

Universidade Federal do Amazonas
Faculdade de Tecnologia
Programa de Pós-graduação em Engenharia Elétrica



Escalonamento (Algoritmos Clássicos)

Lucas Cordeiro

lucascordeiro@ufam.edu.br

Notas de Aula



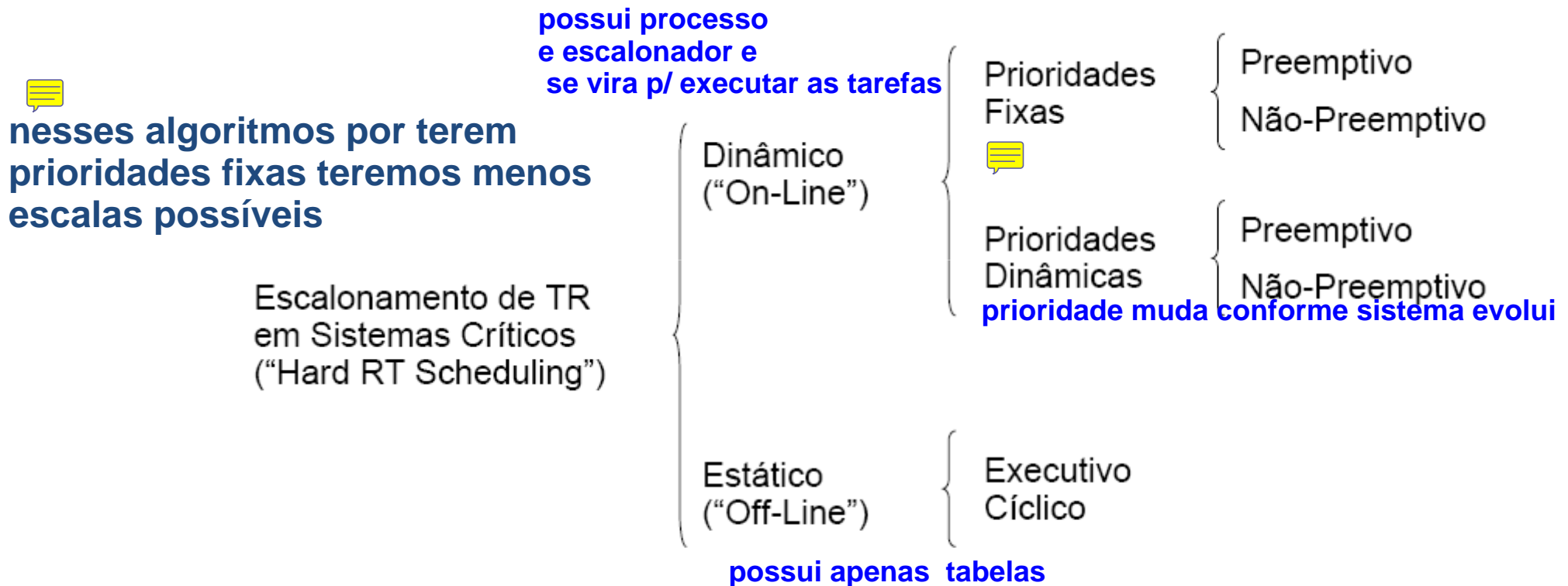
Baseado nas Notas de Aula do Prof. Francisco Vasques, da Faculdade de Engenharia da Universidade do Porto.

<http://www.fe.up.pt/~vasques>

Baseado nas notas de aula de Automação Industrial do Prof. Peter Göhner do *Institut für Automatisierungs- und Softwaretechnik (IAS)* da Universidade de Stuttgart,

<http://www.ias.uni-stuttgart.de/>

Classificação Alg. Escalonamento



prioridades c/base na frequencia dará prioridade a eventos Q ocorrem c/+frequencia

OBS.; frequencia e inversamente proporcional ao período.

Escal. em Sistemas Críticos e não Críticos

noexecutivo cíclico a escala e calculada antes, de acordo c/dados da tabela.

- O escalonamento em *sistemas críticos* é baseado na **adequabilidade de recursos**
 - **garantia** em fase de concepção de um **tempo de resposta** adequado para a execução de cada uma das tarefas
 - Os *testes de escalonabilidade* devem ser **determinísticos**
 - *deve-se verificar que, para os pressupostos de carga assumidos, o tempo de resposta máximo é inferior à sua meta temporal*
- Para o caso de *sistemas não-críticos*, sabendo que a *violação de metas temporais* não é crítica, admite-se a utilização de **métodos probabilísticos**
 - *por exemplo*, baseados em **simulações**, para a verificação da garantia de escalonabilidade

Executivo Cíclico (1)

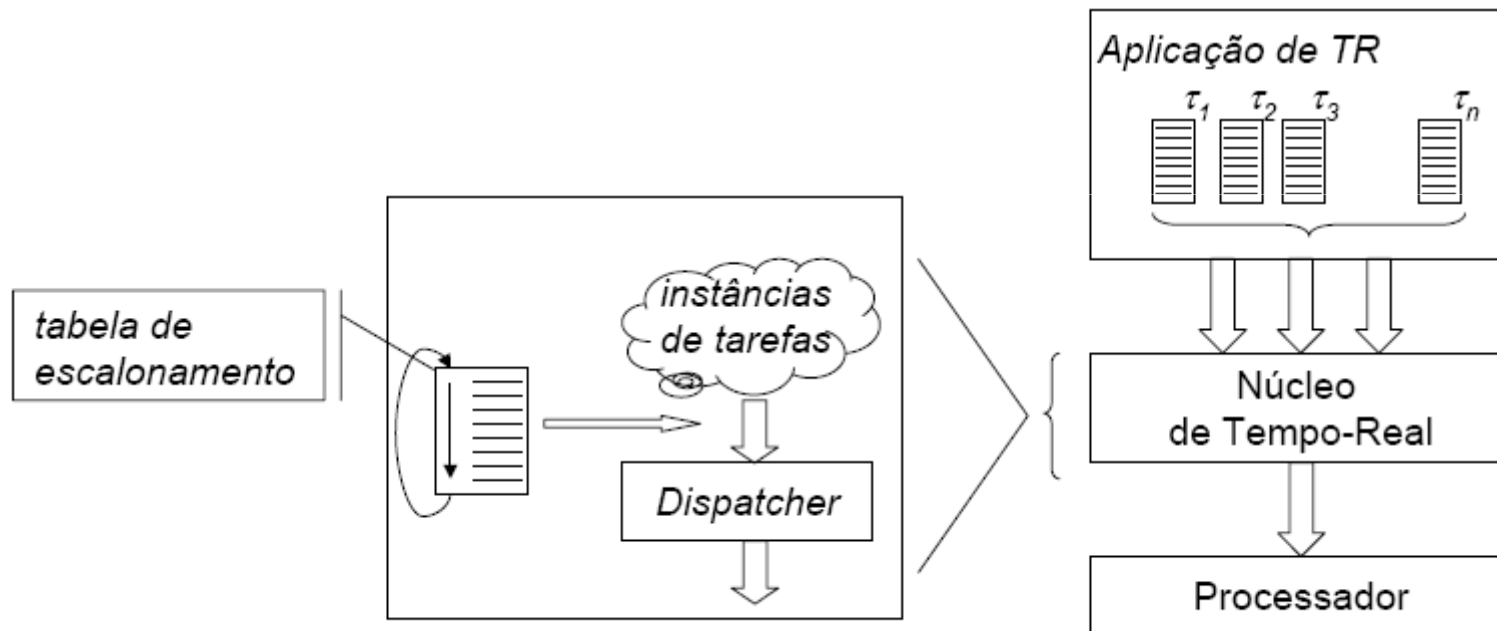


■ Escalonamento Estático

- Um escalonador é considerado estático se **tomar previamente** todas as decisões de escalonamento, gerando, em tempo de compilação, uma **tabela de escalonamento** com a sequência de execução das tarefas
 - Em tempo de execução, esta tabela será repetidamente executada pelo processo de *“dispatcher”*, com a *periodicidade adequada*
- Qualquer **variação no modelo** de tarefas implicará a geração de uma nova tabela de escalonamento
 - A *garantia de escalonabilidade é fornecida por simples inspeção da tabela de escalonamento*

Executivo Cíclico (2)

- Escalonamento Estático (Executivo Cíclico)



Executivo Cíclico (3)

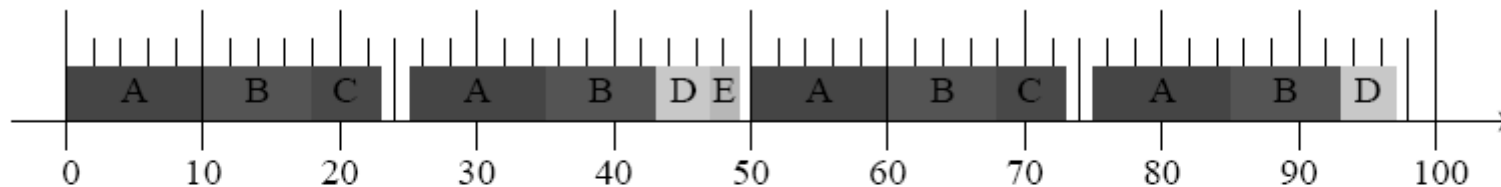
■ Exemplo de Escalonamento Estático

$$U_i = \frac{C_i}{T_i}$$

$$U_A = \frac{10}{25} = 0,4$$

tarefa	<i>C</i>	<i>T</i>	<i>d</i>	<i>U</i>
<i>A</i>	10	25	25	0,4
<i>B</i>	8	25	25	0,32
<i>C</i>	5	50	50	0,1
<i>D</i>	4	50	50	0,08
<i>E</i>	2	100	100	0,02
				<i>U</i> total: 0,92

Percentual de CPU diz quanto % cada sist.. usa de carga.
 Tempo de uso total nos indica a folga de tempo p/encaixar
 tarefas ex.: $U_{total}(\text{tempo de uso total}) = 0,92 \Rightarrow \text{folga de } 0,8$



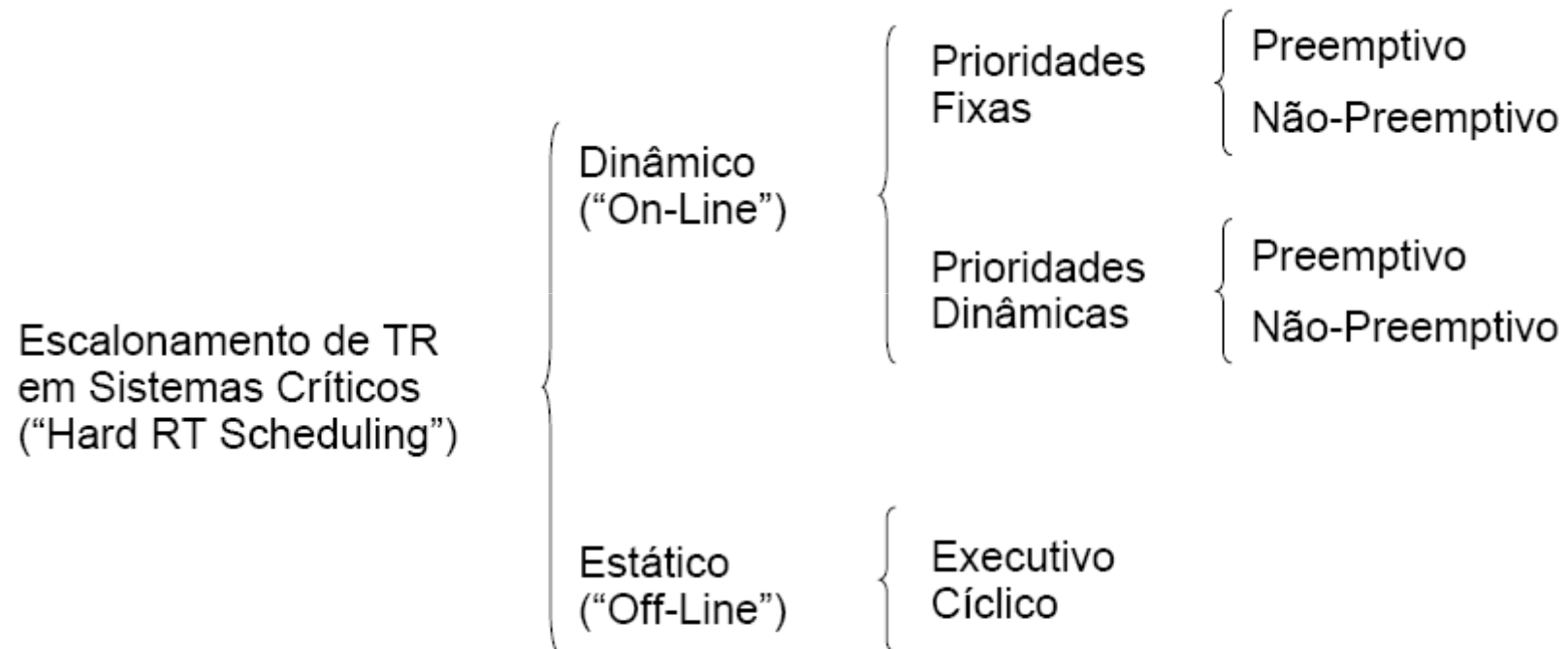
- Tabela de escalonamento organizada em micro-ciclos com duração igual a $m.d.c.\{T\}=25$, sendo o comprimento total da tabela de escalonamento (macro-ciclo) igual ao $m.m.c.\{T\}=100$.

Executivo Cíclico (4)



- Vantagens de um escalonamento estático:
 - Suporte eficaz à implementação de **relações de precedência** entre tarefas
 - Comportamento do sistema completamente **previsível**
 - **Sobrecarga mínima** em tempo de execução
 - Muito utilizado para suporte de aplicações de **elevada criticalidade** (devido à simplicidade do processo de certificação)
- Desvantagens de um escalonamento estático:
 - Escalonamento de **baixa flexibilidade**, dificultando tanto operações de modificação do conjunto de tarefas, como a execução de tarefas esporádicas e/ou aperiódicas

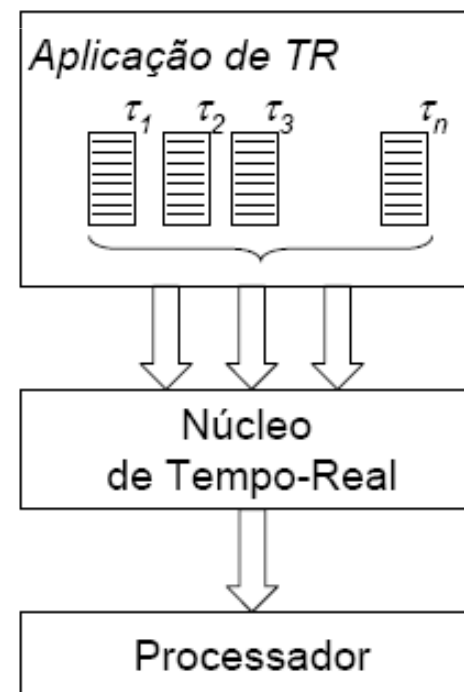
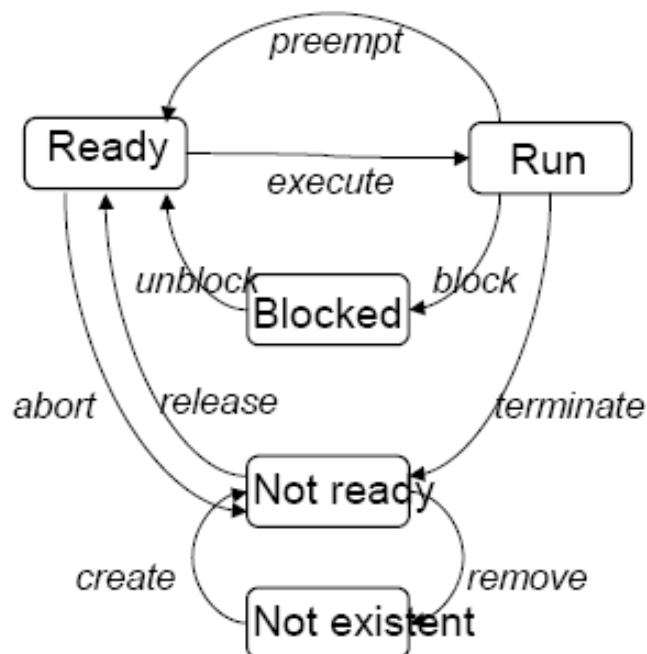
Classificação Alg. Escalonamento



Escalonamento Dinâmico (1)

- Toma as decisões de escalonamento em **tempo de execução**, em função dos pedidos de execução pendentes
 - Atribuição do processador (“ready” → “run”) à tarefa pendente mais prioritária
 - Também referido como “*escalador por prioridades*”

o escalonador poe os processos c+/prioridade p/a CPU



Escalonamento Dinâmico (2)

■ Vantagens:

- Adapta-se facilmente à presença de tarefas com **requisitos variáveis no tempo**, visto efetuar o escalonamento em tempo de execução
- Permite considerar a **importância relativa** das tarefas em função dos seus **requisitos temporais** *olha-se o modelo e consegue-se mais facilmente definir a possibilidade e as prioridades*

■ Aspectos relevantes a considerar:

- A transformação “*requisitos temporais*” → “*prioridade das tarefas*” é da responsabilidade do utilizador
- O **escalonador não efetua a verificação** do respeito das metas temporais associadas à execução das tarefas

■ Desvantagens:

- Escalonador mais **complexo** do que no caso estático, devido à necessidade de efetuar o escalonamento em tempo de execução
- Maior dificuldade de **detecção de sobrecargas**

Prioridades Fixas / Dinâmicas (1)

Escalonamento de TR
em Sistemas Críticos
("Hard RT Scheduling")

Dinâmico
("On-Line")

Prioridades
Fixas

Preemptivo

Não-Preemptivo

Prioridades
Dinâmicas

Preemptivo

Não-Preemptivo

Estático
("Off-Line")

Executivo
Cíclico

Prioridades Fixas / Dinâmicas (2)

fixas- são mais simples de trabalhar e de analisarsão

- *Escalonamento dinâmico com prioridades fixas:* a cada tarefa é atribuído um determinado nível de prioridade na **fase de concepção**

- Enquanto o modo de funcionamento do sistema se mantiver, o nível de prioridade de cada tarefa permanece inalterado
 - a menos que exista mudanças impostas por mecanismos de sincronização no acesso a recursos partilhados

- *Escalonamento dinâmico com prioridades dinâmicas:* o nível de prioridade **evolui ao longo do tempo** em função da política de escalonamento selecionada.

dinamico mais proximos da realidade

há políticas que pode-se mudar as prioridades desse jeito o escalonador trabalha de forma mais eficiente

- Exemplo: algoritmo EDF (*“Earliest Deadline First”*), para o qual o nível de prioridade de uma tarefa será tanto maior quanto mais próxima estiver a sua meta temporal



Rate Monotonic

prioridade c/base na frequencia

periodo =deadline

$p=1/f$

obs.: neste exemplo não é ppossivel uar o rate monotônico, pois o o deadline é diferente dp periodo.

	C	Deadline	T(periodo)
A1	7	12	15
A2	4	18	20
A3	11	11	14

Rate Monotonic (1)

é uma política para atribuir prioridades e gerar escalas fixas

- Algoritmo definido para um modelo simplificado de tarefas:
 - As tarefas são **independentes** entre si (não têm relações de precedência, nem necessidades de sincronização)
 - Todas as tarefas têm uma **ativação periódica**, com uma meta temporal igual ao seu período (ou seja, qualquer execução deve ser finalizada antes da próxima data de ativação da tarefa)
 - Todas as tarefas têm um **tempo máximo de execução** conhecido
- Este algoritmo define a **ordem de atribuição de prioridades** a um conjunto de tarefas, na ordem inversa da periodicidade das tarefas:
 - Desde a tarefa de **menor período** à qual é atribuída a **maior prioridade**, até à tarefa de **maior periodicidade** à qual é atribuída a **menor prioridade**
 - As situações de empate serão resolvidas arbitrariamente

Rate Monotonic (2)

- Trata-se de um **algoritmo ótimo** para sistemas mono-processador
 - Se qualquer conjunto de tarefas (periódicas, independentes) pode ser escalonado por um escalonador **dinâmico com prioridades fixas**, então também pode ser escalonado pelo RM

- Teste suficiente de escalonabilidade escalonabilidade é a capacidade de escalonar ele diz se é possível escalonar: -se passar pode-se escalonar; se não passa DEPENDE(talvez possa ou talvez não possa)
Ci->tempo de execução
T-> periodo
$$U = \sum_{i=1}^n \frac{C_i}{T_i} \leq n(\sqrt[n]{2} - 1)$$

se o teste não passar não ha informação suficiente daí necessita-se de um modelo melhor (pode ser uma escala na mão) se conseguir gerar um esquema na mão => dá p/fazer

- **um teste suficiente (não necessário):** pode haver conjuntos de tarefas escalonáveis apesar de **não respeitarem o teste**

teste suficiente é tão restritivo que se passar com certeza dá , se não passar pode ser que dê, pode ser que não dê

$$n = 2, \quad U = 0,83$$

$$n = 3, \quad U = 0,78$$

$$n = 4, \quad U = 0,76$$

$$n \rightarrow +\infty, \quad U \rightarrow 0,693$$

U= tempo Q uso/ Tempo total

Rate Monotonic (3)

- Vantagens do algoritmo RM
 - Simplicidade
 - Adequado para utilização em sistemas operativos existentes
 - Pode ser utilizado para a atribuição de **prioridades a níveis de interrupção**
- Desvantagens do algoritmo RM
 - Não adequado quando se têm **metas temporais inferiores ao período**
 - Não suporta **exclusão mútua** no acesso a recursos **quando há exclusão mútua não funciona rate monotônico**
- Resultado fundamental
 - Em [Liu and Layland, 1973] foi demonstrado que, **se a meta temporal da tarefa de menor prioridade for respeitada após um instante crítico, então o conjunto de tarefas é sempre escalonável**

Exemplos de Rate Monotonic (1)

- Três cenários de escalonamento utilizando o algoritmo RM (considerando conjuntos de tarefas com diferentes taxas de ativação) :
 - Conjunto de tarefas para o qual o teste de escalonabilidade é respeitado (**o conjunto de tarefas é sempre escalonável**)
 - Conjunto de tarefas para o qual o teste de escalonabilidade não é respeitado
 - No entanto, após o instante crítico, a meta temporal da tarefa de menor prioridade é respeitada (**o conjunto de tarefas é sempre escalonável**) **respeitado pelo escalonador**
 - Conjunto de tarefas para o qual nem o teste de escalonabilidade, nem a meta temporal da tarefa de menor prioridade (após o instante crítico) são respeitados (**o conjunto de tarefas não é escalonável**) **se a tarefa perder o dead line ela não funciona**

obs quando o tempo de resposta < deadline => vamos conseguir fazer a escala funcionar

Exemplos de Rate Monotonic (2)

■ 1º Exemplo:

» tarefas periódicas: $\tau = \{C_i, T_i\}$; $d_i = T_i$; $U = 77,5\%$;

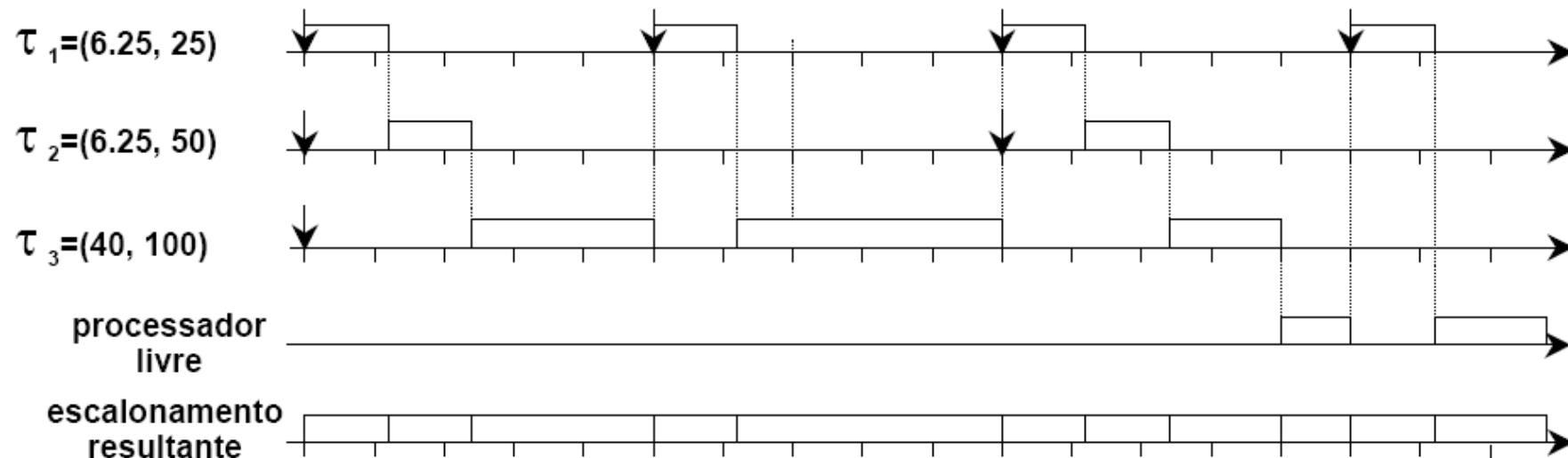
» data de activação das tarefas simultânea;

» escalonamento sempre realizável:

■ teste *suficiente* de escalonabilidade *respeitado*:

tarefa	C	T	U
τ_1	6.25	25	0.25
τ_2	6.25	50	0.125
τ_3	40	100	0.4

$$U = 0,775 \leq 0,7798$$



obs.: quando reduz o periodo aumenta-se a frecuencia e a prioridade aumenta

Exemplos de Rate Monotonic (3)

■ 2º Exemplo:

» tarefas periódicas: $\tau = \{C_i, T_i\}$; $d_i = T_i$; $U = 87,5\%$;

» data de activação das tarefas simultânea;

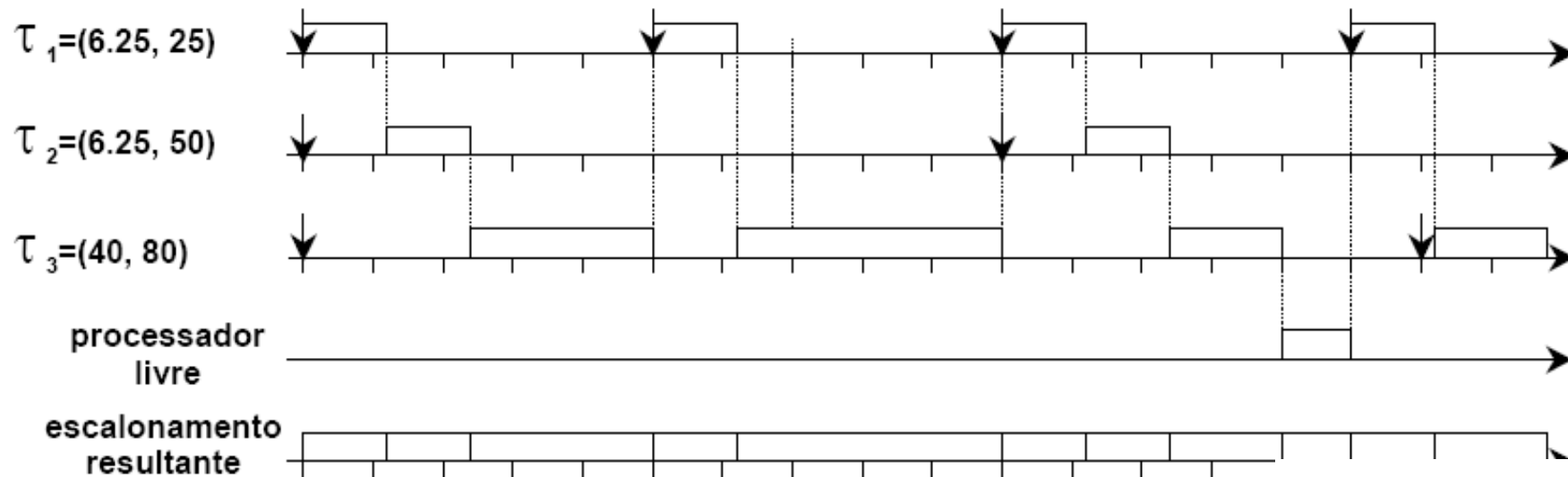
» escalonamento sempre realizável:

tarefa	C	T	U
τ_1	6.25	25	0.25
τ_2	6.25	50	0.125
τ_3	40	80	0.5

■ teste suficiente de escalonabilidade não é respeitado:

$$U = 0,8750 \leq 0,7798$$

■ no entanto, a meta temporal da tarefa τ_3 é respeitada após o instante crítico, logo o conjunto de tarefas é sempre escalonável.



aquele que tiver menor prioridade consegue

Exemplos de Rate Monotonic (4)

■ 3º Exemplo:

» tarefas periódicas: $t=\{C_i, T_i\}$; $d_i=T_i$; $U=96,3\%$;

» data de activação das tarefas simultânea;

» escalonamento não é realizável:

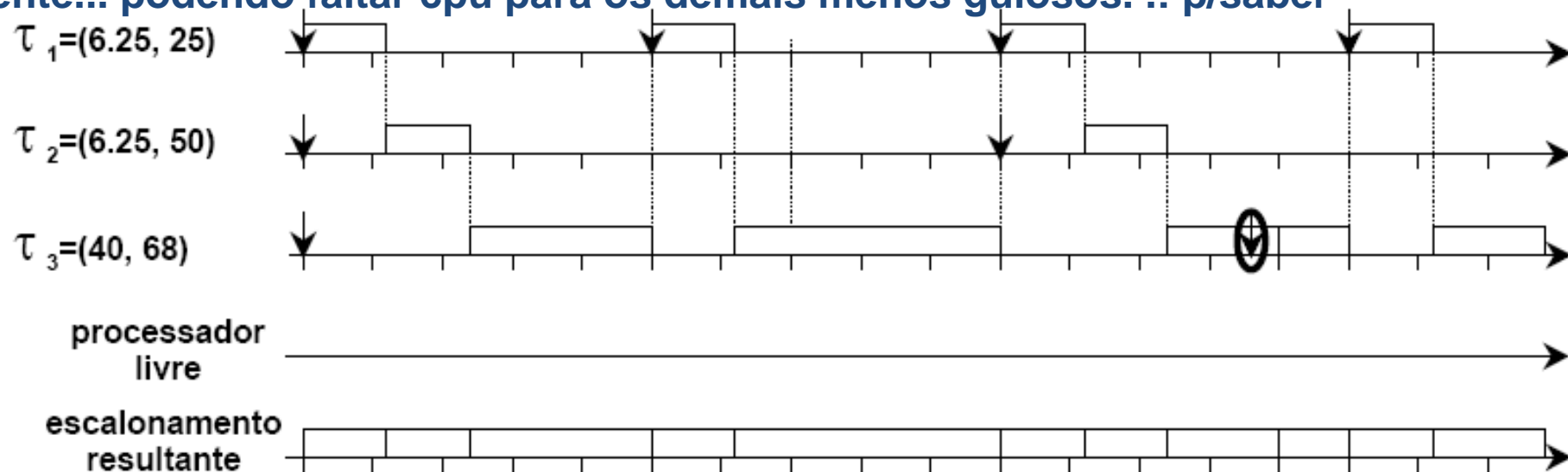
tarefa	C	T	U
τ_1	6.25	25	0.25
τ_2	6.25	50	0.125
τ_3	40	68	0.588

■ teste suficiente de escalonabilidade não é respeitado:

$$U = 0,963 \leq 0,7798$$

■ meta temporal da tarefa t_3 não é respeitada após o instante crítico.

metafora: tarefa muito gulosa possui frequencia maior e termina tudo 1º daí vai querer rodar novamente... podendo faltar cpu para os demais menos gulosos. ∴ p/saber



as tarefas de maior prioridade vão interferir nas de menor prioridade, daí a necessidade... a tarefa de menor periodo tem maior prioridade

RM: Tempo de Resposta (1)

- A análise de escalonabilidade de um conjunto de tarefas através do cálculo da utilização apresenta grandes limitações
 - devido ao fato de **não ser exata** e de apenas ser aplicável a modelos de tarefas muito simples
- Através do cálculo do **Tempo de Resposta** obtém-se um teste de escalonabilidade exato:
 - se o teste for positivo, então o conjunto de tarefas é sempre escalonável
 - se o teste for negativo, então algumas metas temporais serão ultrapassadas durante a execução
 - exceto se os tempos máximos de execução das tarefas forem muito pessimistas

RM: Tempo de Resposta (2)

- A análise de escalonabilidade através do cálculo do tempo de resposta leva em consideração modelos de tarefas mais elaborados:
 - Permite a consideração de **relações de precedência** e de **exclusão**
 - É **válido para qualquer escalonamento dinâmico** com prioridades estáticas, qualquer que seja a regra de atribuição de prioridades às tarefas
- Esta análise é baseada no cálculo da **máxima interferência** que o escalonamento de uma determinada tarefa pode sofrer, devido ao escalonamento das tarefas de maior prioridade

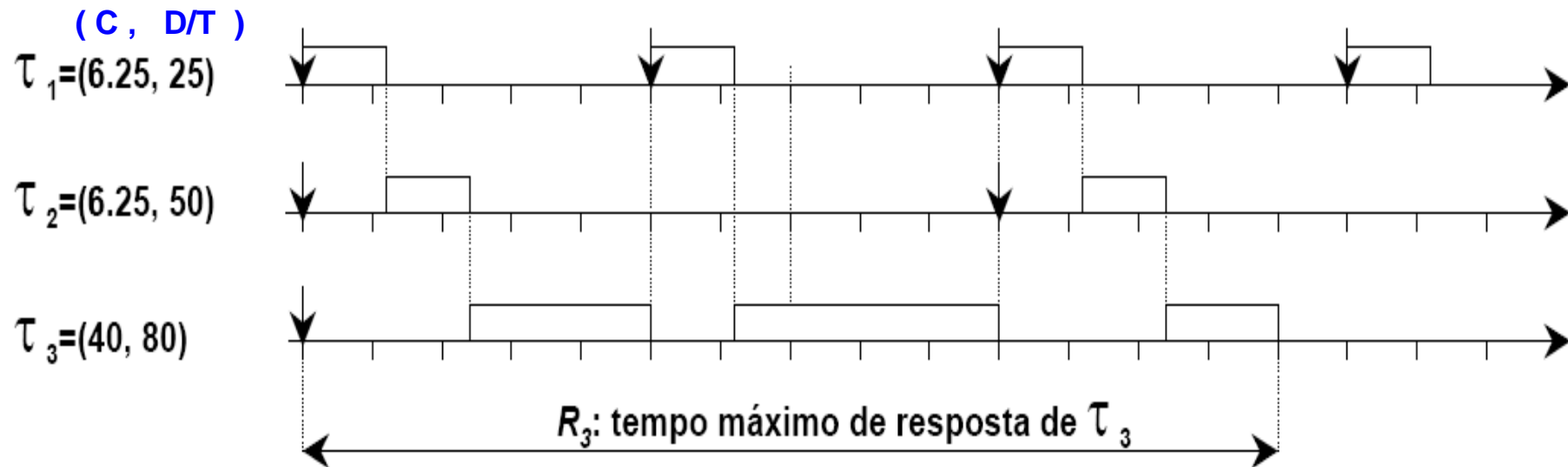
RM: Tempo de Resposta (3)

- Tempo de Resposta da tarefa τ_i : $R_i = C_i + I_i$
 - I_i representa a interferência que o escalonamento da tarefa τ_i sofre devido ao escalonamento das tarefas de maior prioridade ($I_1=0$)

- Interferência:

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times C_j$$

$hp(i)$ é o conjunto de tarefas com prioridade superior à prioridade da tarefa τ_i



RM: Tempo de Resposta (4)

- Tempo Máximo de Resposta:
$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times C_j$$
- A equação é recursiva, pelo que deve ser calculada através de iterações sucessivas até que:
 - ou o tempo de resposta da tarefa seja superior à sua meta temporal (logo a tarefa não será escalonável)
 - ou o resultado convergir, ou seja o tempo de resposta na iteração $x+1$ seja igual ao tempo de resposta na iteração x

RM: Tempo de Resposta (5)

- Cálculo do Máximo Tempo de Resposta:
 - Através de uma equação recursiva, para o qual
 - valor inicial: $w_i^0 = C_i$
 - iterações posteriores: $w_i^{x+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^x}{T_j} \right\rceil \times C_j$
- O conjunto de valores $w_i^0, w_i^1, w_i^2, \dots, w_i^n, \dots$ é monotonicamente não decrescente
- Quando $w_i^{n+1} = w_i^n$ a solução para a equação foi encontrada

se violar o deadline => não vai dar

RM: Tempo de Resposta (6)

■ Cálculo do Tempo de Resposta:

» Tarefa t1: $R_1 = C_1 = 6,25$ como não há interferência $R_1 = C_1 = 6,25$

» Tarefa t2: $w_2^0 = 6,25$

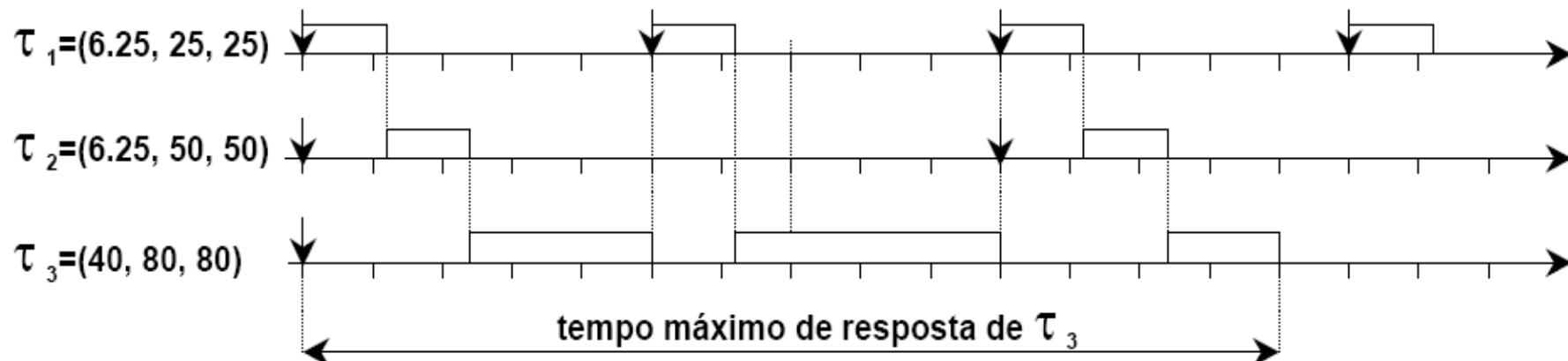
$$w_2^1 = 6,25 + \left\lceil \frac{6,25}{25} \right\rceil \times 6,25 = 12,5$$

tempo w2 período do cara q ta atrapalhando

$$w_2^2 = 6,25 + \left\lceil \frac{12,5}{25} \right\rceil \times 6,25 = 12,5$$

$R_2 = 12,5$ se o resultado se repetir

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times C_j$$



RM: Tempo de Resposta (7)

■ Cálculo do Tempo de Resposta:

» Tarefa t_3 : $w_3^0 = 40$

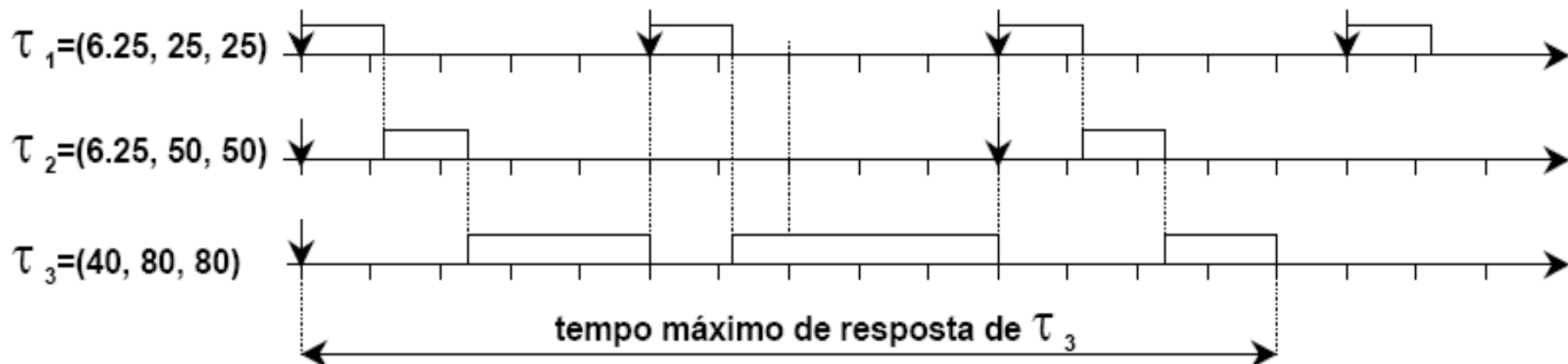
$$w_3^1 = 40 + \left\lceil \frac{40}{25} \right\rceil \times 6,25 + \left\lceil \frac{40}{50} \right\rceil \times 6,25 = 58,75$$

$$w_3^2 = 40 + \left\lceil \frac{58,75}{25} \right\rceil \times 6,25 + \left\lceil \frac{58,75}{50} \right\rceil \times 6,25 = 71,25$$

$$w_3^3 = 40 + \left\lceil \frac{71,25}{25} \right\rceil \times 6,25 + \left\lceil \frac{71,25}{50} \right\rceil \times 6,25 = 71,25$$

$$R_3 = 71,25$$

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times C_j$$



RM: Tempo de Resposta (8)

- O conjunto de tarefas é escalonável (tempo de resposta de cada uma das tarefas inferior à sua meta temporal), apesar do teste suficiente de escalonabilidade (baseado na utilização) ser negativo.

<i>Tarefa</i>	<i>C (duração)</i>	<i>T (período)</i> <i>d (deadline)</i>	<i>R (tempo de</i> <i>resposta)</i>	<i>U (utilização)</i>
τ_1	6,25	25	6,25	0,25
τ_2	6,25	50	12,5	0,125
τ_3	40	80	71,25	0,5
				$0,875 > 0,7798$

Algoritmo do Tempo de Resposta

```
for i in 1..N loop -- para cada processo por vez
  n := 0
   $w_i^n := C_i$ 
  loop
    calculate new  $w_i^{n+1}$ 
    if  $w_i^{n+1} = w_i^n$  then
       $R_i = w_i^n$ 
      exit value found
    end if
    if  $w_i^{n+1} > T_i$  then estourar o dead line
      exit value not found
    end if
    n := n + 1
  end loop
end loop
```

Exercício sobre RM

- Verifique se o seguinte conjunto de tarefas é escalonável usando o algoritmo do *Rate Monotonic*. Em caso positivo, forneça o diagrama temporal

obs.: quando temos 2 períodos iguais considero qualquer um como prioritário p/iniciar o calculo

Tarefa	Período (T) Deadline (D)	Tempo de Execução (C)
A	25	10
B	25	8
C	50	5
D	50	4
E	100	2

1º calcula tempo de execução da CPU: $U=C/T$, se



DEADLINE MONOTONIC

Deadline Monotonic (1)

- Limitação do algoritmo RM: segundo este algoritmo, a cada tarefa é atribuída uma **prioridade** proporcional à sua **cadência de ativação**
 - No entanto, a importância de uma tarefa pode ser independente da sua cadência de ativação
 - Existem outros parâmetros temporais que podem ser considerados

Deadline Monotonic (2)

- Algoritmo de atribuição de prioridades a um conjunto de tarefas periódicas, independentes e com metas temporais menores ou iguais ao respectivo período ($d_i \leq T_i$)
- [Leung and Whitehead, 1982]
- A atribuição de prioridades às tarefas é efetuada na ordem inversa do valor da sua meta temporal
 - desde a tarefa com **menor meta temporal** à qual é atribuída a **maior prioridade**, até à tarefa de **maior meta temporal** à qual é atribuída a **menor prioridade**
 - as situações de **empate** são resolvidas **arbitrariamente**
- Trata-se de um algoritmo ótimo para sistemas mono-processador

Deadline Monotonic (3)



- Vantagens do algoritmo DM
 - **Simples e adequado** para utilização em sistemas operacionais existentes
 - Pode ser utilizado para a atribuição de prioridades a **níveis de interrupção**
 - Admite tarefas com **metas temporais inferiores ao período**
- Desvantagens do algoritmo DM
 - Modelo de tarefas também muito **limitado**
 - Não suporta **exclusão mútua** no acesso a recursos partilhados

Deadline Monotonic – Exemplos (1)



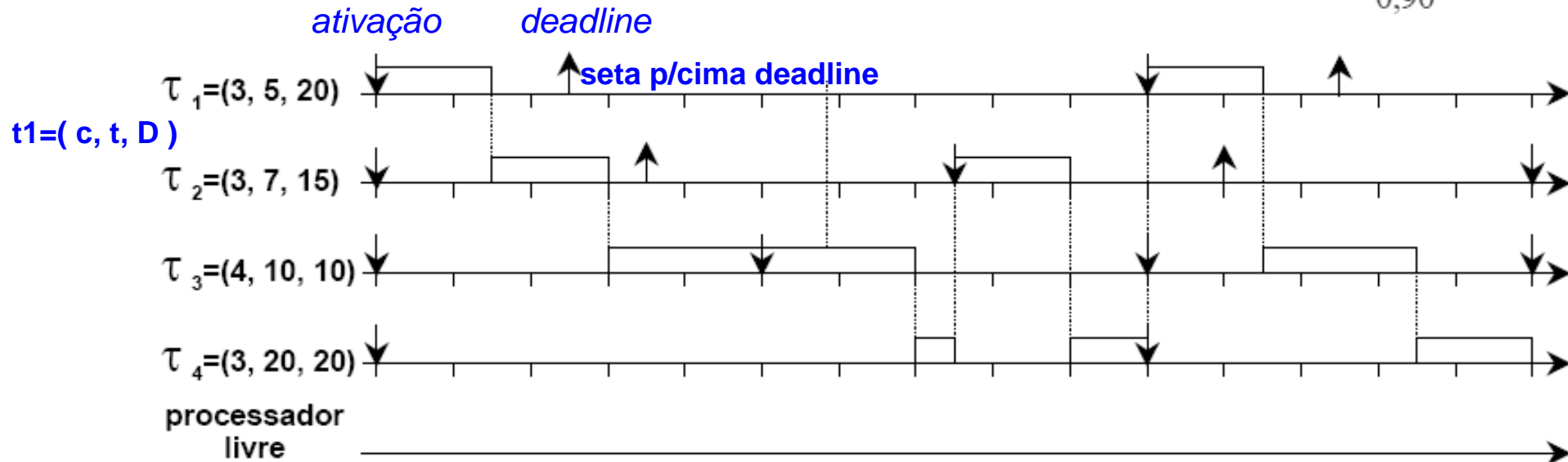
- Apresentam-se 2 cenários de escalonamento por prioridades fixas (idêntico conjunto de tarefas), considerando:
 - prioridades atribuídas segundo o algoritmo DM (tarefas ordenadas por valor de meta temporal crescente):
 - calcula-se do tempo de resposta de cada uma das tarefas
 - verifica-se que o conjunto de tarefas é sempre escalonável
 - prioridades atribuídas segundo o algoritmo RM (tarefas ordenadas por periodicidade crescente):
 - verifica-se que o conjunto de tarefas não é escalonável

Deadline Monotonic – Exemplos (2)

- 1° Exemplo:

» conjunto de tarefas ordenado por metas temporais crescentes

	<i>Tarefa</i>	<i>T (período)</i>	<i>C (duração)</i>	<i>d (m. temporal)</i>	<i>P (prioridade)</i>	<i>U (utilização)</i>
	τ_1	20	3	5	1	0,15
	τ_2	15	3	7	2	0,20
	τ_3	10	4	10	3	0,40
	τ_4	20	3	20	4	0,15
						0,90



Deadline Monotonic – Exemplos (3)

■ 1º Exemplo (cálculo do tempo de resposta):

» Tarefa t1:

» Tarefa t2:

$$R_1 = C_1 = 3$$

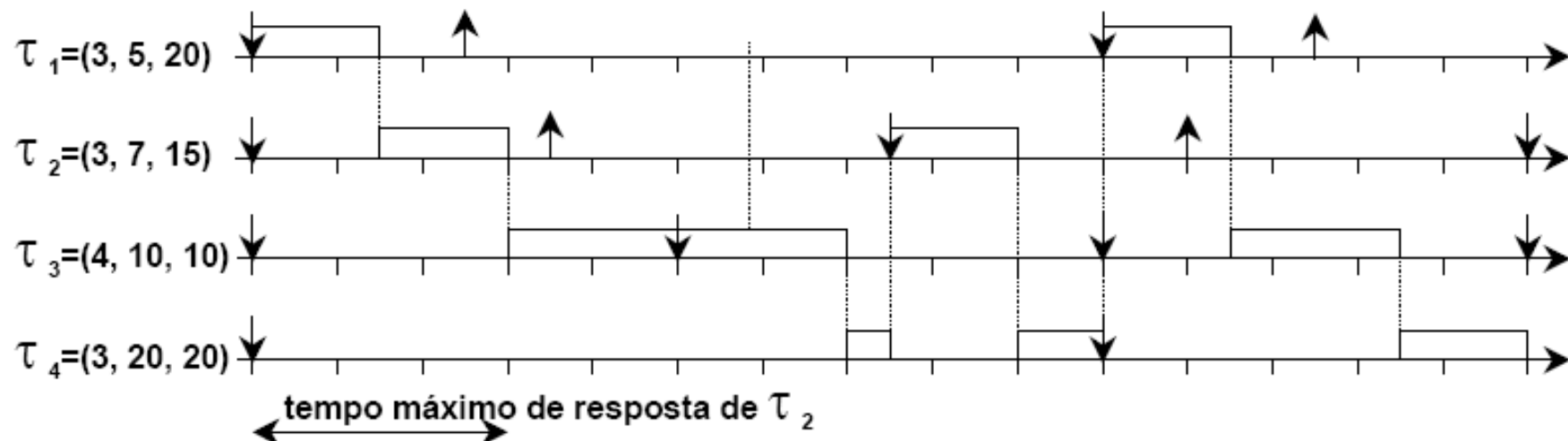
$$w_2^0 = 3$$

$$w_2^1 = 3 + \left\lceil \frac{3}{20} \right\rceil \times 3 = 6$$

$$w_2^2 = 3 + \left\lceil \frac{6}{20} \right\rceil \times 3 = 6$$

$$R_2 = 6$$

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times C_j$$



Deadline Monotonic – Exemplos (4)

■ 1º Exemplo (cálculo do tempo de resposta):

» Tarefa τ_3 :

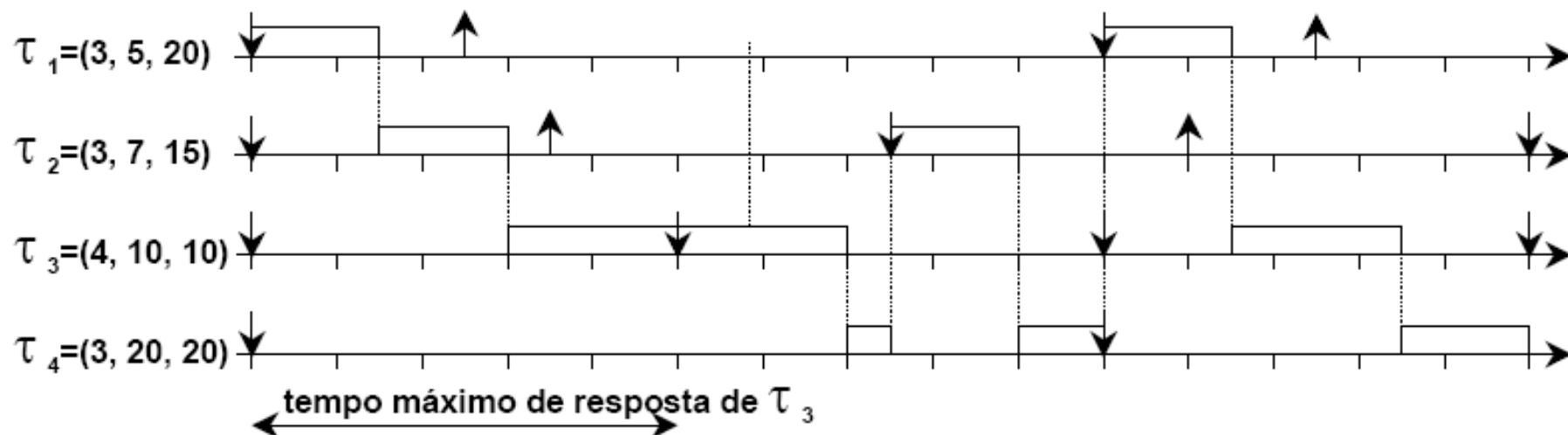
$$w_3^0 = 4$$

$$w_3^1 = 4 + \left\lceil \frac{4}{20} \right\rceil \times 3 + \left\lceil \frac{4}{15} \right\rceil \times 3 = 10$$

$$w_3^2 = 4 + \left\lceil \frac{10}{20} \right\rceil \times 3 + \left\lceil \frac{10}{15} \right\rceil \times 3 = 10$$

$$R_3 = 10$$

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times C_j$$



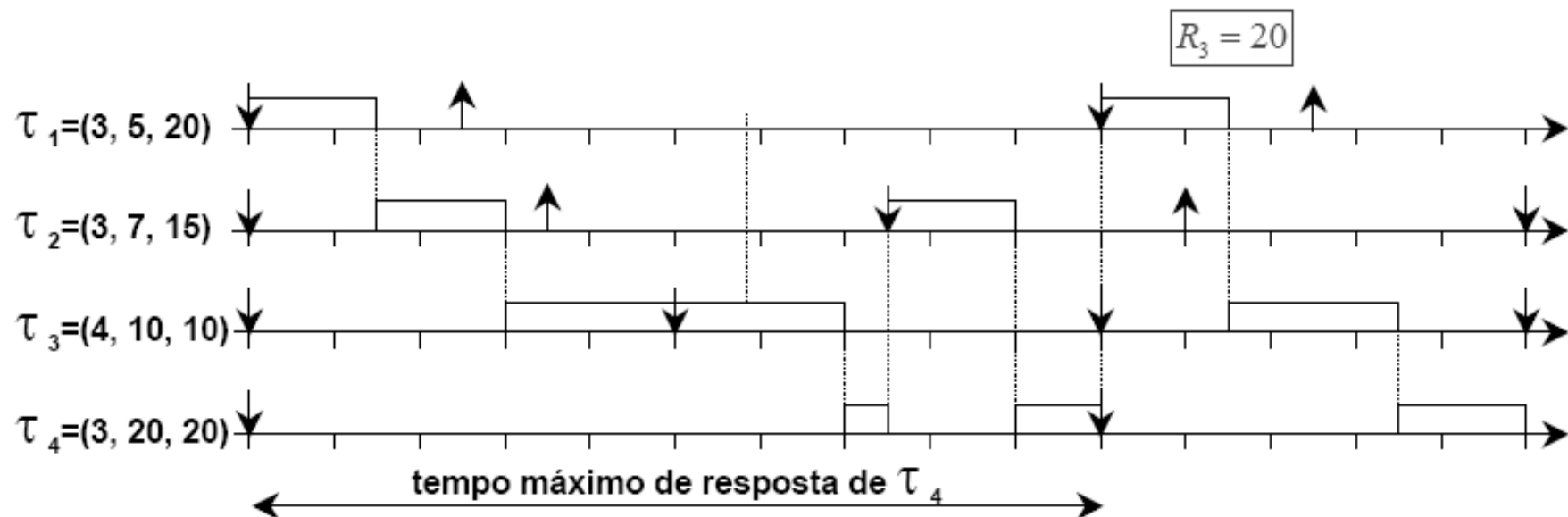
Deadline Monotonic – Exemplos (5)

■ 1º Exemplo (cálculo do tempo de resposta):

» Tarefa τ_4 :

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_j}{T_j} \right\rceil \times C_j$$

$$\begin{aligned} w_4^0 &= 3 & w_4^1 &= 3 + \left\lceil \frac{3}{20} \right\rceil \times 3 + \left\lceil \frac{3}{15} \right\rceil \times 3 + \left\lceil \frac{3}{10} \right\rceil \times 4 = 13 & w_4^3 &= 3 + \left\lceil \frac{17}{20} \right\rceil \times 3 + \left\lceil \frac{17}{15} \right\rceil \times 3 + \left\lceil \frac{17}{10} \right\rceil \times 4 = 20 \\ w_4^2 &= 3 + \left\lceil \frac{13}{20} \right\rceil \times 3 + \left\lceil \frac{13}{15} \right\rceil \times 3 + \left\lceil \frac{13}{10} \right\rceil \times 4 = 17 & w_4^4 &= 3 + \left\lceil \frac{20}{20} \right\rceil \times 3 + \left\lceil \frac{20}{15} \right\rceil \times 3 + \left\lceil \frac{20}{10} \right\rceil \times 4 = 20 \end{aligned}$$



Deadline Monotonic – Exemplos (6)

- 1º Exemplo (conclusão):
 - O conjunto de tarefas é **sempre escalonável** (tempo de resposta de cada uma das tarefas inferior ou igual ao valor da sua meta temporal)

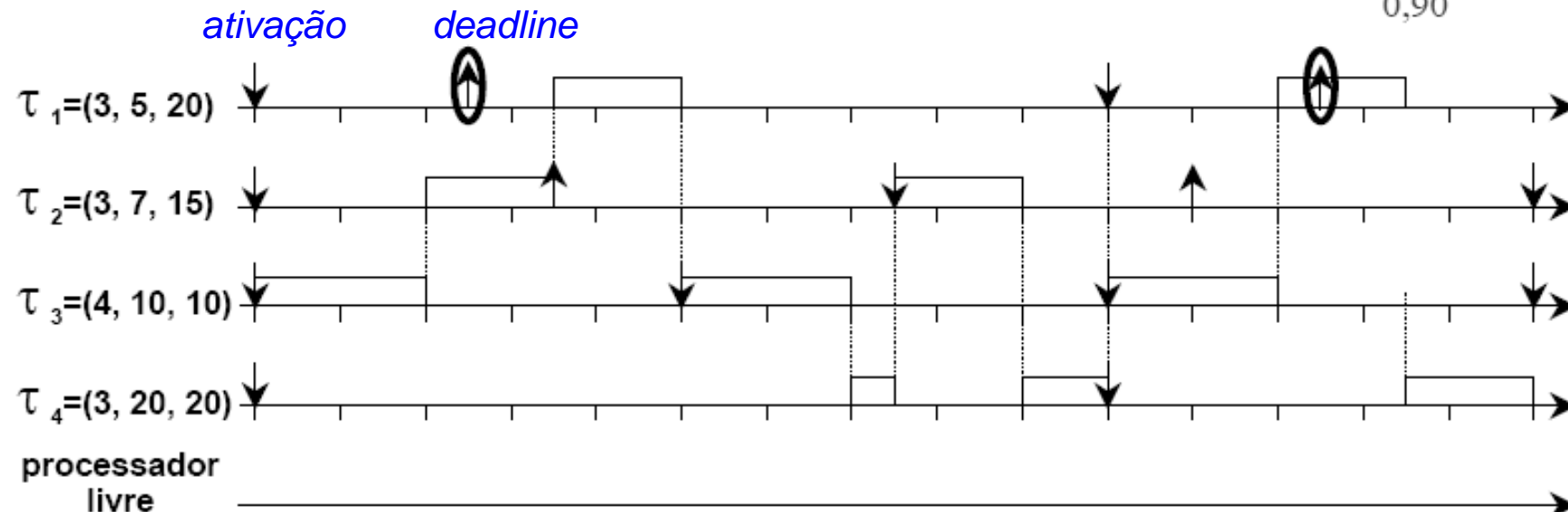
<i>Tarefa</i>	<i>T (período)</i>	<i>C (duração)</i>	<i>d (m. temporal)</i>	<i>R (tempo de resposta)</i>
τ_1	20	3	5	3
τ_2	15	3	7	6
τ_3	10	4	10	10
τ_4	20	3	20	20

Deadline Monotonic – Exemplos (7)

■ 2º Exemplo:

» conjunto de tarefas ordenado por períodos crescentes

	Tarefa	T (período)	C (duração)	d (m. temporal)	P (prioridade)	U (utilização)
<div></div>	τ_1	20	3	5	3	0,15
<div></div>	τ_2	15	3	7	2	0,20
<div></div>	τ_3	10	4	10	1	0,40
<div></div>	τ_4	20	3	20	4	0,15
						0,90



Deadline Monotonic – Exemplos (8)



■ 2º Exemplo (conclusão):

- Por simples inspeção da figura anterior, verifica-se que o conjunto de tarefas não é escalonável quando se utiliza o algoritmo RM
- O algoritmo RM é um algoritmo que **não é ótimo** quando se consideram conjuntos de tarefas com **metas temporais inferiores aos períodos**
- Para este tipo de conjunto de tarefas ($d < T$), a atribuição dos níveis de prioridade deverá ser efetuada utilizando o algoritmo DM (**algoritmo ótimo**)



Earliest Deadline First

Earliest Deadline First (1)

- Proposto por [Liu and Layland, 1973]
- Algoritmo de atribuição dinâmica de prioridades a um conjunto de **tarefas periódicas, independentes** (sem restrições de precedência) e com metas temporais iguais ao respectivo período ($d_i = T_i$)
- Trata-se de um **algoritmo ótimo** para **sistemas mono-processador**, no sentido que se existir um algoritmo capaz de escalonar um conjunto de tarefas periódicas, independentes e com metas temporais iguais ao respectivo período, então o algoritmo EDF também é capaz de o escalonar

Earliest Deadline First (2)



- A atribuição dinâmica de prioridades às tarefas é efetuada na **ordem inversa da distância**, em cada momento, à meta temporal:
 - no momento da ativação de uma tarefa, será atribuída uma **prioridade tanto maior quanto menor a sua distância à meta temporal**
 - relativamente ao estado de todas as tarefas pendentes no momento
 - sempre que uma nova tarefa é ativada, a fila de tarefas pendentes deverá ser reordenada em função da prioridade da tarefa ativada
 - fila de prioridade mantém um conjunto S de elementos, cada qual com um valor associado chamado **chave** com operações de *insert*, *maximum*, *extract-max* e *increase-key*

Earliest Deadline First (3)

- Teste necessário e suficiente de escalonabilidade para o caso preemptivo:

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

- o que significa que qualquer conjunto de tarefas será escalonável pelo algoritmo EDF, desde que a utilização do processador não exceda 100%

Earliest Deadline First (4)



- Vantagens:
 - algoritmo ótimo, capaz de escalonar conjuntos de tarefas com **utilizações até 100%**
- Desvantagens:
 - maior **complexidade** associada à sua implementação, consequência do carácter dinâmico da atribuição de prioridades
 - **perda de metas temporais**, pois é difícil de prever para o caso de sobrecargas transitórias

Exemplos de escalonamento EDF vs RM (1)



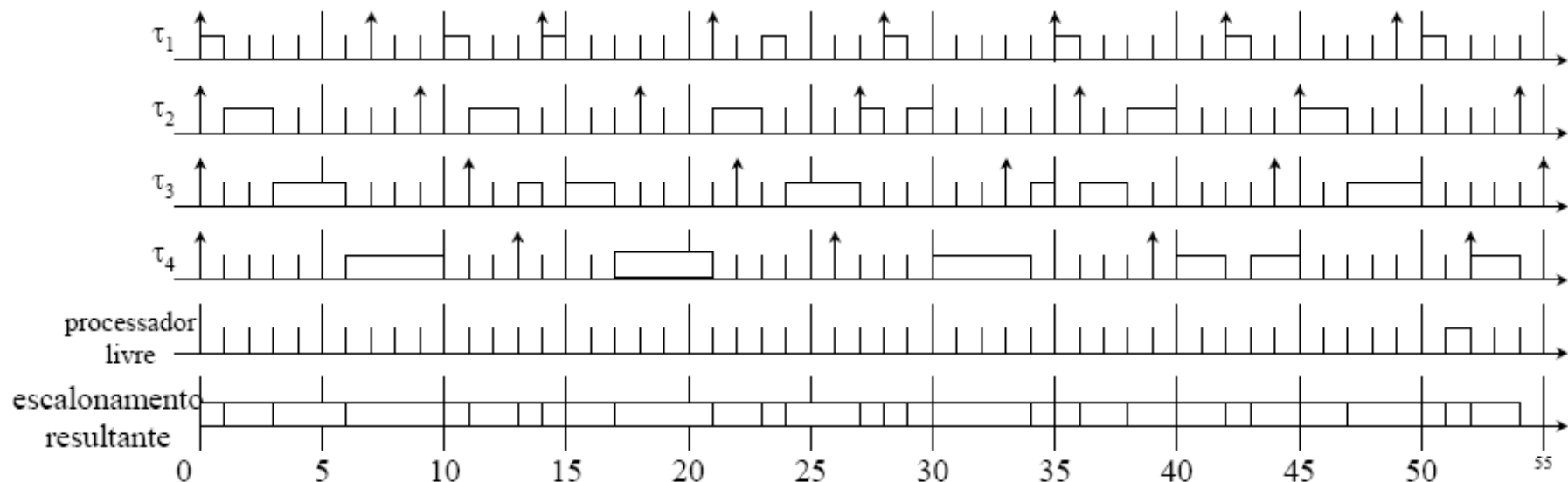
- Apresentam-se 2 cenários de escalonamento (idêntico conjunto de tarefas), considerando:
 - prioridades dinâmicas atribuídas segundo o algoritmo EDF:
 - tarefas ordenadas (em tempo de execução) por distância à meta temporal crescente
 - verifica-se que o conjunto de tarefas é sempre escalonável
 - prioridades fixas atribuídas segundo o algoritmo RM:
 - tarefas pré-ordenadas (em fase de concepção) por valor de meta temporal crescente
 - verifica-se que o conjunto de tarefas não é escalonável

Exemplos de escalonamento EDF vs RM (2)

■ 1º Exemplo (algoritmo EDF):

- activações simultâneas;
- teste de escalonabilidade respeitado, logo o conjunto de tarefas é sempre escalonável;
- a prioridade das tarefas varia ao longo do tempo, o que torna o sistema de difícil previsibilidade no caso de sobrecargas transitórias.

<i>tarefa</i>	<i>C</i>	<i>T</i>	<i>d</i>	<i>U</i>
τ_1	1	7	7	0,1429
τ_2	2	9	9	0,2222
τ_3	3	11	11	0,2727
τ_4	4	13	13	0,3077
				<i>U total: 0,9455</i>

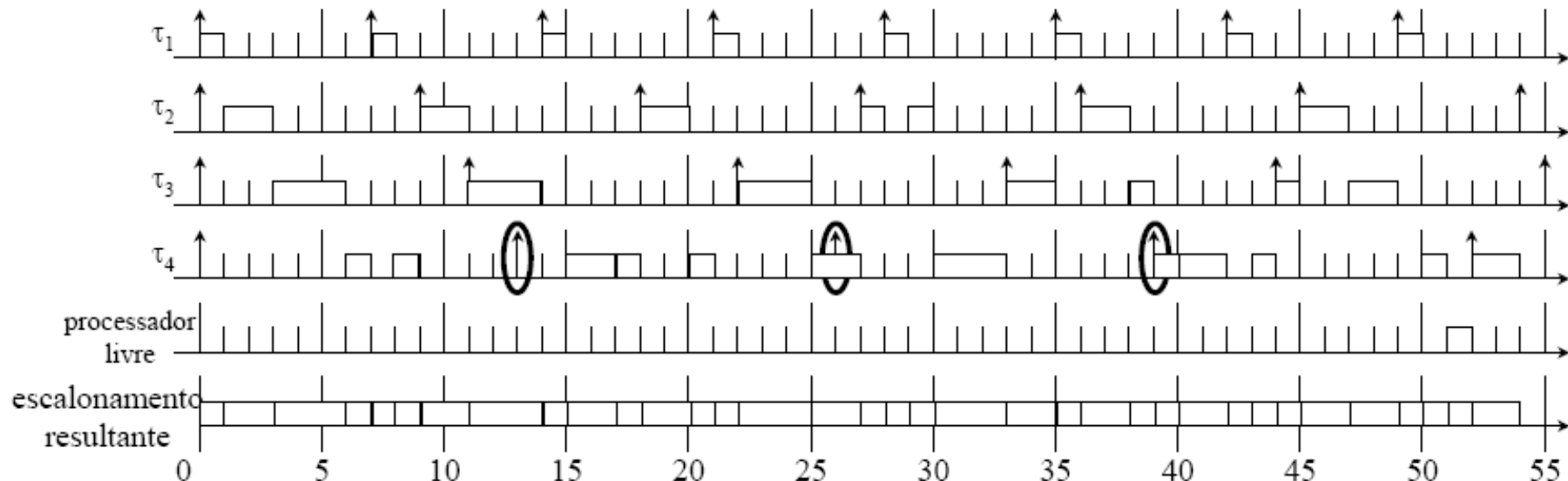


Exemplos de escalonamento EDF vs RM (3)

■ 2º Exemplo (algoritmo RM):

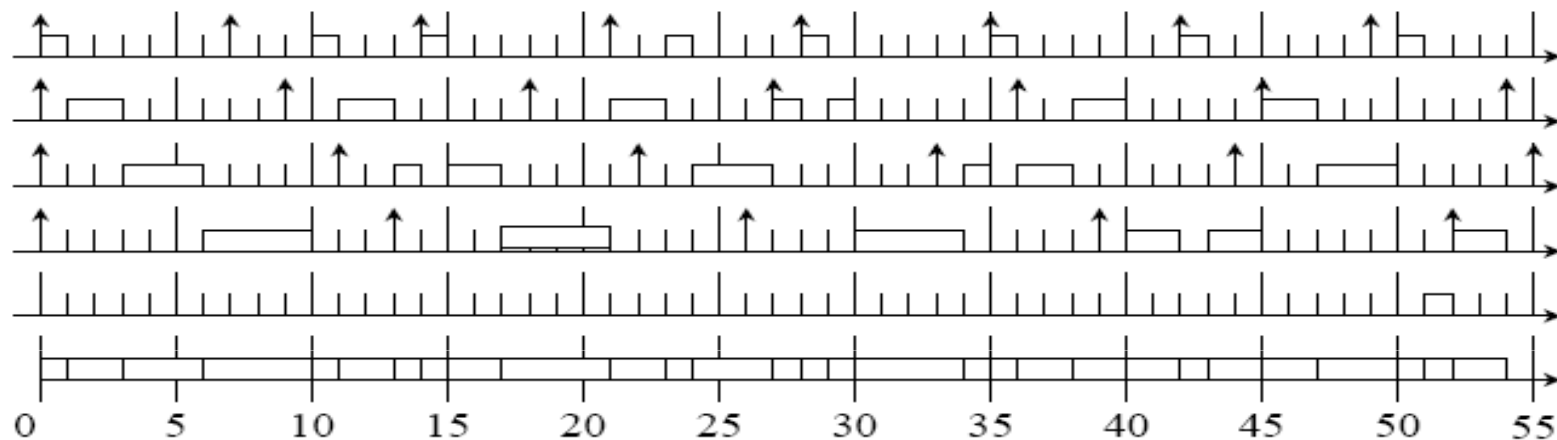
- activações simultâneas;
- A meta temporal da tarefa de menor prioridade após o instante crítico não é respeitada, logo o conjunto de tarefas não é escalonável;
- no caso de sobrecargas transitórias, as tarefas que perderão as suas metas temporais serão as tarefas de menor prioridade (sistema previsível).

<i>tarefa</i>	<i>C</i>	<i>T</i>	<i>d</i>	<i>U</i>
τ_1	1	7	7	0,1429
τ_2	2	9	9	0,2222
τ_3	3	11	11	0,2727
τ_4	4	13	13	0,3077
<i>U_{total}: 0,9455</i>				

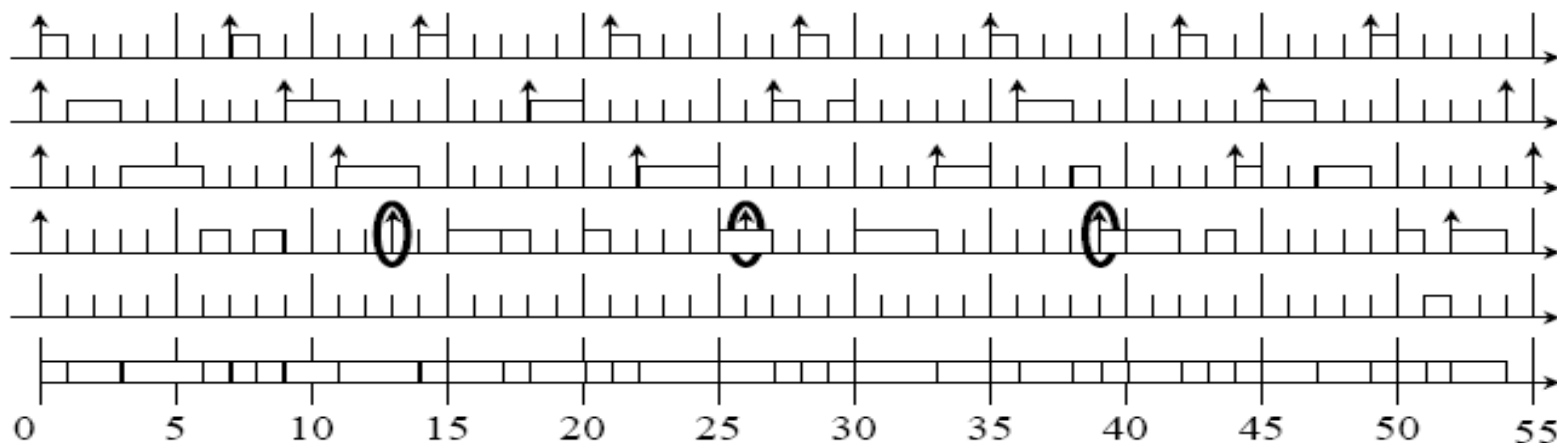


Exemplos de escalonamento EDF vs RM (4)

» EDF



» RM



Exercício sobre EDF

- Sejam $P_1 = (5, 10, 10)$ e $P_2 = (20, 40, 40)$.
 - a) Calcule a utilização do processador U
 - b) Mostre um escalonamento praticável usando EDF
 - c) Demonstre que um escalonamento praticável baseado em prioridades fixas existe ou mostre que não pode existir

Worst-Case Execution Time (WCET)



- Obtido por **medição** ou **análise**
- O problema com a medição é que é difícil ter certeza de quando o pior caso foi observado
- A desvantagem da análise é que um modelo eficaz do processador (incluindo caches, pipelines, estados de espera de memória e assim por diante) devem estar disponíveis

WCET— Encontrando C



- A maioria das técnicas de análise envolvem duas atividades distintas:
 - O primeiro passo é obter a tarefa e decompor o seu código em um grafo dirigido de blocos básicos
 - Estes blocos básicos representam simples linha de código
 - O segundo componente da análise obtém o código da máquina correspondente a um bloco básico e usa o modelo do processador para estimar o seu WCET
 - Uma vez que os tempos para todos os blocos básicos são conhecidos, o grafo dirigido pode ser computado
 - Uma escolha simples entre dois blocos será unificado em um valor (o maior dos dois valores para os blocos alternativos)
 - Loops são limitados por um *bound* máximo

Necessidade de Informação Semântica

```
for I in 1.. 10 loop
  if Cond then
    -- bloco básico de custo 100
  else
    -- bloco básico de custo 10
  end if;
end loop;
```

- Custo simples 10×100 (+ custo da construção do loop), digo 1005
- Mas se Cond é somente verdadeiro em 3 ocasiões então o custo é 375

Exemplo de loop dinâmico – MDC (1)

- Algoritmo de Euclides (*greatest common divisor*)
 - Loop depende de ***b***
 - Não possui padrão simples
 - Interação com a variável ***a***

```
int gcd(int a, int b) {  
    while (b>0) {  
        if (a>b) {  
            a = a - b;  
        }  
        else {  
            b = b - a;  
        }  
    }  
  
    return a;  
}
```

Exemplo de loop dinâmico – MDC (2)

- Algoritmo de Euclides (*greatest common divisor*)
 - Loop depende de ***b***
 - Não possui padrão simples
 - Interação com a variável ***a***
- Calcular WCET
 - Variável incrementada pelo número de ciclos em um dado ponto
 - *Assert* do valor desta variável
 - Usa valor do contra-exemplo como nova estimativa

```
int gcd(int a, int b) {  
    int __time = 3;  
    while (b>0) {  
        __time += 3;  
        if (a>b) {  
            __time += 4;  
            a = a - b;  
        }  
        else {  
            __time += 2;  
            b = b - a;  
        }  
    }  
    __time += 2;  
    assert(__time<=-1);  
    return a;  
}
```

Exemplo de Procedimento – Análise Estática

```
gcc -c -g -Wa,-a,-ad gcd.c > gcd.lst
```

```
gcc -O2 -S -c gcd.c
```

```
int gcd( int a, int b )  
{
```

```
    __asm( "#BB_0" );
```

```
    while ( b > 0 )
```

```
    {
```

```
        __asm( "#BB_1" );
```

```
        if ( a > b )
```

```
        {
```

```
            __asm( "#BB_2" );
```

```
            a = a - b;
```

```
        }
```

```
        else
```

```
        {
```

```
            __asm( "#BB_3" );
```

```
            b = b - a;
```

```
        }
```

```
    }
```

```
    __asm( "#BB_4" );
```

```
    return a;
```

```
}
```

> #BB_0

.LL10:

cmp %o1, 0

ble .LL8

nop

.LL12:

> #BB_1

cmp %o0, %o1

ble .LL4

nop

> #BB_2

cmp %o1, 0

bg .LL12

sub %o0, %o1, %o0

b, a .LL8

.LL4:

> #BB_3

b .LL10

sub %o1, %o0, %o1

.LL8:

> #BB_4

retl

nop

Exemplo de Procedimento – Análise Estática

```
#BB_0
.LL10:
    cmp %o1, 0
    ble .LL8
    nop
.LL12:
#BB_1
    cmp %o0, %o1
    ble .LL4
    nop
#BB_2
    cmp %o1, 0
    bg .LL12
    sub %o0, %o1, %o0
    b,a .LL8
.LL4:
#BB_3
    b .LL10
    sub %o1, %o0, %o1
.LL8:
#BB_4
    retl
    nop
```

3
3
4
2
2

```
int gcd(int a, int b) {
    int __time = 3;
    while (b>0) {
        __time += 3;
        if (a>b) {
            __time += 4;
            a = a - b;
        }
        else {
            __time += 2;
            b = b - a;
        }
    }
    __time += 2;
    assert(__time<=-1);
    return a;
}
```

Exercício sobre WCET

- Calcule o WCET do código abaixo considerando que as operações de lock e unlock consomem 4 u.t., a operação de incremento e comparação consomem 2 u.t. e 1 u.t. respectivamente

```
void *thread_A(void *arg) {  
    pthread_mutex_lock(&mutex);  
    A_count++;  
    if (A_count == 1)  
        pthread_mutex_lock(&lock);  
    pthread_mutex_unlock(&mutex);  
  
    pthread_mutex_lock(&mutex);  
    A_count--;  
    if (A_count == 0)  
        pthread_mutex_unlock(&lock);  
    pthread_mutex_unlock(&mutex);  
}
```

```
void *thread_B(void *arg) {  
    pthread_mutex_lock(&mutex);  
    B_count++;  
    if (B_count == 1)  
        pthread_mutex_lock(&lock);  
    pthread_mutex_unlock(&mutex);  
  
    pthread_mutex_lock(&mutex);  
    B_count--;  
    if (B_count == 0)  
        pthread_mutex_unlock(&lock);  
    pthread_mutex_unlock(&mutex);  
}
```



Round Robin (time slice method)

Round Robin

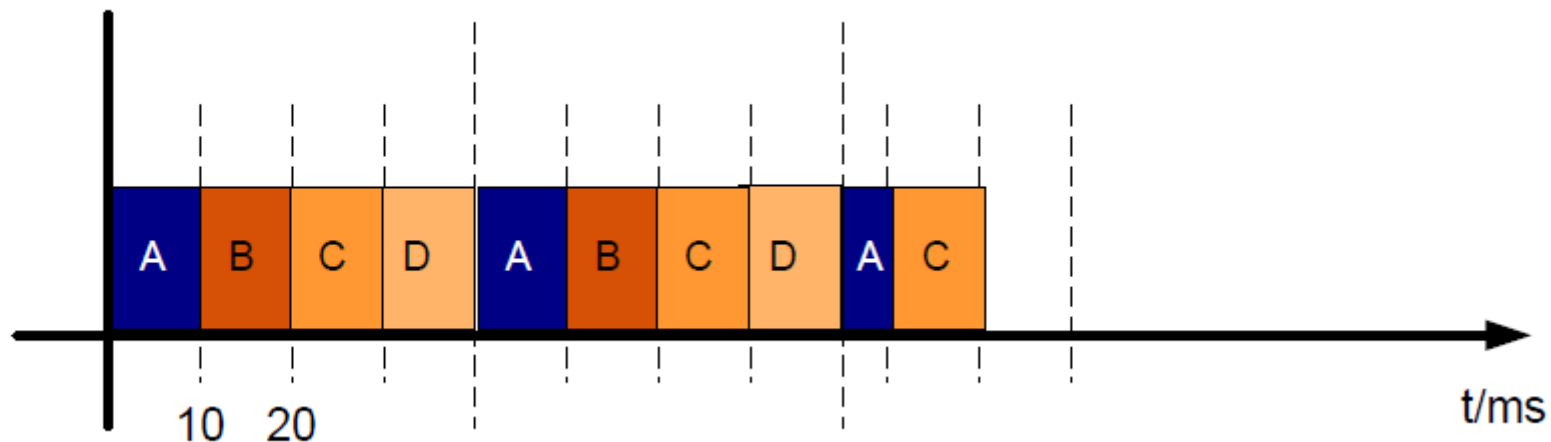


- Cada tarefa tem um determinado *slot* de tempo no qual o processador é alocado
- Sequência é determinada estaticamente
- Execução de uma tarefa “passo a passo”
- Usado em sistemas de diálogo (sistemas multi-tarefa)
- O algoritmo de escalonamento é simples, fácil de implementar e *starvation-free*
- Inadequado para sistemas de tempo real crítico

Exemplo: Round-Robin

- Cada fatia de tempo tem 10ms e as tarefas foram organizadas na seguinte ordem: A-B-C-D
- Tempo de execução:

Tarefa	Período (T) Deadline (D)	Tempo de Execução (C)
A	100	25
B	80	20
C	100	30
D	80	20





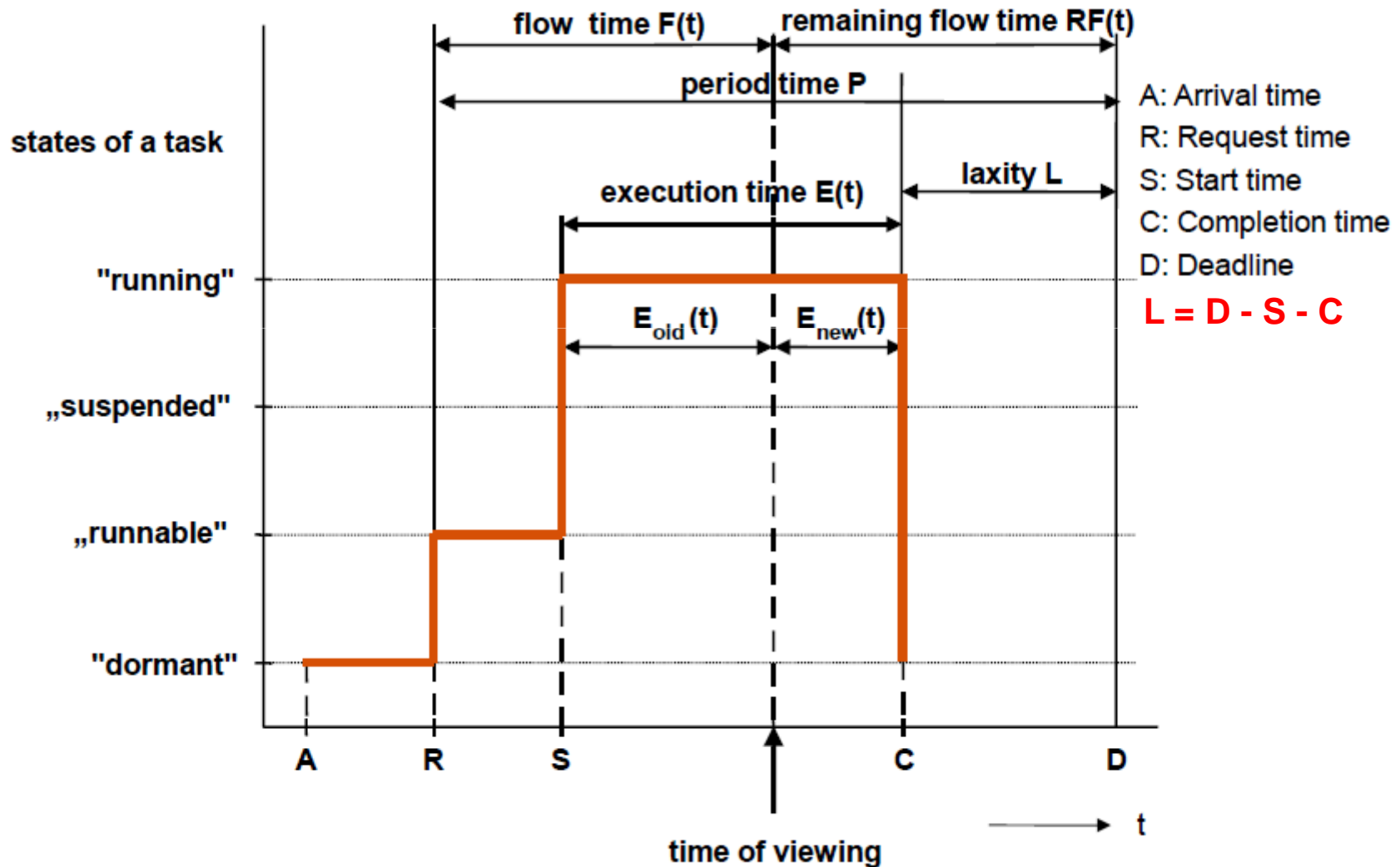
Least Laxity

Least Laxity



- Processor é alocado para tarefa com menor *laxity* (“relaxamento”)
- Considera restrições temporais e tempo de execução
- Método bastante caro computacionalmente
- Adequado para sistemas críticos de tempo real

Parâmetros de Tempo de uma Tarefa

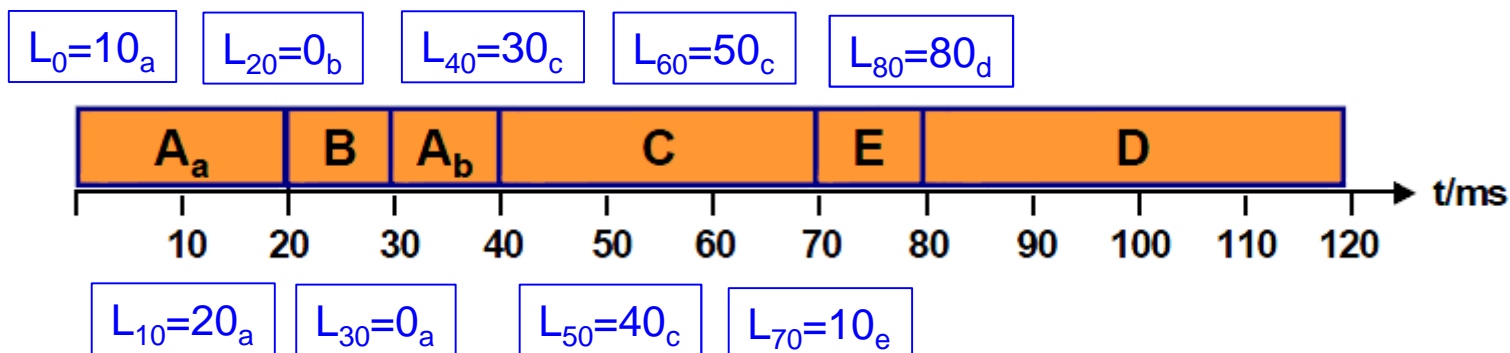


Exemplo: Least Laxity

Tarefa	Tempo de Execução (C)	Ativação (A)	Deadline (D)
A	30	0	40
B	10	0	30
C	30	30	100
D	40	50	200
E	10	70	90

■ Seqüência de execução

$$L = D - S - C$$



Escalonamento para *Power-Aware*

- Dado um conjunto de requisitos de tempo de execução (C), todas as tarefas respeitarão as metas temporais (D)?
 - Assumi um **processador de velocidade fixa**
- Processadores de velocidade variável: Em qual velocidade o processador deve executar para que as tarefas sejam escalonáveis?
- Recursos de velocidade variável são encontrados em aplicações *power-aware*
 - Por exemplo, sistemas que executam com baterias: dispositivos móveis e nodos em uma rede de sensor
- Todos os sistemas baseados em bateria precisam economizar energia para estender o tempo de operação

Economia de Energia em Sist. *Power-Aware*

- Para economizar energia, a **voltagem** do processador é **reduzida** com a desvantagem de ser **mais lento**
 - A economia de energia é **não-linear**
 - Reduzindo pela metade a velocidade de um processador pode quadruplicar o seu tempo de operação
- Alguns processadores possuem **velocidade variável**
 - Outros suportam um conjunto finito de ajustes de velocidade
- O problema da escalonabilidade agora tem dois estágios:
 - Com o processador executando na sua máxima velocidade, o sistema é escalonável? (teste padrão)
 - Se o sistema é escalonável, qual é o máximo k que podem aumentar todos o valores de C tal que o sist. continue escalonável?

Exemplo Escalonamento para *Power-Aware*

- Considere o exemplo de prioridade fixa onde os valores de C são apropriados para a velocidade máxima do processador

Tarefa	Período (T)	Deadline (D)	Tempo de Execução (C)
a	70	70	5
b	120	100	7
c	200	200	11

- Este conjunto de tarefas é escalonável?
 - Tempos de Resposta: $R_a=5$, $R_b=12$ e $R_c=23$
- Se for atribuído o valor 10 a k , o sistema ainda é escalonável?
 - Claramente não, pois a utilização do processador é maior que 1

Exemplo Escalonamento para *Power-Aware*

- Se for atribuído o valor 5 a k , o sistema ainda é escalonável?

Tarefa	Período (T)	Deadline (D)	Tempo de Execução (C)	Tempo de Resposta (R)
a	70	70	25	25
b	120	100	35	60
c	200	200	55	200

- Qualquer aumento no parâmetro C , causará uma perda de *deadline* da tarefa c
 - $k=5$ é o valor ótimo
- Conclui-se que o conjunto de tarefas é escalonável em um processador com velocidade $Max/5$