

Sistemas Operacionais I

L. F. Friedrich

Capítulo 2 – Parte 3 Escalonamento de Processos

Escalonamento de Processos

- Escalonamento é um importante tópico no estudo e pesquisa de sistemas operacionais.
- Escalonamento é necessário porque o número de recursos computacionais – a UCP – é **limitado**.
- Dois tipos de processos
 - CPU-bound (or compute-bound) process
 - Gasta a maior parte do seu tempo com UCP.
 - Por exemplo, um programa de renderização.
 - I/O-bound process
 - Gasta a maior parte do tempo esperando E/S.
 - Por exemplo, tocador de filmes.

Comportamento escalonamento-processo

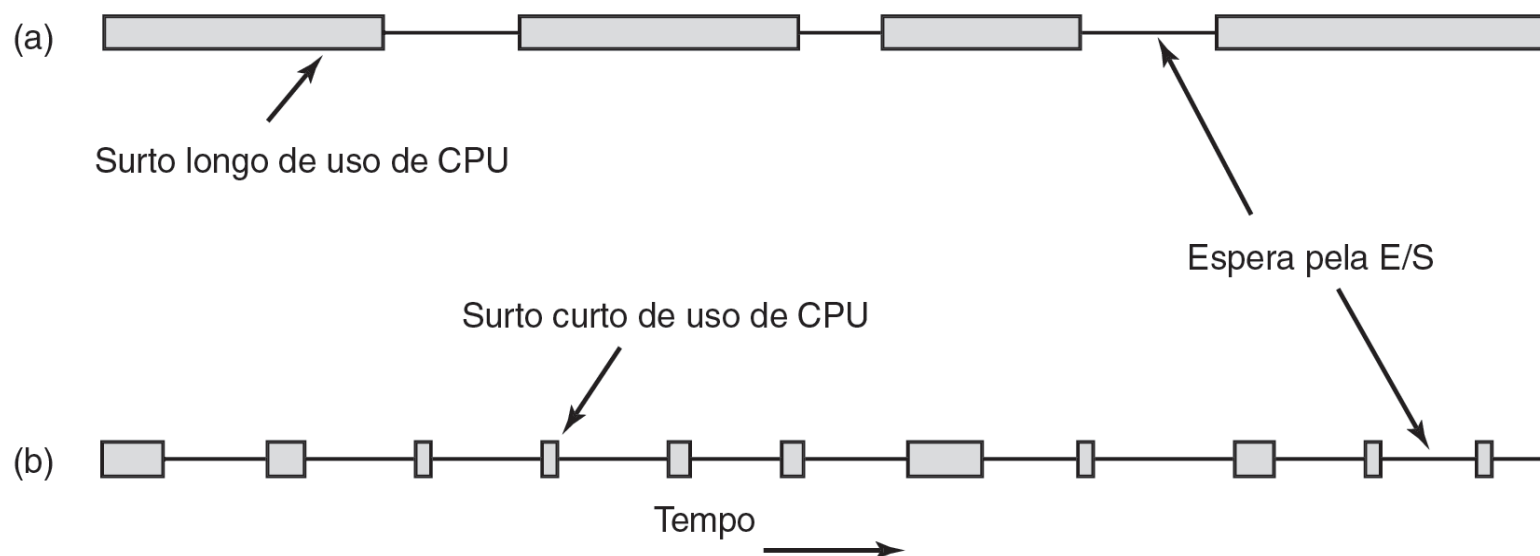
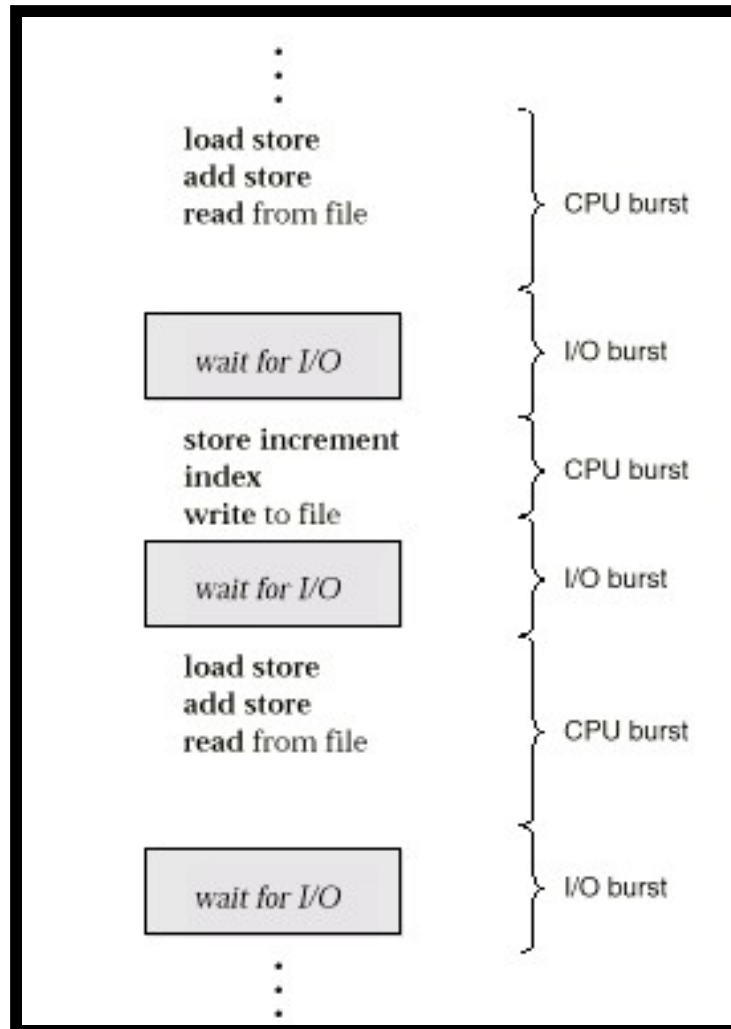
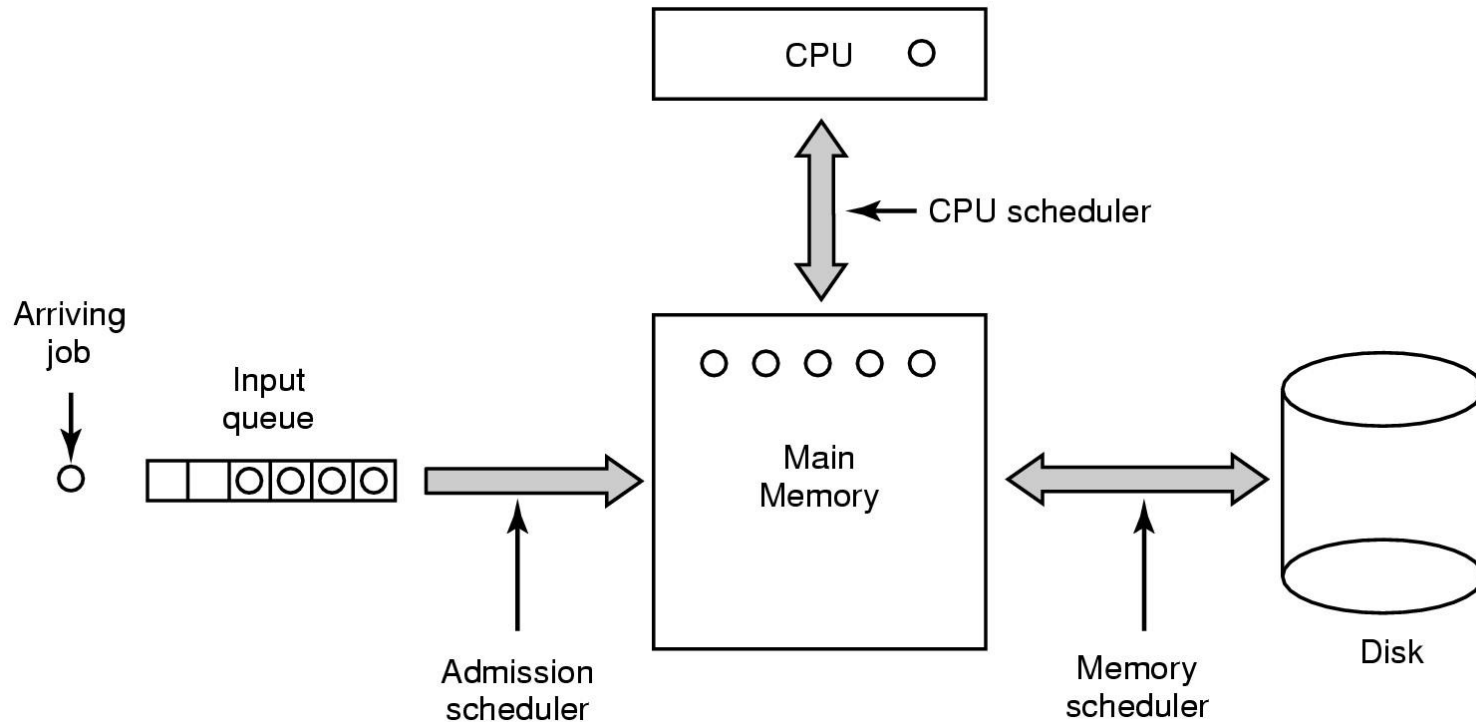


Figura 2.31 Usos de surtos de CPU se alternam com períodos de espera por E/S. (a) Um processo orientado à CPU. (b) Um processo orientado à E/S.

Ciclos do processo

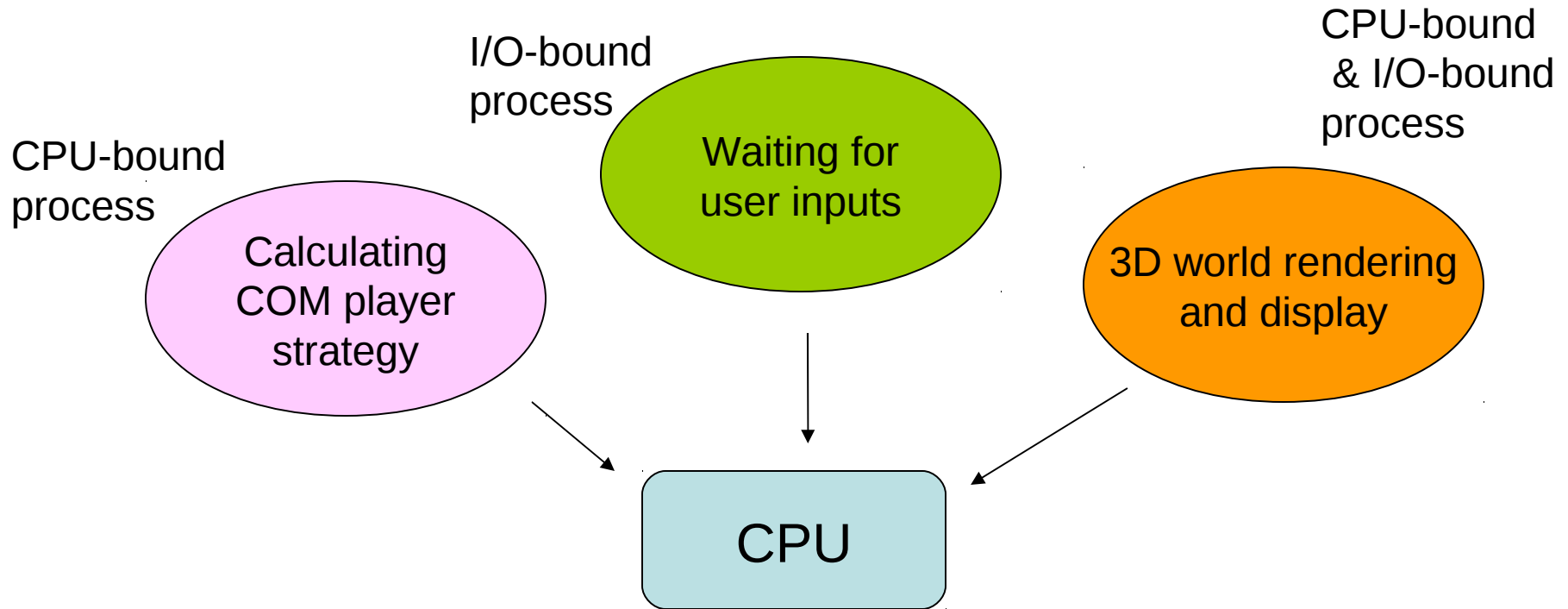


Tipo de Escalonamento



Exemplo: 3D game

- Quando escalonar?
- Como escalonar?



Quando escalonar

- Quando um novo processo é **criado**.
 - Executar o processo pai ou filho é uma escolha do escalonador.
 - Linux: o filho é sempre escalonado antes do pai depois de um `fork()`.
- Quando um processo **termina**.
 - Senão, a UCP ficará ociosa sem um desempenho útil.
- Quando um processo inicia **espera por E/S**.
 - O processo libera a UCP para outro processo executar.
 - De outra forma, a UCP ficará ociosa.
- Quando uma **operação de E/S é completada**.
 - O processo pode precisar urgente do resultado da operação de E/S.

Como escalonar

- O algoritmo **Não-preemptivo**
 - O processo, depois de escolhido pelo escalonador, tem permissão para executar até
 - Ser bloqueado (E/S); ou até
 - Voluntariamente liberar a UCP(exit).
 - Ex., Batch system.

Como escalonar

- O algoritmo **Preemptivo**
 - O processo, depois de escolhido pelo escalonador, executa por um máximo período de tempo.
 - Tal período de tempo é chamado de **quantum**.
 - Quando o quantum de um processo executando **expira**, outro processo será escolhido pelo escalonador.
 - Quando um processo executando é **bloqueado** antes de seu quantum expirar, outro processo será escolhido pelo escalonador.
 - Quando os quanta de todos os processos expiram, o escalonador reseta seus quanta e escolhe outro processo para executar.

Categorias dos algoritmos de escalonamento

- Em lote.
- Interativa.
- Tempo real.

Objetivos dos algoritmos de escalonamento

Todos os sistemas

Justiça — dar a cada processo uma porção justa da CPU

Aplicação da política — verificar se a política estabelecida é cumprida

Equilíbrio — manter ocupadas todas as partes do sistema

Sistemas em lote

Vazão (*throughput*) — maximizar o número de tarefas por hora

Tempo de retorno — minimizar o tempo entre a submissão e o término

Utilização de CPU — manter a CPU ocupada o tempo todo

Sistemas interativos

Tempo de resposta — responder rapidamente às requisições

Proporcionalidade — satisfazer às expectativas dos usuários

Sistemas de tempo real

Cumprimento dos prazos — evitar a perda de dados

Previsibilidade — evitar a degradação da qualidade em sistemas multimídia

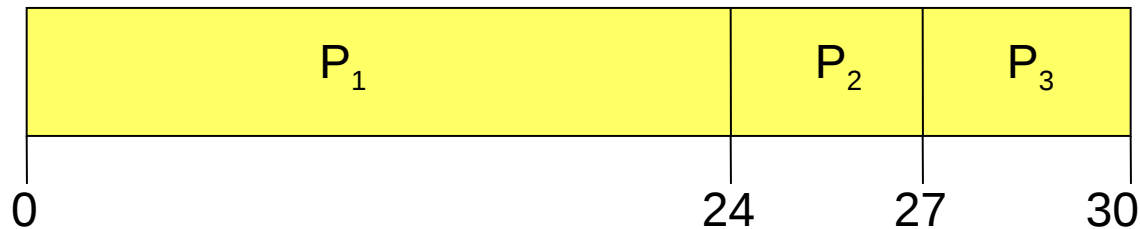
■ **Tabela 2.8** Alguns objetivos do algoritmo de escalonamento sob diferentes circunstâncias.

Escalonamento em sistemas em lotes

- Primeiro a chegar, primeiro a ser servido.
 - FCFS, FIFO
- Tarefa mais curta primeiro.
 - SJF, SJN
- Próximo de menor tempo restante.
 - SRTF, SRTN, SJF preemptivo

Escalonamento FCFS

Diagrama de Gantt



| Process | CPU Req |
|----------------|---------|
| P ₁ | 24 |
| P ₂ | 3 |
| P ₃ | 3 |

Tempo de espera: $P_1 = 0, P_2 = 24, P_3 = 27$.

Tempo médio de espera = $(0 + 24 + 27) / 3 = 17$.

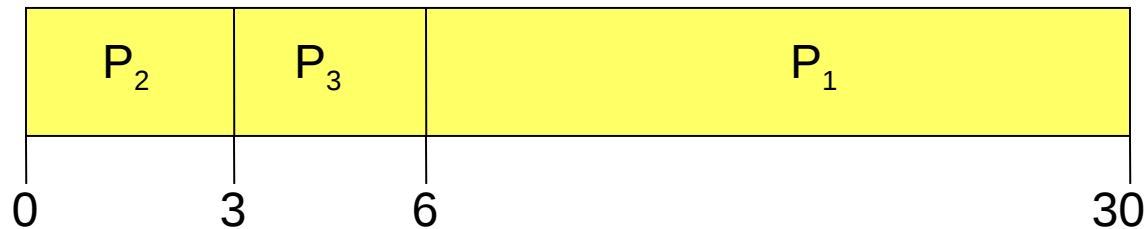
Ordem de chegada

$P_1 \rightarrow P_2 \rightarrow P_3$
(todos no tempo 0)

* Tempo de espera = tempo que o processo espera para usar a CPU.

Escalonamento FCFS

Diagrama de Gantt



Tempo de espera: $P_1 = 6$, $P_2 = 0$, $P_3 = 3$.

Tempo médio de espera = $(6 + 0 + 3) / 3 = 3$.

| Process | CPU Req |
|----------------|---------|
| P ₁ | 24 |
| P ₂ | 3 |
| P ₃ | 3 |

Ordem chegada

$P_2 \rightarrow P_3 \rightarrow P_1$
(todos no tempo 0)

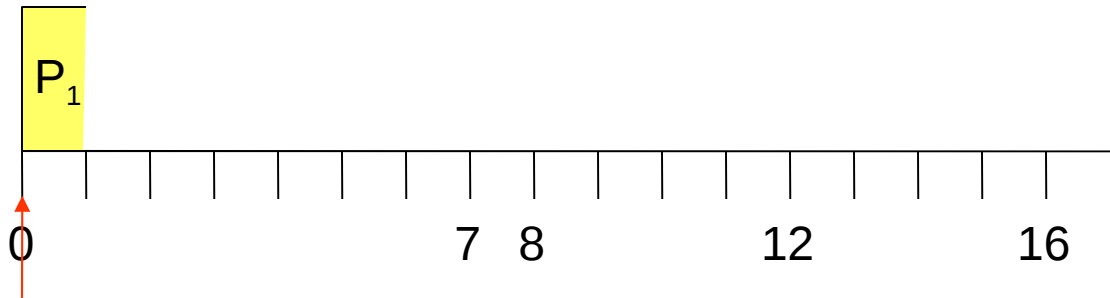
E se mudar a ordem de chegada?

Escalonamento FCFS

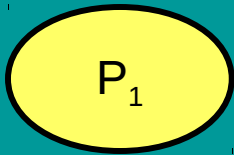
- First-Come, First-Served (FCFS), ou First-In-First-Out (FIFO) scheduling
 - Um algoritmo **não-preemptivo**; e
 - Um algoritmo de escalonamento de sistemas batch.
- Como otimizar o tempo médio de espera?
 - Offline : fácil.
 - Online : difícil.

Shortest Job First

SJF Não-preemptivo



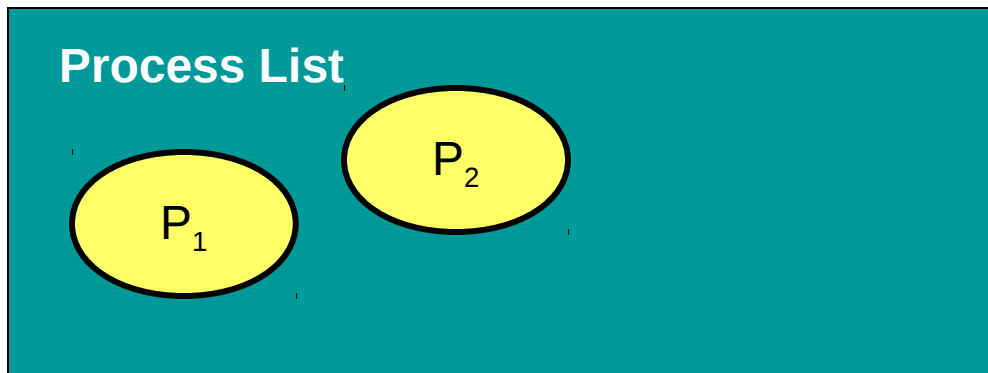
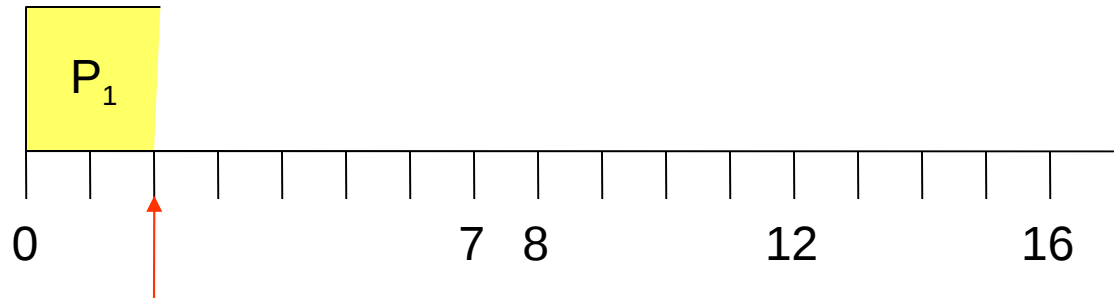
Process List



| Processo | chegada | CPU |
|----------|---------|-----|
| P_1 | 0 | 7 |
| P_2 | 2 | 4 |
| P_3 | 4 | 1 |
| P_4 | 5 | 4 |

Shortest Job First

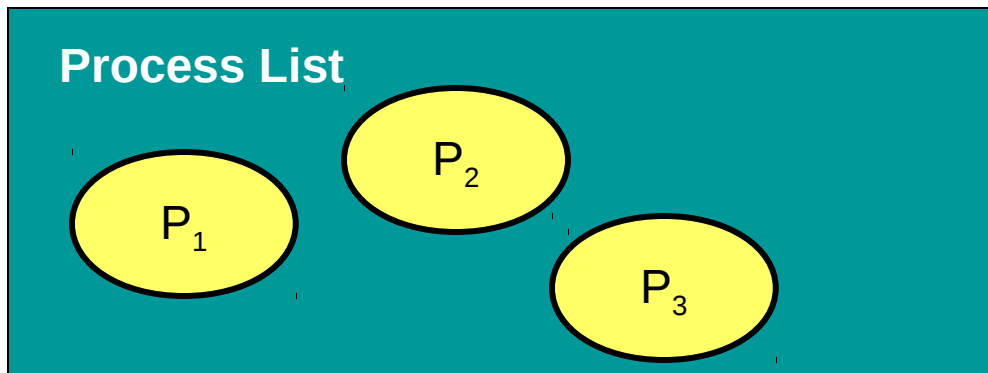
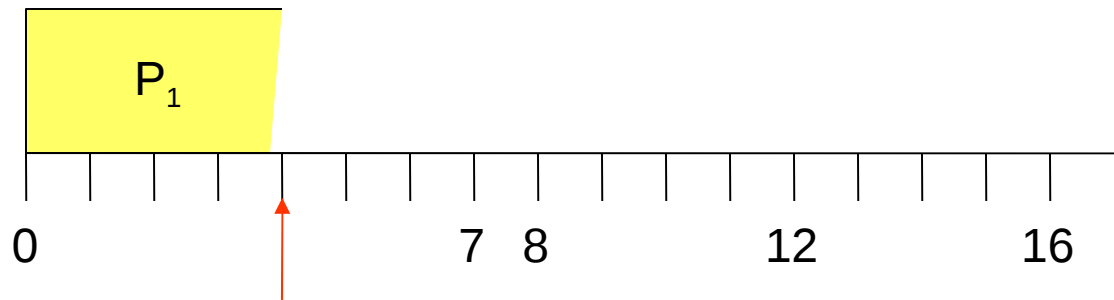
SJF Não-preemptivo



| Processo | chegada | CPU |
|----------|---------|-----|
| P_1 | 0 | 7 |
| P_2 | 2 | 4 |
| P_3 | 4 | 1 |
| P_4 | 5 | 4 |

Shortest Job First

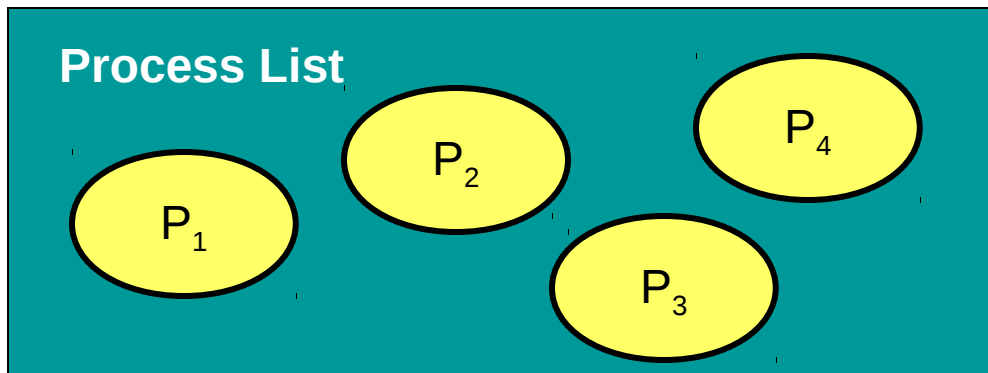
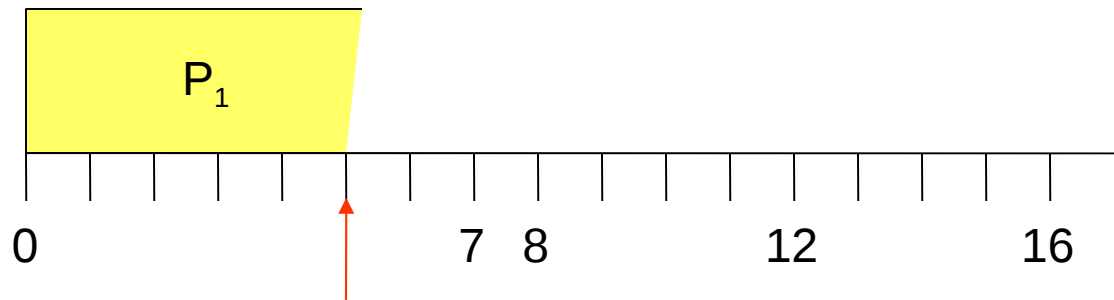
SJF Não-preemptivo



| Processo | chegada | CPU |
|----------|---------|-----|
| P_1 | 0 | 7 |
| P_2 | 2 | 4 |
| P_3 | 4 | 1 |
| P_4 | 5 | 4 |

Shortest Job First

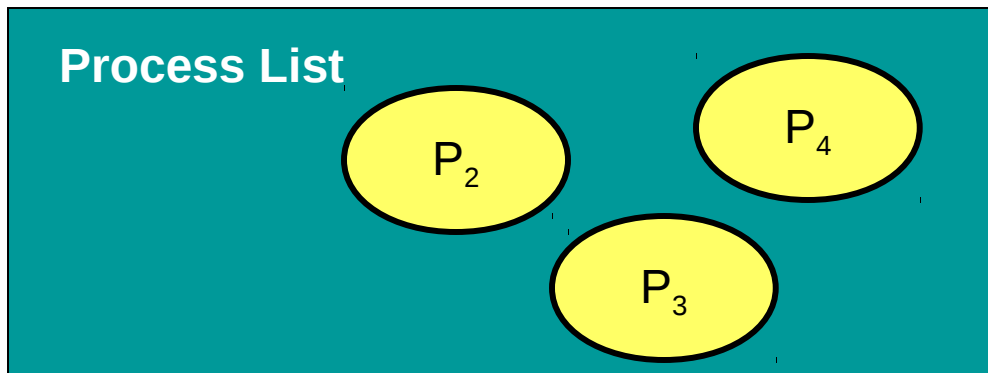
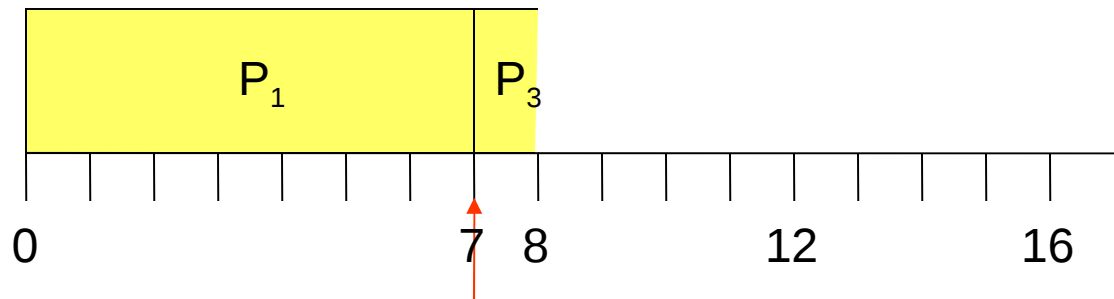
SJF Não-preemptivo



| Processo | chegada | CPU |
|----------|---------|-----|
| P_1 | 0 | 7 |
| P_2 | 2 | 4 |
| P_3 | 4 | 1 |
| P_4 | 5 | 4 |

Shortest Job First

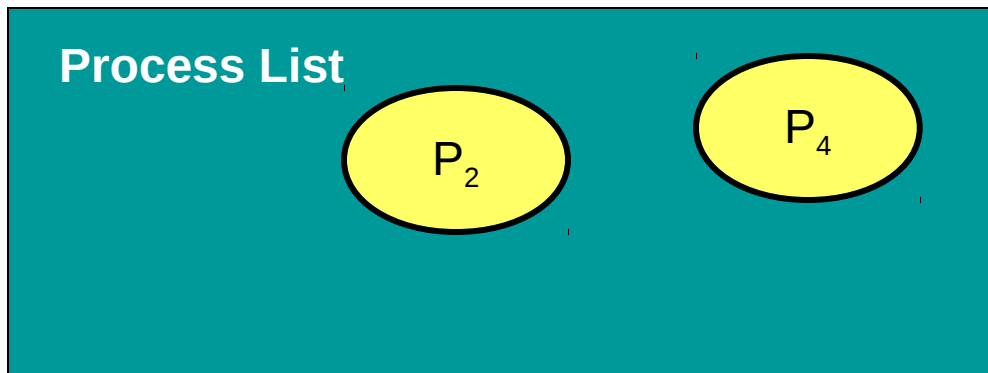
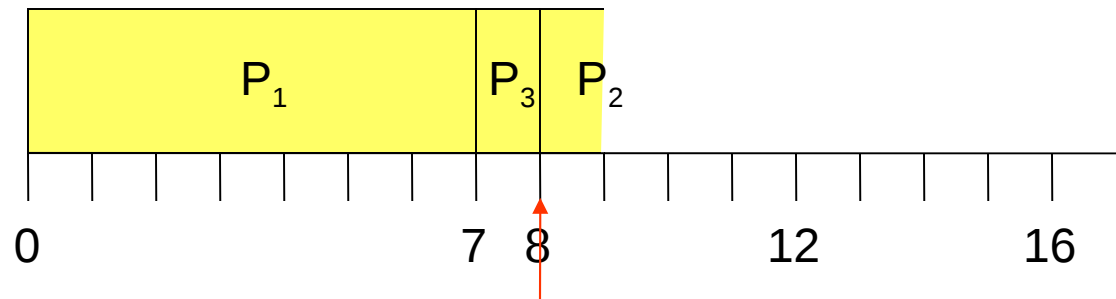
SJF Não-preemptivo



| Processo | chegada | CPU |
|----------------|---------|-----|
| P ₁ | 0 | 7 |
| P ₂ | 2 | 4 |
| P ₃ | 4 | 1 |
| P ₄ | 5 | 4 |

Shortest Job First

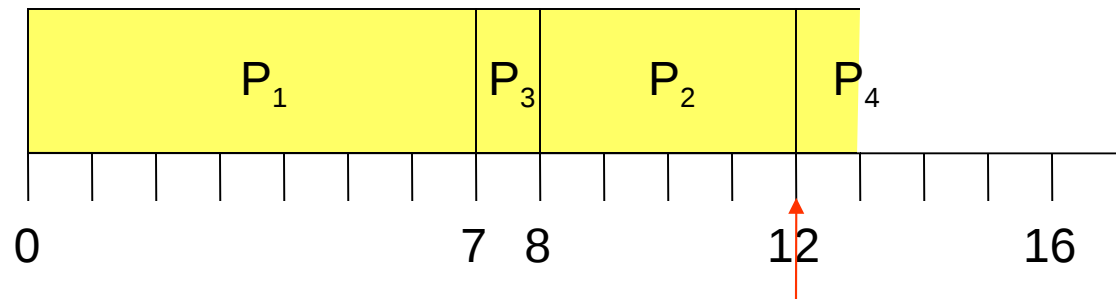
SJF Não-preemptivo



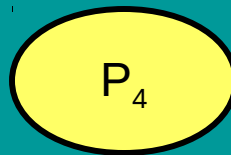
| Process | Arrival time | CPU Requirement |
|----------------|--------------|-----------------|
| P ₁ | 0 | 7 |
| P ₂ | 2 | 4 |
| P ₃ | 4 | 1 |
| P ₄ | 5 | 4 |

Shortest Job First

SJF Não-preemptivo



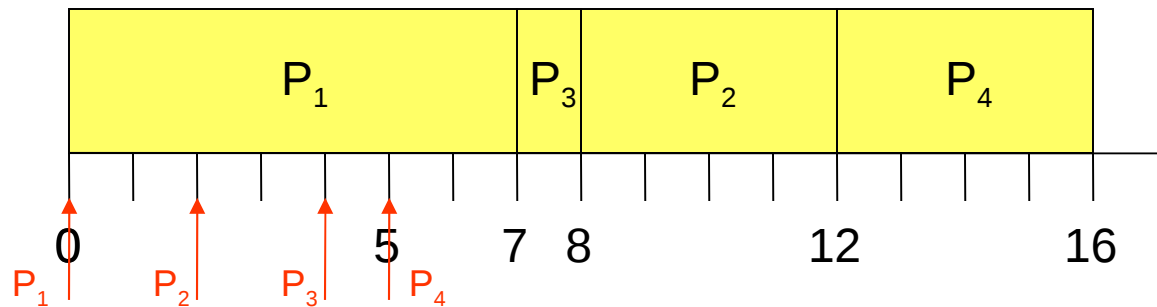
Process List



| Process | Arrival time | CPU Requirement |
|----------------|--------------|-----------------|
| P ₁ | 0 | 7 |
| P ₂ | 2 | 4 |
| P ₃ | 4 | 1 |
| P ₄ | 5 | 4 |

Shortest Job First

SJF Não-preemptivo



*Tempo de espera =
tempo que o processo
espera para usar a CPU

Tempo de espera

$P_1 = 0, P_2 = 6, P_3 = 3, P_4 = 7.$

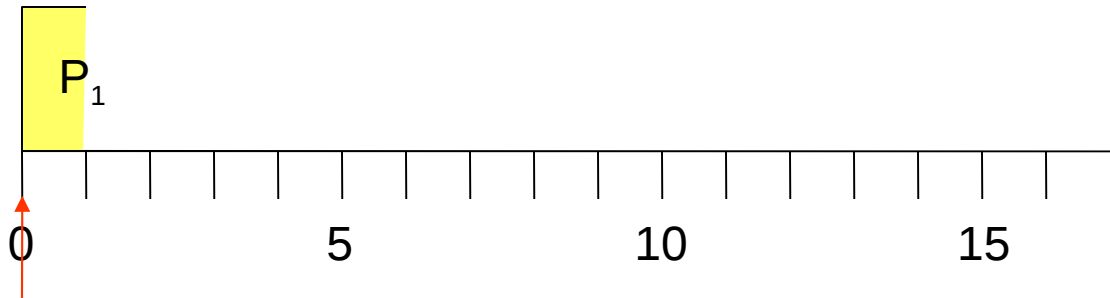
Tempo médio de espera

$(0 + 6 + 3 + 7) / 4 = 4.$

| Process | Arrival time | CPU Requirement |
|----------------|--------------|-----------------|
| P ₁ | 0 | 7 |
| P ₂ | 2 | 4 |
| P ₃ | 4 | 1 |
| P ₄ | 5 | 4 |

Shortest Job First

SJF Preemptivo



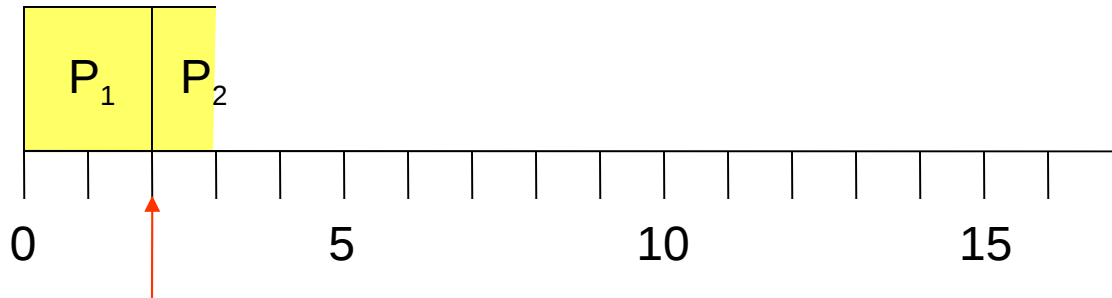
Process List

$P_1: 7$

| Process | Arrival time | CPU Requirement |
|---------|--------------|-----------------|
| P_1 | 0 | 7 |
| P_2 | 2 | 4 |
| P_3 | 4 | 1 |
| P_4 | 5 | 4 |

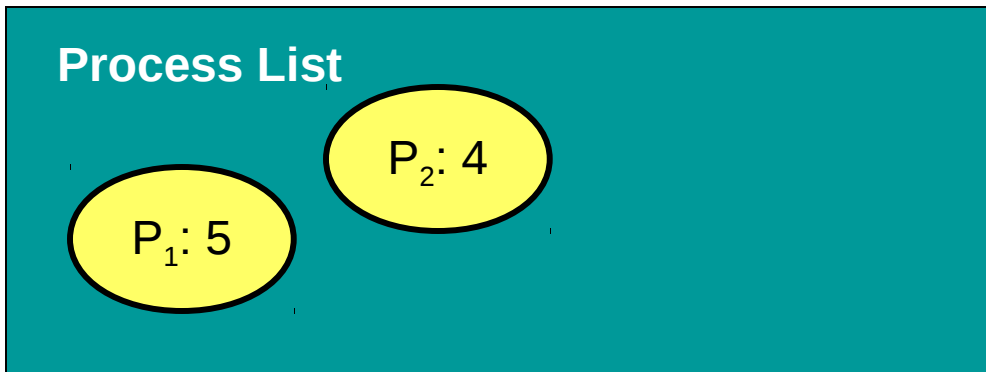
Shortest Job First

SJF Preemptivo



* escalonador é invocado sempre que um novo job chega no sistema ou um job termina sua execução

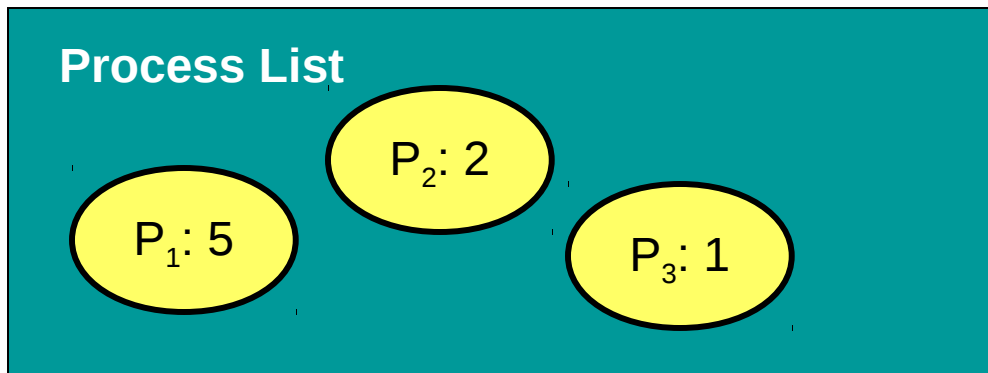
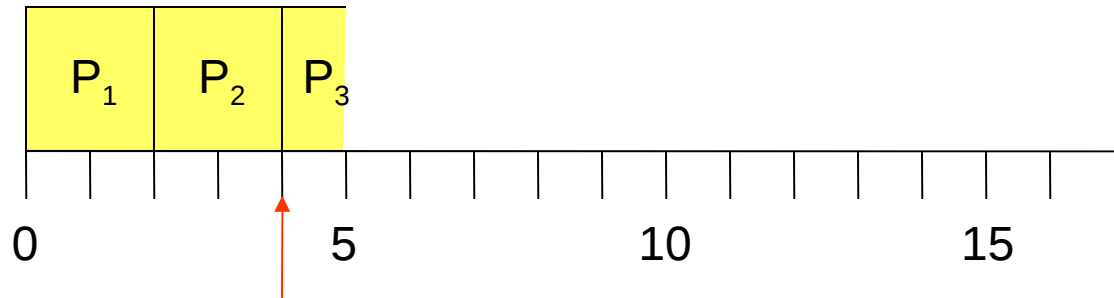
Process List



| Process | Arrival time | CPU Requirement |
|----------------|--------------|-----------------|
| P ₁ | 0 | 7 |
| P ₂ | 2 | 4 |
| P ₃ | 4 | 1 |
| P ₄ | 5 | 4 |

Shortest Job First (SRTF)

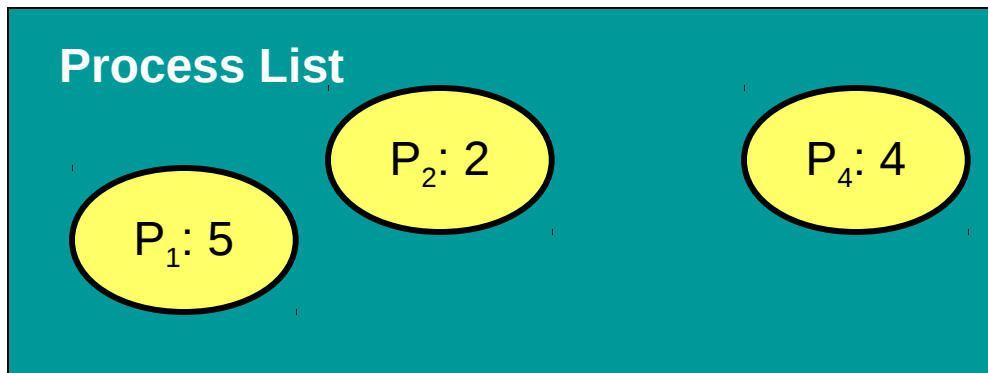
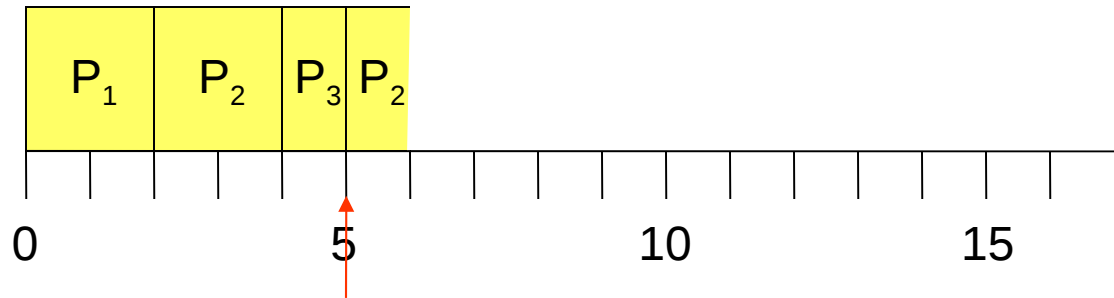
SJF Preemptivo



| Process | Arrival time | CPU Requirement |
|---------|--------------|-----------------|
| P_1 | 0 | 7 |
| P_2 | 2 | 4 |
| P_3 | 4 | 1 |
| P_4 | 5 | 4 |

Shortest Job First

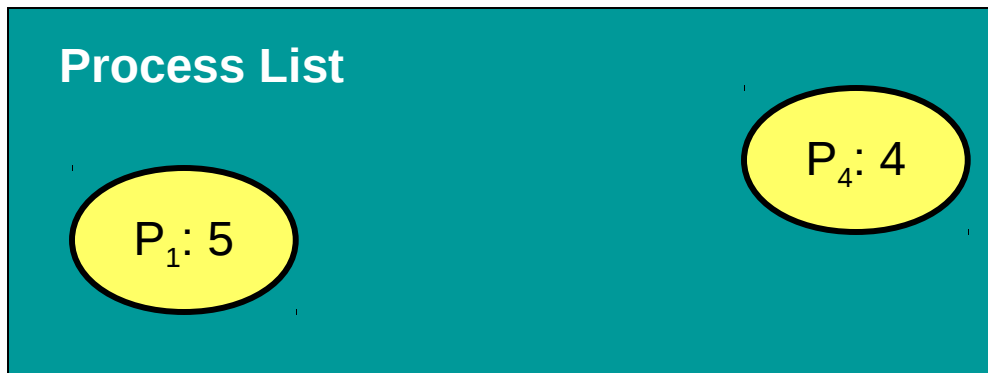
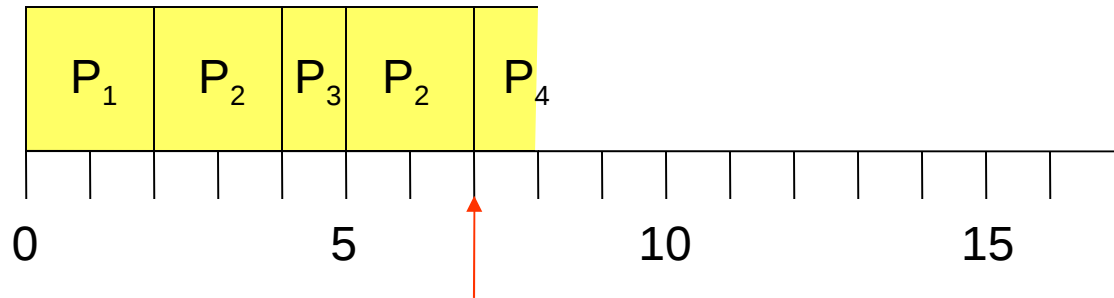
SJF Preemptivo



| Process | Arrival time | CPU Requirement |
|----------------|--------------|-----------------|
| P ₁ | 0 | 7 |
| P ₂ | 2 | 4 |
| P ₃ | 4 | 1 |
| P ₄ | 5 | 4 |

Shortest Job First

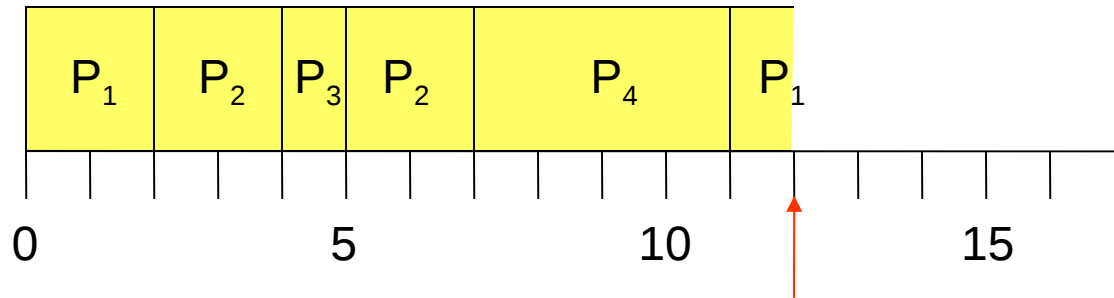
SJF Preemptivo



| Process | Arrival time | CPU Requirement |
|----------------|--------------|-----------------|
| P ₁ | 0 | 7 |
| P ₂ | 2 | 4 |
| P ₃ | 4 | 1 |
| P ₄ | 5 | 4 |

Shortest Job First

SJF Preemptivo



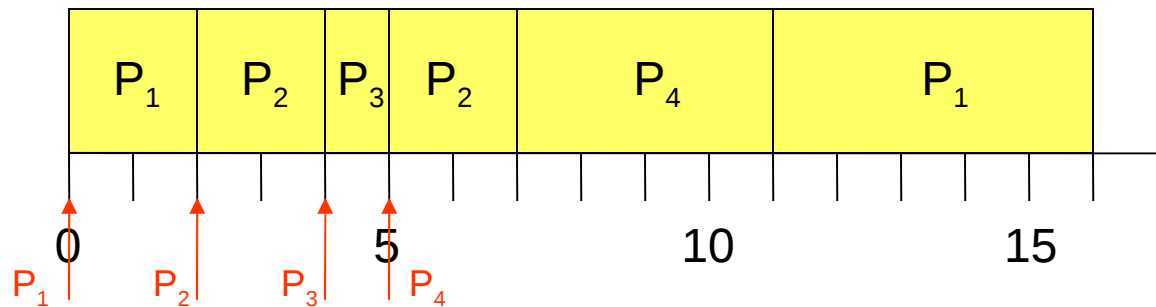
Process List

P₁: 5

| Process | Arrival time | CPU Requirement |
|----------------|--------------|-----------------|
| P ₁ | 0 | 7 |
| P ₂ | 2 | 4 |
| P ₃ | 4 | 1 |
| P ₄ | 5 | 4 |

Shortest Job First

SJF Preemptivo



*Tempo de espera =
tempo que o processo
espera para usar a CPU

Tempo de espera

$P_1 = 9, P_2 = 1, P_3 = 0, P_4 = 2.$

Tempo médio de espera

$(9 + 1 + 0 + 2) / 4 = 3.$

| Process | Arrival time | CPU Requirement |
|----------------|--------------|-----------------|
| P ₁ | 0 | 7 |
| P ₂ | 2 | 4 |
| P ₃ | 4 | 1 |
| P ₄ | 5 | 4 |

Shortest Job First

- Shortest Job First (SJF)
 - Pode ser um algoritmo de escalonamento não-preemptivo ou preemptivo;
 - É um algoritmo de escalonamento para sistema batch.

Highest Response Ratio Next

- Highest Response Ratio Next (HRRN)
 - Tenta eliminar a desvantagem do SJN e SRTN relacionada a tempos longos;
 - O tempo de espera afeta a taxa de resposta que é a base do escalonamento :
 - $RR = w + e / e$ (w=espera e=execução)
 - A taxa de resposta é dinâmico, periodicamente modificado
 - Vantagens: turnaround, throughput
 - Desvantagens: overhead

Escalonamento em sistemas interativos

- Escalonamento por chaveamento circular.
- Escalonamento por prioridades.
- Filas múltiplas.
- Próximo processo mais curto.
- Escalonamento garantido.
- Escalonamento por loteria.
- Escalonamento por fração justa.

Round-Robin

- Round-Robin (RR) é um algoritmo de escalonamento **preemptivo**.
 - Todo processo recebe um quantum.
 - Para uma simples implementação do RR:
 - Todo processo tem o mesmo quantum.
 - Processos estão executando um após outro em uma fila.

Escalonamento por chaveamento circular

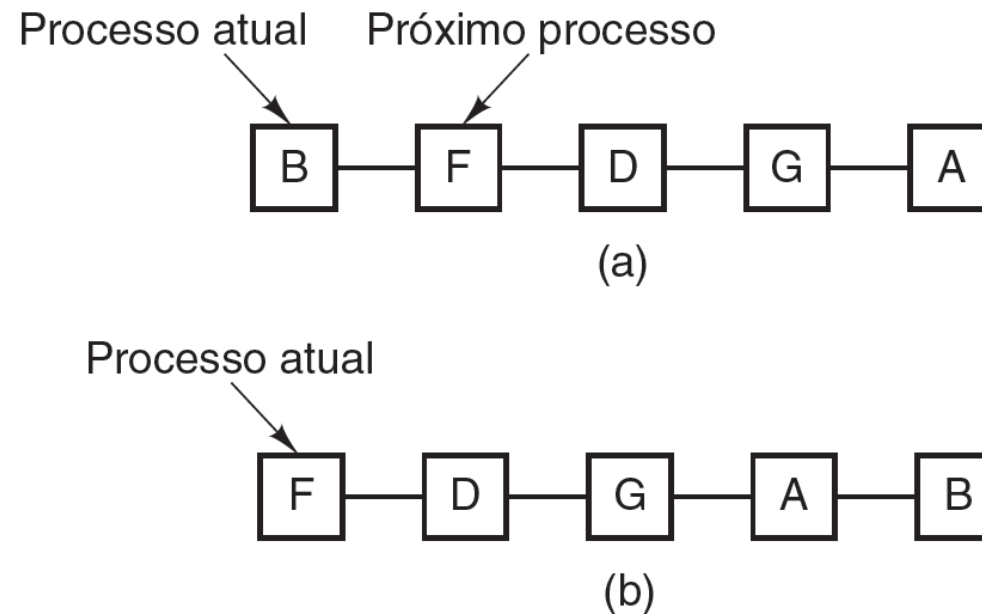
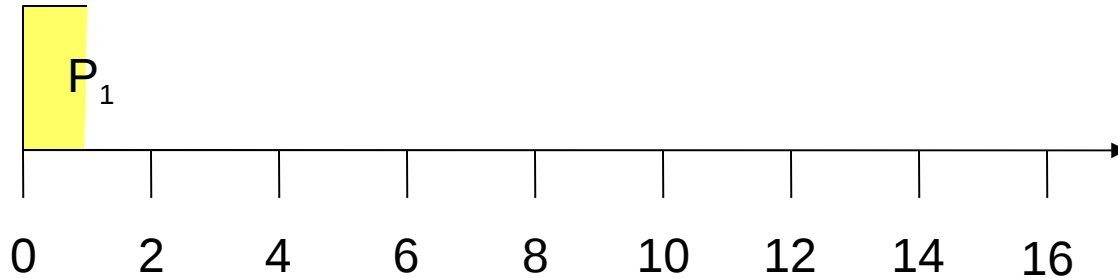
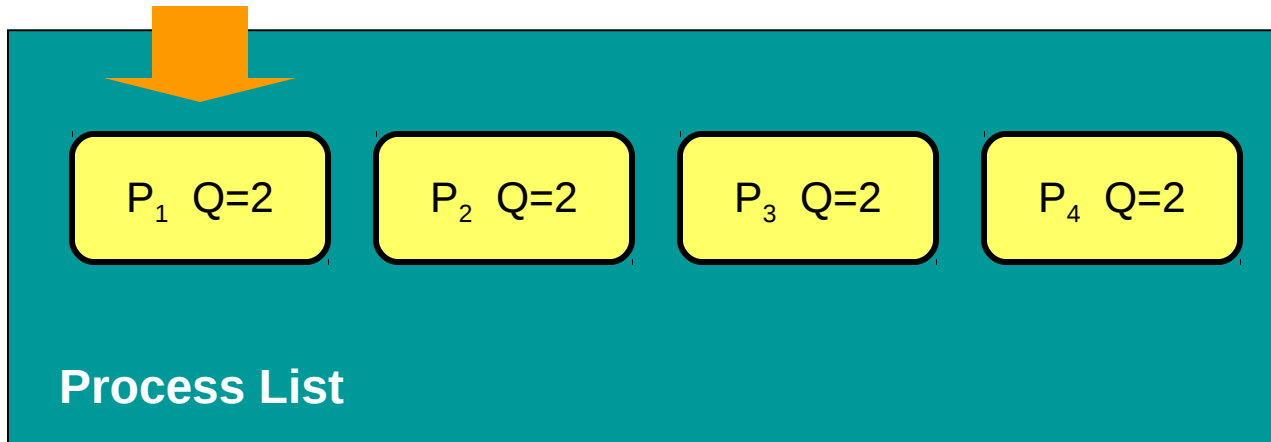


Figura 2.33 Escalonamento circular (*round robin*). (a) Lista de processos executáveis. (b) Lista de processos executáveis depois que *B* usou todo o seu quantum.

Round-Robin

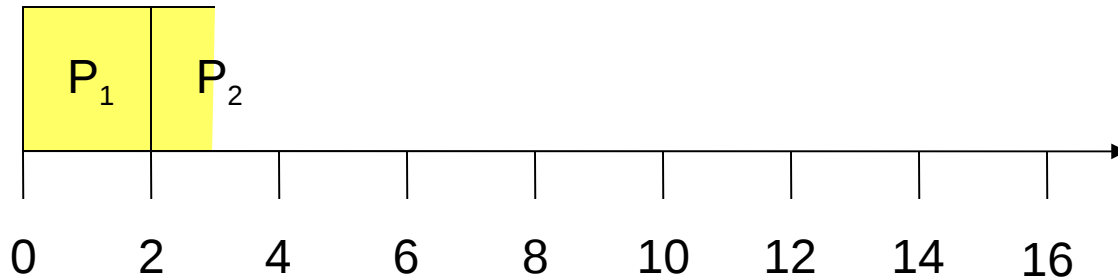


RR Scheduler

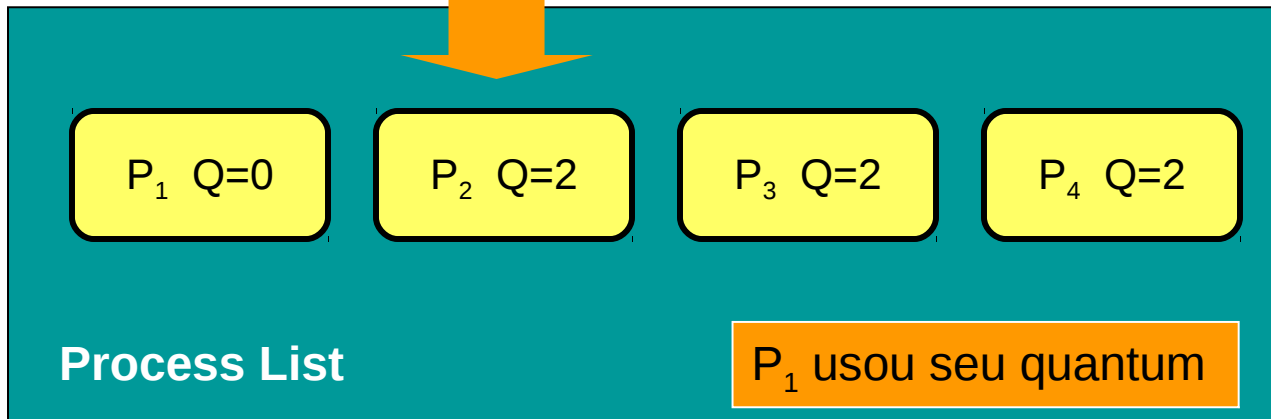


| Process | CPU Requirement |
|---------|-----------------|
| P_1 | 7 |
| P_2 | 4 |
| P_3 | 1 |
| P_4 | 4 |

Round-Robin

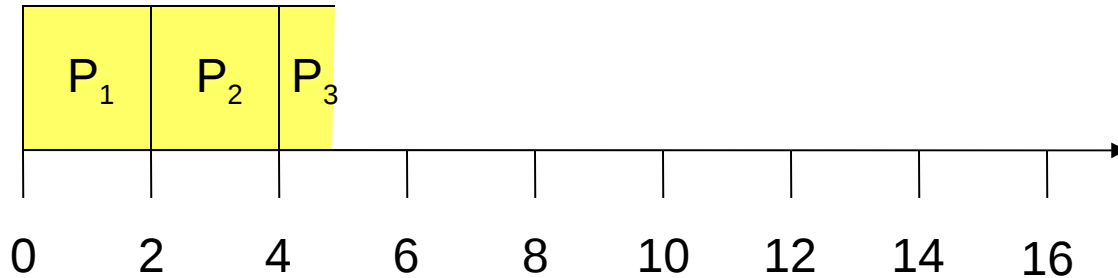


RR Scheduler

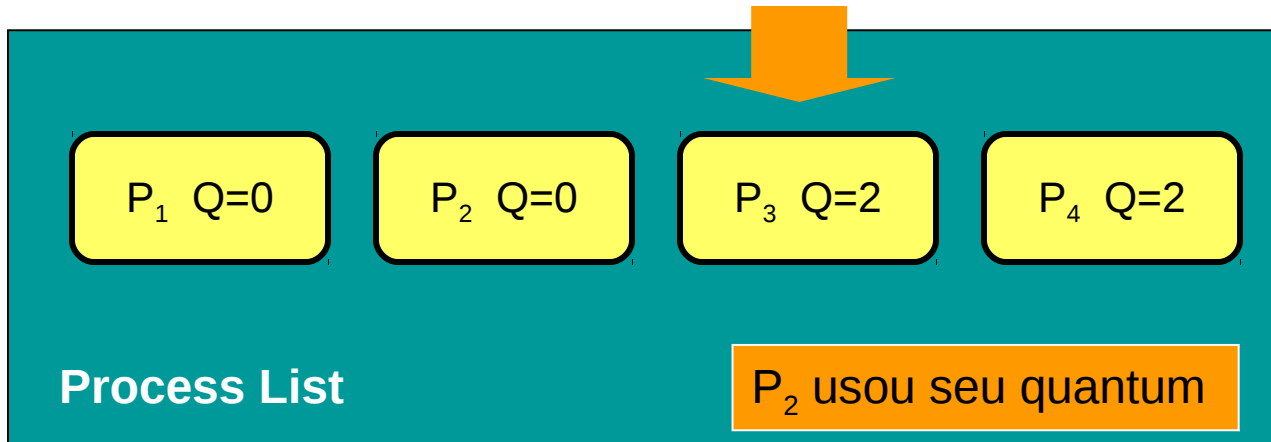


| Process | CPU Requirement |
|---------|-----------------|
| P_1 | 5 |
| P_2 | 4 |
| P_3 | 1 |
| P_4 | 4 |

Round-Robin

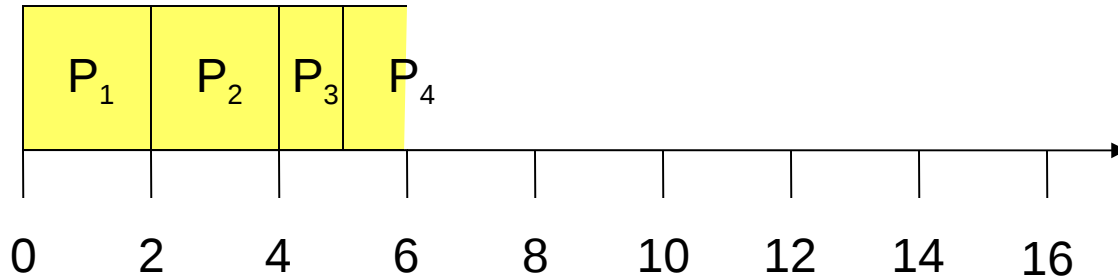


RR Scheduler

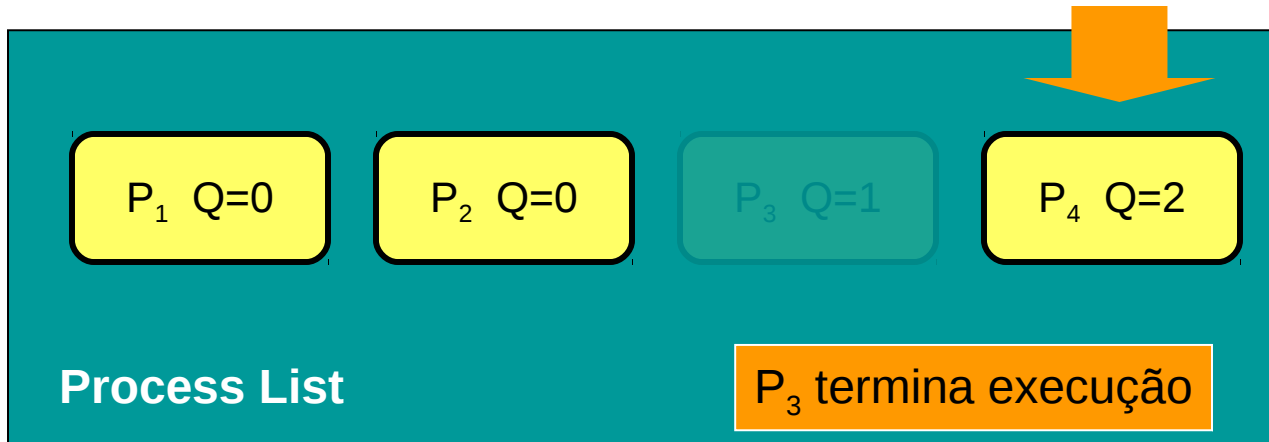


| Process | CPU Requirement |
|----------------|-----------------|
| P ₁ | 5 |
| P ₂ | 2 |
| P ₃ | 1 |
| P ₄ | 4 |

Round-Robin

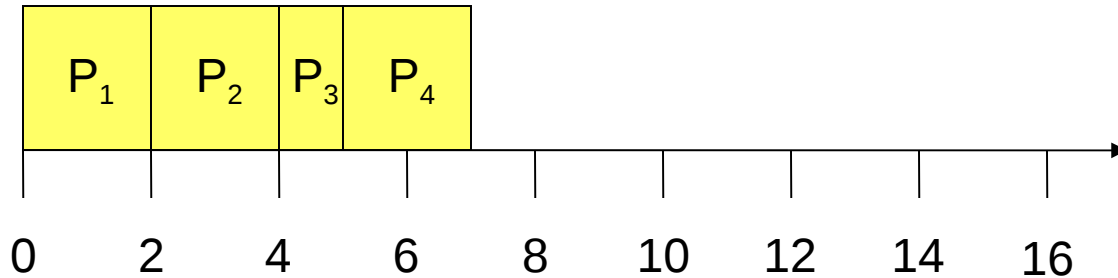


RR Scheduler

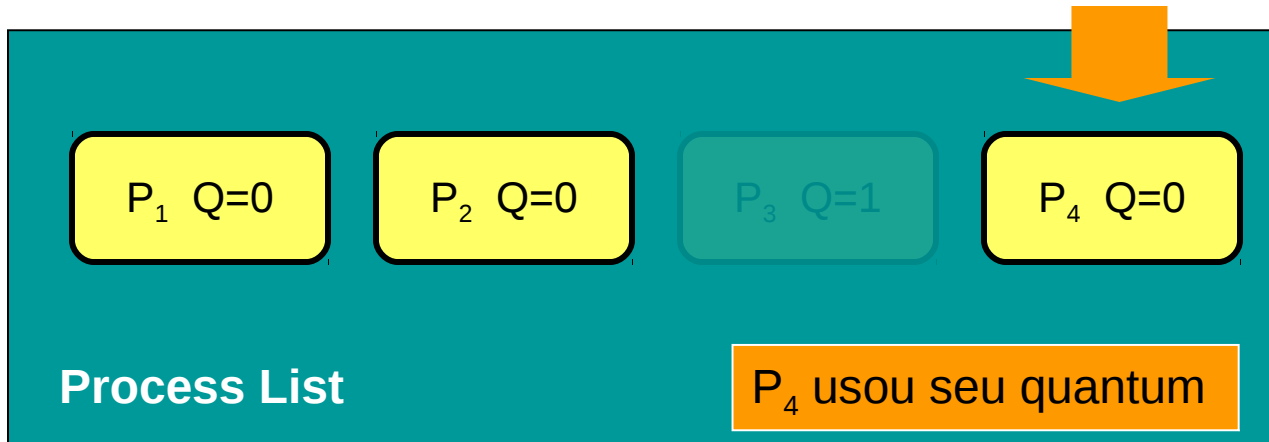


| Process | CPU Requirement |
|---------|-----------------|
| P_1 | 5 |
| P_2 | 2 |
| P_3 | 0 |
| P_4 | 4 |

Round-Robin

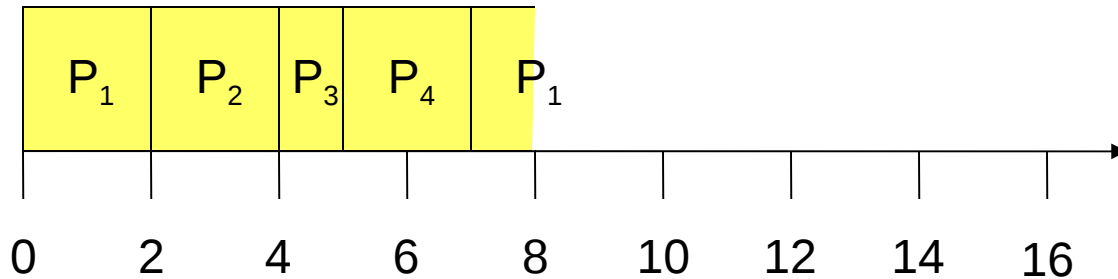


RR Scheduler

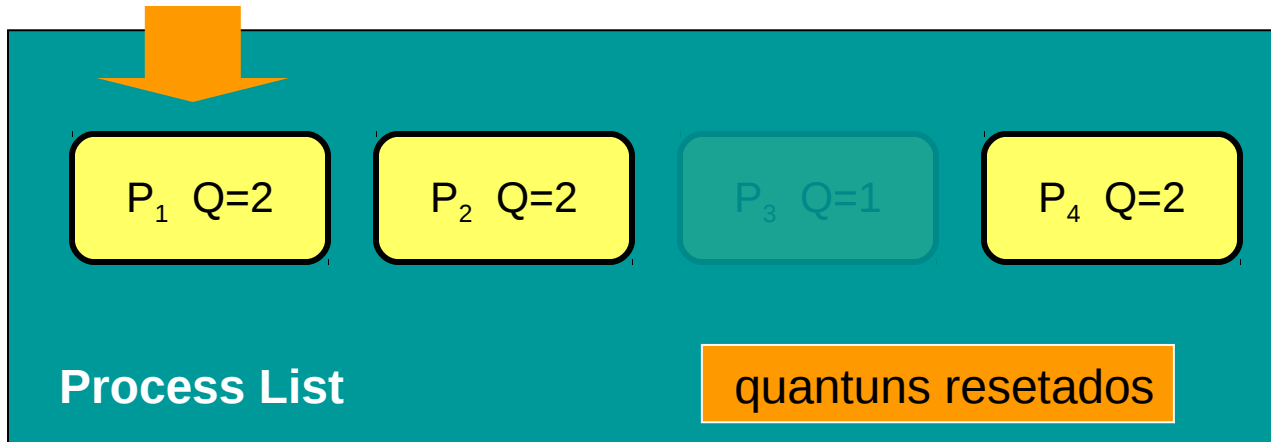


| Process | CPU Requirement |
|---------|-----------------|
| P_1 | 5 |
| P_2 | 2 |
| P_3 | 0 |
| P_4 | 2 |

Round-Robin

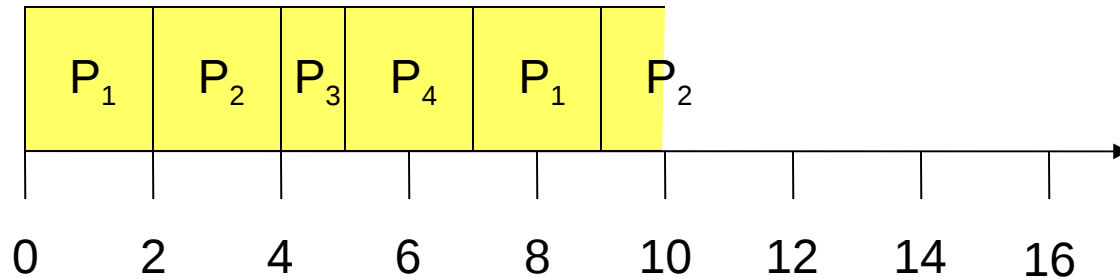


RR Scheduler

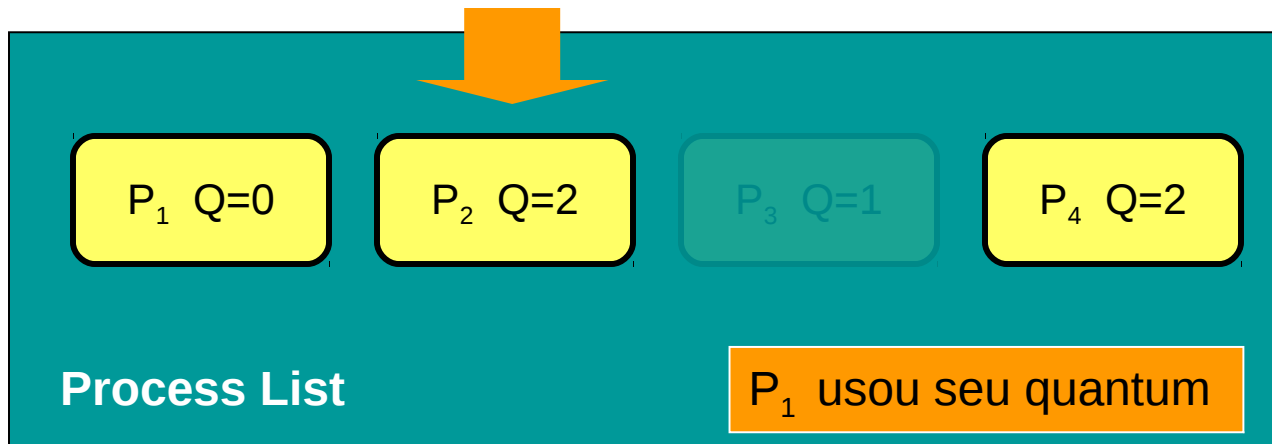


| Process | CPU Requirement |
|----------------|-----------------|
| P ₁ | 5 |
| P ₂ | 2 |
| P ₃ | 0 |
| P ₄ | 2 |

Round-Robin

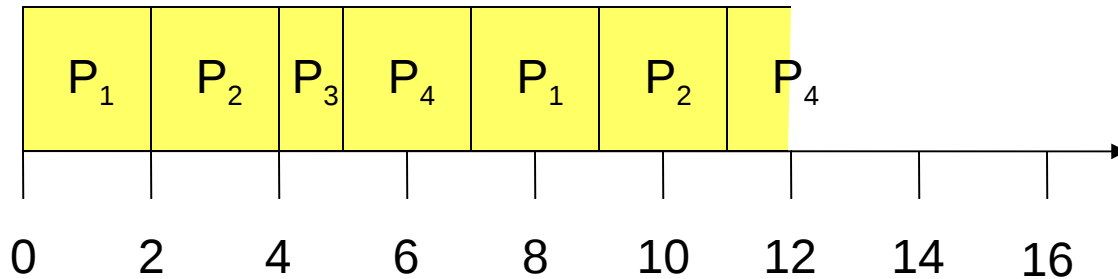


RR Scheduler

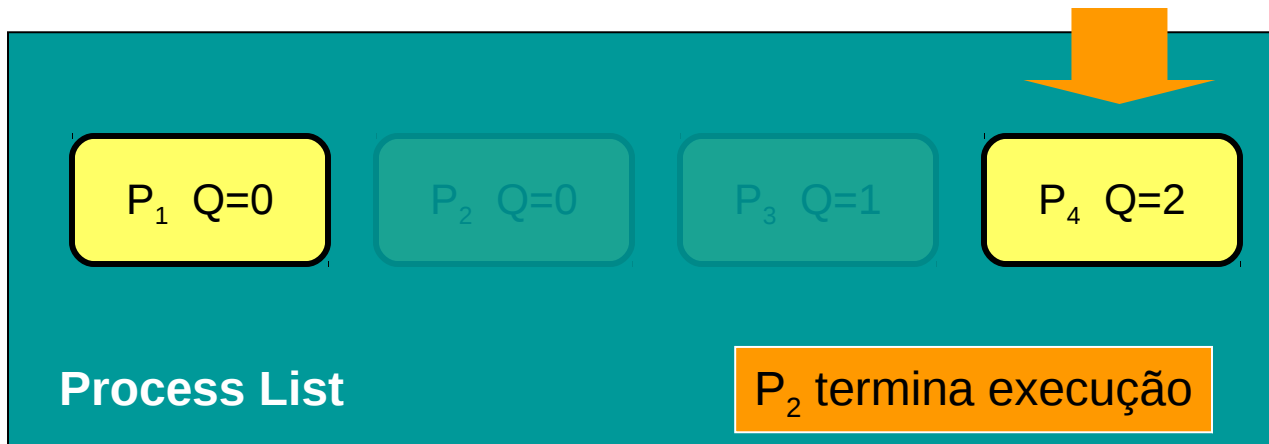


| Process | CPU Requirement |
|----------------|-----------------|
| P ₁ | 3 |
| P ₂ | 2 |
| P ₃ | 0 |
| P ₄ | 2 |

Round-Robin

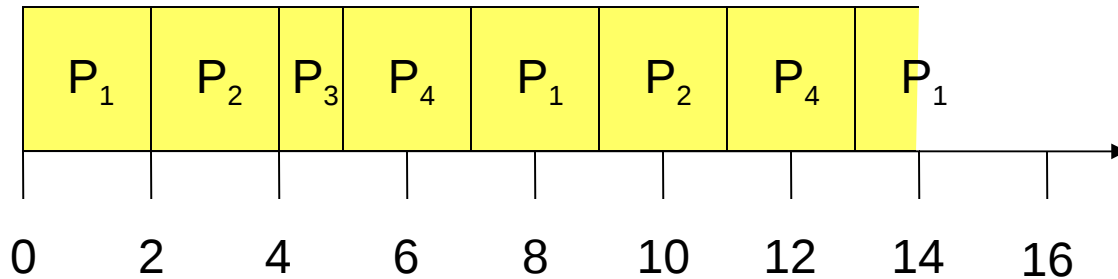


RR Scheduler

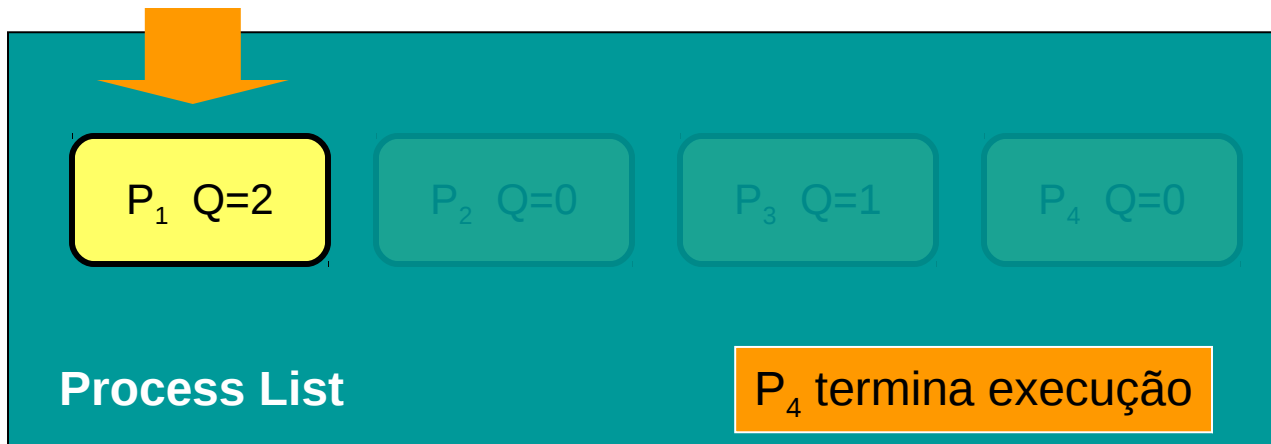


| Process | CPU Requirement |
|----------------|-----------------|
| P ₁ | 3 |
| P ₂ | 0 |
| P ₃ | 0 |
| P ₄ | 2 |

Round-Robin

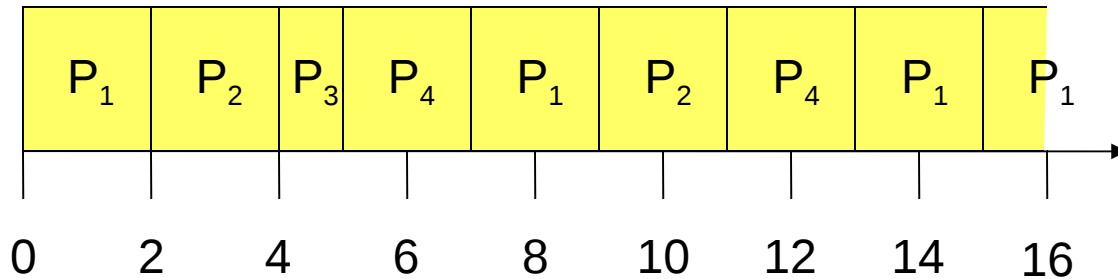


RR Scheduler

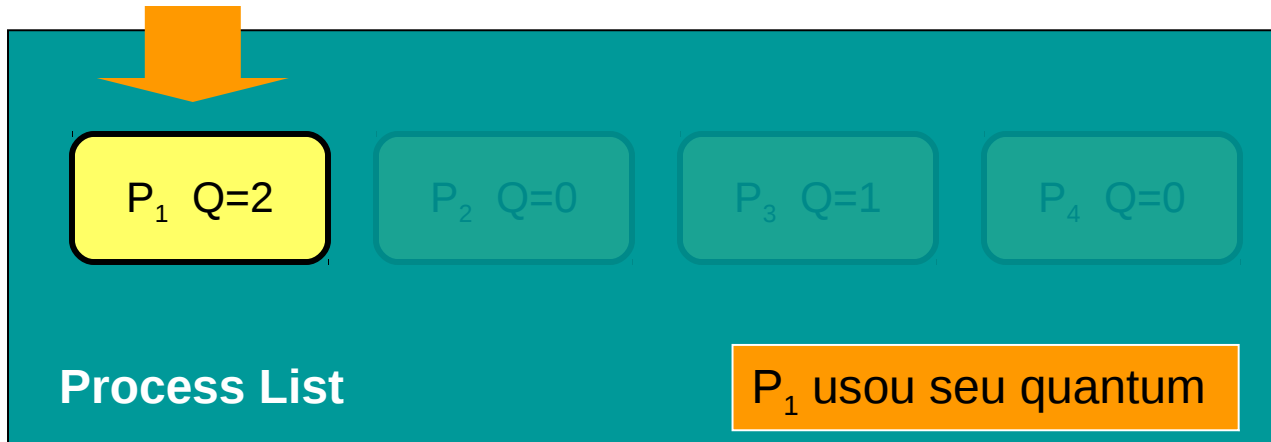


| Process | CPU Requirement |
|----------------|-----------------|
| P ₁ | 3 |
| P ₂ | 0 |
| P ₃ | 0 |
| P ₄ | 0 |

Round-Robin

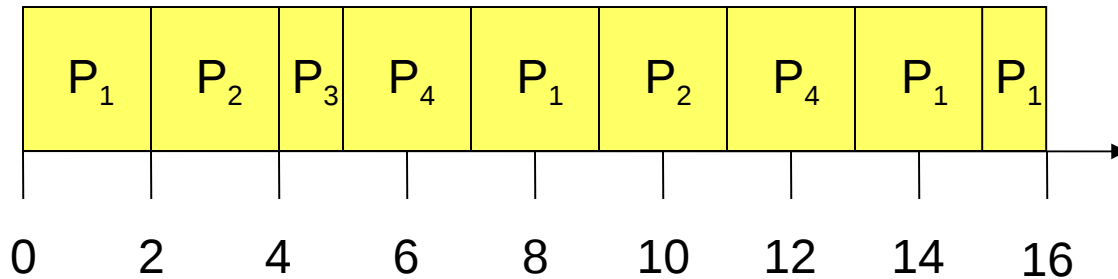


RR Scheduler

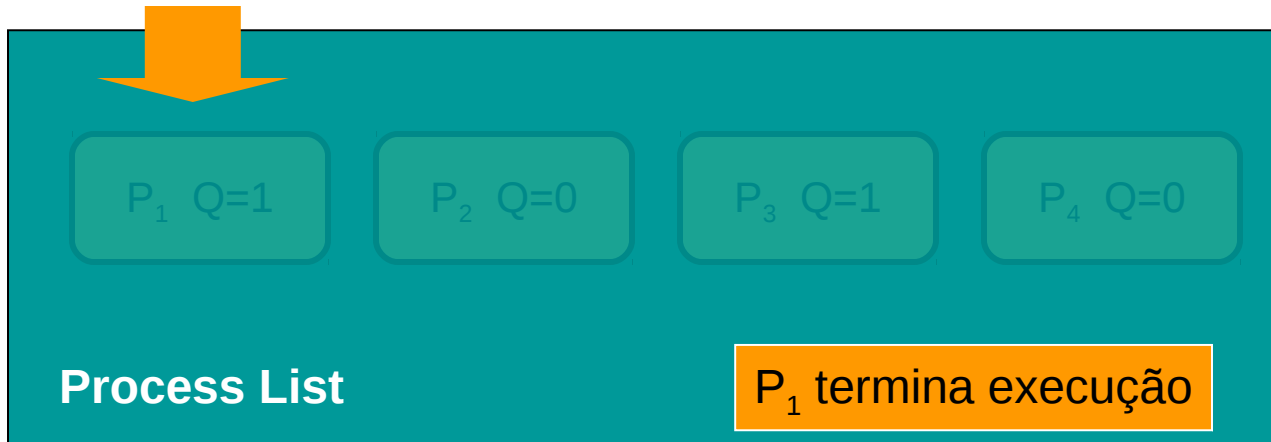


| Process | CPU Requirement |
|----------------|-----------------|
| P ₁ | 1 |
| P ₂ | 0 |
| P ₃ | 0 |
| P ₄ | 0 |

Round-Robin

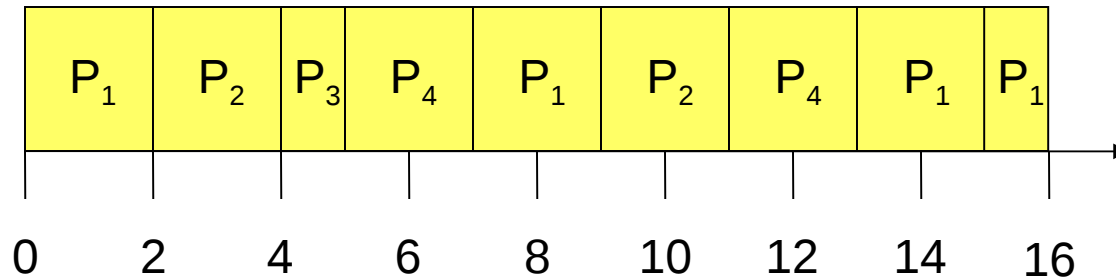


Escalonador RR



| Process | CPU Requirement |
|----------------|-----------------|
| P ₁ | 0 |
| P ₂ | 0 |
| P ₃ | 0 |
| P ₄ | 0 |

Round-Robin



Tempo de espera

$$P_1 = 9, P_2 = 7, P_3 = 4, P_4 = 9.$$

Tempo médio de espera

$$(9 + 7 + 4 + 9) / 4 = 7.25.$$

Tempo de Turnaround

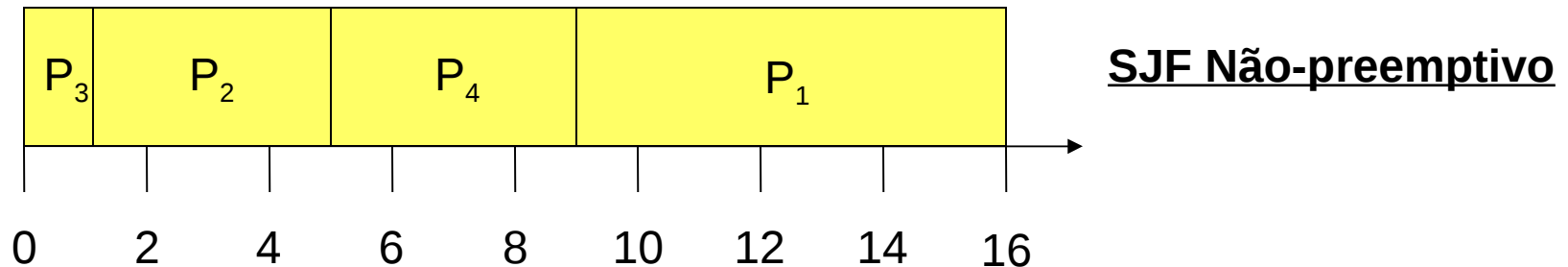
$$P_1 = 16, P_2 = 11, P_3 = 5, P_4 = 13.$$

Tempo médio de Turnaround

$$(16 + 11 + 5 + 13) / 4 = 11.25$$

* Turnaround = o tempo entre submissão e terminação.

RR vs SJF



Tempo de espera

$$P_1 = 9, P_2 = 1, P_3 = 0, P_4 = 5.$$

Tempo médio de espera

$$(9 + 1 + 0 + 5) / 4 = 3.75.$$

Tempo de Turnaround

$$P_1 = 16, P_2 = 7, P_3 = 1, P_4 = 9.$$

Tempo médio de Turnaround

$$(16 + 7 + 1 + 9) / 4 = 8.25.$$

* Turnaround = o tempo entre submissão e terminação.

RR vs SJF

- Tipicamente, RR tem um tempo de espera maior e um tempo de turnaround maior do que o SJF.
- Entretanto, a **resposta** dos processos em um escalonador RR é **mais rápida**.
 - Você não irá sentir que um job está “freeze” porque ele está executando de de tempo em tempo.
 - Adequado para jobs interativos.

Escalonamento por Prioridade

- Cada processo recebe uma **prioridade** atribuída pelo sistema.
- O escalonamento sempre escolhe o processo com a **prioridade mais alta** para executar.
- Prioridade pode ser estática ou dinâmica.
 - Prioridade estática
 - Ex., um processo de administrador recebe uma prioridade mais alta que um processo de um usuário normal.
 - Ex., podemos alterar a prioridade de um processo usando o comando “**nice**” no Unix/Linux.
 - Prioridade dinâmica
 - Ex., a prioridade de um processo cai enquanto o mesmo esta executando.

Escalonamento por prioridades

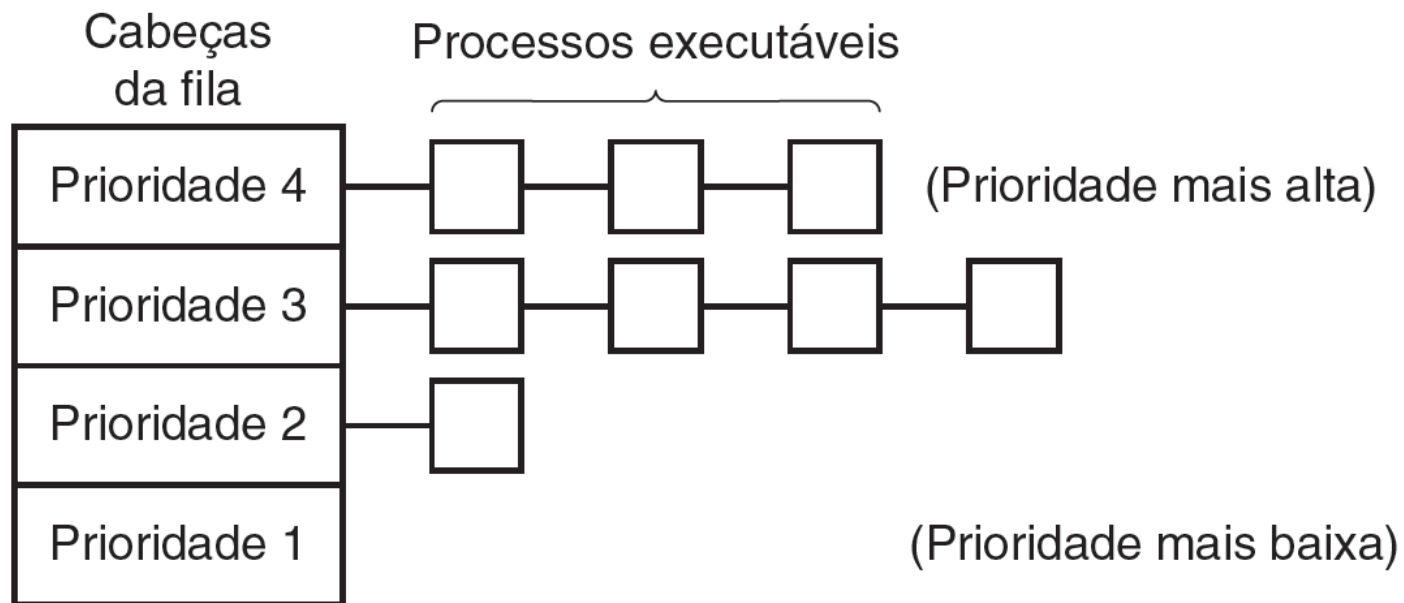
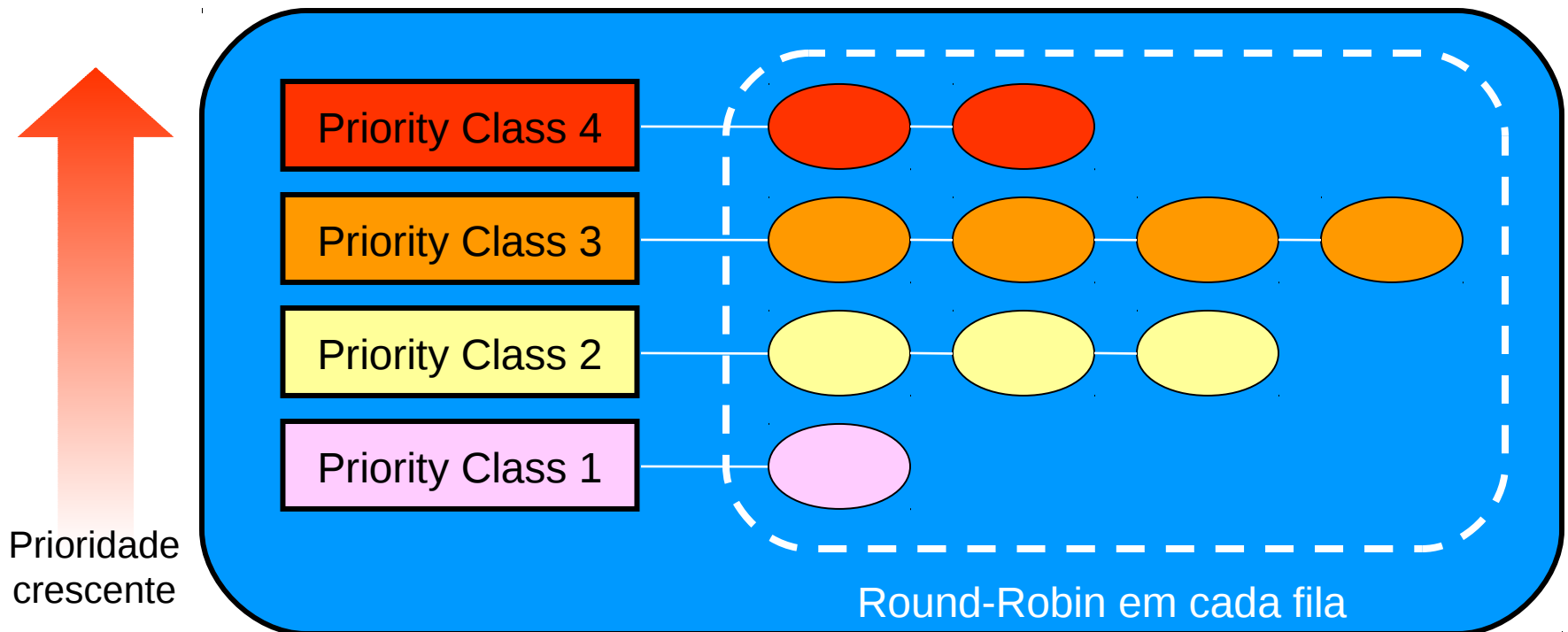


Figura 2.34 Um algoritmo de escalonamento com quatro classes de prioridade.

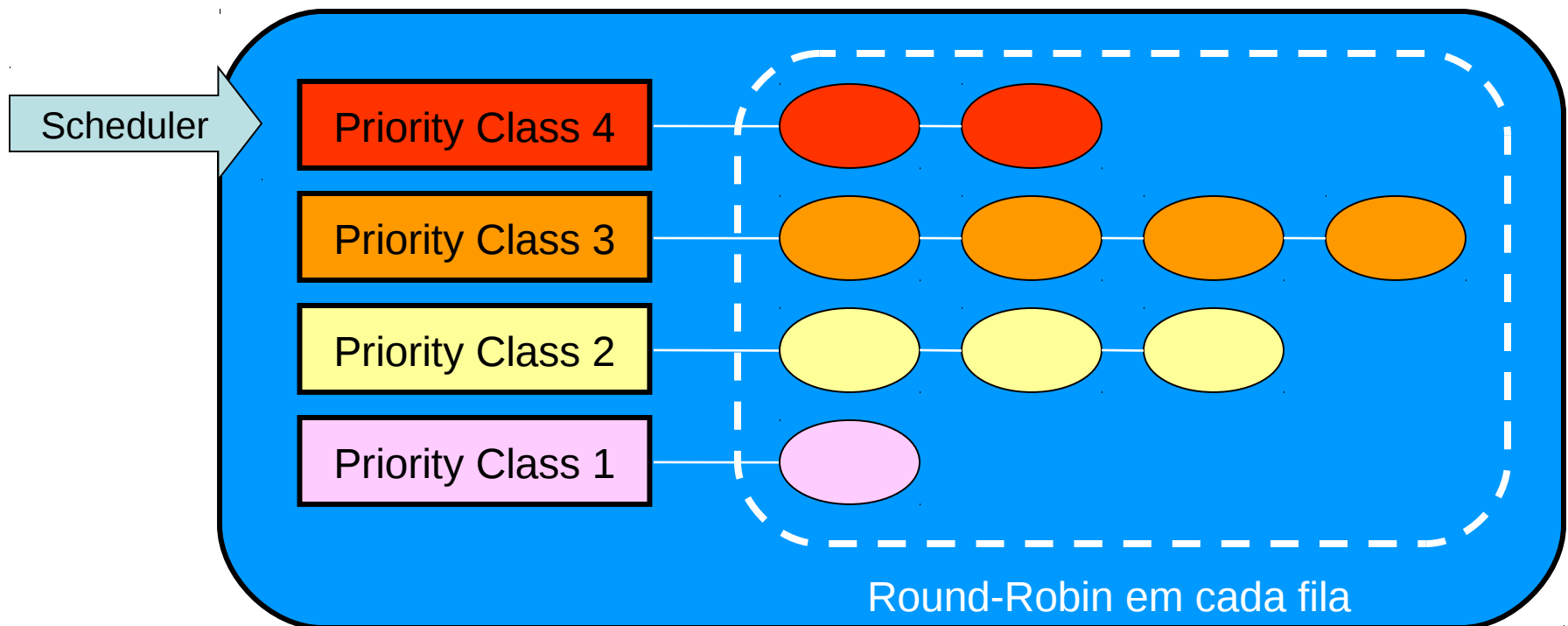
Escalonamento Prioridade Estática

- **escalonamento prioridade estática** : processos recebem uma prioridade fixa quando são submetidos ao sistema.



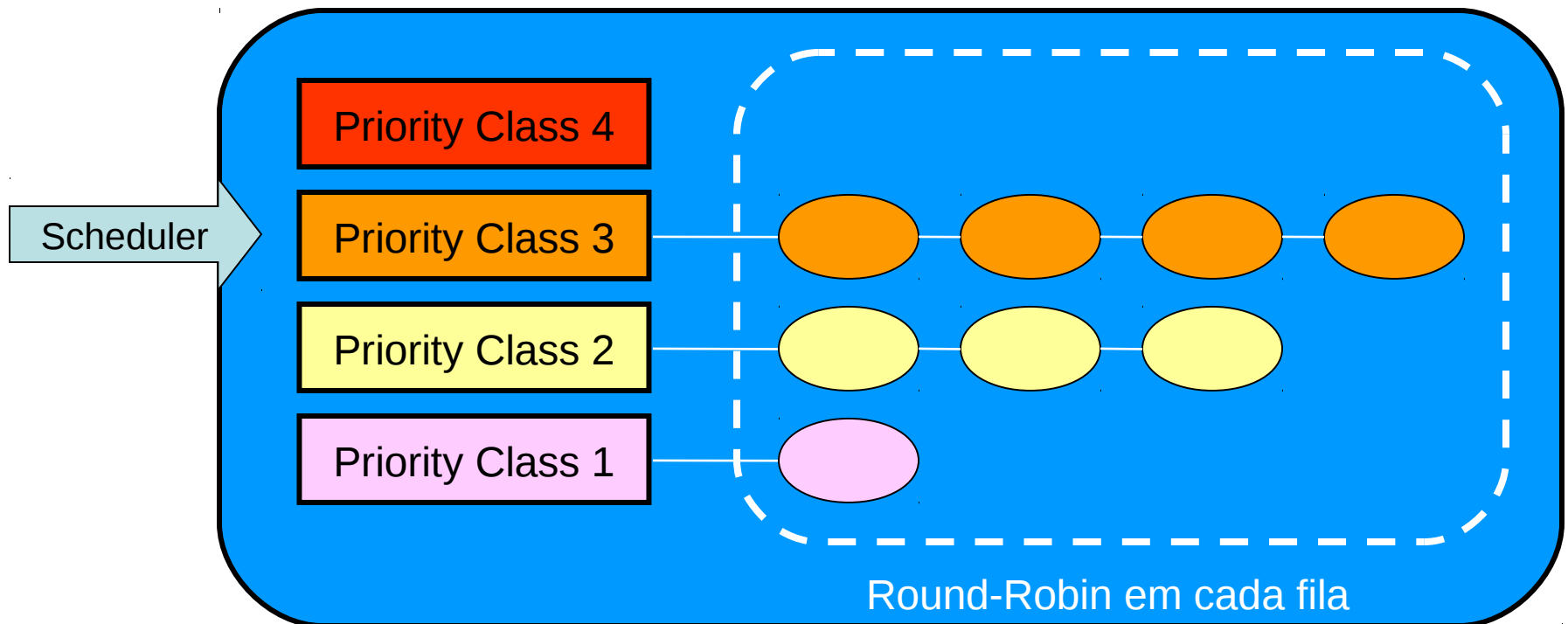
Escalonamento Prioridade Estática

- Processos na Classe 4 serão escalonados primeiro.
 - Eles devem ter vida curta.
 - Para prevenir processos de alta-prioridade de executar indefinidamente.



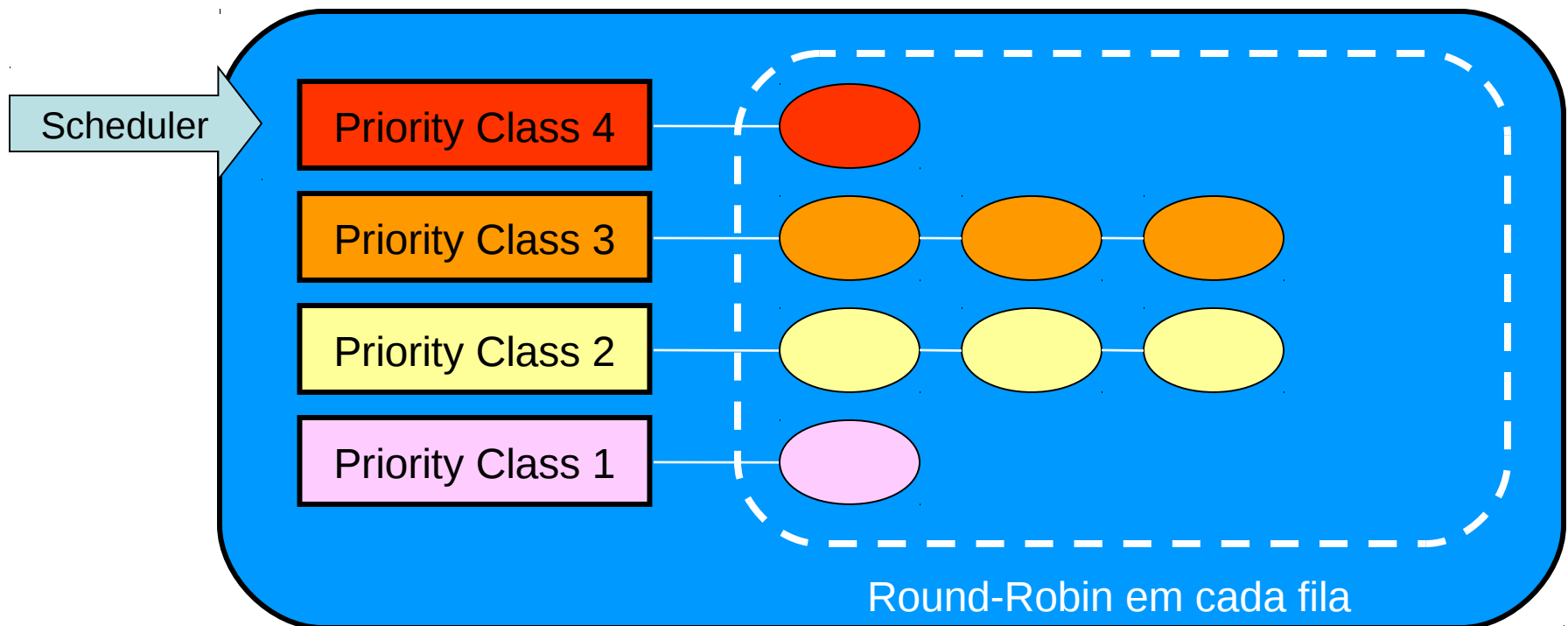
Escalonamento Prioridade Estática

- Quando não existe mais processos na Classe 4, os processos da Classe 3 serão escalonados.



Escalonamento Prioridade Estática

- Quando um processo de mais alta prioridade aparece,
 - O processo executando pode ser preemptado ou não.
 - Mas, eventualmente, o processo de mais alta prioridade deve ser escalonado.



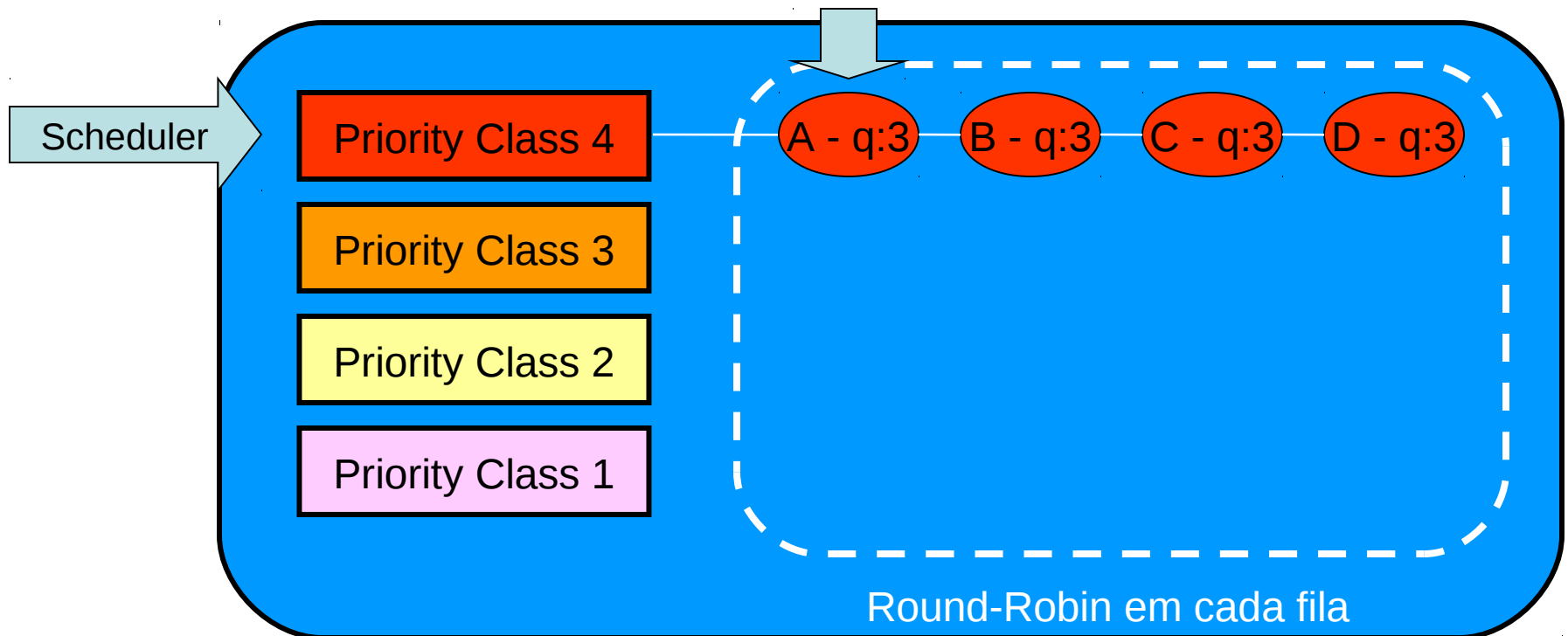
Escalonamento Prioridade Dinâmica

- Estudo de caso –
- Inicialização:
 - Todos processos iniciam na fila de mais alta prioridade;
 - Todos processos recebem o mesmo quantum, e quando os quanta são resetados, o valor é setado para ser o mesmo do valor inicial.
- Regras:
 - Um processo é preemptado somente quando seu quantum é usado ou quando o processo espera por E/S.
 - Quando um processo não esta usando a UCP, sua prioridade é mudada com base no seu quantum restante:

| |
|--|
| $\text{Nova prioridade} = \text{Num de prioridades} - \text{quantum usado.}$ |
|--|

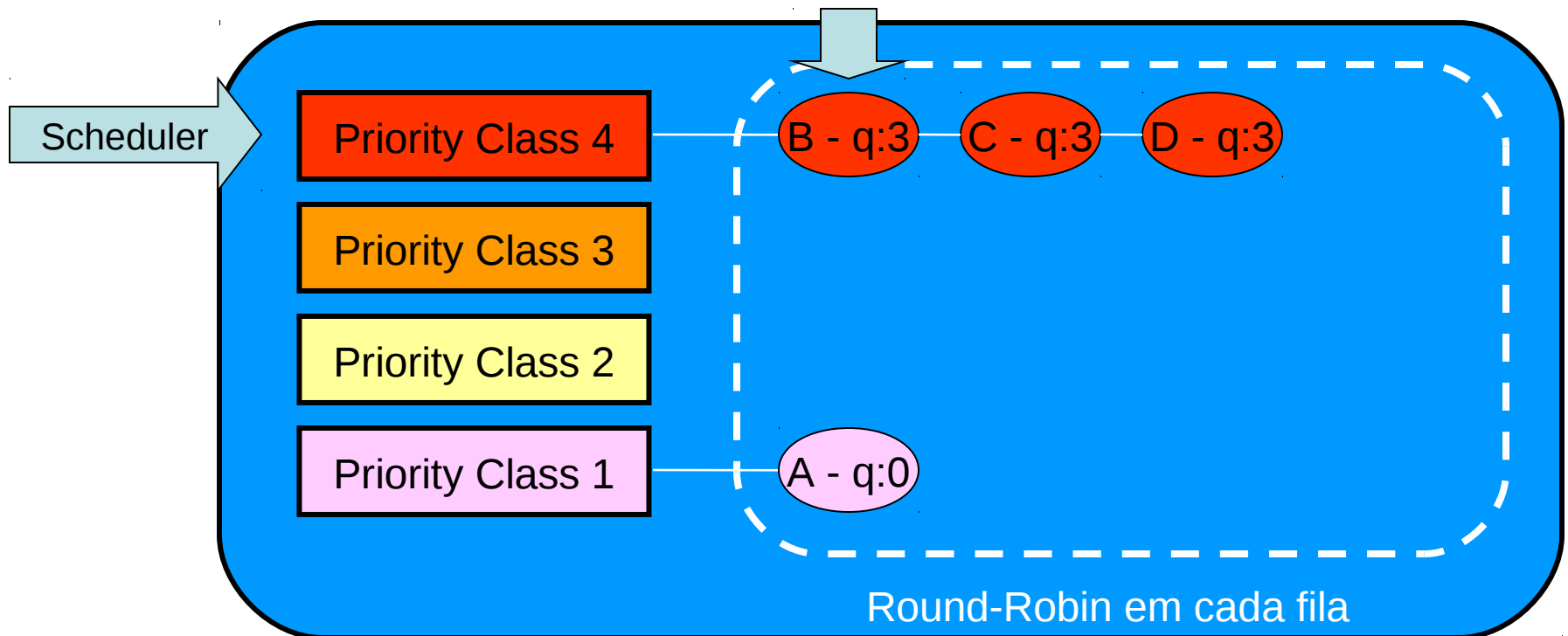
Escalonamento Prioridade Dinâmica

- Processo A é escalonado primeiro.
 - Processo A é um processo CPU-bound e irá usar todo seu quantum.



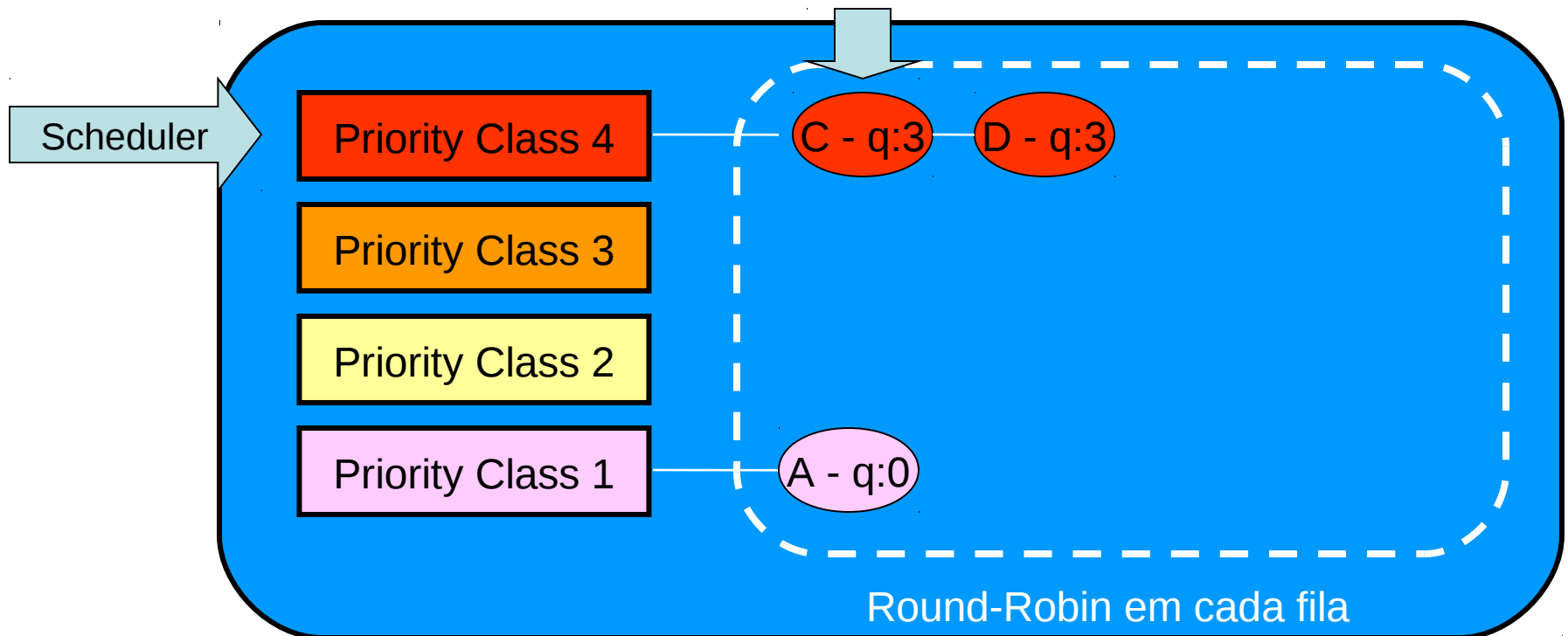
Escalonamento Prioridade Dinâmica

- Quando o Processo A esta fora da CPU, ele se torna um processo de Classe 1.
- Como existem processos na Classe 4, Processo B é escalonado.



Escalonamento Prioridade Dinâmica

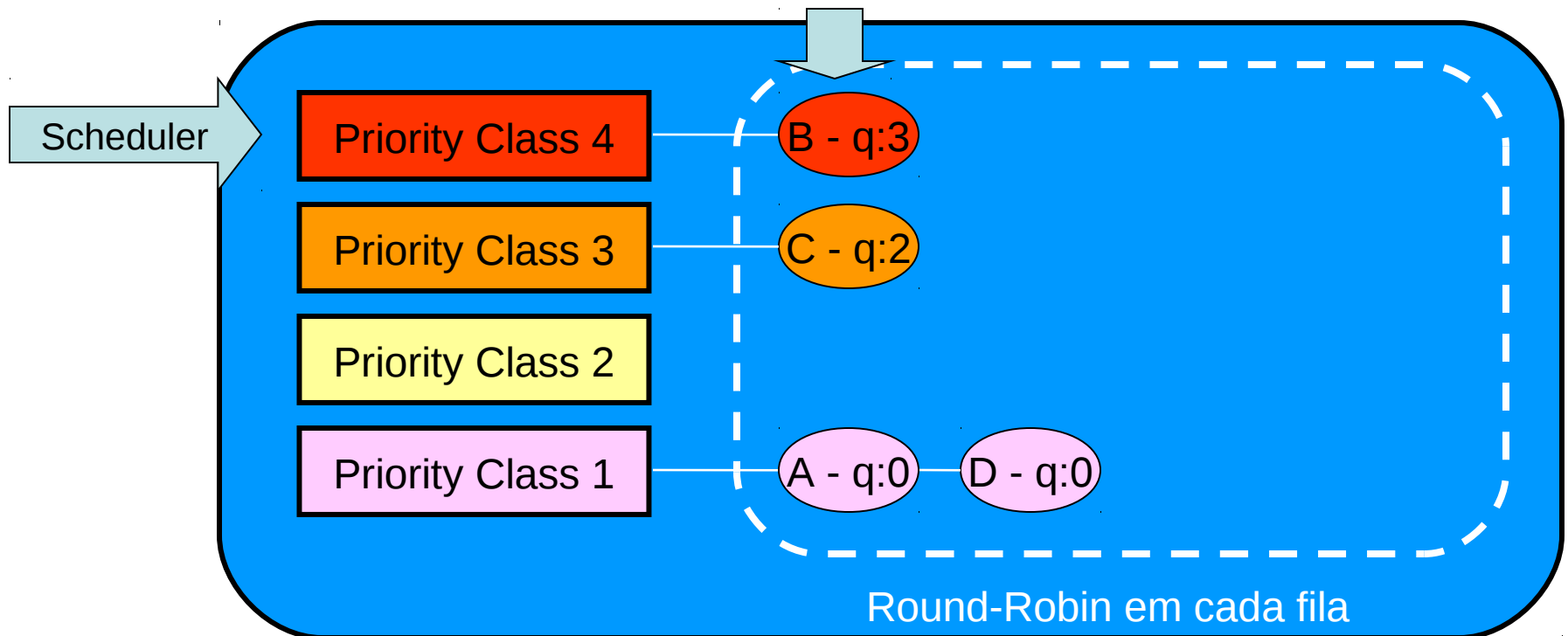
- Processo B é um processo I/O bound, e é bloqueado logo após iniciar sua execução.
 - Assim, ele é removido até que a E/S complete.



Escalonamento Prioridade Dinâmica

Processo B volta depois que o processo D termina.

Como B não usou o seu quantum antes, B é colocado na fila Classe 4.



Escalonamento Prioridade Dinâmica

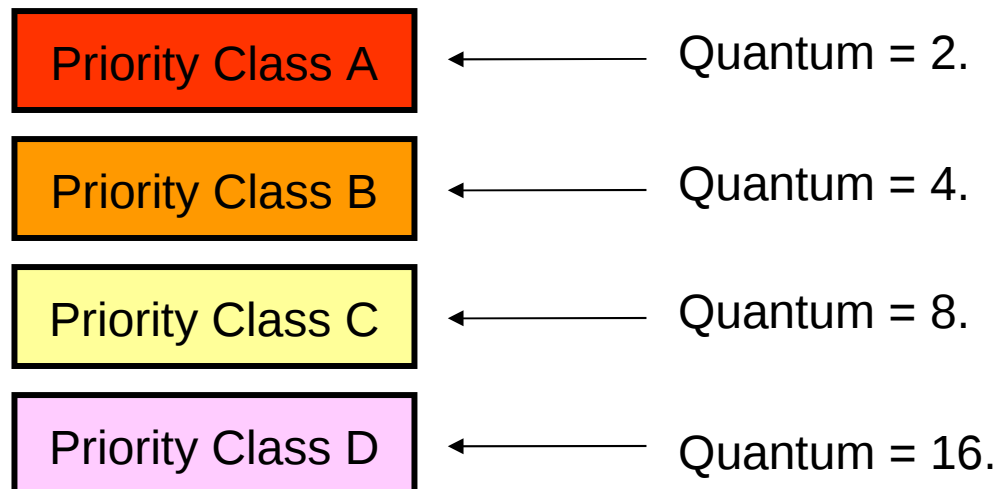
- Vantagem
 - Nenhum processo de alta prioridade pode executar por um período prolongado.
 - Processo de baixa prioridade não é **postergado indefinidamente**.
 - Processos I/O-bound tem prioridade mais alta para acessar a CPU.

Starve – “a process is starving”

Significa que o processo não tem uma chance de usar a CPU por um longo tempo.

Escalonamento com Múltiplas Filas

- Similar ao algoritmo de escalonamento por prioridade, exceto:
 - Cada fila tem sua própria política. Ex.,
 - Cada fila tem um quantum.
 - A classe de mais alta prioridade tem o menor quantum.



Escalonamento com Múltiplas Filas

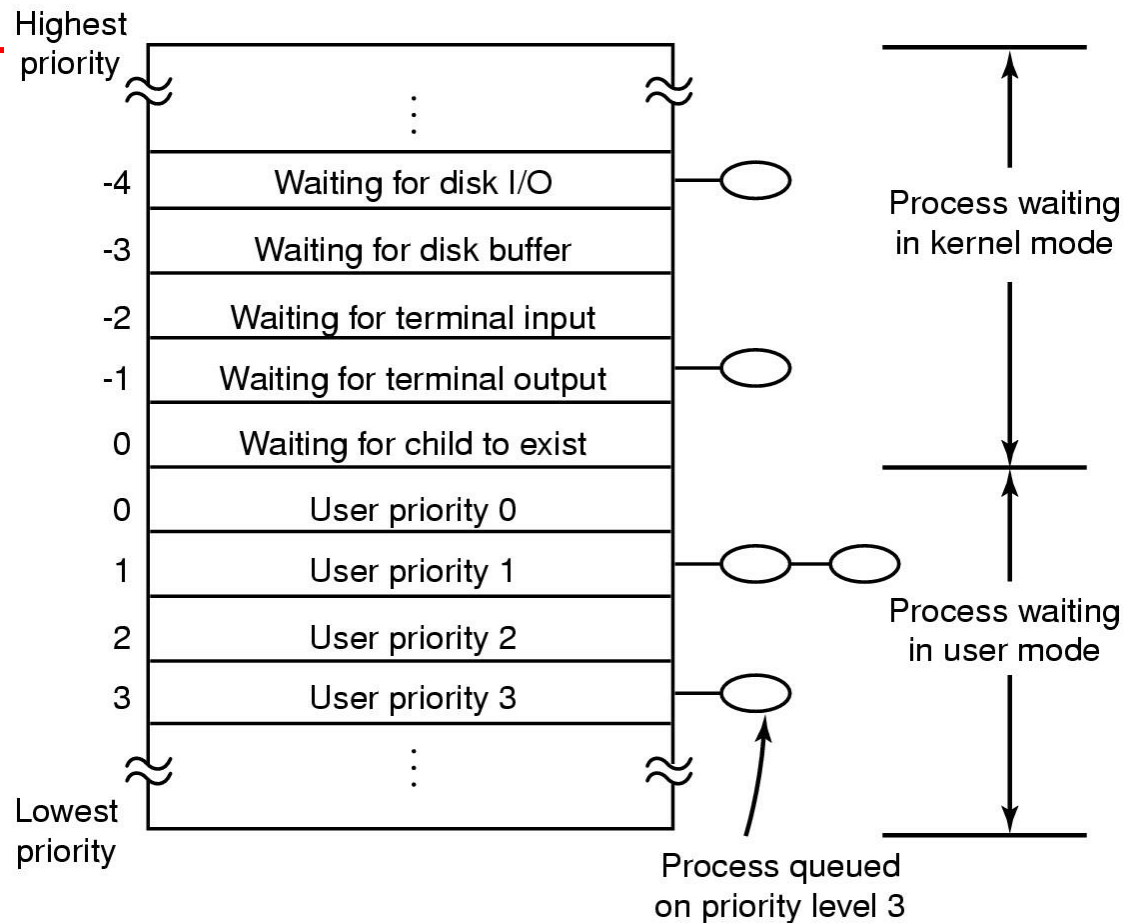
- Vantagens

- “punir” os processos CPU-bound, e dar aos processos I/O-bound a chance de executar.
- Minimizar o número de **chaveamento de contexto** para processos CPU-bound.

- Desvantagens

- Quando um processo CPU-bound torna-se um processo interativo, não tem como aumentar sua prioridade.

Escalonamento UNIX



O escalonador UNIX é baseado em uma estrutura de fila multinível