

Periodic task scheduling

An exact test for EDF

Relative deadlines less than periods

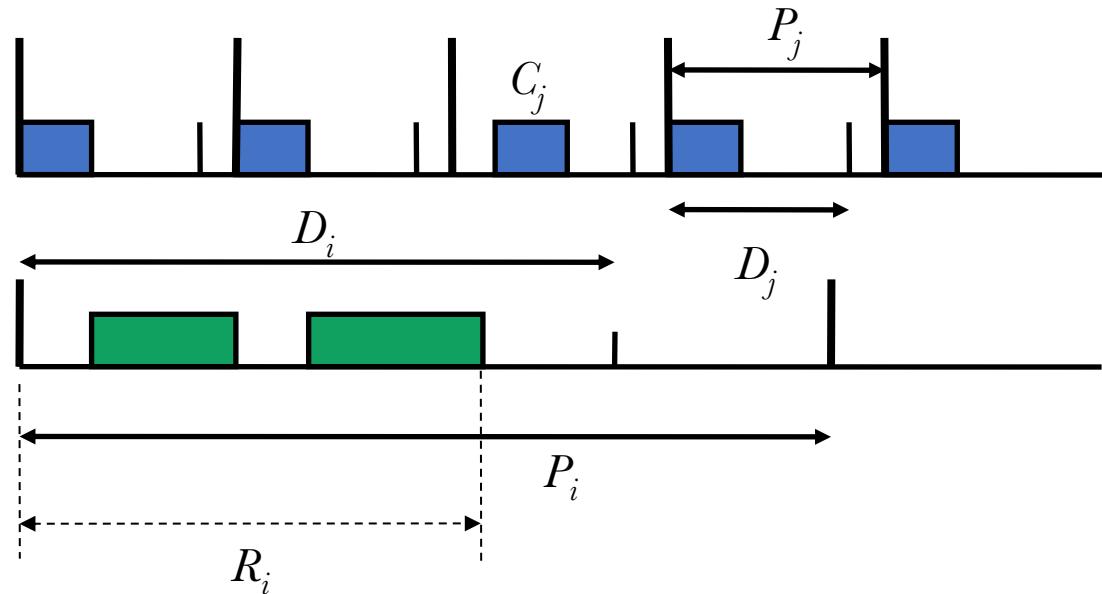
Processor demand criterion

The earliest deadline first policy

- Optimal scheduling policy when relative deadlines are equal to task periods
- Is a **task**-dynamic priority policy (but *not job*-dynamic)
- How does it behave ***when relative deadlines are less than periods?***
- Exact analysis for EDF

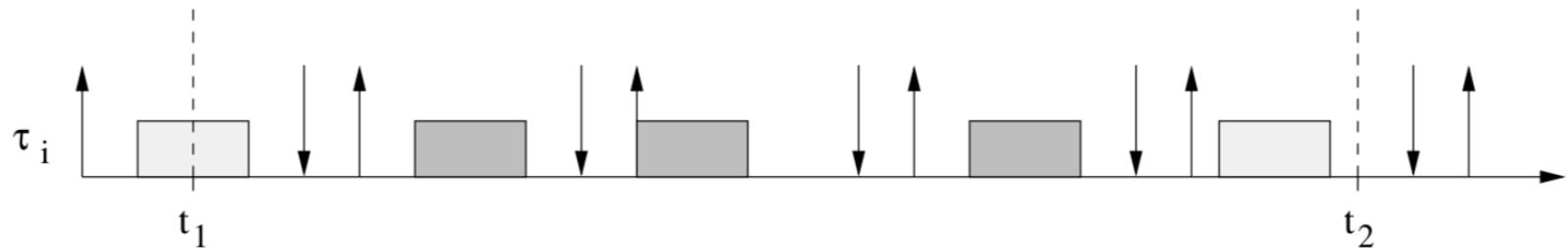
Why not analyze task interference as we did with DM?

- Interference is due to only those jobs with **earlier absolute deadlines**
- Task j exercises interference on task i , and task i exercises interference on task j
- Interference relations are complex due to dynamic priorities → *will abandon this view*



EDF and processor demand

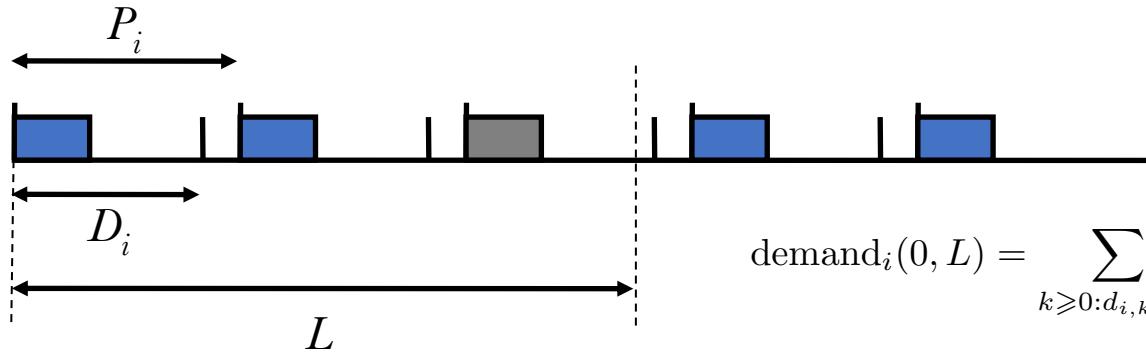
- Task i **requests** service (work) C_i at every arrival $r_{i,k}$
- A request becomes **demand** only at the deadline $d_{i,k}$
 - Before the deadline, we are not obliged to complete the request from a feasibility perspective



$$\text{demand}_i(t_1, t_2) = \sum_{\substack{k \geq 0: r_{i,k} \geq t_1, \\ d_{i,k} \leq t_2}} C_i$$

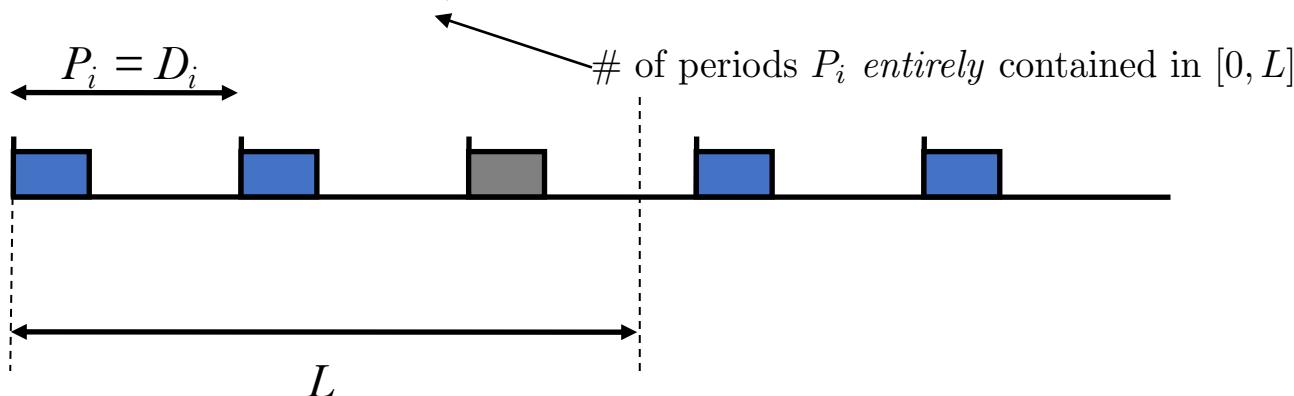
EDF and processor demand

- Task i **requests** service (work) C_i at every arrival $r_{i,k}$
- What is the demand in $[0, L]$ for $L > 0$? ($t_1 = 0$ and $t_2 = L$)



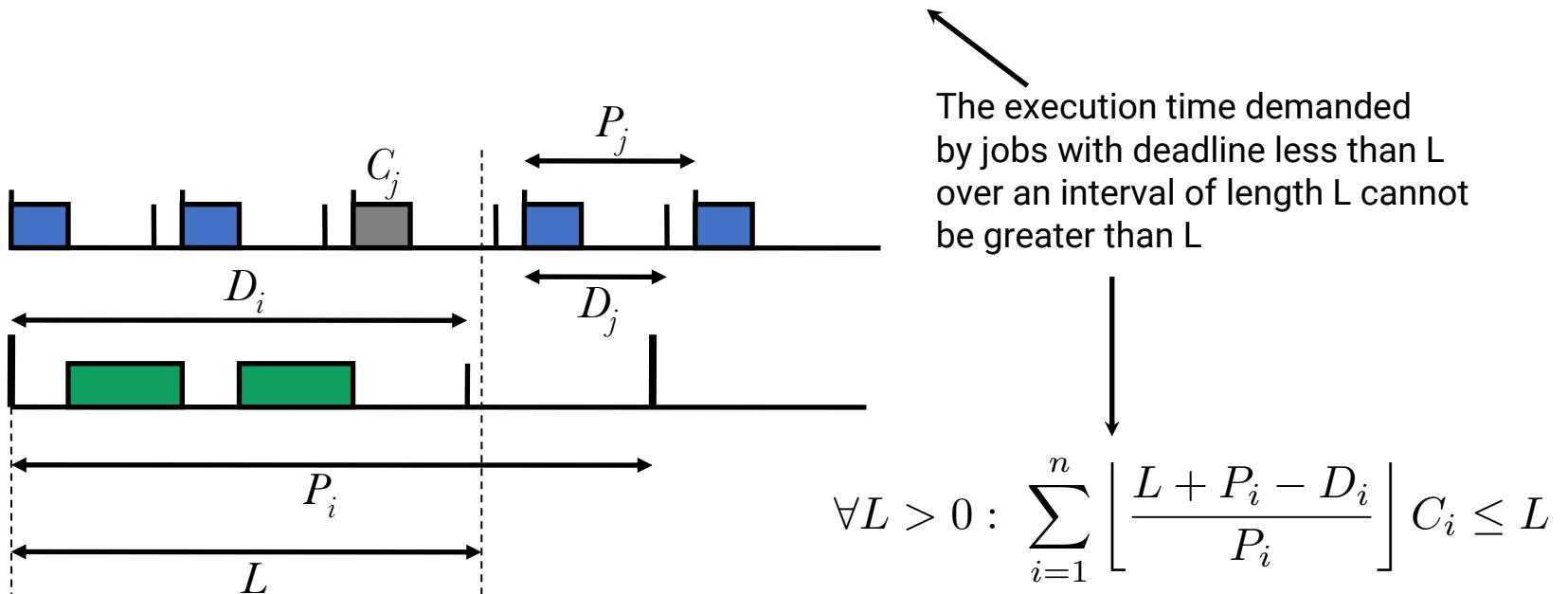
$$\text{demand}_i(0, L) = \sum_{k \geq 0: d_{i,k} \leq L} C_i = \left\lfloor \frac{L + P_i - D_i}{P_i} \right\rfloor C_i$$

If $P_i = D_i \Rightarrow \text{demand}_i(0, L) = \left\lfloor \frac{L}{P_i} \right\rfloor C_i$



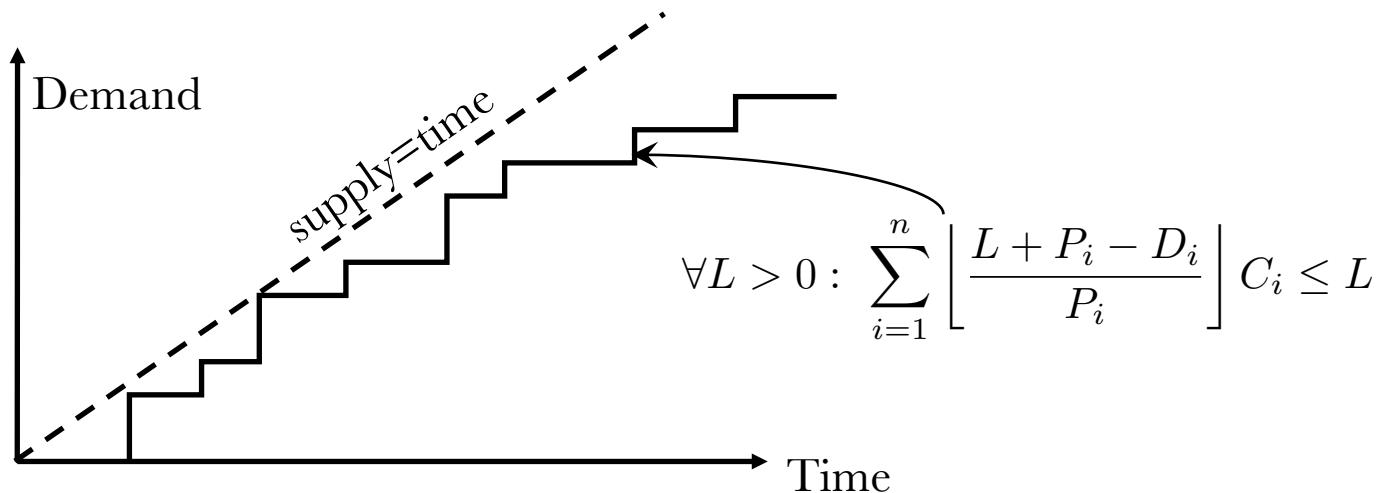
EDF and processor demand

- Consider demand on the processor due to tasks whose deadlines have passed
- Within any time interval, L, the demand must be less than L.



Processor demand

- Checking the schedulability for all time instants is not possible
- Overwhelming complexity
- **Observation 1:** Sufficient to check up to the hyperperiod (schedule repeats itself)
- **Observation 2:** Check only at absolute deadlines. Why?

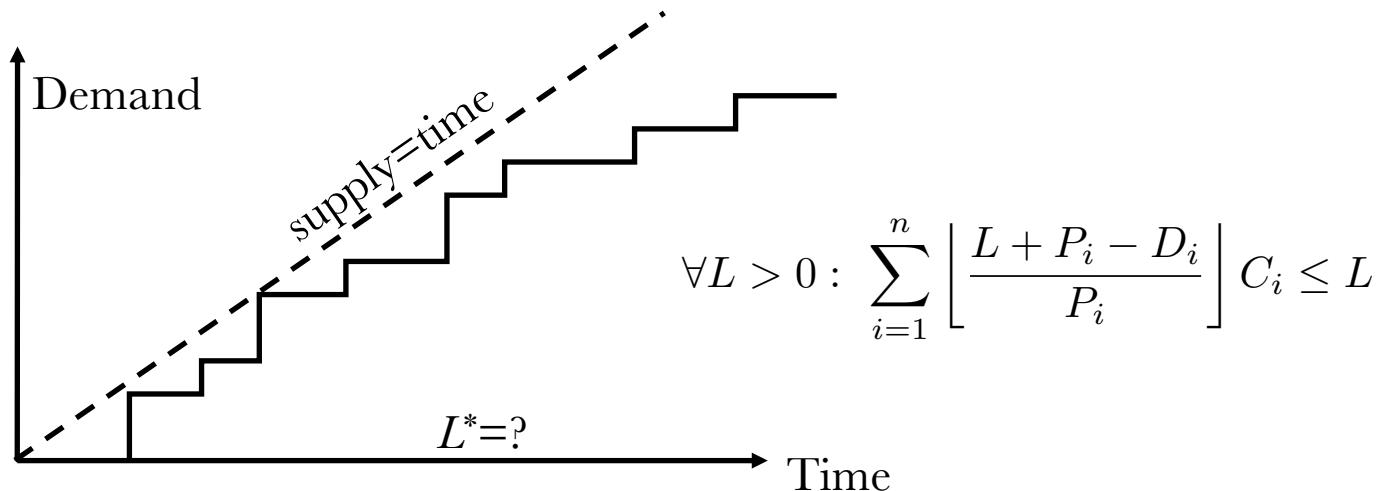


The map $t \mapsto \left\lfloor \frac{t+P_i-D_i}{P_i} \right\rfloor$ is a staircase function with jumps (discontinuities) at absolute deadlines $d_{i,k} = (k-1)P_i + D_i$

Processor demand

- Checking the schedulability for all time instants is not possible
- Overwhelming complexity
- **Observation 1:** Sufficient to check up to the hyperperiod (schedule repeats itself)
- **Observation 2:** Check only at absolute deadlines. Why?
- Can we do better?

Find a point L^* such that if $\text{demand}(0, L) \leq L$ for all $L \leq L^*$, then $\text{demand}(0, L) \leq L$ for all $L \geq L^*$ \rightarrow sufficient to test only up to L^*

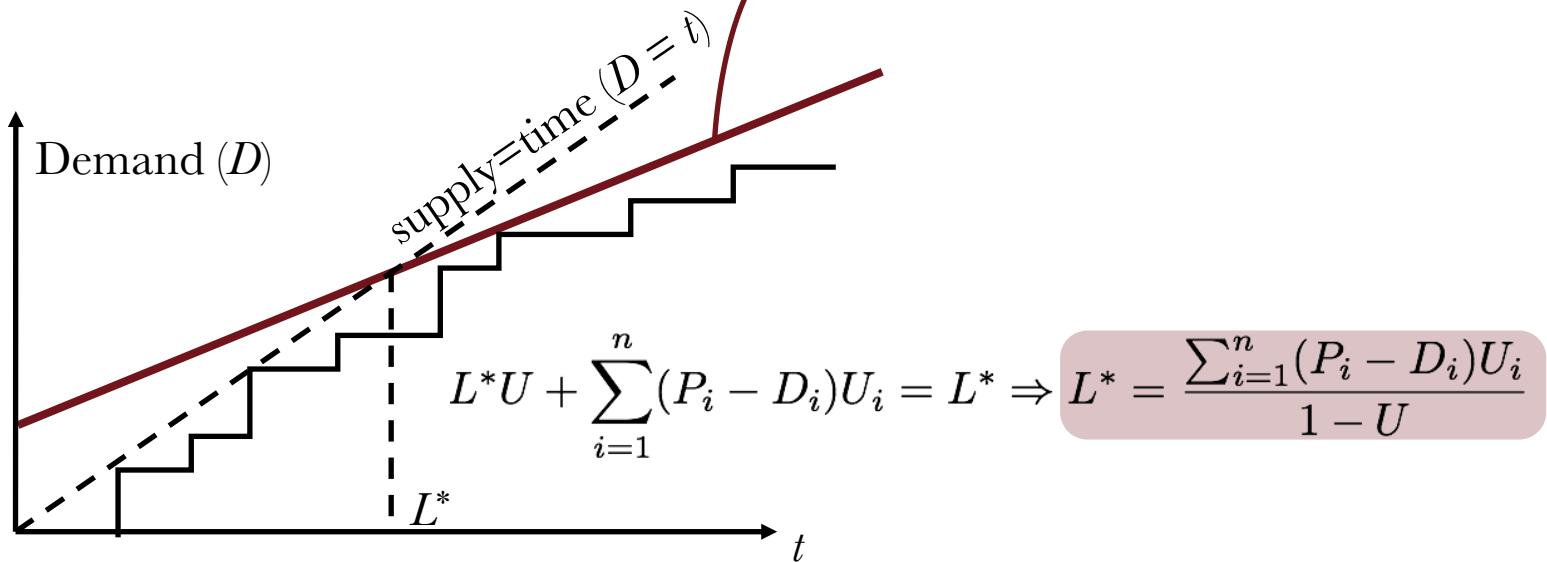


Processor demand

- Deriving L^*

$$\sum_{i=1}^n \left\lfloor \frac{L + P_i - D_i}{P_i} \right\rfloor C_i \leq L$$

$$\sum_{i=1}^n \left\lfloor \frac{t + P_i - D_i}{P_i} \right\rfloor C_i \leq \sum_{i=1}^n \frac{t - D_i + P_i}{P_i} C_i = tU + \sum_{i=1}^n (P_i - D_i)U_i$$



Processor demand criterion

- Check if

$$\sum_{i=1}^n \left\lfloor \frac{L + P_i - D_i}{P_i} \right\rfloor C_i \leq L$$

for all L that are absolute deadlines in the interval $[0, L^*]$, where

$$L^* = \frac{\sum_{i=1}^n (P_i - D_i) U_i}{1 - U}$$

- This is an **exact** test for EDF when relative deadlines are **less** than task periods
- What is the complexity of this test? Is it polynomial-time?
- **Complexity:** Testing constrained-deadline EDF-schedulability is **coNP-hard**.
(Eisenbrand & Rothvoß - SODA'10)

Example using processor demand

- Consider the following task set
 - $T_1: (C_1=1, P_1=3, D_1=2)$
 - $T_2: (C_2=2, P_2=7, D_2=5.5)$
 - $T_3: (C_3=2, P_3=10, D_3=6)$
- $U = 86/105 = 0.8190$
- The task set has a hyperperiod of 210
- However, we only need to test deadlines up to $L^* = 8.63$
- At $L = 2$:

$$L^* = \frac{\sum_{i=1}^n (P_i - D_i) U_i}{1 - U}$$

$$\lfloor \frac{L + P_1 - D_1}{P_1} \rfloor C_1 = \lfloor \frac{2 + 3 - 2}{3} \rfloor 1 = 1$$

Example using processor demand

- Consider the following task set
 - $T_1: (C_1=1, P_1=3, D_1=2)$
 - $T_2: (C_2=2, P_2=7, D_2=5.5)$
 - $T_3: (C_3=2, P_3=10, D_3=6)$
- $U = 86/105 = 0.8190$
- The task set has a hyperperiod of 210
- However, we only need to test deadlines up to $L^* = 8.63$
- At $L=5.5$: (also need to check at $L=5$; not shown here)

$$L^* = \frac{\sum_{i=1}^n (P_i - D_i) U_i}{1 - U}$$

$$\lfloor \frac{L + P_1 - D_1}{P_1} \rfloor C_1 + \lfloor \frac{L + P_2 - D_2}{P_2} \rfloor C_2 = \lfloor \frac{5.5 + 3 - 2}{3} \rfloor 1 + \lfloor \frac{5.5 + 7 - 5.5}{7} \rfloor 2 = 2 + 2 = 4$$

Example using processor demand

- Consider the following task set
 - $T_1: (C_1=1, P_1=3, D_1=2)$
 - $T_2: (C_2=2, P_2=7, D_2=5.5)$
 - $T_3: (C_3=2, P_3=10, D_3=6)$
- $U = 86/105 = 0.8190$
- The task set has a hyperperiod of 210
- However, we only need to test deadlines up to $L^* = 8.63$
- At $L=6$:

$$L^* = \frac{\sum_{i=1}^n (P_i - D_i) U_i}{1 - U}$$

$$\lfloor \frac{6 + 3 - 2}{3} \rfloor 1 + \lfloor \frac{6 + 7 - 5.5}{7} \rfloor 2 + \lfloor \frac{6 + 10 - 6}{10} \rfloor 2 = 6$$

Example using processor demand

- Consider the following task set

- T₁: (C₁=1, P₁=3, D₁=2)
- T₂: (C₂=2, P₂=7, D₂=5.5)
- T₃: (C₃=2, P₃=10, D₃=6)

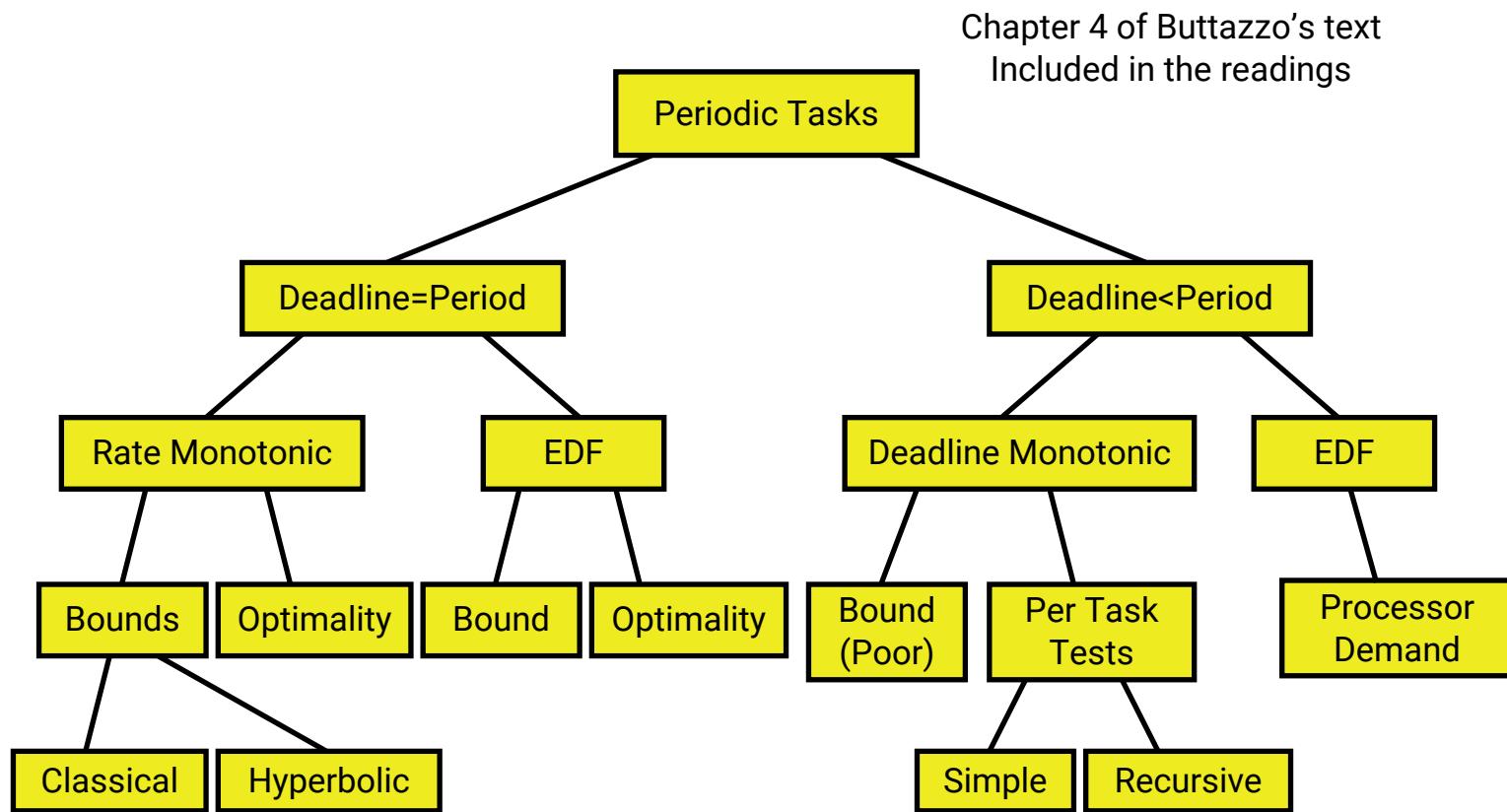
$$L^* = \frac{\sum_{i=1}^n (P_i - D_i)U_i}{1 - U}$$

- $U = 86/105 = 0.8190$
- The task set has a hyperperiod of 210
- However, we only need to test deadlines up to $L^* = 8.63$
- At $L=8$:

$$\lfloor \frac{8+3-2}{3} \rfloor 1 + \lfloor \frac{8+7-5.5}{7} \rfloor 2 + \lfloor \frac{8+10-6}{10} \rfloor 2 = 3 + 2 + 2 = 7$$

No more absolute deadlines < 8.63. We are done!

Topics covered so far



In-class Activity 1

A single on-board computer controls several features on a new car. The on-board computer plays music from a local hard-disk, and this task (Task T_1) requires 20ms of time every 100ms. The GPS system (Task T_2) that provides directions requires 30ms of execution time every 250ms. The one other task that runs on this computer manages the temperature and humidity in the car; this task (Task T_3) is performed every 400ms and requires 100ms of execution each period. Tasks are scheduled using the **rate monotone** priority assignment.

1. You are a new engineer on this project and you have been assigned the task of integrating an additional feature on this on-board computer. This feature, a traffic monitor, scans a special communications channel and identifies routes that are congested or under repair. The traffic monitor (Task T_4), as it is currently implemented, runs for 100ms every 280ms. You need to determine if this task can be introduced without causing any deadline violations. If not, you need to instruct the feature engineers to redesign the traffic monitor and reduce its execution time (you cannot alter the frequency because it has been determined to provide drivers with sufficient time to change routes). What would your recommendation be?

In-class Activity 2

A single on-board computer controls several features on a new car. The on-board computer plays music from a local hard-disk, and this task (Task T_1) requires 20ms of time every 100ms. The GPS system (Task T_2) that provides directions requires 30ms of execution time every 250ms. The one other task that runs on this computer manages the temperature and humidity in the car; this task (Task T_3) is performed every 400ms and requires 100ms of execution each period. Tasks are scheduled using the **rate monotone** priority assignment.

2. Assume that only the original three tasks (T_1, T_2, T_3) are running on the on-board computer. In a redesign stage, it is determined that these tasks need to update a display by sending some information over a data bus. This communication takes time but the display needs to be updated within the task's period. As a result, the relative deadlines for the three tasks need to be shortened and the tasks scheduled using the **deadline monotone** priority assignment. For simplicity, all tasks will have their deadlines reduced by a factor f . In other words, the relative deadline D_i for task T_i will become $D_i = f \cdot P_i$ where P_i is the period of the task. What is the smallest value of f such that the tasks will continue to meet their deadlines.

RM vs. EDF: Judgement Day

- Which policy incurs more preemptions?
- Which has the edge in implementation simplicity and running time complexity?
- How does each behave under **transient** and **permanent** overload conditions?
 - Is one more predictable than the other when overload happens? In what sense?
- Release jitter



Misconceptions?

- EDF incurs more preemptions than RM
- RM is easier to analyze than EDF
- RM introduces less runtime overhead
- EDF is not predictable in transient and permanent overloads
- RM is more predictable in overload conditions, and causes less jitter in task execution



Generally, these statements are either wrong or not precise!
We need to look more carefully into the situation



Implementation Cost

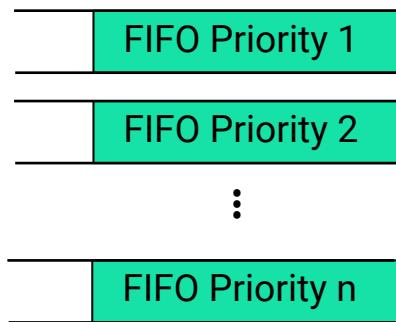
- For a precise comparison, must distinguish two situations
 - **Case 1:** Scheduler is developed on top of a generic priority-based operating system
 - **Case 2:** Scheduler is developed from scratch

Implementation Cost

- **Case 1:** Scheduler is developed on top of a generic priority-based operating system (limited number of priority levels available)
- **RM wins!**
 - Most commercial kernels contain a *fixed* number of priority levels (nominally 256)
- Consider the case in which two deadlines d_a and d_b are mapped into two adjacent priority levels
- A new periodic instance is released with absolute deadline d_c s.t. $d_a < d_c < d_b$.
- There is no priority level that can be selected to map d_c , even when the number of active tasks is less than the number of priority levels in the kernel.
- **Solution:** remap d_a and d_b into two new priority levels that are not consecutive → in worst case, all current deadlines might need to be remapped (high cost)

Implementation Cost

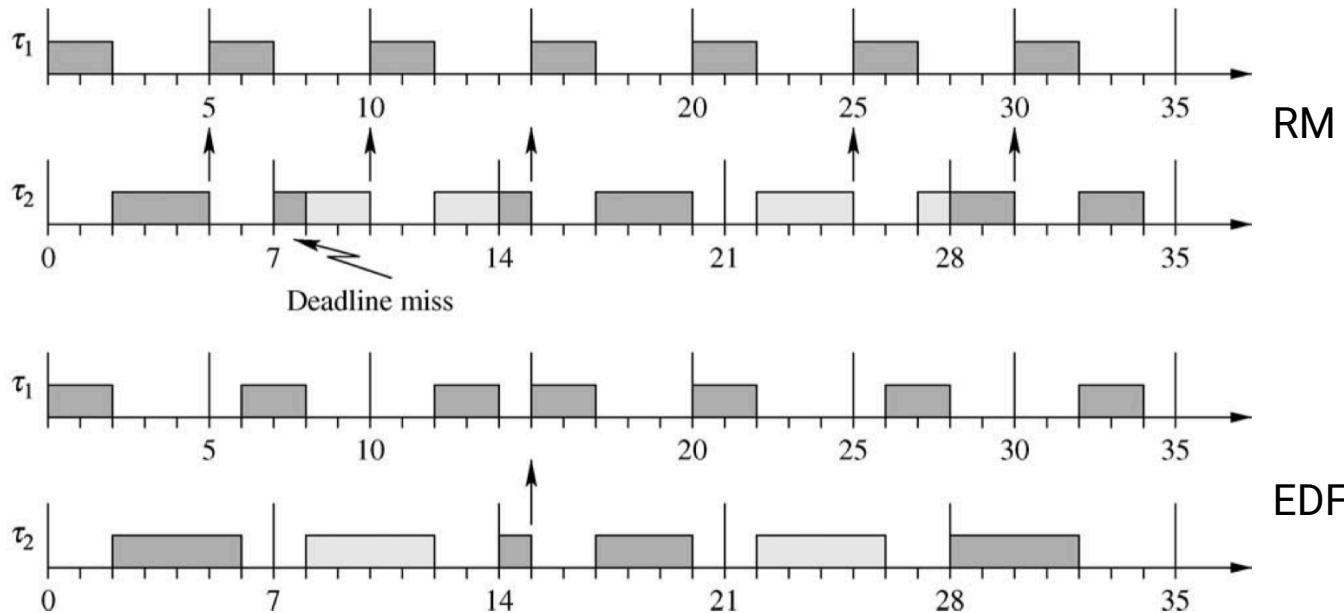
- **Case 2:** Scheduler is developed from scratch
- **A tie!**
- **RM has a tiny edge:** If # of priority levels is low, can split ready queue into separate FIFO queues, one for each priority level → Insertion takes $O(1)$



- This is not possible for EDF → number of queues would be too large (e.g., equal to 2^{32} if system **time** is represented by four byte variables).

Runtime Cost

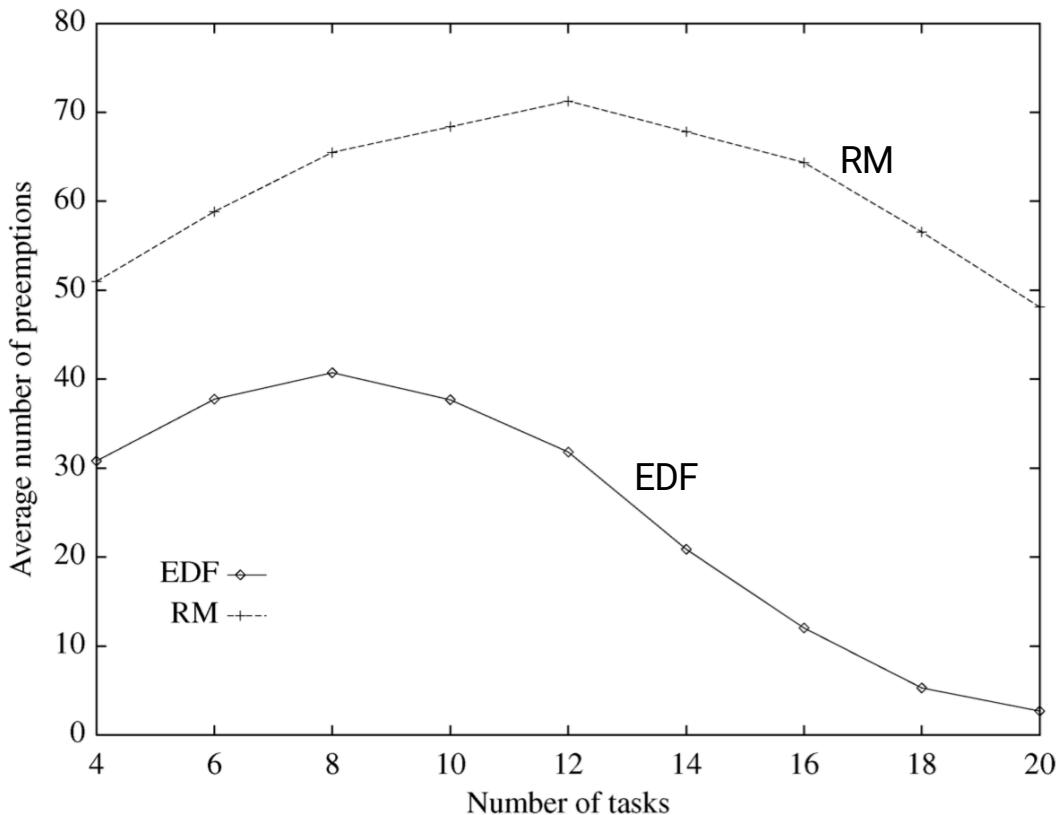
- EDF incurs **less** context switches than RM!



- Under RM, to respect the priority order given by periods, the high priority task τ_1 must preempt **every** instance of τ_2 , whereas under EDF preemption occurs only once in the entire hyperperiod

Context Switching Cost: Interesting Phenomenon

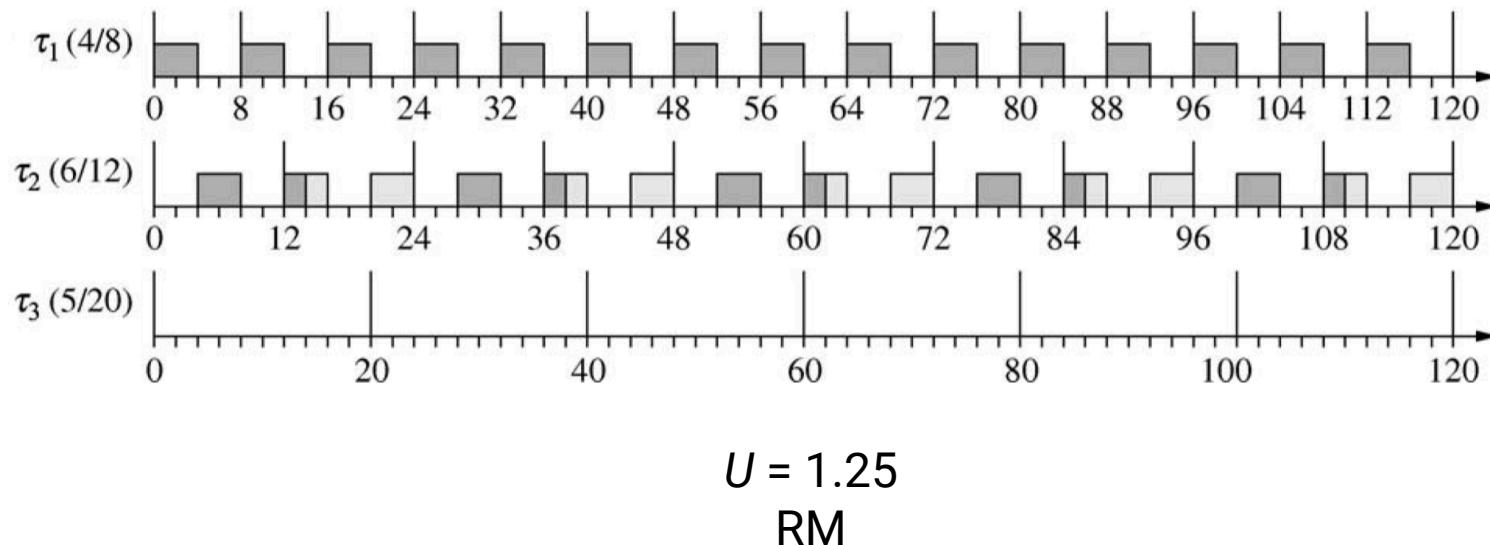
- Utilization kept at 0.9 as number of tasks increases



For each point in the graph, the average was computed over 1000 independent simulations, each running for 1000 units of time. In each simulation, periods were generated as random variables with uniform distribution in the range of 10 to 100 units of time, whereas execution times were computed to create a total processor utilization $U = 0.9$.

Predictability and Stability: Permanent Overload

- Permanent Overload $\rightarrow U > 1$
- Behavior of both RM and EDF *is predictable* in some sense
- **RM:** Might cause complete blocking of *lower* priority tasks



Predictability and Stability: Permanent Overload

- Permanent Overload $\rightarrow U > 1$
- **EDF:** Interesting phenomenon happens; *Period rescaling*
 - In the long run, EDF will behave such that it appears that tasks were released at a lower rate $\rightarrow \bar{P}_i = UP_i$

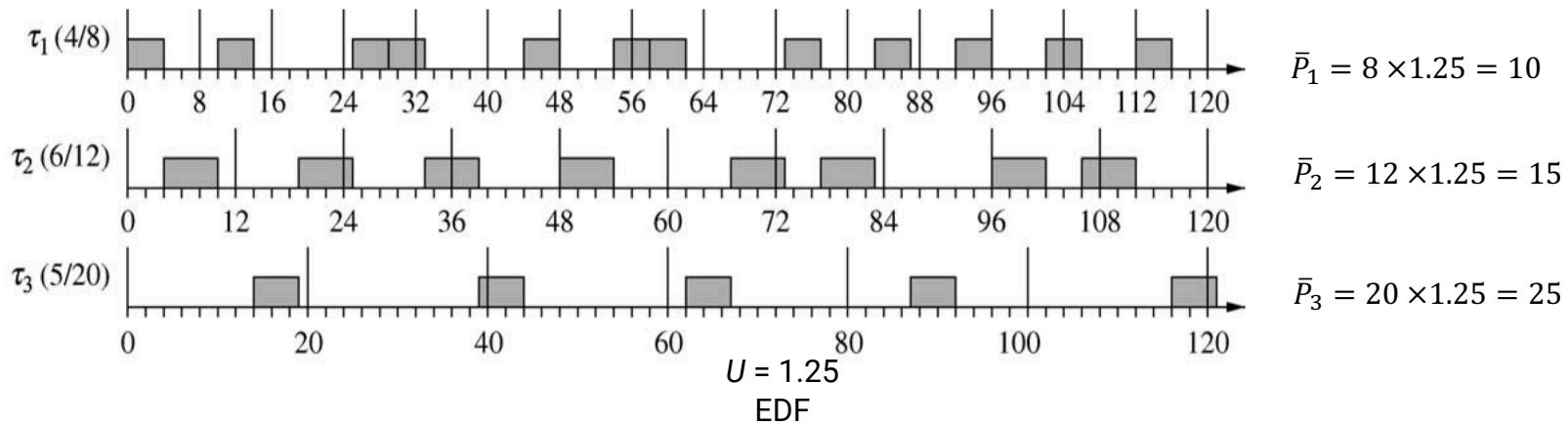
Theorem 1 [Cervin]. Assume a set of n periodic tasks, where each task is described by a fixed period T_i , a fixed execution time C_i , a relative deadline D_i , and a release offset Φ_i . If $U > 1$ and tasks are scheduled by EDF, then, in stationarity, the average period \bar{T}_i of each task τ_i is given by $\bar{T}_i = T_i U$.

$$\bar{T}_i := \lim_{t \rightarrow \infty} \frac{t}{k_i(t)} = T_i U$$

$k_i(t)$: # of instances of task τ_i that completed up to time $t > 0$

Predictability and Stability: Permanent Overload

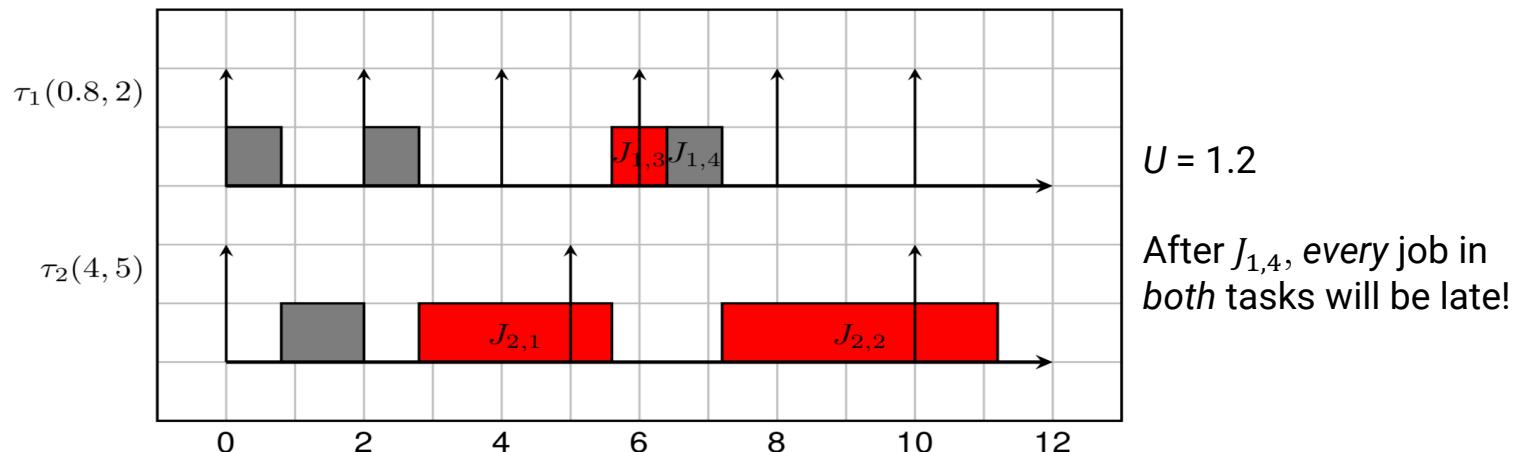
- Permanent Overload $\rightarrow U > 1$
- **EDF** Period rescaling



$$\bar{U} = \frac{c_1}{\bar{P}_1} + \frac{c_2}{\bar{P}_2} + \frac{c_3}{\bar{P}_3} = \frac{4}{10} + \frac{6}{15} + \frac{5}{25} = 1$$

A deadline miss in EDF can be dangerous

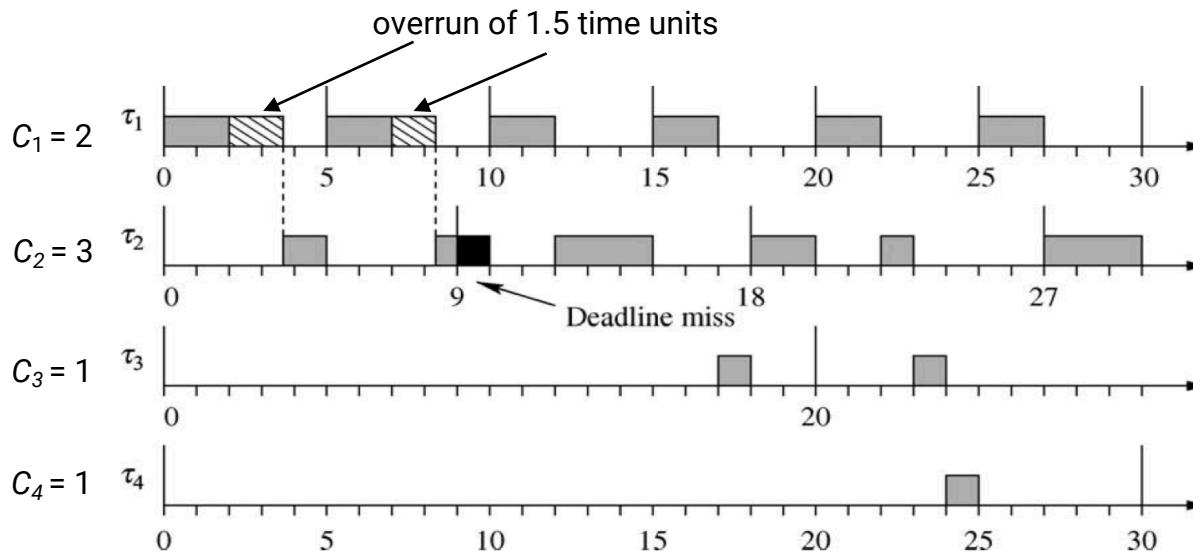
- If job misses deadline it executes at a very high priority level (higher than all tasks that have not missed their deadlines)
- If late job allowed to continue, will cause other jobs to miss their deadlines!



- Need overrun management strategy (application dependent)
 - E.g., schedule each late job at a lower priority than jobs that are not late
 - Drop late job altogether
 - Enforce execution budget through reservation-based approaches (periodic servers) → soon!

Predictability and Stability: Transient Overload

- **Misconception:** In RM, in the presence of transient overload conditions, deadlines are missed predictably, that is, the **first** tasks that fail are those with the **longest period** (lowest priority tasks fail first)



RM schedule: It's **not** τ_4 that misses its deadline, but the higher priority τ_2 !

If a task τ_i overruns, only the tasks with *higher priority than τ_i* are protected under **RM**, but nothing can be ensured for the other tasks, not even τ_i

- **EDF:** situation is NOT better than RM!

Handling aperiodic work

- EDF has the edge since it has higher utilization
- **Slack-stealing:** If U_p is utilization of periodic work, then the remaining $1 - U_p$ utilization can be used for aperiodic tasks
 - Increased utilization given to periodic servers (discussed later)

Lecture summary

- Exact test for EDF scheduling (relative deadlines < periods)
- Processor demand criterion
- RM vs. EDF

Giorgio C. Buttazzo. **Rate monotonic vs. EDF: Judgment day.**
Real-Time Systems, 29:5–26. 2005