BC1518 - Sistemas Operacionais

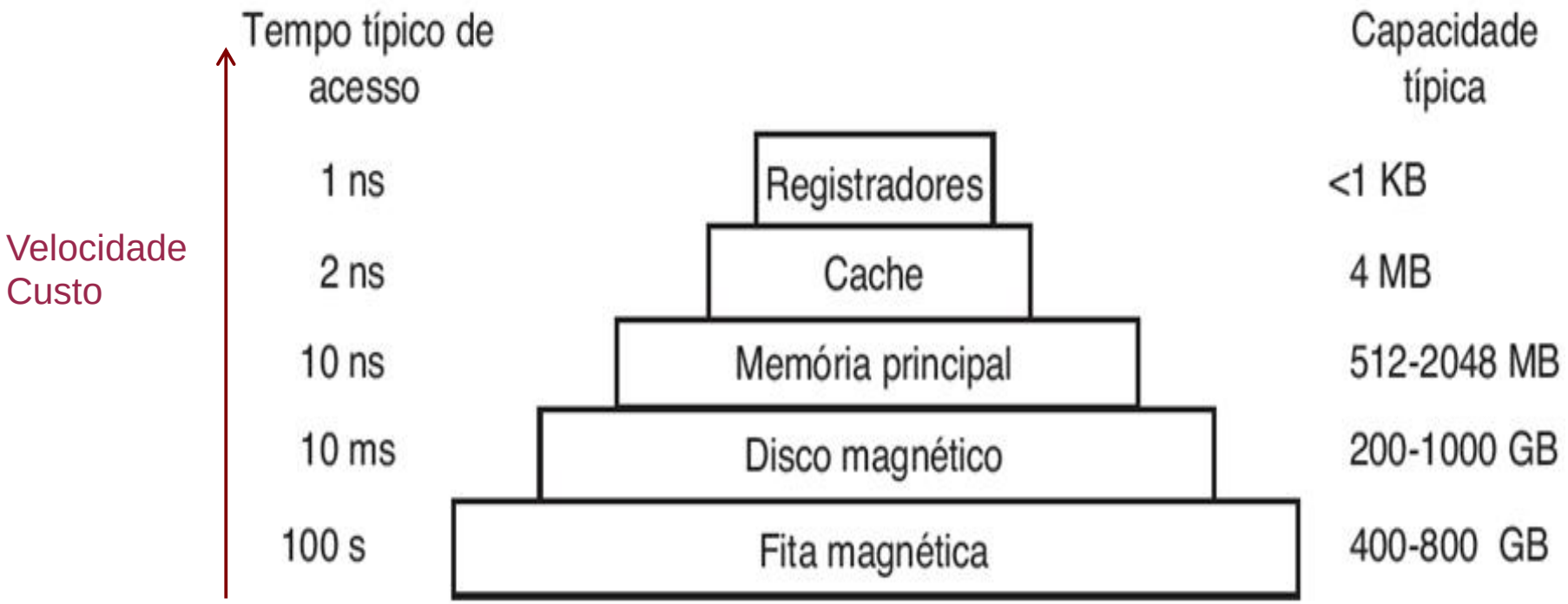
Aula 8: Gerenciamento de Memória

Material parcialmente baseado nos slides do Prof. José Artur Quilici Gonzalez]

Tópicos desta aula

- Conceitos Básicos – Gerenciamento de Memória
- Endereços lógicos e físicos
- Estratégias de Alocação de Memória
  - Partição fixa
  - Partição dinâmica
  - Paginação
  - Segmentação

Hierarquia de memória



Hierarquia de memória típica. Os números são aproximações.

[Tanenbaum]

Registradores, cache e memória principal são voláteis, enquanto que discos e fitas são nã

Gerência de memória

➢Ao longo do tempo, a memória principal sempre foi vista como um recurso escasso e caro

➢A Gerência de Memória é um dos tópicos mais importantes do projeto de um SO, se preocupa com:

- Como alocar o espaço de memória disponível aos processos
- Mantendo o maior número de processos na memória a fim de maximizar o compartilhamento da CPU e demais recursos → Multiprogramação
- Utilizando alguma estratégia de gerenciamento de memória
- E garantindo a proteção de áreas de memória ocupadas por cada processo e a área onde reside o próprio sistema, impedindo acessos indevidos

Conceitos básicos

➢Os programas em geral estão armazenados em ex.: disco como um arquivo executável binário

➢Para ser executado, um **Programa** precisa ser **alocado na memória** e **associado a um Processo**

➢O processador executa somente instruções que estão na memória principal

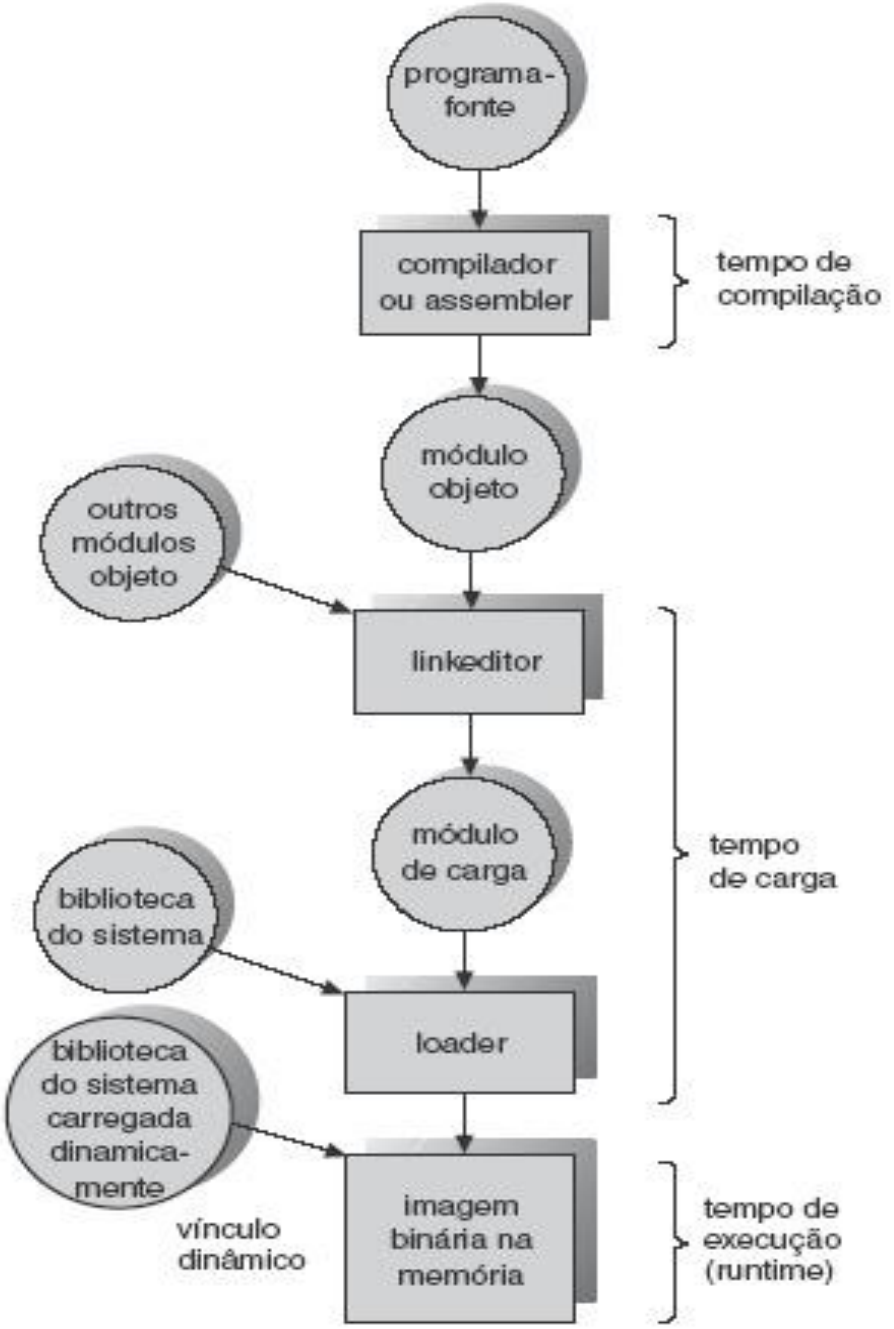
- O SO deve transferir os programas do disco para a memória principal antes de serem executados

➢**Fila de Entrada** – coleção de processos no disco esperando serem levados para a memória para execução

- Um processo é selecionado da fila e é carregado na memória
- O processo executa, acessando instruções e dados na memória
- O processo termina e o seu espaço na memória é liberado

➢Programas de usuários passam por várias etapas antes de

Processamento de um programa de usuário



➢Quando um programador escreve programas em linguagens de alto nível (C, C++, Java, etc.) utiza apenas referências a entidades abstratas ou símbolos (nomes de variáveis, funções, parâmetros,...)

➢Ou seja, o programador não precisa definir ou manipular diretamente endereços de memória

➢A associação dos trechos de código (instruções) e dados (variáveis) em endereços de memória pode ser feita em diferentes etapas: durante a compilação, durante a carga do código do processo na memória ou durante a execução

[Silberschatz]

Mapeamento de código e dados na memória

Endereçamento de instruções e dados na memória pode ocorrer em três diferentes etapas:

➢**Tempo de Compilação:** Se a posição na memória já for conhecida a *priori*, um *código de endereçamento* ***absoluto*** (endereços físicos de memória) pode ser gerado; é necessário recompilar o código se o endereço inicial mudar (Ex.: Programas no formato .COM do MS-DOS)

➢**Tempo de Carga:** Se a localização de memória não for conhecida durante o tempo de compilação, o compilador terá de gerar um *código de endereçamento* ***relocável*** (endereços relativos ao início do código do programa – ex.: 432 bytes a partir do início do módulo); a associação final dos endereços é feita durante a carga do código na memória pelo carregador (*loader*) – se o endereço inicial na memória mudar, é preciso apenas recarregar o código a partir do novo endereço

➢**Tempo de Execução:** Se o processo puder ser movido (na memória) durante a sua execução, a associação de endereços é adiada para o momento da execução do processo; requer um dispositivo de hardware especial para que esse esquema funcione; é a abordagem usada nos computadores atuais

Endereços lógicos e físicos

➢**Endereços Lógicos** – gerados pela CPU à medida que executa instruções do programa; também conhecidos como *endereços virtuais*; o conjunto de todos os endereços lógicos gerados por um programa forma o espaço de endereços lógicos

➢**Endereços Físicos** – endereços que a unidade de memória trabalha; o conjunto de todos os endereços físicos correspondentes aos endereços lógicos de um processo corresponde ao espaço de endereços físicos

Endereços Lógico e Físico são **iguais** nos métodos de resolução de endereços em **tempo de compilação** e **tempo de carga**; endereços lógicos e físicos **diferem** no esquema de resolução de endereços em **tempo de execução**

Unidade de Gerenciamento de Memória
Memory-Management Unit (MMU)

- Dispositivo de hardware que faz o **mapeamento de endereços lógicos** (do programa do usuário) **em endereços físicos na memória**
- O programa do usuário lida sempre com **endereços *lógicos***; ele nunca vê os **endereços físicos *reais***
-

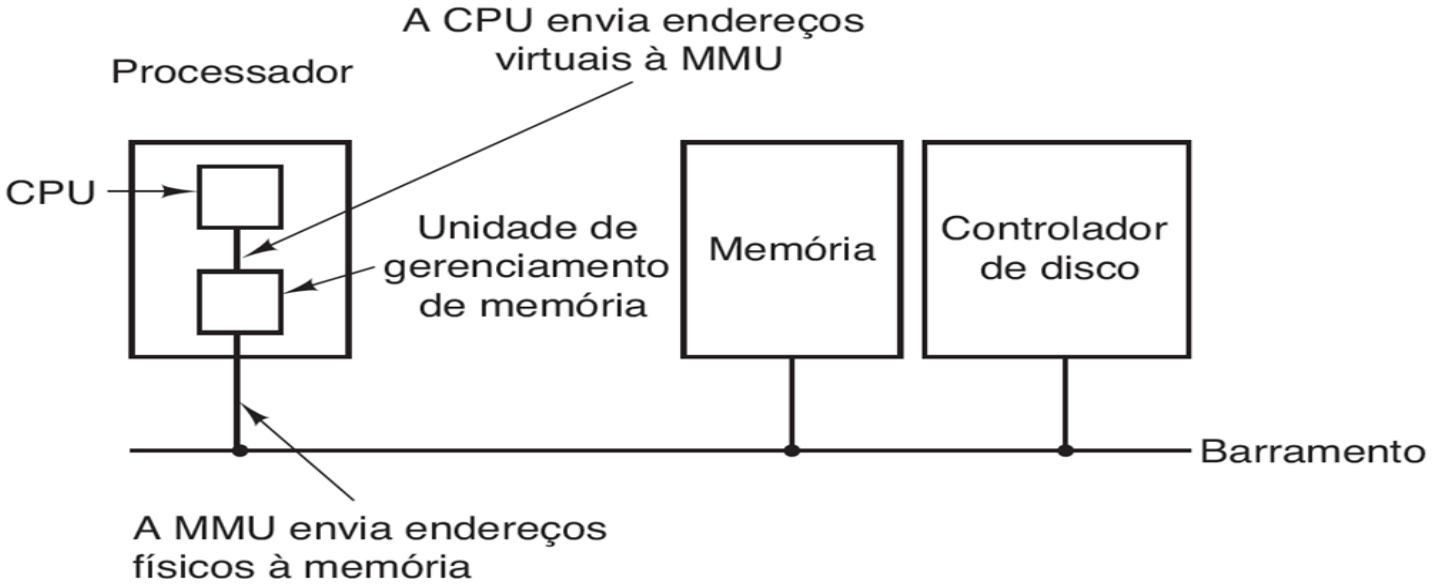
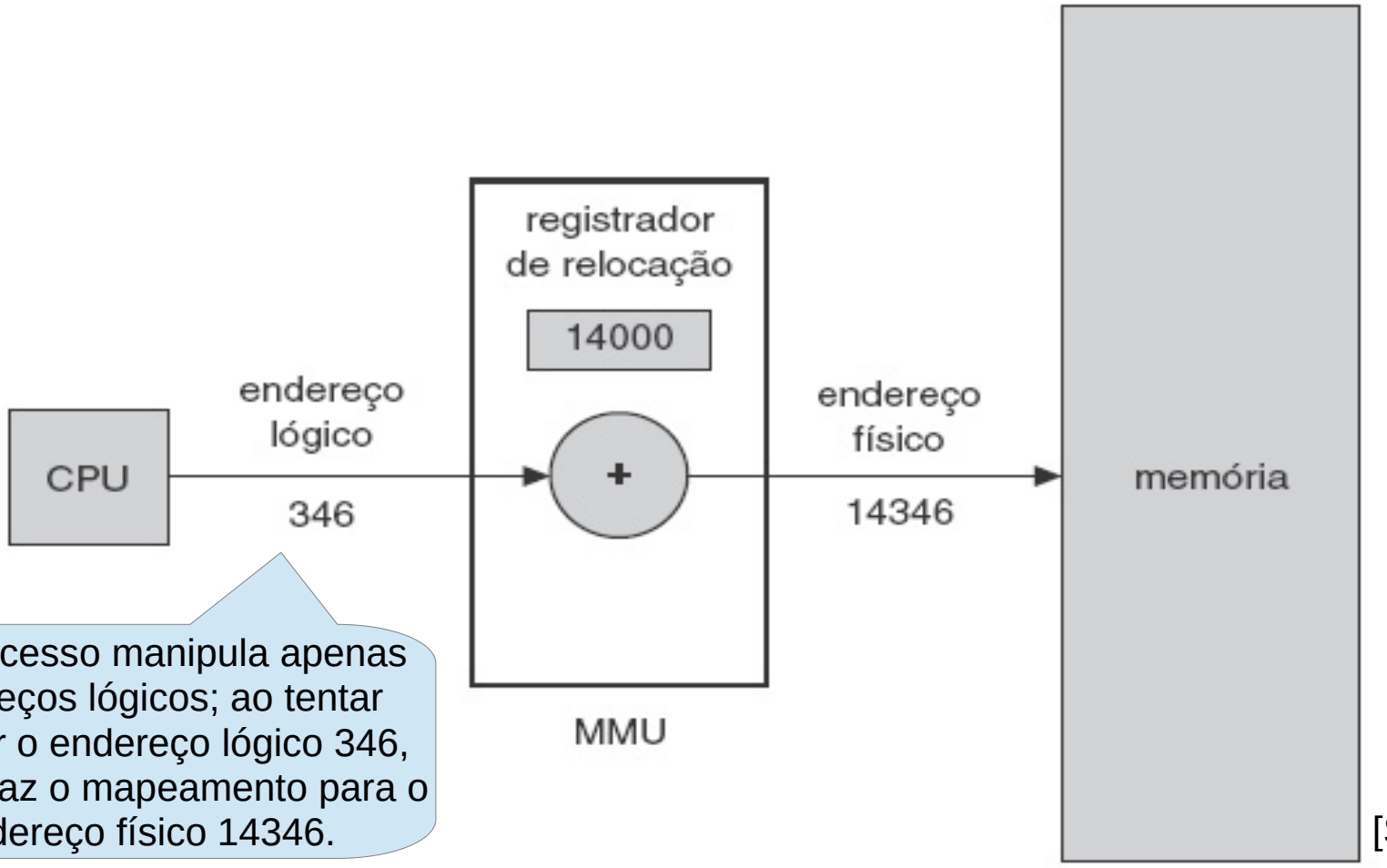


Figura 3.8 A posição e a função da MMU. Aqui a MMU é mostrada como parte do chip da CPU (processador) porque isso é comum atualmente. Contudo, em termos lógicos, poderia ser um chip separado, como ocorria no passado. [Tanenbaum]

Relocação dinâmica usando registrador de relocação



[Silberschatz]

- No esquema MMU simples, é utilizado um registrador de relocação (indica o menor endereço
- O valor do **registrador de relocação** é adicionado a todo endereço gerado por um processo

Como melhorar a utilização da memória?

➢Algumas técnicas para melhorar a utilização do espaço de memória limitado:

- Carregamento dinâmico
- Ligação dinâmica
- Overlays (sobreposição)
- Troca de processos (*swapping*)

Carregamento dinâmico

- Com o **Carregamento Dinâmico**, uma rotina só é carregada quando é chamada
- Melhor utilização do espaço da memória; uma rotina não-utilizada, nunca é carregada
- Útil quando há necessidade de grande quantidade de código para lidar com casos que ocorrem com pouca frequência (como rotinas de erro)
- Não requer suporte especial do sistema operacional; é **responsabilidade do programador** escrever seus programas de modo a aproveitar esse método (SOs podem fornecer rotinas de biblioteca para implementar o carregamento dinâmico)

Ligação dinâmica

- Alguns SOs oferecem suporte apenas à **ligação estática**
  - As bibliotecas de linguagem do sistema são tratadas como qualquer outro módulo objeto, e são combinadas pelo loader na imagem do programa binário
  - Cada programa precisa ter uma cópia de sua biblioteca de linguagem (ou pelo menos as rotinas referenciadas pelo programa) → desperdício de espaço em disco e memória principal
- Com a **ligação dinâmica**, a **ligação** de um programa a uma biblioteca é adiada até o **tempo de execução**
  - Um pequeno trecho de código, ***stub***, é usado para localizar a rotina de biblioteca apropriada residente na memória
  - Se a rotina não estiver na memória, ela é carregada
  - O *stub* substitui a si mesmo pelo endereço da rotina e a executa (esse endereço fica na memória para posterior utilização)
  - Todos os processos que utilizam uma biblioteca de linguagem executam a mesma cópia do código da biblioteca
  - Diferentemente da carga dinâmica, a **Ligação Dinâmica** geralmente requer ajuda do SO
- O SO precisa checar se uma rotina está carregada no espaço de memória de determinado processo (no caso de os processos estarem protegidos uns dos

Overlays (sobreposições)

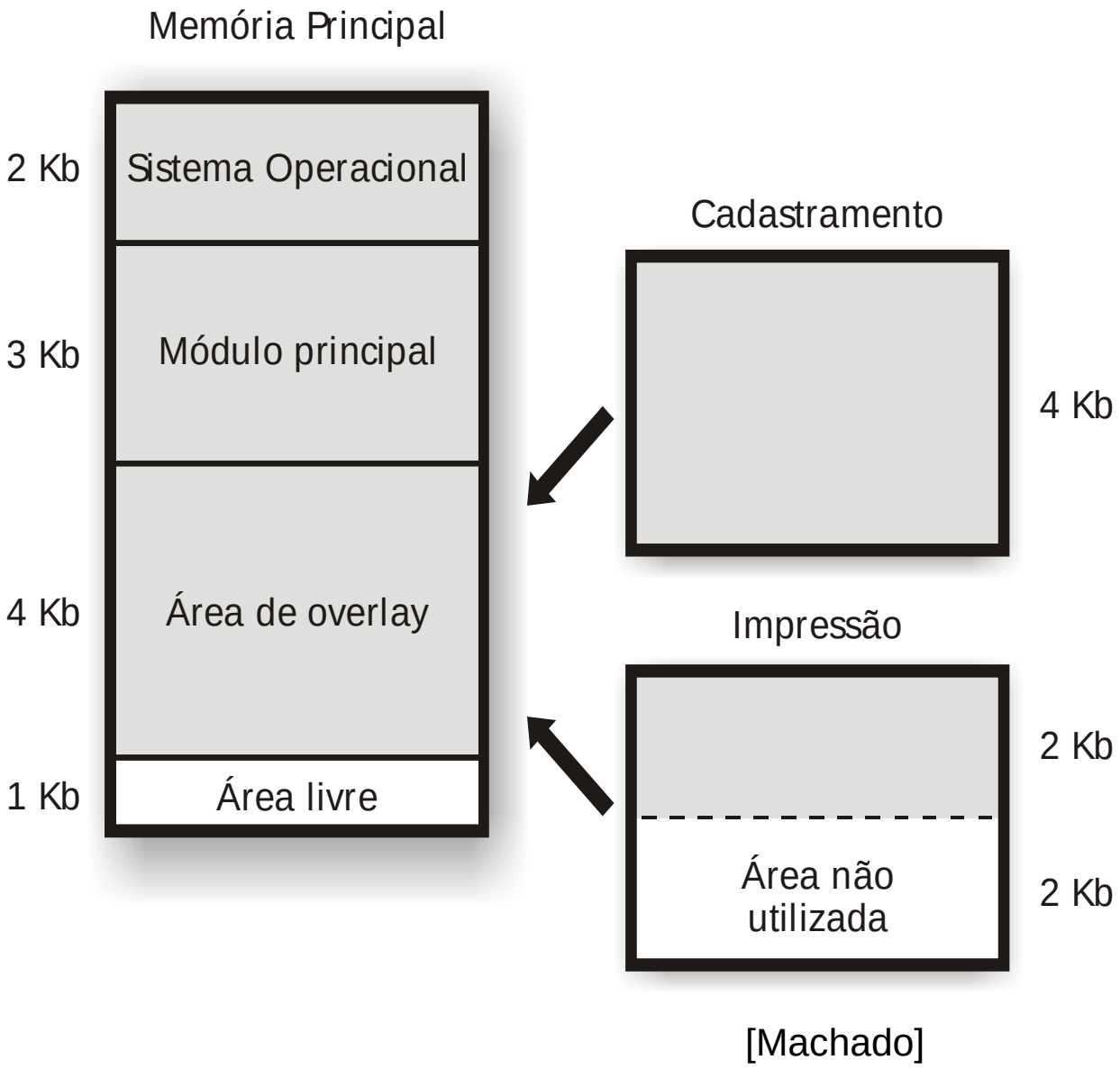
➢Uma técnica que pode ser utilizada quando o tamanho de um processo é maior do que a quantidade de memória física disponível

➢O programa é dividido em módulos de modo que é possível a execução independente de cada módulo utilizando a mesma área de memória (que é sobreposta)

➢Mantém na memória apenas as instruções e dados que são necessários em determinado momento (os outros módulos ficam no disco)

➢**Implementada pelo usuário**, sem necessidade de suporte especial do sistema operacional; o projeto da estrutura do *overlay* é complexo (o programador precisa fazer uma análise minuciosa para descobrir quais rotinas são excludentes entre si)

Técnica de Overlay

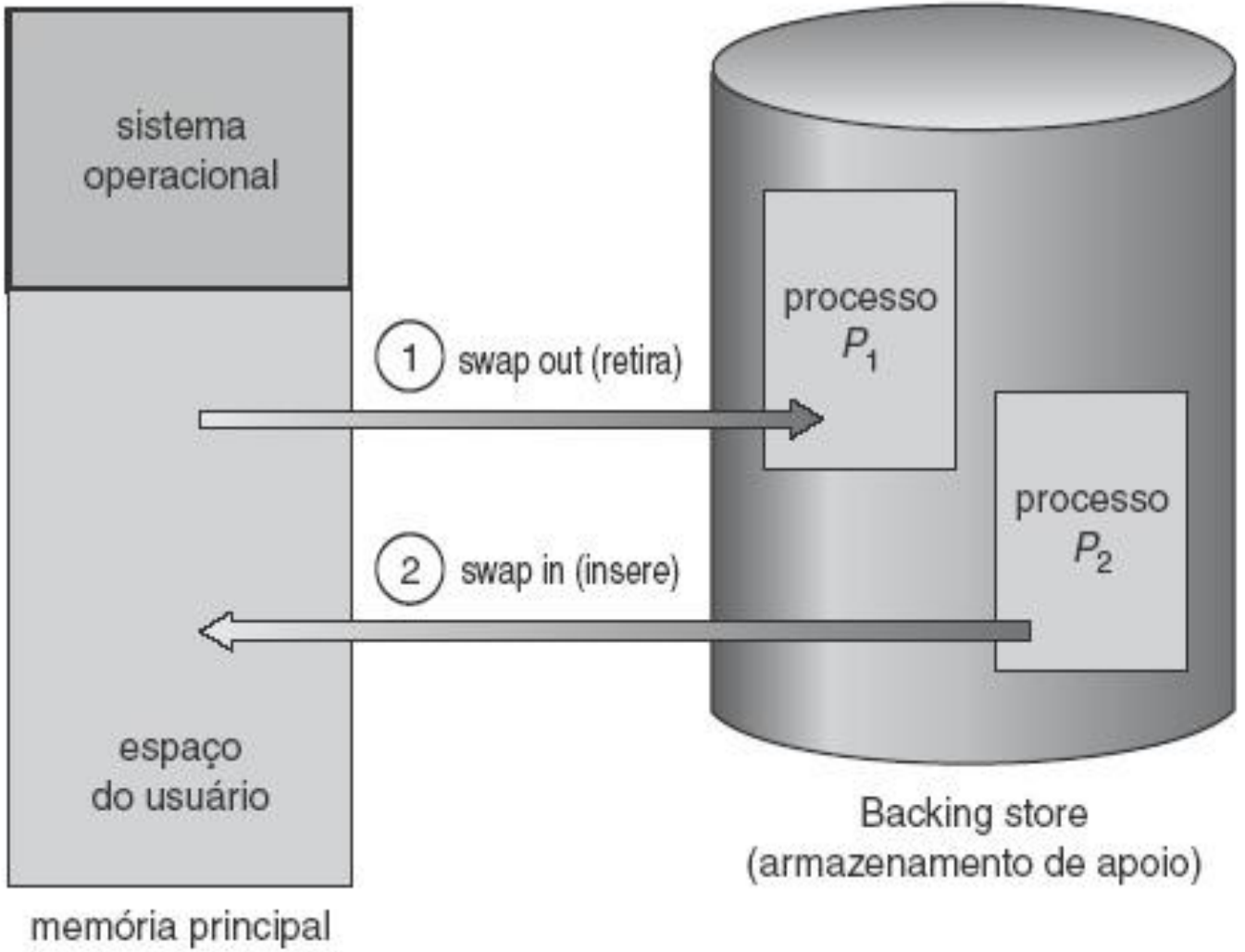


- Considere um programa com 3 módulos: Principal, Cadastramento e Impressão, sendo que a memória disponível é insuficiente para armazenar todo o programa
- A técnica de *overlay* utiliza uma área de memória comum onde os módulos Cadastramento e Impressão poderão compartilhar (área de *overlay*)
- Quando um dos módulos é referenciado pelo módulo principal, o módulo é carregado do disco para a área de *overlay*

Troca de processos – Swapping

- Técnica para solucionar o **problema da insuficiência** de memória
- Um processo pode ser removido temporariamente da memória para um armazenamento auxiliar (disco), e depois é retornado à memória para continuar sua execução
  - Ex.: Escalonamento de CPU Round Robin: Quando o *quantum* de tempo expira, o processo que estava executando é retirado (*swap-out*) da memória pelo gerenciador de memória e um outro processo é colocado (*swap-in*) no espaço de memória liberado
  - Ex.: Escalonamento por prioridade: variante de *swapping* usada por algoritmos de escalonamento baseados em prioridade; processo com prioridade menor é removido para que um processo com prioridade maior possa ser carregado e executado (*Roll out, roll in*)
  - A maior parte do tempo de troca é tempo de transferência
    - O tempo de transferência total é **diretamente proporcional** à **quantidade de memória trocada**

Visão esquemática do swapping



[Silberschatz]

Troca de processos

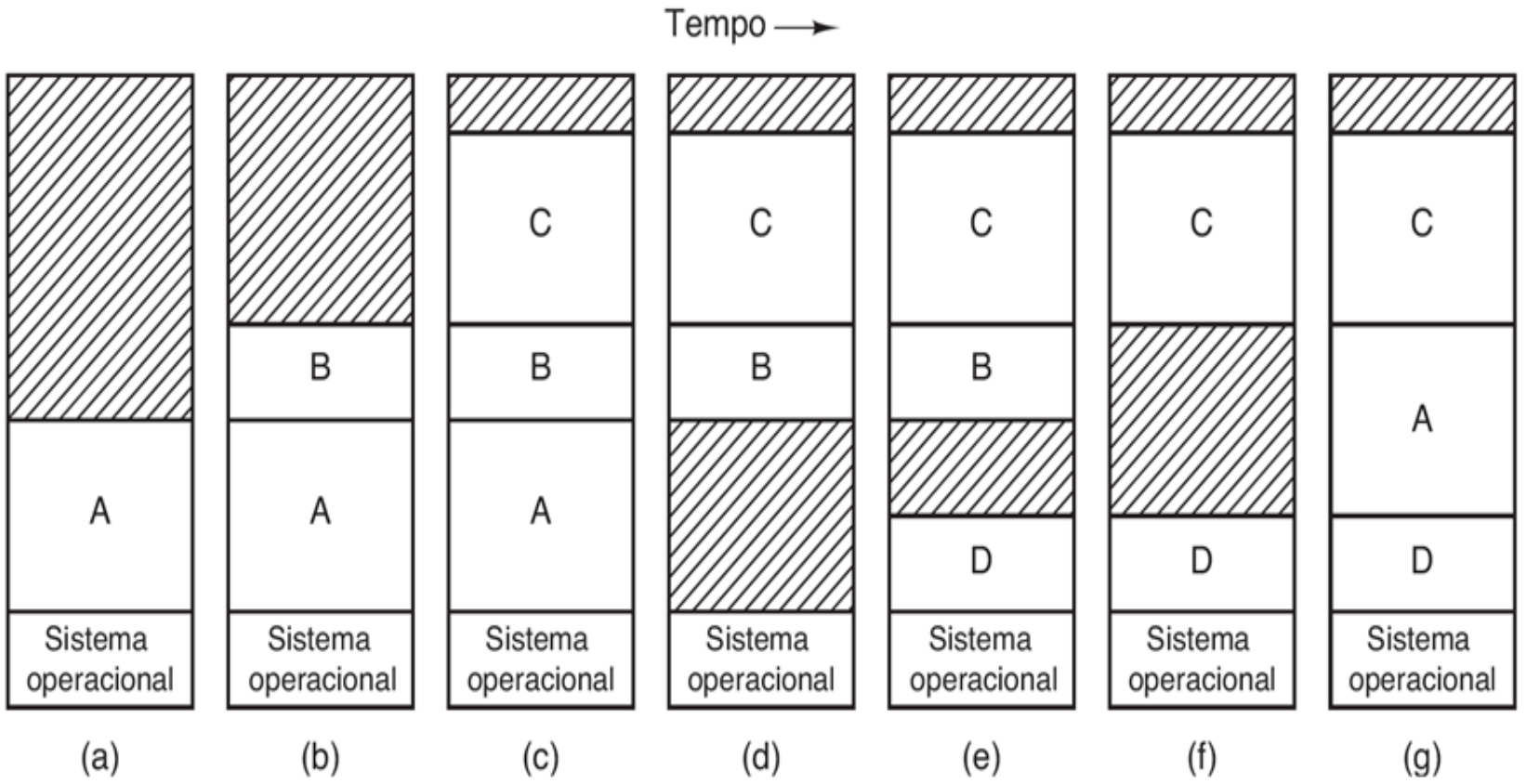
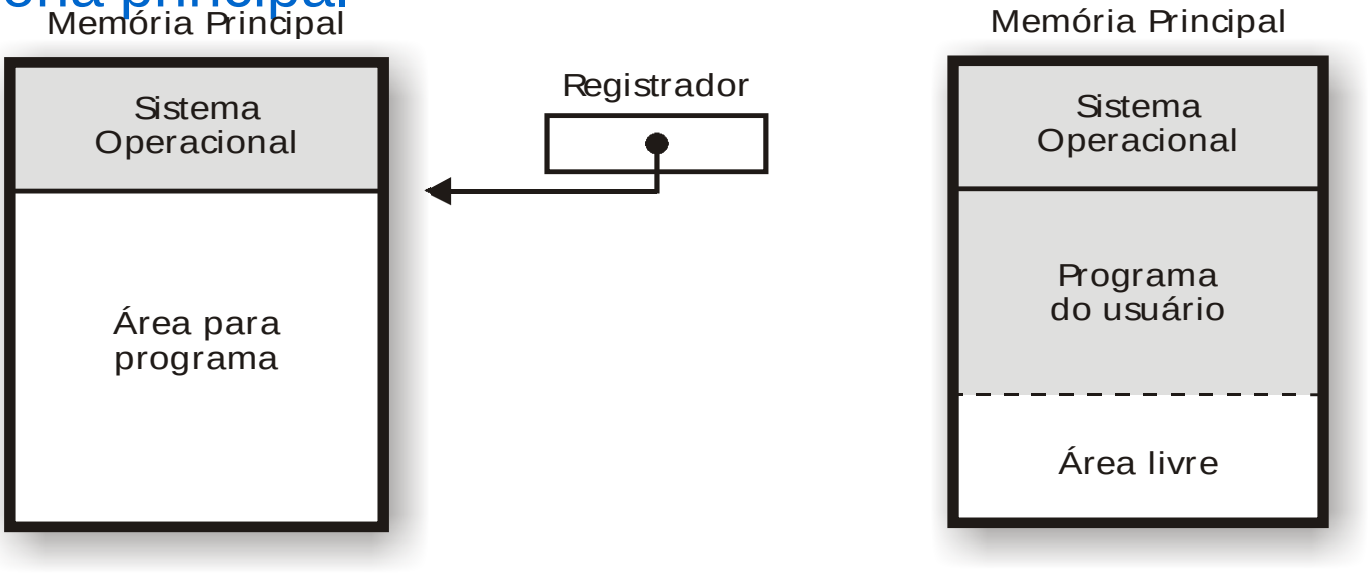


Figura 3.4 Alterações na alocação de memória à medida que processos entram e saem dela. As regiões sombreadas correspondem a regiões da memória não utilizadas naquele instante. [Tanenbaum]

Estratégias de alocação de memória

- Sistemas monoprogramáveis (apenas um processo é carregado na memória para execução)
  - Gerenciamento de memória é simples: a memória principal é dividida em duas áreas: uma para o SO e outra para o programa do usuário
  - Usuários tem acesso a toda a memória principal, alguns sistemas implementam proteção através de um registrador para delimitar as áreas do SO e do usuário
  - Não permite o uso eficiente dos recursos; subutilização da memória principal



[Machado]

Estratégias de alocação de memória

➢Sistemas multiprogramáveis (vários processos são carregados na memória para execução)

- O espaço de memória destinado aos processos deve ser dividido entre eles de forma eficiente e flexível

- Principais estratégias de alocação da memória física:
  - Partições fixas (estáticas)
  - Partições dinâmicas (variáveis)
  - Paginação
  - Segmentação

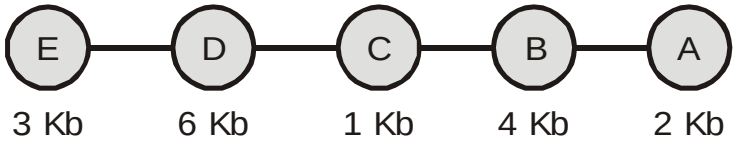
Partições fixas

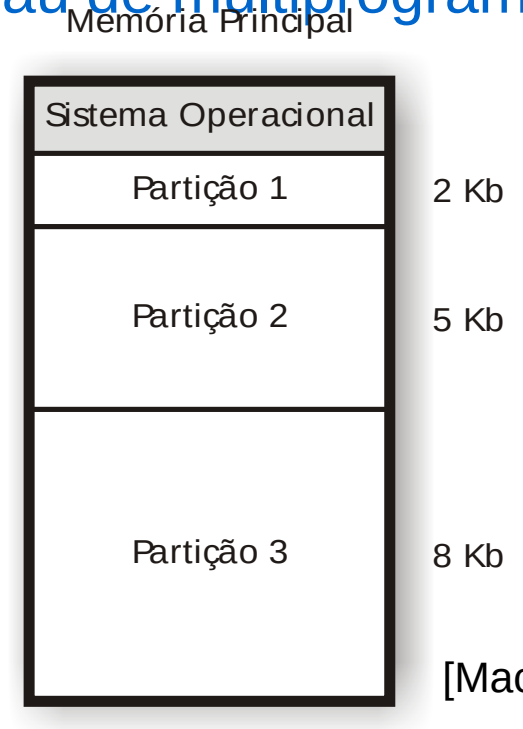
- Esquema mais simples de alocação de memória: dividir a memória em várias partições de tamanho fixo, de mesmo tamanho ou não
- Em cada partição pode ser carregado um processo
- **Tabela de partições:** controla quais partições estão ocupadas, mantendo informações como o endereço inicial de cada partição, seu tamanho e se está em uso ou não
- O número de processos na memória (grau de multiprogramação) é limitado ao número de partições

Tabela de partições

Partição	Tamanho	Livre
1	2 Kb	Sim
2	5 Kb	Sim
3	8 Kb	Sim

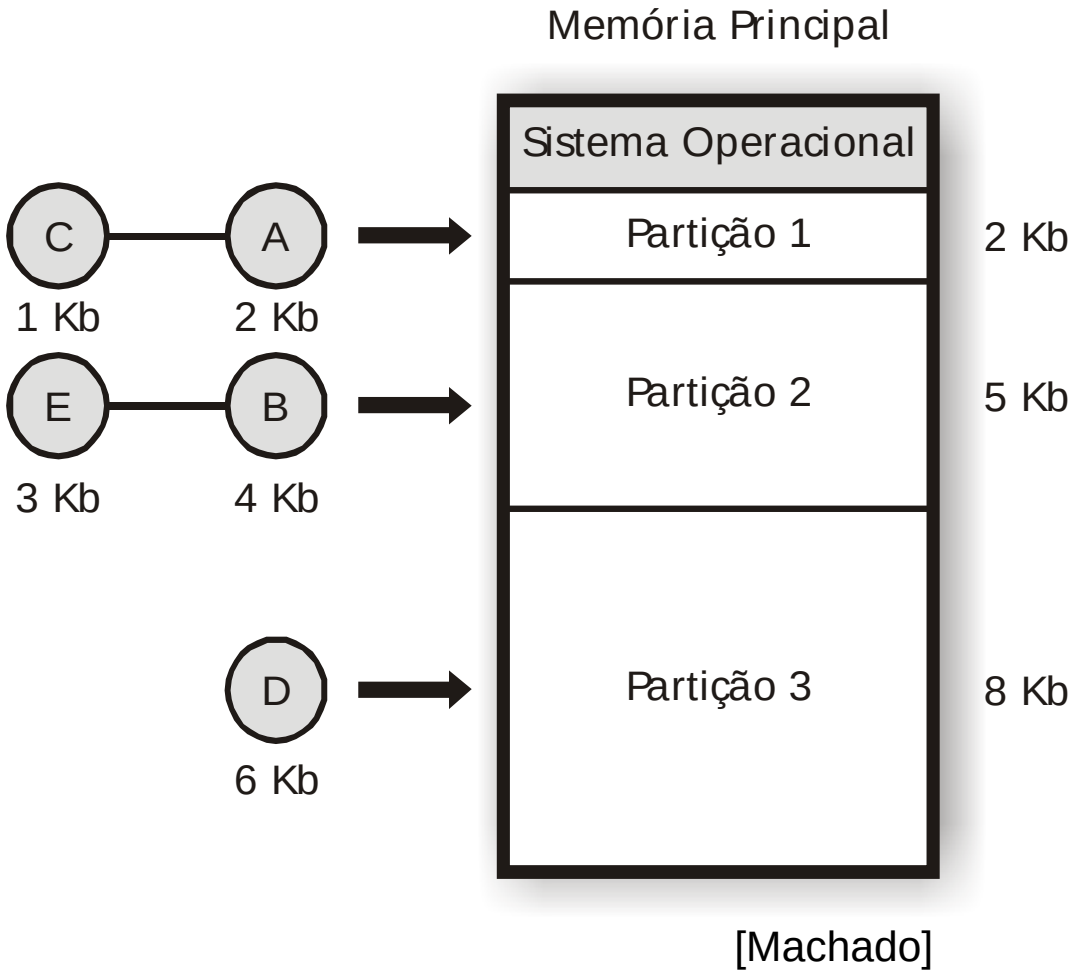
Programas a serem executados:





[Machado]

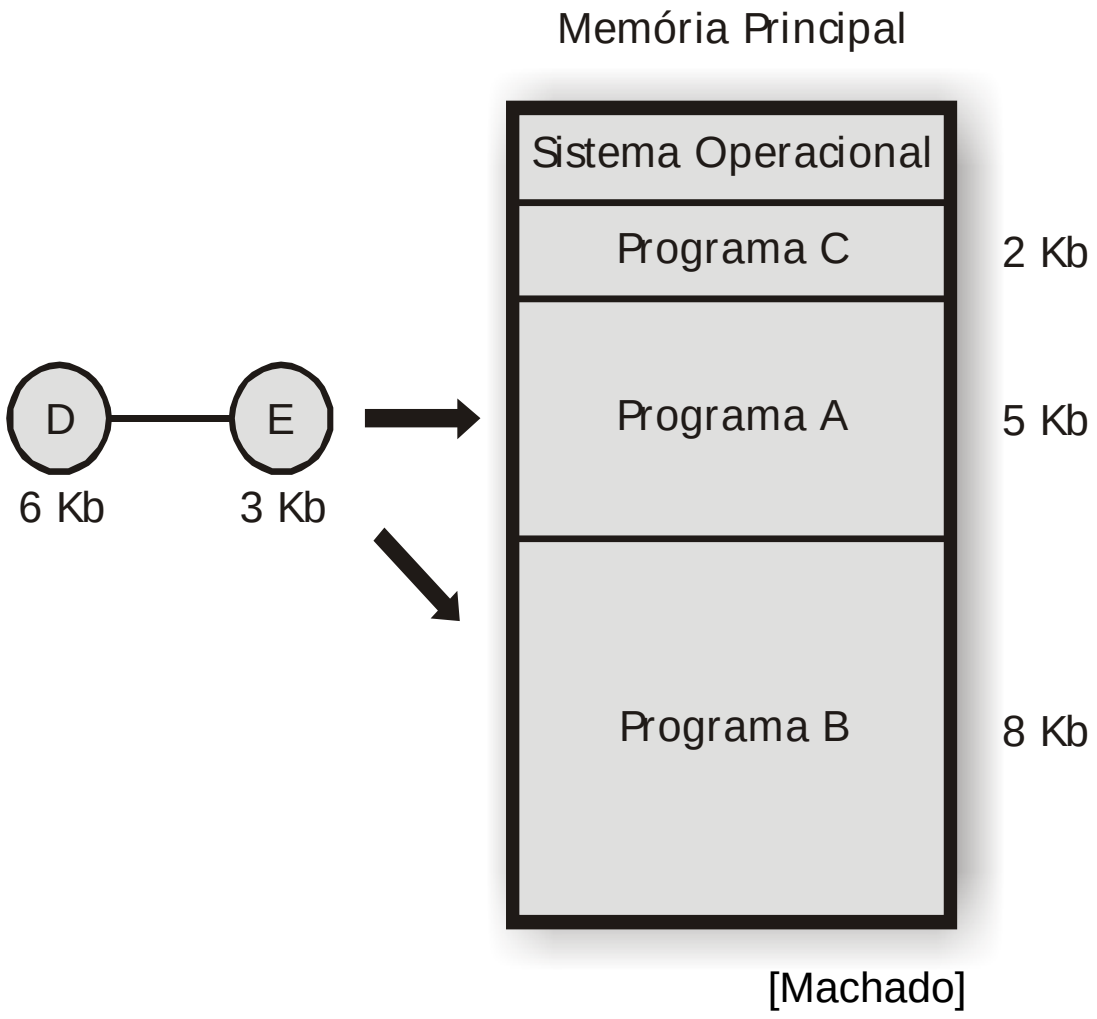
Partições fixas



[Machado]

- Alocação estática absoluta
  - No início os compiladores geravam apenas código absoluto (endereços físicos) e dessa forma, os programas só podiam ser carregados e executados em uma partição específica, mesmo que houvessem outras disponíveis

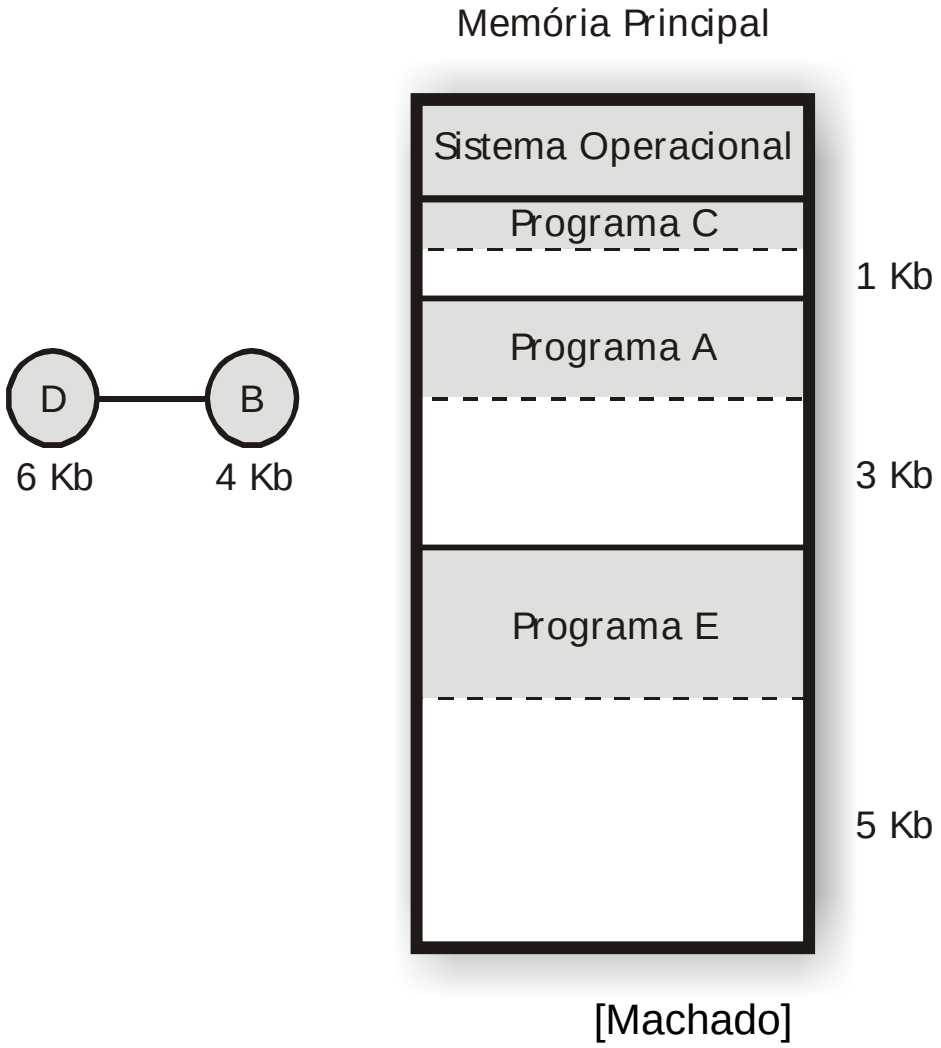
Partições fixas



[Machado]

- Alocação estática relocável
  - O compilador gera código relocável
    - Todas as referências a endereços no programa são relativas ao início do código e não a endereços físicos da memória
    - Quando o programa é carregado o *loader* (carregador) calcula todos os endereços a partir da posição inicial onde o programa foi alocado
    - O programa pode ser carregado e executado em qualquer partição cujo tamanho seja grande o suficiente para acomodar o processo

Fragmentação interna

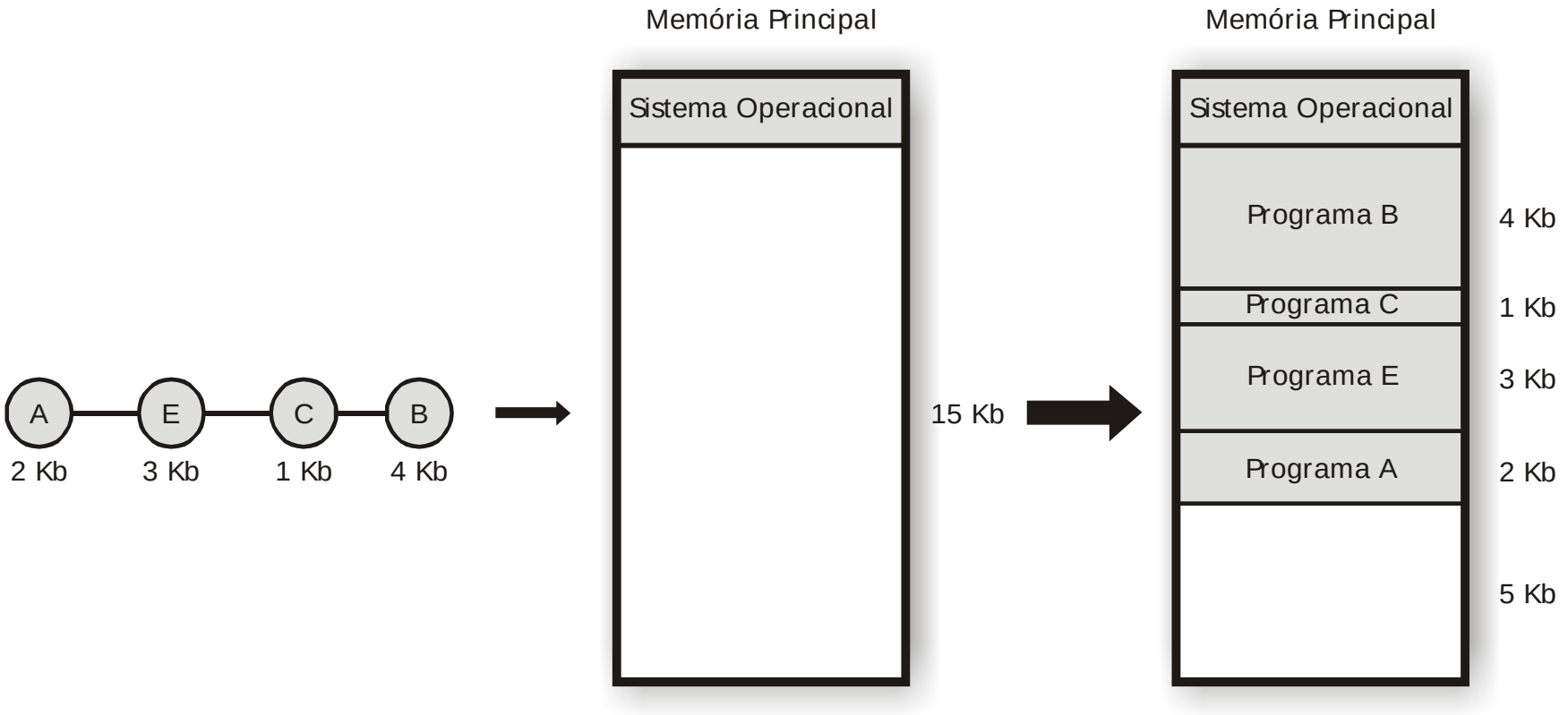


[Machado]

- Um problema da alocação utilizando partições de tamanho fixo é a fragmentação interna
- Os processos normalmente não preenchem as partições onde são carregados, deixando áreas livres (ou *buracos*)
- Na figura, os processos C, A e E não ocupam integralmente as partições onde foram alocados, deixando 1KB, 3 KB e 5 KB de áreas livres, respectivamente
- Um outro problema com a partição fixa é que se houver algum processo maior do que a maior partição, este não poderá ser carregado na memória, mesmo que

Partições dinâmicas (ou variáveis)

➢Em vez de partições fixas, neste esquema o tamanho da partição é definido de acordo com a demanda do processo (cada processo utiliza apenas o espaço de memória necessário)



[Machado]

Alocação de espaço de memória

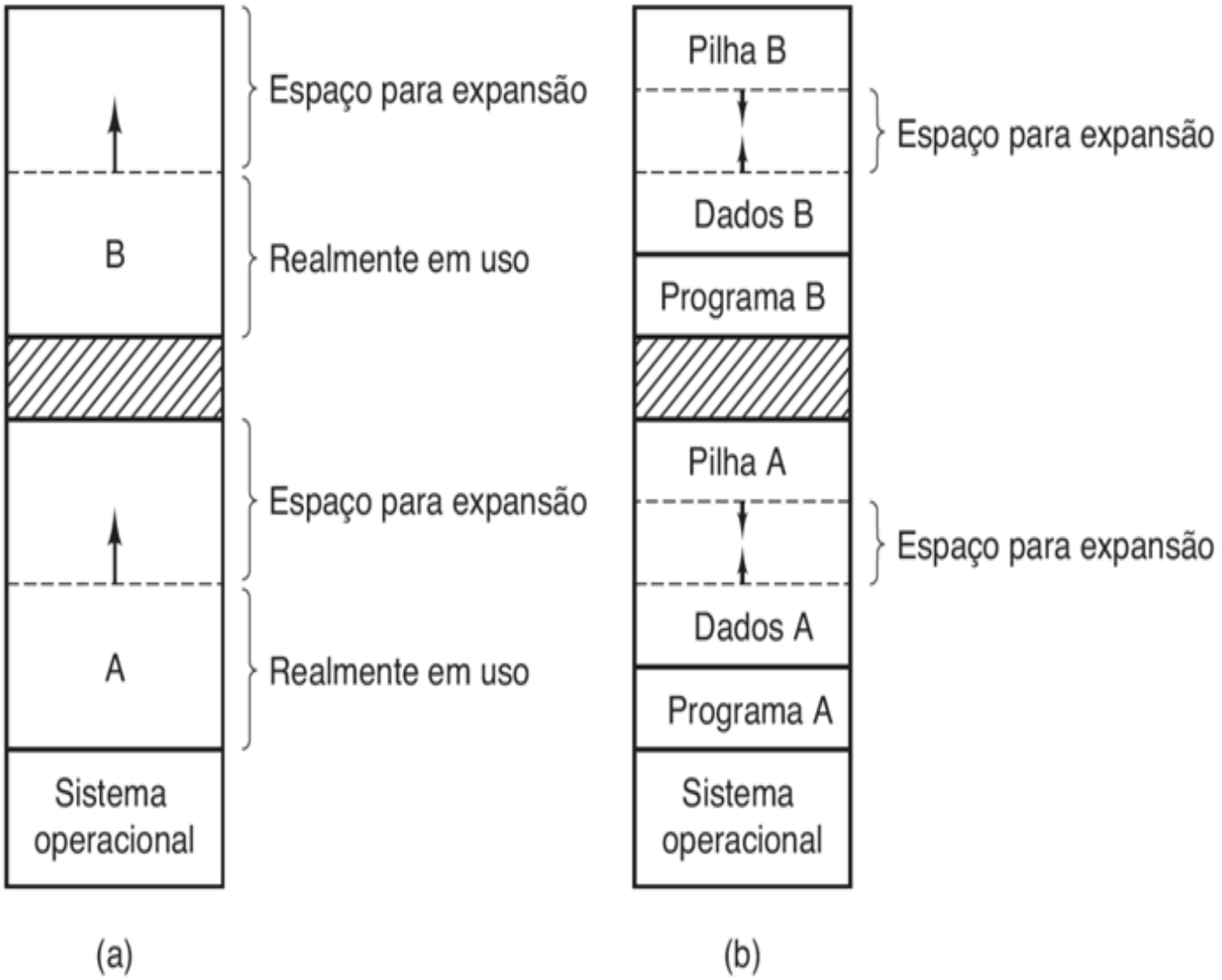
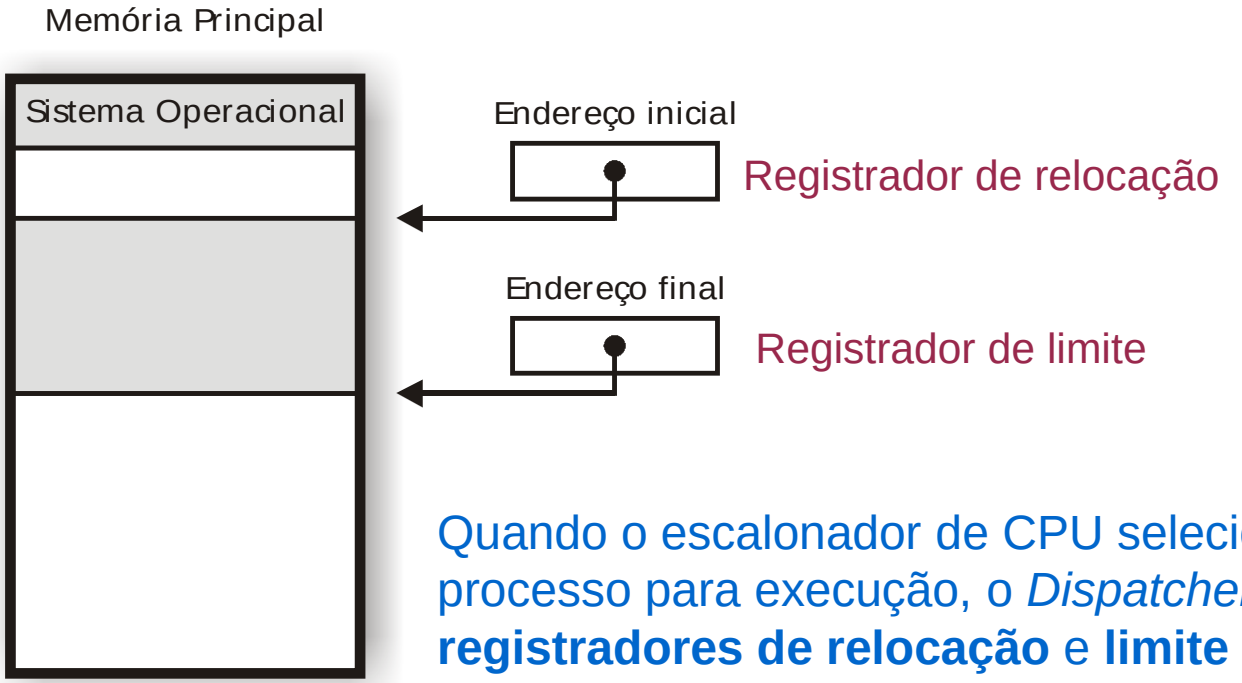


Figura 3.5 (a) Alocação de espaço para um segmento de dados em expansão. (b) Alocação de espaço para uma pilha e um segmento de dados em crescimento. [Tanenbaum]

Partições dinâmicas

- Proteção
  - Esquema de registrador de relocação usado para proteger os processos de usuários uns dos outros, e também para proteger os códigos e dados do sistema operacional
  - **Registrador de relocação** contém o **endereço inicial da partição ativa**; o **Registrador de limite** contém o **tamanho em bytes** da partição



[Machado]

Quando o escalonador de CPU seleciona um processo para execução, o *Dispatcher* carrega os **registradores de relocação** e **limite** com os valores corretos como parte da troca de contexto

Mapeamento de endereços lógicos - físicos

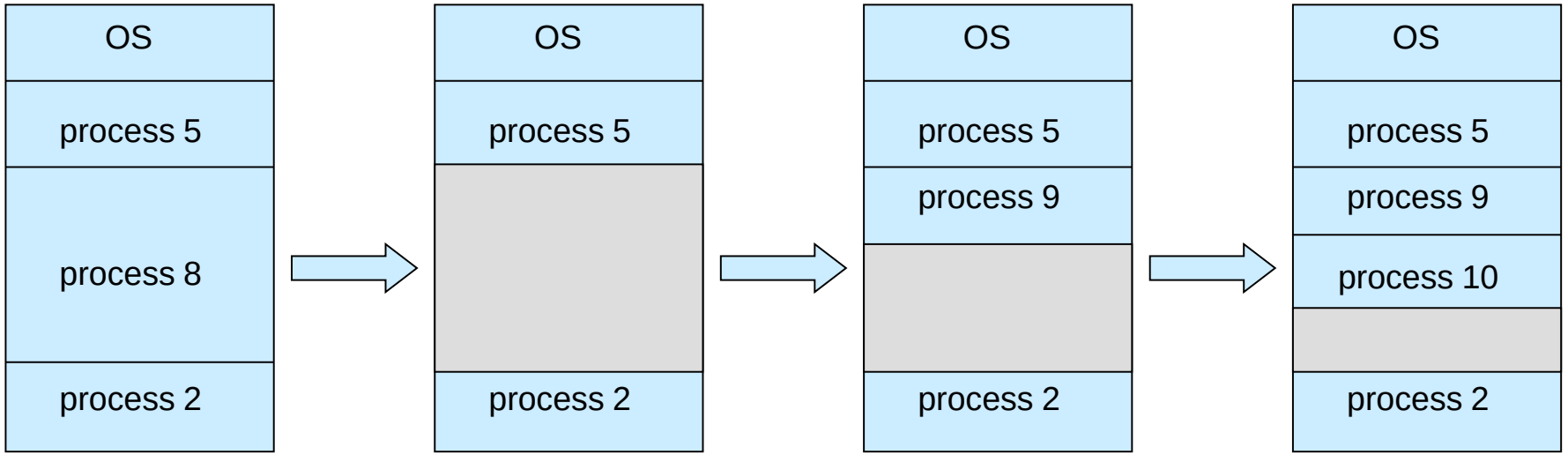
[Silberschatz]

- No mapeamento de endereços lógicos em físicos (na MMU):
  - Caso o processo tente acessar uma posição de memória fora dos limites definidos pelos registradores é gerada uma interrupção para a CPU, indicando o endereço inválido, a CPU interrompe o que está fazendo e executa a rotina de interrupção

Partições dinâmicas

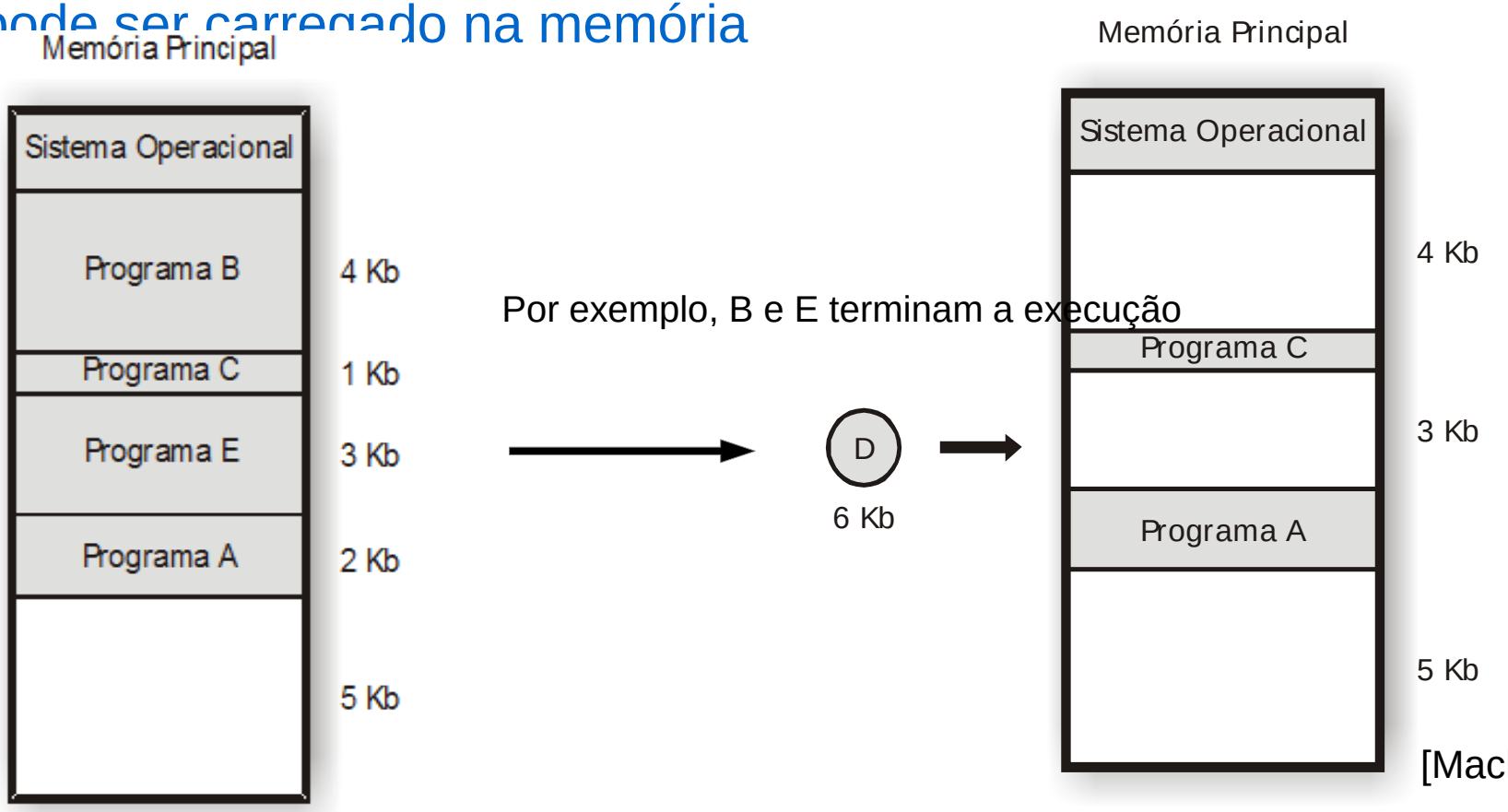
- Alocação com Partições Dinâmicas
  - Área livre ou *buraco* – bloco de memória disponível; em geral, áreas livres de vários tamanhos estão dispersos na memória
  - Quando um processo chega, é alocada memória em área livre com tamanho grande o suficiente para acomodá-lo
  - O sistema operacional mantém informações sobre:

a) partições alocadas b) partições livres (buracos)



Fragmentação externa

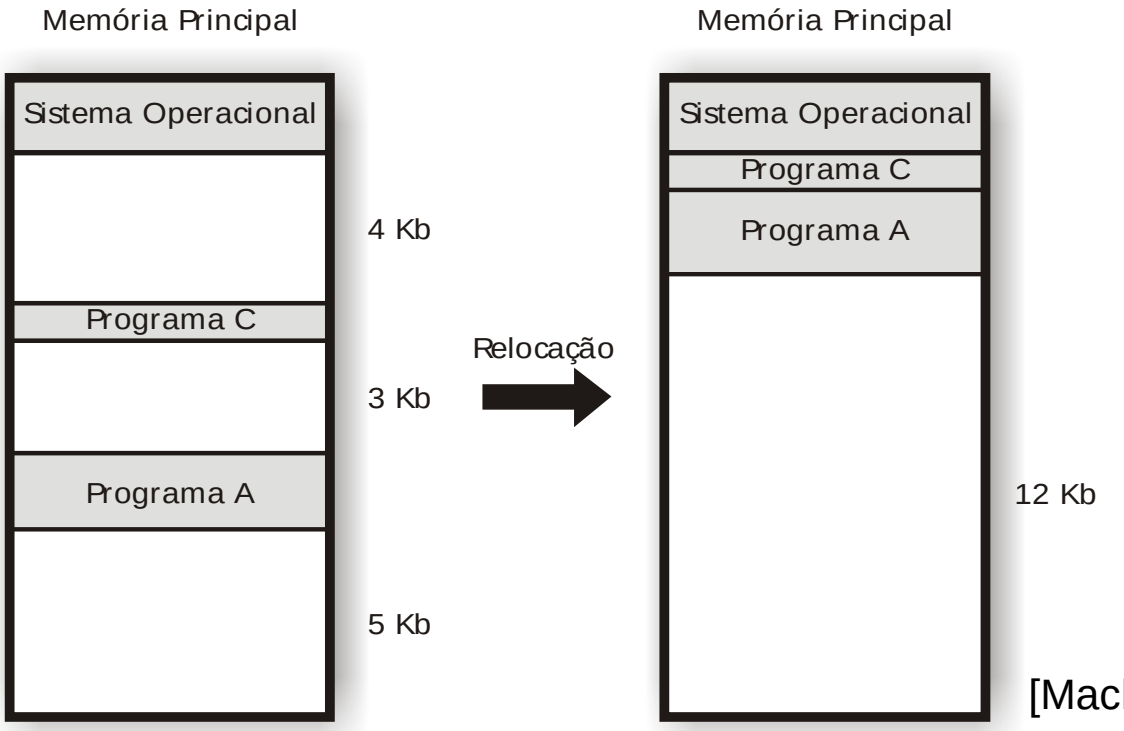
- O uso de partições dinâmicas evita o problema de fragmentação interna, mas causa um outro problema: fragmentação externa
  - A entrada e saída de processos na memória cria vários espaços de memória livre (buracos)
  - Problema: embora exista espaço de memória total suficiente para um novo processo, esse espaço não é contíguo e o processo não pode ser carregado na memória



[Machado]

Fragmentação externa

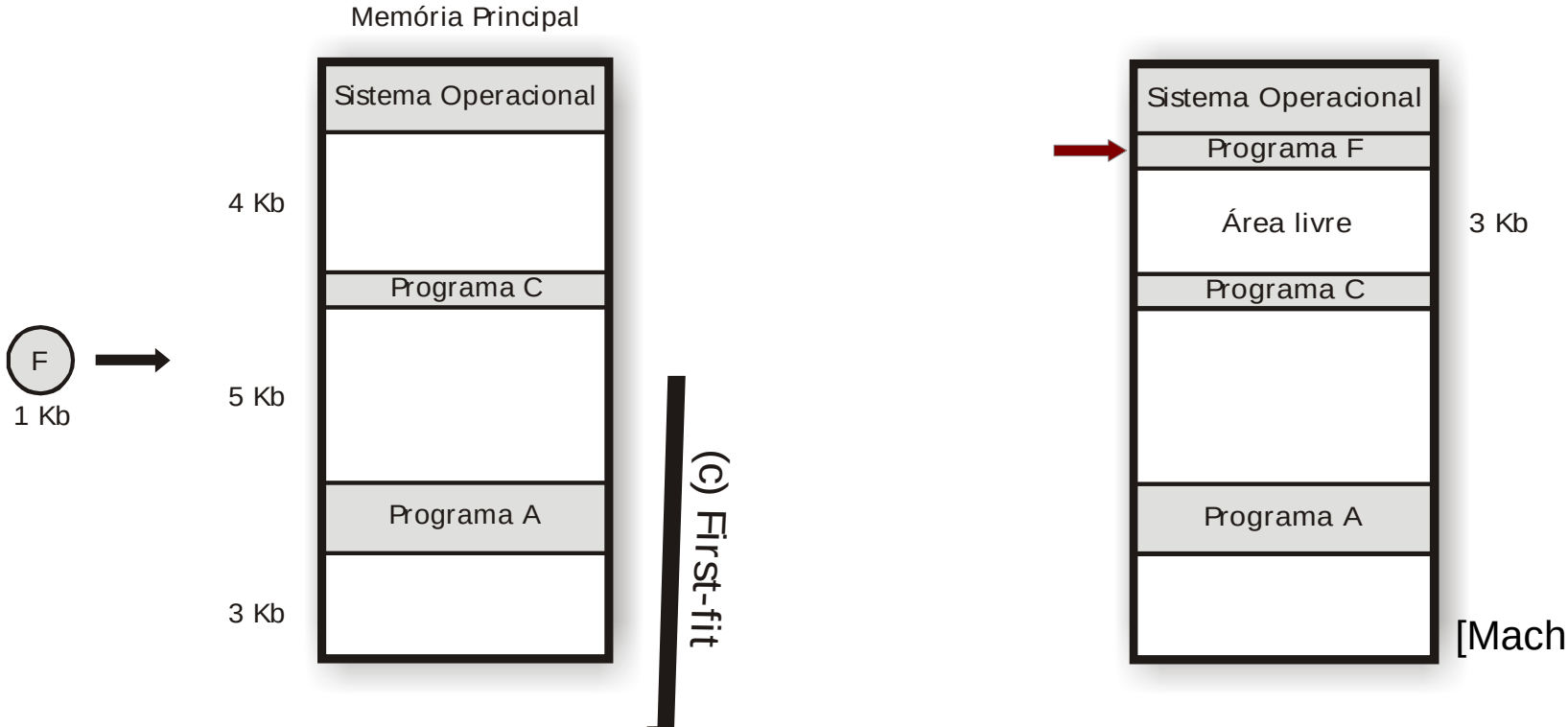
- Uma solução para a fragmentação externa é a **compactação**
  - Reorganizar o conteúdo da memória para reunir toda área de memória livre em um grande bloco
  - A compactação *só* será possível se a relocação for dinâmica e for feita em tempo de execução (requer mover o programas e dados e alterar o registrador de base (relocação)
- Algoritmos são complexos e dispendiosos (consumo de recursos)



[Machado]

Estratégias de alocação de partição (em áreas livres da memória)

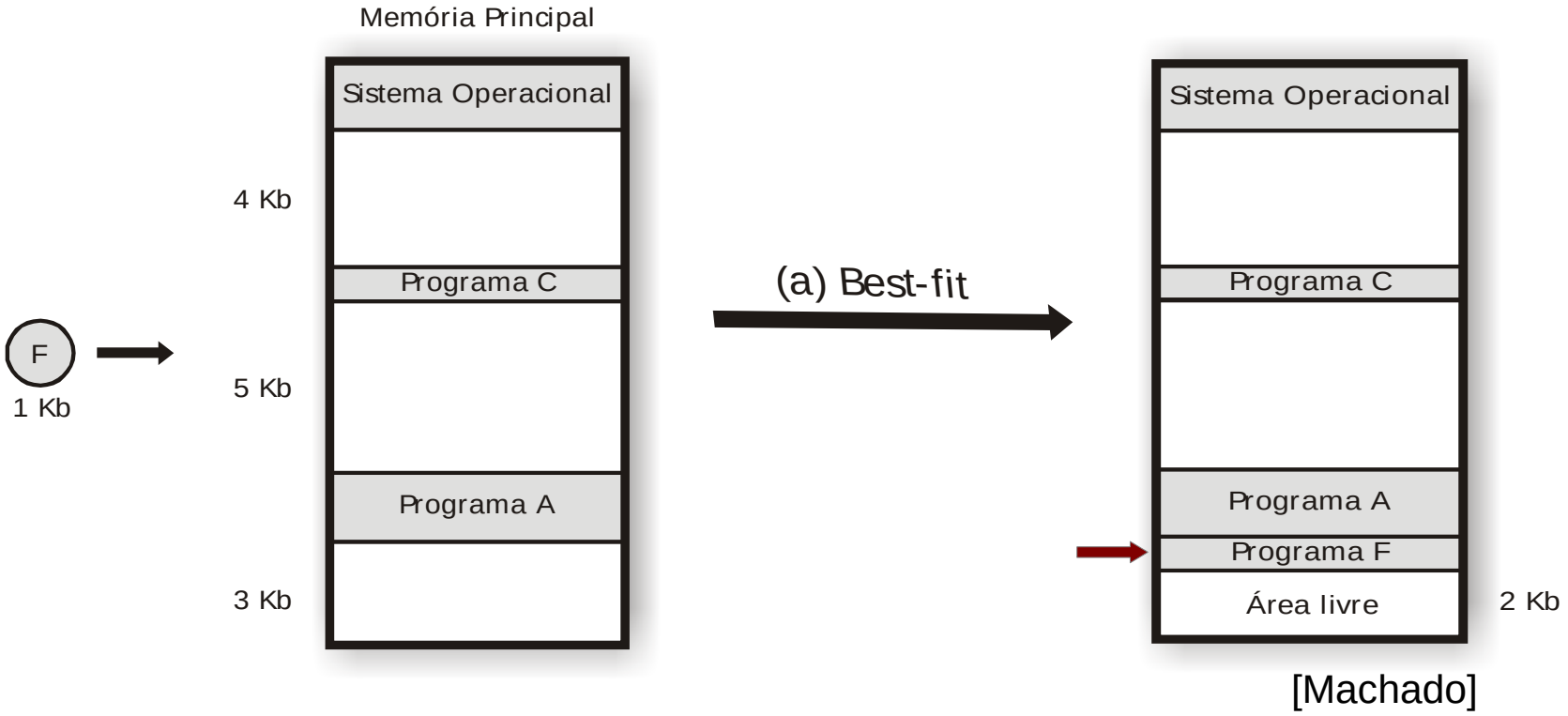
- Basicamente, há 3 estratégias para determinar em qual área livre (bloco) um programa será carregado para execução
  - Essas estratégias tentam evitar ou diminuir a fragmentação externa
  - O sistema possui uma **lista de áreas livres** com endereço e tamanho de cada área
  - ***First-fit:*** Aloca o ***primeiro*** bloco grande o suficiente para carregar o programa (não é preciso percorrer toda a lista de áreas livres)



[Machado]

Estratégias de alocação de partição

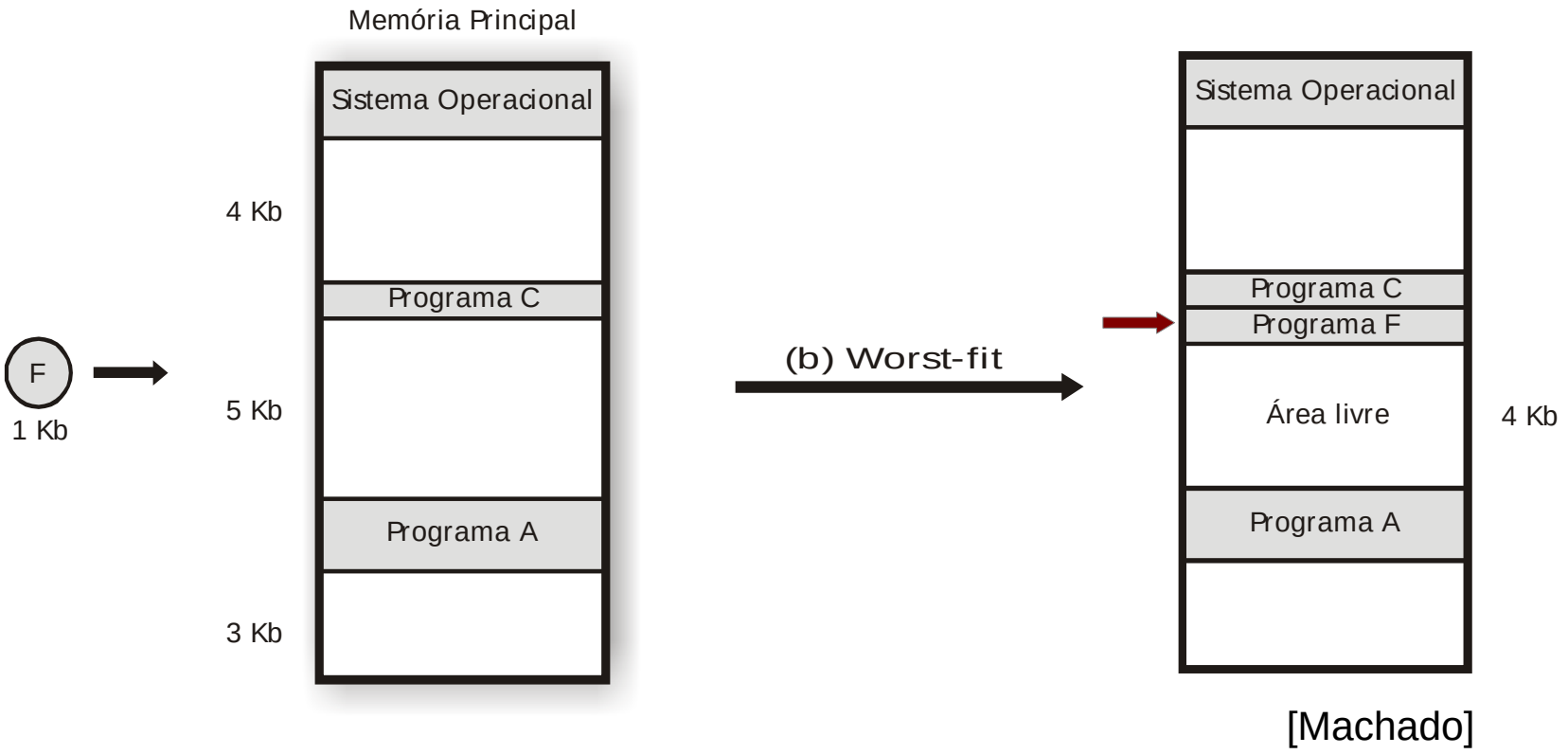
- ***Best-fit:*** Melhor partição, aloca o ***menor*** bloco grande o suficiente; é preciso procurar na lista inteira, a menos que a lista esteja ordenada por tamanho; gera o menor espaço de memória restante (sem utilização)



[Machado]

Estratégias de alocação de partição

- ***Worst-fit:*** Aloca a maior bloco; também é preciso procurar na lista inteira, caso não esteja ordenada; gera o maior bloco de memória restante



[Machado]

First-fit e ***best-fit*** são melhores do que ***worst-fit*** em termos de **redução de tempo** e **utilização da memória;** em geral, o **First-fit é**

Paginação

➢Alternativamente, o **espaço de endereçamento físico** de um processo pode ser **não-contíguo**; o processo é alocado na memória física onde houver blocos livres

➢A memória física é dividida em **blocos de tamanho fixo** chamados quadros (*frames*)

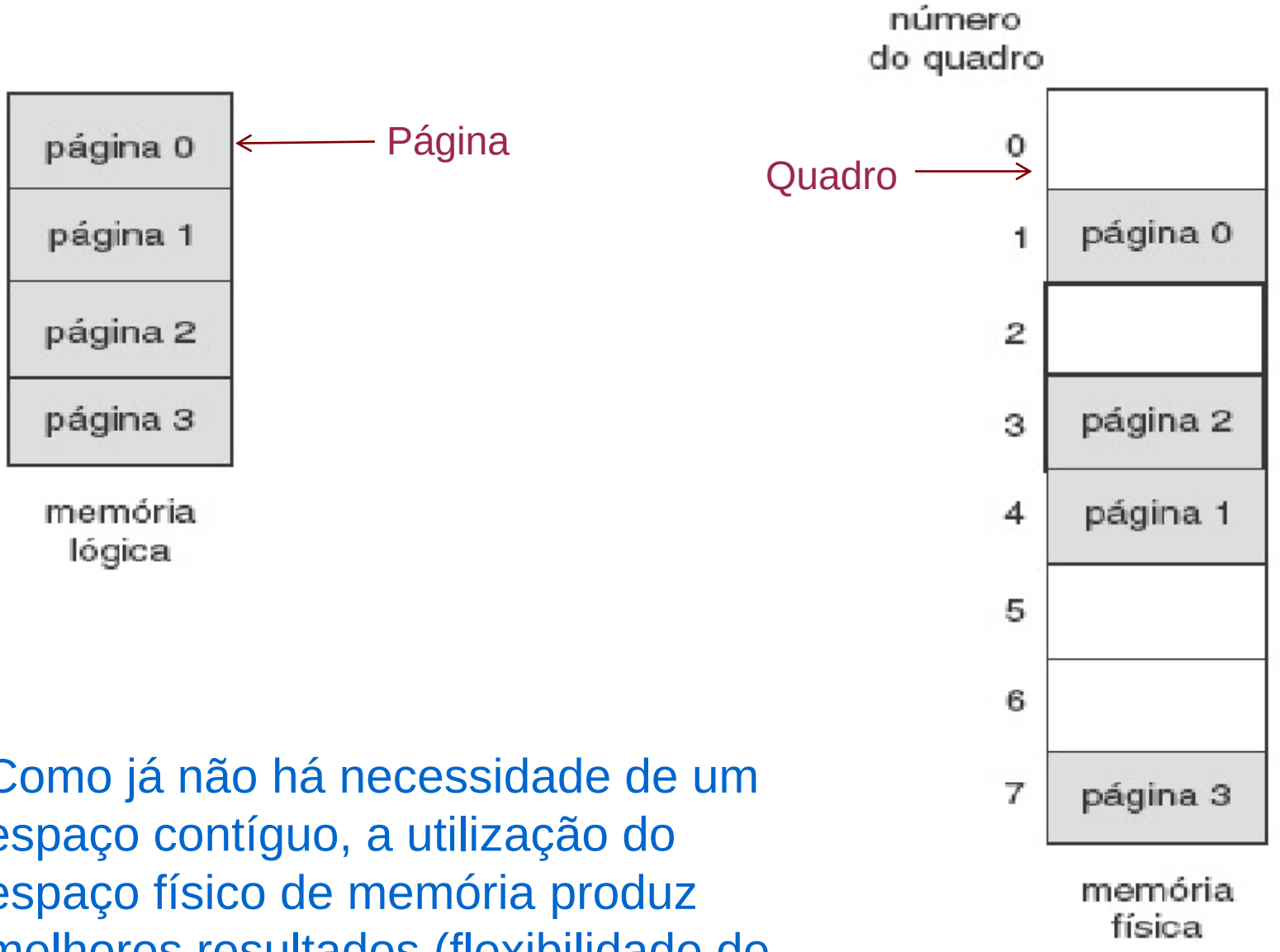
- O SO mantém registro de todos os quadros livres

➢A memória lógica (espaço de endereços lógicos) é dividida em **blocos de mesmo tamanho** chamados páginas (*pages*)

- Normalmente potência de 2, entre 512 *bytes* e 16 MB por página
- Para rodar um programa de ***n*** páginas, o SO precisa encontrar ***n*** quadros livres e carregar o programa

➢É usada uma **Tabela de Páginas** para **traduzir endereços lógicos em físicos**

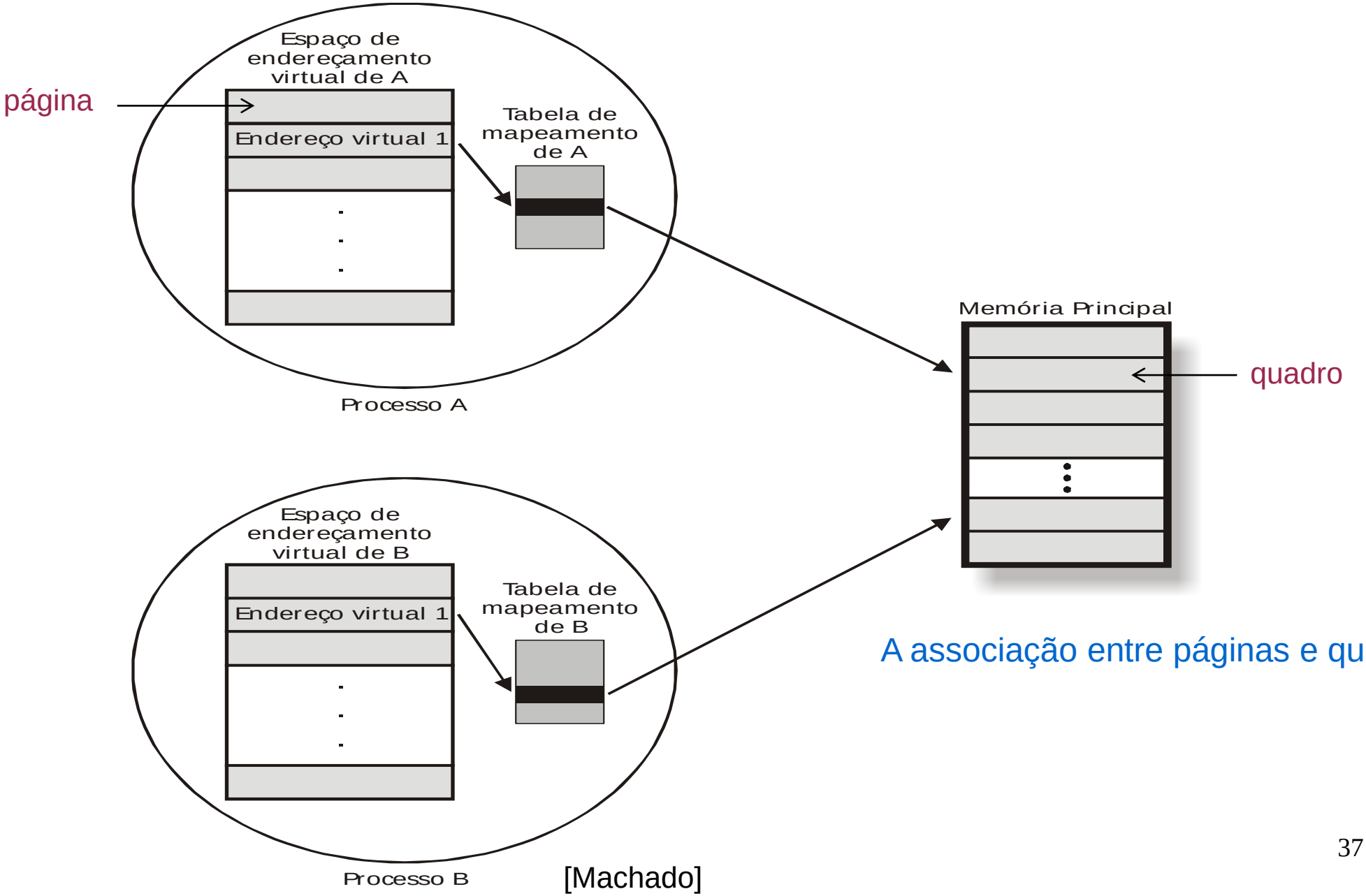
Paginação



Como já não há necessidade de um espaço contíguo, a utilização do espaço físico de memória produz melhores resultados (flexibilidade de alocação)

[Silberschatz]

Paginação



A associação entre páginas e quadros

[Machado]

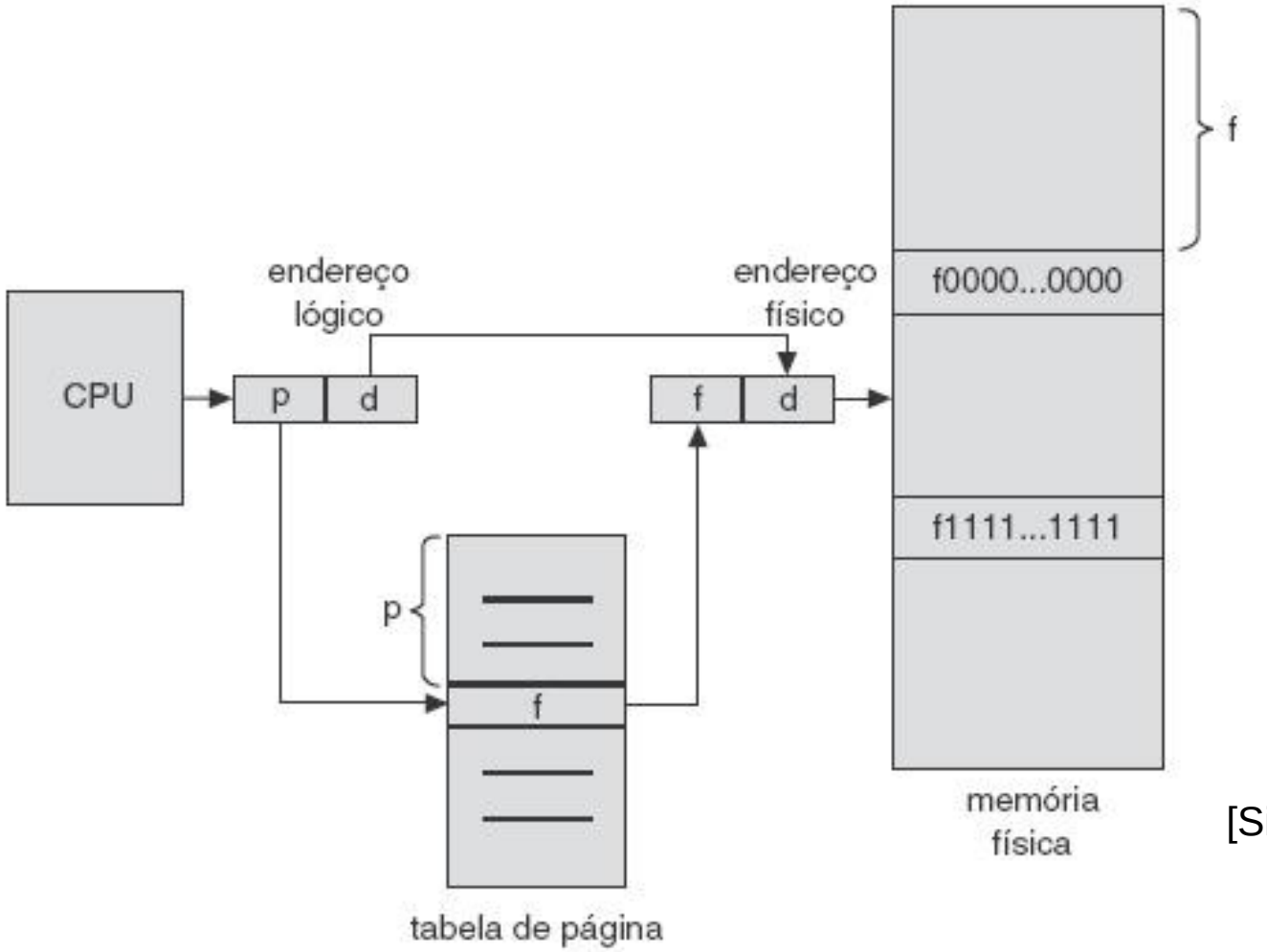
Tradução de endereço

➢O endereço gerado pela CPU é dividido em:

- ***Número de página (p)*** – usado como índice na tabela de página que contém o endereço base de cada página na memória física

- ***Deslocamento de página (d)*** – é combinado com o endereço base para definir o endereço da memória física que é enviado à unidade de memória (deslocamento dentro da página)

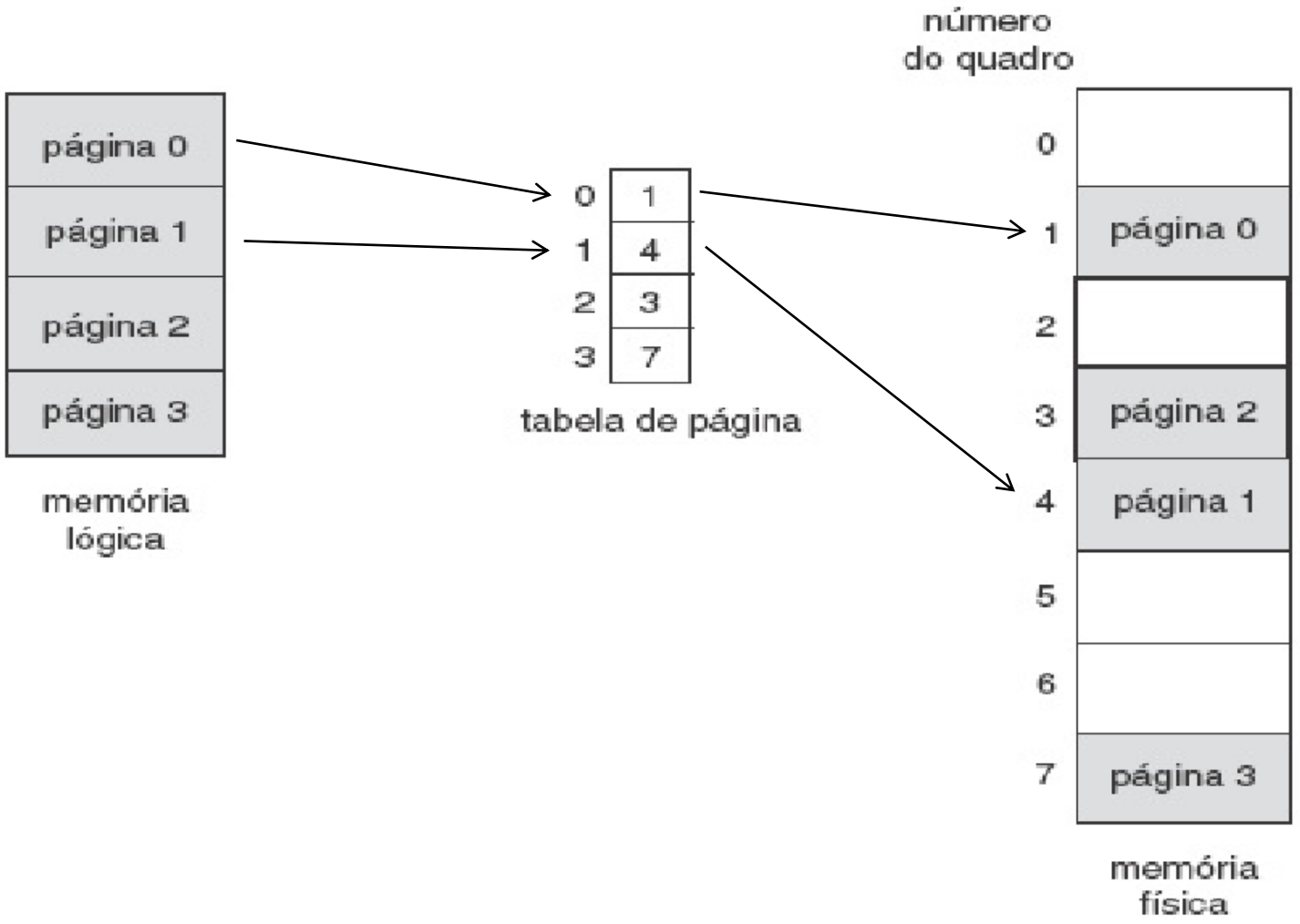
Arquitetura de tradução de endereço



[Silberschatz]

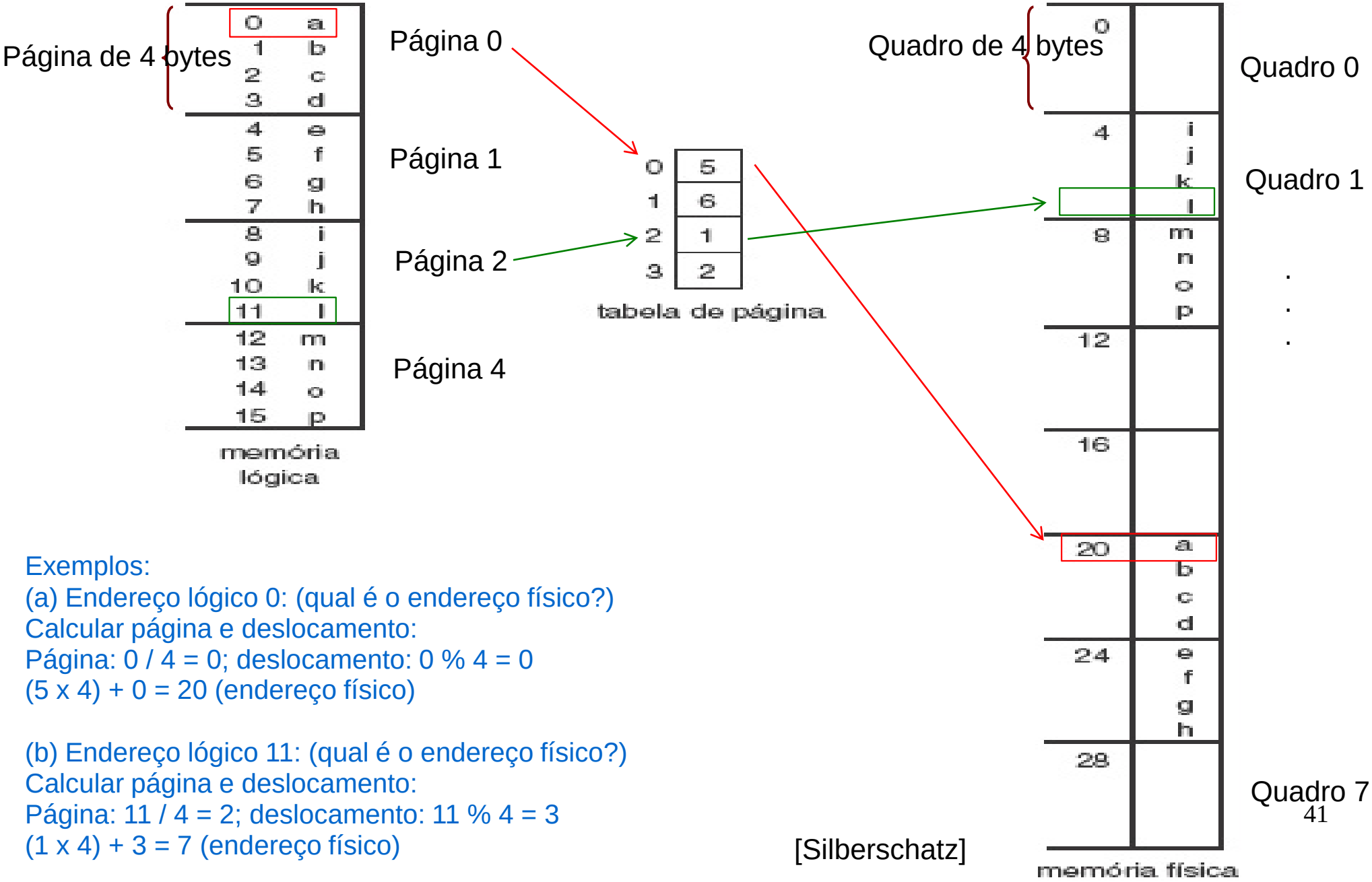
O número da página (p) é usado como índice na tabela de página (que contém o endereço base da página

Exemplo de paginação



[Silberschatz]

Outro exemplo de paginação

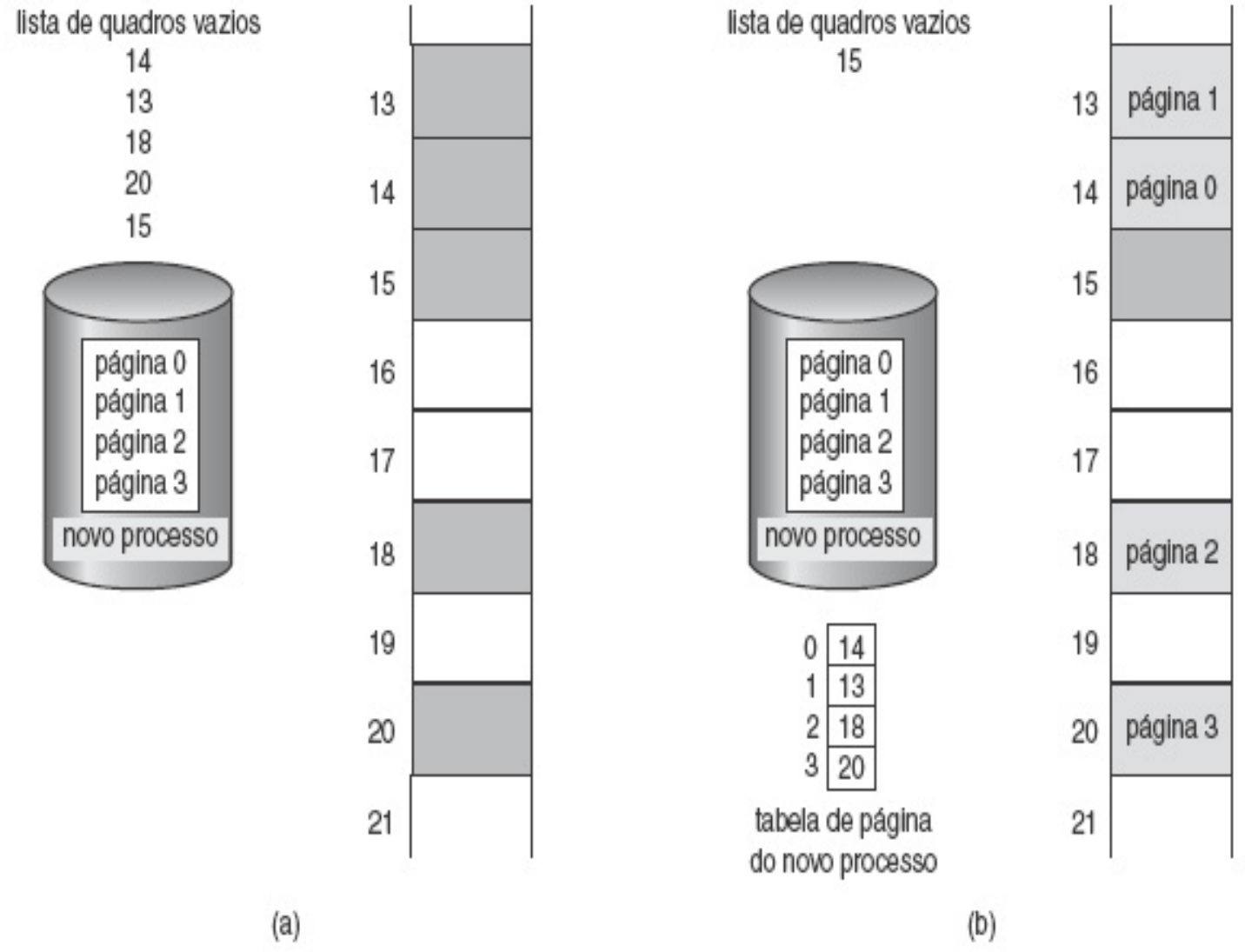


Exemplos:
(a) Endereço lógico 0: (qual é o endereço físico?)
Calcular página e deslocamento:
Página: 0 / 4 = 0; deslocamento: 0 % 4 = 0
(5 x 4) + 0 = 20 (endereço físico)

(b) Endereço lógico 11: (qual é o endereço físico?)
Calcular página e deslocamento:
Página: 11 / 4 = 2; deslocamento: 11 % 4 = 3
(1 x 4) + 3 = 7 (endereço físico)

[Silberschatz]

Quadros livres



Quando um processo chega para ser executado, seu tamanho, expresso em número de páginas, é examinado, e cada **página** (**lógica**) é alocada em um **quadro livre** (**físico**)

[Silberschatz]

Proteção de memória

➢A proteção de memória é implementada associando-se um **bit de proteção a cada página na Tabela de Páginas**

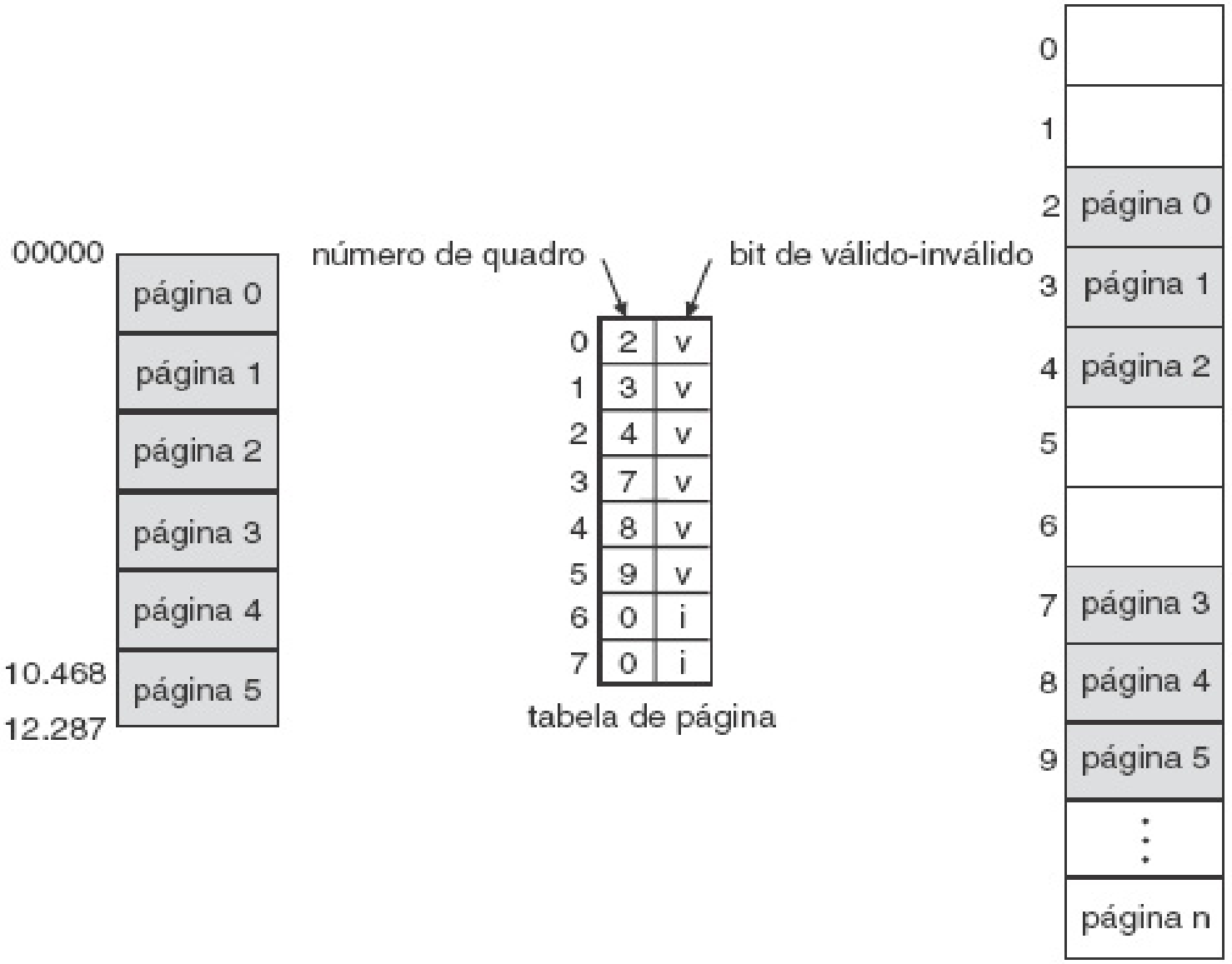
➢Um bit pode definir se uma página pode ser de leitura/escrita ou somente leitura – uma tentativa de escrita em uma página somente leitura gera uma interrupção

➢Um **bit válido-inválido** é associado a cada entrada na tabela de página (indica se o acesso a uma página é ou não válido)

- "**válido**" indica que a página associada **está no espaço de endereçamento lógico** do processo, e é portanto uma página legal

- "**inválido**" indica que a página **não está no espaço de endereçamento lógico** do processo

Bit válido (v) ou inválido (i) em uma tabela de página



[Silberschatz]

Segmentação

➢Esquema de **gerenciamento de memória** baseado na **visão de memória do usuário**

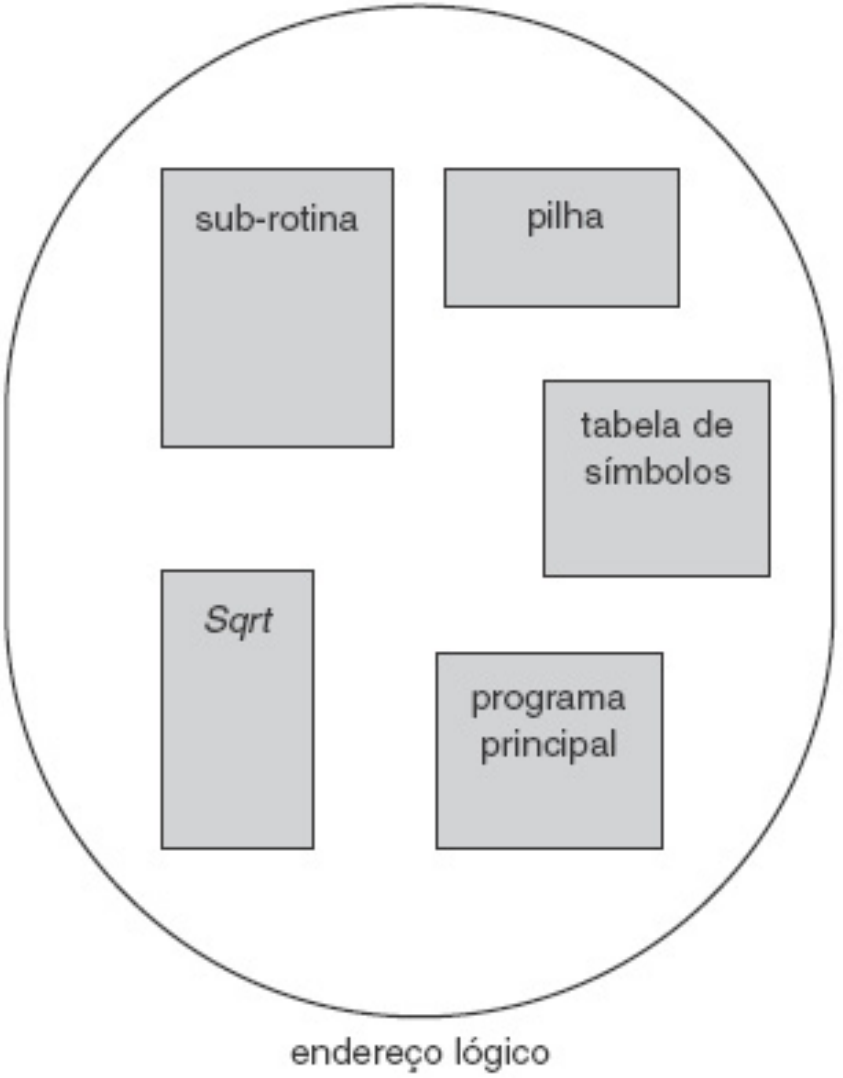
➢

➢Um programa é uma coleção de segmentos. Um segmento é uma unidade lógica, tal como:

- programa principal,
- *procedure* (sub-rotina),
- variável local, variável global,
- estruturas de dados (arrays)
- *stack* (pilha),
- tabelas de símbolos, etc.

➢A memória não é dividida em tamanhos fixos, e sim de acordo com o tamanho de cada segmento

Visão de usuário de um programa

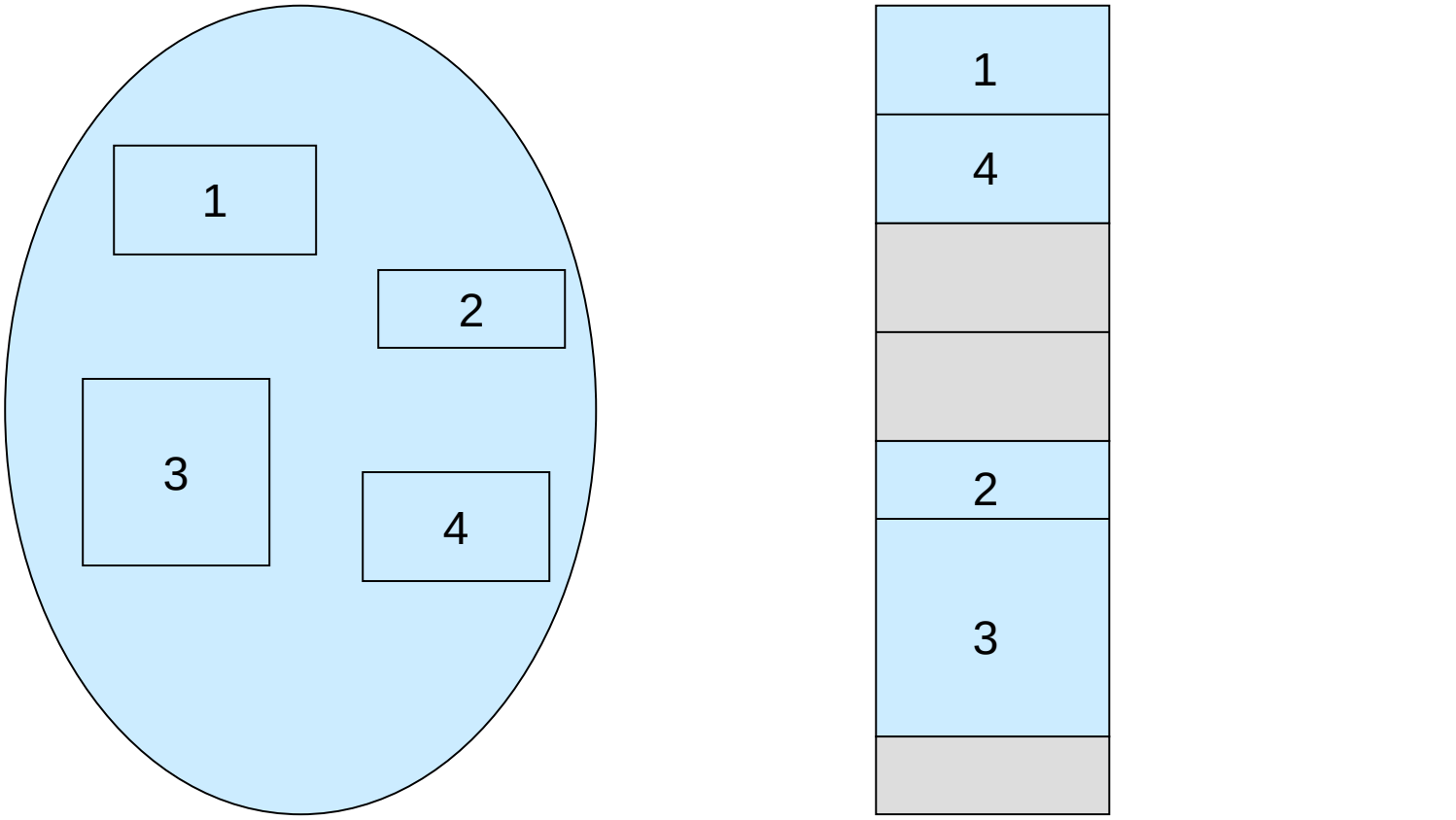


Um **espaço de endereçamento lógico** é uma **coleção de segmentos**

Cada **segmento** tem um **nome** (ou número) e **tamanho**

[Silberschatz]

Visão lógica da segmentação



Espaço do usuário

Espaço da memória física

[Silberschatz]

Um compilador pode criar segmentos separados para as **variáveis globais**, a **pilha de chamada a funções**, as **variáveis locais,** etc.

Pode causar fragmentação externa

Referências

- [Silberschatz] SILBERCHATZ, A., GALVIN, P. B. e GAGNE, G. **Sistemas Operacionais com Java**. 7ª ed., Rio de Janeiro: Elsevier, 2008.
- [Tanenbaum] TANENBAUM, A. **Sistemas Operacionais Modernos**. 3ª ed. São Paulo: Prentice Hall, 2009.
- [MACHADO] MACHADO, F. B. e MAIA, L. P. **Arquitetura de Sistemas Operacionais**. 4ª ed., Rio de Janeiro: LTC, 2007.