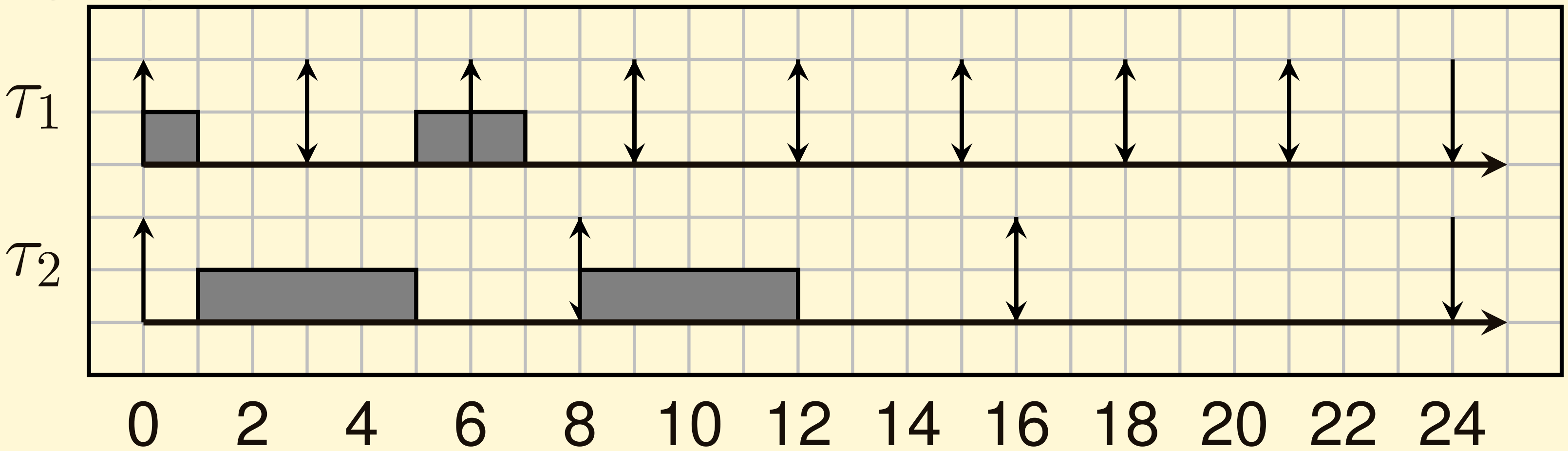
Real-Time Operating Systems

Luca Abeni
luca.abeni@santannapisa.it

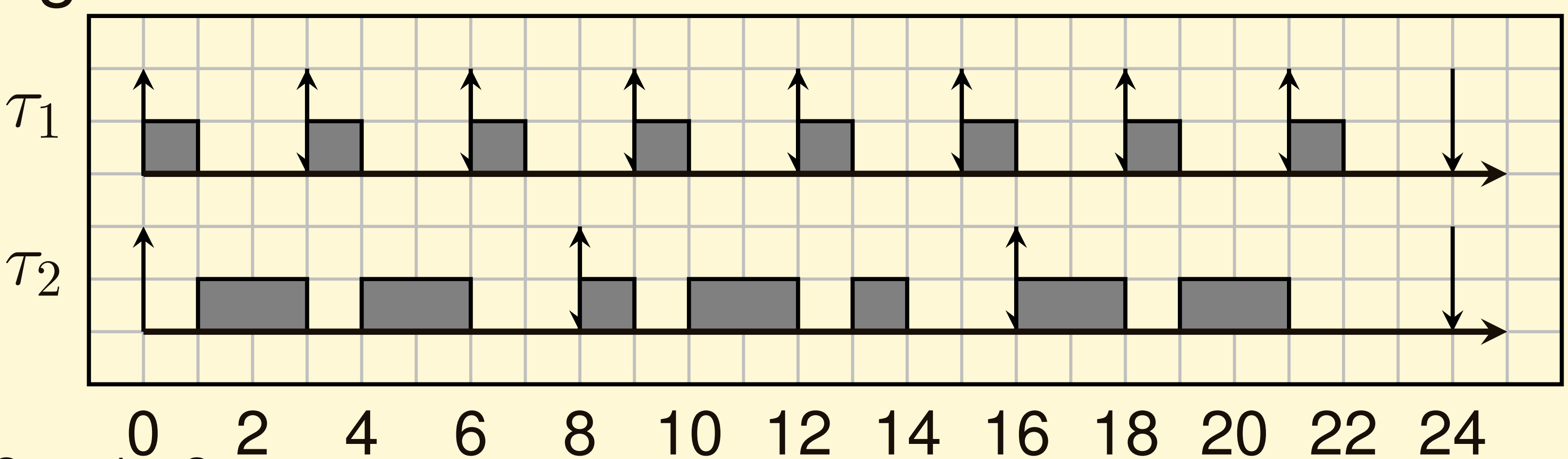
March 13, 2018

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

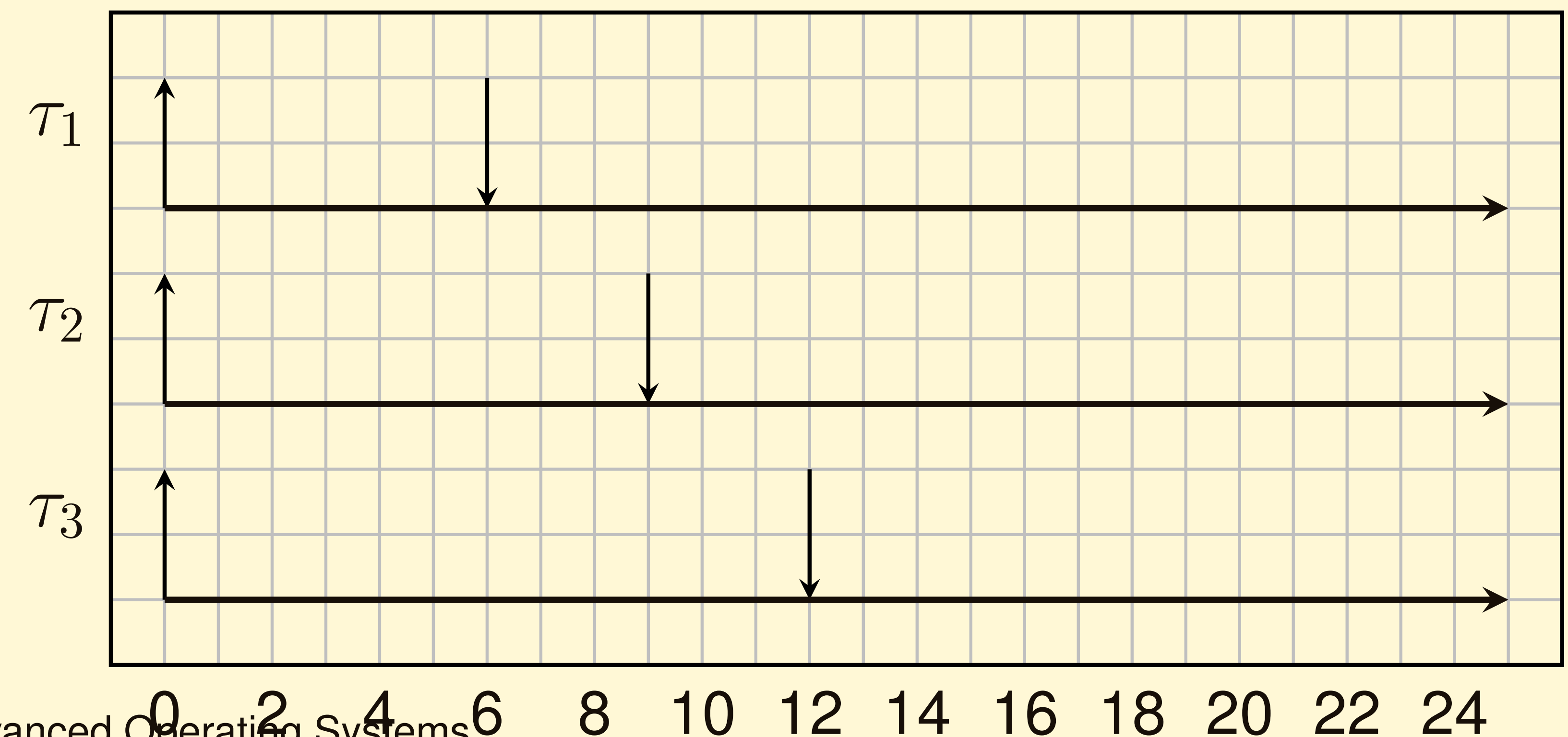


Fixed Priority Scheduling

- 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

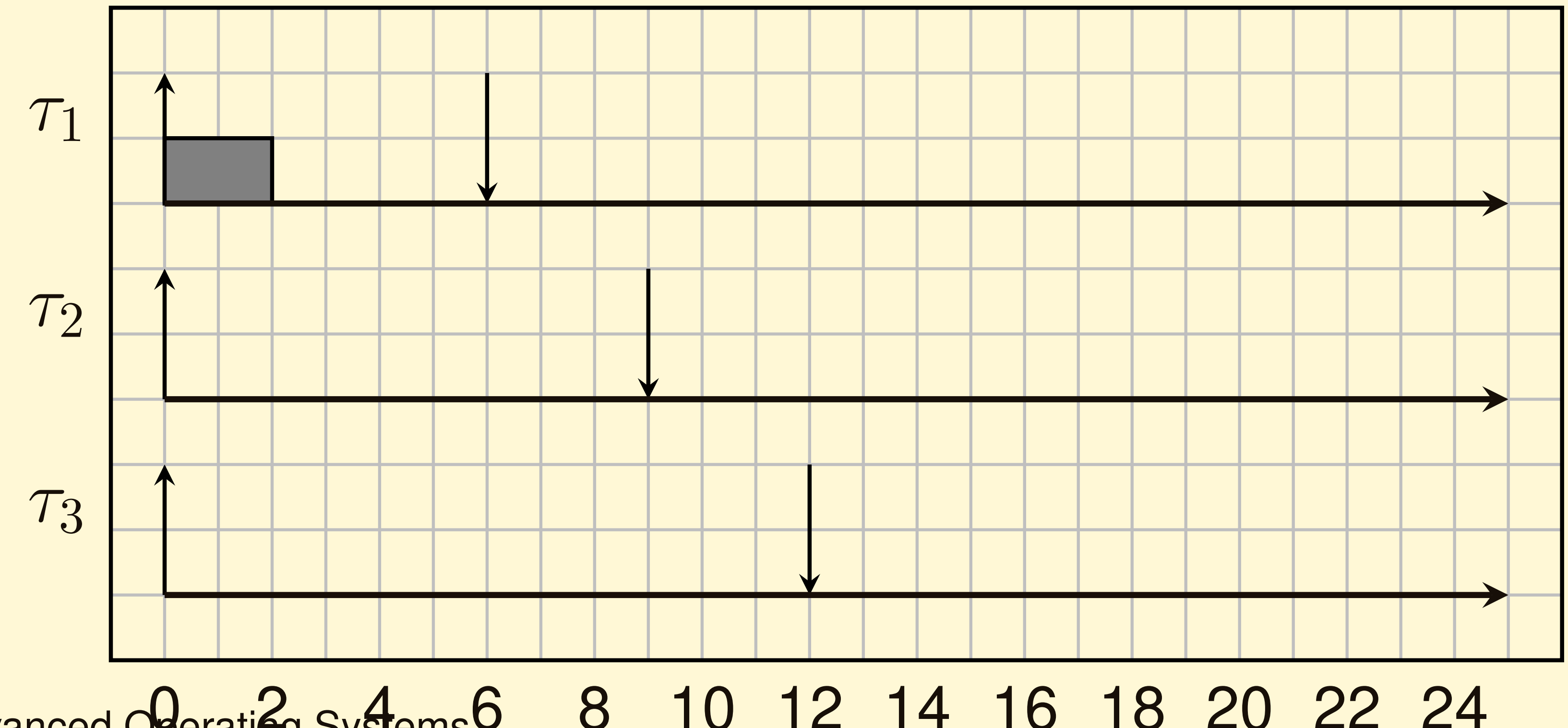
Example of Schedule

- 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)



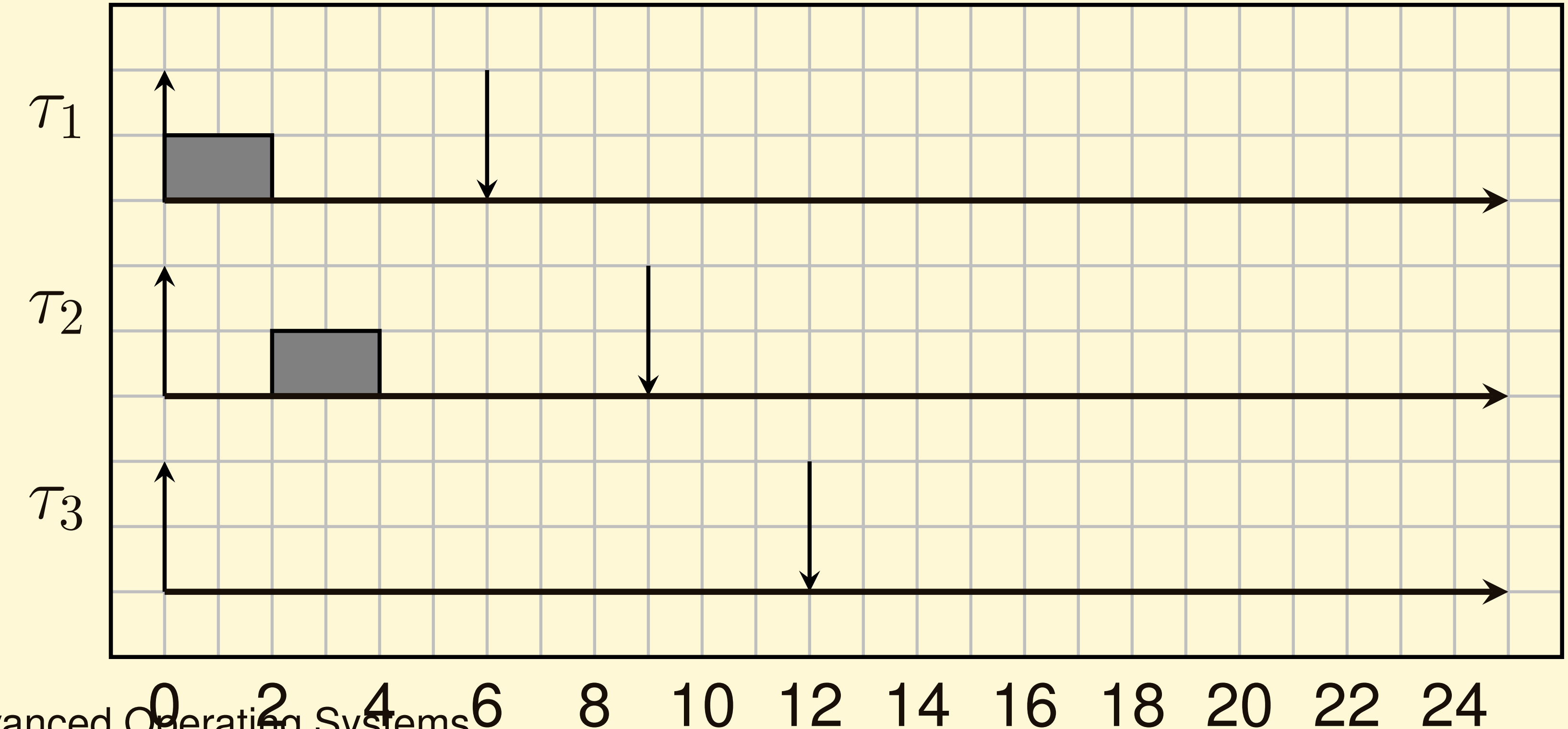
Example of Schedule

- 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)



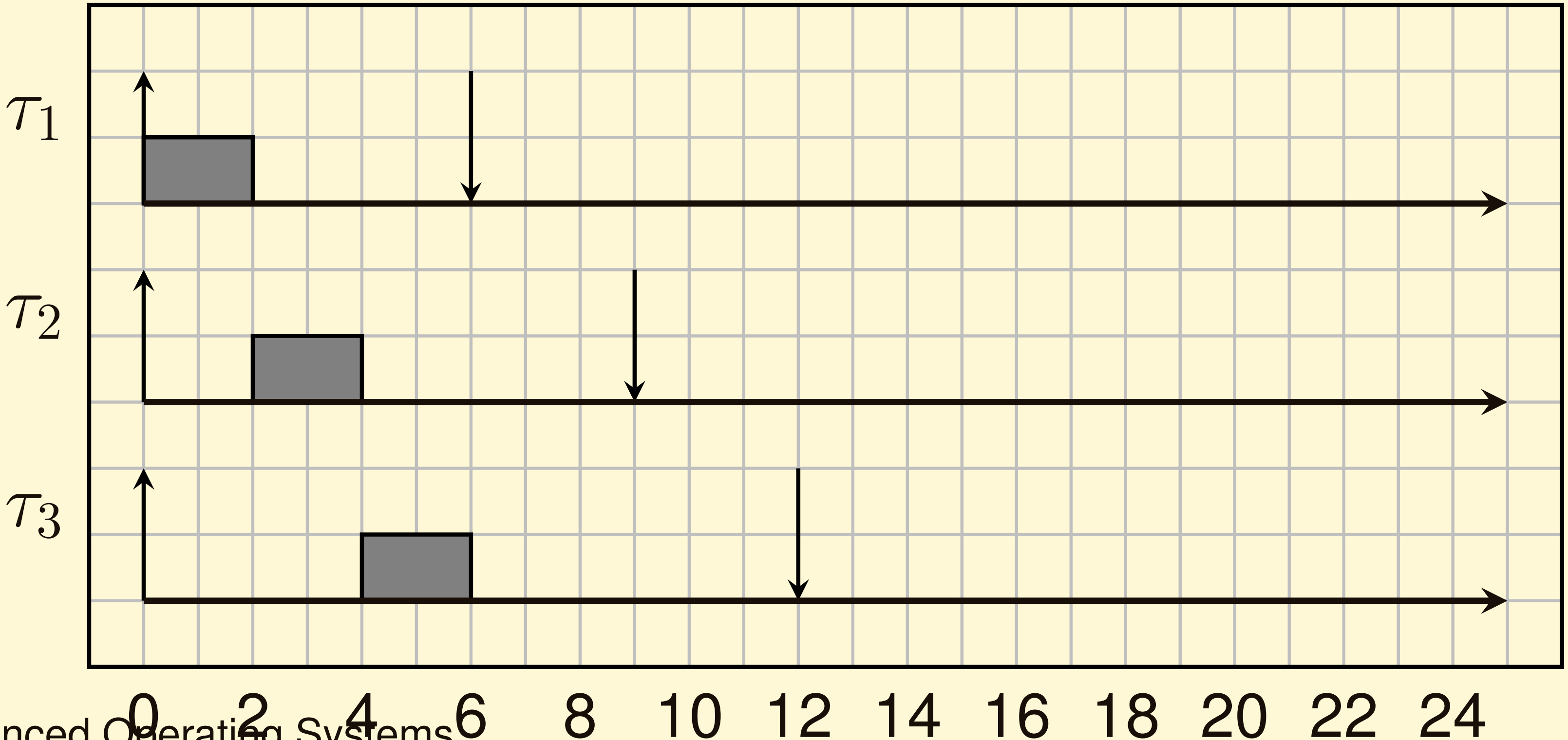
Example of Schedule

- 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)



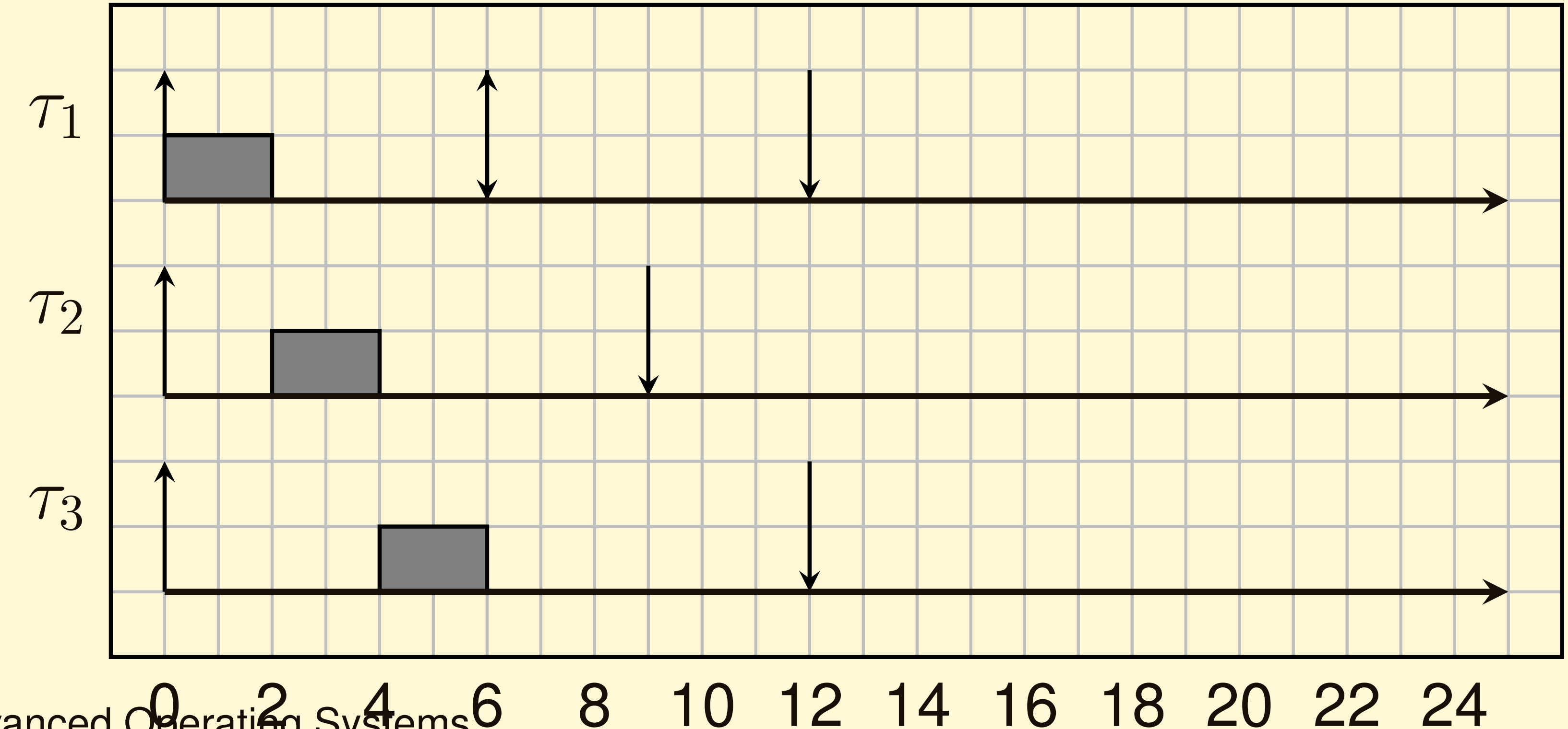
Example of Schedule

- 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)



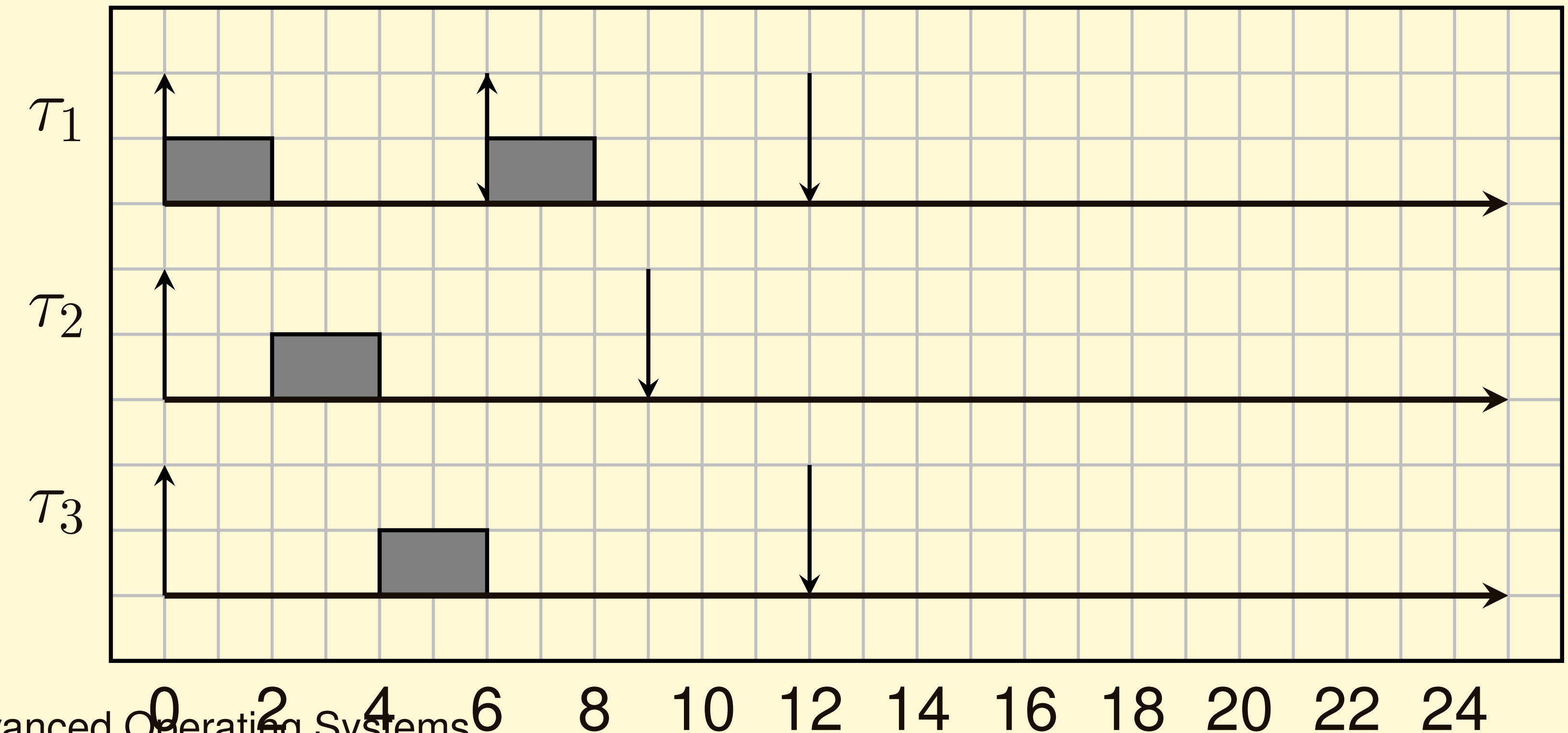
Example of Schedule

- 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)



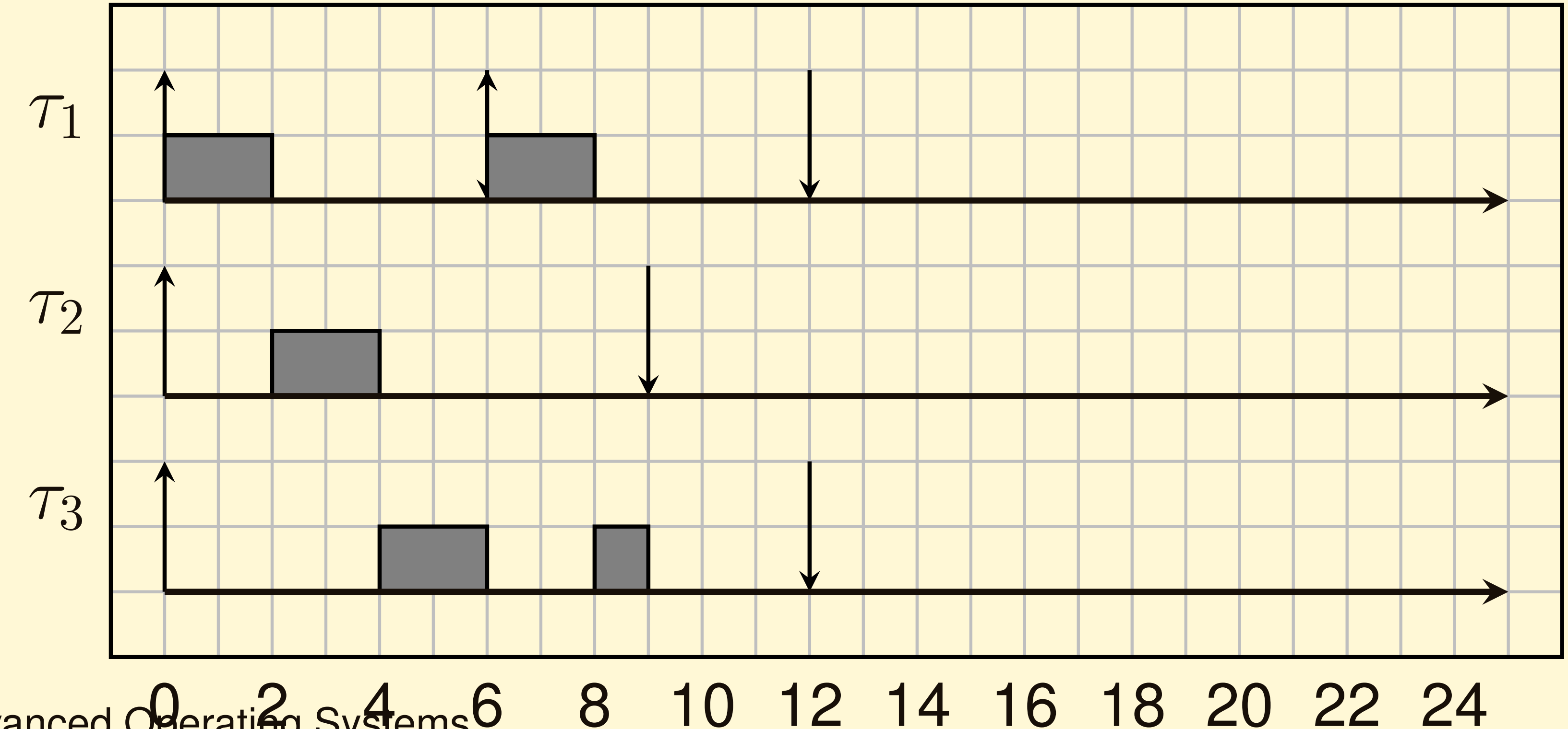
Example of Schedule

- 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)



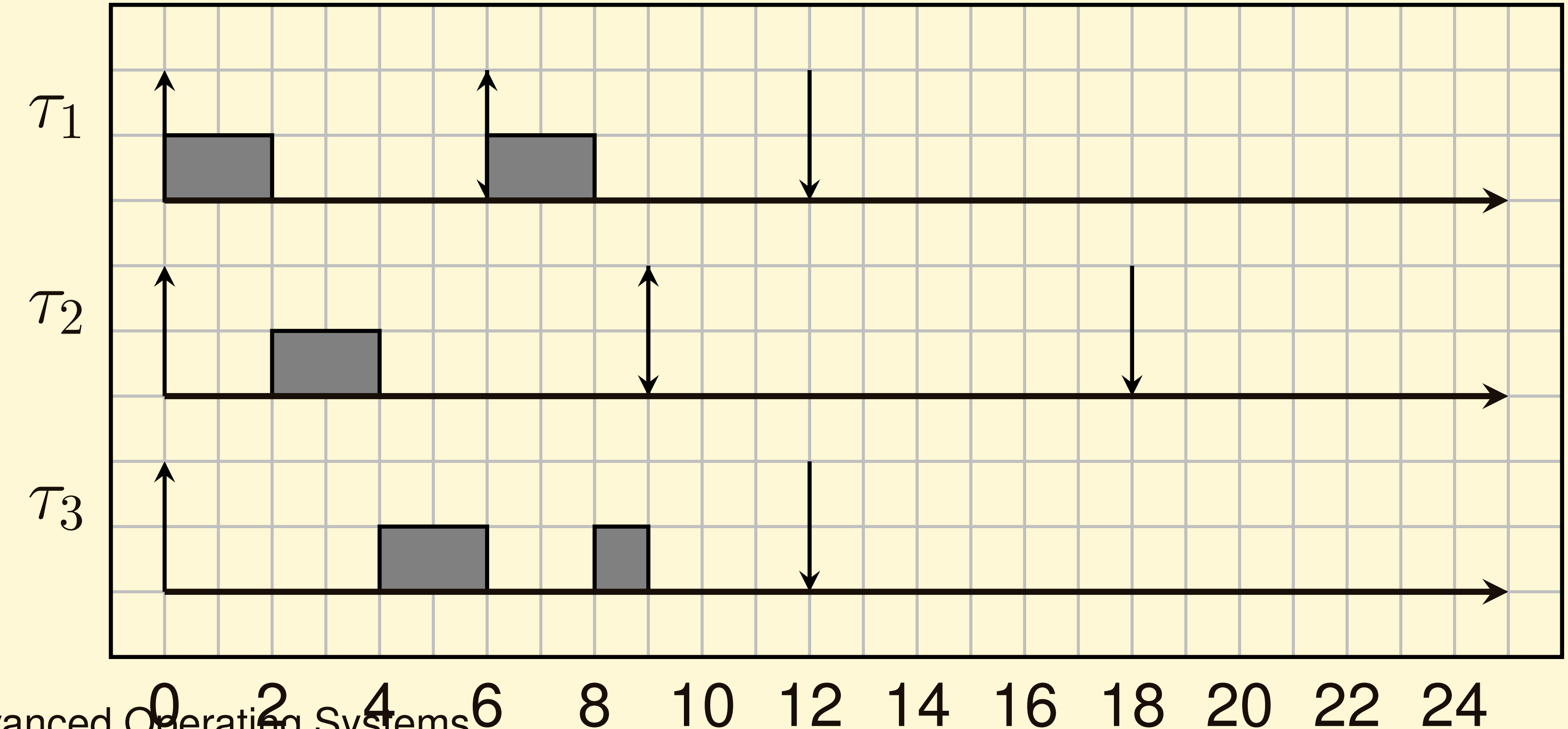
Example of Schedule

- 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)



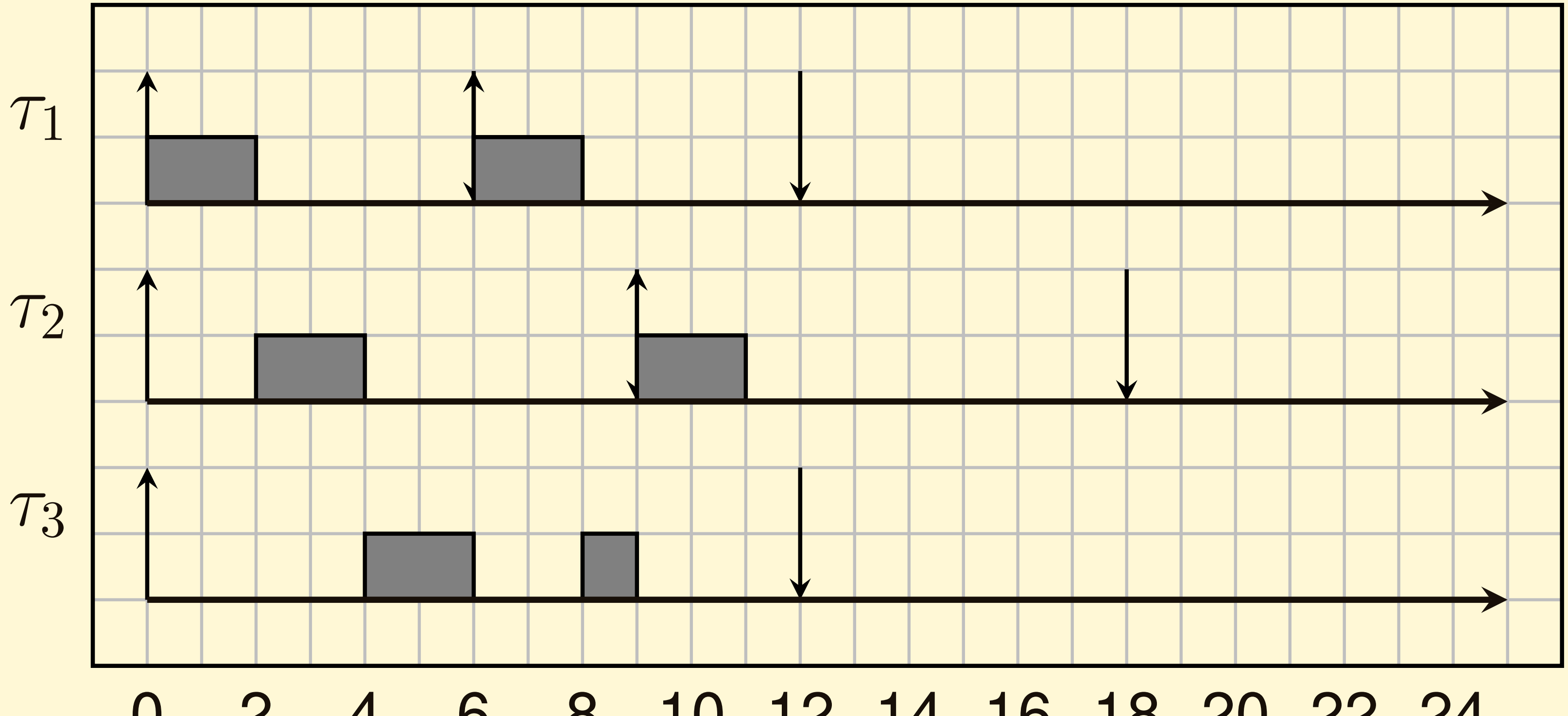
Example of Schedule

- 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)



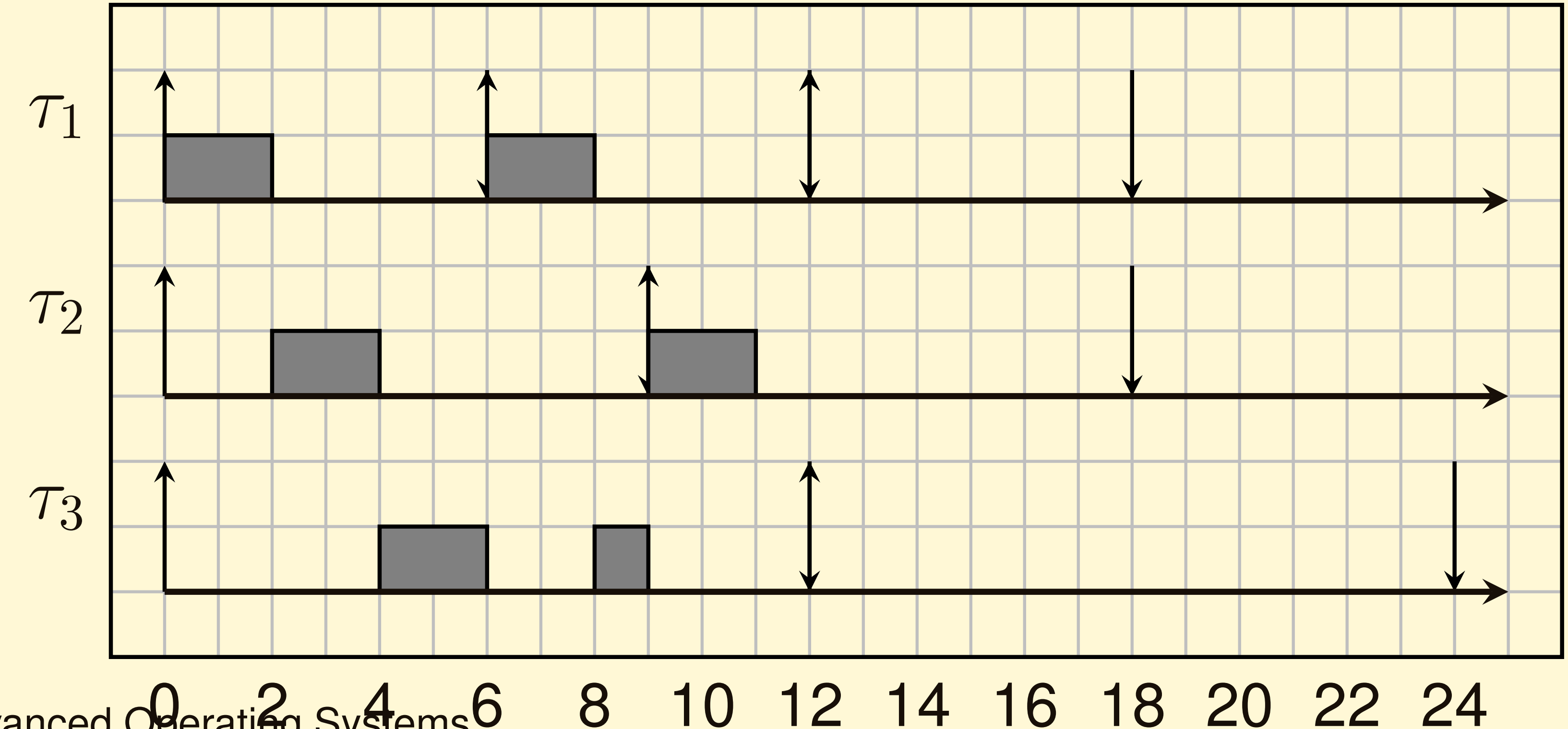
Example of Schedule

- 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)



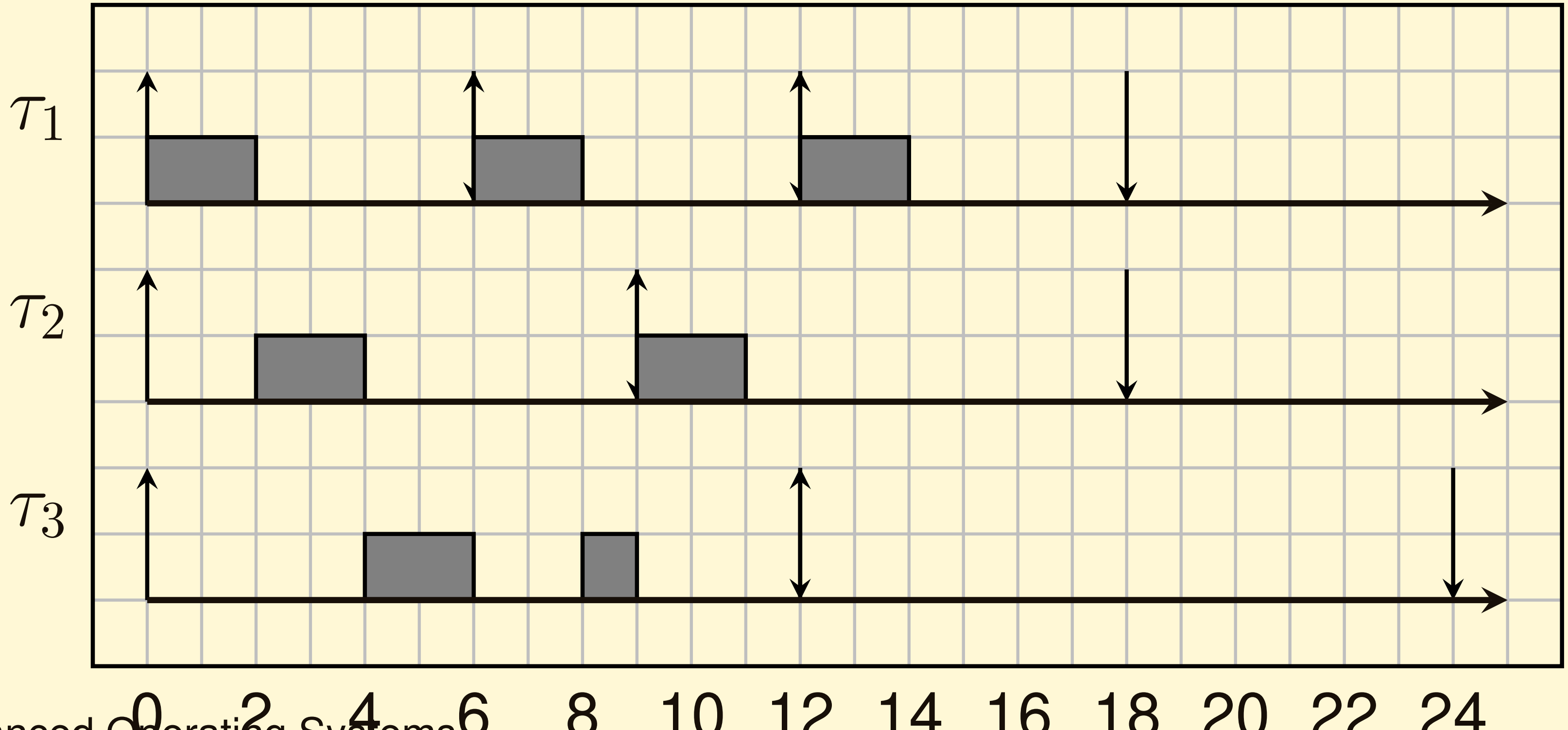
Example of Schedule

- 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)



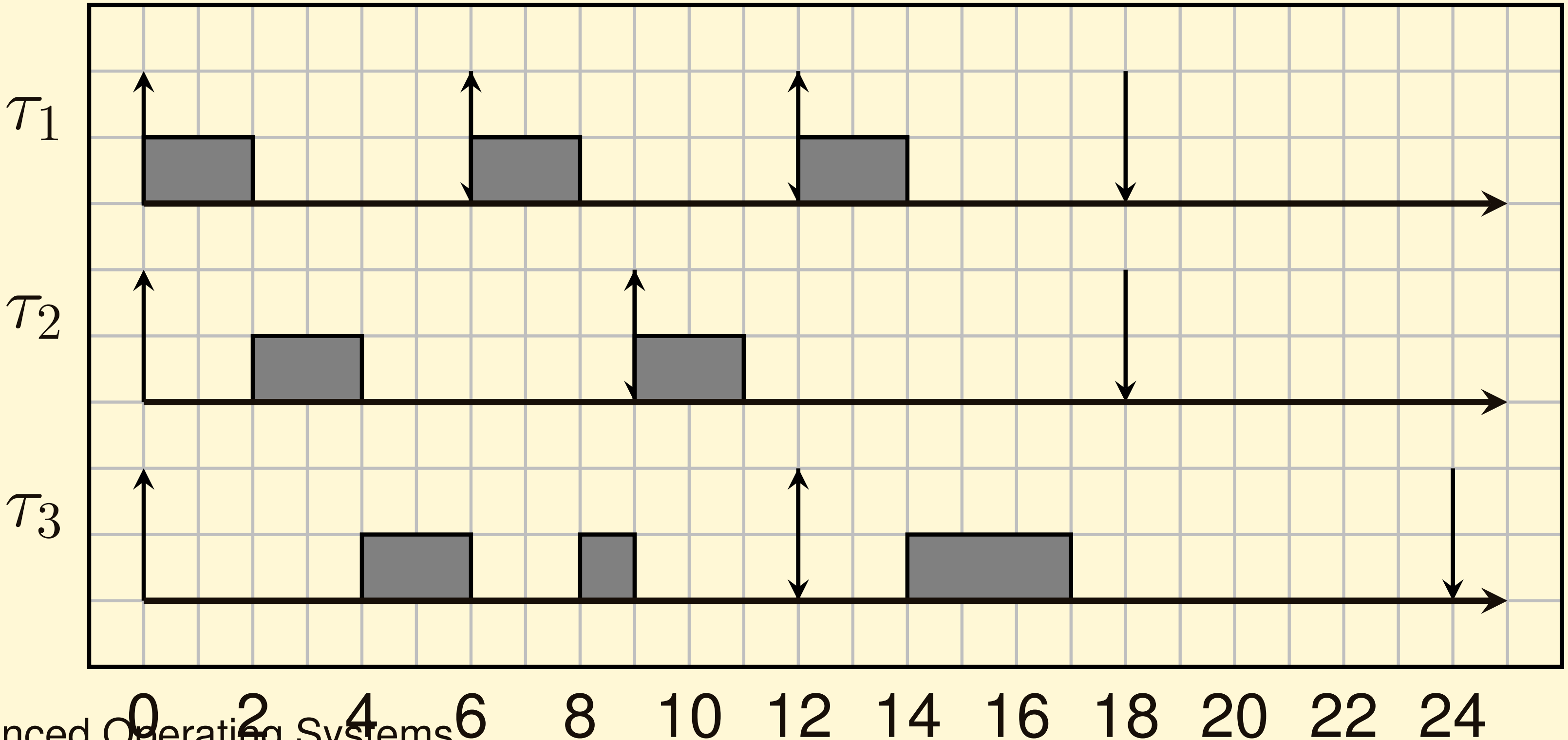
Example of Schedule

- 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)



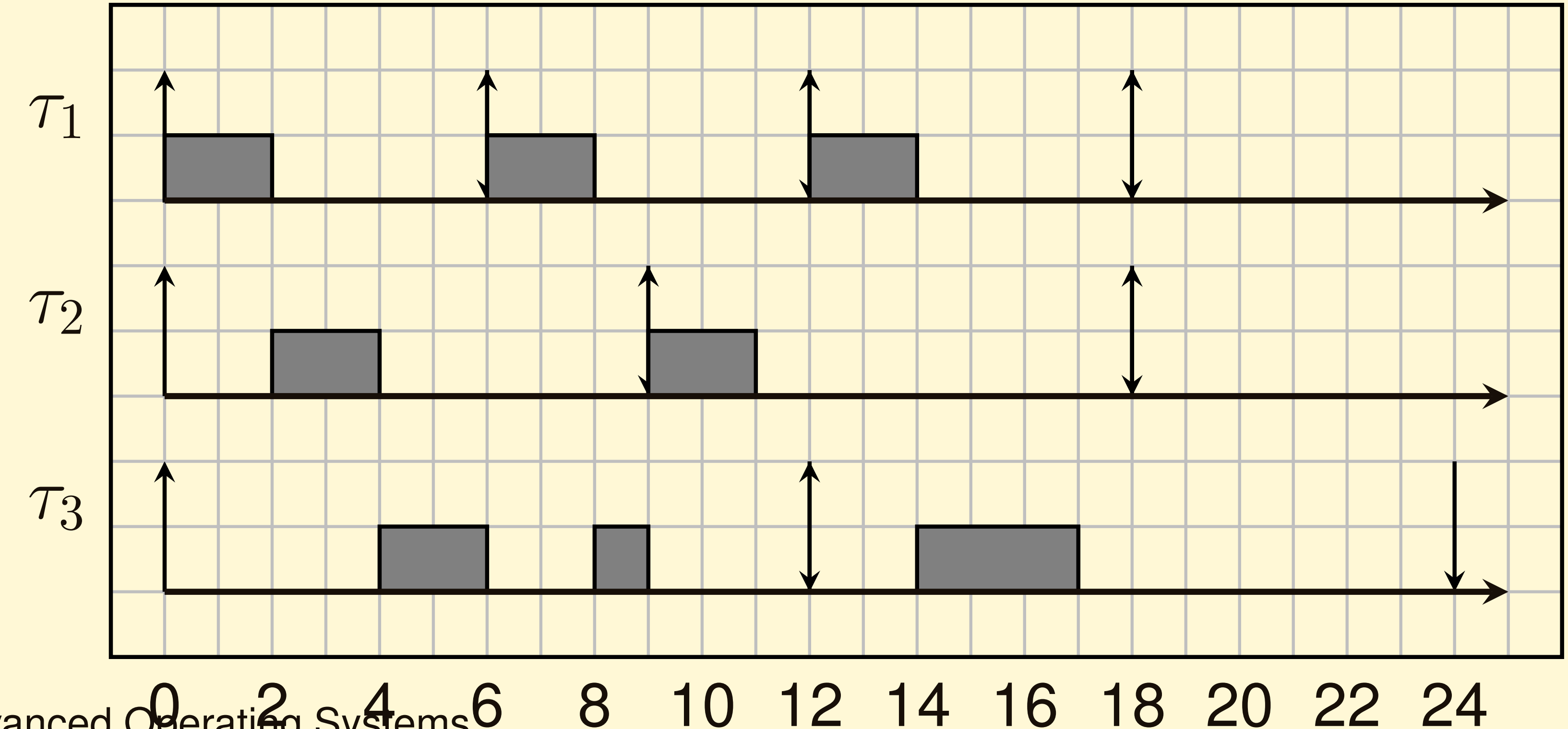
Example of Schedule

- 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)



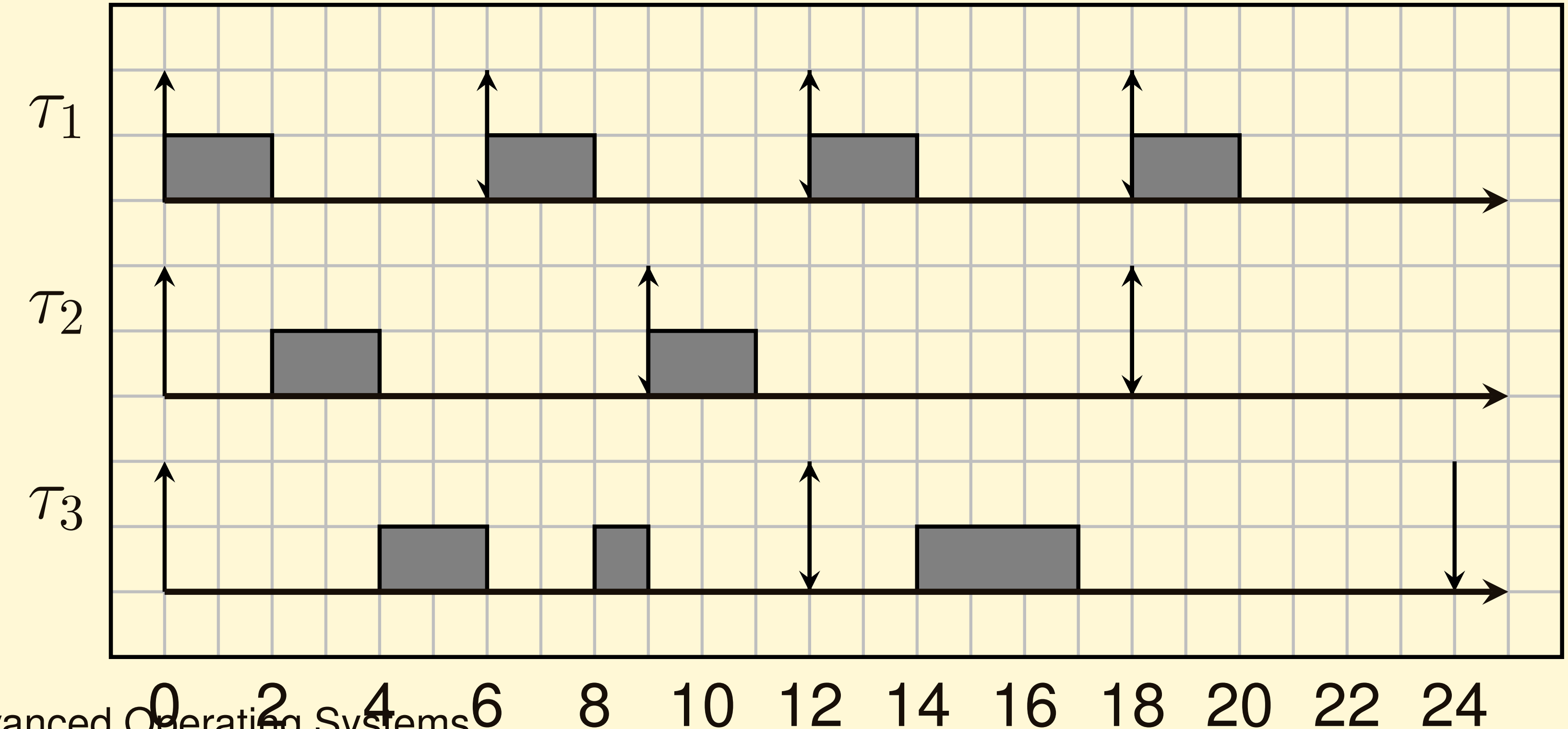
Example of Schedule

- 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)



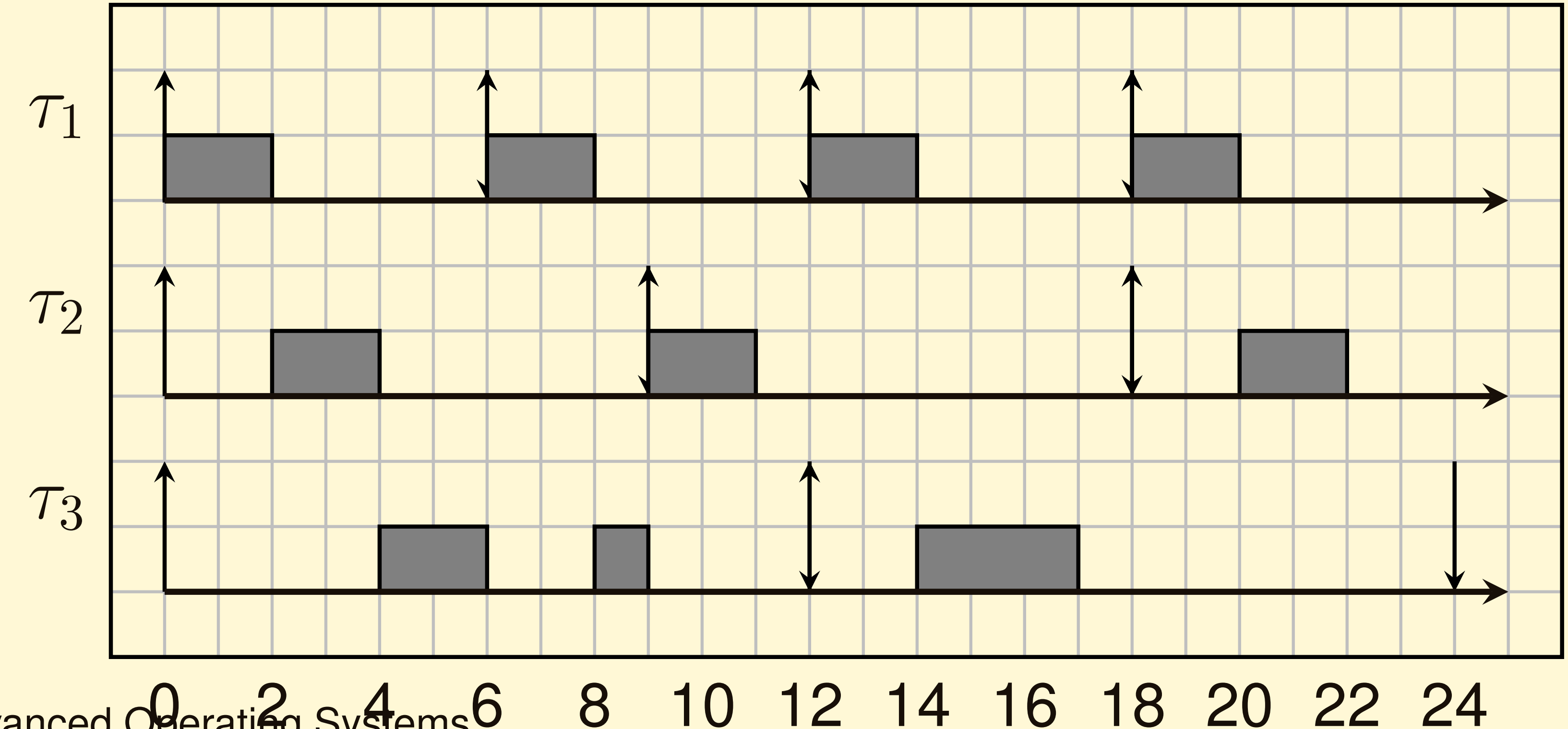
Example of Schedule

- 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)



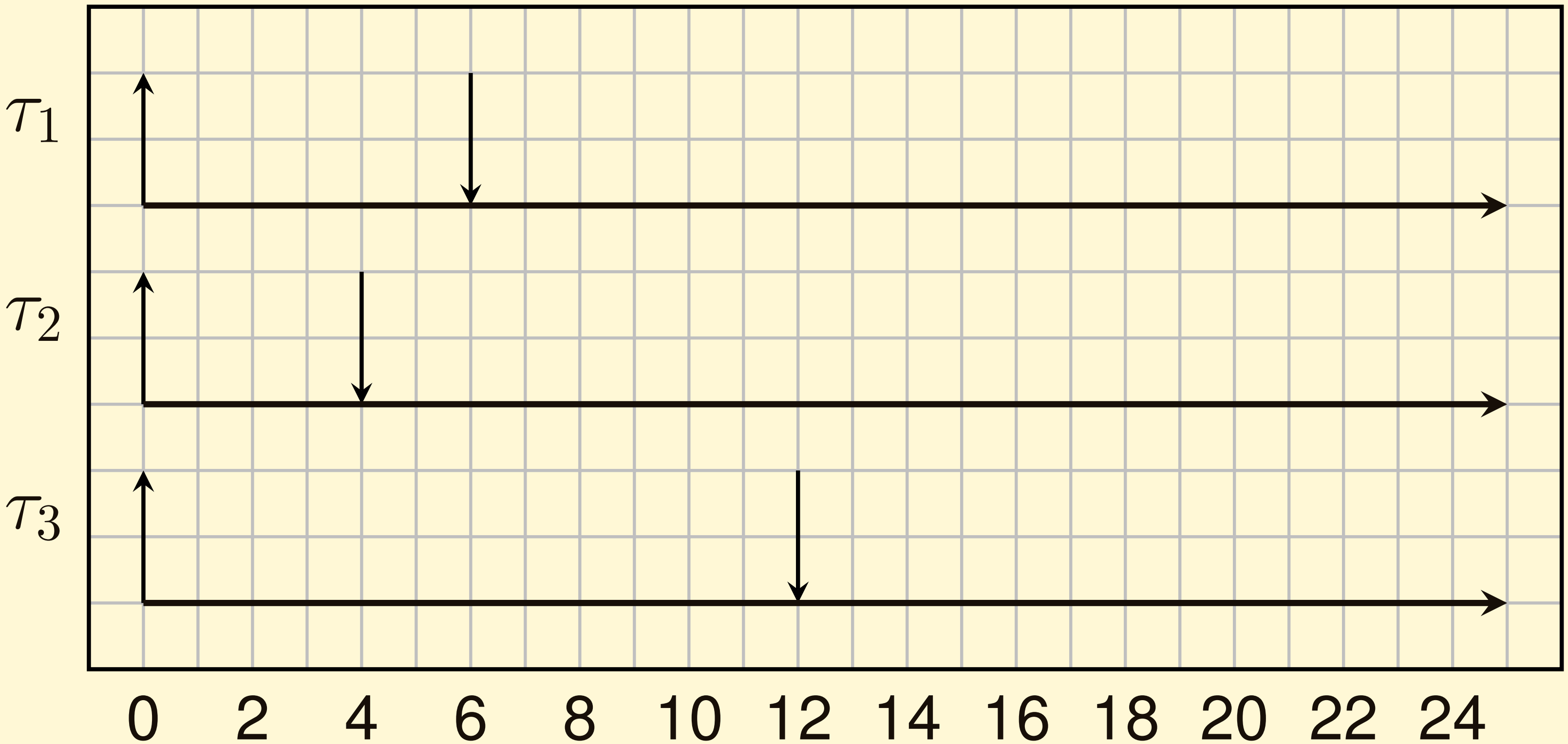
Example of Schedule

- 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)



Another Example (non-schedulable)

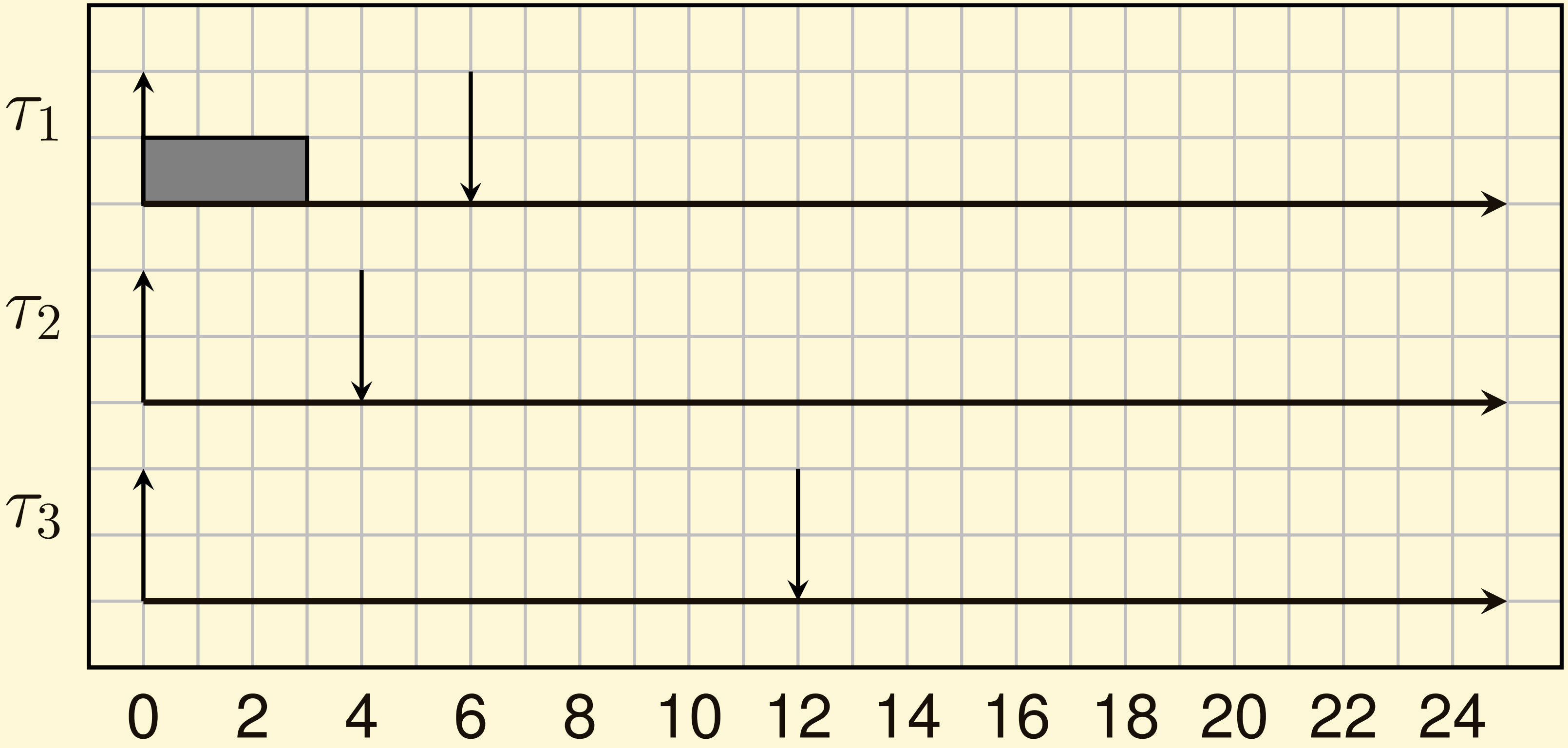
- Consider the following task set: $\tau_1 = (3, 6, 6)$, $p_1 = 3$, $\tau_2 = (2, 4, 8)$, $p_2 = 2$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



In this case, task τ_2 misses its deadline!

Another Example (non-schedulable)

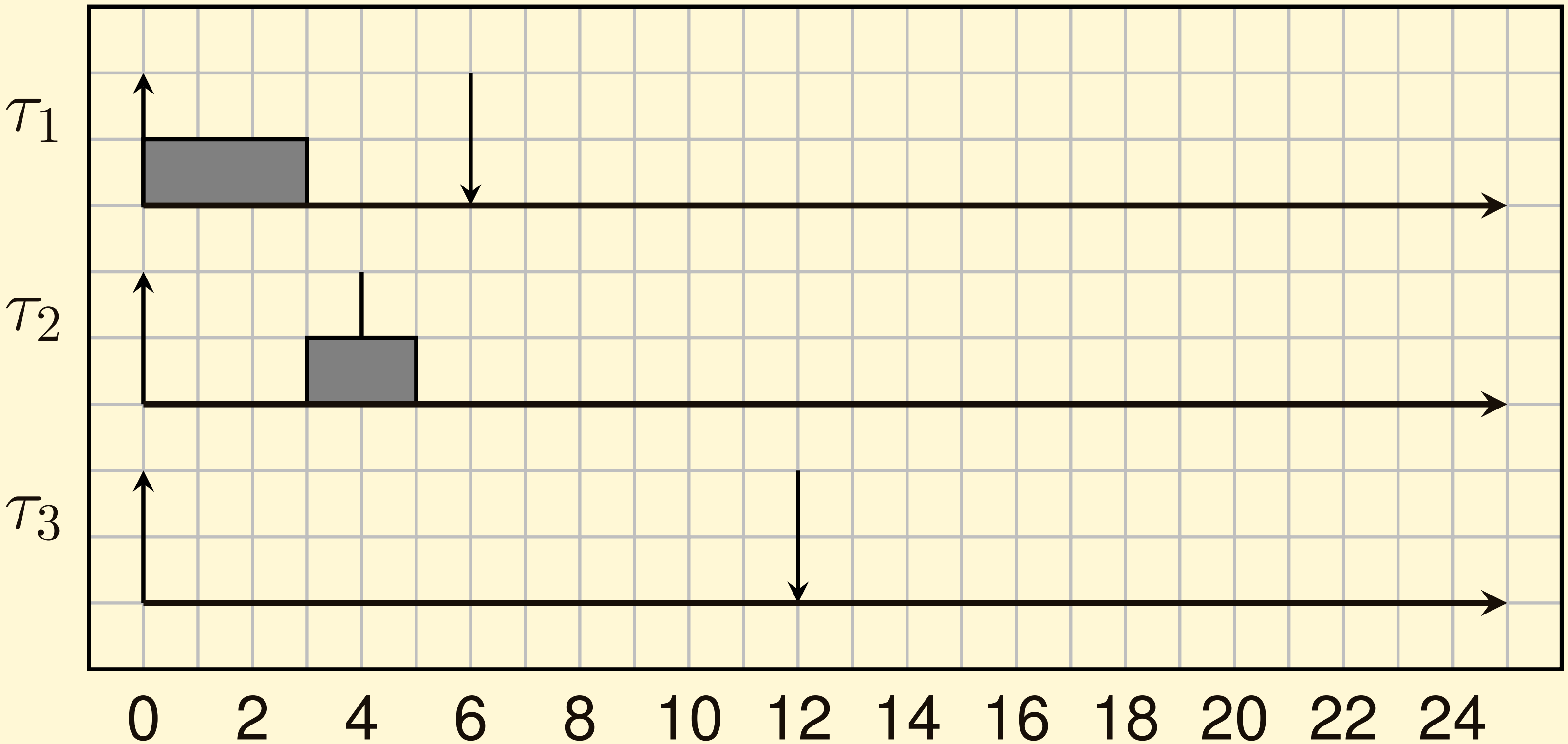
- Consider the following task set: $\tau_1 = (3, 6, 6)$, $p_1 = 3$, $\tau_2 = (2, 4, 8)$, $p_2 = 2$, $\tau_3 = (2, 12, 12)$, $p_3 = 1$



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?

Response Time Analysis

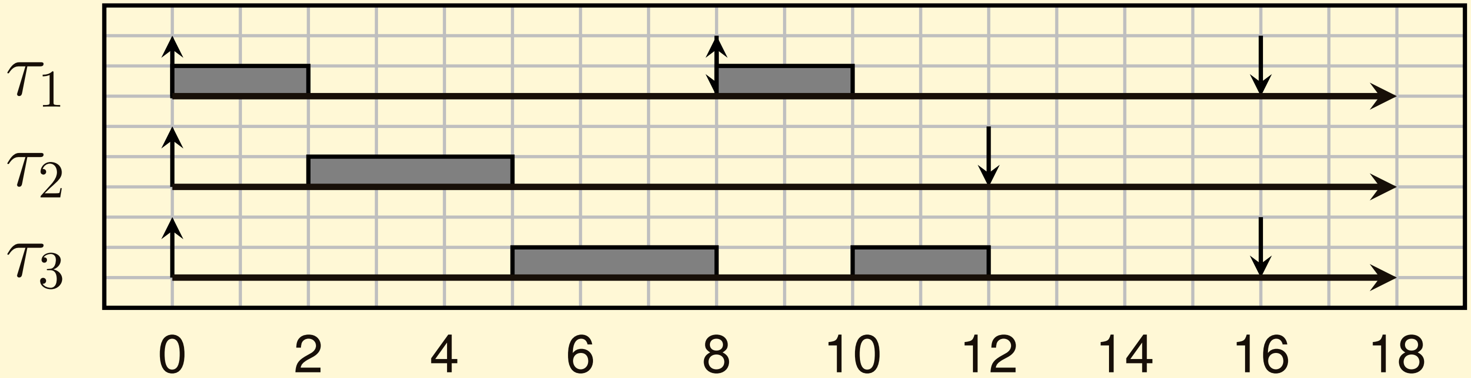
- Necessary and sufficient test: compute the *worst-case response time* for every task
- For every task τ_i:
  - Compute worst case response time R_i for τ_i
    - Remember? $R_i = \max_j\{\rho_{i,j}\}$; $\rho_{i,j} = f_{i,j} - r_{i,j}$
  - If $R_i \leq D_i$, then the task is schedulable
  - otherwise, the task is not schedulable
- No assumption on the priority assignment
  - Algorithm valid for arbitrary priority assignments
  - Not only RM / DM...
- Periodic tasks with no offsets, or sporadic tasks

The Critical Instant

- Tasks ordered by decreasing priority $(i < j \rightarrow p_i > p_j)$
- No assumptions about tasks offsets
  - $\Rightarrow$ Consider the *worst possible offsets combination*
  - A job $J_{i,j}$ released at the *critical instant* experiences the maximum response time for τ_i: $\forall k, \rho_{i,j} \geq \rho_{i,k}$
    - Simplified definition (jobs deadlines should be considered...)
  - **Theorem:** The critical instant for task τ_i occurs when job $J_{i,j}$ is released at the same time with a job in every high priority task
- **If all the offsets are 0, the first job of every task is released at the critical instant!!!**

Worst Case Response Time

- Worst case response time R_i for task τ_i depends on:
  - Its execution time...
  - ...And the execution time of higher priority tasks
    - Higher priority tasks can *preempt* task τ_i, and increase its response time

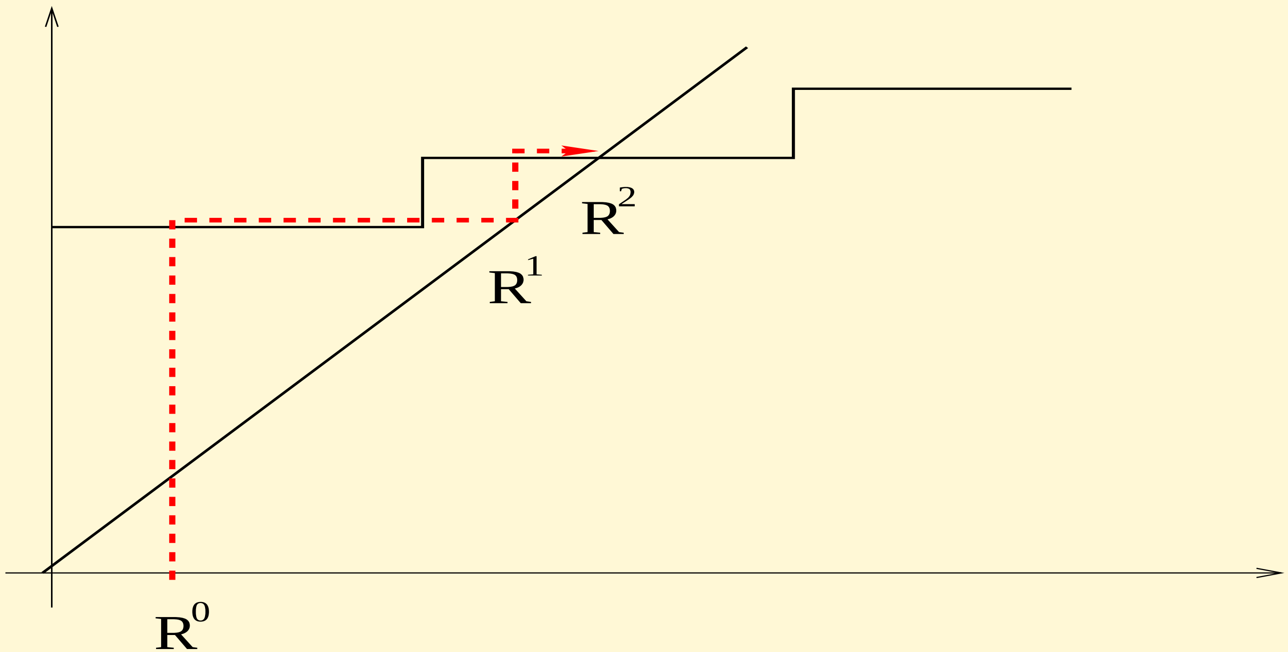


- $R_i = C_i + \sum_{h=1}^{i-1} \left\lceil \frac{R_i}{T_h} \right\rceil C_h$

Computing the Response Time - I

$$R_i = C_i + \sum_{h=1}^{i-1} \left\lceil \frac{R_i}{T_h} \right\rceil C_h$$

- Urk!!! $R_i = f(R_i)$... How can we solve it?
- There is no closed-form expression for computing the worst case response time R_i
- We need an iterative method to solve the equation



Computing the Response Time - II

- Iterative solution
  - $R_i = \lim_{k\to\infty} R_i^{(k)}$
  - $R_i^{(k)}$: worst case response time for τ_i, at step k
- $R_i^{(0)}$: first estimation of the response time
  - We can start with $R_i^{(0)} = C_i$
  - $R_i^{(0)} = C_i + \sum_{h=1}^{i-1} C_h$ saves 1 step

$$
\begin{aligned}
R_i^{(0)} &= C_i(+\sum_{h=1}^{i-1} C_h) \\
R_i^{(k)} &= C_i + \sum_{h=1}^{i-1} \left\lceil \frac{R_i^{(k-1)}}{T_h} \right\rceil C_h
\end{aligned}
$$

Computing the Response Time - III

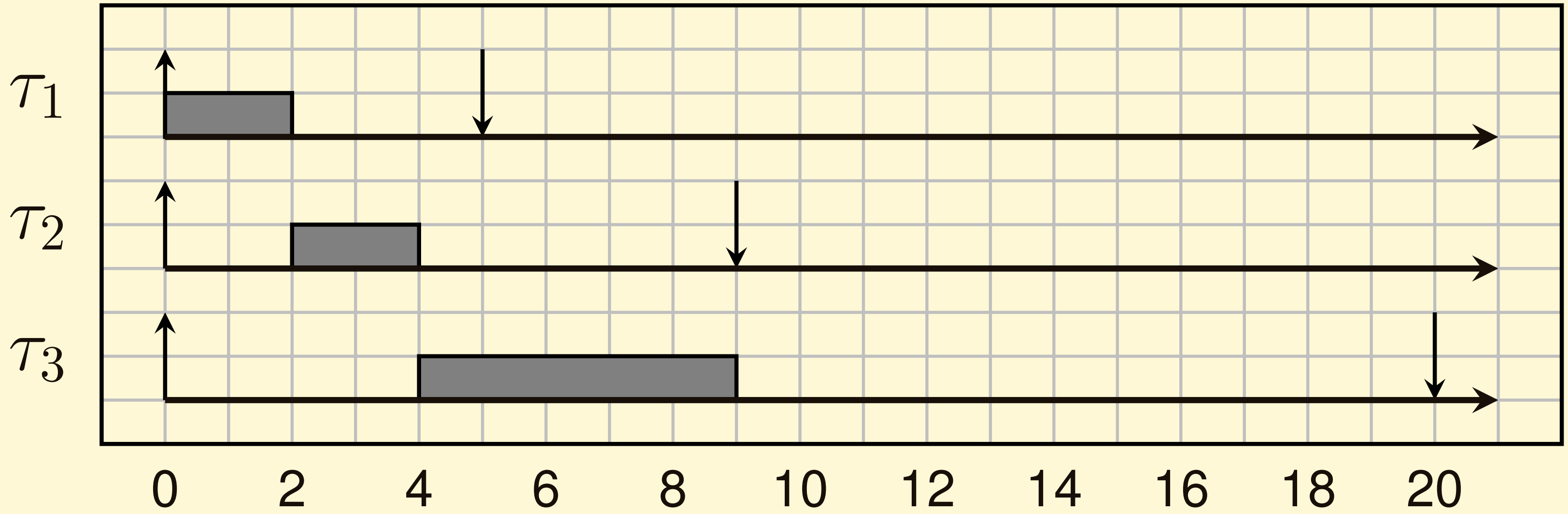
- Problem: are we sure that we find a valid solution?
- The iteration stops when:
  - $R_i^{(k+1)} = R_i^{(k)}$ *or*
  - $R_i^{(k)} > D_i$ (non schedulable);
- This is a standard method to solve non-linear equations in an iterative way
- If a solution exists (the system is not overloaded), $R_i^{(k)}$ converges to it
- Otherwise, the "$R_i^{(k)} > D_i$" condition avoids infinite iterations

Example

Task set: $\tau_1 = (2,5)$, $\tau_2 = (2,9)$, $\tau_3 = (5,20)$; $U = 0.872$

$$R_i^{(k)} = C_i + \sum_{h=1}^{i-1} \left\lceil \frac{R_i^{(k-1)}}{T_h} \right\rceil C_h$$

$$R_3^{(0)} = C_3 + 1 \cdot C_1 + 1 \cdot C_2 = 9$$

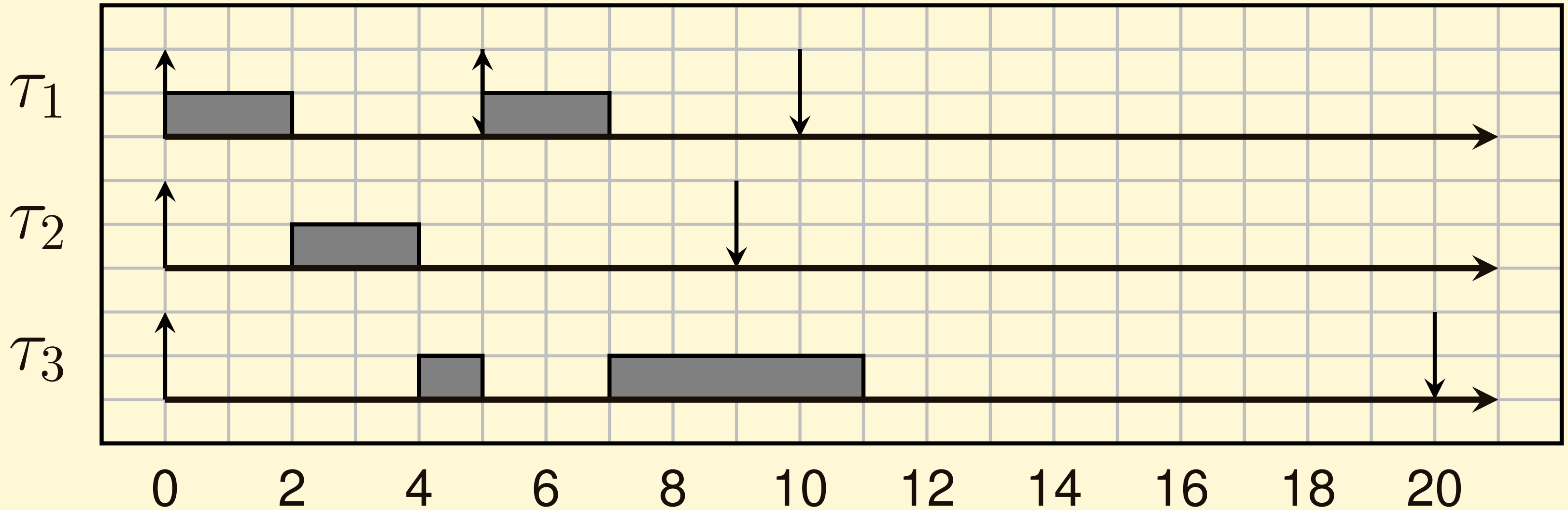


Example

Task set: $\tau_1 = (2,5)$, $\tau_2 = (2,9)$, $\tau_3 = (5,20)$; $U = 0.872$

$$R_i^{(k)} = C_i + \sum_{h=1}^{i-1} \left\lceil \frac{R_i^{(k-1)}}{T_h} \right\rceil C_h$$

$$R_3^{(1)} = C_3 + 2 \cdot C_1 + 1 \cdot C_2 = 11$$

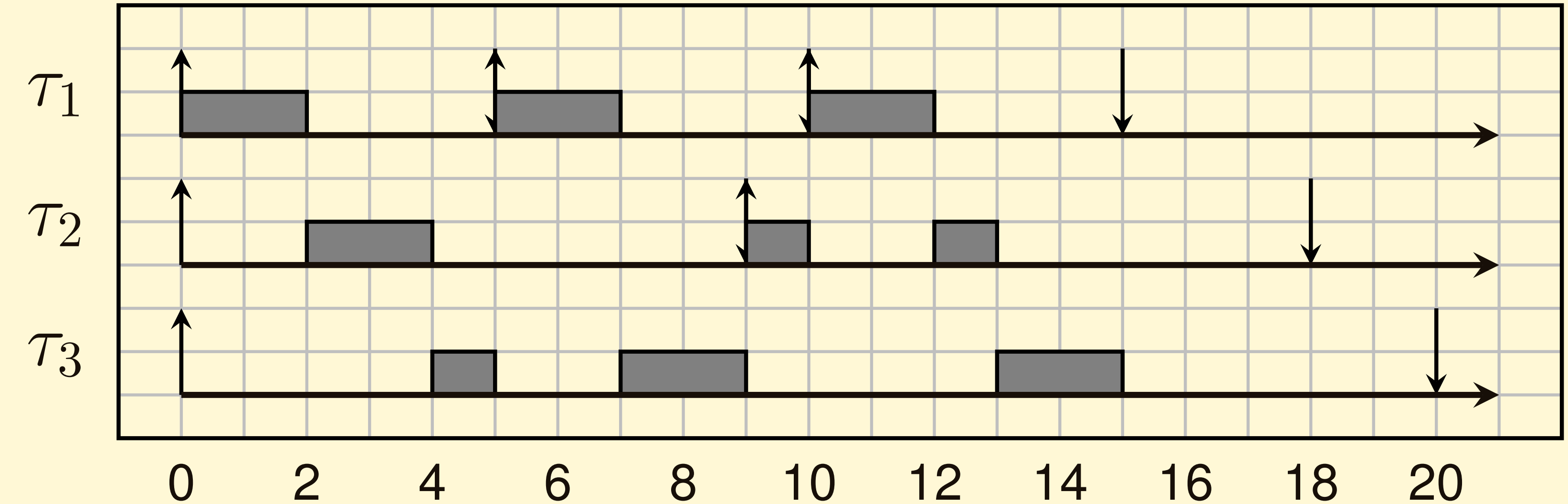


Example

Task set: $\tau_1 = (2,5)$, $\tau_2 = (2,9)$, $\tau_3 = (5,20)$; $U = 0.872$

$$R_i^{(k)} = C_i + \sum_{h=1}^{i-1} \left\lceil \frac{R_i^{(k-1)}}{T_h} \right\rceil C_h$$

$$R_3^{(2)} = C_3 + 3 \cdot C_1 + 2 \cdot C_2 = 15$$

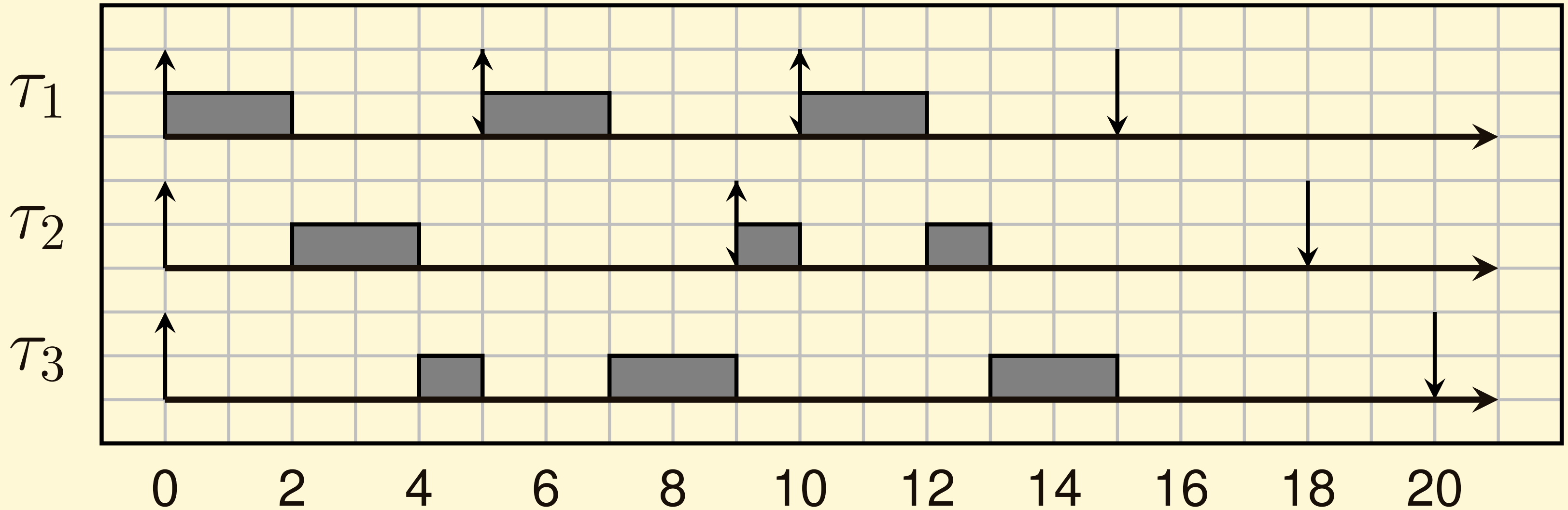


Example

Task set: $\tau_1 = (2,5)$, $\tau_2 = (2,9)$, $\tau_3 = (5,20)$; $U = 0.872$

$$R_i^{(k)} = C_i + \sum_{h=1}^{i-1} \left\lceil \frac{R_i^{(k-1)}}{T_h} \right\rceil C_h$$

$$R_3^{(3)} = C_3 + 3 \cdot C_1 + 2 \cdot C_2 = 15 = R_3^{(2)}$$



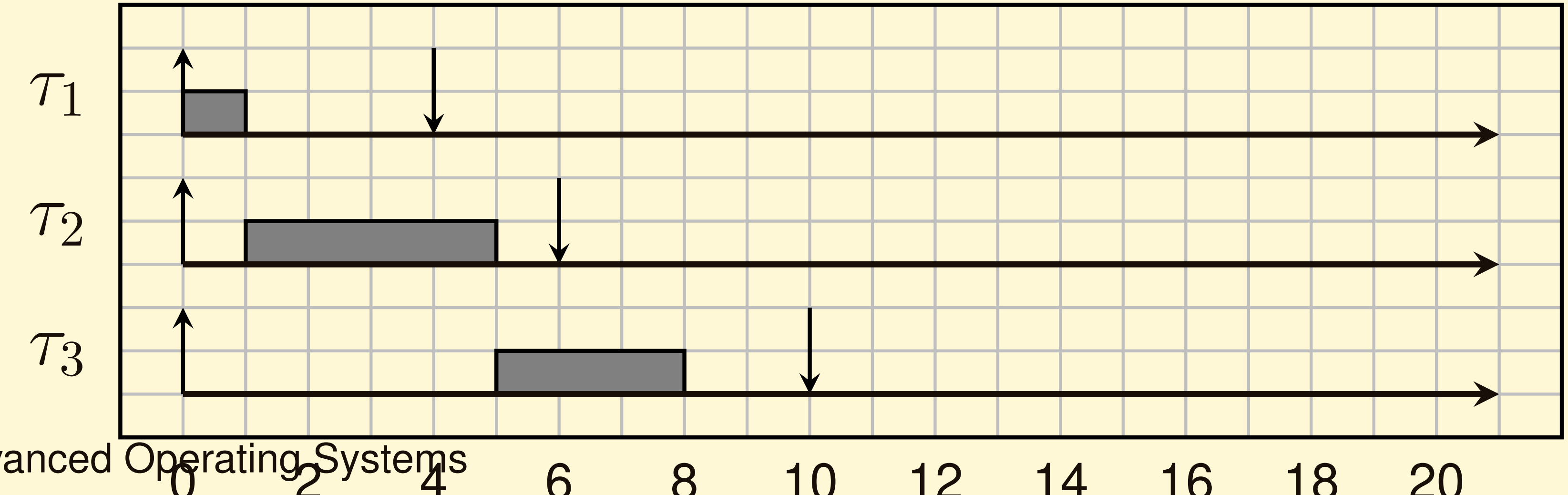
Another Example with DM

What about different priority assignments and deadlines different from periods?

$\tau_1 = (1, 4, 4), p_1 = 3$, $\tau_2 = (4, 6, 15), p_2 = 2$,
$\tau_3 = (3, 10, 10), p_3 = 1; U = 0.72$

$$R_i^{(k)} = C_i + \sum_{h=1}^{i-1} \left\lceil \frac{R_i^{(k-1)}}{T_h} \right\rceil C_h$$

$R_3^{(0)} = C_3 + 1 \cdot C_1 + 1 \cdot C_2 = 8$



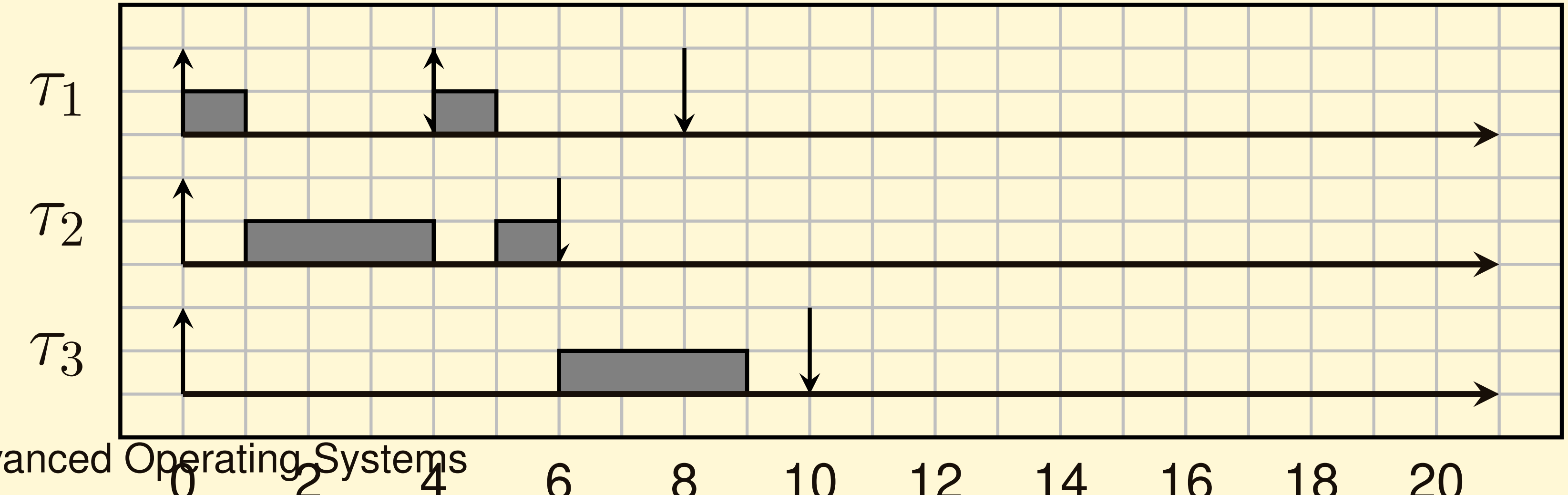
Another Example with DM

What about different priority assignments and deadlines different from periods?

$\tau_1 = (1, 4, 4), p_1 = 3$, $\tau_2 = (4, 6, 15), p_2 = 2$,
$\tau_3 = (3, 10, 10), p_3 = 1; U = 0.72$

$$R_i^{(k)} = C_i + \sum_{h=1}^{i-1} \left\lceil \frac{R_i^{(k-1)}}{T_h} \right\rceil C_h$$

$R_3^{(1)} = C_3 + 2 \cdot C_1 + 1 \cdot C_2 = 9$



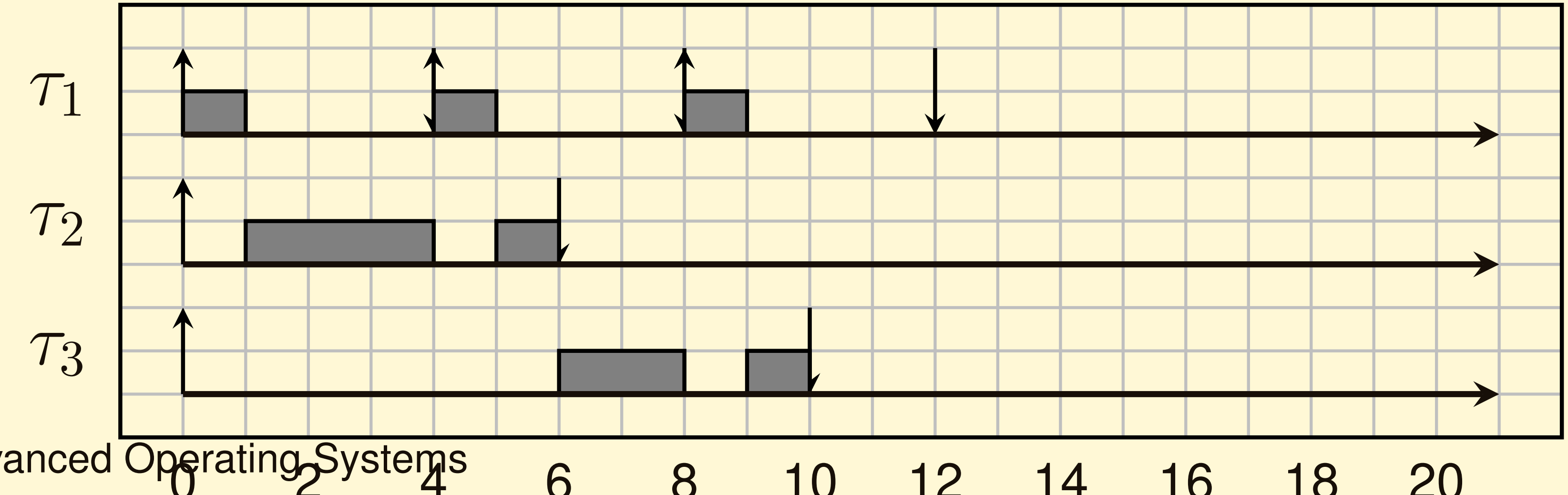
Another Example with DM

What about different priority assignments and deadlines different from periods?

$\tau_1 = (1, 4, 4), p_1 = 3$, $\tau_2 = (4, 6, 15), p_2 = 2$,
$\tau_3 = (3, 10, 10), p_3 = 1; U = 0.72$

$$R_i^{(k)} = C_i + \sum_{h=1}^{i-1} \left\lceil \frac{R_i^{(k-1)}}{T_h} \right\rceil C_h$$

$R_3^{(2)} = C_3 + 3 \cdot C_1 + 1 \cdot C_2 = 10$



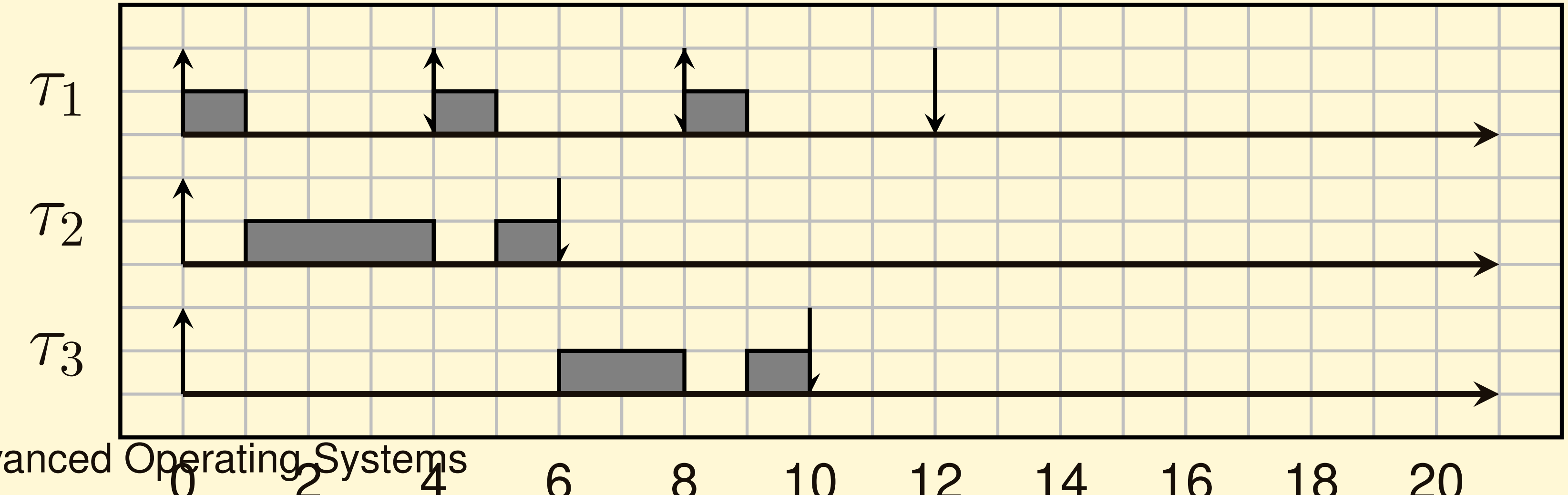
Another Example with DM

What about different priority assignments and deadlines different from periods?

$\tau_1 = (1, 4, 4), p_1 = 3$, $\tau_2 = (4, 6, 15), p_2 = 2$,
$\tau_3 = (3, 10, 10), p_3 = 1; U = 0.72$

$$R_i^{(k)} = C_i + \sum_{h=1}^{i-1} \left\lceil \frac{R_i^{(k-1)}}{T_h} \right\rceil C_h$$

$R_3^{(3)} = C_3 + 3 \cdot C_1 + 1 \cdot C_2 = 10 = R_3^{(2)}$



Considerations

- The response time analysis is an efficient algorithm
  - In the worst case, the number of steps N for the algorithm to converge is exponential
    - Depends on the total number of jobs of higher priority tasks in the interval $[0, D_i]$:

$$N \propto \sum_{h=1}^{i-1} \left\lceil \frac{D_h}{T_h} \right\rceil$$

    - If s is the minimum granularity of the time, then in the worst case $N = \frac{D_i}{s}$;
  - However, such worst case is very rare: usually, the number of steps is low.

Real-Time Operating Systems

- Real-Time operating system (RTOS): OS providing support to Real-Time applications
- Real-Time application: the correctness depends not only on the output values, but also on the time when such values are produced
- Operating System:
  - Set of computer programs
  - Interface between applications and hardware
  - Control the execution of application programs
  - Manage the hardware and software resources

Different Visions of an OS

- An OS manages resources to provide services...
- ...hence, it can be seen as:
  - A Service Provider for user programs
    - Exports a programming interface...
  - A Resource Manager
    - Implements schedulers...

Operating System Services

- Services (Kernel Space):
  - Process Synchronisation, Inter-Process Communication (IPC)
  - Process / Thread Scheduling
  - I / O
  - Virtual Memory

RT-POSIX API?

Task Scheduling

- *Kernel*: core part of the OS, allowing multiple tasks to run on the same CPU
  - Task set $\mathcal{T}$ composed by N tasks running on M CPUs ($M < N$)
  - All tasks τ_i have the illusion to run in parallel
  - Temporal multiplexing between tasks
- Two core components:
  - *Scheduler*: decides which task to execute
  - *Dispatcher*: actually switches the CPU context (context switch)

Synchronization and IPC

- The kernel must also provide a mechanism for allowing tasks to communicate and synchronize
- Two possible programming paradigms:
  - Shared memory (threads)
  - Message passing (processes)

Programming Paradigms

- Shared memory (threads)
  - The kernel must provide mutexes + condition variables
  - Real-time resource sharing protocols (PI, HLP, NPP, ...) must be implemented
- Message passing (processes)
  - Interaction models: pipeline, client / server, ...
  - The kernel must provide some IPC mechanism: pipes, message queues, mailboxes, RPC, ...
  - Some real-time protocols can still be used

Real-Time Scheduling in Practice

- An adequate scheduling of system resources removes the need for over-engineering the system, and is necessary for providing a predictable QoS
- Algorithm + Implementation = Scheduling
- RT theory provides us with good algorithms...
- ...But which are the prerequisites for correctly implementing them?

Theoretical and Actual Scheduling

- Scheduler, IPC subsystem, ... $\rightarrow$ must respect the theoretical model
  - Scheduling is simple: fixed priorities
  - IPC, HLP, or NPP are simple too...
  - But what about (for example) timers?
- Problem:
  - Is the scheduler able to select a high-priority task as soon as it is ready?
  - And the dispatcher?

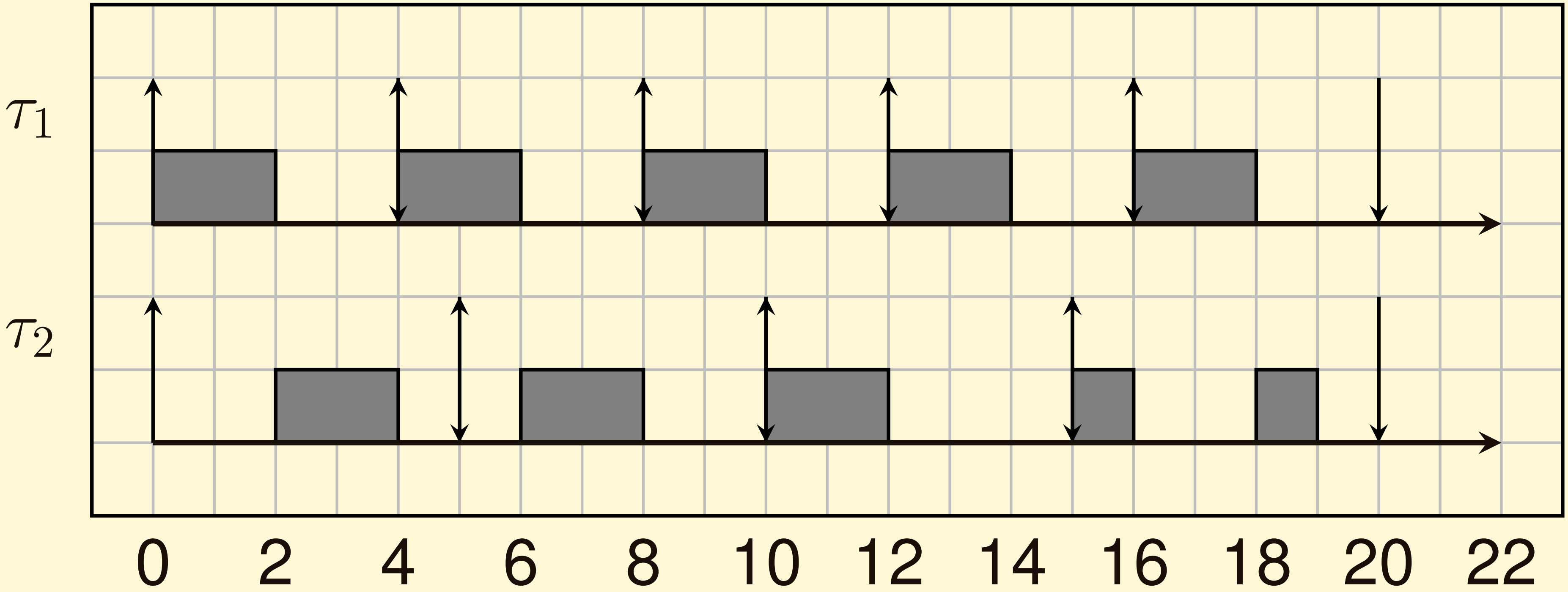
Periodic Task Example

- Consider a periodic task

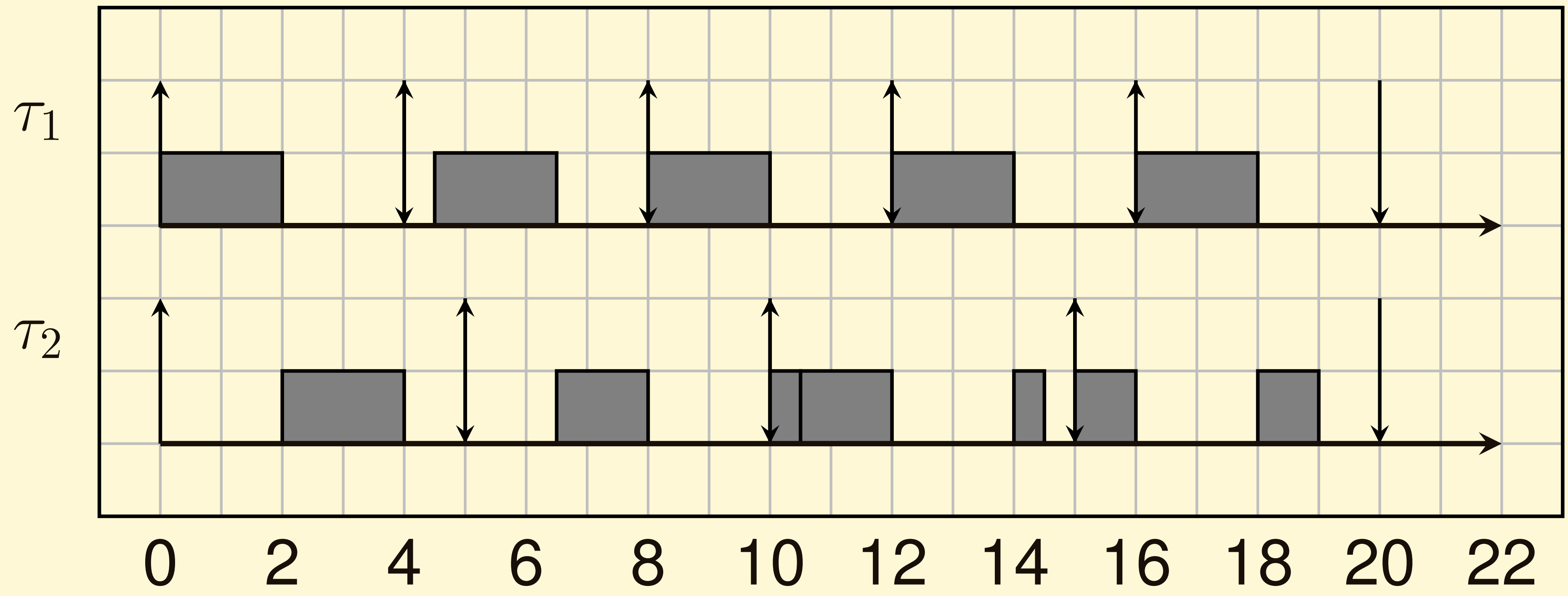
```
/* ... */
while(1) {
  /* Job body */
  clock_nanosleep(CLOCK_REALTIME,
                  TIMER_ABSTIME, &r, NULL);
  timespec_add_us(&r, period);
}
```

- The task expects to be executed at time `r` $(= r_0 + jT)$...
- ...But is sometimes delayed to $r_0 + jT + \delta$

Example - Theoretical Schedule



Example - Actual Schedule



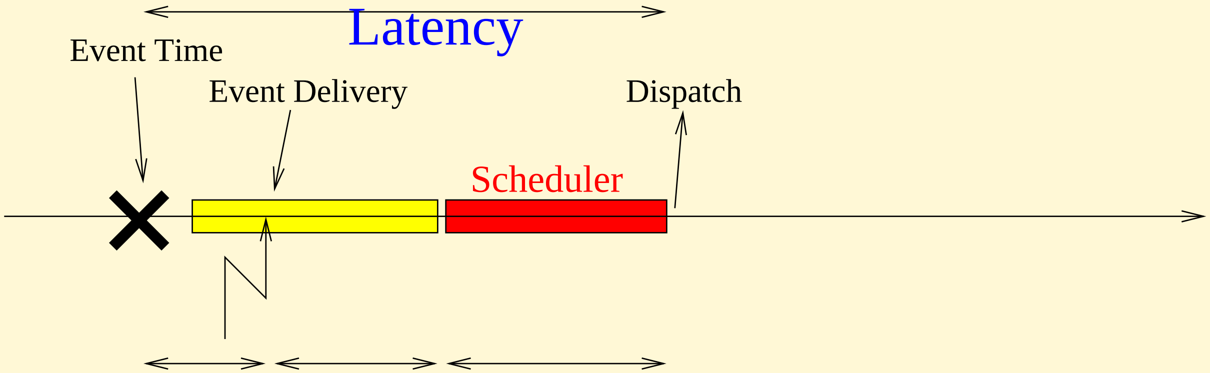
- What happens if the 2^{nd} job of τ_1 arrives a little bit later???
  - The 2^{nd} job of τ_2 misses a deadline!!!

Kernel Latency

- The delay δ in scheduling a task is due to *kernel latency*
- Kernel latency can be modelled as a blocking time
  - $\sum_{k=1}^{N} \frac{C_k}{T_k} \leq U_{lub} \rightarrow \forall i,\ 1 \leq i \leq n, \sum_{k=1}^{i-1} \frac{C_k}{T_k} + \frac{C_i + \delta}{T_i} \leq U_{lub}$
  - $R_i = C_i + \sum_{h=1}^{i-1} \left\lceil \frac{R_i}{T_h} \right\rceil C_h \rightarrow R_i = C_i + \delta + \sum_{h=1}^{i-1} \left\lceil \frac{R_i}{T_h} \right\rceil C_h$
  - $\exists 0 \leq t \leq D_i : W_i(0,t) = C_i + \sum_{h=1}^{i-1} \left\lceil \frac{t}{T_h} \right\rceil C_h \leq t \rightarrow$
    $\exists 0 \leq t \leq D_i : W_i(0,t) = C_i + \sum_{h=1}^{i-1} \left\lceil \frac{t}{T_h} \right\rceil C_h \leq t - \delta$

Kernel Latency

- Scheduler $\rightarrow$ triggered by internal (IPC, signal, ...) or external (IRQ) events
- Time between the triggering event and dispatch:
  - Event generation
  - Event delivery (interrupts may be disabled)
  - Scheduler activation (nonpreemptable sections)
  - Scheduling time



Theoretical Model vs Real Schedule

- In real world, high priority tasks often suffer from blocking times coming from the OS (more precisely, from the kernel)
  - Why?
  - How?
  - What can we do?
- To answer the previous questions, we need to recall how the hardware and the OS work...