Real-Time systems

System Model

Giuseppe Lipari

Scuola Superiore Sant’Anna

Pisa -Italy

Task Model

Mathematical model of a task

- A task τ_i is a (infinite) sequence of jobs (or instances) $J_{i,k}$.
- Each job $J_{i,k} = (a_{i,k}, c_{i,k}, d_{i,k})$ is characterized by:
  - an activation time (or arrival time) $a_{i,k}$;

    $\rightarrow$ It is the time when the job is *activated* by an event of by a condition;
  - a computation time $c_{i,k}$;

    $\rightarrow$ It is the time it takes to complete the job;
  - an absolute deadline $d_{i,k}$

    $\rightarrow$ it is the absolute instant by which the job must complete.
  - the job finishes its execution at time $f_{i,j}$;

    $\rightarrow$ the response time of job $J_{i,j}$ is $\rho_{i,j} = f_{i,j} - a_{i,j}$;

    $\rightarrow$ for the job to be correct, it must be $f_{i,j} \leq d_{i,j}$.

Task model

A task can be:

- *periodic*: has a regular structure, consisting of an infinite cycle, in which it executes a computation and then suspends itself waiting for the next periodic activation. An example of pthread library code for a periodic task is the following:

```
void * PeriodicTask(void *arg)
{
  <initialization>;
  <start periodic timer, period = T>;
  while (cond) {
    <read sensors>;
    <update outputs>;
    <update state variables>;
    <wait next activation>;
  }
}
```

Model of a periodic task

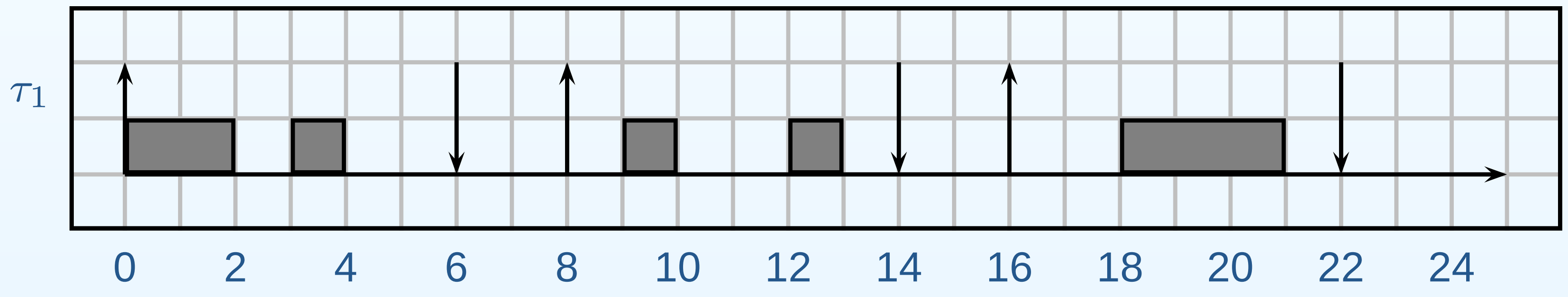
From a mathematical point of view, a periodic task $\tau_i = (C_i, D_i, T_i)$ consists of a (infinite) sequence of jobs $J_{i,k}$, $k = 0, 1, 2, \ldots$, with

$$\begin{array}{rrcl} & a_{i,0} & = & 0 \\ \forall k > 0 & a_{i,k} & = & a_{i,k-1} + T_i \\ \forall k \geq 0 & d_{i,k} & = & a_{i,k} + D_i \\ & C_i & = & \max\{k \geq 0 | c_{i,k}\} \end{array}$$

- T_i is the task's period;
- D_i is the task's relative deadline;
- C_i is the task's worst-case execution time (WCET);
- R_i is the worst-case response time: $R_i = max_j\{\rho_{i,j}\}$;
  - for the task to be schedulable, it must be $R_i \leq D_i$.

Graphical representation

In this course, the tasks will be graphically represented with a GANNT chart. In the following example, we graphically show periodic task $\tau_1 = (3, 6, 8)$.



Notice that, while job $J_{i,0}$ and $J_{i,3}$ execute for 3 units of time (WCET), job $J_{i,2}$ executes for only 2 units of time.

Sporadic tasks

- *sporadic* tasks are very similar to periodic tasks. But, instead of suspending themselves on timer events, they wait for other events, which are not periodic in general (for example, the arrival of a packet from the network).
- however, for sporadic tasks, it is possible to define a *minimum interarrival time* between two occurrences of the event.

```
void * SporadicTask(void *)
{
  <initialization>;
  while (cond) {
    <computation>;
    <wait event>;
  }
}
```

Mathematical model of a sporadic task

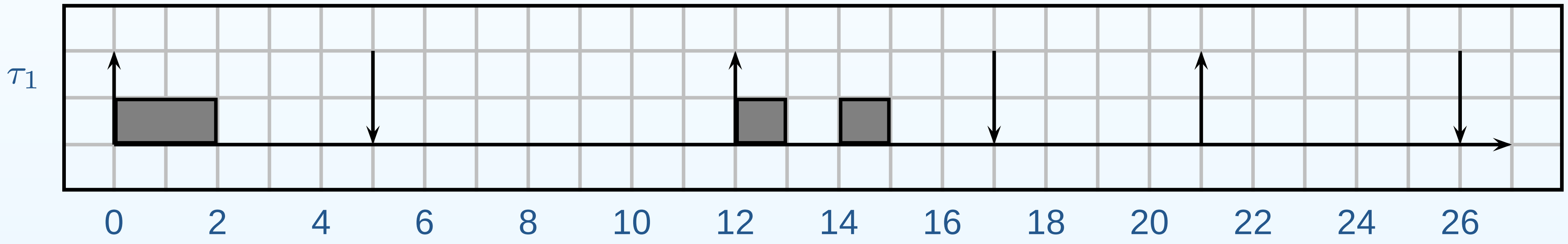
Similar to a periodic task: sporadic task $\tau_i = (C_i, D_i, T_i)$ consists of a (infinite) sequence of jobs $J_{i,k}$, $k = 0, 1, 2, \ldots$, with

$$\begin{aligned} \forall k > 0 \quad a_{i,k} &\enclose{circle}{\geq} a_{i,k-1} + T_i \\ \forall k \geq 0 \quad d_{i,k} &= a_{i,k} + D_i \\ C_i &= \max\{k \geq 0 | c_{i,k}\} \end{aligned}$$

- T_i is the task's minimum interarrival time (MIT);
- D_i is the task's relative deadline;
- C_i is the task's worst-case execution time (WCET).

Graphical representation

In the following example, we show sporadic task $\tau_1 = (2, 5, 9)$.



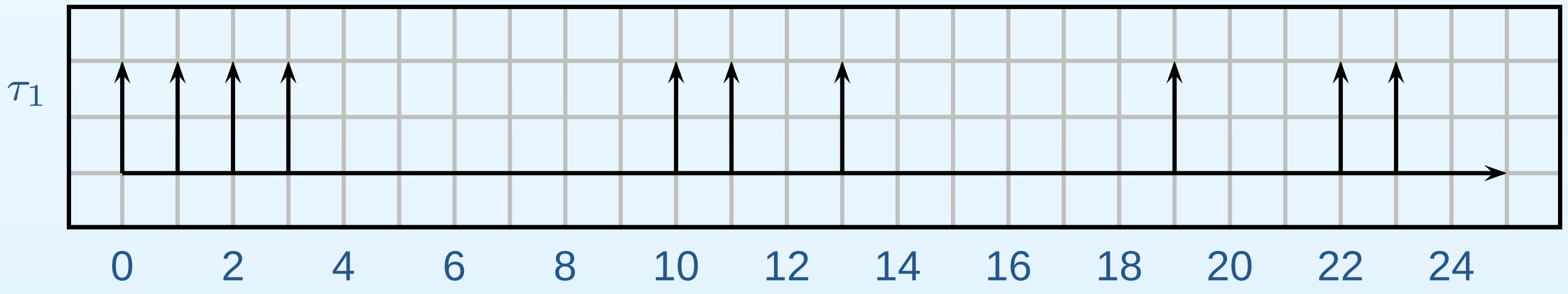
Notice that

$$\begin{aligned} a_{1,1} &= 12 > a_{1,0} + T_1 = 9. \\ a_{1,2} &= 21 = a_{1,1} + T_1 = 9 \end{aligned}$$

Aperiodic Tasks

For them, it is not possible to set a minimum separation time between two consecutive jobs. Also, they do not have a particular structure. With this kind of task we can model:

- Tasks that respond to events that occur rarely. Example: a mode change.
- Tasks that respond to events that happen with an irregular structure. Example: bursts of packects arriving from the network.



Hyperperiod

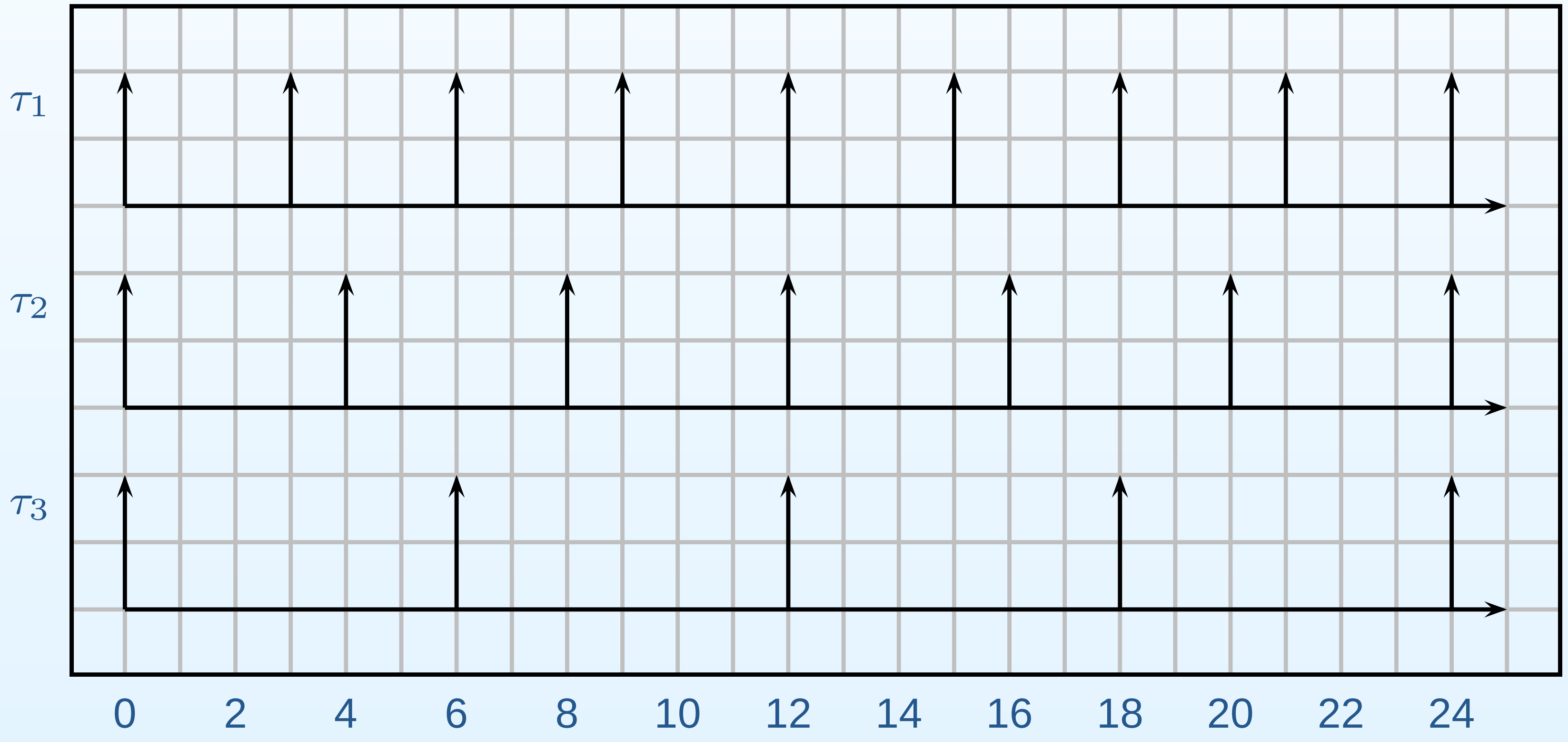
- A task set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ is periodic if it consists of periodic tasks only.
- The *hypeperiod* of a periodic task set is the least common multiple (lcm) of the task's periods;

$$H(\mathcal{T}) = \text{lcm}_{\tau_i \in \mathcal{T}}(T_i)$$

- The patterns of arrival repeats every hypeperiod. In practice, if two tasks arrive at the same time t, they will arrive at the same time $t + kH$, for every integer number $k \geq 0$;
- Sometimes, the hyperperiod is defined also for task sets consisting or periodic and sporadic tasks. The meaning is slightly different.

Hyperperiod: example

- Consider a task set with 3 tasks, with the following periods: $T_1 = 3$, $T_2 = 4$, $T_3 = 6$. The hyperperiod is $H = 12$. We show below the pattern of arrivals.



Offsets

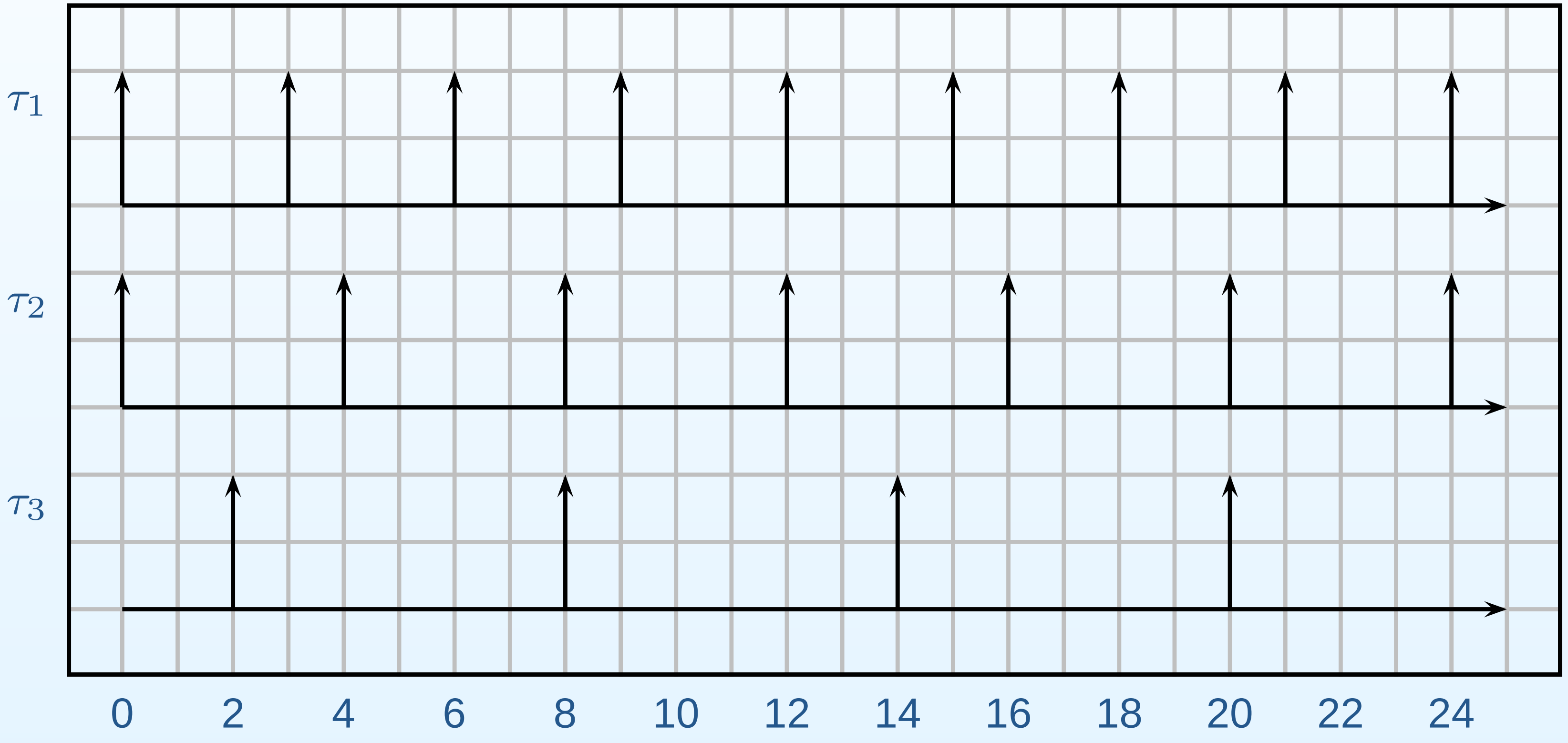
- A periodic task can have an *initial offset* ϕ_i
- The offset is the arrival time of the first instance of a periodic task;
- Hence:

$$\begin{aligned} a_{i,0} &= \phi_i \\ a_{i,k} &= \phi_i + kT_j \end{aligned}$$

- In some case, offsets are set to a value different from o to avoid all tasks starting at the same time.

Example of offsets

- Consider the previous example, 3 tasks with the following periods and offsets: $T_1 = 3\ \phi_1 = 0$, $T_2 = 4\ \phi_2 = 0$, $T_3 = 6\ \phi_3 = 2$.



Hard real-time tasks

- a task is said *hard real-time* if all the its deadlines must be respected, otherwise a critical failure occurs in the system;
  - therefore, we have to guarantee *a-priori*, before the system runs, that all deadlines will be respected under all possible conditions;
  - $R_i \leq D_i$;
- An example of hard real-time task:
  - from the detection of an enemy missile, a maximum of 2 seconds must pass before the defense missile is launched and directed on target, otherwise it will be too late!
  - Notice the double difficulty: the computation of the trajectory must be as much precise as it is possible, *and* it must be performed within 2 seconds!

Soft real-time tasks

- in a *soft real-time* task, nothing “catastrophic” happens if a deadline is missed;
- some deadline can be missed with little or no consequences on the correcteness of the system;
- however, the number of missed deadline must be kept under control, because the “quality” of the results depend upon on the number of deadline missed;

Requirements on soft real-time tasks

- Example of requirements on soft tasks;
  - no more than X consecutive deadlines can be missed;
  - no more than X deadline in an interval of time T;
  - the *deadline miss ratio* (i.e. percentage or total missed deadlines over the total number of deadlines) must not exceed a certain threshold;
  - the maximum *tardiness* (i.e. the ratio of the worst-case response time and the relative deadline) must not exceed a certain threshold;
  - etc.

Example of soft real-time task

The classical example of soft real-time task is a MPEG player.

- usually, the frame rate of a video movie is 25 fps;
- this means that one video frame has to be loaded from disk, decoded and displayed every 40 msec;
- if some of the frames is displayed with a little delay, the user may not even be able to perceive the effect; however, if frames are skipped or displayed too late, the disturbance is evident, and the user is not satisfied;
- the ideal objective is to avoid $\underline{\text{any}}$ delay (hard real-time task); however, this may involve a costly hardware;
- therefore, it is better to allow an occasional delay and save on the processor power (soft real-time task);
- the goal is then to *minimize* the number of missed deadlines.

Hard and soft

- In a system, often there is a mixture of hard and soft real-time tasks. For example:
    - In a critical system, in addition to critical control tasks, we have logging tasks that collect information on the functionality of the system for monitoring;
    - Also, communication over the network of this information is not a critical task, and if some packet is lost nothing catastrophic happens;
    - Some control task can also be non critical; for example, in a robotic system, some actuation may be delayed a little more with little consequences (degradation of the quality of the control).

Hard and soft

- In reality, very few tasks require strict hard real-time behaviour. However, it is important to always know a-priori is a task will miss its deadline.
- *In this course, we will concentrate on hard real-time tasks.*
- In a second part, we will show some technique to deal with soft real-time tasks and mixtures of both hard and soft tasks.

Computation times and workload

Worst case and average case

- Tasks can have variable execution times between different jobs
- The execution time depends on different things:
  - input data;
  - the hardware architecture (presence of cache, pipeline, etc.);
  - internal conditions;

Worst-case computation: example

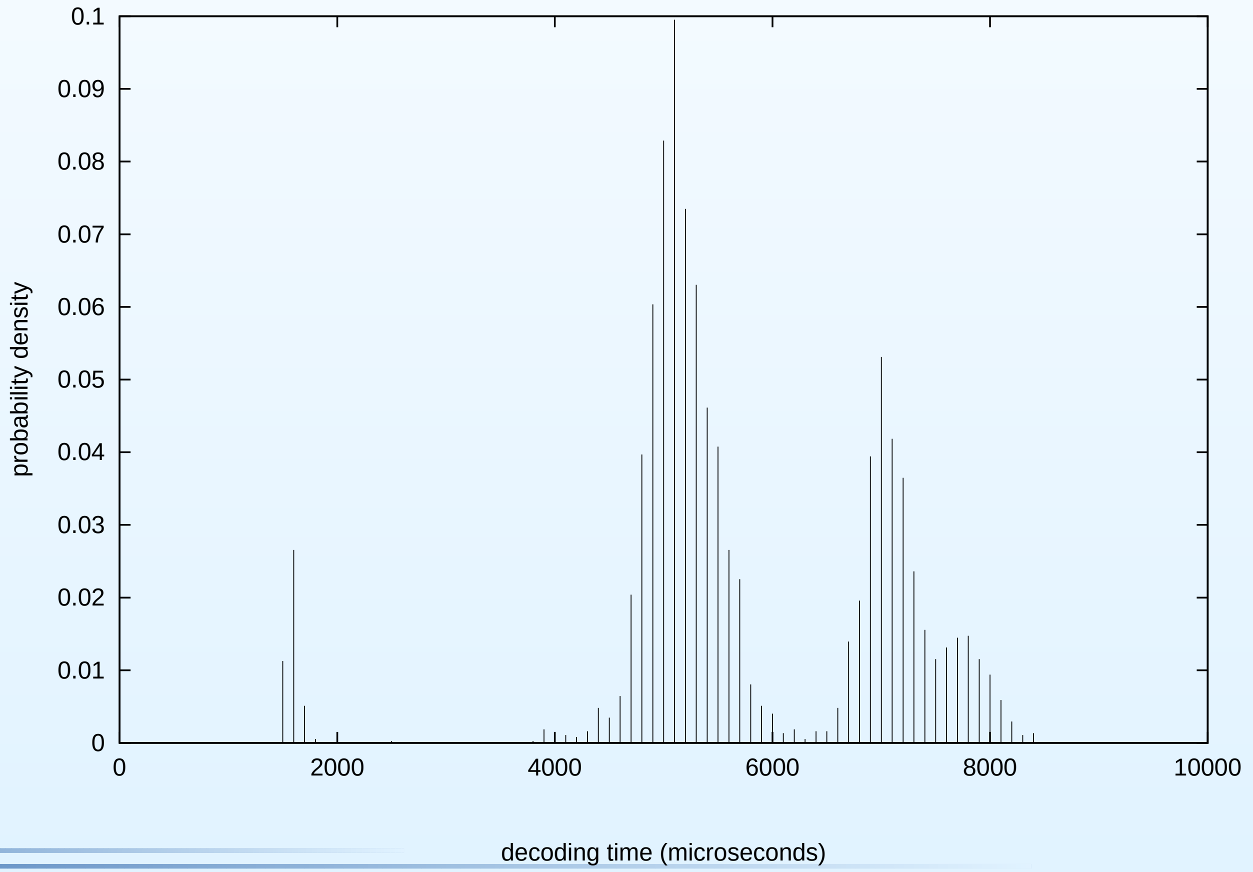
As an example, consider the following pseudo-code:

```
while (cond) {
  if (a > 10) {
    // long computation
  } else {
    // short computation
  }
  a = // external input;
}
```

We have two sources of variability: the value of variable `a` influences which computation is performed (long or short); the value of `cond` influences the number of times the cycle is executed.

Example: MPEG player

- An algorithm with highly variable computation time is the decoding of a MPEG frame: in the figure below, we show the distribution of the execution times of decoding “Star Wars”.



Computing the WCET

- Computing the WCET of a task is possible in principle;
  - It consists in computing all possible paths that the program may follow;
  - Computing the number of processor cycles needed for every path;
  - Selecting the path with the maximum number of cycles;
- However, in practice the problem is almost intractable:
  - the presence of the cache and of the pipeline makes the problem of computing the number of cycles very difficult;
  - most algorithm can only give an upper bound (i.e. the selected path may not be possible in the program execution, the number of cycles is overestimated, etc.);
  - unfortunately such bounds are often too large (like 2 or 3 times the real WCET);

Computing the WCET - II

- practically, today the WCET is estimated by executing the program several times over a large number of different input data;
- however, since we cannot exeute the program for *every* input data, the measure bound is not safe!
  - (i.e. the real WCET may be *larger* than the one that has been measured).

Maximum Utilization

- The maximum *utilization* (or load) of a task τ_i is defined as:

$$U_i = \frac{C_i}{T_i}$$

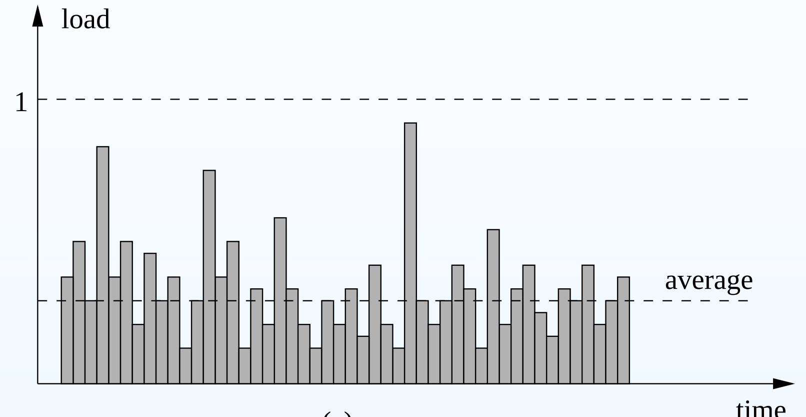
- The maximum *system utilization* (or workload) is defined as:

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

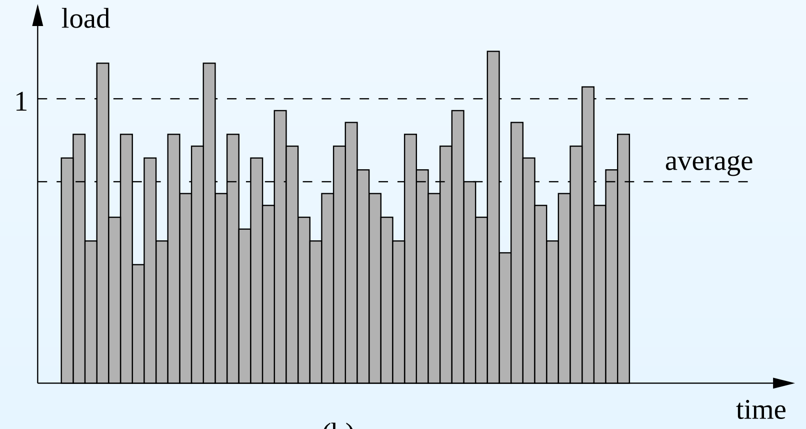
- Example:
  - $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 8, 8)$, $\tau_3 = (4, 10, 12)$
  - $U = 0.91\overline{6} = 92\%$

Maximum and average workload

- The maximum system workload is computed based on the WCET (C_i); this means that the actual workload may be less than the maximum;
- If the actual load is more than 1, some deadline can be missed!



(a)



(b)

Bound on the workload

- To be safe, in a hard real-time task we require the maximum workload to be below the *utilization bound*;
  - the utilization bound is the maximum utilization such that no deadline is missed with a given scheduling algorithm (see later. . .);
  - the utilization bound depends on the scheduling algorithm. For example, for the EDF scheduling algorithm, the utilization bound is 100%, for RM is 69%.
  - this is a *necessary* condition for schedulability!
- In a soft real-time system, designing the system for the maximum workload may be too conservative;
- Therefore, we take into account the *average workload*, that is the workload computed considering the average-case execution times (ACET) of the tasks.

Scheduling

Schedule

Let us consider a system consisting of one processor.

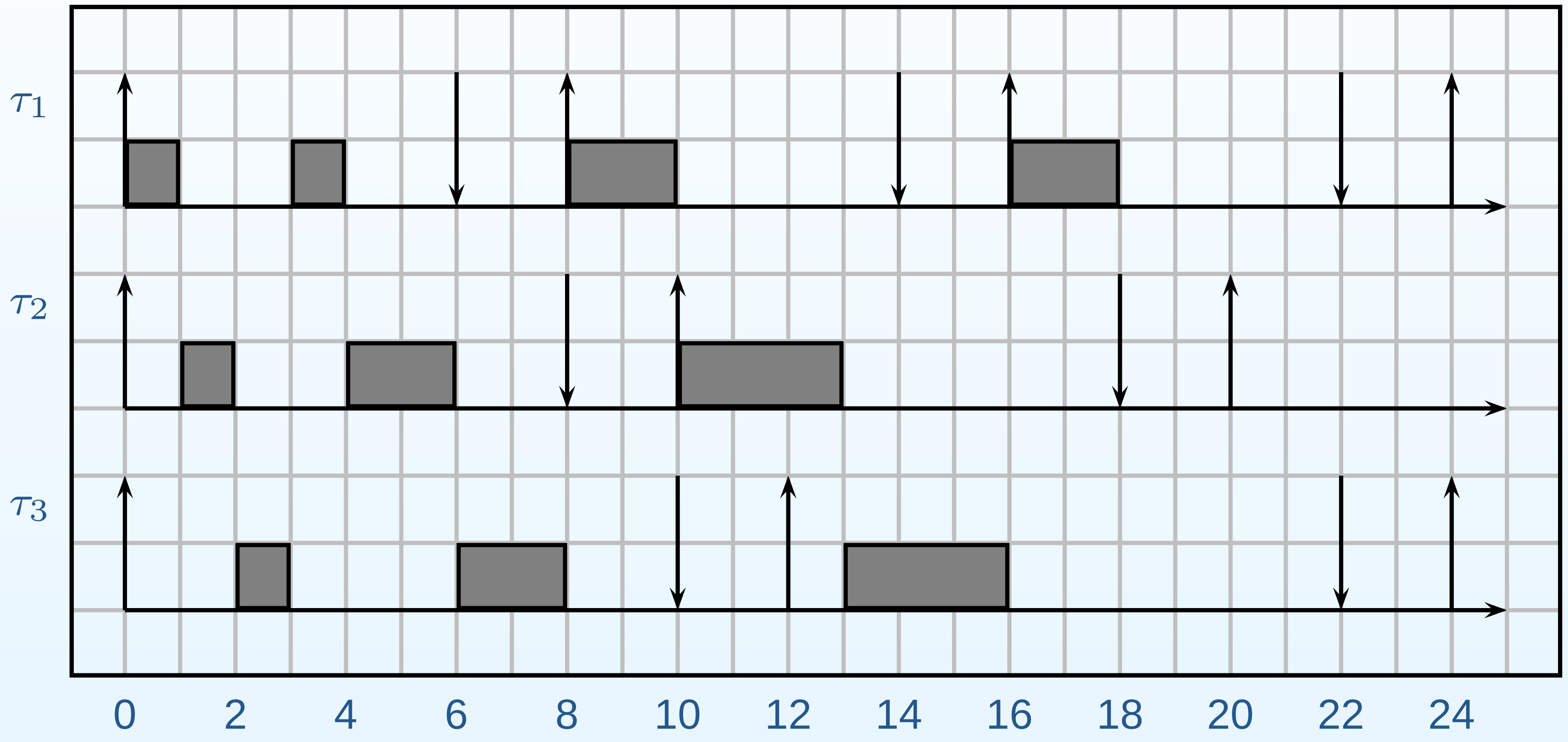
- a *schedule* is a function $\sigma(t)$ that maps each instant of time into a task or to $\emptyset$.

$$\sigma : t \to \mathcal{T} \cup \emptyset$$

- If $\sigma(t) = \emptyset$, then the processor is idle.
- The definition can be easily extended to m processors, by considering a function $\sigma(t)$ that maps instants of time into vectors of elements of $\mathcal{T} \cup \emptyset$.

Graphical representation of a schedule

We will use GANNT charts to show a schedule, like the following:



$$\sigma(3) = \tau_1, \quad \sigma(10) = \tau_2, \quad \sigma(18) = \emptyset, \ldots$$

Scheduling algorithm

- A scheduling algorithm $\mathcal{A}$ is an algorithm that selects for each instant of time t a task to be executed on the processor among the ready tasks. Given a task set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, a scheduling algorithm $\mathcal{A}$ generates a schedule $\sigma_{\mathcal{A}}(t)$.
- A task set is schedulable by algorithm $\mathcal{A}$ if, in the generated schedule, all jobs of all tasks complete before deadlines:

$$\forall i, j \quad f_{i,j} \leq d_{i,j}$$

- A schedulability test for algorithm $\mathcal{A}$ is an algorithm that, given a task set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, returns **YES** if the task set is schedulable by algorithm $\mathcal{A}$.

Necessary and sufficient tests

- The schedulability test is necessary and sufficient if returns **YES** if and only if the task set is schedulable. Otherwise it returns **NO**.
- If the schedulability test is only sufficient, then if it returns **YES**, the task set is schedulable; if it returns **NO**, then the task set may or may not be schedulable.

Scheduling algorithms

- There are several scheduling algorithms; for each scheduling algorithm, there are many schedulability tests
- In this course we will study:
  - The Fixed Priority (FP) scheduling algorithm, and its derivates Rate Monotonic (RM) and Deadline Monotonic (DM);
  - For this algorithm, we will present the Utilization Bound test, and the Response Time Analysis;
  - Then, we will present the Earliest Deadline First EDF scheduling algorithm;
  - For this algorithm, we will present both the Utilization Bound test, and the Processor Demand Bound test.

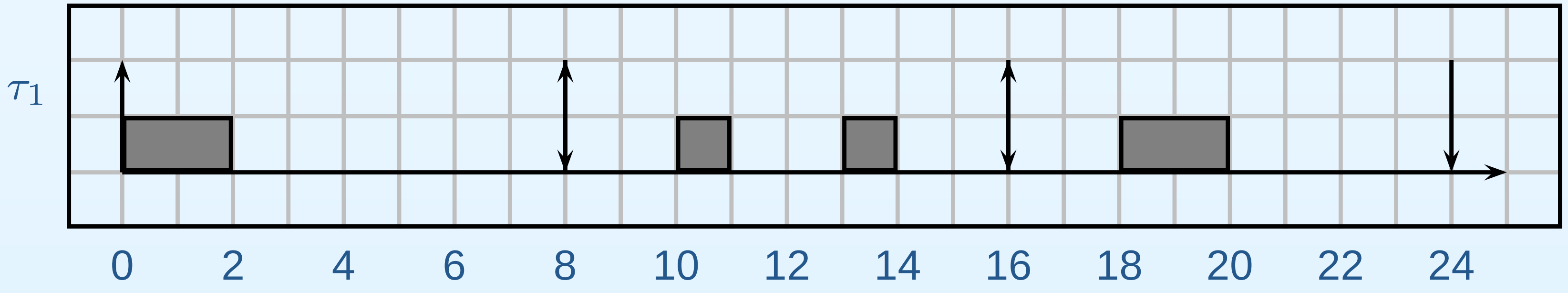
Other parameters

Other important parameters

- in some case, even if the job has been activated, cannot start executing because it must wait an additional event;
  - the *start time jitter* stj_i is the maximum delay between the activation time a_{ij} and the time the job can actually start executing;
- the *output jitter* oj_i is the maximum of the absolute value of the difference between the distance of two consecutive finishing times and the period:

$$oj_i = max_k\{|f_{i,k+1} - f_{i,k} - T_i|\}$$

Example:



$oj_1 = max\{|14 - 2 - 8|, |20 - 14 - 8|\} = 4.$

Considerations on the output jitter

- Usually, a job produces its outputs at the end of its execution;
- for example, just before suspending, a task sends the control command to the actuators;
- in general, it is a good idea to provide the actuation command at regular (periodic) instants;
- If we know the delay, and it is constant, it is possible to compensate for it at design time;
  - ideally, the output jitter should be equal to 0;
  - This means that the task finishes always at the same time with respect to the activation instant: $\rho_{i,k} = const$
  - however, due to the presence of other tasks, and to scheduling, this may be impossible.
- the larger is the output jitter, the larger is the “noise” on the actuator.