



---

# **Sistemas Operacionais**


## **Gerência de Memória**

**Fragmentação e  
Compartilhamento de memória**

# Objetivos

---

- Uso racional da memória principal
  - Apresentação do problema – fragmentação
    - Proposta de soluções
  - Técnica de compartilhamento de memória



# **Sistemas Operacionais**

## **Parte I Fragmentação**

A decorative graphic on the left side of the slide, consisting of a vertical light blue bar, a vertical orange bar, and a grey rectangle. A horizontal dark blue line extends from the grey rectangle across the top of the slide.

# **1. Fragmentação Externa**

# Fragmentação

---

## Ao longo da vida de um sistema:

- Áreas de memória liberadas por processos;
- Outras áreas são alocadas por novos processos;
- Podem surgir áreas livres **entre** os processos;

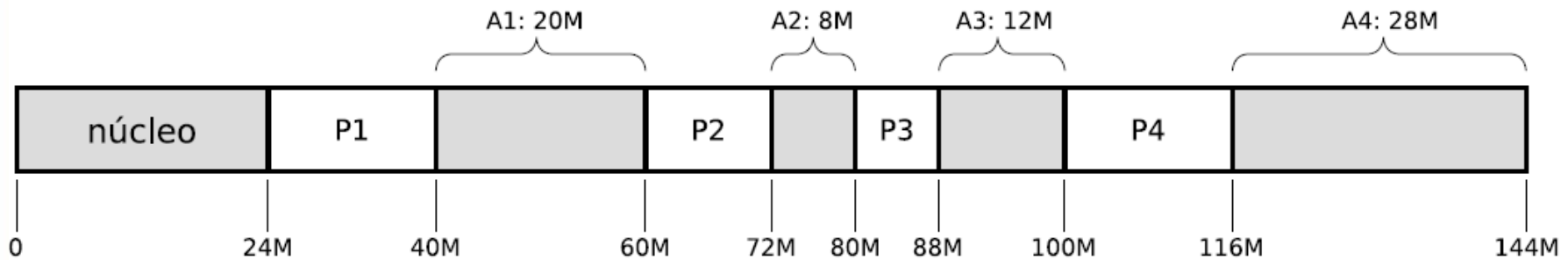
# Fragmentação

---

## Ao longo da vida de um sistema:

- Áreas de memória liberadas por processos;
- Outras áreas são alocadas por novos processos;
- Podem surgir áreas livres **entre** os processos;
  - *Fragmentação Externa*

# Fragmentação Externa



# Fragmentação Externa

---

**Somente afeta alocação com blocos de tamanho variável:**

- Alocação contígua
- Alocação segmentada



# Fragmentação Externa

---

**Somente afeta alocação com blocos de tamanho variável:**

- Alocação contígua
- Alocação segmentada

**Alocação paginada** → blocos de tamanho fixo

⇒ imune à *fragmentação externa*

# Fragmentação Externa

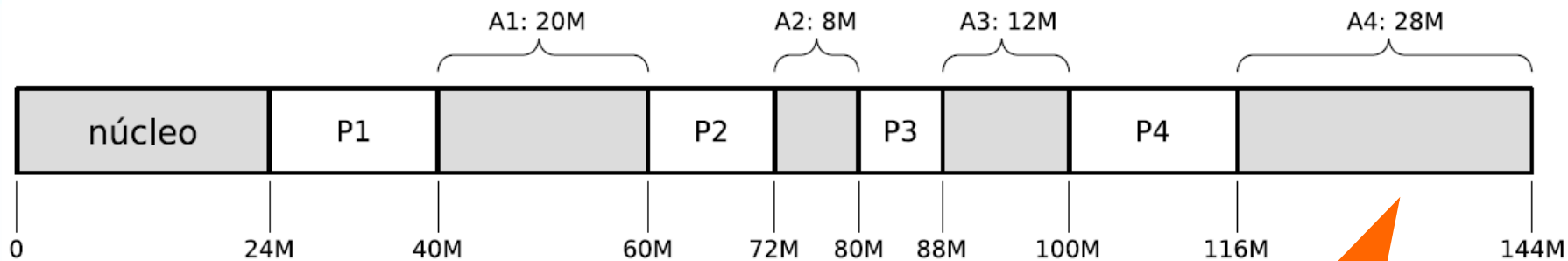


---

É **prejudicial** porque limita a capacidade de alocação de memória no sistema.

# Fragmentação Externa

Exemplo: [Alocação contígua](#)



- 68 MB livres
- Somente processos com até 28 MB podem ser alocados

# Fragmentação Externa

## Cálculo

$$F = 1 - \frac{N_{free}}{N_{total}}$$

$N_{total}$  – # de bytes livres no total

$N_{free}$  – # de bytes do maior bloco livre



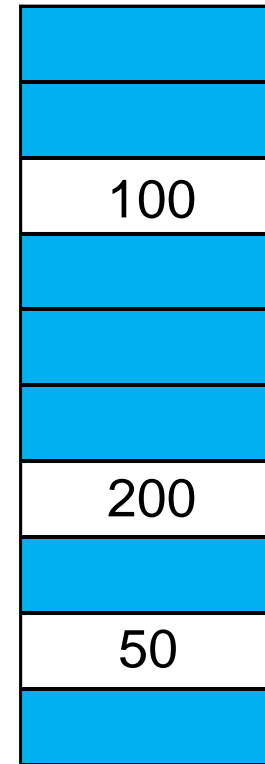
# Fragmentação Externa

## Cálculo

$$F = 1 - \frac{N_{free}}{N_{total}} = 1 - \frac{200}{200 + 100 + 50} = 0.428$$

$N_{total}$  – # de bytes livres no total

$N_{free}$  – # de bytes do maior bloco livre



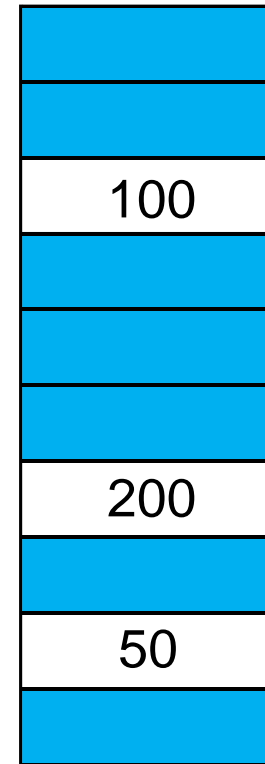
# Fragmentação Externa

## Cálculo

$$F = 1 - \frac{N_{free}}{N_{total}} = 1 - \frac{200}{200 + 100 + 50} = 0.428$$

$N_{total}$  – # de bytes livres no total

$N_{free}$  – # de bytes do maior bloco livre



Nem sempre é a forma mais adequada!

# Fragmentação Externa

---

## Solução do problema:

- a. Minimizando sua ocorrência
- b. Desfragmentando a memória

# a - Minimizar ocorrência

---

- Para **minimizar a ocorrência** de fragmentação externa:
- Cada pedido de alocação deve ser **analisado** para encontrar a área de memória livre que **melhor** o atenda.



# Critérios de alocação

---

## i. Melhor encaixe (*best-fit*) :

Escolher a **menor** área possível que possa atender à solicitação de alocação.

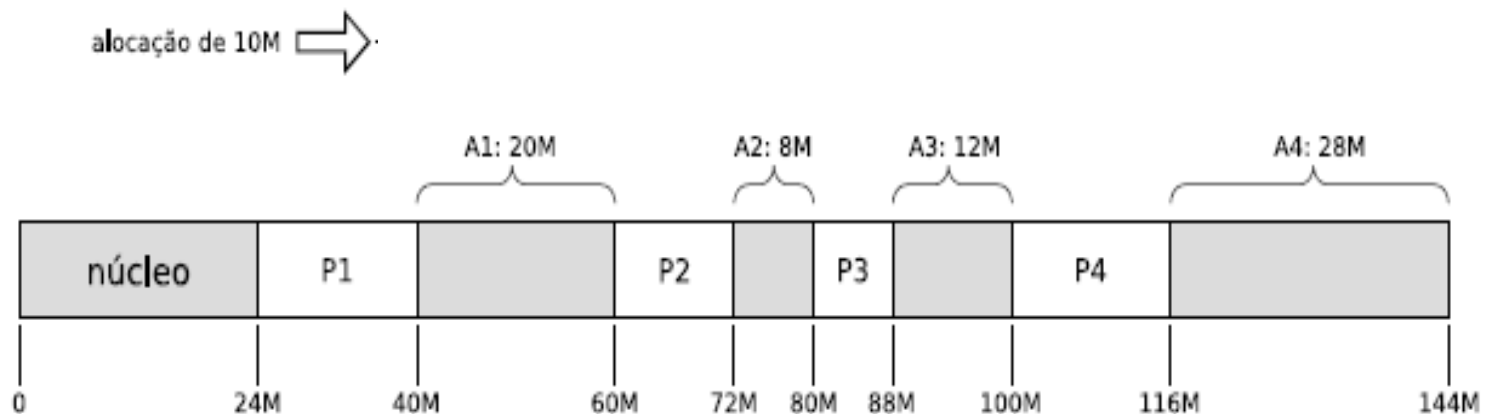
Áreas livres são usadas de forma otimizada

**Desvantagem:** Eventuais resíduos podem ser pequenos demais para ter alguma utilidade.

# Critérios de alocação

## i. Melhor encaixe (*best-fit*) :

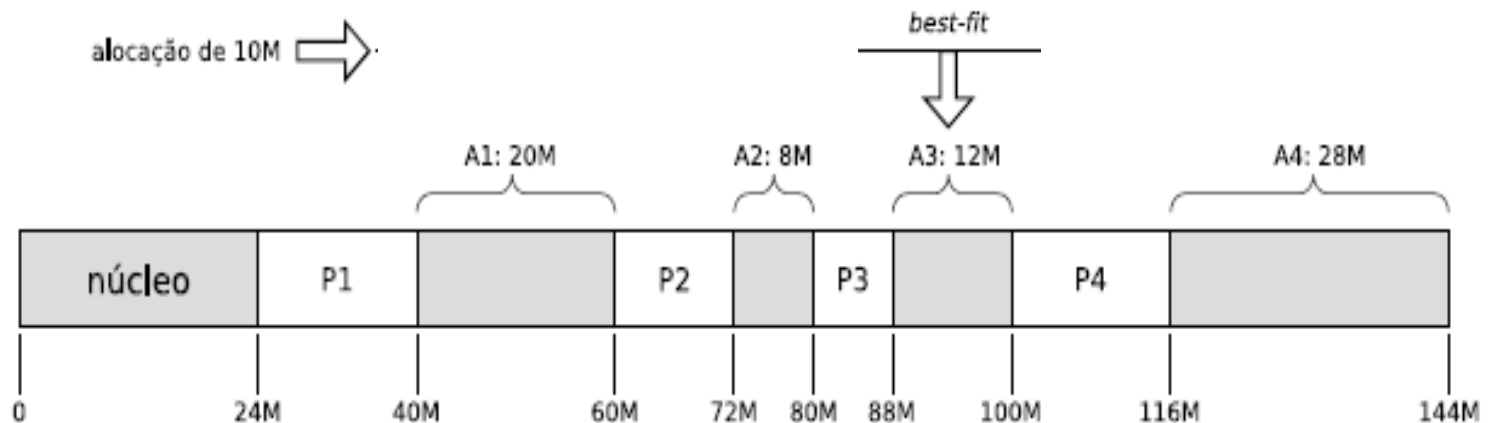
Escolher a **menor** área possível que possa atender à solicitação de alocação.



# Critérios de alocação

## i. Melhor encaixe (*best-fit*) :

Escolher a **menor** área possível que possa atender à solicitação de alocação.

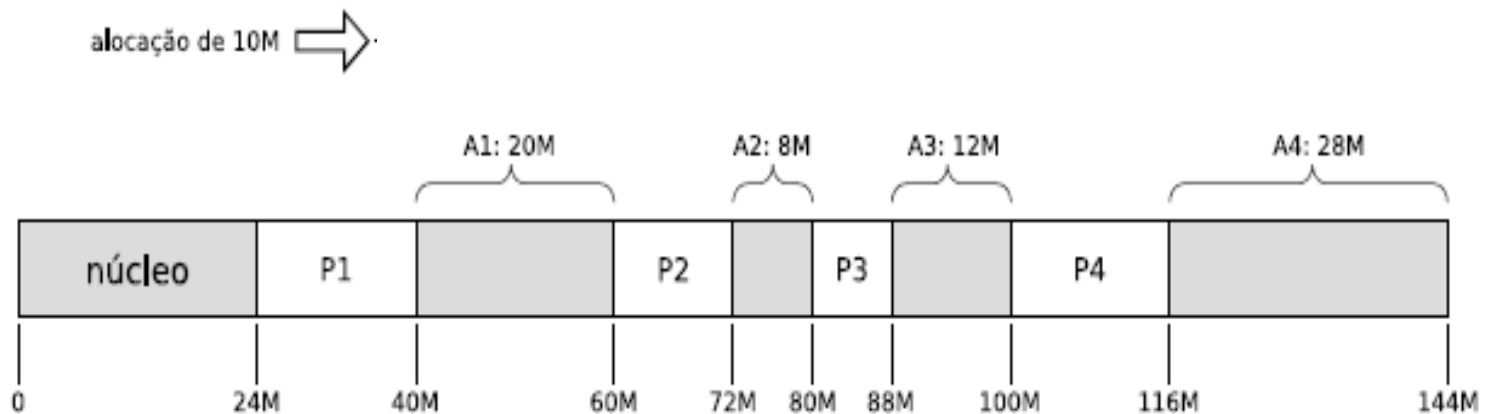


# Critérios de alocação

## ii. Pior encaixe (*worst-fit*) :

Escolher sempre a **maior** área livre possível

⇒ Resíduos grandes podem ser usados em outras alocações.

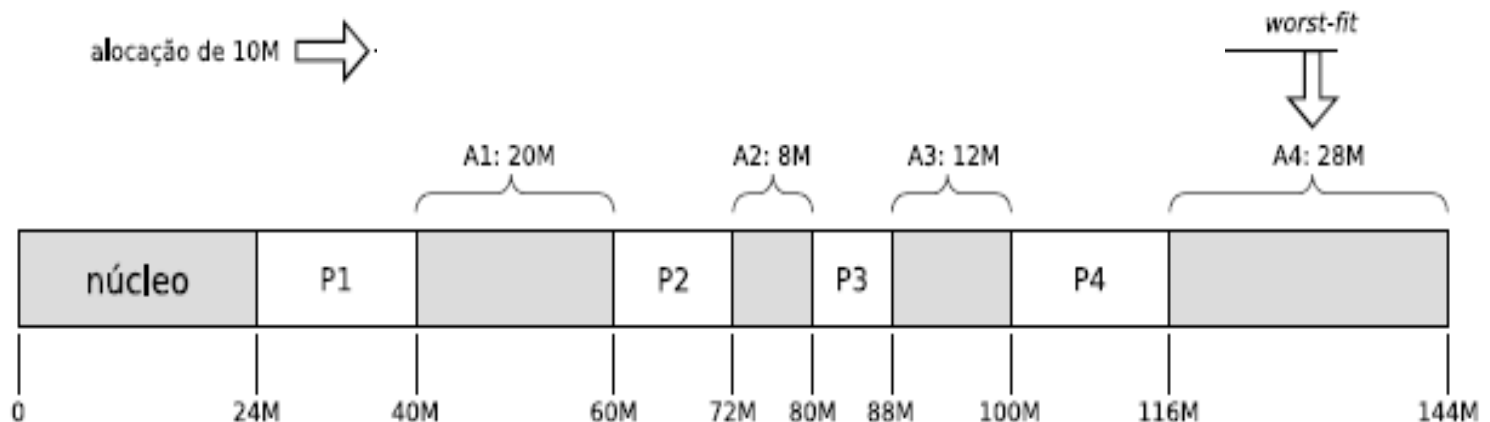


# Critérios de alocação

## ii. Pior encaixe (*worst-fit*) :

Escolher sempre a **maior** área livre possível

⇒ Resíduos grandes podem ser usados em outras alocações.

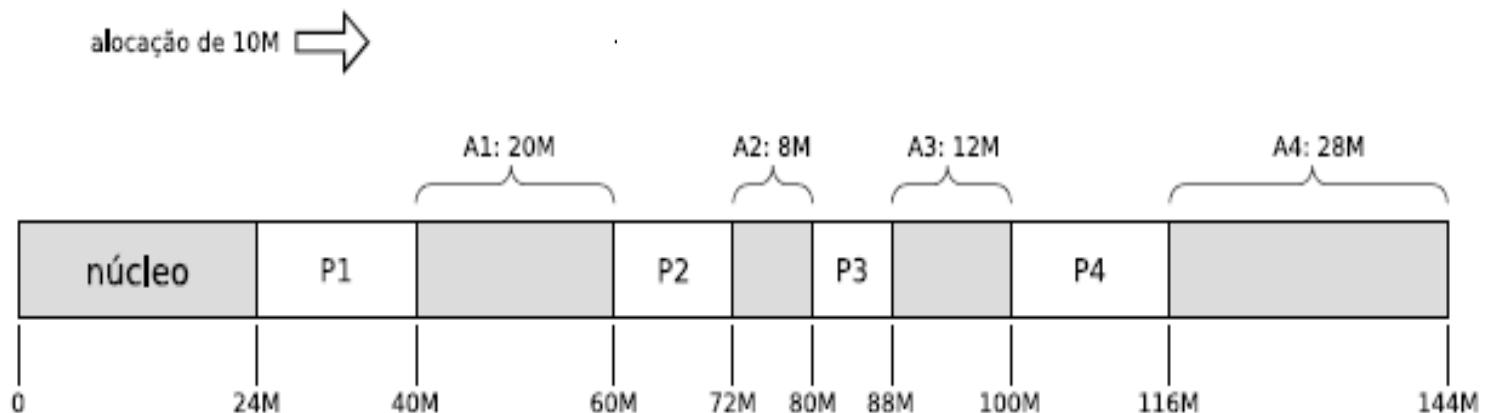


# Critérios de alocação

## iii. Primeiro encaixe (*first-fit*) :

Escolher a **primeira** área livre que satisfaça o pedido de alocação

**Vantagem:** rapidez, sobretudo se a lista de áreas livres for muito longa.

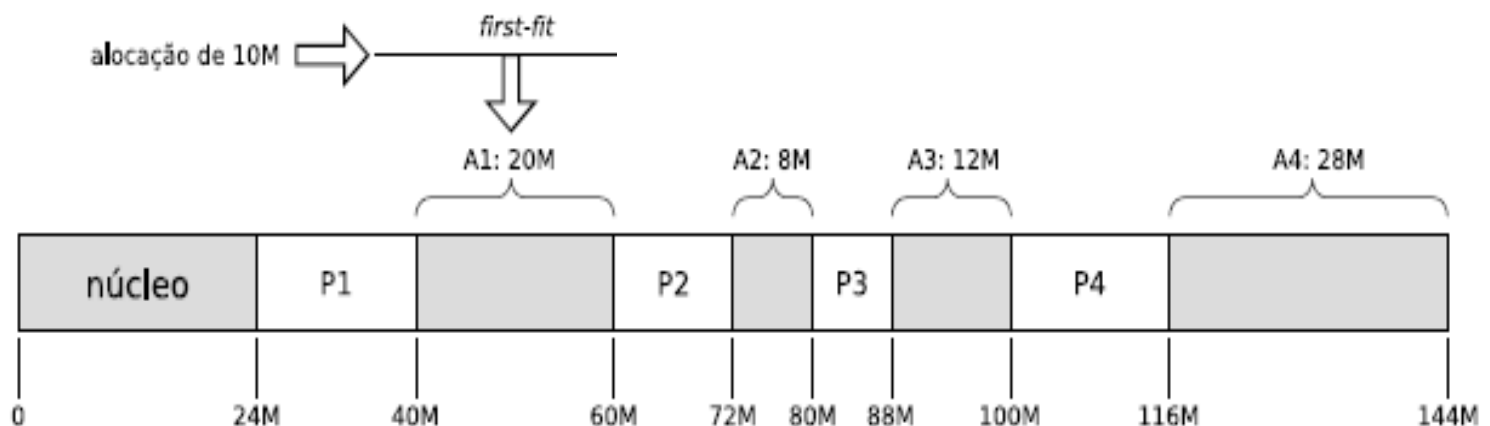


# Critérios de alocação

## iii. Primeiro encaixe (*first-fit*) :

Escolher a **primeira** área livre que satisfaça o pedido de alocação

**Vantagem:** rapidez, sobretudo se a lista de áreas livres for muito longa.



# Critérios de alocação

---

## iv. Próximo encaixe (*next-fit*) :

Variante da anterior (*first-fit*)

Percorrer a lista **a partir** da última área alocada ou liberada

⇒ Uso das áreas livres distribuído de forma mais **homogênea** no espaço de memória.



# Critérios de alocação



---

**Resultado de pesquisas mostram que:**

Melhor encaixe e primeiro encaixe têm melhores resultados!

→ **Bem mais rápido**

# Ordem de alocação de processos

Best fit ou First fit?

300k, 25k, 125k e 50k



## b - Desfragmentação



# Desfragmentação

---

Áreas de memória usadas pelos processos são **movidas** na memória de forma a concatenar as áreas livres e diminuir a fragmentação.

# Desfragmentação

---

Áreas de memória usadas pelos processos são **movidas** na memória de forma a concatenar as áreas livres e diminuir a fragmentação.

## **IMPORTANTE LEMBRAR QUE...**

...ao mover um processo na memória, suas **informações de alocação** (registrador base ou tabela de segmentos) devem ser ajustadas.

# Desfragmentação

---

**PORTANTO,**

**nenhum** processo pode executar durante a desfragmentação.

É importante que esse procedimento seja executado **rapida- e esporadicamente**.

# Desfragmentação

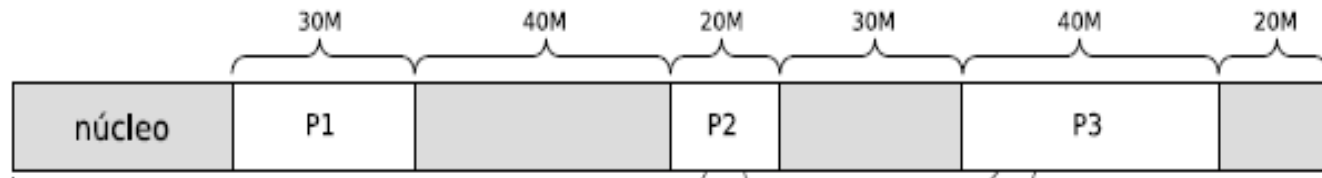
---

Um problema de otimização combinatória

⇒ Solução ótima pode ser difícil de calcular.

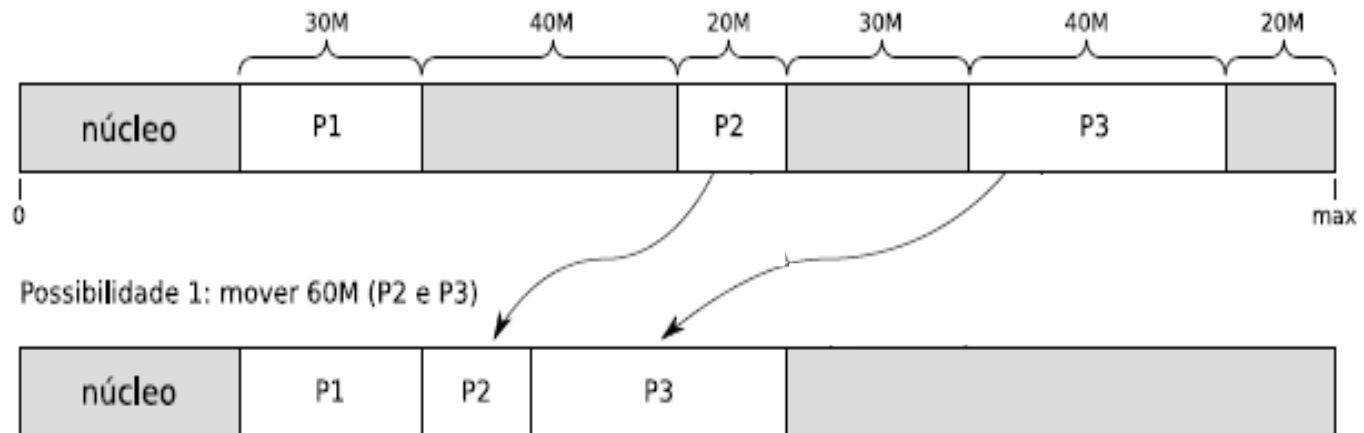
Próximo exemplo: 3 soluções com mesmo resultado e custos distintos.

# Desfragmentação

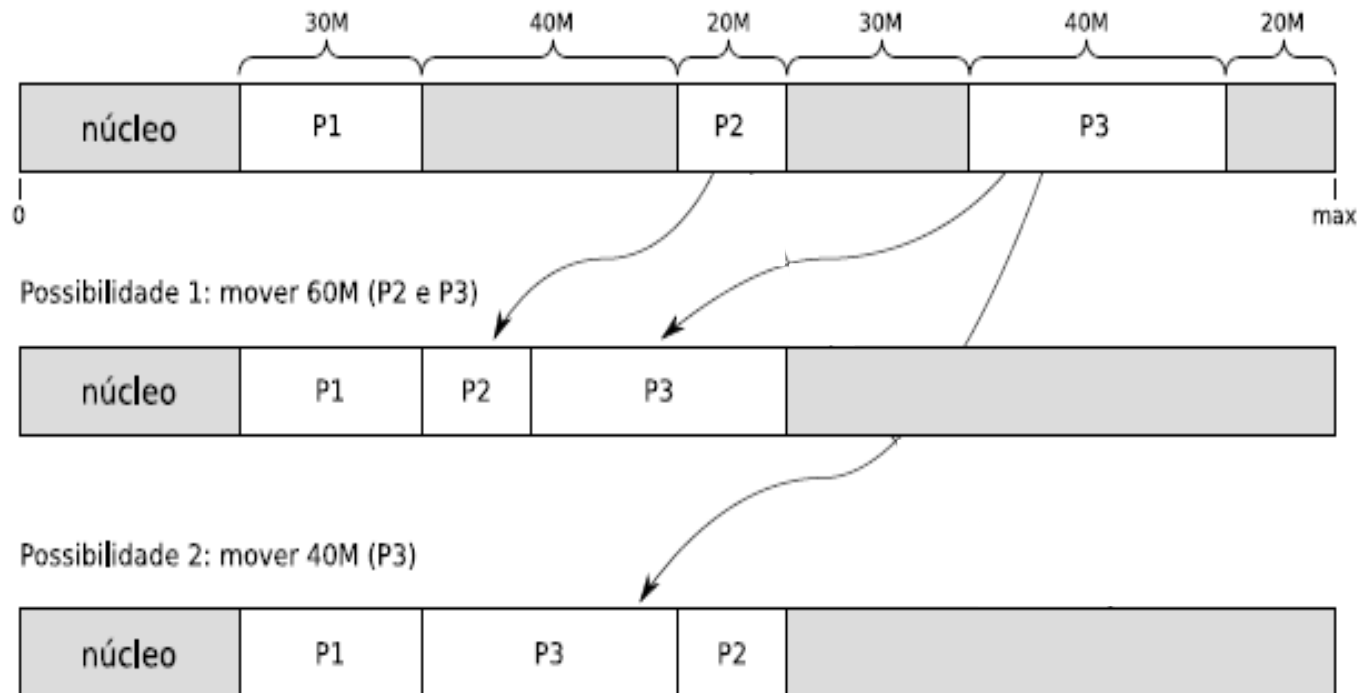




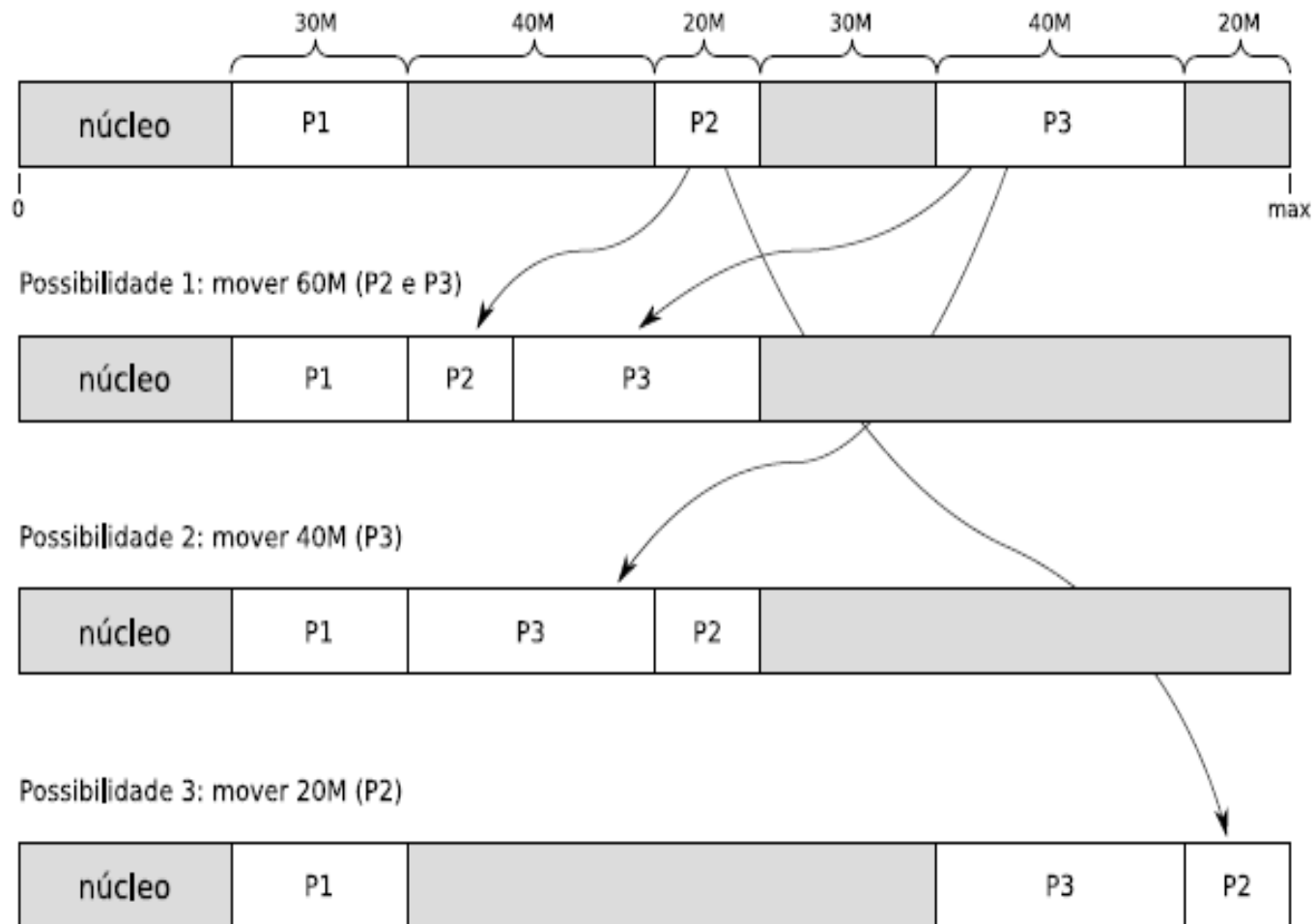
# Desfragmentação



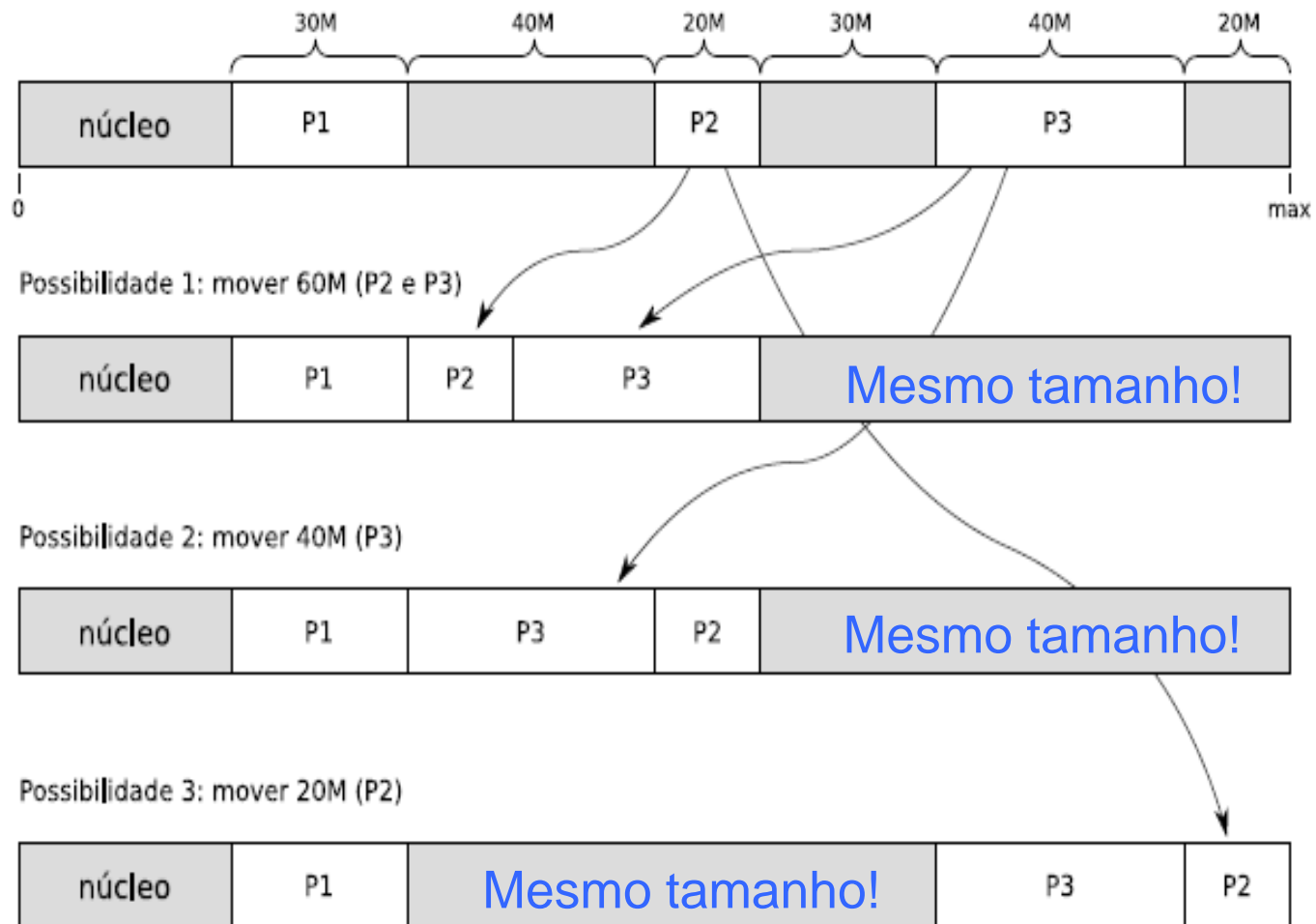
# Desfragmentação



# Desfragmentação



# Desfragmentação

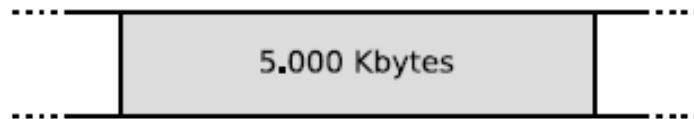


A decorative graphic on the left side of the slide, consisting of a vertical light blue bar, a vertical orange bar, and a grey rectangle. A horizontal dark blue bar extends from the grey rectangle across the top of the slide.

## **2. Fragmentação interna**

# Fragmentação interna

Pode ocorrer dentro das áreas alocadas aos processos (partição ou página/quadro).



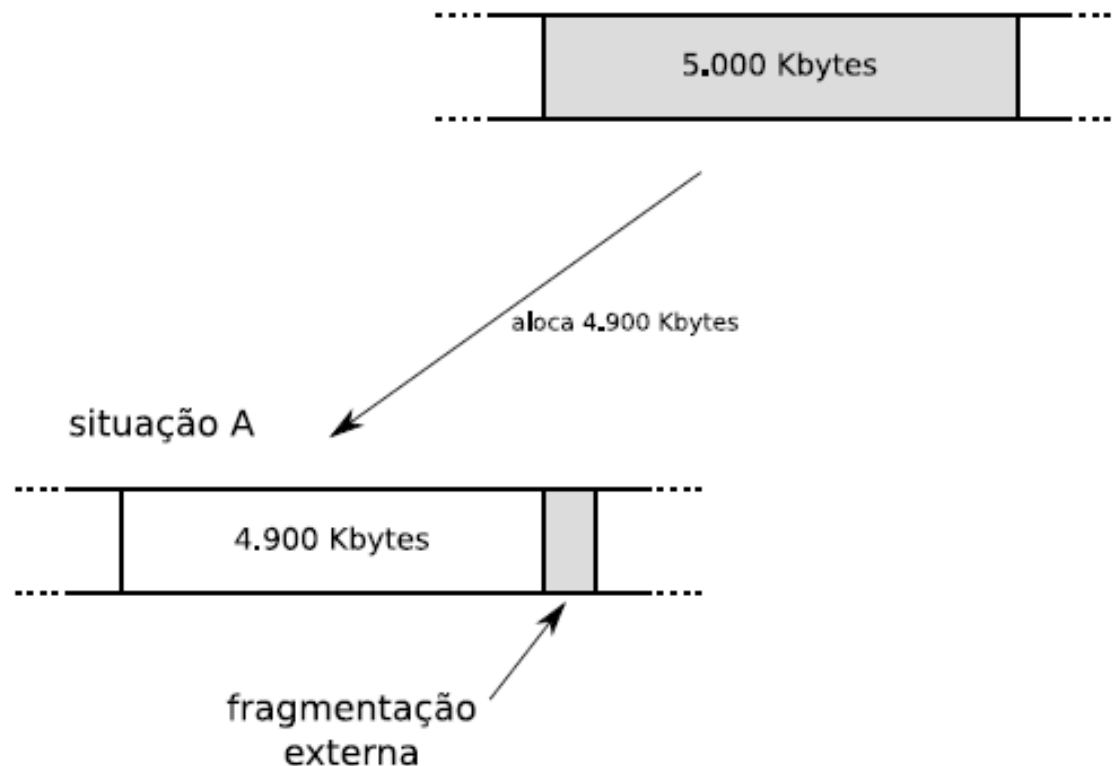
Exemplo: considere

Partição Livre: 5.000 kbytes

Problema: alocar 4.900 kbytes

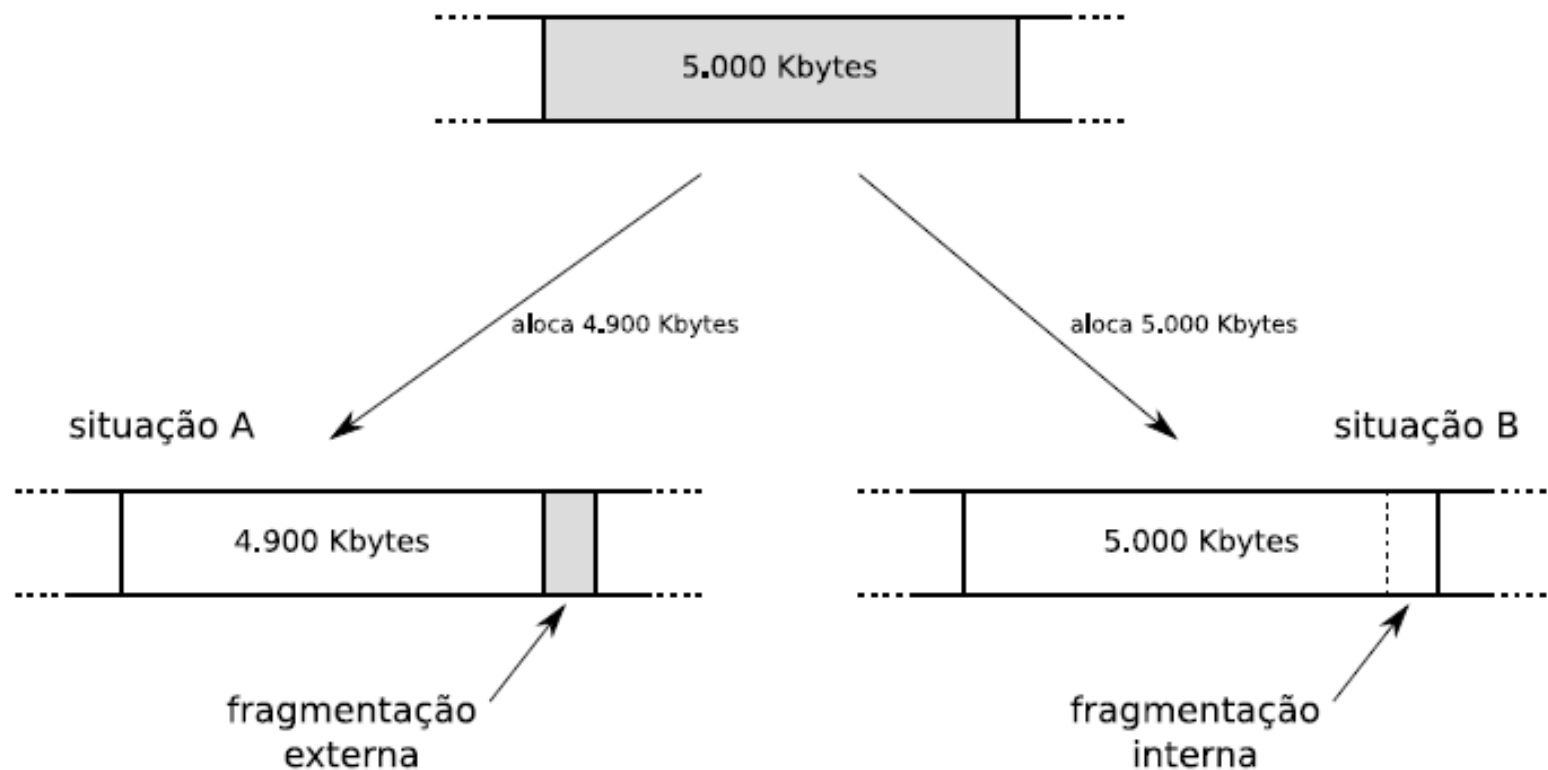
# Fragmentação interna

Pode ocorrer dentro das áreas alocadas aos processos (partição ou página/quadro).



# Fragmentação interna

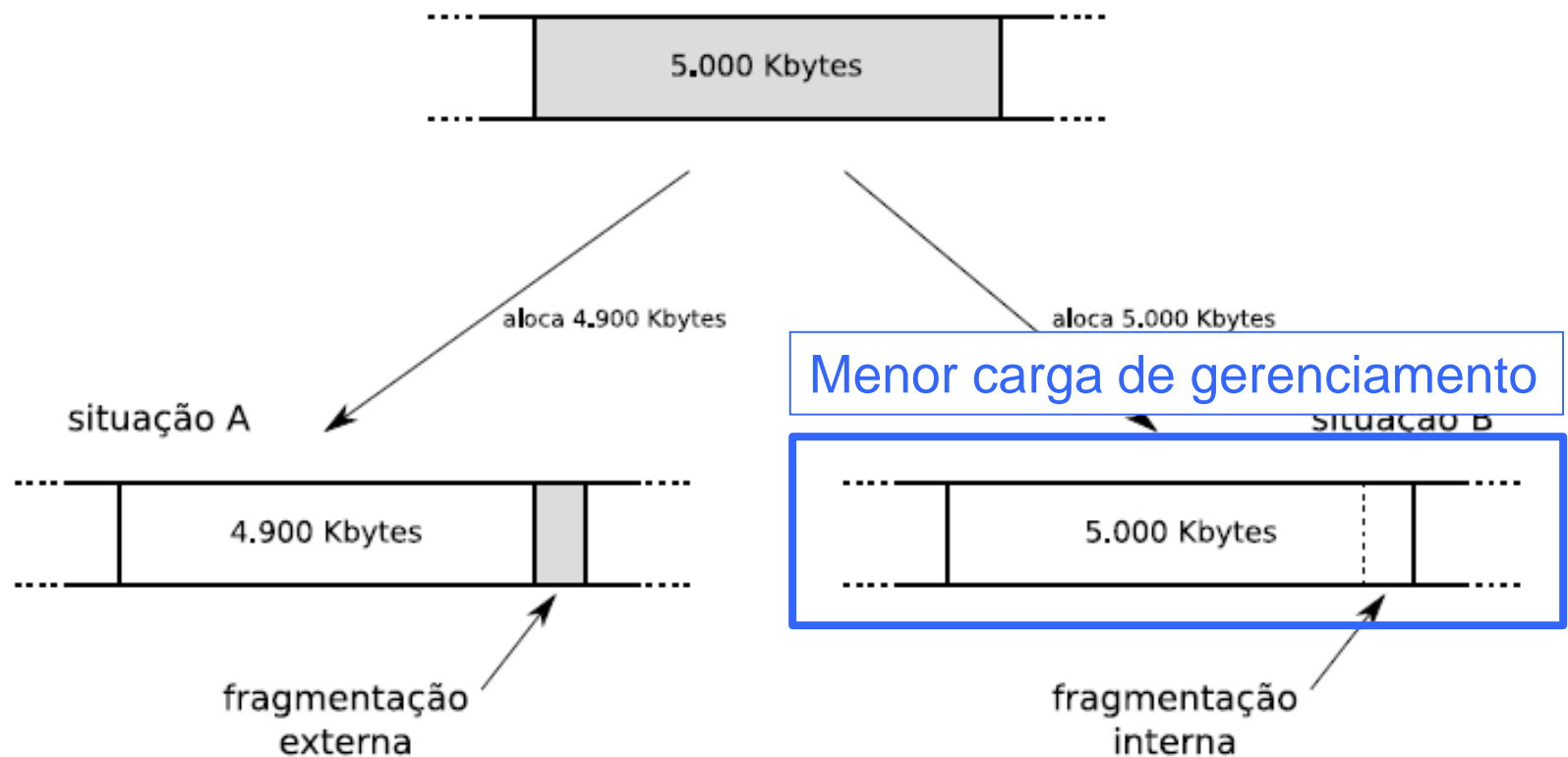
Ou fora das áreas alocadas aos processos (partição ou página/quadro).





# Fragmentação interna

Ou fora das áreas alocadas aos processos (partição ou página/quadro).



# Fragmentação interna

---

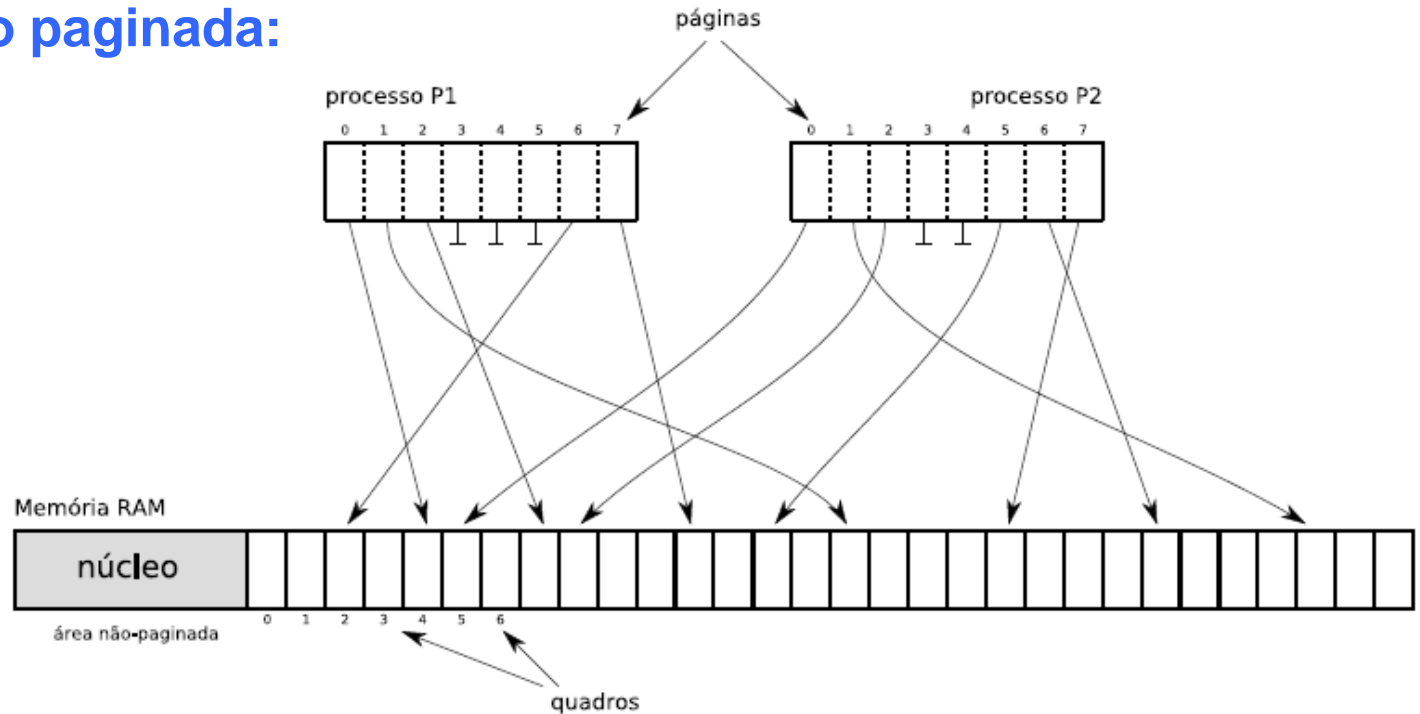
**Afeta todas as formas de alocação.**

**Alocações contígua e segmentada sofrem menos com esse problema.**

Pois o nível de arredondamento das alocações pode ser decidido caso a caso.

# Fragmentação interna

## Alocação paginada:



Tamanho de página: 4 kB

Alocação de 550.000 bytes (134,284 páginas) → 552.960 bytes (135 páginas) → 2.960 bytes a mais que o necessário

# Fragmentação interna

---

**Em média, para cada processo haverá uma perda de  $1/2$  página.**

**Uma forma de minimizar a perda por fragmentação interna:**

- Usar páginas de menor tamanho (2K, 1K, 512 bytes ou ainda menos).

# Fragmentação interna

---


**Em média, para cada processo haverá uma perda de  $1/2$  página.**

**Uma forma de minimizar a perda por fragmentação interna:**

- Usar páginas de menor tamanho (2K, 1K, 512 bytes ou ainda menos).

**Porém:**

- Mais páginas por processo;
- Tabelas de páginas maiores e com maior custo de gerência.



---

# **Sistemas Operacionais**

## **Parte II Compartilhamento de memória**

# Compartilhamento de memória

---

- Memória RAM é um recurso caro
- Deve ser usado de forma eficiente

# Compartilhamento de memória

---

- Memória RAM é um recurso **caro**
- Deve ser usado de forma **eficiente**

**Fato:** É comum ter várias instâncias do mesmo programa em execução:

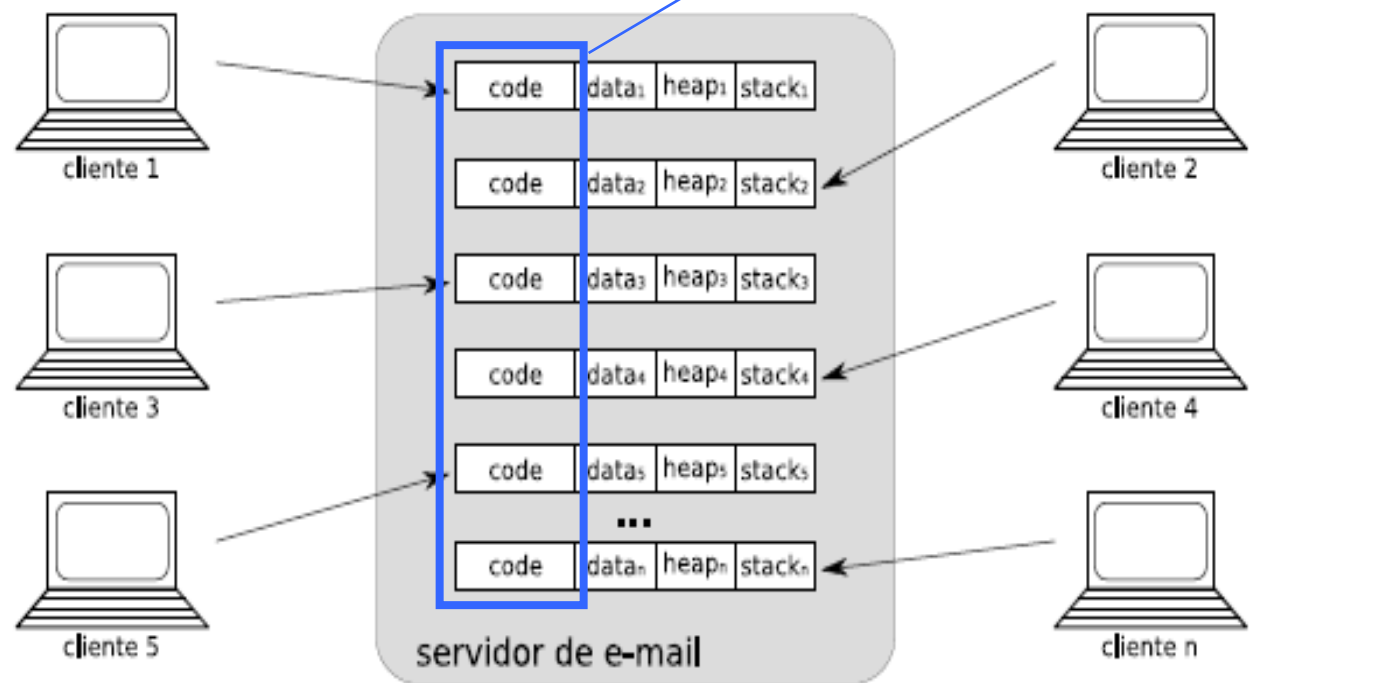
**Ex.:** Várias instâncias de editores de texto, de navegadores, etc.



# Compartilhamento de memória

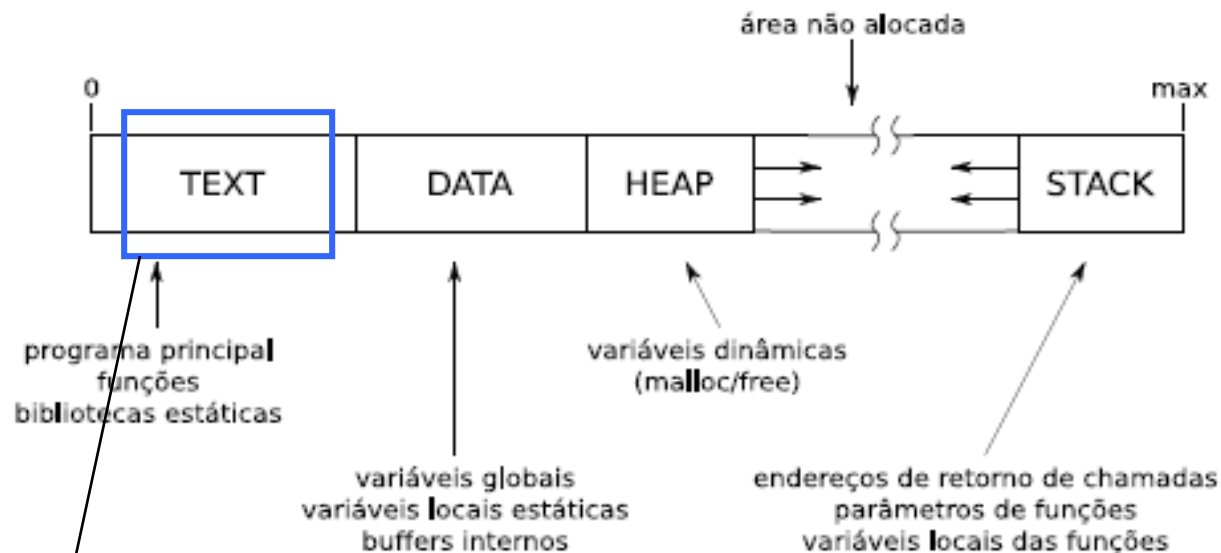
Em servidores: centenas ou milhares de instâncias do mesmo programa carregadas na memória.

Várias instâncias do mesmo processo:



# Compartilhamento de memória

Estrutura típica da memória de um processo:

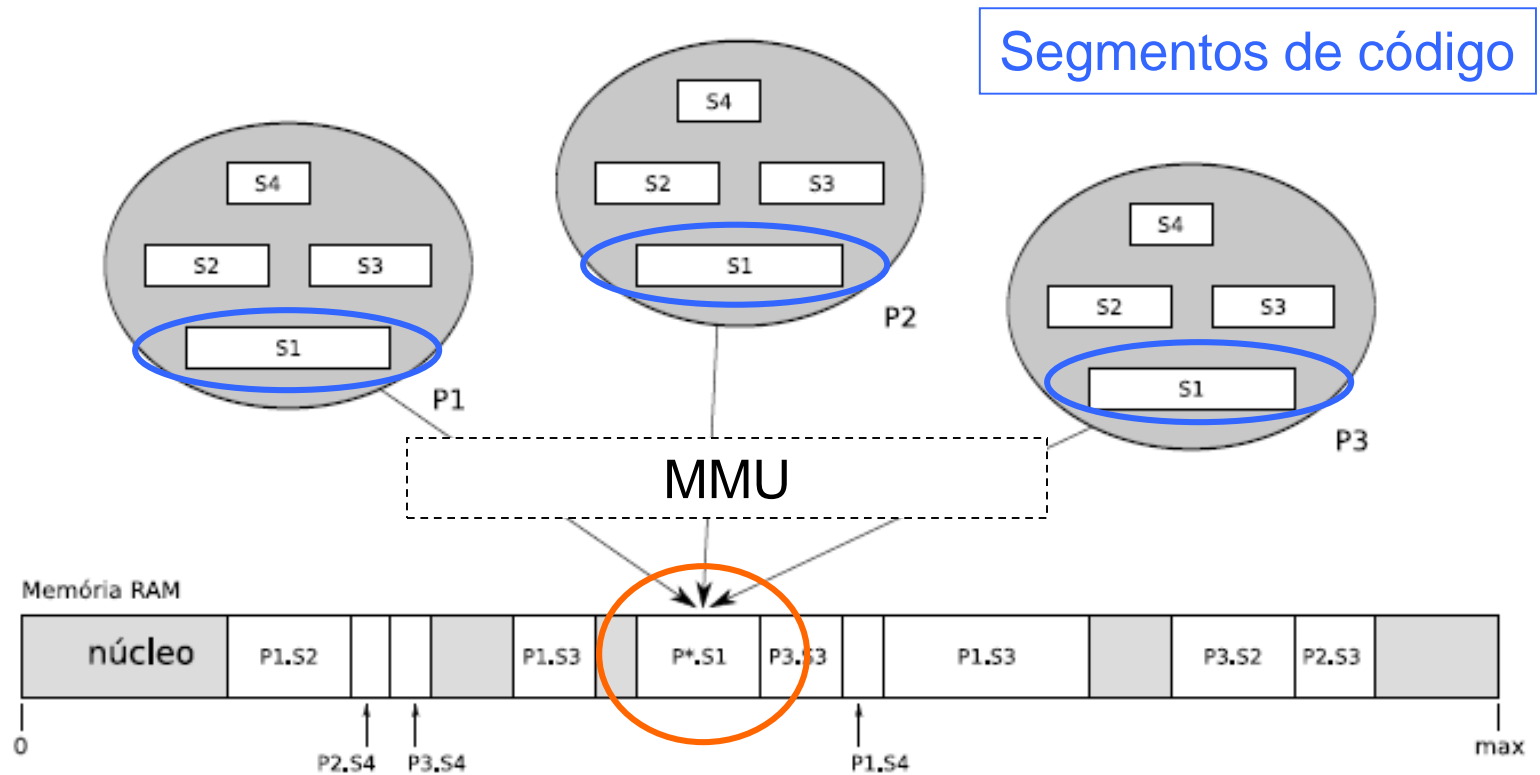


Área de código não é alterada durante execução.

⇒ Possível compartilhar essa área entre todos os processos que executam o mesmo código!

# Compartilhamento de memória

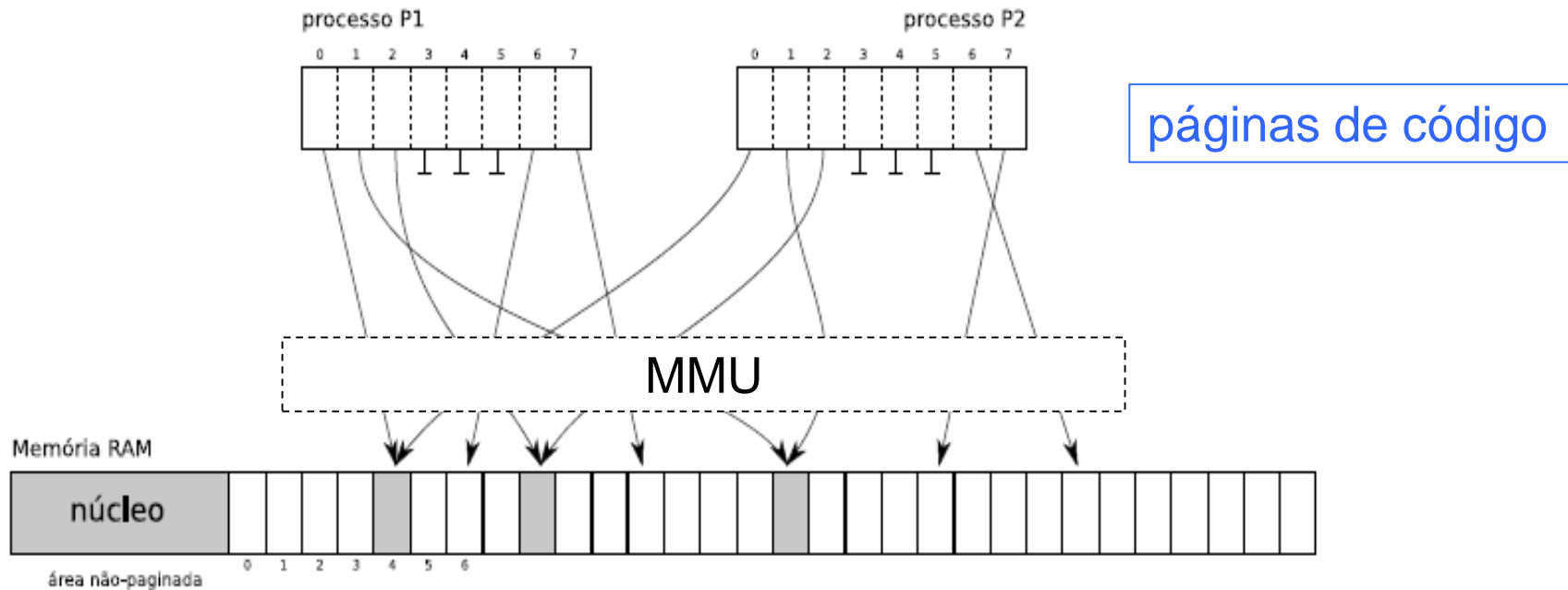
## Segmentação:



Economia de RAM

# Compartilhamento de memória

## Paginação:



Economia de RAM

# Compartilhamento de memória

---

## Economia de memória

Ex: Processador de textos necessita de 100MB

***60MB*** ocupados por ***código executável***.

## Sem compartilhamento:

10 instâncias do editor consumiriam 1.000 MB de memória.

# Compartilhamento de memória

---

## Economia de memória

Ex: Processador de textos necessita de 100MB

**60MB** ocupados por **código executável**.

## Sem compartilhamento:

10 instâncias do editor consumiriam 1.000 MB de memória.

## Com compartilhamento:

Consumo cairia para 460 MB (**60MB** + 10×40MB).

# + Compartilhamento de memória

---

- Compartilhamento de memória não é implementado apenas com **áreas de código**;

# + Compartilhamento de memória

---

- Compartilhamento de memória não é implementado apenas com **áreas de código**;
- Em princípio, **toda área de memória protegida contra escrita** pode ser **compartilhada**;
- Áreas de **dados constantes**, como tabelas de constantes, textos de ajuda, etc.  
→ Mais economia de memória.



# **++ Compartilhamento de memória**

---

**Copiar-ao-escrever (*COW: Copy-On-Write*)**

**Uma forma mais agressiva de compartilhamento de memória.**

# ++ Compartilhamento de memória

---

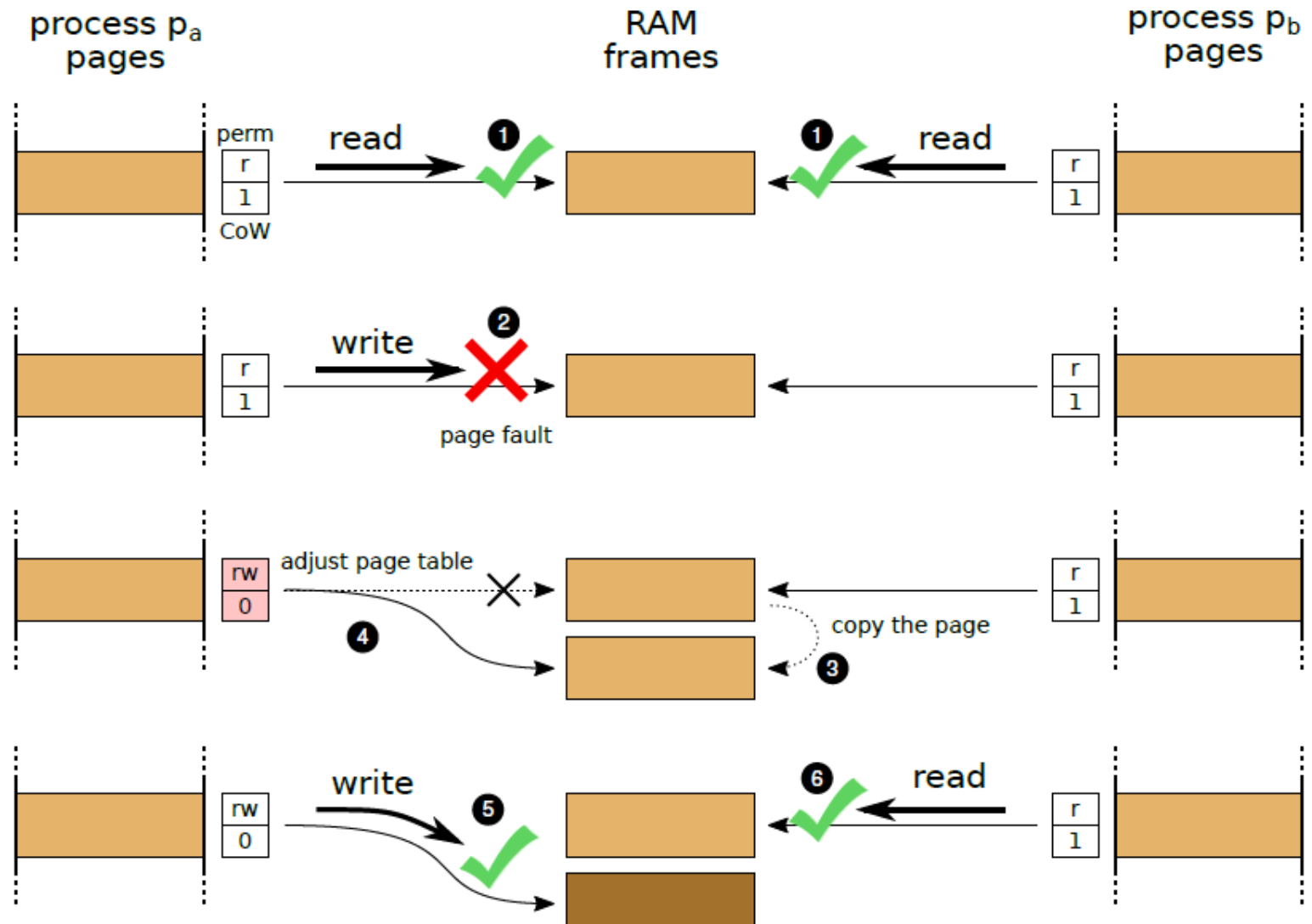
## Copiar-ao-escrever (*COW: Copy-On-Write*)

Uma forma mais agressiva de compartilhamento de memória.

Todas as áreas de memória de um processo são passíveis de compartilhamento.

**Condição:** conteúdo “ainda” não modificado.

# Copy-On-Write



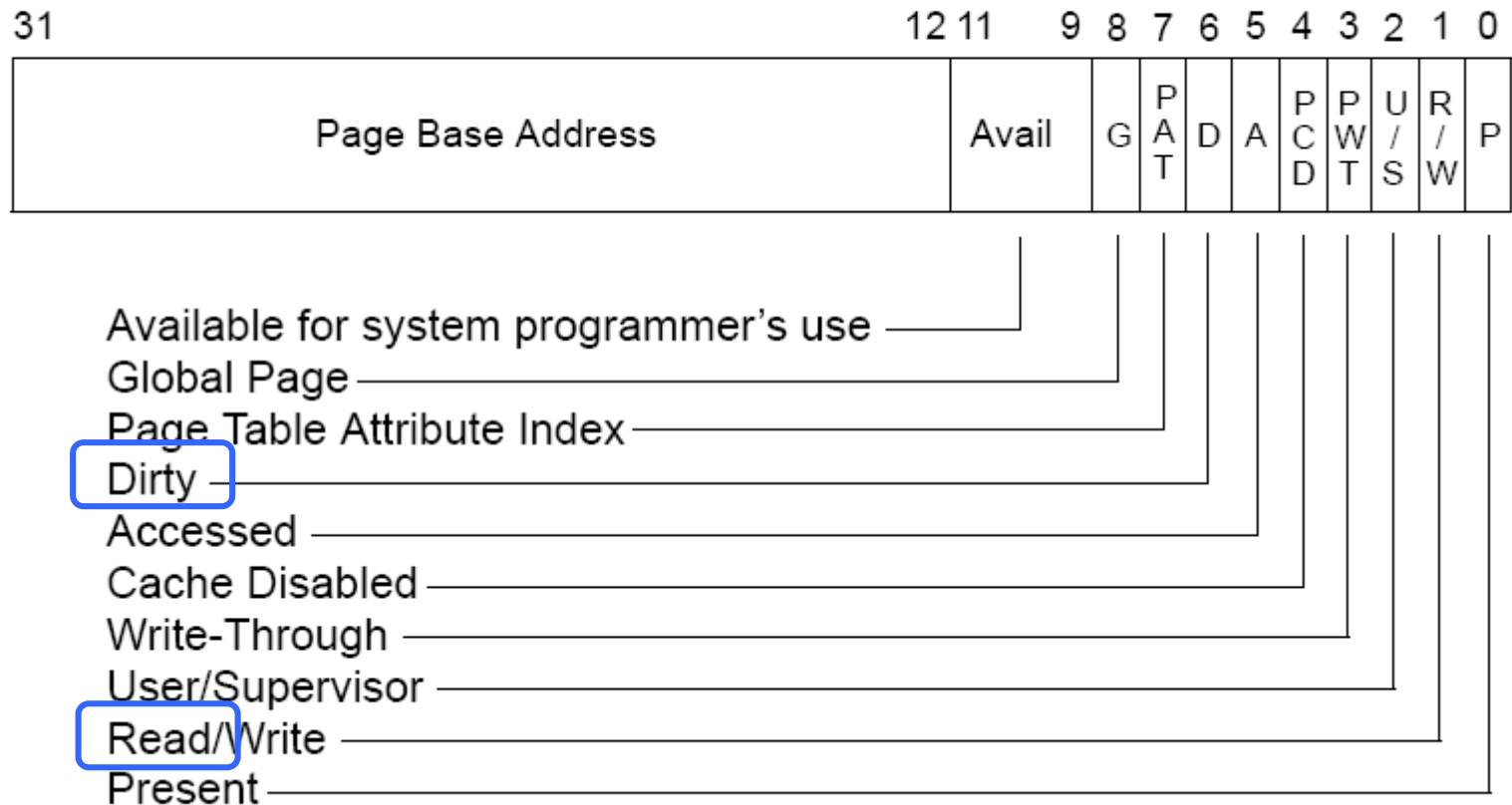
# Copy-On-Write

---

1. Ao carregar um **novο processo**, todas as áreas de memória do processo são protegidas contra escrita → *flags* da tabela de páginas (ou de segmentos);
2. Quando o processo tentar **escrever** na memória, a MMU gera uma **interrupção** para o núcleo do sistema operacional (caso página protegida);
3. O sistema operacional ajusta os *flags* da área para **permitir a escrita** e devolve a execução ao processo;

A vertical bar chart with three bars. The top bar is grey, the middle bar is blue, and the bottom bar is orange. The bars are of different heights, with the grey bar being the tallest and the orange bar being the shortest.

\_\_\_\_\_



# Copy-On-Write

---

4. Processos subsequentes **idênticos** ao primeiro serão mapeados sobre as mesmas áreas de memória física do primeiro processo que **ainda** estiverem **protegidas contra escrita**;
5. Se um dos processos envolvidos tentar **escrever** em uma dessas **áreas compartilhadas**, a MMU gera uma **interrupção** para o núcleo;
6. O núcleo faz uma **cópia** separada daquela área física para o processo que deseja escrever nela e **desfaz** seu **compartilhamento**.

# Copy-On-Write

---

- Todo esse procedimento é feito de forma **transparente** para os processos envolvidos
- Esse mecanismo é **mais efetivo** em sistemas baseados em **páginas**, porque normalmente as páginas são menores que os segmentos.
- A maioria dos sistemas operacionais atuais (Linux, Windows, Solaris, FreeBSD, etc) usa esse mecanismo.

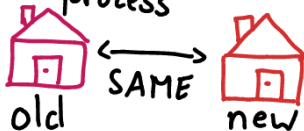
# Copy-On-Write

JULIA EVANS  
@b0rk

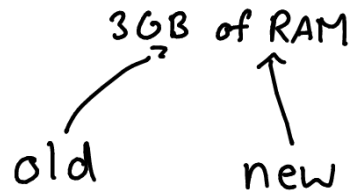
copy on write [drawings.jvns.ca](http://drawings.jvns.ca)

every time you start  
a new process on  
Linux, it does a

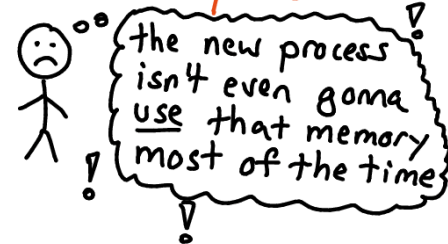
`fork()` "clone"  
which copies the parent  
process



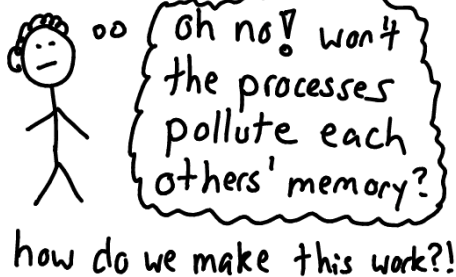
the cloned process  
has EXACTLY the  
same memory



copying all the memory  
every time we fork  
would be **slow** and a  
**waste of space**.



so Linux lets them  
share RAM instead  
of copying



Linux marks all the memory for  
both processes as **read-only** (in the  
page table)

①



I'm going to write  
to the shared  
memory!

②



UHHH that  
is not allowed!  
Linux! PAGE FAULT!

③

Linux

no problem!  
I will just make a  
copy of that piece  
of memory.

④

everyone is  
happy 😊