

Infra-Estrutura de Software

Gerência de Memória

Tópicos

- Gerenciamento básico de memória
- Troca de processos (*swapping*)
- Memória virtual
- Troca de páginas/Paginação (*paging*)

Gerenciador de Memória

- Componente do Sistema Operacional que **aloca memória principal para os processos** e **gerencia a hierarquia de memória** (Caches, RAM e Disco)
 - Garante isolamento mútuo entre processos (**proteção**)
 - Mantém informação das áreas de memória em uso
 - Aloca memória RAM para novos processos (`fork()`)
 - Faz o **swapping** transparente entre memória principal e disco
 - Atende a requisições de aumento de memória
 - Mantém o mapeamento de memória virtual para memória física
 - Implementa a política de alocação de memória para os processos

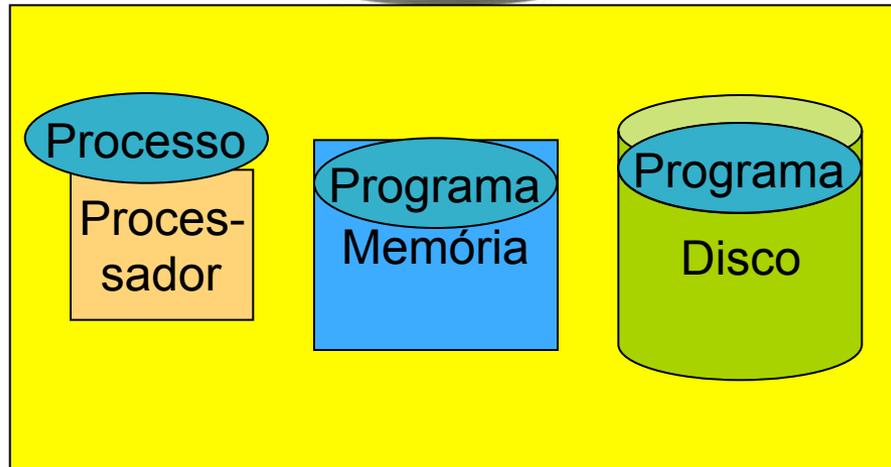
Gerenciamento de Memória

- Idealmente, o que todo programador deseja é dispor de uma memória que seja
 - grande
 - rápida
 - não volátil
- **Hierarquia de memórias**
 - pequena quantidade de memória rápida, de alto custo - **cache**
 - quantidade considerável de **memória principal** de velocidade média, custo médio
 - gigabytes de armazenamento em **disco** de velocidade e custo baixos
- **O gerenciador de memória trata a hierarquia de memórias**

Software

Como rodar um programa?

2. PC aponta para o endereço de memória onde o programa foi escrito
3. Processador executa instruções do programa trazidas da memória



1. Dado o comando para executar um programa, é realizada uma seqüência de instruções para copiar código e dados do programa objeto do disco para a memória principal



Gerenciamento de Memória sem Swapping ou Paginação

Monoprogramação: um único programa de usuário

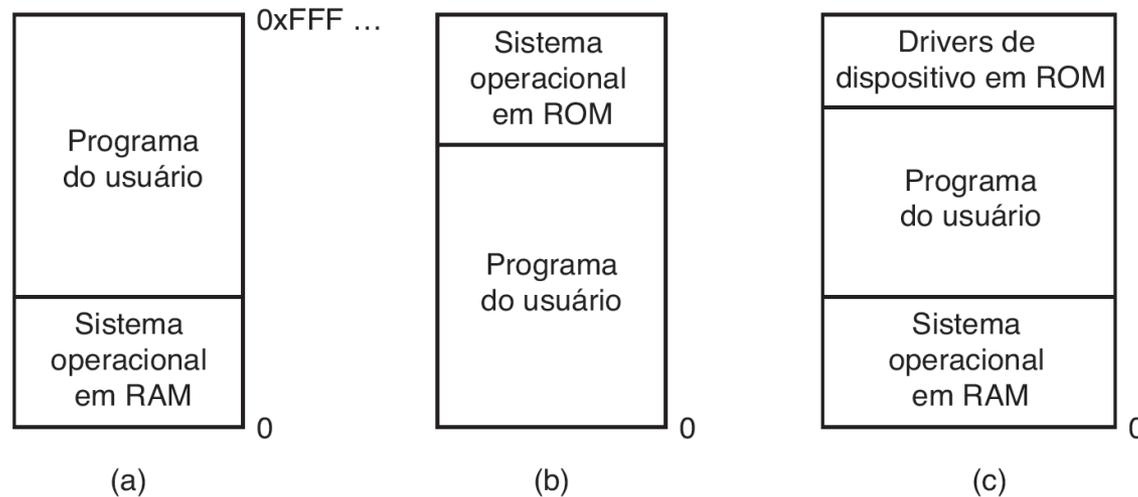


Figura 3.1 Três modos simples de organizar a memória com um sistema operacional e um processo de usuário. Também existem outras possibilidades.

Múltiplos programas em memória

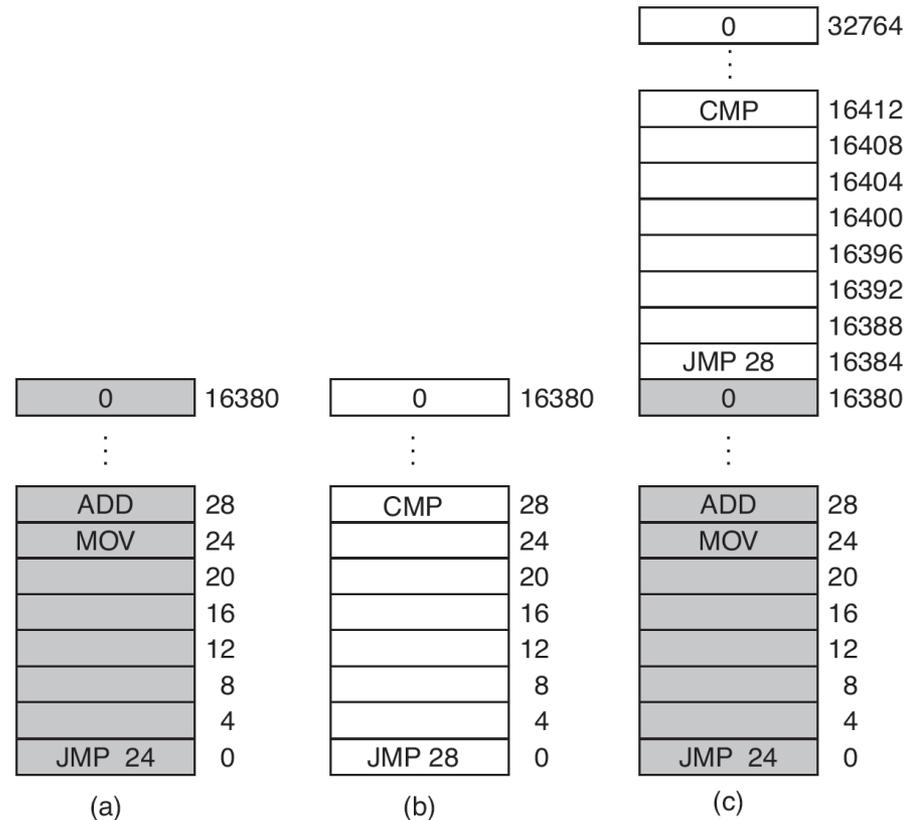


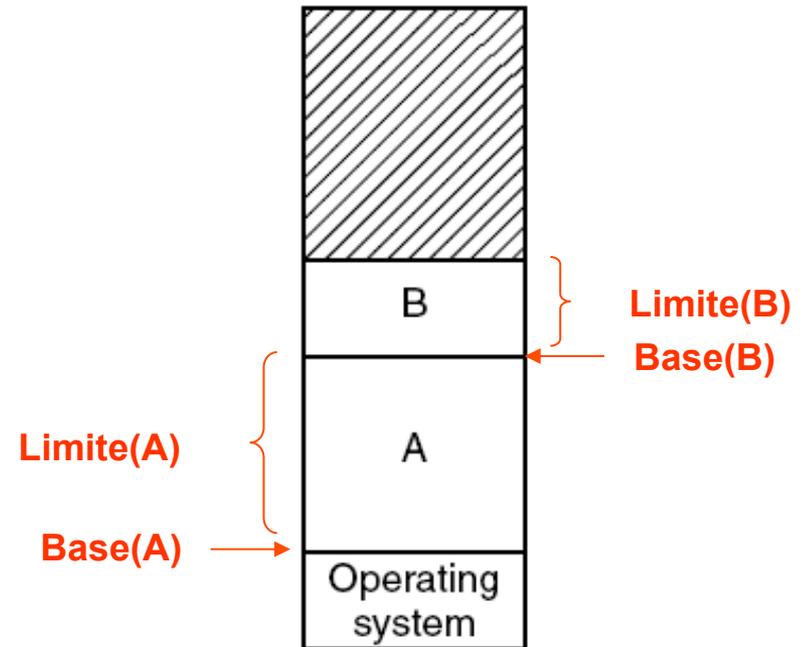
Figura 3.2 Ilustração do problema de realocação. (a) Um programa de 16 KB. (b) Outro programa de 16 KB. (c) Os dois programas carregados consecutivamente na memória.

Realocação e Proteção

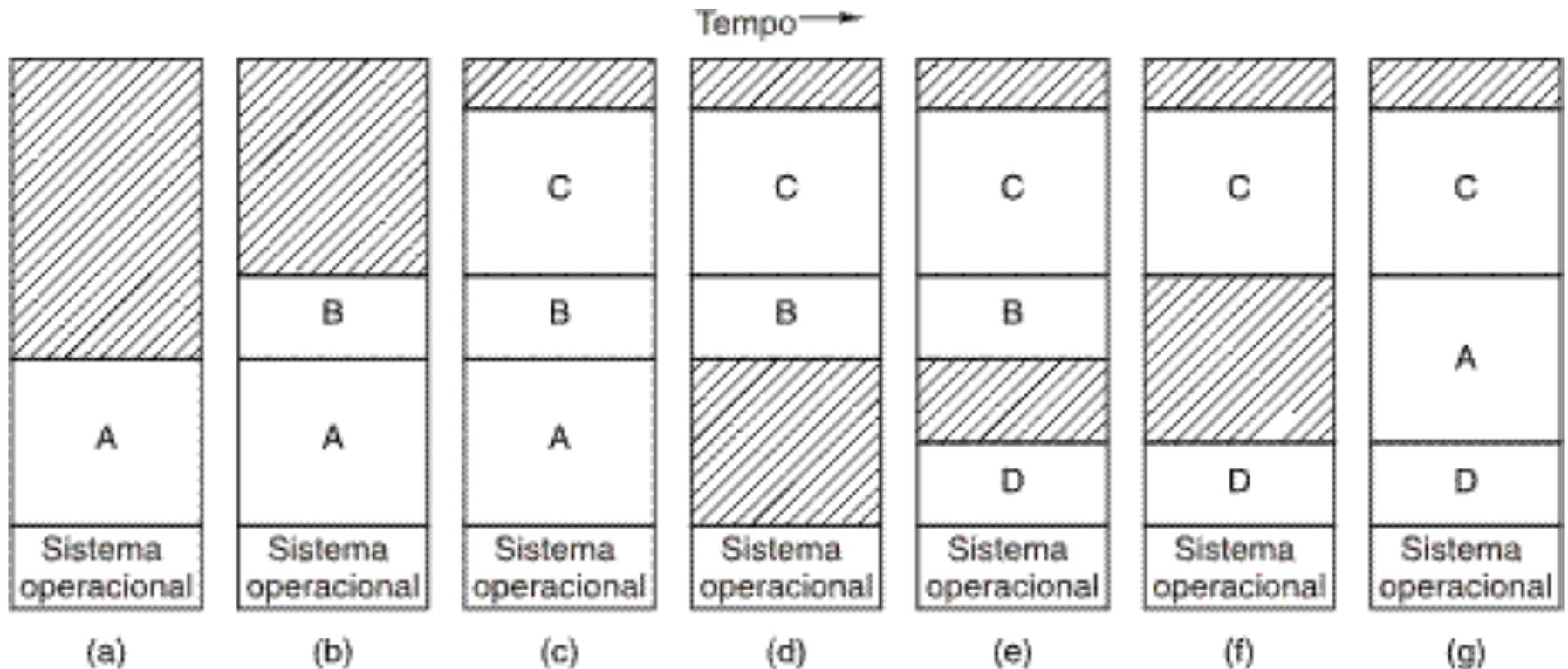
- Realocação: não se sabe com certeza onde o programa será carregado na memória
 - Localizações de endereços de variáveis e de código de rotinas não podem ser absolutos
- Proteção: evitar que um processo acesse uma região usada por outro processo
- Uma solução para realocação e proteção: uso de **valores base** e **limite**

Registradores Base e Limite

- Usados para dar a cada processo um **espaço de endereçamento** separado (**protegido**) – partição
- Base = início da partição
- Limite = tamanho da partição



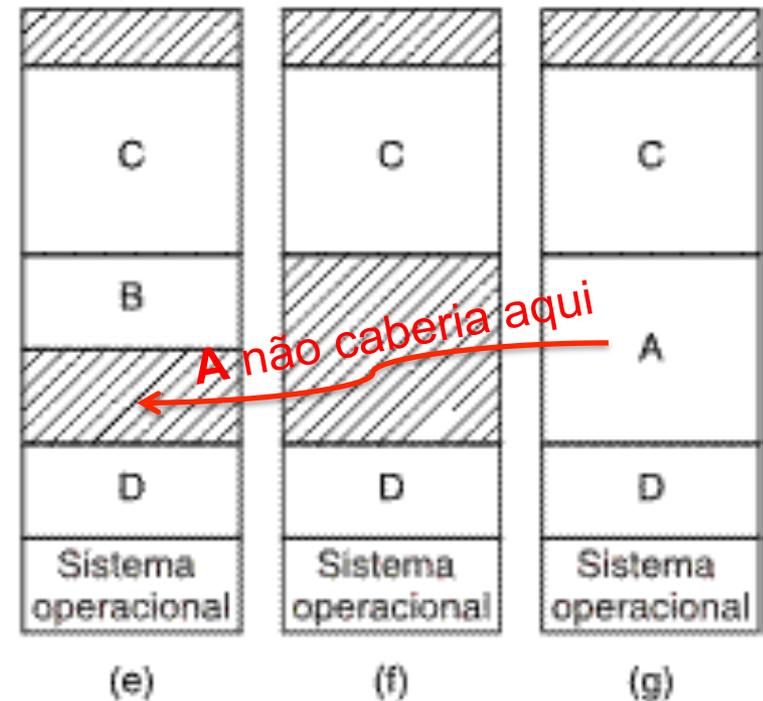
Swapping: Troca de Processos (1)



- Alterações na alocação de memória à medida que processos entram e saem da memória
- Regiões sombreadas correspondem a regiões de memória não utilizadas naquele instante

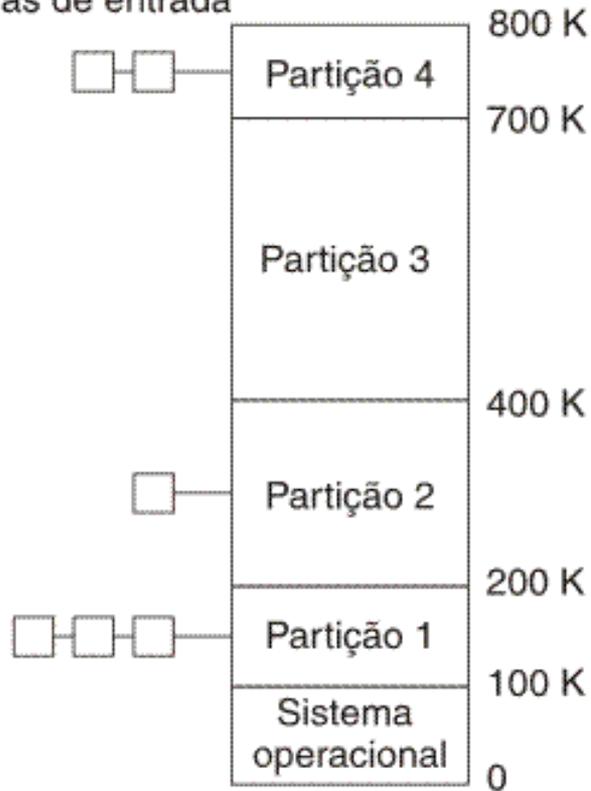
Troca de Processos (2)

- Principal problema do swapping: *fragmentação externa* da memória (espaços pequenos não utilizados)
- **Fragmentação**: parte da memória desperdiçada
 - *Frag. externa*: quando não há partições
 - *Frag. interna*: quando há partições fixas (a seguir)
- Compactação/defragmentação de memória é muito custosa

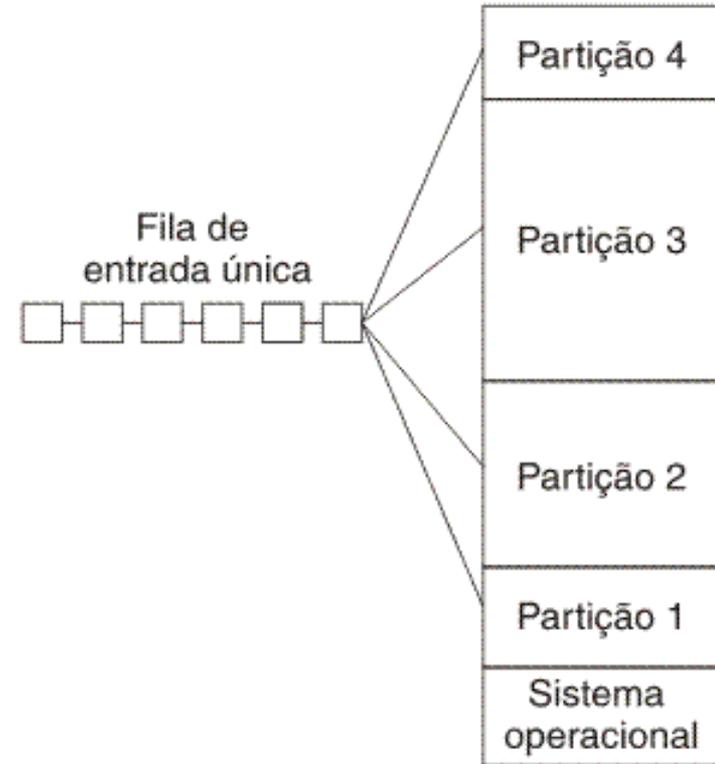


Alocação de Processos na Memória: Multiprogramação com Partições Fixas

Múltiplas filas de entrada



(a)

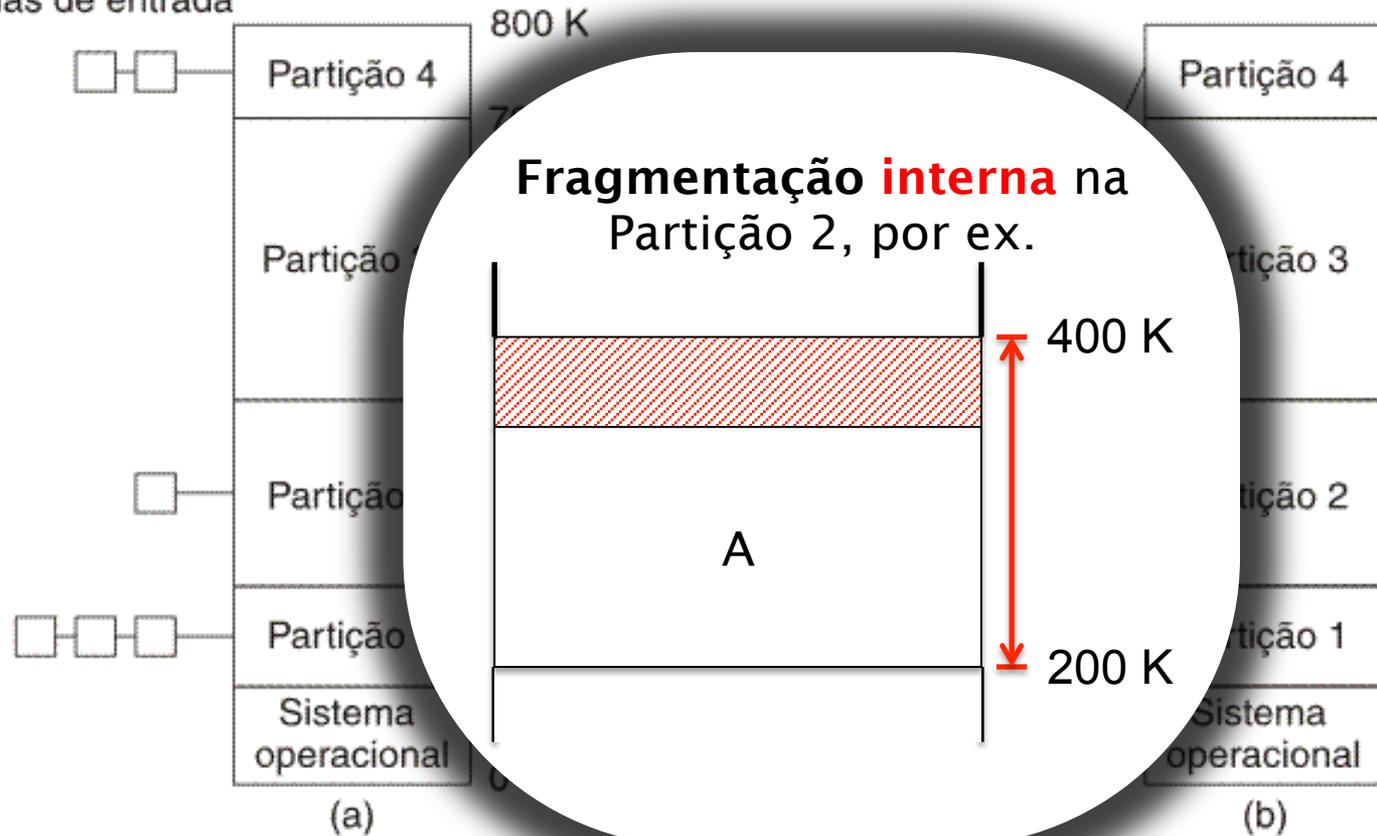


(b)

- Partições fixas de memória
 - a) filas de entrada separadas para cada partição
 - b) fila única de entrada

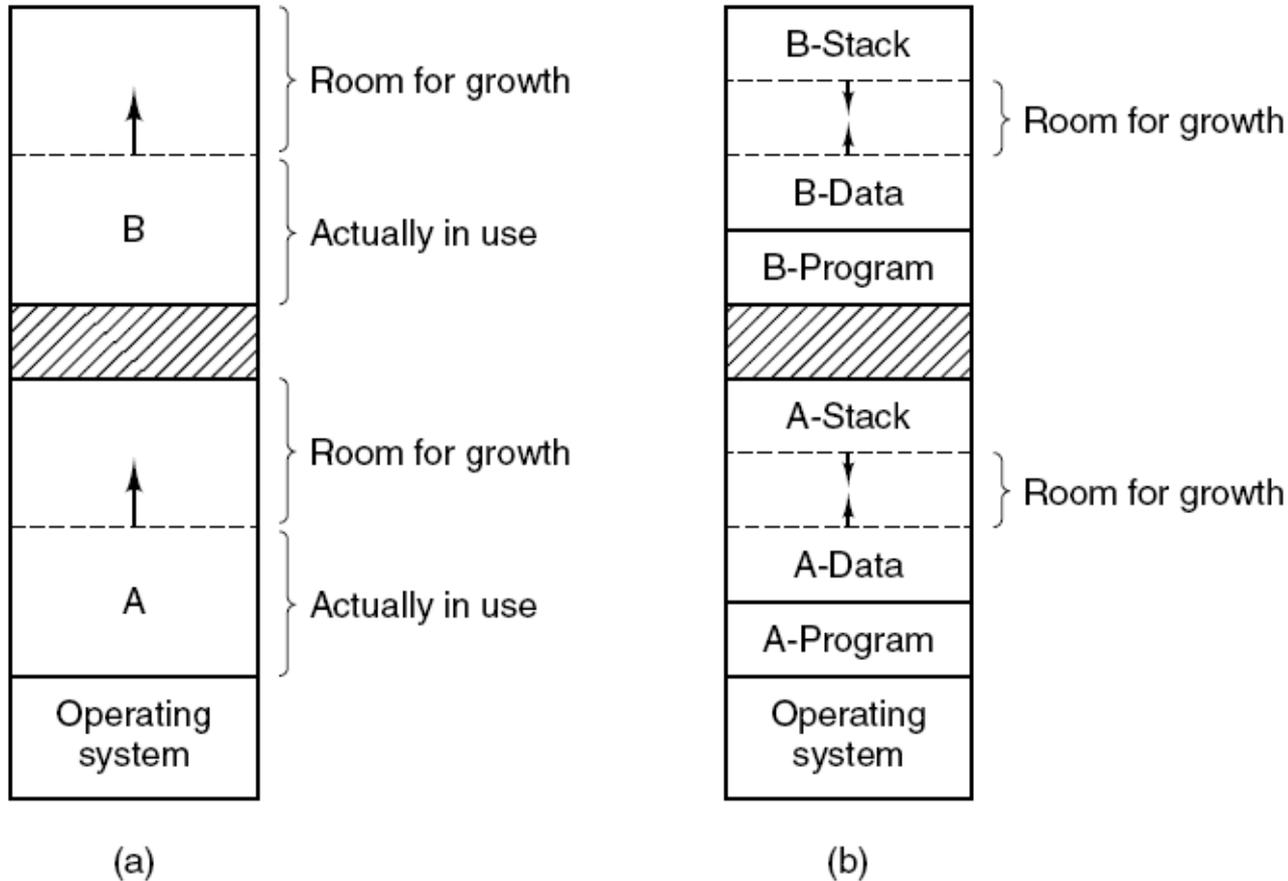
Alocação de Processos na Memória: Multiprogramação com Partições Fixas

Múltiplas filas de entrada



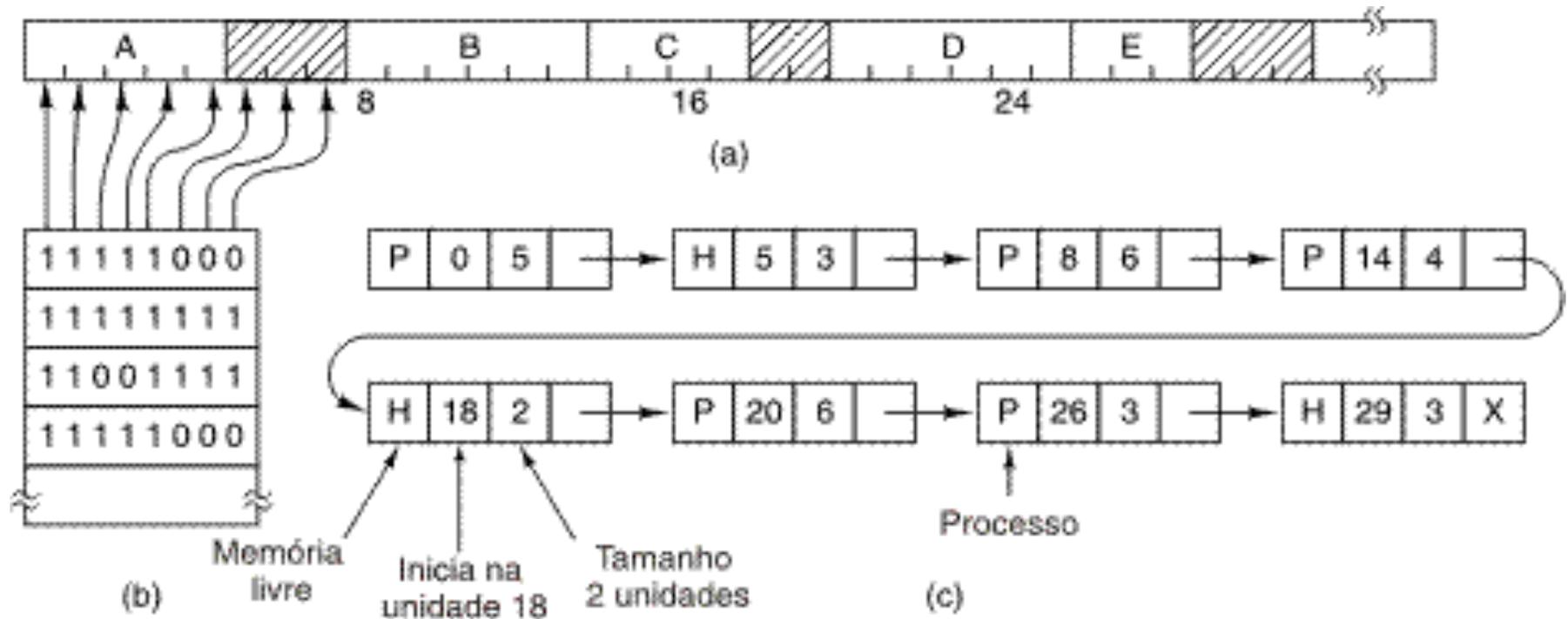
- Partições fixas de memória
 - a) filas de entrada separadas para cada partição
 - b) fila única de entrada

Troca de Processos (4)



- a) Alocação de espaço para uma área de dados em expansão
- b) Alocação de espaço para uma pilha e uma área de dados, ambos em expansão

Gerenciamento de Memória com Mapas de Bits

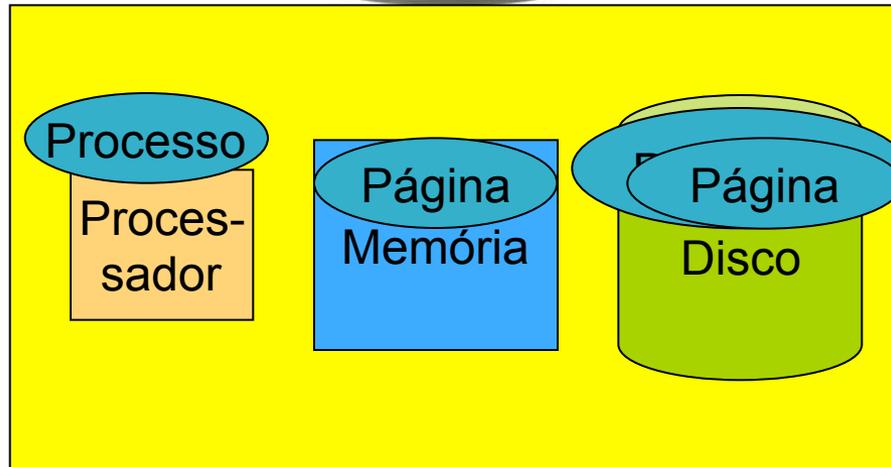


- a) Parte da memória com 5 segmentos de processos (P) e 3 segmentos de memória livre (*Hole* - H)
- b) Mapa de bits correspondente
- c) Mesmas informações em uma lista encadeada

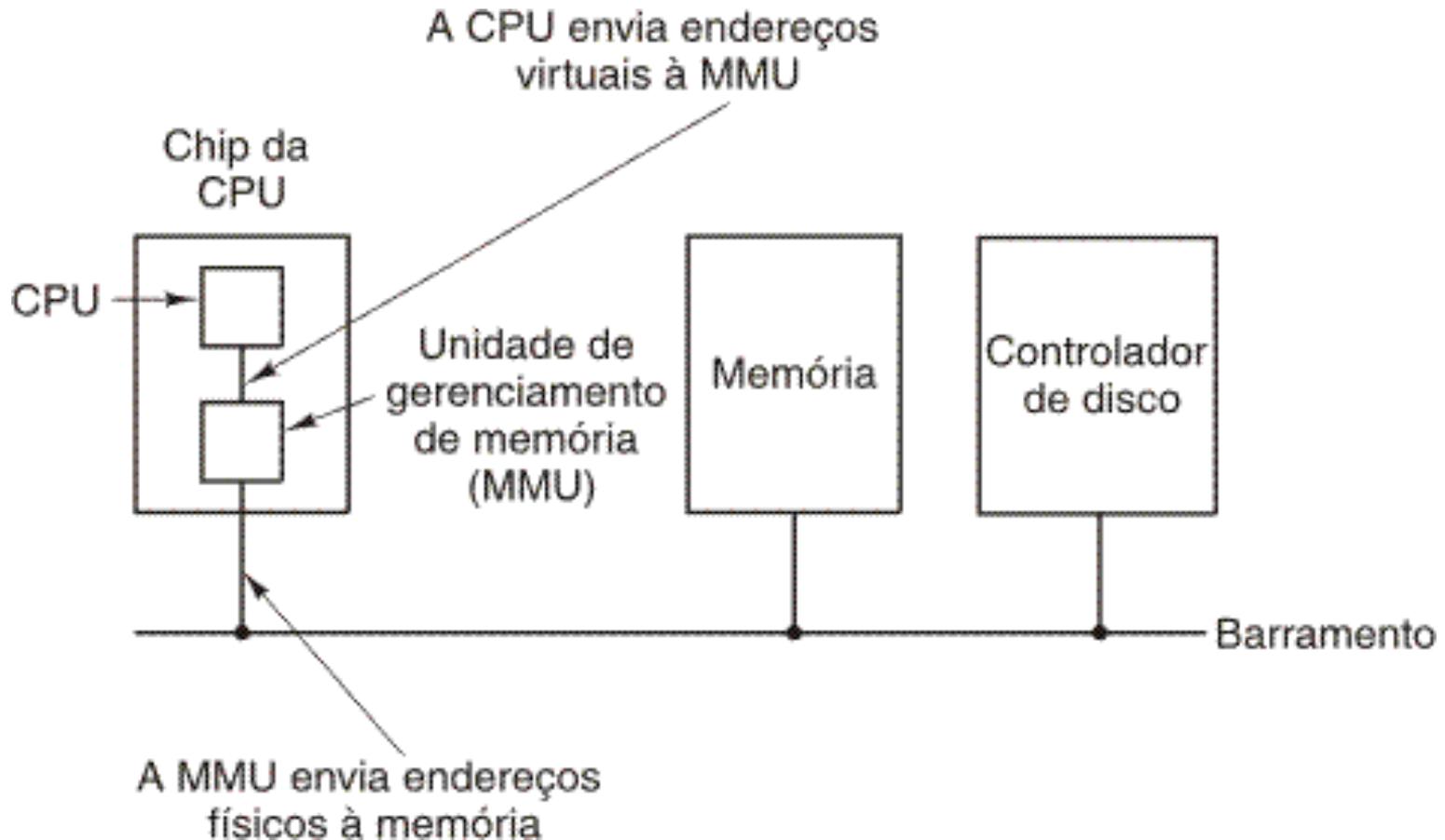
Software

Como rodar um programa se ele for maior do que o espaço de memória disponível?

Os conceitos de “Página” e “Memória Virtual”



Memória Virtual Paginação (1)



Localização e função da **MMU (Memory Management Unit)**: nos dias de hoje é comum se localizar no chip da CPU

Gerenciamento de Memória

- ✓ Alocação e proteção
- ✓ Swapping
- ✓ Mapeamento da memória: mapa de bits, lista encadeada
- ✓ Memória virtual e página
- Paginação: troca de páginas entre memória virtual e memória real (física)

Memórias virtual e real (física)

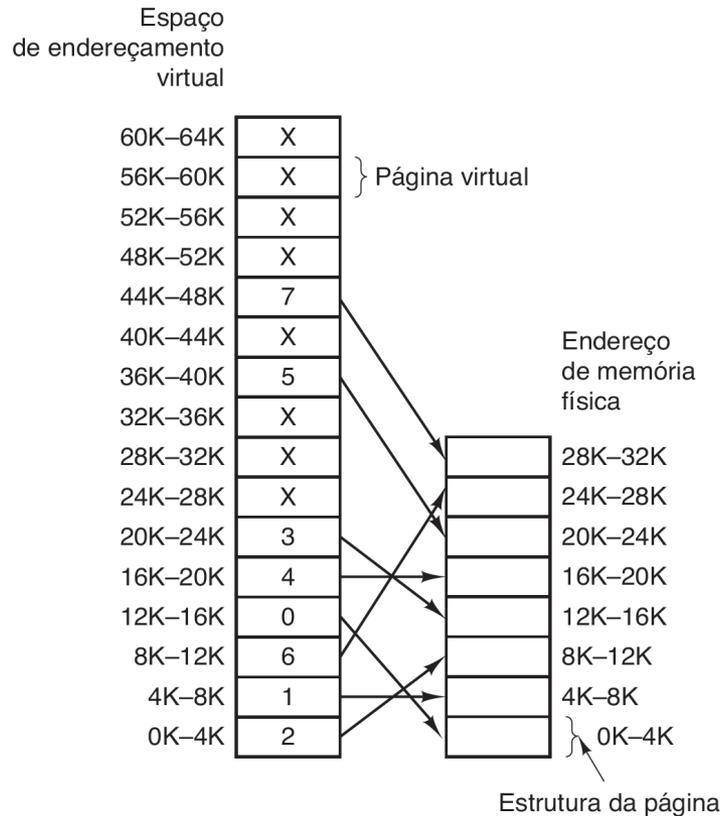


Figura 3.9 A relação entre endereços virtuais e endereços de memória física é dada pela tabela de páginas. Cada página começa com um múltiplo de 4096 e termina 4095 endereços acima; assim, 4K-8K na verdade significa 4096-8191 e 8K-12K significa 8192-12287.

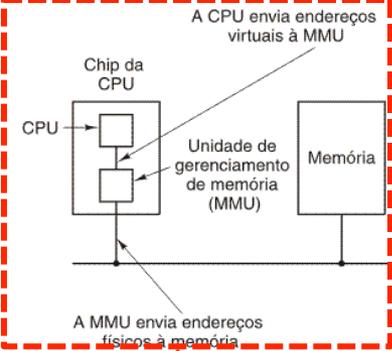
- Neste exemplo simples, temos um computador que gera endereços de 16 bits (0 a 64K [$2^{16} = 65536$]) - **endereços virtuais**
- Mas tem apenas 32KB de memória física, ou seja, programas > 32KB não cabem inteiramente na memória física, mas sim na memória virtual em disco
- O espaço de endereçamento virtual é dividido em **páginas** de tamanho fixo (ex. 4KB)
- Em memória física são chamadas de quadros ou estruturas de páginas (**page frames**), e geralmente são do mesmo tamanho (ex. 4KB)

Tabela de Páginas

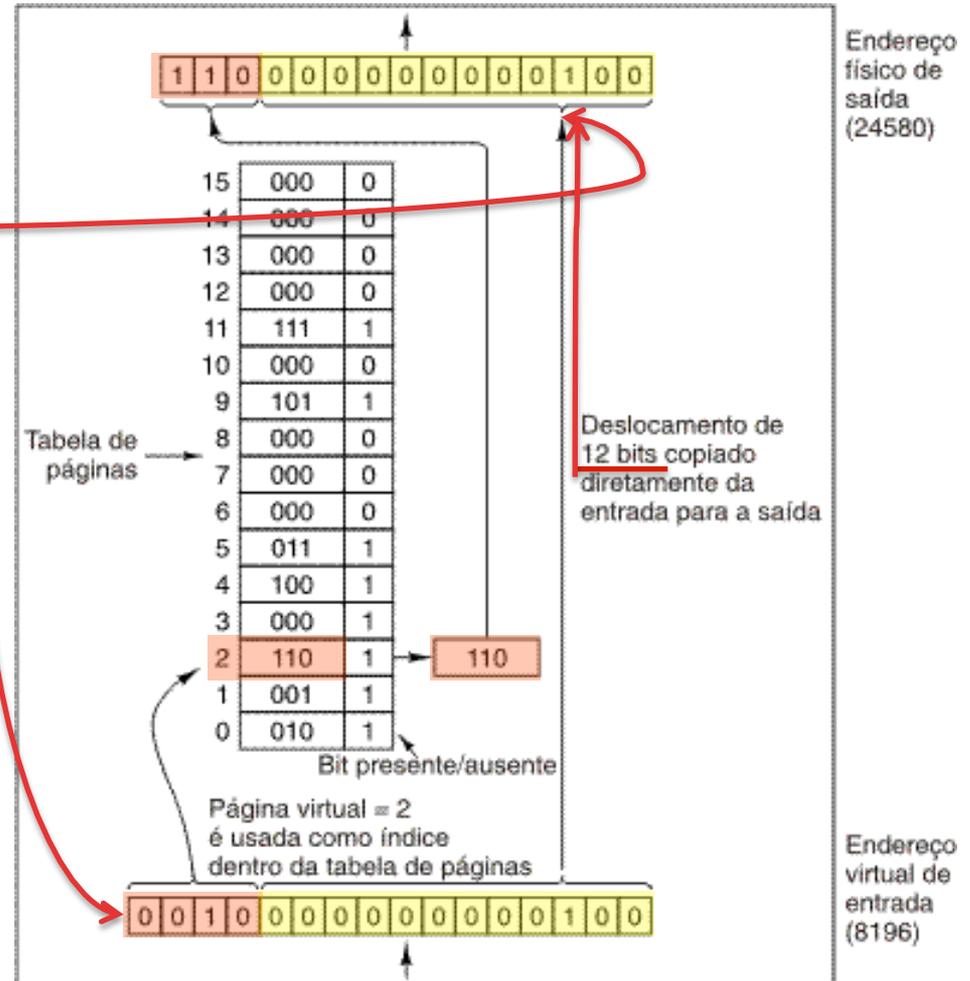
- Uma tabela de página por processo
 - Por isso não precisa armazenar ID de processo na tabela
- Endereço lógico: $\langle p, d \rangle$
- Endereço físico: $\langle f, d \rangle$
- Considerando uma tabela de páginas como um vetor (*array*)
 $f = \text{tabela_páginas}[p]$

p = no. página virtual na tabela
f = endereço do *page frame*
d = deslocamento

Paginação

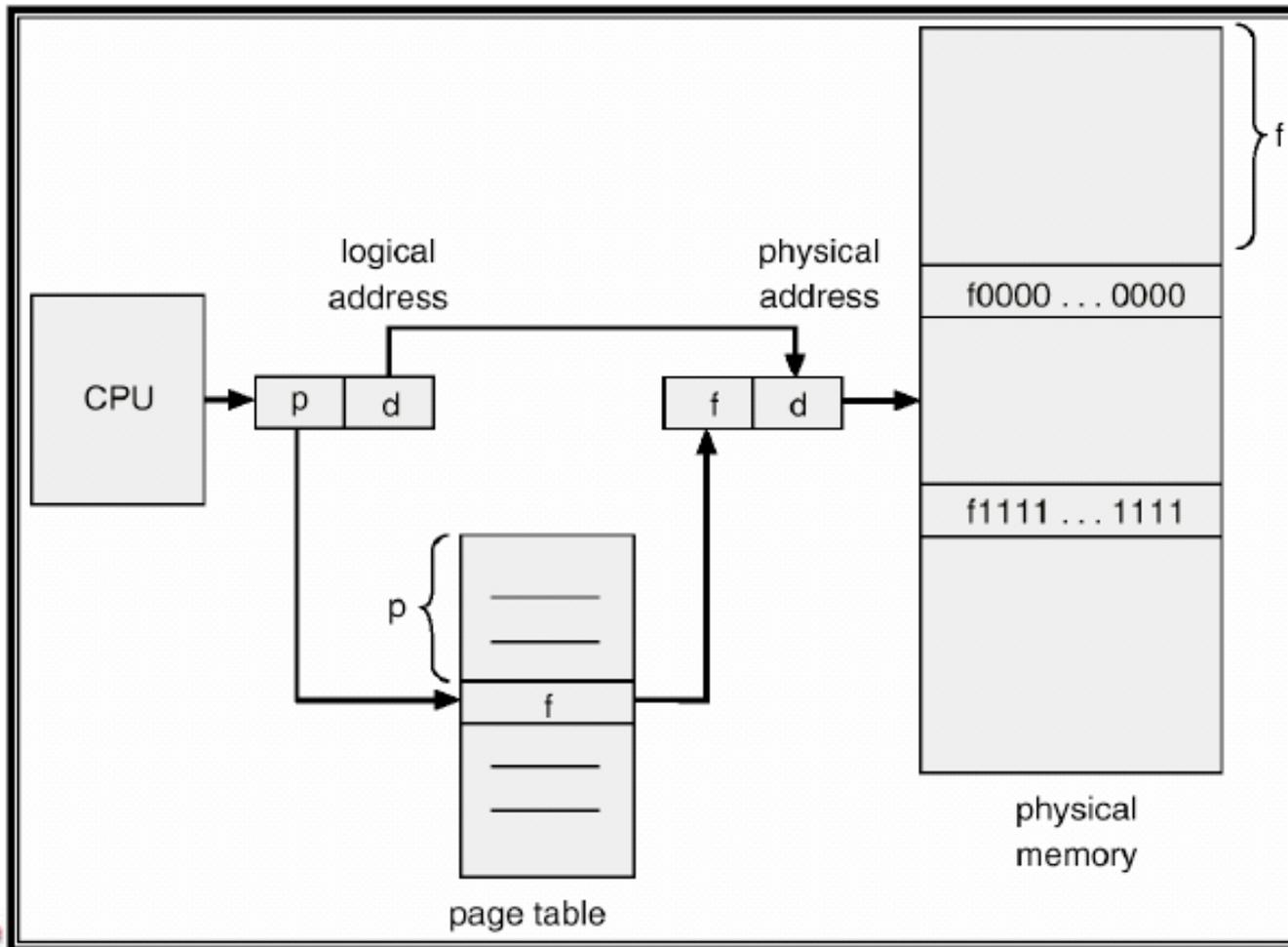


- Operação interna da MMU com 16 (2^4) páginas de 4KB ($4096 = 2^{12}$): determinação do endereço físico de memória em função do endereço virtual vindo da CPU (PC) e da tabela de páginas
- Tabela de páginas fica dentro da MMU



Encontrando um endereço físico dado um endereço lógico

p: no. da página
f: no. do quadro (frame)
d: deslocamento



Assumindo **tamanho (da moldura) de página = 4KB (4096 B)**

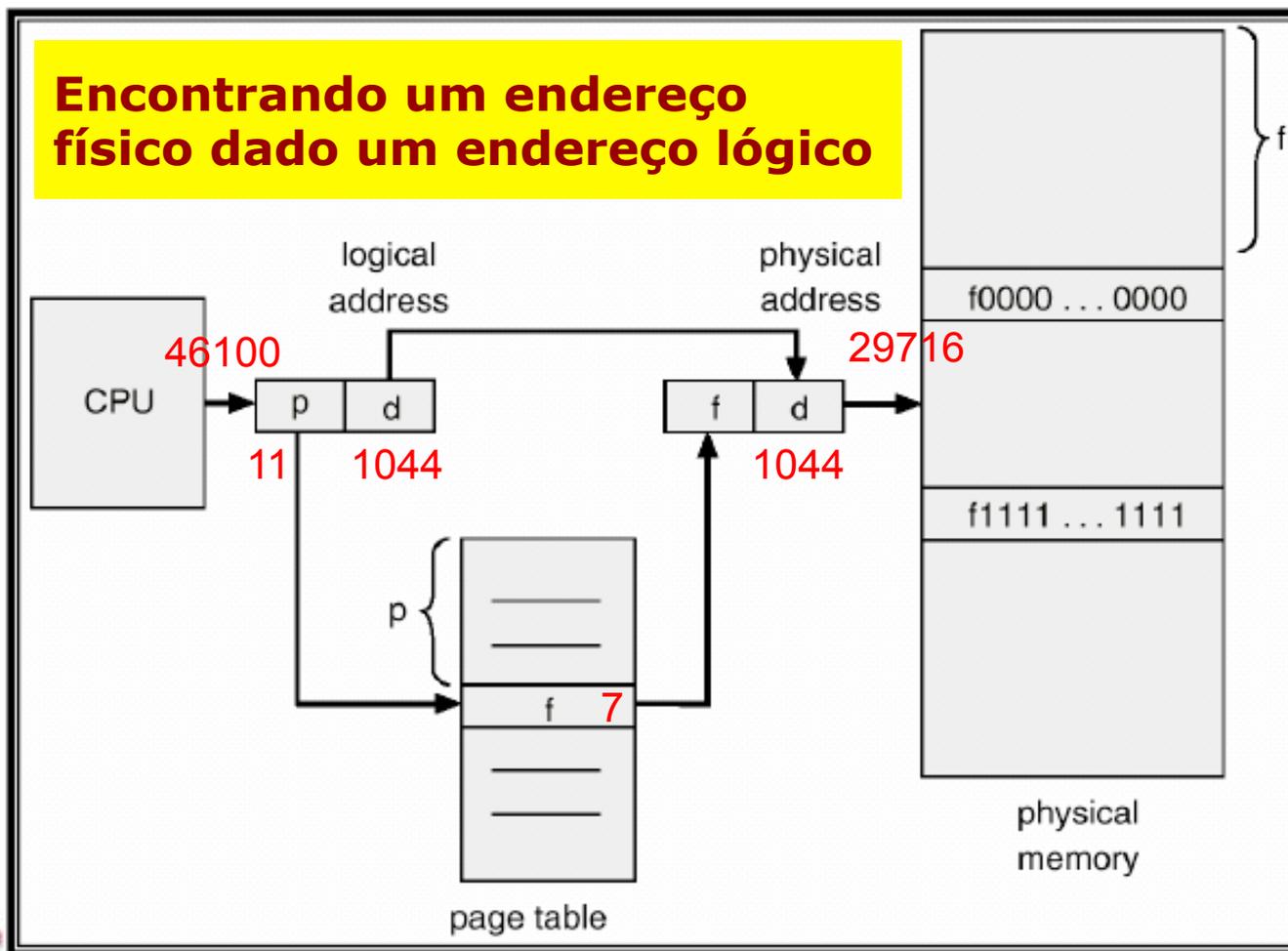
Ex.: endereço de referência (lógico) = **46100**

$p = 46100 \text{ div } 4096 = 11$; $d = 46100 \text{ mod } 4096 = 1044$,

onde **p** é a entrada na tabela de página e **d** é o deslocamento (*displacement*)

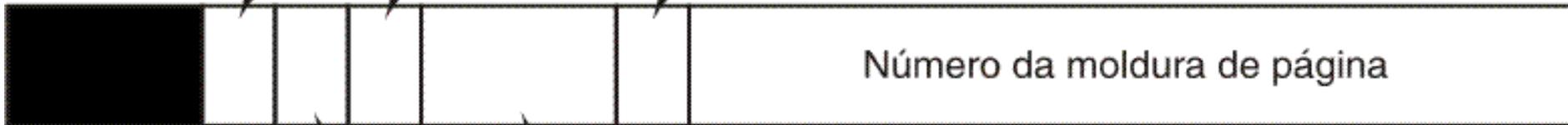
Endereço físico = $f * \text{tamanho de página} + d$, onde **f** é o número da moldura de página física

Endereço físico = $7 * 4096 + 1044 = 29716 \Rightarrow$ Endereço na moldura 7 da memória (física)



Entrada típica de uma tabela de páginas

Cache
desabilitado Modificada Presente/ausente



Referenciada Proteção

Calculando o tamanho de uma tabela de páginas

- Considerando um sistema com:
 1. logical address space: 32-bit
 2. page size: 4KB (2^{12})
 3. page table entry size: 4 bytes (32 bits)
 4. physical memory: 2GB
- **Tamanho de uma tabela de páginas = no._entradas * tamanho_entrada**

Calculando o tamanho de uma tabela de páginas (cont.)

- Um endereço lógico de 32 bits, considerando um tamanho de páginas de 4KB (2^{12}), é particionado como:
 - $\langle 2^{32}/2^{12}$ bits por no. página, 12 bits de deslocamento dentro da página $\rangle = \langle 2^0$ bits por no. página, 12 bits de deslocamento \rangle
- Assim, o número de entradas na tabela de páginas (total de números de páginas virtuais) = 2^{20}
- Ou seja, o tamanho de uma tabela de páginas = $\text{no.entradas} * \text{tamanho_entrada} = 2^{20} * 4 \text{ bytes} = 4.194.304 = 4\text{MB}$ (por processo)

No sistema considerado suponha **100 processos**, por exemplo: se cada processo possui uma tabela de páginas de 4MB, apenas as tabelas ocuparão **400MB** de um total de 2GB, ou seja, cerca de 20% do espaço da memória física...

Tópicos

- ✓ Gerenciamento básico de memória
- ✓ Troca de processos
- ✓ Memória virtual
- ✓ Paginação
 - Aceleração da paginação
 - Substituição de páginas
 - Segmentação

Acelerando a Paginação

1. O mapeamento de endereço virtual para endereço físico deve ser rápido
2. Se o espaço de endereçamento virtual for grande, a tabela de páginas será grande...

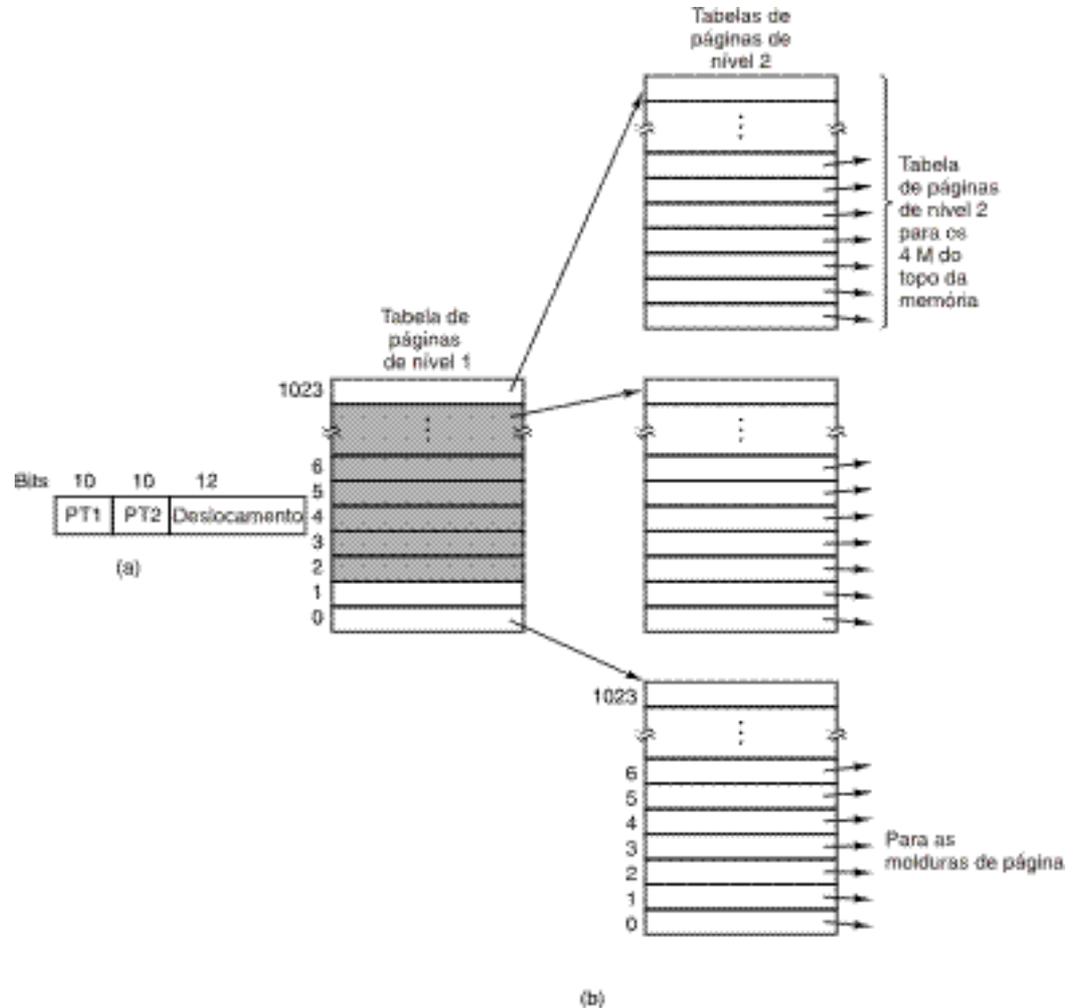
Memória Associativa ou TLB (Translation Lookaside Buffers)

- Tabela das traduções de endereços mais recentes
- Funciona como uma *cache* para tabelas de página

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Tabelas de Páginas Multi-Níveis

- Para minimizar o problema de continuamente armazenar tabelas de páginas muito grandes na memória

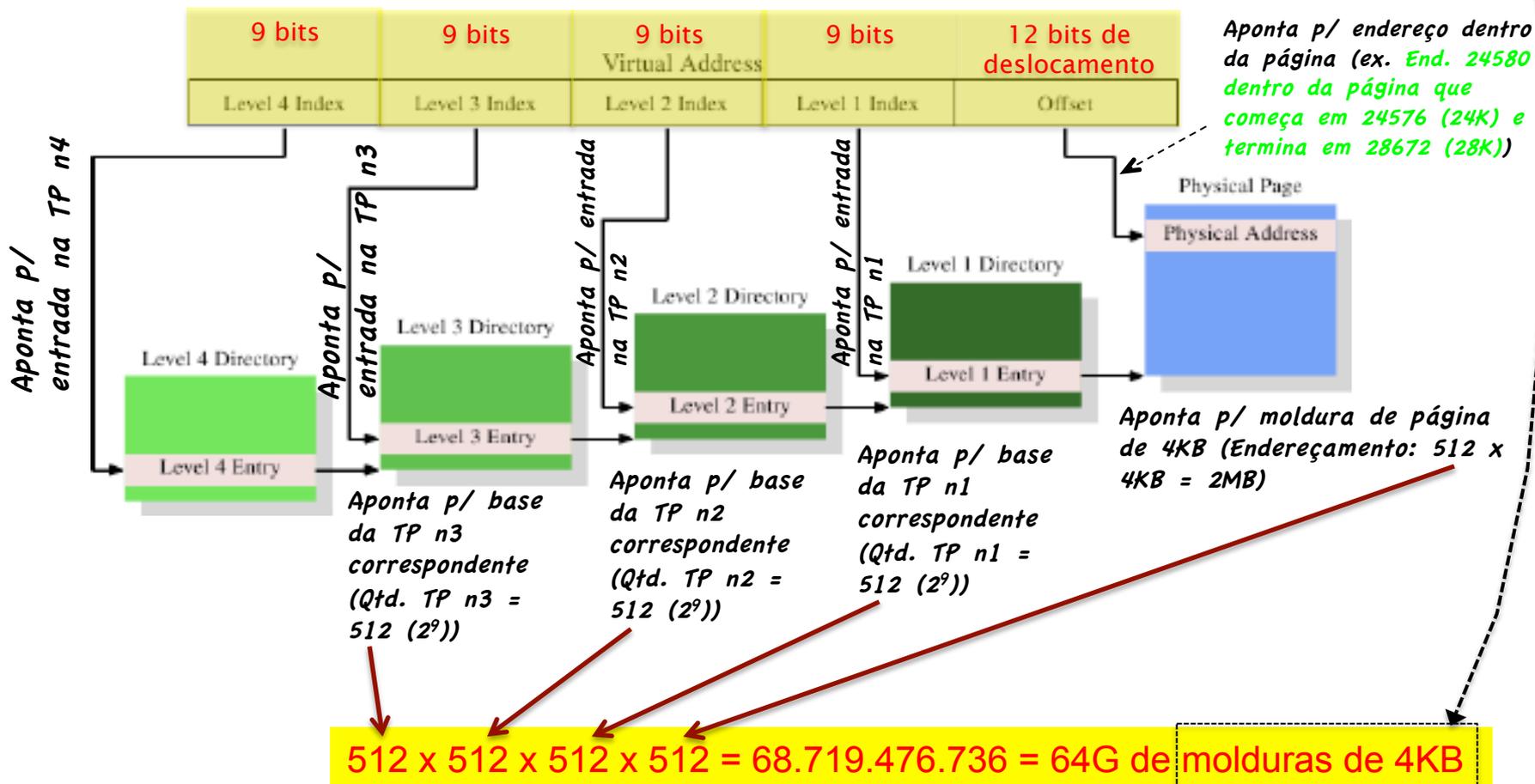


- Endereço de 32 bits com 2 campos (Page Table - PT1, PT2) para endereçamento de tabelas de páginas
- Tabelas de páginas com 2 níveis

Intel x86-64: tabelas de páginas em 4 níveis

Apenas 48 bits (dos 64) são verdadeiramente utilizados

$$2^{12} = 4096 = 4K$$



Intel x86-64: tabelas de páginas em 4 níveis (cont.)

Endereço virtual

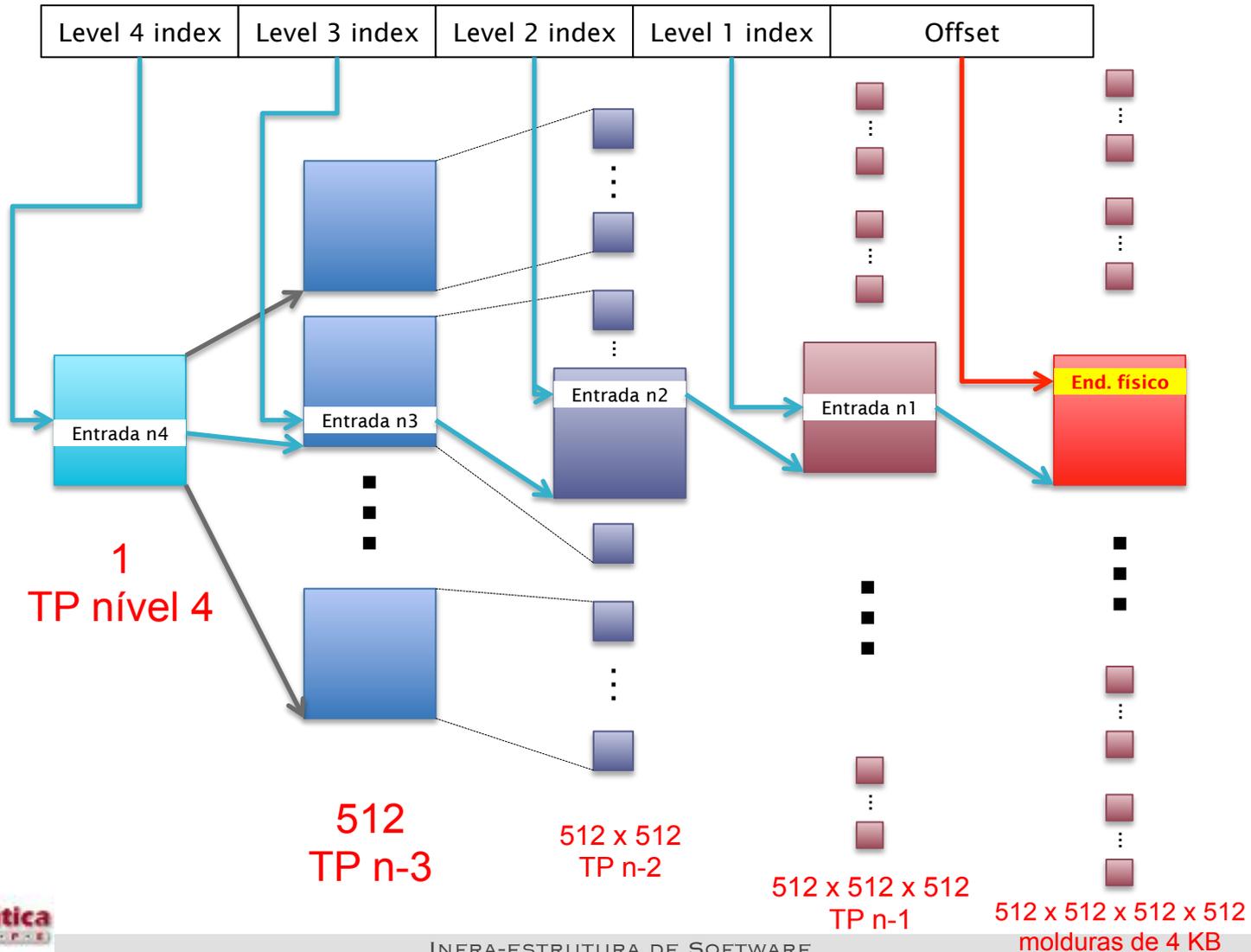


Tabela de páginas invertidas

- Objetivo: reduzir a quantidade de memória física necessária para armazenar tabelas de páginas (lembrando: uma tabela por processo)
- Uma **tabela de páginas invertidas** é uma tabela de páginas *global* para todos os processos – apenas uma tabela de páginas para todo o sistema
- Informações adicionais precisam ser armazenadas na tabela de páginas (invertidas) para identificar entradas correspondentes a cada processo

Tabela de páginas invertidas (cont.)

- Antes, tabela de páginas: $f = \text{tabela_p\u00e1ginas}[p]$
- Tabela de páginas invertidas: $\langle \text{pid}, p \rangle = \text{tabela_p\u00e1ginas}[f]$
- Considerando o mesmo sistema anterior (tabela de páginas) com:
 1. logical address space: 32-bit
 2. page size: 4KB (2^{12})
 3. page table entry size: 4 bytes (32 bits)
 4. **physical memory**: 2GB (2^{31})
- **Tamanho de uma tabela de páginas = no._entradas * tamanho_entrada**
 - **no._entradas** = número de páginas físicas = $2^{31}/2^{12} = 2^{19}$
- Considerando que um PID ocupa 1 byte (8 bits),
 - **tamanho_entrada** = 8 bits (PID) + 20 bits (virtual page number) + 4 bits (access information) = 32 bits = 4 bytes
- Ou seja, o **tamanho de uma tabela de páginas** = $2^{19} * 4 \text{ bytes} = 2.097.152 = 2\text{MB}$ (para o sistema inteiro)

Tabela de páginas invertidas (cont.)

- Antes, tabela de páginas: $f = \text{tabela_p\u00e1ginas}[p]$
- Tabela de páginas invertidas: $\langle \text{pid}, p \rangle = \text{tabela_p\u00e1ginas}[f]$
- Considerando o mesmo sistema anterior (tabela de páginas) com:
 1. Antes, **100 processos**, 100 tabelas de páginas de 4MB ocupando **400MB** de um total de 2GB, ou seja, cerca de **20% do espa\u00e7o da mem\u00f3ria f\u00edsica...**
 2. Com tabela de páginas invertidas: **2MB** de um total de 2GB, ou seja, cerca de **0,1% do espa\u00e7o da mem\u00f3ria f\u00edsica**
 3. Mas **tabela de páginas invertidas n\u00e3o \u00e9 boa op\u00e7\u00e3o para compartilhamento...**
 4. Considerando que ocupa 1 byte (8 bits),
 - tamanho_entrada (PID) + 20 bits (virtual page number) + 4 bits (access information) = 32 bits = 4 bytes
- Ou seja, o **tamanho de uma tabela de páginas** = $2^{19} * 4 \text{ bytes} = 2.097.152 = 2\text{MB}$ (para o sistema inteiro)

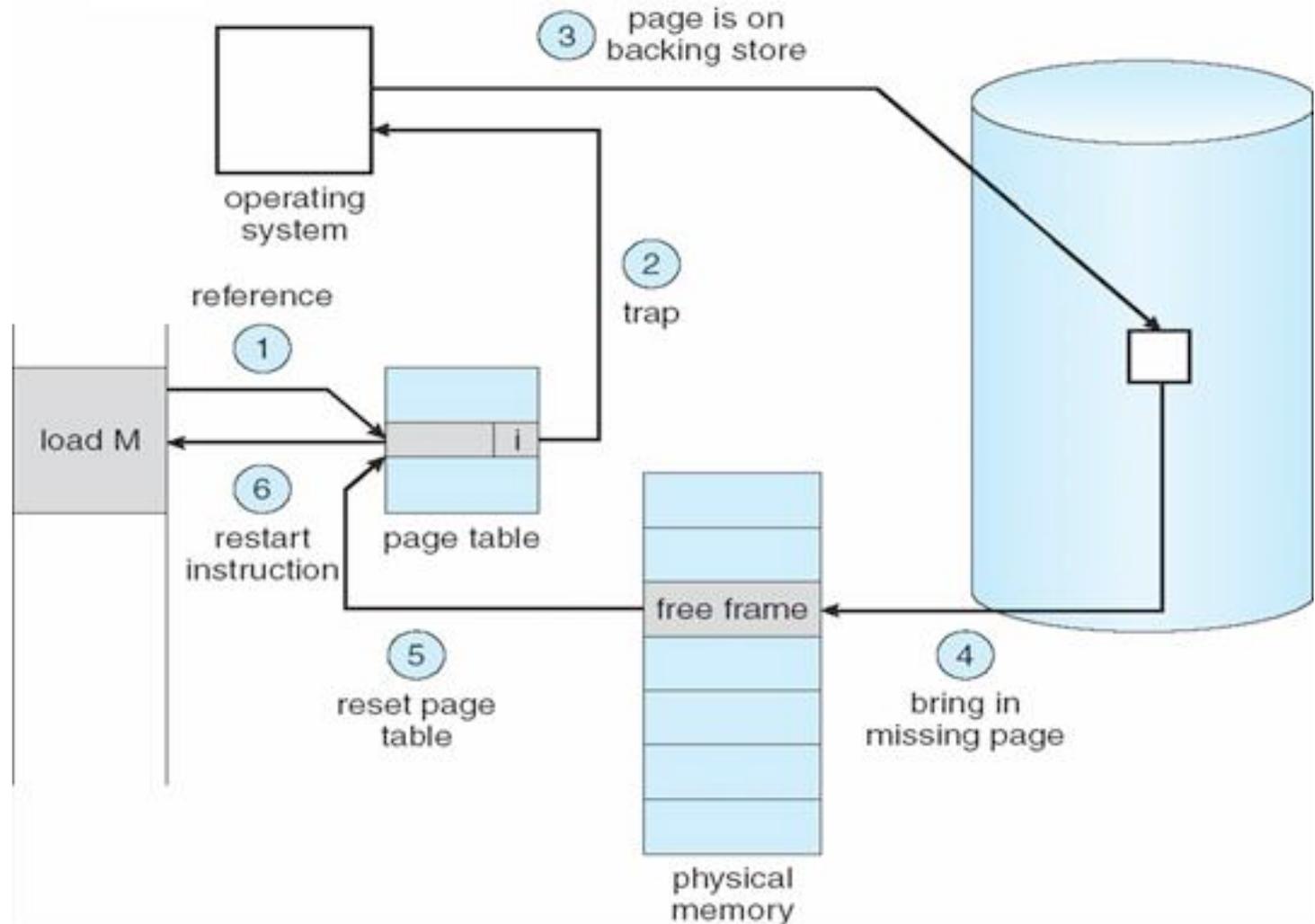
Tabela de páginas invertidas (cont.)

- Tabela de páginas invertidas **ocupa espaço significativamente menor**, mas...
- a tradução de endereços lógicos → físicos fica **bem mais lenta**, i.e.
 - para cada página p , precisa-se fazer uma varredura pela tabela de páginas invertidas para encontrar o quadro correspondente
- Só é viável, se TLB for usado para guardar as associações correntes
 - somente quando ocorre um *TLB miss*, a tabela de páginas invertidas precisa ser pesquisada
- Usa-se também uma função de *hash* para indexar as entradas da tabela de páginas invertidas

Tópicos

- ✓ Gerenciamento básico de memória
- ✓ Troca de processos
- ✓ Memória virtual
- ✓ Paginação
- ✓ Aceleração da paginação
 - Substituição de páginas
 - Segmentação

Passos para lidar com Falta de Página



Desempenho de Paginação

- **Page Fault Rate:** $0 \leq p \leq 1.0$
 - if $p = 0$, no page faults
 - if $p = 1$, every reference is a fault
- **Effective Access Time (EAT)**
- $EAT = (1 - p) \times \text{memory access} + p \times (\text{page fault overhead})$
- **overhead** = swap page out + swap page in + restart overhead

Exemplo

- Memory access time = 200 nanoseconds
- Average overhead time = 8 milliseconds
- $EAT = (1 - p) \times 200 \text{ (ns)} + p \times 8 \text{ (ms)}$
- If 1 (one) access out of 1,000 causes a page fault, then
 - $EAT \approx 8200$ nanoseconds.
 - This is a slowdown by a factor of 40!! [$40 \times 200 = 8000$]

Substituição de Páginas

- Falta de página (*page-fault*) na memória:
 - qual página deve ser removida?
 - alocação de espaço para a página a ser trazida para a memória
- A página modificada deve primeiro ser salva
 - se não tiver sido modificada é apenas sobreposta
- Melhor não escolher uma página que está sendo muito usada
 - provavelmente precisará ser trazida de volta logo

Substituição de Páginas: Algoritmos

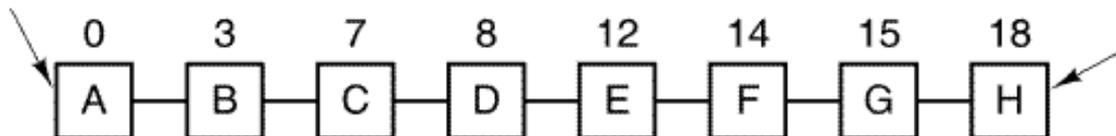
- Ótimo: procura substituir o mais tarde possível - impraticável
- First-In, First-Out (FIFO)
- Not Recently Used (NRU)
- Segunda chance (SC)
- Least Recently Used (LRU)
- Conjunto de trabalho (Working Set - WS)
- Relógio (Clock)
- WSClock

Usado apenas em simulações para avaliar o quanto os algoritmos concretos diferem do algoritmo ideal

Desvantagem FIFO: página há mais tempo na memória (First-In) pode ser usada com muita frequência, mas terá que ser substituída (First-Out)

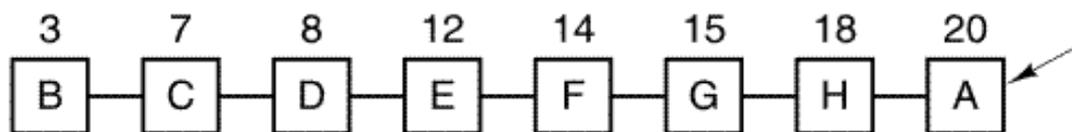
Segunda Chance (SC)

Primeira página carregada



(a)

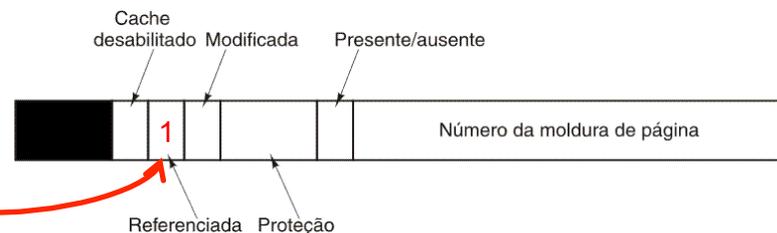
Página mais recentemente carregada



(b)

Página A é tratada como página mais recentemente carregada

- Operação do algoritmo segunda chance
 - lista de páginas em ordem FIFO
 - estado da lista em situação de falta de página no instante 20, com o bit R da página A em 1 - números representam instantes de carregamento das páginas na memória



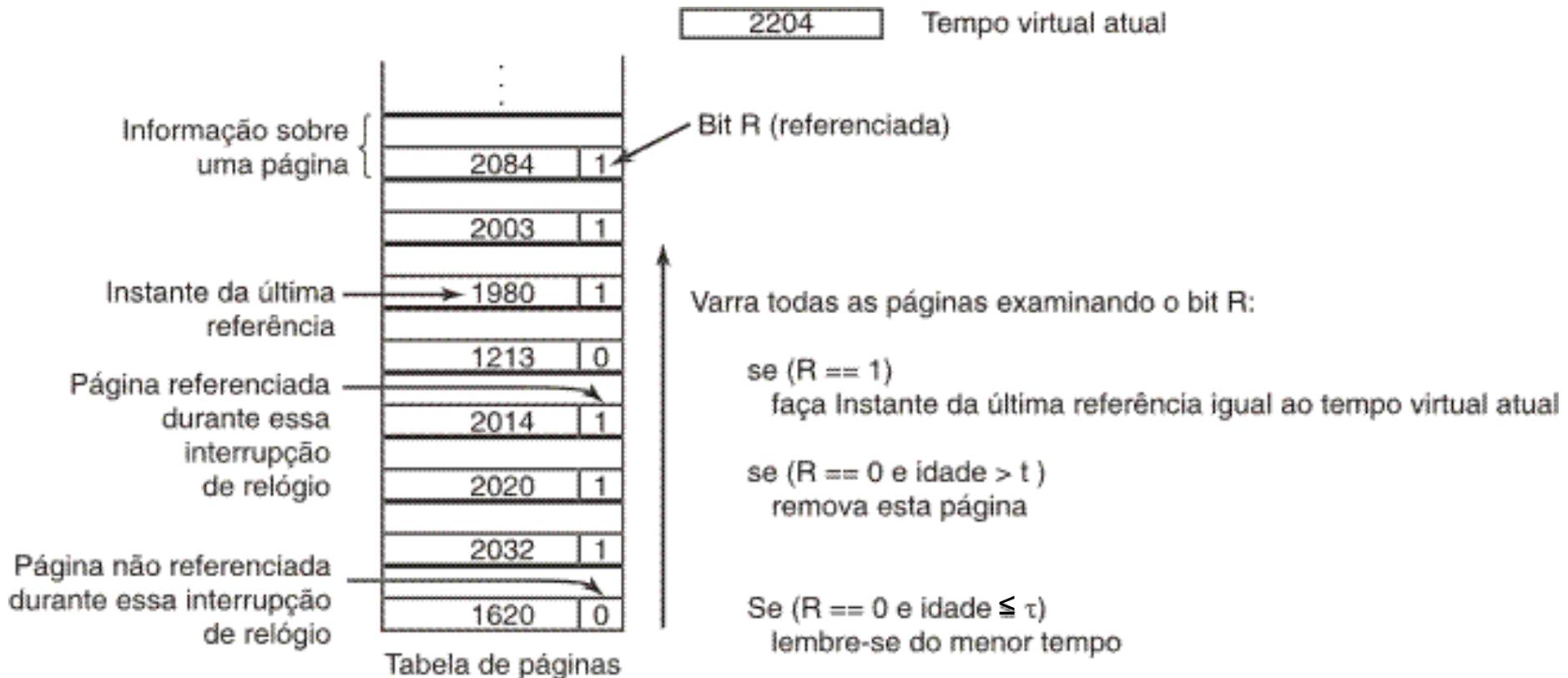
Não Usada Recentemente (NUR/ NRU)

- Cada página tem os bits Referenciada (R) e Modificada (M)
 - Bits são colocados em 1 quando a página é referenciada e modificada
- As páginas são classificadas
 - Classe 0: não referenciada (0), não modificada (0)
 - Classe 1: não referenciada (0), modificada (1)
 - Classe 2: referenciada (1), não modificada (0)
 - Classe 3: referenciada (1), modificada (1)
- NUR remove página aleatoriamente
 - da classe de ordem mais baixa que não esteja vazia

Menos Recentemente Usada (MRU/LRU)

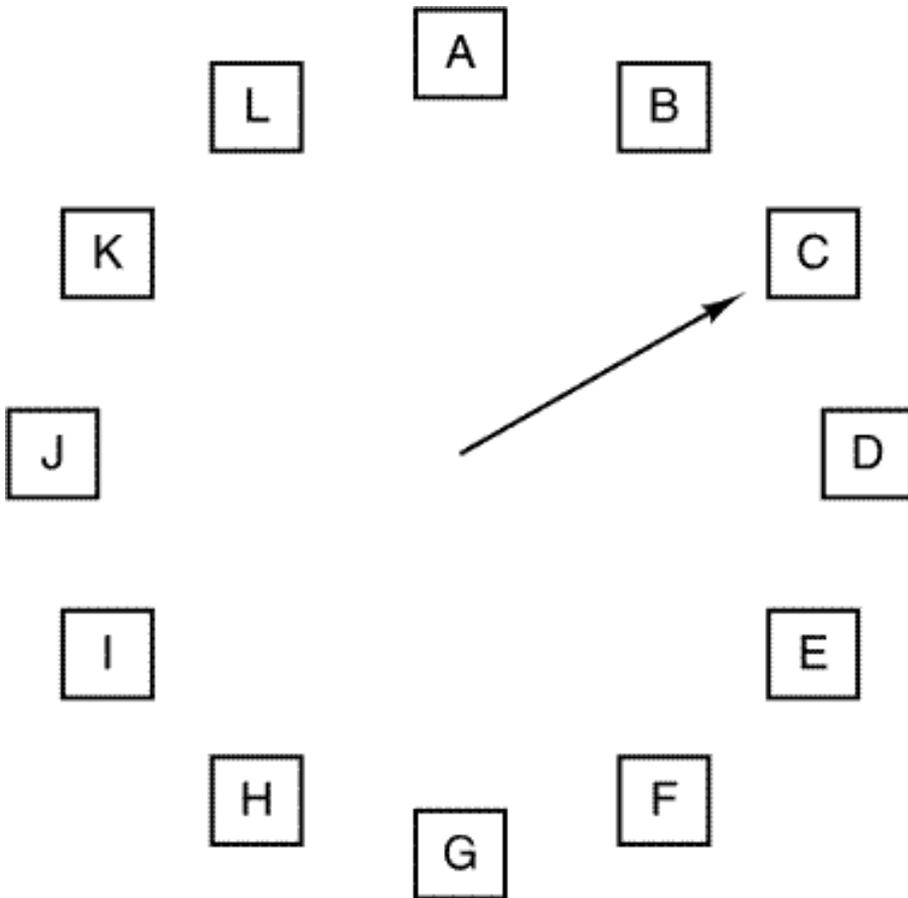
- Assume que páginas usadas recentemente logo serão usadas novamente
 - retira da memória a página que há mais tempo não é usada
- Uma lista encadeada de páginas deve ser mantida
 - página mais recentemente usada no início da lista, menos usada no final da lista
 - atualização da lista à cada referência à memória
- Alternativamente manter contador em cada entrada da tabela de página
 - escolhe página com contador de menor valor
 - zera o contador periodicamente

Conjunto de Trabalho (WS)



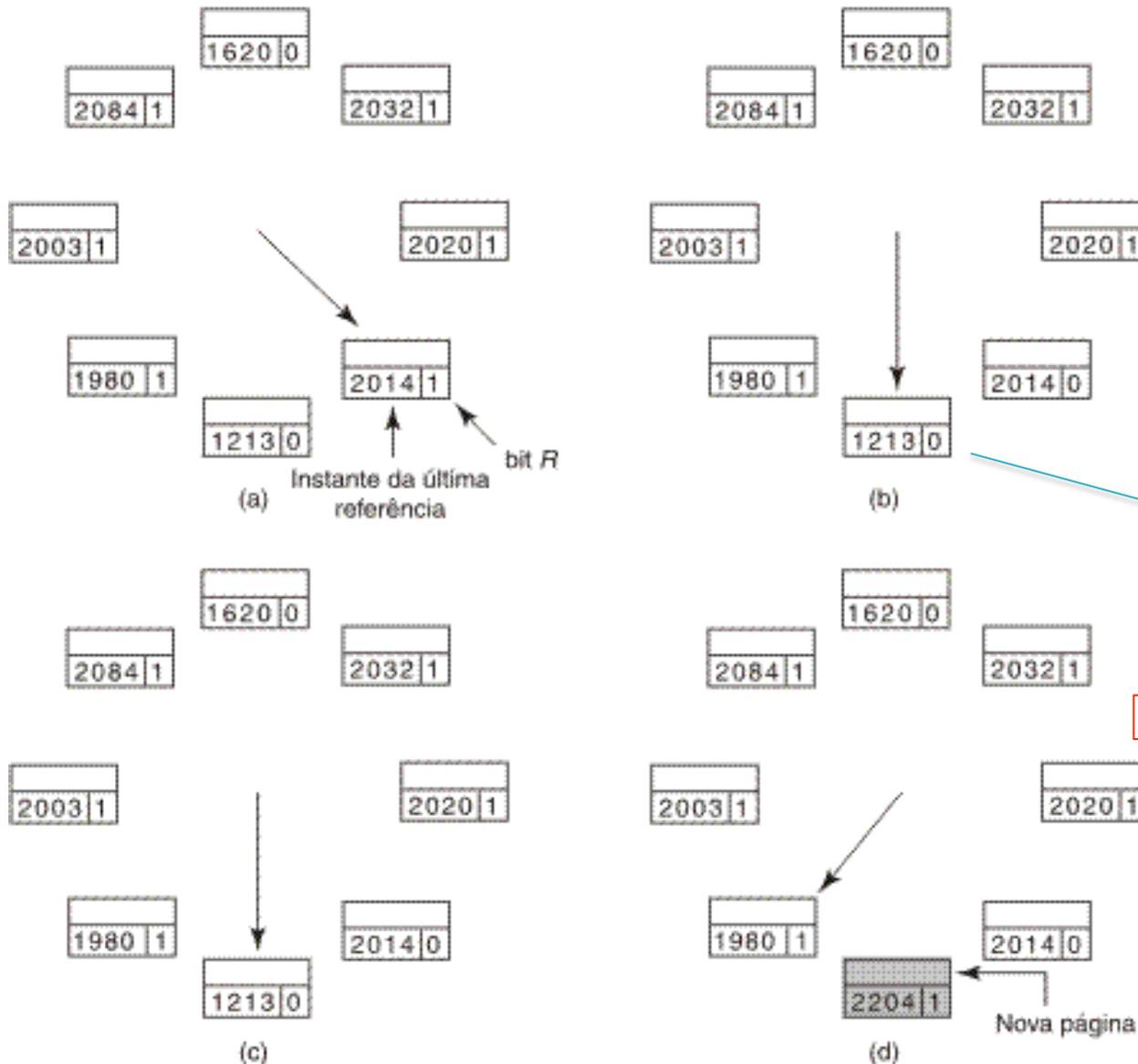
idade = tempo virtual atual – instante da última referência

Relógio



Quando ocorre uma falta de página, a página apontada é examinada.
A atitude a ser tomada depende do bit R:
R = 0: Retira a página,
R = 1: Faz R = 0 e avança o ponteiro.

WSClock



Assim como no WS:

Se $(R==0 \text{ e } idade > t)$
 remova esta página
// pois ela está fora do conjunto de trabalho e há uma cópia válida em disco

$idade = \text{tempo virtual atual} - \text{instante da última referência}$

O Algoritmo Ótimo!

- O algoritmo FIFO sempre seleciona a **página mais antiga** para ser trocada – First-In, First-Out
- O algoritmo LRU sempre seleciona a **página que não vem sendo usada há mais tempo** – Least Recently Used (Menos Recentemente Usada - MRU)
- O algoritmo ótimo sempre seleciona a **página que não será usada por mais tempo...**
 - Mas como o SO pode determinar quando cada uma das páginas será referenciada? Daqui a 10 instruções, 100 instruções, 1000 instruções...
 - **IMPOSSÍVEL!!!**

Revisão dos Algoritmos de Substituição de Página

Algoritmo	Comentário
Ótimo	Não implementável, mas útil como um padrão de desempenho
NUR (não usada recentemente)	Muito rudimentar
FIFO (primeira a entrar, primeira a sair)	Pode descartar páginas importantes
Segunda chance	Algoritmo FIFO bastante melhorado
Relógio	Realista
MRU (menos recentemente usada)	Excelente algoritmo, porém difícil de ser implementado de maneira exata
NFU (não freqüentemente usada)	Aproximação bastante rudimentar do MRU
Envelhecimento (<i>aging</i>)	Algoritmo bastante eficiente que se aproxima bem do MRU
Conjunto de trabalho	Implementação um tanto cara
WSClock	Algoritmo bom e eficiente

No. de molduras de páginas x No. de faltas de página :: Comparação ::

- FIFO

Memória
com 3
molduras
(frames) de
páginas



Memória
com 4
molduras
(frames) de
páginas



Comparação

- FIFO

Solicitações de página=12

Faltas de página=9; taxa de falta=9/12=75%; taxa de sucesso=3/12=25%

3 molduras
(frames) de
páginas

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

4 molduras
(frames) de
páginas

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

Faltas de página=10; taxa de falta=10/12=83,3%; taxa de sucesso=2/12=16,7%

Comparação

- FIFO

Solicitações de página=12

Faltas de página=9; taxa de falta=9/12=75%; taxa de sucesso=3/12=25%

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

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

3 molduras (frames) de páginas

0 1 2 3 0 1 4 0 1 2

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

4 molduras (frames) de páginas

Anomalia de Belady: neste exemplo, o algoritmo de substituição FIFO tem mais faltas de página (10P) para mais molduras (4)...

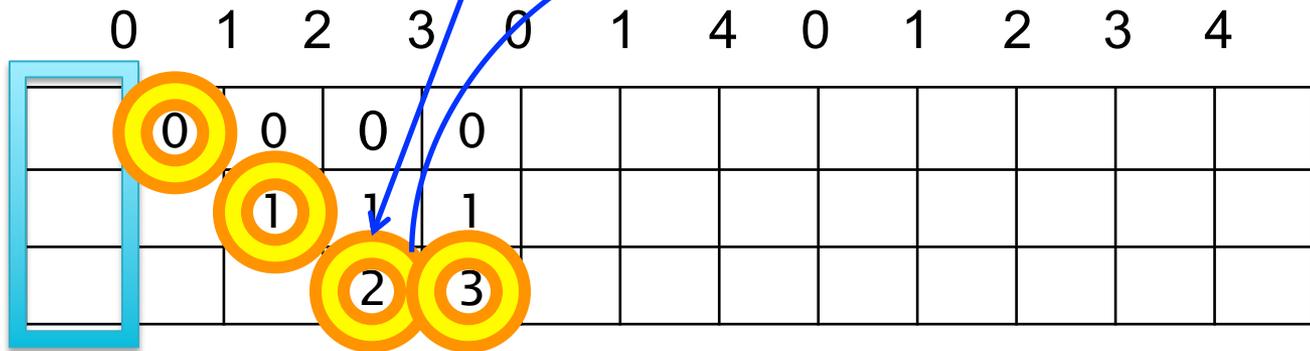
Faltas de página=10; taxa de falta=10/12=83,3%; taxa de sucesso=2/12=16,7%

Comparação

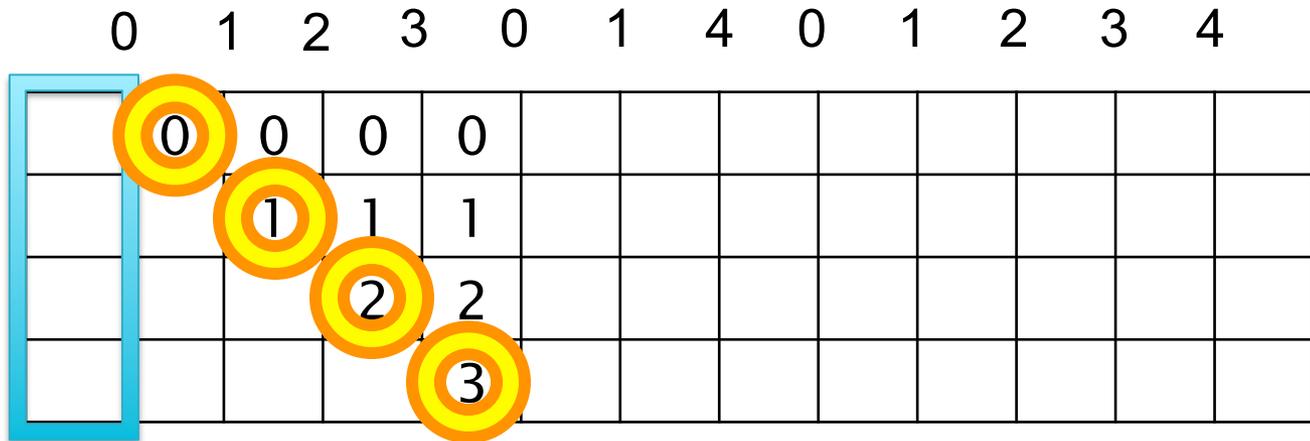
- Algoritmo Ótimo

Sabendo que 2 só será usada mais tarde....

3 molduras (frames) de páginas



4 molduras (frames) de páginas



Comparação

- Algoritmo Ótimo

Faltas de página=7; taxa de falta=7/12=58,3%; taxa de sucesso=5/12=41,7%

3 molduras
(frames) de
páginas

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

4 molduras
(frames) de
páginas

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

Faltas de página=6; taxa de falta=6/12=50%; taxa de sucesso=6/12=50%

Comparação

- LRU

Faltas de página=10; taxa de falta=10/12=83,3%; taxa de sucesso=2/12=16,7%

3 molduras
(frames) de
páginas

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

4 molduras
(frames) de
páginas

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

Faltas de página=8; taxa de falta=8/12=66,7%; taxa de sucesso=4/12=33,3%

Comparação

- FIFO
 - 3 molduras: sucesso = 25%
 - 4 molduras: sucesso = 16,7%
- Ótimo
 - 3 molduras: sucesso = 41,7%
 - 4 molduras: sucesso = 50%
- LRU
 - 3 molduras: sucesso = 16,7%
 - 4 molduras: sucesso = 33,3%

É de se esperar que quanto mais molduras, maior o sucesso, ou seja, menor a chance de "falta de página"

Controle de Carga

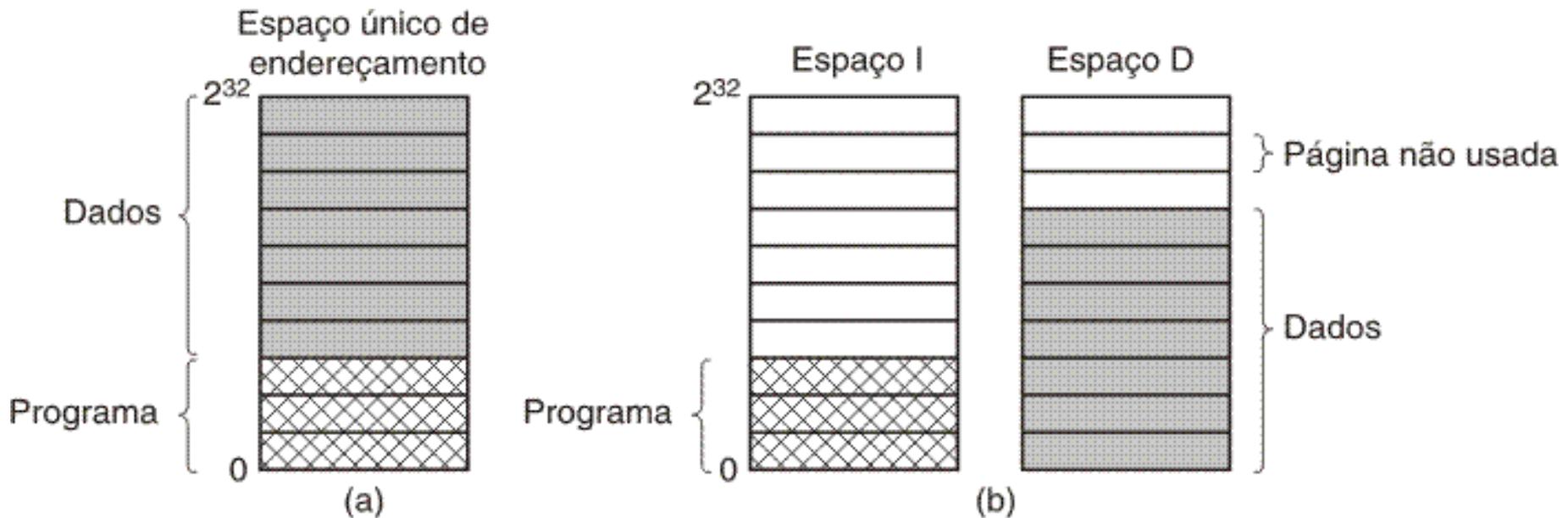
- Mesmo com um bom projeto, o sistema ainda pode sofrer paginação excessiva (*thrashing*)
- Quando
 - alguns processos precisam de mais memória
 - mas nenhum processo precisa de menos (ou seja, nenhum pode ceder páginas)
- Solução:
Reduzir o número de processos que competem pela memória
 - levar alguns deles para disco (*swap*) e liberar a memória a eles alocada
 - **reconsiderar grau de multiprogramação**

Tamanho de Página

Tamanho de página pequeno

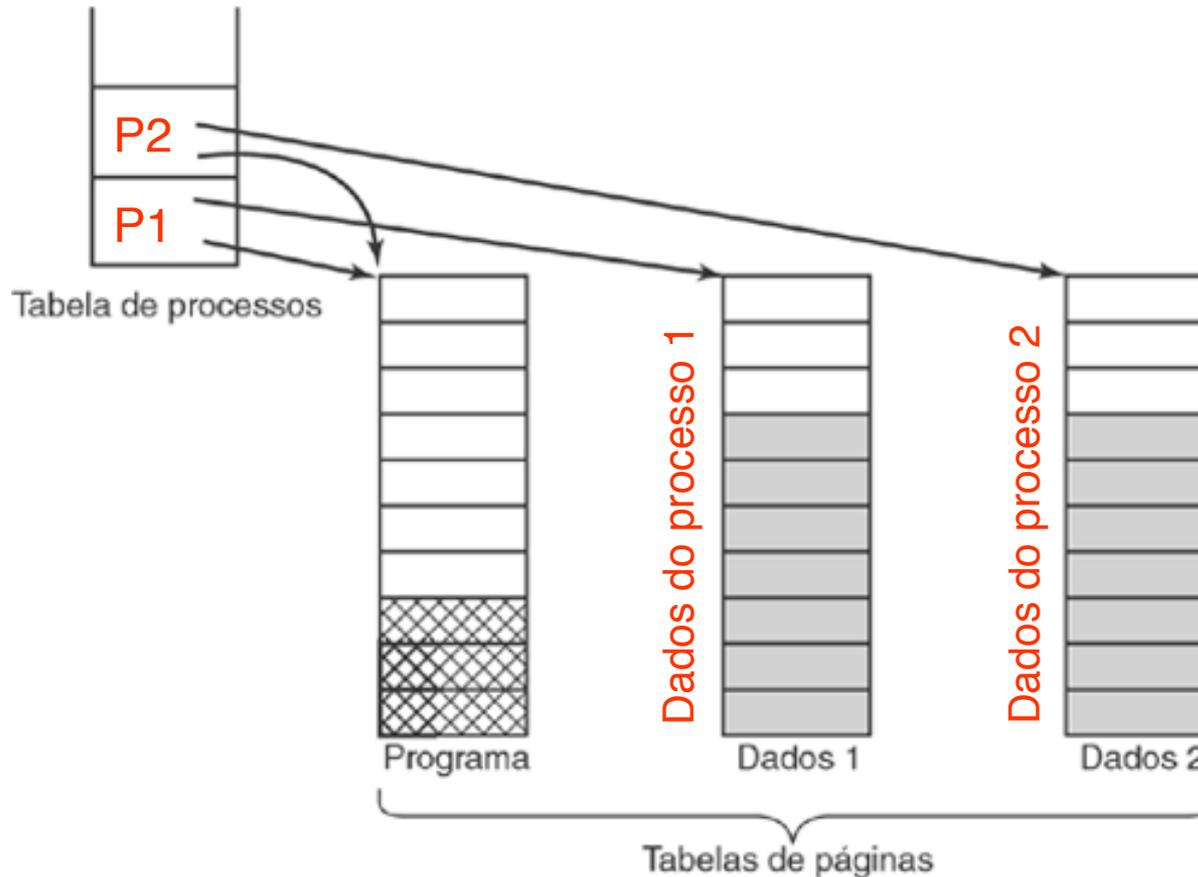
- Vantagens
 - menos fragmentação interna
 - menos programa não usado na memória
- Desvantagens
 - programas precisam de mais páginas, tabelas de página maiores

Espaços Separados de Instruções e Dados



- a) Espaço de endereçamento único
- b) Espaços separados de instruções (I) e dados (D)

Páginas Compartilhadas

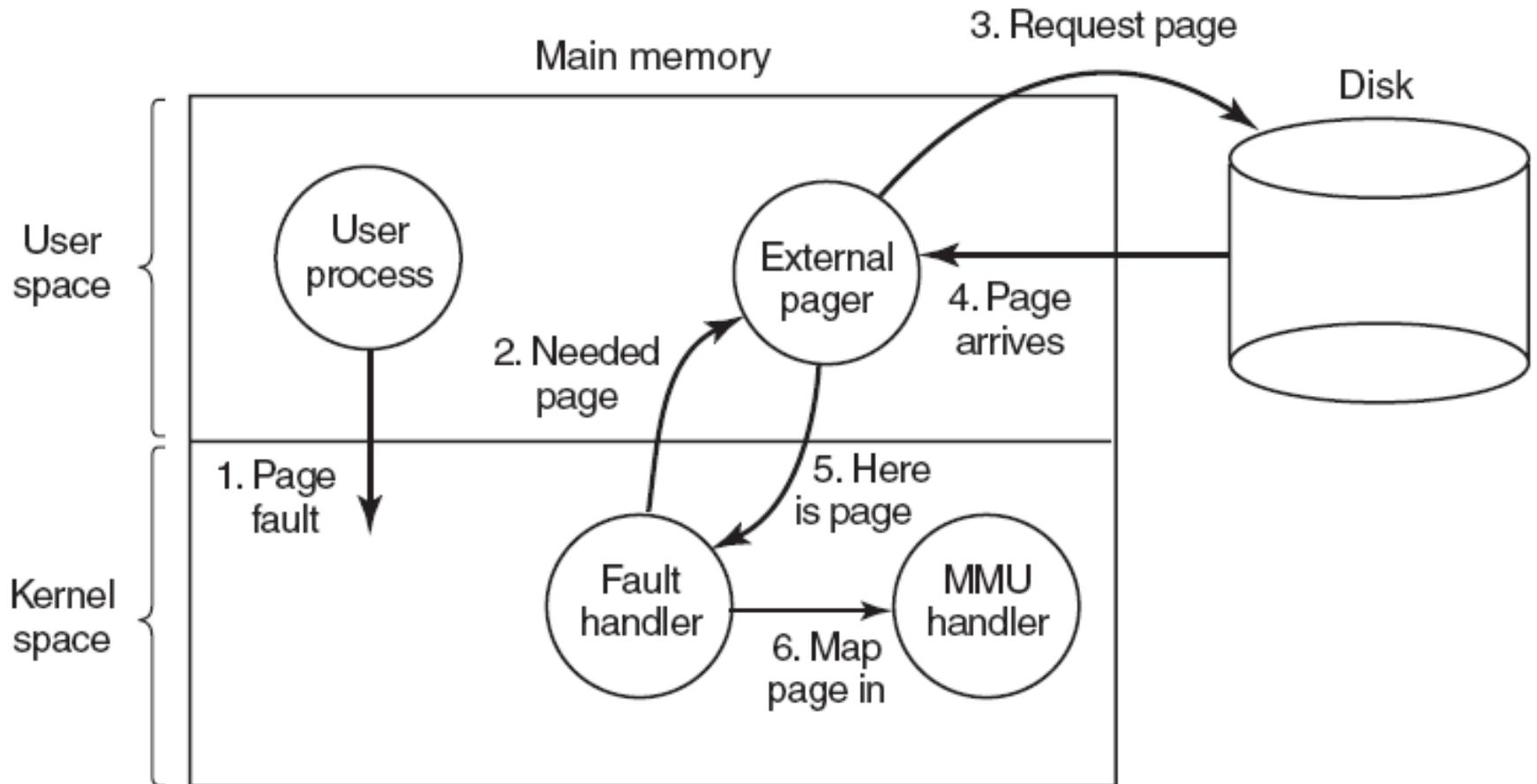


Dois processos que compartilham o mesmo código de programa e, por consequência, a mesma tabela de páginas para instruções

Envolvimento do S.O. com Paginação

1. Criação de processo
 - determina tamanho do programa
 - cria tabela de página
2. Execução de processo
 - Inicia MMU (Unidade de Gerenciamento de Memória) para novos processos
3. Ocorrência de falta de página
 - determina endereço virtual que causou a falta
 - descarta, se necessário, página antiga
 - carrega página requisitada para a memória (*swap*)
4. Terminação de processo
 - Libera tabela de páginas, páginas, e espaço em disco que as páginas ocupam

Lidando com uma falta de página: resumo



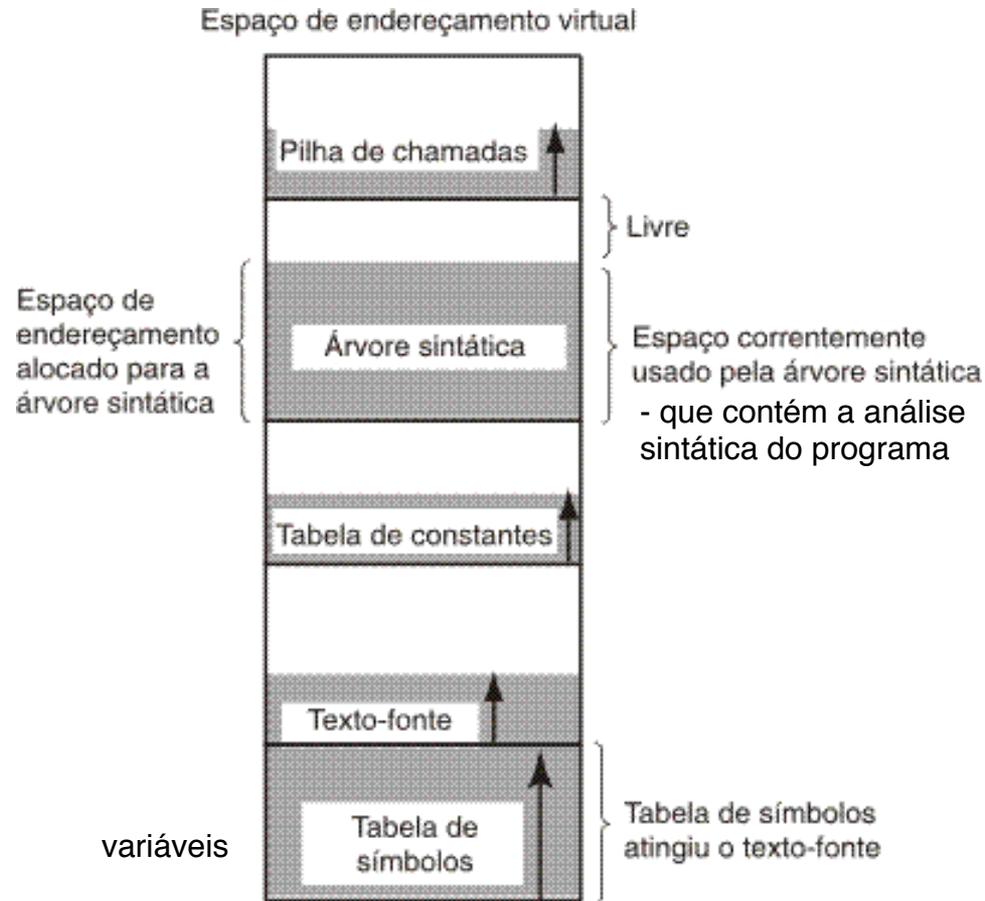
Fixação de Páginas na Memória

- Memória virtual e E/S interagem ocasionalmente
- Processo (1) emite chamada ao sistema para ler do disco para o *buffer*
 - enquanto espera pela E/S, outro processo (2) inicia
 - ocorre uma falta de página para o processo 2
 - *buffer* do processo 1 pode ser escolhido para ser levado para disco - **problema!**
- Solução possível
 - Fixação de páginas envolvidas com E/S na memória

Tópicos

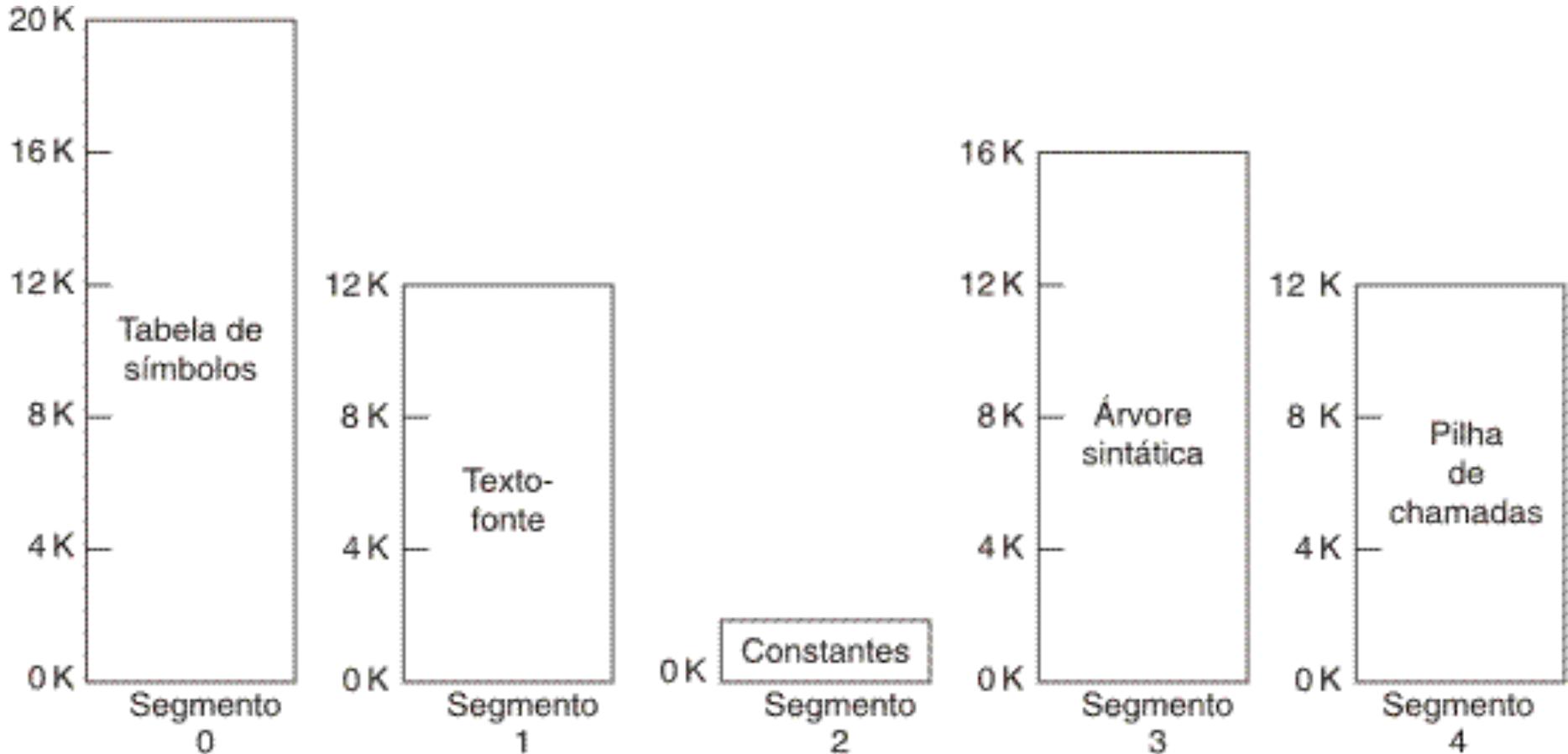
- ✓ Gerenciamento básico de memória
- ✓ Troca de processos
- ✓ Memória virtual
- ✓ Paginação
- ✓ Aceleração da paginação
- ✓ Substituição de páginas
- Segmentação

Segmentação (1)



- Espaço de endereçamento unidimensional com tabelas crescentes
- Uma tabela pode atingir outra...

Segmentação (2)

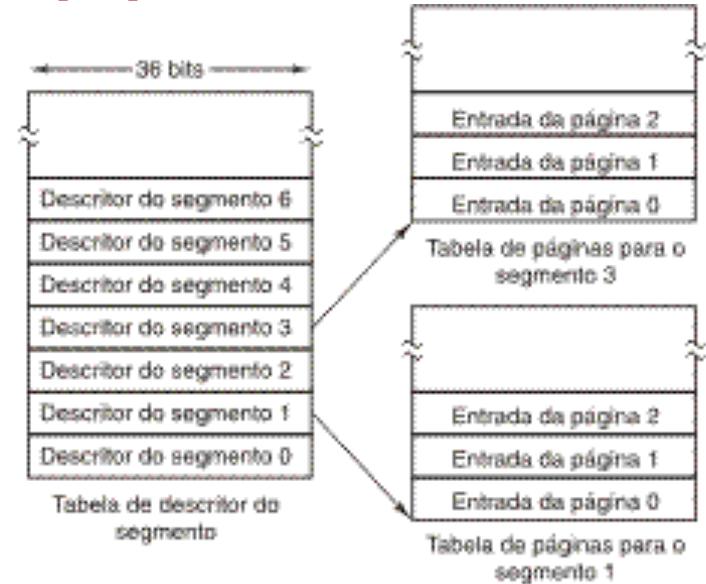


Permite que cada tabela cresça ou encolha, independentemente

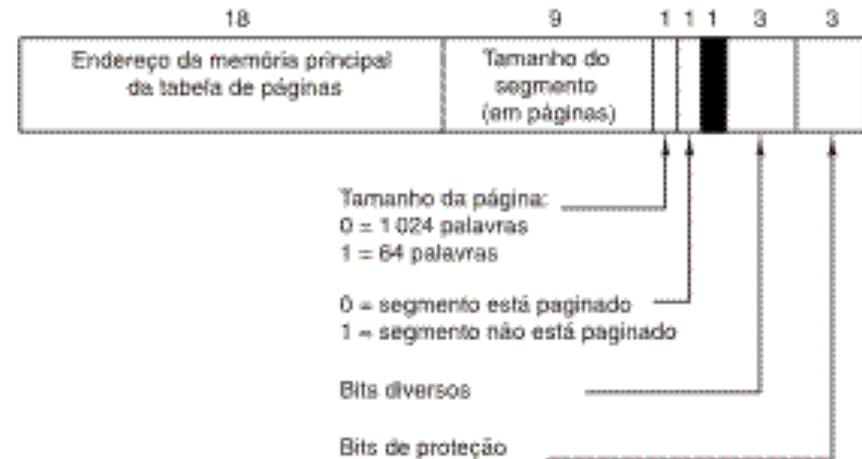
Comparação entre paginação e segmentação

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

Segmentação com Paginação: MULTICS (1)



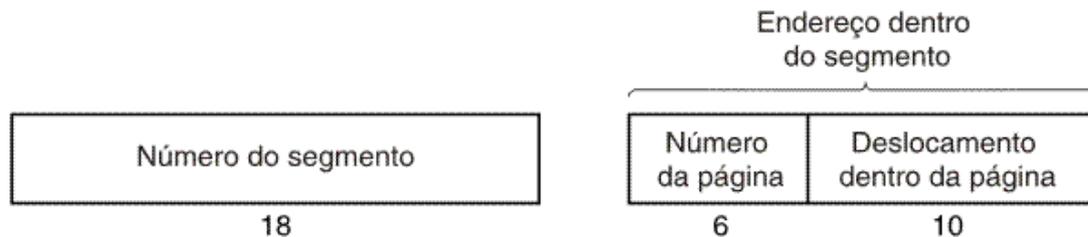
(a)



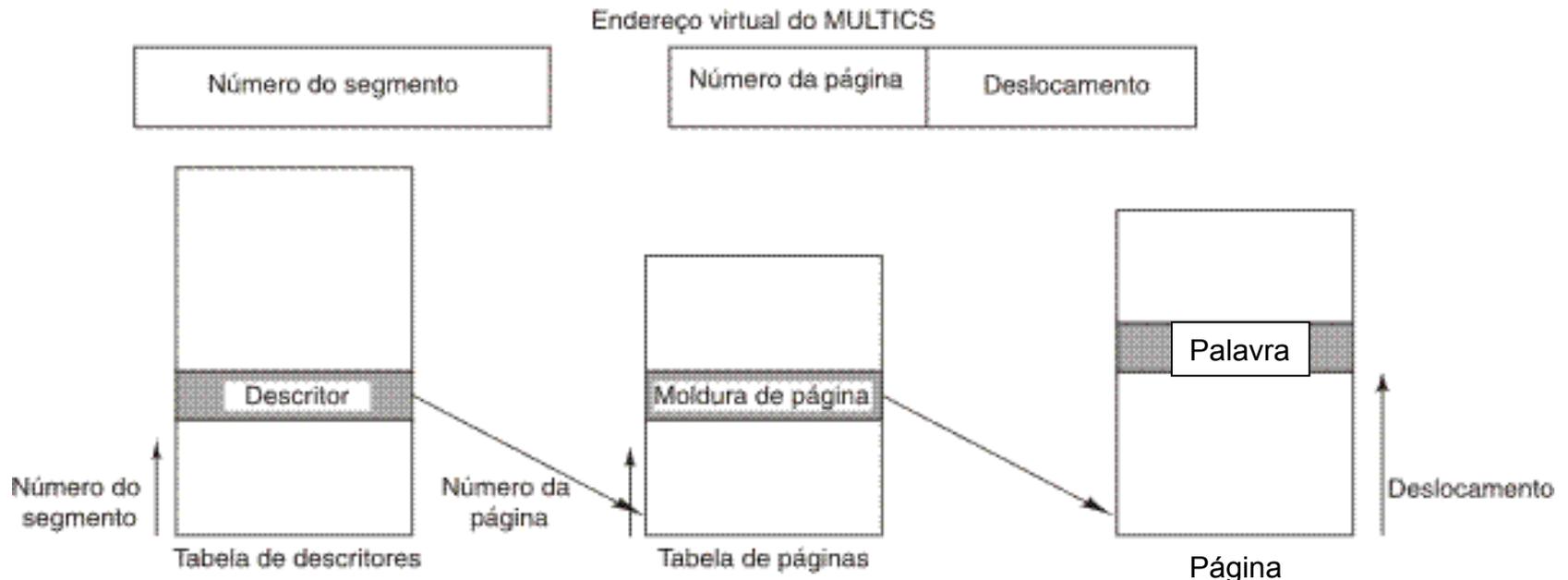
(b)

- Descritores de segmentos apontam para tabelas de páginas
- Descritor de segmento

Segmentação com Paginação: MULTICS (2)



Um endereço virtual de 34 bits no MULTICS



Conversão de um endereço MULTICS de duas partes em um endereço da memória principal

Conceitos Importantes

:: comentários ::

- Hierarquia de memórias
- Relocação e Proteção
- *Swapping* (troca de processos na memória)
- Mapa de bits
- Página
- Memória virtual
- Paginação (*paging*), falta de página (*page fault*)
- Moldura de página
- Grau de multiprogramação
- Segmentação

Conclusões

- Na forma mais simples de memória virtual, cada espaço de endereçamento de um processo é dividido em páginas de tamanho uniforme, que podem ser colocadas em qualquer moldura de página disponível na memória
- Dois dos melhores algoritmos de substituição de páginas são o Envelhecimento (aging) e o WSClock

Conclusões

- No projeto de sistemas de paginação, a escolha de um algoritmo não é suficiente
Outras considerações:
 - Política de alocação
 - Determina quantas molduras de páginas cada processo pode manter na memória principal
 - Tamanho de página
- Segmentação ajuda a lidar com estruturas de dados que mudam de tamanho durante a execução e
 - simplifica o compartilhamento
 - permite proteção diferente para segmentos diferentes
- Segmentação e paginação podem ser combinados para fornecer uma memória virtual de duas dimensões