

# Sistemas de Tempo-Real

## Aula 6

### Escalonamento usando prioridades dinâmicas

Escalonamento *on-line* com prioridades dinâmicas

O critério *Earliest Deadline First* – limite de utilização de CPU

Optimalidade e comparação com RM:

nível de escalonabilidade, número de preempções,

*jitter* de disparo e tempo de resposta

Outros critérios de prioridades dinâmicas:

*Least Slack First*, *First Come First Served*

## Aula anterior (5)

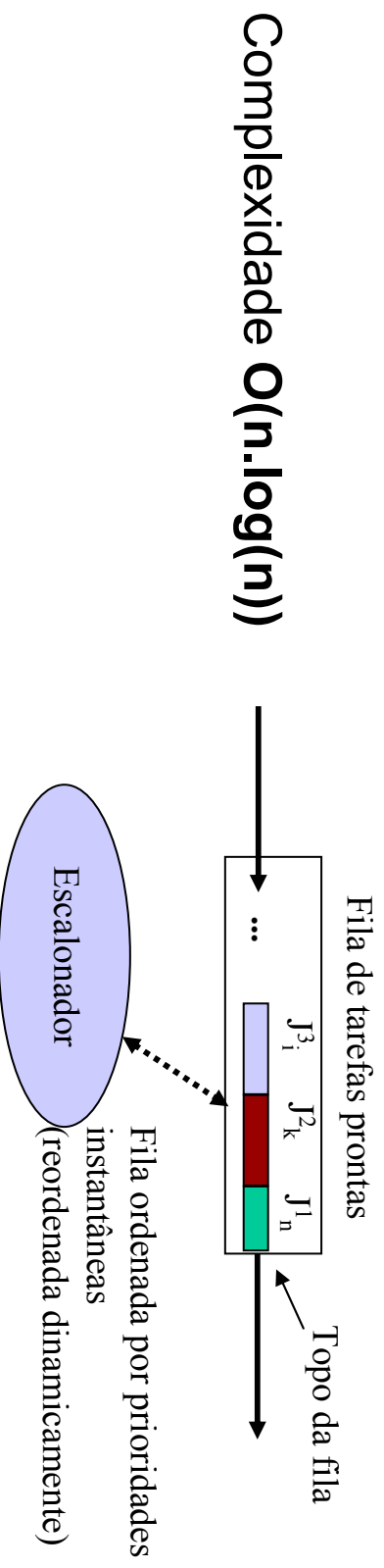
- Escalonamento **on-line** usando **prioridades fixas**
- O critério de escalonamento **Rate Monotonic** – análise de escalonabilidade baseada na **utilização**
- O critério **Deadline Monotonic** e de **prioridade fixa arbitrária**
- Análise usando **tempo de resposta** de pior caso.

# Escalonamento *on-line* com prioridades dinâmicas

O escalonamento é **construído** com o sistema **em funcionamento** (*on-line*) e baseia-se num **parâmetro dinâmico** (só conhecido em tempo de execução do escalonador).

O **parâmetro dinâmico** usado para orientar o escalonamento pode ser entendido como uma **prioridade dinâmica**

A fila de tarefas prontas a executar é ordenada por prioridades decrescentes sempre que há um alteração de prioridades relativas. Executa **primeiro** a que tem **maior prioridade instantânea**.



# Escalonamento on-line com prioridades dinâmicas

## A favor

- Facilmente escalável (alterações nas tarefas podem ser imediatamente tidas em conta pelo *scheduler*)
- Acomoda facilmente tarefas esporádicas

## Contra

- Implementação mais complexa (requer um kernel com prioridades dinâmicas)
- *Overhead* de execução mais elevado (reordenação dinâmica da fila das tarefas prontas – depende do algoritmo)
- Instabilidade face a sobrecargas (não é possível saber *a priori* quais os subconjuntos de tarefas que vão e que não vão cumprir a *deadline*)

# Escalonamento on-line com prioridades dinâmicas

## Atribuição de prioridades

- Inversamente proporcional ao tempo para a *deadline*  
(**EDF – Earliest Deadline First**)  
(ótimo em relação aos critérios de prioridades dinâmicas)
- Inversamente proporcional ao tempo livre (*laxity* ou *slack*)  
(**LSF (LST ou LLF) – Least Slack First**)  
(ótimo em relação aos critérios de prioridades dinâmicas)
- Inversamente proporcional ao tempo de espera por serviço  
(**FCFS – First Come First Served**)  
(não ótimo relativamente ao cumprimento de *deadlines*)

• ...

# Escalonamento on-line com prioridades dinâmicas

## Verificação de escalonabilidade

Como o escalonamento só é construído *on-line* pode ser importante saber *a priori* se um dado conjunto de tarefas cumpre ou não os seus requisitos temporais.

Existem três tipos principais de testes de escalonabilidade:

- Baseados na **taxa de utilização** do CPU
- Baseados na **carga imposta** ao CPU (*processor demand*)
- Baseados no **tempo de resposta**

# Escalonamento EDF

## Testes para EDF baseados na taxa de utilização

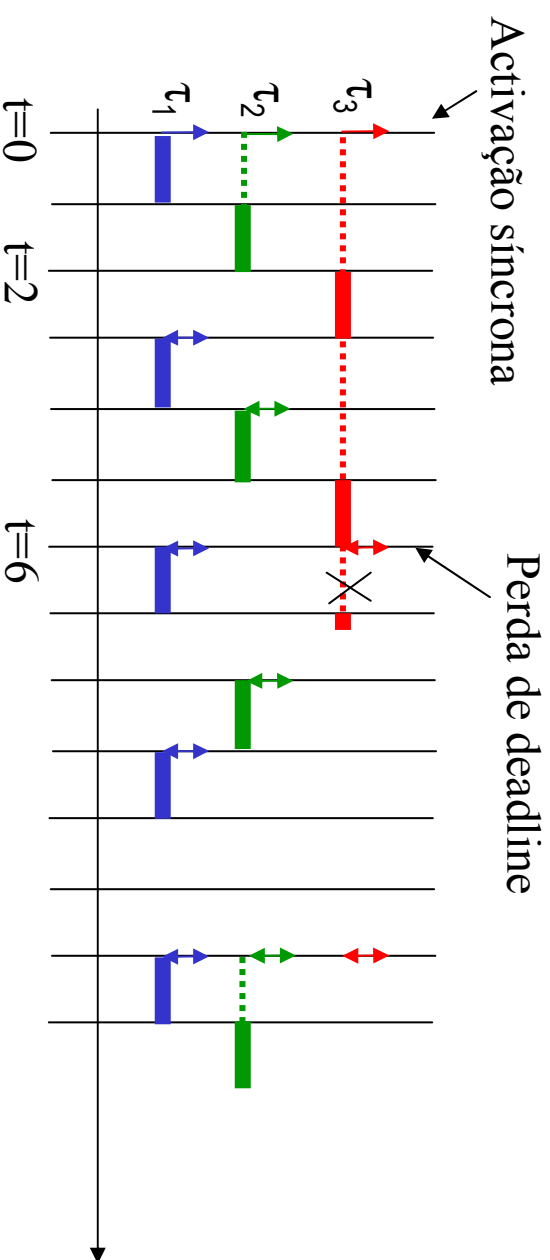
(com preempção e  $n$  tarefas independentes)

- $D=T$
- $U(n) = \sum_{i=1}^n (C_i/T_i) \leq 1 \Leftrightarrow$  conjunto escalonável
- Permite usar **100% do CPU** mantendo as **garantias temporais**  
 $D < T$
- $U'(n) = \sum_{i=1}^n (C_i/D_i) \leq 1 \Rightarrow$  conjunto escalonável
- $D$  arbitrária
- $\sum_{i=1}^n (C_i/\min(D_i, T_i)) \leq 1 \Leftrightarrow \Rightarrow$  conjunto escalonável

# Escalonamento RM – exemplo

Tabela de propriedades das tarefas

$\tau_i$	$T_i$	$C_i$
1	3	1
2	4	1
3	6	2.1



$U = 1/3 + 1/4 + 2.1/6 = 0.93 > 0.78 \Rightarrow$  1 activação por período **NÃO** garantida, com perda de deadline em  $\tau_3$

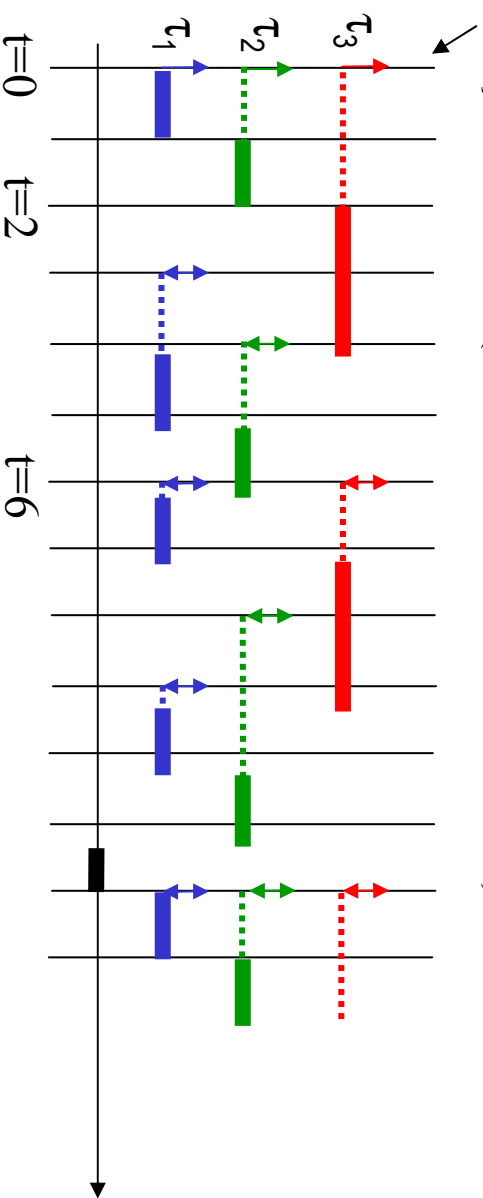


# Escalonamento EDF – mesmo exemplo

Tabela de propriedades das tarefas

$\tau_i$	$T_i$	$C_i$
1	3	1
2	4	1
3	6	2.1

Activação síncrona (irrelevante em EDF se  $D=T$ )

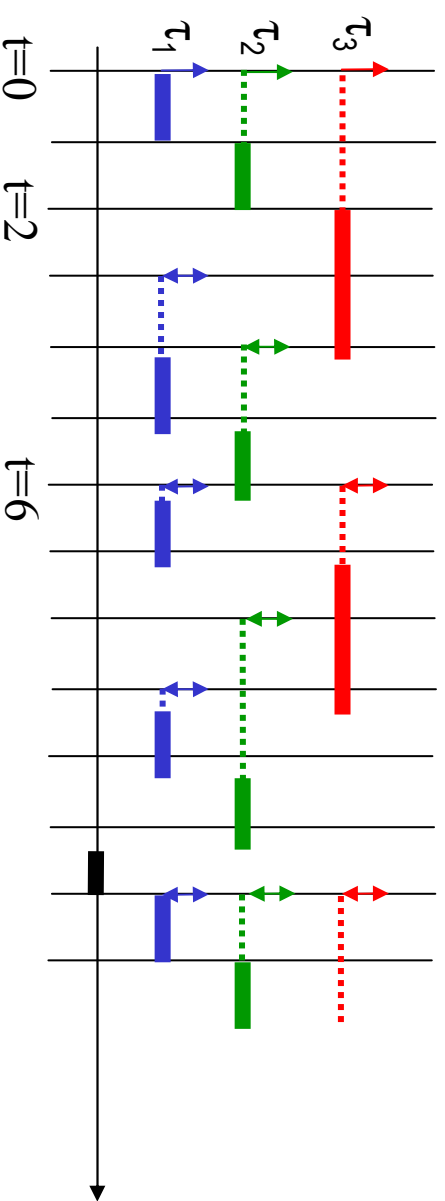


$$U = 1/3 + 1/4 + 2.1/6 = 0.93 \leq 1 \Leftrightarrow 1 \text{ activação por período garantida}$$

# Escalonamento EDF – mesmo exemplo

Tabela de propriedades das tarefas

$\tau_i$	$T_i$	$C_i$
1	3	1
2	4	1
3	6	2.1



**Notar:**

- Não há perdas de deadline
- Menos preempções
- *Jitter* nas tarefas rápidas
- A resposta de pior caso não é necessariamente na activação síncrona

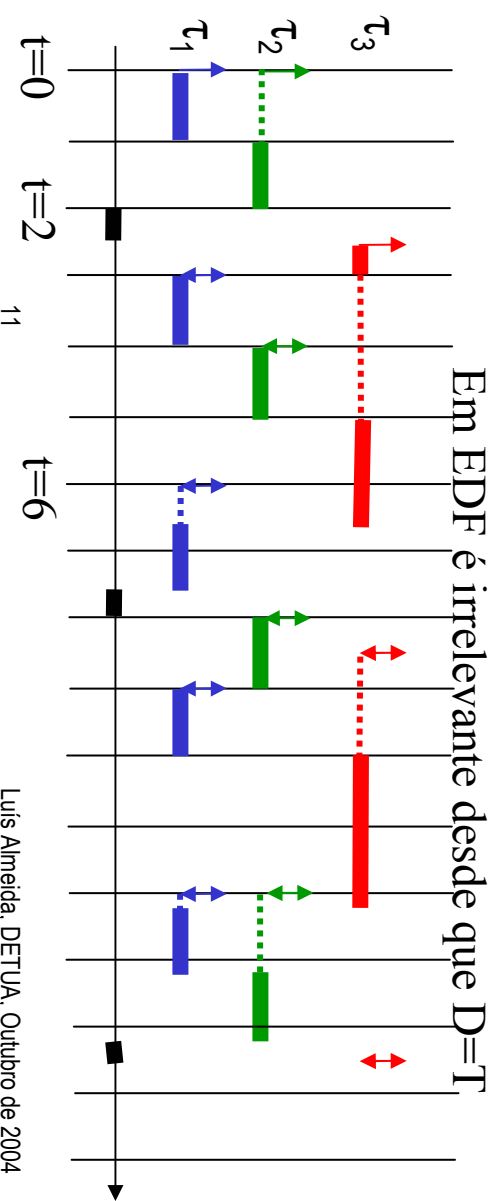
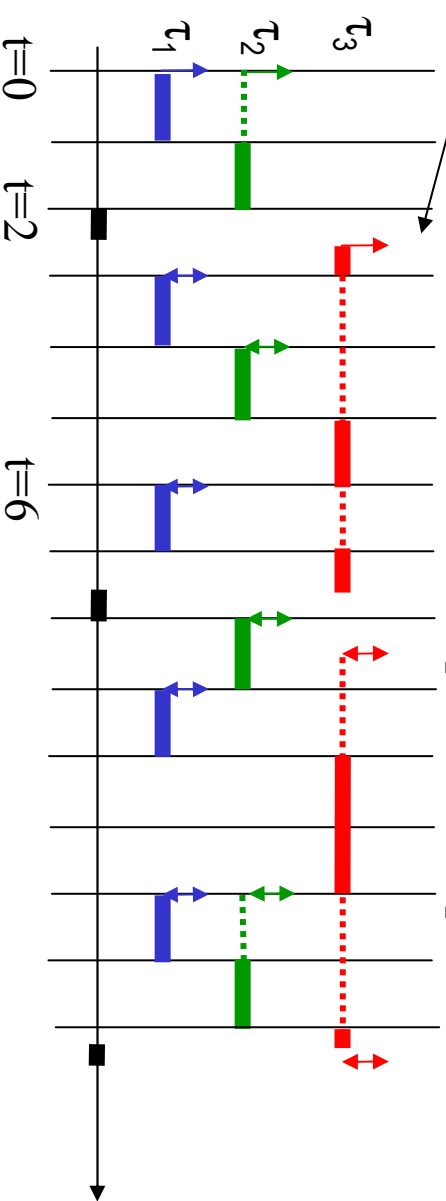
# Escalonamento RM vs EDF – fases iniciais

Fase inicial  $O_3$

Escalonamento RM passou a ser praticável !

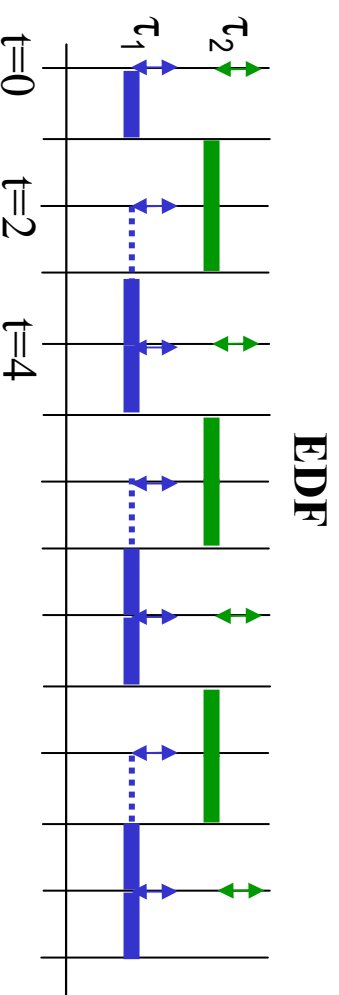
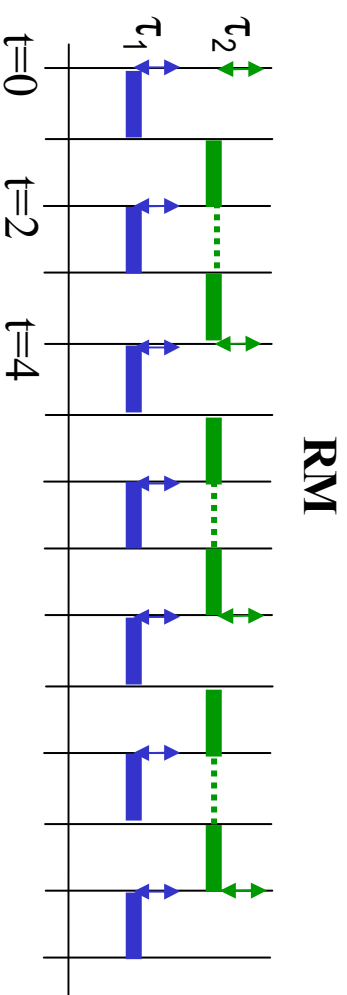
Tabela de propriedades das tarefas

$\tau_i$	$T_i$	$C_i$	$O_i$
1	3	1	0
2	4	1	0
3	6	2.1	2.5



# Escalonamento RM vs EDF – casos particulares

$$U = 1/2 + 2/4 = \mathbf{1}$$



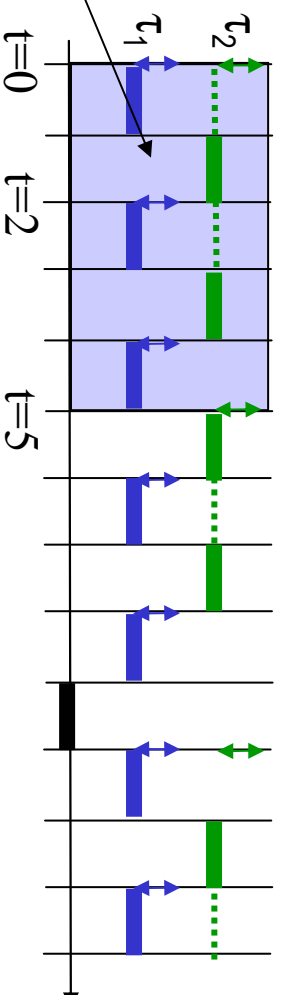
O escalonamento efectivo depende do critério de desempate mas as deadlines são cumpridas de qualquer modo

# Escalonamento RM vs EDF – casos particulares

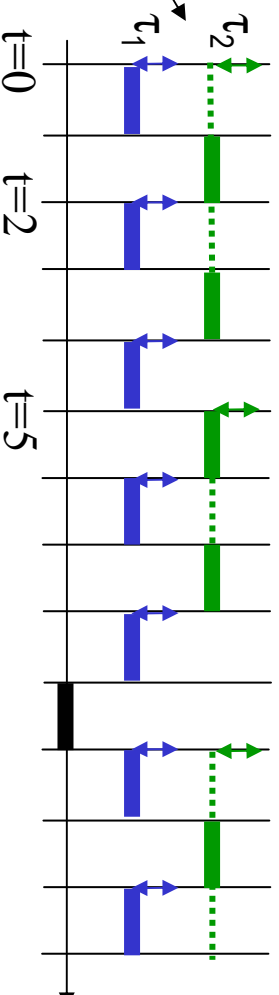
$$U = 1/2 + 2/5 = \mathbf{0.9}$$

$C_1$  ou  $C_2$  **não** podem ser aumentados sem causar perda de deadlines

RM

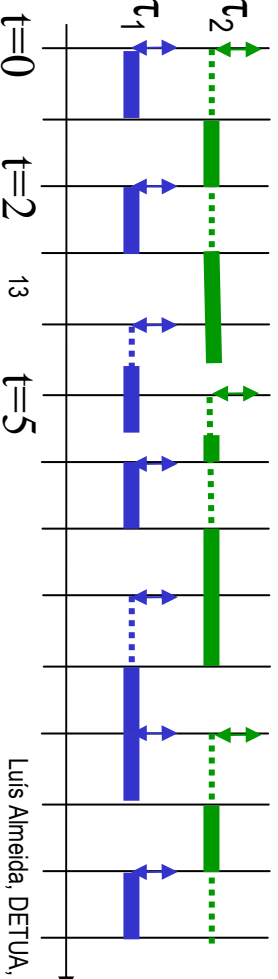


EDF



$C_1$  ou  $C_2$  **podem** ser aumentados sem causar perda de deadlines até  $U=1$

$$C_2 = \mathbf{2.5} \Rightarrow U=1$$



# Escalonamento EDF

## Noção de fairness

Justeza na distribuição de um recurso (e.g. CPU)

EDF é intrinsecamente mais justo (*fair*) que RM no sentido de que as tarefas vêm a sua prioridade elevada à medida que se aproximam da deadline, independentemente do seu período ou de outro parâmetro estático.

## Consequências:

- Facilita-se o cumprimento das deadlines
- Evitam-se as preempções quando as tarefas se aproximam da deadline
- Usa-se o tempo disponível (*slack*) das tarefas de ritmo mais rápido mas cuja deadline é mais tardia (maior *jitter* nas tarefas de ritmo mais rápido)

## Análise da carga imposta ao CPU

Para  $D \leq T$ , o período de maior carga consecutiva de CPU (i.e. sem interrupção, tempo morto) corresponde à situação em que todas as tarefas são activadas sincronamente. Esse período chama-se *synchronous busy period* e tem duração  $L$

$L$  calcula-se pelo método iterativo seguinte, que nos devolve o primeiro instante desde a activação síncrona em que o CPU executa todas as instâncias que lhe são submetidas

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

$$L(m+1) = \sum_i \lceil L(m)/T_i \rceil * C_i$$

## Análise da carga imposta ao CPU

Sabendo  $L$ , temos de garantir a condição de carga, i.e.

$h(t) \leq t \quad \forall t \in [0, L) \Leftrightarrow$  conjunto escalonável (ativações síncronas)  
em que  $h(t)$  é a função de carga

$$h(t) = \sum_{D_i \leq t} (1 + \lfloor (t - D_i) / T_i \rfloor) * C_i$$

O cálculo de  $h(t)$  para  $\forall t \in [0, L)$  é impraticável... Mas basta verificar a condição de carga para os pontos em que a função de carga varia, i.e.

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

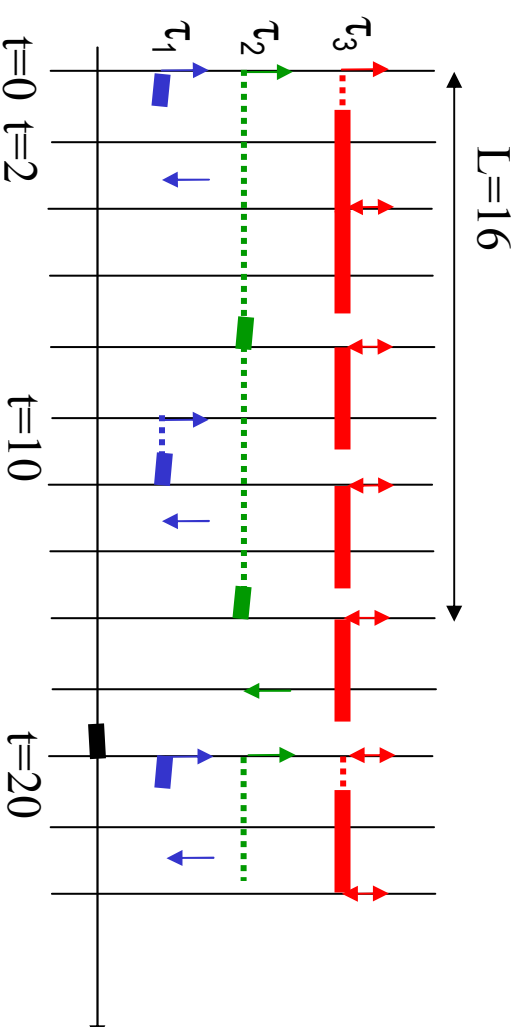
Nota: existem outros limites possivelmente mais curtos do que  $L$



# Escalonamento EDF

Tabela de propriedades das tarefas

$\tau_i$	$C_i$	$D_i$	$T_i$
1	1	3	10
2	2	18	20
3	3	4	4



$\sum_{i=1}^n (C_i / \min(D_i, T_i)) = 1/3 + 2/18 + 3/4 = \mathbf{1.194} > \mathbf{1} \Rightarrow$  escalonabilidade **não** garantida  
mas praticável !

A análise de carga imposta ao CPU diz que o conjunto é escalonável

## Análise do tempo de resposta

Em EDF, a análise do **tempo de resposta** é mais complexa que em prioridades fixas pois não sabemos a priori qual instância sofre a máxima interferência.

Contudo é possível determinar o tempo de resposta de pior caso recorrendo também à noção de busy period mas relativo à deadline.

Um majorante do tempo de resposta pode ser obtido muito facilmente com a seguinte expressão, desde que  $U \leq 1$

$$A_i, R_{WC_i} \leq T_i^* U$$

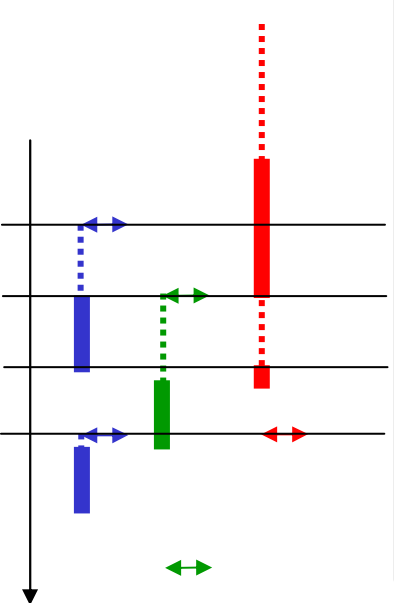
Notar que este majorante é substancialmente pessimista!

# Escalonamento LSF

## Algumas propriedades de LSF vs EDF

Ótimo (tal como EDF)

“Tempo livre” (slack) ↓ ⇒ Prioridade ↑



Prioridade das tarefas prontas aumenta à medida que o tempo passa

Prioridade da tarefa em execução mantém-se constante

(em EDF, as prioridades de todas as tarefas prontas e em execução aumentam de igual modo à medida que o tempo passa)

Reescalonamento nos pontos em que há ativações ou terminações

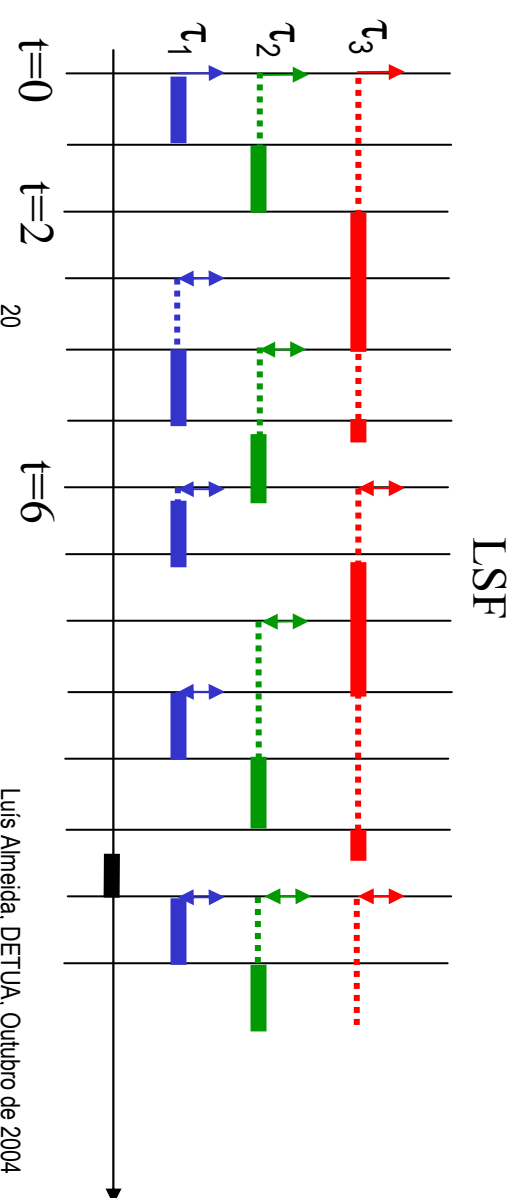
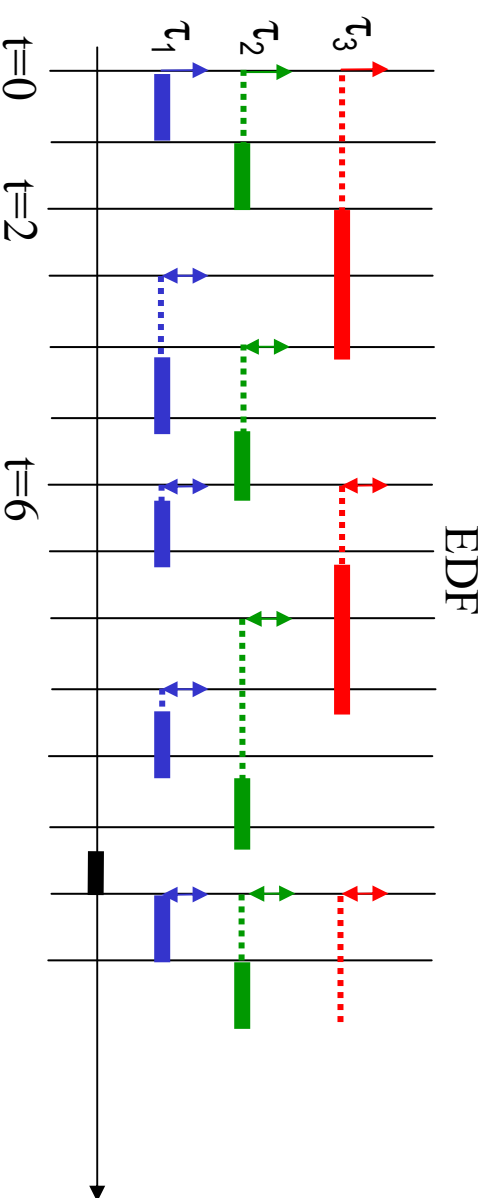
Causa maior número de preempções do que EDF (maior overhead)

**Não apresenta vantagens face a EDF !**

# Escalonamento LSF – mesmo exemplo

Tabela de propriedades das tarefas

$\tau_i$	$T_i$	$C_i$
1	3	1
2	4	1
3	6	2.1



# Escalonamento FCFS

## Alguas propriedades de FCFS vs EDF/LLF

Não óptimo (causa perda de deadlines com facilidade)

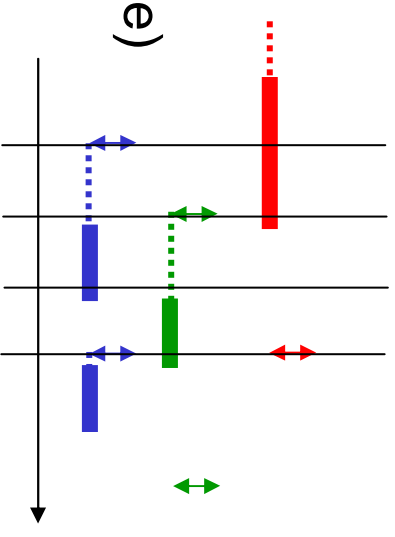
“Idade da instância”  $\uparrow \Rightarrow$  Prioridade  $\uparrow$

Prioridade das tarefas prontas e em execução aumenta à medida que o tempo passa (tal como EDF)

Quando chega uma instância nova é-lhe sempre atribuída a menor prioridade

Não causa preempções (menor overhead – fácil implementação)

**Comportamento temporal pobre !**

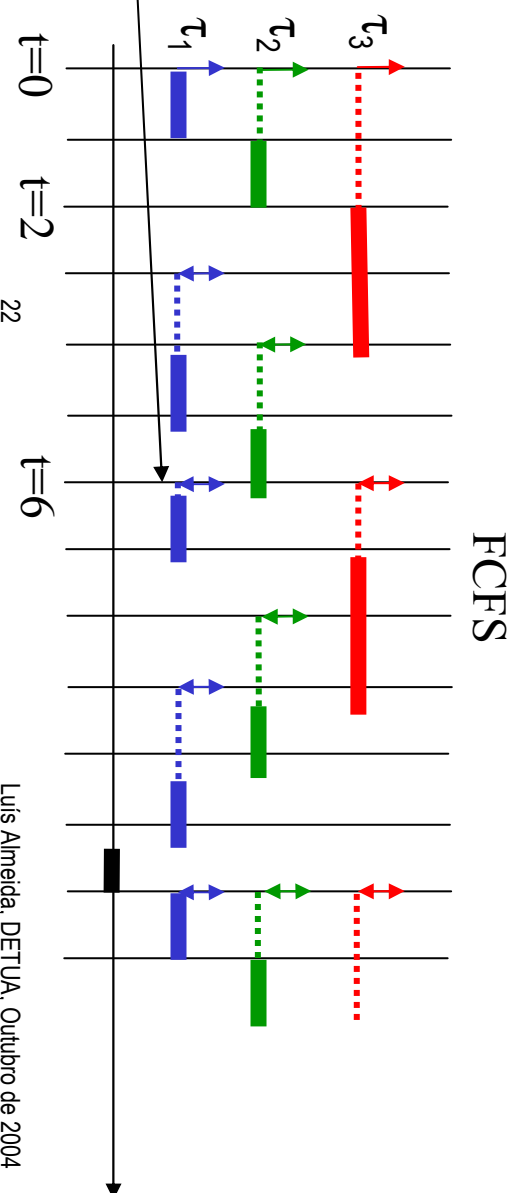
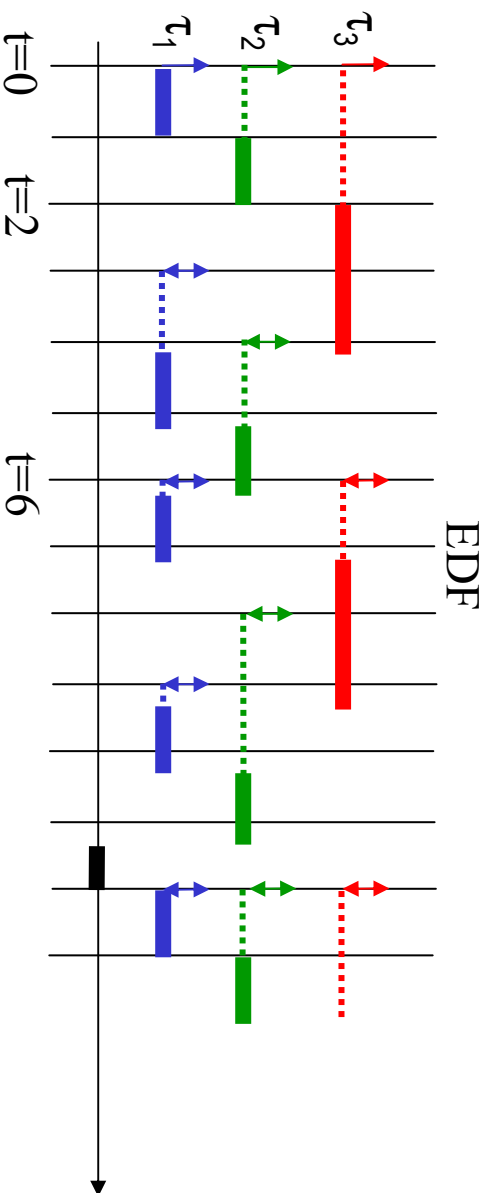


# Escalonamento FCFS – mesmo exemplo

Tabela de propriedades das tarefas

$\tau_i$	$T_i$	$C_i$
1	3	1
2	4	1
3	6	2.1

Quando a idade é igual o critério de desempate é determinante!



## Resumo da Aula 6

- Escalonamento *on-line* com prioridades dinâmicas
- O critério *EDF - Earliest Deadline First*: limite de utilização de CPU
- **Optimalidade de EDF e comparação com RM:**
  - nível de escalonabilidade, número de preempções, *jitter* de disparo e tempo de resposta
- Outros critérios de prioridades dinâmicas:
  - *LLF (LST) - Least Laxity (Slack) First*
  - *FCFS - First Come First Served*