Sistemi in tempo reale

Earliest Deadline First

Giuseppe Lipari

Scuola Superiore Sant'Anna

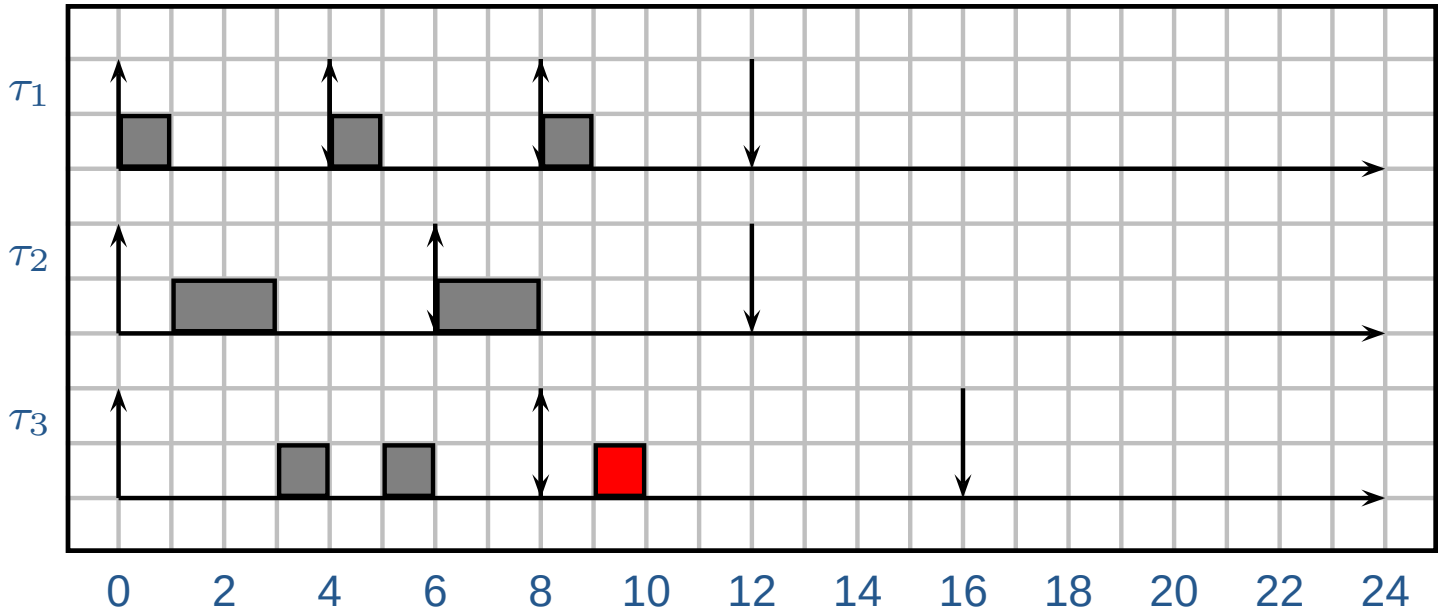
Pisa -Italy

Earliest Deadline First

- An important class of scheduling algorithms is the class of *dynamic priority* algorithms
  - In dynamic priority algorithms, the priority of a task can change during its execution
  - Fixed priority algorithms are a sub-class of the more general class of dynamic priority algorithms: the priority of a task does not change.
- The most important (and analyzed) dynamic priority algorithm is Earliest Deadline First (EDF)
  - The priority of a job (istance) is inversely proportional to its absolute deadline;
  - In other words, the highest priority job is the one with the earliest deadline;
  - If two tasks have the same absolute deadlines, chose one of the two at random (*ties can be broken arbitrarly*).
  - The priority is dynamic since it changes for different jobs of the same task.

Example: scheduling with RM

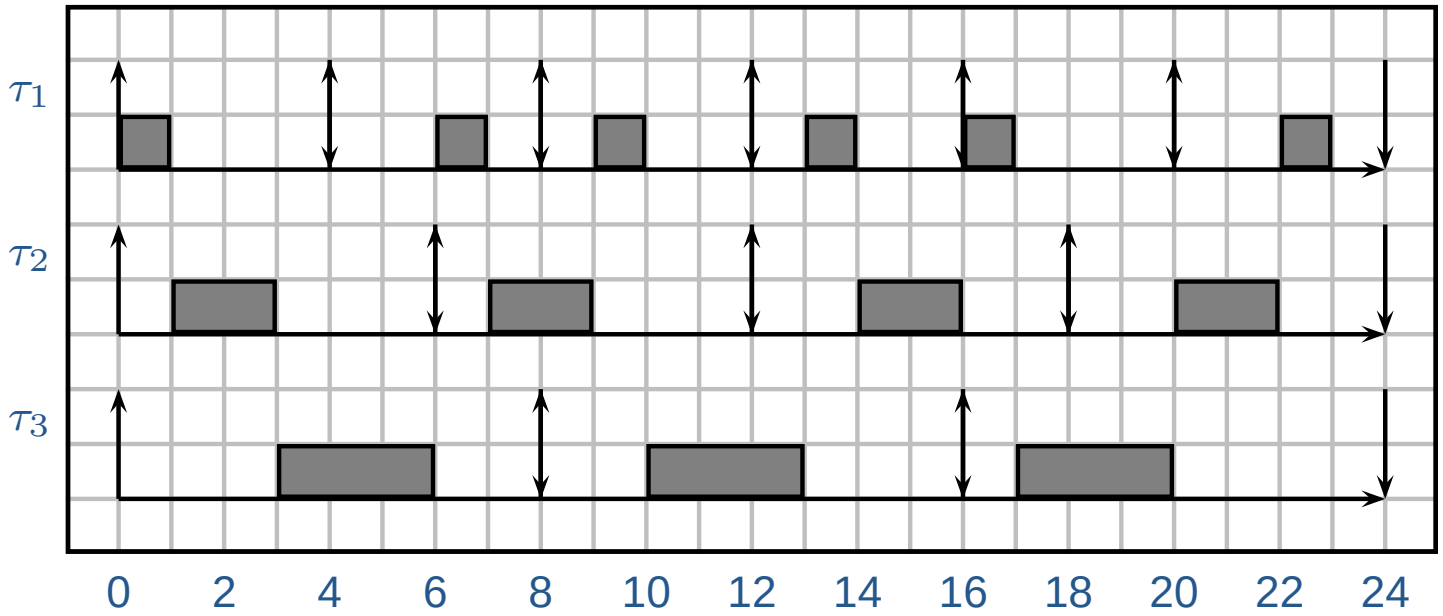
- We schedule the following task set with FP (RM priority assignment).
- $\tau_1 = (1, 4)$, $\tau_2 = (2, 6)$, $\tau_4 = (3, 8)$.
- $U = \frac{1}{4} + \frac{2}{6} + \frac{3}{8} = \frac{23}{24}$
- The utilization is greter than the bound: there is a deadline miss!



- Observe that at time 6, even if the deadline of task τ_3 is very close, the scheduler decides to schedule task τ_2. This is the main reason why τ_3 misses its deadline!

Example: scheduling with EDF

- Now we schedule the same task set with EDF.
- $\tau_1 = (1, 4)$, $\tau_2 = (2, 6)$, $\tau_4 = (3, 8)$.
- $U = \frac{1}{4} + \frac{2}{6} + \frac{3}{8} = \frac{23}{24}$
- Again, the utilization is very high. However, no deadline miss in the hyperperiod.



- Observe that at time 6, the problem does not appear, as the earliest deadline job (the one of τ_3) is executed.

Schedulability bound with EDF

- **Theorem** Given a task set of periodic or sporadic tasks, with relative deadlines equal to periods, the task set is schedulable by EDF if and only if

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$$

- **Corollary** EDF is an *optimal algorithm*, in the sense that if a task set if schedulable, then it is schedulable by EDF.
  - In fact, if $U > 1$ no algorithm can succesfully schedule the task set;
  - if $U \leq 1$, then the task set is schedulable by EDF (and maybe by other algorithms).
- In particular, EDF can schedule all task sets that can be scheduled by FP, but not vice versa.
- Notice also that offsets are not relevant!

Advantages of EDF over FP

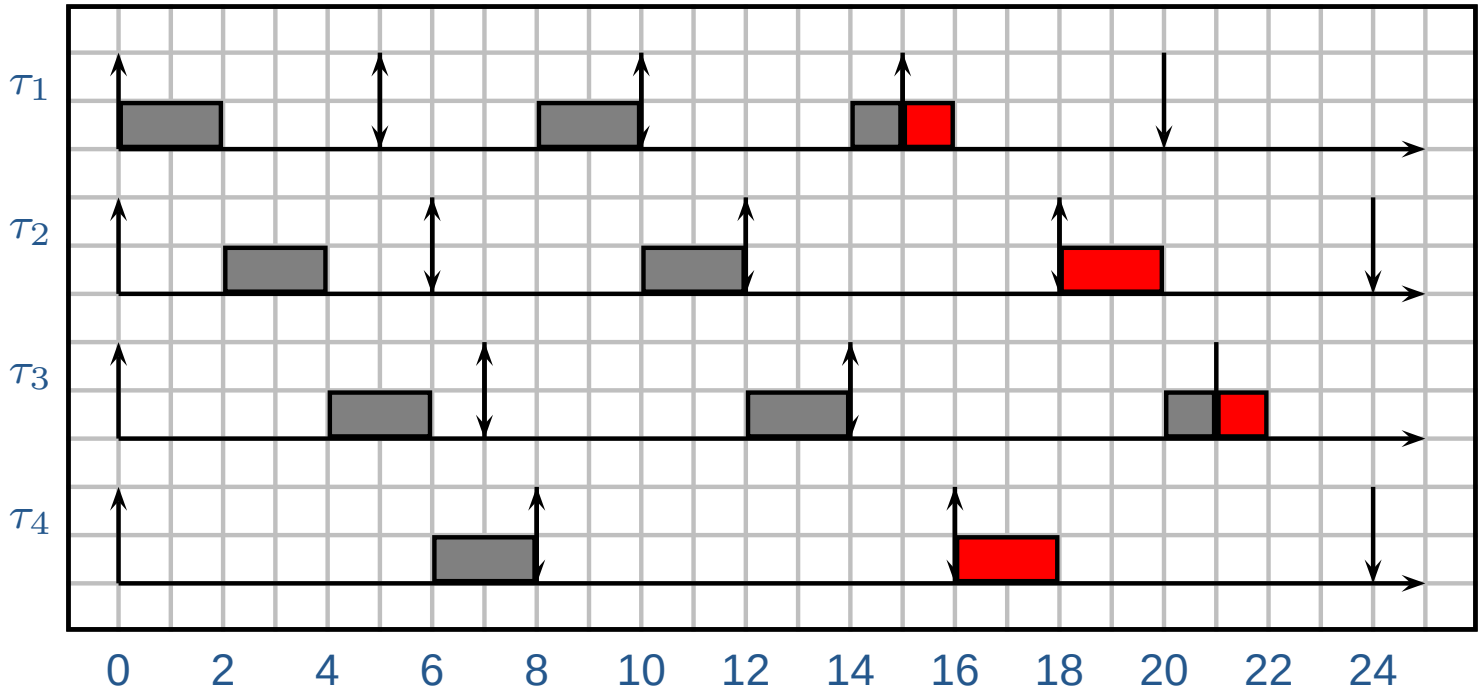
- There is not need to define priorities
  - Remember that in FP, in case of offsets, there is not an optimal priority assignment that is valid for all task sets
- In general, EDF has less context switches
  - In the previous example, you can try to count the number of context switches in the first interval of time: in particular, at time 4 there is no context switch in EDF, while there is one in FP.
- Optimality of EDF
  - We can fully utilize the processor, less idle times.

Disadvantages of EDF over FP

- EDF is not provided by any commercial RTOS, because of some disadvantage
- Less predictable
  - Looking back at the example, let's compare the response time of task τ_1: in FP is always constant and minimum; in EDF is variable.
- Less controllable
  - if we want to reduce the response time of a task, in FP is only sufficient to give him an higher priority; in EDF we cannot do anything;
  - We have less control over the execution
- More implementation overhead
  - FP can be implemented with a very low overhead even on very small hardware platforms (for example, by using only interrupts);
  - EDF instead requires more overhead to be implemented (we have to keep track of the absolute deadline in a long data structure);
  - There are method to implement the queueing operations in FP in O(1); in EDF, the queueing operations take O(log N), where N is the number of tasks.

Domino effect

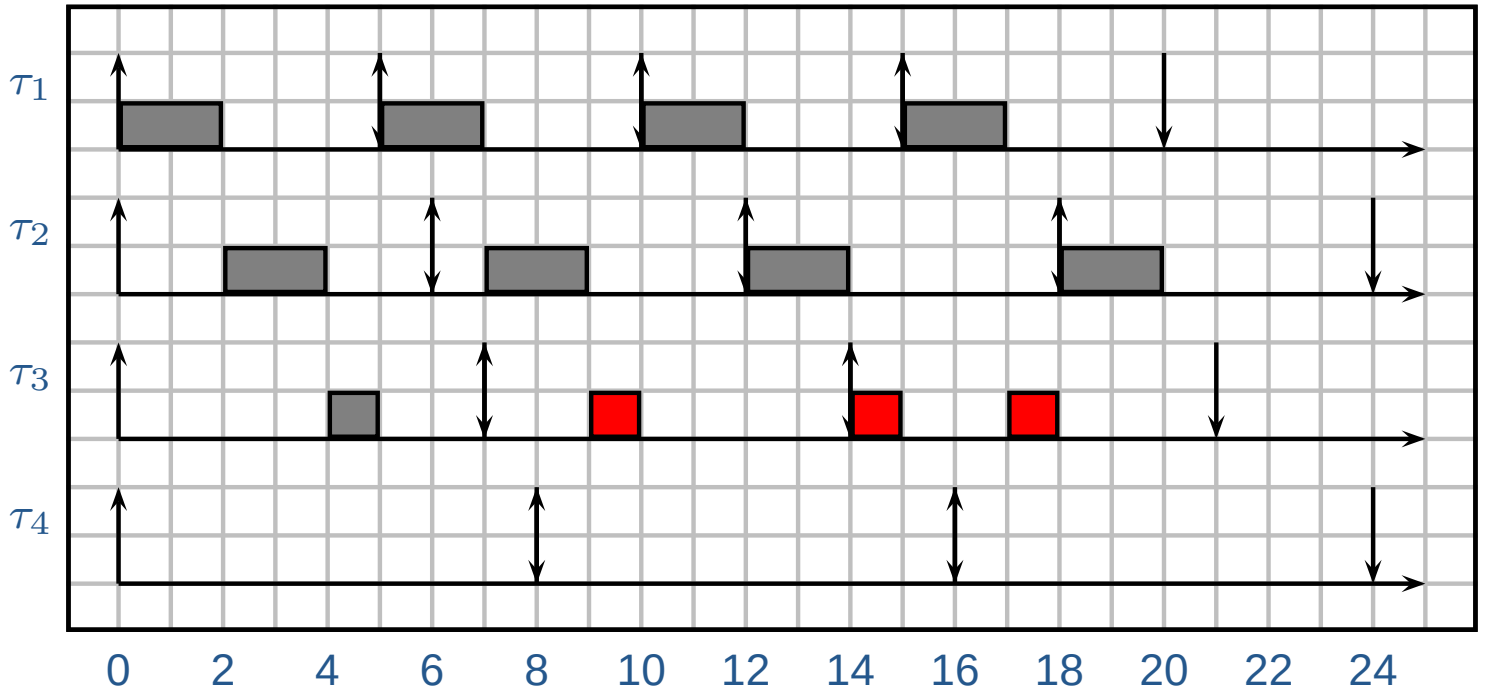
- In case of overhead ($U > 1$), we can have the *domino effect* with EDF: it means that all tasks miss their deadlines.
- An example of domino effect is the following;



- All tasks missed their deadline almost at the same time.

Domino effect: considerations

- FP is more predictable: only lower priority tasks miss their deadlines! In the previous example, if we use FP:



- As you can see, while τ_1 and τ_2 never miss their deadlines, τ_3 misses a lot of deadline, and τ_4 does not execute!
- However, it may happen that some task never executes in case of high overload, while EDF is more *fair* (all tasks are treated in the same way).

Response time computation

- Computing the response time in EDF is very difficult, and we will not present it in this course.
  - In FP, the response time of a task depends only on its computation time and on the interference of higher priority tasks
  - In EDF, it depends in the parameters of all tasks!
  - If all offset are 0, in FP the maximum response time is found in the first job of a task,
  - In EDF, the maximum response time is not found in the first job, but in a later job.

Generalization to deadlines different from period

- EDF is still optimal when relative deadlines are not equal to the periods.
- However, the schedulability analysis formula become more complex.
- If all relative deadlines are less than or equal to the periods, a first trivial (sufficient) test consist in substituting T_i with D_i:

$$U' = \sum_{i=1}^{N} \frac{C_i}{D_i} \leq 1$$

- In fact, if we consider each task as a sporadic task with interarrival time D_i instead of T_i, we are increasing the utilization, $U < U'$. If it is still less than 1, then the task set is schedulable. If it is larger than 1, then the task set may or may not be schedulable.

Demand bound analysis

- In the following slides, we present a general methodology for schedulability analysis of EDF scheduling;
- Let's start from the concept of *demand function*
- **Definition:** the demand function for a task τ_i is a function of an interval $[t_1, t_2]$ that gives the amount of computation time that *must* be completed in $[t_1, t_2]$ for τ_i to be schedulable:

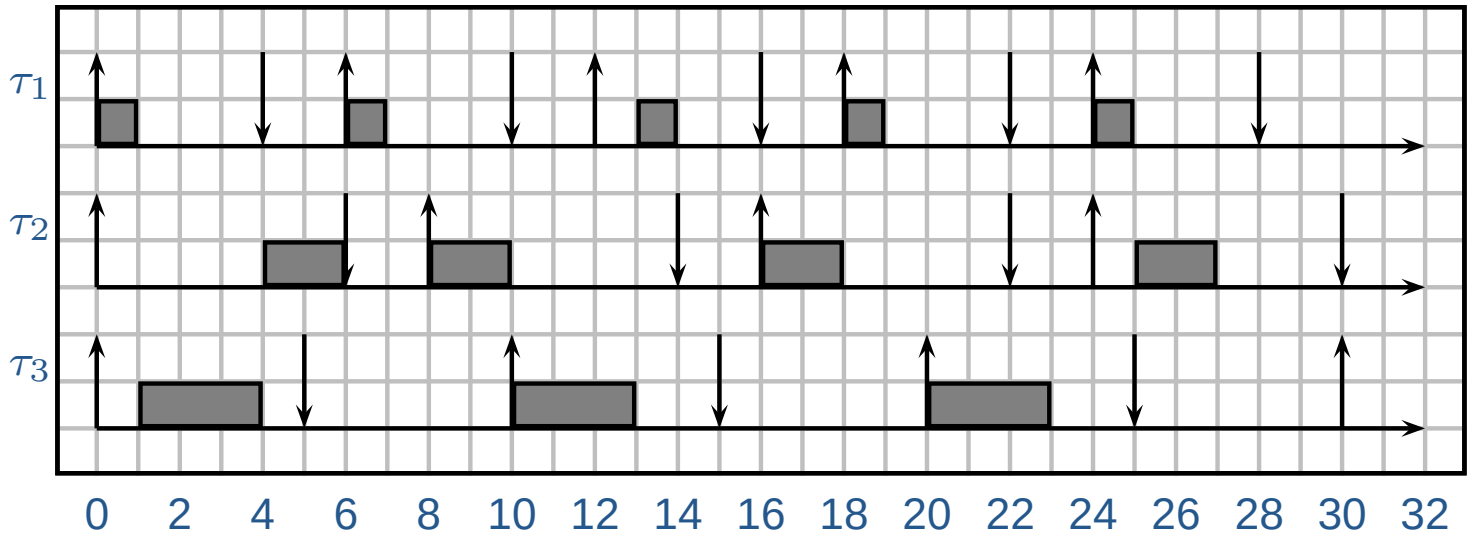
$$df_i(t_1, t_2) = \sum_{\substack{a_{ij} \geq t_1 \\ d_{ij} \leq t_2}} c_{ij}$$

- For the entire task set:

$$df(t_1, t_2) = \sum_{i=0}^{N} df_i(t_1, t_2)$$

Example of demand function

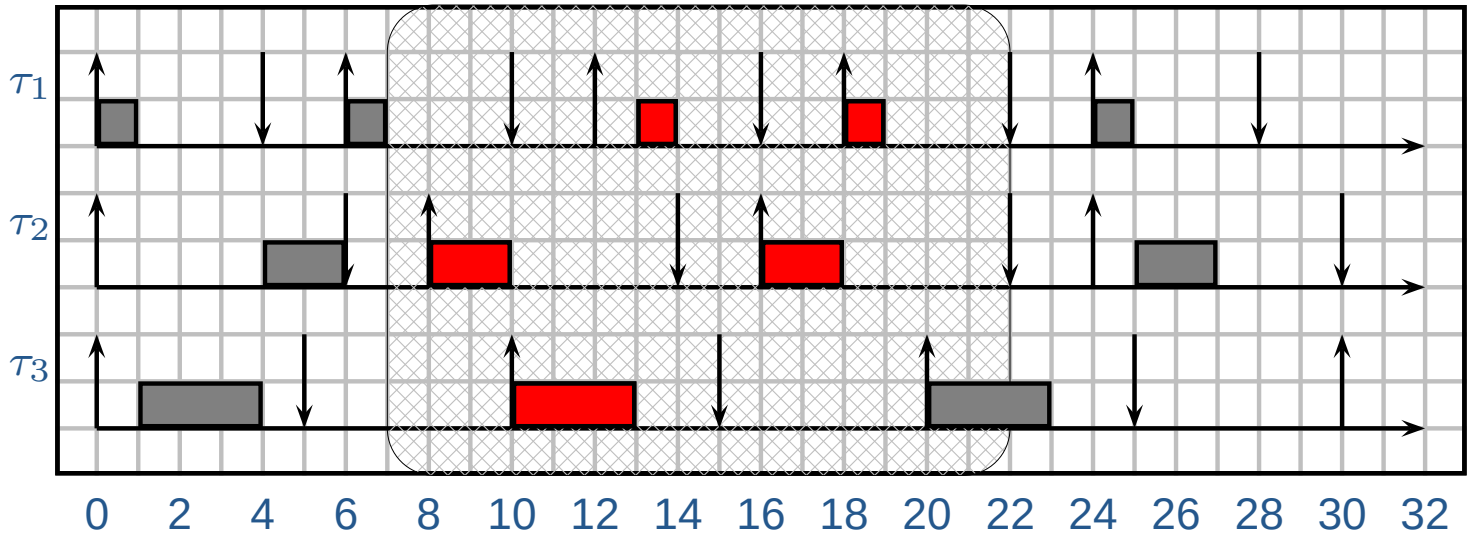
- $\tau_1 = (1, 4, 6), \tau_2 = (2, 6, 8), \tau_3 = (3, 5, 10)$



- Let's compute $df()$ in certain intervals;

Example of demand function

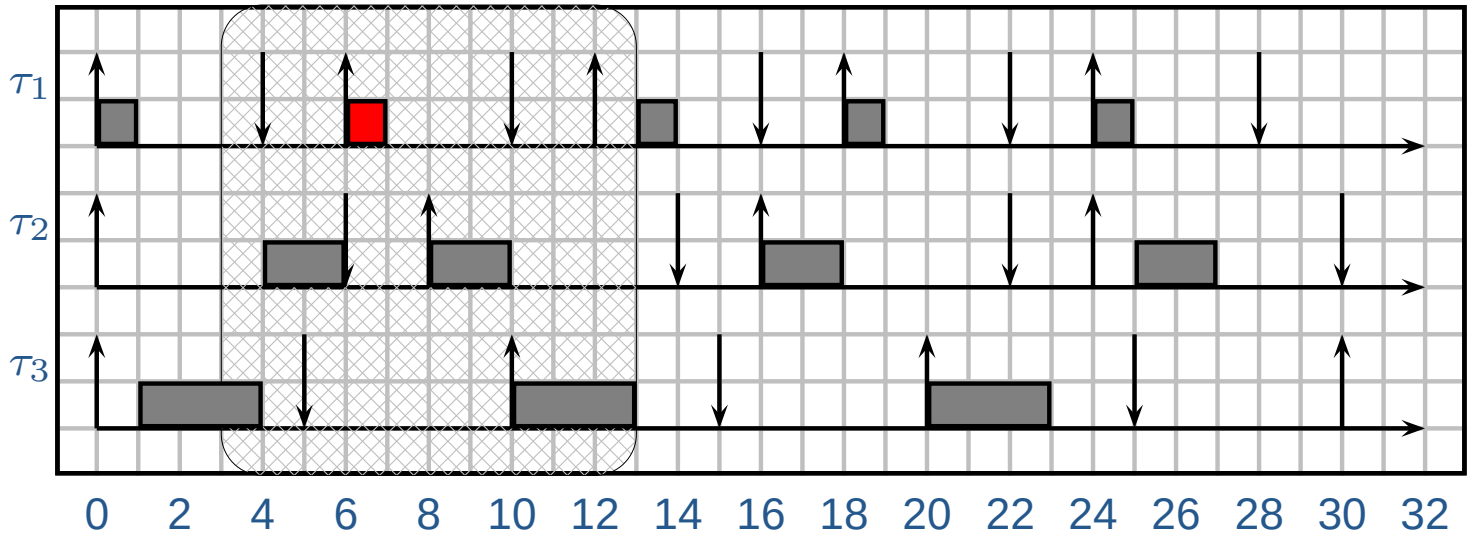
- $\tau_1 = (1, 4, 6), \tau_2 = (2, 6, 8), \tau_3 = (3, 5, 10)$



- Let's compute $df()$ in certain intervals;
- $df(7, 22) = 2 \cdot C_1 + 2 \cdot C_2 + 1 \cdot C_3 = 9$;

Example of demand function

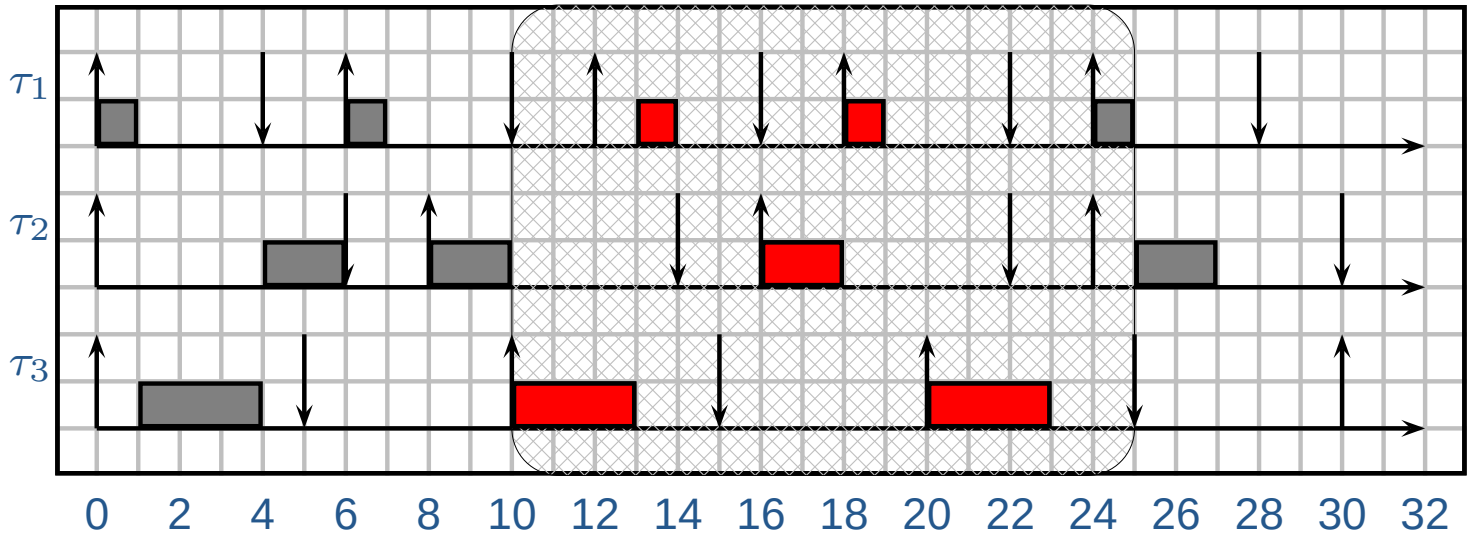
- $\tau_1 = (1, 4, 6), \tau_2 = (2, 6, 8), \tau_3 = (3, 5, 10)$



- Let's compute $df()$ in certain intervals;
- $df(7, 22) = 2 \cdot C_1 + 2 \cdot C_2 + 1 \cdot C_3 = 9$;
- $df(3, 13) = 1 \cdot C_1 = 1$;

Example of demand function

- $\tau_1 = (1, 4, 6), \tau_2 = (2, 6, 8), \tau_3 = (3, 5, 10)$



- Let's compute $df()$ in certain intervals;
- $df(7, 22) = 2 \cdot C_1 + 2 \cdot C_2 + 1 \cdot C_3 = 9$;
- $df(3, 13) = 1 \cdot C_1 = 1$;
- $df(10, 25) = 2 \cdot C_1 + 1 \cdot C_2 + 2 \cdot C_3 = 7$;

Condition for schedulability

- **Theorem:** A task set is schedulable under EDF if and only if:

$$\forall t_1, t_2 > t_1 \quad df(t_1, t_2) \leq t_2 - t_1$$

  - In the previous example:
    - $\rightarrow$ $df(7, 22) = 9 < 15 \rightarrow$ OK;
    - $\rightarrow$ $df(3, 13) = 1 < 9 \rightarrow$ OK;
    - $\rightarrow$ $df(10, 25) = 7 < 15 \rightarrow$ OK;
    - $\rightarrow$ $\ldots$
  - We should check for an infinite number of intervals!
  - We need a a way to simplify the previous condition.

Demand bound function

- **Theorem:** For a set of synchronous periodic tasks (i.e. with no offset),

$$\forall t_1, t_2 > t_1 \quad df(t_1, t_2) \geq df(0, t_2 - t_1)$$

  - In plain words, the worst case demand is found for intervals starting at 0.

- **Definition:** Demand Bound function:

$$dbf(L) = \max_t (df(t, t + L)) = df(0, L).$$

Condition on demand bound

- **Theorem:** A set of synchronous periodic tasks is schedulable by EDF if and only if:

$$\forall L \quad dbf(L) < L;$$

  - However, the number of intervals is still infinite. We need a way to limit the number of intervals to a finite number.

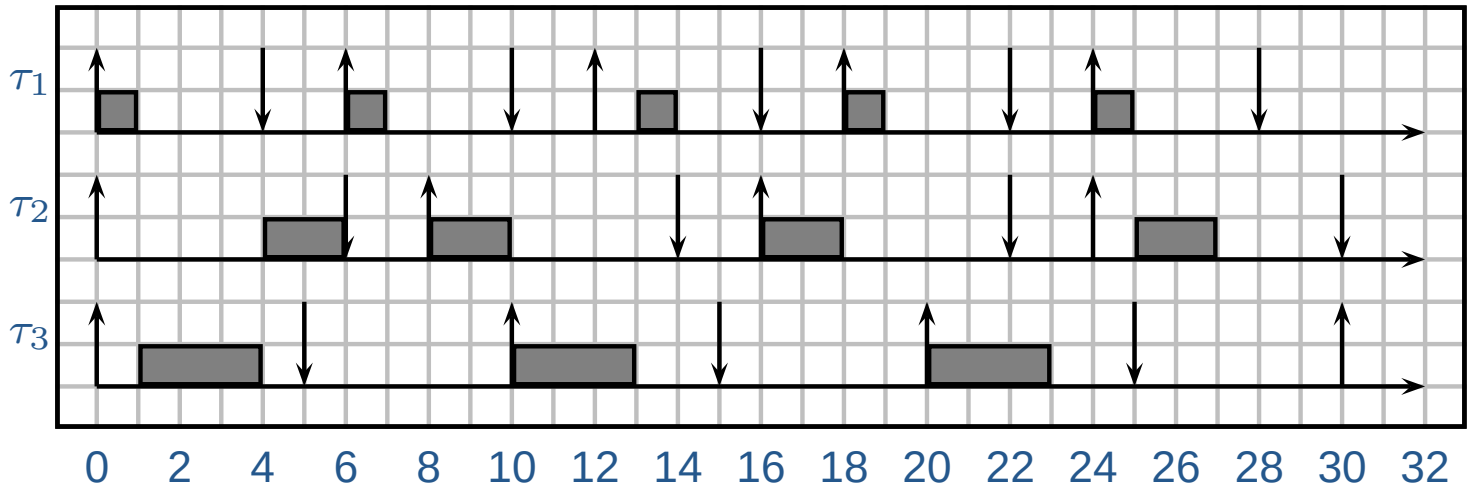
- **Theorem:** A set if synchronous periodic tasks, with $U < 1$, is schedulable by EDF if and only if:

$$\forall L \leq L^* \quad dbf(L) \leq L$$

$$L^* = \frac{U}{1-U} \max(T_i - D_i)$$

Example of computation of the dbf

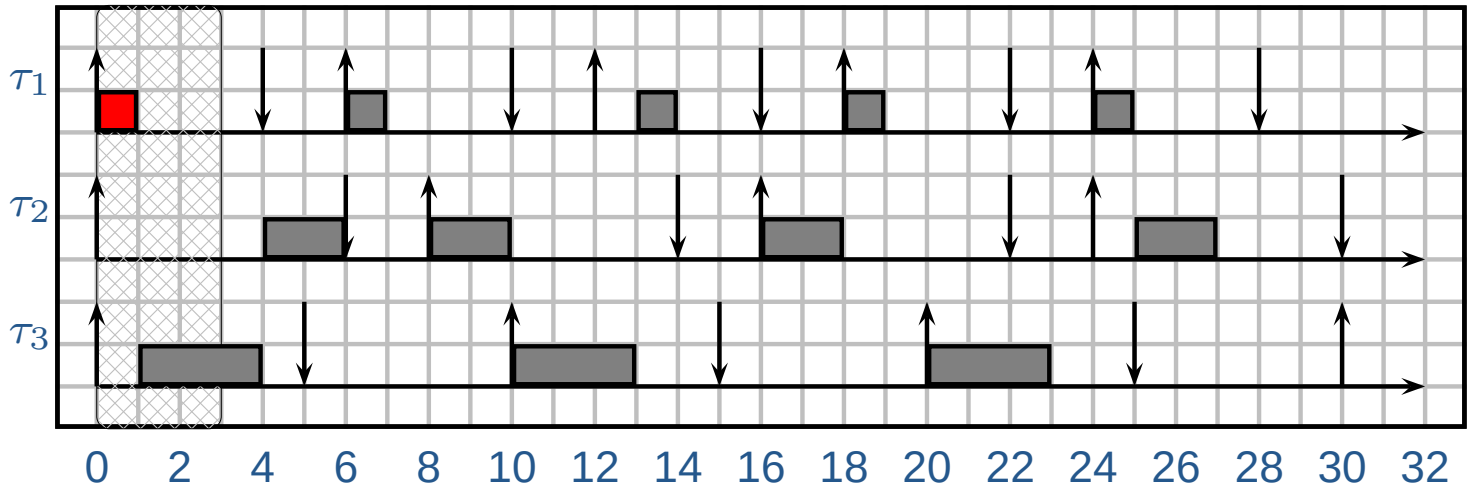
- $\tau_1 = (1, 4, 6)$, $\tau_2 = (2, 6, 8)$, $\tau_3 = (3, 5, 10)$
- $U = 1/6 + 1/4 + 3/10 = 0.7167$, $L^* = 12.64$.
- We must analyze all deadlines in $[0, 12]$, i.e. $(3, 5, 6, 10)$.



- Let's compute $dbf()$

Example of computation of the dbf

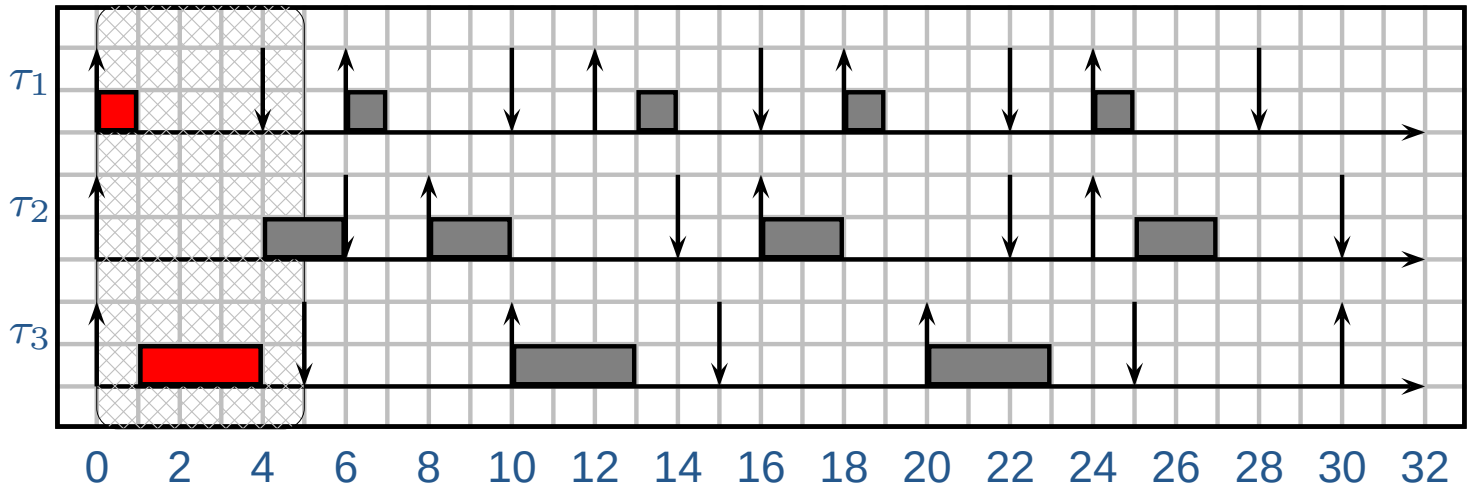
- $\tau_1 = (1, 4, 6)$, $\tau_2 = (2, 6, 8)$, $\tau_3 = (3, 5, 10)$
- $U = 1/6 + 1/4 + 3/10 = 0.7167$, $L^* = 12.64$.
- We must analyze all deadlines in $[0, 12]$, i.e. $(3, 5, 6, 10)$.



- Let's compute $dbf()$
- $df(0, 4) = C_1 = 1 < 4$;

Example of computation of the dbf

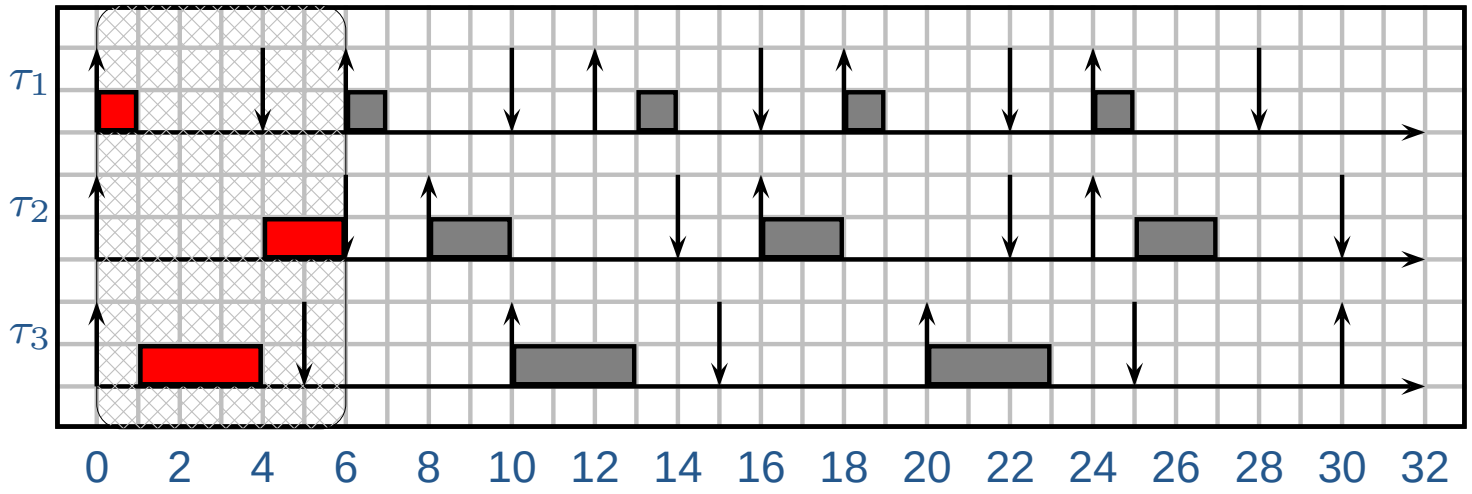
- $\tau_1 = (1, 4, 6)$, $\tau_2 = (2, 6, 8)$, $\tau_3 = (3, 5, 10)$
- $U = 1/6 + 1/4 + 3/10 = 0.7167$, $L^* = 12.64$.
- We must analyze all deadlines in $[0, 12]$, i.e. $(3, 5, 6, 10)$.



- Let's compute $dbf()$
- $df(0, 4) = C_1 = 1 < 4$;
- $df(0, 5) = C_1 + C_3 = 4 < 5$;

Example of computation of the dbf

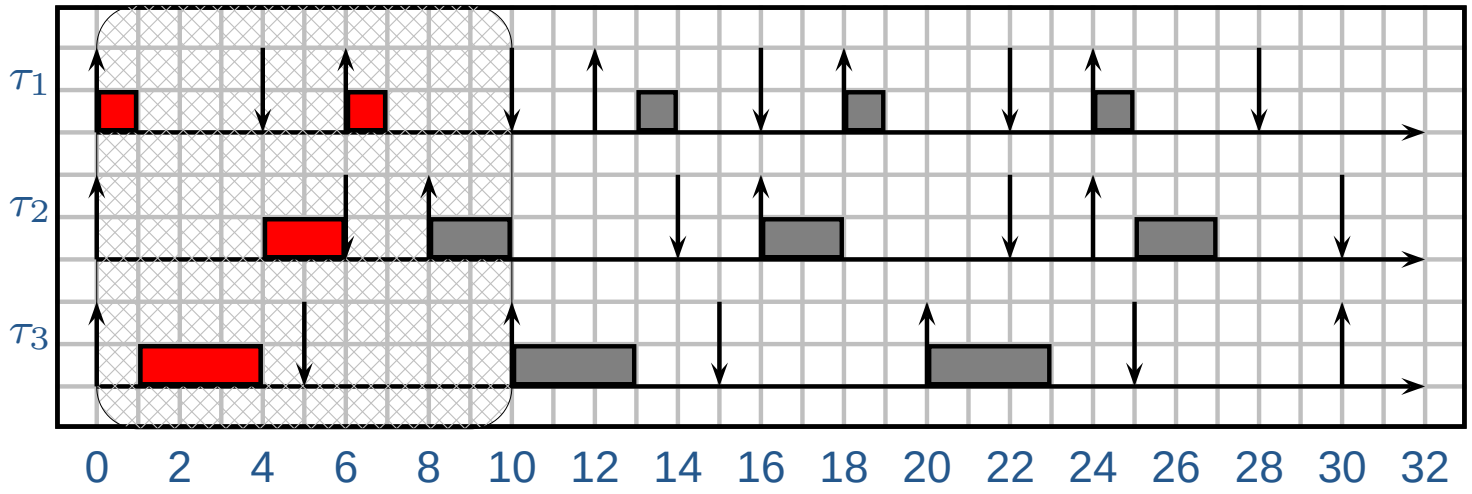
- $\tau_1 = (1, 4, 6)$, $\tau_2 = (2, 6, 8)$, $\tau_3 = (3, 5, 10)$
- $U = 1/6 + 1/4 + 3/10 = 0.7167$, $L^* = 12.64$.
- We must analyze all deadlines in $[0, 12]$, i.e. $(3, 5, 6, 10)$.



- Let's compute $dbf()$
- $df(0, 4) = C_1 = 1 < 4$;
- $df(0, 5) = C_1 + C_3 = 4 < 5$;
- $df(0, 6) = C_1 + C_2 + C_3 = 6 \leq 6$;

Example of computation of the dbf

- $\tau_1 = (1, 4, 6)$, $\tau_2 = (2, 6, 8)$, $\tau_3 = (3, 5, 10)$
- $U = 1/6 + 1/4 + 3/10 = 0.7167$, $L^* = 12.64$.
- We must analyze all deadlines in $[0, 12]$, i.e. $(3, 5, 6, 10)$.



- Let's compute $dbf()$
- $df(0, 4) = C_1 = 1 < 4$;
- $df(0, 5) = C_1 + C_3 = 4 < 5$;
- $df(0, 6) = C_1 + C_2 + C_3 = 6 \leq 6$;
- $df(0, 10) = 2C_1 + C_2 + C_3 = 7 \leq 10$;

Algorithm

- Of course, it should not be necessary to draw the schedule to see if the system is schedulable or not.
- First of all, we need a formula for the dbf:

$$dbf(L) = \sum_{i=1}^{N} \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i$$

- The algorithm works as follows:
  - We list all deadlines of all tasks until L^*.
  - Then, we compute the dbf for each deadline and verify the condition.

The previous example

τ_1	4	10
τ_2	6	
τ_3	5	

L	4	5	6	10
dbf	1	4	6	7

- Since, for all $L < L^*$ we have $dbf(L) \leq L$, then the task set is schedulable.

Another example

	C_i	D_i	T_i
τ_1	1	2	4
τ_2	2	4	5
τ_3	4.5	8	15

- $U = 0.9$; $L^* = 9 * 7 = 63$;
- hint: if L^* is too large, we can stop at the first idle time.
- The first idle time can be found with the following recursive equations:

$$W(0) = \sum_{i=1}^{N} C_i$$

$$W(k) = \sum_{i=1}^{N} \left\lceil \frac{W(k-1)}{T_i} \right\rceil C_i$$

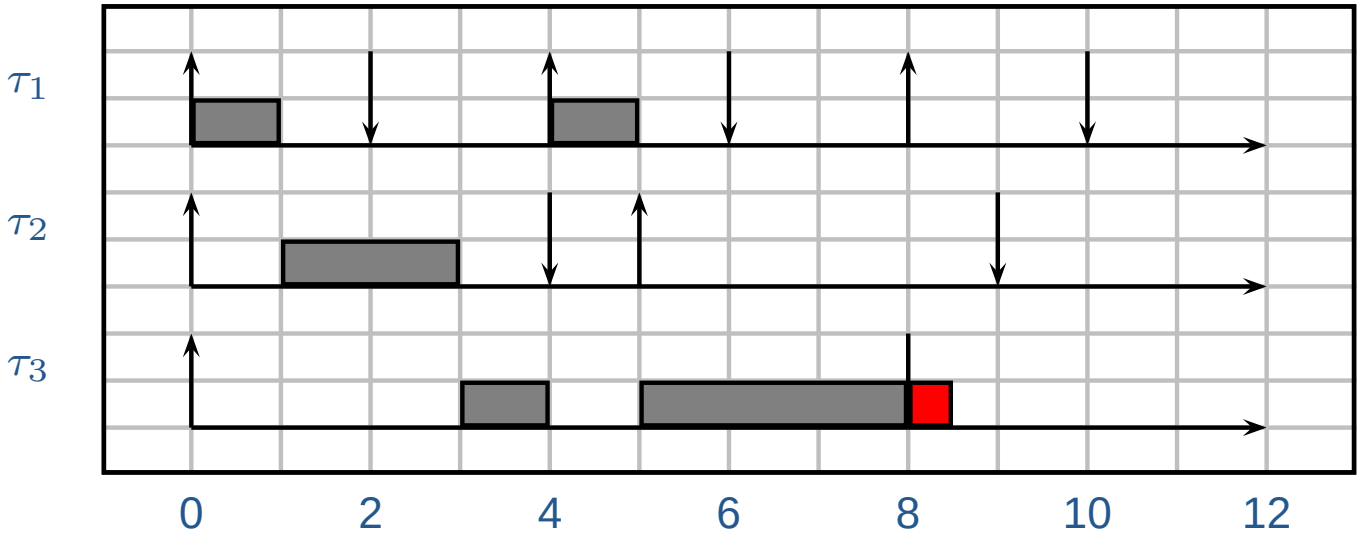
Example

τ_1	2	6	10	14
τ_2	4	9	14	
τ_3	8			

t	2	4	6	8	9	10	14
dbf	1	3	4	8.5			

- The task set is not schedulable! Deadline miss at 8.

In the schedule...



EDF and Shared resources

Synchronization protocols

- Both the Priority inheritance Protocol and thr Stack Resource Policy can be used under EDF without any modification.
- Let’s first consider PI.
  - When a higher priority *job* is blocked by a lower priority job on a shared mutex sempahore, then the lower priority job *inherits* the priority of the blocked job.
  - In EDF, the priority of a job is inversely proportional to its absloute deadline.
  - Here, you should substiture *higher priority job* with *job with an early deadline* and *inherits the priority* with *inherits the absolute deadline*.

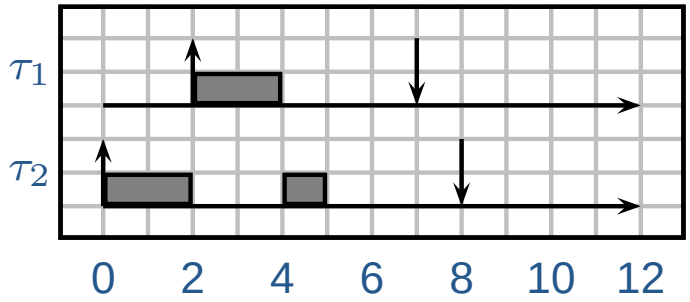
Preemption levels

- To compute the blocking time, we must first order the tasks based on their *preemption levels*:
- **Definition:** Every task τ_i is assigned a preemption level π_i such that it can preempt a task τ_j if and only if $\pi_i > \pi_j$.
  - In fixed priority, the preemption level is the same as the priority.
  - In EDF, the preemption level is defined as $\pi_i = \frac{1}{D_i}$.
  - In fact, as the following figures shows, if τ_i can preempt τ_j, then the following two conditions must hold:
    - τ_i arrives after τ_j has started to execute and hence $a_i > a_j$,
    - the absolute deadline of τ_i is shorter than the absolute deadline of τ_j ($d_i \leq d_j$).
    - It follows that

$$\begin{aligned} d_i &= a_i + D_i \leq d_j = a_j + D_j \Rightarrow \\ D_i - D_j &\leq a_j - a_i < 0 \Rightarrow \\ D_i &< D_j \Rightarrow \\ \pi_i &> \pi_j \end{aligned}$$

Preemption levels

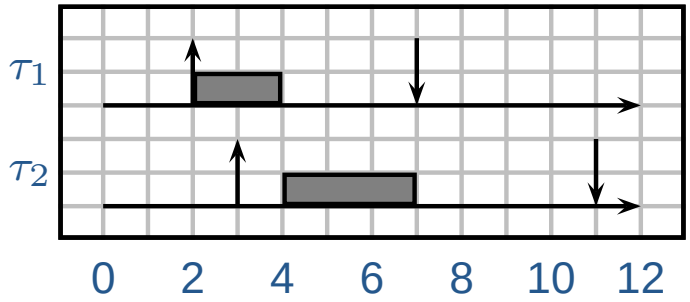
- With a graphical example:



- Notice that $\pi_1 > \pi_2$;
- In this case, τ_1 preempts τ_2.

Preemption levels

- With a graphical example:



- Notice that $\pi_1 > \pi_2$;
- τ_2 cannot preempt τ_1 (because its relative deadline is greater than τ_1).

Computing the blocking time

- To compute the blocking time for EDF + PI, we use the same algorithms as for FP + PI. In particular, the two fundamental theorems are still valid:
  - Each task can be blocked only once per each resource, and only for the length of one critical section per each task.
- In case on non-nested critical sections, build a *resource usage table*
  - At each row put a task, ordered by decreasing preemption levels
  - At each column, put a resource
  - In each cell, put the worst case duration ξ_{ij} of any critical section of task τ_i on resource S_j
- The algorithm for the blocking time for task τ_i is the same:
  - Select the rows below the i-th;
  - we must consider only those column on which it can be blocked (used by itself or by higher priority tasks)
  - Select the maximum sum of the $\xi_{k,j}$ with the limitation of at most one $\xi_{k,j}$ for each k and for each j.

Schedulability formula

- In case of relative deadlines equal to periods, we have:

$$\forall i = 1, \ldots, N \quad \sum_{j=1}^{i} \frac{C_j}{T_j} + \frac{B_i}{T_i} \leq 1$$

- In case of relative deadlines less than the periods:

$$\forall i = 1, \ldots, N \quad \forall L < L^*$$

$$\sum_{j=1}^{N} \left(\left\lfloor \frac{L - D_j}{T_j} \right\rfloor + 1 \right) C_j + B_i \leq L$$

$$L^* = \frac{U}{1 - U} \max_i (T_i - D_i)$$

Complete example

Here we analyze a complete example, from the parameters of the tasks, and from the resource usage table, we compute the B_is, and test schedulability.

	C_i	T_i	U_i	R_1	R_2	B_i
τ_1	2	10	.2	1	0	?
τ_2	5	15	.33	2	1	?
τ_3	4	20	.2	0	2	?
τ_4	9	45	.2	3	4	?

Complete example: blocking times

- Blocking time for τ_1:

	C_i	T_i	U_i	R_1	R_2	B_i
τ_1	2	10	.2	1	0	3
τ_2	5	15	.33	2	1	?
τ_3	4	20	.2	0	2	?
τ_4	9	45	.2	3	4	?

Complete example: blocking times

- Blocking time for τ_2:

	C_i	T_i	U_i	R_1	R_2	B_i
τ_1	2	10	.2	1	0	3
τ_2	5	15	.33	2	1	5
τ_3	4	20	.2	0	2	?
τ_4	9	45	.2	3	4	?

Complete example: blocking times

- Blocking time for τ_3:

	C_i	T_i	U_i	R_1	R_2	B_i
τ_1	2	10	.2	1	0	3
τ_2	5	15	.33	2	1	5
τ_3	4	20	.2	0	2	4
τ_4	9	45	.2	3	4	?

Complete example: blocking times

- Blocking time for τ_4:

	C_i	T_i	U_i	R_1	R_2	B_i
τ_1	2	10	.2	1	0	3
τ_2	5	15	.33	2	1	5
τ_3	4	20	.2	0	2	6
τ_4	9	45	.2	3	4	0

Complete Example: schedulability test

- General formula:

$$\forall i = 1, \ldots, 4 \quad \sum_{j=1}^{i} \frac{C_j}{T_j} + \frac{B_i}{T_i} \leq 1$$

- Task τ_1:

$$\frac{C_1}{T_1} + \frac{B_1}{T_1} = .2 + .3 = .5 \leq 1$$

Complete Example: schedulability test

- General formula:

$$\forall i = 1, \ldots, 4 \quad \sum_{j=1}^{i} \frac{C_j}{T_j} + \frac{B_i}{T_i} \leq 1$$

- Task τ_2:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{B_2}{T_2} = .5333 + .3333 = .8666 \leq 1$$

Complete Example: schedulability test

- General formula:

$$\forall i = 1, \ldots, 4 \quad \sum_{j=1}^{i} \frac{C_j}{T_j} + \frac{B_i}{T_i} \leq 1$$

- Task τ_3:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} + \frac{B_3}{T_3} = .2 + .333 + .2 + .2 = 0.9333 \leq 1$$

Complete Example: schedulability test

- General formula:

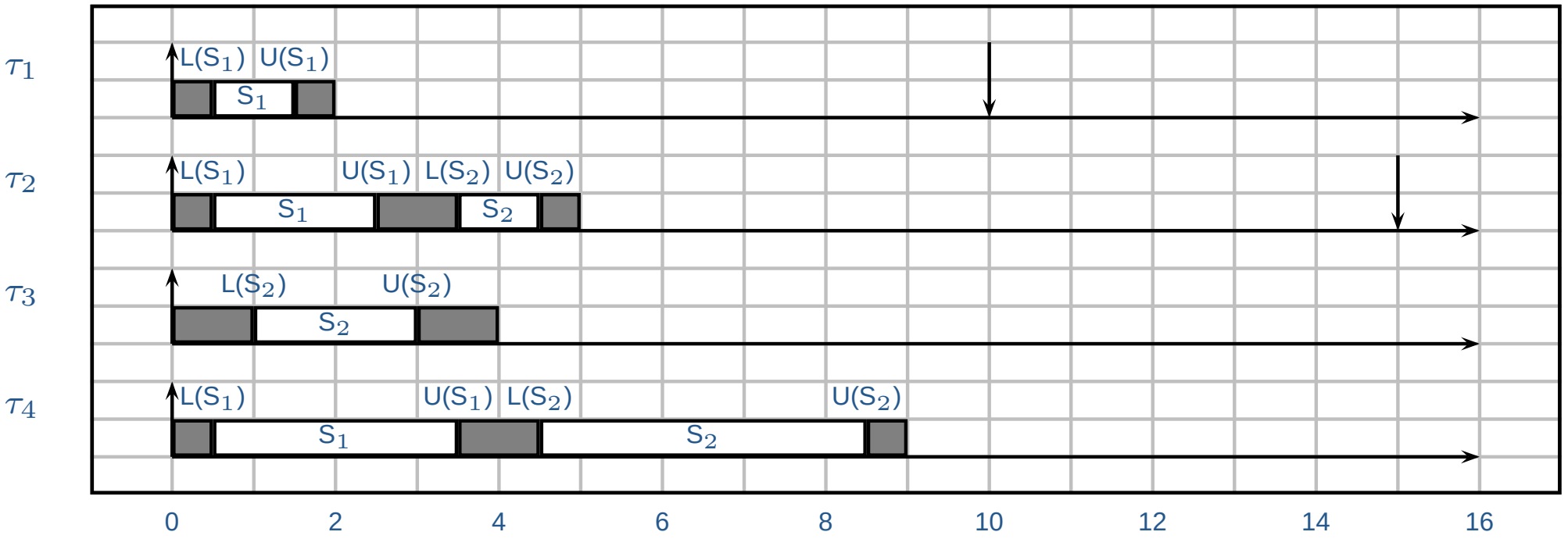
$$\forall i = 1, \ldots, 4 \quad \sum_{j=1}^{i} \frac{C_j}{T_j} + \frac{B_i}{T_i} \leq 1$$

- Task τ_4:

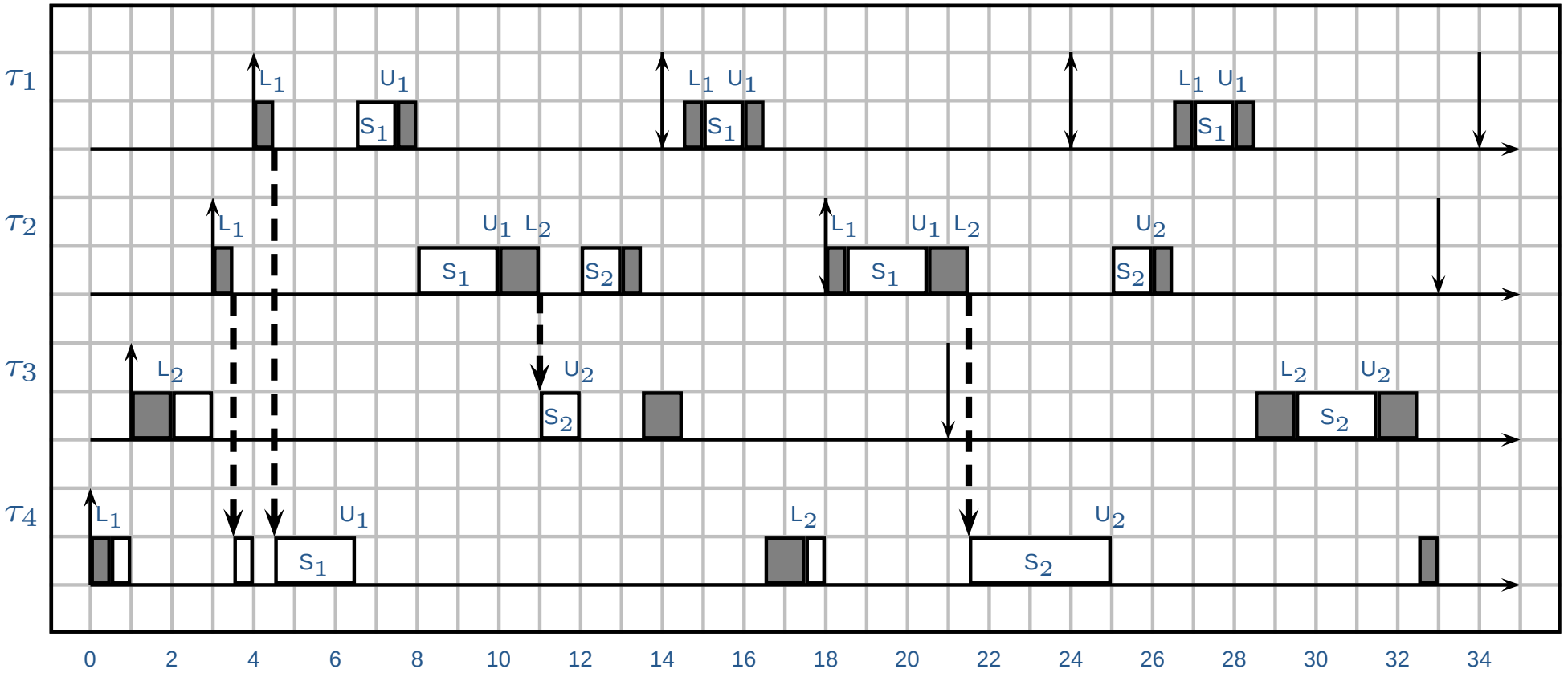
$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} + \frac{C_4}{T_4} + \frac{B_4}{T_4} = .2 + .3333 + .2 + .2 + 0 = .9333 \leq 1$$

Complete example: scheduling

- Now we do an example of possible schedule.
- We assume that the task access the resources as follows:



Complete example: schedule



- In the graph, $L_1 = \text{Lock}(S_1)$, $U_1 = \text{Unlock}(S_1)$, $L_2 = \text{Lock}(S_2)$, $U_2 = \text{Unlock}(S_2)$.
- The tasks start with an offset, because in the example we want to highlight the blocking times at the beginning.

Stack Resource Policy

- Once we have defined the preemption levels, it is easy to extend the stack resource polity to EDF.
- The main rule is the following:
  - The *ceiling* of a resource is defined as the highest preemption level among the ones of all tasks that access it;
  - At each instant, the *system ceiling* is the highest among the ceilings of the locked resources;
  - A task is not allowed to start executing until its deadline is the shortest one and its preemption level is strictly greater than the system ceiling;

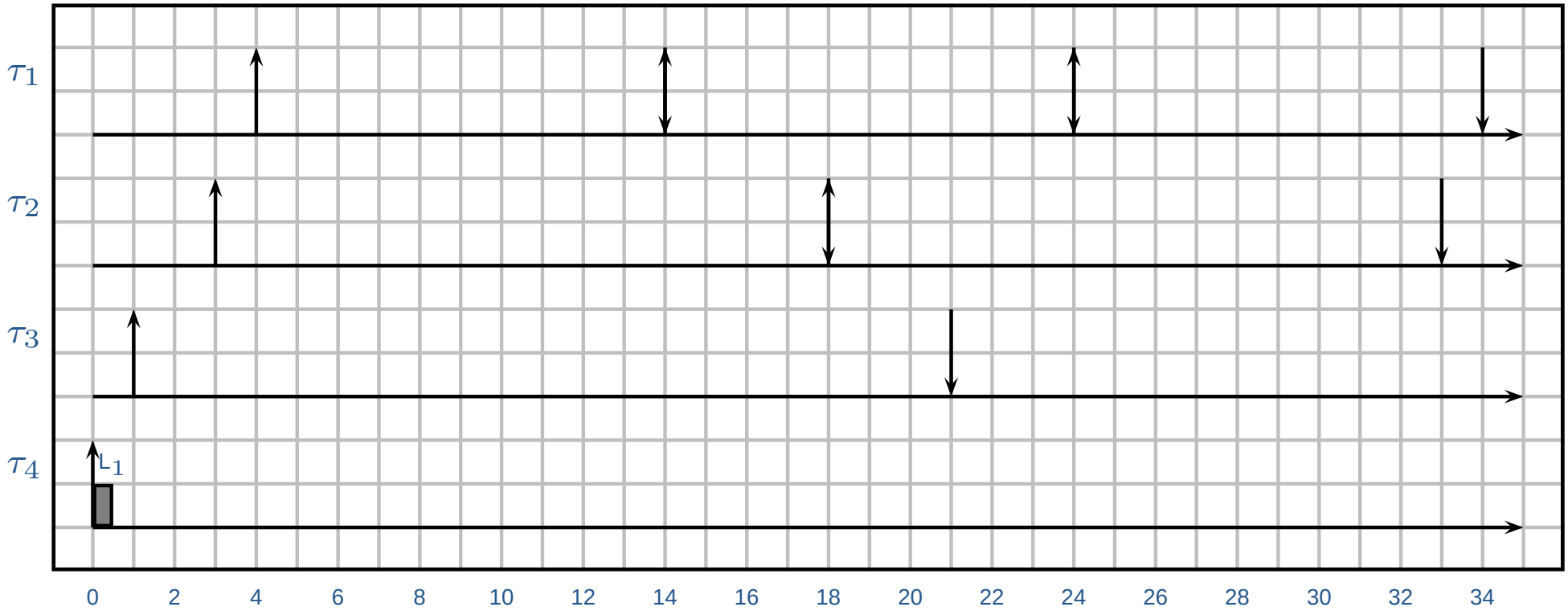
Complete Example

Now we analyze the previous example, assuming EDF+SRP.

	C_i	T_i	U_i	R_1	R_2	B_i
τ_1	2	10	.2	1	0	?
τ_2	5	15	.33	2	1	?
τ_3	4	20	.2	0	2	?
τ_4	9	45	.2	3	4	?

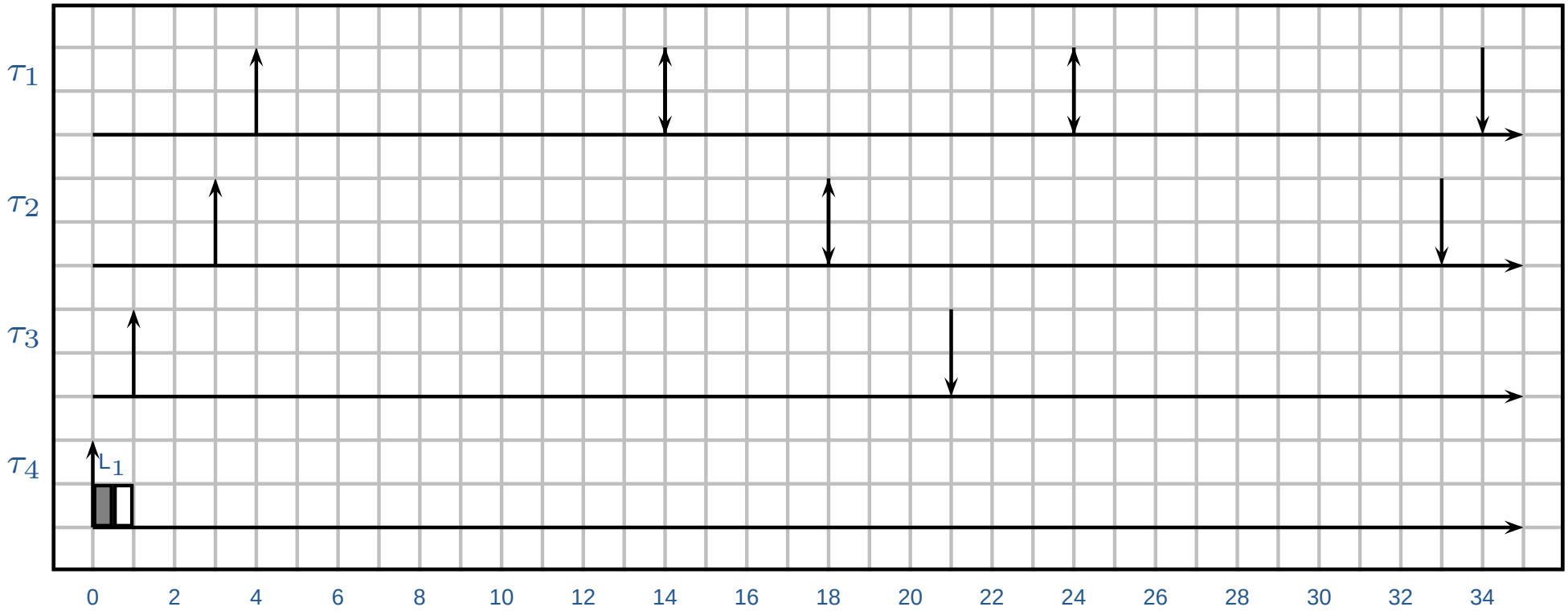
- Let us first assign the preemption levels.
  - The actual value of the preemption levels is not important, as long as they are assigned in the right order.
  - To make calcumations easy, we set $\pi_1 = 4$, $\pi_2 = 3$, $\pi_3 = 2$, $\pi_4 = 1$.
- Then the resource ceilings:
  - ceil$(R_1) = \pi_1 = 4$, ceil$(R_2) = \pi_2 = 3$.

Schedule



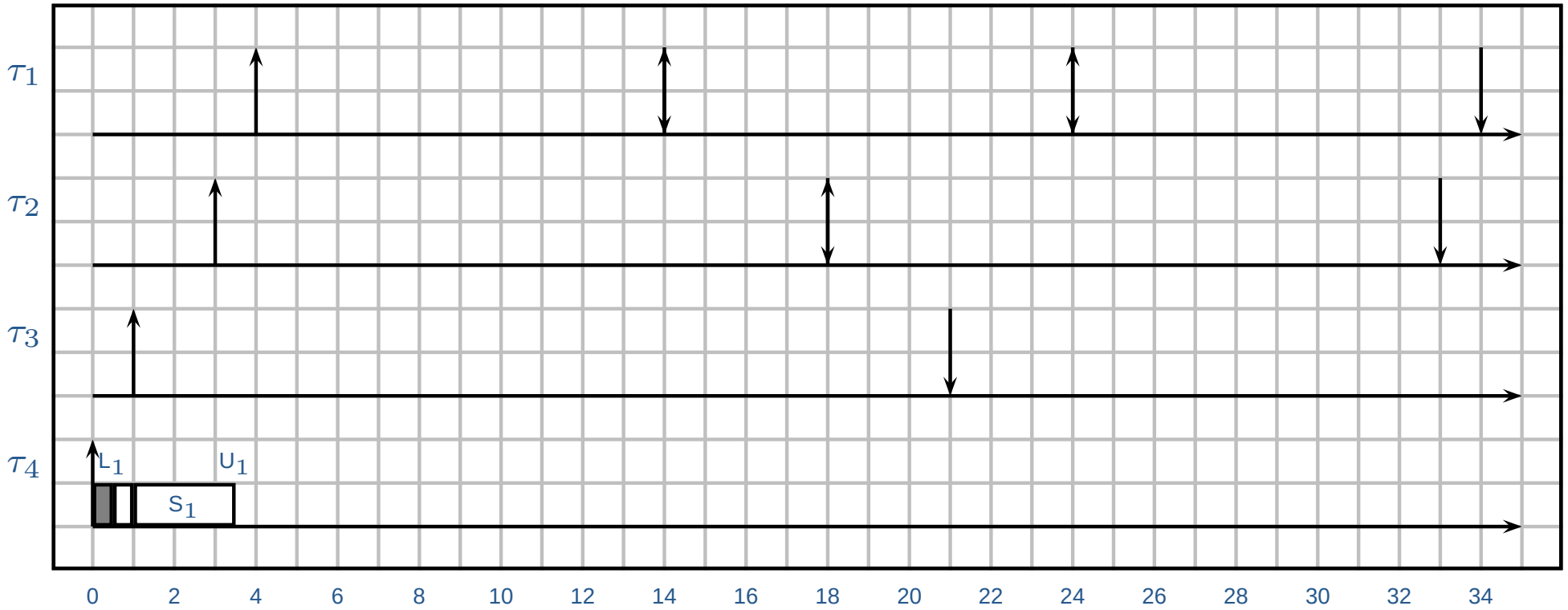
- At this point, the system ceiling is raised to π_1 (the ceiling of R_1).

Schedule



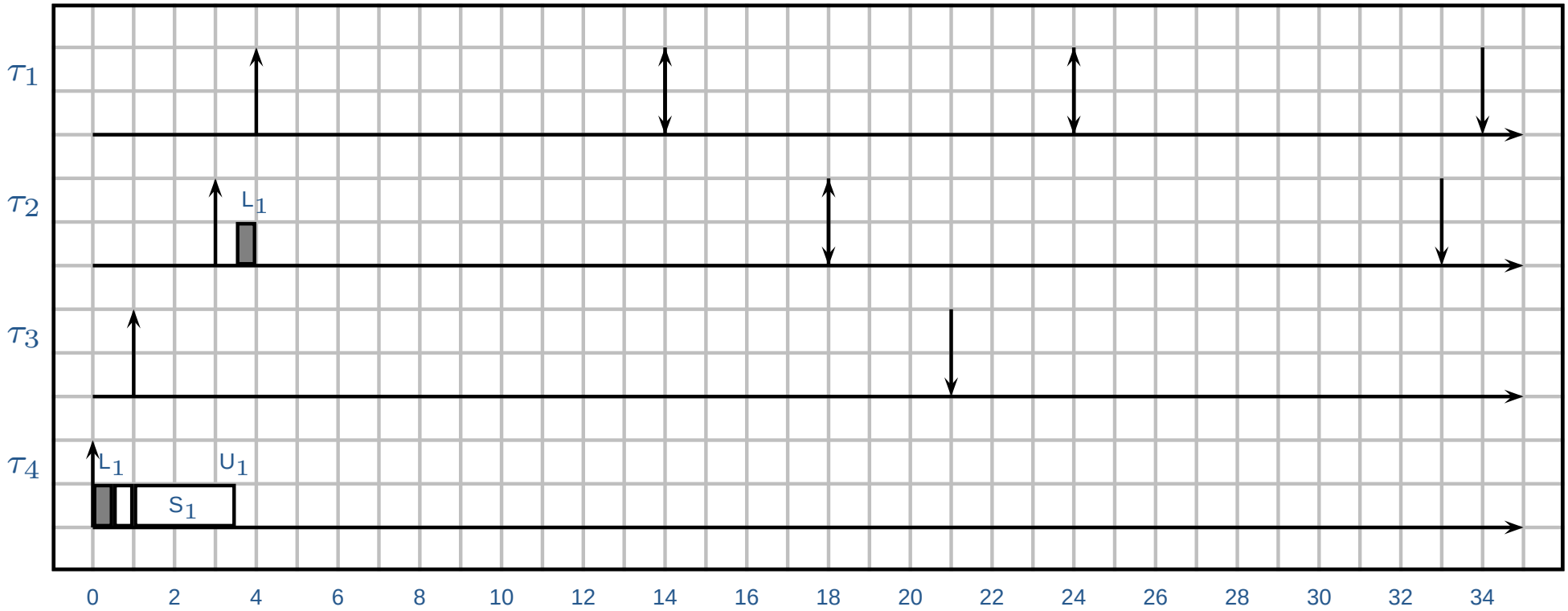
- At this point, the system ceiling is raised to π_1 (the ceiling of R_1). Task τ_3 cannot start executing, because $\pi_3 < \pi_1$. Same for τ_2.

Schedule



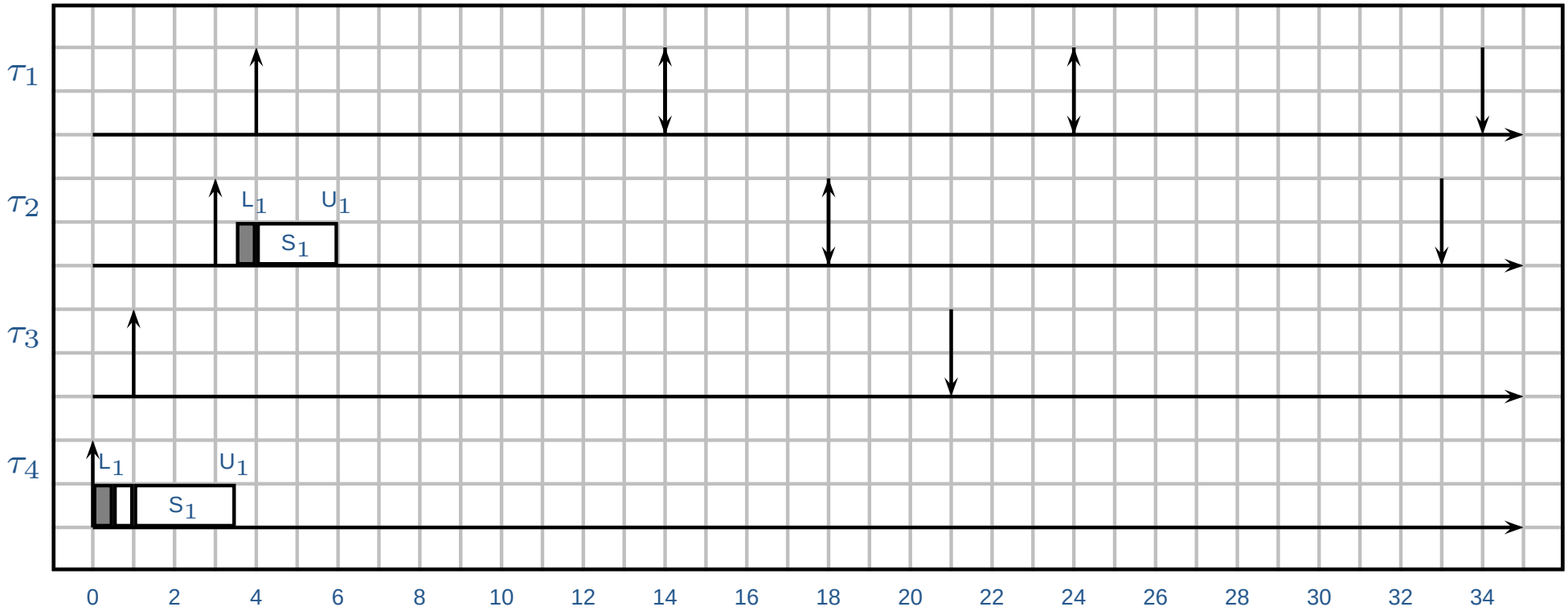
- At this point, the system ceiling is raised to π_1 (the ceiling of R_1). Task τ_3 cannot start executing, because $\pi_3 < \pi_1$. Same for τ_2.
- The system ceiling goes back to 0. Now τ_2 can start.

Schedule



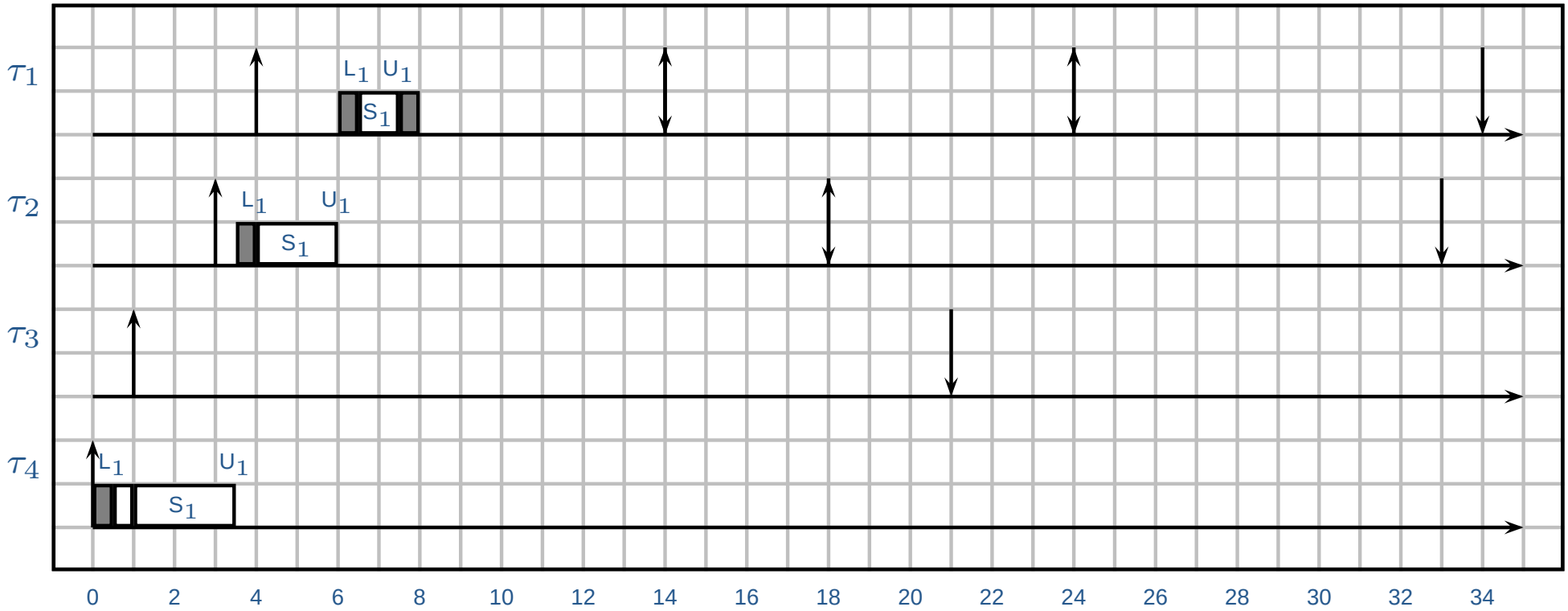
- At this point, the system ceiling is raised to π_1 (the ceiling of R_1). Task τ_3 cannot start executing, because $\pi_3 < \pi_1$. Same for τ_2.
- The system ceiling goes back to 0. Now τ_2 can start.
- in this example, we assume that τ_2 locks R_1 just before τ_1 arrives. Then, sys ceil = π_1 and τ_1 cannot preempt.

Schedule



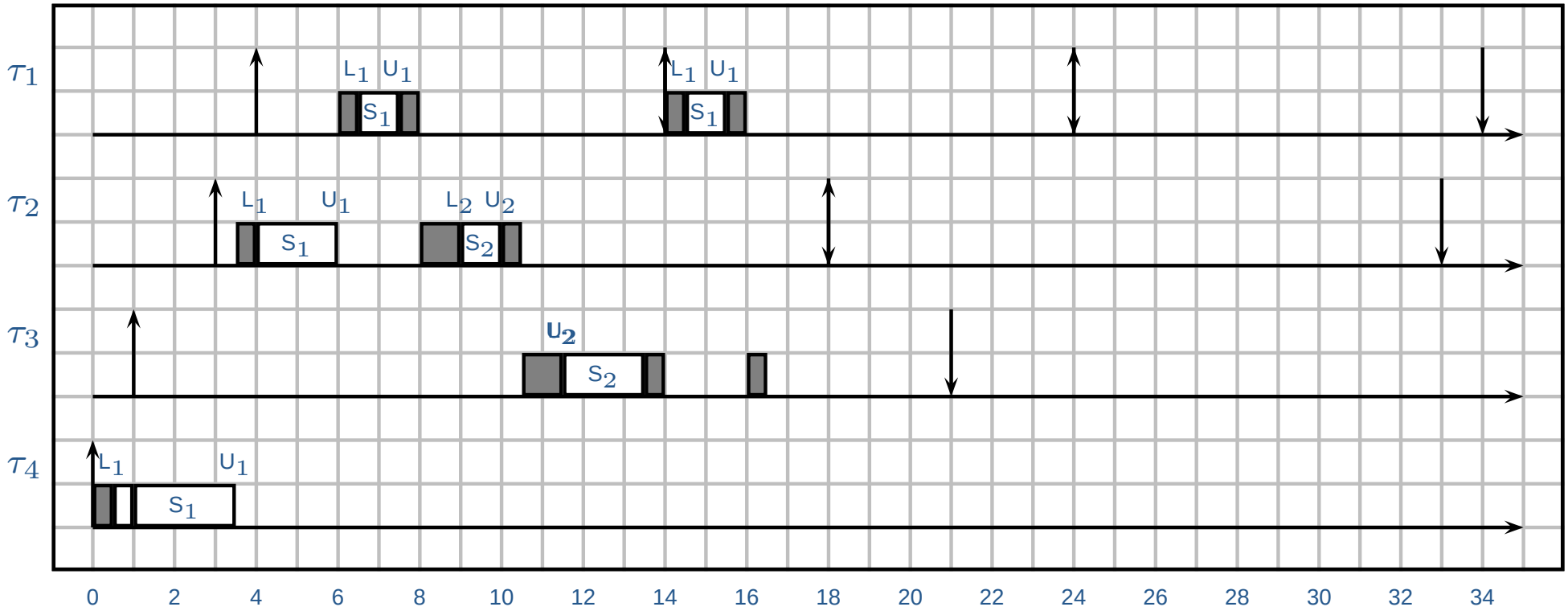
- At this point, the system ceiling is raised to π_1 (the ceiling of R_1). Task τ_3 cannot start executing, because $\pi_3 < \pi_1$. Same for τ_2.
- The system ceiling goes back to 0. Now τ_2 can start.
- in this example, we assume that τ_2 locks R_1 just before τ_1 arrives. Then, sys ceil = π_1 and τ_1 cannot preempt.

Schedule



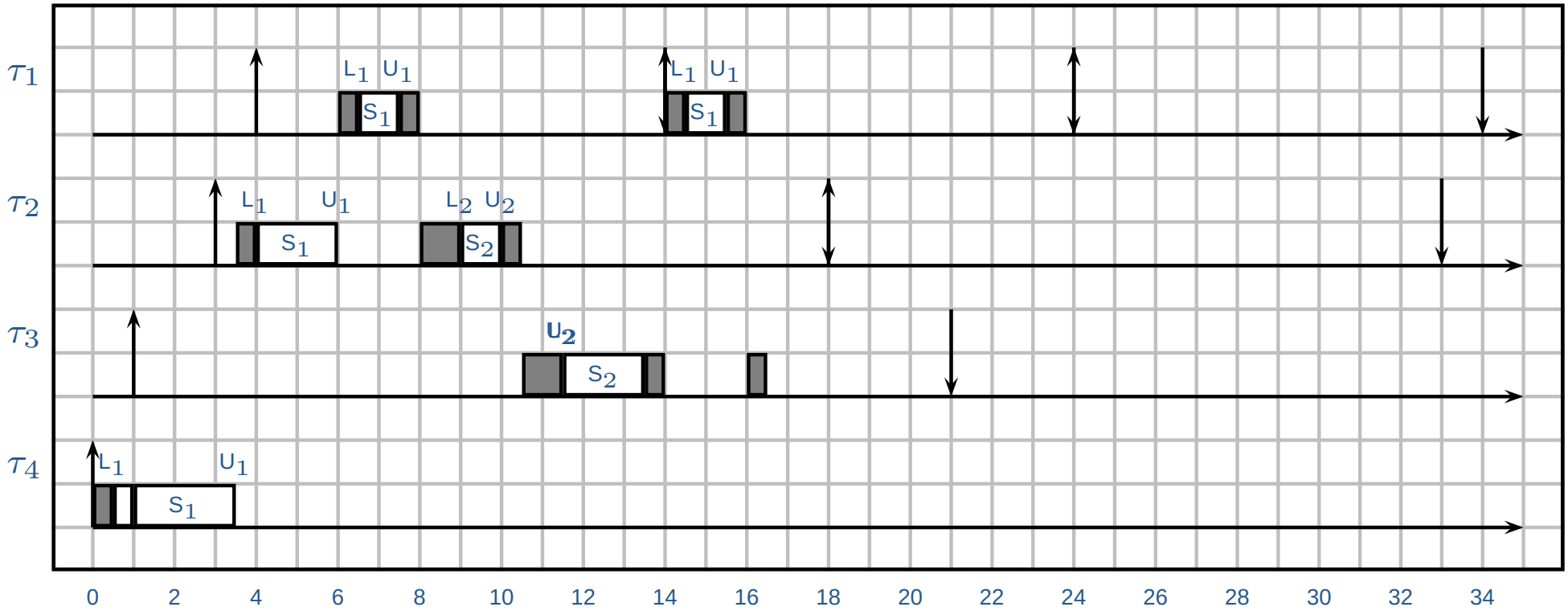
- At this point, the system ceiling is raised to π_1 (the ceiling of R_1). Task τ_3 cannot start executing, because $\pi_3 < \pi_1$. Same for τ_2.
- The system ceiling goes back to 0. Now τ_2 can start.
- in this example, we assume that τ_2 locks R_1 just before τ_1 arrives. Then, sys ceil = π_1 and τ_1 cannot preempt.

Schedule



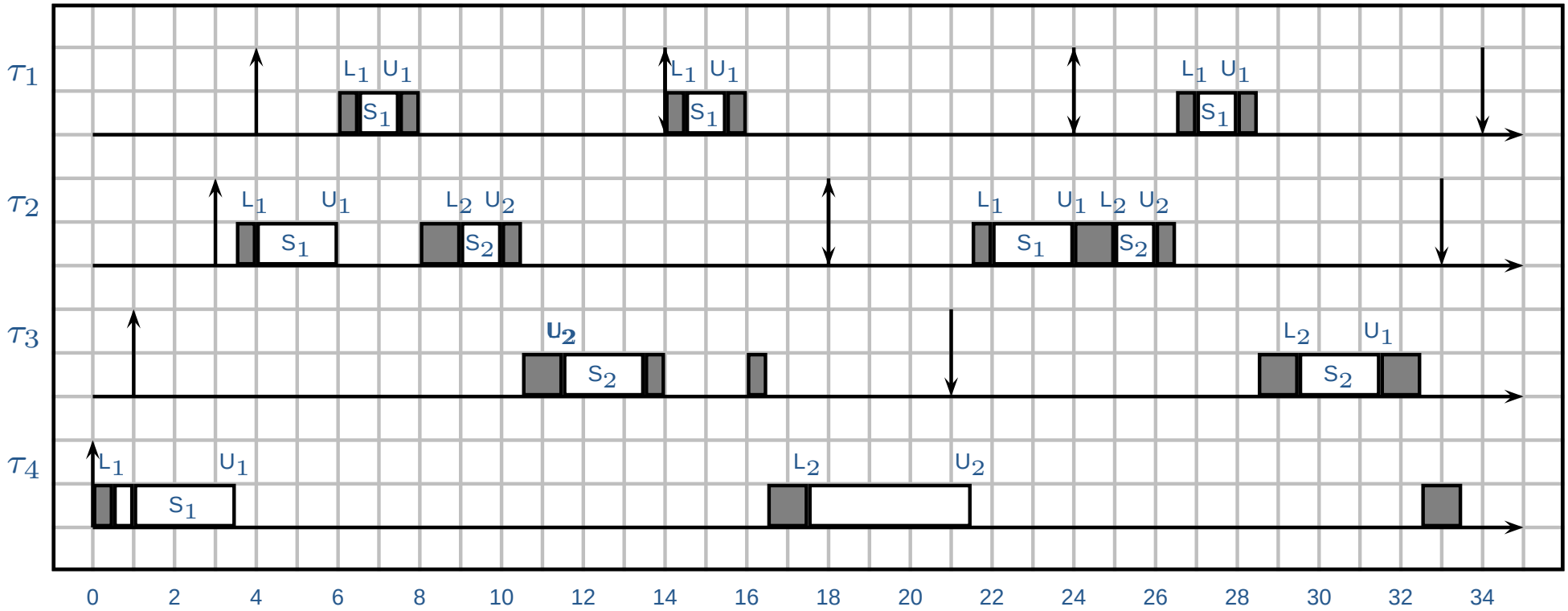
- At this point, the system ceiling is raised to π_1 (the ceiling of R_1). Task τ_3 cannot start executing, because $\pi_3 < \pi_1$. Same for τ_2.
- The system ceiling goes back to 0. Now τ_2 can start.
- in this example, we assume that τ_2 locks R_1 just before τ_1 arrives. Then, sys ceil = π_1 and τ_1 cannot preempt.

Schedule



- At this point, the system ceiling is raised to π_1 (the ceiling of R_1). Task τ_3 cannot start executing, because $\pi_3 < \pi_1$. Same for τ_2.
- The system ceiling goes back to 0. Now τ_2 can start.
- in this example, we assume that τ_2 locks R_1 just before τ_1 arrives. Then, sys ceil = π_1 and τ_1 cannot preempt.

Schedule



- At this point, the system ceiling is raised to π_1 (the ceiling of R_1). Task τ_3 cannot start executing, because $\pi_3 < \pi_1$. Same for τ_2.
- The system ceiling goes back to 0. Now τ_2 can start.
- in this example, we assume that τ_2 locks R_1 just before τ_1 arrives. Then, sys ceil = π_1 and τ_1 cannot preempt.

Blocking time computation

- The computation of the blocking time is the same as in the case of FP + SRP;
- The only difference is that, when the resource access table is built, tasks are ordered by decreasing preemption level, instead than by priority.

Complete example

Same example of before, but with SRP instead of PI.