Memória central

João Paulo

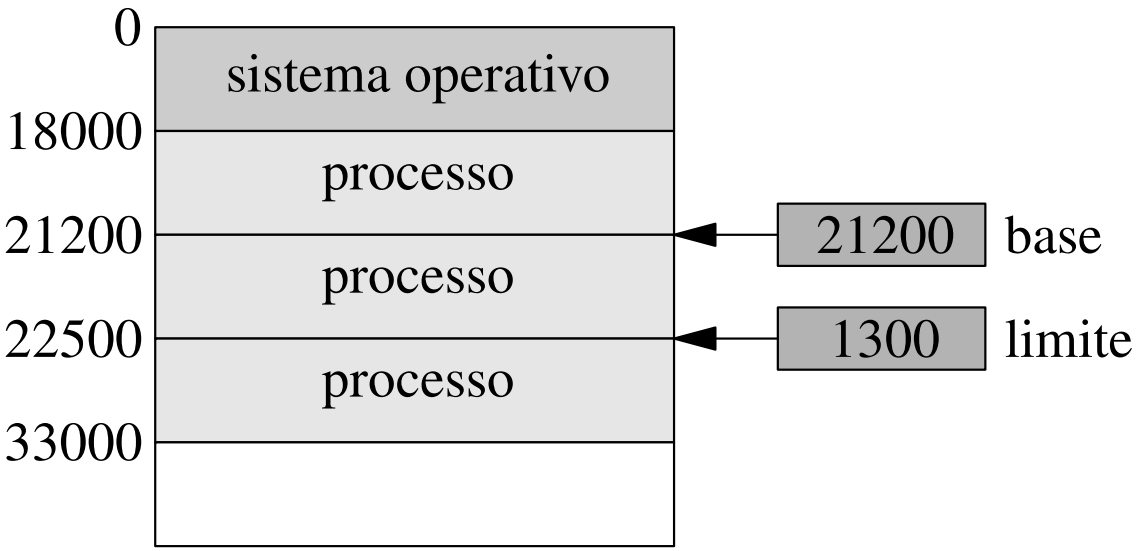
Grupo de Sistemas Distribuídos
Departamento de Informática
Universidade do Minho

Acesso à memória

- Programas têm que ser trazidos de disco para memória e instânciados num processo para correrem
- O processador acede directamente a registos e memória central
- Acesso aos registos muito rápido; e.g. 1 ciclo do relógio
- Acesso à memória rápido; alguns ciclos
- Cache do processador entre memória e registos
- Acessos à memória necessitam de protecção para isolar processos

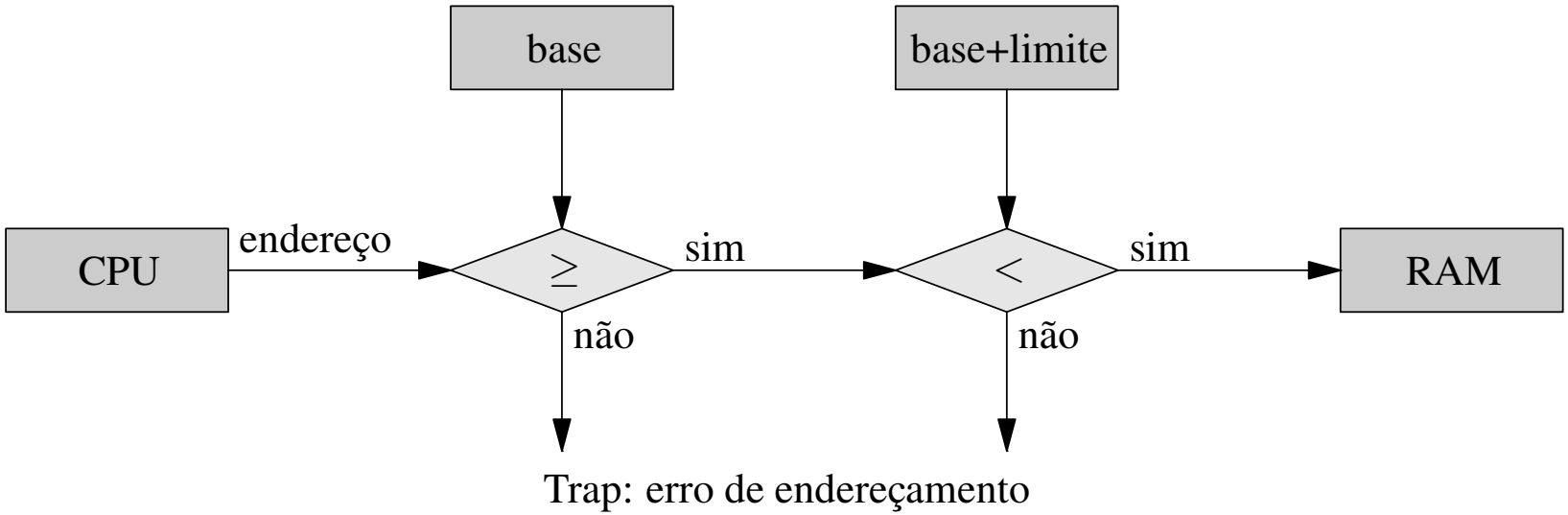
Protecção via registos base e limite

- Solução mais simples: delimitar uma zona de memória
- Registo base contém primeiro endereço da zona válida
- Registo limite contém tamanho da zona



Protecção via registos base e limite

- Endereço é comparado com base e base+limite
- Se fora da zona permitida é gerada uma trap e o SO recupera controlo



Fases de processamento de um programa

Um programa desde escrito até correr passa por várias fases:

- Compilação: transforma o código fonte em módulo de código objecto
- Linking: agrupa vários módulos num programa executável, tranformando referências simbólicas em endereços
- Carregamento (loading): carrega o programa para memória, juntamente com bibliotecas necessárias

Carregamento dinâmico (Dynamic Loading)

- Adiar carregamento de código de rotinas até estas serem invocadas
- Melhora uso da memória: algumas rotinas podem nunca ser invocadas, e nunca são carregadas
- Exemplos: rotinas de tratamento de erros raramente utilizadas
- Existe no contexto de um processo; não necessita de suporte do sistema operativo

Dynamic Linking

- Linking adiado para tempo de execução
- Pedaço de código (stub), localiza a rotina ou carrega-a se ainda não estiver em memória
- Partilha das rotinas por vários processos: rotinas são usadas mesmo que tenham sido pedidas previamente por outro processso
- É particularmente útil para bibliotecas: shared libraries
- Necessita de suporte do sistema operativo: envolve partilha entre processos

Associação de código e dados à memória

Existem 3 alternativas para associar código e dados a endereços de memória:

- Tempo de compilação
  - se localização conhecida, código gerado pode conter endereço absoluto atribuído a priori
  - problema: se localização muda necessário recompilar
- Tempo de carregamento:
  - se a localização não é conhecida em tempo de compilação
  - é gerado código recolocável
- Tempo de execução:
  - se processo pode mudar de localização durante execução
  - associação é adiada até à execução de cada instrução
  - necessita de suporte de hardware

Endereços lógicos versus físicos

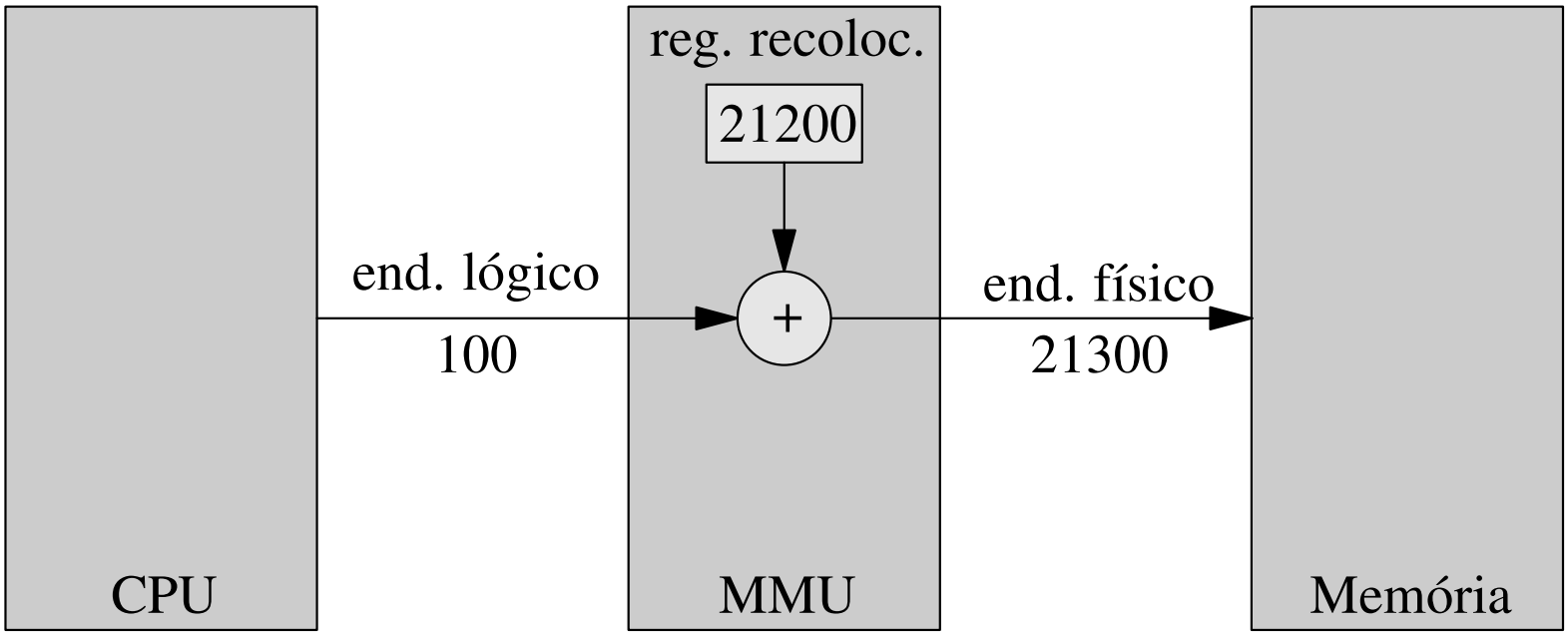
- Essêncial para gestão de memória a separação entre 2 conceitos:
  - Endereço lógico ou virtual: manipulado pelo processador, existente nas instruções e dados do programa / processo
  - Endereço físico: manipulado pela unidade de gestão de memória do processador
- Na associação em tempo de compilação ou carregamento – endereços lógicos e físicos coincidem
- Na associação em tempo de execução – é feita uma tradução de endereços em cada acesso

Unidade de gestão de memória (MMU)

- A memory management unit é o hardware (tipicamente parte do processador) que traduz endereços lógicos em físicos
- A tradução é feita sempre que o processo necessita de aceder à memória; e.g. adicionando o valor de um registo
- Os programas contém endereços lógicos, nunca manipulam endereços físicos

Recolocação dinâmica via registo

- A mais simples tradução de endereços lógicos em físicos
- Registo de recolocação é adicionado a cada endereço lógico



Swapping

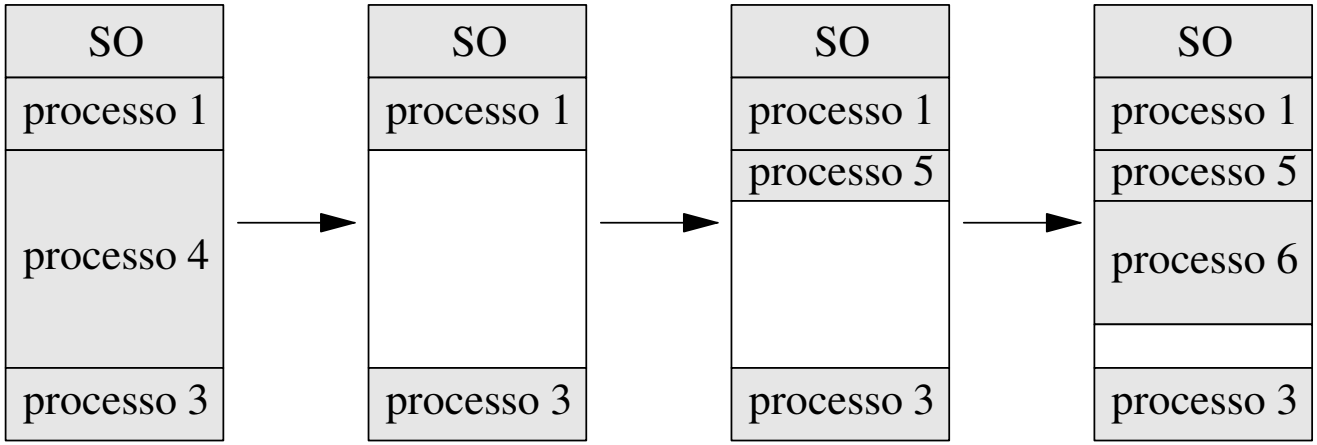
- Quando um processo é swapped out, a sua imagem é guardada em disco; quando é swapped in, é carregado de volta
- Questão: volta para a mesma localização ou outra?
- Depende da associação de endereços:
  - se feita em tempo de compilação ou carregamento, tem que voltar para a mesma localização
  - se feita em tempo de execução, pode ser carregado para outros endereços
- Diferentes combinações de processos vão existindo quando o sistema corre
- Ter que voltar para o mesmo sítio seria prejudicial para a utilização da memória ou liberdade de escalonamento

Gestão de memória contígua

- Memória dividida em duas grandes partições:
  - sistema operativo, geralmente em endereços baixos
  - processos em endereços altos
- Memória é atribuída aos processos quando carregados ou swapped in e libertada quando terminam ou swapped out
- Partição para processos é dividida pelos vários processos
- Como a repartir?
- Solução simples e má: partições de tamanho fixo
- Que outras hipóteses?

Partições de tamanho variável

- A partição para processos vai sendo subdividida e alocada à medida das necessidades
- O número de partições resultante é variável
- Quando processos saiem deixam partições livres: “buracos”
- Os buracos são em número e tamanho variável
- O sistema operativo tem que manter informação sobre as partições livres e ocupadas



Problema de alocação dinâmica

- Como escolher um bloco livre para satisfazer um pedido?
- O bloco tem que ter pelo menos o tamanho requesitado
- E se houver vários que pudessem servir?
- Diferentes políticas possíveis:
  - first-fit: alocar o primeiro suficientemente grande
  - best-fit: alocar o mais pequeno que seja suficientemente grande
  - worst-fit: alocar o maior bloco
- Os dois primeiros são mais eficientes no uso da memória
- Os dois últimos necessitam de percorrer a lista toda ou que esta esteja ordenada por tamanhos

Fragmentação

- Ao particionar a memória disponível surge o problema da fragmentação: pode a totalidade da memória livre ser suficiente, mas não existir nenhum bloco contíguo que sirva
- Fragmentação externa:
  - fragmentação que diz respeito ao blocos livres;
  - podem existir muitos mas pequenos e nenhum servir
- Fragmentação interna:
  - diz respeito a memória interna a cada partição, mas sem uso
  - motivação: não faz sentido marcar um bloco livre de tamanho mais pequeno do que o necessário para o gerir
  - assim pode ser dado a cada processo ligeiramemente mais memória do que o pedido

Compactação

- A fragmentação externa poderia ser atacada com compactação
- Consiste em mover os blocos ocupados para ficarem juntos, ficando o espaço livre contíguo
- Problemas:
  - recolocação de endereços: exige que esta seja feita em tempo de execução (não é problema)
  - custo de mover conteúdo da memória
  - não pode ser feito se existir I/O em curso com DMA para memória do processo; solução: usar buffers do SO

Paginação

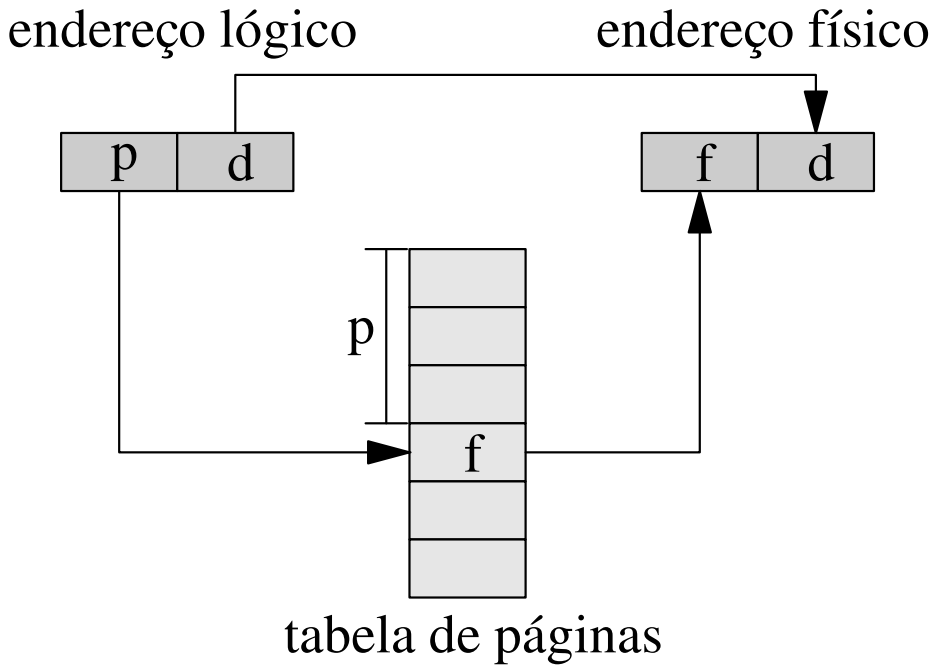
- Os espaços de endereçamento lógico e físico são divididos em blocos de tamanho fixo; tamanho uma potência de 2; tipicamente entre 512 e 8192 bytes
- Os blocos de endereçamento lógico são chamados páginas
- Os blocos de endereçamento físico são chamados frames
- Tabela de páginas mapeia páginas em frames
- Blocos iguais: alocação simples, sem fragmentação externa
- Fragmentação interna na última página de cada processo

Tradução de endereços

- Endereço lógico é dividido em
  - número de página: usado para indexar tabela de página que contém endereço base da frame
  - deslocamento: combinado com o endereço base da frame
- Exemplo: endereços de 32 bits; páginas de 4KB
  - deslocamento entre 0 e 4095
  - 12 bits para guardar deslocamento
  - 20 bits para número de página

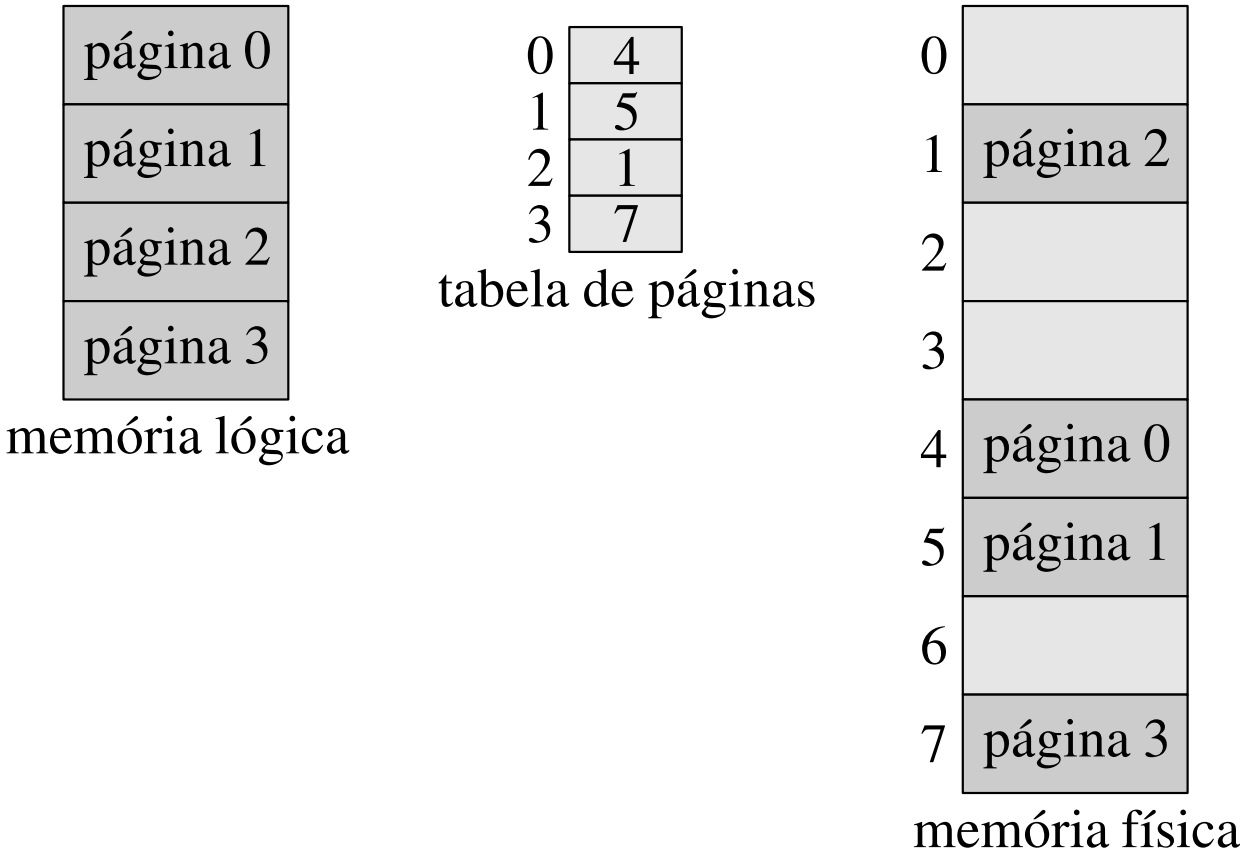
Tradução de endereços

- Componente *p* serve de índice na tabela de páginas
- Número da frame *f* é extraído da tabela
- *f* é composto com o deslocamento *d*



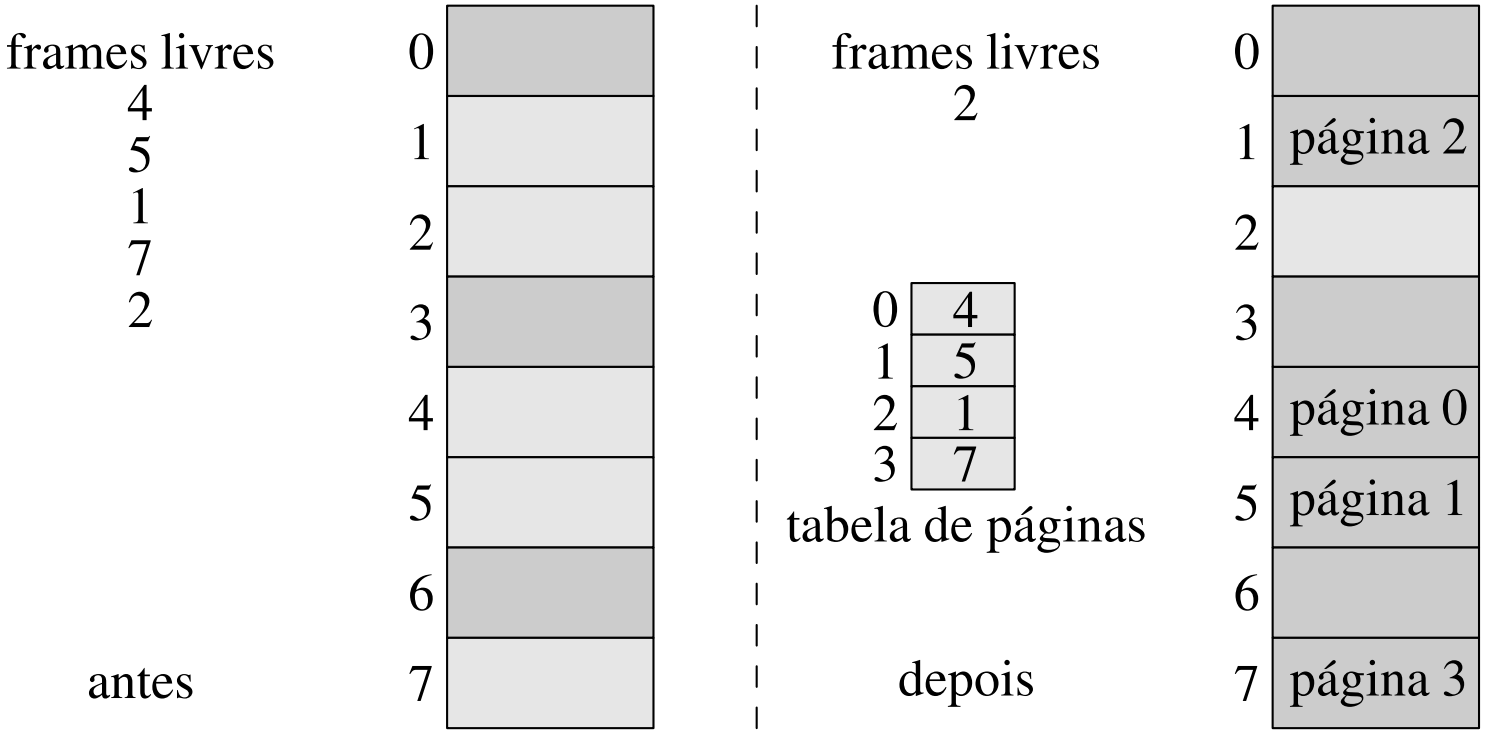
Memória lógica e física

- Processos vêem memória lógica
- Páginas estão dispersas pela memória física



Lista de frames livres

- Para criar processo é consultada lista de frames livres
- Exemplo: criação de processo com 4 páginas



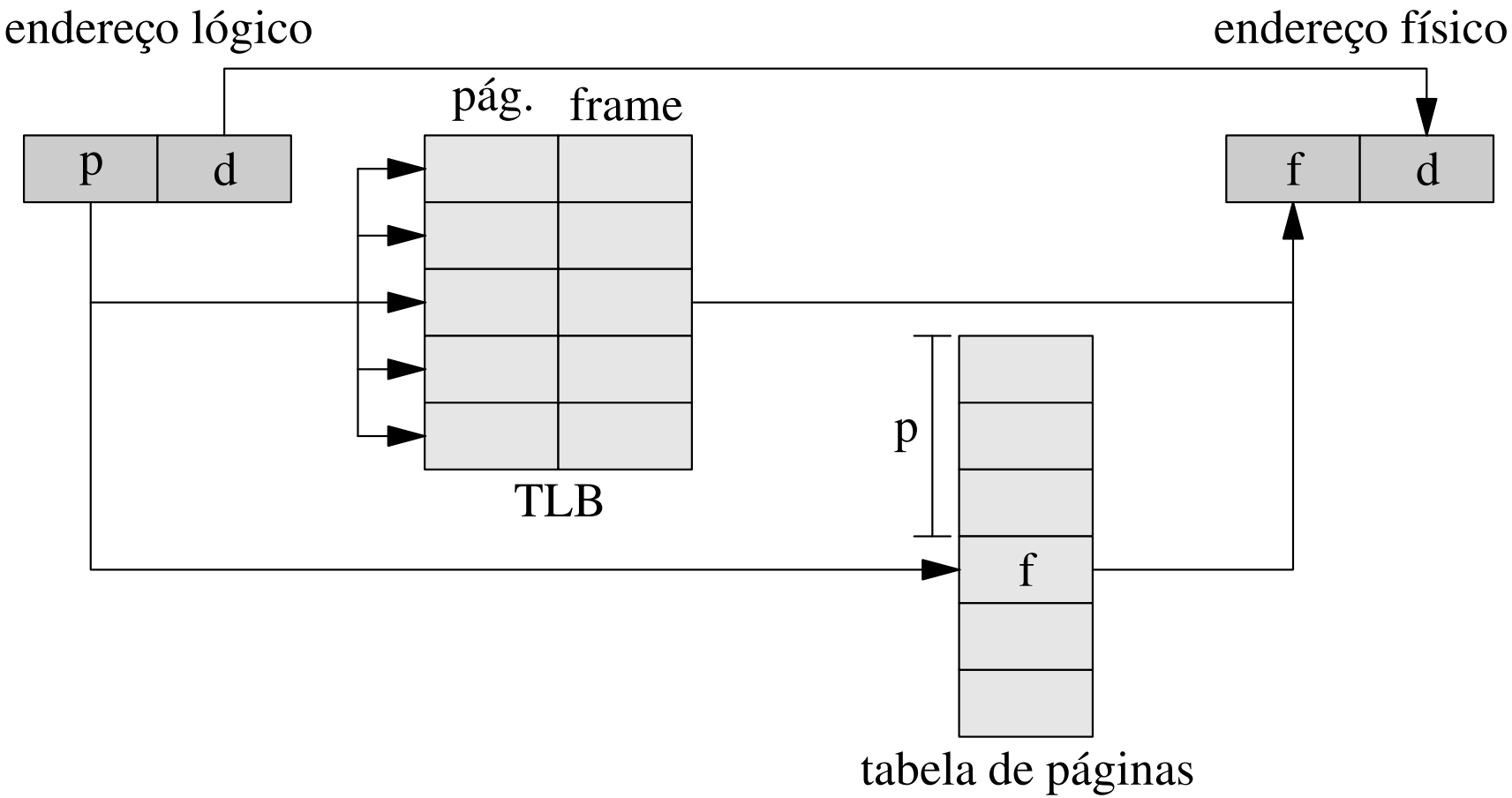
Tabelas de páginas e acesso à memória

- Tabela de páginas está em memória
- Registos guardam posição e tamanho da tabela
- cada acesso a um endereço lógico implica dois acessos a memória:
  - acesso à tabela de páginas
  - acesso à posição mapeada
- Inaceitável
- Solução: uso de cache de mapeamentos com memória associativa – translation look-aside buffer (TLB)

Translation look-aside buffer

- Usa memória associativa: dado uma chave (número de página), devolve valor associado (número de frame), se chave presente
- Mecanismo caro: apenas guarda algumas associações (e.g de 64 a 1024)
- Quando número de página em cache no TLB, mapeamento é imediato
- Caso contrário, é acedida a tabela de página e colocada a associação no TLB
- Caso TLB cheio, é rejeitada outra associação, segundo uma política; e.g. least recently used (LRU)
- Um TLB pode guardar address-space identifiers (ASID) para distinguir processos; caso contrário necessário descartar entradas aquando comutação de processo

Hardware de paginação com TLB



TLB: tempo de acesso efectivo

- Para tempo de procura no TLB ϵ
- E hit ratio α: percentagem de vezes em que página está no TLB
- O tempo de acesso efectivo (relativamente ao acesso à RAM) é:

$$(1 + \epsilon)\alpha + (2 + \epsilon)(1 - \alpha) = 2 + \epsilon - \alpha$$

- Exemplo:
  - $\alpha = 0.96$
  - $\epsilon = 0.1$
  - acesso efectivo: 1.14 vezes tempo de acesso à RAM

Protecção de memória

- Pode ser associado a cada página permissões: leitura, escrita, execução
- Bits de permissão são guardados na tabela de páginas; hardware valida acessos
- Cada entrada na tabela de páginas pode também ser marcada como válida/inválida
- Permite definir zonas de endereçamento válidas para o processo
- Extende controlo do limite máximo dado pelo registo que guarda o tamanho da tabela de páginas

Páginas partilhadas

- Código partilhado
  - basta ter em memória uma única cópia de código
  - código não pode ser modificável (garantido pelo mecanismo de protecção)
  - o código é mapeado nos mesmos endereços lógicos em cada processo
- Memória partilhada
  - memória partilhada é um dos mecanismos de comunicação entre processos
  - pode ser obtida mapeando as diferentes zonas partilhadas que cada processo vê nas mesmas frames

Estrutura da tabela de páginas

- Gama de endereçamento lógico pode ser grande
- Processos podem usar endereços muito separados, ainda que ocupem pouca memória; e.g. stack em endereços grandes
- Tabela de páginas poderia ocupar muita memória
- Exemplo: 32 bits para endereços, páginas com 4KB (12 bits para deslocamento)
- Leva a: 1M páginas (2^{20}), ou seja, 4MB ocupados por processo para a sua tabela de páginas (assumindo entradas com 4 bytes)
- Inaceitável
- Soluções:
  - Tabelas de páginas hierárquicas
  - Tabelas de páginas hashed
  - Tabelas de páginas invertidas

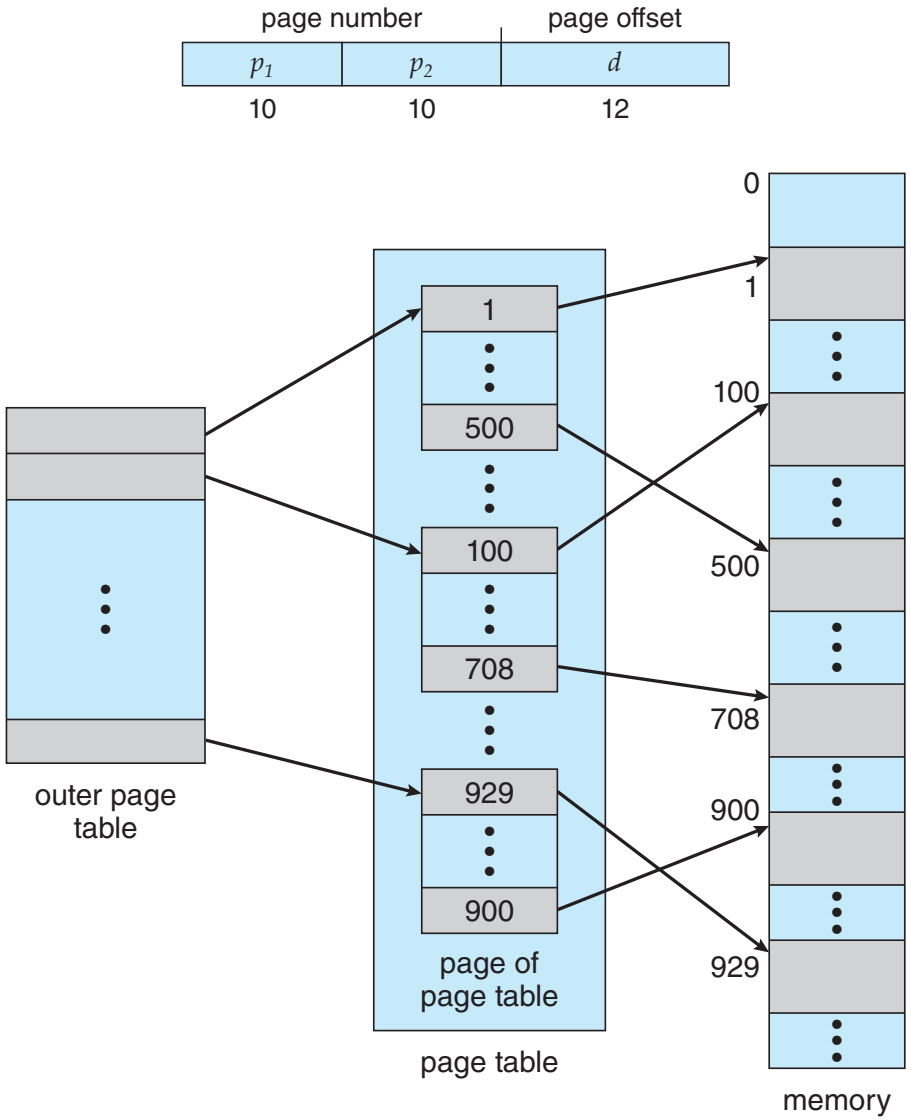
Tabelas de páginas hierárquicas

- Usar uma hierarquia de tabelas de páginas para mapear o espaço de endereçamento
- Tabela de páginas raíz contém endereços de outras tabelas de páginas.
- Vários níveis podem ser usados
- O último nível contém endereço das frames
- É possível apenas alocar as tabelas de página internas se contiverem 1 ou mais mapeamentos para frames (poupa espaço!)

Tabelas de páginas hierárquicas: exemplo

- Assumindo: Endereços lógicos de 32 bits; páginas de 4KB
  - Endeços lógicos com deslocamento de 12 bits
  - Entradas na tabela de páginas: 4 bytes
- Então: Número de página p dividido em:
  - p_1 – índice na tabela raíz, 10 bits
  - p_2 – índice na tabela de páginas (que ocupa uma página) apontada por p_1, 10 bits
  - d – deslocamento, 12 bits

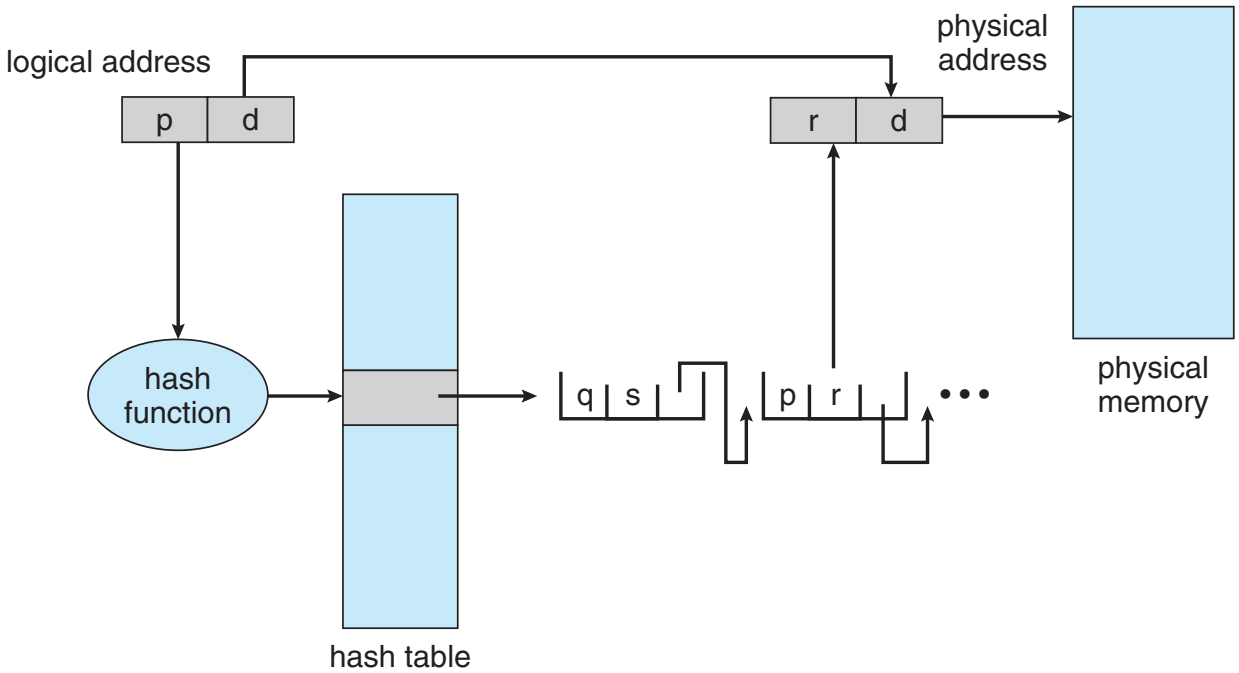
Tabelas de páginas hierárquicas: exemplo



Tabelas de páginas hashed

- Abordagem hierárquica não é apropriada para 64 bits.
  Tabela raíz endereça 2^{42} endereços (demasiado grande!) se p_2 tiver 10 bits e deslocamento for de 12 bits
- Solução má: usar mais níveis na hierarquia
  Exemplo: para o UltraSPARK 64-bit são necessários 7 níveis (número de acessos a memória excessivo!)
- Alternativa: usar uma tabela de hash
- O número de página é usado numa função de hash
- Cada entrada na tabela de hash contém lista ligada com pares (número de página, número de frame)

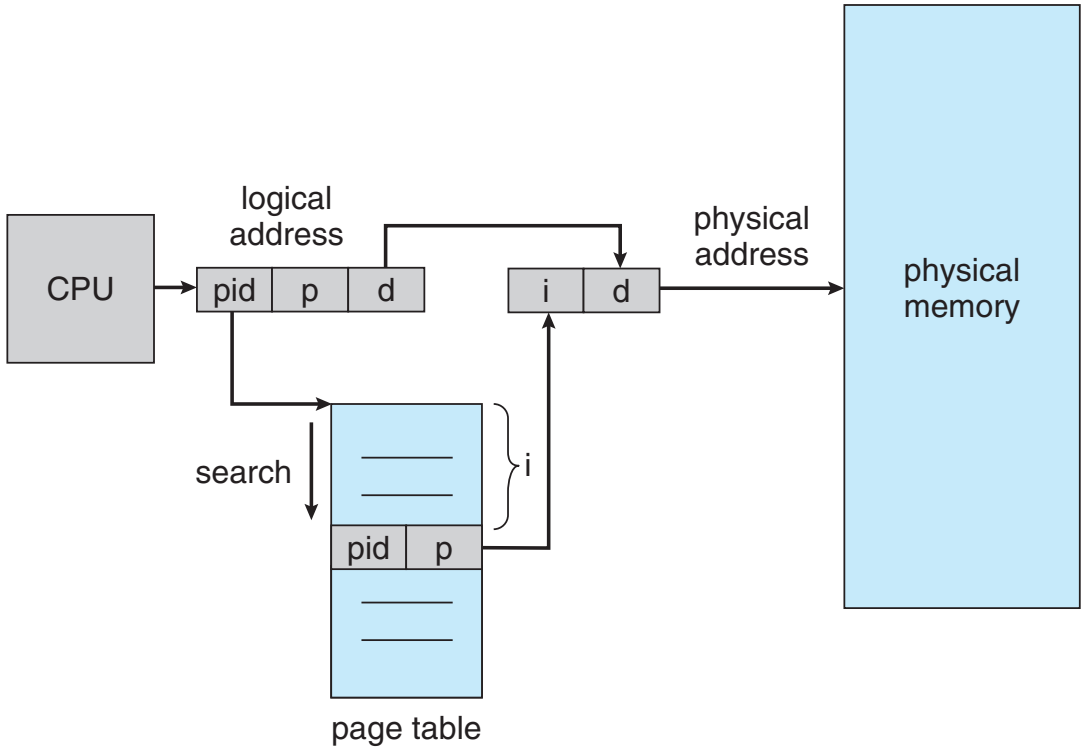
Tabelas de páginas hashed: exemplo



Tabelas de páginas invertidas

- Tabela que mapeia o uso da memória física.
- Cada entrada corresponde a uma frame
- Contém número de página e identificação do processo que usa a frame
- Tamanho limitado pela correspondencia à memória física (não há páginas a apontar para frames livres, poupa espaço!)
- Tradução de endereço envolve pesquisa linear pela tabela; inaceitável
- Solução: Usada em conjunção com tabela de hash
- Desvantagem: dificuldade em implementar partilha de frames. Um único processo por entrada na tabela.

Tabelas de páginas invertidas: exemplo (sem tabela de hash)



Segmentação

- O espaço de endereçamento é dividido em segmentos
- Cada segmento é uma unidade lógica como:
  - programa
  - rotina
  - stack
  - heap
  - zona de variáveis globais
- Cada endereço lógico é composto pelo número de segmento e deslocamento

Arquitectura da segmentação

- A tabela de segmentos contém a localização física dos segmentos; cada entrda contém:
  - base – endereço físico onde começa o segmento
  - limite – tamanho do segmento
- Registos contêm a localização da tabela de segmentos e o número de segmentos em uso pelo processo
- Privilégios (leitura, escrita, execução) são associados a cada segmento – apropriado
- Partilha feita a nível de segmento – apropriado
- Segmentos têm tamanho variável – alocação complexa

Segmentação + paginação

- Segmentação e paginação resolvem problemas diferentes
- Segmentação: estruturação do espaço de endereçamento em unidades lógicas do ponto de vista dos programas
- Paginação: gestão simples e eficiente da memória para uso por vários processos
- Podem ser combinados
- Exemplo: na arquitectura da Intel
  - endereços lógicos são mapeados pela segmentação em endereços lineares
  - endereços lineares são mapeados pela paginação em endereços físicos