

# Gerenciamento de Memória

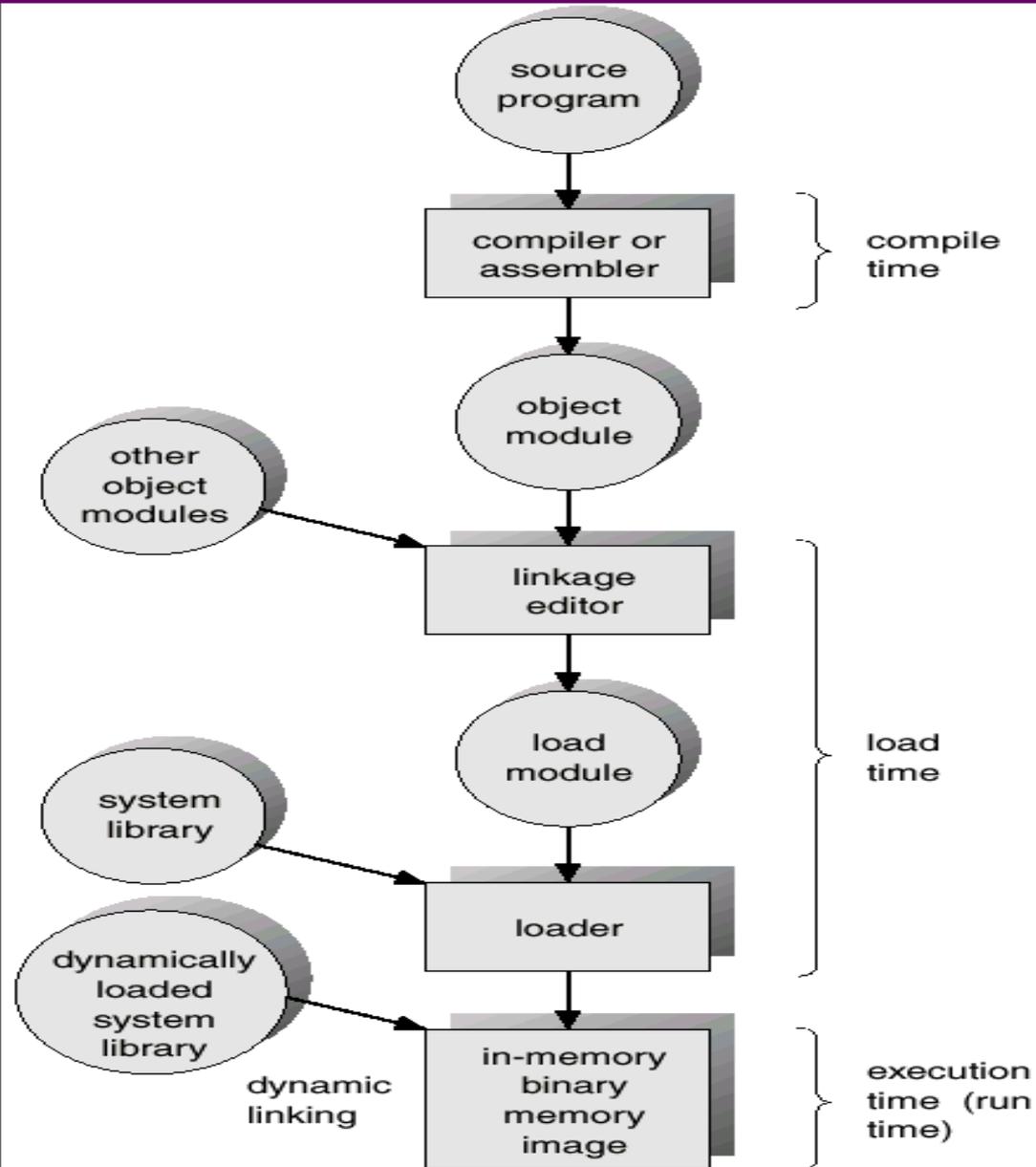
- Conceitos básicos
- Swapping
- Alocação contígua
- Paginação
- Segmentação
- Segmentação com paginação

# Atribuição de endereços (ligação) de código e dados na memória

A atribuição de endereços de instruções e de dados para endereços de memória podem ser feitos em três diferentes estágios:

- **Em tempo de compilação:** Se a localização na memória já é conhecida a priori, um código de endereçamento absoluto por ser gerado. É necessário recompilar o código se o endereço inicial muda.
- **Em tempo de carregamento:** Precisa gerar código relocável se a localização de memória não é conhecida em tempo de compilação.
- **Em tempo de execução:** se o processo em execução pode ser movido de um segmento de memória para outro, então a vinculação é atrasada até o tempo de execução. Necessita de hardware para suportar mapeamento de endereço como por exemplo, registradores base e limite.

# Processamento de um programa



# Espaços de endereçamento lógico e físico

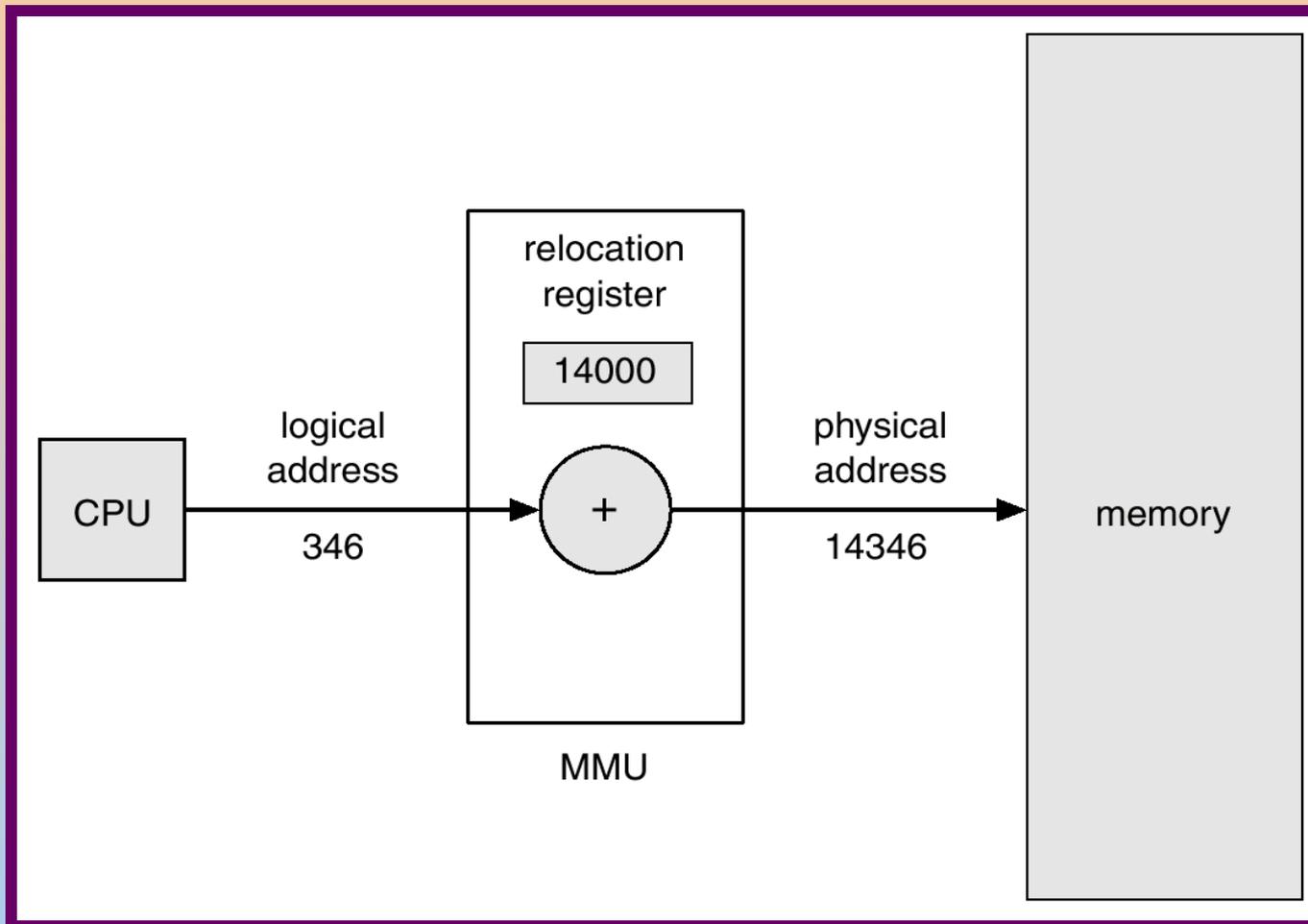
- A separação entre espaço de endereçamento lógico e físico é fundamental para gerenciamento de memória.
  - ◆ *Endereço lógico* – gerado pela CPU; também conhecido como *endereço virtual*.
  - ◆ *Endereço físico* – endereço visto por uma unidade de memória.
- Endereços lógicos e físicos são os mesmos em esquemas de atribuição de endereços em tempo de compilação e de carregamento; porém, endereços lógicos (virtuais) e endereços físicos diferem nos esquemas de vinculação de endereço em tempo de execução.

# Memory-Management Unit (MMU)

## Unidade de gerenciamento de memória

- Dispositivo de hardware que mapeia endereços virtuais em físicos.
- Dentro do esquema da MMU, o valor de um registrador de relocação é adicionado a todo endereço gerado pelo processo de um usuário, antes de ser enviado para a memória.
- O programa do usuário lida sempre com endereços lógicos; ele nunca vê endereços físicos.

# Relocação dinâmica usando um registrador de relocação



# Carregamento dinâmico

- Rotina não é carregada até que ela seja invocada
- Consiste numa utilização mais otimizada de memória; rotinas não utilizadas nunca são carregadas.
- Útil quando grandes quantidades de código são necessários para manipular casos pouco frequentes.

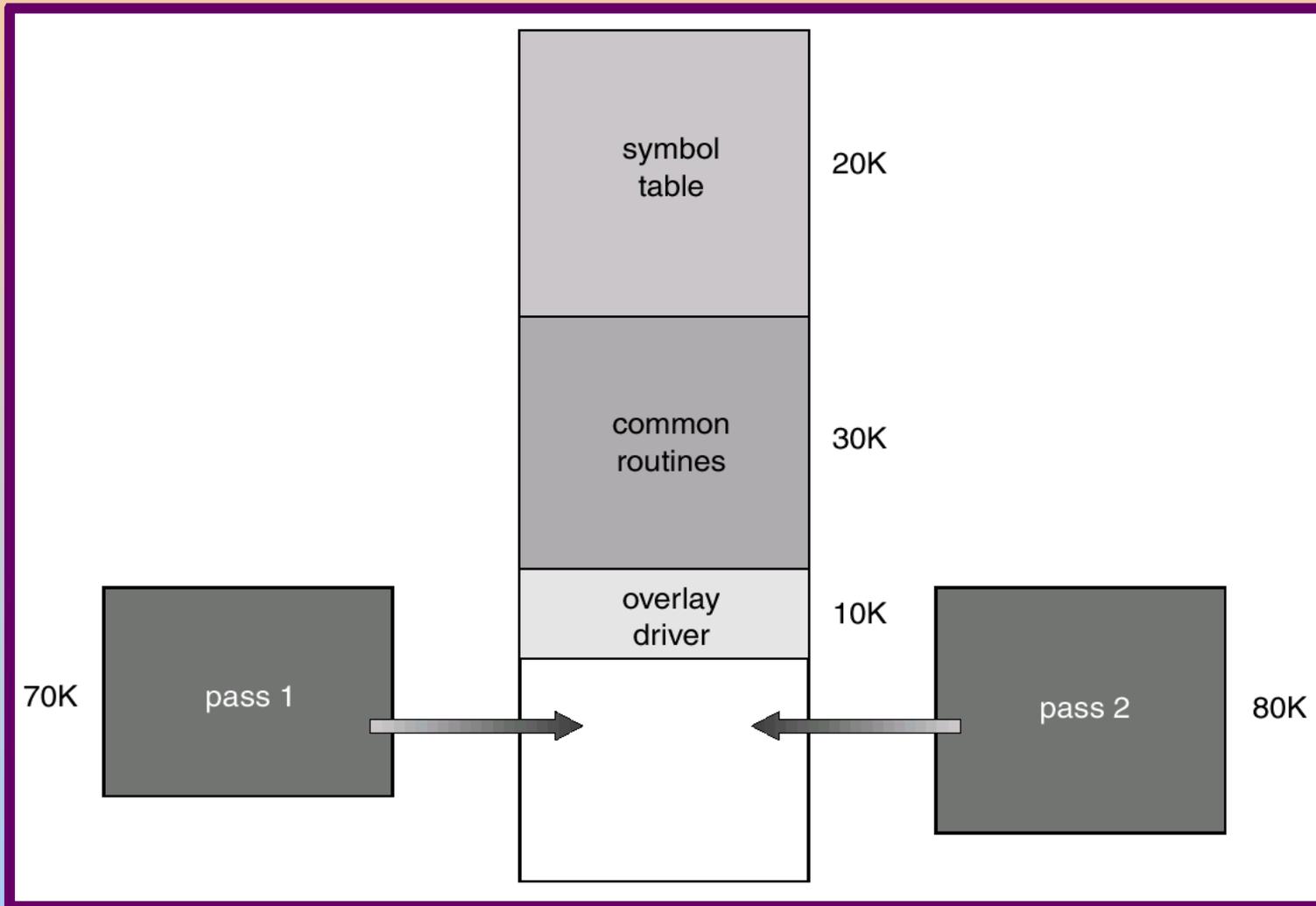
# Ligação dinâmica

- Ligação é adiada até o tempo de execução.
- Pequeno pedaço de código, o **stub**, é usado para localizar a apropriada rotina, residente na memória.
- O **stub** substitui a si mesmo com o endereço da rotina e a executa.
- O sistema operacional necessita apenas checar se a rotina está no espaço de endereçamento do processo.
- Ligação dinâmica é particularmente útil para bibliotecas.

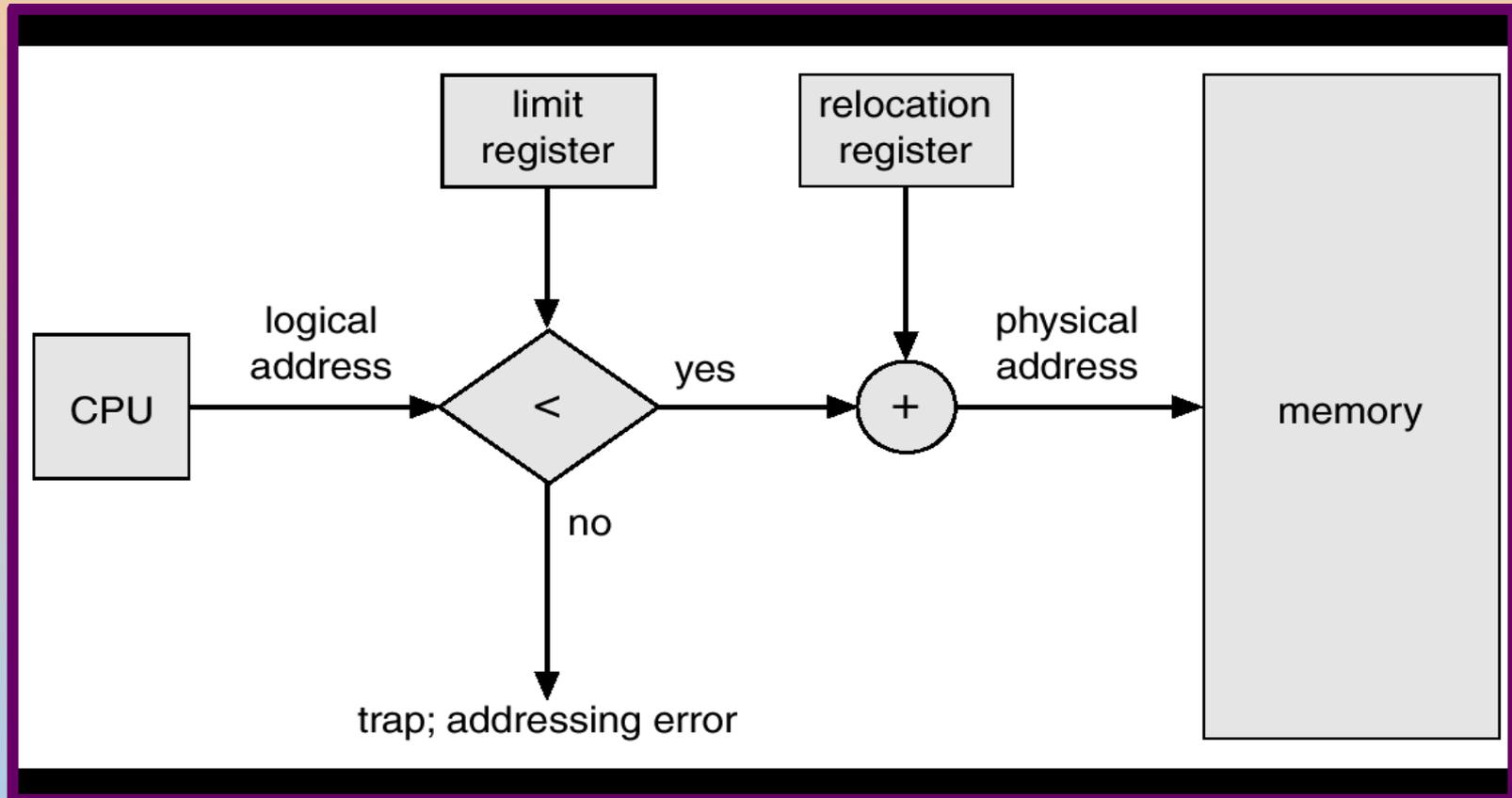
# Overlays(sobreposições)

- Manter na memória somente aquelas instruções e dados que são necessários num determinado instante.
- Necessárias quando um processo é maior que a quantidade de memória alocada para ele.
- Implementada pelo usuário, sem necessidade de suporte especial pelo sistema operacional. O projeto da estrutura de overlay é complexo.

# Overlays para um montador de dois passos



# Suporte de hardware para registradores de relocação e limite



# Paginação

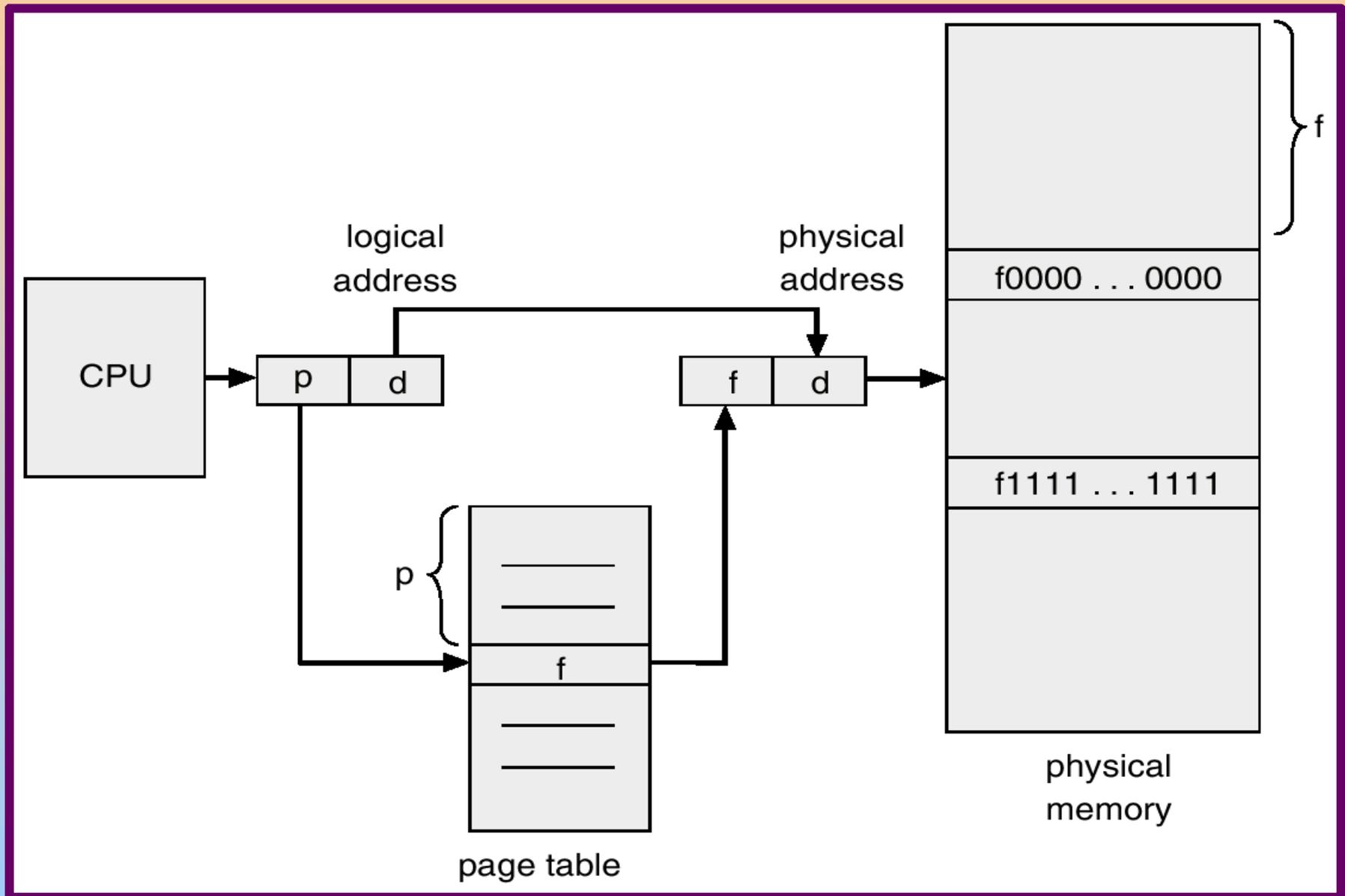
- O espaço de endereçamento lógico de um processo pode ser não contíguo;
- Dividir a memória física em blocos de tamanho fixo chamados **quadros**(frames), normalmente potência de 2, entre 512 e 8192 bytes.
- Dividir a memória lógica em blocos de mesmo tamanho chamados **páginas**.
- Manter registro de todos os quadros livres.
- Para rodar um programa dividido em  $n$  páginas, o SO precisa encontrar  $n$  quadros livres e carregar o programa.
- Usar uma tabela de páginas para traduzir o endereço lógico para um endereço físico.
- Pode ocorrer fragmentação interna.

# Esquema de tradução de endereço

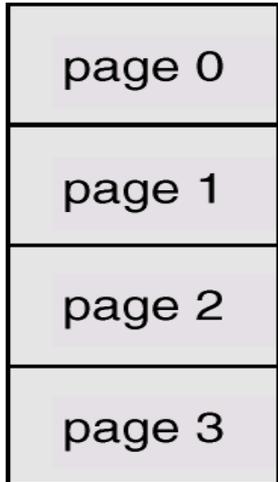
## ■ Endereço gerado pela CPU é dividido em:

- ◆ *Número da página ( $p$ )* – usado como um índice na tabela de páginas que contém o endereço-base de cada página na memória física.
- ◆ *Deslocamento(offset) na página ( $d$ )* – combinado com o endereço base para definir o endereço físico que será enviado à unidade de memória.

# Arquitetura de tradução de endereço



# Exemplo de paginação

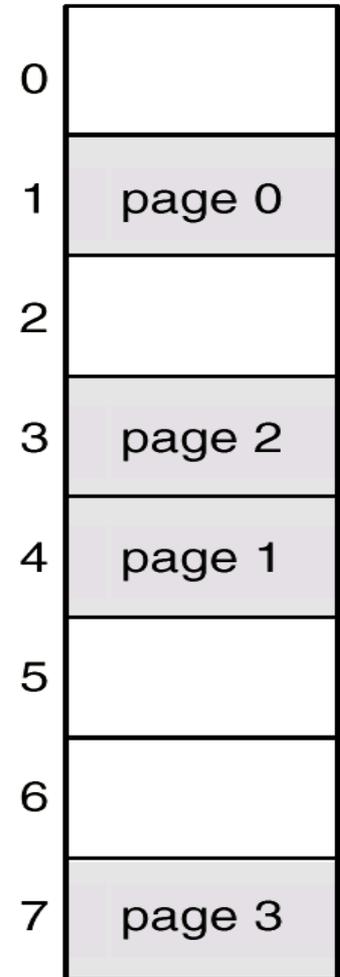


logical  
memory

0	1
1	4
2	3
3	7

page table

frame  
number



physical  
memory

# Exemplo de paginação: uma memória com 32bytes e páginas de 4 bytes

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p

logical memory

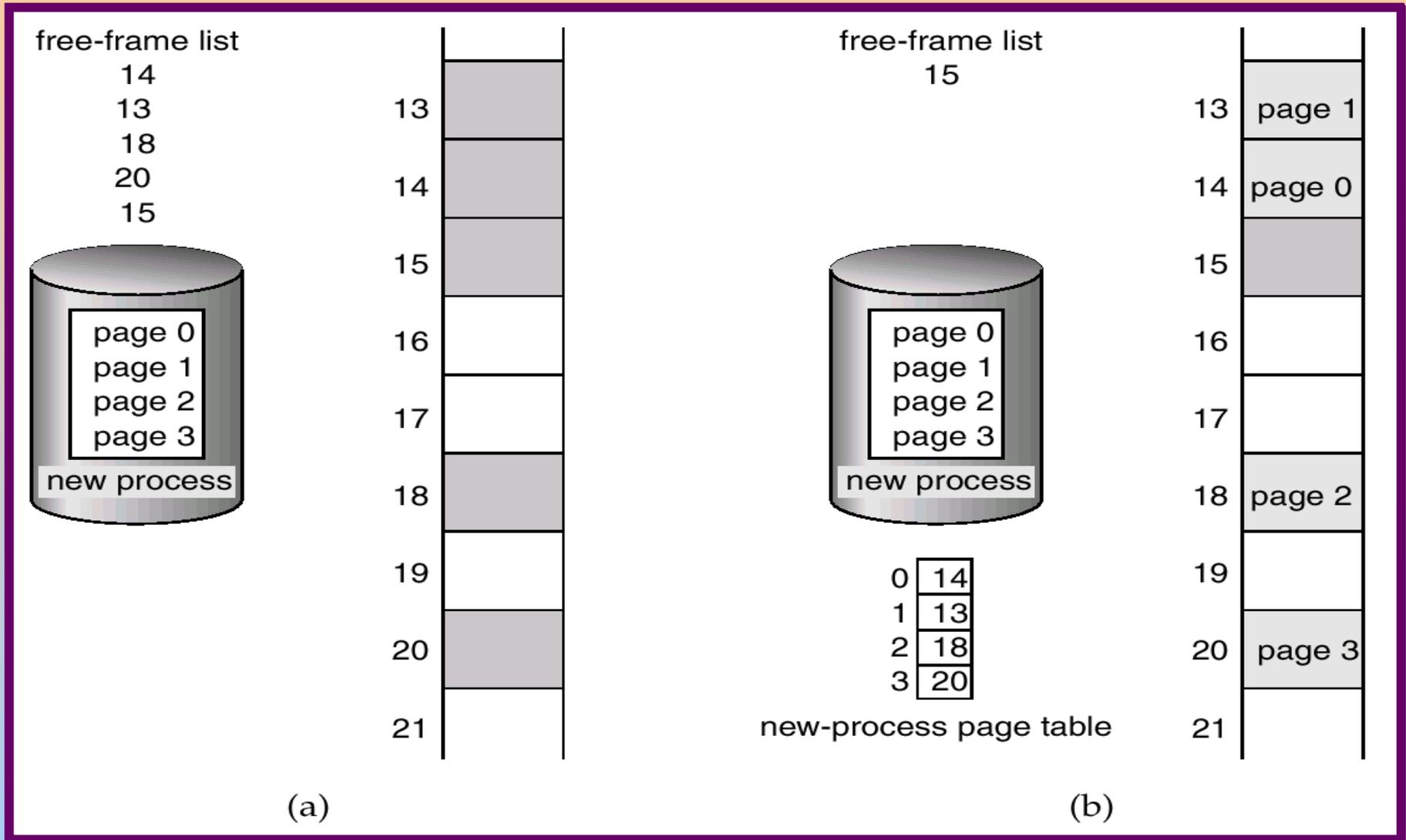
0	5
1	6
2	1
3	2

page table

0	
4	i j k l
8	m n o p
12	
16	
20	a b c d
24	e f g h
28	

physical memory

# Quadros livres



Antes da alocação

Depois de alocação

# Implementação da tabela de páginas

- Tabela de páginas é mantida na memória principal
- *Registrador-base para a tabela de páginas (PTBR)* aponta para a tabela de páginas.
- *Registrador de tamanho da tabela de páginas (PRLR)* indica o tamanho da tabela de páginas.
- Neste esquema, cada acesso a dados/instrução requer dois acessos à memória. Um para a tabela de páginas e outro para dados/instrução.
- O problema dos dois acessos à memória podem ser resolvidos usando usando um hardware de cache especial de busca rápida chamado memória associativa ou registradores associativos(*translation look-aside buffers (TLBs)* ).
- Um TLB armazena somente algumas entradas da tabela de páginas.

# Exercício

1. Suponha que um sistema implemente páginas de 2048 bytes e cujos endereços lógicos fossem de 16 bits
  - a) Qual seria a quantidade de páginas endereçáveis na memória virtual desse sistema ?
    - ◆ Supondo que esse sistema disponha de 16 páginas ou blocos físicos de memória, desenhe a tabela de páginas desse sistema com os seguintes mapeamentos:  $0 \rightarrow 5$ ,  $1 \rightarrow 2$ ,  $2 \rightarrow 10$ ,  $3 \rightarrow$  indisponível,  $4 \rightarrow 8$ ,  $5 \rightarrow 3$
  - c) Traduza os seguintes endereços lógicos em endereços físicos: 02C5h, 2162h, 3B34h
2. Qual seria o aproveitamento efetivo da memória alocada po esse sistema, no caso de:
  - a) Um processo que necessitasse alocar 3000 bytes
  - b) Um processo que necessitasse alocar 10000 bytes

# Memória associativa

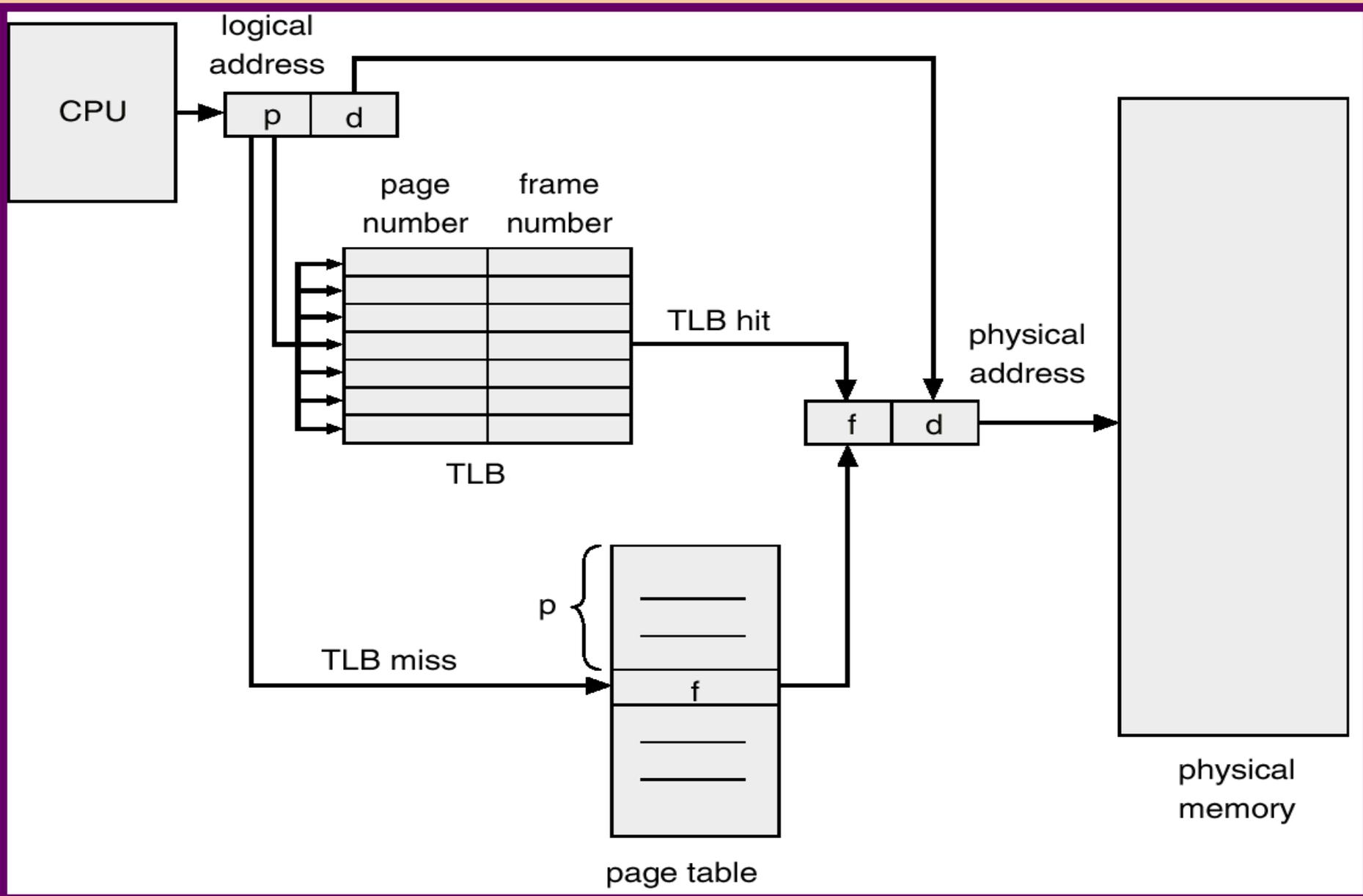
## ■ Memória associativa: busca paralela

Número página	Número quadro

### Tradução de endereço ( $A'$ , $A''$ )

- ◆ Se  $A'$  está no registrador associativo, devolva o número do quadro  $A''$ .
- ◆ Caso contrário, carregue o número da página na memória associativa, juntamente com o número do frame.

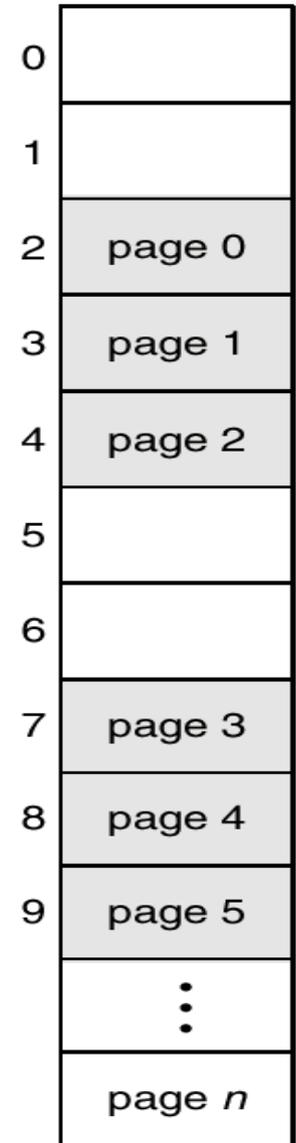
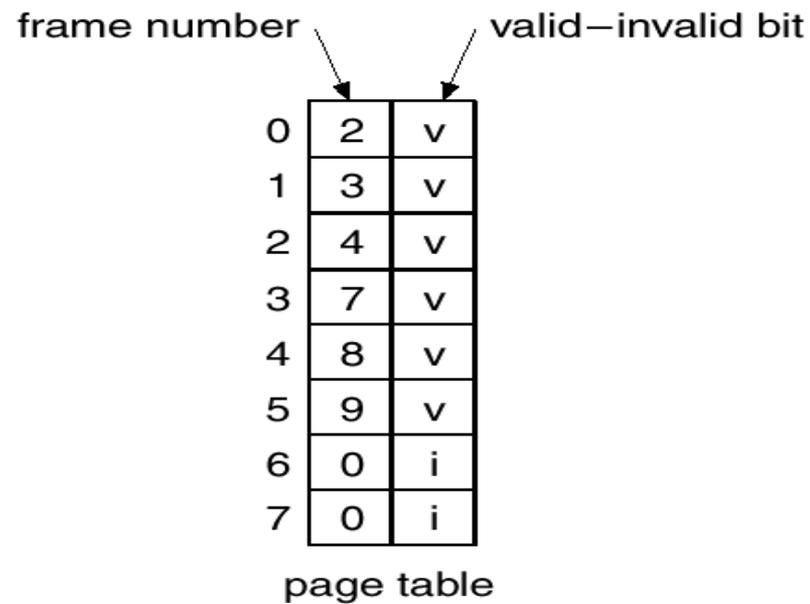
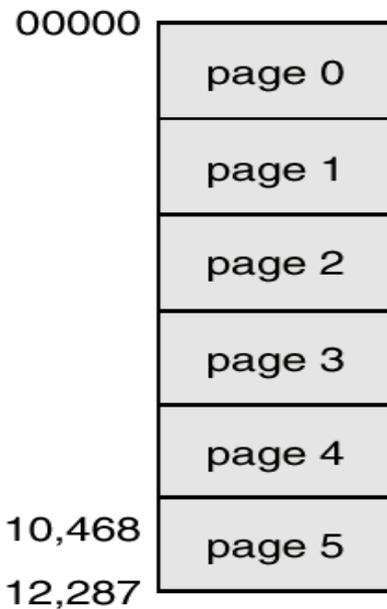
# Hardware de paginação com TLB



# Proteção de memória

- Proteção de memória é implementada associando-se um bit de proteção com cada quadro.
- Um bit de válido-não válido é associado com cada entrada na tabela de páginas:
  - ◆ “válido” indica que a página associada está no espaço de endereçamento lógico do processo e é uma página legal.
  - ◆ “inválido” indica que a página não está no espaço de endereçamento lógico do processo.

# Bits válido (v) ou inválido (i) Bit em uma tabela de páginas



# Estrutura da tabela de páginas

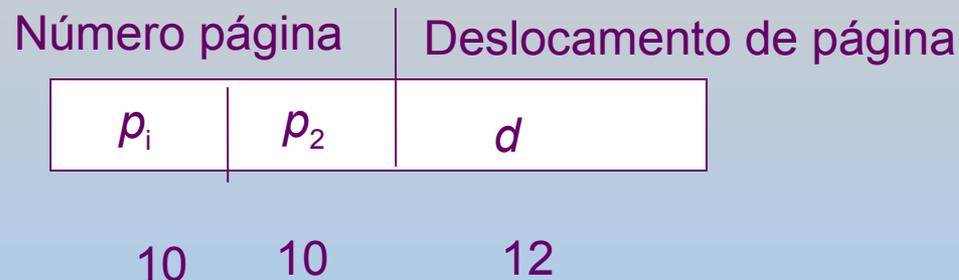
- Paginação hierárquica
- Tabelas de paginação com hashing

# Tabelas de paginação hierárquica

- Quebra o espaço de endereçamento lógico em várias tabelas de páginas.
- Um técnica simples é a paginação em dois níveis.

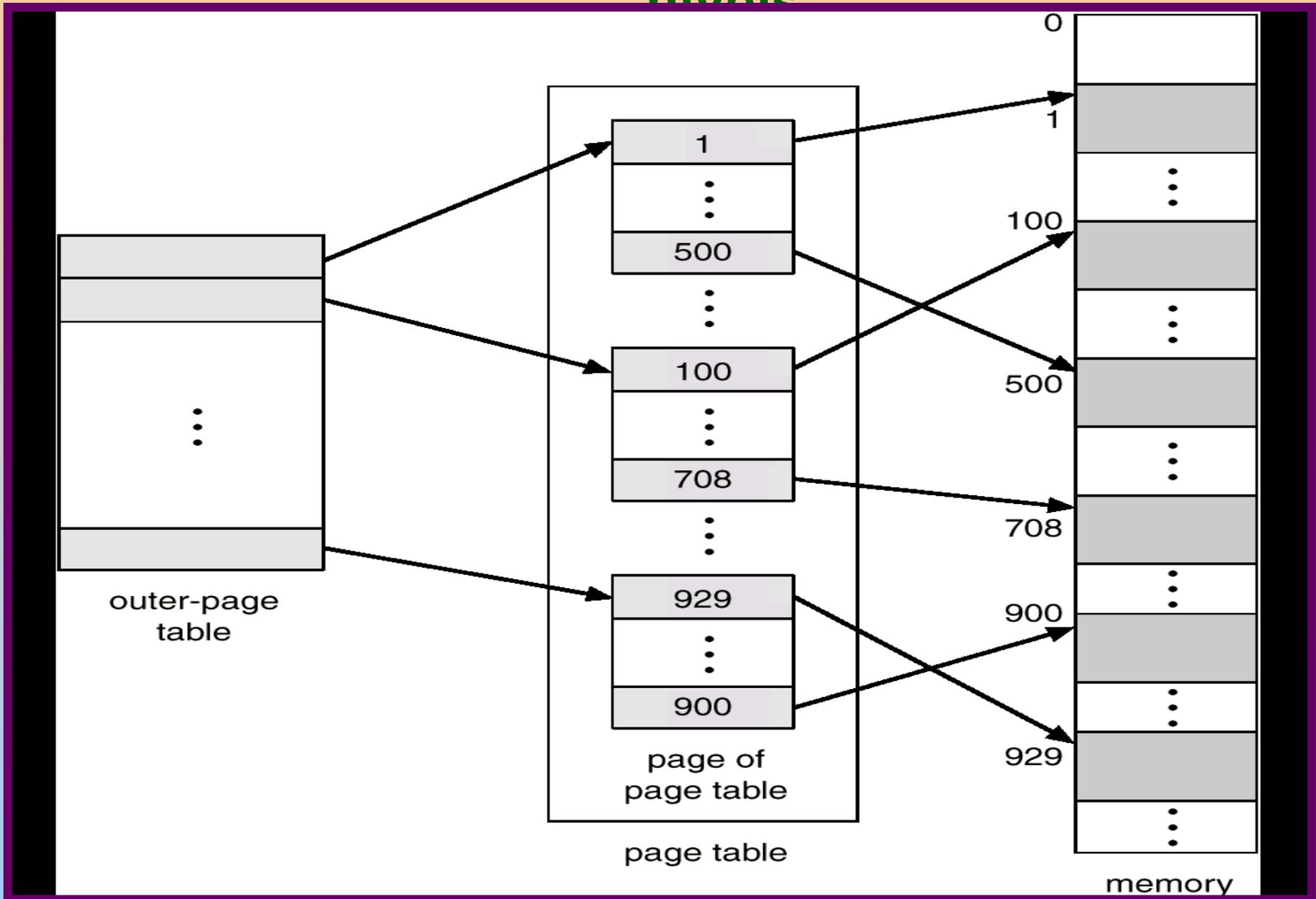
# Exemplo de paginação em dois níveis

- Um endereço lógico (em máquinas de 32 bits com tamanho de página 4k) é dividido em:
  - ◆ Um número de página consistindo de 20 bits.
  - ◆ Um deslocamento na página de 12 bits.
- Como a tabela de páginas será paginada, o número da página é dividido em:
  - ◆ Um número de página com 10 bits.
  - ◆ Um deslocamento de página com 10 bits.
- Então, um endereço lógico é como:



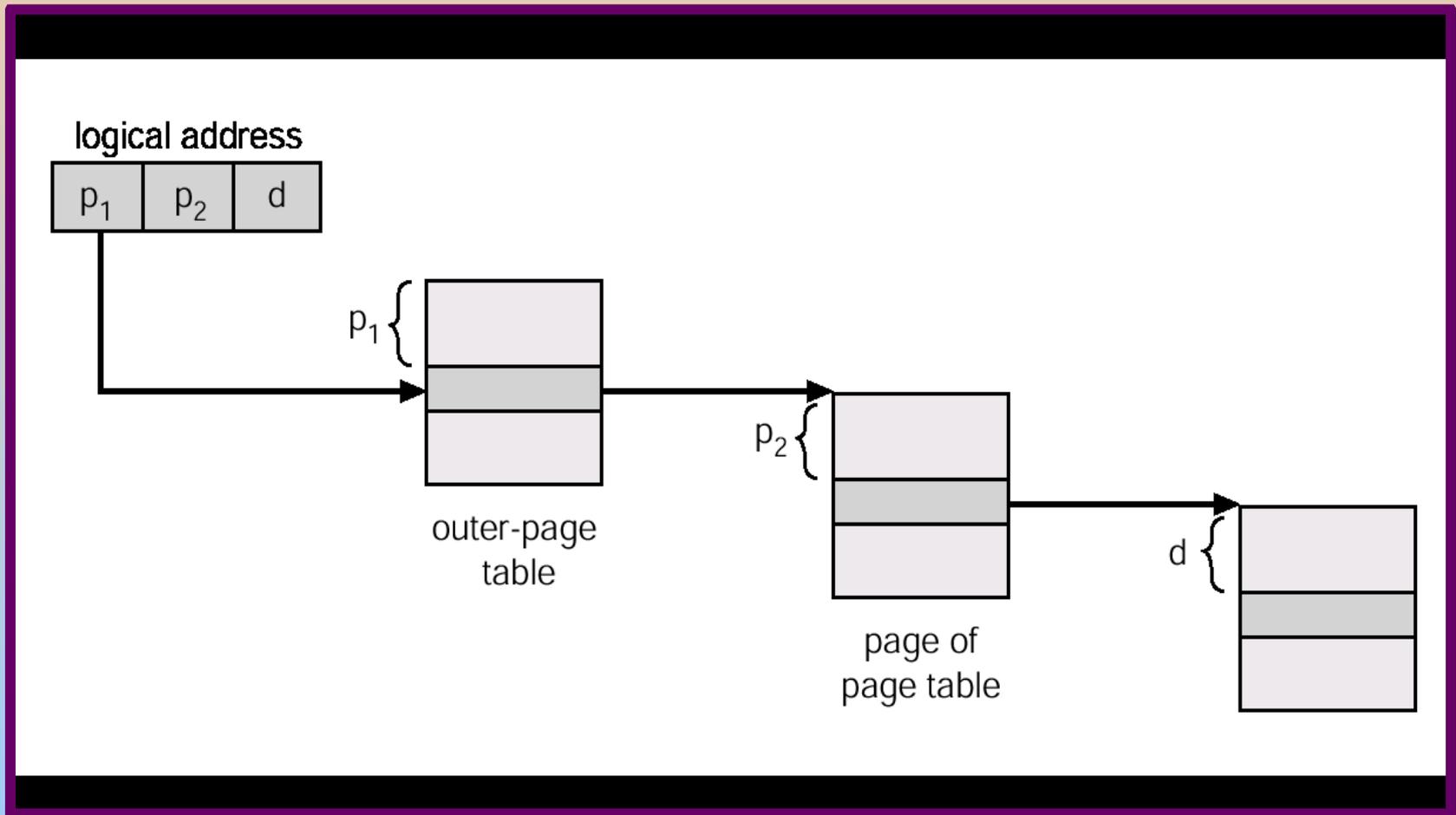
onde  $p_1$  é um índice numa tabela externa(outer-table), e  $p_2$  é o deslocamento dentro da tabela interna.

# Esquema de tabela de páginas em dois níveis



# Esquema de tradução de endereço

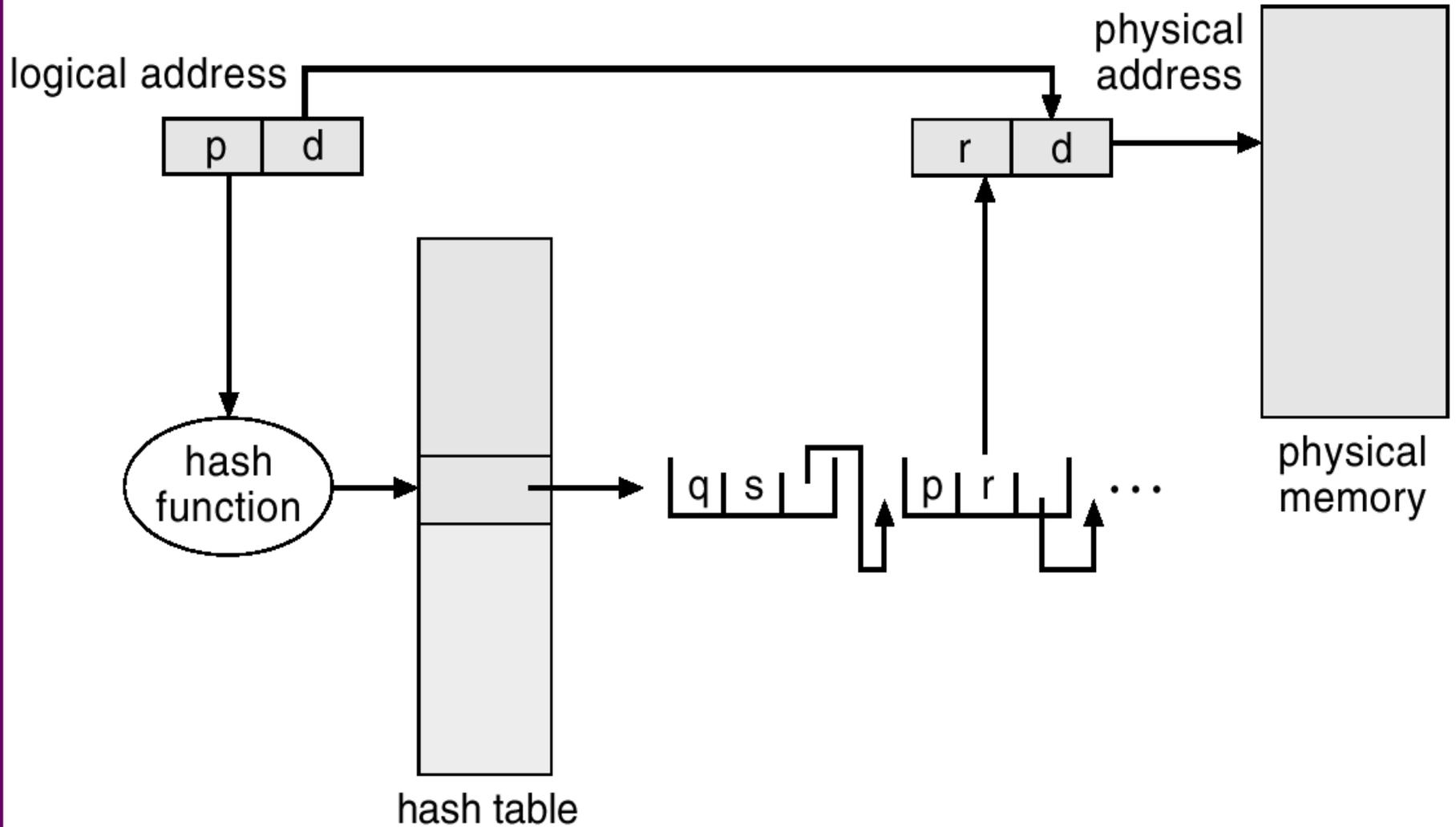
- Esquema de tradução de endereço com arquitetura de paginação 32-bits em dois níveis.



# Tabelas de paginação com hashing

- Comum em espaços de endereçamento  $> 32$  bits.
- Utiliza-se hashing com o número da página. Na entrada correspondente a esta chave existe uma lista ligada de páginas mapeados para a mesma posição.
- Efetua-se uma busca nesta lista de páginas. Encontrando-se a página desejada, o número do quadro é retornado.

# Tabela de paginação com hashing



# Páginas compartilhadas

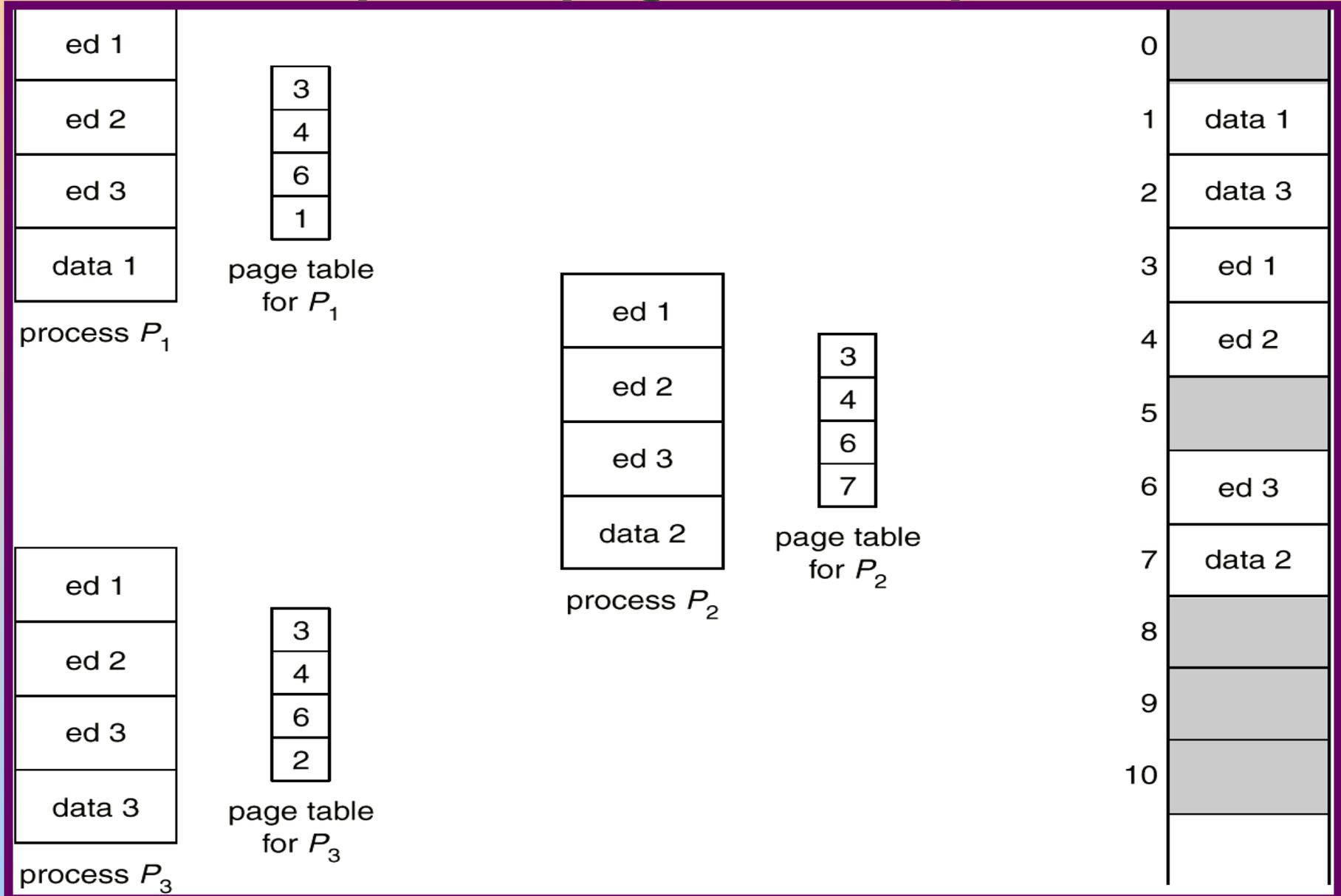
## ■ Código compartilhado

- ◆ Uma cópia de código de somente leitura compartilhado entre processos ( código reentrante). Exemplo: editores de texto, compiladores, sistemas de janela.
- ◆ Código compartilhado precisa aparecer na mesma localização no espaço de endereamento lógico de todos os processos.

## ■ Código e dados privados

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

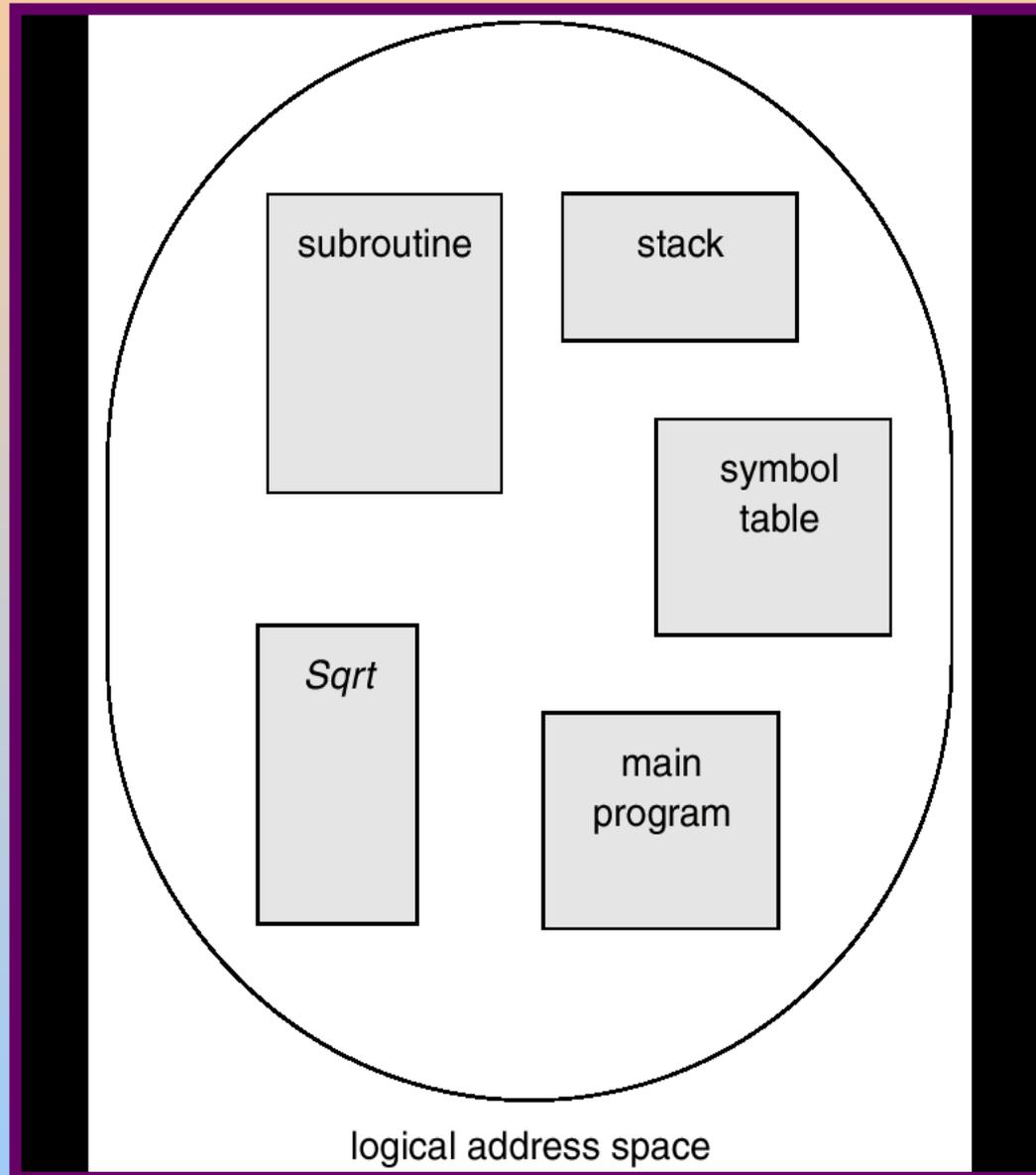
# Exemplo de páginas compartilhadas



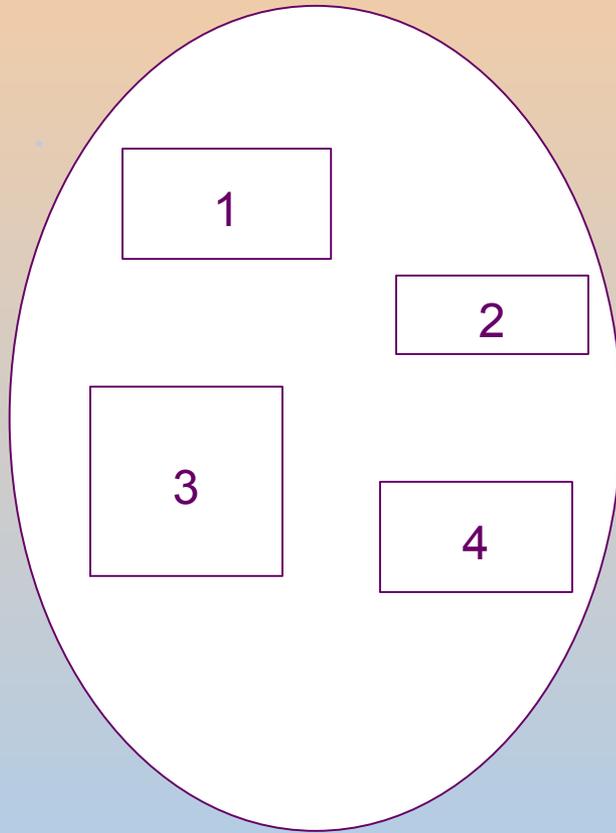
# Segmentação

- Esquema de gerenciamento de memória que suporta visão do usuário da memória.
- Um programa é uma coleção de segmentos. Um segmento é unidade lógica, tais como:
  - Programa principal,
  - procedimento,
  - função,
  - método,
  - objeto,
  - variáveis locais, variáveis globais,
  - blocos comuns,
  - pilha,
  - tabela de símbolos, vetores

# Visão de usuário de um programa



# Visão lógica da segmentação



Espaço do usuário



Espaço de memória física

# Arquitetura de segmentação

- Endereço lógico consiste de dois campos:
  - <número-segmento, deslocamento> ,
- *Tabela de segmentos* – mapeia endereços lógicos em endereços físicos; cada entrada da tabela contém
  - ◆ base – contém o endereço físico inicial onde os segmentos estão na memória.
  - ◆ *limite* – especifica o tamanho do segmento
- *Segment-table base register (STBR)* aponta para a localização da tabela de segmentos em memória.
- *Segment-table length register (STLR)* indica o número de segmentos usados por um programa
  - número de segmento  $s$  é válido se  $s < \text{STLR}$ .

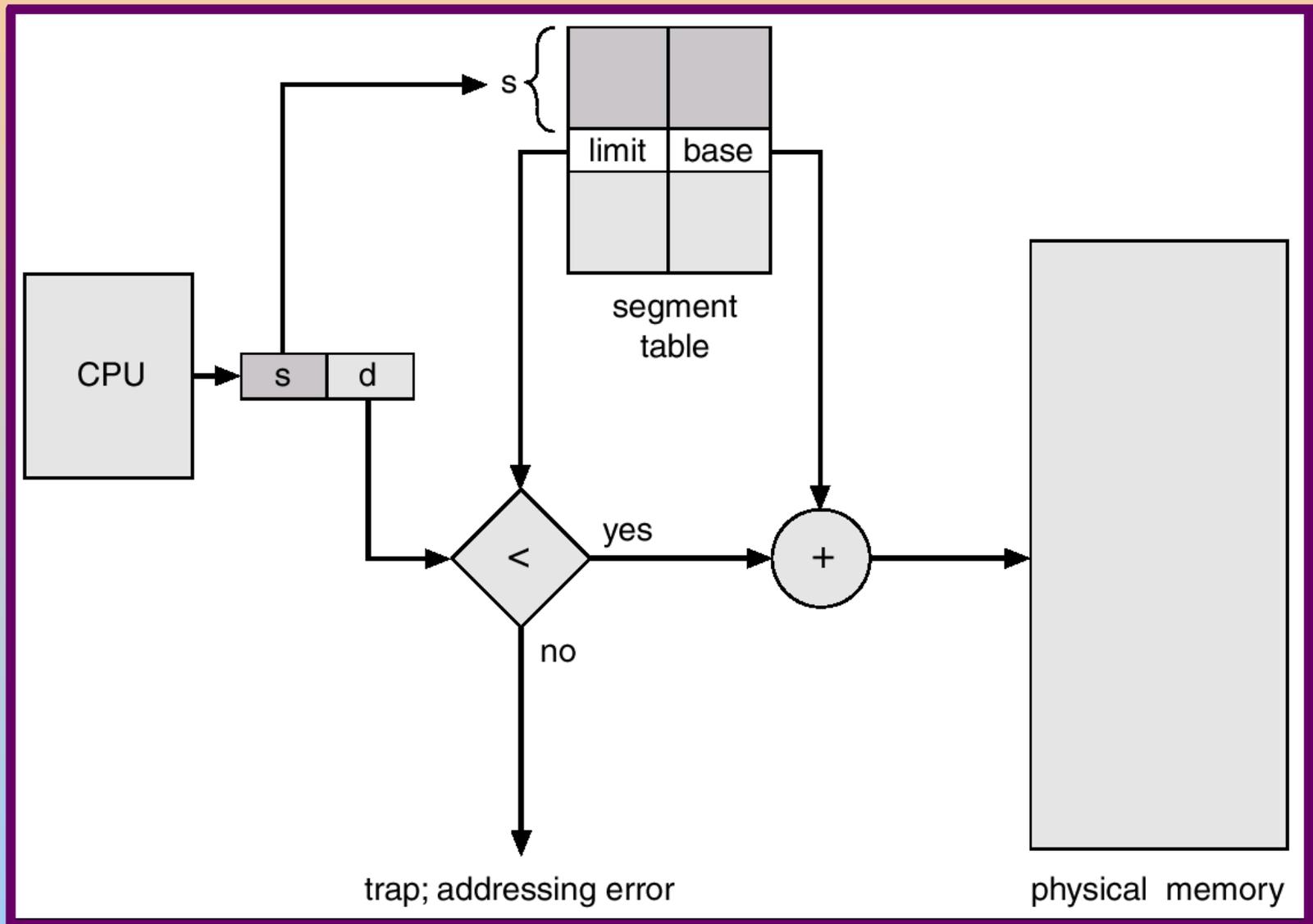
# Arquitetura de segmentação (Cont.)

- Relocação.
  - ◆ dinâmica
  - ◆ Pela tabela de segmentos
  
- Compartilhamento.
  - ◆ Segmentos compartilhados
  - ◆ Mesmo número de segmento
  
- Alocação.
  - ◆ first fit/best fit
  - ◆ Fragmentação externa

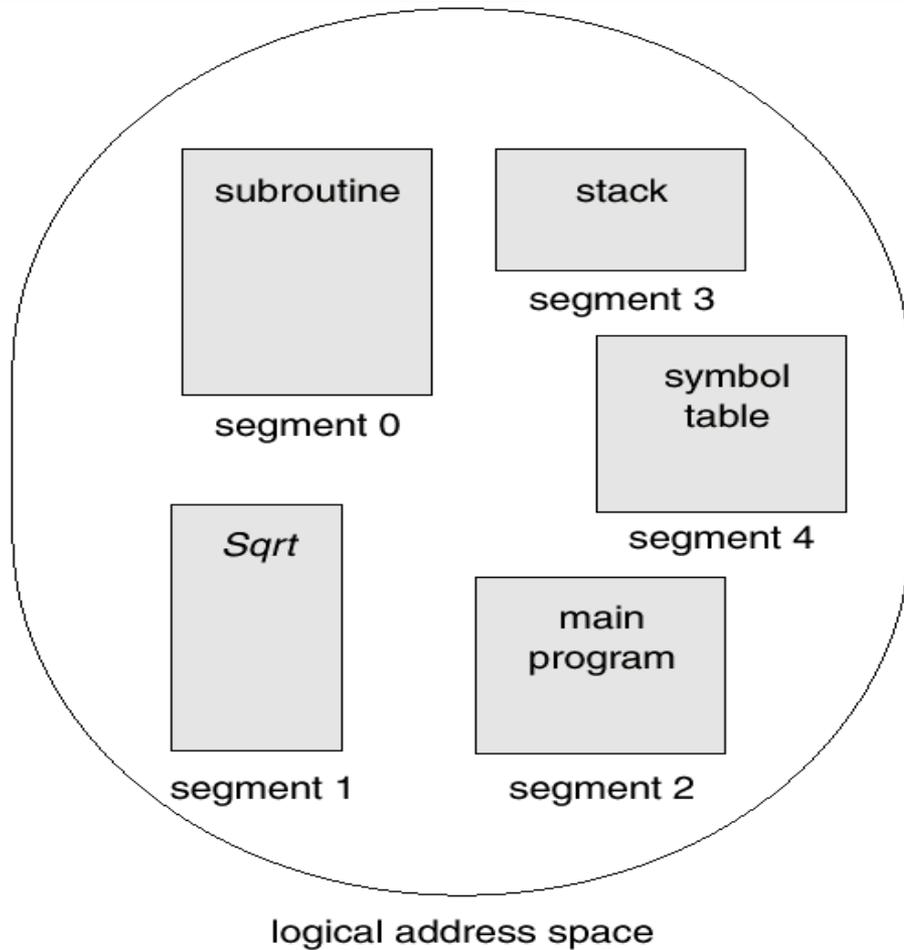
# Arquitetura de segmentação (Cont.)

- Proteção. Cada entrada da tabela de segmentos contém:
  - ◆ Bit da validação = 0  $\Rightarrow$  segmento ilegal
  - ◆ Privilégios de leitura/escrita/execução
- Possui bits de proteção associados com segmentos; compartilhamento de código ocorre no nível de segmento
- Como os segmentos variam de tamanho, alocação de memória é um problema de alocação dinâmica de recursos.

# Hardware de segmentação

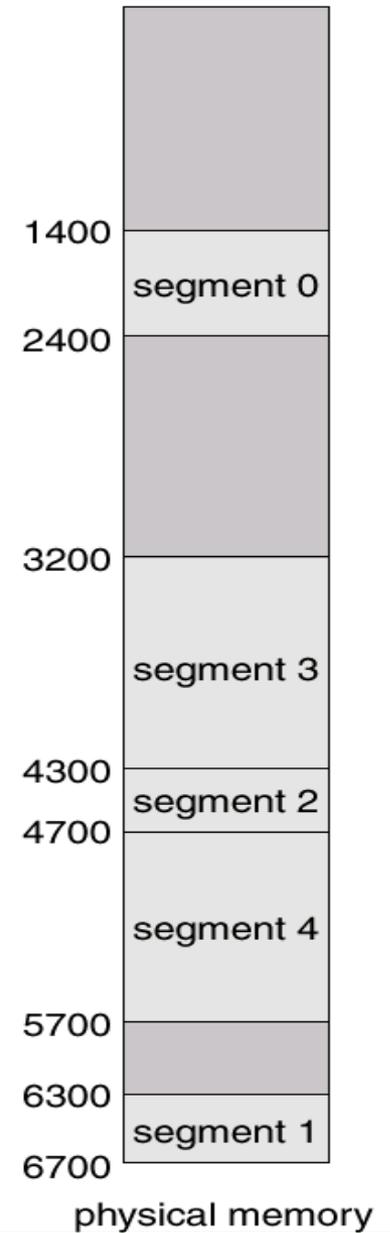


# Exemplo de segmentação

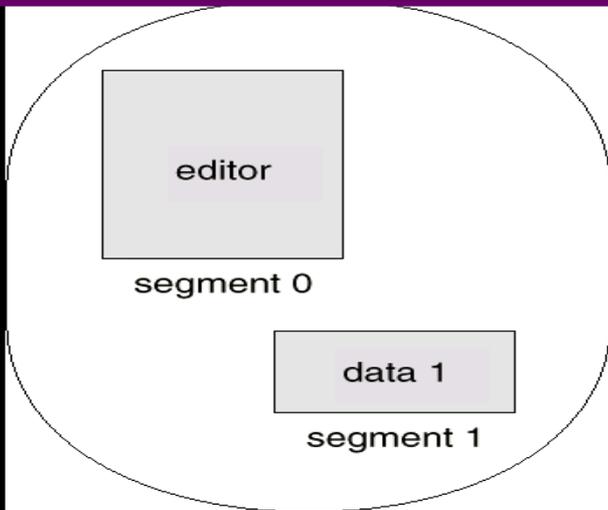


	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

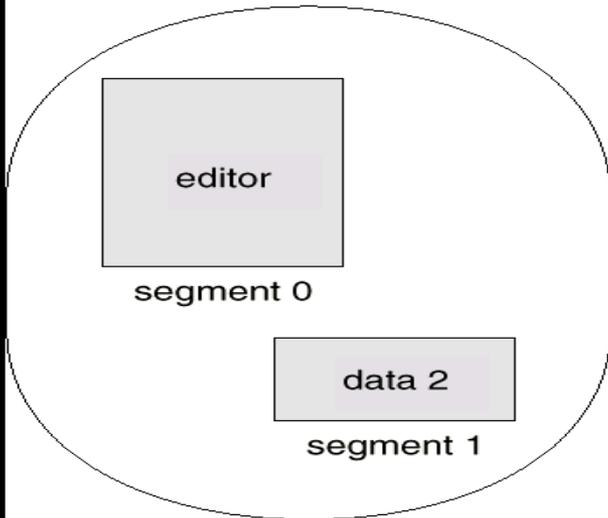
segment table



# Compartilhamento de segmentos



logical memory  
process  $P_1$



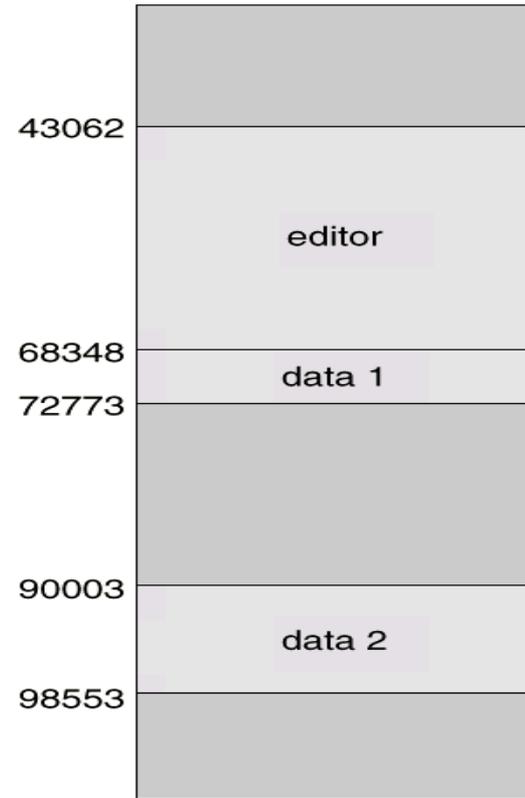
logical memory  
process  $P_2$

	limit	base
0	25286	43062
1	4425	68348

segment table  
process  $P_1$

	limit	base
0	25286	43062
1	8850	90003

segment table  
process  $P_2$

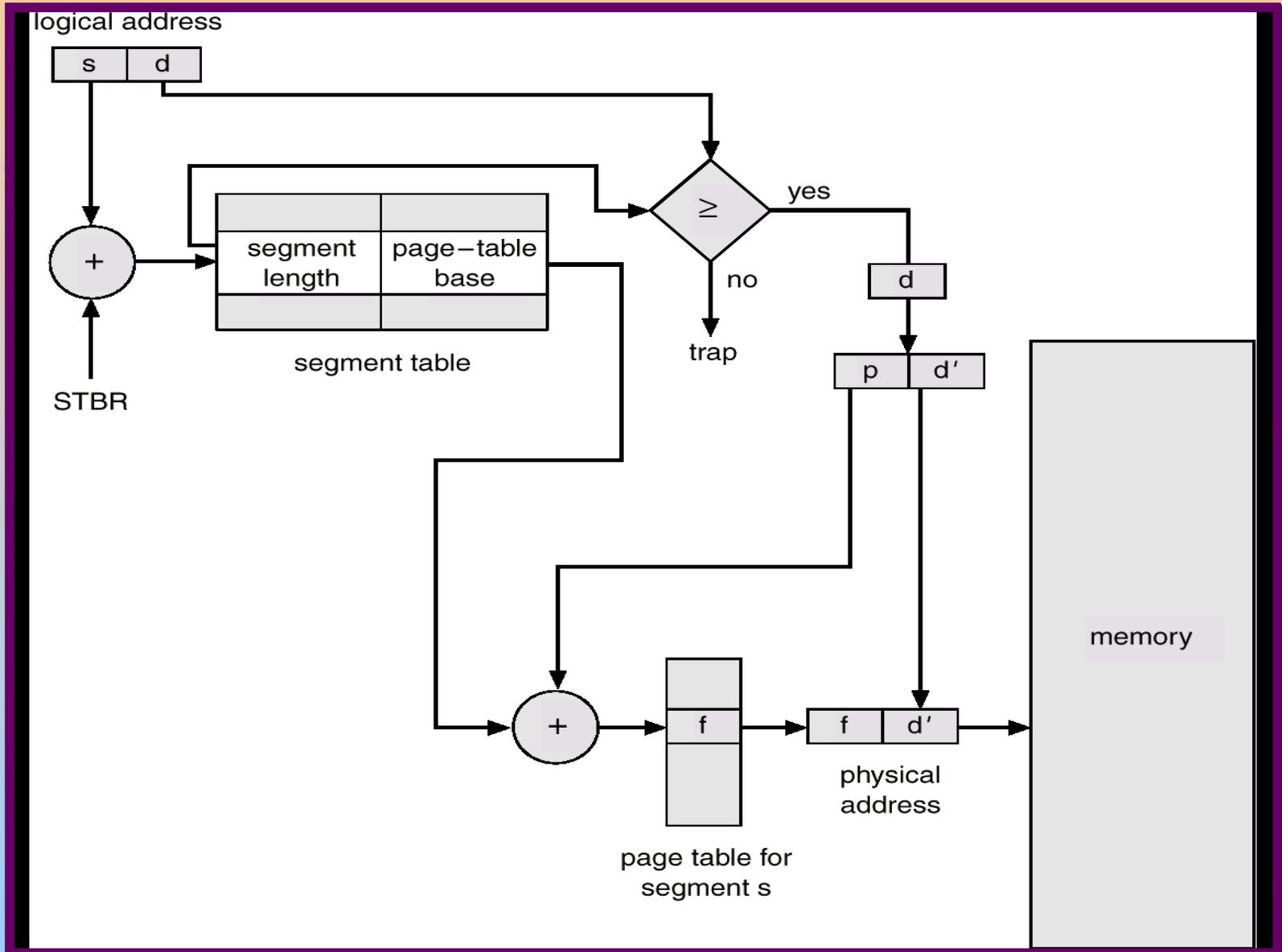


physical memory

# Segmentação com paginação – MULTICS

- O sistema MULTICS resolveu os problemas de fragmentação externa utilizando paginação de segmentos.
- Solução difere da segmentação pura no sentido em que a entrada da tabela de segmentos não contém mais o endereço do segmento, mas um endereço-base para a tabela de páginas deste segmento.

# Esquema de tradução de endereço do MULTICS



# Segmentação com paginação – Intel 386

- O processador Intel 386 usava segmentação com paginação para gerenciamento de memória com um esquema de paginação em dois níveis.

# Esquema de tradução de endereço Intel 386

