



# SISTEMAS OPERATIVOS

## Gestão de memória

António Godinho

# OBJETIVOS E BIBLIOGRAFIA

Descrever as várias formas de **organização** de memória

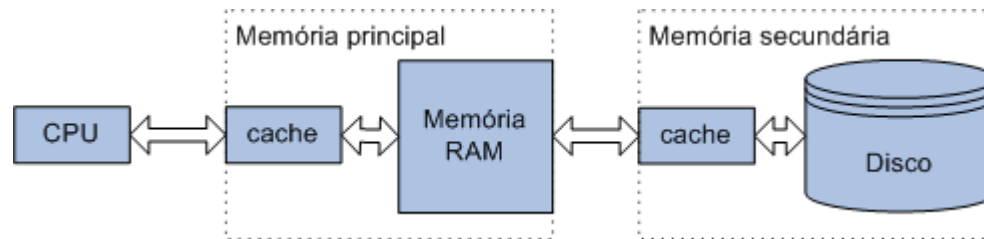
Discutir várias técnicas de **gestão** de memória

“Operating System Concepts”. Abraham Silberschatz, Peter B. Galvin, Greg Gagne, 9th edition, 2014, Capítulo 8

**NOTA: Os acetatos que se seguem não substituem a bibliografia aqui referida, e deverão por isso ser vistos apenas como um complemento para o estudo da matéria.**

# GESTÃO DA MEMÓRIA

- **Organização da memória (hierarquia de memórias):**
  - Memória principal (memória física ou central): organizada sob a forma de tabela unidimensional, em que cada elemento da tabela é endereçável (acessível) pela CPU (acesso aleatório); armazena código e dados dos processos; volátil; tempos de acesso reduzidos (+ rápidos); custo elevado; pode incluir memórias cache.
  - Memória secundária: memória em disco; acessível através de operações de E/S (por blocos); persistente; tempos de acesso elevados (+ lentos); custo reduzido; pode incluir memórias cache.



# GESTÃO DA MEMÓRIA

- Espaço de endereçamento de um processo: conjunto de posições de memória que o programa em execução pode aceder.
- **Objectivos:**
  - Gerir o espaço de endereçamento dos processos, ou seja, o espaço de memória alocada ao processo.
  - Assegurar que cada processo dispõe do espaço de memória que necessita.
  - Garantir a protecção no acesso à memória, limitando o acesso dos processos ao seu espaço de endereçamento (ex. garantir que os processos não acessem o espaço de endereçamento do SO).
  - Minimizar as transferências de informação entre memórias (primária, secundária) do sistema computacional. As informações são repartidas ou replicadas por diversos níveis da memória de modo a, simultaneamente, maximizar o número de acessos às memórias mais rápidas e minimizar a ocupação destas.

# GESTÃO DA MEMÓRIA

- Espaço de endereçamento de um processo: conjunto de posições de memória que o programa em execução pode aceder.
- **Objectivos:**
  - Gerir o espaço de endereçamento dos processos, ou seja, o espaço de memória alocada ao processo.
  - Assegurar que cada processo dispõe do espaço de memória que necessita.
  - Garantir a protecção no acesso à memória, limitando o acesso dos processos ao seu espaço de endereçamento (ex. garantir que os processos não acessem o espaço de endereçamento do SO).
  - Minimizar as transferências de informação entre memórias (primária, secundária) do sistema computacional. As informações são repartidas ou replicadas por diversos níveis da memória de modo a, simultaneamente, maximizar o número de acessos às memórias mais rápidas e minimizar a ocupação destas.

# GESTÃO DA MEMÓRIA – ENDEREÇAMENTO REAL E VIRTUAL

- Programa fonte: referências simbólicas (ex. identificadores das variáveis)
- Programa executável:
  - Criado pelo compilador e linker;
  - Tipos de endereçamento:
    - **Endereçamento real:**
      - Não relocáveis: endereçamento físico; colocação estática; o programa tem de ser executado sempre na mesma posição de memória (ex. ficheiros .COM em MS/DOS).
      - Relocáveis: endereços gerados com referência a um endereço base (endereçamento baseado); aquando do carregamento do programa, é definido o endereço base que determina a localização do programa na memória física; a localização do programa é mantida até ao final da sua execução; relocação e alocação estática.
    - **Endereçamento virtual:**
      - os endereços virtuais são traduzidos dinamicamente em endereços reais durante a execução do programa; a localização do programa pode variar ao longo da sua execução; relocação e alocação dinâmica.

# GESTÃO DA MEMÓRIA – ENDEREÇAMENTO REAL E VIRTUAL

Hardware básico para endereçamento real relocável (endereçamento baseado). A geração dos endereços reais é realizada pela unidade de gestão de memória da CPU (MMU Memory Management Unit). Em sistemas multiprogramados, esta forma de endereçamento oferece, ainda, um mecanismo de protecção do espaço de endereçamento dos processos.

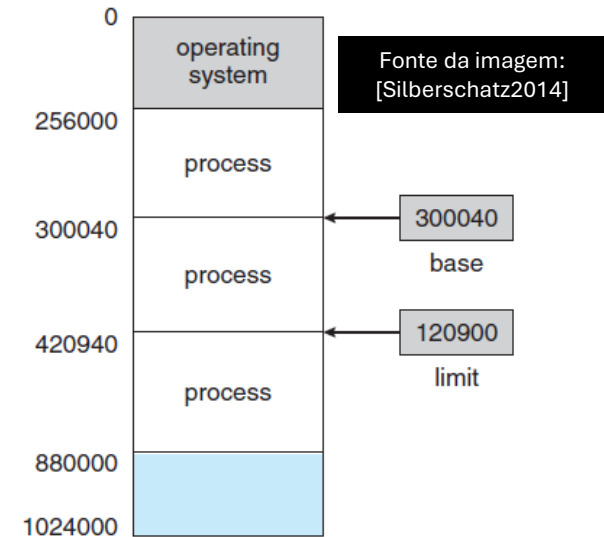


Figure 8.1 A base and a limit register define a logical address space.

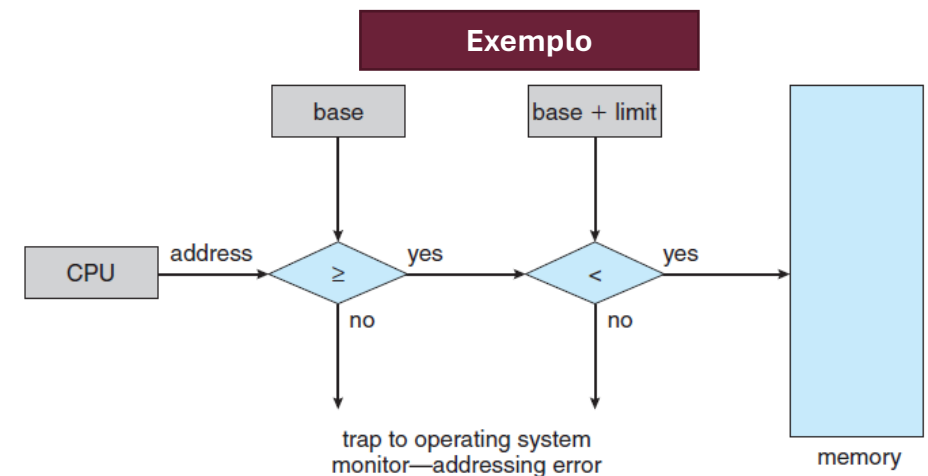


Figure 8.2 Hardware address protection with base and limit registers.

# GESTÃO DA MEMÓRIA - ENDEREÇAMENTO REAL: SISTEMAS MONOPROGRAMADOS

## Sistemas monoprogramados:

- Apenas existe um programa em execução que tem acesso total à memória principal (memória física).
- O espaço de endereçamento do processo está limitado ao espaço de endereçamento físico.
- Alocação contígua de memória.

## Mecanismo de sobreposição(overlay):

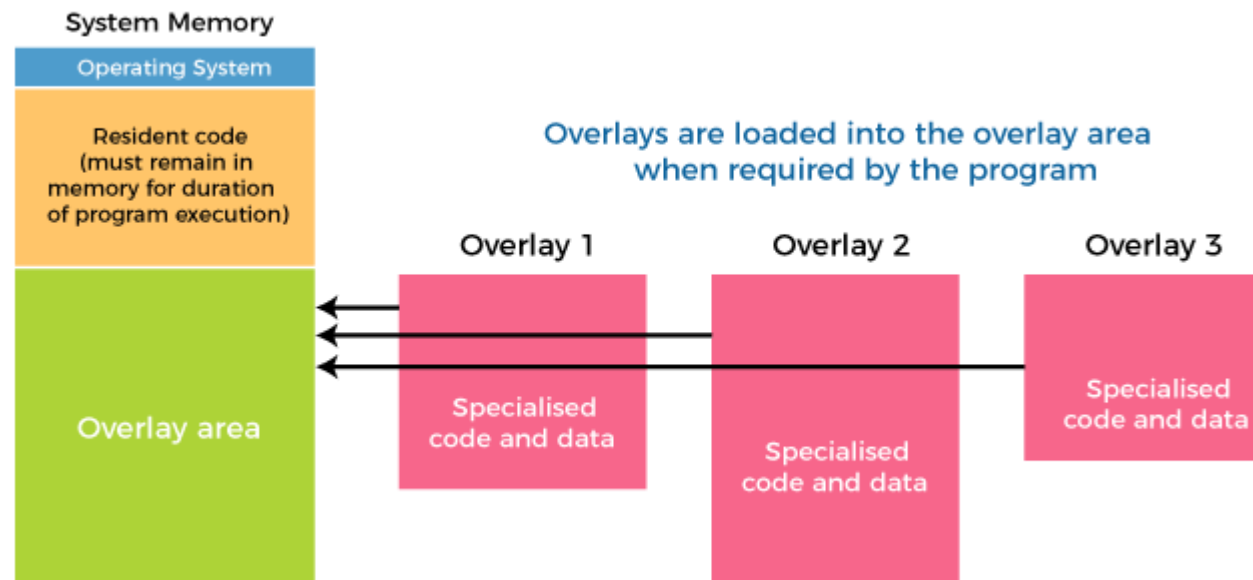
- Permite executar programas com dimensão superior à memória física.
- Continuam a ter de ser dimensionados para a memória física disponível.
- Mantém-se em memória apenas as instruções e os dados que são necessários em determinado momento.

# GESTÃO DA MEMÓRIA - ENDEREÇAMENTO REAL: SISTEMAS MONOPROGRAMADOS

## Mecanismo de sobreposição(overlay):

São geridos pelo utilizador. O programador indica explicitamente quando deve ser carregado um novo overlay. O SO não oferece qualquer suporte a este mecanismo.

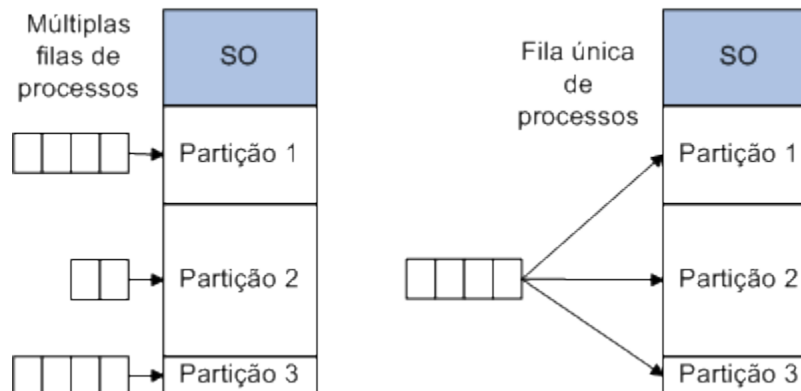
Dificuldades: certos programas não são facilmente organizáveis em overlays; pode condicionar o desempenho se as trocas de overlays ocorrerem com muita frequência; dificulta a manutenção e atualização do programa.



# GESTÃO DA MEMÓRIA - ENDEREÇAMENTO REAL: SISTEMAS MULTIPROGRAMADOS

## Partições com dimensão fixa:

- A memória principal é dividida em partições de dimensão fixa.
- Para cada partição o SO mantém uma fila de espera dos processos prontos a executar. Sempre que um processo fica bloqueado, o SO faz o despacho do processo em fila de espera associada a outra partição. Em alternativa, poderá ser utilizada apenas uma fila de espera sendo selecionada, aquando do despacho, a partição que melhor se adapta ao processo.
- Alocação contígua de memória.
- Programas relocáveis com recurso ao endereçamento baseado.
- Protecção garantida pelo endereçamento baseado.



# GESTÃO DA MEMÓRIA - ENDEREÇAMENTO REAL: SISTEMAS MULTIPROGRAMADOS

## Partições com dimensão fixa:

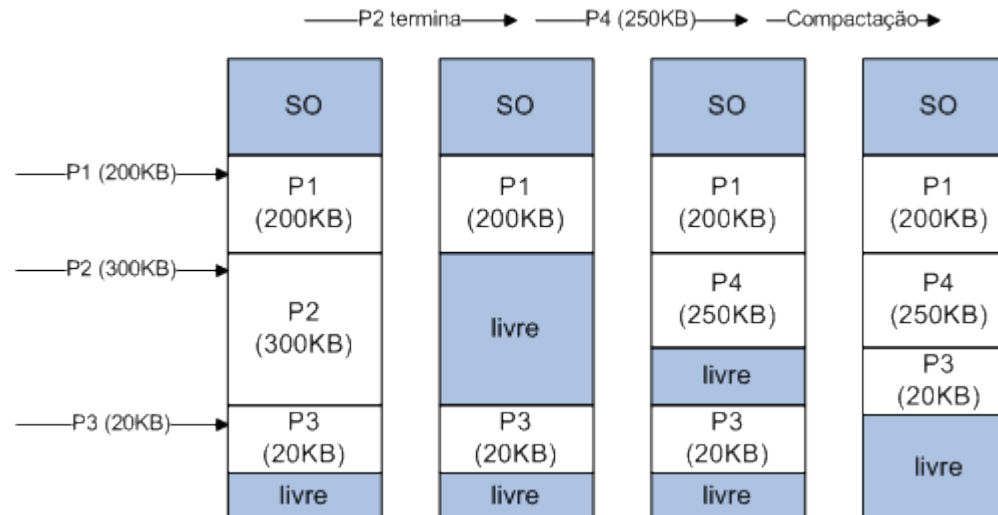
### Desvantagens:

- Qual o número de partições e a sua dimensão?
- Fragmentação interna: a dimensão dos processos não coincide exactamente com a dimensão da partição.
- Poderão existir filas de espera vazias para algumas partições.
- Dimensão do programa limitada à dimensão da maior partição, podendo, no entanto, recorrer ao mecanismo de overlays para suprir esta restrição.

# GESTÃO DA MEMÓRIA - ENDEREÇAMENTO REAL: SISTEMAS MULTIPROGRAMADOS

## Partições com dimensão variável:

- A dimensão da partição é ajustada à dimensão do programa, evitando, assim, a fragmentação interna. No entanto, à medida que os processos vão terminando, ficam espaços de memória livres (fragmentação externa).
- Alocação contígua de memória.
- Programas relocáveis com recurso ao endereçamento baseado.
- Protecção garantida pelo endereçamento baseado.



# GESTÃO DA MEMÓRIA - ENDEREÇAMENTO REAL: SISTEMAS MULTIPROGRAMADOS

## Partições com dimensão variável:

### Desvantagens:

- Fragmentação externa: muitas partições não contíguas de dimensão reduzida.
- Necessário proceder periodicamente à compactação de memória.
- Necessário manter informação sobre as partições livres.
- Algoritmos de alocação de memória (selecção da partição livre a alocar ao processo): first fit; best fit; worst fit.

# GESTÃO DA MEMÓRIA – ENDEREÇAMENTO VIRTUAL

- Objetivo: Dissociar os endereços gerados pelo programa (endereços virtuais) dos endereços físicos da memória principal.
- Programador dispõe de um espaço de endereçamento independente e maior do que a memória física.
- A conversão dos endereços virtuais em endereços reais é realizada pela unidade de gestão de memória da CPU (**MMU Memory Management Unit**).
- Espaço de endereçamento virtual dividido em blocos contíguos:
- Endereço virtual = (bloco, deslocamento)
- Tipo de blocos:
  - Blocos de dimensão variável(segmentos):segmentação.
  - Blocos de dimensão fixa (páginas): paginação.

# ENDEREÇOS LÓGICOS VS FÍSICOS

O conceito de espaço de endereçamento **lógico** ligado a um espaço separado de endereçamento **físico** é central na gestão de memória

Endereço lógico (ou virtual) é aquele gerado pelo CPU; o endereço físico é aquele que é visto pela unidade de memória

endereços lógico e físico são **idênticos** se a atribuição de endereços for feita em tempo de compilação ou de carregamento...

... mas são **diferentes** se for feita em tempo de execução

A MMU (**Memory-Management Unit**) é o hardware que trata dessa

**conversão:**

- endereço lógico → endereço físico
- os programas de utilizador só lidam com endereços lógicos

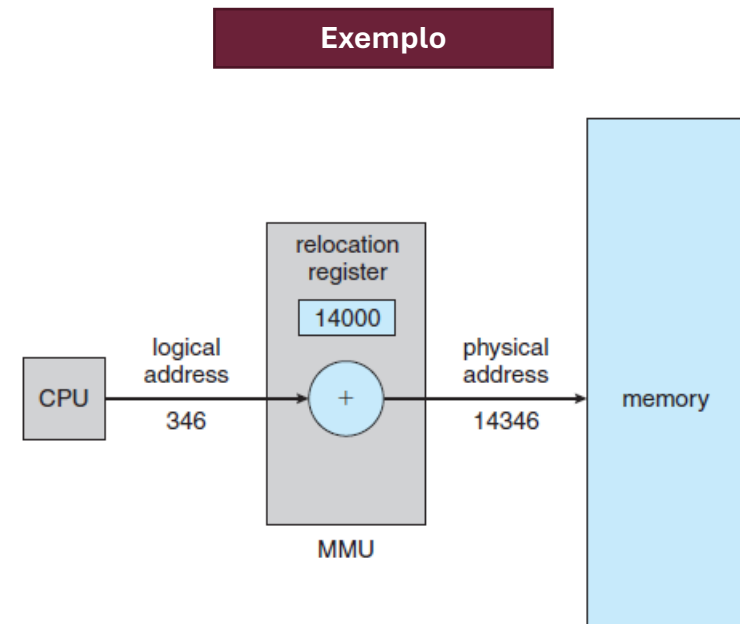
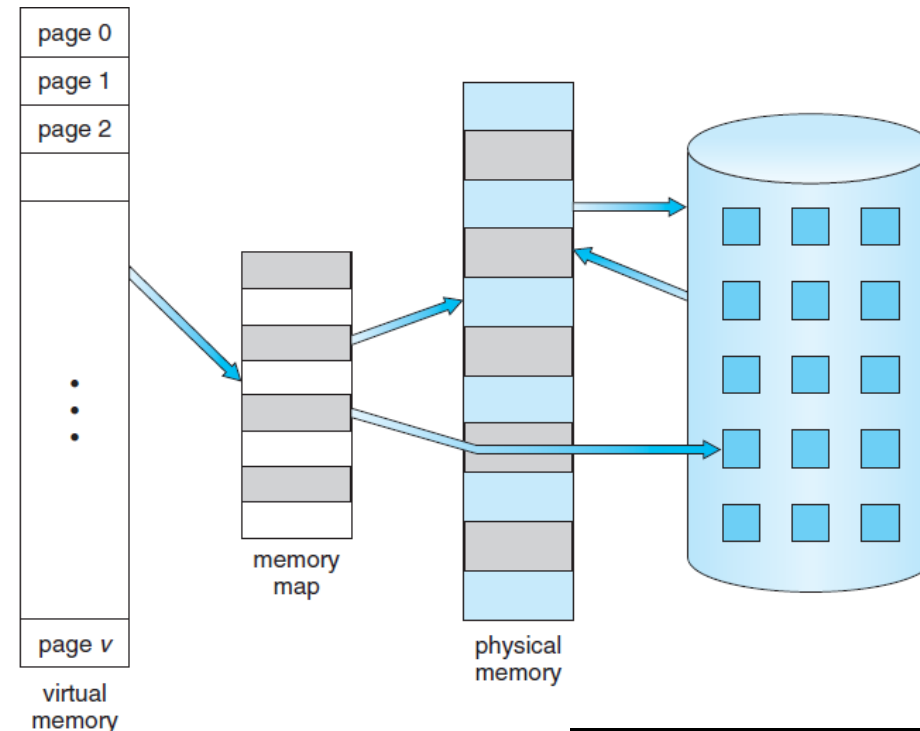


Figure 8.4 Dynamic relocation using a relocation register.

# GESTÃO DA MEMÓRIA - ENDEREÇAMENTO VIRTUAL

- Apenas são mantidos em memória física os blocos necessários a cada processo. Os restantes blocos são mantidos em memória secundária (swapping).
- Permite a alocação não contígua de memória.



# GESTÃO DA MEMÓRIA - ENDEREÇAMENTO VIRTUAL

Separação de memória lógica da memória física

O espaço de endereçamento lógico dum processo **pode ser maior do que a memória física**

A soma dos espaços de endereçamento de todos os processos pode ultrapassar a memória física.

Apenas a parte “ativa” (working set) do programa precisa de estar em memória – mais eficaz!

Fornece mecanismos de simplificação de gestão de memória e proteção

## Implementações

- Demand paging
- Demand Segmentation

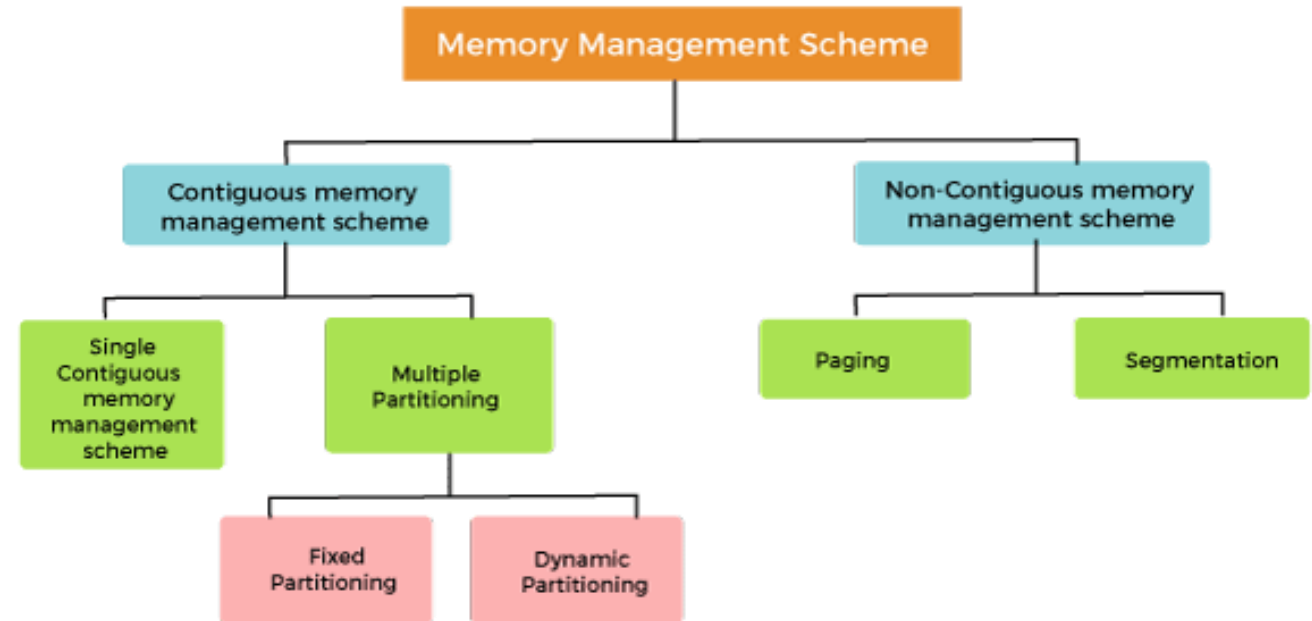
# GESTÃO DE MEMÓRIA CONTIGUA

- A memória principal tem habitualmente duas **partições**:
  - o SO residente, normalmente colocado em endereços baixos de memória, para ficar próximo do vetor de interrupções
  - os processos do utilizador
- Quando se usa o mecanismo de reserva contígua de memória cada processo é colocado **numa única secção** de memória, que é **contígua** à secção de memória de outro processo
- A **proteção e mapeamento** são feitos usando:
  - um **registo limite** que define gama de endereços lógicos
  - um **registo base** (de relocação) com o valor do endereço físico mais baixo

# TÉCNICAS DE GESTÃO DE MEMÓRIA

- **SO** tem tabela com informação sobre os blocos de memória **ocupados e livres**
- **Técnicas de gestão de memória** são métodos utilizados pelo sistema operativo para alocar, utilizar e gerir de forma eficiente os recursos de memória para os processos.
- Várias técnicas ajudam o sistema operativo a gerir a memória de forma eficaz.
- A alocação/reserva de memória ocorre na criação (reserva de memória para código, dados e pilha), ou no pedido de extensão do espaço de endereçamento (área de dados ou pilha). A terminação de processos liberta a memória.

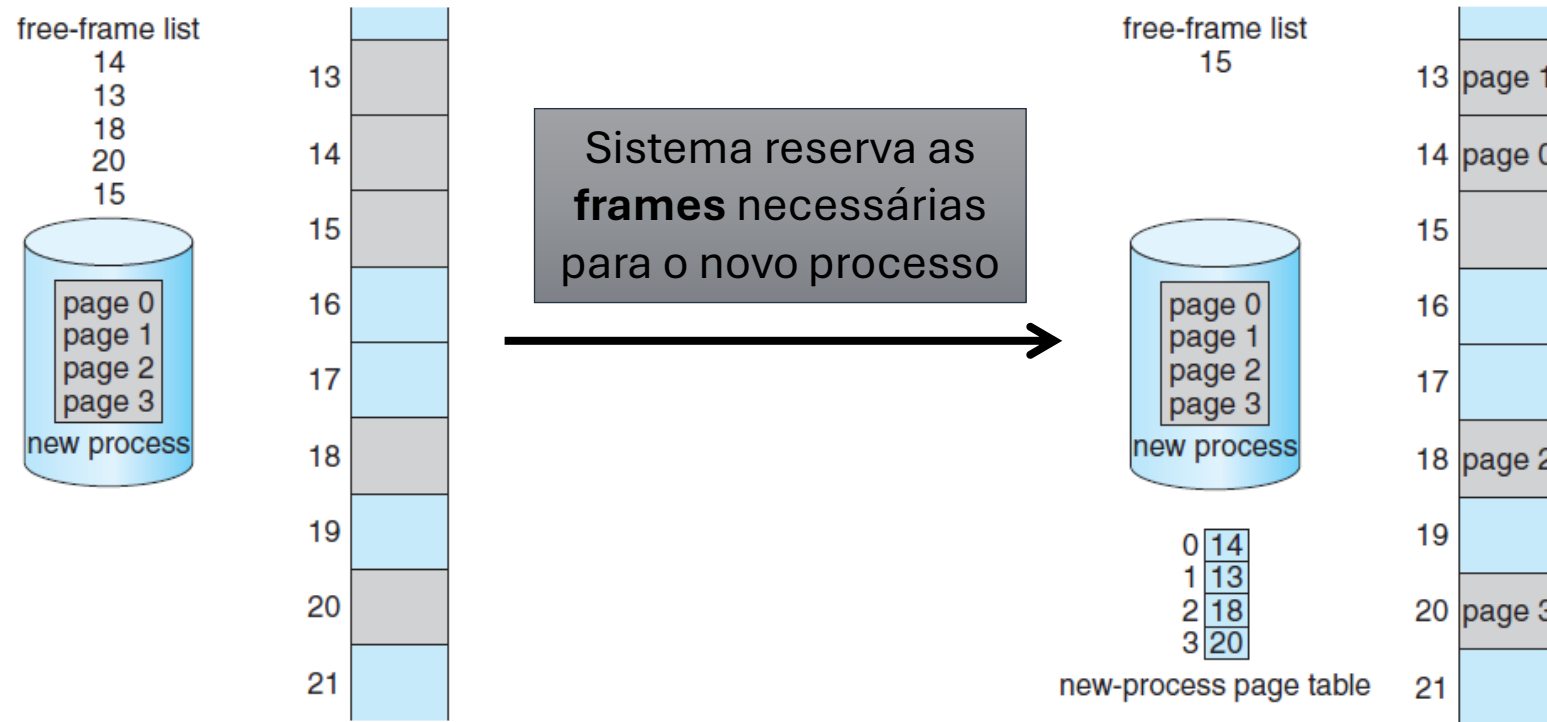
# GESTÃO DE MEMÓRIA VIRTUAL: ALOCAÇÃO / RESERVA



Classification of memory management schemes

- Alocação de páginas (blocos de dimensão fixa): qualquer página de entre as que estão livres.
- Alocação de segmentos (blocos de dimensão variável):
  - Análise da lista de blocos livres para encontrar um com dimensão igual ou superior à desejada.
  - Escolher o bloco livre que provoque menor fragmentação externa.
  - Alocar parte ou totalidade do bloco escolhido de acordo com uma dimensão mínima para o fragmento gerado. Esta dimensão mínima gera fragmentação interna mas reduz a lista de blocos livres.

# RESERVA DE FRAMES



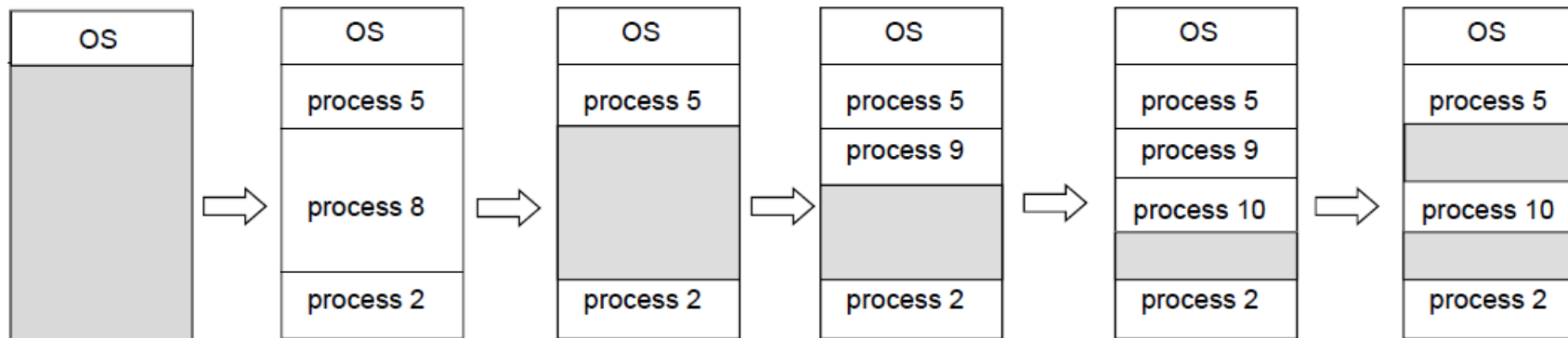
## ▪Em suma:

- o utilizador vê a memória como um **espaço único**, contíguo, mas na realidade o programa está **“espalhado”** pela memória física
- hardware de tradução de endereços **“reconcilia”** a memória lógica e a física

# RESERVA CONTÍGUA DE MEMÓRIA

- **SO tem tabela com informação sobre os blocos de memória ocupados e livres**

- no início toda a memória está disponível e é considerada um único bloco livre – um grande “buraco” (hole)
- os processos vão sendo colocados em memória enquanto existirem buracos onde eles “caibam”
- à medida que os processos terminam o seu bloco fica livre, deixando um buraco, em que os buracos adjacentes “unem-se” para formar buracos maiores



- **Algoritmos de reserva de memória**

- First-fit – escolhe o primeiro buraco livre (o + rápido)
- Best-fit – escolhe o buraco menor com dimensão suficiente
- Worst-fit – escolhe o buraco maior

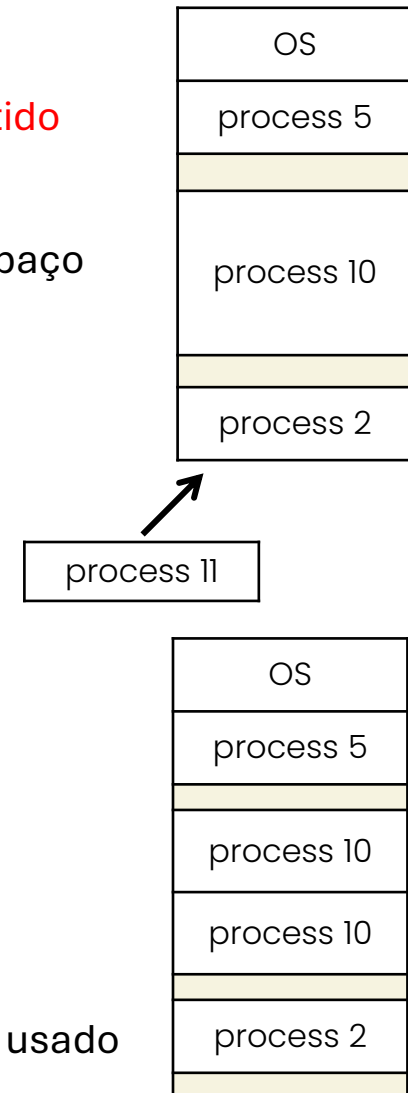
# PROBLEMA: FRAGMENTAÇÃO

## Fragmentação **externa**

- à medida que os processos são carregados e removidos da memória, o espaço livre fica **partido** em buracos pequenos
- pode chegar-se a um ponto em que existe memória livre para satisfazer um pedido **mas** o espaço não é contíguo
- possível solução (muito dispendiosa): **compactação**
  - “baralham-se” os buracos de forma a colocá-los todos juntos, formando um **grande** bloco livre
  - só possível quando a relocação é dinâmica (endereços atribuídos em tempo de execução)
- **Solução melhor**: o espaço de memória de um processo não ter de ser contíguo (paginação segmentação)

## ▪ Fragmentação **interna**

- muitas vezes a memória é dividida em partes de **igual** dimensão, para evitar buracos muito pequenos
- neste caso a reserva é feita em blocos da mesma dimensão
- a memória reservada para um processo pode ser superior à necessária, e o **excedente** não é usado

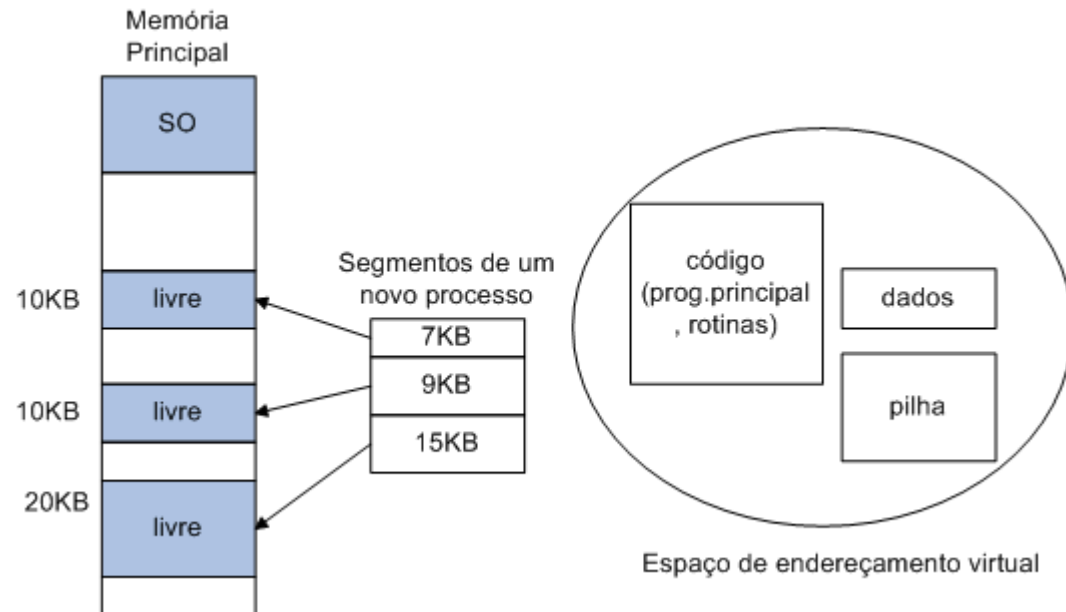


# FRAGMENTAÇÃO - SOLUÇÕES

- Uma possível solução para o problema da fragmentação externa é permitir que o espaço de endereço lógico dos processos não seja contíguo, permitindo assim que um processo seja alocado na memória física onde quer que essa memória esteja disponível.
- Duas técnicas complementares alcançam essa solução: **segmentação** e **paginação**.
- Estas técnicas também podem ser combinadas.

# ENDEREÇAMENTO VIRTUAL: SEGMENTAÇÃO

- Divisão do espaço de endereçamento em segmentos (blocos de dimensão variável).
- Divisão dos programas em segmentos lógicos que reflectem a sua estrutura funcional: rotinas, módulos, código, dados, pilha, etc.
- O espaço de endereçamento virtual é linear dentro de cada segmento.
- Dimensão dos segmentos limitada pela arquitectura.
- Suporte directo das abstracções comuns nas linguagens de programação:
  - Carregamento em memória: considera-se que todas as palavras dentro do segmento têm a mesma probabilidade de serem acedidas;
  - Protecção: aplicada ao segmento como bloco;
  - Eficiência: princípio da localidade de referência.



# ENDEREÇAMENTO VIRTUAL: SEGMENTAÇÃO

- Endereço virtual: (segmento, deslocamento)

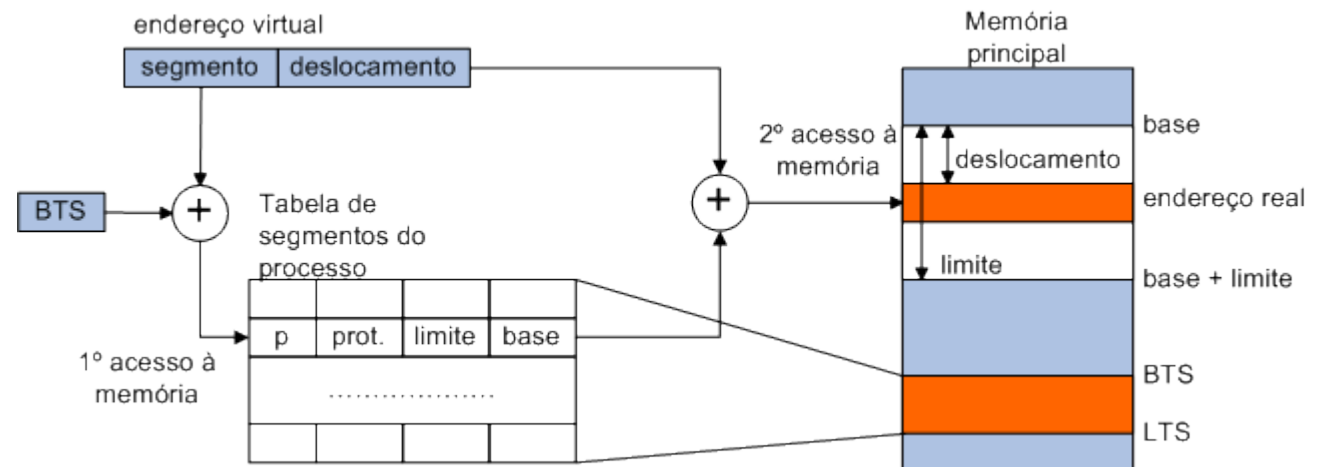
- Determinação do endereço real (MMU)

- Tabela de segmentos:**

- mapeia endereços lógicos (2D) em endereços físicos (1D)

- armazenada na memória principal entre o endereço BTS (base da tabela de segmentos) e LTS (limite da tabela de segmentos). O bit p indica se o segmento está presente na memória principal. Se não estiver, é gerada uma exceção (falta de segmento) que permitirá ao SO carregar o segmento da memória secundária para a memória principal.

- accedida apenas a primeira vez em que um segmento é necessário, em acessos subsequentes, o endereço base é guardado em registos internos à CPU (ex. Intel x86 registos DS, CS, SS, ES).



- Cada entrada da tabela contém:

- base – endereço inicial da memória física onde o segmento reside
  - limit – tamanho do segmento

# ENDEREÇAMENTO VIRTUAL: SEGMENTAÇÃO

## Proteção:

- Processos diferentes têm tabelas de segmentos diferentes. Os espaços de endereçamento virtual são distintos (inacessível a outros processos);
- Verificação do limite de endereçamento dentro de cada um dos segmentos, comparando o deslocamento com esse limite;
- Verificação do tipo de acesso ao segmento: leitura, escrita e execução;

## Partilha de memória entre processos:

- basta colocar nas tabelas de segmentos dos processos o endereço real do segmento a partilhar;
- Os endereços virtuais usados para aceder ao segmento partilhado podem ser diferentes nos vários processos;
- A protecção de um segmento partilhado é definida para cada processo através da respetiva tabela de segmentos.
- Os segmentos têm tamanhos variáveis pelo que os algoritmos de alocação da memória são complicados.
- Fragmentação externa: devido à alocação e libertação de espaços de dimensão variável.
- Os programadores têm de ter uma noção mínima do modelo de endereçamento.
- Partilha de código e dados entre utilizadores facilitada.
- Crescimento dinâmico de segmentos facilitado (alargamento do espaço de endereçamento).
- A protecção de código e dados pode ser realizada separadamente.

# ENDEREÇAMENTO VIRTUAL: PAGINAÇÃO

- O espaço de endereçamento físico de um processo **não é** contíguo.

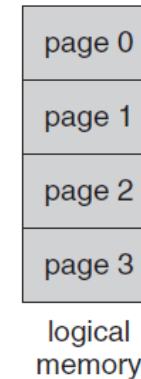
Paginação consiste em dividir um programa em blocos menores de tamanho fixo, chamados páginas.

## ▪ Método base

- memória física dividida em blocos de **tamanho fixo** (frames)
  - tamanho é potência de 2 (facilita tradução; ver slide a seguir)
- memória lógica é dividida em blocos **do mesmo tamanho** (páginas)
- para correr um programa de tamanho igual a n páginas, encontrar n frames disponíveis (onde quer que se encontrem)
- usar uma tabela de páginas **traduz** os endereços lógicos para endereços físicos

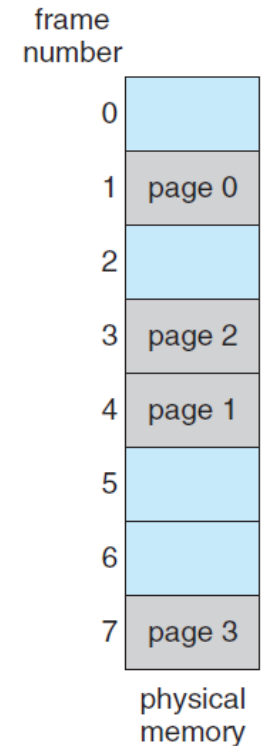
▪ Ao usar a memória que está disponível, evita-se a fragmentação externa e a necessidade de compactação

- não se evita no entanto a fragmentação interna



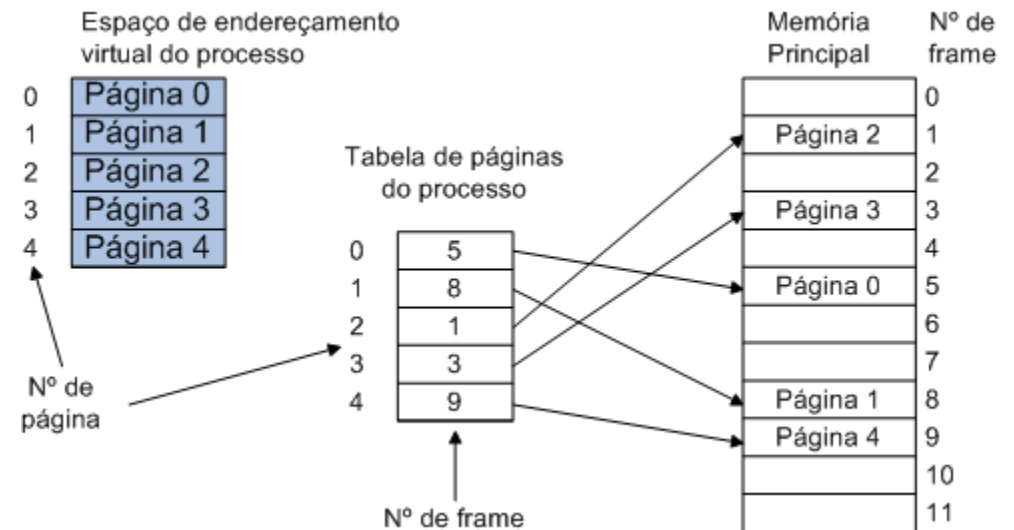
0	1
1	4
2	3
3	7

page table



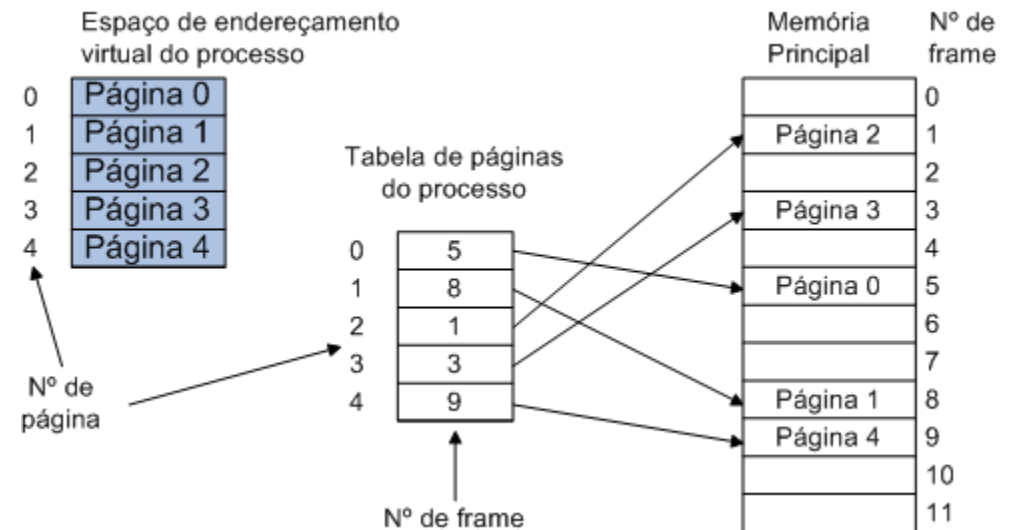
# ENDEREÇAMENTO VIRTUAL: PAGINAÇÃO

- Espaço de endereçamento virtual linear de dimensão superior à da memória principal: Ao contrário da segmentação, o programador não tem em consideração aspectos relacionados com a gestão de memória.
- Divisão do espaço de endereçamento do processo em páginas (blocos de dimensão constante);
- Divisão da memória principal em quadros (frames) de dimensão igual às páginas.
- Em memória principal, o SO mantém apenas algumas páginas do programa, sendo as restantes carregadas da memória secundária sempre que necessário (falta de página).
- A dimensão das páginas (constante) é normalmente muito menor que a da memória principal (valores típicos de 512 a 8192 bytes) e influencia:
  - a fragmentação interna;
  - o número de faltas de páginas;
  - a dimensão das tabelas de páginas e listas de frames mantidas pelo SO.



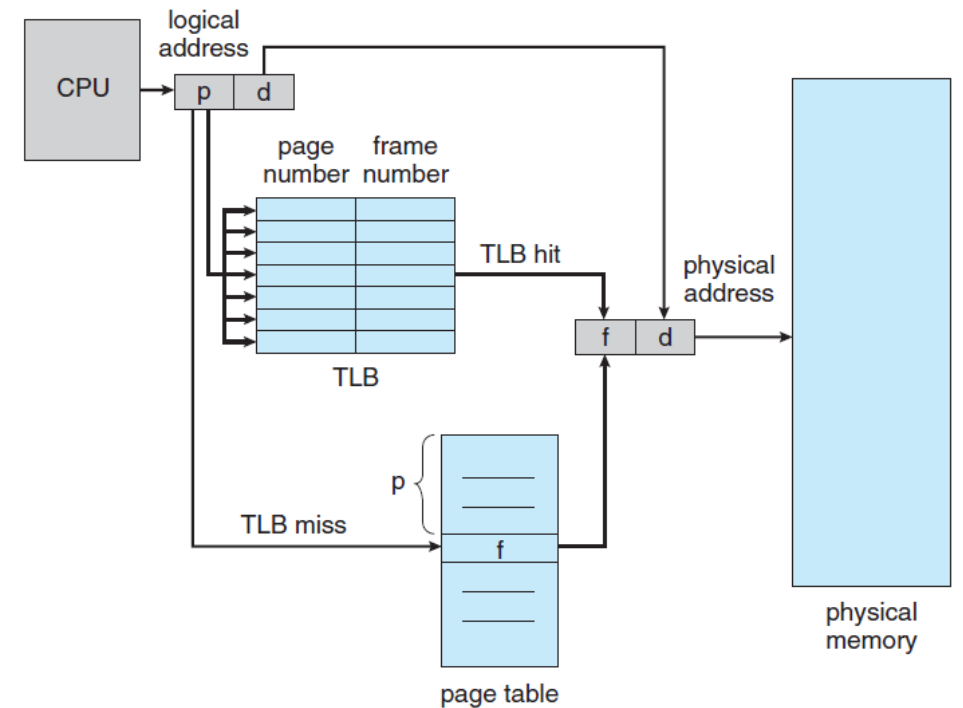
# ENDEREÇAMENTO VIRTUAL: PAGINAÇÃO

- Espaço de endereçamento virtual linear de dimensão superior à da memória principal: Ao contrário da segmentação, o programador não tem em consideração aspectos relacionados com a gestão de memória.
- Divisão do espaço de endereçamento do processo em páginas (blocos de dimensão constante);
- Divisão da memória principal em quadros (frames) de dimensão igual às páginas.
- Em memória principal, o SO mantém apenas algumas páginas do programa, sendo as restantes carregadas da memória secundária sempre que necessário (falta de página).
- A dimensão das páginas (constante) é normalmente muito menor que a da memória principal (valores típicos de 512 a 8192 bytes) e influencia:
  - a fragmentação interna;
  - o número de faltas de páginas;
  - a dimensão das tabelas de páginas e listas de frames mantidas pelo SO.



# ENDEREÇAMENTO VIRTUAL: PAGINAÇÃO

- Um método simples seria manter as tabelas de páginas **em memória** (uma por processo), sendo necessários dois registos
  - PTBR (Page-table base register), **apontador** para a tabela de páginas
  - PRLR (Page-table length register), com o **tamanho** da tabela
  - cada acesso aos dados ou instruções requer **dois** acessos à memória (um para a tabela de paginação; outro para os dados/instrução), logo acesso à memória é **duas vezes** mais lento!
- Possível solução?
  - usar uma **cache** (em hardware) que permita buscas rápidas: um **translation look-aside buffer (TLB)**
  - a cache é **pequena** por isso só se guardam algumas entradas da tabela
  - quando se gera um endereço lógico procura-se **primeiro no TLB**, o que é muito rápido
    - é memória associativa, permitindo comparação **simultânea** entre todas as entradas
    - o TLB lookup faz parte do pipeline de instruções, não havendo por isso qualquer redução de desempenho
  - Se não estiver na TLB (TLB miss) tem de se procurar na tabela que está em memória

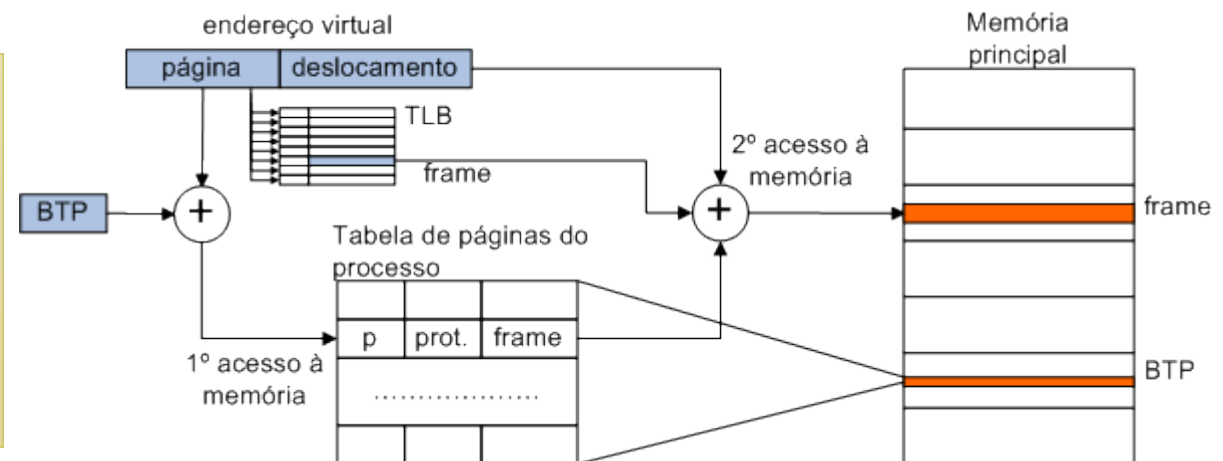


# ENDEREÇAMENTO VIRTUAL: PAGINAÇÃO

## ▪Recurso a memória TLB - Translation Lookaside buffer

- TLB - é uma cache da CPU usada para acelerar o processo de tradução dos endereços virtuais
- Basicamente, mapeia endereços virtuais em endereços físicos
- A memória TLB permite que cada um dos seus elementos seja pesquisado em simultâneo. Dado um valor da página, este é comparado, em simultâneo, com todos os elementos da tabela, devolvendo, em caso de sucesso, o endereço base da referida página (frame).
- A chave de pesquisa é o endereço virtual e o resultado da pesquisa é o endereço físico
- Se o endereço não existir na TLB, a tradução usa a tabela de páginas (mais lenta no acesso) – nalguns sistemas pode estar na memória secundária, o que a torna ainda mais lenta

- Endereço virtual: (página, deslocamento)
- Determinação do endereço real (MMU):
  - O bit p indica se a página está em memória principal. Se não estiver é gerada uma exceção (falta de página) que permitirá ao SO carregar essa página da memória secundária para a memória principal.



# PAGINAÇÃO – PAGE FAULT

**Todos os processadores atuais (desktop e servidor) utilizam TLB**

- Quando um processo tenta aceder a uma página e essa página não está em memória ocorre uma page fault
  - A MMU gera uma exceção, que é processada pelo page fault handler:
    - O page fault handler consulta uma outra estrutura de dados (address map do processo) para determinar:
      - se a referência é válida: protecção;
      - e, em caso afirmativo, onde se encontra a página.
    - O handler obtém uma frame livre.
    - Transfere a página do disco para a frame.
    - Atualiza a page table.
- A CPU reinicia a execução da instrução que causou a page fault:
  - mais fácil dizer do que fazer, mas tem a ver com o Hardware

# GESTÃO DE MEMÓRIA – ADDRESS MAP

- É a estrutura de dados do SO que mapeia espaço no disco:
  - ficheiros;
  - partes da área de swap no espaço de endereçamento dum processo.
- É usada pelo page-fault handler para determinar:
  - a validade da referência;
  - em caso positivo, qual a fonte da página referenciada
- Quando o SO cria um processo:
  - mapeia o ficheiro com o código correspondente no espaço de endereçamento desse processo;
  - reduz o tempo de arranque dum processo.
- Transfere algumas páginas desse ficheiro para a memória (prepaging):
  - evita uma page fault rate excessiva inicialmente.

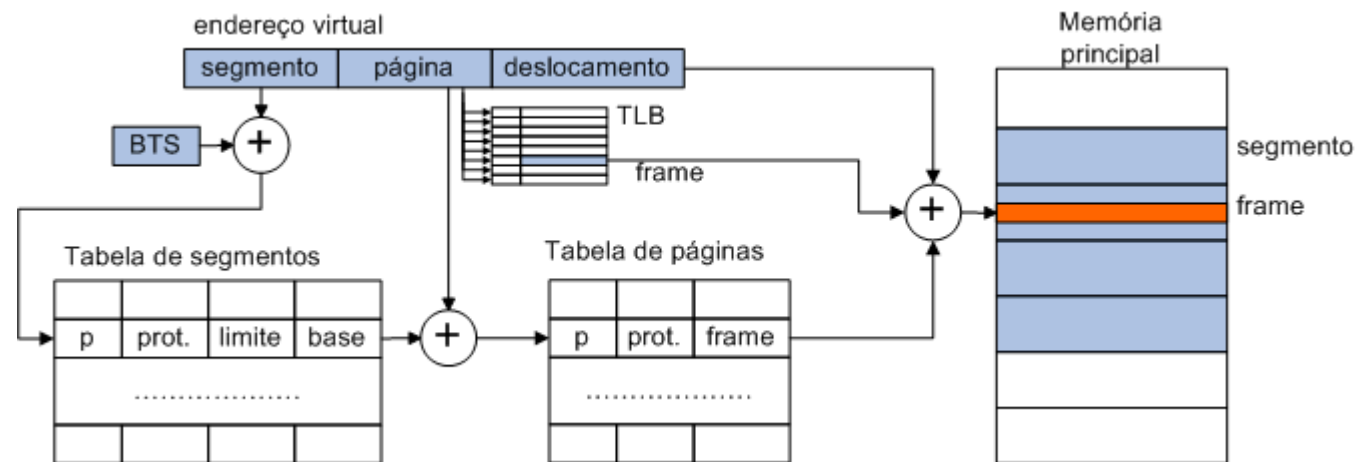
# GESTÃO DE MEMÓRIA

## E se não houver qualquer frame livre?

- Se, quando duma page-fault, todas as frames estiverem ocupadas, o SO tem que libertar uma:
  - se a frame seleccionada tiver sido modificada, terá que ser transferida para o disco
- Por razões de eficiência, o SO tipicamente executa um processo (pageout daemon) que procura manter um certo número de frames livres.
- Em qualquer dos casos, põe-se o problema:
  - Que páginas transferir para o disco?  
É bom que se faça uma boa escolha, doutro modo o desempenho sofre . . .
- Note-se que este problema é comum a qualquer tipo de cache.

# GESTÃO DE MEMÓRIA – SEGMENTAÇÃO/PAGINAÇÃO

- Endereço virtual: (segmento, página, deslocamento)
- Determinação do endereço real (MMU):



# GESTÃO DE MEMÓRIA VIRTUAL

- Organização da memória em blocos: segmentação, paginação ou segmentação/paginação
- Funções do SO:
  - Alocação/reserva: onde alocar um bloco de informação.
  - Transferência: quando transferir um bloco da memória secundária para memória primária e vice-versa.
  - Substituição: quando não existe mais memória livre, qual o bloco a transferir da memória principal para a memória secundária para satisfazer um pedido?

# GESTÃO DE MEMÓRIA VIRTUAL: ALOCAÇÃO/RESERVA(1)

- A alocação/reserva de memória ocorre na criação (reserva de memória para código, dados e pilha), ou no pedido de extensão do espaço de endereçamento (área de dados ou pilha). A terminação de processos liberta a memória.
- Alocação de páginas (blocos de dimensão fixa): qualquer página de entre as que estão livres.
- Alocação de segmentos (blocos de dimensão variável):
  - Análise da lista de blocos livres para encontrar um com dimensão igual ou superior à desejada.
  - Escolher o bloco livre que provoque menor fragmentação externa.
  - Alocar parte ou totalidade do bloco escolhido de acordo com uma dimensão mínima para o fragmento gerado. Esta dimensão mínima gera fragmentação interna mas reduz a lista de blocos livres.

# GESTÃO DE MEMÓRIA VIRTUAL: ALOCAÇÃO/RESERVA(2)

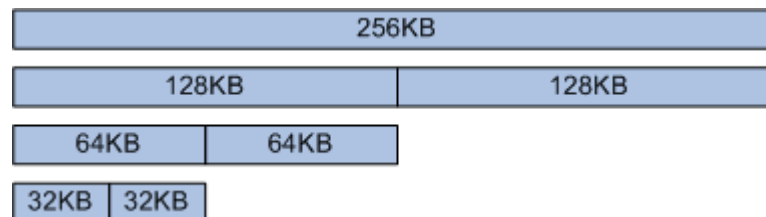
- Algoritmos para selecção do bloco livre (dimensão variável):
  - **best-fit (o melhor possível):** utiliza o melhor espaço de memória, ou seja, aquela que deixa o menor espaço alocado sem utilização. Uma grande desvantagem dessa estratégia é que, como são alocados primeiramente as partições menores, deixando pequenos blocos, existe maior fragmentação
    - Ex.: supondo os seguintes espaços de memória disponíveis para alocação: 11k, 3k, 19k, 18k, 7k, 8k, 13k, 15k. Se o algoritmo best fit for utilizado, as solicitações 5k, 12k, 6k ocupariam os espaços 7k, 13k, 8k respectivamente
  - **worst-fit (o maior/pior possível):** aloca o programa na pior partição, ou seja, aquela que deixa o maior espaço livre.
    - Esta técnica, apesar de aproveitar primeiro as partições maiores, acaba por deixar espaços livres suficientemente grandes para que outros programas utilizem a memória, o que diminui ou, pelo menos, retarda a fragmentação

# GESTÃO DE MEMÓRIA VIRTUAL: ALOCAÇÃO/RESERVA(2)

- Algoritmos para selecção do bloco livre (dimensão variável):
  - **first-fit (o primeiro possível)**: utiliza o primeiro espaço de memória que encontrar com tamanho suficiente.
    - Ex.: suponhamos os seguintes espaços livres: 11k, 3k, 19k, 18k, 7k, 8k, 13k, 15k. Se o First-Fit for utilizado, as solicitações 5k, 12k, 6k, ocupariam os espaços 11k, 19k, 18k respetivamente.
  - **next-fit (o primeiro possível a seguir ao anterior)**: Algoritmo para partição dinâmica que inicia a busca a partir da posição da última alocação até encontrar o primeiro bloco
    - Mais frequentemente são alocados blocos de tamanho grande.
    - Grandes blocos são particionados em blocos menores e existe a necessidade de compactação quando não houver mais memória disponível

# GESTÃO DE MEMÓRIA VIRTUAL: ALOCAÇÃO/RESERVA(2)

- Algoritmos para selecção do bloco livre (dimensão variável):
  - buddy:** A memória livre é dividida em blocos de dimensão  $2^n$ . Para satisfazer um pedido de dimensão  $R$ , percorre-se a lista à procura de um bloco de dimensão  $R$  tal que  $2^{(k-1)} < R < 2^k$ . Se não for encontrado procura-se um de dimensão  $2^{(k+i)}$ ,  $i > 0$ , que será dividido em duas partes iguais (buddies). Um dos buddies será subdividido quantas vezes for necessário até se obter um bloco de dimensão  $2^k$ . Na libertação, um bloco é re combinado com o seu buddy, sendo a associação entre buddies repetida até se obter um bloco com a maior dimensão possível.
  - Consegue-se um bom equilíbrio entre o tempo de procura e a fragmentação interna e externa.
  - Fácil de implementar
  - Não necessita de MMU (pode ser implementado em sistemas antigos
  - como o 80286)
  - Exemplo: pedido de 21 kB



# GESTÃO DE MEMÓRIA VIRTUAL: TRANSFERÊNCIA

- Cenários em que a transferência pode ocorrer:
  - A pedido (on request): o programa ou o SO determinam quando se deve carregar o bloco em memória principal.
  - Por necessidade (on demand): o bloco é acedido e gera-se uma falta (de segmento ou de página), sendo necessário carregá-lo para a memória principal.
  - Por antecipação: O SO recorre a heurísticas para prever quais as páginas ou segmentos que o processo vai aceder a curto prazo. Se existe espaço e a probabilidade de aceder a um bloco é elevada, transfere-se o bloco para a memória principal. Antecipa-se a ocorrência de uma falta.

# GESTÃO DE MEMÓRIA VIRTUAL: TRANSFERÊNCIA

- Transferência de segmentos:
  - Normalmente um processo precisa de ter pelo menos um segmento de código, de dados e de stack em memória.
  - A transferência de segmentos é normalmente feita a pedido.
  - Se não existir espaço disponível em memória, os segmentos de outros processos que não estejam em execução (bloqueados ou suspensos) são colocados em memória secundária (swapping). São, geralmente, usados três critérios para decidir qual o processo a transferir para disco:
    - estado e prioridade do processo: processos bloqueados e pouco prioritários são candidatos preferenciais;
    - tempo de permanência na memória principal: um processo tem que permanecer um determinado tempo a executar-se antes de ser novamente enviado para disco;
    - dimensão do processo.

# GESTÃO DE MEMÓRIA VIRTUAL: TRANSFERÊNCIA

- Transferência de páginas:
  - A transferência de páginas é normalmente feita por necessidade. As páginas de um programa que não sejam acedidas durante a execução de um processo nunca chegam a ser carregadas em memória principal.
  - Na transferência por antecipação, procura-se diminuir o número de faltas de página e otimizar os acessos a disco.
  - Quando não existe espaço em memória principal, as páginas retiradas de memória principal são guardadas numa zona separada do disco. As páginas modificadas são transferidas em grupo para a memória secundária de modo a otimizar os acessos a disco.

# GESTÃO DE MEMÓRIA VIRTUAL: SWAP AREA

- É uma zona do disco para onde é transferido o conteúdo das páginas modificadas expulsas da memória:
  - páginas de ficheiros não alteráveis, por ex. código, não precisam ser transferidas para a swap area.
- Pode ser implementada:
  - directamente sobre o disco – mais eficiente;
  - sobre ficheiros – fácil de ajustar o seu tamanho.
- Em qualquer caso, o SO tem que gerir o espaço disponível na swap area.
- A alocação de espaço na área de swap pode ser:
  - Otimista: só é alocada quando necessário – possibilidade de deadlock?
  - Pessimista: é reservada – pode não vir a ser usada e, consequentemente, reduzir a utilização dos recursos.

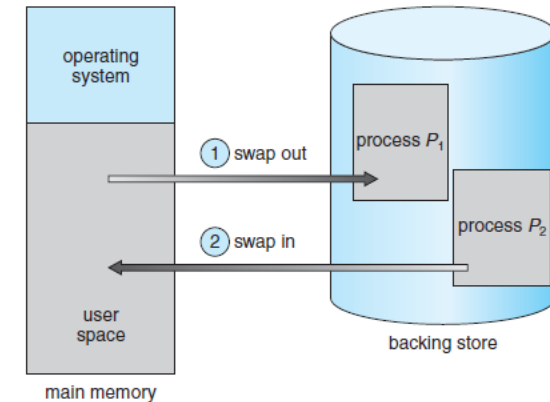


Figure 8.5 Swapping of two processes using a disk as a backing store.

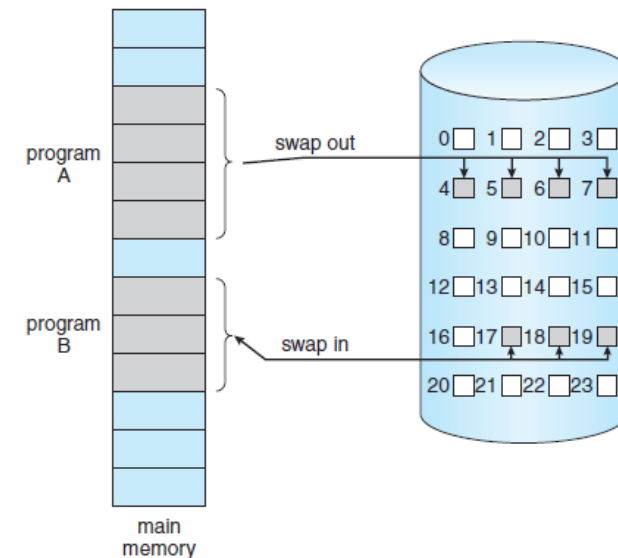


Figure 9.4 Transfer of a paged memory to contiguous disk space.

# SWAPPING

- **Processo retirado temporariamente da memória (para disco, tipicamente)**

- ocorre quando a carga do sistema é elevada e começa a faltar memória

- **Dependendo de como os endereços foram atribuídos, quando o processo volta a ser colocado em memória pode ir para a mesma zona ou para outra**

- se atribuição de endereços é feita em tempo de compilação ou de carregamento tem de ser para a **mesma** zona de memória

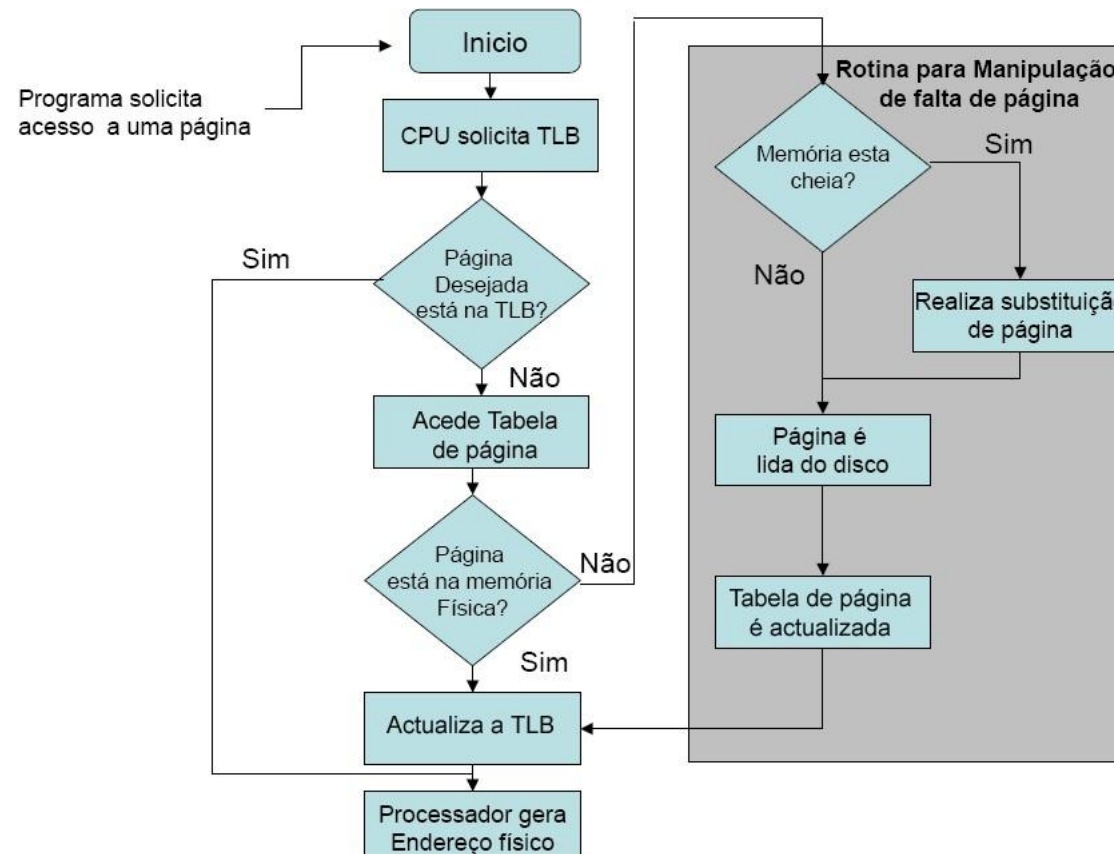
- se é feita em tempo de execução pode ser para local **diferente**

- **Em sistemas móveis normalmente não se usa swapping por limitações de espaço em disco (e porque é limitado o número de escritas em memória flash)**

- quando começa a faltar memória, sistemas como o iOS ou o Android pedem às aplicações para libertarem memória

- se não libertarem a memória suficiente, correm o risco de serem **terminadas** pelo SO

# GESTÃO DE MEMÓRIA VIRTUAL: FLUXOGRAMA DA SOLICITAÇÃO DE ACESSO ÀS PÁGINAS



# GESTÃO DE MEMÓRIA VIRTUAL: SUBSTITUIÇÃO DE PÁGINAS (1)

- Ocorre quando não existem mais páginas disponíveis em memória principal e é necessário escolher qual a página que deve ser substituída.
- Um processo paginador tem acesso às tabelas de páginas de processos, mantendo duas listas de páginas: páginas livres e páginas livres modificadas. Quando o número de páginas livres é inferior a um determinado valor, o processo paginador irá transferir as páginas livres para disco. Se as páginas livres forem insuficientes será necessário transferir páginas livres modificadas. As páginas livres modificadas são periodicamente escritas em disco.
- Os algoritmos de substituição de páginas seguem o princípio da localidade de referência (espacial e temporal). Os processos tendem a favorecer certos subconjuntos de páginas e estas são adjacentes no espaço de endereçamento.

# GESTÃO DE MEMÓRIA VIRTUAL: SUBSTITUIÇÃO DE PÁGINAS (2)

- Algoritmo óptimo (não implementável, mas útil como benchmark):
  - O algoritmo retira da memória a página com menos hipótese de ser referenciada
  - Implementação:
    - Como prever o que vai acontecer de futuro?
    - Praticamente impossível de saber
    - Impraticável
- Usado em simulações para comparação com outros algoritmos.

# ALGORITMO - FIFO – FIRST-IN-FIRST-OUT

O algoritmo seleciona a página que está há mais tempo no sistema

## Implementação:

- Utilização de uma lista duplamente ligada. A última página em falta é colocada na primeira posição da fila. A página a ser substituída está no fim da fila (a mais antiga).
- Não atende ao princípio de localidade temporal (páginas mais recentemente usadas podem ser substituídas). A página que foi carregada há mais tempo pode estar a ser utilizada.
- Exemplo:

- Anomalia de Belady (a 1.<sup>a</sup> linha indica page fault)

Sequência de referência:

1,	2,	3,	4,	1,	2,	5,	1,	2,	3,	4,	5
1	2	3	4	1	2	5	5	5	3	4	4
	1	2	3	4	1	2	2	2	5	3	3
		1	2	3	4	1	1	1	2	5	5

Nº de faltas de página: 9

Sequência de referência:

1,	2,	3,	4,	1,	2,	5,	1,	2,	3,	4,	5
1	2	3	4	4	4	5	1	2	3	4	5
	1	2	3	3	3	4	5	1	2	3	4
		1	2	2	2	3	4	5	1	2	3
			1	1	1	2	3	4	5	1	2

Nº de faltas de página: 10

- Vantagem: **simples** de implementar
- Desvantagem: performance é muitas vezes **má**, com muitos page faults

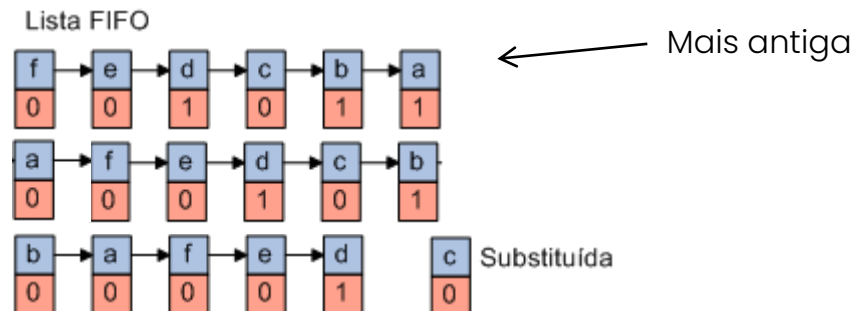
# ALGORITMO - SECOND CHANCE

Algoritmo baseado no FIFO, mas utiliza o bit de referência (R) colocado a 1 se a página foi referenciada

▪ Antes de uma página ser substituída, se o bit R está a 1, é colocado a 0, mas a página não é substituída. A página a substituir será a primeira que for encontrada de entre as mais antigas com R=0.

## ▪ Exemplo:

- Se R=0, a página é retirada da memória
- Caso contrário R é colocado a 0 e dá-se uma nova hipótese à página colocando-a no final da lista



# ALGORITMO LEAST-RECENTLY-USED (LRU)

## O algoritmo seleciona a página menos usada recentemente

- Princípio de localidade temporal.

- **Implementação:**

- Utilização de uma marca temporal do último acesso na tabela de páginas. Aquando da necessidade de substituição, deve ser feita a pesquisa da página com o menor valor.
- Manter uma pilha (lista duplamente ligada das páginas referenciadas), colocando no topo a última página referenciada. Não envolve pesquisa aquando da substituição, mas aumenta o esforço computacional associado a cada acesso.

- **Exemplo:**

Sequência de referência:

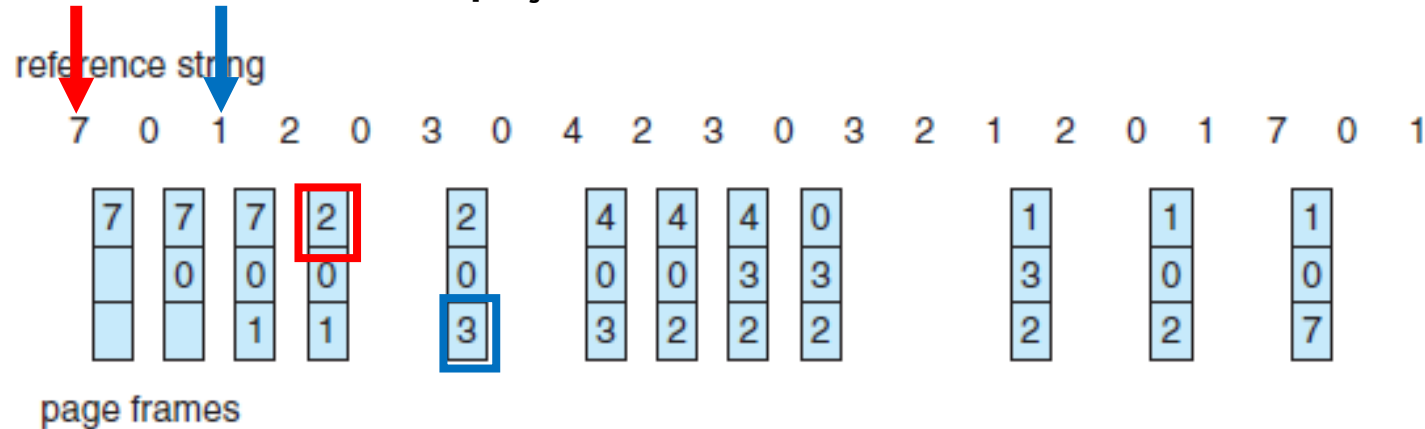
1,	2,	3,	4,	1,	2,	5,	1,	2,	3,	4,	5
1	1	1	1	1	1	1	1	1	1	1	5
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	5	5	5	5	4	4
			4	4	4	4	4	4	3	3	3

Nº de faltas de página: 8

# ALGORITMO LEAST-RECENTLY-USED (LRU)

Substituir a página que não é usada **há mais tempo**

o **passado** é usado como antecipação do **futuro**...



Vantagem: normalmente a **performance** é boa

Desvantagem: eficiência requer **bastante suporte hardware**

- pode usar-se um **contador** a servir de relógio, incrementando-o sempre que haja acessos à memória e copiando o seu valor para a página acedida
- a página que tiver um valor mais pequeno é removida (é a mais "antiga")
- mas isso atrasa bastante os acessos à memória

# ALGORITMO LEAST-FREQUENTLY-USED (LFU)

## O algoritmo seleciona a página menos frequentemente referenciada

- Baseado na heurística de que uma página não frequentemente referenciada não será provavelmente referenciada no futuro.
- Aproximação do algoritmo LRU, mas com menos esforço computacional.

### ▪ Implementação:

- É associado um contador a cada página carregada em memória, inicializado a zero quando a página é carregada. Sempre que ocorre uma interrupção do relógio, e para cada página, soma-se o valor do bit R (página referenciada) ao contador. É substituída a página de menor valor do contador.
- Pode, no entanto, uma página muito referenciada no passado, nunca mais ser referenciada no futuro e manter-se-á na memória principal por ter um valor elevado no contador.

# ALGORITMO LEAST-FREQUENTLY-USED (LFU)

- O algoritmo seleciona a página não usada recentemente, é uma aproximação do algoritmo LRU, mas com menos esforço computacional.

- **Implementação:**

- Para cada página são utilizados dois bits: bit de página referenciada (R) e bit de página modificada (M) que permitem classificar as páginas em quatro grupos:

- 0: (R = 0, M = 0) não referenciada, não modificada (melhor escolha)

- 1: (R = 0, M = 1) não referenciada, modificada

- 2: (R = 1, M = 0) referenciada, não modificada

- 3: (R = 1, M = 1) referenciada, modificada (pior escolha)

- São substituídas as páginas dos grupos com o número mais baixo.

- Periodicamente, um processo coloca a zero o bit R, bem como, grava em disco as páginas modificadas, colocando o bit M a 0.

# ALGORITMO NOT-USED-RECENTLY (NUR)

**O algoritmo seleciona a página não usada recentemente.**

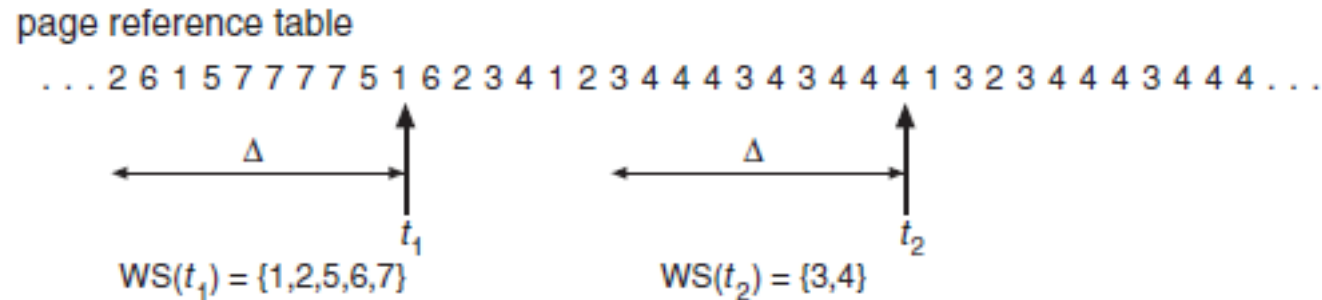
- Aproximação do algoritmo LRU, mas com menos esforço computacional .
- **Implementação:**
  - Para cada página são utilizados dois bits: bit de página referenciada (R) e bit de página modificada (M) que permitem classificar as páginas em quatro grupos:
    - 0: (R = 0, M = 0) não referenciada, não modificada (melhor escolha)
    - 1: (R = 0, M = 1) não referenciada, modificada 2: (R = 1, M = 0) referenciada, não modificada
    - 3: (R = 1, M = 1) referenciada, modificada (pior escolha)
  - São substituídas as páginas dos grupos com o número mais baixo.
  - Periodicamente, um processo coloca a zero o bit R, bem como, grava em disco as páginas modificadas, colocando o bit M a 0.

# GESTÃO DE MEMÓRIA VIRTUAL: WORKING SETS

- A substituição de páginas deve ser feita globalmente, para todos os processos, ou apenas sobre as páginas do processo que gerou a falta de página?
- Com paginação pura, as páginas dum processo só são trazidas para a memória quando o processo tem uma page-fault.
- Quando um processo inicia, tem uma taxa de page-faults elevada, até que eventualmente essa taxa diminui:
  - este comportamento deve-se à localidade de referência.
- O conjunto de páginas acedidas por um processo num determinado intervalo designa-se por working set.
- Se o working set dum processo estiver em memória, um processo tem uma taxa de page-faults muito baixa, reciprocamente . . .
- Tipicamente, o working set dum processo varia no tempo.

# MODELO WORKING-SET

- Uma solução para a questão "**quantas** frames são necessárias?"
- Este modelo baseia-se nas localidades
  - o parâmetro  $\Delta$  (**janela do working-set**) mantém um número fixo de referências a páginas, sendo uma **aproximação** da localidade
  - o working set de um processo é o conjunto de páginas que foram referenciadas na **janela mais recente**



- Se o **somatório** de todos os working-sets for superior ao tamanho da memória disponível → thrashing
  - Que fazer? **Suspender** um dos processos
- Problema 1: qual o **tamanho da janela**?
  - se  $\Delta$  pequeno não abrange toda a localidade
  - se  $\Delta$  grande pode abranger mais do que uma localidade
- Problema 2: como **manter o working-set**?
  - cada referência à memória altera o working-set...

# QUANTAS FRAMES A ATRIBUIR A CADA PROCESSO?

- No esquema de demand paging “puro” as páginas são colocadas em memória só quando são necessárias, **uma a uma**

- quando um processo se inicia será despoletada uma série de page faults, reduzindo o **desempenho** do sistema

- Na prática, um processo pode ter um certo número de páginas

- o número **máximo de páginas** está limitado pela memória física

- mas deveria de ter um número **mínimo** de páginas

- por razões de **desempenho** e porque certas instruções podem estar em **várias** páginas

- Esquemas de atribuição de frames

- atribuição **fixa**

- Uniforme - número de frames **igual** para todos os processos, independentemente do seu tamanho

- Proporcional - número de frames **proporcional** à dimensão do processo

- atribuição **por prioridade do processo**

- Maior prioridade = mais frames

# SUBSTITUIÇÃO LOCAL VS GLOBAL

## ▪Substituição global

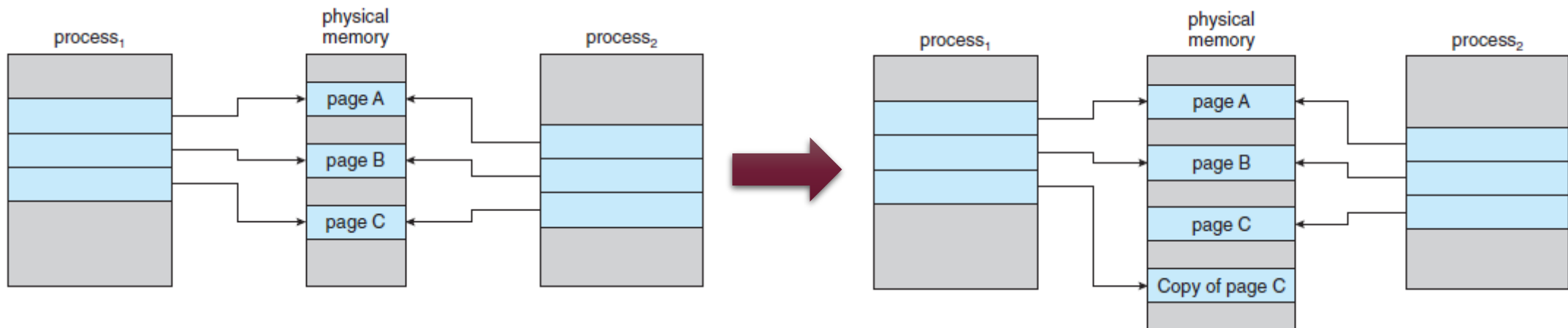
- A frame de substituição é escolhida do conjunto de **todas** as frames, podendo por isso pertencer a **outro** processo
- Melhor desempenho (normalmente), mas processo deixa de ter controlo sobre a sua taxa de page faults

## ▪Substituição local

- A frame de substituição é escolhida do conjunto das frames do **próprio** processo
- Assim, o número de frames atribuídas a cada processo mantém-se **inalterado**

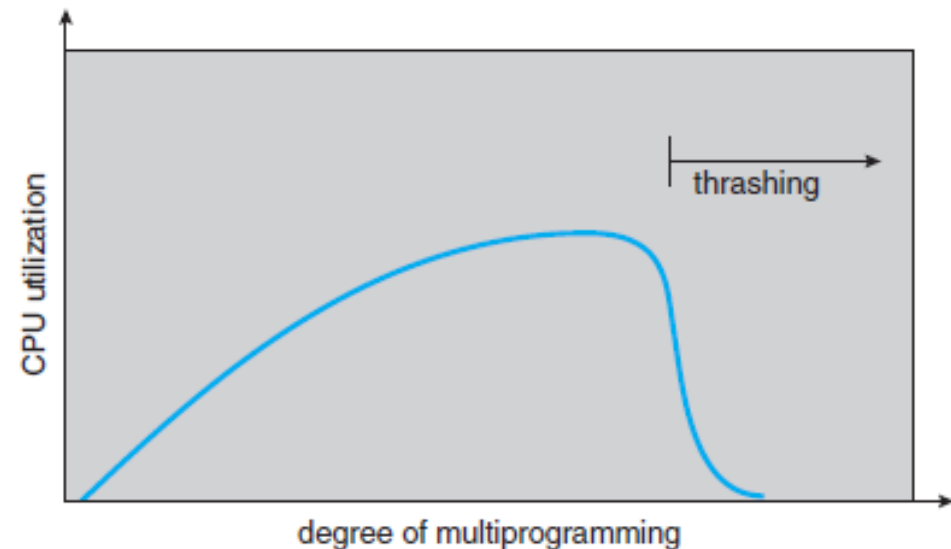
# COPY-ON-WRITE (COW)

- A chamada `fork()` cria um processo filho que é uma **cópia** do pai
  - **causa a duplicação** das páginas usadas pelo pai
  - seria de evitar essa duplicação, até porque muitas vezes os filhos chamam logo a seguir um `exec()` para correr um novo programa
- O esquema COW permite que o processo pai e o processo filho **partilhem** as mesmas páginas de memória inicialmente
  - só quando um dos processos **altera** uma página é que a página é copiada
- Criação de processos mais **rápida** e mais **eficiente**



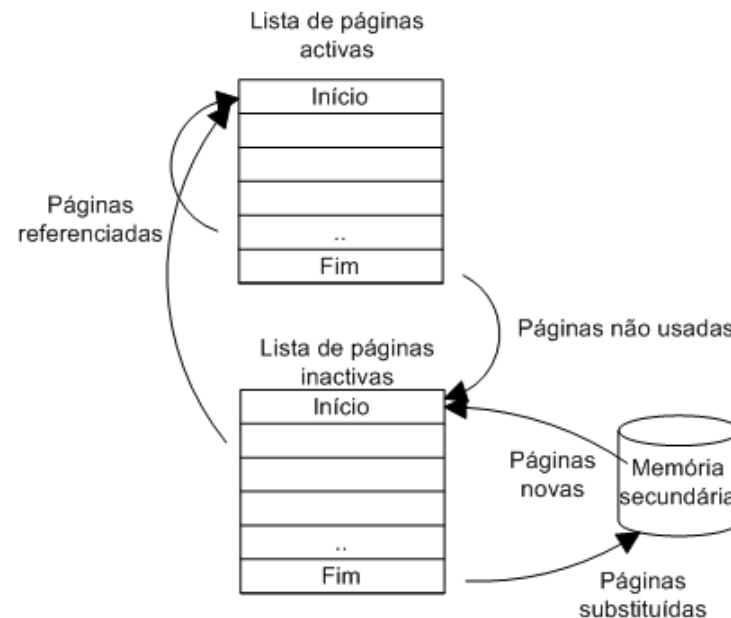
# THRASHING

- Se um processo não tem as páginas **suficientes** para executar, o número de page-faults torna-se muito elevado
  - com a consequência de haver uma **baixa utilização** do CPU
  - que por sua vez leva o SO a pensar que necessita de **aumentar** o nível de multiprogramação, **adicionando** mais um processo ao sistema → efeito "bola de neve"
- **Thrashing**
  - Na maior parte do tempo de execução, o sistema encontra-se a resolver as faltas de páginas, não produzindo trabalho útil. Esta situação deve ser evitada a todo o custo.
- **Como resolver o problema?**
  - um algoritmo de **substituição local limita** o problema
  - melhor: fornecer aos processos as frames de que ele **necessita**!
  - Questão a resolver: **quantas** frames são necessárias?



# GESTÃO DE MEMÓRIA VIRTUAL: LINUX

- O Linux usa uma aproximação do algoritmo LRU.
- O gestor de memória usa duas listas ligadas:
  - Lista de páginas ativas: as mais recentemente usadas são colocadas no topo da lista.
  - Lista de páginas inativas: as páginas menos usadas recentemente são colocadas no fim da lista. Apenas as páginas desta lista são substituídas.



# GESTÃO DE MEMÓRIA VIRTUAL: TRANSFERÊNCIA

## Cenários em que a transferência pode ocorrer:

**on request:** o programa ou o SO determinam quando se deve carregar o bloco em memória principal.

**on demand:** o bloco é acedido e gera-se uma falta (de segmento ou de página), sendo necessário carregá-lo para a memória principal.

**prefetching:** O SO recorre a heurísticas para prever quais as páginas ou segmentos que o processo vai aceder a curto prazo. Se existe espaço e a probabilidade de aceder a um bloco é elevada, transfere-se o bloco para a memória principal. Antecipa-se a ocorrência de uma falta.

# PAGINAÇÃO A PEDIDO (DEMAND PAGING)

A página só é colocada em memória quando  
for **necessária**

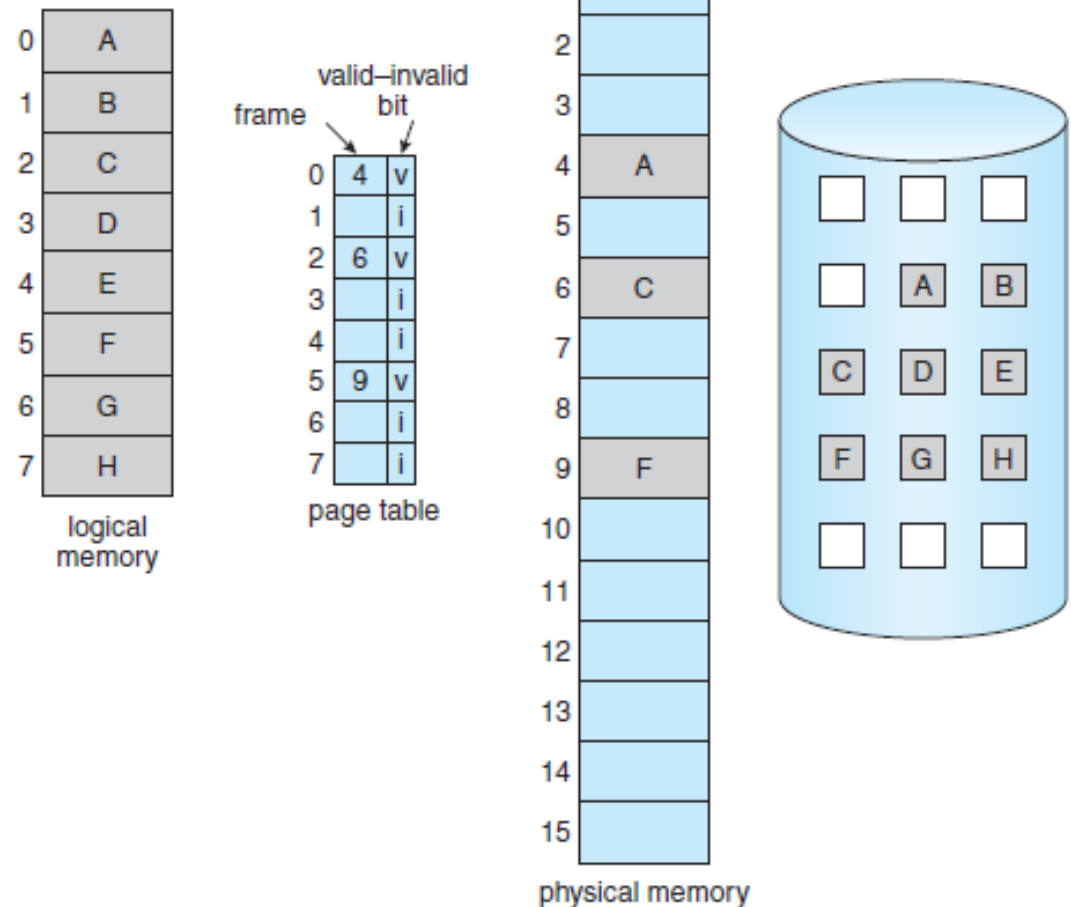
se **nunca** for necessária, **nunca** é colocada em  
memória

Como saber então se uma página está em  
memória ou no disco?

possível solução (hardware): usar o esquema do  
bit **válido-inválido**

se válido, página é válida e está em **memória**

se inválido, ou não é válida, ou está no **disco**



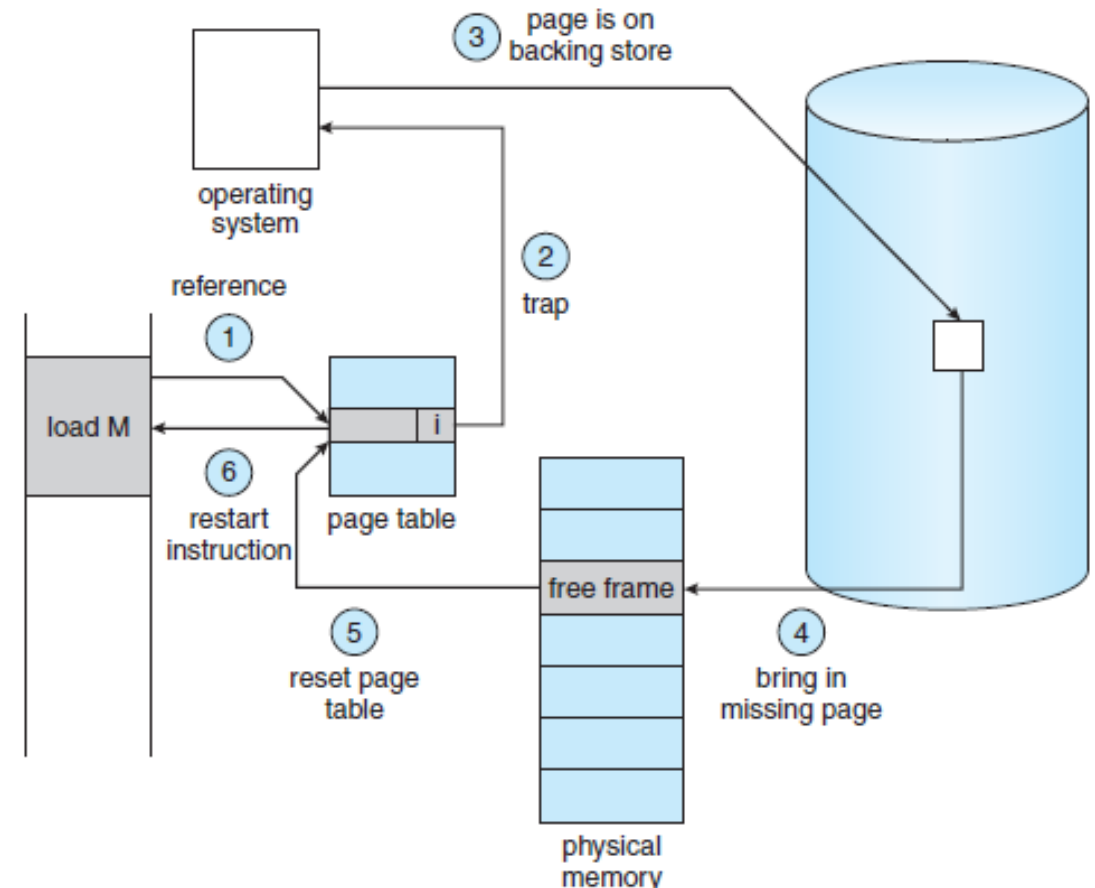
# FALTA DE PÁGINA(PAGE FAULT)

Quando se tenta aceder a uma **página marcada como inválida** é causado um **page fault**

O que fazer?

- (1) SO examina **outra tabela interna** (tipicamente no PCB) e:
- (2) Se referência inválida, **termina** programa
- (3) Caso contrário, está em disco, **trazer** para memória
- (4) Obter frame vazia e colocar a página nessa **frame**
- (5) **Atualizar** as tabelas e o bit de validação
- (6) **Repetir** instrução que causou o page fault

Problema: Desempenho



# PERFORMANCE OF DEMAND PAGING

## Stages in Demand Paging (worse case)

1. Trap to the operating system
2. Save the user registers and process state
3. Determine that the interrupt was a page fault
4. Check that the page reference was legal and determine the location of the page on the disk
5. Issue a read from the disk to a free frame:
  1. Wait in a queue for this device until the read request is serviced
  2. Wait for the device seek and/or latency time
  3. Begin the transfer of the page to a free frame
6. While waiting, allocate the CPU to some other user
7. Receive an interrupt from the disk I/O subsystem (I/O completed)
8. Save the registers and process state for the other user
9. Determine that the interrupt was from the disk
10. Correct the page table and other tables to show page is now in memory
11. Wait for the CPU to be allocated to this process again
12. Restore the user registers, process state, and new page table, and then resume the interrupted instruction

# PERFORMANCE OF DEMAND PAGING (CONT.)

## Three major activities

Service the interrupt - careful coding means just several hundred instructions needed

Read the page - lots of time

Restart the process – again just a small amount of time

## Page Fault Rate $0 \leq p \leq 1$

if  $p = 0$  no page faults

if  $p = 1$ , every reference is a fault

## Effective Access Time (EAT)

$$\text{EAT} = (1 - p) \times \text{memory access} + p (\text{page fault overhead} + \text{swap page out} + \text{swap page in})$$

# DEMAND PAGING EXAMPLE

**Memory access time = 200 nanoseconds**

**Average page-fault service time = 8 milliseconds**

$$\text{EAT} = (1 - p) \times 200 + p (8 \text{ milliseconds})$$

$$= (1 - p) \times 200 + p \times 8,000,000$$

$$= 200 + p \times 7,999,800$$

**If one access out of 1,000 causes a page fault, then**

$$\text{EAT} = 8.2 \text{ microseconds.}$$

This is a slowdown by a factor of 40!!

**If want performance degradation < 10 percent**

$$220 > 200 + 7,999,800 \times p$$

$$20 > 7,999,800 \times p$$

$$p < .0000025$$

- < one page fault in every 400,000 memory accesses

# E SE NÃO HOUVER FRAMES LIVRES?

Escolher uma que esteja em memória sem ser usada e **substituir**

1. Método básico:
2. escrever o conteúdo da frame "**vítima**" para disco
3. atualizar bit validação
4. ler a nova página para memória atualizar tabela

Um método mais eficiente implica o uso de um **bit de modificação**

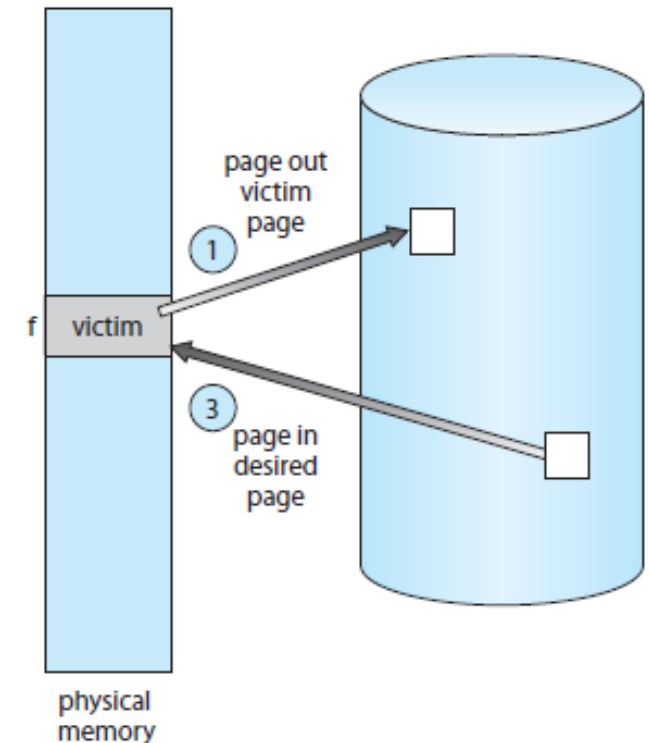
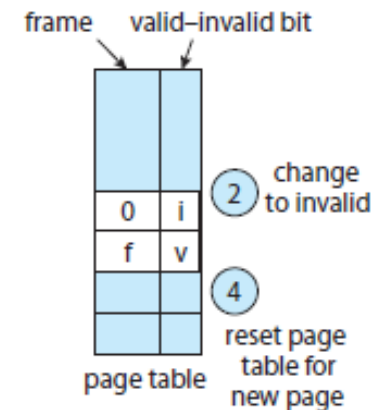
se a vítima não tiver sido modificada não é necessário escrever para disco (passo 1)

Mas que página remover?

vários **algoritmos** de substituição: FIFO, Ótimo, LRU e variantes, etc.

Qual o melhor?

o que **reduza** o número de page faults



# FICHEIROS MAPEADOS EM MEMÓRIA

- Ao mapear um ficheiro do disco para memória o acesso a esse ficheiro pode ser tratado como um **normal** acesso à memória
- O ficheiro é inicialmente lido usando demand paging (resultando num page-fault)
  - parte do ficheiro (normalmente do tamanho de uma página) é depois lida do sistema de ficheiros **para memória**
  - as leituras/escritas subsequentes são tratadas como acesso normal à memória
- Vantagens
  - **simplifica** acesso ao ficheiro (é mais **fácil** e mais **rápido** aceder através da memória do que com as chamadas ao sistema `read()` e `write()`)
  - permite que vários processos mapeiem o mesmo ficheiro e assim partilhem um pedaço de memória (comunicação através de **memória partilhada**)

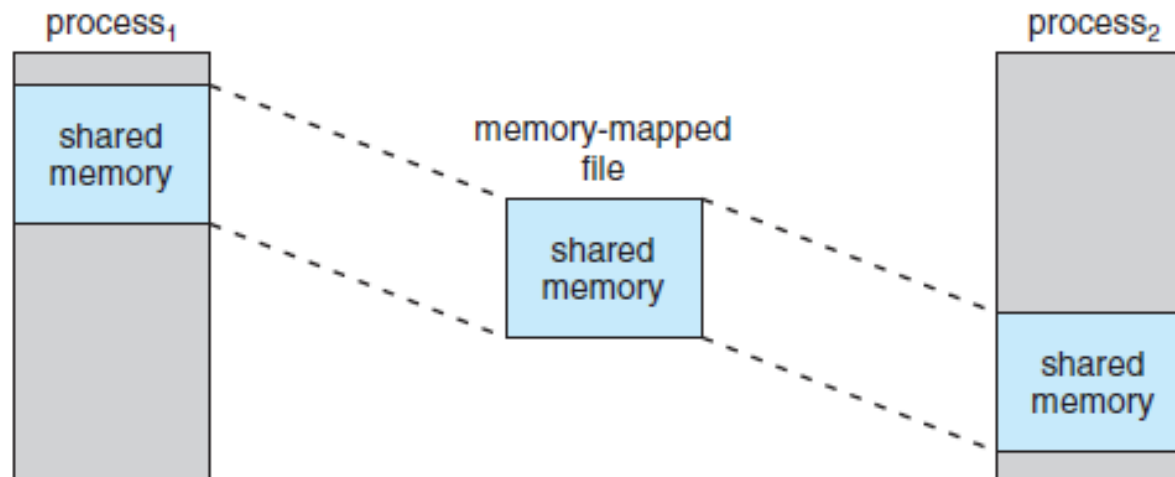


Figure 9.23 Shared memory using memory-mapped I/O.