Sistemas Operacionais

Professor:
Cristiano Bonato Both

Sumário

- Paginação
- Segmentação
- Memória virtual
- Mapeamento
- Problemas clássicos
- Referências

Introdução

- Problemas com alocação particionada
  - Necessidade de uma área contígua de memória (tamanho do processo)
    - Fragmentação interna (partições fixas) ou externa (partições variáveis)
- Nova abordagem é considerar a existência de um espaço de endereçamento lógico e de um espaço de endereçamento físico

Introdução

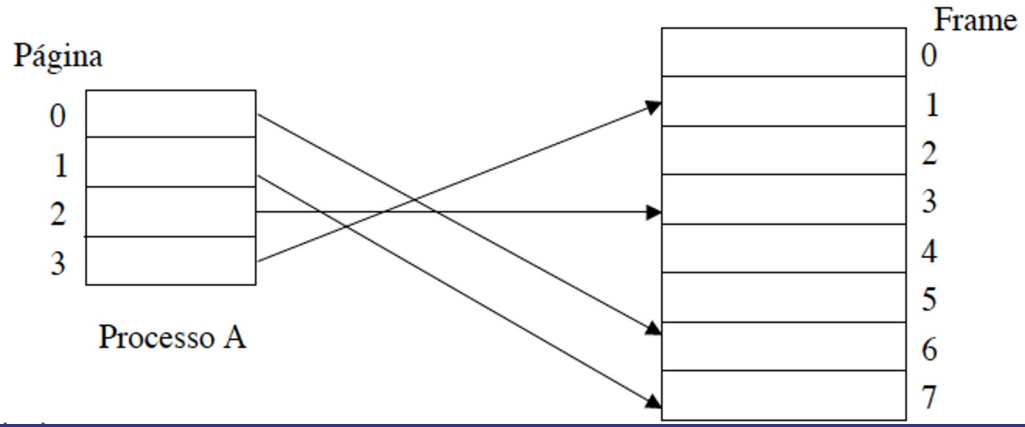
- O espaço de endereçamento físico não precisa ser contíguo
- “Mapear” o espaço lógico no espaço físico
  - Dois métodos básicos:
    - Paginação
    - Segmentação
- Suposição: para ser executado o processo necessita estar completamente carregado em memória

Paginação

- A memória física e a memória lógica são divididas em blocos de tamanho fixo e idênticos
  - Memória física dividida em blocos de tamanho fixo denominados de frames
  - Memória lógica dividida em blocos de tamanho fixo denominados de páginas
- Elimina a fragmentação externa e reduz a fragmentação interna

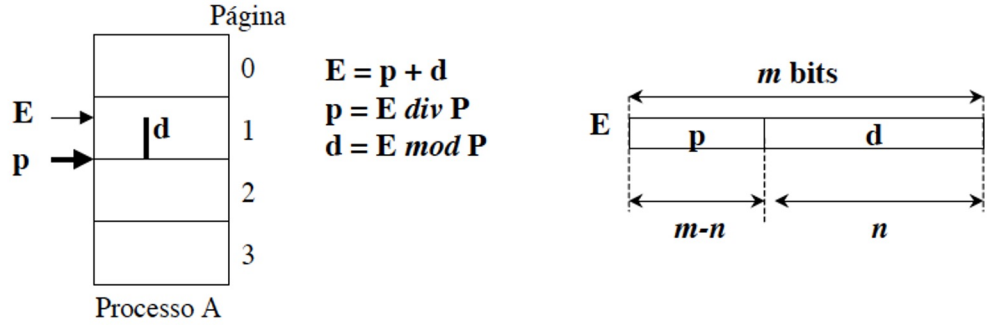
Paginação

- Para executar um processo de *n* páginas, basta encontrar *n* frames livres na memória
  - Páginas são carregadas em frame livre
- Necessidade de traduzir endereços lógicos (páginas) em endereços físicos (frames)

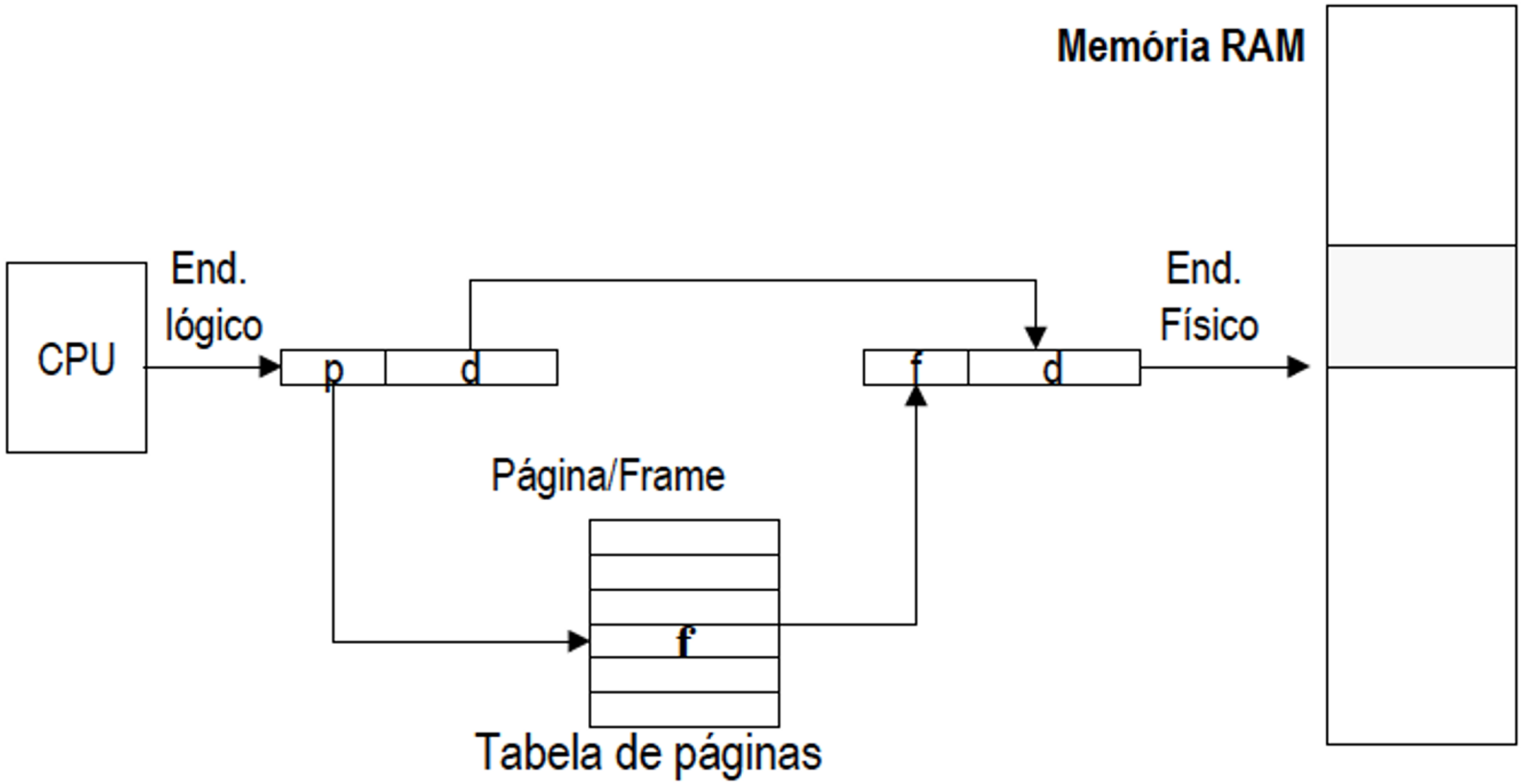


Paginação: endereço lógico

- Endereço lógico é dividido em duas componentes:
  - Número da página
  - Deslocamento dentro de uma página
- Tamanho da página (P) pode assumir qualquer tamanho, porém emprega-se um tamanho potência de 2 para facilitar operações *div* e *mod*

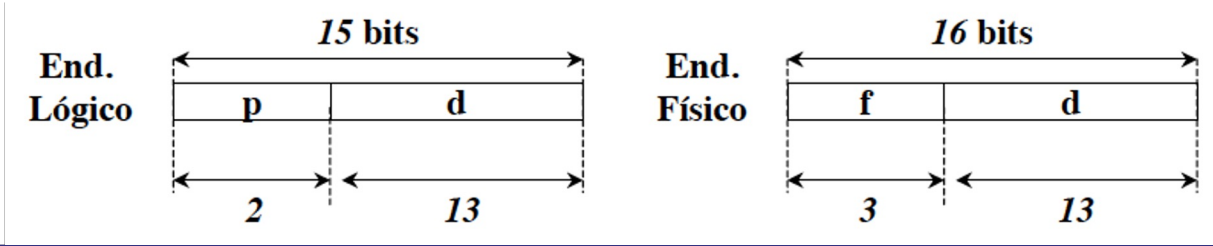


Tradução de endereço lógico em endereço físico

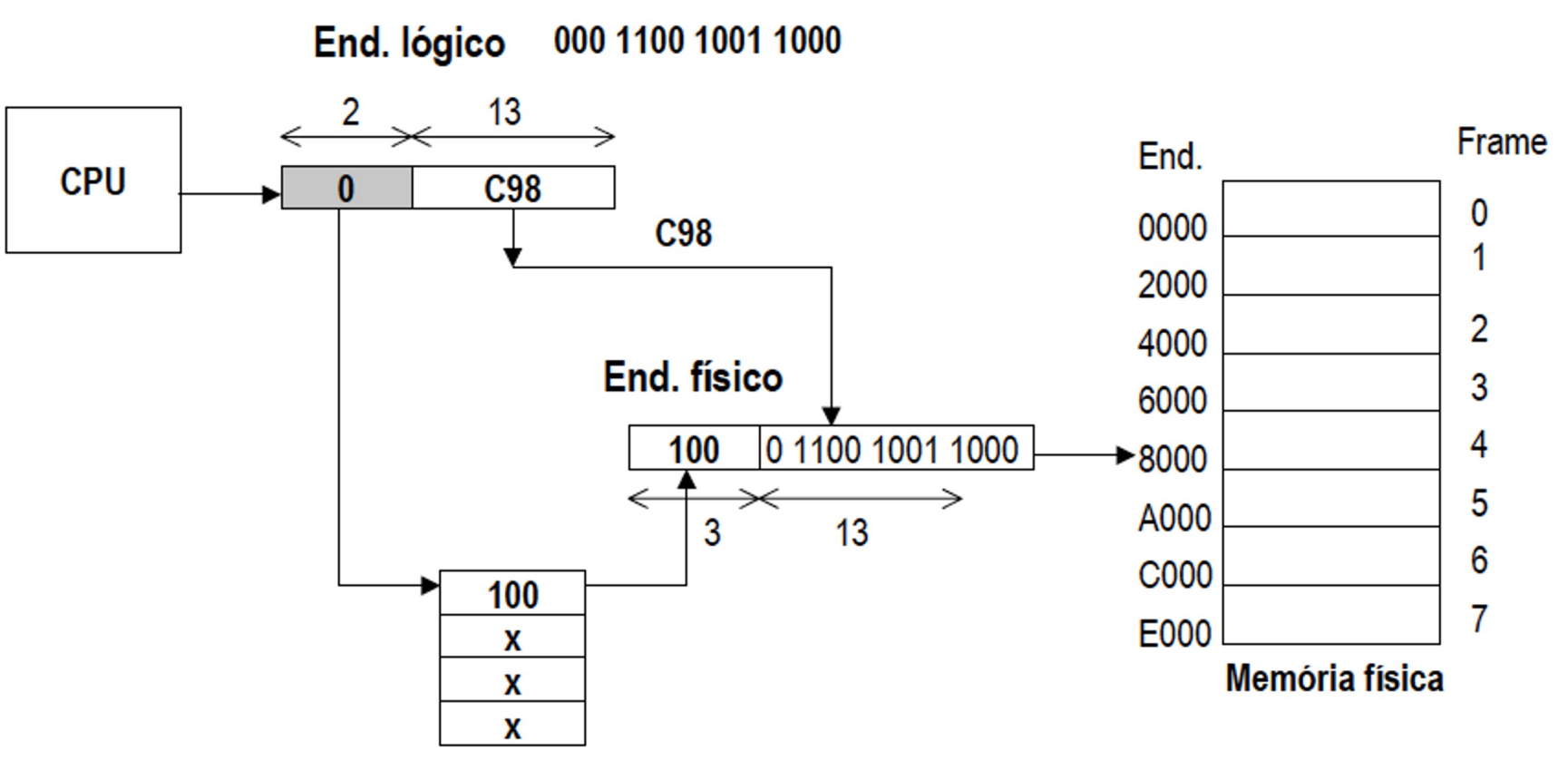


Exemplo de paginação

- Características do sistema:
  - Memória física: 64 kbytes (16 bits)
  - Tamanho processo (máx): 32 kbytes (15 bits)
  - Páginas 8 kbytes
- Paginação
  - Número de frames: 64/8 = 8 (0 a 7) -> 3 bits
  - Número de páginas: 32/8 = 4 (0 a 3) -> 2 bits
  - Deslocamento: 8 kbytes -> 13 bits



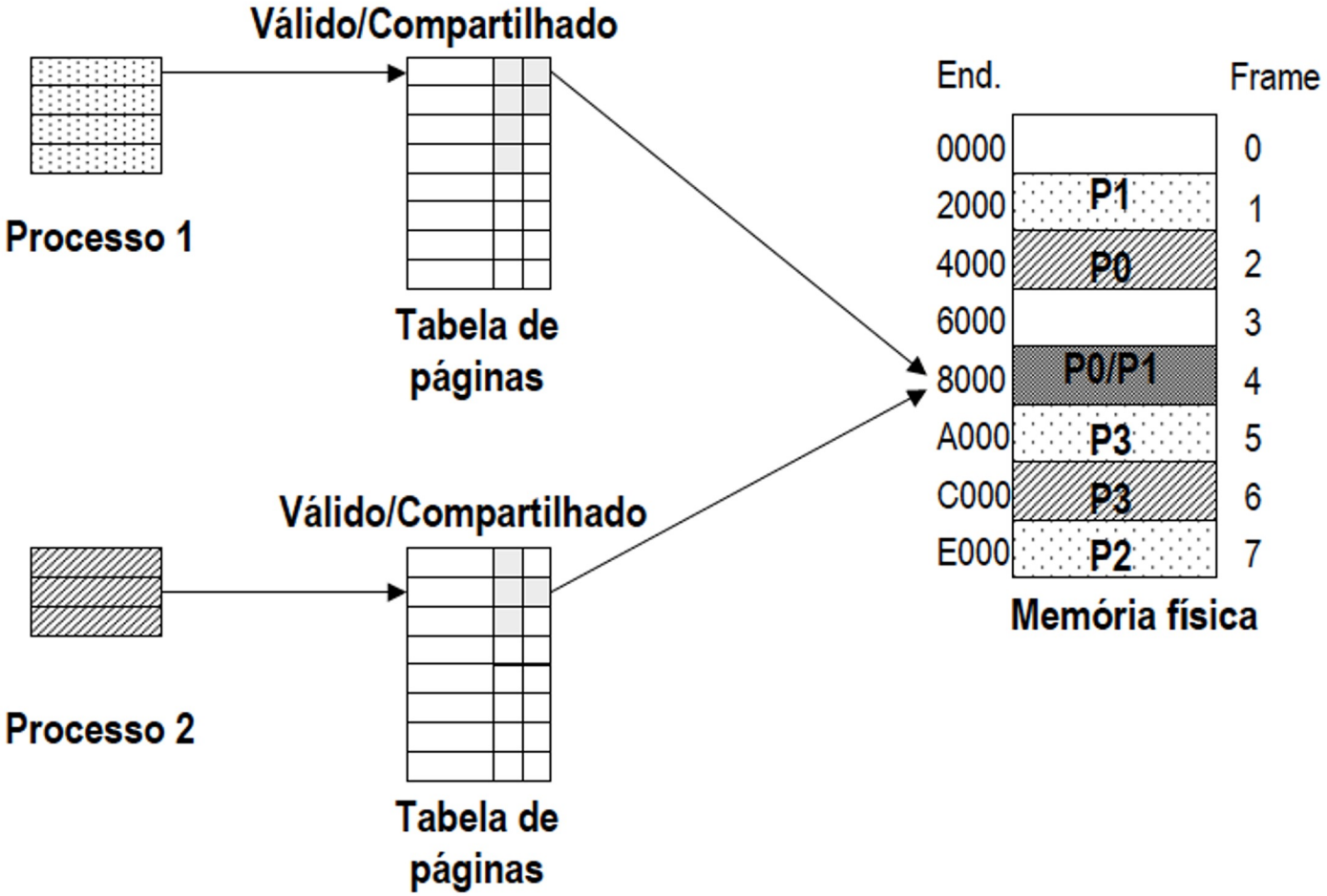
Execução de paginação



Compartilhamento de páginas

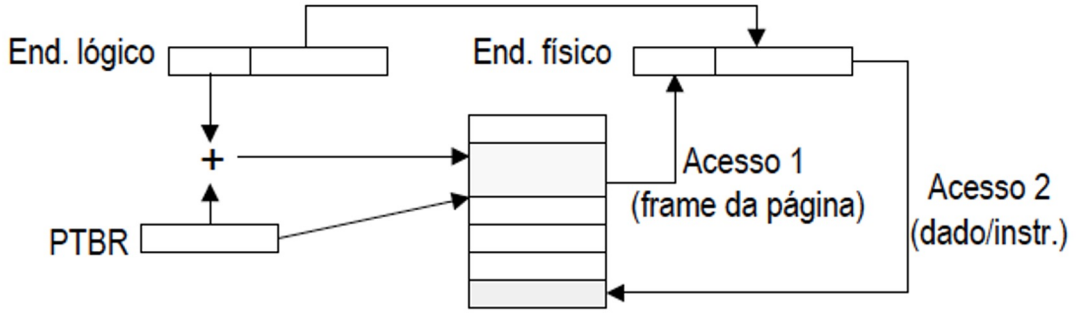
- Código compartilhado
  - Uma cópia do código (*read-only*, re-entrante) pode ser compartilhada entre vários processos (*e.g.*, editores de texto, compiladores, etc.)
  - O código compartilhado pertence ao espaço lógico de todos os processos
- Dados e código próprios
  - Cada processo possui sua própria área de código e seus dados

Exemplo de compartilhamento



Implementação da tabela de páginas em memória

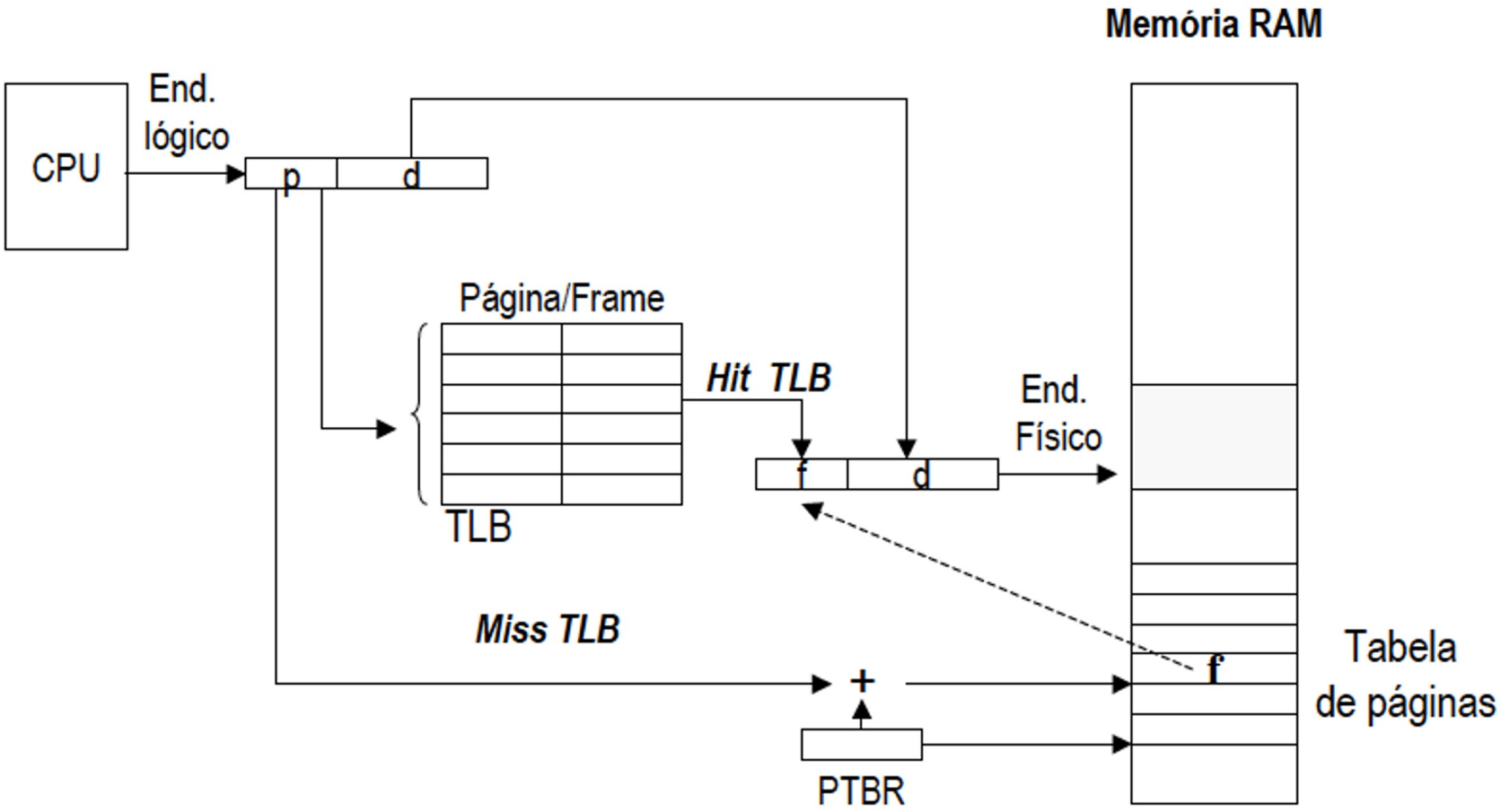
- Tabela de páginas é mantida em memória
  - *Page-Table Base Register* (PTBR): início da tabela de páginas
  - *Page-Table Length Register* (PTLR): tamanho em número de entradas
- Cada acesso a dado/instrução necessita, no mínimo, dois acessos a memória
  - Número de acesso depende da largura da entrada da tabela de página e de como a memória é acessada (bytes, word, etc.)



Translation Look-aside Buffers (TLBs)

- Uma espécie de meio termo entre implementação via registradores e via memória
- Baseada em uma memória cache especial (TLB) composta por um banco de registradores (memória associativa)
- Ideia é manter a tabela de páginas em memória com uma cópia parcial da tabela em um banco de registradores (TLB)
  - Página acessada está na TLB (*hit*): similar a solução de registradores
  - Página acessada não está na TLB (*miss*): similar a solução via memória

Implementação da tabela de páginas via TLB



Aspectos relacionados com o uso de TLB

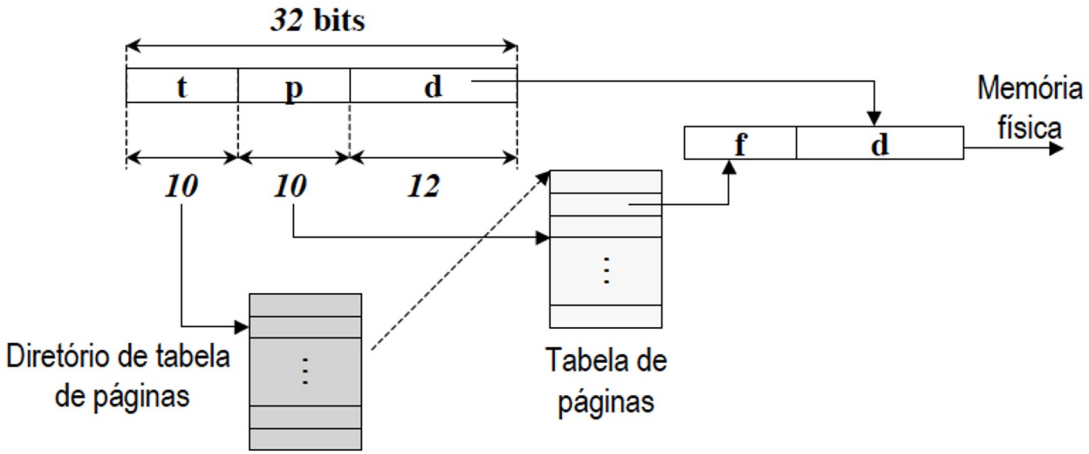
- Melhora o desempenho no acesso a tabela de páginas
  - Tempo de acesso 10 vezes menor que uma memória RAM
- Desvantagem é o seu custo
  - Tamanho limitado (de 8 a 2048 entradas)
  - Uma única TLB (pertence a MMU) que é compartilhada entre todos os processos
    - Apenas as páginas em uso por um processo necessitam estar na TLB
- Um acesso é feito em duas partes:
  - Se página está presente na TLB (*hit*) a tradução é feita
  - Se página não está presente na TLB (*miss*), consulta a tabela em memória e atualiza entrada na TLB

Paginação multinível

- Na prática as tabelas de página possuem tamanho variável
  - Como dimensionar o tamanho da tabela de páginas?
    - Fixo ou variável conforme a necessidade?
  - Como armazenar a tabela de páginas?
    - Contíguo em memória -> fragmentação externa
    - Paginando a própria tabela
- A paginação multinível surge como solução a esses problemas
  - Diretórios de tabela de páginas (n níveis)
  - Tabelas de páginas

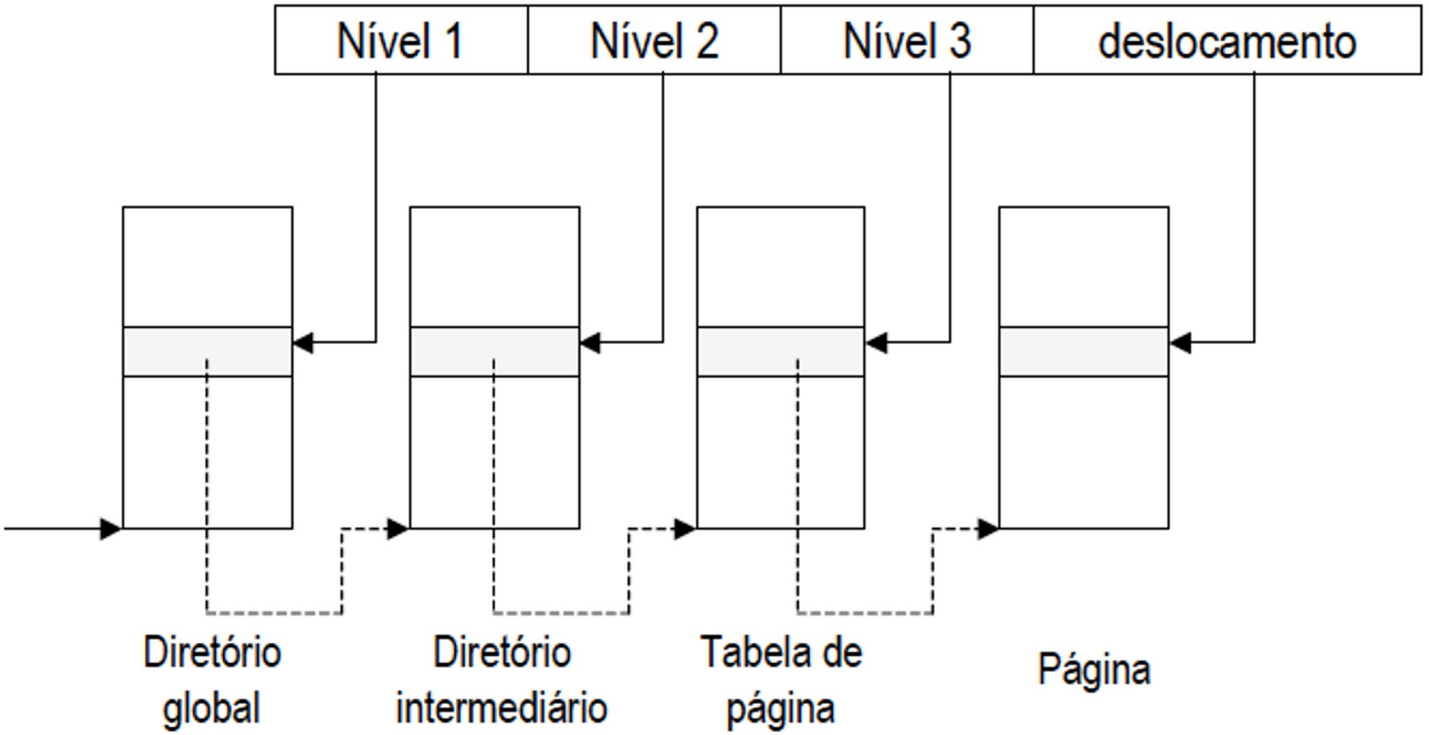
Exemplo: paginação de 2 níveis

- Processadores 80x86 (Intel)
  - End. Lógico: 4 Gbytes (32 bits)
  - Páginas: 4 Kbytes
  - Tamanho da tabela de páginas: 4 Gbytes/4 Kbytes = 1048576 entradas



Paginação de 3 níveis

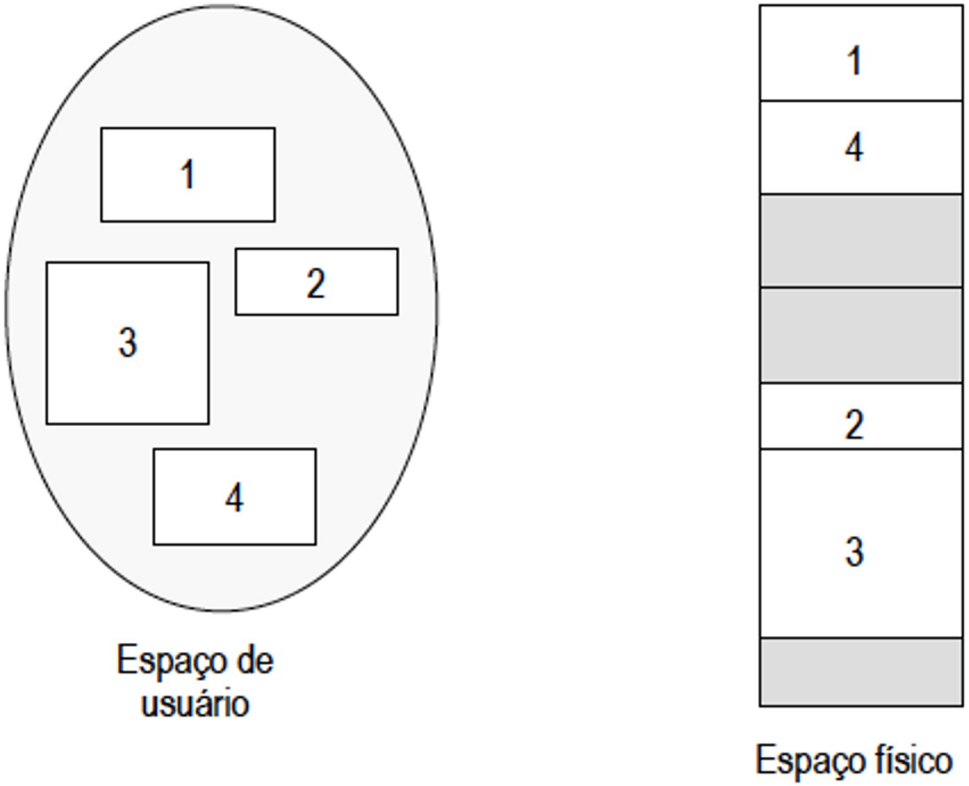
- Típico de arquiteturas de processadores de 64 bits



Segmentação

- Leva em consideração a visão de programadores e compiladores
- Um programa é uma coleção de segmentos, tipicamente:
  - Códigos, dados alocados estaticamente, dados alocados dinamicamente e pilha
- Um segmento pode ser uma unidade lógica
  - *e.g.*, procedimentos (funções), bibliotecas
- Gerência de memória pode dar suporte diretamente ao conceito de segmentos

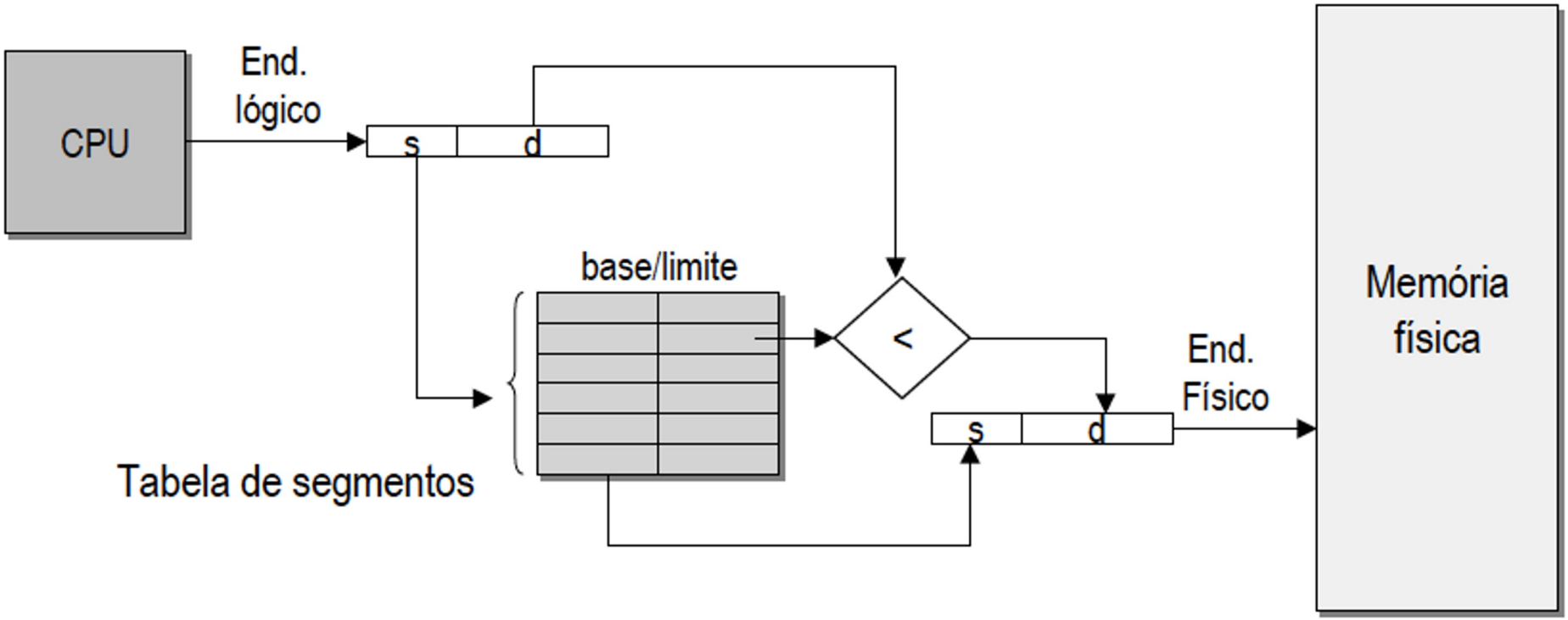
Esquema lógico de segmentação



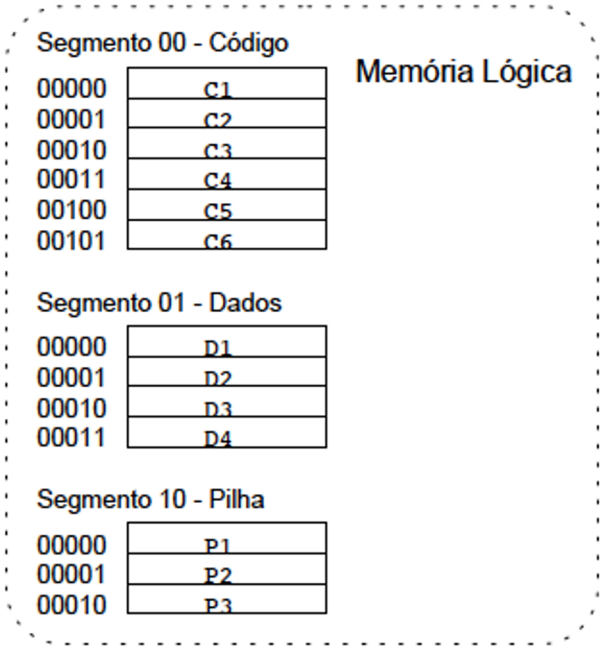
Endereço lógico em segmentação

- Endereço lógico é composto por duas partes:
  - Número de segmento
  - Deslocamento dentro do segmento
- Os segmentos não necessitam ter o mesmo tamanho
- Existe um tamanho máximo de segmento
- Segmentação é similar a alocação particionada dinâmica

Esquema de tradução de segmentação



Tradução de endereço lógico em endereço físico



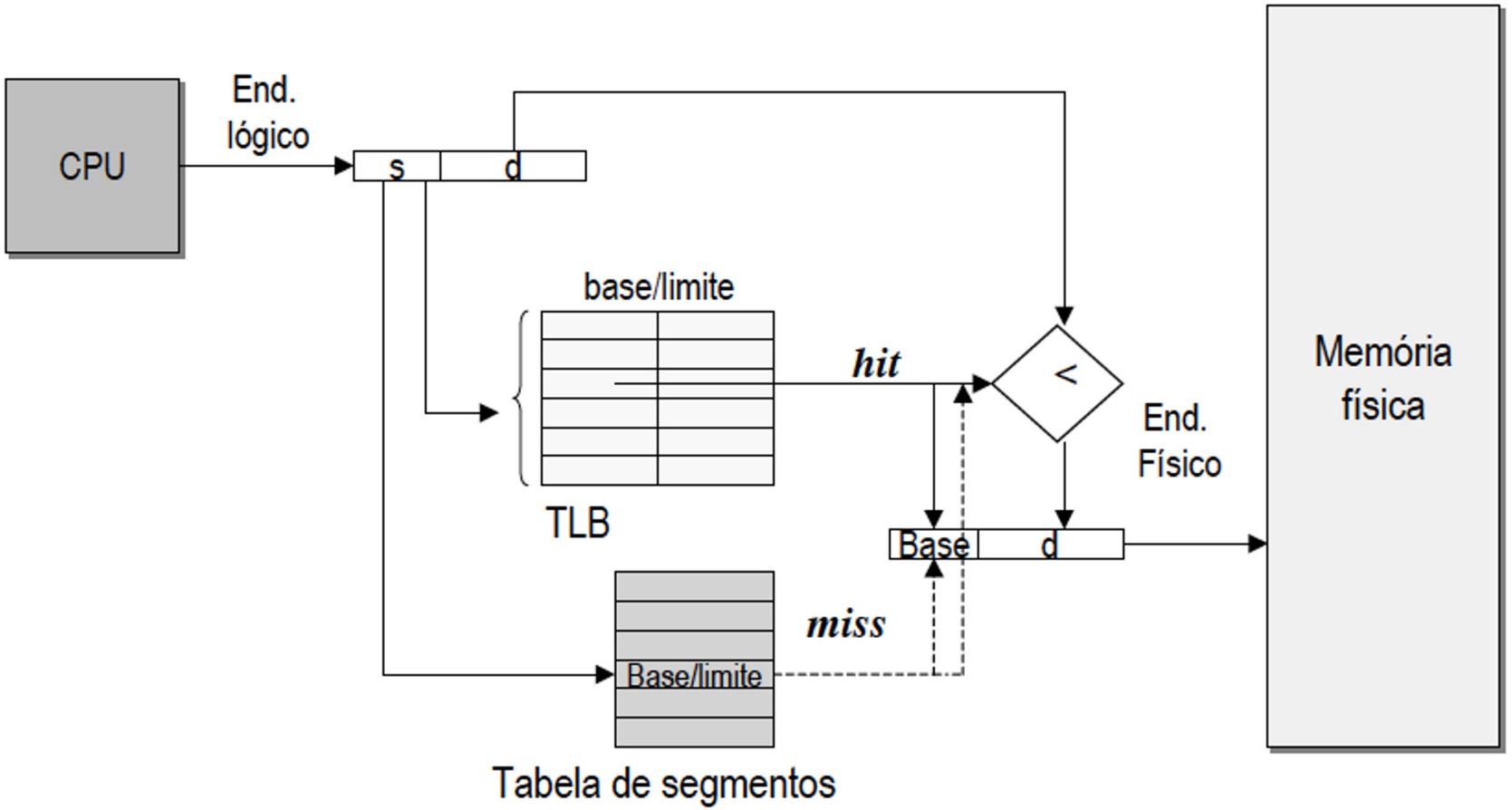
Memória Física

D1	00000
D2	00001
D3	00010
D4	00011
	00100
	00101
	00110
	00111
C1	01000
C2	01001
C3	01010
C4	01011
C5	01100
C6	01101
	01110
	01111
	10000
	10001
	10010
	10011
P1	10100
P2	10101
P3	10110
	10111

Tabela de Segmentos

Segmento	Base	Limite
00	01000	0110
01	00000	0100
10	10100	0011

Implementação da tabela de segmentos via TLB



Desvantagem da segmentação

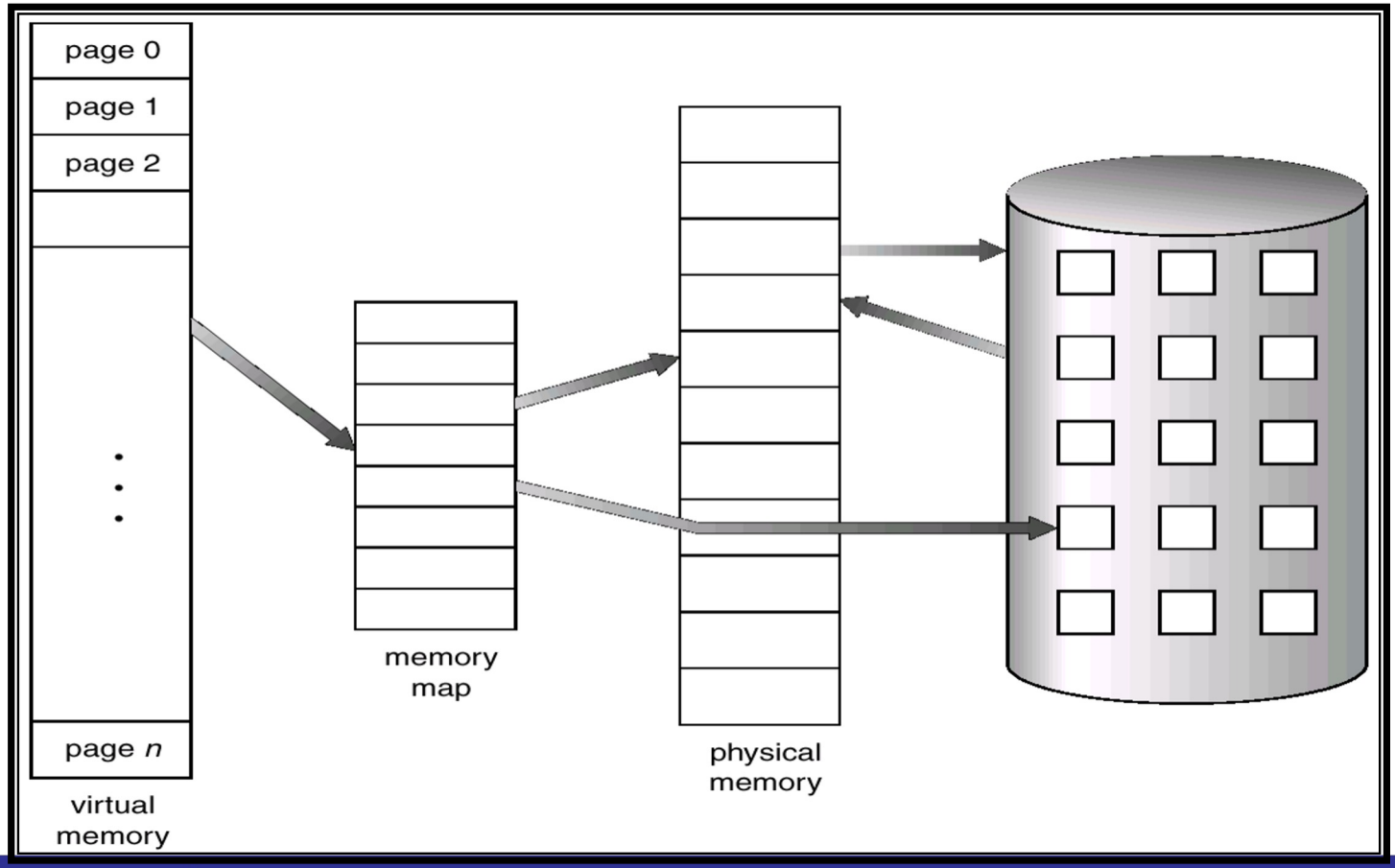
- A segmentação provoca fragmentação externa quando segmentos começam a liberar memória
- Mesmo problema de alocação de partições variáveis com as mesmas soluções:
  - Concatenação de segmentos adjacentes
  - Compactação da memória

Memória Virtual - conceitos

- Permite uma criação mais eficiente de processos
- Memória virtual pode ser criada via:
  - Paginação por demanda (*Demand Paging*)
  - Segmentação por demanda

Memória Virtual - conceitos

- Muito maior que a memória física:



Paginação por demanda

- Página inserida na memória somente quando for necessário:
  - Necessita de menos I/O
  - Necessita de menos memória
  - Resposta mais rápida
  - Mais processos
- Se uma página é necessária basta referenciá-la:
  - Referência inválida -> erro
  - Não está na memória -> carregá-la na memória (a partir disco)

Bit de validade

- Com cada entrada na tabela de página um bit válido-inválido é associado (1 = na memória, 0 = não está na memória)
- Inicialmente, todos os bits estão em 0

Quadro #	Bit válido-inválido
	1
	1
	1
	1
	0
	0
	0

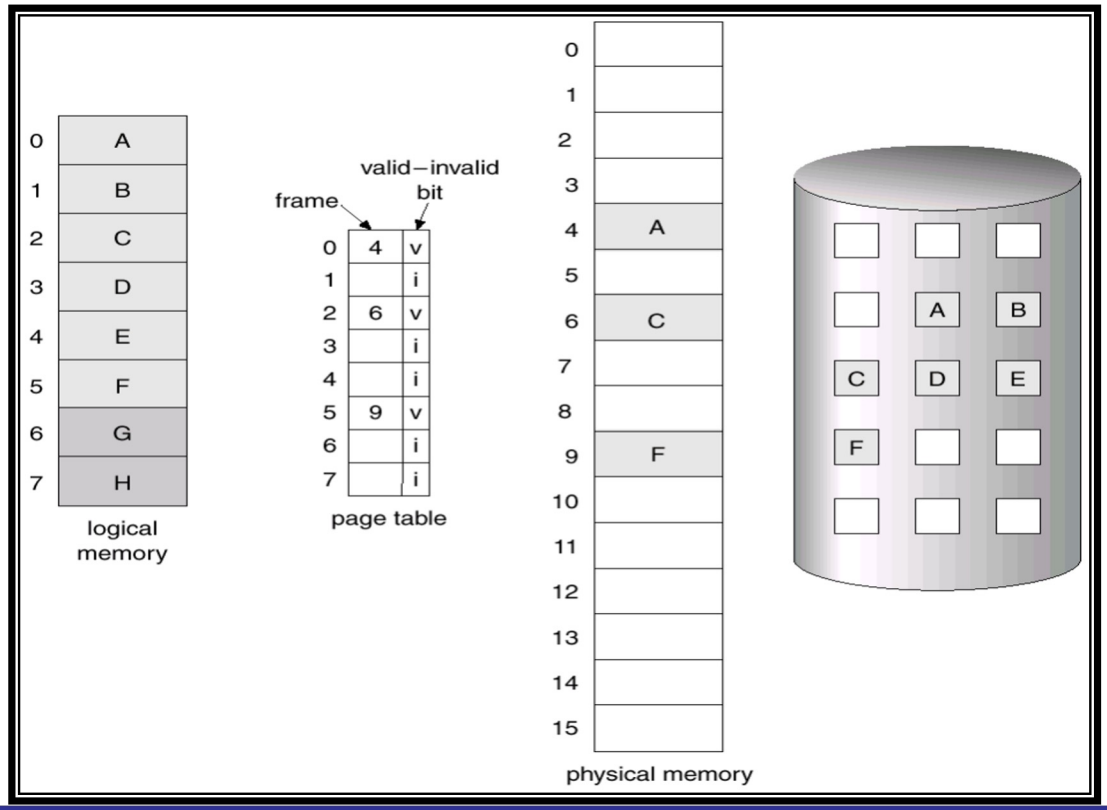
Bit de validade

- Durante a tradução de endereço
  - Se o bit válido-inválido na entrada é 0 = falha de página

- Acarreta suspensão do processo e inserção no estado bloqueado, até a página ser carregada na memória para o processo ser inserido na fila de aptos

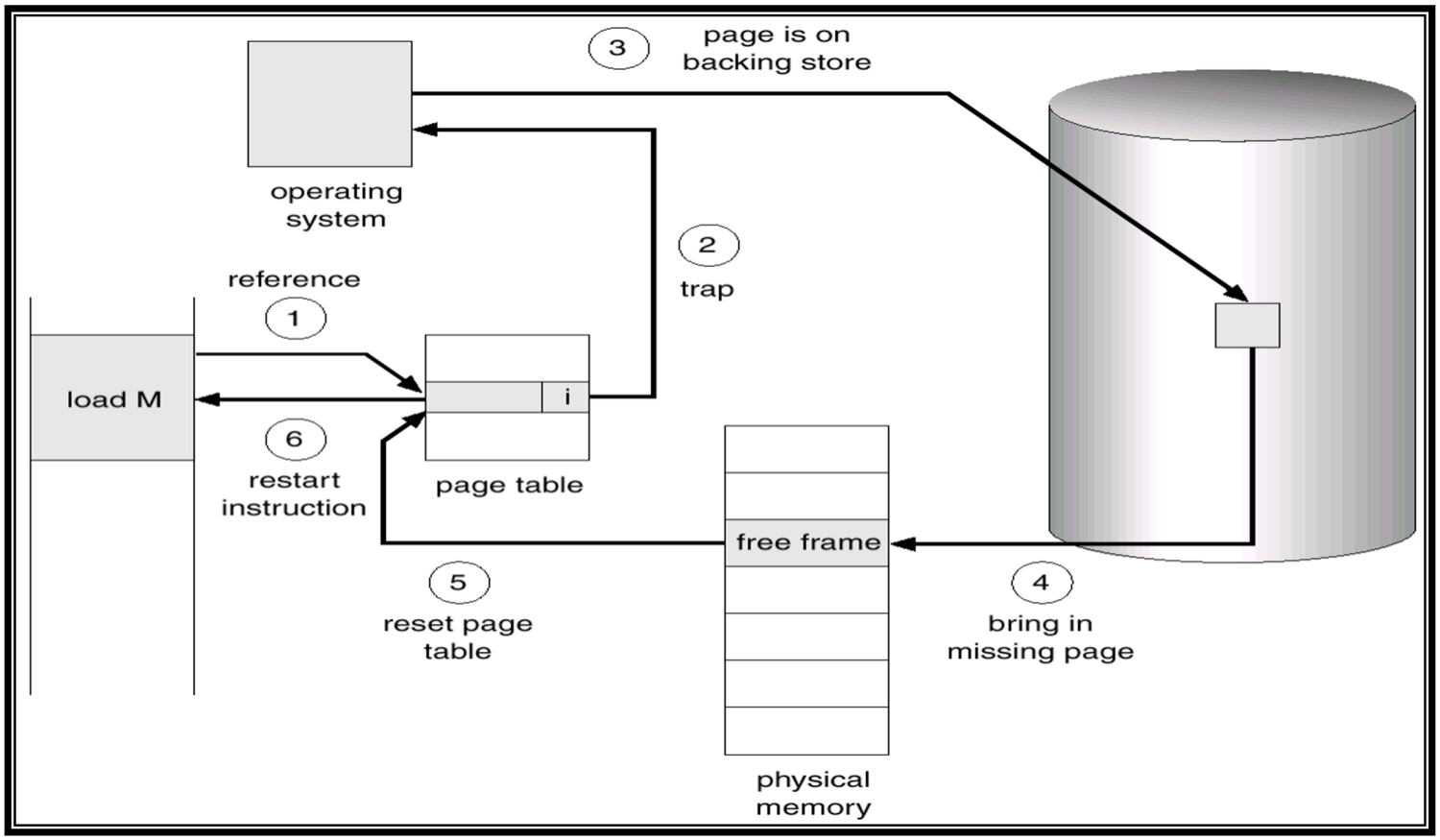
Paginação por demanda

- Tabela de páginas com algumas páginas que não estão na memória:



Paginação por demanda

- Passos para o tratamento de falha de página:



Sem quadros livres

- Substituição de páginas:
  - Encontrar alguma página em memória e armazená-la em disco:
    - Algoritmo
    - Desempenho:
      - Queremos um algoritmo que resulte no menor número de falhas de página
- Algumas páginas podem sair da memória várias vezes

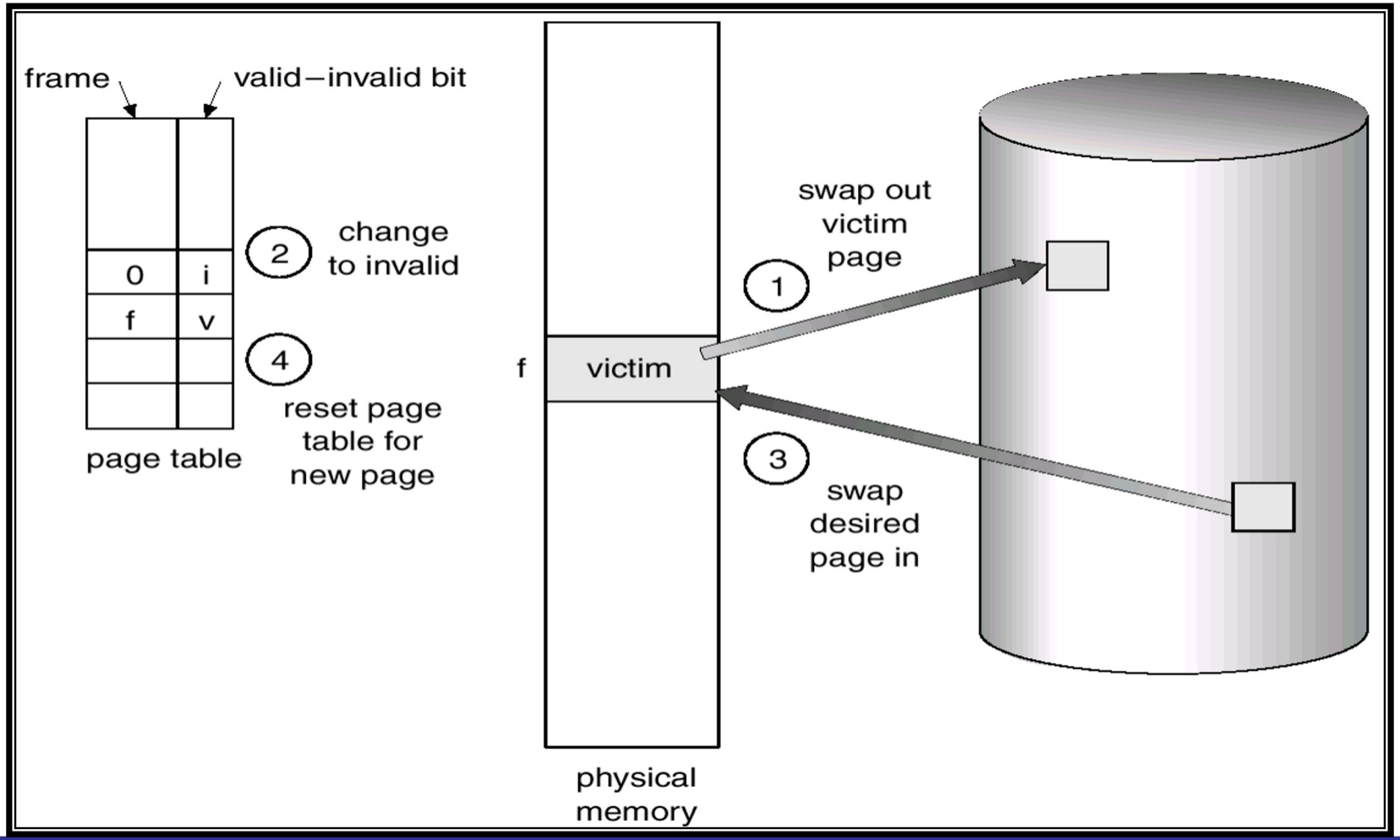
Criação de processos

- Memória virtual permite outros benefícios durante a criação de processos:
  - Cópia-na-escrita (*Copy-on-Write*)
  - Arquivos mapeados em memória (*mapped*)

Substituição de páginas

- Previne super-alocação de memória modificando a rotina de serviço de falha de página para incluir substituição de página
- Usa bit de modificação (*dirty bit*) para reduzir a sobrecarga da transferência de página, só as páginas modificadas são escritas no disco
- Substituição de páginas:
  - Separação entre memória lógica e memória física, assim, uma grande memória virtual pode ser mapeada para uma pequena memória física

Substituição de páginas



Falta de página

1. *Trap* para Sistema Operacional
2. Armazenamento dos registradores e estados
3. Determinação que é uma falta de página
4. Teste de referência e determinação do local no disco
5. Leitura em um *frame* livre
6. Enquanto espera, escalona-se processos
7. Interrupção do disco
8. Armazenamento dos registradores e troca de contexto
9. Determinação de interrupção de disco
10. Correção da tabela de páginas
11. Espera-se pela relocação do processo
12. Continua a instrução interrompida

Troca de página

- Algoritmo Ótimo para troca de páginas:
  - Tirar da memória a página que será usada por último

- Impossível de saber qual é a próxima página que será acessada

Troca de página

- Algoritmo *First-in First-out Page Replacement* (FIFO):
  - Sistema Operacional mantém uma listas das páginas correntes na memória
  - A página no início da lista é a mais antiga e a página no final da lista é a mais nova
  - Simples, mas pode levar a não eficiência, pois uma página que está em uso constante pode ser retirada
  - Pouco utilizado

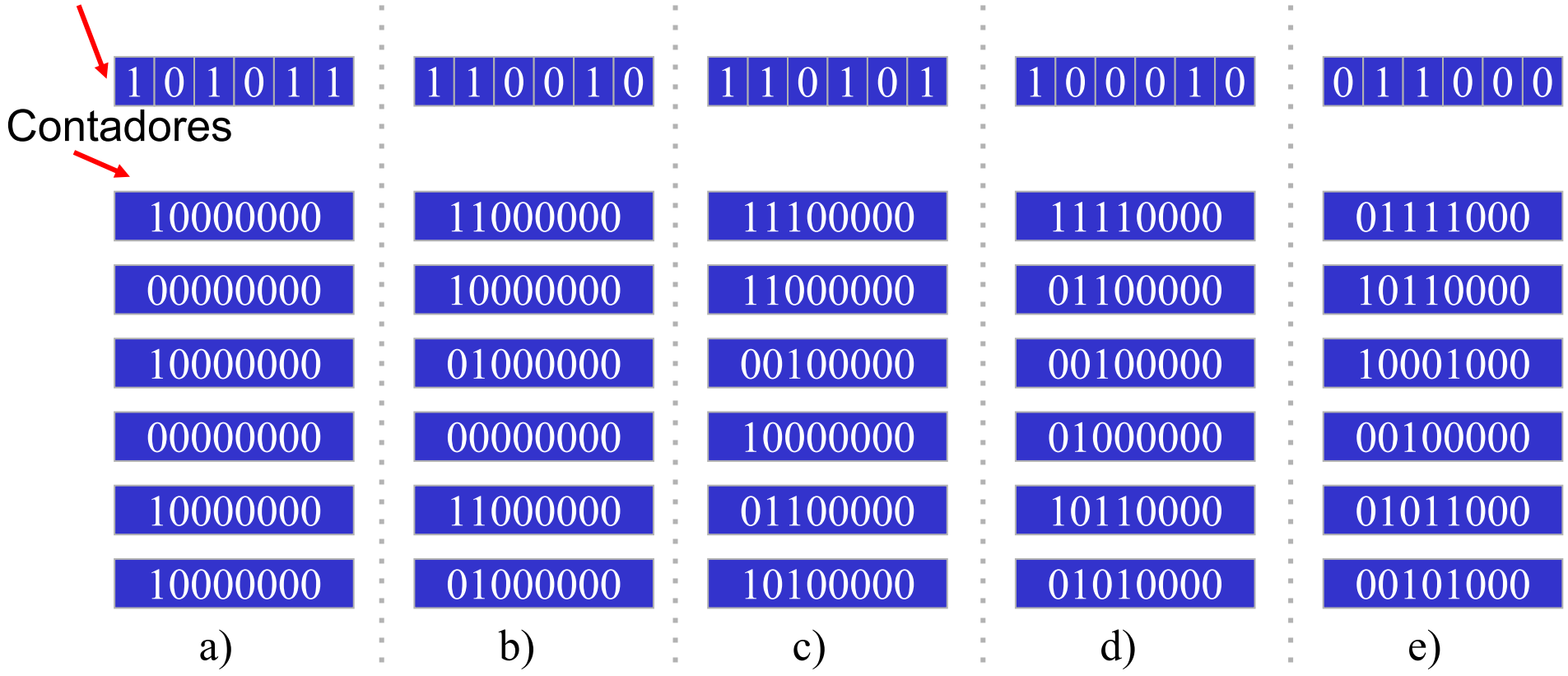
Troca de página

- Algoritmo *Least Recently Used Page Replacement* (LRU):
  - Troca de página menos referenciada/modificada recentemente
  - Alto custo: lista encadeada com as páginas que estão na memória, com as mais recentemente utilizadas no início e as menos utilizadas no final
  - A lista deve ser atualizada a cada referência da memória

Troca de página

- Algoritmo *Aging*: página com menor contador é removida

Bits R para páginas 0-5



Troca de página

- Algoritmo *Working Set*:
  - Geralmente, páginas são carregadas na memória somente quando são necessárias e não antecipadamente – *Demand Paging*
  - *Working set*: conjunto de páginas que um processo está utilizando

Troca de página

- Reduzir a falta de páginas: um processo só é executado quando todas as suas páginas estão carregadas na memória
- O Sistema Operacional deve saber quais páginas estão no *working set*
- Tempo virtual corrente (*Current Virtual Time*):
  - Tempo que o processo efetivamente utiliza a CPU

Localidade de referência

- Tempos típicos de acesso:
  - Memória 40 nanossegundos
  - Disco 3.5 milissegundos
- Se o tempo de acesso a disco pode ser até 100 mil vezes maior, então como a paginação funciona de forma eficiente?
- Princípio de localidade das referências:
  - Processos apresentam um padrão de acesso à memória
  - Segundo esse padrão, cada processo possui conjuntos de páginas que são mais intensamente utilizadas

Thrashing

- CPU Estressada (*Working Set Model*):
  - Examinar as *Working Set Window* páginas mais recentemente referenciadas
- Problemas:
  - W pequeno: não conterá localidade
  - W grande: poderá conter várias localidades
- Sistema Operacional deve controlar o *Working Set* de cada processo
  - Quando a demanda não for atendida, suspende-se processos

Referências Bibliográficas

- SILBERSCHATZ, A.; GALVIN, Peter; GAGNE Greg, Operating System Concepts Essentials. John Wiley & Sons, Inc. 2th edition, 2013.
- TANENBAUM, Andrew S. Sistemas operacionais modernos. 3a. ed. São Paulo: Pearson, 2009-2013. p. 653.
- OLIVEIRA, Rômulo; CARÍSSIMI, Alexandre; TOSCANI, Simão. Sistemas Operacionais. Porto Alegre: Bookman, 4a. ed. 2010.