IMC4-2RT

Real-time scheduling

Damien MASSON
http://esiee.fr/~massond/Teaching/

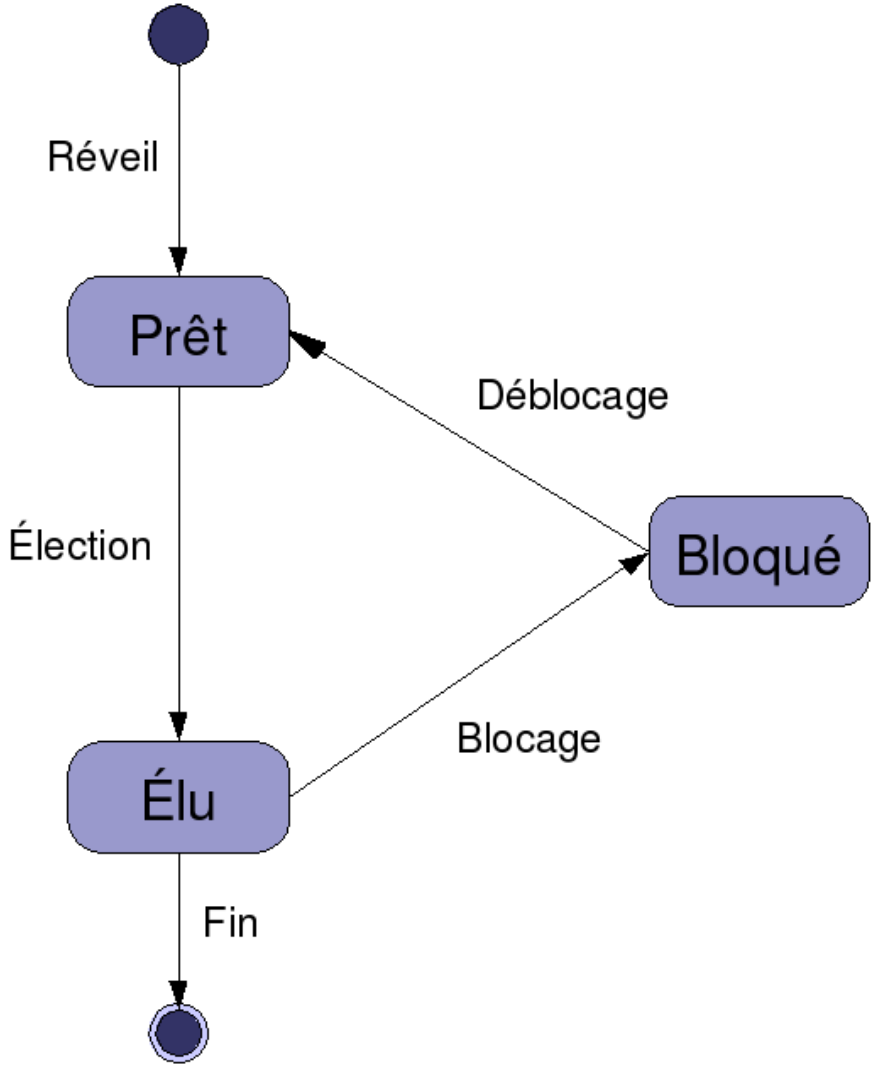
last modification: December 1, 2014

References

- Hard real-time computing systems: predictable scheduling algorithms and applications, Giorgio C. Buttazzo, Springer, 2005 - 425 pages (`http://books.google.com/books/about/Hard_real_time_computing_systems.html?id=fpJAZM6FK2sC`)
- Internet

Concurrency

Operating systems are mutlitasks, even on monoprocessors architectures. How is it possible ? with processes !



Real-Time

Real-Time: different from quick, more synonymous to deterministic

Real-Time

Several kind of real-time systems:

- RTS with strict constraints (hard real-time systems): deadline miss equal human life lost / mission failure (avionic industry)
- RTS with relative constraints(soft real-time systems): deadline misses are tolerated (multimedia)
- RTS with mixed constraints

Real-Time

Several kind of real-time systems:

- RTS with strict constraints (hard real-time systems): deadline miss equal human life lost / mission failure (avionic industry)
- RTS with relative constraints(soft real-time systems): deadline misses are tolerated (multimedia)
- RTS with mixed constraints

Example

Standard DO-178B developed for the avionic industry in USA distinguish 5 criticality levels, e.g:

- Safety Critical: failure = human lost (e.g. engines control, automatic pilot)
- Mission Critical: navigation systems, ...

Scheduling

- Scheduling algorithm: the algorithm used to decide which task is executed when
- Schedule: the result of the scheduling algorithm (a sequence of task)
- Scheduler: the task responsible to apply the scheduling algorithm to produce the schedule

- two families: preemptive, non preemptive
- two methodologies: offline / online

Scheduling

- Scheduling algorithm: the algorithm used to decide which task is executed when
- Schedule: the result of the scheduling algorithm (a sequence of task)
- Scheduler: the task responsible to apply the scheduling algorithm to produce the schedule

- two families: preemptive, non preemptive
- two methodologies: offline / online

In this class, we will study online preemptive algorithms

Periodic Model

Real-Time: ensure constraints respect. Which constraints ?

The most studied model (from the control command field): periodic task systems

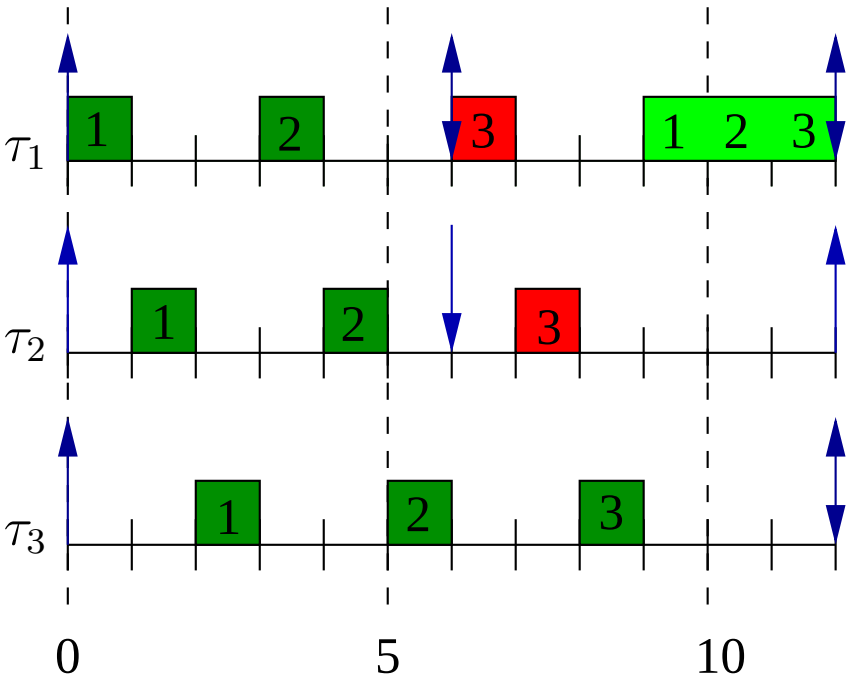
A periodic task τ_i is defined by:

- its first release time instant: r_i
- its worst case execution time (WCET): C_i
- its period: T_i
- its relative deadline: D_i
- from which we can deduce the absolute deadline of its instance k: $d_{i,k}$
- the logic is: upper cases for durations, lower cases for instants
- ... (extensible model !)

Example

	C_i	T_i	D_i
τ_1	3	6	6
τ_2	3	12	6
τ_3	3	12	12

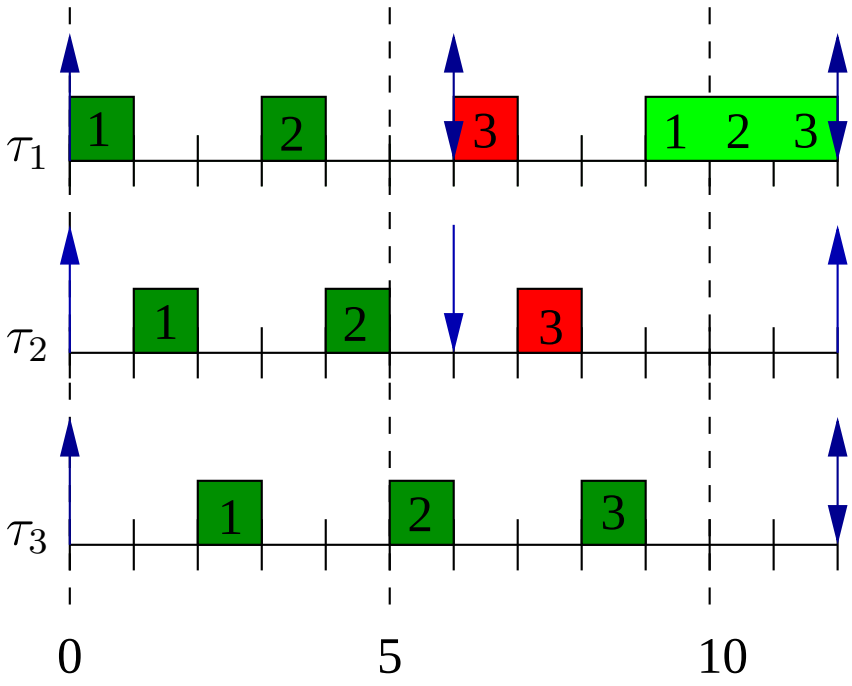
Round Robin



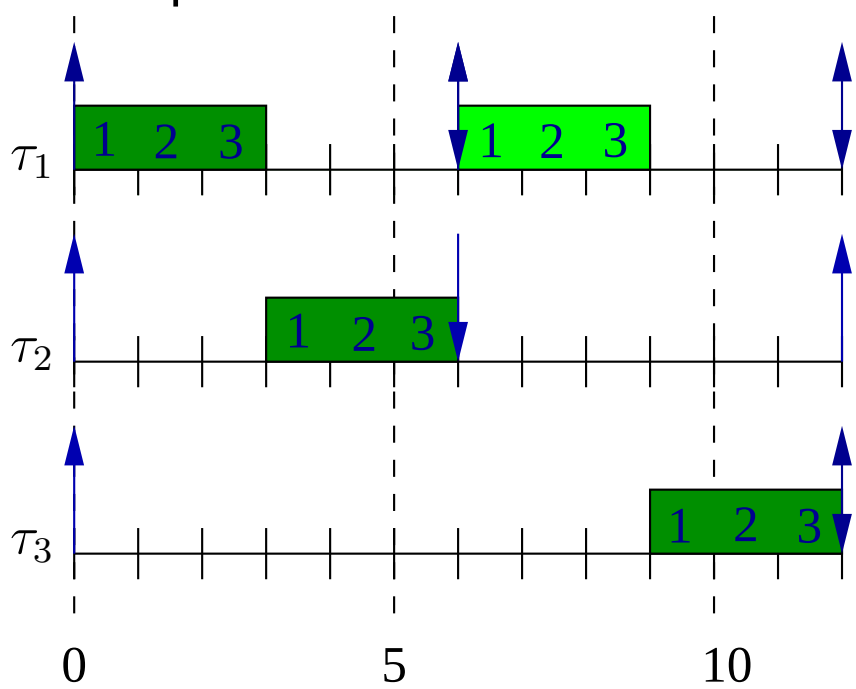
Example

	C_i	T_i	D_i
τ_1	3	6	6
τ_2	3	12	6
τ_3	3	12	12

Round Robin



fixed priorities



Online preemptive real-time scheduling

(Mono Processor)

Tasks instances (job) are sorted by priorities. At each time instant, the scheduler gives CPU to the task with the highest priority.

- fixed priority: tasks priority are fixed once and for all (according to a constant like period, deadline, importance... or arbitrarily),
- dynamic priority: priorities can change, they are given according to a variable like the next deadline proximity, the system laxity...

Evaluating a scheduling algorithm ??

Online preemptive real-time scheduling

(Mono Processor)

Tasks instances (job) are sorted by priorities. At each time instant, the scheduler gives CPU to the task with the highest priority.

- fixed priority: tasks priority are fixed once and for all (according to a constant like period, deadline, importance... or arbitrarily),
- dynamic priority: priorities can change, they are given according to a variable like the next deadline proximity, the system laxity...

Evaluating a scheduling algorithm ??

- optimality
- schedulability bound
- easy or not to implement ?
- execution overhead
- jitter, stability, average response times...

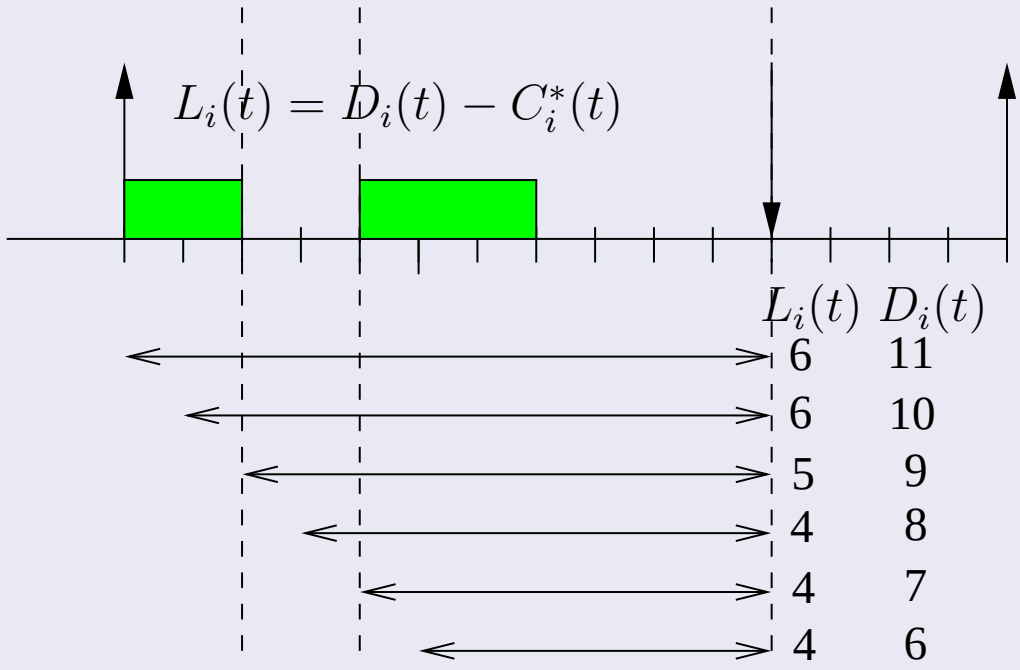
Main algorithms

- Rate Monotonic (RM): priority to the task with the smallest period
- Deadline Monotonic (DM) : priority to the task with the smallest relative deadline
- EDF : priority to the most urgent JOB (not task)
- LLF : priority to the task with the smallest laxity (variable function of time)

Main algorithms

- Rate Monotonic (RM): priority to the task with the smallest period
- Deadline Monotonic (DM) : priority to the task with the smallest relative deadline
- EDF : priority to the most urgent JOB (not task)
- LLF : priority to the task with the smallest laxity (variable function of time)

Laxity

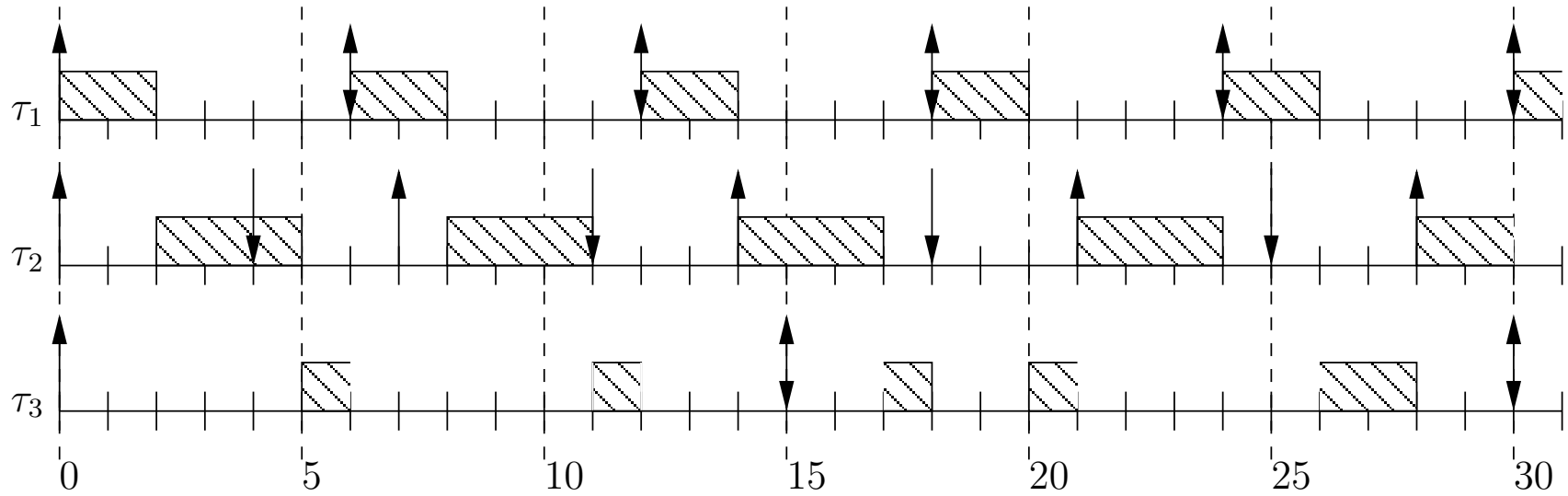


Exercises

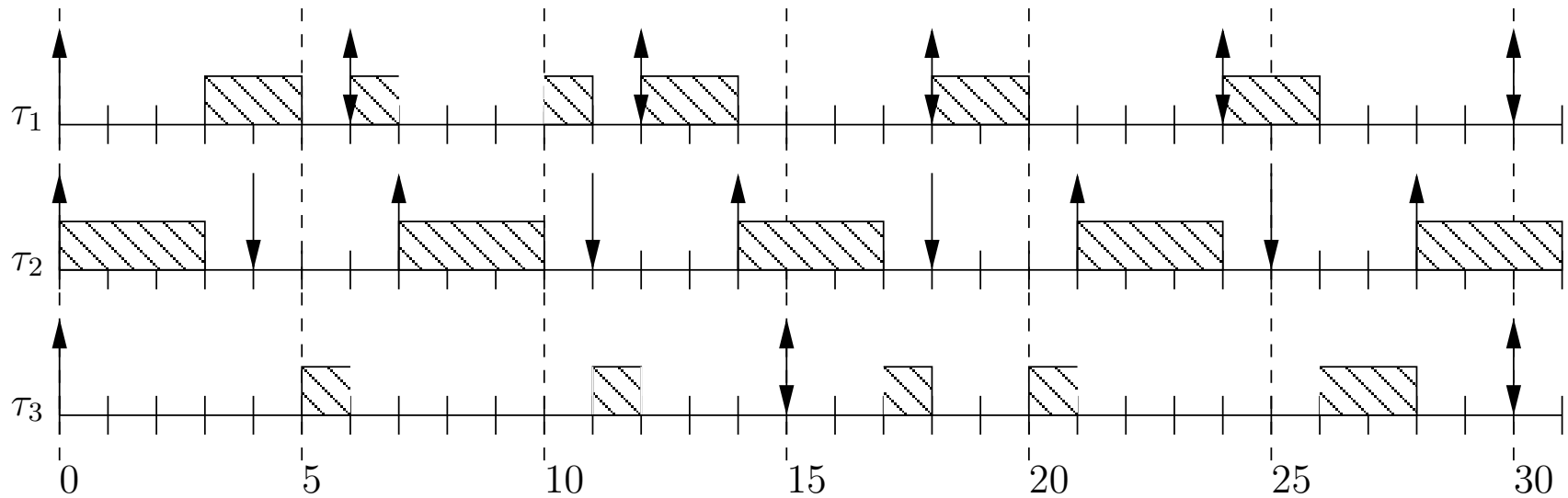
Give the schedules obtain with these 4 algorithms between $t = 0$ et $t = 30$ for taskset:

	r_i	T_i	D_i	C_i
τ_1	0	6	6	2
τ_2	0	7	4	3
τ_3	0	15	15	3

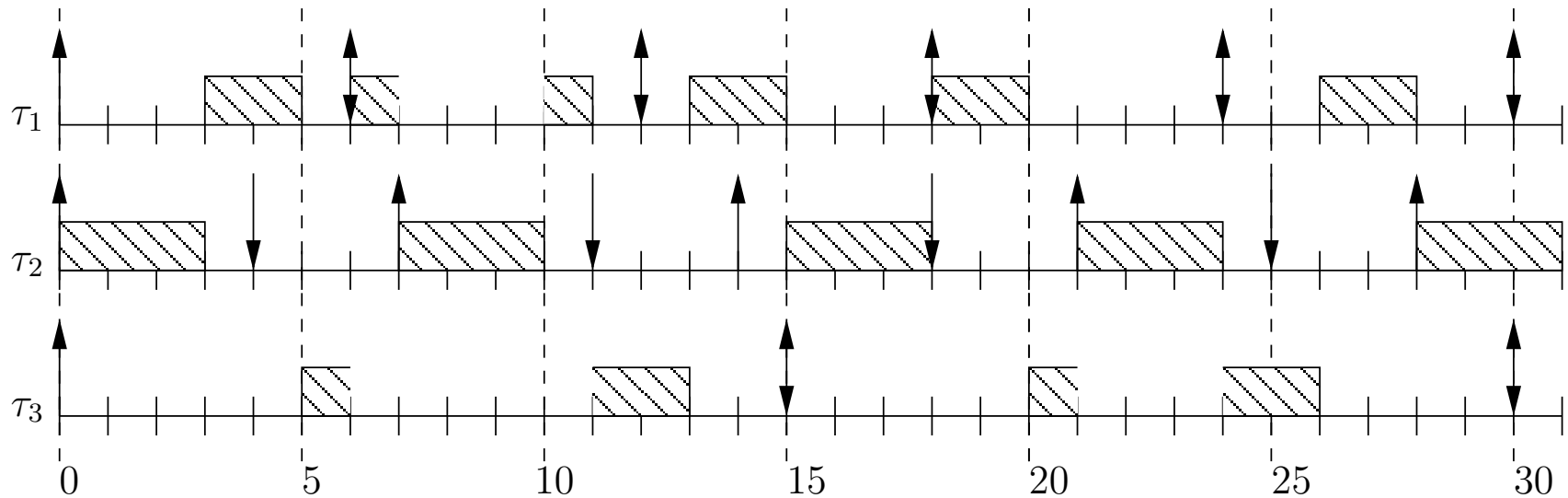
RM



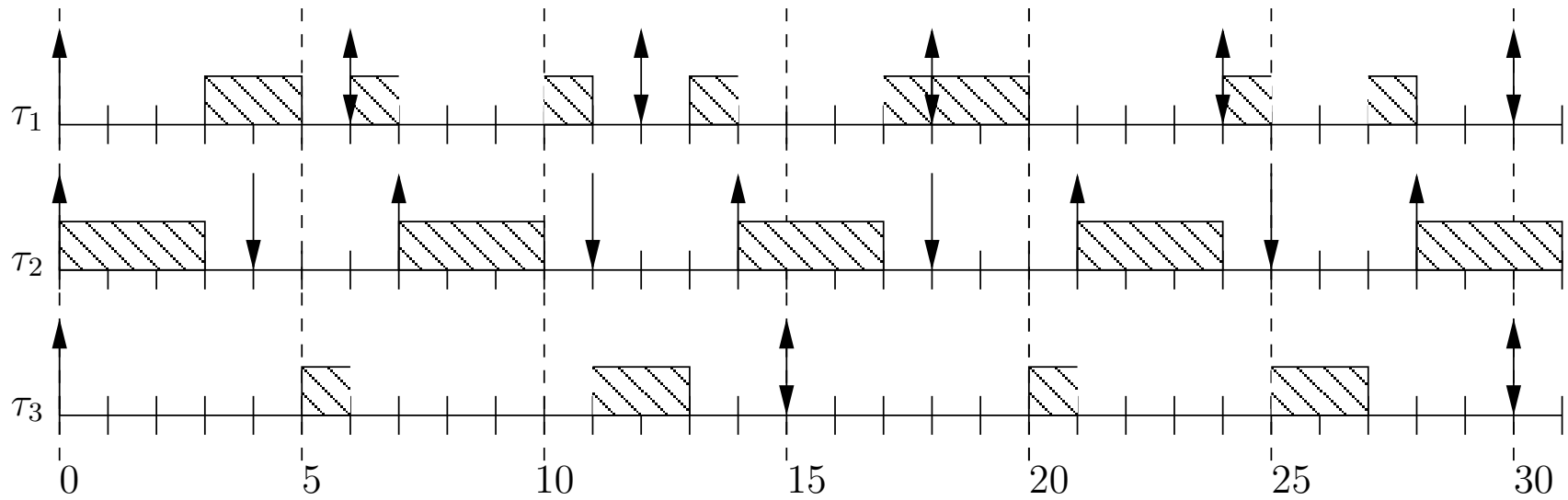
DM



EDF



LLF



Schedulability (validity)

$\neq$ Feasibility

- **Feasibility:** given a taskset, is it possible to propose a schedule that respect all timing constraints?
- **Schedulability:** given a taskset, is it possible to propose a deterministic algorithm that generates a valid schedule?
- **Schedulability with A:** given a taskset and an algorithm A, is A produces a valid schedule?

Several approaches depending on the studied problem and the system criticality:

- sufficient condition for an admission control scheme,
- fault and/or overload detection,
- exact analysis with feasibility/schedulability analysis theory.

System Load study

Processor load: $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$

This can be enough to conclude under certain assumptions:

- $U \leq n(2^{\frac{1}{n}} - 1)$ is sufficient (but not necessary) condition for the schedulability iff $\forall_i D_i = T_i$ (implicit deadlines) with a fixed priority algorithm,
- $U \leq 1$ is a necessary and sufficient condition for the schedulability under EDF iff $\forall_i D_i = T_i$,
- ...

But if $D_i \leq T_i$ (constrained deadlines), or when D_i not related to T_i (general case) things are not so simple...

Schedulability study when $D_i = T_i$

(fixed priority)

- $U \leq n(2^{\frac{1}{n}} - 1)$ is a sufficient condition and $U \leq 1$ is a necessary condition
- when $1 \leq U > n(2^{\frac{1}{n}} - 1)$?
  - workload study: we are looking for a time instant before the deadline where all the cumulated demand is satisfied
  - worst case response time computation
    - job j response time R_i^j: time between the request and the end of the job
    - task worst case response time ($WCRT_i$): maximum amongst the R_i^j for all j

Demand study

If $\forall i, D_i \leq T_i$, the system is schedulable with a fixed priority algorithm iff it exists a time instant t in the interval $(0, Di]$ such that $t = w_i(t)$

- with $w_i(t) = \sum_{k \leq i} \left\lceil \frac{t}{T_k} \right\rceil C_k$
- recursive algorithm: computation of $t_1 = w_i(0)$, then $t_2 = w_i(t_1)$, ..., $t_n = w_i(t_{n-1})$
- the algorithms ends either when the deadline is reached or if a t with $t = w_i(t)$ is found.

Demand study

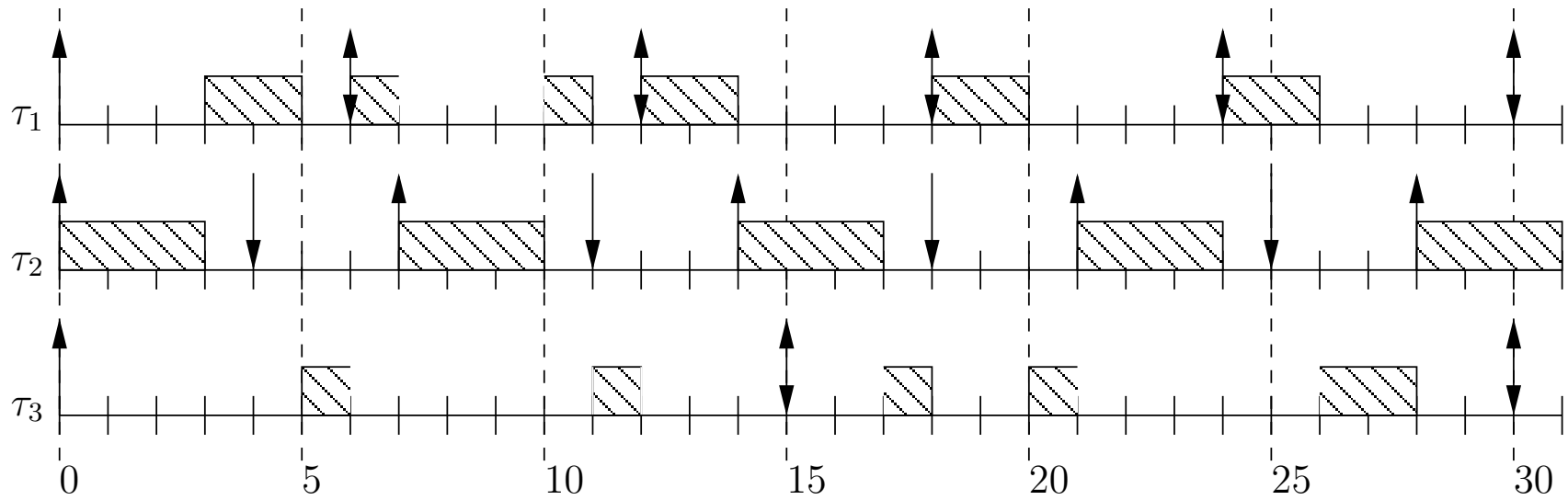
If $\forall i, D_i \leq T_i$, the system is schedulable with a fixed priority algorithm iff it exists a time instant t in the interval $(0, Di]$ such that $t = w_i(t)$

- with $w_i(t) = \sum_{k \leq i} \left\lceil \frac{t}{T_k} \right\rceil C_k$
- recursive algorithm: computation of $t_1 = w_i(0)$, then $t_2 = w_i(t_1)$, ..., $t_n = w_i(t_{n-1})$
- the algorithms ends either when the deadline is reached or if a t with $t = w_i(t)$ is found.

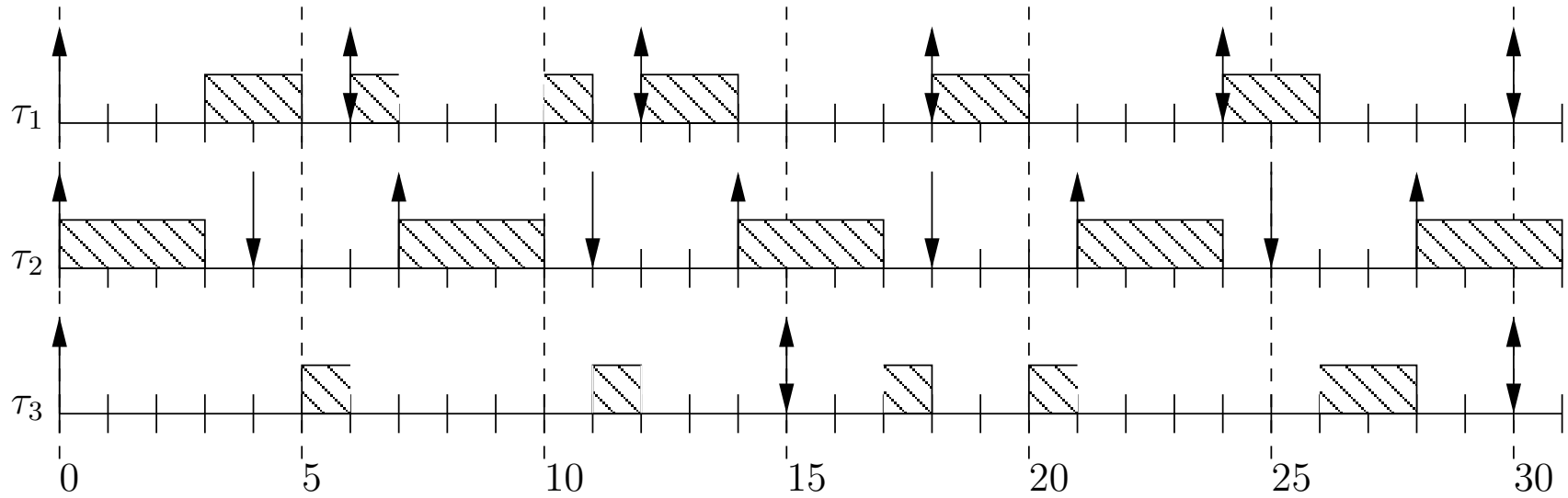
Exercises

- study the level 2 demand on the preceding example with RM
- study the level 2 demand on the preceding example with DM
- study the level 3 demand on the preceding example with RM
- is the level 3 demand with *DM* different ?

DM



DM



t	8	13	15
$w_3(t)$	13	15	18

Limitations

- this test permits only to conclude on the schedulability, it does not provide any other informations. It can be of interest to compute the response times, allowing the system designer to have a better idea of the tasks behaviors (jitter, average response time...).
- it works only for tasks with implicit deadlines $D_i \leq T_i$
- to convince yourself, try to analyze this example:

	r_i	C_i	T_i	D_i	Priority P_i
τ_1	0	4	8	10	high
τ_2	0	3	6	8	low

  What is the value of $w_2(7)$? Is it relevant to compare this to D_2 ?
- anyway this system is not schedulable, but one has to wait until time $t = 21$ to see that the second instance of task τ_2 misses its deadline (the worst response time is no longer the one of the first job!)

Computing response times

- recursive computation very similar to the demand analysis
- a task may be delayed only by tasks with an higher priority
- we will compute the response time of task τ_i 's job number j, job 1 being the one starting at r_i. Its termination instant, denoted F_i^j, is given by equation:

$$F_i^j = \min_{t>0}\{t = w_{i-1}(t) + j * C_i\} \tag{1}$$

- its response time, R_i^j, is then the difference between its termination instant and its release instant:

$$R_i^j = F_i^j - (r_i + (j-1)T_i) \tag{2}$$

Busy Period

processor continuous activity

- a level i busy period is the time interval between two level-i idle time: the processor is idle or occupied with lower priorities Level-i idle times are solution of the equation $w_i(t) = t$
- we want to compute the duration of the one starting at time $t = 0$, because the worst scenario for task τ_i must occur during it (assuming a synchronous activation scenario). The algorithm is the same as the one used for the demand analysis excepted that we do not stop when the deadline is reached.

It is sufficient to study a task during the first busy period to encounter its worst case response time.

Busy Period

processor continuous activity

- a level i busy period is the time interval between two level-i idle time: the processor is idle or occupied with lower priorities Level-i idle times are solution of the equation $w_i(t) = t$
- we want to compute the duration of the one starting at time $t = 0$, because the worst scenario for task τ_i must occur during it (assuming a synchronous activation scenario). The algorithm is the same as the one used for the demand analysis excepted that we do not stop when the deadline is reached.

It is sufficient to study a task during the first busy period to encounter its worst case response time.

Exercise

- compute the level 2 busy period (bp_2) for the previous example
- compute Q_i, the τ_2 activations number during bp_2
- compute the Q_i first response times of τ_2

Other Task Models

- until now, we made the hypothesis that tasks were independants, but other constraints can exists:
  - precedence constraints between tasks
  - resource sharing with mutual exclusion
- non periodic task have to be handle:
  - by setting a bound on their interarrival time, and worst case study (sporadis model)
  - encapsulate their handling inside a server with limited ressources
  - handling in backgroud or with a slack stealing algorithm

ressource sharing

- when two concurrent task access ressources, we have to protect the resources accesses with lock (semaphore, mutex, ...)
- to access a ressource, a task have to obtain the associated lock
- one task at once ca have a given lock
- when asking a lock, a task is blocked until the lock is available
- special attention must be given to the lock attribution algorithm

Issues and solution with fixe priorities

- Bounding the priority inversions
- Avoid deadlocks
- Prevent blocking chains

Issues and solution with fixe priorities

- Bounding the priority inversions

when a task execute whereas another one with an higher priority is blocked

- Avoid deadlocks
- Prevent blocking chains

Issues and solution with fixe priorities

- Bounding the priority inversions
- Avoid deadlocks

when a task has a first lock, and ask for another one previously given to a second task, which waits for the first lock

- Prevent blocking chains

Issues and solution with fixe priorities

- Bounding the priority inversions
- Avoid deadlocks
- Prevent blocking chains

When a task instance is blocked several times

Issues and solution with fixe priorities

- Bounding the priority inversions
- Avoid deadlocks
- Prevent blocking chains

Three algorithms :

- Priority Inheritance Protocol (PIP)
- Priority Ceiling Protocol (PCP)
- Priority Ceiling Emulation (PCE)

Issues and solution with fixe priorities

- Bounding the priority inversions
- Avoid deadlocks
- Prevent blocking chains

Three algorithms :

- Priority Inheritance Protocol (PIP)
- Priority Ceiling Protocol (PCP)
- Priority Ceiling Emulation (PCE)

Issues and solution with fixe priorities

- Bounding the priority inversions
- Avoid deadlocks
- Prevent blocking chains

Three algorithms :

- Priority Inheritance Protocol (PIP)
- Priority Ceiling Protocol (PCP)
- Priority Ceiling Emulation (PCE)

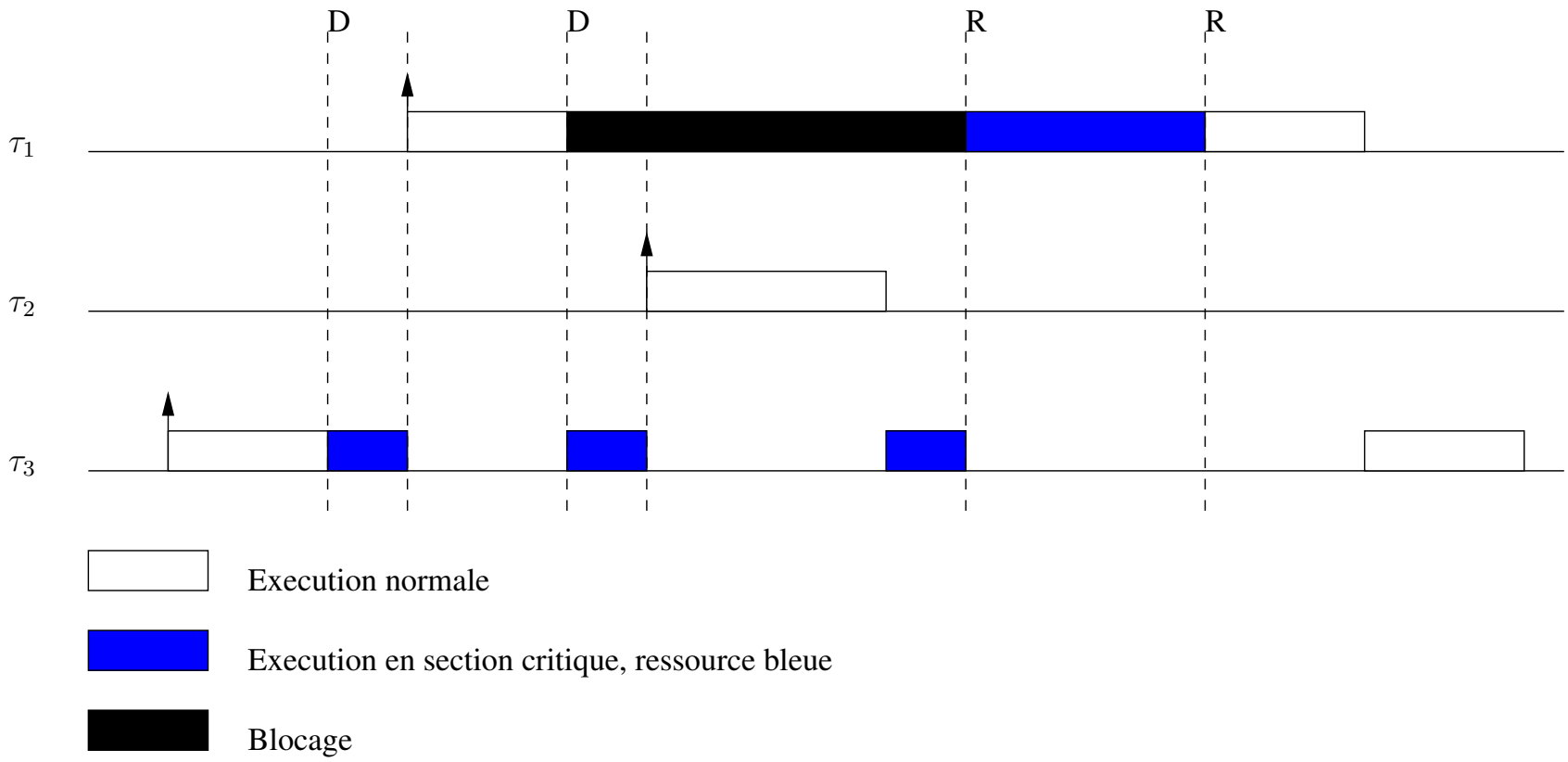
Issues and solution with fixe priorities

- Bounding the priority inversions
- Avoid deadlocks
- Prevent blocking chains

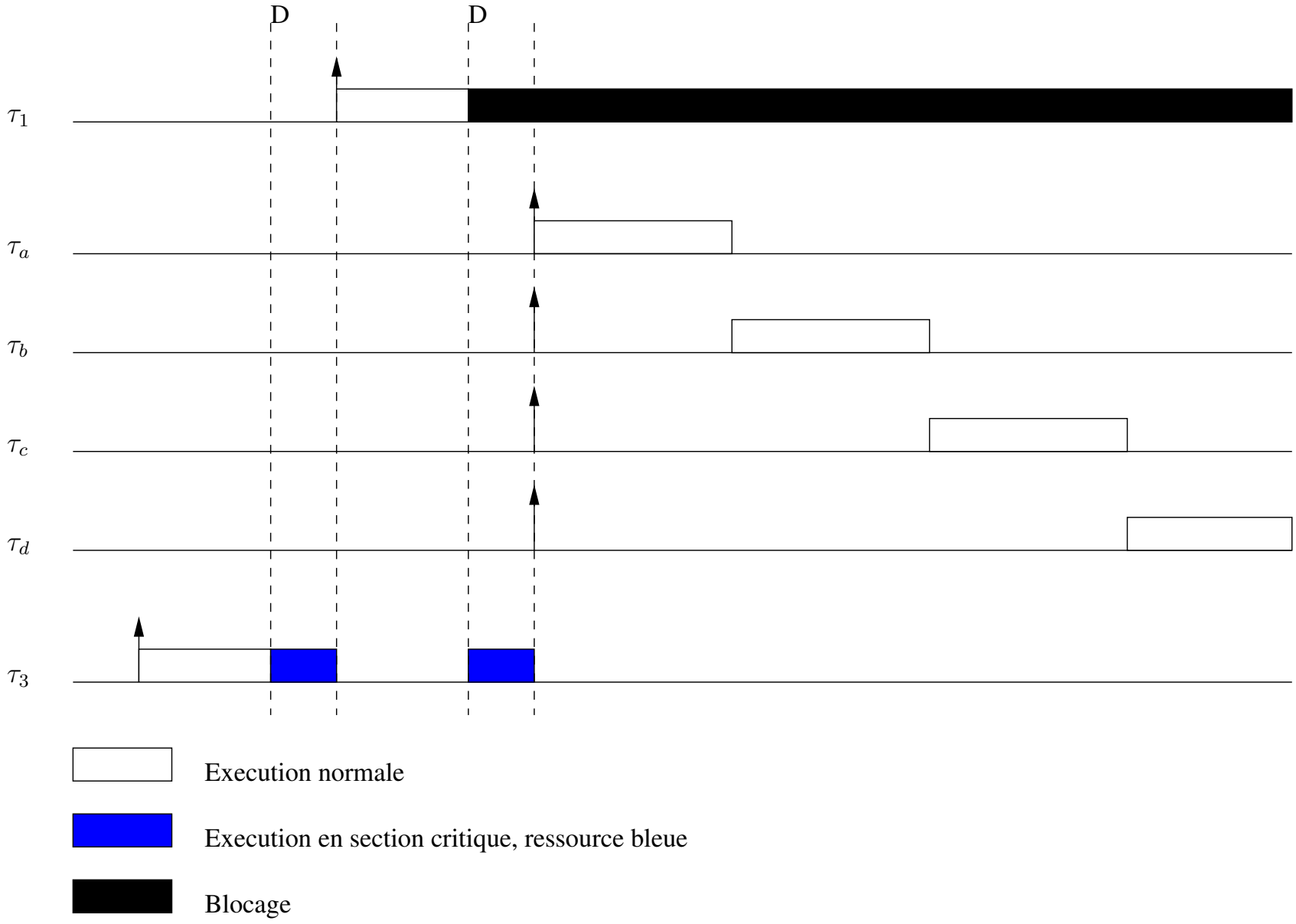
Three algorithms :

- Priority Inheritance Protocol (PIP)
- Priority Ceiling Protocol (PCP)
- Priority Ceiling Emulation (PCE)

Unbound Inversion



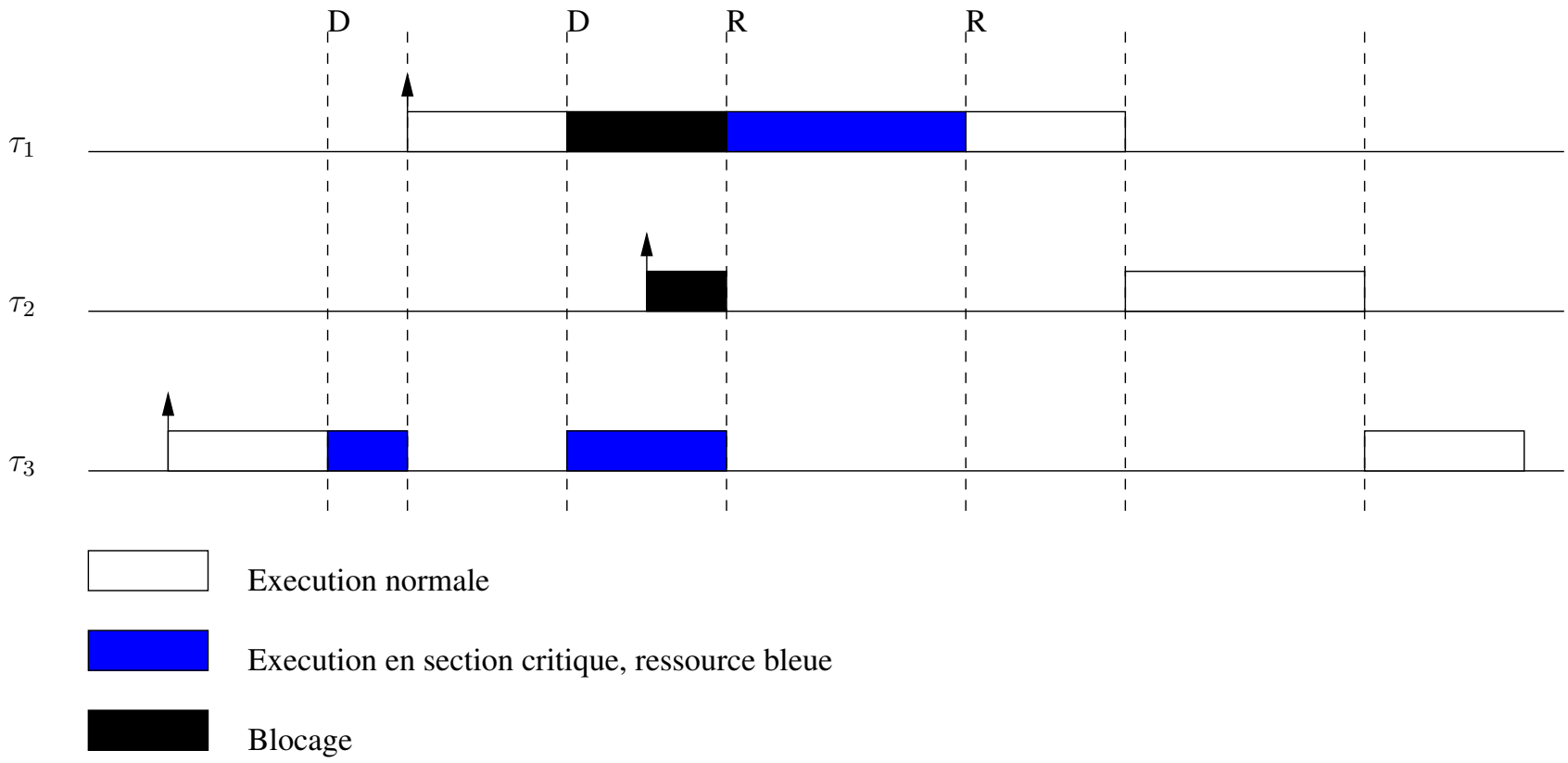
Unbound Inversion



Priority Inheritance Protocol (PIP)

- when τ_1 asks for a ressource used by a lower priority task τ_2, τ_1 is blocked and τ_2 inherits τ_1 priority
- the inheritance is transitive, when τ_3 blocks τ_2 and τ_2 blocks τ_1, then τ_3 inherits τ_1 priority from τ_2 (with $P_1 > P_2 > P_3$)
- when τ_2 free the ressource, it goes back to its initial priority
- when a ressource is free, it is allocated to the task with the highest priority (amongs ones waiting for it)

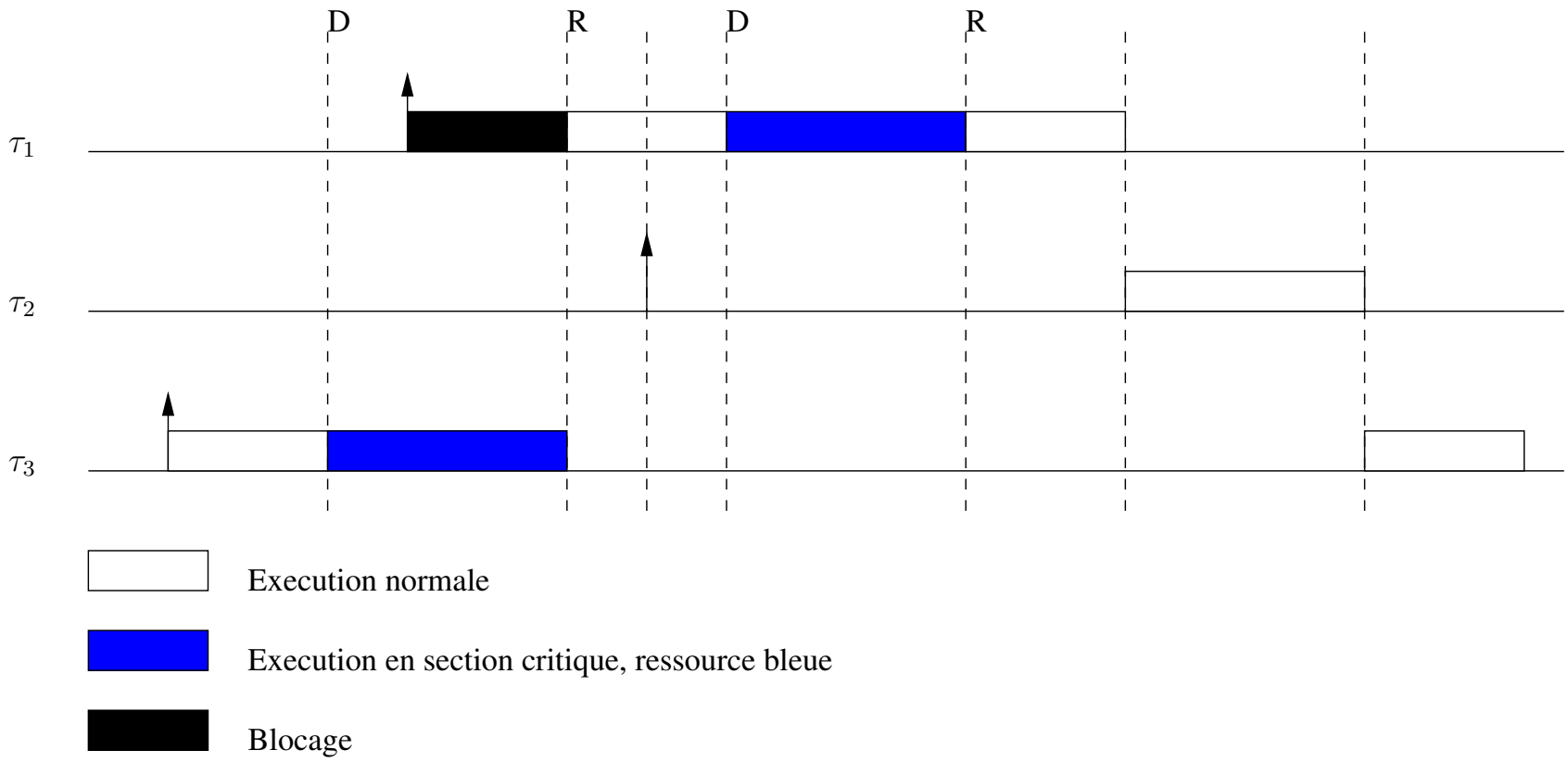
Priority Inheritance Protocol (PIP)



Priority Ceiling Emulation (PCE)

- a ceiling priority is statically computed for each ressources
- when a task enters a critical section, it takes the ceiling priority of the ressource

Priority Ceiling Emulation (PCE)

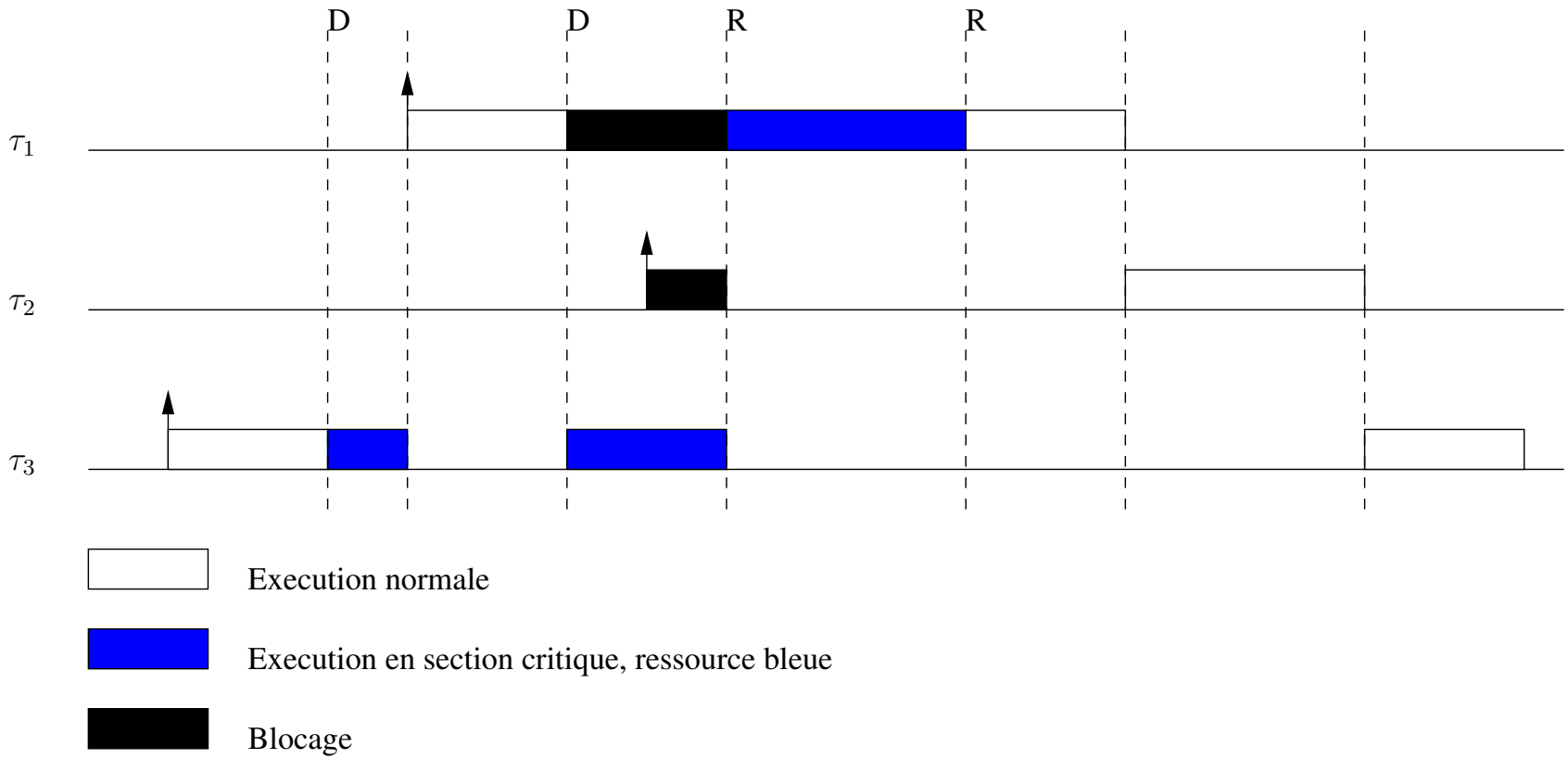


Priority Ceiling Protocol (PCP)

- a ceiling priority is statically computed for each ressources
- a task can enter into a critical section iff its priority is greater than all the ceiling priorities of currently used ressources
- as with PIP, there is priority inheritance
- when a task is already into a critical section, it can obtain other lock without verifying the preceeding condition

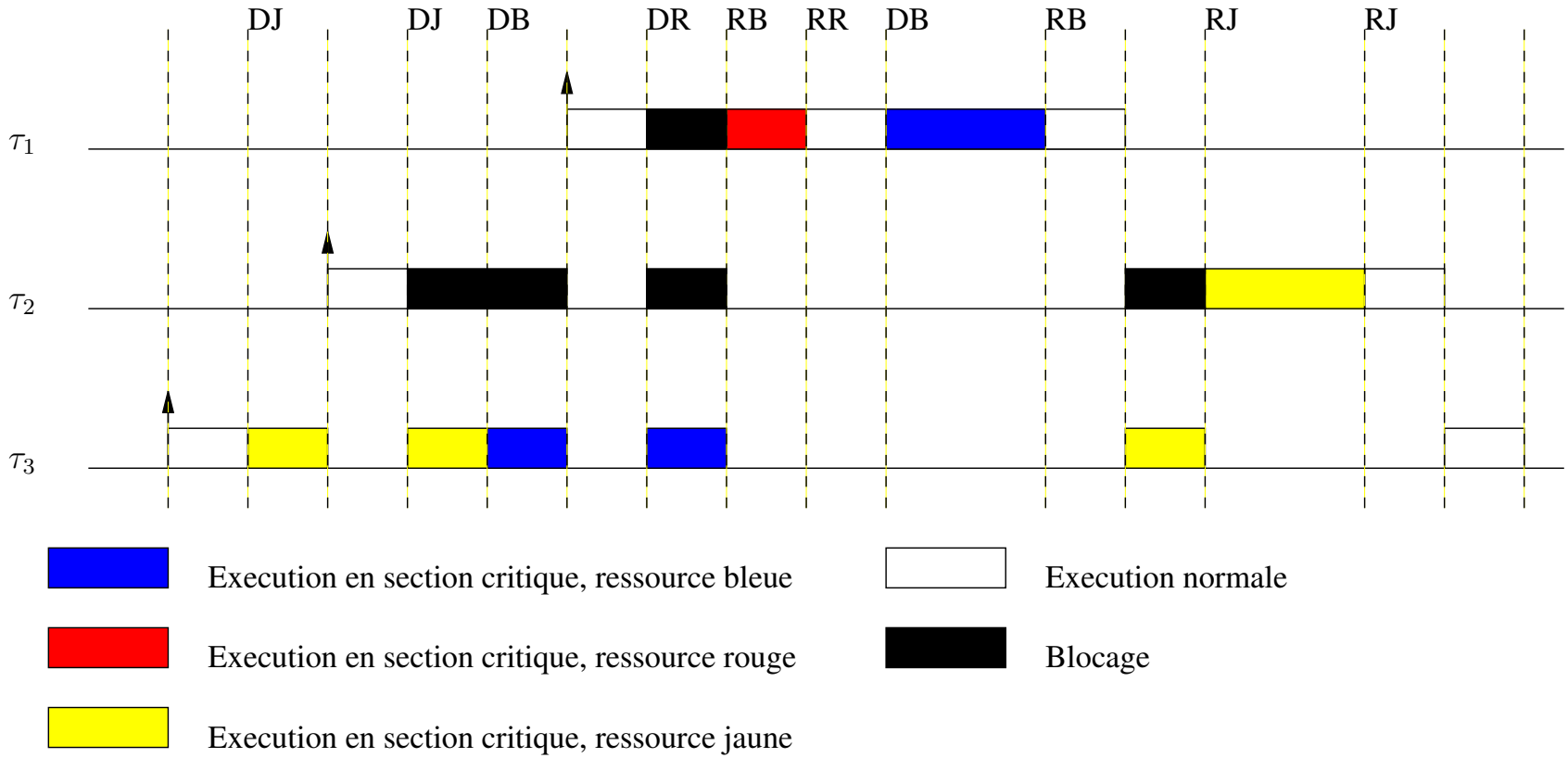
Priority Ceiling Protocol (PCP)

Example 1



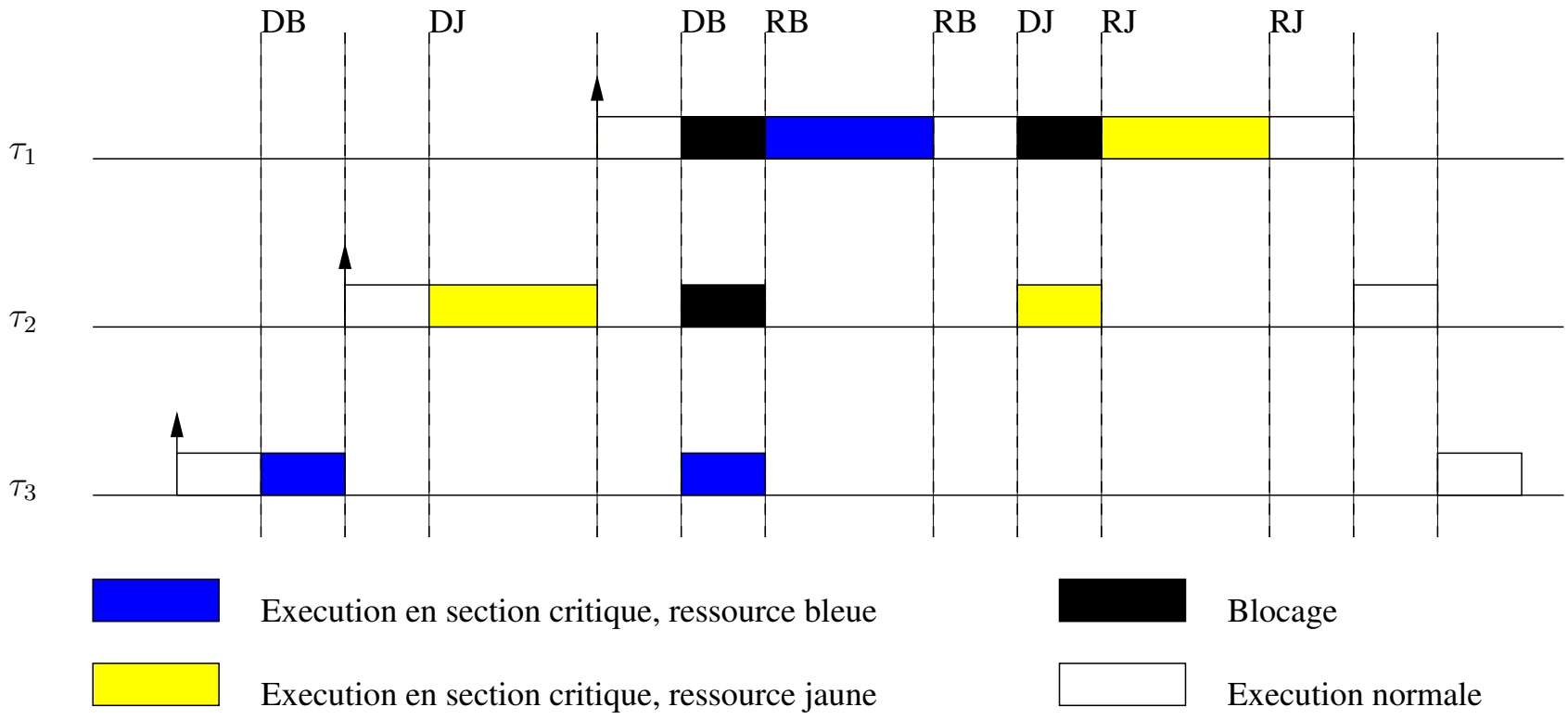
Priority Ceiling Protocol (PCP)

Example 2



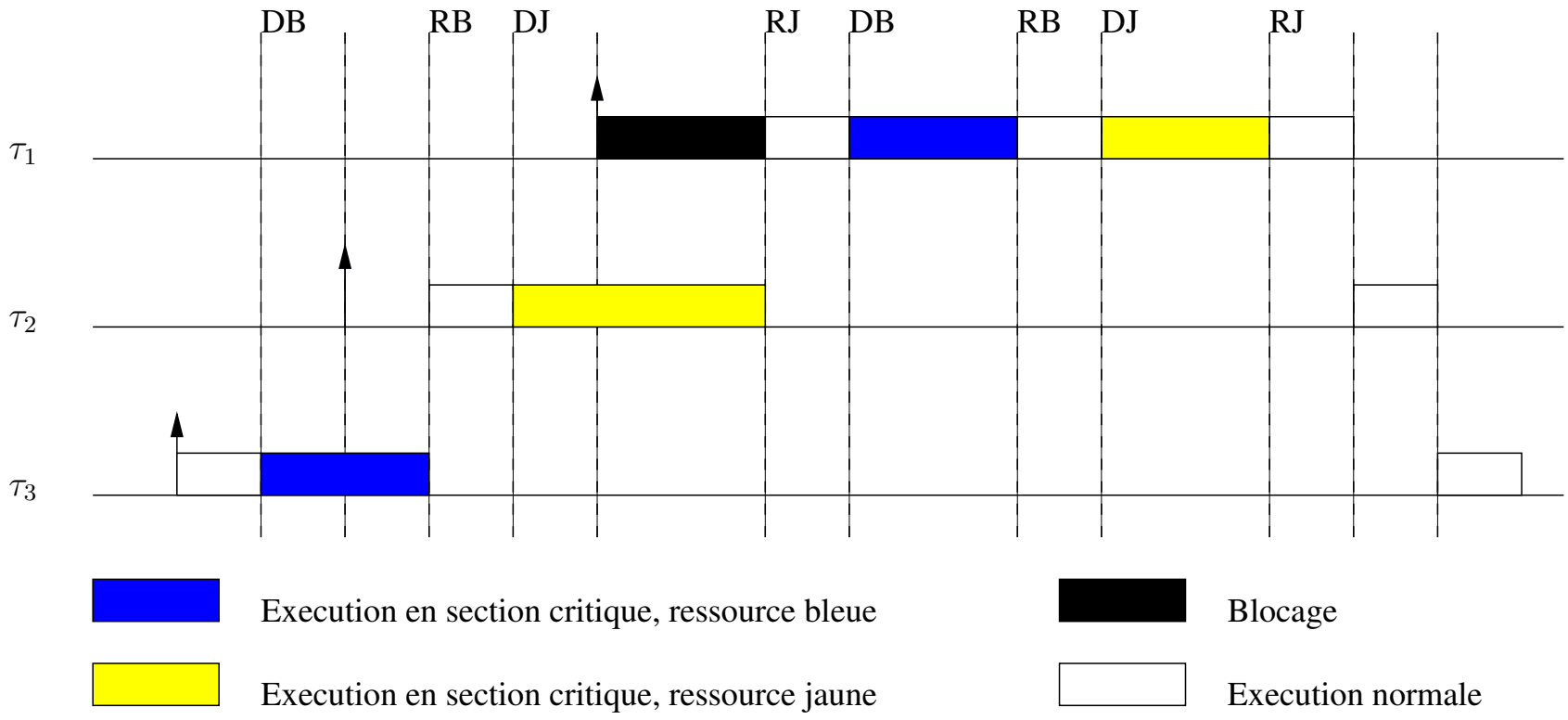
Chaîne de blocage

PIP



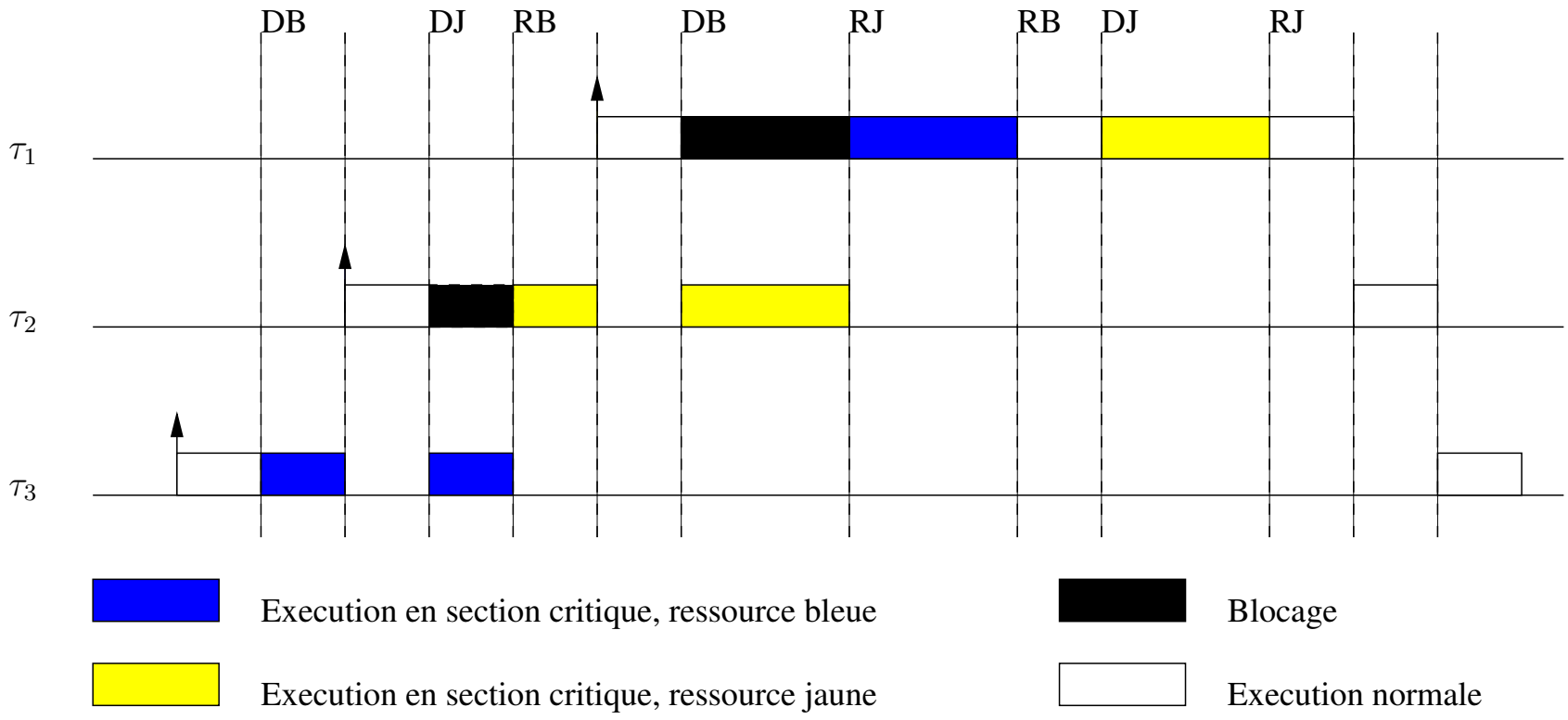
(Pas de) Chaîne de blocage

PCE



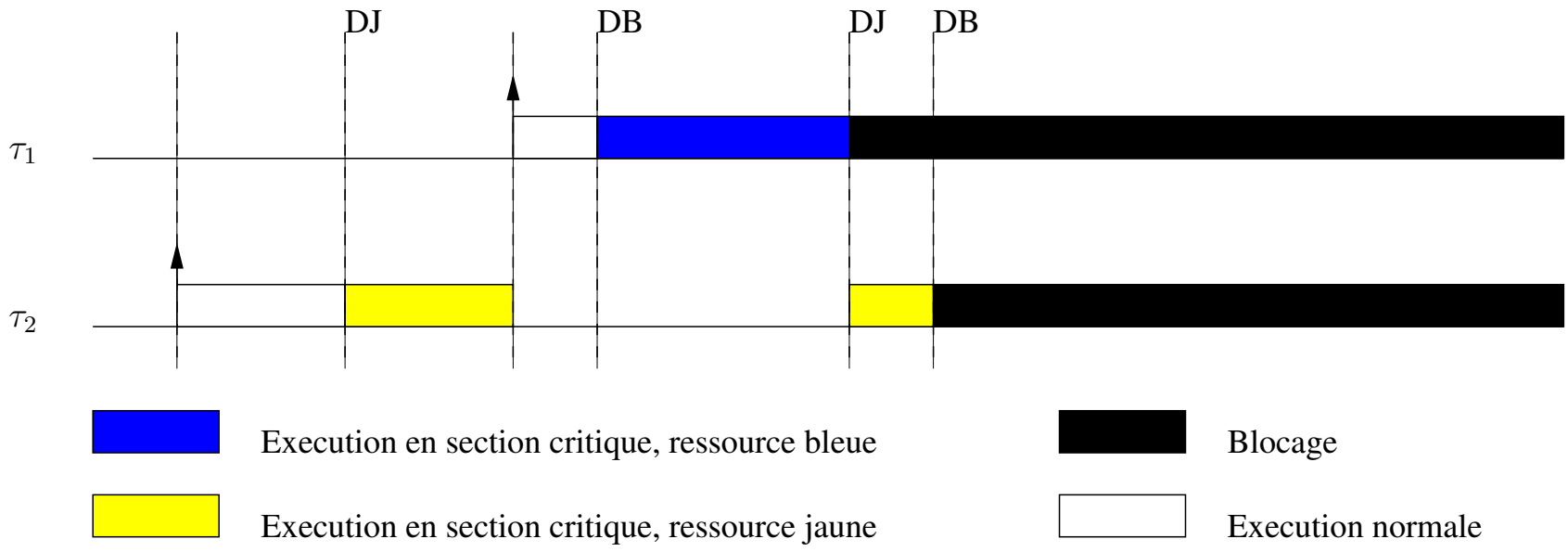
(Pas de) Chaîne de blocage

PCP



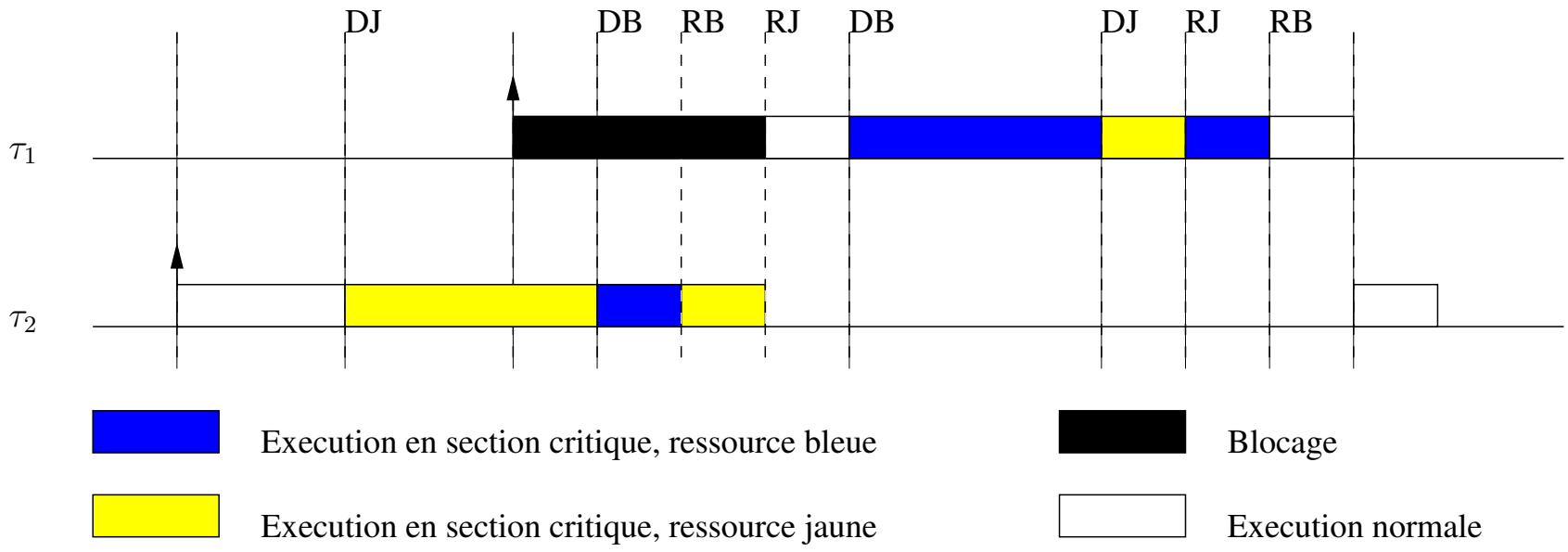
Interblocage

PIP



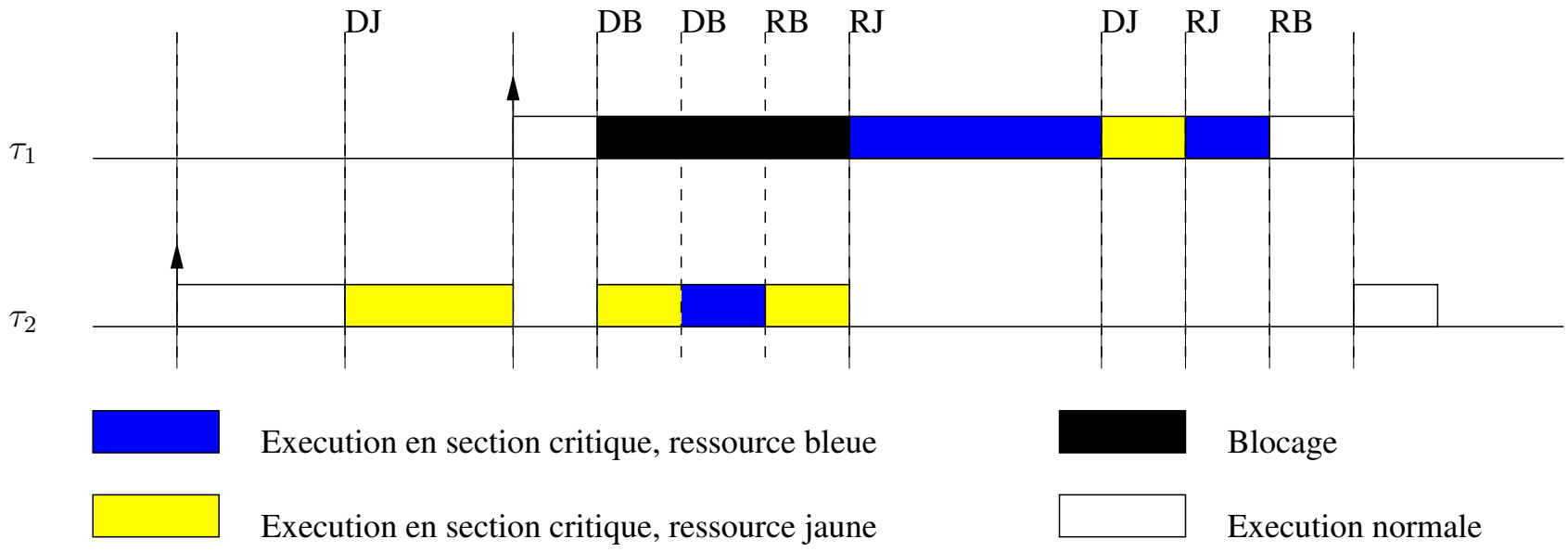
(Pas d')Interblocage

PCE



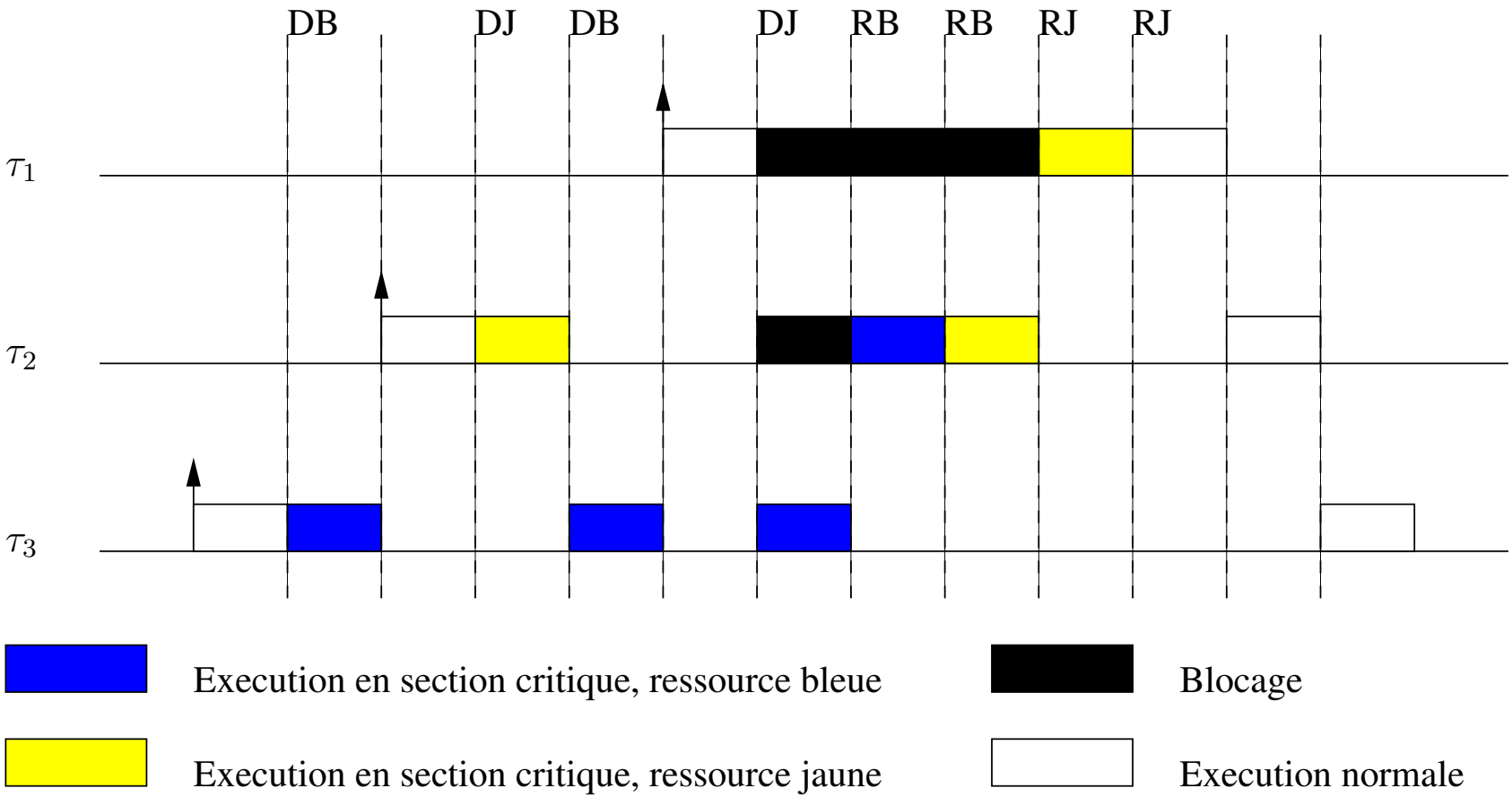
(Pas d')Interblocage

PCP



PIP & PCP : Héritage transitif

PIP



Aperiodic Model

Mix Scheduling Periodic/aperiodic

1. ensure deadlines respect for the periodic traffic
2. minimize the response times for the aperiodic traffic

- Impossible to offer temporal warranty for an aperiodic task:
  - it can arrive at any instant
  - an unbounded number can arrive simultaneously at any instant

Schedule in background (BS)

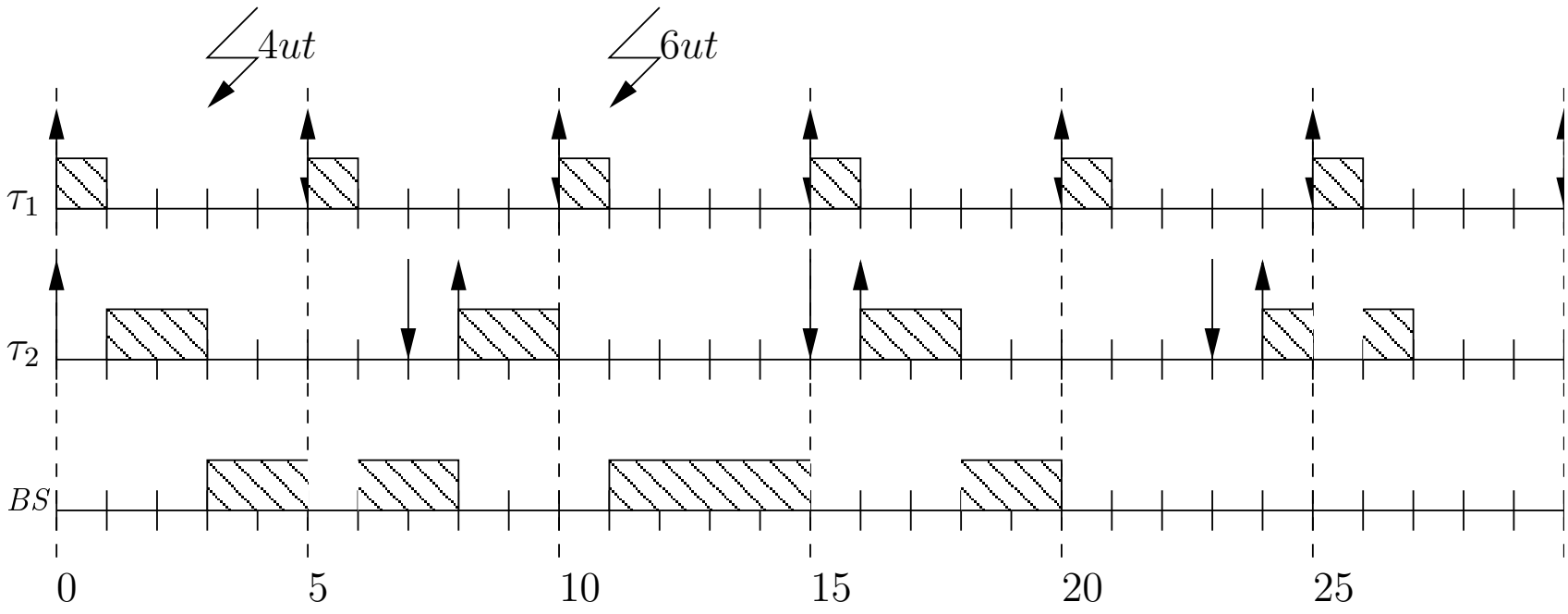
Background Scheduling

the lowest priorities are reserved for aperiodic tasks

- very simple to setup
- there is no interference with the periodic traffic (warning: no resource sharing between periodic and aperiodic is permitted)

- just solve the first point! nothing is done to minimizing the aperiodic response times

BS



Aperiodic-task server

Task server

- resource reservation
- bound the interference on other tasks

The aperiodic traffic is delegated to an other specific task with:

- a budget
- a policy to replenish this budget

Lot of algorithms available: PS, DS, SS, PE, EPE...

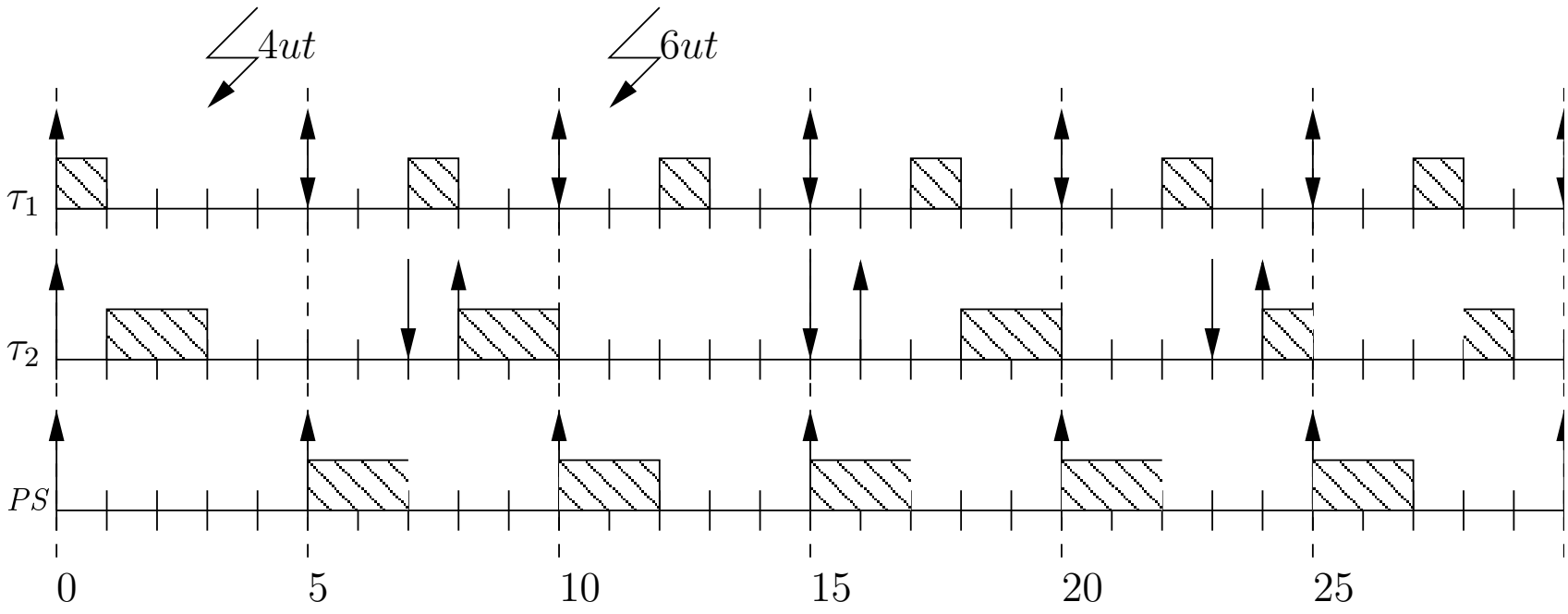
Polling Server (PS)

Ou *serveur à scrutation*

Il s'agit d'une tâche périodique et il s'analyse comme telle.

- $PS = \{(r_s, C_s, T_s)\}$
- éventuellement : $PS = \{(r_s, C_s, T_s, P_s)\}$
- les apériodiques sont ajoutées dans une file d'attente lors de leur activation,
- lorsque le serveur obtient le processeur, il exécute les tâches de la file dans la limite de sa capacité,
- la capacité revient au maximum périodiquement,
- si la file est vide alors que le serveur a la main, la capacité tombe à zéro.

Polling Server (PS)

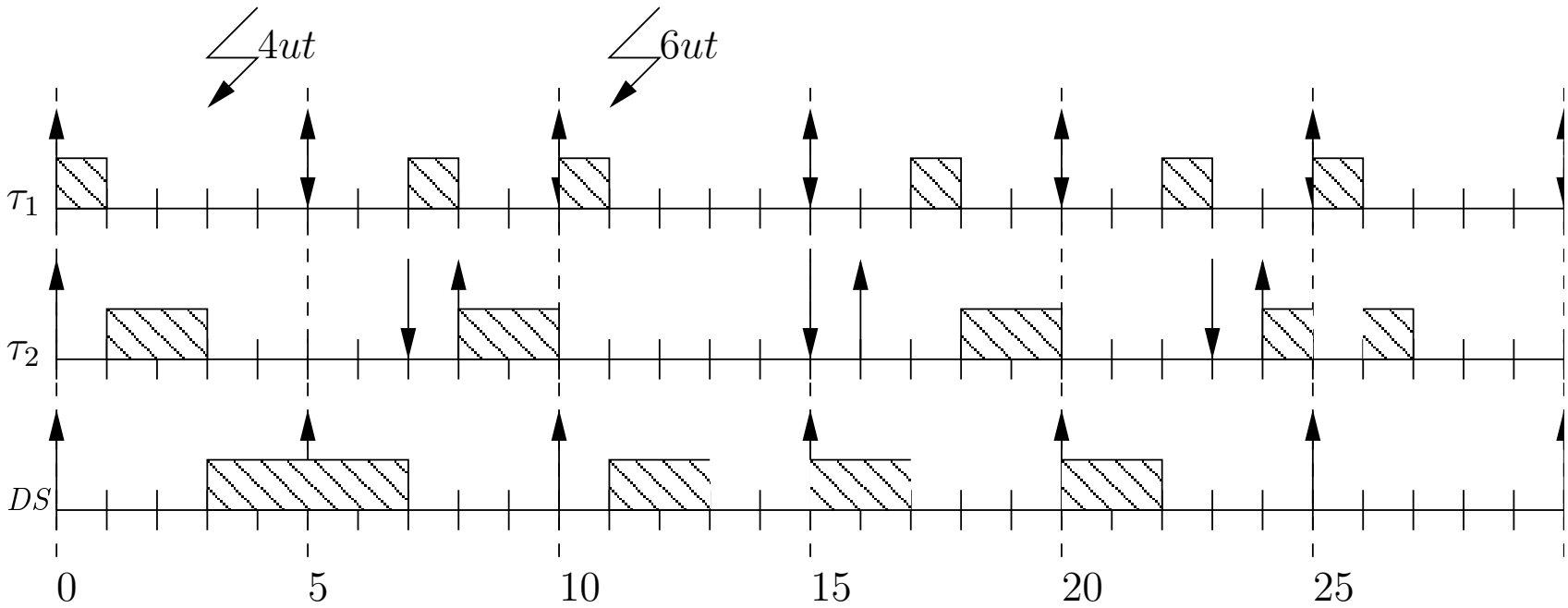


Deferrable Server (DS)

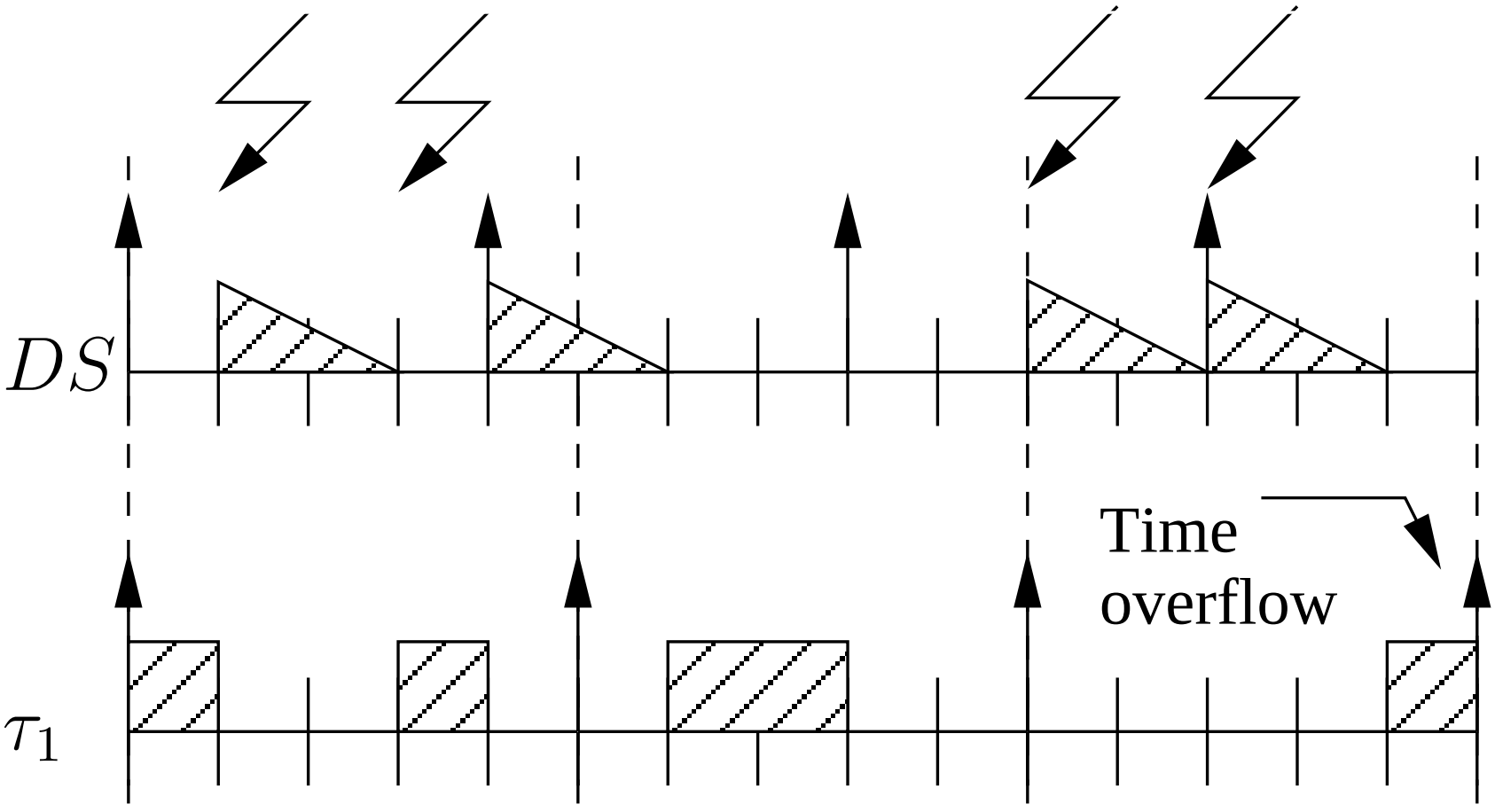
Ou *serveur ajournable*

- $DS = \{(r_s, C_s, T_s)\}$
- éventuellement : $DS = \{(r_s, C_s, T_s, P_s)\}$
- identique au PS, mais conserve sa capacité lorsque la file est vide,
- n'est plus une tâche périodique, et ne s'analyse plus comme telle.

Deferrable Server (DS)



Deferrable Server (DS)



DS: Schedulability analysis

- Sufficient condition on the load with Rate Monotonic :

$$U = U_s + \sum_{i=1}^{n} \frac{C_i}{T_i} \leq U_s + n \times \left(\left(\frac{U_s + 2}{2U_s + 1} \right)^{\frac{1}{n}} - 1 \right) \quad (3)$$

  with $U_s = \frac{C_s}{T_s}$ and n the number of periodic tasks (without the server).

- when n tends to infinity:

$$U \leq U_s + ln \left(\frac{U_s + 2}{2U_s + 1} \right) \quad (4)$$

- For the response time analysis, the worst case now is:
  - for periodic task with lesser priorities than the server, synchronous activation at $t = r_s + T_s - C_s$,
  - for the server, synchronous activation with higher priority tasks,
  - equivalent to feasibility analysis for tasks with activation jitters.

Slack stealing

Slack Stealing

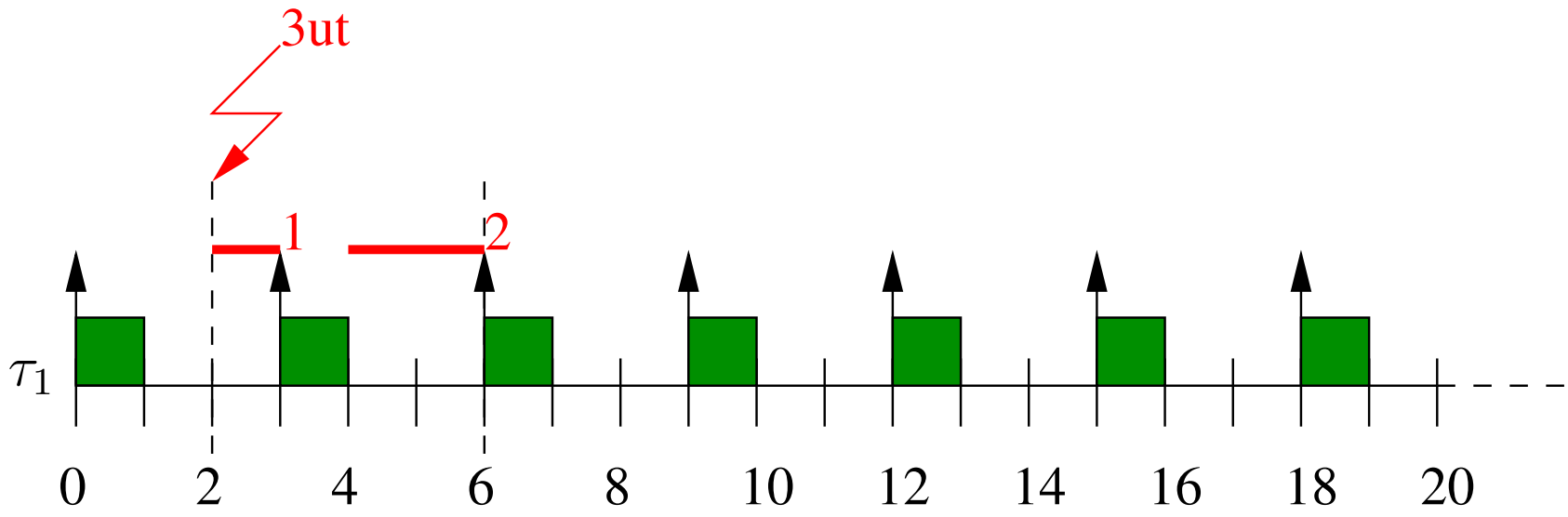
- no reservation
- dynamic computation of the system laxity (additional load the system can handler at time t)

- better performances (average random case)
- but no more reservation (no more “warranty”)

Laxity

Definitions

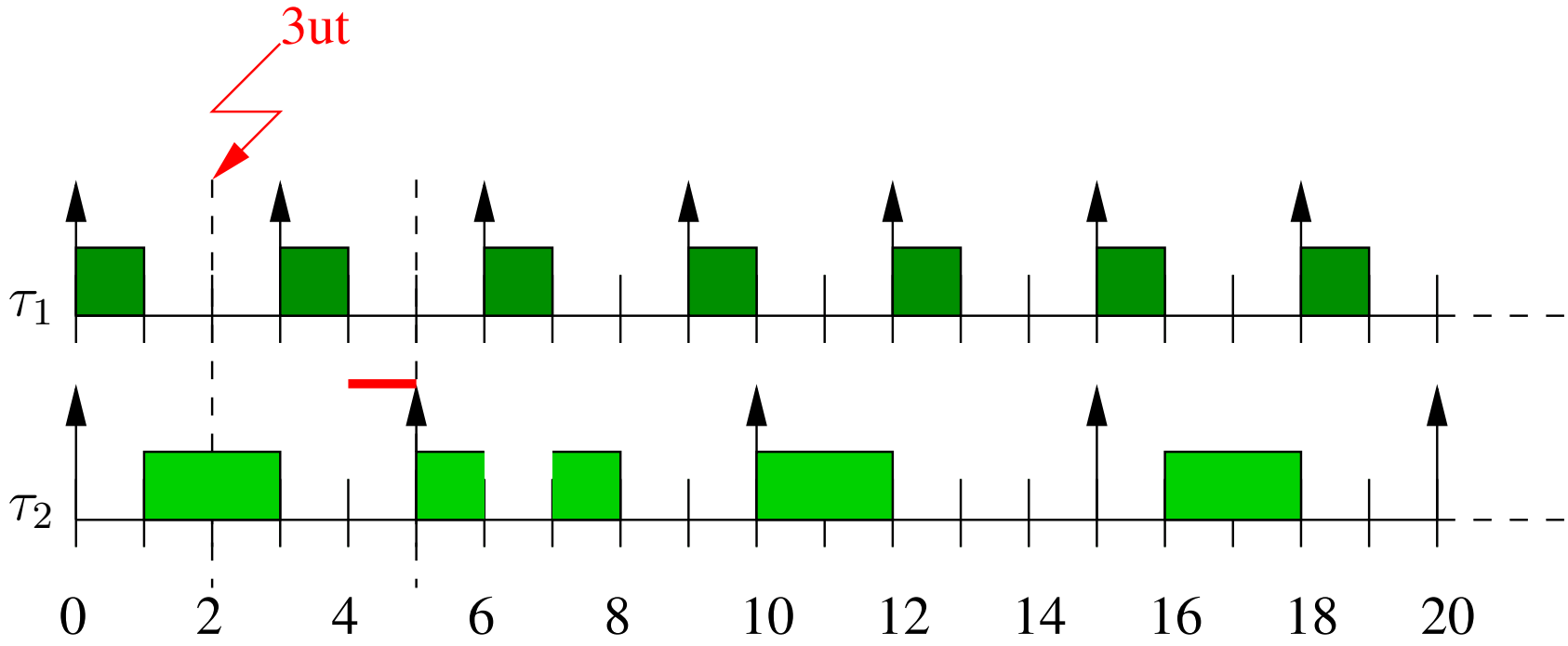
- $S_i(t)$: additional possible work at priority levels $\geq i$ until next τ_i deadline



Laxity

Definitions

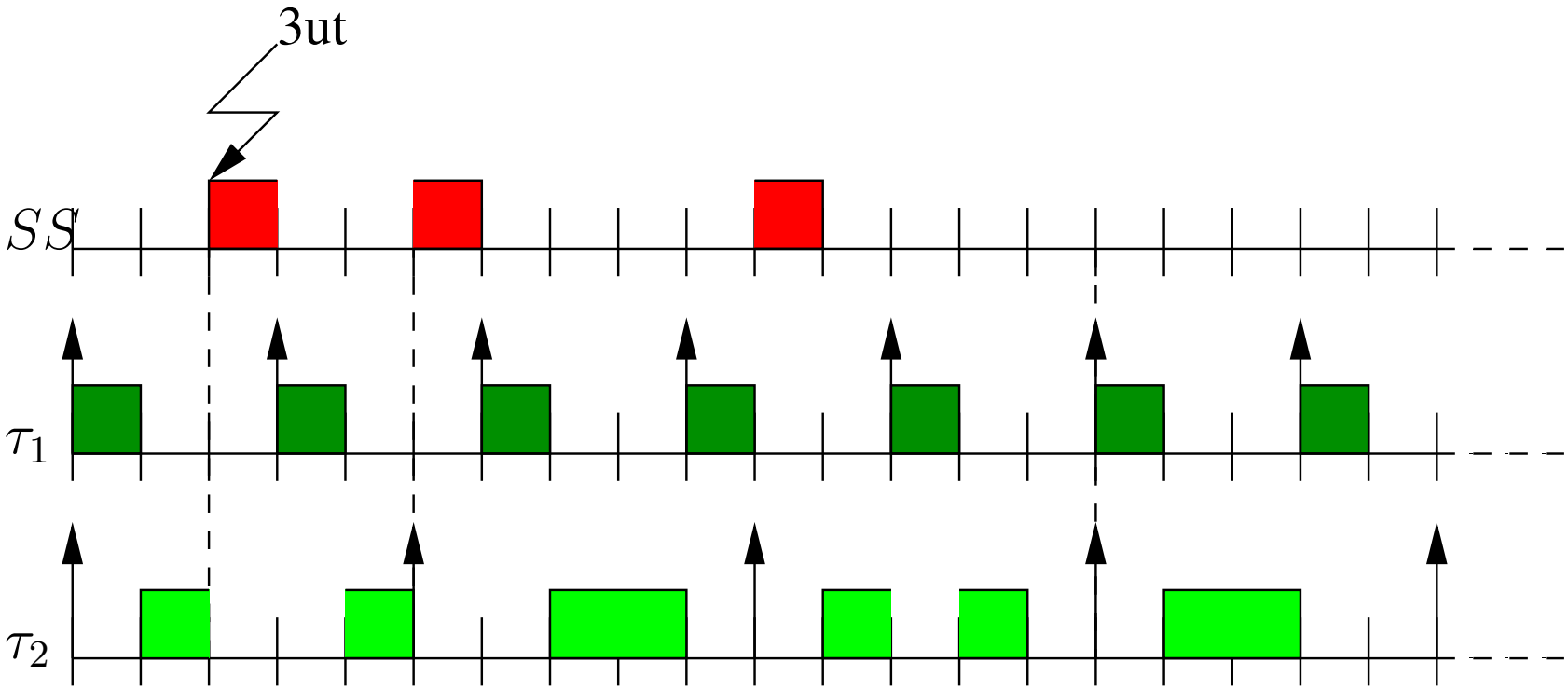
- $S_i(t)$: additional possible work at priority levels $\geq i$ until next τ_i deadline
- $S_t = \min S_i(t)$



Laxity

Definitions

- $S_i(t)$: additional possible work at priority levels $\geq i$ until next τ_i deadline
- $S_t = \min S_i(t)$



Dynamic computation

$S_i(t)$ computation

- polynomial time complexity
- principle: compute the duration of the next busy period, then the one of the following idle period, and so on until the next deadline.

- the complexity of the wall process is too high to be usable in practice
- solution: use a lower bound on the slack time

Principle

- $S_i(t) \geq \left(d_i(t) - t - \sum_{j \leq i} I_j^i(t, d_i(t)) \right)_0$
- $\mathcal{O}(n)$

- $S_i(t)$ increase only when τ_i complete an instance
- between two time instants, S_i decrease by the time spent to execute higher priority tasks

- S_i must so be recomputed at the end of each periodic tasks
- we need to know at each instant how much processor time each time has consumed (not so easy to implement)

Going further

- multiprocessor real-time scheduling,
- more complex arrival time patterns: activation jitter...
- more complex constraints model: (m,k)firm model...
- temporal fault tolerance, overload detection, ...