

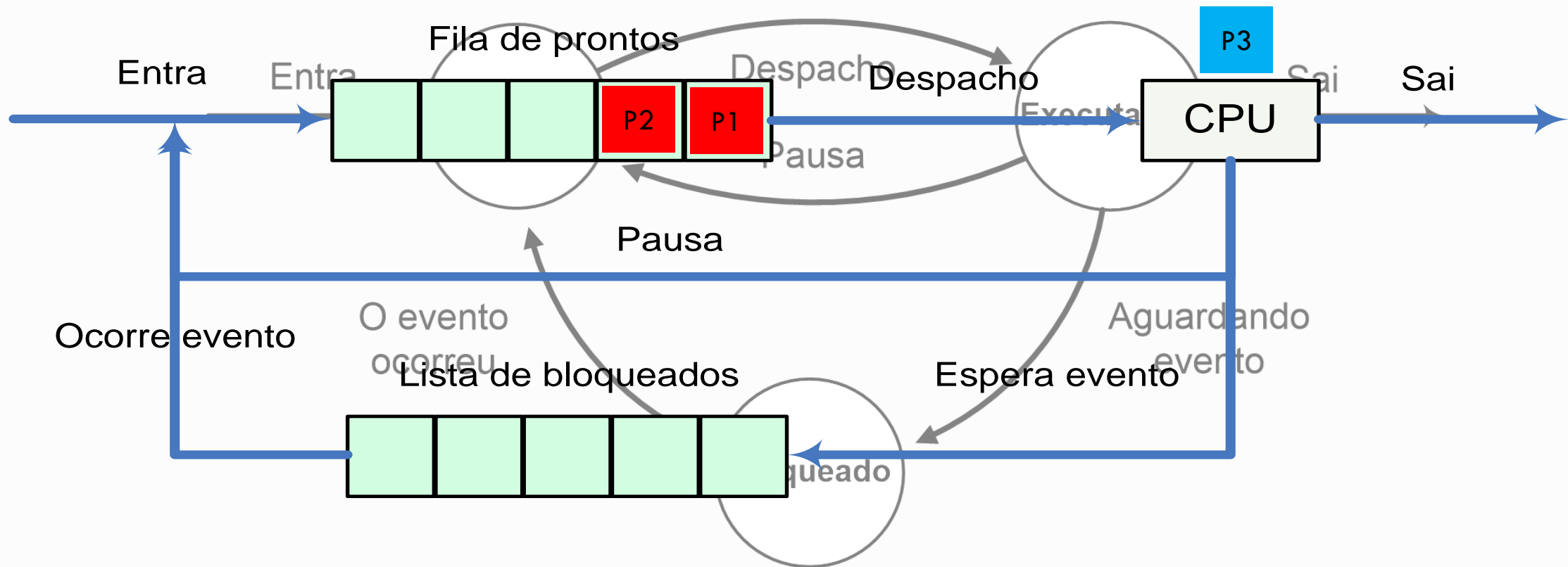
# Sistemas Operacionais

Escalonamento de Processos

---

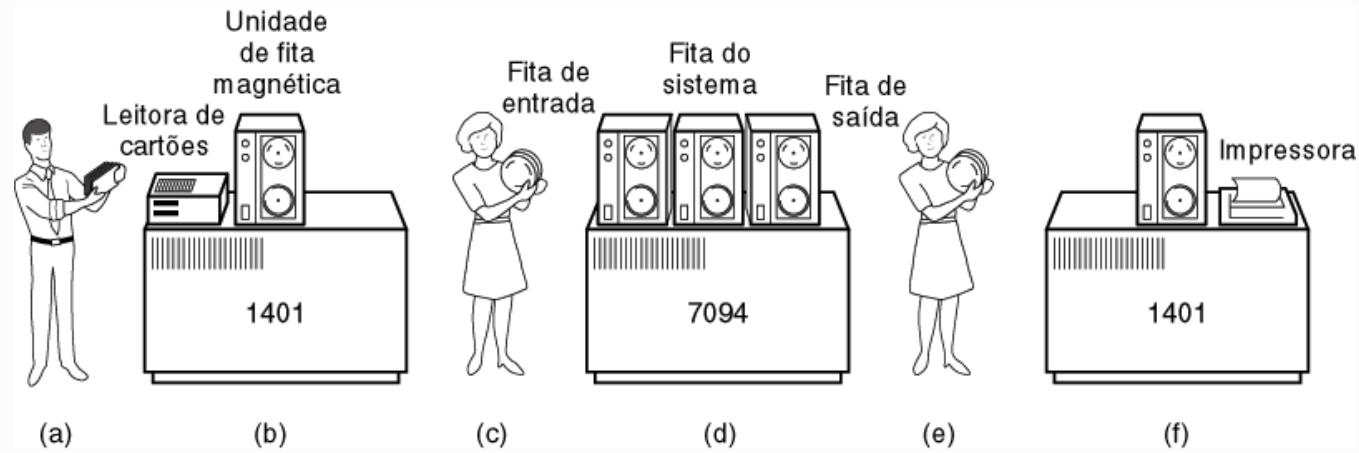
# Escalonamento

Situação que ocorre sempre que dois ou mais processos estão ao mesmo tempo no estado pronto.



# Escalonamento

Em sistemas de lote, algoritmos de escalonamento eram triviais.



A dificuldade surge quando diferentes usuários aguardam por um determinado serviço.


- Sistemas Interativos

Nas décadas passadas, o tempo de CPU era um recurso extremamente escasso.

Consequentemente, os algoritmos de escalonamento se tornaram fundamentais para uma performance razoável do sistema.

# Escalonamento

---



**O que isso quer dizer?**

Que os algoritmos de escalonamento de processo implementados para computadores “menos potentes” devem efetuar um balanceamento de carga bom, ou próximo ao ótimo.

Com os computadores modernos, os algoritmos podem não ser tão rigorosos assim, pois a percepção do usuário é inibida pelo *clock* das novas máquinas.

Mas historicamente, ambos evoluíram (balanceamento e *clock*).

# Escalonamento

---

Com o passar dos anos, os computadores ganharam poder de processamento, e a política de escalonamento perdeu um pouco sua grande importância atribuída no passado.

A maioria dos programas para computadores pessoais é limitada pela velocidade de entrada dos dados, e não pela CPU.

- Em computadores pessoais, não é muito comum termos vários processos requisitando a CPU durante um longo período de tempo.

Exemplo: Compiladores gastam alguns segundos para gerar arquivos executáveis.

- Depois deste período, eles entram no estado bloqueado, aguardando novas entradas do usuário.

# Escalonamento

---

Em computadores pessoais, também existe a característica da CPU ficar ociosa a maioria do tempo.

O que não deixa de também diminuir a importância dos algoritmos de escalonamento.

Quando falamos de servidores, o quadro muda completamente:

- Disponibilizam serviços muito requisitados ou;
- Disponibilizam uma grande quantidade de serviços.

# Escalonamento

---

Obrigações do escalonador:

- **Escolher o processo** que deve ser executado.
- Fazer um **uso eficiente** da CPU:
  - **Alterar entre processos é muito caro computacionalmente.**
    - Deve guardar os registradores do processo na tabela de processos.
    - O conteúdo da cache também deve ser armazenado.
    - O novo processo deve ser selecionado.
    - O novo processo precisa ser iniciado (envolvendo carregar todas as informações da tabela de processos na memória, etc.).
  - Troca de contexto!

# Quando Escalonar?

---



Existem algumas situações triviais;

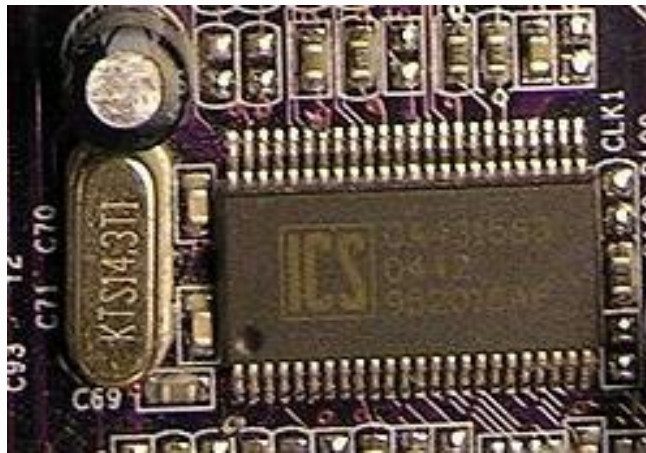
- Quando um **processo termina**.
- Quando um **processo é bloqueado** ao se deparar com um semáforo, por exemplo.
  - Neste caso, outro processo específico deve ser escalonado, mas o escalonador de processos não tem conhecimento sobre isso.
- Portanto, o que acontece é: o processo “acordado” vai para a fila de processos prontos.



# Quando Escalonar?

---

Quando existe um relógio no computador, o escalonador pode aproveitar as interrupções de *clock* para criar políticas de escalonamento.



A decisão de escalonamento pode ser tomada a cada interrupção de *clock*, ou depois da *k*-ésima interrupção de *clock*.

# Quando Escalonar?

Existem duas categorias de escalonamento:

## Não preemptivo

O processo é executado até terminar.

Ou, o processo é executado até se bloquear.

Resumindo: nenhuma decisão de escalonamento é tomada em função das interrupções de relógio.

## Preemptivo

O escalonador escolhe um processo, e este é executado até o momento que o escalonador queira.

Ou o processo deixa a CPU quando se bloqueia.

Ou o processo deixa a CPU quando termina.

**Se não existir relógio disponível, o escalonamento não preemptivo será a única opção simples!**  
O projetista pode utilizar de outra interrupção para isso, mas nenhuma delas está inserida em um contexto temporal.

# Algoritmos de Escalonamento

---

Categorias de algoritmos de escalonamento:

## **Lote:**

- Não existem usuários aguardando impacientes por respostas. Com isso, podem optar por políticas não preemptivas e melhorar o desempenho do sistema.

## **Interativo:**

- Já nos ambiente interativos, a preempção é fundamental.

# Objetivos gerais de cada tipo

Serão analisados os objetivos de sistema:

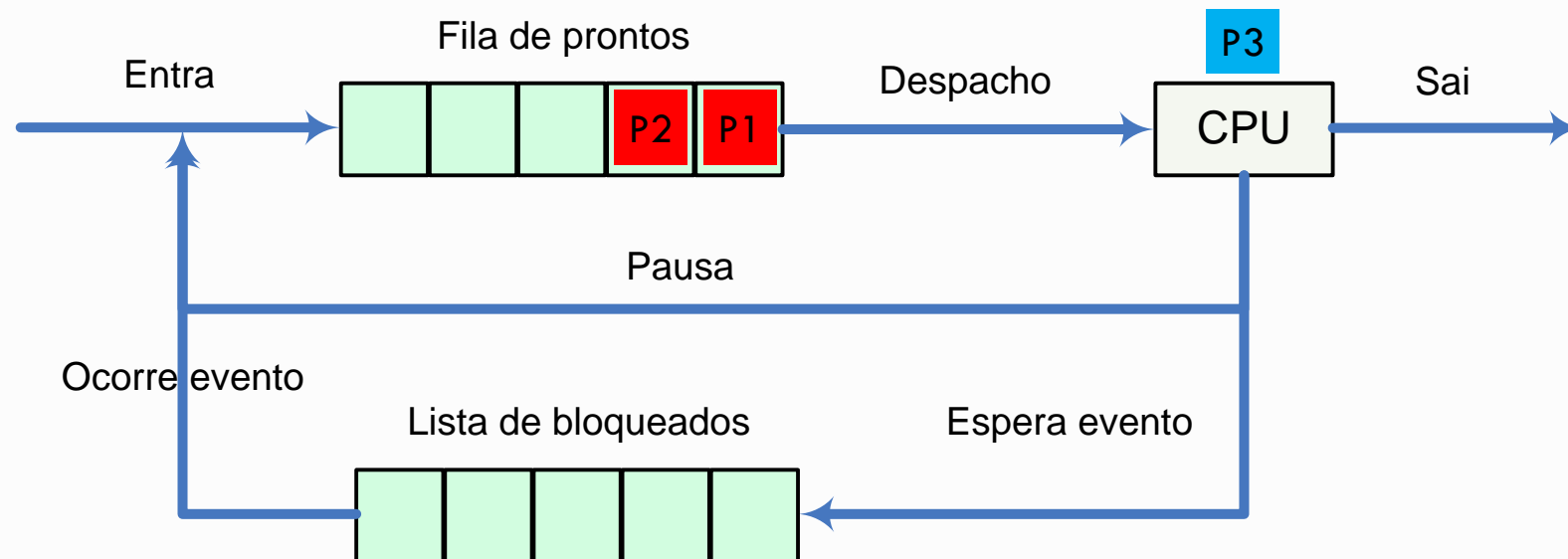
- (i) Todos os sistemas,
- (ii) Sistemas em Lote e
- (iii) Sistemas Interativos.



# Escalonamento – Sistemas em Lote

## Primeiro a chegar, primeiro a ser servido – *First come, First served* (FCFS)

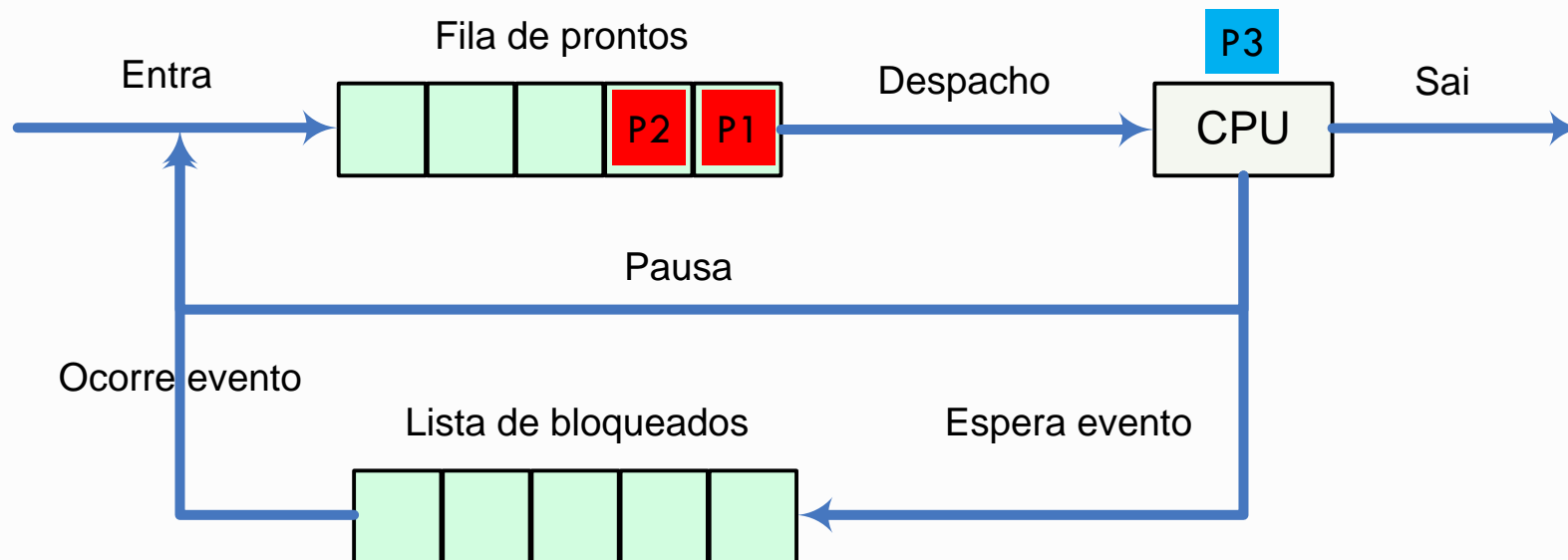
- Fácil entendimento e implementação.
- É uma política não preemptiva.
- **Quais são as desvantagens?**



# Escalonamento – Sistemas em Lote

## Primeiro a chegar, primeiro a ser servido – *First come, First served* (FCFS)

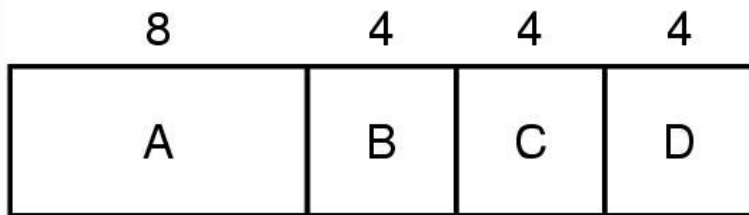
- **Quais são as desvantagens?**
  - Não pode ser usado para sistemas interativos;
  - Se o primeiro *job* for demorado demais, os outros, mesmo sendo curtos, irão esperar muito tempo!



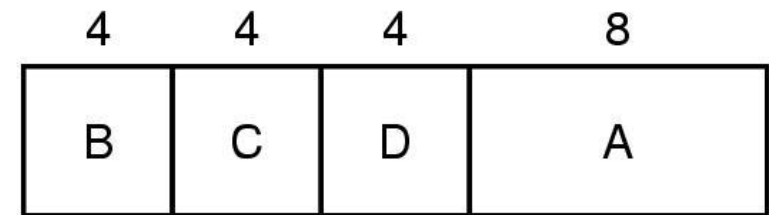
# Escalonamento – Sistemas em Lote

## **Job mais curto primeiro**

- Necessita conhecer todos os tempos totais de processamento (ao menos uma estimativa).
- É uma política **não preemptiva**.
  - a) Execução na ordem original
  - b) Execução na ordem do job mais curto



(a)



(b)

# Escalonamento – Sistemas em Lote

---

## Próximo de menor tempo restante

- Bem parecido com a política do *Job* mais curto primeiro.
- A diferença fundamental é que este método é **Preemptivo**.
- Quando um novo *job* entra no estado pronto, o escalonador pode interromper a execução do atual, e fornecer a CPU ao novo *job*, bloqueando o *job* em execução.

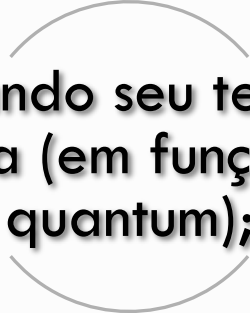


# Escalonamento – Sistemas Interativos

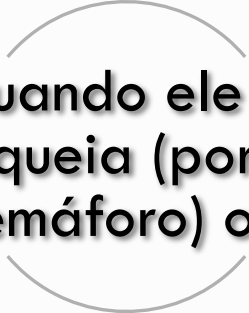
---

## **Round-Robin** (Escalonamento por alternância circular)

- Um dos mais antigos algoritmos, considerado justo e simples.
- Amplamente utilizado.
- Conceito de quantum: Intervalo de tempo que cada processo pode se ocupar da CPU sem interrupções.
- Existem 3 formas de um processo deixar a CPU no Round-Robin:



Quando seu tempo acaba (em função do quantum);



Quando ele se bloqueia (por um semáforo) ou;



Quando ele termina.

# Escalonamento – Sistemas Interativos

## Round-Robin (Escalonamento por alternância circular)

Implementação necessita de uma lista encadeada, ou outra estrutura equivalente.

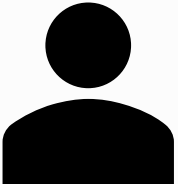


- a) Fila de processos prontos. B está na CPU;
- b) Fila de processos prontos, depois que B deixa a CPU (fim de seu quantum).

# Escalonamento – Sistemas Interativos

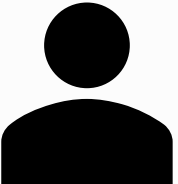
---

## Round-Robin (Escalonamento por alternância circular)



Quais são as implicações de um quantum curto?

Quais são as implicações de um quantum muito longo?



Tanenbaum sugere um **quantum entre 20ms e 50ms** para sistemas interativos; Este valor, obviamente, muda com o avanço dos processadores e com a evolução dos Sistemas Operacionais. Muda também com a redução da complexidade dos algoritmos de escalonamento.

# Escalonamento – Sistemas Interativos

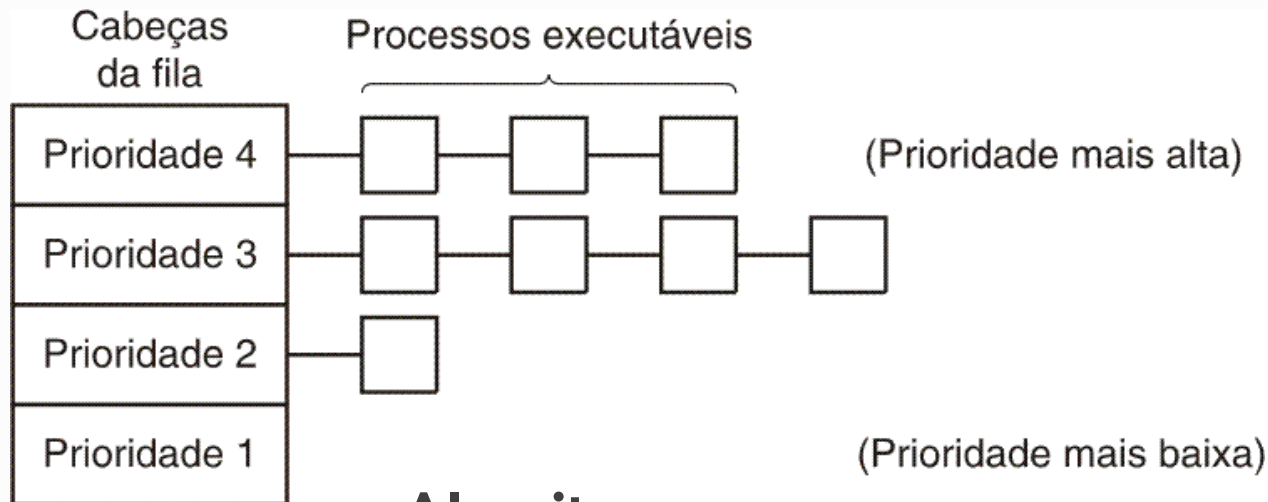
---

## Escalonamento por Prioridades

- Diferentes processos geralmente possuem diferentes necessidades.
- Geralmente, em sistemas interativos, uma atenção maior é dada para processos que se comunicam com os usuários.
- Sistemas operacionais comerciais implementam um híbrido entre a abordagem Round-Robin, e a abordagem por prioridades.
- Ou seja, a Fila utilizada não é do tipo “FIFO”.
- Fila de prioridades.
- É necessária uma atenção maior para não deixar ocorrer o starvation nos processos/threads de baixa prioridade.

# Escalonamento – Sistemas Interativos

## Escalonamento por Prioridades – Exemplo de estrutura



Qual é o problema deste algoritmo para um sistema interativo?

### Algoritmo:

Executa todos de prioridade 4.  
Depois executa todos de prioridade 3.  
Depois executa todos de prioridade 2.  
Depois executa todos de prioridade 1.

Cada categoria pode ser tratada com o algoritmo de alternância circular (*Round Robin*)

# Escalonamento – Sistemas Interativos

---

## Escalonamento por Prioridades – Exemplo de estrutura

Com o problema *starvation*, na abordagem apresentada, o sistema operacional deve regularmente efetuar uma manutenção na prioridade dos processos ativos, com o objetivo de garantir que nenhum deles morra de fome.

### Exemplo de manutenção:

Se um processo aguardar mais de 3 segundos, sua prioridade é incrementada em uma unidade.

Quando ele voltar para a fila de prontos, volta com sua prioridade original.

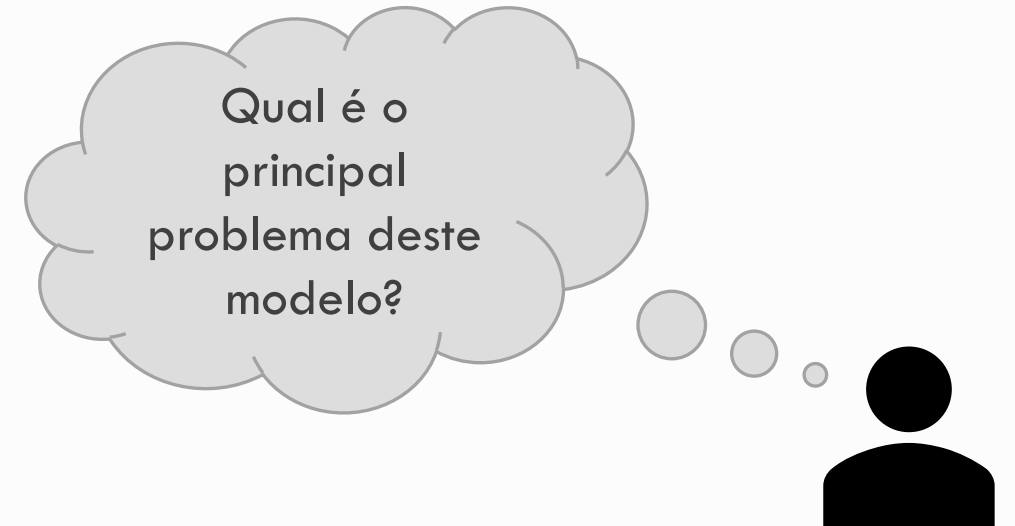
# Escalonamento – Sistemas Interativos

---

## Escalonamento Garantido

O sistema operacional pode garantir que se existem 10 usuários conectados, cada um vai receber 10% do tempo de CPU.

De forma equivalente, o S.O. pode fazer esta divisão considerando a quantidade de processos ativos, e não a quantidade de usuários.

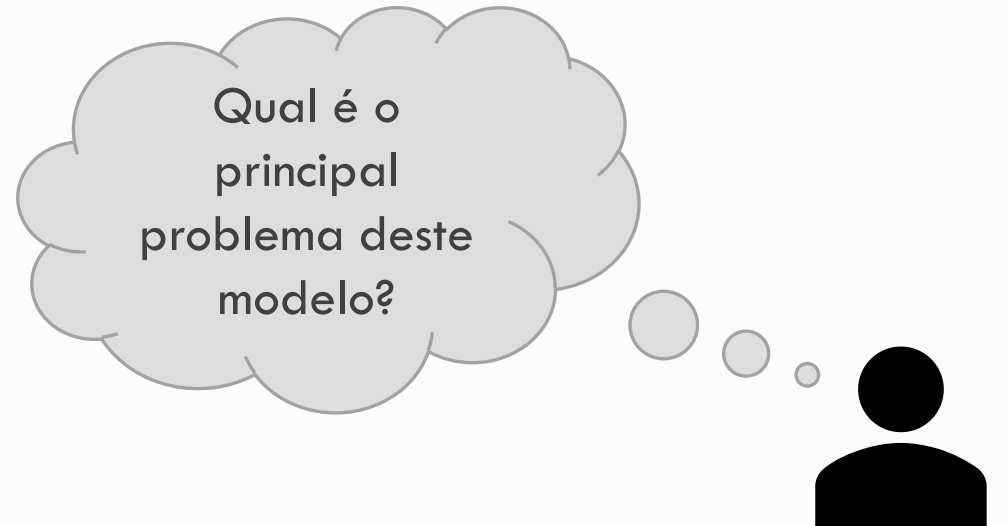
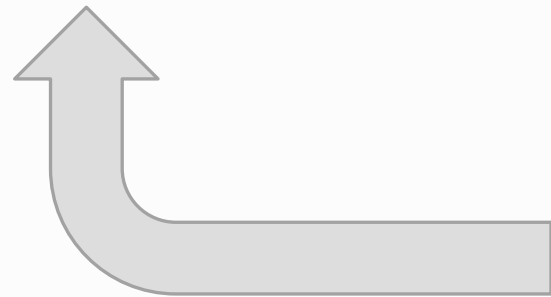


# Escalonamento – Sistemas Interativos

---

## Escalonamento Garantido

Algoritmo de difícil implementação, se a GARANTIA for levada 100% a sério.  
Possíveis métodos de adaptação on-line devem agir para corrigir os contrapesos ao longo da execução do algoritmo.





# Escalonamento – Sistemas Interativos

---

## **Escalonamento por loteria**

Diferente do escalonamento garantido, o escalonamento por loteria possui implementação trivial.

O S.O. fornece “bilhetes” a cada processo.

Ele sorteia um bilhete, e o processo que tem aquele bilhete ganha como prêmio, acesso a CPU.



# Escalonamento – Sistemas Interativos

---

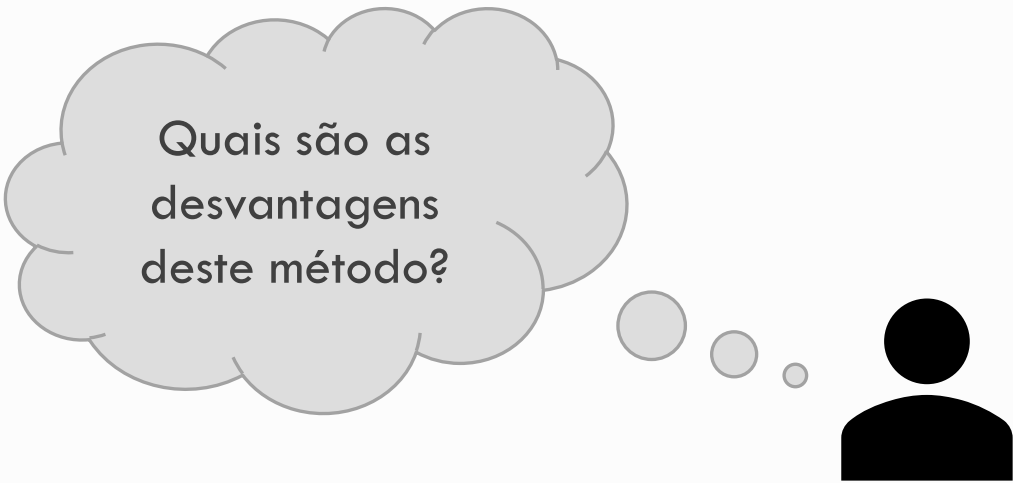
## Escalonamento por loteria

Aos processos mais importantes podem ser atribuídos bilhetes extras para aumentar sua chance de vitória.

Aumenta naturalmente sua prioridade!

A probabilidade do processo ter a CPU a cada escalonamento é:

$$\frac{\text{Quantidade de bilhetes do processo}}{\text{Quantidade total de bilhetes}}$$



Quais são as  
desvantagens  
deste método?

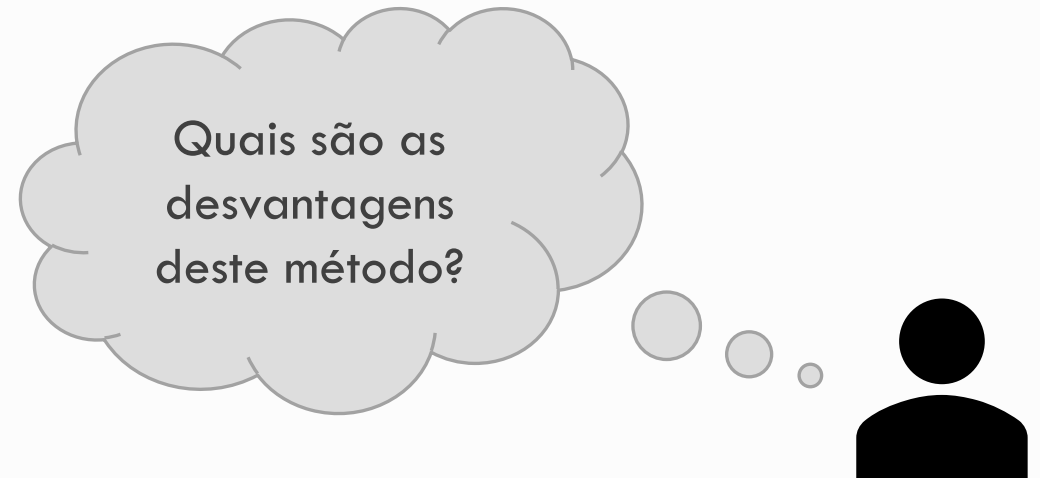
# Escalonamento – Sistemas Interativos

---

## Escalonamento por loteria

O S.O. perde o controle sobre os processos que eventualmente morrem de fome (embora com uma probabilidade mínima de ocorrer, um processo pode ficar muito tempo sem receber recursos).

Uma desvantagem a ser considerada é a probabilidade depender diretamente de “números aleatórios”.



# Política vs Mecanismo

---

Um processo, ao criar processos filhos, pode ter noção da importância do processo que está sendo criado.

## Exemplo: SGBD

Processo para realizar uma análise sintática de uma consulta efetuada no terminal do SGBD.

Processo para realizar uma análise sintática de uma consulta efetuada em uma aplicação de usuário.

Processo para acesso a disco.

Processo para disparar a resposta de uma consulta na rede (Socket).

O pai pode “controlar” como seus filhos devem ser escalonados.

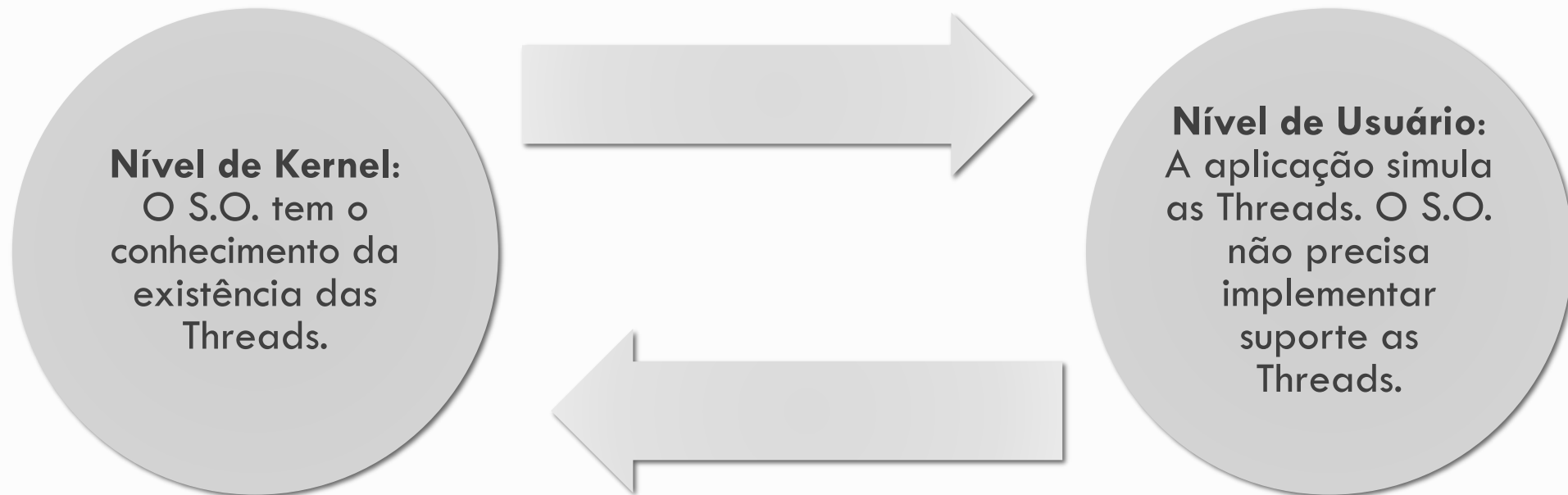
Desta forma, dizemos que o mecanismo de escalonamento está no núcleo.

A política é estabelecida pelo processo (prioridade passada por parâmetro).

# Escalonamento de Threads

---

Vamos relembrar o conceito de Threads implementadas no nível do Kernel, e threads implementadas no nível do usuário:



# Escalonamento de Threads

---

## Nível de Usuário:

- O S.O. não sabe da existência de Threads.
- Desta forma, se existir um processo A com 30 threads simuladas, e o processo B com apenas 1 linha de execução, o sistema operacional não tem como priorizar a execução do processo A, a não ser por uma política que oferece uma maior probabilidade de execução.
- Exemplos:
  - Mais bilhetes (escalonamento por loteria).
  - Maior prioridade (escalonamento por prioridades).

# Escalonamento de Threads

---

## Nível de Usuário:

- A implementação do escalonamento de threads a nível do usuário possui uma vantagem que pode representar uma diferença significativa.
- Imagine o problema dos Produtores e Consumidores:
  - Se o processo consumidor é bloqueado, quer dizer que um produtor deve produzir um item.
  - No escalonamento de threads a nível de kernel, o processo consumidor dorme, aguardando a sorte de um processo produtor ser acordado pelo S.O.
  - Já no escalonamento a nível de usuário, o desenvolvedor pode controlar a chamada imediata de um produtor que estiver dormindo, acabando com a dependência da sorte.

# Próxima Aula

---

Leitura:

Sistemas Operacionais Modernos

**Jantar dos filósofos**



# Referências

---

Sistemas Operacionais Modernos. Tanenbaum, A. S. 2ª edição. 2003.

Sistemas Operacionais. Conceitos e Aplicações. A. Silberschatz; P. Galvin; G. Gagne. 2000.

Sistemas Operacionais – Projeto e Implementação. Tanenbaum, A. S. 2ª edição. 2000.

Slides Prof. Humberto Brandão