

Capítulo 9: Memória virtual



Capítulo 9: Memória virtual

- ❑ Base
- ❑ Paginação por demanda
- ❑ Cópia na escrita
- ❑ Substituição de página
- ❑ Alocação de frames
- ❑ Thrashing
- ❑ Arquivos mapeados na memória
- ❑ Alocando memória ao kernel
- ❑ Outras considerações
- ❑ Exemplos de sistema operacional



Objetivos

- ❑ Descrever os benefícios de um sistema de memória virtual
- ❑ Explicar os conceitos de paginação por demanda, algoritmos de substituição de página e alocação de frames de página
- ❑ Discutir os princípios do modelo de conjunto de trabalho

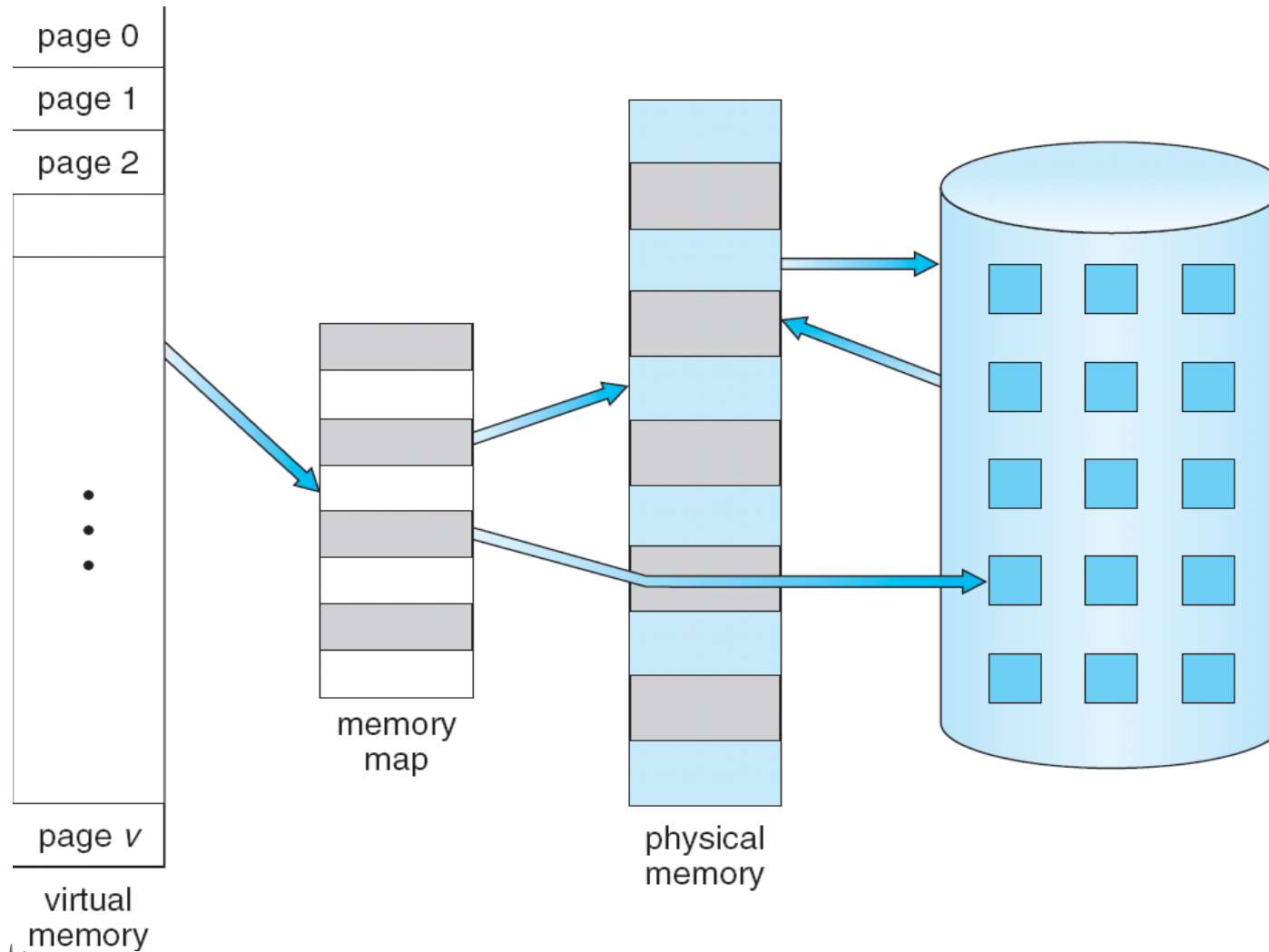


Base

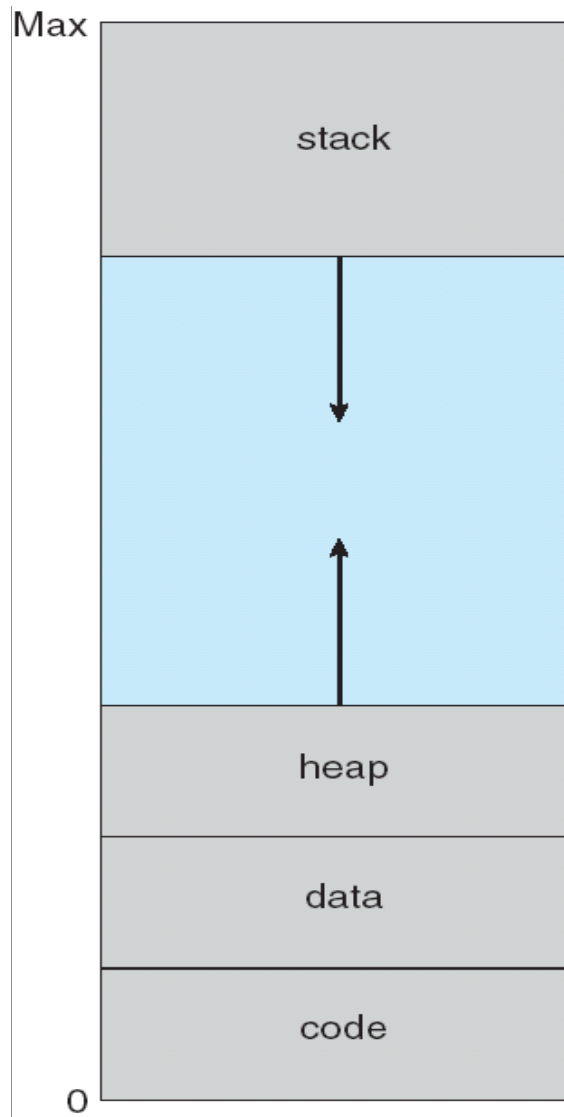
- ❑ **Memória virtual** – separação da memória lógica do usuário da memória física.
 - Somente parte do programa precisa estar na memória para execução
 - Logo, espaço de endereço lógico pode ser muito maior que o espaço de endereços físicos
 - Permite que espaços de endereço sejam compartilhados por vários processos
 - Permite criação de processo mais eficiente
- ❑ A memória virtual pode ser implementada por:
 - Paginação por demanda
 - Segmentação por demanda



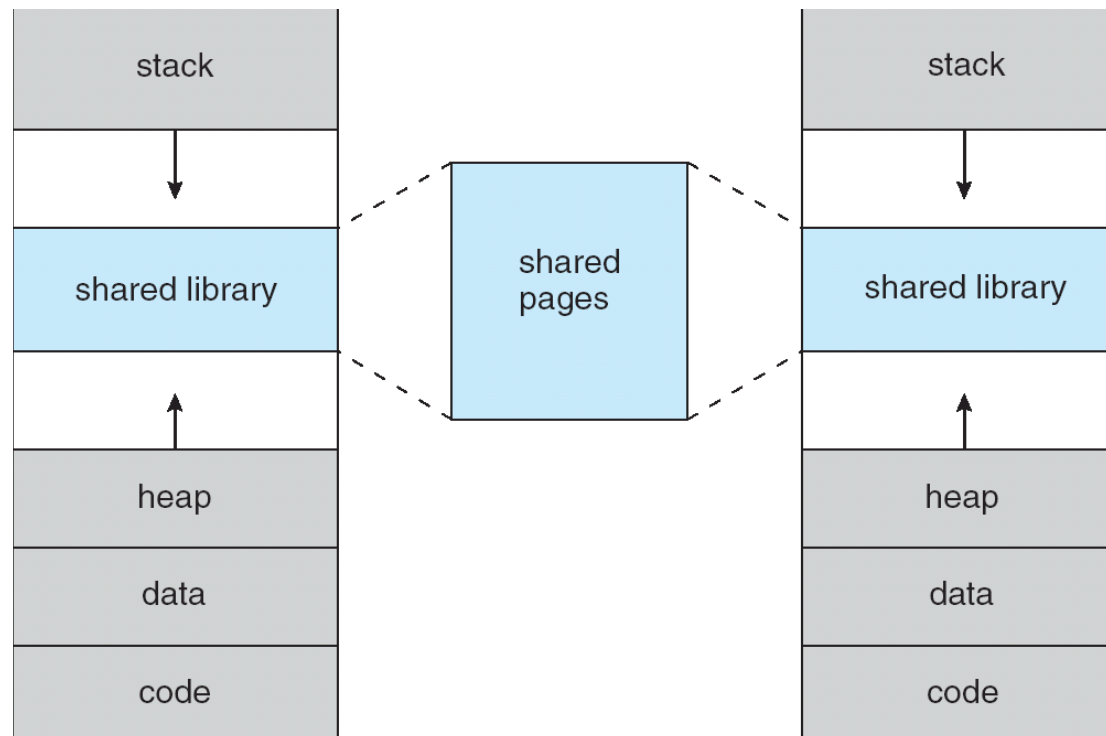
Memória virtual maior que a memória física



Espaço de endereço virtual



Biblioteca compartilhada usando memória virtual

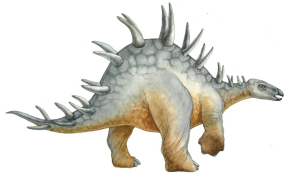
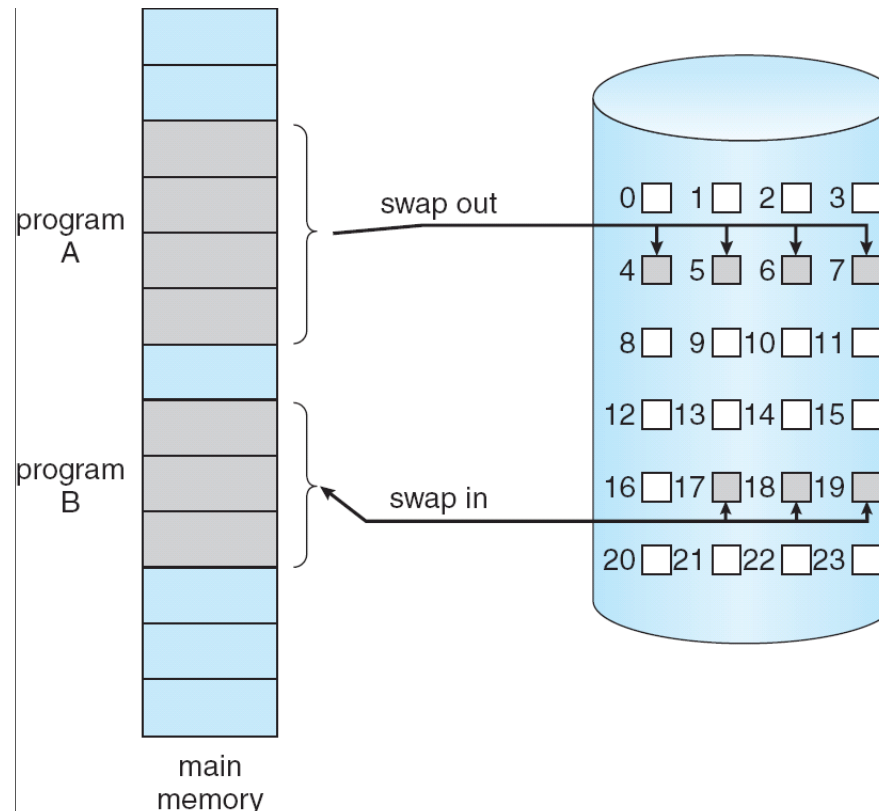


Paginação por demanda

- Traz a página para memória somente quando necessário
 - Menos E/S necessária
 - Menos memória necessária
 - Resposta mais rápida
 - Mais usuários
 - Referência inválida, então aborta
 - Não está na memória, então traz para memória
- **Lazy swapper** – nunca traz uma página para memória, a menos que a página seja necessária



Transferência de página da memória para espaço contíguo em disco



Bit válido-inválido

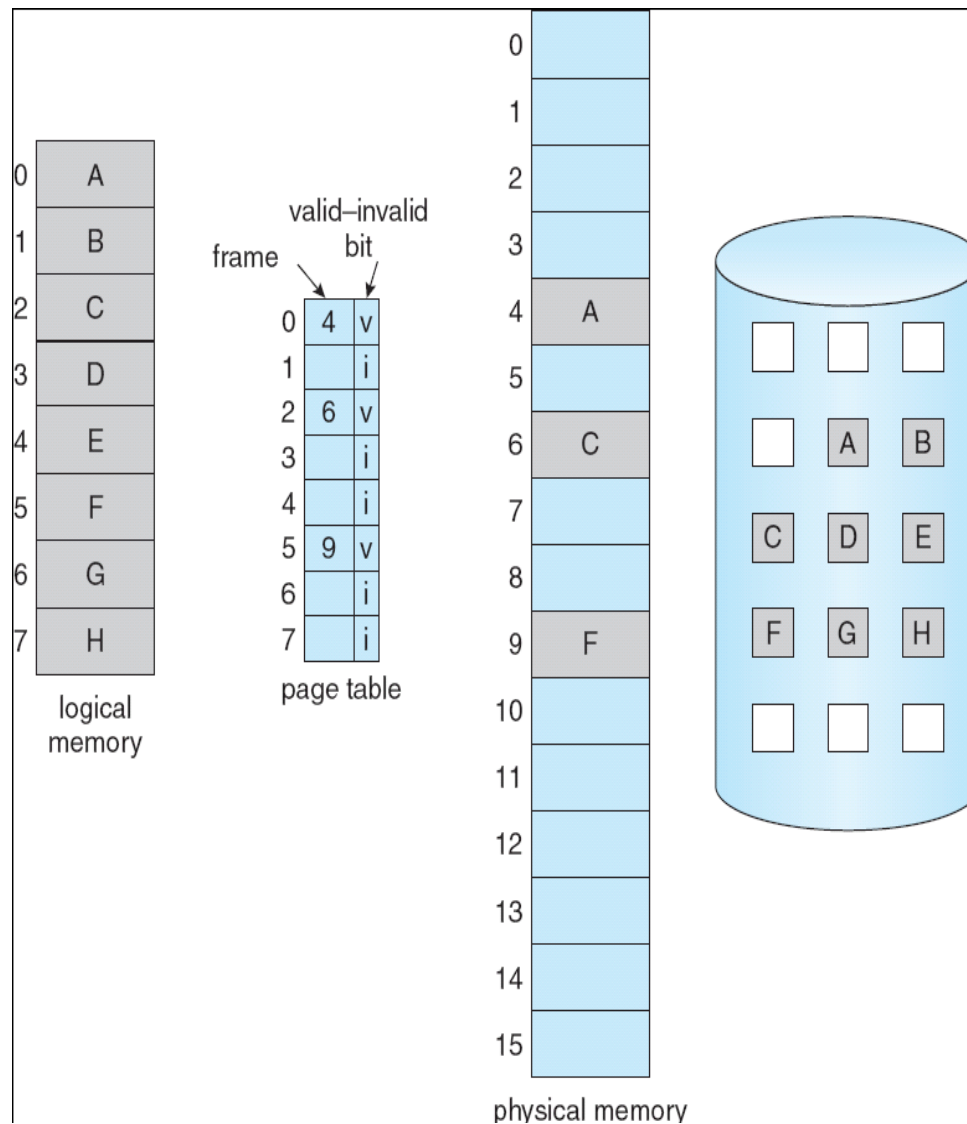
- A cada entrada de tabela de página, um bit de válido-inválido é associado (**v** = está na memória, **i** caso contrário)
- Inicialmente, o bit válido–inválido é definido como **i** em todas as entradas
- Exemplo

Quadro #	Bit válido-inválido
	v
	v
	v
	v
	i
....	
	i
	i

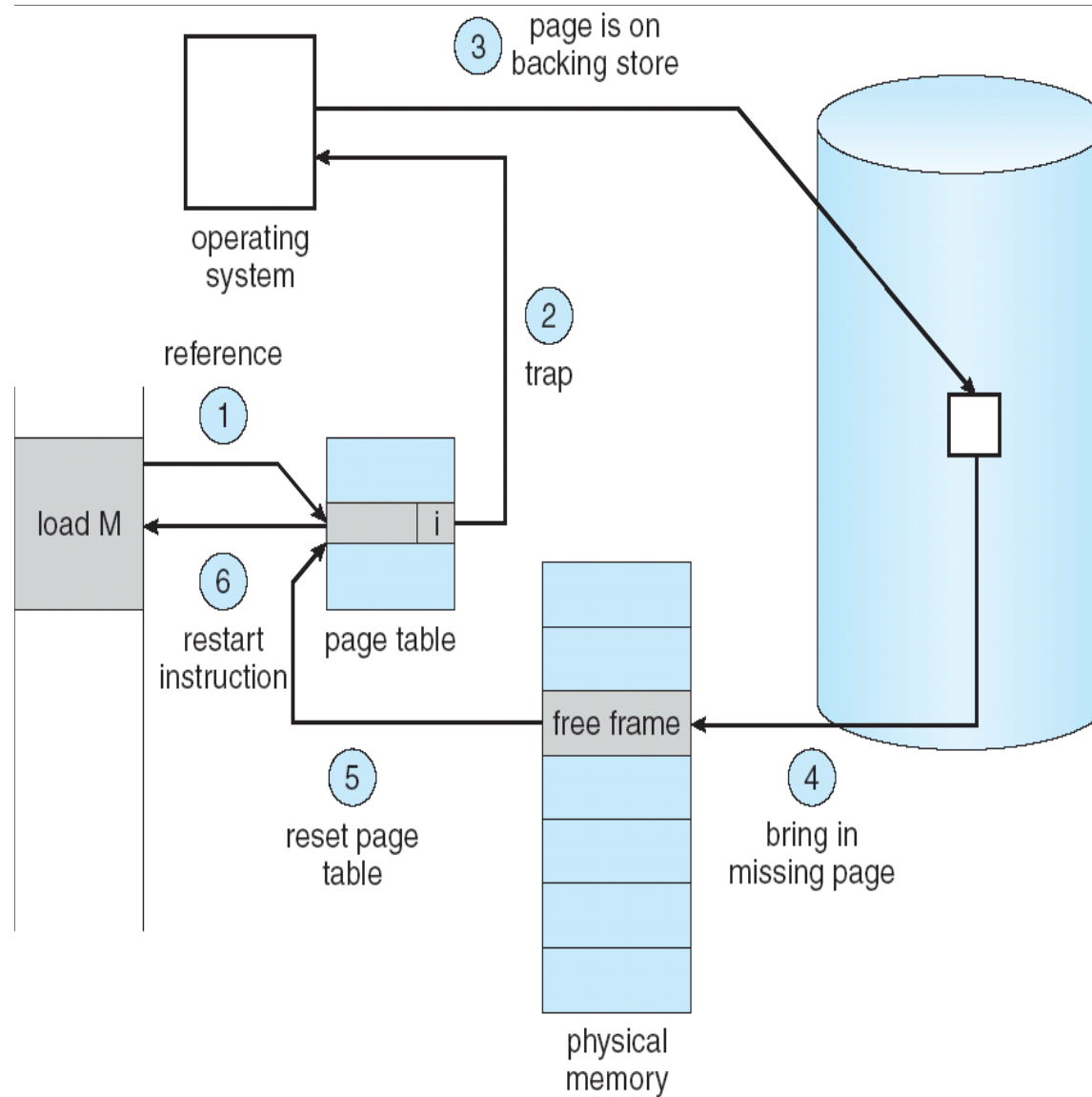
tabela de página



Exemplo



Etapas no tratamento de falta de página



Desempenho da paginação por demanda

- Taxa de falta de página $0 < p < 1.0$
 - se $p = 0$ nenhuma falta de página
 - se $p = 1$, cada referência é uma falta
- Tempo de acesso efetivo (EAT)
$$\text{EAT} = (1 - p) \times \text{acesso à memória} +$$
$$p \times (\text{overhead de falta de página}$$
$$+ \text{swap página fora}$$
$$+ \text{swap página dentro}$$
$$+ \text{reiniciar overhead})$$



Exemplo de paginação por demanda

- Tempo de acesso à memória = 200 nanossegundos
- Tempo médio de serviço de falta de página = 8 milissegundos
- $EAT = (1 - p) \times 200 + p (8 \text{ milissegundos})$
 $= (1 - p) \times 200 + p \times 8.000.000$
 $= 200 + p \times (8.000.000 - 200)$
 $= 200 + p \times 7.999.800$
- Se um acesso dentre 1.000 causar uma falta de página, então
 $EAT = 8.2 \text{ microssegundos.}$
Isso é um atraso por um fator de 40!!



Criação de processo

- A memória virtual permite outros benefícios durante a criação do processo:
 - Cópia na escrita
 - Arquivos mapeados na memória



Cópia na escrita

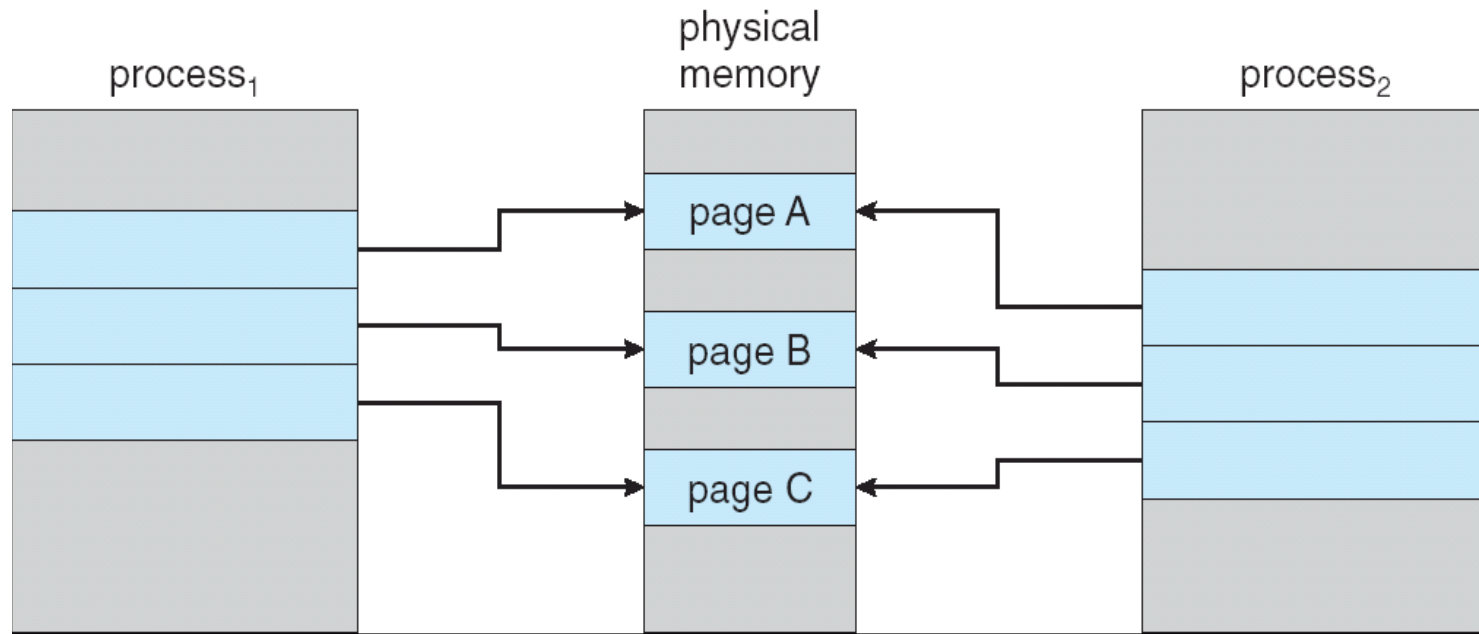
- ❑ Cópia na escrita permite que processos pai e filho inicialmente *compartilhem* as mesmas páginas na memória

Se qualquer processo modificar uma página compartilhada, somente então a página é copiada

- ❑ A cópia na escrita permite criação de processo mais eficiente, pois somente páginas modificadas são copiadas
- ❑ Páginas livres são alocadas de um ***pool***



Antes que processo 1 modifique página C



Depois que processo 1 modifica página C

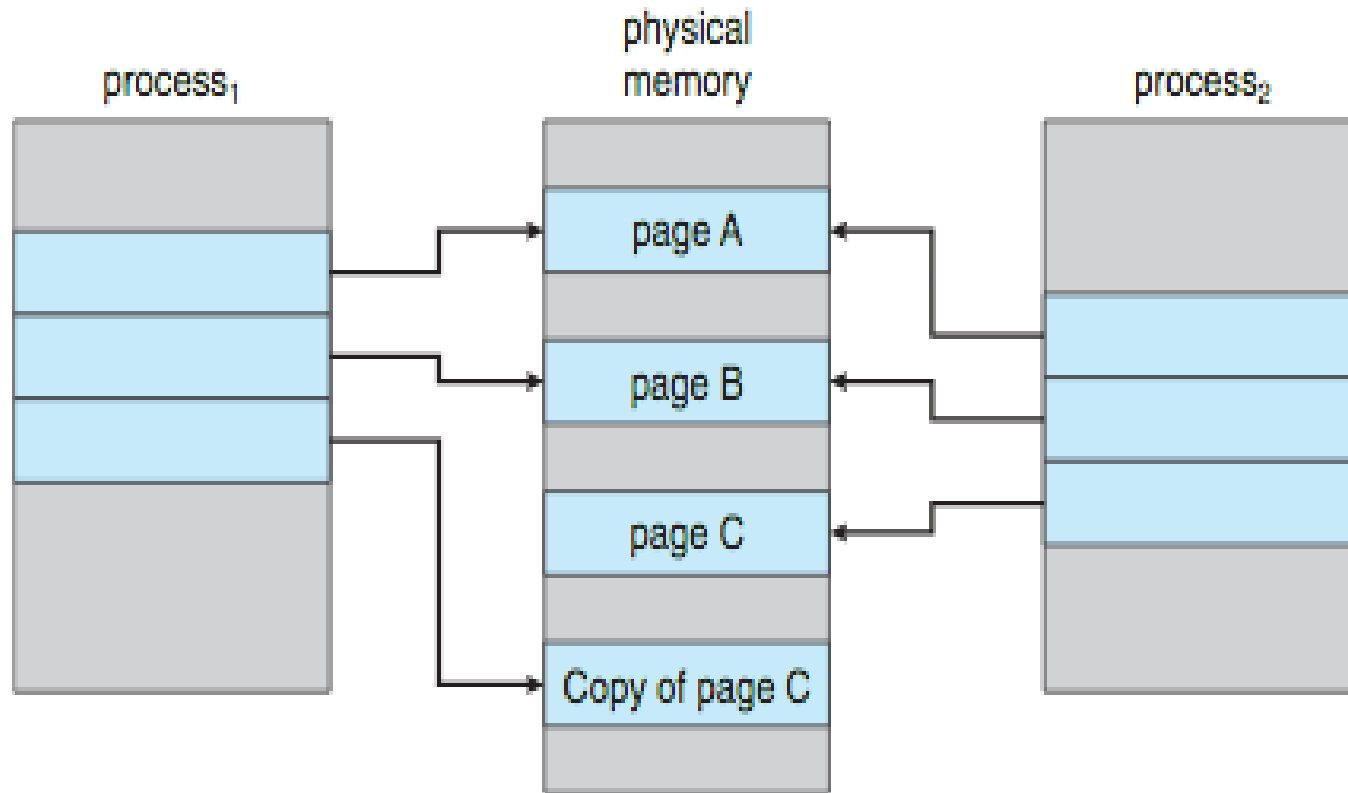


Figure 9.8 After process 1 modifies page C.



O que acontece se não houver frame livre?

- ❑ Substituição de página – encontre alguma página na memória, mas se não está em uso, passa para fora (swap out)
- ❑ Mesma página pode ser trazida para a memória várias vezes

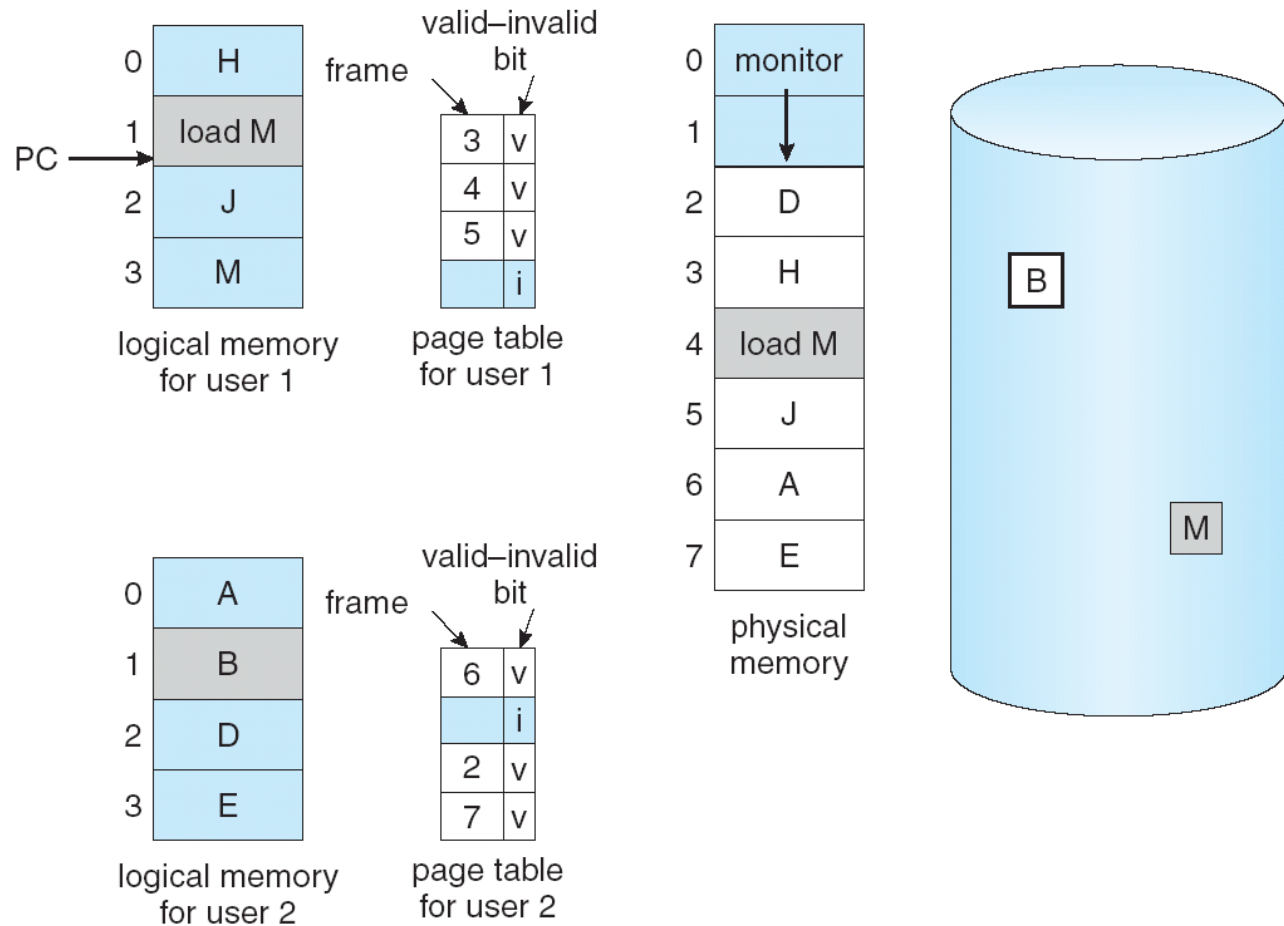


Substituição de página

- ❑ Usa **bit de modificação (sujo)** para reduzir overhead das transferências de páginas
- ❑ Somente páginas modificadas são gravadas em disco (o bit indica se a página na memória está igual à página no disco; se sim, o *swap out* iria reescrever o mesmo conteúdo no disco: desnecessário)
- ❑ Memória virtual grande pode ser fornecida em uma memória física menor



Necessidade de substituição de página

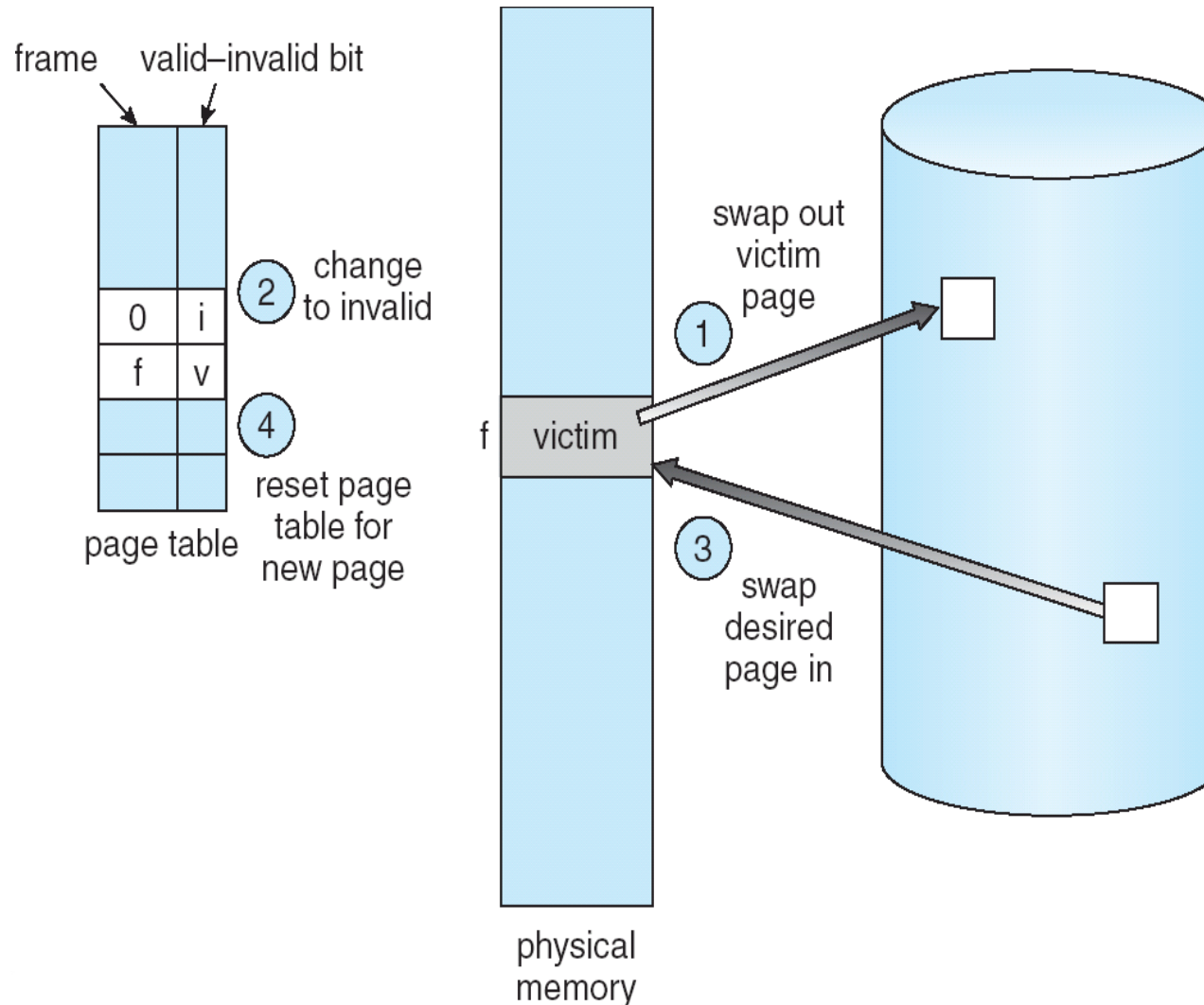


Substituição de página básica

1. Ache o local da página desejada no disco
2. Ache um quadro livre:
 - Se houver um quadro livre, use-o
 - Se não houver quadro livre, use um algoritmo de substituição de página para selecionar um quadro **vítima**
3. Traga a página desejada para o quadro (recém) livre; atualize a página e as tabelas de quadro
4. Reinicie o processo



Substituição de página



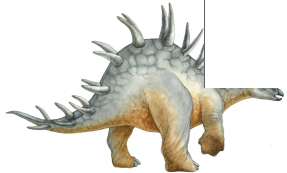
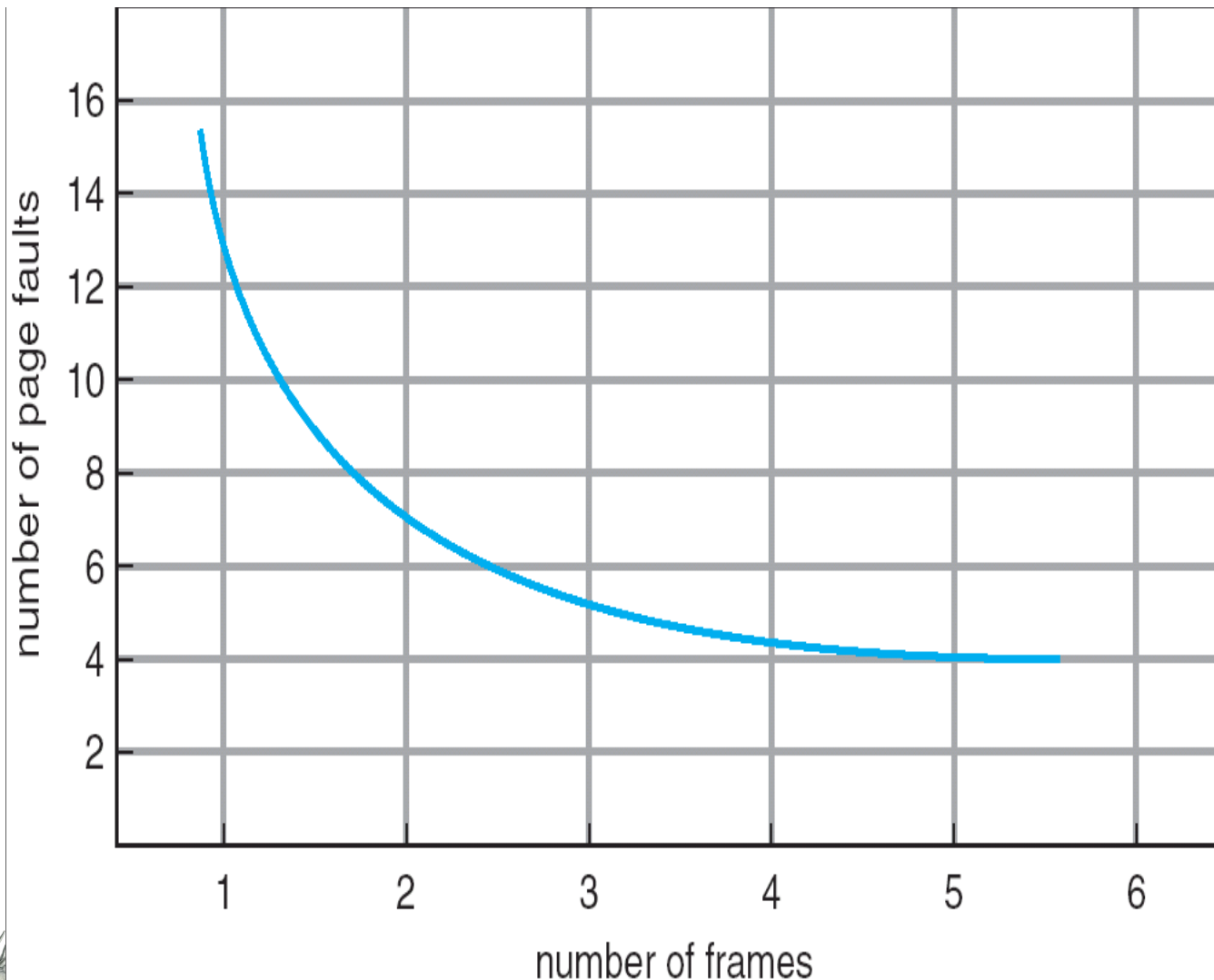
Algoritmos de substituição de página

- ❑ Deseja taxa de falta de página mais baixa
- ❑ Avalia algoritmo executando-o em uma string em particular de referências de memória (string de referência) e calculando o número de faltas de página nessa string
- ❑ Em todos os nossos exemplos, a string de referência é

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



Gráfico de faltas de página versus número de quadros (o que se esperaria de todo método de substituição)



Algoritmo First-In-First-Out (FIFO)

- String de referência: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 quadros (3 páginas podem estar na memória ao mesmo tempo por processo)

1	1	4	5	9 faltas de página
2	2	1	3	
3	3	2	4	

- 4 frames

1	1	5	4	10 faltas de página
2	2	1	5	
3	3	2		
4	4	3		



Substituição de página FIFO

reference string

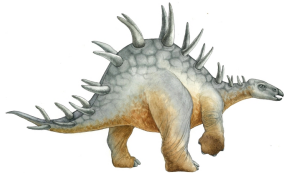
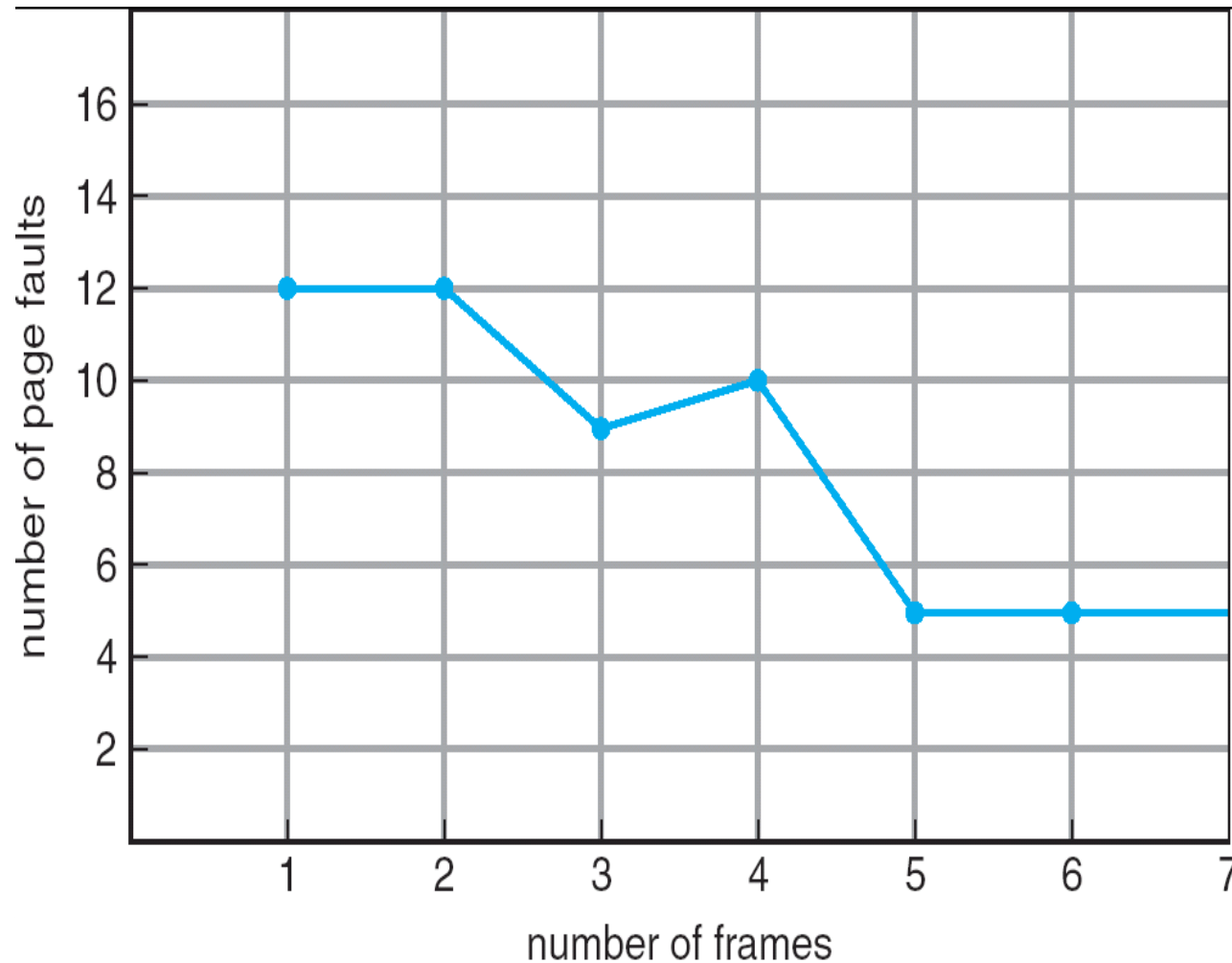
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2																
	0	0	0		2	2	4	4	4	0			0	0			7	7	7
		1	1		3	3	3	2	2	2			1	1			1	0	0
					1	0	0	0	3	3			3	2			2	2	1

page frames



FIFO ilustrando anomalia de Belady



Algoritmo ideal

- ❑ Substitua página que não será usada pelo maior período de tempo
- ❑ Exemplo de 4 frames

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1
2
3
4

4

6 faltas de
página

5

- ❑ Como você sabe disto?
- ❑ Usado para medir como seu algoritmo funciona



Substituição de página ideal

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2		2		2			2		2					7		
	0	0	0		0		4			0		0					0		
		1	1		3		3			3		1					1		

page frames



Algoritmo Least Recently Used (LRU)

- String de referência: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	1	1	1	5
2	2	2	2	2
3	5	5	4	4
4	4	3	3	3

- Implementação do contador
 - Cada entrada de página tem um contador; toda vez que a página é referenciada por essa entrada, copia o clock para o contador
 - Quando uma página precisa ser mudada, veja os contadores para determinar quais devem mudar



Substituição de página LRU

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2		2		4	4	4	0			1		1		1		
	0	0	0		0		0	0	3	3			3		0		0		
		1	1		3		3	2	2	2			2		2		7		

page frames



Algoritmo LRU (cont.)

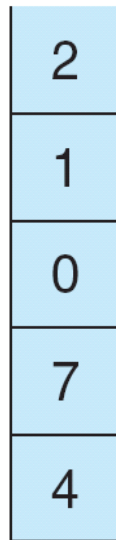
- Implementação com pilha – mantenha uma pilha de números de página em um formato de link duplo:
 - Página referenciada:
 - mova-a para o topo
 - requer que 6 ponteiros sejam trocados
 - Nenhuma busca para substituição



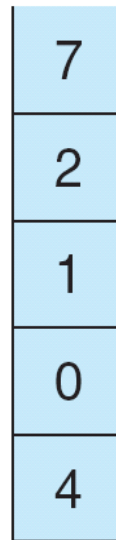
Pilha para registrar as referências de página mais recentes

reference string

4 7 0 7 1 0 1 2 1 2 7 1 2



stack
before
a



stack
after
b



Algoritmos de aproximação LRU

□ Bit de referência

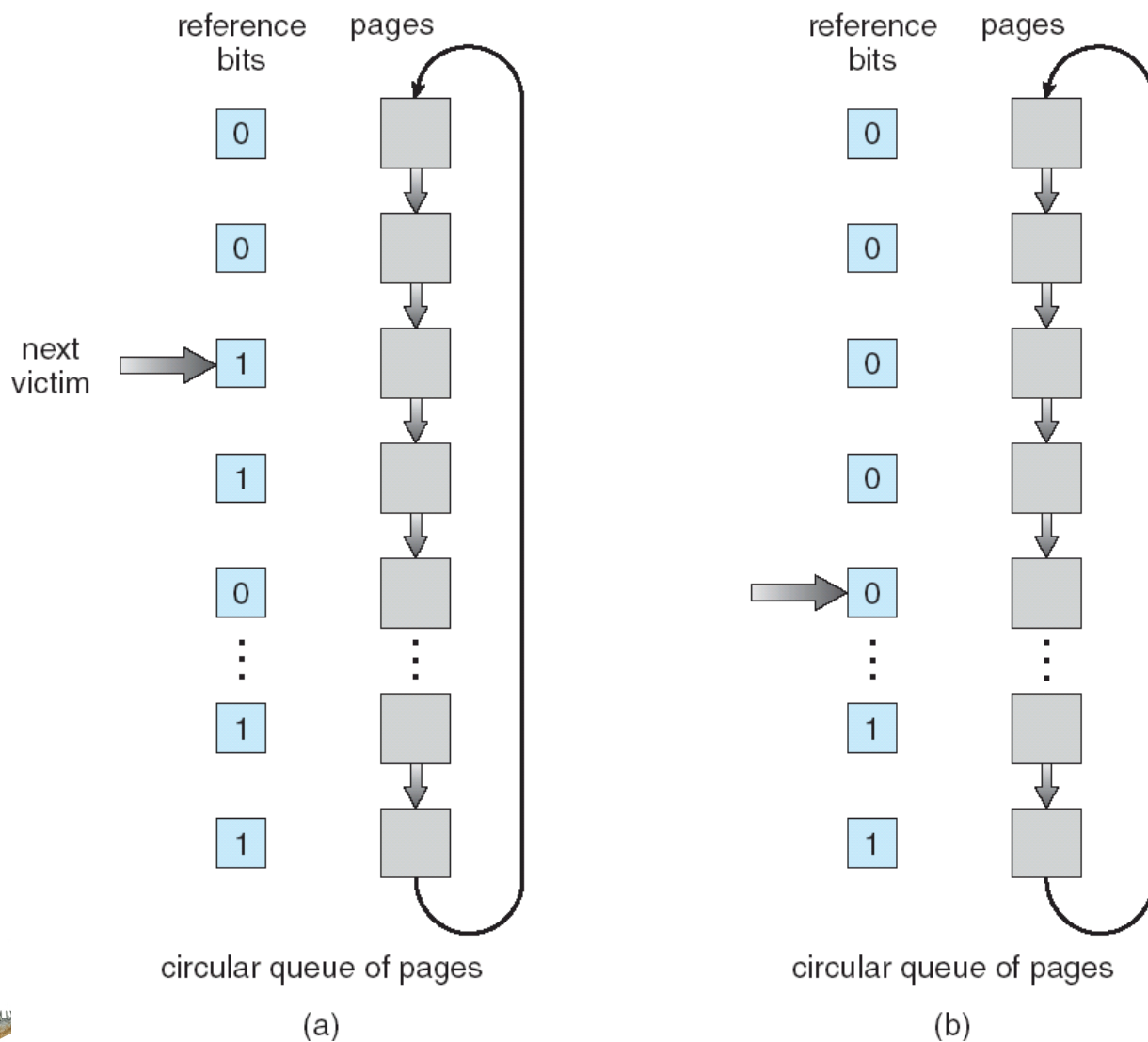
- A cada página associe um bit, inicialmente = 0
- Quando a página é referenciada, bit passa para 1
- Substitua a que é 0 (se uma existir)

□ Segunda chance

- Precisa de bit de referência
- Se página a ser substituída tem bit de referência = 0 então substitui
- Caso contrário (ou seja, bit de referência = 1)
 - define bit de referência 0 (“dá uma segunda chance”)
 - deixa página na memória
 - substitui próxima página (há ordem das páginas), sujeito às mesmas regras



Substituição de página com “segunda chance”



Algoritmos de contagem

- ❑ Mantenha um contador do número de referências que foram feitas a cada página
- ❑ **Algoritmo LFU:** substitui página por menor contador
- ❑ **Algoritmo MFU:** baseado no argumento de que a página com a menor contagem provavelmente acabou de ser trazida e ainda está para ser usada



Alocação de quadros

- Cada processo precisa do número *mínimo* de páginas
- Exemplo: IBM 370 – 6 páginas para tratar da instrução SS MOVE:
 - instrução tem 6 bytes, pode espalhar por 2 páginas
 - 2 páginas para tratar *de*
 - 2 páginas para tratar *para*
- Dois esquemas de alocação principais
 - alocação fixa
 - alocação por prioridade



Alocação fixa

- Alocação igual – Por exemplo, se houver 100 quadros e 5 processos, dá a cada processo 20 quadros.
- Alocação proporcional – Aloca de acordo com o tamanho do processo

— s_i ! tamanho do processo p_i
 — S ! $\sum s_i$
 — m ! número total de quadros
 — a_i ! alocação para p_i ! $\frac{s_i}{S} \cdot m$

m ! 64

s_1 ! 10

s_2 ! 127

a_1 ! $\frac{10}{137} \cdot 64$! 5

a_2 ! $\frac{127}{137} \cdot 64$! 59



Alocação por prioridade

- Usa um esquema de alocação proporcional ao invés de tamanho
- Se processo P_i gera uma falta de página,
 - seleciona para substituição um de seus quadros
 - seleciona para substituição um quadro de um processo com número de prioridade menor



Alocação global versus local

- ❑ **Substituição global** – processo seleciona um quadro de substituição do conjunto de todos os quadros; um processo pode apanhar um quadro de outro
- ❑ **Substituição local** – cada processo seleciona apenas do seu próprio conjunto de quadros alocados

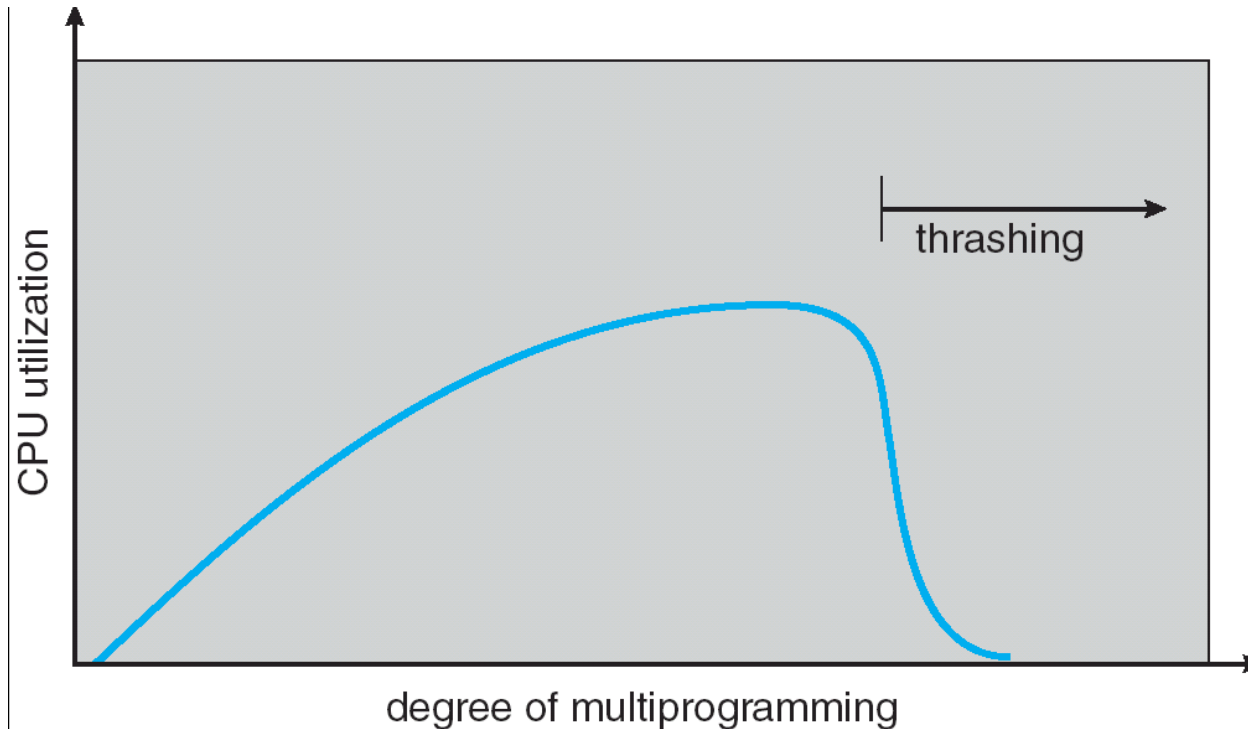


Thrashing

- ❑ Se um processo não tem páginas “suficientes”, a taxa de falta de página é muito alta. Isso leva a:
 - baixa utilização de CPU
 - sistema operacional pensa que precisa aumentar o grau de multiprogramação
 - outro processo acrescentado ao sistema
- ❑ **Thrashing** ! um processo está ocupado trocando páginas pra dentro e pra fora



Thrashing (cont.)

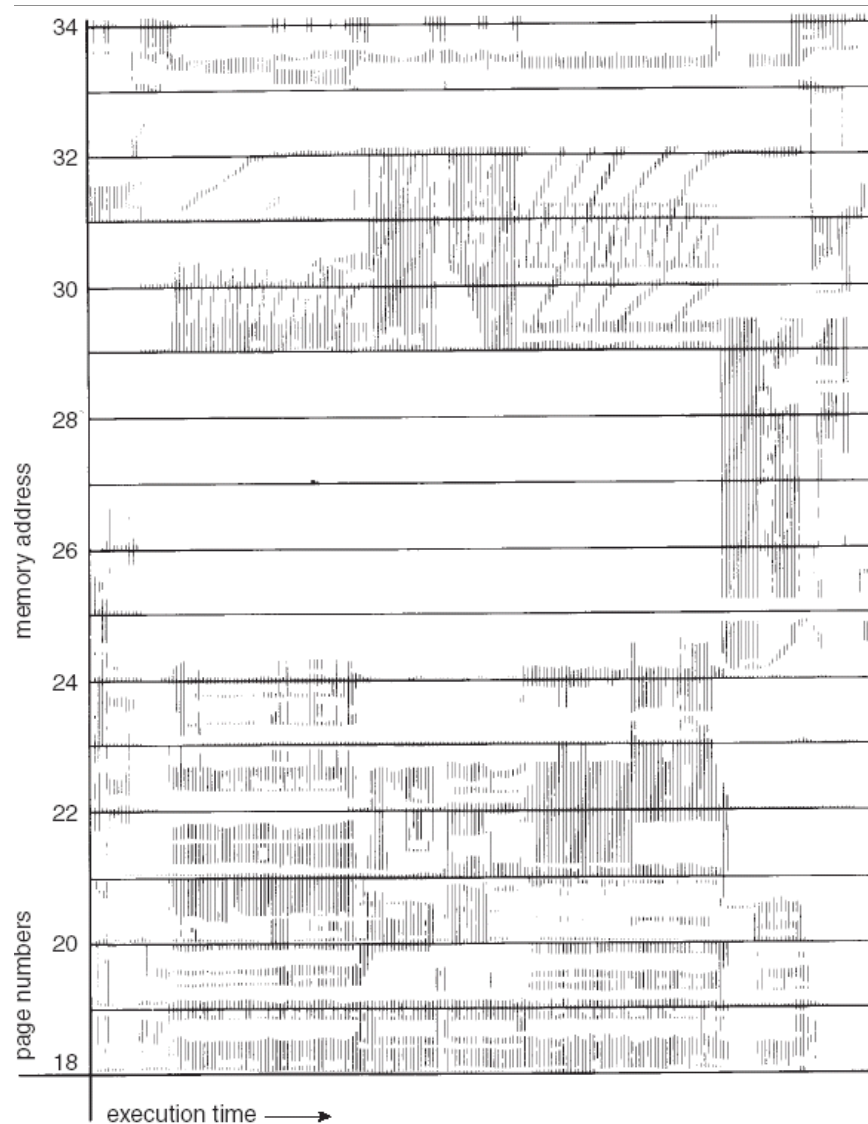


Paginação por demanda e thrashing

- Por que a paginação por demanda funciona?
Modelo de localidade
 - Processo migra de uma localidade para outra
 - Localidades podem se sobrepor
- Por que ocorre o thrashing?
! tamanho da localidade > tamanho total da memória



Localidade em um padrão de referência de memória



Modelo de conjunto de trabalho

janela conjunto de trabalho -- Δ um número fixo de referências de página

Exemplo: 10.000 instruções

WSS_i (conjunto de trabalho do Processo P_i) =
número total de páginas referenciadas no Δ mais recente (varia no tempo)

se Δ muito pequeno, não abrangerá localidade inteira

se Δ muito grande, abrangerá várias localidades

se $\Delta = \text{infinito}$ -> abrangerá programa inteiro

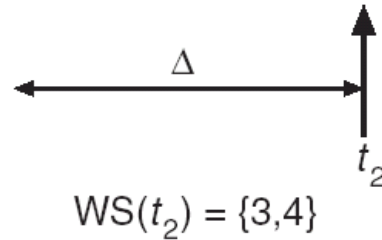
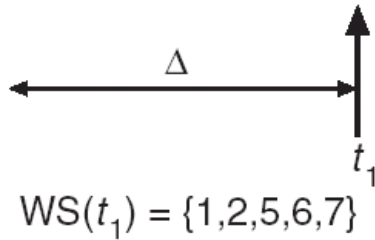
- ▣ $D = \sum WSS_i$! ! total de quadros por demanda
- ▣ se $D > m$ (*total quadros disponíveis na mem*) ! !
Thrashing
- ▣ Política se $D > m$, então suspenda um dos processos



Modelo de conjunto de trabalho

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 4 1 3 2 3 4 4 4 3 4 4 4 . . .



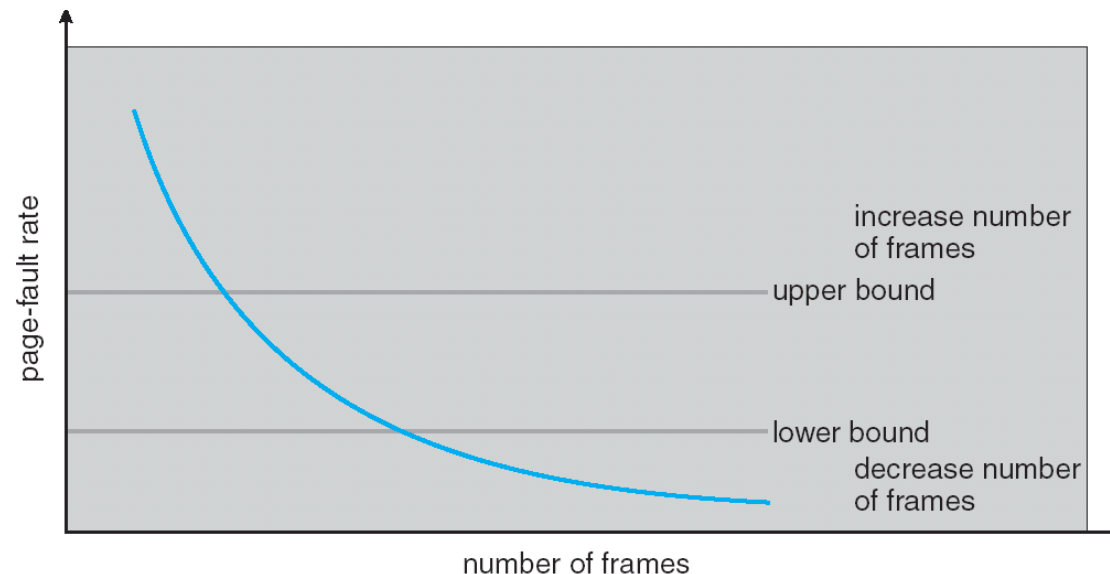
Acompanhando o conjunto de trabalho

- Aproximado com timer de intervalo + um bit de referência
- Exemplo: ! = 10.000
 - Interrupções de timer após cada 5000 unidades de tempo
 - Mantém na memória 2 bits para cada página
 - Sempre que um timer interrompe a cópia e define os valores de todos os bits de referência como 0
 - Se um dos bits na memória = 1 ! ! página está no conjunto de trabalho
- Por que isso não é completamente preciso?
- Melhoria = 10 bits e interrupção a cada 1000 unidades de tempo



Esquema de freqüência de falta de página

- Estabelece taxa de falta de página “aceitável”
 - Se a taxa real for muito baixa, processo perde quadro
 - Se a taxa real for muito alta, processo ganha quadro

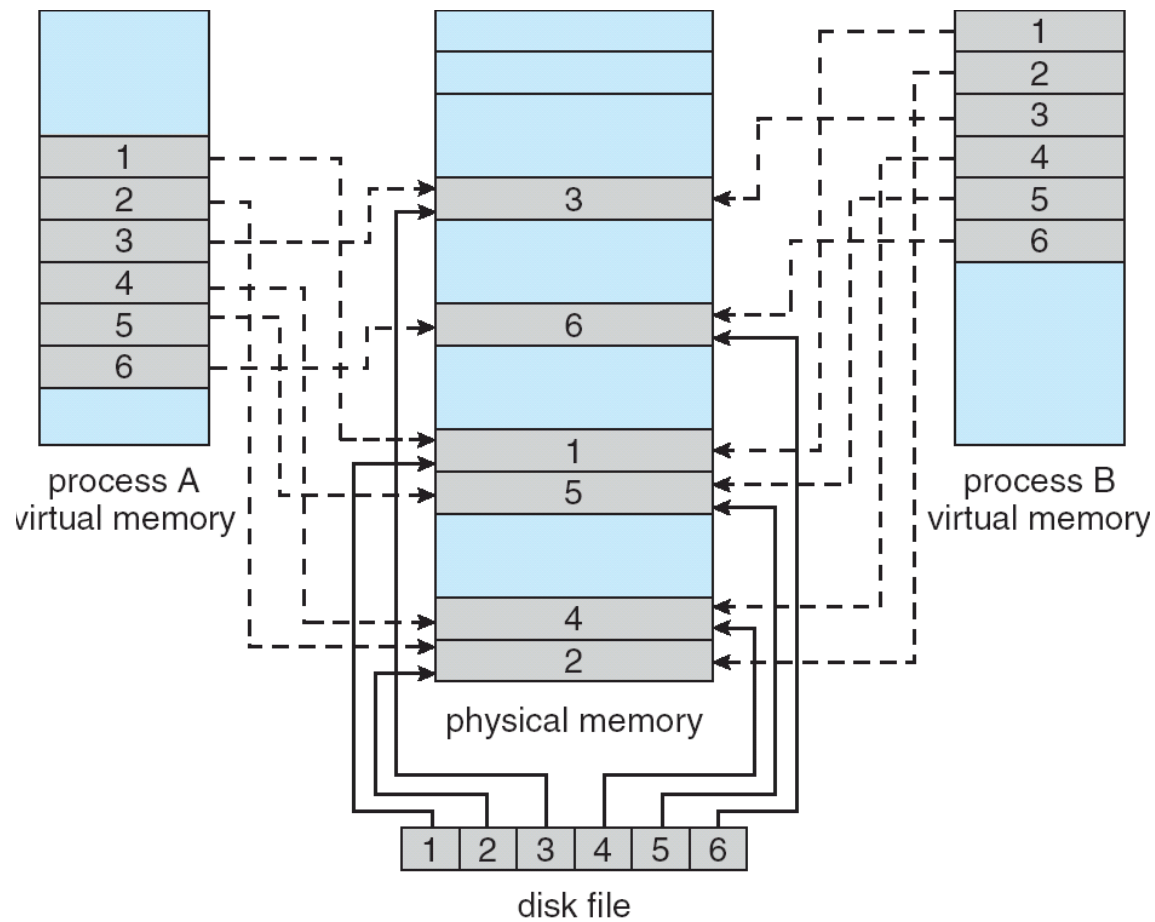


Arquivos mapeados na memória

- ❑ E/S de arquivo mapeado na memória permite que a E/S de arquivo seja tratada como acesso de rotina à memória, **mapeando** um bloco de disco para uma página na memória
- ❑ Um arquivo é lido inicialmente usando paginação por demanda. Uma parte do arquivo com tamanho da página é lida do sistema para uma página física. Leituras/escritas subseqüentes de/para o arquivo são tratadas como acessos comuns à memória.
- ❑ Simplifica o acesso ao arquivo, tratando a E/S do arquivo por meio da memória, ao invés das chamadas do sistema **read()** e **write()**
- ❑ Também permite que vários processos sejam mapeados para o mesmo arquivo, permitindo que as páginas na memória sejam compartilhadas



Arquivos mapeados na memória



Final do Capítulo 9

