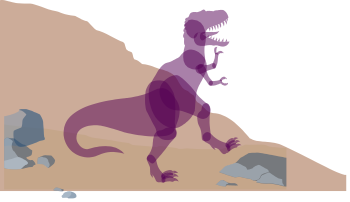
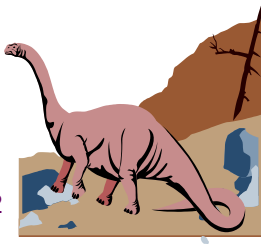
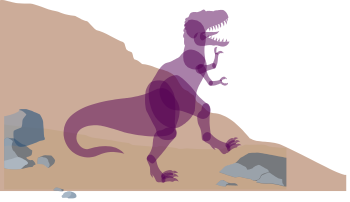


Capítulo 8: Gestão de Memória

SUMÁRIO:

- Conhecimentos de base
- Gestão de memória? Porquê?
- Amarração de instruções e dados à memória
- Endereços lógicos e físicos
- Locação contígua
- Paginação
- Segmentação
- Segmentação com paginação

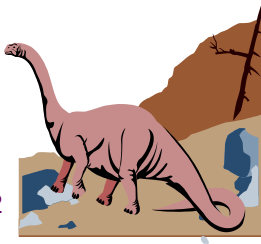




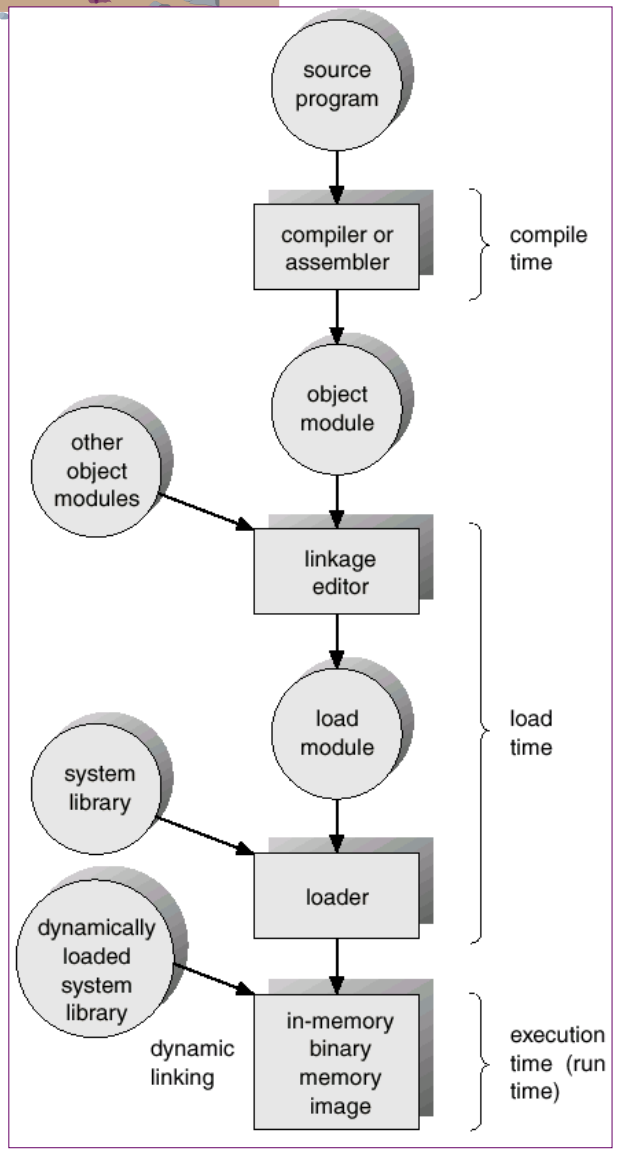
Conhecimentos de base

- Um programa para ser executado precisa de estar em memoria.
  - The stored-program in memory concept
    - A. Turing “Universal Turing Machine “ 1936/37
      - fita com dados e instruções
    - John von Neumann – Von Neumann Architecture 1945
    - Eckert and Mauchly – ENIAC 1945

- Conceito Moderno : Um programa deve ser levado para a memória e colocado dentro do contexto dum processo para que possa ser executado.
- Os programas dum utilizador passam por vários passos antes de serem executados.
  - Compilação.
  - Linkagem Estática.
  - Carregamento e Linkagem Dinâmica.
  - etc



Amarração de instruções e dados à memória

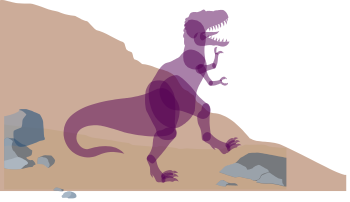


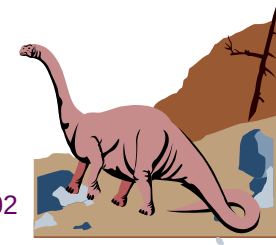
A amarração **(binding)** ou conexão de endereços de instruções e dados a endereços de memória pode acontecer em três estágios diferentes:

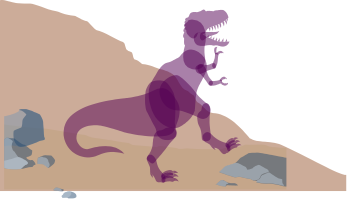
- ***Compile time***: Se a localização em memória dum programa é conhecida *a priori*, então o código absoluto pode ser gerado; se a localização inicial muda, o código deve ser recompilado (ex.: ficheiros .COM do MS-DOS).
- ***Load time***: Se a localização em memória não é conhecida em *compile time*, então torna-se necessário gerar código *relocável*. Neste caso, a amarração de endereços é retardada até à fase de carregamento ⇒ *unificação estática*

  (p.ex. c/ bibliotecas .a / .lib)
- ***Execution time***: Se um processo pode ser transladado durante a sua execução dum segmento de memória para outro, então a amarração é adiada até à altura de execução. Requer hardware especial para suportar transformações de endereços ⇒ *unificação dinâmica* (p.ex. c/ bibliotecas .so /.dll)

Processamento multi-passo dum programa

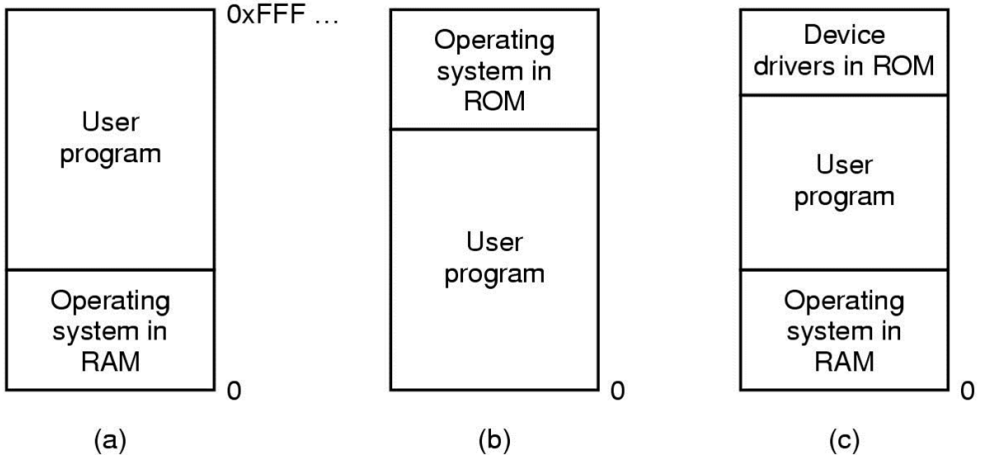






Recordar
Partição de Memória em Sistemas Batch

Sistema Mono-programado

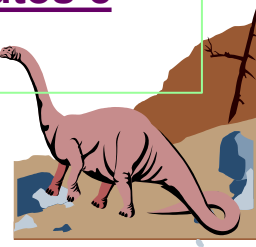


Três possibilidades de organização de memoria para um sistema operativo mono programado. (Tanenbaum, Modern Operating Systems 2008 Prentice-Hall)

Programs com **Endereços absolutos e estáticos.**

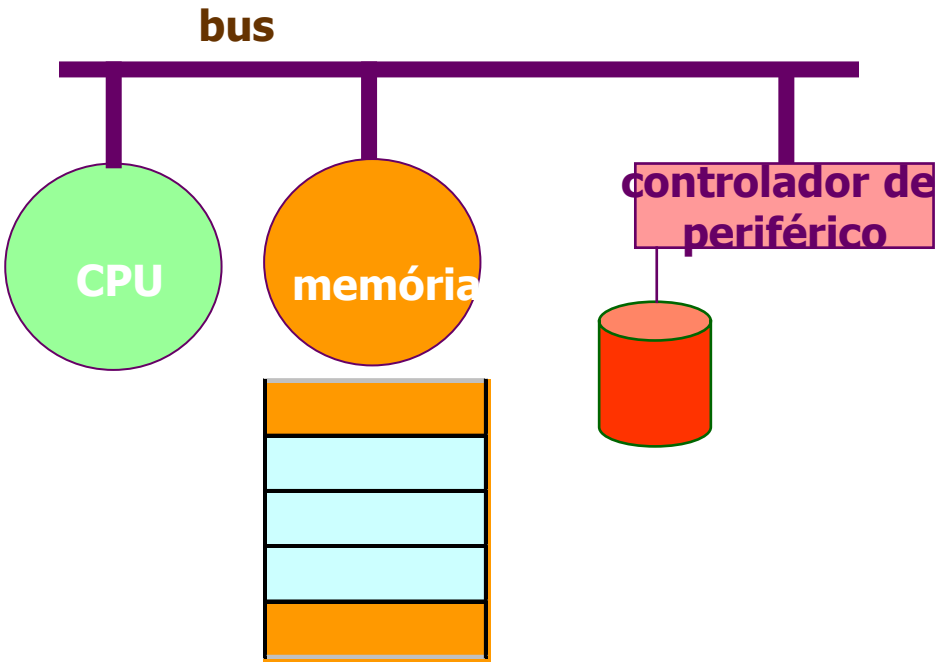
Sistema Multi-Programado

Endereços (dos user programs) não podem ter **Endereços absolutos e estáticos !**



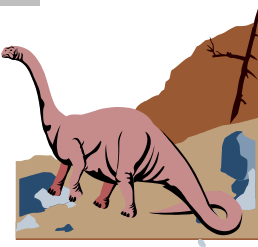
Gestão de memória? PORQUÊ?

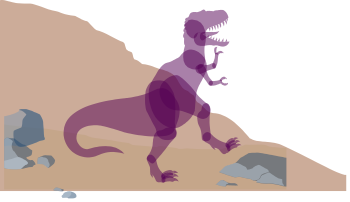
- Um programa reside no disco sob a forma de ficheiro executável.
- Para ser executado, o programa tem de ser colocado em memória e associado a um processo.
- Em função da política de gestão de memória, **o processo poderá transitar entre o disco e a memória durante o seu tempo de execução.**
- À medida que o processo é executado, o CPU acede a instruções e dados residentes em memória.
- Ao terminar, a memória utilizada é libertada.



- **ocupação de memoria**
- **libertação de memoria**
- **Reorganização de memoria**

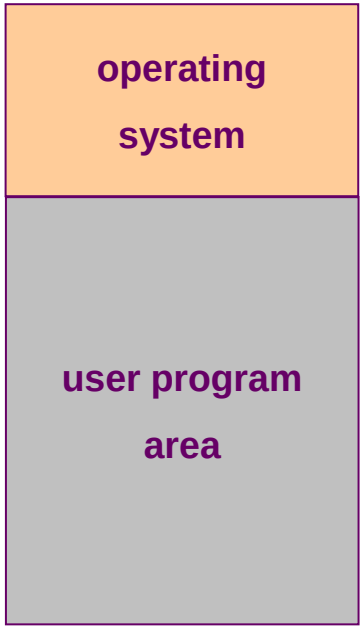
Memória – Organização ??



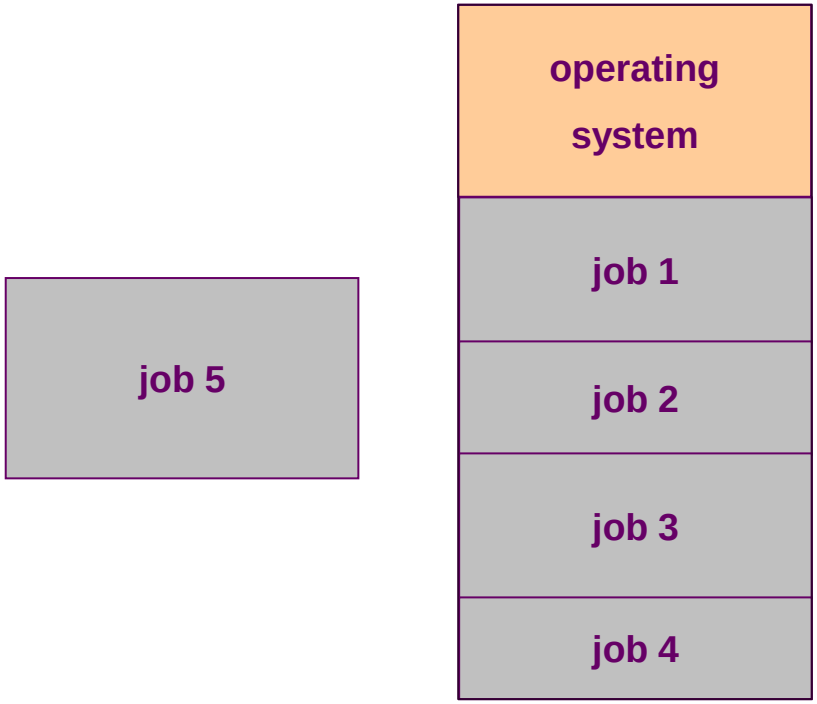


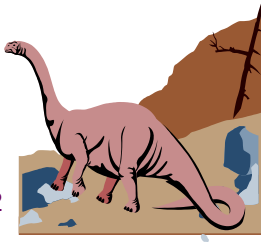
Partição de Memória em Sistemas Batch

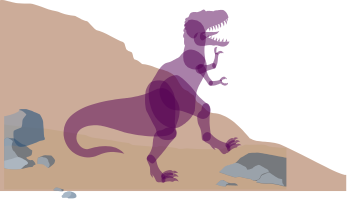
Sistema Mono-programado



Sistema Multi-Programado

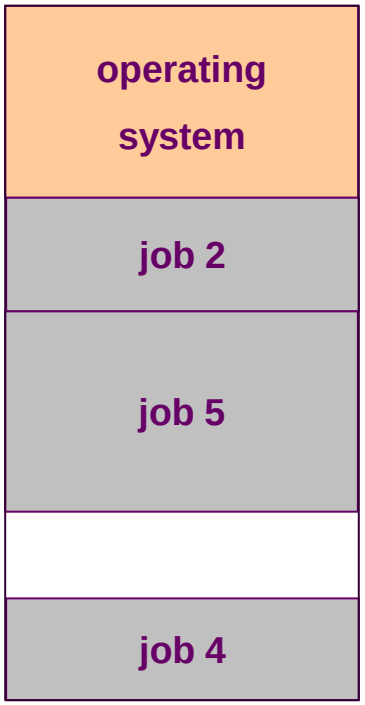


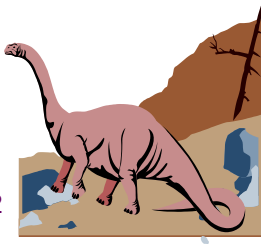


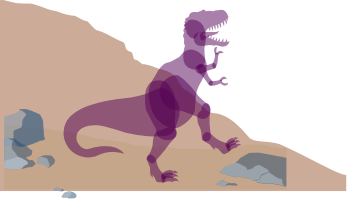


Partição de Memória em Sistemas Batch

Sistema Multi-Programado

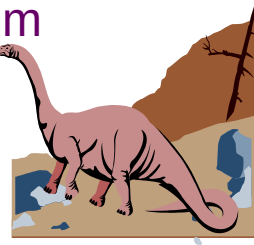


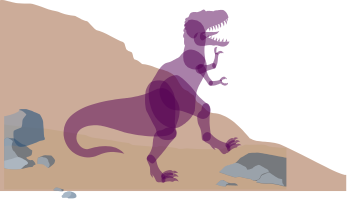




Relocation and Protection

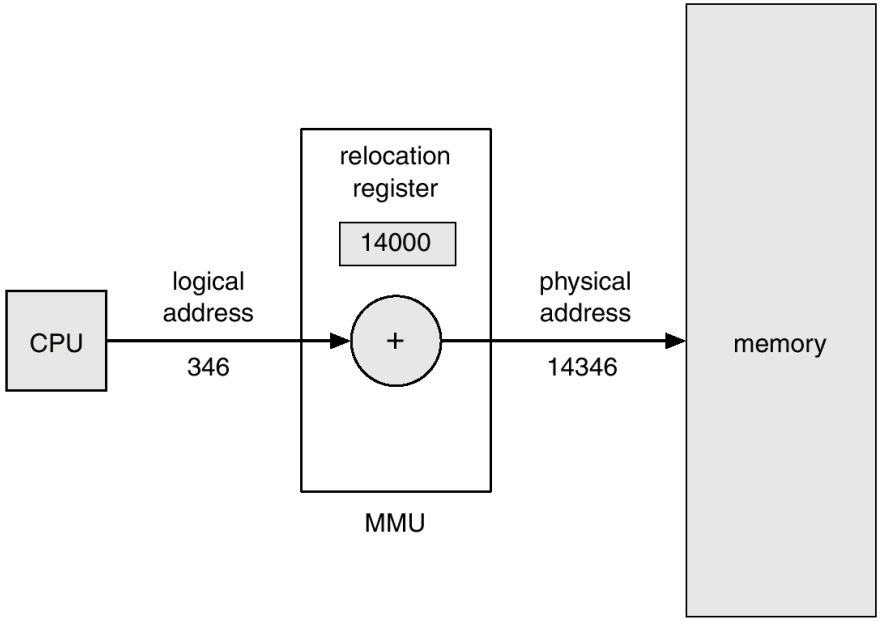
- Dois problems da multiprogramação
  - ***Código Relocável*** : Quando um programa é executado não sabe a-priori qual é a locação em memoria física onde vai ser carregado. Portanto endereços dum programa não podem ser endereços estáticos , absolutos. Terão que ser endereços relativos – relativos ao endereço inicial do programa.
  - ***Proteção*** : Uma vez que se permite dois programas em memoria ao mesmo tempo existe o perigo que um programa escreve para o “”address space”” do outro – obviamente perigoso e para ser evitado.
- Solução :
  - Usar dois registos:
  - **registo base** (contém o end. físico mais baixo da memoria)
  - **registo limite** (contém o end. físico mais elevado da memoria).
- Uma vantagem adicional deste método é de possibilitar de mudar um processo dum parte de memoria para outra.





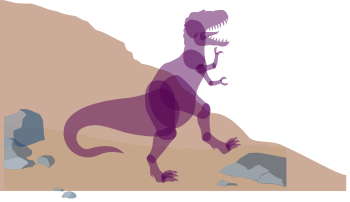
Espaço de endereços lógicos vs. espaço de endereços físicos

- ***Endereço lógico*** – gerado pela CPU; também chamado endereço ***virtual***.
- ***Endereço físico*** – gerado pela MMU (*memory management unit*).
- A amarração de endereços durante a *compilação* ou durante o *carregamento* dum programa implica a **igualdade** dos endereços lógicos e físicos.
- A amarração de endereços na *execução* dum programa e o facto que um processo pode ser deslocado em memória durante a sua execução implica a **desigualdade** dos endereços lógicos e físicos.



MMU (Memory Management Unit)

- Dispositivo de hardware que transforma endereços virtuais em endereços físicos.
- Na MMU, o valor no registo de relocação é adicionado a todo o endereço lógico gerado por um processo do utilizador na altura de ser enviado para a memória.
- O programa do utilizador manipula endereços lógicos; ele nunca vê endereços físicos reais.



Locação contígua de memória
(8.1 Partição Única)

- *A memória principal tem aqui duas partições para:*
  - *1: O sistema operativo.*
  - *2: Os programas dos utilizadores*
- É necessário **proteger**
  - o espaço de endereçamento do sistema operativo de modo a evitar alterações (acidentais ou maliciosas) por parte dos processos dos utilizadores.
  - o espaço de endereçamento dos processos uns dos outros.

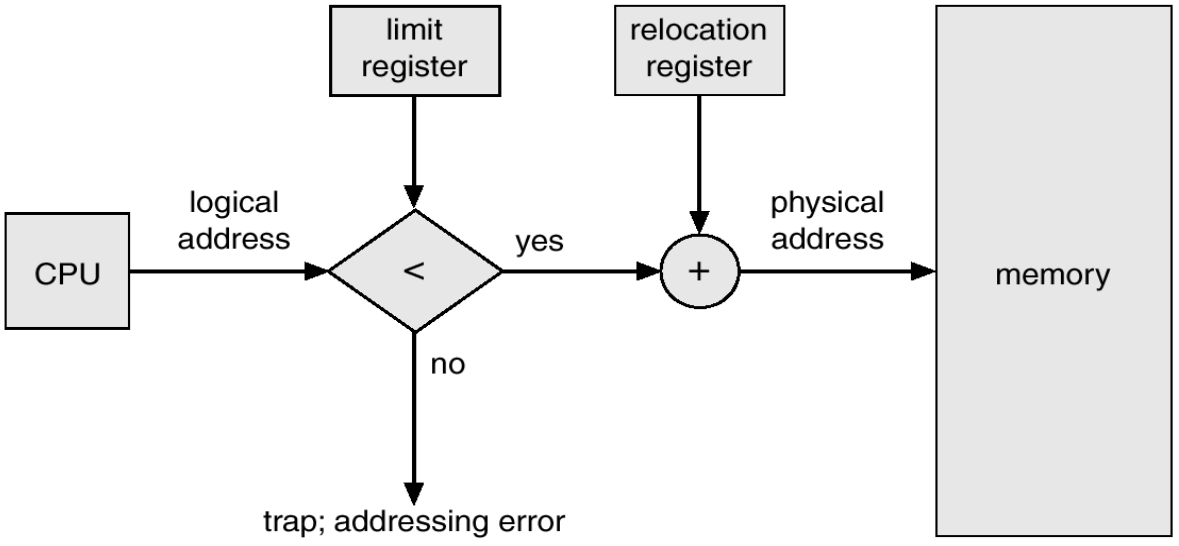
Solução:

registo de recolocação

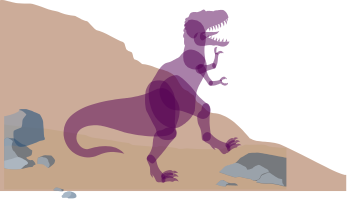
- contém o endereço físico mais baixo que a CPU pode gerar

registo limite

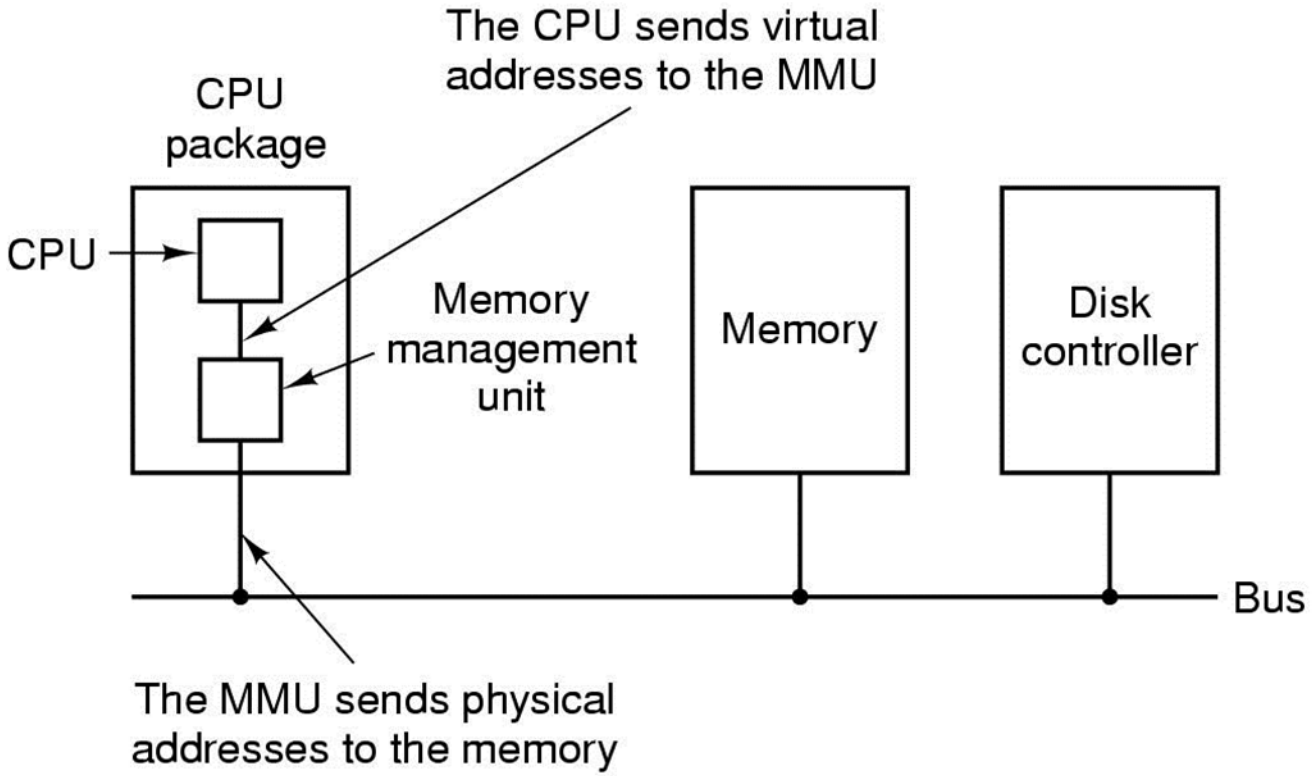
- gama de endereços lógicos usados pelo processo



Hardware Support for Relocation and Limit Registers

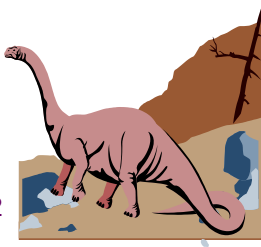


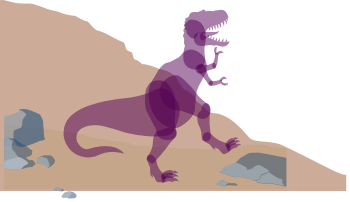
MMU



Arquitectura Fisica do MMU

The position and function of the MMU – shown as being a part of the CPU chip (it commonly is nowadays). Logically it could be a separate chip, was in years gone by. Tannenbaum





Locação contígua de memória

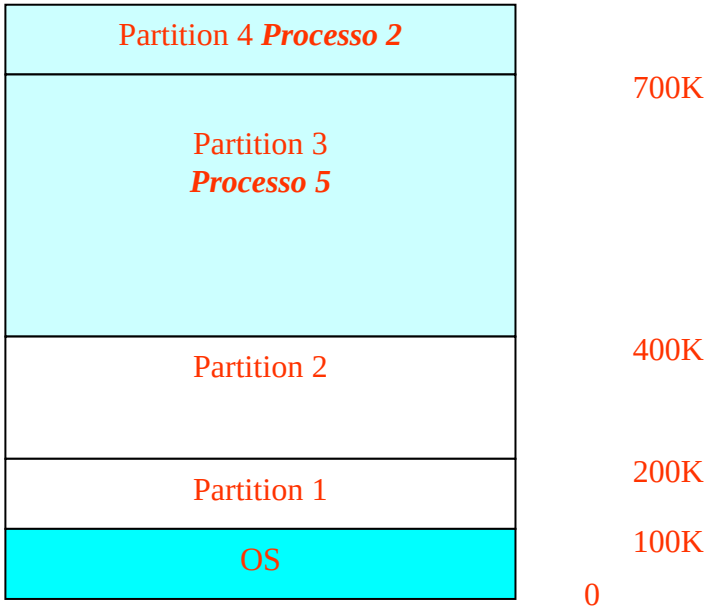
(8.2 1ª solução – partições múltiplas fixas)

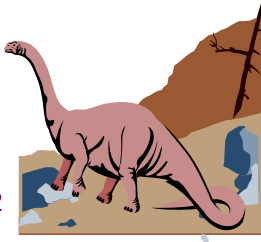
- Subdivisão da memória em partições de tamanho fixo:

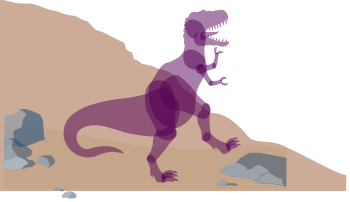
  1 processo ↔ 1 partição.
- Partições podem ter tamanhos diferentes
- Quando uma partição fica livre, é selecionado outro processo que será carregado nesta partição.
- Quando o processo termina a execução, a partição por ele ocupada será libertada.
- Proteção

  registo de recolocação o endereço físico mais baixo da partição.

  registo limite gama de endereços possíveis (tamanho da partição)



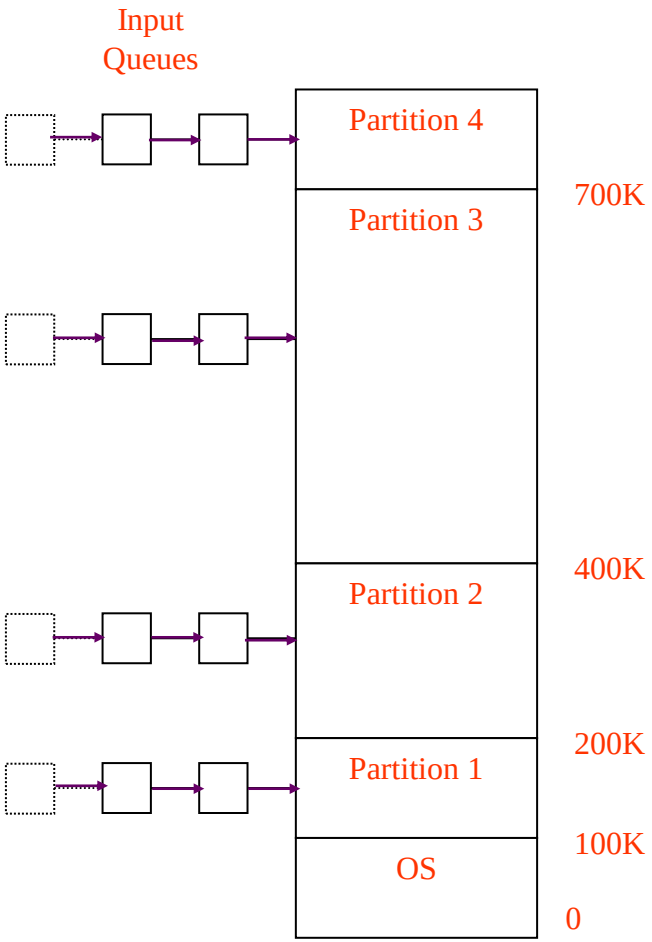




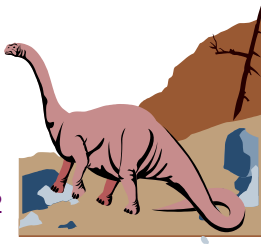
Partições múltiplas fixas

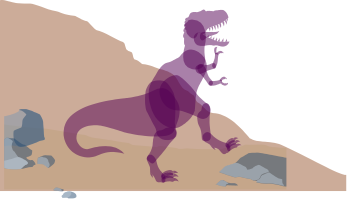
Exemplo com “multi-filas”

- Memoria é dividido em quatro partições
- Quando uma nova tarefa chegar é colocada na fila para a partição mais pequena que possa englobar a tarefa.



- Desvantagens
  1. Pode ser difícil saber o tamanho de partição que um processo irá necessitar.
  2. Um processo posto numa das filas pode ser proibido de executar devido ao facto que haja outros processos nesta fila também a espera que esta fila fica livre .. E outras partições podem ser livres !!

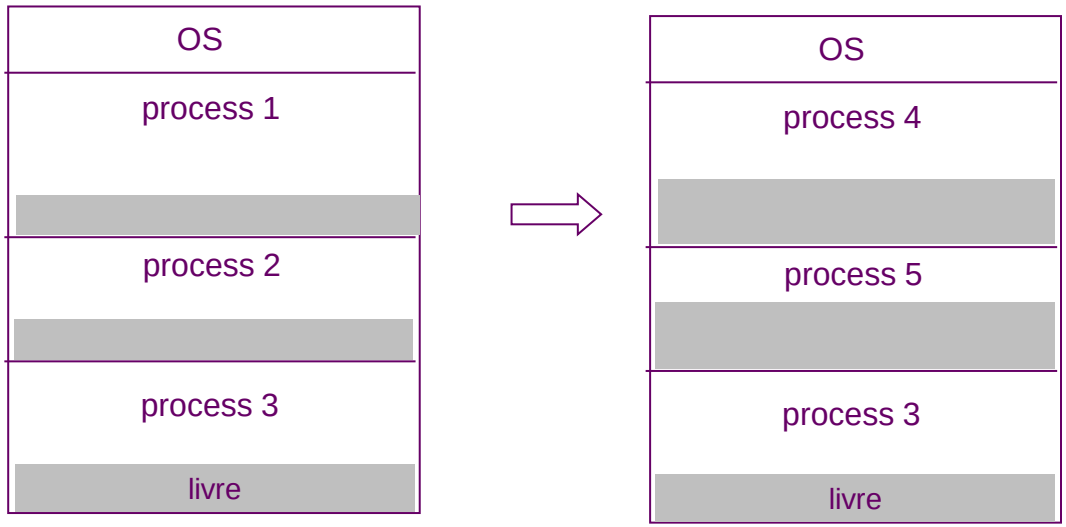




Locação contígua de memória (1ª solução – Partições Múltiplas fixas)

Desvantagens (cont.)

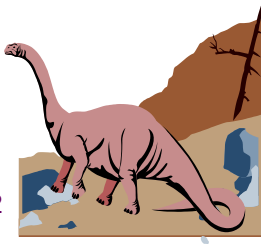
3. Como os tamanhos das partições são fixos q.q espaço não utilizado por um processo é perdido. Chamada "**fragmentação interna**" visto que o programa tem em geral um tamanho menor que a partição

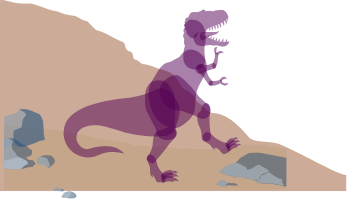


ocupado

livre

Problema: FRAGMENTAÇÃO INTERNA !





Partições múltiplas fixas

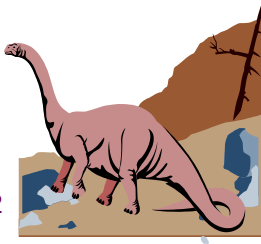
- Alternativa às filas múltiplas: uma fila única
- Implica que haja uma "**Estratégia de Escolha**"

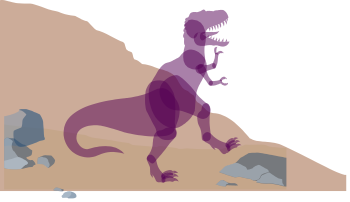
1: Exemplo duma Estratégia: Quando uma partição fica livre pesquisar a fila procurando a primeira tarefa, do conjunto de processos com maior prioridade, que cabia nesta partição

- MAS : Gastamos partições grandes com processos pequeno

2: Pesquisa a fila procurando a maior tarefa que cabia nesta partição

- Assim partições grandes não são desperdiçados com tarefas pequenas.
- MAS : discriminação contra pequenas tarefas → necessidade de um mecanismo de assegurar que pequenas tarefas são ignoradas apenas um numero finito de vezes.

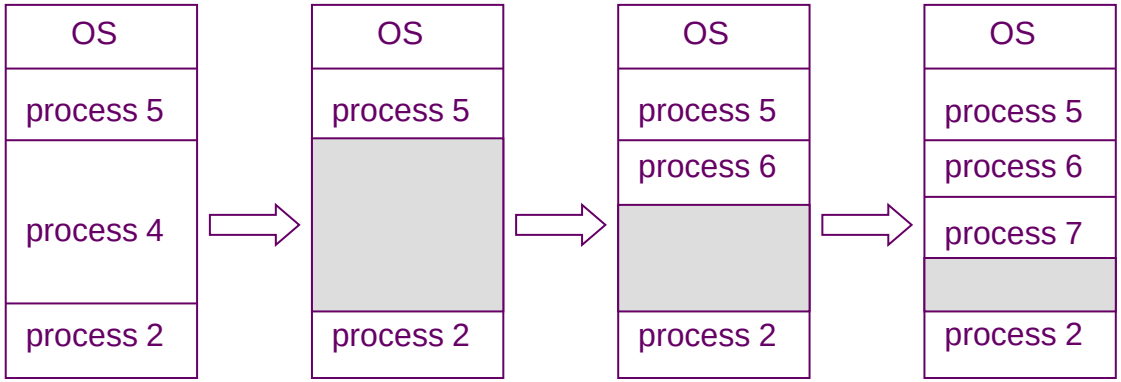




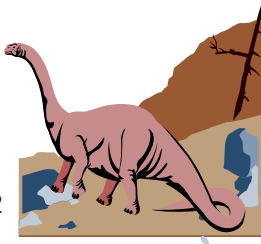
Locação contígua de memória

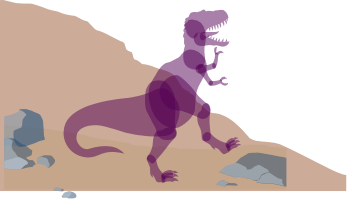
(2ª solução - partições múltiplas variáveis)

- Subdivisão da memória em partições de tamanho variável e **igual** ao tamanho de cada programa:
  1 processo ↔ 1 partição.
- O sistema operativo terá que manter uma tabela de partições livres e partições ocupadas.
- Quando o processo precisa de ser carregado em memória, procura-se na tabela um bloco suficientemente grande para o processo.
- Encontrado o bloco, subdivide-se o bloco em dois blocos: um bloco ocupado pelo programa e um bloco livre.
- Quando um processo termina, o seu bloco é libertado, e **se** é adjacente ao qualquer bloco livre, ocorre a coalescência de ambos.



Problema:
FRAGMENTAÇÃO EXTERNA

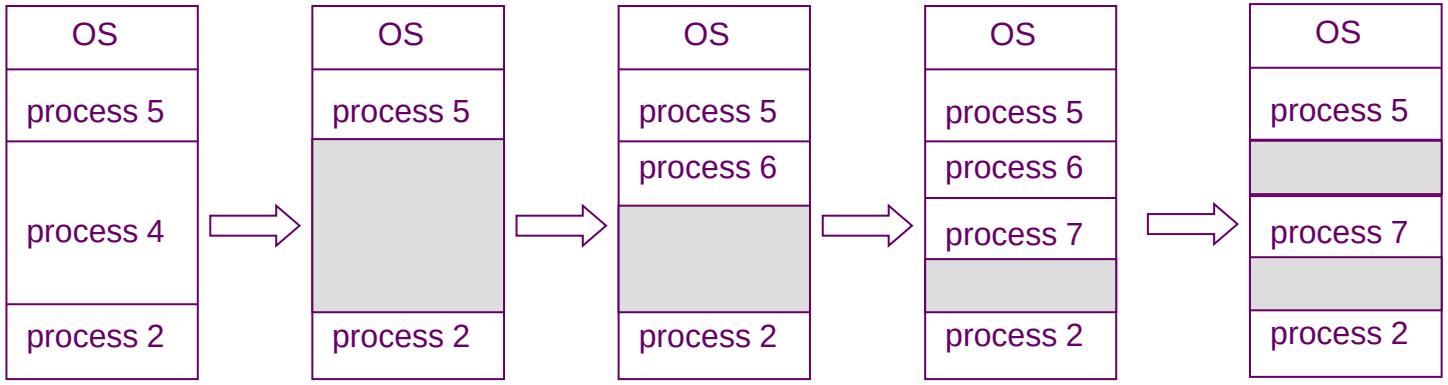


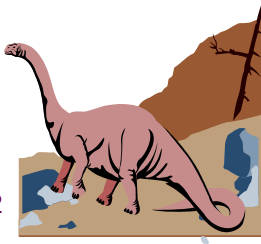


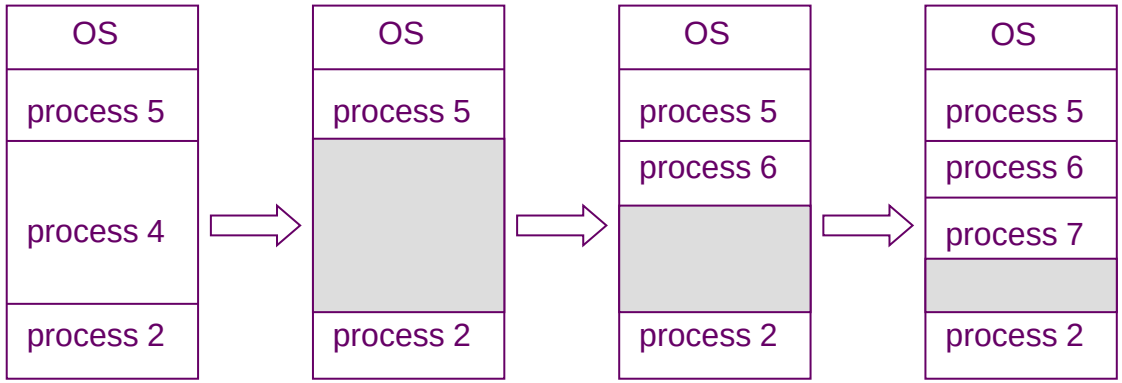
Locação contígua de memória

(2ª solução - partições múltiplas variáveis)

Problema: FRAGMENTAÇÃO EXTERNA !





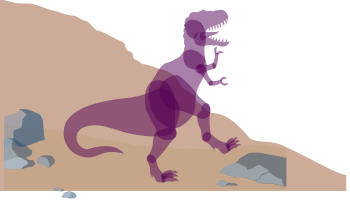


Problema:

Como satisfazer um pedido de alocação de memória para um conjunto de processos {P i} de tamanhos {Size i} a partir duma lista de blocos livres {B k} ?

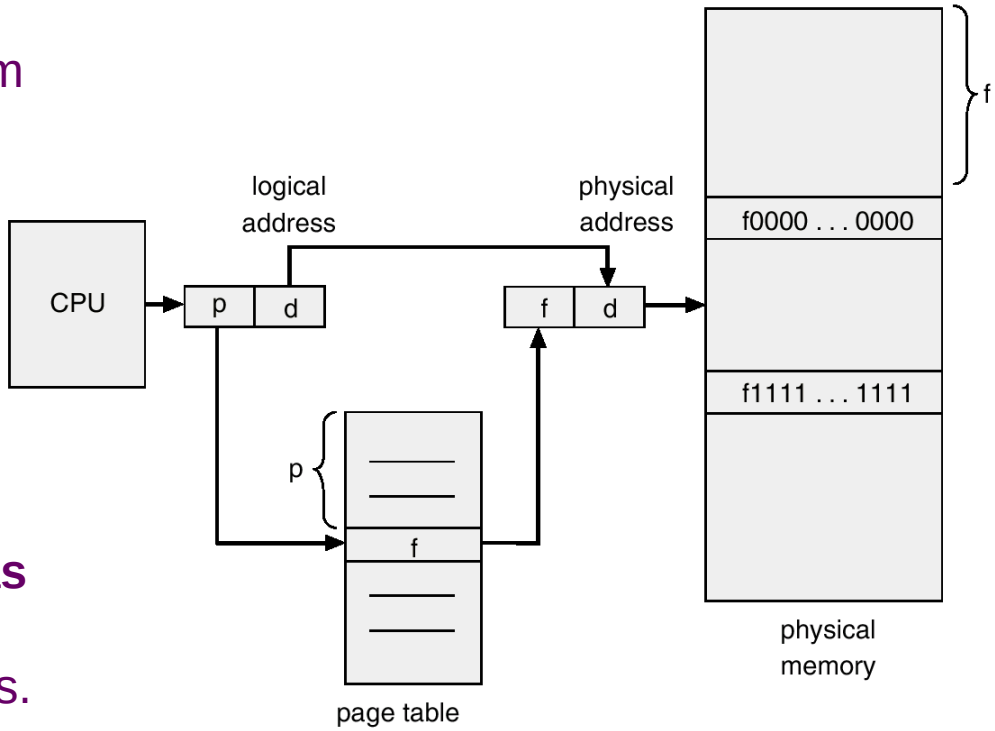
Soluções:

Algoritmo : First-fit (Primeiro Ajuste)
Algoritmo : Best-fit (Melhor Ajuste)
Algoritmo : Worst-fit (Pior Ajuste)



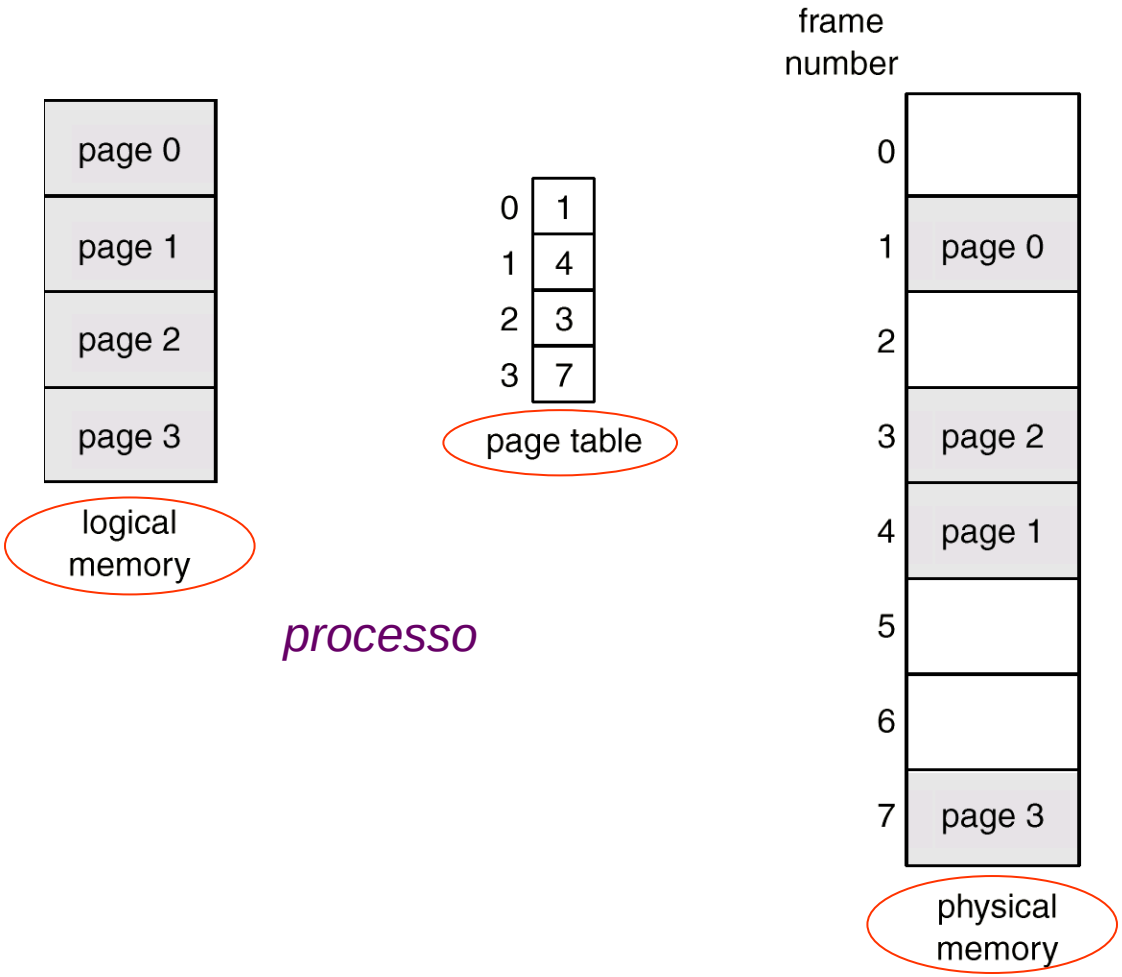
3ª solução Paginação

- O espaço de endereçamento lógico dum processo pode ser não-contíguo; a um processo será atribuída memória física sempre que esta estiver disponível.
- Divide-se a memória física em blocos de tamanho fixo, chamados **molduras** (*frames*), cujo tamanho é uma potência de 2, entre 512 e 8192 bytes.
- Divide-se a memória lógica em blocos do mesmo tamanho, chamados **páginas** (*pages*).
- Há que registar todas as molduras livres.
- Para correr um programa com um tamanho de *n* páginas, é necessário
  - encontrar *n* molduras livres e carregar o programa.
  - activar uma tabela de páginas para converter endereços lógicos em endereços físicos.
- A fragmentação externa é eliminada, mas não a fragmentação interna.



- Um endereço gerado pela CPU tem:
  - *Page number (p)* – usado como índice na tabela de páginas que contém o endereço base de cada página em memória física.
  - *Page offset (d)* – combinado com o endereço de base para definir o endereço físico que é enviado para a unidade de memória.

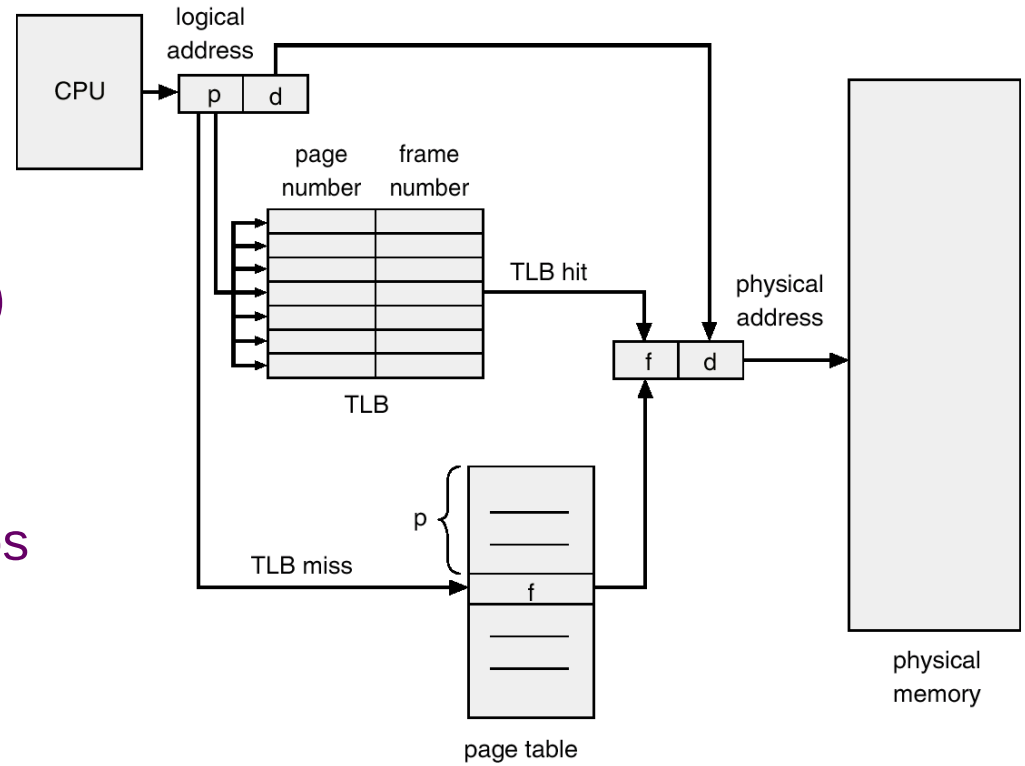
Exemplo: paginação



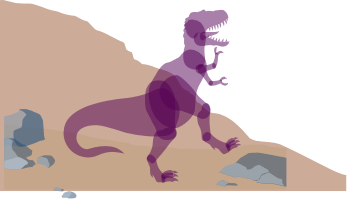
Implementação da tabela de páginas

- A tabela de páginas é guardada na memória principal.

  SO ESTRUTARAS
- *Page-table base register (*PTBR) aponta para a tabela de páginas.
- *Page-table length register* (PRLR) indica o tamanho da tabela de páginas.
- Qualquer acesso a dados/instruções requer 2 acessos à memória: um para a tabela de páginas e outro para os dados/instruções.
- O problema dos dois acessos à memória pode ser resolvido através duma cache de pesquisa rápida, designada por *memória associativa* ou *translation look-aside buffers (TLBs)*



- Tradução de endereços ($A^{vas} \rightarrow A^{fisico}$)
  - Se A^{vas} está no registo associativo, obtém número de frame A^{fisico}.
  - Senão, obtém número de frame a partir da tabela de páginas em memória.



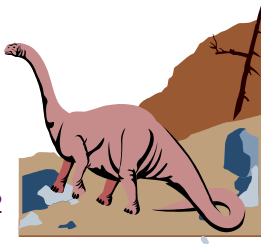
Effective Access Time

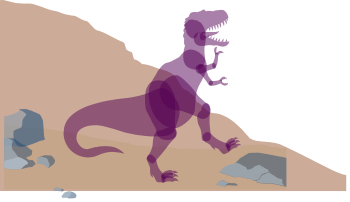
- Assume memory cycle time is 1 time unit
- Assume Associative Lookup is ε time unit
- Hit ratio α - percentage of times that a page number is found in the associative registers; (dependerá do tamanho do TLB, algoritmos de cache etc)
  - Hit ratio = α ($0 \leq \alpha \leq 1$)
- **Effective Access Time** (EAT)

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

- Itanium2 (1.66 GHz processor)
  - L1-I (16 kbytes), 1 cycle
  - L1-D (16 kbytes), 1 cycle
  - L2 (256 KB), 5, 7 or 9+ cycles
  - L3 (3 MB, 4 MB, 6 MB or 9 MB), 12+ cycles

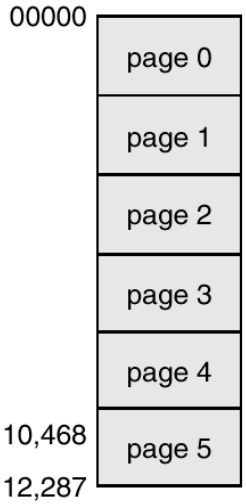
- http://www.behardware.com/../desktop-intel-core-duo.html

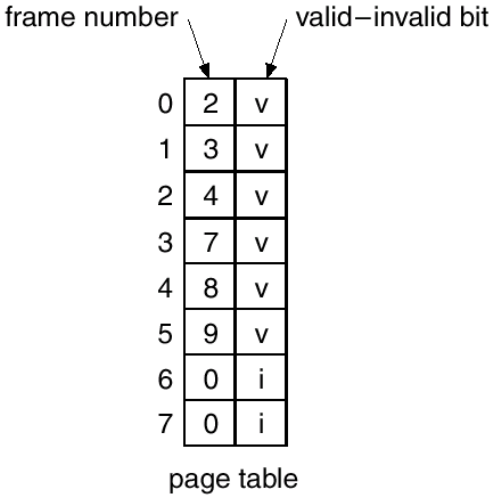




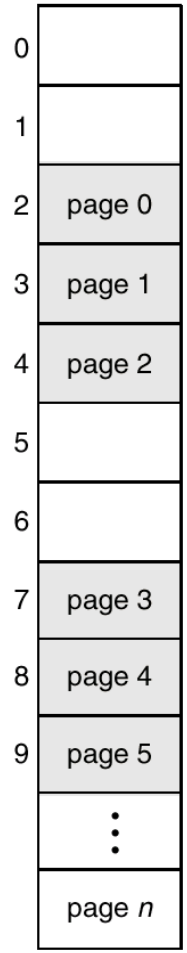
Protecção de memória

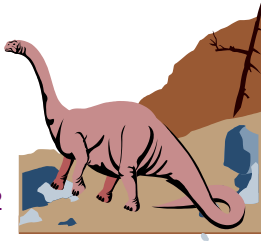
- A proteção de memória é feita através da associação dum *bit de proteção* com cada moldura (frame), na tabela de páginas.
- *O bit valid-invalid* está associado a cada verbete na tabela de páginas :
  - "valid" indica que a página associada está no espaço de endereçamento lógico do processo, sendo por isso uma página legal.
  - "invalid" indica que a página não está no espaço de endereçamento lógico do processo.





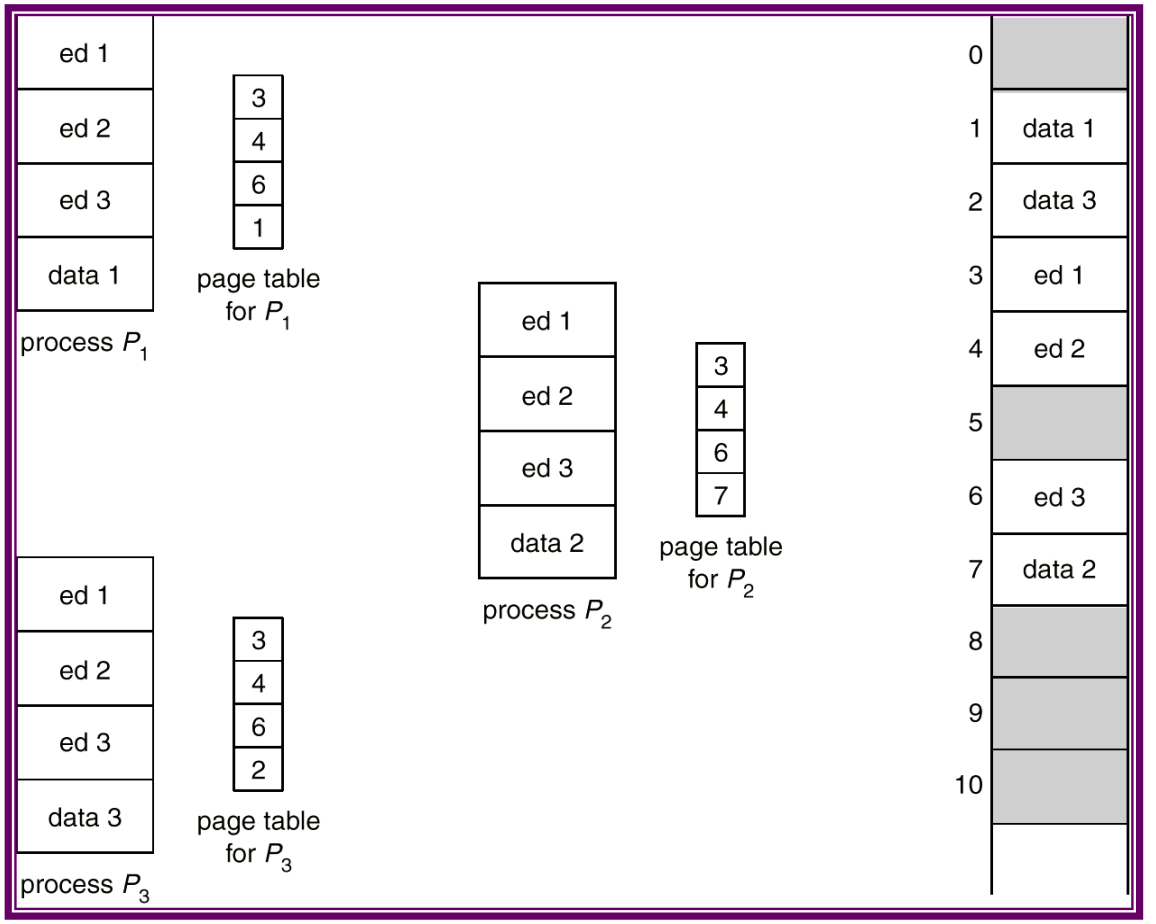
Mem. Fisica

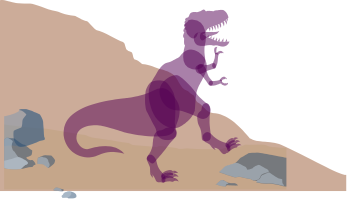




Multiprogramação do Mesmo Programa Usando Páginas partilhadas

- Código partilhado
  - Uma cópia de código **reentrante (** *read-only* **)** partilhado por vários processos (i.e. editores de texto, compiladores, sistemas de janelas).
  - Código partilhado tem de aparecer na mesma localização do espaço de endereçamento lógico de todos os processos partilhantes.
- Código e dados privados
  - Cada processo guarda uma cópia separada de registadores e dados associado a execução do código (Program Counter etc.) e dados próprios associado ao código partilhado.





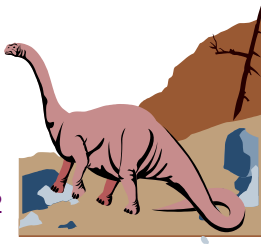
Implementação das Tabelas

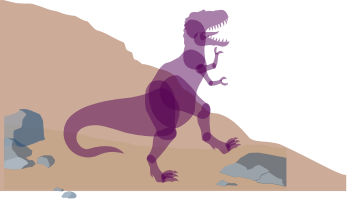
Problema – tabelas da paginas são estruturas per processo-Podem ser grandes !

Exemplo 1GB Ram - Computador32 bit

Implementações Possíveis

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

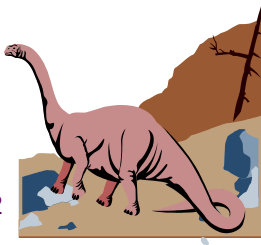


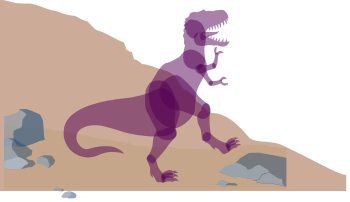


Características da arquitectura de paginação

- ***Relocação***:
  - dinâmica
  - através da tabela de páginas
- ***Partilha***:
  - páginas partilhadas (só código re-entrante)
  - o mesmo número de página
- ***Locação***.
  - first fit/best fit
  - fragmentação interna
- ***Protecção***.

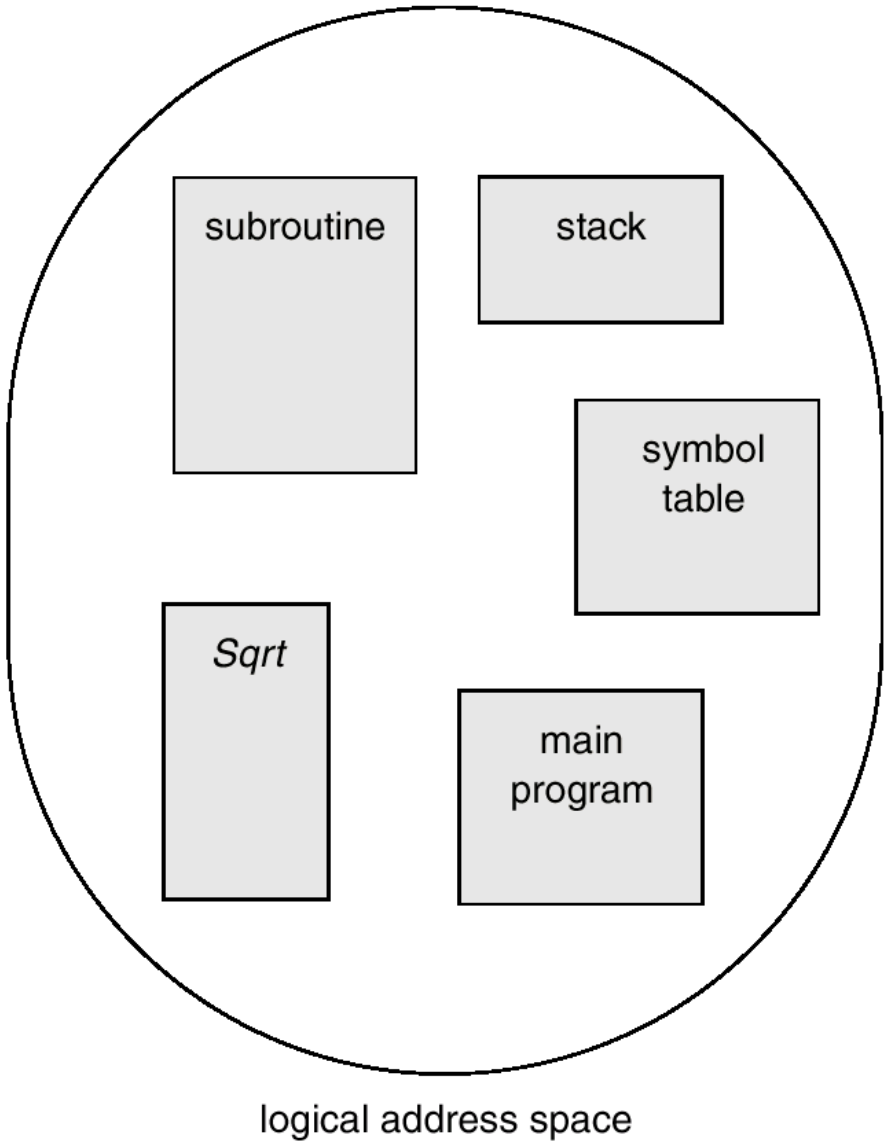
  Cada verbete na tabela de páginas tem associado:
  - *validation bit* = i/v $\Rightarrow$ página invalida/valida
  - Pode ser extendida .. Read Only, Read/Write etc

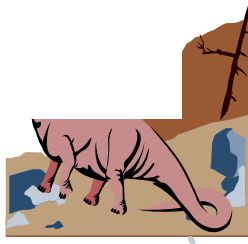




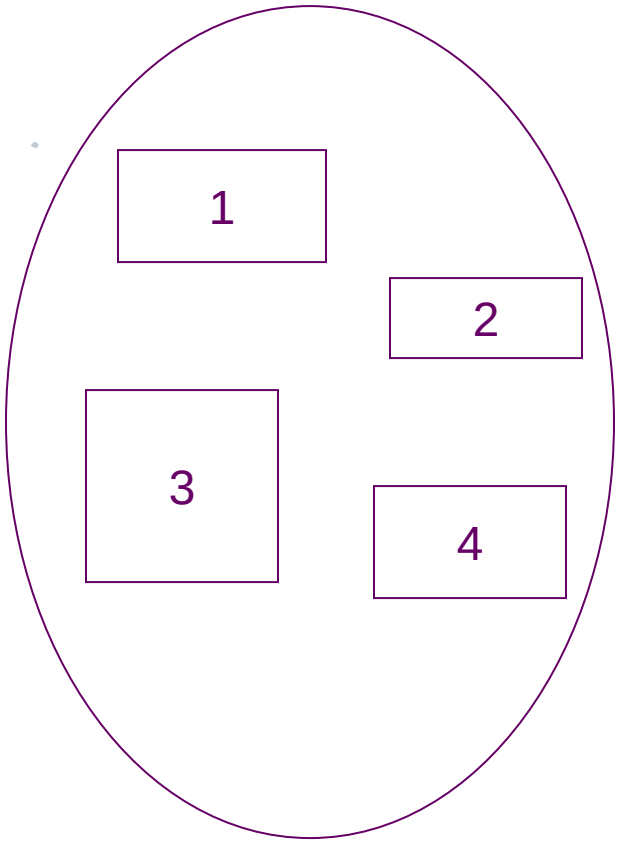
Segmentação

- Esquema de gestão de memória que reflecte a visão que o utilizador tem da memória.
- Um programa é uma colecção de segmentos. Um segmento é uma unidade lógica referente a um(a):
  - programa principal,
  - procedimento,
  - função,
  - método,
  - objecto,
  - variáveis locais, variáveis globais,
  - bloco comum,
  - pilha,
  - tabela de símbolos, arrays

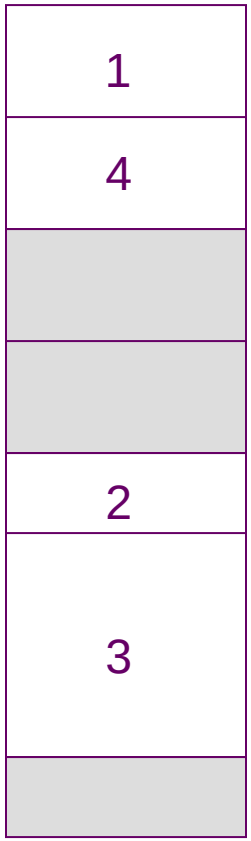




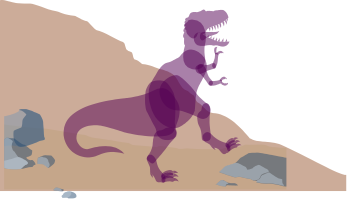
Visão lógica da segmentação



user space



physical memory space

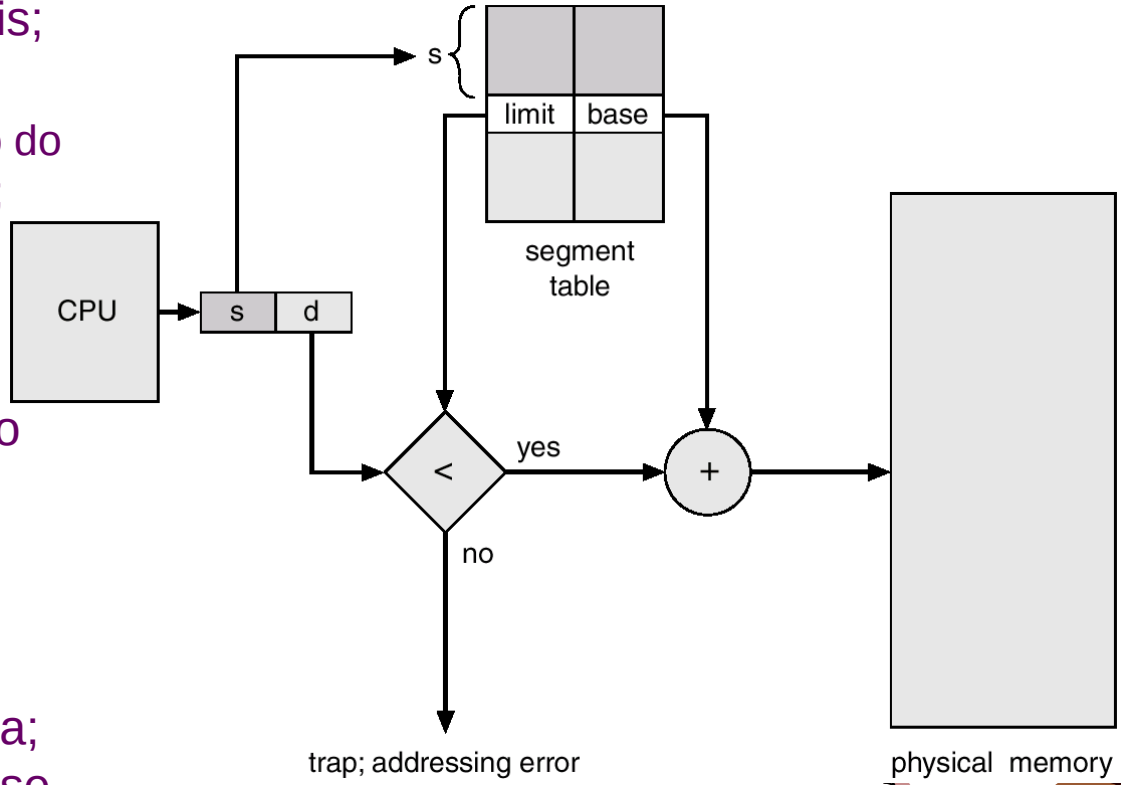


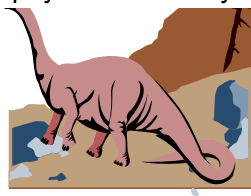
Arquitectura da segmentação

- Endereço lógico consiste num par:

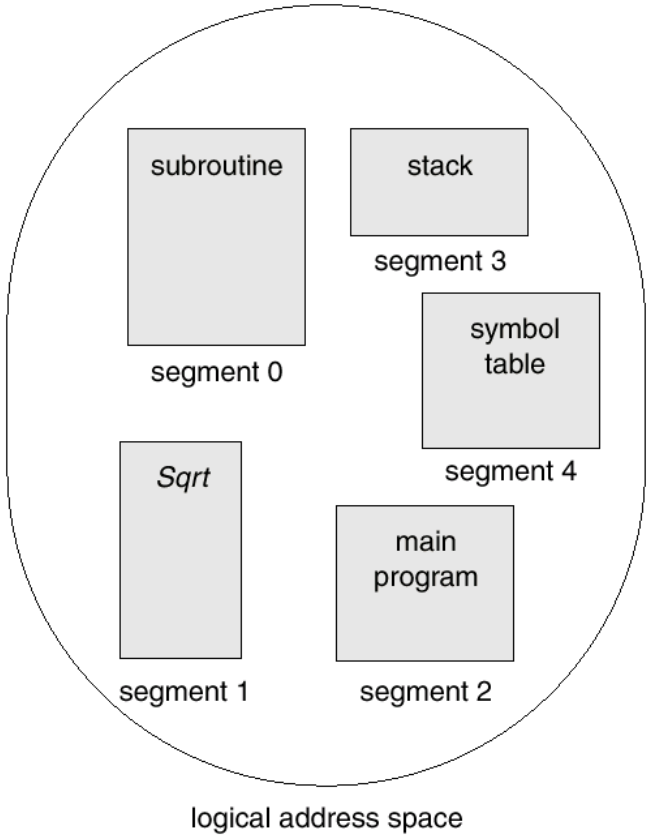
  <***segment-number***, ***offset***>

- *Segment table* – traduz endereços lógicos bidimensionais em endereços físicos unidimensionais; cada verbete da tabela tem:
  - *base* – contém o endereço físico do início do segmento em memória;
  - *limit* – especifica o comprimento do segmento.
- *Segment-table base register (STBR)* aponta para a localização da tabela de segmentos em memória.
- *Segment-table length register (STLR)* indica o número de segmentos usados pelo programa; o número *s* do segmento é legal se *s* < STLR.



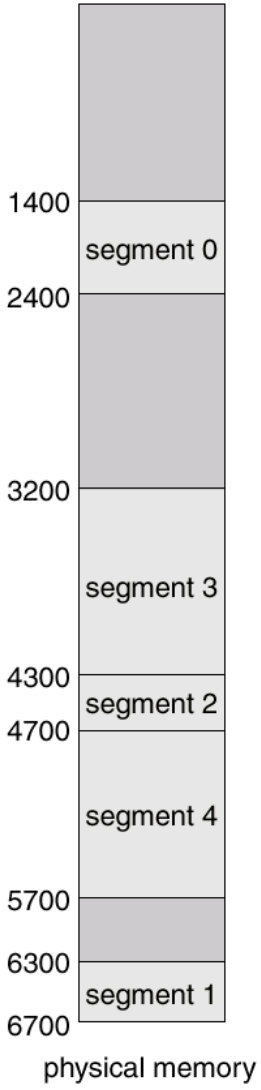


Exemplo: segmentação

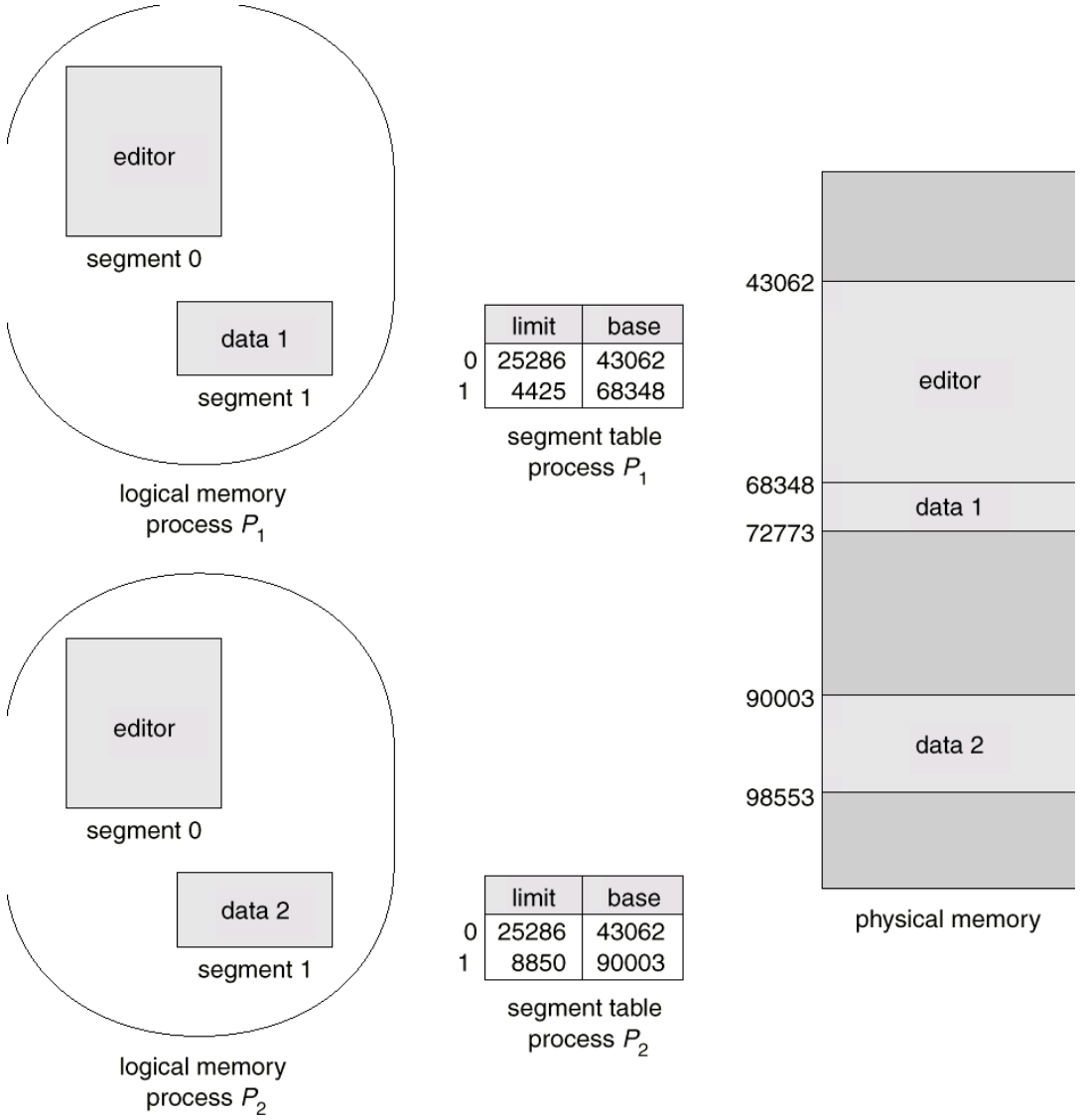


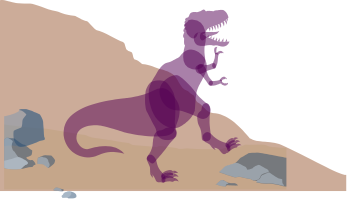
	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



Partilha de segmentos



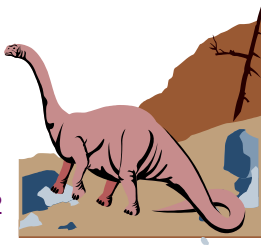


Características da arquitectura de segmentação

- ***Relocação***:
  - dinâmica
  - através da tabela de segmentos
- ***Partilha***:
  - segmentos partilhados
  - o mesmo número de segmento
- ***Locação***.
  - first fit/best fit
  - fragmentação externa
- ***Protecção***.

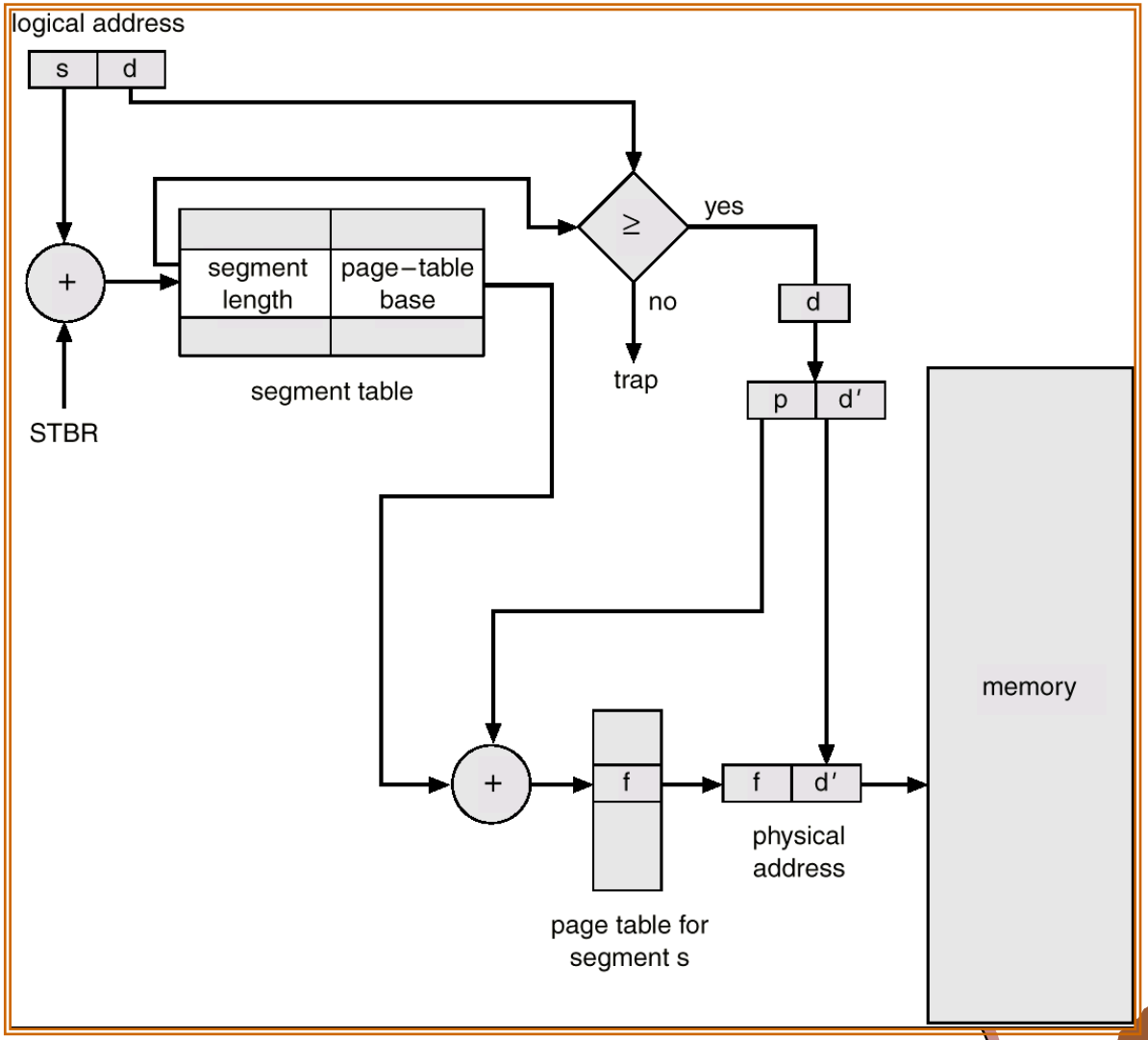
  Cada verbete na tabela de segmentos tem associados:
  - *validation bit* = 0 $\Rightarrow$ segmento ilegal
  - privilégios read/write/execute

Fim de Capítulo



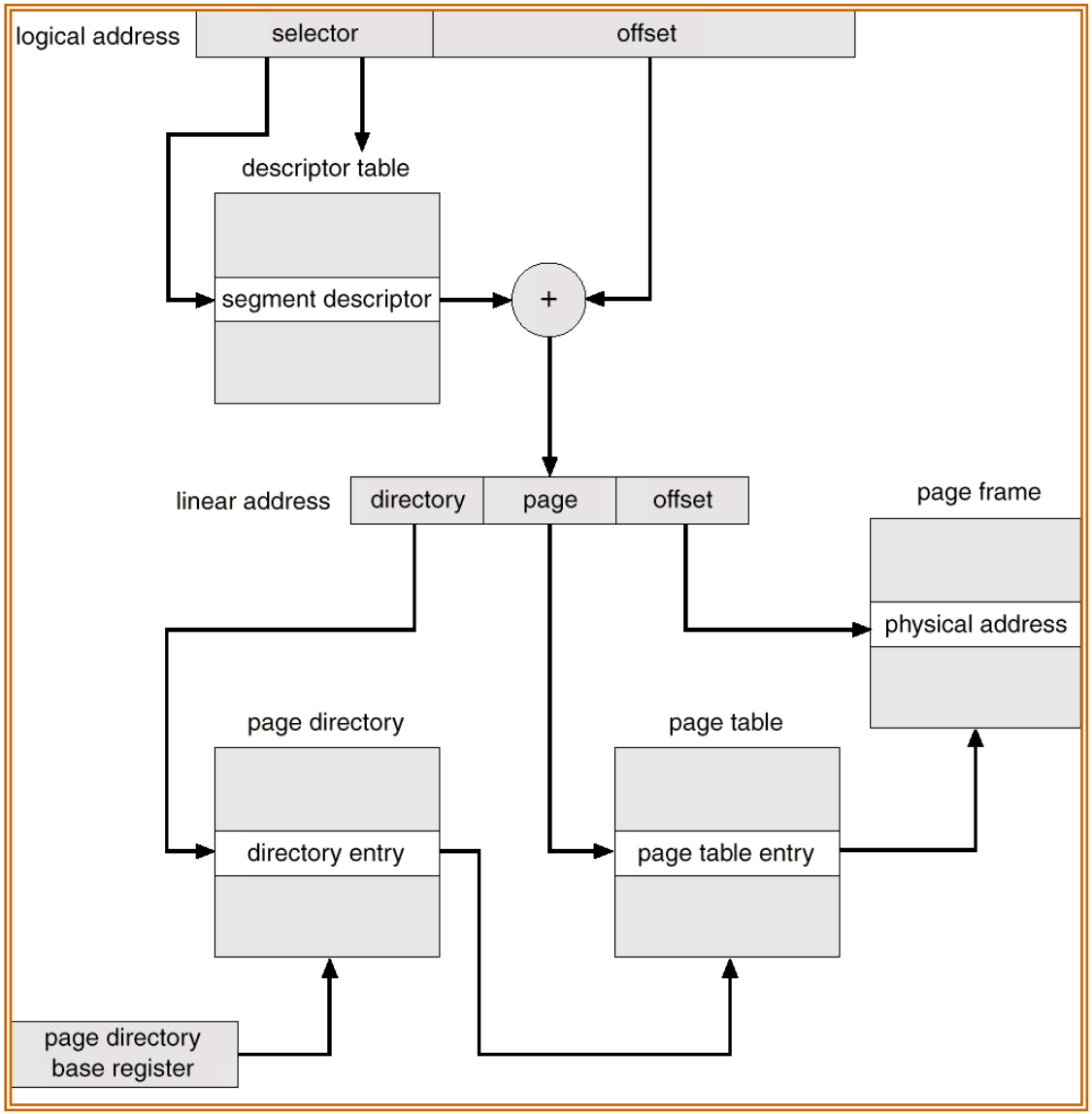
Segmentation with Paging – MULTICS

- MULTICS resolveu problemas de fragmentação externa e tempos de pesquisa longos usando um sistema de segmentos com paginação
- Diferença da segmentação pura – Entrada na tabela de segmentos não é o endereço dum segmento mas endereço duma tabela de paginas para este segmento



Tradução dum endereço

Intel 30386 Address Translation



o Intel 386 utilize um sistema de gestão de memória de segmentação com paginação com uma esquema de dois níveis para a paginação.

Linux no Intel 80x86

- Utilize um numero mínimo de segmentos para simplificar a gestão de memória e melhorar a portabilidade
- Utilize apenas 6 segmentos:
  - Kernel Código
  - Kernel dados
  - User Código (partilhado por todos os processos de utilizadores)
  - User dados (partilhado por todos os processos de utilizadores)
  - Estado da tarefa/processo (per-process hardware context)
  - GDT/LDT
- Dois níveis de proteção:
  - Kernel/Protected mode
  - User/Real mode