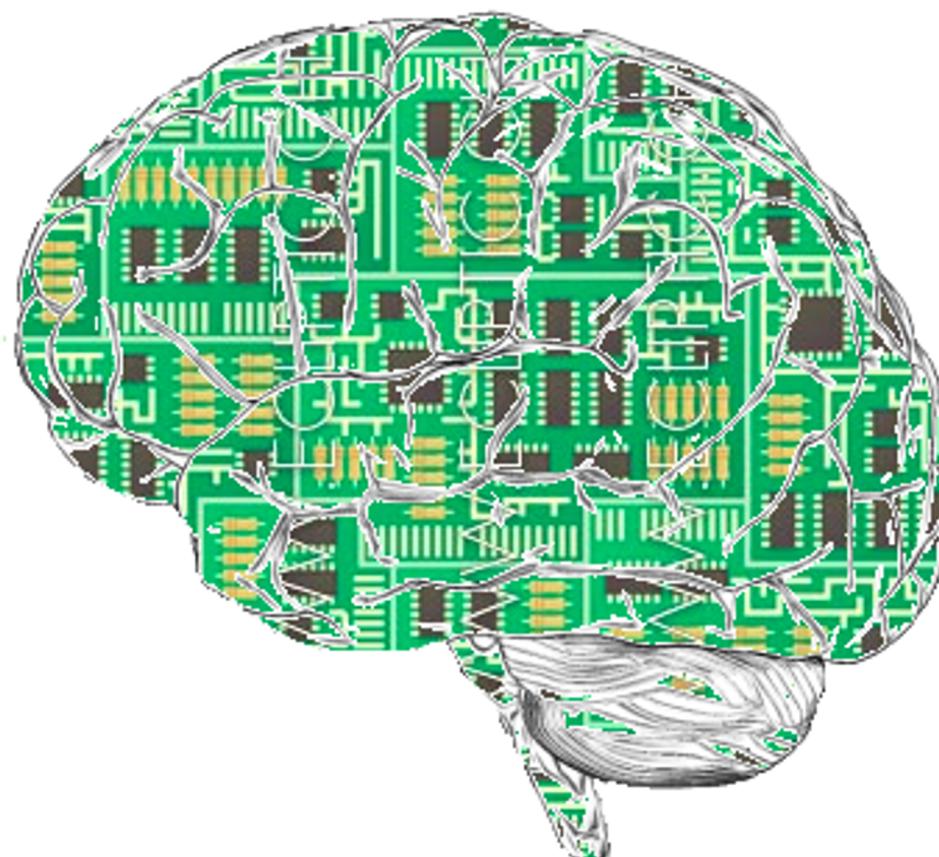


# Capítulo 4

## Gerenciamento de Memória



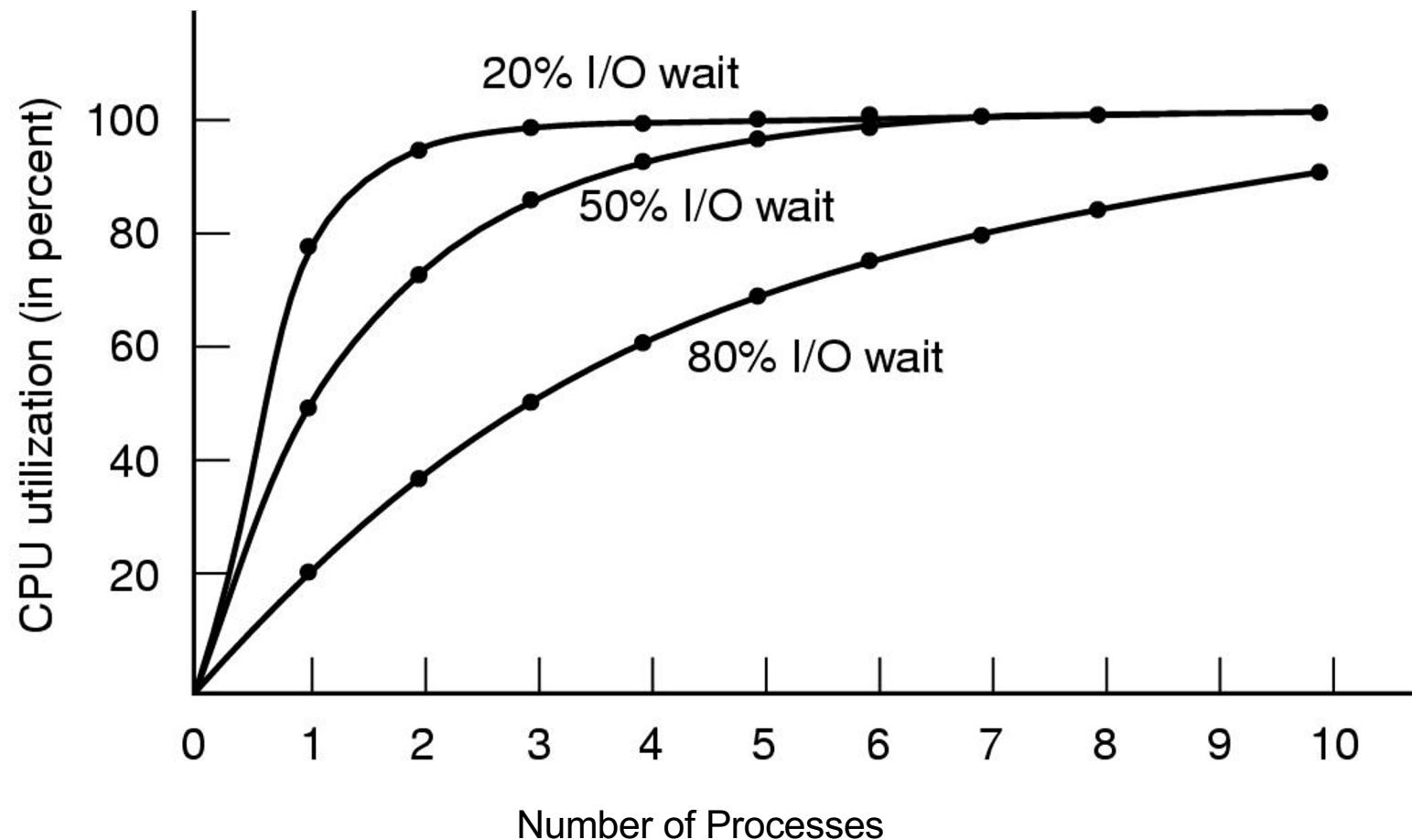
# Introdução

- Importante função de um SO é gerenciar o recurso memória principal principal (mem. física, RAM, core)
- Memória principal: um recurso relativamente caro e limitado
- O tamanho de todos os softwares aumentou muito
- O uso da memória pode ser o gargalo para ter uma alta utilização da CPU
- Usa-se memória secundária (disco) para colocar os processos ativos (suspenso) que “não cabem” mais na memória principal
- Mas o acesso a essa memória secundária é  $10^6$  vezes mais lento que acesso à RAM !!

# Requisitos do Gerenciamento de Memória

1. Permitir que muitos processos estejam na memória RAM (para ter maior benefício da multiprogramação)
2. Executar processos que sejam maiores do que o tamanho da RAM
3. Permitir que processos parcialmente carregados possam iniciar a execução (isso reduz o tempo de início do programa)
4. Permitir processos re-alocáveis, que possam executar em qualquer parte da memória RAM e possam ser movidos de um lugar para outro
5. Permitir que processos possam solicitar mais memória ao longo de suas execuções.
6. Permitir que processos possam compartilhar memória.

# Efeito da Multiprogramação na Utilização da CPU



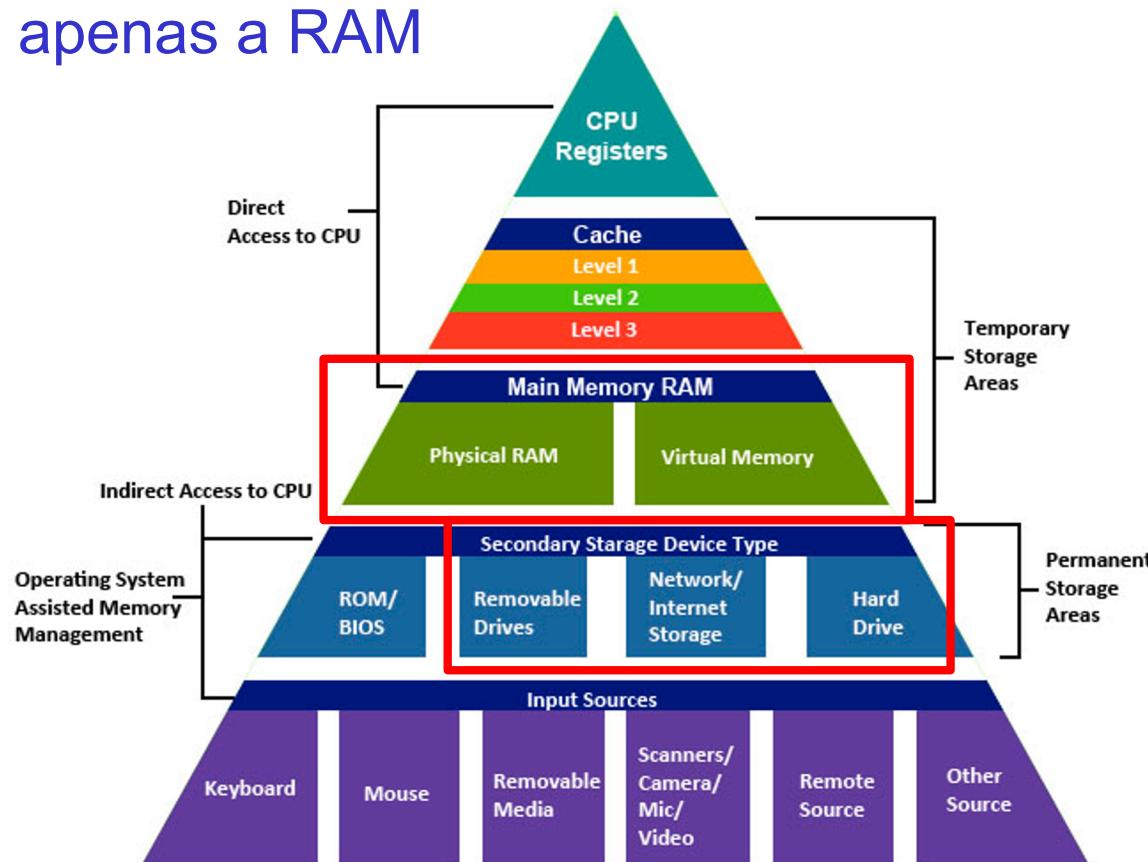
Utilização da CPU em função do grau de multiprogramação (i.e. do número de processos na memória)

# Desafios introduzidos pela multiprogramação

- Reallocação: não determinar, a priori, em qual área de memória RAM um processo vai ser executado, dado que outros processos podem estar na memória também
  - Endereço de memória (dados, pilha e código) precisam ser endereços lógicos que são mapeados dinamicamente
- Proteção/Isolamento: precisa-se evitar que processos accessem áreas de memória alocadas a outros processos
  - o mapeamento de memória precisa ser individual para cada processo
  - e não pode conflitar com o mapeamento determinado para os outros processos
- Compartilhamento: deve ser possível também permitir um compartilhamento do segmento de código executável por vários processos (ex. bibliotecas)

# Hierarquia de Memória

- Papel do Subsistema **Gerenciador de Memória (GM)** é de mover dados entre a memória principal e a secundária de *forma transparente*
- Como se o sistema dispusesse de muito mais memória do que apenas a RAM



# Swapping

- É necessário sempre que não é possível manter todos os processos em execução na memória
- Processos podem ser *swapped out* e posteriormente *swapped in*.
- Quando é swapped-in para a memória, pode ser carregado em outra região da memória, permitindo uma utilização mais eficaz da memória
- Processos swapped-out permanecem no estado "suspenso"
- Swapping impõe um alto custo em tempo de execução (e de resposta) a um processo
- Por isso, evita-se fazer isso para processos interativos e controlados pelo terminal

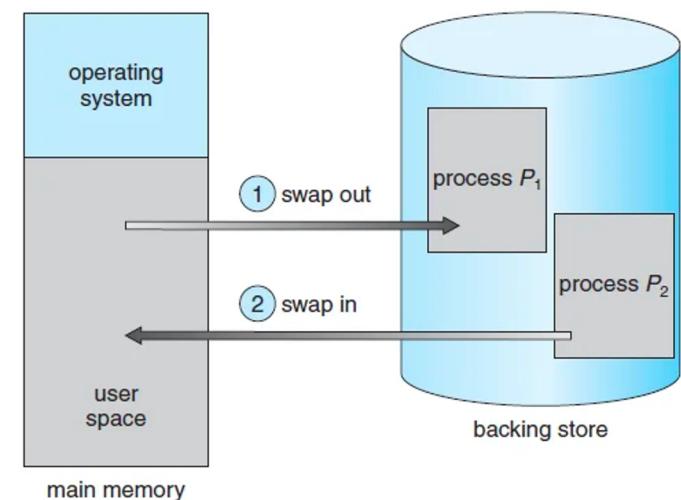
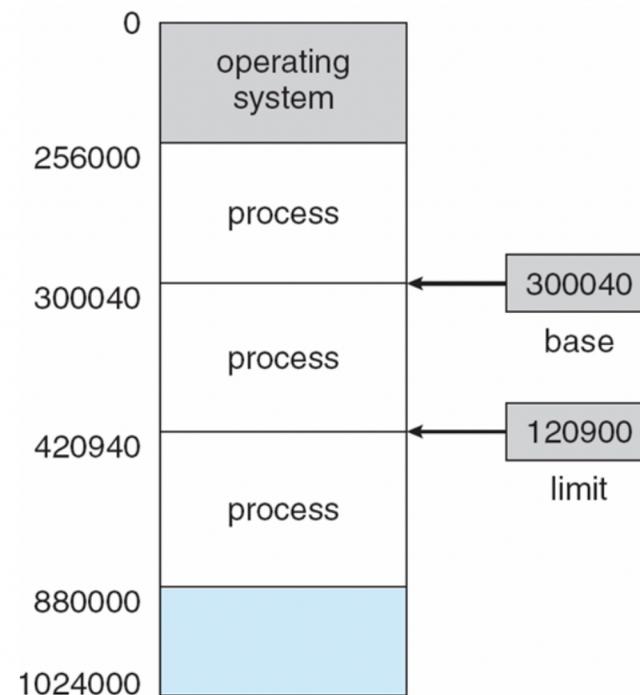


Figure 8.5 Swapping of two processes using a disk as a backing store.

# Tradução de endereços lógicos para endereços físicos

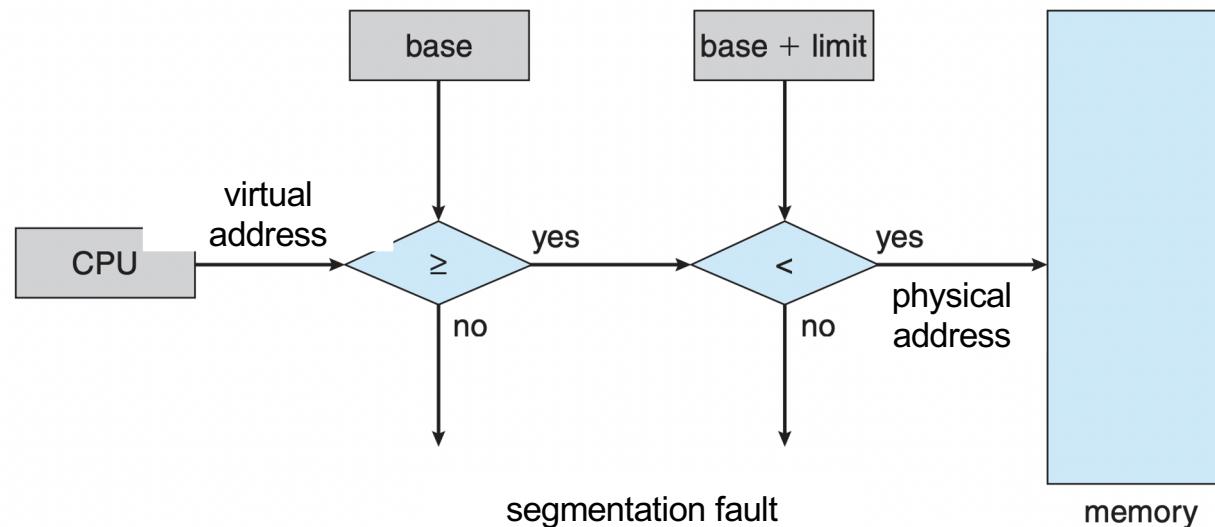
- **endereço lógico** = qualquer endereço de memória gerado pela CPU (também chamado endereço virtual)
- **endereço físico** = um endereço de memória como visto pela unidade (ou controladora) da memória
- Um par de **registrador base** e **registrador limite** definem o espaço de endereçamento do processo na memória



# Tradução de endereços lógicos para endereços físicos

Cada processo tem associado a ele um

- par de **registrador base e registrador limite** definem o espaço de endereçamento
- a cada endereço virtual de memória gerado pela CPU verifica-se que o endereço está entre a base e o limite
- a cada troca de contexto de processos o par (BR/LR) precisa ser trocado também



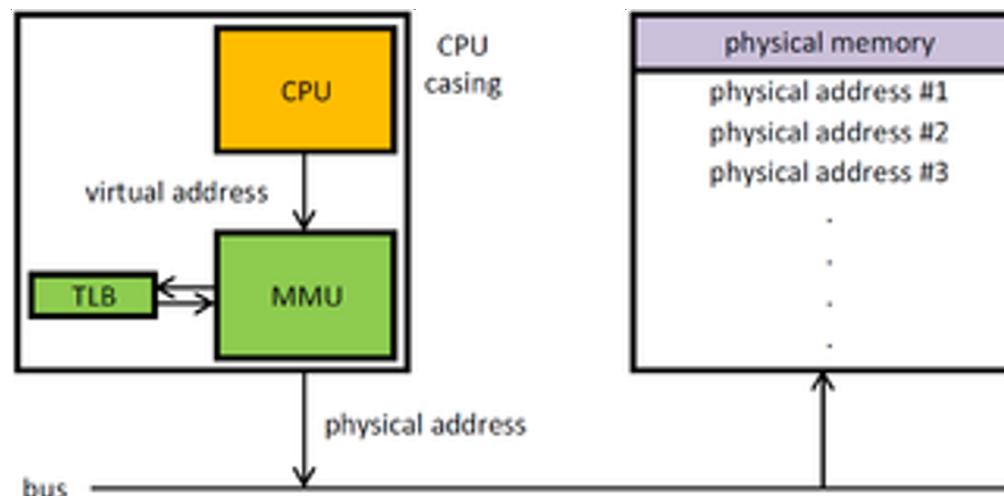
# Espaços de Endereçamento

**Espaço de endereçamento lógico/virtual**:: o conjunto de endereços que um processo consegue referenciar

- endereços lógicos devem ser traduzidos para o espaço físico da memória. (um offset a partir do base register)
- Considere que o programa tem um espaço de endereço lógico/virtual contíguo que começa com 0, e tem o tamanho do Limit register e um espaço de endereço físico contíguo que começa no endereço contido no base register

# Memory Management Unit (MMU)

- O Ger. Memória interage com uma componente de hardware: **Memory Management Unit (MMU)**
- A MMU é responsável por traduzir **endereços lógicos/virtuais** (do programa executável) para **endereços físicos** (da memória RAM)
- Têm um TBL Cache com os endereços recentemente acessados

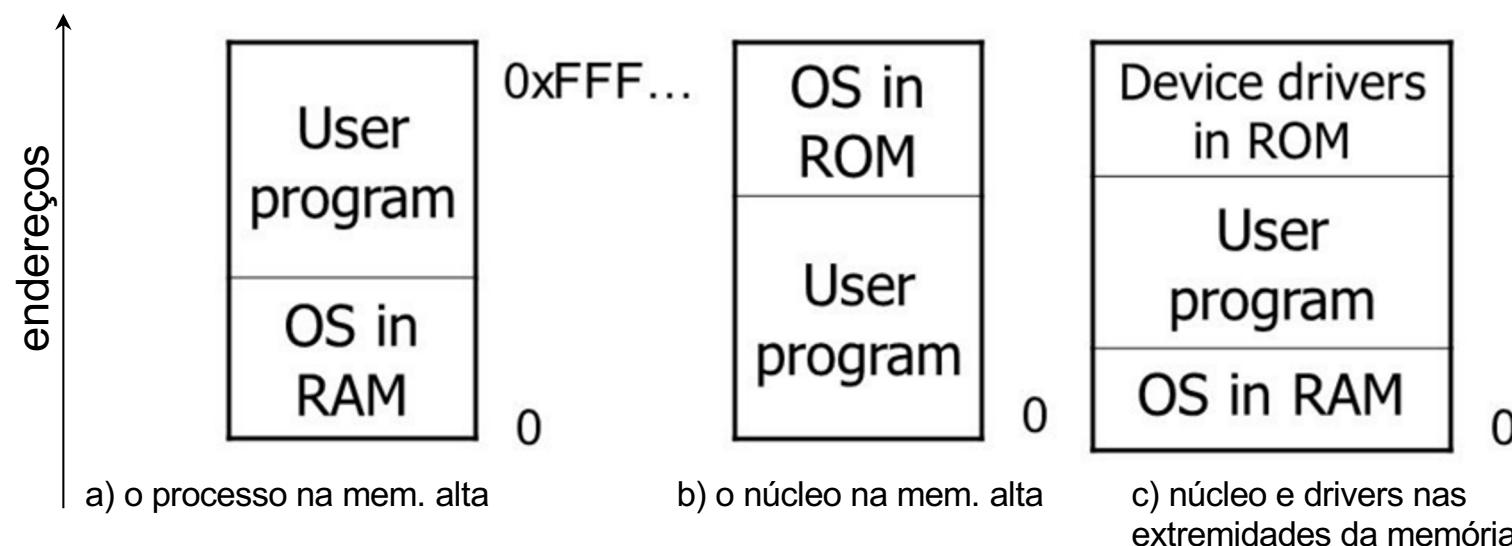


CPU: Central Processing Unit  
MMU: Memory Management Unit  
TLB: Translation lookaside buffer

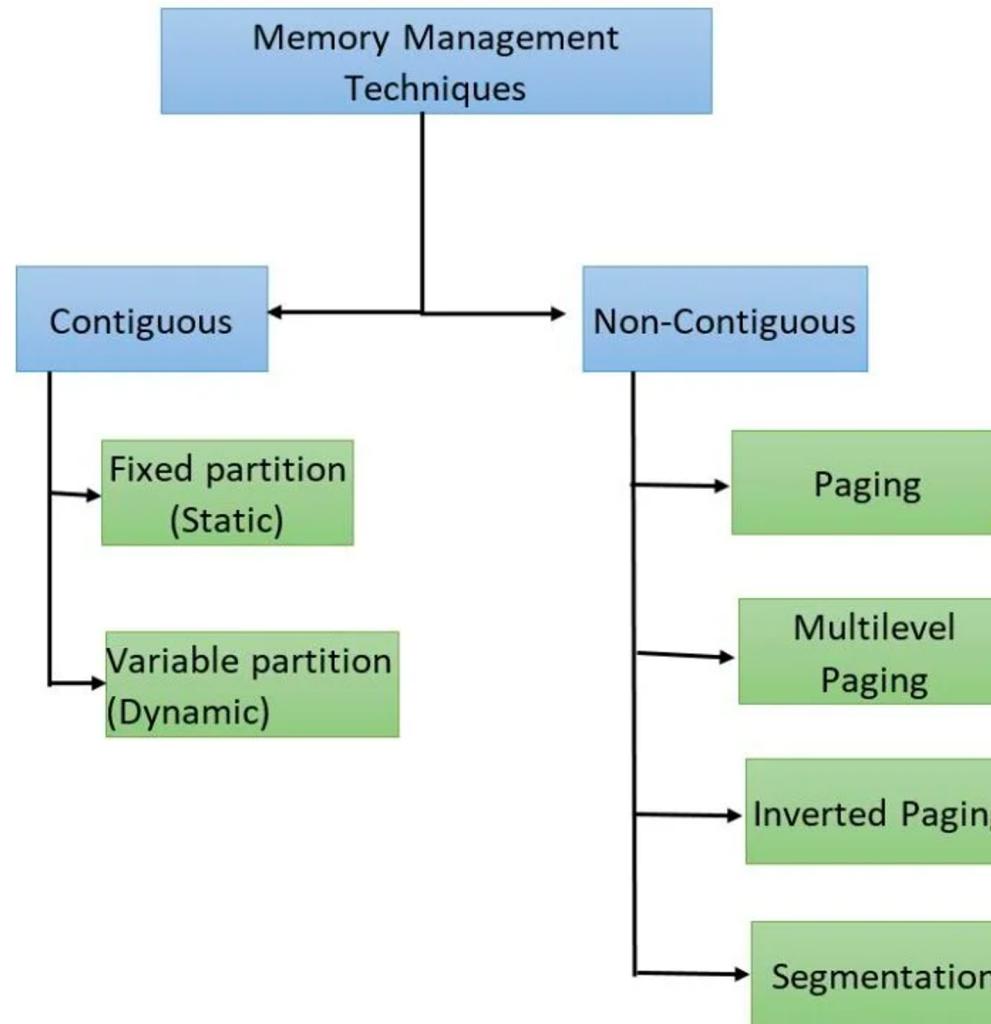
# Gestão de memória para Monoprogramação

Para sistemas com um único usuário e/ou dispositivos simples embarcados:

- Só é possível executar um único processo por vez. Se este faz E/S, a CPU irá esperar ociosa.
- Baixa utilização do processador
- tamanho da memória limita o tamanho do programa em execução
- Não serve para computadores de propósito geral com multiprogramação, **ter vários processos na memória é essencial!!**



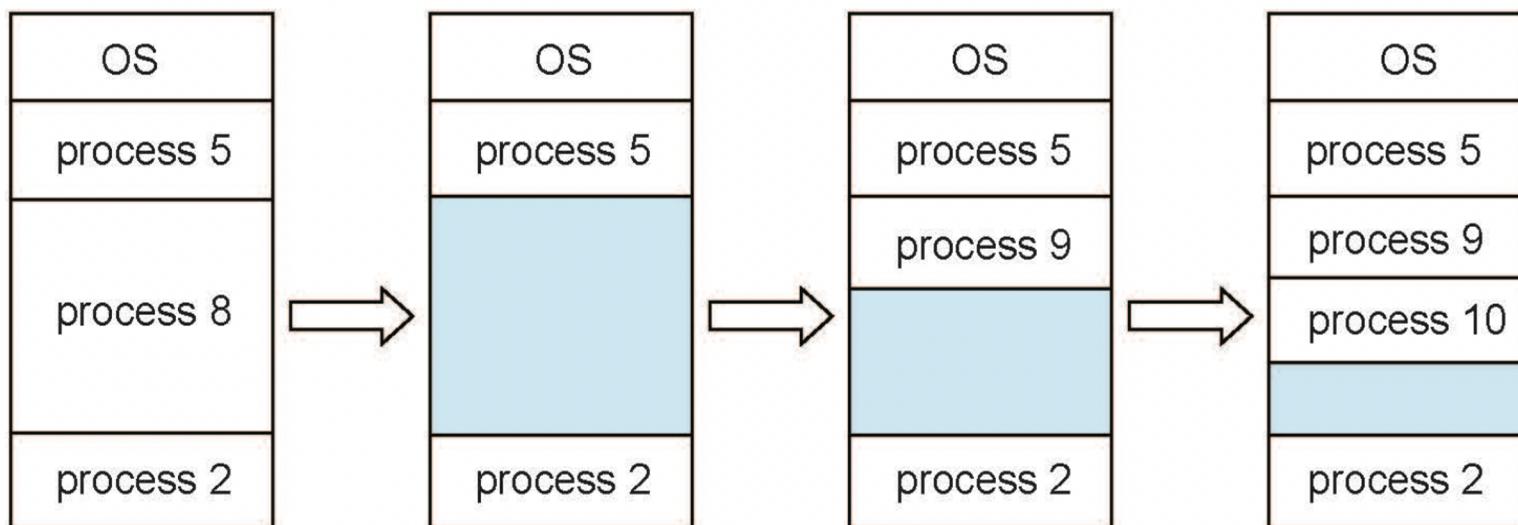
# Formas de Alocação de Memória



- alocação contígua = processo é carregado em uma região contígua da memória principal
- alocação não-contígua = processo é carregado em regiões distribuídas da memória

# Alocação de memória contígua com partições

- Os esquemas de alocação com particionamento podem ser de dois tipos:
  - Particionamento fixo
  - Particionamento variável
- Em ambos os casos, objetivo é encontrar uma partição livre que comporte o espaço de endereçamento do novo processo



# Alocação de memória contígua com partições fixas

- Pode-se dividir a memória em n partições de tamanhos pré-definidos (provavelmente de tamanhos diferentes).
- Essa forma de alocar memória quase não é mais usada
- Os processos à espera de serem carregados na memória são colocados em filas de entrada associadas à menor partição capaz de armazená-los.
- Partições fixas fragmentação interna (o resto de espaço de memória não utilizado pelo processo)

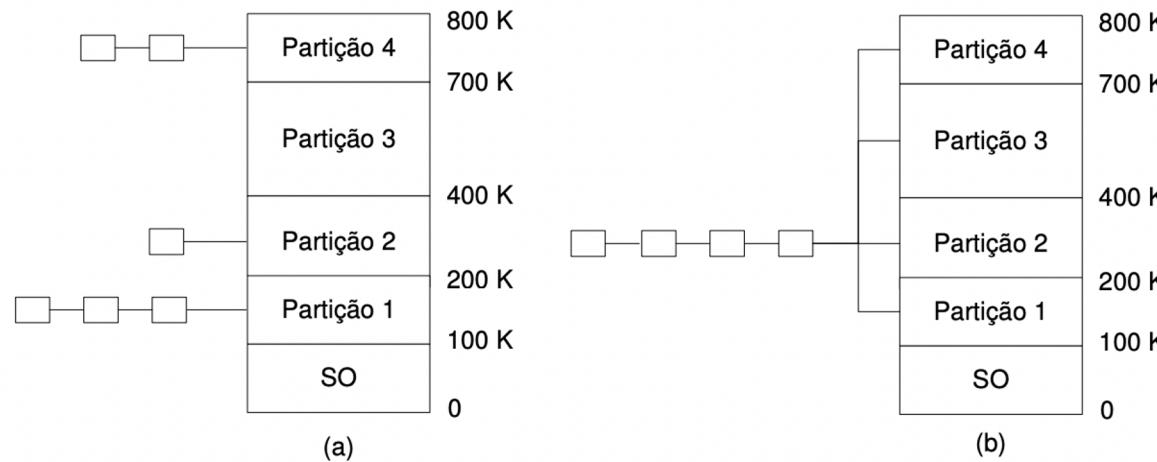


Figura 2. Partições fixas com filas de entrada separadas (a) e com uma única fila (b).

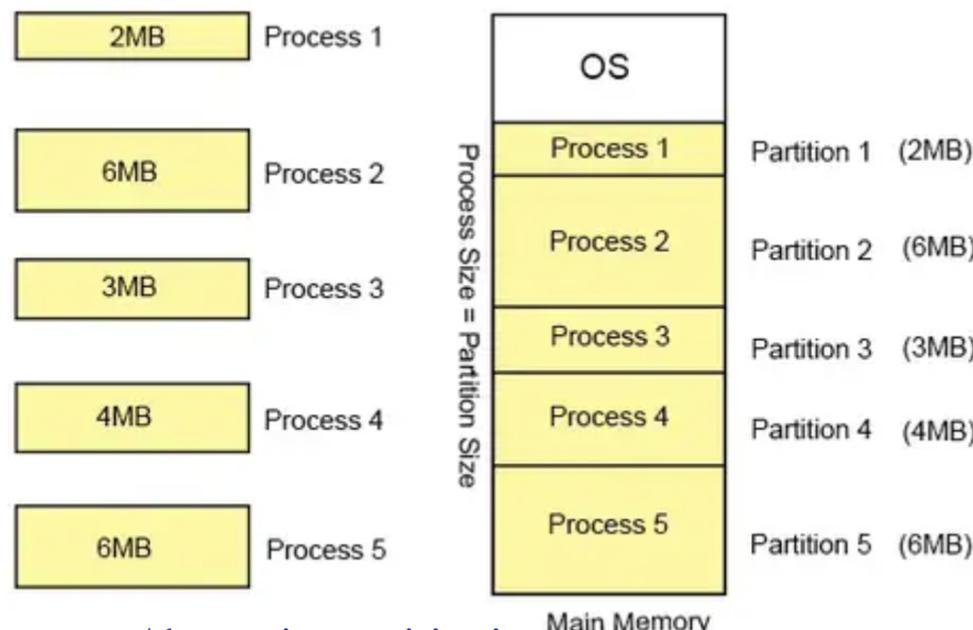
# Alocação de Memória contígua com partições variáveis

Em alguns sistemas a memória física é dividida logicamente em partições variáveis

- As partições utilizadas são de tamanho variável (adaptadas aos tamanhos dos processos) e o número de partições não é definido no momento da geração do sistema.

## Vantagens

- não há fragmentação interna
- basta criar a partição do tamanho que seja suficientemente grande para o processo

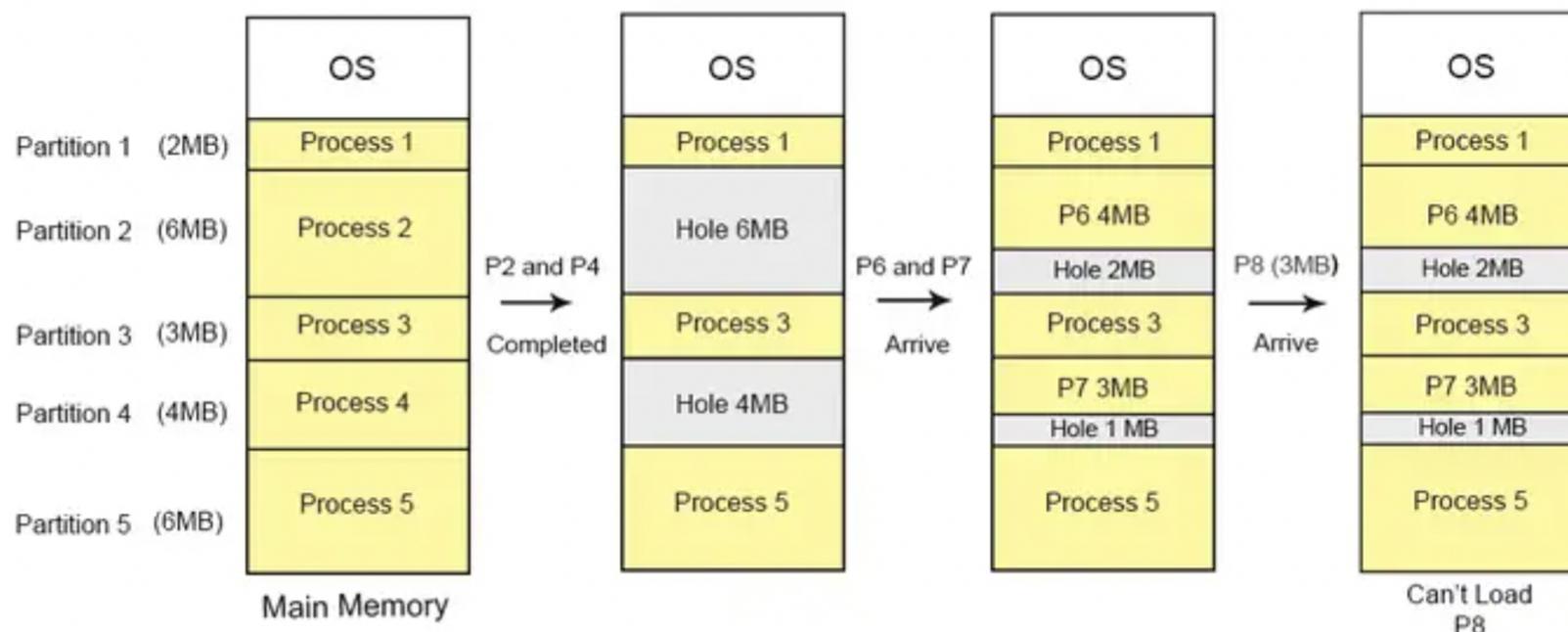


Fonte: <https://cstaleem.com/dynamic-partitioning>

# Alocação de Memória contígua com partições variáveis

Principal Problema:

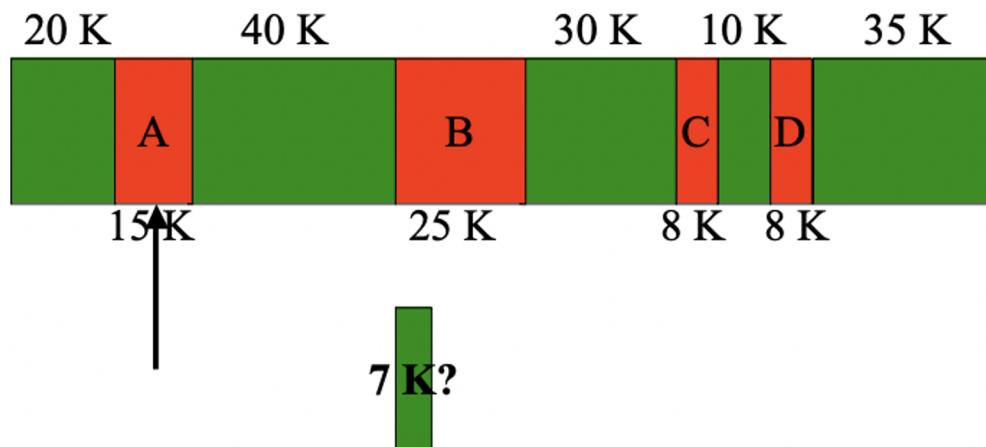
- ocorre **fragmentação externa** - à medida que carregamos e descarregamos processos, vão sendo deixados vários buracos de pequenos espaços disponíveis espalhados pela memória física



# Qual lacuna alocar a um processo que pede por um tamanho X?

Alternativas:

- FIRST-FIT: a primeira lacuna onde cabem X bytes;
  - Procura pode começar sempre no início da lista, ou a partir do último bloco alocado (“isso seria NEXT-FIT”).
- BEST-FIT: a lacuna de tamanho mais próximo a X (e maior que X);
  - Pode se aproveitar da ordenação da lista de lacunas.
- WORST-FIT: a maior lacuna disponível.
  - Bom, pois a nova lacuna criada será grande.



FIRST-Fit alocaria A  
BEST-Fit alocaria C  
WORST-Fit alocaria B

# Alocação de Memória contígua com partições variáveis

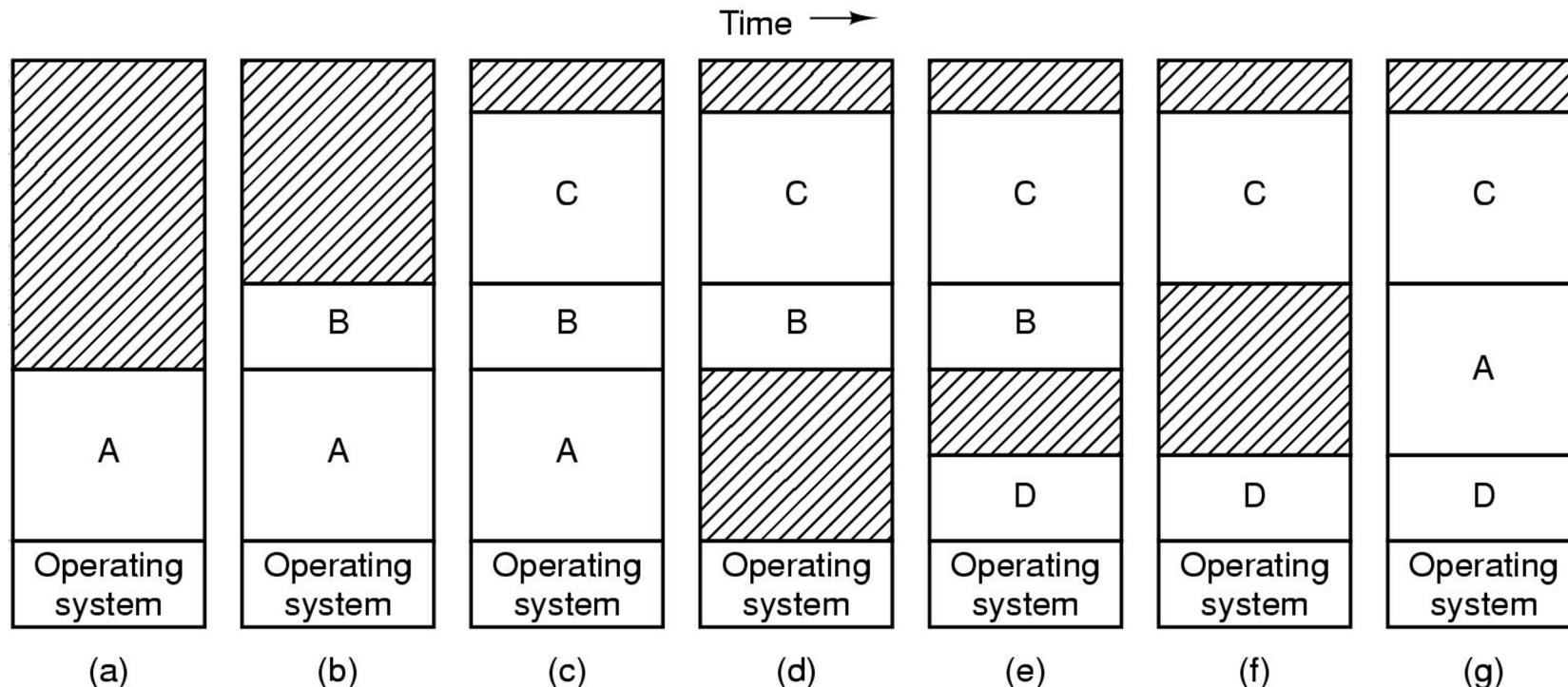


Fig.: Sequência de alocação de memória usando swapping para 4 processos e particionamento variável.

Carregamento de processos de tamanhos diferentes e em diferentes regiões de memória pode causar:

- fragmentação de memória: acúmulo de pequenas regiões de memória não utilizada
- Realocação para “desfragmentação de memória” é processo custoso

# Gerenciamento de espaços disponíveis

Idéia: dividir a memória em **unidades de alocação** de  $n$  bytes (1 KB no Minix) e representar a ocupação de (livre/ocupado) de lotes de unidades usando um *bit map* ou então uma *lista encadeada*

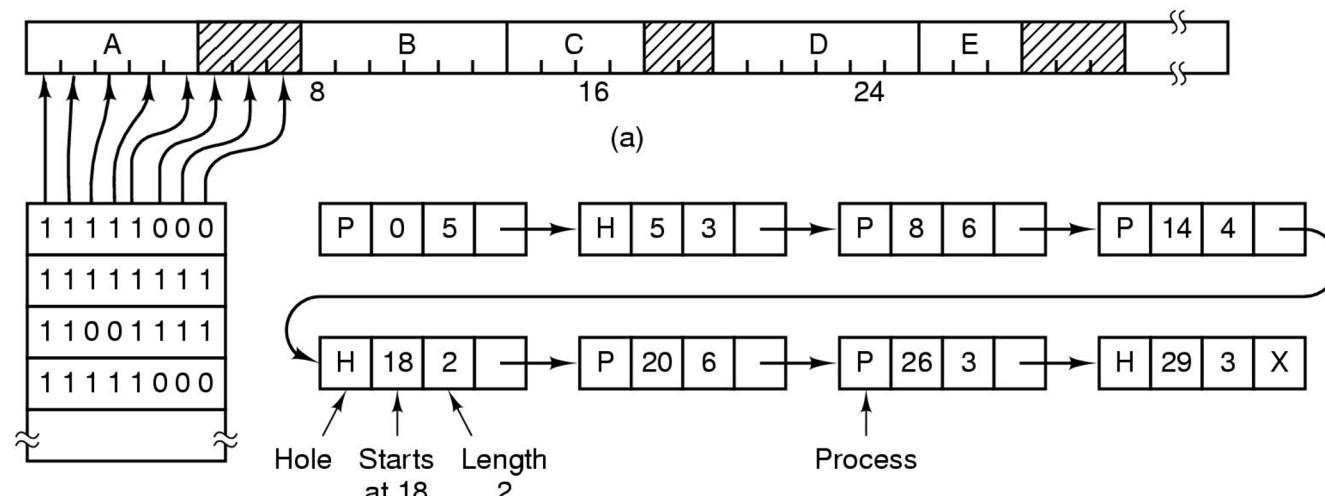


Fig: Representação da ocupação da memória com 5 processos e 3 lacunas em Bit Map ou Lista encadeada

Bit Map: armazenamento compacto e simples, mas busca por determinado tamanho de lote livre pode envolver análise de várias palavras (no bit map)

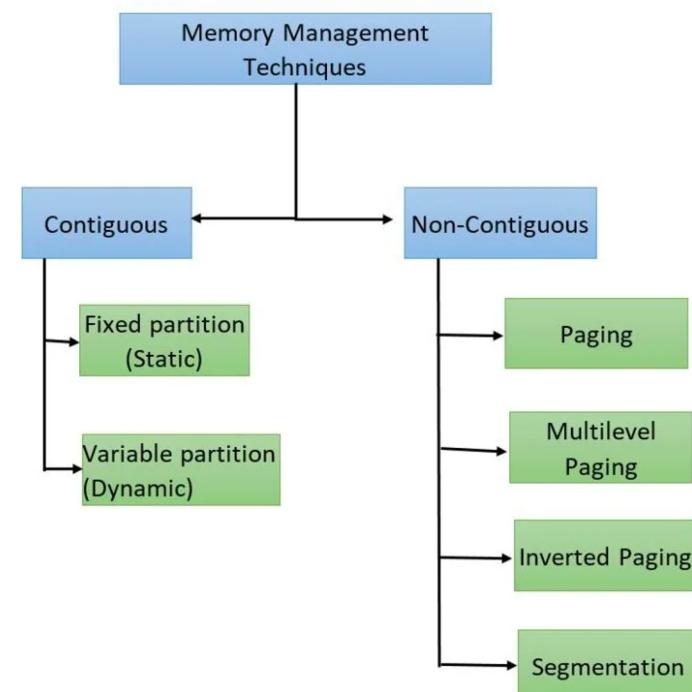
Lista ligada : cada nó contém o endereço inicial e o tamanho de uma partição ocupada ou livre

# Alocação de Memória

A necessidade de encontrar uma região contígua da memória física para abrigar um processo mostrou ter desvantagens em termos de fragmentação interna e externa.

E no caso da fragmentação externa, a questão de como encontrar um espaço livre suficientemente grande.

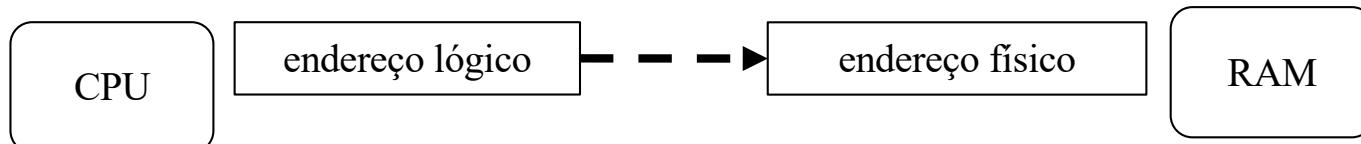
Por isso, formas de alocação de memória não contígua são interessantes: pois permitem um grau de multiprogramação ainda maior.



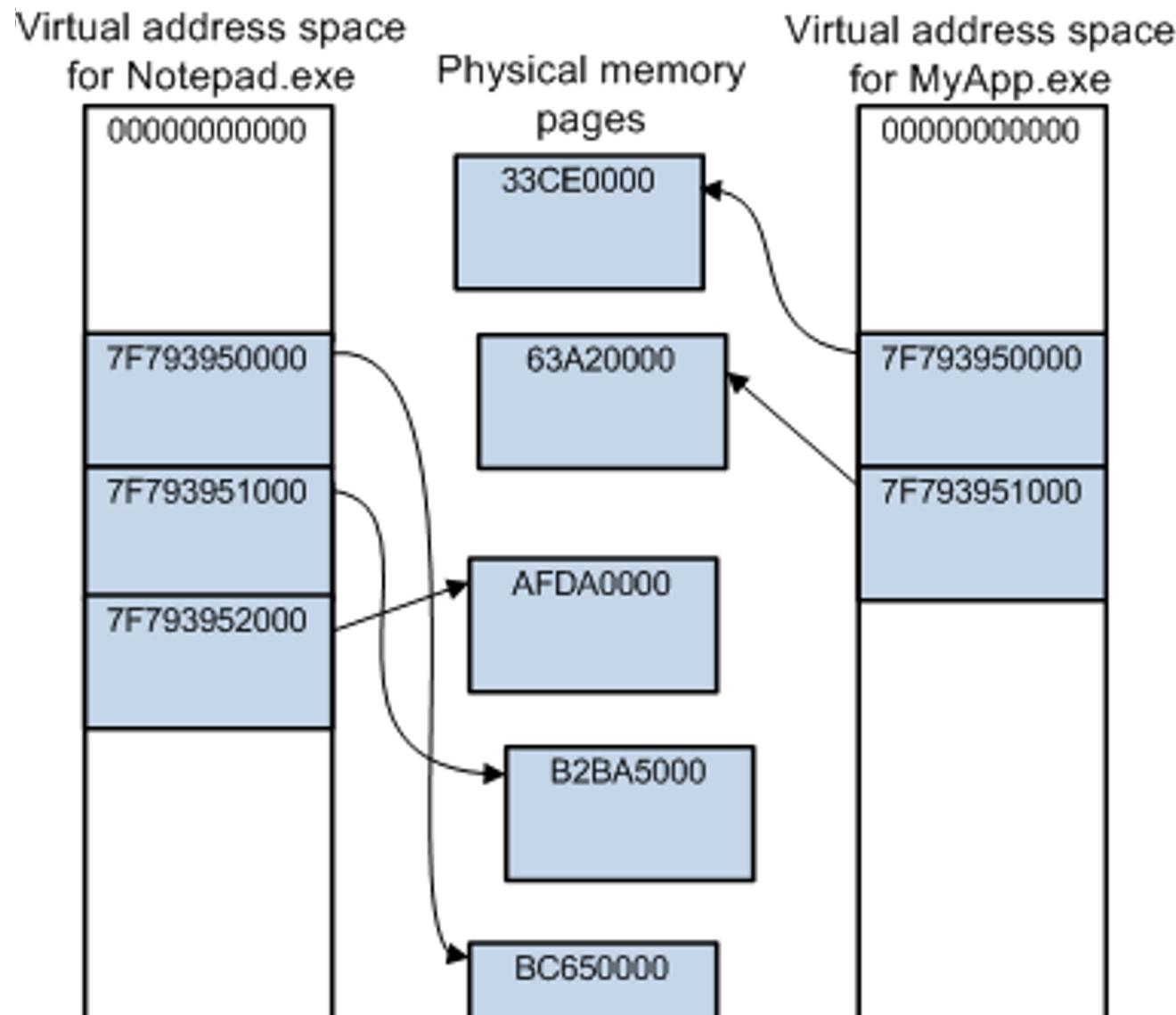
# Memória Virtual

- Motivação: A expansão dos softwares e a necessidade de multiprogramação faz com que memória física (RAM) se torne insuficiente para acomodar todos os processos
- Memória virtual é uma técnica que permite a execução de um processo sem que ele esteja completamente em memória
- Cria um espaço de endereçamento virtual dos processos que não tem correspondência com a memória RAM disponível
- Usa memória secundária (em disco) como um “buffer” para armazenamento das partes que não cabem na RAM.
- O Processador lida com um espaço de endereçamento virtual [0,lim\_max], e cada endereço lógico é mapeado para o espaço de endereçamento físico.

Mapeamento entre endereços (para cada instrução que possui uma referência à memória).



# Espaços de Endereçamento lógico/físico (virtual/real)



# Memória Virtual - Paginação

Mem.Virtual usa a técnica de paginação.

## Princípio:

Memória física e espaço de endereçamento lógico/virtual de cada processo são divididos em partições de mesmo tamanho (uma potência de 2):

- Espaço de endereçamento é dividido em **páginas**
- Memória física é dividida em **quadros de página (frames)**

Cada página "cabe" exatamente em um quadro de página.

Mapeamento de página p/ quadro é feito por uma tabela de páginas.

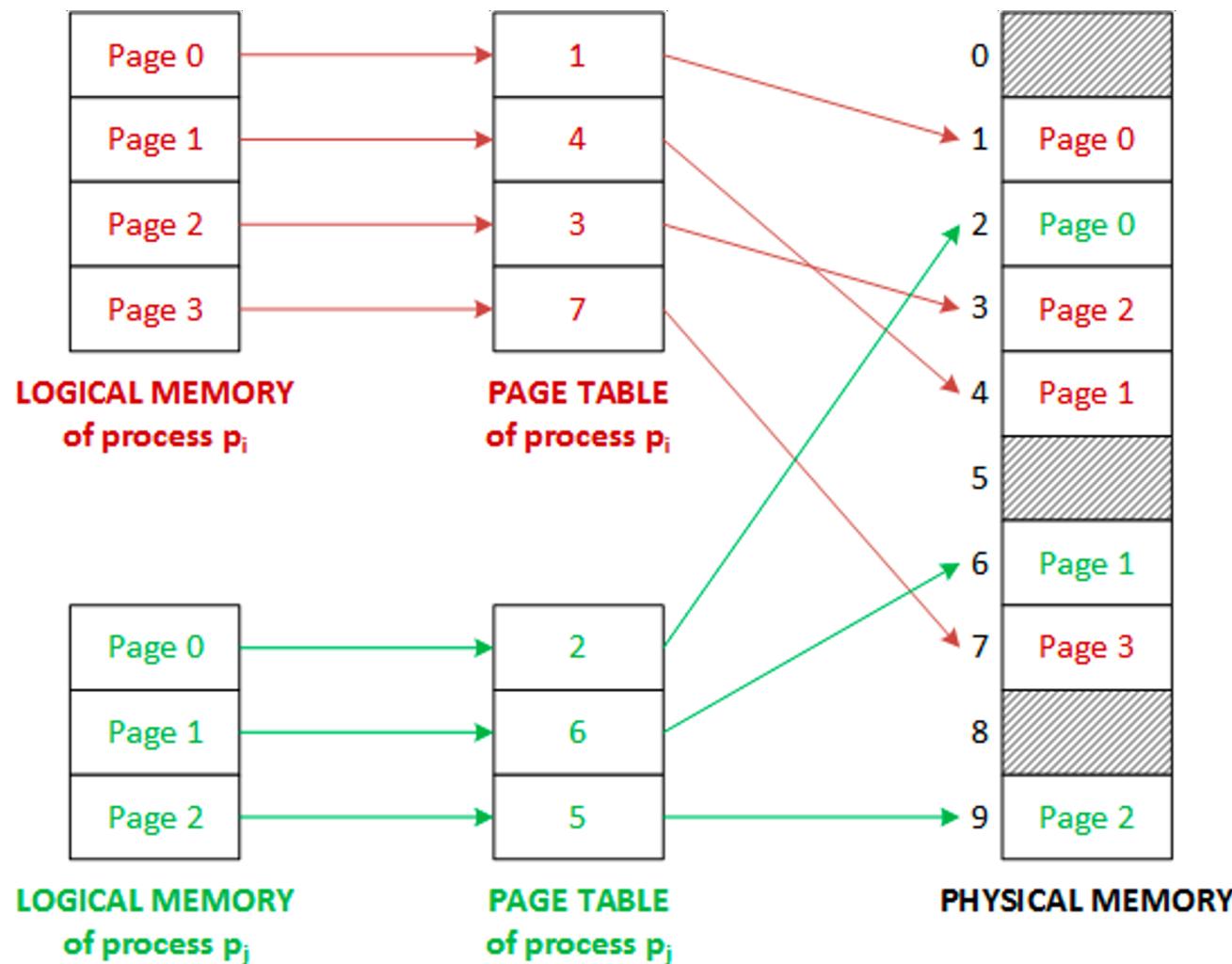
Em vez de fazer o swap in/out de todo processo, move-se páginas individualmente entre a memória principal e a área de swap

## Vantagens:

- 1) As páginas de cada processo podem ficar em regiões não consecutivas da RAM
- 2) cada processo pode não estar completamente na memória, mas apenas as páginas sendo referenciadas no momento (isso

# Paginação

TP contém mapeamento entre endereço lógico  
e endereço físico



# Paginação

- Memória física é partitionada em quadros de página (frames) de tamanho  $2^x$  bytes
- Espaço de endereçamento lógico é dividido em páginas de tamanho  $2^x$  bytes
- Cada endereço lógico é composto pelo par (número-da-página, deslocamento),  
onde **número-página** é usado como índice para uma entrada na tabela de páginas  
**deslocamento** é o "offset" do endereço dentro da página/quadro de página

Exemplo: páginas de 4 K para endereços de 32 bits:

Número-da-página	deslocamento
20 bits	12 bits

**Obs:**

- A tradução do endereço lógico para o físico é feito por um hardware dedicado (*Memory Management Unit*)
- Paginação evita fragmentação externa, mas apresenta um pouco de fragmentação interna

# Memória Virtual por paginação

Usa-se uma tabela de página (TP) para cada processo, que registra em que quadro de memória cada página está carregada:

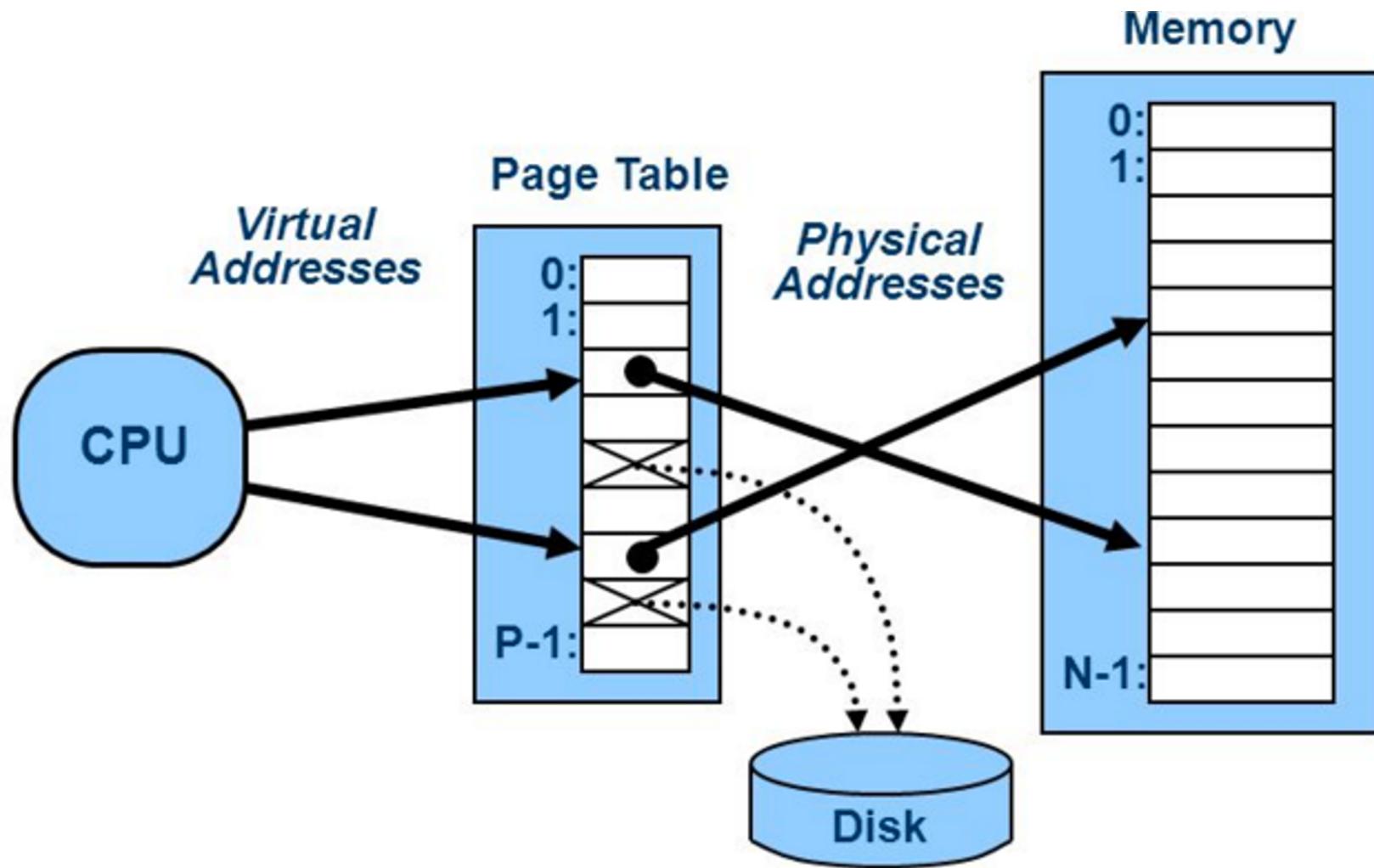
- TPs são usadas para a tradução (de end-lógico  $\Rightarrow$  end-físico)
- TPs de todos os processos precisam ser mantidas no núcleo (pois fazem parte do contexto dos processos);

Uma tabela de quadros livres registra quais quadros de página da RAM estão livres (inicialmente todos).

A cada acesso a um endereço lógico verifica-se se a página correspondente está em um quadro da memória;

- se não estiver, ocorre uma interrupção e um tratador de páginas faltantes (page-fault) acha um quadro livre e copia a página requisitada do disco para a memória.

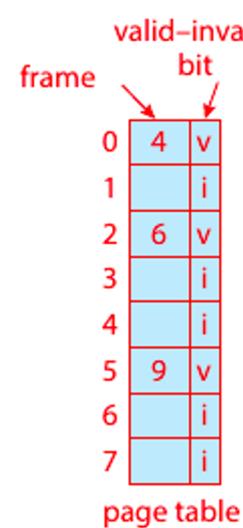
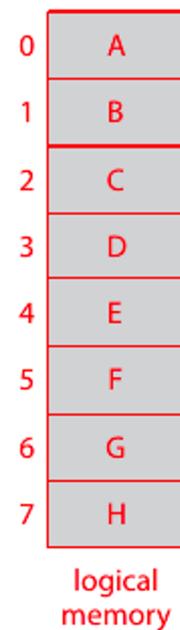
# Memória Virtual: paginação



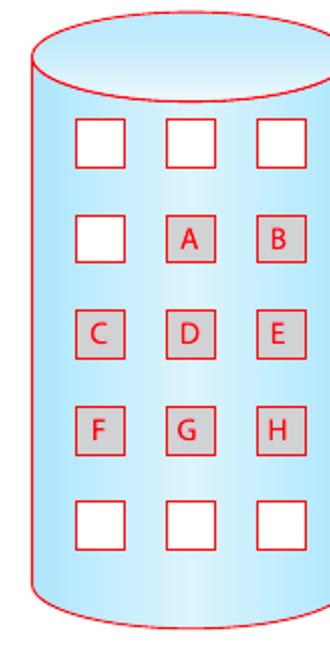
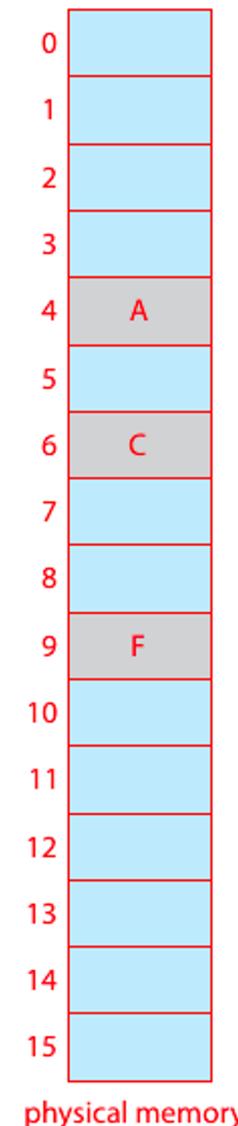
Fonte: Fabián Bustamante

# Memória Virtual

Espaço de Endereçamento lógico do processo e suas páginas

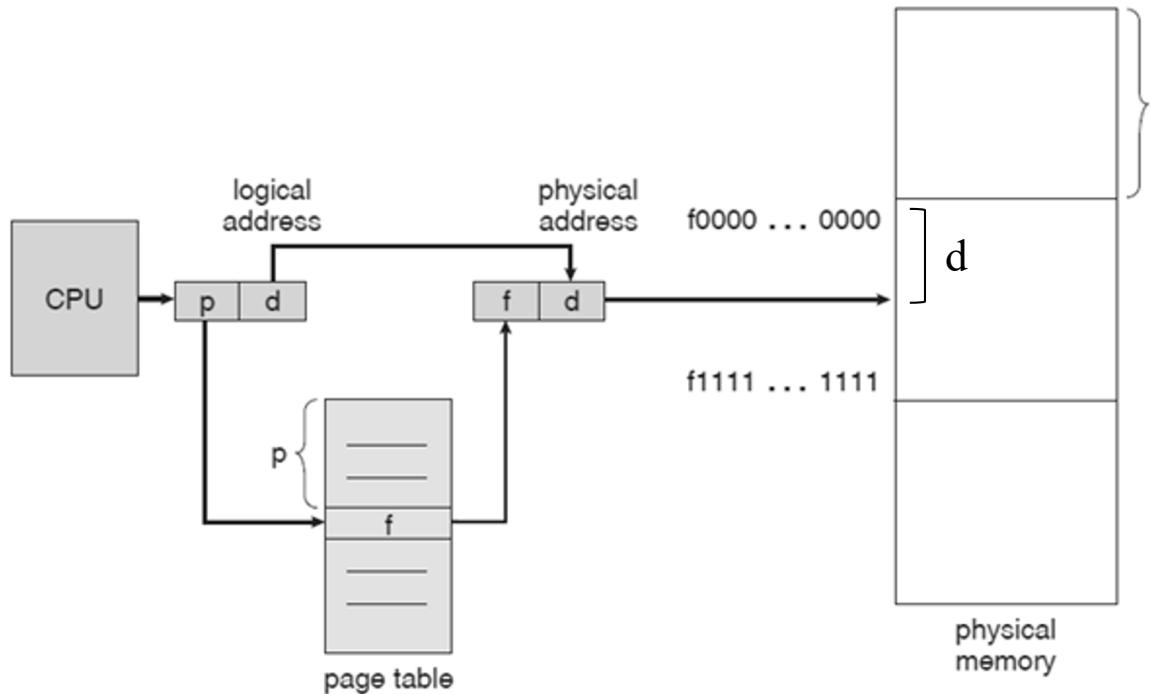


Memória física com Quadros de páginas (page frames)



Partição de Paginação (ou swapping) Paginas

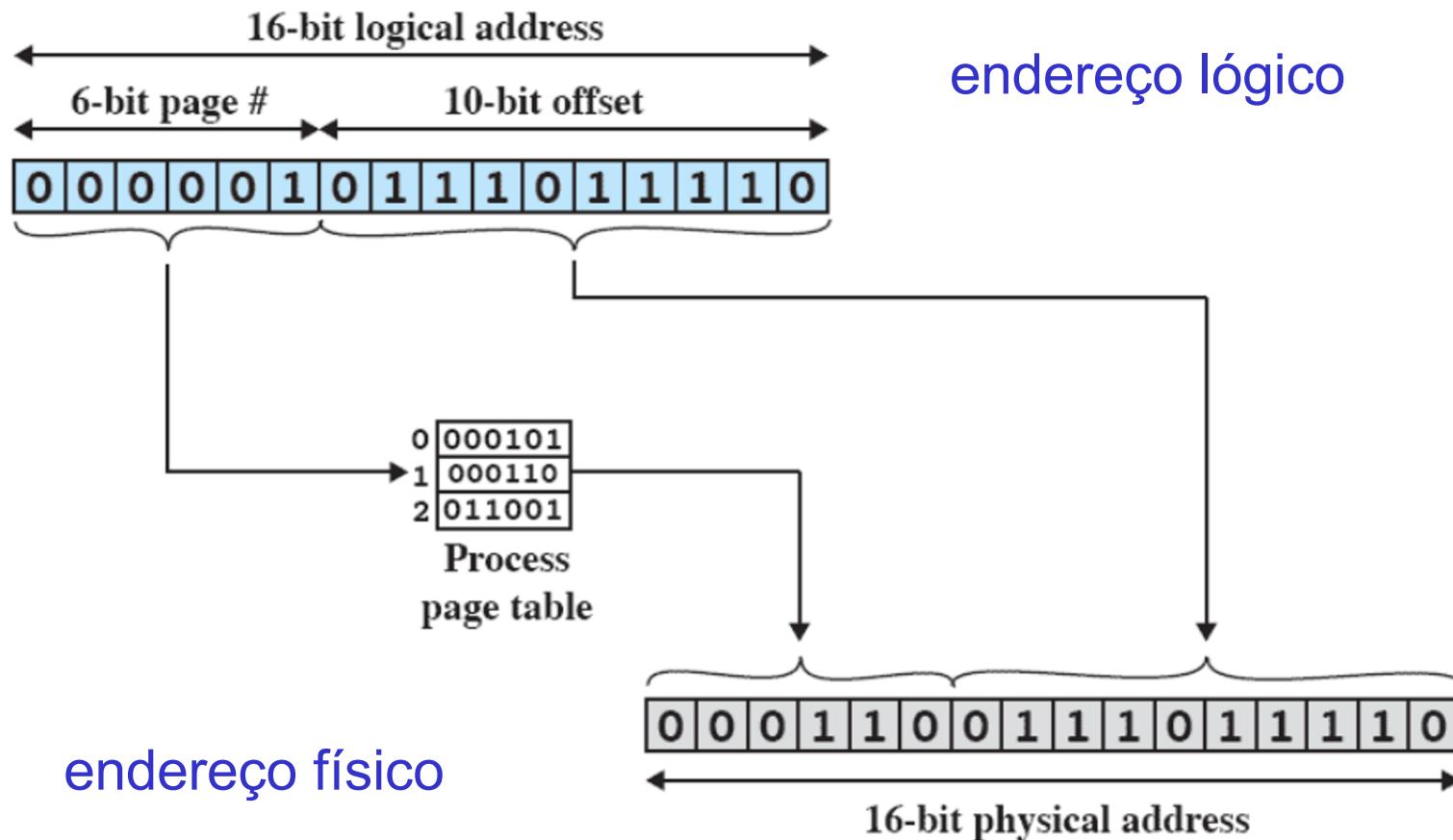
# Paginação: tradução de endereços



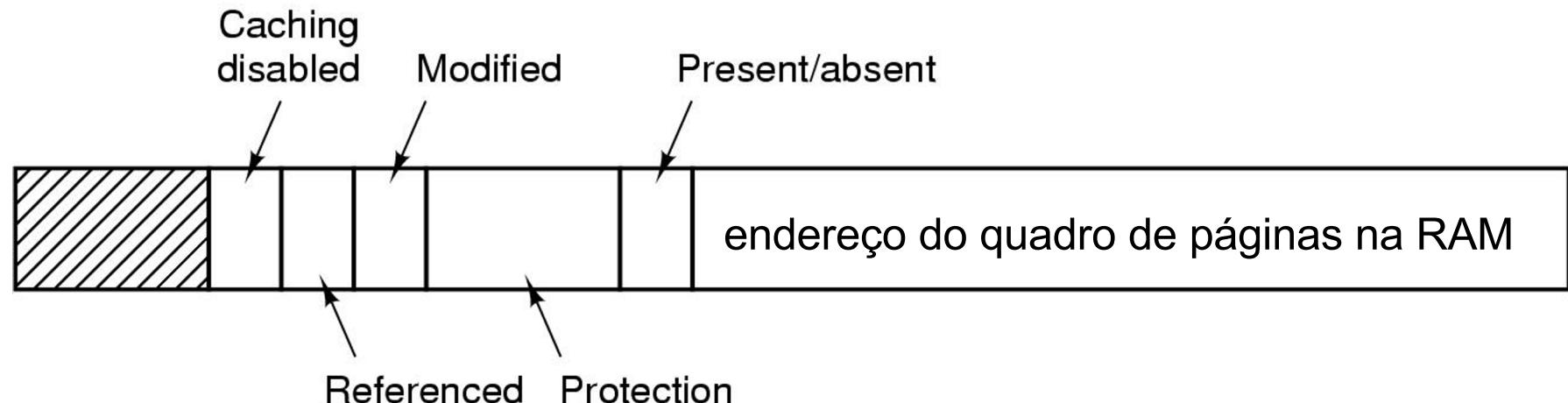
- O endereço lógico gerado é dividido em duas partes: um número de página lógica *p* e um deslocamento dentro da página, *d*.
- O número da página lógica é usado como índice no acesso à tabela
- Cada entrada da tabela de páginas possui o endereço do quadro de páginas da RAM onde cada página lógica se encontra (ou se está no disco de swapping)..
- O deslocamento do endereço dentro da página física será o mesmo deslocamento dentro da página lógica
- Basta concatenar os bits do número de página física, *f*, (obtido da tabela de páginas) com o deslocamento *d*, já presente no endereço lógico para obter-se o endereço físico (*f,d*) correspondente.

# Paginação:

## exemplo de tradução do endereço lógico p/ endereço físico



# Entrada da Tabela de Páginas



Utilidade de cada campo:

- Caching: se essa entrada da TP deve ser cacheada
- Referenced: se houve acesso a algum endereço da página no último intervalo de tempo  $\Delta t$
- Modified: se houve acesso de escrita a algum endereço da página em  $\Delta t$
- Protection: permissões de acesso (p.ex. somente leitura ou nenhum acesso), para evitar modificações acidentais ou mal-intencionadas da memória.
- Present/absent: se página está na memória RAM.
- endereço do quadro de páginas na memória física (page frame number)

# Carregamento de Páginas

Ao se carregar uma página para a memória pode-se adotar duas estratégias diferentes:

**Paginação por Demanda** As páginas são transferidas da memória secundária para a principal apenas quando são referenciadas

Traz para a memória apenas as páginas realmente necessárias à execução do programa.

**Paginação Antecipada** Usa o princípio da localidade espacial/temporal com relação a páginas já carregadas

- para pré-carregar na RAM outras páginas que provavelmente serão acessadas pelo processo nas próximas instruções

Se o programa estiver armazenado sequencialmente no disco, economiza-se tempo ao levar um conjunto de páginas

Trata-se de uma otimização, mas que tem um custo em potencial:

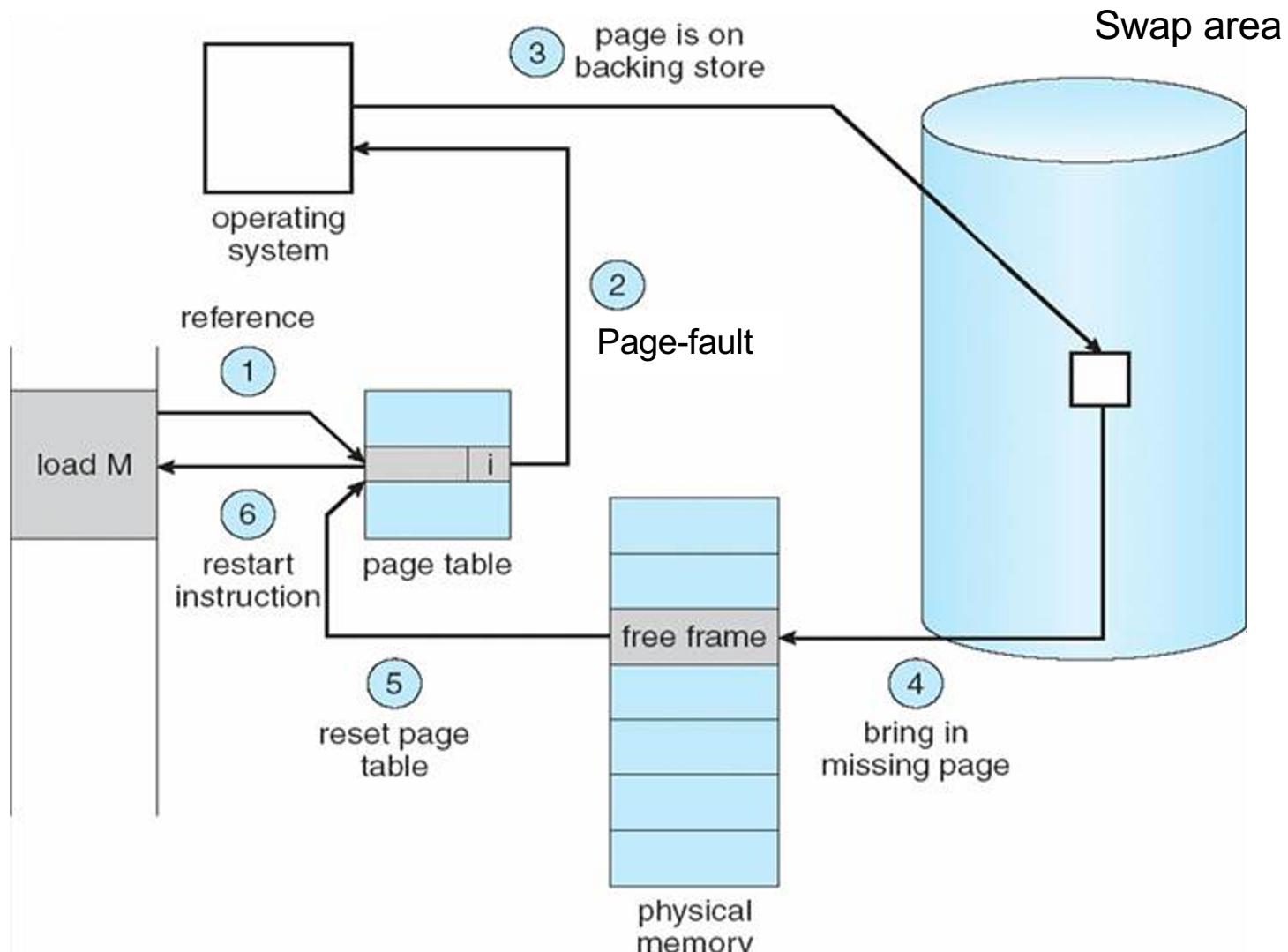
- Caso o processo não acesse as páginas transferidas antecipadamente, o sistema terá perdido tempo e ocupado memória desnecessariamente.

# Paginação sob demanda: Princípio de Localidade e Working Set

- Um processo geralmente requer muitas páginas lógicas.
- Mas, durante um período de tempo, apenas um determinado subconjunto dessas páginas é efetivamente acessado. Isso é chamado de **modelo de localidade de execução** e tem sido amplamente observado.
  - **Localidade temporal**: reuso de dados ou instruções dentro de um curto período de tempo.
  - **Localidade espacial**: reuso de dados relativamente próximos.
- O subconjunto de páginas acessadas varia ao longo de diferentes períodos de tempo, mas monitorando esse conjunto é possível saber **quantos quadros de página são suficientes para cada processo** a cada momento.
- **Working Set** é o conjunto de páginas acessadas por um processo durante um intervalo de tempo. Pelo princípio de localidade, o Working Set não é grande
- e ajuda a dimensionar o número de quadros de página necessários para o processo a cada momento.

# Paginação sob Demanda

Sequência de ações como consequência de um page fault.



Obs: page-fault é sinalizado pelo bit absent (i) na entrada da tabela de páginas. 57

# Tratamento da interrupção “Falta de página”

Interrupção causada quando um processo acessa a uma página cujo bit de validade está zero na TP:

1. Gera um trap (interrupção por SW) para o núcleo;
2. Salvamento do contexto de P, que passa a ser bloqueado;  
(Possivelmente no meio de uma instrução! )
3. O núcleo assume a CPU;
4. Identificação do IRQ (vetor de interrupções);
5. Localização do endereço da página no disco de swaping;  
Solicitação de E/S ao controlador do disco
6. Inicia-se a transferência de dados disco/RAM
7. Escalona-se outros processos.
8. Quando a E/S tiver terminado, gera uma interrupção de HW.
9. O núcleo atualiza a tabela de páginas de P.
10. O núcleo passa P do estado bloqueado para o estado pronto.
11. Ao retomar a execução, a instrução interrompida é re-executada

# Tabela de Páginas

Um aspecto importante da paginação é a forma como a tabela de páginas é implementada. (TP)

**As principais opções são:**

1) TP em registradores da CPU dedicados, que são carregados através de instruções privilegiadas (ex. DEC PDP-11)

Vantagem: não necessita de MMU e tradução é veloz

Desvantagem: número de entradas é pequeno (tipicamente 16-256)

2) TP é mantida em memória principal (do núcleo)

- mantém-se dois registradores por processo:

o PTBR com o endereço inicial da tabela, e o PTLR com o tamanho da tabela de página

Vantagem:

- a troca de contexto envolve somente PTBR e PTLR

Desvantagens:

- só funciona quando as tabelas de páginas não são muito grandes
- toda referência à memória virtual causa dois acessos à memória física: um para acessar entrada desejada da Tabela de Páginas, e outro de acesso à página desejada.

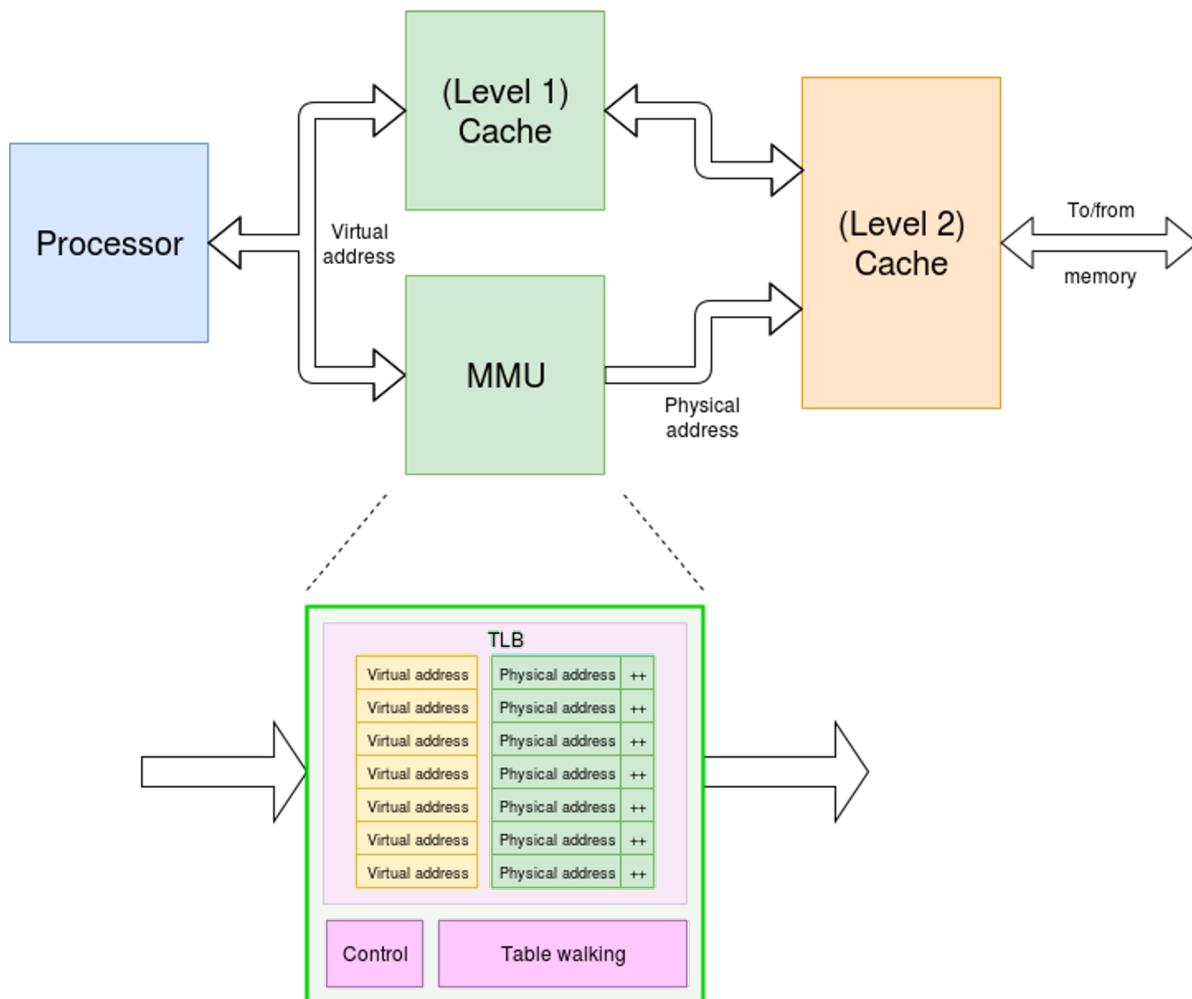
# Tabela de Páginas

Para reduzir o tempo médio de acesso à memória (problema do duplo acesso à memória à TP e ao endereço virtual em si)

3) utilização de um cache para a tradução de endereços, chamado **Translation Lookaside Buffer (TLB)**:

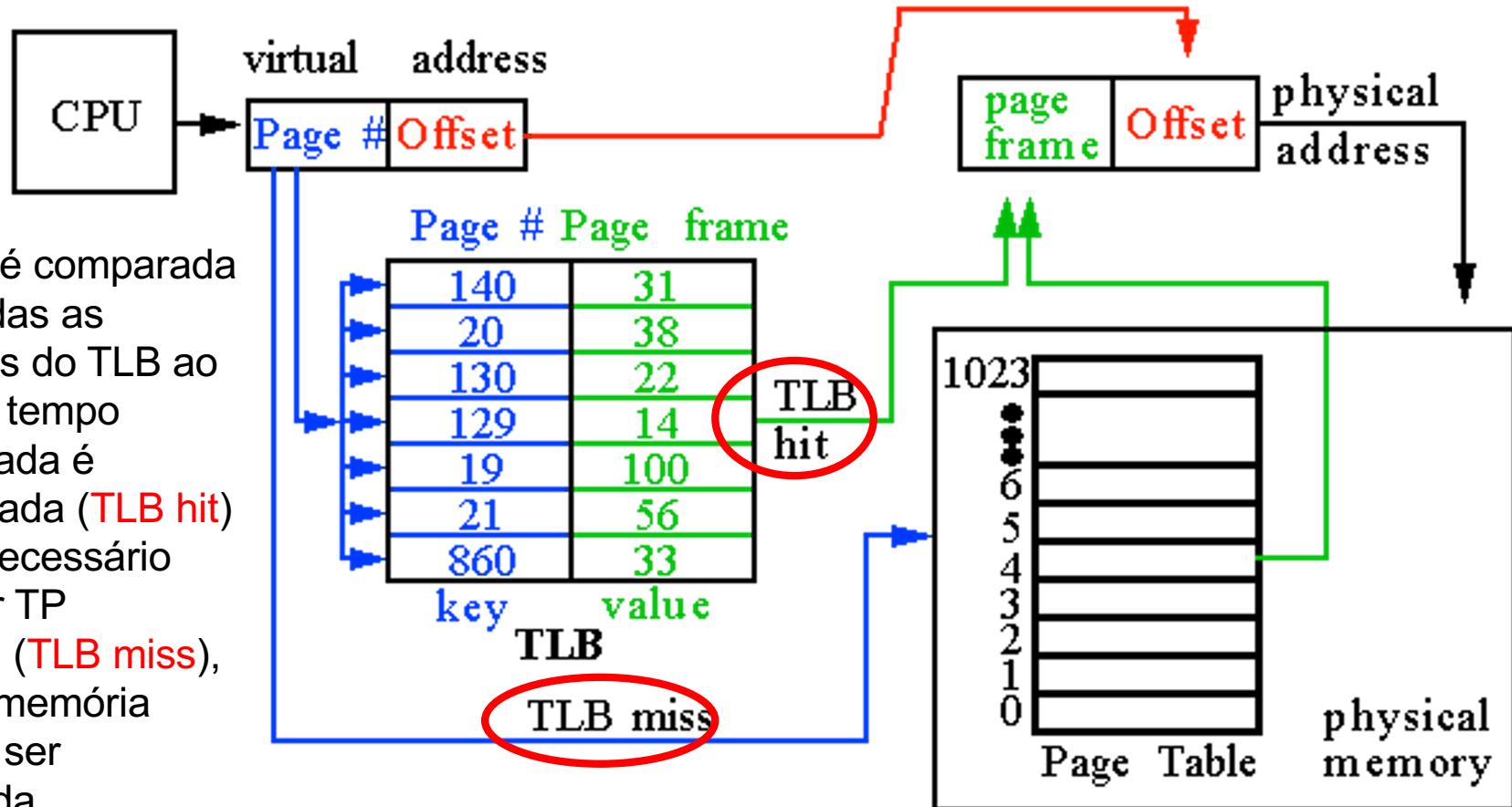
- Utiliza o esquema de mapeamento associativo (muito veloz)
- É implementado na MMU
- Funciona como uma memória cache de entradas da Tabela de Página,
- Mantém apenas as traduções dos endereços virtuais das páginas mais recentemente referenciadas.
- usa o princípio da localidade
- Não necessariamente requer do acesso à tabela de páginas.
- Quando uma página lógica está no TLB (ocorre *TLB hit*). e se não está no TLB (ocorre *TLB miss*):
  - apenas no *TLB miss* acessa-se a TabPaginas e copia-se a entrada para o TLB
  - se a página está em memória, operação é completada; senão gera-se uma interrupção de *Page-fault*

# O Translation Look-aside Buffer na MMU



# Translation Look-aside Buffer (TLB)

- Página é comparada com todas as entradas do TLB ao mesmo tempo
- Se entrada é encontrada (**TLB hit**) não é necessário acessar TP
- Se não, (**TLB miss**), TP em memória precisa ser acessada.



Razão de TLB hits depende do tamanho do TLB e da estratégia de troca de entradas. O quanto maior for o número de entradas no TLB, mais eficiente será o acesso à memória.

A razão de (*TLB hits/acessos à memória*) determina a **eficiência da tradução** e é usada para calcular o tempo médio de acesso à memória

# Lidando com muitas tabelas de páginas grandes

- Principal problema é que a TP de cada processo precisaria estar em RAM.
- Se o tamanho da TP for grande e com muitas entradas (e.g. 1K entradas em cada TP) isso pode tornar inviável manter todas elas na memória.
- O que fazer?
  - => paginar a própria tabela de páginas
  - => criar um esquema hierárquico de paginação

# Tabela de Páginas de 2 níveis

Resolve o problema de manter grandes tabelas de página na memória (para todos os processos).

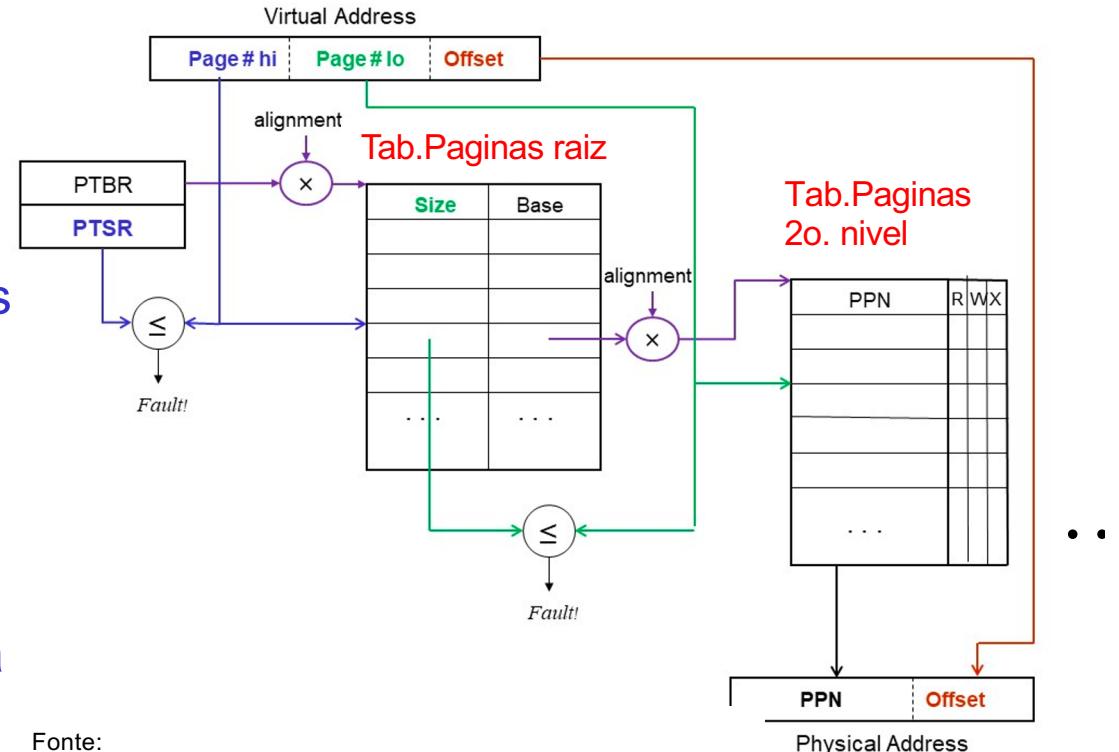
Ideia: A própria tab. de páginas é paginada (guarda apenas as partes da TabPagina cujo mapeamento esteja sendo usado).

Dois níveis: tabela de páginas raiz (de 1o. nível) e TabelaPágina de 2o. nível em páginas

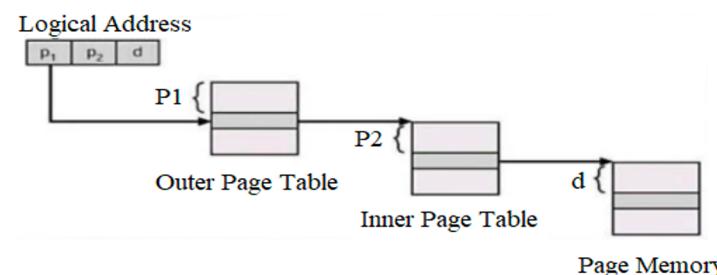
A tabela raiz contém os ponteiros para o início dessas partes da TabelaPáginas de 2o nível.

O page #hi é a entrada na tabela raiz

O page #lo é o deslocamento na tabela de páginas de 2o. nível



Fonte:  
<https://pages.cs.wisc.edu/~bart/537/lecturenotes/s16.html>



# Tabela de páginas Invertidas

Para arquiteturas de 64 bits, o espaço de endereçamento lógico é  $2^{64}$  bytes! Mesmo se o tamanho de página é 4KB, isso leva a TPs muito grandes, e.g.  $2^{52} \approx$  alguns Tera Bytes!).

Por isso, em alguns sistemas usa-se uma TP invertida:

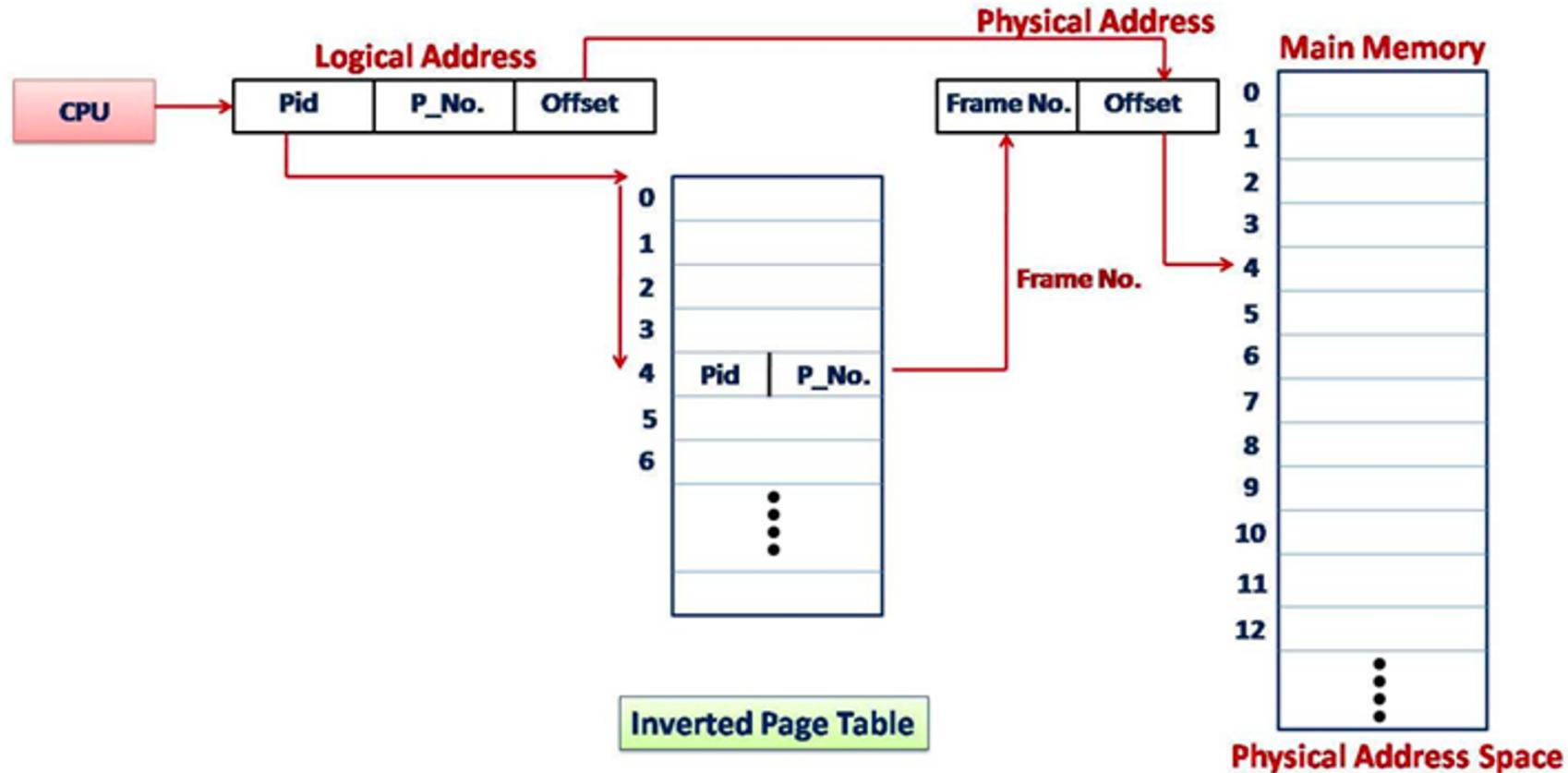
- Uma entrada p/ cada quadro de página (da memória física)
- Cada entrada contém uma lista de:  
(processID, Presence Bit, número da página)

Desvantagem: a tradução de endereços lógicos -> físicos fica bem mais lenta, i.e. para encontrar um par (Processo P, página p), precisa-se fazer uma busca na lista da TP invertida para um determinado quadro.

O gerenciamento dessas filas é complexo. A cada vez que uma página é copiada para outro frame, a tabela invertida precisa refletir isso nas listas

Só é viável, se o TLB for usado para guardar os mapeamentos recentes. Somente quando ocorre um TLB miss, então a TP invertida precisa ser pesquisada.

# Tabela de Páginas Invertidas



Se página não está no TLB, então precisa-se percorrer a tabela invertida procurando pelo PID e índice da página (P\_No). Por isso, usa-se um hash para indexar as entradas da TP invertida.

# Segmentação

- É a técnica de gerência de memória onde o espaço de endereçamento virtual é dividido em segmentos de **tamanhos diferentes**.
- Um segmento pode ser um procedimento, um vetor/matriz, um módulo, uma biblioteca ou a pilha, etc.
- Cada segmento tem o seu **controle de acesso** e é **carregado contiguamente na RAM**.
- Na Paginação, o espaço lógico do processo é dividido em páginas de igual tamanho, independentemente do que se encontra no interior das páginas. Isso diminui a eficiência do acesso a código e dados.
- Cada segmento é descrito por um *descritor de segmento*, que contém o endereço físico inicial e seu tamanho
- A cada acesso a memória a MMU verifica se o acesso está válido (dentro do limite e no modo de acesso prescrito)

# Segmentação

Segmento é uma parte do espaço de endereçamento virtual que contém um único tipo de dados e do qual o programador está consciente.

Segmentação cria um espaço de endereçamento lógico multi-dimensional de segmentos que podem crescer e diminuir independentemente.

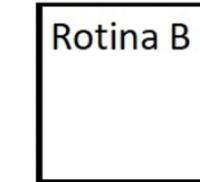
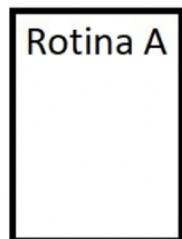
Exemplo: um compilador pode ter segmentos de árvore sintática, tabela de símbolos, código executável de cada fase da compilação, etc.

## Vantagens da segmentação:

- facilita definir permissões de acesso específicos para cada segmento
- segmentos podem crescer independentemente (tabelas dinâmicas, pilhas de threads)
- facilita o compartilhamento de partes do programa (p.ex. shared libraries)
- tabela de segmentos é sempre muito menor do que uma tabela de páginas, permitindo maior eficiência na tradução

Exemplos de segmentos:

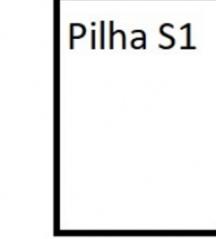
Código



Dados:



Pilha

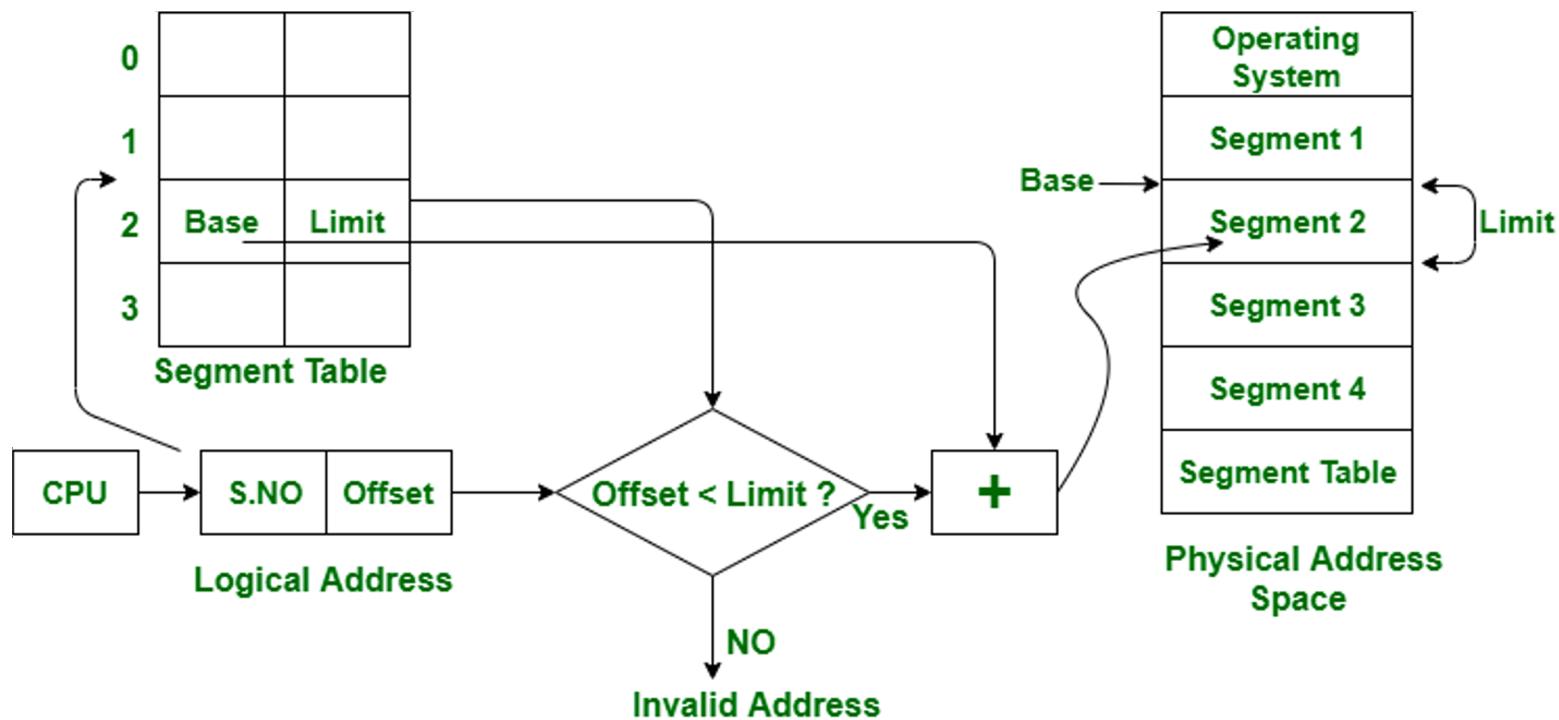


Pode ser compartilhado

Pode ser lido  
por outros

Só pode ser usado  
pelo S.O.

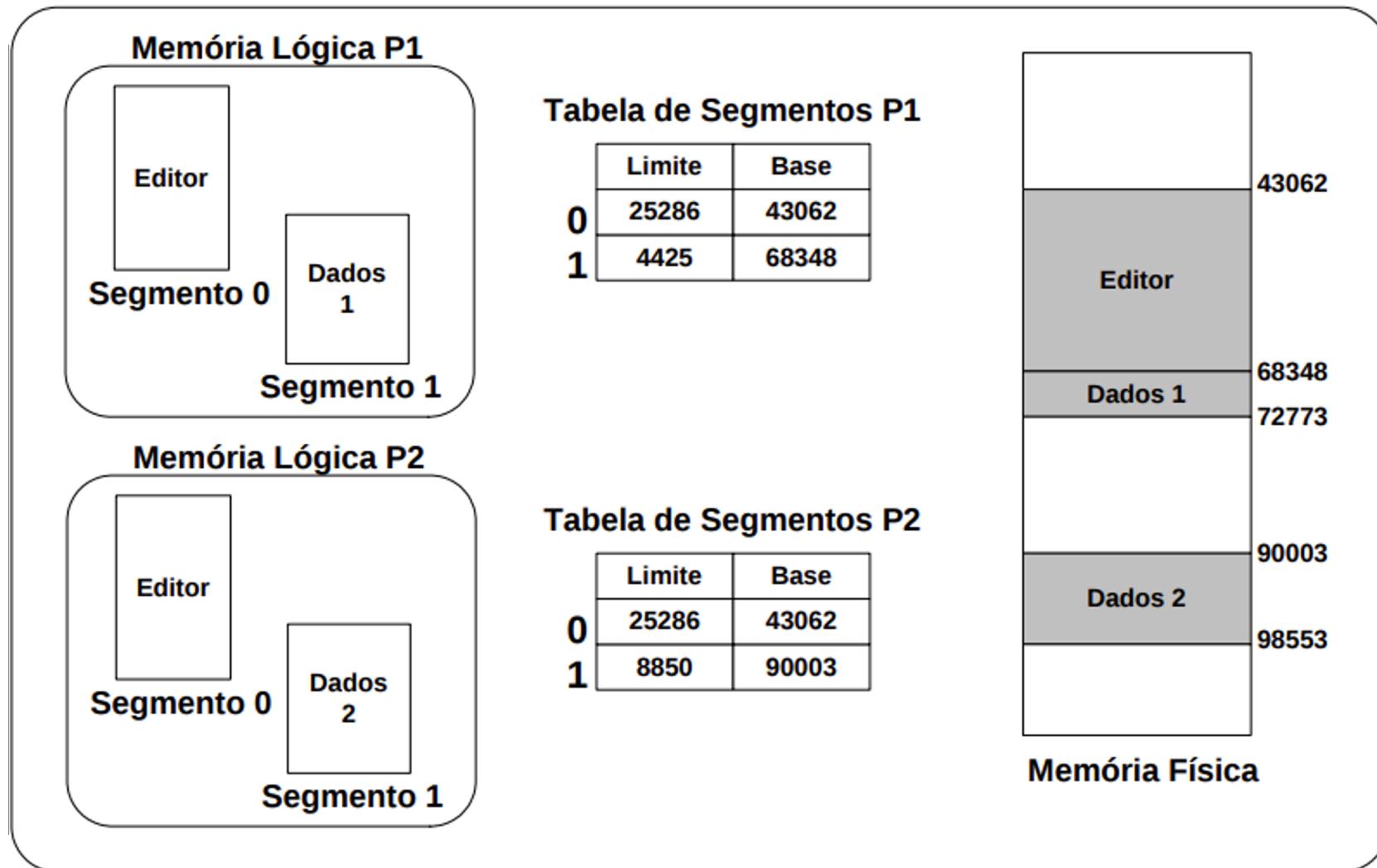
# Segmentação: tradução de end. lógico para físico



Principal diferença com relação a paginação: é verificado se o offset está dentro do Limit

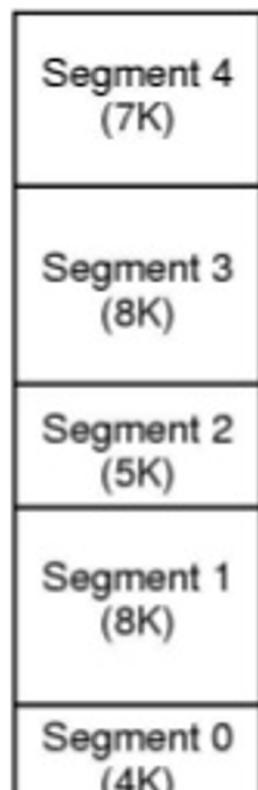
# Segmentação

Facilita compartilhamento de segmentos

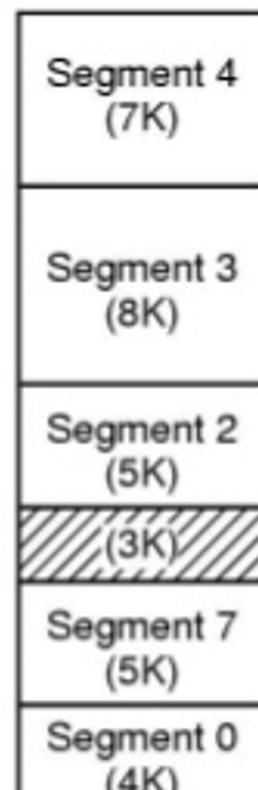


# Exemplo de Segmentação pura

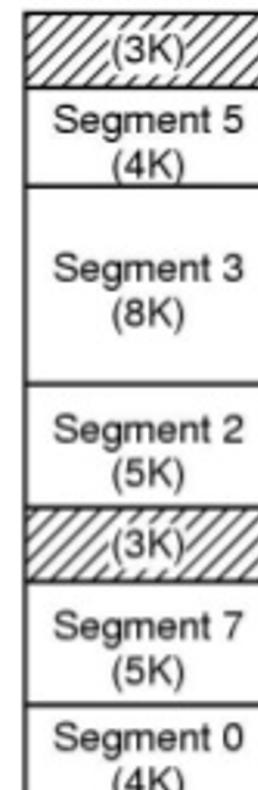
- É raro na prática, pois causa fragmentação externa.



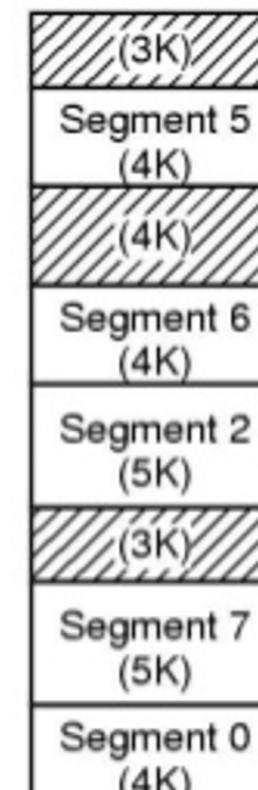
(a)



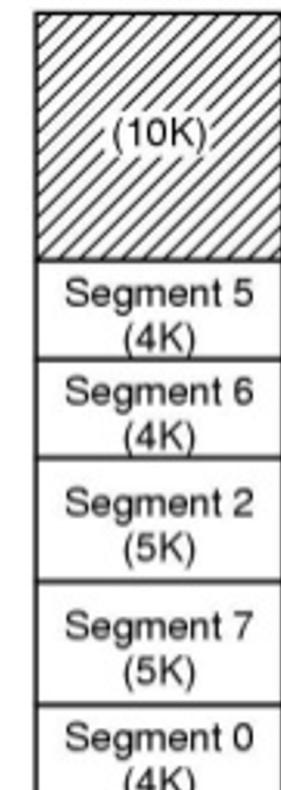
(b)



(c)



(d)



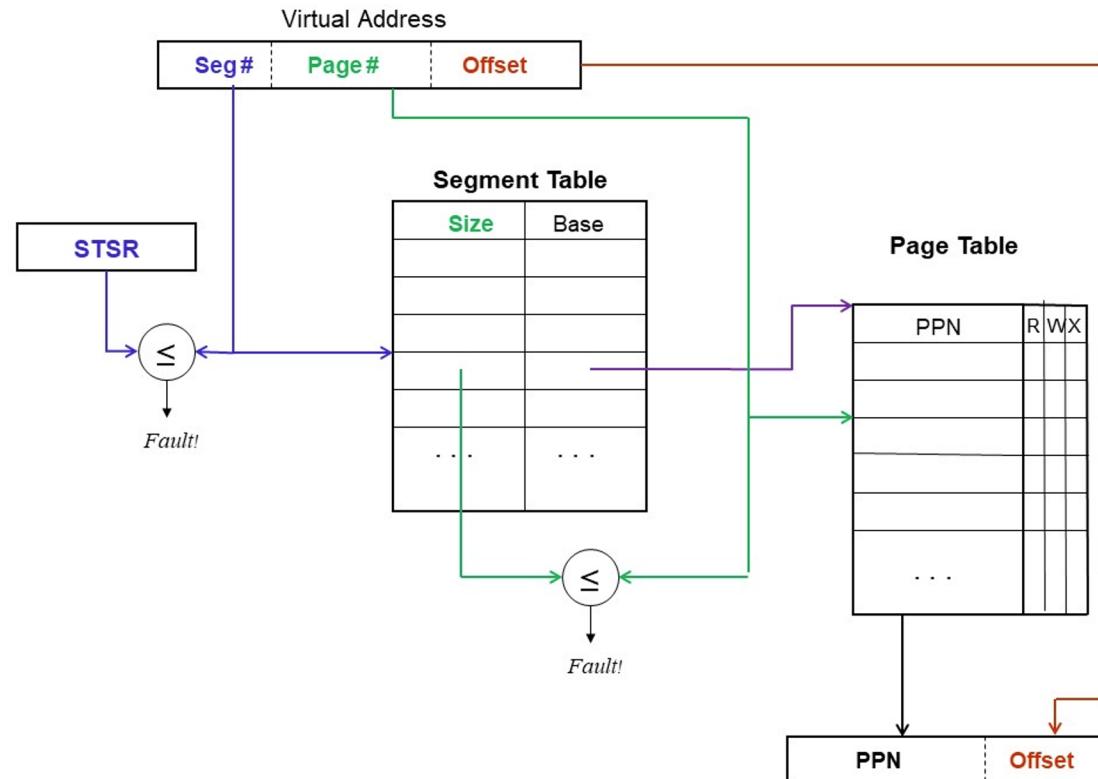
(e)

# Segmentação com Paginação

Na maioria dos sistemas atuais, combina-se segmentação com paginação. Para cada segmento é alocado em certo conjunto de páginas da memória virtual.

Agora, segmentos não precisam mais estar contíguos na RAM.

É como se Seg # fosse o índice para a tabela de Página raiz da Paginação em 2 níveis.



# Comparação entre Paginação e Segmentação

Consideração	Paginação	Segmentação
O programador precisa estar ciente de que essa técnica está sendo usada?	Não	Sim
Quantos espaços de endereçamentos lineares existem?	Um	Muitos
O espaço de endereçamento total pode exceder o tamanho da memória física?	Sim	Sim
Os procedimentos e os dados podem ser diferenciados e protegidos separadamente?	Não	Sim
As tabelas com tamanhos variáveis podem ser acomodadas facilmente?	Não	Sim
O compartilhamento de procedimentos entre usuários é facilitado?	Não	Sim
Por que essa técnica foi inventada?	Para fornecer um grande espaço de endereçamento linear sem a necessidade de comprar mais memória física	Para permitir que programas e dados sejam quebrados em espaços de endereçamento logicamente independentes e para auxiliar o compartilhamento e a proteção

# Algoritmos de Substituição de Páginas

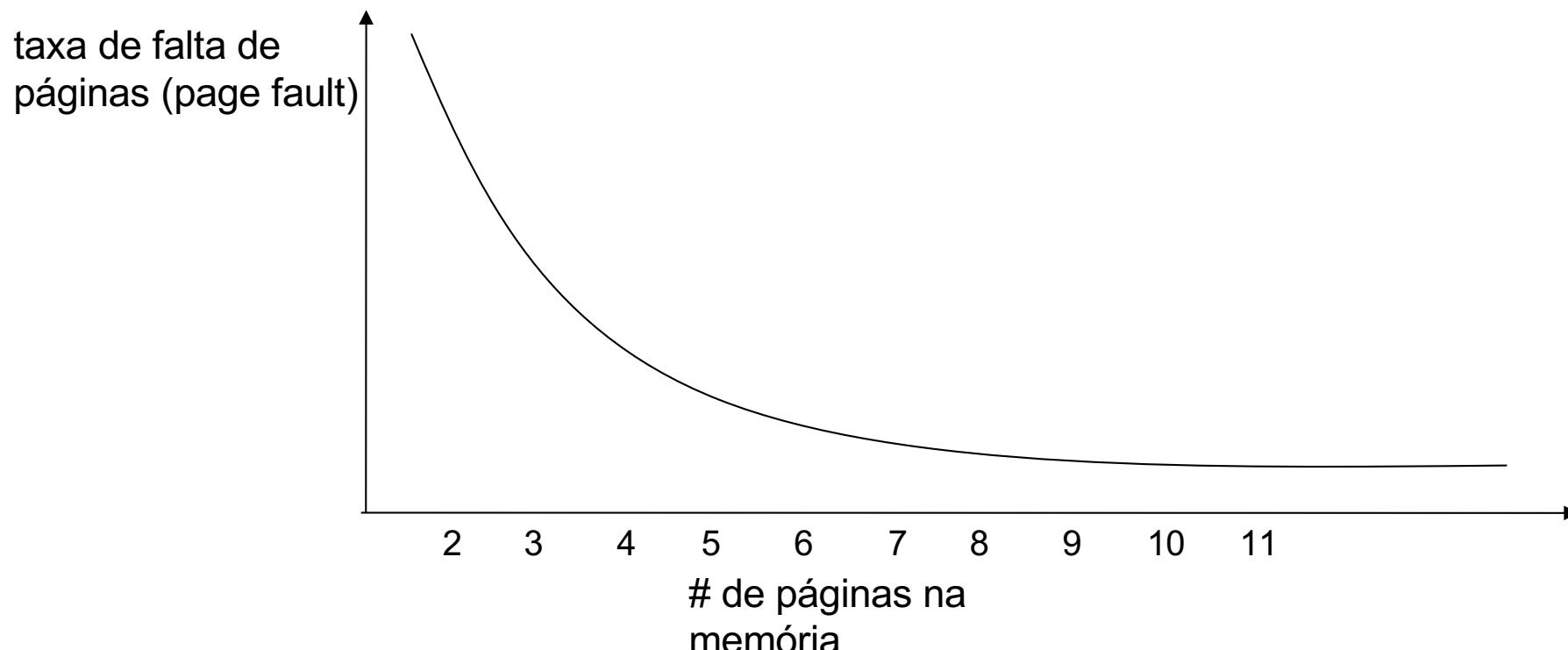
A cada page-fault:

- o gerenciador de memória aloca um quadro de páginas para a página requisitada e
- se nenhuma estiver livre (geralmente!) precisa escolher qual página deve ser removida da memória.
- O objetivo geral é reduzir a taxa de page faults de todos os processos
- selecionando a “melhor página vítima para remover”, aquela que provavelmente não será usada no futuro imediato

# Ultra-paginação (“Thrashing”)

Thrashing ocorre quando um - ou mais processos - começam a gerar *page faults* em uma frequência muito alta tornando execução de todo sistema mais lenta,

- o gerenciador de memória fica sobrecarregado com transferência de páginas entre memória RAM e disco (swap).
- alta taxa de faltas de página ocorre quando um processo possui muito menos páginas na memória do que o seu working set.



# Algoritmos de Substituição de Páginas

O algoritmo de substituição de páginas determina *qual* página do processo (ou de todos os processos na memória) é o melhor a ser swapped out e

- usa-se informações sobre acessos contidas nas tabelas de páginas para cada índice de página (o deslocamento é irrelevante)

Nessa escolha de página, deve-se considerar que:

- uma página modificada precisa ser escrita na memória secundária (swapping), mas uma página não-modificada pode ser sobreescrita no quadro de página
- não é boa escolha tirar uma página que “está em uso”, pois ela provavelmente terá que ser carregada para a memória em breve

# Substituição de Páginas

Duas classes de políticas de substituição de páginas

## Políticas Globais

- /> Todas as páginas alocadas a RAM são consideradas candidatas dentro de um pool comum de páginas substituíveis.
- /> Qualquer página, de qualquer processo, pode ser escolhida.  
Exemplos: OPT, NRU, FIFO, Segunda Chance, LRU, Aging, Relógio, PFF

## Políticas Locais

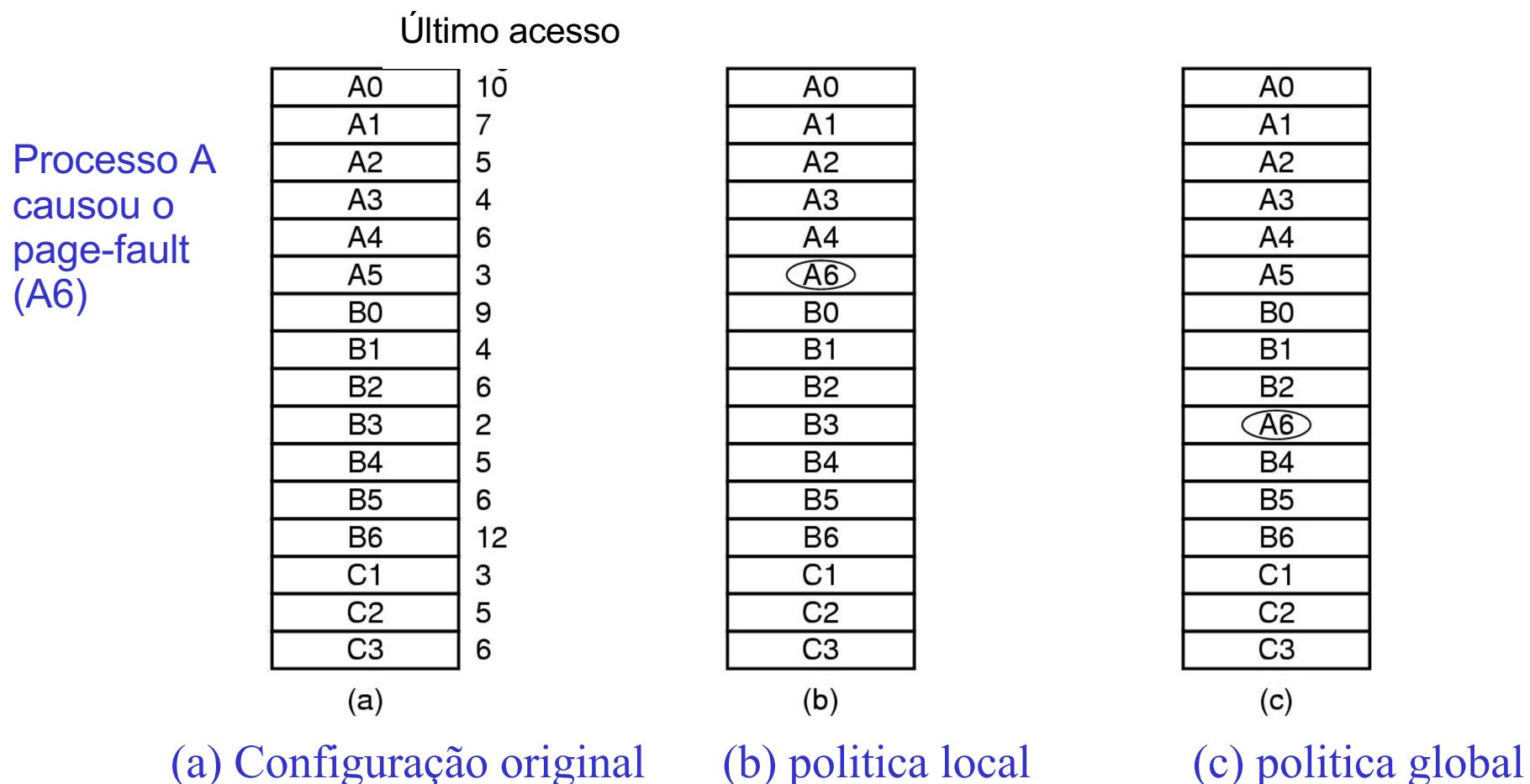
- /> Apenas as páginas do processo que gerou o page fault podem ser candidatas a substituição. As páginas dos demais processos não são consideradas para substituição

Exemplo: Working Set

- /> Para cada processo, monitora-se um conjunto de páginas em uso (“working set”)
- /> A página a ser substituída será do working set do processo que gerou o page-fault, acessada no passado mais remoto

# Política de Substituição de Páginas: Local vs. Global?

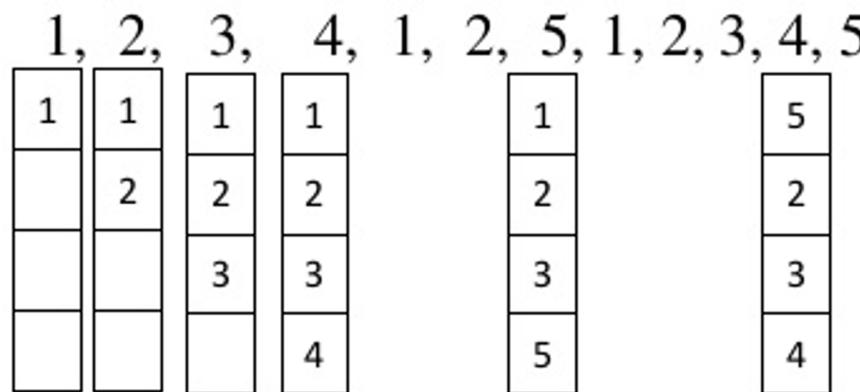
- O número de quadros de páginas alocados a cada processo deve variar?
- A página a ser substituída deve ser do processo que causou a falta de página (pol. local), ou de qualquer processo (pol. global)?



# OPT: Algoritmo de substituição ótimo (Belady)

- Substitui a página que será acessada no futuro mais distante
  - Não é viável na prática, pois exigiria um conhecimento sobre todos os acessos futuros a endereços

Seq. de referência  
(índices de página)



- Por isso, o OPT é usado apenas como referência para avaliar o quanto os algoritmos concretos estão distantes do ótimo

# Algoritmo Página não recentemente Utilizada (Not Recently Used - NRU)

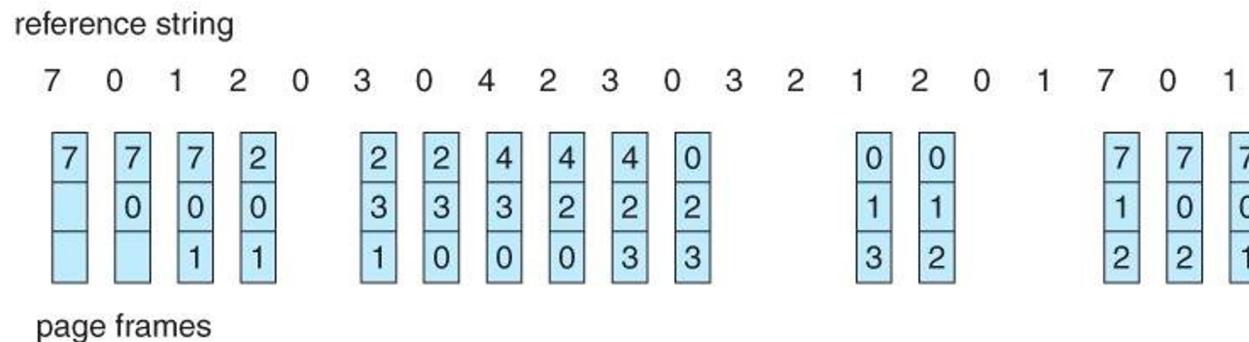
- Usa o bit de Referência (R) e de Modificação (M) de cada página
  - Página é carregada com permissão somente para leitura e com bit M=0.
  - No primeiro acesso para escrita, o mecanismo de controle de acesso na MMU notifica o núcleo, que seta M=1, e troca para permissão de escrita
  - Periodicamente, seta-se bit R=0
- Páginas são classificadas em 4 categorias
  - 1) Não referenciada (R=0), não modificada (M=0)
  - 2) Não referenciada (R=0), modificada (M=1)
  - 3) referenciada (R=1), não modificada (M=0)
  - 4) referenciada (R=1), modificada (M=1)
- Substitui-se qualquer página na menor categoria não vazia (procura-se na ordem crescente de categoria)
- Motivo da prioridade de categoria #2 sobre #3: É melhor manter na memória as páginas sendo referenciadas do que modificadas mas pouco referências.

# Algoritmo FIFO de substituição

Mantém uma fila de todas as páginas na ordem em que foram carregadas na memória

Na ocorrência de falta de página:

- página no começo da fila é substituída, e nova página adicionada no final da fila



Simulação de referências a páginas  
e comportamento da política FIFO

**Principal Vantagem:**

- Simplicidade de implementação.

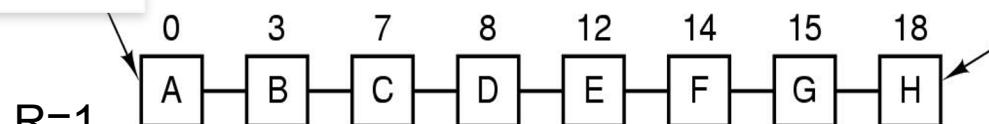
**Principal Desvantagem:**

- Não leva em conta se uma página está sendo frequentemente acessada ou não.
- Por exemplo: a página mais antiga (a ser descartada) pode estar sendo acessada com alta frequência.

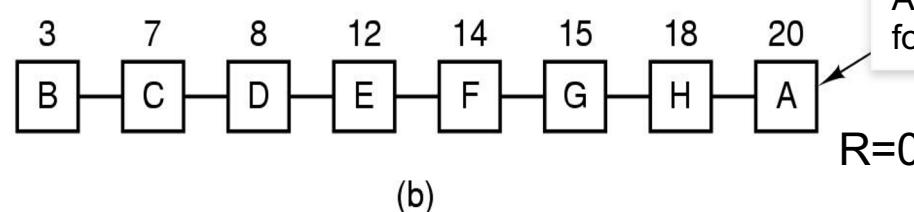
# Algoritmo da Segunda Chance

- É uma variante do algoritmo FIFO que leva em conta acessos recentes às páginas:
- Idéia básica: um acesso à pagina "rejuvenece a página"
- Funcionamento:
  - Páginas são mantidas em lista circular FIFO (ordenada por momento de carregamento)
  - Inspeciona-se a página no começo da lista:
  - Se página mais antiga possui bit  $R=0$ , ela é removida.
  - Se tiver bit  $R=1$ , o bit é zerado, e a página é movida para final da fila,. Ou seja: dá se uma 2<sup>a</sup> chance.

Página carregada há mais tempo.

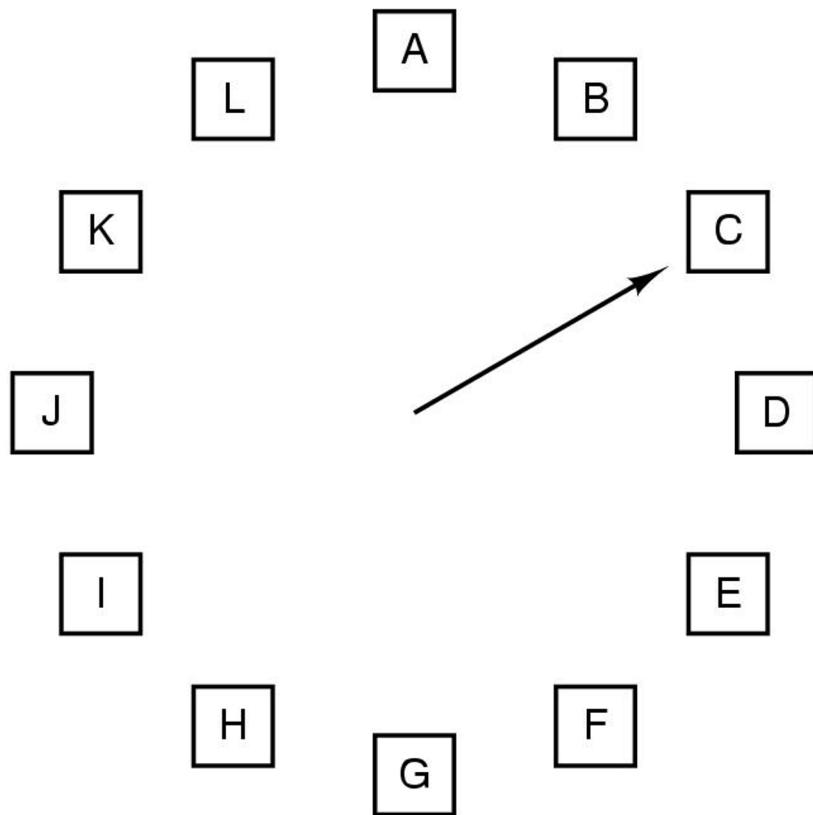


Página carregada mais recentemente.



A página é tratada como se fosse carregada recentemente.

# Algoritmo do Relógio



Objetivo: Substituir a página que ficou um ciclo sem ter sido referenciada.

Quando ocorre um page-fault, a página sendo referenciada pelo ponteiro é inspecionada.

A ação depende do bit R:

R=0: descarta a página

R=1: zera o bit R e avança o ponteiro

- Trata-se de uma implementação alternativa ao algoritmo da 2a. Chance (porém agora com uma lista circular)
- Considerar que todos os quadros de página estão em uma lista circular e analisar o reference bit das páginas lá contidas.

# Algoritmos contadores

Ideia: conta-se o número total de referências a cada página (usa-se R como contador e não como flag)

## Not Frequently Used

- substitui-se a página com o menor contador
- assume que a página ter sido a menos referenciada até o momento, diminui a probabilidade dela ser referenciada no futuro próximo

## Most Frequently Used

- substitui-se a página com o maior contador acumulado
- assume que uma página que já foi muito referenciada provavelmente irá demorar mais tempo para ser referenciada novamente

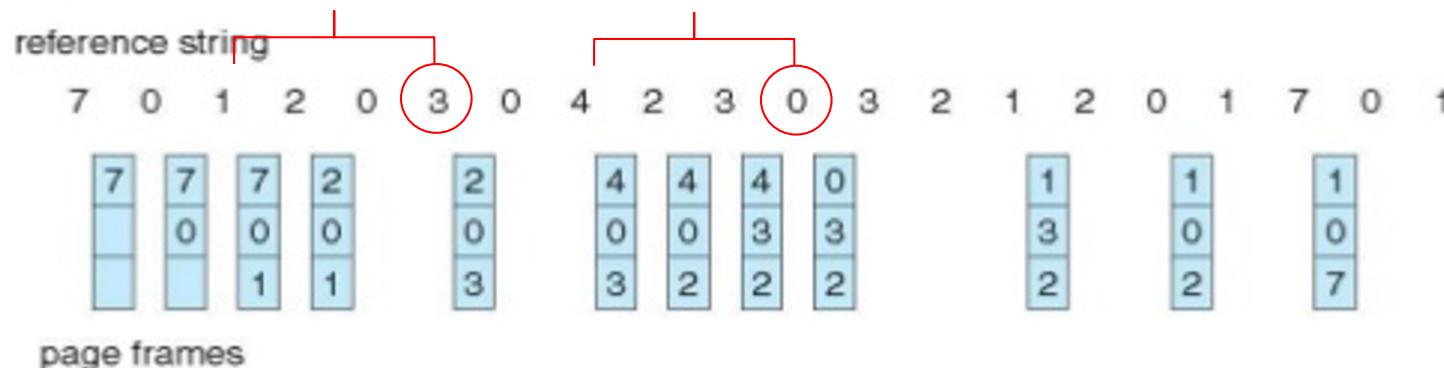
Desvantagem comum: o valor acumulado do contador de referências é cumulativo e não diz nada sobre o padrão recente de referências.

# Algoritmo “Menos recentemente utilizada” Least Recently Used (LRU)

Assume que páginas usadas recentemente, deverão ser usadas em breve novamente.

Princípio básico: descartar a página que ficou sem acesso durante o maior período de tempo no passado recente.

Exemplo: em memória física de 3 quadros de página



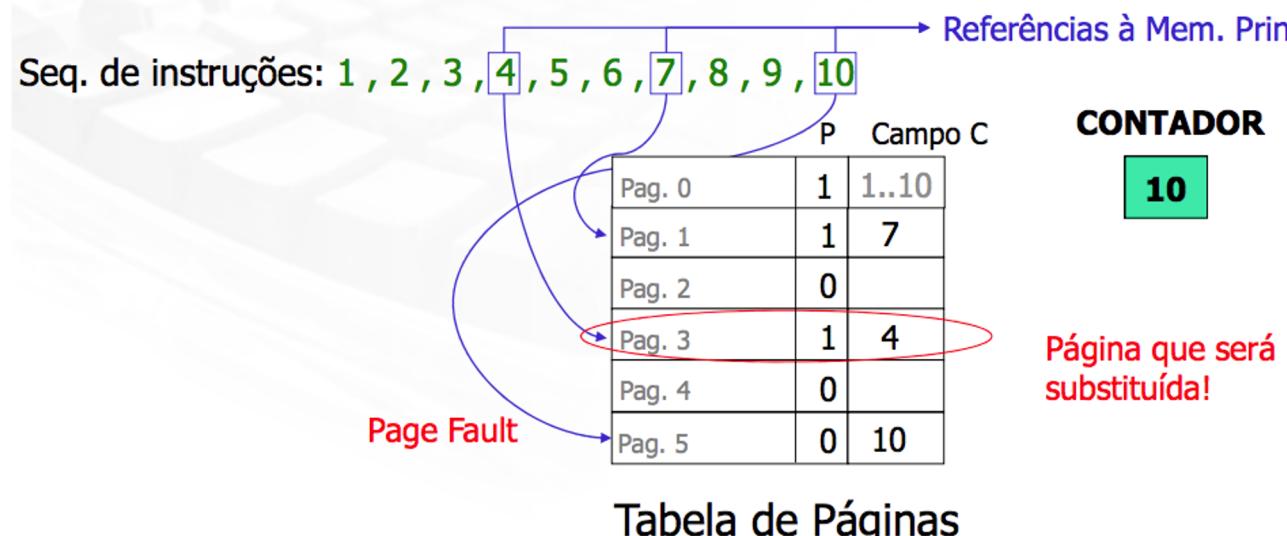
A Implementação ideal, porém impraticável:

- lista duplamente encadeada, ordenada de acordo com a ordem de ocorrência das referências
- a página mais recentemente acessada no início da lista e a menos recentemente utilizada no final.
- É inviável atualizar /manipular essa lista a cada acesso de memória!!

# Algoritmo “Menos Recentemente utilizada” Least Recently Used (LRU)

Alternativa à fila ordenada (necessita de apoio de hardware):

- usar um contador “de tempo decorrido” incrementado por hardware a cada acesso à memória
- Cada vez que uma página é acessada, atualiza este contador para página correspondente
- Na hora da substituição, precisaria-se comparar todos os contadores e achar o de menor valor (a cada page fault).



# LRU com Hardware especial

Marcador da idade pode ser realizada por um hardware específico.

Manipular uma matriz  $n \times n$  (para  $n$  entradas na TP) com o registro de acessos recentes (linha  $i$  = contador da página  $i$ )

Idéia: Ao acessar página  $i$  preencha toda linha  $i$  com bit 1 e, em seguida, toda coluna  $i$  com bit 0. A página mais antiga é aquela com menor valor.

Desvantagem: necessita hardware onde  $n$  é o número de quadros de página

	Page			
	0	1	2	3
0	0	1	1	1
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

Acesso Pág. 0

	Page			
	0	1	2	3
0	0	0	1	1
1	0	1	1	1
2	0	0	0	0
3	0	0	0	0

Acesso Pág. 1

	Page			
	0	1	2	3
0	0	0	0	1
1	0	0	0	1
2	1	0	0	1
3	0	0	0	0

Acesso Pág. 2

	Page			
	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	1	0	0	0
3	1	1	1	0

Acesso Pág. 3

	Page			
	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	1	0	0	0
3	1	1	0	0

Acesso Pág. 2

0	0	0	0
1	0	1	1
1	0	0	1
1	0	0	0

Acesso Pág. 1

0	1	1	1
0	0	1	1
0	0	0	1
0	0	0	0

Acesso Pág. 0

0	1	1	0
0	0	1	0
0	0	0	0
1	1	1	0

Acesso Pág. 3

0	1	0	0
0	0	0	0
1	1	0	1
1	1	0	0

Acesso Pág. 2

0	1	0	0
0	0	0	0
1	1	0	0
1	1	1	0

Acesso Pág. 3

# Algoritmo do envelhecimento (Aging): Simulando LRU em Software

Manter um contador para cada página;

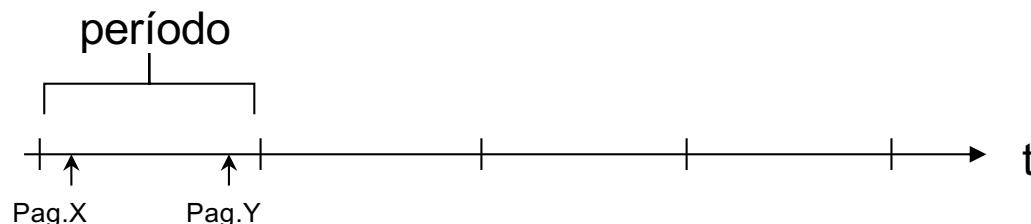
- Vetor de bits, onde bit  $i$  registra se página  $P_i$  foi acessada durante último período de tempo (cada tick de relógio):
- faz-se um shift de um bit para a direita no contador de cada página e adiciona-se o R-bit como bit mais significativo no contador
- Desvantagem: o vetor precisa ter tantos bits quanto houver de quadros de página

Cada page frame tem um contador	Vetor de bits atualizado periodicamente				
	R bits for pages 0-5, clock tick 0	R bits for pages 0-5, clock tick 1	R bits for pages 0-5, clock tick 2	R bits for pages 0-5, clock tick 3	R bits for pages 0-5, clock tick 4
Page	1 0 1 0 1 1	1 1 0 0 1 0	1 1 0 1 0 1	1 0 0 0 1 0	0 1 1 0 0 0
0	10000000	11000000	11100000	11110000	01111000
1	00000000	10000000	11000000	01100000	10110000
2	10000000	01000000	00100000	00100000	10001000
3	00000000	00000000	10000000	01000000	00100000
4	10000000	11000000	01100000	10110000	01011000
5	10000000	01000000	10100000	01010000	00101000

(a) (b) (c) (d) (e)

# Principais limitações do Algoritmo do Envelhecimento (comparado ao LRU com hardware)

- **Imprecisão:** Quando duas páginas possuem o mesmo valor de contador (de idade), não há como saber qual delas foi acessada por último (no último período de tempo)

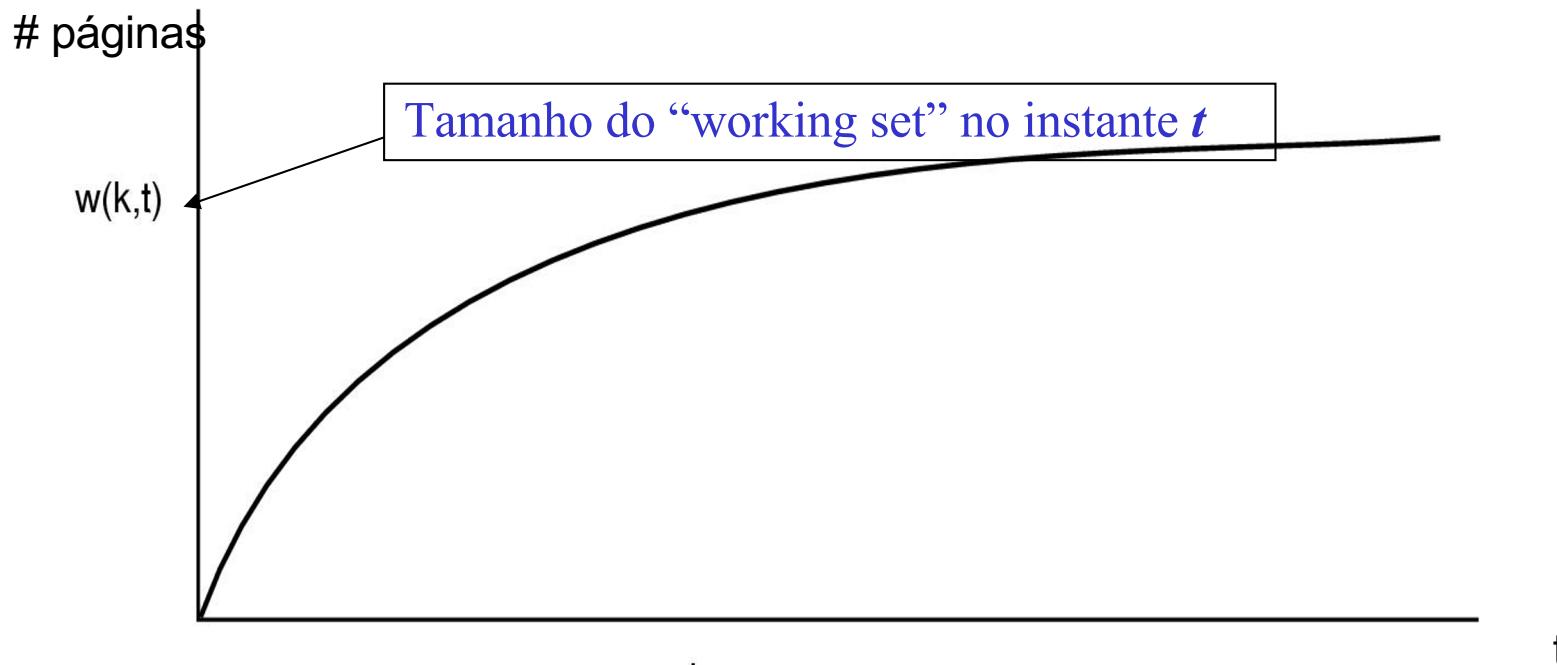


- **Impossível comparar referências antigas:** Os contadores de idade têm um número finito de bits. Portanto quando o valor atinge 0, não há como distinguir se a última referência ocorreu há apenas n ticks do relógio ou há muito mais tempo.

→ Normalmente, basta um contador com 8 bits e períodos de 20 ms entre ticks. Se uma página não foi acessada há 160 ms, provavelmente não é mais relevante.

# O Conjunto de páginas em uso (Working Set)

A maioria dos processos apresenta uma localidade de referência (a cada momento, acessa apenas uma fração pequena de suas páginas).



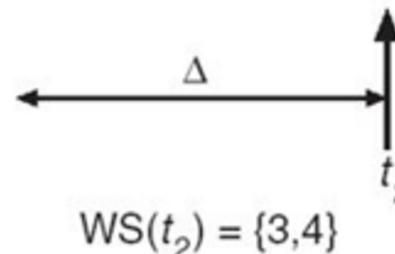
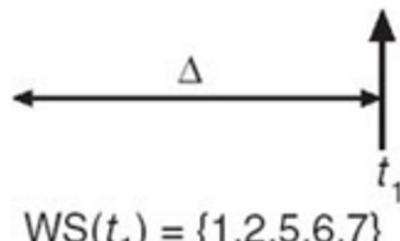
- O conjunto de trabalho (working set) é o conjunto das páginas acessadas pelas  $k$  referências mais recentes. O sistema de paginação deve gerenciar este conjunto usando envelhecimento.
- Quando um processo começa sua execução seu working-set pode ser pré-paginado (em vez de fazer paginação por demanda)

# O Working Set

- Working set  $WS(t, \Delta)$  é o conjunto de páginas lógicas referenciadas no intervalo  $(t - \Delta, t)$  acessos
- $\Delta$  é a working set window
- O tamanho do working set muda dependendo do grau de localidade dos acessos a memória
- Nos períodos de maior dispersão dos acessos, há acessos a mais páginas, e o working set fica maior.

page reference table

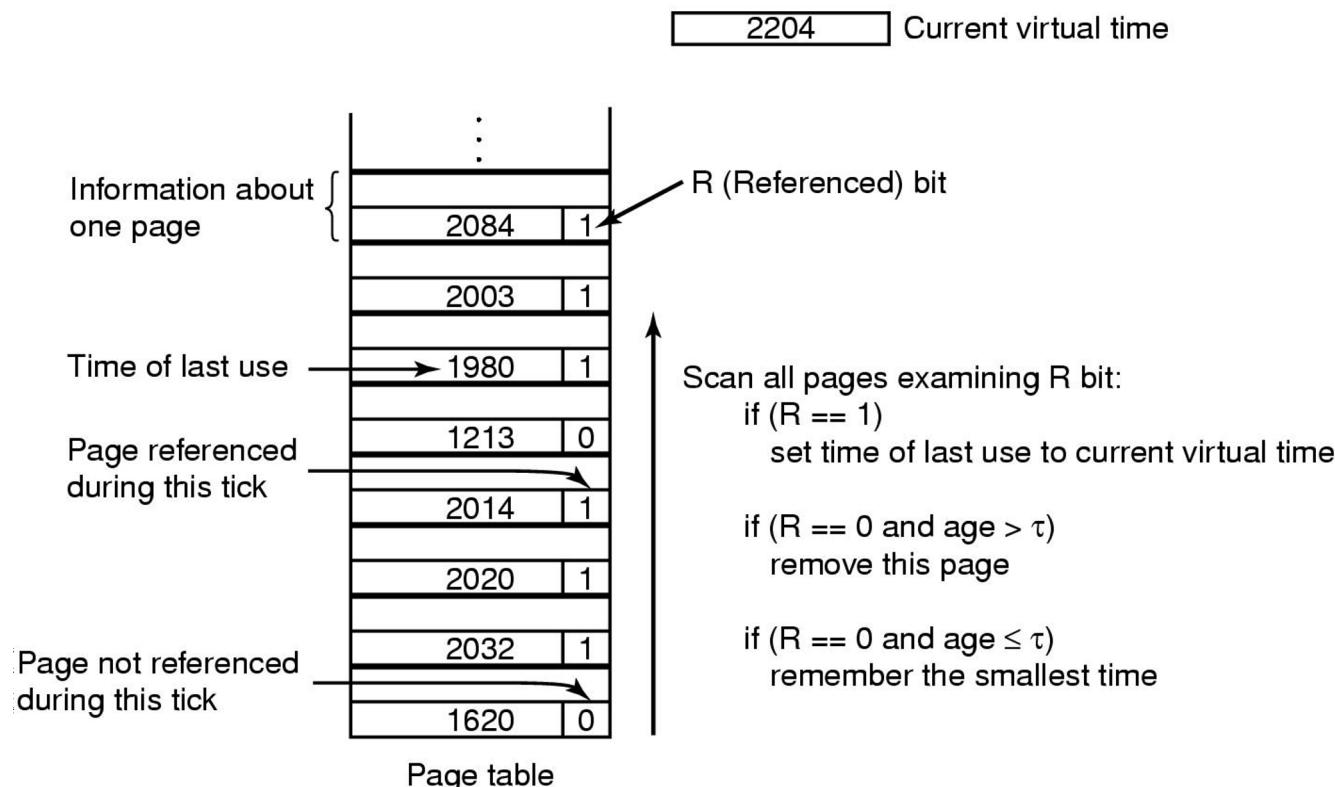
... 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 ...



- O Working set é uma aproximação do que o processo irá acessar no futuro próximo, e indica quais páginas devem permanecer na RAM.

# Algoritmo de substituição baseado no working set

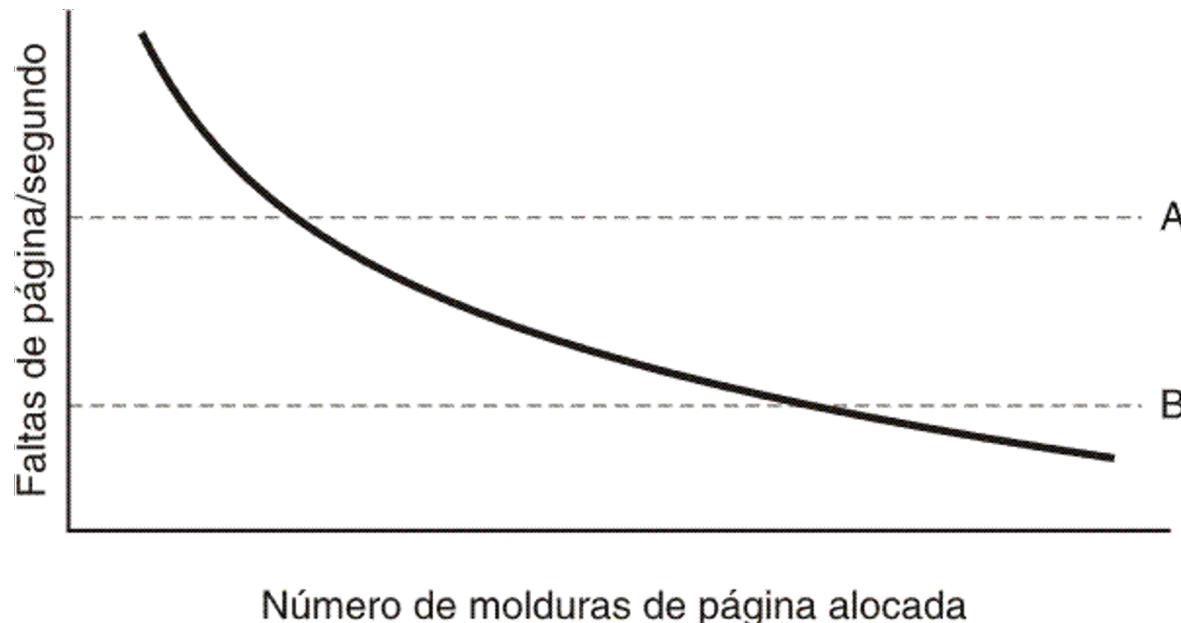
Mantém-se um contador virtual de tempo e periodicamente, atualiza-se o contador em cada página que tenha o bit R=1. Working Set consiste das páginas com contador dentro da janela de tempo  $[t - \Delta, t]$



Se  $R=0$ , não é feita atualização. Se  $(R==0 \&& \text{age} \leq \tau)$  avança com tempo virtual e depois verifica novamente. Remove-se a página com  $R=0$  e mais velhas do que  $\tau$

# Algoritmo de Frequênciade Page-Faults

O Algoritmo do Working Set é interessante, mas é bastante custoso analisar a atualizar as entradas da Tabela de página.



Freqüência de faltas de página como função do número de molduras de página alocadas

- (A)= taxa muito alta de page-fault,
- (B)= taxa muito baixa de page-faults

O Algoritmo Page Fault Frequency (PFF) fica monitorando essas taxas para decidir de qual processo irá diminuir o número de quadros de página.

- se a taxa de falhas estiver acima de um limite superior, aloque mais quadros de página para o processo, e/ou suspenda alguns processos
- se a taxa estiver abaixo de um limite inferior, retire quadros de página do processo,

# Resumo dos algoritmos de substituição de páginas

Nome do Algoritmo	Avaliação/ Comentário
Ótimo – prevendo o futuro	Impossível, mas usado como benchmark
Not Recently Used (NRU)	Pouco preciso
FIFO	Não leva referências em consideração
Segunda Chance	Melhoria significativa comparado com o FIFO
Least Recently Used (LRU)	Alta acurácia, mas requer implementacão em hardware
Not Frequently Used (NFU)	Contador de referências não leva em conta comportamento recente
Aging	Algoritmo com resultado próximo ao LRU em software
Working Set	Preciso, mas é custoso manter o WS