Memória virtual - paginação e segmentação

MAC0344 - Arquitetura de Computadores
Prof. Siang Wun Song
Slides usados:
https://www.ime.usp.br/∼song/mac344/slides07-virtual-memory.pdf

Baseado em W. Stallings

Computer Organization and Architecture

Memória virtual

- Veremos Memória virtual
- Ao final das aulas, vocês saberão
  - A memória virtual armazena o programa em disco e apenas pedaços (páginas no caso de paginação ou segmentos no caso de segmentação) são trazidos para a memória quando necessários.
  - Com isso, o tamanho do programa pode exceder o tamanho da memória principal.
  - O estudo de memória virtual é um assunto de Sistemas Operacionais. Aqui veremos o apoio em hardware para ajudar o Sistema Operacional no gerenciamento da memória virtual

Apoio do sistema operacional por hardware

- O sistema operacional é o software que controla a execução de programas em um processador e gerencia os recursos.
- Para obter eficiência e velocidade, funções do sistema operacional como escalonamento de processos (*scheduling*) e gerenciamento de memória contam com o apoio de hardware.
- Esse apoio consiste em registradores e *buffers* especiais, e circuitaria para realizar tarefas de gerenciamento de recursos.
- Veremos agora o gerenciamento de memória virtual e o apoio de hardware. O uso de memória virtual tem os benefícios:
  - Um processo pode executar sem ter todas as instruções e dados dentro da memória principal.
  - O espaço de memória disponível ao programa pode exceder o tamanho da memória principal.

Serviços de um sistema operacional

De forma resumida, um sistema operacional (S.O.) típico fornece os seguintes serviços. Detalhes são estudados na disciplina Sistemas Operacionais.

- Criação do programa: editores, depuradores, etc. Não fazem parte propriamente do S.O. mas são acessíveis através do S.O.
- Execução do programa: carrega o programa (instruções e dados) na memória, inicializa e prepara dispositivos de entrada e saída, arquivos e demais recursos.
- Acesso a dispositivos de entrada e saída: livra do programador detalhes e conhecimento de instruções específicas, sinais de controle, etc.
- Acesso controlado a arquivos: S.O. se preocupa com os detalhes de acesso a disco, fita, etc., além de prover mecanismos de proteção sobre direito de acesso.
- Acesso a recursos do sistema: em caso de uso compartilhado, o S.O. controla o acesso a recursos compartilhados e resolve conflitos.
- Detecção e resposta a erros: S.O cuida de erros de hardware (como erros de memória e erros de dispositivos) e erros de software (como *overflow* em aritmética, acesso proibido de posições de memória) e responde de acordo (eventualmente abortando o programa).
- Contabilidade: coleta estatísticas de vários recursos, monitora desempenho.

S.O. é parte do sistema sendo controlado

- O S.O. controla os recursos de um computador.
- Em geral um controlador de um sistema é externo ao sistema.

  Exemplo: um semáforo que controla o fluxo de automóveis.

S.O. é parte do sistema sendo controlado

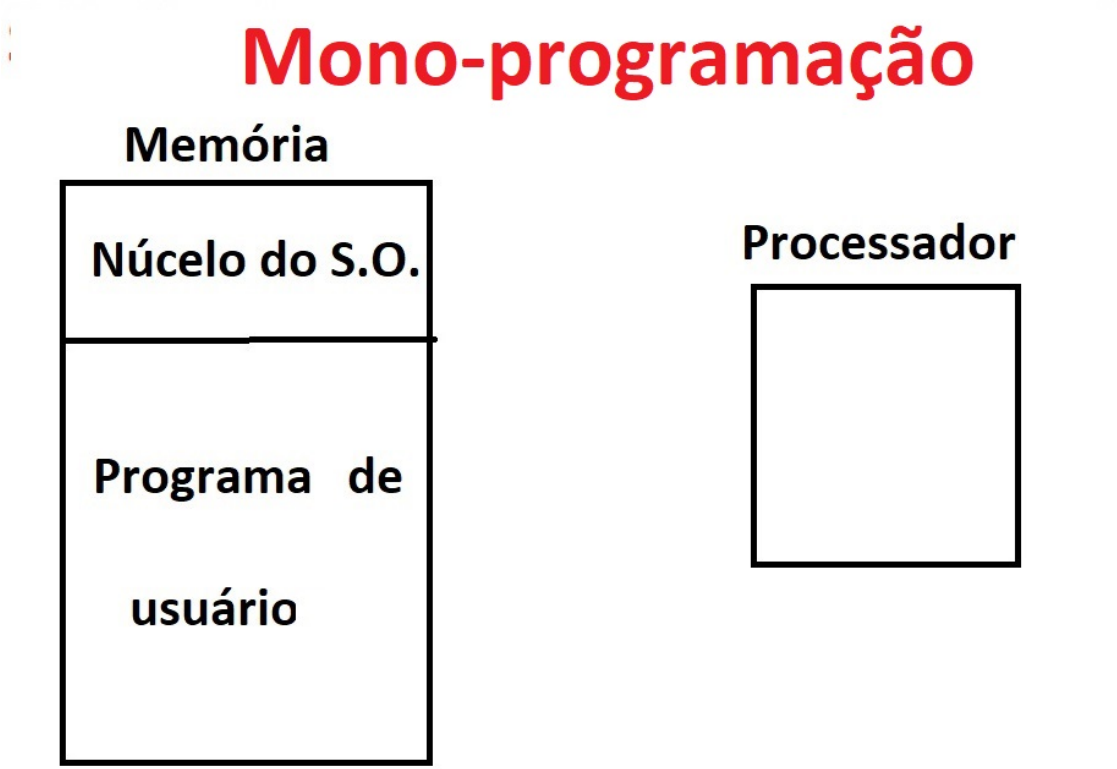
- No caso do S.O., ele é um software como qualquer outro e também usa os mesmos recursos como os demais.

Exemplo 1: Um carro oficial controlando o tráfego. O carro que controla também tem que usar a estrada junto com os demais veículos.

S.O. é parte do sistema sendo controlado

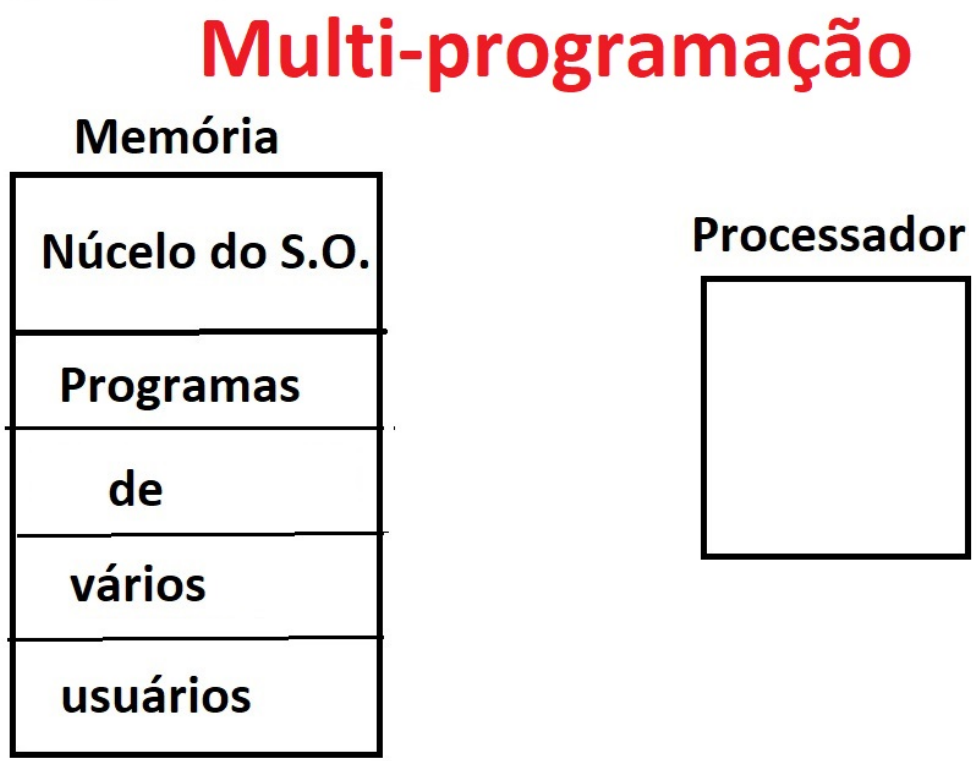
Exemplo 2: automóveis oficiais da Polícia Rodoviária controlam as rodovias para melhor fluir o tráfego. Os carros oficiais também precisam usar a rodovia, para poder prover socorro em caso de acidente. Para otimizar o bom uso da rodovia, carros oficiais precisam usar a mesma rodovia, numa aparente contradição.

Mono-programação: S.O. + um processo



- Num sistema de mono-programação, a memória é dividida em duas partes: uma para o S.O. (kernel ou núcleo) e outra para o programa sendo executado.

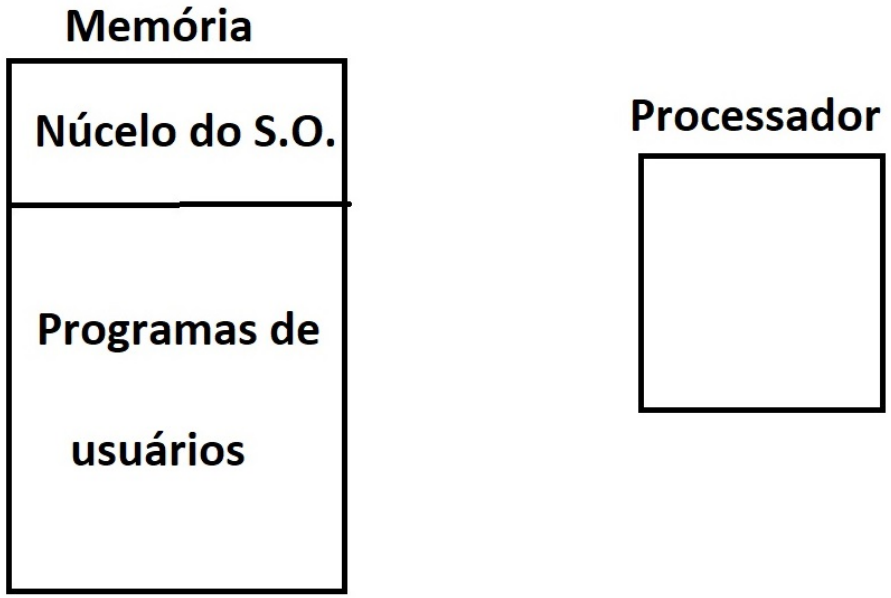
Multi-programação: S.O. + vários processos



- Num sistema de multi-programação, a parte da memória de usuário é subdividida para acomodar múltiplos processos.

S.O. é parte do sistema sendo controlado

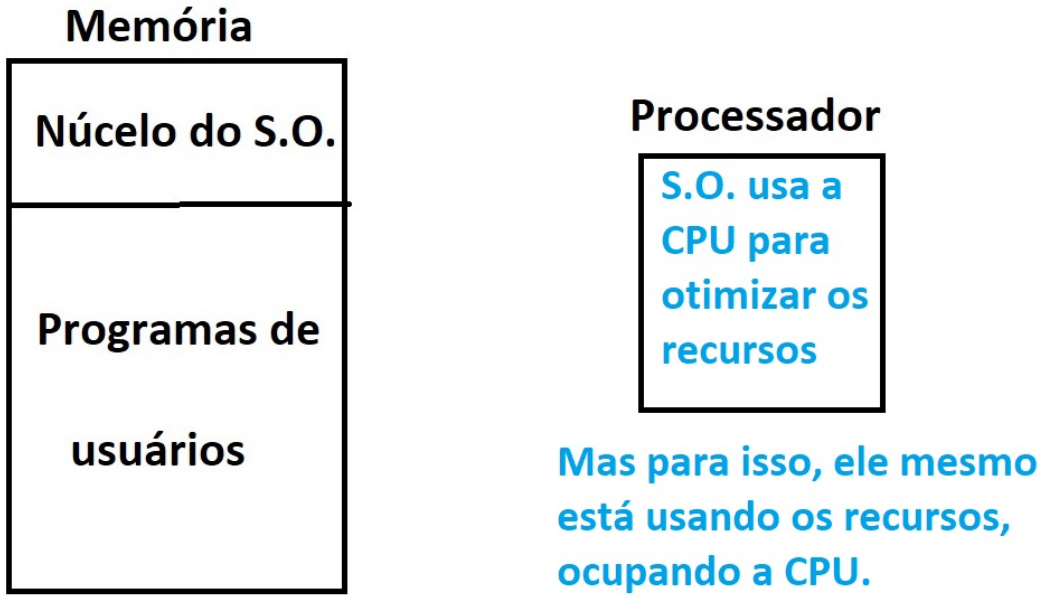
S.O. é software e executa no processador



- Uma parte do S.O., chamada *kernel* ou *núcleo*, reside na memória, junto com programas e dados de usuários.
- O S.O. passa o controle do processador para executar programas de usuários e, quando necessário, retoma o controle do processador.

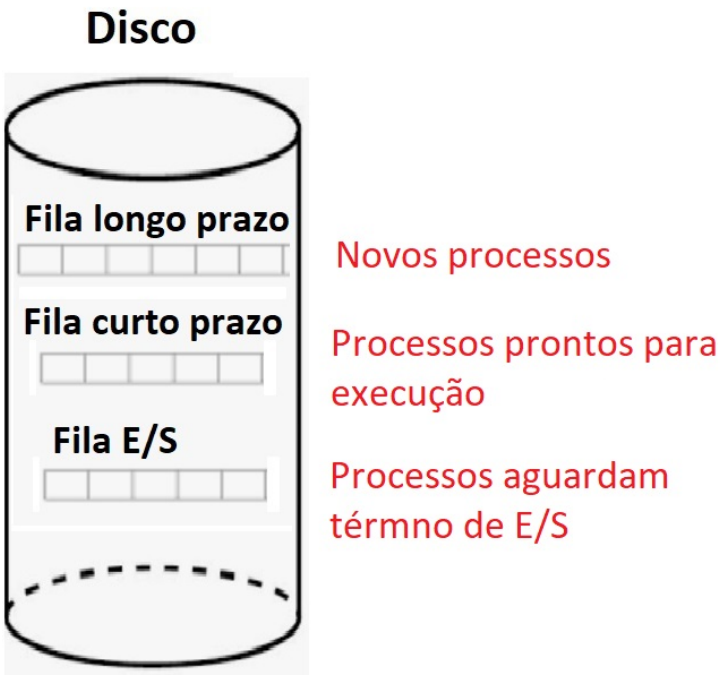
S.O. é parte do sistema sendo controlado

S.O. é software e executa no processador



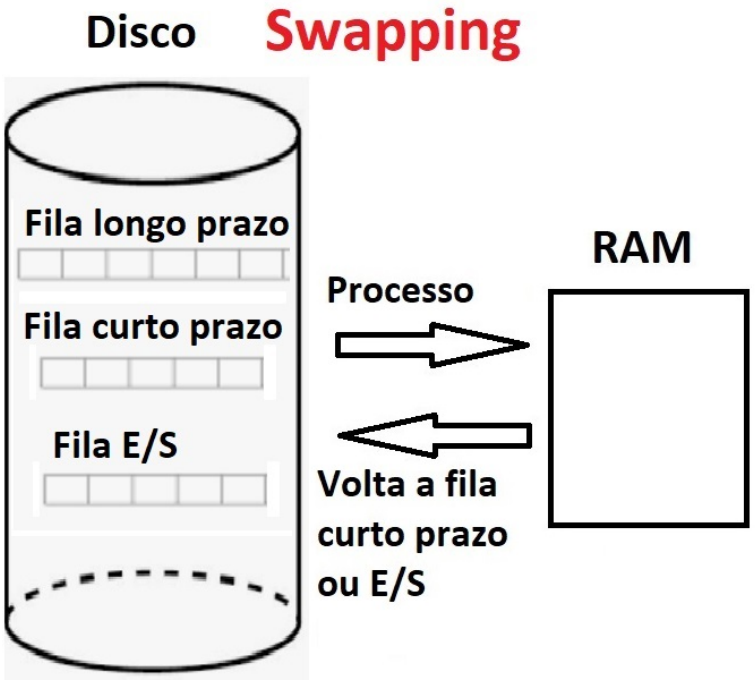
- O S.O. é software e executa no processador. É portanto parte do sistema sendo controlado.
- S.O. tem que eficiente e não pode consumir muito tempo do processador. Daí surge a necessidade de apoio de hardware para aumentar o seu desempenho..

Swapping



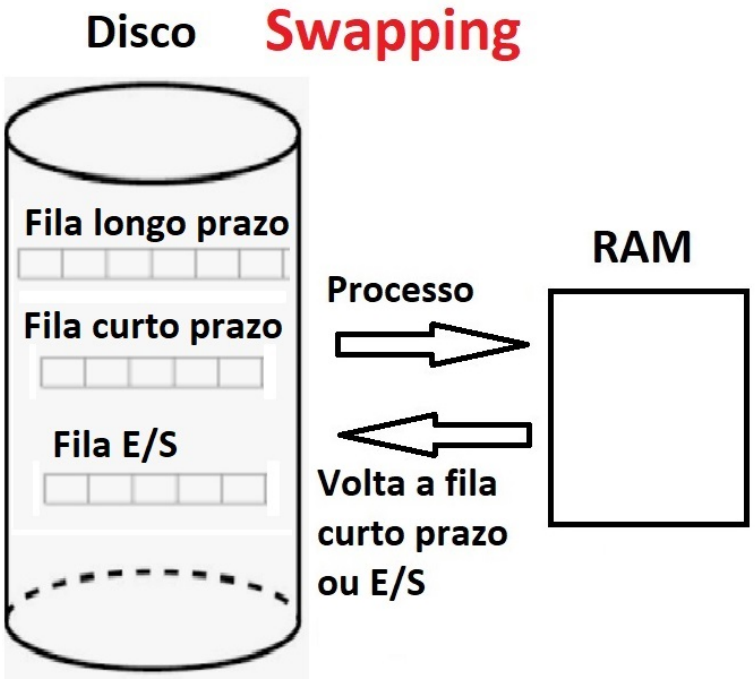
- O S.O. tem várias funções. Uma delas é o gerenciamento de memória.
- Uma outra função é o escalonamento de processos. O S.O. mantém e administra 3 filas.
- Processos que não estão em execução aguardam no disco, em umas das 3 filas: fila longo prazo, fila curto prazo, fila E/S.

Swapping



- Um processo é escolhido da fila de longo prazo e entra na fila de curto prazo.
- Um processo é tirado da fila de curto prazo e entra em execução.
- O processo usou sua quota de tempo (interrupção de relógio): É retirado pelo S.O. da memória e volta para fila de curto prazo.

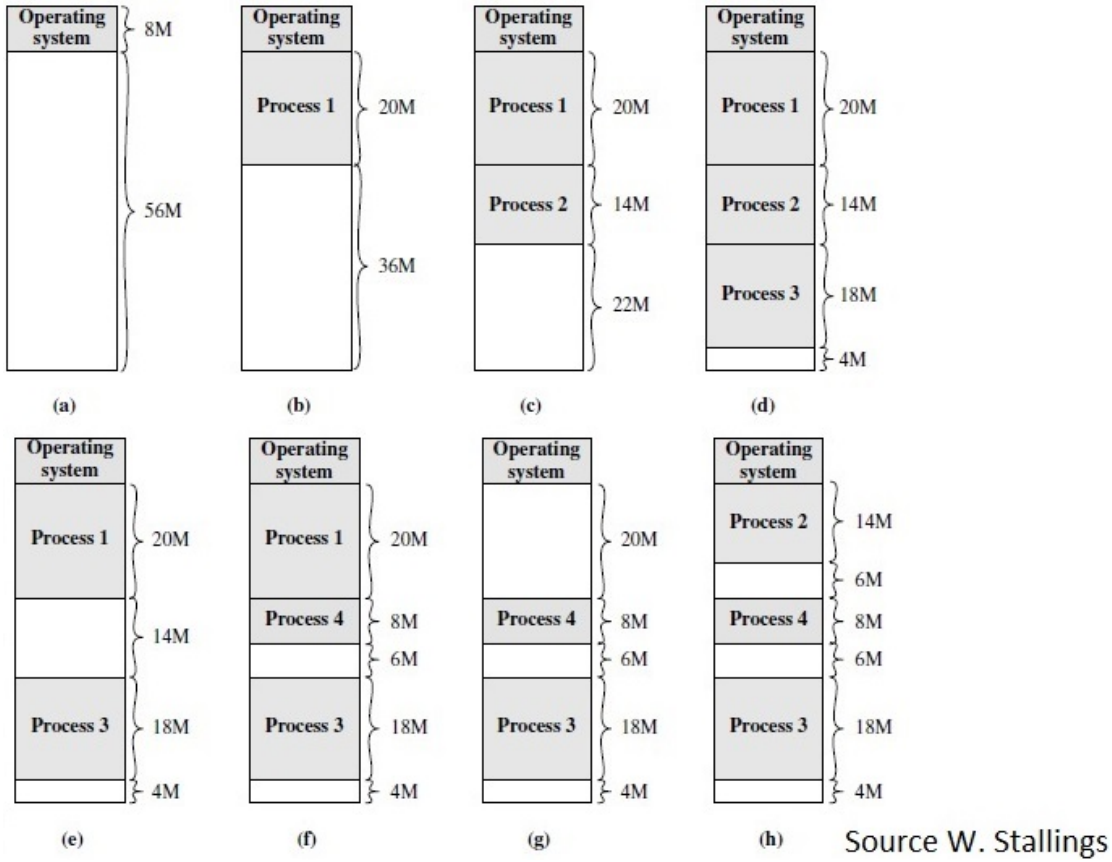
Swapping



- Um processo em execução necessita de fazer E/S. É retirado pelo S.O. da memória e colocado na fila de E/S.
- Terminada a E/S, o processo é retirado da fila de E/S e colocado na fila de curto prazo.
- O movimento de um processo para dentro e fora da memória recebe o nome de *swapping*.

Particionamento, fragmentação e compactação

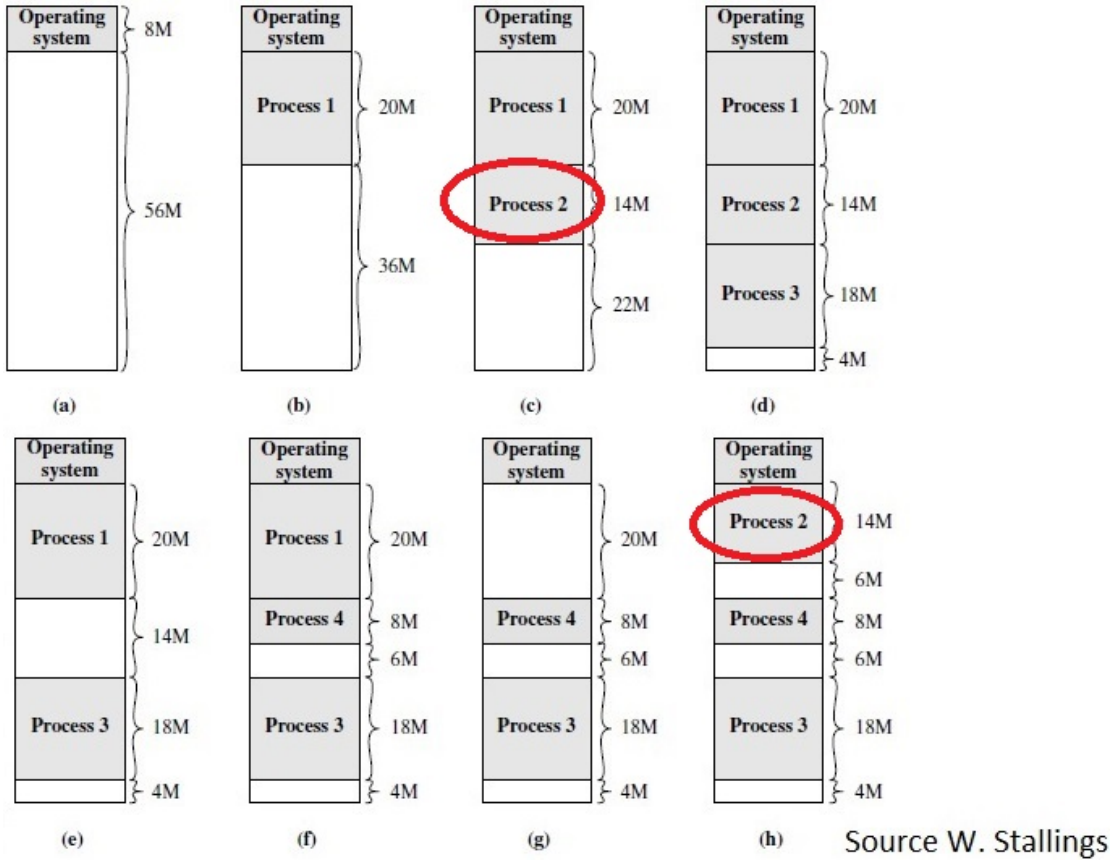
- Uma maneira simples de organizar a memória é dividir em partições de tamanho fixo (não necessariamente iguais).
- Quando um processo é trazido para dentro da memória, ele é colocado na menor partição que é suficiente para acomodá-lo.
- Como o tempo, a memória pode ficar fragmentada ao deixar pequenos buracos.
- Para resolver este problema, usa-se compactação, um processo demorado, em que o S.O. junta todos os processos em um mesmo bloco contíguo.



Source W. Stallings

Endereço lógico e endereço físico

- Fica óbvio que, através de *swapping*, um processo não necessariamente vai ser carregado sempre na mesma posição da memória. Ver Processo 2.
- A compactação pode deslocar um processo que já está na memória.
- Num processo há instruções e dados. Instruções podem conter endereços de dados da memória ou endereço de instruções usado em desvios.
- Quando um processo é recarregado na memória, esses endereços podem mudar. Para resolver isso, usam-se endereço lógico e endereço físico.

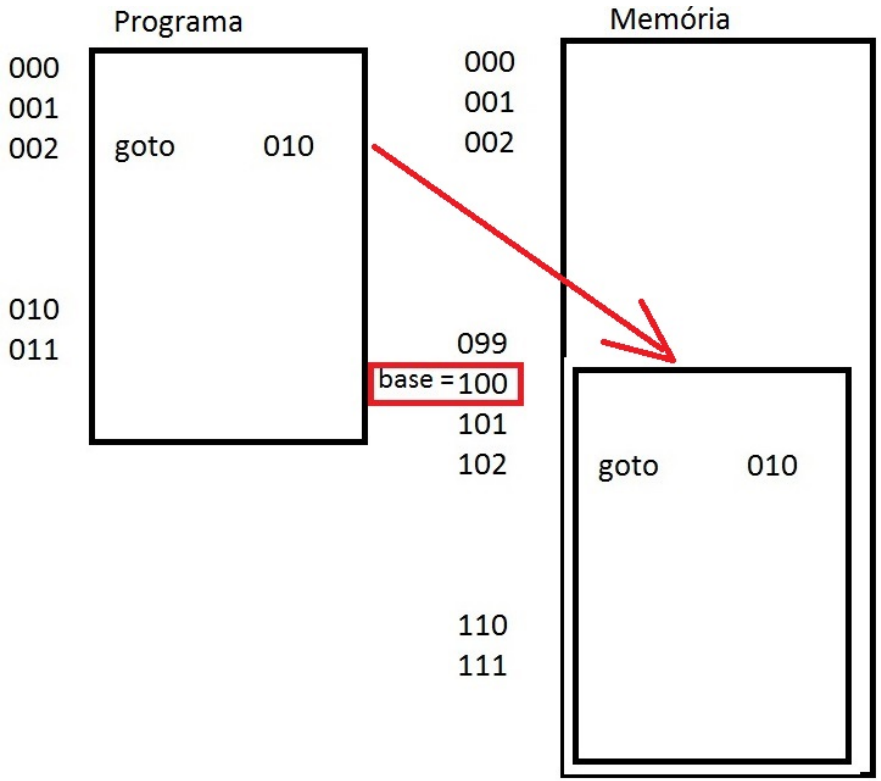


Source W. Stallings

Endereço lógico e endereço físico

- Endereço lógico: Expresso como uma posição relativa ao início do programa. Endereços em um programa contêm somente endereços lógicos.
- Endereço físico: É a localização real na memória.
- Quando um processo é carregado na memória com seu início na posição *base*, então para obter o endereço físico:

  Endereço físico = endereço lógico + base. Esse processo é feito por hardware.



Programa carregado na posição base = 100.

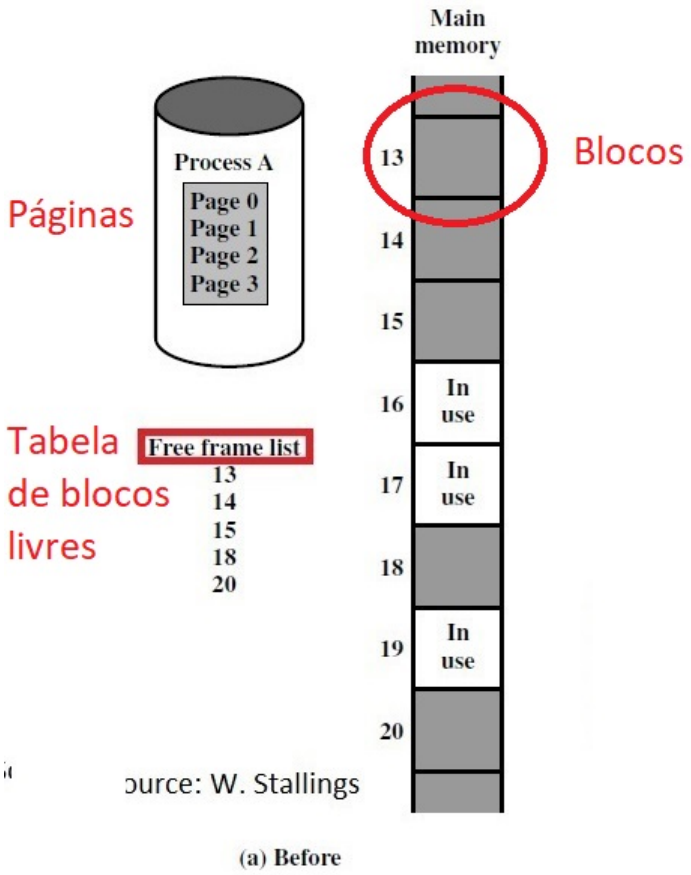
Ao executar a instrução 102 o endereço lógico 010 é convertido para base + 010 = 110

Memória virtual

- Com o aparecimento de programas cada vezes maiores em tamanho e a percepção de que não é necessário carregar todo o programa na memória, aparece o conceito de memória virtual.
  - Um processo pode executar sem ter todas as instruções e dados dentro da memória principal.
  - O espaço de memória disponível ao programa pode exceder o tamanho da memória principal.
- Veremos duas maneiras de implementação de memória virtual.
  - Paginação.
  - Segmentação.

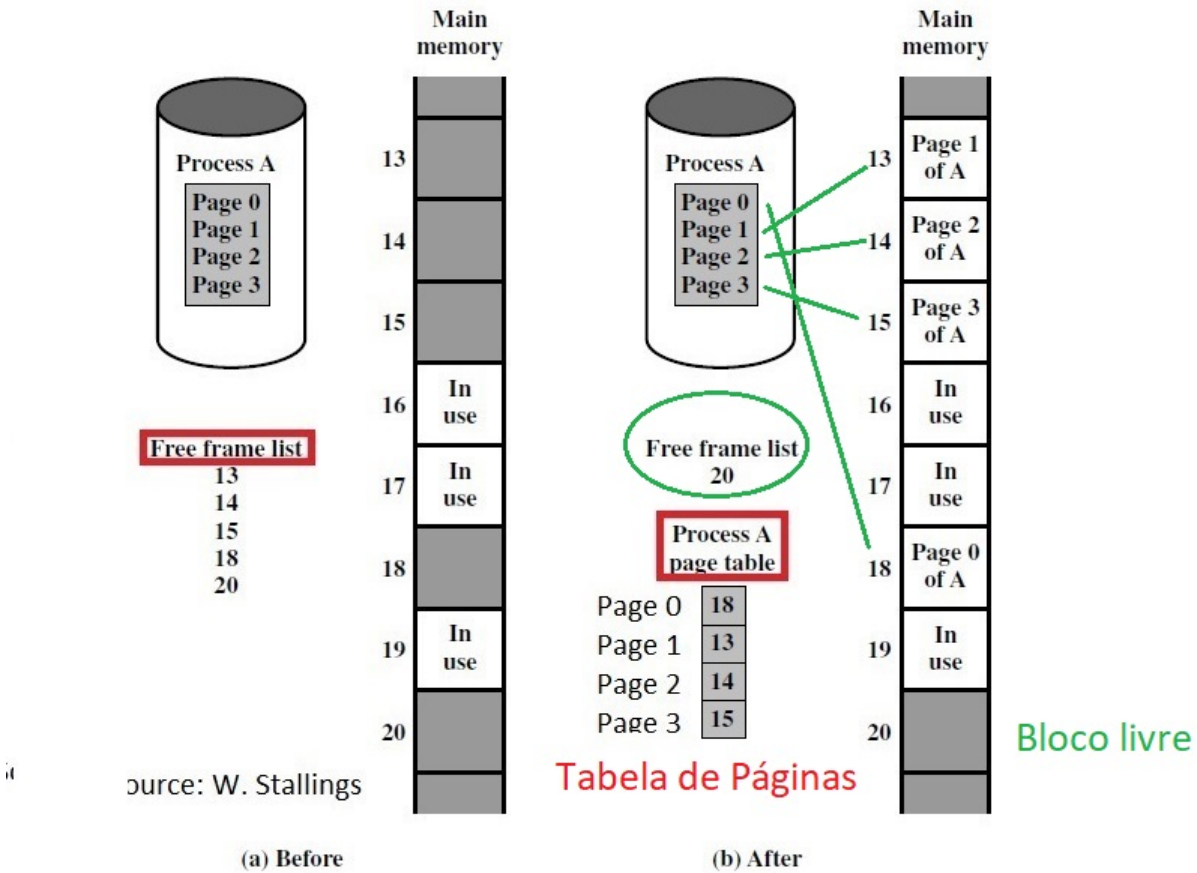
Paginação

- A memória é organizada em blocos ou quadros (*frames*) de tamanho fixo. A Tabela de Blocos Livres registra quais blocos estão livres (inicialmente todos).
- Cada processo é dividido em páginas de igual tamanho que o bloco. Assim, uma página de um processo pode ser carregado em um bloco de memória.



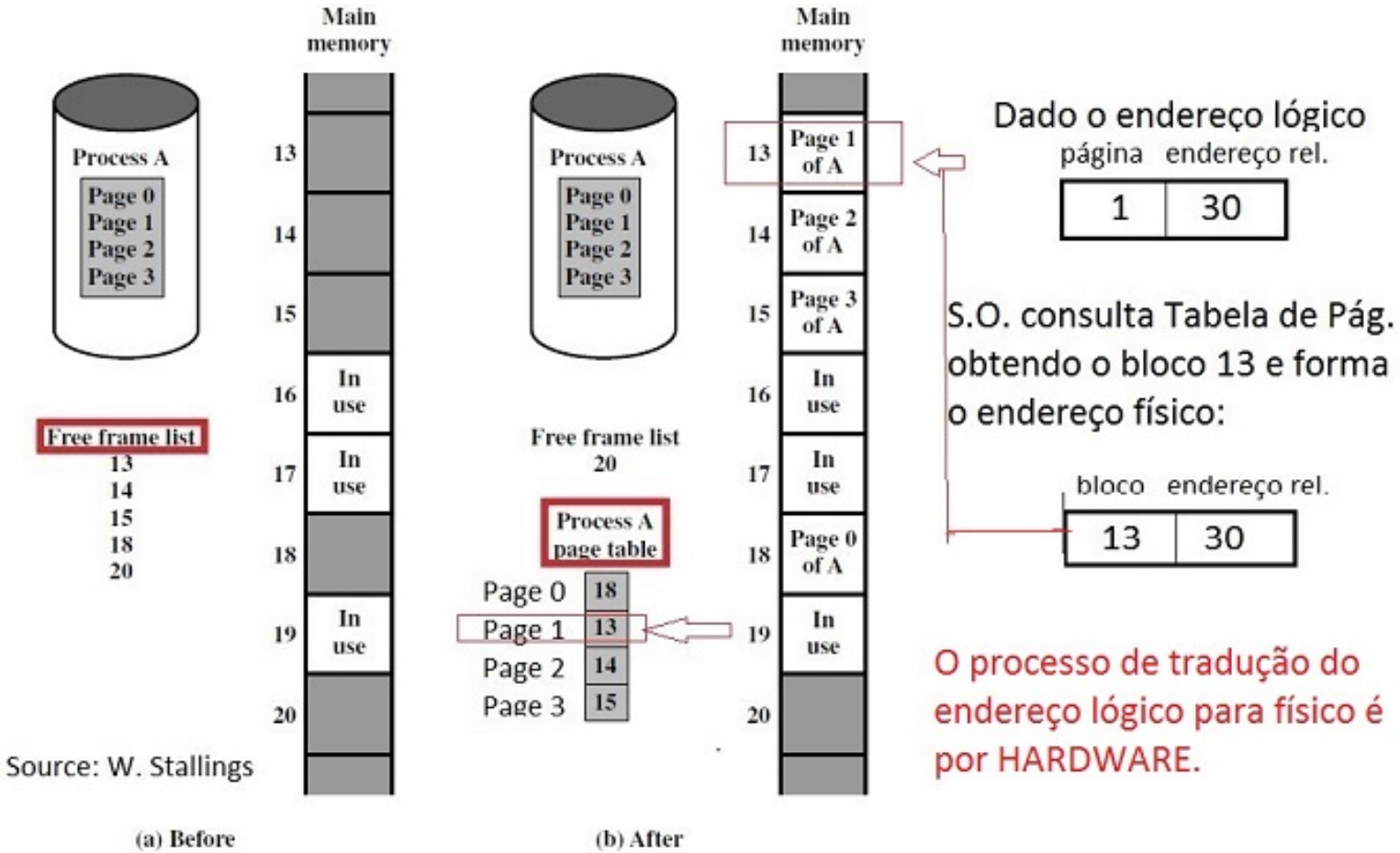
Paginação

- A Tabela de Páginas (uma para cada processo) registra em que bloco de memória cada página está carregada.
- As tabelas de páginas são mantidas pelo S.O. Se uma página é acessada mas não está na memória, o S.O. consulta a tabela de blocos livres, aloca a página no bloco selecionado e atualiza a tabela de páginas.



Paginação

- Endereço lógico: número da página e o endereço relativo (ou deslocamento) dentro da página. O S.O. transforma para um endereço físico, assim:
- Através da Tabela de Páginas, para uma dada página, o S.O. localiza o número do bloco em que ela está carregada.
- Com esse bloco e o endereço relativo, o S.O. forma o endereço físico.
- Endereço físico: número do bloco e o endereço relativo (ou deslocamento) dentro do bloco.



Memória virtual com paginação sob demanda

- Mesmo para um programa grande, num período de tempo curto, a execução pode se restringir a apenas um pequeno trecho do programa e talvez a uma ou duas estruturas de dados. É o Princípio de localidade:
  - Localidade temporal: reuso de dados ou instruções dentro de um curto período de tempo.
  - Localidade espacial: reuso de dados relativamente próximos.
- O mecanismo de paginação viabilizou a implementação de memória virtual, em que um programa reside no disco e apenas páginas necessárias são trazidas à memória, sob demanda, é a chamada Paginação sob Demanda.
- Working Set é o conjunto de páginas acessadas por um processo durante um intervalo de tempo. Pelo princípio de localidade, o Working Set não é grande e por isso a paginação sob demanda funciona.

Memória virtual com paginação sob demanda

Tabela de Páginas

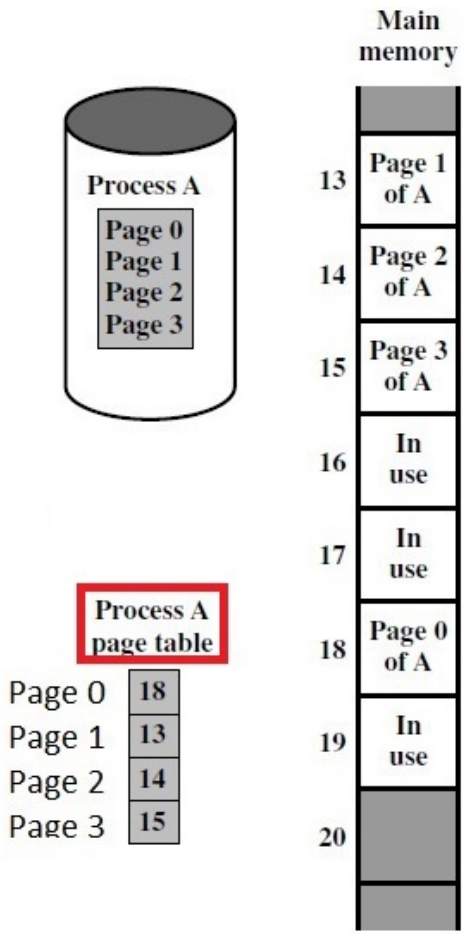
Página	Bloco
0	18
1	13
2	14
3	15
4	x
5	x
6	20
7	x
8	x
9	x
10	x

Algumas páginas estão na memória.

A maior parte das páginas não estão na memória.

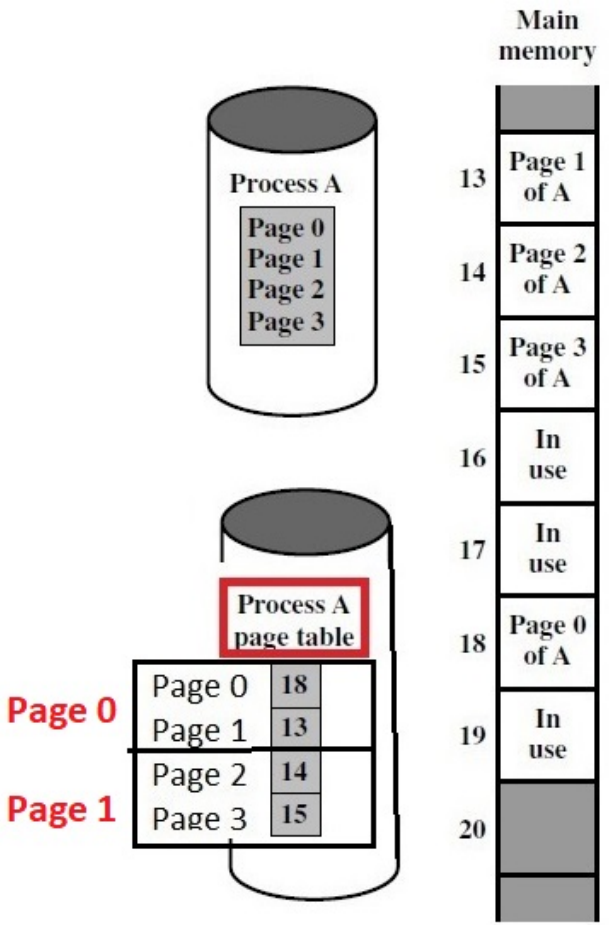
- Se a execução precisa uma página que não está na memória, o S.O. será acionado através de uma interrupção de falha de página (ou *page fault*) a fim de trazer a página para a memória.
- Para isso, um bloco livre é usado para receber a página. Se não há blocos livres, então o S.O. seleciona e desocupa uma página na memória, dando o bloco liberado para a nova página. Algum algoritmo de substituição de página é usado, e.g. LRU (visto em Memória Cache).

Implementação da Tabela de Páginas



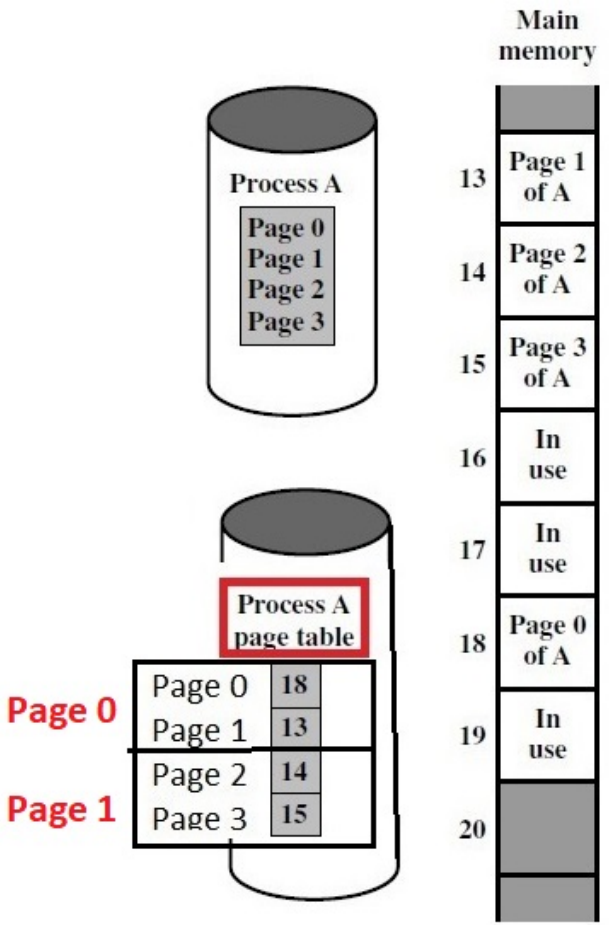
- Cada processo tem uma Tabela de Páginas, onde há uma entrada para cada página do processo.
- Com a memória virtual, um processo pode consistir de um grande número de páginas, impossibilitando alocar a Tabela de Páginas dentro da memória física.
- Então onde vamos alocar as muitas tabelas de páginas?

Implementação da Tabela de Páginas



- As Tabelas de Páginas desses processos são implementadas em disco, também divida em páginas. Apenas parte de cada tabela está armazenada em blocos de memória física.
- No exemplo, a tabela de páginas está no disco. Page 0 contém a Page 0 e Page 1. Page 1 contém a Page 2 e Page 3.

Implementação da Tabela de Páginas



- Para acessar a Page 3, primeiro acessamos Page 1. Se Page 1 está na memória, então sabemos que Page 3 está no bloco 15.
- Se Page 1 não está na memória, temos que carregá-la do disco e então prosseguimos como acima. Se a página desejada não foi ainda carregada na memória, então temos que acessar o disco e carregá-la na memória.

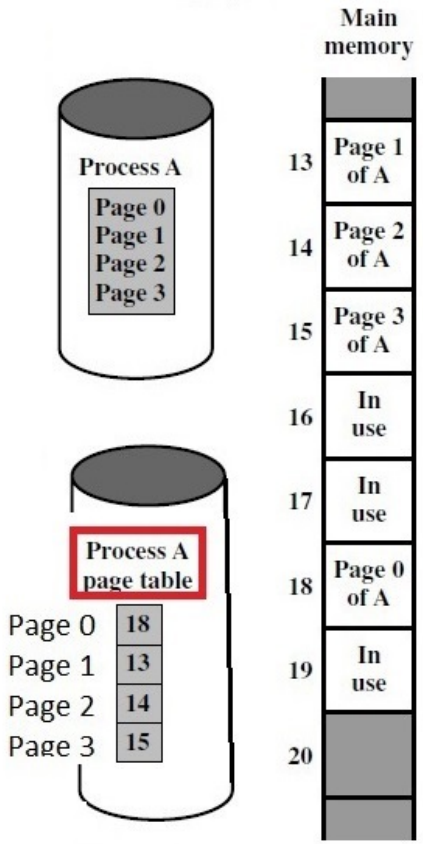
Implementação da Tabela de Páginas

Ex: Página 1 e Página 3 são acessadas. Então as linhas corresp. da tabela de pág. vão para a cache TLB.

Cache TLB

Página 1 Bloco 13
Página 3 Bloco 15
...

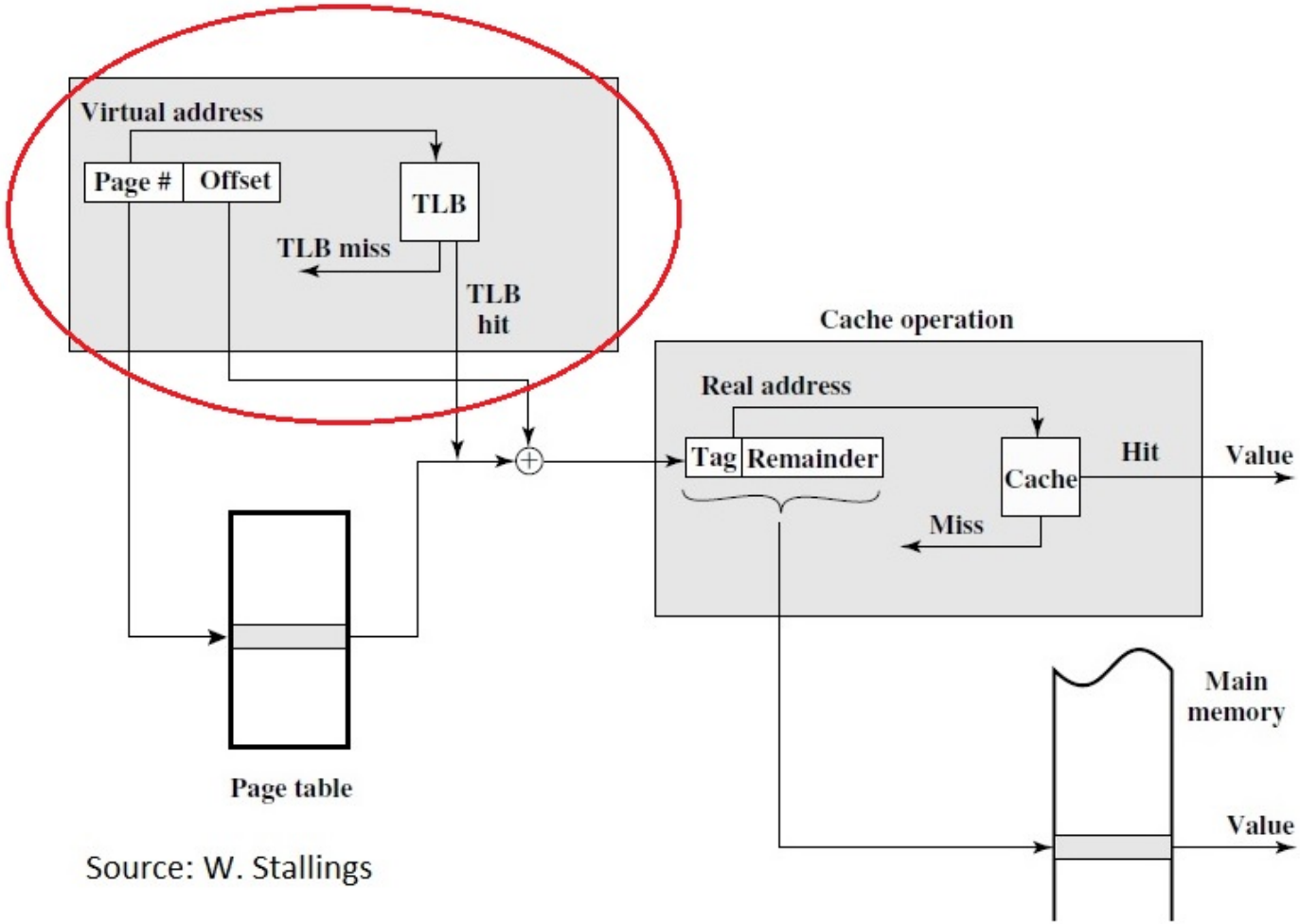
Quando a Página 1 ou a Página 3 é novamente acessada, então o bloco desejado já está na TLB.



- Com memória virtual, cada referência à memória virtual pode acarretar em dois acessos à memória física: um para acessar entrada desejada da Tabela de Páginas, e outro para a página desejada.
- Para evitar esse problema, é usada uma memória cache especial *Translation Lookaside Buffer (TLB)*, para entradas da Tabela de Páginas. Isso agiliza quando a mesma página é acessada depois de um primeiro acesso.

Translation Lookaside Buffer (TLB)

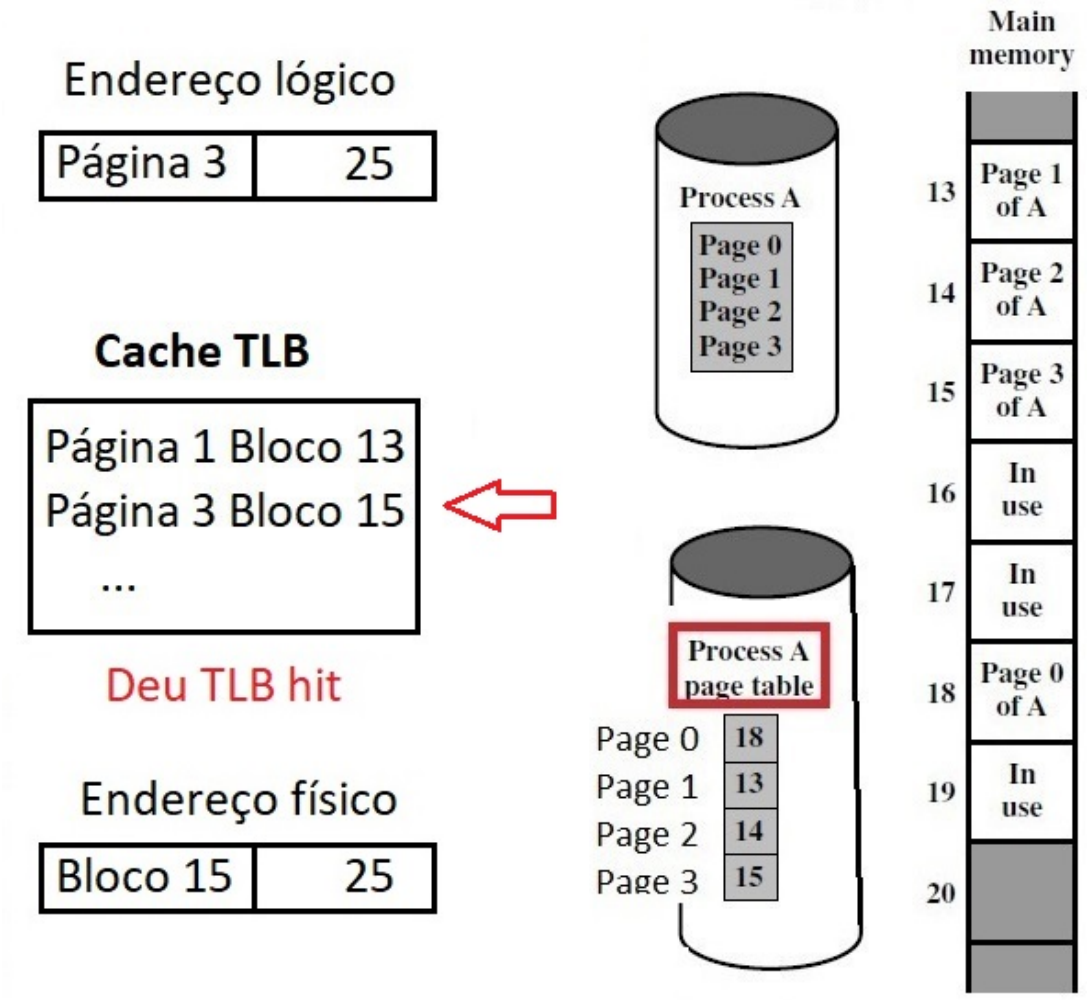
- O *Translation Lookaside Buffer* TLB é uma memória cache que contém as entradas (i.e. as linhas) da Tabela de Páginas mais recentemente usadas.
- Para localizar uma dada página, TLB é consultado. Se encontrar (*TLB hit*), o número do bloco correspondente é obtido. Pelo princípio de localidade, há grande probabilidade de isso ocorrer.



Source: W. Stallings

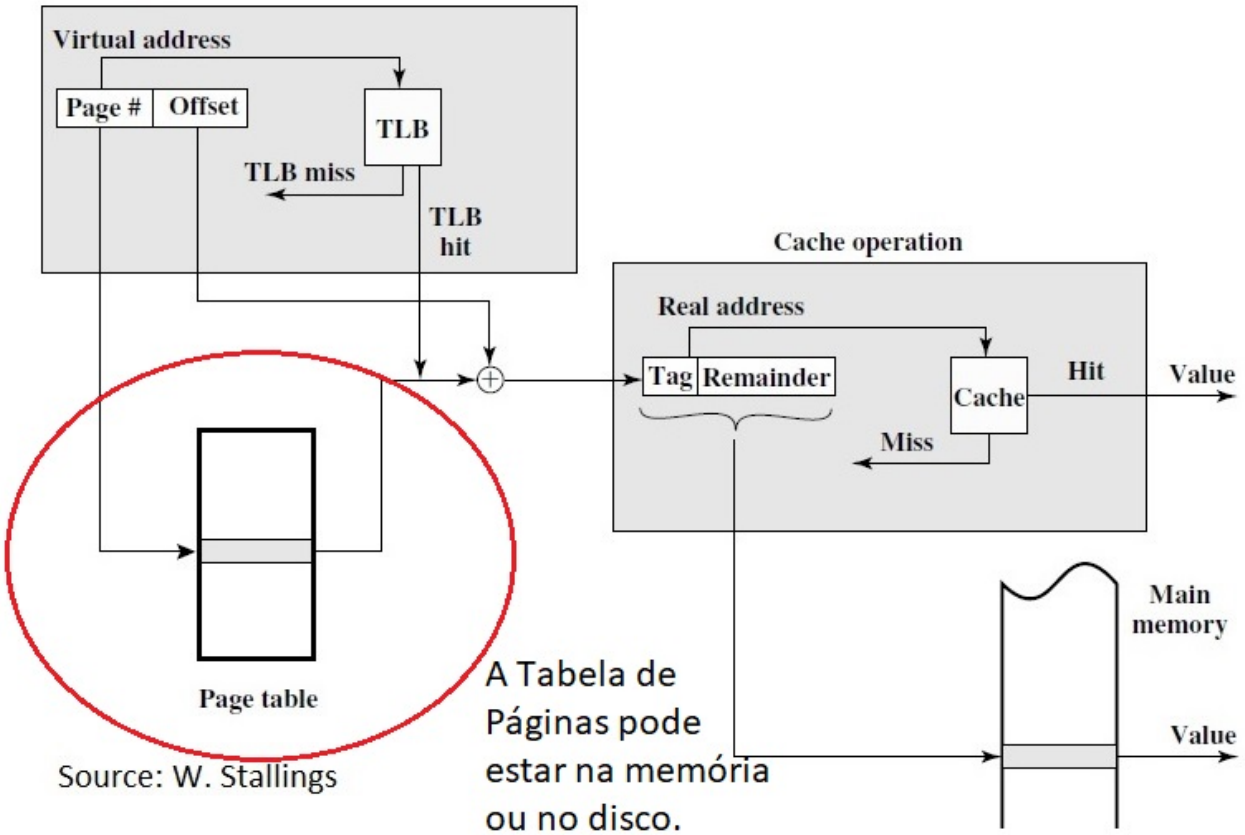
Translation Lookaside Buffer (TLB)

- O *Translation Lookaside Buffer* TLB é uma memória cache que contém as entradas (i.e. as linhas) da Tabela de Páginas mais recentemente usadas.
- Para localizar uma dada página, TLB é consultado. Se encontrar (*TLB hit*), o número do bloco correspondente é obtido. Pelo princípio de localidade, há grande probabilidade de isso ocorrer.



Translation Lookaside Buffer (TLB)

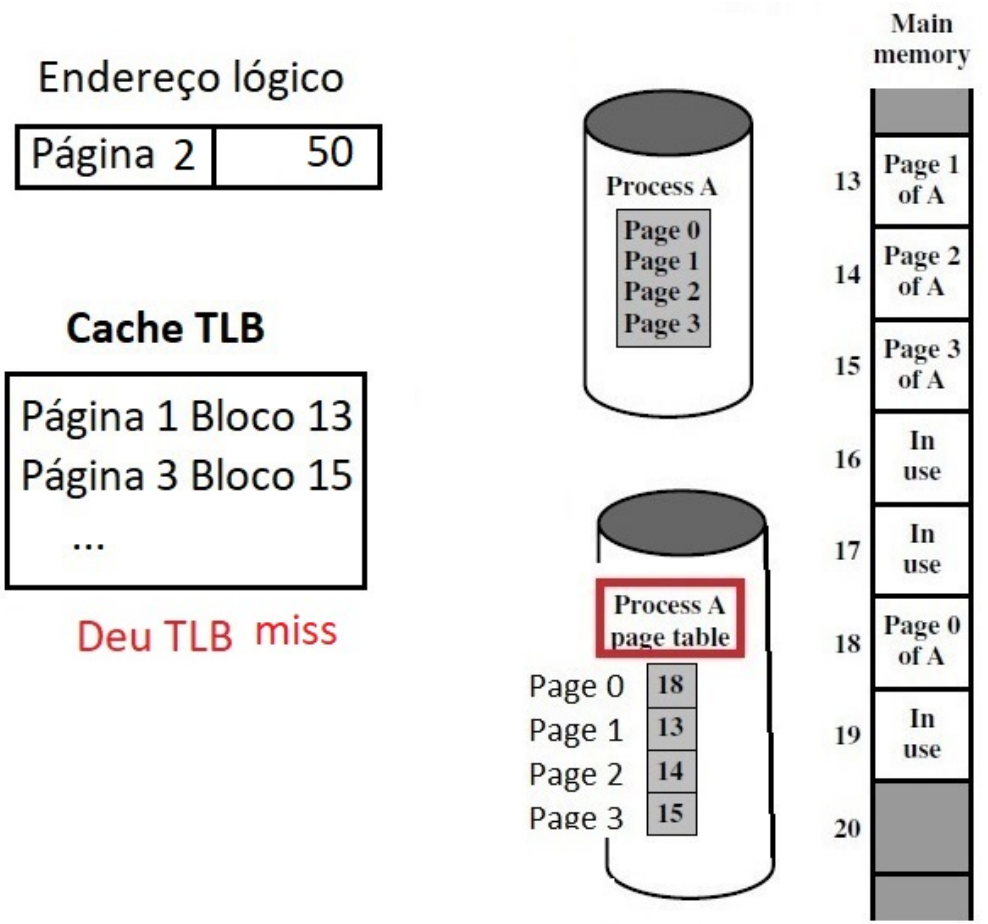
- Se a parte da Tabela de Páginas presente no TLB não contém a página desejada (*TLB miss*), então passa-se a acessar a parte da Tabela de Páginas na memória.
- Se não encontrar na memória, então ocorre uma interrupção *page fault* e é trazida a entrada desejada da Tabela de Páginas do disco, atualizando a parte de Tabela de Páginas na memória física.



Source: W. Stallings

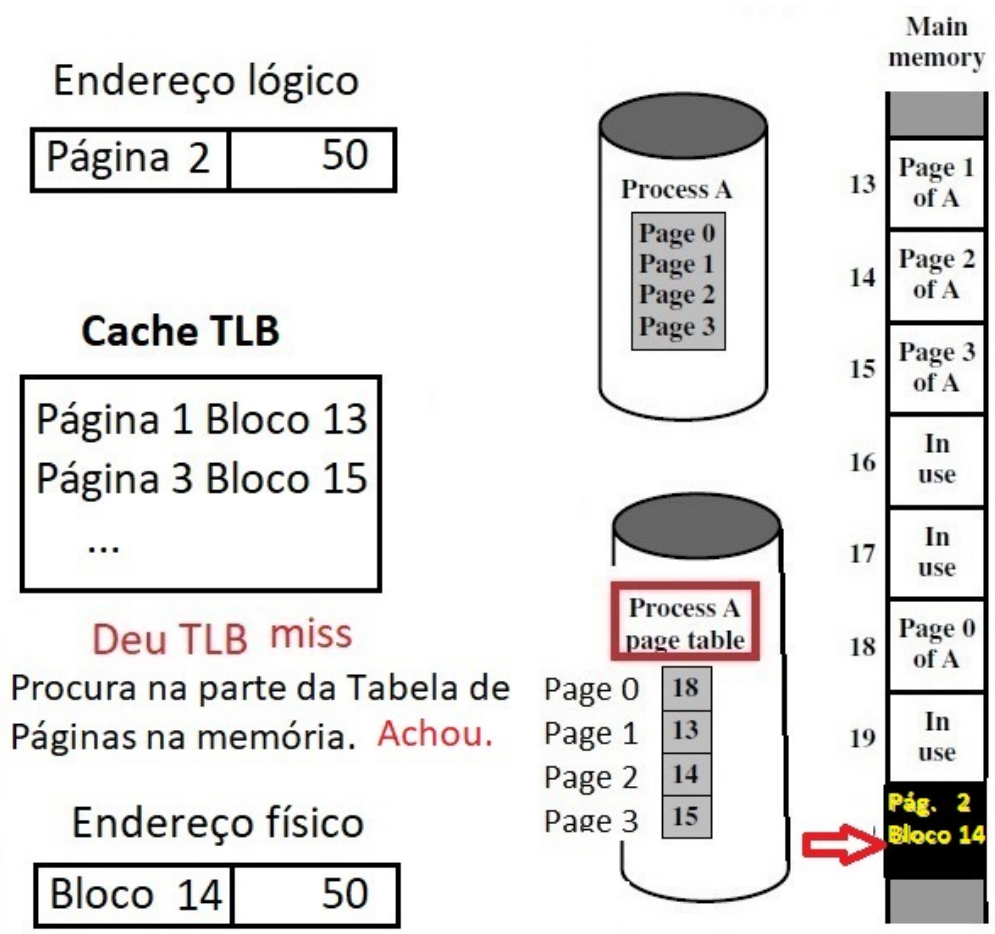
Translation Lookaside Buffer (TLB)

- Se a parte da Tabela de Páginas presente no TLB não contém a página desejada (*TLB miss*), então passa-se a acessar a parte da Tabela de Páginas na memória.
- Se não encontrar na memória, então ocorre uma interrupção *page fault* e é trazida a entrada desejada da Tabela de Páginas do disco, atualizando a parte de Tabela de Páginas na memória física.



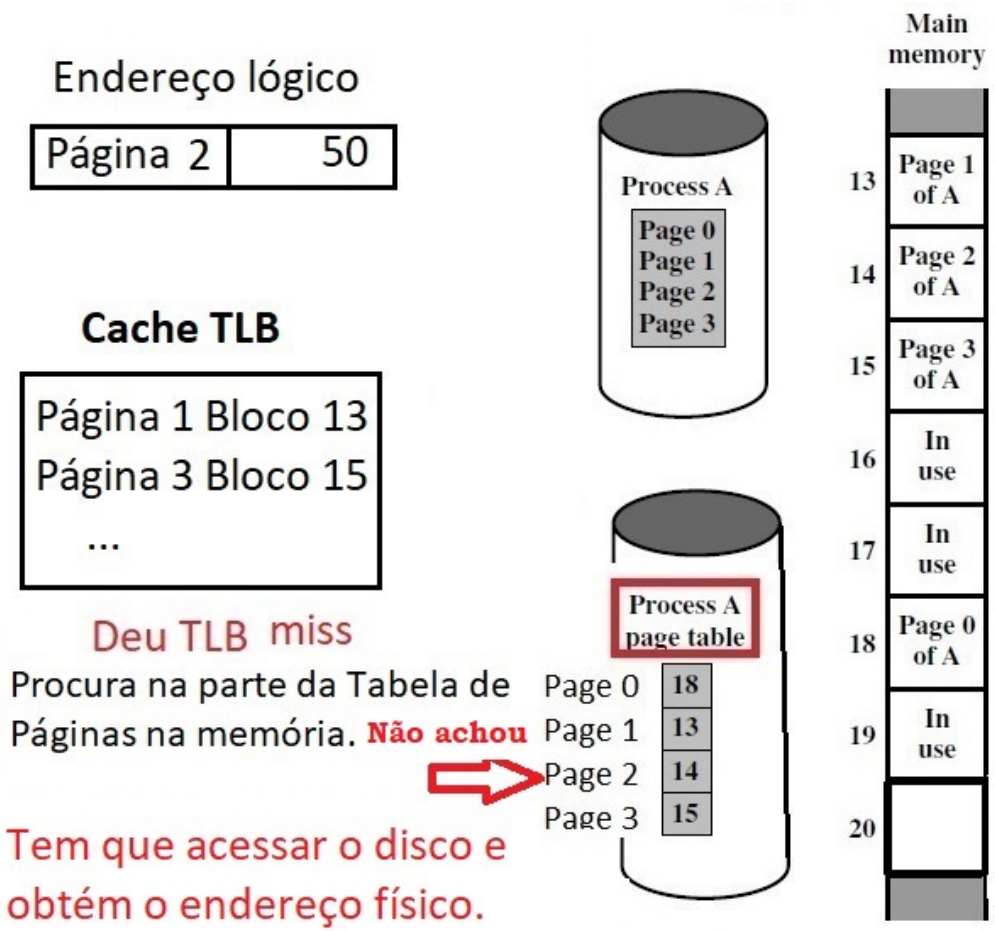
Translation Lookaside Buffer (TLB)

- Se a parte da Tabela de Páginas presente no TLB não contém a página desejada (*TLB miss*), então passa-se a acessar a parte da Tabela de Páginas na memória.
- Se não encontrar na memória, então ocorre uma interrupção *page fault* e é trazida a entrada desejada da Tabela de Páginas do disco, atualizando a parte de Tabela de Páginas na memória física.



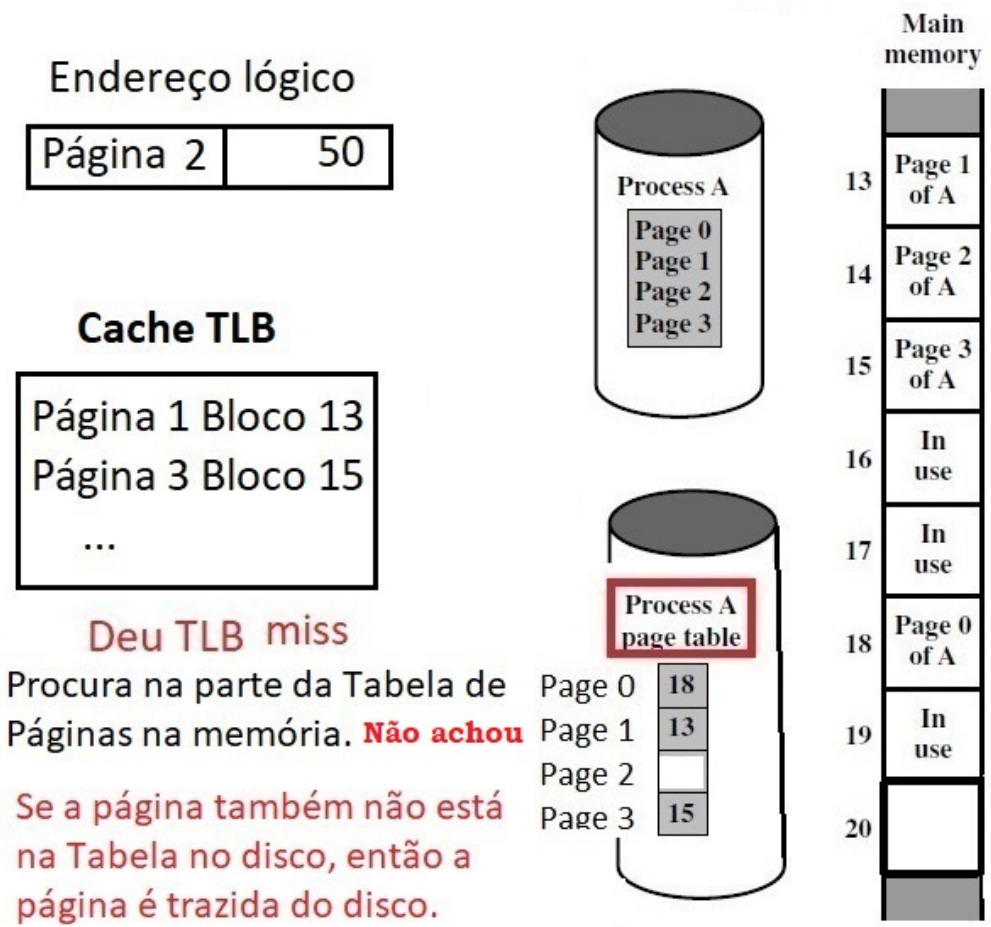
Translation Lookaside Buffer (TLB)

- Se a parte da Tabela de Páginas presente no TLB não contém a página desejada (*TLB miss*), então passa-se a acessar a parte da Tabela de Páginas na memória.
- Se não encontrar na memória, então ocorre uma interrupção *page fault* e é trazida a entrada desejada da Tabela de Páginas do disco, atualizando a parte de Tabela de Páginas na memória física.



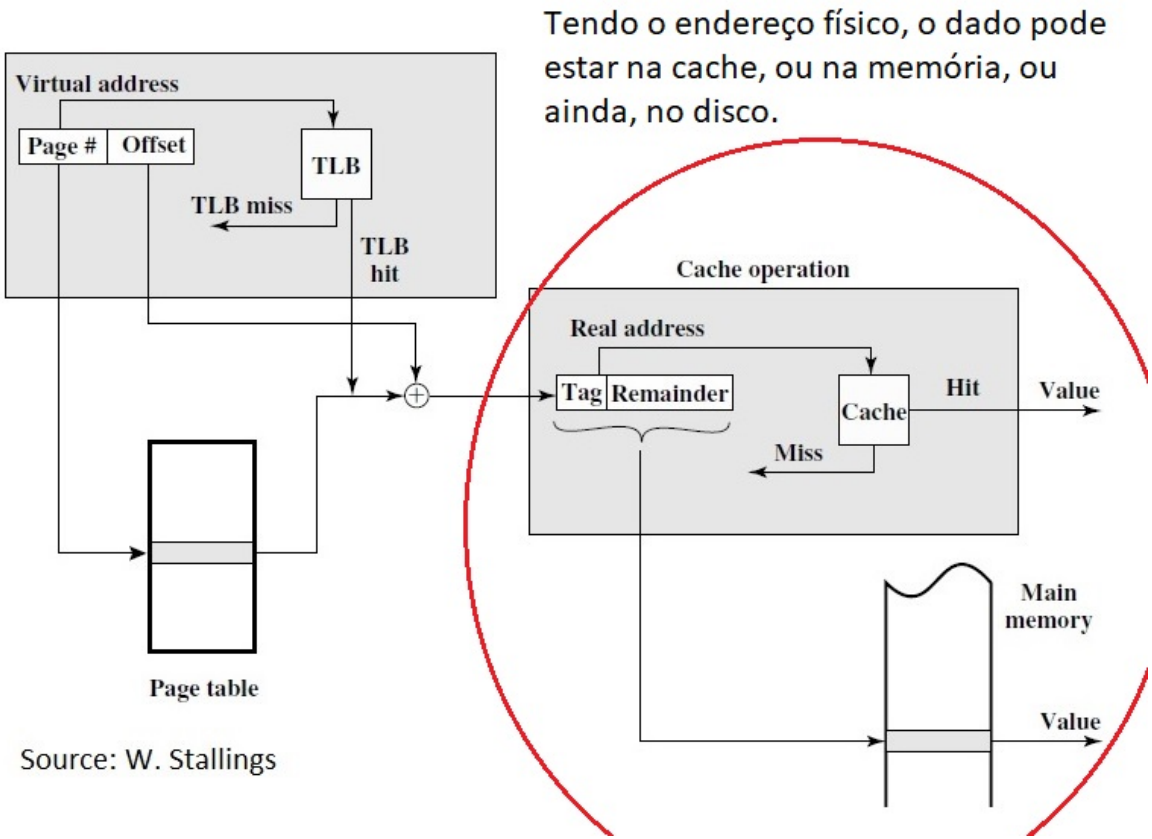
Translation Lookaside Buffer (TLB)

- Se a parte da Tabela de Páginas presente no TLB não contém a página desejada (*TLB miss*), então passa-se a acessar a parte da Tabela de Páginas na memória.
- Se não encontrar na memória, então ocorre uma interrupção *page fault* e é trazida a entrada desejada da Tabela de Páginas do disco, atualizando a parte de Tabela de Páginas na memória física.



Translation Lookaside Buffer (TLB)

- O endereço físico obtido (bloco + endereço relativo) correspondente ao endereço lógico é então buscado na memória cache normal. No caso de *cache miss*, a memória física é acessada.
- Veja a complexidade envolvida em uma simples referência à memória: 1) uma referência à Tabela de Páginas, que pode estar no TLB, na memória, ou disco, e 2) o endereço referenciado pode estar na cache, memória ou disco.



Tamanho de uma página

Qual o tamanho de uma página?

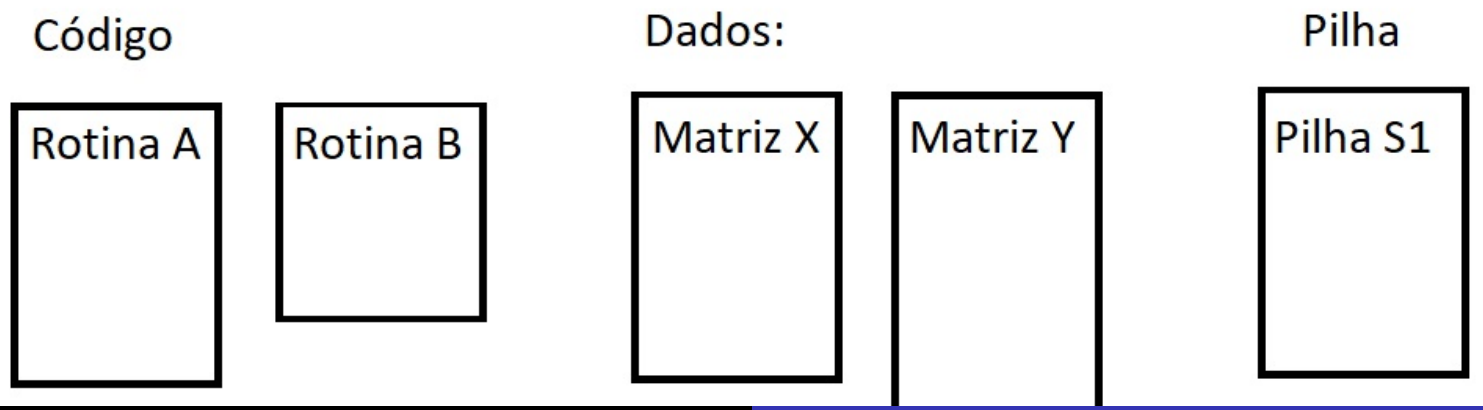
- Tipicamente, a página tem tamanho de 4 Kbytes.
- No artigo Using 4KB page size for Virtual Memory is obsolete, os autores discutem esse assunto.
- Observam que uma página maior pode melhorar o desempenho e cobertura da TLB.
- Propõem páginas de 16 Kbytes para código e de 256 Kbytes para dados.

Segmentação

- Segmentação é uma outra maneira de subdividir a memória.

  Difere da paginação nos aspectos:
  - Paginação: invisível ao programador, serve para prover um espaço maior de endereçamento. Memória dividida em páginas de igual tamanho, com qualquer conteúdo.
  - Segmentação: em geral visível ao programador, serve para organizar programas e dados, associando atributos de privilégio e proteção a instruções e dados. Os segmentos de um programa residem no disco. Apenas segmentos em uso são carregados na memória. O tamanho de um segmento não é fixo.
- Em Intel x86 há segmentos código, segmentos dados e segmentos pilha.

Exemplos de segmentos:



Segmentação

- Segmentação viabiliza facilmente o compartilhamento de procedimentos ou rotinas.
- Pode haver vários segmentos de programas e de dados, com diferentes direitos de acesso atribuídos a cada um.
- Referência ou endereço à memória consiste de (número de segmento, endereço relativo ou deslocamento).

Segmentação

Vantagens da segmentação:

- Simplifica o crescimento no tamanho de estruturas de dados. O S.O. pode aumentar ou diminuir o tamanho de um segmento contendo uma estrutura de dado.
- Em caso de alteração de alguns segmentos, facilita a recompilação sem ter que religar e recarregar todo o programa.
- Facilita o compartilhamento entre processos: um usuário pode colocar um utilitário ou uma tabela útil em um segmento que pode ser acessado por outros usuários.
- Facilita a proteção: privilégios de acesso podem ser atribuídos de maneira conveniente.

Segmentação combinada com paginação:

- Por outro lado, a paginação tem a vantagem de prover uma forma eficiente de gerenciamento de memória.
- A fim de combinar as vantagens de ambas, alguns sistemas equipam o hardware e S.O. para prover segmentação e paginação.

Gerenciamento de memória do Intel Pentium II

- O Intel Pentium II possui hardware para segmentação e paginação. Ambos os mecanismos podem ser desativados, permitindo ao usuário a escolha de uma das 4 formas de visualizar a memória:
  - Memória sem segmentação e sem paginação: o endereço virtual é igual ao endereço físico, útil para aplicações de controle de baixa complexidade e alto desempenho.
  - Memória sem segmentação e com paginação: usado no S.O. Berkeley Unix.
  - Memória com segmentação e sem paginação: tem a vantagem de ter a Tabela de Segmentos *on chip* quando o segmento está na memória.
  - Memória com segmentação e com paginação: segmentação usada para definir partições lógicas de memória e paginação usada para gerenciar a alocação de memória dentro da partição. Usado em Unix System V.

Gerenciamento de memória do Intel Pentium II

- Um endereço virtual ou lógico consiste de:
  - Um seletor de segmento de 16 bits (dos quais 2 bits são para o mecanismo de proteção, sobrando 14 bits para especificar o segmento).
  - Um deslocamento (*offset*) de 32 bits.
- Qual o espaço total da memória virtual?
  - O tamanho de um segmento é no máximo $2^{32} = 4$ Gbytes.
  - Com segmentação, o espaço total da memória virtual pode ser $2^{14+32} = 2^{46} = 64$ TBytes.

Mecanismos de proteção no Intel Pentium II

- No seletor de segmento de 16 bits, 2 bits são para o mecanismos de proteção.
- Esses 2 bits especificam o nível de privilégio do segmento, que determina qual segmento de programa pode acessar qual segmento de dado.
  - 4 níveis de privilégio: Do nível 0 (mais protegido) ao nível 3 (menos protegido).
  - Um segmento de programa mais protegido pode acessar um segmento de dado menos protegido ou de igual nível.
  - Isto é: Um segmento de programa de nível de privilégio i pode acessar um segmento de dado de nível de privilégio j se $i \leq j$.

Mecanismos de proteção no Intel Pentium II

- Dois bits especificam o nível de privilégio do segmento,
- A definição dos níveis de privilégio depende de cada S.O.
- Em geral os níveis 0 e 1 são reservados para o S.O. sendo o nível 0 para gerenciamento de memória e proteção de acesso.
- Certas instruções para gerenciamento de memória só podem ser executadas no nível 0.

Mecanismos de proteção no Intel Pentium II

- No seletor de segmento de 16 bits, 2 bits são para o mecanismos de proteção.
- Esses 2 bits podem especificar um atributo de acesso do segmento.
  - No caso de segmento de dados, o atributo de acesso pode especificar se o acesso permitido é do tipo *read/write* ou *read only*.
  - No caso de segmento de programa, o atributo de acesso pode especificar se o acesso permitido é do tipo *read/execute* ou *read only*.

Como foi o meu aprendizado?

- Sabemos da importância do apoio de hardware no bom desempenho do S.O. Cite exemplos desses apoios de hardware.
- Conversão do endereço lógico para endereço físico quando o processo é carregado na memória. (Ver slide 11.)
- *TLB* ou *Translation Lookaside Buffer*: uma memória cache para entradas da tabela de páginas. (Ver slides 21 a 25.)
- Mecanismo de conversão do endereço virtual para o endereço final, com segmentação e paginação. (Ver slides 34 e 35.)

Como foi o meu aprendizado?

- Sabemos da importância do apoio de hardware no bom desempenho do S.O. Cite exemplos desses apoios de hardware.
- Conversão do endereço lógico para endereço físico quando o processo é carregado na memória. (Ver slide 11.)
- *TLB* ou *Translation Lookaside Buffer*: uma memória cache para entradas da tabela de páginas. (Ver slides 21 a 25.)
- Mecanismo de conversão do endereço virtual para o endereço final, com segmentação e paginação. (Ver slides 34 e 35.)

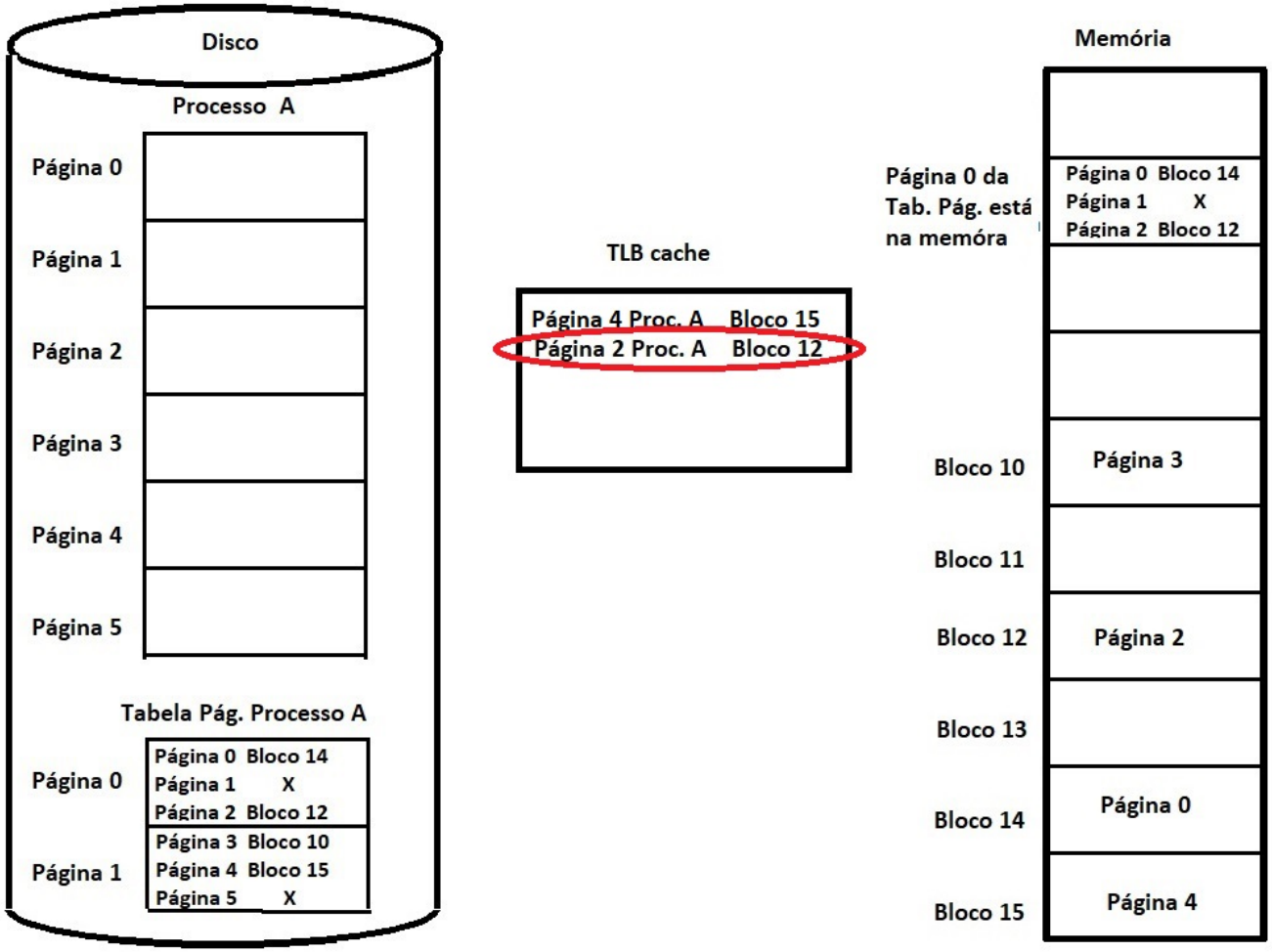
Como foi o meu aprendizado?

Vamos fazer um exercício/exemplo de paginação juntos. Dada uma página de um processo, queremos saber em que bloco ela está alocada.

Consideramos as seguintes situações:

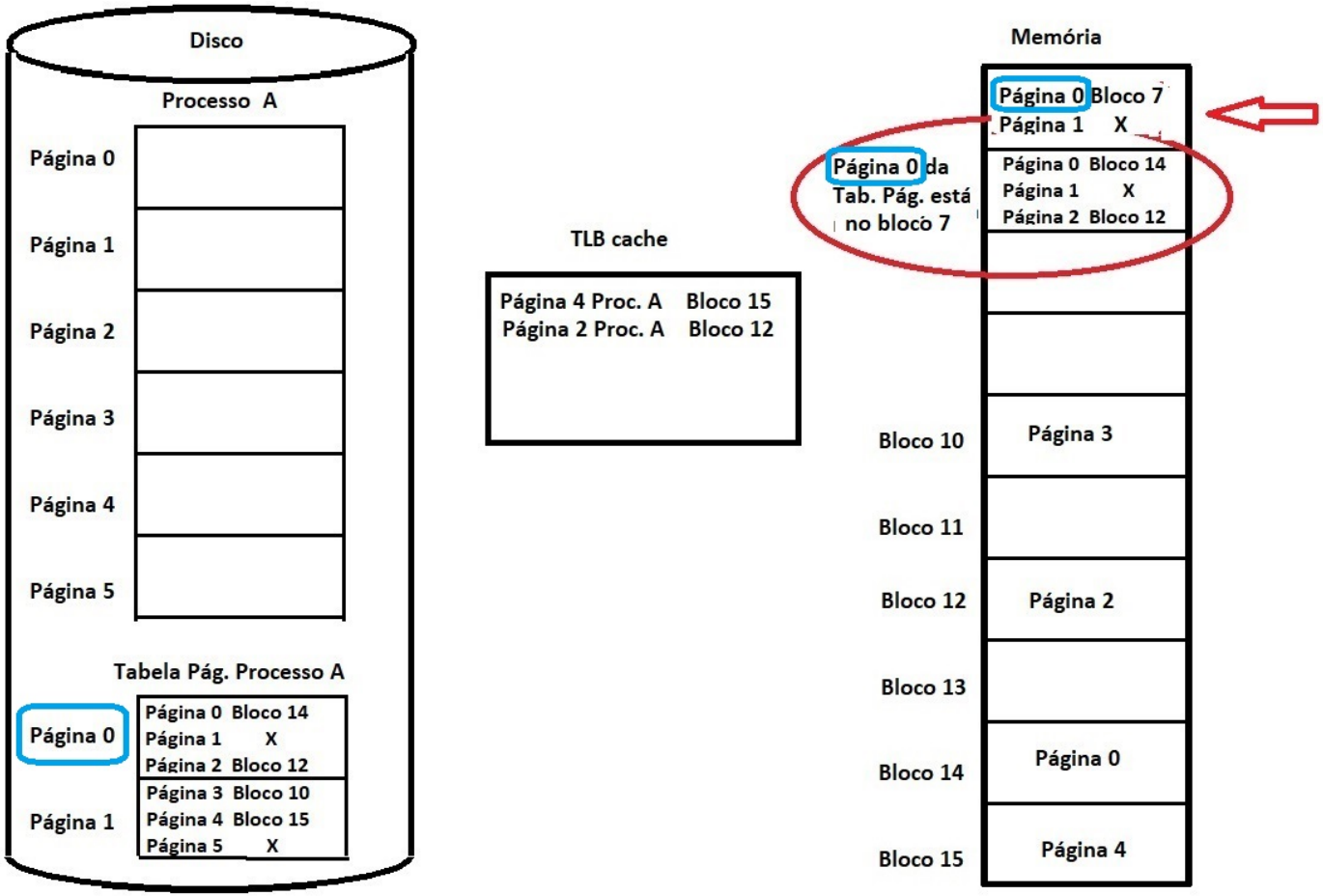
1. A página está na TLB cache.
2. A página não está na TLB. Mas consta na parte da Tabela de Página que está na memória.
3. A parte da Tabela de Páginas interessada não está na memória.
4. A parte da Tabela de Páginas interessada não está na memória. A página desejada não foi alocada ainda na memória.

1. A página está na TLB



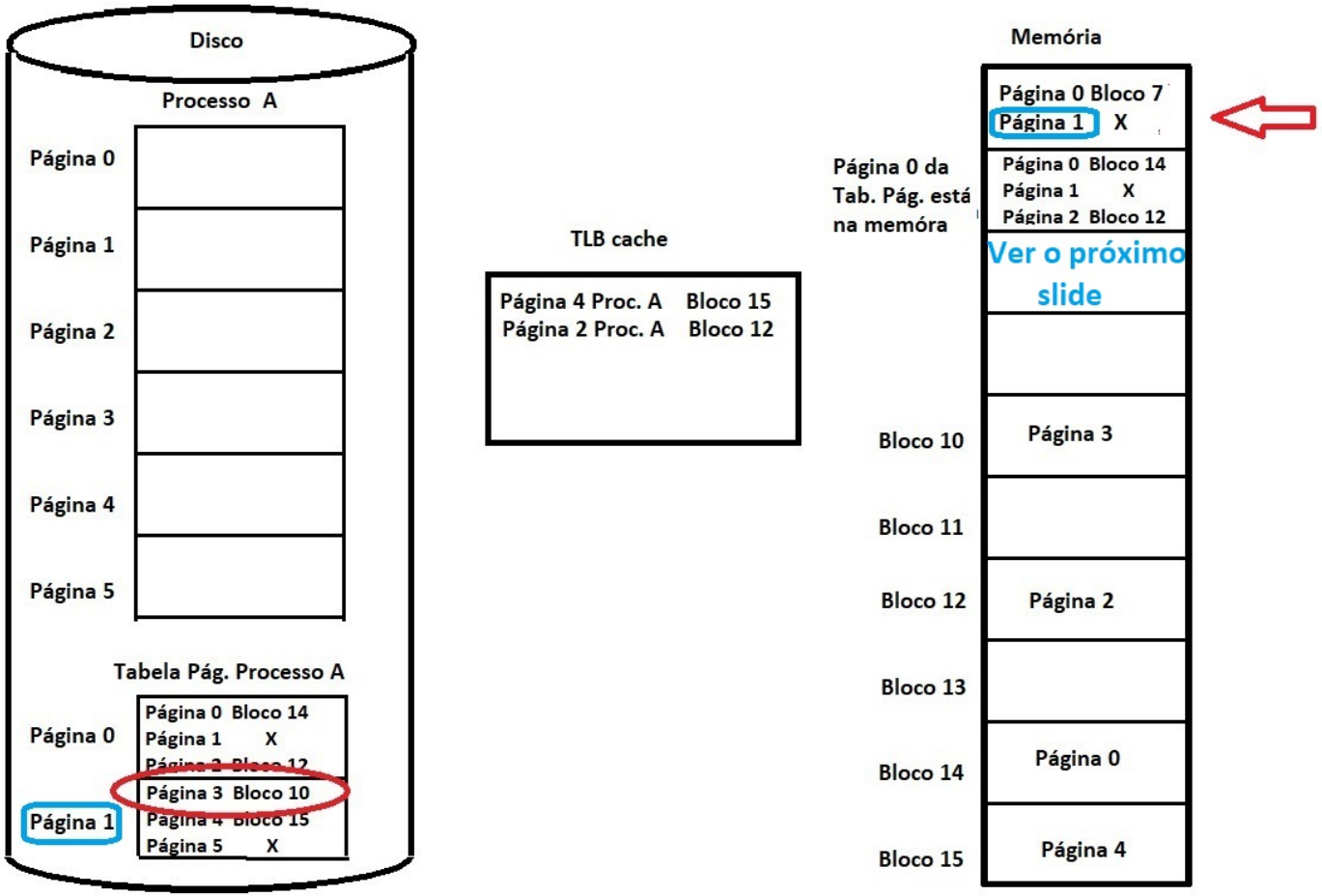
- Em que bloco está alocada a Página 2 do Processo A?
- A página desejada está na TLB, obtém-se imediatamente o bloco 12.

2. Parte da Tab. de Pág. interessada está na memória



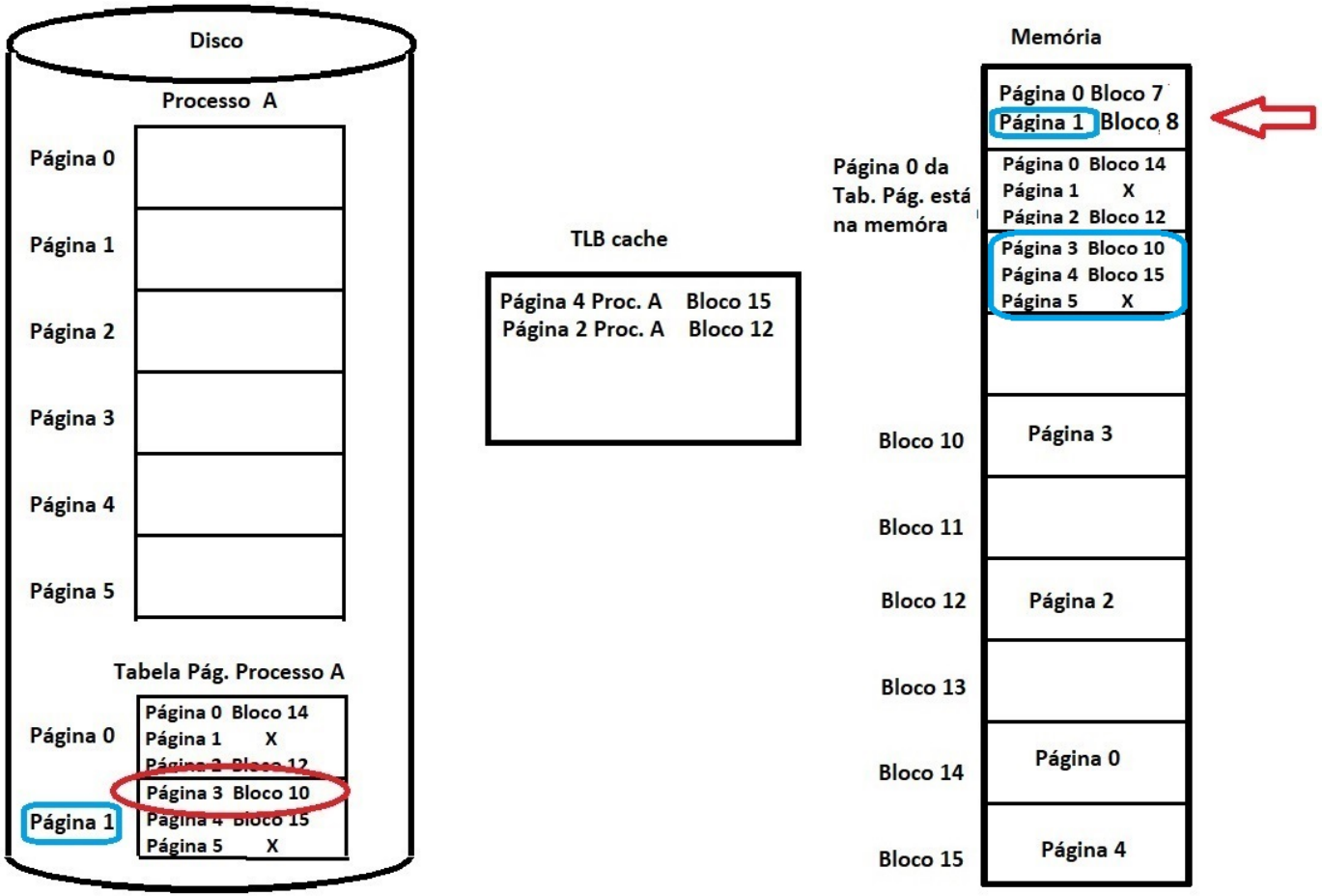
- Em que bloco está alocada a Página 0 do Processo A?
- A Página 0 do Proc. A está na Página 0 da Tabela de Páginas do Proc. A, que está na memória, no bloco 7. Localizamos assim o Bloco 14.
- Como sabemos isso? Pela Tabela de Páginas da Tabela de Páginas do Proc. A. Essa página é pequena e supomos que está sempre na memória. Veja a seta vermelha apontando para essa tabela.

3. Tab. de Pág. interessada não está na memória



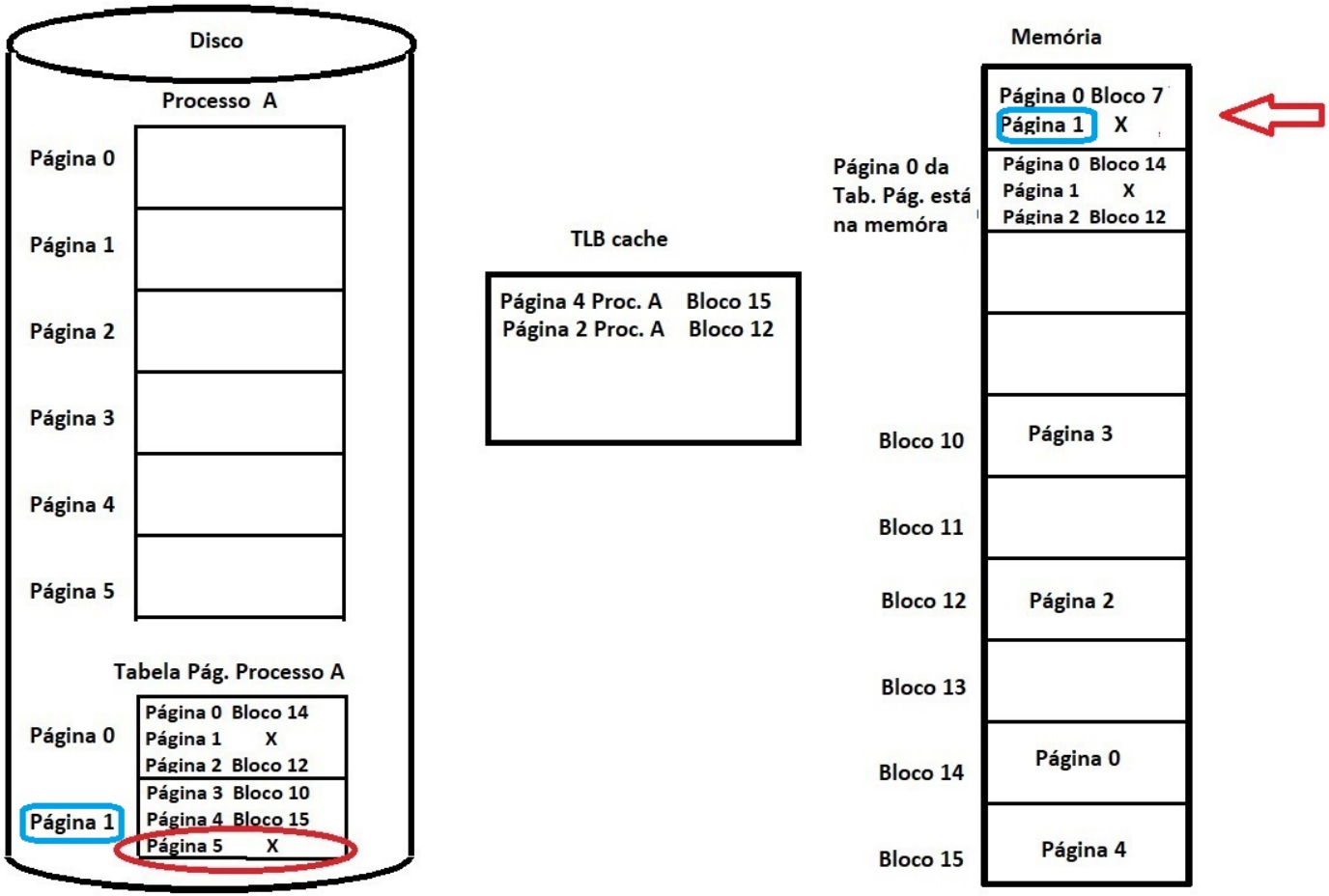
- Em que bloco está alocada a Página 3 do Processo A?
- A Página 3 do Proc. A está na Página 1 da Tabela de Páginas do Proc. A, que só está no disco, conforme indicada pela seta vermelha.
- Temos que trazer a Página 1 para a memória digamos para o bloco 8 e atualizamos a sua tabela de páginas. Localizamos o Bloco 10.

3. Tab. de Pág. interessada não está na memória



- Em que bloco está alocada a Página 3 do Processo A?
- A Página 3 do Proc. A está na Página 1 da Tabela de Páginas do Proc. A, que só está no disco, conforme indicada pela seta vermelha.
- Temos que trazer a Página 1 para a memória digamos para o bloco 8 e atualizamos a sua tabela de páginas. Localizamos o Bloco 10.

4. Tab. de Pág. interessada não está na memória



- Em que bloco está alocada a Página 5 do Processo A?
- A Página 5 do Proc. A está na Página 1 da Tabela de Páginas do Proc. A que, como já vimos no caso anterior, só está no disco.
- Repetimos o mesmo e agora descobrimos que a página 5 não está na memória. Precisamos achar um bloco livre para alocar a Página 5. Precisamos atualizar a Tabela de Páginas, a TLB etc.