Aula 08: Arquitetura de Computadores – Suporte ao sistema operacional

Prof. Hugo Puertas de Araújo
hugo.puertas@ufabc.edu.br
Sala: 509-2 (5º andar / Torre 2)

Arquitetura de Computadores

Objetivos de aprendizagem

- Resumir as funções-chave de um sistema operacional (SO).
- Discutir a evolução dos sistemas operacionais dos sistemas em lote (batches) simples para os sistemas complexos modernos.
- Explicar o escalonamento de longo, médio e curto prazo.
- Compreender a razão para particionar a memória.
- Analisar vantagens relativas da paginação e da segmentação.
- Definir a memória virtual.

Exercício?

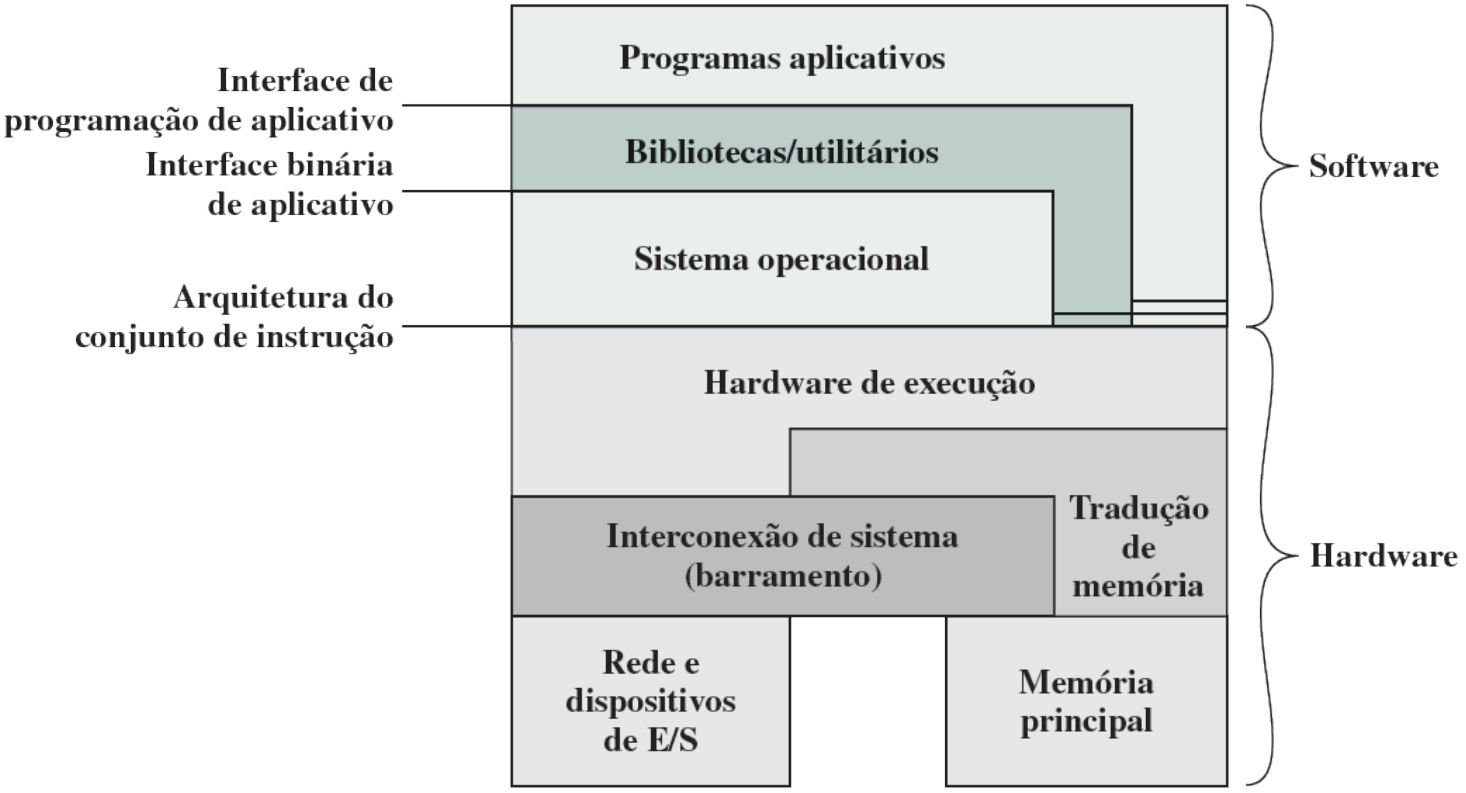
- O que é e pra que serve um Sistema Operacional?

Visão geral do sistema operacional

- Um SO é um programa que controla a execução dos programas aplicativos e atua como uma interface entre o usuário e o hardware do computador.
- Ele pode ser imaginado como tendo dois objetivos:
- **Conveniência**: um SO torna um computador mais conveniente para uso.
- **Eficiência**: um SO permite que os recursos do sistema computacional sejam usados de uma maneira eficiente.

O sistema operacional como uma interface usuário/computador

- Hardware de computador e estrutura de software:



O sistema operacional como uma interface usuário/computador

- O SO normalmente oferece serviços nas seguintes áreas:
  - Criação de programas
  - Execução do programa
  - Acesso aos dispositivos de E/S
  - Acesso controlado aos arquivos
  - Acesso ao sistema
  - Detecção e resposta a erros
  - Contabilização

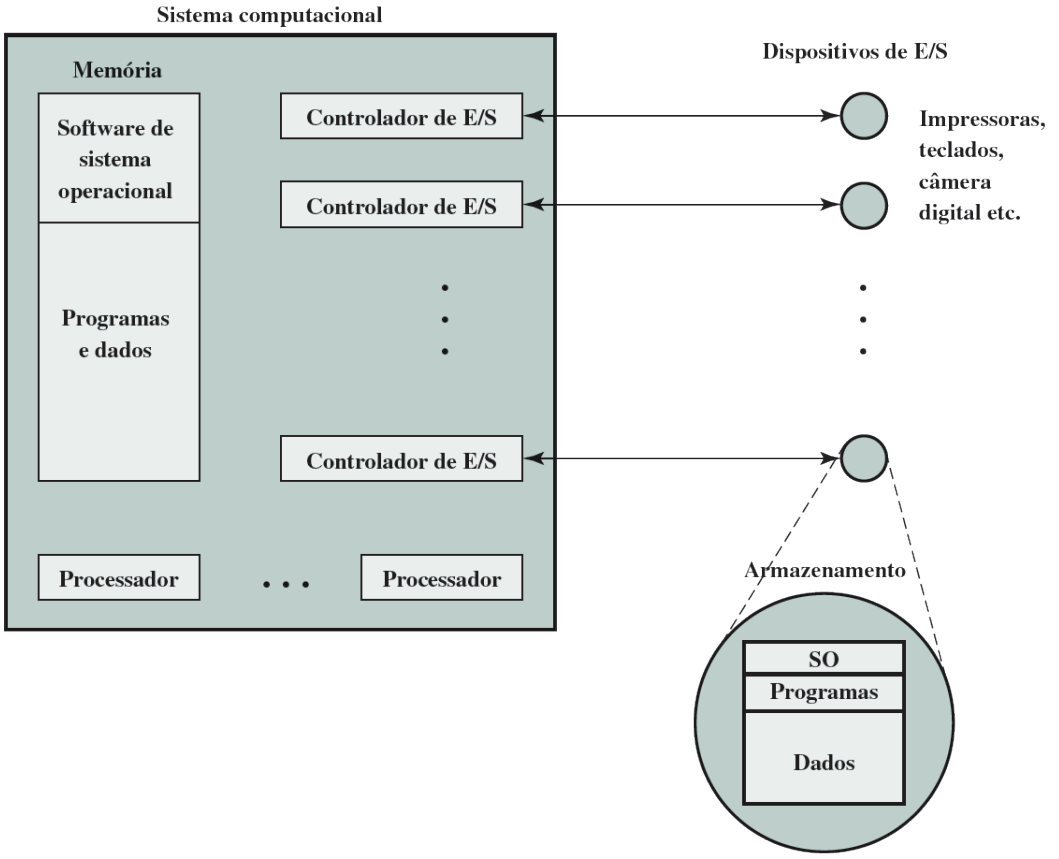
O sistema operacional como uma interface usuário/computador

- As três interfaces-chave em um sistema computacional comum são:
  - Arquitetura do conjunto de instruções (ISA — do inglês, Instruction Set Architecture)
  - Interface binária de aplicativo (ABI — do inglês, Application Binary Interface)
  - Interface de programação de aplicação (API — do inglês, Application Programming Interface)

O sistema operacional como gerenciador de recursos

- O **SO funciona da mesma maneira que o software comum** do computador; ou seja, ele é um programa executado pelo processador.
- O SO muitas vezes abre mão do controle e deve **depender do processador para permitir que ele readquira o controle**.
- Ele **direciona o processador no uso dos outros recursos** do sistema e na temporização de sua execução dos outros programas.
- Mas, para que o processador faça essas coisas, **ele deve deixar de executar o programa do SO** e executar outros programas.

O sistema operacional como gerenciador de recursos



Tipos de sistemas operacionais

- Em um **sistema interativo**, o usuário/programador interage diretamente com o computador.
- Um **sistema em lote** é o oposto do interativo:
  - O programa de um usuário é mantido junto com programas de outros usuários e submetido por um operador de computador.
- Uma dimensão independente especifica se o sistema emprega **multiprogramação** ou não.
- A alternativa é um sistema de **uniprogramação**, que trabalha apenas com um programa de cada vez.

Sistemas antigos

- Com os computadores mais antigos, desde o final da década de 1940 até meados dos anos 1950, o programador interagia diretamente com o hardware do computador; não havia um SO.
- Os programas no código do processador eram carregados por meio de um dispositivo de entrada.
- Se um erro interrompesse o programa, a condição era indicada por lâmpadas.
- Se o programa terminasse normalmente, a saída aparecia na impressora.

Sistemas em lote simples (monitor)

- Com esse tipo de sistema, também chamado de monitor, o usuário não tem mais acesso direto ao processador.
- Em vez disso, ele submete o **job** nos cartões ou em fita a um operador de computador, que dispõe os jobs sequencialmente e coloca o lote inteiro em um dispositivo de entrada, para uso pelo monitor.
- O monitor lê os jobs um de cada vez pelo dispositivo de entrada.
- Quando o job termina, ele retorna o controle ao monitor, que imediatamente lê o job seguinte.

Sistemas em lote simples (monitor)

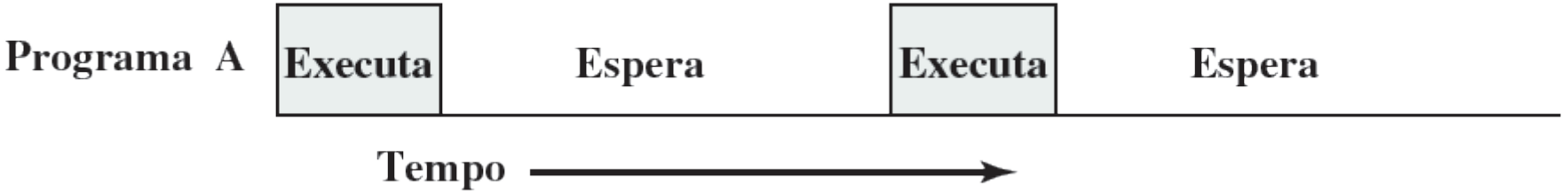
- Agora, considere essa sequência do ponto de vista do processador.
- Quando um job tiver sido lido, o processador encontrará no monitor uma instrução de desvio que instrui o processador a continuar a execução no início do programa do usuário.
- O processador, então, executará a instrução no programa do usuário até encontrar um final ou uma condição de erro.
- Qualquer um desses eventos faz o processador buscar sua próxima instrução no programa monitor.

Sistemas em lote simples (monitor)

- E o tempo de preparação?
- O monitor trata disso também.
- Com cada job, as instruções são incluídas em uma linguagem de controle de job (JCL — do inglês, Job Control Language).
- Esse é um tipo especial de linguagem de programação, usada para fornecer instruções ao monitor.
- O compilador traduz o programa do usuário para um código objeto, que é armazenado na memória ou no armazenamento em massa.

Sistemas em lote multiprogramados

- Exemplo de uniprogramação:



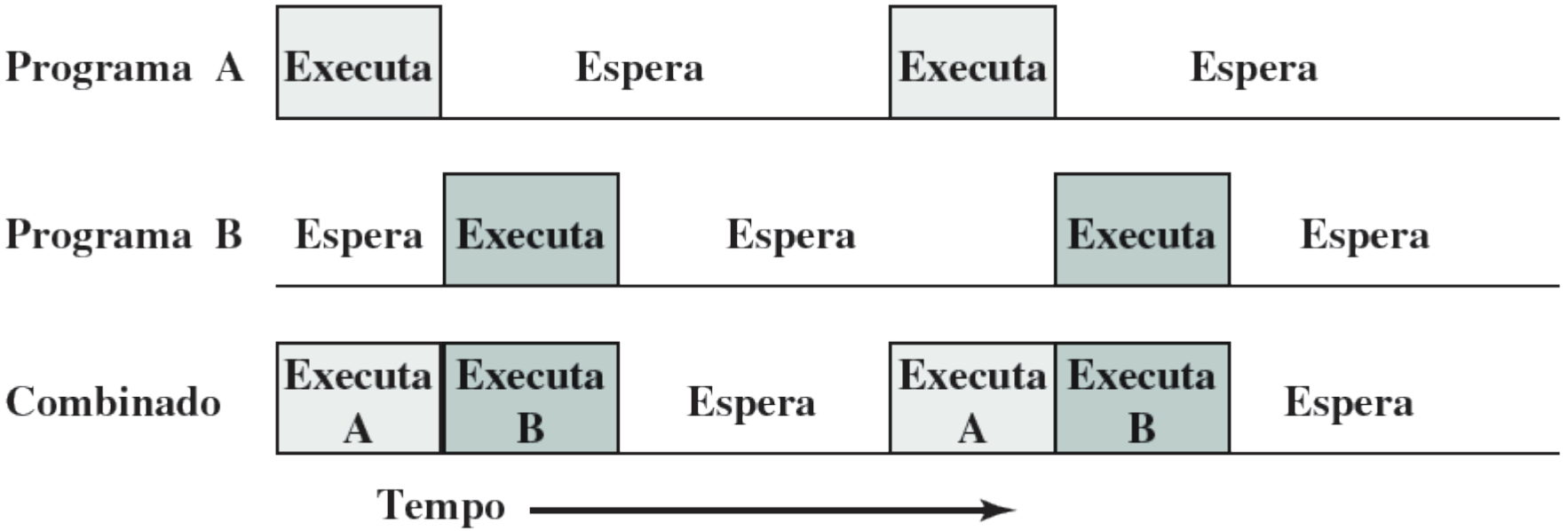
Sistemas em lote multiprogramados

- Exemplo de uniprogramação:

Exercício

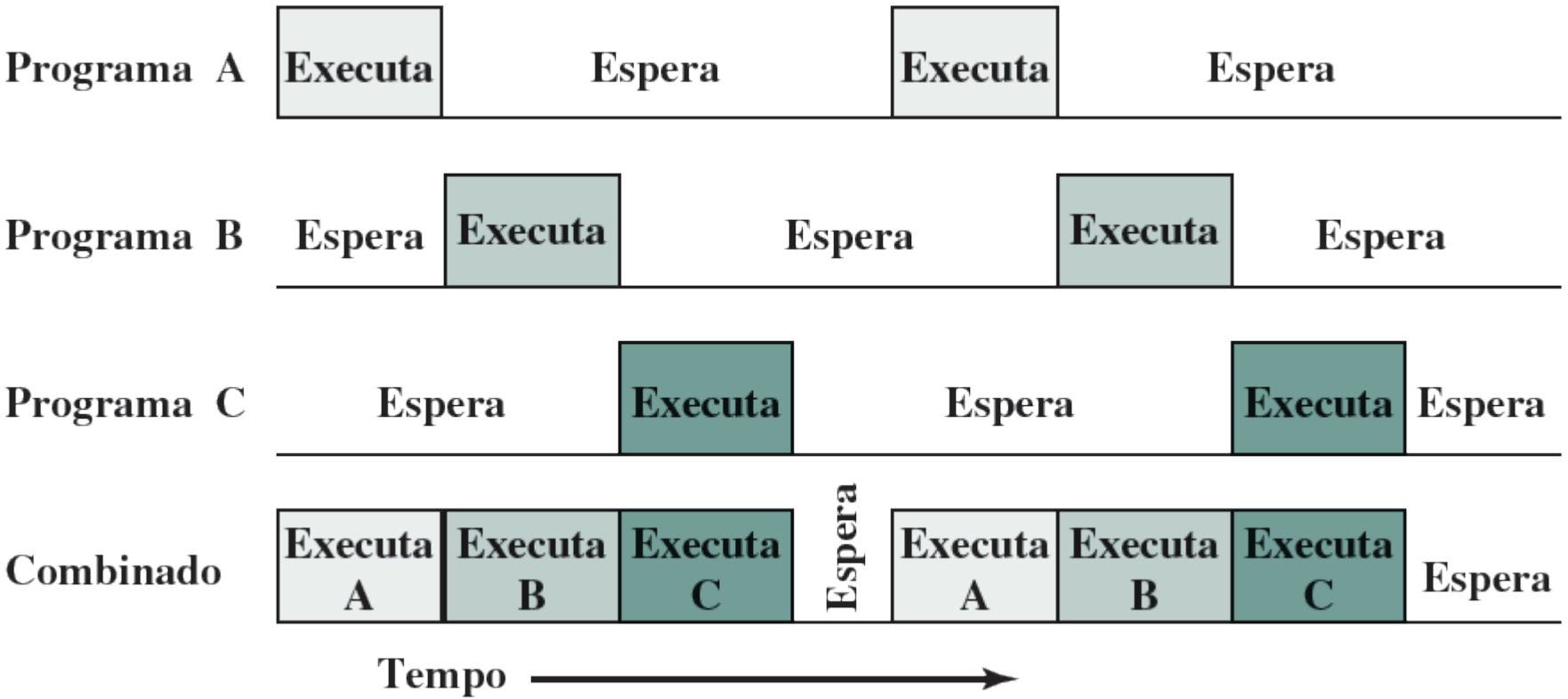
Sistemas em lote multiprogramados

- Exemplo de multiprogramação com dois programas:



Sistemas em lote multiprogramados

- Exemplo de multiprogramação com três programas:



Sistemas em lote multiprogramados

Exemplo

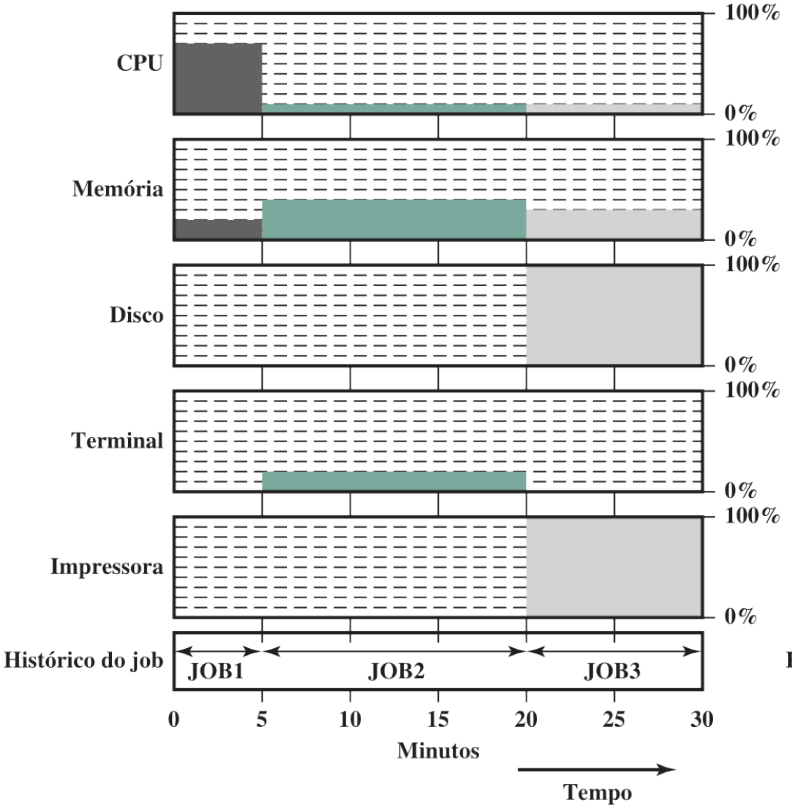
- Efeitos da multiprogramação sobre a utilização de recursos:

	Uniprogramação	Multiprogramação
Uso de processador (%)	20	40
Uso de memória (%)	33	67
Uso de disco (%)	33	67
Uso de impressora (%)	33	67
Tempo decorrido (min)	30	15
Taxa de *throughput* (jobs/h)	6	12
Tempo médio de resposta (min)	18	10

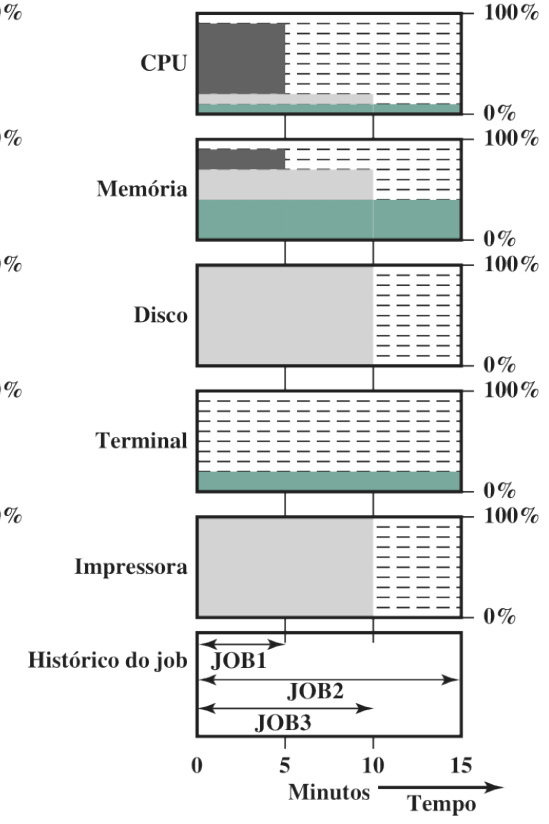
Sistemas em lote multiprogramados

Exemplo

- Efeitos da multiprogramação sobre a utilização de recursos:



(a) Uniprogramação



(b) Multiprogramação

Sistemas de tempo compartilhado

- Em um sistema de tempo compartilhado (time-sharing system), vários usuários acessam o sistema simultaneamente por meio de terminais, com o SO intercalando a execução de cada programa do usuário em um curto intervalo de tempo ou quantum de computação.
- Se houver **n** usuários ativamente solicitando serviço de uma só vez, cada usuário só verá uma média de **1/n** da velocidade efetiva do computador, sem contar a sobrecarga do SO.
- Dado o tempo de reação relativamente lento do ser humano, o tempo de resposta em um sistema corretamente projetado deverá ser comparável ao de um computador dedicado.

Sistemas de tempo compartilhado

- Multiprogramação em lote versus tempo compartilhado:

	Multiprogramação em lote	Tempo compartilhado
Objetivo principal	Maximizar uso do processador	Minimizar tempo de resposta
Origem das diretivas ao sistema operacional	Comandos da JCL fornecidos com a tarefa	Comandos digitados no terminal

Usuários

Job Control Language

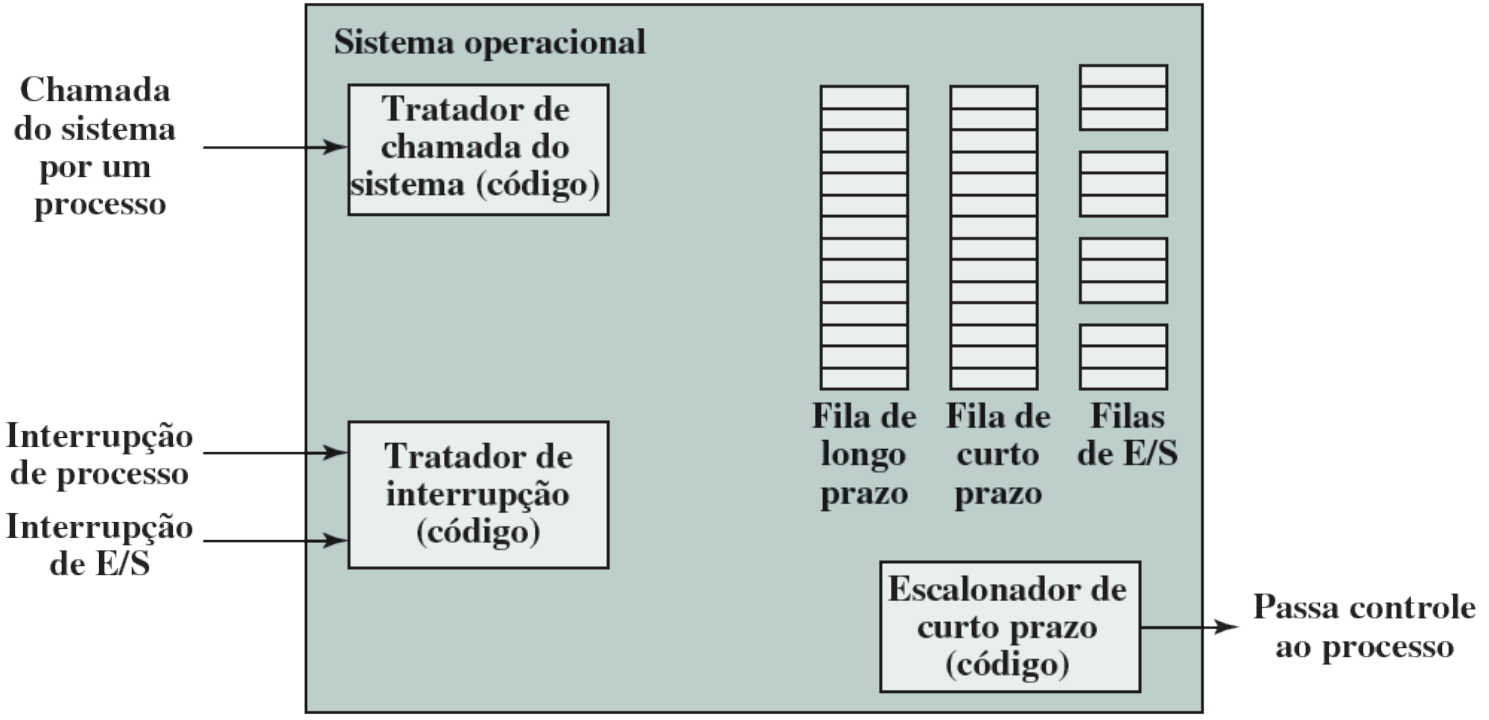
Escalonamento

- Tipos de escalonamento:

Escalonamento de longo prazo	A decisão de acrescentar ao conjunto de processos a serem executados
Escalonamento de médio prazo	A decisão de acrescentar ao número de processos que estão parcial ou totalmente na memória principal
Escalonamento de curto prazo	A decisão sobre qual processo disponível será executado pelo processador
Escalonamento de E/S	A decisão sobre qual solicitação de E/S pendente do processo será tratada por um dispositivo de E/S disponível

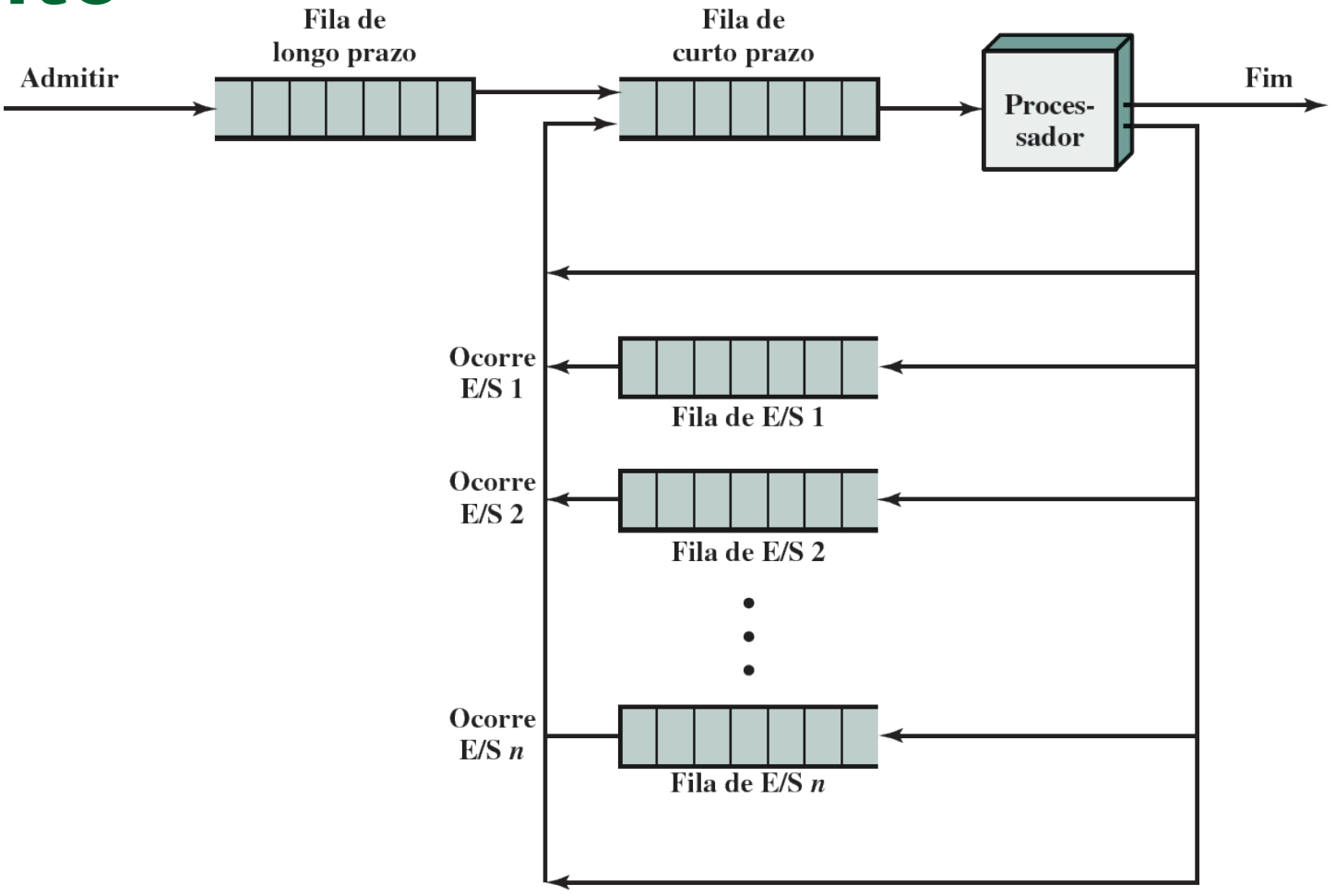
Escalonamento

- Principais elementos de um sistema operacional para multiprogramação:



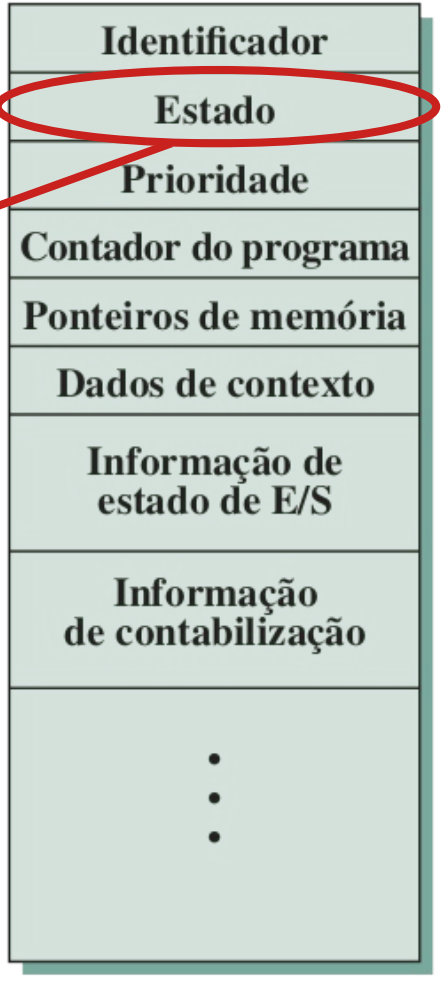
Escalonamento

- Diagrama de filas do escalonamento de processador:



Escalonamento

- Modelo de processo & Bloco de controle:



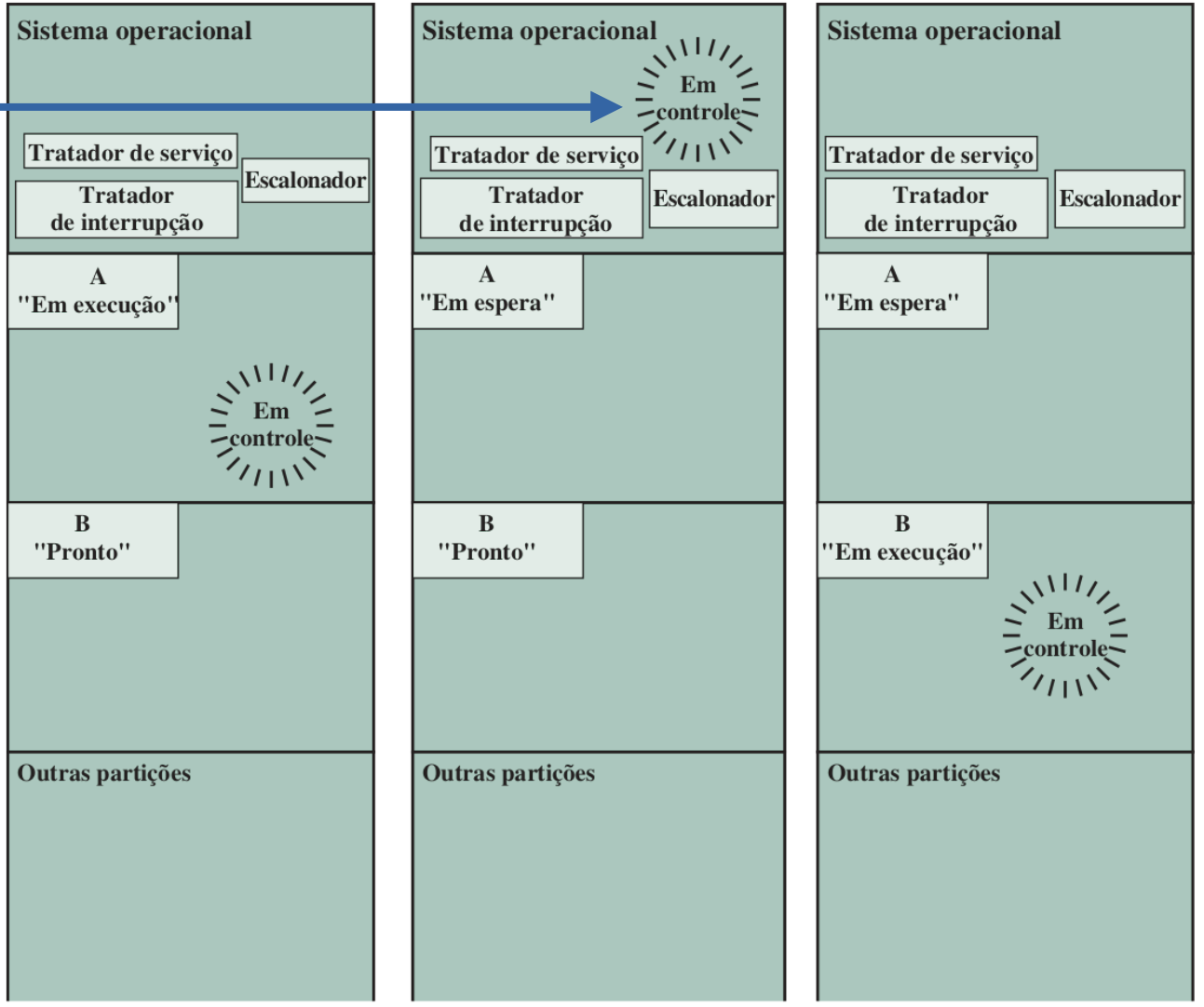
Escalonamento

Motivos p/ controle retornar ao SO:

1. O processo A emite uma chamada de serviço (por exemplo, uma requisição de E/S) ao SO. A execução de A é suspensa até que essa chamada seja satisfeita pelo SO.

2. O processo A causa uma interrupção, que é um sinal gerado pelo hardware ao processador. Quando esse sinal é detectado, o processador deixa de executar A e transfere a execução ao tratador de interrupção do SO. Diversos eventos relacionados a A causarão uma interrupção. Um exemplo é um erro, como a tentativa de executar uma instrução privilegiada. Outro exemplo é um tempo limite esgotado (timeout); para impedir que qualquer processo monopolize o processador, cada processo só recebe a atenção do processador por um curto período de cada vez.

3. Algum evento não relacionado ao processo A que requeira atenção causa uma interrupção. Um exemplo é o término de uma operação de E/S.

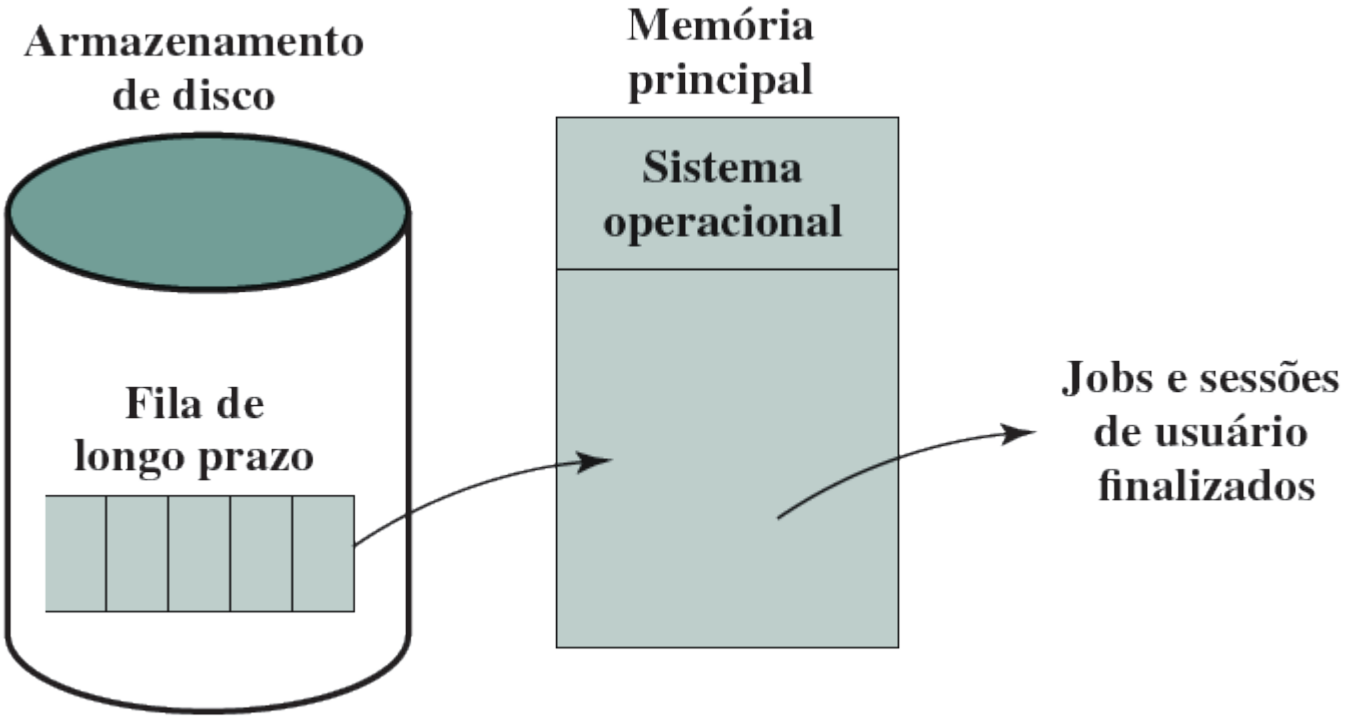


Gerenciamento de memória

- Em um **sistema de multiprogramação**, a parte do “usuário” da memória é subdividida para acomodar diversos processos.
- A tarefa de **subdivisão é executada dinamicamente pelo SO** e é conhecida como **gerenciamento de memória**.
- Se apenas alguns processos estiverem na memória, então, em grande parte do tempo, todos os processos estarão esperando pela E/S e o processador estará ocioso.
- Desse modo, a memória precisa ser alocada de modo eficiente para colocar o máximo possível de processos na memória.

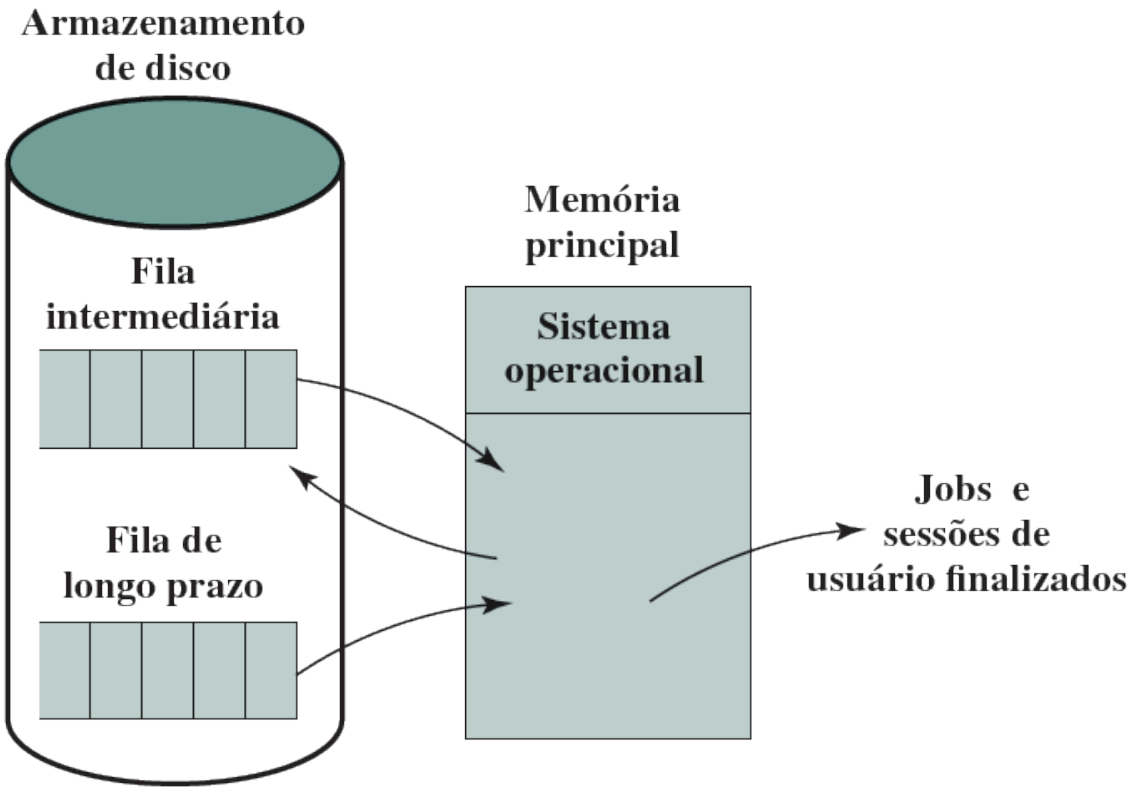
Troca de processos na memória (swapping)

- O uso da troca de processo na memória – Escalonamento de job simples:



Troca de processos na memória (swapping)

- O uso da troca de processo na memória – Troca de processo na memória (swapping):



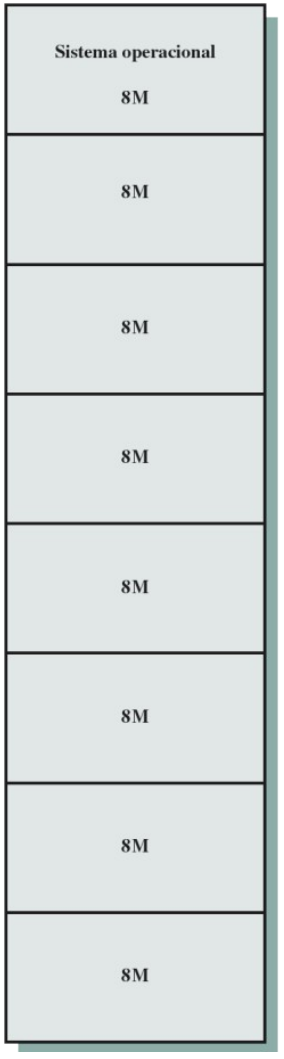
Particionamento

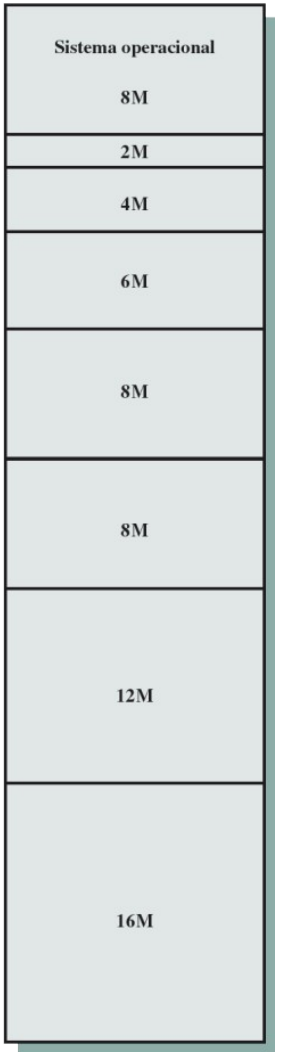
- O esquema mais simples para o particionamento da memória disponível é usar **partições de tamanho fixo**.
  - Observe na figura a seguir que, embora as partições tenham tamanho fixo, elas não precisam ter o mesmo tamanho.
  - Quando um processo é trazido para a memória, ele é colocado na menor partição possível que o poderá manter.
- Uma técnica mais eficiente é usar partições de tamanho variável.
  - Quando um processo é levado para a memória, ele recebe exatamente a quantidade de memória exigida, e nada mais.

Particionamento

- Exemplo de particionamento fixo de uma memória de 64 MB:

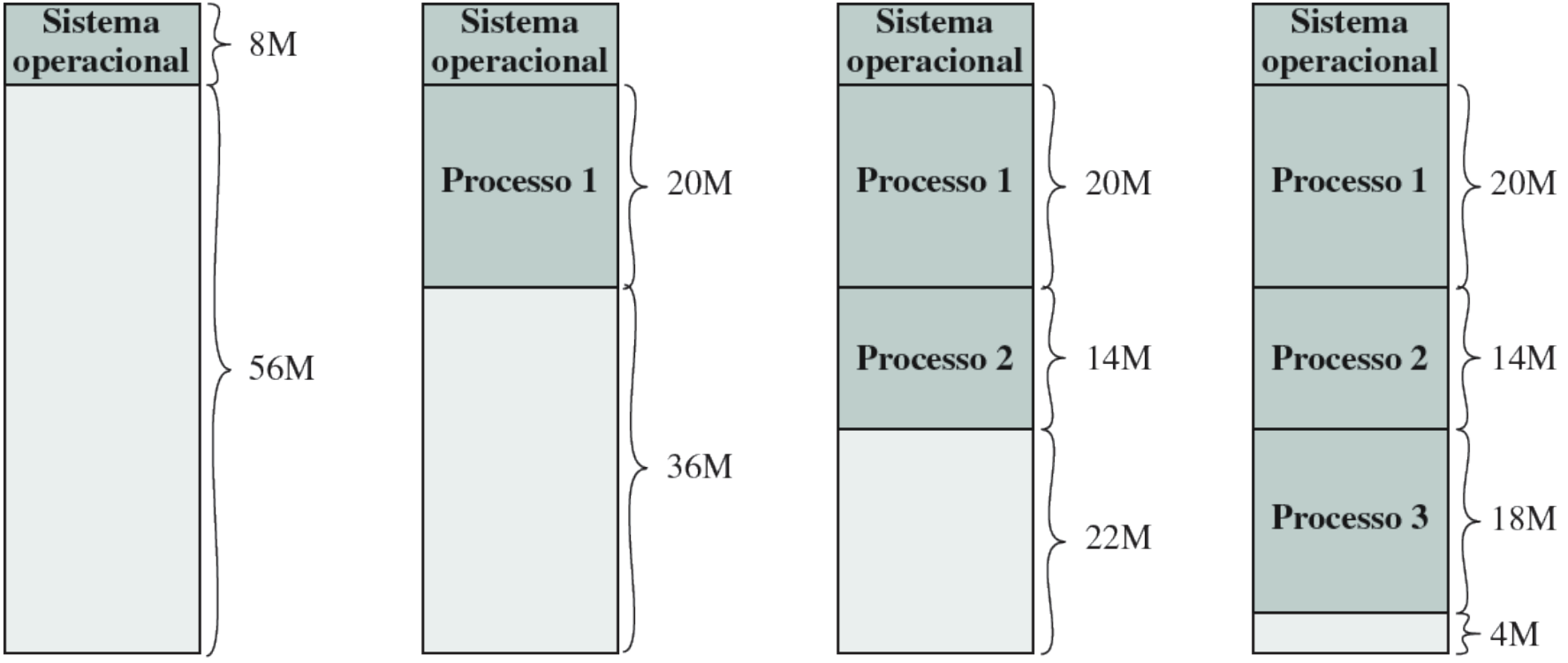
Partições de tamanhos iguais vs tamanhos desiguais





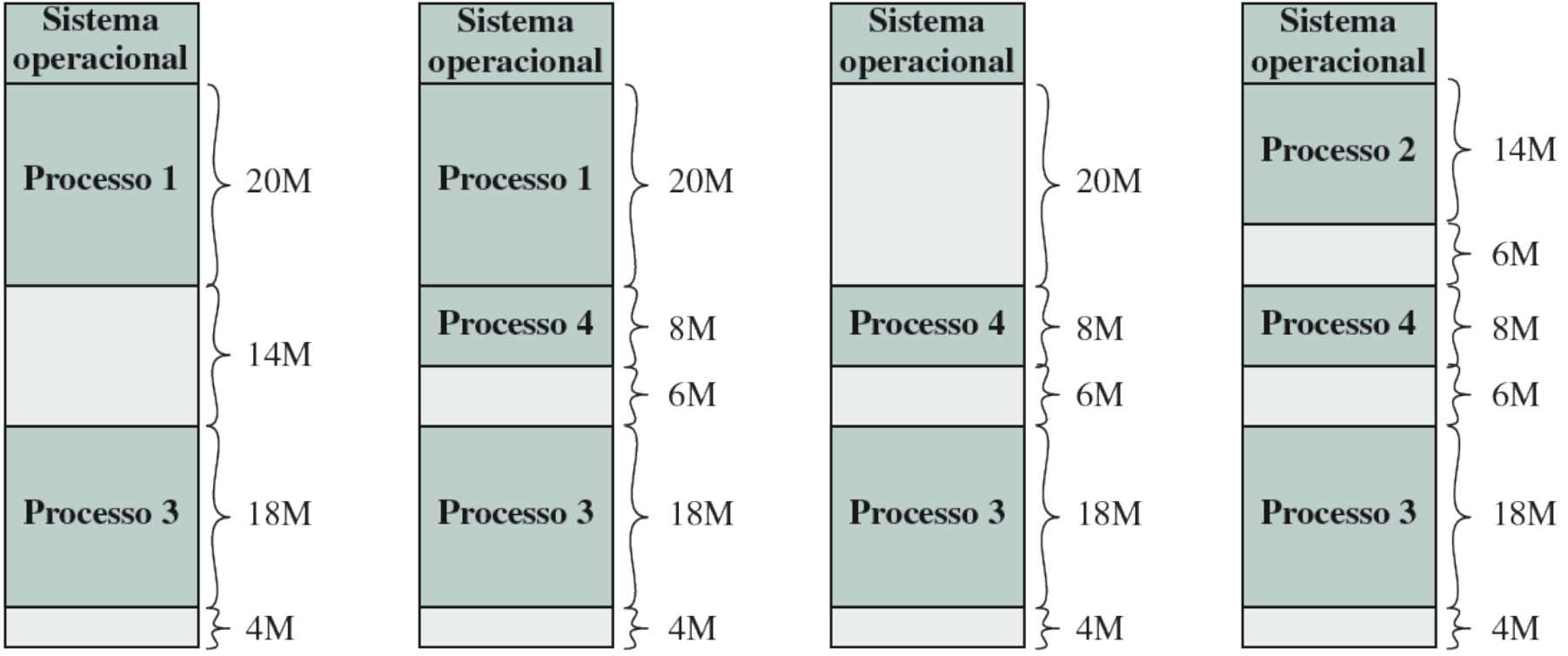
Particionamento

- O efeito do particionamento dinâmico:



Particionamento

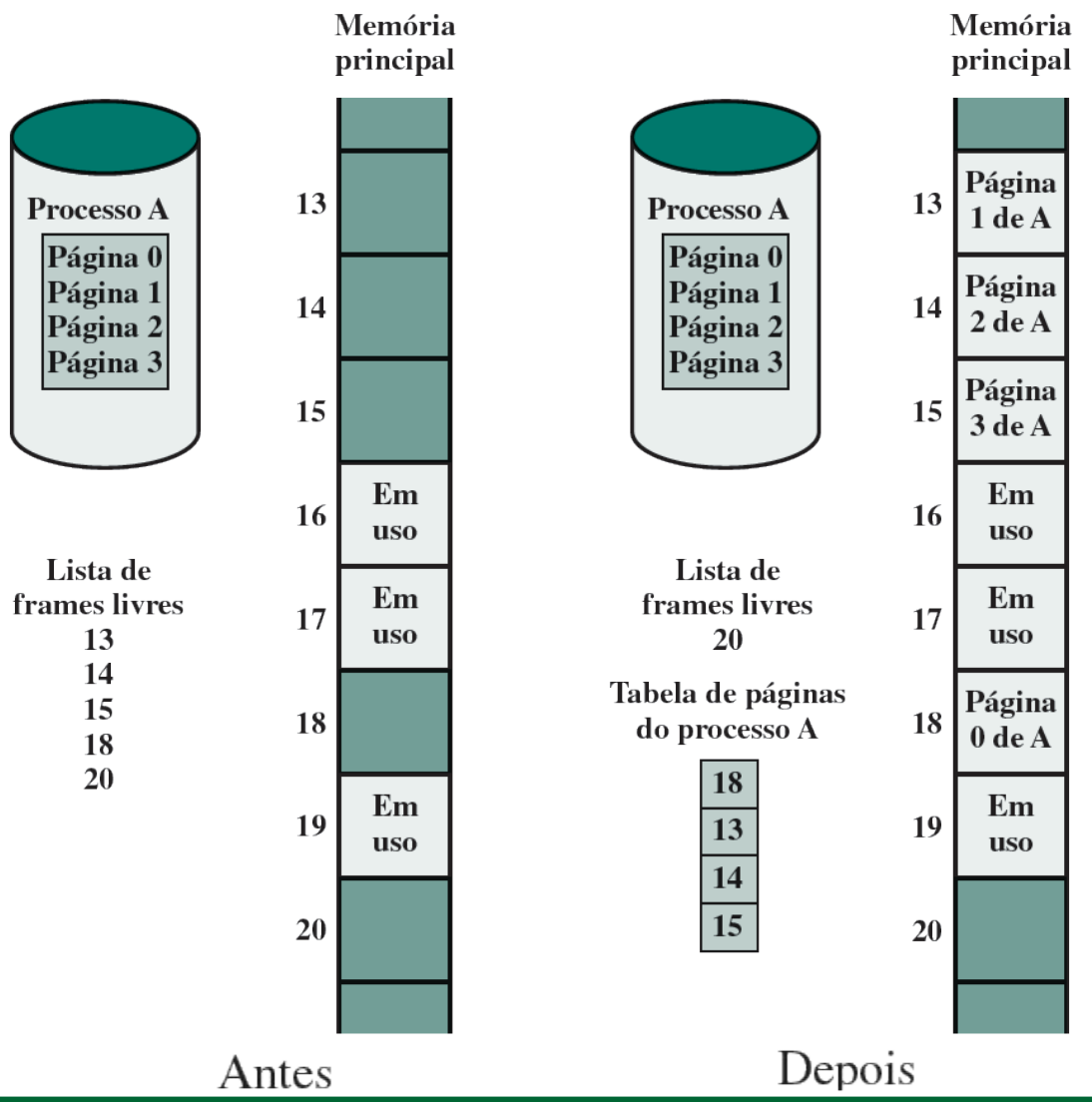
- O efeito do particionamento dinâmico:



Paginação

- Partições desiguais de tamanho fixo e de tamanho variável são ineficazes no uso da memória.
- Então, os **pedaços de um programa**, conhecidos como **páginas**, poderiam ser atribuídos aos pedaços disponíveis de memória, conhecidos como **frames**, ou frames de página.
- No máximo, então, o espaço desperdiçado na memória para esse processo é uma fração da última página.
- A figura a seguir mostra um exemplo do uso de páginas e frames.

Paginação

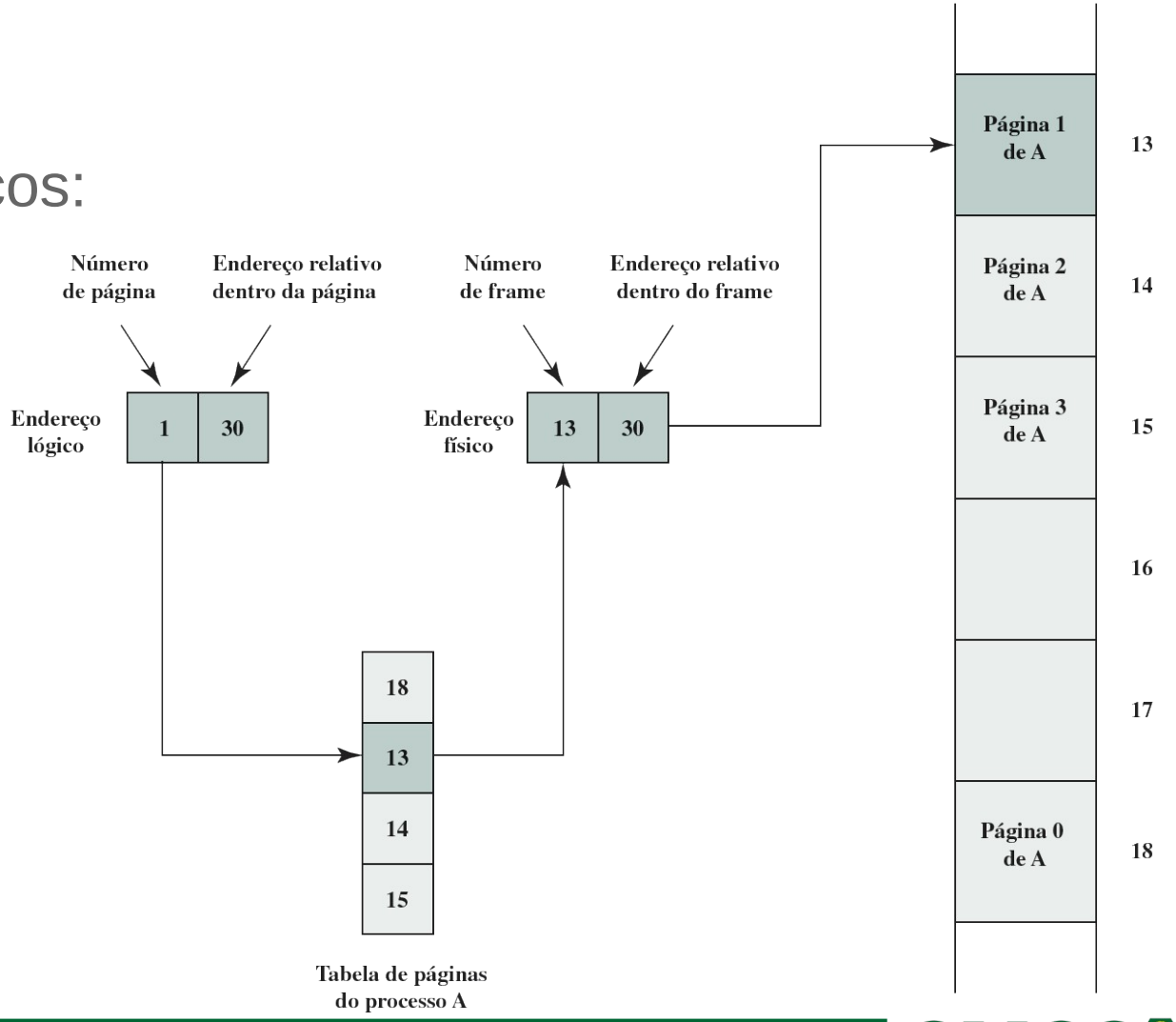

Memória principal
Processo A
Página 0
Página 1
Página 2
Página 3
13
14
15
16
17
18
19
20
Em uso
Em uso
Em uso
Lista de frames livres
13
14
15
18
20
Antes
Memória principal
Processo A
Página 0
Página 1
Página 2
Página 3
Página 1 de A
Página 2 de A
Página 3 de A
Página 0 de A
Lista de frames livres
20
Tabela de páginas do processo A
18
13
14
15
Depois

Paginação

- O SO mantém uma tabela de páginas para cada processo.
- A tabela de páginas mostra o local de cada página no processo.
- Dentro do programa, cada **endereço lógico** consiste em um **número de página** e um **endereço relativo dentro da página**.
- Lembre-se de que, no caso do particionamento simples, um endereço lógico é o local de uma palavra em relação ao início do programa; o processador traduz isso para um endereço físico.
- Com a paginação, a tradução de endereço de lógico para físico ainda é feita pelo hardware do processador.

Paginação

- Endereços lógicos e físicos:



Memória virtual

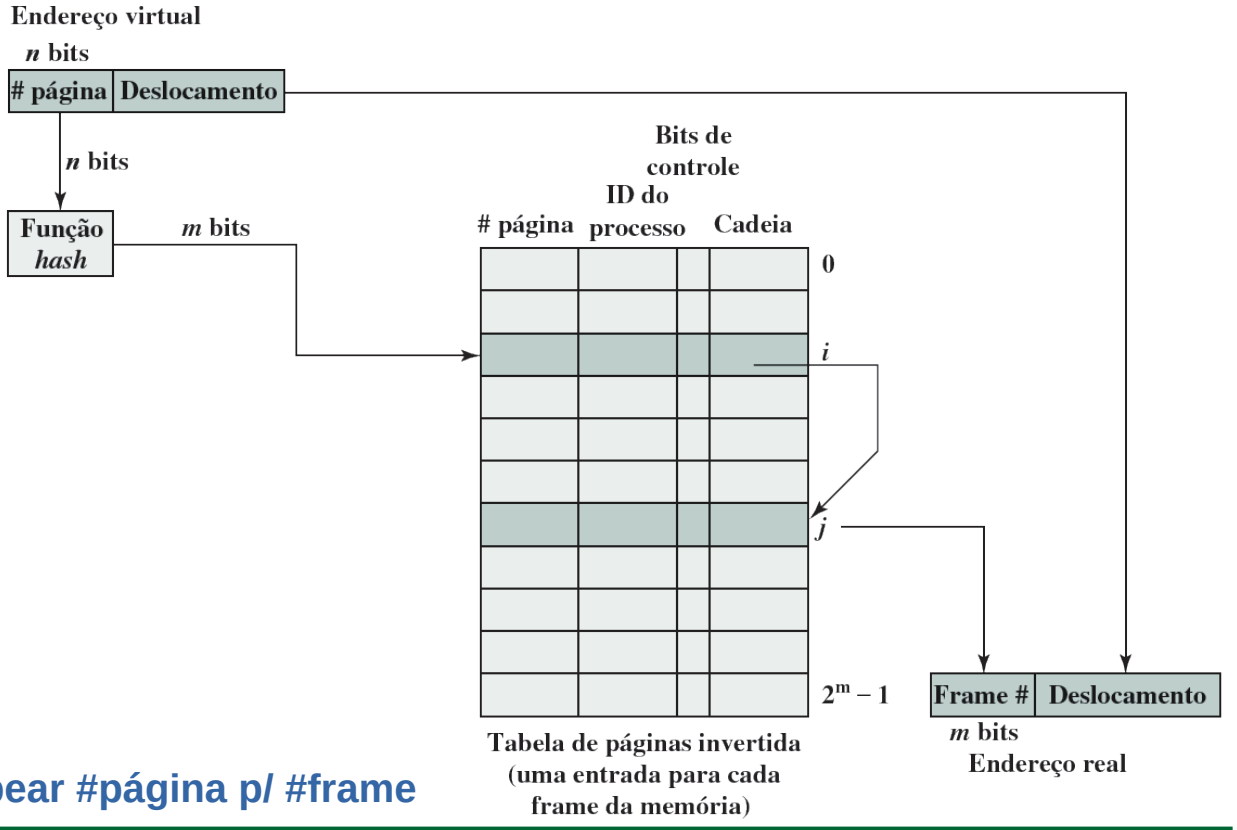
- Para entender a memória virtual, devemos acrescentar uma melhoria ao esquema de paginação que discutimos.
- Essa melhoria é a **paginação por demanda**, que simplesmente significa que cada página de um processo é trazida apenas quando necessária, ou seja, por demanda.
- Com a paginação por demanda, não é necessário carregar um processo inteiro na memória principal.
- Esse fato tem uma consequência marcante: é possível que um **processo seja maior do que toda a memória principal**.

Memória virtual

- Como um processo só é executado na **memória principal**, essa memória é conhecida como **memória real ou física**.
- Mas um programador ou usuário percebe uma **memória** muito maior — aquela que está **alocada no disco**.
- Essa última, portanto, é denominada **memória virtual**.
- A memória virtual permite uma multiprogramação bastante eficaz e alivia o usuário das restrições desnecessariamente rígidas da memória principal.

Memória virtual

- Estrutura da tabela de páginas invertida:

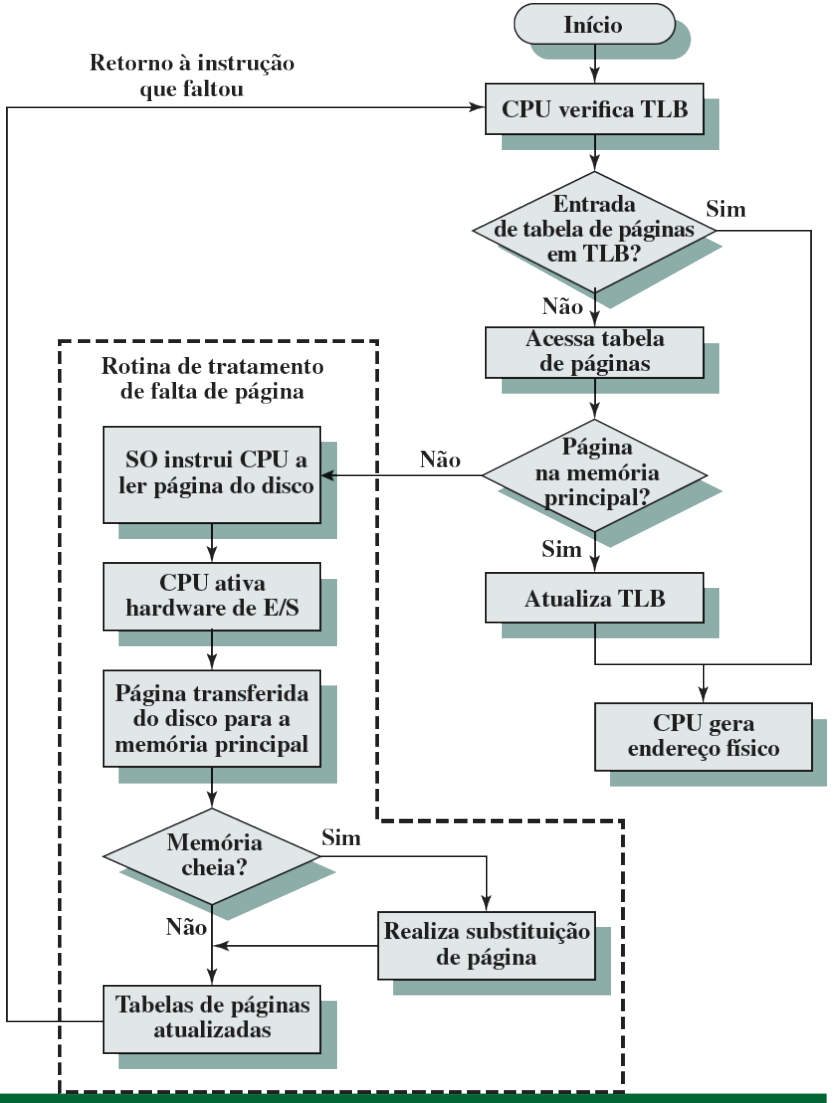


Técnica mais eficaz p/ mapear #página p/ #frame

Memória virtual

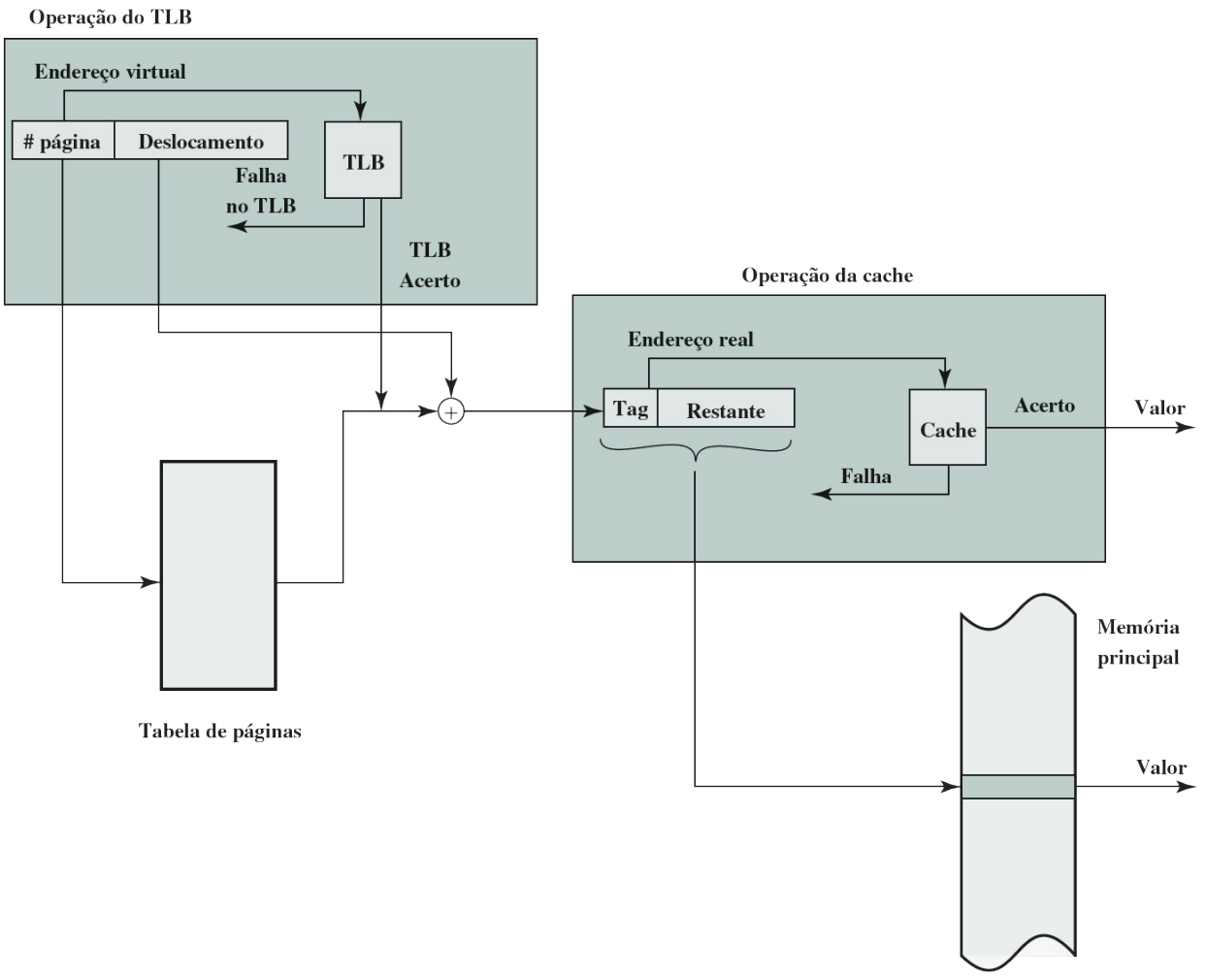
- A maioria dos esquemas de memória virtual utiliza uma **cache especial** para entradas da tabela de páginas, normalmente chamada de **Translation Lookaside Buffer** (TLB).
- Essa cache funciona da mesma maneira que uma cache de memória e contém as entradas da tabela de páginas que foram usadas recentemente.
- A figura a seguir é um fluxograma que mostra o uso do TLB.
- Os estudos do TLB do VAX mostraram que esse esquema pode melhorar o desempenho Significativamente.

Memória virtual



Memória virtual

- Translation lookaside buffer (TLB) e operação da cache:



Segmentação

- A segmentação normalmente é visível ao programador e é fornecida como uma conveniência para organizar programas e dados.
- Essa organização tem diversas vantagens para o programador em relação a um espaço de endereços não segmentado:
  i. Ela simplifica o tratamento de estruturas de dados que crescem.
  ii. Ela permite que os programas sejam alterados e recompilados de modo independente
  iii. Ela permite o compartilhamento entre os processos.
  iv. Ela serve para proteção.

Gerenciamento de memória do x86 da Intel

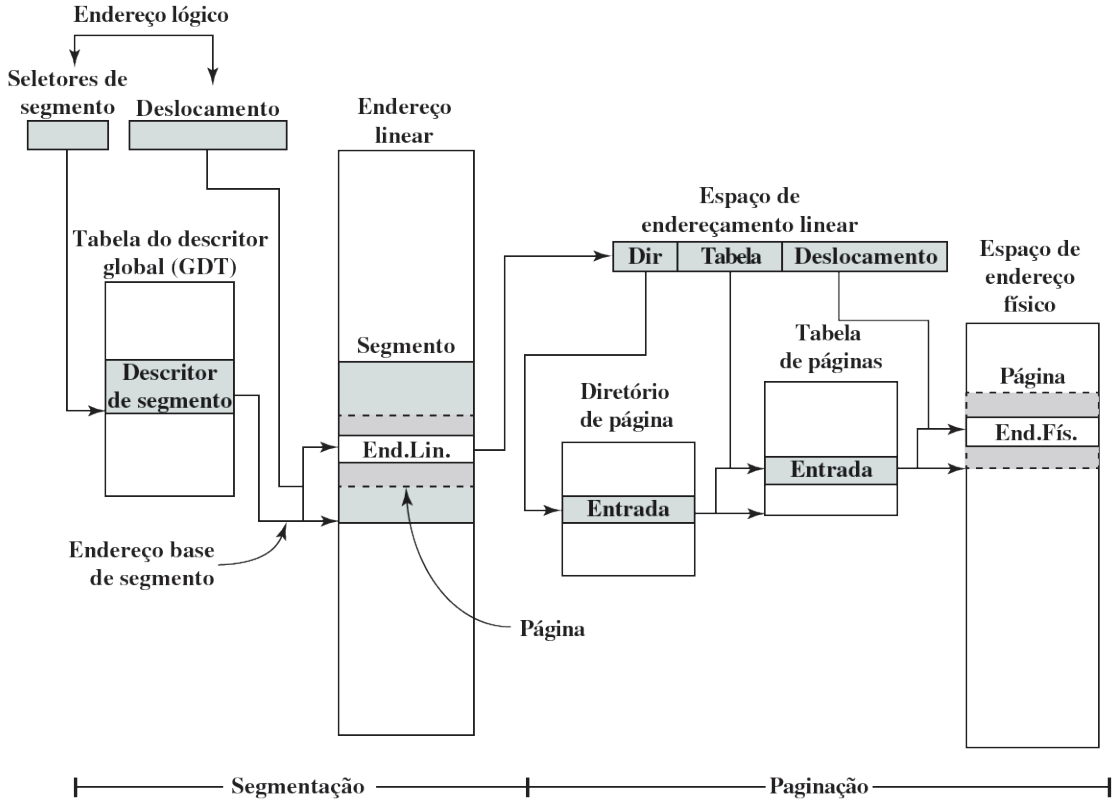
- O x86 inclui hardware para segmentação e paginação.
- Os dois mecanismos podem ser desativados, permitindo que o usuário escolha a partir de quatro visões distintas da memória:
  i. Memória não paginada não segmentada
  ii. Memória paginada não segmentada
  iii. Memória não paginada segmentada
  iv. Memória paginada segmentada

Gerenciamento de memória do x86 da Intel

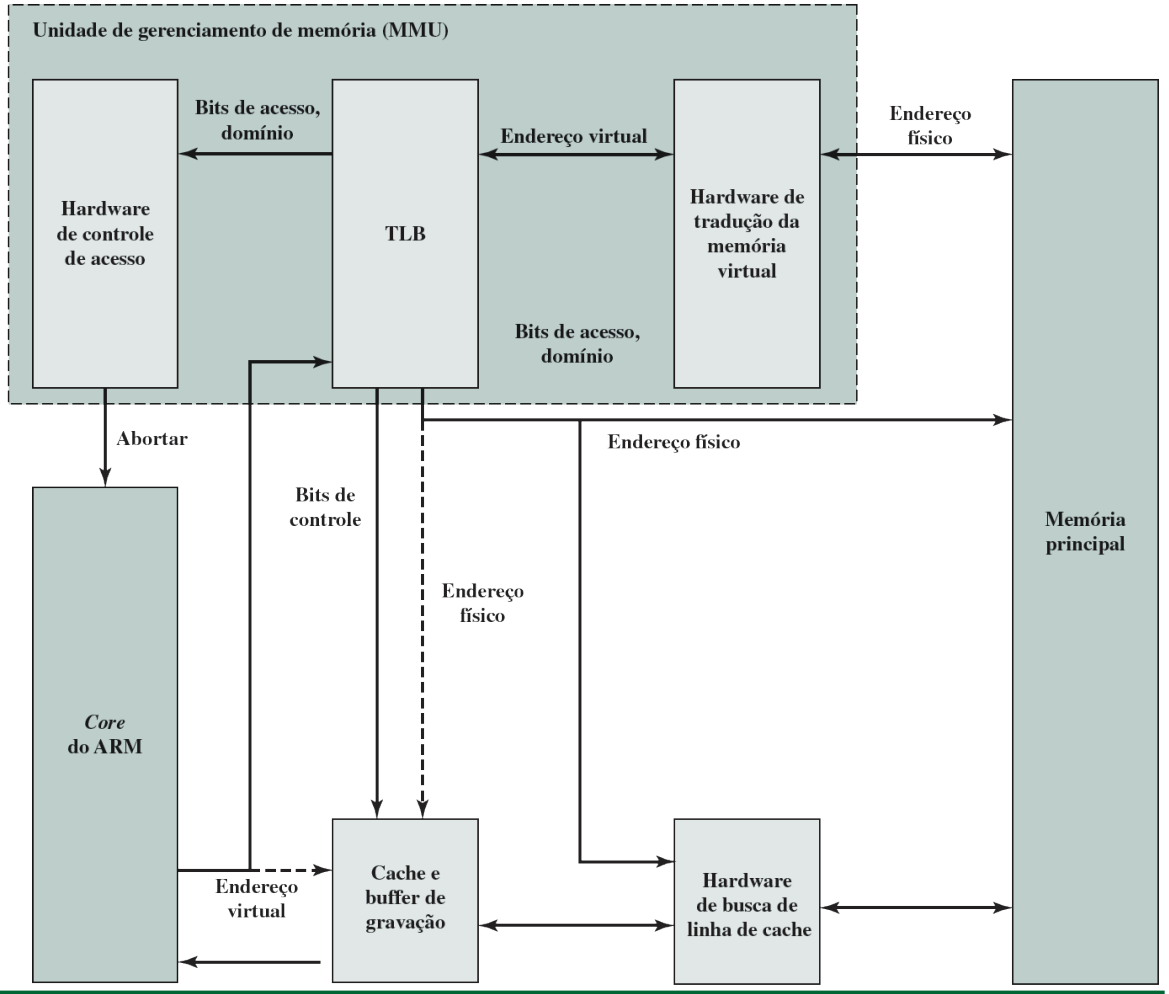
- Quando a segmentação é usada, cada endereço virtual consiste em uma referência de segmento de 16 bits e um deslocamento (offset) de 32 bits.
- A quantidade de memória virtual pode ser maior que 64 TB.
- O mecanismo de tradução de endereço para a segmentação envolve o mapeamento de um endereço virtual que é conhecido como **endereço linear**.
- Um endereço virtual consiste no deslocamento de 32 bits e um **seletor de segmento** de 16 bits.

Gerenciamento de memória do x86 da Intel

- Mecanismos de tradução de endereço de memória no x86 da Intel:



Gerenciamento de memória no ARM

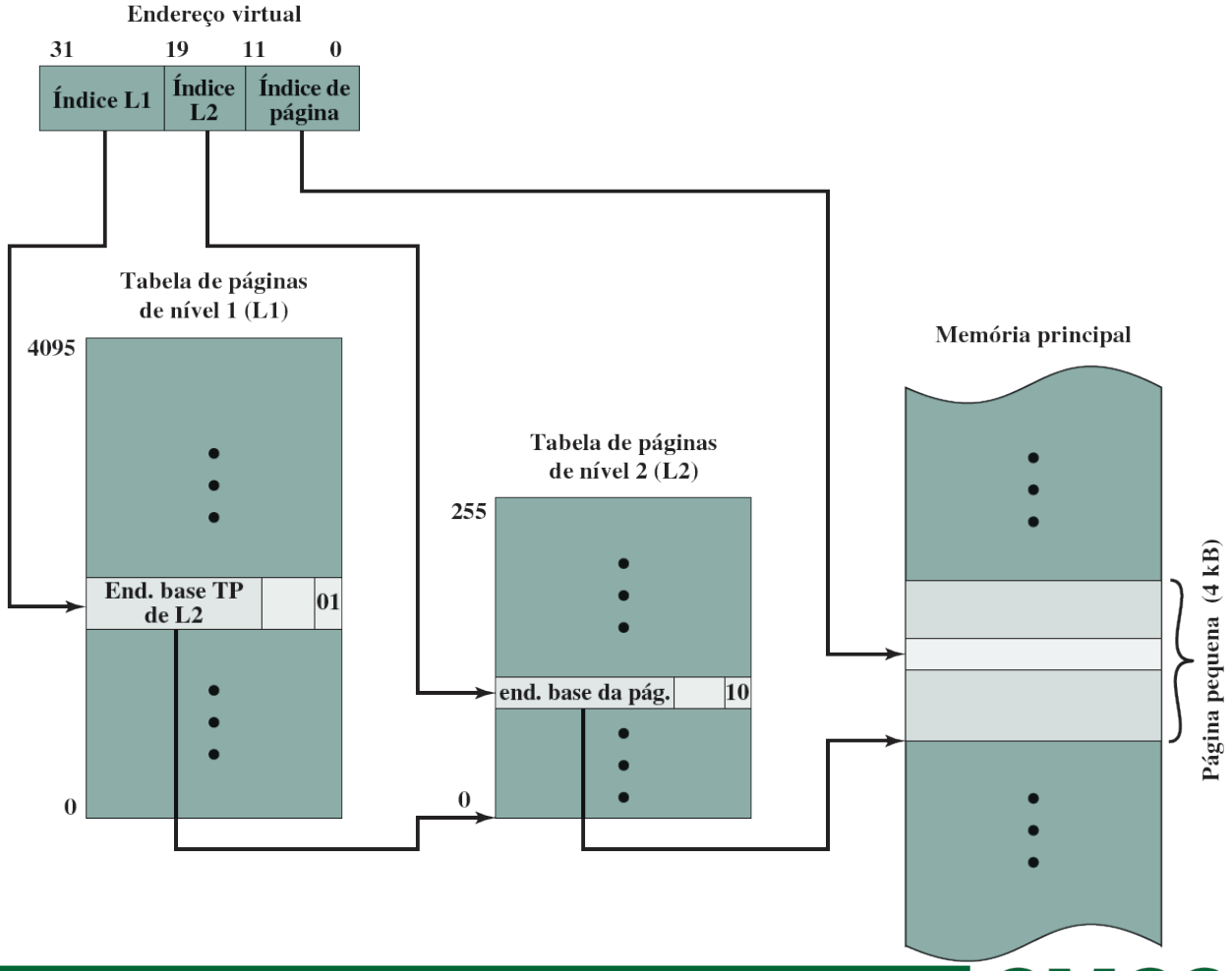


Gerenciamento de memória no ARM

- O ARM admite acesso à memória com base em seções ou páginas:
  - **Superseções** (opcional): consistem em blocos de 16 MB de memória principal.
  - **Seções**: consistem em blocos de 1 MB de memória principal.
  - **Páginas grandes**: consistem em blocos de 64 kB de memória principal.
  - **Páginas pequenas**: consistem em blocos de 4 kB de memória principal.

Gerenciamento de memória no ARM

- Tradução de endereço de memória virtual do ARM para páginas pequenas:

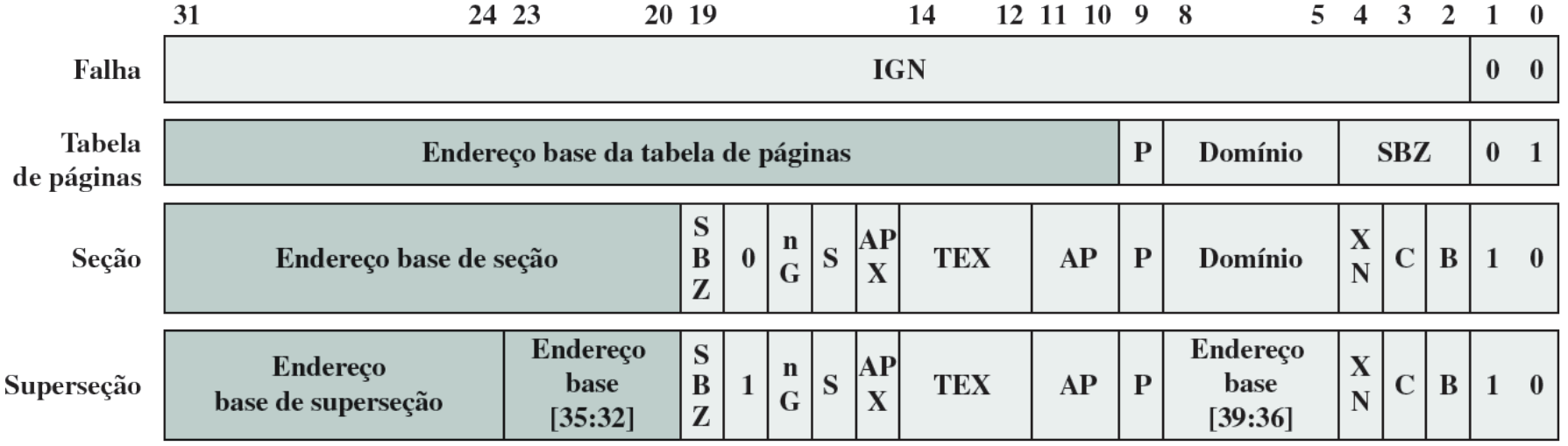


Gerenciamento de memória no ARM

- Para entender melhor o esquema de gerenciamento de memória do ARM, consideramos os principais formatos, como mostra a figura a seguir.
- Os bits de controle mostrados nessa figura são definidos na tabela seguinte.
- Cada entrada tem um dos quatro formatos alternativos:
  i. Bits [1:0] = 00
  ii. Bits [1:0] = 01
  iii. Bits [1:0] = 01: e bit 19 = 0
  iv. Bits [1:0] = 01: e bit 19 = 1

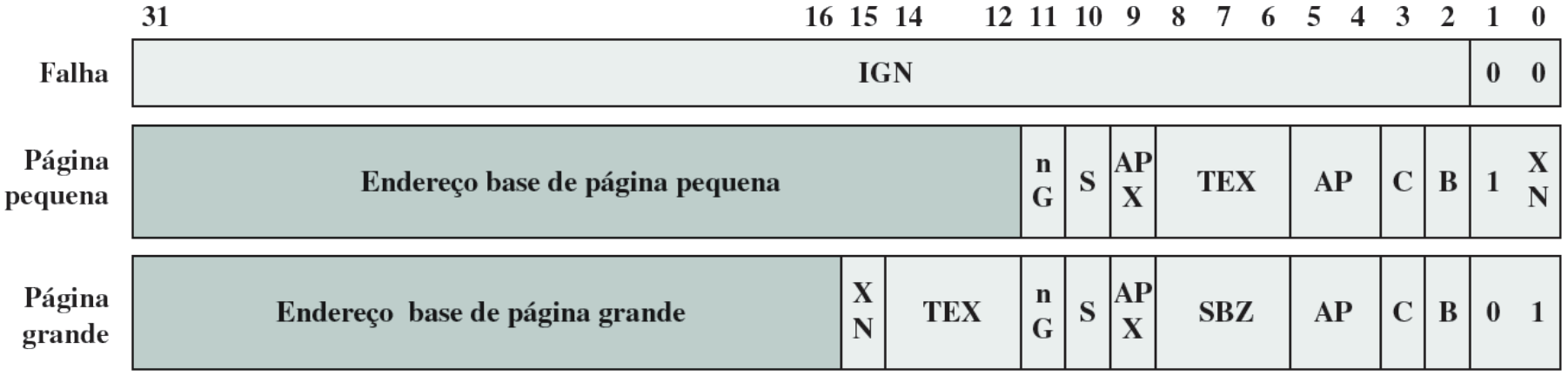
Gerenciamento de memória no ARM

- Formatos alternativos de descritor de primeiro nível:



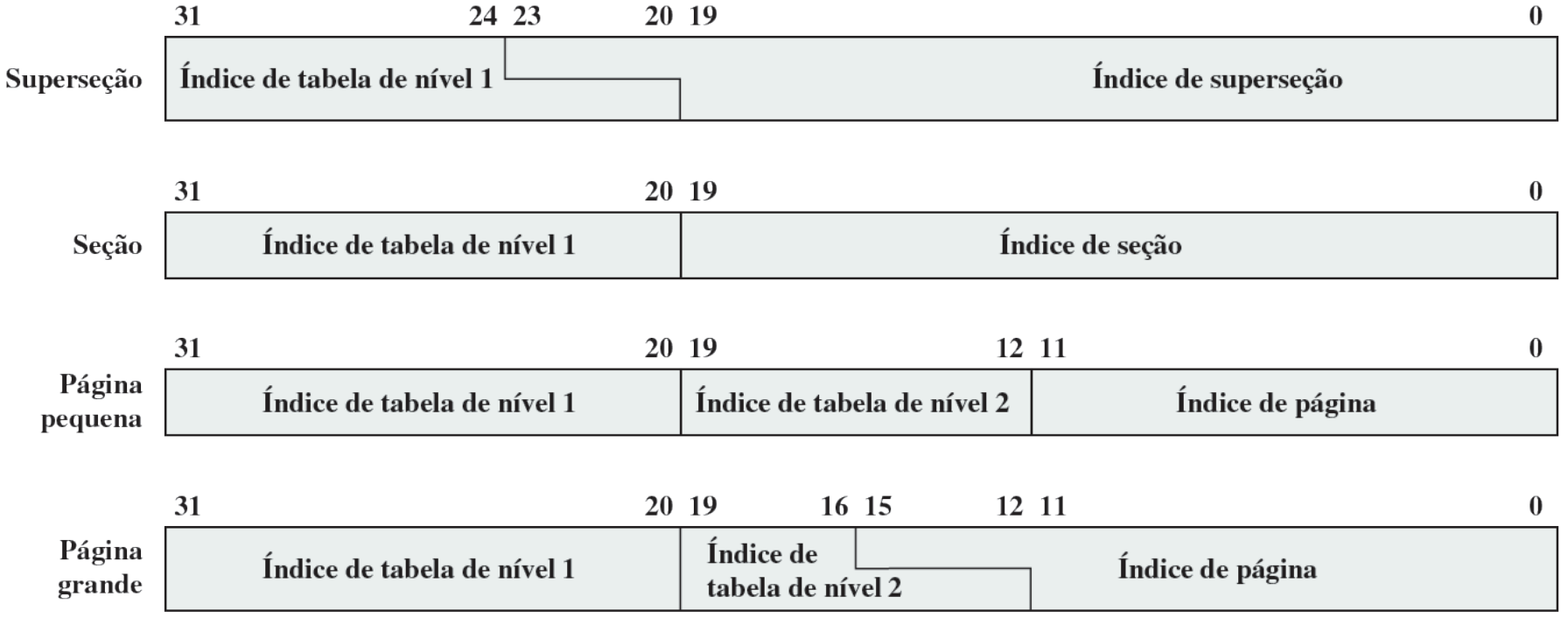
Gerenciamento de memória no ARM

- Formatos alternativos de descritor de segundo nível:



Gerenciamento de memória no ARM

- Formatos de endereço de memória virtual:



Gerenciamento de memória no ARM

- Parâmetros de gerenciamento de memória do ARM:

Permissão de acesso (AP — *Access Permission*), extensão de permissão de acesso (APX — *Access Permission eXtension*) Esses bits controlam o acesso à região de memória correspondente. Se um acesso for feito a uma área da memória sem as permissões exigidas, uma Falta de Permissão é levantada.
Bit bufferizável (B) Determina, com os bits TEX, como o buffer de gravação é usado para a memória cacheável.
Bit cacheável (C) Determina se essa região da memória pode ser mapeada pela cache.
Domínio Coleção de regiões da memória. O controle de acesso pode ser aplicado com base no domínio.
não Global (nG) Determina se a tradução deve ser marcada como global (0) ou específica ao processo (1).
Compartilhado (S — *Shared*) Determina se a tradução é para a memória não compartilhada (0) ou compartilhada (1).
SBZ Deverá ser zero (should be zero).
Extensão de tipo (TEX — *Type EXtension*) Esses bits, juntamente com os bits B e C, controlam os acessos às caches, como o buffer de gravação é usado e se a região da memória é compartilhável e, portanto, deve ser mantida coerente.
Executar Nunca (XN — eXecute Never) Determina se a região é executável (0) ou não executável (1).

Gerenciamento de memória no ARM

- O ARM emprega o conceito de um domínio, que é uma coleção de seções e/ou páginas que possuem permissões de acesso particulares. A arquitetura ARM admite 16 domínios.
- Dois tipos de acesso de domínio são aceitos:
  i. **Clientes**: usuários de domínios que devem observar as permissões de acesso das seções individuais e/ou páginas que compõem esse domínio.
  ii. **Gerentes**: controlam o comportamento do domínio e contornam as permissões de acesso para entradas de tabela nesse domínio.