Real-Time Systems

Lecture 6

Dynamic Priority Scheduling

Online scheduling with dynamic priorities:
Earliest Deadline First **scheduling– CPU utilization bound**
Optimality and comparison with RM:
Schedulability level, number of preemptions, jitter and response time
Other dynamic priority criteria
Least Slack First, First Come First Served

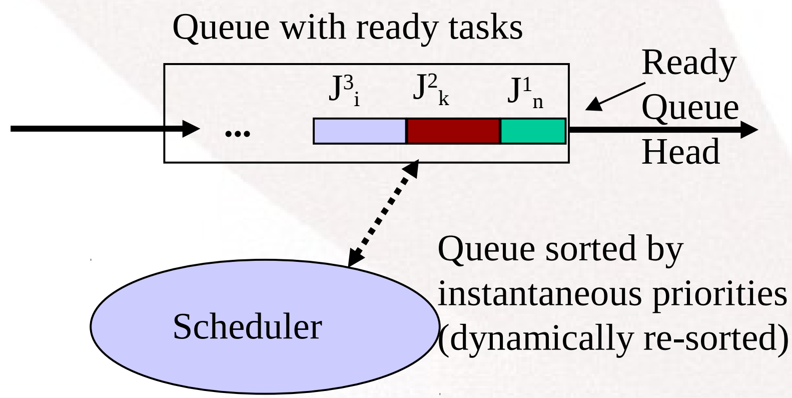
Last lecture (5)

- **On-line** scheduling with **fixed-priorities**
- The **Rate Monotonic** scheduling policy – schedulabilty analysis based on utilization
- The **Deadline Monotonic** and arbitrary deadlines scheduling policies
- Response-time analysis

On-line scheduling with dynamic priorities

- Scheduling is based on **dynamic criteria**, i.e. one that is known only at run-time
- The **dynamic parameter** used to sort the ready tasks can be understood as a **dynamic priority**
- The ready queue is sorted according with decreasing priorities whenever there is a priority change. Executes first the task that has the **greater instantaneous priority**

Complexity **O(n.log(n))**



On-line scheduling with dynamic priorities

Pros

- Scales well
  - Changes made to the task set are immediately seen by the scheduler
- Accommodates easily sporadic tasks

Cons

- Complex implementation
  - Requires a *kernel* supporting dynamic priorities
- *Higher overhead*
  - Re-sorting of ready queue; depends on the algorithm
- Imprevisibility on overloads
  - It is not possible to know *a priory* which tasks will fail deadlines

On-line scheduling with dynamic priorities

Priority allocation

- Inversely proportional to the time to the deadline
  - ***EDF – Earliest Deadline First***
    - Optimal among all dynamic priority criteria
- Inversely proportional to the *laxity* or *slack*
  - ***LSF (LST or LLF) – Least Slack First***
    - Optimal among all dynamic priority criteria
- Inversely proportional to the service waiting time
  - ***FCFS –First Come First Served***
    - Not optimal with respect to meet deadlines; extremely poor real-time performance
- etc.

On-line scheduling with dynamic priorities

Schedulability tests

- Since the schedule is built online it is important to determine ***a priori*** if a given task set **meets or not** its **temporal requirements**
- There are three types of schedulability tests:
  - Based on the **CPU utilization**
  - Based on the **CPU load** (*processor demand*)
  - Based on the **response time**

EDF Scheduling

EDF tests based on CPU utilization

(*n* independent tasks, with preemption)

- D=T

$$U(n)=\sum_{i=1}^{n}\frac{C_i}{T_i}\leq 1 \Leftrightarrow \text{Task set is schedulable}$$

  - **Allows using 100% of CPU with timeliness guarantees**

- D<T

$$\sum_{i=1}^{n}\frac{C_i}{D_i}\leq 1 \Rightarrow \text{Task set is schedulable}$$

  - **Pessimistic test**

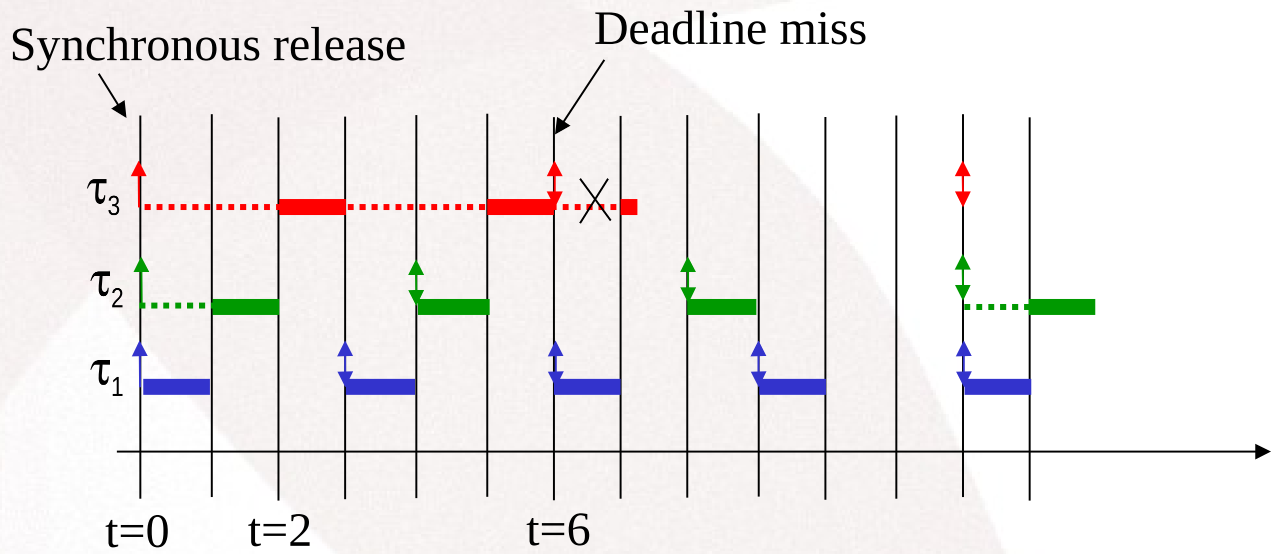
- D≤T

$$\sum_{i=1}^{n}\frac{C_i}{min(D_i,T_i)}\leq 1 \Rightarrow \text{Task set is schedulable}$$

RM Scheduling - example

Task set

τ_i	T_i	C_i
1	**3**	**1**
2	**4**	**1**
3	6	**2.1**



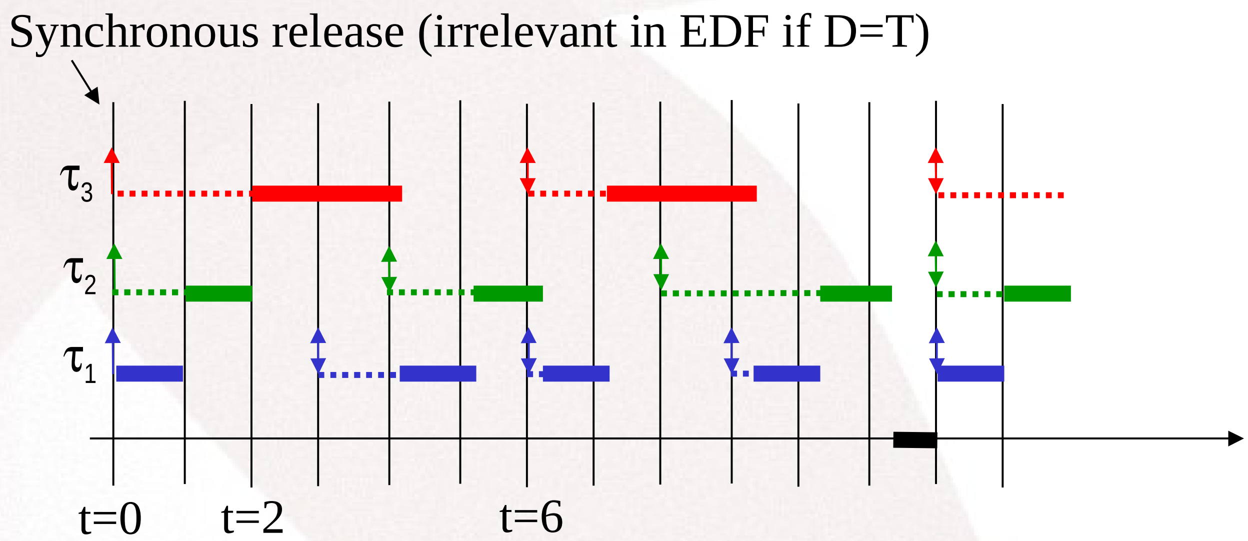
U = $1/3 + 1/4 + 2.1/6 =$ **0.93 > 0.78** $\Rightarrow$ **1 activation per period NOT guaranteed.**

τ_3 fails a deadline!

EDF Scheduling – same example

Task set

τ_i	T_i	C_i
1	**3**	**1**
2	**4**	**1**
3	6	**2.1**



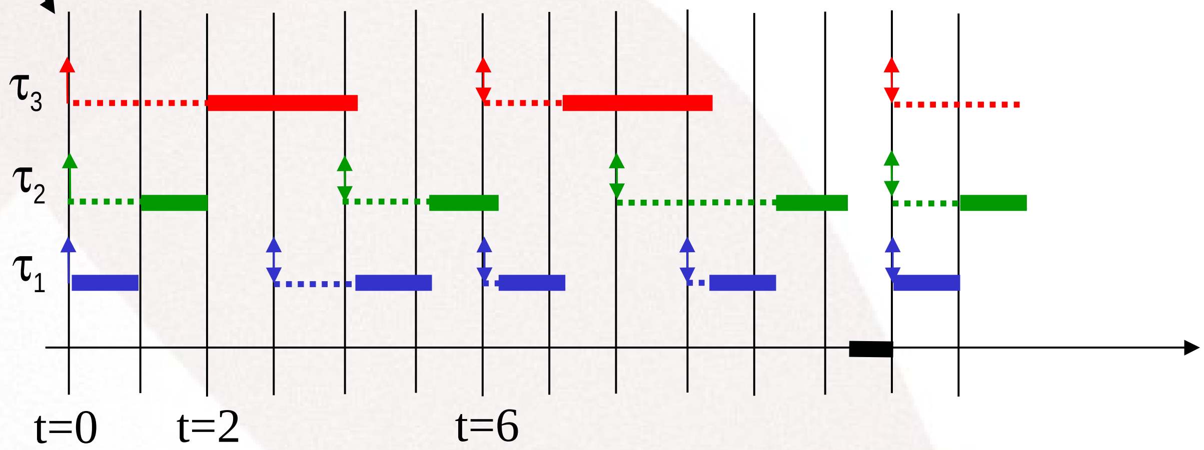
U = 1/3 + 1/4 + 2.1/6 = **0.93 ≤ 1 ⇔ 1 activation per period guaranteed**

EDF Scheduling – same example

Task set

τ_i	T_i	C_i
1	3	1
2	4	1
3	6	2.1

Synchronous release (irrelevant in EDF if D=T)



Note:

- No deadline misses
- Less preemptions
- Higher jitter on quicker tasks
- The worst-case response time does not coincide necessarily with the synchronous release

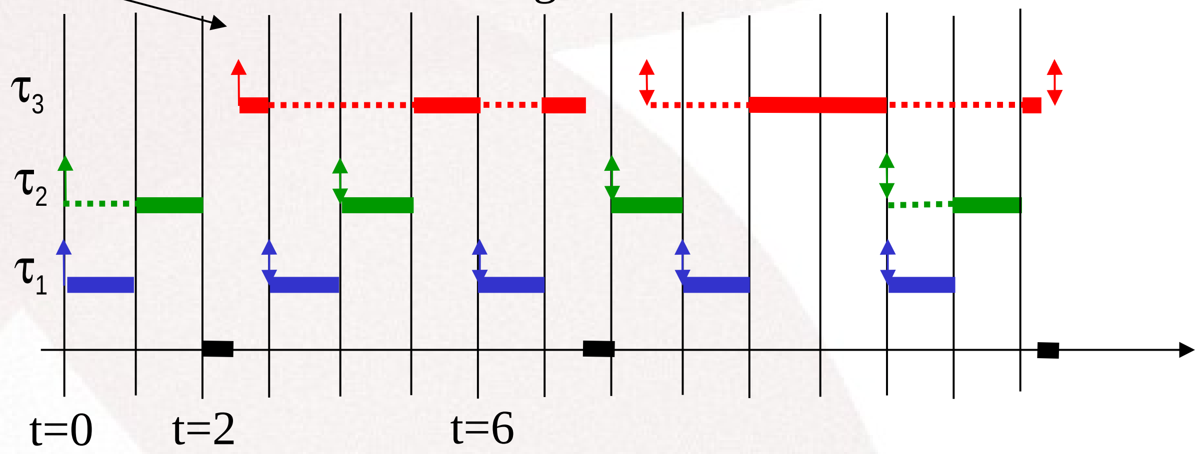
RM vs EDF scheduling – initial phases

Task set

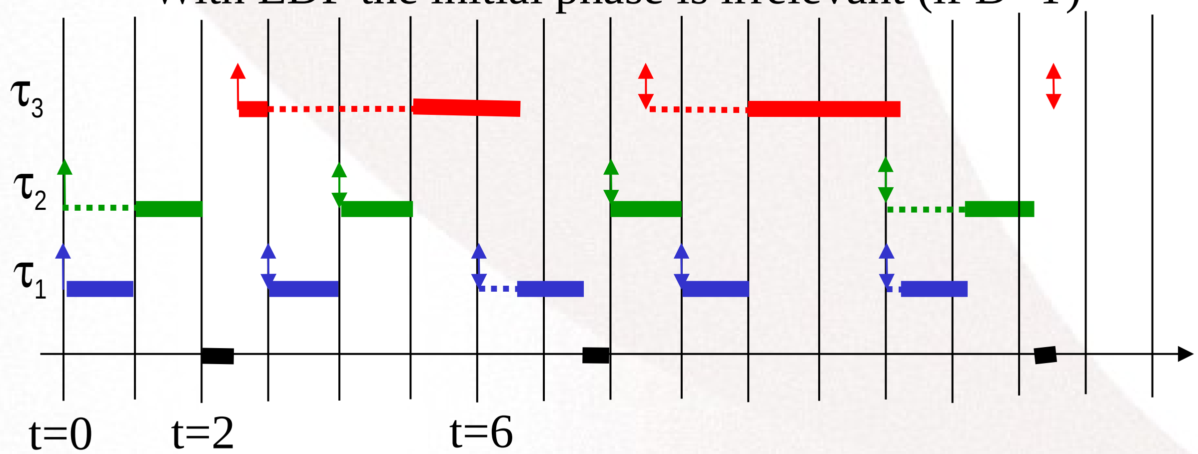
τ_i	T_i	C_i	O_i
1	3	1	0
2	4	1	0
3	6	2.1	2.5

Initial phase O_3

RM scheduling became feasible!



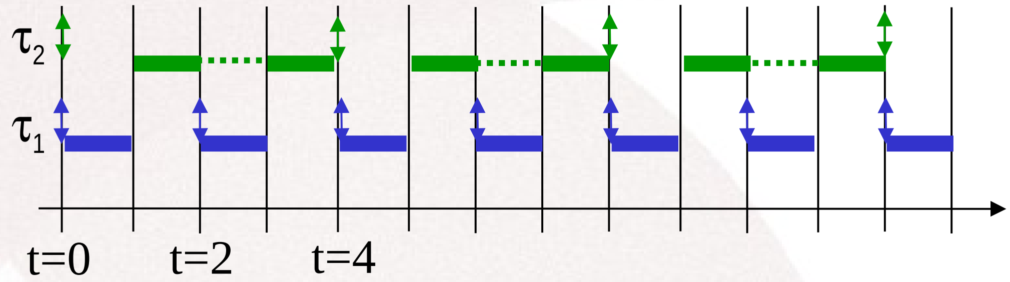
With EDF the initial phase is irrelevant (if D=T)



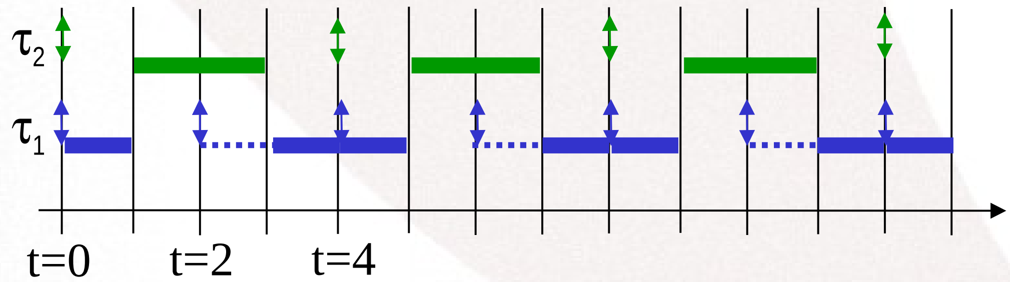
RM vs EDF scheduling – particular cases

U = 1/2 + 2/4 = **1**

RM



EDF



The actual resulting schedule depends on the criteria to break ties. Independently of the criteria, deadlines are met.

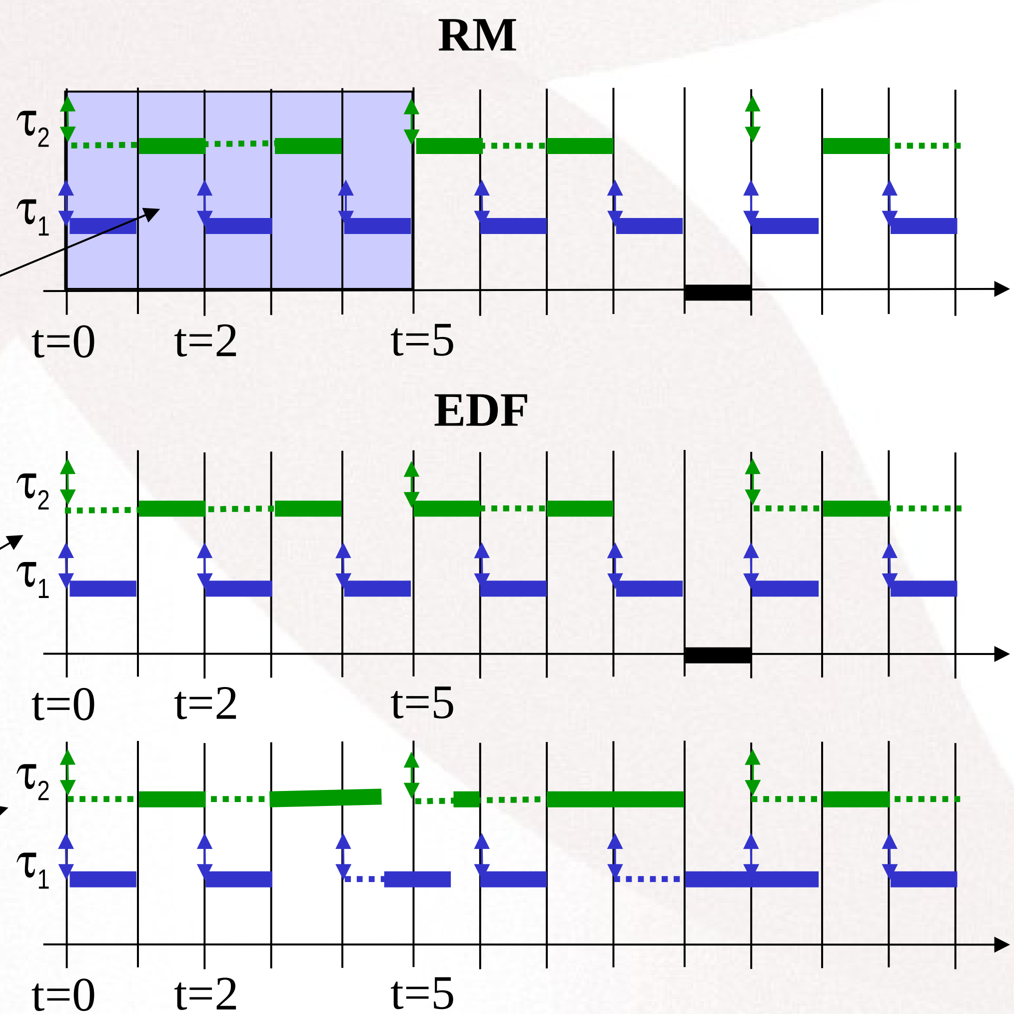
RM vs EDF scheduling – particular cases

U = 1/2 + 2/5 = **0.9**

C_1 or C_2 **cannot increase**, otherwise deadlines will be missed!

C_1 or C_2 **may increase** without causing deadlines misses, until U=1

C_2 = **2.5** ⇒ U=1



EDF Scheduling

Notion of fairness

- Be fair on the attribution of resources (e.g. CPU)
- EDF is intrinsically fairer than RM, in the sense that tasks see its relative deadline increased as the absolute deadline approaches, independently of its period or any other static parameter.

Consequences:

- *Deadlines are easier to met*
- As the deadlines approach preemptions are reduced
- The *slack of tasks that are quick but have large deadlines can be used by other task* (higher jitter *on tasks with shorter periods*)

CPU Load Analysis

- For D ≤ T, the bigger period during which the CPU is permanently used (i.e. without interruption, idle time) corresponds to the scenario in which all tasks are activated synchronously. This period is called ***synchronous busy period*** and has duration **L**

- L can be computed by the following iterative method, which returns the first instant since the synchronous activation in which the CPU completes all the submitted jobs

$$L(0)=\sum_i C_i$$

$$L(m+1)=\sum_i \left(\left\lceil \frac{L(m)}{T_i} \right\rceil * C_i\right)$$

CPU Load Analysis

Knowing L, we have to guarantee the load condition, i.e.

$$h(t) \leq t, \forall_{t \in [0,L]} \Rightarrow \text{Task ser is schedulable (synchronous activations)}$$

In which h(t) is the load function

$$h(t) = \sum_{i=1..n} max(0, 1 + \lfloor \frac{(t - D_i)}{T_i} \rfloor) * C_i$$

The computation of h(t) for $\forall_{t \in [0,L)}$ is unfeasible. However it is enough computing the load condition for the instants in which the load function varies, i.e.

$$S = U_i(S_i), S_i = \{m * T_i + D_i : m = 0, 1, ...\}$$

Note: there are other, possibly shorter, values for L

$$L = \frac{\sum_{i=1..n} (T_i - D_i) * U_i}{1 - U}, if\ D_i \leq T_i, \forall i$$

I. Ripoll, A. Crespo, and A.K. Mok. Improvement in Feasibility Testing for Real-TimeTasks. Journal of Real-Time Systems, 11 (1):19-39, 1996.

just another example

CPU Load Analysis

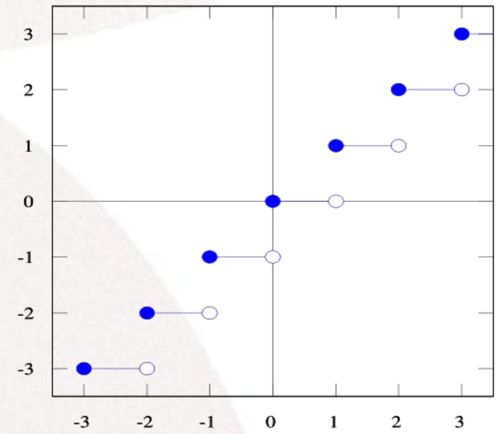
Illustration of h(t)

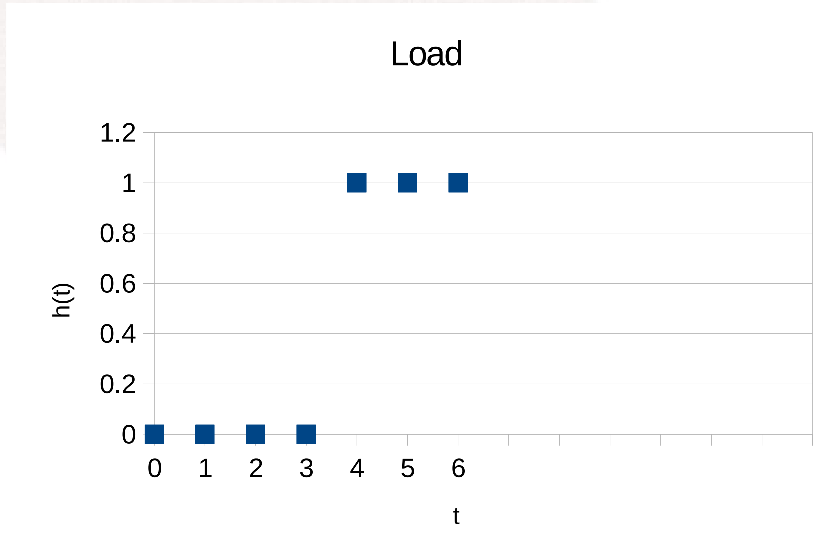
- Task with T=D=4; C=1

t	t-D	floor((t-D)/T)	h(t)
0	-4	-1	0
1	-3	-1	0
2	-2	-1	0
3	-1	-1	0
4	0	0	1
5	1	0	1
6	2	0	1

Floor function

http://upload.wikimedia.org/wikipedia/commons/thumb/e/e1/Floor_function.svg/500px-Floor_function.svg.png

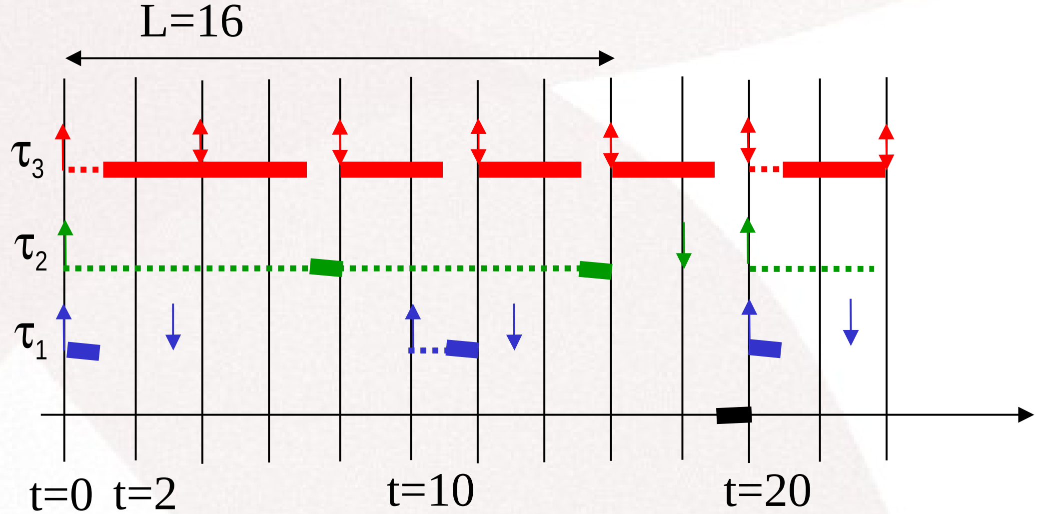




EDF Scheduling

Task set

τ_i	C_i	D_i	T_i
1	1	3	10
2	2	18	20
3	3	4	4



$$\sum_{i=1}^{n} \frac{C_i}{min(D_i, T_i)} = \frac{1}{3} + \frac{2}{18} + \frac{3}{4} = 1.194 > 1 \Rightarrow \text{Schedulability not guaranteed}$$

The CPU load analysis indicates that the task set is schedulable!

Response-time analysis

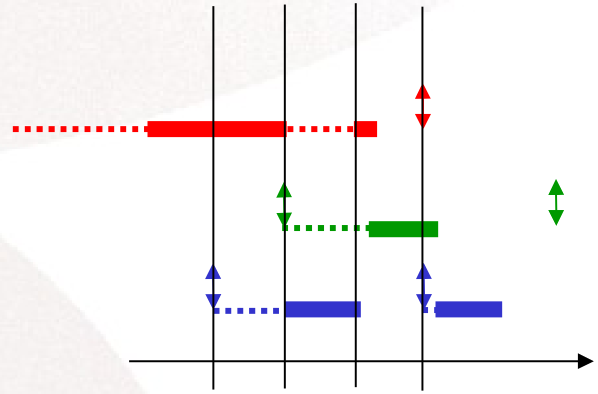
- With EDF, the **response time analysis** is more complex than with fixed priorities because we don't know *a priori* which instance suffers the maximum interference
- However, it is possible computing the worst-case response time using the notion of *busy period* relative to the deadline
- An upper bound to the response time can be easily obtained with the following expression, valid id U ≤ 1

$$\forall_i, Rwc_i \leq T_i * U$$

Note that thus upper bound is very pessimistic!

LSF Scheduling

LSF vs EDF short comparison

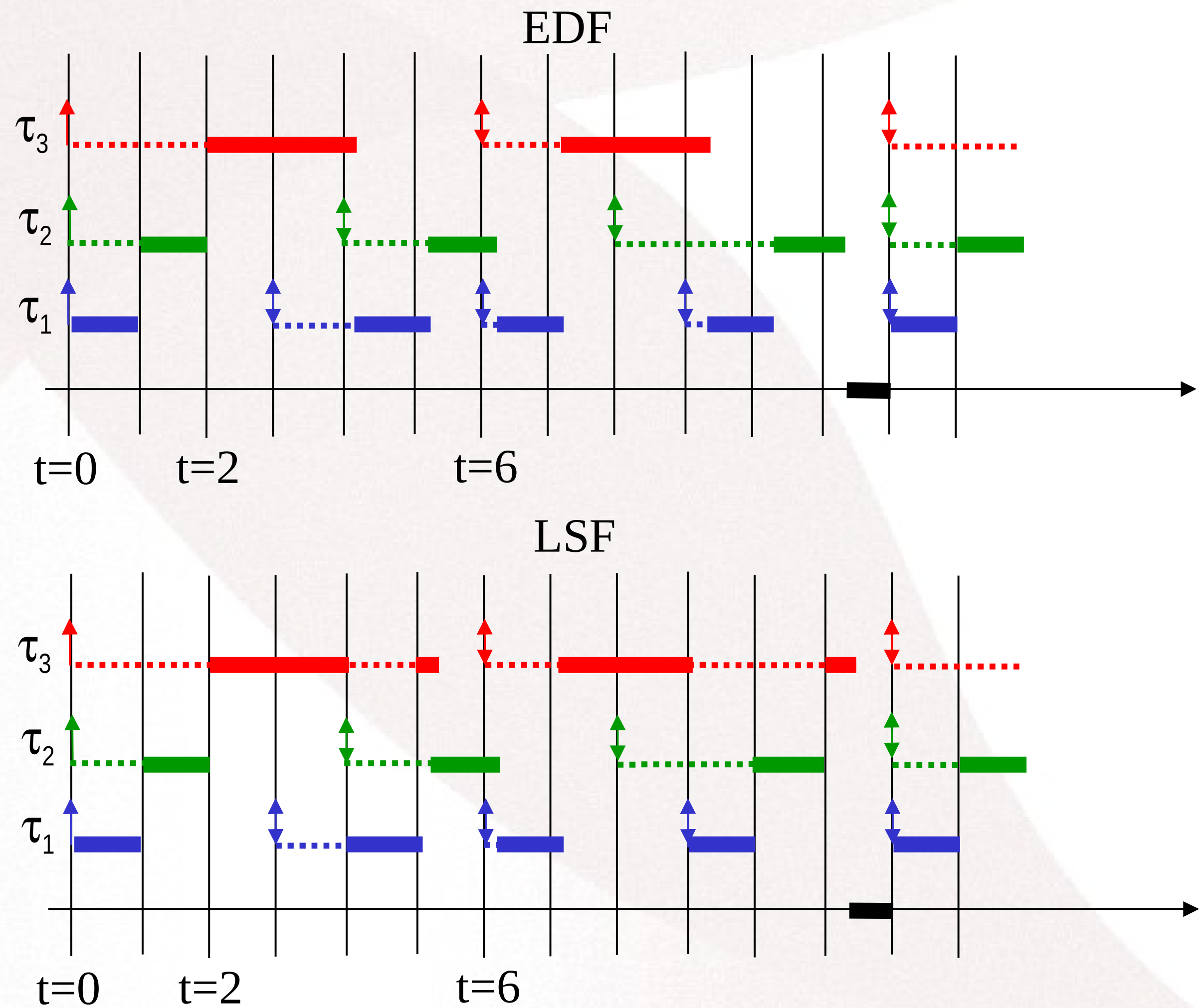


- LS is optimal (as EDF)
- As *slack* ↓ ⇒ Priority ↑
- Priority of ready tasks increases as time goes by
- Priority of the task in the running state does not change
  - On EDF the priorities of all tasks (ready and executing) increase equally as time goes by
- **Rescheduling** on instants where there are activations or terminations
- Causes and higher number of preemptions than EDF (and thus higher *overhead*)
- **No significant advantages with respect to EDF!**

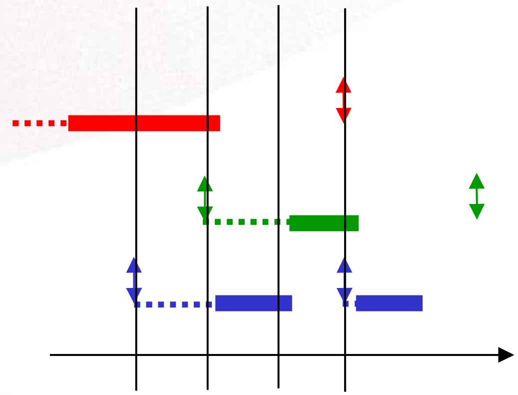
LSF scheduling– same example

Task set

τ_i	T_i	C_i
1	**3**	**1**
2	**4**	**1**
3	6	**2.1**



FCFS Scheduling



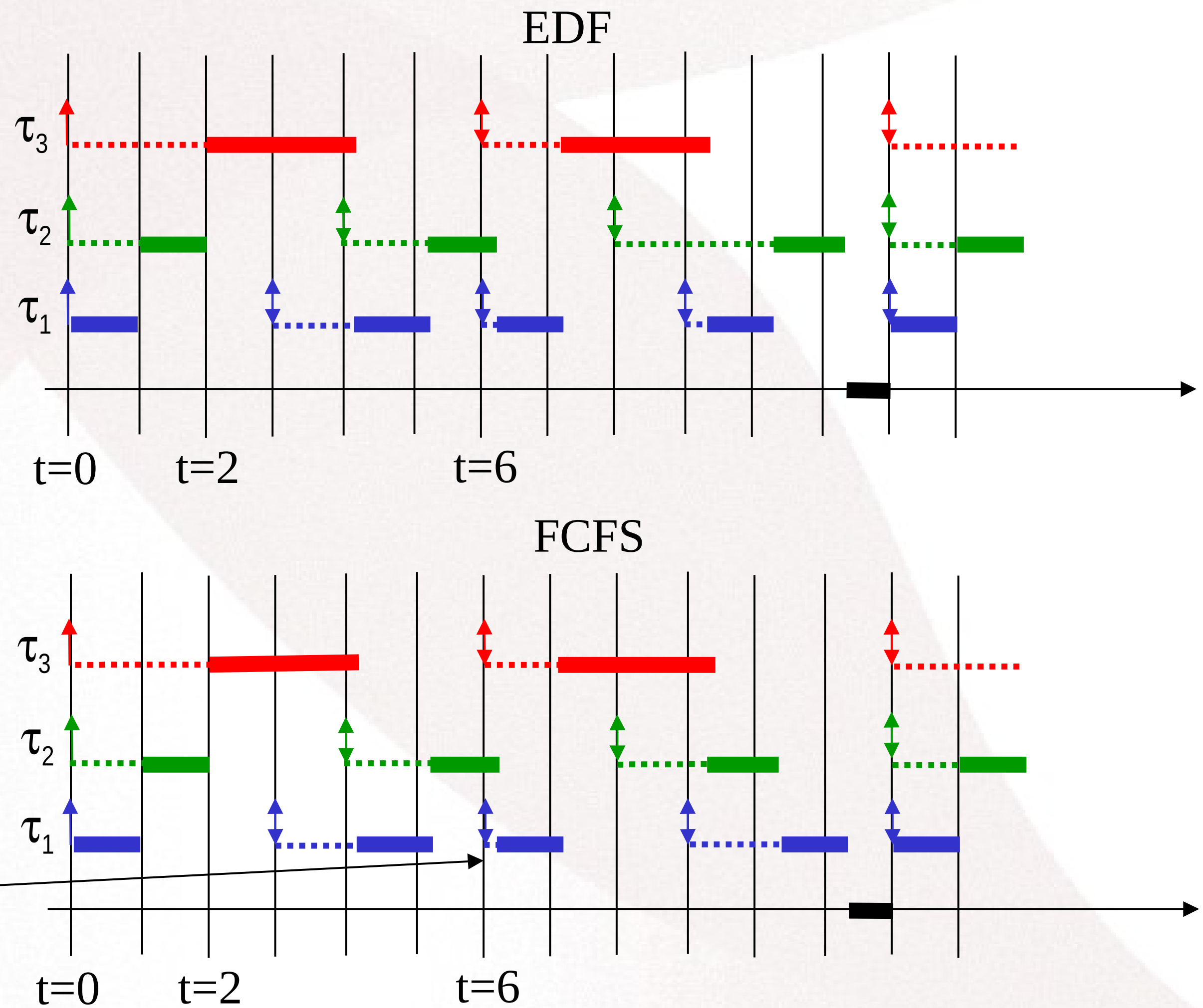
A brief comparison between FCFS and EDF/LLF

- Non optimal
  - May lead to deadline misses even with very low CPU utilization rates
- "Job age" ↑⇒Priority ↑
- Priority of the ready and running tasks increases as time goes by (an in EDF)
- New jobs always get the lower priority
- There are no preeemptions (smaller overhead and facilitates the implementation)
- **Very poor temporal behavior!**

FCFS – same example

Task set

τ_i	T_i	C_i
1	3	1
2	4	1
3	6	2.1



When the "age" is the same the tie break criteria is decisive!

Summary of lecture 6

- ***On-line*** **scheduling with dynamic priorities**
- **The *EDF - Earliest Deadline First* criteria: CPU utilization bound**
- **Optimality of EDF and comparison with RM:**
  - **Schedulability level, number of preemptions, jitter and response time**
- **Other dynamic priority criteria:**
  - ***LLF (LST) - Least Laxity (Slack) First***
  - ***FCFS - First Come First Served***