Capítulo 8: Gerenciamento de Memória

Sobre a apresentação (About the slides)

Os slides e figuras dessa apresentação foram criados por Silberschatz, Galvin e Gagne em 2005. Esse apresentação foi modificada por Cristiaino Costa (cac@unisinos.br). Basicamente, os slides originais foram traduzidos para o Português do Brasil.

É possível acessar os slides originais em http://www.os-book.com

Essa versão pode ser obtida em http://www.inf.unisinos.br/~cac

The slides and figures in this presentation are copyright Silberschatz, Galvin and Gagne, 2005. This presentation has been modified by Cristiano Costa (cac@unisinos.br). Basically it was translated to Brazilian Portuguese.

You can access the original slides at http://www.os-book.com

This version could be downloaded at http://www.inf.unisinos.br/~cac

Capítulo 8: Gerenciamento de Memória

- Fundamentos
- Troca de Processos (*Swapping*)
- Alocação Contígua
- Paginação
- Segmentação
- Segmentação com Paginação

Fundamentos

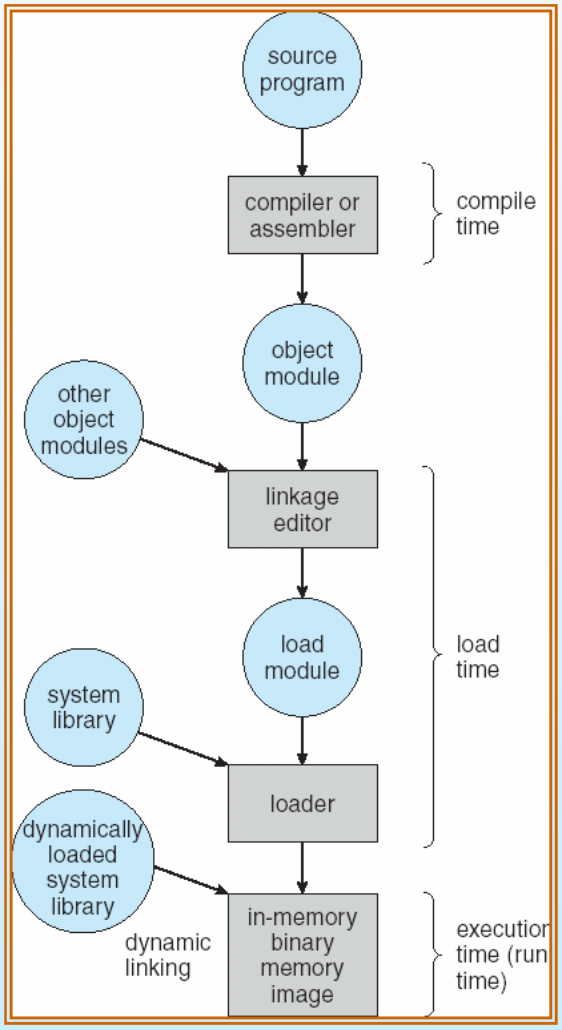
- Programa deve ser trazido para a memória e colocada dentro de um processo para ser executado.
- **Fila de Entrada** – coleção de processos no disco que estão esperando para ser colocados em memória para serem executados.
- Programas do usuário passa por vários passos antes de serem executados.

Atribuição de Instruções e Dados na Memória (*Binding*)

Atribuição de Endereços a instruções e dados pode ocorrer em três diferentes estágios.

- **Em Tempo de Compilação**: Se a posição de memória é conhecida a priori, *código absoluto* pode ser gerado; deve recompilar o código se posição inicial mudar.
- **Em Tempo de Carga**: Deve gerar *código relocável* se o endereço de memória não é conhecido em tempo de compilação.
- **Em Tempo de Execução**: Atribuição é feita somente em tempo de execução se o processo pode se mover durante sua execução de um segmento de memória para outro. Necessita de suporte de hardware para mapeamento de endereços (ex.: registradores *base* e *limite*).

Processamento em Múltiplos Passos de um Programa do Usuário



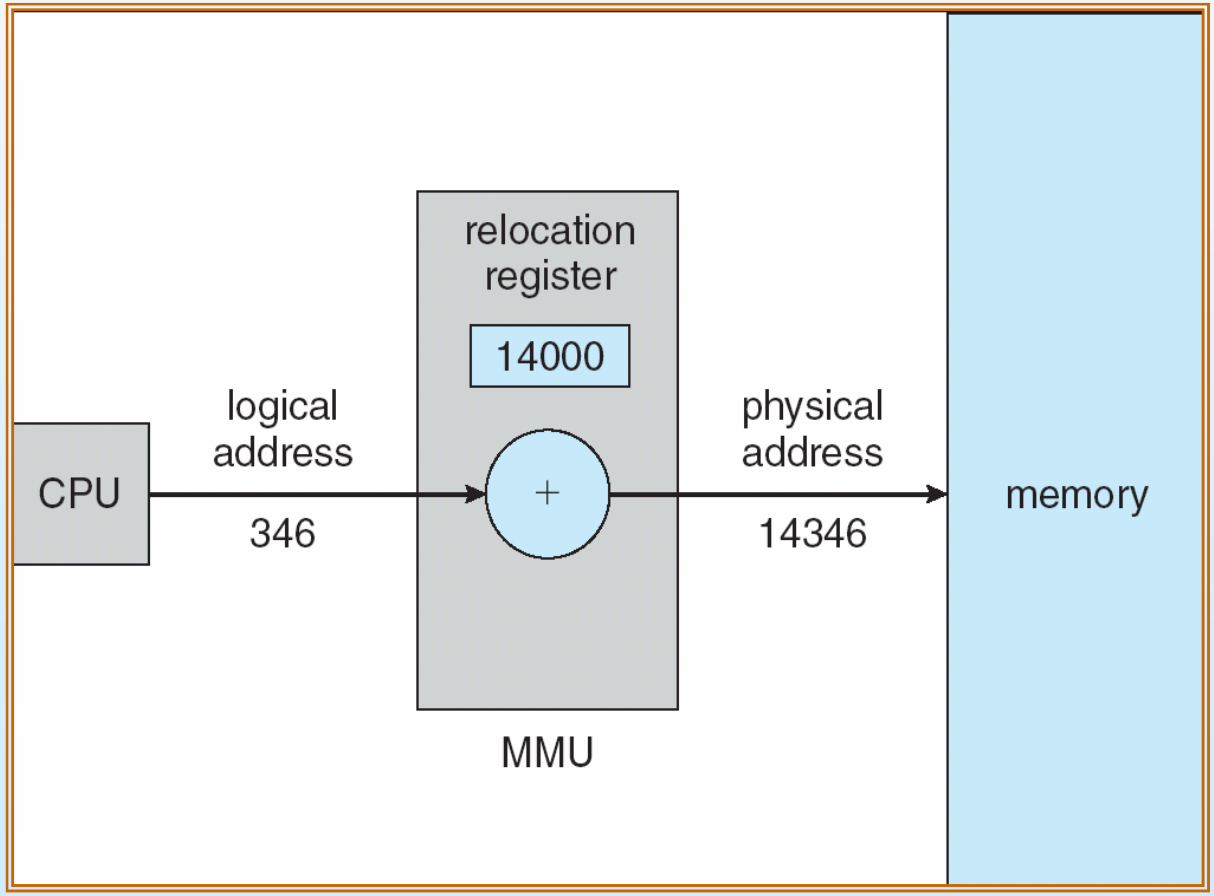
Espaço de Endereçamento Lógico vs. Físico

- O conceito de um *espaço de endereçamento lógico* que é atribuído a um *espaço de endereçamento físico* separado é central para um gerenciamento de memória apropriado.
  - **Endereço Lógico** – gerado pela CPU; também chamado de *endereço virtual*.
  - **Endereço Físico** – endereço visto pela unidade de memória.
- Os esquemas de atribuição de endereços em tempo de compilação e em tempo de carga usam endereços lógicos; endereços lógicos (virtuais) e físicos são diferentes em esquemas de atribuição de endereços em tempo de execução.

Unidade de Gerenciamento de Memória (MMU)

- UGM em inglês é *Memory Management Unit* (MMU)
- Dispositivo de Hardware que mapeia endereços virtuais para endereços físicos.
- No esquema do MMU, o valor no registrador relocador (ou base) é adicionado a cada endereço gerado pelo processo do usuário no momento que é enviado para a memória.
- O programa do usuário lida com endereços *lógicos*; ele nunca trata os endereços físicos *reais*

Relocação Dinâmica usando um registrador de relocação



Carga Dinâmica

- Rotina não é carregada até que seja chamada
- Melhor utilização do espaço de memória; rotinas não utilizadas nunca são carregadas.
- Útil quando uma grande quantidade de código é necessária para manipular casos com ocorrências infreqüentes.
- Nenhum suporte especial do sistema operacional é necessário. É responsabilidade do usuário projetar os programas de modo a tirar proveito deste esquema.

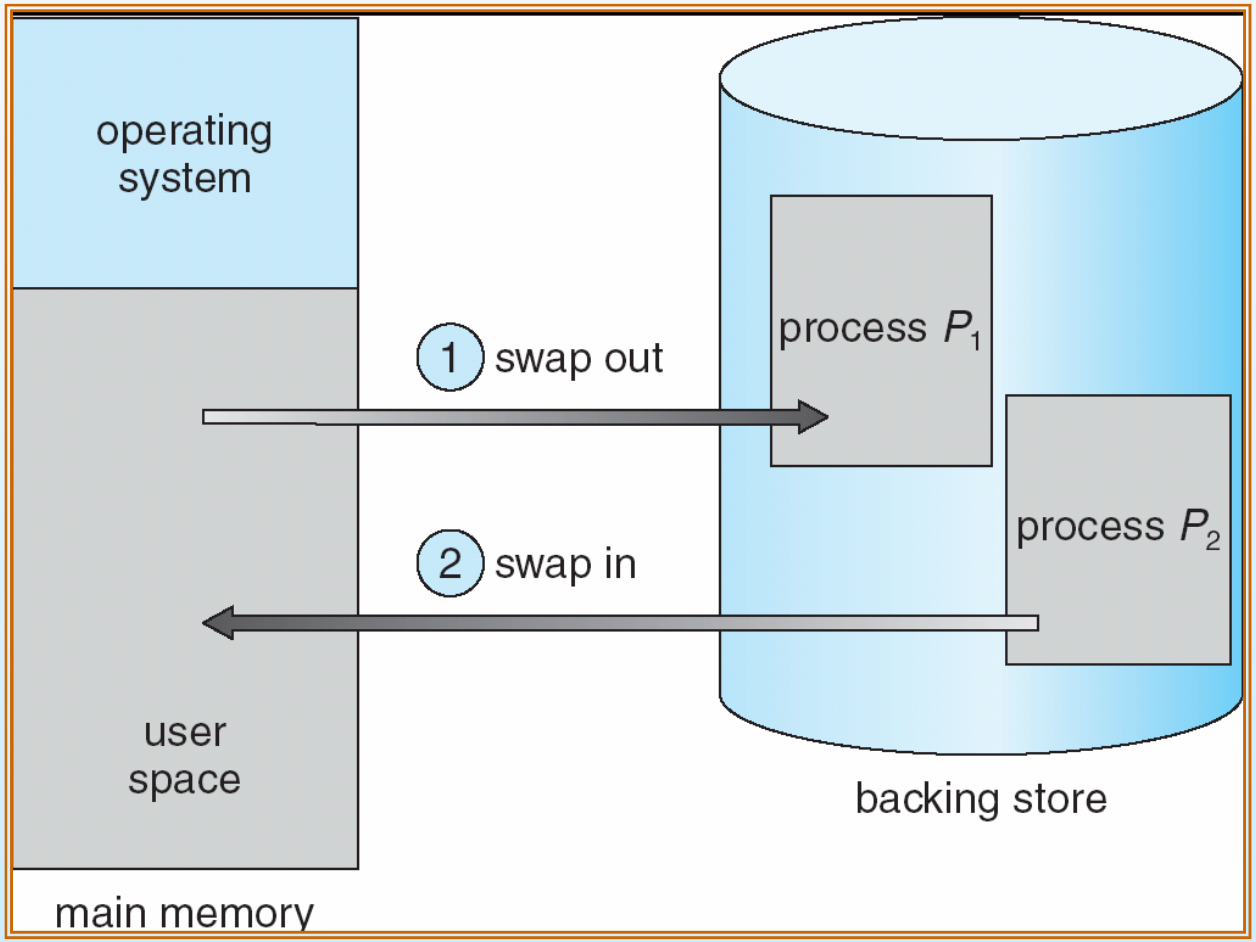
Ligação Dinâmica

- A Ligação (*linking*) é postergada até o momento da execução.
- Pequenas porções de código, *stub*, que indicam o local da memória onde está armazenada a rotina da biblioteca.
- S*tub* sobrepõe ela mesma com o endereço da rotina, a ser executada.
- Sistemas Operacionais devem detectar se a rotina está na área de endereçamento do processo.
- Ligação Dinâmica é particularmente útil para bibliotecas

Troca de Processos (*Swapping*)

- Um processo pode ser transferido (*swapped)* temporariamente da memória principal para uma memória secundária (*backing store)*, para depois ser transferido de volta à memória principal, a fim de que a execução do processo continue.
- **Memória Secundária** – disco rápido grande o suficiente para acomodar copias de todas as imagens da memória principal para todos os usuários; deve prover acesso direto as imagens da memória.
- ***Roll out, roll in*** – variação da política de troca de processos usada para algoritmos de escalonamento baseados em prioridade; Processo de baixa prioridade é transferido para a memória secundária para um processo de prioridade mais alta ser carregado e executado.
- Maior parte do tempo de troca de processos é tempo de transferência; tempo total de transferência é diretamente proporcional a quantidade de memória transferida.
- Versões modificadas de *swapping* são encontradas em muitos sistemas como UNIX, Linux e Microsoft Windows.

Visão Esquemática do *Swapping*

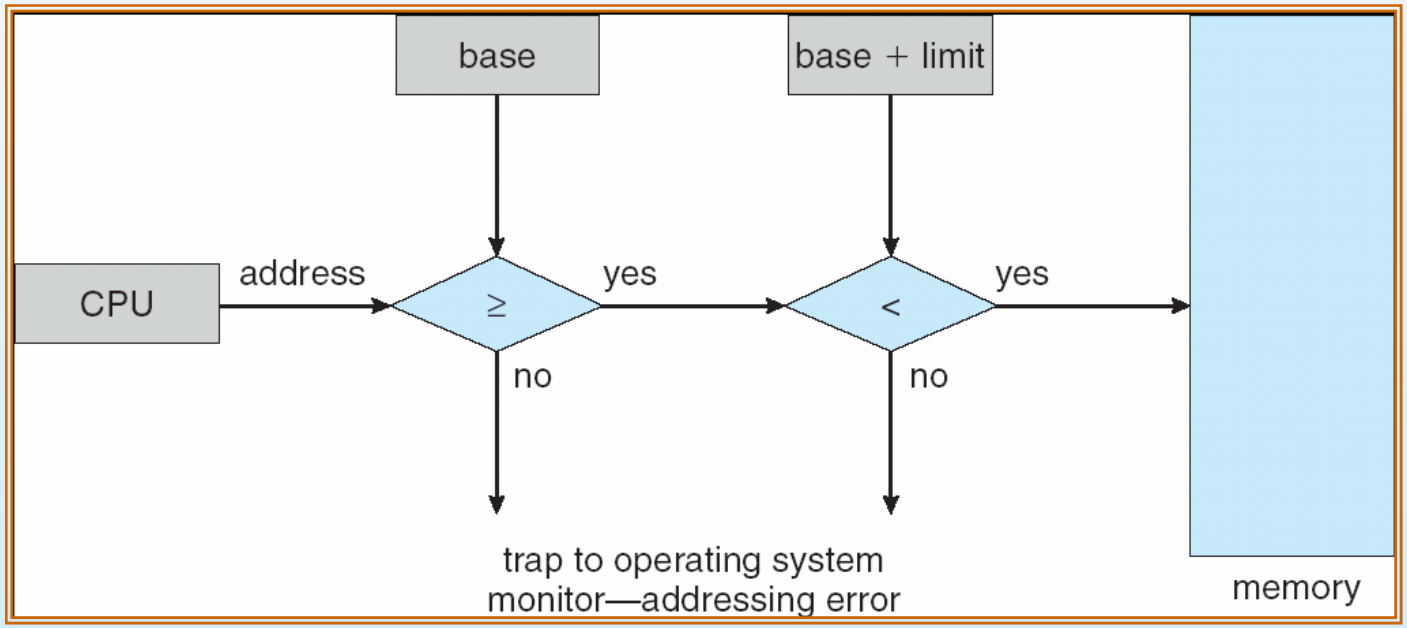


Alocação Contígua

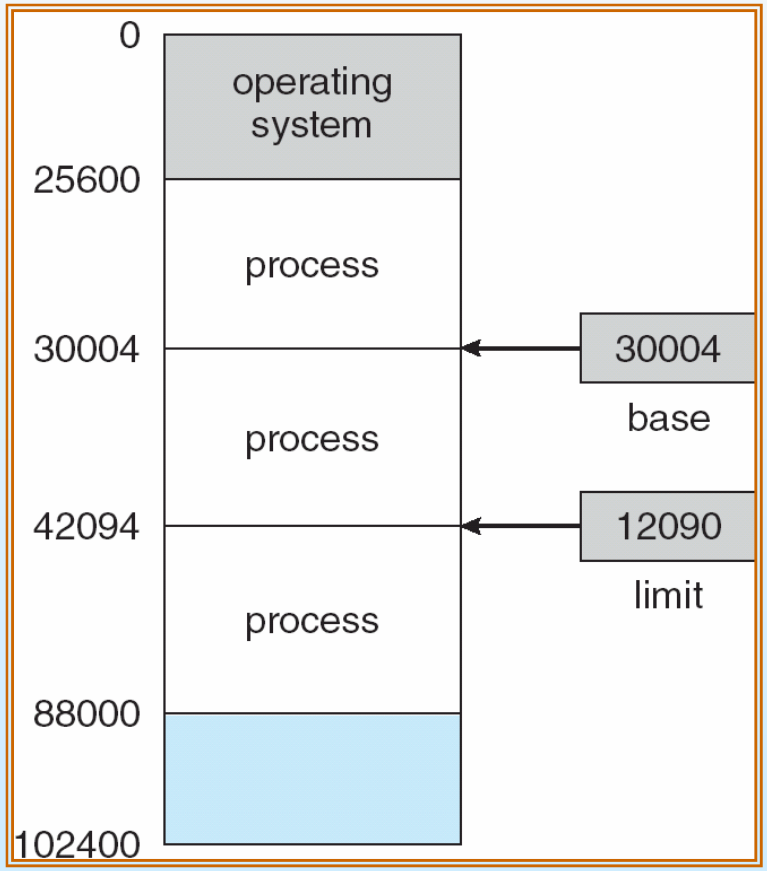
- A memória principal é normalmente divida em duas partes:
  - Parte residente do sistema operacional, normalmente mantida na parte baixa da memória com o vetor de interrupções.
  - Processos do usuário mantidos na parte alta da memória.

- Registradores de relocação são usados para proteger processos dos usuários uns dos outros, e de alterar os códigos e dados do sistema operacional.
  - Registrador *base* contém o valor do menor endereço físico;
  - Registrador *limite* contém o tamanho do intervalo dos endereços lógicos – cada endereço lógico deve ser menor que o registrador limite.

Proteção de endereços por Hardware com registradores base e limite

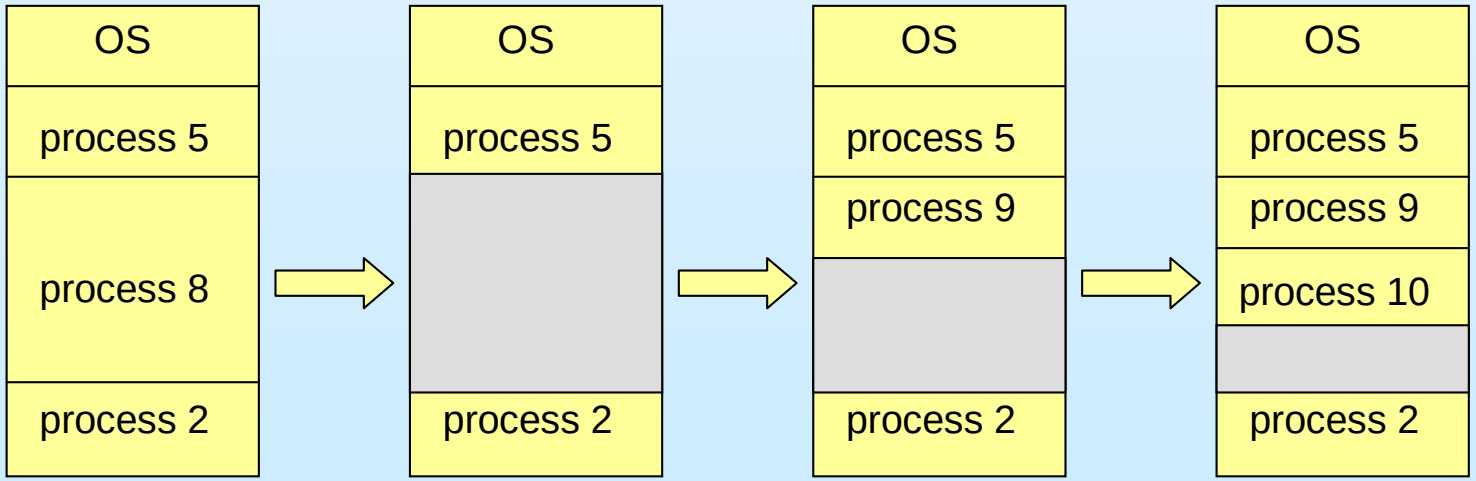


Os registradores base e limite definem um espaço de endereçamento lógico



Alocação Contígua (Cont.)

- Alocação com Diversas Partições
  - *Bloco Livre (Hole)* – bloco de memória disponível; blocos de vários tamanhos são espalhados pela memória.
  - Quando um processo chega, é alocada memória de um bloco livre grande o suficiente para acomodá-lo.
  - Sistema Operacional mantém informações sobre:
    a) partições alocadas b) partições livres (*holes*)



Armazenamento Dinâmico-Problema de Alocação

Como satisfazer uma requisição de tamanho *n* com uma lista de blocos livres.

- ***First-fit* (Primeira)**: Aloca o primeiro bloco livre que seja grande o suficiente para satisfazer a requisição.
- ***Best-fit* (Melhor)**: Aloca o menor bloco livre que seja grande o suficiente; deve procurar na lista inteira, a menos que esta esteja ordenada por tamanho. Produz de sobra o menor bloco livre.
- ***Worst-fit* (Pior)**: Aloca o maior bloco livre; deve também procurar na lista inteira. Produz de sobra o maior bloco livre.

First-fit e *best-fit* são melhores do que *worst-fit* em termos de velocidade e utilização de armazenamento

Fragmentação

- **Fragmentação Externa** – espaço de memória total existe para satisfazer uma requisição, mas é não contíguo.
- **Fragmentação Interna** – memória alocada pode ser ligeiramente maior que a memória requerida; esta diferença de tamanho é na memória interna a partição, que não está sendo utilizada.
- Fragmentação externa é reduzida com **compactação**
  - Deslocar os blocos de memória de maneira a colocá-los juntos em um grande bloco.
  - Compactação é possível *somente* se relocação é dinâmica, e é feita em tempo de execução.
  - Problemas de E/S
    - Trancamento de *jobs* na memória enquanto ele está envolvido em E/S.
    - Realizar E/S somente em *buffers* do SO.

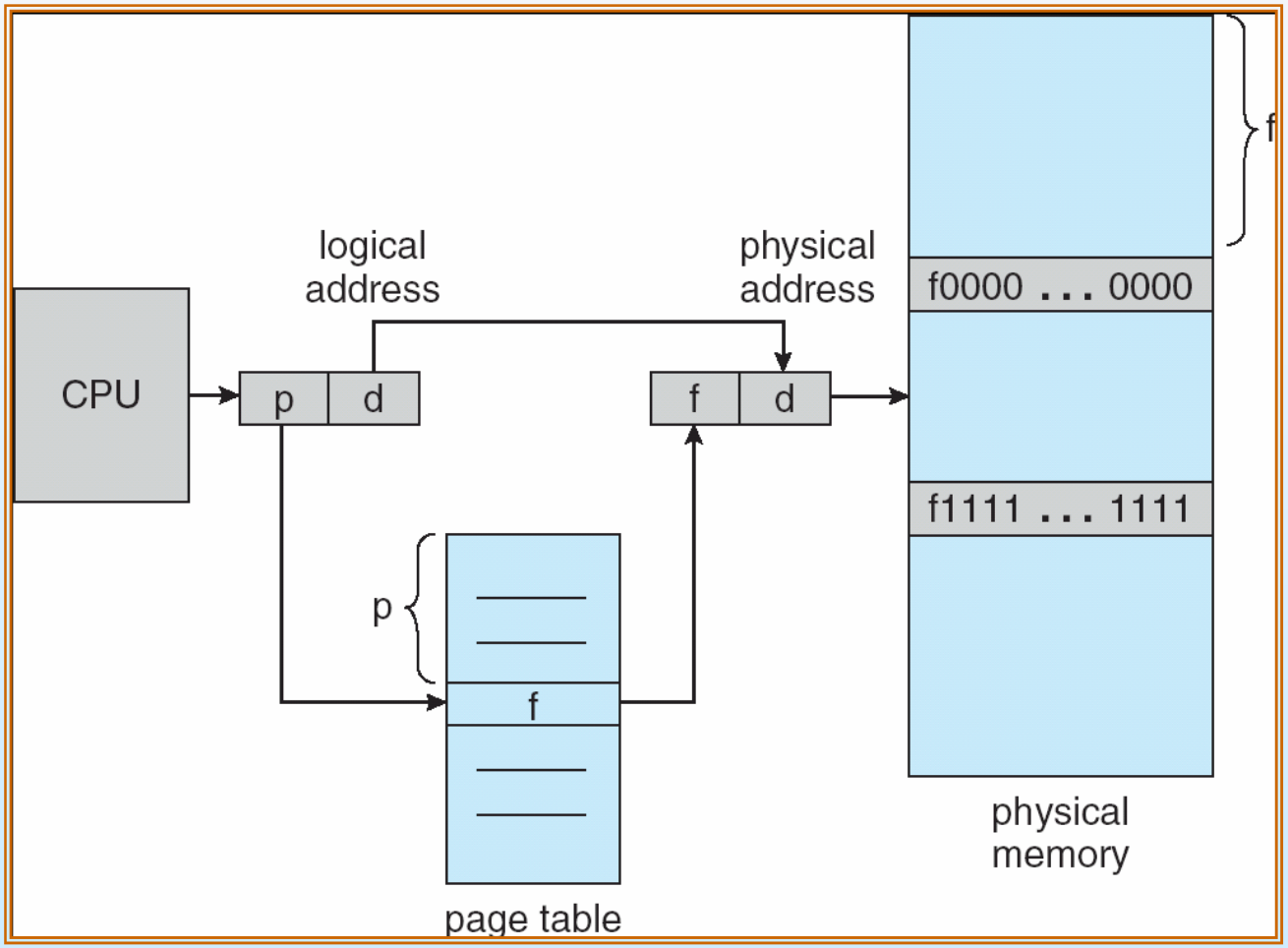
Paginação

- Espaço de endereçamento Lógico de um processo pode ser não contíguo; processo é alocado para a memória física sempre que existir espaço disponível
- Divide a memória física em partes de tamanho fixo chamadas de **blocos** (*frames*) (tamanho é potência de 2, entre 512 bytes e 8.192 bytes)
- Divide memória lógica em partes do mesmo tamanho chamadas de **páginas**
- Mantém controle de todos os *blocos* livres
- Para executar um programa com *n* páginas, necessita encontrar *n* blocos livres e carregar o programa
- Alterar uma *tabela de páginas* para traduzir endereços lógicos em físicos
- Fragmentação Interna

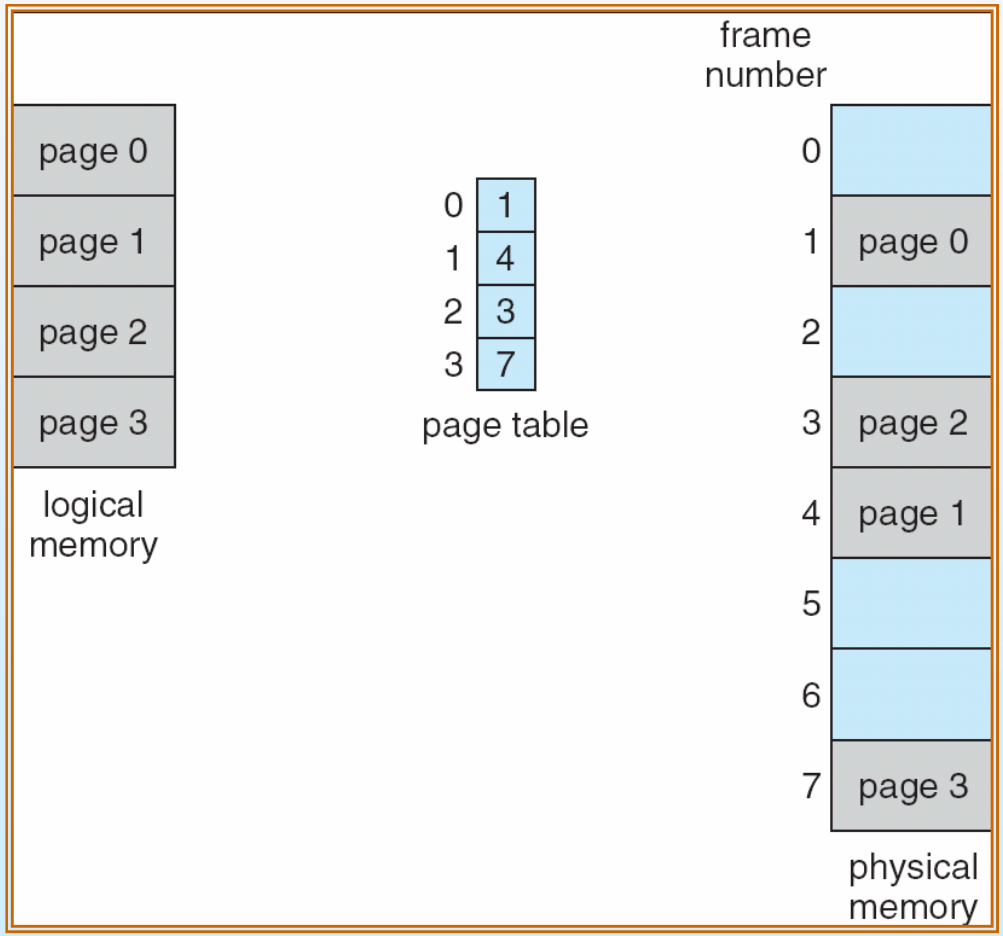
Esquema de Tradução de Endereços

- Endereço gerado pela CPU é divido em:
  - *Número da Página* (***p***) – usada como um índice em uma tabela de páginas que contém o endereço base de cada página na memória física
  - *Deslocamento na Página* (***d***) – combinado com o endereço base para definir o endereço de memória que é enviado a unidade de memória

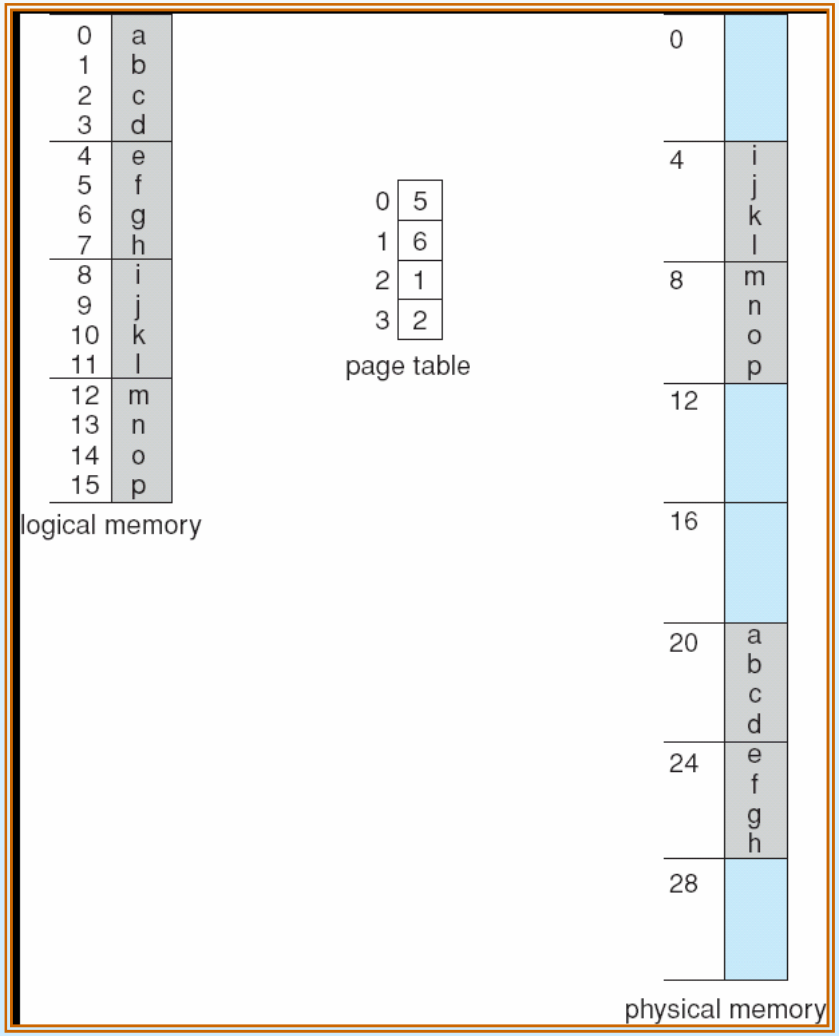
Arquitetura para Tradução de Endereços



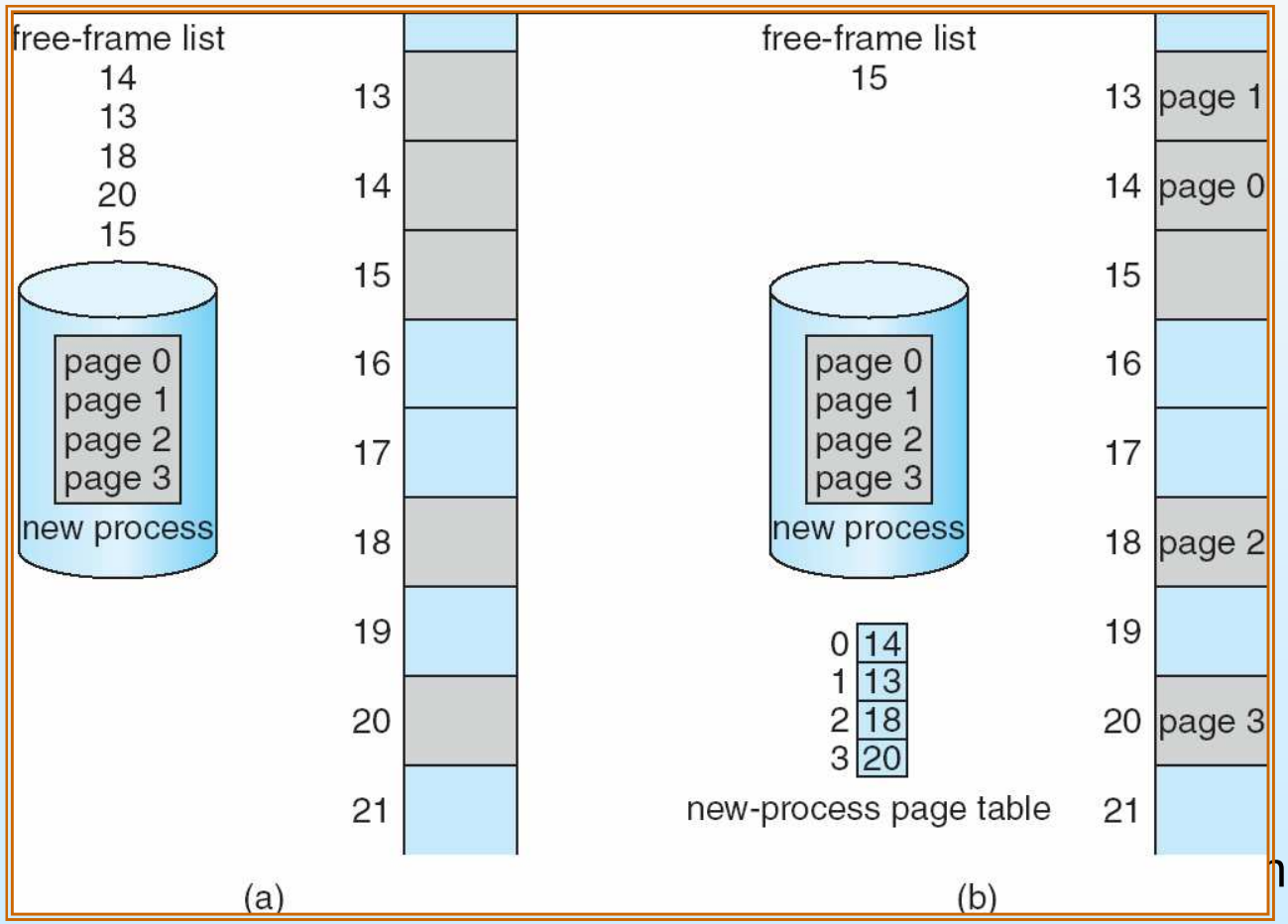
Exemplo de Paginação



Exemplo de Paginação



Blocos Livres



Implementação de Tabela de Páginas

- Tabela de Páginas é mantida na memória principal.
- *Registrador base da tabela de páginas (Page-table base register* - PTBR) aponta para a tabela de páginas
- *Registrador tamanho da tabela de páginas* (*Page-table length register* - PRLR) indica quantos endereços ela ocupa
- Neste esquema cada acesso a dado/instrução requer dois acessos a memória. Um para a tabela de páginas e outro para o dado/instrução
- O problema pode ser resolvido com o uso de uma memória *cache* especial, pequena, de acesso rápido, chamada de **memória associativa** ou ***translation look-aside buffers*** (**TLBs**)

Memória Associativa

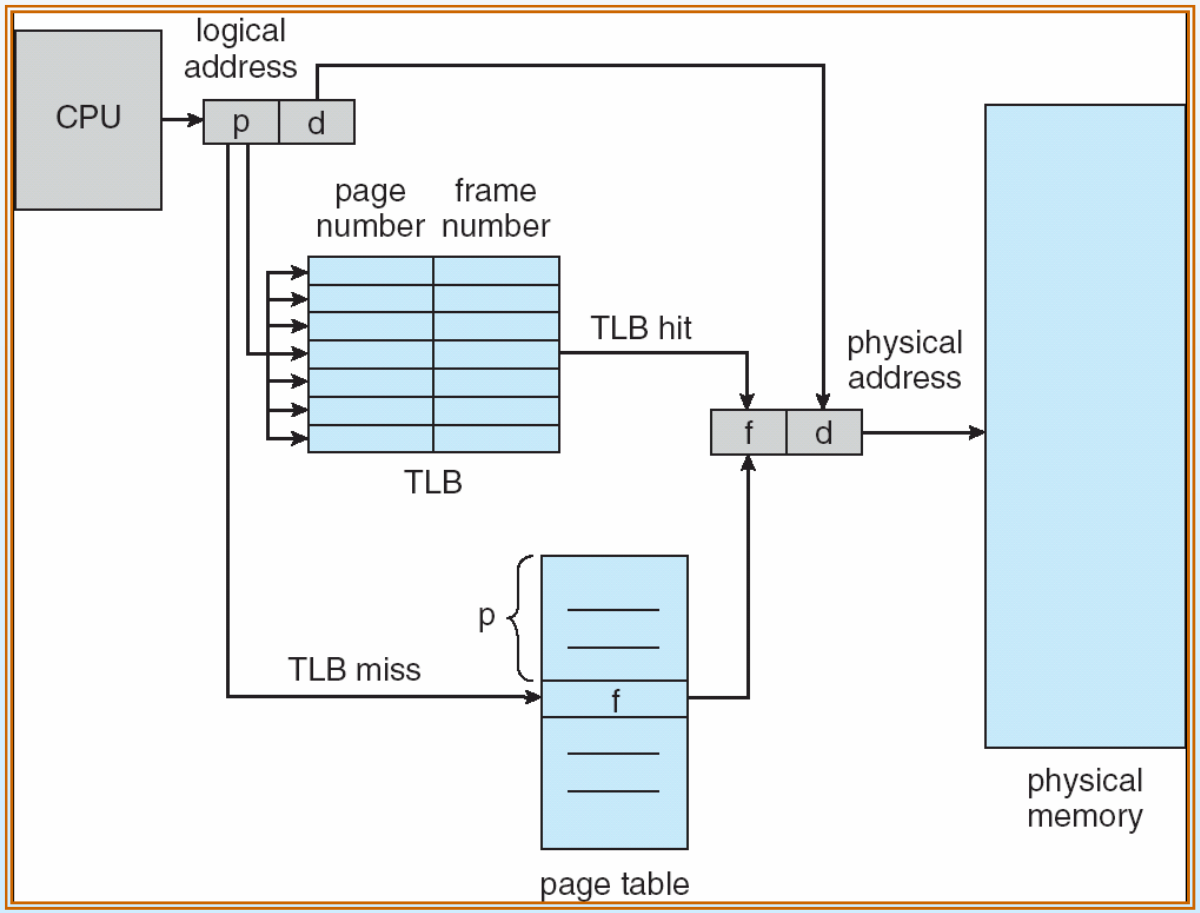
- Memória Associativa – busca em paralelo

nº da Página	nº do Bloco

Tradução de Endereços (p, d)

- Se p está em um registrador associativo, obtém o nº do bloco diretamente
- Em caso contrário obtém o nº do bloco da tabela de páginas na memória

Hardware de Paginação com TLB



Tempo de Acesso Efetivo

- Busca Associativa = ε unidades de tempo
- Assume que o tempo de ciclo de memória é 1 microssegundo
- Taxa de Sucesso (*Hit ratio*) – porcentagem de vezes que um número de página é encontrado nos registradores associativos; razão é relacionada com o número de registradores associativos.
- Taxa de Sucesso = α
- **Tempo Médio de Acesso** ou ***Effective Access Time*** (EAT)

$$EAT = (1 + \varepsilon)\,\alpha + (2 + \varepsilon)(1 - \alpha)$$

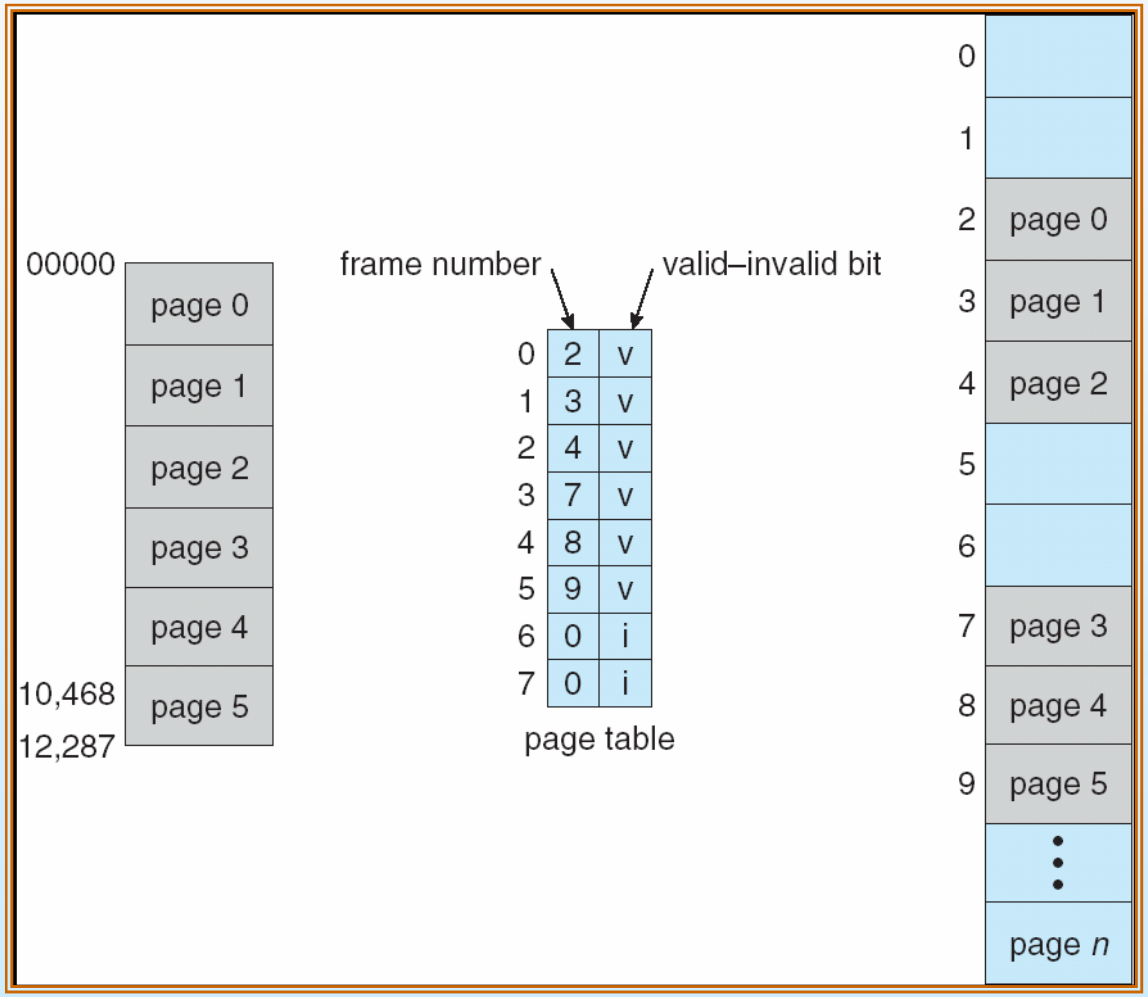
$$= 2 + \varepsilon - \alpha$$

Proteção de Memória

- Proteção de Memória implementada através de bits de proteção associados a cada bloco.

- Bit ***válido-inválido*** associado para cada entrada na tabela de páginas:
  - "válido" indica que a página associada está no espaço de endereçamento lógico do processo, e portanto é o acesso é legal.
  - "inválido" indica que a página não está no espaço de endereçamento lógico do processo.

Bit Valido (v) ou Invalido (i) em uma Tabela de Páginas

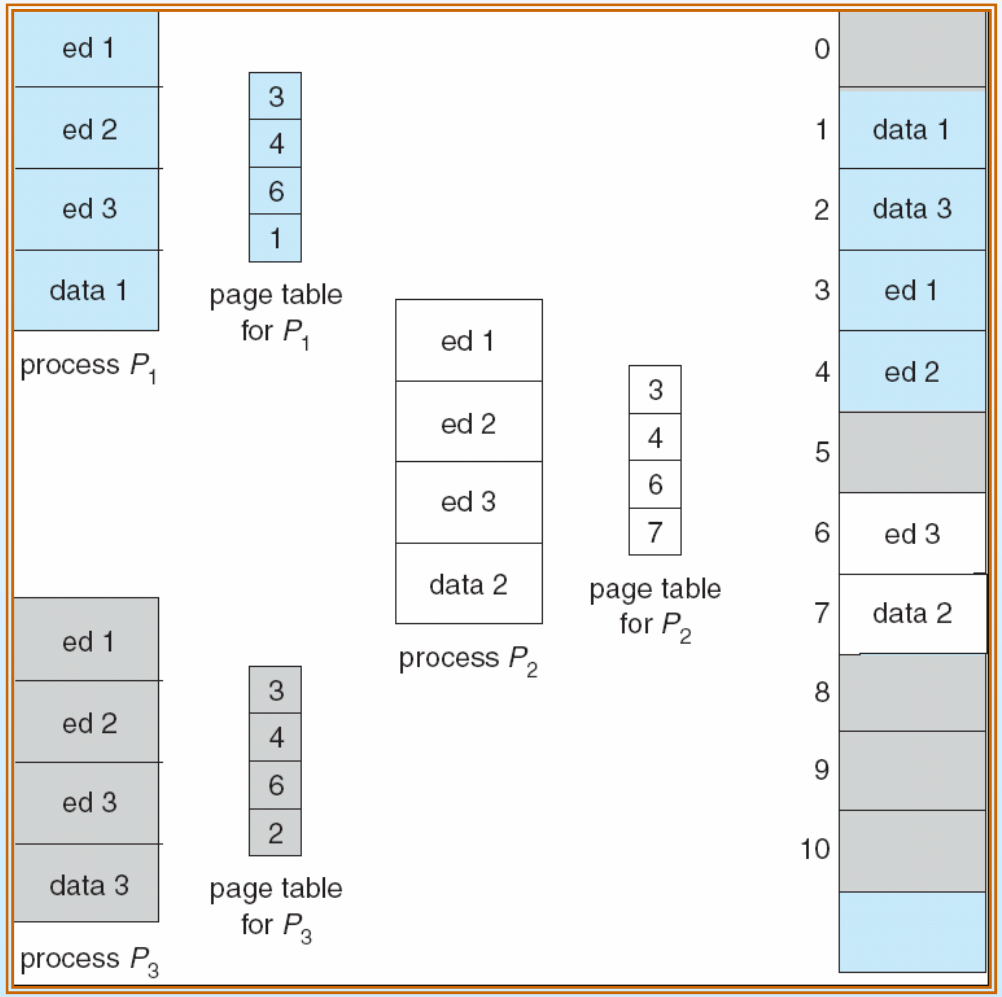


Páginas Compartilhadas

- **Compartilhamento de Código**
  - Uma cópia de código somente leitura (reentrante) compartilhada entre processos (ex.: editores de texto, compiladores, sistemas de janelas)
  - Código compartilhado deve aparecer na mesma localização no espaço de endereçamento lógico de todos processos

- **Códigos e Dados privados**
  - Cada processo mantém uma cópia separada de códigos e dados
  - As páginas para códigos e dados privados podem aparecer em qualquer endereço no espaço de endereçamento lógico

Exemplo de Páginas Compartilhadas



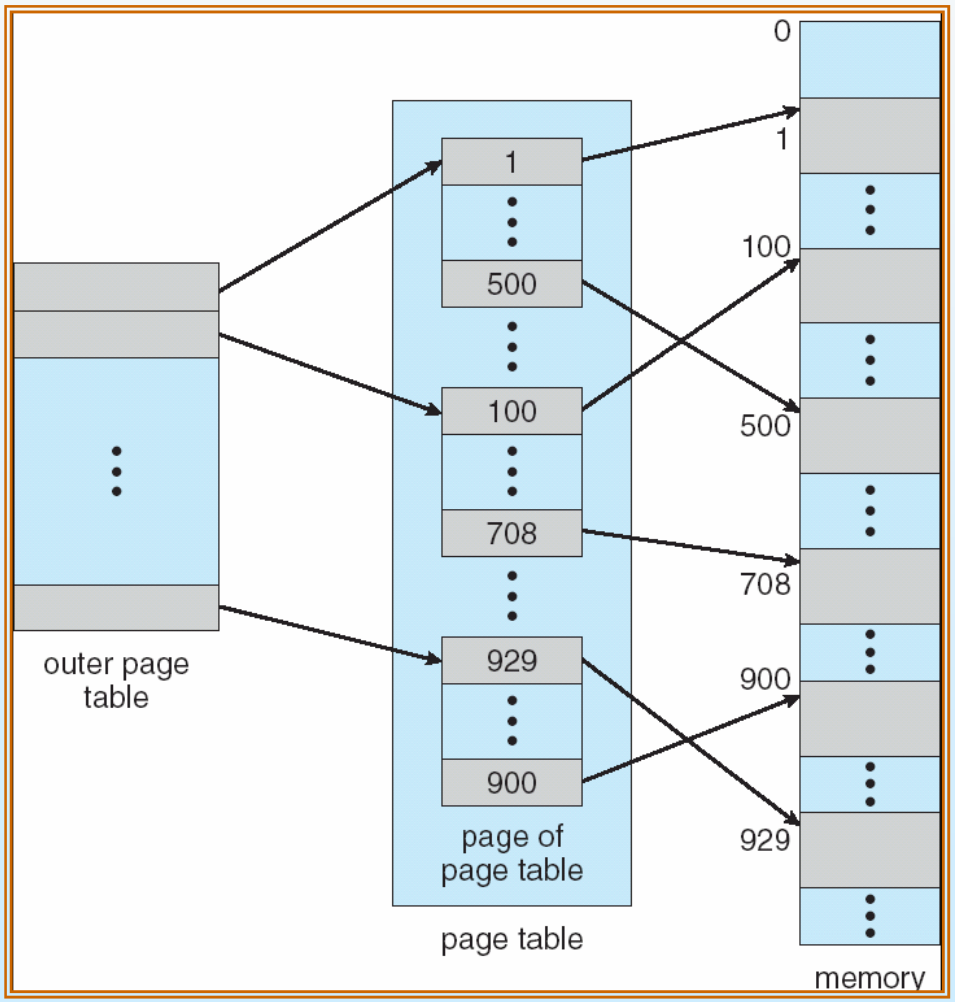
Estrutura da Tabela de Páginas

- Tabelas de Páginas Hierárquicas
- Tabela de Páginas com função *Hash* (*Hashed Page Tables*)
- Tabela de Página Invertida

Tabelas de Páginas Hierárquicas

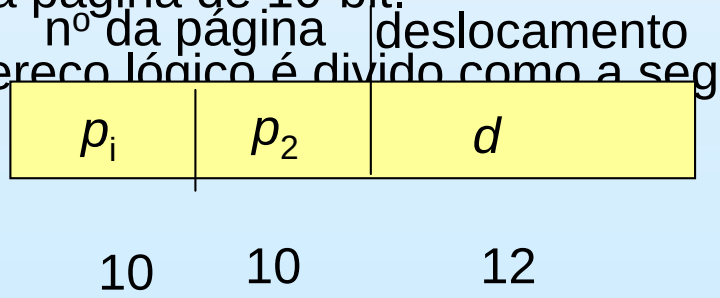
- Quebrar o espaço de endereço lógico em múltiplas tabelas de páginas
- Uma técnica simples é tabela de páginas em dois níveis

Tabela de páginas em dois níveis



Exempo de Paginação em dois níveis

- Um endereço lógico (em máquinas de 32-bit com tamanho de páginas 4K) é divido em:
  - um número de páginas de 20 bits.
  - um deslocamento na página de 12 bits.
- Uma vez que a tabela de páginas é paginada, o número da página é dividido em:
  - um número de página de 10-bit.
  - uma posição na página de 10-bit.
- Portanto, um endereço lógico é divido como a seguir:



Esquema de Tradução de Endereços

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

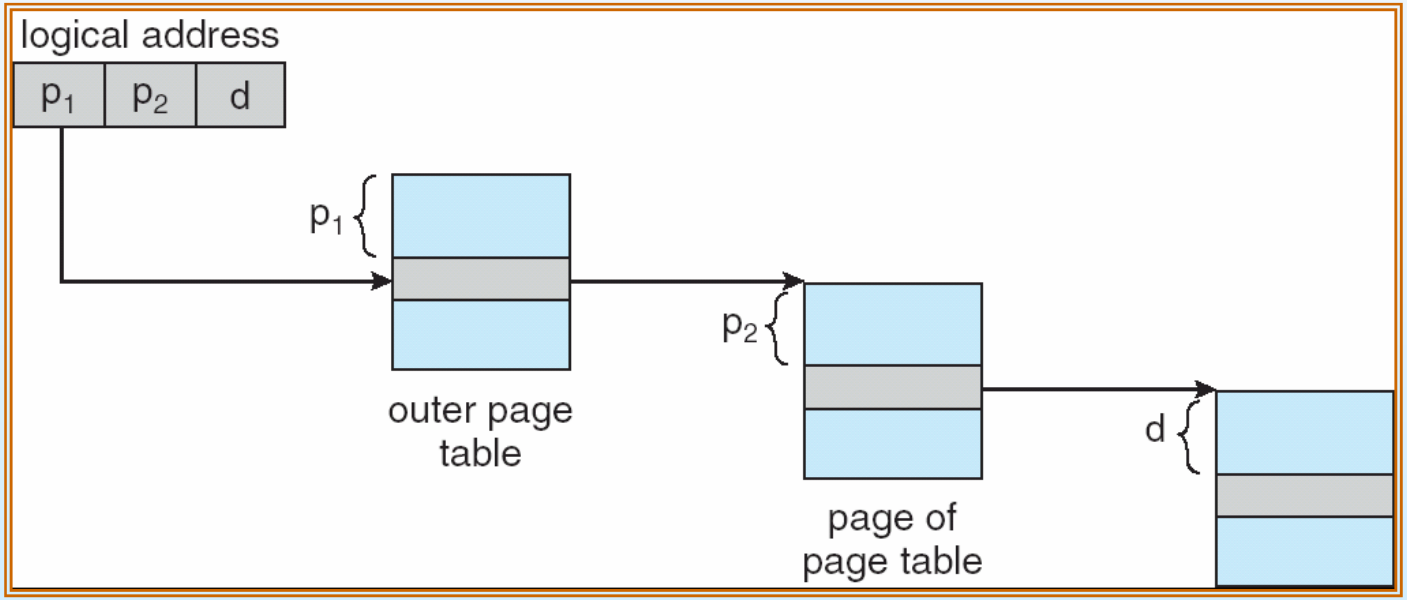


Tabela de Páginas com função *Hash*

- Comum em espaços de endereçamentos > 32 bits
- Ao número da página virtual é aplicada uma função *hash* que gera a localização na tabela de páginas. Em cada posição da tabela de páginas pode existir um encadeamento de elementos cuja função *hash* gera a mesma localização.
- Números de página virtual são comparados nesse encadeamento procurando por endereço igual. Se é encontrado, o bloco físico correspondente é obtido.

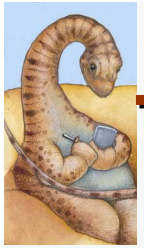


Tabela de Páginas com função *Hash* (Cont.)

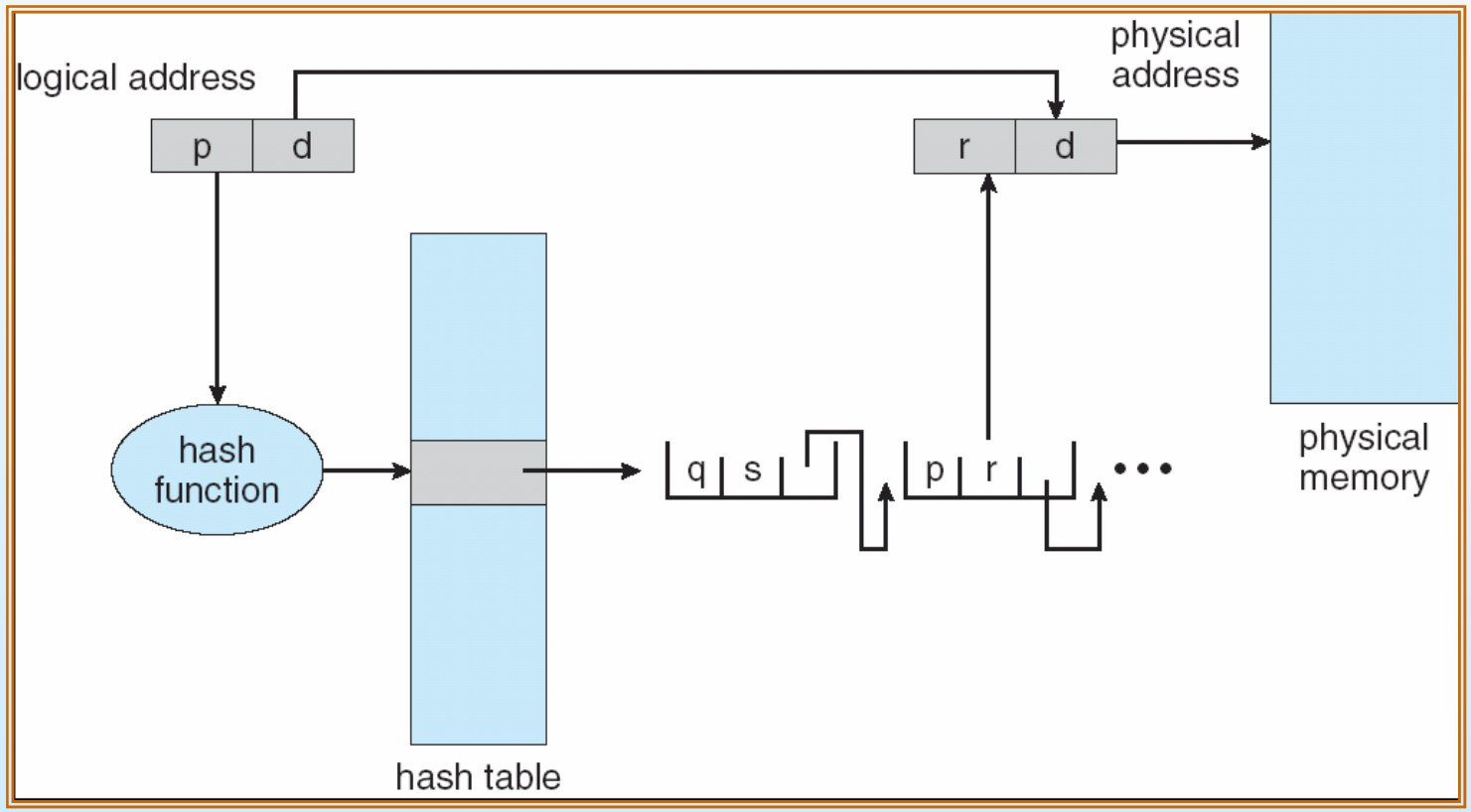
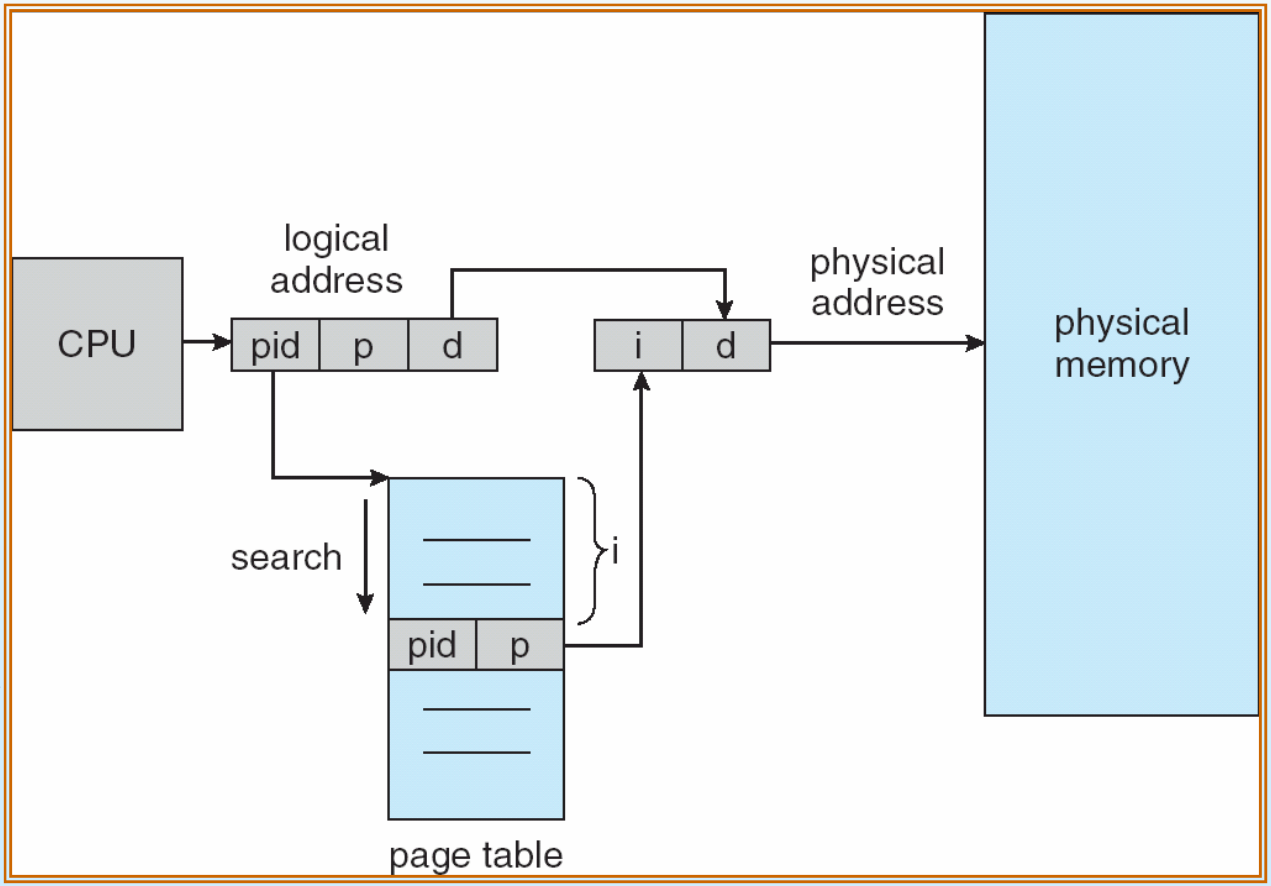


Tabela de Página Invertida

- Uma entrada para cada página real (*bloco*) de memória
- Cada entrada contém o endereço virtual da página armazenada naquele bloco da memória, com informações sobre o processo do qual essa página faz parte
- Diminui a quantidade de memória necessária para armazenar cada tabela de páginas, mas aumenta o tempo de pesquisa na tabela em cada referência a uma página
- Uso de função *hash* para limitar a pesquisa a apenas uma — ou no máximo a algumas — entradas na tabela de páginas

Arquitetura de Tabela de Página Invertida



Segmentação

- Esquemas de gerenciamento de memória que suportam a visão do usuário da memória

- Um programa é uma coleção de segmentos. Um segmento é uma unidade lógica, como por exemplo:

  programa principal,

  procedimento,

  função,

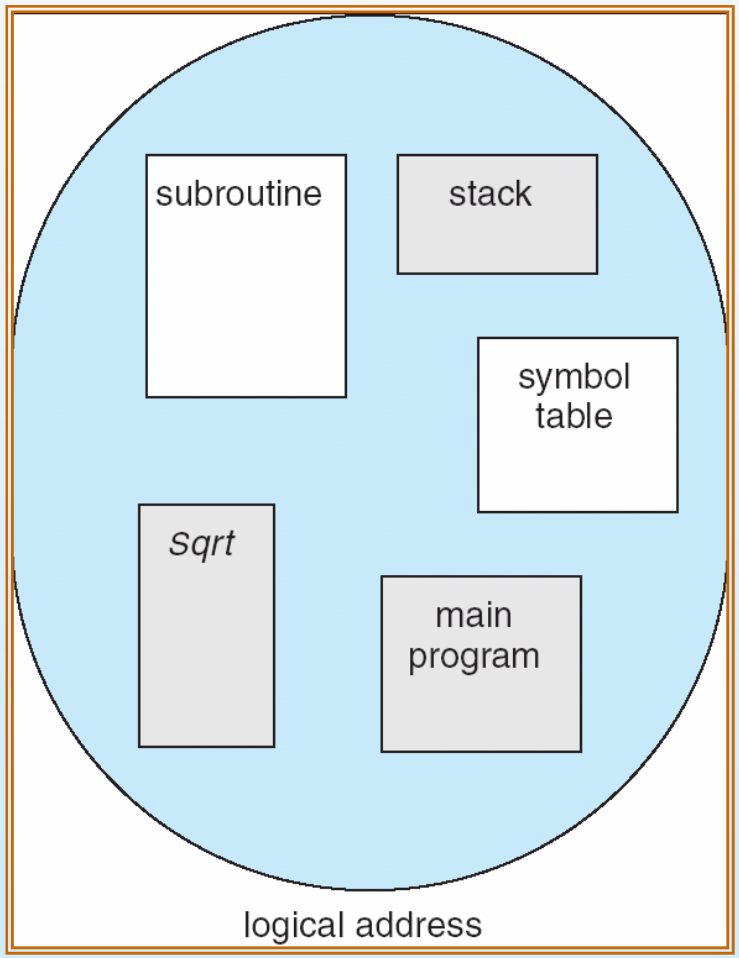
  variáveis locais, variáveis globais,

  bloco comum,

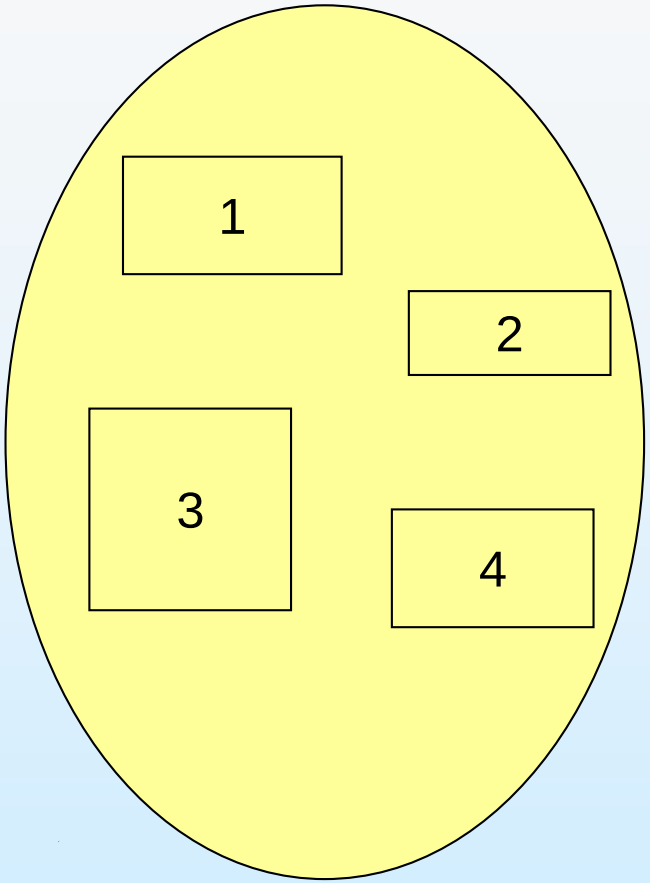
  pilha,

  tabela de símbolos, vetores

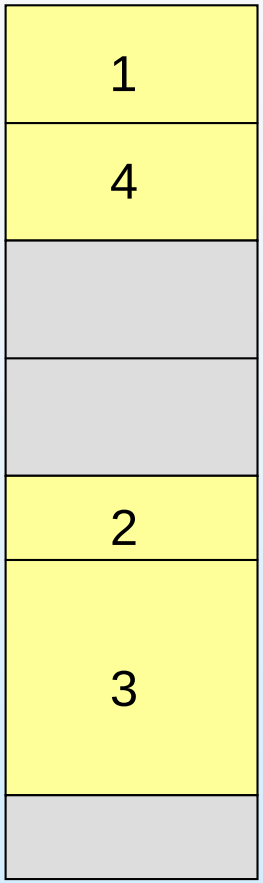
Visão do Usuário de um Programa



Visão Lógica da Segmentação



user space



physical memory space

Arquitetura da Segmentação

- Endereço lógico consiste de duas partes:

  < número do segmento, posição nesse segmento >,

- **Tabela de Segmentos** – mapeia endereços físicos bi-dimensionais; cada entrada na tabela possui:
  - **base** – contém o endereço físico inicial no qual o segmento reside na memória
  - **limite** – especifica o tamanho do segmento
- Registrador Base da Tabela de Segmentos ou *Segment-table base register (***STBR***)* aponto para a localização da tabela de segmentos na memória
- Registrador de tamanho da tabela de segmentos ou *Segment-table length register (***STLR***)* indica o número de segmentos usados por um programa

  número de segmento ***s*** é legal se ***s*** < **STLR**

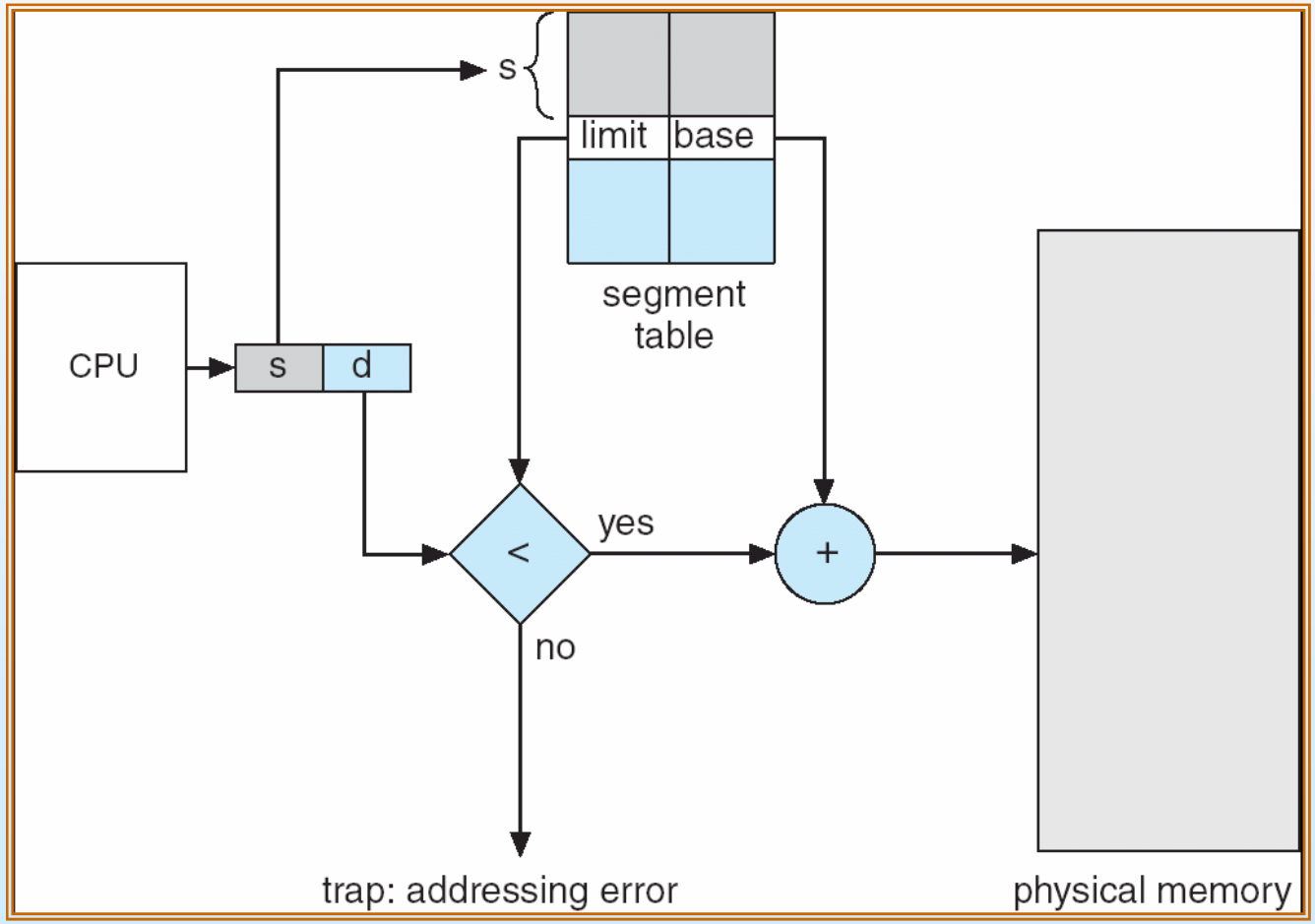
Arquitetura da Segmentação (Cont.)

- Relocação
  - Dinâmica
  - Por tabela de segmento
- Compartilhamento
  - Segmentos compartilhados
  - Mesmo número de segmento
- Alocação
  - *First-fit* (Primeira)/ *Best-fit* (Melhor)
  - Fragmentação Externa

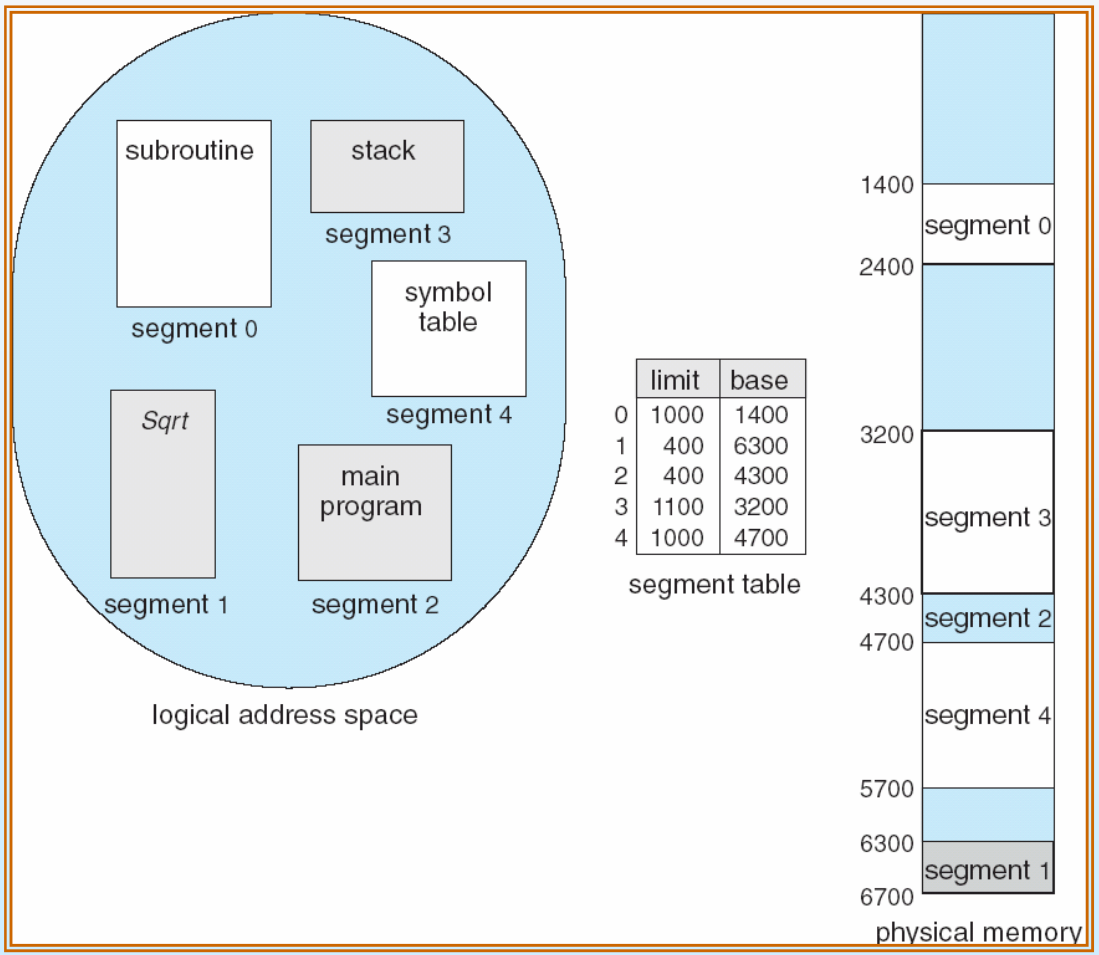
Arquitetura da Segmentação(Cont.)

- Proteção. Com cada entrada na tabela de segmento é associado:
  - Bit de validação = 0 $\Rightarrow$ segmento ilegal
  - Privilégios de leitura/escrita/execução
- Bits de proteção associados com segmentos; compartilhamento de código ocorre em nível de segmento
- Uma vez que segmentos variam em tamanho, alocação de memória é um problema dinâmico
- Um exemplo de segmentação é apresentado no diagrama a seguir

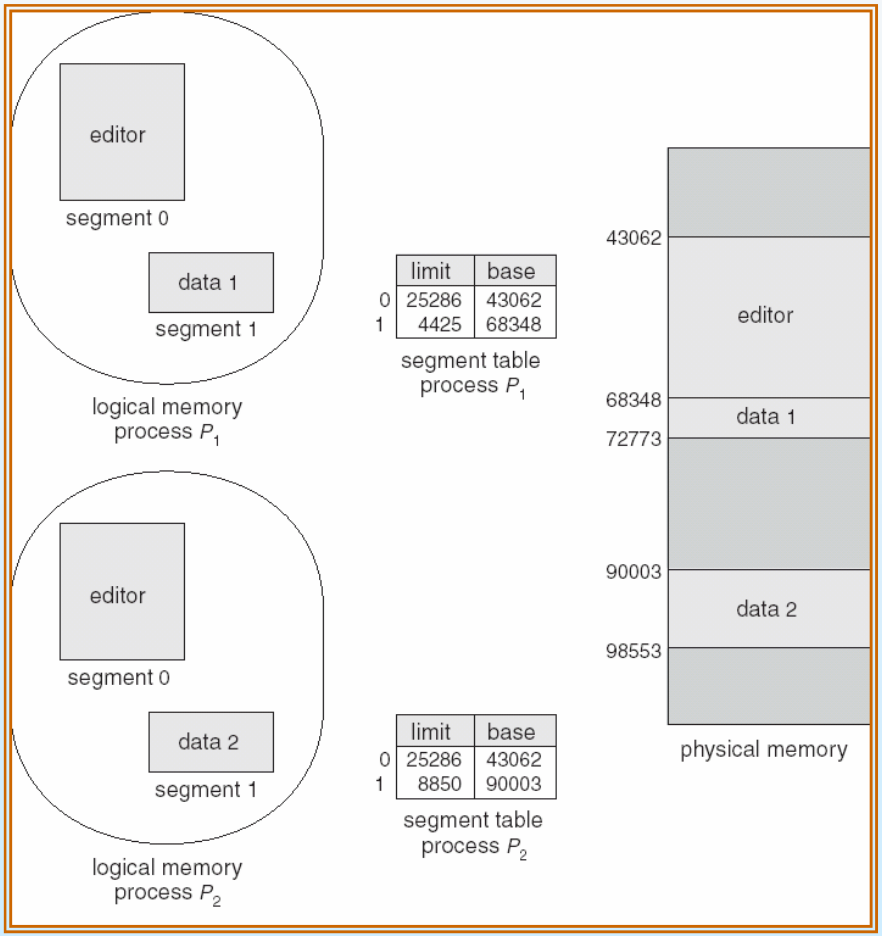
Arquitetura de Tradução de Endereços



Exemplo de Segmentação



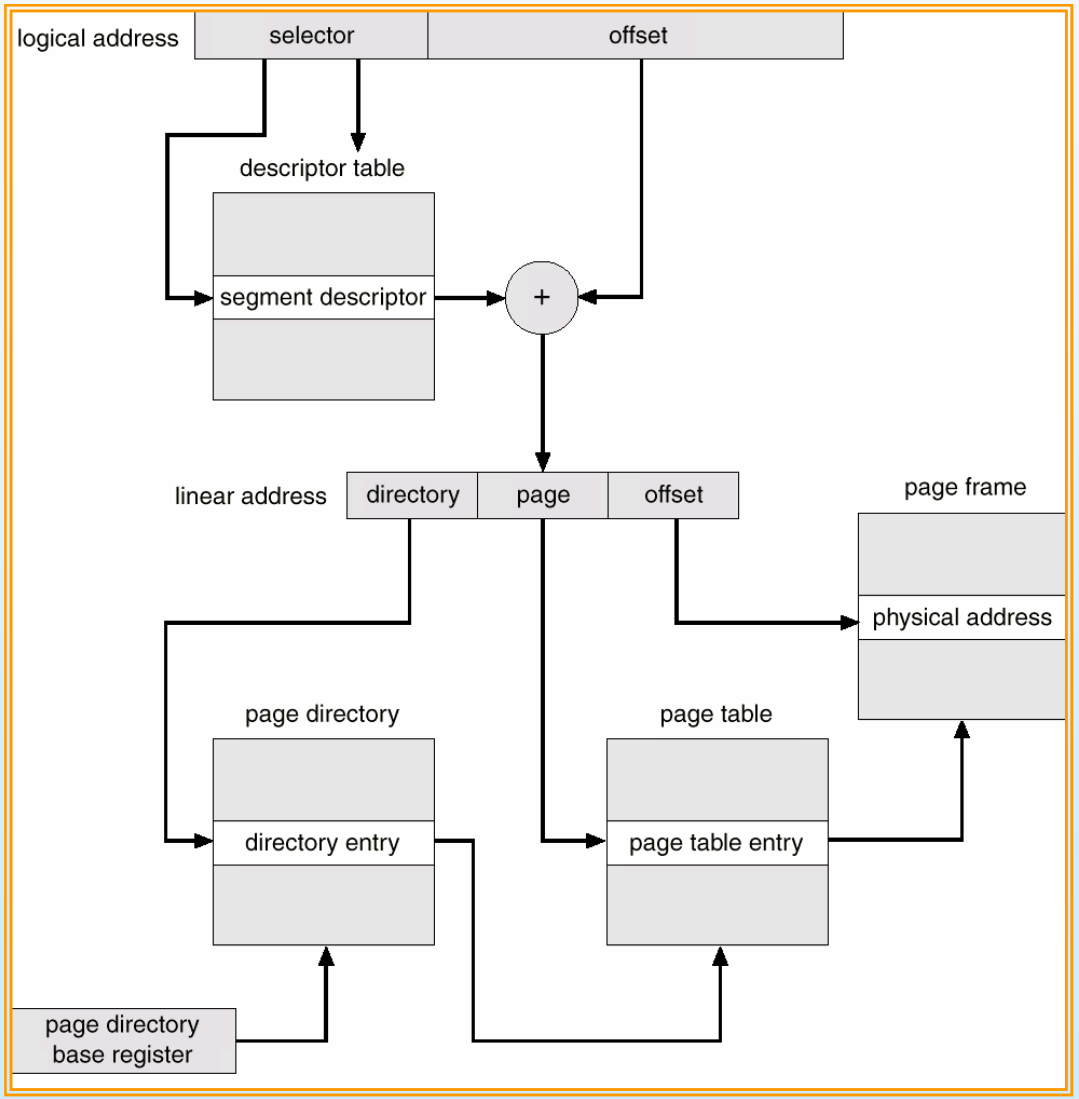
Compartilhamento de Segmentos



Segmentação com Paginação – Intel 386

- Como apresentado no próximo diagrama, o Intel 386 usa segmentação com paginação para gerenciamento de memória com um esquema de paginação em dois níveis.

Tradução de Endereços no Intel 30386



Fim do Capítulo 8