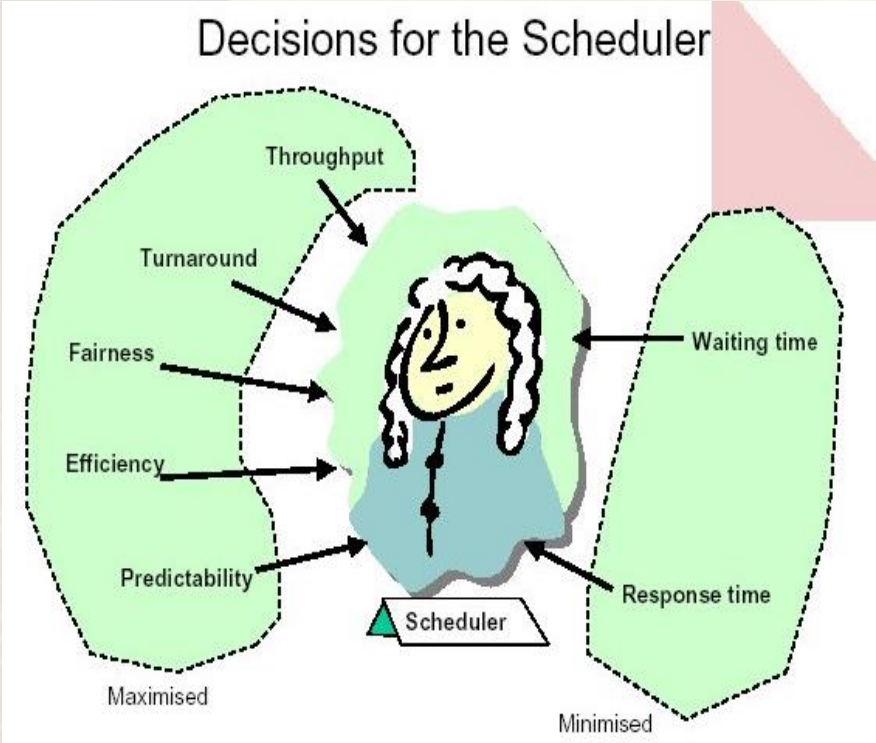
SISTEMAS OPERACIONAIS

Escalonamento de Processos

Objetivos do Escalonamento

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



Algoritmos de Escalonamento

- Os algoritmos buscam:
  - Obter bons tempos médios ao invés de maximizar ou minimizar um determinado critério.
  - Privilegiar a variância em relação a tempos médios.
- As políticas de escalonamento podem combinar diferentes algoritmos
- Algoritmos/políticas podem ser ser:
  - Preemptivas;
  - Não-preemptivas.

Políticas de Escalonamento

- Preemptivas:
  - O processo de posse da UCP pode perdê-la a qualquer momento, 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 emitindo, p.ex., uma operação de E/S.

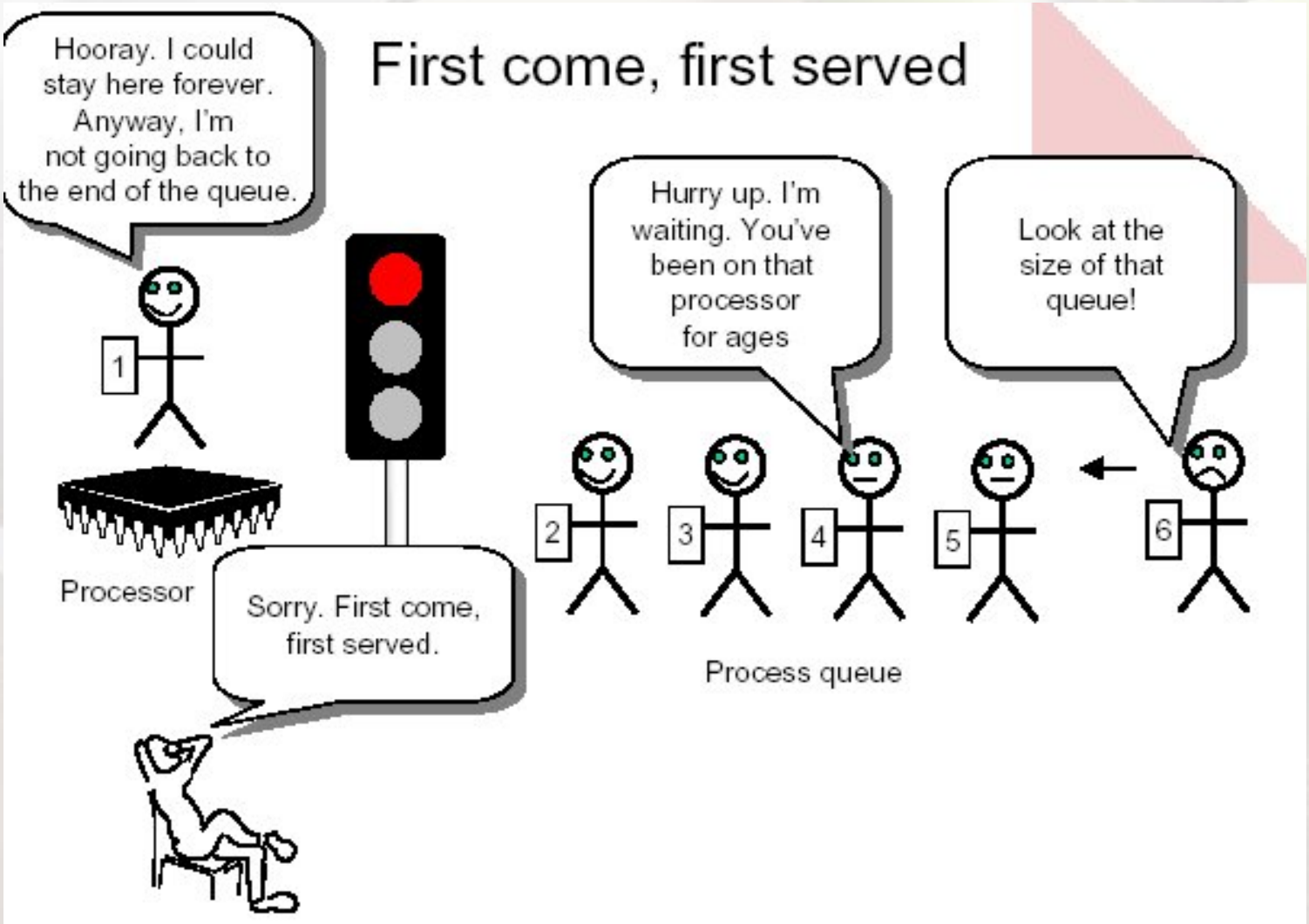
Exemplos de Algoritmos

- 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) ou SRTN (Shortest Remaining Time Next).
- Round-Robin.
- Priority.
- Multiple queue
- Garantido;
- Lottery;
- Fair-Share;

First-Come First-Served (1)

- Algoritmo de baixa complexidade.
- Exemplo de abordagem não-preemptiva.
- 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)



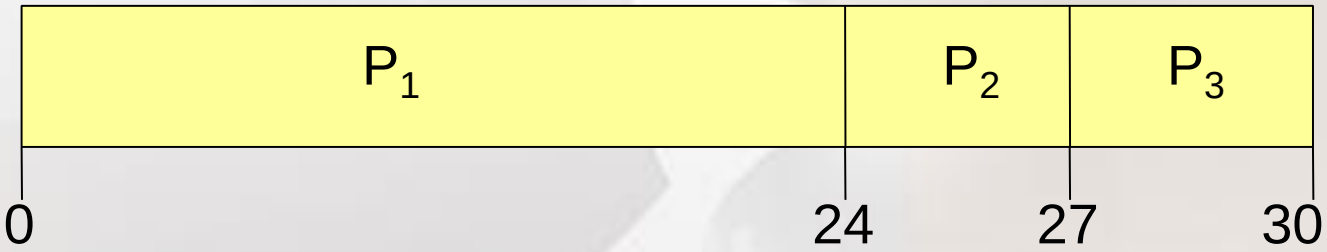
First-Come First-Served (3)

- 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 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.

First-Come First-Served (4)

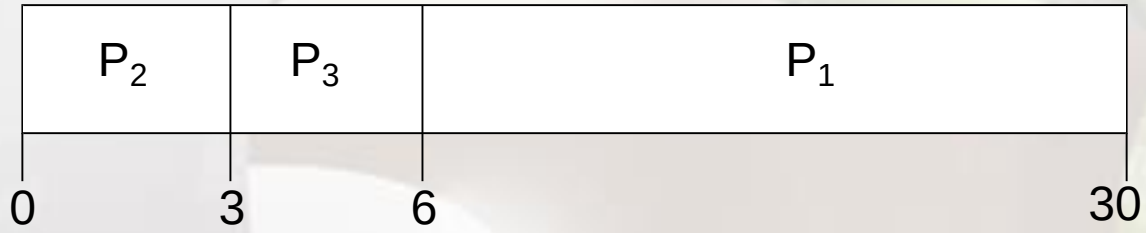
- Suponha que os processos cheguem agora na seguinte ordem:
  - P1, P2, P3

Process	Burst Time
P1	24
P2	3
P3	3



- Tempo de espera de cada processo:
  - Waiting time: P1 = 0; P2 = 24; P3 = 27
- Tempo médio de espera:
  - Average waiting time: : (0 + 24 + 27)/3 = 17
- Tempo médio de término
  - Average turn-around time: (24 + 27 + 30)/3 = 27

First-Come First-Served (4)



- Suponha que os mesmos processos cheguem agora na seguinte ordem:
  - P2, P3, P1
- Tempo de espera de cada processo:
  - Waiting time: P1=6; P2=0; P3=3
- Tempo médio de espera:
  - Average waiting time: (6 + 0 + 3)/3 = 3
- Tempo médio de término
  - Average turn-around time: (3 + 6 + 30)/3 = 13

Shortest Job First (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.
- É um algoritmo ótimo, de referência.

Shortest Job First (2)

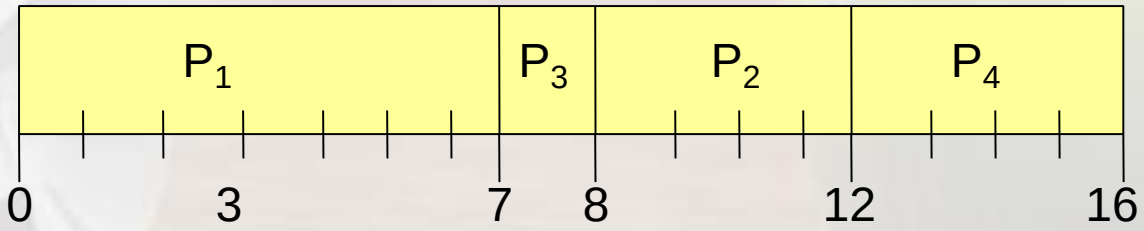
- Abordagem 1:
  - Processo com menor expectativa de tempo de processamento é selecionado para execução.
- Abordagem 2:
  - Associado com cada processo está o tamanho do seu próximo *CPU burst*.
  - Esse tamanho é usado como critério de escalonamento, sendo selecionado o processo de menor próximo *CPU burst*.

Shortest Job First (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).

Exemplo de SJF Não-Preemptivo

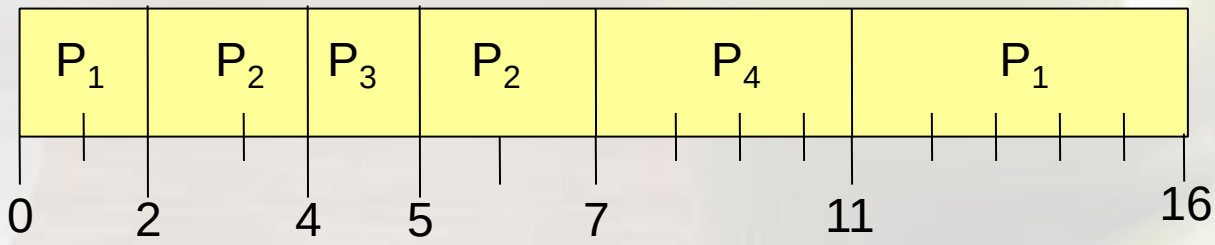
Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



- Average waiting time
  - = ((0 – 0) + (8 – 2) + (7 – 4) + (12 – 5))/4
  - = (0 + 6 + 3 + 7)/4 = 4
- Average turn-around time:
  - = ((7 – 0) + (12 – 2) + (8 - 4) + (16 – 5))/4
  - = (7 + 10 + 4 + 11)/4 = 8

Exemplo de SJF Preemptivo (Algoritmo SRTF)

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



- Average waiting time
  - = ([(0 – 0) + (11 - 2)] + [(2 – 2) + (5 – 4)] + (4 - 4) + (7 – 5))/4
  - = 9 + 1 + 0 + 2)/4 = 3
- Average turn-around time
  - = (16 + 7 + 5 + 11)/4 = 9.75

Tamanho do Próximo CPU burst

- 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 Burst (1)

- Uma maneira de se aproximar do SJF é *prevendo* o tamanho do próximo CPU burst.
  - Solução: realizar uma estimativa com base no comportamento passado e executar o processo cujo tempo de execução estimado seja o menor;
  - Processo de envelhecimento
    - t_0
    - $t_0/2+t_1/2$
    - $t_0/4+t_1/4+t_2/2$
    - $t_0/8+t_1/8+t_2/4+t_3/2$

- Normalmente isso é feito usando uma média exponencial das medidas dos bursts anteriores.
  - τ_n = *actual lenght of* n^{th} *CPU burst*
  - τ_{n+1} = *predicted value for the next CPU burst*
  - $\alpha, 0 <= \alpha <= 1$
  - *Define:*

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

Prevendo o Tamanho do Burst (2)

- O parâmetro α controla o peso relativo do histórico recente e passado na fórmula.
  - Se $\alpha = 0$

    $\tau_{n+1} = \tau_n$

    Histórico recente não é considerado relevante

  - Se $= 1$

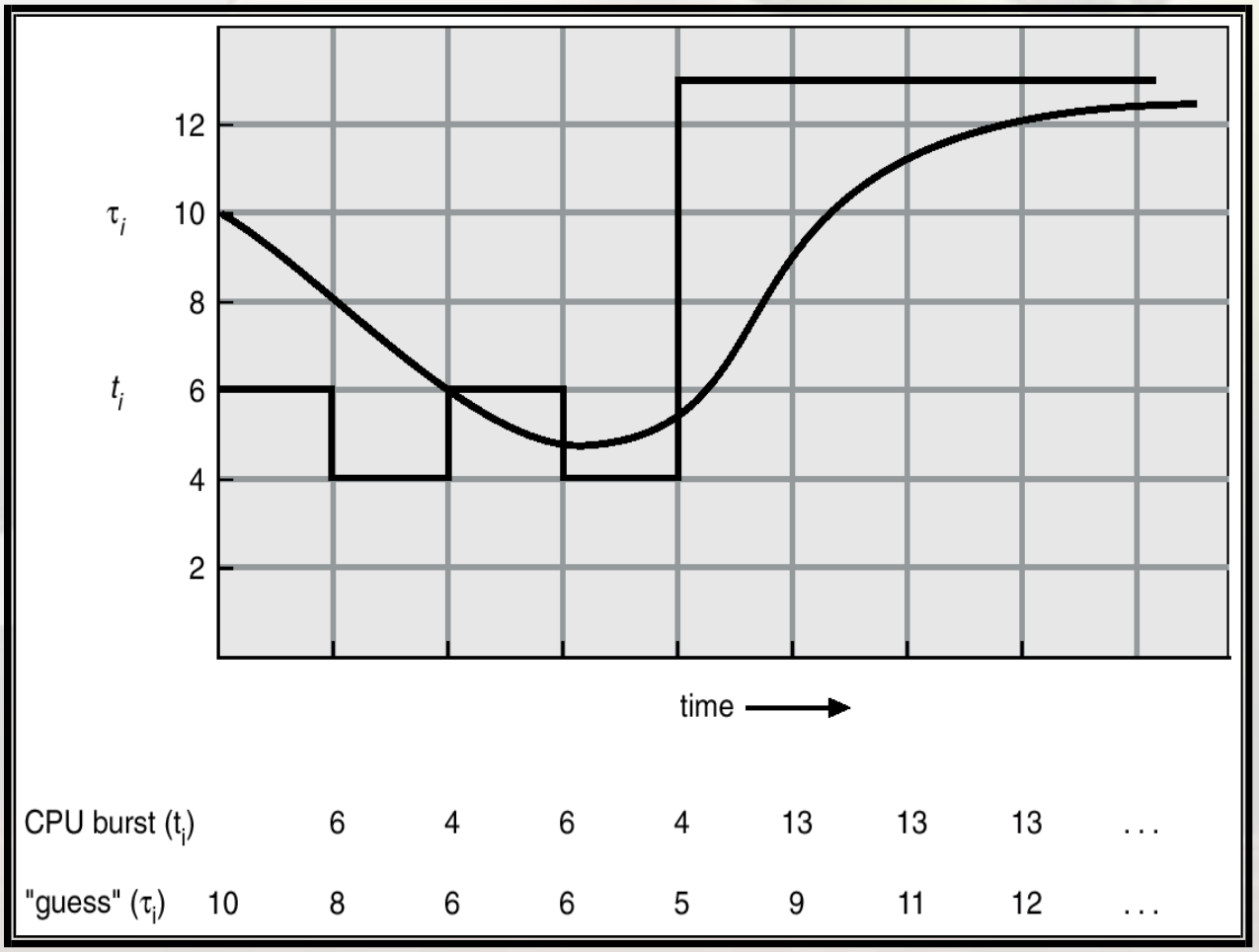
    $\tau_{n+1} = \tau_n$

    Apenas o último CPU burst é levado em conta.

Prevendo o Tamanho do Burst (3)

$\alpha = 1/2$

$\tau_0 = 10$



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, menor valor = maior prioridade
- Estratégia muito usada em S.O. de tempo real.
- Problema: "*starvation*"
  - Processos de baixa prioridade podem nunca executar
- Solução: "*Aging*"
  - Prioridade aumenta com o passar do tempo.

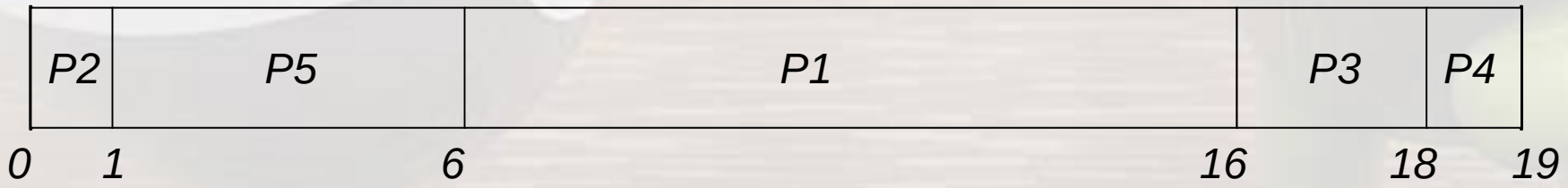
Escalonamento por Prioridade (2)

- 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.
      - Ex: Processos limitados por E/S recebem prioridade mais alta
  - Definição externa:
    - Definida por algum critério externo ao S.O (tipo do processo, departamento responsável, custo, etc.)

Escalonamento por Prioridades (3)

Escalonamento por Prioridade (4)

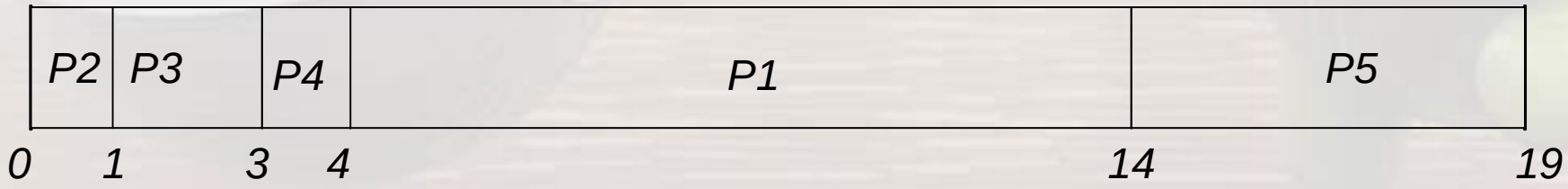
Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2



- *Average waiting time* = (6+0+16+18+1)/5 = 8,2ms

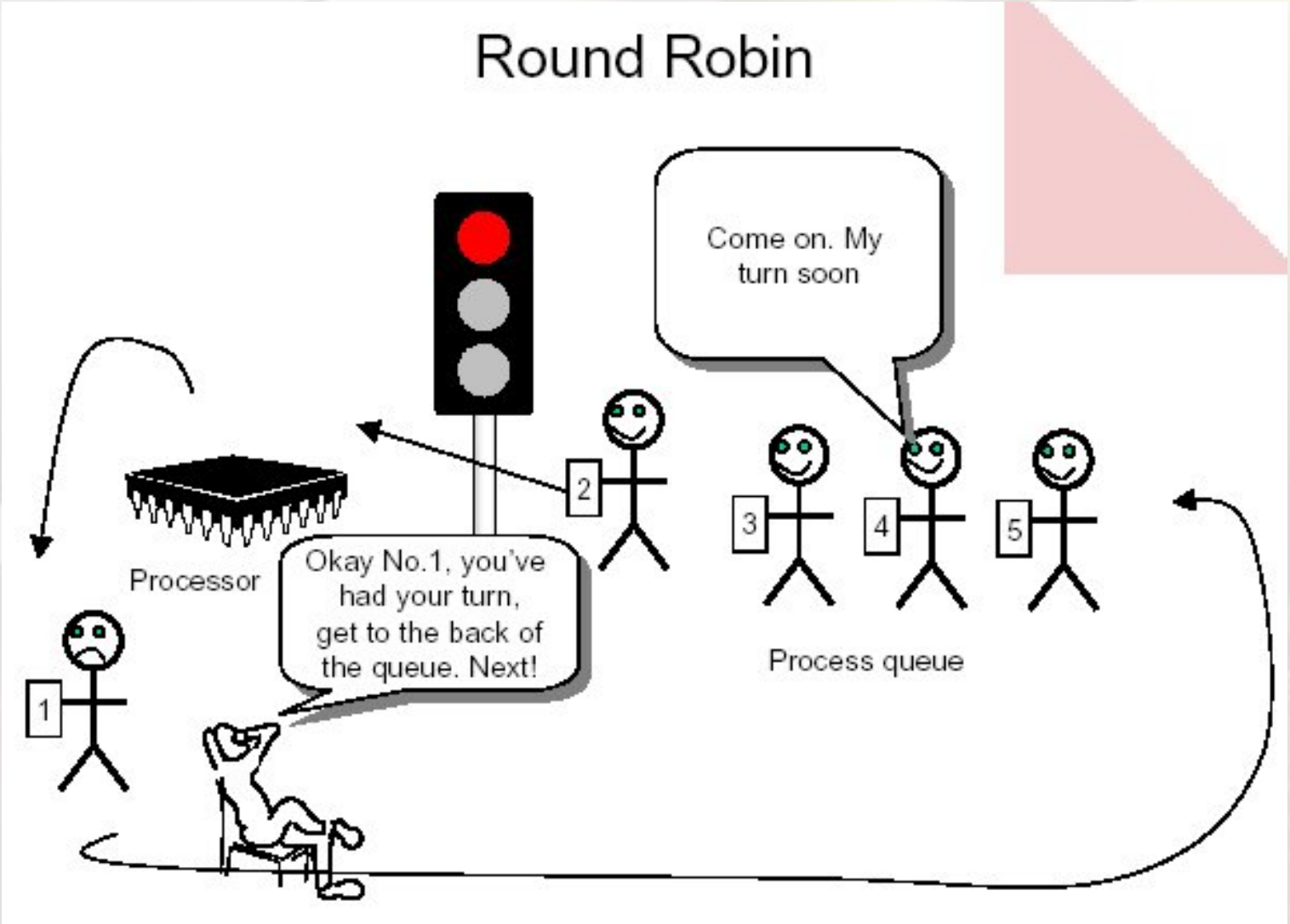
Escalonamento por Prioridade (5)

Process	Burst Time	Priority
$P_{1\ background}$	10	1
$P_{2\ Interativo}$	1	0
$P_{3\ Interativo}$	2	0
$P_{4\ Interativo}$	1	0
$P_{5\ Background}$	5	1



- *Average waiting time* = (4+0+1+3+14)/5 = 4,4ms

Escalonamento Round-Robin (1)



Escalonamento Round-Robin (2)

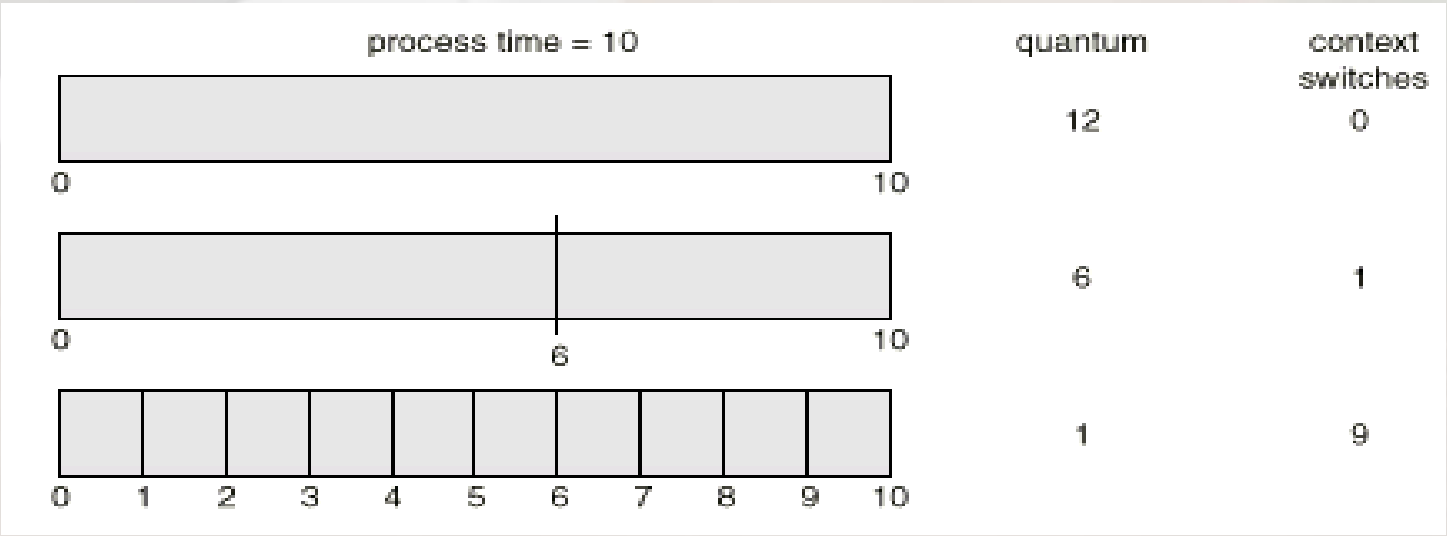
- Algoritmo típico de sistemas operacionais de tempo compartilhado.
- Cada processo recebe uma pequena fatia de tempo de CPU (quantum),usualmente entre 10 e 100 ms.
- Após o término da sua fatia de tempo o processo é "interrompido" e colocado no final da fila de prontos ("preempção" baseada na interrupção de relógio).
- É um algoritmo justo???

Escalonamento Round-Robin (3)

- 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?

Escalonamento Round-Robin (4)

- Dependente do tamanho do quantum:
  - q grande ⇒ tende a FIFO
  - q pequeno ⇒ gera muito overhead devido às trocas de contexto.



quantum	context switches
12	0
6	1
1	9

Escalonamento Round-Robin (5)

- Exemplos:
- Δt = 4 mseg
- x = 1mseg

quantum

→ 20% de tempo de CPU é perdido
→ menor eficiência

- Δt = 99 mseg
- x = 1mseg

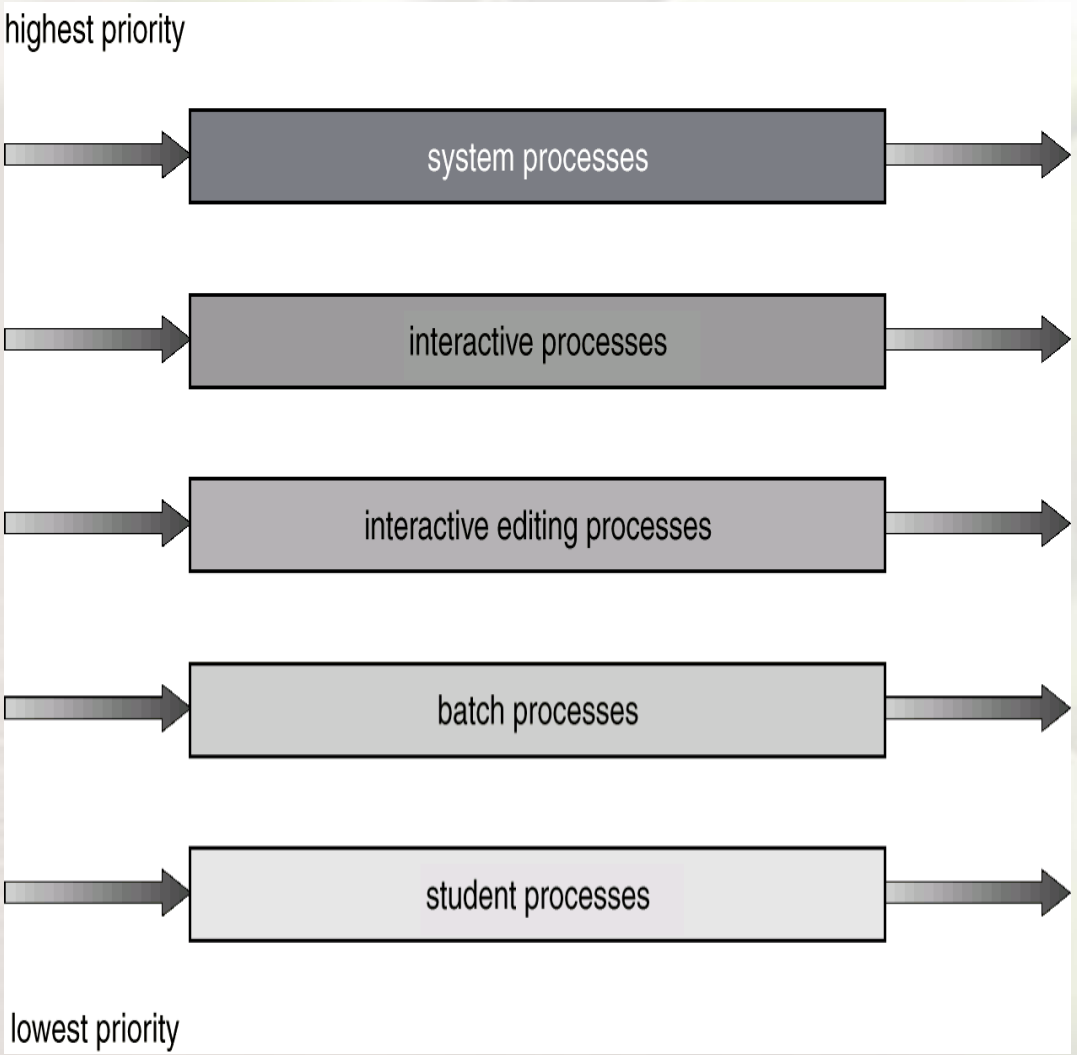
→ 1% de tempo de CPU é perdido
→ Tempo de espera dos processos é maior

chaveamento

quantum razoável: 20-50 mseg

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 de prioridade, conforme a sua importância .



Escalonamento Multinível (2)

- Por exemplo, 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)

- Escalonamento deve ser feito entre as filas:
  - Prioridades fixas – atende primeiro aos processos da fila *foreground* e somente depois aos da fila *background*.
  - *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.

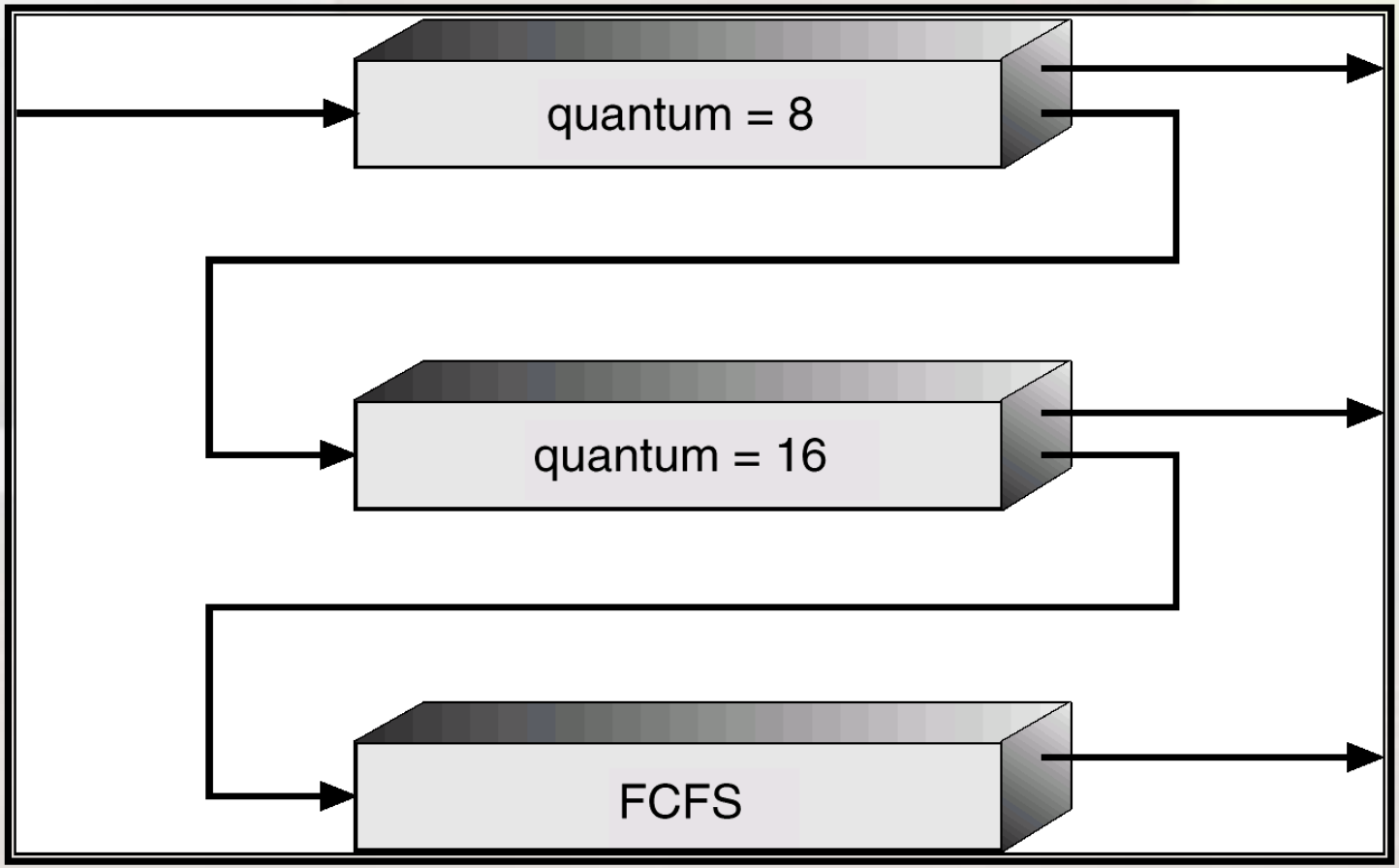
Escalonamento Multinível com Feedback

- O processo pode se mover entre as várias filas. Deste modo, a estratégia de *aging* pode ser implementada.
- 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;
  - Método usado para se determinar em que fila o processo será inserido.

Exemplo (1)

- 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 Q2.
  - Em Q_2, FCFS

Exemplo (2)



Algoritmo Garantido

- Garantias são dadas aos processos dos usuários
- Exemplo: n processos → 1/n do tempo de CPU para cada processo;
  - Deve ser mantida taxa de utilização de cada processo
  - Tem prioridade o que estiver mais distante do prometido
- Difícil de implementar
  - Monitorar quanto da CPU recebeu cada processo desde sua criação.
  - Busca na lista encadeada

Algoritmo por Loteria

- Cada processo recebe “tickets” que lhe dão direito de execução;
- A cada troca de processo um “tickets” é sorteado
- O dono do “tickets” sorteado recebe o direito de ocupar a CPU
- Possível definir prioridade entre os processos por meio do número de “tickets” atribuído a cada processo
- Processos cooperativos podem trocar bilhetes (cliente-servidor)
- Fácil de implementar e de adaptar

Algoritmo por Fração Justa (Fair-Share)

- O escalonamento é feito considerando o dono dos processos
- Cada usuário recebe uma fração da CPU e processos são escalonados visando garantir essa fração
- Se um usuário A possui mais processos que um usuário B e os dois têm a mesma prioridade, os processos de A demorarão mais que os do B

Referências

- Slides adaptados de Roberta Lima Gomes (UFES)

- Bibliografia
  - Silberschatz A. G.; Galvin P. B.; Gagne G.; "Fundamentos de Sistemas Operacionais", 6a. Edição, Editora LTC, 2004.
    - Capítulo 5
  - A.S. Tanenbaum, "Sistemas Operacionais Modernos", 3a. Edição, Editora Prentice-Hall, 2010.
    - Seção 2.4 (até 2.4.5 inclusa)
  - Deitel H. M.; Deitel P. J.; Choffnes D. R.; "Sistemas Operacionais", 3ª. Edição, Editora Prentice-Hall, 2005
    - Capítulo 8 (até seção 8.8 inclusa)