SISTEMAS OPERACIONAIS I

Prof. Me. Fábio Nascimento
Email: prof.fabionascimento1@gmail.com

O **gerenciamento de memória** (ou **gestão da memória**) é uma função crucial dos sistemas operacionais (SO) e constitui uma das principais responsabilidades do núcleo (*kernel*) do sistema.

Gerenciamento de Memória

O gerenciamento de memória (ou gestão da memória) é uma função crucial dos sistemas operacionais (SO) e constitui uma das principais responsabilidades do núcleo (kernel) do sistema.

A memória é um dos recursos computacionais mais importantes, pois é o local onde são armazenados dados e informações no computador.

O gerenciamento da memória é fundamental e complexo, sendo essencial para garantir a eficiência das aplicações que rodam no computador, visto que a tendência das aplicações é consumir cada vez mais esse recurso

O objetivo principal do gerenciamento de memória é permitir que vários processos sejam executados ao mesmo tempo, mantendo um bom desempenho do sistema.

O **gerenciamento de memória** (ou **gestão da memória**) é uma função crucial dos sistemas operacionais (*SO*) e constitui uma das principais responsabilidades do núcleo (*kernel*) do sistema.

Gerenciamento de Memória

Funções e Responsabilidades

O gerenciamento de memória é realizado pelo gerenciador de memória, que tem as seguintes funções:

1. **Controle de Uso e Alocação**: Gerenciar quais partes da memória estão em uso e quais estão disponíveis. O gerenciador aloca a memória quando os processos precisam e a libera após o término deles.

2. **Gerenciamento da Hierarquia**: Controlar a hierarquia de memória, que combina diferentes tipos de memória, como memória cache (rápida, volátil e de alto custo), memória principal (RAM – volátil, médio custo) e memória secundária (disco – não volátil, baixo custo e alto armazenamento). O gerenciador garante que essas memórias estejam interligadas para o bom funcionamento do SO.

O **gerenciamento de memória** (ou **gestão da memória**) é uma função crucial dos sistemas operacionais (*SO*) e constitui uma das principais responsabilidades do núcleo (*kernel*) do sistema.

Gerenciamento de Memória

Funções e Responsabilidades

3. **Proteção e Integridade**: O sistema operacional deve proteger as áreas de memória utilizadas pelos processos, impedindo que um programa tente acessá-la indevidamente. Na multiprogramação, o gerenciamento de memória deve resolver os problemas de relocação e proteção.

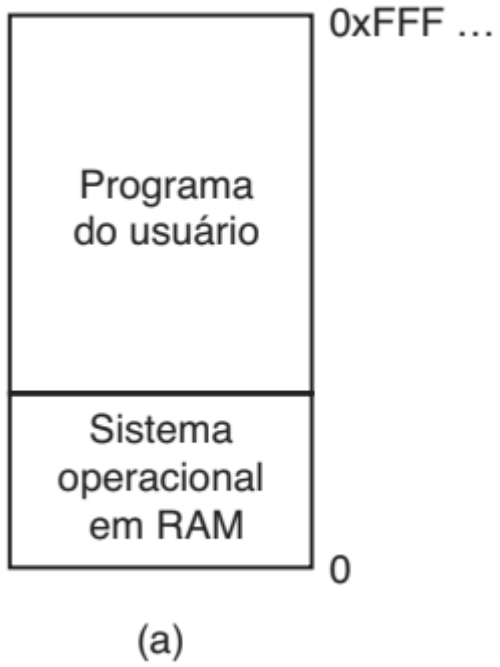
4. **Troca de Processos (Swapping)**: Controlar a troca de processos entre a memória principal e o disco quando a memória principal não é suficiente para manter todos os processos em execução.

Gerenciamento de Memória

Mesmo nesse cenário simples, havia algumas variações na forma como o sistema operacional era organizado na memória. Elas são:

Gerenciamento de Memória

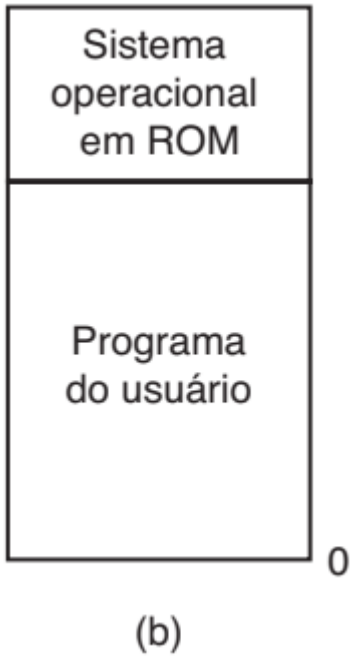
Sistema operacional na parte inferior da RAM – usado em alguns mainframes e minicomputadores antigos, mas pouco comum.



(a)

Gerenciamento de Memória

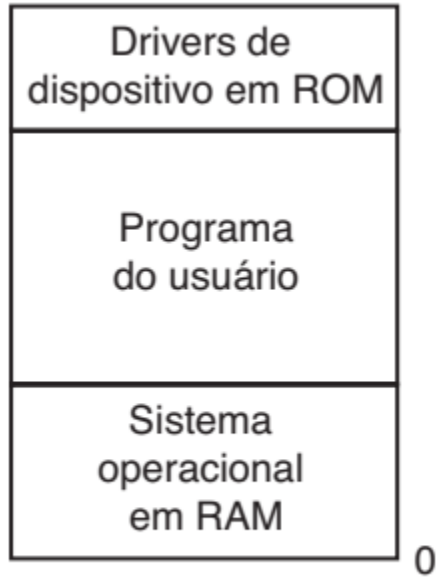
Sistema operacional em ROM no topo da memória – encontrado em alguns portáteis e sistemas embarcados.



(b)

Gerenciamento de Memória

Parte do sistema em ROM (BIOS) no topo da memória e o restante em RAM abaixo – modelo usado nos primeiros PCs que executavam MS-DOS.



(c)

Gerenciamento de Memória

Os modelos (a) e (c) tinham uma grande desvantagem: **se um programa do usuário apresentasse erro e sobrescrevesse trechos do sistema operacional, o computador poderia falhar completamente.**

Nessas arquiteturas, apenas um processo podia ser executado por vez. O fluxo típico era simples:

1. o usuário digitava um comando
2. o sistema operacional carregava o programa solicitado do disco para a memória e o executava
3. ao terminar, o sistema exibia novamente o prompt de comando, pronto para receber outro programa, sobrescrevendo o anterior na memória.

Gerenciamento de Memória

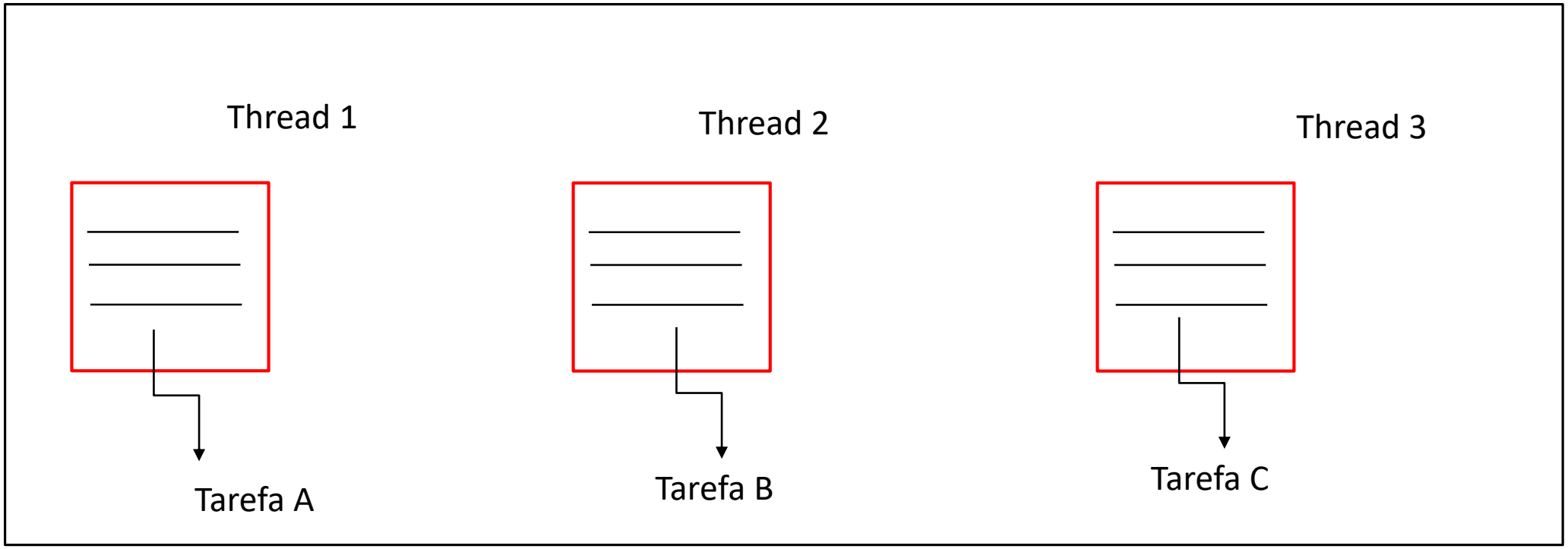
Uma forma limitada de paralelismo em sistemas sem abstração de memória era o uso de **threads**. Como todos os threads de um processo compartilham a mesma área de memória, esse modelo funcionava.

No entanto, seu uso era bastante restrito, já que, em geral, os usuários desejavam executar programas independentes ao mesmo tempo — algo impossível sem abstrações de memória mais avançadas.

Além disso, sistemas tão primitivos a ponto de não oferecerem abstração de memória dificilmente ofereciam suporte a threads.

Gerenciamento de Memória

PROCESSO



Gerenciamento de Memória

Mesmo sem uma abstração de memória, é possível executar múltiplos programas.

A forma mais simples de fazer isso é através do **swapping (troca de processos)**:

- o sistema operacional salva todo o conteúdo da memória em um arquivo no disco,
- carrega o próximo programa e o executa.

Como apenas um programa fica na memória de cada vez, não há risco de conflito. Porém, com o avanço do hardware, tornou-se possível rodar mais de um programa ao mesmo tempo sem precisar de swapping.

Gerenciamento de Memória

Para isso, o IBM 360 usou a chamada **realocação estática**. Nela, o carregador ajustava os endereços durante o carregamento. Esse método funcionava, mas tinha desvantagens:

- deixava o carregamento mais lento;
- exigia que os programas indicassem claramente o que era endereço real e o que era apenas um número.

Esse tipo de endereçamento físico direto não é mais usado em computadores modernos, mas ainda é comum em **sistemas embarcados** e **cartões inteligentes**, onde os programas já vêm prontos e não podem ser alterados pelo usuário. É o caso de rádios, micro-ondas e máquinas de lavar.

Gerenciamento de Memória – espaços de endereçamento

Permitir que os programas acessem diretamente a **memória física** traz muitos problemas.

1. Primeiro, qualquer programa poderia sobrescrever partes críticas do sistema operacional, causando falhas graves, mesmo que só um aplicativo estivesse rodando.
2. Segundo, esse modelo dificulta rodar vários programas ao mesmo tempo, algo comum em computadores pessoais, onde abrimos diferentes aplicativos (editor de texto, navegador, e-mail etc.) e alternamos entre eles.

Para que isso seja possível, dois problemas precisam ser resolvidos:

1. **proteção** (evitar que um programa interfira em outro)
2. **realocação** (permitir que cada programa use endereços independentes).

Gerenciamento de Memória – espaços de endereçamento

O IBM 360 usava uma solução inicial de proteção: cada bloco de memória recebia uma **chave de proteção**, e apenas processos com a chave correta podiam acessá-lo. Isso resolvia parte do problema, mas não a realocação, que era feita de forma lenta e complicada.

A solução mais eficiente foi criar uma **nova abstração da memória: o espaço de endereçamento**.

- Assim como o conceito de processo cria uma “CPU virtual” para cada programa, o espaço de endereçamento cria uma “memória virtual”.
- Cada processo passa a ter seu próprio conjunto de endereços, independente dos outros.
- Em alguns casos, processos podem compartilhar parte desse espaço, mas só quando isso é necessário.

Gerenciamento de Memória – espaços de endereçamento

Um espaço de endereçamento é simplesmente o conjunto de endereços que podem ser usados.

Gerenciamento de Memória – espaços de endereçamento

O desafio está em fazer com que cada programa enxergue seu próprio espaço de endereçamento, mesmo que os endereços se repitam entre processos.

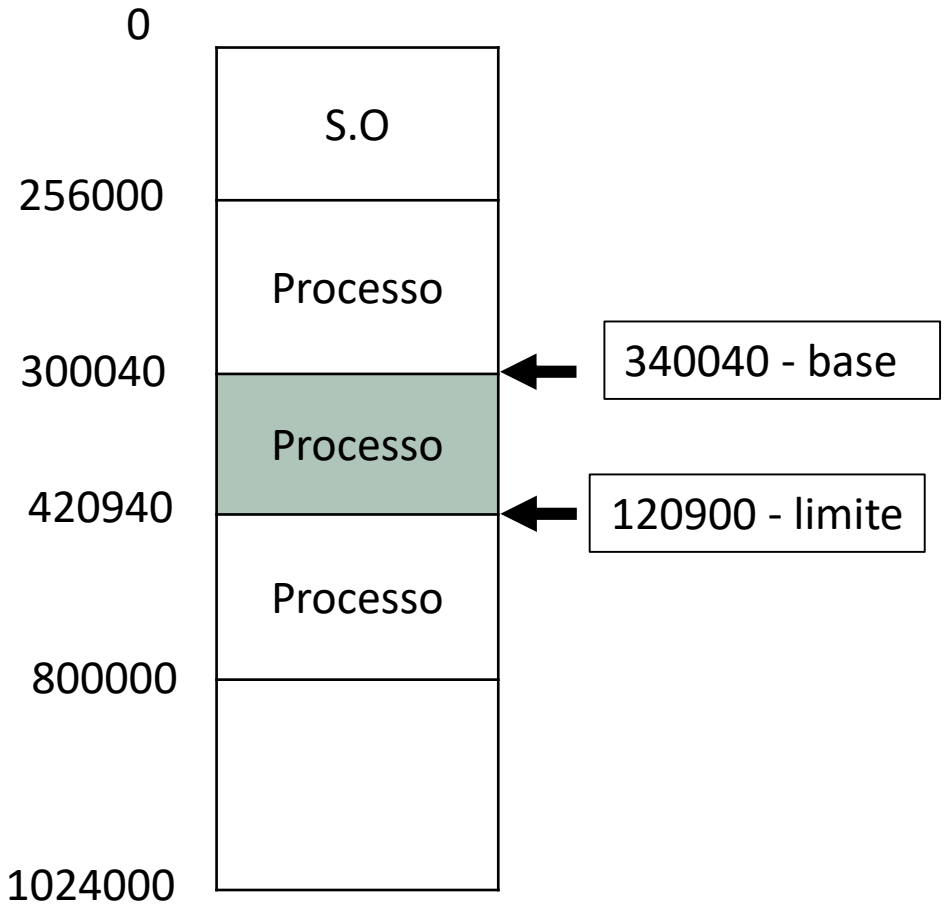
Uma solução simples, foi o uso de **registradores base e limite**:

- O **registrador base** guarda o endereço inicial do programa na memória.
- O **registrador limite** guarda o tamanho do programa ou seja a quantidade de memória alocada para esse processo.
- Sempre que o processo acessa a memória, o hardware adiciona automaticamente o valor do registrador base ao endereço solicitado e confere se o resultado não ultrapassa o limite.

Gerenciamento de Memória – espaços de endereçamento

A CPU deve verificar cada processo que esta em execução.
Caso algum deles tente ser executado fora dos limites **base** e **limite** a CPU deve acusar um erro.

Lembrando que o valor base e limite são determinados pelo S.O no momento de execução.



Gerenciamento de Memória – espaços de endereçamento

Esse método garante que cada processo tenha sua própria área de memória, protegida contra invasão de outros. No entanto, tem desvantagens:

- É necessário fazer uma **adição** e uma **comparação** a cada acesso de memória, o que pode ser lento sem circuitos especiais.
- Algumas implementações, como a do Intel 8088, nem tinham registrador limite, oferecendo menos proteção.

Mesmo com limitações, esse modelo foi um passo importante na evolução do gerenciamento de memória.

Gerenciamento de Memória – Troca de processos (Swapping)

Em sistemas como Windows, macOS ou Linux, logo após a inicialização já existem dezenas de processos ativos (entre 50 e 100 ou mais).

Muitos deles são criados automaticamente, como serviços que verificam atualizações, monitoram conexões de rede ou checam e-mails. Mesmo esses processos simples podem ocupar alguns megabytes de memória.

Já aplicações maiores, como o Photoshop, podem precisar de centenas de megabytes apenas para iniciar, chegando a usar vários gigabytes durante o processamento.

Manter todos os processos carregados na memória ao mesmo tempo, portanto, exigiria uma quantidade enorme de RAM, o que geralmente não é viável.

Para lidar com isso, foram criadas duas estratégias:

Gerenciamento de Memória – Troca de processos (Swapping)

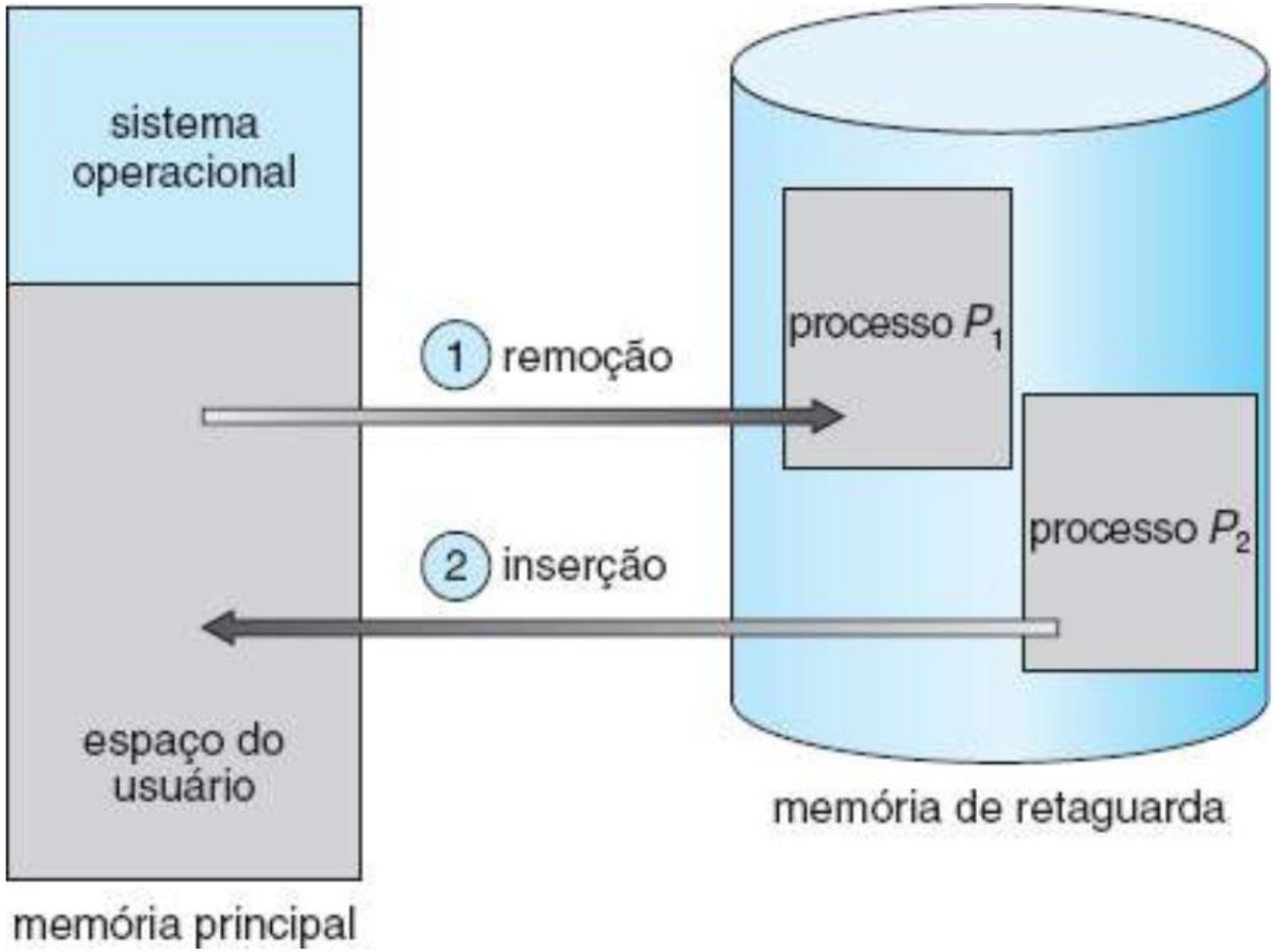
1- Swapping (troca de processos): **o processo é carregado inteiro na memória, executado por um tempo e depois devolvido ao disco. Assim, processos que estão inativos permanecem armazenados no disco, liberando espaço na memória. Quando precisam rodar novamente, são recarregados, possivelmente em outro lugar da memória.**

Se isso acontecer, os endereços do programa precisam ser realocados, tarefa feita pelo hardware (como com registradores base e limite).

Quando várias trocas deixam espaços “quebrados” na memória, pode-se realizar a **compactação de memória**, que junta todos os processos para liberar um espaço maior contínuo.

Porém, isso é raro porque consome muito tempo de CPU.

Gerenciamento de Memória



Gerenciamento de Memória – Troca de processos (Swapping)

2- Memória virtual: permite que os programas sejam executados mesmo que apenas parte deles esteja carregada na memória física.

Gerenciamento de Memória – Troca de processos (Swapping)

Outro desafio é decidir **quanto de memória alocar para um processo**.

Se o processo tiver tamanho fixo, a alocação é simples: o sistema dá exatamente o espaço necessário.

Mas se o processo puder crescer (como quando variáveis são alocadas dinamicamente em linguagens de programação), surge o problema:

- Se houver espaço livre logo após ele, o processo pode expandir.
- Se não houver, pode ser necessário movê-lo para outra área com espaço suficiente, ou até trocar outros processos de lugar.
- Se não houver espaço disponível na memória nem no disco de troca, o processo deve ser suspenso (ou até encerrado).

Uma solução é **reservar espaço extra** quando um processo é criado ou movido, prevendo que ele crescerá. Isso reduz a necessidade de trocas futuras. Porém, ao enviar um processo para o disco, só a memória efetivamente usada deve ser transferida, para evitar desperdício.

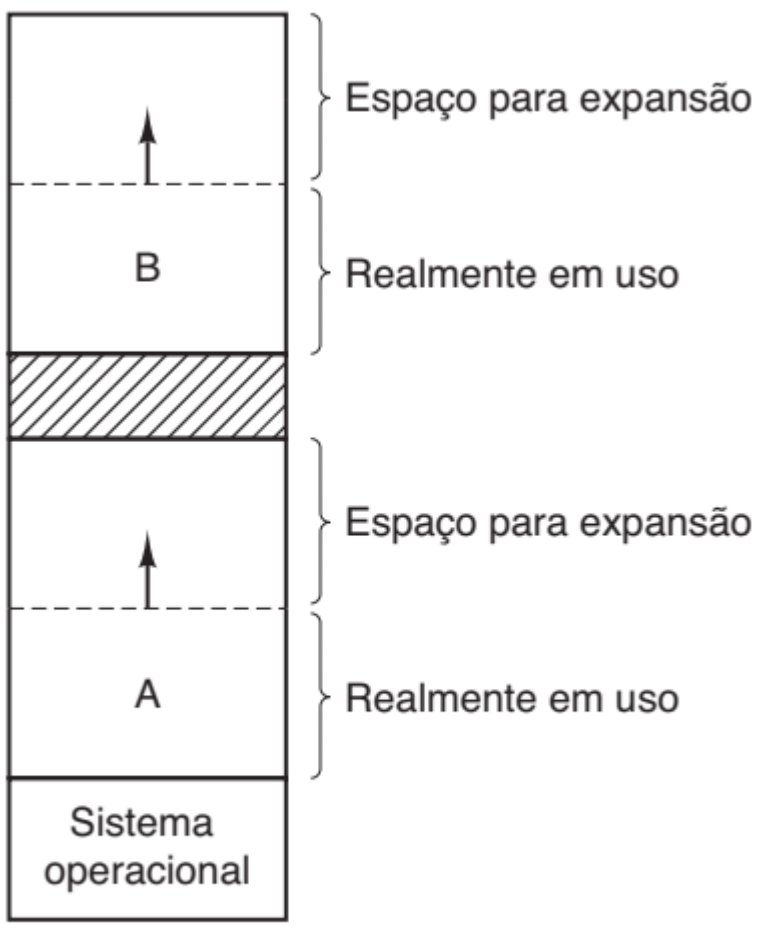
Gerenciamento de Memória – Troca de processos (Swapping)

Em sistemas mais complexos, o processo pode ter dois segmentos de crescimento:

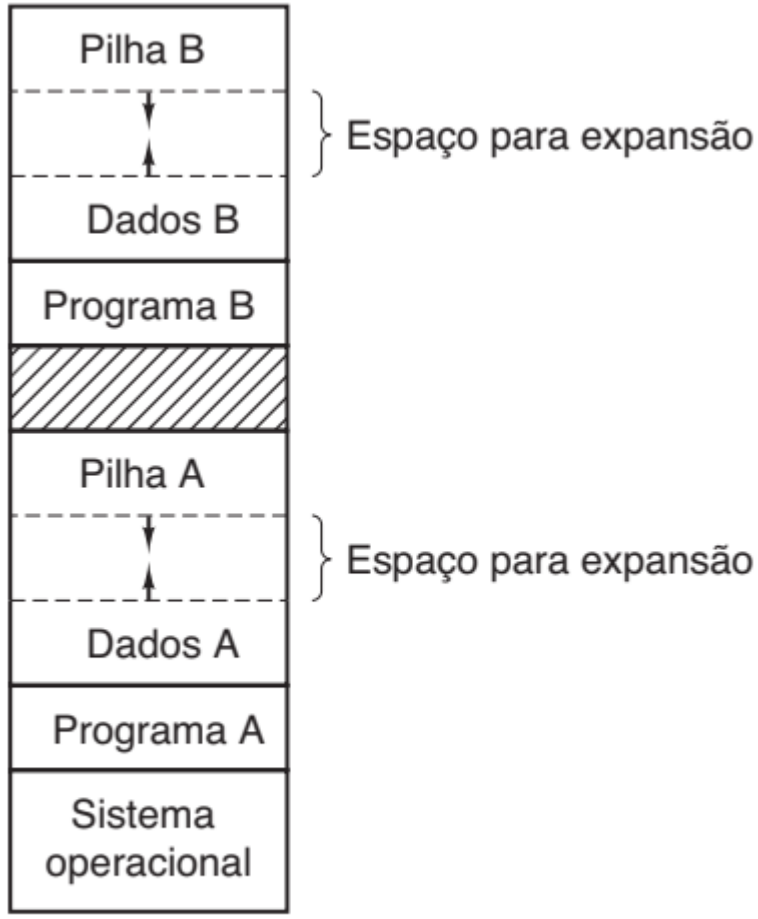
- Um **segmento de dados**, que cresce para cima. Também chamado ne *heap,* ele aumenta a memória disponível para um processo, sendo **"para cima" um termo comum para indicar o crescimento de endereços de memória mais baixos para mais altos.**
- Uma **pilha**, que cresce para baixo.

A "stack" (pilha) é um segmento de memória fundamental na execução de um programa, funcionando como uma estrutura de dados de tipo LIFO (Last In, First Out - Último a Entrar, Primeiro a Sair). Ela é usada para alocação de memória de forma automática e temporária, e seu comportamento de crescimento "para baixo“. **O crescimento "para baixo", em direção a endereços de memória mais baixos, é uma escolha de design em muitos sistemas operacionais e arquiteturas de processador.**

Gerenciamento de Memória – Troca de processos (Swapping)



(a)



(b)

Gerenciamento de Memória – Gerenciando a memória livre

Quando a memória é alocada dinamicamente, o sistema operacional precisa controlá-la.
Existem duas formas principais de fazer isso: mapas de bits e listas encadeadas.

Gerenciamento de Memória – Gerenciando a memória livre

1. Gerenciamento com mapas de bits

A memória é dividida em pequenas unidades (podem ter desde alguns bytes até alguns KB).

Para cada unidade existe um **bit no mapa**:

- **0** → espaço livre
- **1** → espaço ocupado (ou vice-versa).

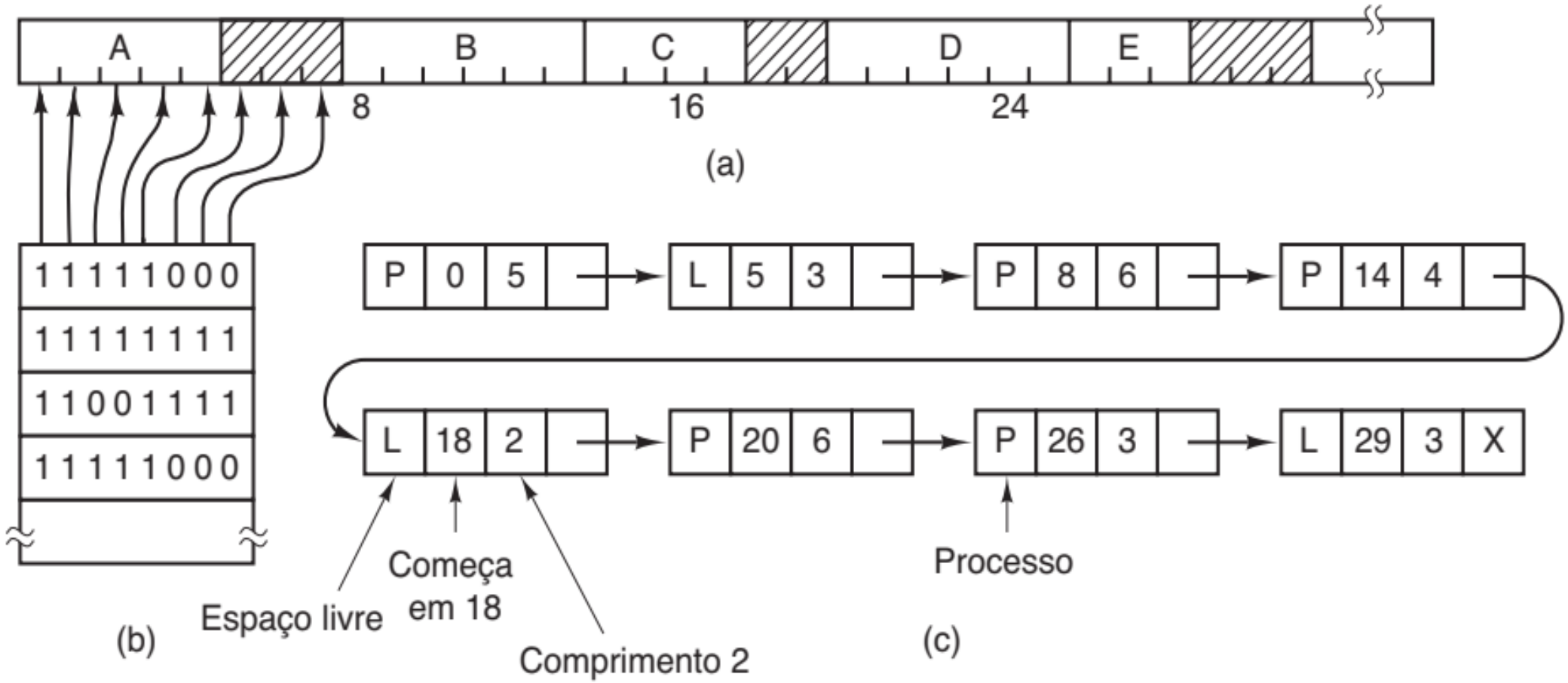
O tamanho da unidade de alocação é um ponto importante:

- Se for **pequena**, o mapa de bits será grande, mas o desperdício de memória é mínimo.
- Se for **grande**, o mapa será pequeno, mas pode haver desperdício quando o processo não ocupa múltiplos exatos da unidade.

O mapa de bits é simples de usar, mas tem um problema:

- Para carregar um processo que precisa de **k unidades seguidas**, o sistema precisa **procurar no mapa uma sequência de k bits livres**, o que pode ser lento.

(a) Uma parte da memória com cinco processos e três espaços. As marcas indicam as unidades de alocação de memória. As regiões sombreadas (0 no mapa de bits) estão livres. (b) Mapa de bits correspondente. (c) A mesma informação como lista.



Gerenciamento de Memória – Gerenciando a memória livre

2. Gerenciamento com listas encadeadas

- A memória é organizada como uma **lista de segmentos** (ocupados ou livres).
- Cada entrada indica: tipo (livre ou ocupado), endereço inicial, tamanho e ponteiro para o próximo segmento.
- Quando um processo termina, o espaço ocupado pode virar **livre** e até ser fundido com vizinhos, reduzindo a fragmentação.
- É comum usar **listas duplamente encadeadas**, que facilitam encontrar o segmento anterior e juntar espaços vizinhos.

Gerenciamento de Memória – Gerenciando a memória livre

3. Algoritmos de alocação em listas

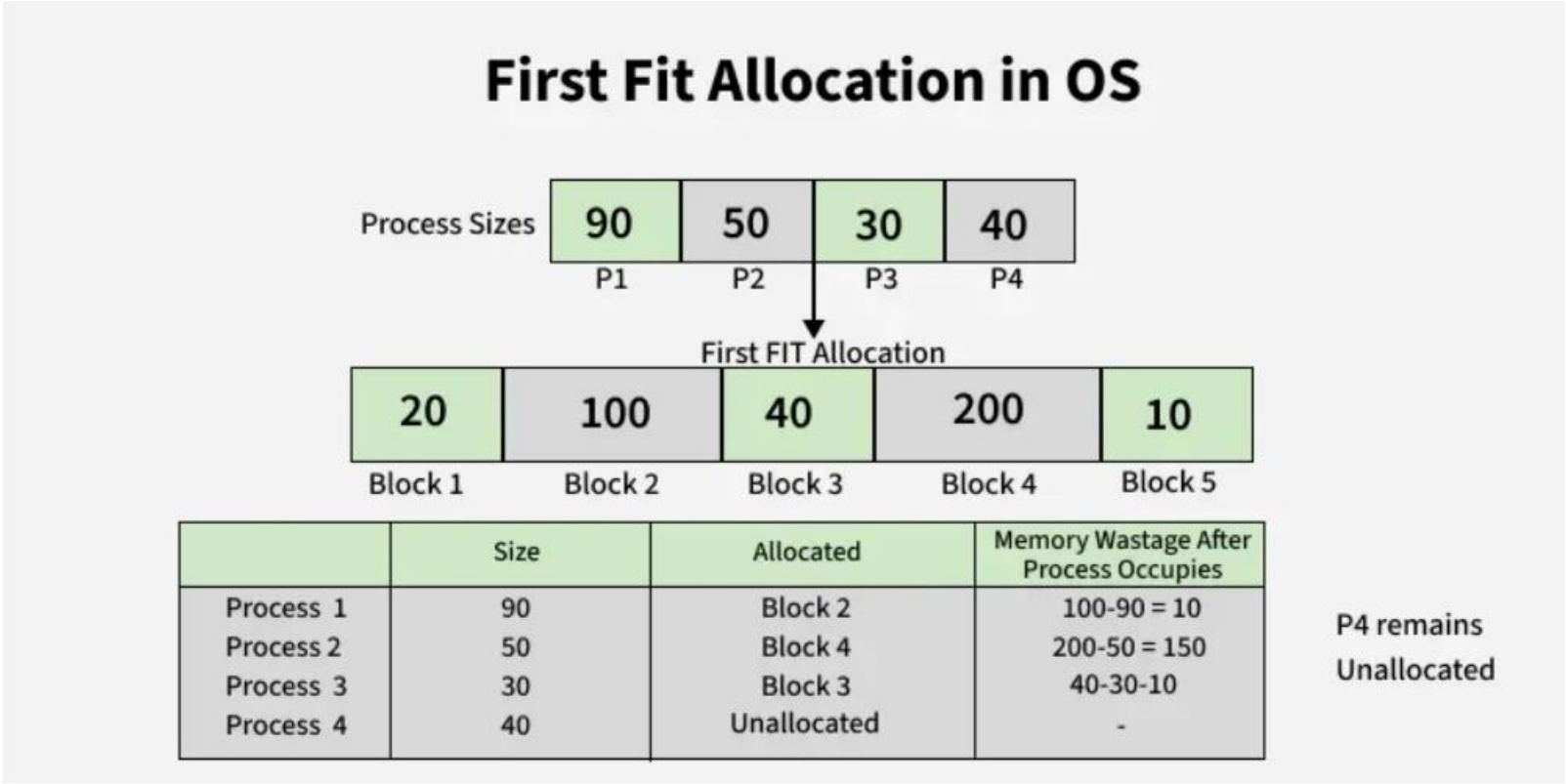
Quando o sistema precisa encontrar espaço para um processo, pode usar diferentes estratégias:

1. **First Fit (primeiro encaixe)**
2. **Next Fit (próximo encaixe)**
3. **Best Fit (melhor encaixe)**
4. **Worst Fit (pior encaixe)**
5. **Quick Fit**

Gerenciamento de Memória – Gerenciando a memória livre

First Fit (primeiro encaixe):

- Encontra o primeiro espaço livre grande o suficiente e o divide.
- É rápido e eficiente na média.



	Size	Allocated	Memory Wastage After Process Occupies
Process 1	90	Block 2	100-90 = 10
Process 2	50	Block 4	200-50 = 150
Process 3	30	Block 3	40-30-10
Process 4	40	Unallocated	-

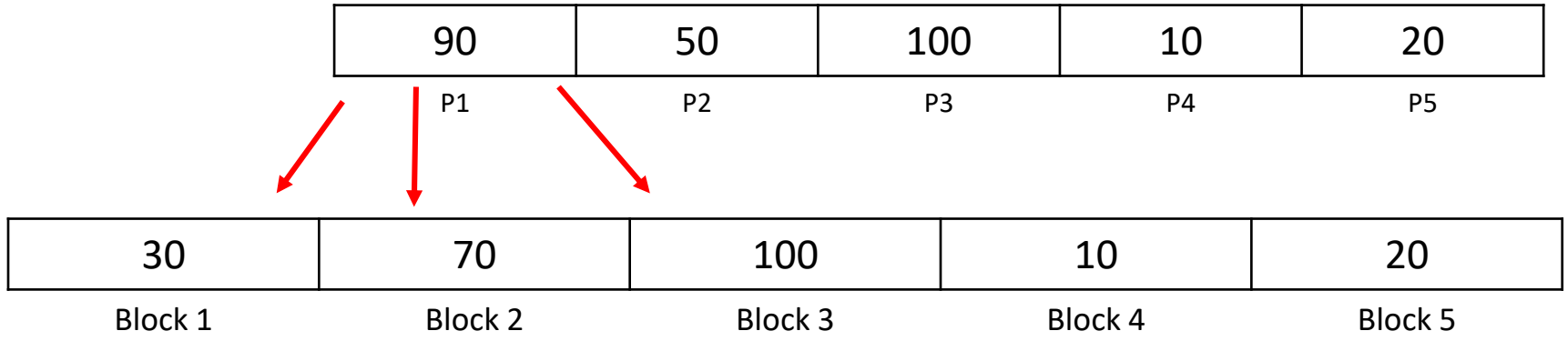
P4 remains Unallocated

Gerenciamento de Memória – Gerenciando a memória livre

Next Fit (próximo encaixe):

- Parecido com o First Fit, mas começa a busca de onde parou da última vez.
- Um pouco menos eficiente que o First Fit.

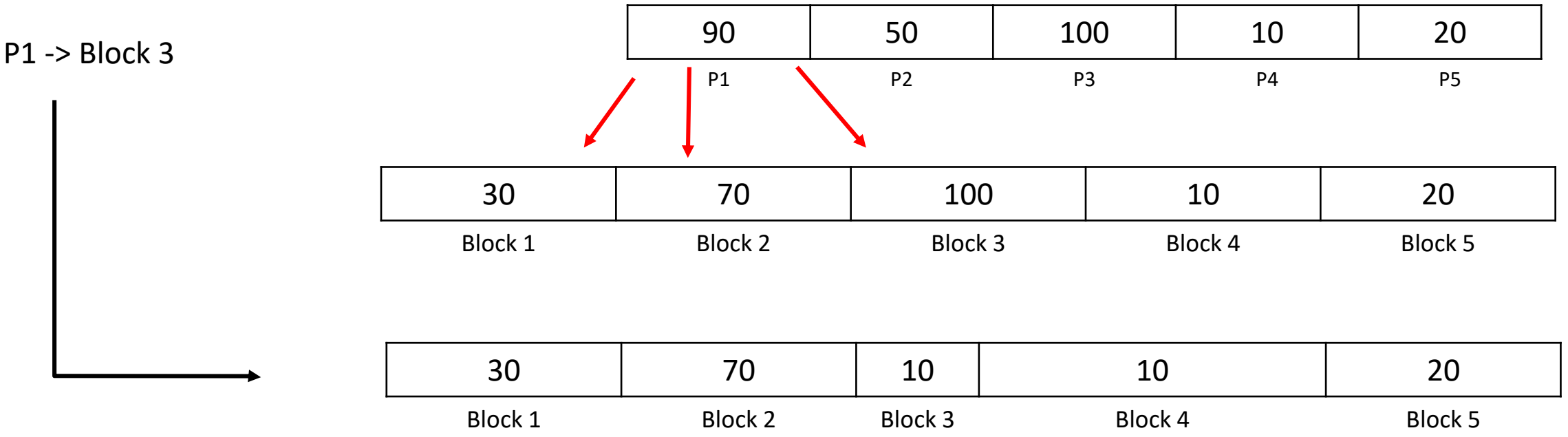
P1 -> Block 3



Gerenciamento de Memória – Gerenciando a memória livre

Best Fit (melhor encaixe):

- Procura toda a lista e escolhe o menor espaço livre que caiba no processo.
- Gera mais fragmentação, pois tende a deixar pedaços muito pequenos.

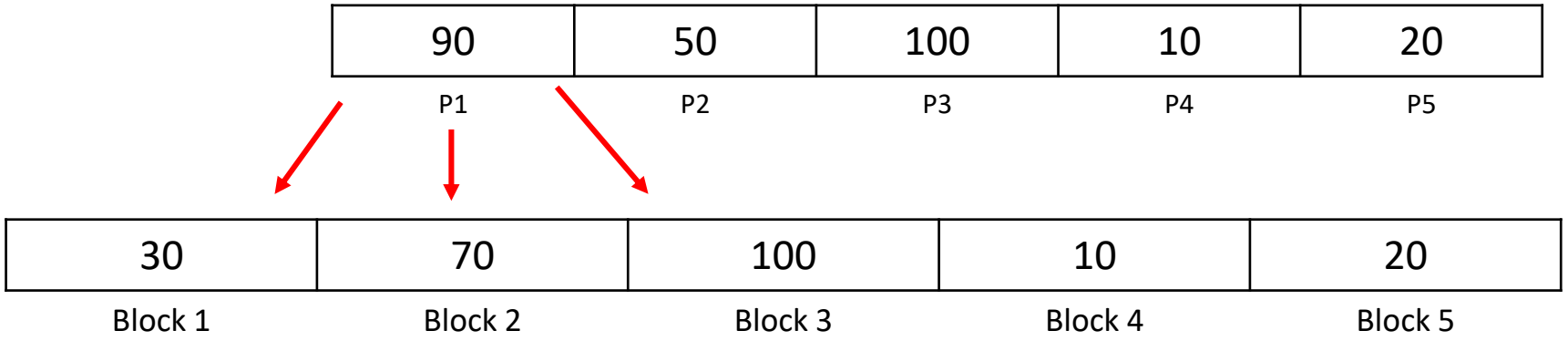


Gerenciamento de Memória – Gerenciando a memória livre

Worst Fit (pior encaixe):

- Sempre escolhe o maior espaço livre.
- A ideia é deixar os fragmentos restantes grandes o suficiente para uso futuro, mas na prática não funciona tão bem.

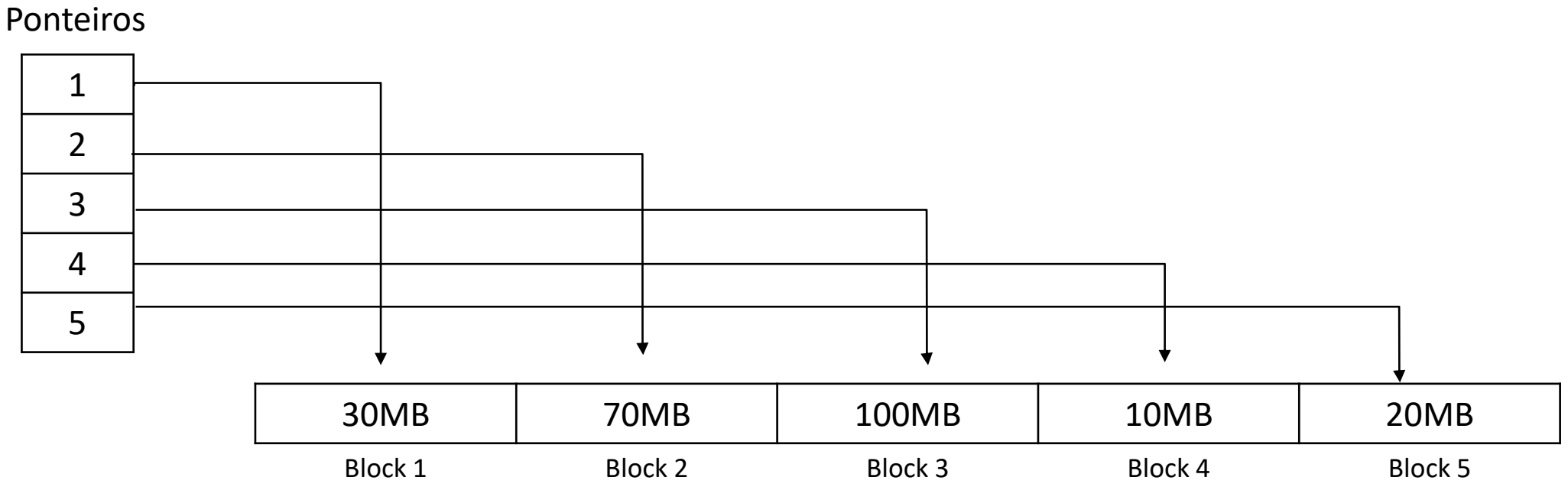
P1 -> Block 3



Gerenciamento de Memória – Gerenciando a memória livre

Quick Fit:

- Mantém listas separadas para tamanhos comuns (ex.: blocos de 4 KB, 8 KB, 12 KB, etc.).
- Muito rápido para encontrar espaço, mas dificulta a fusão de blocos livres, causando fragmentação.



Gerenciamento de Memória – Gerenciando a memória livre

- **Mapa de bits**: simples, mas lento para encontrar blocos grandes contíguos.
- **Listas encadeadas** : mais flexíveis, permitem algoritmos como First Fit, Best Fit, etc.
- Cada algoritmo tem vantagens e desvantagens, variando entre velocidade, fragmentação e aproveitamento da memória.

Gerenciamento de Memória – Memória virtual

Os registradores base e limite ajudam a criar espaços de endereçamento, mas há outro problema: **o bloatware, ou seja, programas que consomem memória em excesso.**

Apesar do aumento da capacidade das memórias, os softwares cresceram ainda mais rápido.

Na década de 1980, sistemas com apenas **4 MB de RAM** atendiam dezenas de usuários simultaneamente, enquanto hoje o **Windows 11** exige **mínimo de 4 GB** e, na prática, muitos consideram **8 GB ou 16 GB** o ideal para bom desempenho.

Gerenciamento de Memória – Memória virtual

Com isso, surgiram duas necessidades principais:

- Executar programas **maiores do que a memória disponível**.
- Suportar **vários programas ao mesmo tempo**, mesmo que, juntos, excedam o tamanho total da memória física.

Uma alternativa seria fazer **troca constante de processos entre memória e disco**, mas isso é muito lento. Um disco comum (como um SATA) transfere alguns **centenas de MB por segundo**, o que significa que carregar ou salvar um programa de **1 GB** pode levar **vários segundos**.

Gerenciamento de Memória – Memória virtual

Nos anos 1960, a solução encontrada foi dividir os programas em módulos menores, chamados de sobreposições (***overlays***). Quando o programa era iniciado, apenas o gerenciador de sobreposições era carregado.

Funciona assim:

1. O **programa era dividido em partes menores**, chamadas **módulos ou sobreposições**.
2. Quando o programa começava a rodar, **apenas o gerenciador de sobreposições** era carregado na memória.
3. Esse gerenciador trazia para a memória **somente o módulo necessário naquele momento** (por exemplo, a sobreposição 0).
4. Quando o programa precisava executar outra parte, o gerenciador **descartava a sobreposição atual** e **carregava outra** (sobreposição 1, 2, etc.) a partir do **disco**.

Gerenciamento de Memória – Memória virtual

Em resumo, o sistema **trocava dinamicamente partes do programa na memória, permitindo que aplicações grandes funcionassem em máquinas com pouca RAM.**
Essa técnica foi um passo importante na evolução do gerenciamento de memória antes do surgimento da memória virtual.

Gerenciamento de Memória – Memória virtual

Embora o sistema operacional fizesse a parte mecânica de carregar e remover as sobreposições, o programador precisava dividi-las manualmente, o que era um trabalho cansativo, demorado e propenso a erros. Por isso, logo surgiu a ideia de automatizar o processo.

Em 1961, foi apresentada uma solução chamada **memória virtual**.

Gerenciamento de Memória – Memória virtual

A ideia é simples e eficiente:

- Cada programa tem seu próprio espaço de endereçamento, dividido em blocos chamados **páginas**.
- Cada página contém uma **sequência contínua** de endereços.
- As páginas são mapeadas na memória física, mas **nem todas precisam estar na memória ao mesmo tempo.**
- Quando o programa tenta acessar uma página que já está na memória, o hardware faz o acesso diretamente.
- Quando o programa tenta acessar uma página que não está carregada, o sistema operacional é avisado, busca a página no disco e repete a instrução que falhou.

Gerenciamento de Memória – Memória virtual

A memória virtual é uma evolução da ideia dos registradores base e limite.

Em vez de permitir apenas a realocação de partes específicas, ela permite mapear todo o espaço de endereçamento em pequenas unidades (as páginas), tornando o sistema muito mais flexível.

Essa técnica funciona especialmente bem em sistemas multiprogramados, nos quais vários programas compartilham a memória.

Enquanto um programa espera que uma parte sua seja carregada do disco, a CPU pode ser usada por outro processo, garantindo maior eficiência e desempenho.

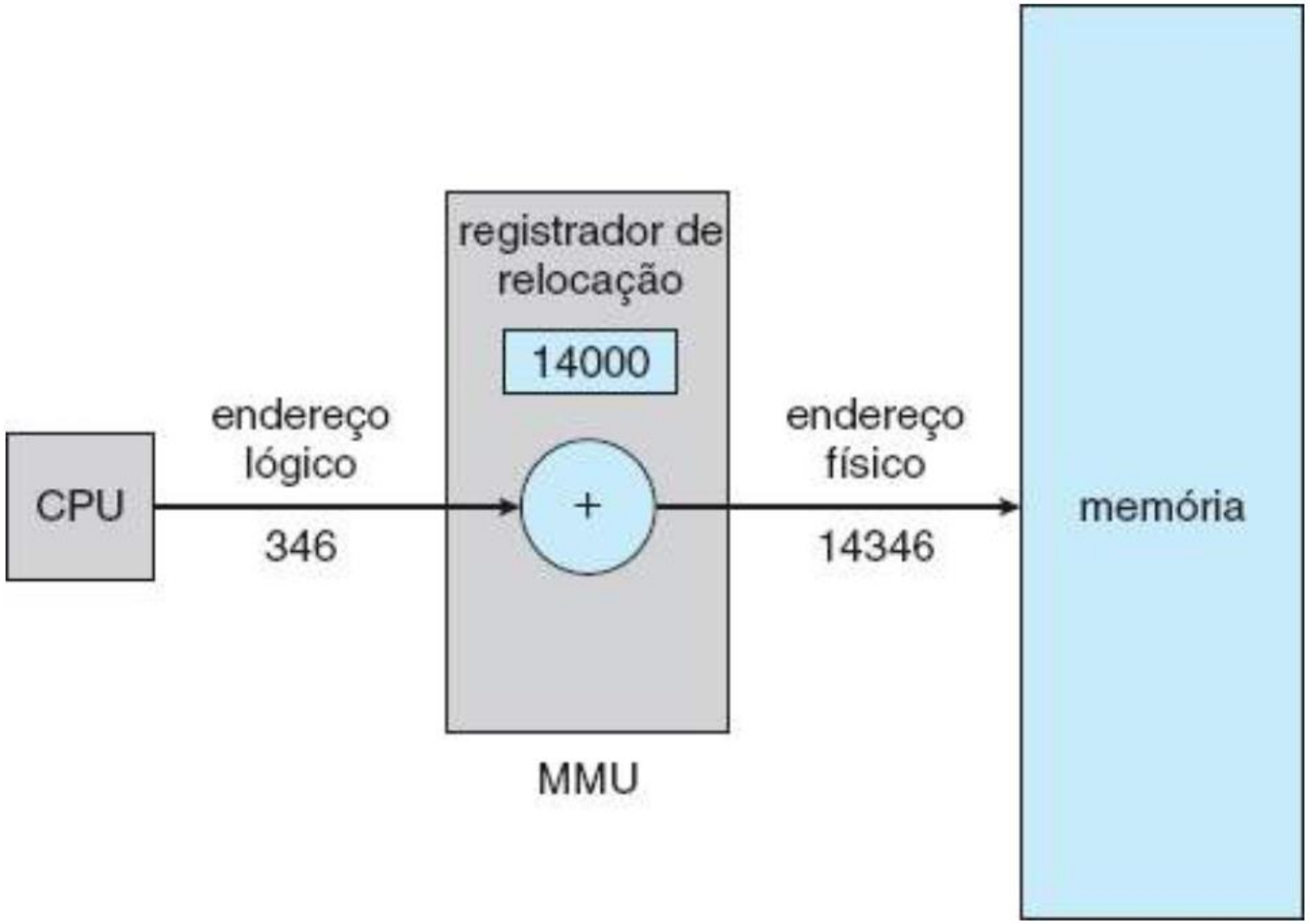
Espaço de Endereçamento Lógico x Espaço de Endereçamento Físico

Um endereço gerado pela CPU é chamado de endereço lógico, enquanto o endereço utilizado pela unidade de memória — aquele efetivamente carregado no registrador de endereços — é conhecido como endereço físico.

O espaço de endereçamento lógico é o conjunto de todos os endereços lógicos gerados por um programa, enquanto o espaço de endereçamento físico é o conjunto dos endereços reais correspondentes na memória. Portanto, quando a vinculação ocorre em tempo de execução, os espaços de endereçamento lógico e físico são distintos.

A conversão dos endereços virtuais em endereços físicos é feita durante a execução por um componente de hardware chamado unidade de gerenciamento de memória (MMU — Memory Management Unit).O MMU se localiza integrado a CPU.

Espaço de Endereçamento Lógico x Espaço de Endereçamento Físico



Espaço de Endereçamento Lógico x Espaço de Endereçamento Físico

0x00001000 (virtual)

0xABCD1000 (físico)

CPU: é o processador
MMU: é a unidade dentro da CPU que traduz endereços virtuais em endereços físicos e gerencia o uso da memória.
TLB: é uma pequena memória cache dentro da MMU que armazena traduções recentes para acelerar o acesso à memória.

Gerenciamento de Memória – Paginação

A maioria dos sistemas que usam **memória virtual adota uma técnica chamada paginação**.

Todo programa, ao ser executado, faz referência a endereços de memória.

Esses endereços usados pelo programa são chamados de endereços virtuais, e **o conjunto deles forma o espaço de endereçamento virtual.**

Em computadores sem memória virtual, esses endereços são enviados diretamente ao barramento de memória, acessando o endereço físico correspondente.

Gerenciamento de Memória – Paginação

Nos sistemas com memória virtual, o endereço virtual não vai direto para a memória.

Ele é enviado antes para uma unidade especial chamada **MMU (Memory Management Unit) — Unidade de Gerenciamento de Memória.**

A MMU tem a função de traduzir cada endereço virtual em um endereço físico.

Desse modo, a MMU mapeia dinamicamente os endereços virtuais para endereços físicos, conforme a tabela de páginas definida pelo sistema.

Gerenciamento de Memória – Paginação

Objetivo

Evitar o desperdício de memória

- Das partições fixas: Reduzindo a quantidade de fragmentações internas
- Das partições variáveis: Maximiza o uso de espaços de memória
- Usando MMU para definir o tamanho das páginas que serão realocadas e utilizadas

Como: Permite que processos utilizem áreas não contíguas de memória, ou seja, áreas dispersas em disco que não estão alocadas.

Gerenciamento de Memória – Segmentação

A visão que o usuário tem da memória não corresponde à memória física real.
Isso é igualmente verdadeiro para a visão que o programador tem da memória.
Na verdade, lidar com a memória em termos de suas propriedades físicas é inconveniente tanto para o sistema operacional quanto para o programador.

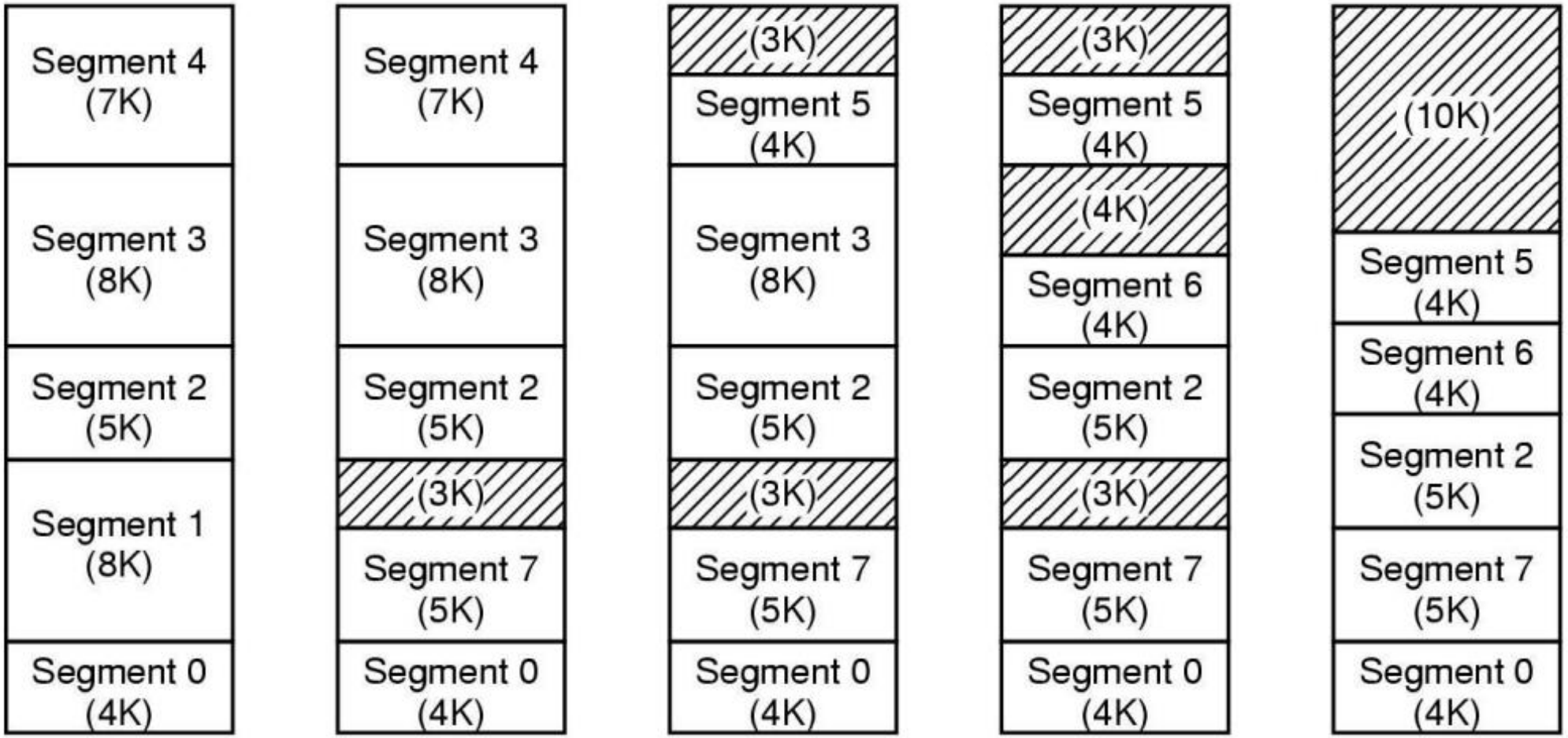
Mas e se o hardware pudesse fornecer um mecanismo de memória que mapeasse a visão do programador para a memória física real? O sistema teria mais liberdade para gerenciar a Memória enquanto o programador teria um ambiente de programação mais natural.
A segmentação fornece esse mecanismo.

Gerenciamento de Memória – Segmentação

Segmentação é um processo virtual que cria espaços de endereço de vários tamanhos em um sistema computacional, chamados segmentos. Cada segmento é um espaço de endereço virtual diferente que corresponde diretamente aos objetos do processo.

Cada segmento tem um nome e um tamanho. Os endereços especificam tanto o nome do segmento quanto o deslocamento dentro do segmento.

Gerenciamento de Memória – Segmentação



Gerenciamento de Memória – Tabela de páginas

Em uma implementação simples, o processo de converter endereços virtuais em endereços físicos funciona da seguinte maneira: o endereço virtual é dividido em duas partes — o número da página virtual (bits mais significativos) e o deslocamento (bits menos significativos).

O número da página virtual é usado como índice na tabela de páginas, onde se encontra a entrada correspondente a essa página. Essa entrada indica o número do quadro físico (caso a página esteja carregada na memória).

Em seguida, o número do quadro é combinado com o deslocamento, substituindo o número da página virtual, formando assim o endereço físico que será enviado à memória.

Gerenciamento de Memória – Tabela de páginas

Simplificando:

Cada **endereço virtual** (aquele que o programa usa) é dividido em duas partes:

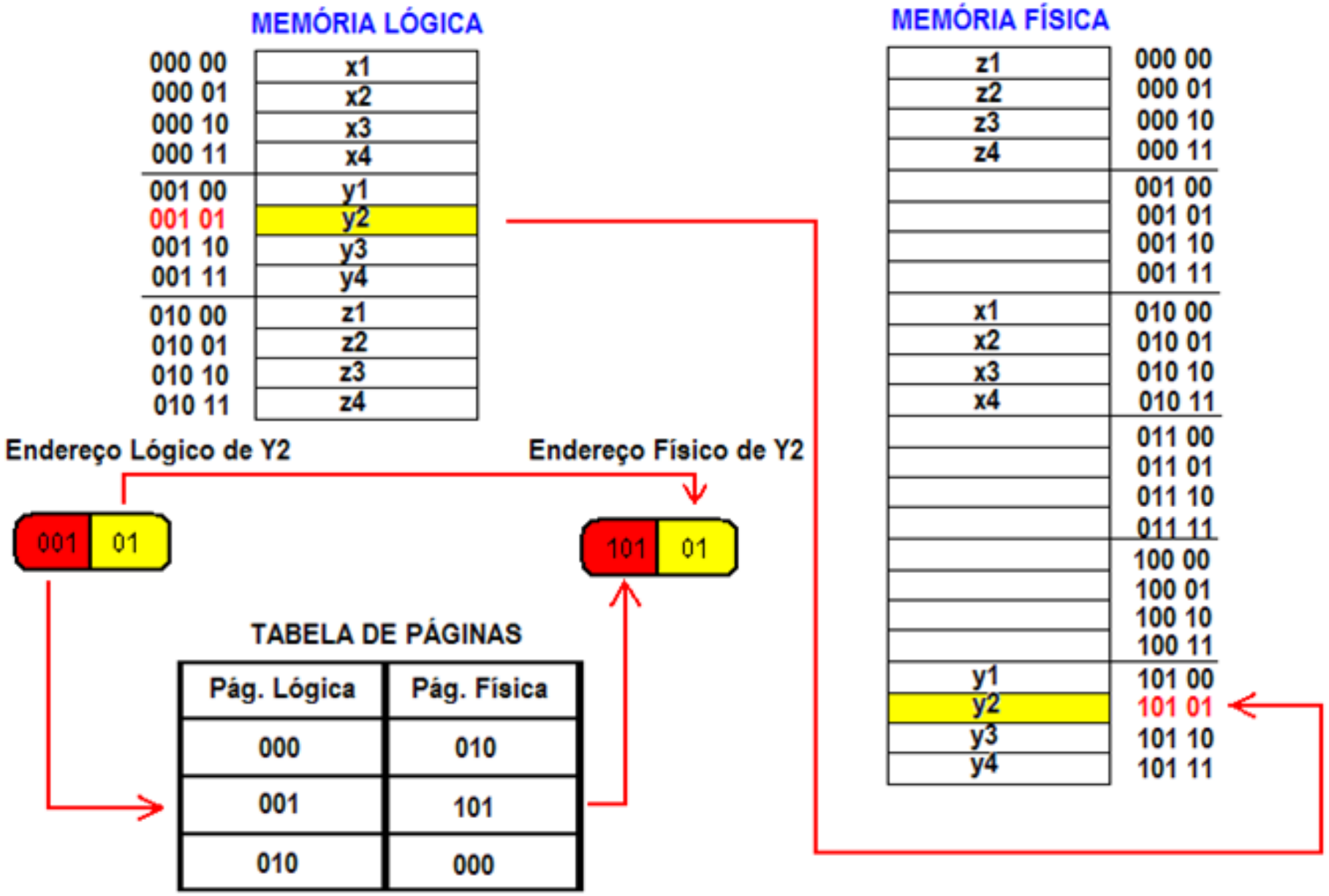
1. **Número da página virtual**: diz *qual página* queremos acessar.
2. **Deslocamento**: diz *onde exatamente* dentro dessa página está a informação.

Gerenciamento de Memória – Tabela de páginas

Simplificando:

Essa tabela guarda a correspondência entre:

- páginas virtuais (que o programa vê)
- quadros físicos (os locais reais na memória RAM).



MECANISMO BÁSICO DE PAGINAÇÃO

1- O endereço lógico de Y2 (001 01) é dividido em **página lógica (001)** e **deslocamento (01)**.

2- A tabela de páginas mostra que essa página lógica corresponde à **página física 101**.

3- Assim, o endereço físico final é **101 01**, resultado da combinação entre a página física e o deslocamento.

O **gerenciamento de memória** é a parte do sistema operacional responsável por **controlar e organizar o uso da memória principal (RAM)**.

Seu objetivo é garantir que:

- Cada processo tenha espaço suficiente para ser executado;
- Os programas não interfiram na memória uns dos outros;
- O uso da memória seja eficiente e equilibrado.

Principais Funções

1.Alocação de memória
O SO decide **quanto e onde** cada processo vai usar a memória.

1. Pode ser **contígua** (em blocos contínuos) ou **não contígua** (páginas espalhadas).

2.Proteção e isolamento
Impede que um processo acesse áreas de memória de outro, garantindo **segurança e estabilidade**.

3.Gerenciamento de espaço livre
O SO mantém controle das áreas livres e ocupadas da RAM para poder reutilizá-las.

4.Troca (Swapping)
Quando a RAM fica cheia, o sistema pode mover processos inativos para o **disco (memória secundária)** e liberar espaço para outros.

5.Tradução de endereços
Converte **endereços virtuais** (usados pelos programas) em **endereços físicos** (locais reais na RAM).

Técnicas Importantes

a) **Particionamento**
   Divide a memória em partes fixas ou variáveis para alocar processos.
b) **Paginação**
   Divide a memória em **páginas** (tamanho fixo) e o espaço físico em **quadros**. Facilita a alocação e evita fragmentação externa.
c) **Segmentação**
   Divide o espaço em **segmentos lógicos** (ex.: código, pilha, dados). Ajuda na organização e proteção.

Memória virtual
Permite que o sistema execute programas **maiores do que a RAM disponível**, armazenando partes temporariamente no disco.

Objetivos Principais

a) **Eficiência:** usar a memória de forma inteligente.
b) **Segurança:** evitar que processos interfiram entre si.
c) **Desempenho:** reduzir o tempo de acesso e troca de dados.
d) **Flexibilidade:** permitir a execução de múltiplos programas ao mesmo tempo.

Objetivos Principais

a) **Eficiência:** usar a memória de forma inteligente.
b) **Segurança:** evitar que processos interfiram entre si.
c) **Desempenho:** reduzir o tempo de acesso e troca de dados.
d) **Flexibilidade:** permitir a execução de múltiplos programas ao mesmo tempo.