

Sistemas Operacionais

Escalonamento

Parte 2

Prof. Otávio Gomes

otavio.gomes@unifei.edu.br



Algoritmos de escalonamento

- Sistemas Batch
- Sistemas Interativos
- Sistemas Tempo Real



Algoritmo de escalonamento

Sistemas Interativos



Algoritmo de Escalonamento

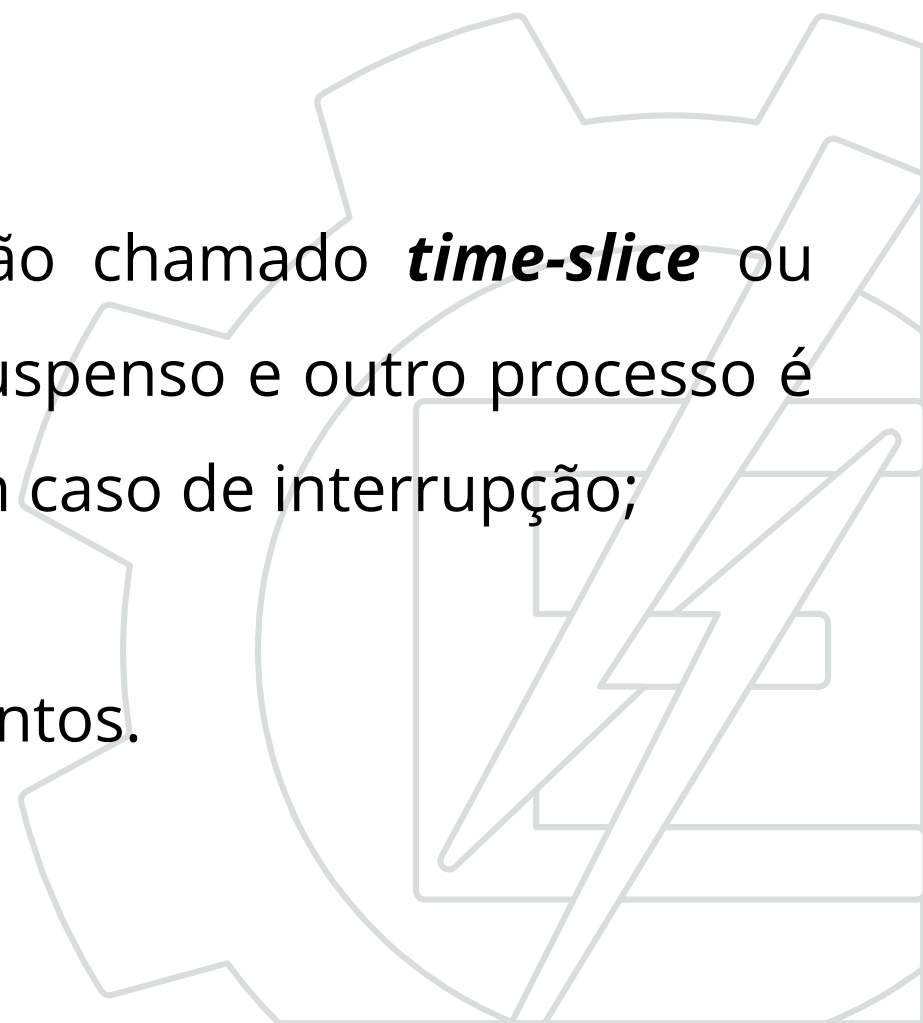
Interativos

Possuem o objetivo de tempo de resposta, resposta rápida às requisições; proporcionalidade (*time-sharing*); satisfazer as expectativas dos usuários.

1. *Round-Robin*
2. Prioridade (Múltiplas filas)
3. Múltiplas filas com realimentação
4. SPN (*Shortest Process Next*)
5. Garantido
6. Loteria
7. *Fair-Share*



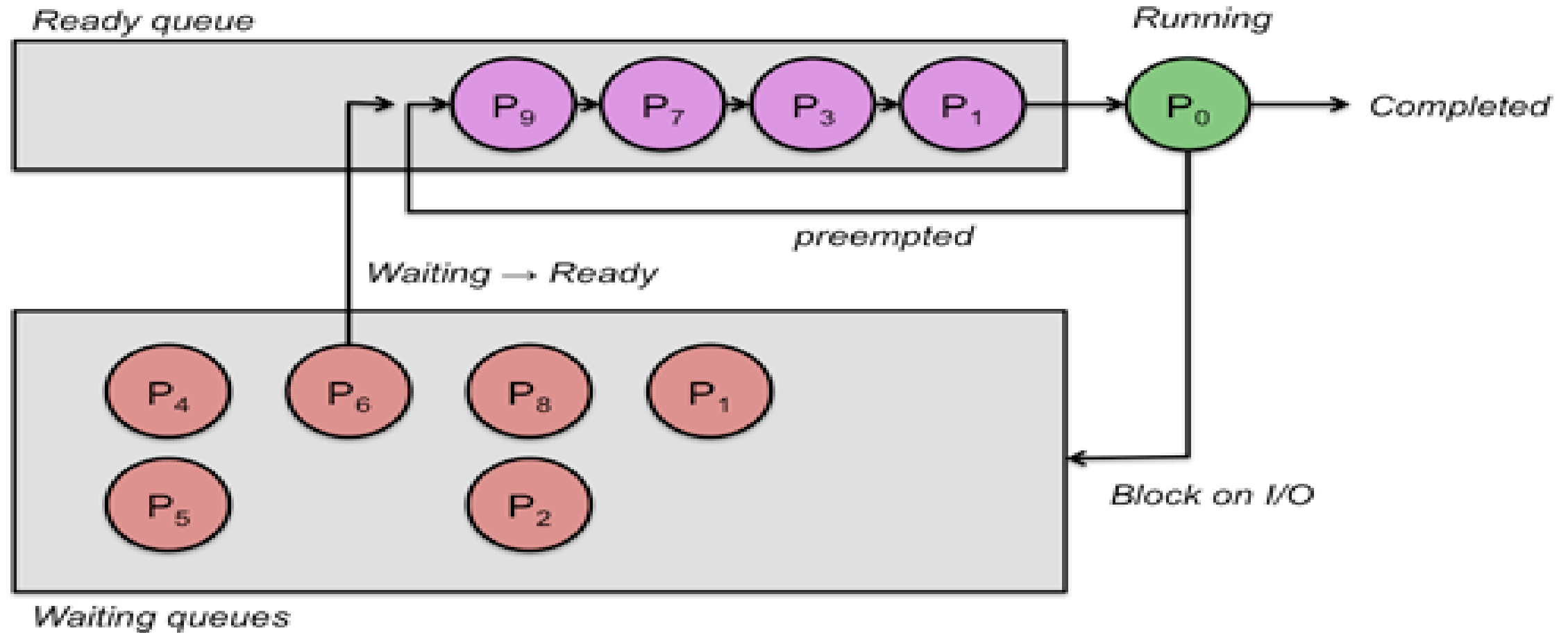
1. *Round-Robin*

- Antigo, mais simples e mais utilizado;
 - **Preemptivo**;
 - Cada processo recebe um tempo de execução chamado ***time-slice*** ou ***quantum***. Ao final desse tempo o processo é suspenso e outro processo é colocado em execução. Também é suspenso em caso de interrupção;
 - Troca de contexto frequente (***quantum***);
 - Escalonador mantém uma fila de processos prontos.
- 

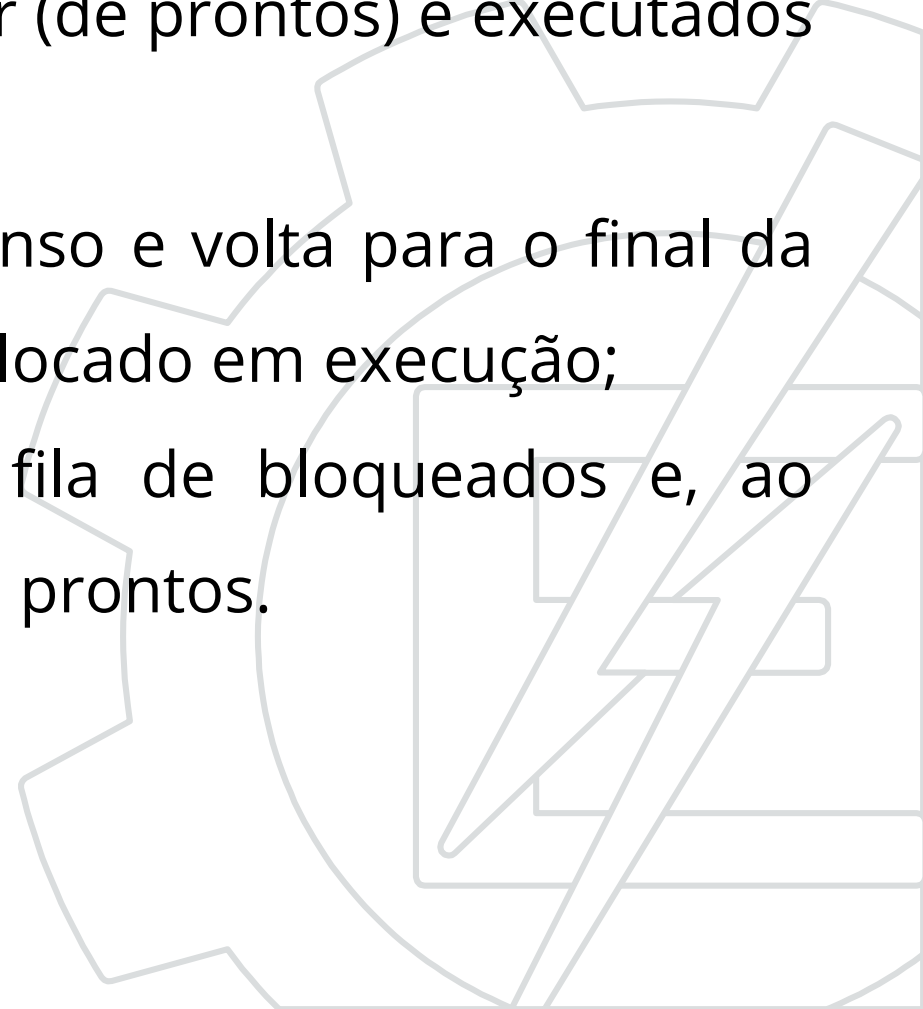
Algoritmo de Escalonamento

Interativos

1. Round-Robin



1. Round-Robin

- Os processos são colocados em uma fila circular (de prontos) e executados um a um;
 - Quando seu tempo acaba, o processo é suspenso e volta para o final da fila. Outro processo (primeiro da fila) é então colocado em execução;
 - Quando o processo solicita E/S, vai para a fila de bloqueados e, ao terminar a operação, volta para o final da fila de prontos.
- 

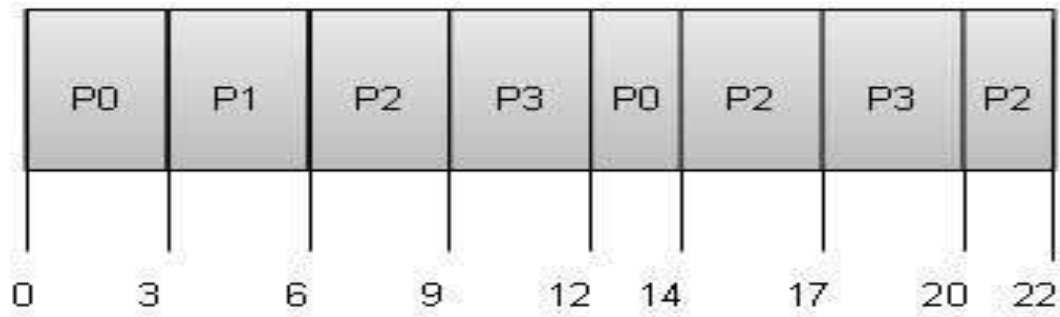
Algoritmo de Escalonamento

Interativos

1. Round-Robin

| Process | Arrival Time | Execute Time |
|---------|--------------|--------------|
| P0 | 0 | 5 |
| P1 | 1 | 3 |
| P2 | 2 | 8 |
| P3 | 3 | 6 |

Quantum = 3



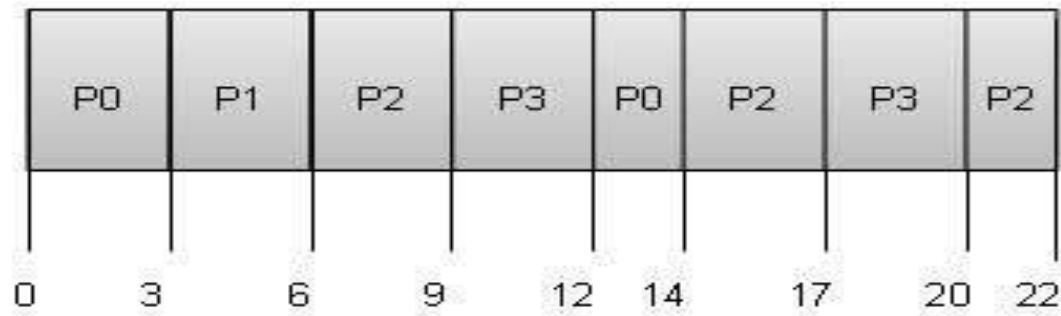
Algoritmo de Escalonamento

Iterativos

1. Round-Robin

| Process | Arrival Time | Execute Time |
|---------|--------------|--------------|
| P0 | 0 | 5 |
| P1 | 1 | 3 |
| P2 | 2 | 8 |
| P3 | 3 | 6 |

Quantum = 3

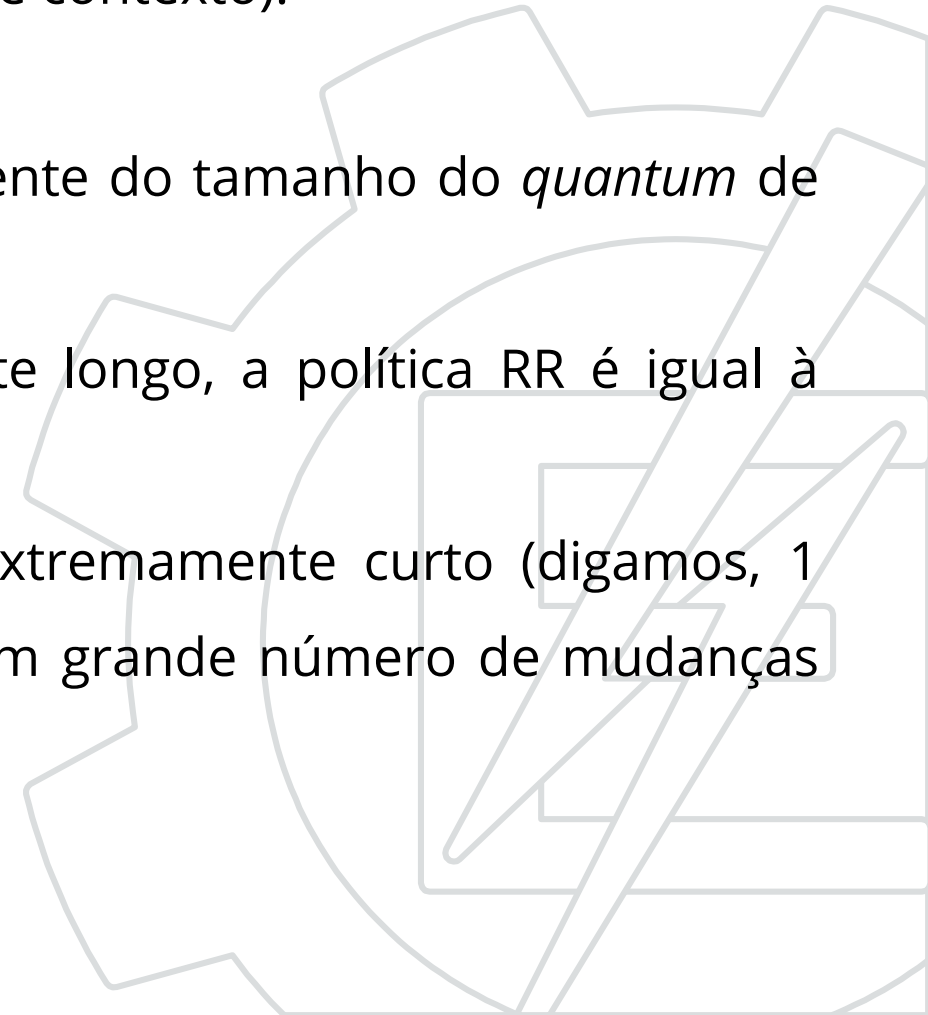


| Processo | Tempo de espera |
|----------|---------------------------------------|
| P0 | $(0 - 0) + (12 - 3) = 9$ |
| P1 | $(3 - 1) = 2$ |
| P2 | $(6 - 2) + (14 - 9) + (20 - 17) = 12$ |
| P3 | $(9 - 3) + (17 - 12) = 11$ |

Tempo médio de espera:
 $(9+2+12+11) / 4 = 8,5$

1. *Round-Robin*

- **Problema:** tempo de chaveamento de processos (troca de contexto).
- O desempenho do algoritmo RR depende substancialmente do tamanho do *quantum* de tempo.
 - Por um lado, se o *quantum* de tempo é extremamente longo, a política RR é igual à política FCFS.
 - Por outro lado, quando o *quantum* de tempo é extremamente curto (digamos, 1 milissegundo), a abordagem RR pode resultar em um grande número de mudanças de contexto.



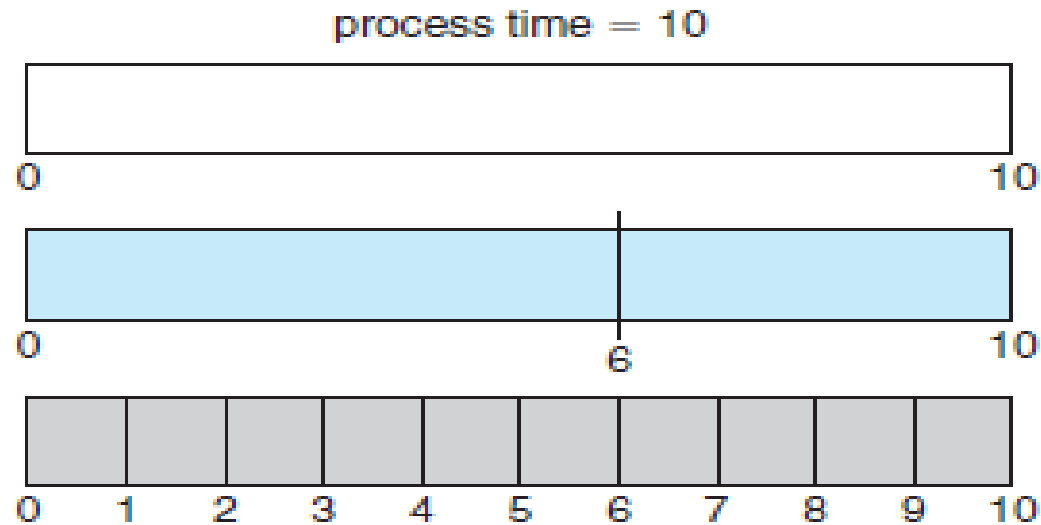
Algoritmo de Escalonamento

Interativos

1. Round-Robin

- **Quantum**

- Se for muito pequeno, ocorrem muitas trocas diminuindo, assim, a eficiência da CPU;
- Se for muito longo, o tempo de resposta é comprometido.



quantum

12

6

1

context
switches

0

1

9

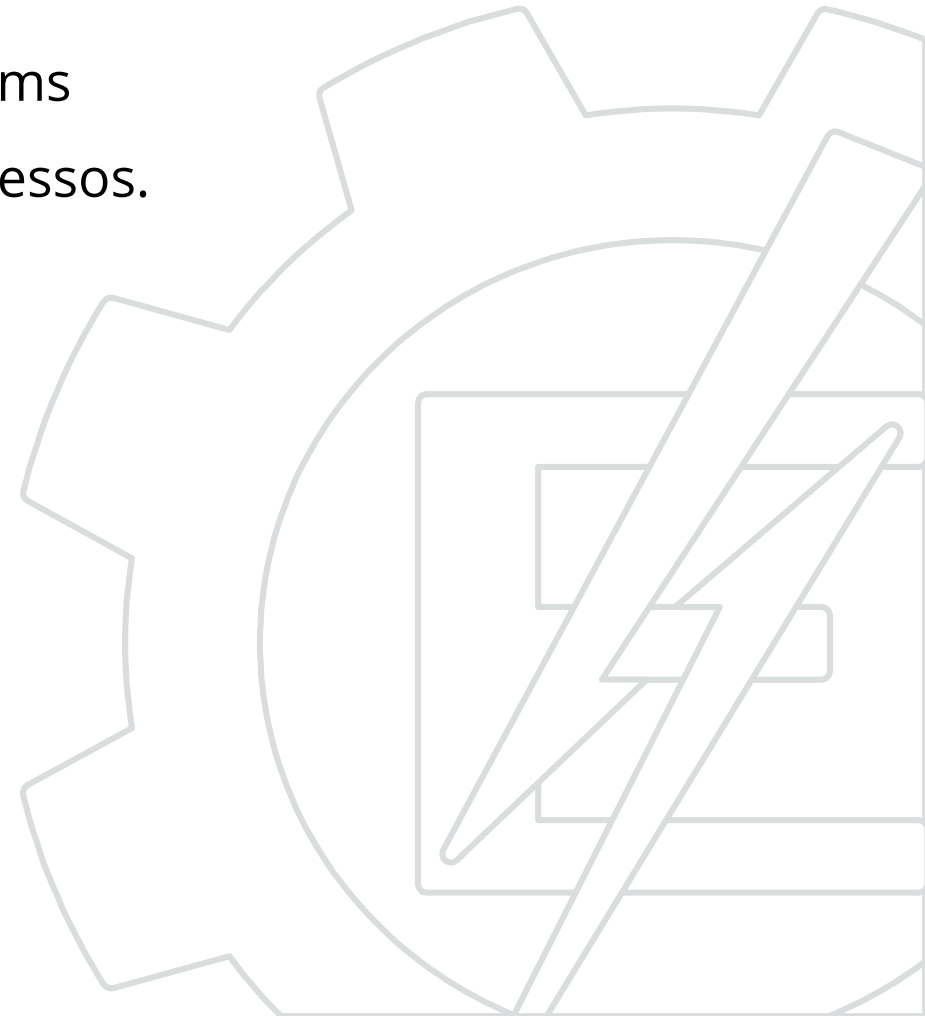
- Dilema: *quantum* - pequeno ou grande?

Algoritmo de Escalonamento

Interativos

1. *Round-Robin*

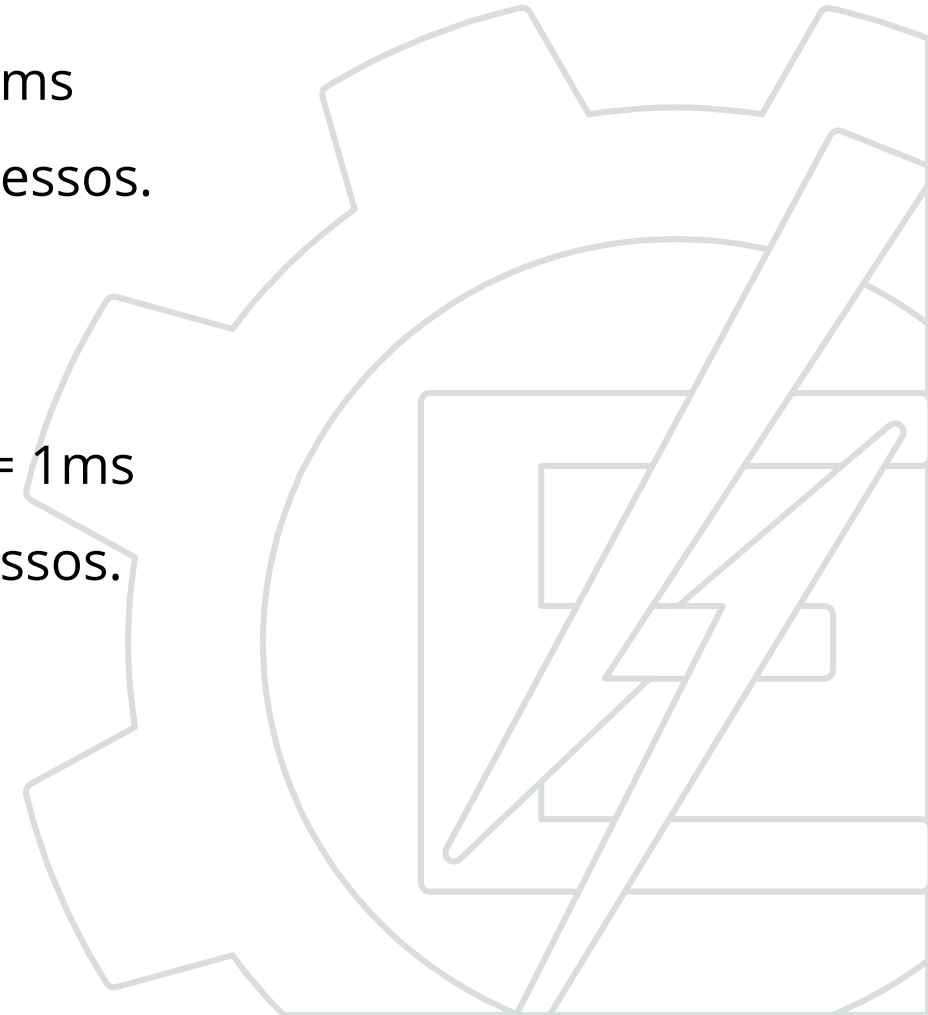
- ***Quantum:***
 - 1ª abordagem: $t = 4\text{ms}$ (*quantum*) e troca de contexto = 1ms
 - 25% do tempo de CPU é consumido na troca de processos.
 - Menor eficiência



1. *Round-Robin*

- ***Quantum:***

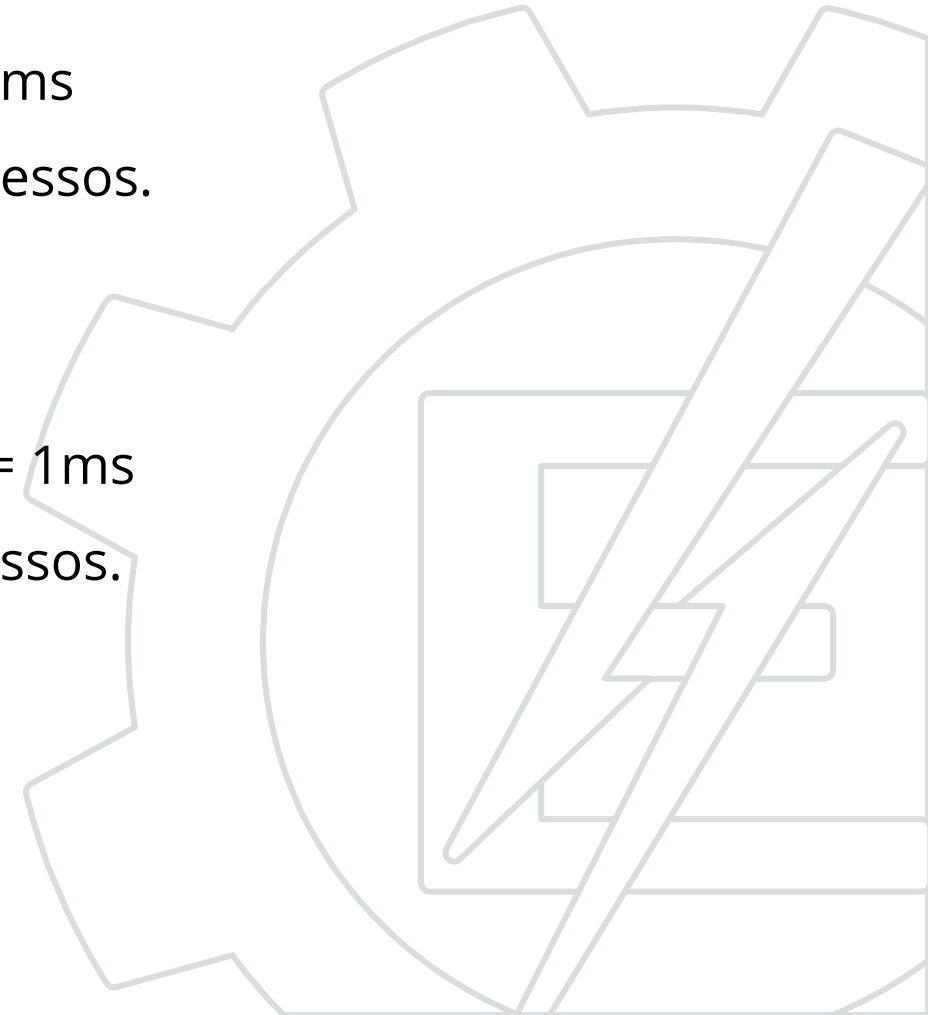
- 1ª abordagem: $t = 4\text{ms}$ (*quantum*) e troca de contexto = 1ms
 - 25% do tempo de CPU é consumido na troca de processos.
 - Menor eficiência
- 2ª abordagem: $t = 100\text{ms}$ (*quantum*) e troca de contexto = 1ms
 - 1% do tempo de CPU é consumido na troca de processos.



1. *Round-Robin*

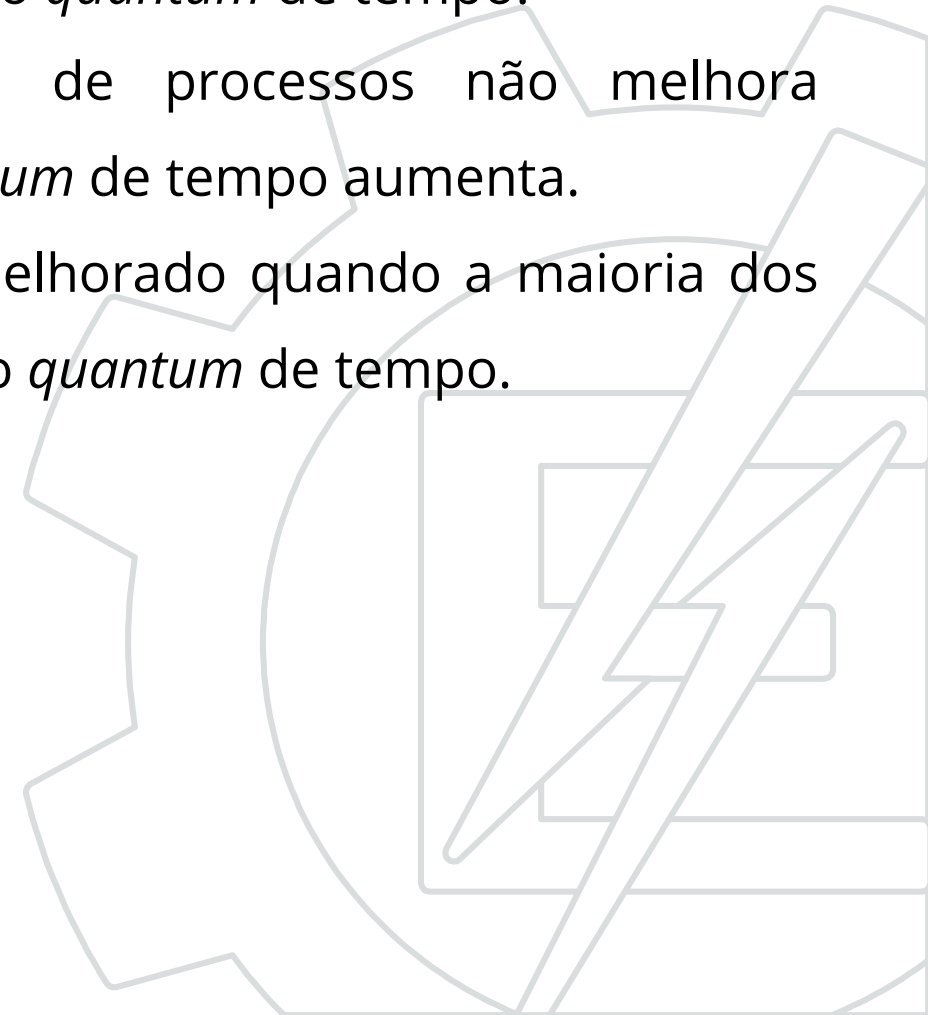
- ***Quantum:***

- 1ª abordagem: $t = 4\text{ms}$ (*quantum*) e troca de contexto = 1ms
 - 25% do tempo de CPU é consumido na troca de processos.
 - Menor eficiência
- 2ª abordagem: $t = 100\text{ms}$ (*quantum*) e troca de contexto = 1ms
 - 1% do tempo de CPU é consumido na troca de processos.
- Valor razoável: $t = 20\text{-}50\text{ms}$ (*quantum*)



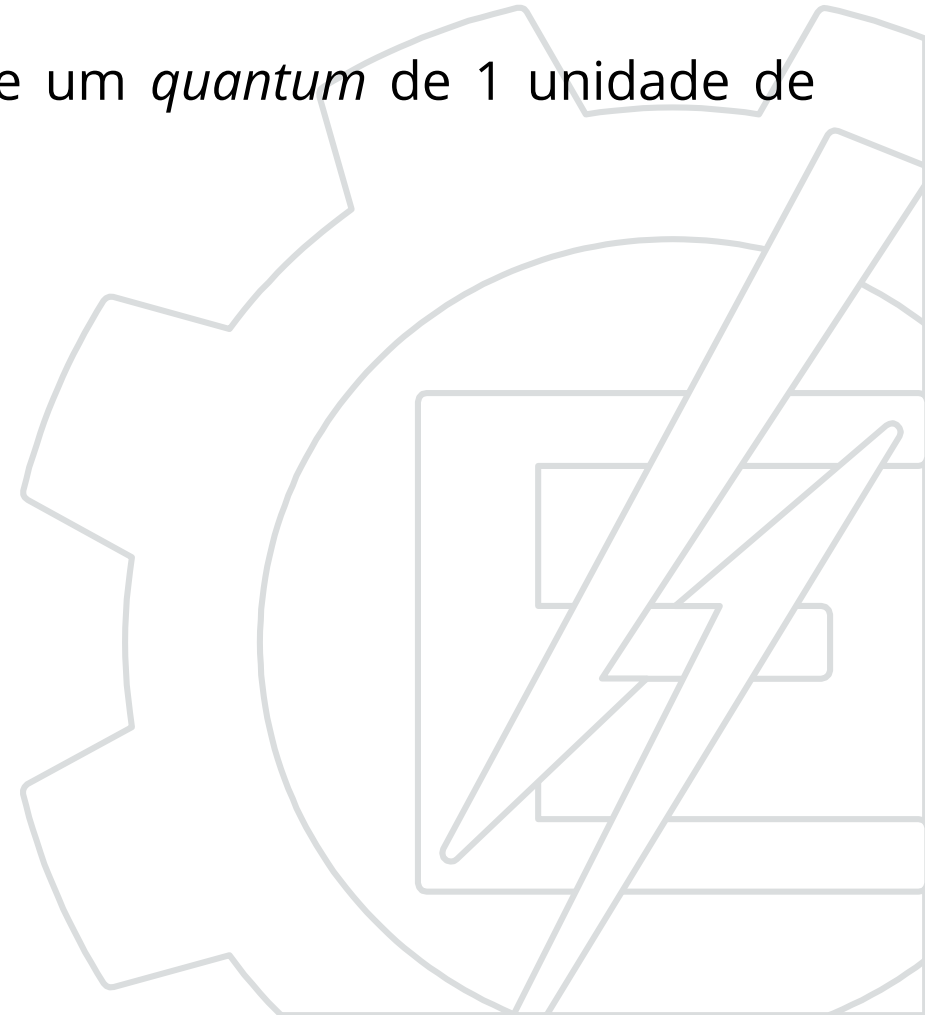
1. *Round-Robin*

- O tempo de ***turnaround*** também depende do tamanho do *quantum* de tempo.
- O tempo médio de turnaround de um conjunto de processos não melhora necessariamente na medida em que o tamanho do *quantum* de tempo aumenta.
- Geralmente, o tempo médio de *turnaround* pode ser melhorado quando a maioria dos processos termina seu próximo pico de CPU em um único *quantum* de tempo.



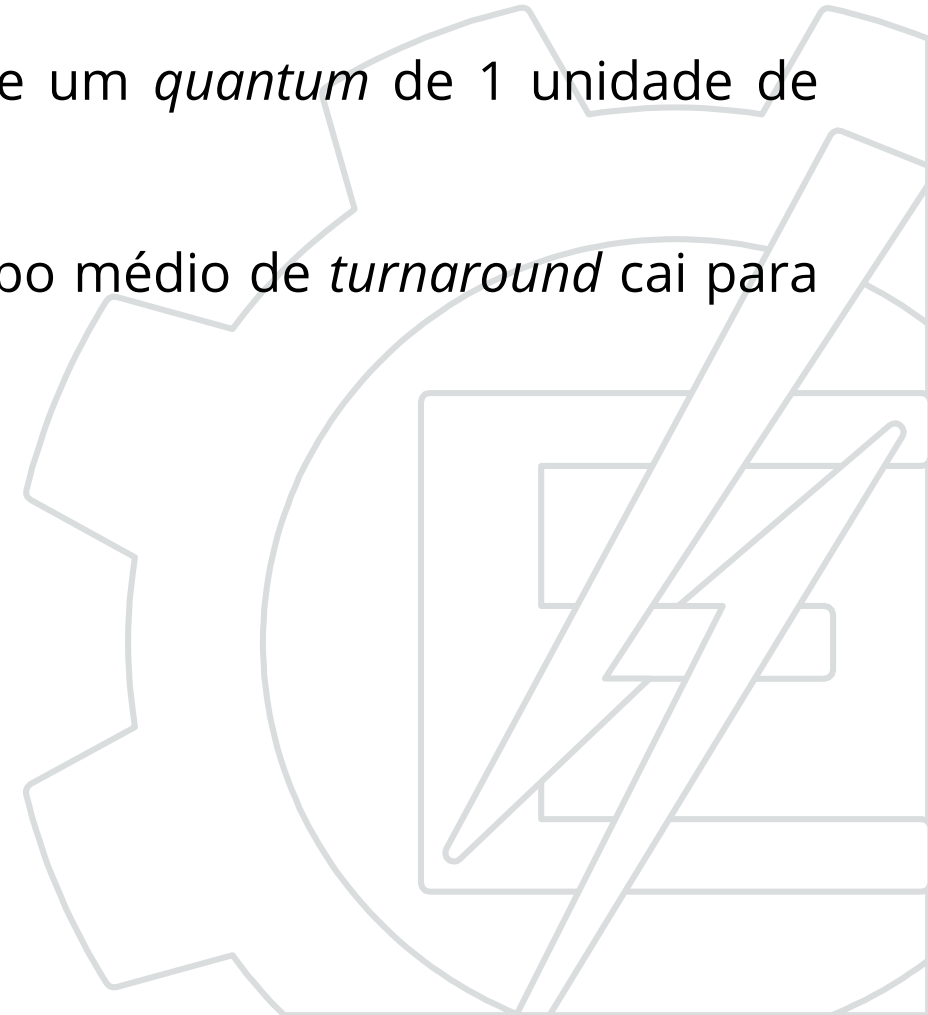
1. *Round-Robin*

- ***Turnaround*** – Exemplo:
 - Dados três processos de 10 unidades de tempo cada e um *quantum* de 1 unidade de tempo, o tempo médio de *turnaround* é de 29.



1. *Round-Robin*

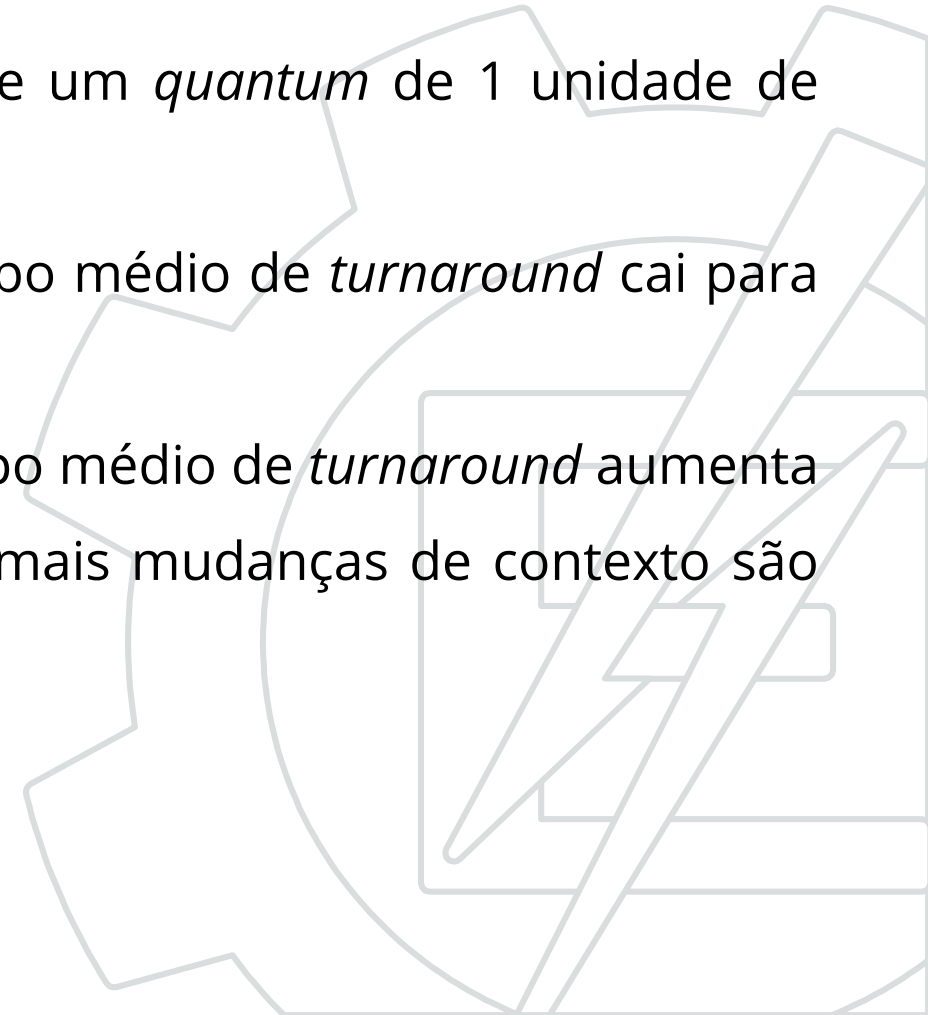
- ***Turnaround*** – Exemplo:
 - Dados três processos de 10 unidades de tempo cada e um *quantum* de 1 unidade de tempo, o tempo médio de *turnaround* é de 29.
 - Se o *quantum* de tempo é igual a 10, no entanto, o tempo médio de *turnaround* cai para 20.



1. **Round-Robin**

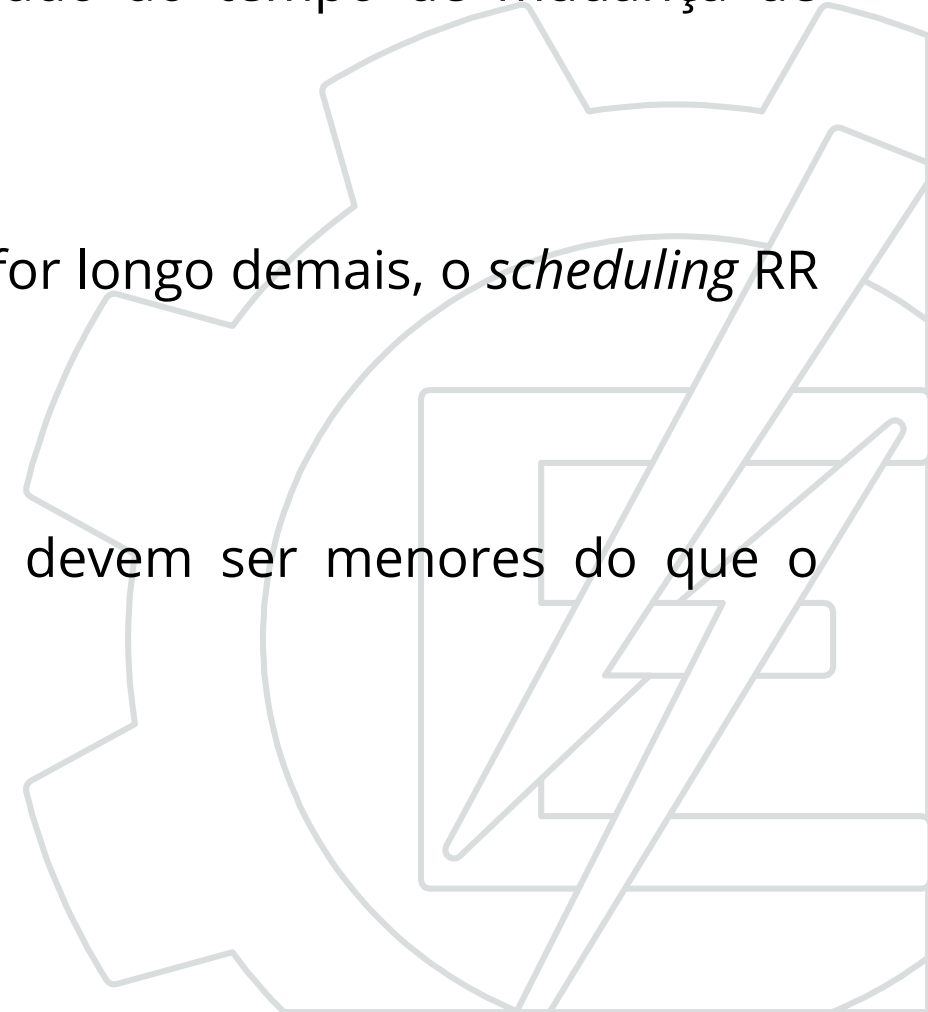
- **Turnaround** – Exemplo:

- Dados três processos de 10 unidades de tempo cada e um *quantum* de 1 unidade de tempo, o tempo médio de *turnaround* é de 29.
- Se o *quantum* de tempo é igual a 10, no entanto, o tempo médio de *turnaround* cai para 20.
- Se o tempo de mudança de contexto for incluído, o tempo médio de *turnaround* aumenta ainda mais para um *quantum* de tempo menor, já que mais mudanças de contexto são necessárias



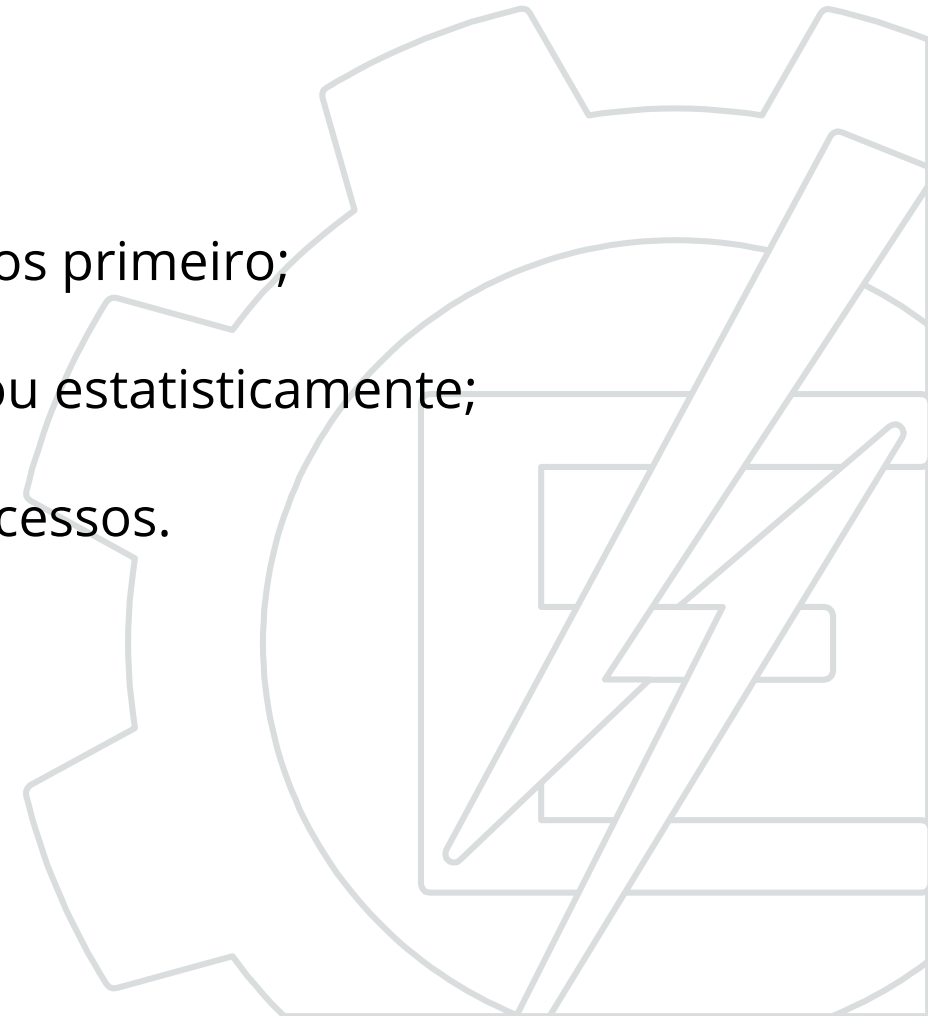
1. **Round-Robin**

- Embora o *quantum* de tempo deva ser longo, comparado ao tempo de mudança de contexto, ele não deve ser longo demais.
- Como apontado anteriormente, se o *quantum* de tempo for longo demais, o *scheduling* RR degenerará para uma política FCFS.
- Uma regra prática é a de que 80% dos picos de CPU devem ser menores do que o *quantum* de tempo.



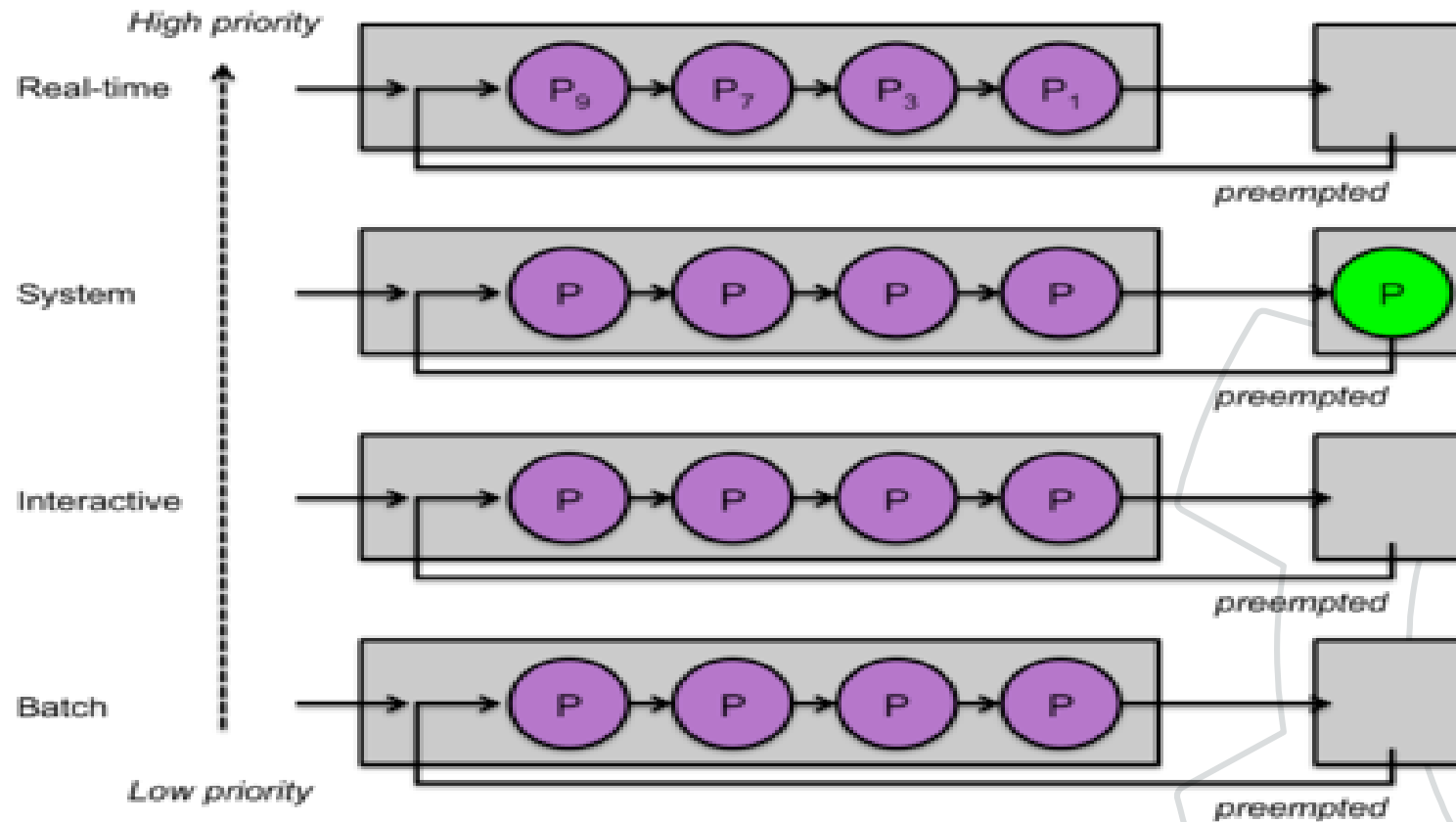
2. Prioridade (Múltiplas filas)

- Preemptivo;
- Cada processo possui uma prioridade;
- Os processos prontos com maior prioridade são executados primeiro;
- Prioridades são atribuídas dinamicamente (pelo sistema) ou estatisticamente;
- *Round-Robin* pressupõe igual prioridade para todos os processos.



Algoritmo de Escalonamento Interativos

2. Prioridade (Múltiplas filas)

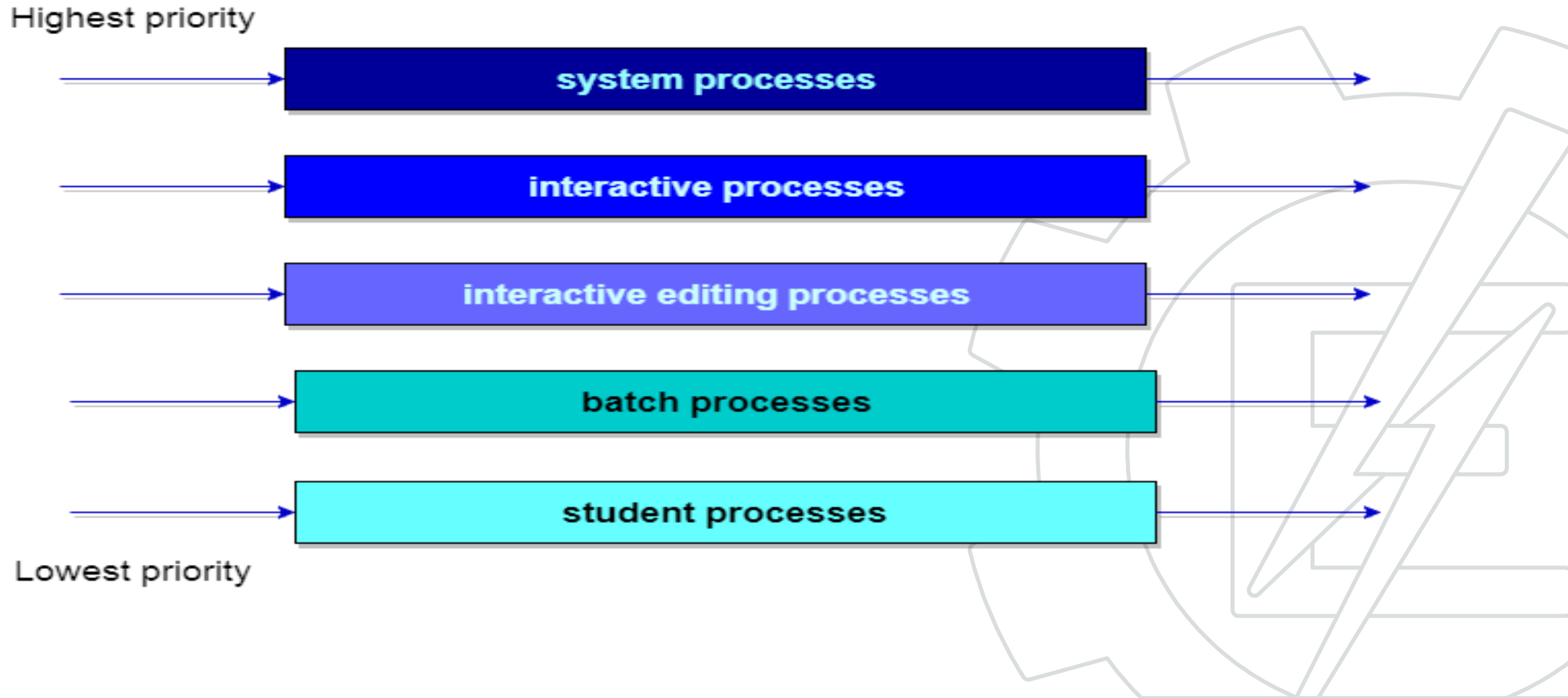


```
top - 19:00:06 up 7:47, 1 user, load average: 0.65, 0.57, 0.51
Tasks: 198 total, 2 running, 196 sleeping, 0 stopped, 0 zombie
%Cpu(s): 12.6 us, 0.6 sy, 0.0 ni, 86.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 11894.0 total, 1511.5 free, 5763.0 used, 4619.4 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 5706.3 avail Mem
```

| PID | USER | PR | NI | VIRT | RES | SHR | S | %CPU | %MEM | TIME+ | COMMAND |
|-------|------|-----|-----|---------|--------|--------|---|------|------|-----------|-----------------|
| 2844 | root | 20 | 0 | 4169800 | 1.9g | 152908 | R | 46.8 | 16.4 | 126:45.88 | Web Content |
| 2758 | root | 20 | 0 | 2560304 | 462792 | 162424 | S | 5.6 | 3.8 | 49:01.37 | firefox-esr |
| 1383 | root | 20 | 0 | 521120 | 113500 | 82664 | S | 0.3 | 0.9 | 12:35.23 | Xorg |
| 2494 | root | 20 | 0 | 6347740 | 1.6g | 37592 | S | 0.3 | 13.7 | 31:22.93 | java |
| 3030 | root | 20 | 0 | 625936 | 50372 | 31888 | S | 0.3 | 0.4 | 0:34.02 | gnome-terminal- |
| 11209 | root | -51 | 0 | 17868 | 3504 | 3028 | R | 0.3 | 0.0 | 0:00.03 | top |
| 1 | root | 20 | 0 | 202592 | 8988 | 6760 | S | 0.0 | 0.1 | 0:17.10 | systemd |
| 2 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.02 | kthreadd |
| 3 | root | 0 | -20 | 0 | 0 | 0 | I | 0.0 | 0.0 | 0:00.00 | rcu_gp |
| 5 | root | 0 | -20 | 0 | 0 | 0 | I | 0.0 | 0.0 | 0:00.48 | kworker/0:0H |
| 7 | root | 0 | -20 | 0 | 0 | 0 | I | 0.0 | 0.0 | 0:00.00 | mm_percpu_wq |
| 8 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.35 | ksoftirqd/0 |
| 9 | root | 20 | 0 | 0 | 0 | 0 | I | 0.0 | 0.0 | 0:12.91 | rcu_sched |
| 10 | root | 20 | 0 | 0 | 0 | 0 | I | 0.0 | 0.0 | 0:00.00 | rcu_bh |
| 11 | root | rt | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.01 | migration/0 |
| 12 | root | rt | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.08 | watchdog/0 |
| 13 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 | cpuhp/0 |
| 14 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 | cpuhp/1 |
| 15 | root | rt | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.09 | watchdog/1 |
| 16 | root | rt | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 | migration/1 |
| 17 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.71 | ksoftirqd/1 |
| 19 | root | 0 | -20 | 0 | 0 | 0 | I | 0.0 | 0.0 | 0:00.00 | kworker/1:0H |
| 20 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 | cpuhp/2 |
| 21 | root | rt | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.09 | watchdog/2 |
| 22 | root | rt | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.01 | migration/2 |
| 23 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.38 | ksoftirqd/2 |
| 25 | root | 0 | -20 | 0 | 0 | 0 | I | 0.0 | 0.0 | 0:00.00 | kworker/2:0H |
| 26 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 | cpuhp/3 |
| 27 | root | rt | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.08 | watchdog/3 |
| 28 | root | rt | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.01 | migration/3 |
| 29 | root | 20 | 0 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.31 | ksoftirqd/3 |
| 31 | root | 0 | -20 | 0 | 0 | 0 | I | 0.0 | 0.0 | 0:00.00 | kworker/3:0H |

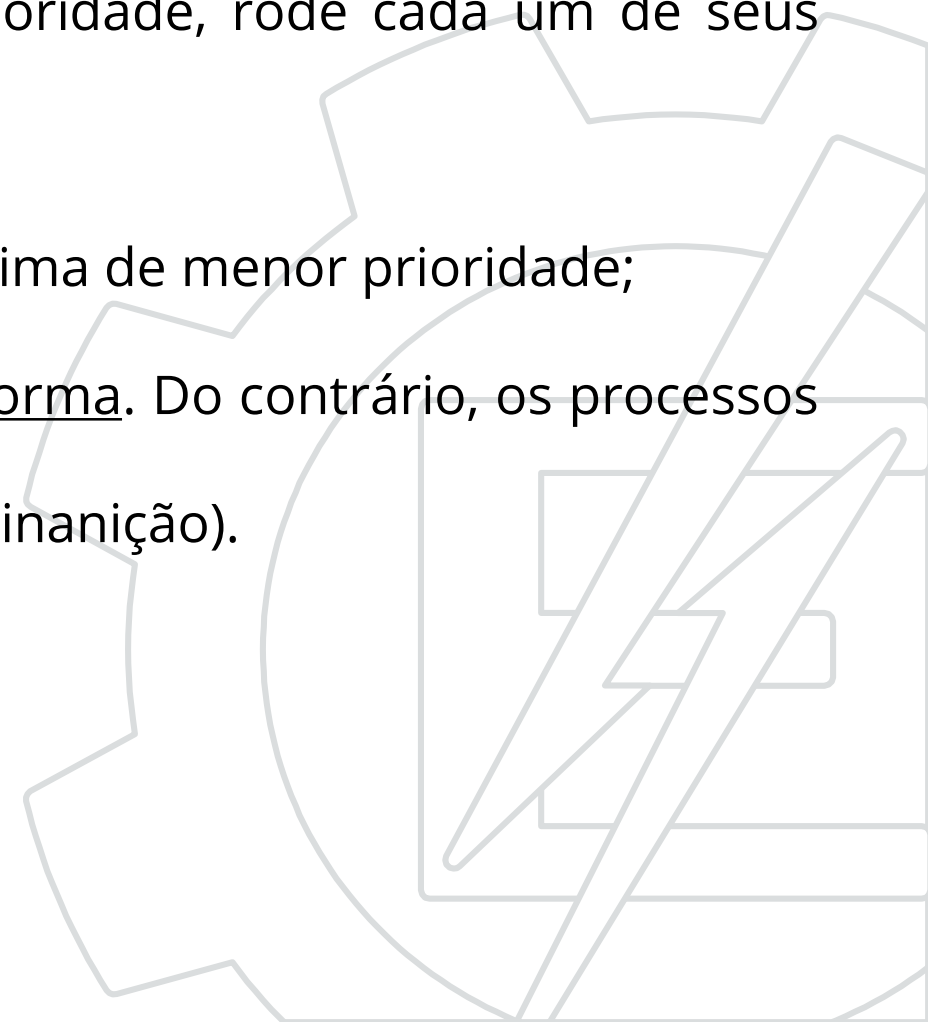
S - Em série
I - Interativos
R - Tempo real

2. Prioridade (Múltiplas filas)



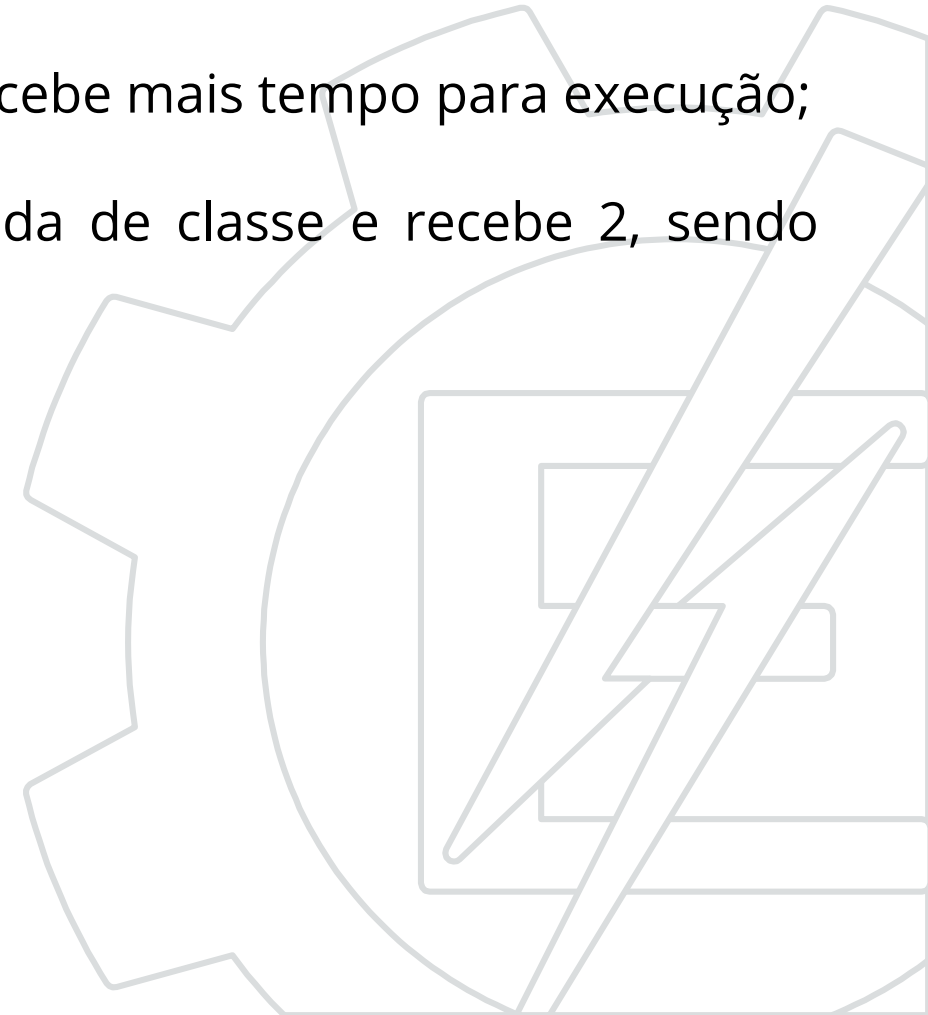
2. Prioridade (Múltiplas filas)

- Enquanto houver processos na classe (fila) de maior prioridade, rode cada um de seus processos usando *Round-Robin*;
- Se essa classe não tiver mais processos: passe para a próxima de menor prioridade;
- É necessário realizar o ajuste das prioridades de alguma forma. Do contrário, os processos nas filas menos prioritárias podem nunca ser executados (inanição).

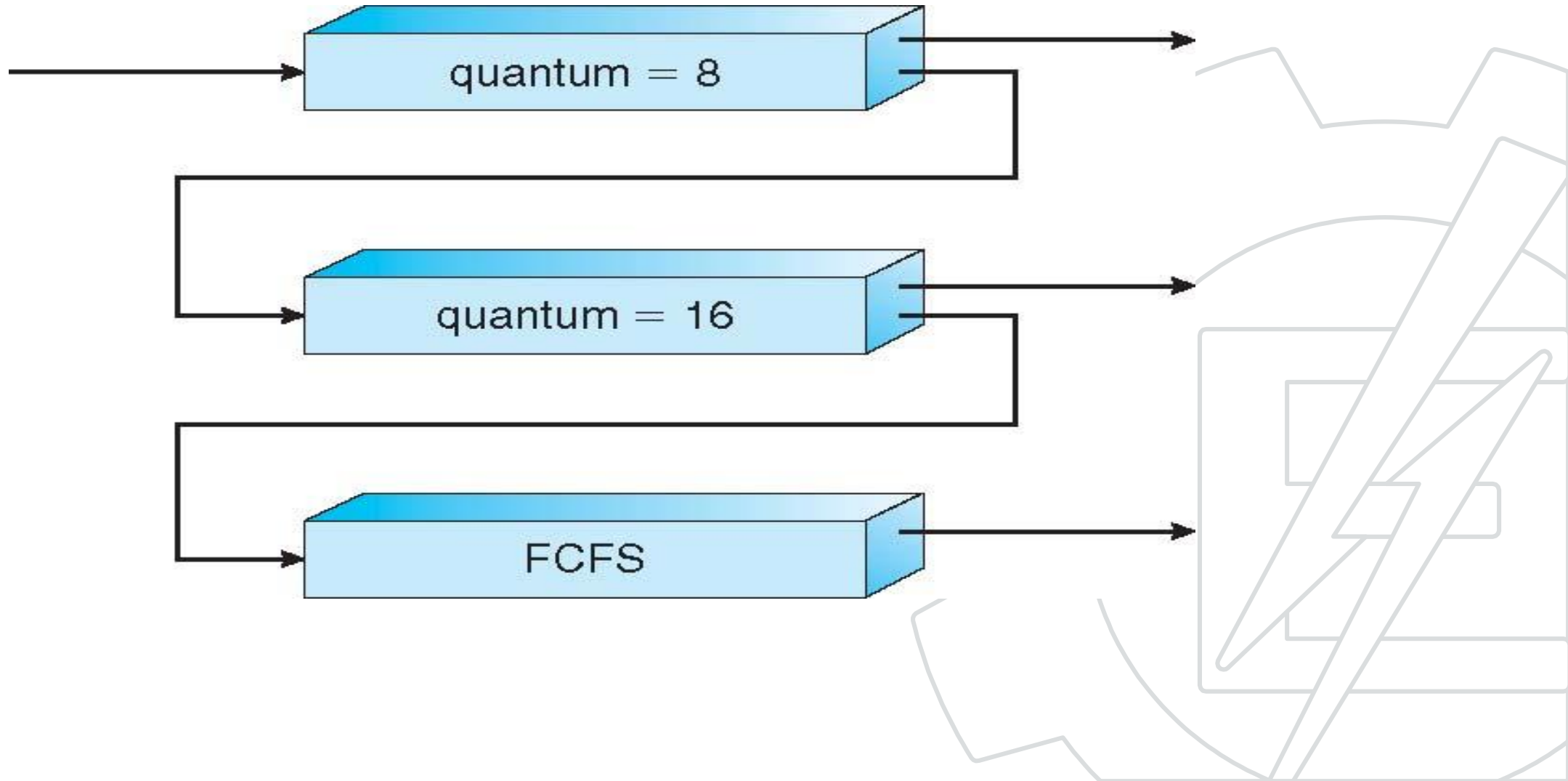


3. Múltiplas filas com realimentação

- Preemptivo;
- Cada vez que um processo é executado e suspenso, ele recebe mais tempo para execução;
- Inicialmente recebe 1 *quantum* e é suspenso; então muda de classe e recebe 2, sendo suspenso; e, assim, ocorre sucessivamente.
- Reduz o número de trocas de processo:
 - Os processos mais curtos terminam logo.
 - Aos mais longos é dado mais tempo, progressivamente.

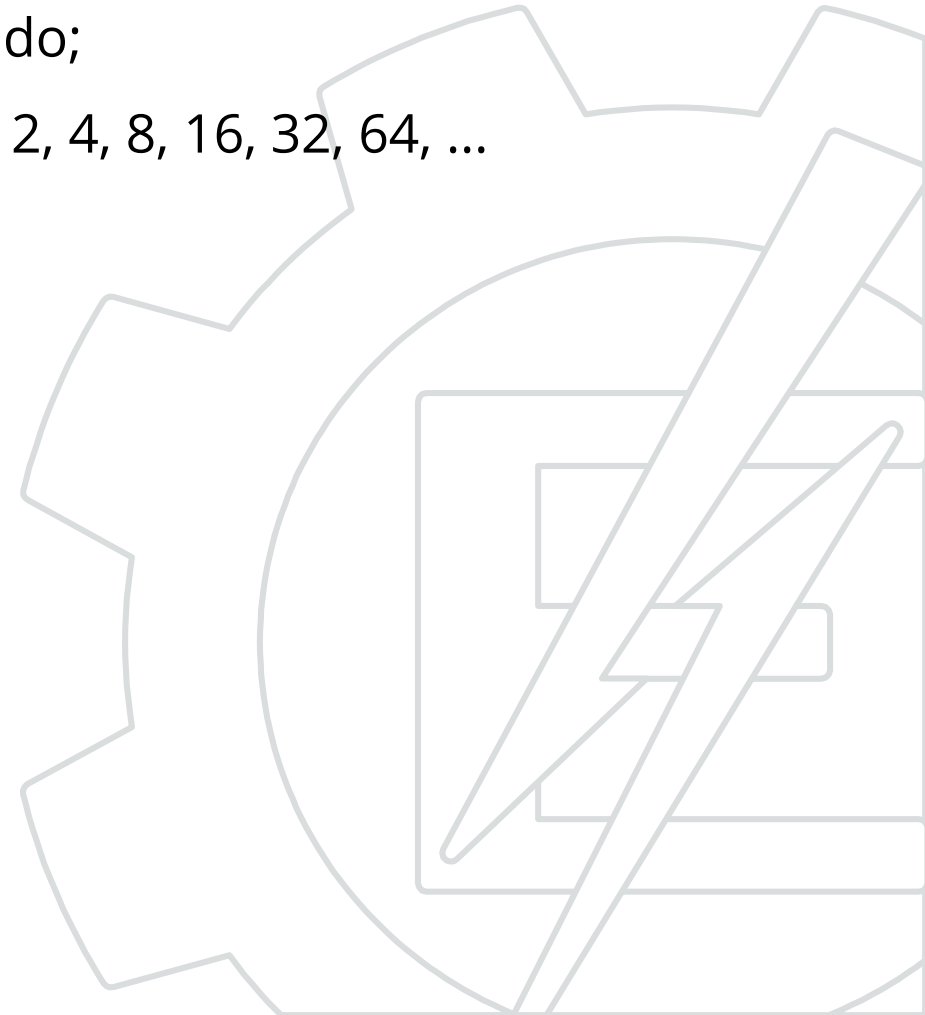


3. Múltiplas filas com realimentação



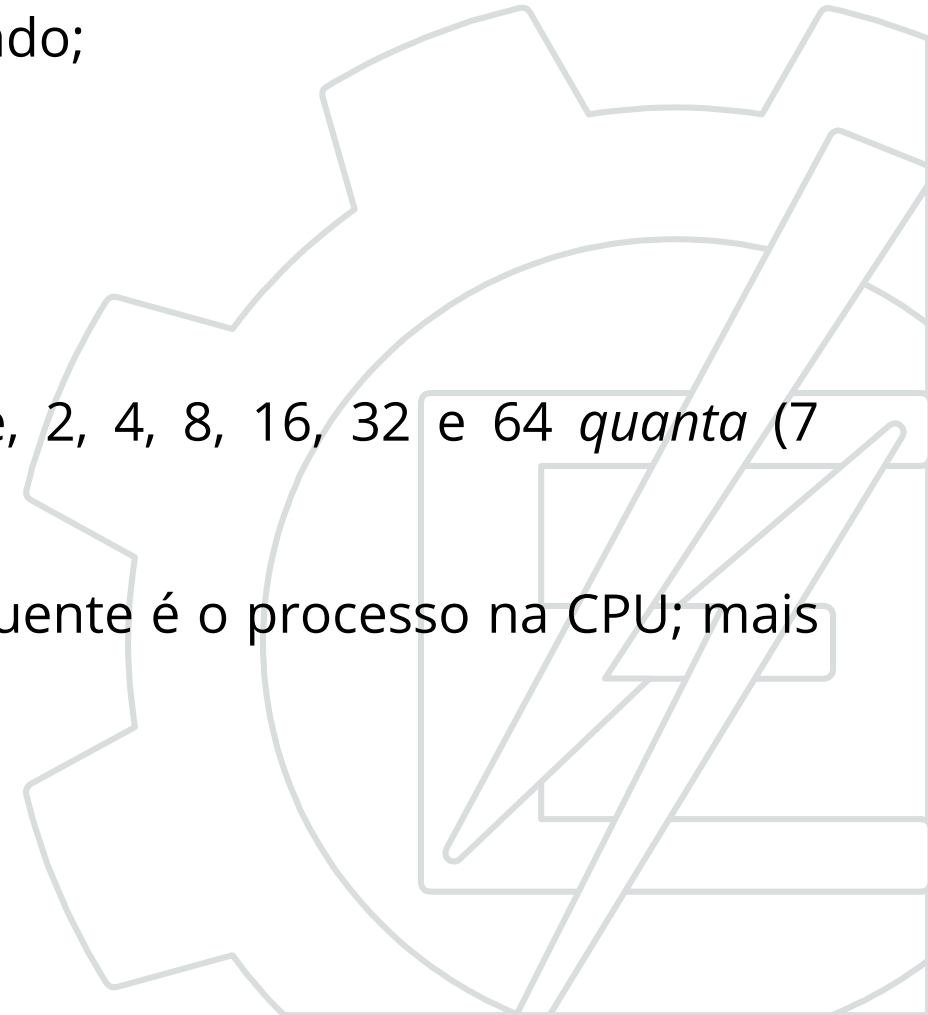
3. Múltiplas filas com realimentação

- Exemplo:
 - Um processo precisa de 100 *quanta* para ser executado;
 - As filas disponíveis possuem *quantum* nos valores: 1, 2, 4, 8, 16, 32, 64, ...

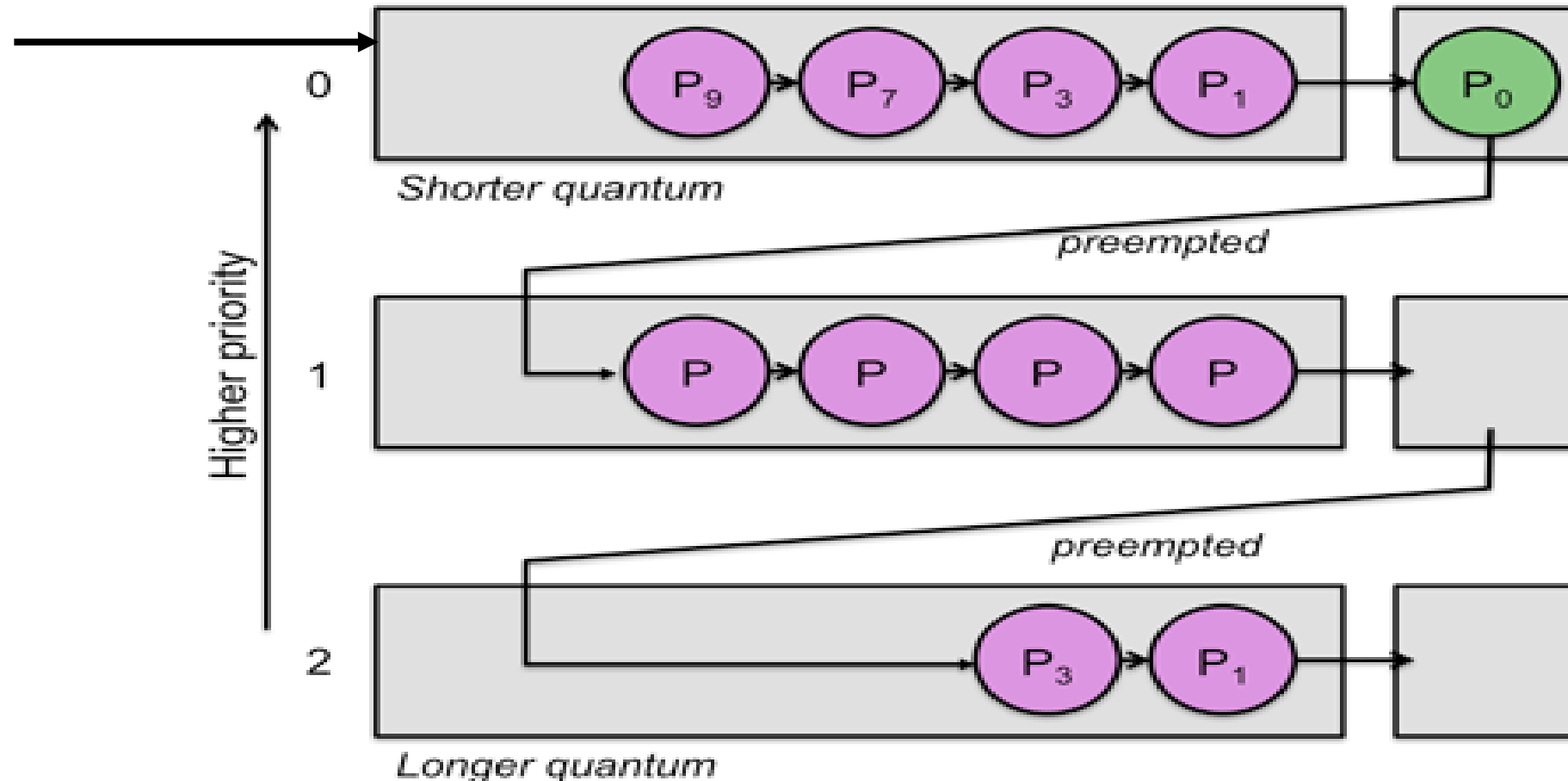


3. Múltiplas filas com realimentação

- Exemplo:
 - Um processo precisa de 100 *quanta* para ser executado;
 - Inicialmente, ele recebe um *quantum* para execução;
 - Das próximas vezes ele receberá, respectivamente, 2, 4, 8, 16, 32 e 64 *quanta* (7 chaveamentos) para execução;
 - Quanto mais próximo de ser finalizado, menos frequente é o processo na CPU; mais ele desce na fila de prioridade.

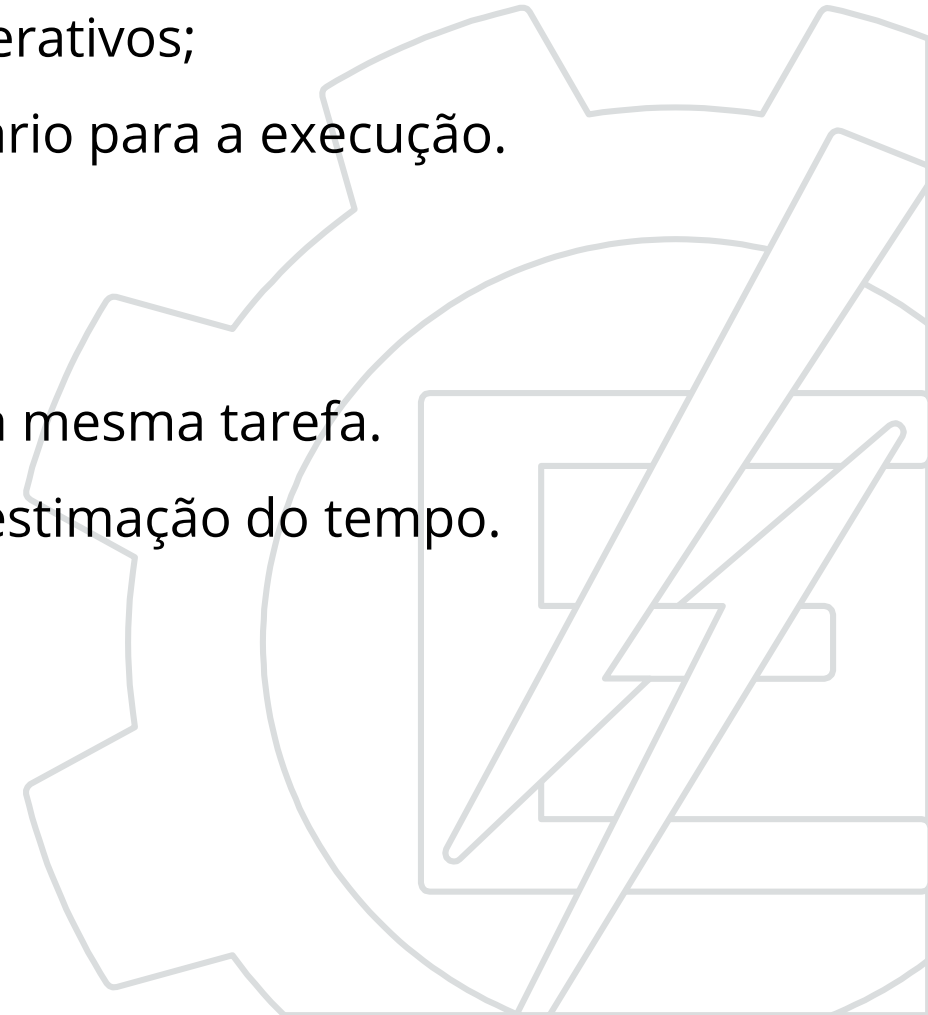


3. Múltiplas filas com realimentação



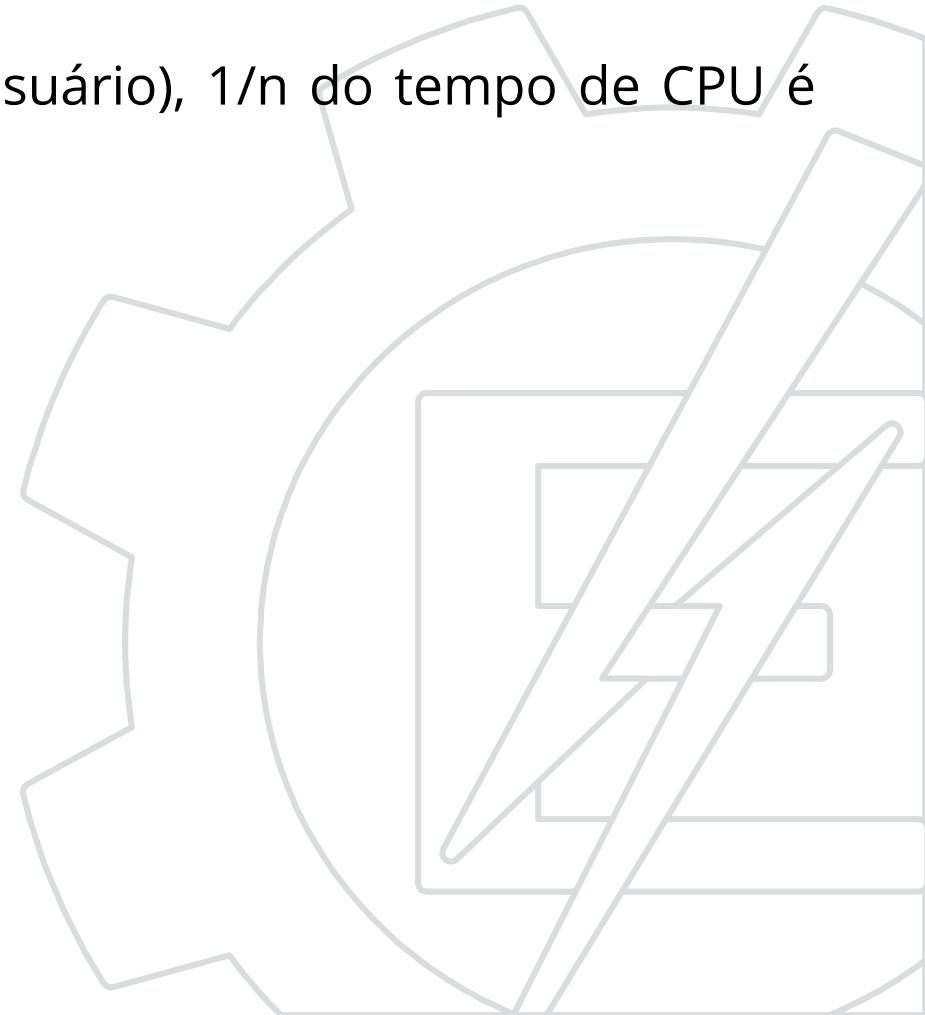
4. SPN (*Shortest Process Next*)

- É a versão do algoritmo *Shortest Job First* para sistemas interativos;
- Em processos interativos não se conhece o tempo necessário para a execução.
- Como empregar este algoritmo?
 - Estimativa de tempo com base em execuções antigas da mesma tarefa.
 - Verificação do comportamento passado do processo e estimação do tempo.



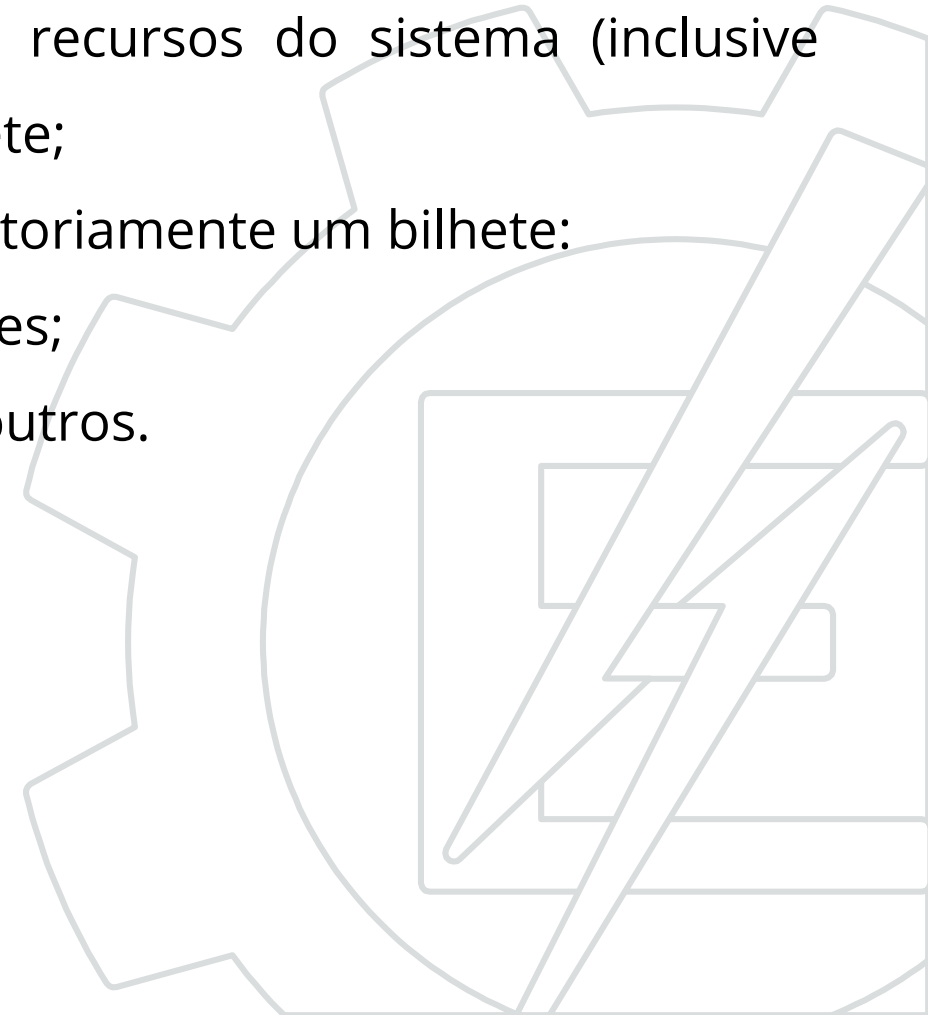
5. Garantido

- Garantias são dadas aos processos dos usuários:
 - Com n usuários (ou processos em sistemas monousuário), $1/n$ do tempo de CPU é reservado para cada usuário.



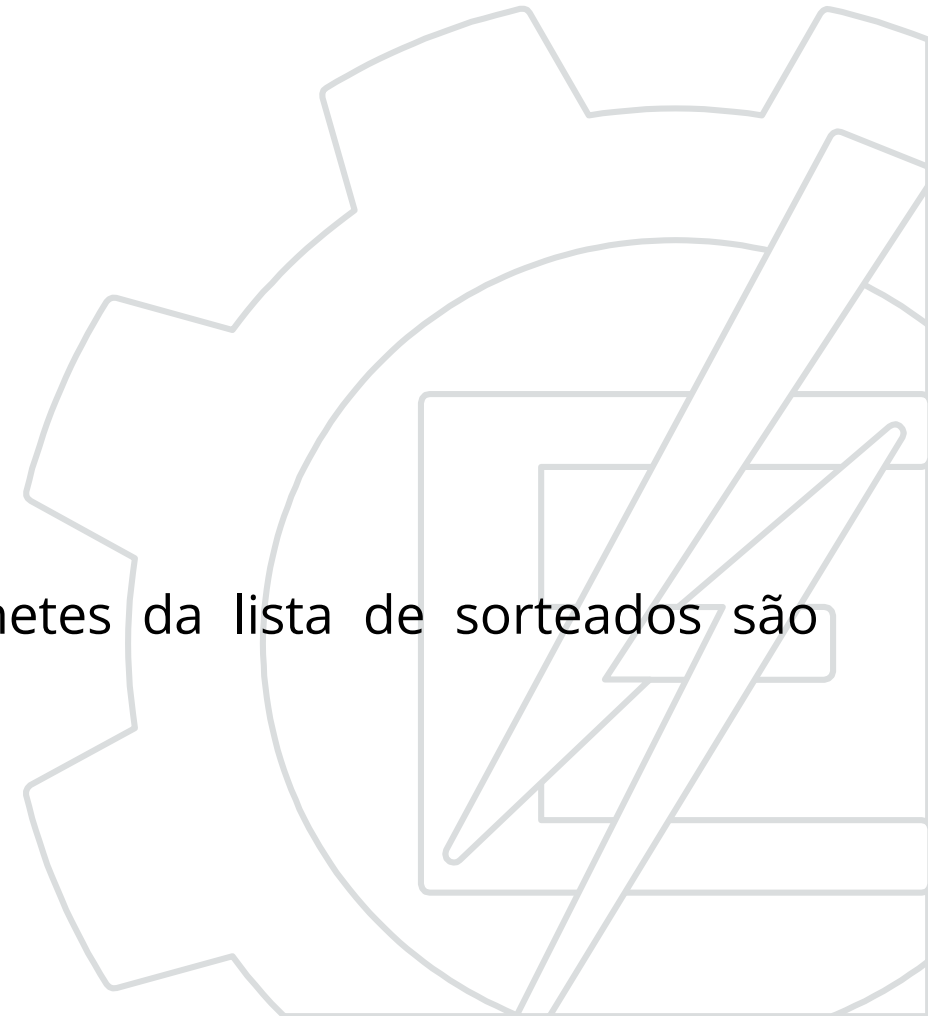
6. Loteria

- Cada processo recebe “bilhetes” que lhe dão direito a recursos do sistema (inclusive processador), com fatia de processamento iguais por bilhete;
- Quando um escalonamento deve ser feito, escolhe-se aleatoriamente um bilhete:
 - Processos mais importantes podem receber mais bilhetes;
 - Processos podem doar bilhetes para colaboração com outros.



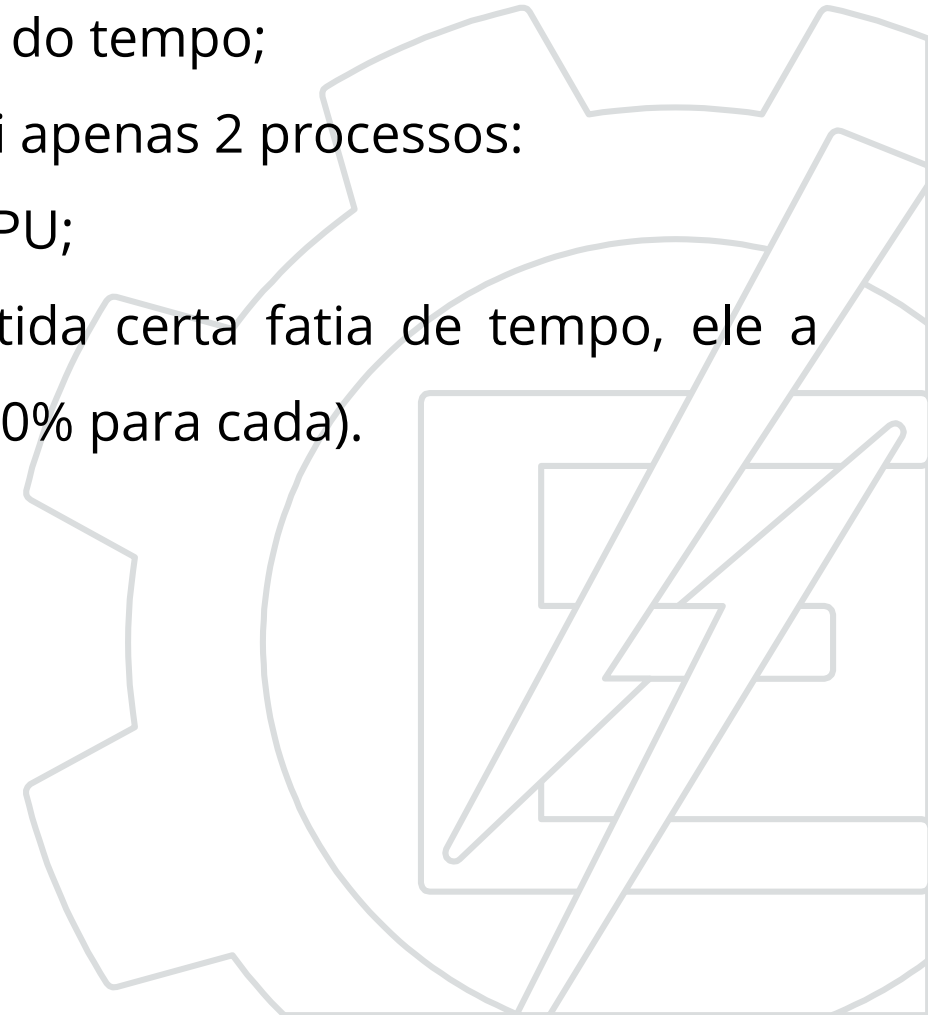
6. Loteria

- Precisa garantir que todos terão sua vez de execução;
- Um modo é manter duas filas:
 - Bilhetes já sorteados;
 - Bilhetes ainda não sorteados.
- Quando a lista de não sorteados se esvazia, os bilhetes da lista de sorteados são transferidos a ela, reiniciando o processo.



7. *Fair-Share*

- O dono do processo é levado em consideração na partilha do tempo;
- Se um usuário A possui 8 processos e um usuário B possui apenas 2 processos:
 - Com *round-Robin* o usuário A ganharia 80% do uso da CPU;
 - Com a distribuição justa, se a um usuário foi prometida certa fatia de tempo, ele a receberá, independente do número de processos (ex.: 50% para cada).



7. *Fair-Share*

- Usuário 1 – Processos: A, B, C, D
- Usuário 2 – Processo: E
- Foi prometido 50% da CPU a cada um e foi utilizado *Round-Robin*:
A, E, B, E, C, E, D, E, A, E, ...
- Se 2/3 devem ir ao Usuário 1:
A, B, E, C, D, E, A, B, E, ...



Bibliografia

- TANENBAUM, Andrew S; BOS, Herbert. Sistemas operacionais modernos. 4a ed. São Paulo: Pearson Education do Brasil, 2016.

Capítulo 2.

<https://plataforma.bvirtual.com.br/Acervo/Publicacao/1233>

- DEITEL, H.M; DEITEL, P.J; CHOFFNES,D.R. Sistemas Operacionais. 3a ed. São Paulo: Pearson Prentice Hall, 2005. **Capítulo 8.**

<https://plataforma.bvirtual.com.br/Acervo/Publicacao/315>



Sistemas Operacionais

Prof. Otávio Gomes

otavio.gomes@unifei.edu.br

