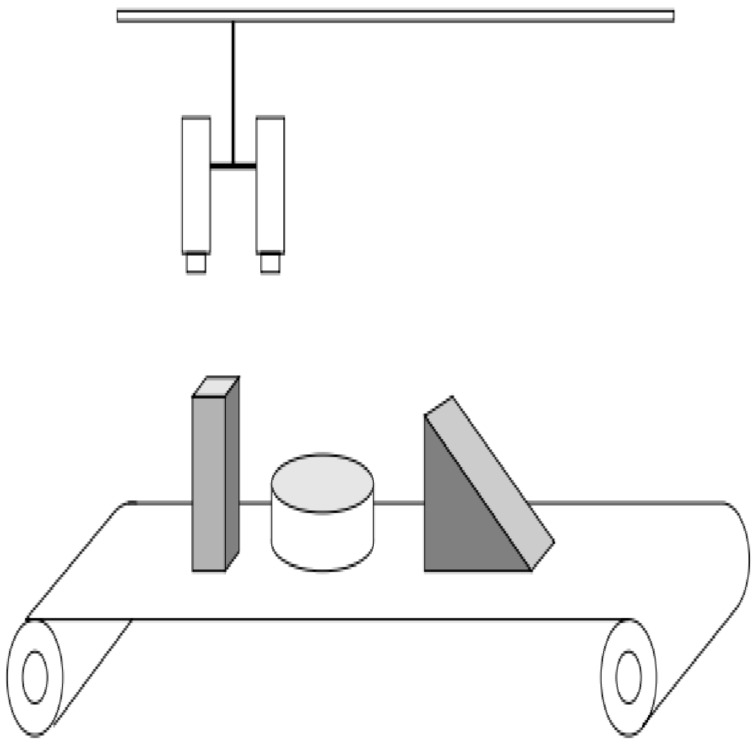


Figure . Industrial application that requires a visual recognition of objects on a conveyor belt.

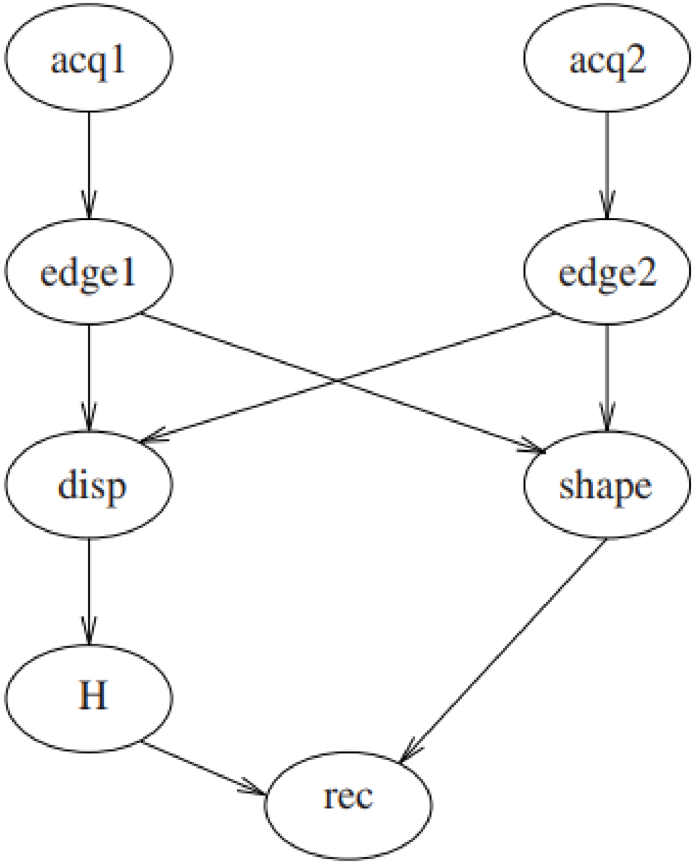


Figure . Precedence task graph associated with the industrial application

Basic Concepts : Types of Task Constraints

- **Precedence constraints**

From the logic relations existing among the computations, it is easy to see that tasks *acq1* and *acq2* can be executed in parallel before any other activity. Tasks *edge1* and *edge2* can also be executed in parallel, but each task cannot start before the associated acquisition task completes. Task *shape* is based on the object contour extracted by the low-level image processing; therefore, it must wait for the termination of both *edge1* and *edge2*. The same is true for task *disp*, which however can be executed in parallel with task *shape*. Then, task *H* can only start as *disp* completes and, finally, task *rec* must wait the completion of *H* and *shape*.

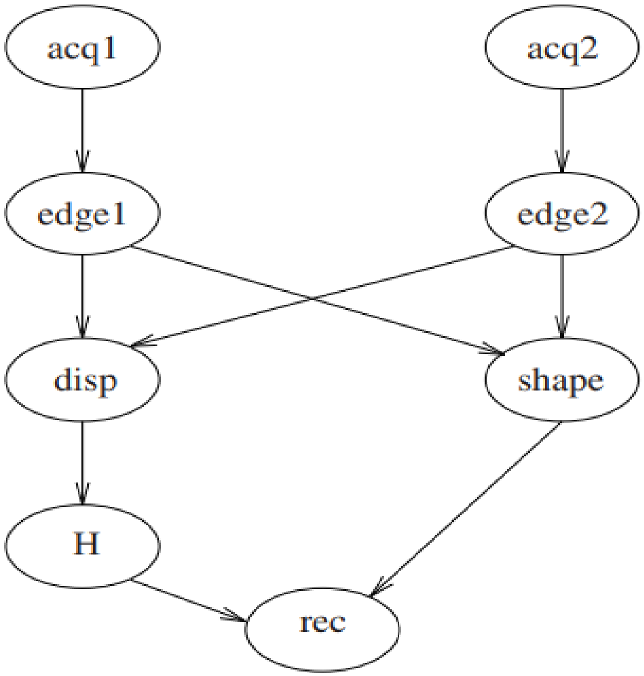


Figure . Precedence task graph associated with the industrial application

Basic Concepts : Types of Task Constraints

- **Resource constraints**

From a process point of view, a *resource* is any software structure that can be used by the process to advance its execution. Typically, a resource can be a data structure, a set of variables, a main memory area, a file, a piece of program, or a set of registers of a peripheral device. A resource dedicated to a particular process is said to be *private*, whereas a resource that can be used by more tasks is called a *shared resource*.

To maintain data consistency, many shared resources do not allow simultaneous accesses by competing tasks, but require their mutual exclusion. This means that a task cannot access a resource R if another task is inside R manipulating its data structures. In this case, R is called a *mutually exclusive resource*. A piece of code executed under mutual exclusion constraints is called a *critical section*.

Basic Concepts: Definition of Scheduling Problems

Basic Concepts : Definition of Scheduling Problems

In general, to define a scheduling problem we need to specify three sets: a set of n tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$, a set of m processors $P = \{P_1, P_2, \ldots, P_m\}$ and a set of s types of resources $R = \{R_1, R_2, \ldots, R_s\}$. Moreover, precedence relations among tasks can be specified through a directed acyclic graph, and timing constraints can be associated with each task. In this context, scheduling means assigning processors from P and resources from R to tasks from Γ in order to complete all tasks under the specified constraints . This problem, in its general form, has been shown to be NP-complete and hence computationally intractable.

Indeed, the complexity of scheduling algorithms is of high relevance in dynamic real-time systems, where scheduling decisions must be taken on line during task execution.

1. Classification of Scheduling Algorithms

Among the great variety of algorithms proposed for scheduling real-time tasks, the following main classes can be identified:

- **Preemptive vs. Non-preemptive.**
  - In preemptive algorithms, the running task can be interrupted at any time to assign the processor to another active task, according to a predefined scheduling policy.
  - In non-preemptive algorithms, a task, once started, is executed by the processor until completion. In this case, all scheduling decisions are taken as the task terminates its execution.

Basic Concepts : Definition of Scheduling Problems

1. Classification of Scheduling Algorithms

- **Static vs. Dynamic.**
  - Static algorithms are those in which scheduling decisions are based on fixed parameters, assigned to tasks before their activation.
  - Dynamic algorithms are those in which scheduling decisions are based on dynamic parameters that may change during system evolution.
- **Off-line vs. Online.**
  - A scheduling algorithm is used off line if it is executed on the entire task set before tasks activation. The schedule generated in this way is stored in a table and later executed by a dispatcher.
  - A scheduling 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.

1. Classification of Scheduling Algorithms

- **Optimal vs. Heuristic.**

  – An algorithm is said to be optimal if it minimizes some given cost function defined over the task set. When no cost function is defined and the only concern is to achieve a feasible schedule, then an algorithm is said to be optimal if it is able to find a feasible schedule, if one exists.

  – An algorithm is said to be heuristic if it is guided by a heuristic function in taking its scheduling decisions. A heuristic algorithm tends toward the optimal schedule, but does not guarantee finding it.

Moreover, an algorithm is said to be *clairvoyant* if it knows the future; that is, if it knows in advance the arrival times of all the tasks. Although such an algorithm does not exist in reality, it can be used for comparing the performance of real algorithms against the best possible one.

1. Classification of Scheduling Algorithms

GUARANTEE-BASED ALGORITHMS

In hard real-time applications that require highly predictable behavior, the feasibility of the schedule should be guaranteed in advance; that is, before task execution. In this way, if a critical task cannot be scheduled within its deadline, the system is still in time to execute an alternative action, attempting to avoid catastrophic consequences. In order to check the feasibility of the schedule before tasks' execution, the system has to plan its actions by looking ahead in the future and by assuming a worst-case scenario.

BEST-EFFORT ALGORITHMS

In certain real-time applications, computational activities have soft timing constraints that should be met whenever possible to satisfy system requirements. In these systems, missing soft deadlines do not cause catastrophic consequences, but only a performance degradation.

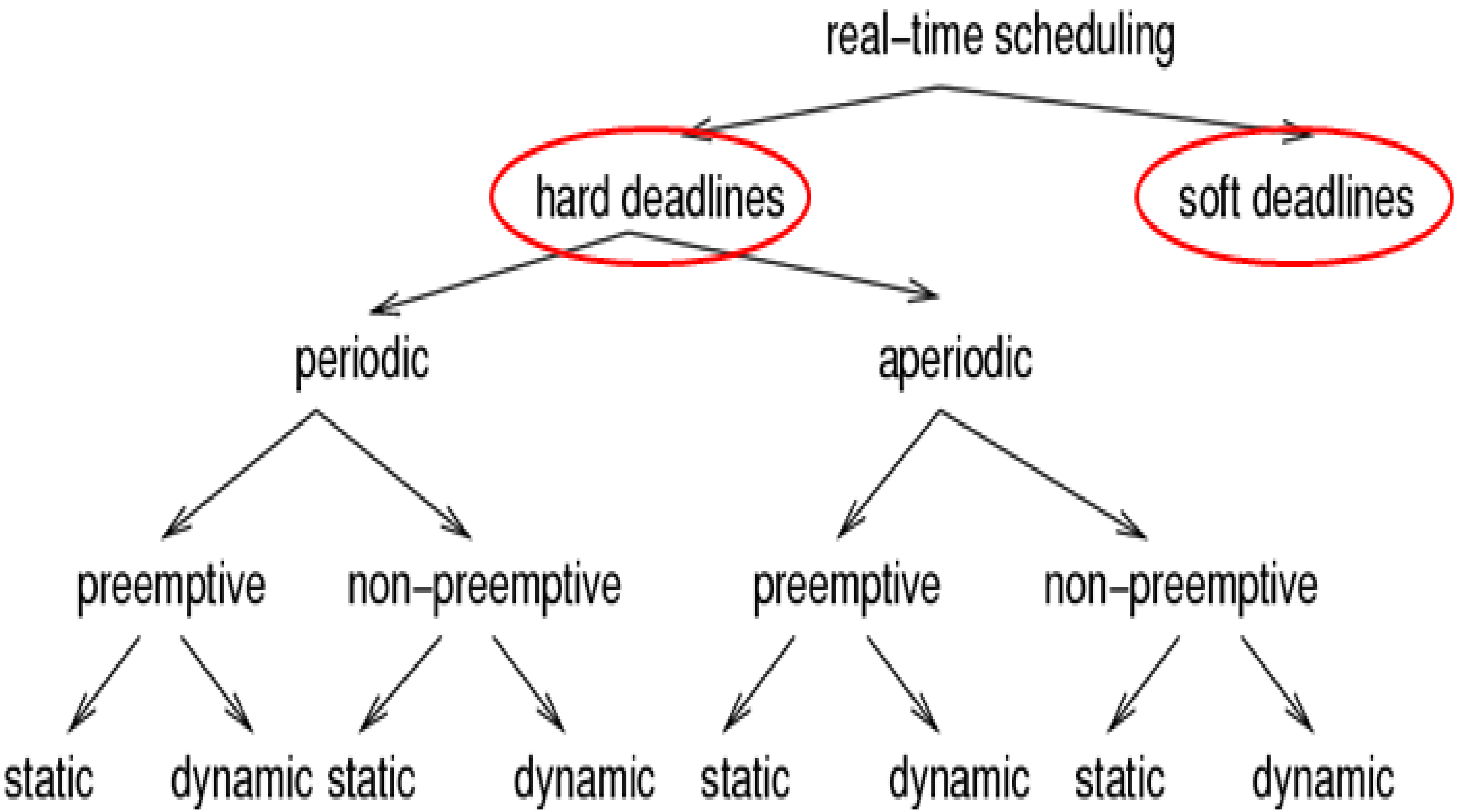
1. Classification of Scheduling Algorithms

Periodic Vs Aperiodic

A real-time system consists of both **aperiodic** and **periodic** tasks.

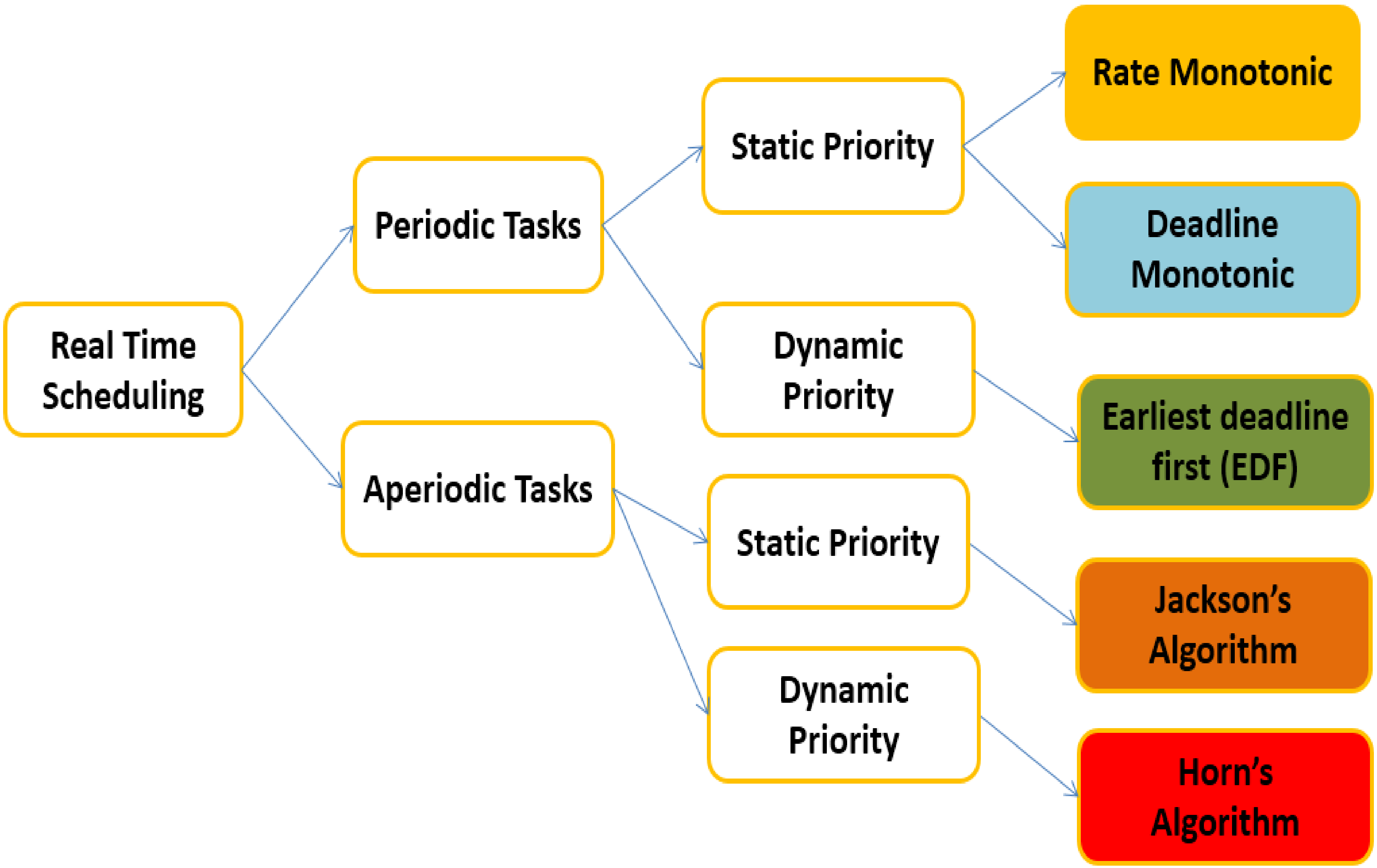
- **Periodic tasks** have regular arrival times and hard deadlines.
- **Aperiodic tasks** have irregular arrival times and either soft or hard deadlines.

1. Classification of Scheduling Algorithms



Basic Concepts : Definition of Scheduling Problems

2. Some Scheduling Algorithms



3. Metrics for Performance Evaluation

The performance of scheduling algorithms is typically evaluated through a cost function defined over the task set. For example, classical scheduling algorithms try to minimize the average response time, the total completion time, the weighted sum of completion times, or the maximum lateness. When deadlines are considered, they are usually added as constraints, imposing that all tasks must meet their deadlines. If some deadlines cannot be met with an algorithm A, the schedule is said to be infeasible by A.

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 completion 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

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

Table Example of cost functions.

Periodic Task Scheduling:

Rate Monotonic Scheduling

Periodic Task Scheduling: Rate Monotonic Scheduling

The Rate Monotonic (RM) scheduling algorithm is a simple rule that assigns priorities to tasks according to their request rates. Specifically, tasks with higher request rates (that is, with shorter periods) will have higher priorities. Since periods are constant, RM is a fixed-priority assignment: a priority P_i is assigned to the task before execution and does not change over time. Moreover, RM is intrinsically preemptive: the currently executing task is preempted by a newly arrived task with shorter period.

Periodic Task Scheduling: Rate Monotonic Scheduling

Processor utilization analysis

Liu & Layland (1973) proved that for a set of n periodic tasks with unique periods, a feasible schedule that will always meet deadlines exists if the CPU utilization is below a specific bound (depending on the number of tasks). The schedulability test for RMS is:

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

where C_i is the computation time, T_i is the release period (with deadline one period later), and n is the number of processes to be scheduled.

Periodic Task Scheduling: Rate Monotonic Scheduling

Simple feasibility test for RM (Sufficient condition)

Observe that it is possible to derive a conservative lower bound on utilization by letting $n \rightarrow \infty$

$$\lim_{n \to \infty} n\left(2^{1/n} - 1\right) = \ln 2 \approx 0.693$$

This means that a set of tasks (**regardless of number of tasks**) whose total utilization does not exceed **0.693** is **always schedulable with RM**!

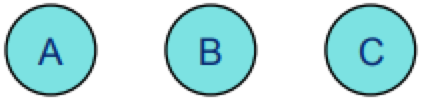
Periodic Task Scheduling: Rate Monotonic Scheduling

Example: Scheduling using RM

Problem (1):

Assume a system with tasks according to the figure below. The timing properties of the tasks are given in the table.

1. Schedule the tasks using rate-monotonic scheduling (RM).
2. What is the utilization of the task set?
3. What is the outcome of Liu & Layland's feasibility test for RM?



Task T	Execution Time C	Period P
T1	3	20
T2	2	5
T3	2	10

Lets solve this problem together on the blackboard

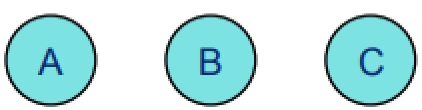
Periodic Task Scheduling: Rate Monotonic Scheduling

Example: Scheduling using RM

Problem (2):

Assume a system with tasks according to the figure below. The timing properties of the tasks are given in the table.

1. Schedule the tasks using rate-monotonic scheduling (RM).
2. What is the utilization of the task set?
3. What is the outcome of Liu & Layland's feasibility test for RM?



Task T	Execution Time C	Period P
T1	1	4
T2	2	5
T3	5	20

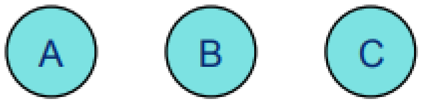
Periodic Task Scheduling: Rate Monotonic Scheduling

Example: Scheduling using RM

Problem (3):

Assume a system with tasks according to the figure below. The timing properties of the tasks are given in the table.

1. Schedule the tasks using rate-monotonic scheduling (RM).
2. What is the utilization of the task set?
3. What is the outcome of Liu & Layland's feasibility test for RM?



Task T	Execution Time C	Period P
T1	1	3
T2	1	4
T3	1	5

Periodic Task Scheduling:

Deadline Monotonic Algorithm

Periodic Task Scheduling: Deadline Monotonic Algorithm

Definition

DMA: a Hard dynamic preemptive algorithm based on static priorities

- DMA assigns priorities to tasks based on their **deadlines.**
- DMA assigns **higher priorities** to tasks with **shorter deadlines.**
- An optimal algorithm in the case of static priority algorithms with **smaller deadlines than periods : Deadline < Period**

Rules:

- **The priority of a task is set by its deadline.**
- **The shorter the deadline, the higher the priority.**

Periodic Task Scheduling: Deadline Monotonic Algorithm

2

Exemples

How to apply the algorithm?

Periodic Task Scheduling: Deadline Monotonic Algorithm

Exemple 1

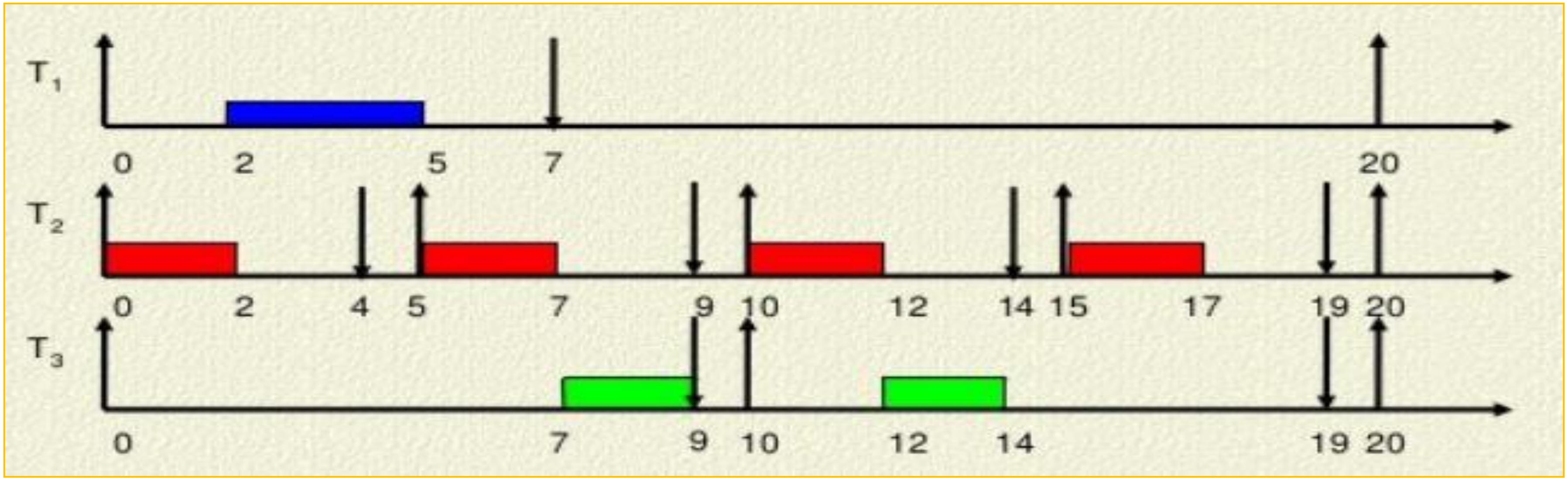
Task	Cost	Deadline	Period
T1	3	7	20
T2	2	4	5
T3	2	9	10

Periodic Task Scheduling: Deadline Monotonic Algorithm

Solution :Example 1

$\Sigma\ Ci/Pi = 3/7+2/4+2/9 = 1{,}14 \geqslant n(2^{1/n}-1)=0{,}77$

Task	Cost	Deadline	Period
T1	3	7	20
T2	2	4	5
T3	2	9	10



Exemple 1

Cost : mean run time cost of generated processes.

Deadline : mean deadline value for generated processes.

Periodic Task Scheduling: **Deadline Monotonic Algorithm**

DMA

- **Algorithm with constant priority**
- **Acceptability test (sufficient condition) :**

$$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N\left(2^{1/N} - 1\right)$$

- when n is very large:

$$n(2^{1/n} - 1) \sim \ln 2 = 0{,}69$$

- Equivalent to RMA for tasks that are due on request, better in other cases.

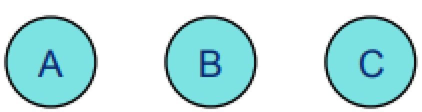
Periodic Task Scheduling: Deadline Monotonic Algorithm

Example: Scheduling using DMA

Problem (4):

Assume a system with tasks according to the figure below. The timing properties of the tasks are given in the table.

1. Schedule the tasks using rate-monotonic scheduling (RM).
2. What is the utilization of the task set?
3. What is the outcome of Liu & Layland's feasibility test for RM?

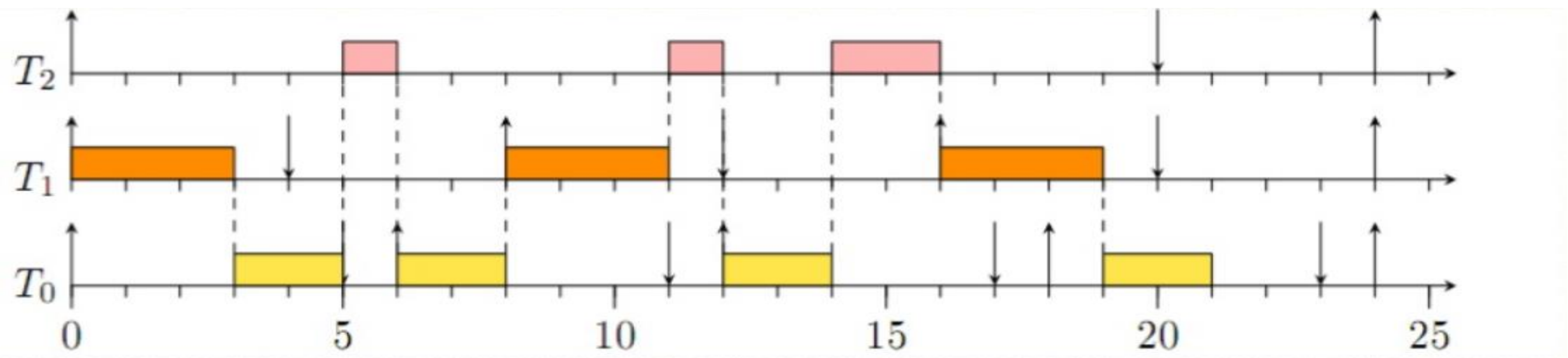


Task	Cost	Deadline	Period
T0	2	6	5
T1	3	5	4
T2	4	24	20

Periodic Task Scheduling: Deadline Monotonic Algorithm

Solution: Example 2

Task	Cost	Deadline	Period
T0	2	6	5
T1	3	5	4
T2	4	24	20



- **Note that task T1 has the highest priority because it has a shorter deadline than the other tasks, while task T2 is the last one to execute because it corresponds to the largest deadline.**

RMS vs DMS

- Rate monotonic is a special case when all Di = T
- Rate Monotonic Scheduling has shown to be optimal among static priority policies.
- Some task sets that aren't schedulable using RMS can be scheduled using dynamic strategies.
- An example is a task set where the deadline for completing processing is not the task period (the deadline is some time shorter than the task period).

References

- [Web1] : http://www.cis.upenn.edu/~lee/home/research/index.shtml
- [Web2] https://engineering.wustl.edu/Pages/PageNotFoundError.aspx?requestUrl=https://engineering.wustl.edu/newsstory.aspx
- [Web3] https://sites.google.com/site/realtimexen/
- [Web4] https://www.philadelphia.edu.jo/academics/kaubaidy/uploads/RTS-lec4.pdf
- [Web5] https://en.wikipedia.org/wiki/Real-time_computing
- [Web6] https://www.techopedia.com/definition/16991/real-time-computing-rtc
- [Web7] http://beru.univ-brest.fr/~singhoff/ENS/USTH/intro.pdf
- [Web8] https://ti.tuwien.ac.at/cps/teaching/courses/real-time-systems/slides/rts01_definitions.pdf
- [Web9] https://en.wikipedia.org/wiki/Real-time_computing
- [Web10] http://www.smartcockpit.com/docs/A380_Briefing_For_Pilots_Part%202.pdf
- [Web11] https://en.wikipedia.org/wiki/Real-time_computing
- [STA 88] John Stankovic. « Misconceptions about real-time computing ». IEEE Computer, October 1988.
- [SCH92] Schuster, Herbert Der neue Airbag von Volkswagen Automobiltechnische Zeitung (ATZ), 1992, Heft 3, Seite 110f
- [DOR 91] A. Dorseuil and P. Pillot. Le temps réel en millieu industriel. Edition DUNOD, Collection Informatique Industrielle, 1991.
- [DEM 99] I. Demeure and C. Bonnet. Introduction aux systèmes temps réel. Collection pédagogique de télécommunications, Hermès, septembre 1999.
- [COU 94] G. Coulouris, J. Dollimore, and T. Kindberg. Distributed Systems—Concepts and Design, 2nd Ed. Addison-Wesley Publishers Ltd., 1994.
- [Web12]http://elib.dlr.de/106629/1/LUBETA_DLRK2016.pdf
- [R.Thomas] Advances in air to air refuelling Peter R. Thomas a,n , Ujjar Bhandari a , Steve Bullock a , Thomas S. Richardson a , Jonathan L. du Bois b, https://doi.org/10.1016/j.paerosci.2014.07.001, Volume 71, November 2014, Pages 14-35
- [Book1] Michael Blaha , James Rumbaugh , Object-Oriented Modeling and Design with UML Second Edition, ISBN 0-13-015920-4, Pearson Prentice Hall, 2005.
- [Book2] BUI MINH DUC, Real-Time Object Uniform Design Methodology with UML, ISBN 978–1–4020–5976–6, ISBN 978–1–4020–5977–3, Published by Springer, P.O. Box 17, 3300 AA Dordrecht, The Netherlands, 2007
- [Book3] , Giorgio C. Buttazzo, Hard Real-Time Computing Systems, Predictable Scheduling Algorithms and Applications, Third Edition, © Springer Science+Business Media, LLC 2011 , e-ISSN 1867-3228, e-ISBN 978-1-4614-0676-1
- [Web13] https://www.smartdraw.com/state-diagram/
- [Web14] https://en.wikipedia.org/wiki/Rate-monotonic_scheduling

Greek alphabet list

Upper Case Letter	Lower Case Letter	Greek Letter Name	English Equivalent
Α	α	Alpha	a
Β	β	Beta	b
Γ	γ	Gamma	g
Δ	δ	Delta	d
Ε	ε	Epsilon	e
Ζ	ζ	Zeta	z
Η	η	Eta	h
Θ	θ	Theta	th
Ι	ι	Iota	i
Κ	κ	Kappa	k
Λ	λ	Lambda	l
Μ	μ	Mu	m
Ν	ν	Nu	n
Ν	ν	Nu	n
Ξ	ξ	Xi	x
Ο	ο	Omicron	o
Π	π	Pi	p
Ρ	ρ	Rho	r
Σ	σ,ς *	Sigma	s
Τ	τ	Tau	t
Υ	υ	Upsilon	u
Φ	φ	Phi	ph
Χ	χ	Chi	ch
Ψ	ψ	Psi	ps
Ω	ω	Omega	o