Real-Time Scheduling

Real Time Operating Systems and Middleware

Luca Abeni
luca.abeni@unitn.it

Definitions

- Algorithm $\rightarrow$ logical procedure used to solve a problem
- Program $\rightarrow$ formal description of an algorithm, using a *programming language*
- Process $\rightarrow$ instance of a program (program in execution)
  - Program: static entity
  - Process: dynamic entity
- The term *task* is used to indicate a schedulable entity (either a process or a thread)
  - Thread $\rightarrow$ flow of execution
  - Process $\rightarrow$ flow of execution + private resources (address space, file table, etc...)

Scheduling

- Tasks do not run on bare hardware...
  - How can multiple tasks execute on one single CPU?
  - The OS OS kernel creates the illusion of having more CPUs, so that multiple tasks execute in parallel
    - Tasks have the illusion of executing concurrently
    - A dedicated CPU per task
- Concurrency is implemented by multiplexing tasks on the same CPU...
  - Tasks are alternated on a real CPU...
  - ...And the *task scheduler* decides which task executes at a given instant in time
- Tasks are associated temporal constraints (deadlines)
  - The scheduler must allocate the CPU to tasks so that their deadlines are respected

Scheduler - 1

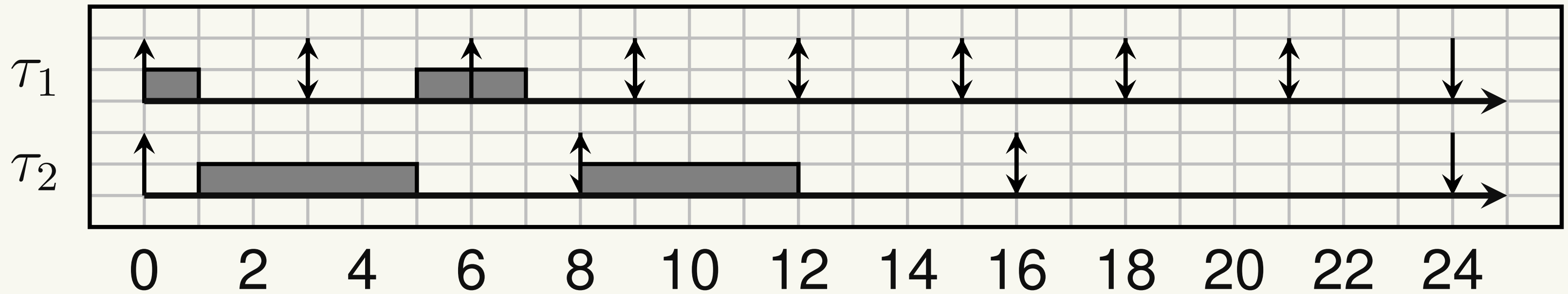
- Scheduler: responsible for generating a *schedule* from a set of ready tasks
  - Interesting definition: the scheduler is the thing that generates the schedule
- Let's be serious... Start from a mathematical model
  - First, consider UP systems (simpler definition)
    - A schedule $\sigma(t)$ is a function mapping time t into an executing task
      $$\sigma : t \rightarrow \mathcal{T} \cup \tau_{idle}$$
      where $\mathcal{T}$ is the set of tasks in the system
    - τ_{idle} is the *idle task*: when it is scheduled, the CPU becomes idle
  - For an SMP system (m CPUs), $\sigma(t)$ can be extended to map t in vectors $\tau \in (\mathcal{T} \cup \tau_{idle})^m$

Scheduler - 2

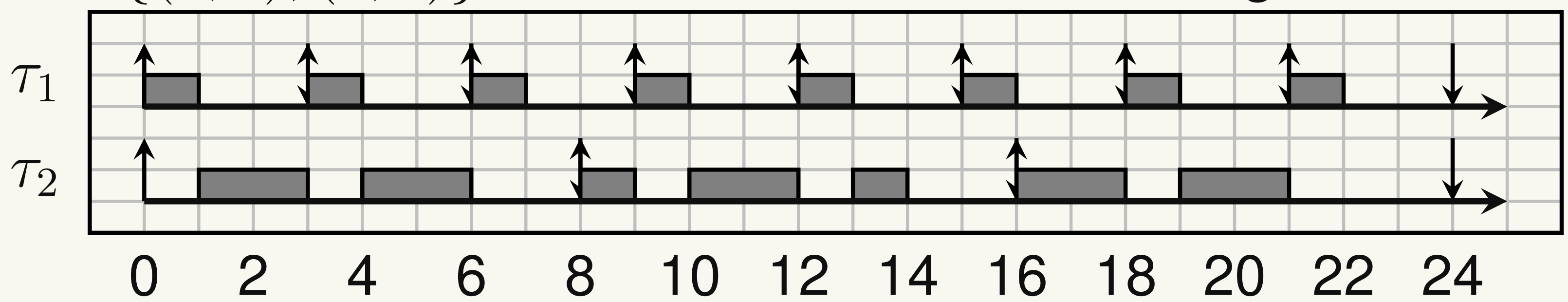
- Scheduler: implements $\sigma(t)$
  - The scheduler is responsible for selecting the task to execute at time t
- From an algorithmic point of view
  - Scheduling algorithm $\rightarrow$ Algorithm used to select for each time instant t a task to be executed on a CPU among the ready task
  - Given a task set $\mathcal{T}$, a scheduling algorithm $\mathcal{A}$ generates the schedule $\sigma_{\mathcal{A}}(t)$
- A task set is schedulable by an algorithm $\mathcal{A}$ if $\sigma_{\mathcal{A}}$ does not contain missed deadlines
- Schedulability test $\rightarrow$ check if $\mathcal{T}$ is schedulable by $\mathcal{A}$

RT Scheduling: Why?

- The task set $\mathcal{T} = \{(1, 3), (4, 8)\}$ is not schedulable by FCFS



- $\mathcal{T} = \{(1, 3), (4, 8)\}$ is schedulable with other algorithms



Cyclic Executive Scheduling

- Very low overhead (scheduling decisions taken off-line)
- Very simple and well-tested
  - Mainly used in legacy applications and where reliability is fundamental
  - Example: military and avionics systems
    - Air traffic control
    - Space Shuttle
    - Boeing 777
- Also called timeline scheduling or cyclic scheduling
- Originally used for periodic tasks

The idea

- Static scheduling algorithm
- Jobs are not preemptable
  - A scheduled job executes until termination
- The time axis is divided in time slots
- Slots are statically allocated to the tasks (scheduling table)
- A periodic timer activates execution (allocation of a slot)
  - Major Cycle: least common multiple (lcm) of all the tasks' periods (also called *hyperperiod*)
  - Minor Cycle: greatest common divisor (gcd) of all the tasks' periods
  - A timer fires every Minor Cycle Δ

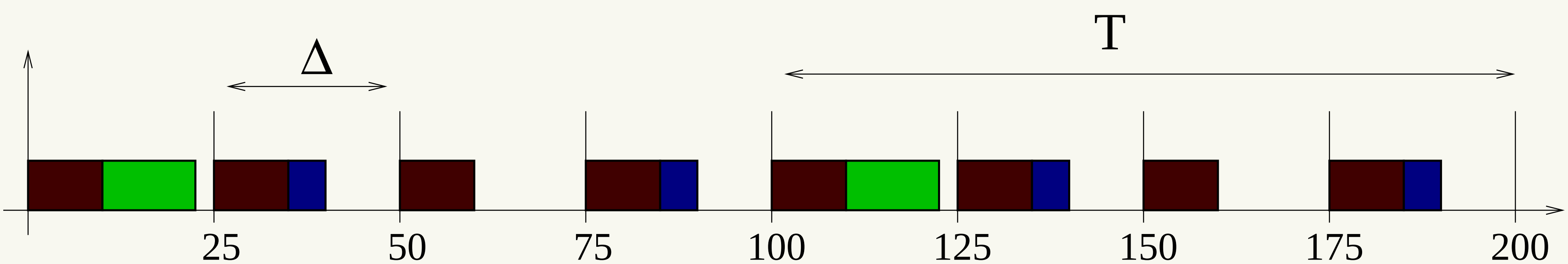
Example

- Consider a taskset $\Gamma = \{\tau_1, \tau_2, \tau_3\}$
  - Periodic tasks $\tau_i = (C_i, D_i, T_i)$, $D_i = T_i$
  - $T_1 = 25ms$, $T_2 = 50ms$, $T_3 = 100ms$

1. Minor Cycle $\Delta = gcd(25, 50, 100) = 25ms$
2. Major Cycle $T = lcm(25, 50, 100) = 100ms$
3. Compute a schedule that respects the task periods
   - Allocate tasks in slots of size $\Delta = 25ms$
   - The schedule repeats every $T = 100ms$
   - τ_1 must be scheduled every $25ms$, τ_2 must be scheduled every $50ms$, τ_3 must be scheduled every $100ms$
   - In every minor cycle, the tasks must execute for less than $25ms$

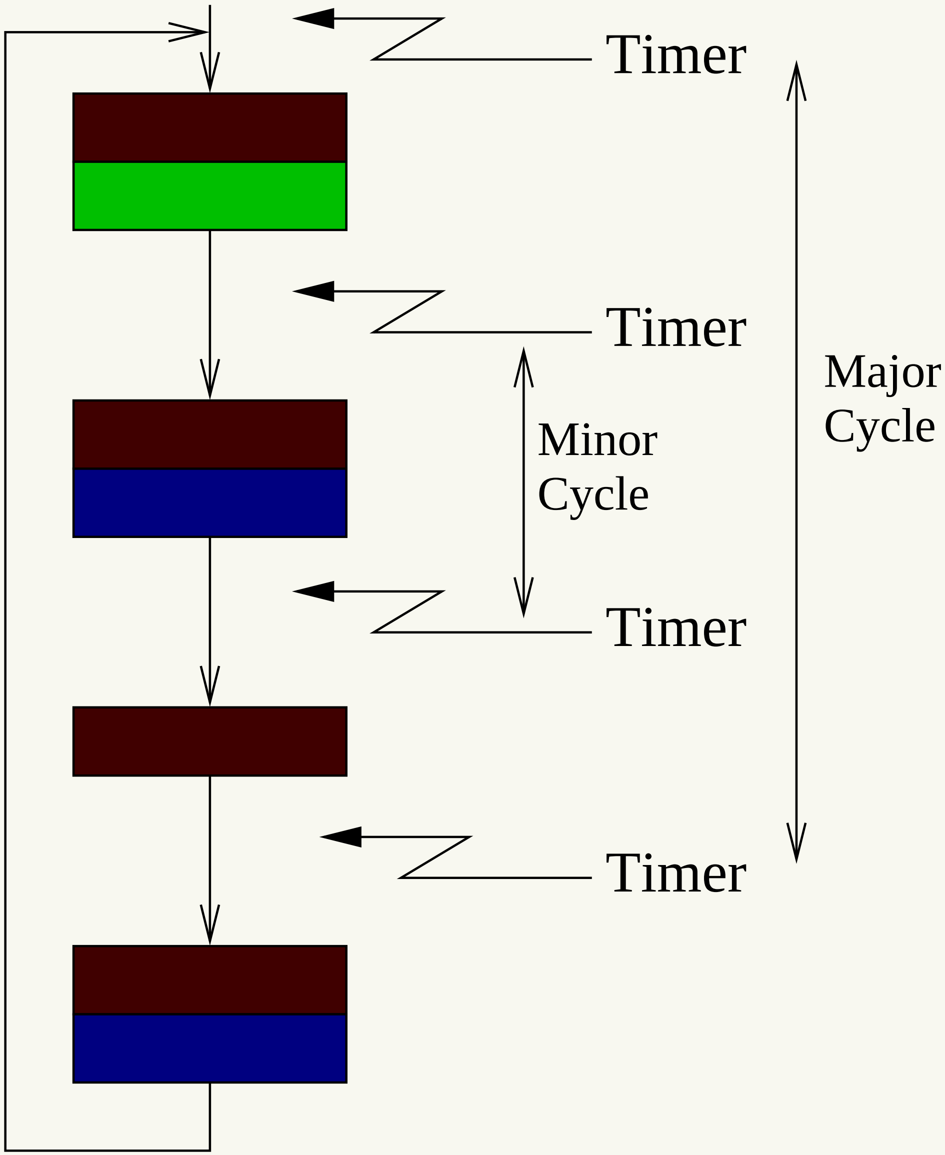
Example - The Schedule

- The schedule repeats every 4 minor cycles
  - τ_1 must be scheduled every $25ms \Rightarrow$ scheduled in every minor cycle
  - τ_2 must be scheduled every $50ms \Rightarrow$ scheduled every 2 minor cycles
  - τ_3 must be scheduled every $100ms \Rightarrow$ scheduled every 4 minor cycles



- First minor cycle: $C_1 + C_3 \leq 25ms$
- Second minor cycle: $C_1 + C_2 \leq 25ms$

Implementation



- Periodic timer firing every minor cycle
- Every time the timer fires...
- ...Read the scheduling table and execute the appropriate tasks
- Then, sleep until next minor cycle

Advantages

- Simple implementation (no real-time operating system is required)
  - No real task exist: just function calls
    - One single stack for all the “tasks”
- Non-preemptable scheduling $\Rightarrow$ no need to protect data
  - No need for semaphores, pipes, mutexes, mailboxes, etc.
- Low run-time overhead
- Jitter can be explicitly controlled

Drawbacks

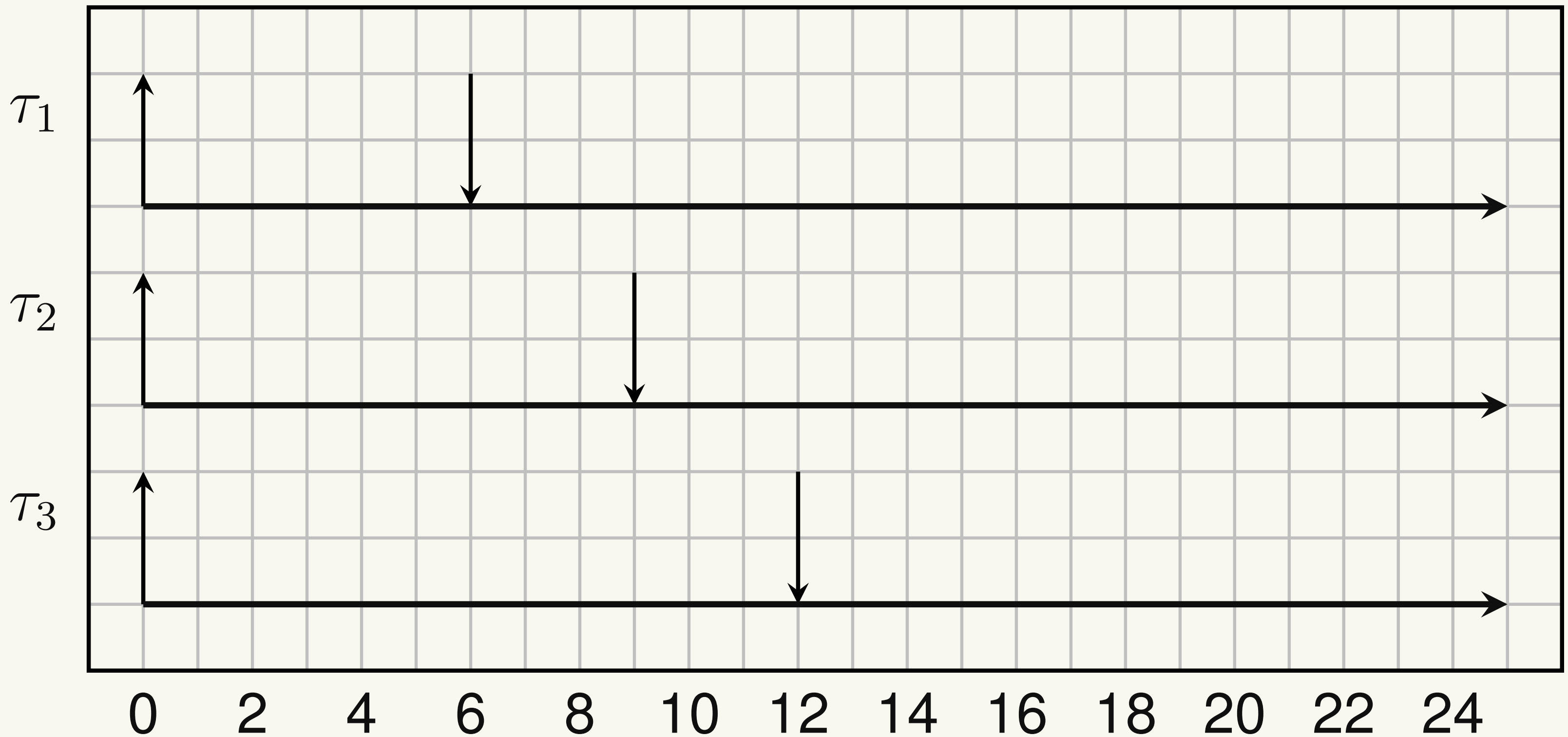
- Not robust during overloads
- Difficult to expand the schedule (static schedule)
  - New task $\Rightarrow$ the whole schedule must be recomputed
- Not easy to handle aperiodic/sporadic tasks
- All task periods must be a multiple of the minor cycle time
- Difficult to incorporate processes with long periods (big tables)
- Variable computation time $\Rightarrow$ it might be necessary to split tasks into a fixed number of fixed size procedures

Fixed Priority Scheduling

- Very simple *preemptive* scheduling algorithm
  - Every task τ_i is assigned a fixed priority p_i
  - The active task with the highest priority is scheduled
- Priorities are integer numbers: the higher the number, the higher the priority
  - In the research literature, sometimes authors use the opposite convention: the lowest the number, the highest the priority
- In the following we show some examples, considering periodic tasks, constant execution times, and deadlines equal to the period

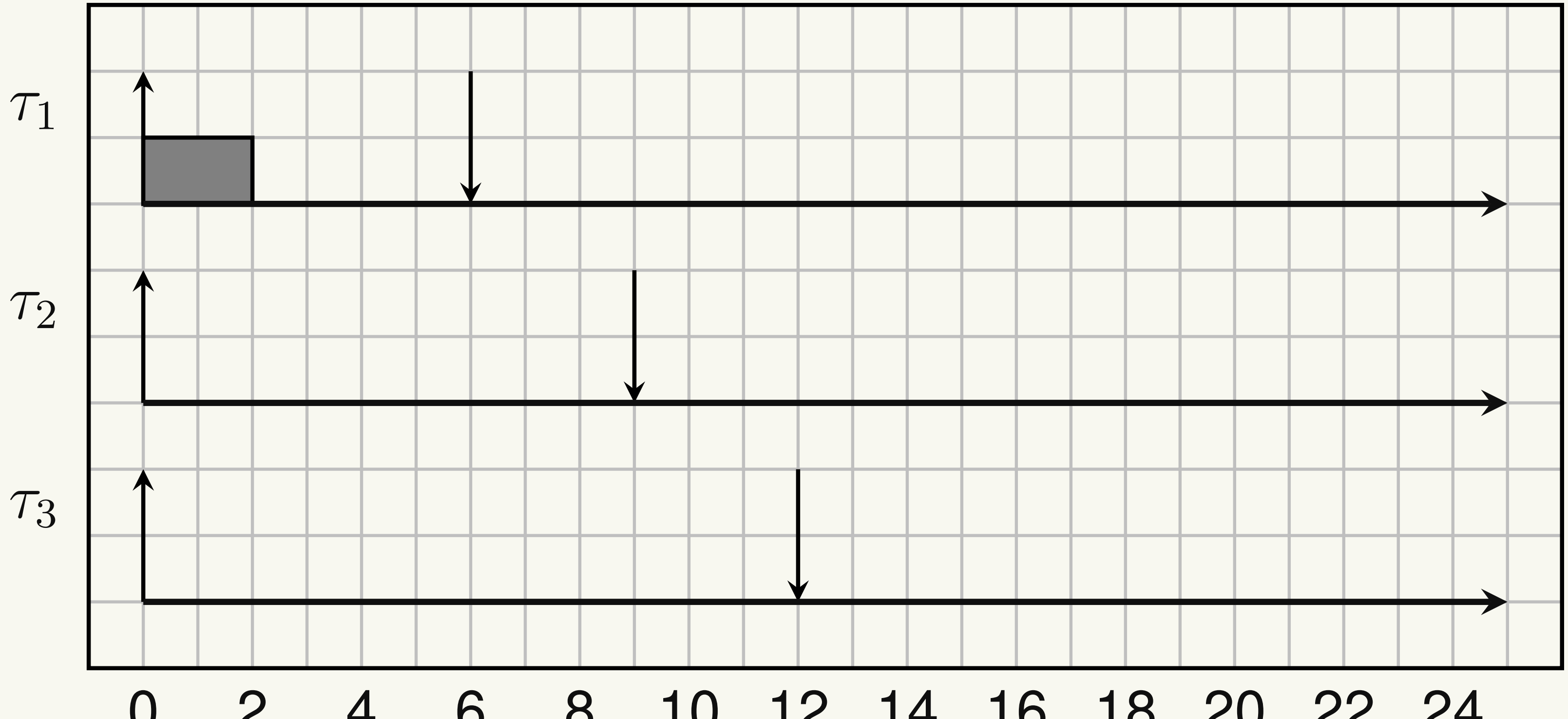
Example of Schedule

- Consider the following task set: $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 9, 9)$, $\tau_3 = (3, 12, 12)$. Task τ_1 has priority $p_1 = 3$ (highest), task τ_2 has priority $p_2 = 2$, task τ_3 has priority $p_3 = 1$ (lowest)



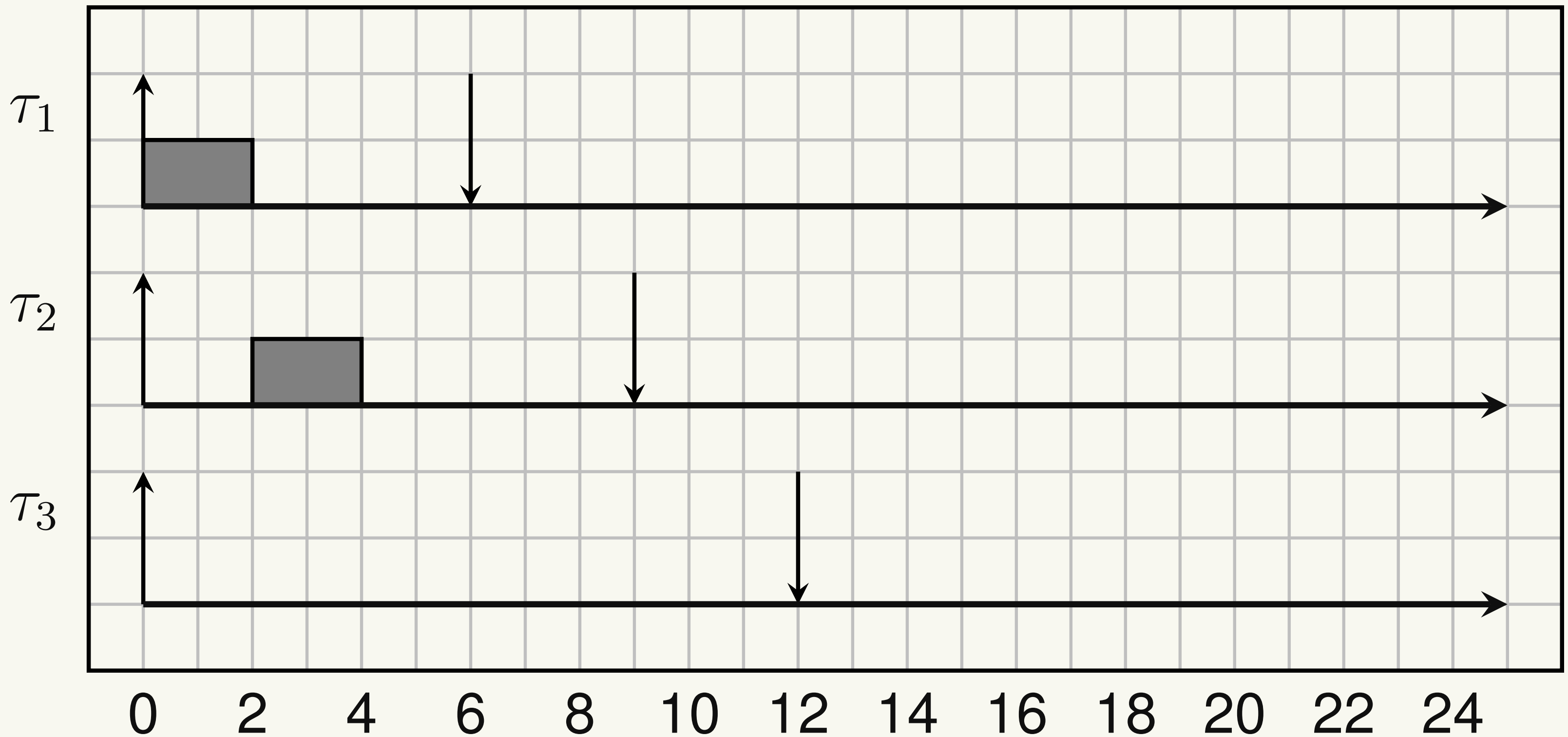
Example of Schedule

- Consider the following task set: $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 9, 9)$, $\tau_3 = (3, 12, 12)$. Task τ_1 has priority $p_1 = 3$ (highest), task τ_2 has priority $p_2 = 2$, task τ_3 has priority $p_3 = 1$ (lowest)



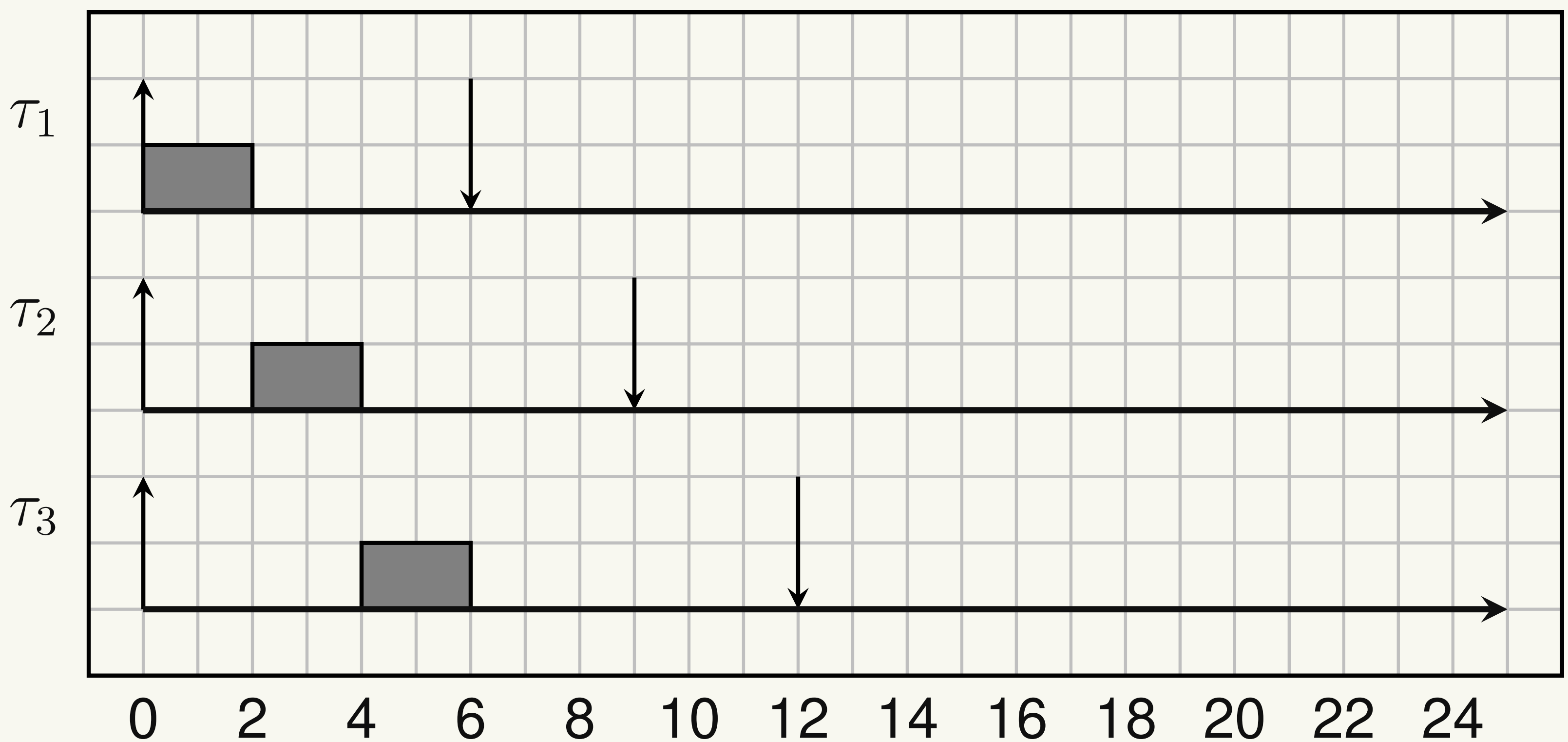
Example of Schedule

- Consider the following task set: $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 9, 9)$, $\tau_3 = (3, 12, 12)$. Task τ_1 has priority $p_1 = 3$ (highest), task τ_2 has priority $p_2 = 2$, task τ_3 has priority $p_3 = 1$ (lowest)



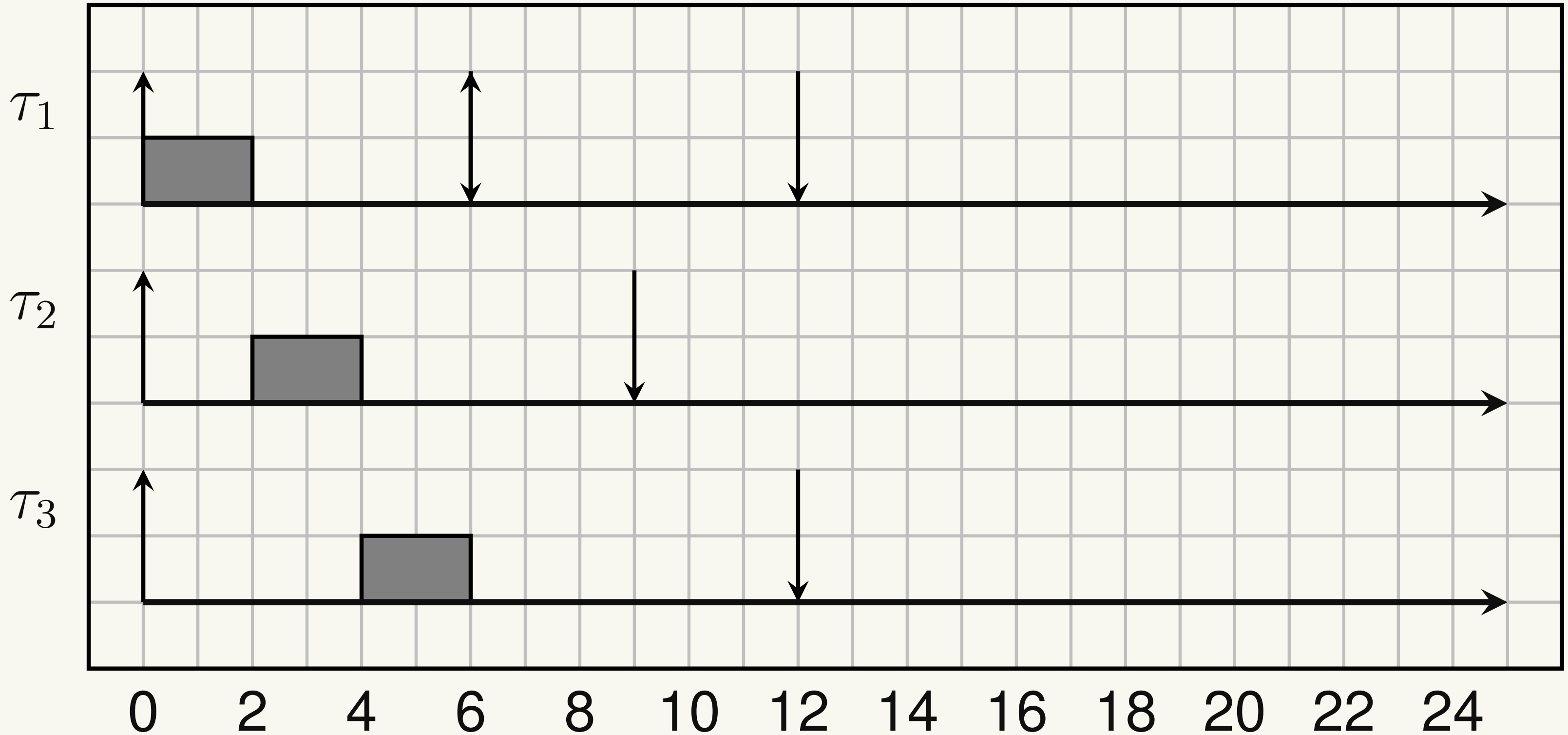
Example of Schedule

- Consider the following task set: $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 9, 9)$, $\tau_3 = (3, 12, 12)$. Task τ_1 has priority $p_1 = 3$ (highest), task τ_2 has priority $p_2 = 2$, task τ_3 has priority $p_3 = 1$ (lowest)



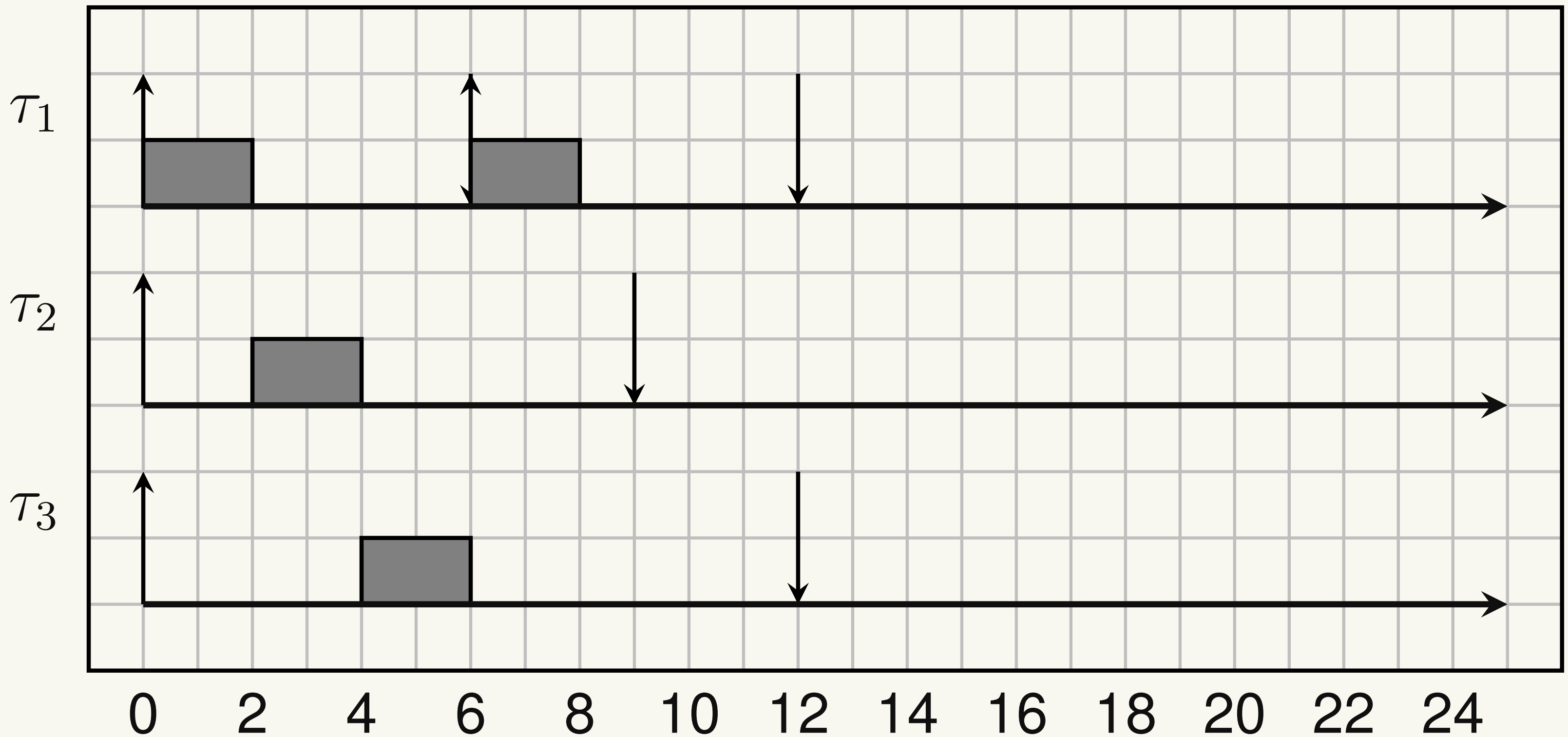
Example of Schedule

- Consider the following task set: $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 9, 9)$, $\tau_3 = (3, 12, 12)$. Task τ_1 has priority $p_1 = 3$ (highest), task τ_2 has priority $p_2 = 2$, task τ_3 has priority $p_3 = 1$ (lowest)



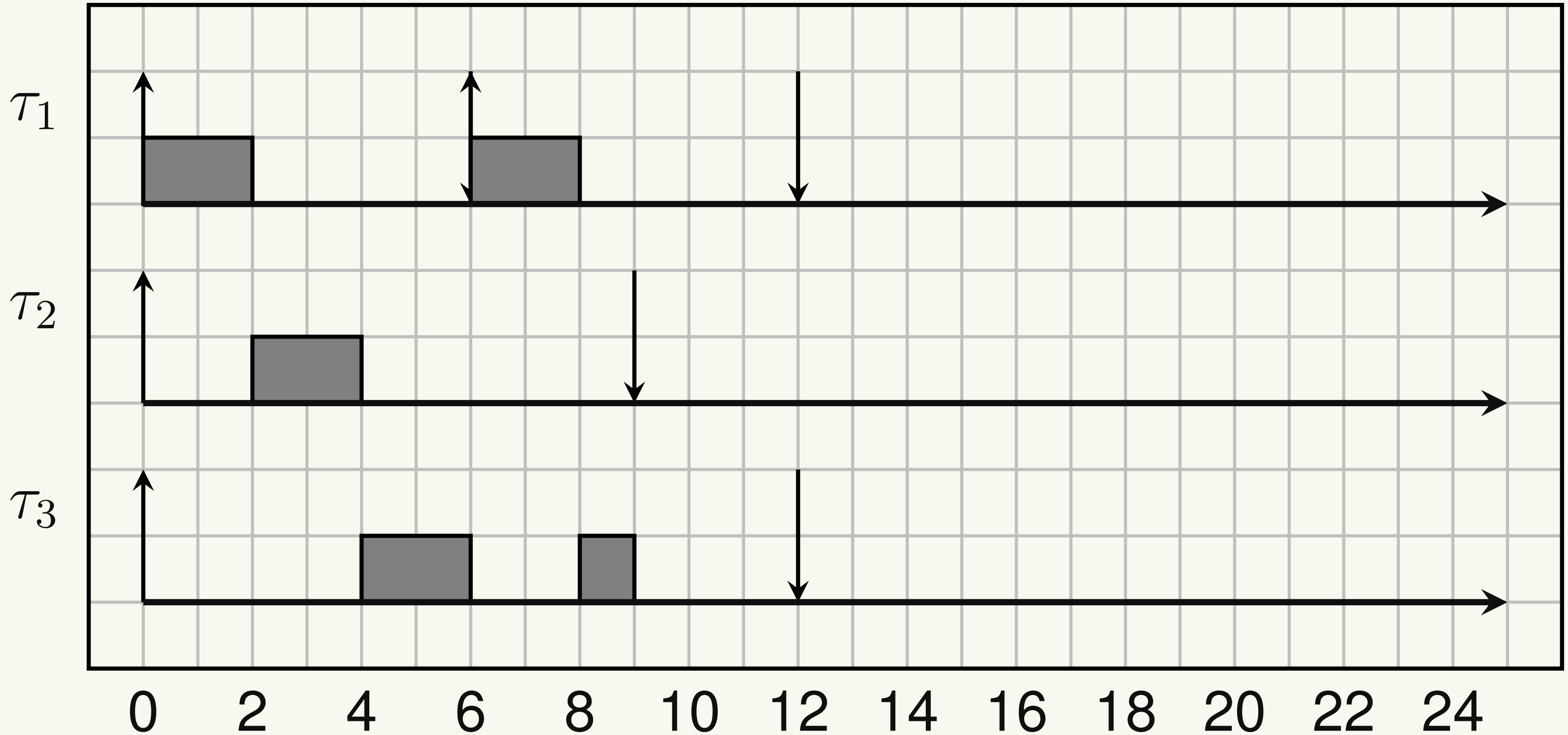
Example of Schedule

- Consider the following task set: $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 9, 9)$, $\tau_3 = (3, 12, 12)$. Task τ_1 has priority $p_1 = 3$ (highest), task τ_2 has priority $p_2 = 2$, task τ_3 has priority $p_3 = 1$ (lowest)



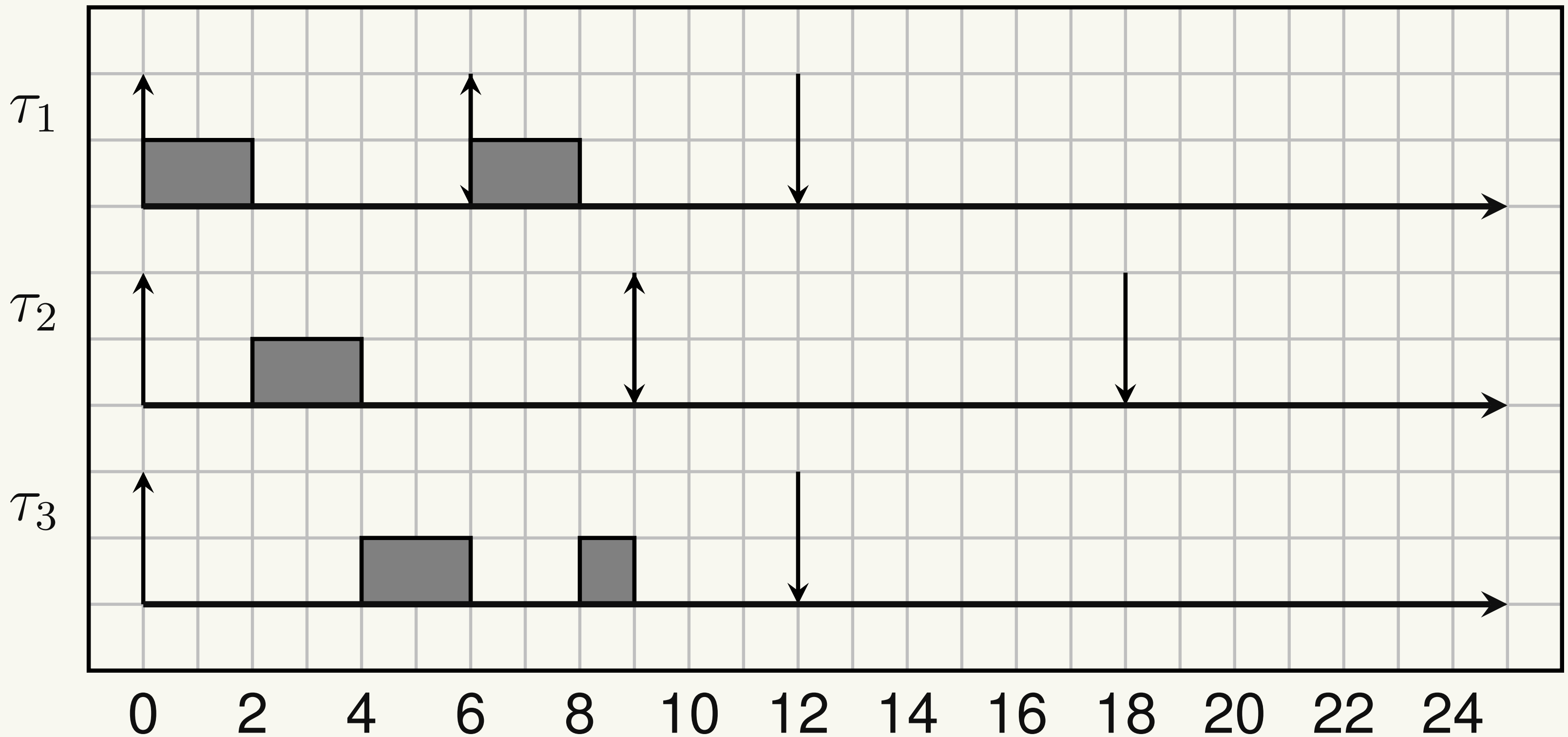
Example of Schedule

- Consider the following task set: $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 9, 9)$, $\tau_3 = (3, 12, 12)$. Task τ_1 has priority $p_1 = 3$ (highest), task τ_2 has priority $p_2 = 2$, task τ_3 has priority $p_3 = 1$ (lowest)



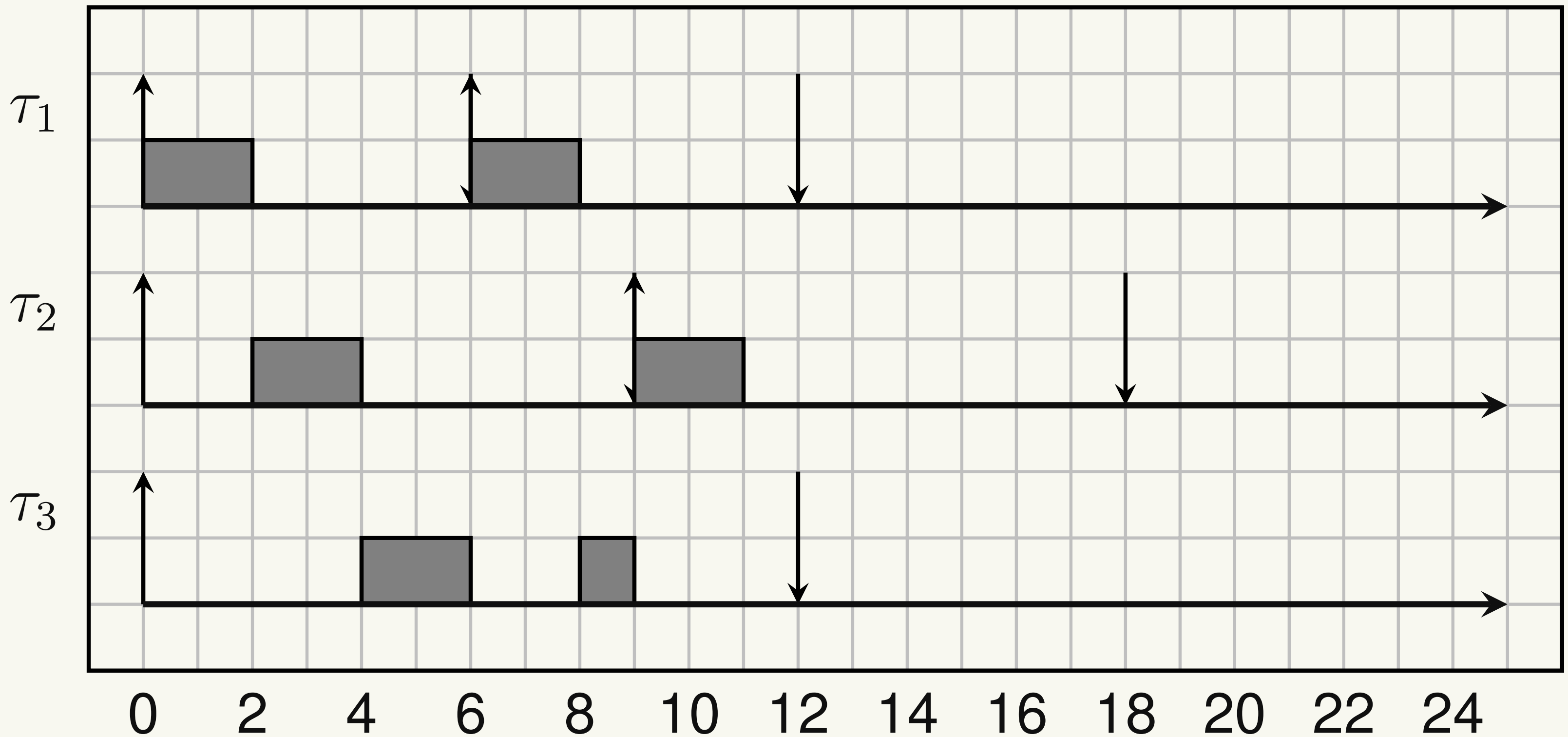
Example of Schedule

- Consider the following task set: $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 9, 9)$, $\tau_3 = (3, 12, 12)$. Task τ_1 has priority $p_1 = 3$ (highest), task τ_2 has priority $p_2 = 2$, task τ_3 has priority $p_3 = 1$ (lowest)



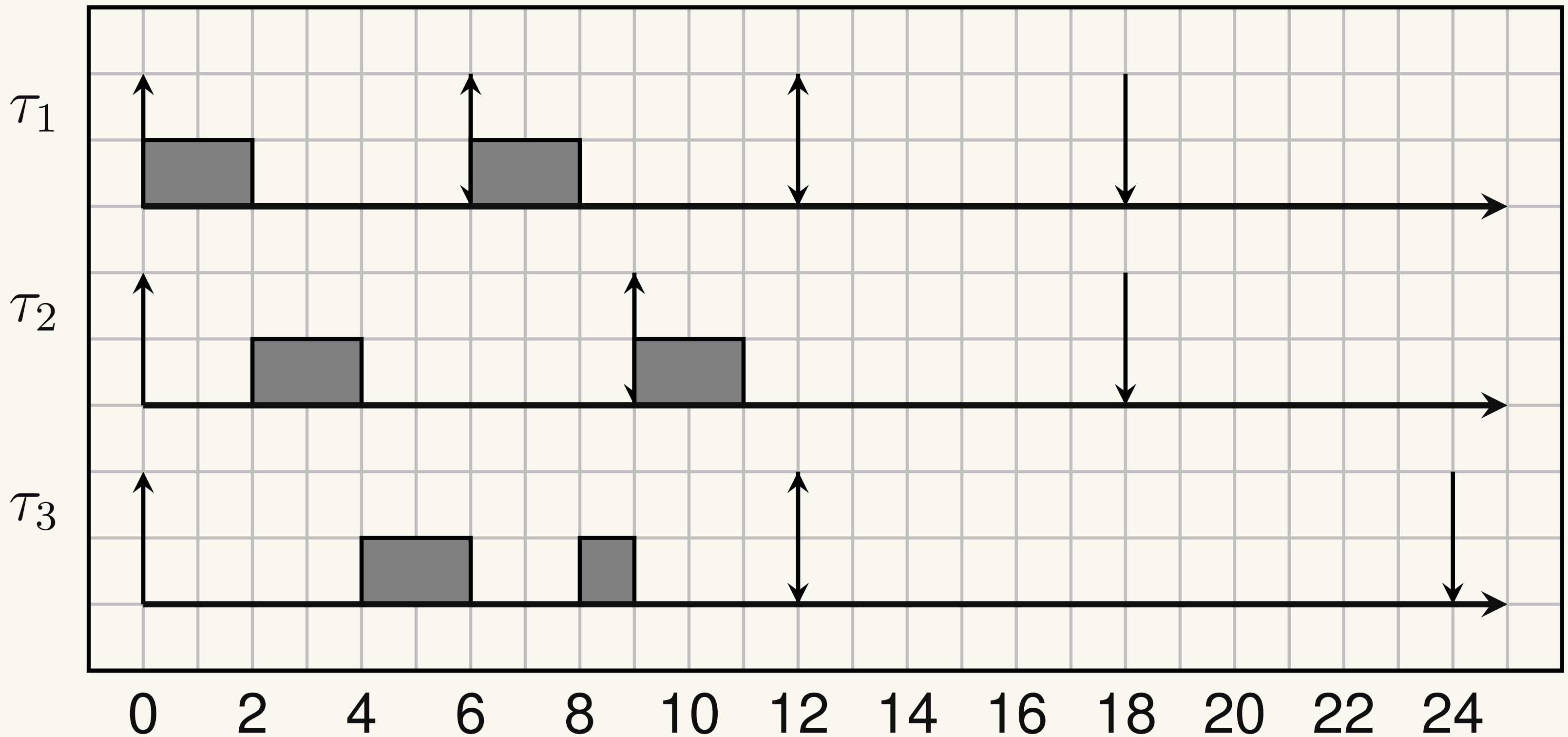
Example of Schedule

- Consider the following task set: $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 9, 9)$, $\tau_3 = (3, 12, 12)$. Task τ_1 has priority $p_1 = 3$ (highest), task τ_2 has priority $p_2 = 2$, task τ_3 has priority $p_3 = 1$ (lowest)



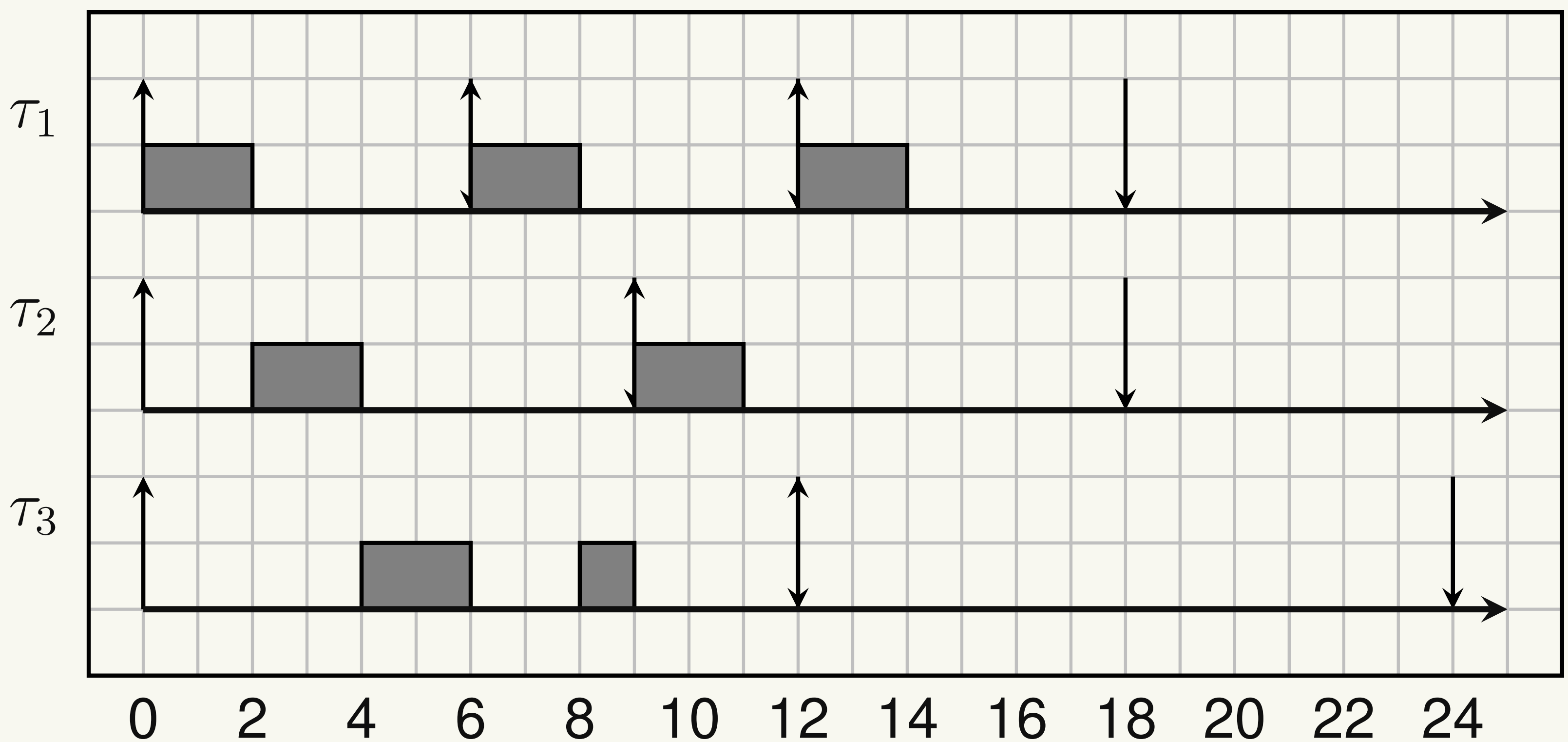
Example of Schedule

- Consider the following task set: $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 9, 9)$, $\tau_3 = (3, 12, 12)$. Task τ_1 has priority $p_1 = 3$ (highest), task τ_2 has priority $p_2 = 2$, task τ_3 has priority $p_3 = 1$ (lowest)



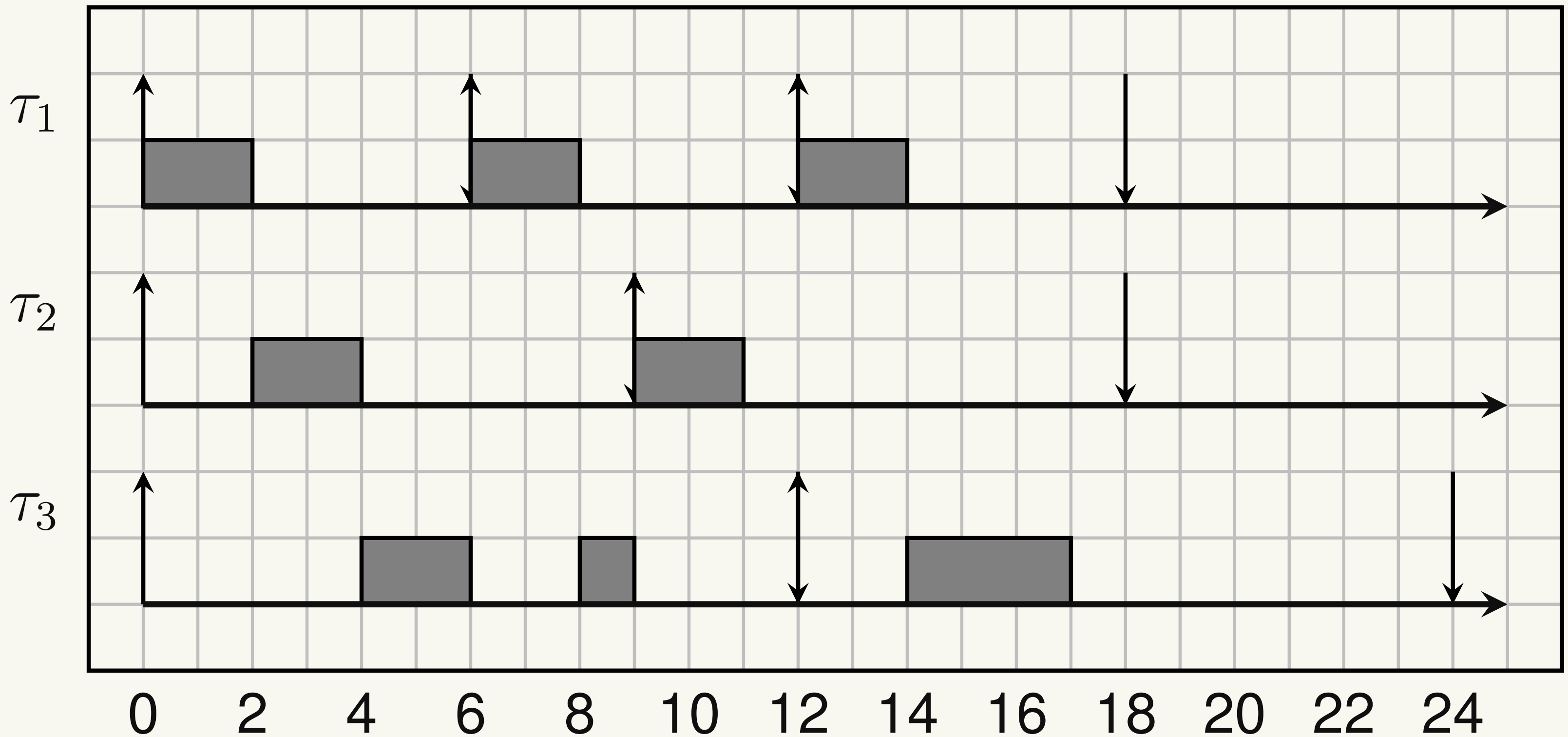
Example of Schedule

- Consider the following task set: $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 9, 9)$, $\tau_3 = (3, 12, 12)$. Task τ_1 has priority $p_1 = 3$ (highest), task τ_2 has priority $p_2 = 2$, task τ_3 has priority $p_3 = 1$ (lowest)



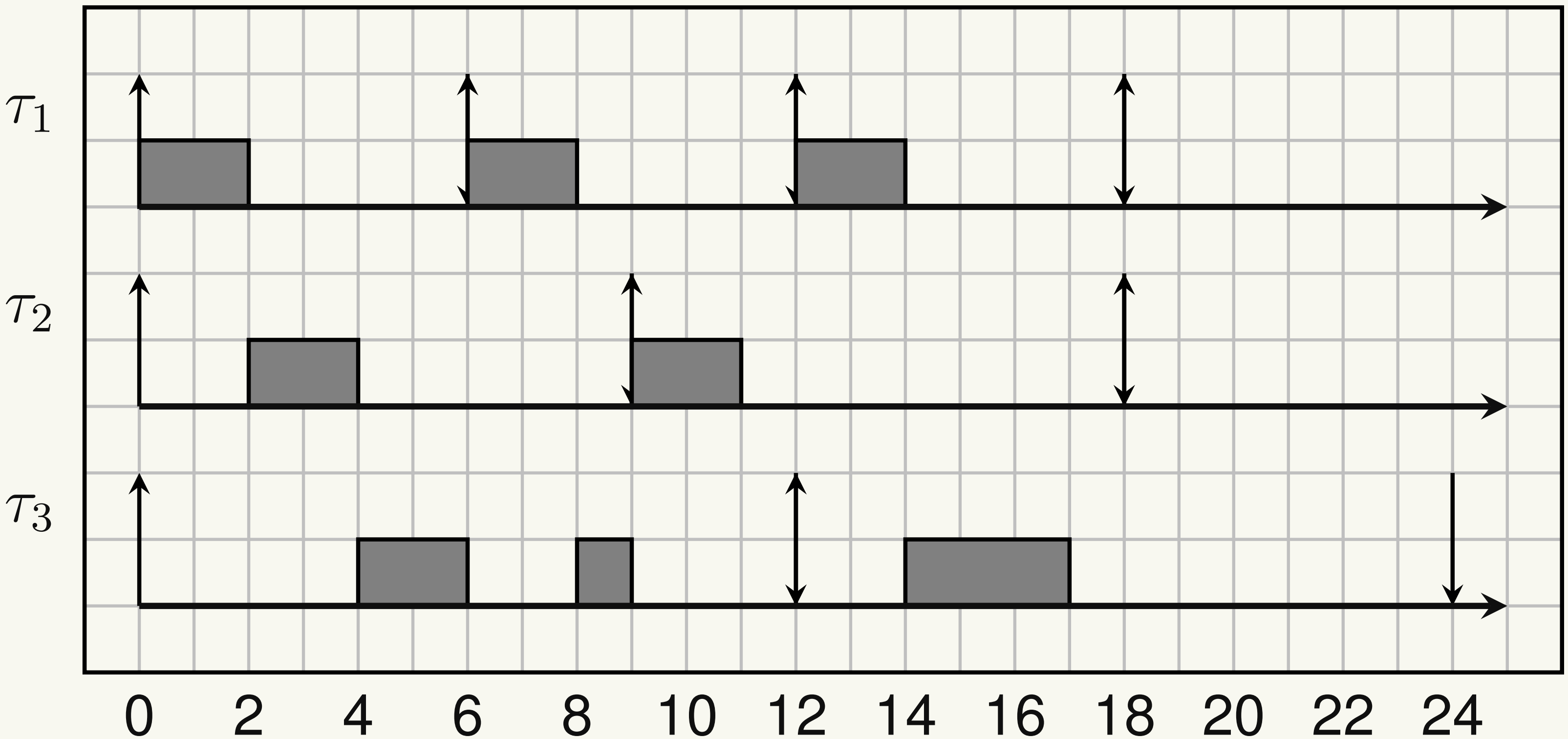
Example of Schedule

- Consider the following task set: $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 9, 9)$, $\tau_3 = (3, 12, 12)$. Task τ_1 has priority $p_1 = 3$ (highest), task τ_2 has priority $p_2 = 2$, task τ_3 has priority $p_3 = 1$ (lowest)

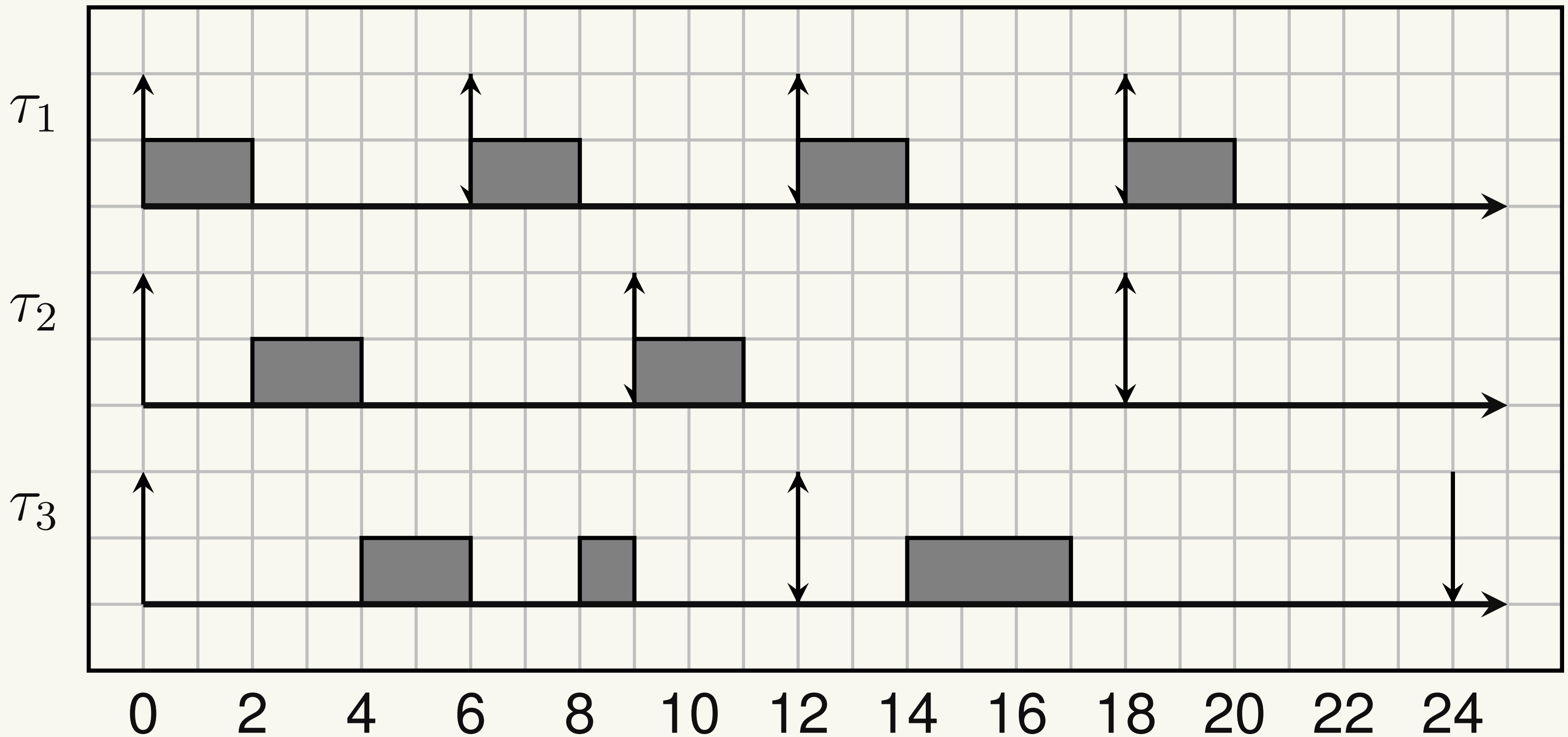


Example of Schedule

- Consider the following task set: $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 9, 9)$, $\tau_3 = (3, 12, 12)$. Task τ_1 has priority $p_1 = 3$ (highest), task τ_2 has priority $p_2 = 2$, task τ_3 has priority $p_3 = 1$ (lowest)

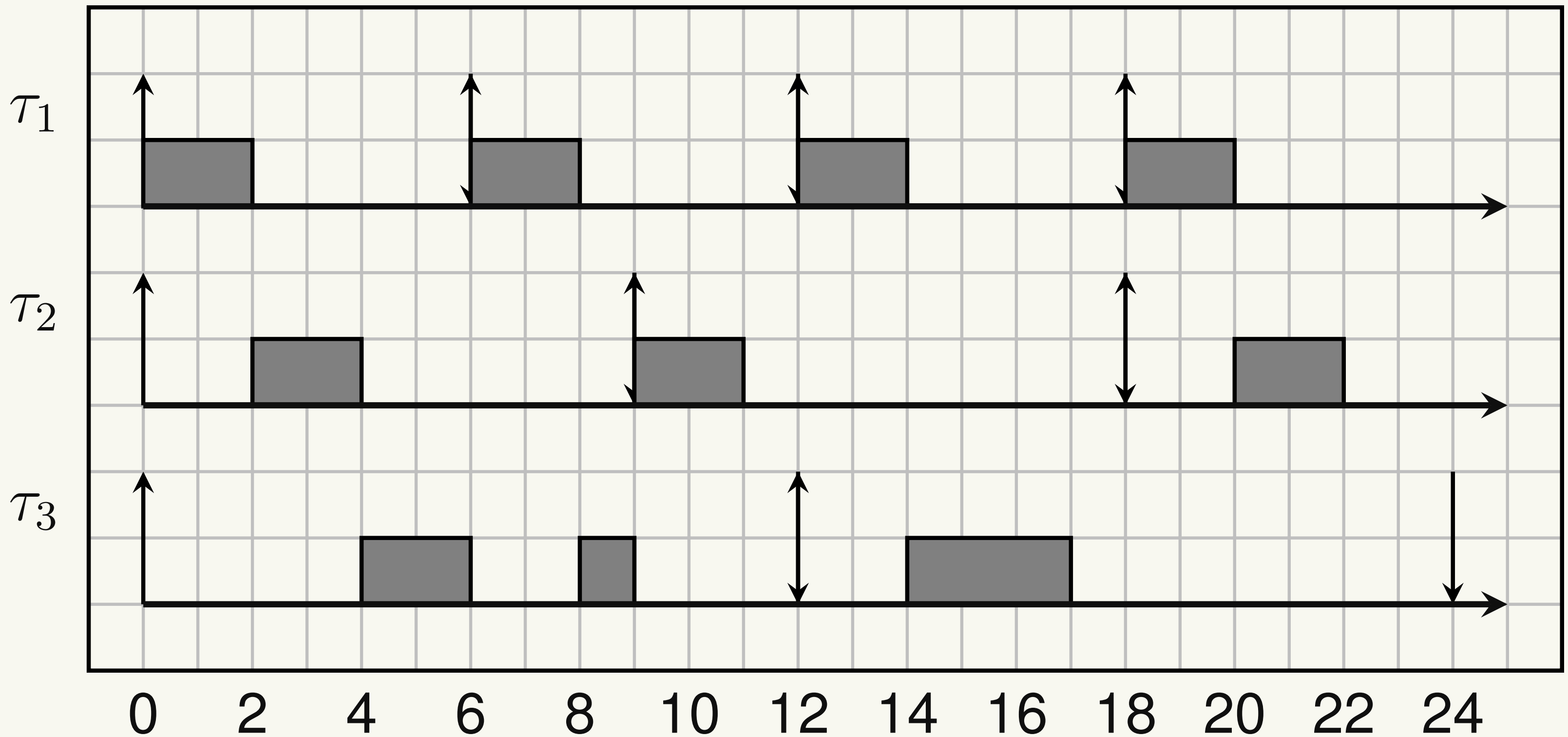


- Consider the following task set: $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 9, 9)$, $\tau_3 = (3, 12, 12)$. Task τ_1 has priority $p_1 = 3$ (highest), task τ_2 has priority $p_2 = 2$, task τ_3 has priority $p_3 = 1$ (lowest)



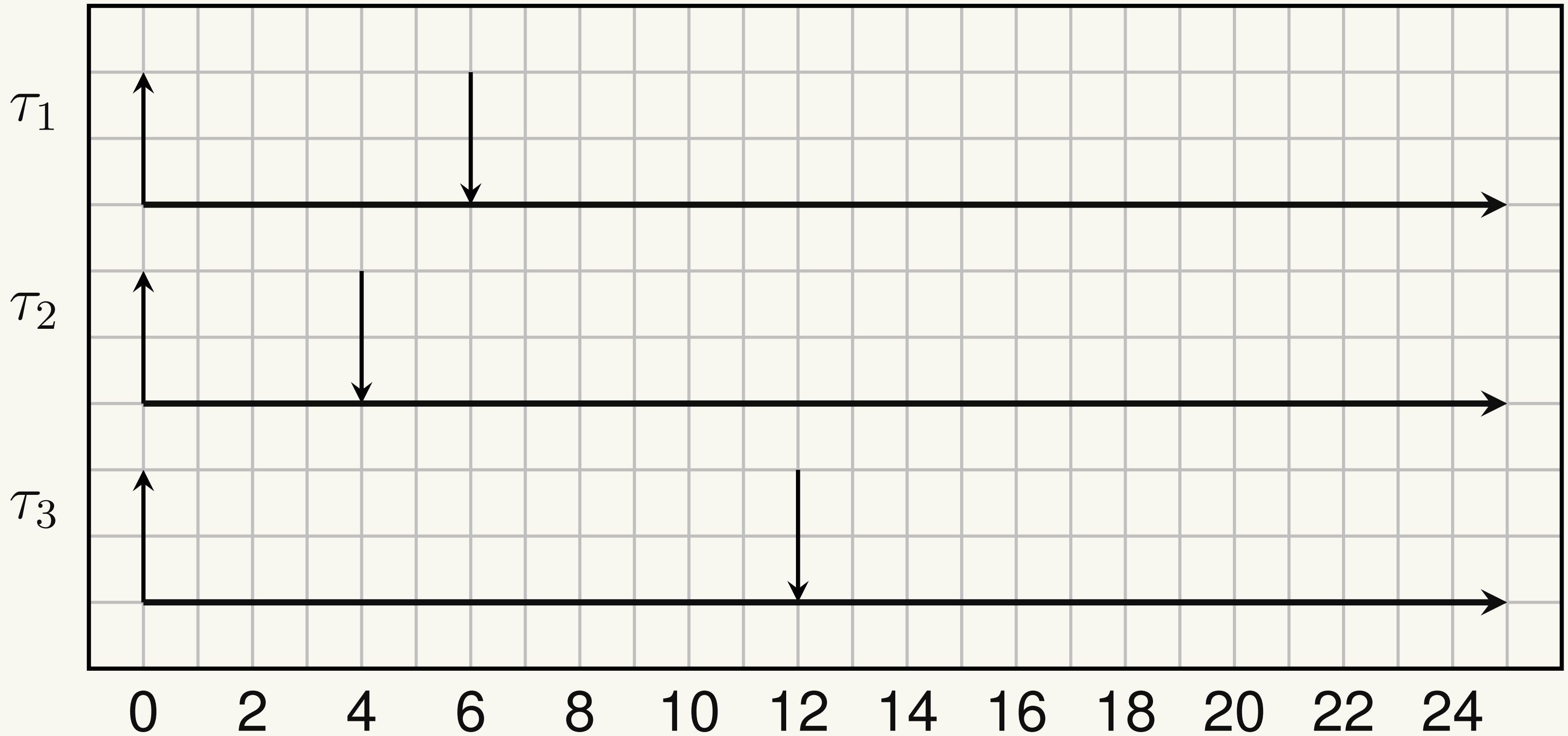
Example of Schedule

- Consider the following task set: $\tau_1 = (2, 6, 6)$, $\tau_2 = (2, 9, 9)$, $\tau_3 = (3, 12, 12)$. Task τ_1 has priority $p_1 = 3$ (highest), task τ_2 has priority $p_2 = 2$, task τ_3 has priority $p_3 = 1$ (lowest)



Another Example (non-schedulable)

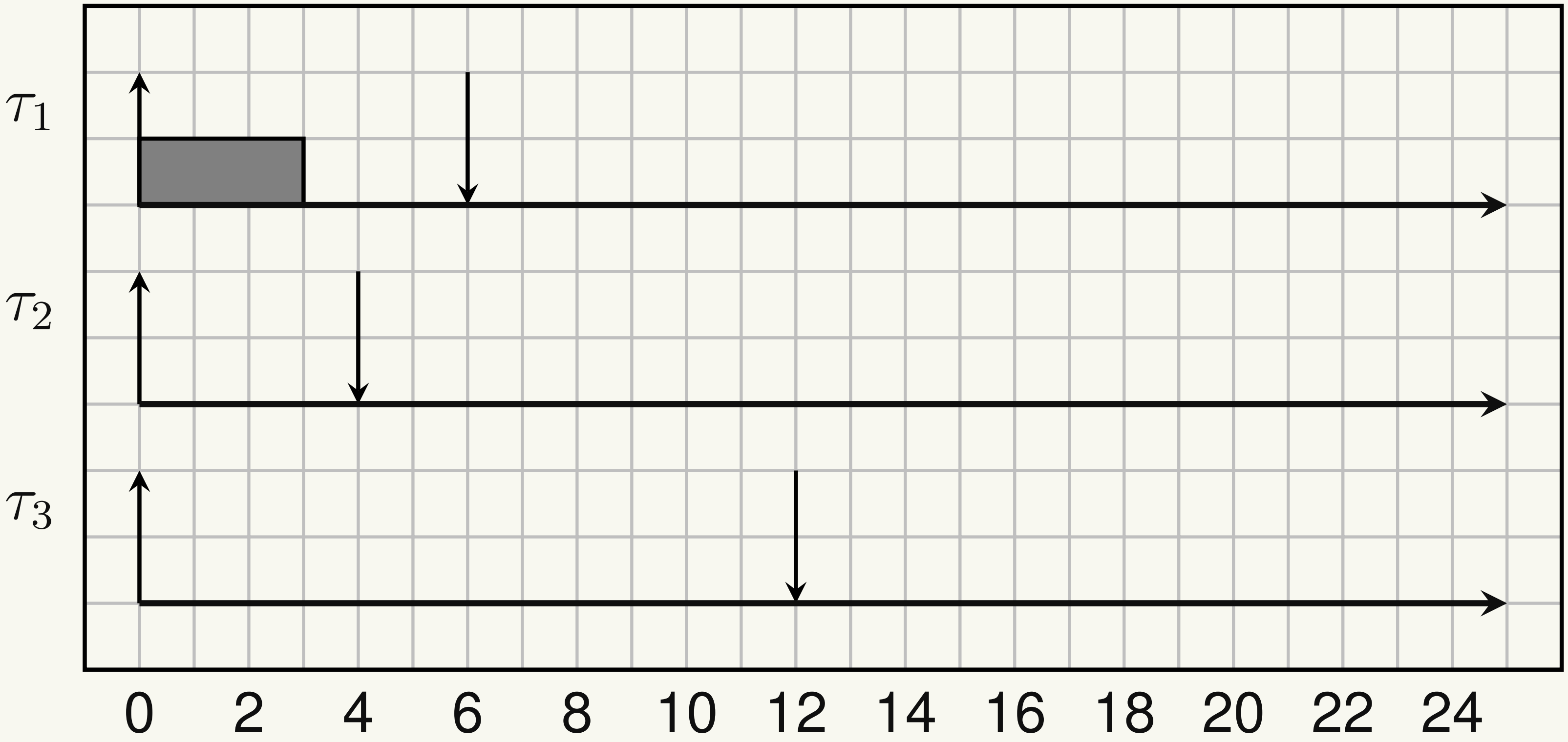
- Consider the following task set: $\tau_1 = (3, 6, 6)$, $p_1 = 3$, $\tau_2 = (2, 4, 8)$, $p_2 = 2$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



In this case, task τ_2 misses its deadline!

Another Example (non-schedulable)

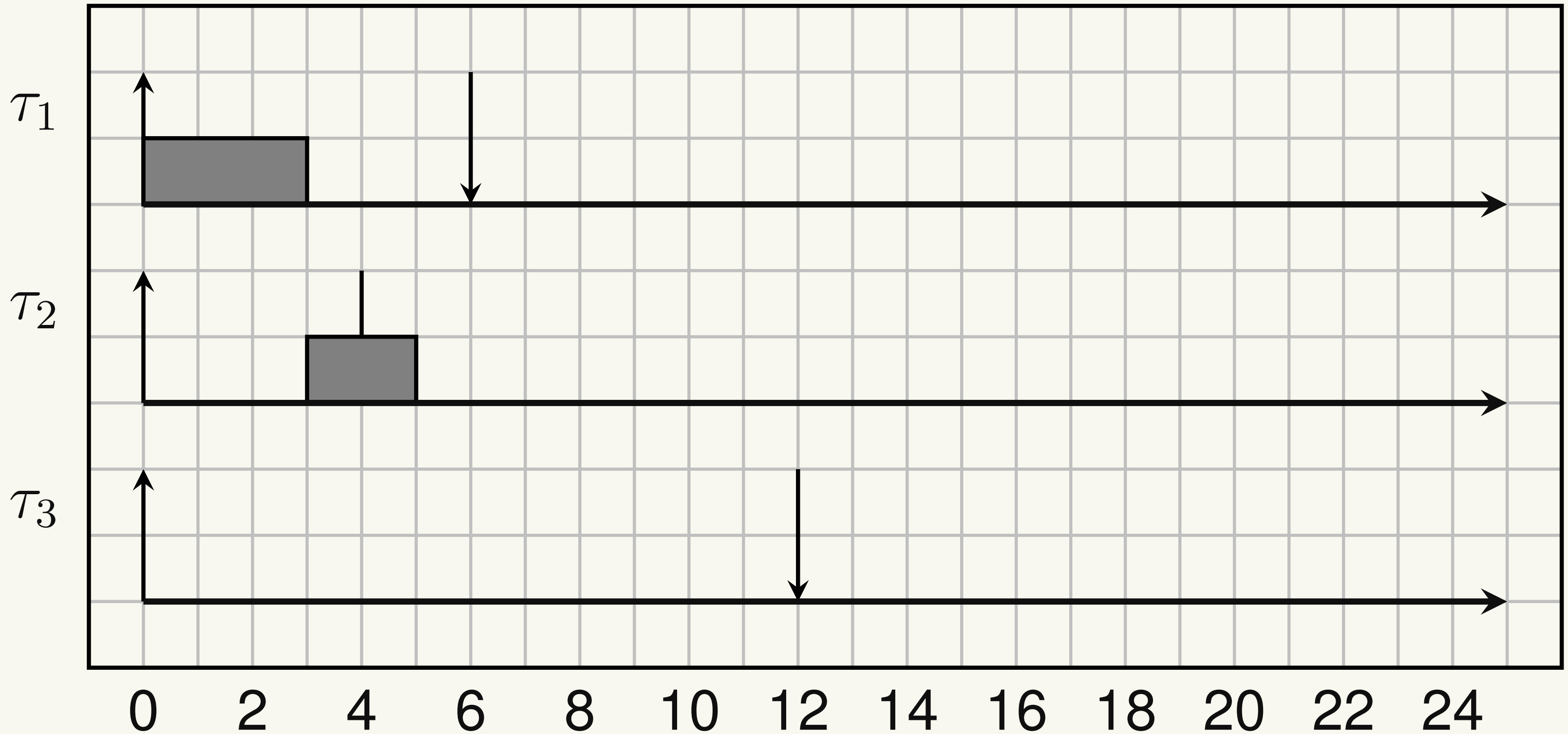
- Consider the following task set: $\tau_1 = (3, 6, 6)$, $p_1 = 3$, $\tau_2 = (2, 4, 8)$, $p_2 = 2$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



In this case, task τ_2 misses its deadline!

Another Example (non-schedulable)

- Consider the following task set: $\tau_1 = (3, 6, 6)$, $p_1 = 3$, $\tau_2 = (2, 4, 8)$, $p_2 = 2$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



In this case, task τ_2 misses its deadline!

Notes about Priority Scheduling

- Some considerations about the schedule shown before:
  - The response time of the task with the highest priority is minimum and equal to its WCET
  - The response time of the other tasks depends on the *interference* of the higher priority tasks
  - The priority assignment may influence the schedulability of a task set
    - Problem: how to assign tasks' priorities so that a task set is schedulable?

Priority Assignment

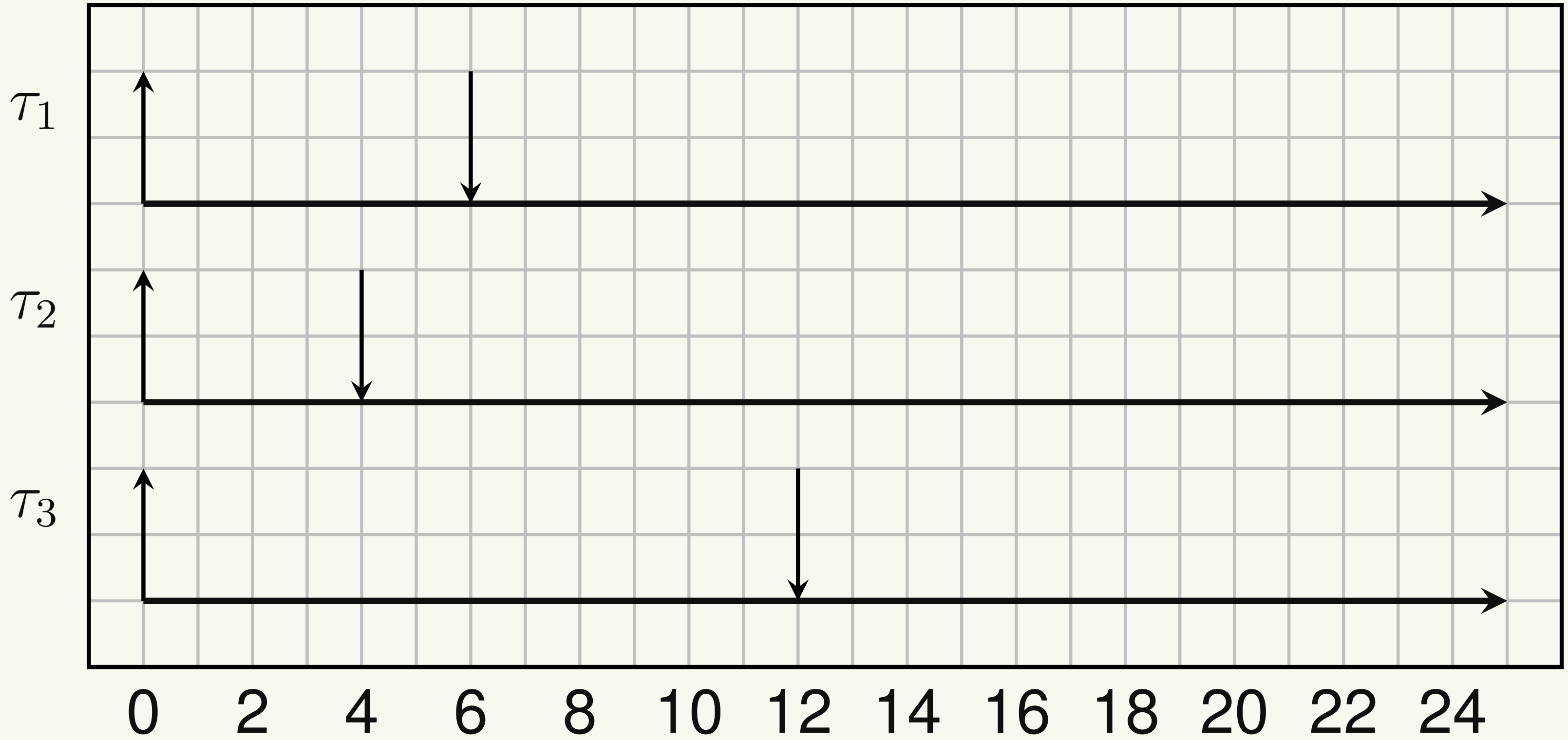
- Given a task set, how to assign priorities?
- There are two possible objectives:
  - Schedulability (i.e. find the priority assignment that makes all tasks schedulable)
  - Response time (i.e. find the priority assignment that minimise the response time of a subset of tasks)
- By now we consider the first objective only
- An *optimal* priority assignment *Opt* is such that:
  - If the task set is schedulable with another priority assignment, then it is schedulable with priority assignment *Opt*
  - If the task set is not schedulable with *Opt*, then it is not schedulable by any other assignment

Optimal Priority Assignment

- Given a periodic task set T with all tasks having relative deadline D_i equal to the period T_i ($\forall i, \;\; D_i = T_i$), and with all offsets equal to 0 ($\forall i, \;\; r_{i,0} = 0$):
  - The best assignment is the *Rate Monotonic* (RM) assignment
  - Shorter period $\rightarrow$ higher priority
- Given a periodic task set with deadline different from periods, and with all offsets equal to 0 ($\forall i, \quad r_{i,0} = 0$):
  - The best assignment is the *Deadline Monotonic* assignment
  - Shorter relative deadline $\rightarrow$ higher priority
- For sporadic tasks, the same rules are valid as for periodic tasks with offsets equal to 0

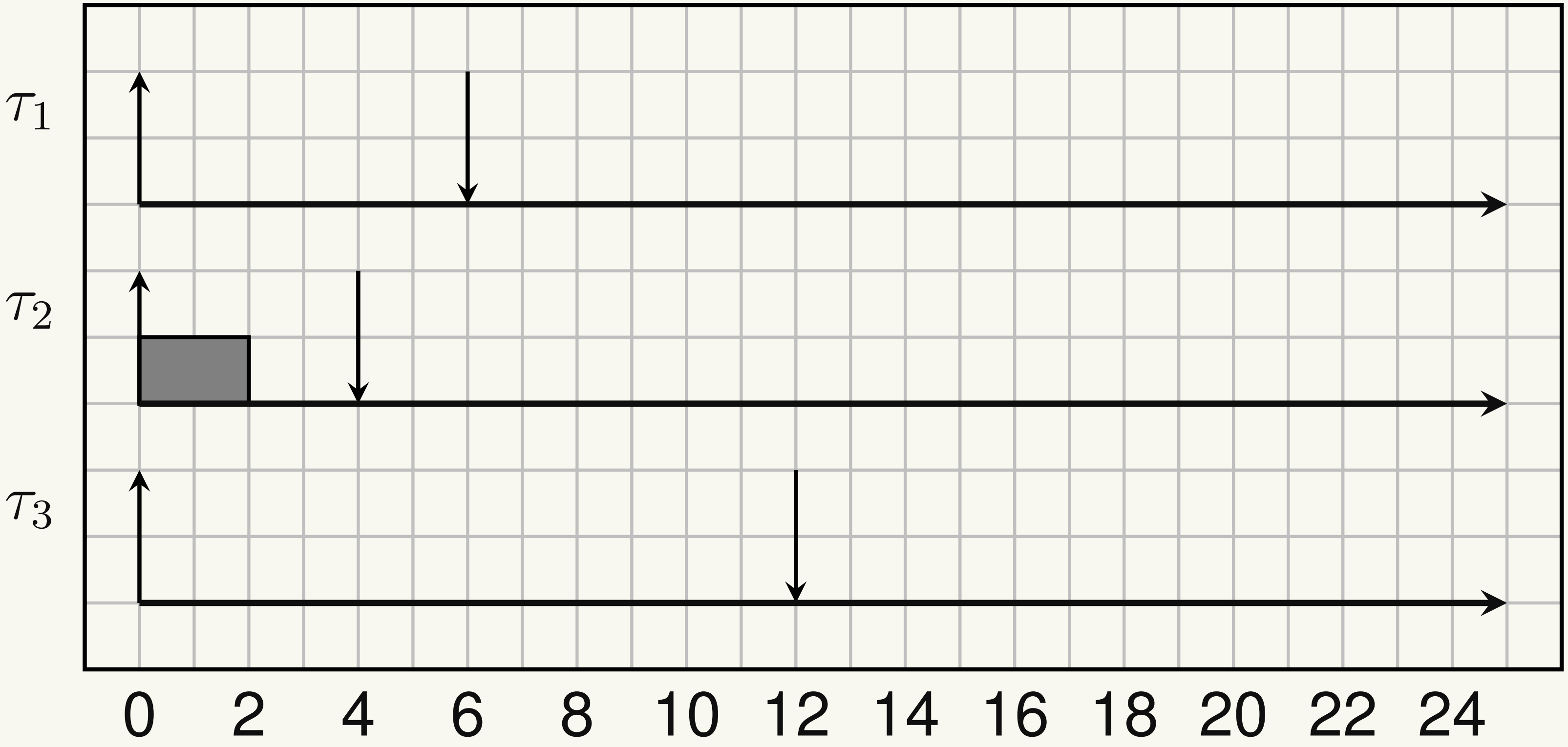
Example revised

- Consider the example shown before with deadline monotonic:
  $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



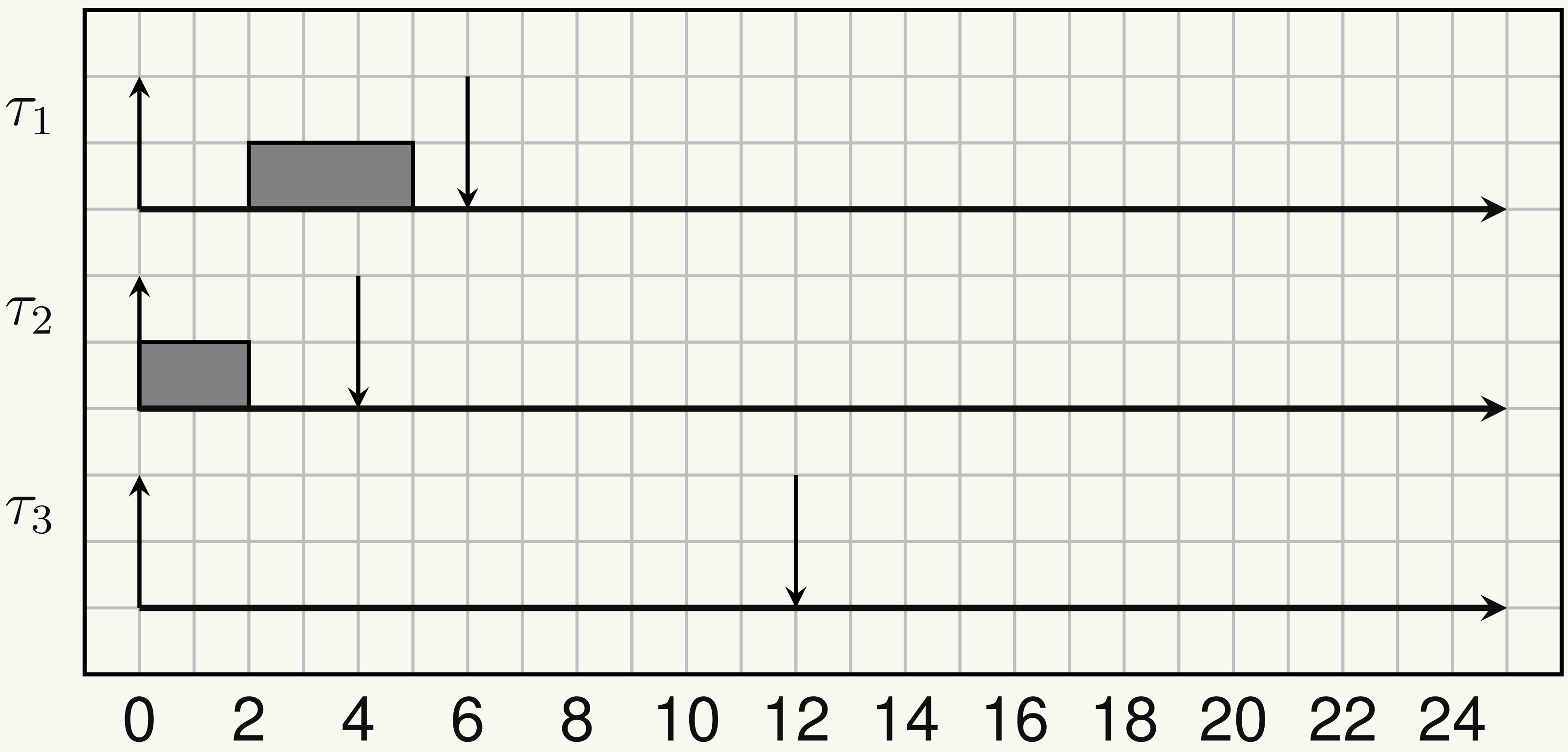
Example revised

- Consider the example shown before with deadline monotonic:
  $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



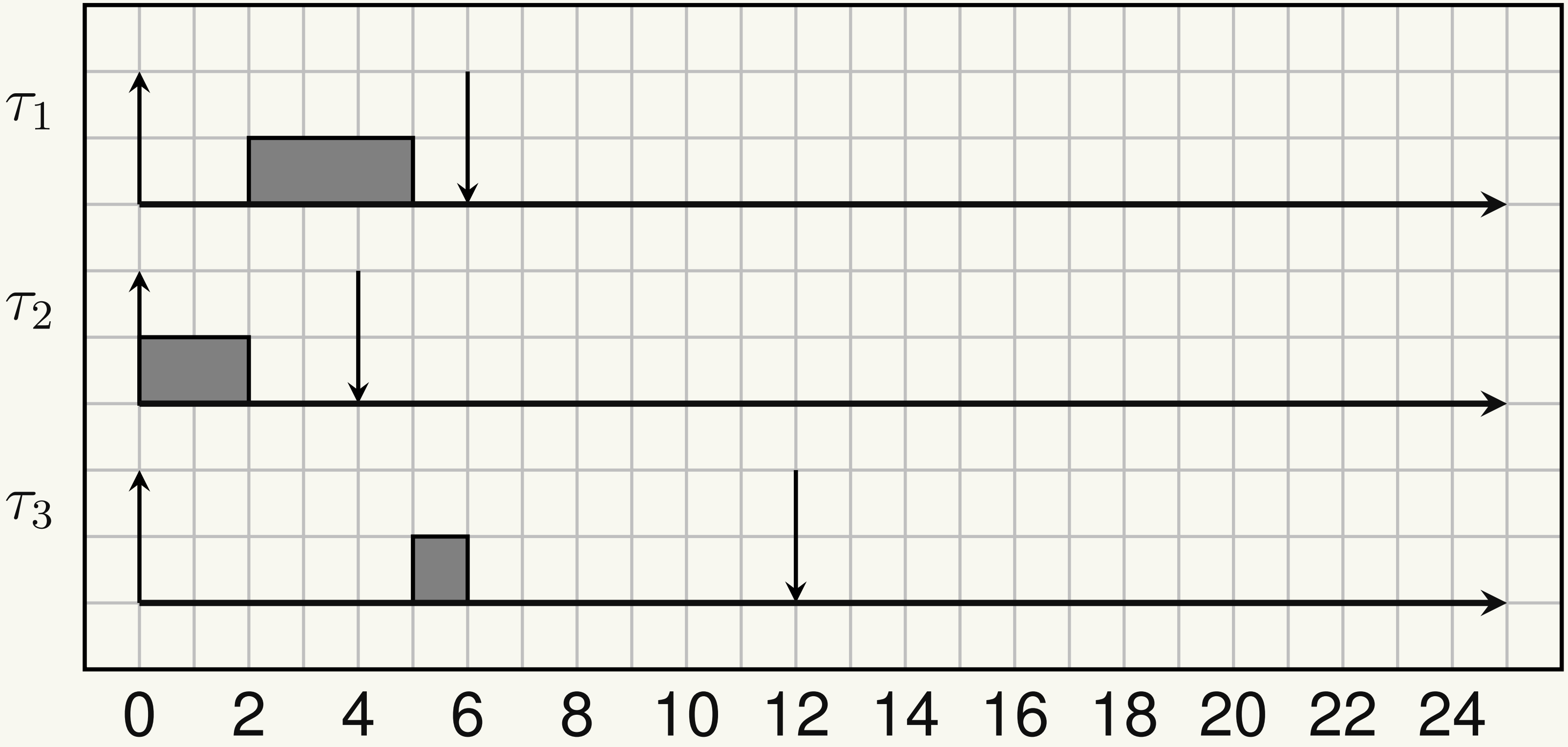
Example revised

- Consider the example shown before with deadline monotonic:
  $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



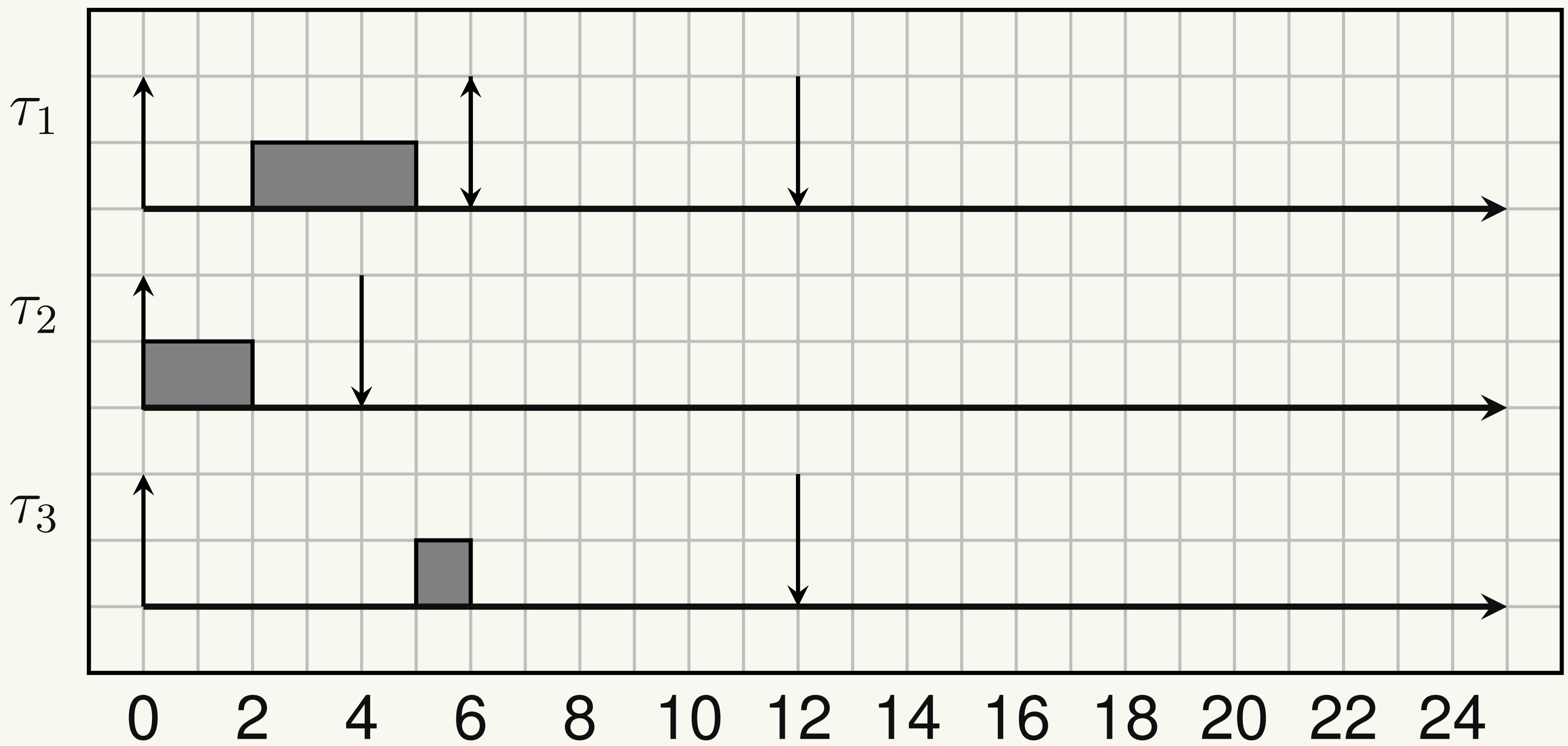
Example revised

- Consider the example shown before with deadline monotonic:
  $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



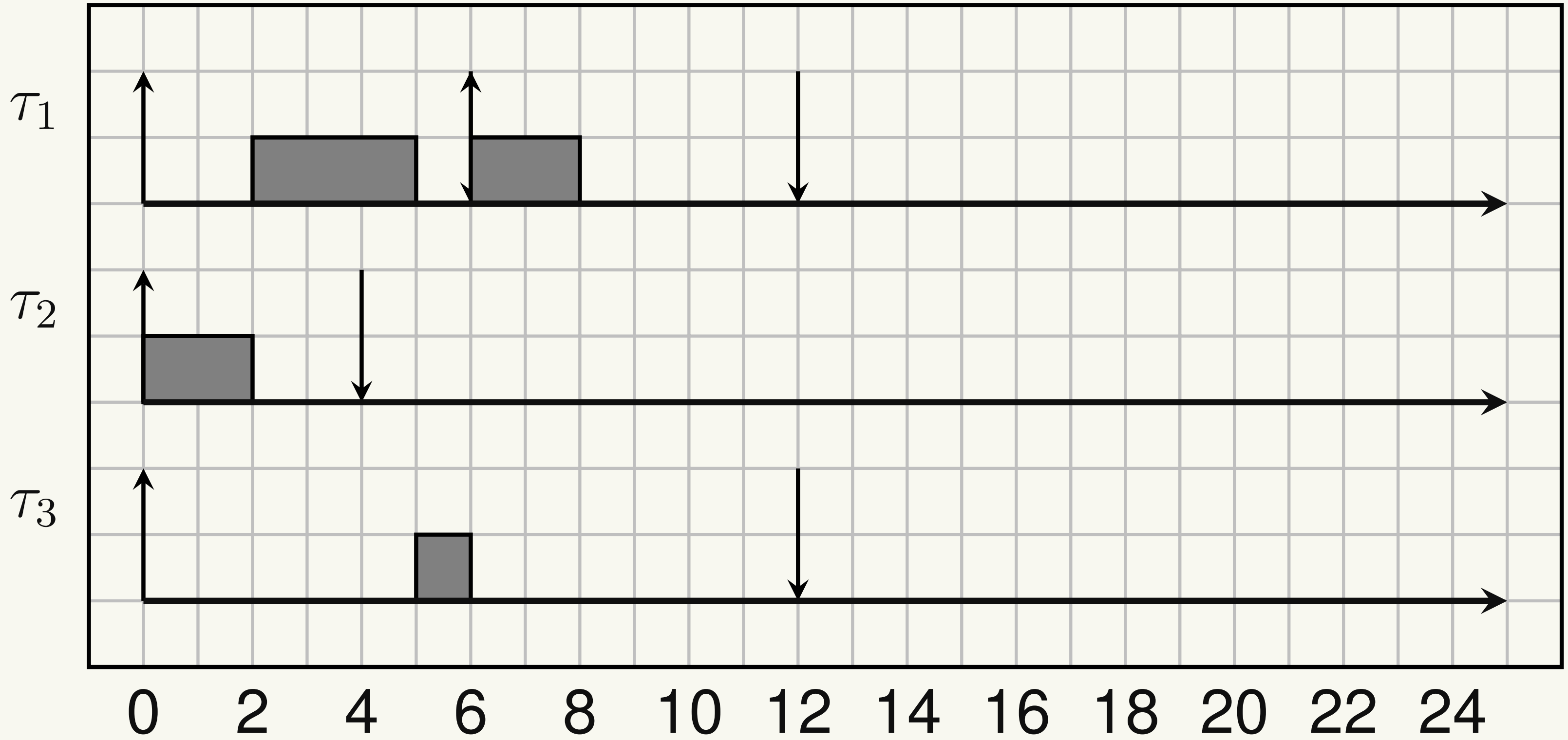
Example revised

- Consider the example shown before with deadline monotonic:
  $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



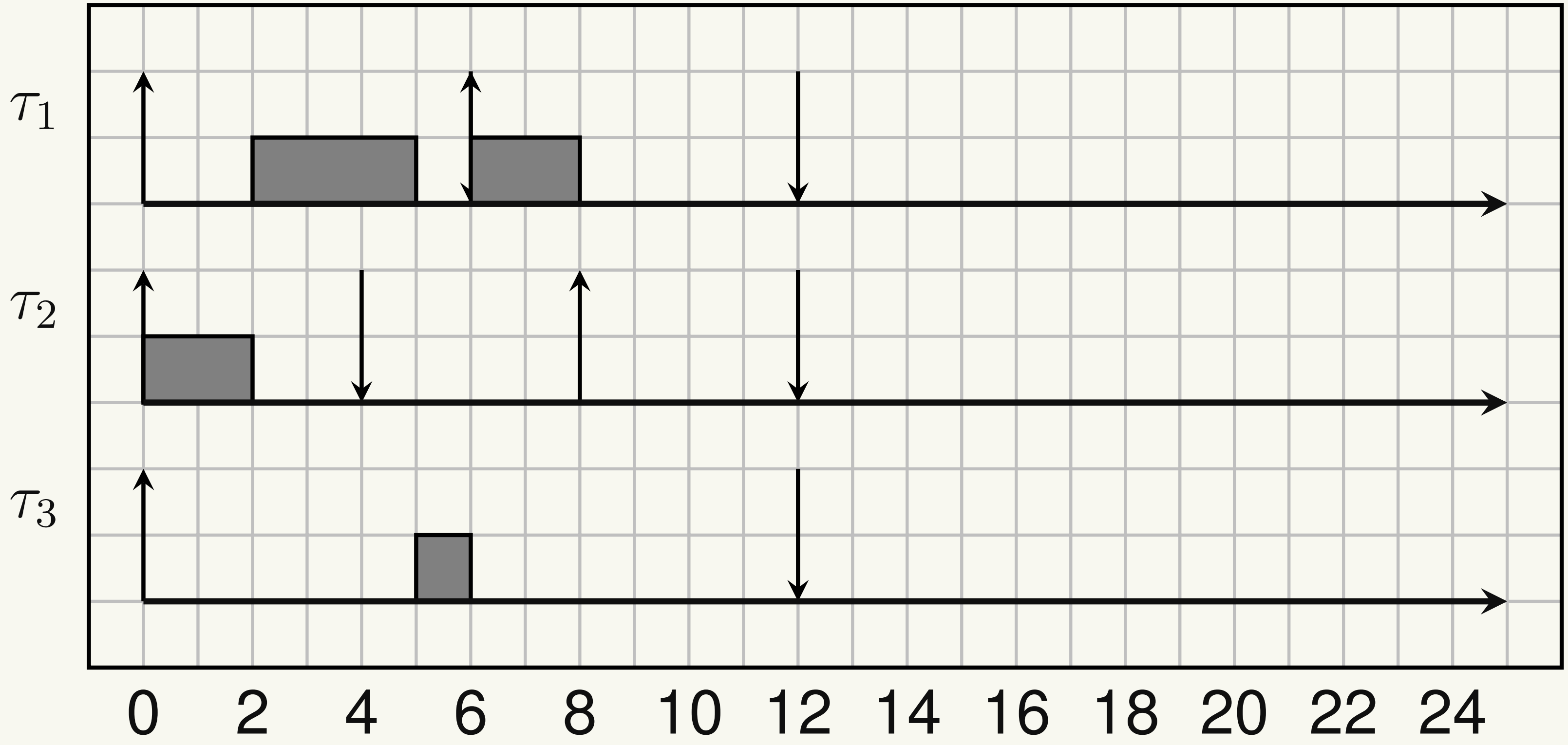
Example revised

- Consider the example shown before with deadline monotonic: $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



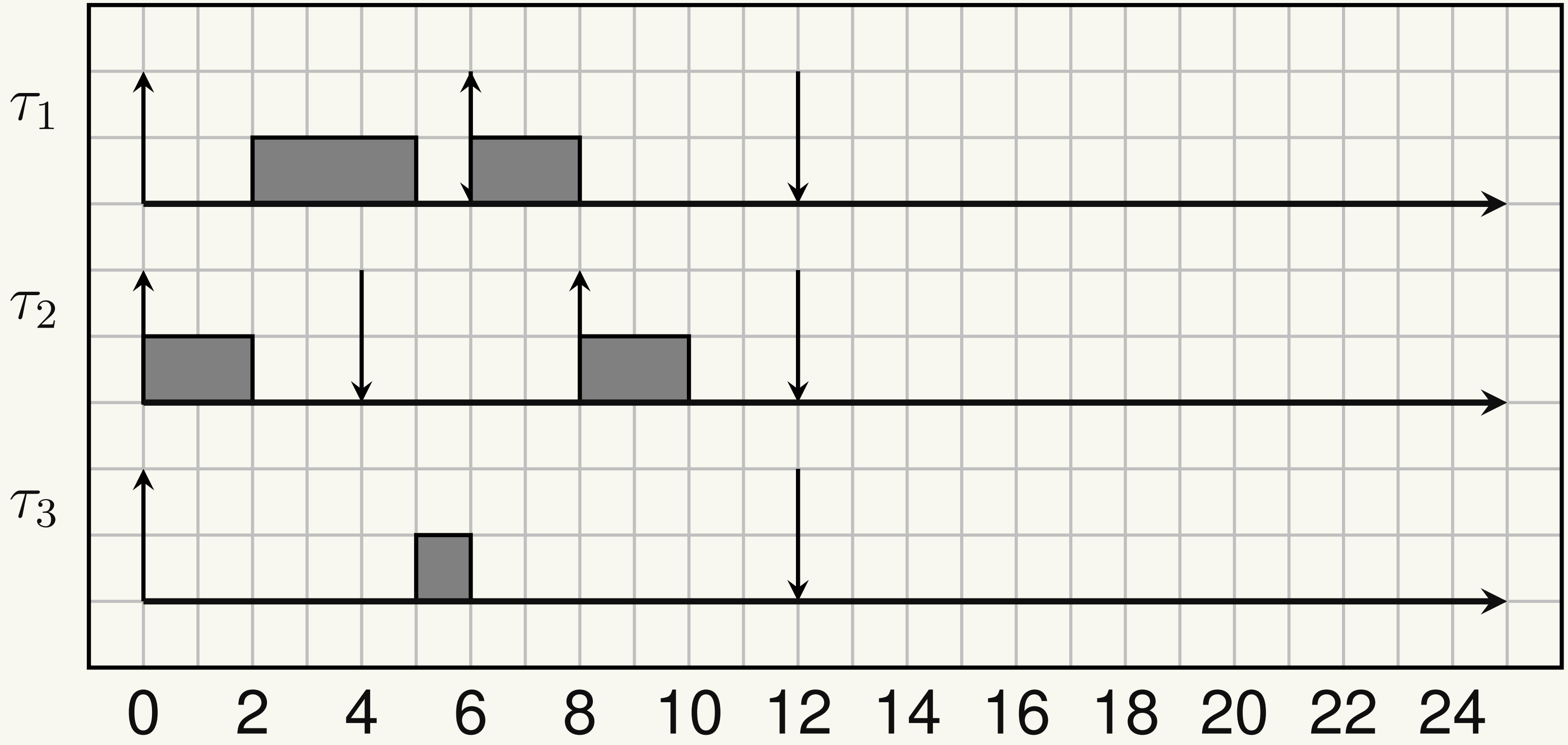
Example revised

- Consider the example shown before with deadline monotonic:
  $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



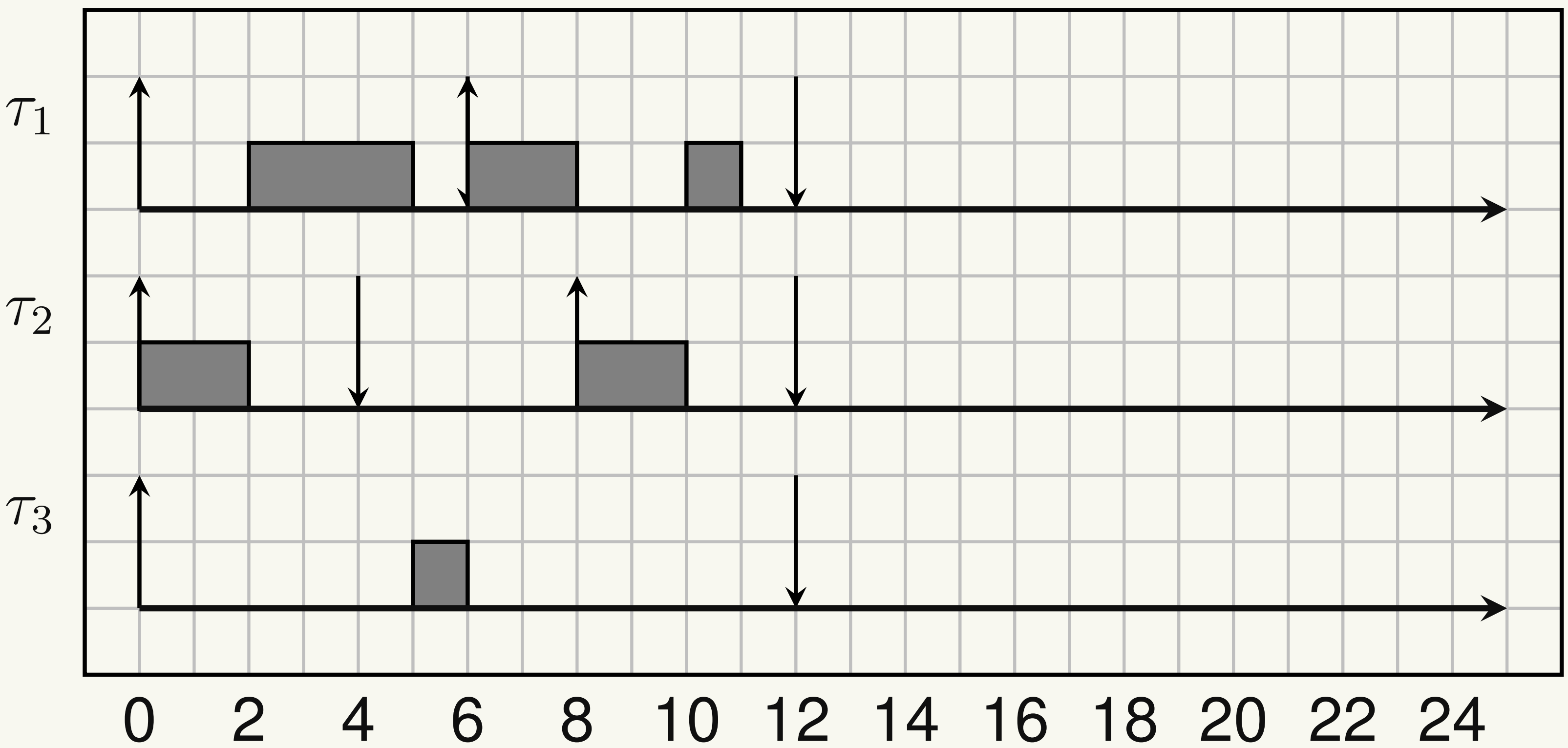
Example revised

- Consider the example shown before with deadline monotonic: $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



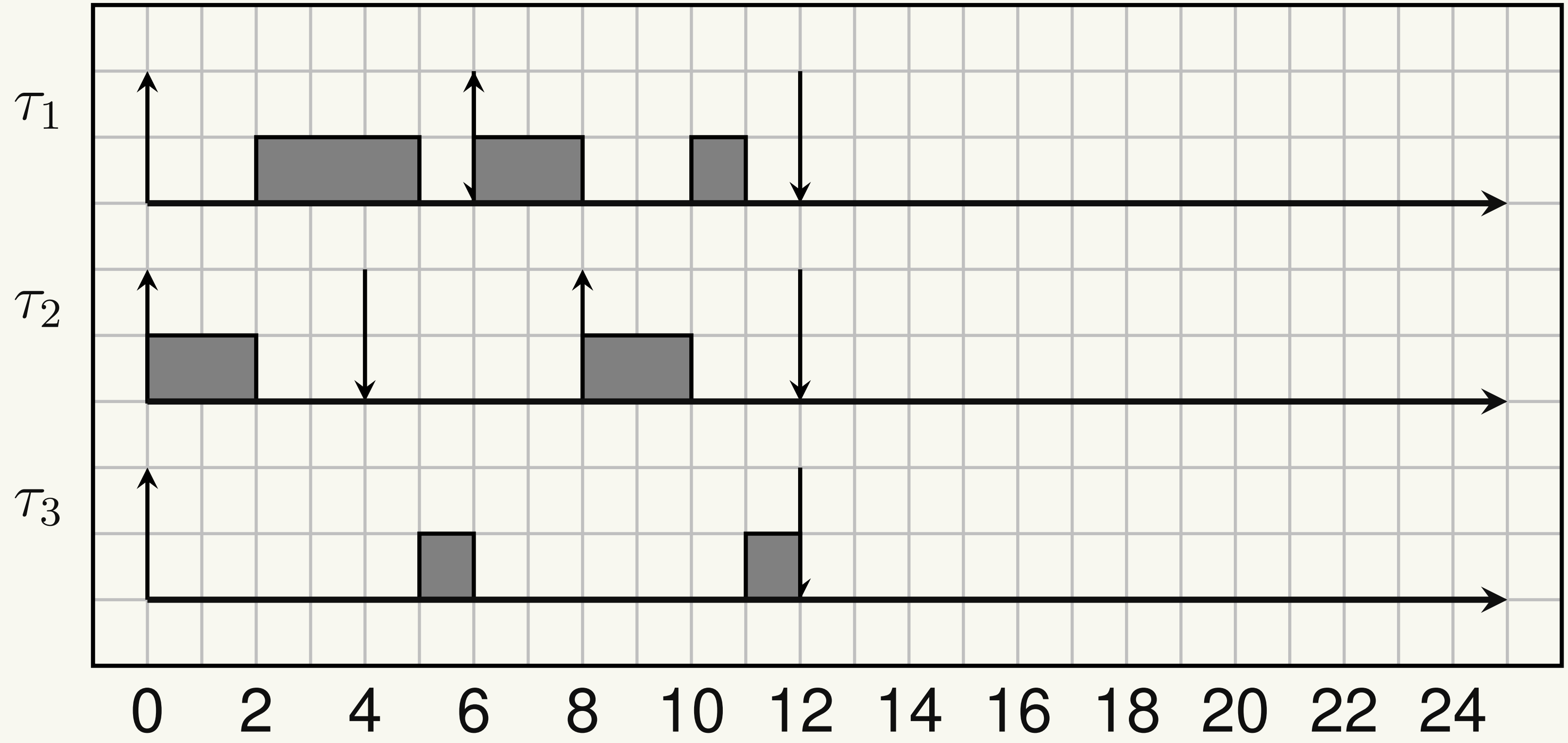
Example revised

- Consider the example shown before with deadline monotonic:
  $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



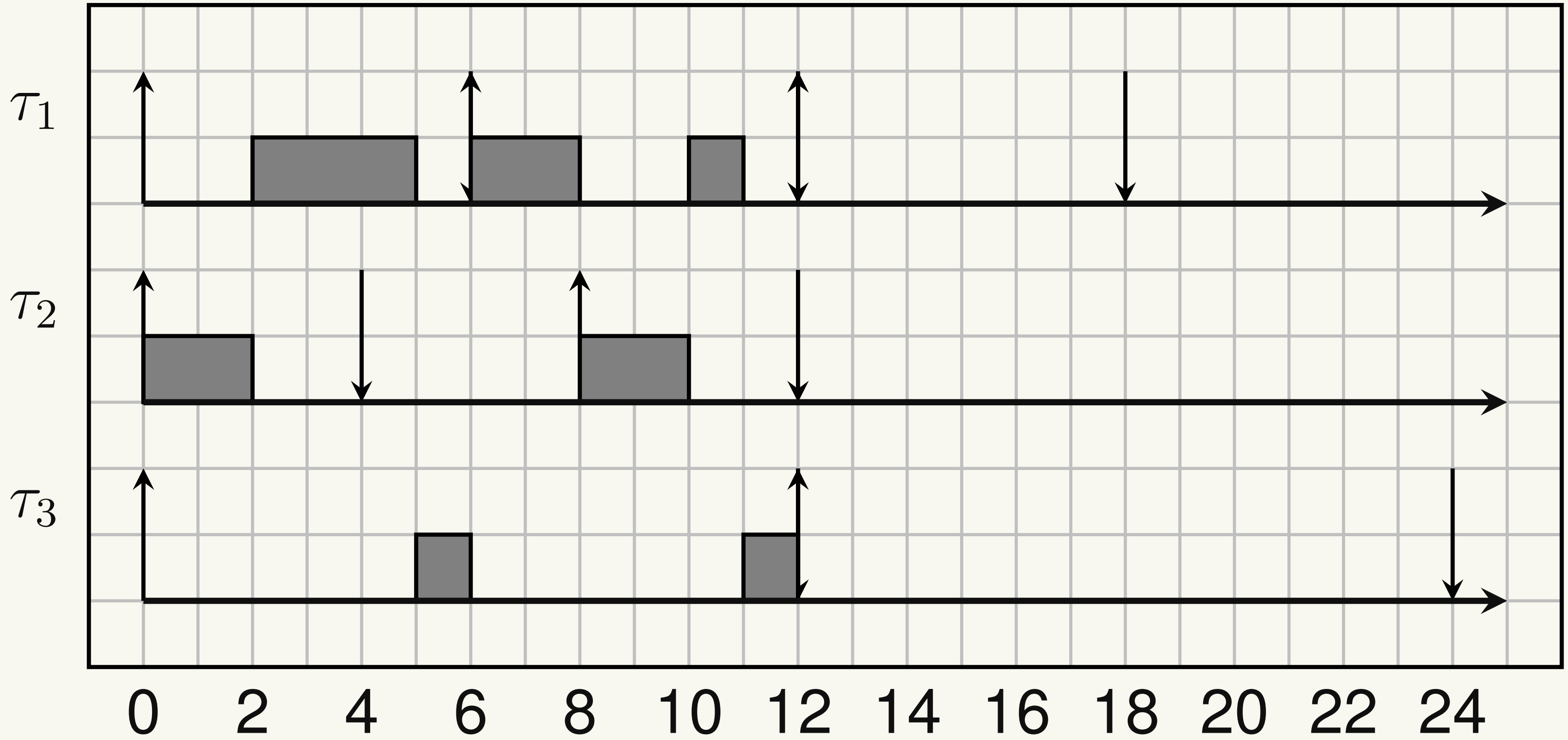
Example revised

- Consider the example shown before with deadline monotonic: $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



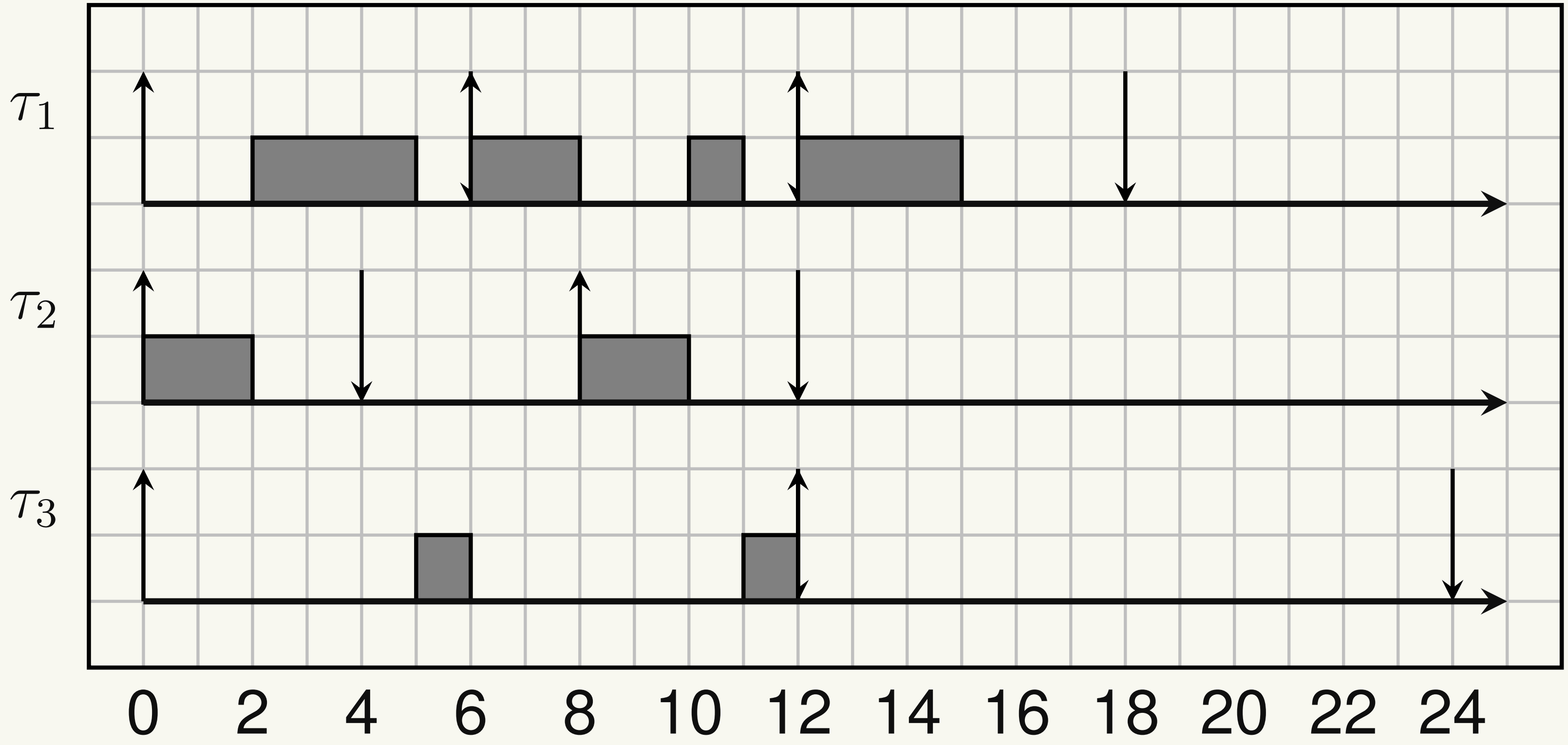
Example revised

- Consider the example shown before with deadline monotonic:
  $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



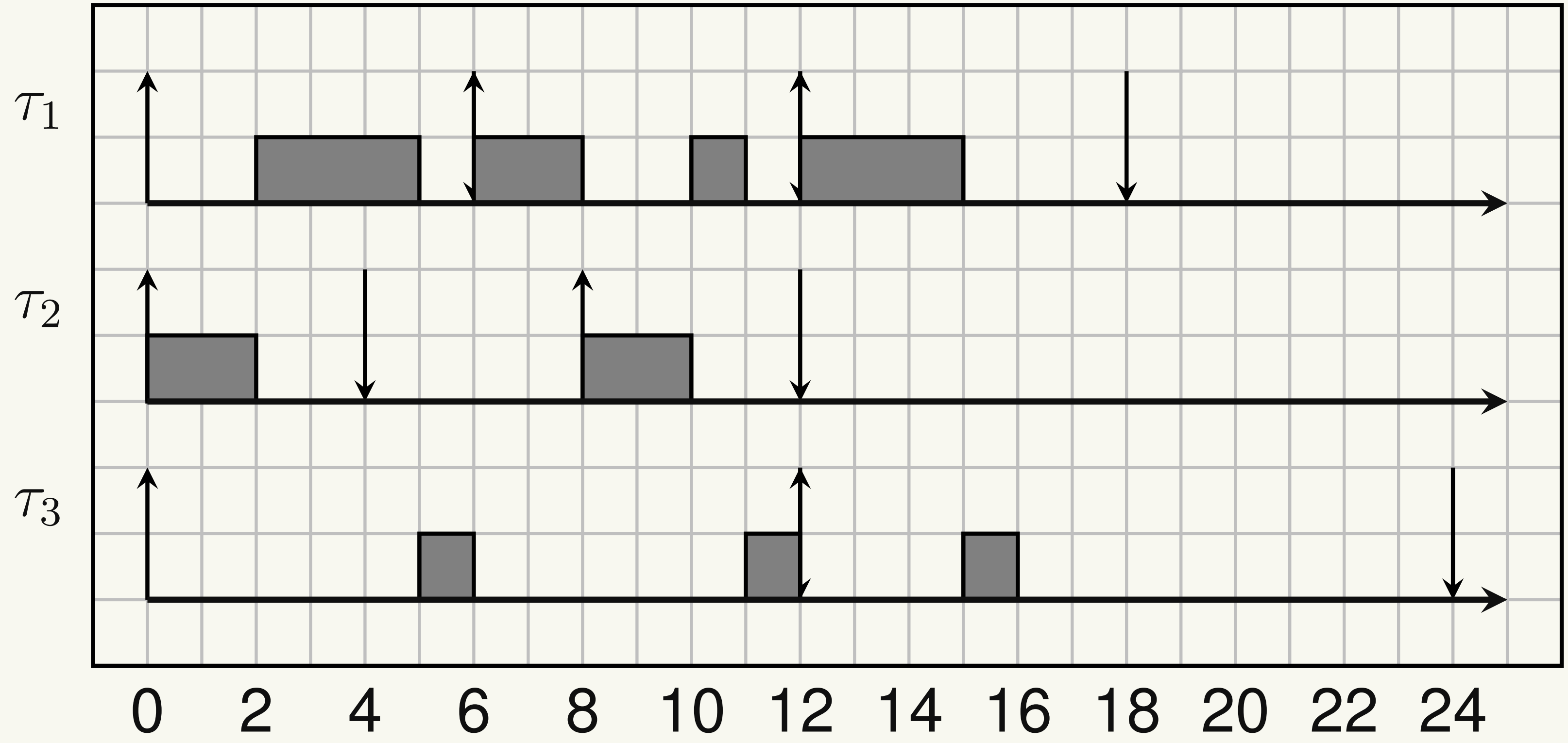
Example revised

- Consider the example shown before with deadline monotonic: $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



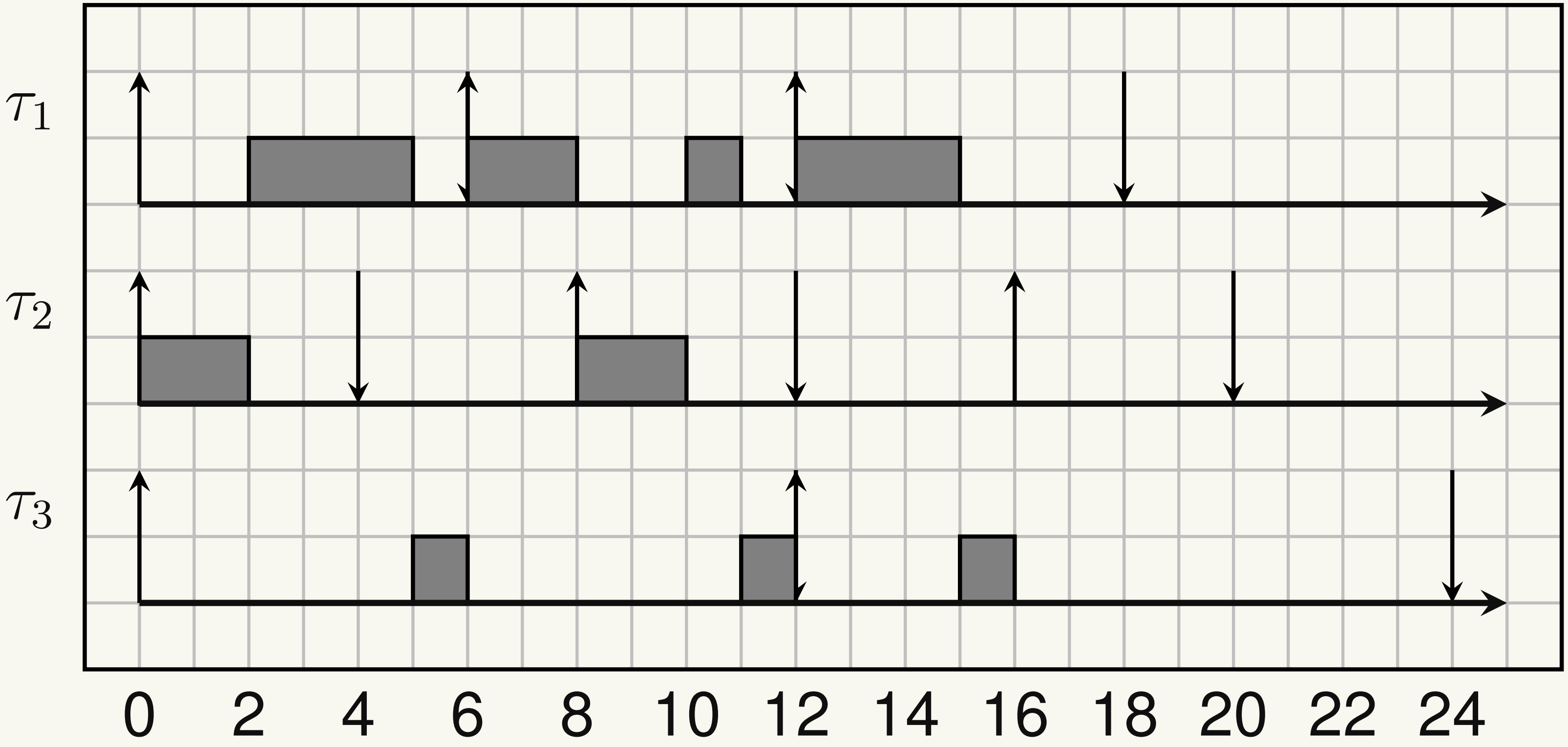
Example revised

- Consider the example shown before with deadline monotonic:
  $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



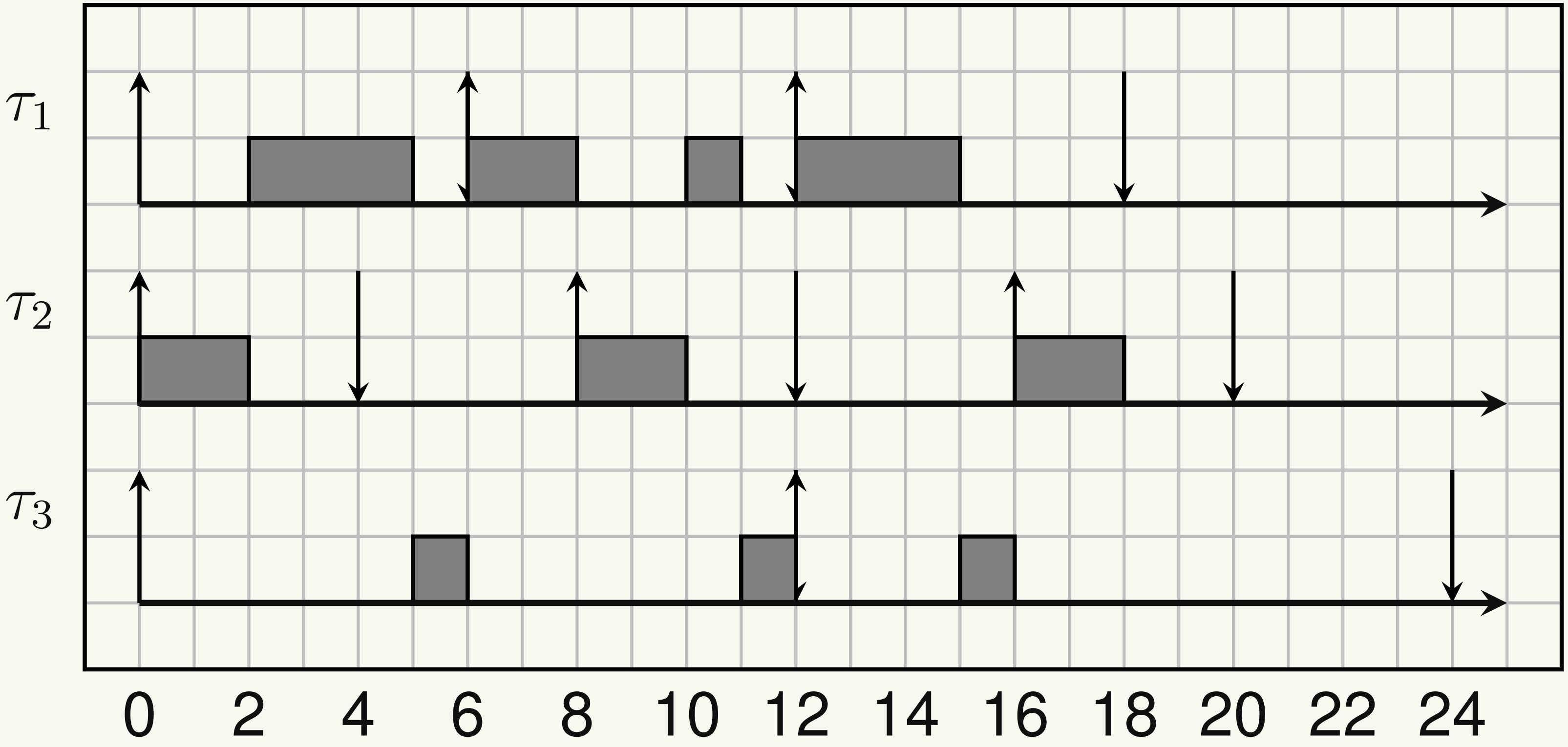
Example revised

- Consider the example shown before with deadline monotonic:
  $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



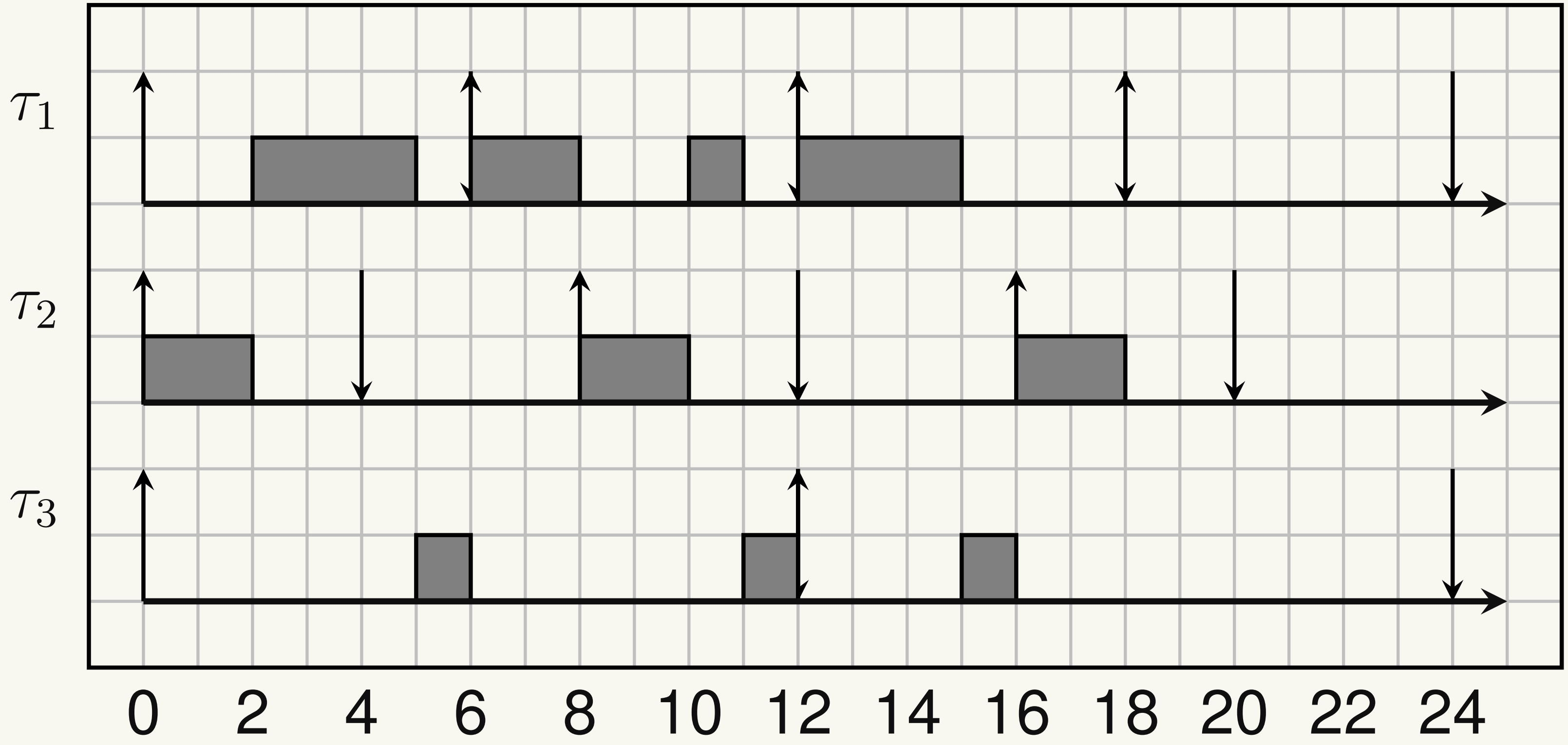
Example revised

- Consider the example shown before with deadline monotonic:
  $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



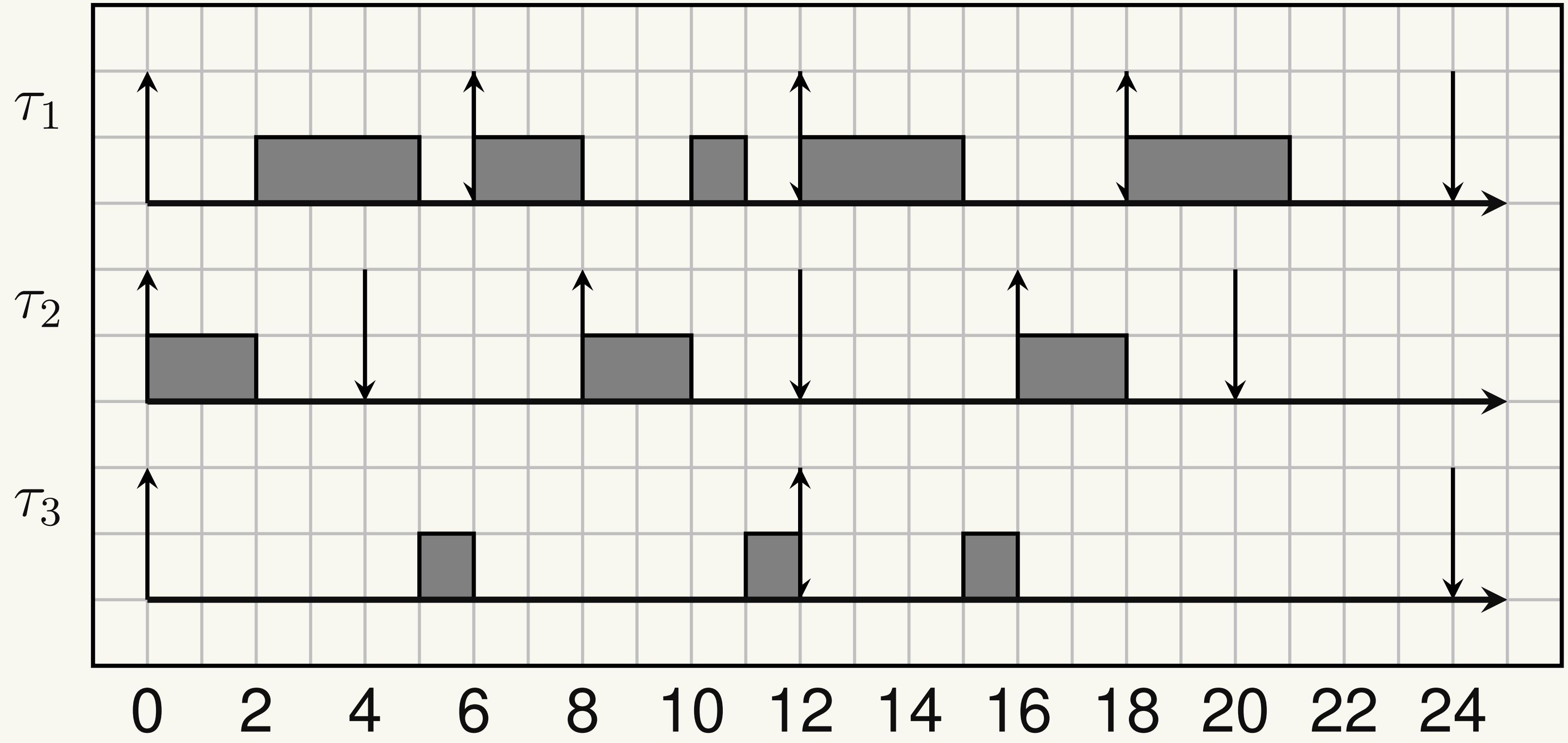
Example revised

- Consider the example shown before with deadline monotonic:
  $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



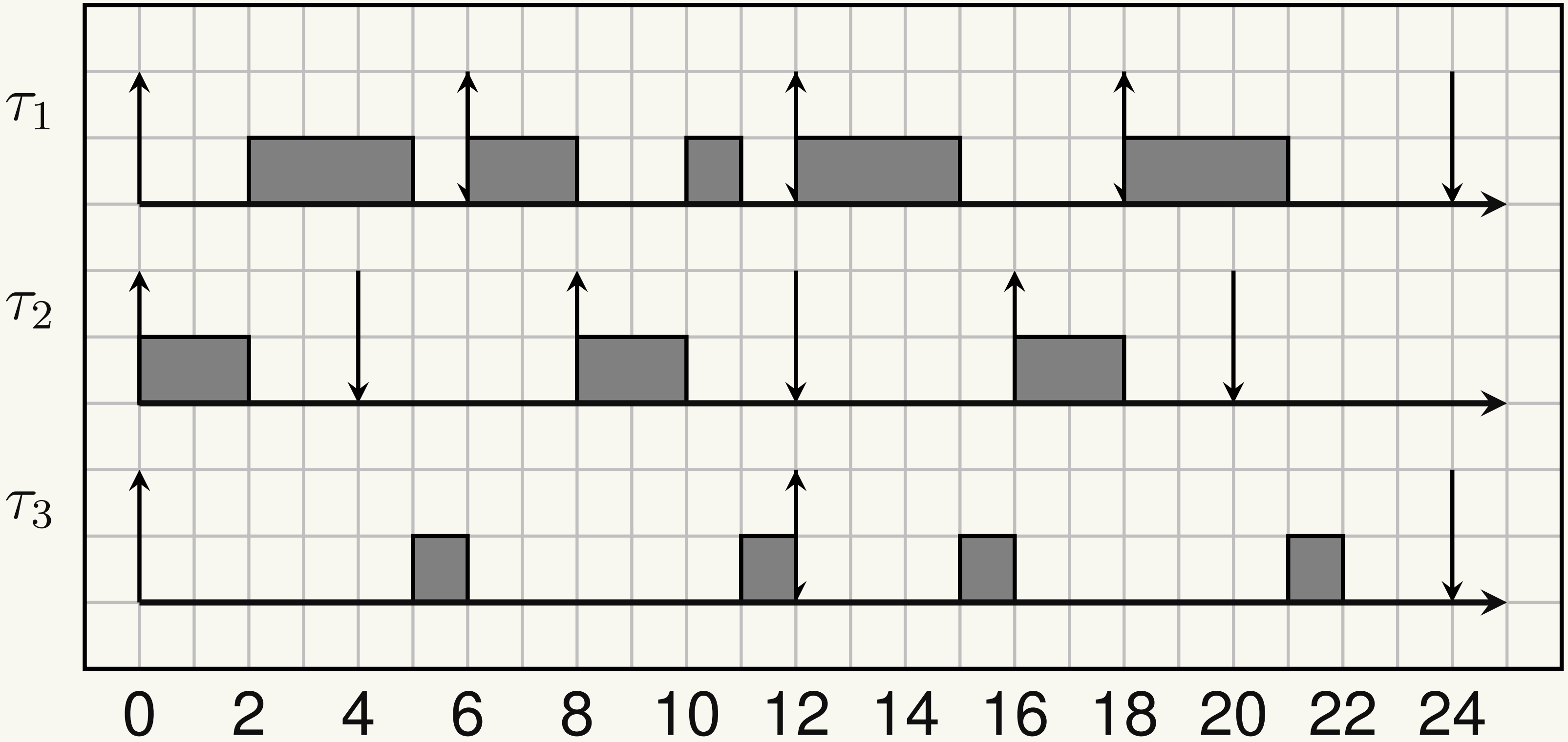
Example revised

- Consider the example shown before with deadline monotonic:
  $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



Example revised

- Consider the example shown before with deadline monotonic:
  $\tau_1 = (3, 6, 6)$, $p_1 = 2$, $\tau_2 = (2, 4, 8)$, $p_2 = 3$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



Analysis

- Given a task set, how can we guarantee if it is schedulable of not?
- The first possibility is to *simulate* the system to check that no deadline is missed;
- The execution time of every job is set equal to the WCET of the corresponding task;
  - Periodic tasks with no offsets $\Rightarrow$ sufficient to simulate the schedule until the *hyperperiod* ($H = lcm\{T_i\}$).
  - Offsets $\phi_i = r_{i,0} \Rightarrow$ simulate until $2H + \phi_{\max}$.
  - If tasks periods are prime numbers the hyperperiod can be very large!
- Note: RM $\rightarrow$ hyperperiod; Cyclic Executive $\rightarrow$ Major Cycle

Example

- Exercise: Compare the hyperperiods of this two task sets:
  - $T_1 = 8, T_2 = 12, T_3 = 24$
  - $T_1 = 7, T_2 = 12, T_3 = 25$
- In case of sporadic tasks, we can assume them to arrive at the highest possible rate, so we fall back to the case of periodic tasks with no offsets!

Utilisation-Based Analysis

- In many cases it is useful to have a very simple test to see if the task set is schedulable.
- A sufficient test is based on the *Utilisation bound*:
  - The *utilisation least upper bound* for scheduling algorithm $\mathcal{A}$ is the smallest possible utilisation U_{lub} such that, for any task set $\mathcal{T}$, if the task set's utilisation U is not greater than U_{lub} ($U \leq U_{lub}$), then the task set is schedulable by algorithm $\mathcal{A}$

Utilisation

- Each task uses the processor for a fraction of time

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

- The total processor utilisation is

$$U = \sum_i \frac{C_i}{T_i}$$

- This is a measure of the processor's load

Necessary Condition

- If $U > 1$ the task set is surely not schedulable
- However, if $U < 1$ the task set may or may not be schedulable

  $\ldots$

- If $U < U_{lub}$, the task set is schedulable!!!
  - "Gray Area" between U_{lub} and 1
  - We would like to have U_{lub} near to 1
  - $U_{lub} = 1$ would be optimal!!!

Utilisation Bound for RM

- We consider n periodic (or sporadic) tasks with relative deadline equal to periods.
- Priorities are assigned with Rate Monotonic;
- $U_{lub} = n(2^{1/n} - 1)$
  - U_{lub} is a decreasing function of n;
  - For large n: $U_{lub} \approx 0.69$

n	**U**$_{lub}$	**n**	**U**$_{lub}$
2	0.828	7	0.728
3	0.779	8	0.724
4	0.756	9	0.720
5	0.743	10	0.717
6	0.734	11	...

Schedulability Test

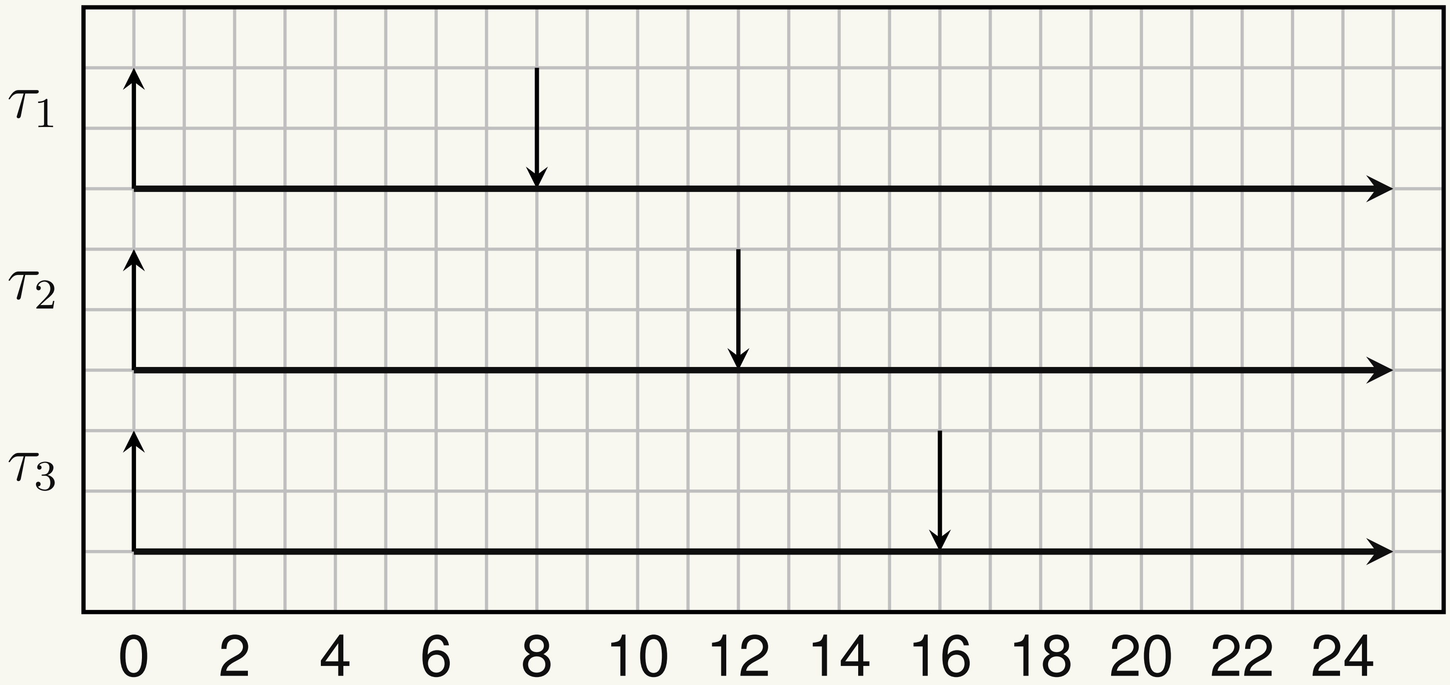
- Therefore the schedulability test consist in:
  - Computing $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$
  - if $U \leq U_{lub}$, the task set is schedulable
  - if $U > 1$ the task set is not schedulable
  - if $U_{lub} < U \leq 1$, the task set may or may not be schedulable

Example

Task set $\mathcal{T}$ composed by 3 periodic tasks with $U < U_{lub}$: the system is schedulable.

$\tau_1 = (2, 8), \tau_2 = (3, 12), \tau_3 = (4, 16);$

$$U = 0.75 < U_{lub} = 0.77$$

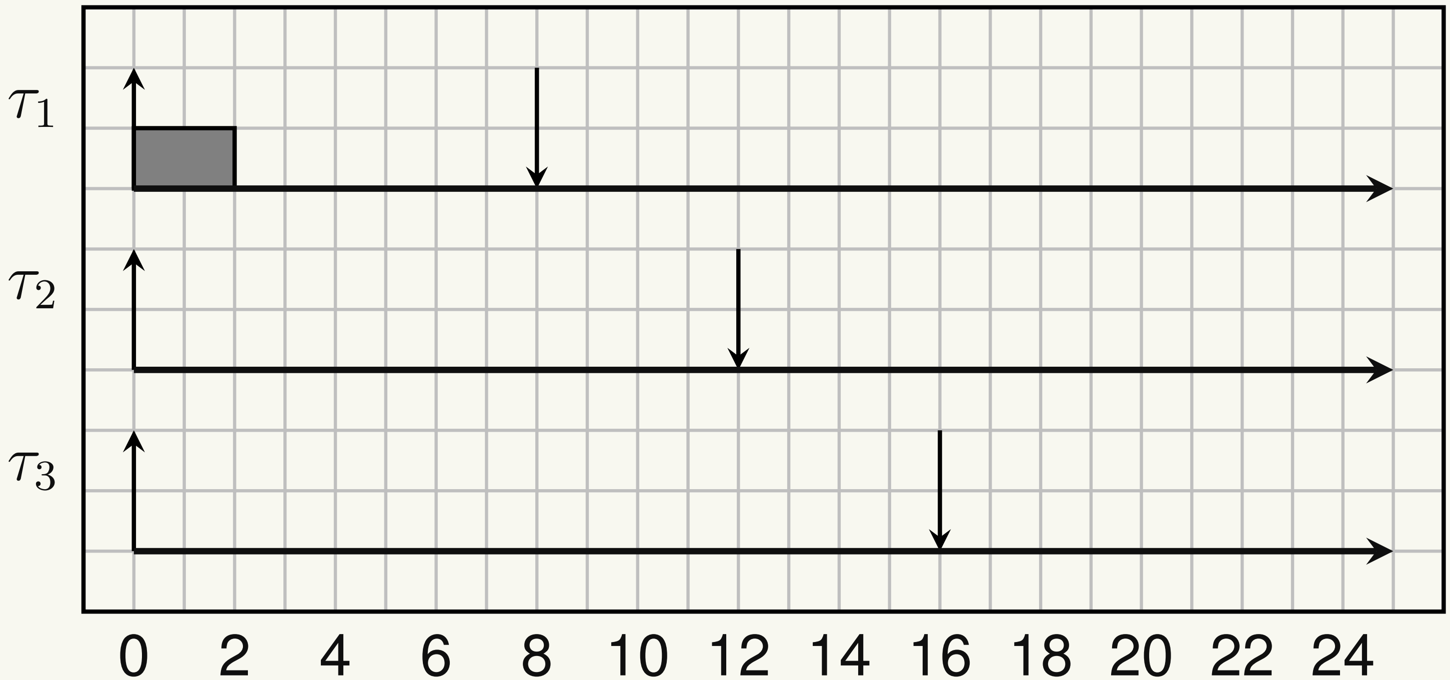


Example

Task set $\mathcal{T}$ composed by 3 periodic tasks with $U < U_{lub}$: the system is schedulable.

$\tau_1 = (2, 8), \tau_2 = (3, 12), \tau_3 = (4, 16);$

$$U = 0.75 < U_{lub} = 0.77$$

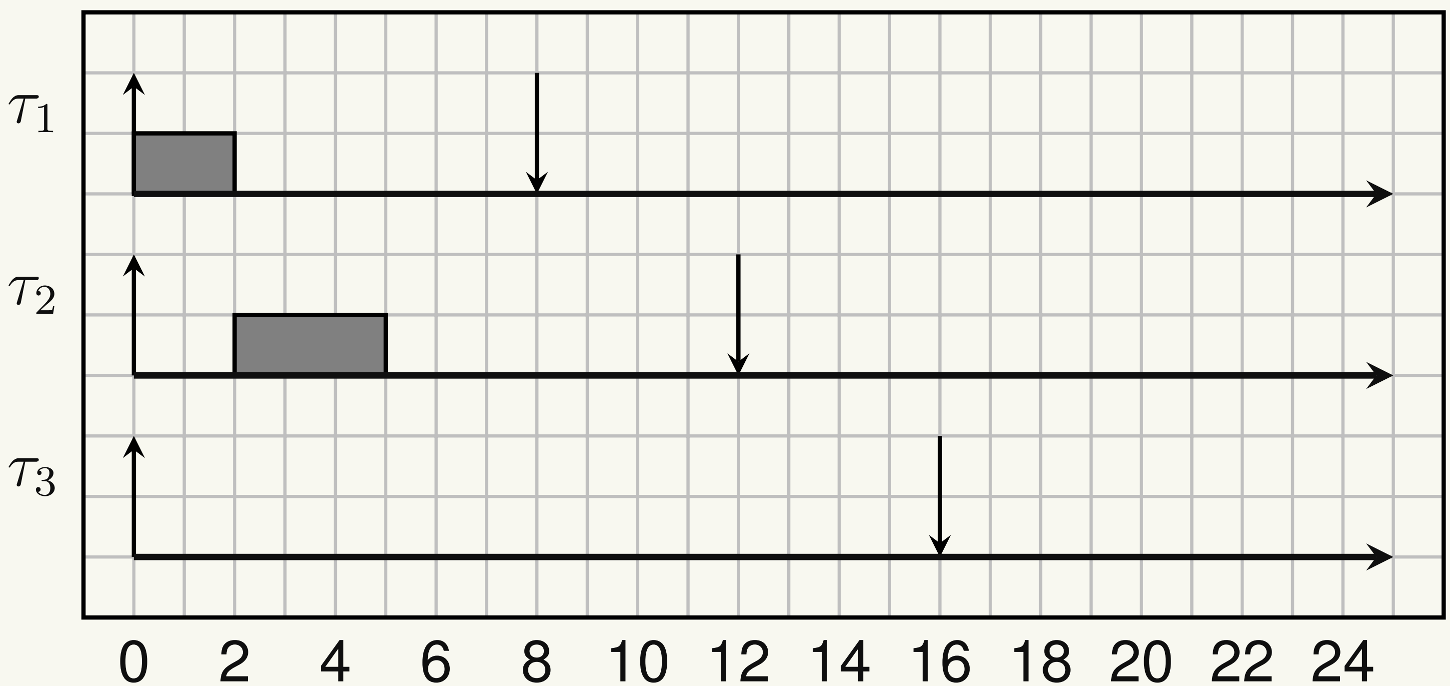


Example

Task set $\mathcal{T}$ composed by 3 periodic tasks with $U < U_{lub}$: the system is schedulable.

$\tau_1 = (2, 8), \tau_2 = (3, 12), \tau_3 = (4, 16)$;

$$U = 0.75 < U_{lub} = 0.77$$

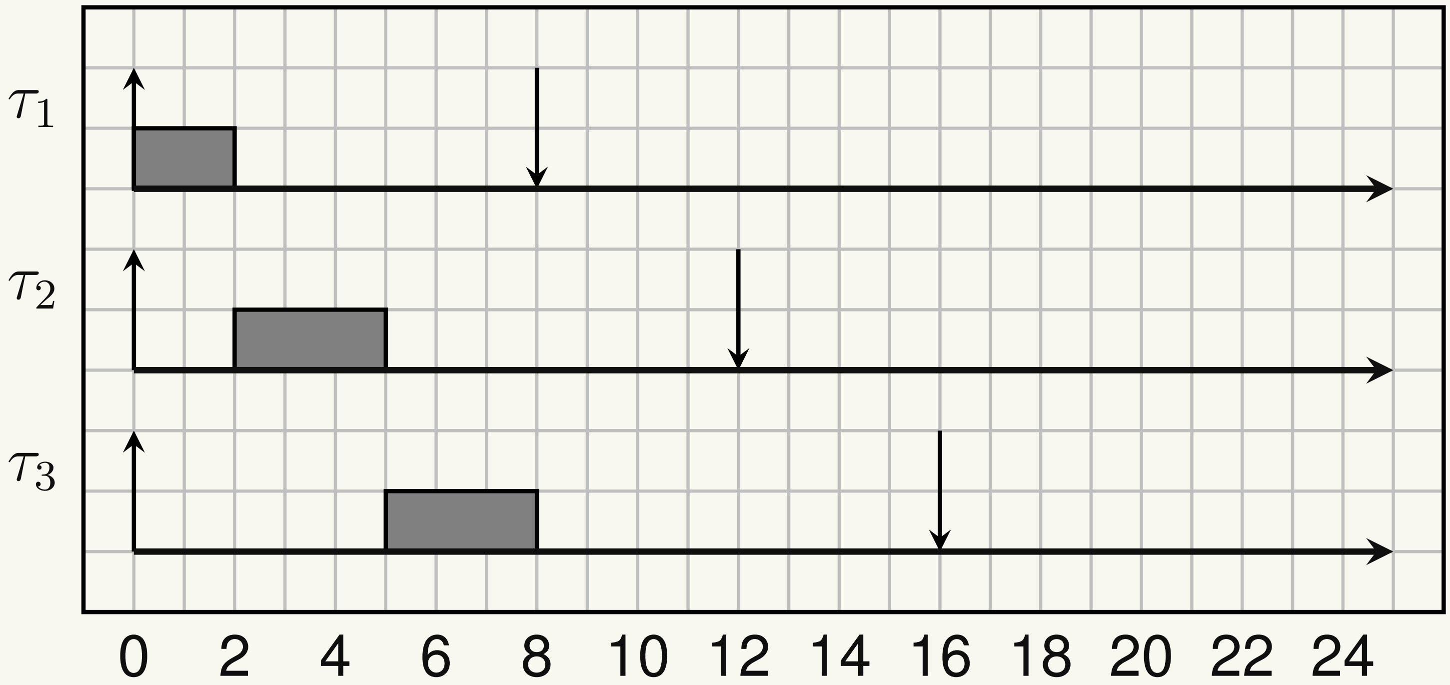


Example

Task set $\mathcal{T}$ composed by 3 periodic tasks with $U < U_{lub}$: the system is schedulable.

$\tau_1 = (2, 8), \tau_2 = (3, 12), \tau_3 = (4, 16);$

$$U = 0.75 < U_{lub} = 0.77$$

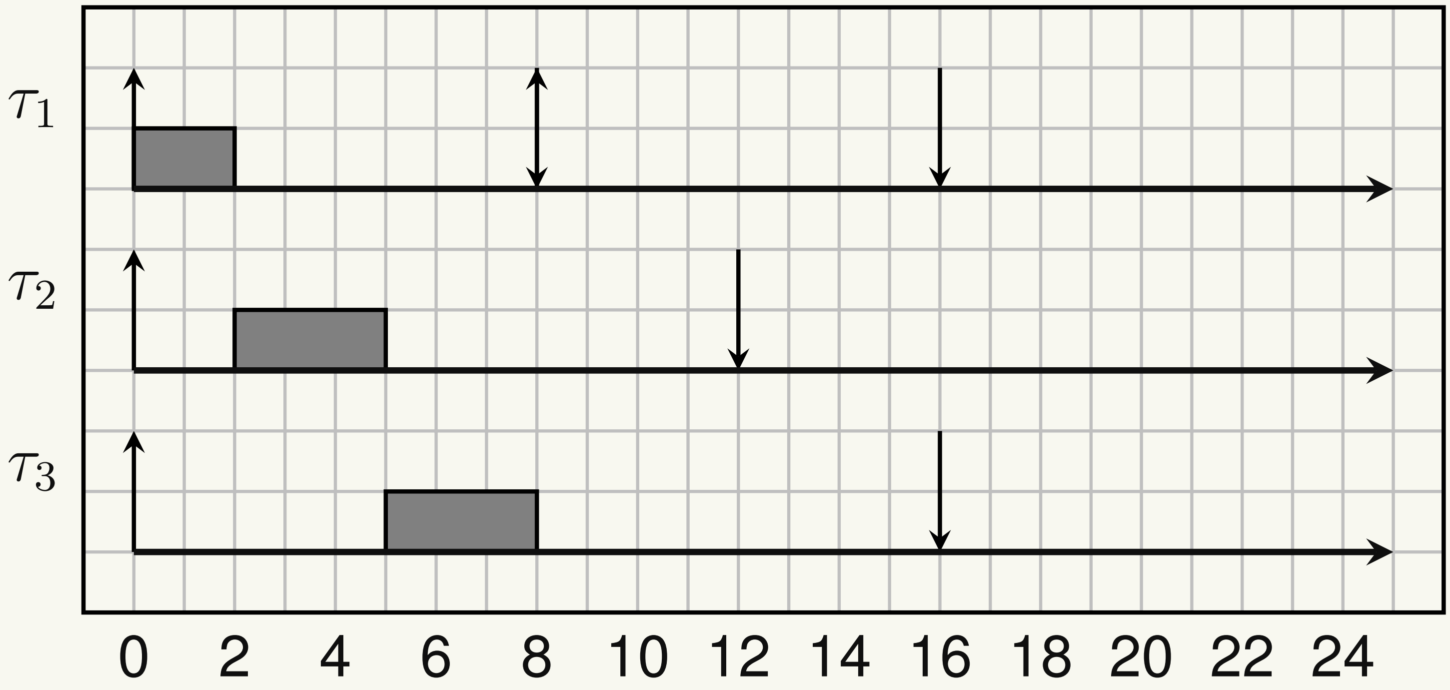


Example

Task set $\mathcal{T}$ composed by 3 periodic tasks with $U < U_{lub}$: the system is schedulable.

$\tau_1 = (2, 8), \tau_2 = (3, 12), \tau_3 = (4, 16);$

$$U = 0.75 < U_{lub} = 0.77$$

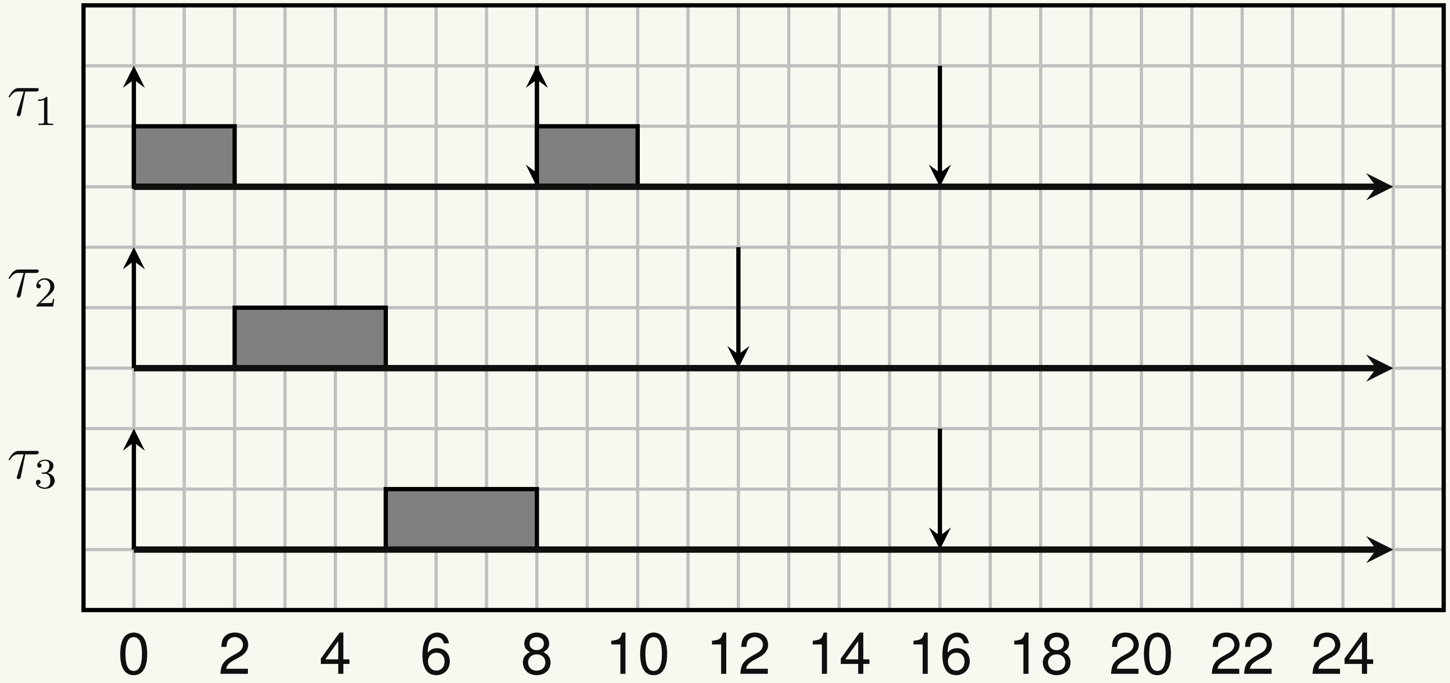


Example

Task set $\mathcal{T}$ composed by 3 periodic tasks with $U < U_{lub}$: the system is schedulable.

$\tau_1 = (2, 8), \tau_2 = (3, 12), \tau_3 = (4, 16);$

$$U = 0.75 < U_{lub} = 0.77$$

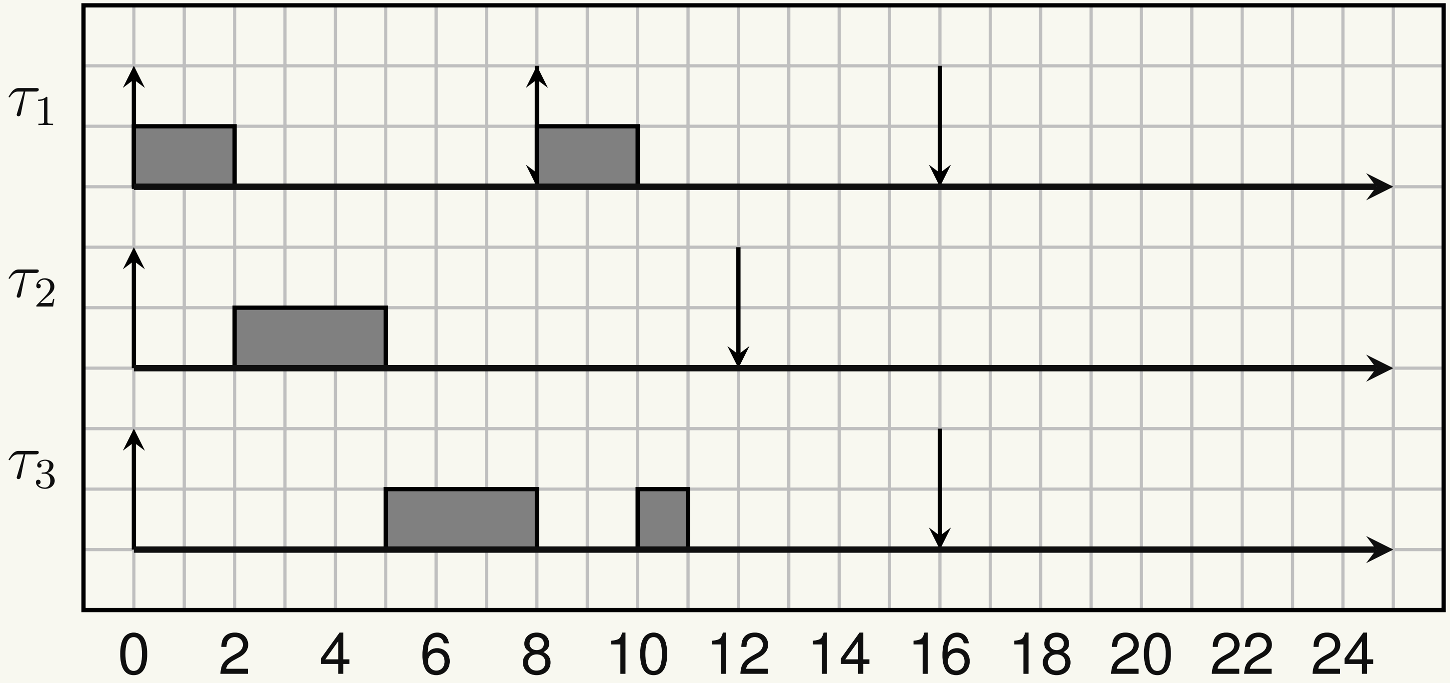


Example

Task set $\mathcal{T}$ composed by 3 periodic tasks with $U < U_{lub}$: the system is schedulable.

$\tau_1 = (2, 8), \tau_2 = (3, 12), \tau_3 = (4, 16);$

$$U = 0.75 < U_{lub} = 0.77$$

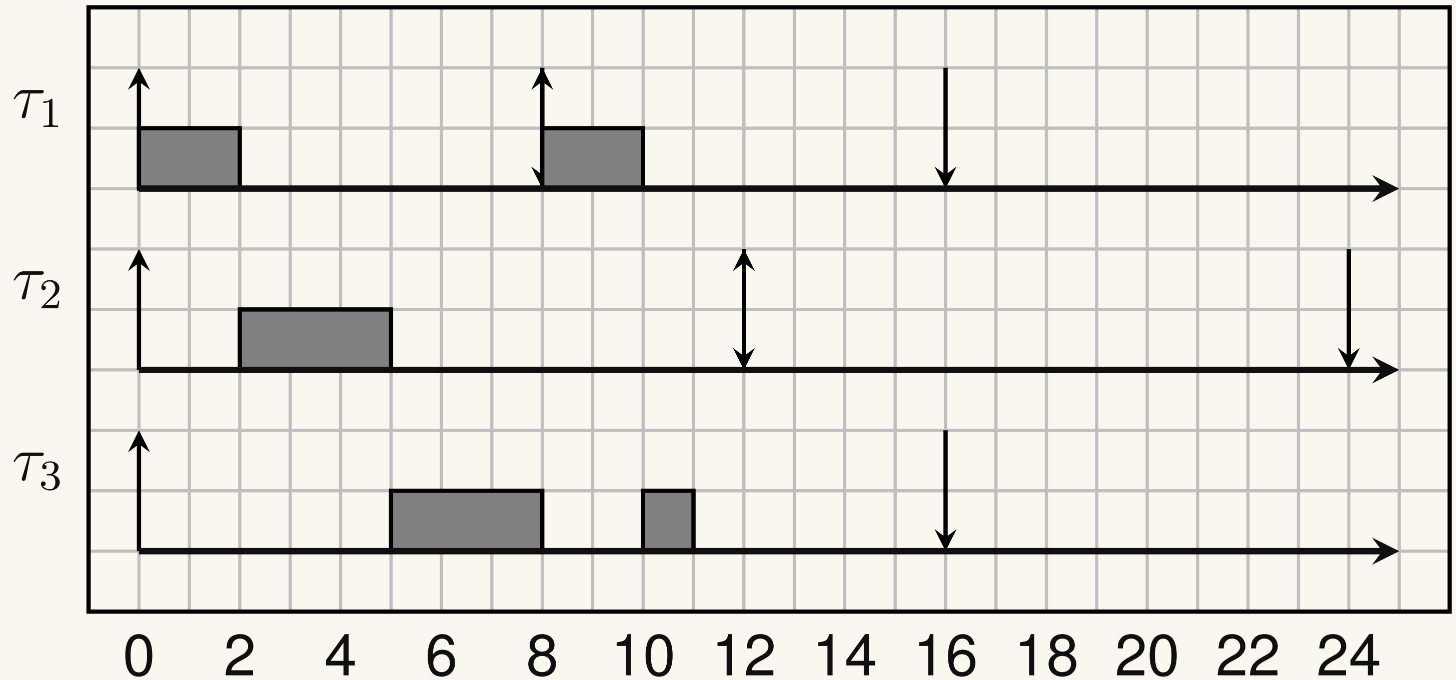


Example

Task set $\mathcal{T}$ composed by 3 periodic tasks with $U < U_{lub}$: the system is schedulable.

$\tau_1 = (2, 8), \tau_2 = (3, 12), \tau_3 = (4, 16);$

$$U = 0.75 < U_{lub} = 0.77$$

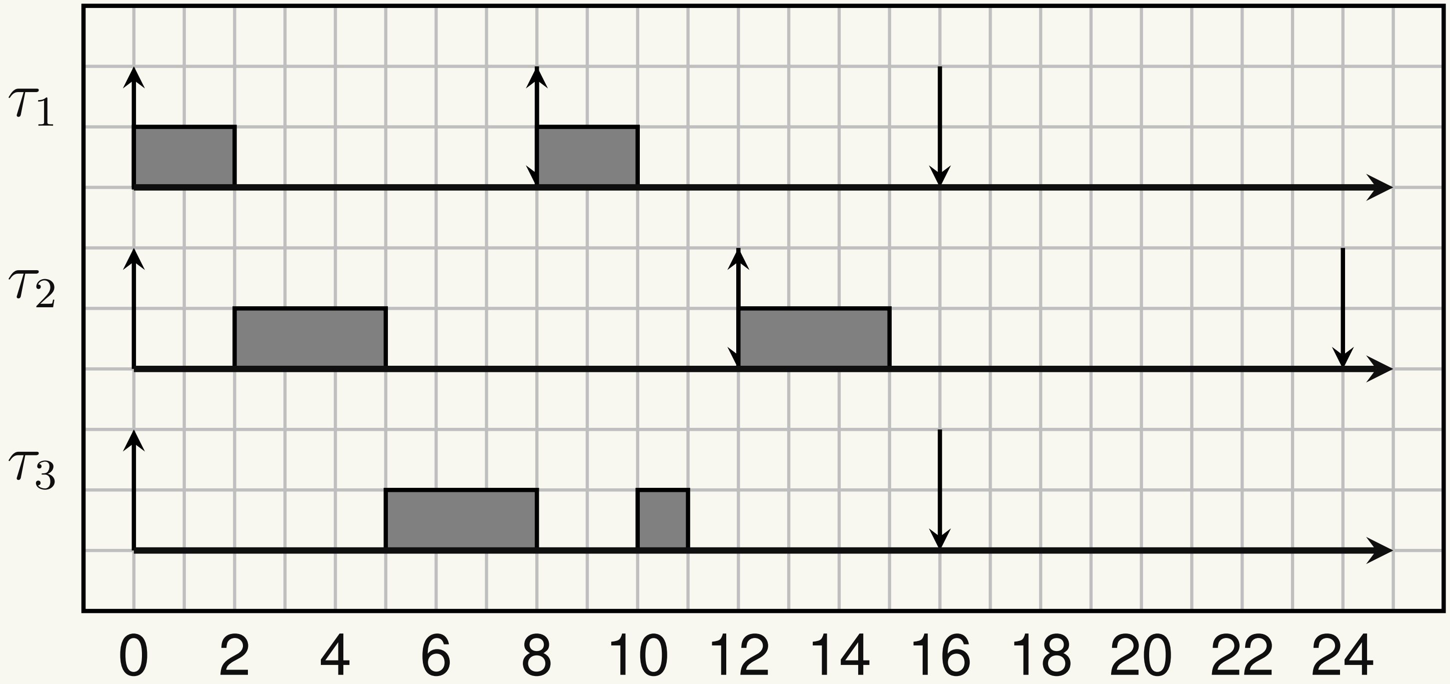


Example

Task set $\mathcal{T}$ composed by 3 periodic tasks with $U < U_{lub}$: the system is schedulable.

$\tau_1 = (2, 8), \tau_2 = (3, 12), \tau_3 = (4, 16);$

$$U = 0.75 < U_{lub} = 0.77$$

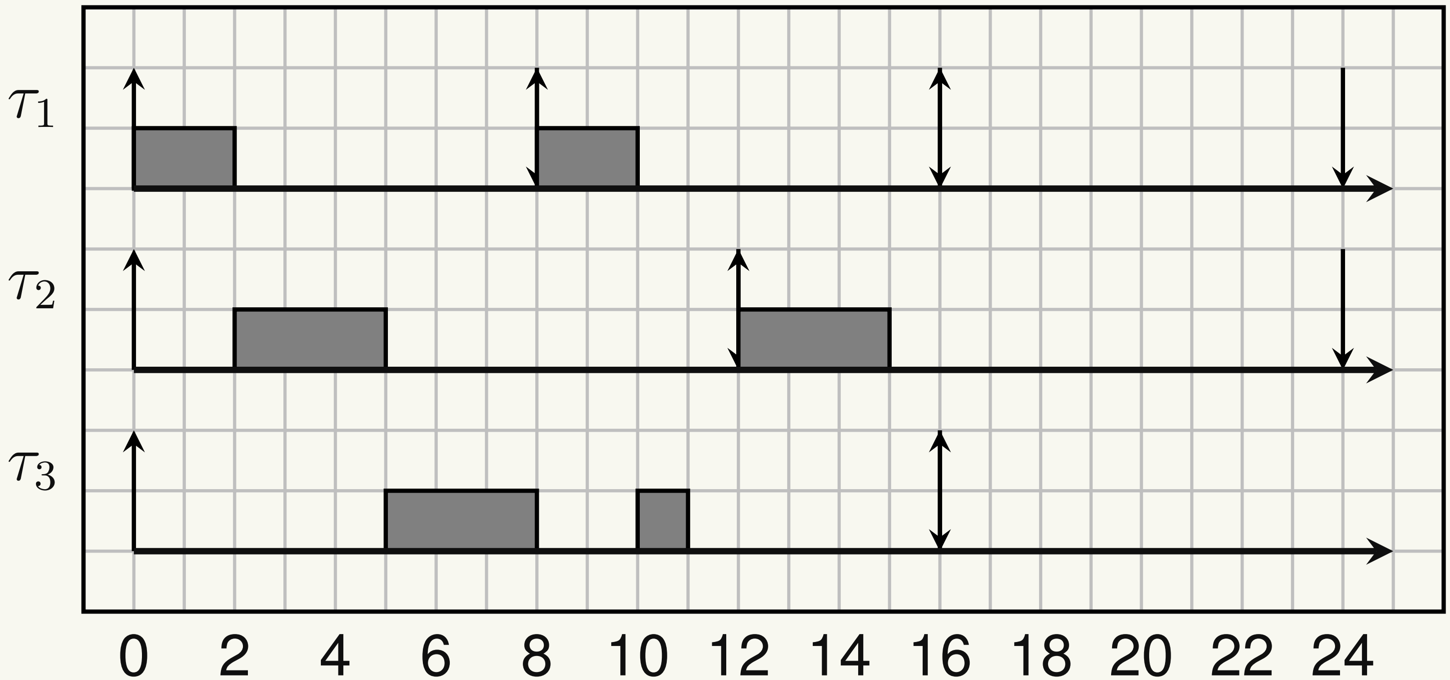


Example

Task set $\mathcal{T}$ composed by 3 periodic tasks with $U < U_{lub}$: the system is schedulable.

$\tau_1 = (2, 8), \tau_2 = (3, 12), \tau_3 = (4, 16);$

$$U = 0.75 < U_{lub} = 0.77$$

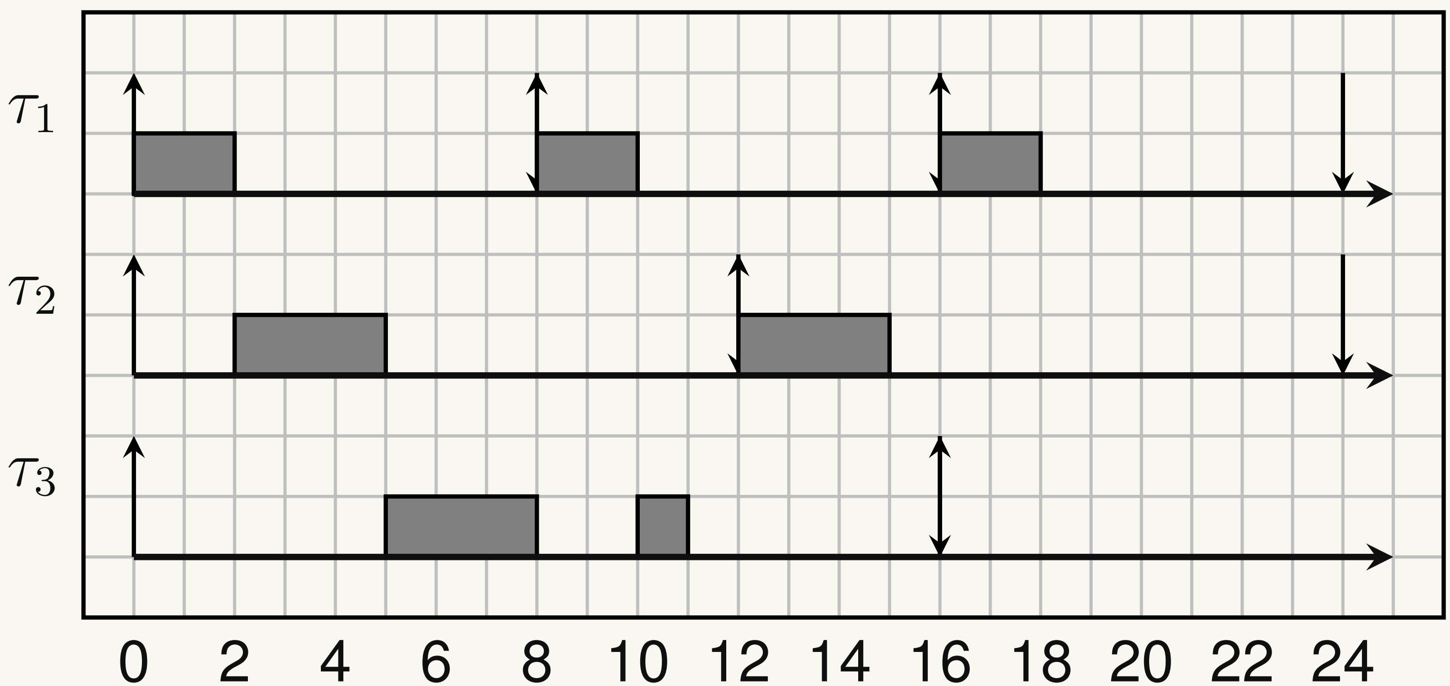


Example

Task set $\mathcal{T}$ composed by 3 periodic tasks with $U < U_{lub}$: the system is schedulable.

$\tau_1 = (2, 8), \tau_2 = (3, 12), \tau_3 = (4, 16);$

$$U = 0.75 < U_{lub} = 0.77$$

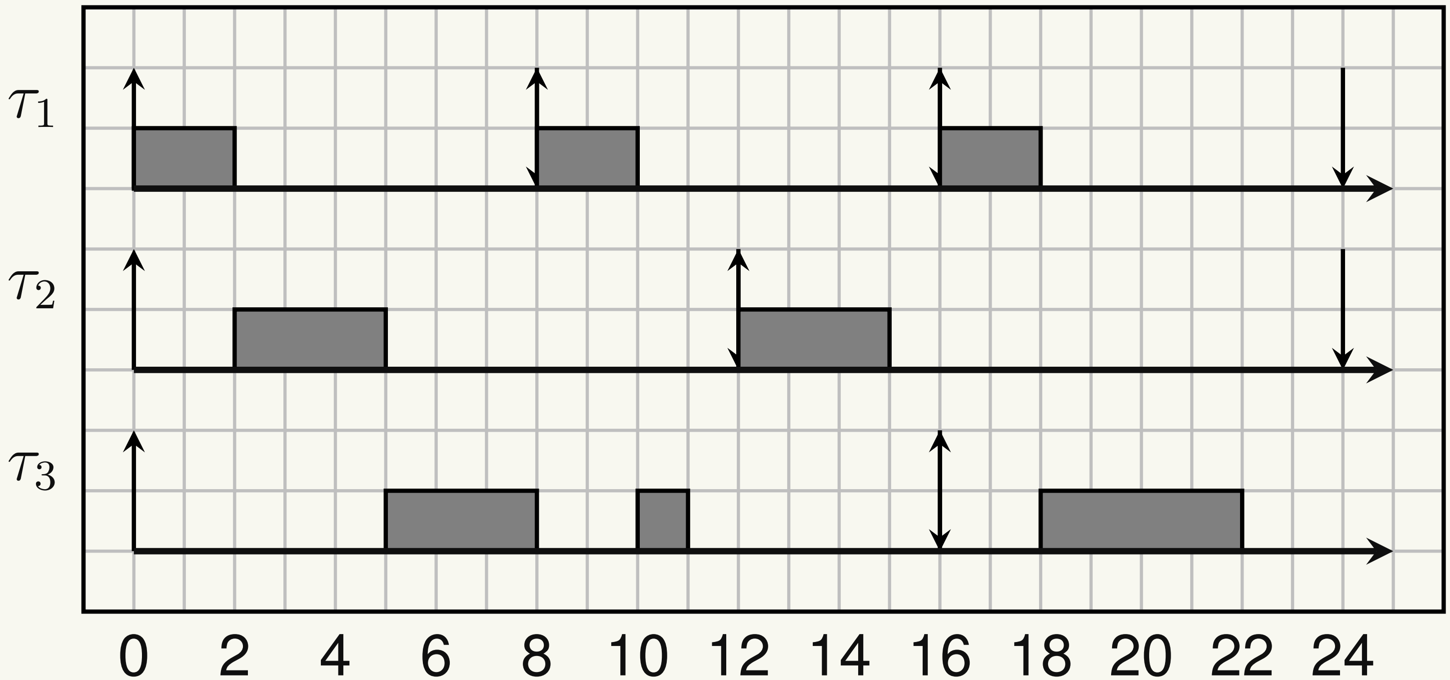


Example

Task set $\mathcal{T}$ composed by 3 periodic tasks with $U < U_{lub}$: the system is schedulable.

$\tau_1 = (2, 8), \tau_2 = (3, 12), \tau_3 = (4, 16);$

$$U = 0.75 < U_{lub} = 0.77$$

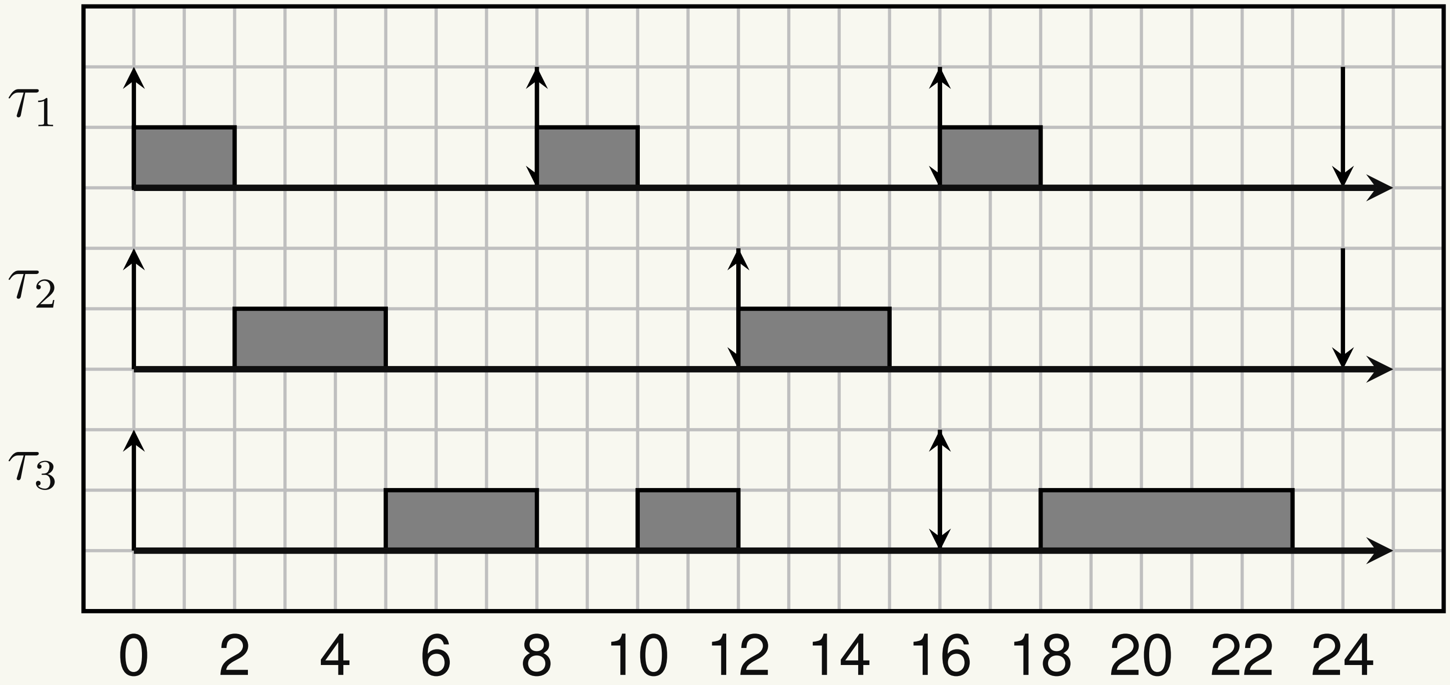


Example 2

By increasing the computation time of task τ_3, the system may still be schedulable

$\tau_1 = (2, 8), \tau_2 = (3, 12), \tau_3 = (5, 16);$

$$U = 0.81 > U_{lub} = 0.77$$



Utilisation Bound for DM

- If relative deadlines are less than or equal to periods, instead of considering $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$, we can consider:

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

- Then the test is the same as the one for RM (or DM), except that we must use U' instead of U.
- Idea: $\tau = (C, D, T) \rightarrow \tau' = (C, D, D)$
  - τ' is a "worst case" for τ
  - If τ' can be guaranteed, τ can be guaranteed too

Pessimism

- The bound is very pessimistic: most of the times, a task set with $U > U_{lub}$ is schedulable by RM.
- A particular case is when tasks have periods that are *harmonic*:
  - A task set is *harmonic* if, for every two tasks τ_i, τ_j, either T_i is multiple of T_j or T_j is multiple of T_i.
- For a harmonic task set, the utilisation bound is $U_{lub} = 1$
- In other words, Rate Monotonic is an *optimal* algorithm for harmonic task sets

Example of Harmonic Task Set

$\tau_1 = (3, 6)$, $\tau_2 = (3, 12)$, $\tau_3 = (6, 24)$;

$U = 1$;