Gestão de Memória
e
Gestão de Memória Virtual

- Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, ***Operating System Concepts, 6ª Ed.***, John Wiley & Sons Inc., 2002 **[cap. 9 e 10]**
- Andrew S. Tanenbaum, ***Modern Operating Systems, 2nd. Ed.***, Prentice Hall, 2001 **[cap. 4]**

Gestão de Memória

Gestão Memória: Introdução

- A partilha do CPU por um conjunto de processos permite melhorar a utilização do CPU e o tempo de resposta aos utilizadores – *CPU scheduling*
- Este aumento de desempenho implica a existência de vários processos em memória
- Estes processos são instancias de programas residentes em disco na forma ficheiros binários (executáveis) que tem de ser carregados em memória
- Necessidade da existência de mecanismos de protecção

- O **gestor de memória** é a componente do sistema operativo responsável pela gestão da memória do sistema:
  - Qual a memória que está a ser utilizada e a que está livre
  - Atribuir memória a um processo e libertá-la quando deixar de ser necessária
  - Gerir a permuta entre a memória e o disco (*swapping*) quando a memória principal não for suficiente para suportar todos os processos

Gestão Memória: Atribuição de endereços de memória

- A atribuição de endereços de memória pode realizar-se em:

  - **Tempo compilação (*Compile time*)**
    - Localização da memória conhecida à priori
    - Geração de código absoluto
    - Necessidade de voltar a compilar se o endereço de inicio se alterar

  - **Tempo de carregamento (*Load time*)**
    - Localização da memória desconhecida durante a compilação
    - Geração de código que possa ser realojado (*relocatable code*) no carregamento

  - **Tempo de execução (*Execution time*)**
    - Determinação da localização da memória na altura da execução
    - A localização do processo em memória pode ser alterada durante a sua execução
    - Necessita de suporte por parte do hardware para determinar o endereço físico (e.g. existência de registos *base* e *limit*)
    - Utilizado pela maior parte dos SO de utilização genérica

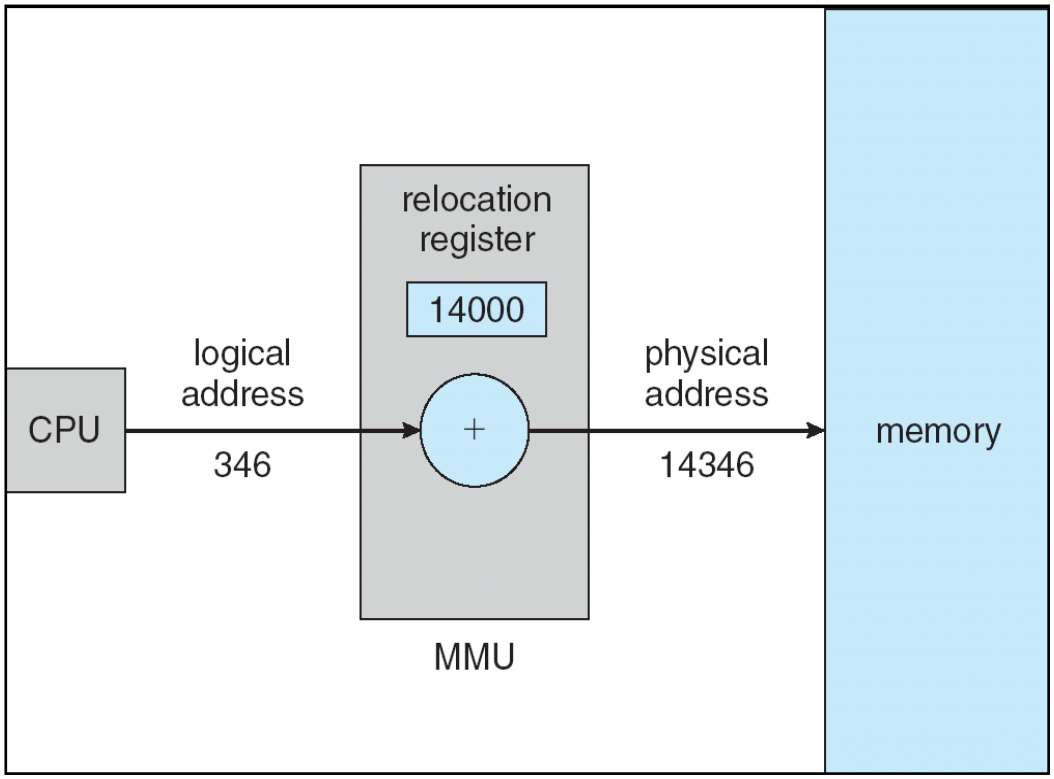
Gestão Memória: Endereços lógicos e físicos

- Um ponto central na gestão de memória reside na existência de um **espaço de endereçamento lógico** distinto do **espaço de endereçamento físico**
  - **Endereço lógico** ou **endereço virtual -** gerado pelo CPU
  - **Endereço físico** – o endereço utilizado no acesso à memória

- Os espaços de endereçamento físico e lógico
  - **são os mesmos** nos esquemas de atribuição de endereços em tempo de compilação e carregamento (*compile-time* e *load-time*)
  - **diferem** no esquema de atribuição de endereços em tempo de execução (*execution-time*)

Gestão Memória: Memory-Management Unit (MMU)

- Dispositivo *hardware* para associar um endereço virtual a um endereço físico
- O valor do registo de realojamento é adicionado a todos os endereços gerados por um processo antes de submeter à memória
- O programa apenas lida com endereços lógicos nunca com os endereços físicos



Realojamento dinâmico através de um registo base

Gestão Memória: *Dynamic Loading*

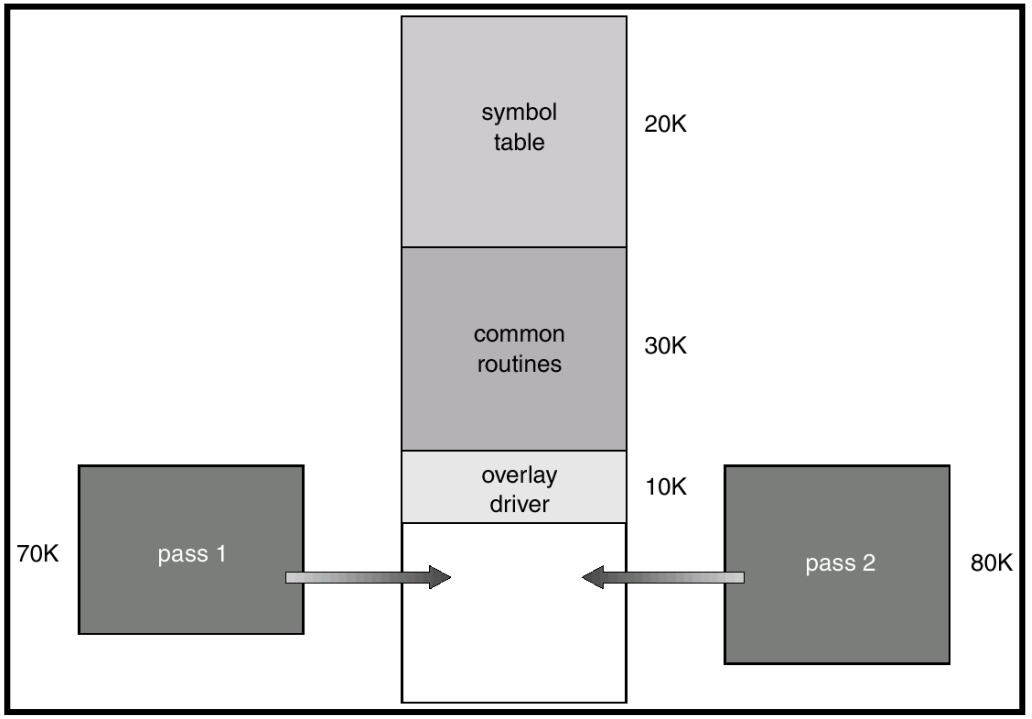
- As rotinas são carregadas quando são necessárias
- Melhor utilização de memória – uma rotina que não seja utilizada nunca necessita de ser carregada
- Útil quando existe uma grande quantidade de código necessário para situações pouco frequentes
- Não requer suporte especial do SO. É da responsabilidade do programador tirar partido deste método. O SO pode auxiliar o programador disponibilizando uma biblioteca de suporte ao *dynamic loading*

Gestão Memória: *Dynamic Linking Libraries*

- Ligação (*linking*) adiada até à altura da execução
- Possui mecanismos para localizar uma rotina de uma biblioteca, residente em memória, ou como carregar uma biblioteca no caso desta ainda não se encontrar em memória
- O SO tem de dar suporte a este mecanismo, verificando se uma biblioteca já se encontra carregada, **permitir o acesso simultâneo à biblioteca por vários processos**
- Útil na definição de bibliotecas
- Simplifica o processo de actualização das bibliotecas
- Também conhecido como *shared libraries*

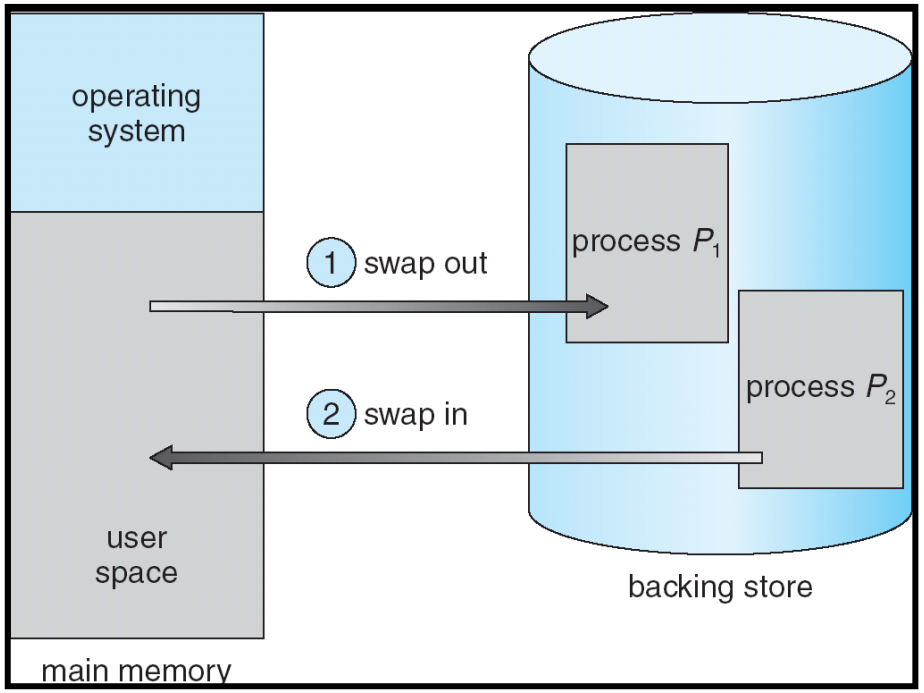
Gestão Memória: *Overlays*

- Mantém em memória o código e dados necessários num dado instante
- Pode ser utilizado quando a **memória total do processo excede a memória física** disponível
- Realizado pelo programador não exigindo suporte especial por parte do SO
- Estruturalmente complexo



Gestão Memória: *Swapping*

- Um processo pode ser temporariamente retirado de memória para uma zona de suporte (*backing store*) e mais tarde voltar à memória para continuar a sua execução
- *backing store* zona de armazenamento não volátil e rápido: um disco, ou um ficheiro, destinado a guardar cópia de toda a memória dos processos.
- *Roll out*, *roll in* uma variante desta técnica utilizada pelos algoritmos de escalonamento baseados em prioridades em que um processo de baixa prioridade é transferido para a zona de *backing store* para que um processo de mais alta prioridade pode ser carregado e executado
- Algumas variantes do *Swapping* são encontrados em sistemas, como por exemplo, UNIX, Linux e Windows (V3.1)
- O SO tem de manter uma fila de processos à espera para se executarem (*ready queue*) cujas as imagens não se encontram carregadas em memória

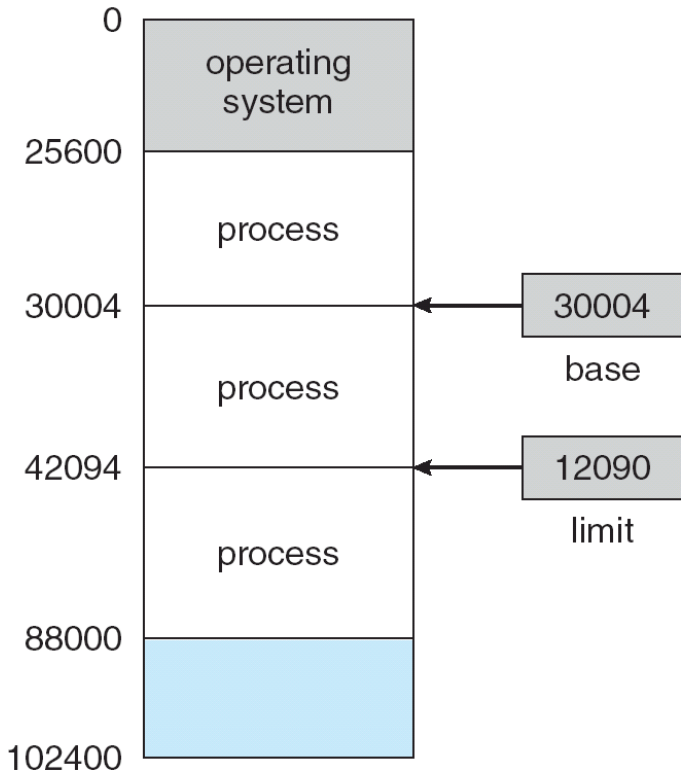


Gestão Memória: Atribuição de memória contígua

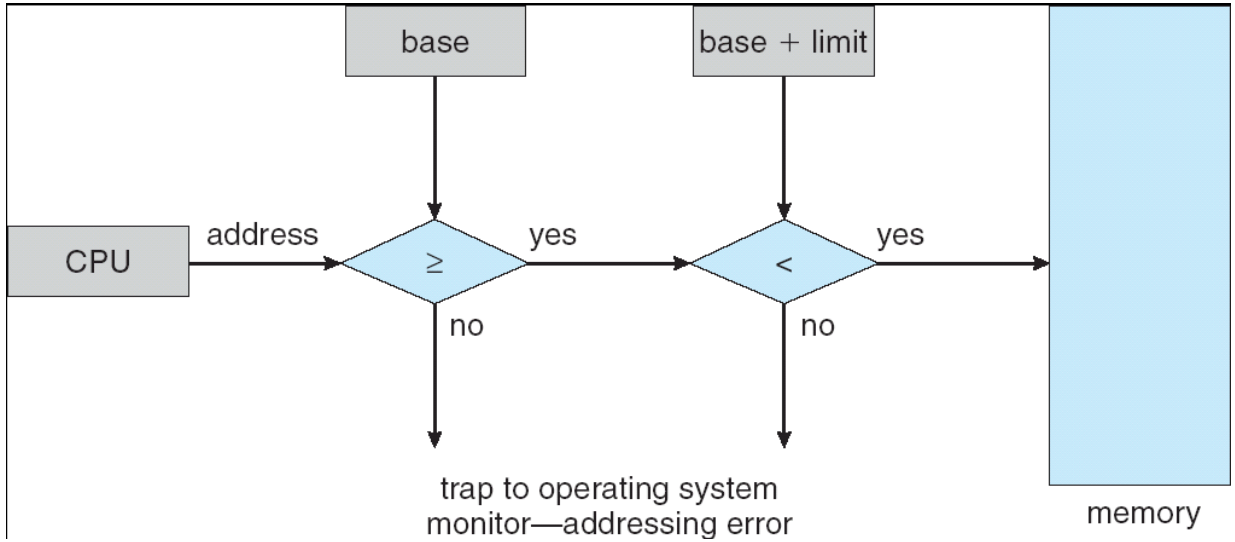
- A memória principal dividida em duas partições
  - Parte residente do Sistema Operativo tipicamente residente na parte baixa da memória
    - O factor preponderante na decisão da localização do SO é a localização da tabela de *interrupts*
  - Os processos localizados na parte alta

- Registos de realojamento utilizados para protecção entre os processos e o Sistema Operativo
  - Registo base contendo o valor do endereço físico mais baixo
  - Registo limite com a dimensão máxima permitida a partir do registo base
  - MMU que associa dinamicamente um endereço lógico a um endereço físico

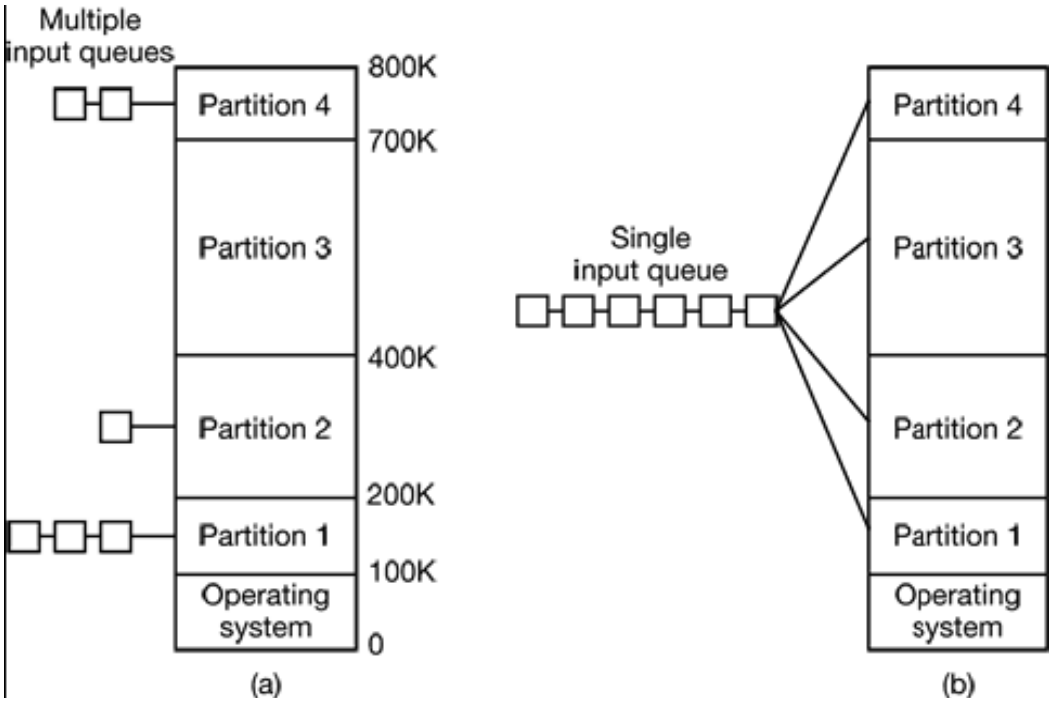
Gestão Memória: Atribuição de memória contígua - Protecção de memória



Suporte hardware para a definição de um espaço de endereçamento virtual, com realojamento dinâmico e protecção de memória

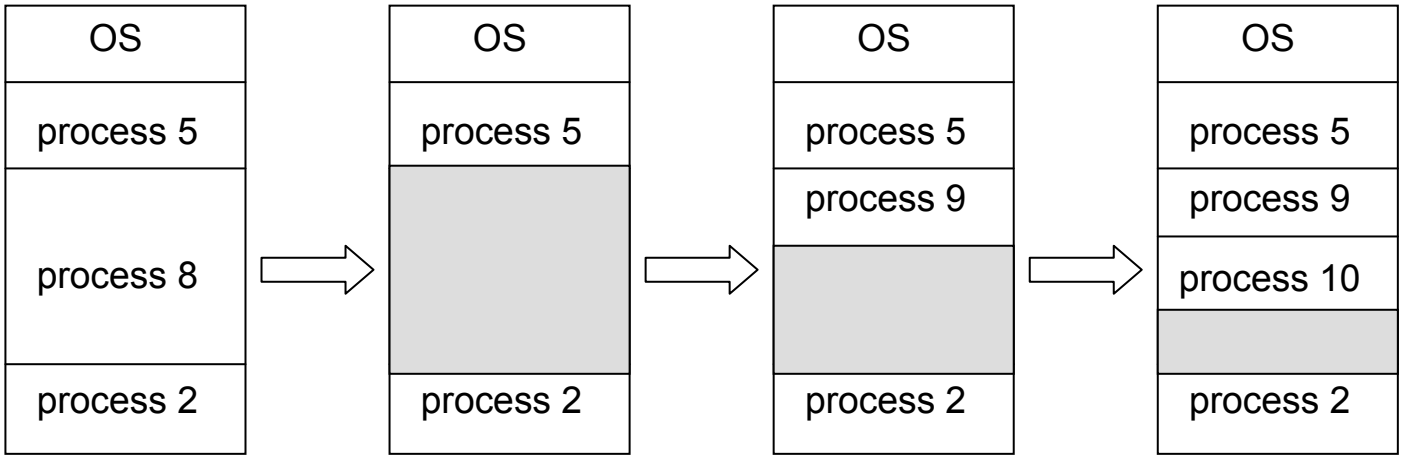


Gestão Memória: Atribuição de memória contígua – partições fixas



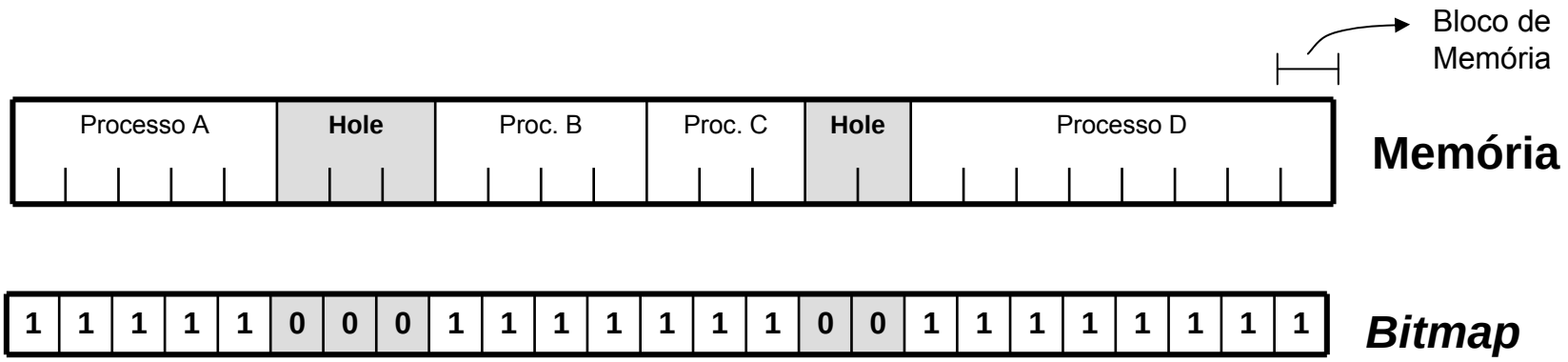
Método utilizado nos sistemas mainframes da IBM OS/360 e designado de *Multiprogramming with a Fixed number of Tasks* (MFT)

- Atribuição de memória dinâmica
  - Espaço livre constituído por vários blocos de dimensões diferentes espalhados pela memória
  - Quando aparece um novo processo atribui-se memória a partir de um bloco livre que seja suficiente para satisfazer as necessidades desse processo
  - O sistema operativo mantém informação sobre:
    - a) espaço ocupado
    - b) espaço livre



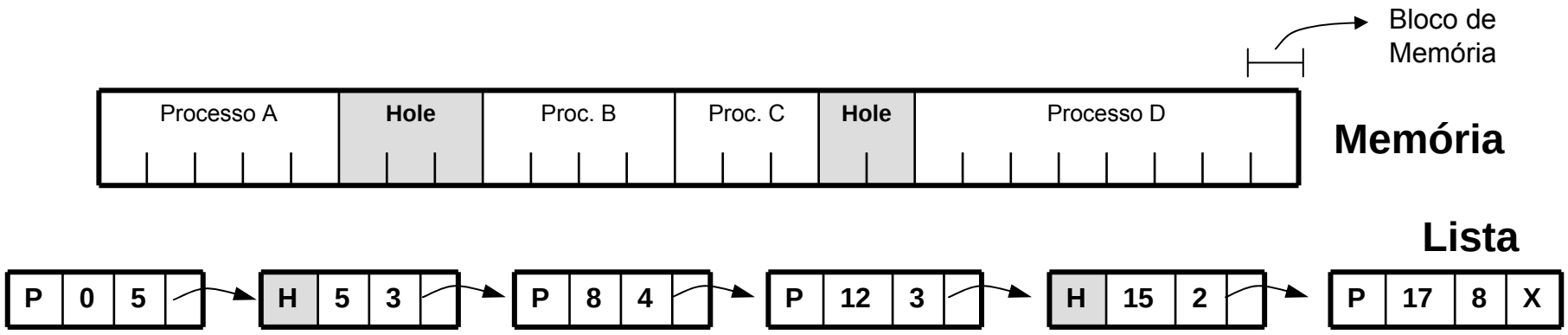
Como atribuir um bloco de memória de dimensão *n* ?

- **Gestão por Bitmaps**
  - *Array* de *bits* para guardar os blocos que estão ocupadas e os que estão vagos
  - Todos os blocos são iguais, o que facilita a compactação
  - O tamanho do *Bitmap* é proporcional ao tamanho da memória (cada bloco de memória apenas necessita de 1 *bit* para gestão)
  - Lento na pesquisa de um bloco livre para uma dada dimensão, especialmente se os *bits* não pertencerem todos à mesma *word*
  - Utiliza-se quando os blocos de memória são poucos e de dimensão fixa



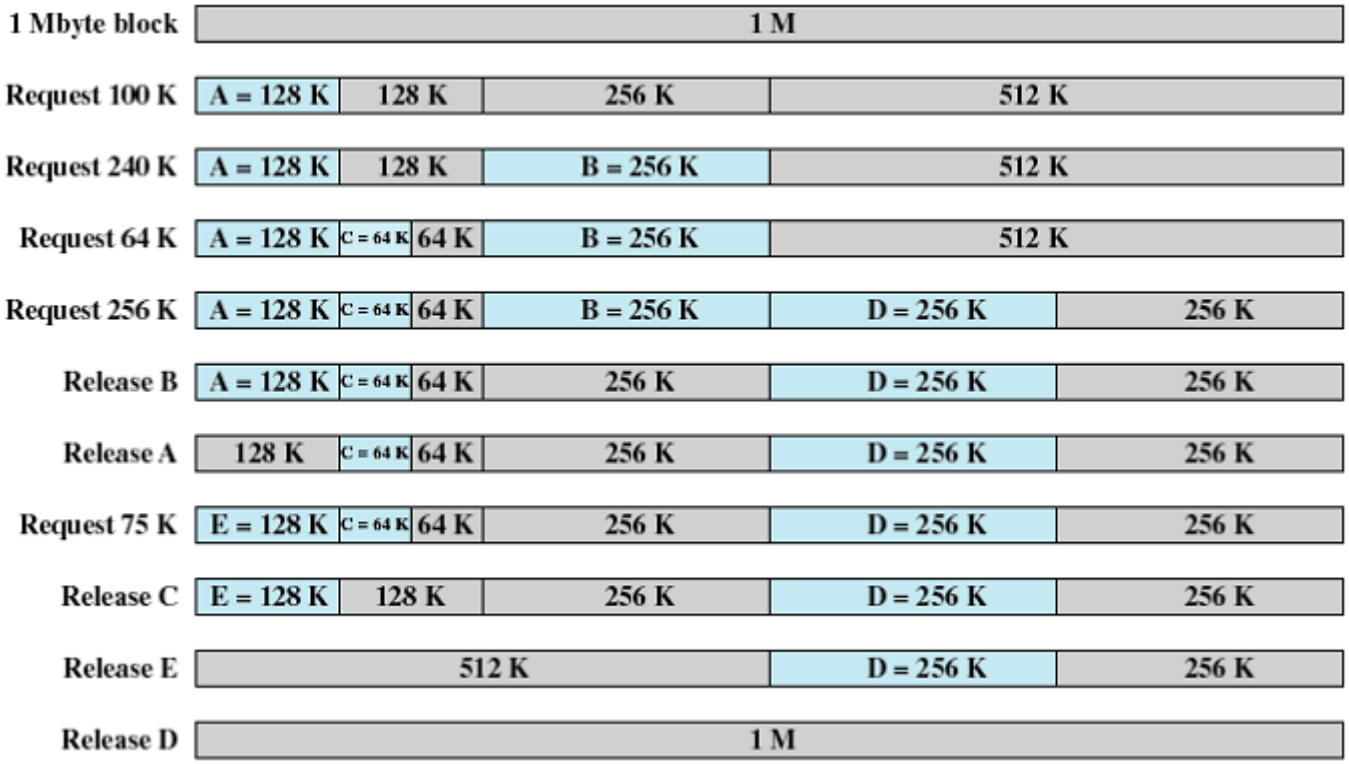
Como atribuir um bloco de memória de dimensão *n* ?

- **Gestão por Listas Ligadas**
  - **First Fit** : Procura o primeiro espaço livre, sempre começando do início
  - **Next Fit** : Procura o primeiro espaço livre começando do anterior
  - **Best Fit** : Procura o melhor espaço em toda a lista
    - Leva há existência de blocos livres muito pequenos
  - **Quick Fit** : Mantém uma tabela de listas cada uma com blocos de um determinada dimensão
  - **Worst Fit** : Procura o maior espaço em toda a lista
    - Leva a que os blocos livres tenham uma dimensão maior

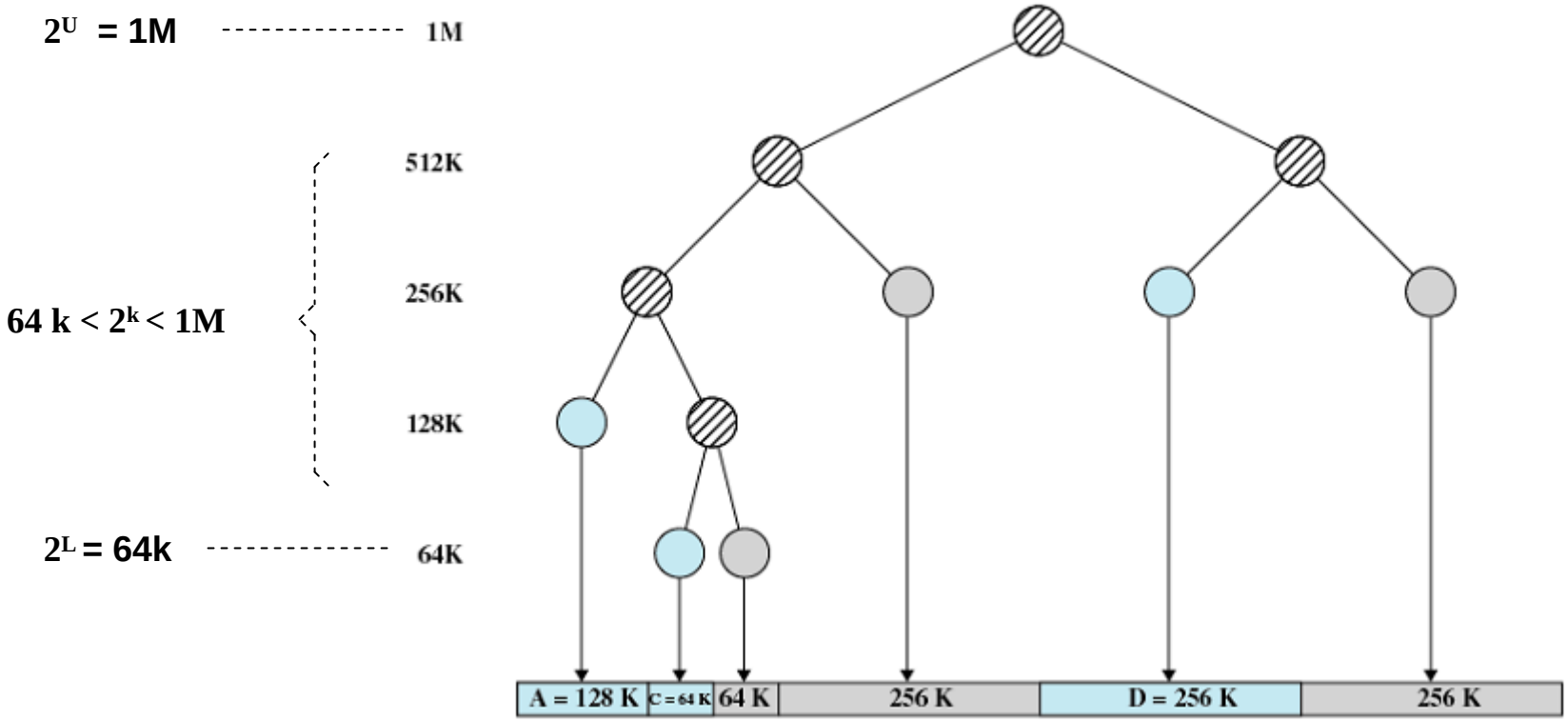


Gestão Memória: Atribuição de memória contigua – atribuição dinâmica

- **Sistema *Buddy***
  - Inicialmente, toda a memória livre constitui um bloco de dimensão 2^U
  - Quando existe necessidade de atribuir memória, os blocos livres são sempre partidos em 2 e assim sucessivamente até que se obtém um bloco capaz de albergar a dimensão pretendida
  - Existem blocos livres de dimensão 2^k satisfazendo condição $2^L <= 2^k <= 2^U$ onde 2^L corresponde ao bloco de menor dimensão pretendida e 2^U a toda a memória inicialmente livre



- **Sistema *Buddy***
  - Facilmente implementável através de uma estrutura em árvore binária. As folhas da árvore contêm os blocos livres
  - Também facilita a compactação que é feita sempre que dois blocos adjacentes ficam livres



Gestão Memória: Fragmentação

- **Fragmentação Externa**

  Este fenómeno verifica-se após o gestor de memória estar em funcionamento um tempo mais ou menos extenso. Durante esse tempo são criados e libertados blocos de memória com dimensão variável; se não forem tomadas precauções corre-se o risco de toda a memória livre estar dividida em pequenos blocos que não é possível recuperar para utilização futura. Estes blocos fazem parte da fragmentação externa. Alguns autores até sugerem a seguinte fórmula para medir a fragmentação numa escala de zero a cem:

  Fragmentação = [1 – (Dimensão_maior_bloco_livre / Soma_dimensões_de_todos_blocos_livres)]x100

  Onde 0 significa ausência de fragmentação e 100 toda a memória fragmentada

- **Fragmentação interna**

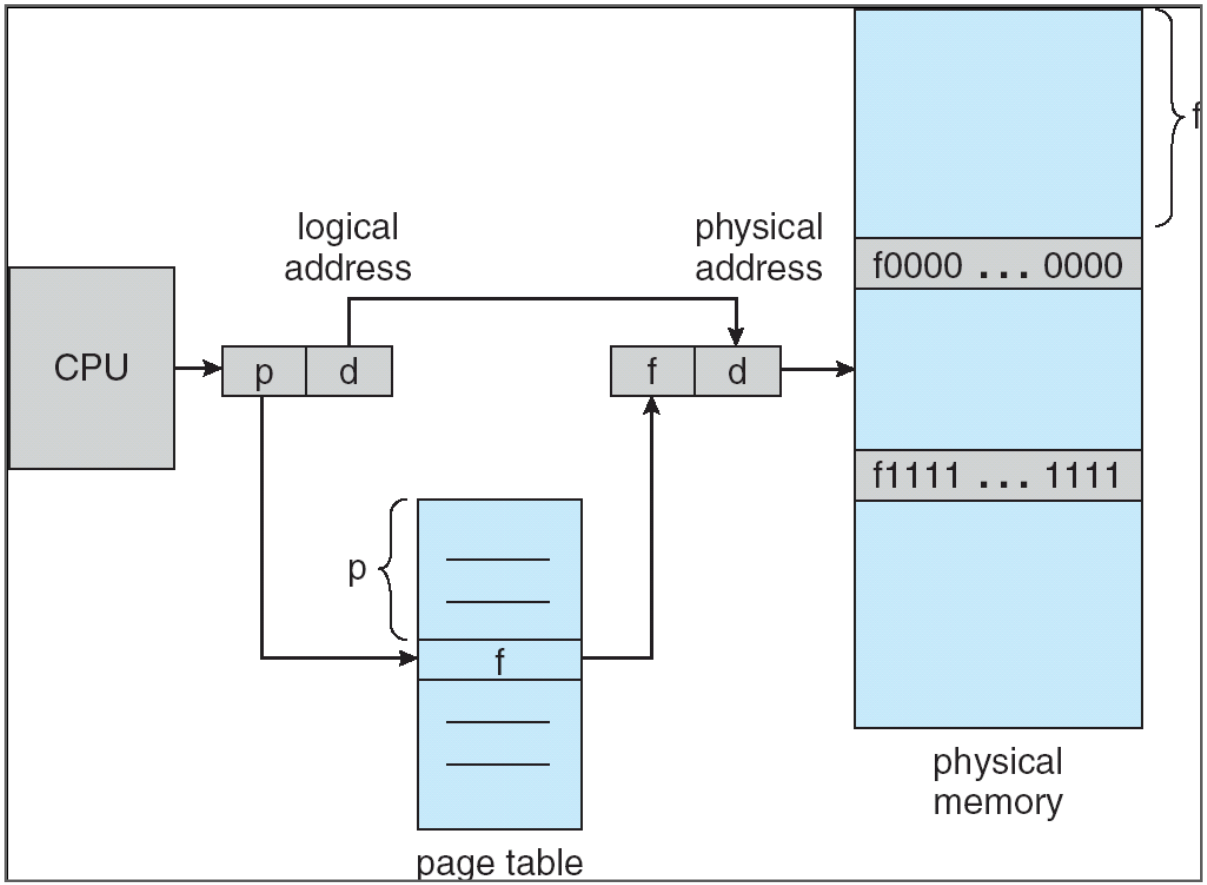
  Este fenómeno acontece quando o gestor de memória satisfaz o pedido com um bloco de memória com dimensão maior do que aquela que lhe foi pedida; neste caso o excesso não é usado. Por exemplo se o gestor de memória limitar a dimensão mínima dos blocos que gere a 16 *bytes*, um pedido de 4 *bytes* vai fazer que existam 12 *bytes* não utilizados nesse bloco

Gestão Memória: Paginação

- Permite que o espaço de endereçamento físico de um processo não necessite de ser contíguo
- A memória física é dividida em blocos de dimensão fixa designados de ***frames*** (tipicamente a dimensão é um valor potência de 2 com valores entre 4KB e 16KB)
- O espaço de endereçamento lógico é dividido em blocos da mesma dimensão designados de **páginas**
- A zona de *swapping* é dividida em blocos de dimensão igual à dimensão das *frames* e páginas
- É necessário gerir as *frames* que se encontram livres (tabela de *frames*)
- A execução de um programa que ocupe n páginas implica encontrar qualquer n *frames* livres sem que estas se encontrem contíguas
- Existência de uma tabela de conversão de endereços lógicos em físicos (esta conversão é controlada pelo SO e escondida do utilizador)
- Separação entre o espaço de endereçamento lógico (visto como um espaço linear) e o espaço de endereçamento físico (que pode estar espalhado pela memória física)
- Existência de fragmentação interna
- Ausência de fragmentação externa

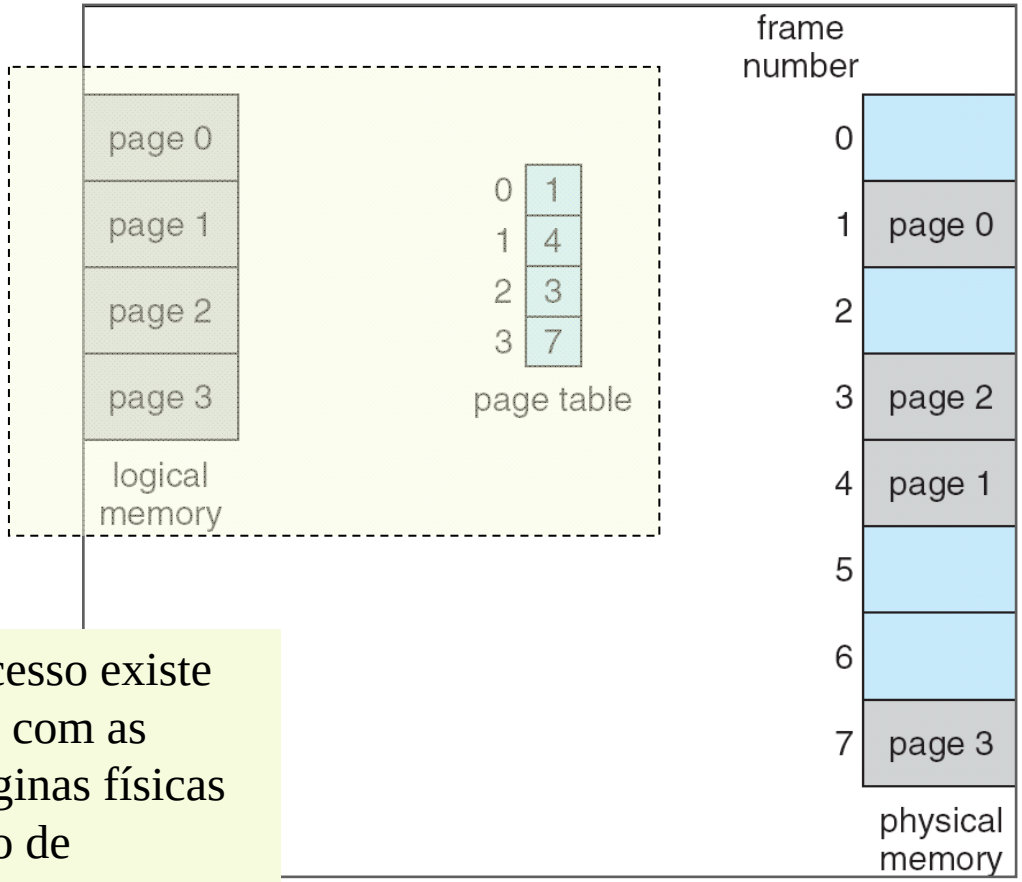
O endereço lógico é dividido em:

- Número Página – serve de índice à tabela de página que contém os endereços base de cada página na memória física
- Deslocamento na Página combinado com o endereço base da página na memória física defini o endereço físico



Gestão Memória: Paginação

Separação entre o espaço de endereçamento lógico (visto como um espaço linear) e o espaço de endereçamento físico (que pode estar espalhado pela memória física)

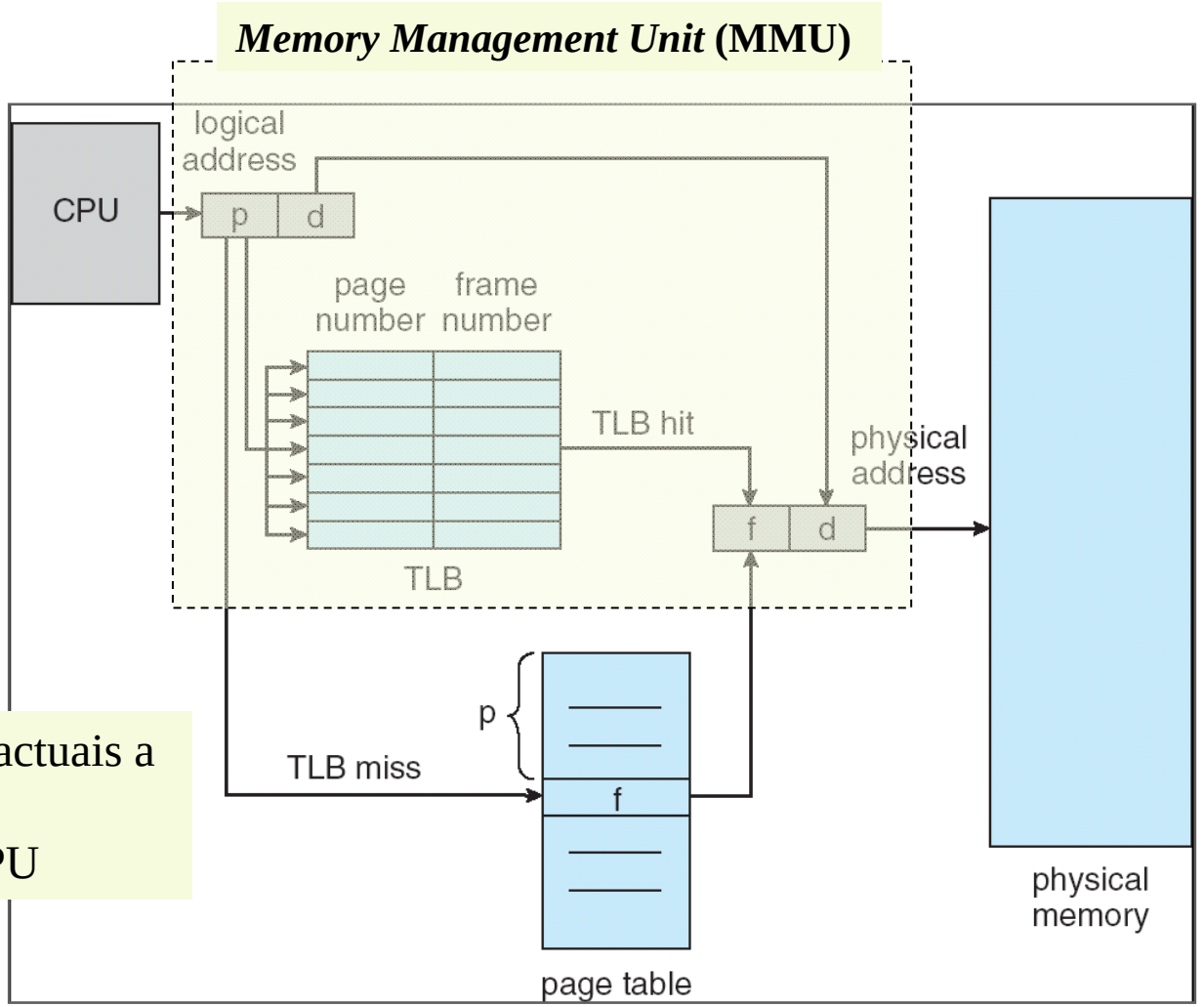


Associado a cada processo existe uma tabela de páginas com as referências para as páginas físicas que suportam o espaço de endereçamento virtual do processo

Gestão Memória: Paginação – suporte hardware

- A tabela de páginas reside na memória principal
- Existe um registo que referencia a tabela de página do processo corrente – ***Page table base register*** **(PTBR)**
- Nesta abordagem cada acesso a dados ou instrução exige 2 acessos a memória: um para consulta da tabela de páginas e outra para acesso aos dados ou instrução
- Para colmatar este problema o hardware deve possuir uma tabela associativa - ***translation look-aside buffer*** **(TLB)**
  - O número de entradas da TLB é reduzido
  - Quando um endereço lógico é gerado pelo CPU a TLB é pesquisada em paralelo e se:
    - Não for encontrada (TLB *miss*) uma entrada relativa a essa página tem de ser feita uma consulta a tabela de páginas para obter a *frame* correspondente. Esta informação é adicionada à TLB
    - Se foi encontrada o endereço físico é gerado de imediato
  - O que acontece à TLB no *context switch*?

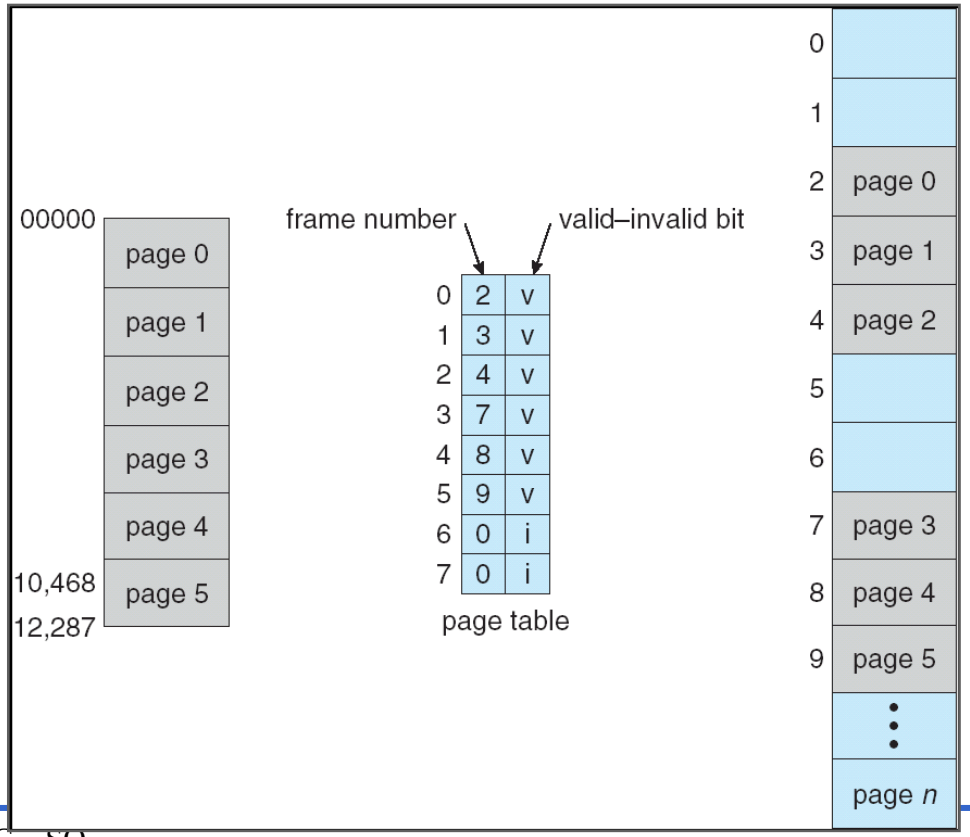
Gestão Memória: Paginação – suporte hardware - TLB



Nos processadores actuais a MMU é uma das componentes do CPU

Gestão Memória: Paginação – protecção

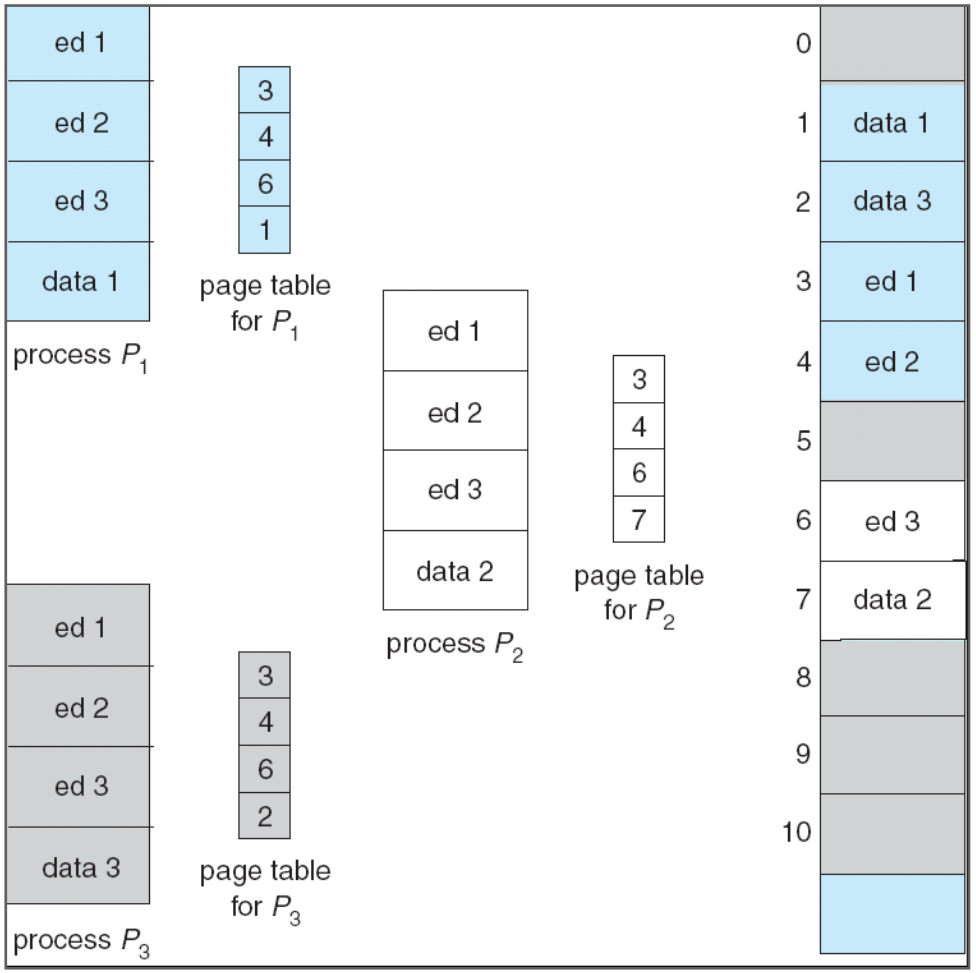
- Protecção de memória realizada associando bits de protecção a cada página
- Bit de página válida é associado a cada entrada da tabela de página:
  - Página válida é uma página válida no espaço de endereçamento do processo
  - Página inválida é uma página que não está atribuído constituindo um acesso ilegal
- Podem existir outros bits indicando permissão de acesso para leitura, leitura e escrita, execução



Gestão Memória: Paginação – partilha de páginas

- **Código Partilhado**
  - O código (das aplicações) é partilhado entre os processos existindo em memória física uma única cópia do código apenas com permissões de leitura
  - O código partilhado tem de estar localizado no mesmo espaço de endereçamento lógico de todos os processos

- **Código e dados privados**
  - Cada processo tem um cópia do código e dados
  - As páginas contendo o código e dados privados podem localizar-se em qualquer parte do espaço de endereçamento



Gestão Memória: Paginação – Estrutura da tabela de páginas

- Os sistemas computacionais actuais suportam espaços de endereçamento lógicos entre 2^{32} a 2^{64}
- Para espaços de endereçamento desta dimensão qual a dimensão da tabela de páginas?
- O sistema de paginação descrito com uma tabela de *lookup* é aplicável para sistemas pequenos. Considerando um sistema actual, com pelo menos 32 *bits* de endereçamento virtual e usando páginas de memória de 4 KB (12 *bits*), temos 20 *bits* para endereçamento lógico de uma tabela de *lookup* com 2^{20} =1 MB entradas (mais de um milhão de entradas)
- O Windows disponibiliza a cada aplicação/processo um espaço de endereçamento próprio de 32 *bits* (4 GB), o que significa que cada aplicação teria de ter uma tabela de um milhão de entradas.
- As técnicas mais comuns na estruturação das tabelas de páginas são:
  - **Tabelas de páginas multinível**
  - **Tabelas de páginas *hash***
  - **Tabelas de páginas invertidas**

Gestão Memória: Paginação – Tabelas de páginas multinível

O sistema descrito anteriormente (uma tabela de *lookup*) é aplicável para sistemas pequenos. Considerando um sistema actual, com pelo menos 32 *bits* de endereçamento virtual e usando páginas de memória de 4 KB (12 *bits*), temos 20 *bits* para endereçamento virtual e uma tabela de *lookup* com 2^{20} =1 M entradas (mais de um milhão de entradas)

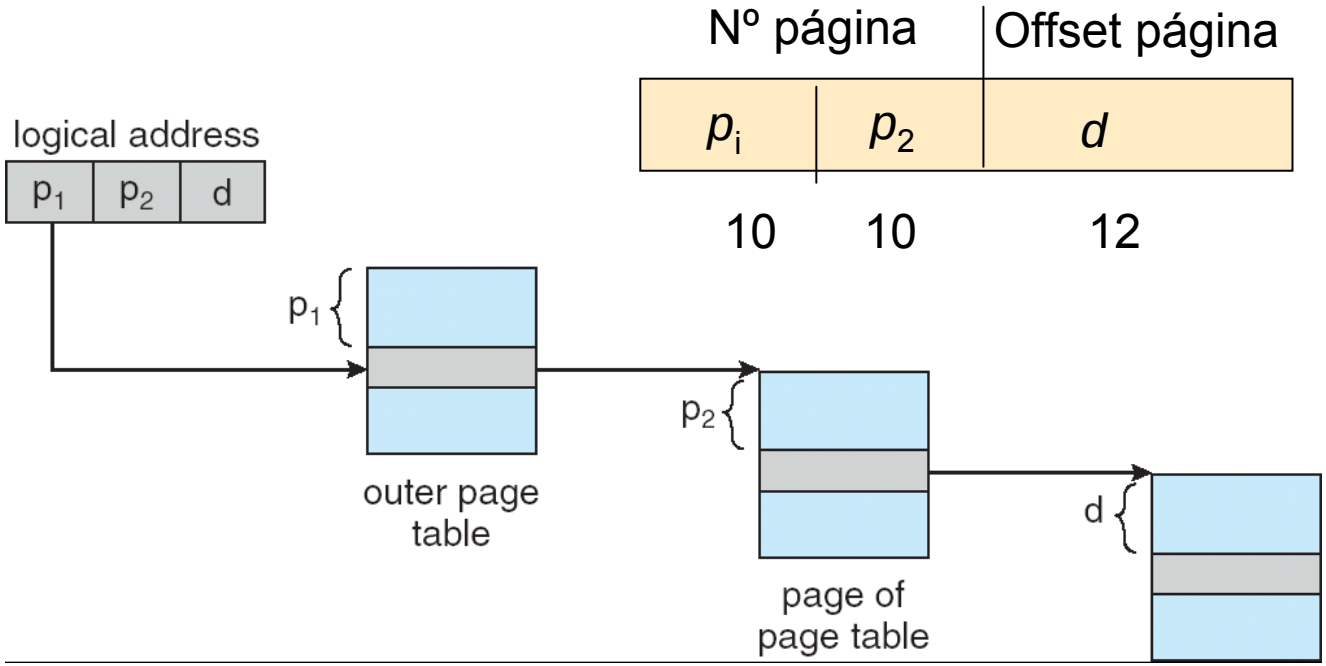
O Windows e Linux, por exemplo, disponibilizam a cada aplicação/processo um espaço de endereçamento próprio de 32 *bits* (4 GB), o que significa que cada processo teria de ter uma tabela de um milhão de entradas

Uma alternativa é dividir o espaço de endereçamento lógico em múltiplas tabelas de páginas, por exemplo, utilizar dois níveis de tabela de páginas

Gestão Memória: Paginação – Tabelas de páginas multinível

- O endereço lógico é dividido em:
  - Número de páginas
  - Offset na página

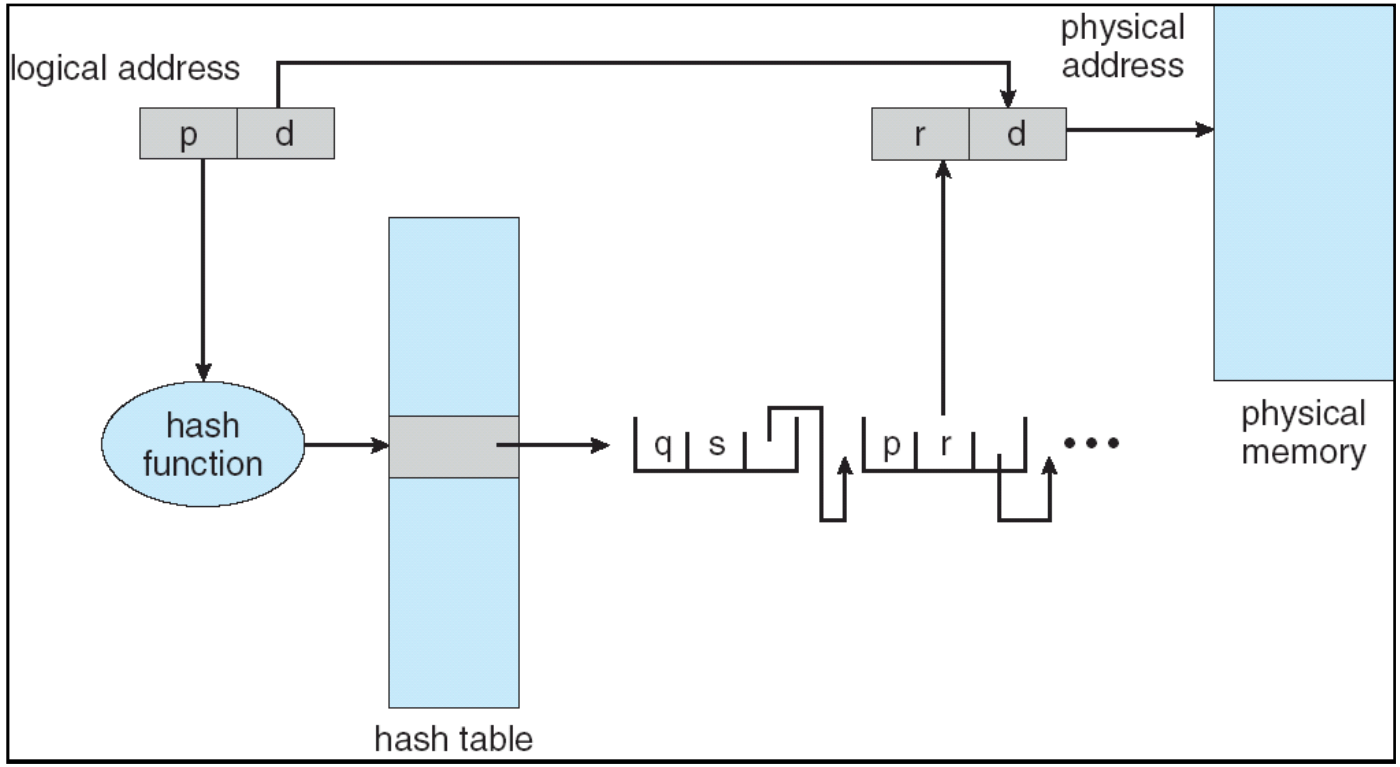
- Como a tabela de páginas vai ser dividida em dois níveis o número de página é dividido em dois grupos de bits formando os índices a utilizar no acesso às tabelas de 1º e 2º nível respectivamente



Exemplo: máquina com endereços lógicos a 32 bits e páginas de 4KB

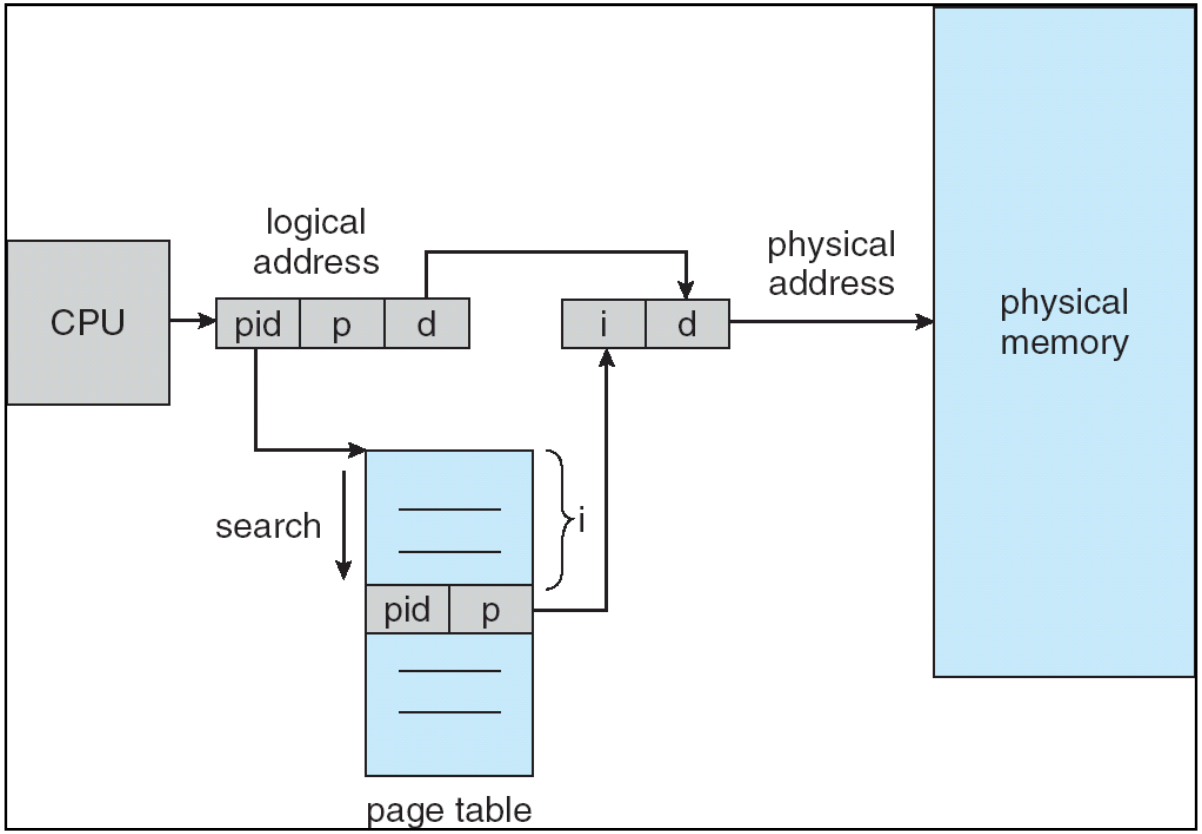
Gestão Memória: Paginação – Tabelas de páginas *hash*

- Solução tipicamente adoptada para espaços de endereçamentos maiores que 32 bits
- O número da página virtual é utilizado como chave para inserção numa tabela de *hash*. Cada entrada da tabela de *hash* contém uma lista de colisões



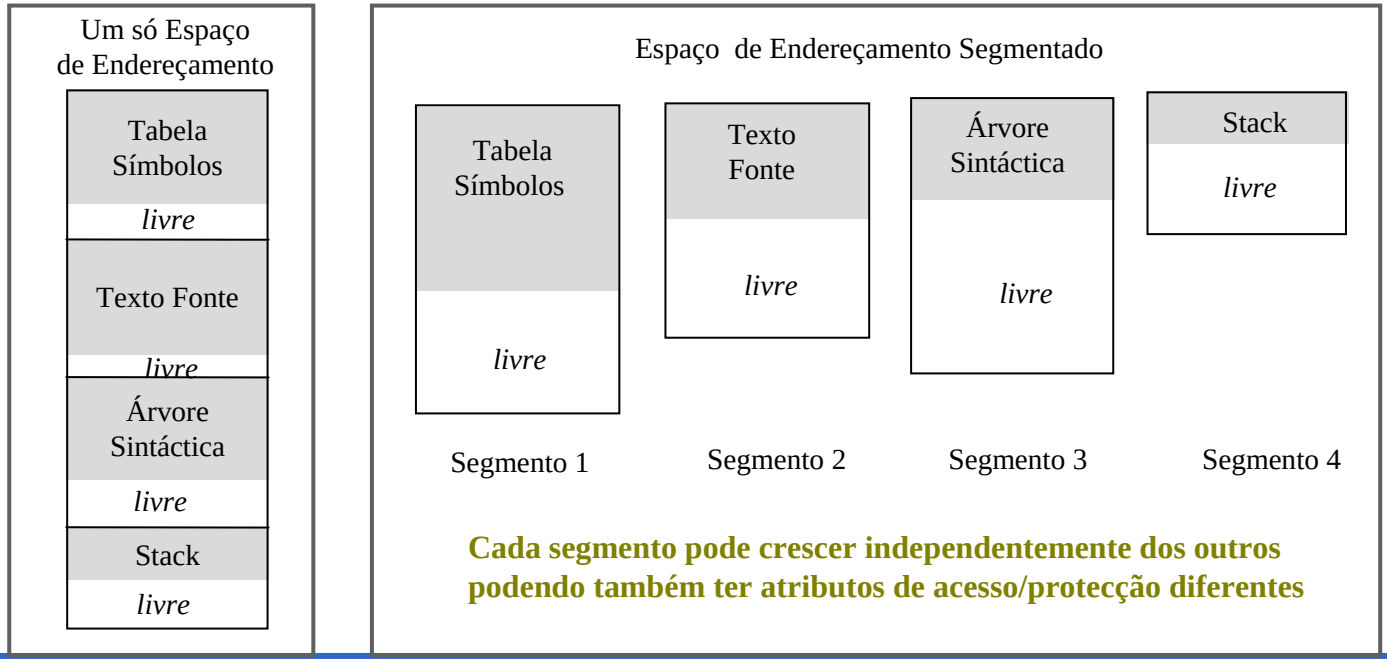
Gestão Memória: Paginação – Tabelas de páginas invertidas

- Tabela com uma entrada por cada página física (*frame*)
- Cada entrada consiste no endereço virtual da página, guardada nessa localização física, e informação de identificação do processo que possui a página
- Reduz a memória necessária para guardar cada tabela de página, aumentando o tempo de pesquisa na tabela quando ocorre um referência
- Optimização na pesquisa pode ser conseguida através de uma tabela de *hash*



Gestão Memória: Segmentação

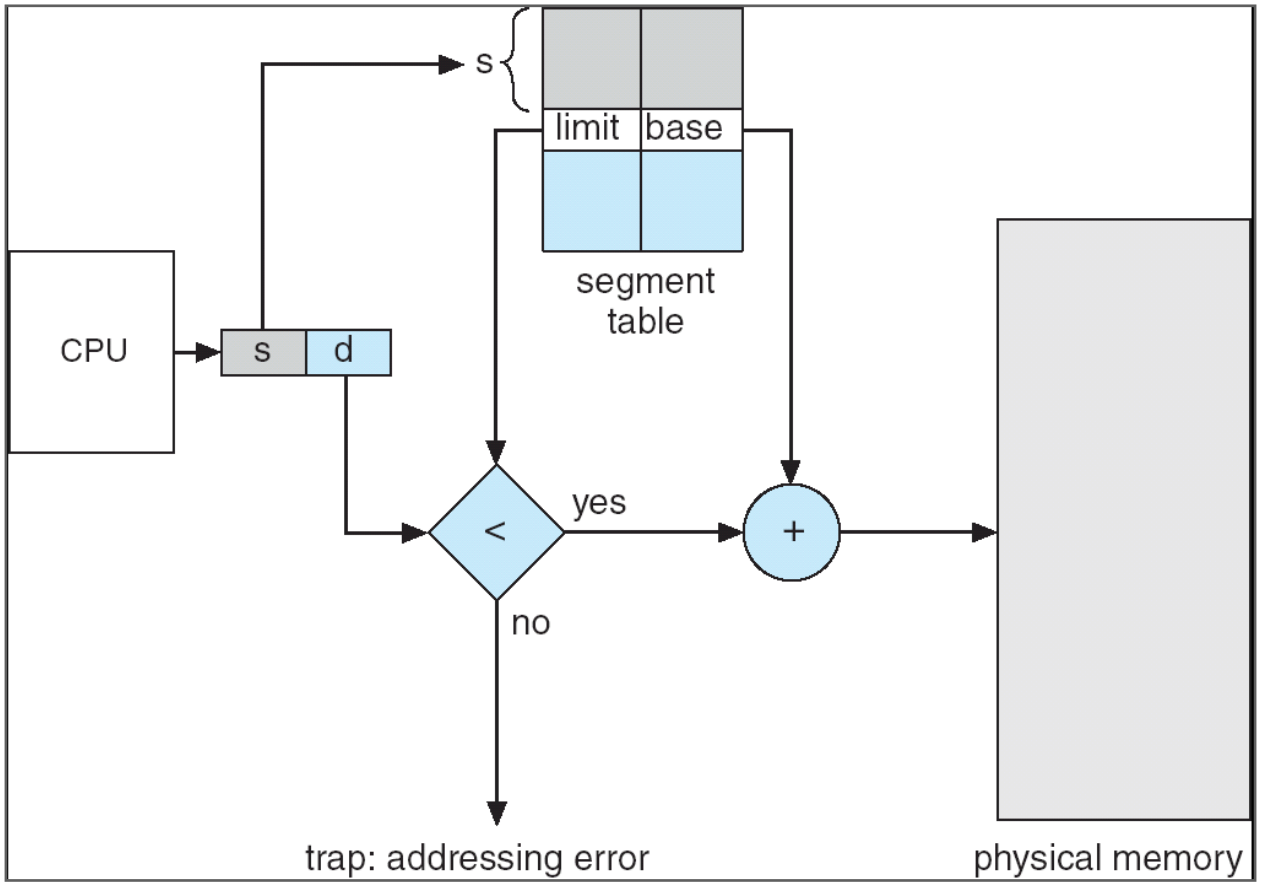
- Organização da memória em **vários espaços de endereçamento**
- Pode ser utilizada para a geração de código mais fácil de ligar (por acção de um *linker*). Se cada procedimento for compilado com base num segmento de código n os seus endereços são feitos à base de (n, 0). Se este variar de dimensão a sua recolocação não afecta os restantes
- **Facilita a partilha** de código, ou memória por vários processos: basta fazer com que o processo possa aceder ao segmento respectivo. Com um único espaço de endereçamento, era necessário fazer o mapeamento da memória partilhada nos espaços de endereçamento dos processos
- Pelo facto de cada segmento formar uma entidade lógica conhecida pelo programador, como por exemplo uma função, um array, stack, código ou dados é possível atribuir-lhe **esquemas de protecção próprio**



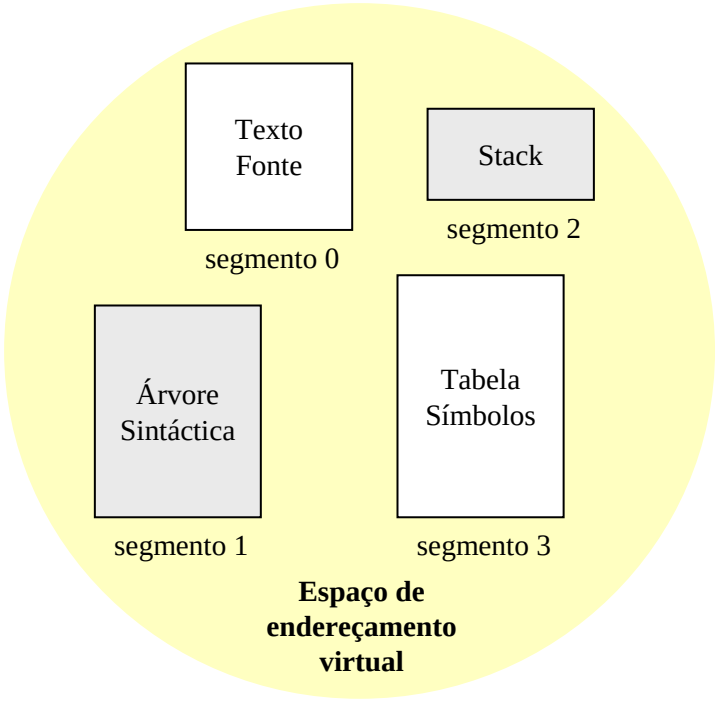
Gestão Memória: Segmentação

- Um **endereço lógico** é composto pela identificação do segmento (número) e um deslocamento dentro desse segmento:
  - <número de segmento>, <offset>
- **Tabela de segmentos –** localiza o segmento em memória física. Cada entrada da tabela tem:
  - Base – endereço físico de início do segmento
  - Limite – dimensão do segmento
- ***Segment-table base register*** **(STBR)** - existência de um registo contendo a localização da tabela de segmentos
- **Protecção –** cada entrada da tabela de segmentos possuí informação adicional sobre o segmento:
  - Segmento válido ou inexistente no espaço de endereçamento do processo
  - Permissões de read, write, execute
- **Partilha –** a partilha de zonas de memória entre processos realiza-se ao nível de segmento, por exemplo, partilha do segmento de código
- Uma vez que os segmentos não têm dimensões iguais a gestão da memória física é realizada através de reserva dinâmica

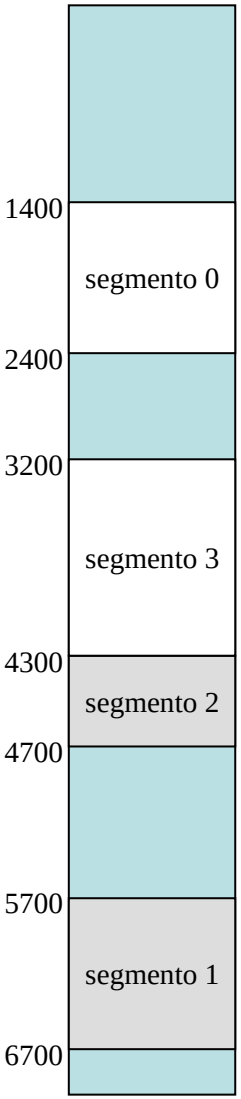
Gestão Memória: Segmentação



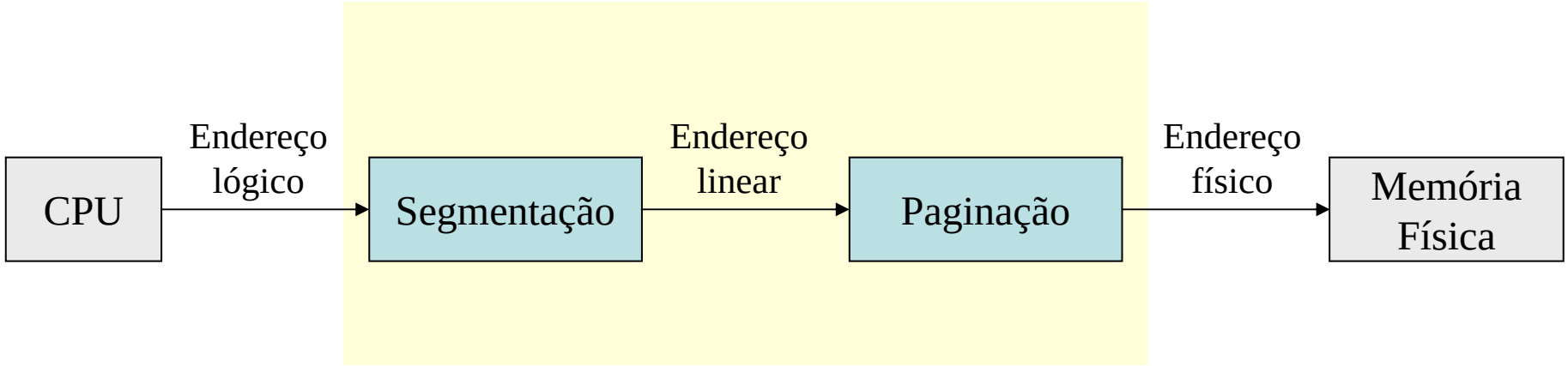
Gestão Memória: Segmentação



	Base	Limite
0	1400	1000
1	5700	1000
2	4300	400
3	3200	1100



Gestão Memória: Modelo híbrido - segmentação e paginação



Gestão Memória: Paginação versus Segmentação

Consideração	Paginado	Segmentado
O programador necessita de estar consciente de que a técnica é usada?	Não	Sim
Quantos espaços lineares existem ?	1	Vários
Pode o espaço de endereçamento total exceder a memória real existente ?	Sim	Sim
Pode-se suportar com facilidade a variação da dimensão de tabelas ?	Não	Sim
Pode-se separar o código e dados com esquemas de protecção próprios ?	Não	Sim
Existe facilidade de implementação de partilha de código entre utilizadores ?	Não ???	Sim
Porque se inventou esta técnica ?	Para obter um espaço de endereçamento linear maior do que a memória física disponível	Para permitir aos programas e dados serem agrupados em módulos lógicos com endereços independentes e para facilitar a partilha e esquemas de protecção

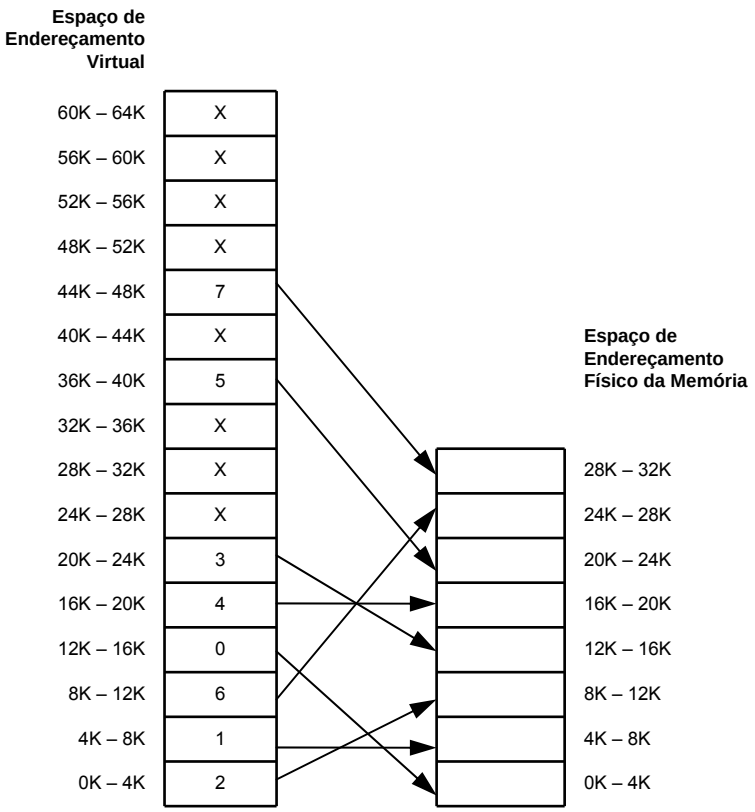
Gestão de Memória Virtual

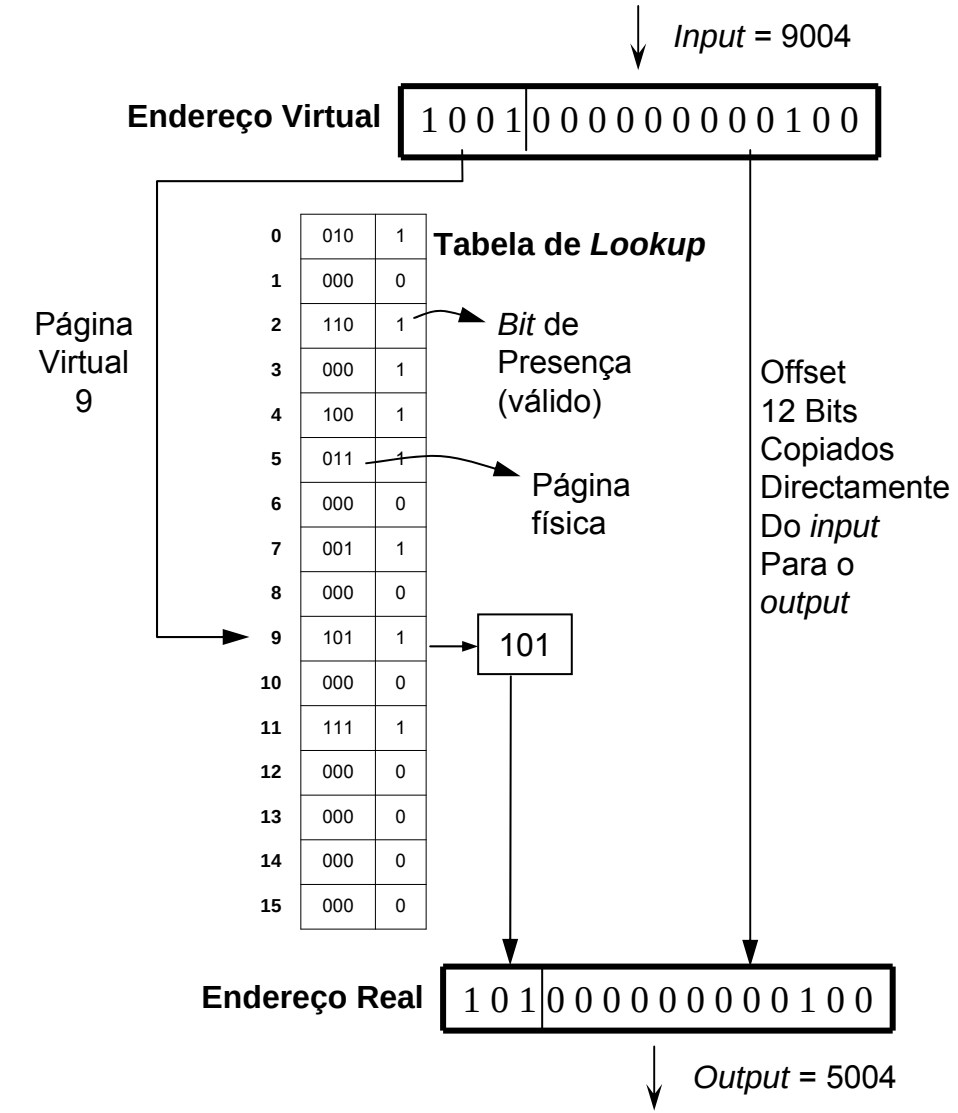
Gestão Memória Virtual

- **Memória virtual**
  - Separação do espaço de endereçamento lógico do espaço de endereçamento físico
  - A execução de um programa não implica que este esteja totalmente carregado em memória
  - Os programas podem usar mais “memória” do que a que está disponível fisicamente
  - O programador deixa-se de preocupar com restrições de memória e com a sua organização
  - Permite a partilha de espaços de endereçamento por vários processos
  - Como os programas necessitam de menos memória física implica que possam existir mais programas parcialmente carregados e em execução concorrente – maior utilização CPU
  - Menos I/O necessário para carregar os programas em memória
  - Criação de processos mais eficiente

- A memória virtual pode ser implementada através:
  - ***Demand Paging***
  - *Demand Segmentation*

Gestão Memória Virtual – *demand paging*





Gestão Memória Virtual – *demand paging*

- As páginas são carregadas só quando são necessárias
  - Menos I/O
  - Menos memória ocupada
  - Resposta mais rápida
  - Mais programas em execução

- Quando uma página é acedida:
  - Referência válida – página residente em memória física
  - Referência inválida -> página não se encontra em memória física gera uma interrupção para o SO (***page fault***)

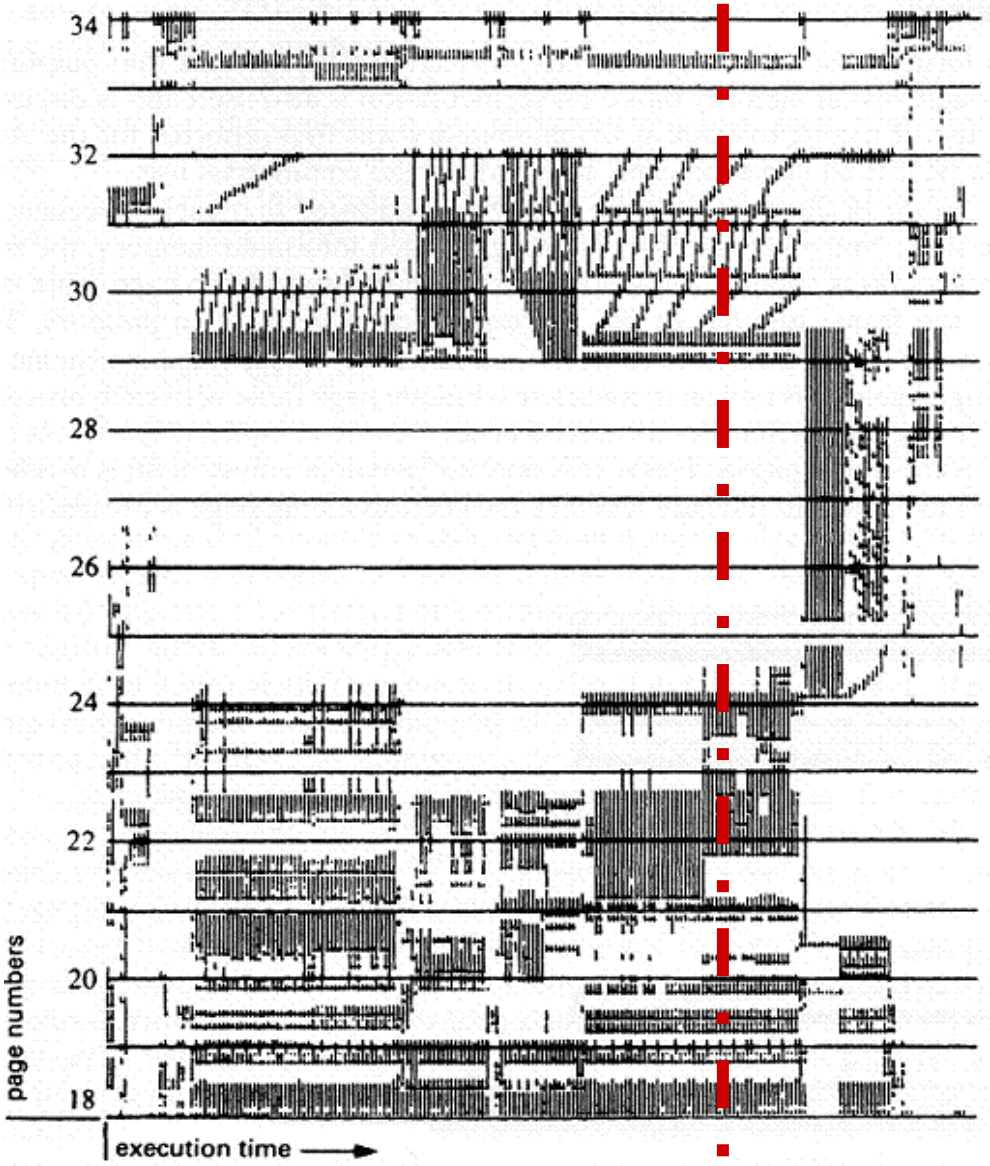
- Esta técnica tem resultados de desempenho razoáveis devido aos programas apresentarem uma característica de localidade (***locality of reference***), isto é, num dado intervalo de tempo a execução do programa necessita apenas de um determinado conjunto de páginas

Utilização da Memória

Ao longo do tempo o processo não necessita de aceder a toda a memória do seu espaço de endereçamento. Os acessos tendem a concentrar-se em zonas bem determinadas (*clusters*) - ***locality of reference***

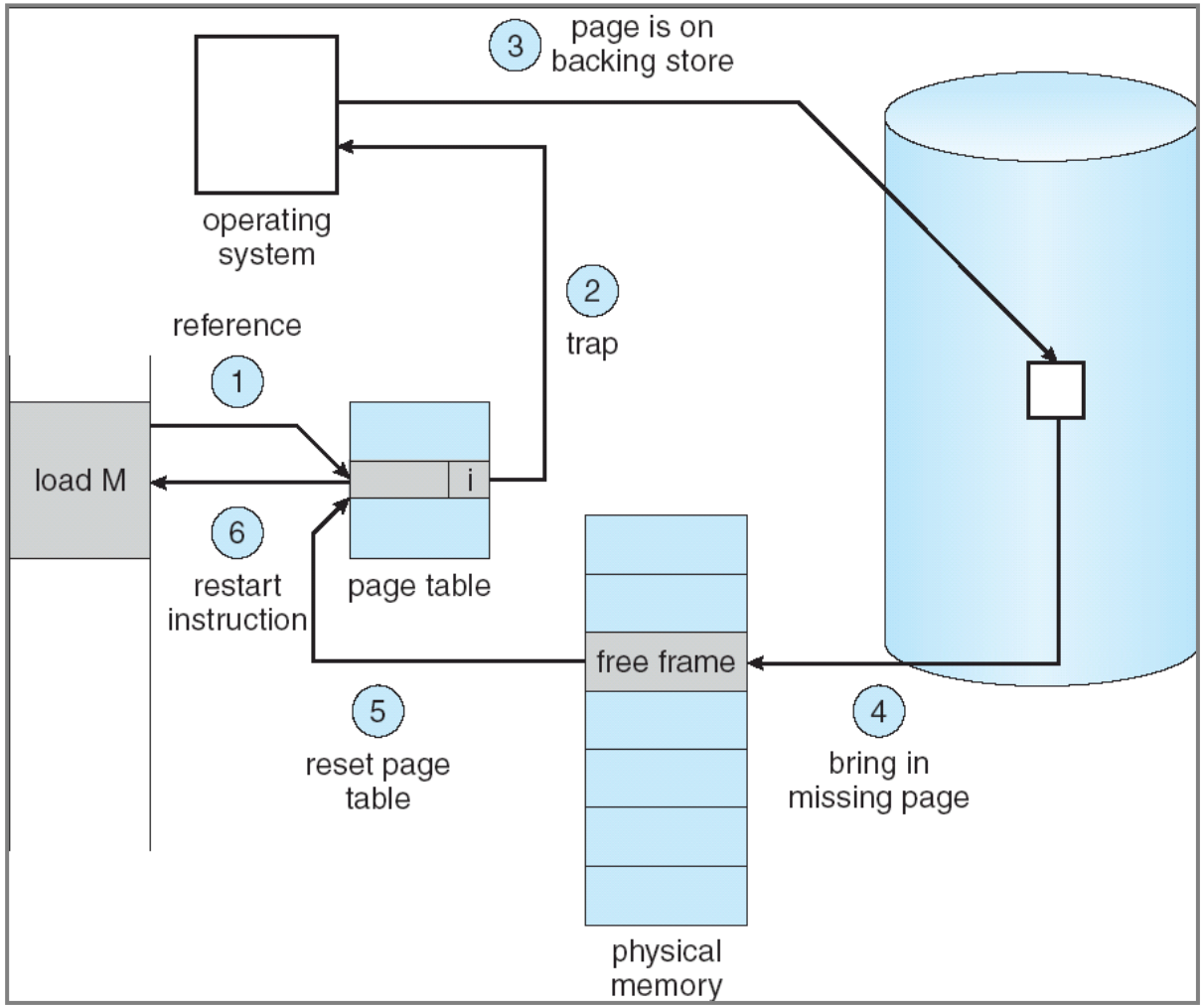
Na figura ao lado ilustra-se este fenómeno. O tempo decorre da esquerda para a direita; na vertical indicam-se as zonas de memória acedidas durante uma janela de tempo de amostra, por exemplo de 20ms em 20ms

O gestor de memória do S.O. será responsável por fazer este tipo de análise e servir o processo de acordo com as suas necessidades

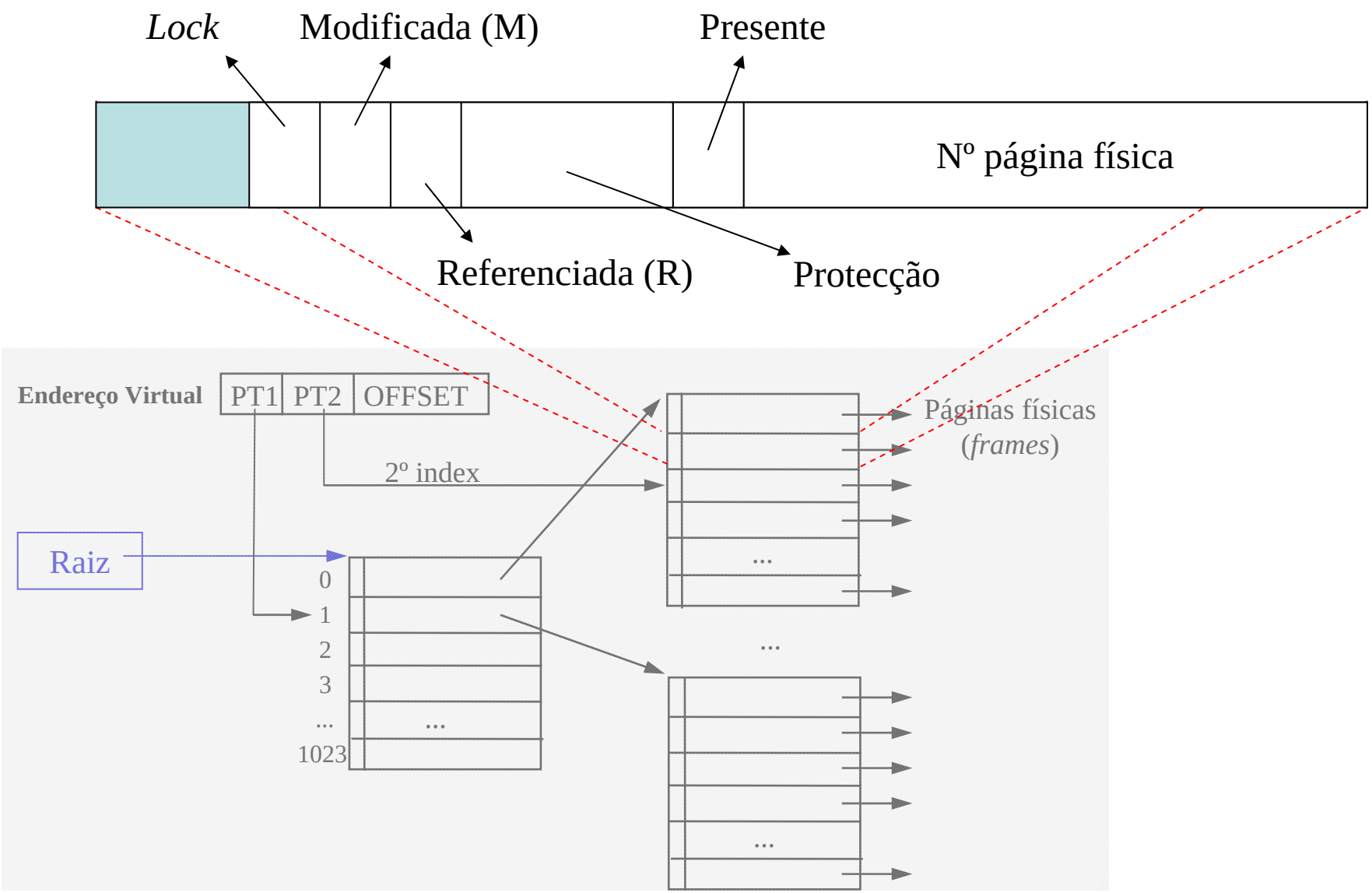


Gestão Memória Virtual – *demand paging*

1. A primeira referência a uma página gera um ***Page Fault***
2. O SO determina se a referência é inválida (endereço não atribuído ao processo) ou se a referência é válida e a página apenas não se encontra em memória
3. Obtém uma *frame* vazia
4. Carrega a página do *swap* para a *frame* seleccionada
5. Altera o bit de presença/válido na tabela de páginas
6. Reinicia a instrução que provocou o *page fault* (requisito essencial no suporte *demand paging*)



Gestão Memória Virtual – *demand paging*

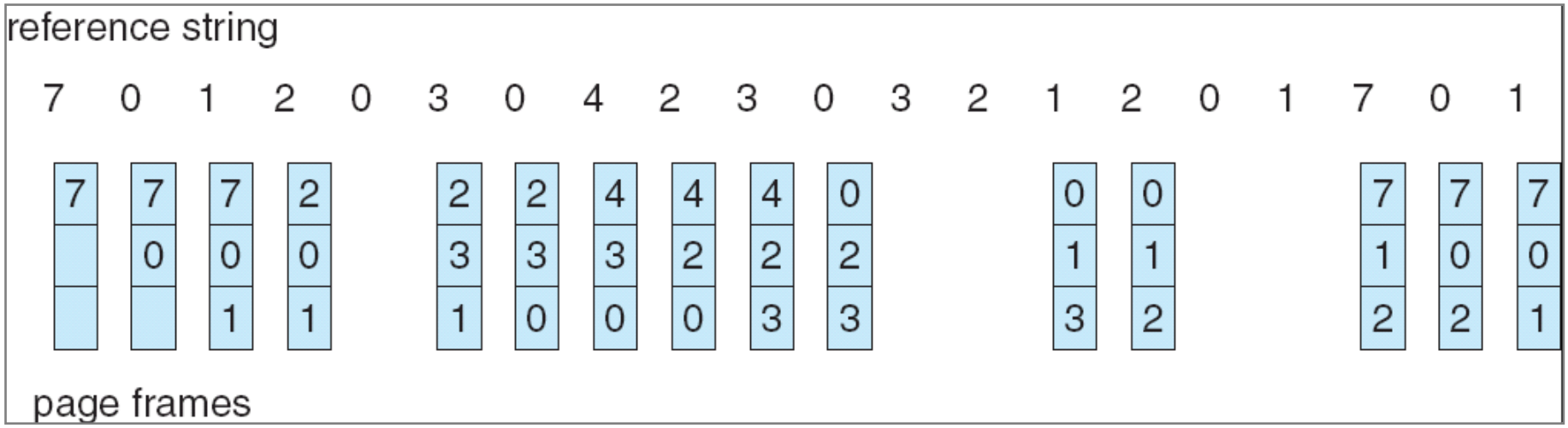


Gestão Memória Virtual: algoritmos de substituição de páginas

- Quando ocorre um *page fault* o gestor de memória de SO tem de carregar numa *frame* a pagina pretendida
- O que fazer se não existir nenhuma *frame* livre?
- O SO pode eleger uma *frame* e retirá-la ao processo a que estava associada implicando:
  - Actualizar a *frame* na zona de *swap*
  - Actualizar a tabela de páginas do processo a que pertencia a *frame*
  - E ler a nova página para a *frame*
  - E actualizar a tabela de páginas do processo que gerou o *page fault*
- Mas qual o critério de selecção de uma *frame*?
  - Uma qualquer?
  - Uma que não seja necessária no maior período de tempo?
- O critério de selecção adoptado influência o ritmo de *page faults*

Gestão Memória Virtual: substituição de páginas - FIFO

FIFO (*First In First Out*) Substituição as páginas segundo a ordem de atribuição de endereços



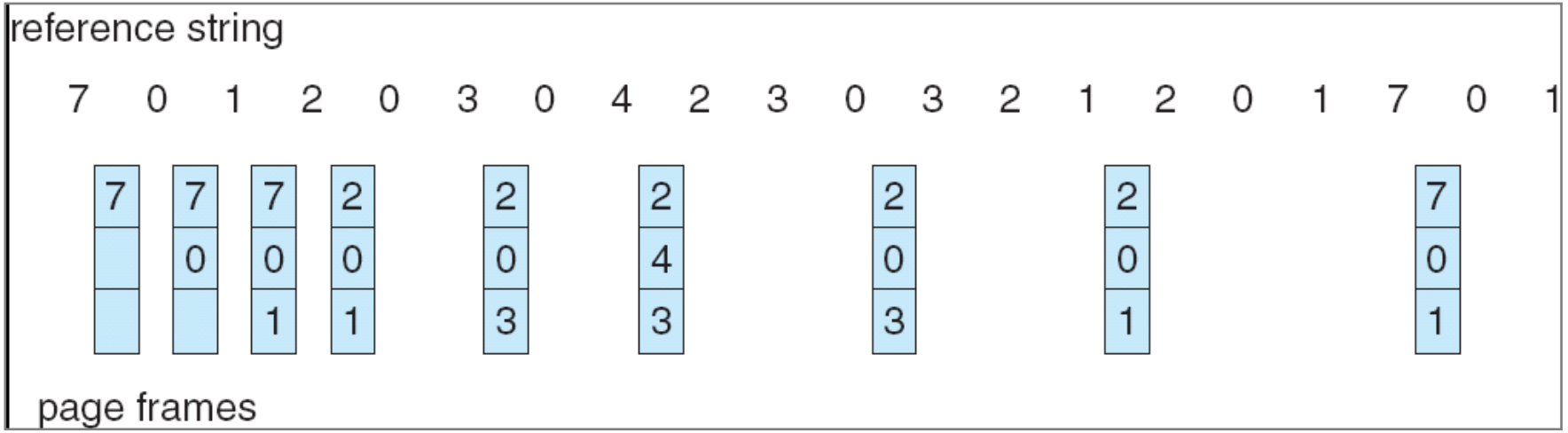
Independentemente do factor de utilização da página elege a página carregada à mais tempo

Gestão Memória Virtual: substituição de páginas – Óptimo !

O critério ideal é fácil de enunciar, todavia impossível de implementar. Será aquele que transfere para disco as páginas que se espera não serem necessárias durante os tempos mais próximos. Resta o problema de saber quais são

Na prática, aplicam-se alguns algoritmos que tentam aproximar-se do óptimo tendo, naturalmente, vantagens e inconvenientes

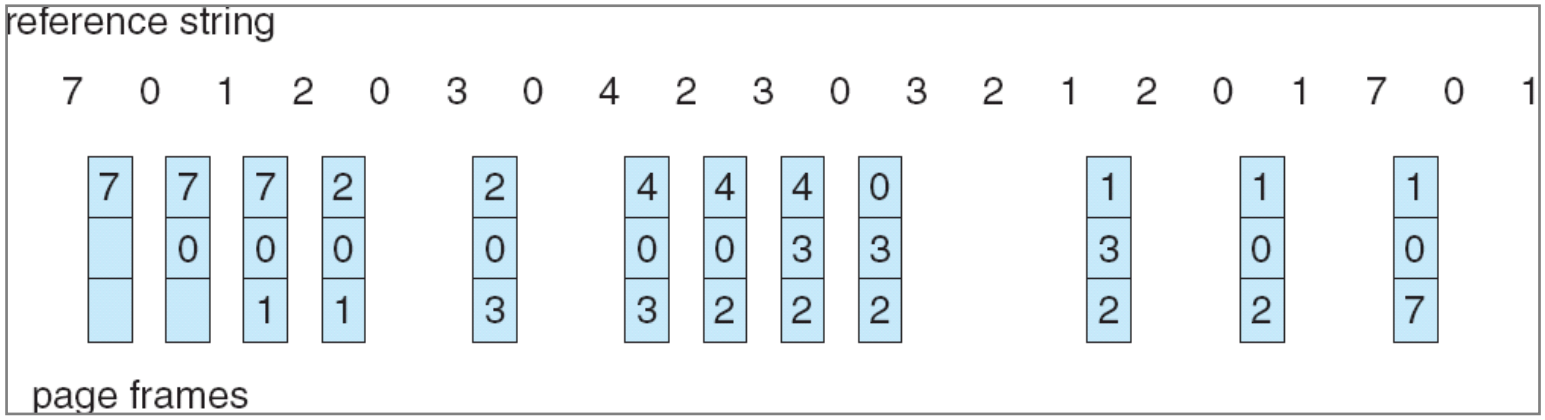
O óptimo pode servir de factor comparativo na avaliação dos outros algoritmos



Gestão Memória Virtual: substituição de páginas - LRU

***Least Recently Used* (LRU)**

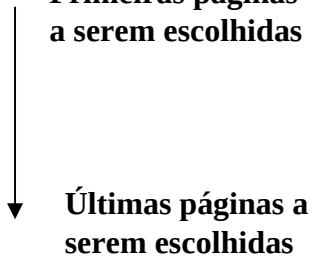
- Difícil implementação
  - Contador
    - Associar a cada página um contador
    - Sempre que a página for referenciada copiar o clock para o contador
    - Escolher a página com o valor de contador menor
  - *Stack*
    - Manter um *stack* das páginas em lista
    - Mover página referenciada para o topo
    - Escolher a página de baixo
- Qualquer das implementações necessita de suporte *hardware* para além da TLB



Gestão Memória Virtual: substituição de páginas – aproximação LRU

- Poucos sistemas possuem suporte hardware para implementarem o LRU
- Alguns hardwares dão uma ajuda com a existência de um bit de referência
  - Cada página tem associado um bit R inicialmente a 0
  - Quando a página é referenciada o bit é alterado para 1
  - Substituir uma página que tenha o bit R = 0
- ***Second chance***
  - Baseado no **FIFO**
  - Utiliza o bit de referência
  - Escolhe a página mais antiga (como no **FIFO**), no entanto se possuir o bit R=1 dá uma segunda oportunidade movendo-a para o fim da Fila
  - Existe uma variante suportada numa fila circular, conhecida como algoritmo do relógio (***clock***) que optimiza a implementação do ***second chance***
- ***Not Recently Used* (NRU)**
  - Um melhoramento do ***Second chance*** utilizando o bit de referência (R) e o de modificação (M)
  - Com este par de bits estabelece a seguinte classificação

classe	R	M
1	0	0
2	0	1
3	1	0
4	1	1



Gestão Memória Virtual: substituição global versus local

- **Substituição Global**
  - É escolhida uma página física do conjunto de todas as páginas físicas existentes
  - Um processo pode retirar uma página física a outro processo

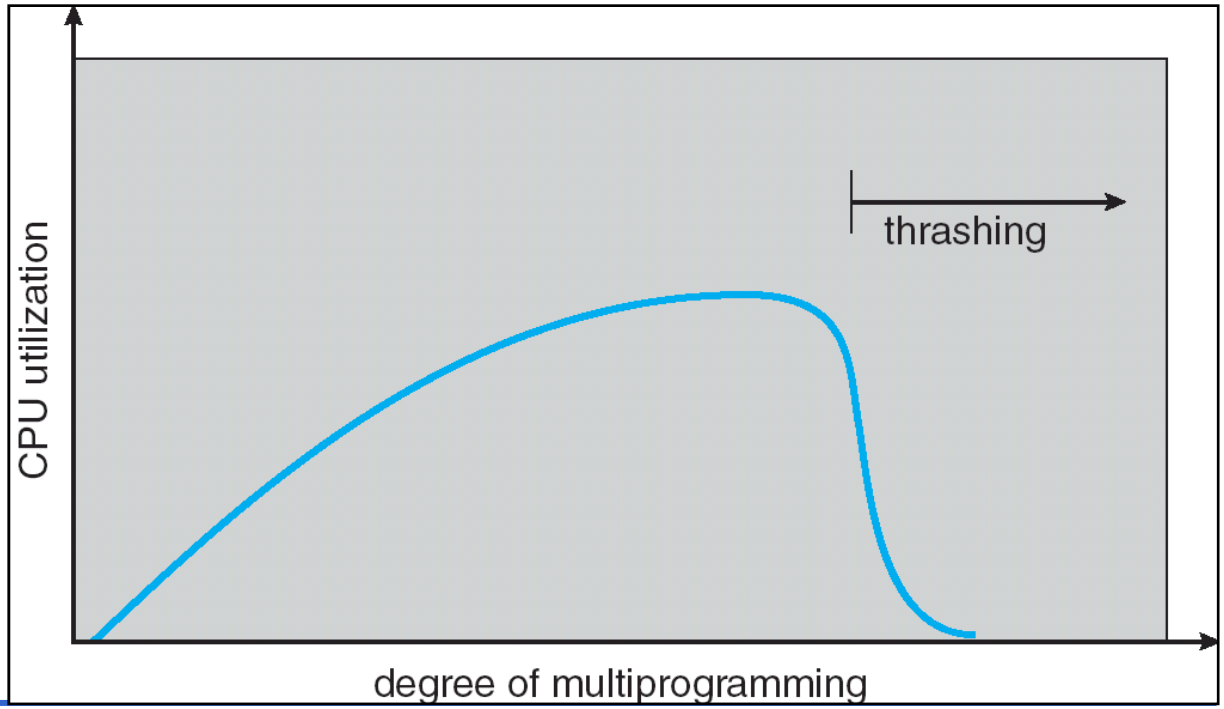
- **Substituição Local**
  - Cada processo escolhe uma página física do seu conjunto de páginas físicas atribuídas

Gestão Memória Virtual: *Trashing*

- Se um processo não tem páginas suficientes, o ritmo de faltas de página aumenta
  - Diminuição da utilização do CPU
    - Pode conduzir ao aumento do nível de multiprogramação
  - Aumento do I/O sobre a zona de paginação do disco
  - Poucas páginas físicas disponíveis
- ***Trashing***
  - O processo está a maior parte do tempo à espera que o SO guarde páginas vítimas em disco (*page out*) e carregue páginas do disco (page in)

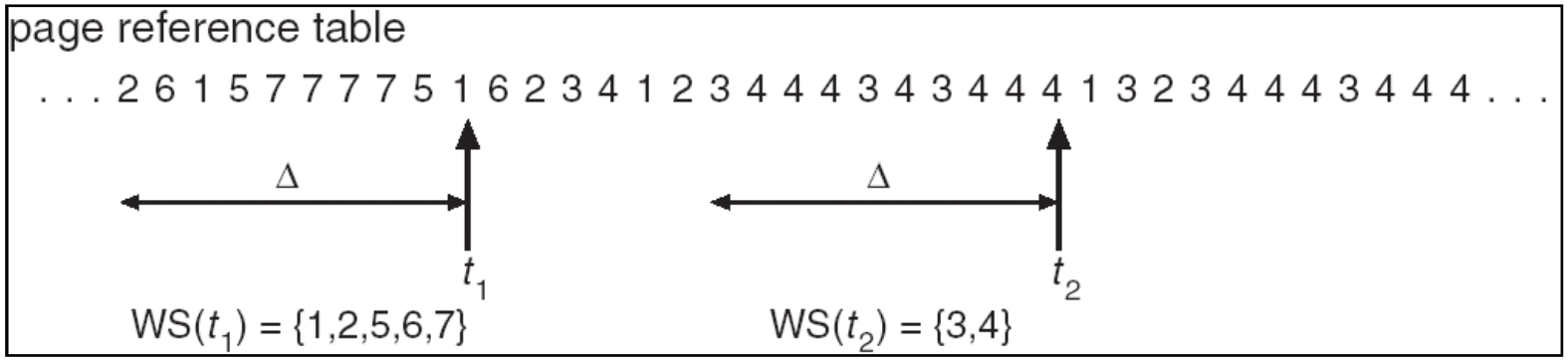
A paginação a pedido (*demand paging*) funciona devido:
- à localidade de referências
- o processo migra de local

Trashing ocorre se
∑ Zona da localidade > Memória Física



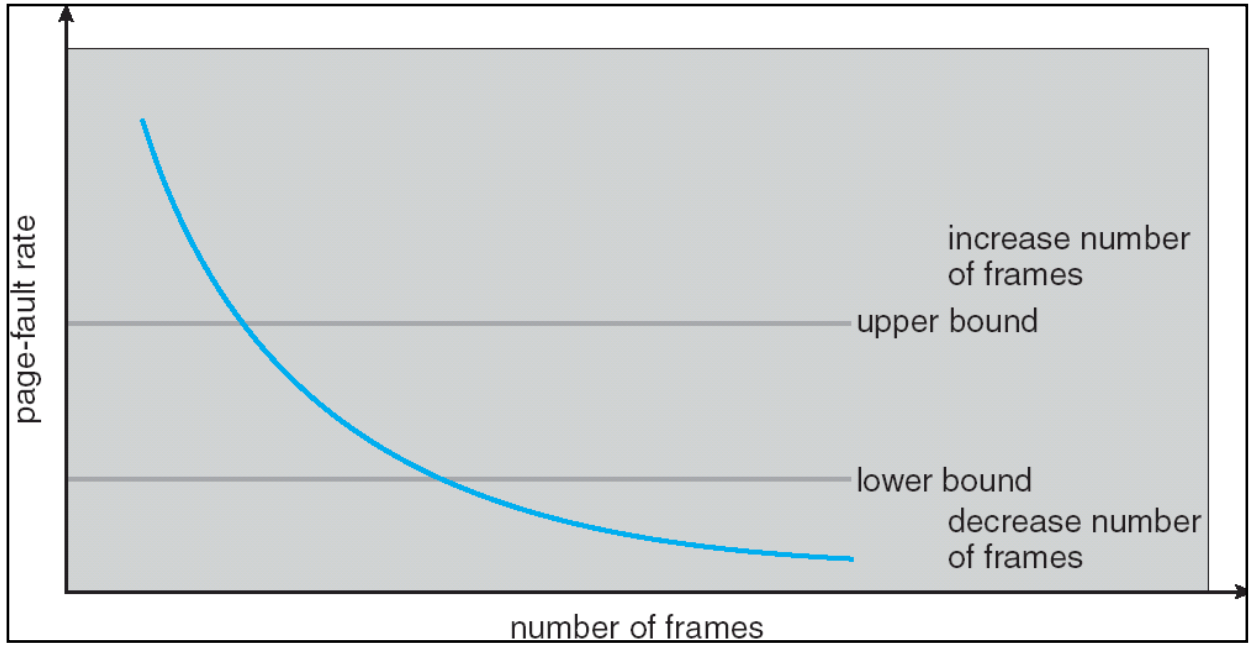
Gestão Memória Virtual: Modelo conjunto de trabalho (*working-set*)

- Um processo quando se executa num determinado intervalo de tempo, necessita de um conjunto de páginas
- O conjunto de páginas pode variar ao longo do tempo



Gestão Memória Virtual: Modelo conjunto de trabalho (*working-set*)

- Controlo do conjunto de trabalho através do ritmo de falta de páginas
  - Ritmo baixo – retirar uma página física ao processo
  - Ritmo elevado – atribuir mais uma página física ao processo



Gestão Memória Virtual: Algoritmo de processamento das falhas de páginas (1)

1. O hardware gera um *trap* de *page fault*, guardando o *program counter* no *stack* do processo. Em algumas máquinas, informação adicional sobre o estado de execução da instrução corrente é guardada em registos especiais do processador.

2. A rotina de interrupção para suporte ao *trap* entra em execução, fazendo a salvaguarda de toda a informação volátil (registos do processador, etc.), fazendo de seguida uma chamada ao sistema de operação.

3. O sistema de operação, sabendo que se trata de uma falha de página, tenta descobrir qual a página do espaço de endereçamento virtual que é necessária. Normalmente os registos hardware contêm essa informação, senão tem que proceder a uma análise do *program counter* do processo interrompido e analisar a instrução que provocou a falha.

4. Uma vez determinada qual a página do endereço virtual que provocou a falha, o sistema de operação detecta se o endereço é válido e a protecção associada coerente com o acesso. Se o acesso é ilegal, o processo recebe um sinal (excepção) ou então é morto. Se é legal, o sistema tenta atribuir uma página livre; se não existem páginas livres, vaga uma das ocupadas segundo um dos critério de substituição de páginas adoptado pelo S.O.

5. Se a página seleccionada está *dirty*, i.e. foi escrita, a página é escalonada para ser transferida para disco, tendo lugar uma comutação de processo: o processo em falta é suspenso, passando o CPU a executar outro até que a transferência se efectue. De qualquer forma, a página em causa é marcada como *busy* para evitar que venha a ser atribuída de novo a outro processo.

Gestão Memória Virtual: Algoritmo de processamento das falhas de páginas (2)

6. Assim que a página esteja livre, isto é, após terem sido guardados os dados que continha, o sistema operativo, detecta qual a zona do disco onde se encontra a informação para carregar nessa página, e desencadeia a operação de leitura dessa nova informação. Continuando o processo suspenso e o CPU a executar outros processos disponíveis.

7. Terminado o preenchimento da página, normalmente detectado por interrupção do controlador de disco, a página é marcada como página normal e também é incluída na tabela de mapeamento de endereços virtuais do processo em causa, por forma que este possa continuar a operação interrompida e que tinha originado a falha.

8. A instrução que originou a falha é reposta no seu estado original e o *program counter* é reiniciado para o inicio da instrução que provocou a falha de página. A instrução é reexecutada.

9. O processo é escalonado e o sistema operativo retorna pela rotina inicial do *trap* que o tinha chamado.

10. A rotina volta a repor os registos e retorna para o início da instrução faltosa, prosseguindo a sua execução normal, como se de uma interrupção normal se tivesse tratado.

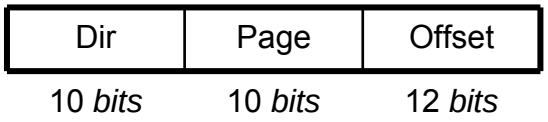
Intel Pentium

- Capítulo 3 do *IA-32 Intel Architecture Software Developers's Manual – Volume 3:System Programming Guide*, Intel (http://www.intel.com)

Gestão Memória: Intel Pentium

- Inclui uma mistura de Segmentação e Paginação
- Possui 6 registos de segmentos (CS, DS, SS, ES, FS e GS) onde cada registo de segmento pode referenciar 8 KB segmentos globais e 8 KB segmentos locais.
- 2 Tabelas
  - ***Local Descriptor Table* (LDT)** associada ao registo *Local Descriptor Table Register* (LDTR)
    - Existe uma por cada processo
    - Descreve os vários segmentos do programa: Código (CS), Dados (DS), *Stack* (SS), etc.
  - ***Global Descriptor Table* (GDT)** associada ao registo *Global Descriptor Table Register* (GDTR)
    - Partilhada por todos os processos
    - Descreve os segmentos partilhados entre processos, incluindo os do próprio Sistema Operativo

Formato de um Endereço Virtual



- Se o processador for configurado para **não usar Paginação**, então o endereço real é calculado apenas da forma:

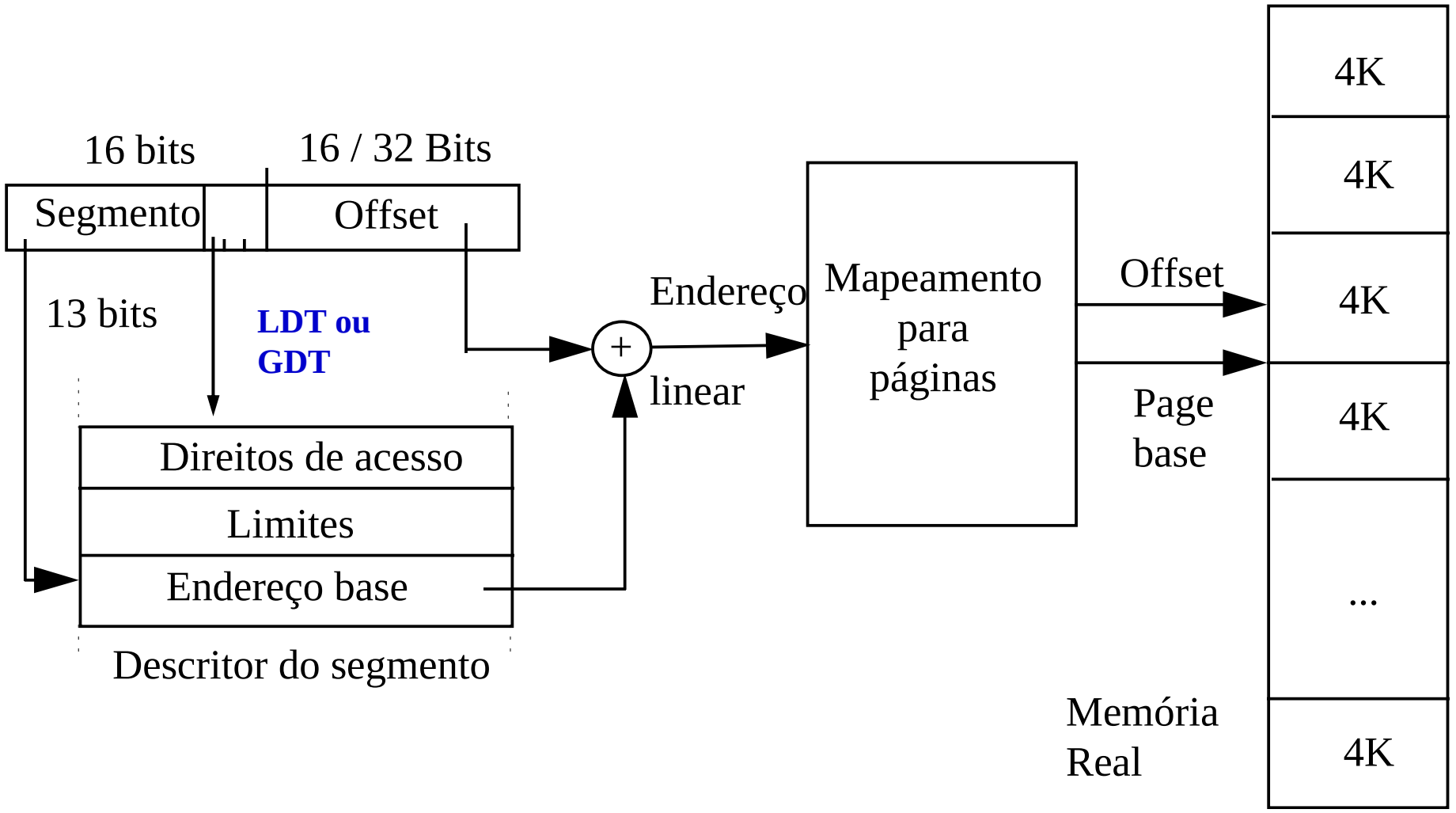
  Endereço Base do Segmento + Offset = Endereço Linear na memória física.

- Se o processador for configurado para **usar Paginação**, então o endereço real é calculado apenas da forma:

  Endereço Base do Segmento + Offset = Endereço Linear;

  Endereço memória física = *MapeamentoVirtualParaReal*(Endereço Linear).

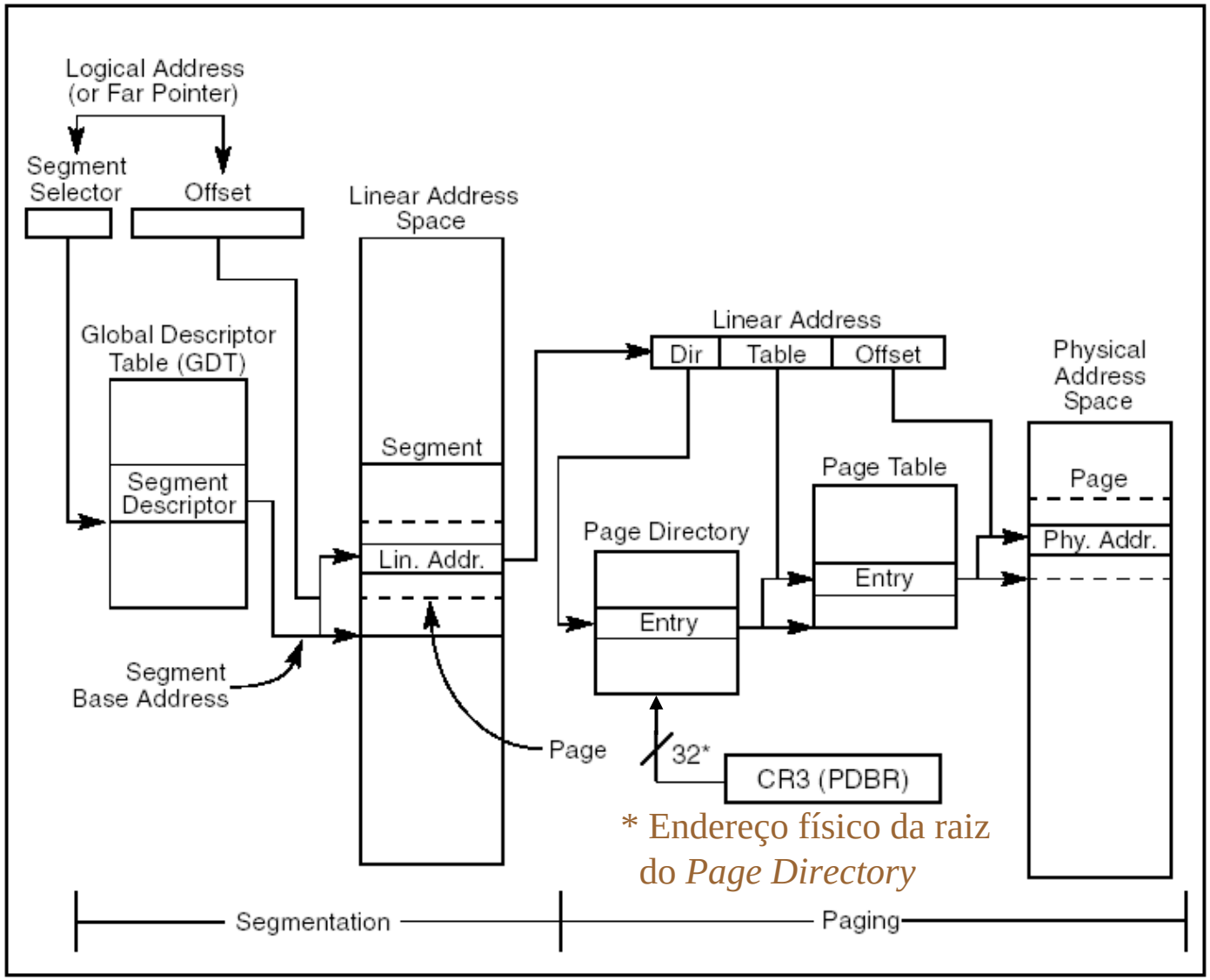
Gestão Memória: Intel Pentium – segmentação/paginação



Gestão Memória: Intel Pentium – segmentação/paginação

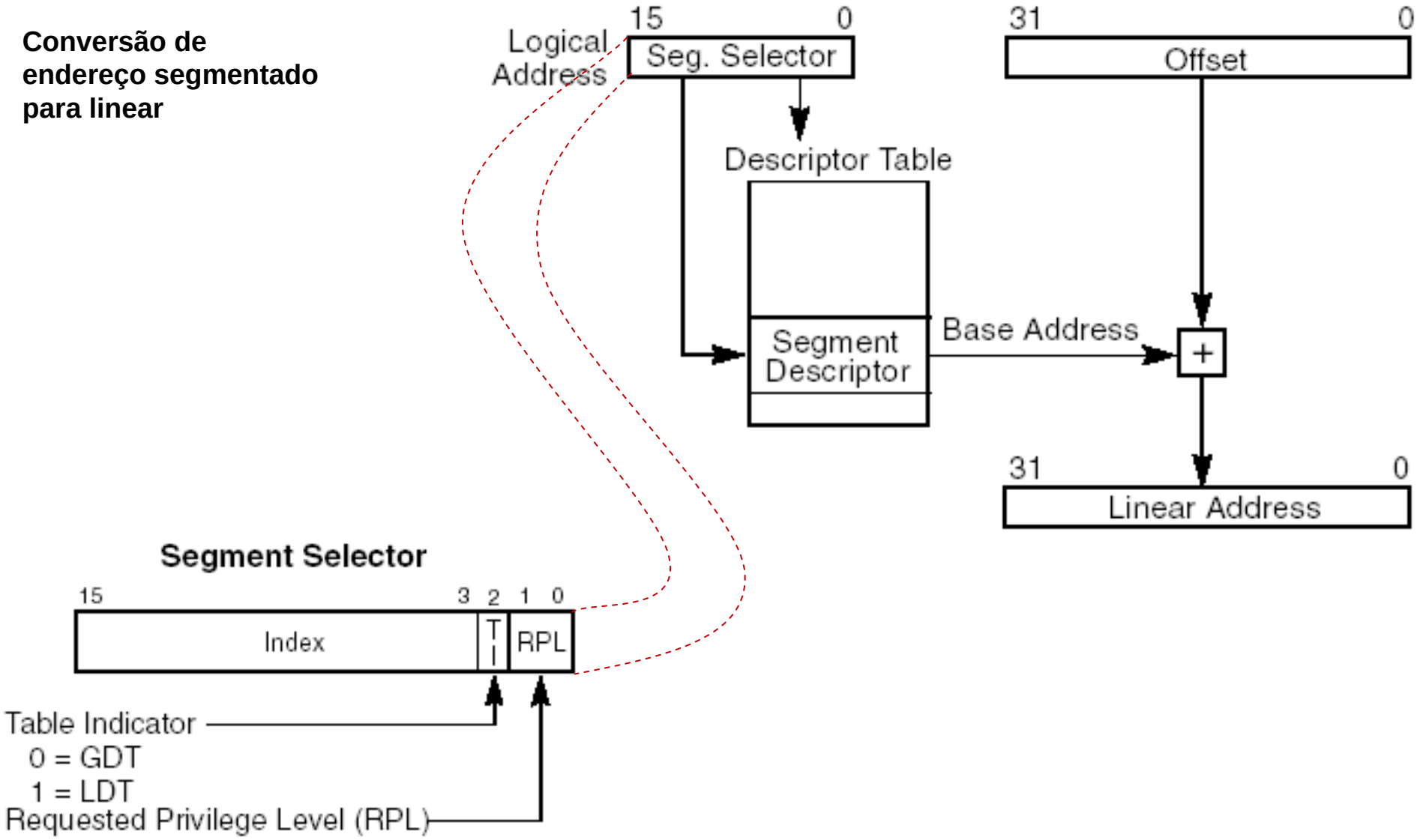
Arquitecturas INTEL 386 e Pentium (modo protegido)

Segmentação e Paginação combinadas



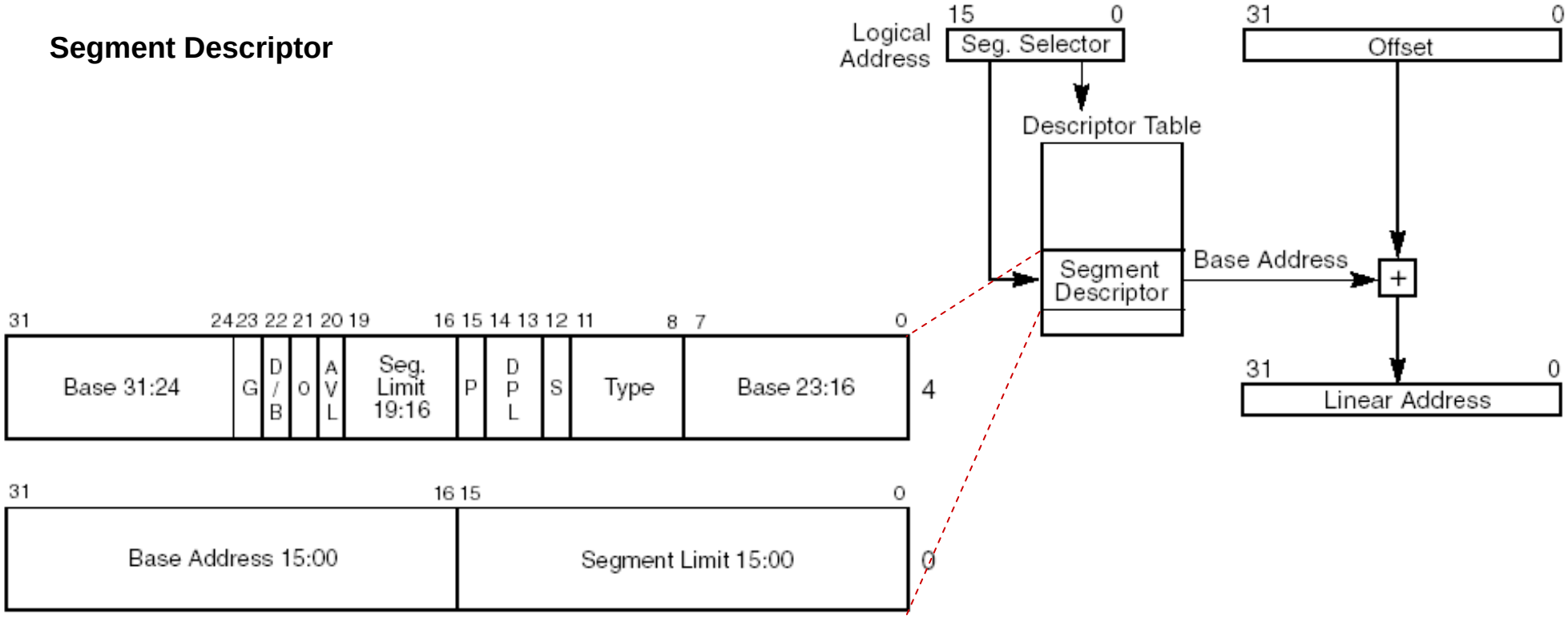
Gestão Memória: Intel Pentium - Segmentação

Conversão de endereço segmentado para linear



Gestão Memória: Intel Pentium - Segmentação

Segment Descriptor



AVL — Available for use by system software
BASE — Segment base address
D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL — Descriptor privilege level
G — Granularity
LIMIT — Segment Limit
P — Segment present
S — Descriptor type (0 = system; 1 = code or data)
TYPE — Segment type

Gestão Memória: Intel Pentium - Segmentação

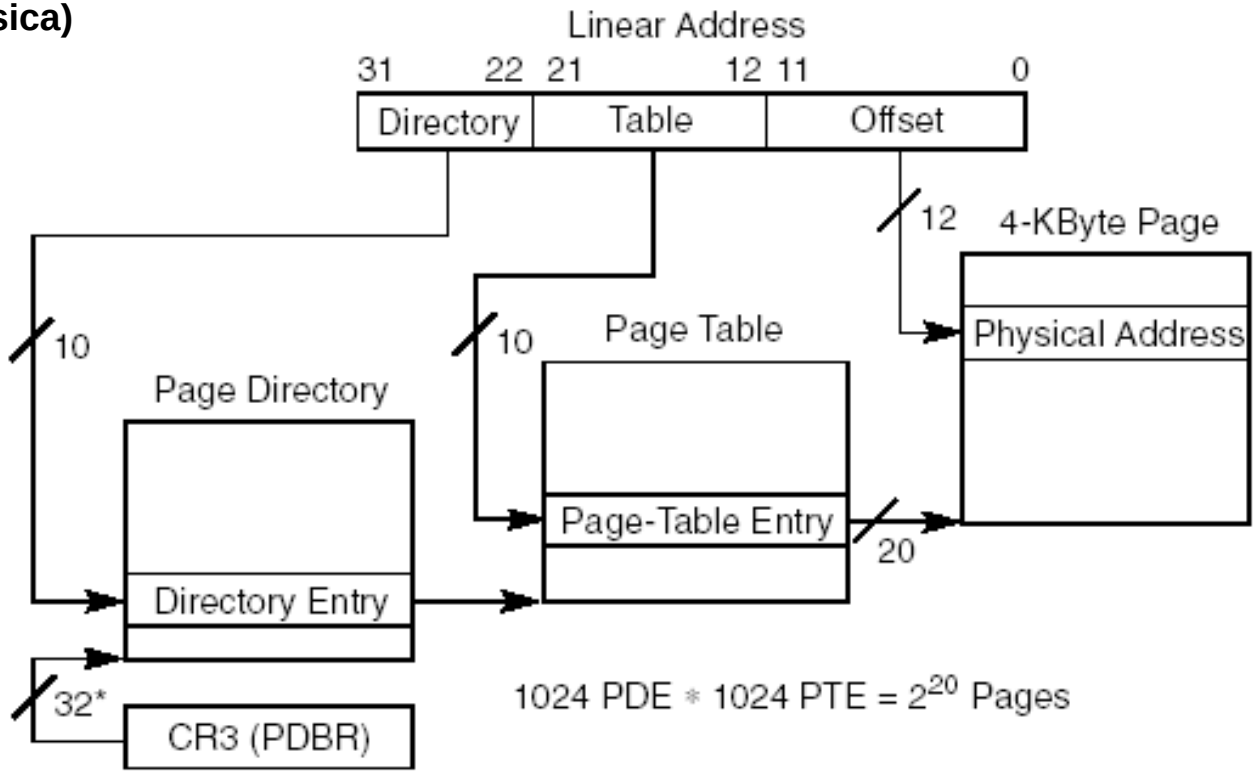
Registos de Segmentos

Visible Part	Hidden Part	
Segment Selector	Base Address, Limit, Access Information	CS
		SS
		DS
		ES
		FS
		GS

Cada registo de segmento está divido em duas partes: a visível e a escondida. A parte escondida é lida da memória e corresponde à informação contida no descritor de segmento. Esta leitura dá-se sempre que se altere o conteúdo de um registo de segmento. Funciona como *cache* do descritor do segmento em uso. Em cada instante um processo pode aceder no máximo a seis segmentos; porém pelo facto de o programa os poder alterar, o limite máximo é de 8K segmentos locais e 8K segmentos globais.

Gestão Memória: Intel Pentium - Paginação

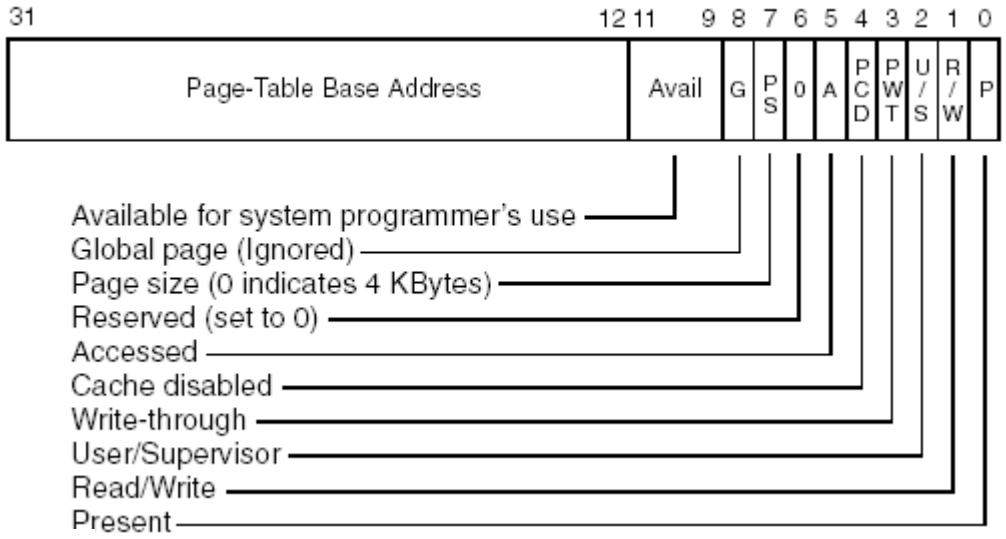
Mapeamento de endereço linear virtual para endereço linear real (memória física)



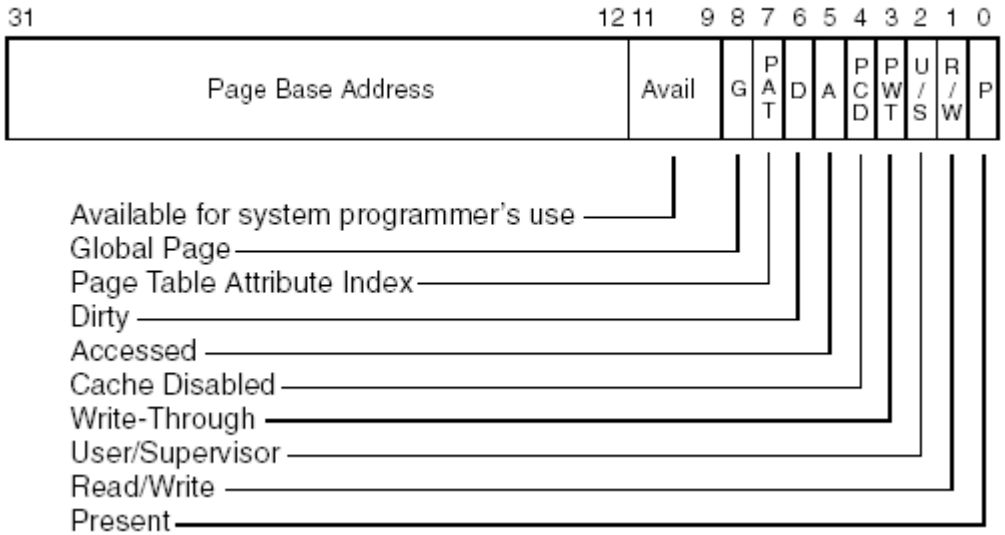
Gestão Memória: Intel Pentium - Paginação

Estruturas de mapeamento

Page-Directory Entry (4-KByte Page Table)



Page-Table Entry (4-KByte Page)



Gestão Memória: Intel Pentium

Processo de endereçamento – caso de segmentação com paginação:

1. O número do segmento é usado para localizar em memória o descritor do segmento.

2. Se a página com o descritor do segmento não se encontrar em memória gera-se um *page fault*. Se estiver em memória a entrada correspondente é localizada. Nessa altura valida-se o tipo de acesso pretendido. Se houver violação gera-se o *trap* correspondente.

3. De seguida, a tabela com as páginas do segmento é examinada. Se a página correspondente não estiver em memória, gera-se um *page fault*. Se estiver, usa-se o número da página para obter da tabela o endereço real da página correspondente.

4. O *offset* do endereço é somado com o endereço base da página real.

5. A operação de leitura ou escrita tem lugar.

Para que este processo seja eficiente, o processador contém uma memória associativa, análoga à que se descreveu anteriormente (TLB). Nas versões mais recentes existem duas TLB uma para endereços de código e outra para endereços de dados.

QUESTÃO: O que acontece ao conteúdo de uma TLB quando se comuta de processo?