Real-Time scheduling

B. Miramond
Polytech Nice Sophia

Outline

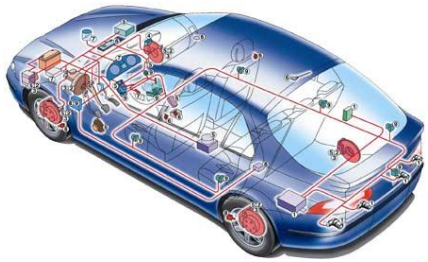
- Task templates: non-functional description of the application
- Fixed priority policies
  - RMS
- Dynamic priority policies
  - EDF
  - LLFS
- Schedulability conditions
- Critical resources and priority inversion
  - Deadlock
  - Priority inheritance
  - Priority ceiling
- Hyper-period
- WCET estimation

Session 1

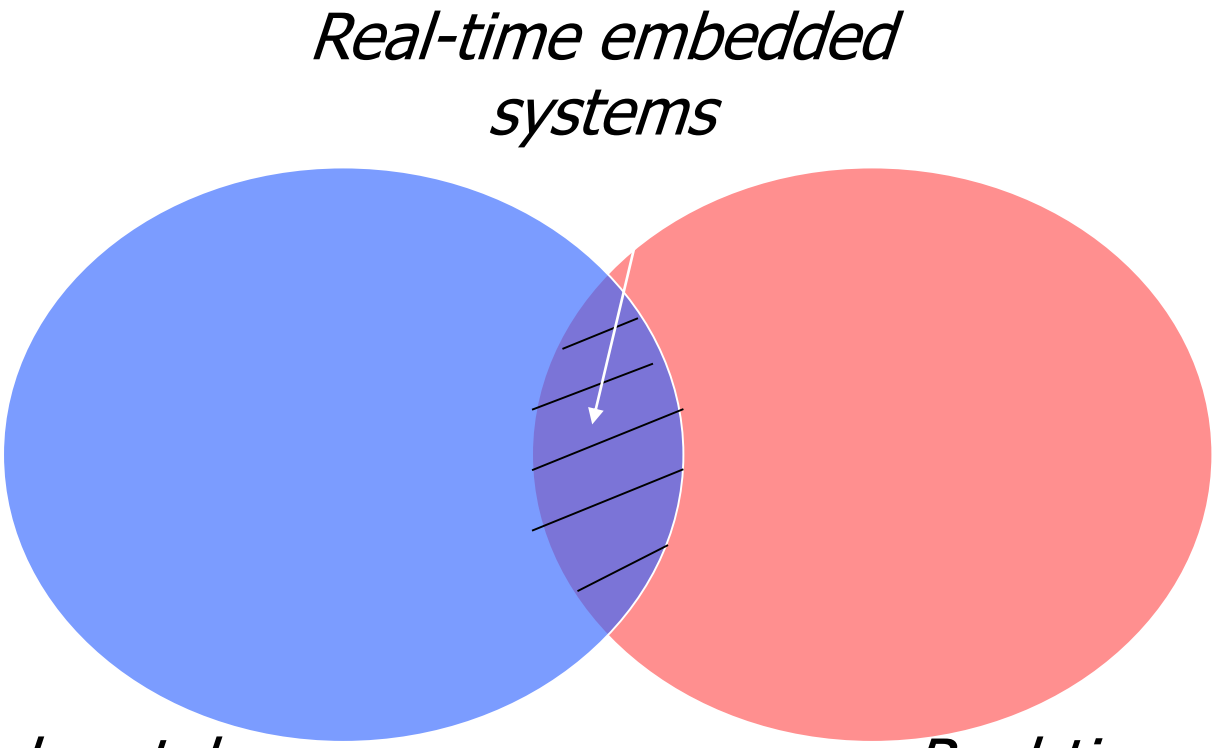
MAIN SCHEDULING ALGORITHMS

Application domains of real-time computing and relation to the sensors

- Automotive (Engine control, ABS, Airbag, etc.),
- Aeronautics and aerospace,
- Military systems,
- Medical systems,
- Industrial processes (nuclear power plants, robotics, chemical production, etc.),
- Surveillance and alarm systems,
- Telecommunication systems…



Realt-time embedded systems



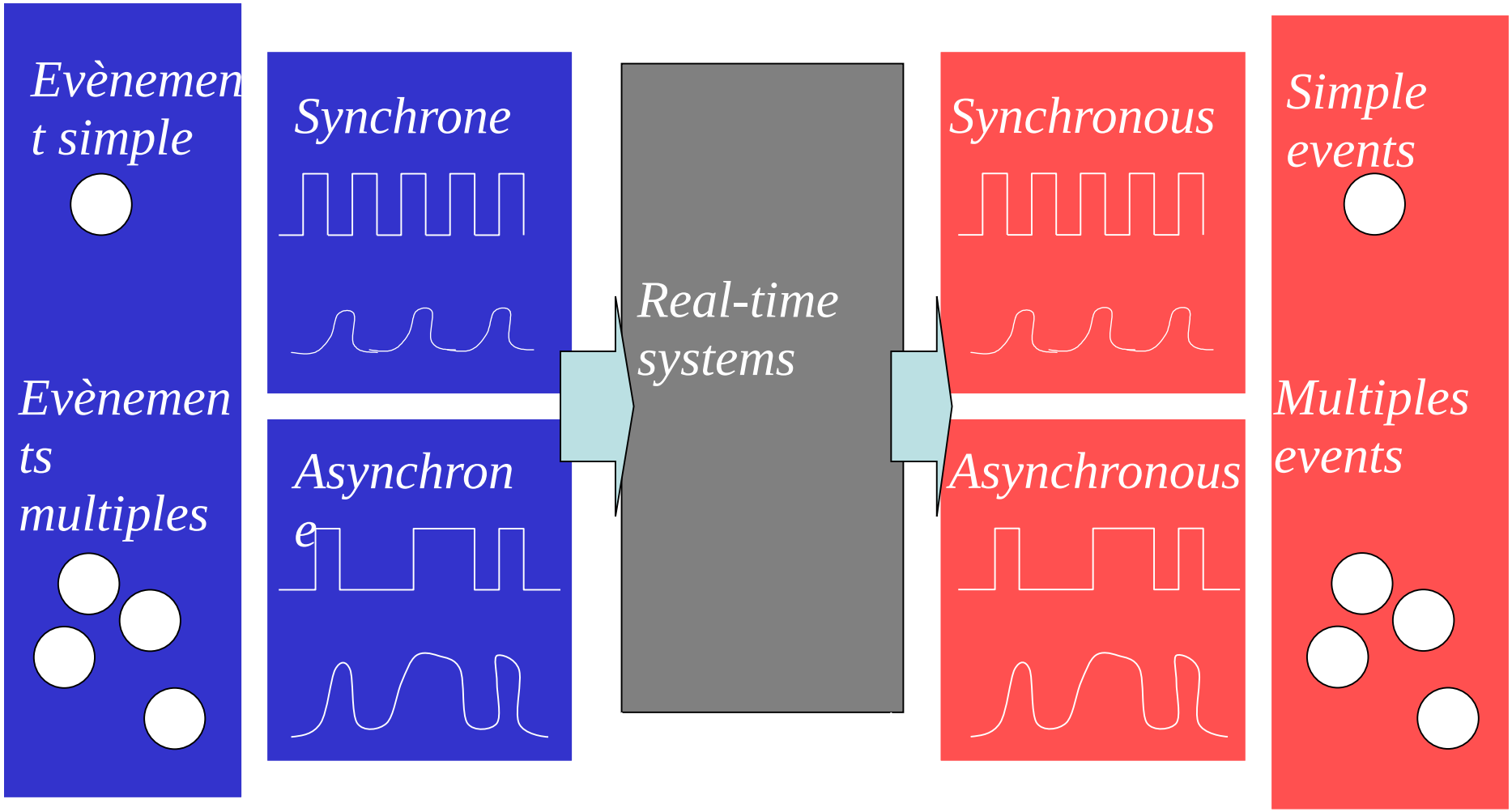
These are systems related to the control of real-world (time) processes.

The execution of processes in these systems must finish before a deadline defined by the speed of change in the real environment, after which the results are no longer valid.

Real-time systems

- It is not a question of delivering the result as quickly as possible, but simply on time.
- The time scale of the deadline can vary from one application to another
  - microseconds in radar control
  - milliseconds for image/sound synchronisation (mpeg)
  - minute for vending machines
  - All can be soft or hard real time

The notion and modeling of time is essential



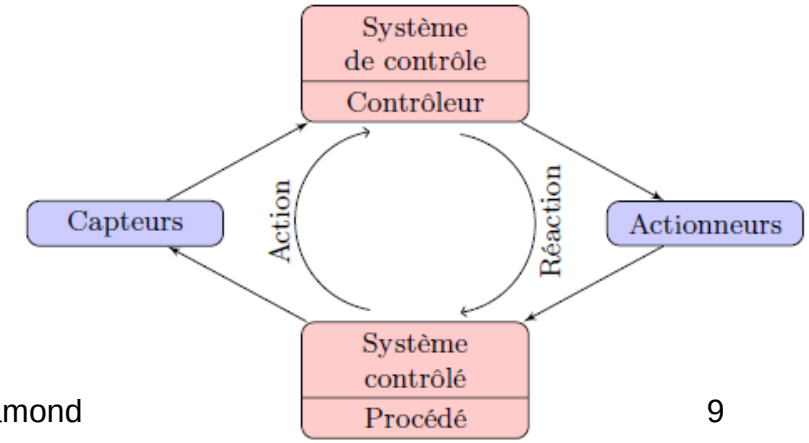
Description of real-time applications

- We are only interested in the temporal aspects of the application,
- We abstract from the functional aspects,
- We describe the functional decomposition into tasks
  - The execution times of these functions
  - The data dependencies between these functions
  - The reactivation periods of these functions

Sensors and multirate systems

Most embedded systems are said to be multirate or multi-period

- The data is captured at a certain rate (from the real world): car wheel revolutions, fps of a camera, ...
- The processing on this data is not necessarily of the same granularity (1 for 64)
- Different processes can occur independently (periodic and aperiodic)
- Actuators operate at a different frequency to sensors



Formal model of tasks

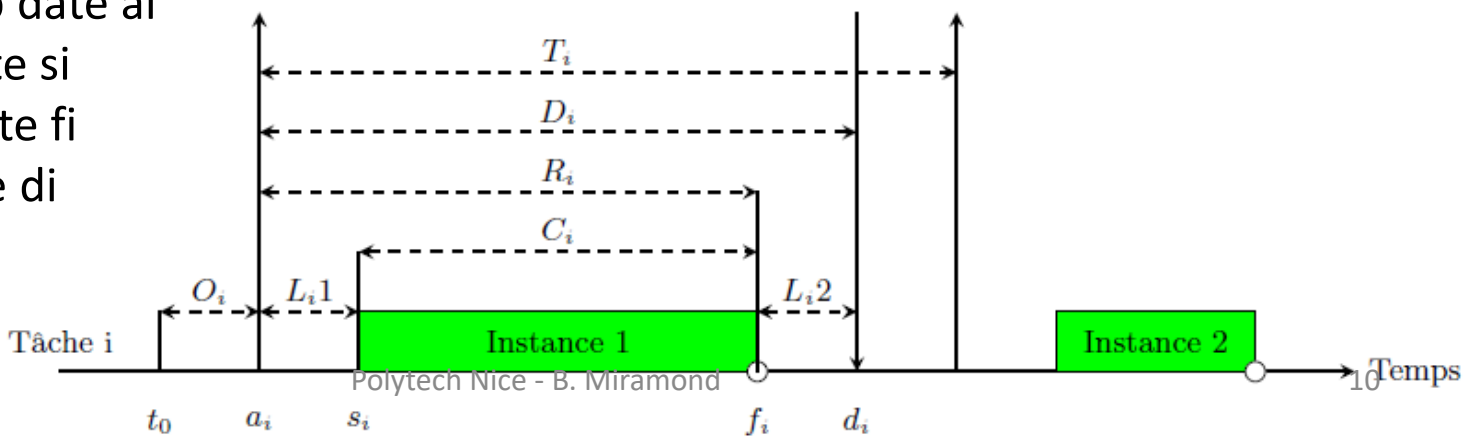
(without dependency between tasks)

A task Ti is modelled by :

- An execution time Ci estimated from the WCET
- A response time Ri, the date between the wake-up and the end of execution (fi - ai)
- A deadline di, date before which the task must be completed
- A reactivation period Ti of a new instance of the task
- An offset Oi = ai - t0. If the offset is known a priori the task is said to be concrete, otherwise it is said to be non-concrete.
- A laxity Li, margin during which the task can be delayed without exceeding the deadline: Li = Di – Ci

Its scheduling is represented by :

- A wake-up date ai
- A start date si
- An end date fi
- A deadline di



Modèle formel de tâches

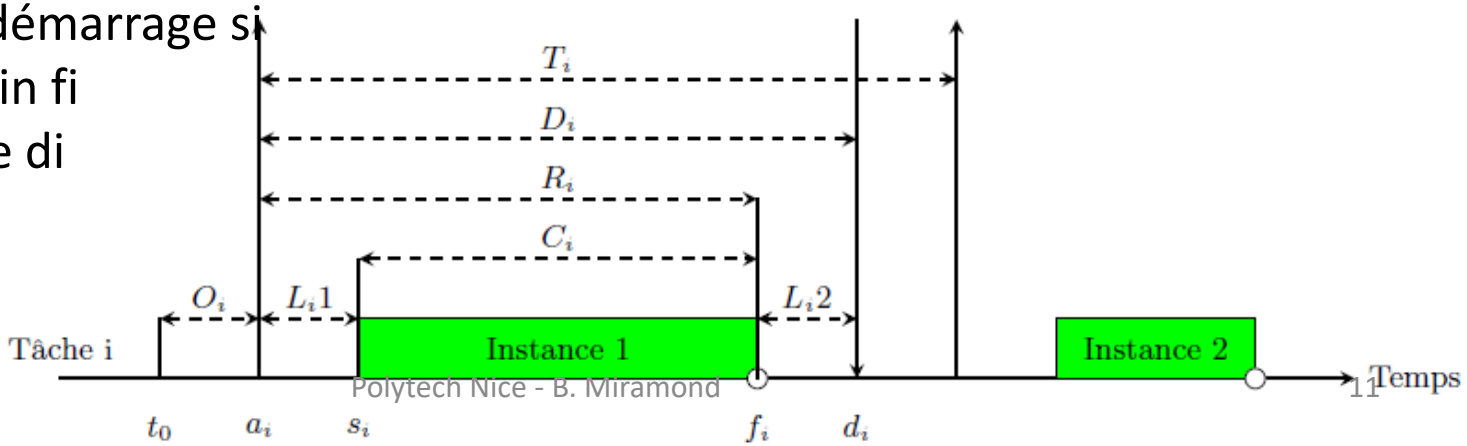
(sans dépendances de données)

Dans ce cours, une tâche Ti sera modélisée par :

- Un temps d'exécution Ci estimé à partir du WCET
- Un temps de réponse Ri, date séparant le réveil de la fin d'exécution (fi – ai)
- Une échéance di, date avant laquelle la tâche doit s'achever
- Une période de réactivation Ti d'une nouvelle instance de la tâche
- Un Offset Oi connu et nul
- Une laxité Li, marge pendant laquelle la tâche peut être retardée sans dépasser l'échéance : Li = Di - Ci

Son ordonnancement est représenté par :

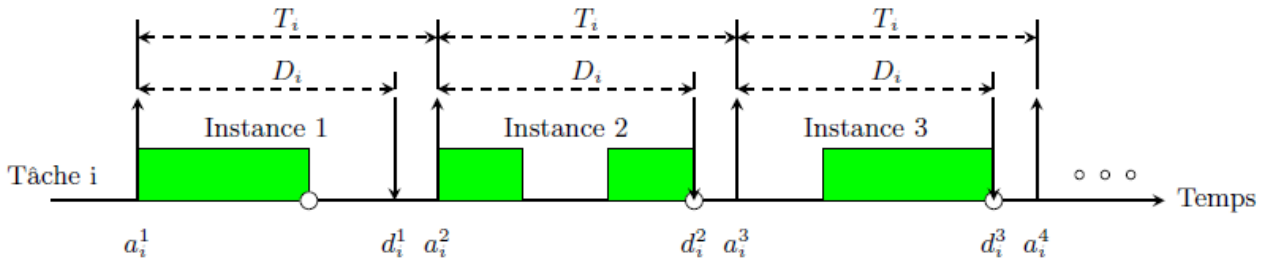
- Une date de réveil ai
- Une date de démarrage si
- Une date de fin fi
- Une échéance di



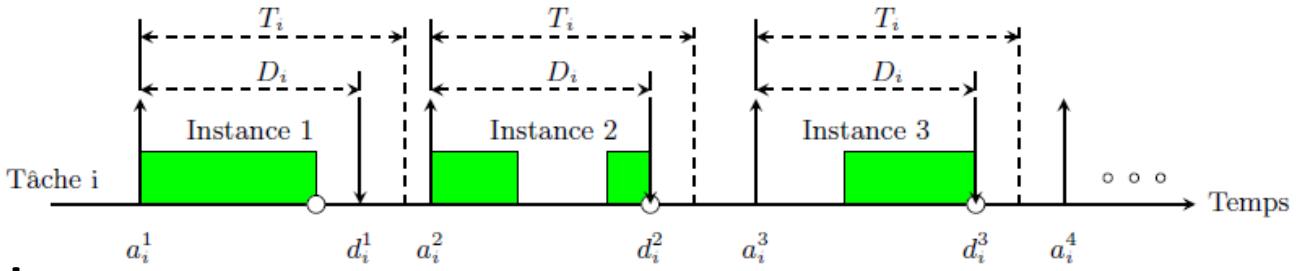
Tasks type

- Periodic tasks

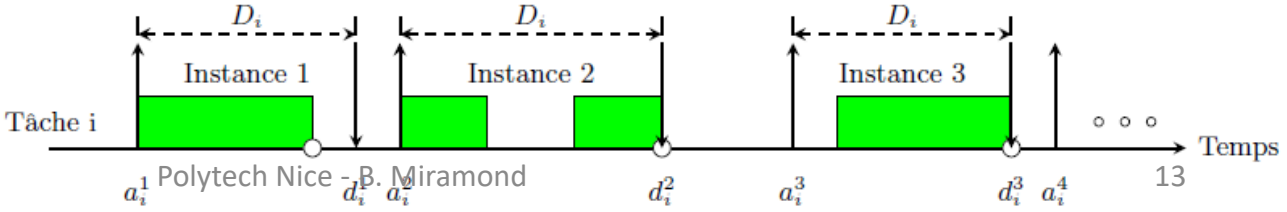
$$a_i^m = a_i^1 + (m-1)T_i \quad ; \quad m \geq 1$$



- Sporadic tasks (cyclical but irregular waking)

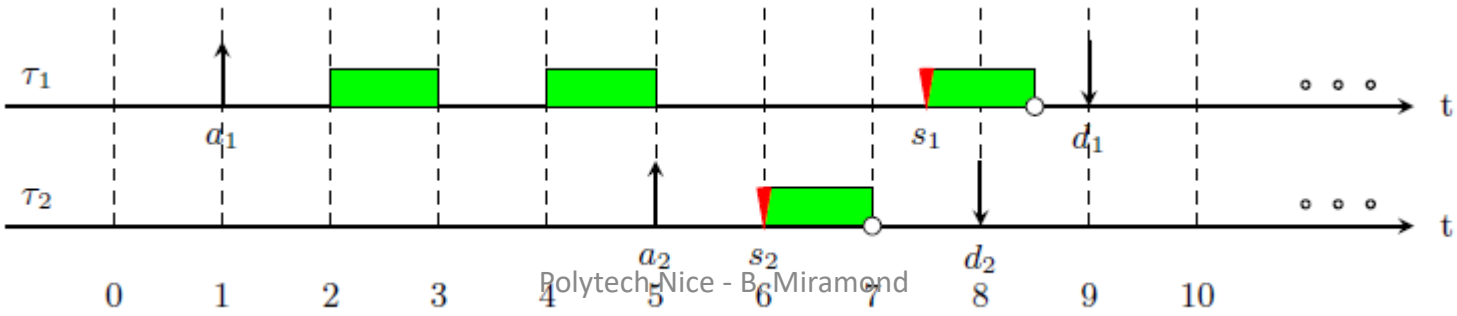


- Aperiodic tasks (non-cyclical, unpredictable awakening)



Scheduling

- Scheduling represents the planning in time of the execution of the tasks of an application
- The result of scheduling is a sequence of task executions usually represented by a timeline or Gantt chart



Classification of scheduling strategies

- Off-line / on-line scheduling
  - Determined before execution in a table, static scheduling
  - Determined during execution, more flexible but more expensive, also called dynamic scheduling
- Pre-emptive / non-pre-emptive scheduling
  - Preemptive when the execution of a task can be interrupted by others: execution overhead
  - Non-preemptive: simpler
- Static / dynamic priority scheduling
  - Scheduling is based on the notion of priority. At each moment, the task with the highest priority (HPT - Highest Priority Task) that is ready (Ready state) is elected by the scheduler for execution.
  - If these priorities can change in the course of time, we speak of dynamic priorities.
- Single Processor / Multi Processor Scheduling

Schduling strategies for periodic tasks

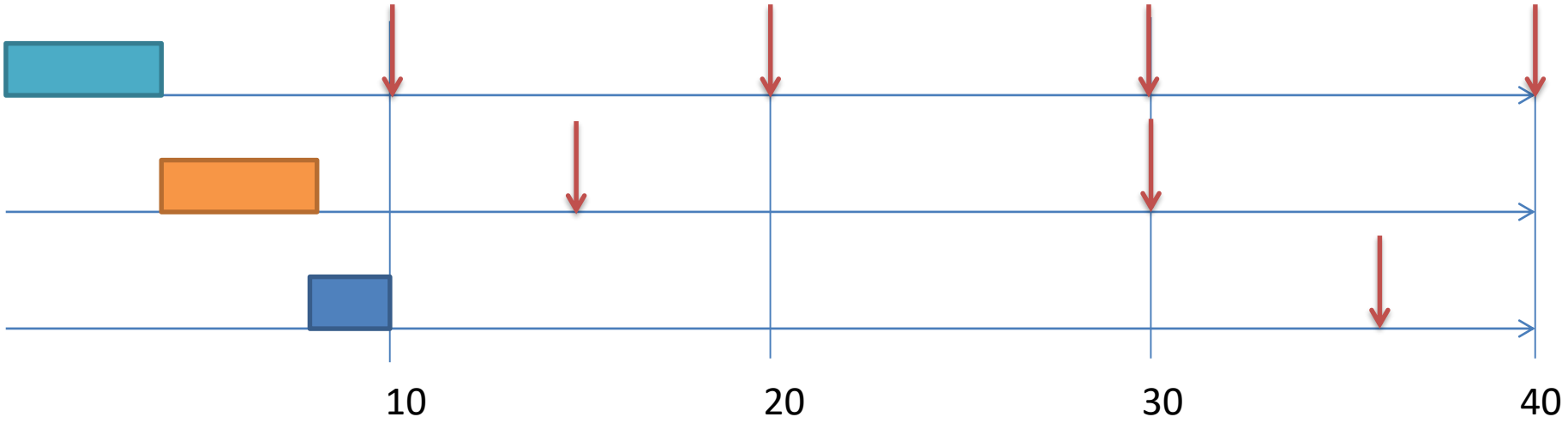
- EDF (Earliest Deadline First)
  - Dynamic priority algorithm: the closer the due date, the higher the priority
  - EDF is said to be optimal in the sense that if a set of tasks cannot be scheduled by EDF, then it cannot be scheduled by any other algorithm
  - A set of periodic tasks can be scheduled by EDF if its load factor U is less than or equal to 1

$$U = \sum_{i=1}^{n} \frac{Ci}{Ti} \leq 1$$

[EDF] J. Jackson. Scheduling a production line to minimize maximum tardiness. Research Report 43. Management Science Research Project, University of California, Los Angeles, 1955.

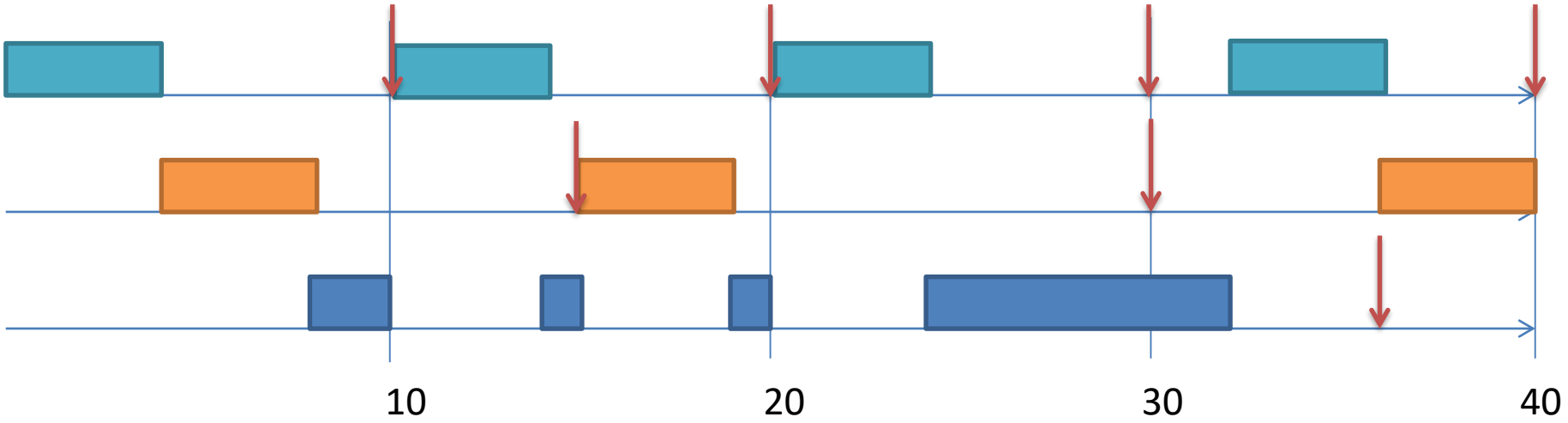
An example of scheduling with EDF

	période	calcul	utilisation
τ_1	10	4	0.400
τ_2	15	4	0.267
τ_3	36	12	0.333



An example of scheduling with EDF

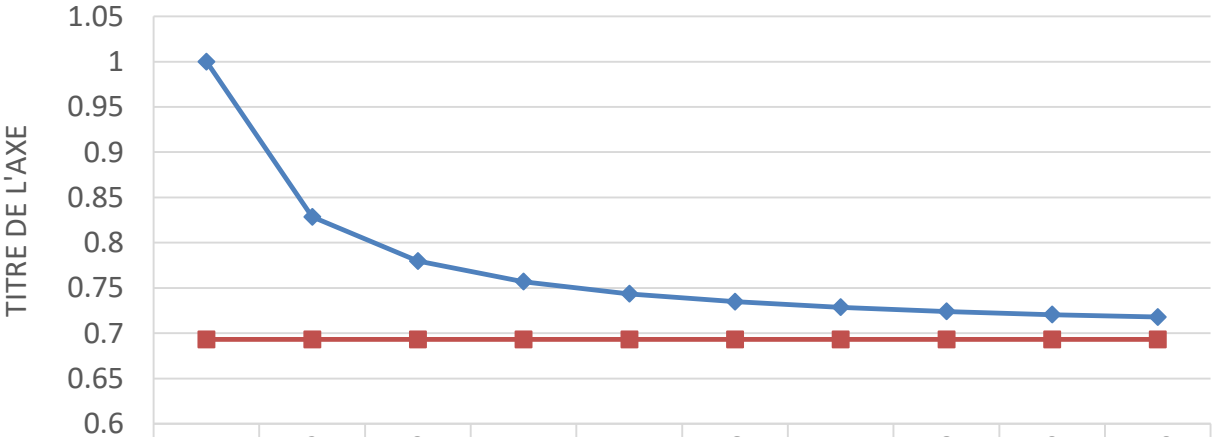
	période	calcul	utilisation
τ_1	10	4	0.400
τ_2	15	4	0.267
τ_3	36	12	0.333



Scheduling strategies for periodic tasks

- Rate Monotonic Analysis / Scheduling (RMA/RMS)
  - RMS is a fixed priority algorithm, a task has a higher priority when its period is smaller
  - The schedulability condition of RMS, for n tasks, is

$$U = \sum_{i=1}^{n} \frac{Ci}{Ti} \leq n(2^{\frac{1}{n}} - 1)$$

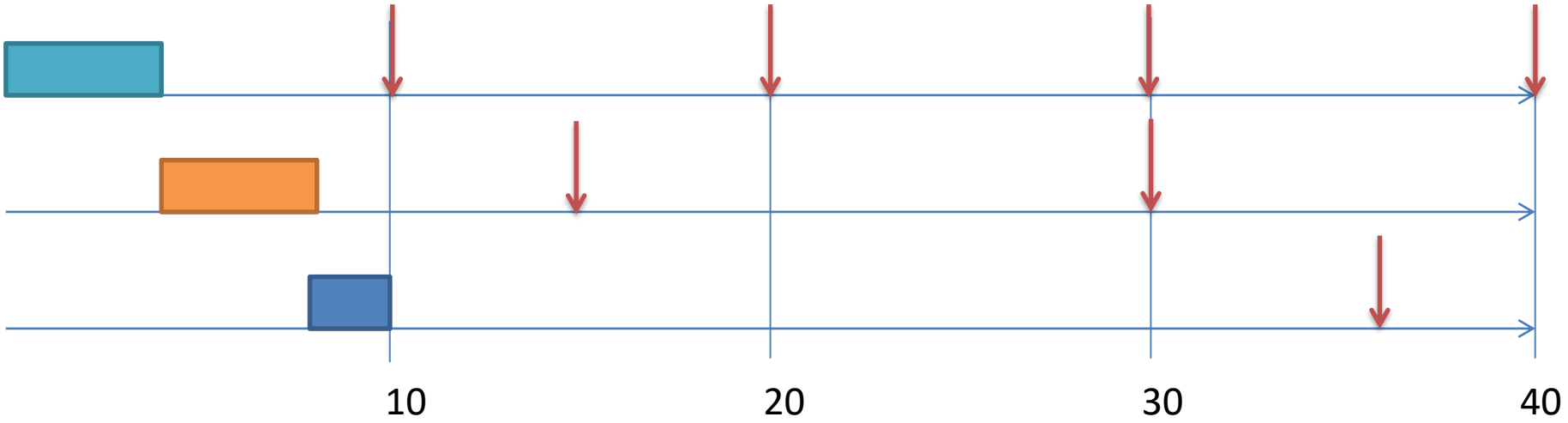


	1	2	3	4	5	6	7	8	9	10
U	1	0.8284	0.7798	0.7568	0.7435	0.7348	0.7286	0.7241	0.7205	0.7177
limite	0.6931	0.6931	0.6931	0.6931	0.6931	0.6931	0.6931	0.6931	0.6931	0.6931

[RMS] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM), 20(1) :46–61, 1973.

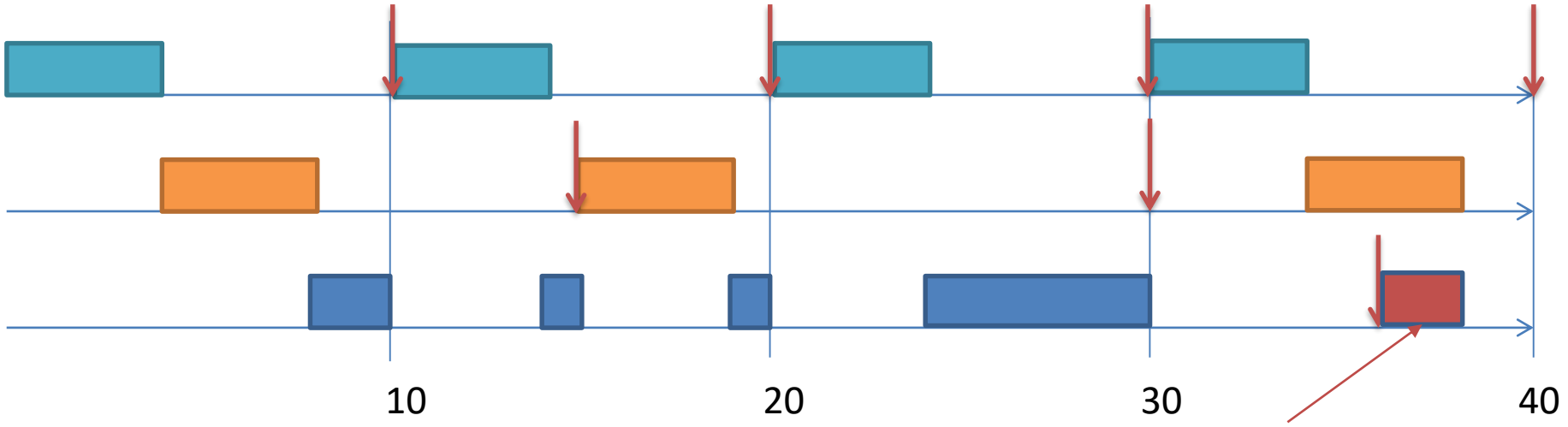
An example of scheduling with EDF

	période	calcul	utilisation
τ_1	10	4	0.400
τ_2	15	4	0.267
τ_3	36	12	0.333



An example of scheduling with EDF

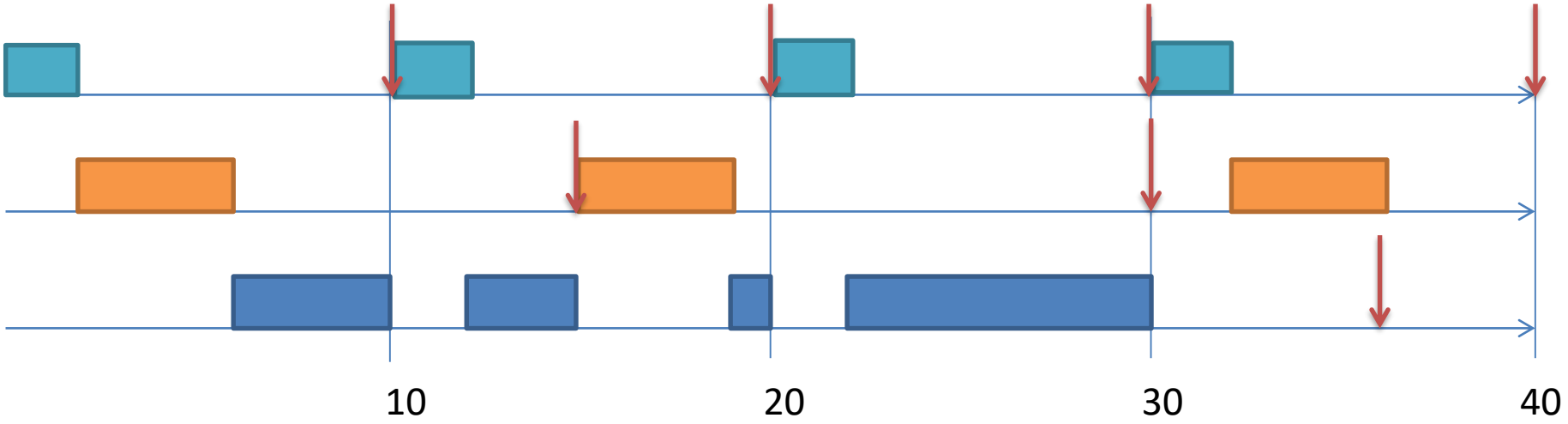
	période	calcul	utilisation
τ_1	10	4	0.400
τ_2	15	4	0.267
τ_3	36	12	0.333



An example of scheduling with EDF

U max (3) = 0,77976315, U = 0,8

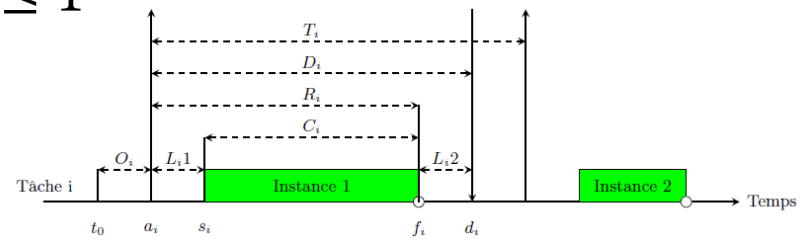
$3 \times (2^{1/3} - 1) \approx 0.78$	période	calcul	utilisation
τ_1	10	2	0.200
τ_2	15	4	0.267
τ_3	36	12	0.333



Scheduling strategies for periodic tasks

- **Least Laxity First (LLF)**
  - Dynamic priority algorithm, the higher the priority, the lower the laxity
  - The laxity at the present time is calculated as the time before the next deadline minus the remaining execution time of the task
  - Disadvantage: prohibitive number of preemptions and therefore context changes, high overhead
  - Same condition as EDF

$$U = \sum_{i=1}^{n} \frac{Ci}{Ti} \leq 1$$



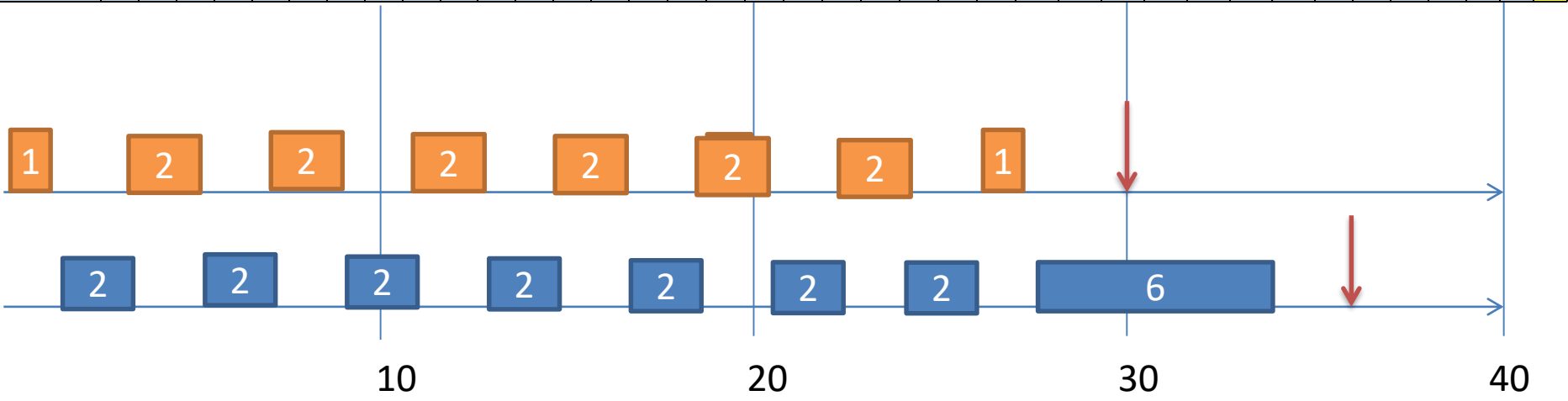
A scheduling example with LLF

Task (execution time, period)
T2 (14, 30)
T3 (20, 36)

When 2 tasks have the same laxity, it generates more preemptions

Laxity

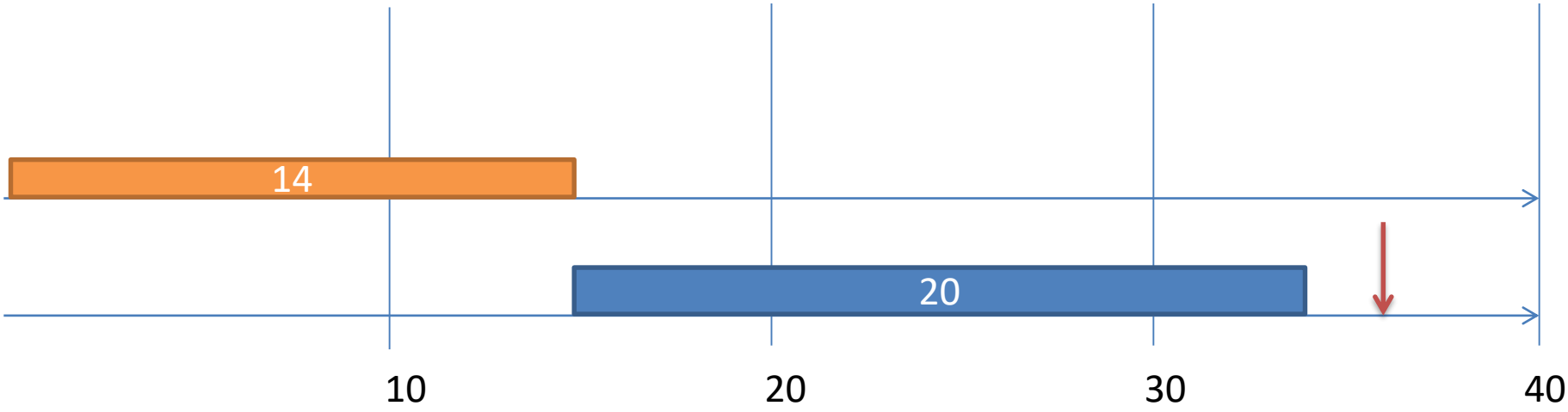
	Tim	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
Laxité	T2	**16**	16	15	**14**	**14**	14	13	**12**	**12**	12	11	**10**	**10**	10	9	**8**	**8**	8	7	**6**	**6**	6	5	**4**	**4**	4	3	**2**									
	T3	16	**15**	**15**	15	14	**13**	**13**	13	12	**11**	**11**	11	10	**9**	**9**	9	8	**7**	**7**	7	6	**5**	**5**	5	4	**3**	**3**	3	**2**	**2**	**2**	**2**	**2**	**2**			



A scheduling example with LLF

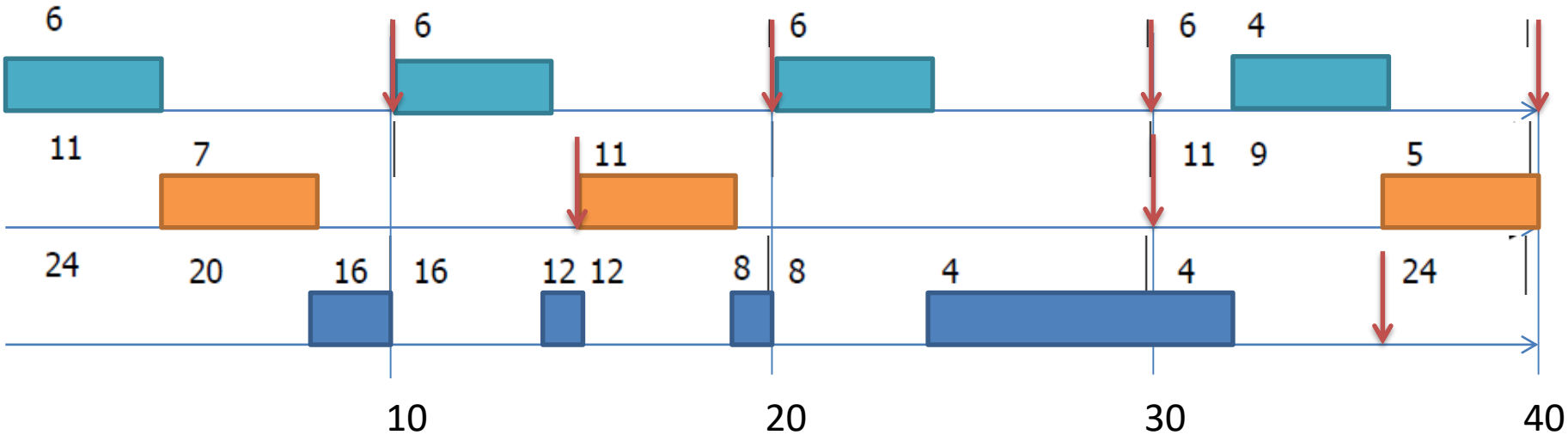
Task (execution time, period)
T2 (14, 30)
T3 (20, 36)

With the same laxity between 2 tasks, this is equivalent to keeping the priority of the current task as long as no other task wakes up



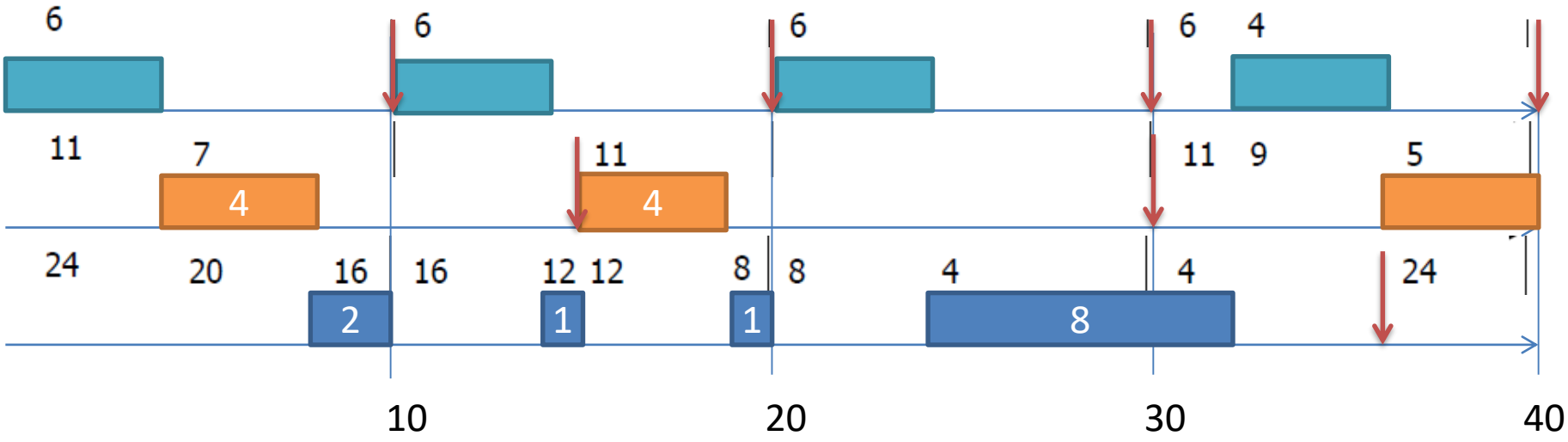
A second example with LLF

	période	calcul	utilisation
τ_1	10	4	0.400
τ_2	15	4	0.267
τ_3	36	12	0.333



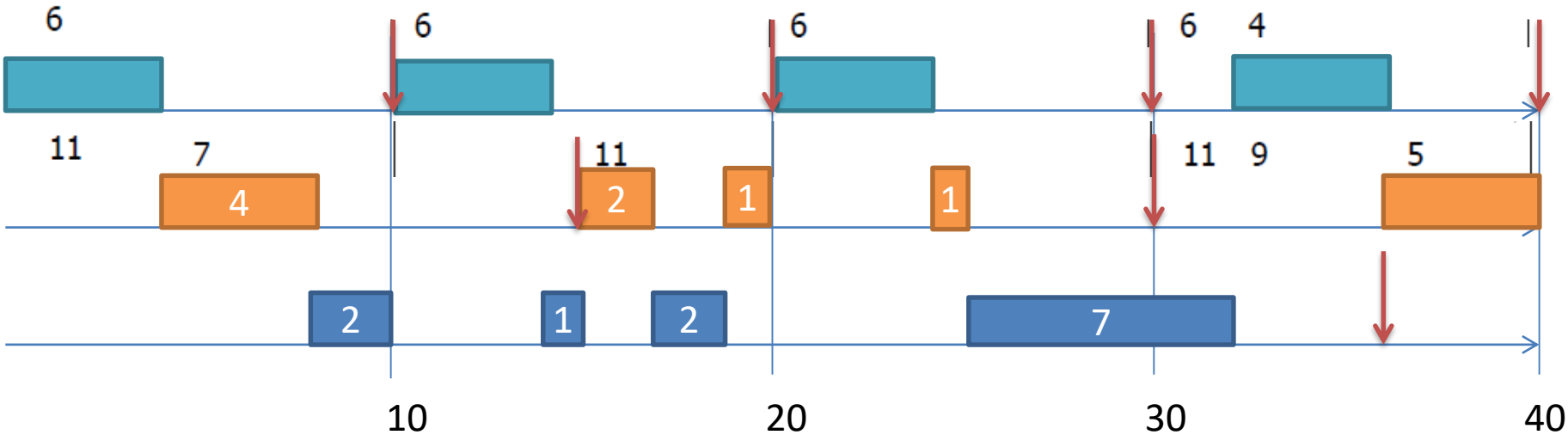
A second example with LLF
Illustration of laxity frame date 15

date	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
T1	-	-	-	-	-	**6**	**6**	**6**	**6**	-	-	-	-	-	-	6	5	**4**	**4**	**4**	**4**	-
T2	**11**	**11**	**11**	**11**	-	-	-	-	-	-	-	-	-	-	-	11	10	9	8	7	6	**5**
T3	12	11	10	9	8	8	7	6	5	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	-	-	-	-	24



A second example with LLF
Illustration of laxity frame date 15

date	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
T1	-	-	-	-	-	**6**	**6**	**6**	**6**	-	-	-	-	-	-	6	5	**4**	**4**	**4**	**4**	-
T2	**11**	**11**	11	10	**9**	9	8	7	6	**5**	**4**	-	-	-	-	11	10	9	8	7	6	**5**
T3	12	11	**10**	**10**	10	9	8	7	6	5	4	**4**	**4**	**4**	**4**	**4**	**4**	-	-	-	-	24



Scheduling strategies for periodic tasks

- **Deadline monotonic scheduling (DMS)**
  - Static priority algorithm, high priority if the critical time D_i of the task is small
  - DMS = RMS when Di = Ti
  - Most commonly used algorithm in practice

[DMS] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. Performance Evaluation, 2(4) :237–250, 1982

Scheduling criteria

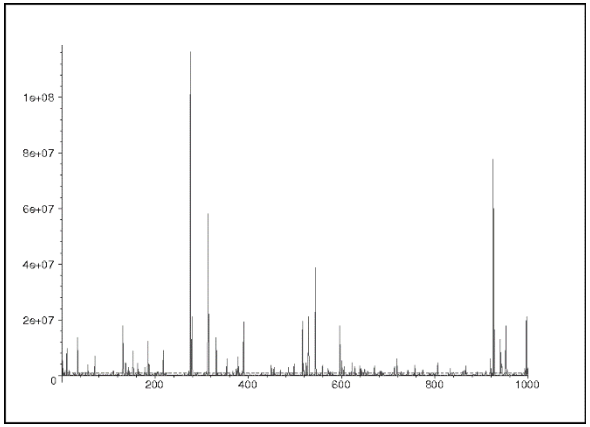
Criteria	Fixed priority	Dynamic priority
Execution time		
Period		
Deadline		
Utilisation of CPU		
Energy consumption		
Laxity		

Critères d'ordonnancement

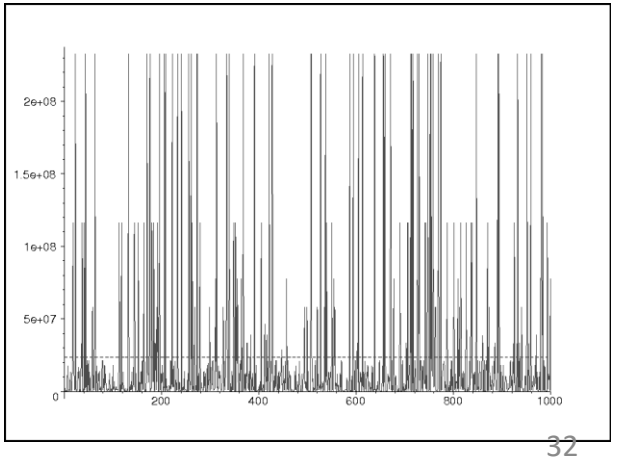
Criteria	Fixed priority	Dynamic priority
Execution time	X	
Period	X	
Deadline	X (DMS)	X (EDF)
Utilisation of CPU		X
Energy consumption		X
Laxity		X

Period of a schedule

- The sequence produced by any preemptive scheduling algorithm on a set of periodic tasks is itself periodic with period equal to the Least Commum Multiple (LCM) of the periods of the tasks in the configuration, denoted P, Hyper-period
- The scheduling will therefore be in the same state at date t + kP, $k \in N$
- Thus, if a scheduling condition is not valid, we need to simulate the scheduling on P to check if this set of tasks is feasible
- But this value P grows exponentially with the number of tasks and the value of the largest period



Set of tasks whose periods are randomly drawn in [1,10].



10 tâches 20 tâches

Theorem of critical zone

- If all tasks initially arrive in the system simultaneously and meet their first deadline, then all deadlines will be met thereafter, regardless of when the tasks arrive.
- To do this, we need to find the termination time t, before the first deadline $_i$ of the task with the largest period:

$$\forall i,\ 1 \le i \le n \qquad \min_{0<t<D_i} \sum_{j=1} C_j / t * \lceil t / T_j \rceil \le 1$$

- The tasks are indexed from the smallest period to the largest
- To solve the problem, an iterative method is applied

Iterative method for the critical zone theorem

The time t is sought such that:

$$\forall i,\ 1 \le i \le n,\ \exists\ t \le Di, \qquad t = \sum_{j=1}^{i} C_j * \lceil t / T_j \rceil$$

$$W_i(t) = \sum_{j=1}^{i} C_j * \lceil t / T_j \rceil$$

Wi(t) represents the cumulative demand for processor time of all tasks up to task i in the interval [0,t].tasks up to task i in the interval [0,t].

Wi(t) / Wi(t)=t is searched by successive iterations for all tasks i

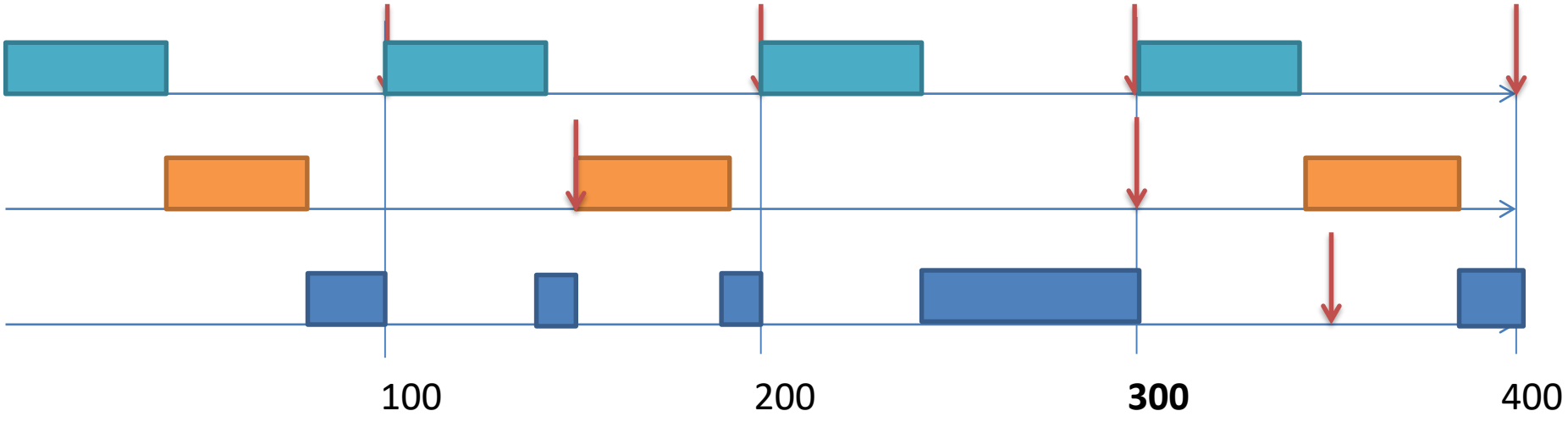
We start with $t0 = \sum_{j=1} Cj$, if the time t does not respect the condition, we iterate by taking as new time t = Wi(t)

Example

Pi	T	C
P1	100	40
P2	150	40
P3	350	100

$\sum$ U = 0.779,

But we will show that the example meets the critical zone theorem.



Example

Pi	T	C
P1	100	40
P2	150	40
P3	350	100

$\sum U = 0.779$,

But we will show that the example meets the critical zone theorem.

$$t = \sum_{j=1}^{i} C_j * \lceil t / T_j \rceil$$

For i from 1 to 3 :

For i=1 t0 = C1 = 40, W1(0) =40*1 = 40, W1(40)=40, i++

For i=2 t0 = C1+C2 = 80 => W2(0) = 80, W2(80)=80, i++

For i=3 t0 = C1+C2+C3 = 180 (>100 et 150)

W 3(180)= 2C1+2C2+C3 = 260 > 180

W 3(260)= 3C1+2C2+C3 = 300 > 260

W 3(300)= 3C1+2C2+C3 = 300

and **< 350 (T3)**, that is the termination condition

$$\min_{0<t<D_i} \sum_{j=1}^{i} C_j / t * \lceil t / T_j \rceil \leq 1$$

3*40/300 + 2*40/300 + 1*100/300 = 300/300 = 1

The three tasks are therefore schedulable according to the critical area theorem. The third task will complete the execution of its first instance at time 300.

Session 2

SHARED RESSOURCES

Shared Ressources

Access problem to a shared resource

- Resource protected by a mutual exclusion access mechanism (Mutex or Semaphore)
- Pre-emptive scheduling can lead to a priority inversion situation where a low priority task blocks a high priority task for a time longer than the mutual exclusion time
- The upper bound of this time cannot be evaluated

Problem solution: Priority inheritance protocol

- Priority inheritance changes the priority of the blocking task to the blocked task
- Once the semaphore is released, the blocked task regains its initial priority

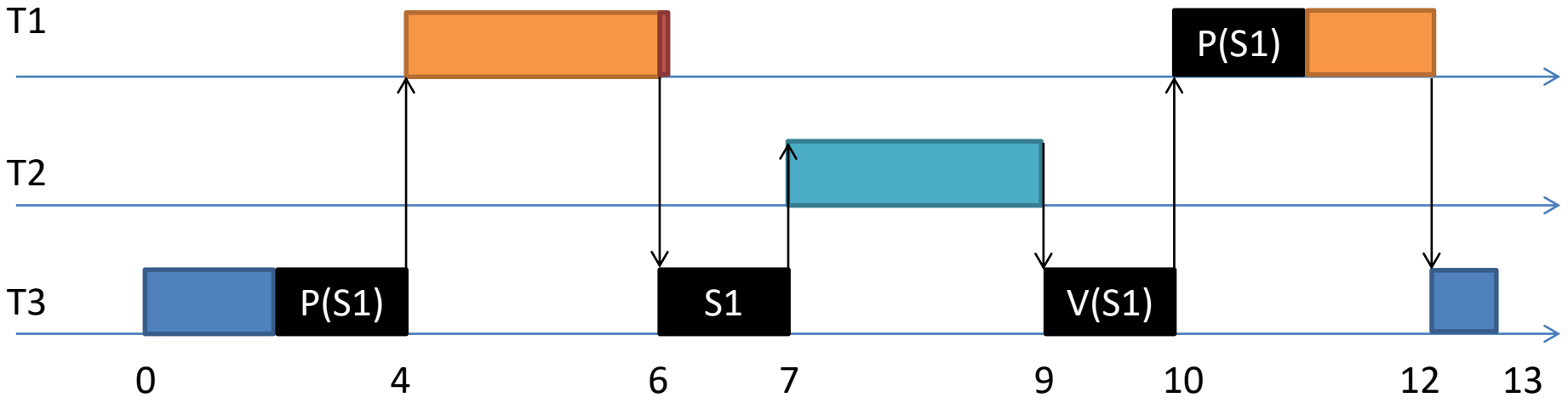
Semaphore

- Purpose :
  - to restrict access to shared resources (e.g. storage space) synchronize processes in a concurrent programming environment...
  - invented by Edsger Dijkstra in 1965
  - The three operations supported are Init, P and V.
    - The P operation waits until a resource is available, which is immediately allocated to the current process.
    - V is the opposite operation; it simply makes a resource available again after the process has finished using it.
    - Init is only used to initialise the semaphore. This operation should only be used once.
  - The binary semaphore is a mutual exclusion (or mutex). It is always initialized with the value 1.

The problem of reversing priorities

- Tasks access shared resources through a mutual exclusion mechanism
- Tasks become dependent on each other
- **Problem**: a lower priority task can block another higher priority task when it is in a critical section, this is the problem of priority inversion.

Example



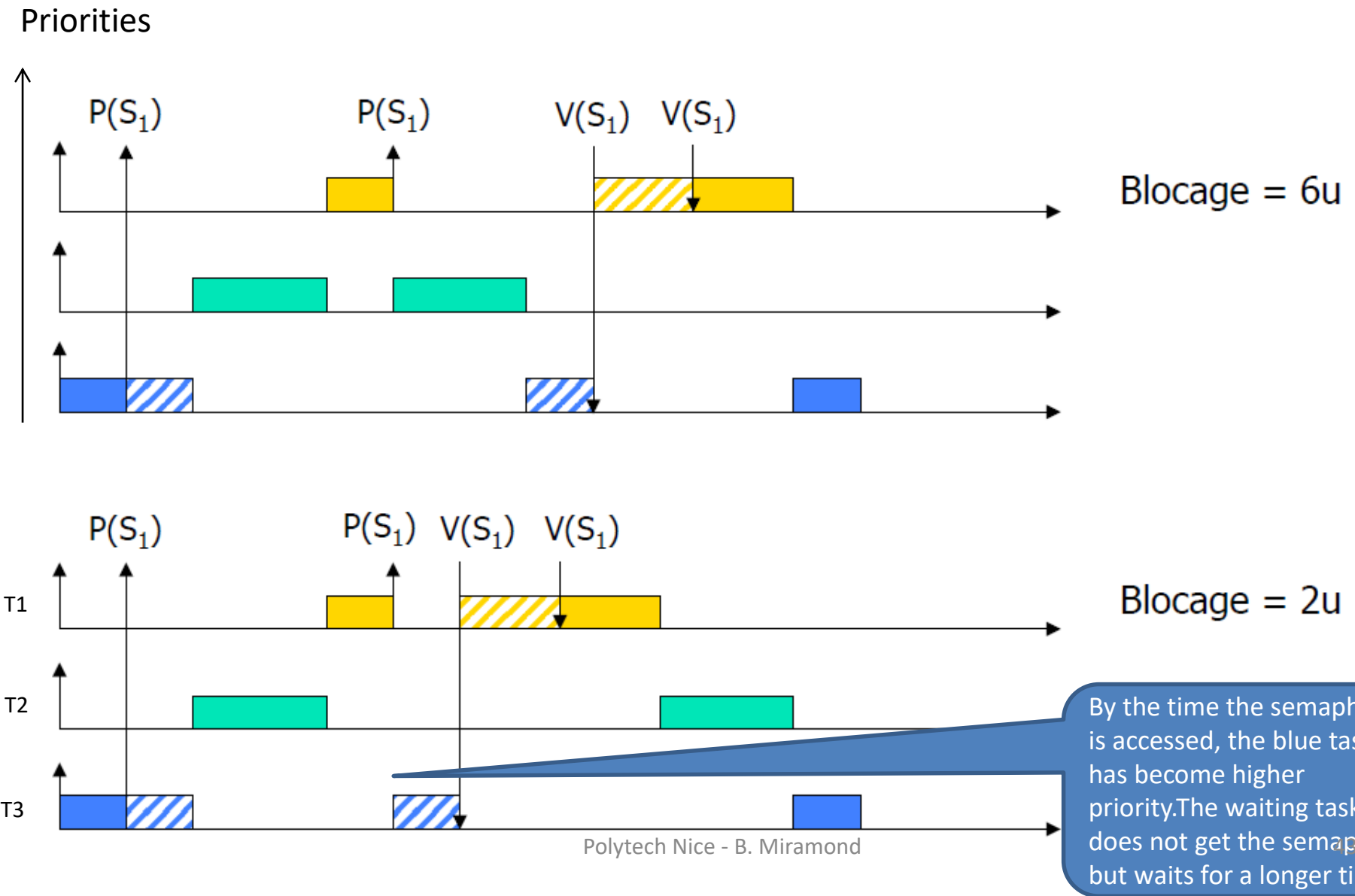
t0 - task T3 starts
t2 - T3 accesses semaphore S1
t4 - T1, with higher priority, preempts T3
t6 - T1 requests S1 taken by T3. T3 takes over
t7 - T2, with higher priority, preempts T3
t9 - T2 finishes and gives back the hand
t10 - T3 releases the semaphore, T1 can execute, there has been a priority inversion
t12 - T1 finishes with a delay

Solution: Priority inheritance method

Priority inheritance

- The system performs dynamic priority management. The idea is to add to the semaphores the notion of possessors.
  - The owner of a semaphore is the task that has requested and obtained the right to enter the mutual exclusion zone protected by the semaphore.
  - In simple priority inheritance, semaphores manage their queue taking into account the priorities of the tasks that wish to take the semaphore.
  - When a task holds a semaphore and another task claims it, the priority of the task that holds the semaphore is increased to the priority of the task that claims it.

Example of priority inheritance



Priority Ceiling Protocol

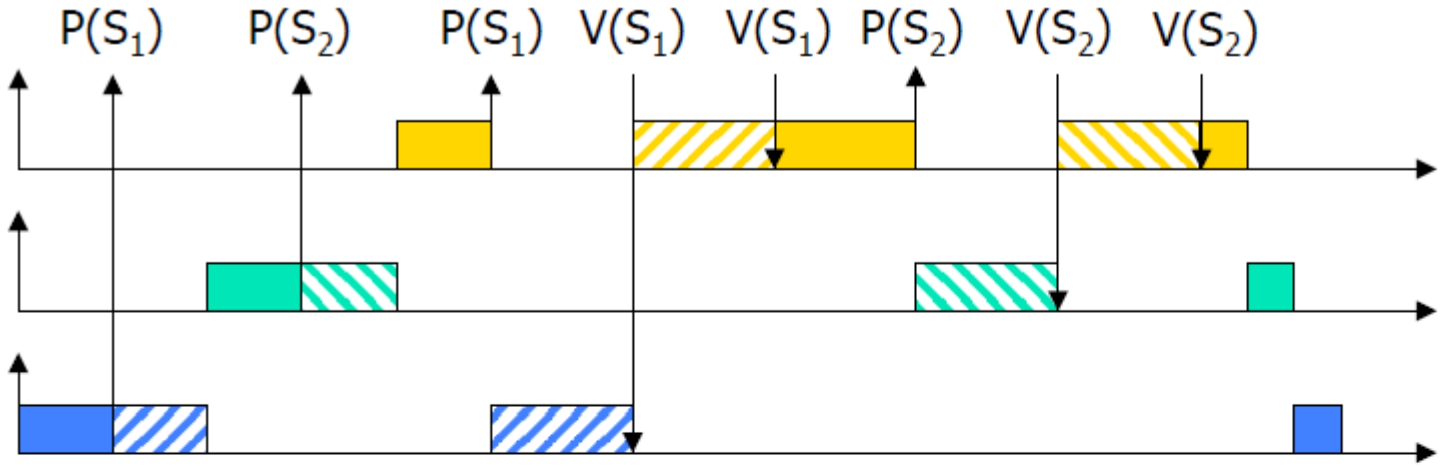
Despite this first protocol,

- Blocking times can occur in sequence
- Priority upgrades can occur
- Interlocking is still possible

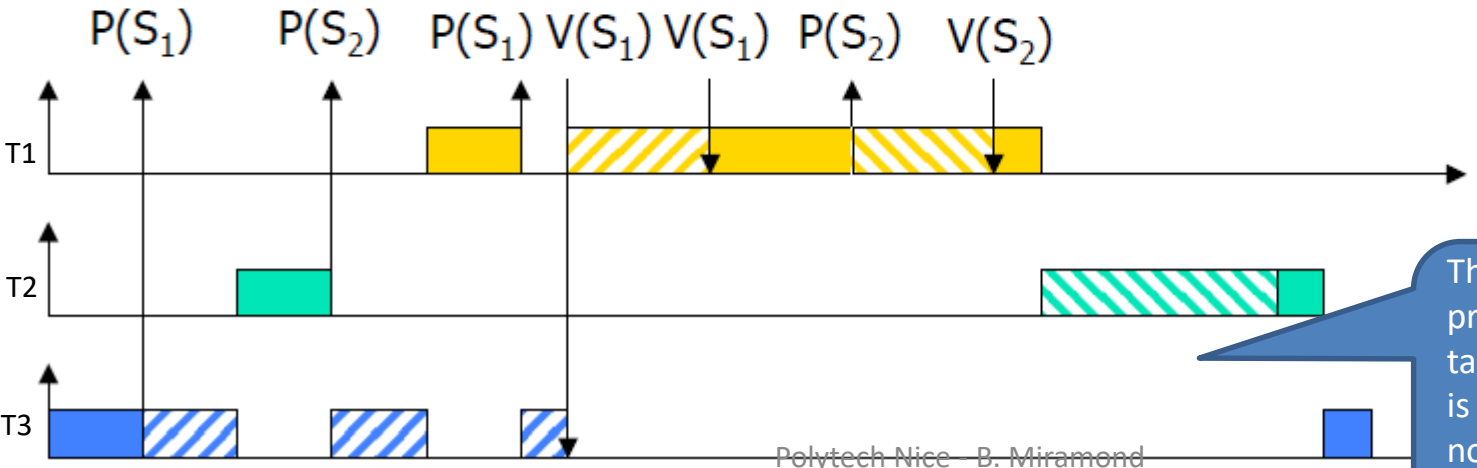
Solution: Priority cap

- The capped (static) priority represents the maximum priority of the tasks that use it
- A task accesses a semaphore when its priority is strictly higher than all the capped priorities of the semaphores used
- In other words, the OS maintains a value that represents the maximum value of the current ceiling.
- When a task tries to execute a critical section, it is suspended unless its priority is higher than the priority ceiling of all semaphores taken by other tasks.

Priority Ceiling Protocol



Blocage = 6u



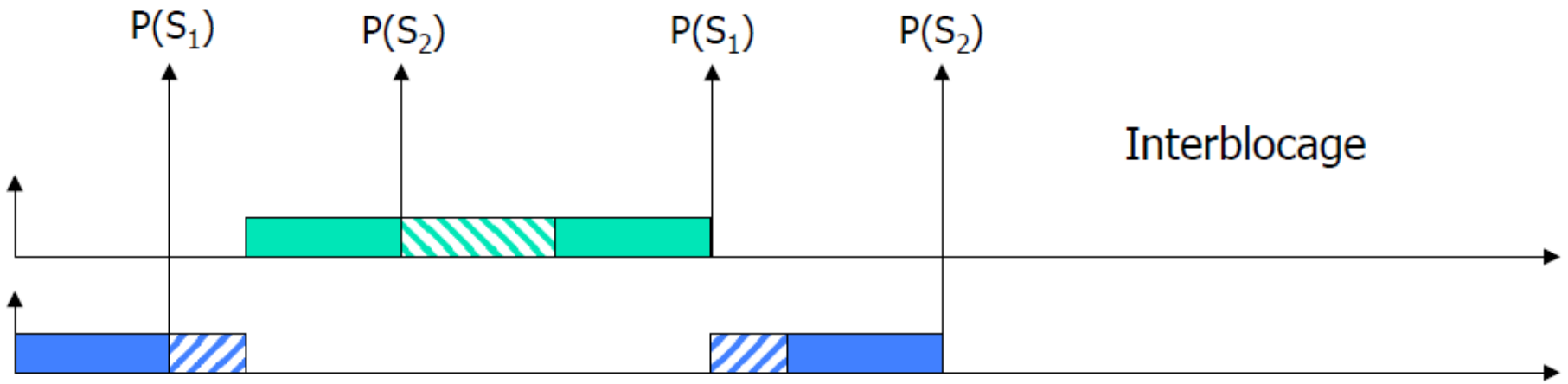
Blocage = 1u

The ceiling is set at the priority of the yellow task.When T2 requests S2, it is under the ceiling and does not get it. T1 is therefore only blocked by T3.

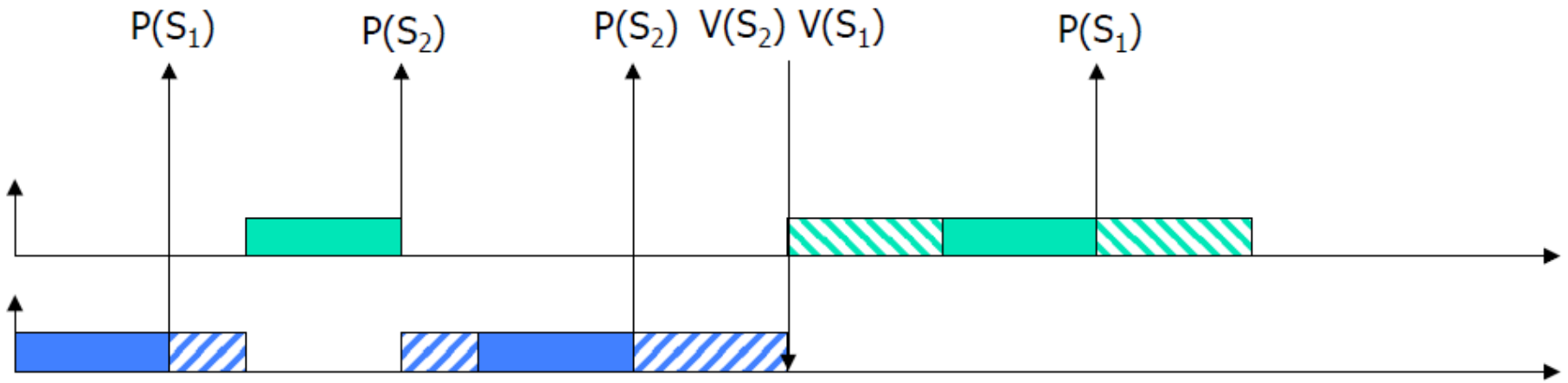
Deadlock

- An interlock, deadlock, or fatal lock is a phenomenon that can occur when two concurrent processes wait for each other through several exclusive accesses

Deadlock example



Avec plafond de priorité



Conclusion

- Several scheduling methods depending on the criteria to be met by the application
  - Task frequencies, RMSDeadlines, EDFLaxity, LLF
- In the case of access to shared resources, sequencing is modified
- Dynamic priority management methods are then necessary