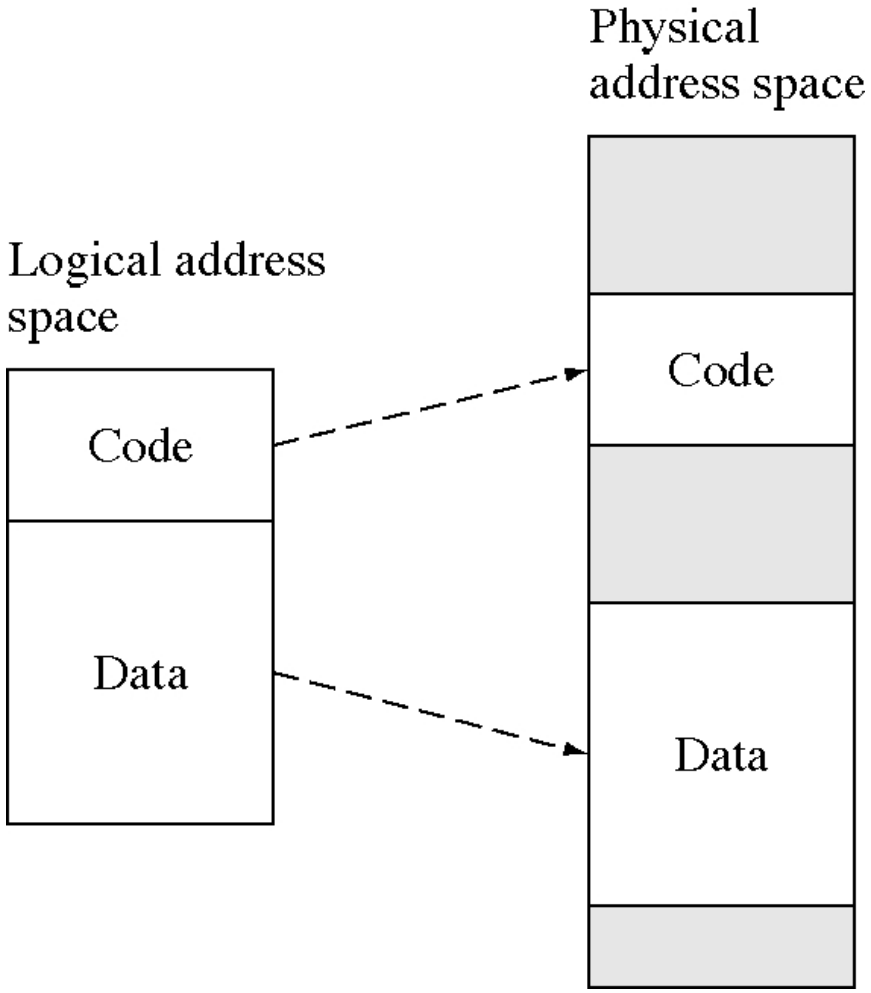
Gerência de Memória

- Paginação
- Segmentação
- Segmentação com Paginação

Alocação Contígua

- Sem mapeamento de memória, programas requerem memória fisicamente contígua
- Grandes blocos significa grandes fragmentos
  - e memória desperdiçada
- É preciso hardware para mapeamento da memória para resolver este problema
  - segmentos
  - páginas

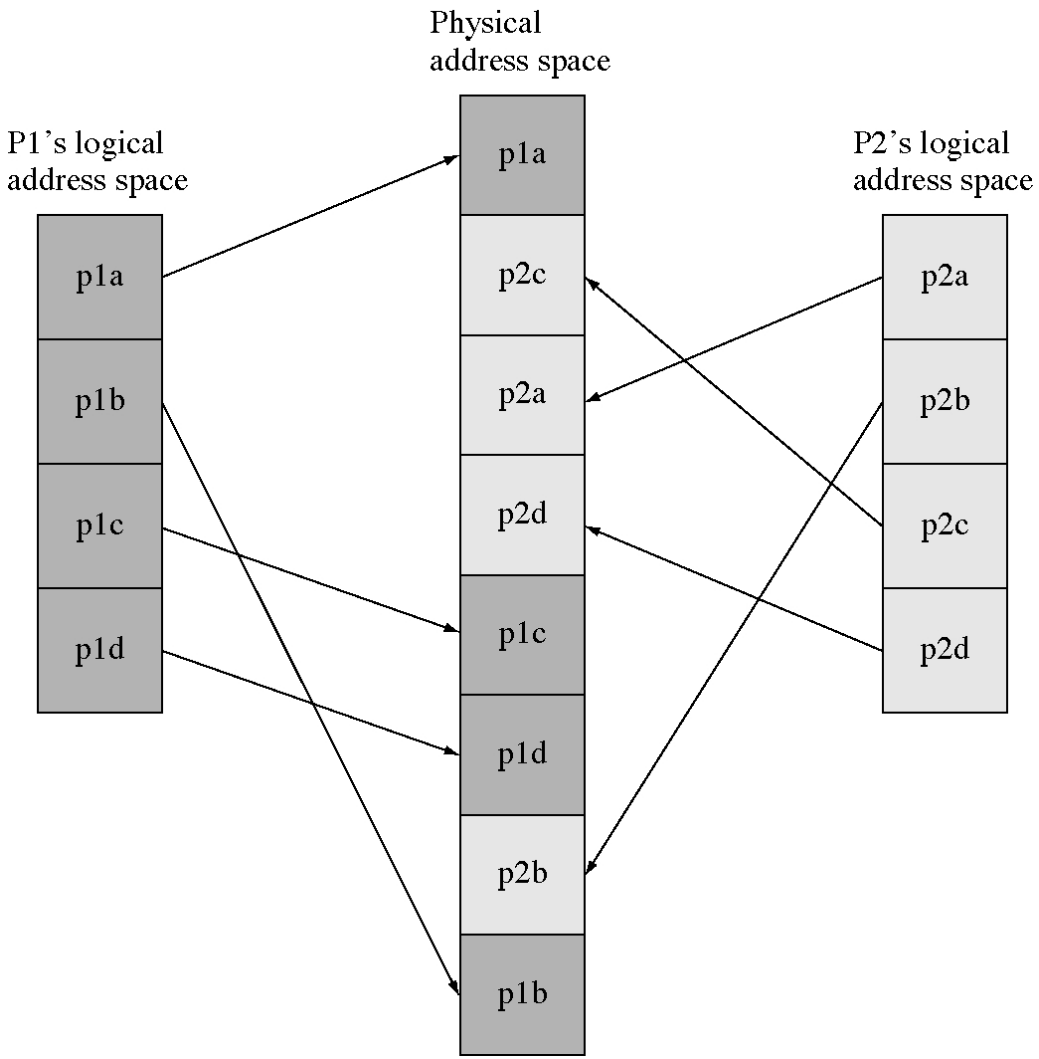
Espaço de código e dados separados



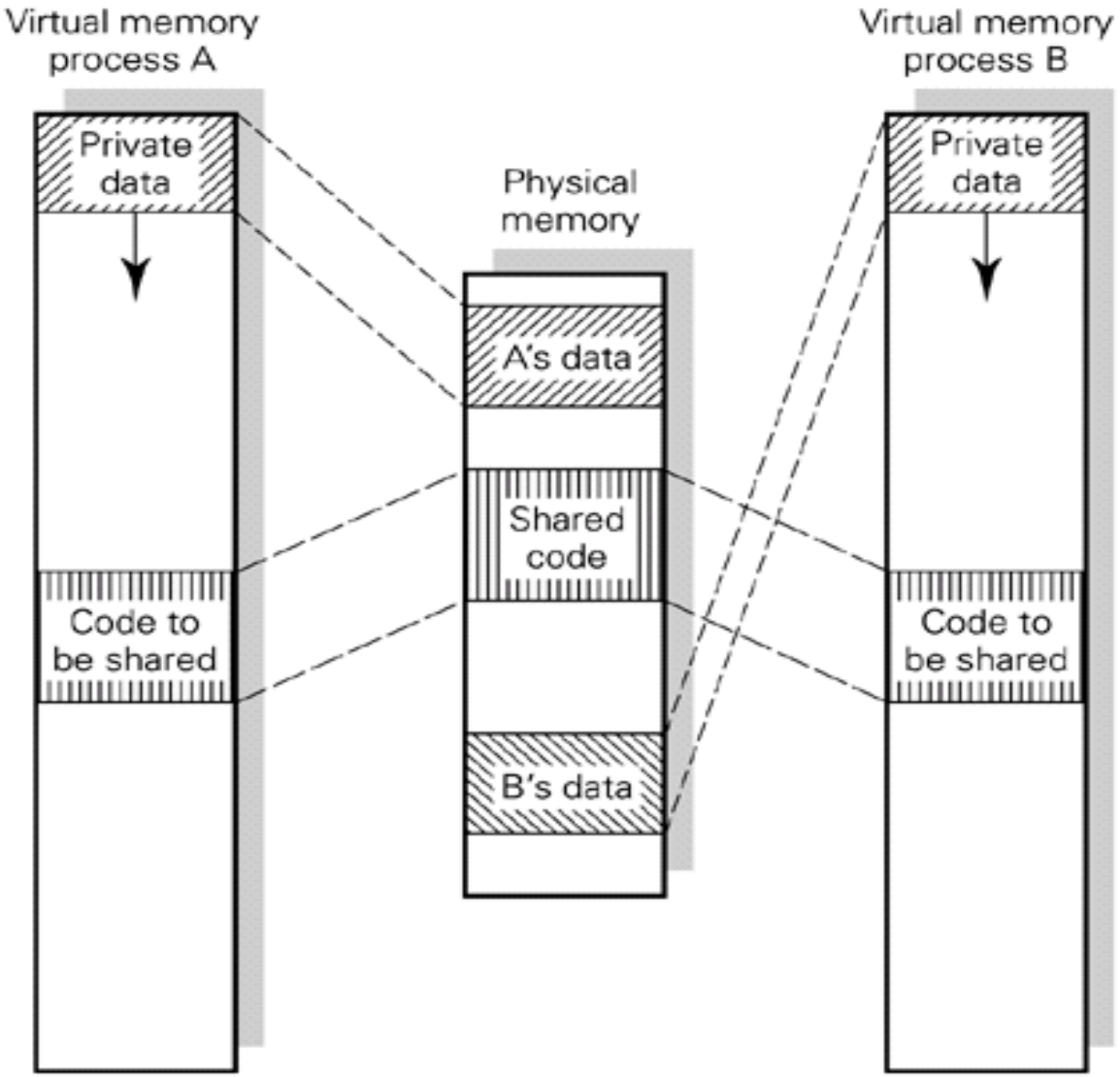
Segmentação

- Divide o espaço de endereçamento lógico em segmentos (pedaços de memória de tamanho variável)
- Cada segmento tem um registrador de base e de limite
  - E assim segmentos não precisam ser contíguos no espaço de endereçamento físico
  - Mas o espaço de endereçamento lógico ainda é contíguo
- DEC PDP11
  - oito segmentos
  - Até 8K bytes por segmento

Espaços de endereçamento segmentado

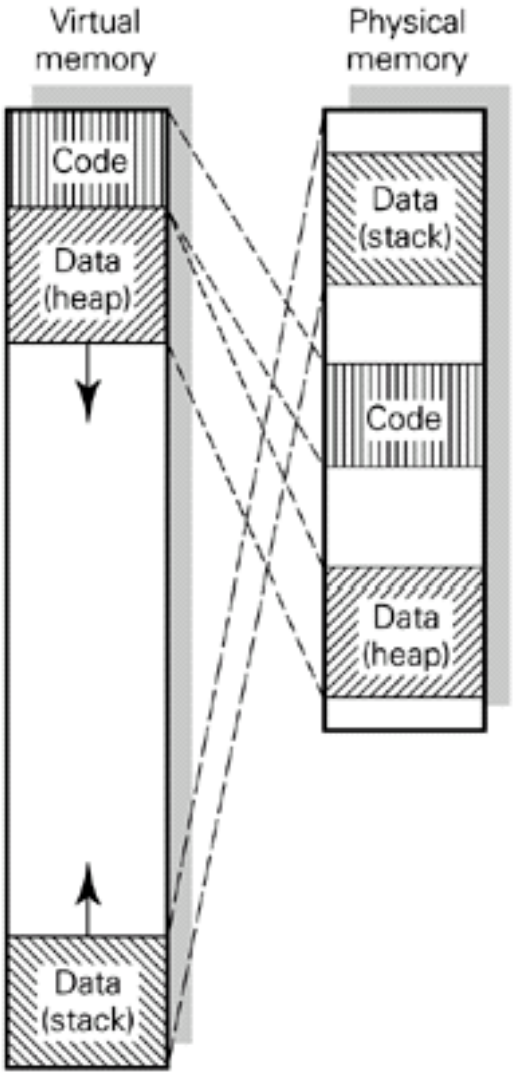


Espaços de endereçamento segmentado



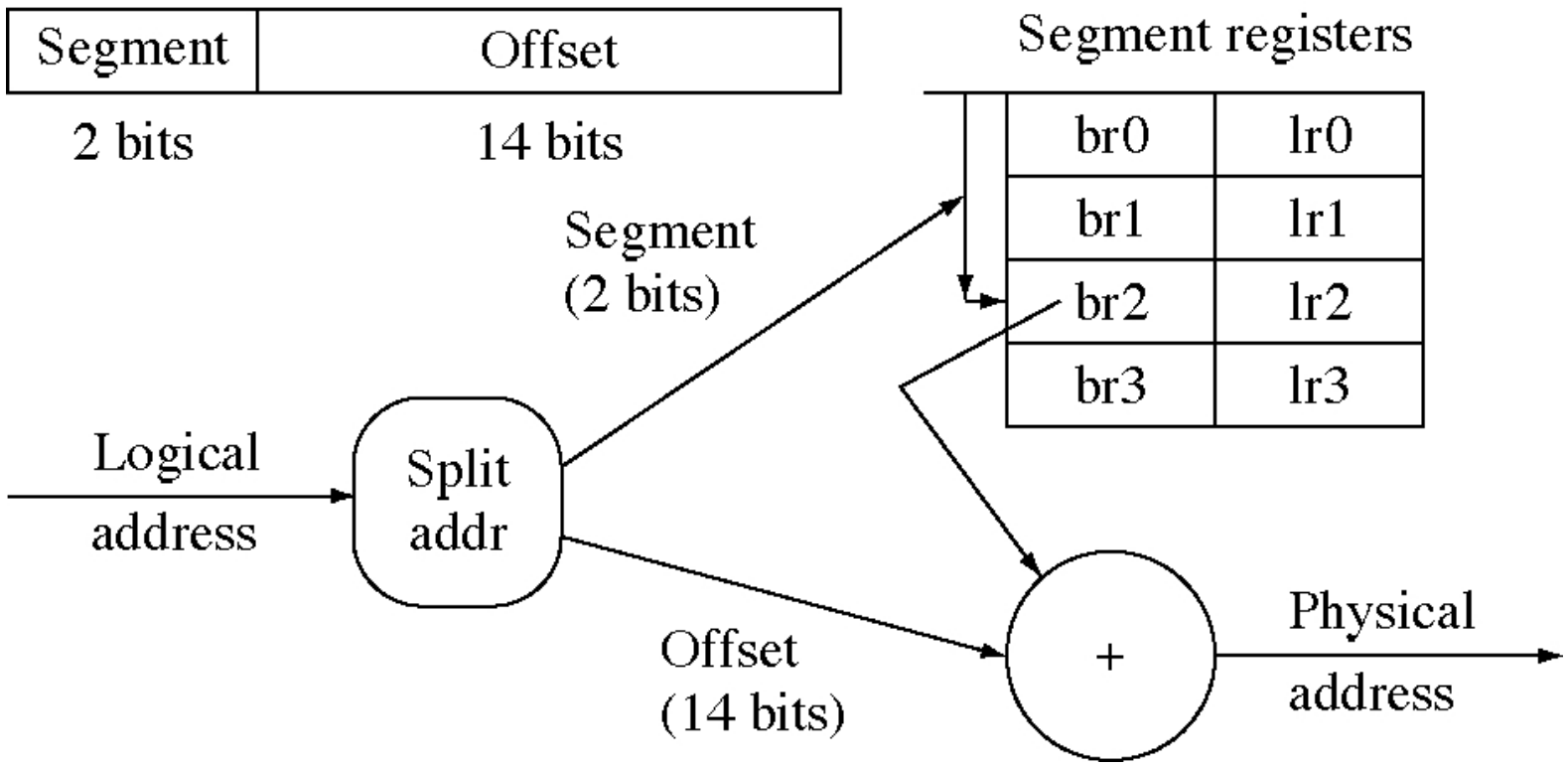
Espaços de endereçamento segmentado

The user's program is shown here starting from address zero. After the code comes a growing data area (the heap). The stack is in a separate data area and grows towards the heap.

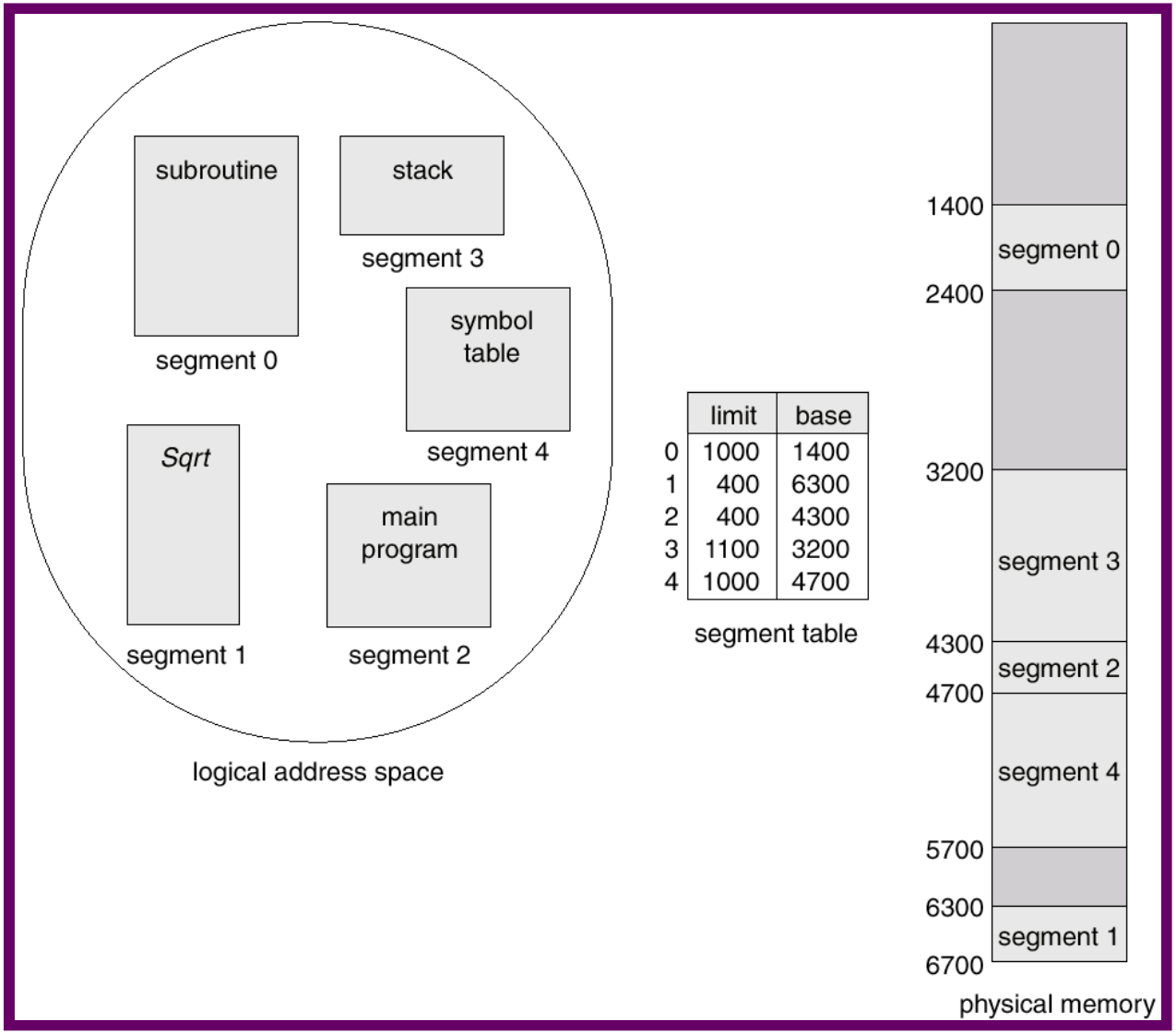


The code area and two data areas are shown separately relocated. This would require three segments per process and corresponding relocation and protection hardware.

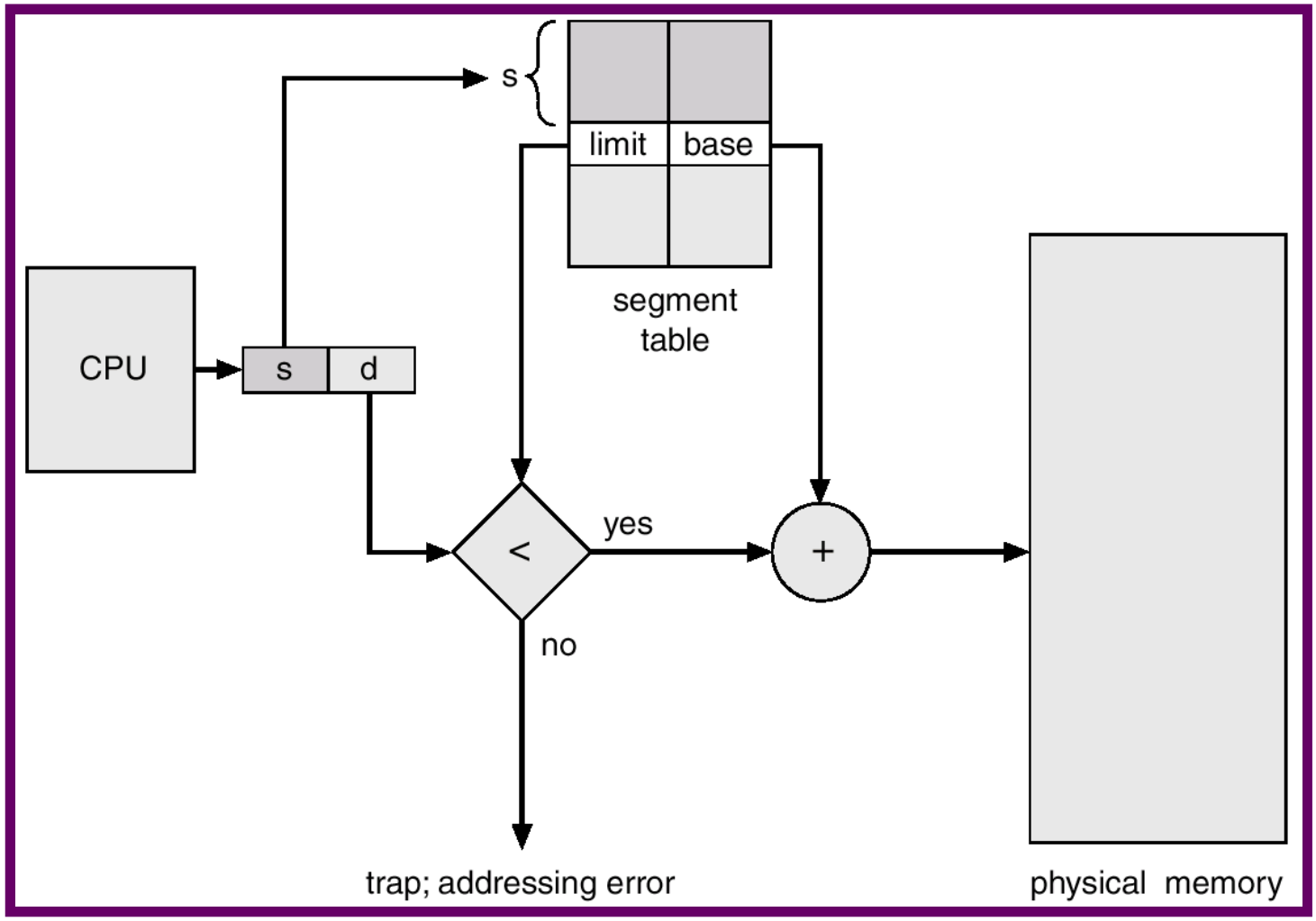
Segmentação: mapeamento da memória



Exemplo de Segmentação



Hardware de Segmentação



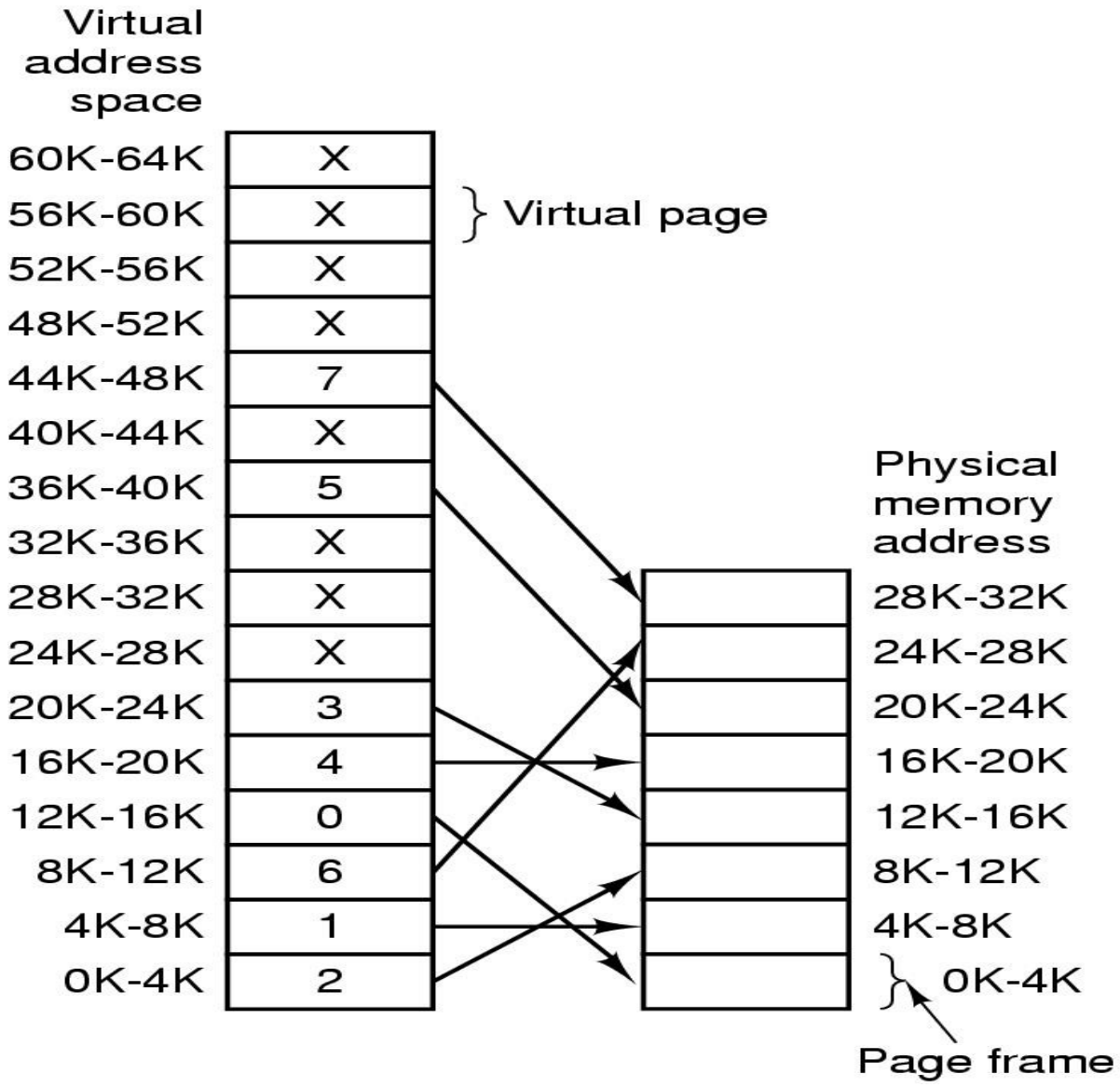
De segmentos para páginas

- Grandes segmentos não ajudam o problema da fragmentação
  - Assim precisamos pequenos segmentos
- Pequenos segmentos estão usualmente cheios
  - Assim não precisamos um registrador de tamanho
  - Apenas fazemos eles todos do mesmo tamanho
- Segmentos de tamanhos Identicos são chamados *páginas*
- Usamos tabelas de páginas no lugar de tabelas de segmentos
  - registrador de base sem registrador de limite

Paginação

- Espaço de endereço de um processo pode ser não contíguo; ao processo é alocado memória física sempre que disponível.
- Divide memória física em blocos de tamanho fixo chamados de **frames** (tamanho é potência de 2, entre 512 bytes- 8192 bytes).
- Divide memória lógica em blocos de mesmo tamanho chamados de **páginas**.
- Mantém informação sobre todos frames livres
- Para executar um programa de tamanho *n* páginas, necessário encontrar *n* frames livres e carregar o programa.
- Prepara uma tabela de páginas para traduzir endereços lógicos em físicos.
- Fragmentação Interna.

Arquitetura



Tradução de endereços

- Endereço gerado pela UCP é dividido em:
  - *Número da Página (p)* – usado como índice para a *tabela de páginas* que contém endereço base de cada página na memória física.
  - *Deslocamento na Página (d)* – combinado com o endereço base define o endereço de memória física que é enviado para a unidade de memória.

Arquitetura

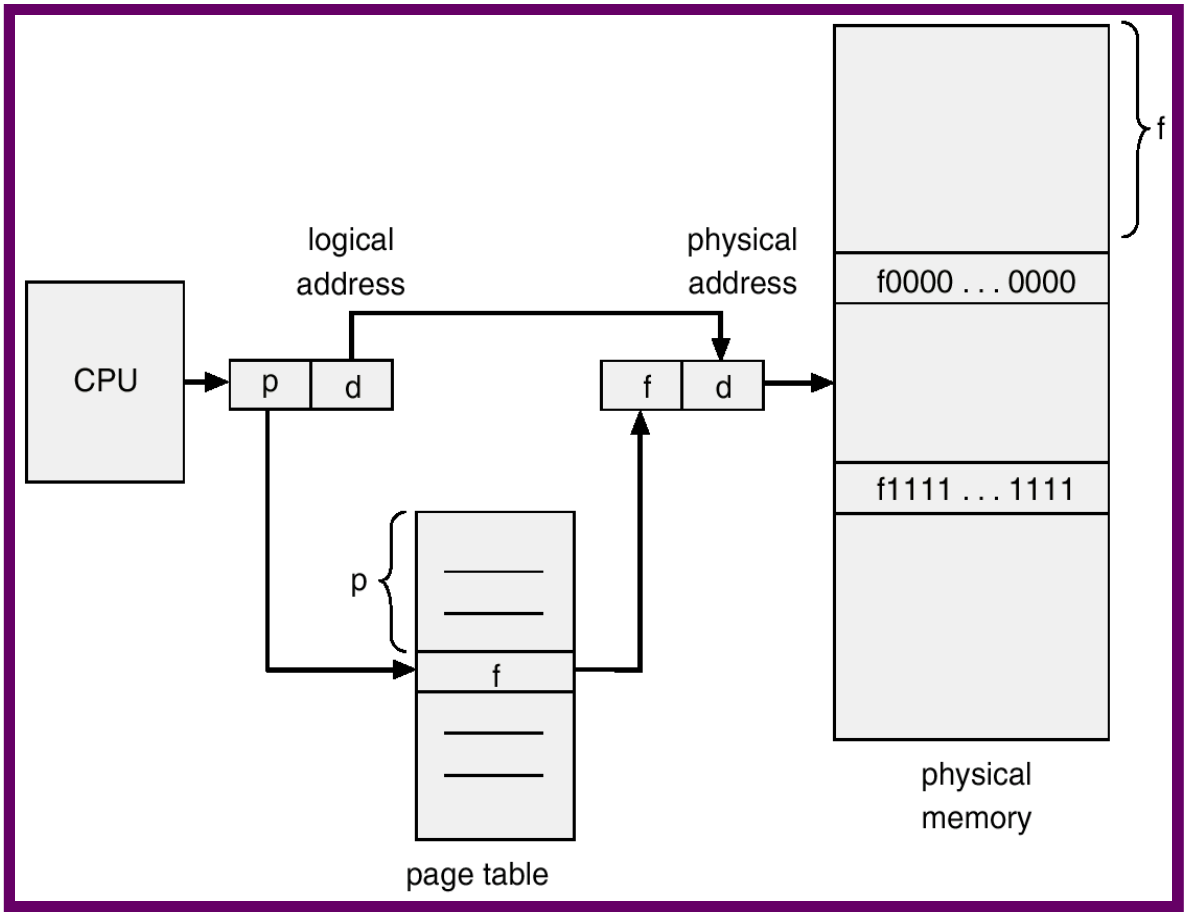
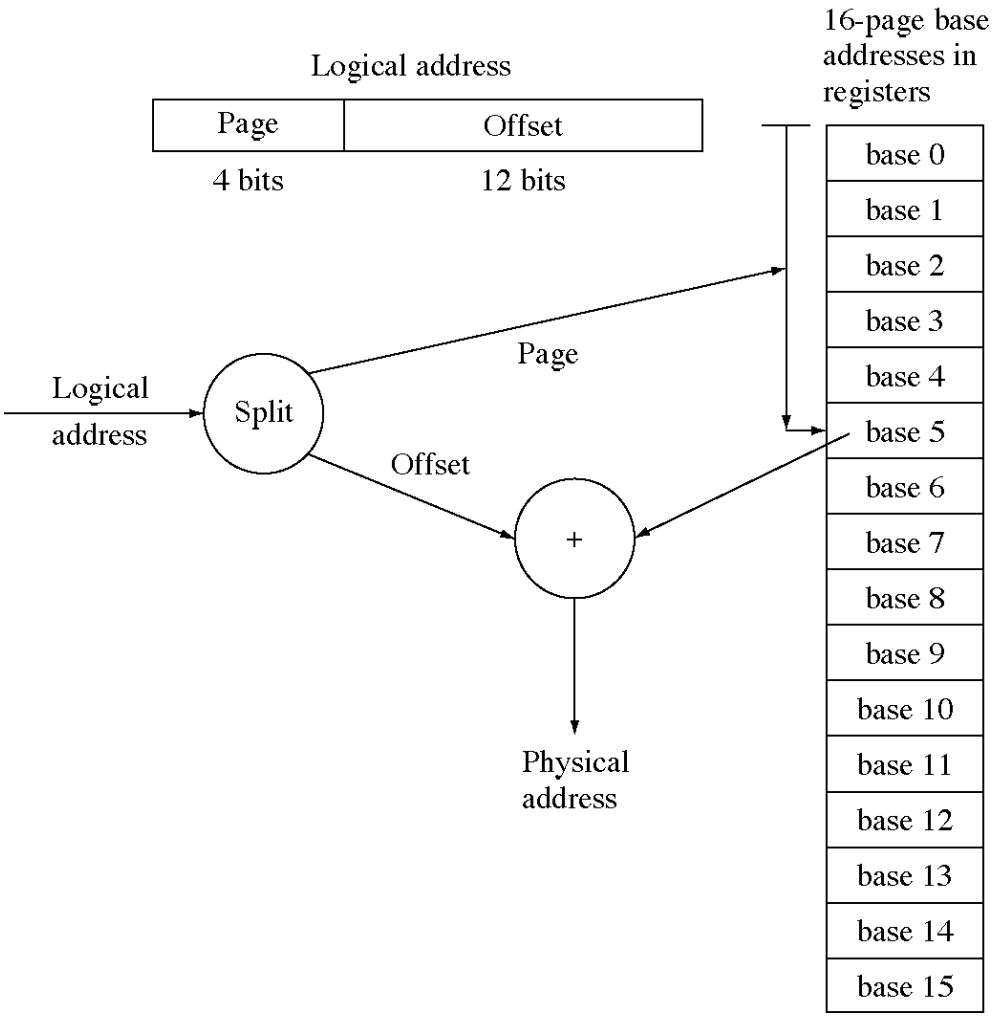


Tabela de páginas : registradores



Problemas com TP em registradores

- Limite no número de páginas
- Tempo para salvar e carregar registradores em chaveamento de contexto
- Custo de registradores de hardware
- *Solução*: colocar a tabela de páginas em memória e ter um registrador apontando para ela

Tabela de páginas: memória

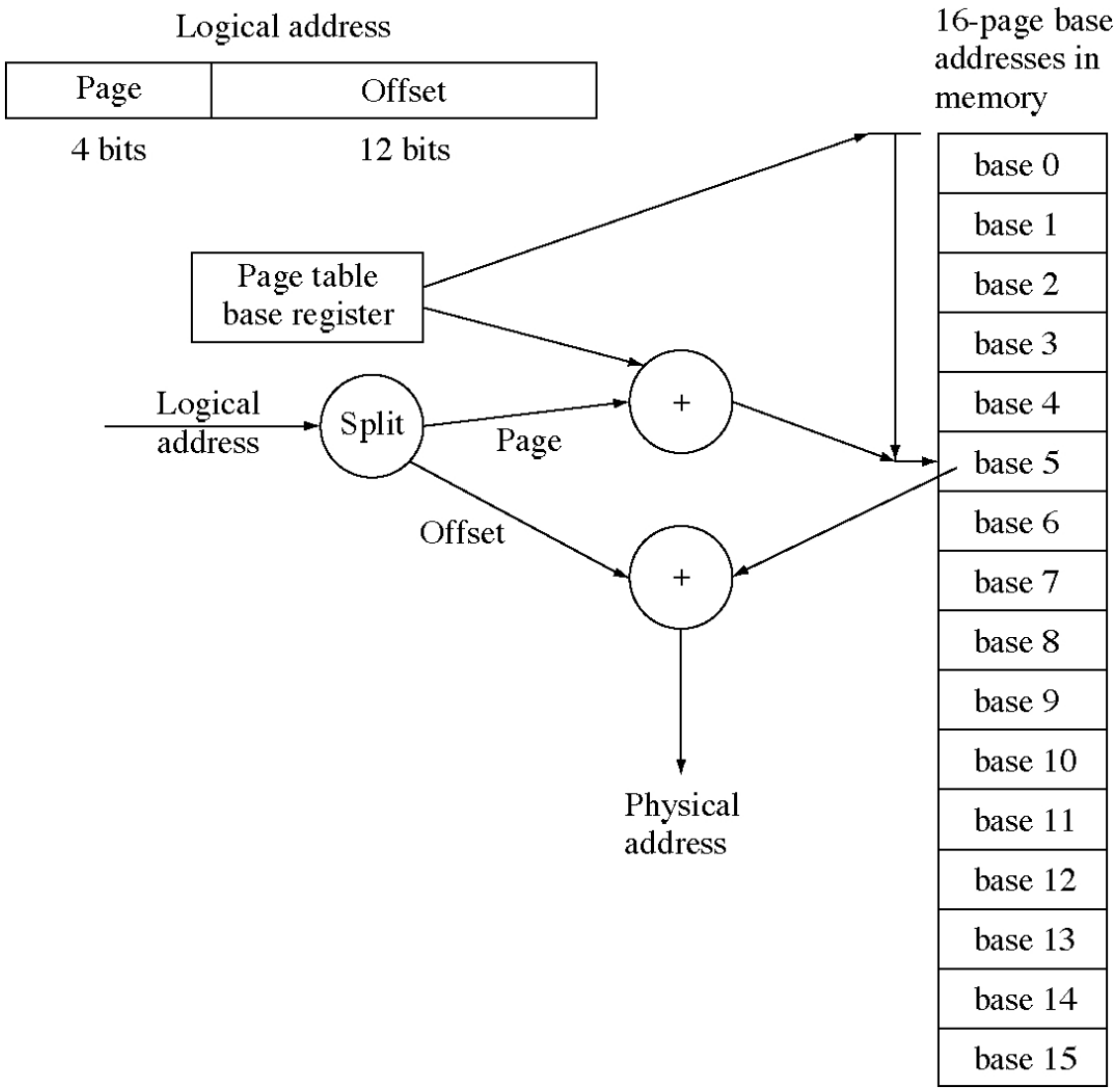
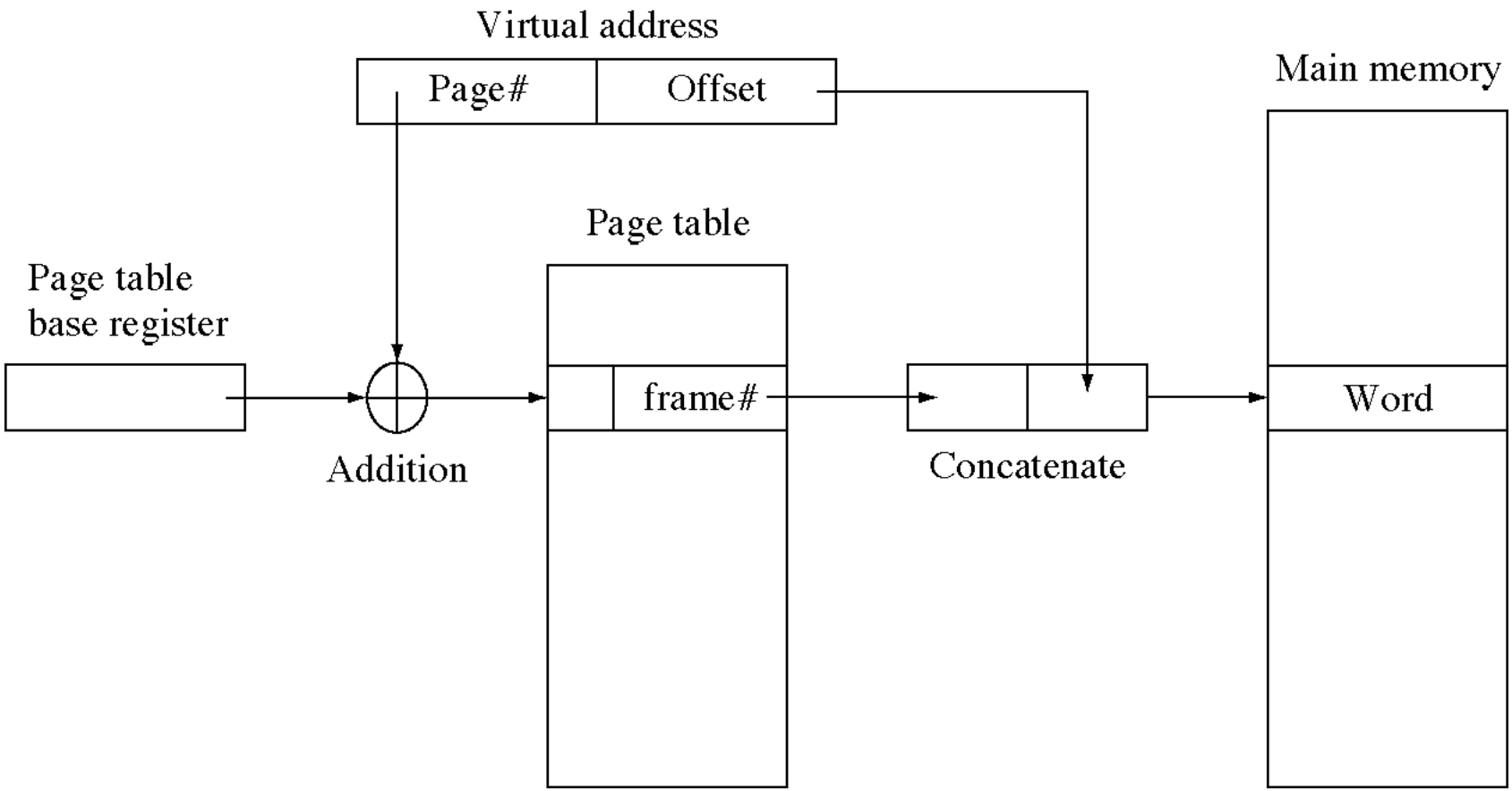
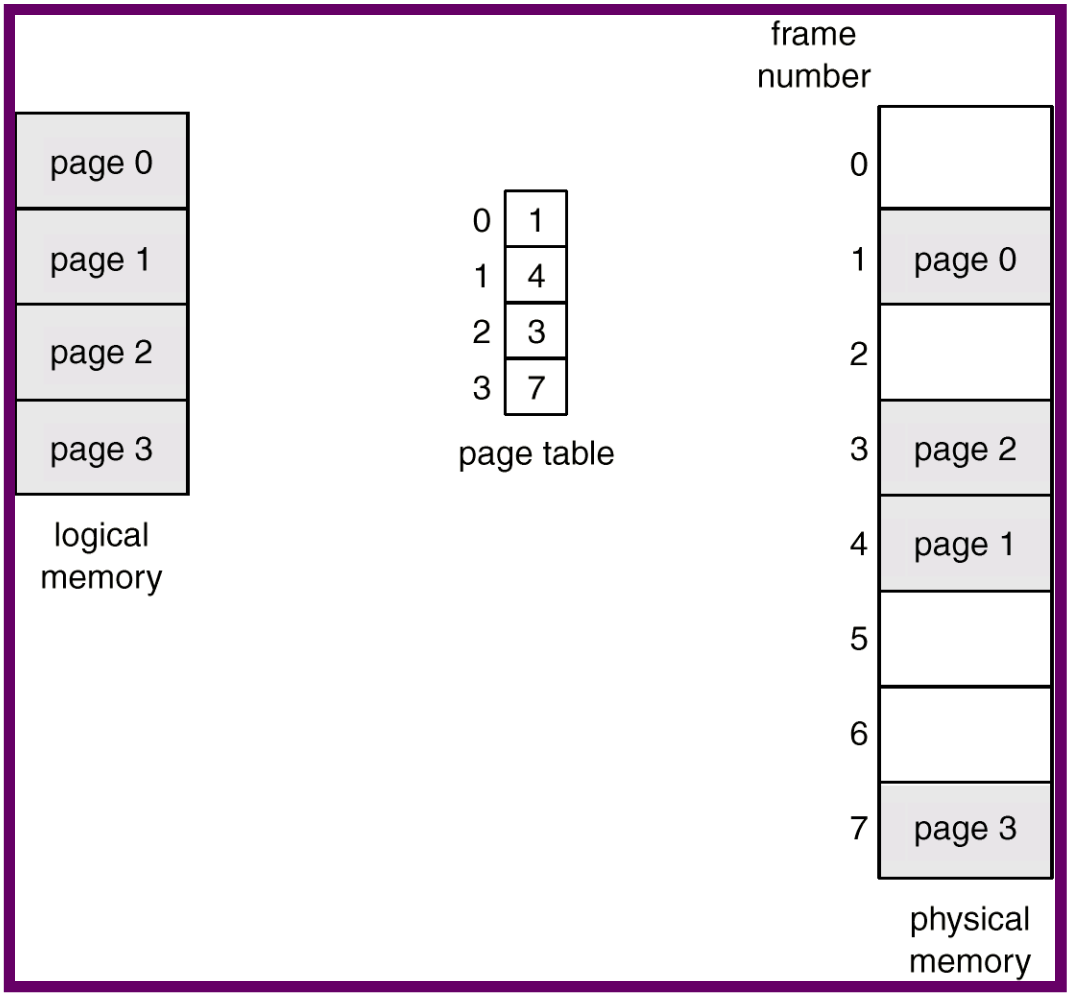


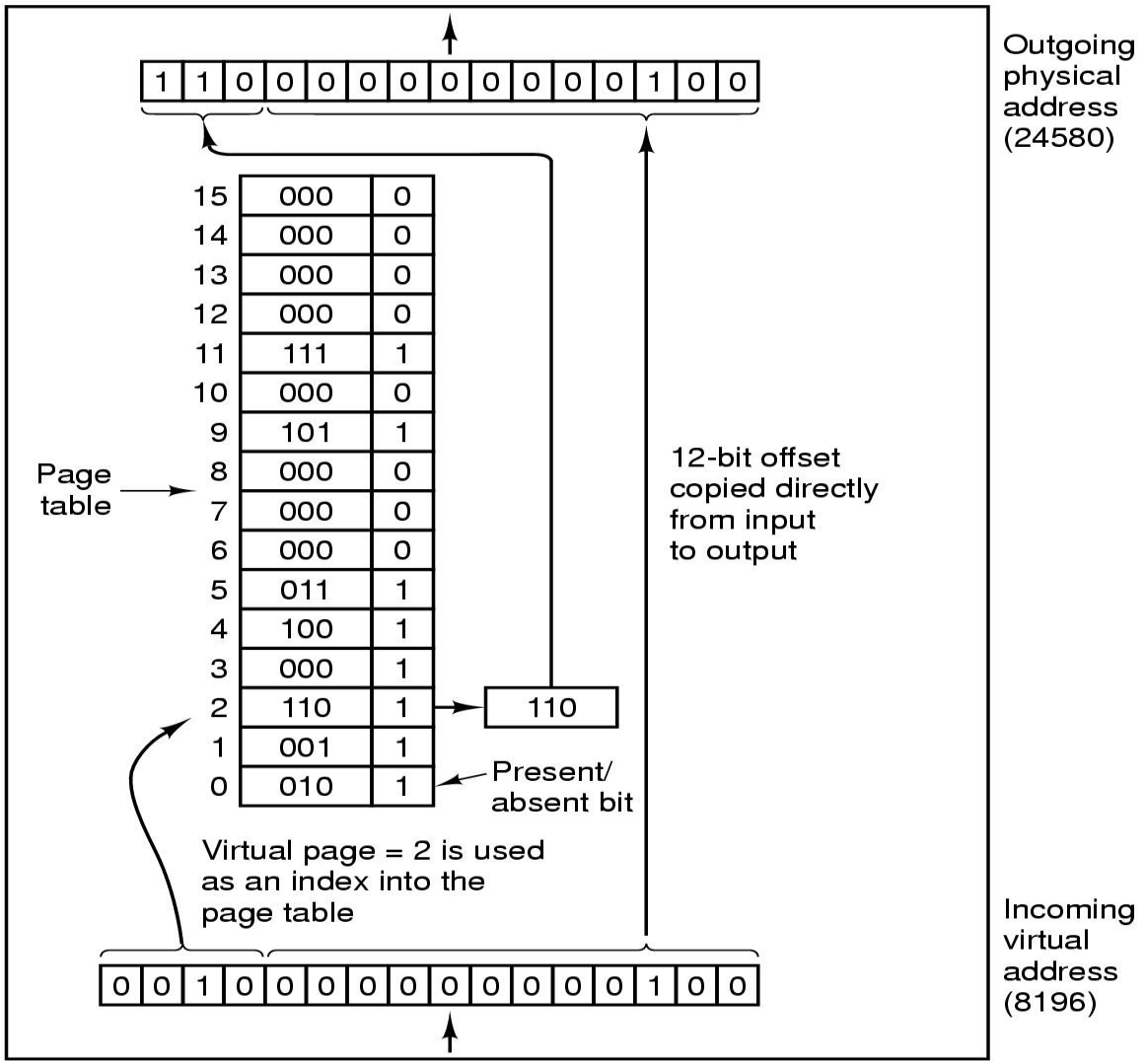
Tabela de páginas: mapeamento



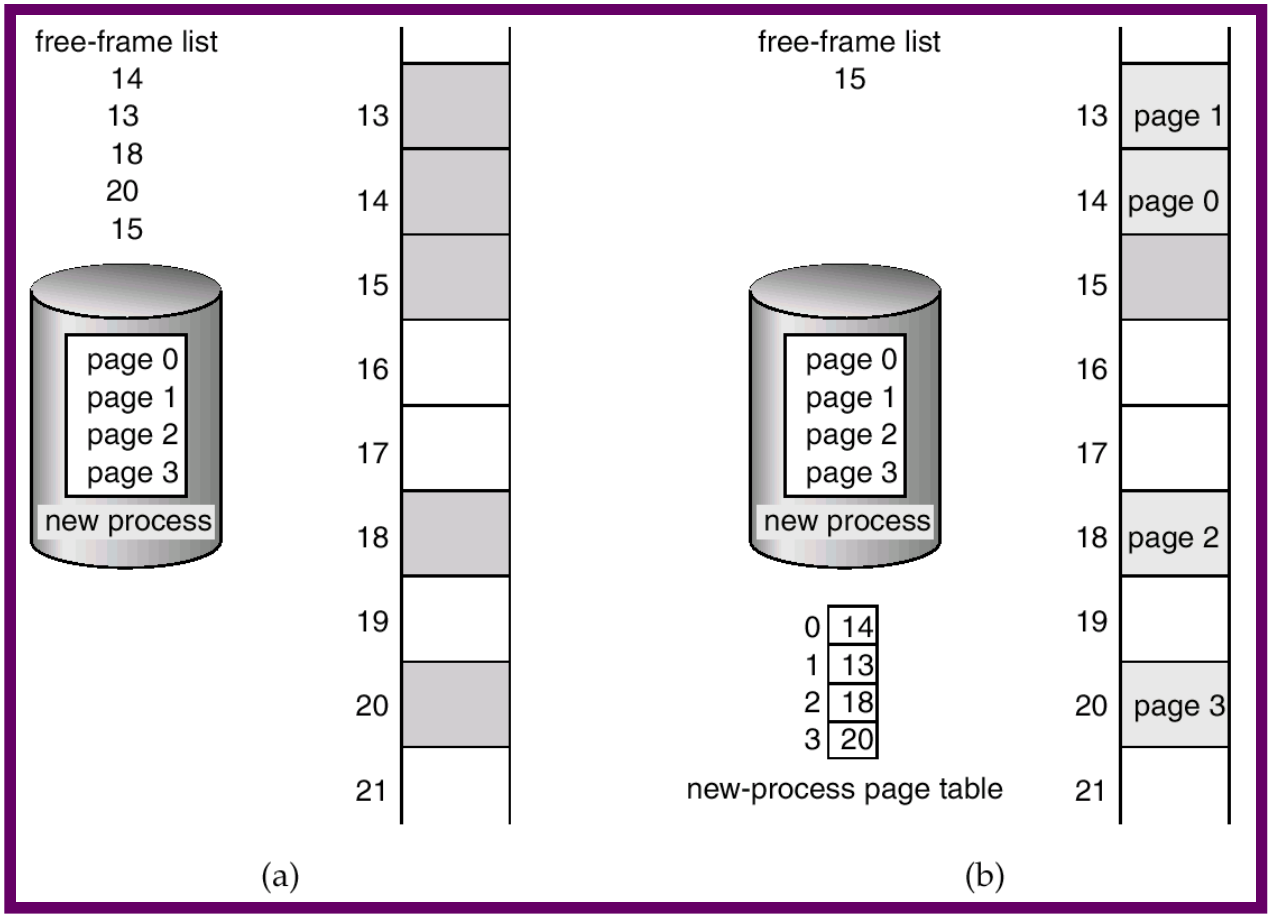
Exemplo


frame
number
page 0
page 1
page 2
page 3
logical
memory
0 1
1 4
2 3
3 7
page table
0
1 page 0
2
3 page 2
4 page 1
5
6
7 page 3
physical
memory

Exemplo


Outgoing physical address (24580)
1 1 0 0 0 0 0 0 0 0 0 1 0 0
15 000 0
14 000 0
13 000 0
12 000 0
11 111 1
10 000 0
9 101 1
8 000 0
7 000 0
6 000 0
5 011 1
4 100 1
3 000 1
2 110 1
1 001 1
0 010 1
Page table
110
Present/ absent bit
12-bit offset copied directly from input to output
Virtual page = 2 is used as an index into the page table
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
Incoming virtual address (8196)

Frames livres

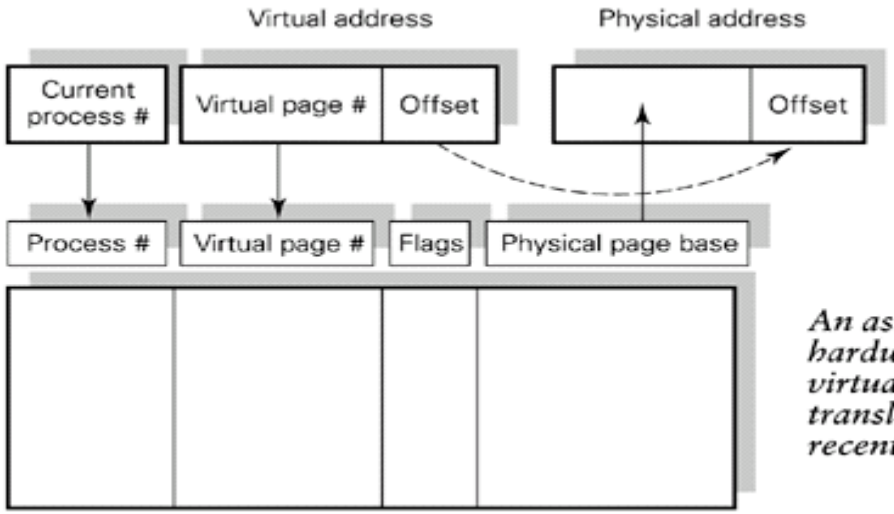


Antes da alocação

Depois da alocação

Problemas com TP em memória

- Neste esquema todo acesso dados/instrução requer dois acessos a memória. Um para a tabela de páginas e um para os dados/instrução
  - A utilização da memória dobra
  - e a velocidade do programa cai pela metade
- *Solução*: caching entradas da TP
  - Chamado de *translation lookaside buffer*
  - ou TLB



An associatively addressed hardware table containing virtual to physical address translations for the most recently addressed pages.

Memória Associativa

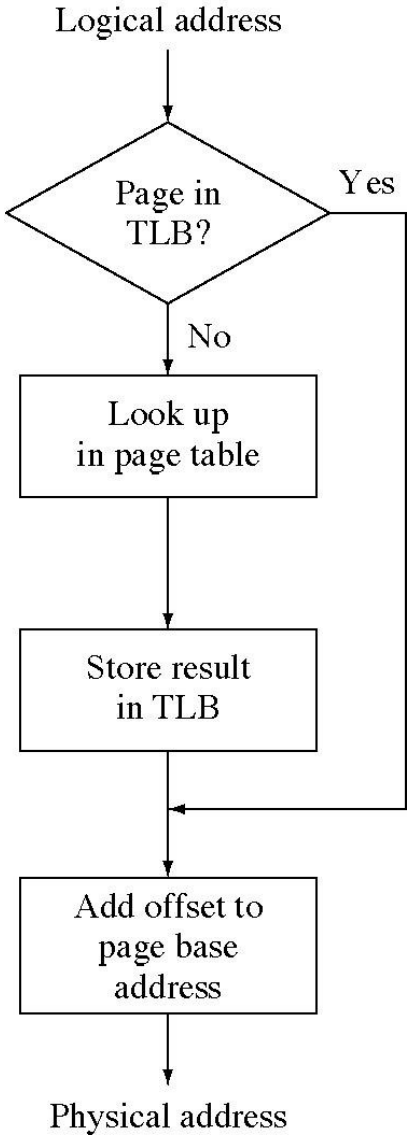
- Memória Associativa – procura paralela

Page #	Frame #

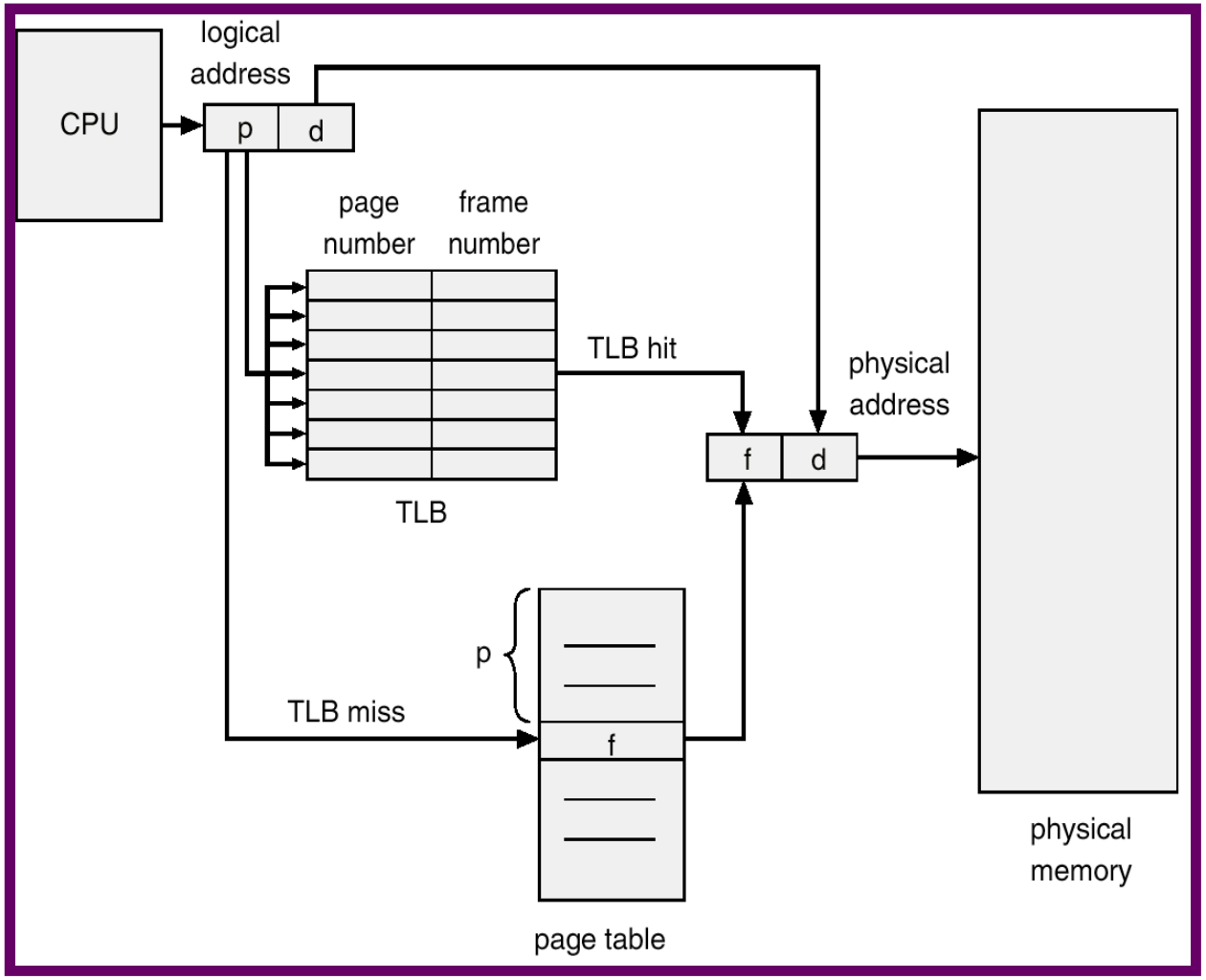
Tradução de endereço (A´, A´´)

- Se A´ esta na memória associativa (registrador), pega o frame #.
- Senão pega o frame # da tabela de páginas na memória

Fluxo com TLB



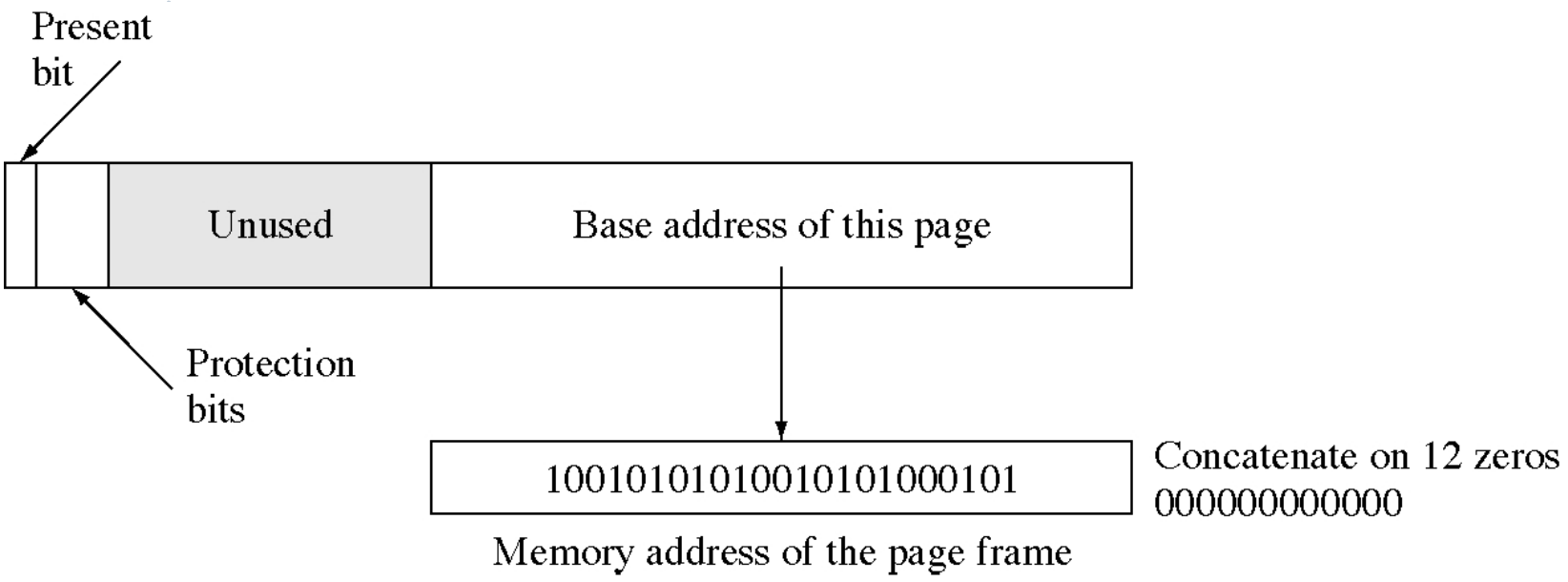
Paginação c/ TLB



Porque TLBs funcionam

- Acesso a memória não é randomico, isto é, nem todas as posições no espaço de endereços tem a mesma chance de ser referenciada
- Referências são localizadas porque
  - Código executado sequencialmente
  - loops no código
  - grupos de dados acessados juntos
  - dados é acessado várias vezes
- Esta propriedade é chamada *localidade*
- Taxa de acertos da TLB é 90+% .

Tabela de páginas : uma entrada



Proteção de páginas

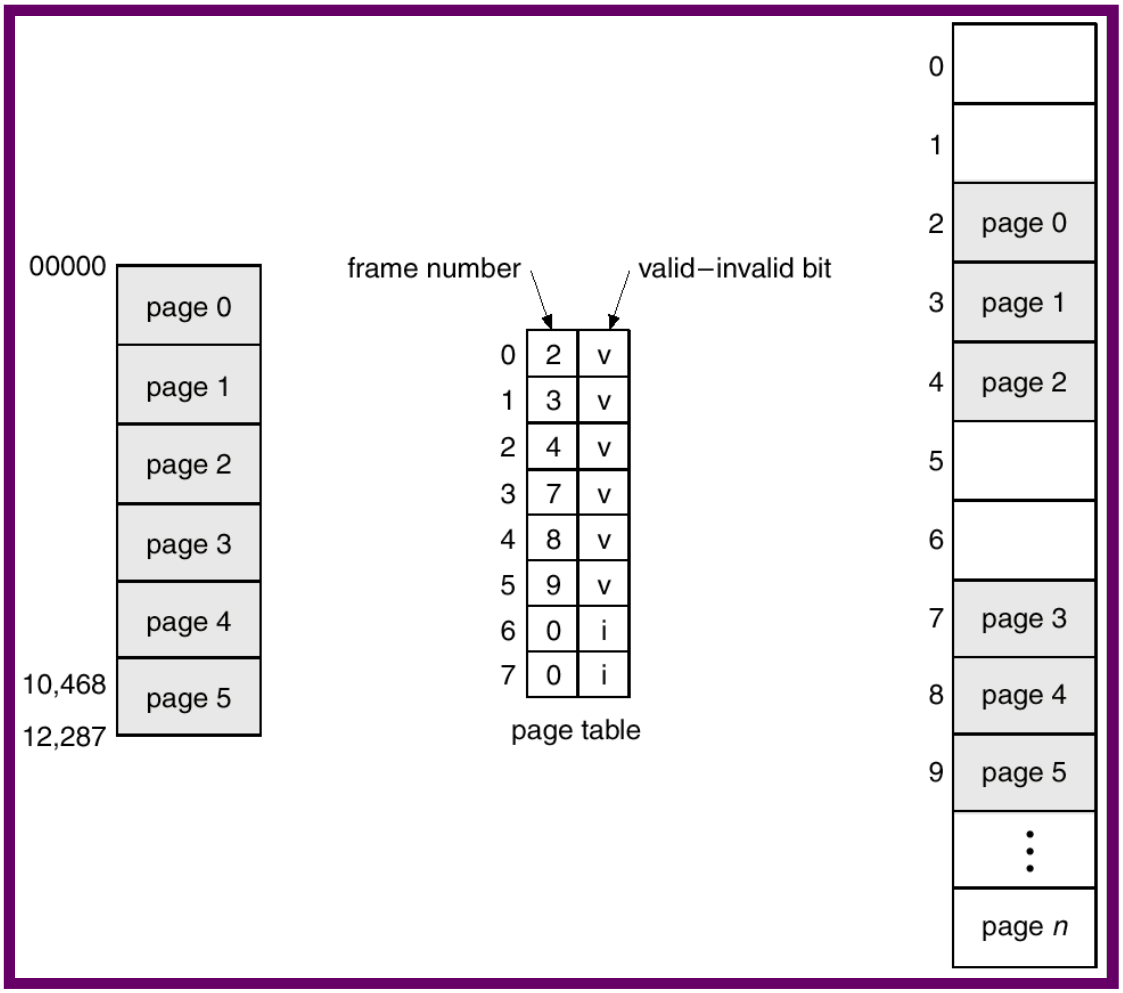
- Proteção de memória implementada através da associação de bit de proteção com cada frame.

- Bit *Valido-invalido* anexado a cada entrada da tabela de página:
  - "valido" indica que a página associada esta no espaço de endereço lógico do processo, e assim é uma página legal.
  - "invalido" indica que a página não esta no espaço de endereço lógico do processo.

Proteção de páginas

- Tres bits de controle: read, write, execute
- Possíveis modos de proteção:
  - 000: página não pode ser acessada
  - 001: página é *read only*
  - 010: página é *write only*
  - 100: página é *execute only*
  - 011: página pode ser *read or written*
  - 101: página pode ser *read as data or executed*
  - 110: *write or execute*
  - 111: qualquer acesso é permitido

Bit Valido (v) Invalido (i)



Estrutura da Tabela de Página

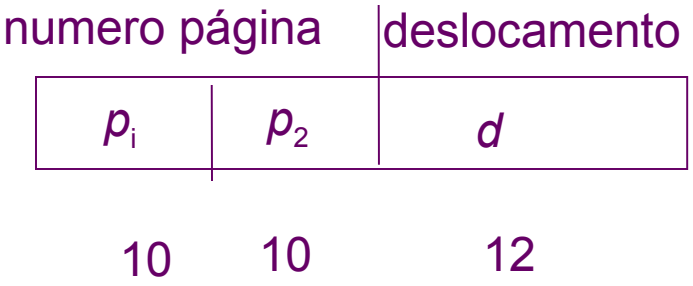
- Tabela de Página Hierárquica
- Tabela de Página c/ Hash
- Tabela de Página Invertida

Tabela de Página Hierárquica

- Particiona espaço de endereço lógico em múltiplas tabelas de páginas.
- Uma técnica simples é tabela de página de dois níveis.

tabela de página de dois níveis

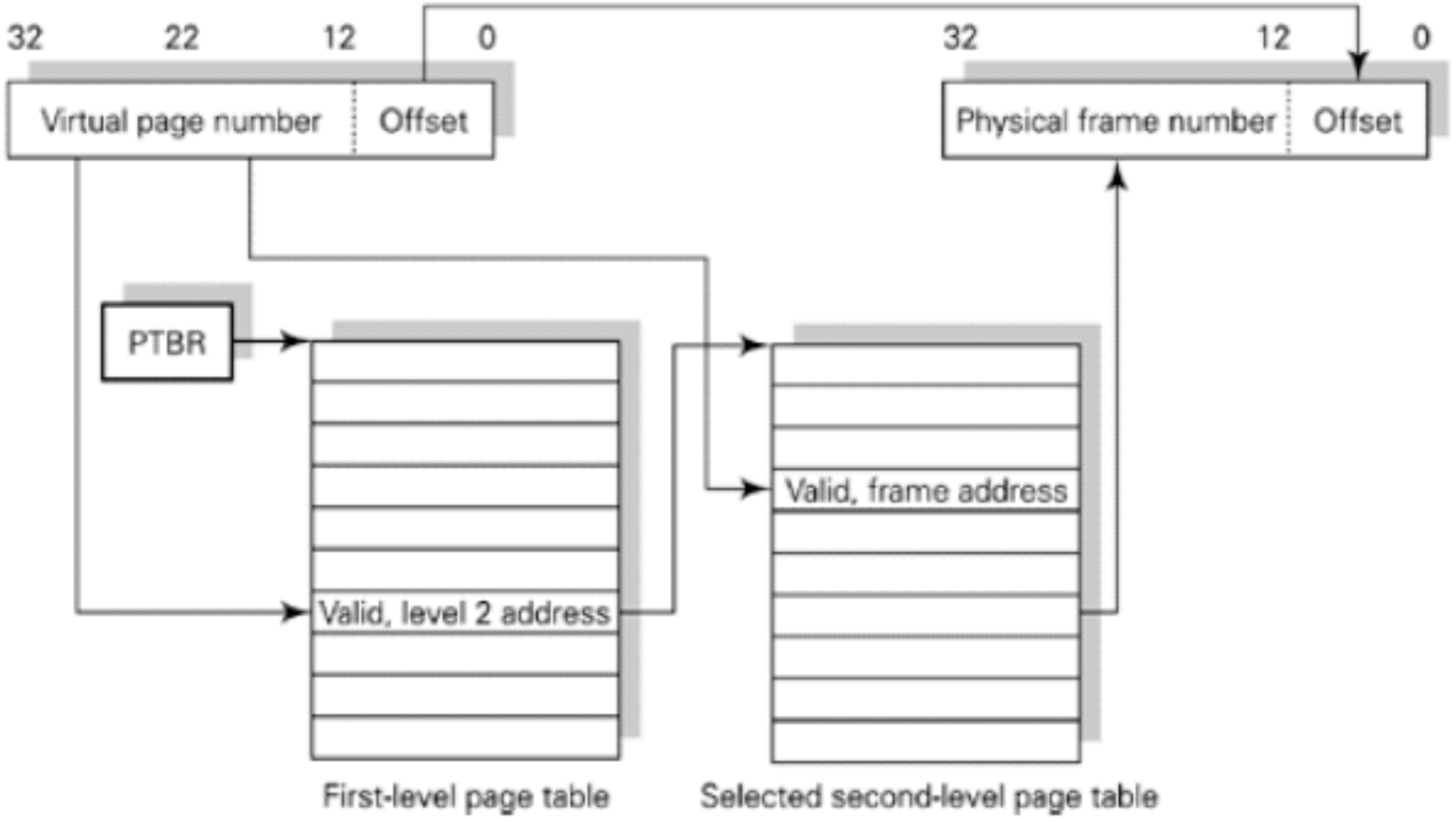
- Um endereço lógico (máquina de 32-bit c/ página de 4K) é dividido em:
  - numero da página de 20 bits.
  - deslocamento de 12 bits.
- Sendo que a tabela de páginas é paginada, o numero da página é subdividido em:
  - Numero de página, 10-bit.
  - Deslocamento, 10-bit.
- Um endereço lógico fica:



.

onde p_i é um índice na página mais externa e p_2 é o deslocamento da página na tabela de páginas mais externa

tabela de página de dois níveis



Tradução de endereços

- Esquema de tradução de endereços para uma arquitetura de paginação em dois níveis, 32-bit

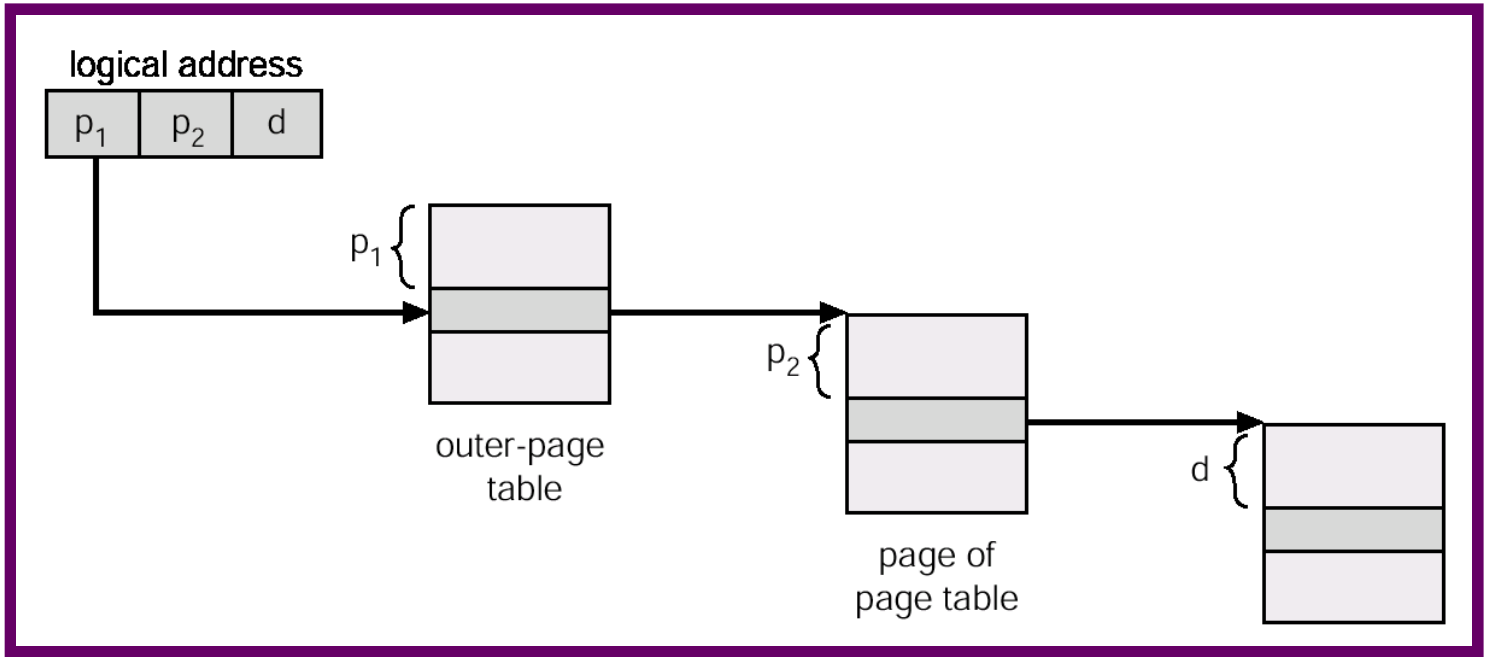


Tabela de Página c/ Hash

- Comum em espaço de endereço > 32 bits.
- O número da página virtual é submetido a uma função hash para a tabela de página. Esta tabela contém uma cadeia de elementos cuja hash aponta para a mesma posição.
- Os numeros de página virtual são comparados na cadeia em busca de igualdade. Quando encontrada, o correspondente frame físico é extraído.

Tabela de Página c/ Hash

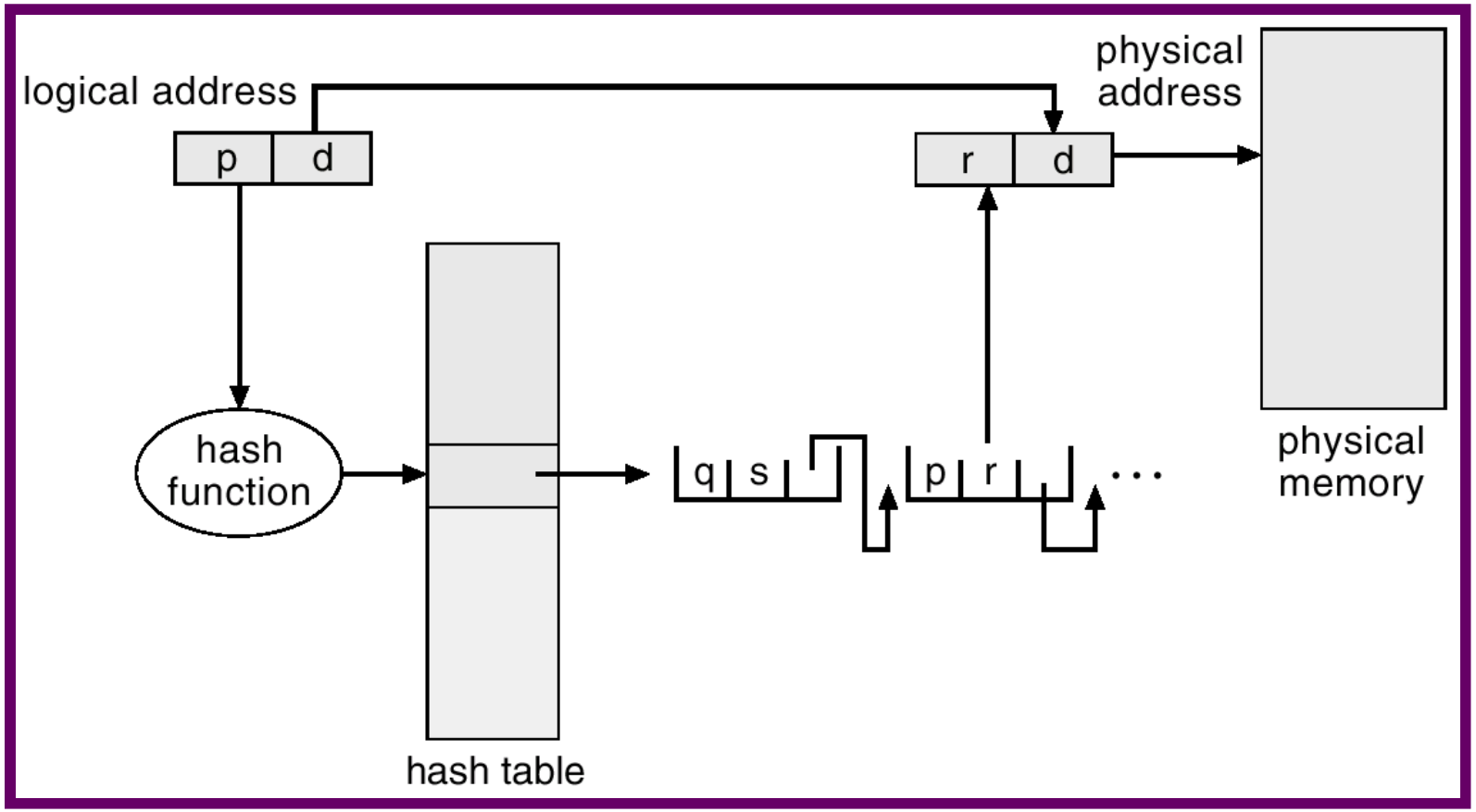


Tabela de Página Invertida

- Uma entrada para cada página real da memória.
- Entrada consiste de o endereço virtual da página armazenada na posição de memória, com informação sobre o processo que é dono da página.
- Diminuição da memória necessária para armazenar cada tabela de página, mas aumenta o tempo necessário para pesquisar a tabela quando uma referência a página ocorre.
- Usar tabela de hash para limitar a pesquisa para uma — ou no máximo poucas — entradas da tabela de página.

Tabela de Página Invertida

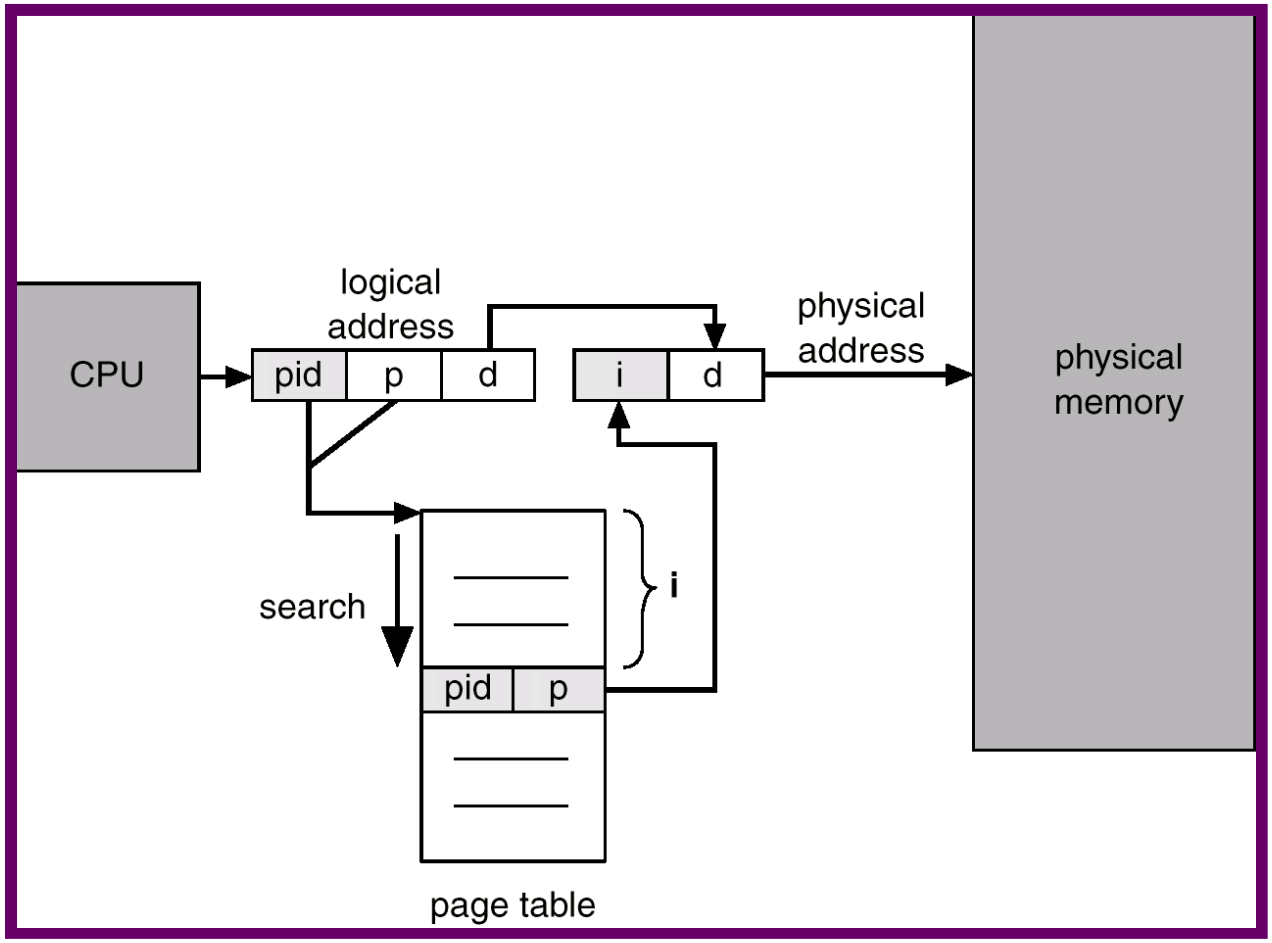
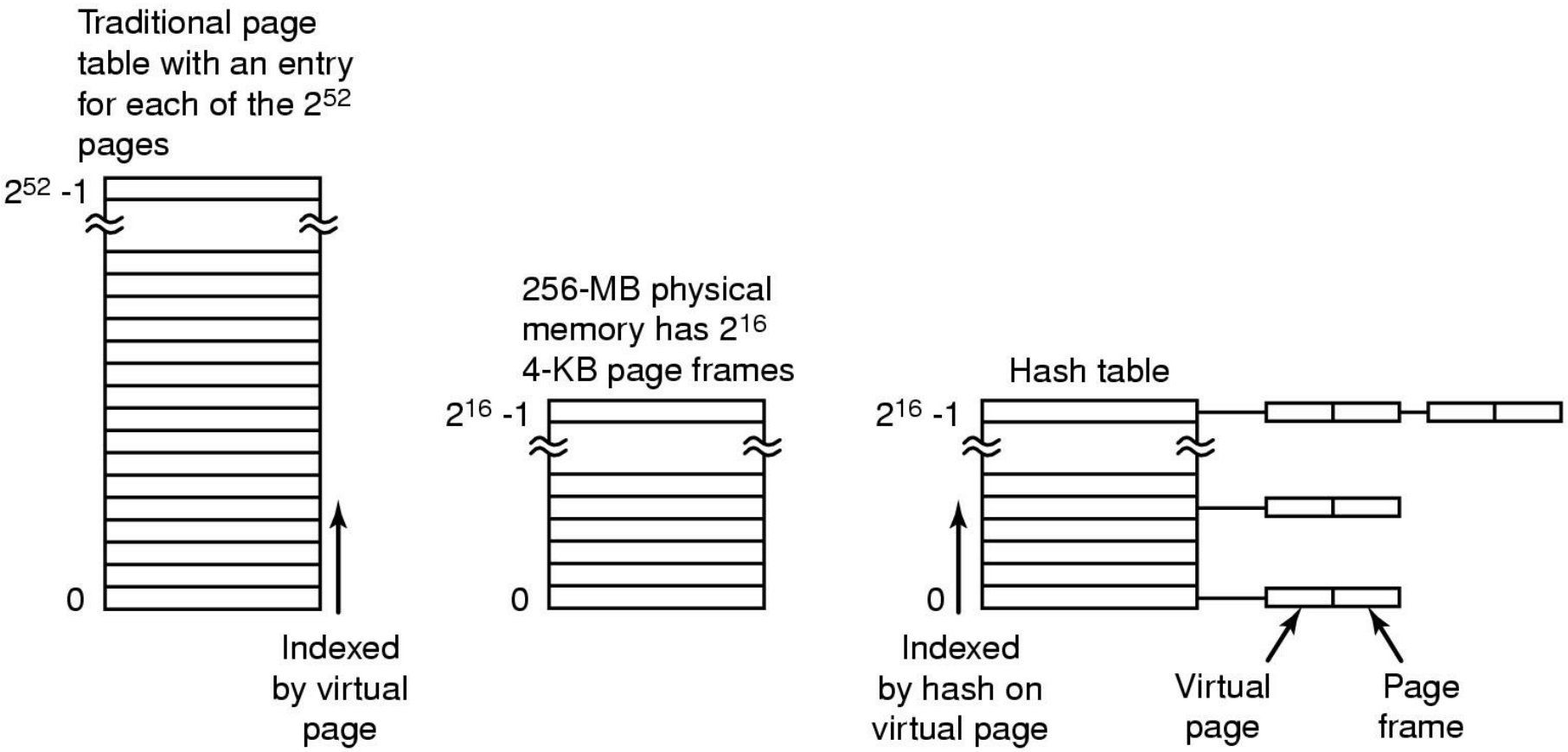


Tabela de Página Invertida

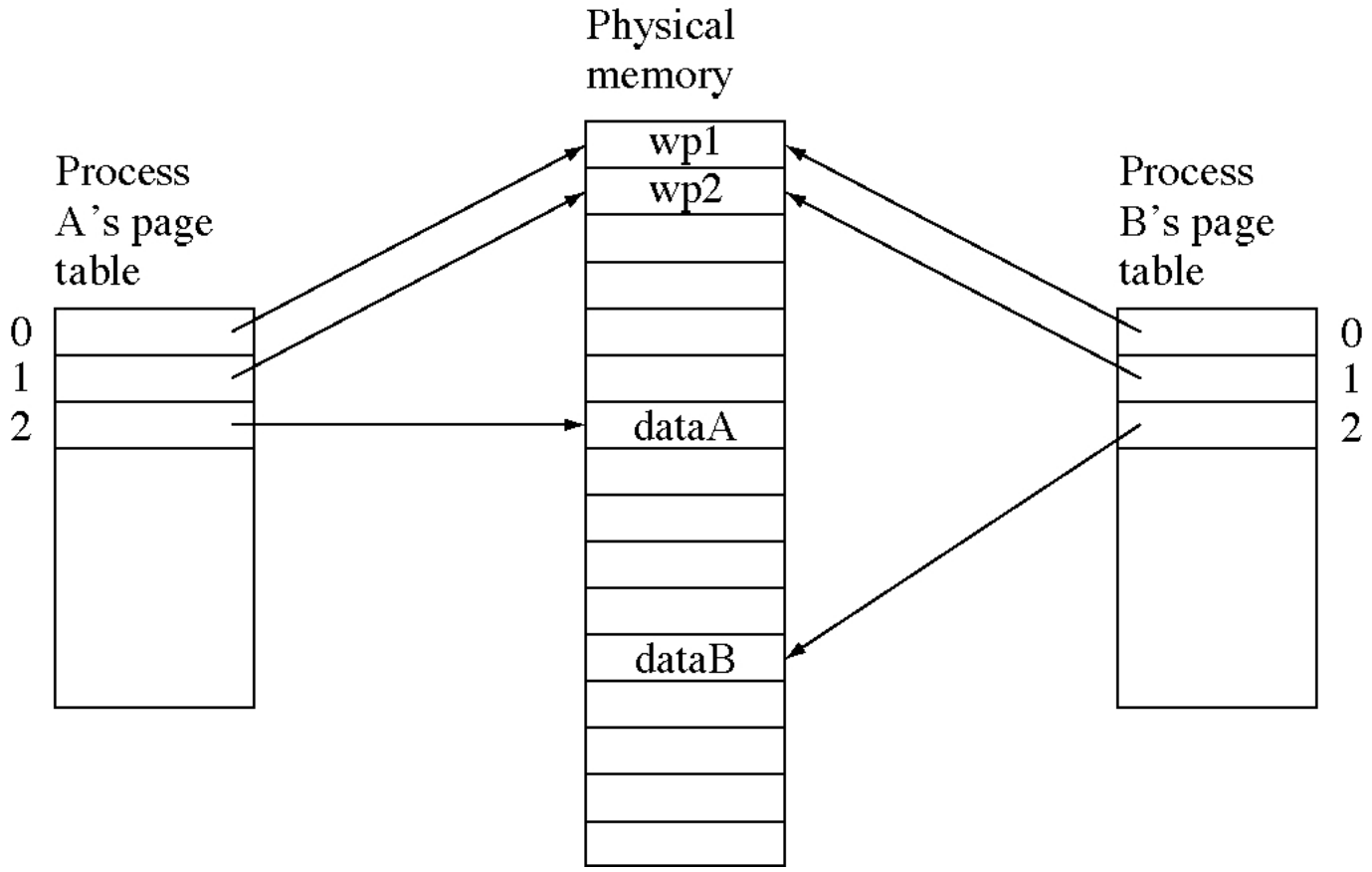


Páginas Compartilhadas

- Código compartilhado
  - Uma cópia de código *read-only* (reentrante) compartilhado entre processos (i.e., editores de texto, compiladores, etc).
  - Código compartilhado deve aparecer na mesma posição no espaço de endereçamento lógico de todos processos.

- Código privado e dados
  - Cada processo mantém uma cópia separada de código e dados.
  - As páginas para o código privado e dados pode aparecer em qualquer lugar no espaço de endereçamento lógico.

Dois processos compartilhando código



Paginação do SO?

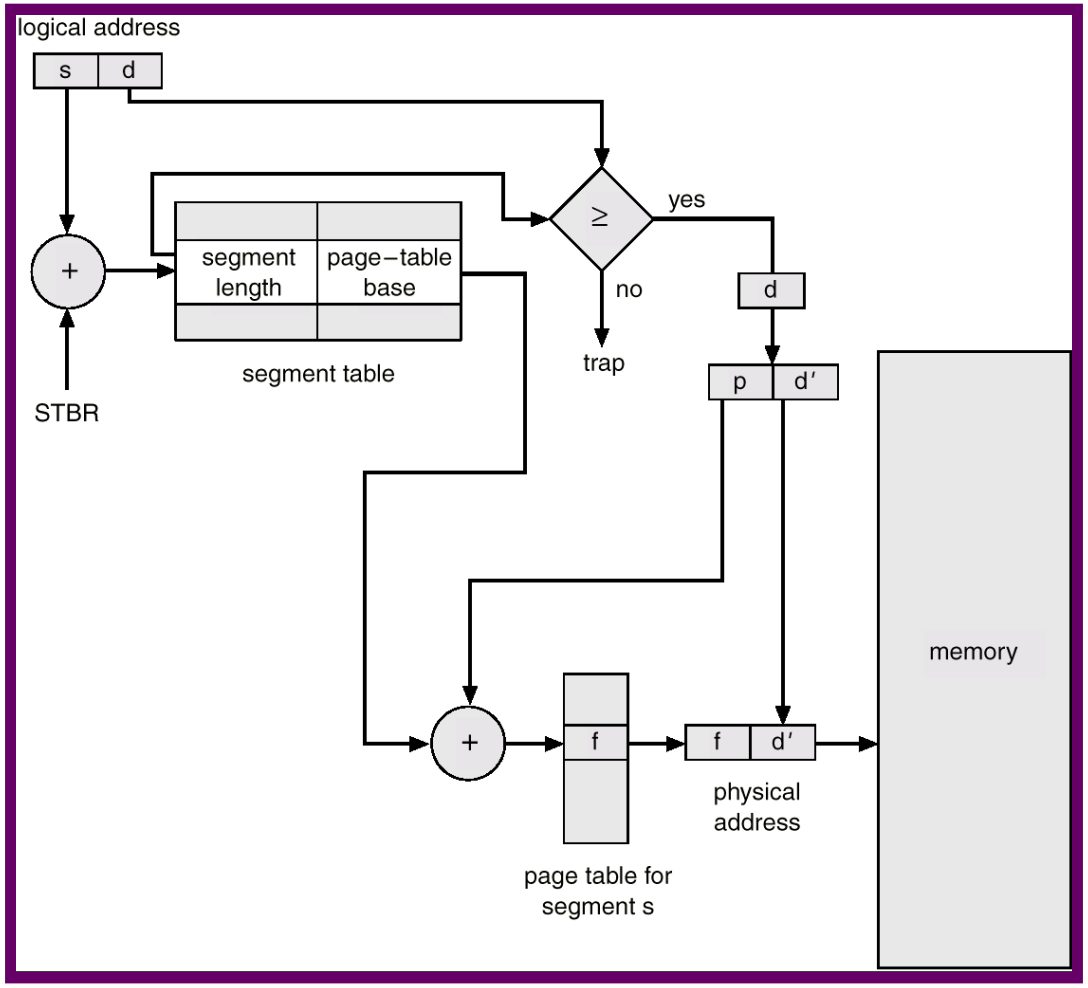
- Sim, podemos paginar código e dados do SO
  - mas algum código deve estar sempre na memória
  - como paginador e despachador
- Bloquear páginas na memória
  - prevenir que elas sejam *paged out*
  - para partes vitais do SO
  - para páginas envolvidas em operações de E/S

Segmentação c/ Paginação – MULTICS

- O MULTICS resolveu problemas de fragmentação externa e longos tempos de pesquisa através da paginação de segmentos.

- Solução difere de segmentação pura no que diz respeito as entradas da tabela que contém o endereço base de uma tabela de páginas para o segmento e não o endereço base do segmento.

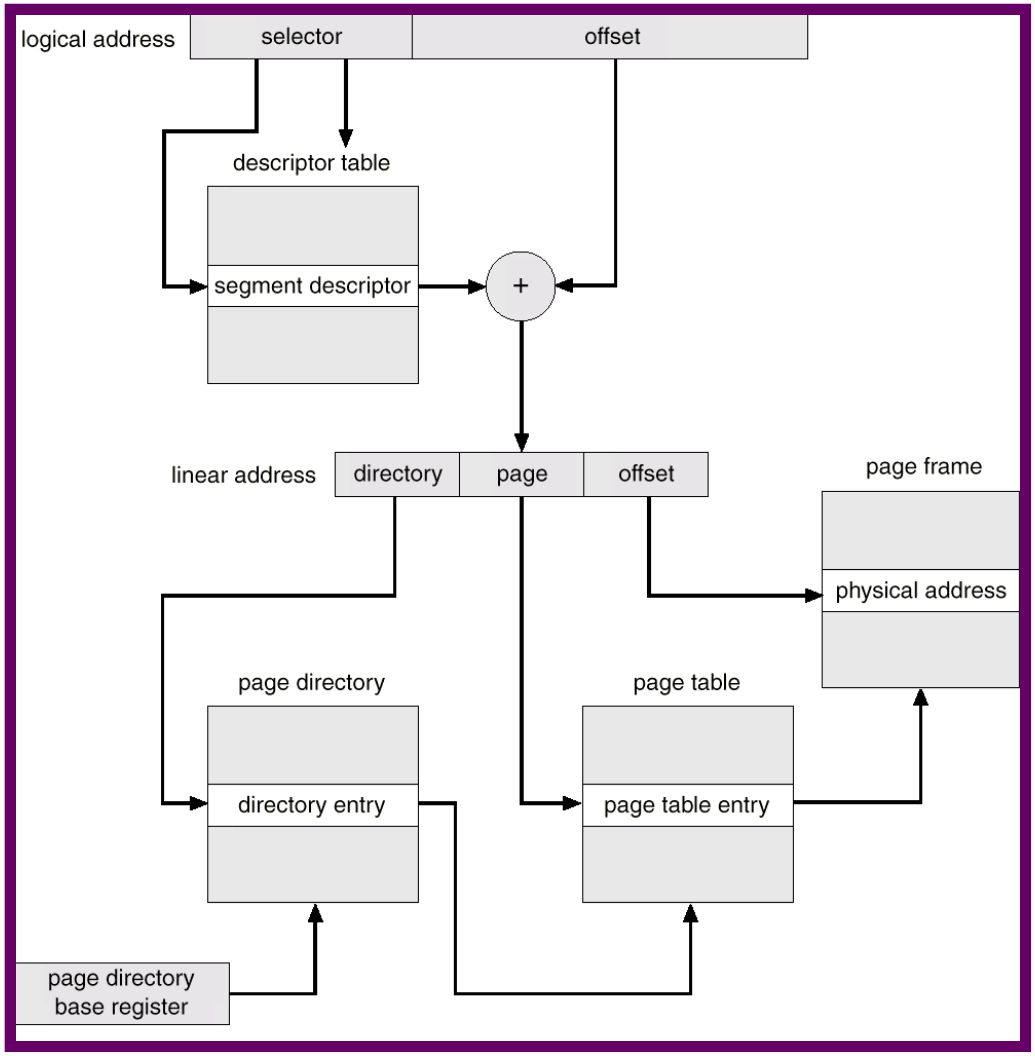
MULTICS - tradução de endereço



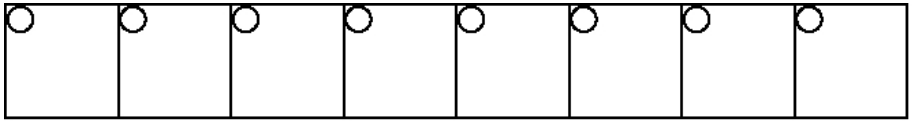
Segmentação c/ Paginação – Intel 386

- Como mostrado a seguir, o Intel 386 usa segmentação com paginação para gerenciamento de memória com um esquema de paginação em dois níveis.

Intel 386 - tradução de endereço



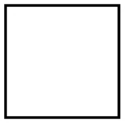
Casos de paginação



```
for(i=0;i<8;++i)
    sum+=a[i][0];
```

(worst case)

○ = Memory reference

 = Page

```
for( j=0;j<8;++j)
    sum+=a[0][j];
```

(best case)