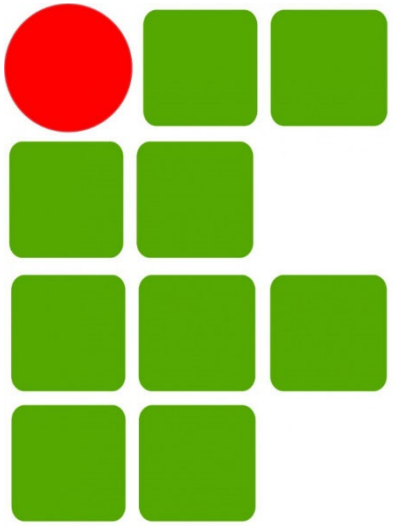


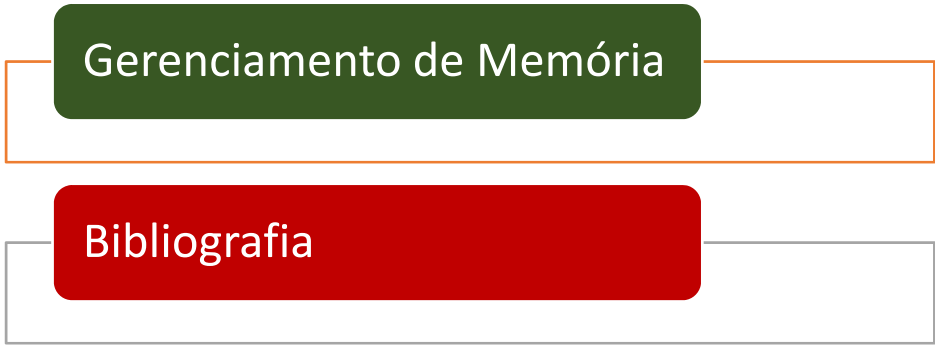
INSTITUTO FEDERAL
CATARINENSE
Câmpus Araquari

Sistemas Operacionais

Prof. Joelmir José Lopes

Sistemas Operacionais

- Esta aula tem como objetivo abordar o assunto Gerenciamento de Memória: localidade de referência, fragmentação e memória virtual.



Gerenciamento de Memória

Gerenciamento de Memória

Alocação segmentada paginada

Cada uma das principais formas de alocação de memória vistas até agora tem suas vantagens: **a alocação contígua** prima pela simplicidade e rapidez; **a alocação por segmentos** oferece múltiplos espaços de endereçamento para cada processo, oferecendo flexibilidade ao programador; **a alocação por páginas** oferece um grande espaço de endereçamento linear, enquanto elimina a fragmentação externa.

Alguns processadores oferecem mais de uma forma de alocação, deixando aos projetistas do sistema operacional a escolha da forma mais adequada de organizar a memória usada por seus processos.

Gerenciamento de Memória

Localidade de referências

A forma como os processos acessam a memória tem um impacto direto na eficiência dos mecanismos de gerência de memória, sobretudo o cache de páginas (TLB) e o mecanismo de memória virtual (a ser visto futuramente).

Processos que concentram seus acessos em poucas páginas de cada vez farão um uso eficiente desses mecanismos, enquanto processos que acessam muitas páginas distintas em um curto período irão gerar frequentes erros de cache (TLB) e faltas de página, prejudicando seu desempenho no acesso à memória.

A propriedade de um processo ou sistema concentrar seus acessos em poucas áreas da memória a cada instante é chamada localidade de referências.

Gerenciamento de Memória

Formas de localidade de referências:

Localidade temporal: um recurso usado há pouco tempo será provavelmente usado novamente em um futuro próximo (esta propriedade é usada pelos algoritmos de gerência de memória virtual);

Localidade espacial: um recurso será mais provavelmente acessado se outro recurso próximo a ele já foi acessado (é a propriedade verificada na primeira execução)

Gerenciamento de Memória

Como exemplo prático da importância da localidade de referências, considere um programa para o preenchimento de uma matriz de 4:096 x 4:096 bytes, onde cada linha da matriz está alocada em uma página distinta (considerando páginas de 4.096 bytes).

O trecho de código a seguir implementa essa operação, percorrendo a matriz linha por linha:

```
1  unsigned char buffer[4096][4096] ;
2
3  int main ()
4  {
5     int i, j ;
6
7     for (i=0; i<4096; i++) // percorre as linhas do buffer
8        for (j=0; j<4096; j++) // percorre as colunas do buffer
9           buffer[i][j]= (i+j) % 256 ;
10 }
```

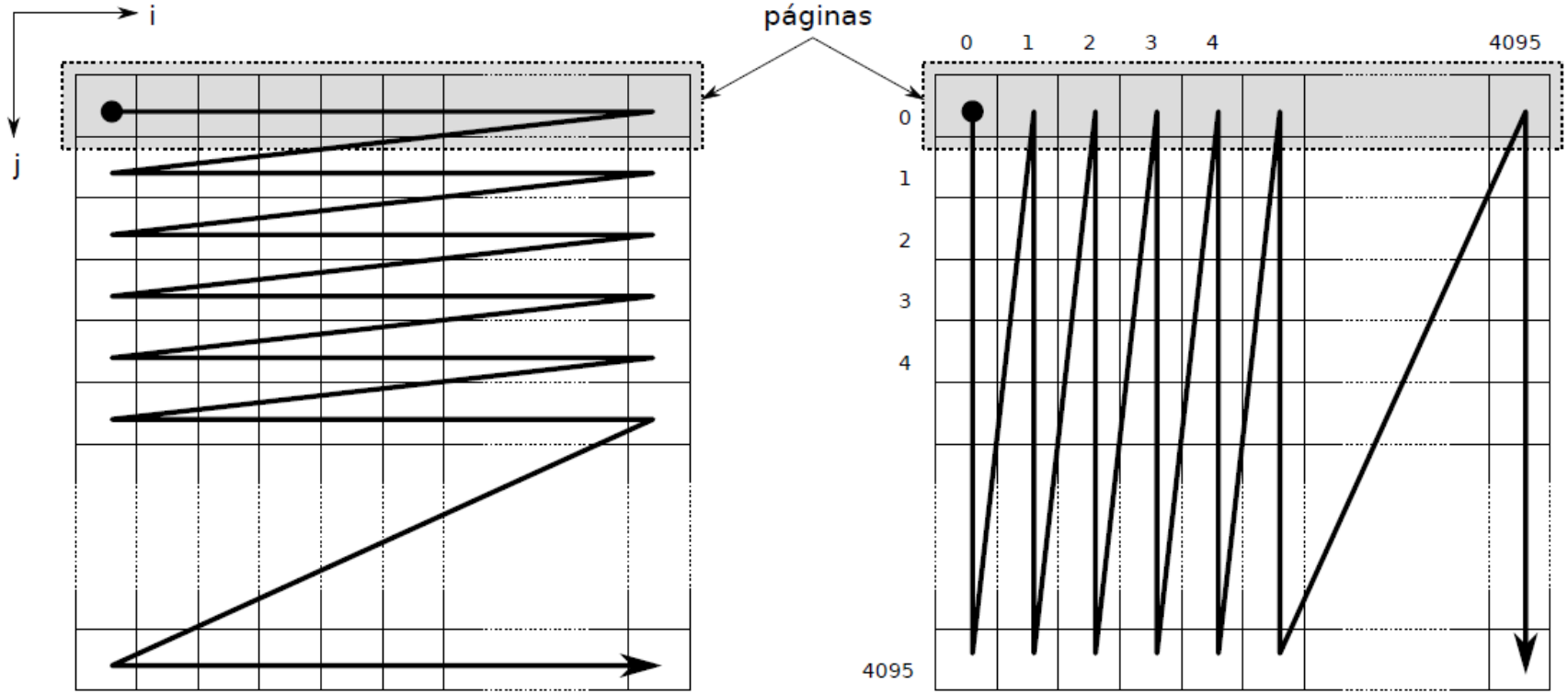
Gerenciamento de Memória

Outra implementação possível seria percorrer a matriz coluna por coluna, conforme o código a seguir:

```
1  unsigned char buffer[4096][4096] ;
2
3  int main ()
4  {
5     int i, j ;
6
7     for (j=0; j<4096; j++) // percorre as colunas do buffer
8        for (i=0; i<4096; i++) // percorre as linhas do buffer
9           buffer[i][j]= (i+j) % 256 ;
10 }
```

Gerenciamento de Memória

Embora percorram a matriz de forma distinta, os dois programas geram o mesmo resultado e são conceitualmente equivalentes (a Figura abaixo mostra o padrão de acesso à memória dos dois programas).



Comportamento dos programas no acesso à memória.

Gerenciamento de Memória

Entretanto, eles não têm o mesmo desempenho. A primeira implementação (percurso linha por linha) usa de forma eficiente o cache da tabela de páginas, porque só gera um erro de cache a cada nova linha acessada. Por outro lado, a implementação com percurso por colunas gera um erro de cache TLB a cada célula acessada, nesse exemplo, a cache guarda as linhas da imagem.

Gerenciamento de Memória

A localidade de referência de uma implementação depende de um conjunto de fatores, que incluem:

- As estruturas de dados usadas pelo programa: estruturas como vetores e matrizes têm seus elementos alocados de forma contígua na memória, o que leva a uma localidade de referências maior que estruturas mais dispersas, como listas encadeadas e árvores;

- Os algoritmos usados pelo programa: o comportamento do programa no acesso à memória é definido pelos algoritmos que ele implementa;

- A qualidade do compilador: cabe ao compilador analisar quais variáveis e trechos de código são usadas com frequência juntos e colocá-los nas mesmas páginas de memória, para aumentar a localidade de referências do código gerado.

Fragmentação

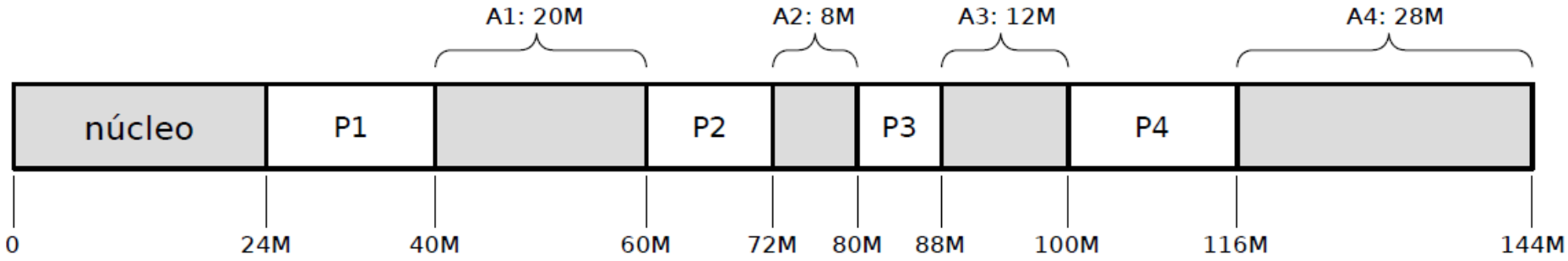
Ao longo da vida de um sistema, áreas de memória são liberadas por processos que concluem sua execução e outras áreas são alocadas por novos processos, de forma contínua.

Com isso, podem surgir áreas livres (vazios ou buracos na memória) entre os processos, o que constitui um problema conhecido como fragmentação externa.

Esse problema somente afeta as estratégias de alocação que trabalham com blocos de tamanho variável, como a alocação contígua e a alocação segmentada. Por outro lado, a alocação paginada sempre trabalha com blocos de mesmo tamanho (os quadros e páginas), sendo por isso imune à fragmentação externa. POR QUÊ?

Gerenciamento de Memória

A fragmentação externa é prejudicial porque limita a capacidade de alocação de memória no sistema. A Figura abaixo apresenta um sistema com alocação contígua de memória no qual ocorre fragmentação externa.



Memória com fragmentação externa.

Na Figura acima, observa-se que existem 68 MBytes de memória livre em quatro áreas separadas (A_1 ... A_4), mas somente processos com até 28 MBytes podem ser alocados (usando a maior área livre, A_4).

Gerenciamento de Memória

Pode-se enfrentar o problema da fragmentação externa de duas formas: minimizando sua ocorrência, através de critérios de escolha das áreas a alocar, ou desfragmentando periodicamente a memória do sistema.

Para minimizar a ocorrência de fragmentação externa, cada pedido de alocação deve ser analisado para encontrar a área de memória livre que melhor o atenda. Essa análise pode ser feita usando um dos seguintes critérios:

Melhor encaixe (*best-fit*): consiste em escolher a menor área possível que possa atender à solicitação de alocação. Dessa forma, as áreas livres são usadas de forma otimizada, mas eventuais resíduos (sobras) podem ser pequenos demais para ter alguma utilidade.

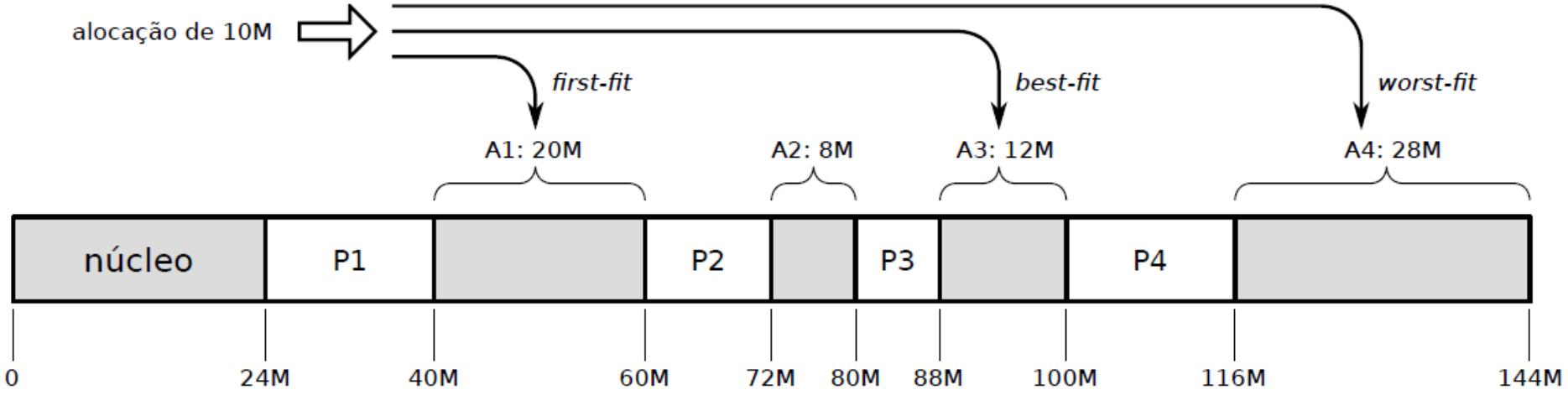
Gerenciamento de Memória

Pior encaixe (worst-fit): consiste em escolher sempre a maior área livre possível, de forma que os resíduos sejam grandes e possam ser usados em outras alocações.

Primeiro encaixe (*first-fit*): consiste em escolher a primeira área livre que satisfaça o pedido de alocação; tem como vantagem a rapidez, sobretudo se a lista de áreas livres for muito longa, etc.

Gerenciamento de Memória

Diversas pesquisas demonstraram que as abordagens mais eficientes são a de ***melhor encaixe*** e a de ***primeiro encaixe***, sendo esta última bem mais rápida. A Figura abaixo ilustra essas estratégias.



Estratégias para minimizar a fragmentação externa.

Gerenciamento de Memória

Outra forma de tratar a fragmentação externa consiste em desfragmentar a memória periodicamente.

Para tal, as áreas de memória usadas pelos processos devem ser movidas na memória de forma a concatenar as áreas livres e assim diminuir a fragmentação. Ao mover um processo na memória, suas informações de alocação (registrador base ou tabela de segmentos) devem ser ajustadas para refletir a nova posição do processo.

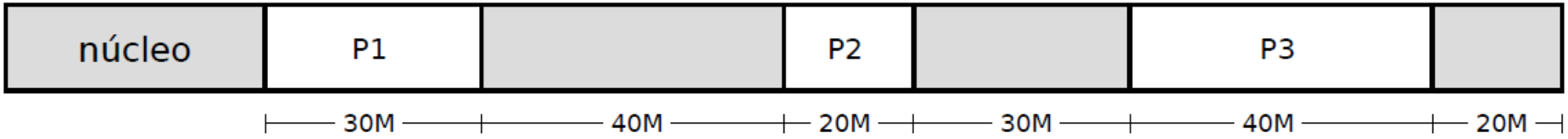
Gerenciamento de Memória

Obviamente, nenhum processo pode executar durante a desfragmentação.

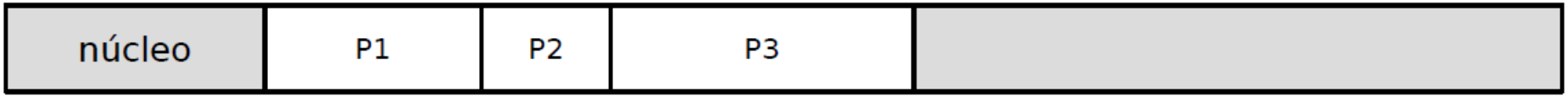
Portanto, é importante que esse procedimento seja executado rapidamente e com pouca frequência, para não interferir nas atividades normais do sistema.

A Figura mostrada no próximo slide ilustra três possibilidades de desfragmentação de uma determinada situação de memória; as três alternativas produzem o mesmo resultado, mas apresentam custos distintos.

Situação inicial



Solução 1: deslocar P2 e P3 (custo: mover 60M)



Solução 2: deslocar P3 (custo: mover 40M)

Solução 3: deslocar P3 (custo: mover 20M)

Possibilidades de desfragmentação.

Gerenciamento de Memória

Além da fragmentação externa, que afeta as áreas livres entre os processos, as estratégias de alocação de memória também podem apresentar a fragmentação interna, que pode ocorrer dentro das áreas alocadas aos processos.

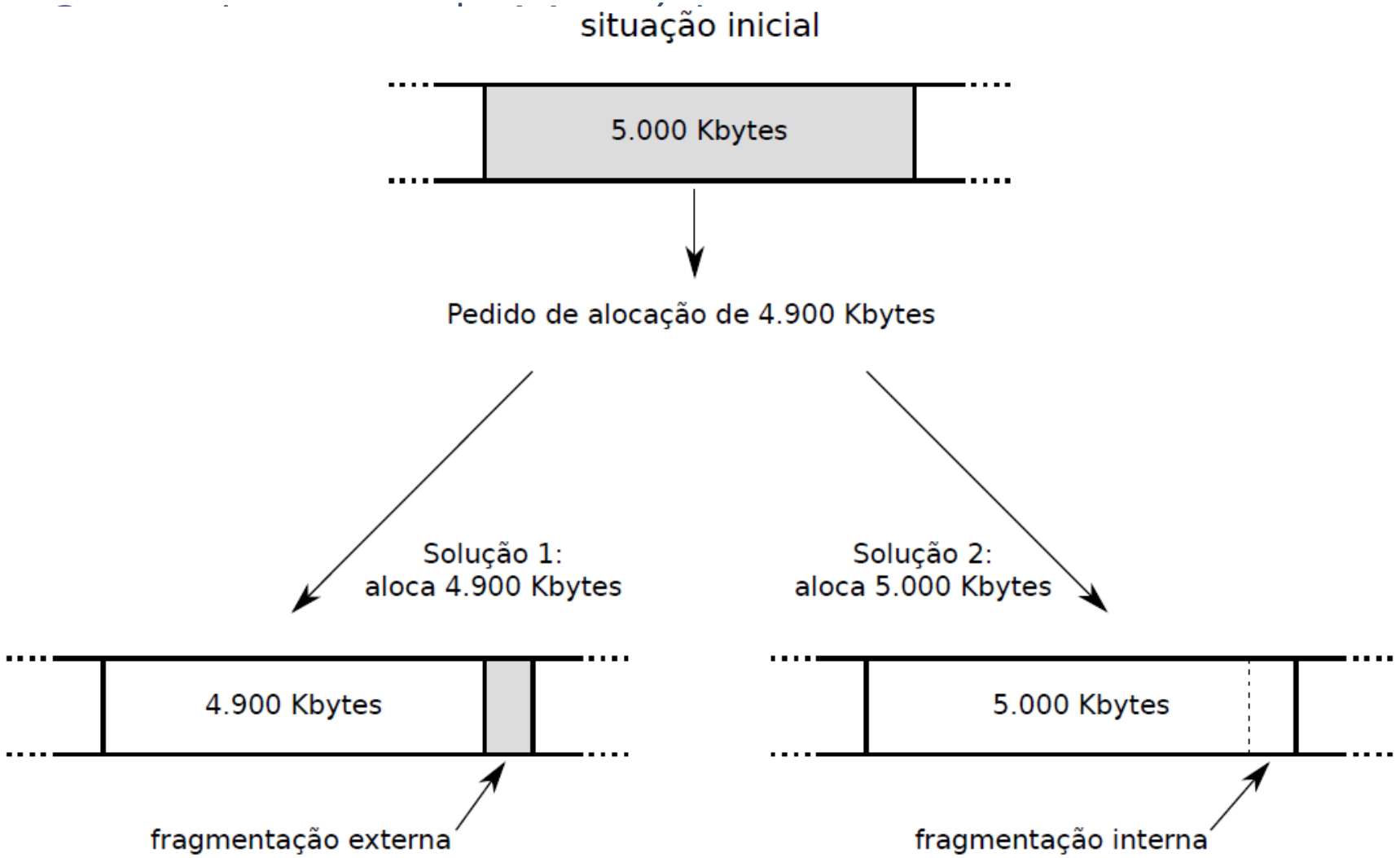
A Figura mostrada no próximo slide apresenta uma situação onde ocorre esse problema: um novo processo requisita uma área de memória com 4.900 Kbytes. Todavia, a área livre disponível tem 5.000 Kbytes.

Se for alocada exatamente a área solicitada pelo processo (situação A), sobrará um fragmento residual com 100 Kbytes, que é praticamente inútil para o sistema, pois é muito pequeno para acomodar novos processos (fragmentação externa).

Gerenciamento de Memória

Além disso, essa área residual de 100 Kbytes deve ser incluída na lista de áreas livres, o que representa um custo de gerência desnecessário.

Outra possibilidade consiste em “arredondar” o tamanho da área solicitada pelo processo para 5.000 Kbytes, ocupando totalmente aquela área livre (situação B). Assim, haverá uma pequena área de 100 Kbytes no final da memória do processo, que provavelmente não será usada por ele (fragmentação interna).


situação inicial
5.000 Kbytes
Pedido de alocação de 4.900 Kbytes
Solução 1:
aloca 4.900 Kbytes
Solução 2:
aloca 5.000 Kbytes
4.900 Kbytes
5.000 Kbytes
fragmentação externa
fragmentação interna

Gerenciamento de Memória

A fragmentação interna afeta todas as formas de alocação; as alocações contígua e segmentada sofrem menos com esse problema, pois o nível de arredondamento das alocações pode ser decidido caso a caso. No caso da alocação paginada, essa decisão não é possível, pois as alocações são feitas em páginas inteiras. Assim, em um sistema com páginas de 4 Kbytes (4.096 bytes), um processo que solicite a alocação de 550.000 bytes (134,284 páginas) receberá 552.960 bytes (135 páginas), ou seja, 2.960 bytes a mais que o solicitado.

Gerenciamento de Memória

Uma forma de minimizar a perda por fragmentação interna seria usar páginas de menor tamanho (2K, 1K, 512 bytes ou ainda menos).
Todavia, essa abordagem implica em ter mais páginas por processo, o que geraria tabelas de páginas maiores e com maior custo de gerência.

Gerenciamento de Memória

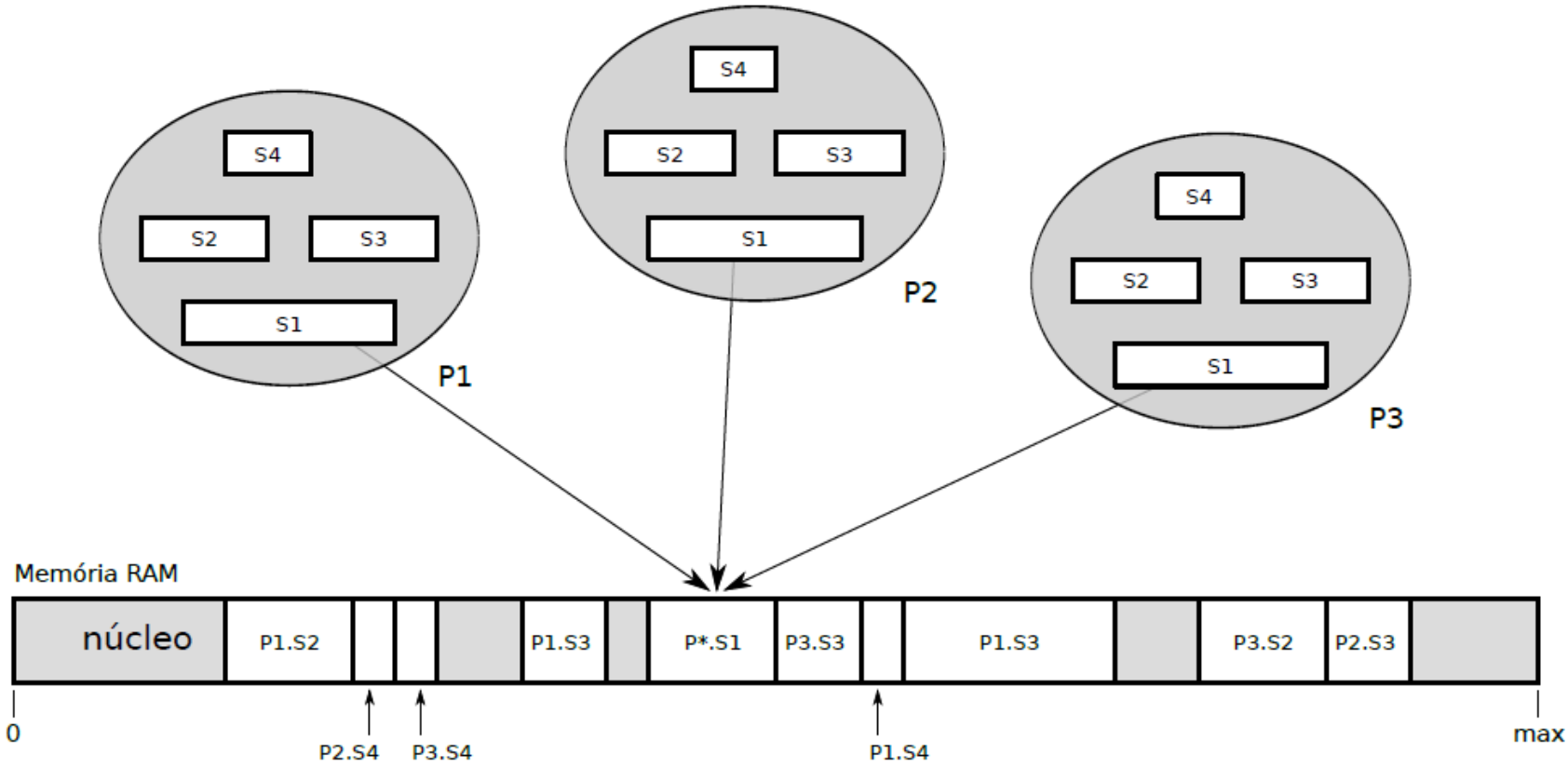
Compartilhamento de memória

A memória RAM é um recurso escasso, que deve ser usado de forma eficiente. Nos sistemas atuais, é comum ter várias instâncias do mesmo programa em execução, como várias instâncias de editores de texto, de navegadores, etc.

Gerenciamento de Memória

O compartilhamento de código entre processos pode ser implementado de forma muito simples e transparente para os processos envolvidos, através dos mecanismos de tradução de endereços oferecidos pela MMU, como segmentação e paginação.

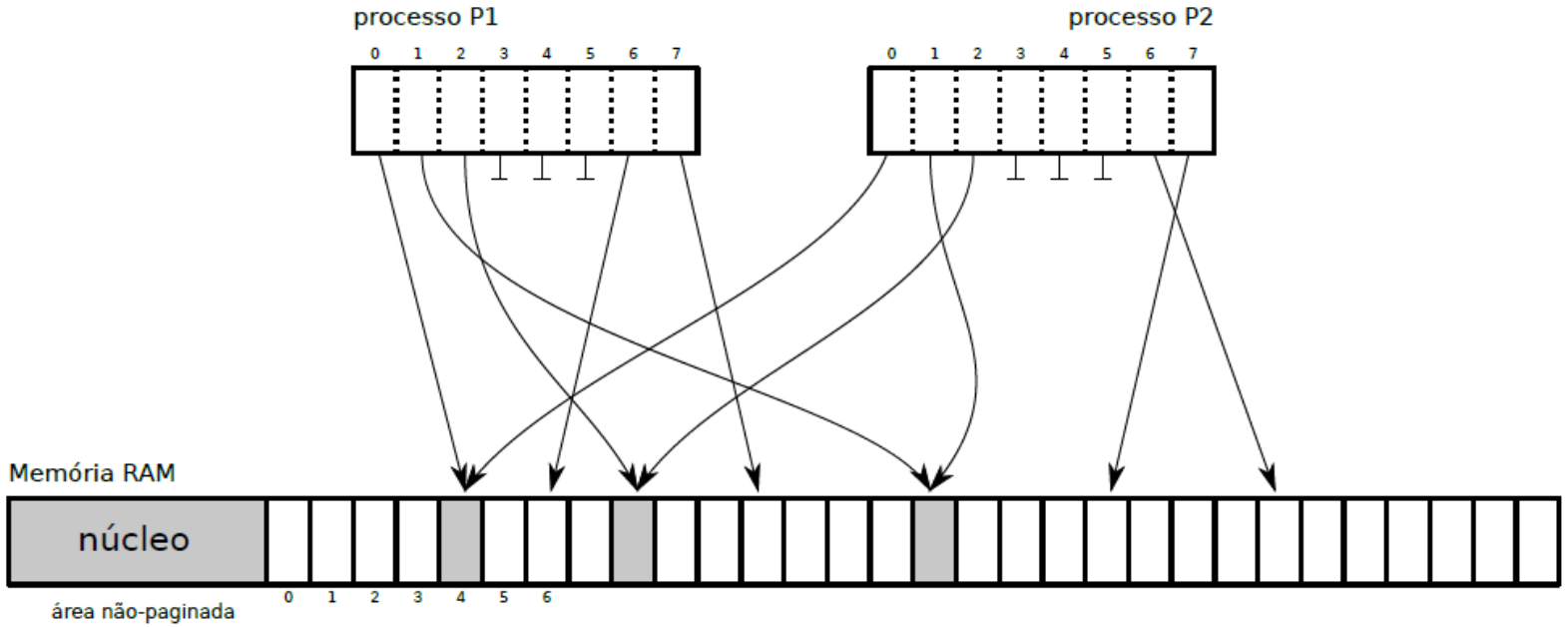
No caso da segmentação, bastaria fazer com que todos os segmentos de código dos processos apontem para o mesmo segmento da memória física, como indica a Figura mostrada no próximo slide. É importante observar que o compartilhamento é transparente para os processos: cada processo continua a acessar endereços lógicos em seu próprio segmento de código, buscando suas instruções a executar.



Compartilhamento de segmentos.

Gerenciamento de Memória

No caso da paginação, a unidade básica de compartilhamento é a página. Assim, as entradas das tabelas de páginas dos processos envolvidos são ajustadas para referenciar os mesmos quadros de memória física. É importante observar que, embora referenciem os mesmos endereços físicos, as páginas compartilhadas podem ter endereços lógicos distintos. A Figura abaixo ilustra o compartilhamento de páginas entre processos.



Compartilhamento de páginas.

Gerenciamento de Memória

O compartilhamento das áreas de código permite proporcionar uma grande economia no uso da memória física, sobretudo em servidores e sistemas multiusuários.

Memória virtual

Um problema constante nos computadores é a disponibilidade de memória física: os programas se tornam cada vez maiores e cada vez mais processos executam simultaneamente, ocupando a memória disponível.

Além disso, a crescente manipulação de informações multimídia (imagens, áudio, vídeo) contribui para esse problema, uma vez que essas informações são geralmente volumosas e seu tratamento exige grandes quantidades de memória livre.

Gerenciamento de Memória – Memória Virtual

Como a memória RAM é um recurso caro e que consome uma quantidade significativa de energia, aumentar sua capacidade nem sempre é uma opção factível.

Observando o comportamento de um sistema computacional, constata-se que nem todos os processos estão constantemente ativos, e que nem todas as áreas de memória estão constantemente sendo usadas.

Gerenciamento de Memória – Memória Virtual

Por isso, as áreas de memória pouco acessadas poderiam ser transferidas para um meio de armazenamento mais barato e abundante, como um disco rígido ou um banco de memória flash, liberando a memória RAM para outros usos. Quando um processo proprietário de uma dessas áreas precisar acessá-la, ela deve ser transferida de volta para a memória RAM.

O uso de um armazenamento externo como extensão da memória RAM se chama memória virtual; essa estratégia pode ser implementada de forma eficiente e transparente para processos, usuários e programadores.

Gerenciamento de Memória – Memória Virtual

Mecanismo básico

Nos primeiros sistemas a implementar estratégias de memória virtual, processos inteiros eram transferidos da memória para o disco rígido e vice-versa. Esse procedimento, denominado troca (swapping) permite liberar grandes áreas de memória a cada transferência, e se justifica no caso de um armazenamento com tempo de acesso muito elevado, como os antigos discos rígidos.

Os sistemas atuais raramente transferem processos inteiros para o disco; geralmente as transferências são feitas por páginas ou grupos de páginas, em um procedimento denominado paginação (paging), detalhado a seguir.

Gerenciamento de Memória – Memória Virtual

Normalmente, o mecanismo de memória virtual se baseia em páginas ao invés de segmentos. As páginas têm um tamanho único e fixo, o que permite simplificar os algoritmos de escolha de páginas a remover, os mecanismos de transferência para o disco e também a formatação da área de troca no disco.

A otimização desses fatores seria bem mais complexa e menos efetiva caso as operações de troca fossem baseadas em segmentos, que têm tamanho variável.

Gerenciamento de Memória – Memória Virtual

A ideia central do mecanismo de memória virtual em sistemas com memória paginada consiste em retirar da memória principal as páginas menos usadas, salvando-as em uma área do disco rígido reservada para esse fim.

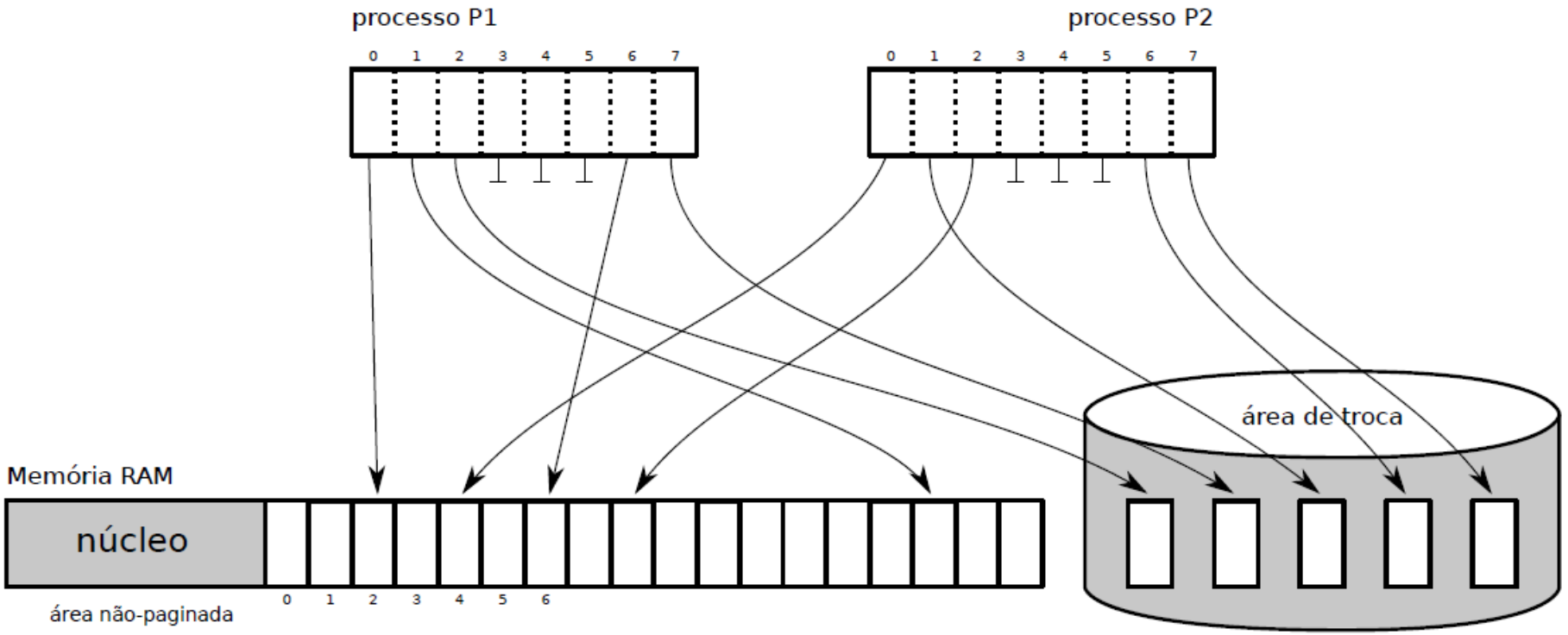
Essa operação é feita periodicamente, de modo reativo (quando a quantidade de memória física disponível cai abaixo de um certo limite) ou pró-ativo (aproveitando os períodos de baixo uso do sistema para retirar da memória as páginas pouco usadas).

Gerenciamento de Memória – Memória Virtual

As páginas a retirar são escolhidas de acordo com algoritmos de substituição de páginas, discutidos em breve.

As entradas das tabelas de páginas relativas às páginas transferidas para o disco devem então ser ajustadas de forma a referenciar os conteúdos correspondentes no disco rígido. Essa situação está ilustrada de forma simplificada na Figura mostrada no próximo slide.

Gerenciamento de Memória – Memória Virtual



Memória virtual paginada

Gerenciamento de Memória – Memória Virtual

O armazenamento externo das páginas pode ser feito em um disco exclusivo para esse fim (usual em servidores de maior porte), em uma partição do disco principal (usual no Linux e outros UNIX - SWAP) ou em um arquivo reservado dentro do sistema de arquivos do disco principal da máquina, geralmente oculto (como no Windows NT e sucessores - pagefile.sys em c:).

Gerenciamento de Memória – Memória Virtual

Por razões históricas, essa área de disco é geralmente denominada área de troca (swap area), embora armazene páginas.

No caso de um disco exclusivo ou partição de disco, essa área geralmente é formatada usando uma estrutura de sistema de arquivos otimizada para o armazenamento e recuperação rápida das páginas.

Gerenciamento de Memória – Memória Virtual

As páginas que foram transferidas da memória para o disco provavelmente serão necessárias no futuro, pois seus processos proprietários provavelmente continuam vivos.

Quando um processo tentar acessar uma página ausente, esta deve ser transferida de volta para a memória para permitir seu acesso, de forma transparente ao processo.

Gerenciamento de Memória – Memória Virtual

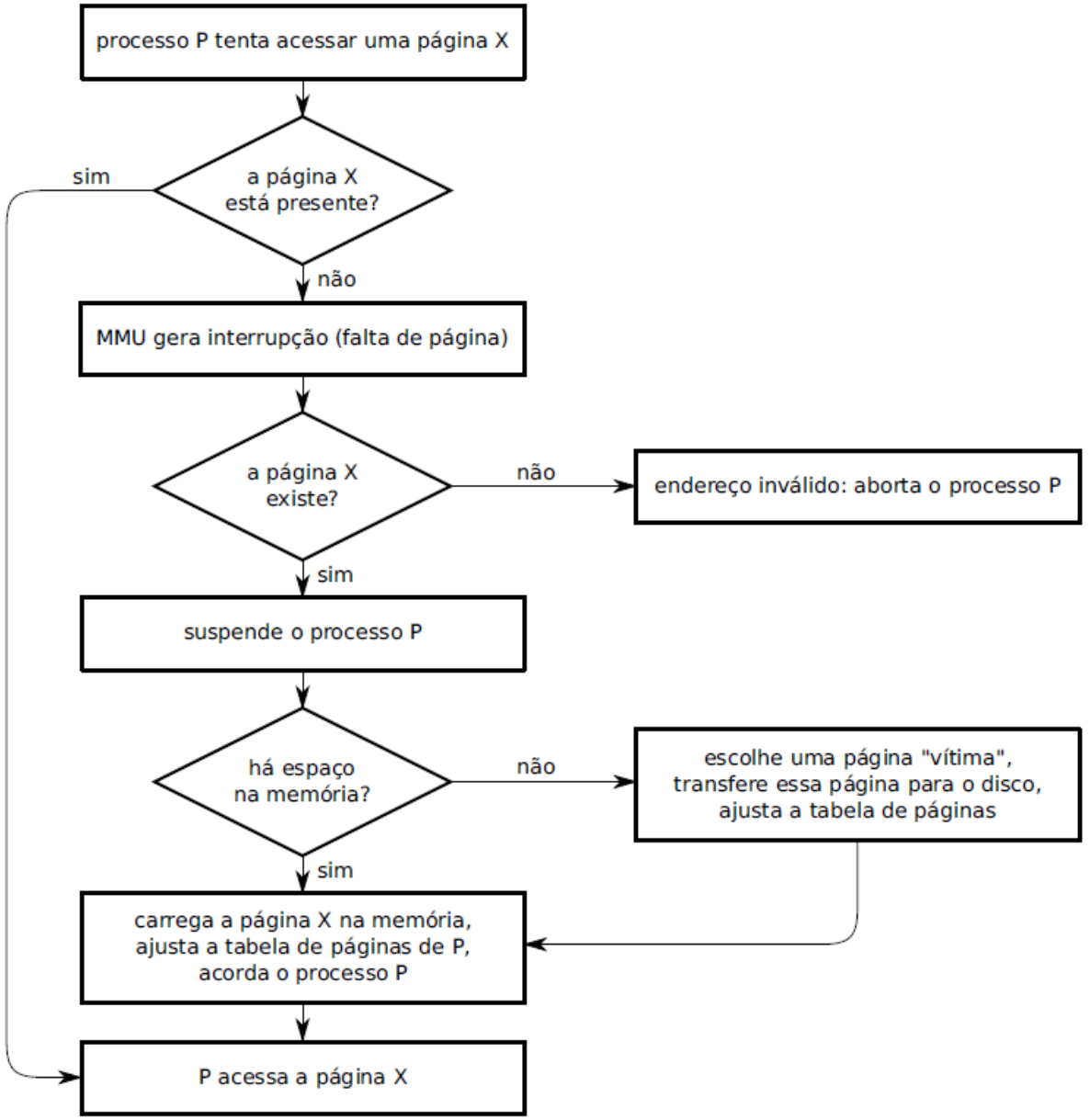
Conforme exposto na **Alocação paginada**, quando um processo acessa uma página, a MMU verifica se a mesma está mapeada na memória RAM e, em caso positivo, faz o acesso ao endereço físico correspondente.

Caso contrário, a MMU gera uma interrupção de falta de página (page fault) que força o desvio da execução para o sistema operacional. Nesse instante, o sistema deve verificar se a página solicitada não existe ou se foi transferida para o disco, usando os flags de controle da respectiva entrada da tabela de páginas.

Gerenciamento de Memória – Memória Virtual

Caso a página não exista, o processo tentou acessar um endereço inválido e deve ser abortado. Por outro lado, caso a página solicitada tenha sido transferida para o disco, o processo deve ser suspenso enquanto o sistema transfere a página de volta para a memória RAM e faz os ajustes necessários na tabela de páginas.

Uma vez a página carregada em memória, o processo pode continuar sua execução. O fluxograma da Figura mostrada no próximo slide apresenta as principais ações desenvolvidas pelo mecanismo de memória virtual.



Ações do mecanismo de memória virtual.

Gerenciamento de Memória – Memória Virtual

Nesse procedimento aparentemente simples há duas questões importantes. Primeiro, caso a memória principal já esteja cheia, uma ou mais páginas deverão ser removidas para o disco antes de trazer de volta a página faltante. Isso implica em mais operações de leitura e escrita no disco e portanto em mais demora para atender o pedido do processo.

Muitos sistemas, como o Linux e o Solaris, mantém um processo daemon com a finalidade de escolher e transferir páginas para o disco, ativado sempre que a quantidade de memória livre estiver abaixo de um limite mínimo.

Gerenciamento de Memória – Memória Virtual

Segundo, retomar a execução do processo que gerou a falta de página pode ser uma tarefa complexa. Como a instrução que gerou a falta de página não foi completada, ela deve ser re-executada.

Bibliografia

Bibliografia

TANENBAUM, A S. **Sistemas Operacionais Modernos**. 3 ed. São Paulo: Pearson Prentice Hall, 2010.

MAZIERO, Carlos A. **Sistemas Operacionais: Conceitos e Mecanismos**. DAINF – UTFPR, 2014.