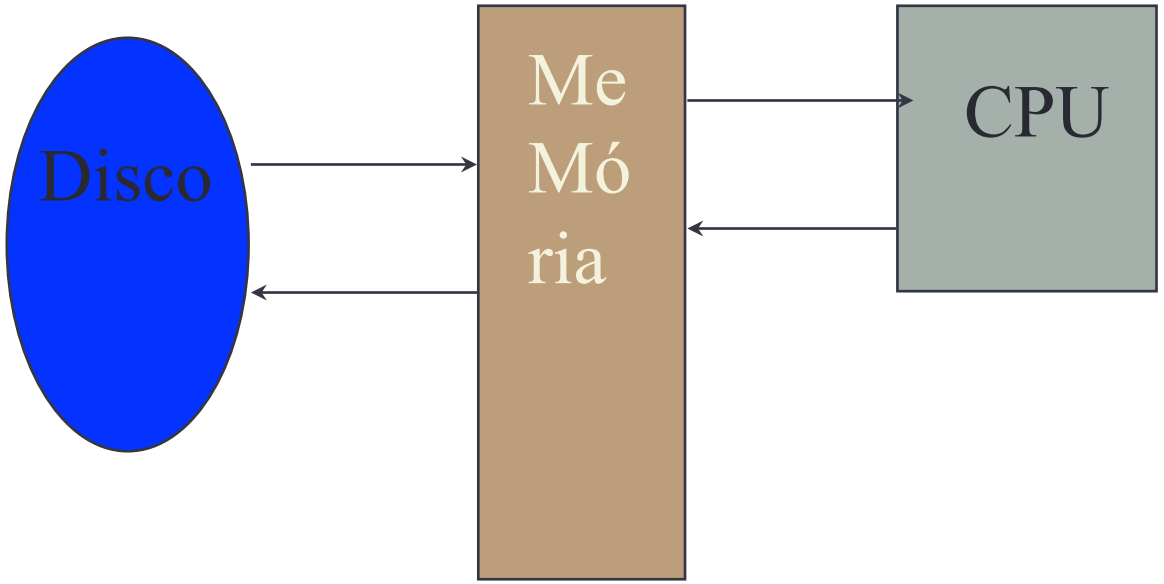
# TÓPICO 6

## Gerência de Memória

# Introdução

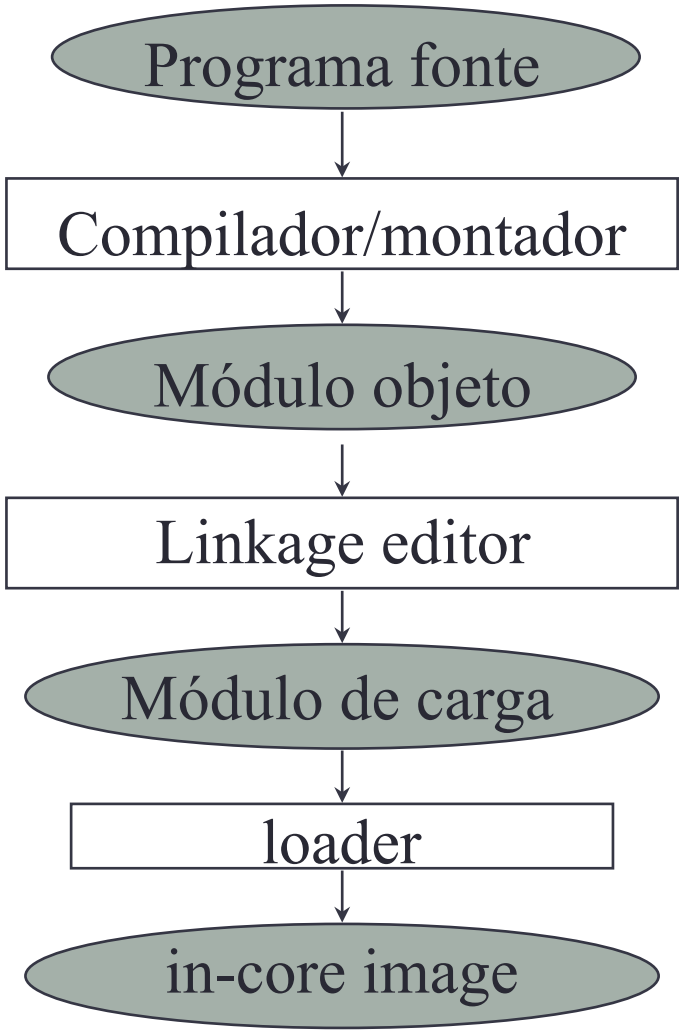
- Estrutura de armazenamento de um processo durante a execução:



# Introdução

- De uma maneira geral, na criação do processo, sua imagem (ou parte dela) é carregada em memória.
- A instrução e os dados que estão sendo manipulados em um determinado instante são carregados na CPU.
- Os dados alterados são escritos novamente em memória.
- Dados permanentes alterados (e.g. arquivos) são gravados em disco.

# Passos na geração de uma imagem de processo (in-core image):




endereços simbólicos (x, a)

endereços relocáveis (deslocamentos em relação ao início do módulo)

endereços absolutos (únicos para o programa como um todo)

# Funções do Gerente de Memória

- Principal função: Determinar o esquema de gerência de memória que será utilizado pelo sistema operacional e implementá-lo eficientemente.
- Funções secundárias, independentes do esquema de gerência de memória utilizado:
  - Controlar a alocação de pedaços da memória a processos
  - Liberar pedaços de memória que estavam alocados a processos e não são mais necessários.

# Esquemas de Gerenciamento de Memória

- Bare memory
- Monitor residente
- Multiprogramação
  - Partições fixas
  - Partições variáveis
  - Memória virtual
    - Paginação
    - Segmentação
    - Segmentação paginada

# Monoprogramação sem Paginação ou Swapping

- Em um ambiente monoprogramado, temos somente um processo executando de cada vez. Assim, podemos reservar toda a memória (ou grande parte dela) para este processo, já que é garantido que ele será o único processo de usuário ativo durante o seu período de execução.
- Podemos ter os seguintes esquemas de gerência de memória em tal ambiente:
  - bare machine
  - monitor residente

# Bare Machine

- Em uma *bare machine*, simplesmente não existe esquema de gerenciamento da memória. O usuário possui controle absoluto sobre todo o espaço de memória.

# Bare Machine

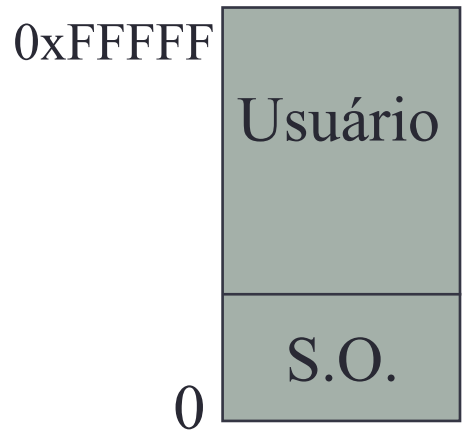
- *Hardware adicional:*Não é necessário nenhum hardware adicional
- *Vantagens:*
  - simplicidade de implementação (não existe implementação!)
  - total flexibilidade no uso da memória
- *Desvantagens:*
  - Não há sistema operacional: o usuário tem que implementar todos os serviços de baixo nível (acesso aos dispositivos, controle de interrupções, etc)

# Bare Machine

- *Exemplos de sistemas que utilizam este esquema:*
  - Sistemas dedicados (controle de processos, tempo real, etc)
  - Alguns sistemas paralelos permitem que o usuário opte por este esquema de gerência de memória

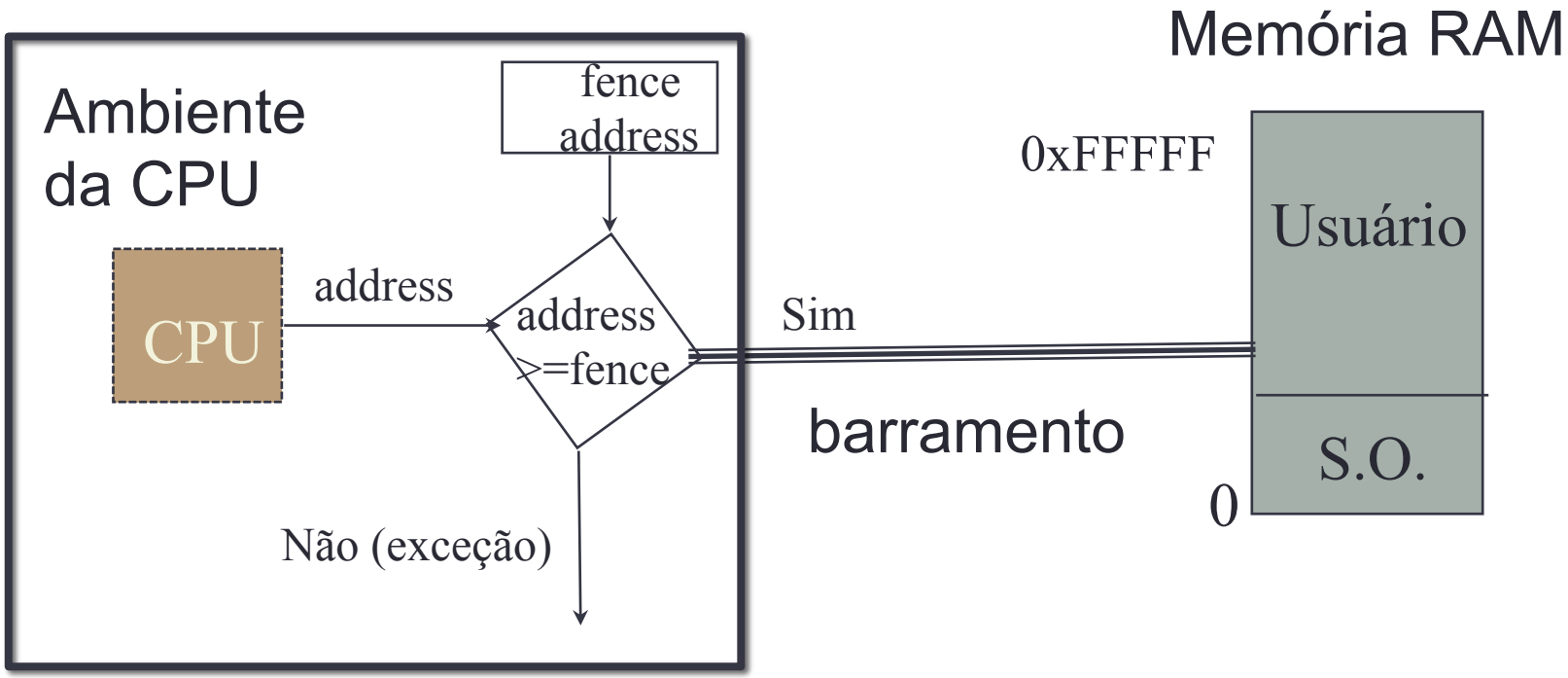
# Monitor Residente

- Em sistemas monoprogramados, o mais utilizado é dividir a memória em duas “áreas”: uma área para o usuário e outra para o sistema operacional.
- Neste esquema, somente um processo *de usuário* está ativo por vez. Além deste processo, o sistema operacional também está sempre ativo.



# Monitor Residente

- *Hardware adicional:* É necessário um dispositivo de hardware que impeça o processo de usuário de acessar a área do sistema operacional
- Esquema genérico:

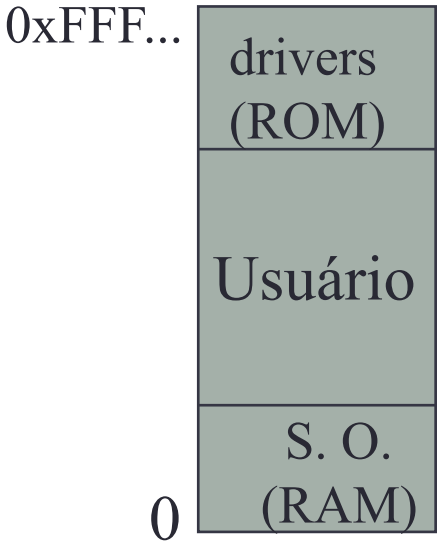


# Monitor Residente

- *Vantagens:*
  - Implementação simples: a rigor, só é necessário um mecanismo de proteção de hardware.
  - Os serviços do sistema operacional podem ser utilizados: drivers, interrupções, etc.
- *Desvantagens:*
  - Sistema mono-usuário (CPU sub-aproveitada).

# Monitor Residente

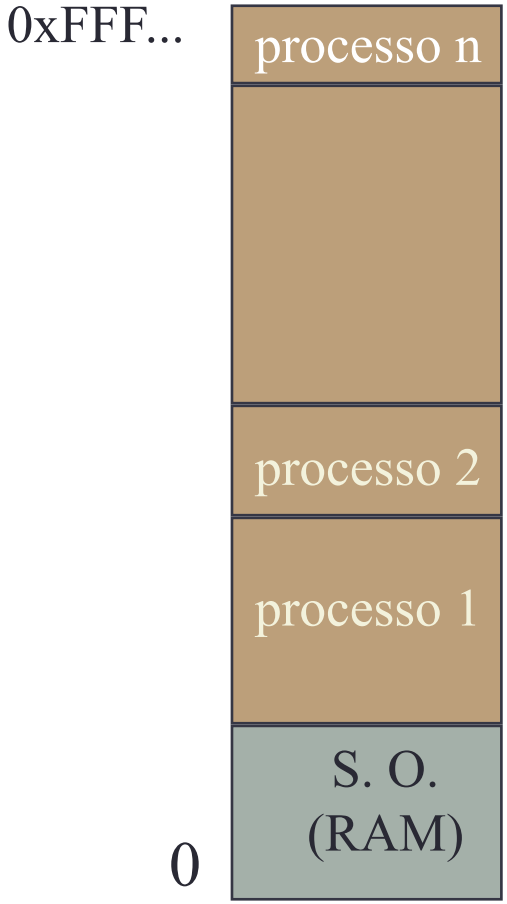
- *Exemplo:*DOS (IBM-PC antigos)



# Multiprogramação

- Atualmente, a maioria dos computadores oferece mecanismos que permitem a implementação da multiprogramação. No âmbito da gerência da memória, a questão que deve ser respondida é:
  - Como manter mais de um processo de usuário ativo em memória simultaneamente?
- Devemos sempre ter em mente que um processo deve ser protegido contra acessos de outros processos (maliciosos ou não).

# Multiprogramação

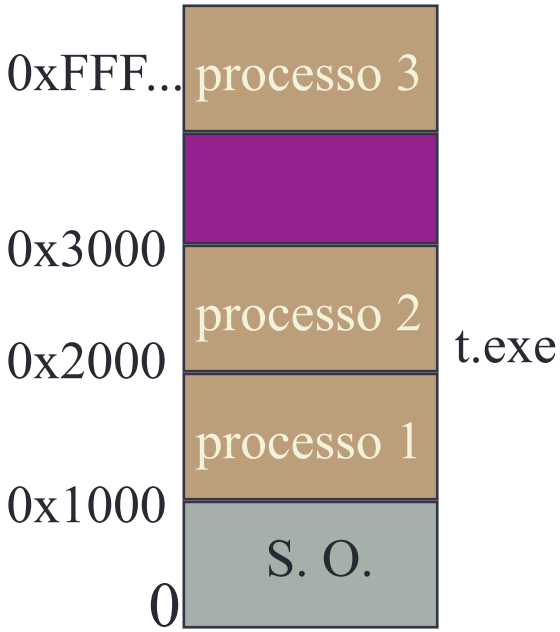


# Multiprogramação
# Partições Fixas

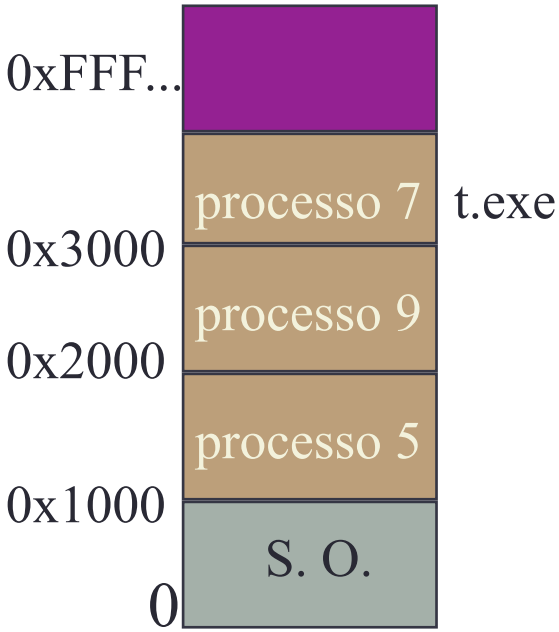
- A memória é dividida em um conjunto de regiões ou *partições* de tamanho fixo.
- Cada partição pode ter um programa executando.
- No momento de carga de um programa, uma partição livre é selecionada para executá-lo. O processo roda na mesma partição até terminar.
- Também conhecida como MFT (multiprogramming with a fixed number of tasks).

# Multiprogramação
# Partições Fixas

- *Problema:*Como arrumar os endereços do programa de maneira que eles referenciem a partição escolhida?




Primeira execução




Segunda execução

# Multiprogramação
# Partições Fixas

- A "rearrumação" de endereços é resolvida através de técnicas de *relocação*.
- Existem basicamente duas técnicas que permitem a relocação de endereços:
  - modificação de instruções e
  - registradores de base e limite.

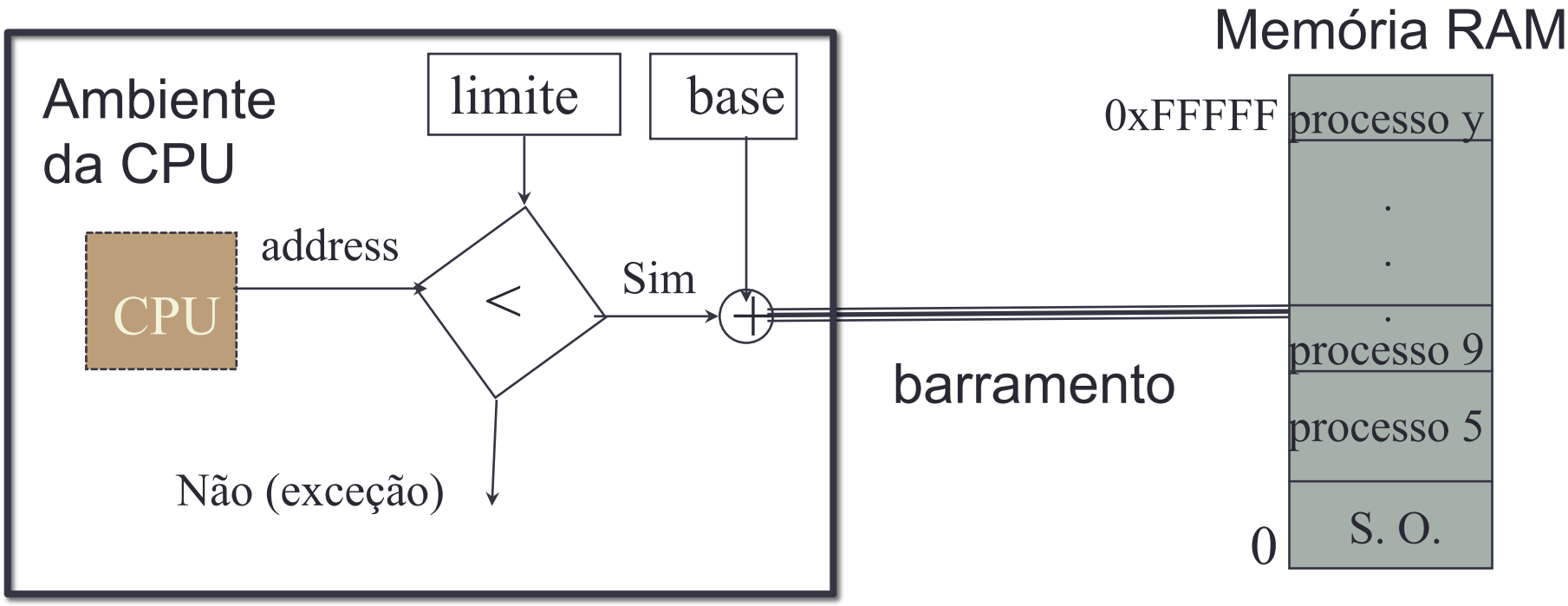
# Modificação de Instruções

- Consiste em modificar os endereços das instruções no momento da carga do programa para que estas referenciem o endereço de início da partição como base.
- No código binário, devem ser especificados que endereços são relocáveis (constantes e códigos de operação são invariantes).
- Não resolve o problema da proteção contra os outros processos ativos.

# Registradores de Base e Limite

- Os endereços gerados no código binário executável são de 0 a x.
- Cada endereço do processo usado para acessar a memória é adicionado do valor contido no registrador de base.
- Assim, obtemos o endereço físico onde o dado reside.

# Registradores de Base e Limite




- O registrador de limite implementa a proteção de memória. O registrador de base implementa a relocação.

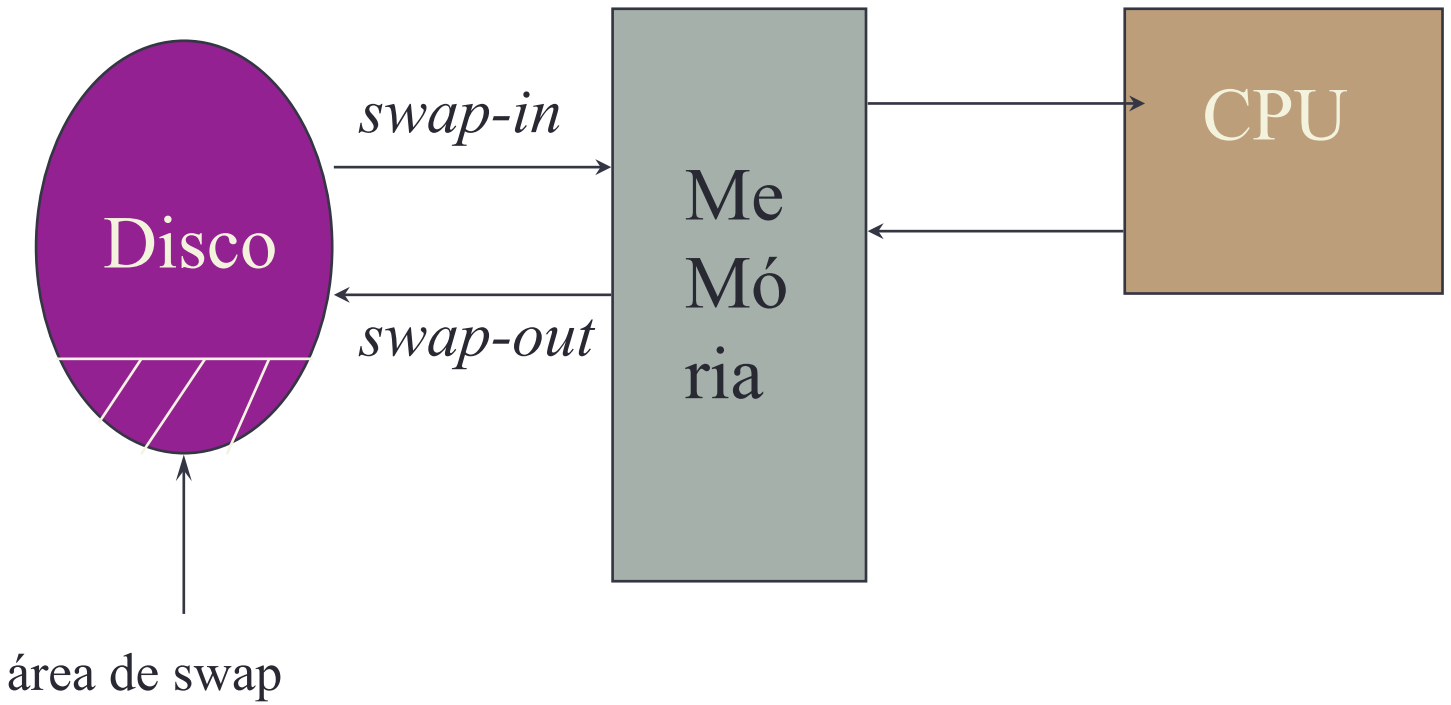
# Swapping

- Geralmente, a memória disponível no computador não é capaz de armazenar todos os processos ativos em um dado instante. Como não há espaço físico na memória, devemos colocar alguns processos ativos em disco.
- O movimento de processos ativos da memória para o disco e vice-versa chama-se **swapping**.

# Swapping

- O movimento de processos ativos da memória para o disco chama-se *swap-out*.
- O movimento de processos ativos do disco para a memória chama-se *swap-in*.
- Para as operações de swapping, existe uma área reservada no disco. Esta área chama-se área de swap.

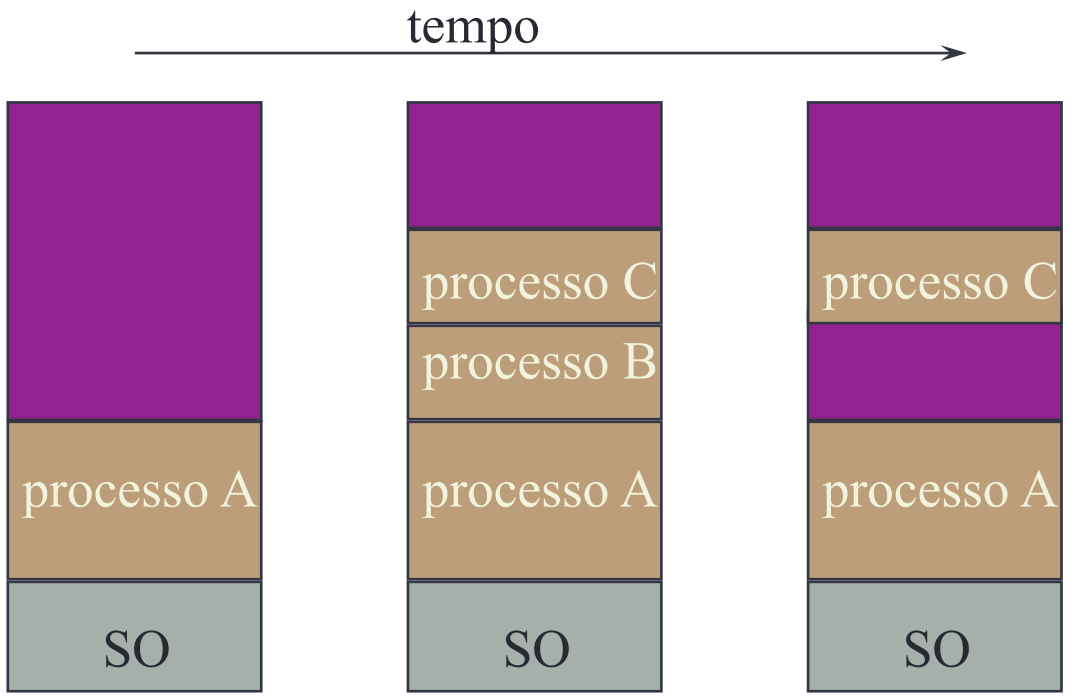
# Swapping



# Partições de Tamanho Variável

- O esquema conhecido como multiprogramação com partições de tamanho variável ou MVT (multiprogramming with a variable number of tasks), foi introduzido para tentar resolver o problema de desperdício de memória do MFT.
- A memória é dividida em um conjunto de regiões ou *partições* de tamanho variável.
- O número, o tamanho e a localização dos processos em memória varia com o tempo.

# Partições de Tamanho Variável



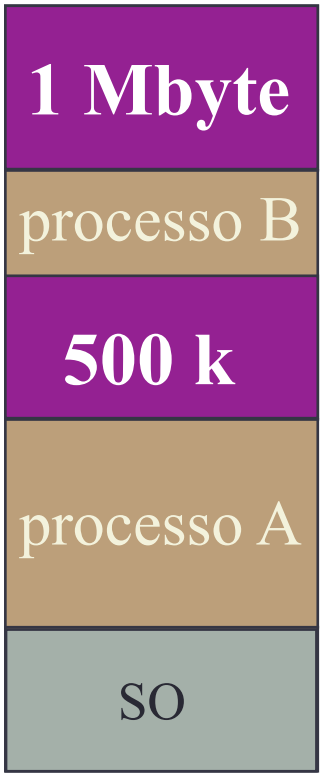

**Como tratar o problema dos buracos????**

# Partições de Tamanho Variável

- Com a alocação da memória em pedaços de tamanho variável, ficamos de face a um problema de alocação dinâmica de espaço. No nosso caso específico, temos um problema de alocação dinâmica de espaço em memória.
- Para se resolver um problema de alocação dinâmica de espaço, devemos satisfazer uma solicitação de tamanho *n*, dada uma lista de buracos livres.

# Partições de Tamanho Variável

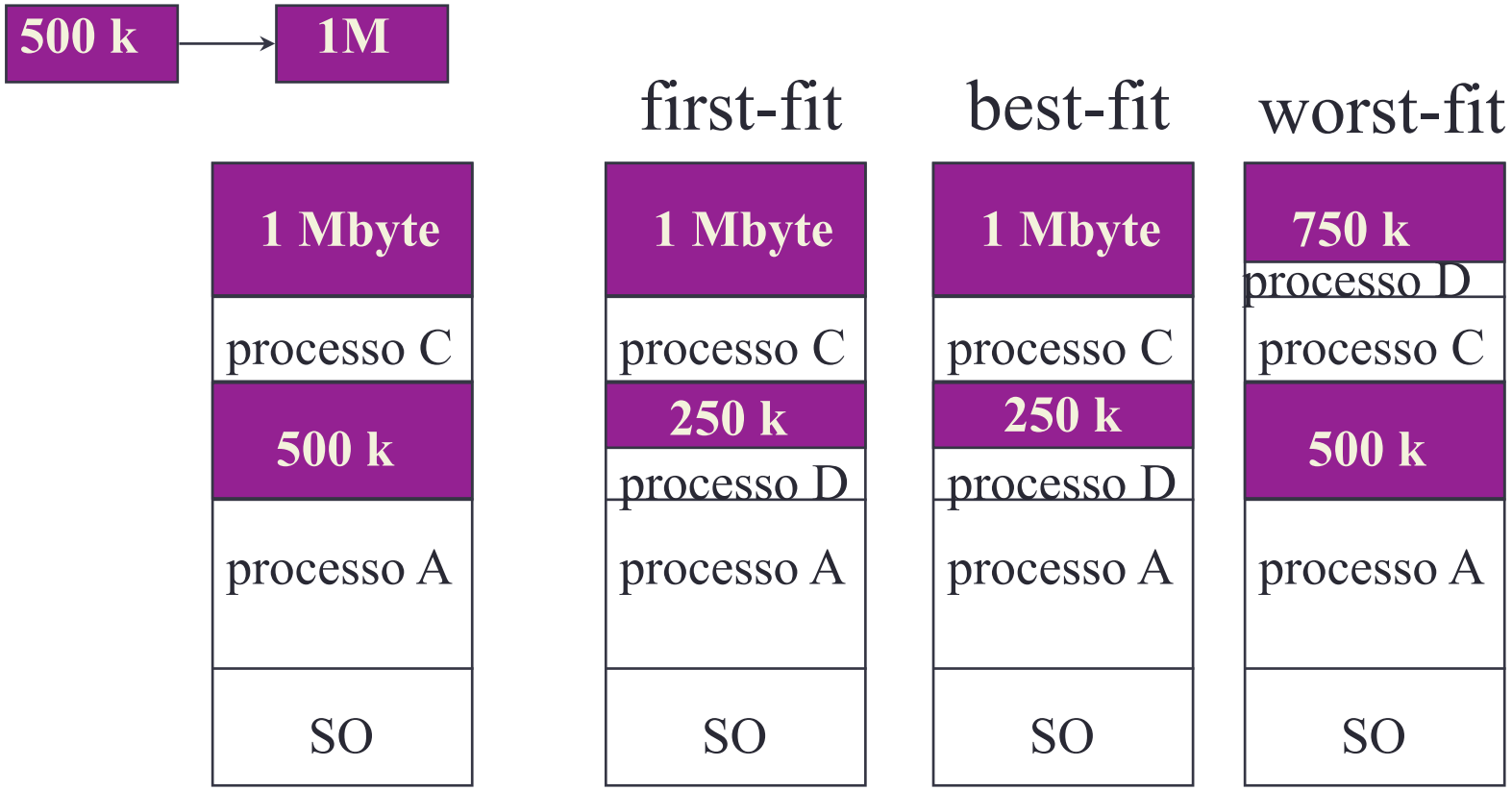
***Onde colocar o processo D, de 250 kbytes?***



# Partições de Tamanho Variável

- Existem 3 estratégias para decidir qual o melhor buraco:
  - *First fit* - Aloca o primeiro buraco que é grande o bastante.
  - *Best fit* - Aloca o menor buraco que é grande o bastante.
  - *Worst fit* - Aloca o maior buraco que é grande o bastante.
  - Simulações mostraram que *first-fit* e *best-fit* apresentam melhores resultados em termos de utilização de espaço. Como estas duas estratégias apresentam desempenho similar, geralmente escolhemos o *first-fit*, pois seu tempo de execução é menor.

# Algoritmos de Alocação de Espaço



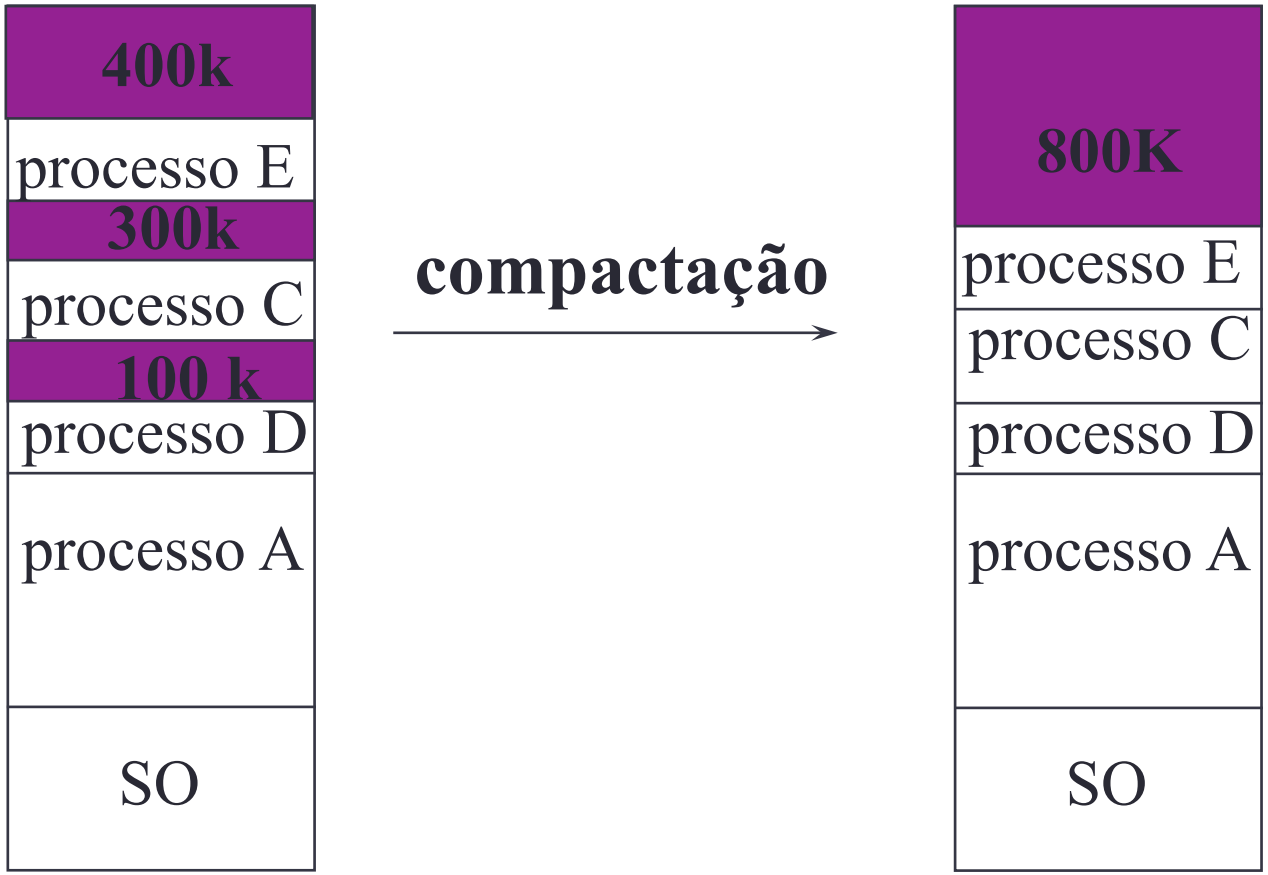
# Fragmentação Externa

- A multiprogramação com partições de tamanho variável sofre de fragmentação externa.
- Existe fragmentação externa quando existe espaço suficiente para a alocação, mas este espaço não é contíguo. A memória é fragmentada em um grande número de pequenos buracos.

# Compactação de Memória

- Há momentos em que a fragmentação externa atinge índices intoleráveis, onde pouco trabalho útil pode ser feito. Nestes momentos, devemos utilizar alguma técnica de compactação de memória.
- A compactação de memória é feita através do movimento de processos na memória, de modo que, ao final da compactação, estejamos de posse de um único e grande buraco.
- A desvantagem da compactação de memória é o grande tempo gasto com o movimento de processos na mémoria. Durante esta operação, nenhum processo se executa.

# Compactação de Memória



# Alocação de Espaço para um Processo

- Resolver um problema de alocação de espaço para um processo consiste em determinar a quantidade de memória que devemos reservar para um processo.
- Em sistemas onde um processo tem tamanho fixo, alocamos a quantidade de memória exatamente igual ao tamanho do código binário do processo.

# Alocação de Espaço para um Processo

- Infelizmente, a maioria dos sistemas permite que o tamanho da área de dados de um processo cresça ao longo da execução.
- Neste caso, é interessante alocar um espaço de memória maior que o tamanho inicial do processo, de forma a evitar o movimento do processo no caso do crescimento de sua área de dados.

# Controle de Alocação de Memória

- Existem basicamente 2 maneiras de implementar o controle da alocação de memória:
  - mapa de bits
  - lista encadeada

# Controle de Alocação de Memória - Mapa de Bits

- Neste esquema, a memória é dividida em unidades de alocação (uas) de tamanho fixo.
- A cada unidade de alocação corresponde um bit no mapa de bits. Se a unidade de alocação estiver livre, o valor do bit é 0. Se estiver alocada, o valor do bit é 1.

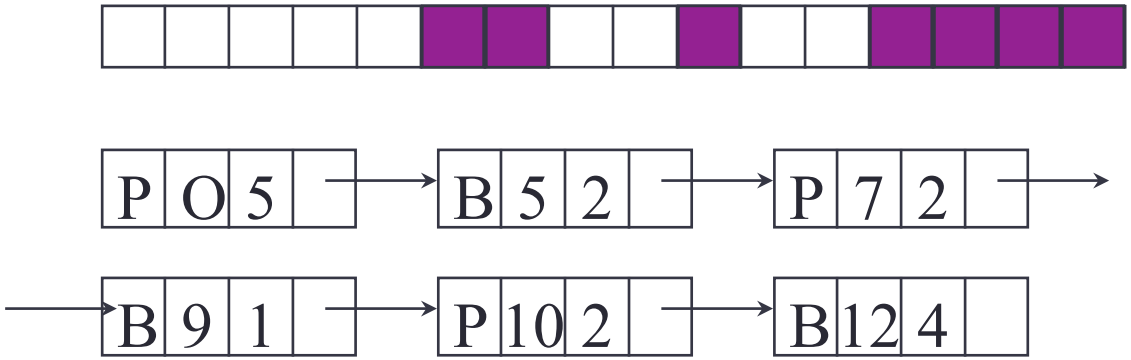

Mapa de bits associado: 1111100110110000

# Controle de Alocação de Memória - Mapa de Bits

- O tamanho do mapa de bits depende do tamanho da memória e do tamanho da unidade de alocação
  - Unidade de alocação grande: desperdício no uso da memória.
  - Unidade de alocação pequena: mapas de bits grandes
- Este esquema é raramente utilizado pois a busca por buracos livres em um mapa de bits pode ser extremamente lenta

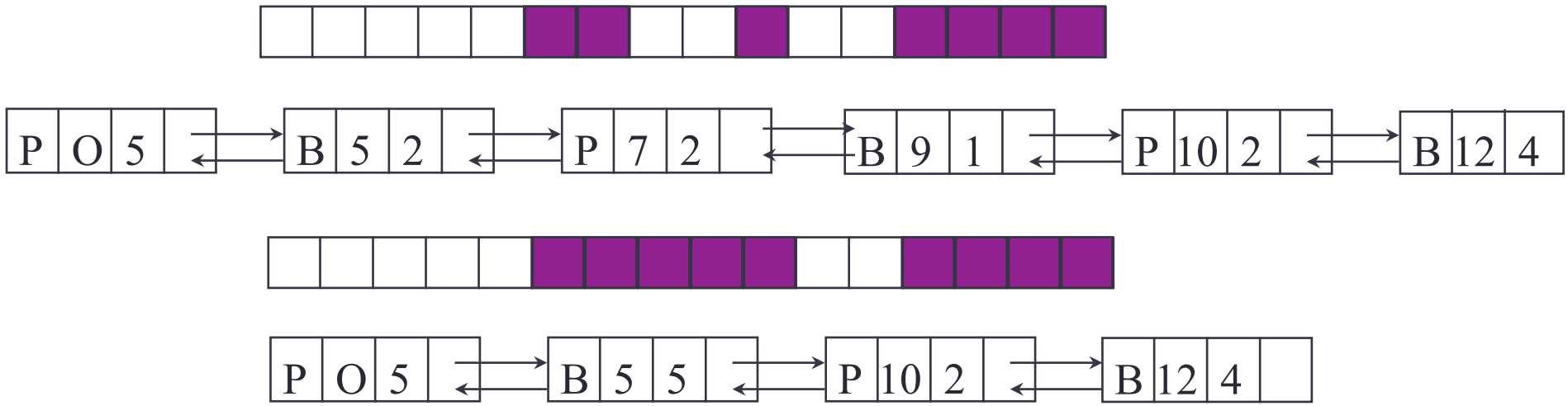
# Controle de Alocação de Memória - Lista Encadeada

- Neste esquema, representamos os segmentos livres e ocupados de memória através de uma lista encadeada.
- Cada entrada contem o tipo do segmento (buraco ou processo), o endereço de início e um ponteiro para o próximo elemento.



# Controle de Alocação de Memória - Lista Encadeada

- Existem variações deste esquema que implementam listas duplamente encadeadas. Esta variação é importante quando necessitamos fazer um merge de buracos (e.g., no caso de término de processo).



# Controle de Alocação de Memória - Lista Encadeada

- Outra variação consiste em manter duas listas distintas: uma lista por buraco e outra lista por processo. Neste caso, a lista de buracos pode ser organizada pelo tamanho do buraco, para que a busca de buracos seja mais rápida. O problema desta abordagem é o grande overhead necessário para manter a lista classificada. Este overhead é pago no momento da liberação de espaço.

☛Para o controle da alocação de memória, são utilizadas geralmente as listas encadeadas, em uma de suas diversas variações

# Memória Virtual
# Introdução

- Até agora, estávamos tratando o problema de manter vários processos em memória e de controlar a alocação de memória para estes processos.
- No final dos anos 60, surgiu um problema diferente que deveria ser resolvido: Como executar um processo que não cabe inteiramente na memória? Neste caso, o tamanho do processo é maior que o tamanho da memória disponível.

# Memória Virtual
# Introdução

- Para solucionar este problema, devemos “quebrar” o processo em unidades menores e carregar estas unidades em memória à medida em que elas são necessárias.
- A questão principal é: quem vai quebrar o processo em pequenas unidades?
  - O próprio programador -> overlay
  - O sistema operacional -> memória virtual

# Overlay

- O programador divide o seu programa em módulos menores, chamados módulos de overlay.
- O sistema operacional é o responsável pela carga e liberação de módulos da memória.
- A tarefa de determinar o número e o tamanho dos módulos de overlay é uma tarefa complexa e geralmente o programa era executado de maneira extremamente ineficiente.

# Memória Virtual

- Com a memória virtual, a tarefa de definição dos módulos passa a ser responsabilidade do sistema operacional.
- A memória virtual é um esquema de gerenciamento de memória proposto em 1961 e é o método que encontramos na grande maioria dos computadores e sistemas operacionais atuais.
- Com a memória virtual, o processo não referencia mais endereços físicos e sim endereços virtuais.

# Memória Virtual

- Os endereços virtuais dependem somente da capacidade de endereçamento da máquina e do suporte do sistema operacional.
- Por exemplo, em uma máquina com endereços de 32 bits, um processo pode endereçar de 0 a $2^{32}$ - 1 endereços virtuais, independente do tamanho da memória física.
- Na execução de cada instrução, os endereços virtuais devem ser traduzidos para endereços físicos.

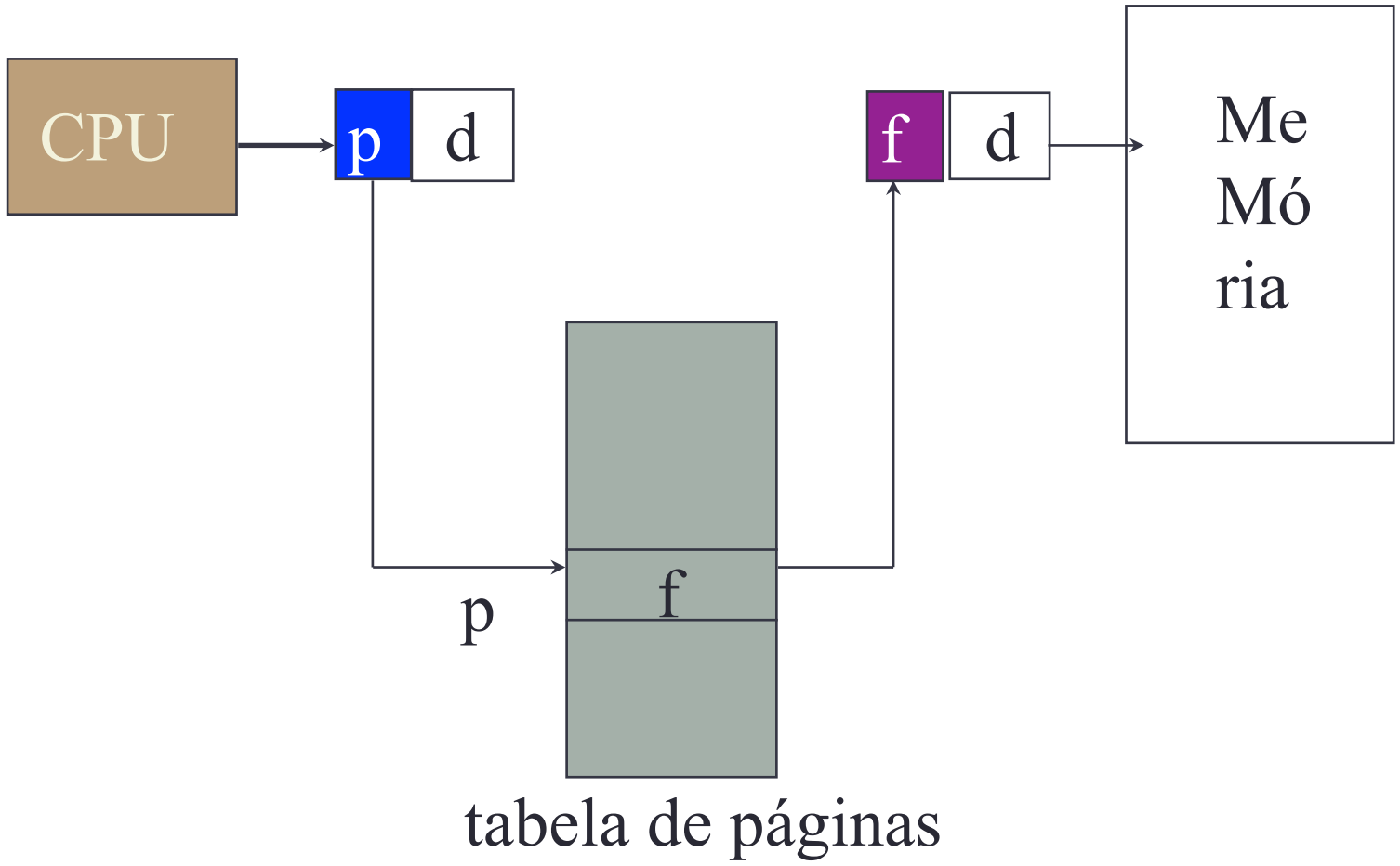
# Memória Virtual

- A memória virtual é um mecanismo genérico que pode ser utilizado tanto em sistemas monoprogramados como em sistemas multiprogramados.
- Existem basicamente dois mecanismos que implementam a memória virtual:
  - paginação: divide a memória física e a memória virtual em unidades de tamanho fixo
  - segmentação: divide a memória física e a memória virtual em unidades de tamanho variável

# Memória Virtual
# Paginação

- Na paginação, a CPU gera uma solicitação para o endereço virtual x.
- Este endereço é convertido para o endereço físico através de uma tabela de páginas, que contem o mapeamento do endereço virtual para o endereço físico.
- O endereço físico obtido assim é colocado no barramento da memória.

# Memória Virtual
# Paginação

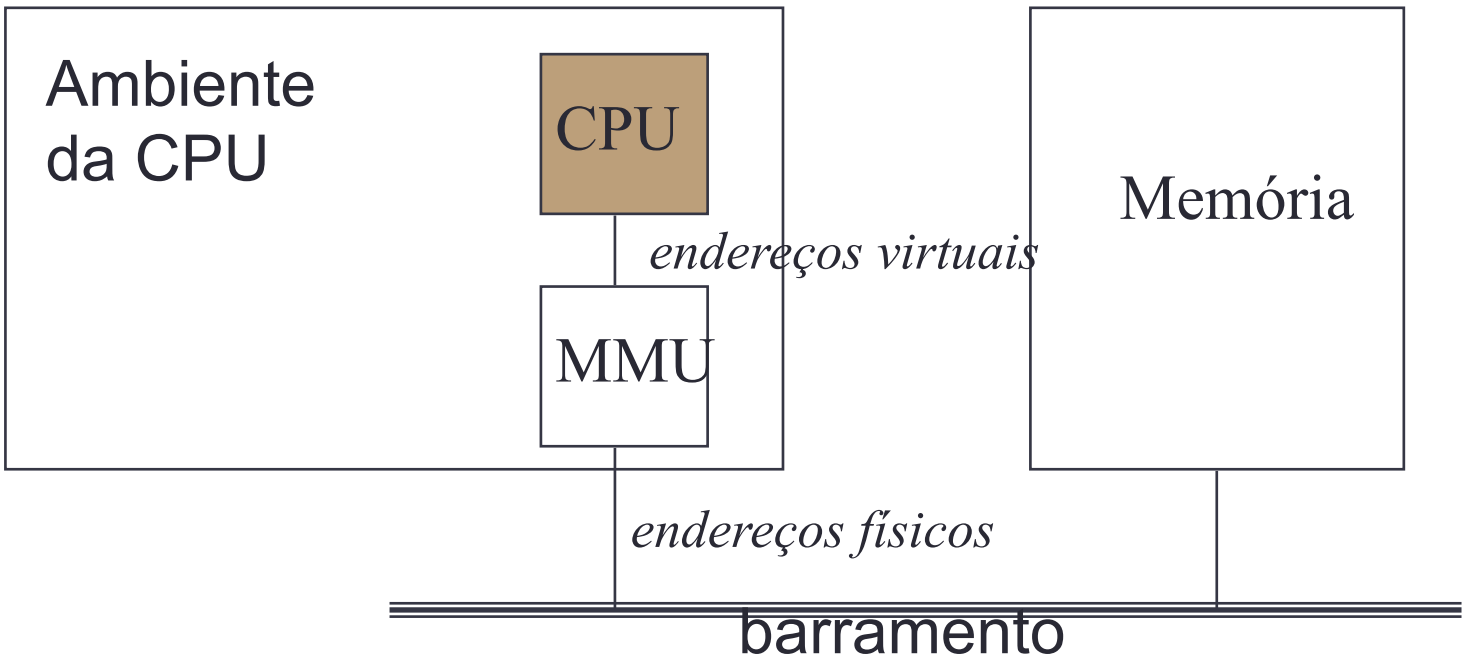


# Memória Virtual
# Paginação

- Na paginação, para cada acesso à memória gerado, necessitamos receber o endereço virtual, acessar uma tabela com o endereço virtual, recuperar o endereço físico e colocar o endereço físico no barramento.
- Se estas operações fossem feitas por software, o esquema de paginação estaria inviabilizado, devido ao alto overhead por acesso.

# Paginação

- *Hardware adicional*: O computador deve ser equipado de um dispositivo complexo de hardware que será o responsável pelo mapeamento de endereços. O nome deste dispositivo é MMU (memory mamagement unit).



# Paginação - Funcionamento

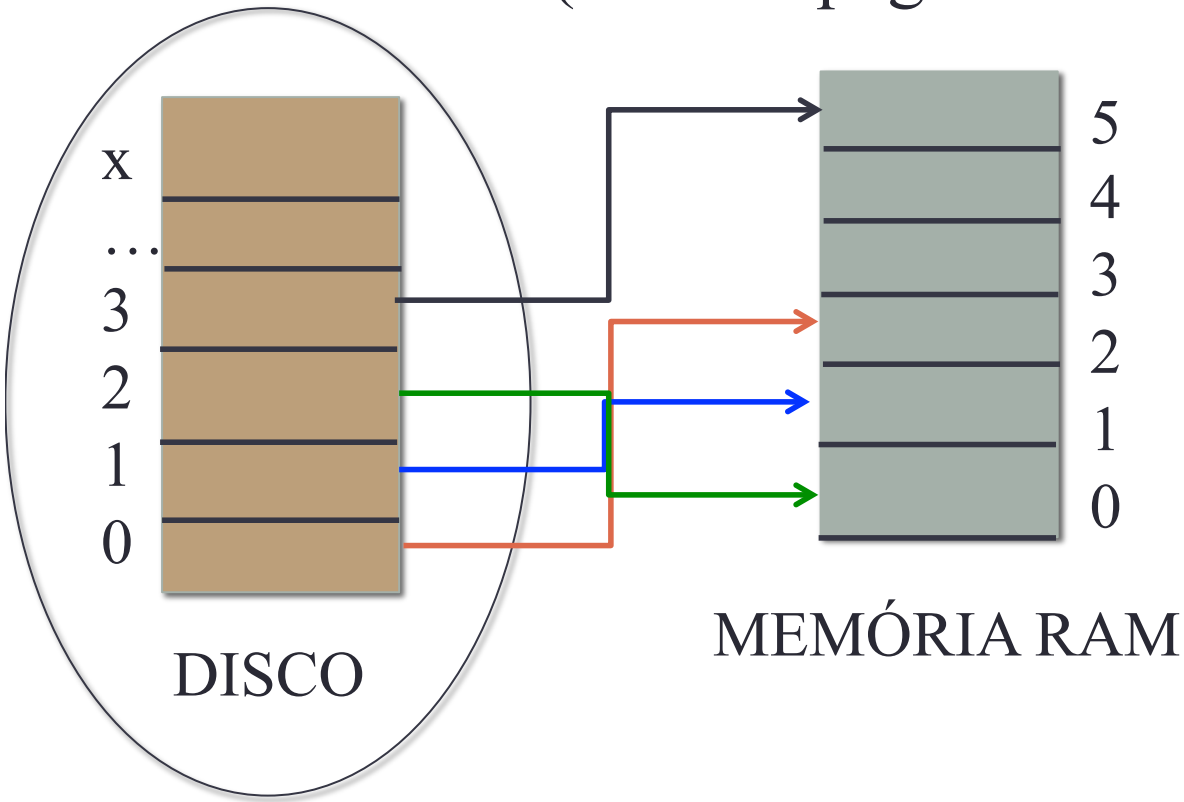
- O endereço virtual é dividido em unidades de tamanho fixo chamadas páginas.
- A memória física é dividida em unidades de tamanho igual ao das páginas, chamadas page frames. As page frames vão ser as “casas” das páginas enquanto estas estiverem em memória.

# Paginação - Funcionamento

- Passos no acesso à memória:
  - A CPU envia o endereço virtual (*v*) à MMU.
  - Na MMU, o endereço virtual é dividido em (*p*,*d*) onde *p* é a página e *d* é o deslocamento dentro da página. A MMU utiliza a página *p* para acessar a tabela de páginas e recuperar a frame *f* na qual a página *p* reside.
  - A MMU substitui *p* por *f* e coloca o endereço *(f,d)* no barramento.

# Paginação - Funcionamento

- Tendo a tabela de páginas abaixo e recebendo o endereço 2253, que endereço a MMU vai colocar no barramento? (Admita páginas de 1Kbyte).

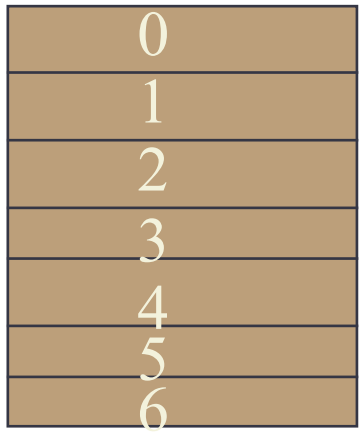



*Tab. Páginas*

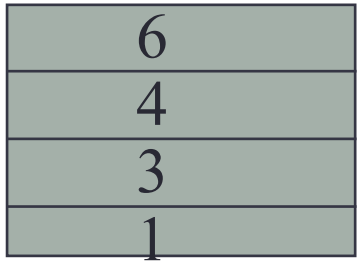
| | |
|---|---|
| 0 | 2 |
| 1 | 1 |
| 2 | 0 |
| 3 | 5 |

# Paginação - Page Fault

- O espaço de endereçamento virtual é, em geral, bem maior que a memória física disponível. Assim, existirão geralmente páginas que não estão mapeadas em memória.



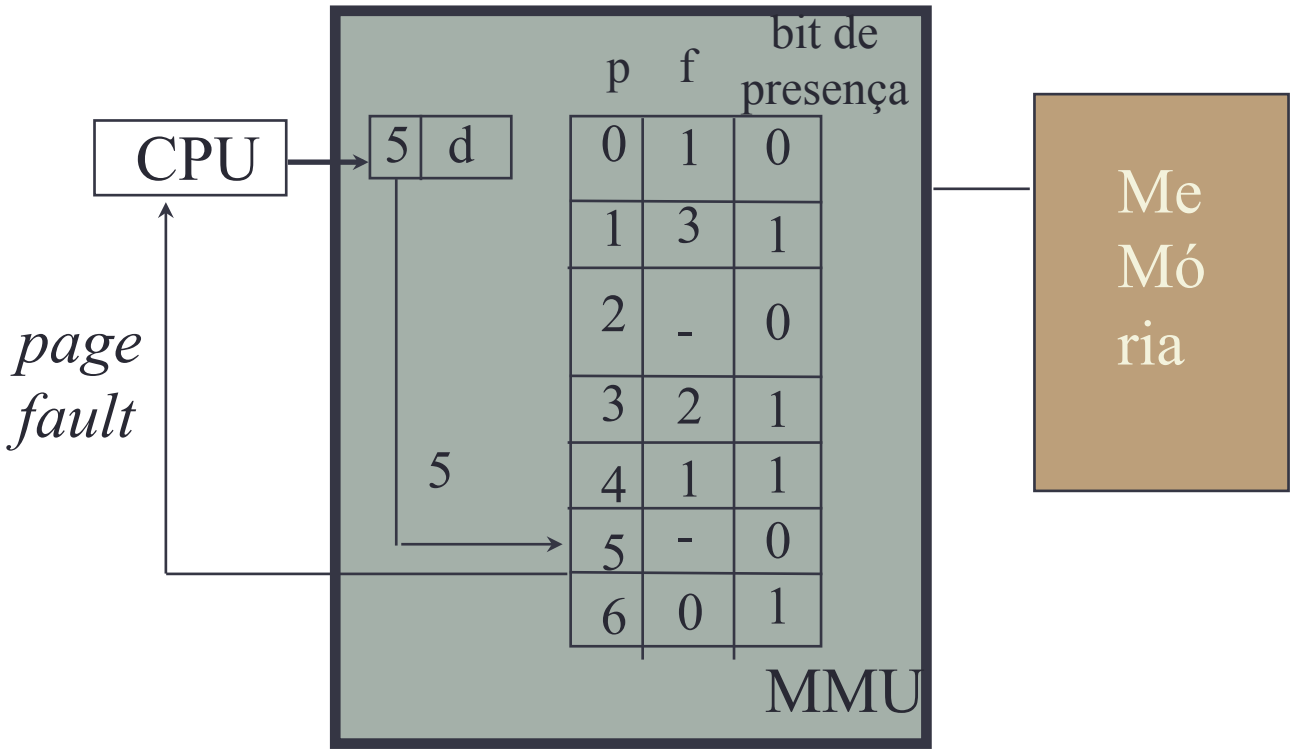

*Processo*




*Memória*

# Paginação - Page Fault

- O que fazer se o endereço gerado pela CPU for referente a uma página não mapeada?



# Paginação - Page Fault

- A MMU detecta o não mapeamento através do bit de presença, contido na tabela de páginas.
- A MMU força um trap para o sistema operacional. O nome deste trap é falta de página (page fault).
- O SO escolhe uma página que deve desocupar a memória (se a memória estiver cheia) e carrega a página referenciada no lugar desta página.
- A tabela de páginas é alterada.
- A instrução que causou o page fault é reexecutada.

# Tabela de Páginas

- A tabela de páginas é a estrutura de dados que guarda informações necessárias ao mapeamento do endereço virtual de um processo em endereço físico.
- Na sua organização mais tradicional (forward mapped) nós temos uma tabela de páginas para cada processo.

# Tabela de Páginas

- Apesar das informações contidas na tabela de páginas dependerem da máquina, existem alguns campos que estão presentes em quase todas as tabelas de páginas. São eles:
  - *bit de cache* - indica se a página pode ser colocada em cache
  - *bit de referência* - indica se a página foi referenciada
  - *bit de modificação* - indica se a página foi alterada
  - *bit de proteção* - indica a proteção associada à página
  - *bit de presente/ausente* - indica a presença da página em memória física
  - *número da page frame* - número da frame onde a página está mapeada.

# Tabela de Páginas

- Quanto à tabela de páginas, devemos observar os seguintes aspectos:
  - tamanho da tabela de páginas: com espaços de endereçamento cada vez maiores (32 bits, 64 bits), a tabela de páginas pode assumir tamanhos assustadores.
    - Exemplo: com o espaço de endereçamento de 32 bits e páginas de 4 kbytes, teremos 1 Mega páginas por processo, ou seja, 1 Mega entradas na tabela de páginas. Admitindo que cada entrada possui 6 bytes, precisaremos de 6 Mbytes para armazenar a tabela de páginas.

# Tabela de Páginas

- o mapeamento de endereços deve ser extremamente rápido, pois precisa ser feito em todas as referências à memória.

- Além disso, devemos lembrar que a tabela de páginas (ou parte dela) deve residir na MMU.

# Localização da Tabela de Páginas

- Quanto à localização da tabela de páginas, podemos utilizar as seguintes abordagens:
  - Armazenar a tabela de páginas totalmente na MMU.
  - Armazenar parte da tabela de páginas na MMU e parte em memória.

# Tabela de Páginas Localizada na MMU

- A tabela de página é armazenada em um conjunto de registradores especiais.
- Cada registrador contem uma entrada da tabela.
- Cada vez que há uma troca de contexto, a tabela de páginas do processo é carregada no conjunto de registradores.

# Tabela de Páginas Localizada na MMU

- *Vantagens*:
  - implementação simples,
  - mapeamento extremamente eficiente.
- *Desvantagens:*
  - inviável para tabelas de páginas grandes (é o caso da grande maioria dos sistemas modernos),
  - aumento do overhead da troca de contexto

# Tabela de Páginas Localizada na Memória e na MMU

- Devido ao tamanho da tabela de páginas,  é inviável mantê-la totalmente na MMU.
- A tabela de página é armazenada na memória principal e um registrador contem o endereço de início da tabela de páginas.
- A MMU possui uma memória associativa chamada *Translation Look-aside Buffer*, onde algumas entradas da tabela de páginas são armazenadas.

# Tabela de Páginas Localizada na Memória e na MMU

- Quando um endereço virtual é apresentado à MMU, ela procura primeiro no TLB.
- Se estiver presente, o mapeamento é imediato.
- Senão, um acesso à memória é feito para recuperar a entrada da tabela de páginas.
- A entrada referenciada é colocada no TLB.
- As próximas referências a esta mesma página serão imediatas.

# Tabela de Páginas Localizada na Memória e na MMU

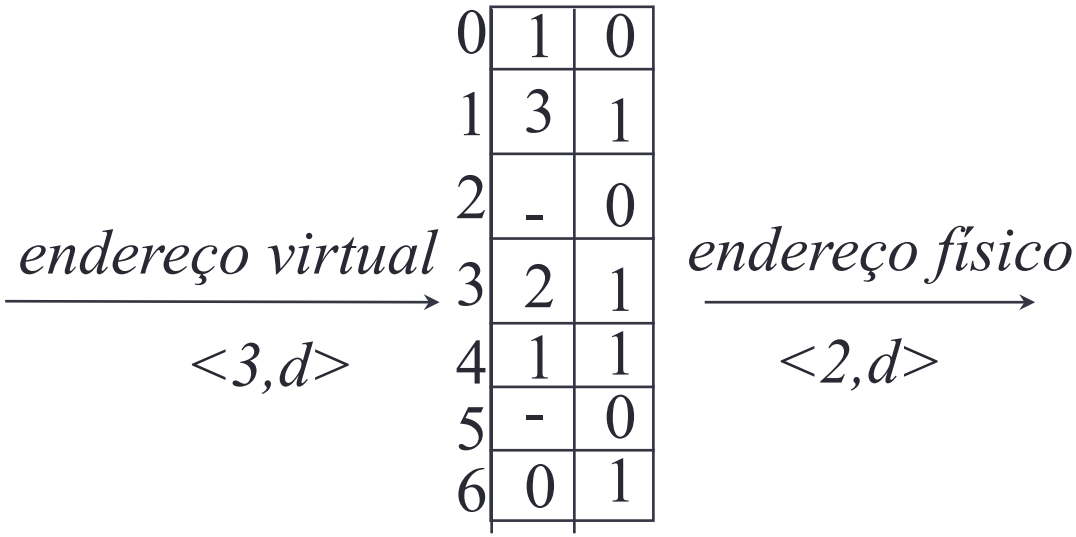
- *Vantagens*:
  - mapeamento eficiente
  - custo relativamente baixo
- *Desvantagens*:
  - alguns acessos à memória necessitarão um acesso adicional à tabela de páginas em memória.

# Organização da Tabela de Páginas

- Quanto à organização, a tabela de páginas pode ser:
  - forward-mapped (mapeamento direto)
  - invertida

# Tabela de Páginas Forward-Mapped

- A tabela de páginas é um vetor de entradas, armazenada em memória.
- Existe uma entrada para cada página e a frame é obtida através desta entrada.



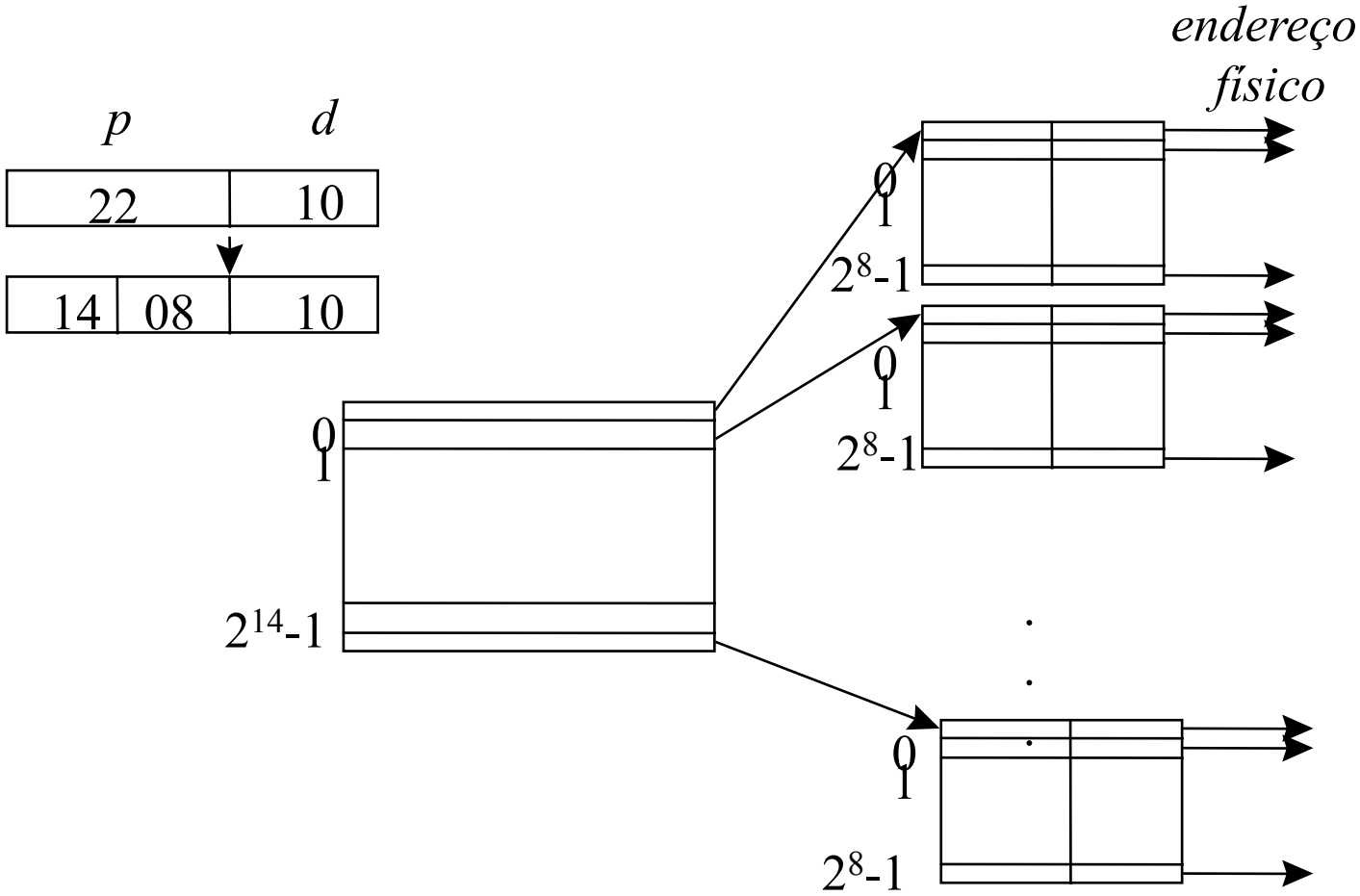
# Tabela de Páginas Forward-Mapped

- É necessária uma tabela de páginas por processo.
- O número de entradas depende do espaço de endereçamento da máquina (e.g. 32 bits) e do tamanho da página.
- Como as tabelas de páginas dependem do tamanho do espaço de endereçamento, atualmente elas são extremamente grandes.
- Para evitar que a tabela de páginas fique em memória o tempo todo, a maioria dos computadores atuais utilizam uma organização multinível.

# Tabela de Páginas Forward-Mapped
# Tabelas Multi-Nível

- A organização multinível baseia-se na seguinte observação: apesar do processo poder endereçar $2^n$ posições de memória, não quer dizer que ele vai endereçar $2^n$ posições de memória.
- Na organização multinível, o número da página é "quebrado" em *n* partes, onde *n* é o número de níveis.
- Existem tabelas de páginas de 1 nível (PDP-11), 2 níveis (VAX), 3 níveis (SunSPARC), 4 níveis (Motorola 68030). Além de 4 níveis, o projeto fica extremamente complexo e o ganho não é muito grande.

# Tabela de Páginas Forward-Mapped
# Tabelas Multi-Nível



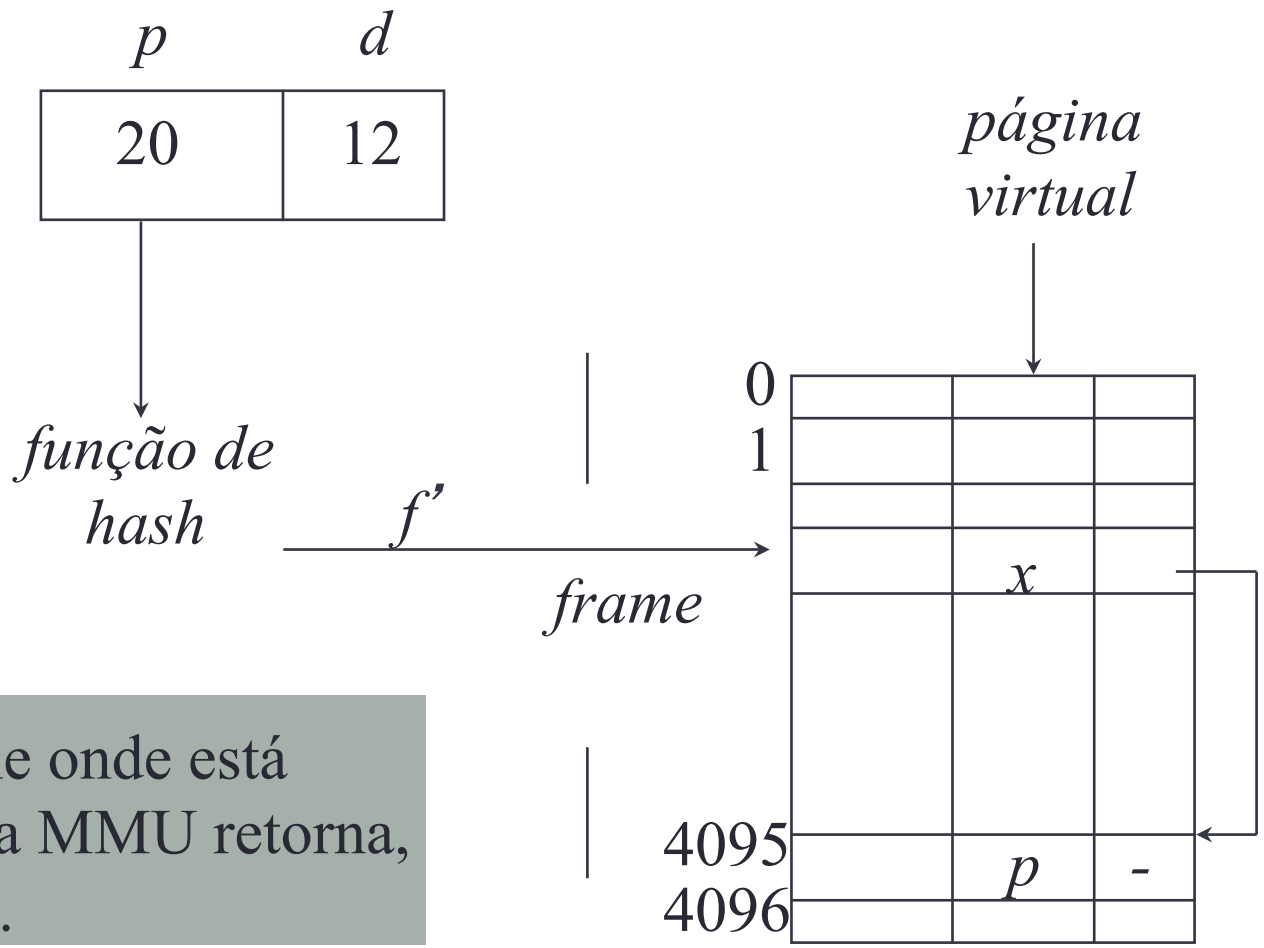
# Tabelas de Páginas Invertidas

- Com o advento dos processadores de 64 bits, temos um espaço de endereçamento de $2^{64}$ de posições. Com uma página de 4 kbytes, necessitariamos de $2^{52}$ entradas na tabela de páginas.
- Fica, então, inviável a utilização de tabelas de página forward-mapped.

# Tabela de Páginas Invertida

- A rigor, como a tabela de páginas fará o mepeamento de páginas virtuais em páginas físicas e o número de páginas físicas depende do tamanho real da memória, podemos criar uma tabela invertida, ou seja, classificada pela página física (frame) e não pela página virtual.
- O problema desta abordagem é que nós possuímos o endereço virtual e desejamos o endereço físico correspondente. Como fazer se a tabela é classificada por endereço físico?
  - utilizar uma tabela de hash

# Tabela de Páginas Invertida Funcionamento




Para a busca da frame onde está mapeada a página *p* a MMU retorna, neste exemplo, 4095.

# Substituição de Páginas

- Quando ocorre um page fault, o SO tem que carregar uma nova página em memória. O que fazer se a memória estiver cheia?
  - Retirar uma página da memória para dar lugar à página que chega.
  - Que página retirar?
    - Isso é decidido pelo algoritmo de substituição de páginas.

# Substituição de Páginas

- Algoritmo ótimo de substituição de páginas: escolhe sempre a página que está em memória física e que será referenciada em um futuro mais distante.
  - Não é um algoritmo realizável, pois não temos como conhecer o futuro.
  - É usado na avaliação de algoritmos realizáveis.

# Substituição de Páginas

- Algoritmos realizáveis de substituição de páginas:
  - NUR
  - FIFO
  - Segunda Chance
  - Relógio
  - LRU

# Substituição de Páginas - NRU

- O algoritmo NRU (não recentemente utilizada) utiliza os bits de referência (R) e modificação (M), contidos na tabela de páginas.
  - R: é setado (R=1) sempre que a página associada a ele for acessada (leitura ou escrita)
  - M: é setado (M=1) sempre que a página associada a ele for modificada (escrita).
  - A atualização destes bits é feita por hardware.

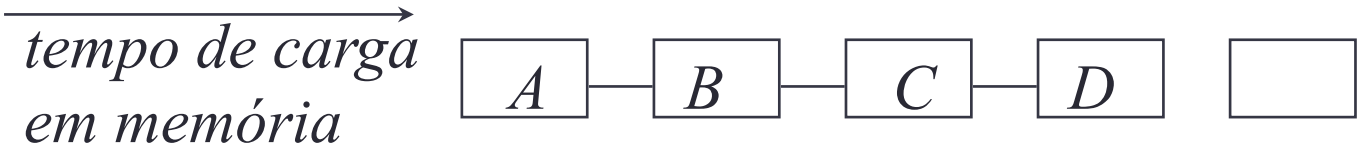
# Substituição de Páginas - NRU

- Quando o contexto do processo é carregado, os bits R e M de todas as páginas são zerados pelo SO.
- Periodicamente, o SO zera o bit R.
- Quando há um page fault, o SO analisa todas as páginas e as coloca em 4 categorias:
  - classe 0: não referenciada, não modificada
  - classe 1: não-referenciada, modificada
  - classe 2: referenciada, não-modificada
  - classe 3: referenciada, modificada
- O SO escolhe uma página da menor classe não vazia e a substitui.

# Substituição de Páginas - FIFO

- O algoritmo FIFO escolhe para substituição a página que foi carregada há mais tempo.
- As páginas em memória são mantidas em uma fila organizada por antiguidade. A página mais antiga está no início e a página mais recente está no final.
- Quando há um page fault, a página removida é aquela que está no início e a página que causou o page fault é colocada no final.

# Substituição de Páginas - FIFO

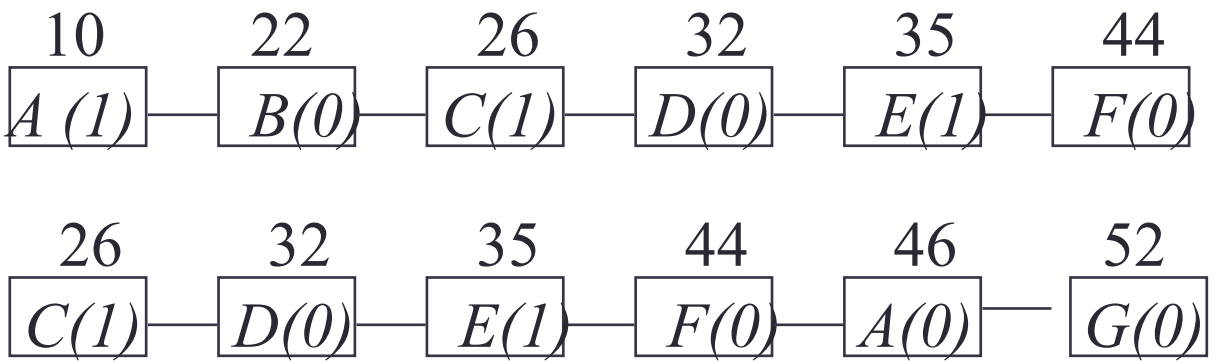



- Apesar de ser um algoritmo simples de se implementar, nada garante que uma página carregada a muito tempo não seja a mais utilizada. A utilização da página não está relacionada com o seu tempo de carga!
- Algoritmo raramente utilizado

# Substituição de Páginas
# Segunda Chance

- O algoritmo da segunda chance utiliza um algoritmo FIFO modificado.
- Quando há um page fault, o algoritmo analisa a página carregada há mais tempo.
- Se o bit R for 0, a página é escolhida para substituição.
- Se o bit R for 1, o bit R é zerado e a página é colocada no final da final.
- A busca continua até que haja uma página onde R=0.

# Substituição de Páginas
# Segunda Chance



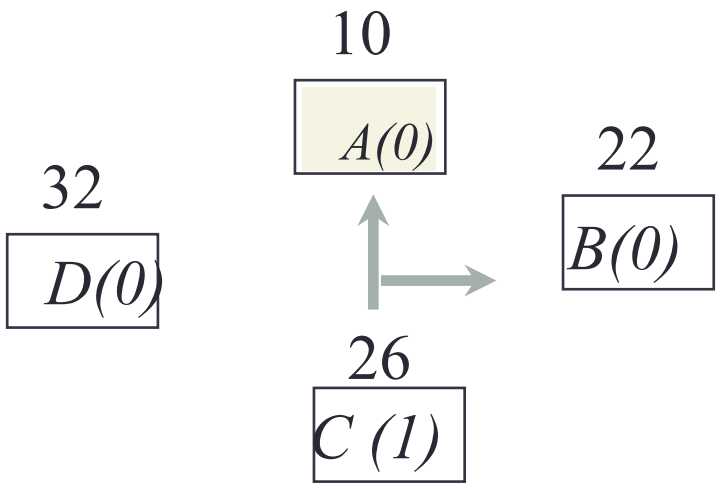

- O algoritmo da segunda chance procura uma página carregada há muito tempo que não tenha sido referenciada.
- Se todas as páginas tiverem sido referenciadas, ele degenera para FIFO puro.

# Substituição de Páginas
# Algoritmo do Relógio

- O algoritmo do relógio utiliza a mesma filosofia do algoritmo da segunda chance, porém não utiliza filas simples para armazenar as páginas.
- As páginas são mantidas em uma fila circular, com um ponteiro que aponta para a página mais antiga.

# Substituição de Páginas
# Algoritmo do Relógio

- Quando há um page fault, o algoritmo analisa a página apontada pelo ponteiro. Se o bit R for 0, a página é escolhida para substituição. Se o bit R for 1, o bit R é zerado e o ponteiro aponta para a próxima página. A busca continua até que haja uma página onde R=0.



# Substituição de Páginas - LRU

- Por causa do princípio da localidade, as páginas muito referenciadas nas últimas instruções tem grande chance de serem referenciadas pelas próximas instruções. Da mesma maneira, as páginas não referenciadas há muito tempo tem grande probabilidade de continuarem a não serem utilizadas por muito tempo.
- O algoritmo LRU (least recently used) determina que a página escolhida para substituição deve ser a página que foi referenciada no passado mais distante.

# Substituição de Páginas - LRU

- O algoritmo LRU apresenta bons resultados teóricos porém sua implementação é cara: necessita de uma lista encadeada organizada  pelo tempo do último acesso. A página escolhida será a página com tempo menos recente do último acesso.
- Existem algumas técnicas de implementação do LRU utilizando-se de hardware especial.
- Para SO de uso genérico, porém, o mais comum é a utilização de uma simulação por software do LRU, conhecida como NFU (não frequentemente utilizada).

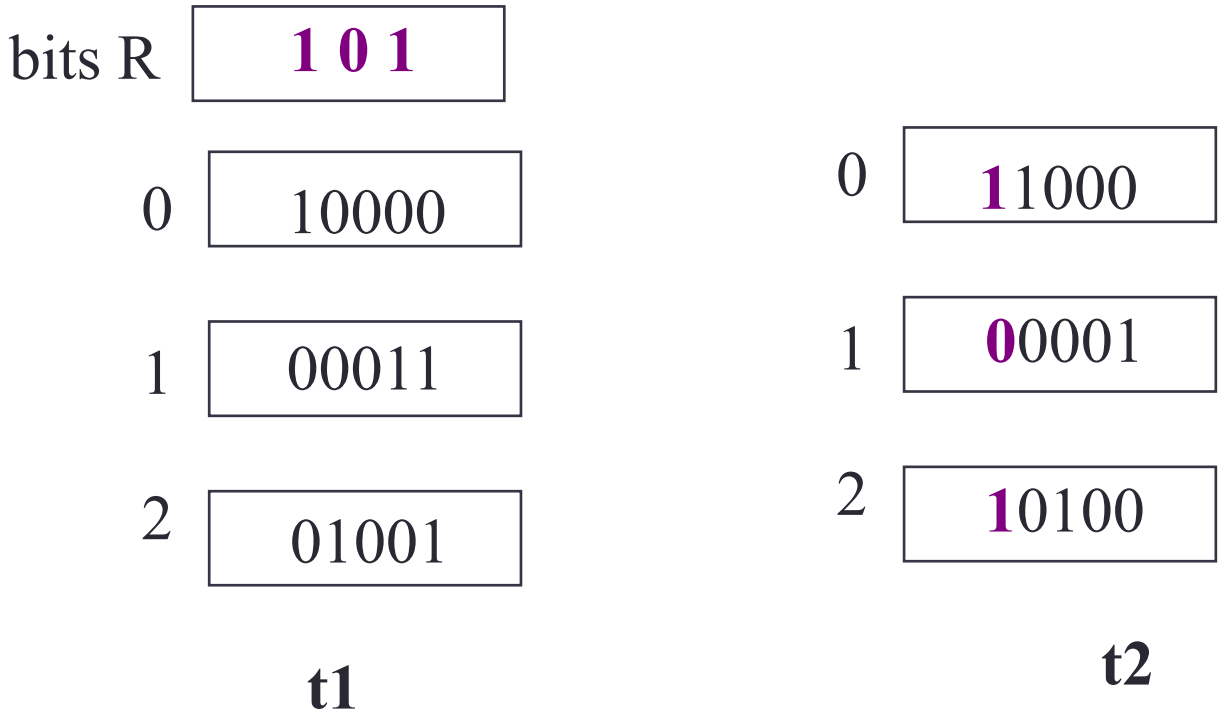
# Substituição de Páginas - NFU

- No NFU, necessitamos de um contador adicional, a ser mantido na tabela de páginas, para cada página. Periodicamente, o SO percorre todas as páginas em memória e adiciona o valor do bit R (0 ou 1) de cada página ao seu contador.
- Quando ocorre um page fault, a página escolhida é aquela que possui o menor contador.
- O problema do NFU é que uma página muito referenciada há muito tempo atrás raramente é escolhida para substituição pois o NFU não “esquece” estas referencias.

# Substituição de Páginas
# NFU com Aging

- Para que as referências sejam esquecidas, devemos utilizar uma técnica de aging. O número de bits no contador é o número de referências passadas que será considerado.
- Funcionamento do aging:
  - A cada interrupção de clock, é feito um shift do contador de cada página para a direita e o bit R é inserido à esquerda (bit mais significativo).
  - Quando há um page fault, o algoritmo escolhe a página com o menor contador.

# Substituição de Páginas
# NFU com Aging



# Substituição de Páginas
# NFU com Aging

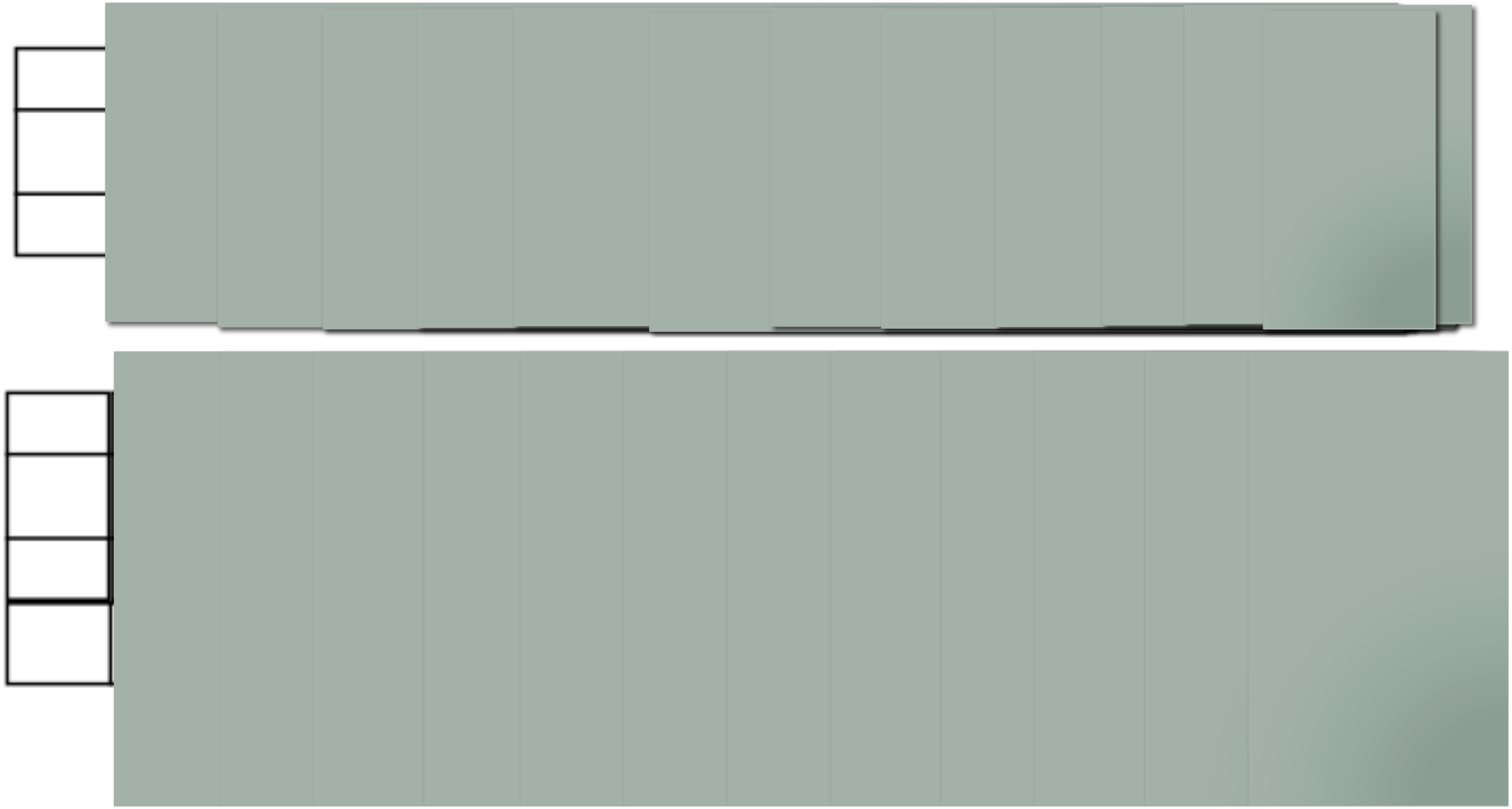
- Diferenças entre LRU e aging:
  - Se duas páginas possuem o contador = 0, o aging escolhe qualquer uma delas para substituição. Assim, assumindo um contador de 8 bits, uma página que foi referenciada há 9 ticks é considerada igual a uma página referenciada há 1000 ticks.
  - Se duas páginas tem o bit R=1, não podemos decidir qual delas foi referenciada por último.

# Modelagem de Sistemas Paginados
# Anomalia de Belady

- Intuitivamente, esperamos que, quanto maior o tamanho da memória real, menor ou igual será o número de page faults.
- Belady et al., em 1969, apresentou um caso onde o mesmo programa apresentava 9 page faults quando a memória possuia 3 frames e 10 page faults quando a memória possuía 4 frames. O algoritmo utilizado era o FIFO.
- Essa é conhecida como a anomalia de Belady.

# Modelagem de Sistemas Paginados
# Anomalia de Belady - Exemplo

Ordem de acesso às páginas: 0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4



# Modelagem de Sistemas Paginados
# Reference string

- A anomalia de Belady fez com que vários cientistas estudassem mais a fundo o comportamento de um sistema sob a estratégia de paginação.
- Ao longo de sua execução, um processo gera referências à páginas. Ao final de uma execução nós temos uma lista ordenada de acessos às páginas, conhecida como string de referência.

Exemplo de reference string: 0, 1, 2, 0, 1, 1, 2, 0, 2, 0,3, 4, 5, 3, 4, 5

# Modelagem de Sistemas Paginados
# Algoritmos de Pilha

- Para estudar um sistema paginado, consideram-se 3 itens:
  - o string de referência dos processos em execução.
  - o algoritmo de substituição de páginas
  - o número de frames disponíveis.
- Estudando este modelo, descobriu-se uma classe de algoritmos, denominados algoritmos de pilha. Para estes algoritmos, ficou provado que, se executarmos o mesmo string de referência com um número maior de frames, todas as páginas que estavam anteriormente nas frames estarão presentes na nova execução, além de páginas adicionais.
- São algoritmos de pilha o LRU e o ótimo. Os algoritmos baseados no critério FIFO não o são.

# Modelagem de Sistemas Paginados
# Paginação por Demanda

- Em um sistema paginado, a rigor, não é necessário que nenhuma página do processo esteja em memória para que ele possa iniciar sua execução.
- Logo que a primeira instrução for buscada, ocorre um page fault e a página referenciada é trazida para a memória física. Assim, um processo pode continuar executando e as páginas são trazidas para a memória à medida em que são referenciadas. Esta
- estratégia é chamada paginação por demanda.
- A paginação por demanda funciona bem porque, normalmente, um programa não referencia todas as suas posições em uma única execução, apresentando, assim, uma certa localidade de referência.

# Modelagem de Sistemas Paginados
# Working Set

- O princípio da localidade de referência diz que, em cada fase da execução, o processo só referencia um pequeno subconjunto de páginas, sendo um subconjunto diferente por fase.
- O subconjunto de páginas de um processo que está atualmente sendo utilizado é chamado conjunto de trabalho.

Exemplo de reference string: 0, 1, 2, 0, 1, 1, 2, 0, 2, 0, 3, 4, 3, 3, 4, 3

*ws1={0,1,2}*  *ws2={3,4}*

# Modelagem de Sistemas Paginados Trashing

- Se a memória física disponível não for capaz de abrigar totalmente o conjunto de trabalho, o processo vai gerar muitos page faults e executar muito lentamente.
- Trashing: o sistema entra no estado de trashing se um processo causa muitos page faults ao executar poucas instruções. Neste caso, a performance do sistema se degrada enormemente.
- Em um modelo puro de paginação à demanda, nós incorremos em trashing limitado sempre que necessitamos trazer o working set para memória.

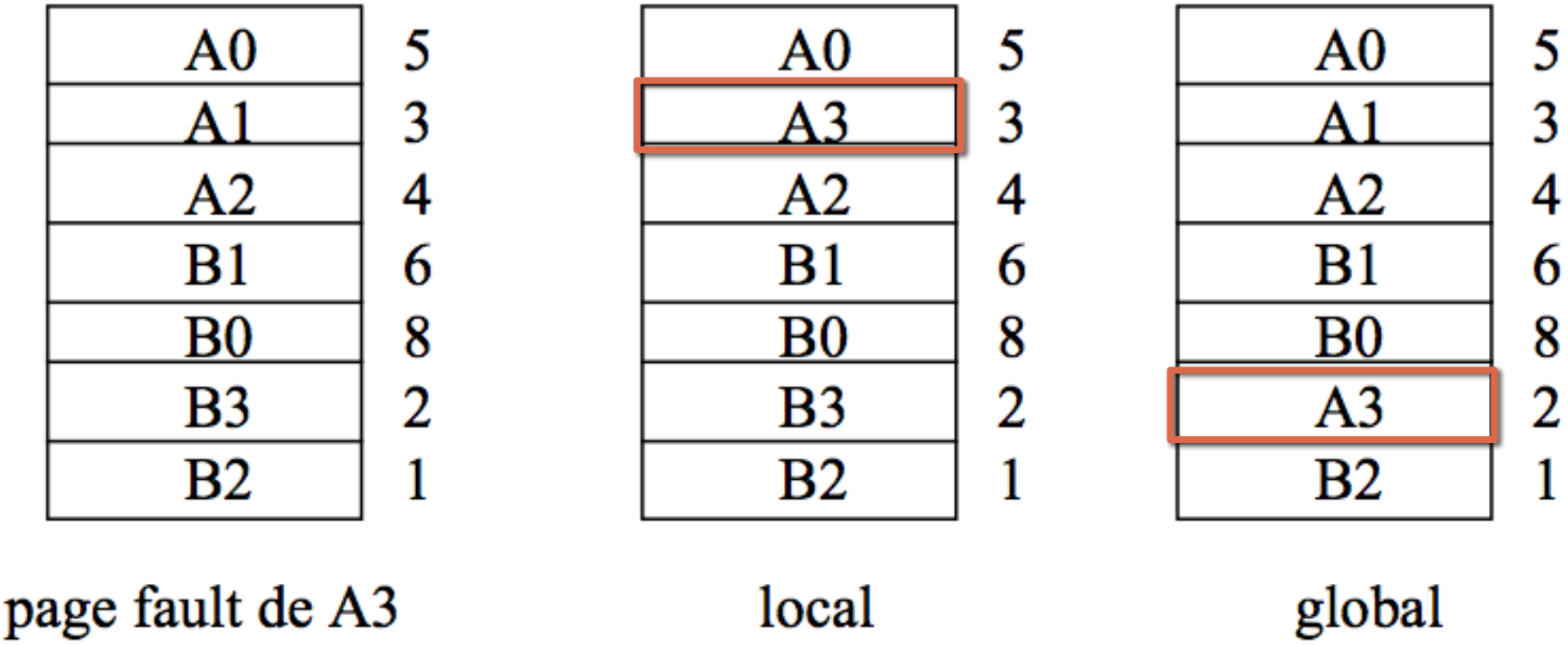
# Modelagem de Sistemas Paginados Prefetching

- Notadamente, no início da execução, o processo incorreria em muitos page faults se algumas de suas páginas não estivessem pré-carregadas em memória.
- Atualmente, ao começar uma execução, as *n* primeiras páginas do processo são carregadas em memória, onde *n* é a taxa de pré-carga.
- Vários outros algoritmos de pré-carga foram propostos mas, como é impossível prever o futuro no caso mais genérico, muitos deles “poluem” a memória com páginas que nunca serão utilizadas e eventualmente retirando páginas que serão usadas em um futuro próximo.

# Substituição Local x Global

- A memória, sendo única, é compartilhada por vários processos. Neste contexto, devemos decidir o quanto de memória será alocada a cada processo.
- Em outras palavras, devemos determinar que conjunto de páginas deve ser analisado pelo algoritmo de substituição.
  - **política de alocação local:** somente as páginas do processo que gerou o page fault devem fazer parte do conjunto de páginas a serem analisadas pelo algoritmo. Neste caso, cada processo tem um número fixo de páginas.
  - **política de alocação global:** todas as páginas que estão em memória devem ser analisadas, independente do processo que gerou o page fault. Em geral, algoritmos globais dão melhores resultados.

# Substituição Local x Global
# Exemplo



# Tamanho da Página

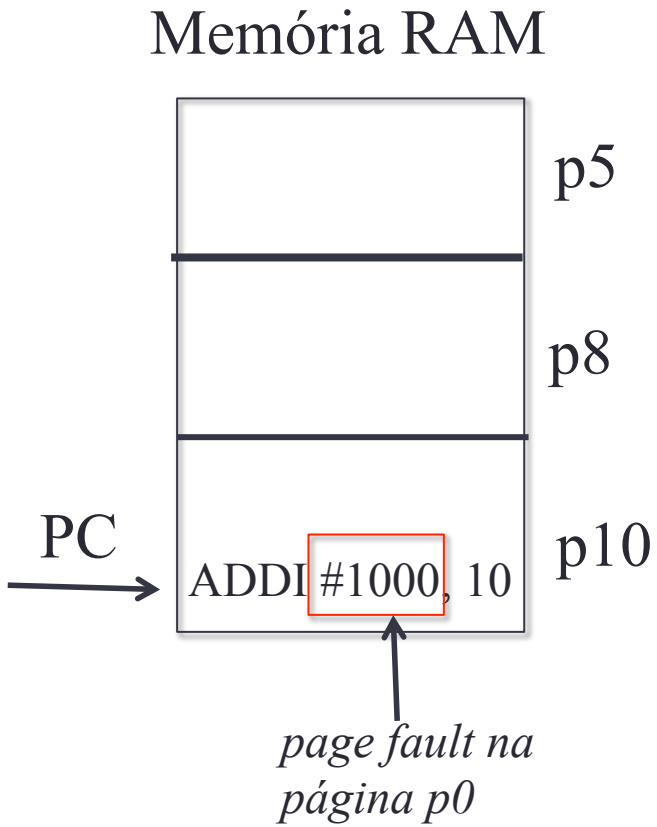
- O tamanho da página geralmente é um parâmetro configurável do SO.
- Mesmo que a MMU trate páginas de um tamanho fixo, e.g. 512 bytes, o SO pode utilizar páginas de 1k, alocando sempre 2 páginas consecutivas a cada page fault.
- Estudos teóricos determinam que o tamanho ótimo da página depende bascamente do tamanho médio de um processo e do tamanho de uma entrada da tabela de páginas. A maioria dos sistemas usa páginas de 512 bytes a 8kbytes.

# Tamanho da Página
# Vantagens (V) x Desvantagens (D)

- Páginas grandes:
  - D: maior fragmentação interna: grande parte da última página do processo é perdida;
  - D: grande parte da memória é ocupada com espaço inativo (não fazem parte do working set mas estão na mesma página).
  - V:  tabela de páginas menor.
- Páginas pequenas:
- V: menor fragmentação interna;
- V: melhor aproveitamento da memória com espaços alocados ativos.
- D: tabela de páginas maior.

# Aspectos de Implementação
# Backup de Instrução

- Se ocorre um page fault na execução de uma instrução, ela deve ser reexecutada. Como uma instrução é composta por vários bytes, o page fault pode ser no meio de uma instrução (no operando, por exemplo). O valor do PC no instante da falha depende do microcódigo e do operando que gerou a falha.
- O hardware deve ser capaz, então, de desfazer o que ele já tinha executado da instrução e reexecutá-la inteiramente quando a página chegar.

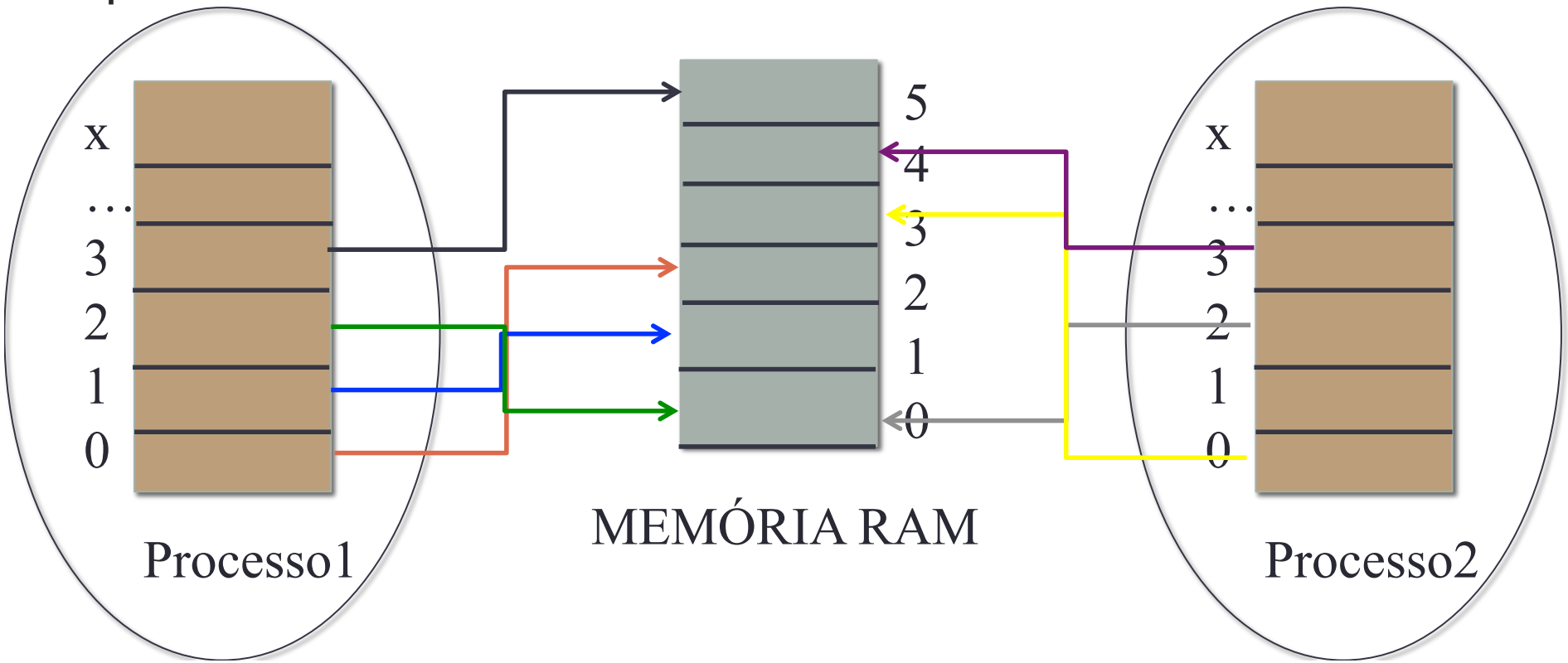


# Lock de páginas

- Em algumas situações, nós desejamos que uma página fique bloqueada um memória, ou seja, que o algoritmo de substituição não a escolha.
- Isso é implementado através de um bit de bloqueio, inserido na tabela de páginas.
- O sistema operacional sempre pode fazer o lock de páginas.
- Em alguns sistemas, o usuário pode fazer o lock que algumas de suas páginas.

# Compartilhamento de Páginas

- Em um sistema multiprogramado, existem geralmente vários processos executando o mesmo código (ex: editor de texto). Neste caso, o código do editor de texto é marcado como read-only e ele é compartilhado entre diversos processos.



# Área de Swap

- Quando uma página modificada é escolhida para substituição ela é escrita em disco, em uma área reservada, denominada área de swap.
- O tamanho da área de swap pode ser fixo ou dinâmico.
- A maioria dos sistemas utiliza tamanhos fixos.

# Paging Daemons

- O tempo do tratamento do page fault é alto (da ordem do milisegundo). Se esperarmos a memória ficar completamente cheia para executarmos o algoritmo de substituição de páginas, o tempo do tratamento do page fault vai ser acrescido do tempo de execução do algoritmo de substituição e da escrita da página escolhida em disco, se esta tiver sido modificada.
- Assim, a maioria dos sistemas utiliza processos que rodam em background (paging daemons). Estes processos são acordados quando a ocupação da memória atinge um limite pré- estabelecido (90% por exemplo) e executam o algoritmo de substituição de páginas.

# Tratamento da Falta de Página

- O hardware gera um trap para o kernel e salva o PC corrente na pilha.
- O hardware executa uma rotina em código de máquina que salva os demais registradores e outras informações. No final de sua execução, a rotina chama o SO.
- O SO recebe o trap e determina que ocorreu um page fault. O SO determina a página virtual necessária para resolver o page fault (geralmente o número da página virtual está em um registrador especial).

# Tratamento da Falta de Página

- Tendo o endereço virtual em falta, o SO verifica se o endereço é válido e se houve violação de proteção
- O SO determina uma page frame livre para a página em falta. Se não houver moldura livre:
  - O algoritmo de substituição é executado
  - Se a página selecionada tiver sido modificada, ela deve ser escrita em disco. A page frame é marcada como reservada. O SO escolhe um novo processo para rodar

# Tratamento da Falta de Página

- Quando o SO tem certeza de que a page frame escolhida está livre, ele determina o endereço de disco onde se encontra a página que causou o page fault.
- Durante a busca da página de disco para a memória, o SO escolhe um novo processo para rodar
- Ao chegar a interrupção de disco que indica o término da cópia da página, a tabela de páginas é alterada e a page frame é marcada como ocupada.

# Tratamento da Falta de Página

- A instrução que causou a falta é carregada no registrador de instruções (RI) e o seu endereço é carregado no PC.
- O processo cuja instrução causou o page fault é colocado para rodar.
- O SO termina a execução, voltando a ser executada a rotina em código de máquina que o chamou.
- A rotina restaura os registradores e demais informações para a situação anterior à falta e dá o processador ao processo de usuário.

# Segmentação

- A segmentação é uma abordagem que divide o espaço de endereçamento em unidades de tamanho variável, ditas segmentos.
- Cada segmento é constituído de uma sequência linear de endereços, de 0 a um valor máximo.
- Segmentos diferentes podem ter (e tem) tamanhos diferentes.
- O segmento é uma entidade lógica e deve ser definido pelo programador. O programador deve agrupar em um mesmo segmento dados com o mesmo significado (tabelas, matrizes, pilhas, etc).

# Segmentação

- Os procedimentos de um programa podem ser colocados em segmentos distintos (um em cada segmento). No caso de alteração do código fonte de um procedimento, só ele deve ser compilado. Os outros segmentos ocupam espaços de endereçamento distintos.
- A segmentação facilita a implementação de bibliotecas compartilhadas. As bibliotecas compartilhadas não são ligadas no código executável do programa. Seu código é colocado em um segmento, que pode ser compartilhado por vários processos, em tempo de execução.

# Segmentação

- A proteção de segmentos é bem mais útil que a proteção de páginas, já que os segmentos contem geralmente só um tipo de objetos.
- A disposição de dados em páginas é “acidental”.
- Na segmentação, a disposição de dados em segmentos é determinada pelo programador.

# Segmentação

- A segmentação, como trata de unidades de tamanho variável, sofre de fragmentação externa e geralmente necessita de compactação periódica de memória.
- Por esta razão, a segmentação pura raramente é empregada. Ao invés disso, utiliza-se a técnica de segmentação paginada, proposta no sistema Multics.

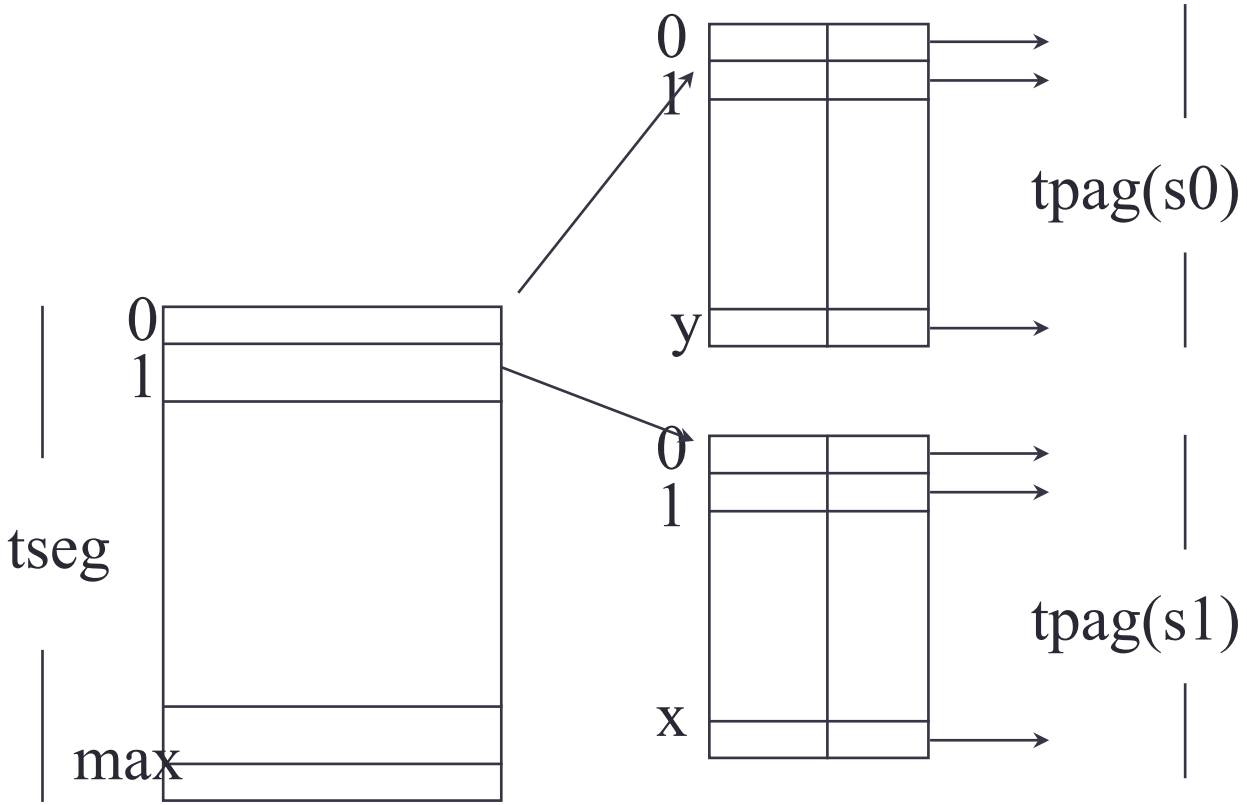
# Segmentação Paginada

- Como agora um processo é quebrado em um número variável de segmentos cada qual com seu espaço de endereçamento, fica difícil manter todos os segmentos de um processo em memória ao mesmo tempo.
- Exemplos típicos de segmentos são os segmentos de código, dados e pilha de um processo.
- Por isso, surgiu a idéia de paginar os segmentos.

# Segmentação Paginada

- Na segmentação paginada, somente as páginas realmente necessárias são mantidas em memória.
- Neste caso, cada processo possui uma tabela de segmentos. Cada segmento tem uma entrada na tabela de segmentos. A tabela de segmentos aponta para a tabela de páginas do segmento em questão.

# Segmentação Paginada



# Hierarquia de Memória (simplificada)