

# Sistemas Operacionais

## Gerenciamento de Memória

*Memória Principal*

Prof. Otávio Gomes

otavio.gomes@unifei.edu.br

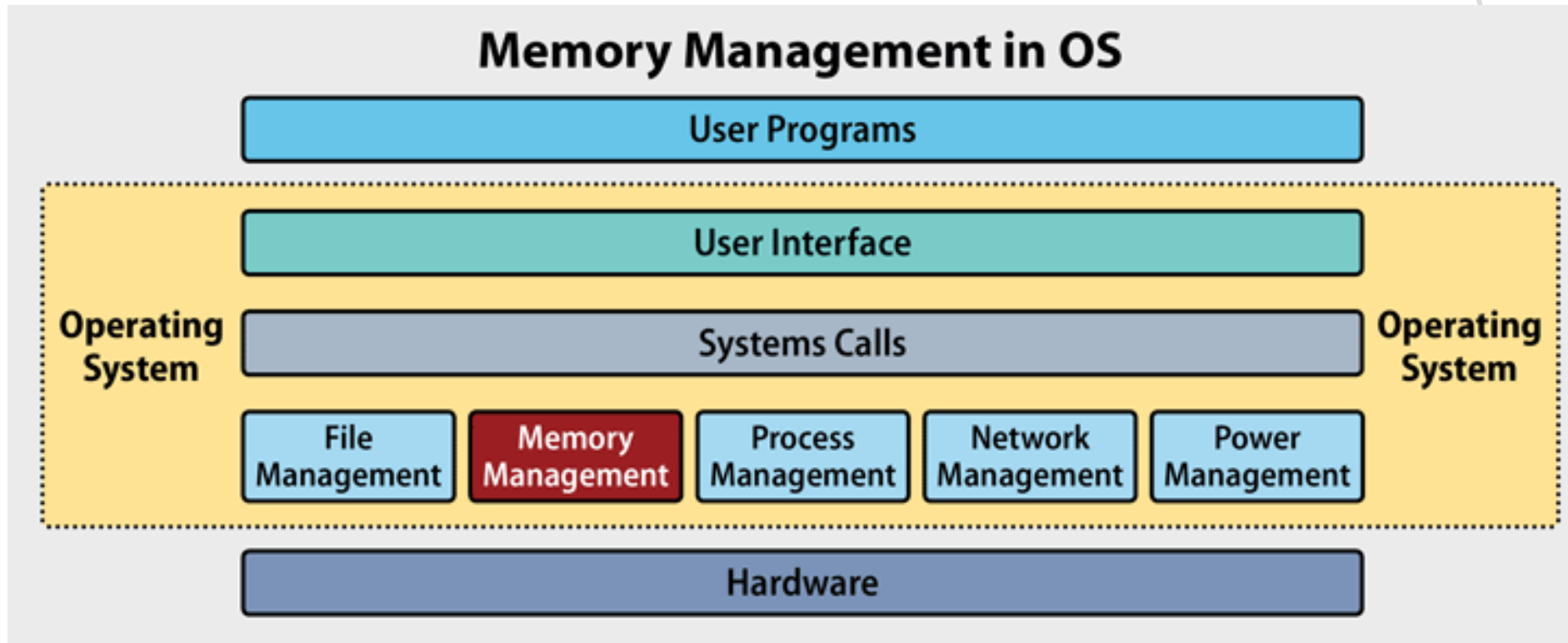


# Gerenciamento de Memória

- O gerenciamento de memória tem por objetivos:
  1. Oferecer uma **área de armazenamento** para os processos;
  - 2. Proteger** os processos contra falhas de terceiros; e
  3. Prover um **desempenho** satisfatório aos usuários.
- Deseja-se também compartilhamento de memória entre processos e acesso transparente à memória.
  - A memória é caracterizada como compartilhada se mais de um programa pode acessá-la simultaneamente. A memória compartilhada pode ser criada eletricamente (*hardware* destinado a este fim) ou logicamente (criada através de estruturas de dados especiais).

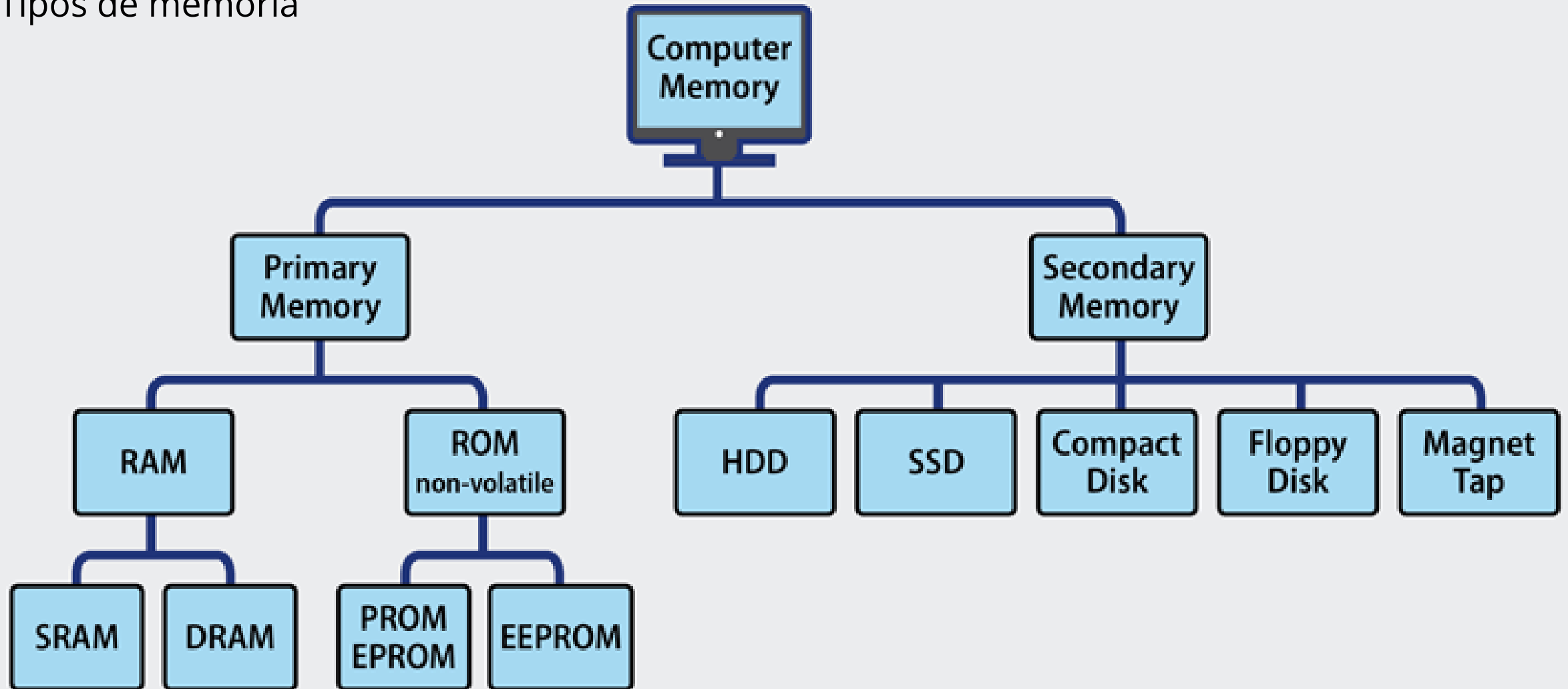
# Gerenciamento de Memória

- Idealmente, os programadores desejam uma memória que seja:
  - Grande
  - Rápida
  - Não-volátil
  - De baixo custo

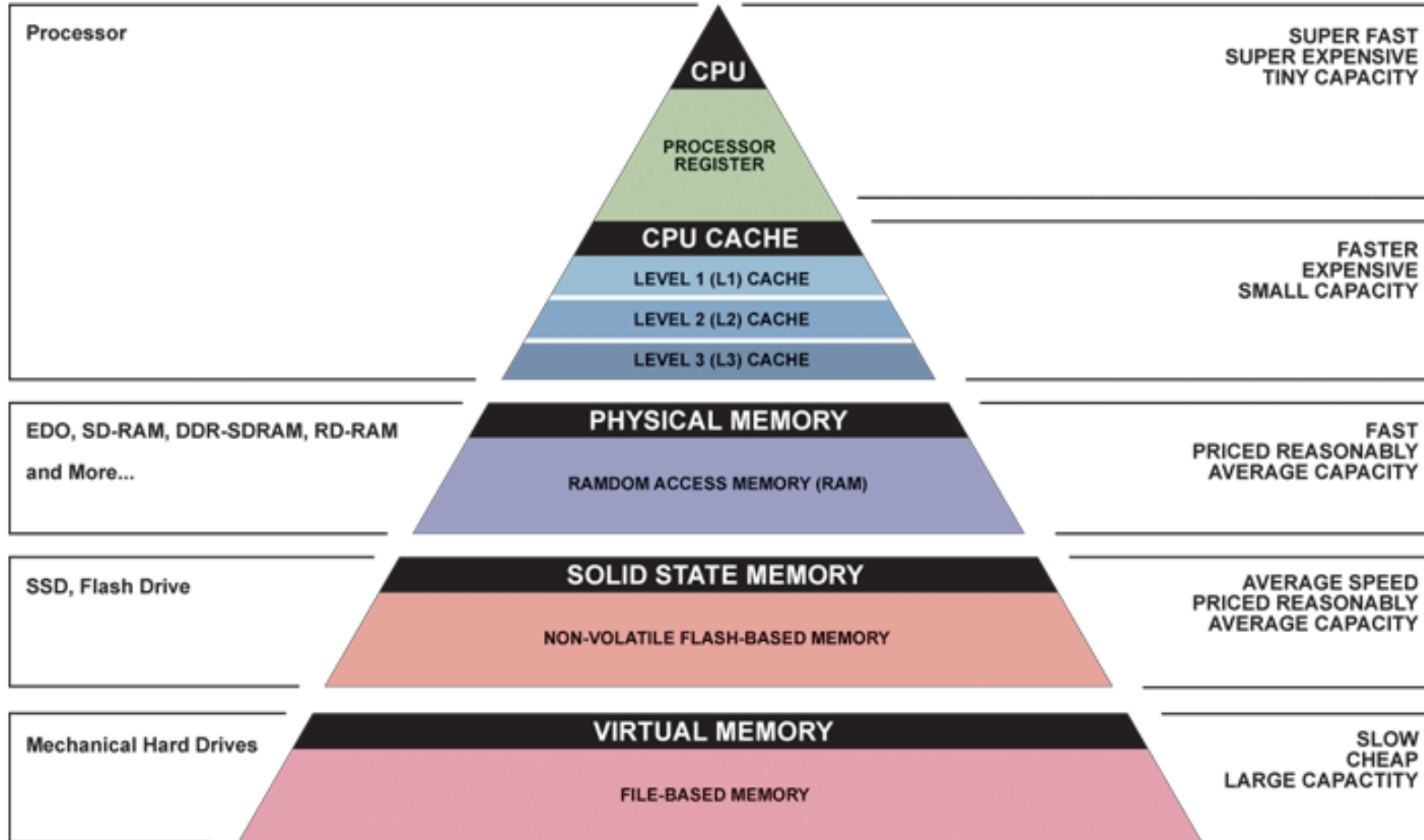


# Gerenciamento de Memória

Tipos de memória

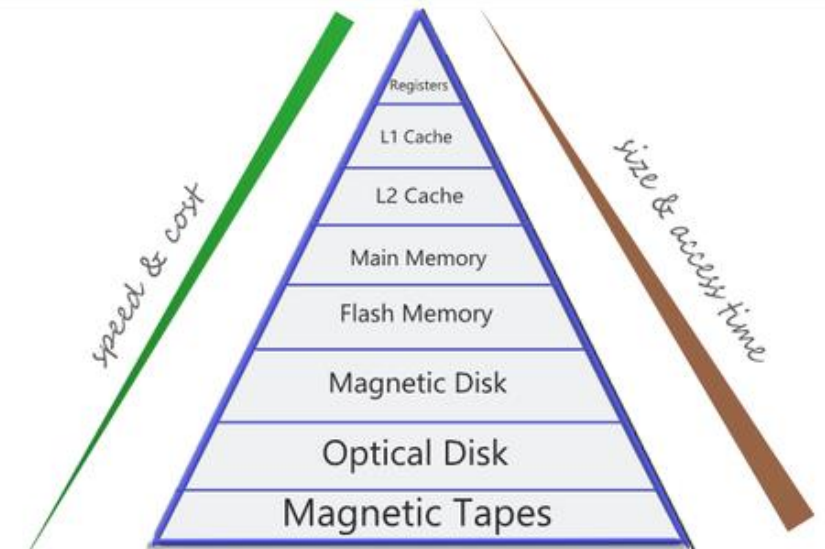
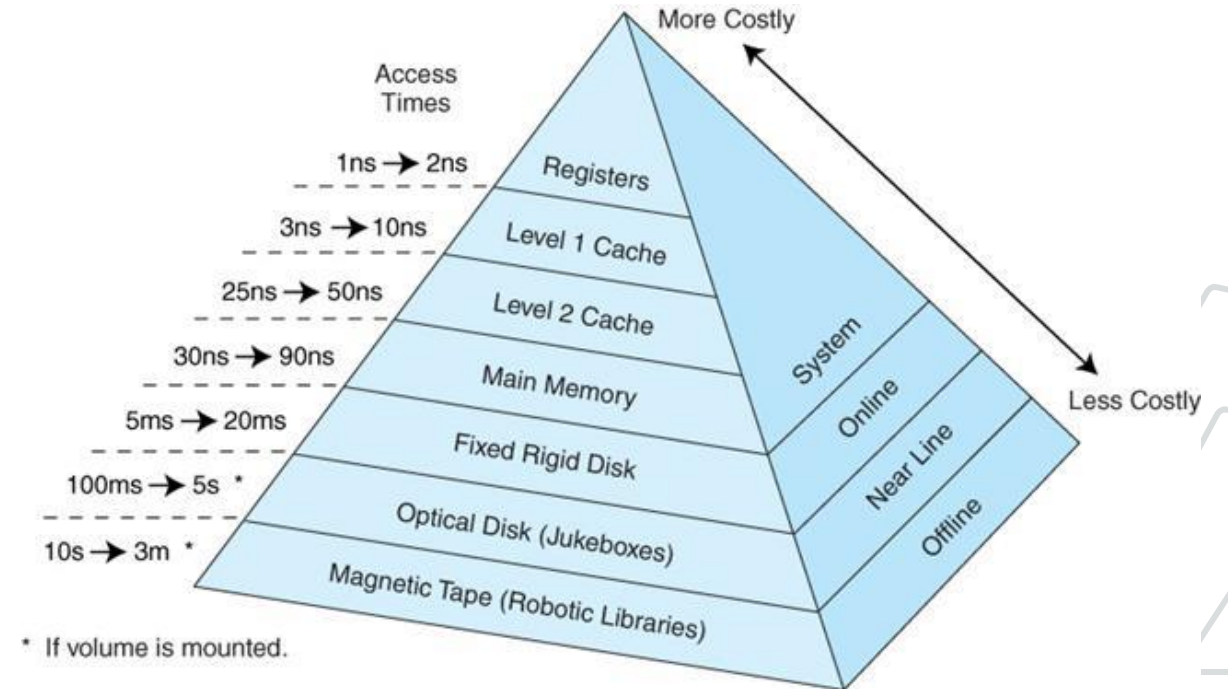
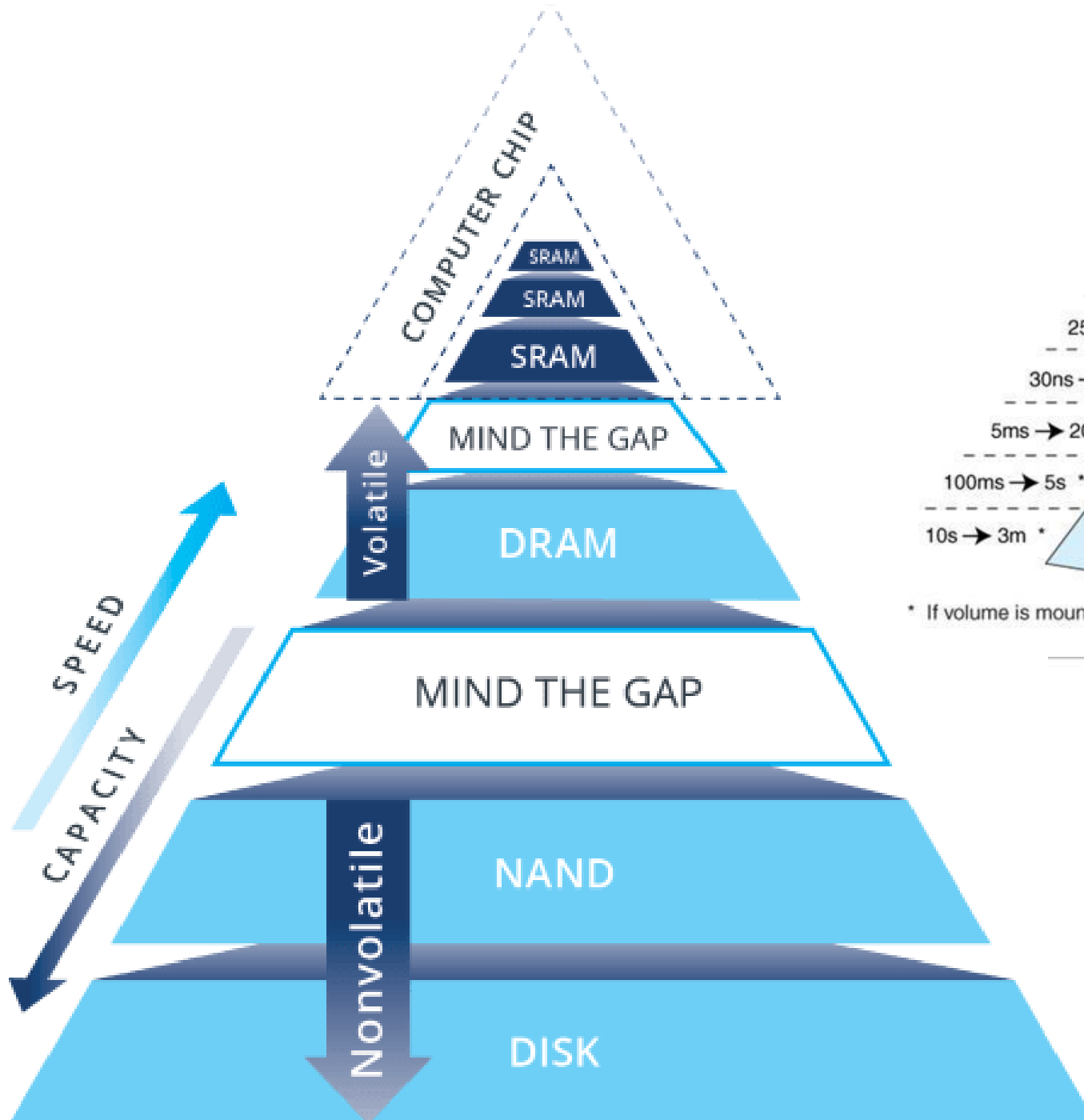


# Gerenciamento de Memória



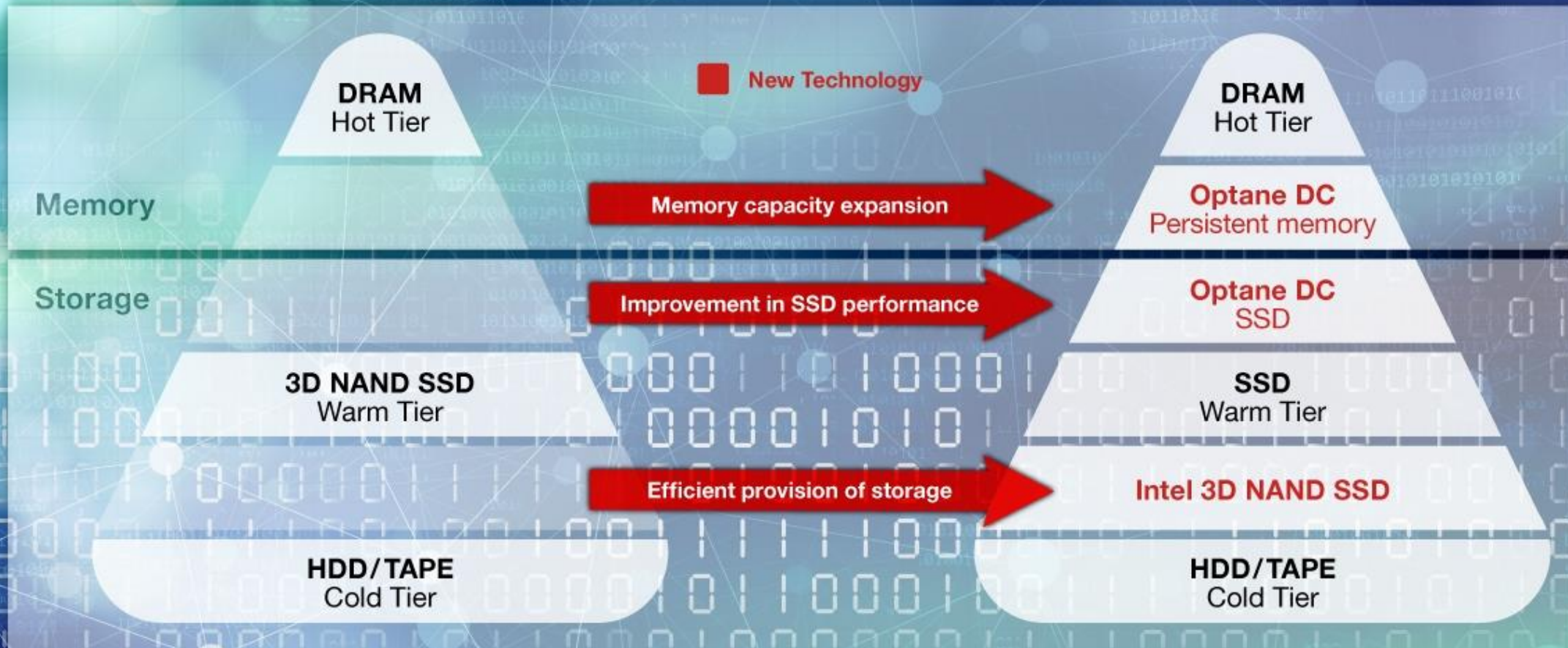
▲ Simplified Computer Memory Hierarchy  
Illustration: Ryan J. Leng

# Gerenciamento de Memória



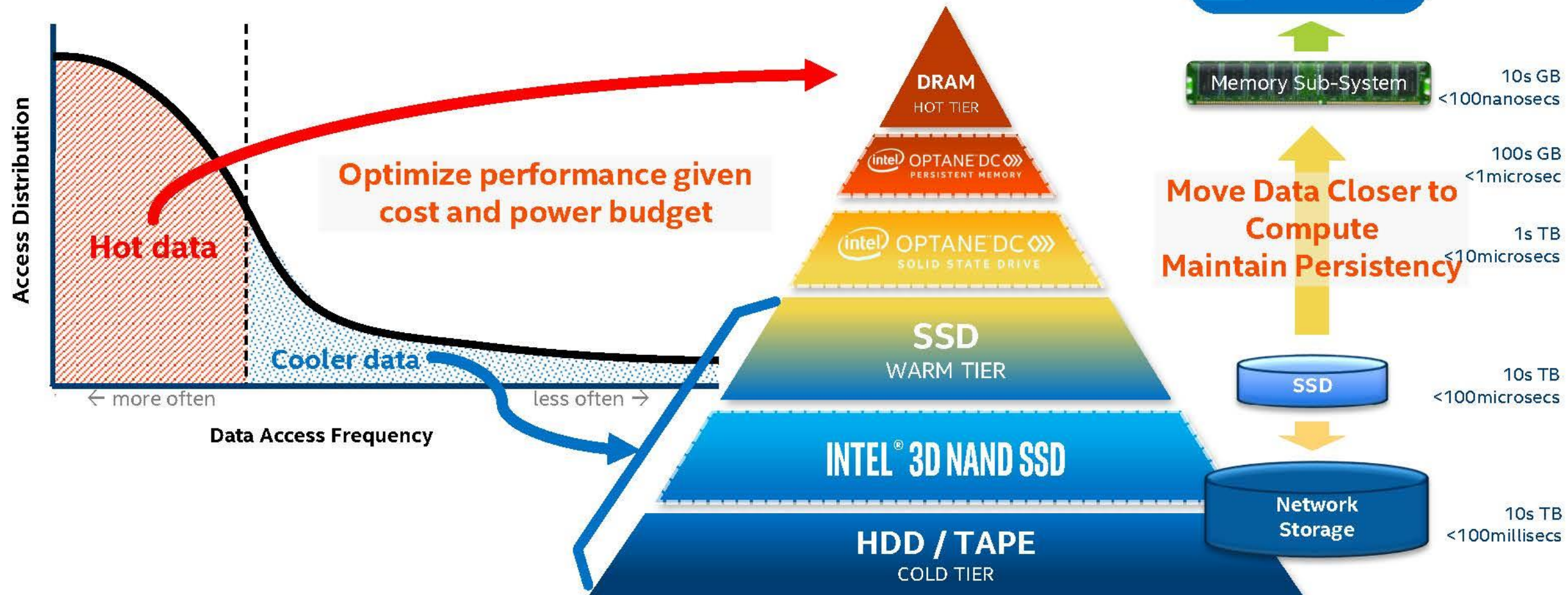


## New technologies changing memory hierarchy

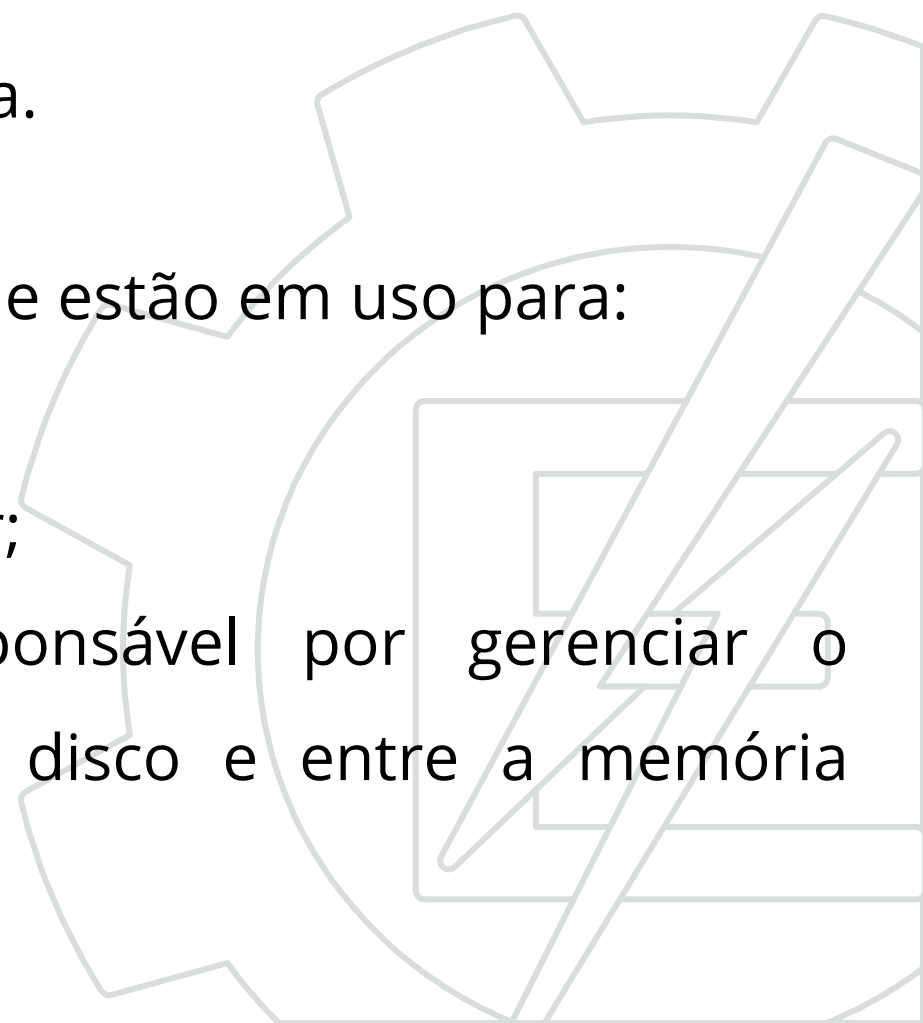




## GOAL: EFFICIENT DATA CENTRIC ARCHITECTURE





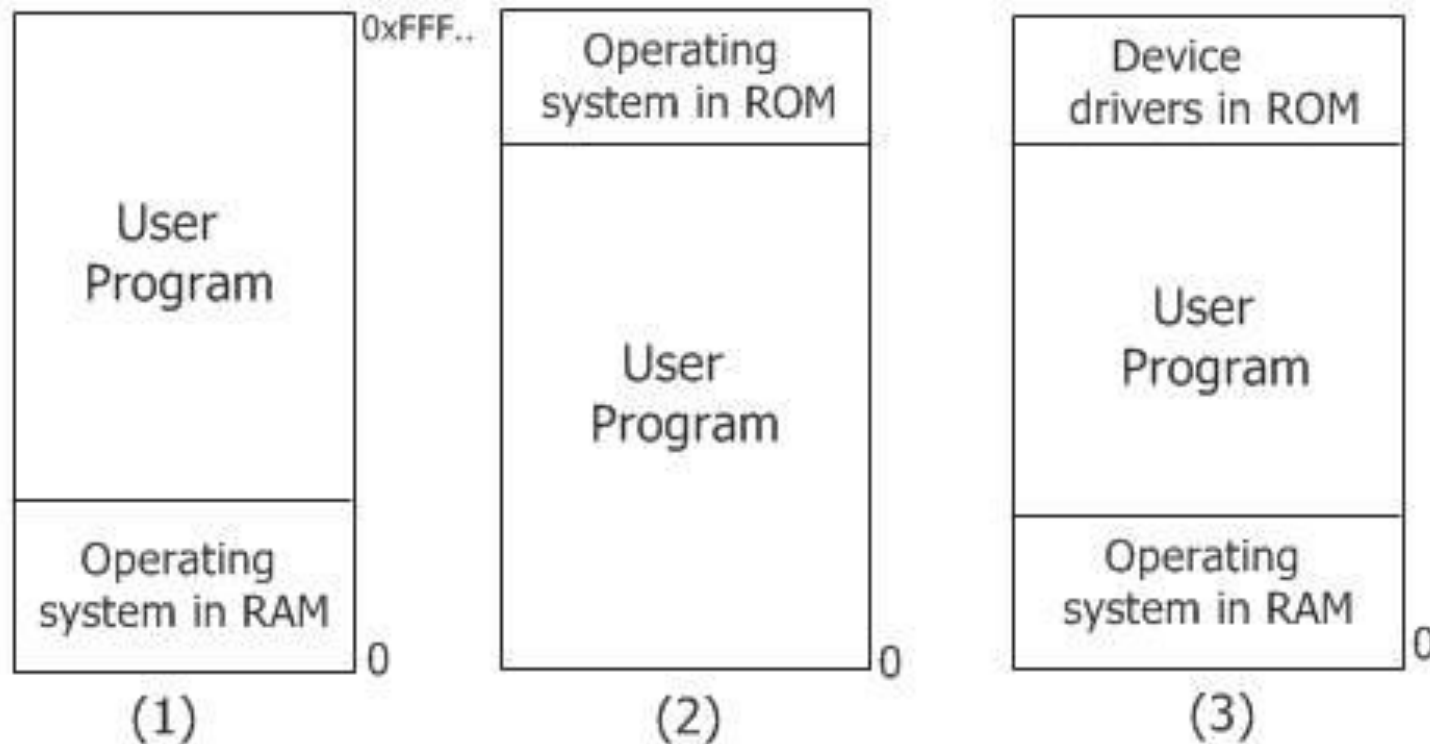
- Gerenciar a hierarquia de memória:
    - Espaços livres e ocupados;
    - Localizar e alocar processos e dados na memória.
  - Controlar as partes da memória (mapeamento) que estão em uso para:
    - **Alocar** processos quando estes precisarem;
    - **Liberar** memória quando um processo terminar;
    - **Tratamento** de *swapping* - processo responsável por gerenciar o chaveamento entre a memória principal e o disco e entre a memória principal e a cache.
- 

# Gerenciamento de Memória

## Monoprogramação

Somente um programa na memória principal.

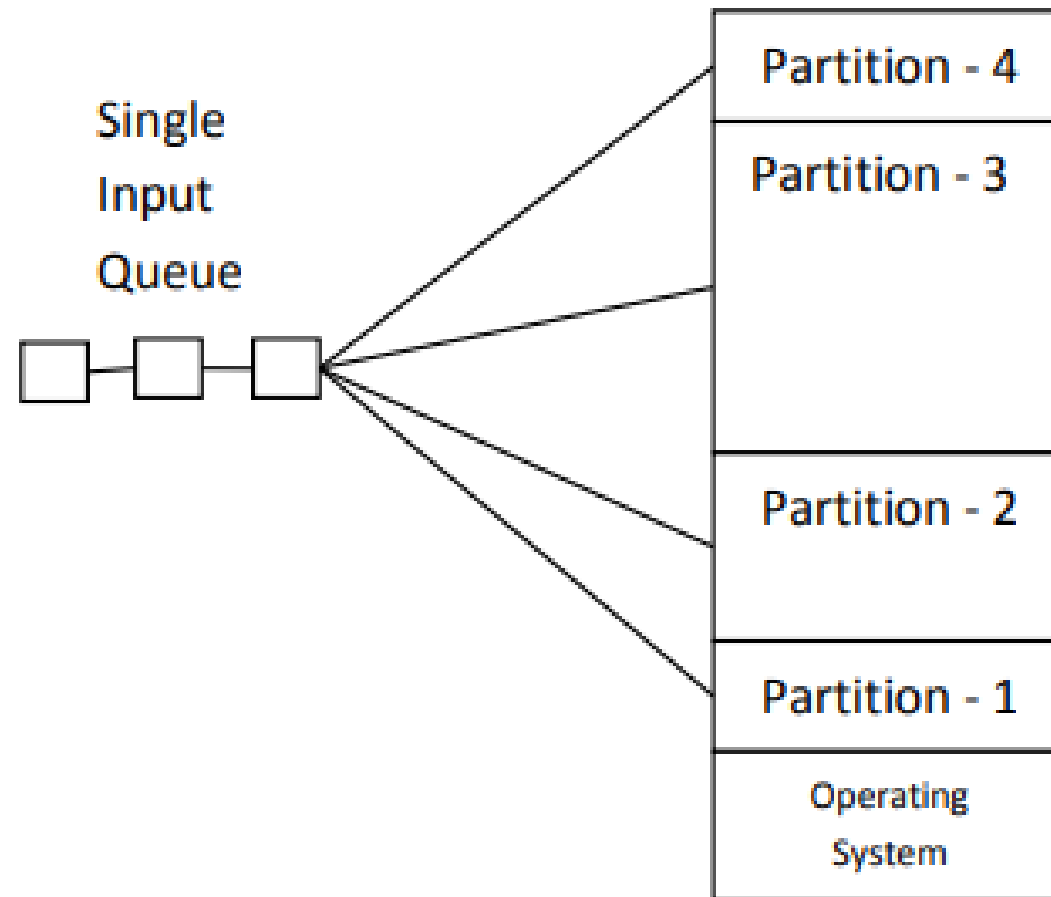
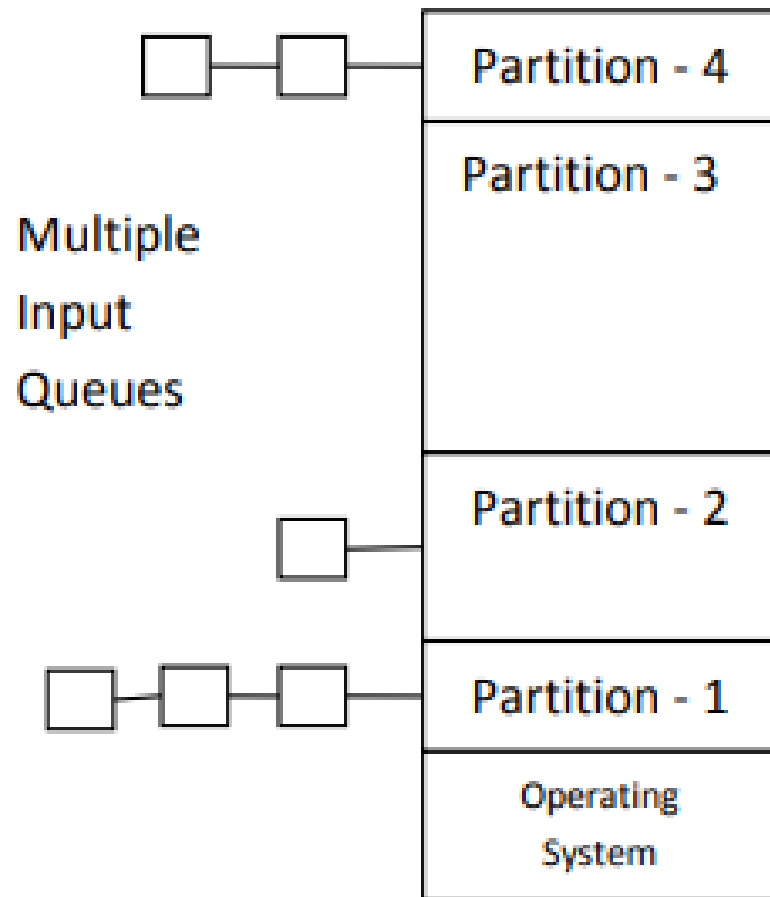
1. Utilizado em antigos *mainframes*
2. Usado em *handhelds*
3. Primeiros computadores pessoais



# Gerenciamento de Memória

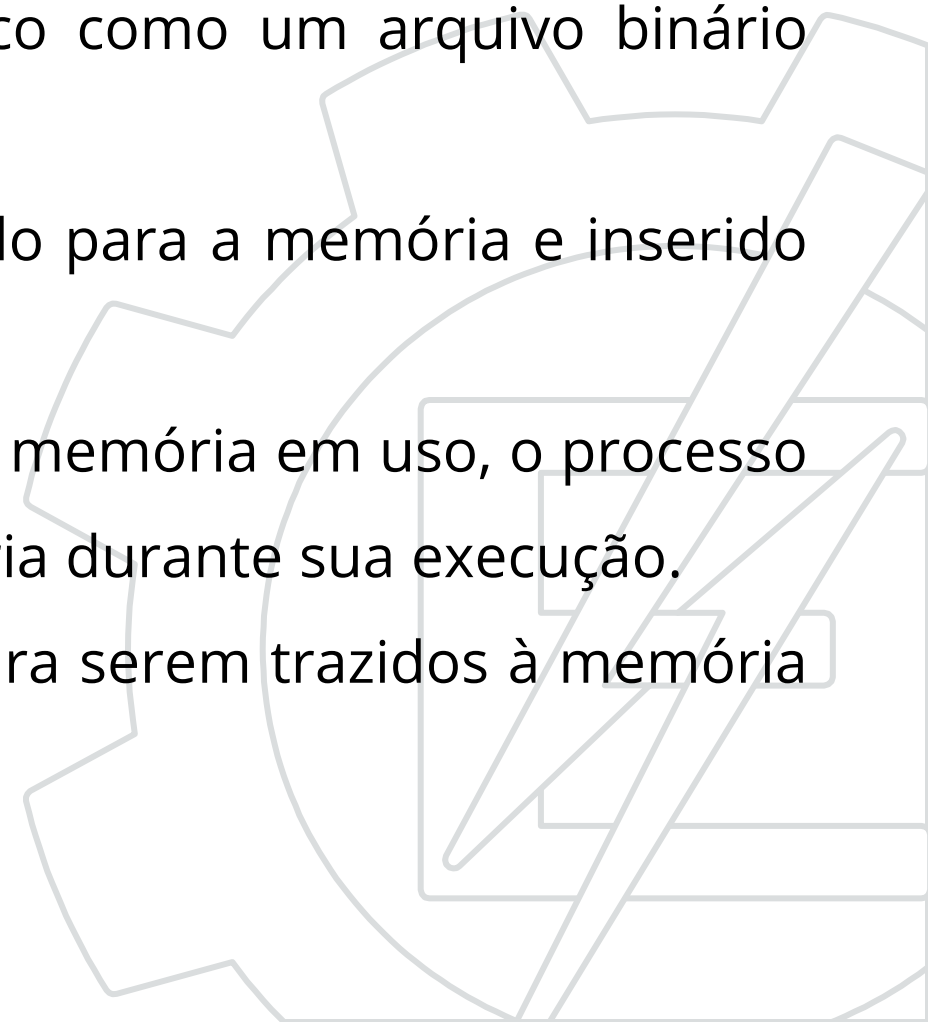
## Multiprogramação

- Divisão da memória em  $n$  partições de tamanho fixo, não necessariamente iguais;
- Filas (simples ou múltiplas) para o controle e execução dos *jobs*.



- **Vinculação de endereços**

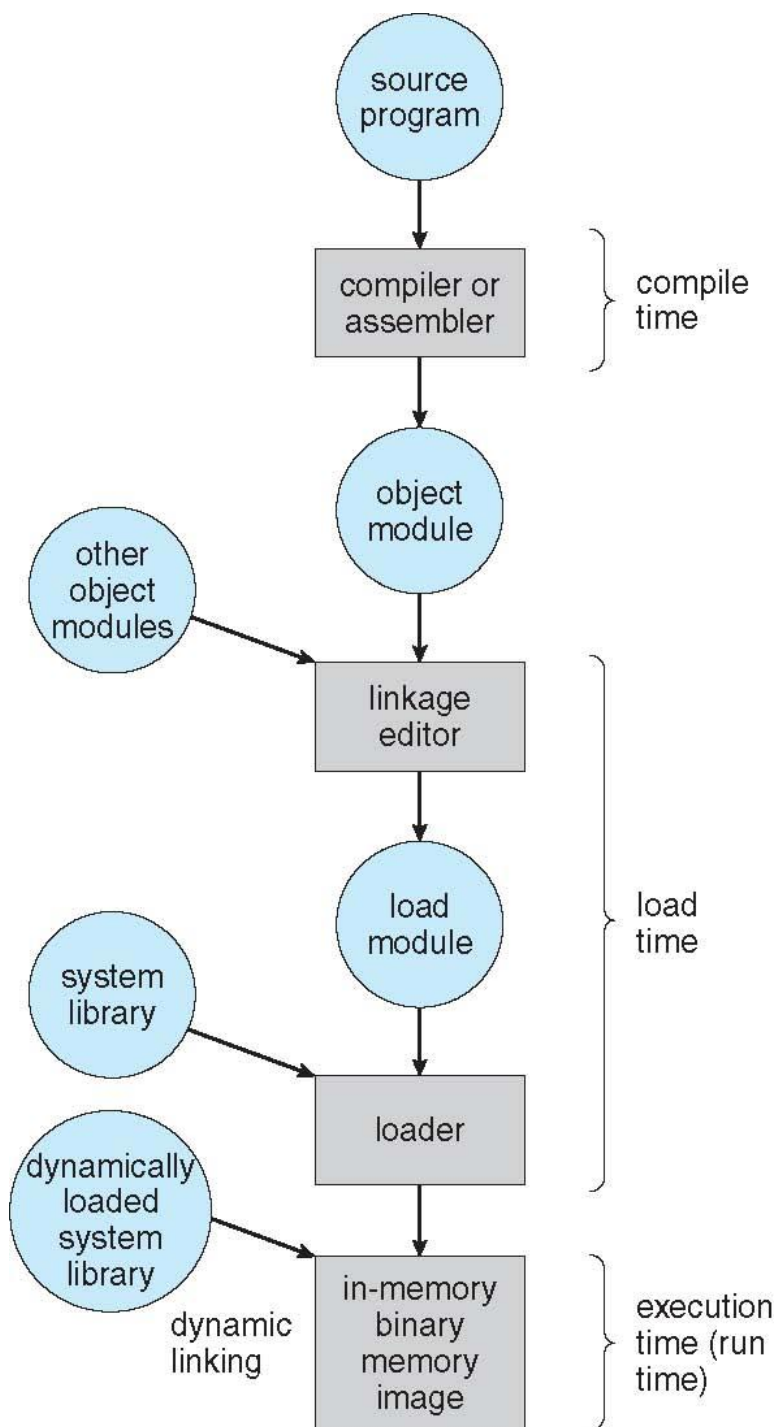
- Usualmente, um programa reside em um disco como um arquivo binário executável.
- Para ser executado, o programa deve ser trazido para a memória e inserido dentro de um processo.
- Dependendo do esquema de gerenciamento da memória em uso, o processo pode ser movimentado entre o disco e a memória durante sua execução.
- Os processos em disco que estão esperando para serem trazidos à memória para execução formam a fila de entrada





# Gerenciamento de Memória

## Vinculação de endereços

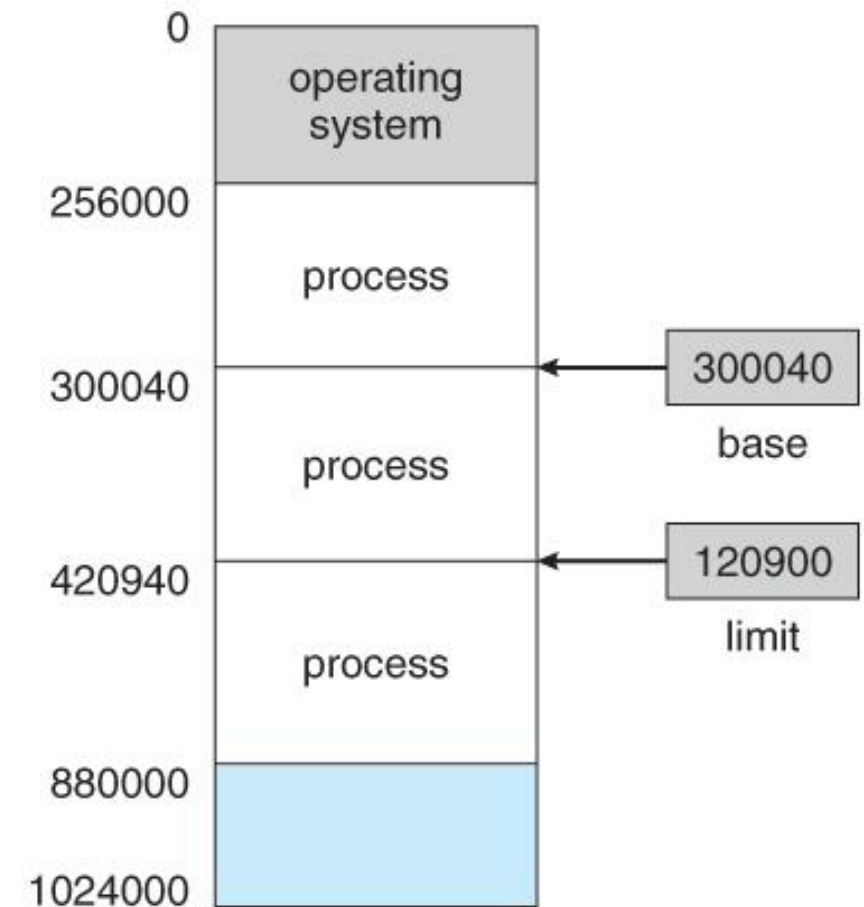
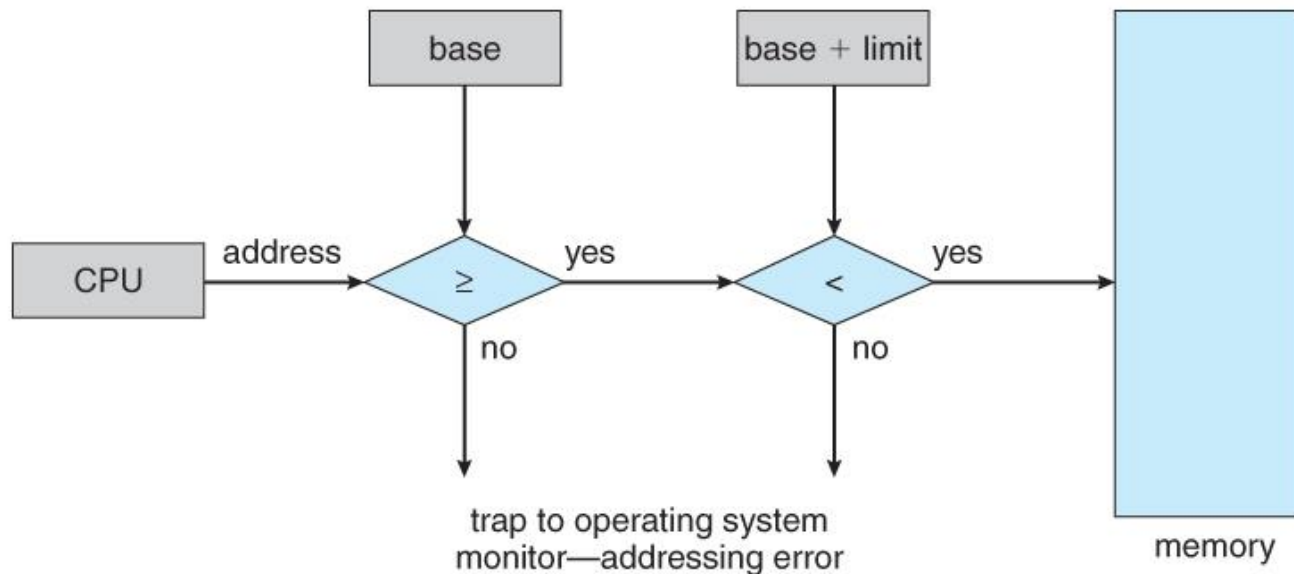


- Programas precisam ser carregados na memória e associados a um processo para poderem ser executados
- Fila de entrada: processos que estejam no disco esperando para serem carregados na mem.
- Há vários passos a serem seguidos e/ou realizados antes dos processos poderem executar

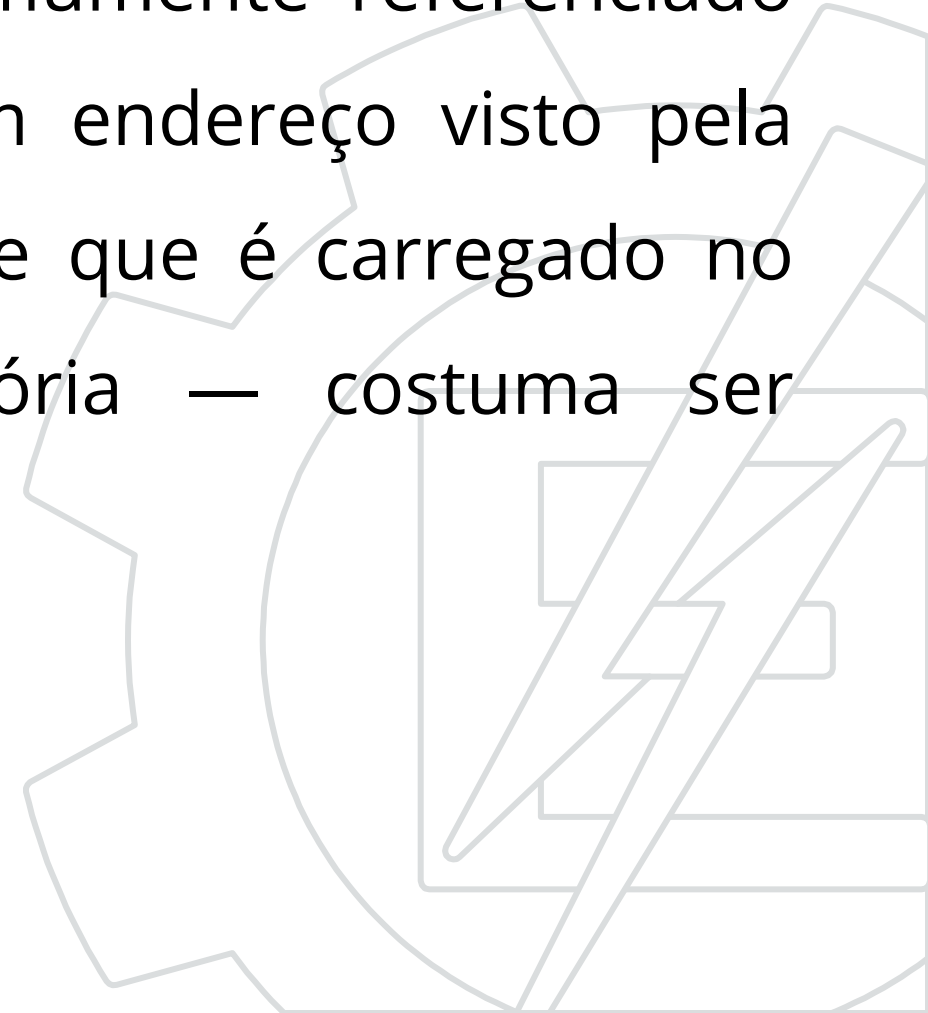
# Gerenciamento de Memória

## Multiprogramação

- Como fornecer a cada programa seu próprio espaço de endereçamento, de modo que o endereço #45 em um seja diferente do endereço #45 em outro?
- Solução: 2 registradores: base e limite



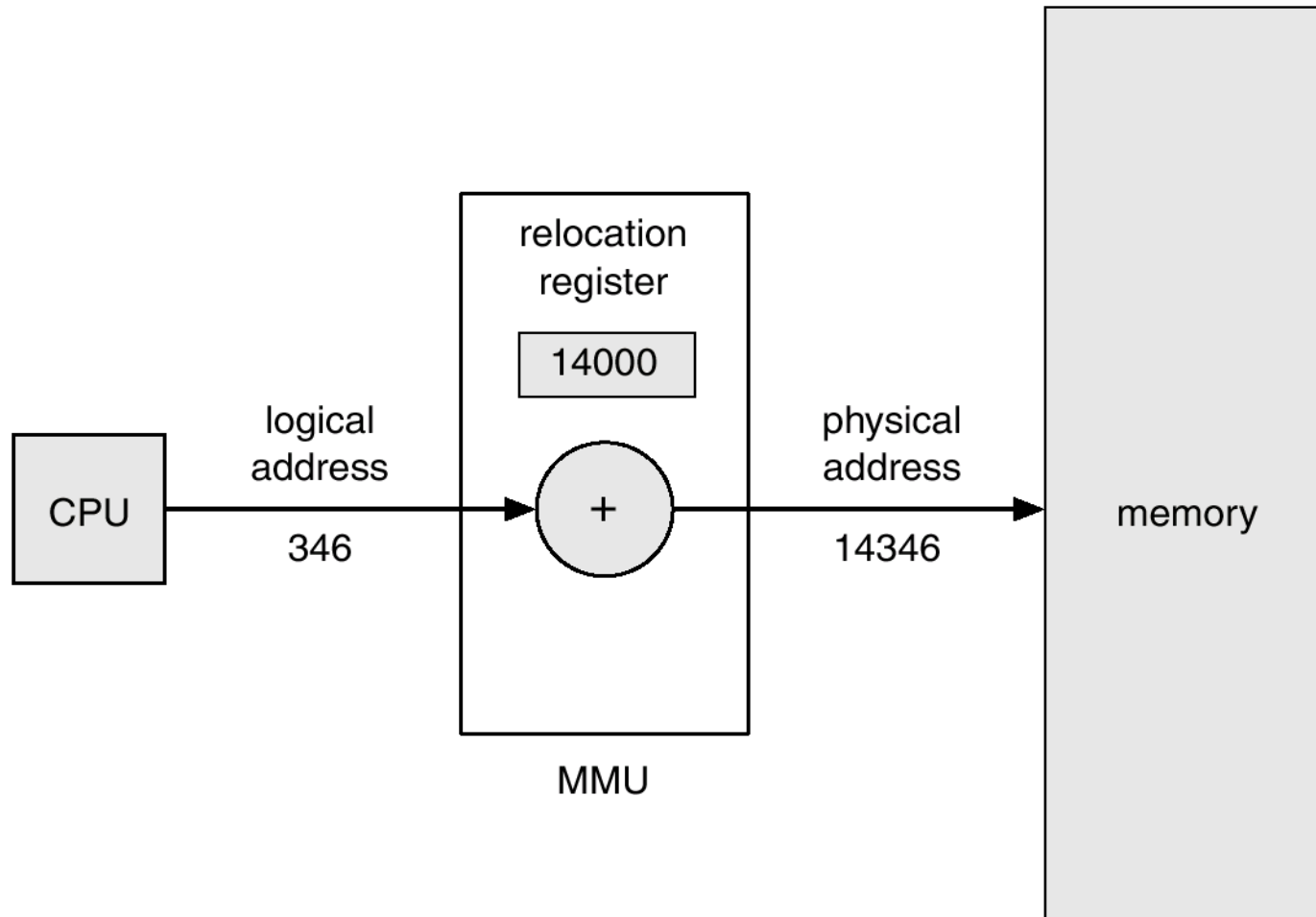
- Um endereço gerado pela CPU é comumente referenciado como **endereço lógico**, enquanto um endereço visto pela unidade de memória — isto é, aquele que é carregado no registrador de endereços da memória — costuma ser referenciado como **endereço físico**.



# Gerenciamento de Memória

Endereços Físicos *versus* Endereços Lógicos

- Endereço lógico: gerado pela CPU (virtual, posição relativa)
- Endereço físico: visto pela unidade de memória (real)





- Os métodos de vinculação de endereços em tempo de **compilação** e em tempo de **carga** geram endereços lógicos e físicos idênticos.
- O esquema de vinculação de endereços em tempo de **execução** resulta em endereços lógicos e físicos diferentes. Nesse caso, usualmente referenciamos o endereço lógico como um endereço virtual.

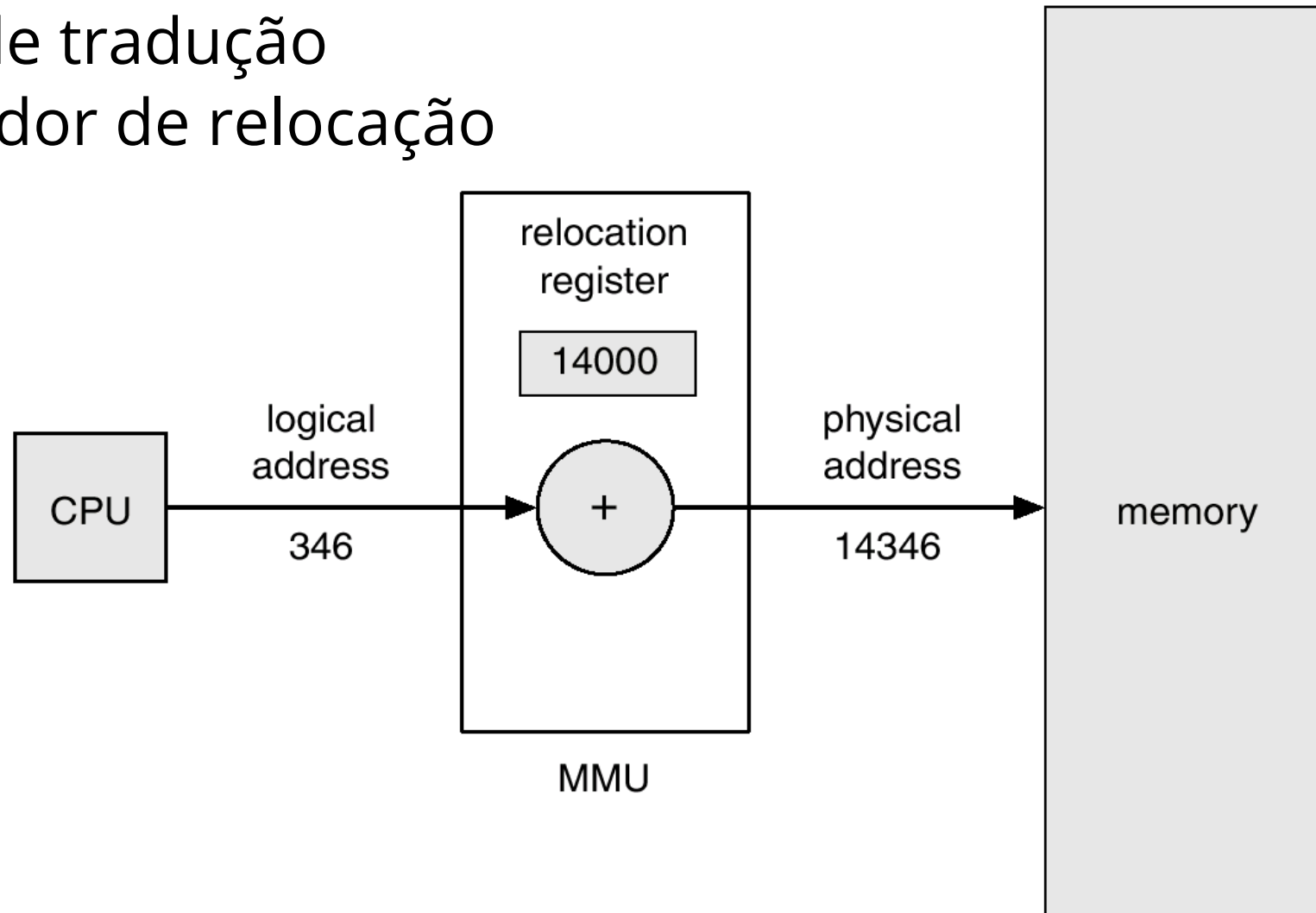
# Gerenciamento de Memória

Endereços Físicos *versus* Endereços Lógicos

- End. lógico: gerado pela CPU (virtual, posição relativa)
- End. físico: visto pela unidade de memória (real)
- Mapeamento depende de recursos do *hardware*
  - Unidade de gerência de memória (MMU)
  - Em sistemas simples (sem MMU), virtual=real



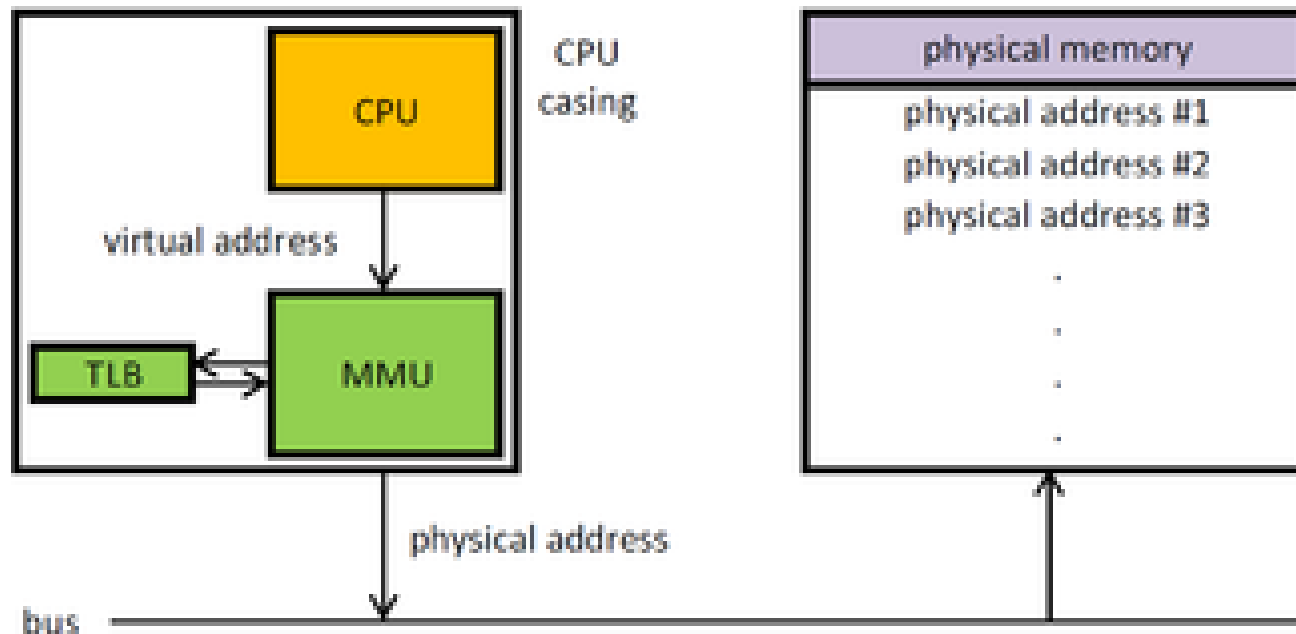
- Endereços lógicos são transformados
  - Tabela de tradução
  - Registrador de relocação



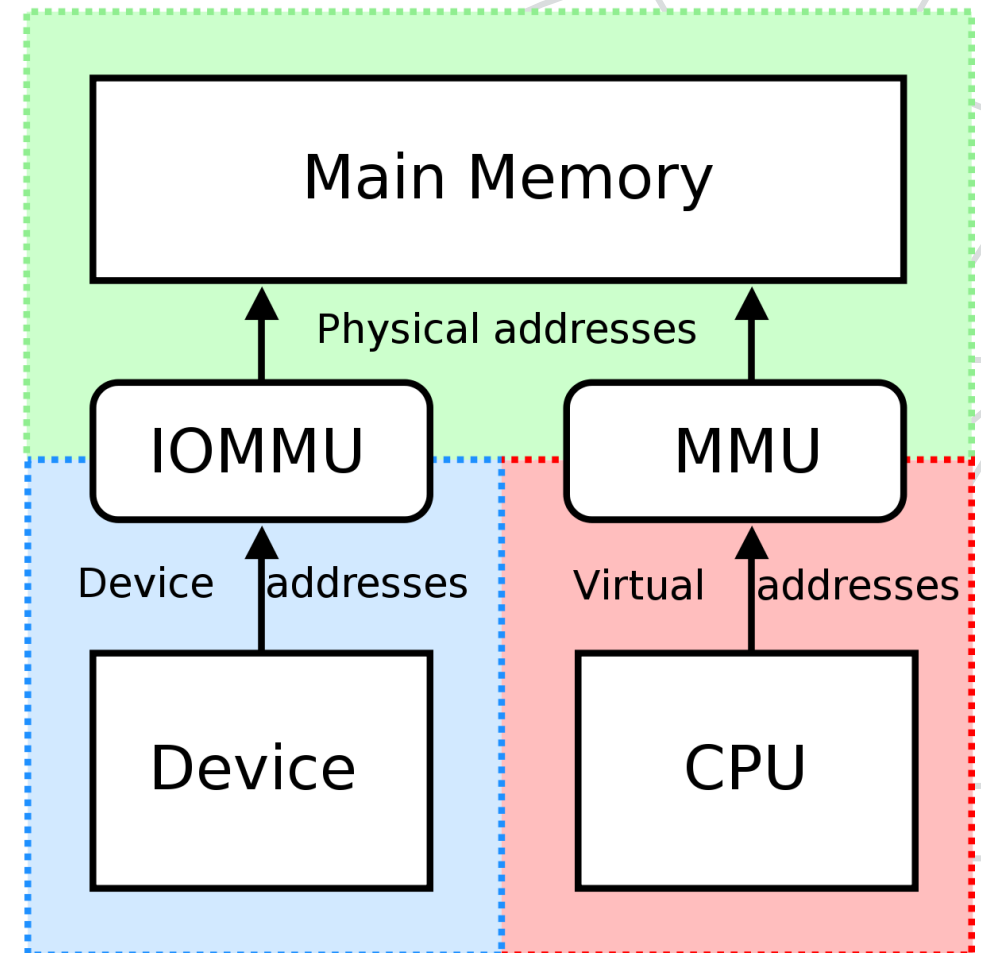
# Gerenciamento de Memória

## Multiprogramação

- **Memory Management Unit (MMU)**
  - Dispositivo (*hardware*) que transforma endereços virtuais/lógicos em endereços físicos.

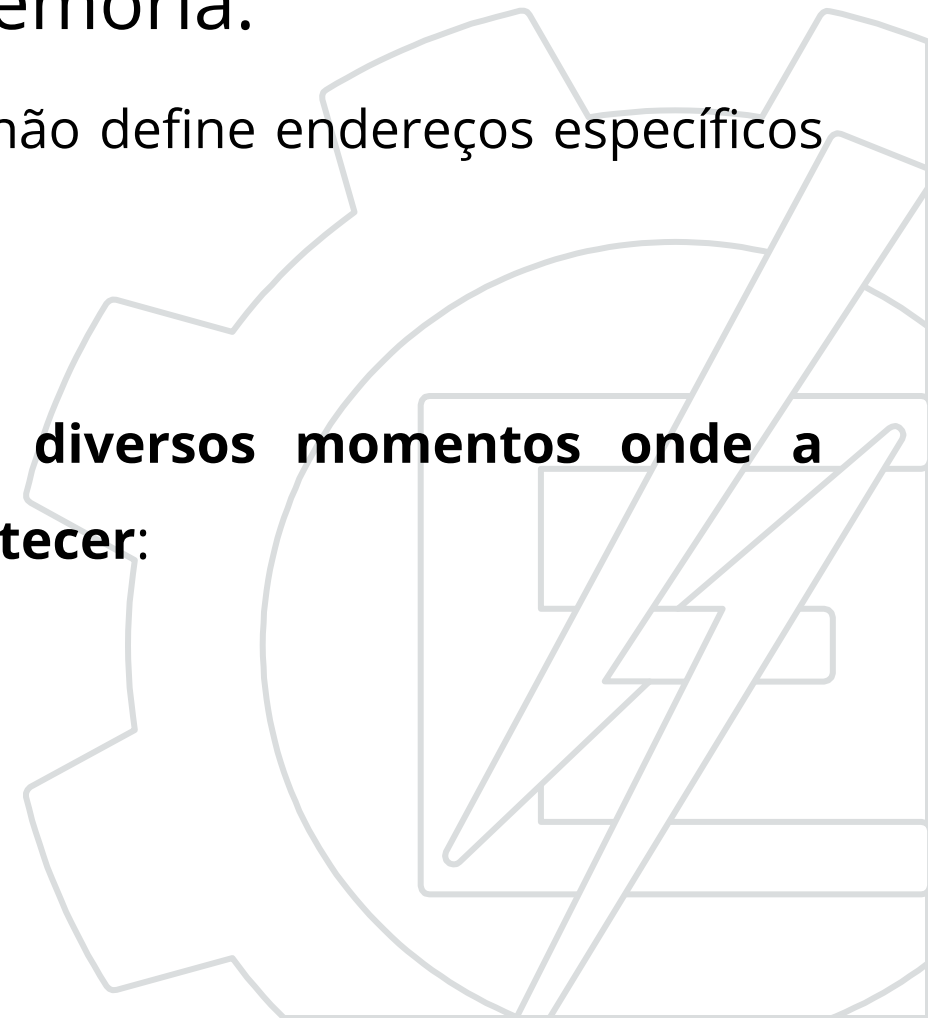


CPU: Central Processing Unit  
MMU: Memory Management Unit  
TLB: Translation lookaside buffer





- Programas em linguagem de alto nível **(quase) nunca referenciam endereços específicos** de memória:
  - Ex.: C define variáveis globais, locais e dinâmicas, mas não define endereços específicos para elas
- Dependendo da linguagem, do S.O. e do HW, **há diversos momentos onde a associação** entre os comandos e a memória **pode acontecer**:
  - Tempo de compilação
  - Tempo de carga
  - Tempo de execução



- **Tempo de compilação:**

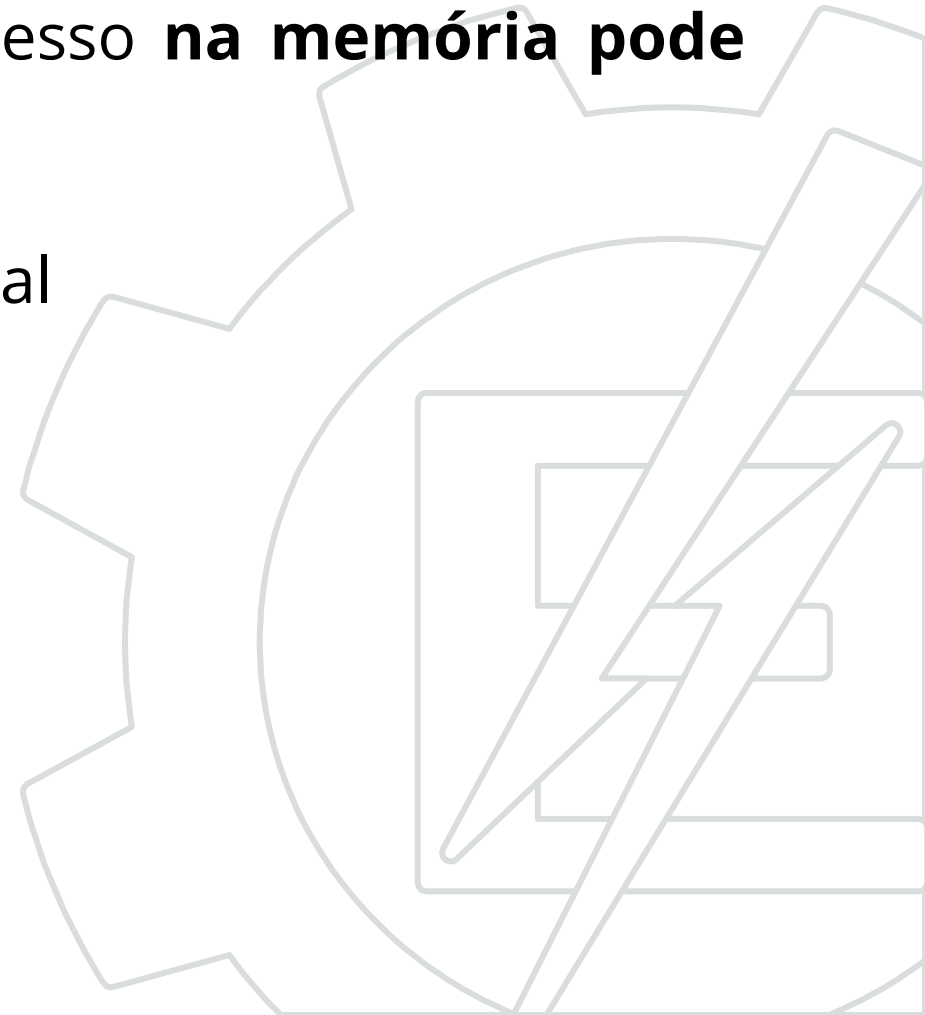
- Se houver uma definição fixa sobre localização **o compilador pode gerar endereços diretamente**
- Se for preciso mudar os endereços, é preciso recompilar o programa
- Comum para elementos com endereços definidos pelo HW (ex., vetor de interrupções)
- Ocorre também na vinculação de símbolos definidos e acessados em módulos diferentes de um programa

- **Tempo de carga:**

- Compilador gera anotações sobre acessos relacionados a certos endereços
- **O carregador (*loader*) lê essas anotações e altera o código** do programa carregado para inserir os endereços adequados a cada caso
- Endereços ganham valor **a partir do momento que sabe-se a posição inicial** do programa

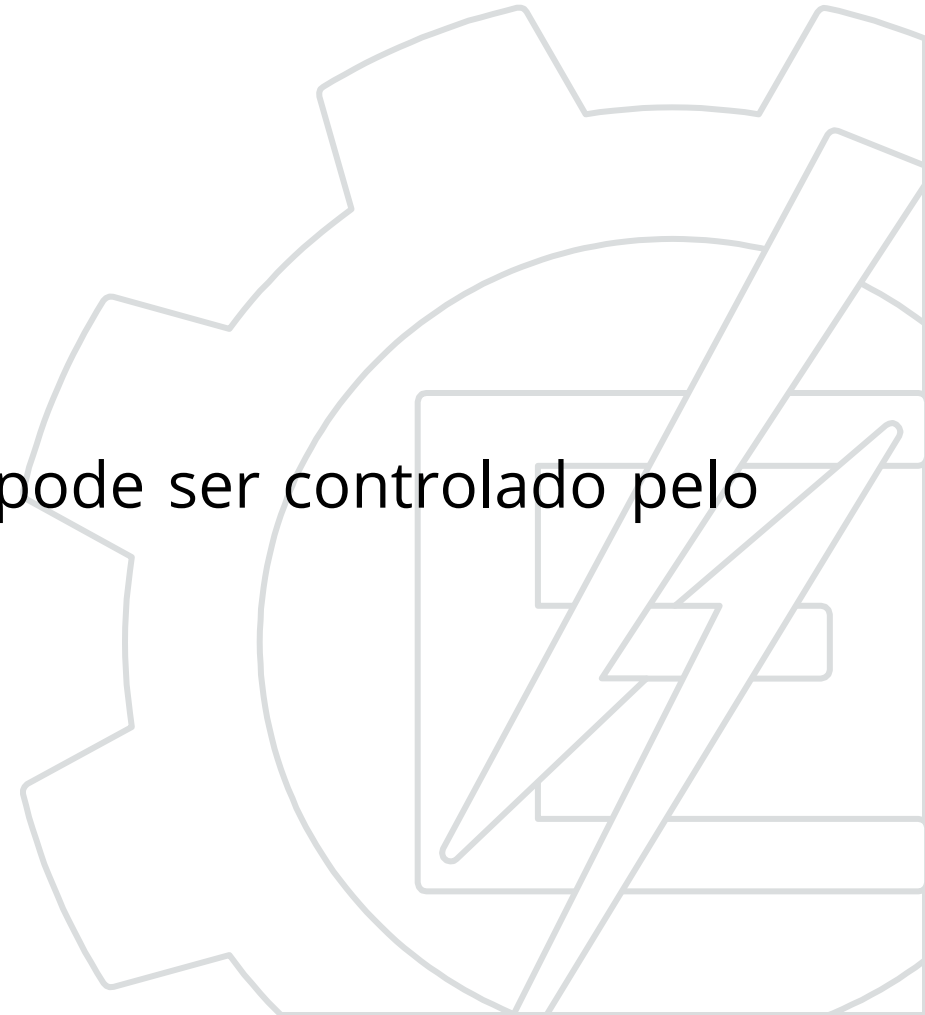
- **Tempo de execução:**

- Em alguns casos **a localização** de um processo **na memória pode mudar** em tempo de execução
- Nesse caso o HW deve prover suporte especial
- **É preciso refazer vínculos eficientemente**
  - Mapas de endereços
  - Tabelas de relação
  - Endereçamento relativo

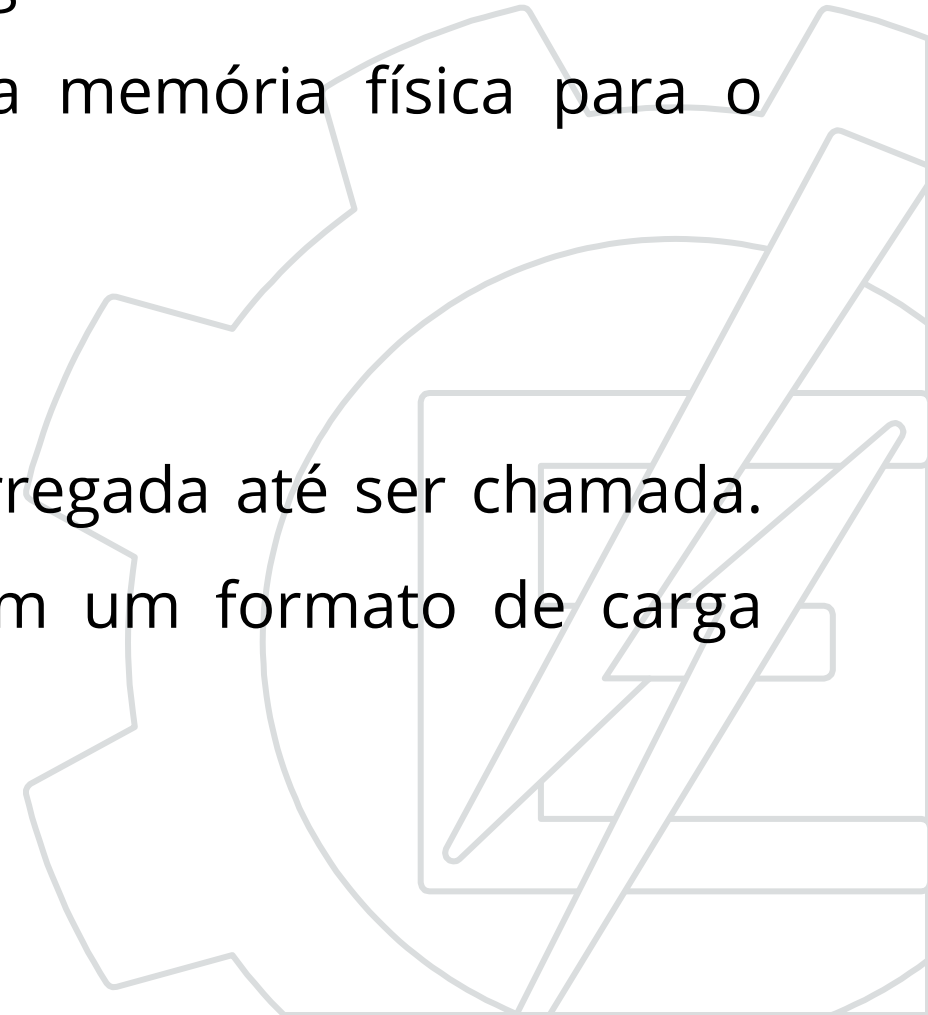




- Permite que partes de um programa só sejam carregados na memória quando chamados
  - Melhor utilização da memória
  - Partes não utilizadas podem ser descarregadas
- Não necessariamente exige suporte do S.O., pode ser controlado pelo programa (complexo)
  - Exemplo: *overlays* no DOS

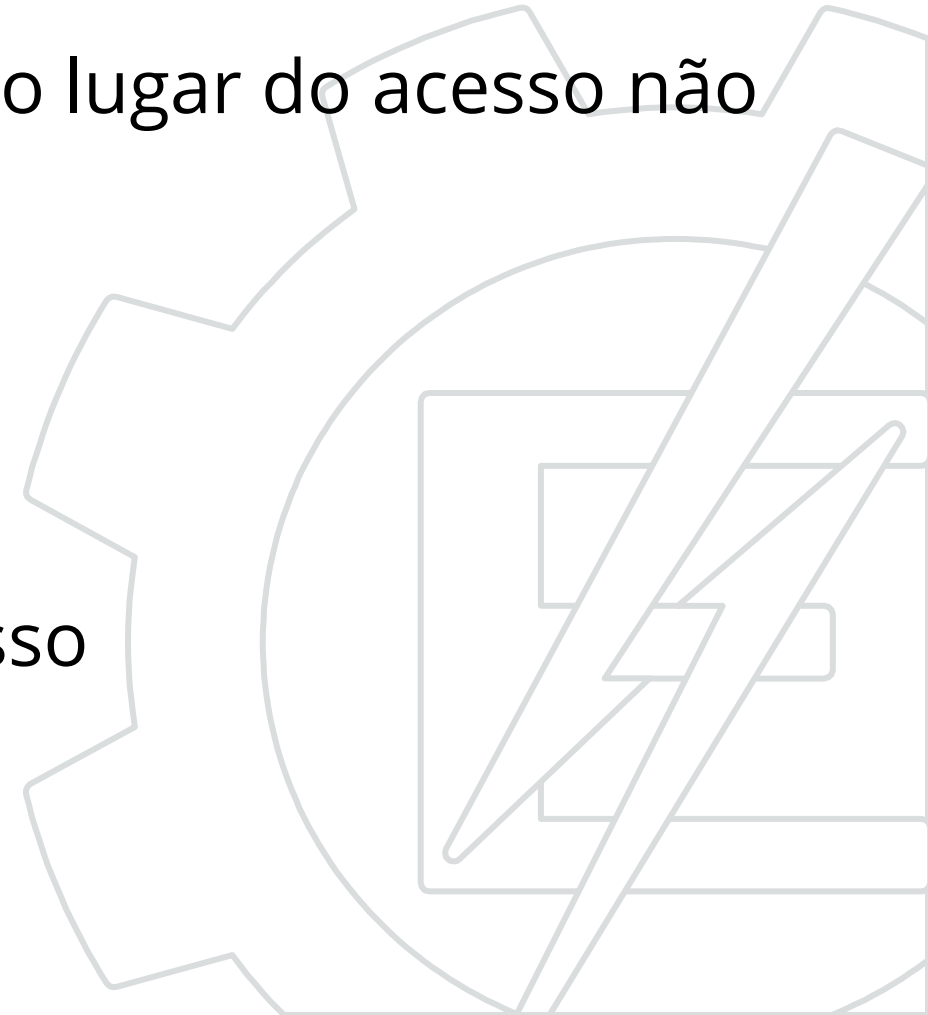


- Em nossa discussão até o momento, o programa inteiro e todos os dados de um processo tinham de estar na memória física para o processo ser executado.
- Com a carga dinâmica, uma rotina não é carregada até ser chamada. Todas as rotinas são mantidas em disco em um formato de carga relocável.



- O programa principal é carregado na memória e executado. Quando uma rotina precisa chamar outra rotina, a rotina chamadora verifica, primeiro, se a outra rotina foi carregada.
- Se não o foi, o carregador de vinculação relocável é chamado para carregar a rotina desejada na memória e atualizar as tabelas de endereços do programa para que reflitam essa alteração.
- Em seguida, o controle é passado à rotina recém-carregada.

- Vinculação de endereços postergada até a execução.
- Pequeno trecho de código (*stub*) ocupa o lugar do acesso não vinculado
  - Ao ser executado, localiza a rotina correta
  - Substitui a si mesmo com o código correto
- Suporte do S.O. para controlar o processo
- Particularmente útil para bibliotecas



# *Swapping*

Permuta de processos

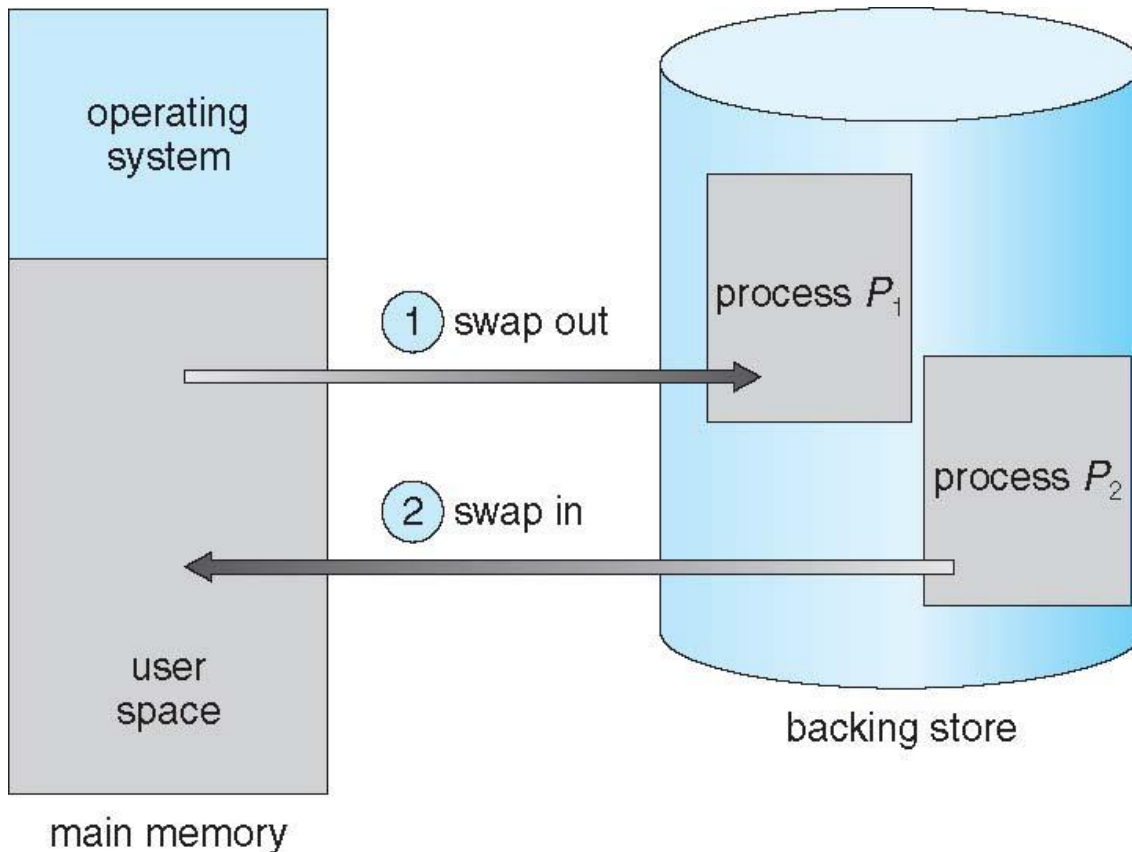


# Gerenciamento de Memória

## Swapping – Permuta de processos

É o chaveamento de processos entre a memória e o disco:

- *Swap-out*: da memória para o disco, para uma área de *swap*.
- *Swap-in*: do disco para a memória.



### Swapping

- Processos inteiros podem ser transferidos temporariamente para memória secundária e posteriormente trazidos de volta
  - ex.: quando são suspensos pelo escalonador
- Requer uma “memória de retaguarda” (*backing store*) grande o suficiente
- Maior *overhead* é o tempo de transferência
- Presente em diversos S.O. (Unix, Linux, Win...)

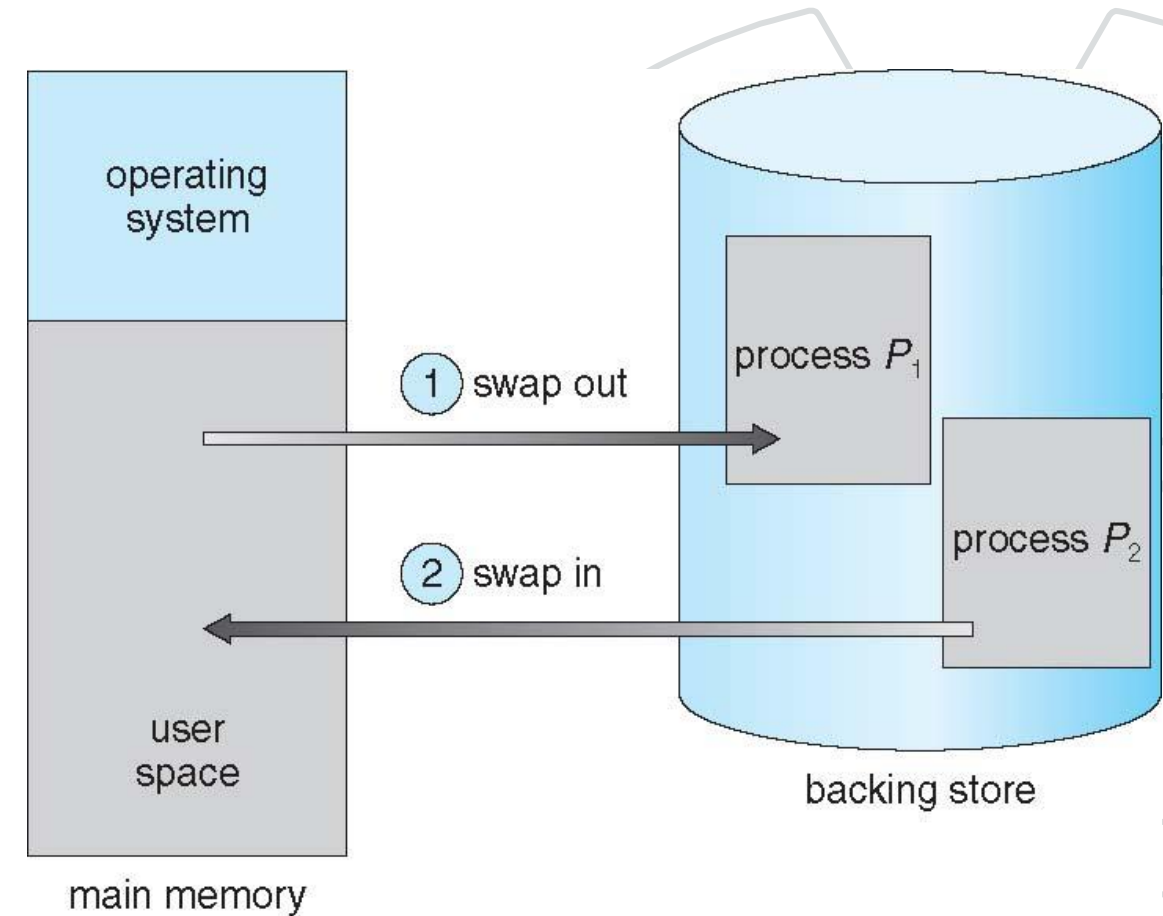
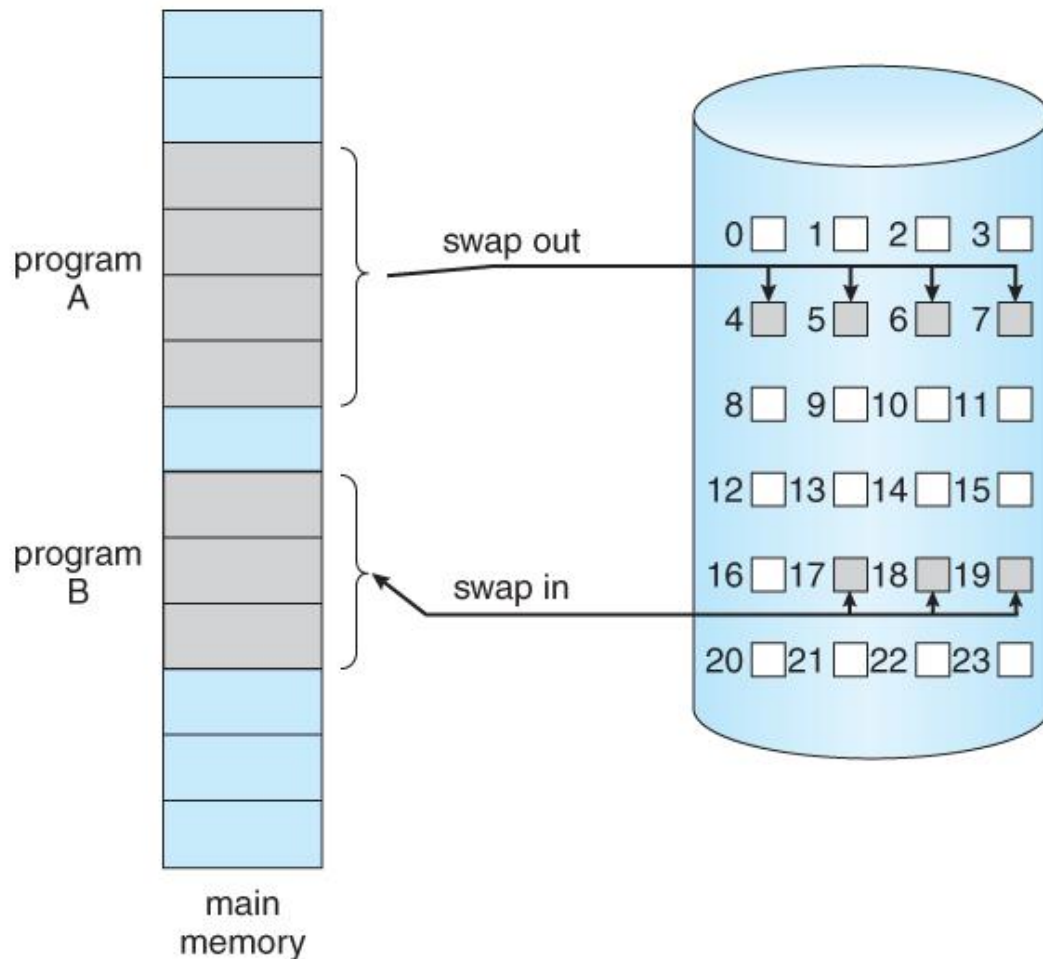


# Gerenciamento de Memória

## Swapping

É o chaveamento de processos entre a memória e o disco.

- Pode gerar fragmentação interna e externa.



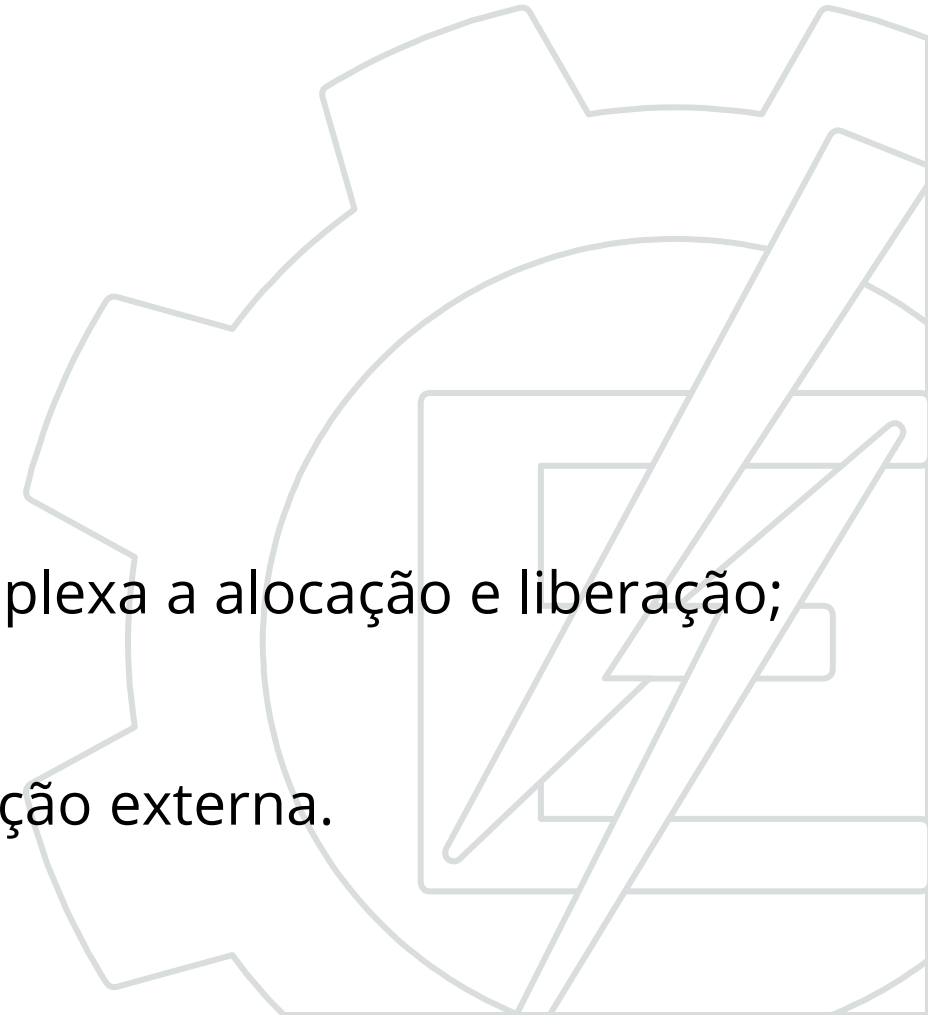
- **Permuta Padrão** - envolve a transferência de processos entre a memória principal e uma memória de retaguarda.
  - A memória de retaguarda é comumente um disco veloz. Ela deve ser suficientemente grande para acomodar cópias de todas as imagens da memória para todos os usuários, e deve fornecer acesso direto a essas imagens da memória.
  - Sempre que o *scheduler* da CPU decide executar um processo, ele chama o despachante. O despachante verifica se o próximo processo na fila está em memória. Caso não esteja, e se não houver uma região de memória livre, o despachante remove um processo correntemente em memória e o permuta com o processo desejado. Em seguida, ele recarrega os registradores e transfere o controle ao processo selecionado.

- **Permuta em Sistemas Móveis** - os sistemas móveis normalmente não suportam forma alguma de permuta.
  - Os dispositivos móveis costumam usar memória *flash*, em vez dos discos rígidos mais volumosos, como seu espaço de armazenamento persistente. A restrição de espaço resultante é uma razão para os projetistas de sistemas operacionais móveis evitarem a permuta.
  - Outras razões incluem o número limitado de gravações que a memória flash pode tolerar antes de se tornar não confiável e o fraco *throughput* entre a memória principal e a memória *flash* nesses dispositivos
  - Tanto o iOS quanto o Android dão suporte à paginação; sendo assim, eles têm recursos de gerenciamento da memória.

Partições



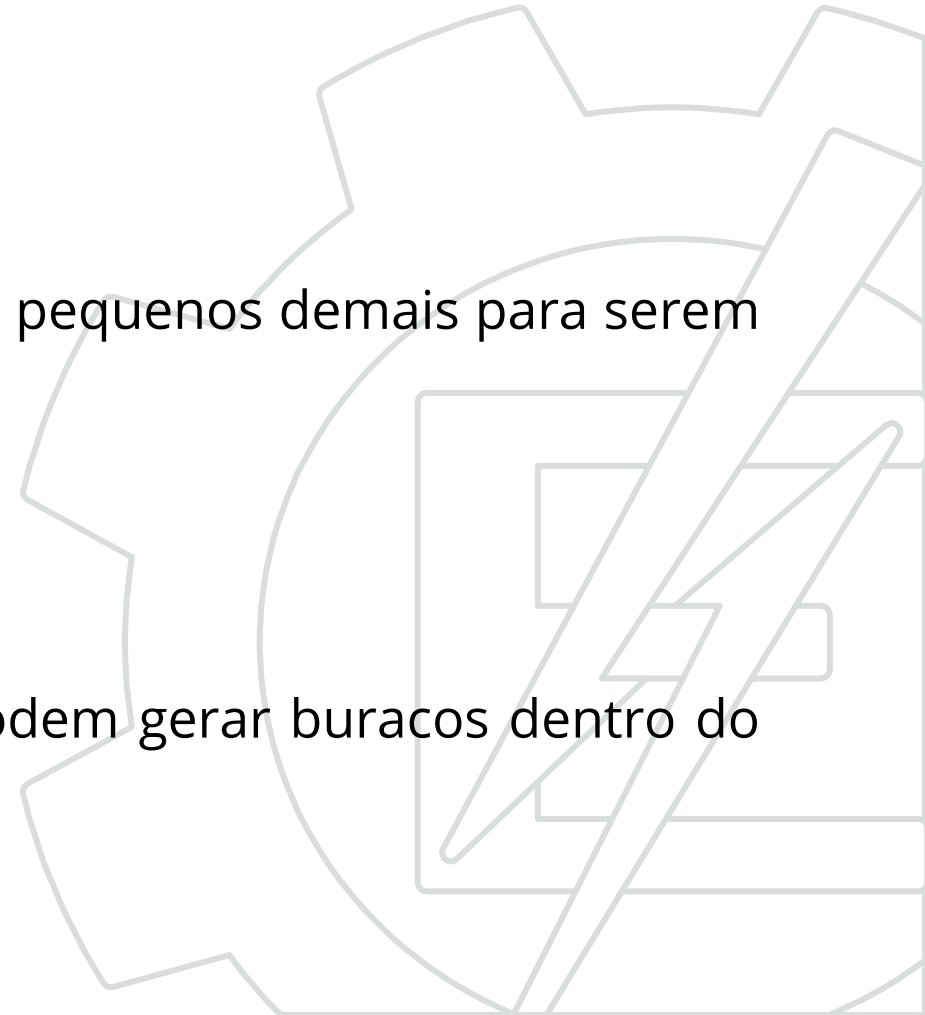
- **Partições fixas** (ou alocação estática):
  - Tamanho e número de partições são fixos (estáticos);
  - Tendem a desperdiçar memória;
  - Mais simples.
- **Partições variáveis** (ou alocação dinâmica):
  - Tamanho e número de partições variam;
  - Otimiza a utilização da memória, mas torna complexa a alocação e liberação;
  - Partições são alocadas dinamicamente.
  - Menor fragmentação interna e maior fragmentação externa.



# Gerenciamento de Memória

## Fragmentação externa e interna

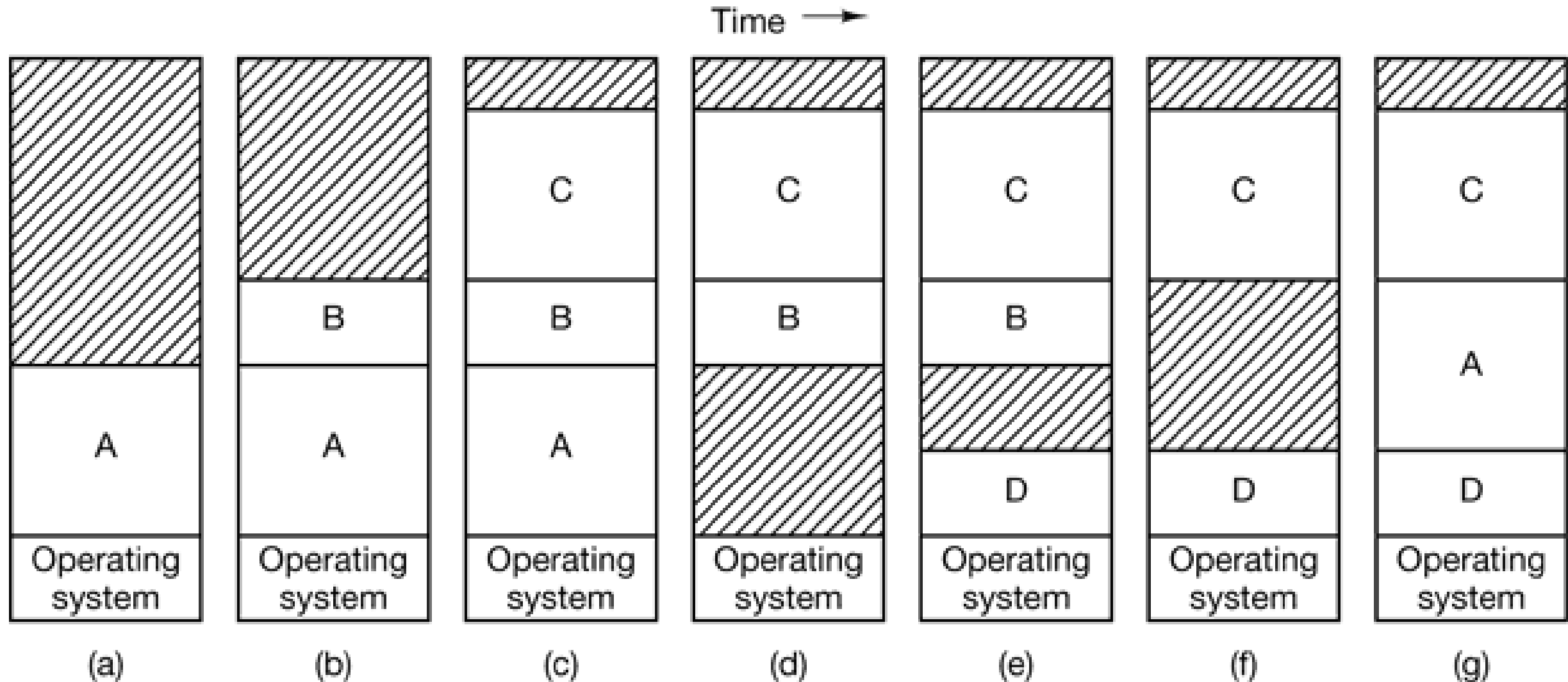
- Fragmentação é a quebra do espaço em frações não utilizáveis.
- Fragmentação **externa**:
  - Memória não utilizada dividida em muitos buracos, pequenos demais para serem úteis
- Fragmentação **interna**:
  - Limitações na forma como blocos são alocados podem gerar buracos dentro do bloco



# Gerenciamento de Memória

## Partições

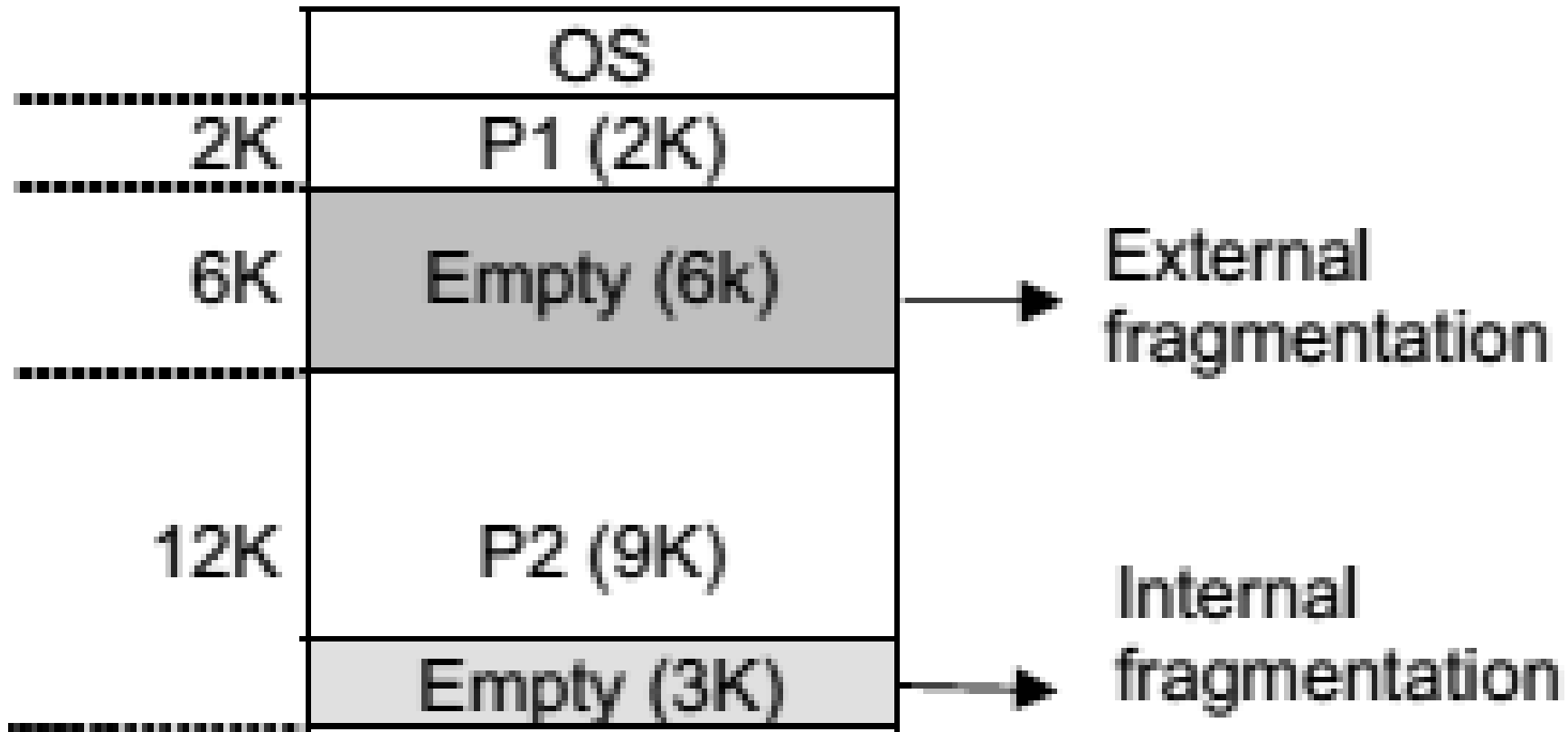
**Partições variáveis** e fragmentação externa:





# Gerenciamento de Memória

Fragmentação externa e interna

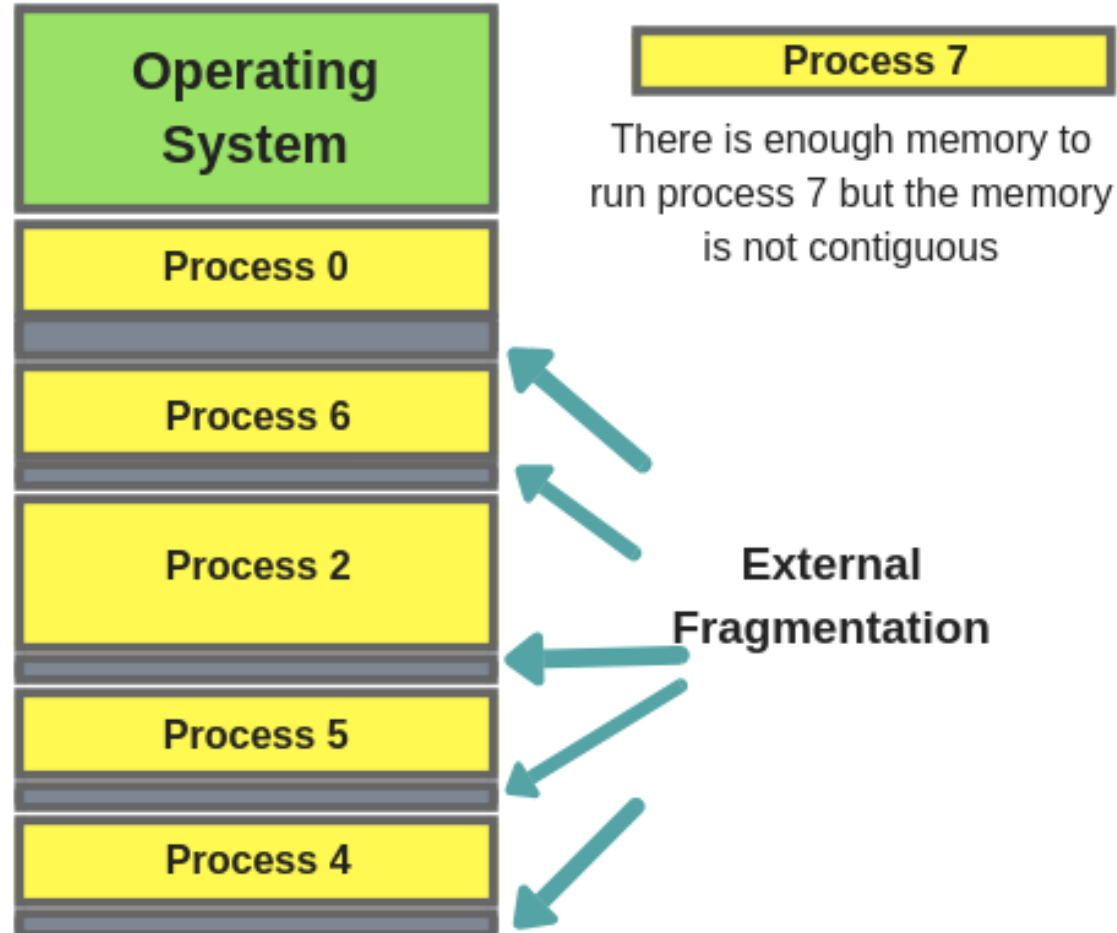


# Gerenciamento de Memória

Fragmentação externa e interna



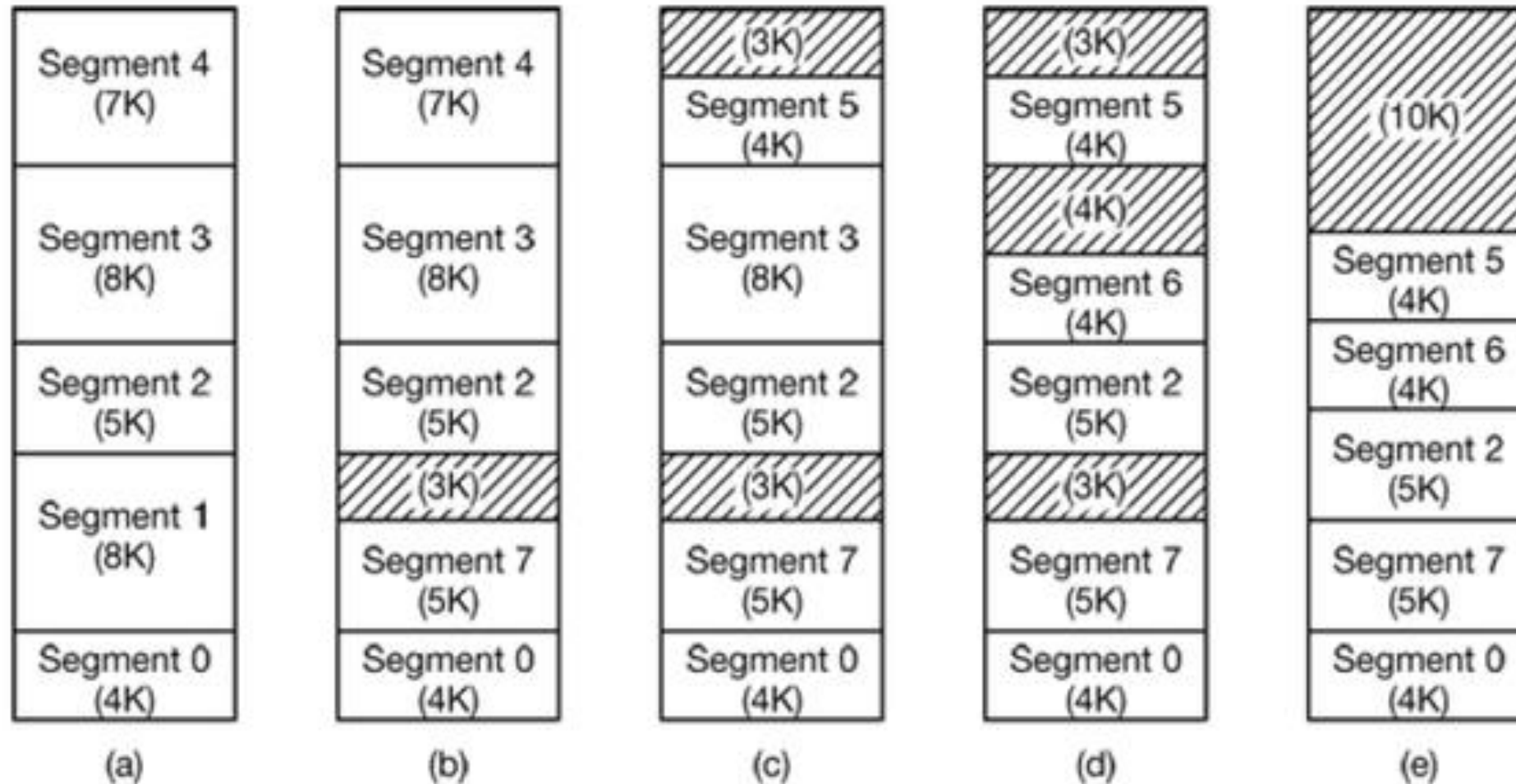
## External Fragmentation



## Internal Fragmentation

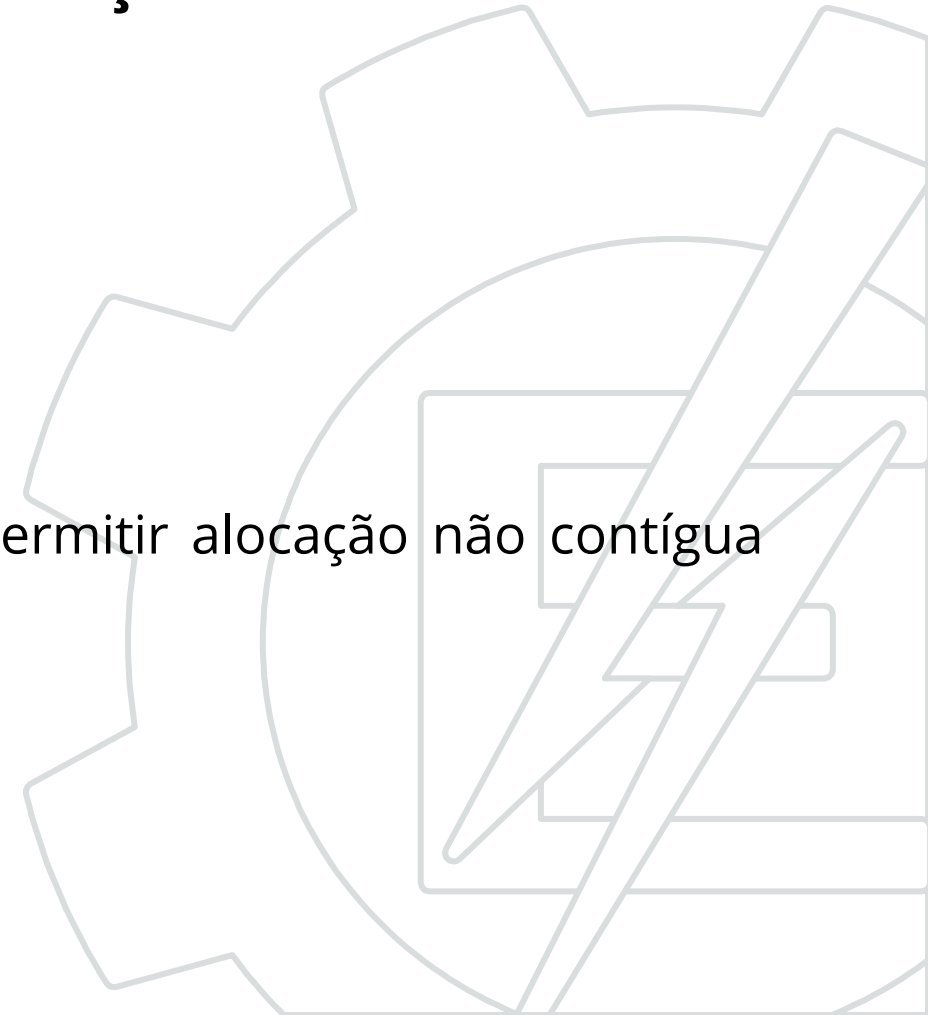


## Fragmentação externa:

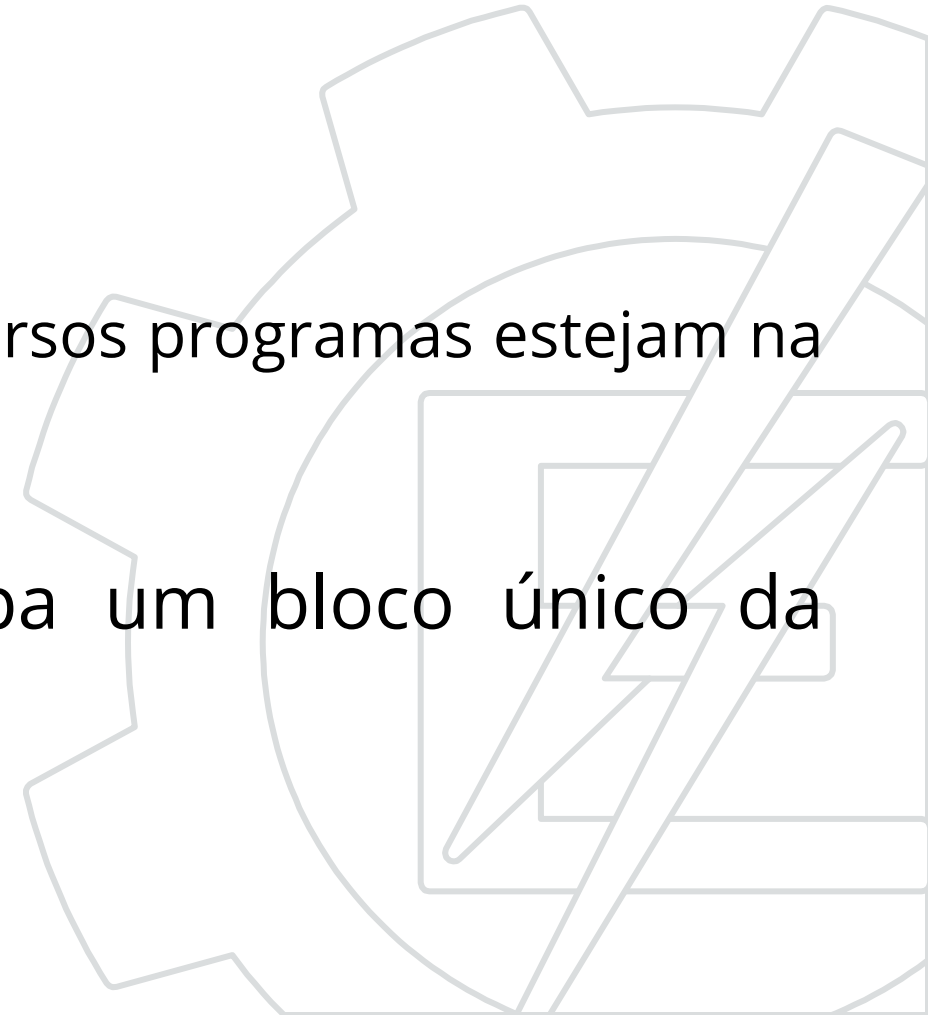


**Figure 4-38.** (a)-(d) Development of checkerboarding. (e) Removal of the checkerboarding by compaction.

- Solução para a fragmentação externa: **Compactação**
  - Mover blocos ocupados para perto uns dos outros
  - Agrupar os buracos em um único bloco maior
  - **Só é possível com relocação dinâmica**
- Outra forma de lidar com fragmentação externa é permitir alocação não contígua (**paginação** ou **segmentação**)



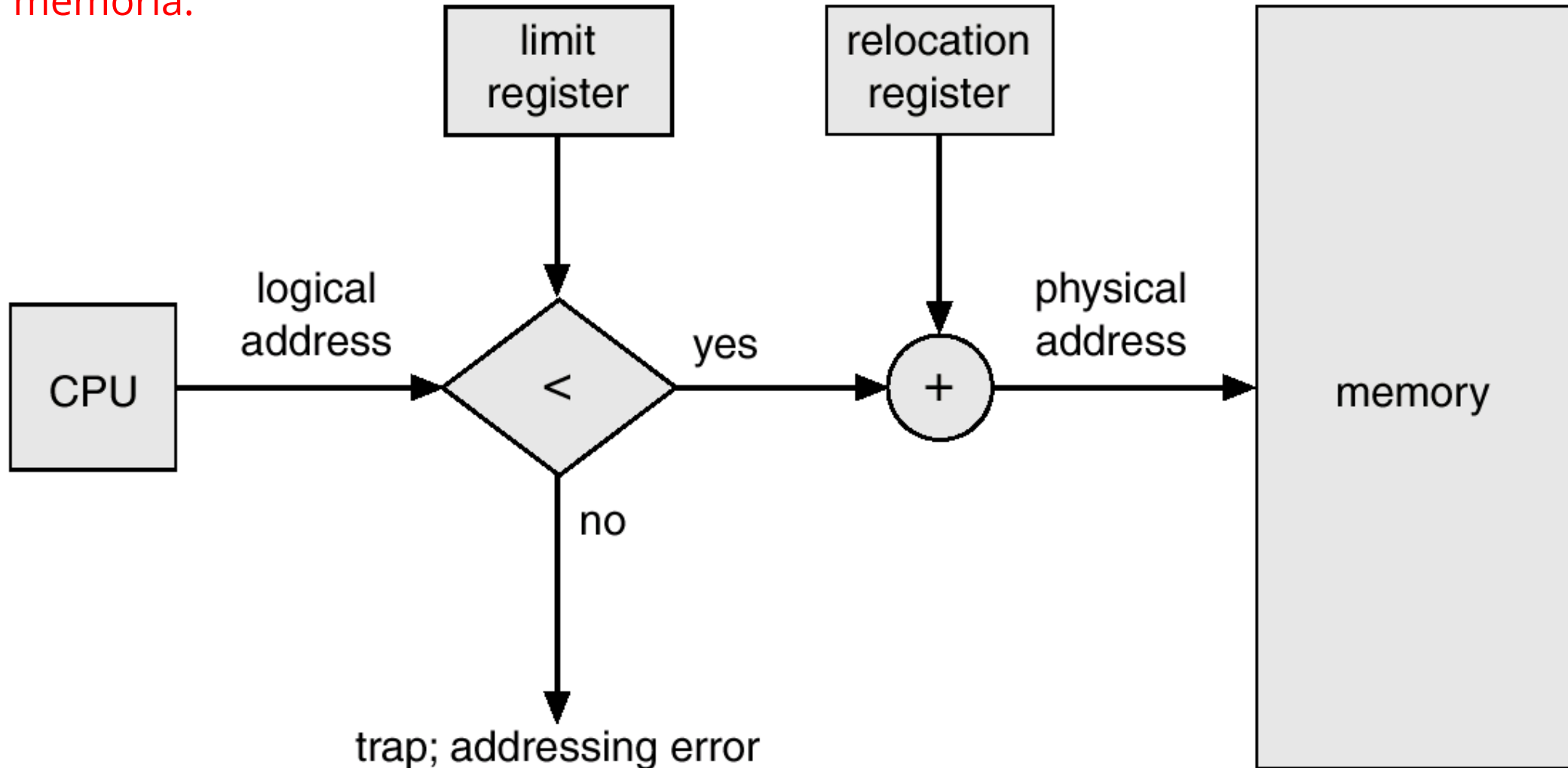
- Forma simples de utilização da memória
  - Duas partições:
    - S.O. residente (memória baixa, vetor interrupções)
    - Processo de usuário
  - Para multiprogramação, é desejável que diversos programas estejam na memória
- Alocação contígua: cada processo ocupa um bloco único da memória física



# Gerenciamento de Memória

Alocação contígua

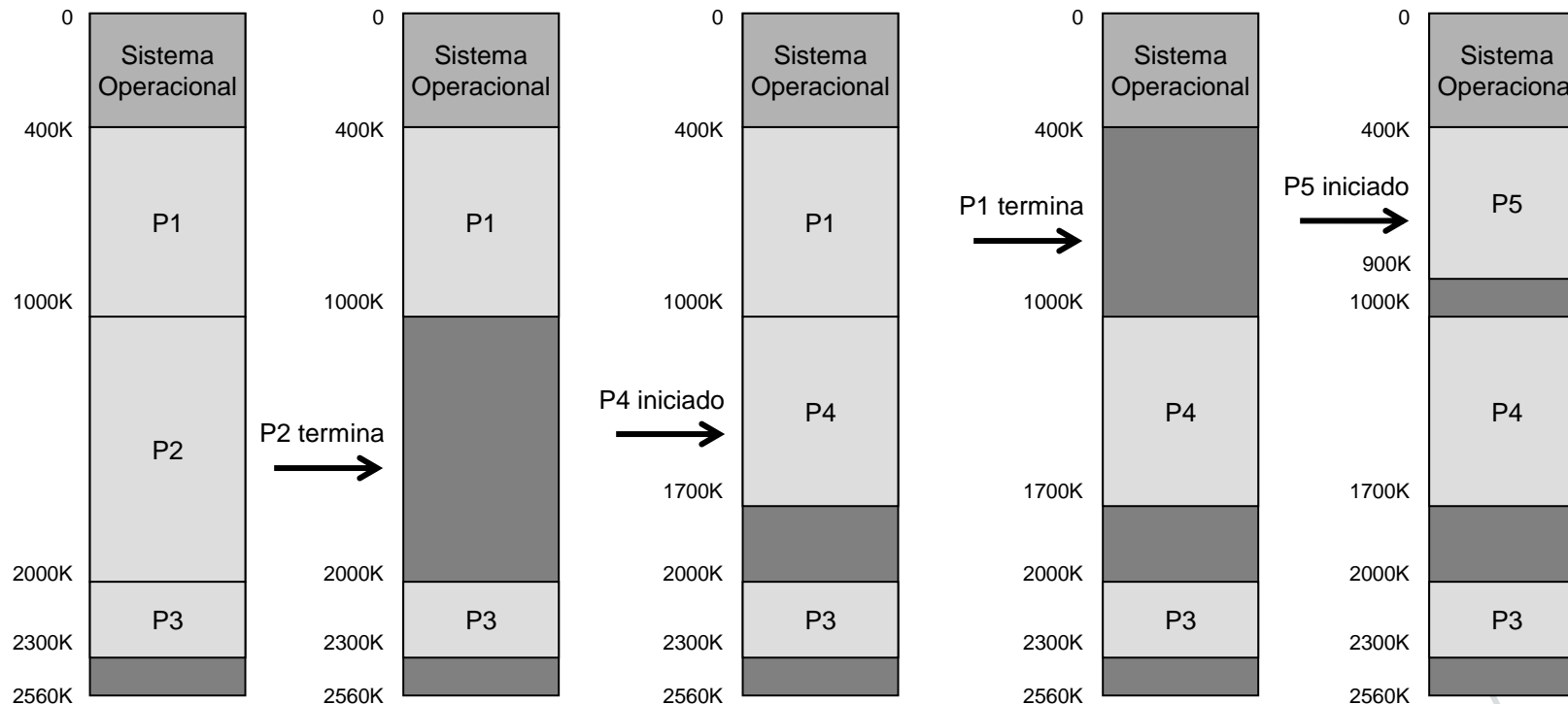
- É essencial que seja possível fazer relocação dinâmica e **proteção entre as áreas de memória:**



# Gerenciamento de Memória

## Alocação contígua com múltiplos processos

- Cada processo ocupa um bloco da memória
- Ao terminar, bloco é liberado (buraco)
- Processos que chegam ocupam buracos
- S.O. deve controlar partições e buracos
- Relocação por tabela ou registrador base





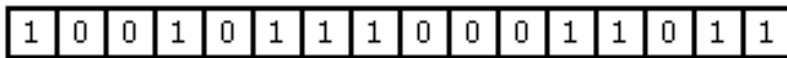
# Gerenciamento de espaço



- **Mapa de bits (*Bitmaps*)**

- A memória é dividida em unidades de alocação, onde uma unidade pode conter vários KB.
- Cada unidade corresponde a um bit no bitmap: 0 equivale a livre; 1 equivale a ocupado.

Bit Map:



Memory:



16 bits handles 16K of memory with the chunk (page) size of 1K

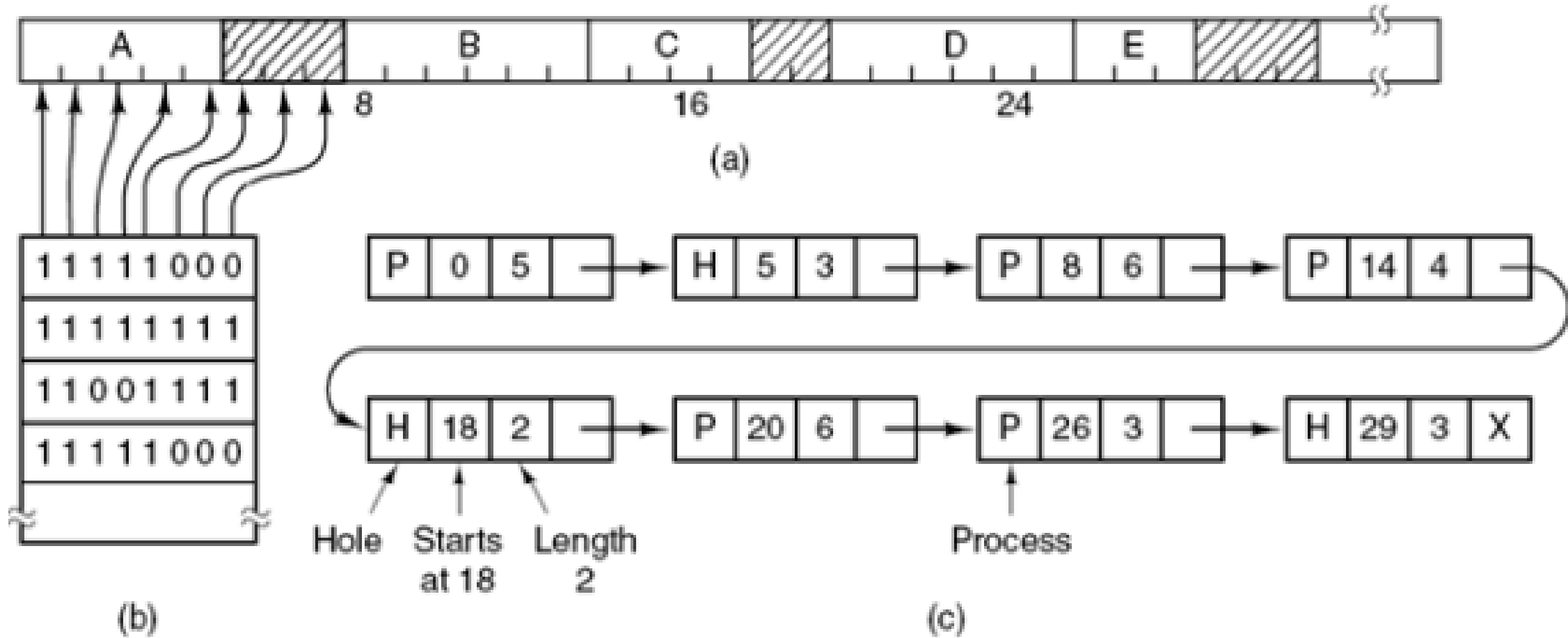
- **Lista Encadeada**

- É mantida uma lista encadeada de segmentos de memória livres e alocados.

# Gerenciamento de Memória

Estruturas para gerenciamento de espaço

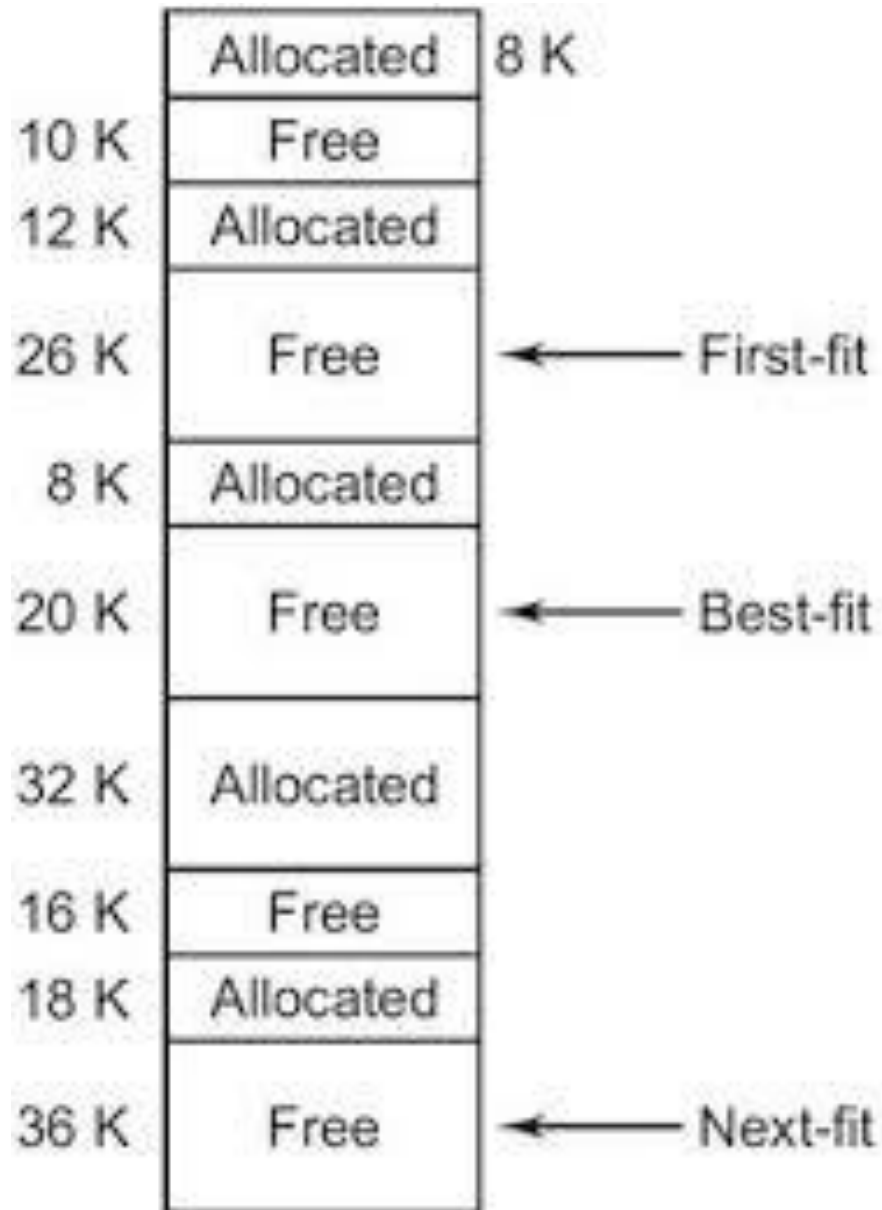
## Mapa de bits (*Bitmaps*) e Lista encadeada



(a) Estrutura da Memória; (b) Mapa de Bits; (c) Lista Encadeada.

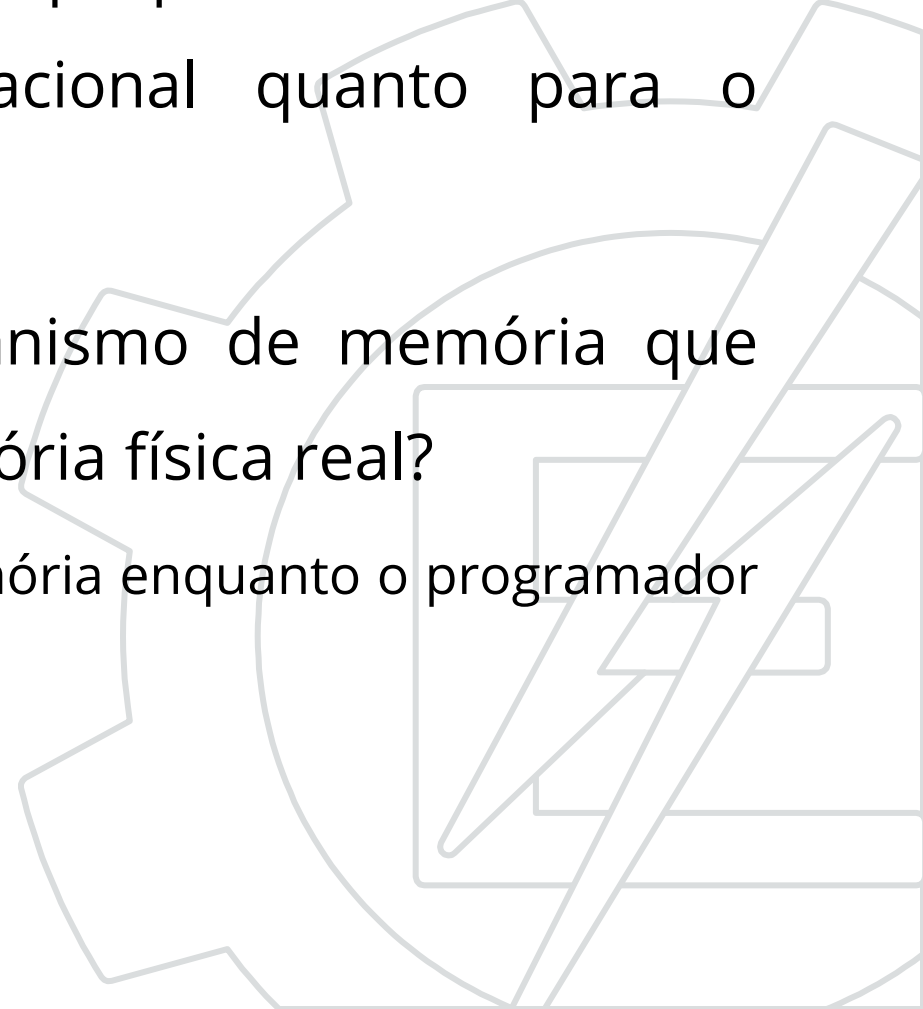
# Gerenciamento de Memória

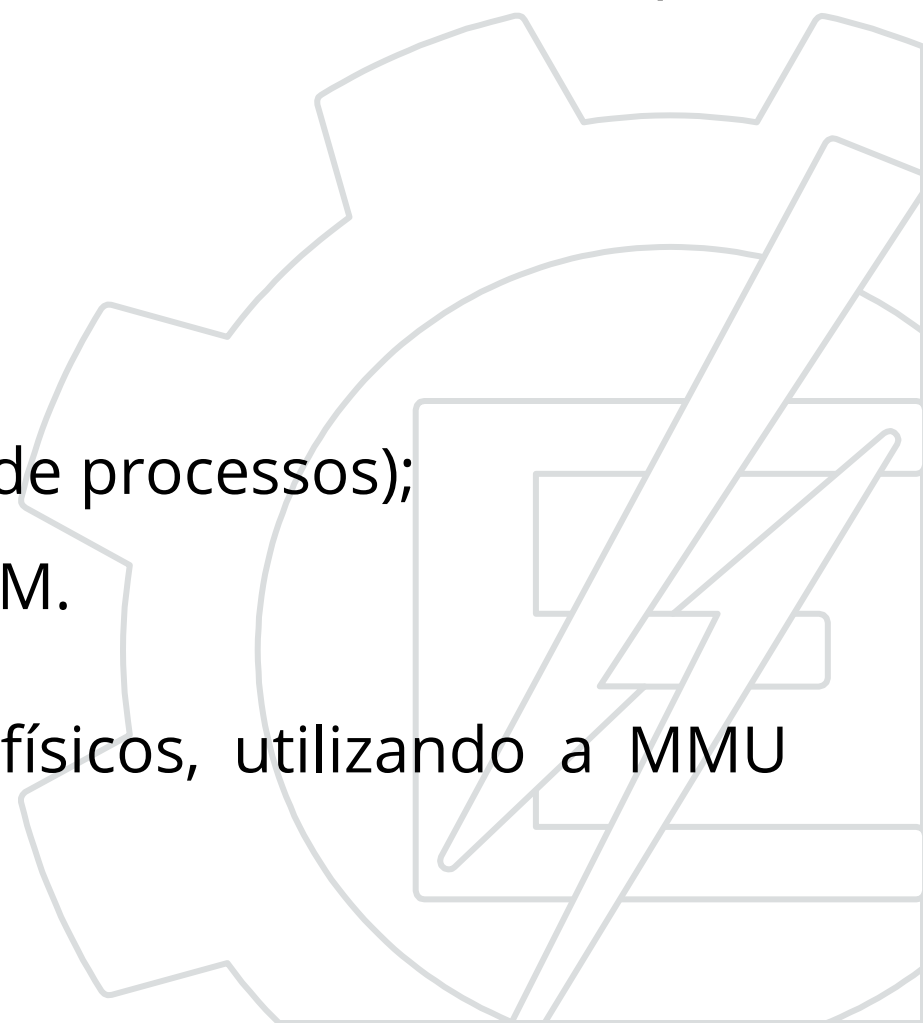
## Algoritmos de alocação



- **Primeira escolha (*first-fit*)**
  - Percorre a lista até que encontre o primeiro *slot* em que caiba a informação.
- **Melhor escolha (*best-fit*)**
  - Busca a lista inteira e toma a menor partição.
- **Pior escolha (*worst-it*)**
  - Busca a lista inteira para a maior partição.
- **Próxima escolha (*next-fit*)**
  - Similar ao primeira escolha,. Percorre a lista a partir do último valor inserido.

# Gerenciamento de Memória

- Lidar com a memória em termos de suas propriedades físicas é inconveniente tanto para o sistema operacional quanto para o programador.
  - E se o *hardware* pudesse fornecer um mecanismo de memória que mapeasse a visão do programador para a memória física real?
    - O sistema teria mais liberdade para gerenciar a memória enquanto o programador teria um ambiente de programação mais natural.
    - A segmentação fornece esse mecanismo.
- 

- O que é Memória Virtual?
    - É uma técnica que utiliza a memória secundária como uma “*cache*” para partes do espaço dos processos.
  - Por que utilizar Memória Virtual?
    - Tamanho dos programas cada vez maior;
    - Maior grau de multiprogramação (quantidade de processos);
    - Permite executar programas maiores que a RAM.
  - Um processo utiliza endereços virtuais e não físicos, utilizando a MMU para a conversão destes endereços.
- 

- **Paginação**

- Blocos de tamanho fixo (em torno de 4KB);
- O espaço de endereçamento virtual é dividido em páginas virtuais.

- **Segmentação**

- Blocos de tamanho arbitrário chamados segmentos;
- Contém o mesmo tipo de informação (ex.: dados, pilha)

- Benefícios do uso de Memória Virtual:

- Menos operações de E/S → reduz *swap* de programas
- Menor área de memória física ocupada → mais programas
- Mais processos/usuários atendidos → melhor utilização da CPU





# Memória Virtual

Physical  
Memory  
Frames

Page Table

Virtual  
Memory  
Pages

	frame	page	m	v
0	0		0	0
1	1		0	0
2	2	1	0	0
3	3		0	0
4	4	0	0	1
5	5		0	0
6	6	3	1	0
7	7		0	0

DS

E000

SS

0000

CS

7000

ES

52B9

data segment

code segment

extra segment

stack segment

FFFFF

FFFFF

E0000

7FFFF

70000

62B8F

52B90

0FFFF

00000

# Segmentação



# Gerenciamento de Memória

## Segmentação

- A segmentação é um esquema de gerenciamento de memória que dá suporte à visão da memória desse programador. Um espaço de endereçamento lógico é um conjunto de segmentos.
- Cada segmento tem um **nome** e um **tamanho**. Os endereços especificam tanto o nome do segmento quanto o deslocamento dentro do segmento. O programador, então, especifica cada endereço com dois valores: um **nome** de segmento e um **deslocamento**.

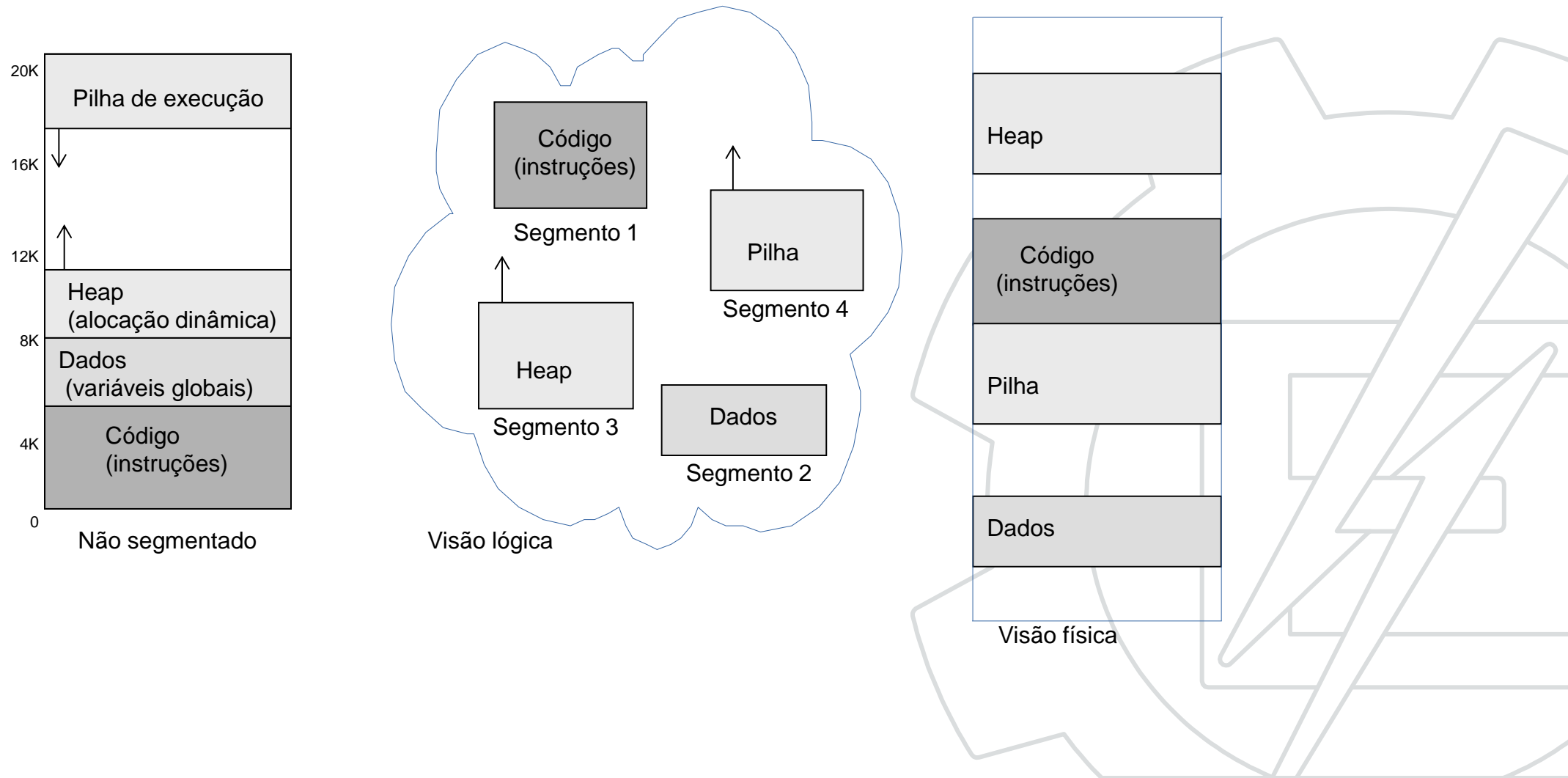
- Endereço lógico é uma dupla: <segm.,offset>
- Tabela de segmentos mapeia segmento em uma área da memória (base + limite)
- Segmentos: pré-definidos (fixos) ou enumeráveis
  - Fixos: *Stack Segment*, *Code Segment*, ... (Intel)
  - Enumeráveis: tabela de segmentos (com limite)
- Segmentos podem ser compartilhados
- Permite controle de acesso refinado
  - ex., código != pilha != dados



# Gerenciamento de Memória

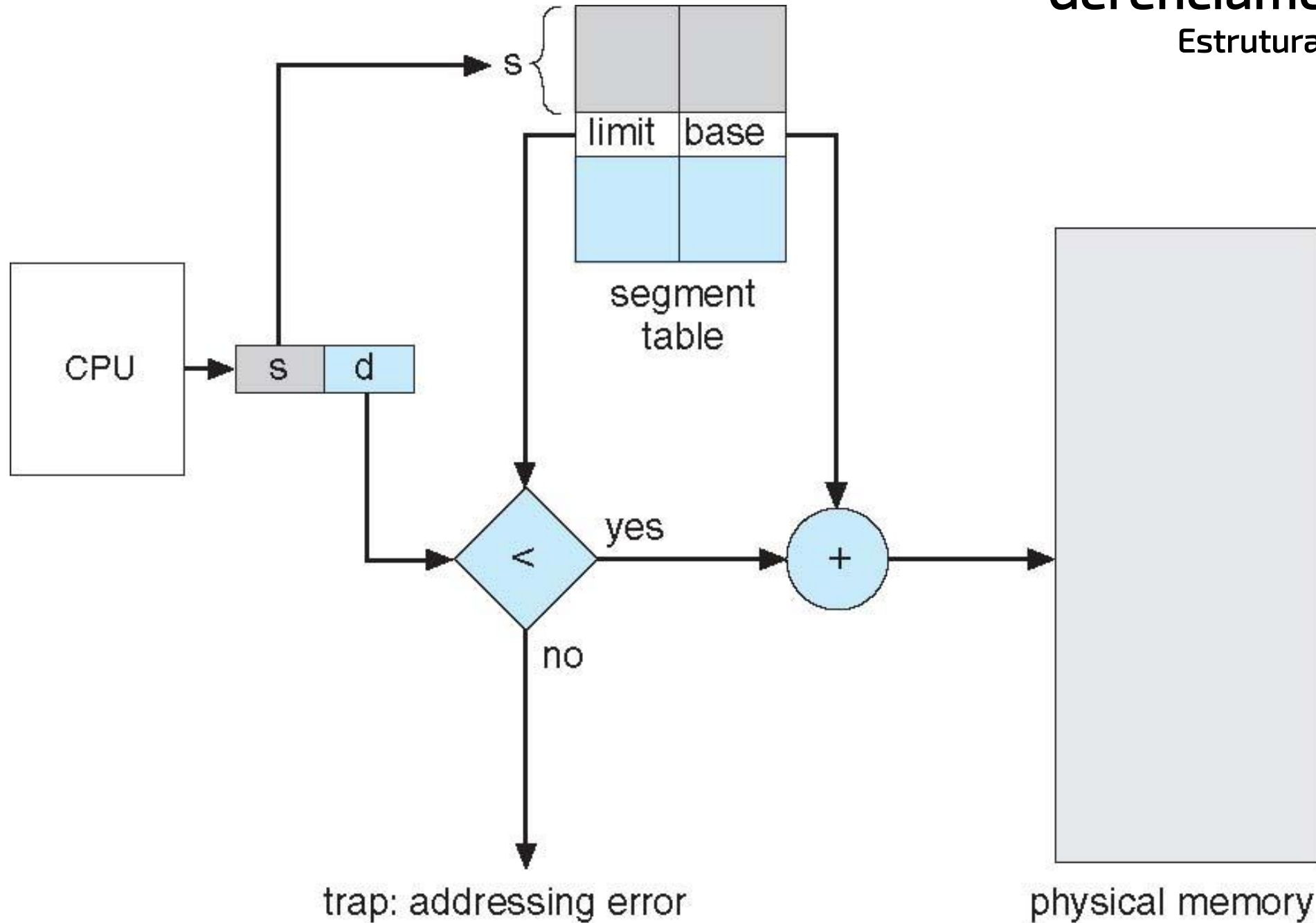
## Segmentação

- Apresenta os mesmos problemas da alocação contígua!



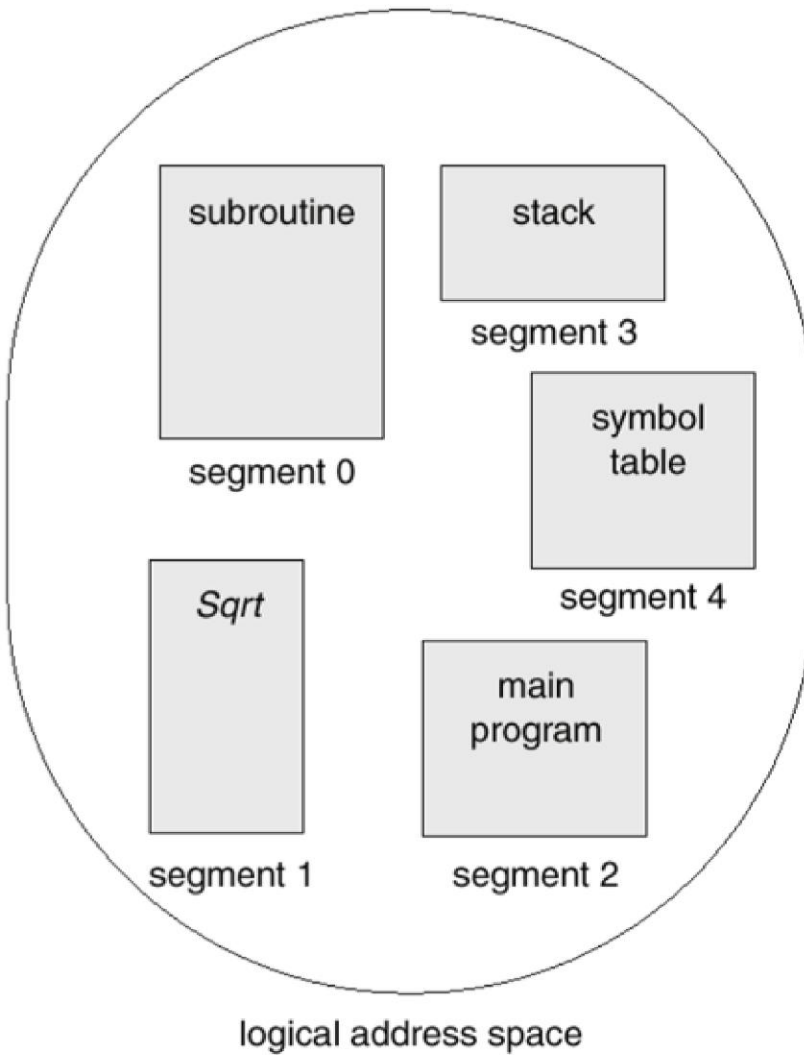
# Gerenciamento de Memória

Estrutura (HW) para Segmentação



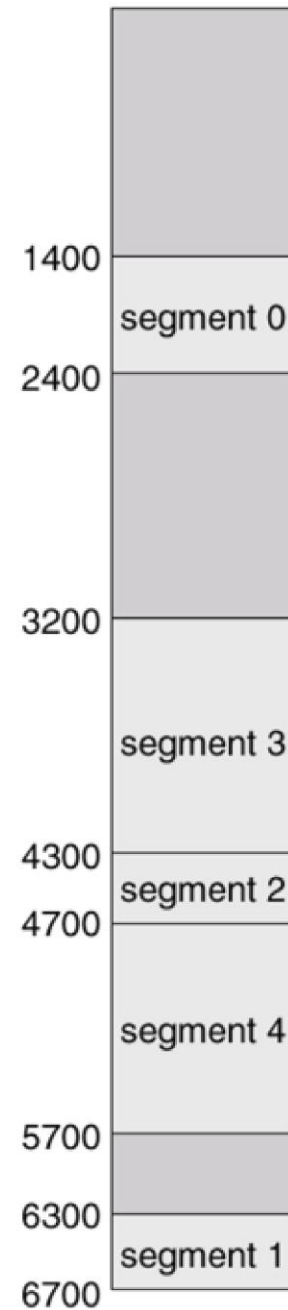
# Gerenciamento de Memória

## Exemplo de Segmentação



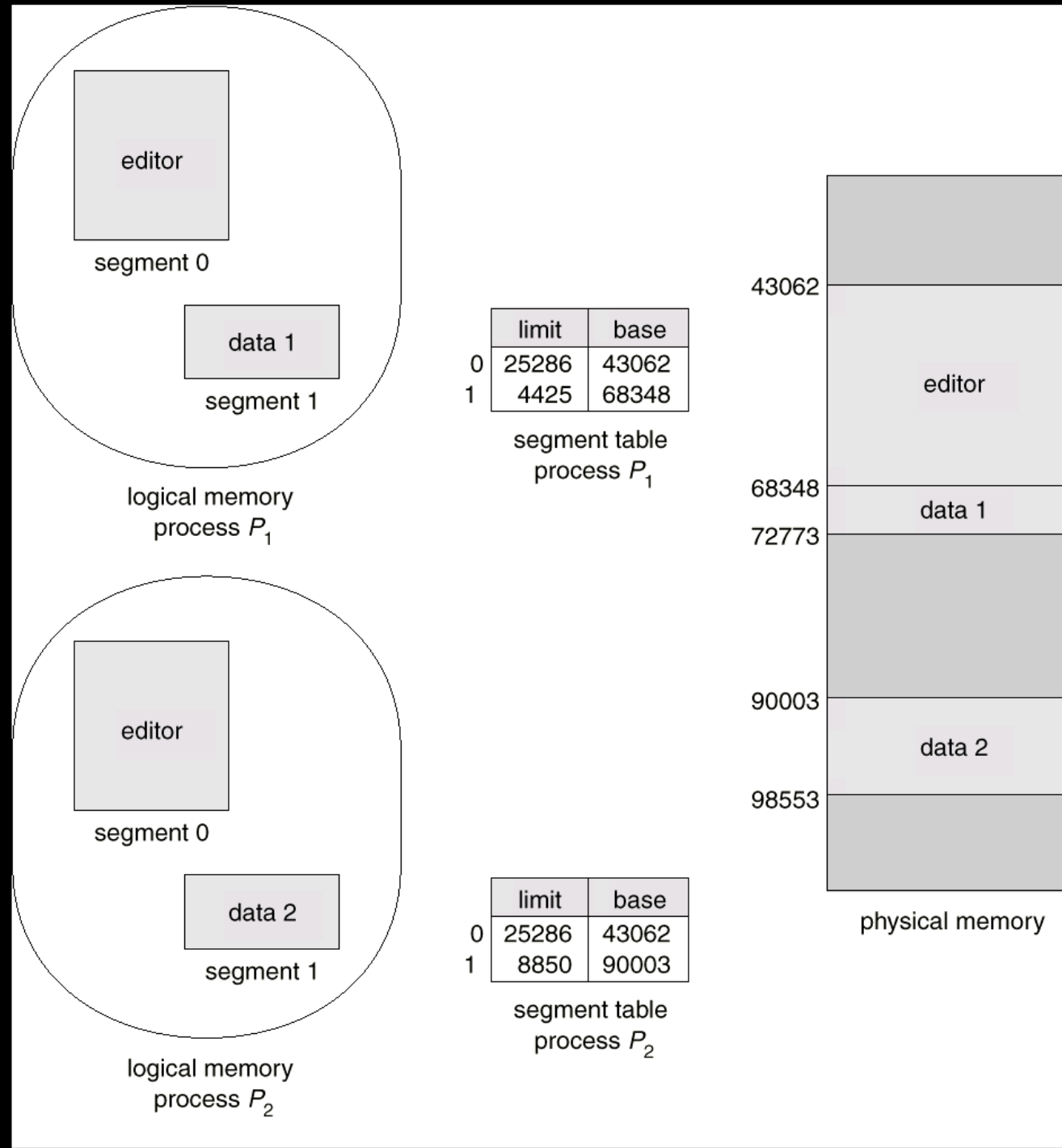
	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



# Gerenciamento de Memória

## Segmentos compartilhados

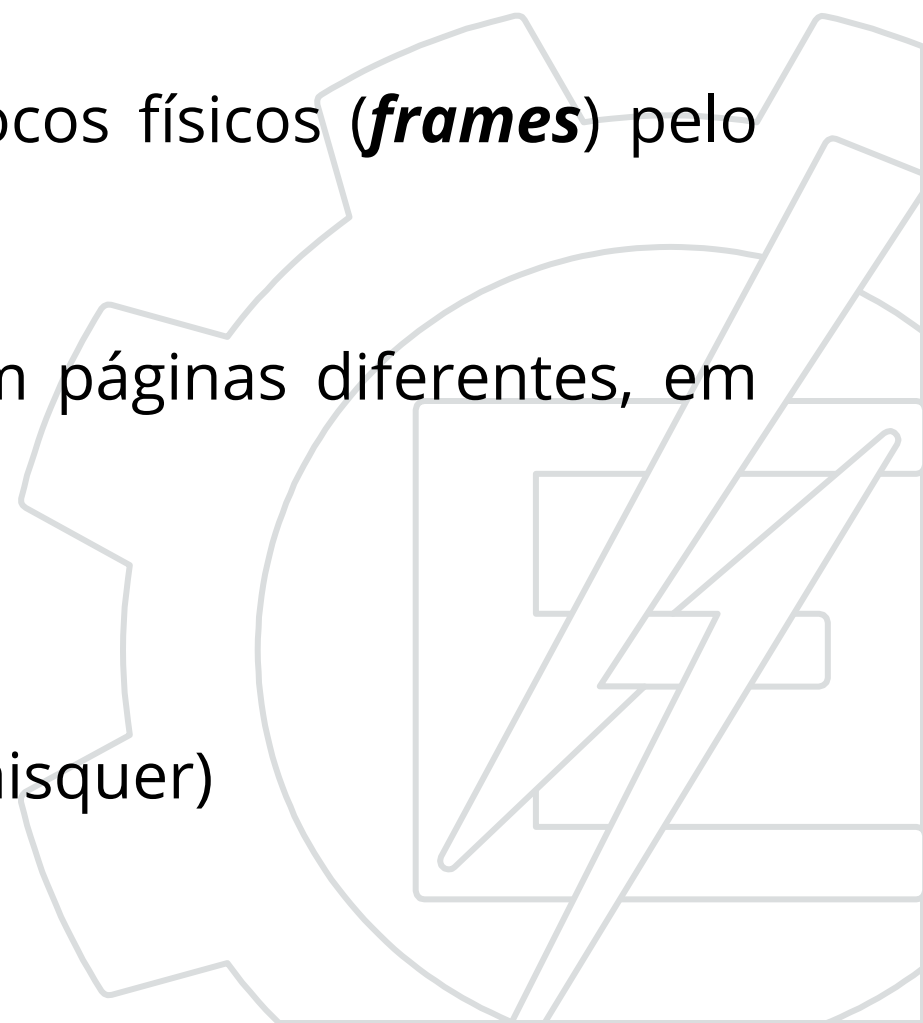




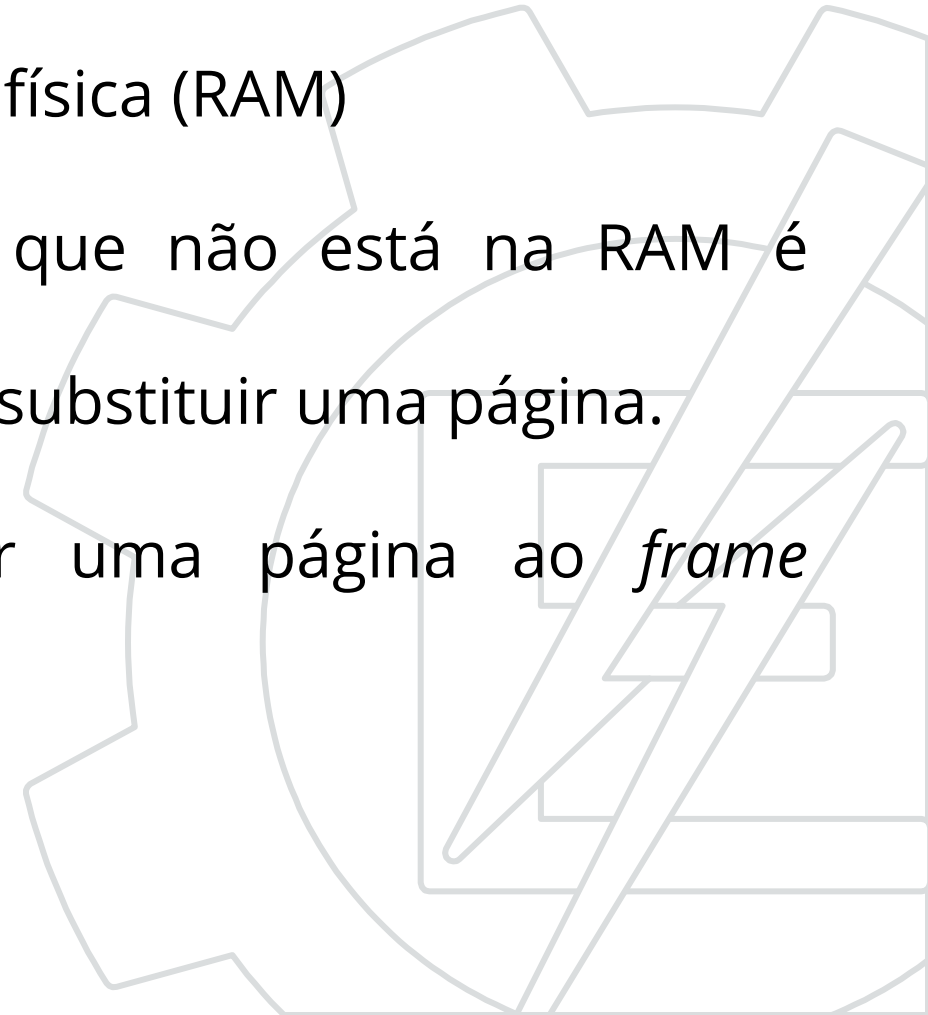
# Paginação



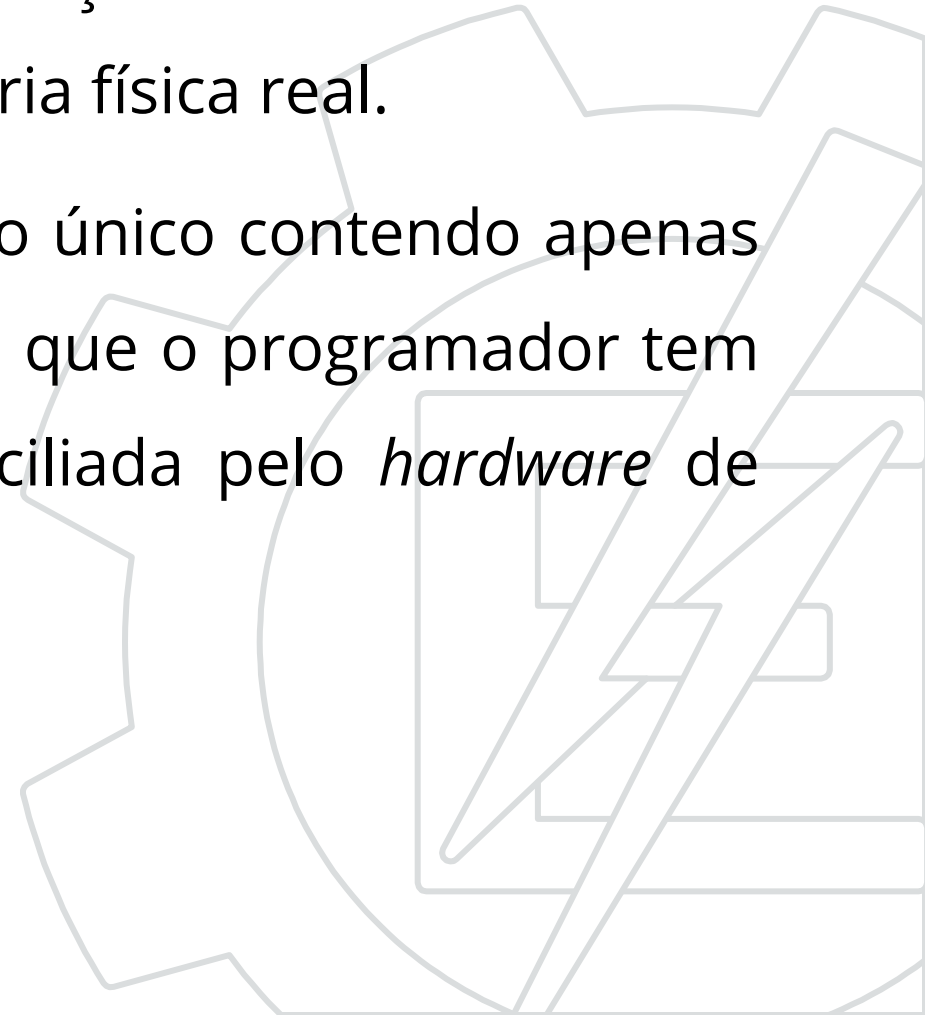
- A segmentação permite que o espaço de endereçamento físico de um processo seja não contíguo. A paginação é outro esquema de gerenciamento da memória que oferece essa vantagem.
- No entanto, a paginação evita a fragmentação externa e a necessidade de compactação, enquanto a segmentação não faz isso.
- Ela também resolve o considerável problema de acomodar trechos de memória de vários tamanhos na memória de retaguarda

- Memória (lógica/física) é dividida em blocos de tamanho fixo – potências de 2; p.ex.: 4 KB.
  - Blocos lógicos (***pages***) são mapeadas em blocos físicos (***frames***) pelo *hardware*
  - Endereços lógicos contíguos podem estar em páginas diferentes, em quadros não contíguos
  - Quadros vazios são gerenciados
  - Programa de  $n$  páginas requer  $n$  quadros (quaisquer)
  - Fragmentação interna (a última página)
- 

- **Páginas:** unidades de tamanho fixo no dispositivo secundário
- **Frames:** unidades correspondentes na memória física (RAM)
- **Page fault:** é o evento quando uma página que não está na RAM é referenciada – Utiliza uma *trap* para carregar ou substituir uma página.
- **Tabela de páginas:** estrutura para mapear uma página ao *frame* correspondente – cada processo possui uma.



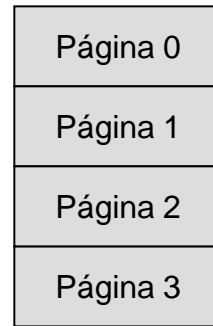
- Um aspecto importante da paginação é a separação clara entre a visão que o programador tem da memória e a memória física real.
- O programador vê a memória como um espaço único contendo apenas o programa corrente. A diferença entre a visão que o programador tem da memória e a memória física real é reconciliada pelo *hardware* de tradução de endereços.



- O sistema operacional deve ter conhecimento de que processos de usuário operam no espaço do usuário, e todos os endereços lógicos devem ser mapeados para produzir endereços físicos.
- Se um usuário faz uma chamada de sistema (para fazer I/O, por exemplo) e fornece um endereço como parâmetro (digamos, um buffer), esse endereço deve ser mapeado para produzir o endereço físico correto.
- O sistema operacional mantém uma cópia da **tabela de páginas** de cada processo, assim como mantém uma cópia do contador de instruções e dos conteúdos dos registradores

# Gerenciamento de Memória

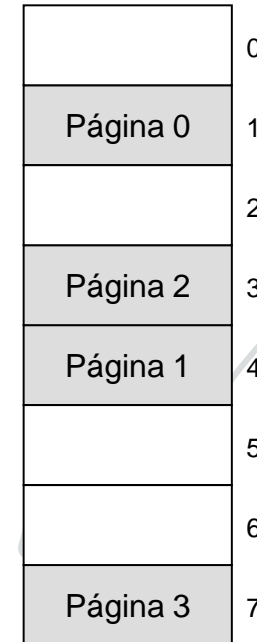
## Tabelas de Páginas



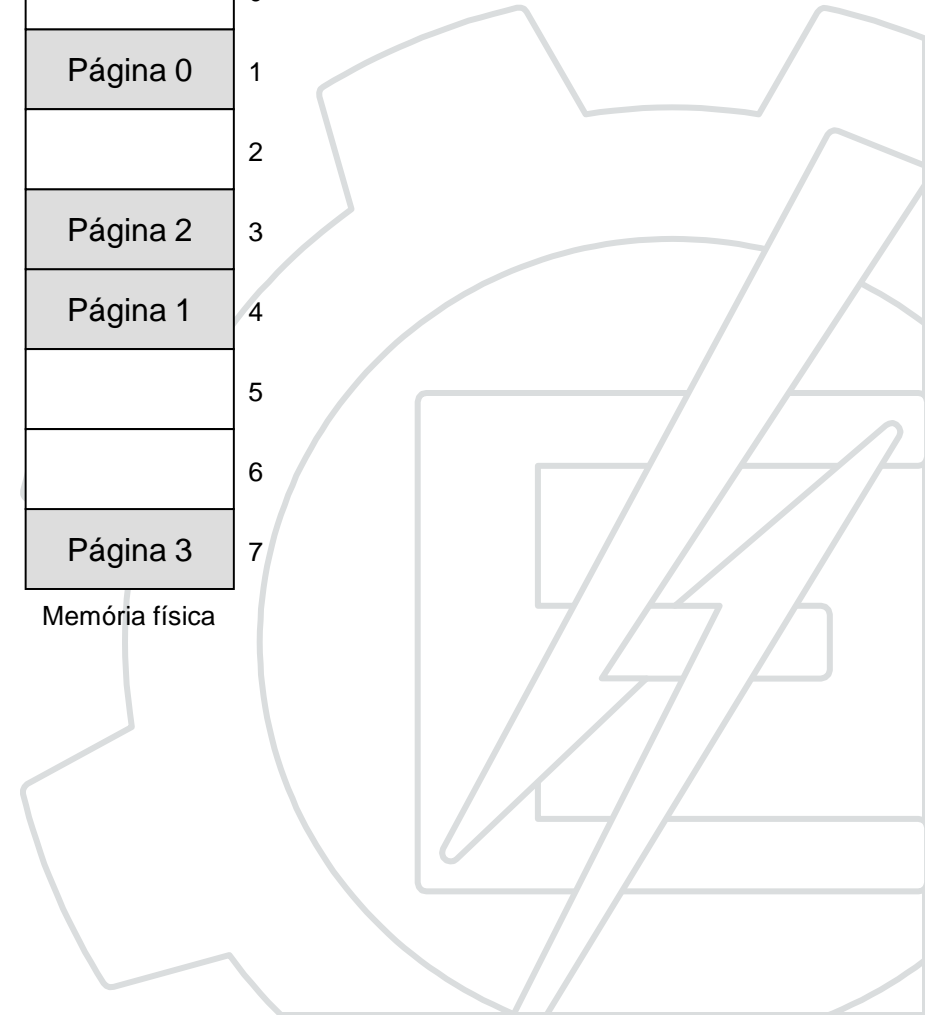
Espaço de  
endereçamento

0	1
1	4
2	3
3	7

Tabela de  
páginas

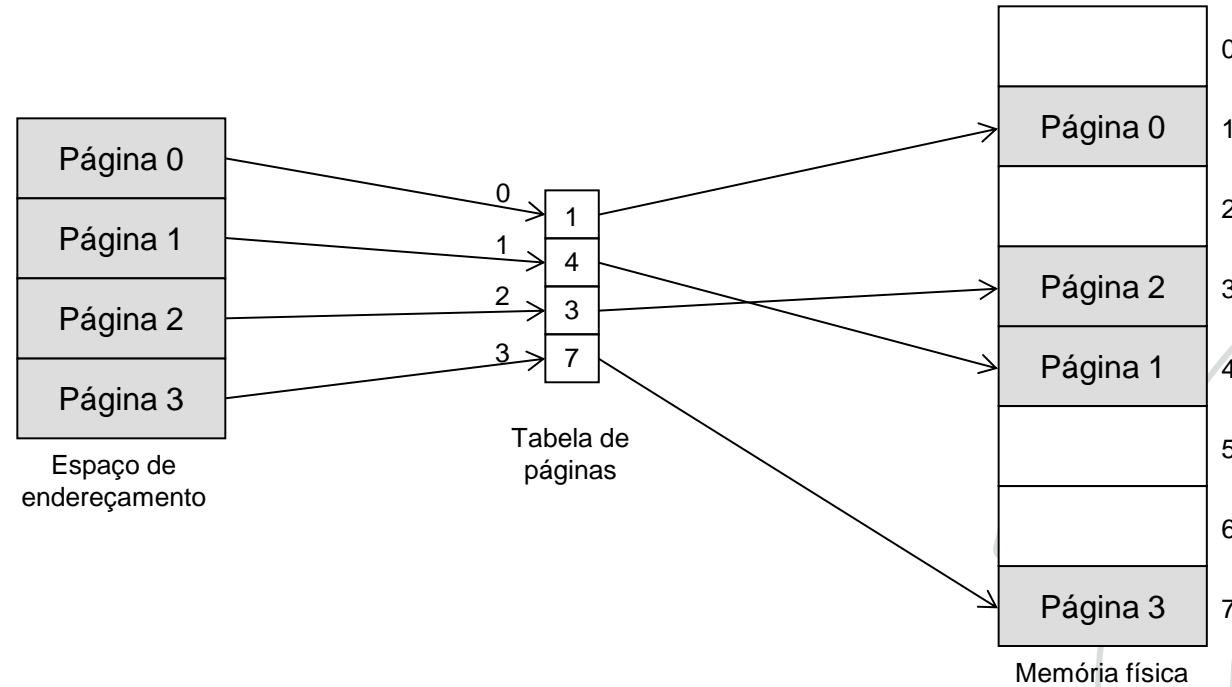


Memória física



# Gerenciamento de Memória

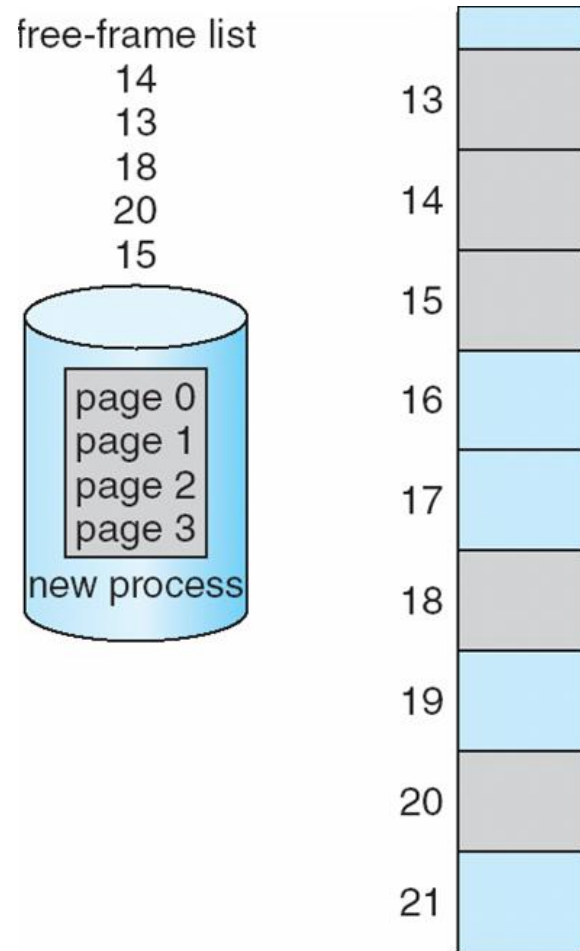
## Tabelas de Páginas



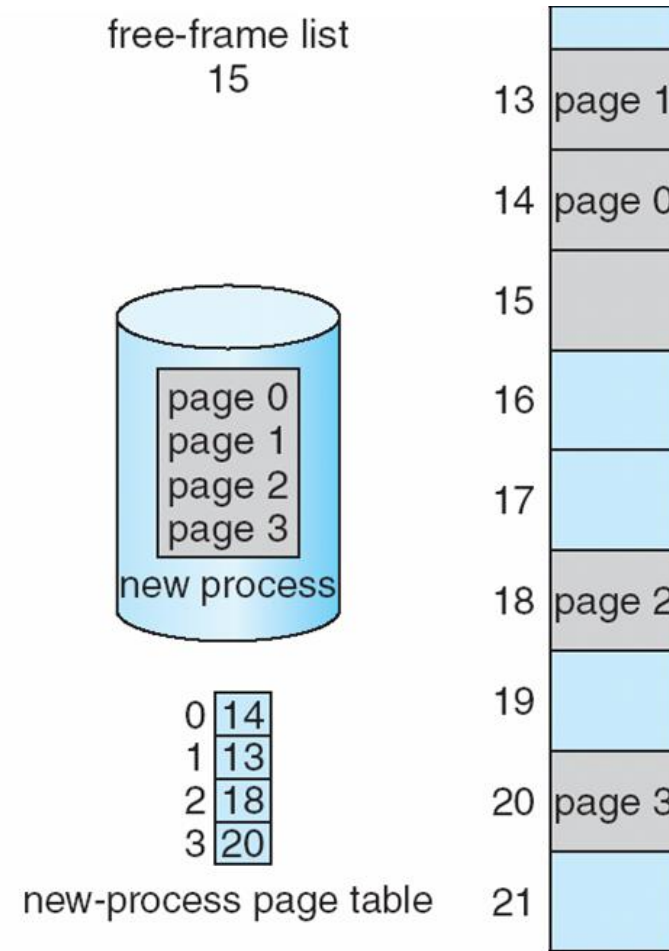


# Gerenciamento de Memória

## Paginação – Gerenciamento de quadros livres



(a)  
Antes da alocação



(b)  
Depois da alocação

# Gerenciamento de Memória

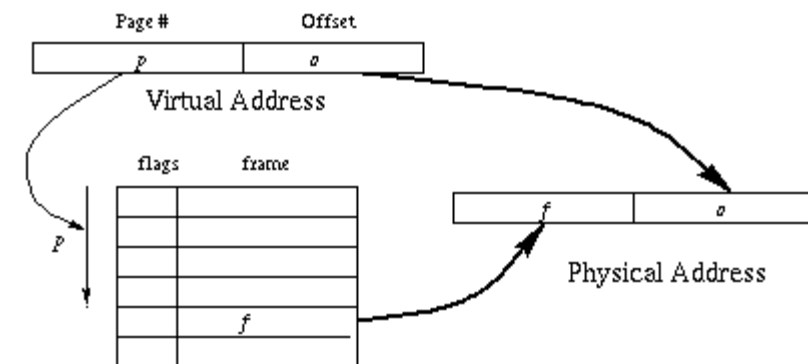
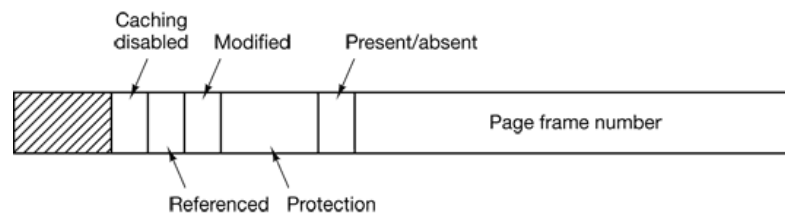
## Paginação – Componentes do endereço

- Número de página (**p**): utilizado como índice para uma tabela de páginas.
- Deslocamento de página ou *offset* (**d**): combinado com o endereço de base para definir o endereço de memória físico que é enviado à unidade de memória.

Virtual Address



Page Table Entry



# Gerenciamento de Memória

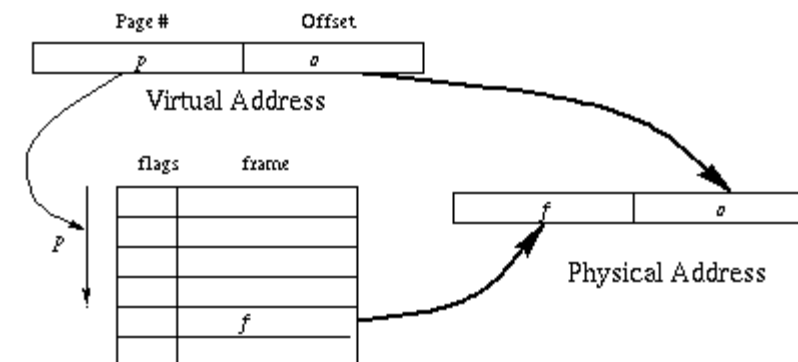
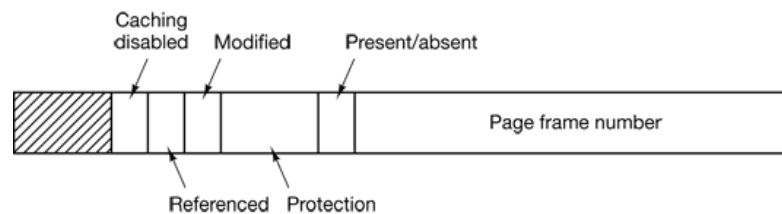
## Paginação – Componentes do endereço

- Páginas maiores: leitura mais eficiente, tabela menor, mais fragmentação interna.
- Páginas menores: leitura menos eficiente, tabela maior, menor fragmentação interna.

Virtual Address



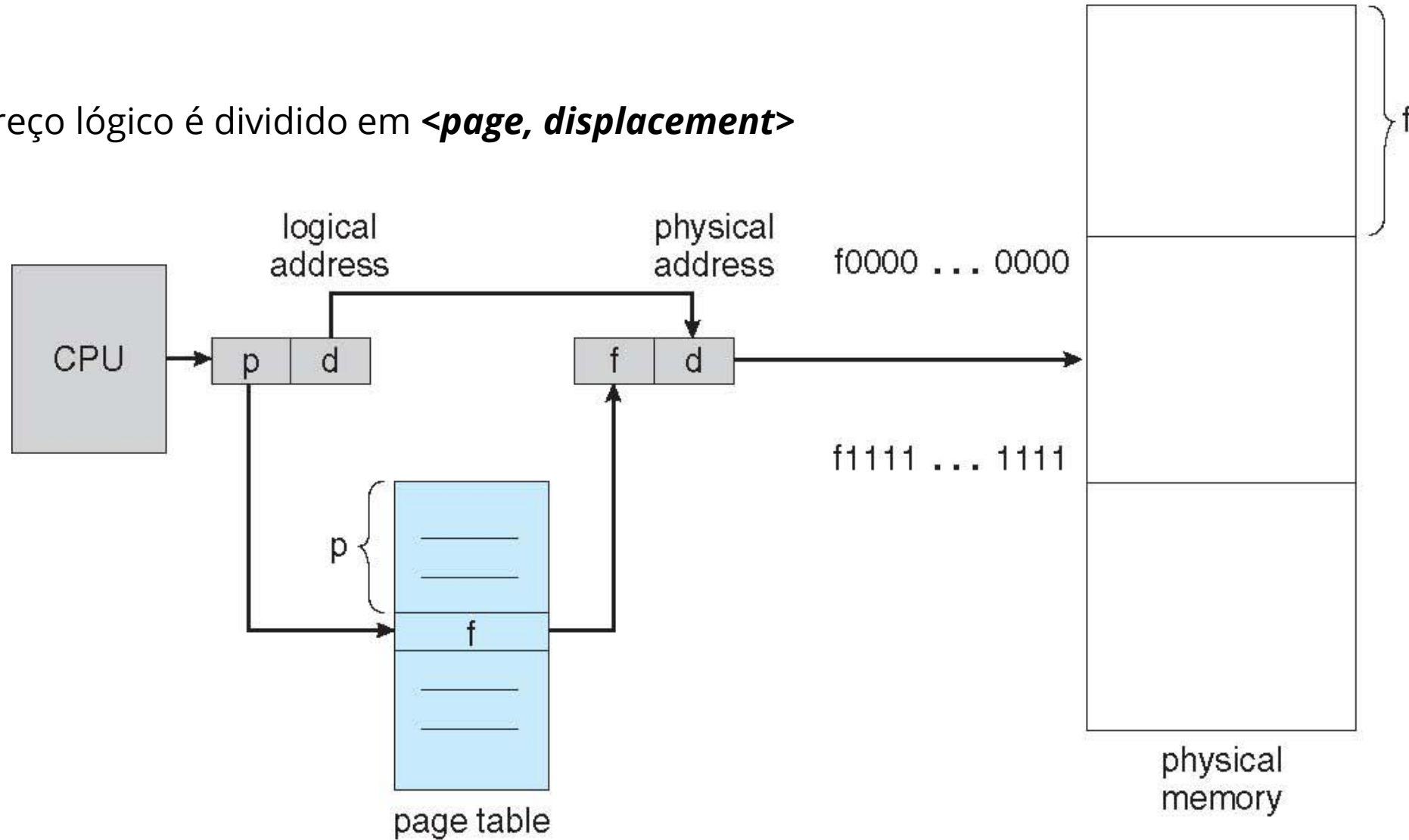
Page Table Entry



# Gerenciamento de Memória

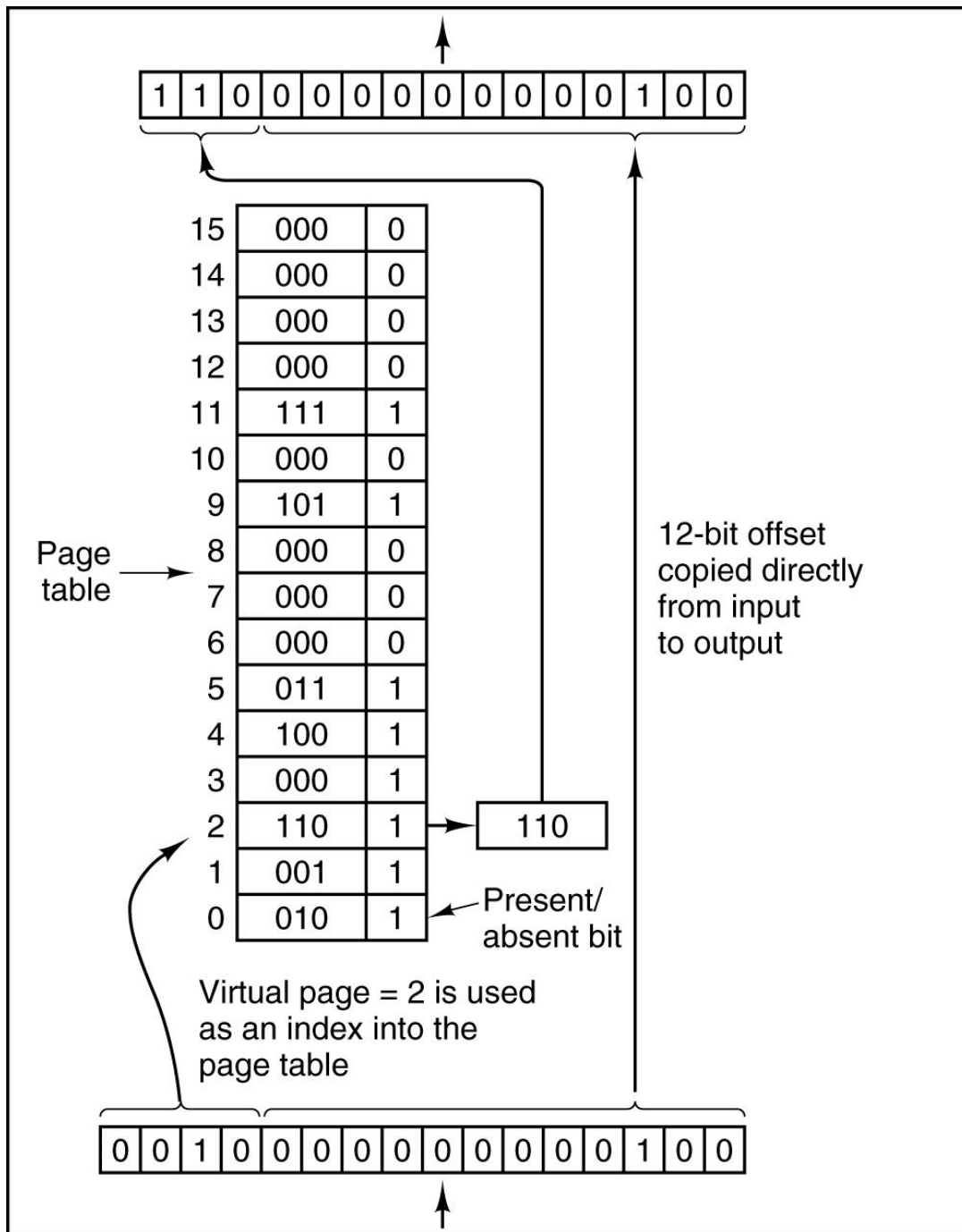
## Paginação – Estrutura de tradução de endereços

- Endereço lógico é dividido em **<page, displacement>**



# Gerenciamento de Memória

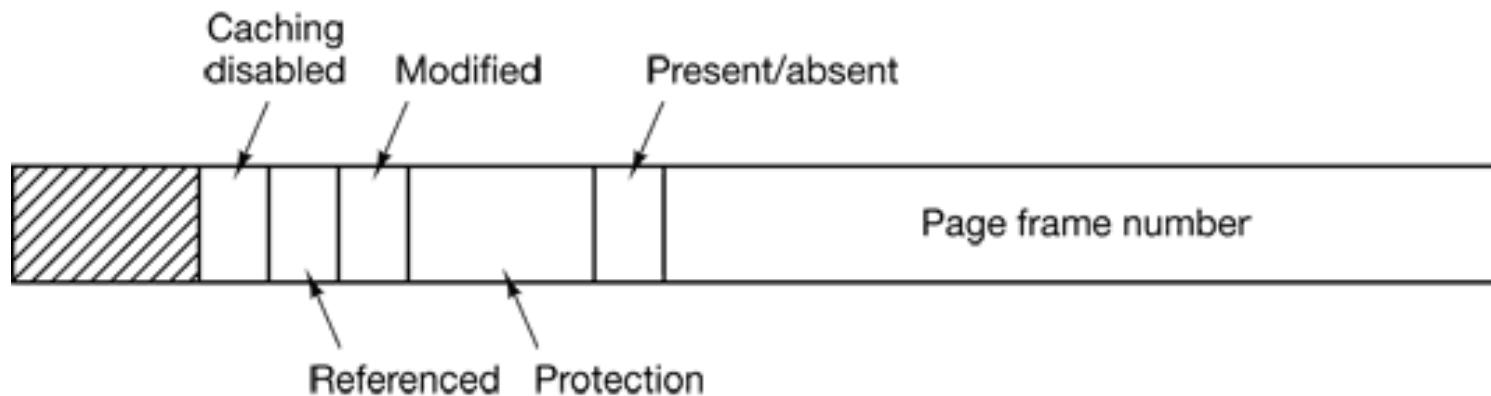
## Paginação – Estrutura de tradução de endereços



- MMU com 16 páginas de 4Kb.
- Endereço virtual de 16 bits.
- *Hardware* com 8 frames

- Componentes

- a) *Page frame number*: identifica o número da página real
- b) Bit de **residência**: se 1 → a página é válida e está presente na RAM.
  - Se 0 → ocorre um *Page fault*.
- a) Bit de **proteção**: 0 → *leitura/escrita*, 1 → *leitura*, 2 → *execução*
- b) Bit de **modificação**: 1 → página alterada, 0 → página não-alterada
- c) Bit de **referência**: 1 → foi referenciada “recentemente”
- d) Bit de **cache**: permite desabilitar o *caching* da página.

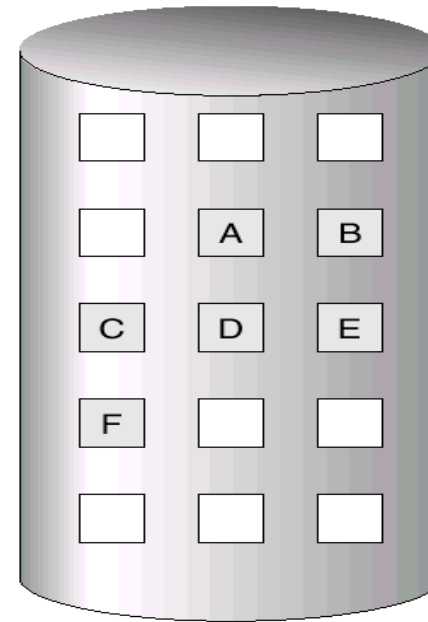
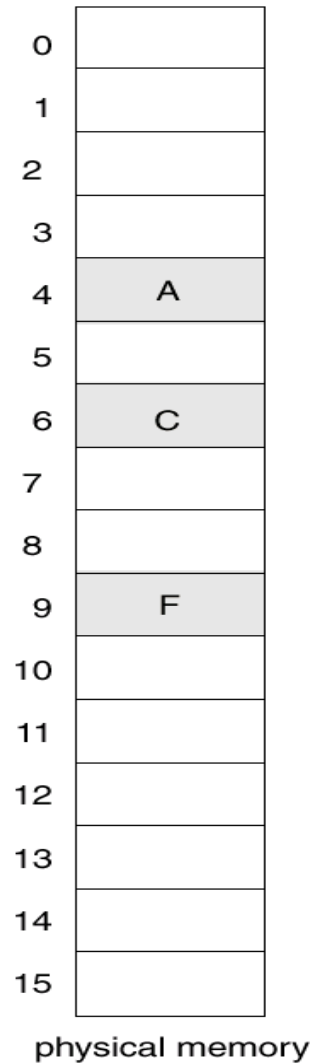
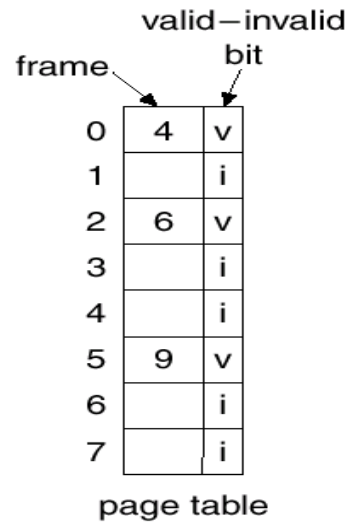
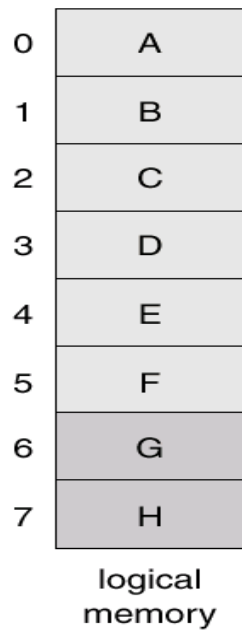


- A proteção da memória em um ambiente paginado é executada por **bits de proteção associados a cada quadro**.
- Um bit pode definir se uma página é de leitura-gravação ou somente-de-leitura.
- A tentativa de gravação em uma página somente-de-leitura provoca uma interceptação de hardware para o sistema operacional (ou violação da proteção à memória).
- Um bit adicional é geralmente anexado a cada entrada da tabela de páginas: um bit válido-inválido.

# Gerenciamento de Memória

## Paginação – Proteção de memória

- O bit válido/inválido é usado para controlar a presença (ou não) da página (lógica) na memória física (principal)



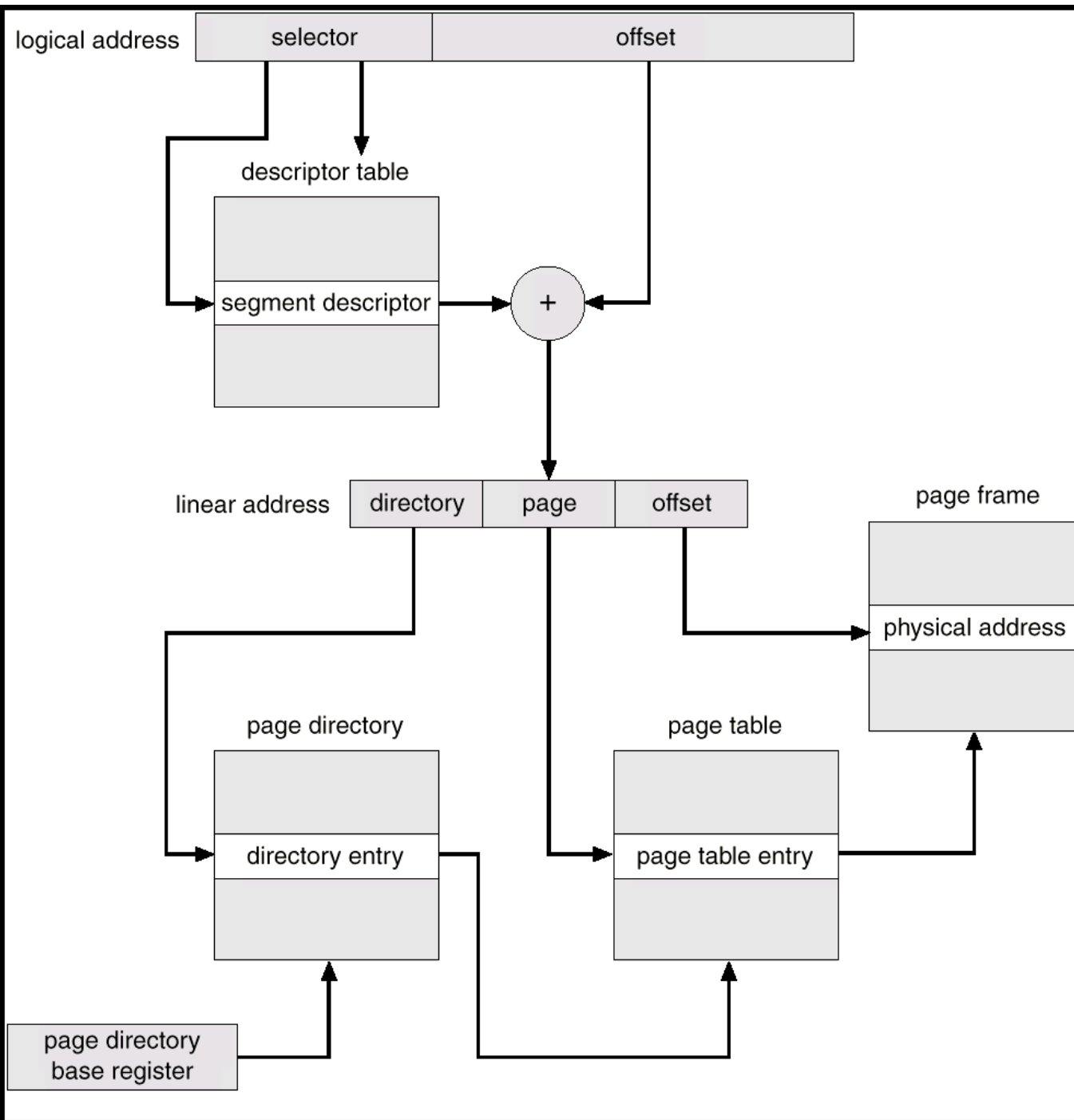


# Gerenciamento de Memória

## Segmentação com paginação

- É possível combinar as duas técnicas e paginar cada segmento
  - Usado no Multics e na arquitetura Intel 386
  - Flexibilidade x complexidade





# Gerenciamento de Memória

## Endereçamento no Intel80386



# Bibliografia

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

## Capítulo 3.

<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 9.**

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



# Sistemas Operacionais

Prof. Otávio Gomes

[otavio.gomes@unifei.edu.br](mailto:otavio.gomes@unifei.edu.br)

