Embedded and Real-Time Systems

Spring 2021

Hamed Farbeh

farbeh@aut.ac.ir

Department of Computer Engineering

Amirkabir University of Technology

Lecture 11

Copyright Notice

This lecture is adopted from

IN4343 Real-Time Systems Course 2018 – 2019, Mitra Nasri, Delft University of Technology

Non-preemptive Scheduling

Paper 1:

Mitra Nasri and Gerhard Fohler, "Non-Work-Conserving Non-preemptive Scheduling: Motivations, Challenges, and Potential Solutions," in the Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS), 2016, pp. 165-175.

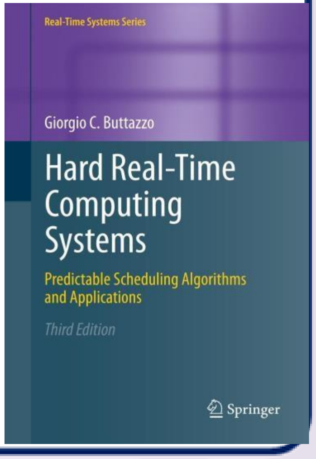
Paper 2:

Mitra Nasri and Björn B. Brandenburg, "An Exact and Sustainable Analysis of Non-Preemptive Scheduling", in the Proceedings of the Real-Time Systems Symposium (RTSS), 2017, pp. 1-12

Disclaimer: A few slides have been taken from Giorgio Buttazzo's website:

http://retis.sssup.it/~giorgio/rts-MECS.html

Buttazzo's book, chapter 4, 8



Agenda

- EDF schedulability analysis — Book: chapter 4
  - **Optimality of EDF for periodic tasks**
  - **EDF response-time analysis**
- Non-preemptive scheduling
  - **Why non-preemptive execution?** — Book: chapter 8
  - **Existing schedulability analysis for NP-FP and NP-EDF** — Book: chapter 8; Paper 1, 2
  - **Behind the scenes!**

    Things you need to know about work-conserving NP scheduling!

- What is the hyperperiod of
  5, 10, 20?

20

- What is the hyperperiod of
  5, 10, 21?

210

EDF

Optimality

The proof comes from Dertouzos 1974

Assumptions:

- Fully preemptive
- Independent tasks
- No self-suspension
- No scheduling overhead
- $\forall i,\ D_i = T_i$

Schedulability test

- **Liu and Layland 1973**

  A task set τ is schedulable by EDF **if and only if:**

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

EDF schedulability analysis for D < T

Processor Demand Criterion [Baruah '90]

In **any interval** of length L, the **computational demand** $g(0, L)$ of the task set must be **no greater** than L.

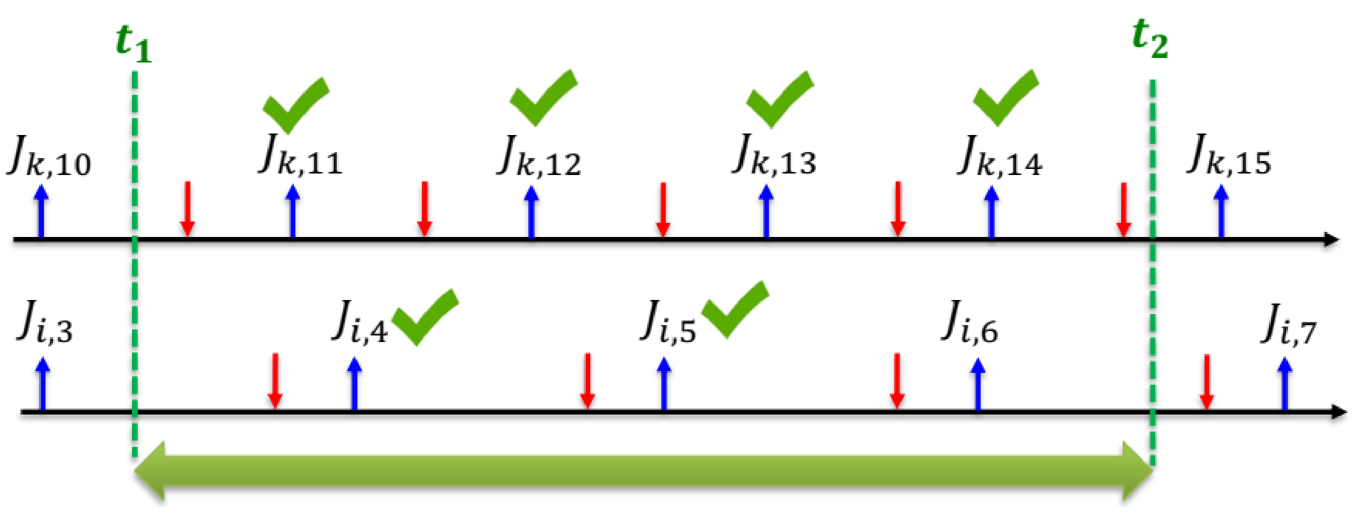
$$\forall L > 0, \qquad g(0, L) \leq L$$

Applicable to periodic tasks with $\forall i, \phi_i = 0$

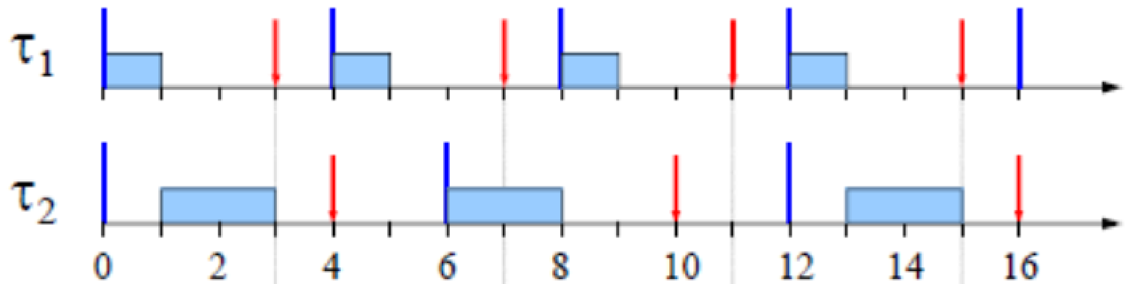
Understanding the processor demand (g function)

- The demand in $[t_1, t_2]$ is the computation time of those tasks <u>arrived</u> **at or after** t_1 with <u>deadline</u> **less than or equal to** t_2:

$$g(t_1, t_2) = \sum_{\forall J_{k,j}, t_1 \le a_{k,j} < d_{k,j} \le t_2} C_k$$



Demand of a periodic task set



$$g_i(0,L) = \max\left\{0, \left\lfloor \frac{L - D_i + T_i}{T_i} \right\rfloor \cdot C_i \right\}$$

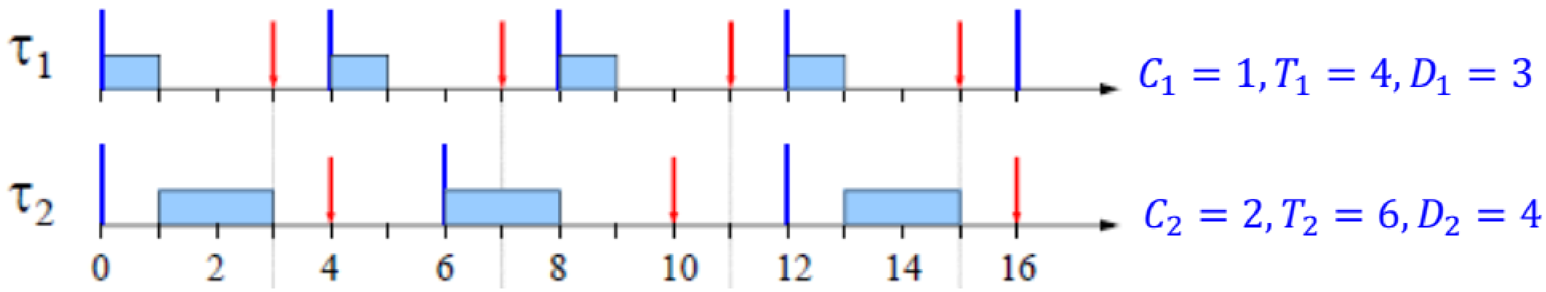
$$g_i(0,L) = \max\left\{0, \left\lfloor \frac{L - D_i}{T_i} + 1 \right\rfloor \cdot C_i \right\}$$

$$g_i(0,L) = \max\left\{0, \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1\right) \cdot C_i \right\}$$

$$\boxed{g(0,L) = \sum_{i=1}^{n} g_i(0,L)}$$

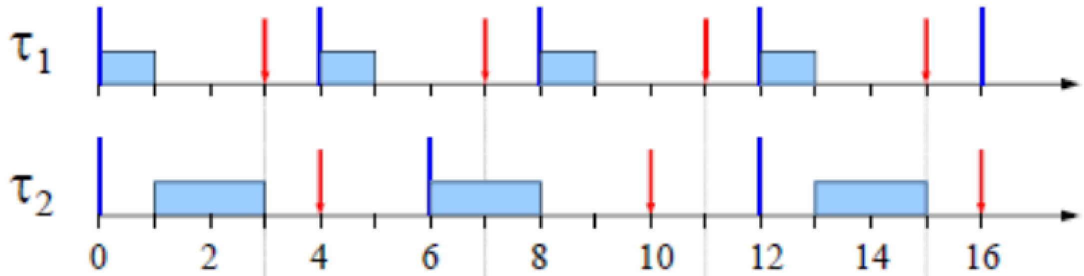
Examples

$$g_i(0,L) = \max\left\{0, \left\lfloor \frac{L - D_i}{T_i} + 1 \right\rfloor \cdot C_i\right\}$$



L	$g_2(0,L)$	L	$g_2(0,L)$
4	$g_2(0,4) = \left\lfloor \frac{4-4}{6} + 1 \right\rfloor \cdot 2 = 2$	9	$g_2(0,9) = \left\lfloor \frac{9-4}{6} + 1 \right\rfloor \cdot 2 = 2$
5	$g_2(0,5) = \left\lfloor \frac{5-4}{6} + 1 \right\rfloor \cdot 2 = 2$	10	$g_2(0,10) = \left\lfloor \frac{10-4}{6} + 1 \right\rfloor \cdot 2 = 4$
6	$g_2(0,6) = \left\lfloor \frac{6-4}{6} + 1 \right\rfloor \cdot 2 = 2$	11	$g_2(0,11) = \left\lfloor \frac{11-4}{6} + 1 \right\rfloor \cdot 2 = 4$
7	$g_2(0,7) = \left\lfloor \frac{7-4}{6} + 1 \right\rfloor \cdot 2 = 2$	15	$g_2(0,15) = \left\lfloor \frac{15-4}{6} + 1 \right\rfloor \cdot 2 = 4$
8	$g_2(0,8) = \left\lfloor \frac{8-4}{6} + 1 \right\rfloor \cdot 2 = 2$	16	$g_2(0,16) = \left\lfloor \frac{16-4}{6} + 1 \right\rfloor \cdot 2 = 6$

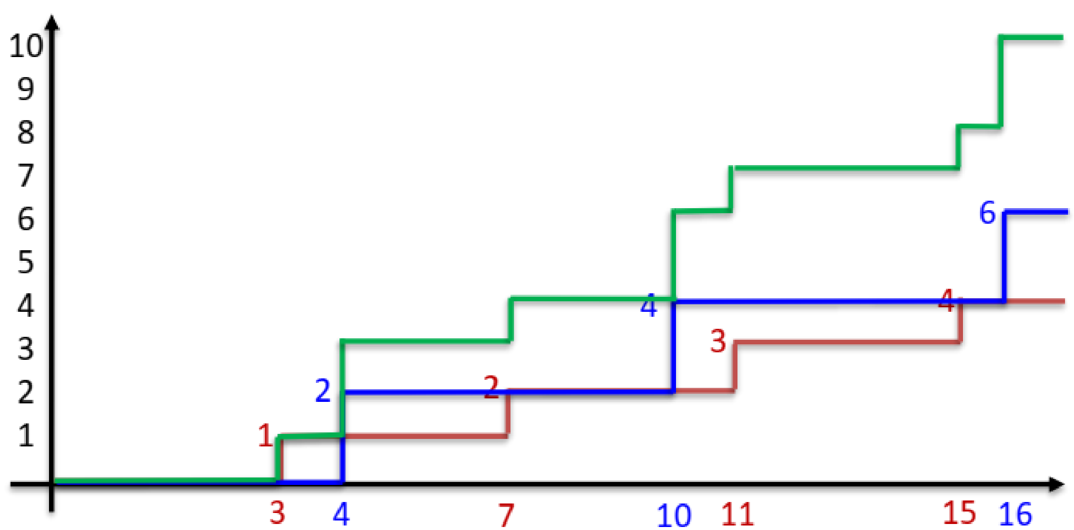
When does Demand Bound Function change?



$C_1 = 1, T_1 = 4, D_1 = 3$

$C_2 = 2, T_2 = 6, D_2 = 4$

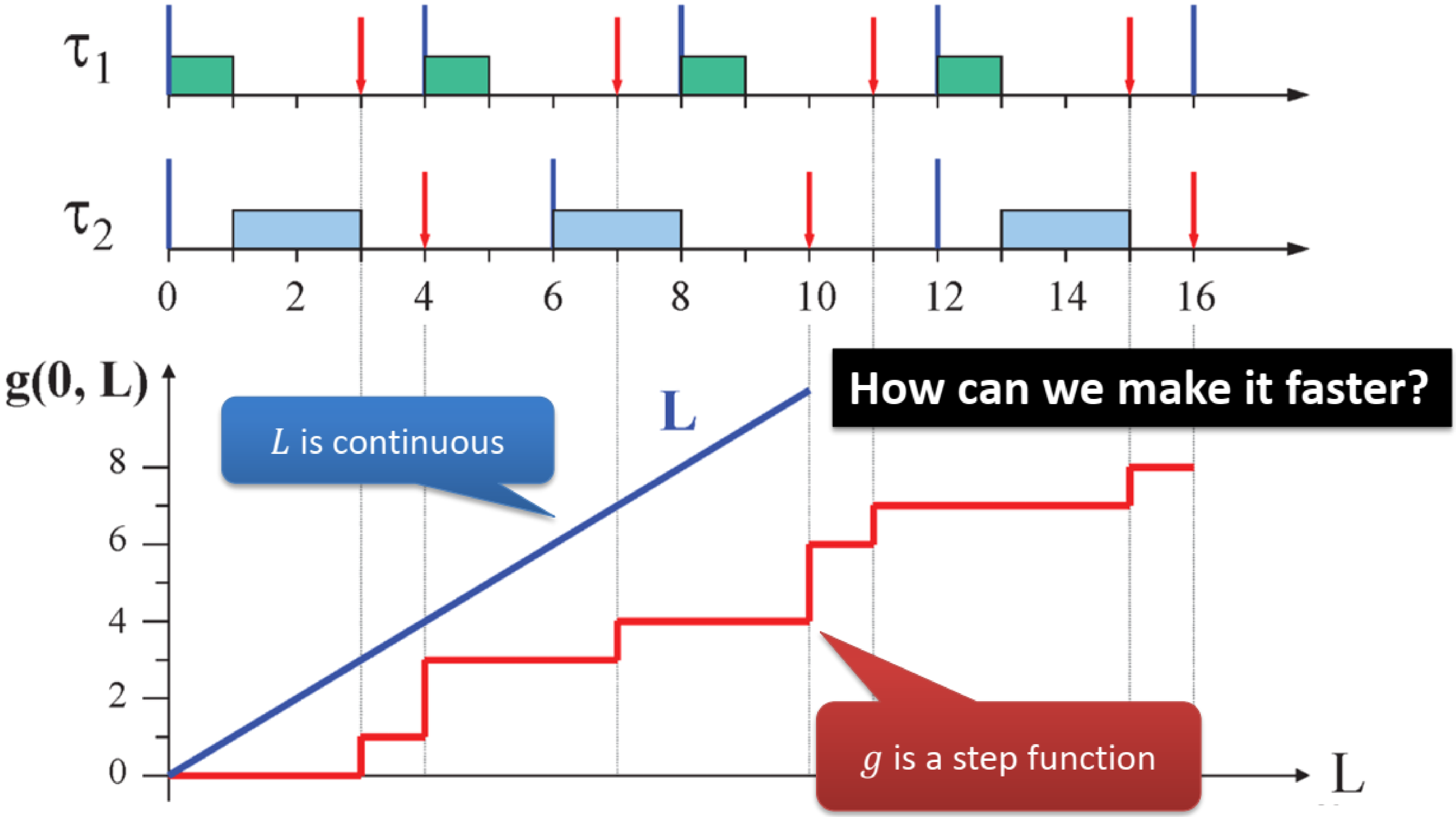
L	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$g_1(0,L)$	0	0	0	**1**	1	1	1	**2**	2	2	2	**3**	3	3	3	**4**	4
$g_2(0,L)$	0	0	0	0	**2**	2	2	2	2	2	**4**	4	4	4	4	4	**6**
$g(0,L)$	0	0	0	**1**	**3**	3	3	**4**	4	4	**6**	**7**	7	7	7	**8**	**10**



$$\forall L > 0, \quad g(0, L) \leq L$$

$$g(0, L) = \sum_{i=1}^{n} g_i(0, L)$$

$$g_i(0, L) = \max\left\{0, \left\lfloor \frac{L - D_i}{T_i} + 1 \right\rfloor \cdot C_i\right\}$$



Bounding complexity

- Since $g(0, L)$ is a step function, we can check feasibility only at **deadline points,** where g changes.

- If tasks are synchronous and $U \leq 1$, we can check the feasibility up to the hyperperiod H:

$$H = lcm(T_1, \dots, T_n)$$

Bounding complexity: finding a safe upper bound for the analysis window

An upper bound on *g*

$$g(0,L) = \sum_{i=1}^{n} \left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor \cdot C_i \leq G(0,L) = \sum_{i=1}^{n} \left(\frac{L + T_i - D_i}{T_i} \right) \cdot C_i$$

$$\boxed{\lfloor x \rfloor \leq x}$$

$$G(0,L) = \sum_{i=1}^{n} L \frac{C_i}{T_i} + \sum_{i=1}^{n} (T_i - D_i) \cdot \frac{C_i}{T_i}$$

$$= L \cdot U + \sum_{i=1}^{n} (T_i - D_i) \cdot U_i$$

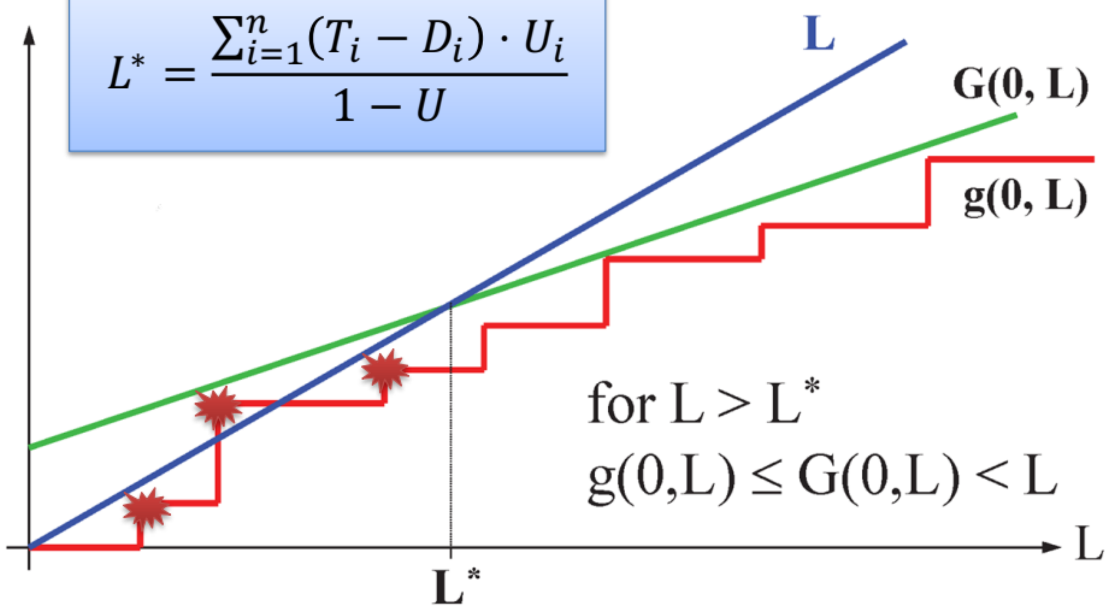
$$g(0,L) < G(0,L)$$

Limiting L

$U \leq 1$

$$G(0, L) = L \cdot U + \sum_{i=1}^{n} (T_i - D_i) \cdot U_i$$

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



For any $L \geq L^*$, it is meaningless to check if $g(0, L) \leq L$ because it will obviously be smaller! Hence, we can stop the search at L^*.

Processor Demand Test

$$\forall L \in D, \qquad g(0, L) \leq L$$

Where D is the set of deadline points for which $g(0, L)$ must be calculated:

$$D = \{d_{i,j} \mid d_{i,j} \leq \min\{H, L^*\}\}$$

$$H = lcm(T_1, \dots, T_n)$$

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

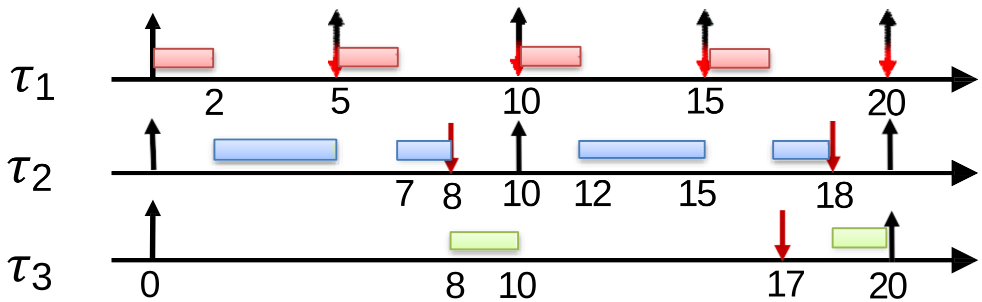
FP v.s. EDF

EDF v.s. FP

τ_i	C_i	T_i	D_i
τ_1	2	5	5
τ_2	4	10	8
τ_3	4	20	17

Context switches

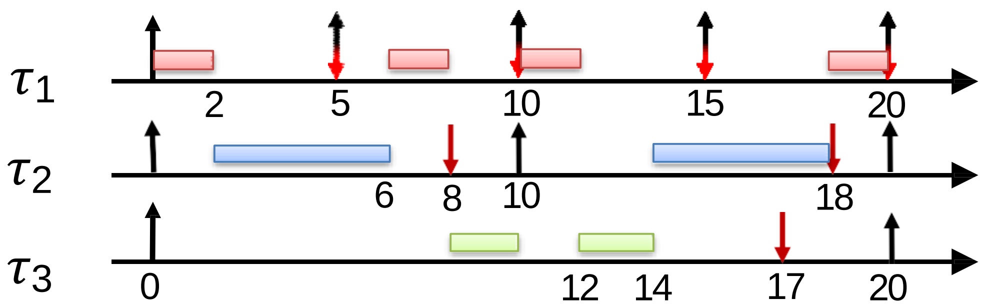
FP



How many preemptions?

3 preemptions in jobs $J_{2,1}, J_{3,1}, J_{2,2}$

EDF



How many preemptions?

1 preemption in job $J_{3,1}$

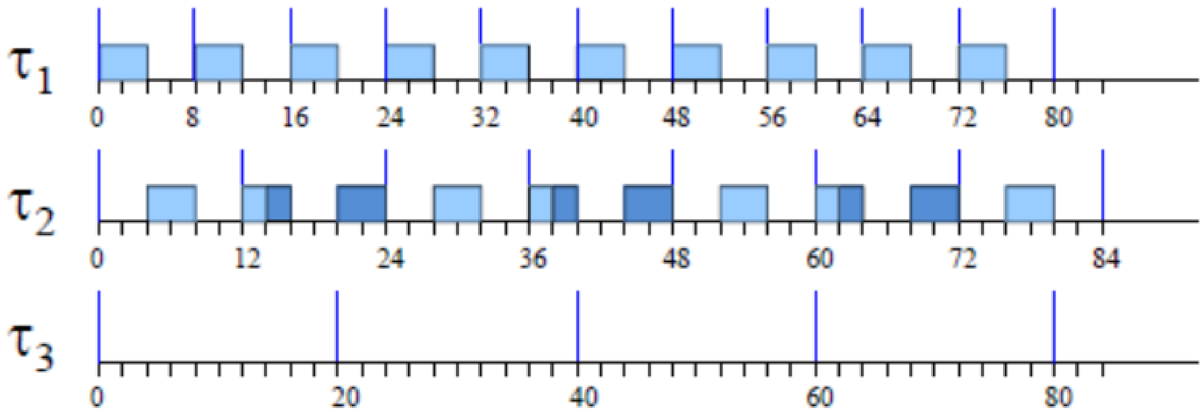
In average, FP has more context switches

EDF v.s. FP

What do you think will happen for FP if U>1?

FP under permanent overload ➡ **starvation for low priority tasks**

$$U = \frac{4}{8} + \frac{6}{12} + \frac{5}{20} = 1.25$$

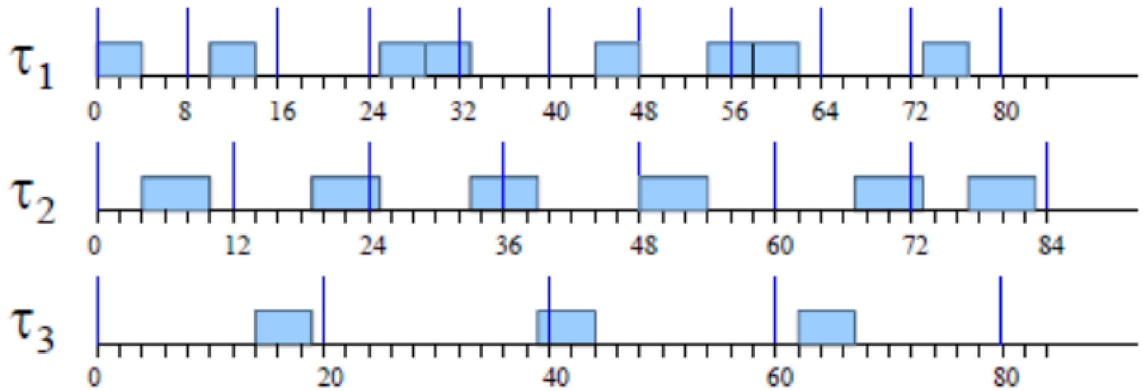


- High priority tasks execute at the proper rate
- Low priority tasks are completely blocked

EDF v.s. FP

EDF under permanent overload

$$U = \frac{4}{8} + \frac{6}{12} + \frac{5}{20} = 1.25$$



- All tasks execute at a slower rate
- No task is blocked

EDF v.s. FP

Schedulability analysis

	$D_i = T_i$	$D_i \leq T_i$
RM	*Suff.: polynomial* $O(n)$ LL: $\Sigma U_i \leq n(2^{1/n} - 1)$ HB: $\Pi(U_i+1) \leq 2$ *Exact* *pseudo-polynomial* RTA	*pseudo-polynomial* Response Time Analysis $\forall i \quad R_i \leq D_i$ $R_i = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$
EDF	*polynomial:* $O(n)$ $\Sigma U_i \leq 1$	*pseudo-polynomial* Processor Demand Analysis $\forall L > 0, \quad g(0, L) \leq L$

FP's response-time analysis is usually faster than EDF

EDF v.s. FP

EDF

- Higher schedulability
- Reduces context switches
- During overloads does not starve low-priority tasks

FP

- Simpler to implement
- Widely implemented in most operating systems
- More predictable during overloads

How does DM compare to EDF?

Agenda

- EDF schedulability analysis
  - Optimality of EDF for periodic tasks
  - EDF response-time analysis

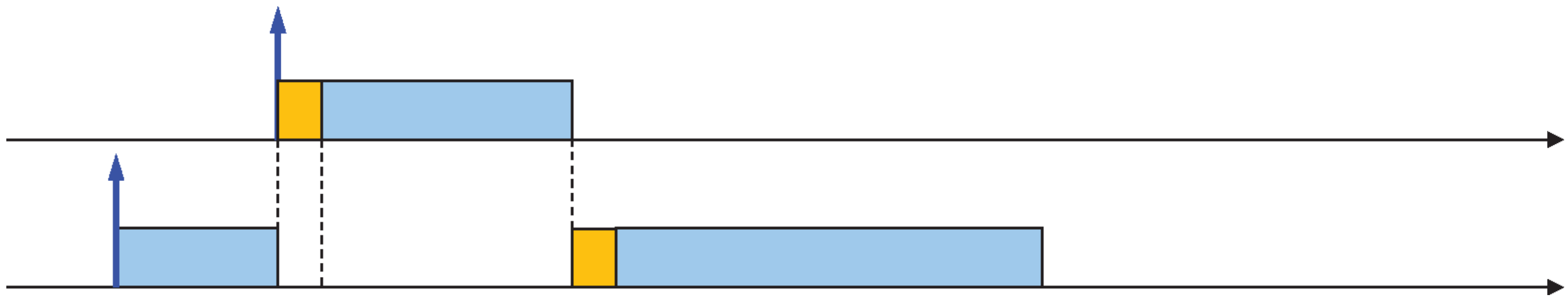
- Non-preemptive scheduling
  - **Why non-preemptive execution?**
  - **Existing schedulability analysis for NP-FP and NP-EDF**
  - **Behind the scenes!**

    Things you need to know about work-conserving NP scheduling!

Disadvantages of preemptions

1- Context switch cost

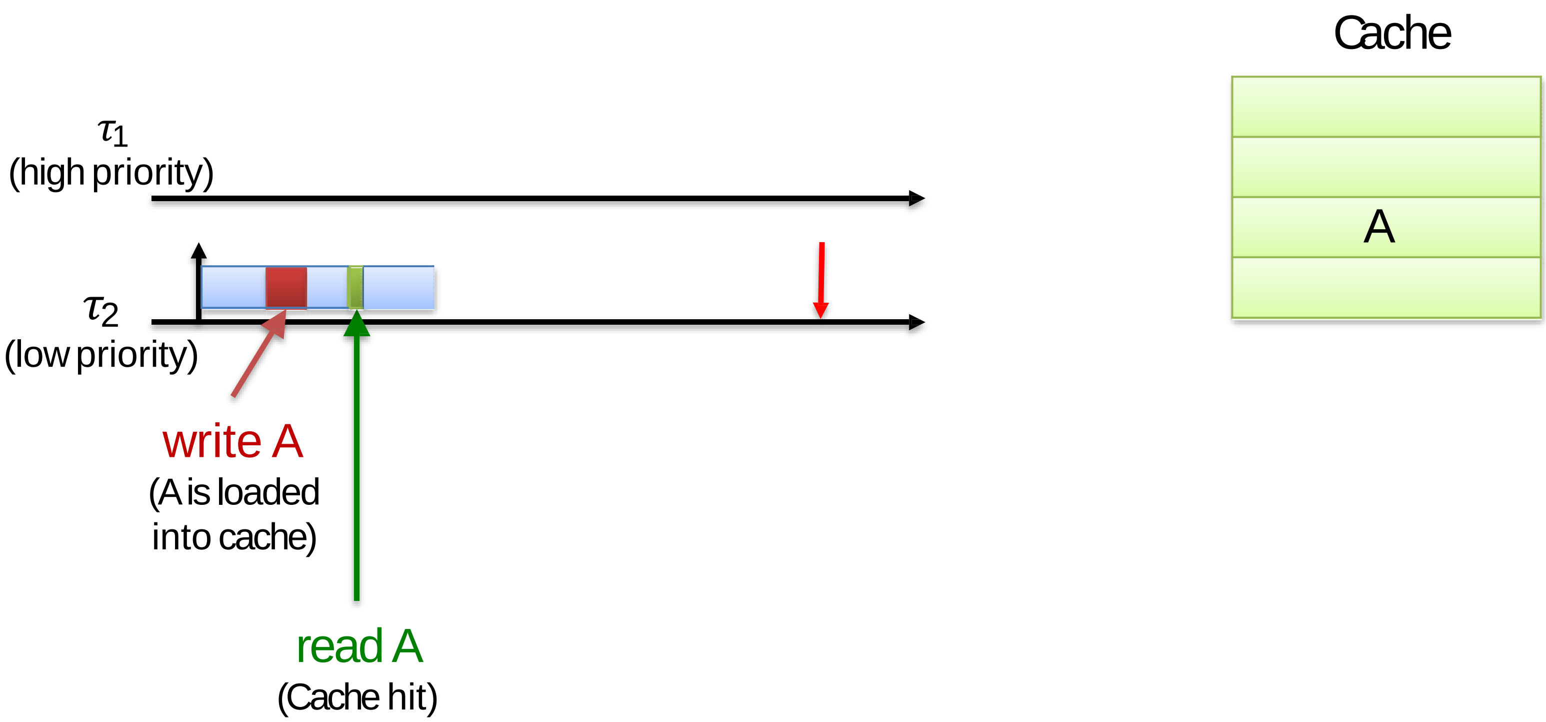
It is the time taken by the scheduler to suspend the running task, switch the context, and dispatch the new incoming task.



Disadvantages of preemptions

2- Cache-related preemption delay (CRPD)

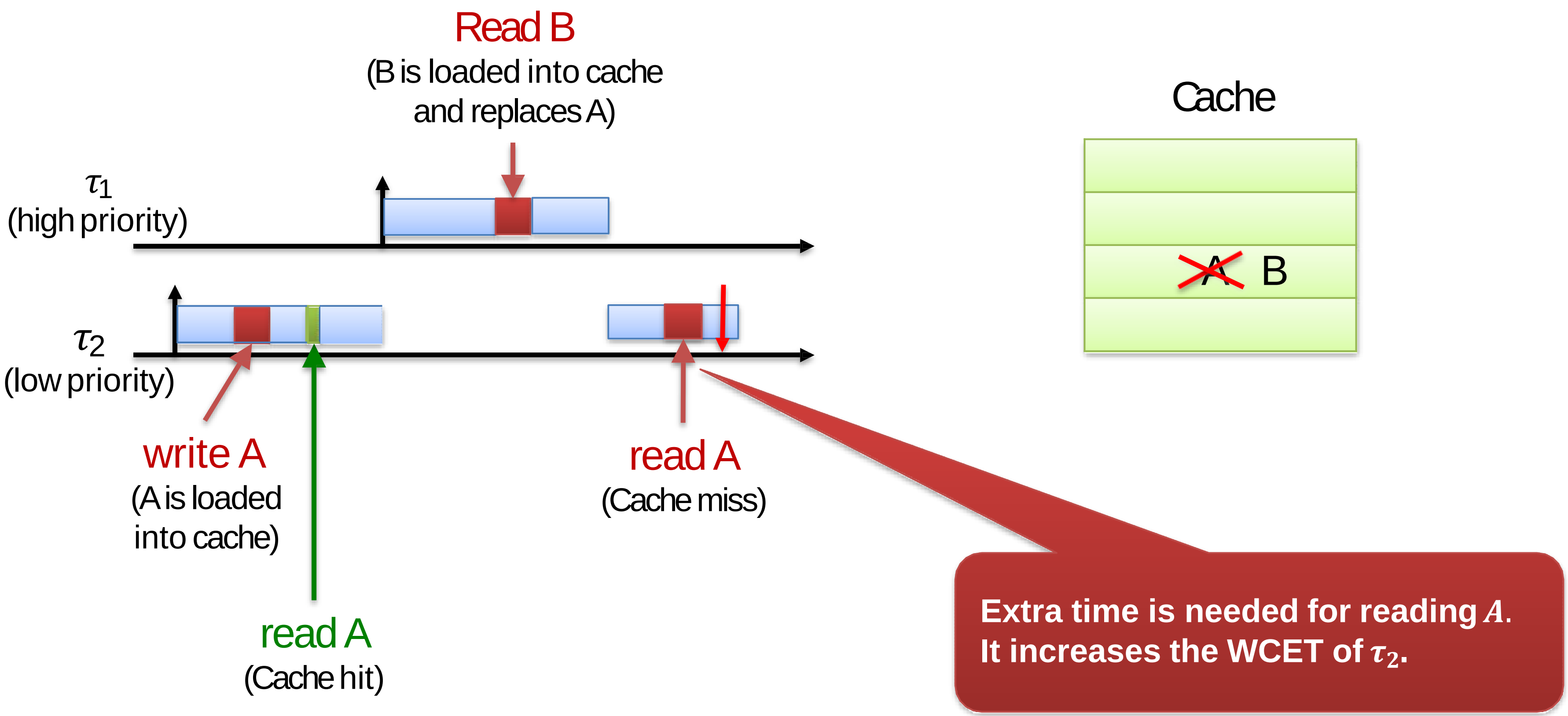
It is the delay introduced by high-priority tasks that evict cache lines containing data used in the future:



Disadvantages of preemptions

2- Cache-related preemption delay (CRPD)

It is the delay introduced by high-priority tasks that evict cache lines containing data used in the future:



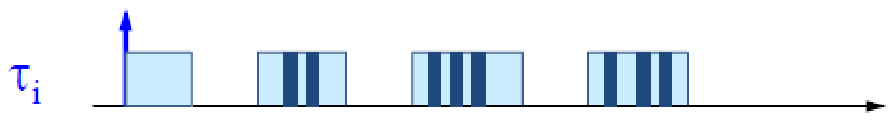
Disadvantages of preemptions

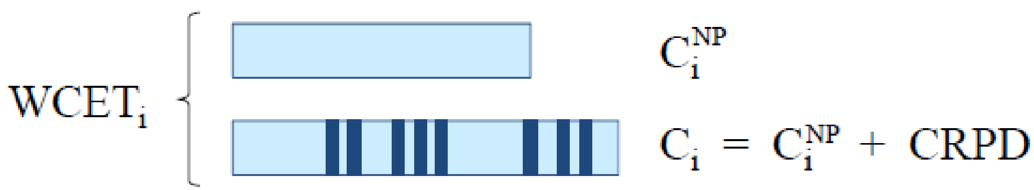
3- larger worst-case execution time

Preemptions cause CRPD which in turn increases the WCET of the task

The amount of CRPD depends on the number of preemptions a job suffers

Task experiencing preemptions by higher priority tasks:





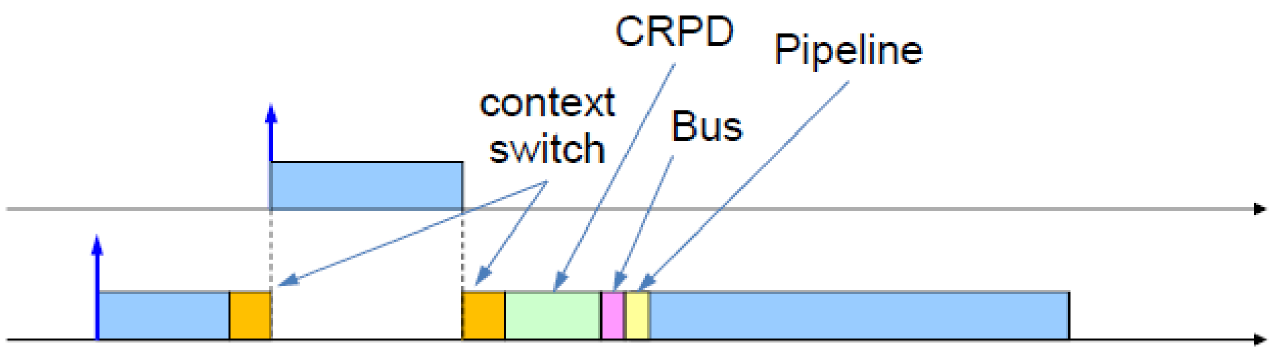
Disadvantages of preemptions

4 Pipeline cost

time to flush the pipeline when a task is interrupted and to refill it when task is resumed.

5 Bus cost

time spent waiting for the bus due to additional conflicts with I/O devices, caused by extra accesses to the RAM for the extra cache misses.

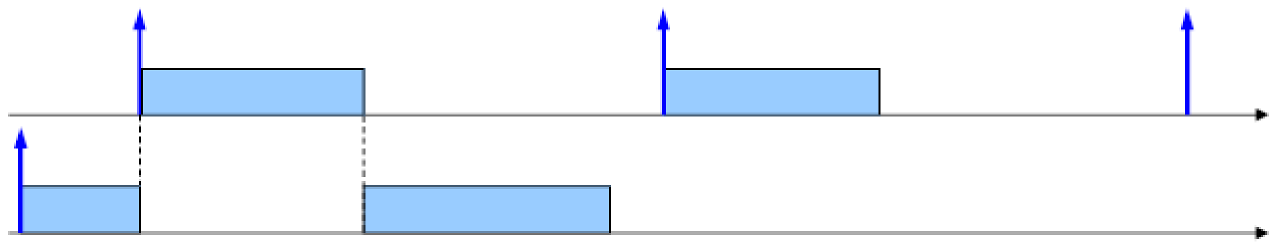


Disadvantages of preemptions

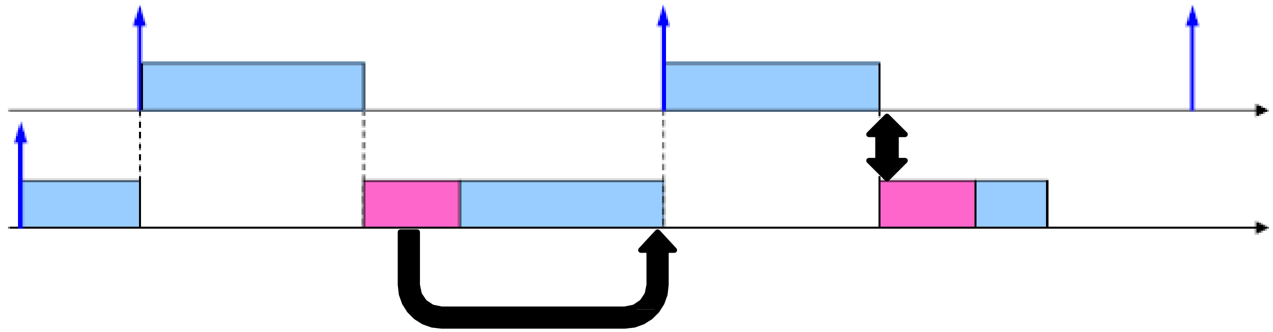
A preemption may cause more preemptions

the extra execution time also increases the number of preemptions:

Preemption overhead is zero



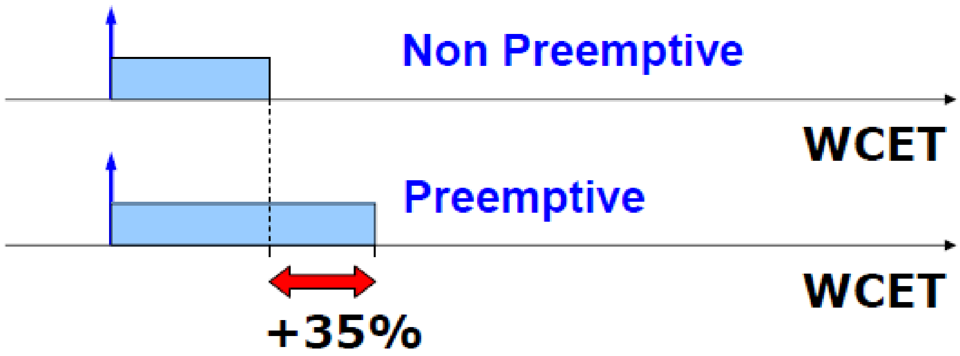
Preemption overhead is NOT zero



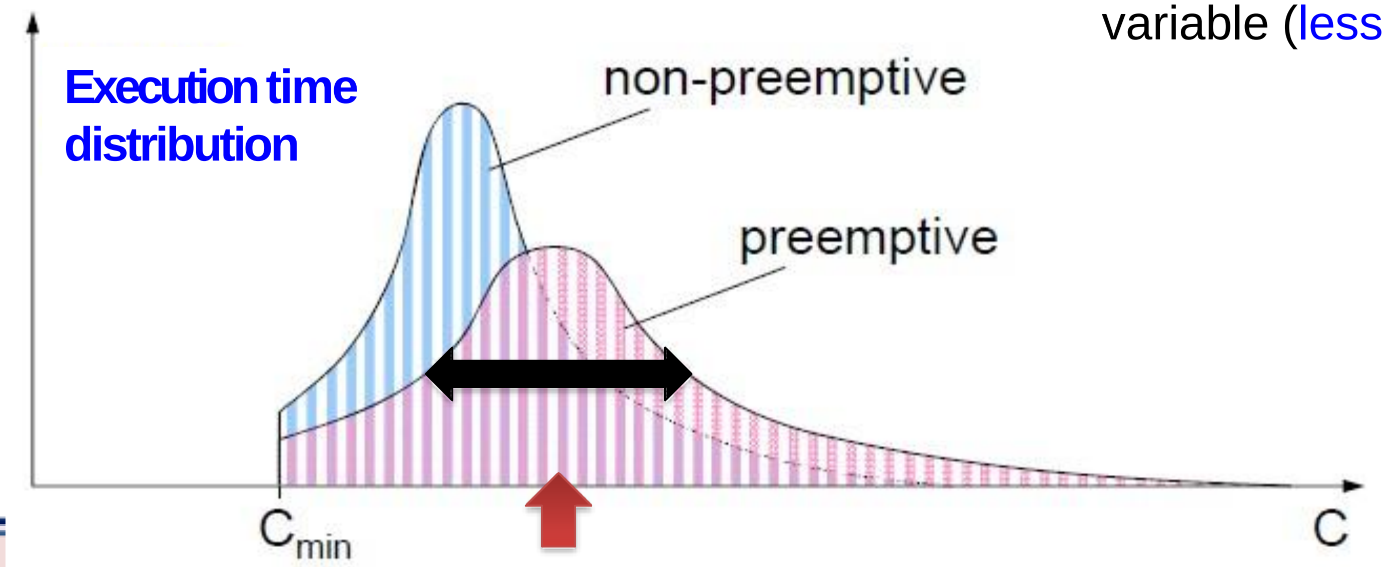
Disadvantages of preemptions

Preemption cost can be very large

- WCETs may increase up to 35% in the presence of preemptions (less efficiency)!



In preemptive execution, the execution times are more variable (less predictability)



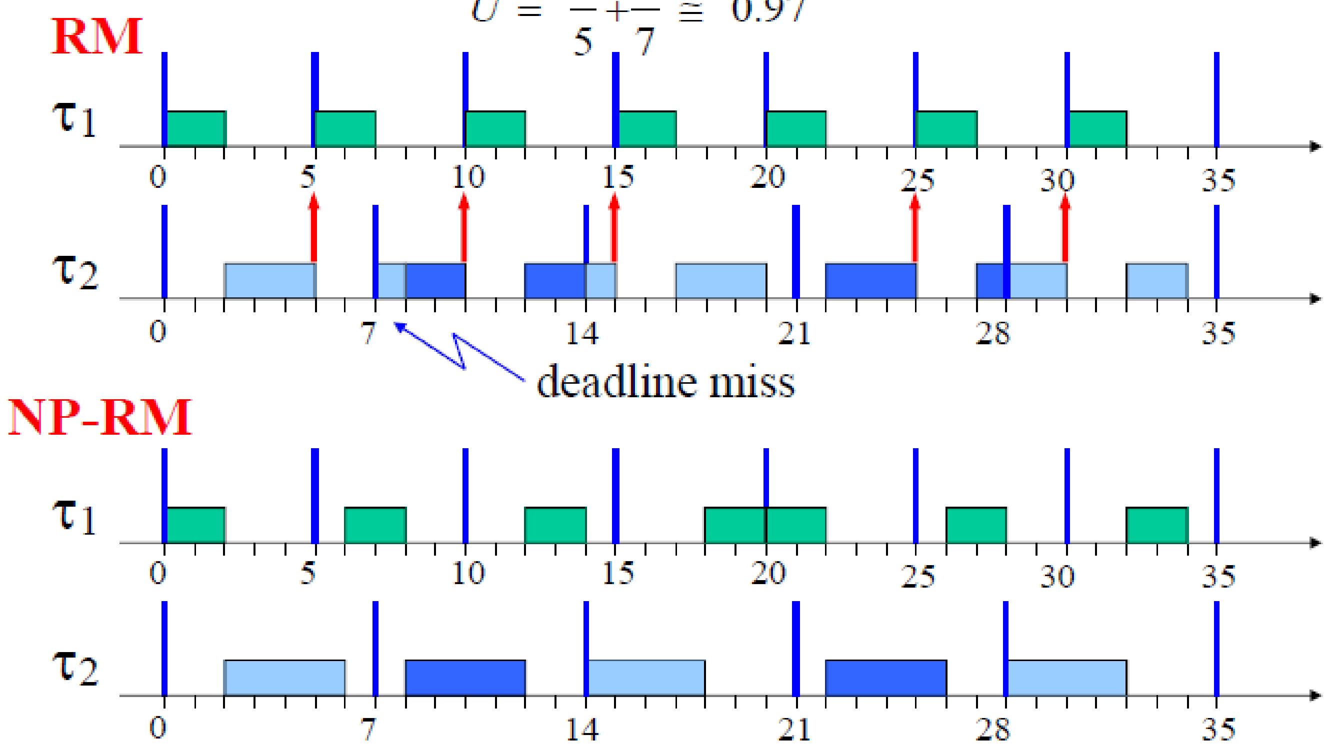
Advantages of NP scheduling

- It reduces **context-switch overhead**:
  - Making WCETs smaller and more predictable
- It simplifies the **access to shared resources**:
  - No semaphores are needed for critical sections
- It reduces **stack size**:
  - Task can share the same stack, since no more than one task can be in execution
- It allows achieving **small I/O Jitter**:
  - "finishing_time – start_time" has a low variation

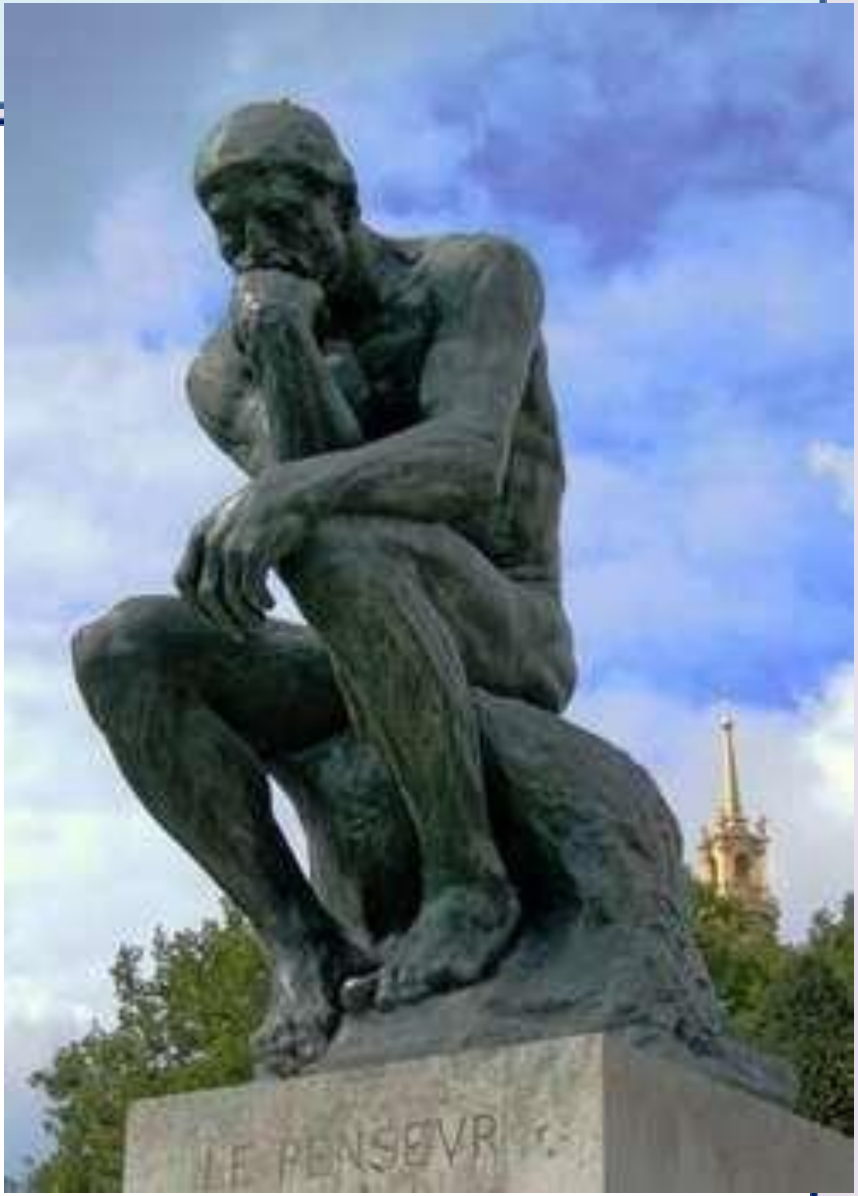
Advantages of NP scheduling

- In fixed-priority systems, non-preemptive execution can even improve schedulability (in some cases)

$$U = \frac{2}{5} + \frac{4}{7} \cong 0.97$$

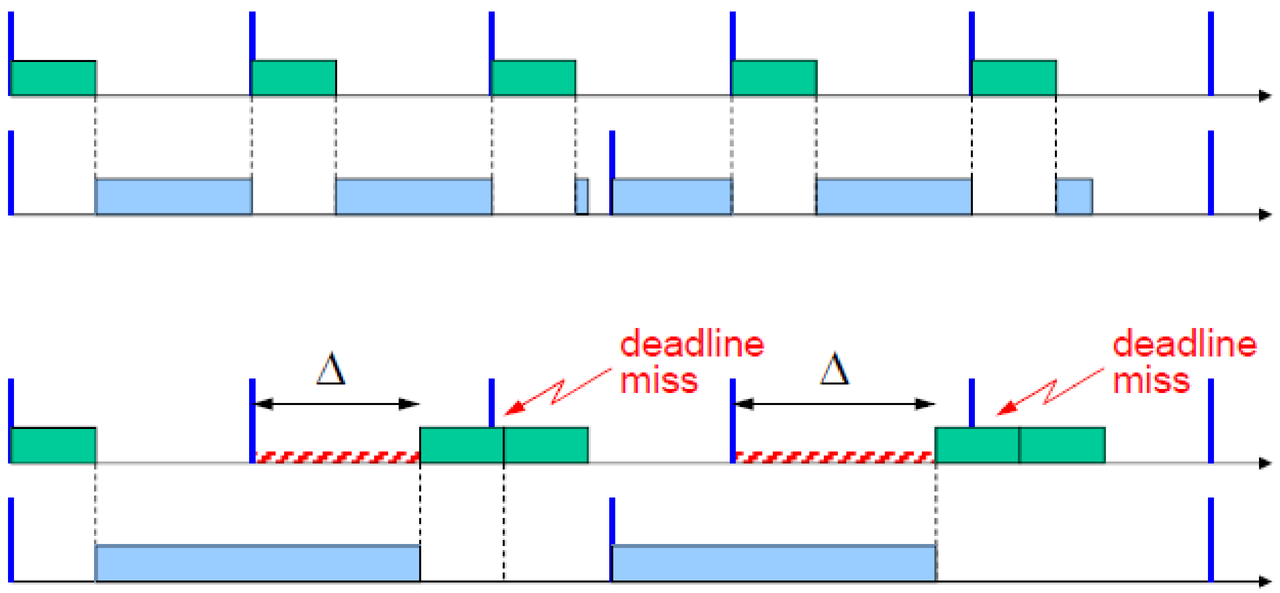


So, why preemptive execution?



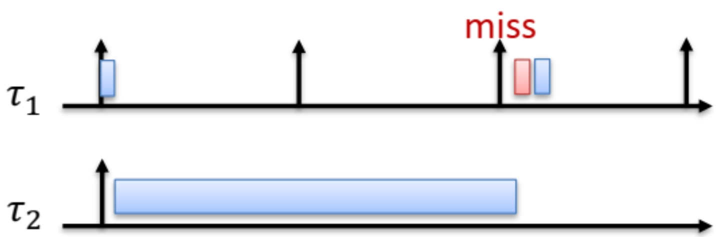
Disadvantages of NP scheduling

- In general, NP scheduling reduces schedulability because of introducing **blocking delays** on the high-priority tasks



Disadvantages of NP scheduling

- The utilization bound under non-preemptive scheduling drops to zero



$$C_1 = \epsilon \sim 0 \quad T_1 = 10 \Rightarrow \; U_1 = \frac{\epsilon}{T_1} \sim 0$$

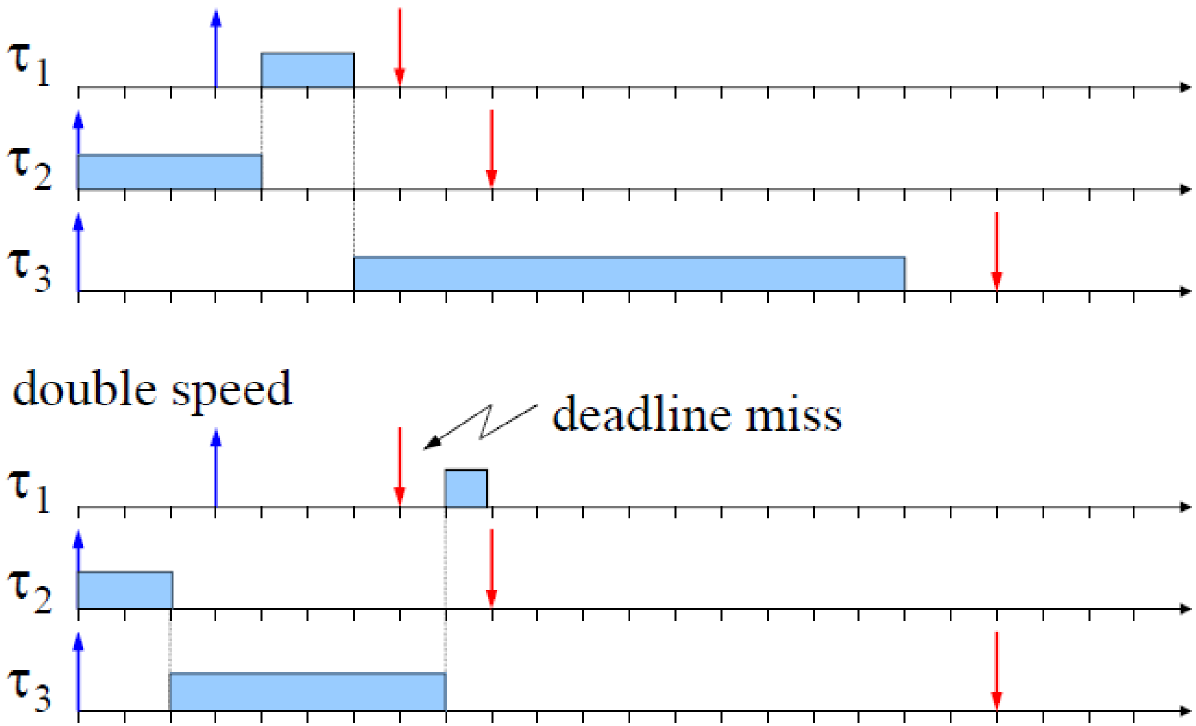
$$C_2 = 2T_1 \quad T_2 = \infty \; \Rightarrow \; U_2 = \frac{2T_1}{\infty} \sim 0$$

$$\boldsymbol{U_1 + U_2 \sim 0}$$

Infeasible! Despite having $U \sim 0$

Disadvantages of NP scheduling

Anomalies



Anomaly is a situation in which a deadline miss happens when we don't expect it to happen!

Example: the task set is feasible on the current processor but when we use a faster one, it becomes unschedulable!