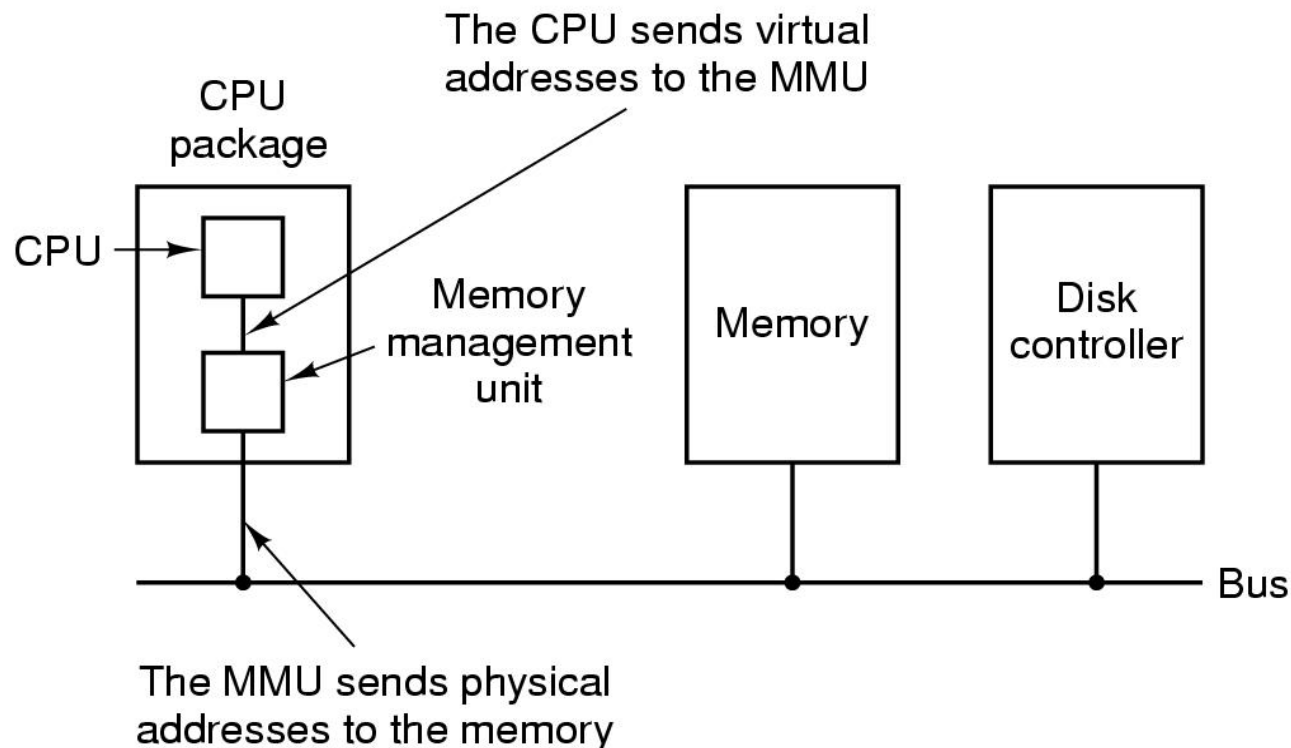


Gerência de Memória

Paginação

Endereçamento Virtual ⁽¹⁾

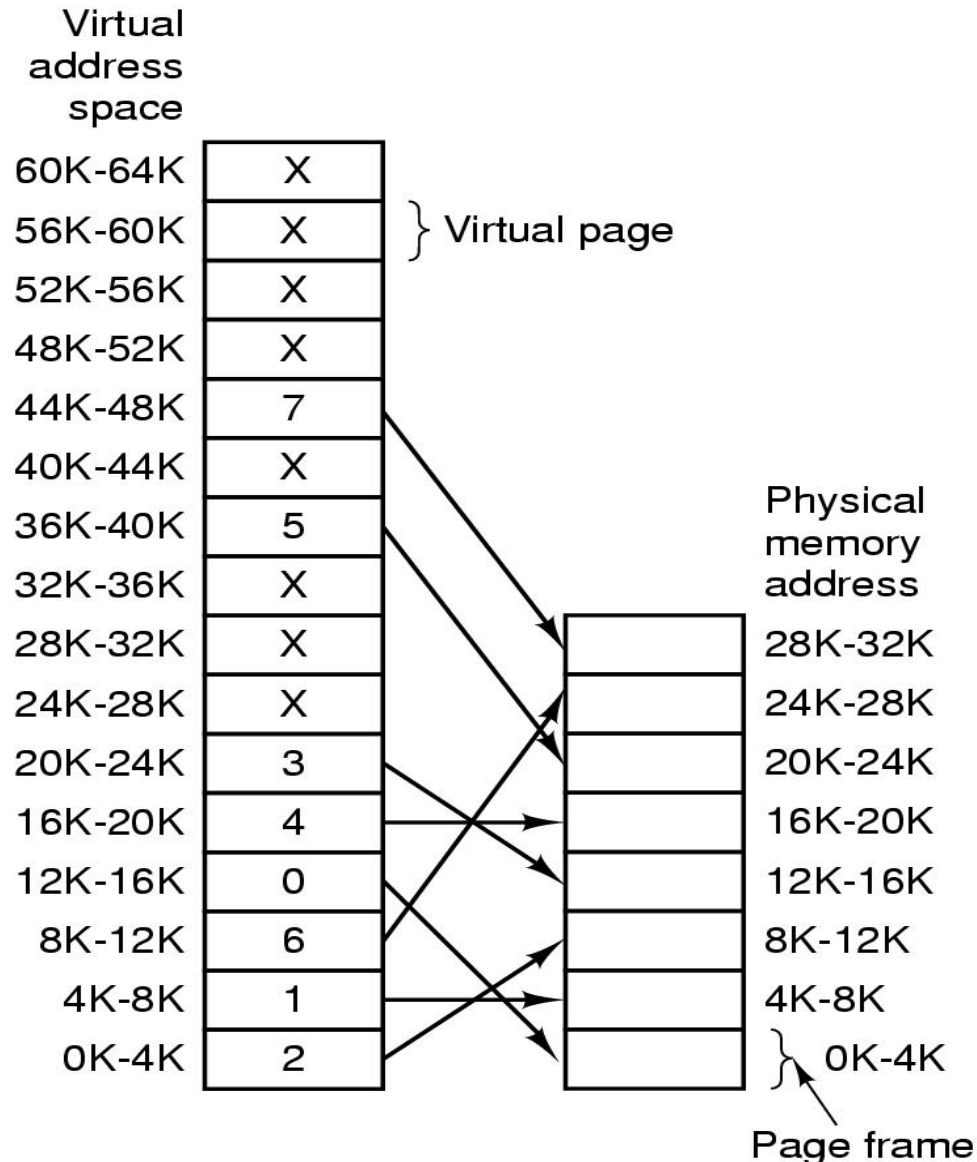
- Espaço de endereçamento dos processos não linearmente relacionado com a memória física
- Cada vez que são usados, os endereços virtuais são convertidos pela MMU para endereços reais



Endereçamento Virtual (2)

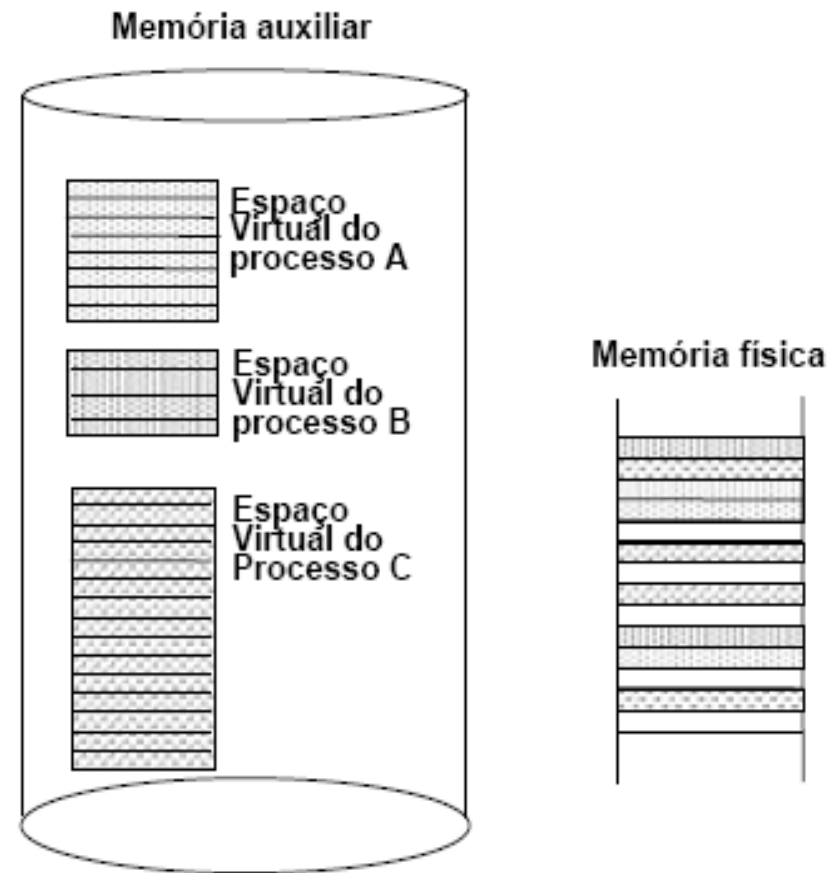
■ Exemplo

- Computador capaz de gerar endereços virtuais de 16 bits (0-64k).
- Memória física de apenas 32k: programas não podem ser carregados por completo na memória física
- Solução: dividir o programa em **Páginas**



Endereçamento Virtual (3)

- **Exemplo** (cont.)
 - Uma cópia completa do programa, de até 64k deve estar presente em disco, de modo que partes (páginas) possam ser carregadas dinamicamente na memória quando necessário
 - Apenas precisam estar na memória principal as páginas que estão sendo utilizadas por cada processo



Memória virtual: Paginação

- Processo é dividido em **Páginas**
- A Memória é dividida em **Molduras** (ou *Frames*) de mesmo tamanho
 - Tamanho das Páginas = tamanho das Molduras
- Páginas/Molduras são de pequeno tamanho (e.g., 1K):
 - fragmentação interna pequena
- Processo não precisa ocupar área contígua em memória
 - Elimina fragmentação externa
- Processo não precisa estar completamente na MP
- SO mantém uma **tabela de páginas** por processo
- Endereços são gerados dinamicamente em tempo de execução

Paginação: Como funciona?

- Para minimizar a informação necessária à conversão, a memória virtual é logicamente dividida em páginas
 - Endereço virtual = (nº da página , deslocamento)
- No exemplo anterior (end. virtuais de 16 bits=> processos de até 64k; memória física de 32k; páginas/molduras de 4k)
 - São necessários 4 bits para referenciar todas as 16 páginas do processo
 - End. virtual = (nº da página [4 bits] , deslocamento [16 - 4 bits])
 - Instrução LDW r1,50
 - O end. virtual 50 é enviado à MMU
 - Ela detecta que esse end. virtual situa-se na página virtual 0 (de 0 a 4095) que, de acordo com o seu mapeamento, corresponde à moldura de página 2 (end. físicos de 8192 - 12287)

Paginação: Como funciona?

(2)

LDW r3, 20500

Qual é a página?

Pag. 5, que contém os endereços de 20k (20480) até 24k-1 (24575)

Esta página está em qual moldura?

Na moldura 3, que contém end. físicos de 12k (12288) a 16k-1 (16384)

Qual o deslocamento do endereço 20500 dentro da página?

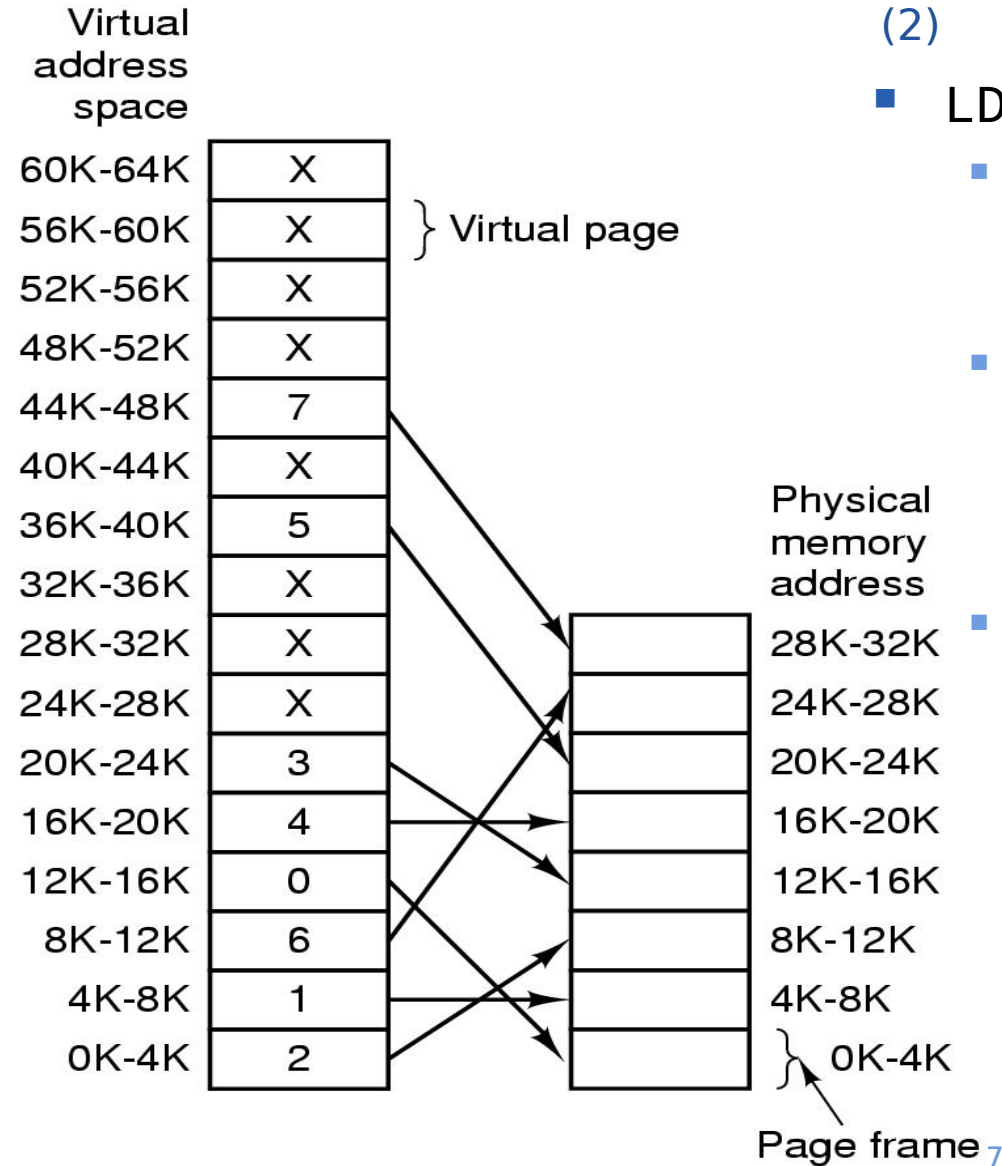
Desl. = End. virtual - End. virtual do 1º byte da página

$$= 20500 - 20480 = 20$$

Qual será o endereço físico correspondendo ao end. virt. 20500?

= End. do 1º byte da moldura + desloca.

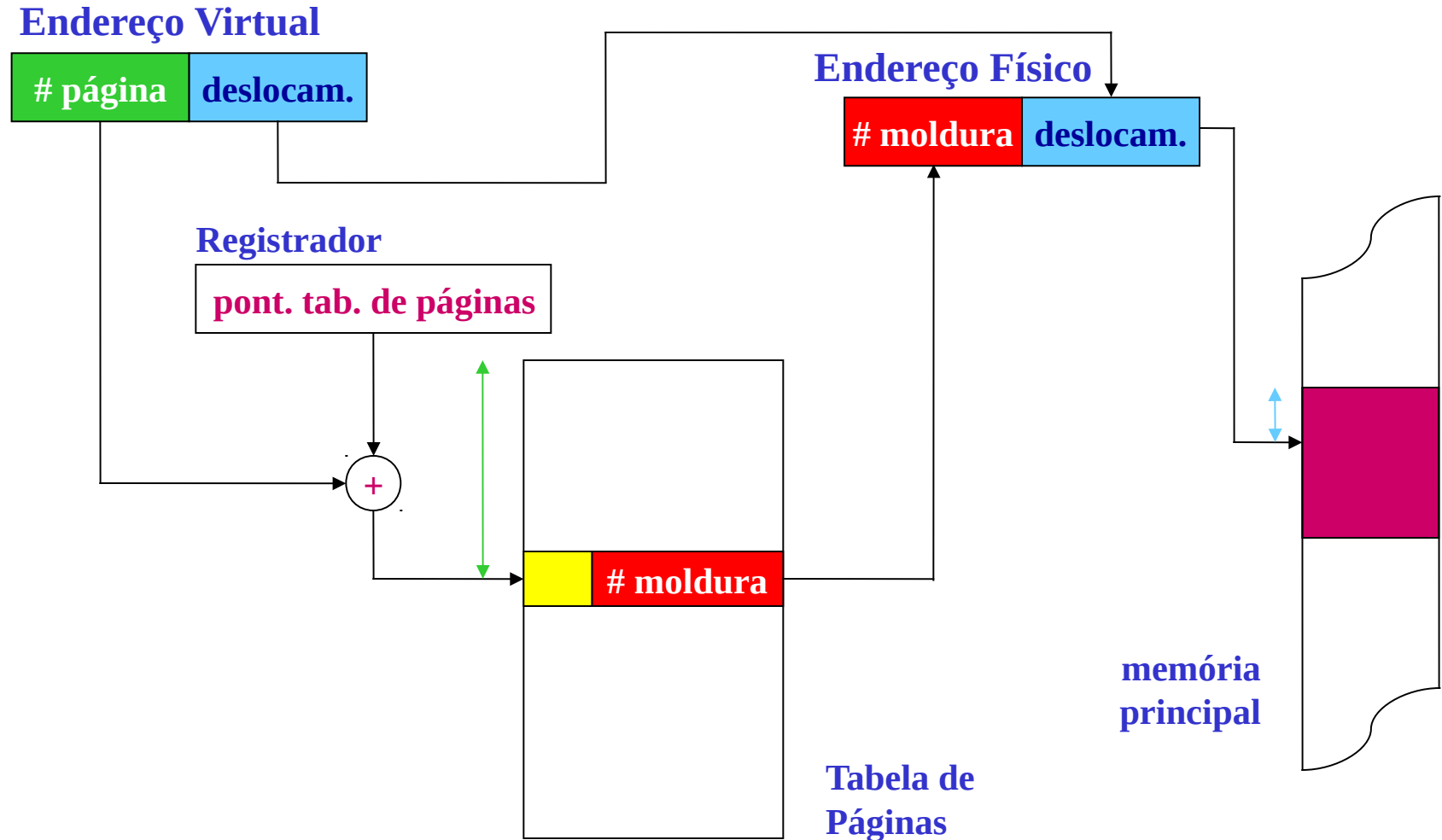
$$= 12288 + 20 = 12308$$



Paginação: Como funciona?

- Cada processo tem sua **Tabela de Páginas**
- Tabela de Páginas faz o mapeamento página x moldura
- O que acontece se o programa faz um acesso a uma página que não está mapeada na memória?
- Tabela de páginas pode estar só parcialmente na MP
- Dois acessos à MP

Paginação: Endereçamento



Paginação: Endereçamento – Exemplo (1)

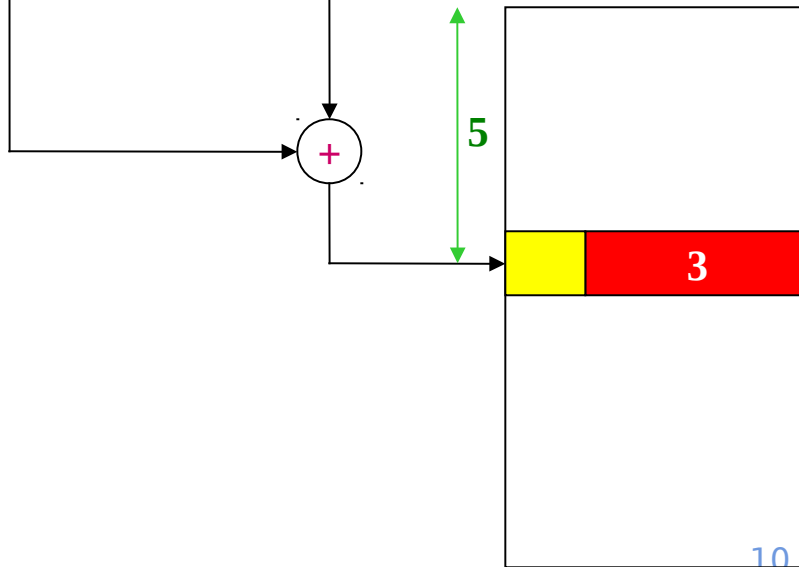
Endereço Virtual 20500

0101 0000 0001 0100
4 bits 12 bits

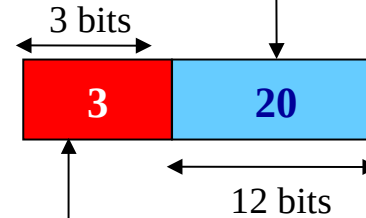


Registrador

pont. tab. de páginas



Endereço Físico



memória principal

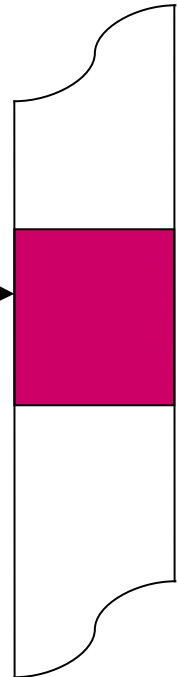
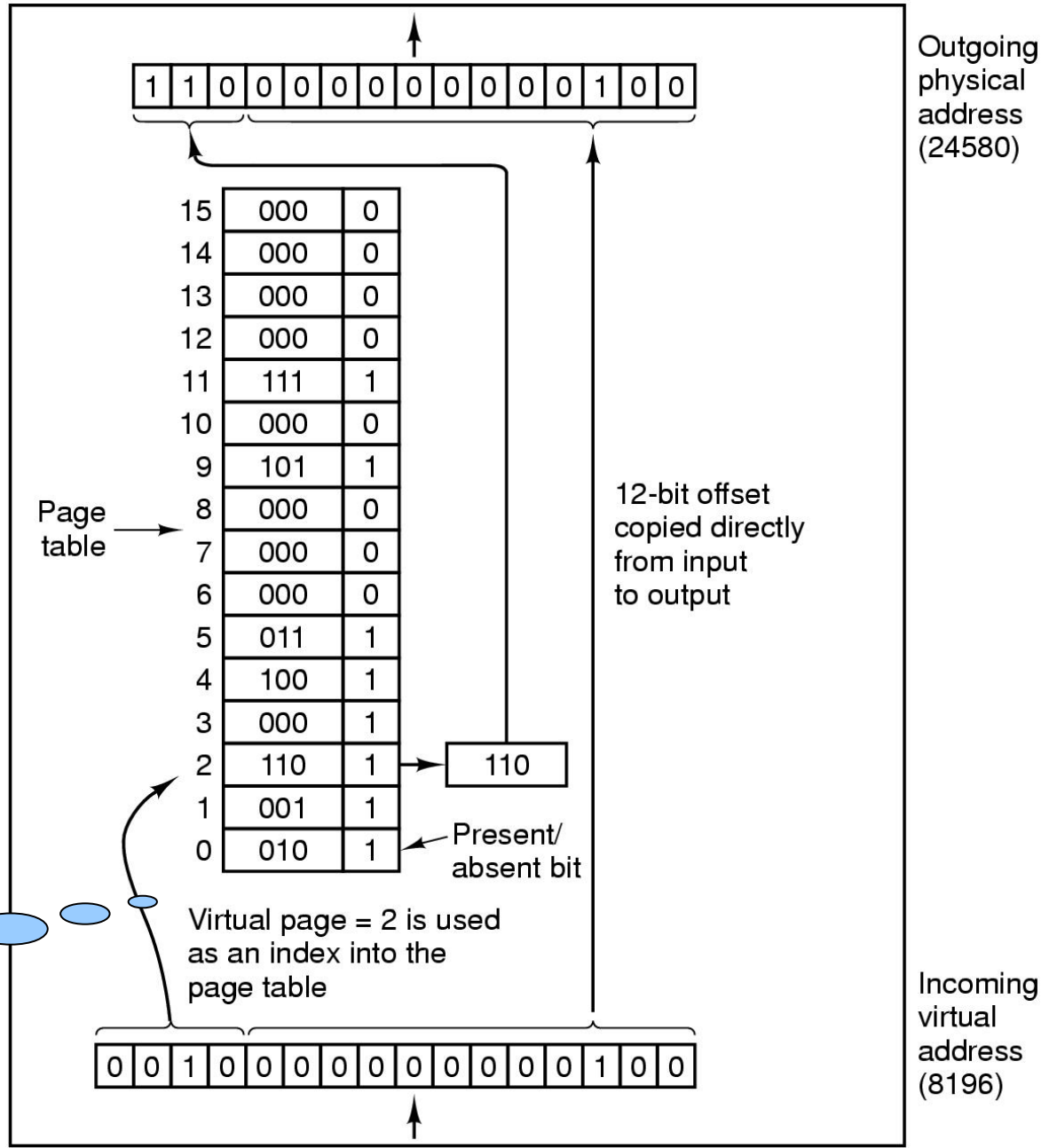


Tabela de Páginas

Paginação: Endereçamento – Exemplo (2)

- Operação interna de uma MMU com 16 páginas de 4 kB

O nº da pag.
é usado
como índice



Paginação: Como funciona?

- O que acontece se o programa faz um acesso a uma página que não está mapeada na memória?
 - Ocorre uma *Page Fault* => a MMU força uma interrupção
- Ação do S.O.
 - Escolher uma página pouco usada, que encontra-se em alguma moldura da memória principal
 - Salvar esta página no disco (caso ela tenha sido modificada)
 - Carregar a página virtual referenciada pela instrução na moldura recém liberada
 - Atualizar o mapeamento da tabela de páginas e reiniciar a instrução causadora da interrupção

Tabela de Páginas ⁽¹⁾

■ Problemas

■ Ela pode ser muito grande

- Suponha uma máquina de 32 bits, 4k por página

$$\frac{2^{32} \text{ endereços virtuais}}{4k = 2^{12}} = 2^{20} \text{ entradas na tabela de páginas}$$

- E uma máquina de 64bits !?! (2^{52} entradas)
- Deve-se utilizar mecanismos para diminuir o tamanho da tabela

■ O mapeamento deve ser rápido

- Mapeamento para buscar a instrução na memória
- Instruções podem conter operandos que também encontram-se na memória

Tabela de Páginas (2)

- Projeto mais simples:
 - uma única tabela de páginas que consista em um vetor de registradores rápidos em hardware (um reg. para cada entrada)
 - Qdo o processo estiver para ser executado, o S.O. carregará esses reg. A partir de uma cópia da tab. de páginas desse processo mantida na memória
 - Vantagem: não requer nenhum acesso à memória durante a tradução
 - Desvantagens:
 - CARO!!!
 - Ter que carregar toda a tabela de páginas em cada troca de contexto

Tabela de Páginas ⁽³⁾

- Segunda opção:
 - Tabela de páginas totalmente na memória
 - O HW necessário resume-se a um único registrador (que aponta para o início da tabela de páginas)
 - Desvantagem:
 - A execução de uma instrução implicará em pelo menos dois acessos a memória
 - O primeiro, para acessar a tabela de páginas (e descobrir o endereço físico desta instrução)
 - O segundo, para buscar a respectiva instrução na memória
 - Isso sem falar nos operandos da instrução que podem estar em memória...

Tabela de Página Multinível ⁽¹⁾

- O objetivo é evitar manter toda a tabela de páginas na memória durante todo o tempo
- Apresenta-se como uma solução para o dimensionamento da tabela de páginas
- Uso de dois apontadores e um deslocamento
- Exemplo: Tabela de dois níveis
 - O endereço de 32 bits de endereço dividido em 3 campos
 - PT1 [10 bits] : indexa o primeiro nível da tabela
 - PT2 [10 bits] : indexa o segundo nível da tabela
 - Deslocamento [12 bits]: => paginas de 4 KB

Tabela de Página Multinível (2)

- 1º nível com 1024 entradas
- Cada uma dessas entradas representa 4 MB
 - $4 \text{ GB} / 1024$

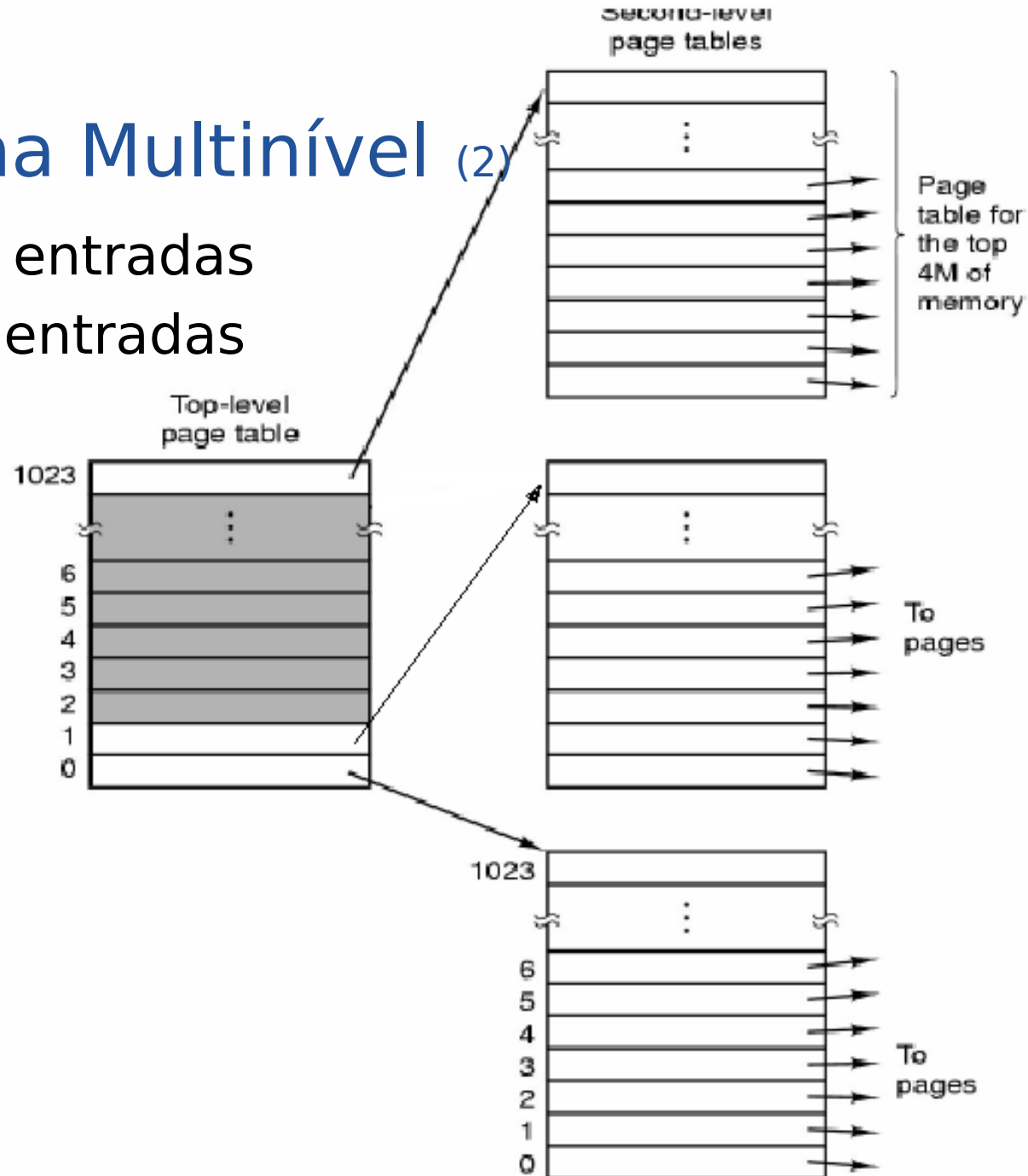
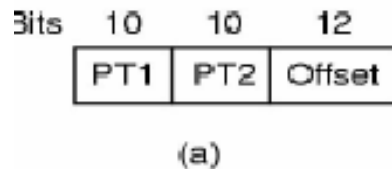


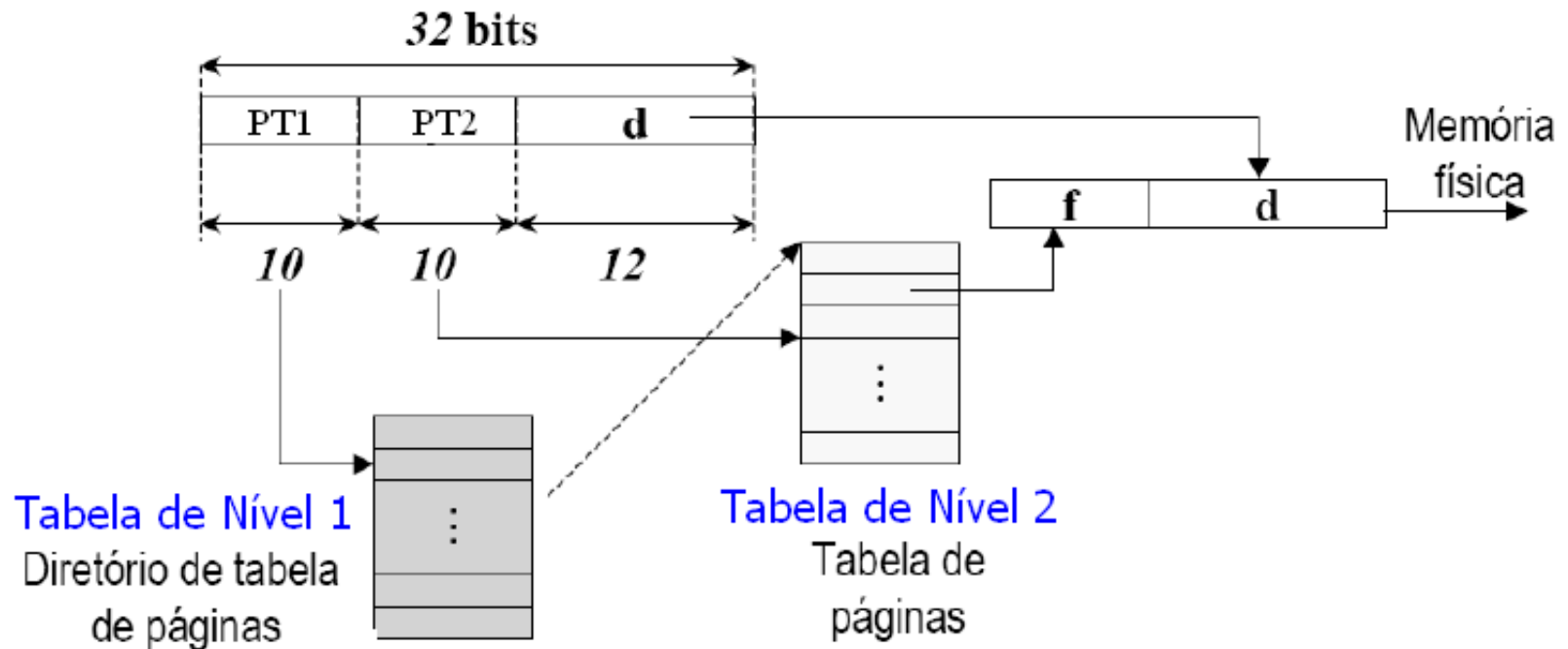
Tabela de Página Multinível ⁽³⁾

- No exemplo anterior:
 - Suponha que um processo utilize apenas 12 MB do seu espaço de endereços virtuais
 - 4MB da base da memória para código de programa
 - Outros 4 MB para dados
 - 4 MB do topo da memória para pilha
 - Portanto:
 - A **entrada 0** da tabela de nível 1 aponta para a tab. de páginas de nível 2 relativa ao código do programa
 - A entrada 1 da tabela de nível 1 aponta para a tab. de páginas de nível 2 relativa aos dados do processo
 - A entrada 1023 da tabela de nível 1 aponta para a tab. de páginas de nível 2 relativa à pilha do processo

Tabela de Página Multinível ⁽³⁾

- Quando um endereço virtual chega à MMU, ela primeiro extrai o campo PT1 e o utiliza como índice da tabela de páginas do nível 1
- A entrada da tab. de páginas de nível 1 aponta para a tabela de páginas do nível 2.
- Então PT2 é usado como índice nesta segunda tabela para localizar a entrada correspondente à página virtual
 - Esta entrada indicará em qual moldura física encontra-se o endereço a ser acessado
- No exemplo anterior:
 - Suponha que um processo utilize apenas 12 MB do seu espaço de endereços virtuais
 - A entrada 0 da tab. de nível 1 aponta para a tab. de páginas de nível 2 relativa ao código do programa

Tabela de Página Multinível (4)



- Considere o end. virtual $0x00403004$ (4206596_d)
 - Qual será o endereço físico correspondente?

Tabela de Página Multinível (5)

PT1	PT2	Deslocamento
0000000001	0000000011	0000 0000 0100

- PT1: Entrada 1 da tabela do 1º nível
 - 2º bloco de 4M (4M a 8M de memória virtual)
- PT2: Entrada 3 da tabela do 2º nível
 - Esta entrada indica em qual moldura encontra-se esta página
 - O endereço físico do primeiro byte dessa moldura é somado ao deslocamento
 - Supondo a página encontre-se na moldura 1 (4k a 8k-1), o endereço físico correspondente será $4096 + 4 = 4100$
- OU:

Nº da moldura	Deslocamento
0... 00001	0000 0000 0100

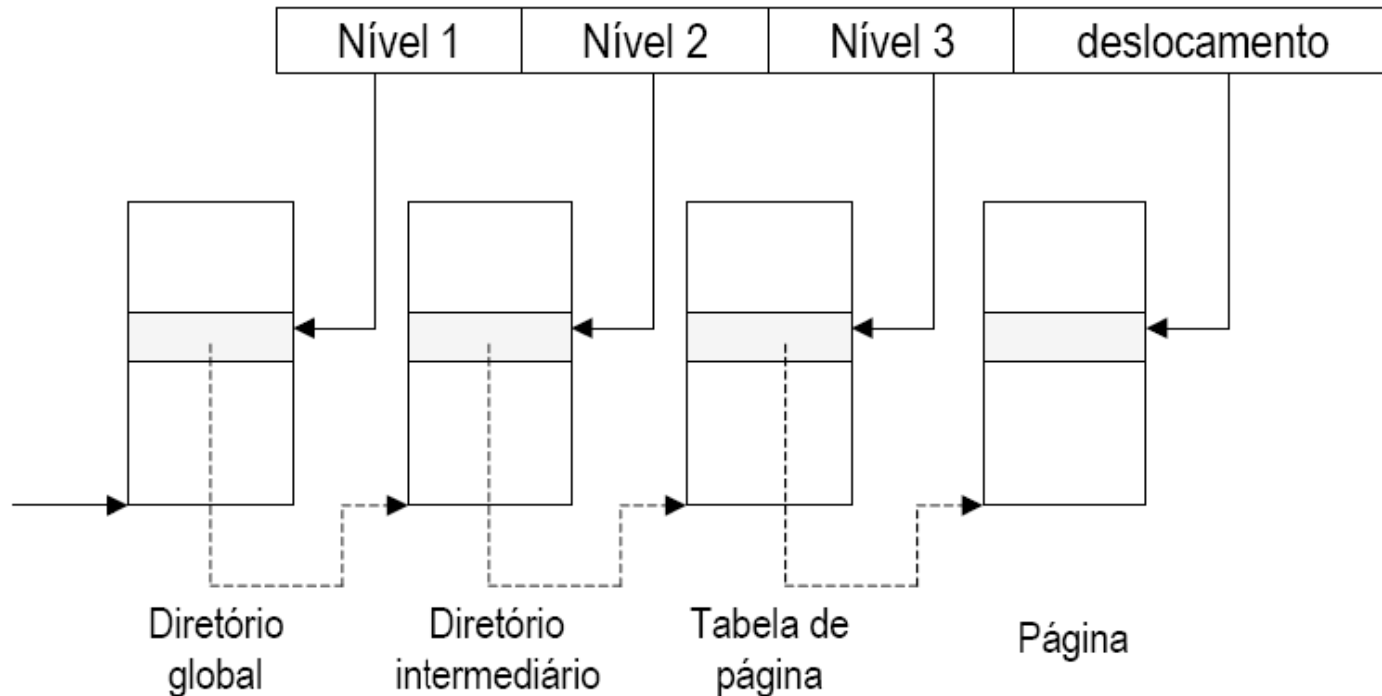
 = 4100_d

Tabela de Página Multinível ⁽⁶⁾

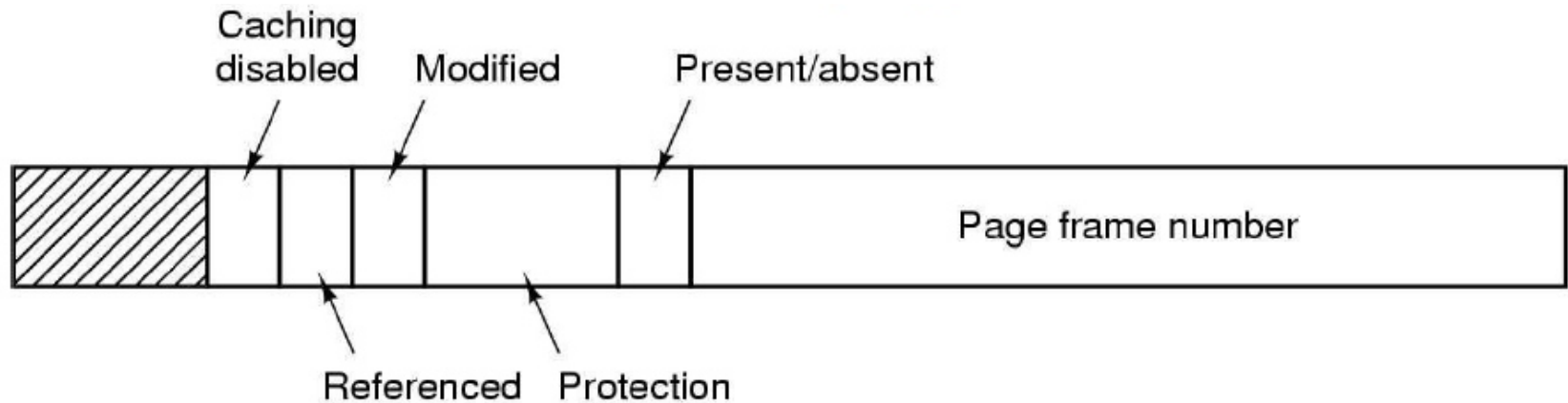
- Para entender as vantagens, considere o exemplo anterior (endereço virtual de 32 bits – página de 4kB)
 - Usando tabela de páginas tradicional:
 - 1 tab. de 2^{20} entradas (1 M entradas)
 - Usando tabela de páginas em 2 níveis
 - 4 tab. de 2^{10} entradas cada (1 K entradas)
 - Se cada entrada da tab. de páginas ocupa 16 bits
 - primeiro caso: $2^{20} \times 2^4 = 16$ Mbits p/ armazenar a tabela de pág.
 - segundo caso: $4 \times 2^{10} \times 2^4 = 64$ Kbits p/ armazenar a tabela de 2 níveis

Tabela de Página Multinível (7)

- Paginação a três níveis
 - Típico de arquiteturas de processadores de 64 bits



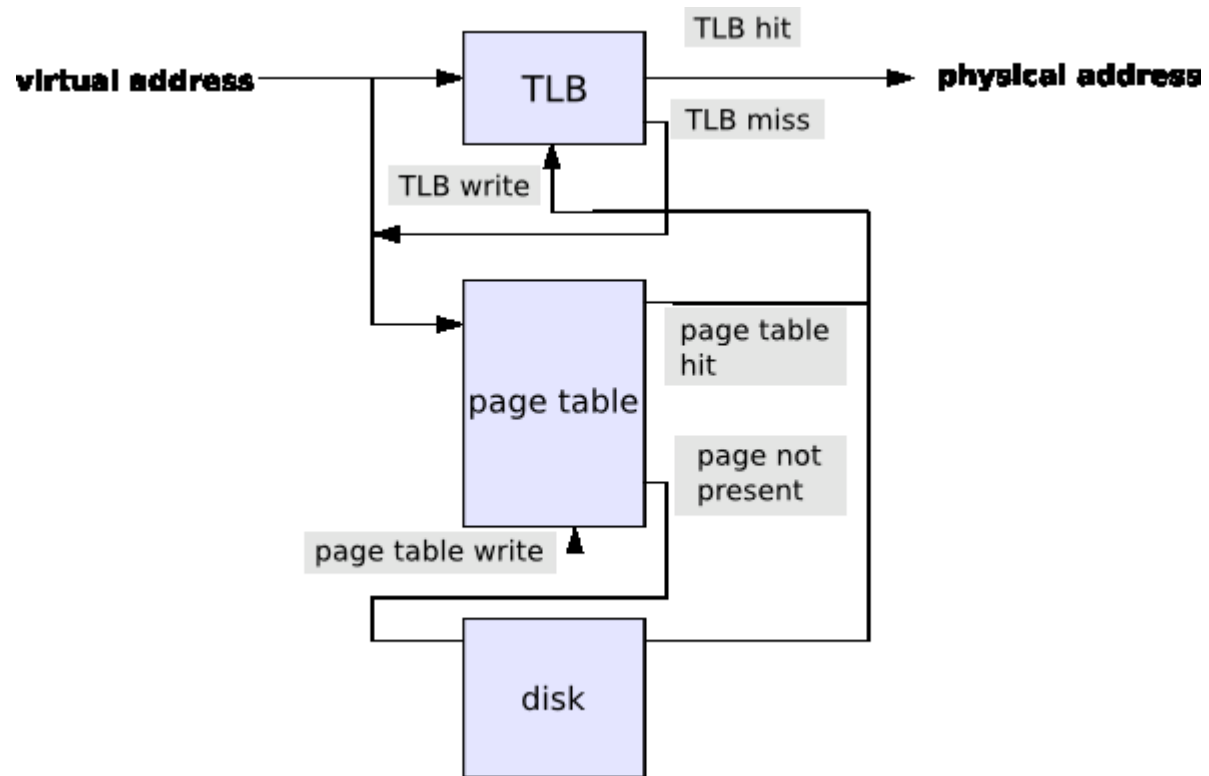
Registro Típico de uma Entrada na Tabela de Páginas



- Número da moldura
- Presente/ausente: diz se página está ou não mapeada em endereço físico
- Proteção: bits de controle de acesso à página (rwx)
- Modificada: indica se página foi alterada
- Referenciada: indica se página foi lida
- Desabilita cache

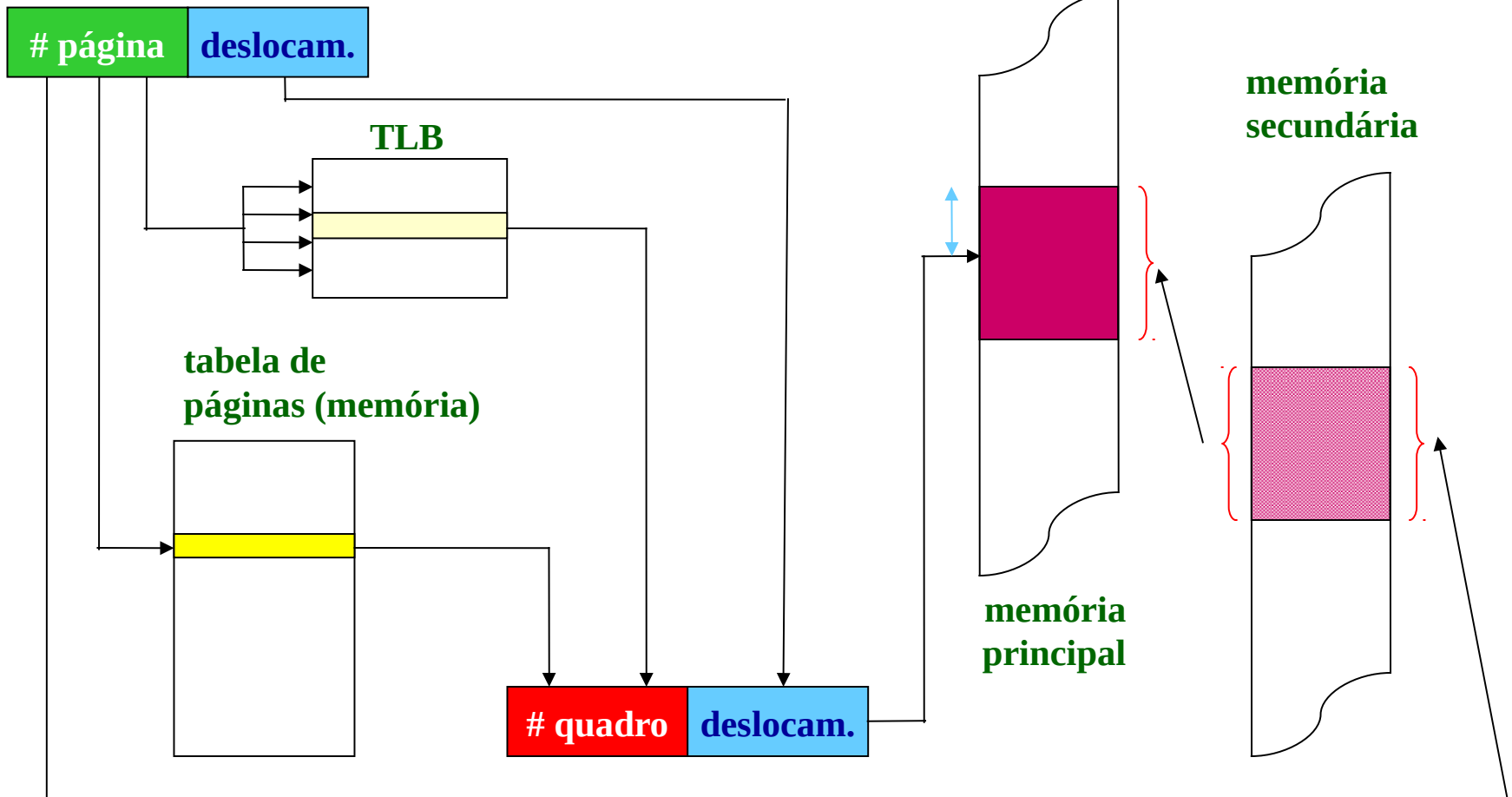
TLB – *Translation Lookaside Buffer* ⁽¹⁾

- Como diminuir o número de referências à MP introduzido pelo mecanismo de paginação?
- Os programas tendem a fazer um grande número de referências a um mesmo pequeno conjunto de páginas virtuais
 - Princípio da localidade temporal e espacial
- Solução: equipar a MMU com uma TLB
 - Também chamada de Memória Associativa
 - Dispositivo de hardware implementado com um reduzido número de entradas
 - Contém algumas entradas (linhas) da tabela de páginas do processo em execução



TLB – Translation Lookaside buffer (2)

endereço virtual



TLB – *Translation Lookaside buffer* (3)

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

- Exemplo de TLB
 - Loop acessando pag. 19, 20, 21
 - Dados principais: pag. 129, 130, 141
 - Pilha: 860, 861

HIT Ratio (Taxa de Sucesso) ⁽¹⁾

- Razão de referências à memória que podem ser satisfeitas a partir da TLB
- \uparrow Hit Ratio $\Rightarrow \uparrow$ performance
- Tempo de acesso com *HIT* (sucesso) à memória via TLB

$$T_{\text{Hit}} = T_{\text{TLB}} + T_{\text{MEM}}$$

- Tempo de acesso com *MISS* (insucesso) à memória via TLB

$$T_{\text{Miss}} = T_{\text{TLB}} + T_{\text{MEM}} + T_{\text{MEM}}$$

- Tempo médio de acesso = $hr \cdot T_{\text{Hit}} + (1-hr) \cdot T_{\text{Miss}}$
 - Onde hr é o Hit Ratio

HIT Ratio (Taxa de Sucesso) (2)

- Suponha: $T_{\text{Hit}} = 20\text{ns}$; $T_{\text{Miss}} = 100\text{ ns}$; H.R. = 90%
 $0.9 \times 20 + 0,1 \times 100 = 28\text{ns}$

Tempo médio de acesso=

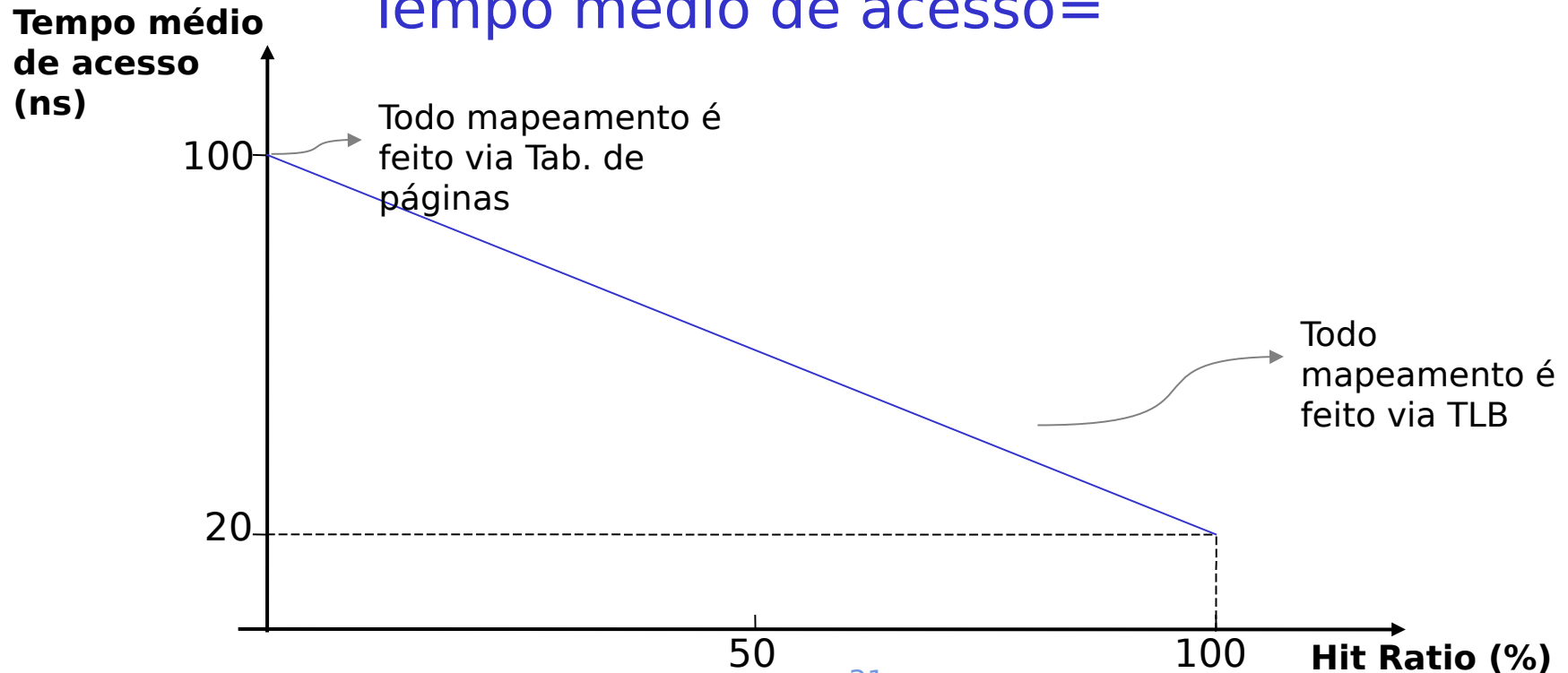


Tabela de páginas invertida ⁽¹⁾

- Espaço de endereçamento virtual pode ser exageradamente grande em máquinas de **64 bits**.
 - Páginas de 4KB
 - 2^{52} entradas na tabela
 - Se cada entrada ocupa 8 B => tabela de ~30.000.000 GB
- O armazenamento da tabela torna-se viável se a mesma for **invertida**, isto é, ter o tamanho da quantidade de molduras (memória real) e não da quantidade de páginas (memória virtual)
 - Se memória real é de 256 Mbytes , e páginas de 4 KB:
 - Tem-se 65536 entradas

Tabela de páginas invertida (2)

- Uma entrada por moldura de memória real
- Cada entrada na tabela informa
 - Par: (PID, # página virtual) alocado naquela moldura
- Entretanto
 - Tradução de virtual/físico mais complicada
 - Quando o processo n endereça a página p
 - p não serve de índice da tabela
 - **Toda a tabela deve ser pesquisada** em busca de uma entrada (p,n)
- Solução muito lenta
 - A busca é feita para toda referência à memória

Tabela de páginas invertida (3)

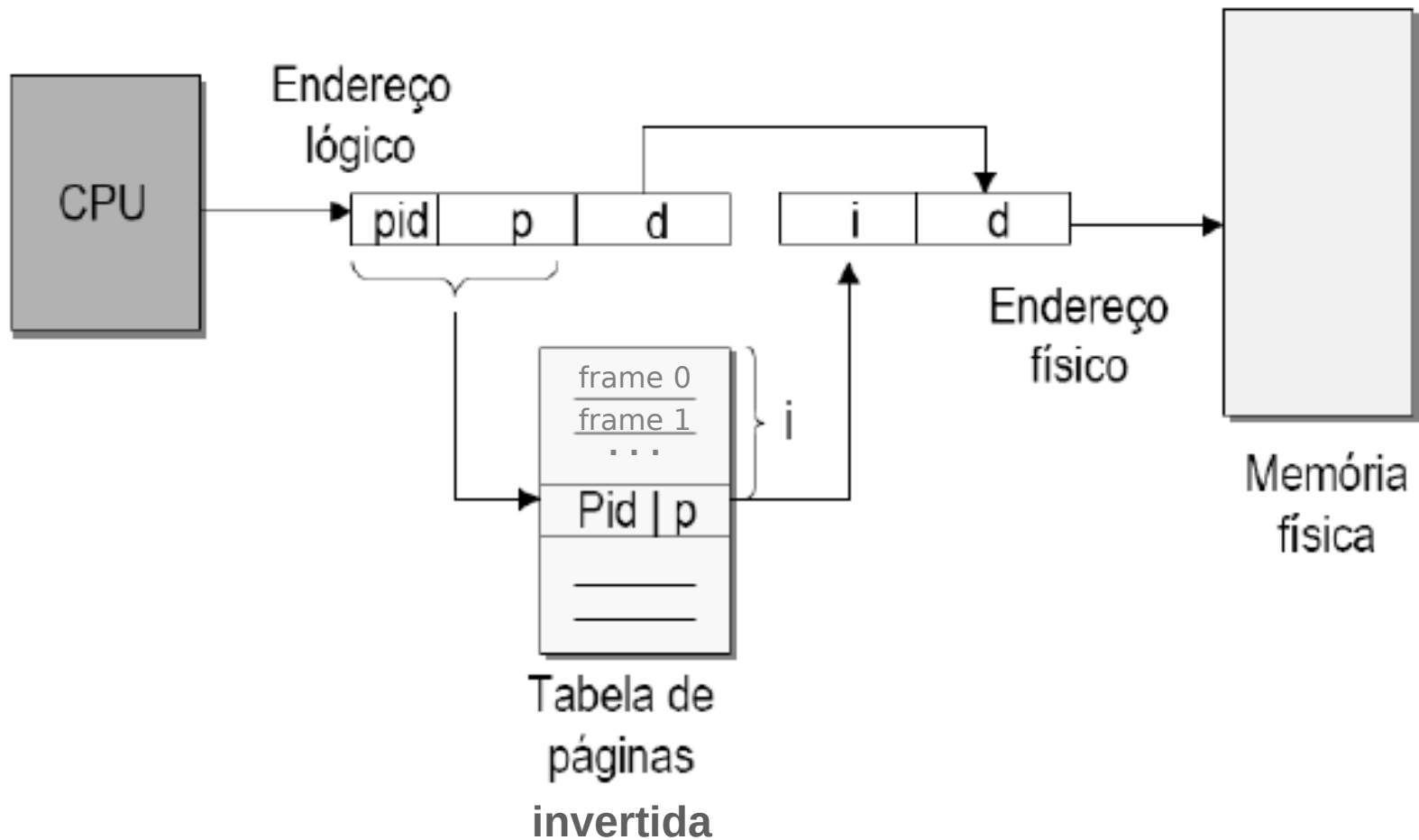
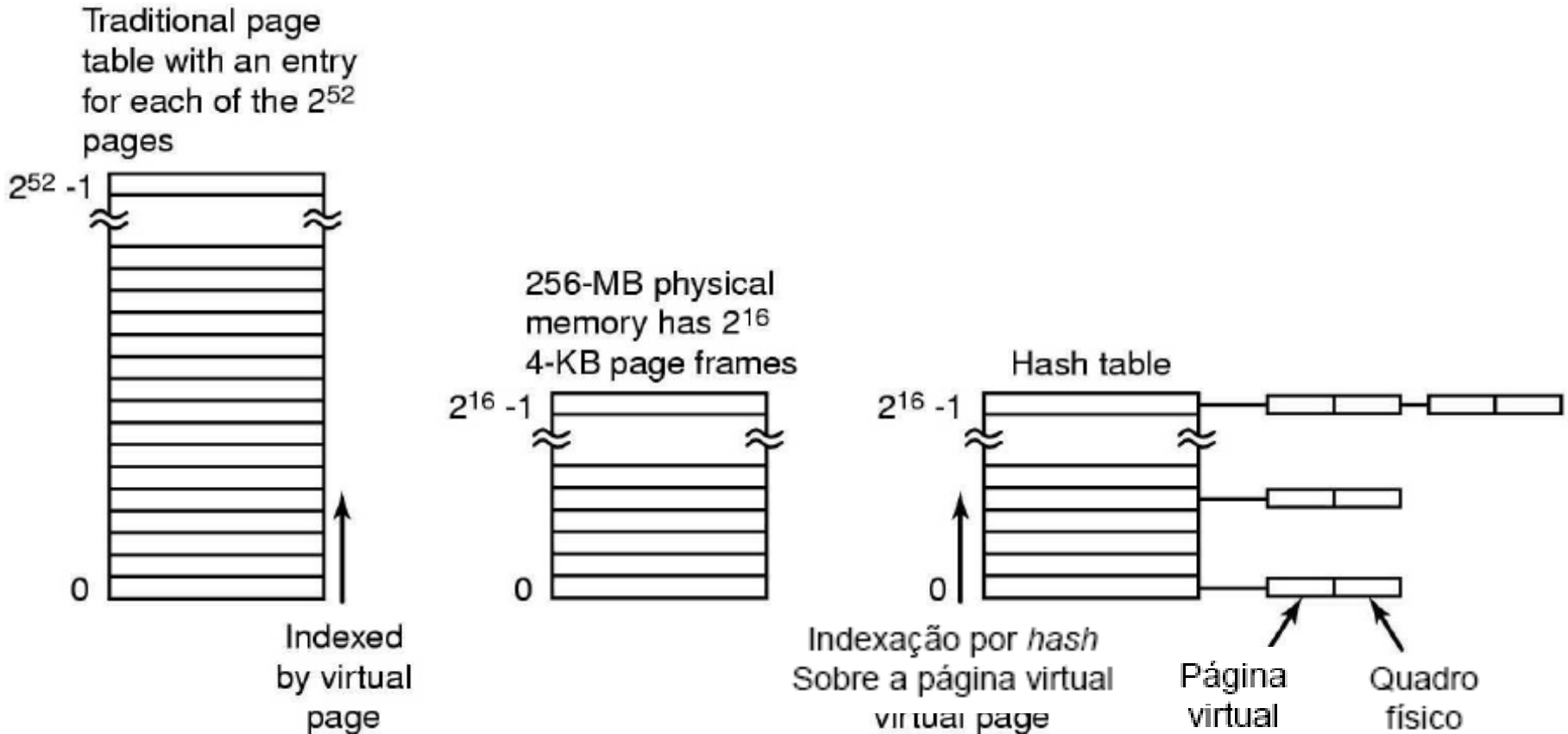


Tabela de páginas invertida (4)

- Aceleração pode ser obtida
 - TLB para páginas mais referenciadas
 - Indexar a tabela por *hash*
 - Uma função hash que recebe o número da página e retorna um entre N valores possíveis, onde N é a quantidade de molduras (memória instalada).
 - Páginas com mesmo *hash* serão encadeadas em uma lista
 - Cada entrada da tabela armazena um par (página/quadro)

Exemplo de tabela de páginas invertida



Comparação de uma *page table* tradicional com uma *page table invertida*