# UFFS
# Ciência da Computação

## Sistemas Operacionais

*Prof. Marco Aurélio Spohn*

# Gerenciamento de Memória

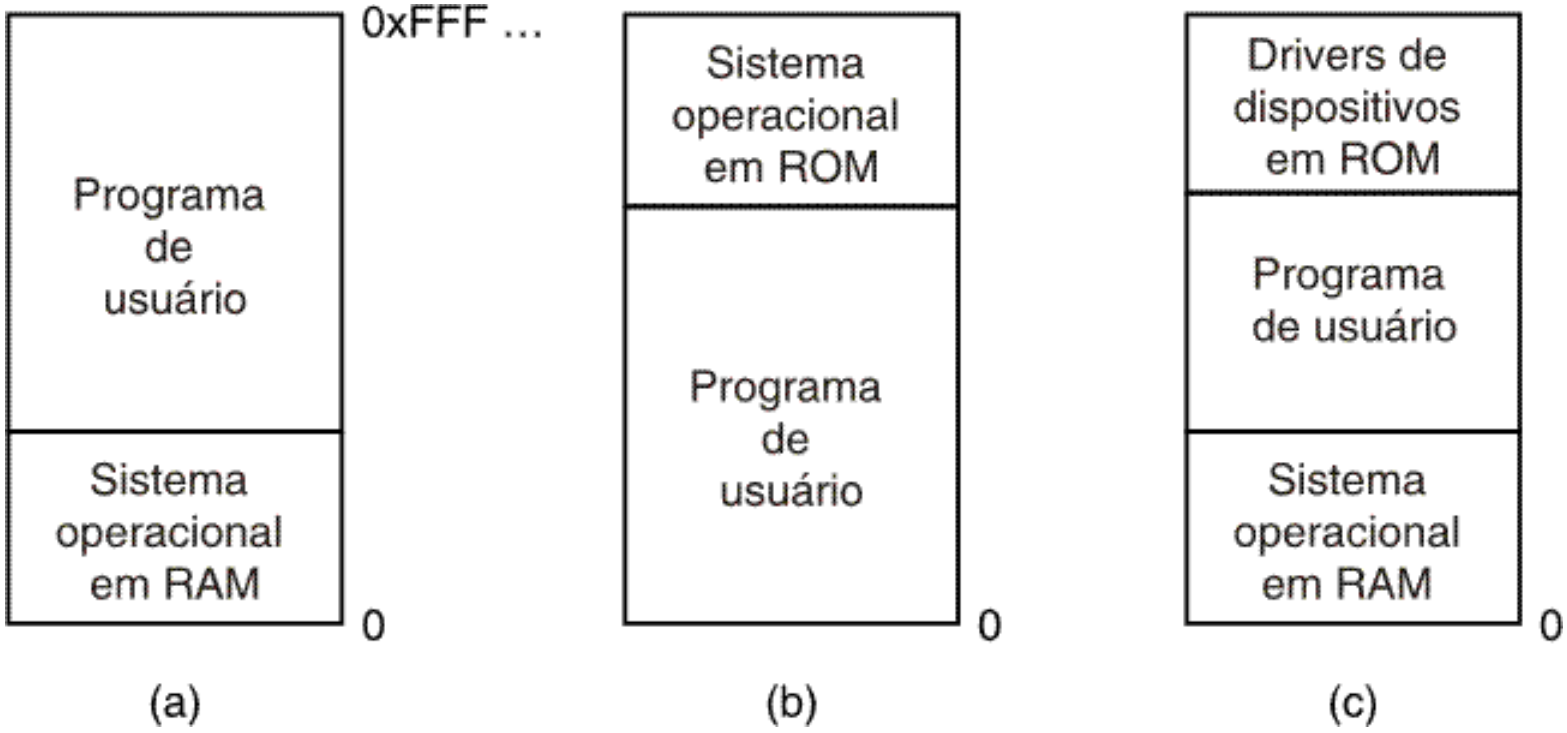
4.1 Gerenciamento básico de memória
4.2 Troca de processos
4.3 Memória virtual
4.4 Algoritmos de substituição de páginas
4.5 Modelagem de algoritmos de substituição de páginas
4.6 Questões de projeto para sistemas de paginação
4.7 Questões de implementação
4.8 Segmentação

# 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

# Gerenciamento Básico de Memória

## Monoprogramação sem Troca de Processos ou Paginação

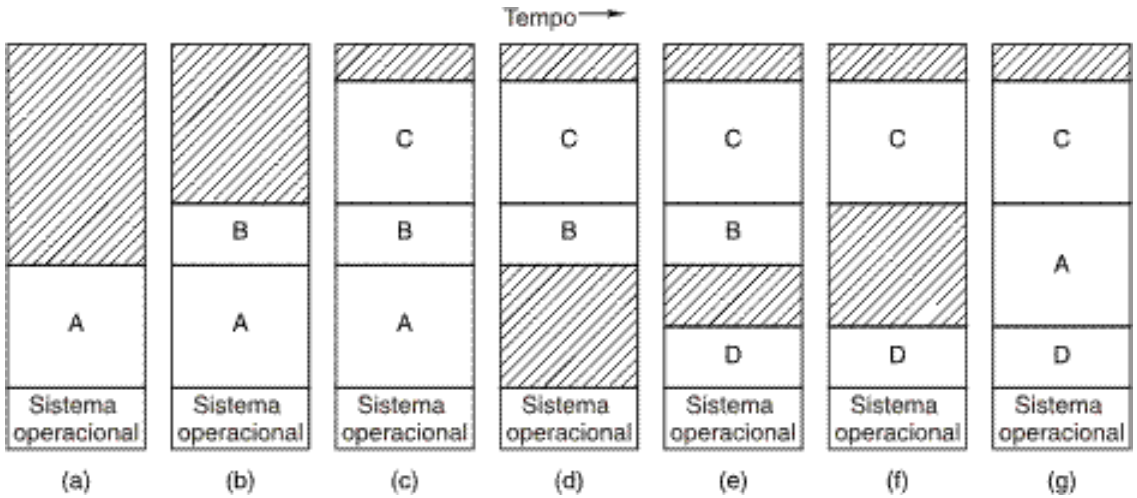



- Três maneiras simples de organizar a memória
- - um sistema operacional e um processo de usuário

# Relocação e Proteção

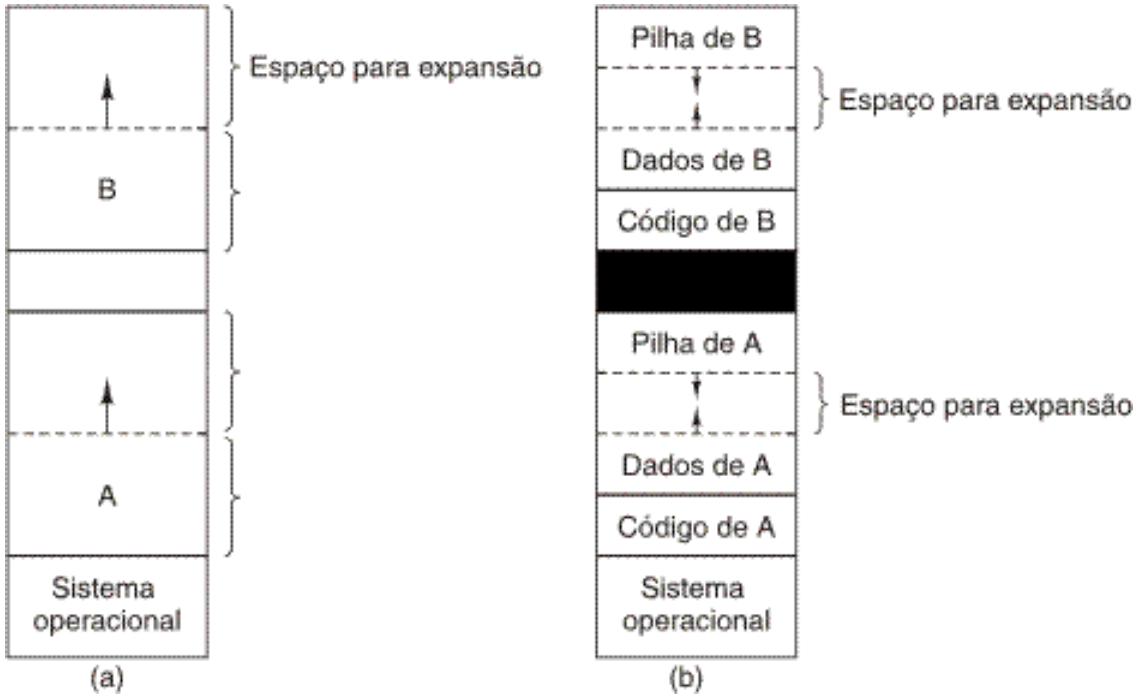
- Não se sabe com certeza onde o programa será carregado na memória
  - Localizações de endereços de variáveis e de código de rotinas não podem ser absolutos
- Uma possível solução: instruções do programa são modificadas segundo a partição de memória em que ele será carregado
- Uma solução para relocação e proteção: uso de valores base e limite
  - localizações de endereços são somadas ao valor base antes de serem mapeadas na memória física
  - localizações de endereços maior que o valor limite indicam erro

# 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

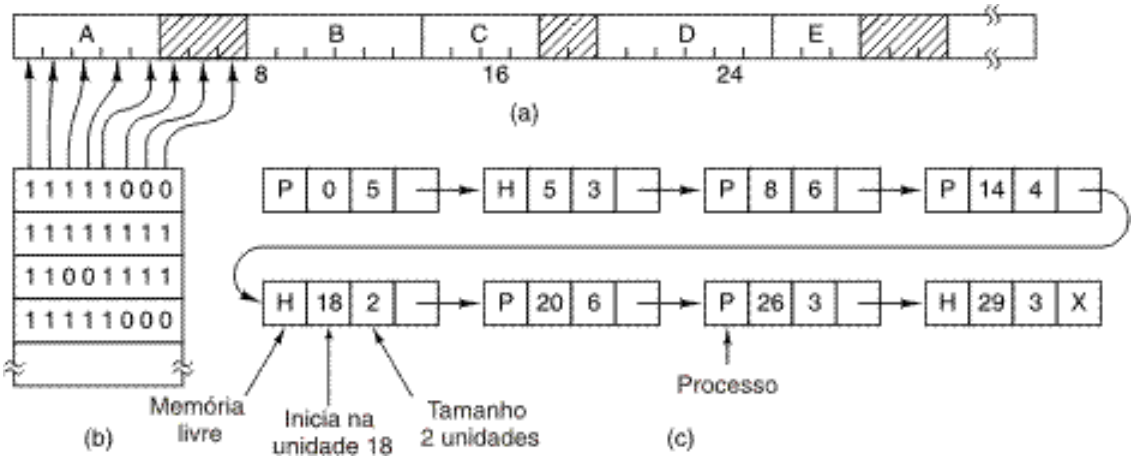
# Troca de Processos (2)




a) Alocação de espaço para uma área de dados em expansão

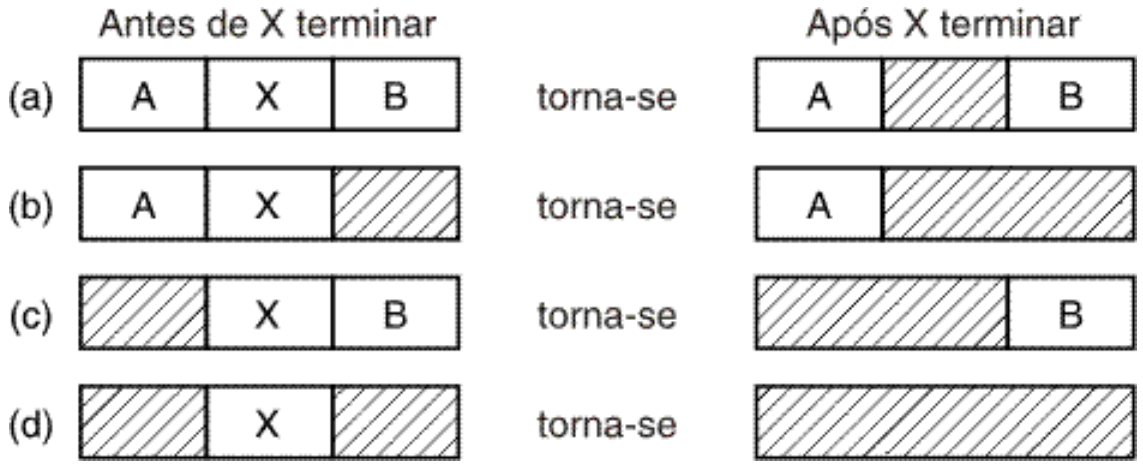
b) Alocação de espaço para uma pilha e uma área de dados, ambos em expansão

# Gerenciamento de Memória com Mapas de Bits




- (a) Parte da memória com 5 segmentos de processos e 3 segmentos de memória livre
- pequenos riscos simétricos denotam as unidades de alocação
- regiões sombreadas denotam segmentos livres
- (b) Mapa de bits correspondente
- (c) Mesmas informações em uma lista encadeada

# Gerenciamento de Memória com Listas Encadeadas

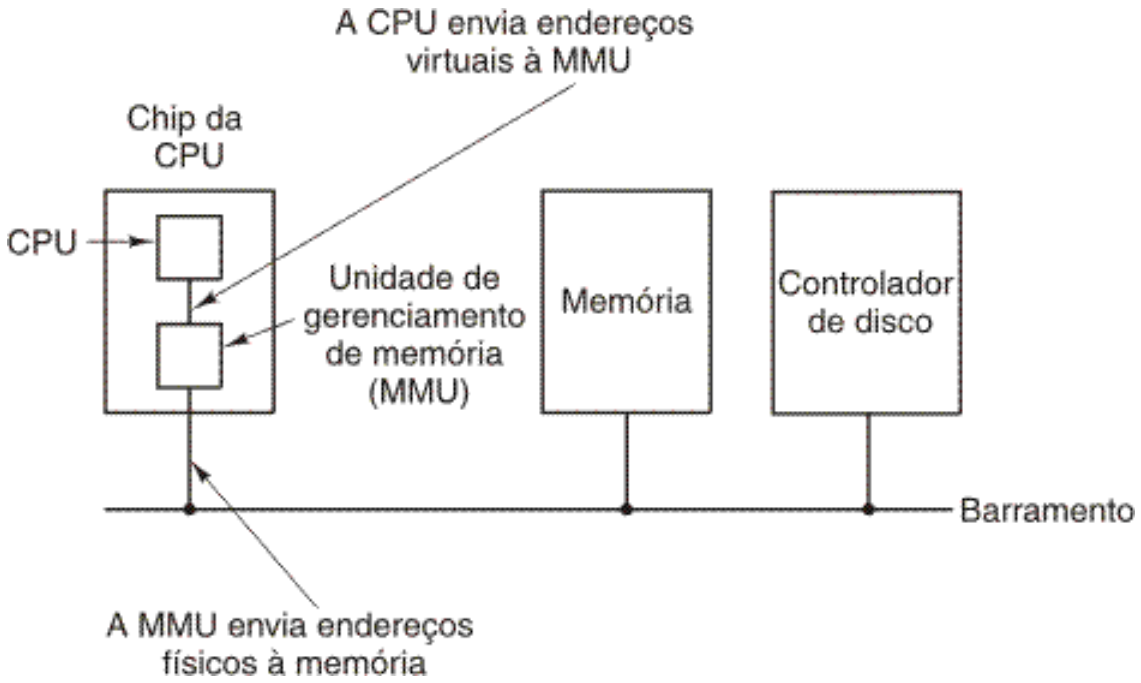



Quatro combinações de vizinhança para o processo X em término de execução

# Memória Virtual
# Paginação (1)

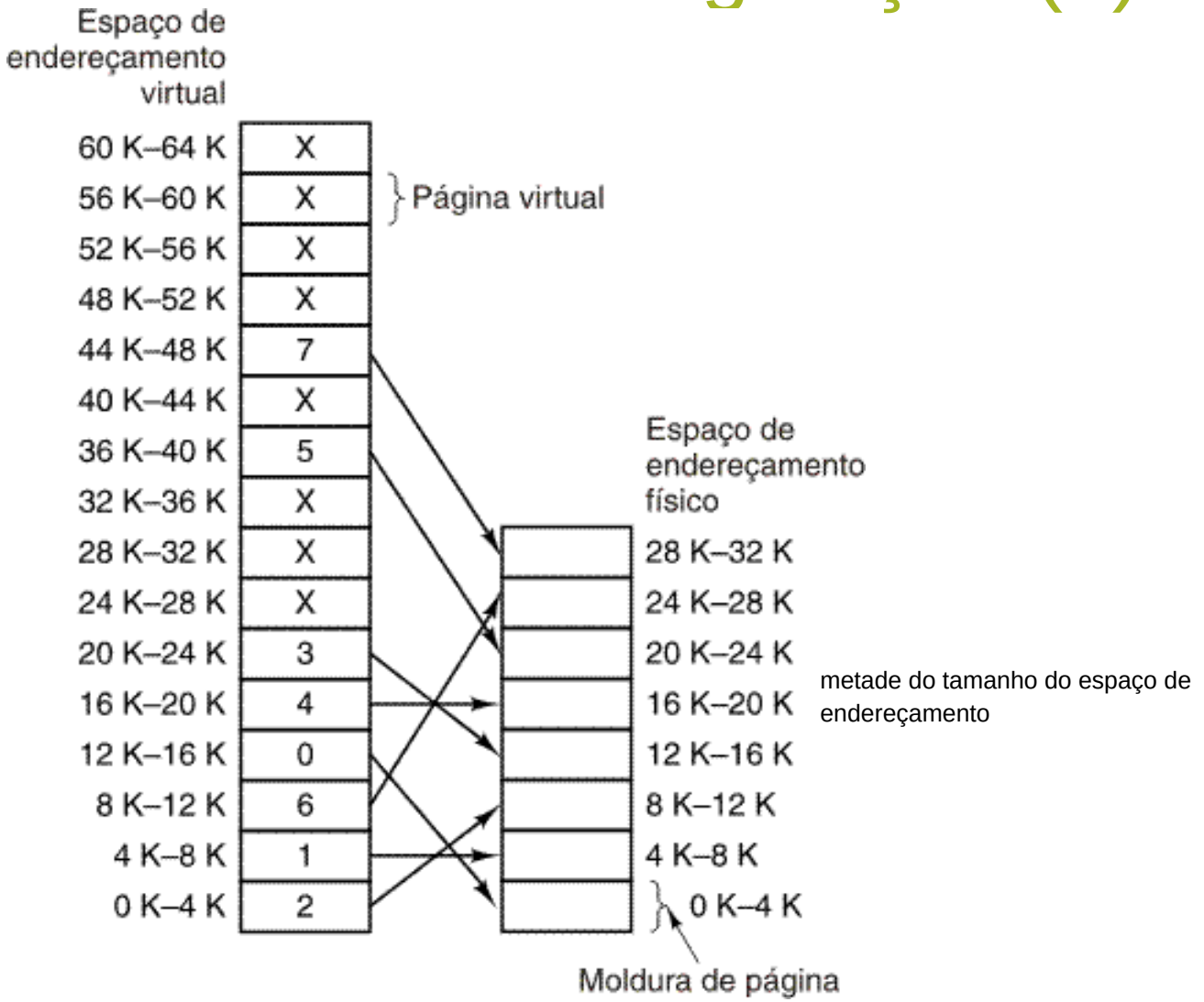
- **Memória física** disponível, geralmente, é muito **inferior ao espaço de endereçamento**. Para aproveitar o máximo possível da capacidade de endereçamento, utiliza-se memória virtual, permitindo aos processos utilizar toda a faixa de endereços disponível.
- E, principalmente, os **processos lidam** apenas **com endereços lógicos**, eliminando a necessidade de se preocupar com a alocação física na memória.

# Memória Virtual
# Paginação (1)



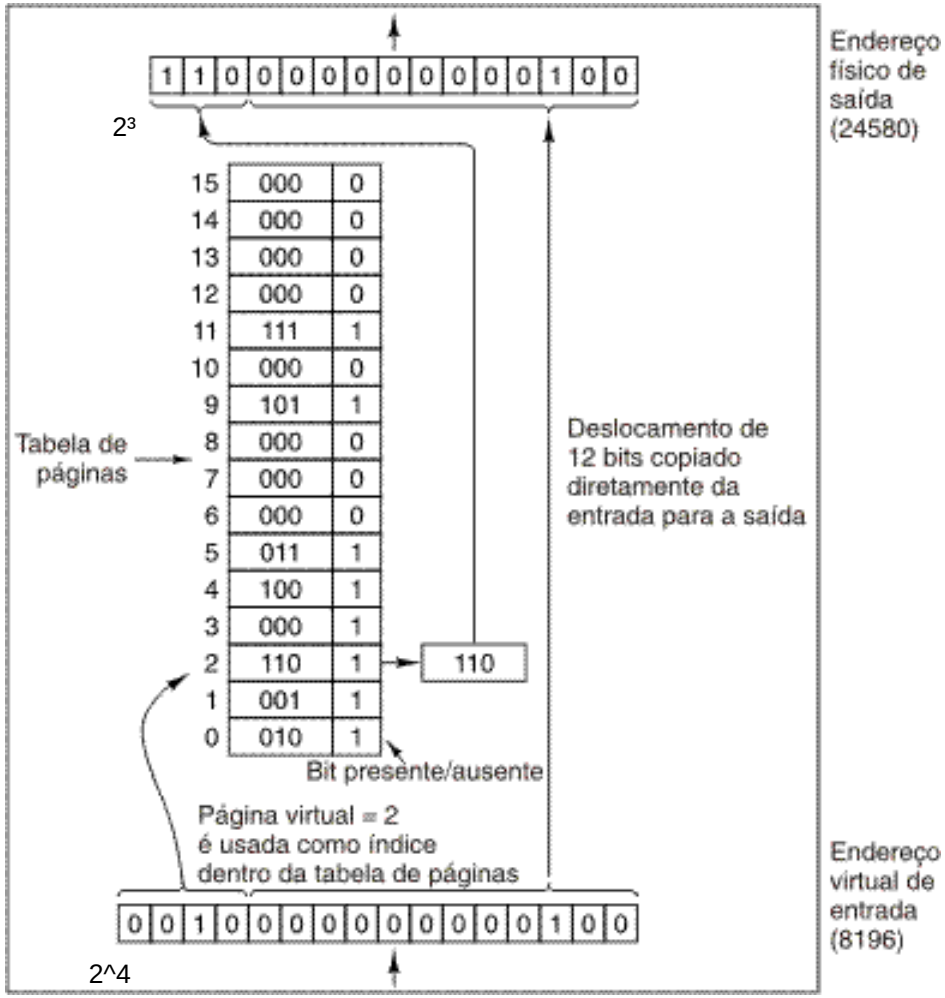

- Localização e função da MMU

# Memória Virtual Paginação (2)



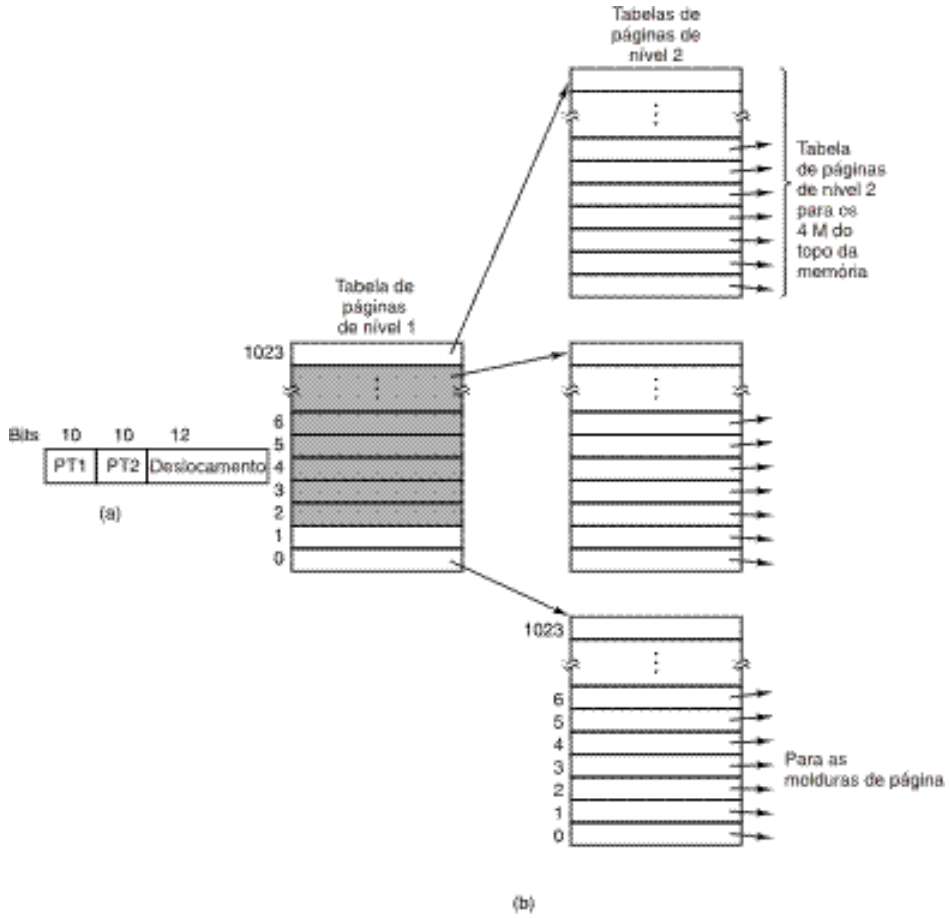

**A relação entre endereços virtuais e endereços físicos de memória dada pela tabela de páginas**

# Tabelas de Páginas (1)




Operação interna de uma MMU com 16 páginas de 4KB

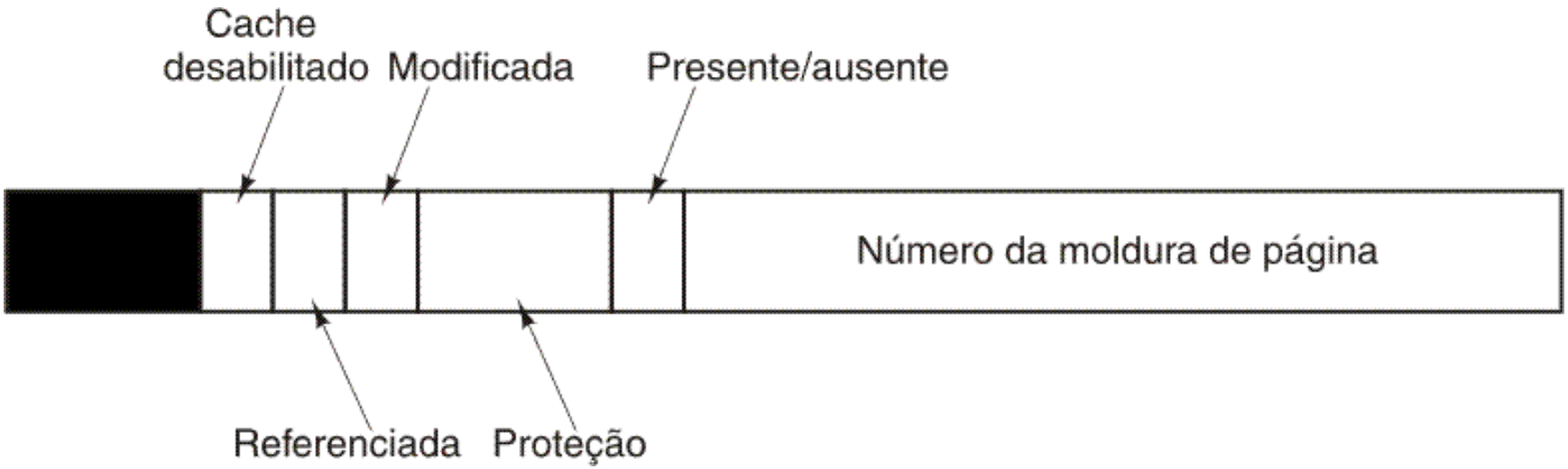
# Tabelas de Páginas(2)




a) Endereço de 32 bits com 2 campos para endereçamento de tabelas de páginas

b) Tabelas de páginas com 2 níveis

# Tabelas de Páginas(3)




- Entrada típica de uma tabela de páginas

# Memória Associativa ou *Translation Lookaside Buffer* (TLB): consultada antes de consultar tabela de páginas

| Válida | Página virtual | Bit modificada | Bits de proteção | Moldura de página |
|---|---|---|---|---|
| 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 |

- TLB para acelerar a paginação: contém as páginas acessadas mais recentemente pelo processo

# Memória Associativa ou TLB

- Gerenciamento de TLB por software:
- 1) Reduz a complexidade da cpu, permitindo expandir outros componentes desta (e.g., mais cache)
- 2) O S.O. pode explorar melhor a TLB porque conhece mais detalhes sobre os processos (e.g., aplicações cliente servidor, podendo antecipar-se e trazer para memória páginas e mapeá-las antes que aconteça TLB faults).

# Tabelas de Páginas Invertidas (para arquiteturas **64 bits**): entradas (n, p) = página p do processo n; caso utilização de uma tab. de pgs convencional, precisaria mais de 32 milhões de GB!!!

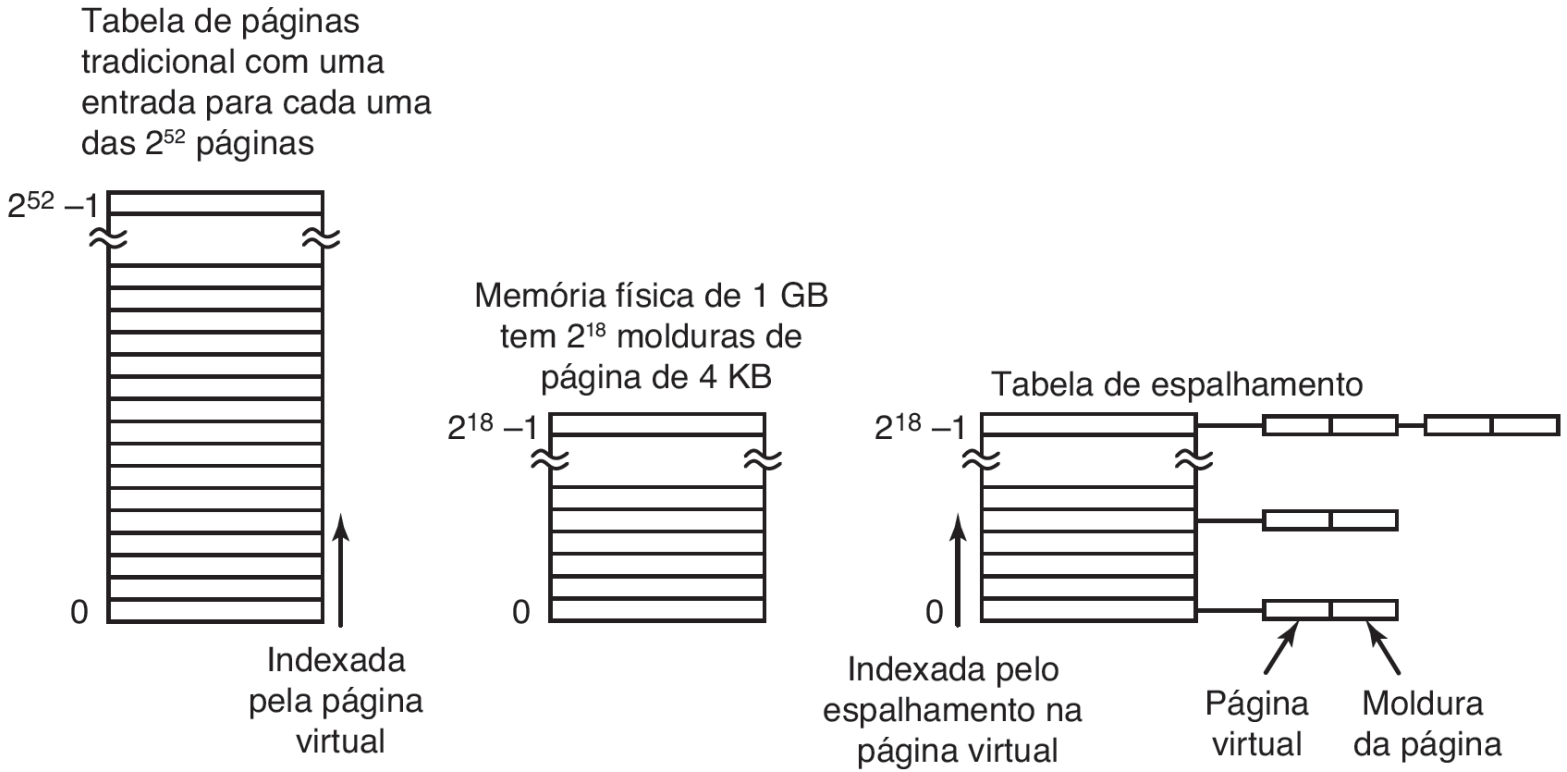
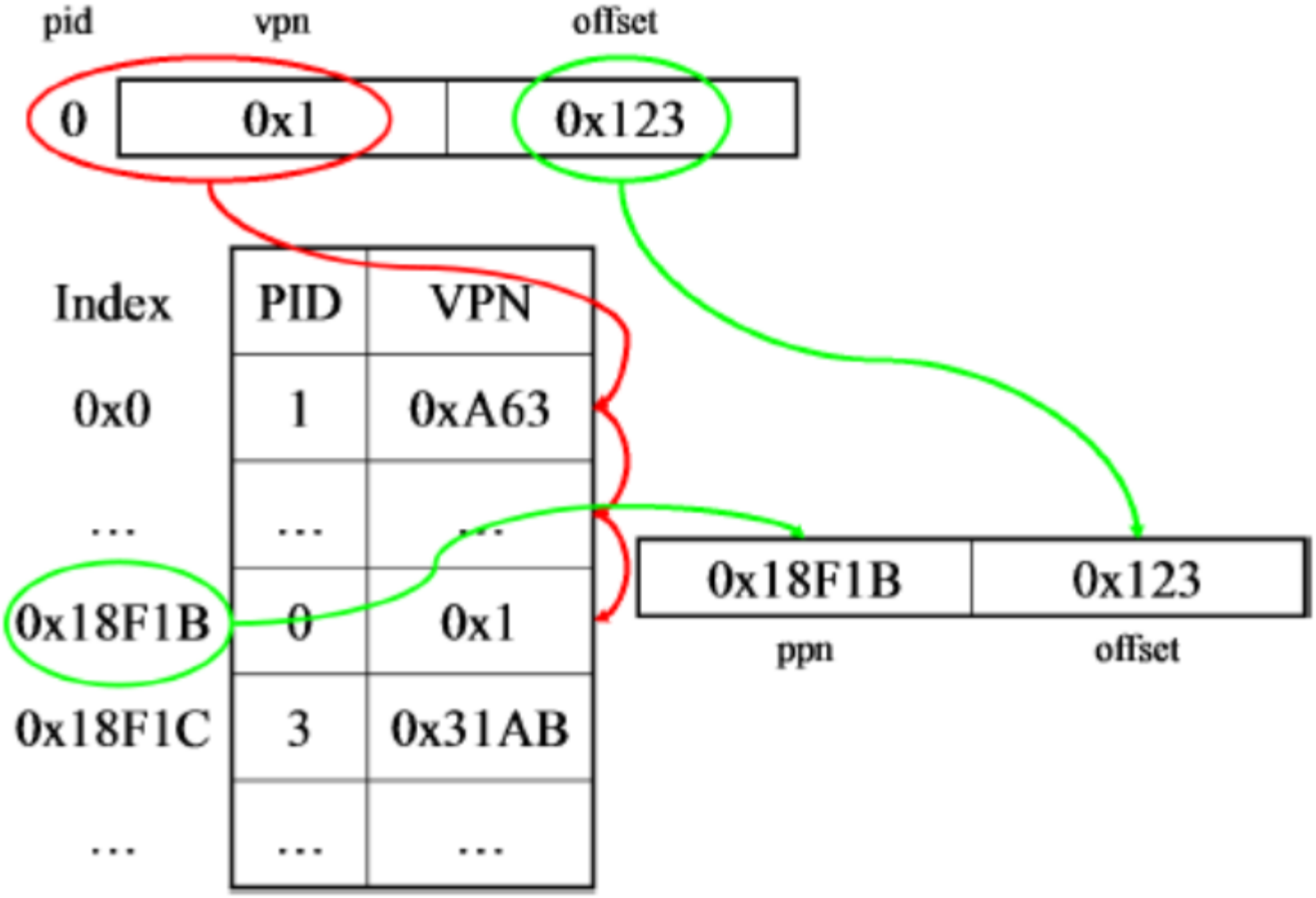


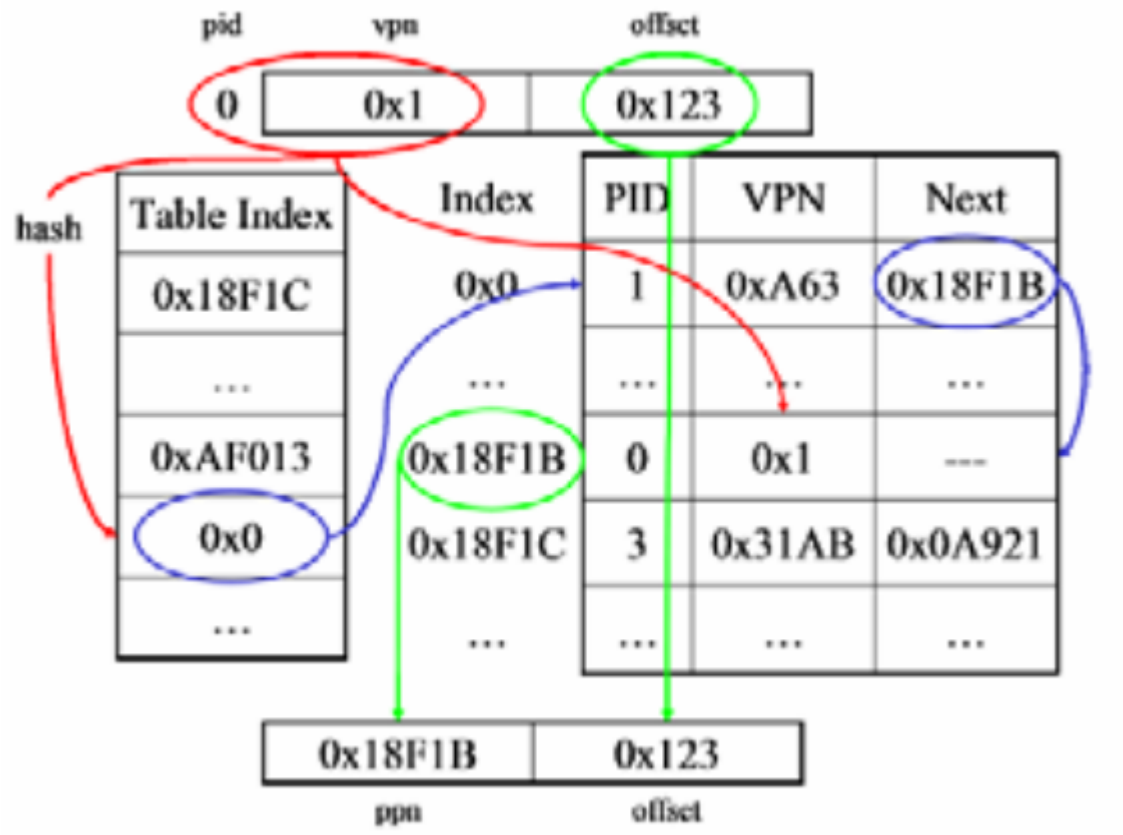

**Figura 3.13** Comparação entre uma tabela de páginas tradicional e uma tabela de páginas invertidas.

- Comparação de uma tabela de páginas tradicional com uma tabela de páginas invertidas. Ou seja, utiliza-se uma tabela de uma entrada por página física da máquina. Todas as páginas virtuais atualmente na memória são mapeadas para uma entrada na tabela invertida utilizando uma função *hash*. Caso a tabela tenha uma entrada para cada página física, em média haverá apenas uma entrada por página virtual.

# Páginas invertidas (custo linear)



# Páginas invertidas: *hash* com espalhamento aberto



# Algoritmos de Substituição de Páginas

- A falta de página força uma escolha
  - qual página deve ser removida (se necessário!)
  - alocação de espaço para a página a ser trazida para a memória
- A página modificada deve primeiro ser salva
  - se não tiver sido modificada, o quadro (*frame*) é apenas sobreposto
- Melhor não escolher uma página que está sendo muito usada
  - provavelmente precisará ser trazida de volta logo

# O Algoritmo de Substituição de Página *Ótimo*

- Substitui a página que será necessária mais adiante possível
  - ótimo mas não realizável!!
- **Estimada** através de...
  - registro do uso da página em execuções anteriores de outra instância do mesmo processo ...
  - No entanto, isso também é impraticável

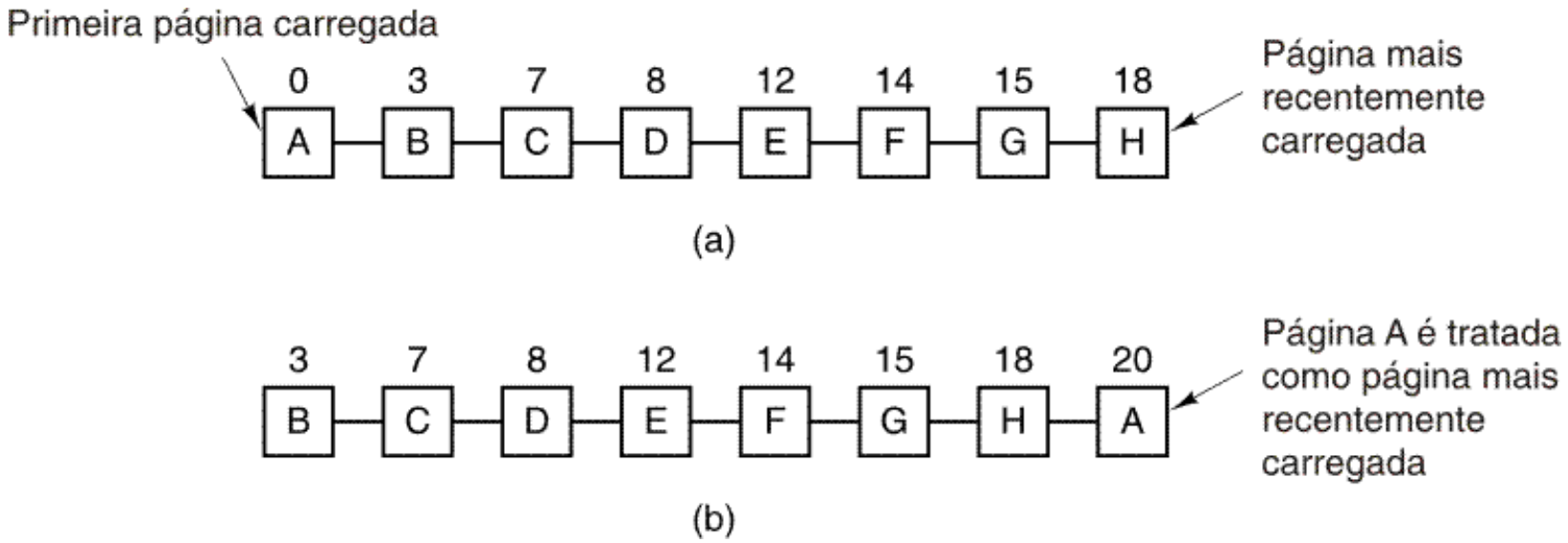
# O Algoritmo de Substituição de Página *Não Usada Recentemente (NUR)*

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

# Algoritmo de Substituição de Página *Primeira a Entrar, Primeira a Sair*

- Mantém uma lista encadeada de todas as páginas
  - página mais antiga na cabeça da lista
  - página que chegou por último na memória no final da lista
- Na ocorrência de falta de página
  - página na cabeça da lista é removida
  - nova página adicionada no final da lista
- Desvantagem
  - página há mais tempo na memória pode ser justamente aquela utilizada com muita freqüência

# Algoritmo de Substituição de Página *Segunda Chance (SC)*

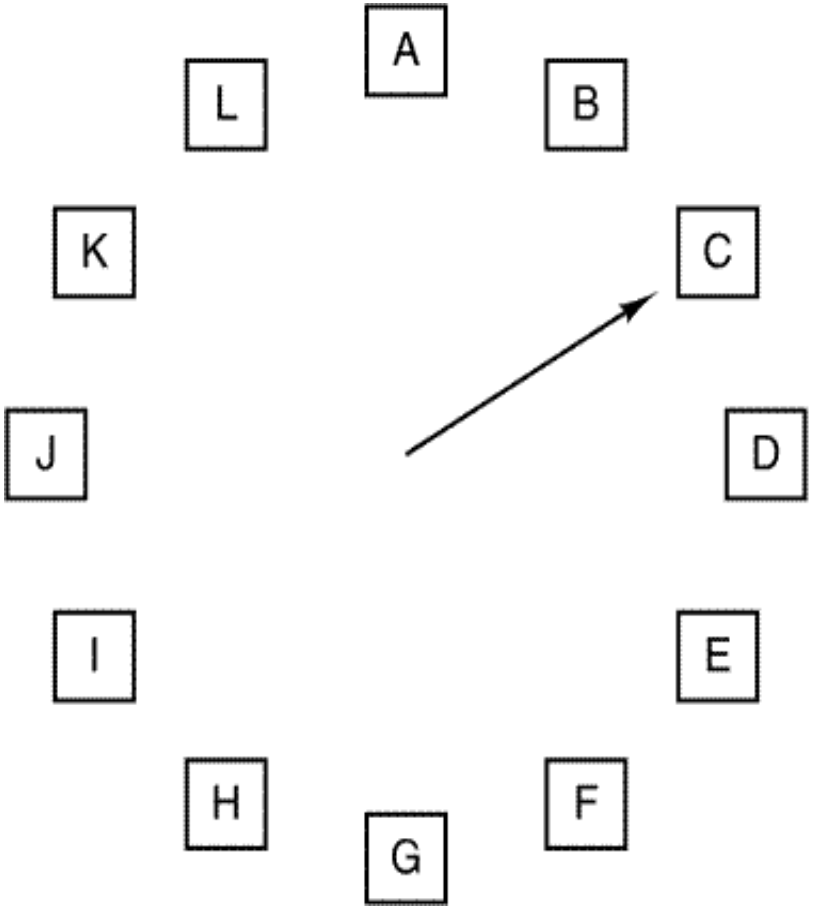



- Operação do algoritmo *segunda chance*

a) lista de páginas em ordem FIFO

b) estado da lista em situação de falta de página no instante 20, com o bit *R* da página *A* em 1, recebe segunda chance colocando-a no final da fila e R = 0 (números representam instante's de carregamento/referência das páginas na memória)

# Algoritmo de Substituição de Página *Relógio*

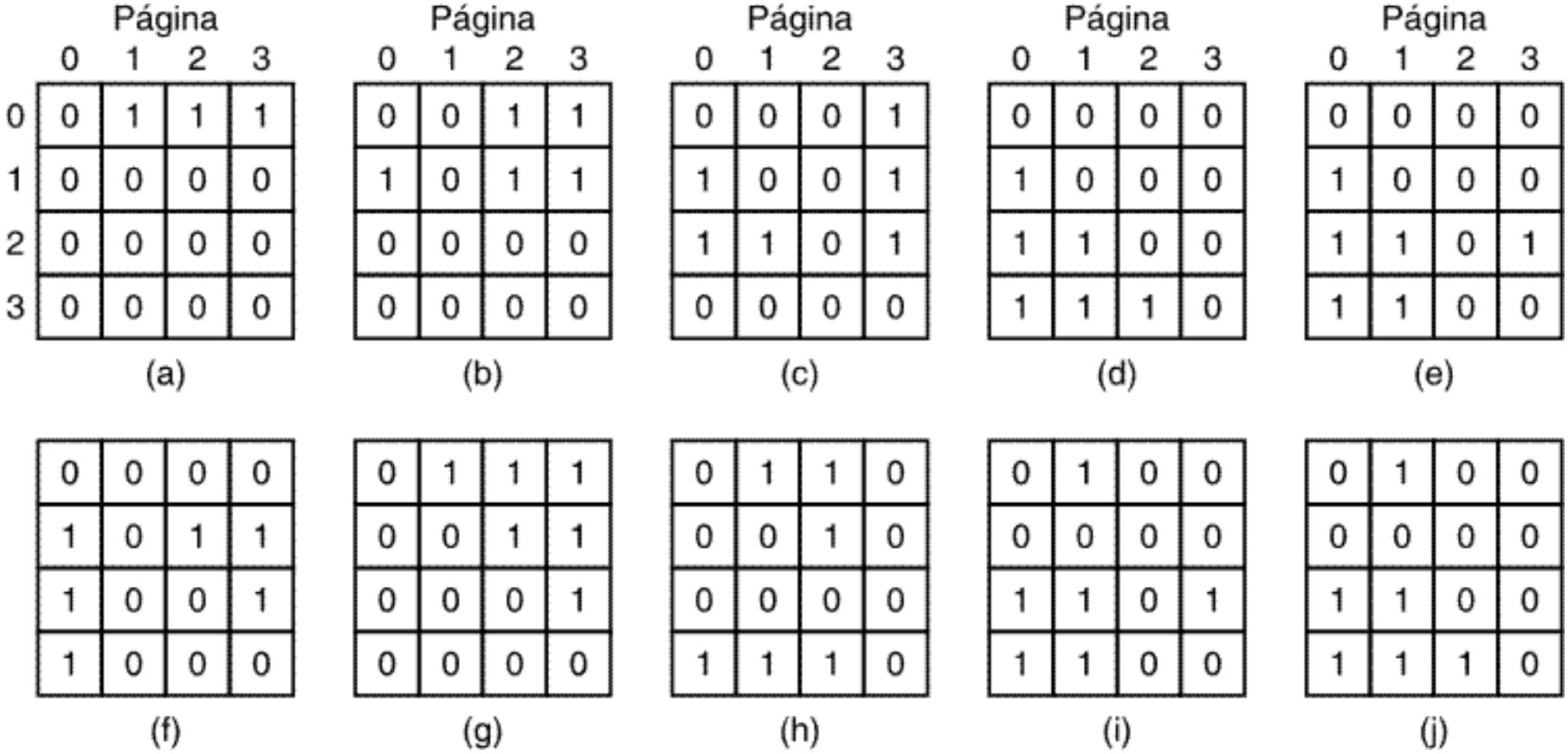



Quando ocorre uma falta de página,
a página apontada é examinada.
A atitude a ser tomada depende do bit R:
R = 0: Retira a página,
R = 1: Faz R = 0 e avança o ponteiro.

# Menos Recentemente Usada (MRU)

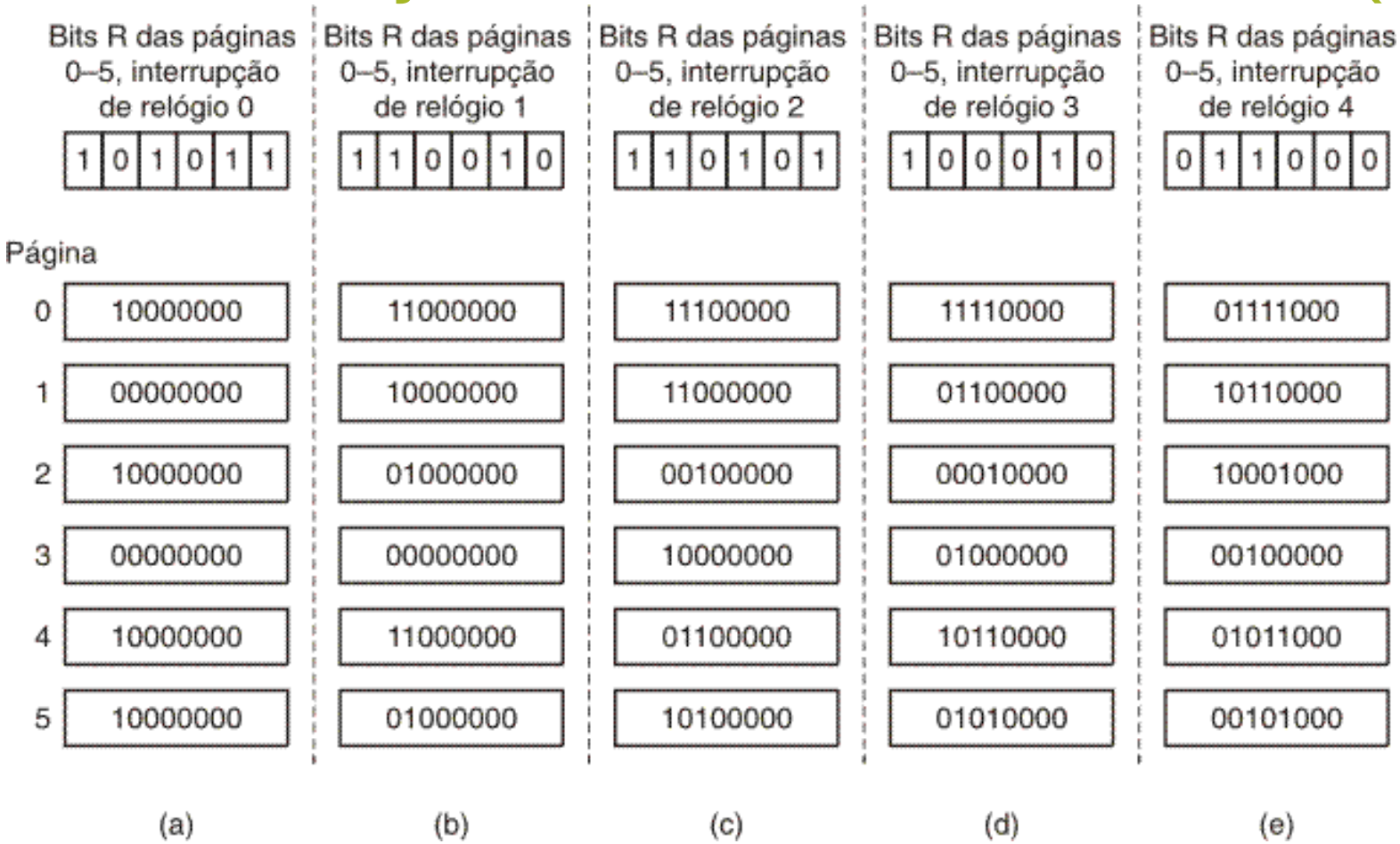
- Assume que páginas usadas recentemente logo serão usadas novamente
  - retira da memória página que há mais tempo não é usada
- Uma lista encadeada de páginas deve ser mantida
  - página mais recentemente usada no início da lista, menos usada no final da lista
  - **atualização da lista a cada referência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

# MRU usando uma matriz: n frames, matriz n x n



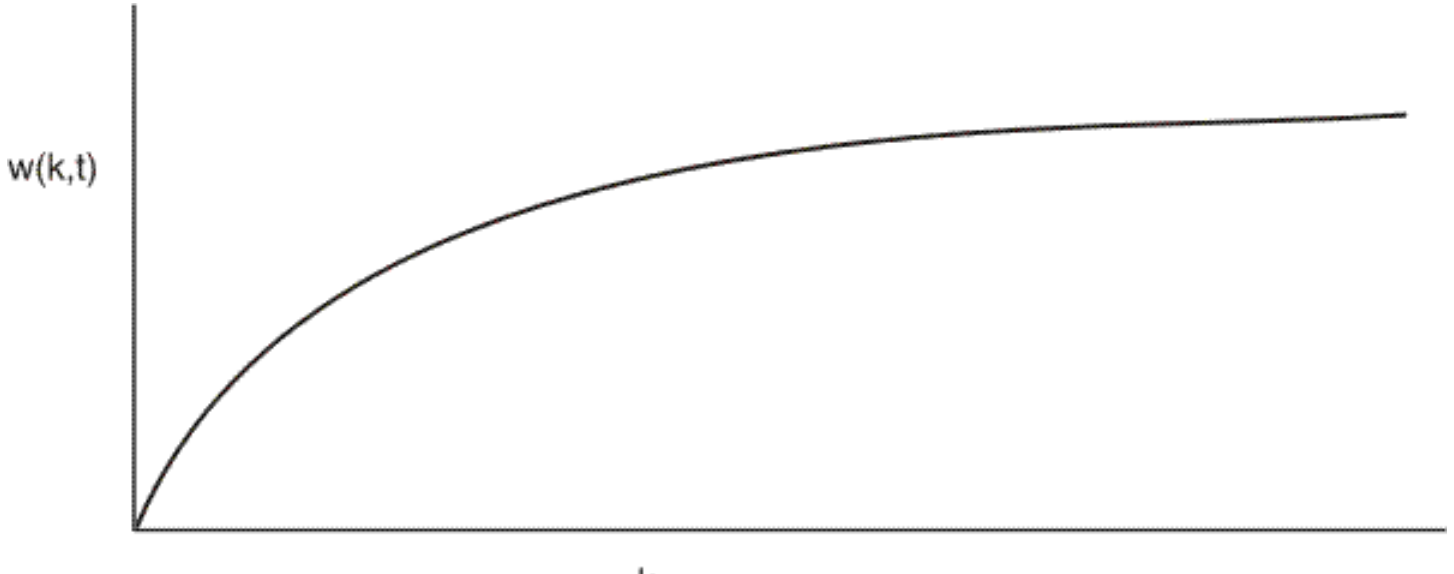

- MRU usando uma matriz – **páginas referenciadas na ordem 0,1,2,3,2,1,0,3,2,3**

# Simulação do MRU em Software (2)



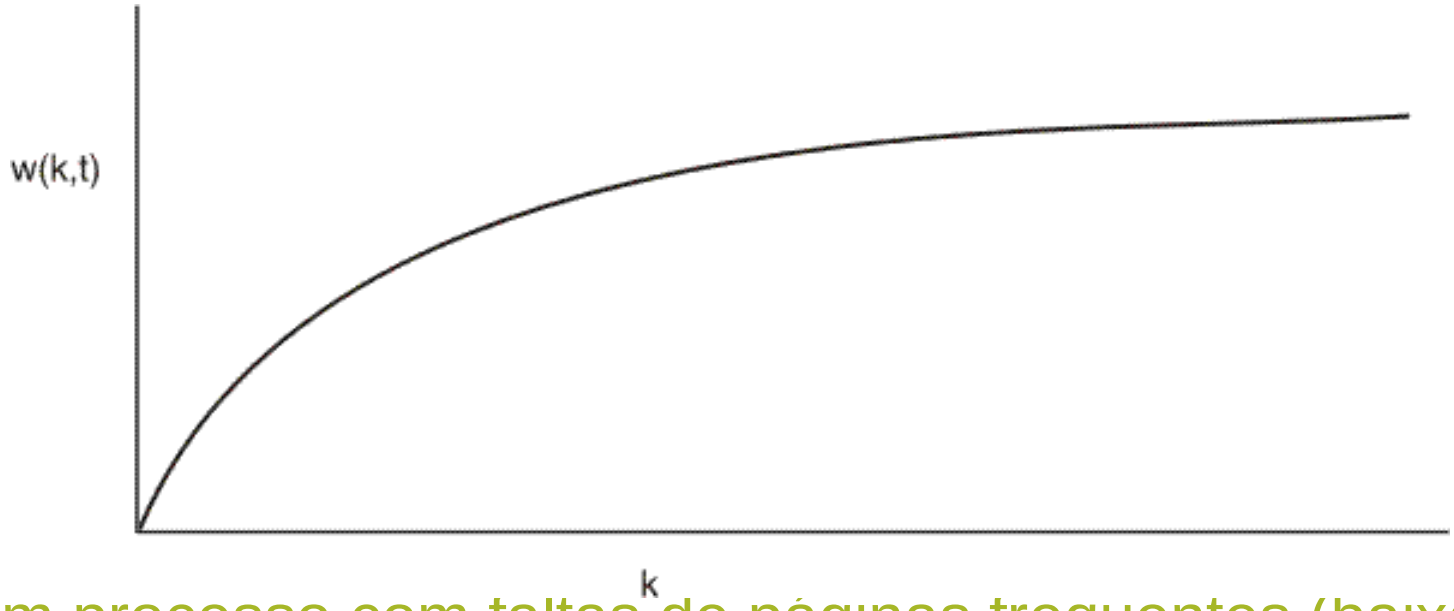

- O algoritmo do envelhecimento *(aging)* simula o MRU em software (atualiza 1 vez por intervalo interrupção)
- Note 6 páginas para 5 tiques de relógio, (a) – (e)

# O Algoritmo de Substituição de Página do *Conjunto de Trabalho* (1)



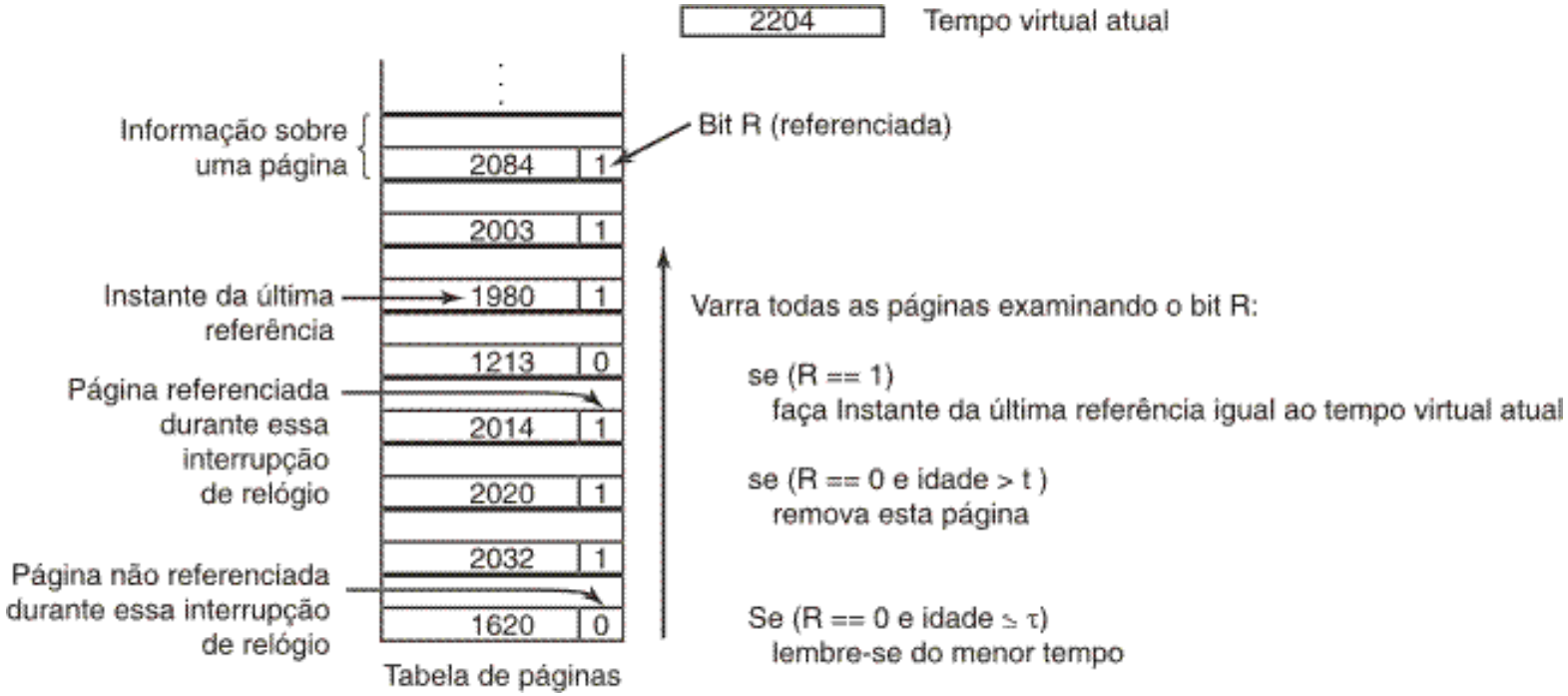

- O conjunto de trabalho é o conjunto de páginas usadas pelas *k* referências mais recentes à memória
- w(k,t) é o tamanho do conjunto de trabalho no instante *t*
- Maior valor k → olhar mais distante no passado
- w(k,t) finito → nenhum programa pode referenciar mais páginas do que suportadas pelo espaço de endereçamento

# O Algoritmo de Substituição de Página do *Conjunto de Trabalho* (1)




- Um processo com faltas de páginas frequentes (baixa localidade de referência) pode provocar ultrapaginação (***thrashing***).
- **Pré-paginação**: carregar um conjunto de páginas do processo antes de iniciar sua execução.

# O Algoritmo de Substituição de Página do *Conjunto de Trabalho* (2)

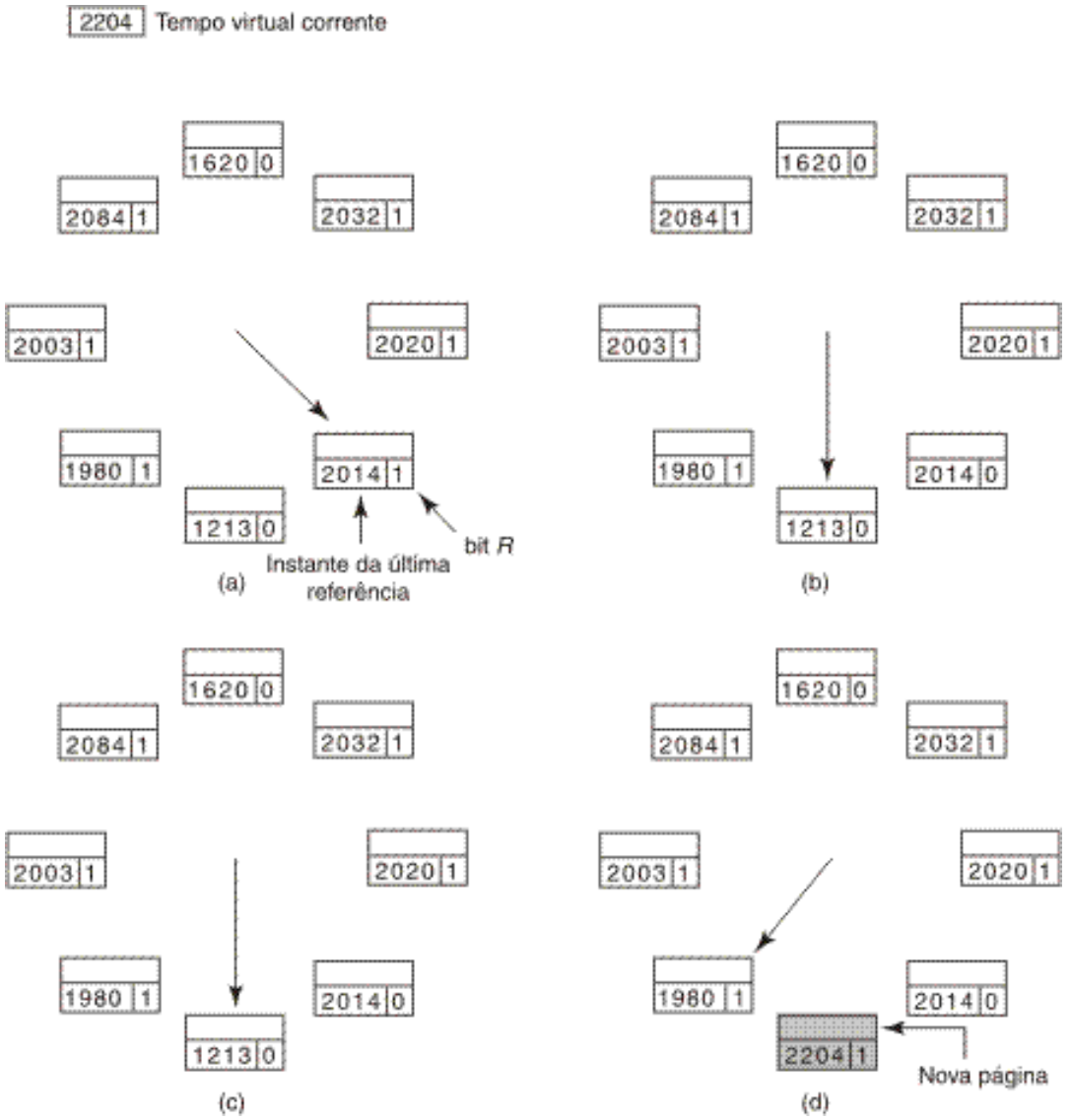



- O algoritmo do conjunto de trabalho: enfadonho, necessita pesquisar a cada *page fault*!

# O Algoritmo de Substituição de Página *WSClock*

## Operação do Algoritmo WSClock:

Caso a página com R=0 esteja suja (*dirty bit* M, o qual não é mostrado na figura), escalona a escrita dela em disco mas continua a procurar por outra (pode ser que haja uma página velha e limpa adiante, evitando assim um escalonamento de outro processo), sendo possível evitar assim uma mudança de contexto! Caso R=0 e M=0 e idade>tau, substitui imediatamente.

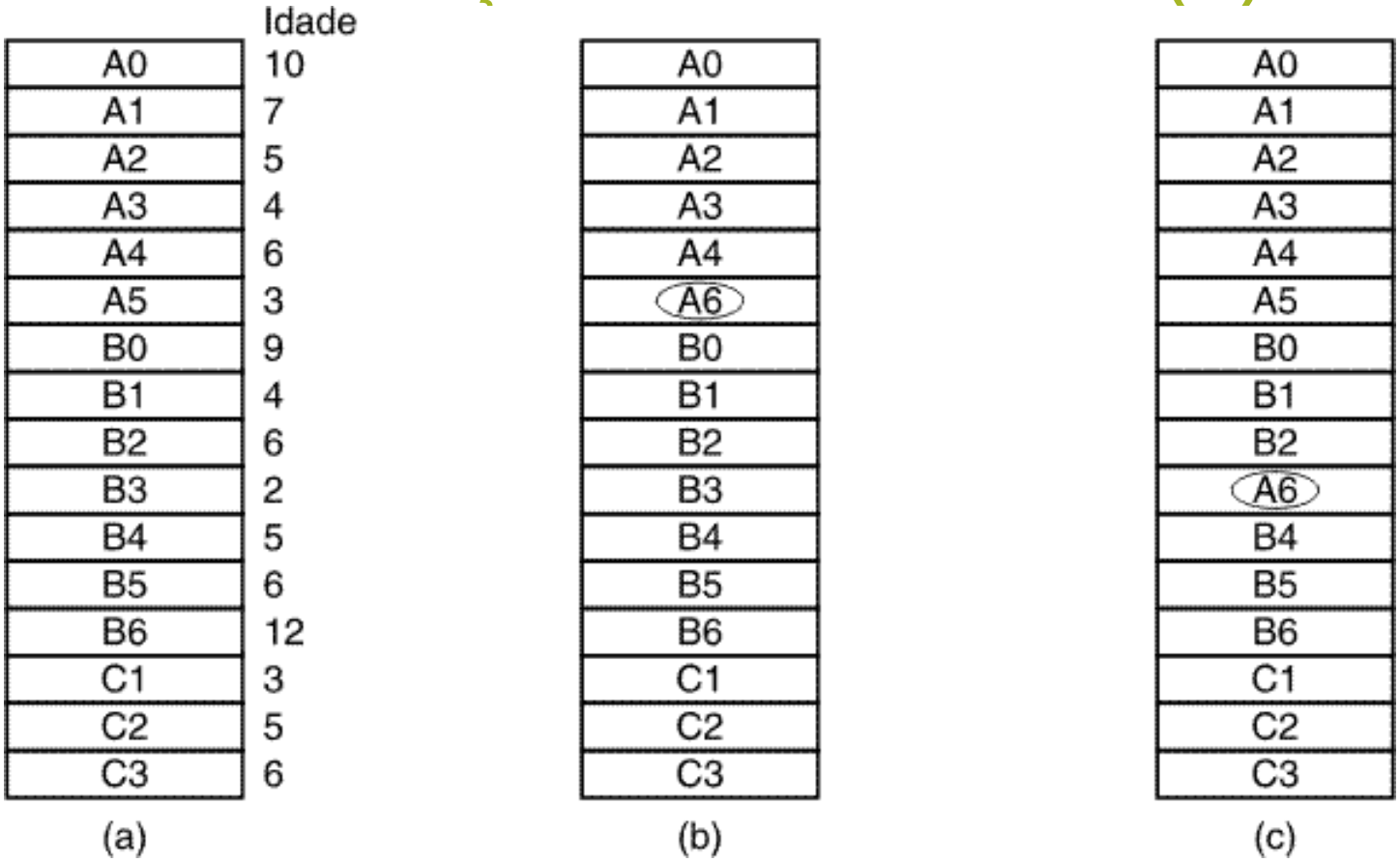


# Revisão dos Algoritmos de Substituição de Página

| 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 (*aging*) | Algoritmo bastante eficiente que se aproxima bem do MRU |
| Conjunto de trabalho | Implementação um tanto cara |
| WSClock | Algoritmo bom e eficiente |

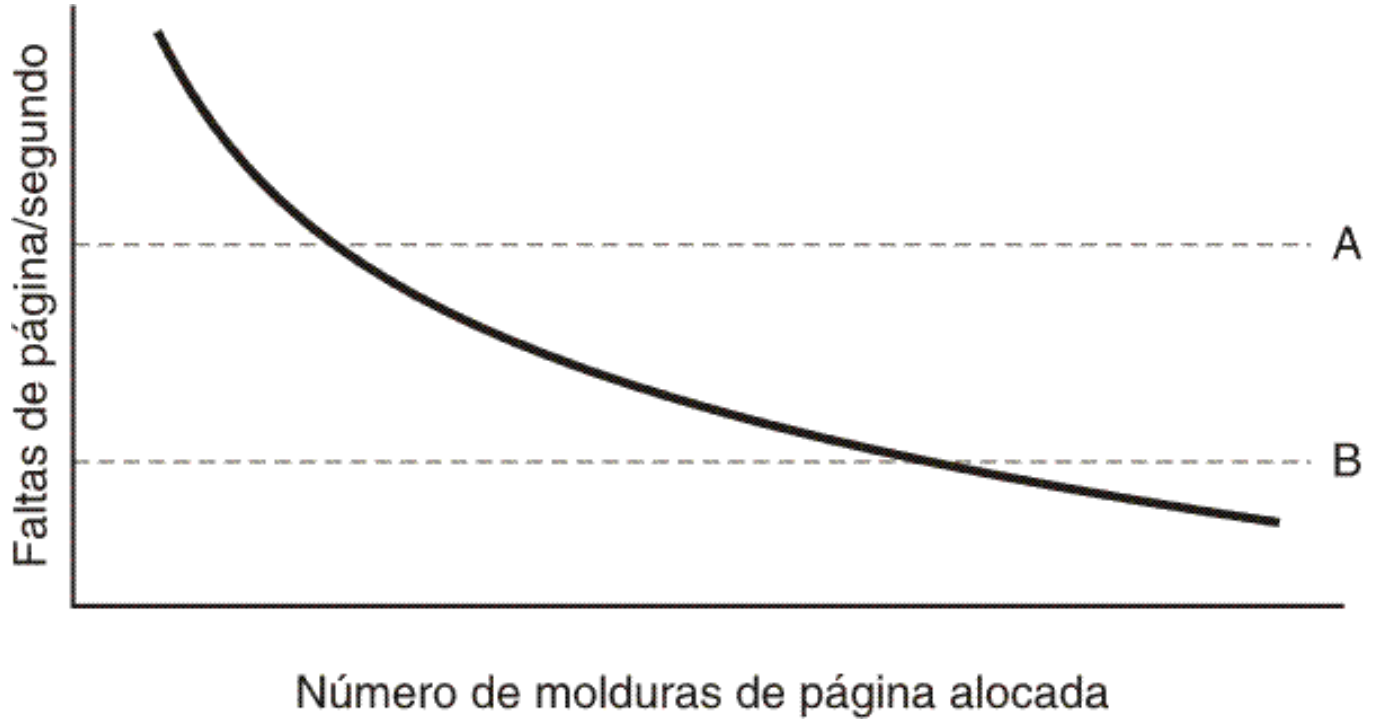
# Questões de Projeto para Sistemas de Paginação

## Política de Alocação Local x Global (1)




- (a) Configuração original (b) Substituição local
- (c) Substituição global

# Algoritmo para tratar a Frequência de Faltas de Páginas (*page fault frequency*, PFF)




Freqüência de faltas de página como função do número de molduras de página alocado: A = *page faults* muito frequentes, significando que o processo tem memória insuficiente; B = *page faults* infrequentes, podendo ser um indicativo que o processo possa ceder um pouco de sua memória.

# Controle de Carga

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

# Tamanho de Página (1)

- Tamanho de página pequeno

- Vantagens
  - menos fragmentação interna (em média p/2 bytes, para páginas com p bytes)
  - menos espaço não usado na memória
- Desvantagens
  - programas precisam de mais páginas, tabelas de página maiores

# Tamanho de Página (2)

- Custo adicional da tabela de páginas e fragmentação interna

$$custo\ adicional$$

Espaço da tabela de páginas

$$overhead = \frac{s \cdot e}{p} + \frac{p}{2}$$

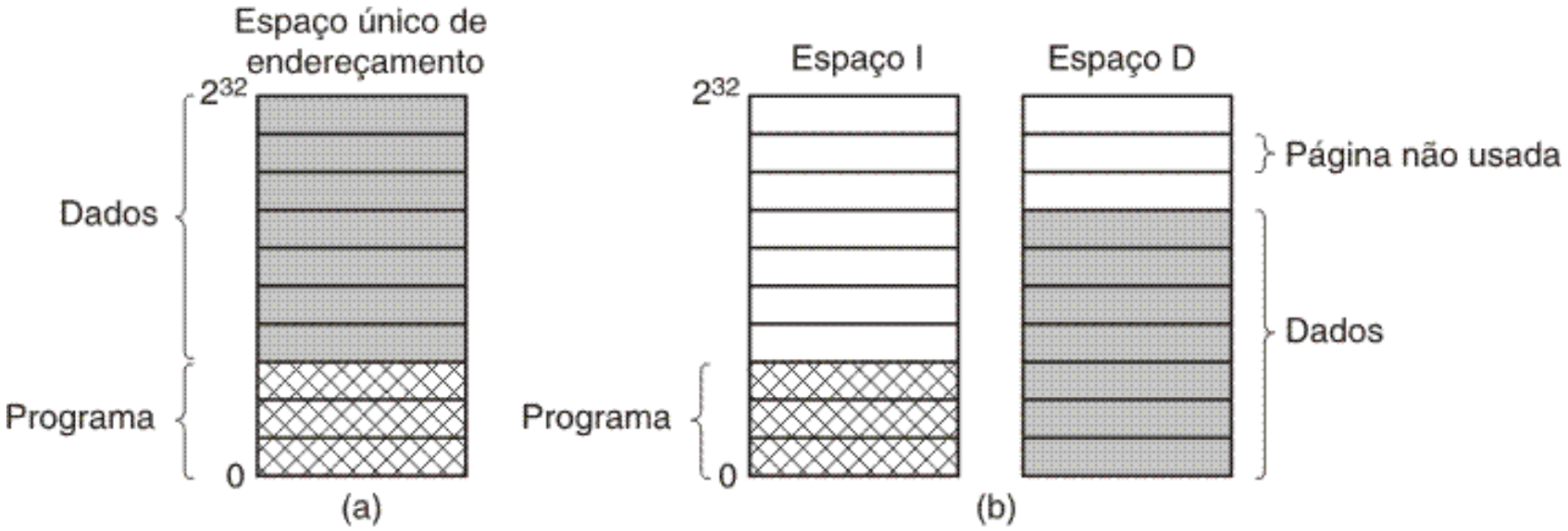
Fragmentação interna

- Onde
  - s = tamano médio do processo em bytes
  - p = tamanho da página em bytes
  - e = tamanho da entrada da tabela de página

O resultado é:

$$p = \sqrt{2se}$$

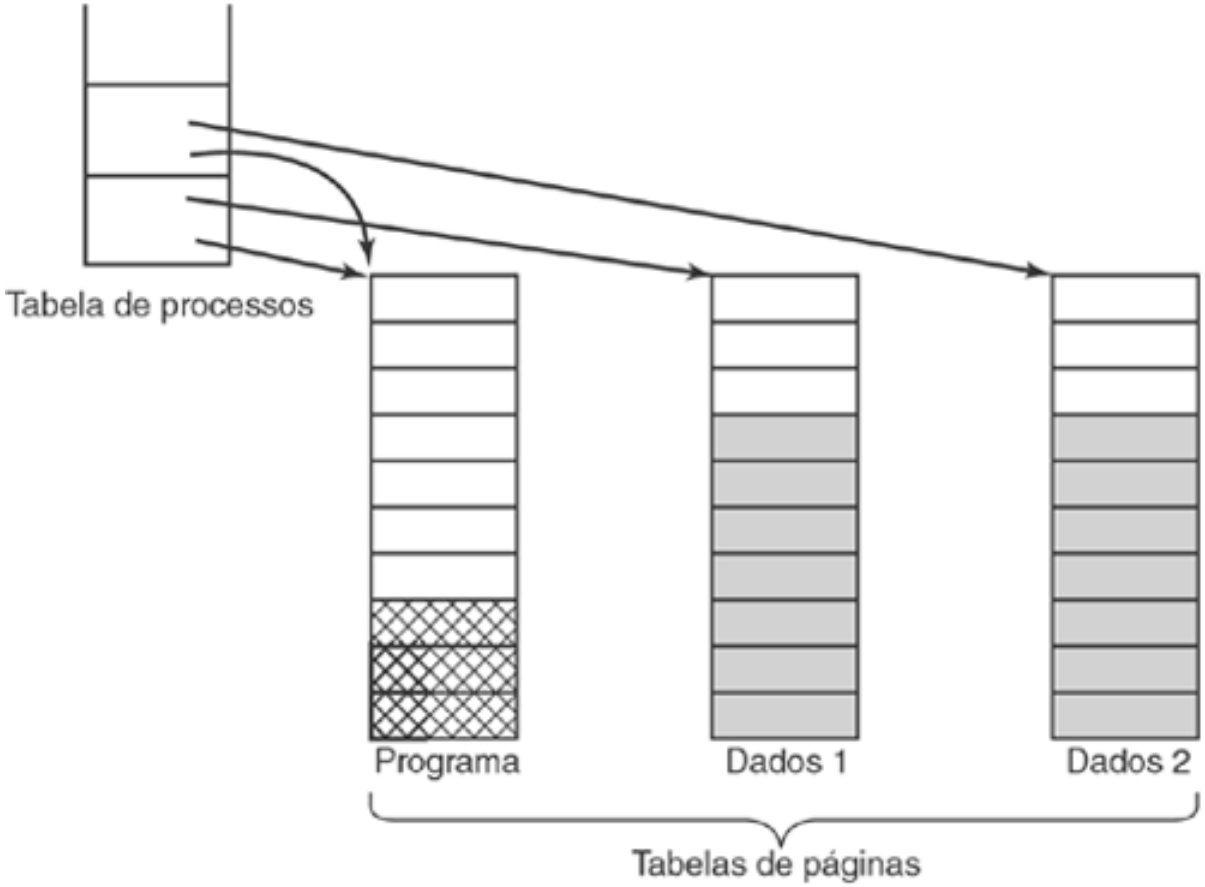
# Espaços Separados de Instruções e Dados




a) Espaço de endereçamento único

b) Espaços separados de instruções (I) e dados (D): duplica o espaço de endereçamento!!

# Páginas Compartilhadas




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

# Política de Limpeza

- Precisa de um **processo de paginação** que executa em *background* (um ***daemon***)
  - Inspeciona periodicamente o estado da memória
- Quando apenas algumas molduras de página estão disponíveis
  - Seleciona páginas a serem removidas usando um algoritmo de substituição
- Pode ser implementada através de lista circular (relógio) com dois ponteiros
  - Ponteiro do início controlado pelo *daemon* de paginação
  - Ponteiro do fim usado para substituição de página (semelhante ao algoritmo do relógio)

# Questões de Implementação
# Envolvimento do S.O. com a Paginação

- Quatro circunstâncias de envolvimento:

1. Criação de processo

– determina tamanho do programa

– cria tabela de página

2. Execução de processo

– MMU reinicia tabela para novo processo

– TLB é esvaziada

3. Ocorrência de falta de página

– determina endereço virtual que causou a falta

– descarta, se necessário, página antiga

– carrega página requisitada para a memória

4. Término de processo

– Libera tabela de páginas, páginas, e espaço em disco que as páginas ocupam

# Tratamento de Faltas de Página (1)

1) Hardware desvia a execução para o núcleo
2) Salva conteúdo de registradores e outras informações voláteis
3) SO determina a página virtual necessária
4) SO checa validade de endereço e tipo de acesso, busca moldura de página disponível (se não disponível, executa algoritmo de substituição);
5) Se moldura de página selecionada foi modificada (suja), salva ela em disco (isso força mudança de contexto para outro processo).

# Tratamento de Faltas de Página (continua lista anterior)(2)

6) SO busca em disco página virtual referenciada

7) Tabela de páginas é atualizada

8) Estado da instrução que causou falta de página é recuperado

9) Processo que causou falta de página é escalado para executar

10) Programa continua

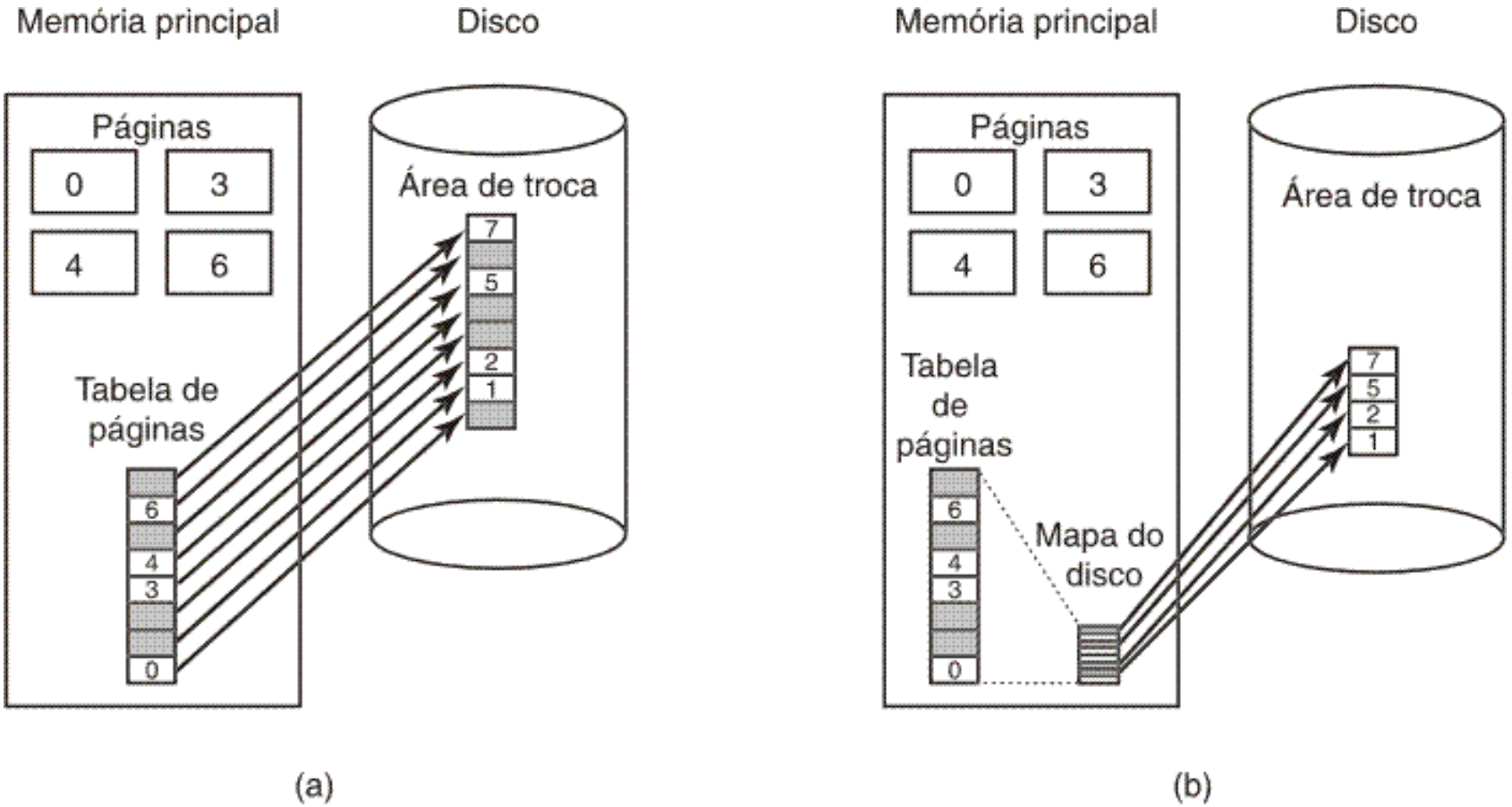
# Recuperação de Instrução


- Uma instrução causadora de falta de página: o **contador de programa** deveria ser salvo (*e.g.*, **em algum registrador escondido**) antes da sua atualização pois, após uma falta de página, nesse exemplo, o contador poderia estar no endereço 1002 ou 1004.

# Retenção de Páginas na Memória

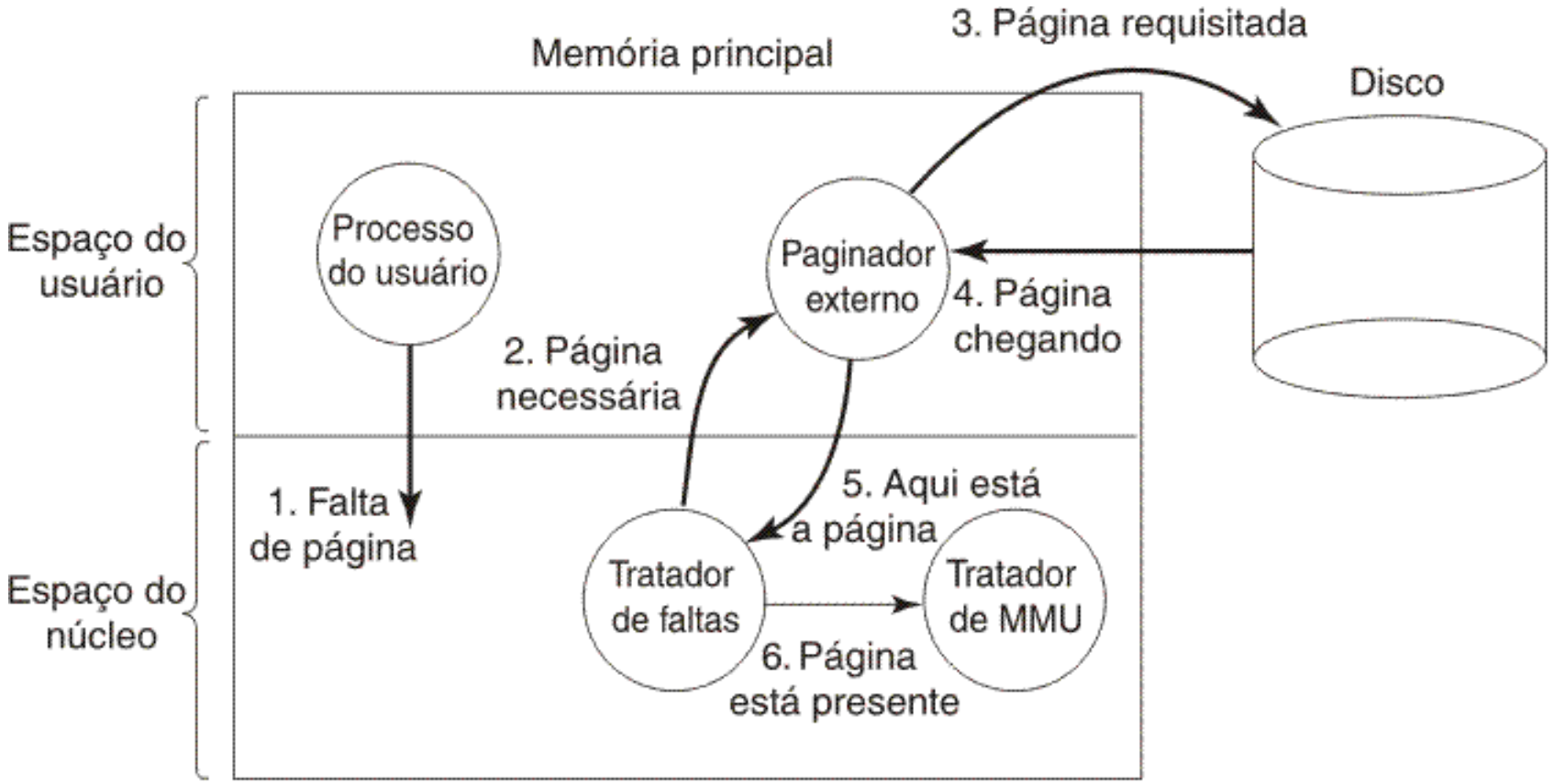
- Memória virtual e E/S interagem ocasionalmente
- Processo emite chamada ao sistema para ler do disco para o *buffer*
  - enquanto espera pela E/S, outro processo inicia
  - ocorre uma falta de página
  - Página onde está o *Buffer* (E/S) do primeiro processo pode ser escolhida para ser levada para disco
- Solução possível
  - Retenção (*pinning*) de páginas envolvidas com E/S na memória
  - Fazer operações E/S para *buffers* do S.O., depois tranferir para área de memória do processo

# Memória Secundária (espaço de ***swap***)



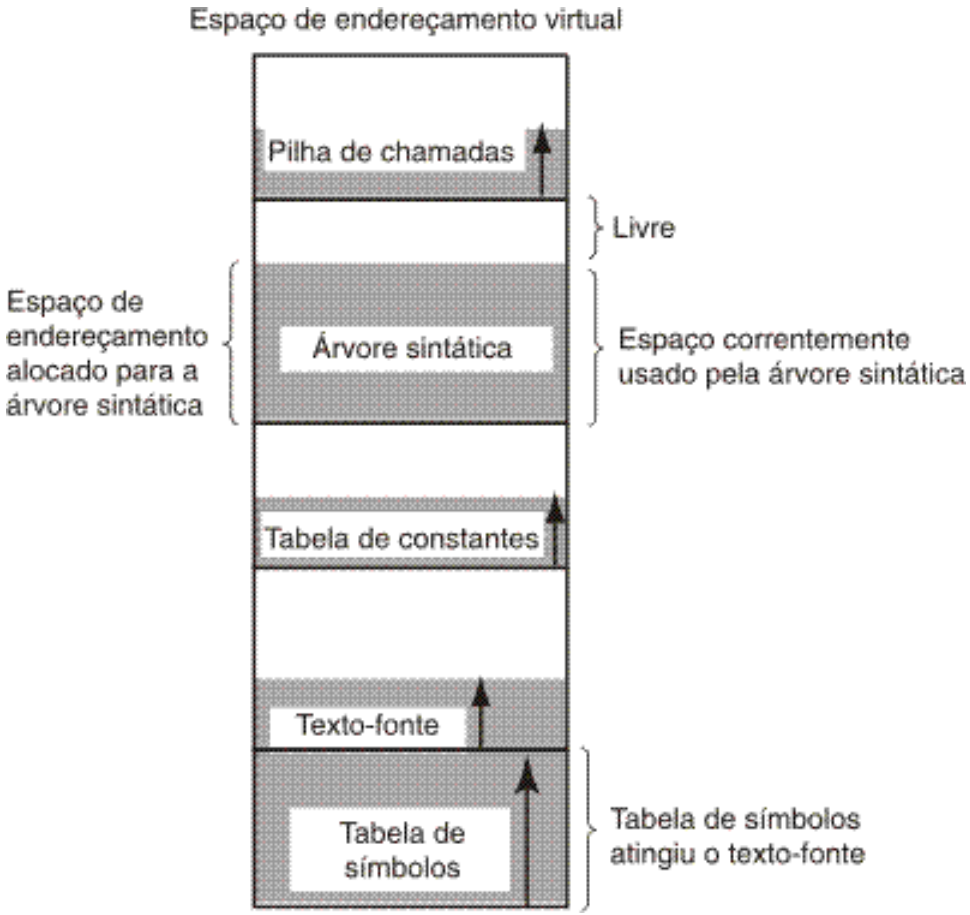

- (a) Paginação para uma área de troca estática (páginas são **espelhadas** no disco)
- (b) Páginas **alocadas dinamicamente em disco** (contém **somente páginas retiradas da memória**)

# Separação da Política e do Mecanismo



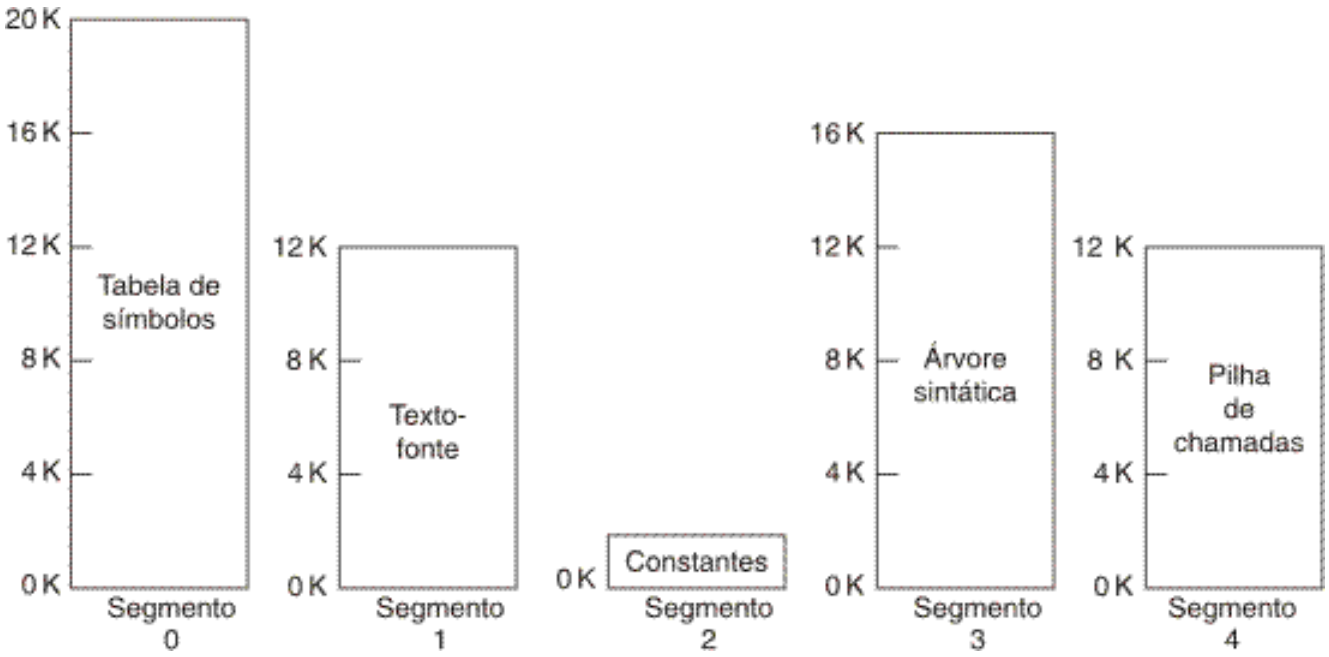

- Tratamento de faltas de página com paginador externo

# Segmentação (1)




- Espaço de endereçamento unidimensional com tabelas crescentes (**nesse exemplo: um compilador**)
- Uma tabela pode atingir outra

# Segmentação (2)



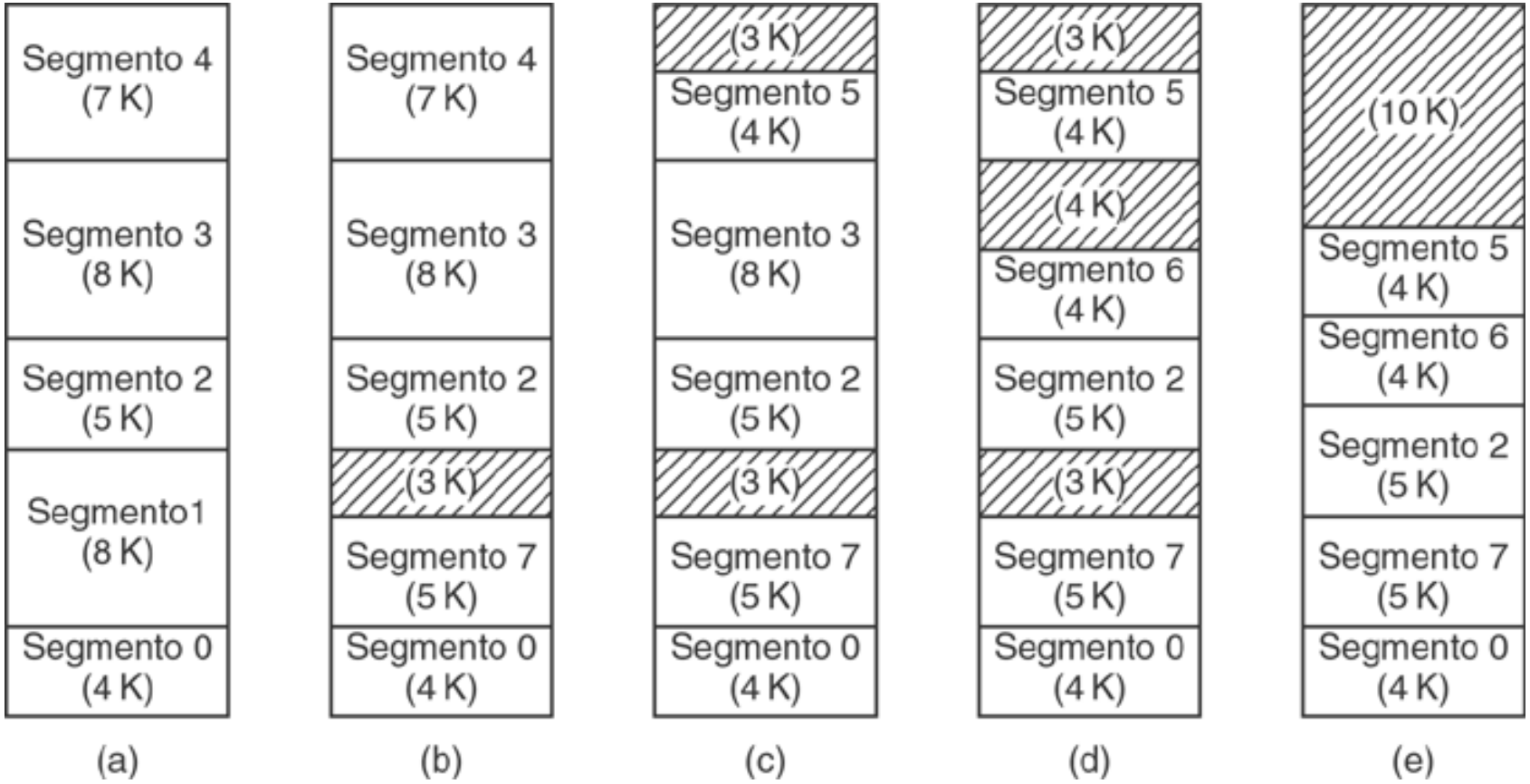

- Permite que cada tabela cresça ou encolha, independentemente

# Segmentação (3)

| 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 |

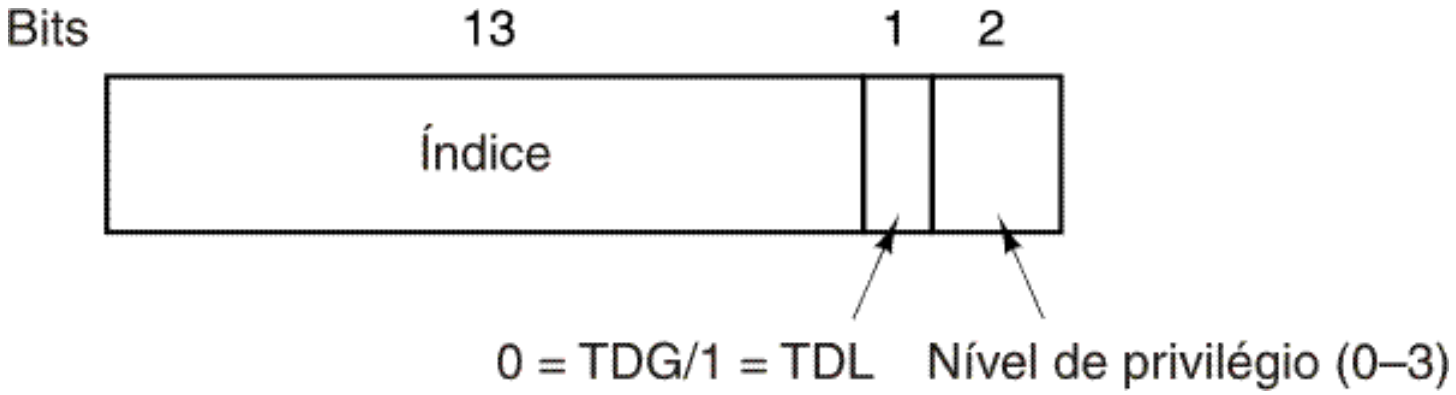
- Comparação entre paginação e segmentação

# Implementação de Segmentação Pura



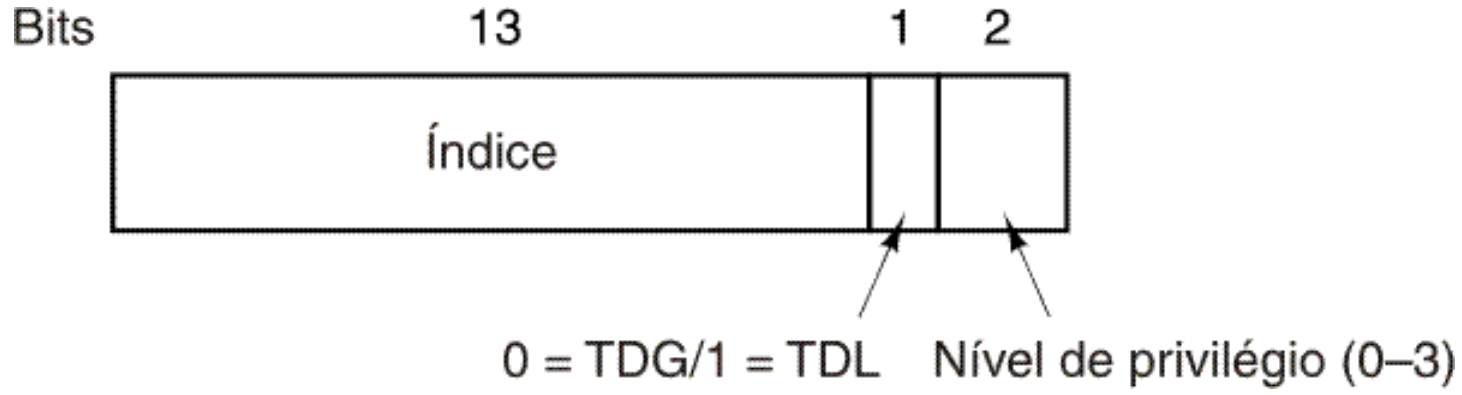

- (a)-(d) Desenvolvimento de fragmentação externa
- (e) Remoção da fragmentação via “compactação”

# Segmentação com Paginação: Pentium (1): seletor do *Pentium*



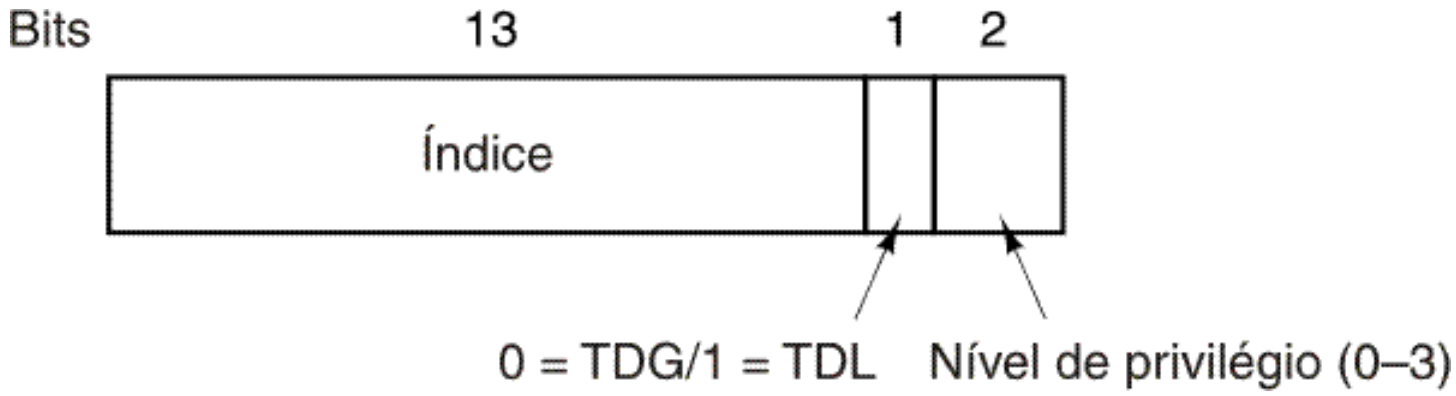

- Endereço: (seletor, offset)
- Índice: número da entrada na tabela TDG ou TDL

# Segmentação com Paginação: Pentium (1)



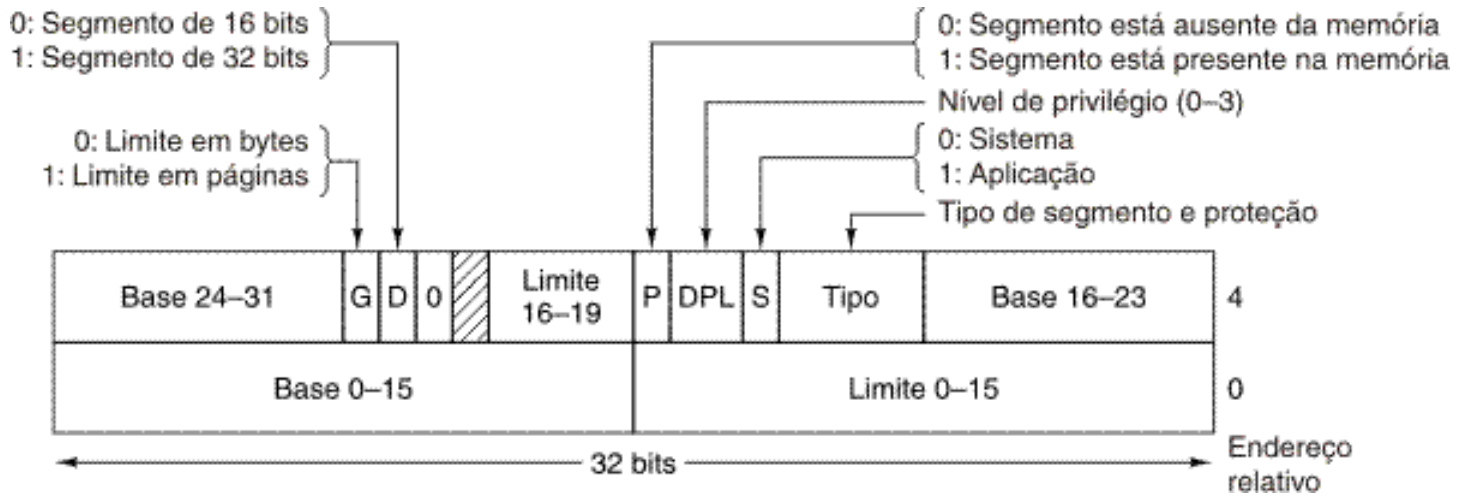

- Um seletor do Pentium: *Local descriptor Table* (LDT) e *Global Descriptor Table* (GDT)
- Há uma LDT por processo e apenas uma GDT (descreve os segmentos do S.O.)

# Segmentação com Paginação: Pentium (1)



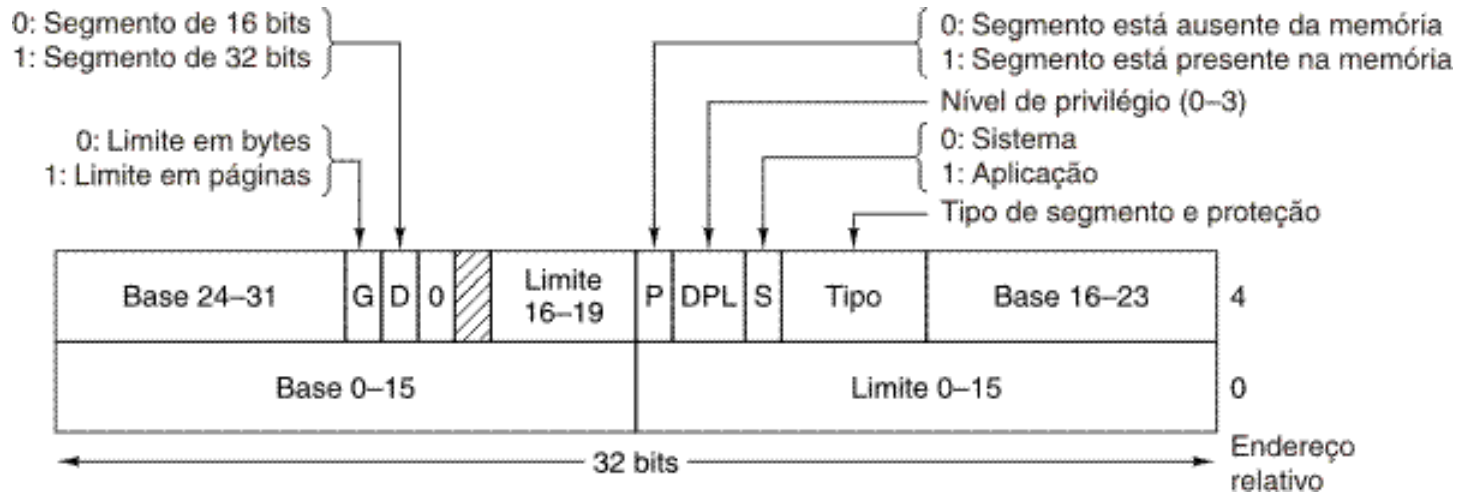

- Em resumo: o seletor de segmento indica se trata-se de uma entrada na TLD ou GDT (8K entradas em cada tabela).
- Seletores de segmentos são carregados em registradores especiais (i.e., registradores de segmentos).

# Segmentação com Paginação: Pentium (2)



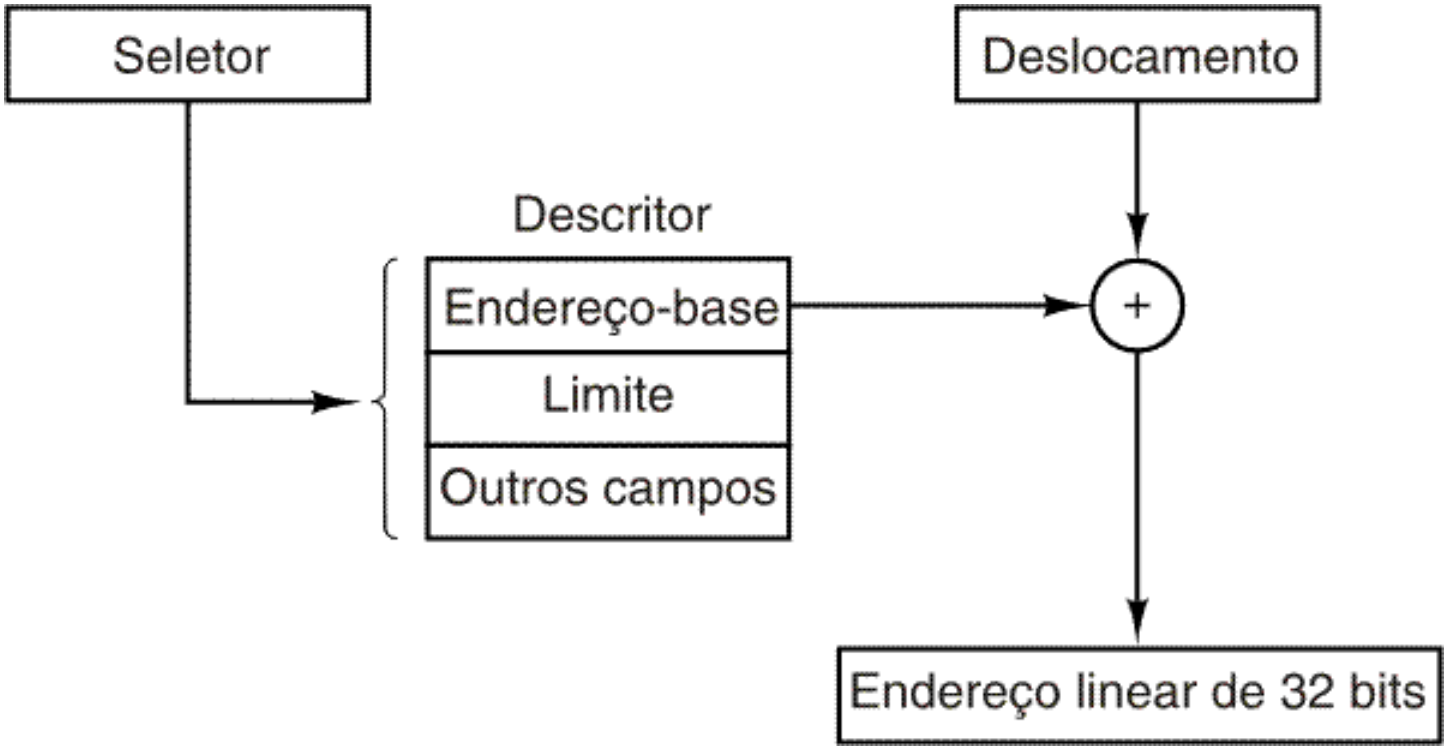

- Descritor de segmento de código do Pentium
- (segmentos de dados diferem ligeiramente)

# Segmentação com Paginação: Pentium (2)



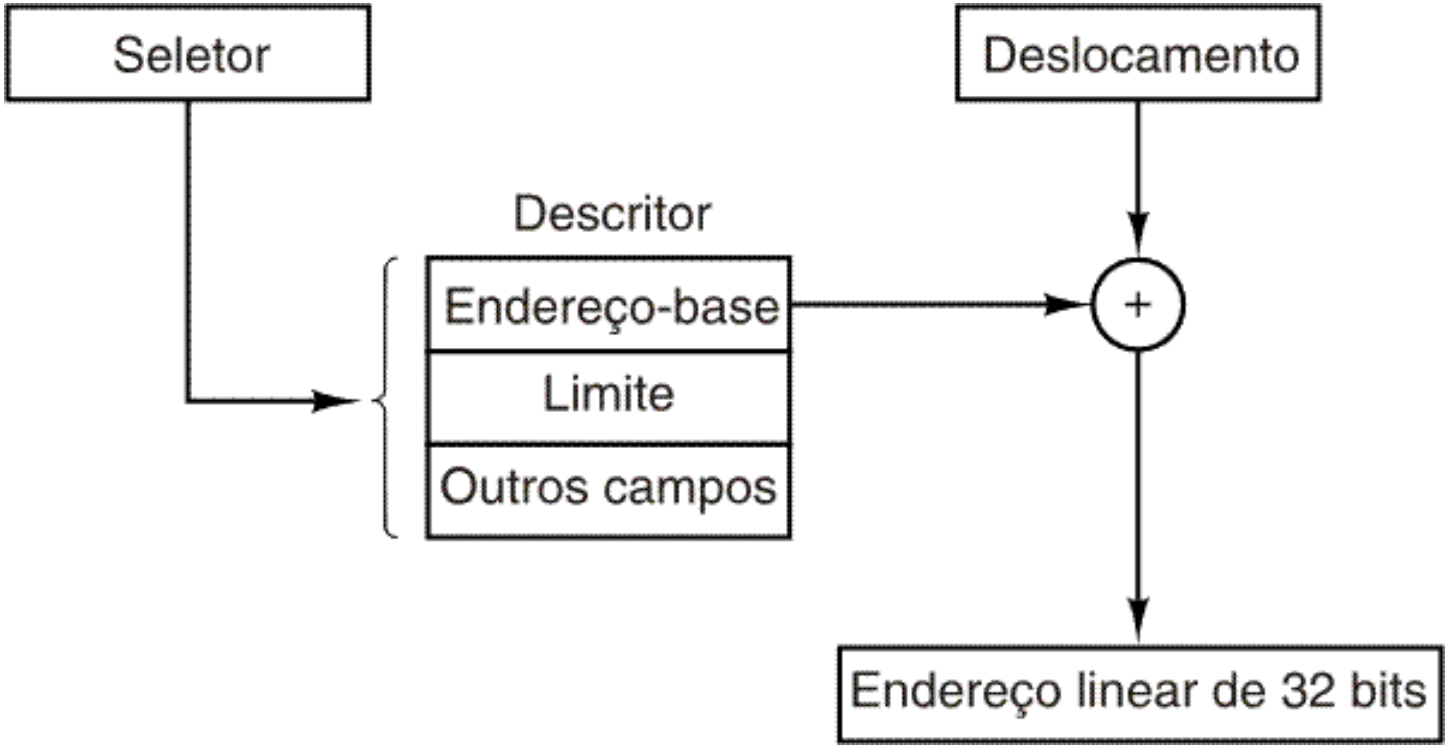

- Caso G=0, tem-se segmentos de até 1 Mbytes; caso contrário, utiliza-se páginas de 4 Kbytes, com até $2^{20}$ páginas totalizando segmentos $2^{32}$ Bytes

# Segmentação com Paginação: Pentium (3)



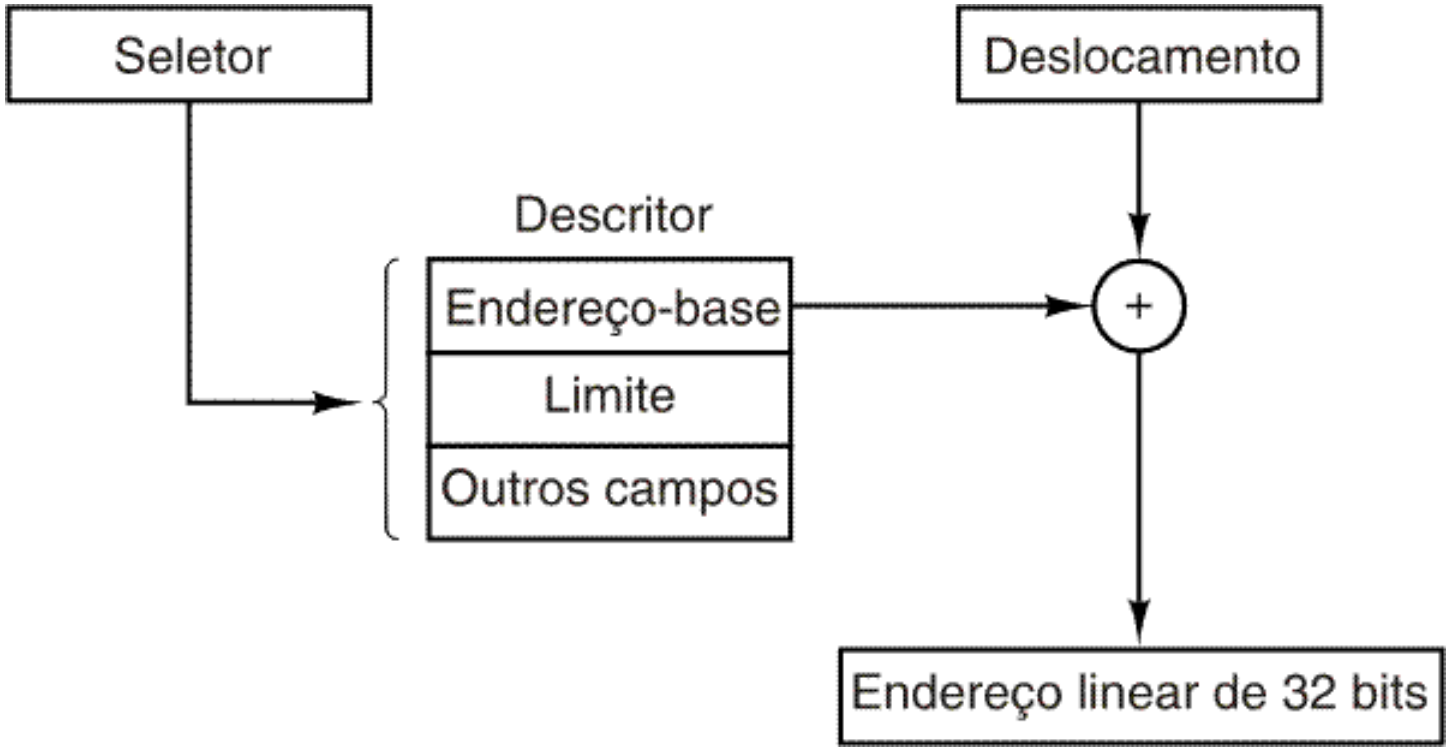

- Conversão de um par (seletor, deslocamento) para um endereço linear

# Segmentação com Paginação: Pentium (3)



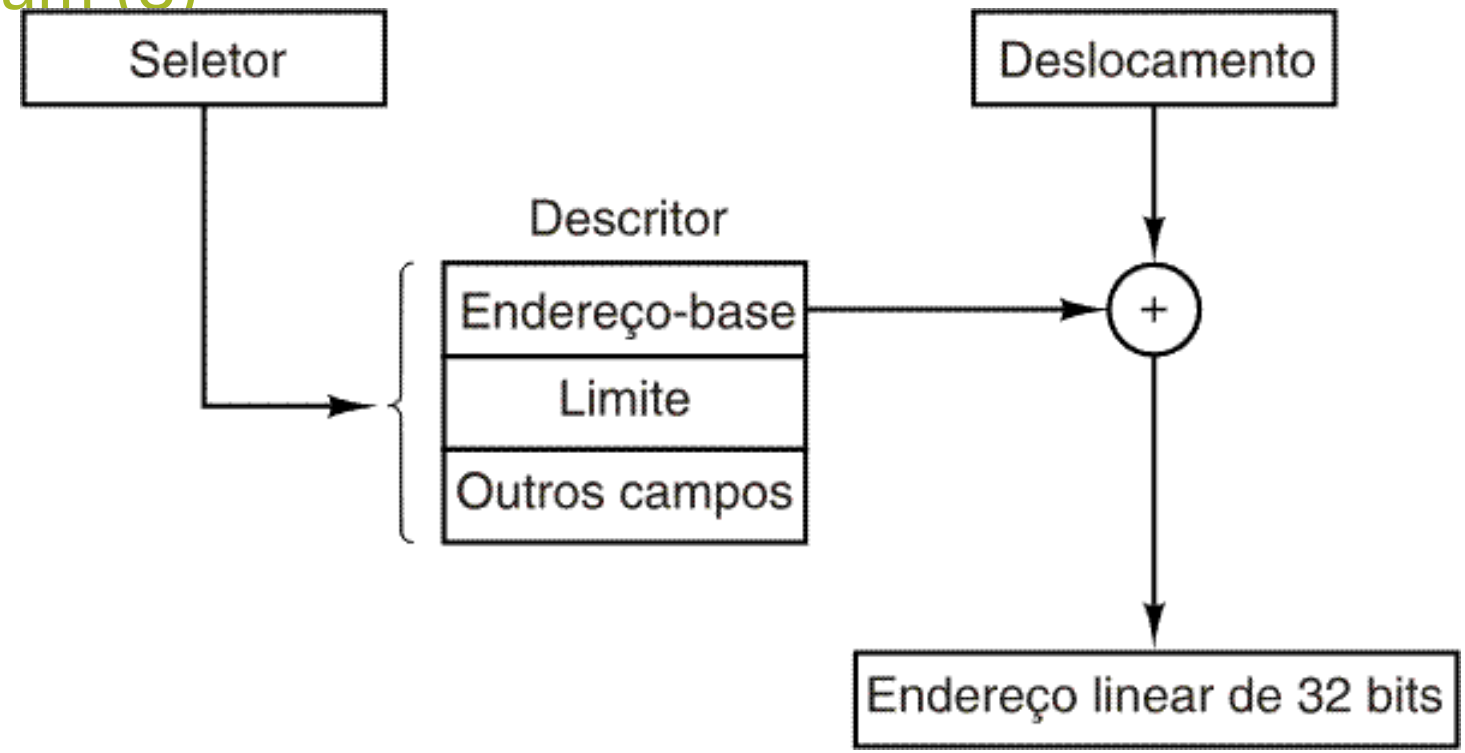

- Seletor + deslocamento: o endereço base é adicionado ao deslocamento produzindo um endereço linear
- O endereço base está “quebrado” no descritor para manter compatibilidade com o 286 (base de 24 bits)

# Segmentação com Paginação: Pentium (3)



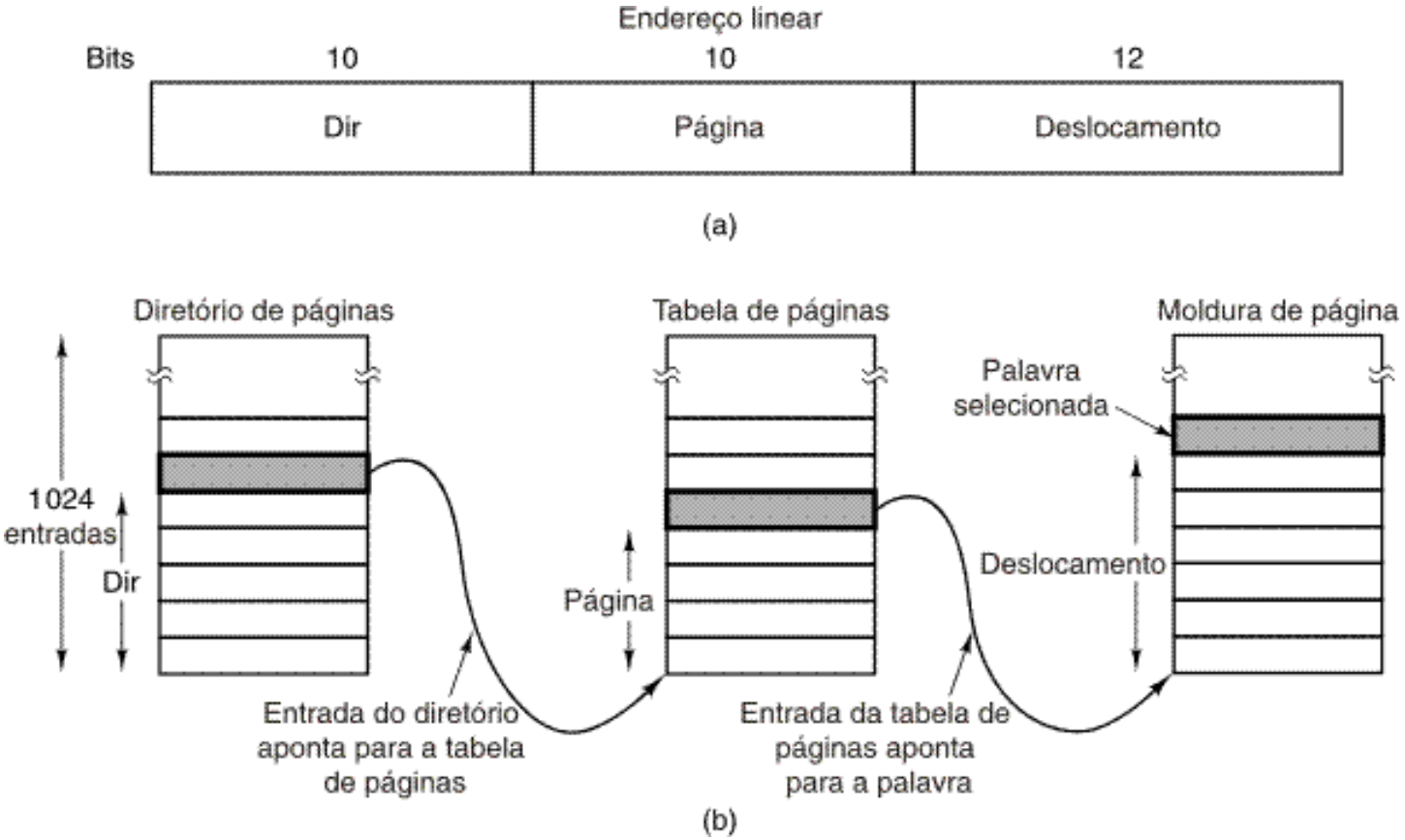

- Caso paginação esteja DESABILITADA (1 bit no registrador de controle global), o endereço linear é interpretado como endereço físico; portanto, tem-se segmentação pura!
- É permitido sobreposição de segmentos (visto que fica muito complexo tratar sobreposição!)

# Segmentação com Paginação: Pentium (3)



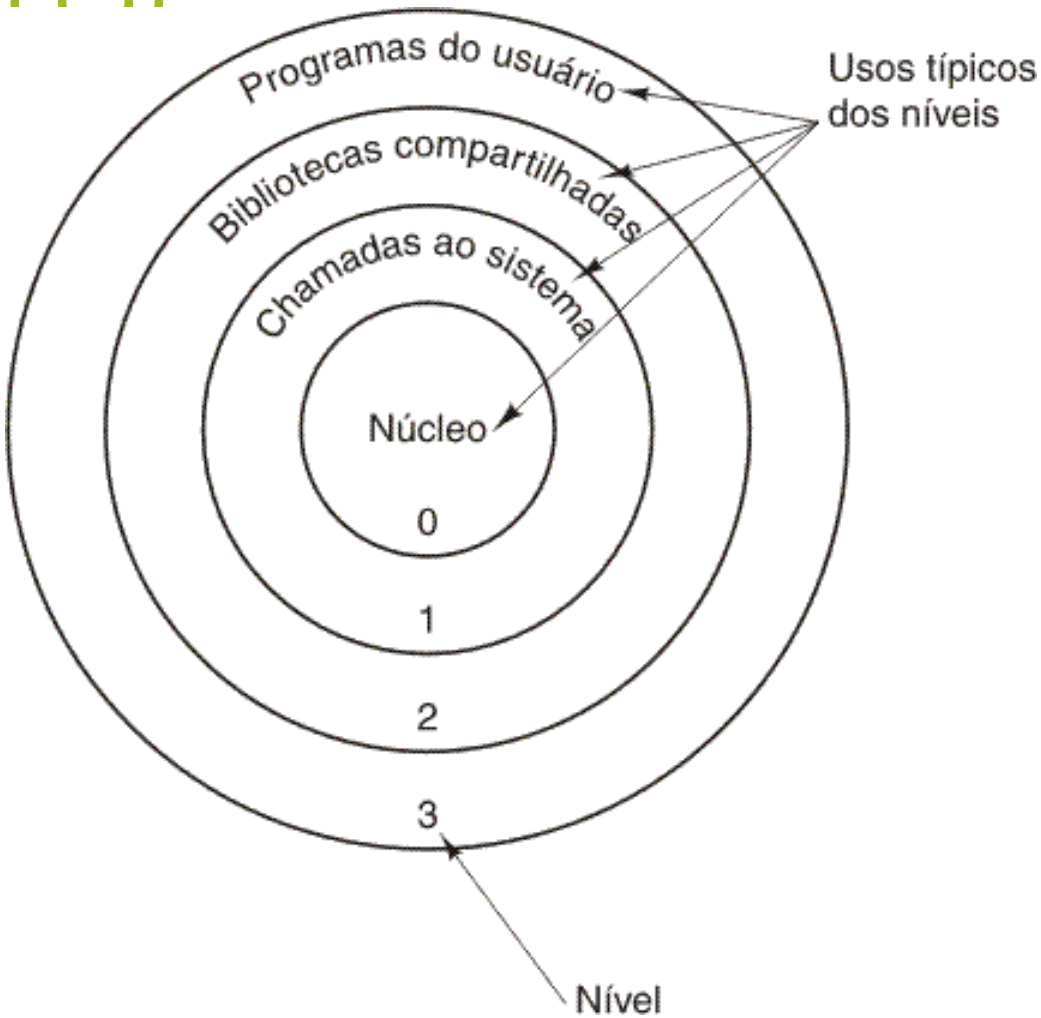

- Caso paginação esteja HABILITADA, utiliza-se tabelas de páginas como visto antes; no entanto, devido ao tamanho da página (mais de um milhão de entradas), utiliza-se mapeamento em dois níveis!

# Segmentação com Paginação: Pentium (4)




- Mapeamento de um endereço linear sobre um endereço físico

# Segmentação com Paginação: Pentium (5)




- Proteção no Pentium