



Laboratório de Pesquisa em Redes e Multimídia

Sistemas Operacionais

Processos - algoritmos de escalonamento



Universidade Federal do Espírito Santo
Departamento de Informática

Escalonamento da UCP

- O objetivo do escalonamento da UCP ("*processor scheduling*") é designar os processos a serem executados pela(s) UCP(s) ao longo do tempo, tendo como base os objetivos definidos para o sistema operacional e sua política de escalonamento.
- O escalonador divide o tempo da UCP entre os diversos processos ativos no sistema (processos de usuário e processos de sistema).

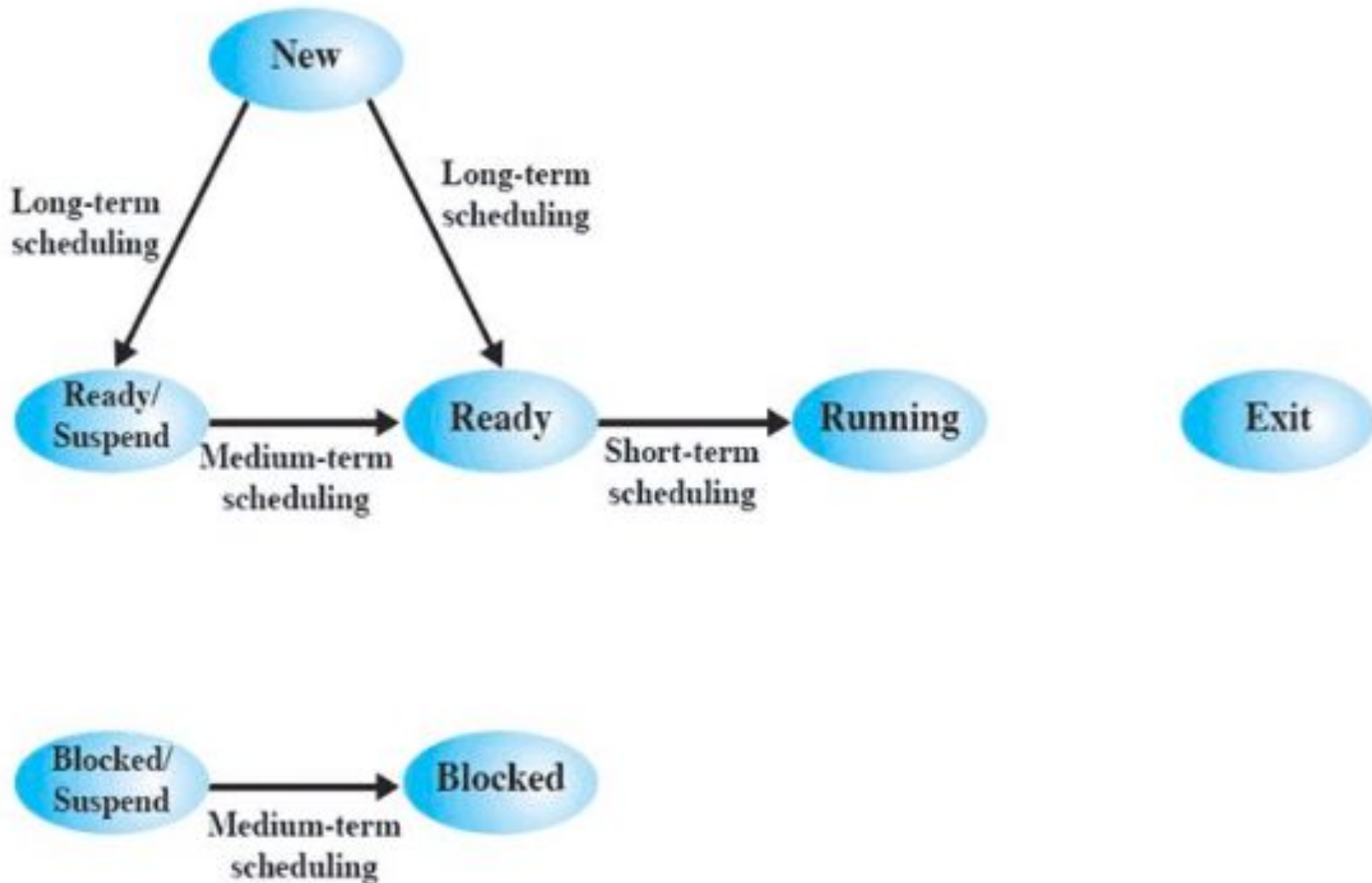
Tipos de Escalonamento

■ Três tipos básicos:

- Escalonamento de longo prazo ("*long-term scheduling*"): adiciona processos ao pool de processos a serem executados;
- Escalonamento de médio prazo ("*medium-term scheduling*"): controla o *swap in* e *swap out* de processos, atendendo às flutuações de carga do sistema.
- Escalonamento de curto prazo ("*short-term scheduling*" ou "*dispatcher*"): responsável pela troca de contexto e alocação efetiva da UCP ao processo selecionado.

Obs: OS também implementa Escalonamento de E/S ("I/O scheduling"): seleciona o processo a obter a posse de dispositivo de I/O.

Escalonamento e a Transição de Estados



Escalonador de Curto Prazo ⁽¹⁾

- Denominado Escalonador da UCP
 - *ou Dispatcher*, ou *CPU Scheduler*
- Seleciona qual processo deve ser executado a seguir (*ready* → *running*).
- É invocado **muito freqüentemente** (ordem de milisegundos).
 - Deve, portanto, ser rápido/eficiente.

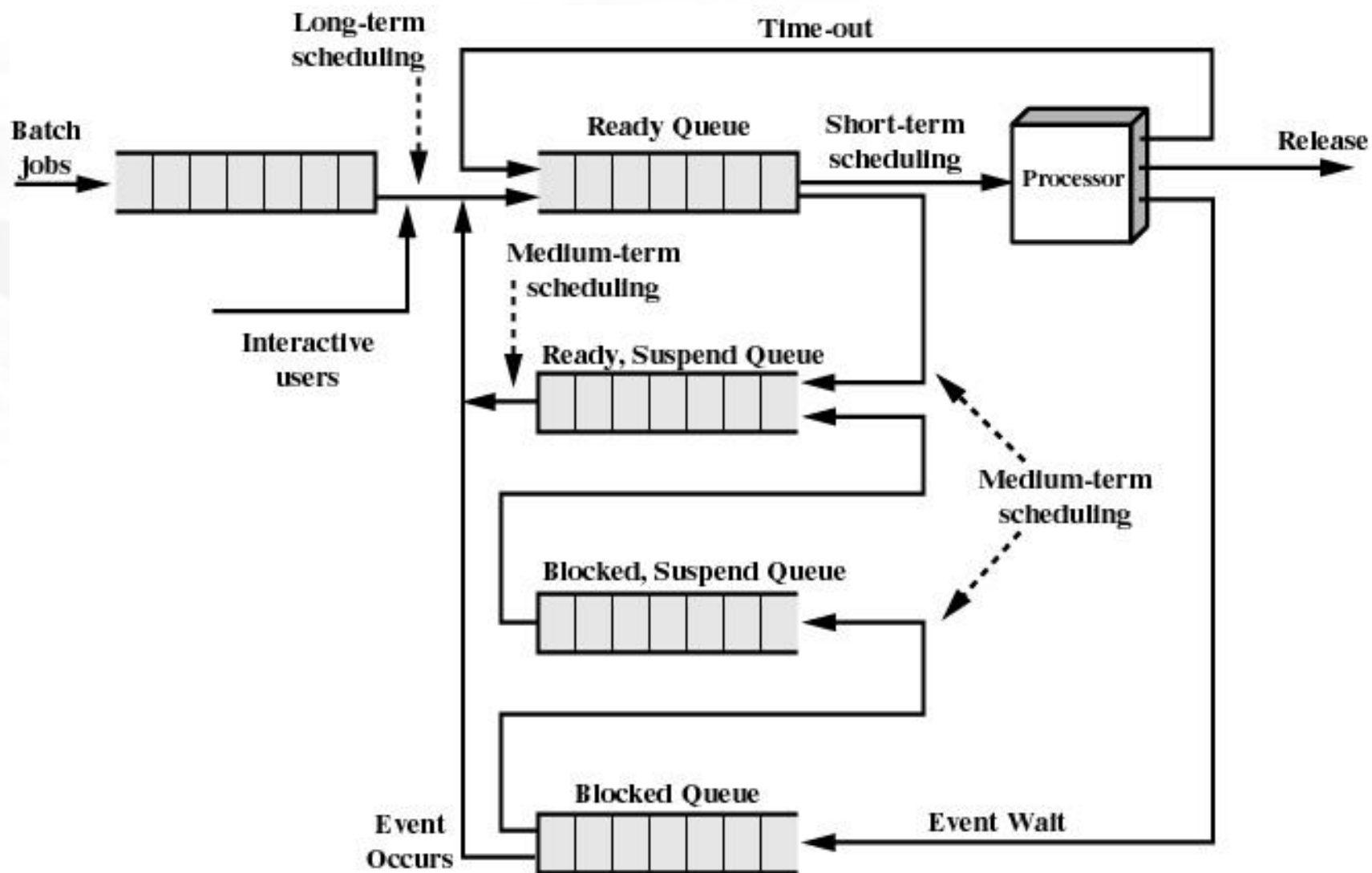
Escalonador de Longo Prazo

- Originalmente conhecido como Escalonador de *Jobs* (“*Job Scheduler*”).
 - Foi criado inicialmente em Sistemas *Batch*
- Seleciona quais processos devem ser levados para a fila de prontos (*new* → *ready*).
- Baixa frequência de invocação (ordem de segundos ou minutos).
- Permite o controle da carga no sistema (controla o grau de multiprogramação... i.e. a quantidade de processos ativos no sistema).

Escalonador de Médio Prazo

- Utiliza a técnica de *swapping*.
 - **Swap out:** o processo é suspenso e o seu código e dados são temporariamente copiados para o disco.
 - **Swap in:** o processo é copiado de volta do disco para a memória; sua execução será retomada do ponto onde parou.
- Está intimamente ligado à **gerência de memória**.

Escalonamento e as Filas do Sistema



Tipos de Escalonadores (Resumo)

Long-term scheduling	The decision to add to the pool of processes to be executed
Medium-term scheduling	The decision to add to the number of processes that are partially or fully in main memory
Short-term scheduling	The decision as to which available process will be executed by the processor
I/O scheduling	The decision as to which process's pending I/O request shall be handled by an available I/O device

Critérios de Escalonamento

- Maximizar a **taxa de utilização da UCP**.
- Maximizar a vazão ("throughput") do sistema.
- Minimizar o **tempo de execução** ("turnaround").
 - *Turnaround*: tempo total para executar um processo.
- Minimizar o **tempo de espera** ("waiting time):"
 - *Waiting time*: tempo de espera na fila de prontos.
- Minimizar o **tempo de resposta** ("response time").
 - *Response time*: tempo entre requisição e resposta (processos interativos).

Requisitos normalmente contraditórios!

Políticas de Escalonamento

■ Preemptivas:

- A UCP pode ser tomada do processo em execução **a qualquer momento**, ou seja, o processo de posse da UCP pode perdê-la compulsoriamente na ocorrência de certos eventos, como fim de fatia de tempo, processo mais prioritário torna-se pronto para execução, etc.
- Não permite a monopolização da UCP.

■ Não-Preemptivas:

- O processo em execução **só perde a posse da UCP caso termine ou a devolva deliberadamente**, isto é, uma vez no estado *running*, ele só muda de estado caso conclua a sua execução ou bloqueie a si mesmo (ex. chamada de sistema para fazer uma operação de E/S.)

Algoritmos de Escalonamento ⁽¹⁾

- Os critérios mostrados (slide 10) podem ser contraditórios
 - Impossível atender a todos eles simultaneamente
- Com isso... os algoritmos buscam:
 - Obter **bons tempos médios** ao invés de maximizar ou minimizar um determinado critério.
 - Privilegiar a variância em relação a tempos médios, i.e., garantir níveis de serviço previsíveis.
- As **Políticas de Escalonamento** podem combinar diferentes algoritmos.

Exemplos de Algoritmos de Escalonamento

- FIFO (First-In First-Out) ou FCFS (First-Come First-Served)
- SJF (Shortest Job First) ou SPN (Shortest Process Next)
- SRTF (Shortest Remaining Time First)
- Round-Robin
- Priority
- Multiple queue (acaba sendo uma política...)

First-Come First-Served ⁽¹⁾

- Algoritmo de baixa complexidade.
- Exemplo de abordagem **não-preemptiva**.
- Algoritmo:
 - Processos que se tornam aptos para execução são inseridos no final da fila de prontos.
 - O primeiro processo da fila é selecionado para execução.
 - O processo executa até que:
 - Termina a sua execução;
 - Realiza uma chamada ao sistema.

First-Come First-Served ⁽²⁾

- **Processos pequenos** podem ter que **esperar** por muito tempo, atrás de processos longos, até que possam ser executados ("*convoy effect*").
- **Favorece** processos **CPU-bound**.
 - Processos I/O-bound têm que esperar muito até que processos CPU-bound terminem a sua execução.
- Algoritmo particularmente **problemático** para sistemas de **tempo compartilhado**, em que os usuários precisam da CPU a intervalos regulares.

... Muito usado em sistemas Batch (sem processos interativos)

Processo	Tempo Exec.
P1	24s
P2	3s
P3	3s

First-Come First-Served (3)



Este é um
“Diagrama
de Gantt”
(simplificado)

- Todos ingressaram na fila de prontos no instante 0s, mas nessa ordem: P_1 , P_2 , P_3
- Tempo de espera para cada processo:
 - **Waiting time:** $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Tempo médio de espera:
 - **Average waiting time (T_w):** $(0 + 24 + 27)/3 = 17$

First-Come First-Served ⁽⁴⁾

Processo	Tempo Exec.
P1	24s
P2	3s
P3	3s

... e se trocássemos a ordem de chegada?!



- Suponha que os mesmos processos cheguem agora na seguinte ordem: P_2, P_3, P_1
- Tempo de espera de cada processo:
 - **Waiting time:** $P_1 = \text{---}; P_2 = \text{---}; P_3 = \text{---}$
- Tempo médio de espera:
 - **Average waiting time (T_w):** _____

First-Come First-Served ⁽⁵⁾

Outro Exemplo... (solução no prox. slide)

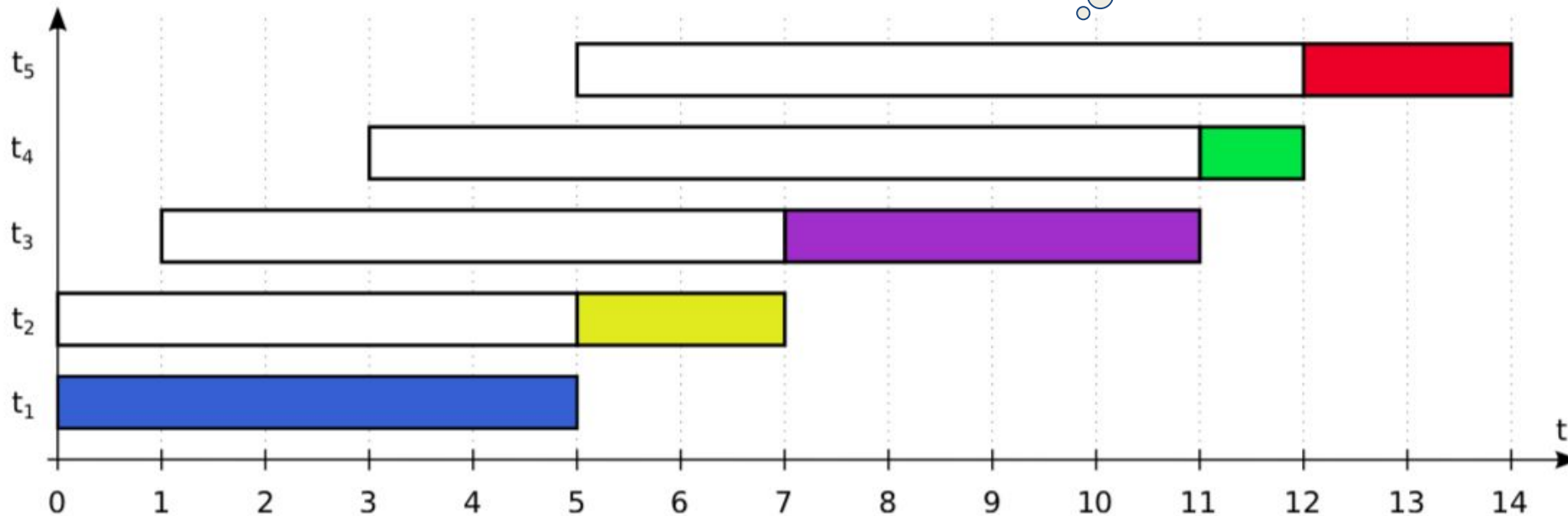
	Ingresso	Duração	Prioridade
t1	0	5	2
t2	0	2	3
t3	1	4	1
t4	3	1	4
t5	5	2	5

Qual é o tempo médio de execução ("*turnaround*") e o tempo médio de espera na fila de prontos?

ATENÇÃO!! Apesar da *prioridade* estar sendo fornecida na tabela, reparem que para o algoritmo FCFS/FIFO, ela não é utilizada!!

Este é um
“Diagrama
de Gantt”
(tradicional)

First-Come First-Served (6)



Ingresso Duração Prioridade

t1	0	5	2
t2	0	2	3
t3	1	4	1
t4	3	1	4
t5	5	2	5

$$Tt = [(5-0)+(7-0)+(11-1)+(12-3)+(14-5)]/5=8,0s$$

$$Tw=[(0-0)+(5-0)+(7-1)+(11-3)+(12-5)]/5=5,2s$$

- **Tt** é a média dos tempos de execução (**turnaround**) dos processos.
- O tempo de execução de cada processo é: tempo de término - tempo de ingresso/chegada na fila de prontos.
- **Tw** é o tempo médio de espera
- O tempo de espera é o tempo que um processo passa no estado “ready” (fila de prontos)

Shortest Job First - SJF ⁽¹⁾

- Baseia-se no fato de que privilegiando processos **pequenos** o **tempo médio de espera decresce**
- O tempo de espera dos processos pequenos decresce mais do que o aumento do tempo de espera dos processos longos.

... vejam slide 16 vs. slide 17

- É um algoritmo **ótimo**, de referência.

... muito difícil de ser implementado!!

Shortest Job First - SJF ⁽²⁾

- Abordagem 1:

- Processo com menor **expectativa** de tempo de processamento total é selecionado para execução.

... esta abordagem faz mais sentido para sistemas Batch

- Abordagem 2:

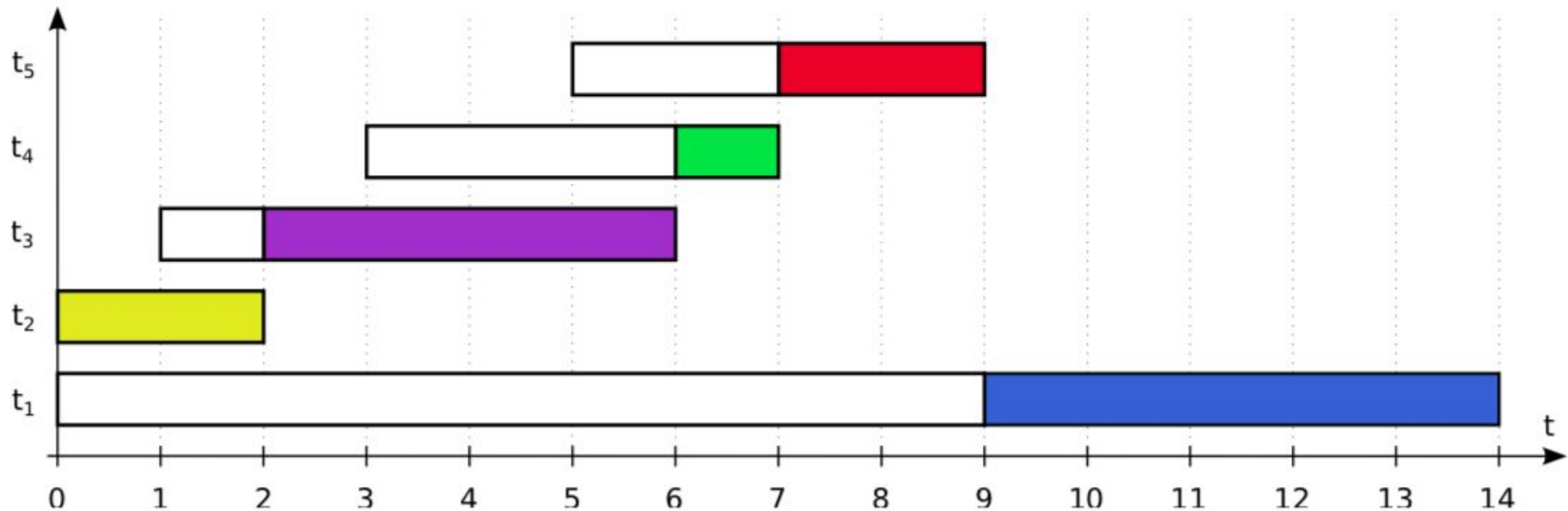
- Associado com cada processo está o tamanho do seu próximo **CPU burst**.
- Esse tamanho é usado como critério de escalonamento, sendo selecionado o processo de menor próximo **CPU burst**.

... o problema é justamente como saber qual é o tamanho do próximo CPU burst

Shortest Job First - SJF₍₃₎

- Dois esquemas:
 - **Não-preemptivo** – uma vez a CPU alocada a um processo ela não pode ser dada a um outro antes do término do CPU burst corrente.
 - **Preemptivo** – se chega um novo processo com CPU burst menor que o tempo remanescente do processo corrente ocorre a preempção. Esse esquema é conhecido como *Shortest-Remaining-Time-First (SRTF)*.

SJF (Não-Preemptivo) - Exemplo



Ingresso Duração Prioridade

t1	0	5	2
t2	0	2	3
t3	1	4	1
t4	3	1	4
t5	5	2	5

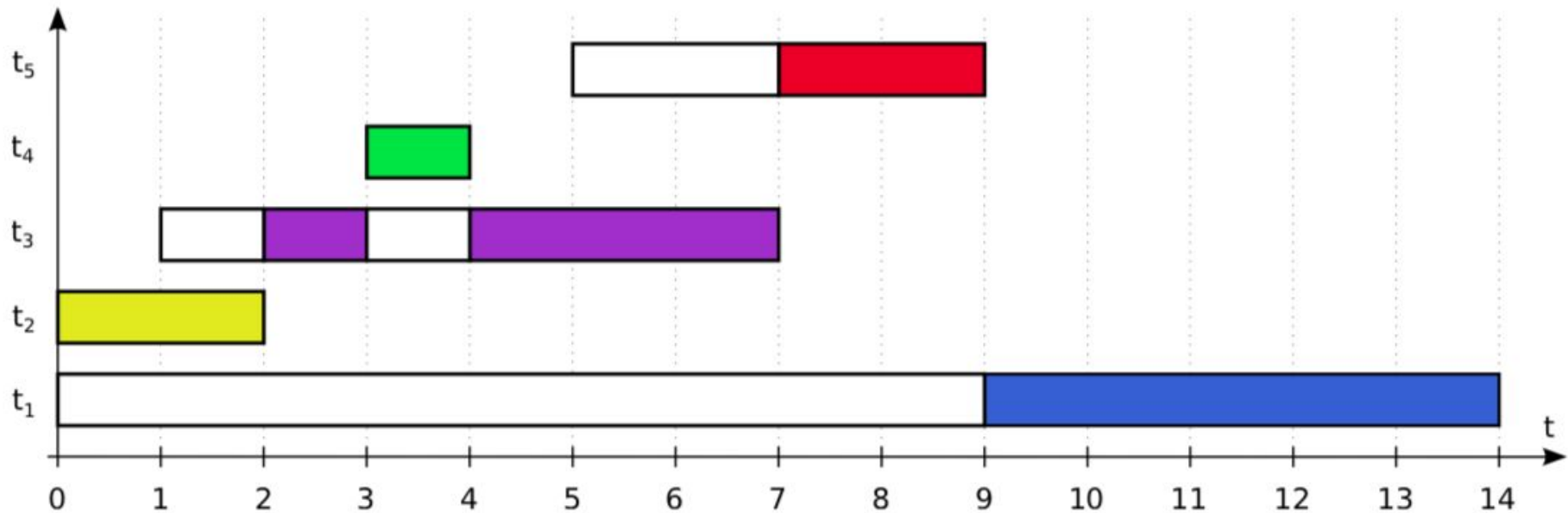
$$Tt = (14 + 2 + 5 + 4 + 4) / 5 = 29 / 5 = 5,8s$$

$$Tw = (9 + 0 + 1 + 3 + 2) / 5 = 15 / 5 = 3,0s$$

- **Tt** é a média dos tempos de execução (**turnaround**) dos processos.
- O tempo de execução de cada processo é: tempo de término - tempo de ingresso/chegada na fila de prontos.
- **Tw** é o tempo médio de espera
- O tempo de espera é o tempo que um processo passa no estado "ready" (fila de prontos)

... prioridade NÃO é utilizada no SJF!!

SJF Preemptivo (ou SRTF)



	Ingresso	Duração	Prioridade
t1	0	5	2
t2	0	2	3
t3	1	4	1
t4	3	1	4
t5	5	2	5

$$T_t = (14 + 2 + 6 + 1 + 4) / 5 = 27 / 5 = 5,4s$$

$$T_w = (9 + 0 + 2 + 0 + 2) / 5 = 13 / 5 = 2,6s$$

Observem que o tempo de espera da tarefa t3 é 2s pois ela passou 2 vezes pela fila de prontos, ficando 1s em cada vez

... prioridade NÃO é utilizada no SRTF!!

Previendo o Tamanho do *Next CPU Burst* ⁽¹⁾

- A real dificuldade do algoritmo é conhecer o tamanho da próxima requisição de CPU.
- Para escalonamento de longo prazo num sistema batch, podemos usar como tamanho o limite de tempo de CPU especificado pelo usuário quando da submissão do job.
- No nível de escalonamento de curto prazo sua implementação pode ser apenas aproximada, já que não há como saber o tamanho da próxima requisição de CPU.

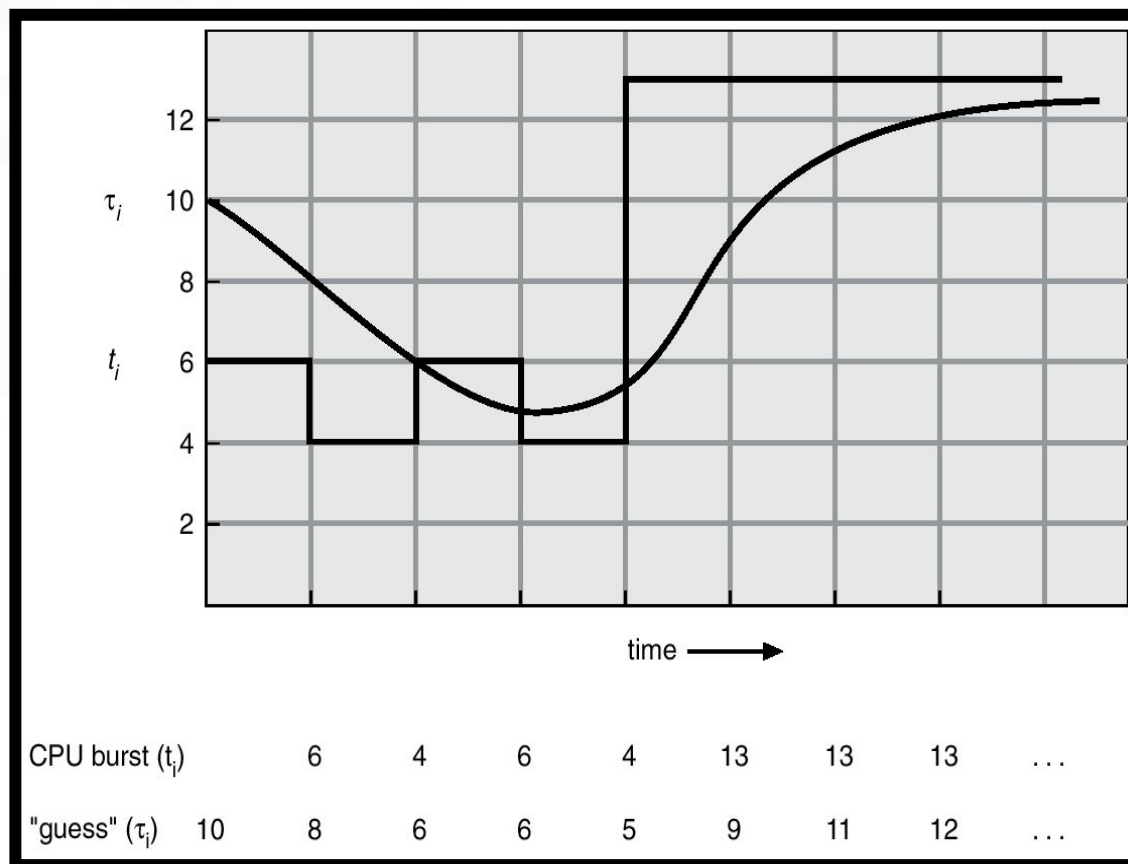
Prevendo o Tamanho do *Next CPU Burst* (2)

$$\alpha = 1/2$$

$$\tau_0 = 10$$

$$\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n$$

1. t_n = actual length of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. α , $0 \leq \alpha \leq 1$



Escalonamento por Prioridade ⁽¹⁾

- Um número inteiro é associado a cada processo, refletindo a sua prioridade no sistema.
- A CPU é alocada ao processo de maior valor de prioridade na fila de prontos.

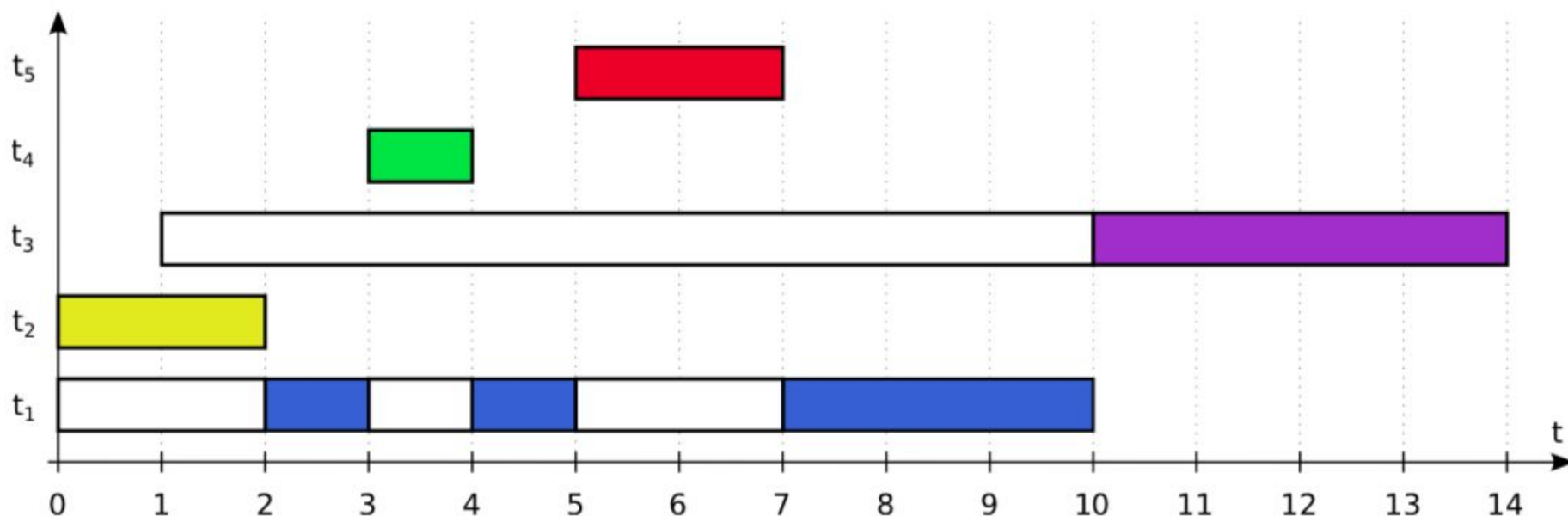
Obs: normalmente no UNIX, é usada uma escala negativa/invertida, i.e., menor valor = maior prioridade (a maior prioridade é 0)

- Estratégia muito usada em S.O. de tempo real.

Escalonamento por Prioridade (2)

- Prioridades podem ser definidas interna ou externamente.
 - Definição **interna**:
 - Usa **alguma medida** (ou uma combinação delas) para computar o valor da prioridade. Por exemplo, limite de tempo, requisitos de memória, nº de arquivos abertos, razão entre *average I/O burst* e *average CPU burst*, etc.
 - Definição **externa**:
 - Definida por algum **critério externo** ao S.O (tipo do processo, departamento responsável, custo, etc.)
- **Problema: "starvation"**
 - Prioridades **estáticas** pode levar um processo de baixa prioridade a nunca executar
- Solução: "envelhecimento" ("**aging**")
 - Prioridade dinâmica... em que **a Prioridade aumenta com o passar do tempo.**

Prioridade Estática (1)



Ingresso Duração Prioridade

t1	0	5	2
t2	0	2	3
t3	1	4	1
t4	3	1	4
t5	5	2	5

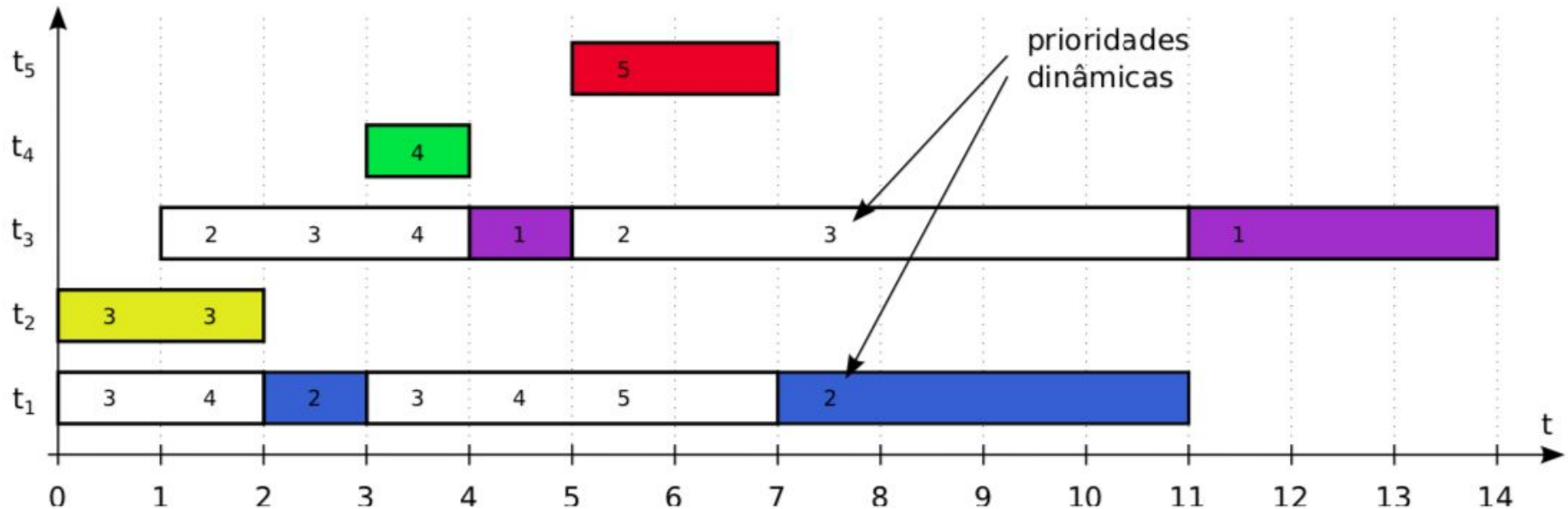
$$Tt = (10 + 2 + 13 + 1 + 2) / 5 = 28 / 5 = 5,6s$$

$$Tw = (5 + 0 + 9 + 0 + 0) / 5 = 14 / 5 = 2,8s$$

OBS: neste exemplo qto maior o número maior a prioridade.

Prioridade Dinâmica com "Aging"

(livro do Maziero, pg.79-80)



	Ingresso	Duração	Prioridade
t1	0	5	2
t2	0	2	3
t3	1	4	1
t4	3	1	4
t5	5	2	5

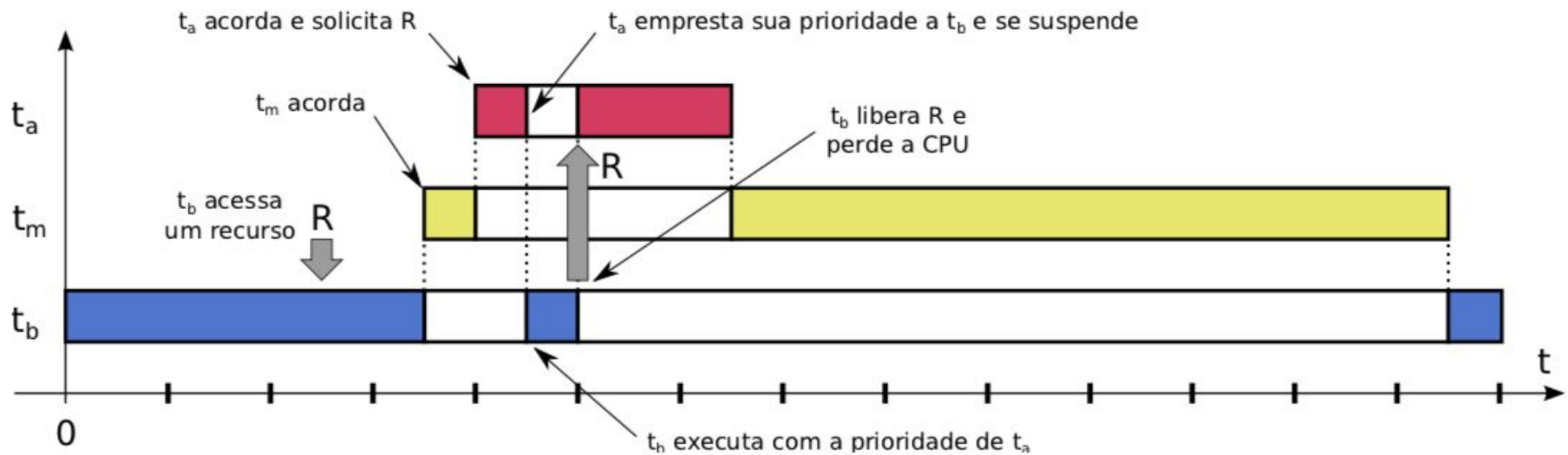
$$T_t = (10 + 2 + 13 + 1 + 2) / 5 = 28 / 5 = 5,6s$$

$$T_w = (5 + 0 + 9 + 0 + 0) / 5 = 14 / 5 = 2,8s$$

Problema da "Inversão de Prioridades" ⁽¹⁾

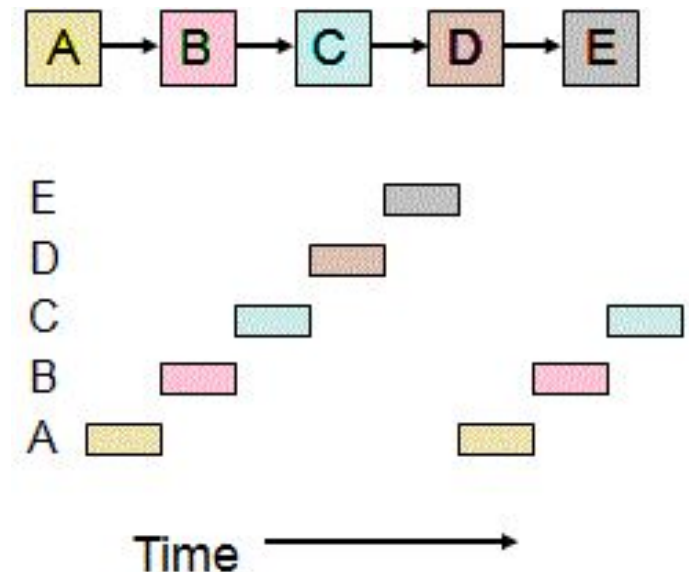


O Protocolo de Herança de Prioridade (PHP) (1)



Escalonamento Circular ("*Round-Robin*") ⁽¹⁾

- Algoritmo **típico** de sistemas operacionais de **tempo compartilhado**.
- Cada processo recebe uma pequena **fatia de tempo** de CPU (**quantum**)
 - Usualmente entre 10 e 100 ms.
- Após o término da sua fatia de tempo o processo é "**interrompido**" (**preemptado!**) e colocado no final da fila de prontos.

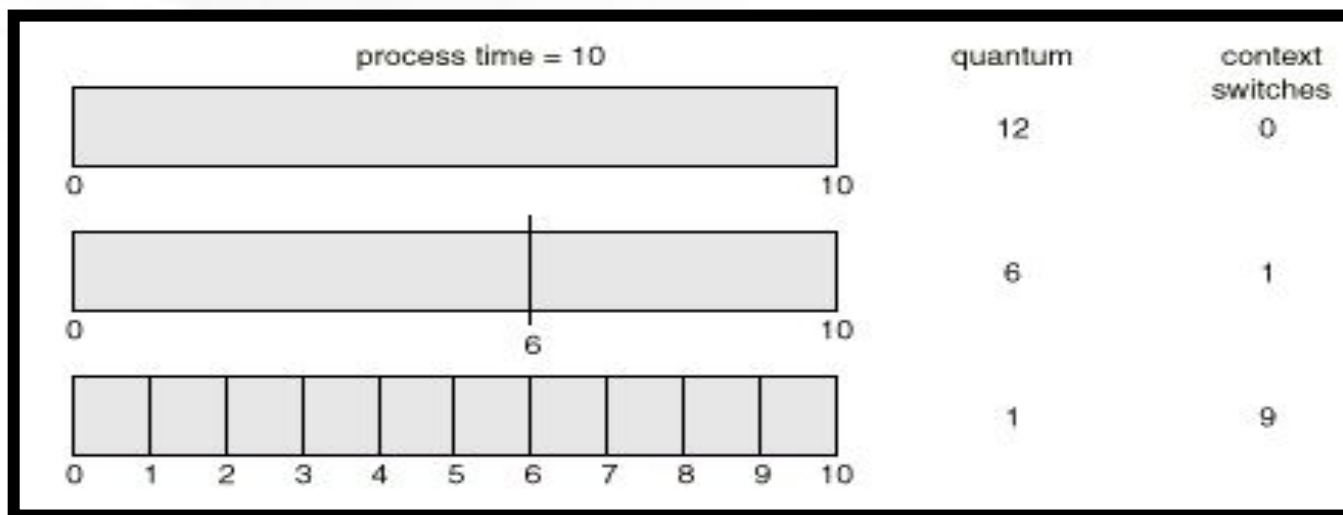


Escalonamento Circular ("*Round-Robin*") (2)

- Se **n processos** existem na fila de prontos e a **fatia de tempo é q** , então cada processo recebe **$1/n$ do tempo de CPU**, em fatias de q unidades de tempo de cada vez.
 - **Nenhum processo** espera mais do que **$(n-1).q$** unidades de tempo.
 - Qual a relação c/ o **tempo de resposta**?
 - Tipicamente, apresenta um tempo de ***turnaround* médio maior** que o SJF, por que?
- É um algoritmo justo???**

Escalonamento Circular ("*Round-Robin*") ⁽³⁾

- Dependente do tamanho do quantum:
 - **q grande** \Rightarrow tende a FIFO.
 - **q pequeno** \Rightarrow gera muito *overhead* devido às trocas de contexto
- Regra geral: **80% CPU burst $< q$**



Escalonamento Circular ("*Round-Robin*") (4)

Ingresso Duração Prioridade

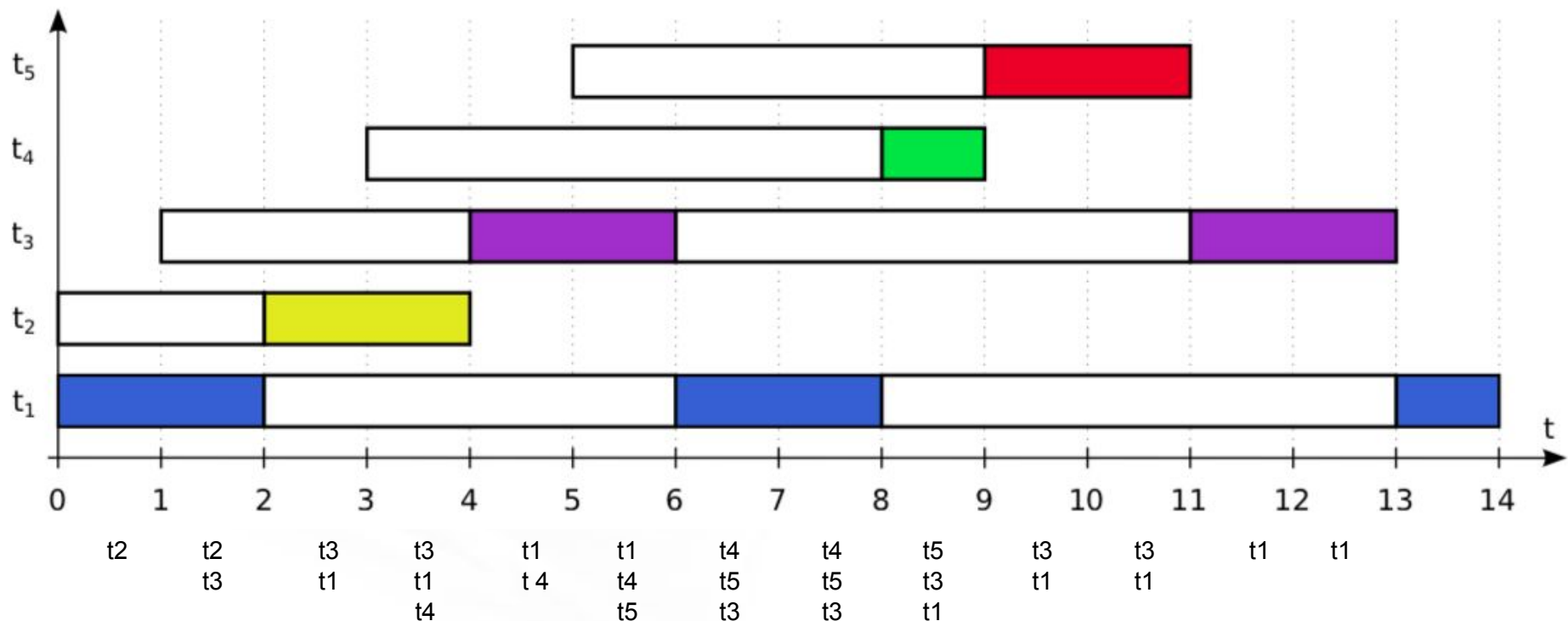
t1	0	5	2
t2	0	2	3
t3	1	4	1
t4	3	1	4
t5	5	2	5

Supondo uma fatia de tempo (*quantum*) = 2, calcular:

Tt = tempo de execução (*turnaround time*)

Tw = tempo de espera na fila de prontos (*waiting time*)

Escalonamento Circular ("*Round-Robin*") ⁽⁵⁾

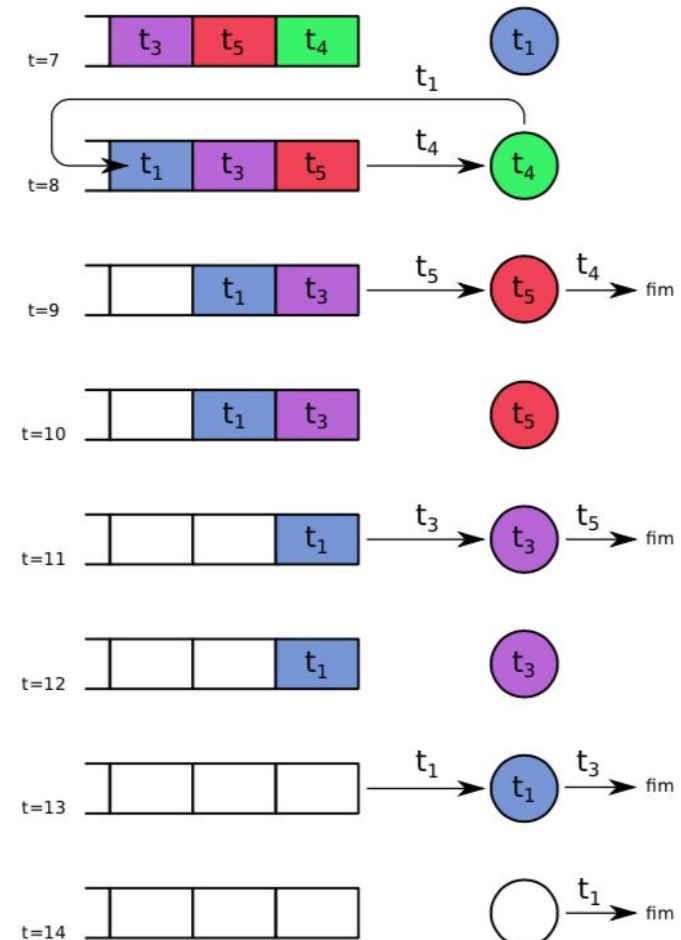
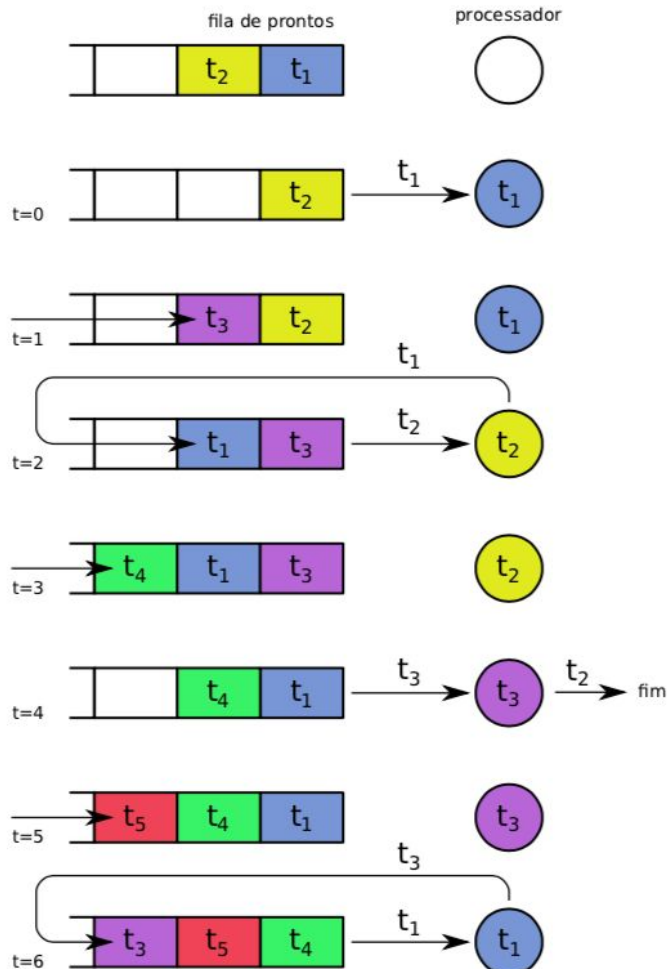


	Ingresso	Duração	Prioridade
t1	0	5	2
t2	0	2	3
t3	1	4	1
t4	3	1	4
t5	5	2	5

$$T_t = [(14-0)+(4-0)+(13-1)+(9-3)+(11-5)]/5 = (14+4+12+6+6)/5 = 42/5 = 8,4s$$

$$T_w = [(4+5)+2+(3+5)+5+4]/5 = (9+2+8+5+4)/5 = 28/5 = 5,6s$$

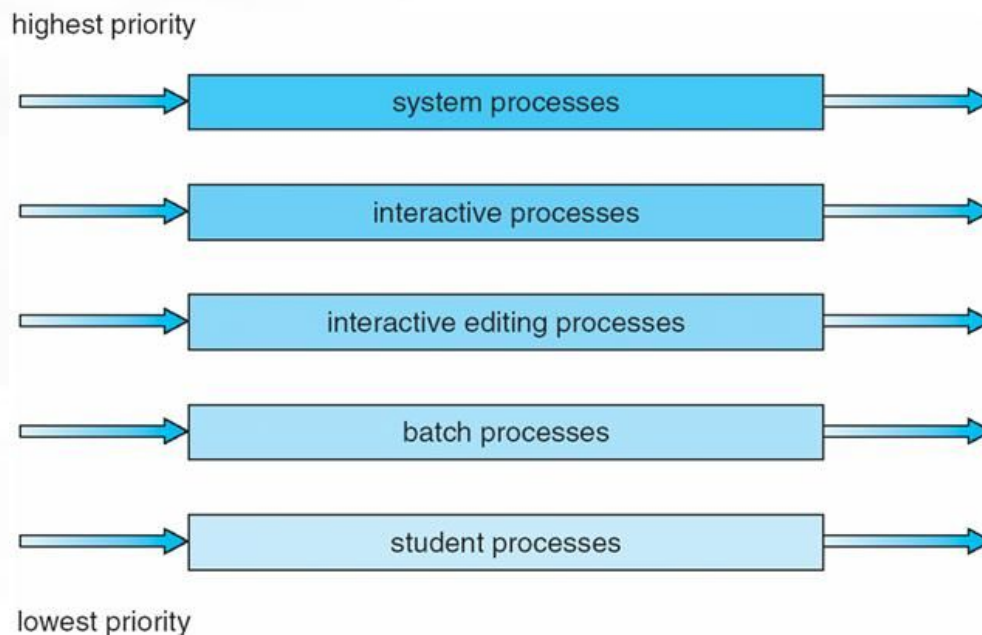
Escalonamento Circular ("*Round-Robin*") (6)



Escalonamento Multinível ⁽¹⁾

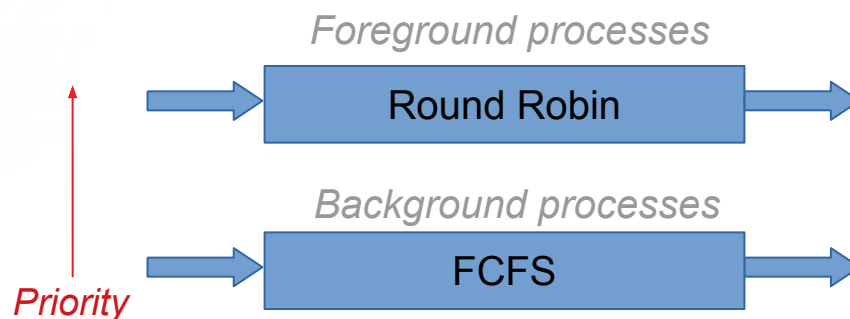
- A idéia base é dividir os processos em **diferentes grupos**, com diferentes requisitos de tempos de resposta.
- A cada grupo é associada uma fila, e **dentro dessa fila** é aplicado um **algoritmo de escalonamento**
- Também deve-se definir um **algoritmo de escalonamento entre as filas**
- Acaba representando uma **POLÍTICA DE ESCALONAMENTO**

EXEMPLO 1



Escalonamento Multinível (2)

- EXEMPLO 2
 - A fila de prontos pode ser dividida em duas filas separadas:
 - *foreground* (p/ processos interativos)
 - *background* (p/ processamento *batch*)
 - Cada fila apresenta o seu próprio algoritmo de escalonamento:
 - *foreground* – RR
 - *background* – FCFS



Escalonamento Multinível ⁽³⁾

- Normalmente, o escalonamento **entre as filas** é implementado usando:
 - **Prioridades fixas** – atende primeiro aos processos da fila *foreground* e somente depois aos da fila *background*.
 - **OU**
 - **Time slice** – cada fila recebe uma quantidade de tempo de CPU para escalonamento entre os seus processos. Ex: 80% para *foreground* em RR e 20% para *background* em FCFS.

Escalonamento Multinível **com Feedback** ⁽¹⁾

- O processo pode se mover entre as várias filas.
- O escalonador trabalha com base nos seguintes parâmetros:
 - Número de filas;
 - Algoritmo de escalonamento de cada fila;
 - Método usado para determinar quando aumentar e quando reduzir a prioridade do processo;
 - Escalonamento entre filas feito com base em **Prioridades**
 - Método usado para se determinar em que fila o processo será inserido.

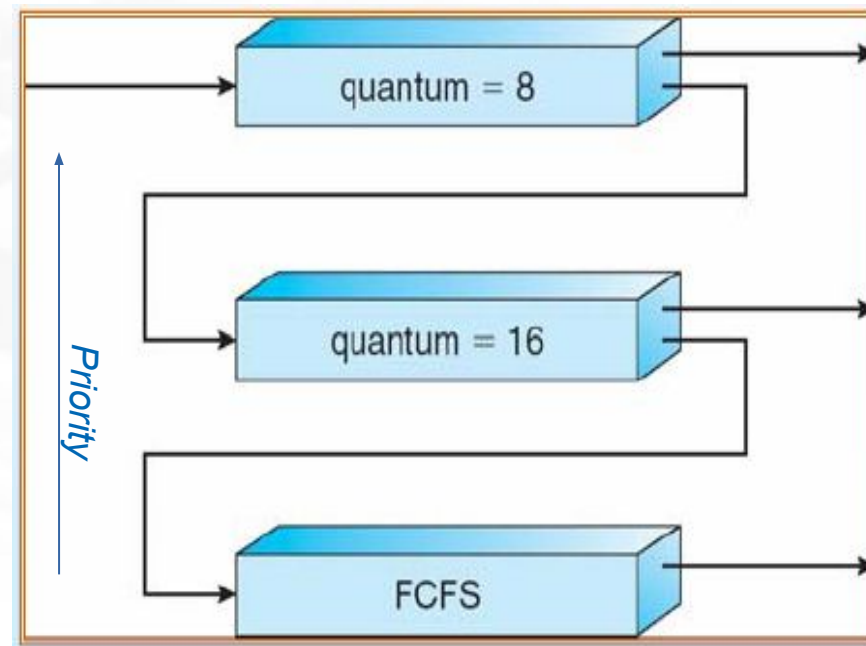
Escalonamento Multinível com Feedback ⁽²⁾

■ EXEMPLO 3:

- Suponha a existência de 3 filas:
 - Q_0 – time quantum 8 milliseconds
 - Q_1 – time quantum 16 milliseconds
 - Q_2 – FCFS
- Escalonamento:
 - Um job novo entra na fila Q_0 , que é servida segundo a estratégia RR. Quando ele ganha a CPU ele recebe 8 ms. Se não terminar em 8 ms, o job é movido para a fila Q_1 .
 - Em Q_1 o job é novamente servido RR e recebe 16 ms adicionais. Se ainda não completar, ele é interrompido e movido para a fila Q_2 .
 - Em Q_2 , FCFS
 - Entre as filas, aplica-se Prioridade

Escalonamento Multinível com Feedback (3)

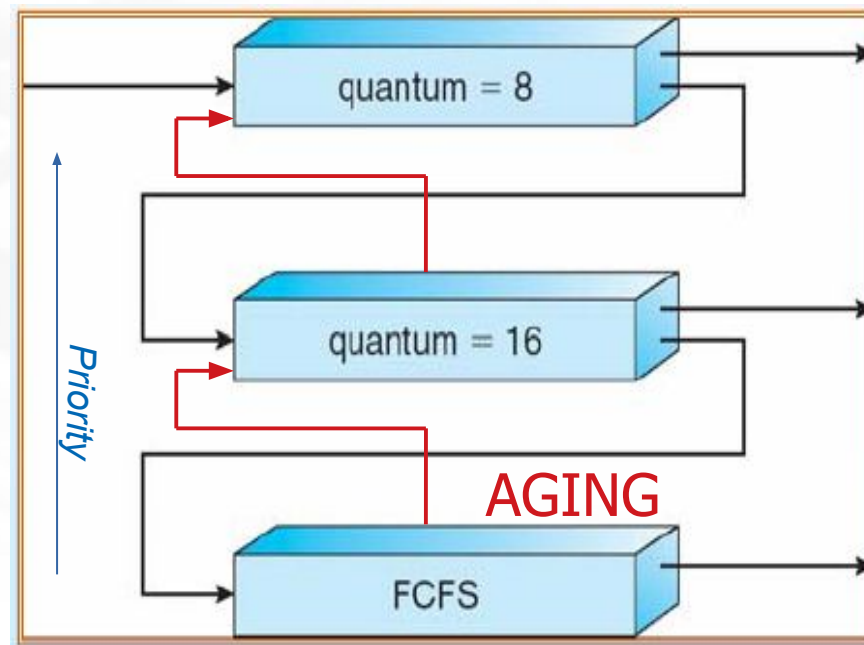
■ EXEMPLO 3 (CONT.)



- E se as filas de cima nunca ficarem vazias???

Escalonamento Multinível com Feedback (4)

■ EXEMPLO 3 (CONT.)

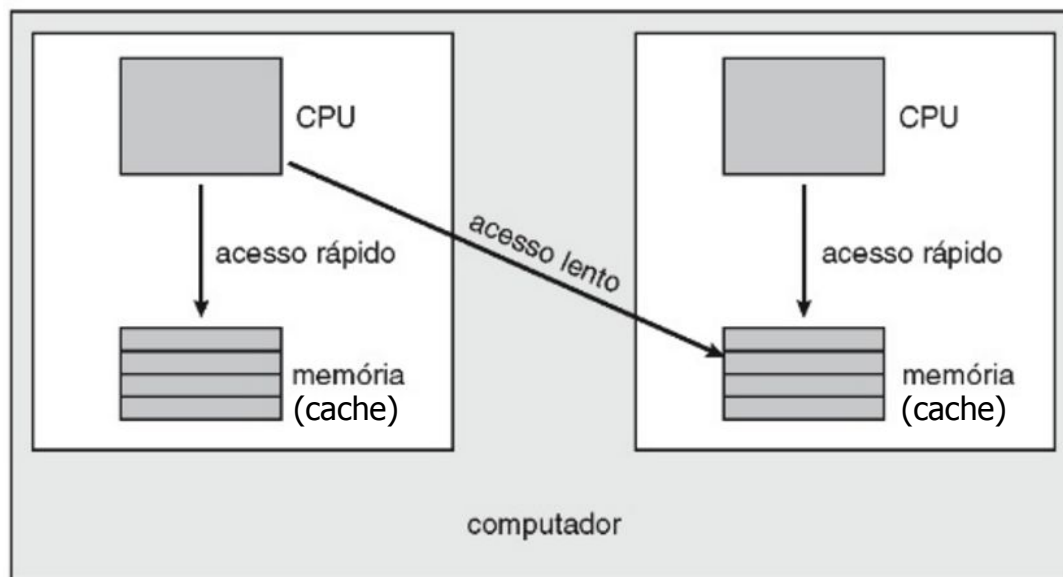


- E se as filas de cima nunca ficarem vazias???
- Starvation!!
- Solução: Usar o feedback pra implementar AGING!

Escalonamento com Multiprocessamento ⁽¹⁾

- No caso de **Asymetric Multi-Processing** (Master-Slave)
 - Todas as decisões de escalonamento são tomadas usando-se uma das CPUs (master)
 - As demais executam apenas programas de usuário.
- No caso de **Symetric Multi-Processing (SMP)**
 - Cada CPU roda código de kernel (escalonador)
 - Pode haver:
 - Uma **fila única** para todos os processadores
 - **Filas diferentes** para processadores diferentes
 - É importante realizar **load balancing**

Escalonamento com Multiprocessamento (2)



■ Processor affinity

- Atualização da Cache é custosa
- Pode ser interessante o processo "ter afinidade" com o processador no qual ele está rodando
- *Soft affinity x hard affinity*

Referências

- C. Maziero, Sistemas Operacionais
 - Capítulo 6 (6.1 a 6.4), p.70-82, Escalonamento de Tarefas. Capítulo 7 (inversão de prioridades)
- A. Sylberschatz, Operating Systems, 6th edition
 - Chapter 6 (6.1 to 6.5), p.151-172, CPU Scheduling.
- W. Stallings, Operating Systems, sixth edition
 - Chapter 9, p.405-418, Uniprocessor Scheduling.
- H. Deitel, Sistemas Operacionais, 3a edição
 - Capítulo 8, p.208-226, Escalonamento de Processador.