INE5412 Sistemas Operacionais I

L. F. Friedrich

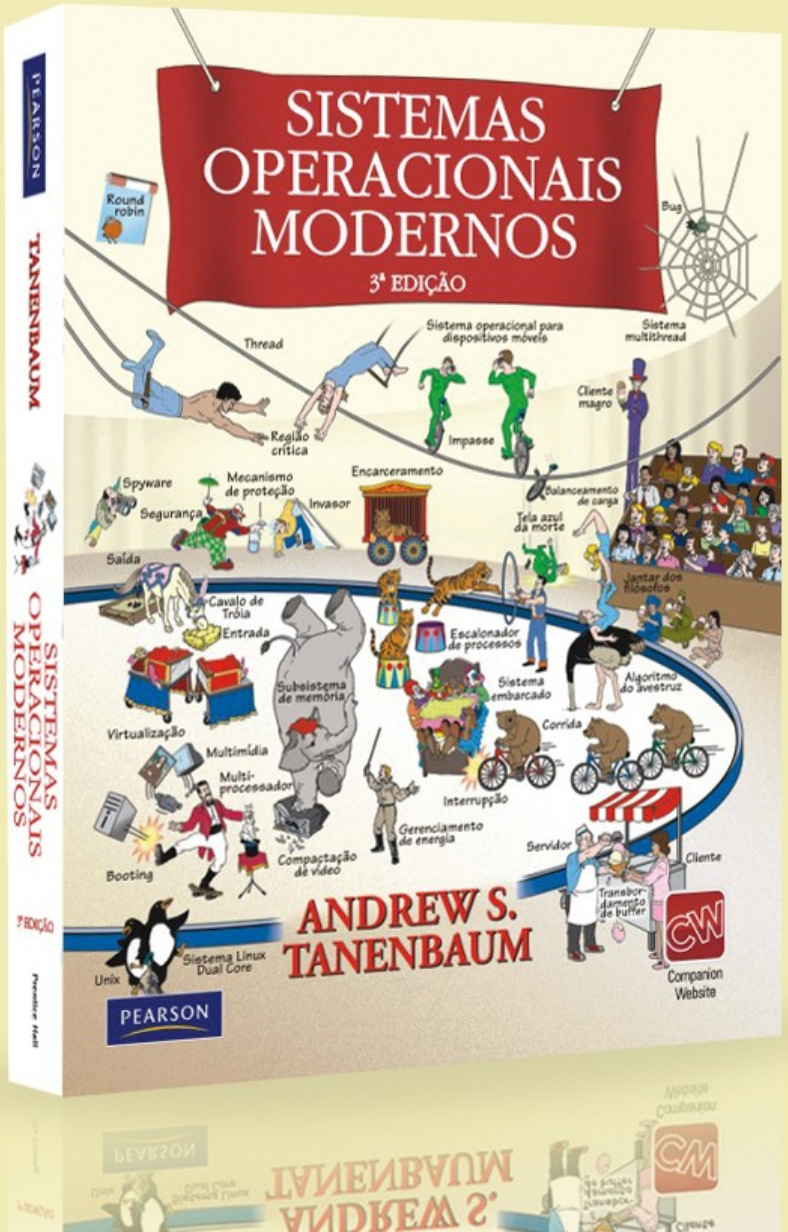
Capítulo 3 - Parte 1

Gerência de Memoria

Sistemas operacionais modernos
Terceira edição

ANDREW S. TANENBAUM

Capítulo 3
Gerenciamento de memória



Gerenciamento de Memória

- Memória principal (RAM) é um recurso importante que deve ser gerenciado com muito cuidado.
- Lei de Parkinson: “programas tendem a se expandir a fim de ocupar toda a memória disponível”
- Soluções: memória infinitamente grande, rápida e não volátil (também a baixo custo) ou a utilização do conceito de **hierarquia de memórias**
- **Função do sistema operacional é abstrair a hierarquia em um modelo útil e gerenciar a abstração.**
- **Abstração mais simples?**

Kernel conhece…

- Quantos processos estão no sistema.
- Quanto espaço cada processo precisa.
- Quanta memória tem no sistema.
- É muito comum…

espaço necessário para processos >> Memória disponível no sistema

- Assim, o kernel deve implementar formas de garantir que:
  - Cada processo no sistema deve ter memória suficiente para rodar.
  - Novo processo pode ter memória suficiente para rodar.

Sem abstração de memória

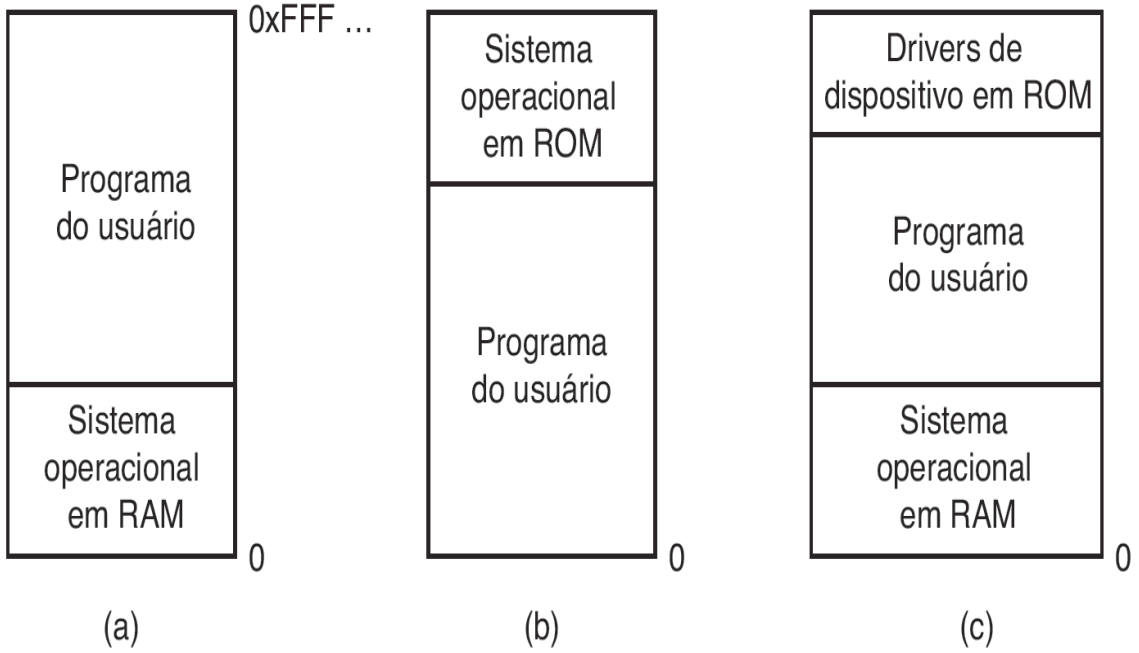
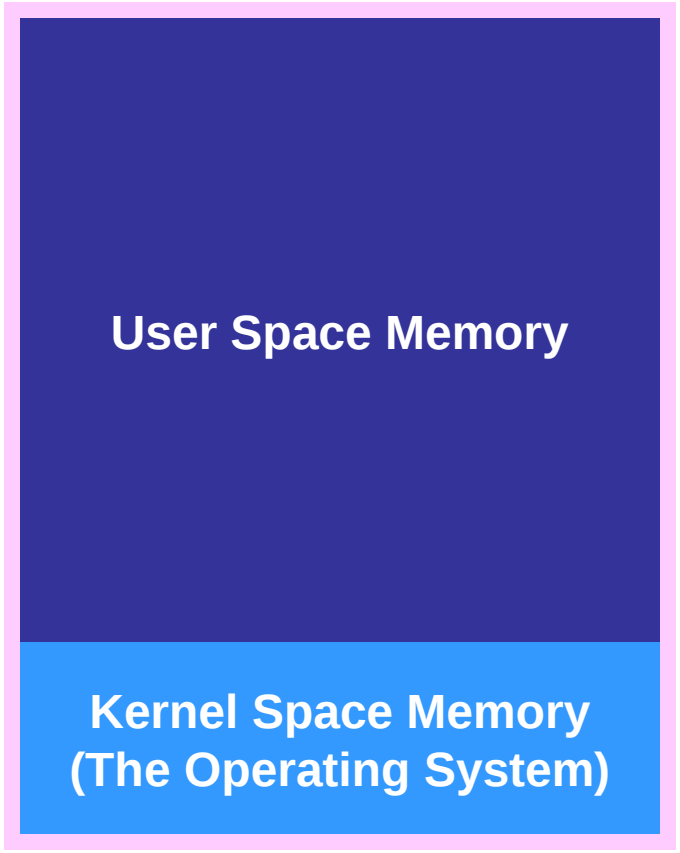


Figura 3.1 Três modos simples de organizar a memória com um sistema operacional e um processo de usuário. Também existem outras possibilidades.

a) grande porte/mini b) sistemas embarcados c) computadores pessoais

Layout Memória Física

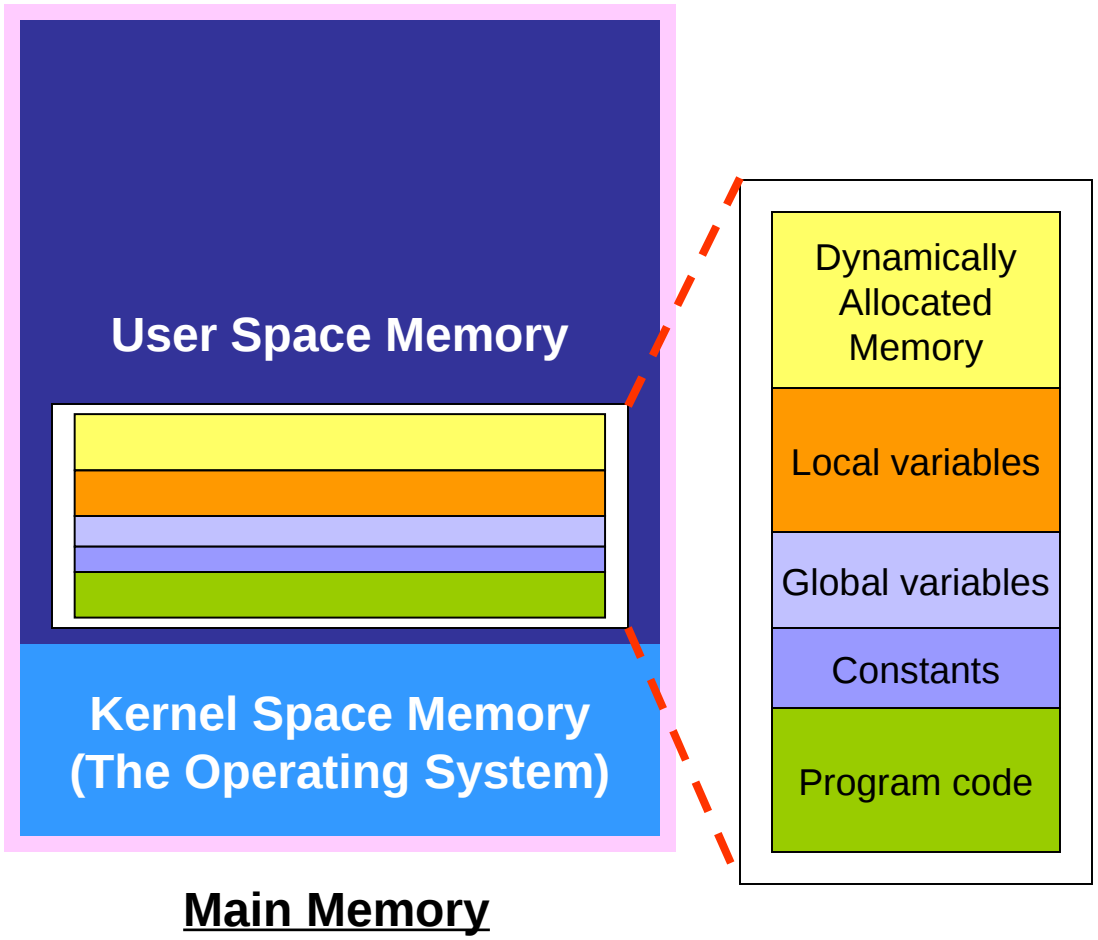


Main Memory

Espaço de usuário armazena os Programas de usuário executando, ex., processos.

Espaço de Kernel armazena o kernel em execução e seus dados.

Layout Memória Física



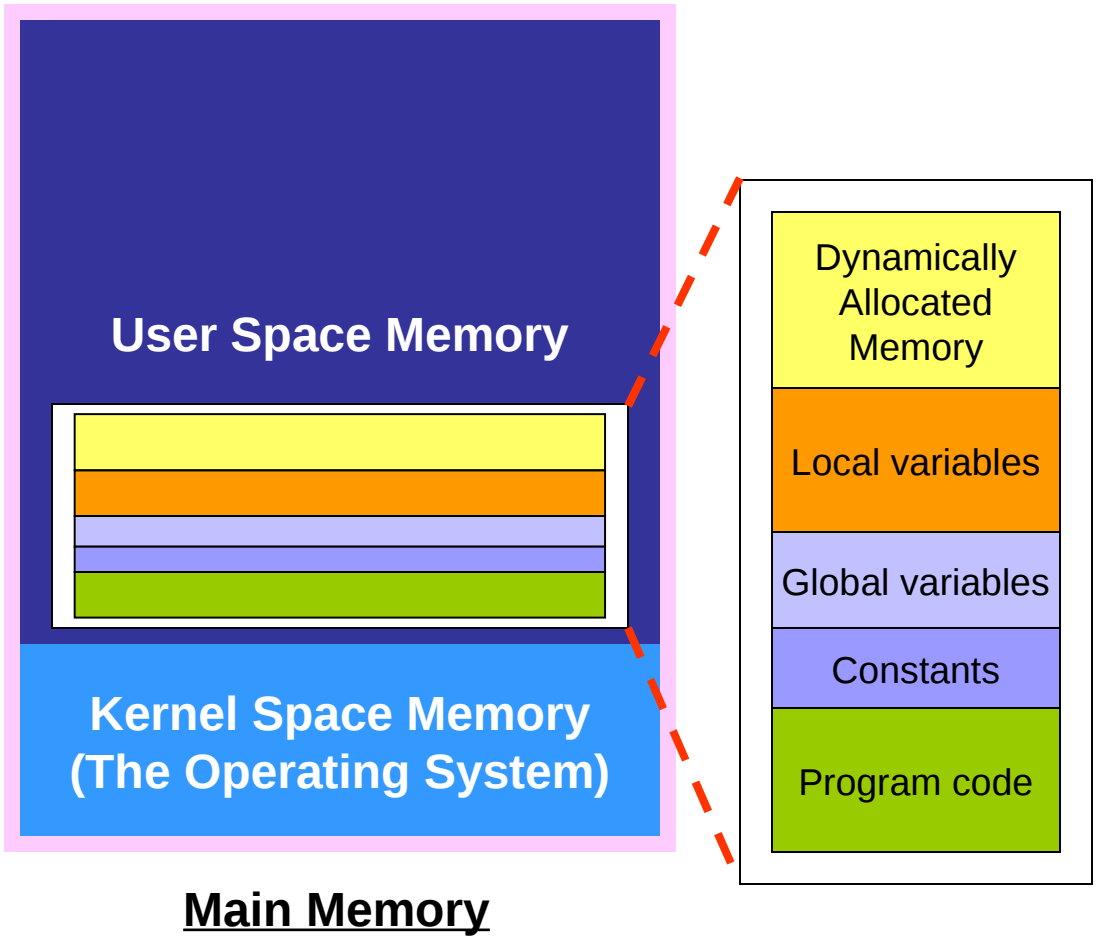
Main Memory

Memória de espaço de usuário é para Processos usuário.

Toda a memória acessível de um processo é encontrada no espaço de memória do usuário. Um processo a cada vez, Sem abstração

MOV REGISTER1, 1000

Layout Memória Física



Multiplos programas sem abstração:
O sistema deve salvar o Conteúdo em disco.

Outra solução: IBM360
Hardware especial:
memória blocos de 2k
chave de 4bits p/bloco
1MB = 512 reg. - 256B
PSW (chave)
chave Proc <> chave PSW
INT
SO altera chaves

Múltiplos programas sem abstração de memória

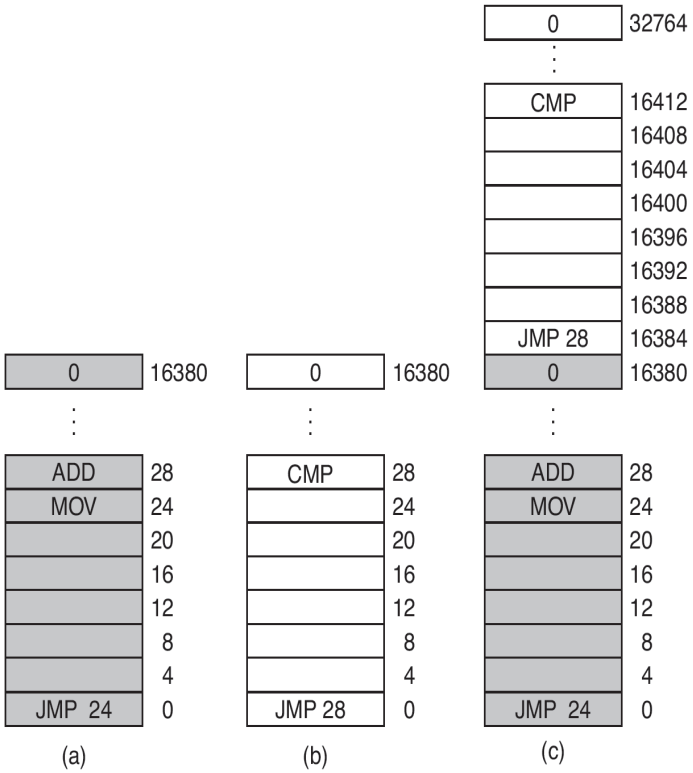
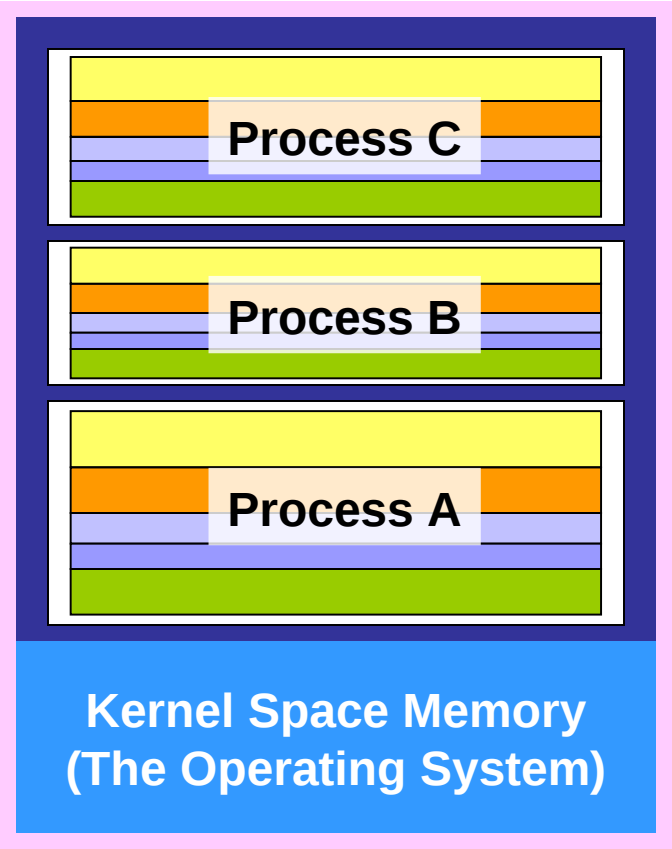


Figura 3.2 Ilustração do problema de realocação. (a) Um programa de 16 KB. (b) Outro programa de 16 KB. (c) Os dois programas carregados consecutivamente na memória.

Problema: ambos referenciam a memória física absoluta
Solução do 360: modificar na carga – **relocação estática** - requer informações adicionais: contém ou não endereços

Layout Memória Física



Main Memory

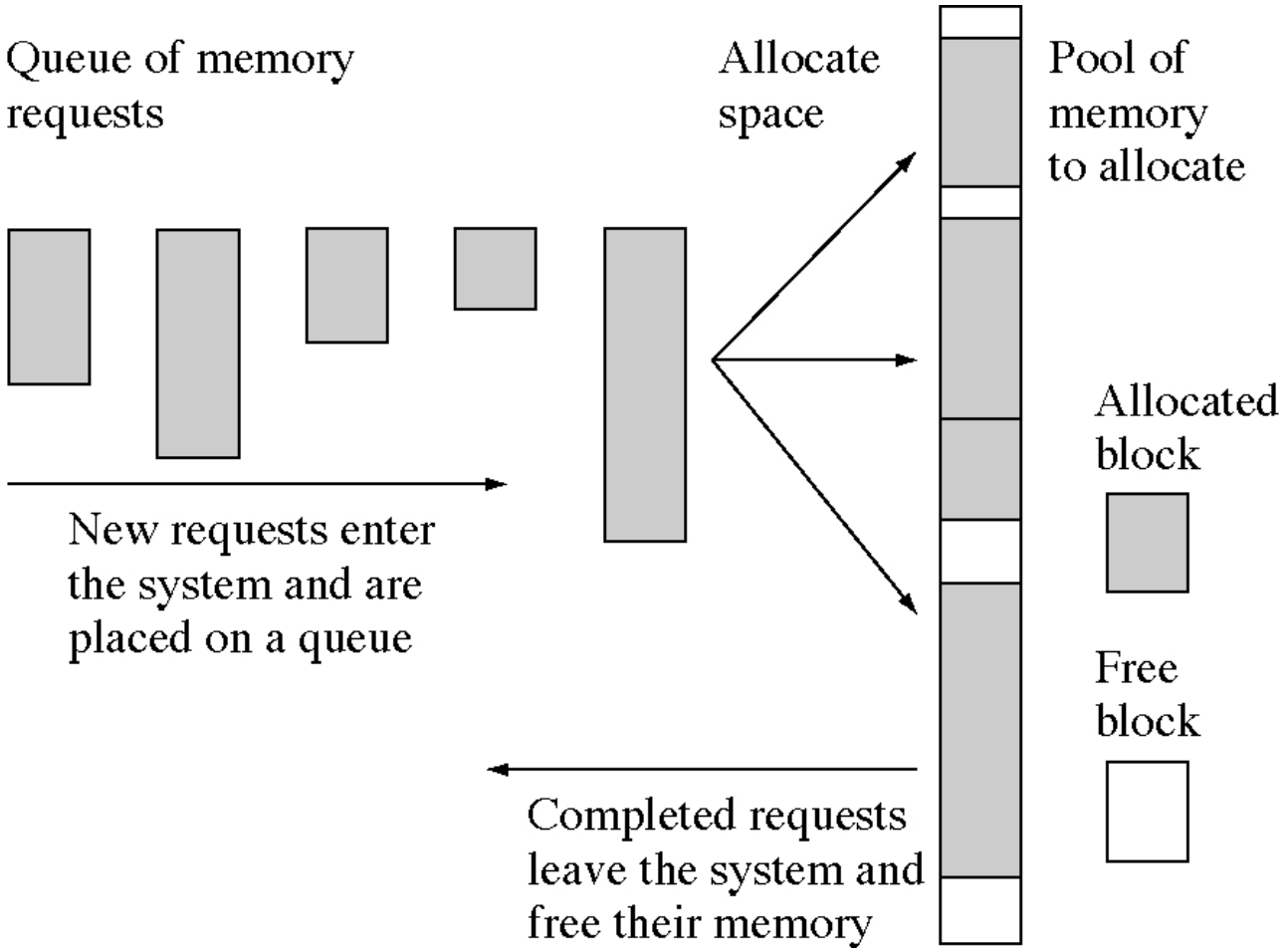
Permitir a execução de múltiplos programas
Sem interferência é necessário resolver dois
Problemas principais: proteção e realocação

Solução IBM360 para proteção funciona para proteção e não para realocação

Solução melhor :
- Abstração de memória
- espaço de endereçamento.

A partir dai o problema é como e para quem alocar memória.

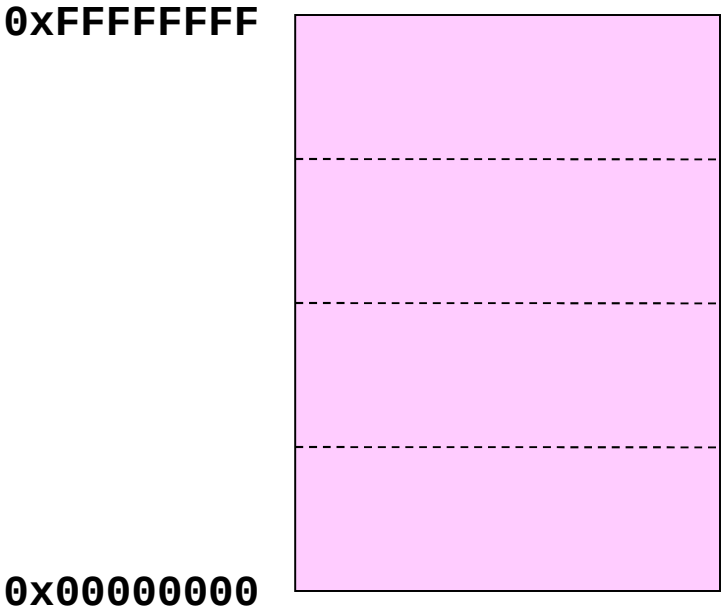
O problema alocação de memória



Espaço de Endereçamento

- O que é espaço de endereçamento?

  Quando a UCP le/escreve na memória, um endereço de memória é necessário.Um conjunto de endereços que um processo pode usar para endereçar a memória.



0xFFFFFFFF = um byte

= 2^{32} – 1

= 4G

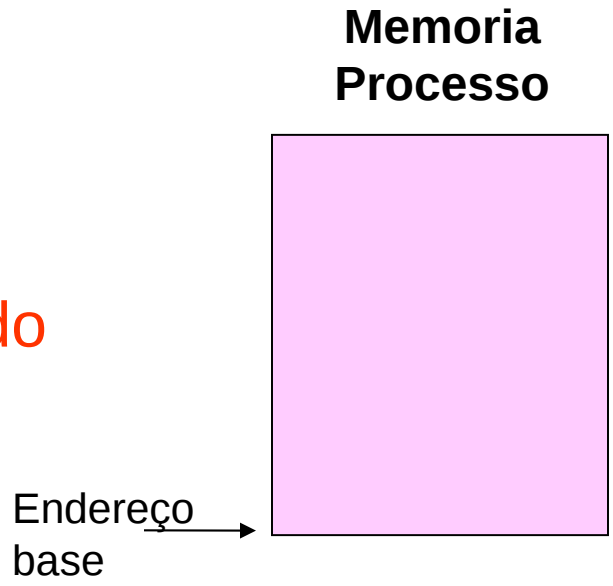
O espaço de memória endereçável é 4G-1.

Endereço lógico X endereço físico

- Dar a cada programa seu próprio espaço de endereçamento, de modo que o endereço 28 em um programa signifique uma localização física diferente do endereço 28 de outro programa:
  - Códigos compilados não usam endereços físicos.
  - Usam endereçamento lógico.
  - Um endereço lógico pode ser transformado em endereço físico pelo kernel ou a UCP.
    - Existe um chip memory management unit (MMU) na UCP.

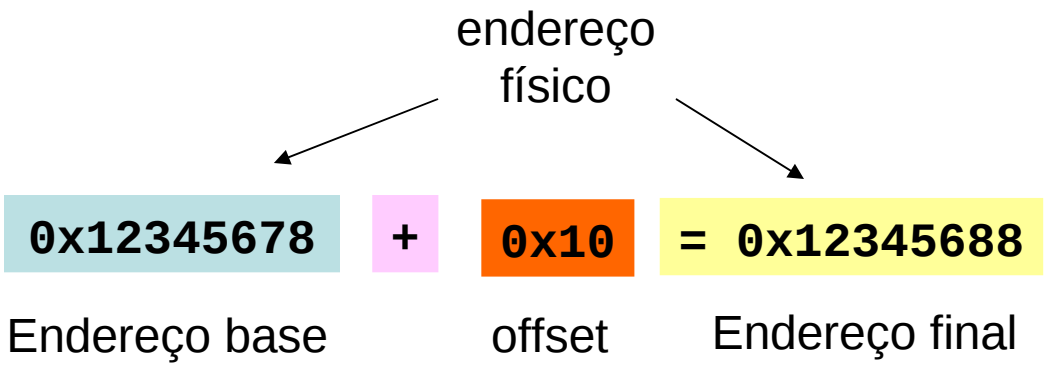
Endereço lógico X endereço físico

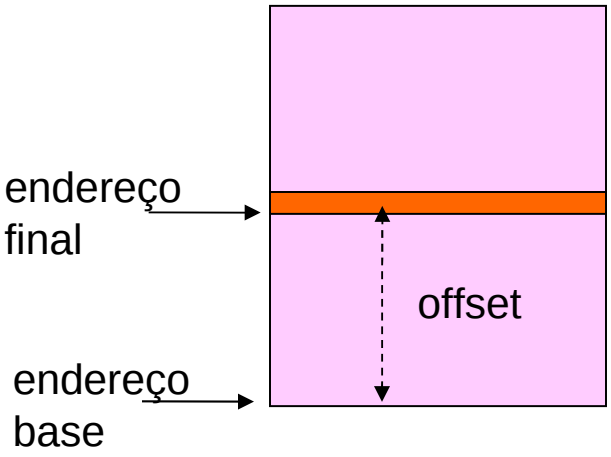
- Relocação dinâmica: Um endereço lógico é um offset na memória de um processo.
  - Todo processo tem um endereço base, que é um endereço físico.
  - O endereço base é o menor endereço do processo.
  - O endereço base irá mudar quando mudar o processo.



Endereço lógico X endereço físico

- Para acessar qualquer endereço no processo
  - O sistema usa o endereço base como a base.
  - Realiza uma soma do endereço base com um deslocamento.
  - Assim, o sistema pode ir para qualquer localização em um processo.





Registrador-base e limite

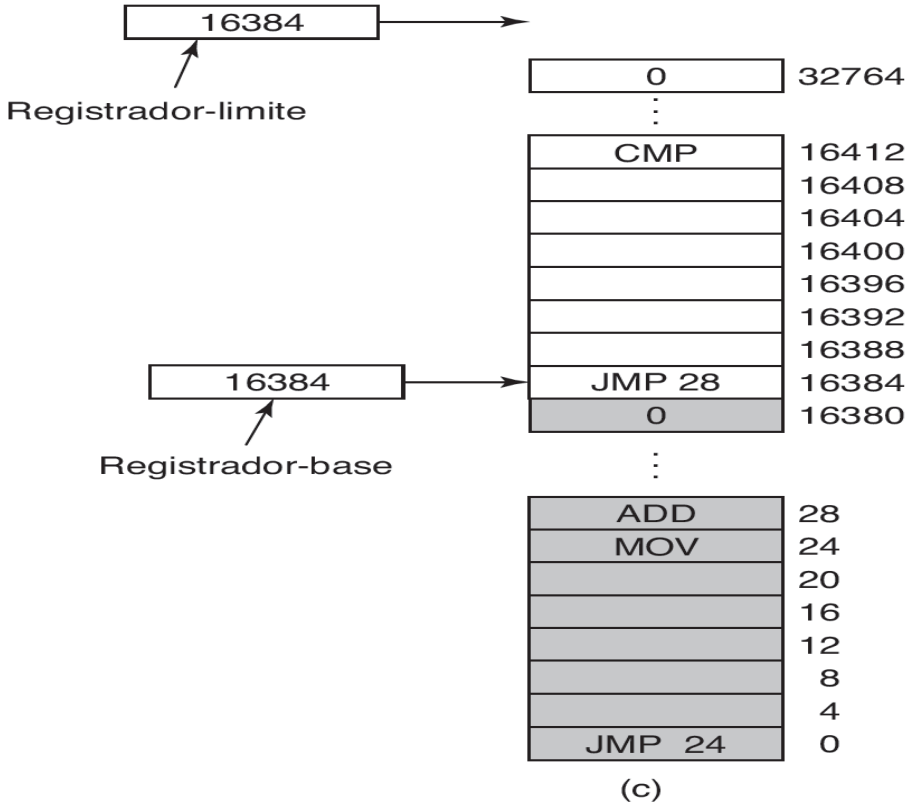
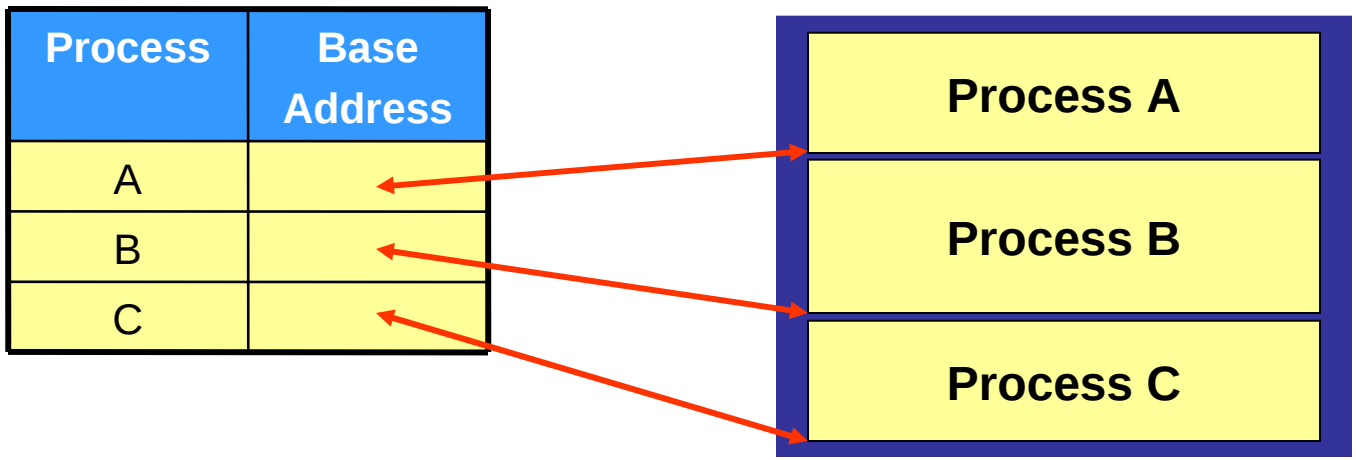


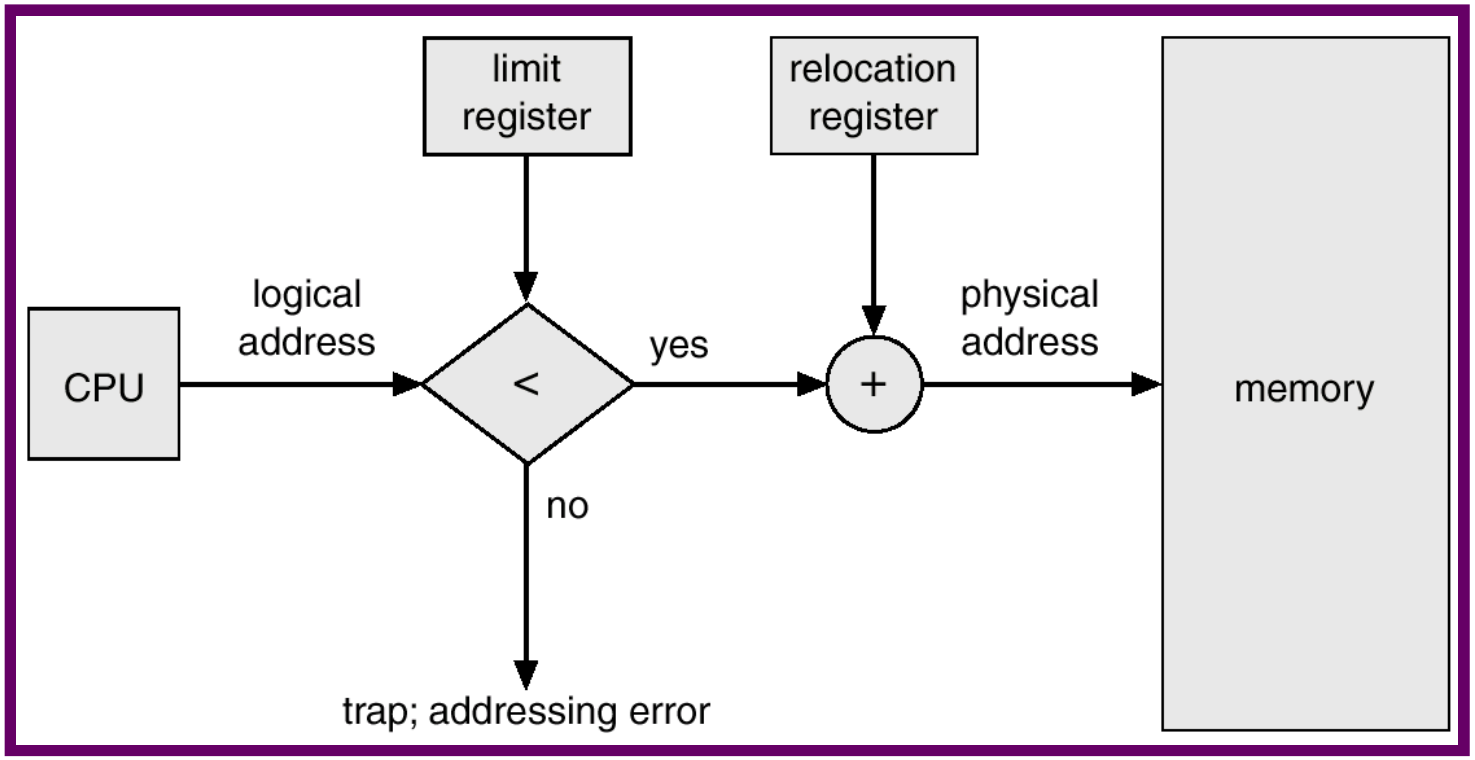
Figura 3.3 O registrador-limite e o registrador-base podem ser usados para dar a cada processo um espaço de endereçamento independente.

Endereço lógico X endereço físico

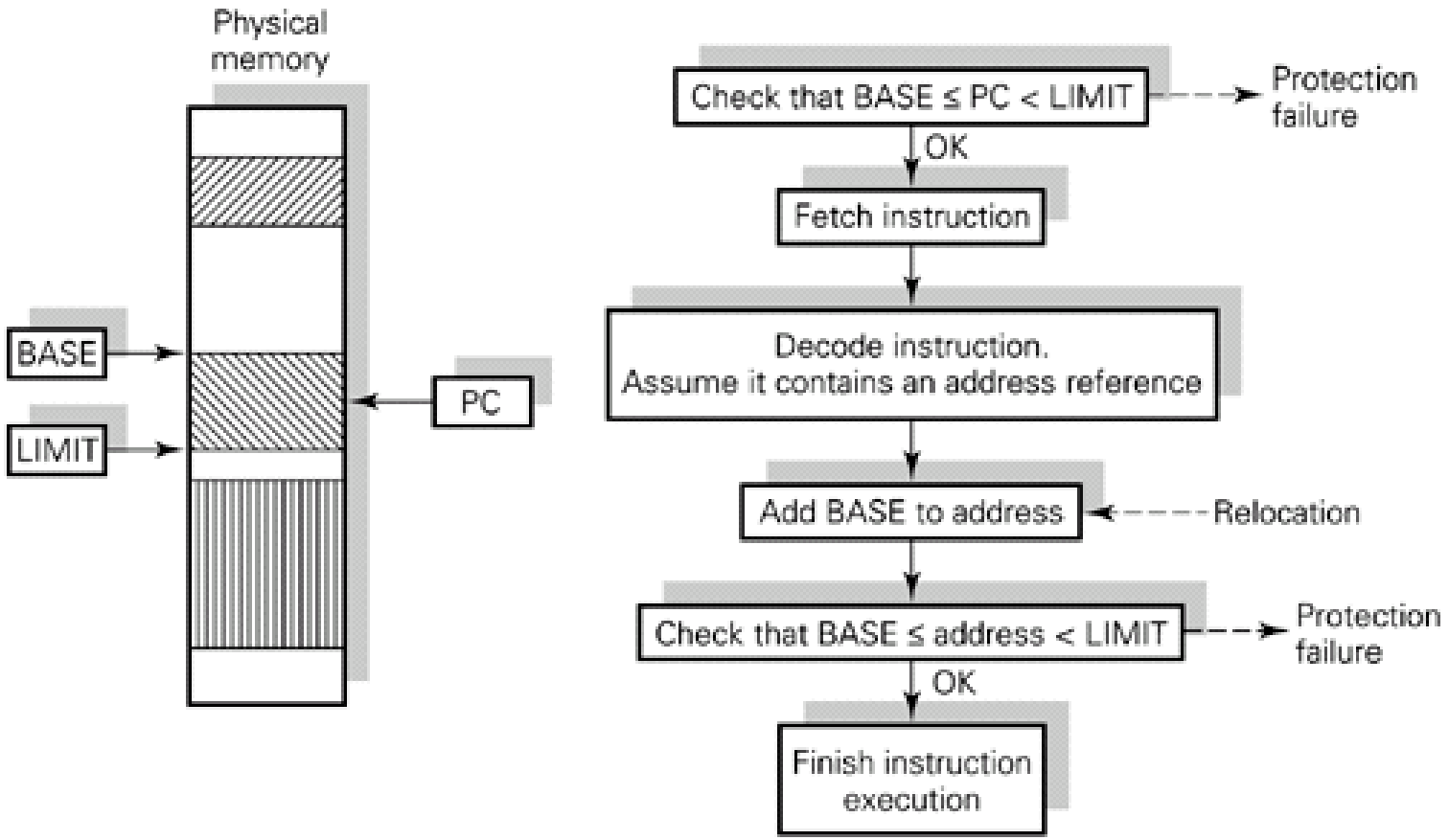
- Quando um novo processo é criado ou um processo antigo é *swapped in* do disco:
  - Apenas o endereço base daquele processo muda.
  - O kernel mantem uma tabela de endereços de base.



Suporte de Hardware: Relocação e Registradores Limite



Suporte de Hardware: Relocação e Registradores Limite

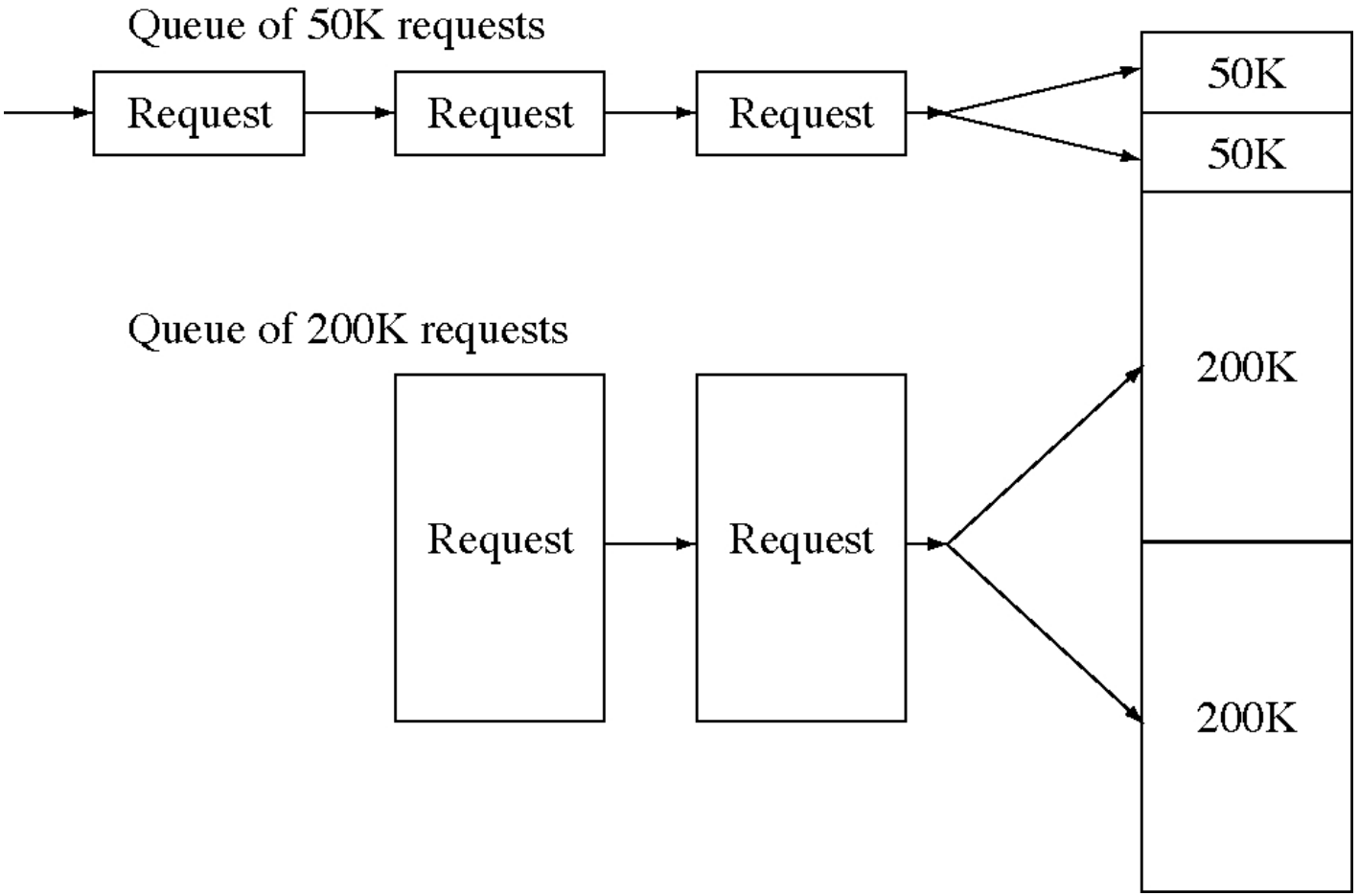


Alocação Contígua

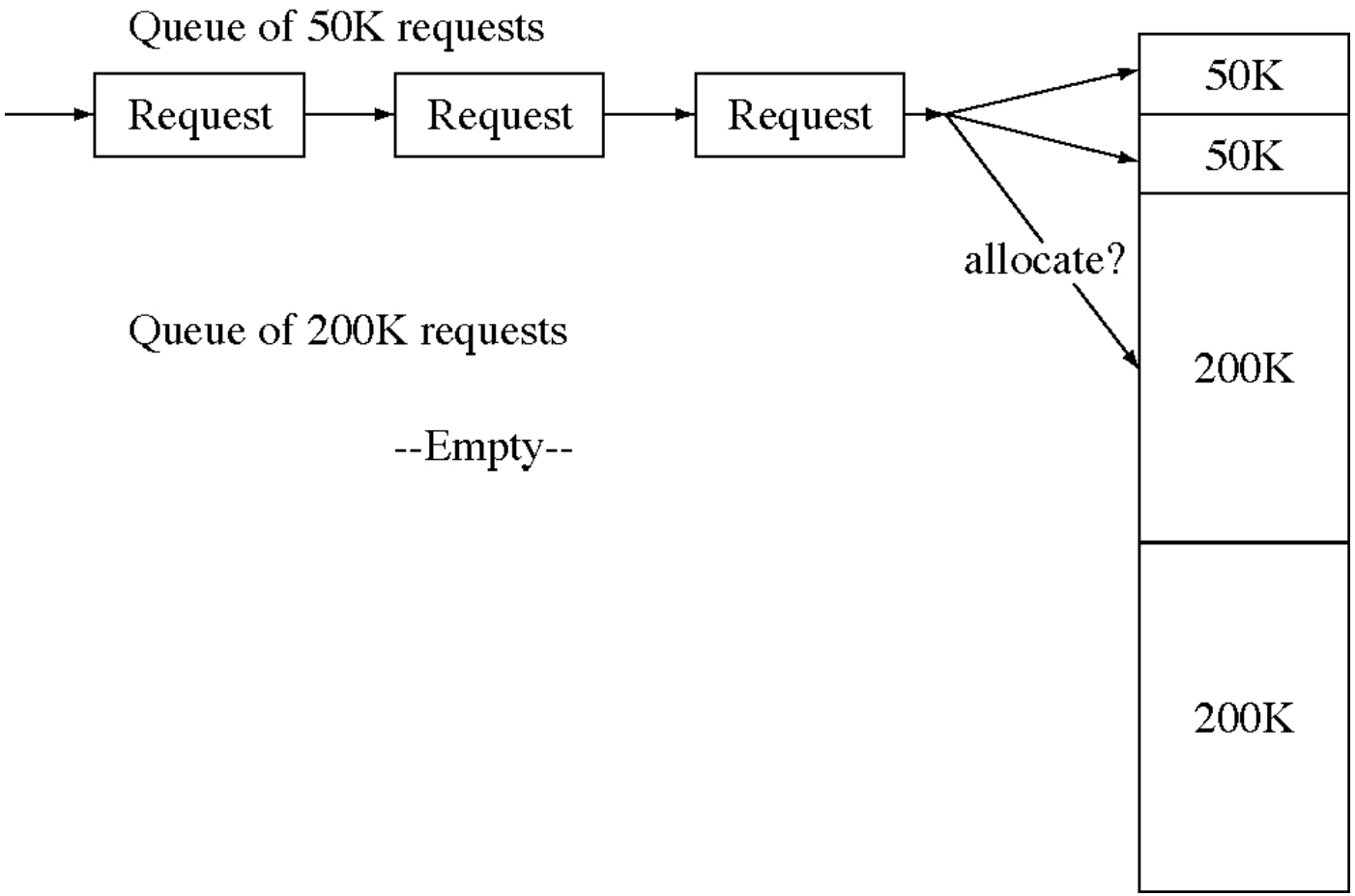
- Memória principal dividida em 2 partições:
  - SO Residente, usualmente mantido em memória baixa com vetor de interrupção.
  - Processos de usuário mantidos em memória alta.

- Alocação de partição única para cada processo:
  - Esquema de registrador de relocação usado para proteção entre processos de usuário e mudança de código e dados do SO.
  - Registrador de relocação contém valor do menor endereço físico; registrador limite contém alcançe dos endereços lógicos – cada endereço lógico deve ser menor que o registrador limite.
- Partições Fixas e Variáveis

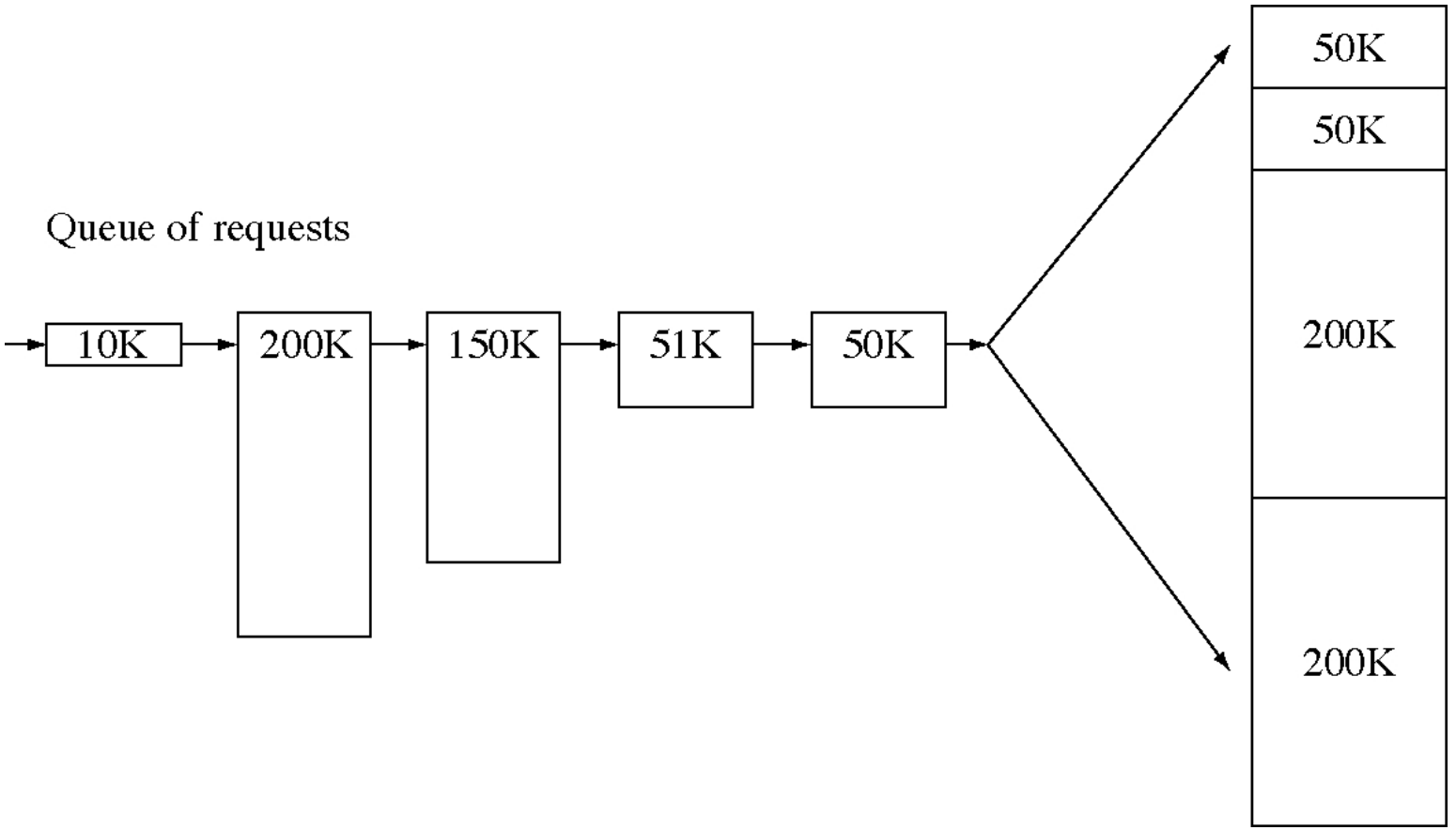
Partições fixas – Múltiplas filas



Bloco maior livre, aloca?



Partições fixas – única fila



Alocação Contígua (1)

- Alocação partições variáveis
  - *Buraco* – bloco de memória disponível; buracos de tamanhos variados estão espalhados pela memória.
  - Quando um processo chega, é alocado memória do buraco grande o suficiente para acomodar o mesmo.
  - SO mantém informação sobre:
    a) partições alocadas b) partições livres (buraco)

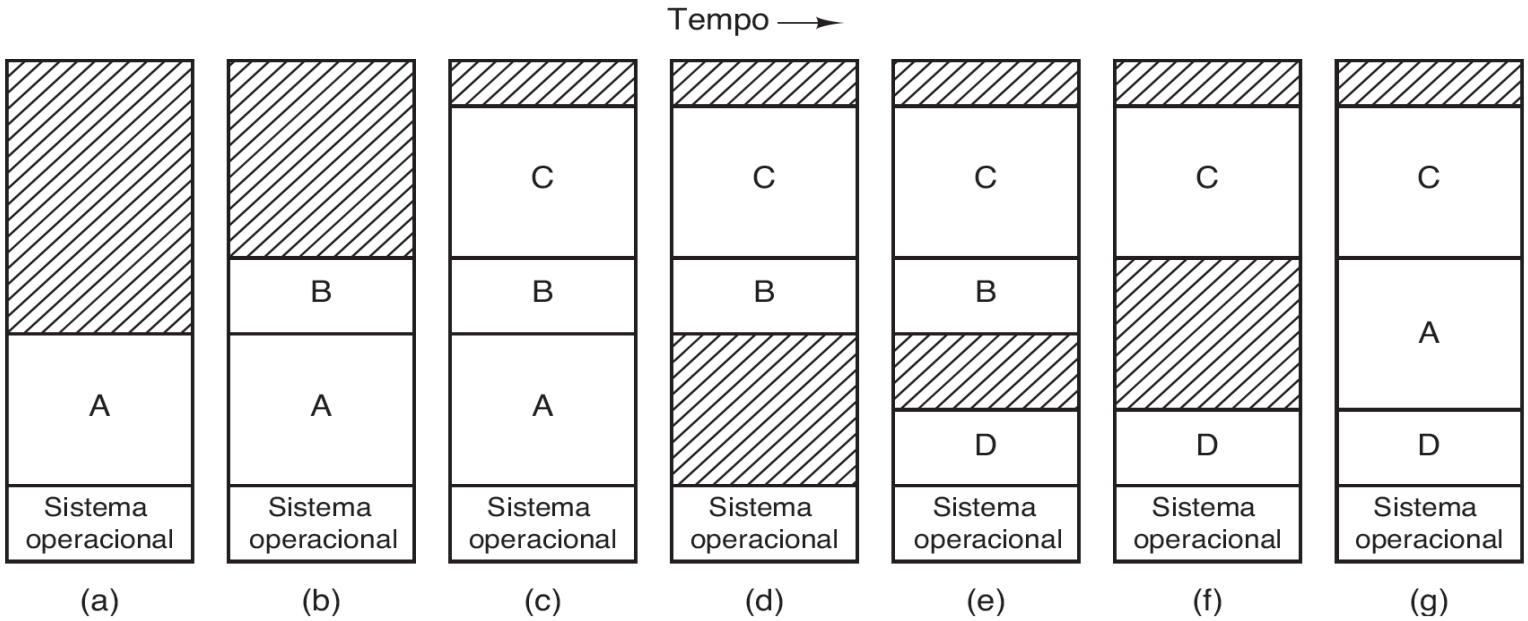


Figura 3.4 Alterações na alocação de memória à medida que processos entram e saem dela. As regiões sombreadas correspondem a regiões da memória não utilizadas naquele instante.

Alocação Contígua (2)

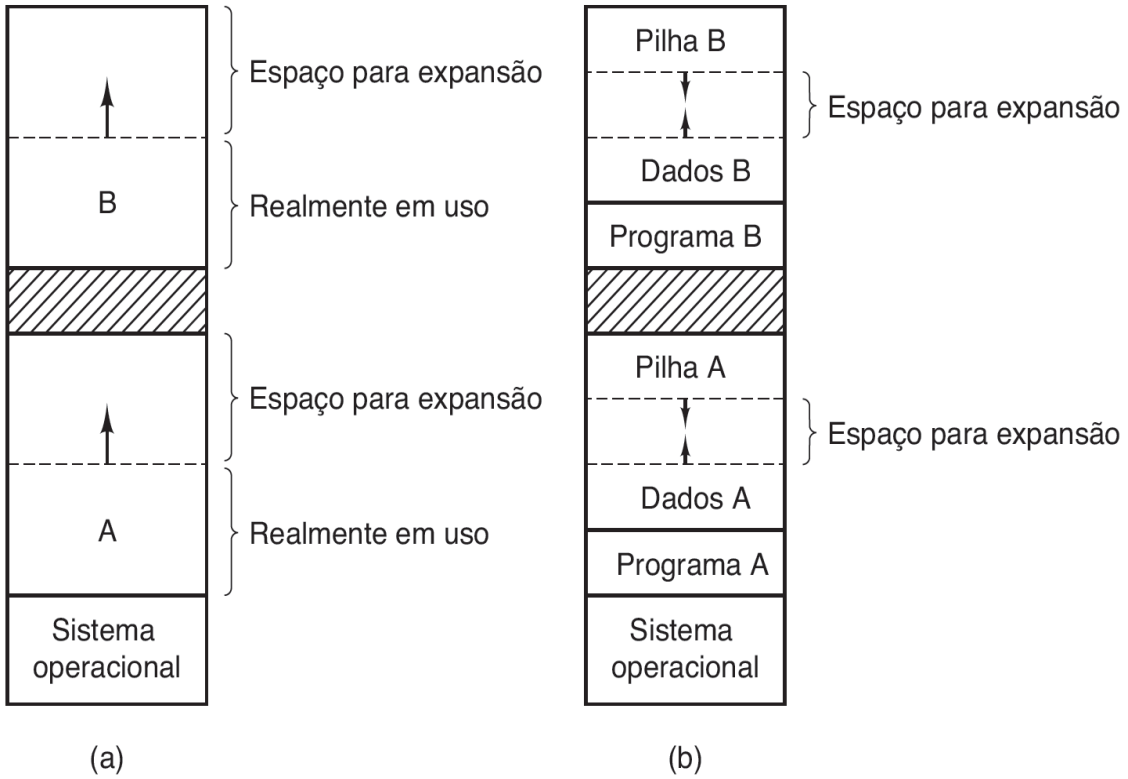
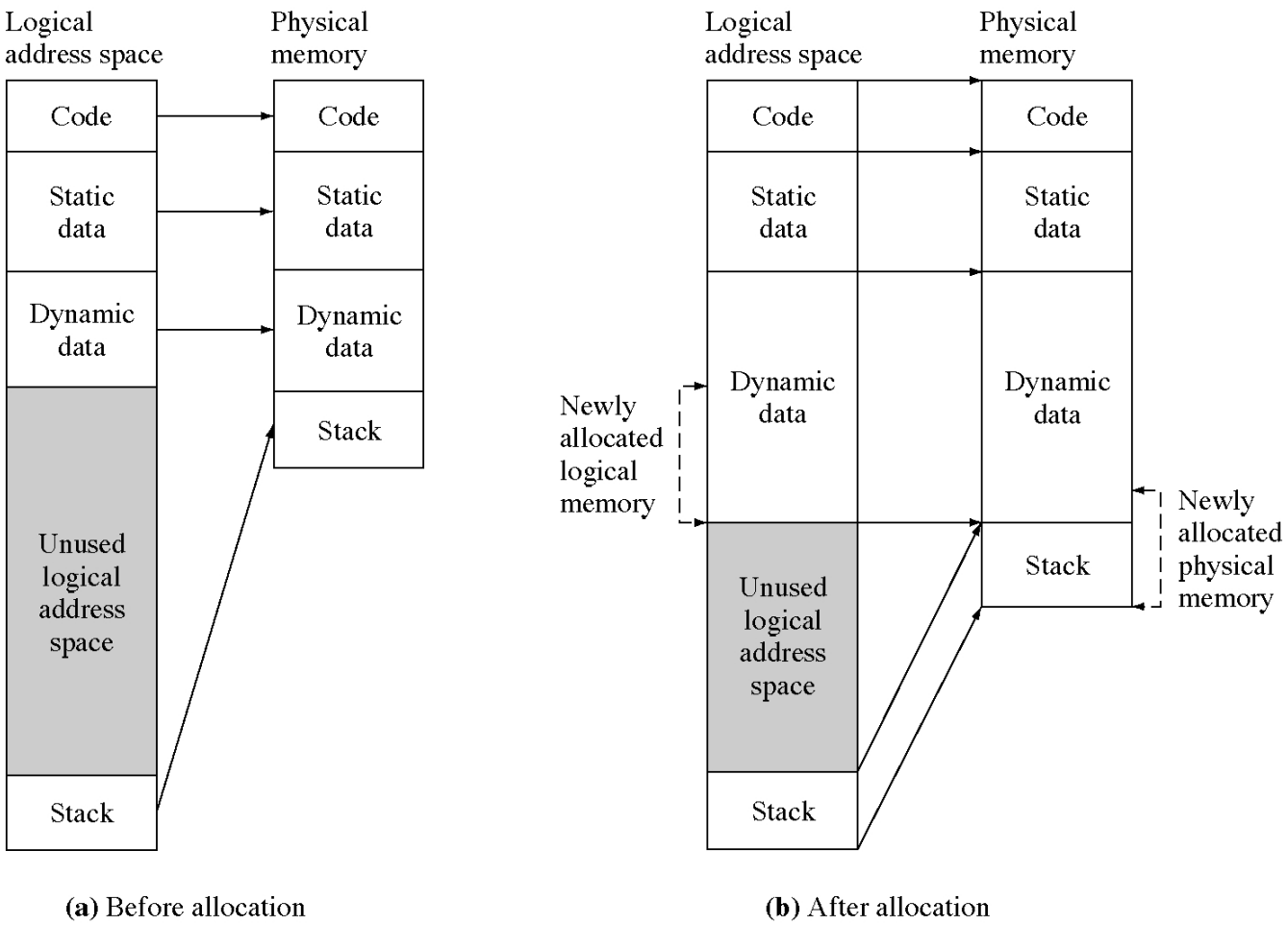
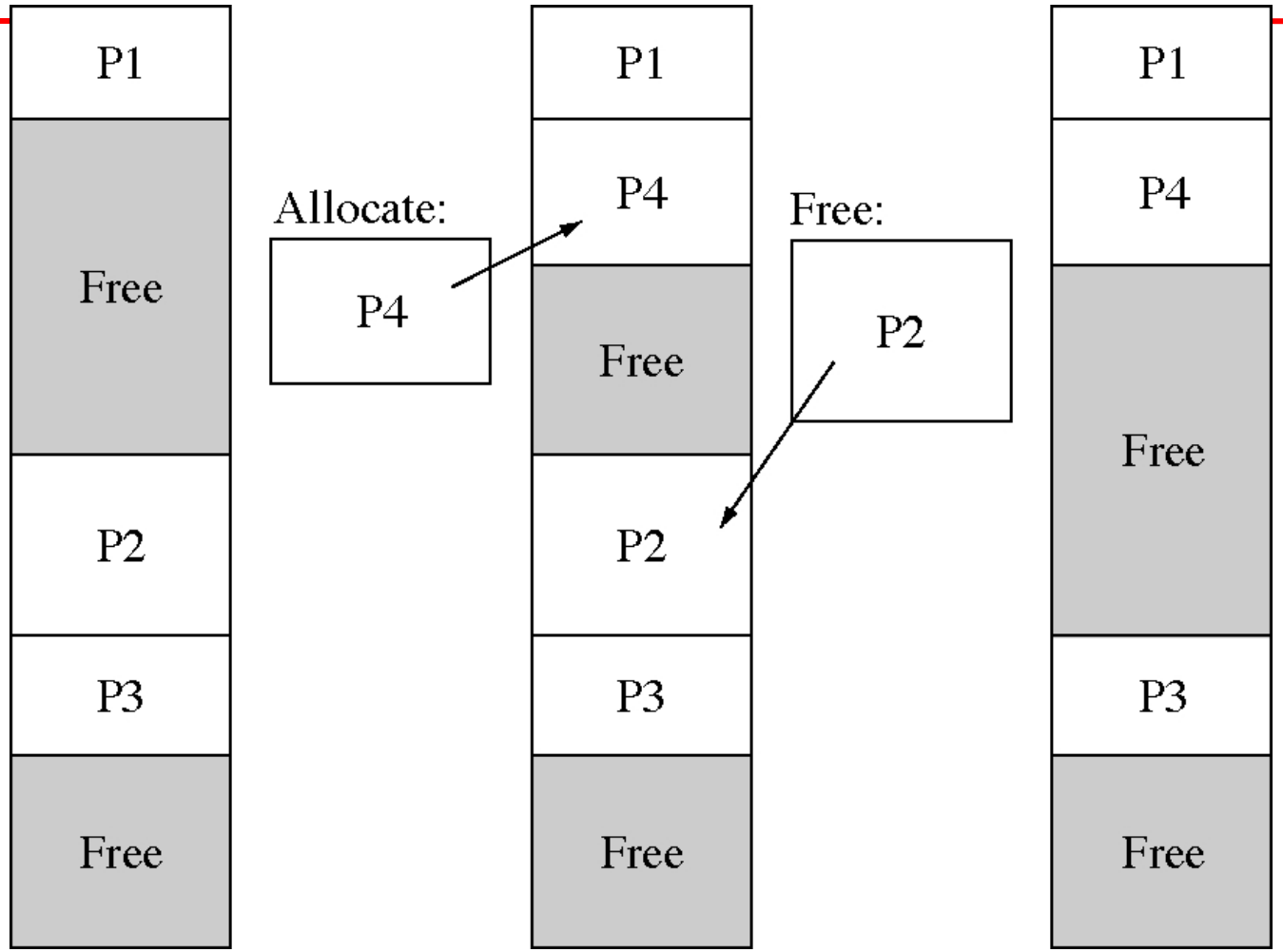


Figura 3.5 (a) Alocação de espaço para um segmento de dados em expansão. (b) Alocação de espaço para uma pilha e um segmento de dados em crescimento.

memória alocada a um processo executando

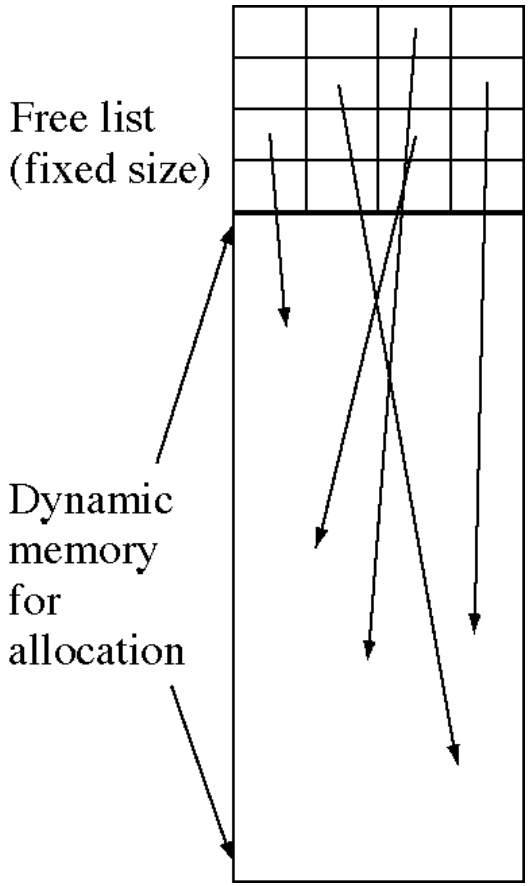


Alocação e liberação de blocos



Gerência de Memória

Quando a memória é alocada dinamicamente, o SO deve gerenciá-la.



Gerenciamento de memória com mapa de bits

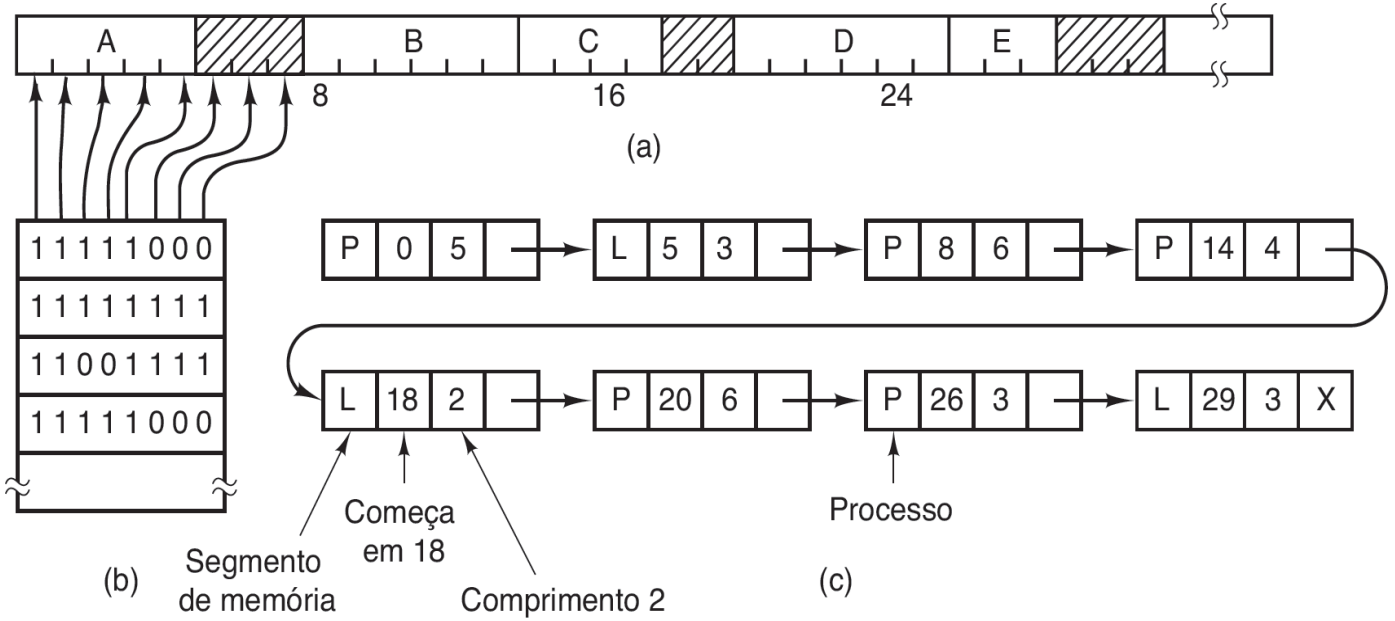


Figura 3.6 (a) Parte da memória com cinco processos e três segmentos de memória. As marcas mostram as unidades de alocação de memória. As regiões sombreadas (0 no mapa de bits) estão livres. (b) O mapa de bits correspondente. (c) A mesma informação como lista.

Gerência de Memória com Mapa de Bits

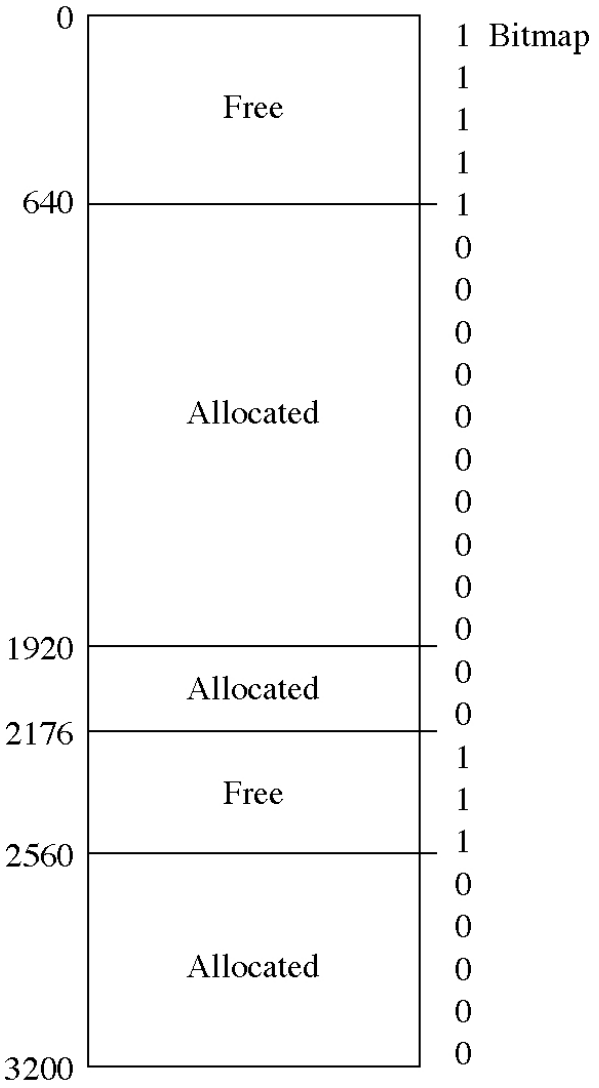
Com mapa de bits a memória é dividida em unidades de alocação, que podem conter apenas poucas palavras ou ter vários Kbytes.

Tamanho da unidade de alocação ???

Ex: 4bytes ---- (1/33) maior?

Problema principal ???

Carregar um processo com *k* unidades



Gerência de Memória com Lista Ligada

Outra maneira de gerenciar o uso de memória é manter uma lista encadeada de segmentos de memória alocados e disponíveis.

Cada elemento dessa lista encadeada especifica um segmento de memória livre (L) ou um segmento de memória alocado a um processo (P), o endereço onde se inicia esse segmento, seu comprimento e um ponteiro para o próximo elemento da lista.

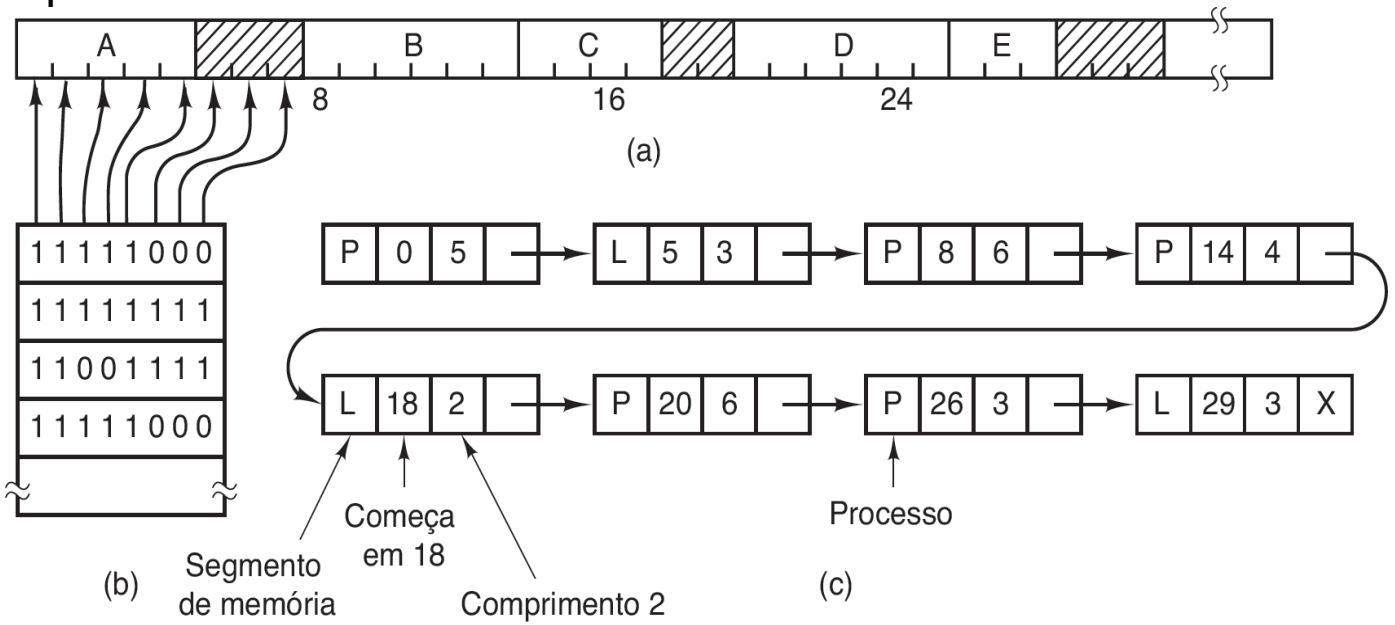


Figura 3.6 (a) Parte da memória com cinco processos e três segmentos de memória. As marcas mostram as unidades de alocação de memória. As regiões sombreadas (0 no mapa de bits) estão livres. (b) O mapa de bits correspondente. (c) A mesma informação como lista.

Gerenciamento de memória com listas encadeadas

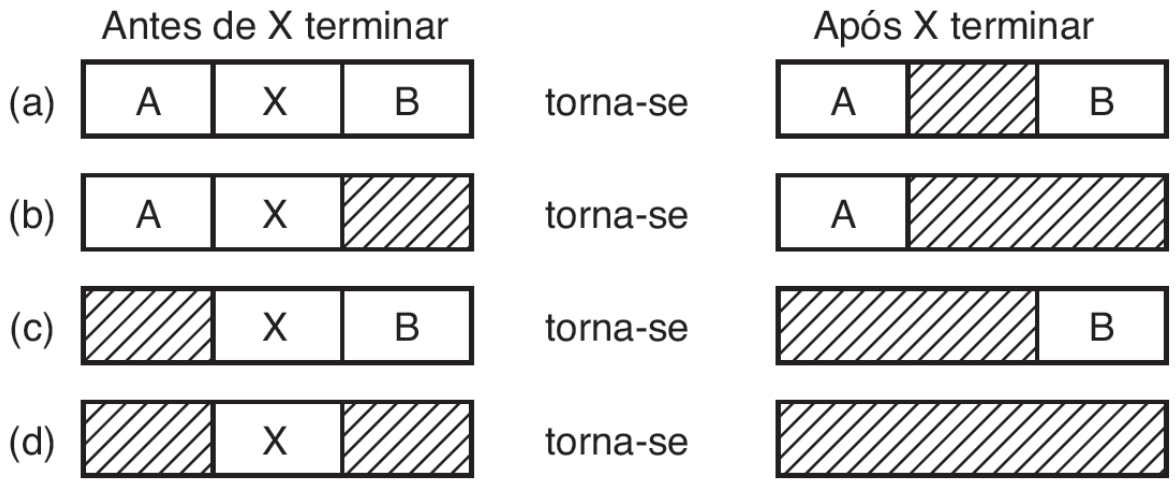


Figura 3.7 Quatro combinações de vizinhos para o processo que termina, *X*.

Um processo que termina sua execução tem dois vizinhos na lista (em geral), esses vizinhos podem ser segmentos alocados ou livres, resultando nas seguintes combinações acima.
O apontador para o segmento fica no descritor do processo.
Lista dupla-encadeada é melhor??

Gerência de Memória com Lista Ligada

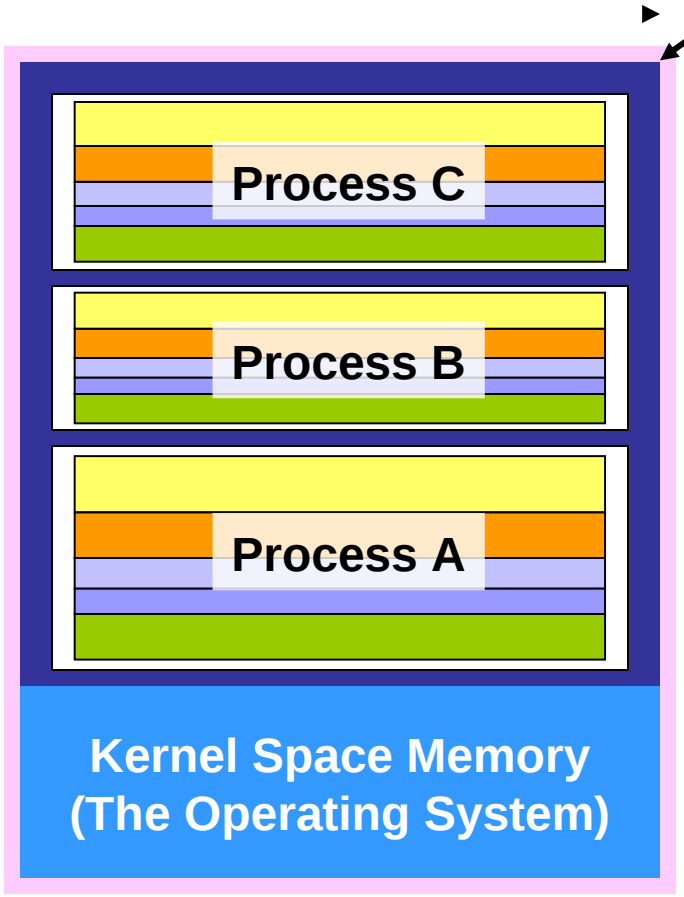
Quando segmentos de memória alocados a processos e segmentos de memória livres são mantidos em lista ordenada por endereço, é possível utilizar diversos algoritmos para alocar memória a um processo. Suponha que o gerenciador sabe o tamanho necessário:

- **First-fit**: Aloca o primeiro buraco grande o suficiente.
- **Best-fit**: Aloca o menor buraco grande o suficiente; deve procurar a lista inteira, ou ….. . Produz buracos ….
- **Worst-fit**: Aloca o maior buraco; deve procurar a lista inteira. Produz buracos …..

First-fit e best-fit melhores que worst-fit em termos de velocidade e utilização de memória.

Quick-fit : diversas listas com tamanhos diferentes, busca fica rápida liberação??

Layout Memória Física



Main Memory

Overflow na memória?!

E se tiver "**muitos**" processos no sistema?

Usamos espaço de disco para armazenar processos.

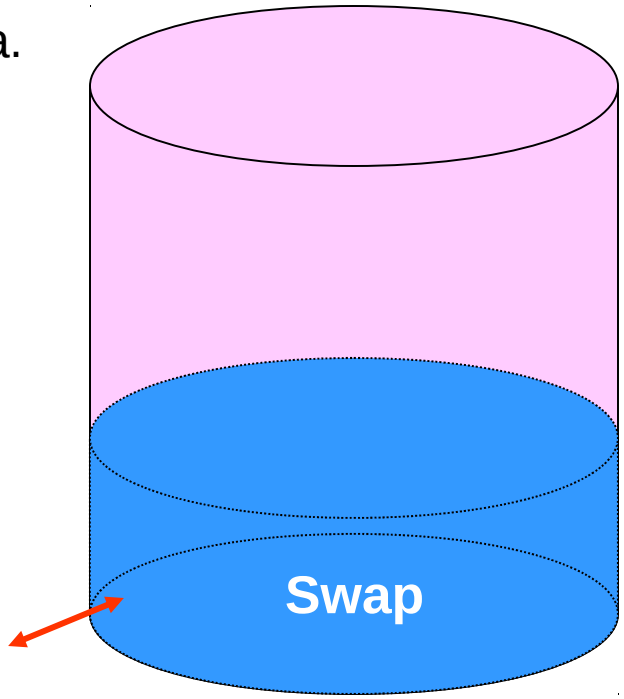
Swapping

Uma porção do disco rígido será reservado para ser usado como memória. Chamamos isto **swap**.

Linux precisa uma partição separada para agir como swap e é chamada **swap partition**.

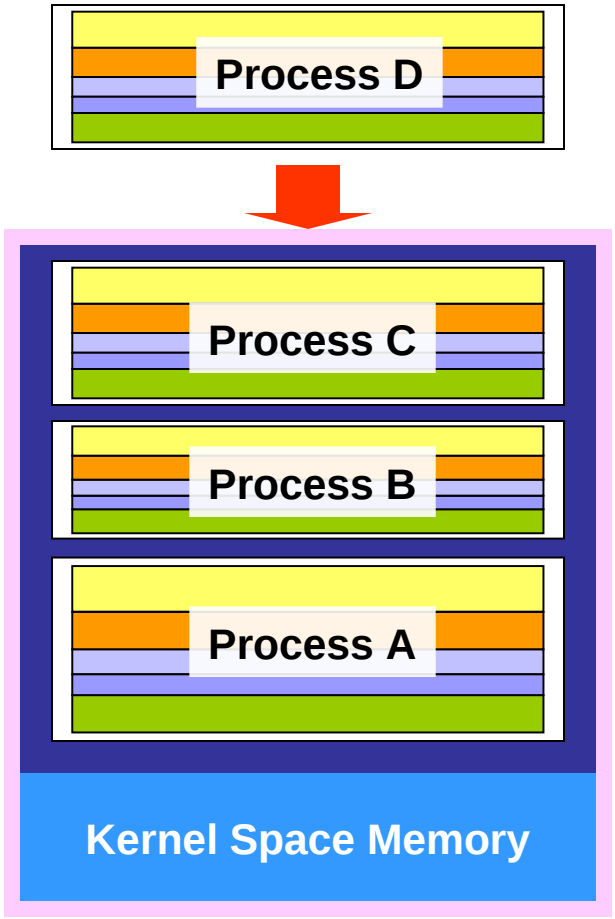
```
# fdisk /dev/sda
......
Command (m for help): p
......
/dev/sda1 ...... Linux
/dev/sda2 ...... Linux swap / Solaris

Command (m for help): _
```



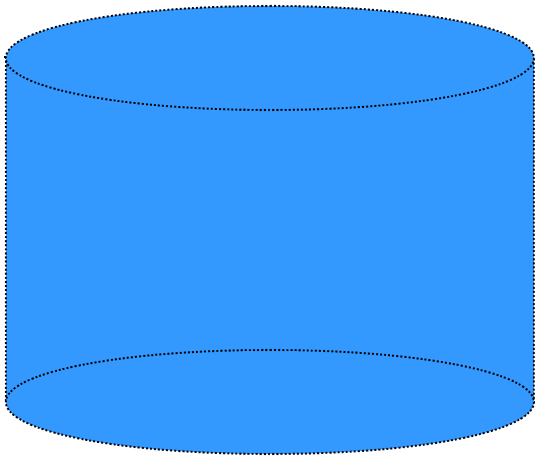
Hard Disk

Swapping



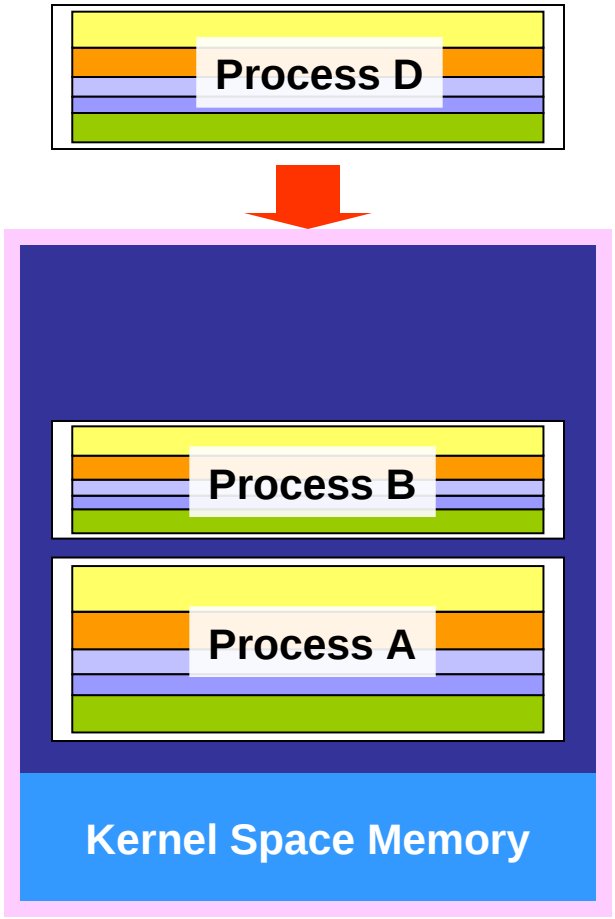
Quando um processo é escalonado, ele precisa memória para executar.

Mas, a memória física esta totalmente ocupada.



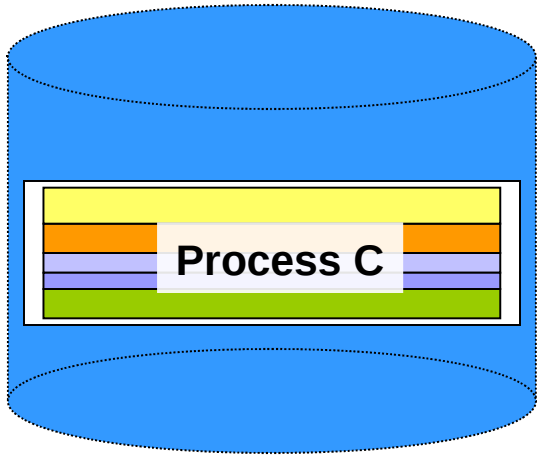
Swap

Swapping



Um dos processos (not running) será swapped out da memória.

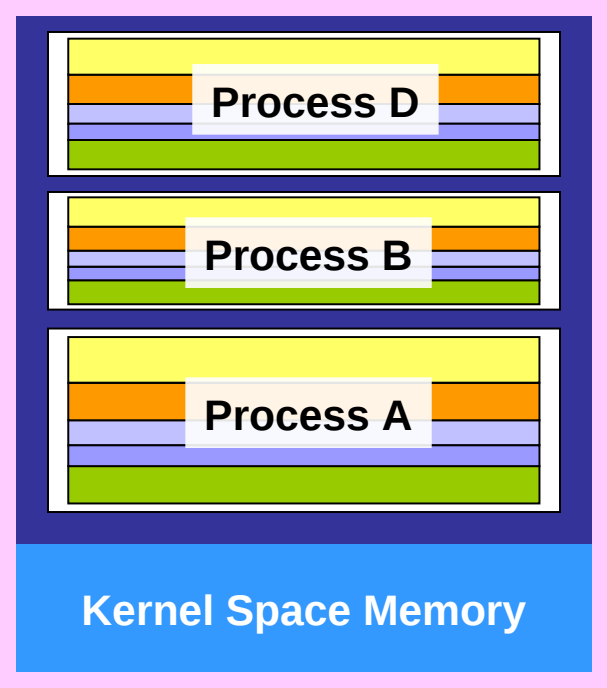
O processo swapped-out será armazenado no swap.



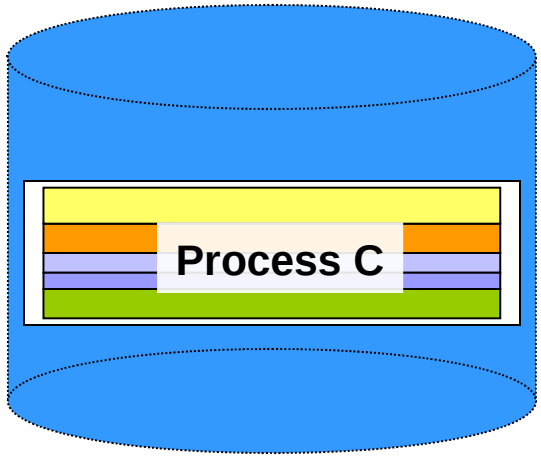
Swap

Swapping

Agora, processo D tem memória necessária para executar.



Processo C será swapped in do disco se ele for escalonado novamente.

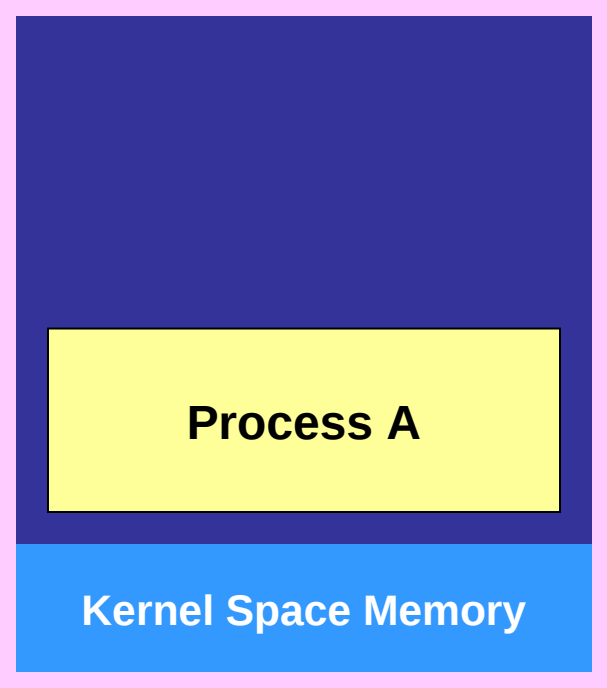


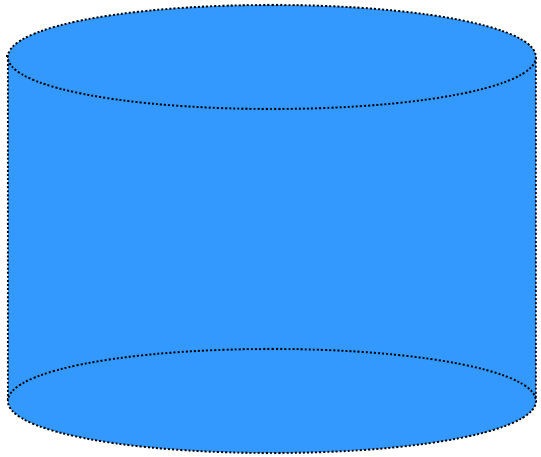
Swap

Swapping: Exemplo

Escalonador :

Processo A é criado e é escalonado.



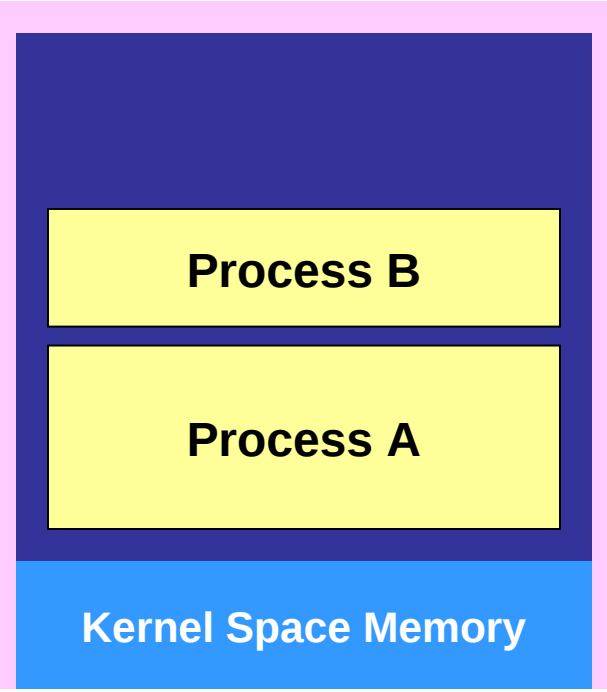


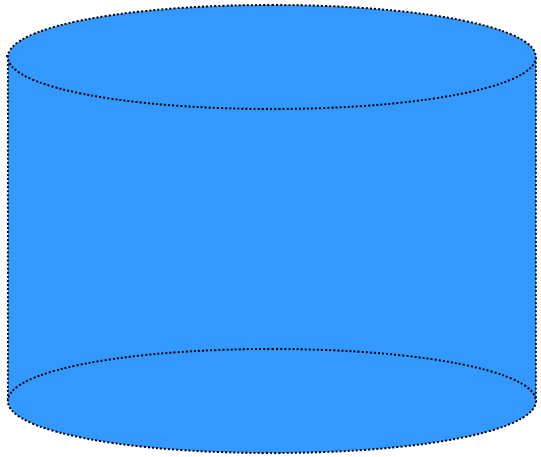
Swap

Swapping: Exemplo

Escalonador :

Processo B é criado e é escalonado.



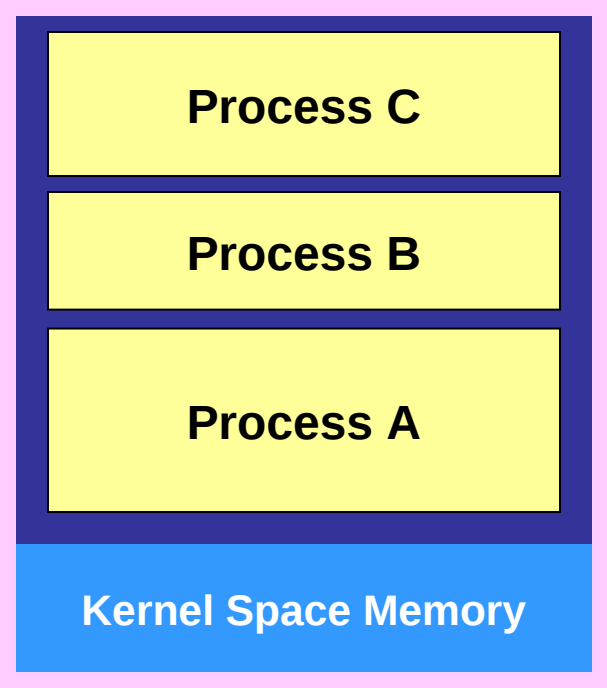


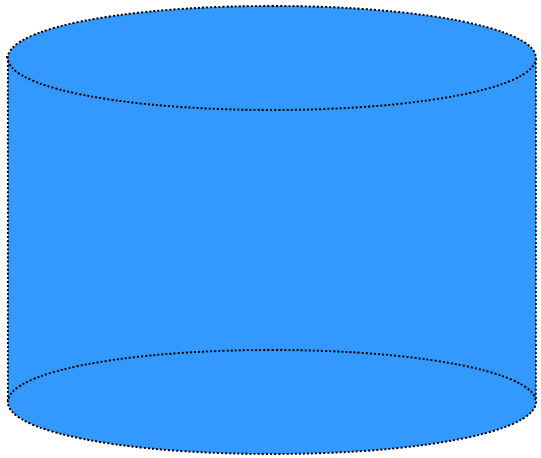
Enlarged Swap

Swapping: Exemplo

Escalonador :

Processo C é criado e é escalonado.



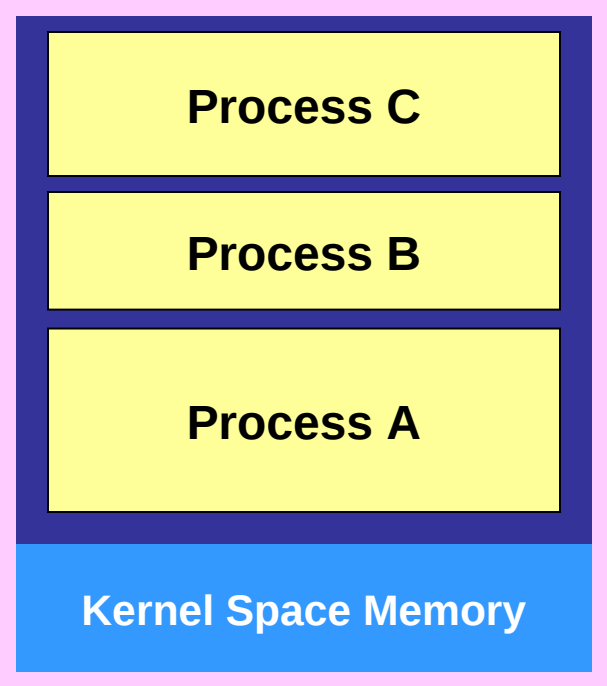


Swap

Swapping: Exemplo

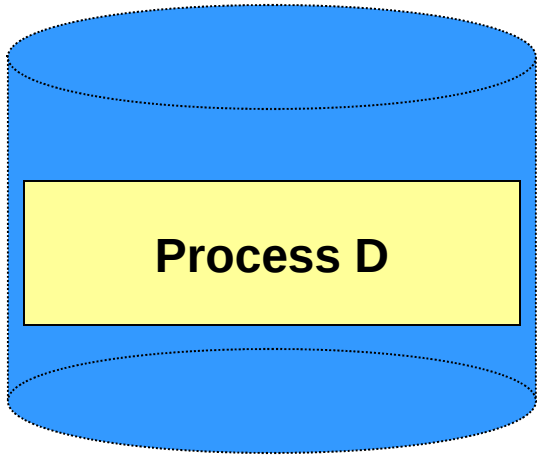
Escalonador :

Processo D é criado e é escalonado.



Gerência de Memória :

Sem memória disponível. Processo D é armazenado no swapped temporariamente.



Swap

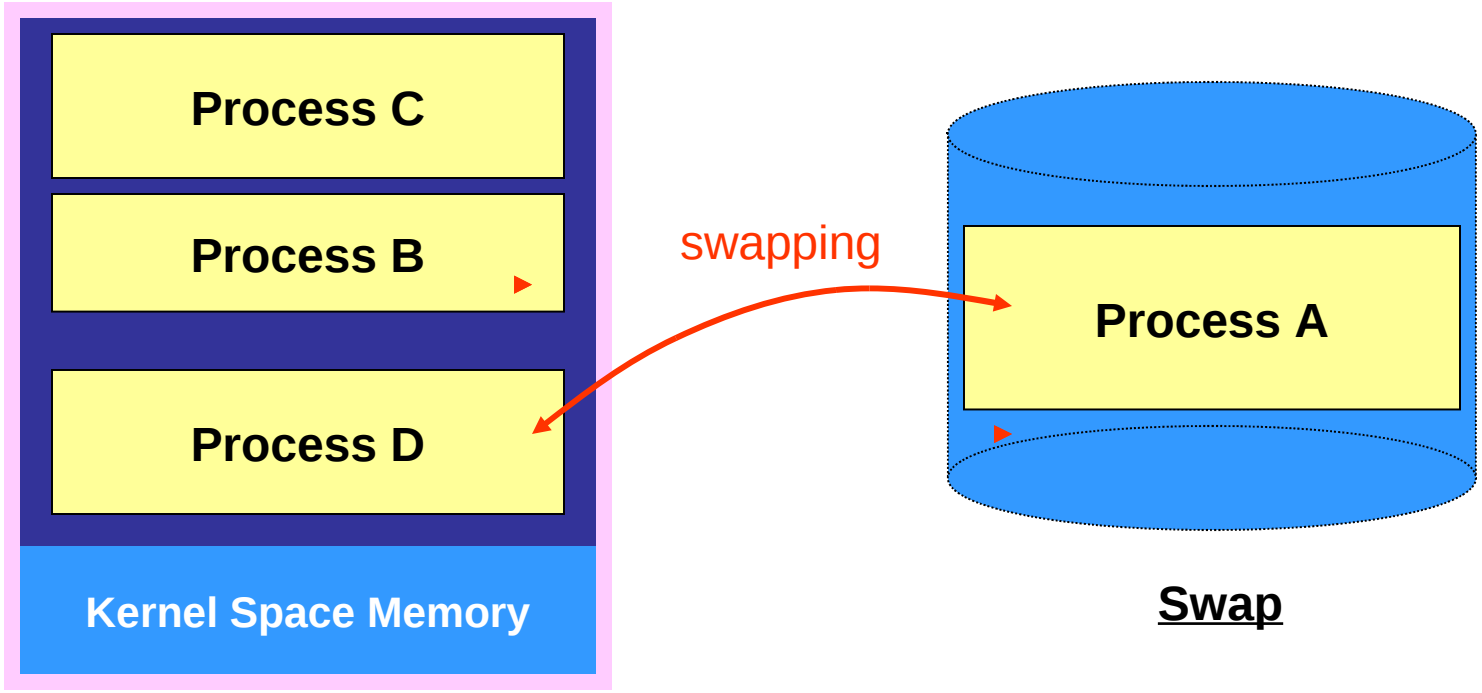
Swapping: Exemplo

Escalonador :

Processo D é criado e é escalonado.

Gerência de Memória :

O kernel decidiu swap out Processo A da memória.



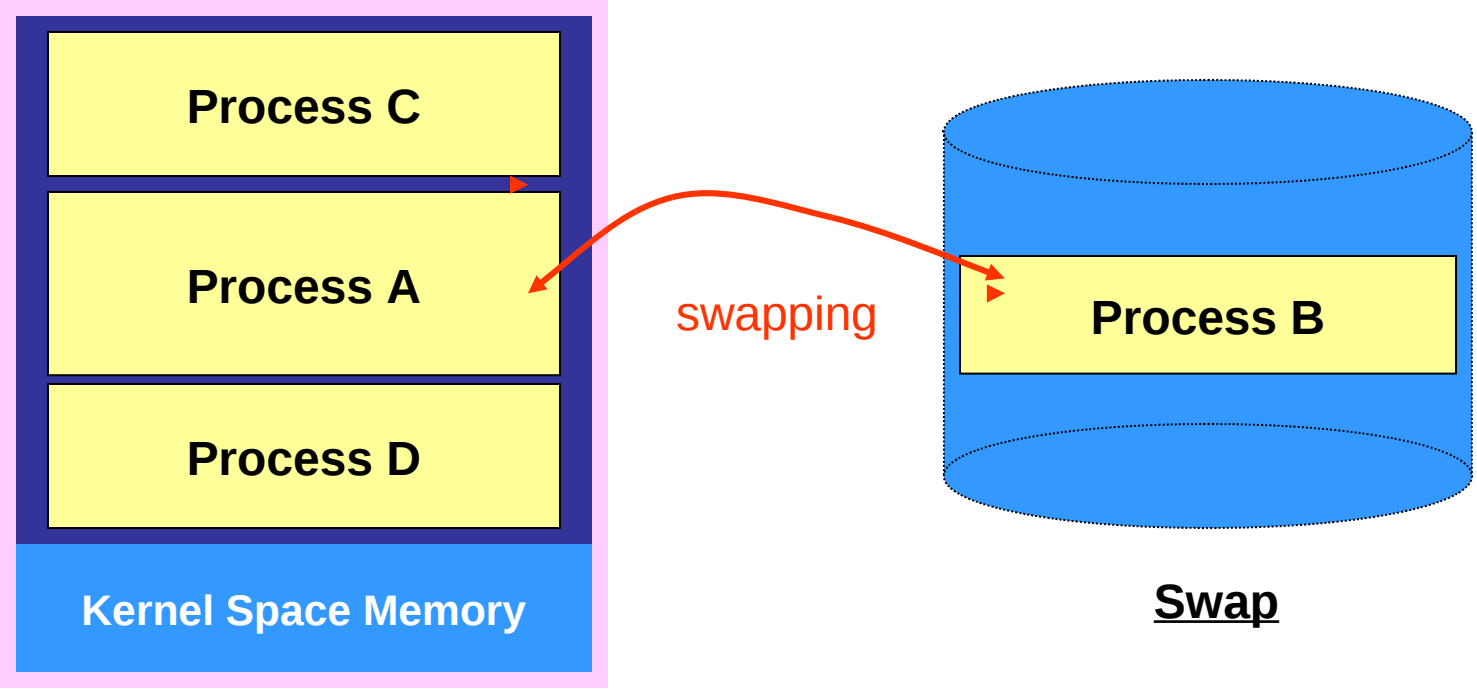
Swapping: Exemplo

Escalonador :

Processo A é escalonado

Gerência de Memória :

O kernel decidiu swap out Processo B da memória.



Swapping

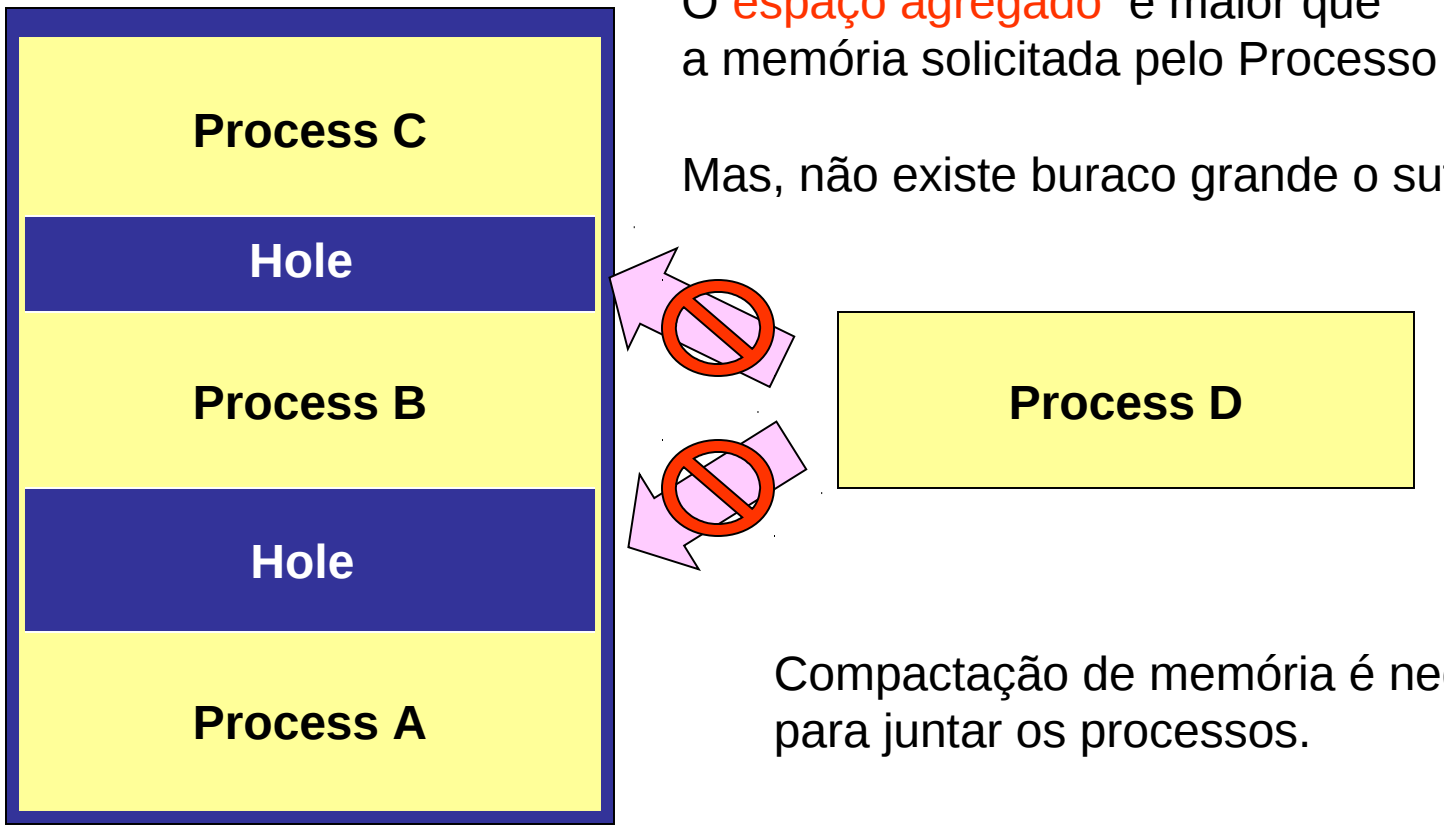
- Exceto pelo fato de suportar mais processos no sistema, swapping é ruim!
  - Aumenta o tempo do chaveamento de contexto.
  - Produz o problema da fragmentação.
    - Fragmentação externa.
  - O que acontece se não existem buracos grandes o suficiente para receber o processo?
    - Desfragmentação: compactação de memória.

Fragmentação

- **Partições Fixas ou Variáveis alocação contígua –** proporcionam problemas de fragmentação da memória:
  - **Fragmentação Externa** – existe espaço de memória suficiente para satisfazer uma requisição, mas não é contíguo.
  - **Fragmentação Interna** – memória alocada pode ser um pouco maior que o requisitado; esta diferença é memória interna da partição, mas não usada.

- Reduzir fragmentação externa por compactação
  - Reorganizar conteúdos de memória de forma a colocar toda memória livre junta em um bloco grande.
  - Compactação é possível apenas com relocação dinâmica e é feita em tempo de execução.
  - Problema E/S

Fragmentação Externa

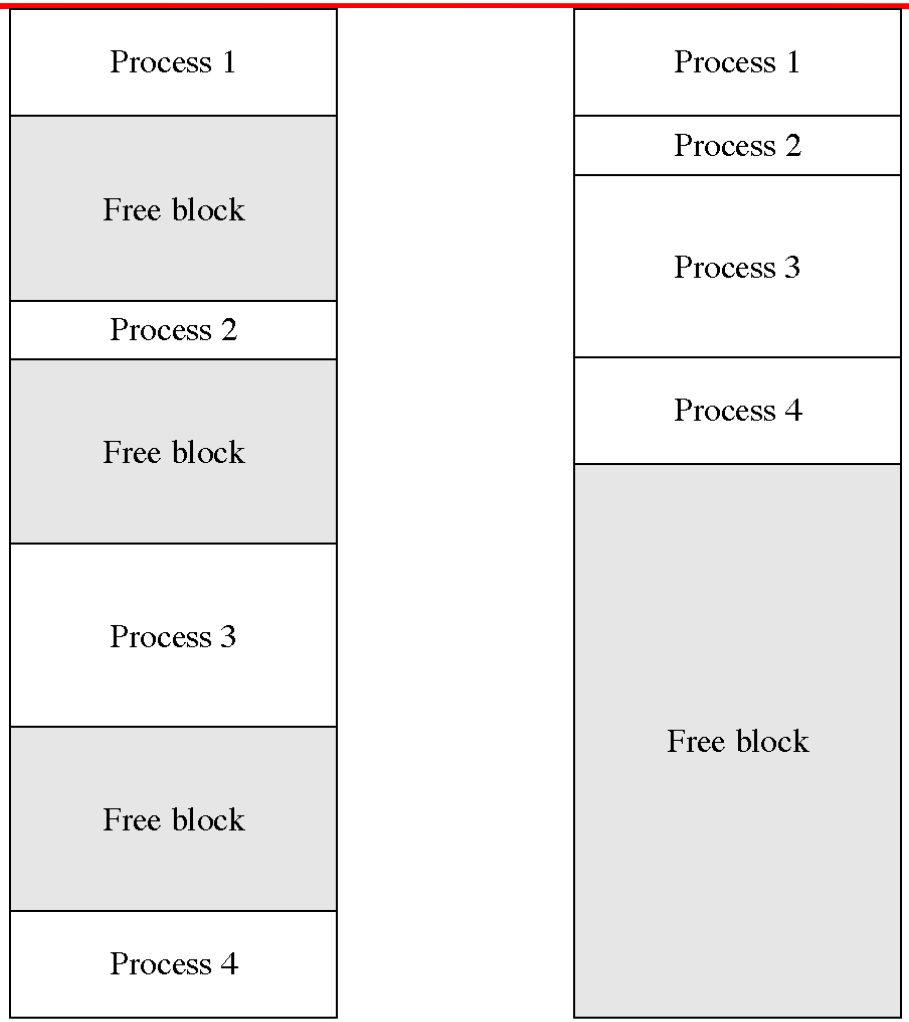


O espaço agregado é maior que a memória solicitada pelo Processo D.

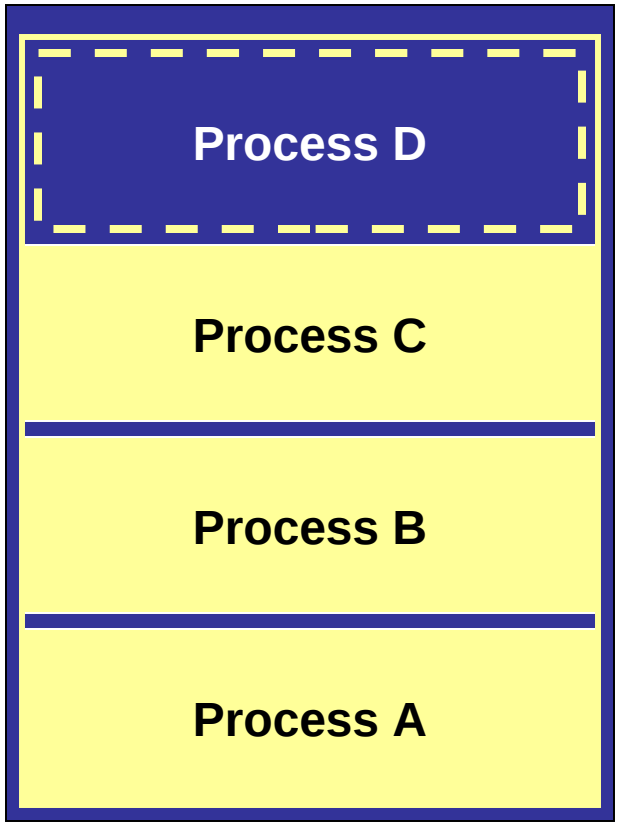
Mas, não existe buraco grande o suficiente.

Compactação de memória é necessária para juntar os processos.

Compactando memória

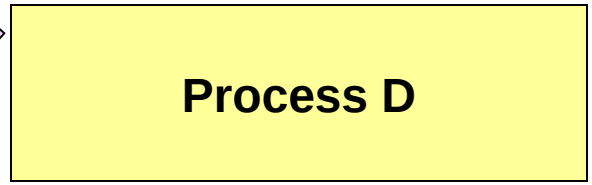


Fragmentação Externa



O processo de compactação de memória toma muito tempo.

UCP precisa mover a memória unidade-por-unidade.



Isto atrasa o chaveamento de contexto.

Fragmentação

- Sem mapeamento de memória, programas requerem memória fisicamente contígua
- Grandes blocos significa grandes fragmentos
  - e memória desperdiçada
- É preciso hardware para mapeamento da memória para resolver este problema
  - segmentos
  - páginas

Outros Problemas

- A memória alocada pode crescer?

- E se a memória necessária, x, é:

Tamanho da memória física < x < 4G - 1

Outros Problemas

- Se o sistema não permite a memória crescer, então qual é o limite fixado?
  - 4G-1 bytes?
  - A memorial física?

- Se o sistema permite memória crescer, então qual é o limite de crescimento?
  - 4G-1 bytes?
  - A memorial física?

Solução?

- Que tal isto? Eu projetei um SO que:
  - Não permite a memória crescer, e
  - o limite superior da memória alocado para cada processo é o tamanho da memória de espaço do usuário.

- …algum problema?
  - Todo programador necessita saber o tamanho da memória do sistema antes de escrever qualquer programas!

Solução?

- Que tal isto? Eu projetei um SO que:
  - Não permite a memória crescer, e
  - o limite superior da memória alocada para cada processo é 4G-1 bytes.
  - Assim, o programador esta livre para escrever qualquer programa sem saber o tamanho da memória do sistema.

- …algum problema?
  - Não tem memória disponível!

continua...