Universidade Federal
de Mato Grosso
Campus Rondonópolis

Sistemas Operacionais

Aula 8 – Deadlocks (2)

Cleyton Slaviero

cslaviero@gmail.com

Deadlocks

- **Recuperação** de deadlock
  - Por meio de **preempção**
    - Possibilidade de retirar temporariamente um recurso de seu atual dono (processo) e entregá-lo a outro processo
    - Depende muito da natureza do recurso
    - Frequentemente difícil ou impossível

Universidade Federal de Mato Grosso
Campus Rondonópolis

Deadlocks

- Recuperação de Deadlocks
  - Por meio de **rollback**:
    - O estado de cada processo (e recurso por ele usado) é periodicamente armazenado em um arquivo de verificação (checkpoint file);
    - Novas checagens são armazenadas em novos arquivos, à medida que o processo executa
    - Quando ocorre um deadlock, o processo que detém o recurso é voltado a um ponto antes de adquirir esse recurso (via o checkpoint file apropriado)
      - O trabalho feito após esse ponto é perdido
      - O processo é retornado a um momento em que não possuía o recurso, que será então dado a outro

Deadlocks

- Recuperação de Deadlocks:
  - Por meio de eliminação de processos (kill):
    - Um ou mais processos que estão no ciclo com deadlock são interrompidos.
      - Talvez isso evite o deadlock. Senão, continue eliminando até quebrar o ciclo
    - Melhor solução para processos que não causam algum efeito negativo ao sistema;
      - Ex1.: compilação – sem problemas;
        Ex2.: atualização de um base de dados – problemas;

Evitando deadlocks

- Até então…
  - Quando um processo quer recursos, pede todos de uma vez
    - Nem sempre é assim!
  - Ha um algoritmo que pode evitar o deadlock fazendo sempre a escolha certa?
    - SIM! Mas
    - Precisamos saber algumas informações de antemão

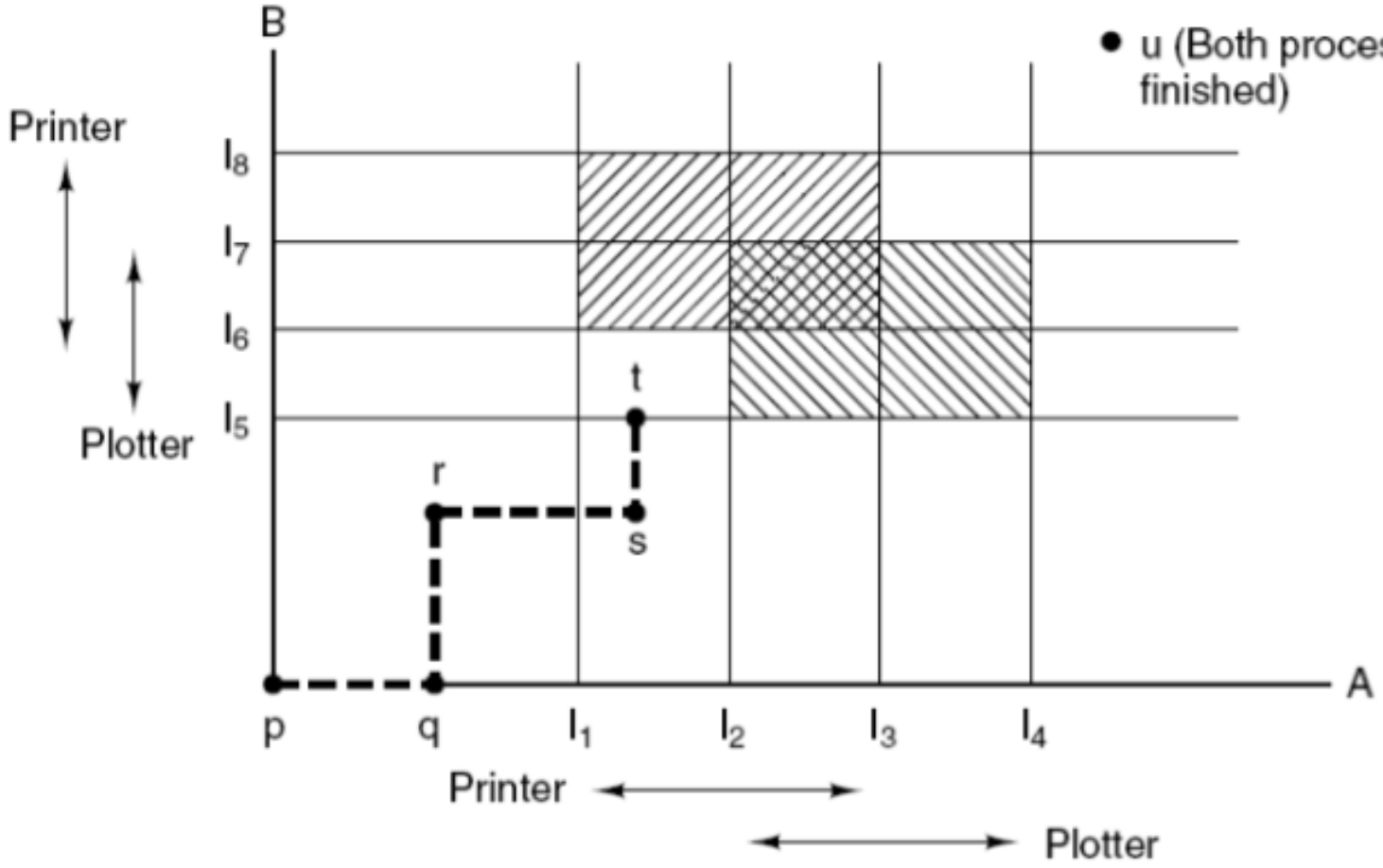
Evitando Deadlocks

- Algoritmos
  - Banqueiro para um único tipo de recurso;
  - Banqueiro para vários tipos de recursos;
- Usam a noção de Estados Seguros e Inseguros;

Universidade Federal
de Mato Grosso
Campus Rondonópolis

Evitando deadlocks

- Trajetória de recursos
  - Eixo horizontal: instruções de A
  - Eixo vertical: instruções de B
  - Cada um necessita de uma impressora e plotter em tempos distintos
  - Regiões cinzas são zonas não seguras
  - Em t, B quer um recurso **.O que fazer?**



Universidade Federal de Mato Grosso
Campus Rondonópolis

Evitando Deadlocks

- Evitar dinamicamente o problema
  - Estados seguros: não provocam deadlocks e há uma maneira de atender a todas as requisições pendentes finalizando normalmente todos os processos;
    - A partir de um estado seguro, existe a garantia de que os processos terminarão;
  - Estados inseguros: podem provocar deadlocks, mas não necessariamente provocam;
    - A partir de um estado inseguro, não é possível garantir que os processos terminarão corretamente;

Evitando Deadlocks

- Evitar dinamicamente o problema
  - Usa as mesmas estruturas da detecção com vários recursos
    - Exemplo: um estado atual consiste das estruturas E, A, C e R
      - Estado seguro
        - Se houver alguma ordem de alocação na qual todo processo irá terminar, mesmo caso todos peçam seu número máximo de recursos imediatamente

Evitando Deadlocks

- Evitar dinamicamente o problema
  - Estado seguro
    - Ex: tabela para um único recurso

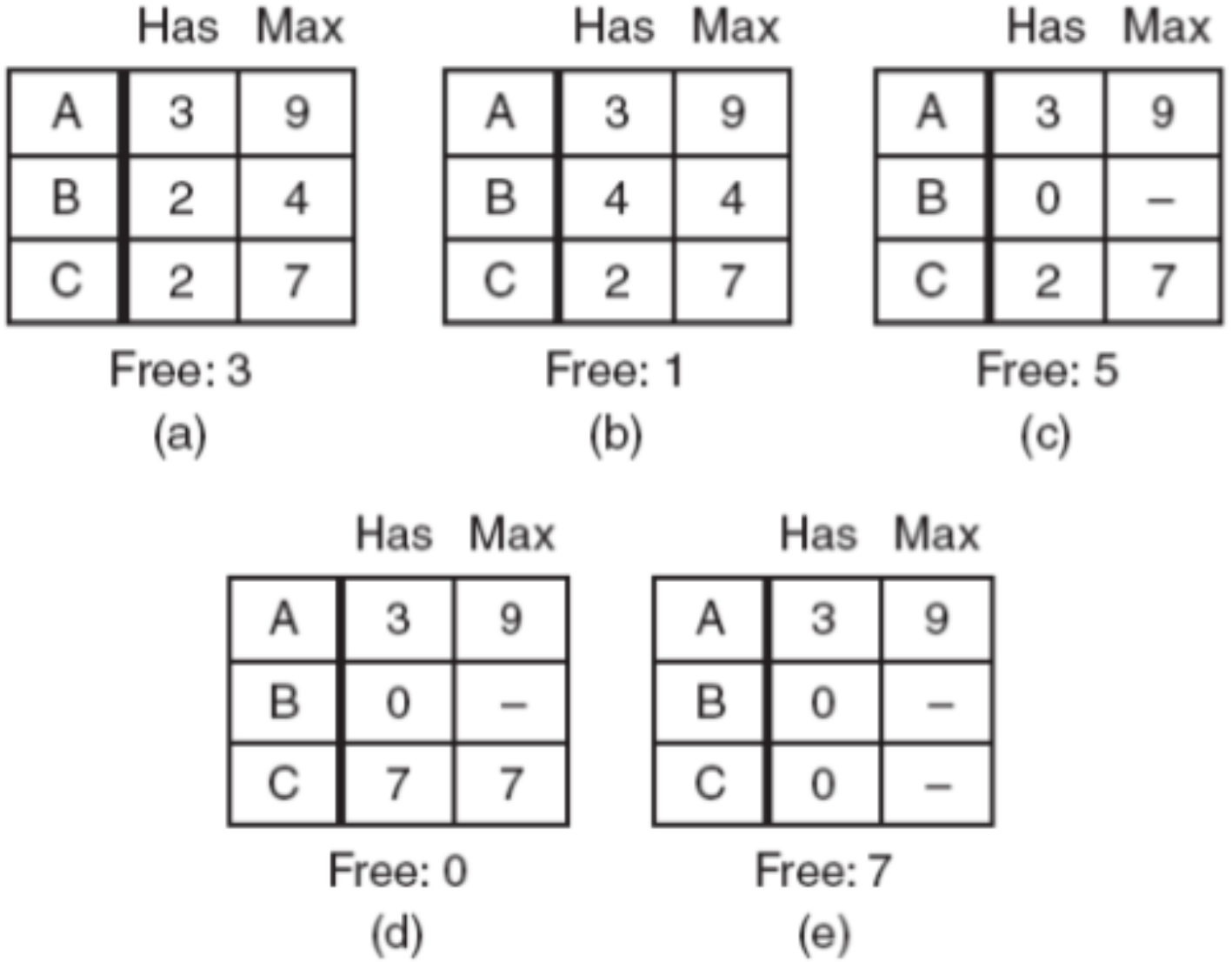
E = 10

	C	R
A	3	9
B	2	4
C	2	7

A 3

Evitando Deadlocks

- Estado seguro
  - Há uma sequência que permite que todos os processos terminem



	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1

(b)

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5

(c)

	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0

(d)

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7

(e)

Evitando Deadlocks

- Evitar dinamicamente o problema
  - Estado seguro
    - Suponha agora a configuração abaixo
      - Essa sequência é segura?

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

(c)

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4

(d)

Evitando Deadlocks

- Evitar dinamicamente o problema
  - Estado seguro
    - Não há seqüência capaz de garantir que os programas terminem
    - A decisão que moveu de (a) para (b) levou de um estado seguro a um inseguro
      - Não deveríamos tê-la tomado

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

(c)

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4

(d)

Evitando Deadlocks

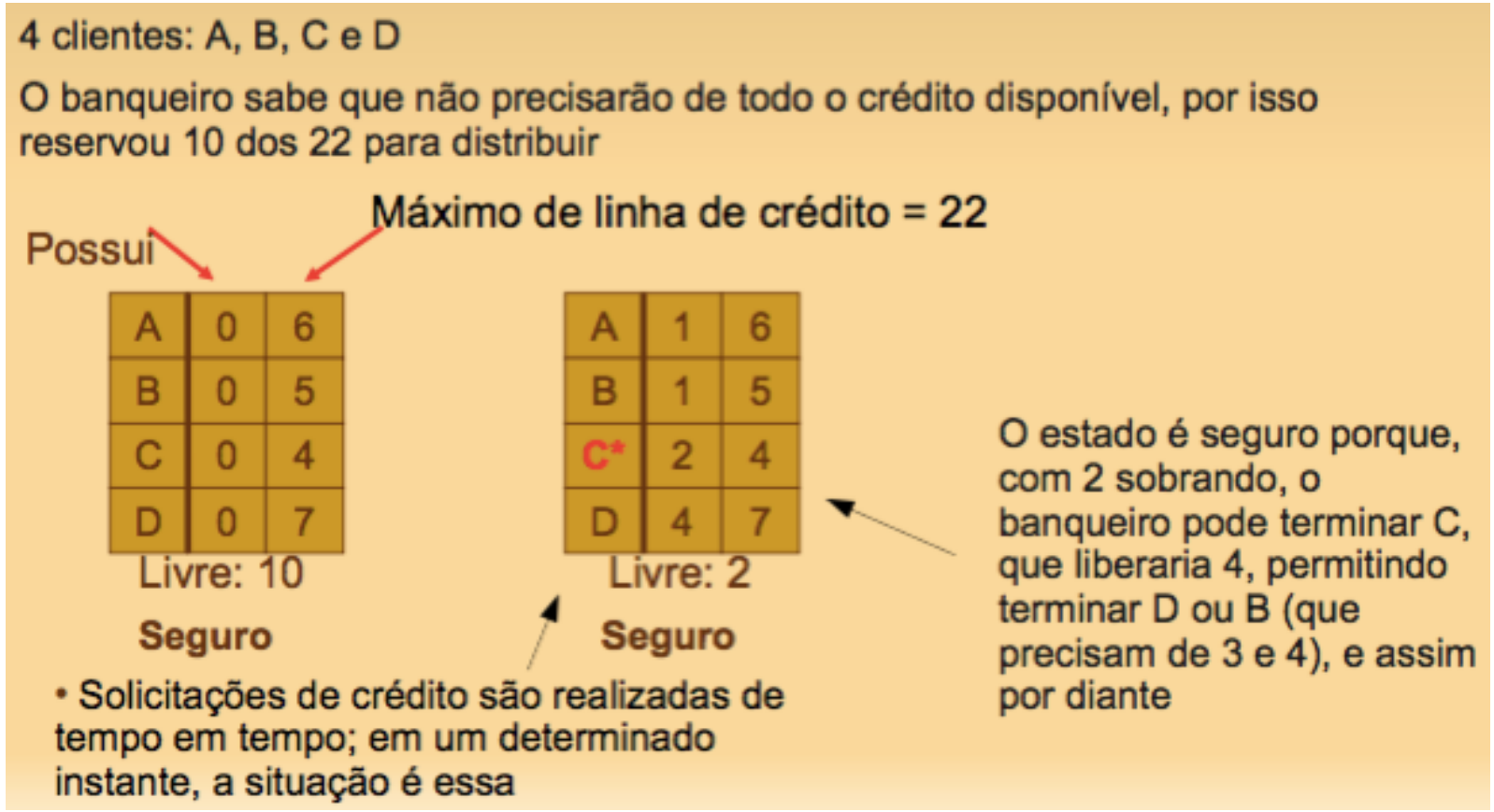
- Algoritmo do banqueiro
  - Idealizado por Dijkstra (1965);
  - Algoritmo de escalonamento capaz de evitar deadlock
  - Premissas adotadas por um ”banqueiro” (SO) para garantir ou não crédito (recursos) para seus clientes (processos);
    - Verifica se atender um pedido leva a um estado inseguro.
    - Se levar, o pedido é negado; senão, é executado
  - A intuição é garantir que sempre haverá possiblidade de alocar recursos solicitados para alguém

Evitando Deadlocks

- Algoritmo do Banqueiro para um único tipo de recurso:
  - Considera cada pedido à medida em que ocorre, e vê se atendê-lo leva a um estado seguro
  - Para ver se o estado é seguro, o banqueiro verifica se tem recursos suficientes para satisfazer algum cliente
  - Se tiver, assume que os ”empréstimos” serão pagos, e o cliente mais próximo do limite é checado
  - Sempre tenta emprestar o máximo a um único por vez

Evitando Deadlocks

- Algoritmo do banqueiro para um único tipo de recurso



Evitando Deadlocks

- Algoritmo do banqueiro para um único tipo de recurso

Evitando Deadlocks

- Algoritmo do Banqueiro para **vários** tipos de recursos:
  - Mesma idéia, mas duas matrizes são utilizadas;

Process	Tape drives	Plotters	Printers	Blu-rays
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Matriz **C**

Process	Tape drives	Plotters	Printers	Blu-rays
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still assigned

Matriz **R**

E = (6342)
P = (5322)
A = (1020)

Evitando Deadlocks

- Algoritmo do Banqueiro para vários tipos de recursos:
  - Busque em R uma linha que possa ser satisfeita pelos valores em A (deve possuir valores todos menores ou iguais aos de A)
    - Se não existir, pode haver deadlock
  - Assuma que o processo na linha escolhida pede todos os recursos de que precisa e termina.
    - Marque este processo como terminado e adicione todos seus recursos a A
  - Repita os passos acima até que todos os processos sejam marcados como terminados (caso em que o estado inicial era seguro) ou reste processo que não possa ser satisfeito (deadlock)

O que aconteceria se atendêssemos uma requisição de B?

Processos	Unidade de Fita	Plotters	Impressoras	Unidade de CD-ROM
A	3	0	1	1
B	0	1	1	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

C = Recursos Alocados

Alocados → P = (5 3 3 2);
Disponíveis → A = (1 0 1 0);

A	1	1	0	0
B	0	1	0	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

R = Recursos ainda necessários

- Podem ser atendidos: D, A ou E, C;

Ao atender E, teríamos um deadlock

Processos	Unidade de Fita	Plotters	Impressoras	Unidade de CD-ROM
A	3	0	1	1
B	0	1	1	0
C	1	1	1	0
D	1	1	0	1
E	0	0	1	0

C = Recursos Alocados

Alocados → P = (5 3 4 2);
Disponíveis → A = (1 0 0 0);

A	1	1	0	0
B	0	1	0	2
C	3	1	0	0
D	0	0	1	0
E	2	1	0	0

R = Recursos ainda necessários

- *Solução*: Adiar a requisição de E por alguns instantes;

Evitando Deadlocks

- Desvantagens
  - Pouco utilizado, pois é raramente se sabe quais recursos serão necessários;
  - Escalonamento cuidadoso é caro para o sistema;
  - O número de processos é dinâmico e pode variar constantemente, tornando o algoritmo custoso;
  - Recursos podem desaparecer/quebrar
- Vantagem
  - Na teoria o algoritmo é ótimo;
- Alguns sistemas usam heurísticas próximas as do banqueiro
  - Exemplo: redes

Prevenindo Deadlocks

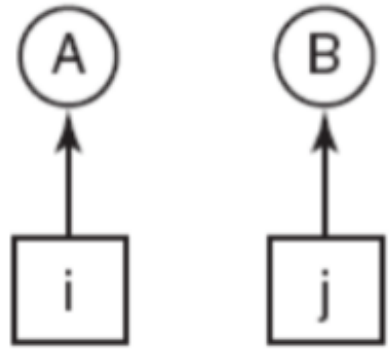
- Atacar uma das quatro condições:
- Exclusão mútua
  - Alocar todos os recursos usando um buffer/spool
    - Processos podem gerar output ao mesmo tempo
    - Deadlock pode ocorrer!!
    - Ainda assim é uma ideia interessante limitar ao máximo os processos que podem de fato solicitar o recurso
- Uso e espera
  - Processos requisitam todos os recursos de que precisam antes da execução
    - Difícil saber!
    - Recursos podem não ser usados de forma ótima
  - Podemos tentar também fazer com que o processo libere todos os recursos e pegue quando precisar, caso solicite um novo

Prevenindo deadlocks

- Não preempção
  - Retirar recursos dos processos
    - Pode ser ruim dependendo do tipo de recurso
    - Praticamente não implementável
- Espera circular
  - Ordenar numericamente os recursos, e realizar solicitações em ordem numérica
    - Não há ciclos
  - Permitir que o processo use um recurso por vez

1. Imagesetter
2. Scanner
3. Plotter
4. Tape drive
5. CD-ROM drive

(a)



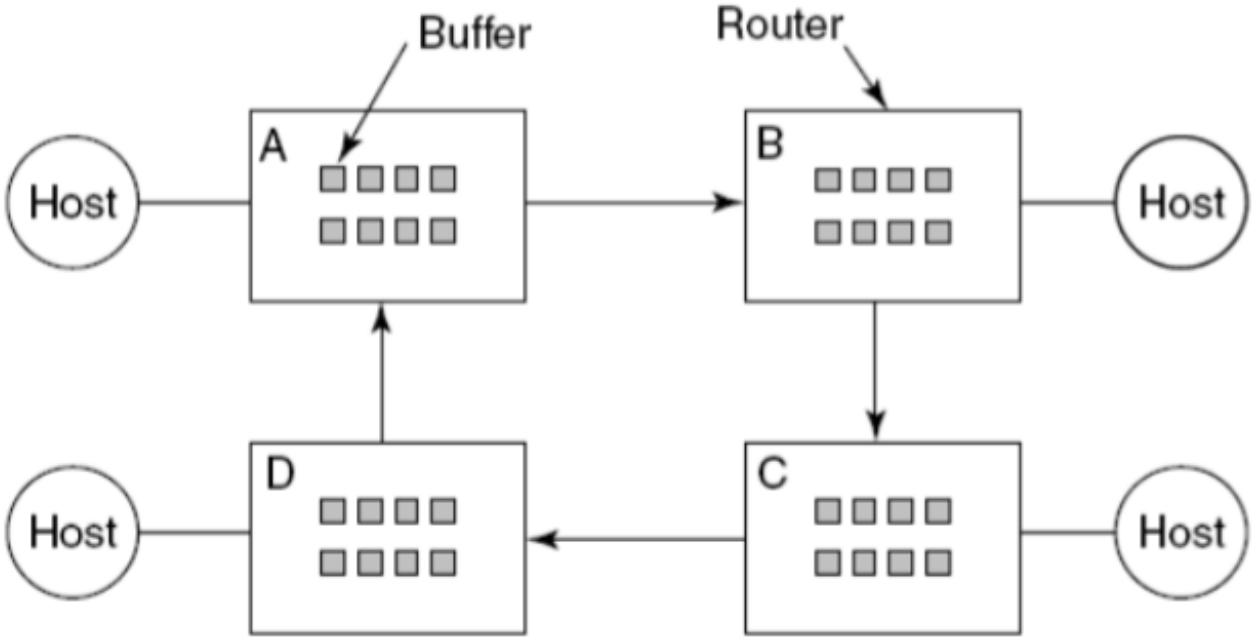
(b)

Outras questões

- Bloqueio em duas fases
  - Num banco de dados
    - Tenta-se travar todos os registros que precisa
    - Em seguida, faz os updates e libera as travas
    - Se na primeira fase alguém estiver bloqueado, tenta com todos de novo depois
  - Não é aplicável em geral
    - Só funciona caso a parada de processos seja possível e bem definida

Outras questões

- Deadlock de comunicação
  - Pode ocorrer quando dois processos esperam a resposta/comando um do outro
- Não dá pra evitar simplesmente numerando processos ou bloqueando
  - Solução: timeouts!
- Deadlocks de recursos também podem ocorrem em ambientes de rede!



Outras questões

- Livelock
  - Exemplo:
    - Duas pessoas andando uma de encontro para a outra na rua
    - Se as duas decidem ir para o mesmo lado...

- Um processo pode tentar desistir de locks que já adquiriu
- Se outro processo tem a mesma ideia…
  - Livelock!

Outras questões

- Inanição (Starvation)
  - Todos os processos devem conseguir utilizar os recursos que precisam, sem ter que esperar indefinidamente;
    - Solução? Colocar os processos em uma fila
- Alguns não fazem a distinção entre starvation e deadlock

Universidade Federal
de Mato Grosso
Campus Rondonópolis

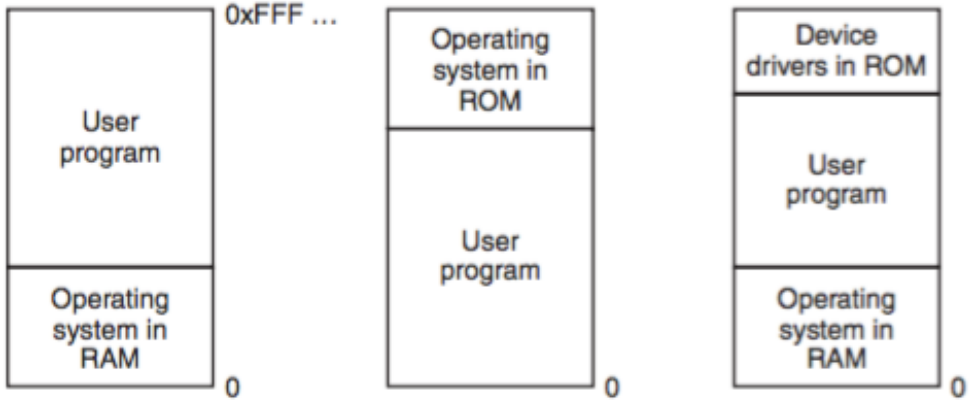
Gerenciamento de Memória (1)

Introdução

- Mundo perfeito
  - Memórias rápidas, infinitas, baratas
  - Não vivemos num mundo perfeito ☹
- Hierarquia de memória
  - Diferentes tamanhos/tipos/velocidades de memória?
- Como gerenciar tudo isso?
  - **Gerenciador de memória**
    - Manter o rastro de memória em uso, alocar memória para processos que precisam, e desalocar quando nao precisa mais

Nenhuma abstração de memória

- O programa vê /manipula a memória diretamente
  - MOV REGISTER1,1000
- Diferentes formas de colocar a memória
- Como rodar diversos programas?
  - Salvando tudo em disco



Troca de processos

- Com um sistema em lote, é simples e eficiente organizar a memória em partições fixas.
- Em sistemas de tempo compartilhado:
  - Pode não existir memória suficiente para conter todos os processos ativos
  - Os processos excedentes são mantidos em disco e trazidos dinamicamente para a memória a fim se serem executados

Troca de processos

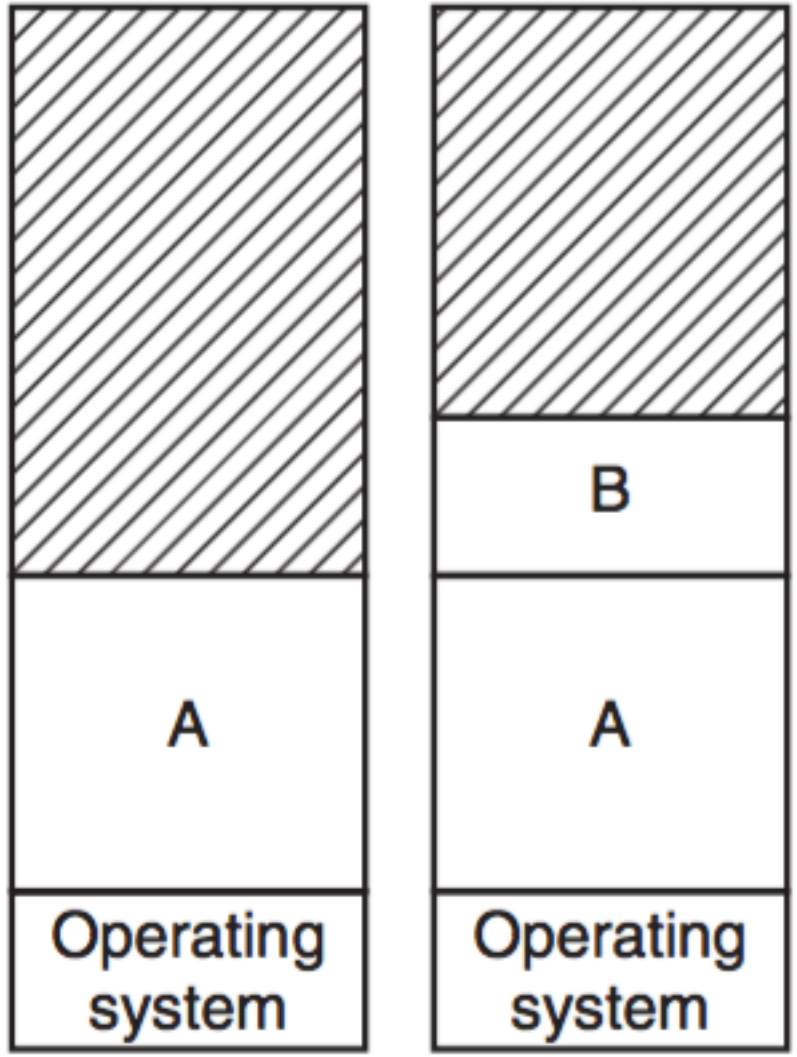
- Existem 2 processos gerais que podem ser usados:
  - A troca de processos (swapping):
    - Forma mais simples
    - Consiste em trazer totalmente cada processo para a memória, executá-lo durante um tempo e, então, devolvê-lo ao disco
  - Memória Virtual:
    - Permite que programas possam ser executados mesmo que estejam parcialmente carregados na memória principal

Swapping

- Chaveamento de processos inteiros entre a memória principal e o disco;
- Swap-out
  - Da memória para uma região especial do disco (swap)
- Swap-in
  - Do disco para a memória
- Pode ser utilizado tanto com partições fixas quanto com partições variáveis

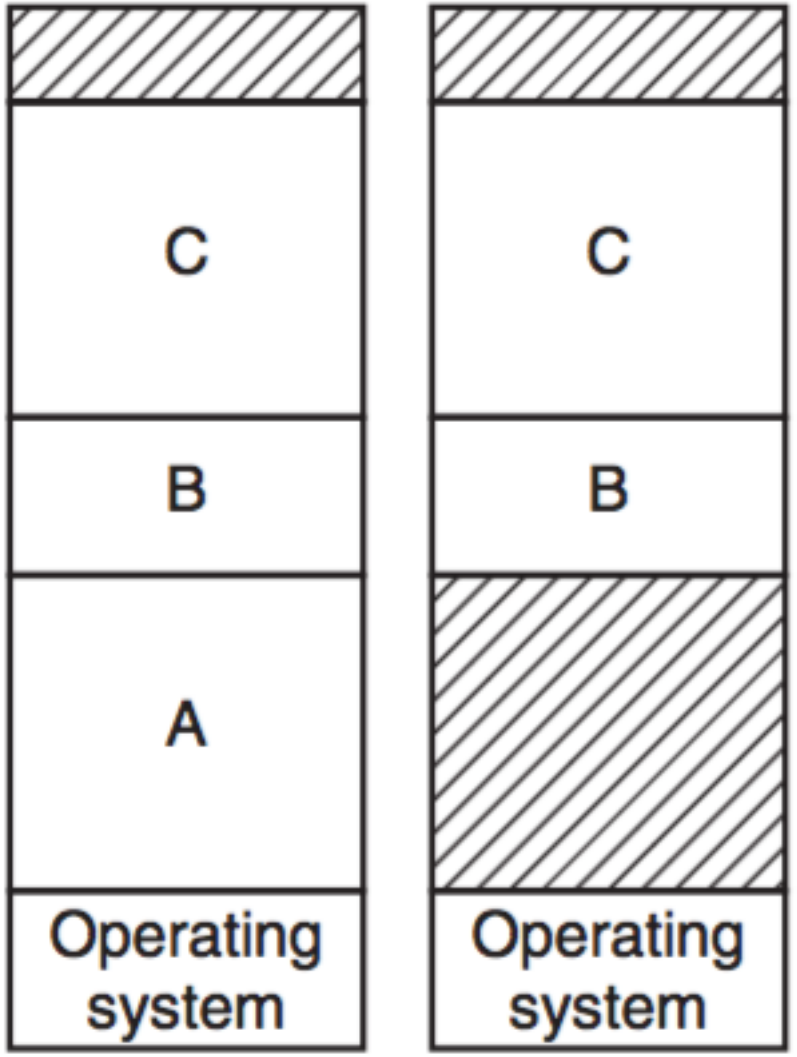
Swapping

- Operação
  - Inicialmente, somente o processo A está na memória
  - Então B ou é criado ou trazido de volta (swap in) à memória



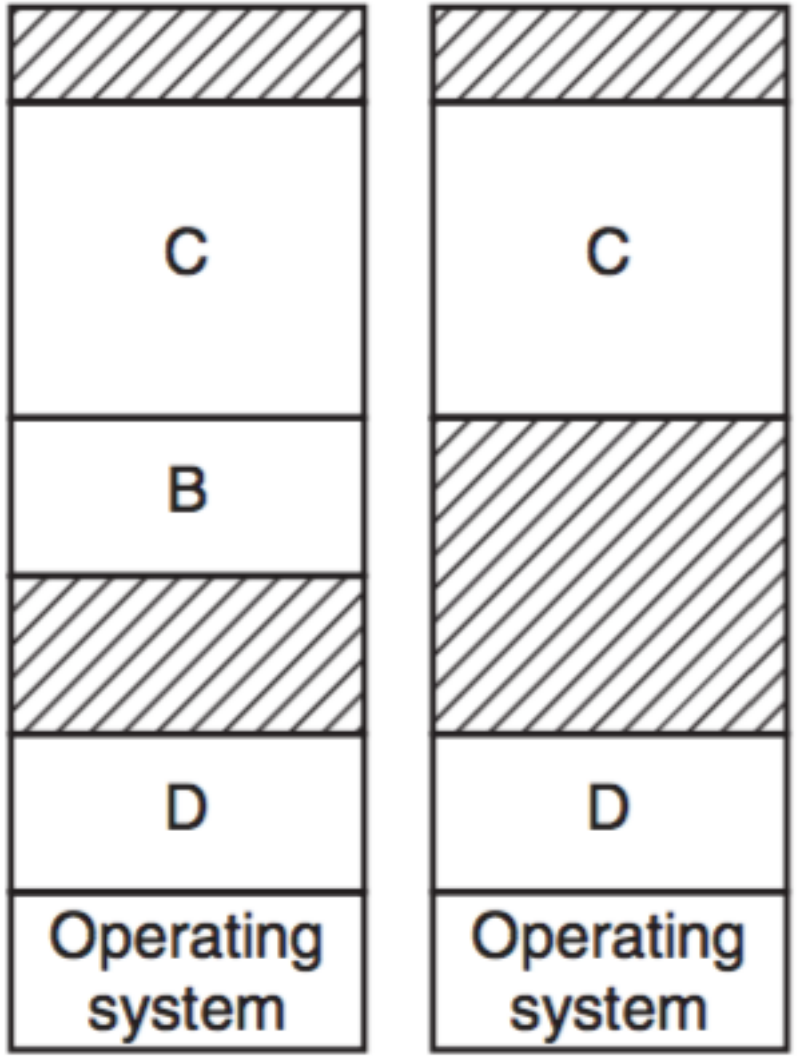
Swapping

- Operação
  - Em seguida, C é criado ou trazido de volta (swap in) à memória
  - A então é levado ao disco (swap out)



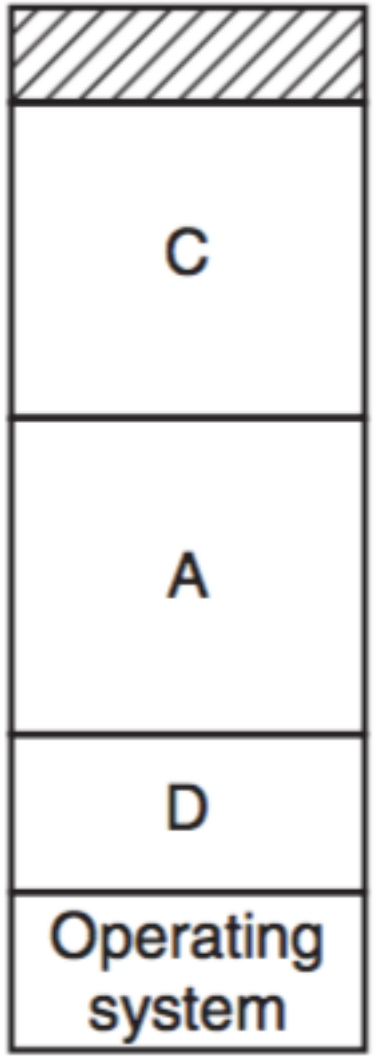
Swapping

- Operação
  - E D é trazido à memória (ou iniciado)
  - B então sai



Swapping

- Operação
  - E A é trazido de volta
- Note que
  - A agora está em outra porção da memória → devemos relocar os endereços
  - As partições são de tamanho variável



Swapping

- Problema
  - A operação de swap pode criar muitos “buracos” na memória
    - Problema de fragmentação externa
- Solução: ???

Swapping

- Problema
  - A operação de swap pode criar muitos “buracos” na memória
    - Problema de fragmentação externa
- Solução: Compactação de memória:
  - Mova todos os processos o mais para baixo possível na memória – haverá um único espaço vazio acima
  - Consome bastante CPU

Swapping – Alocando memória

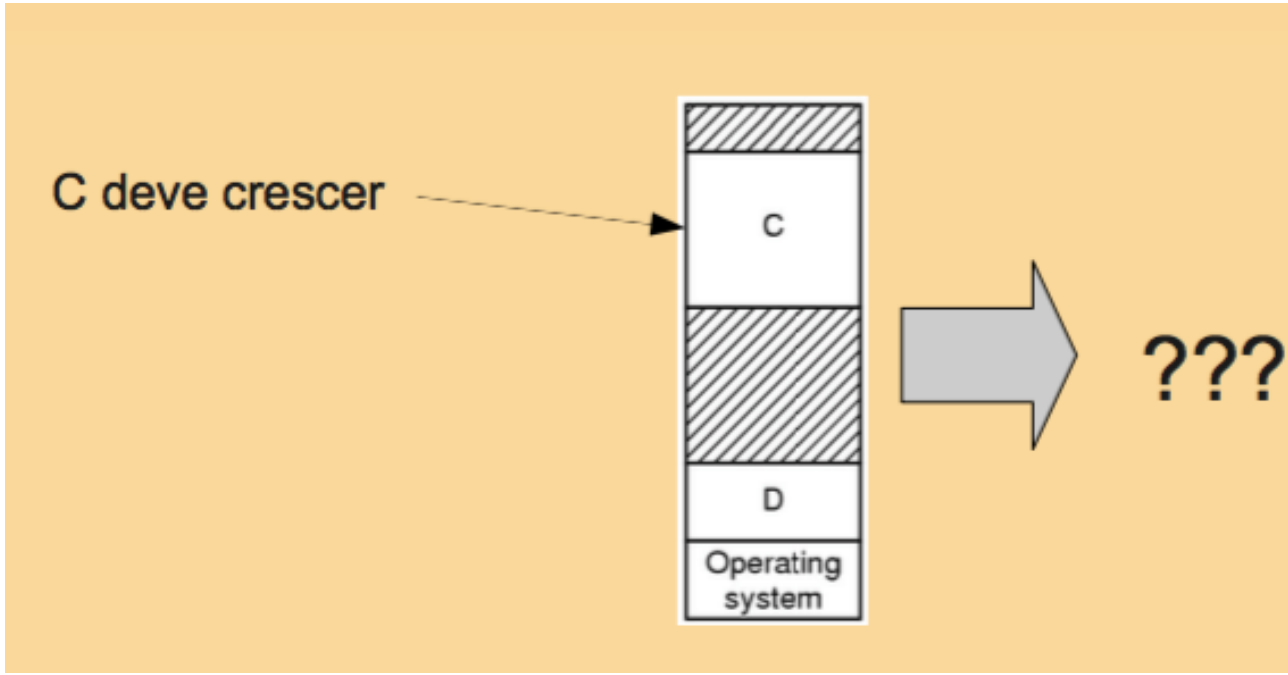
- A alocação de memória muda à medida em que
  - Os processos chegam à memória
  - Os processos deixam a memória
- Quanto de memória devemos alocar a um processo quando ele é criado ou trazido a ela?

Swapping – Alocando memória

- A alocação de memória muda à medida em que
  - Os processos chegam à memória
  - Os processos deixam a memória
- Quanto de memória devemos alocar a um processo quando ele é criado ou trazido a ela?
  - Se eles tiverem tamanho fixo, aloque o que ele precisa

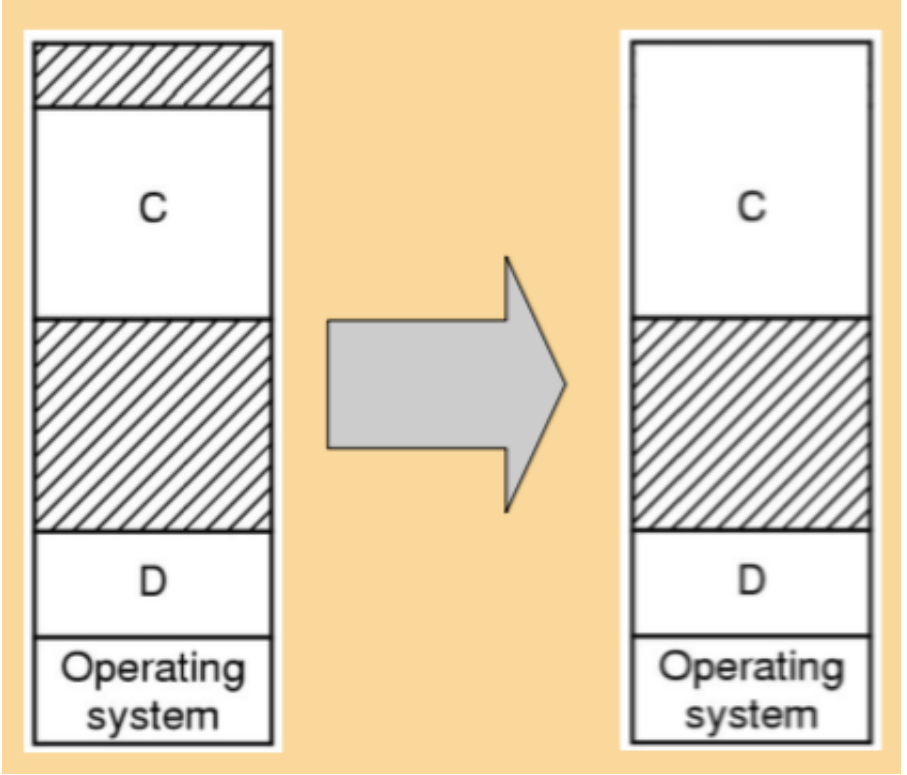
Swapping – Alocando memória

- Quanto de memória devemos alocar a um processo quando ele é criado ou trazido a ela?
  - Mas e se o segmento de dados deles cresce com o tempo?



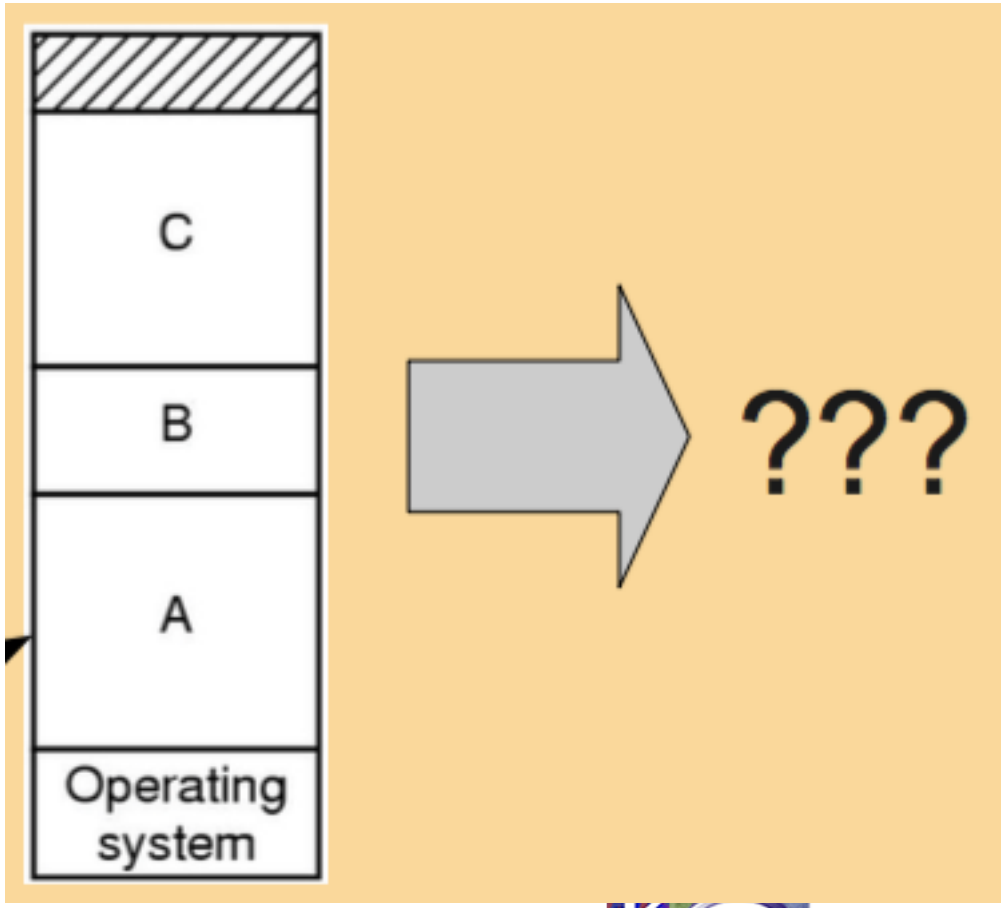
Swapping – Alocando memória

- Quanto de memória devemos alocar a um processo quando ele é criado ou trazido a ela?
  - Mas e se o segmento de dados deles cresce com o tempo?
    - Se houver um “buraco” adjacente à memória atual do processo, ele pode ser alocado, e o processo cresce nesse buraco



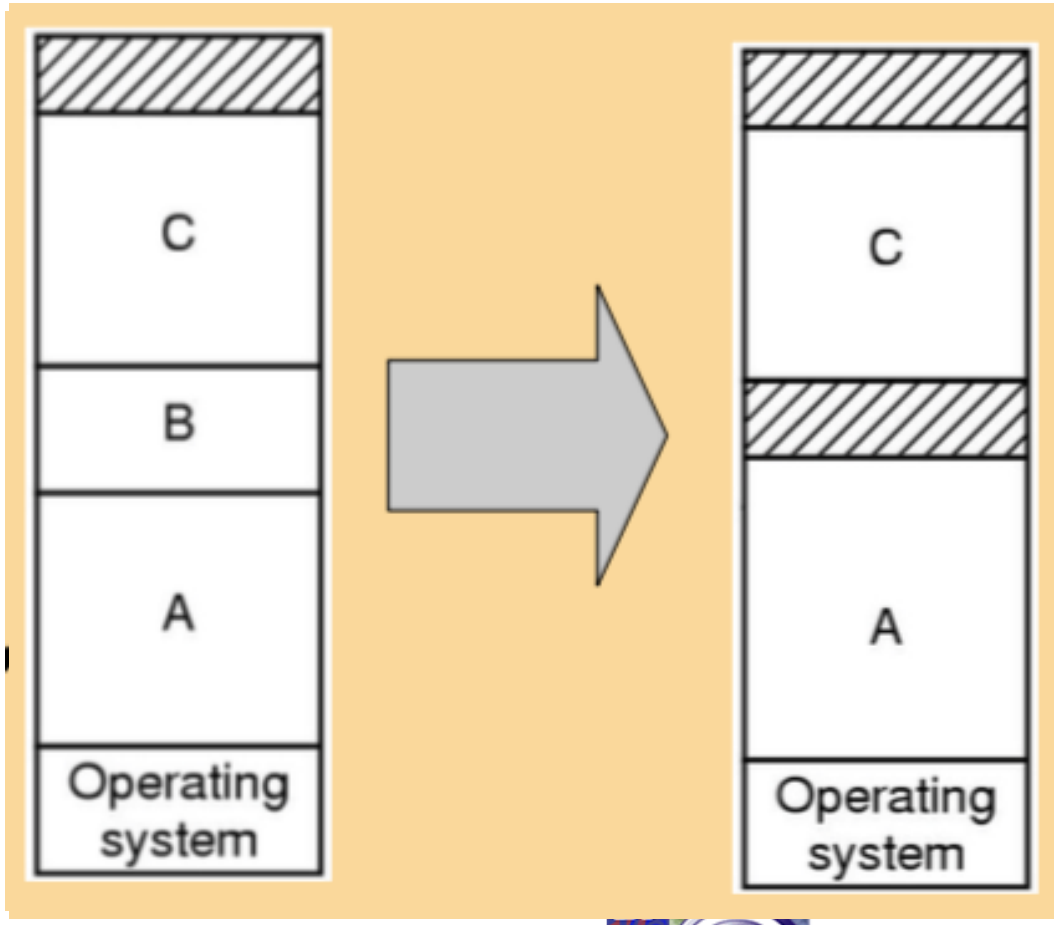
Swapping – Alocando memória

- Quanto de memória devemos alocar a um processo quando ele é criado ou trazido a ela?
  - Mas e se o segmento de dados deles cresce com o tempo?
    - E se ele for adjacente a outro processo?



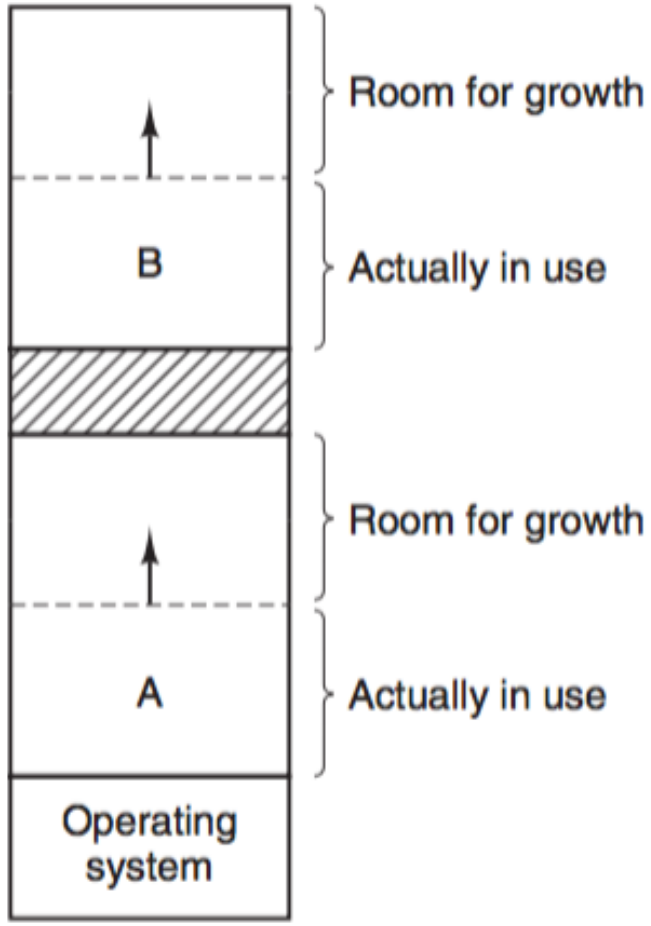
Swapping – Alocando memória

- Quanto de memória devemos alocar a um processo quando ele é criado ou trazido a ela?
  - Mas e se o segmento de dados deles cresce com o tempo?
    - E se ele for adjacente a outro processo?
      - Ou o movemos a um buraco maior na memória
      - Ou um ou mais processos terão que ser removidos para criar esse buraco



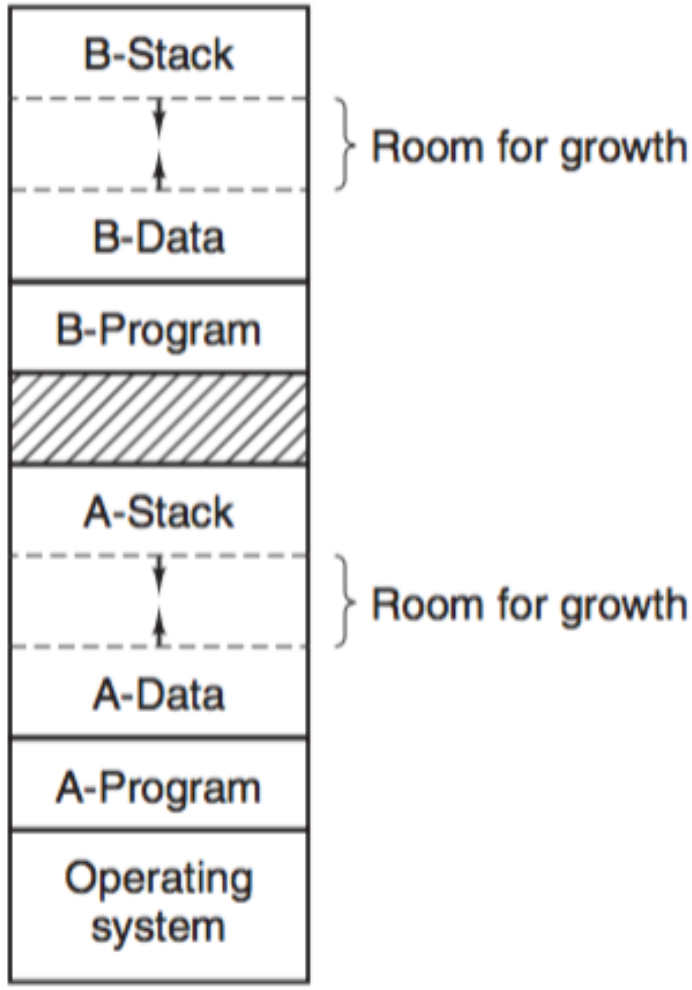
Swapping – Alocando memória

- Quanto de memória devemos alocar a um processo quando ele é criado ou trazido a ela?
  - Alternativamente, podemos alocar memória a mais para cada processo carregado
  - Reduz o overhead de ter que fazer swap
  - Se ainda assim o processo for para o disco, somente a memória realmente em uso é gravada



Swapping – Alocando memória

- Quanto de memória devemos alocar a um processo quando ele é criado ou trazido a ela?
  - Se os processos tiverem dois segmentos que crescem (dados e pilha, por exemplo), podemos usar a mesma ideia
    - Nesse caso, a pilha cresce para baixo enquanto que o segmento de dados cresce para cima
  - Se ainda assim ficar sem espaço
    - Swap, como antes

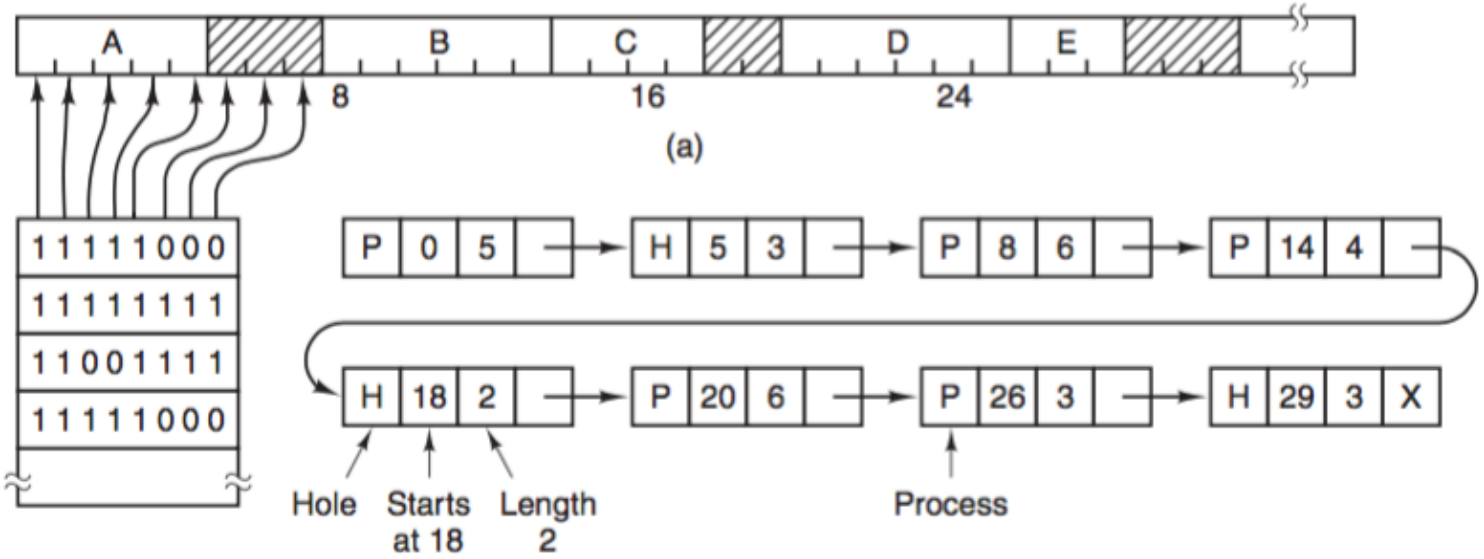


Swapping
Técnicas para alocação dinâmica de memória

- Mapas de Bits (Bitmaps):
  - Memória é dividida em unidades de alocação
    - Pode conter vários KB
  - Cada unidade corresponde a um bit no bitmap:
    - 0 → livre
    - 1 → ocupado
  - Tamanho do bitmap depende do tamanho da unidade e do tamanho da memória;
    - unidades de alocação pequenas → bitmap grande;
    - unidades de alocação grandes → perda de espaço;

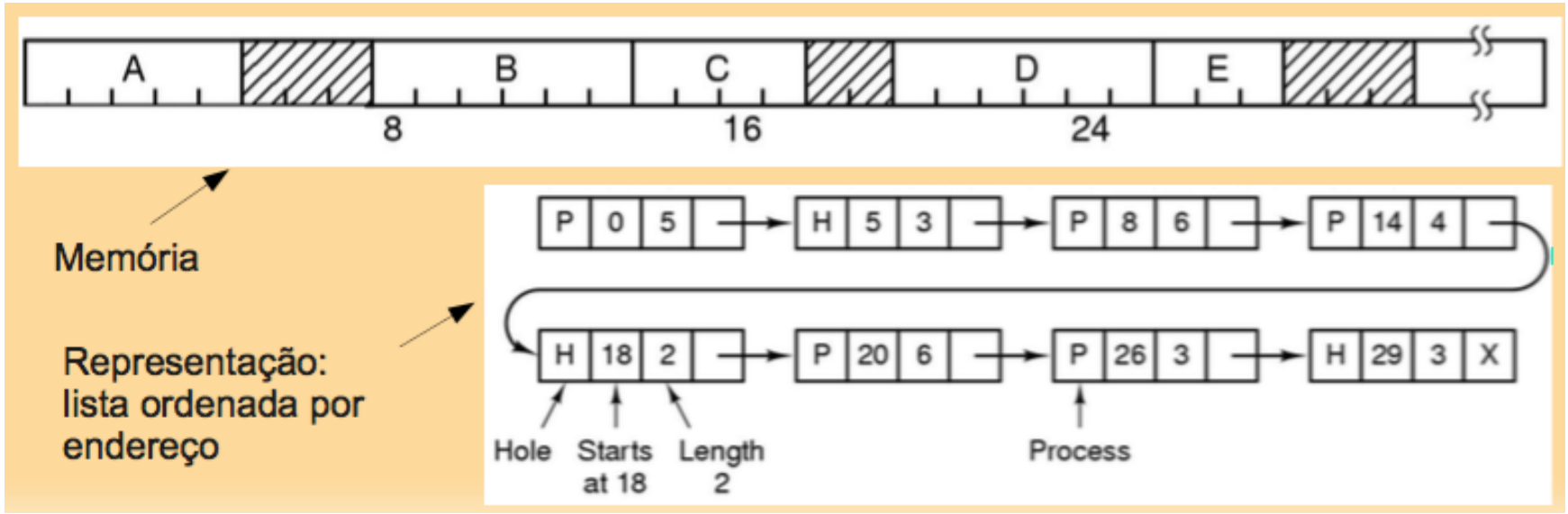
Swapping
Técnicas para alocação dinâmica de memória

- Mapas de Bits (Bitmaps):
  - Problema
    - Quando um novo processo (de k unidades) é trazido à memória, o gerenciador deve buscar, no bitmap, uma seqüência de k zeros consecutivos – operação potencialmente lenta

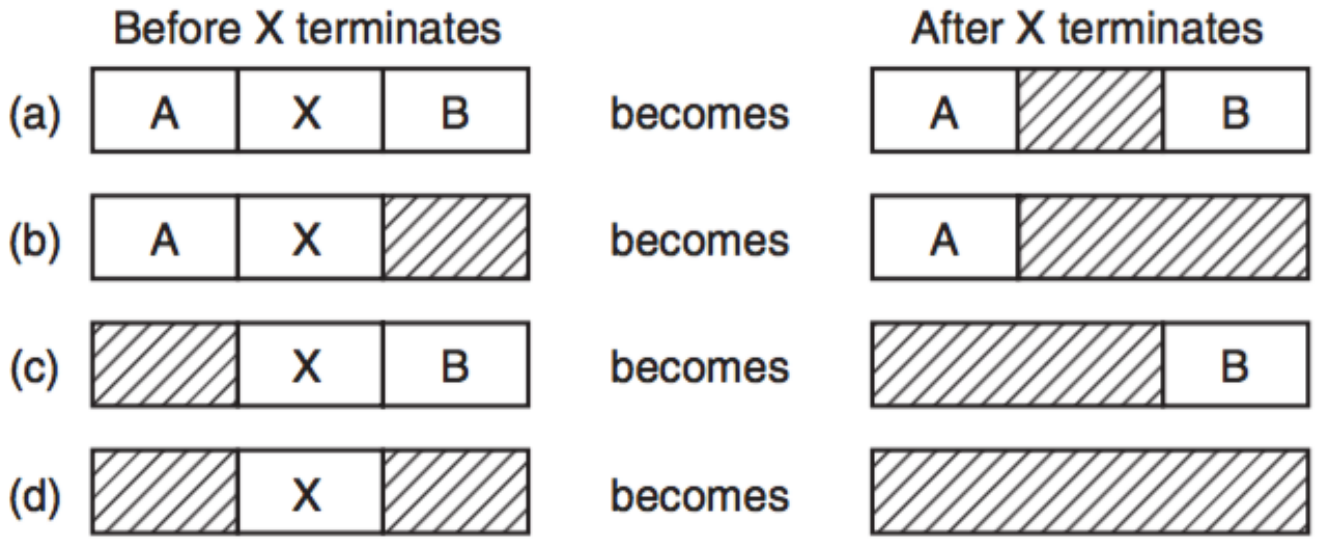


Swapping
Técnicas para alocação dinâmica de memória

- Lista Ligada
  - Manter uma lista ligada de segmentos de memória livres e alocados
  - Cada segmento ou contém um processo ou é um buraco vazio entre dois processos



- Listas Ligadas
  - A lista ordenada por endereço fica fácil de ser atualizada



Troque P por H na posição correspondende da lista

Elimine o nó que representadva X, atualizando o seguinte e o anterior

Swapping
Técnicas para alocação dinâmica de memória

- Existem três métodos que podem ser usados para selecionar uma região para um processo.
  - Se a lista estiver ordenada por endereço
- Os algoritmos de alocação são:
  - Primeira escolha (First fit):
    - Percorre a lista de segmentos até que encontre um buraco grande o bastante
    - Quebre o buraco em dois pedaços – um para o processo e um para a memória não usada
    - Variação: inicie a busca a partir de onde parou, na vez anterior (next fit)

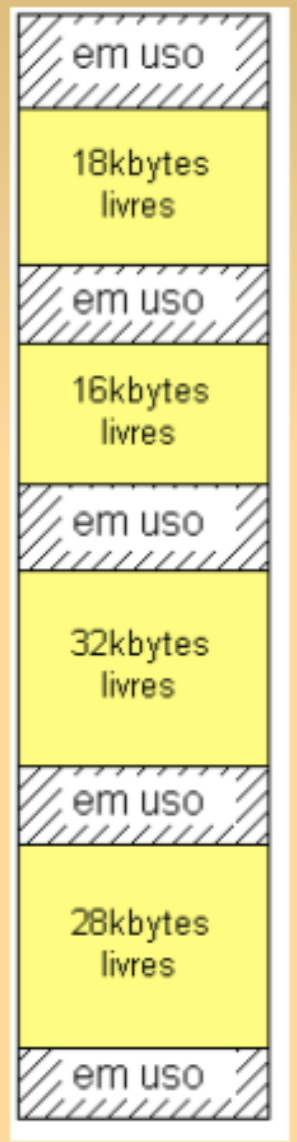
Swapping
Técnicas para alocação dinâmica de memória

- Os algoritmos de alocação são:
- Melhor escolha (Best fit):
  - Busca a lista inteira, até o fim, e toma o menor buraco que seja adequado
  - Não quebra buracos grandes que poderiam ser úteis mais tarde
  - Problema: permite o surgimento de buracos minúsculos
- Pior escolha (worst fit):
  - Sempre tome o maior buraco disponível
  - Reduz a chance de buracos minúsculos e inúteis

Dois processos, **P** e **Q**, são alocados nessa ordem:

14kbytes (P)

20kbytes (Q)



Sentido em que a memória é percorrida

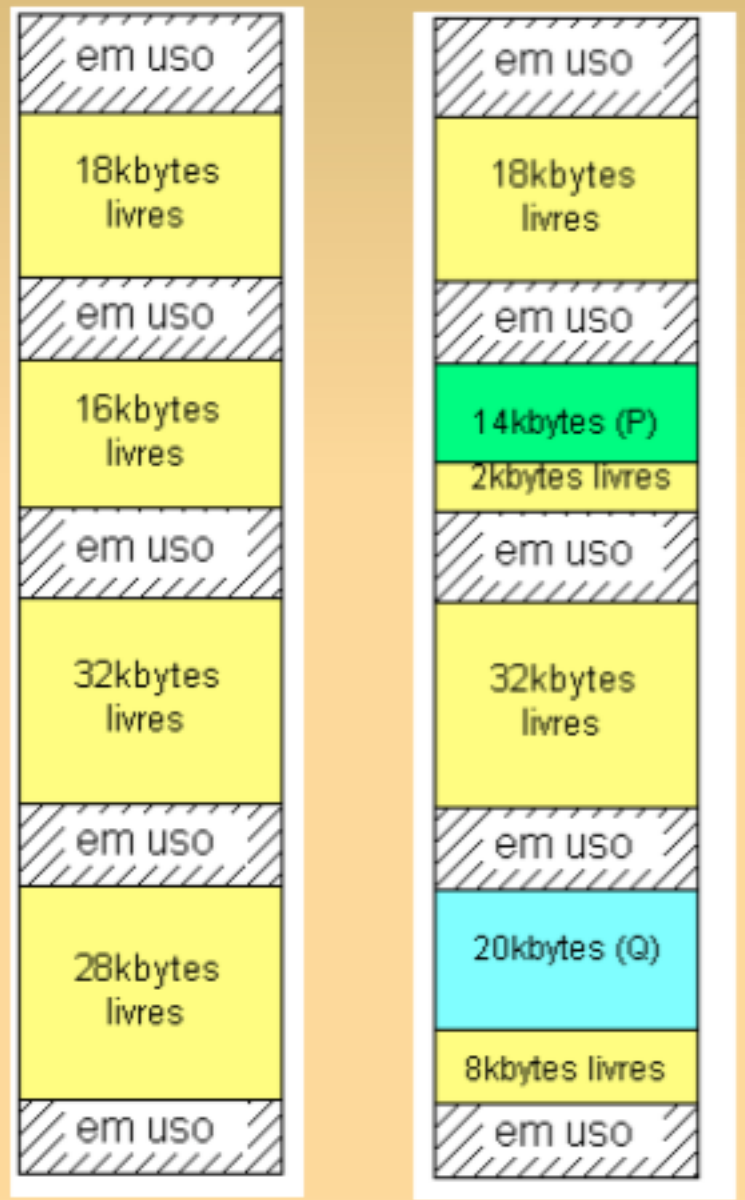
Áreas livres iniciais **Melhor Escolha** **Pior Escolha** **Primeira Escolha**

Dois processos, **P** e **Q**, são alocados nessa ordem:

14kbytes (P)

20kbytes (Q)

Sentido em que a memória é percorrida



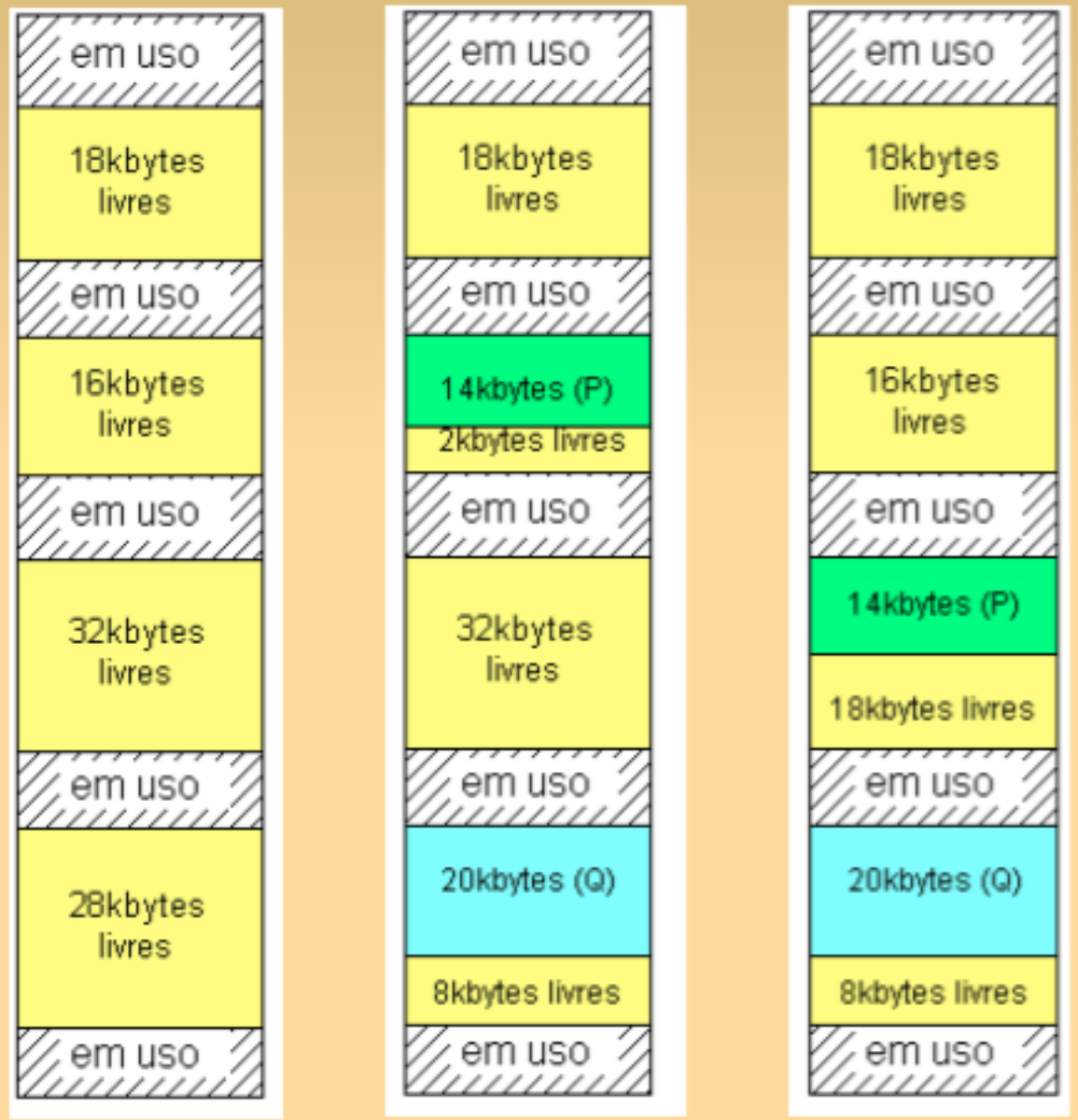
Áreas livres iniciais **Melhor Escolha** **Pior Escolha** **Primeira Escolha**

Dois processos, **P** e **Q**, são alocados nessa ordem:

14kbytes (P)

20kbytes (Q)

Sentido em que a memória é percorrida

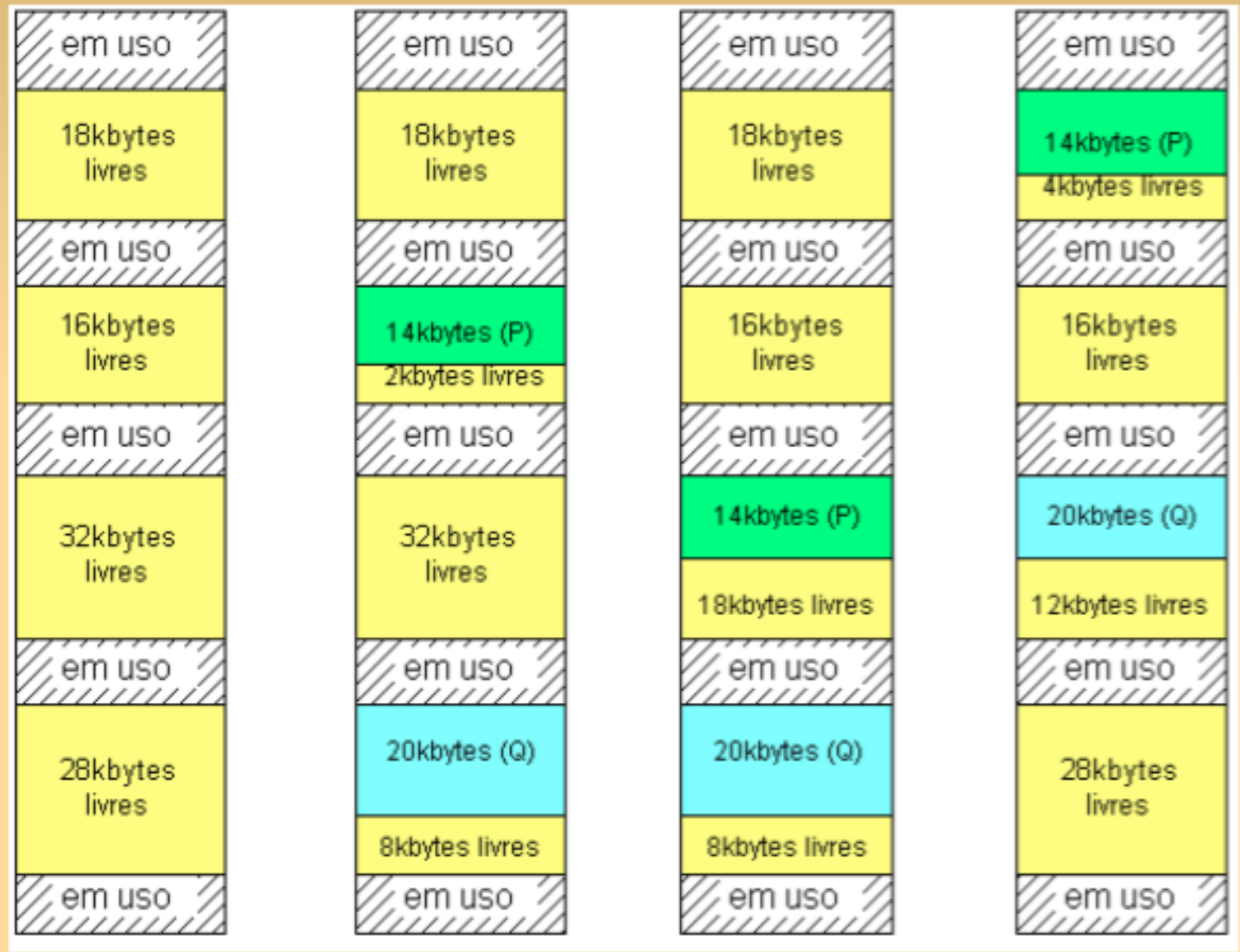


Dois processos, **P** e **Q**, são alocados nessa ordem:

14kbytes (P)

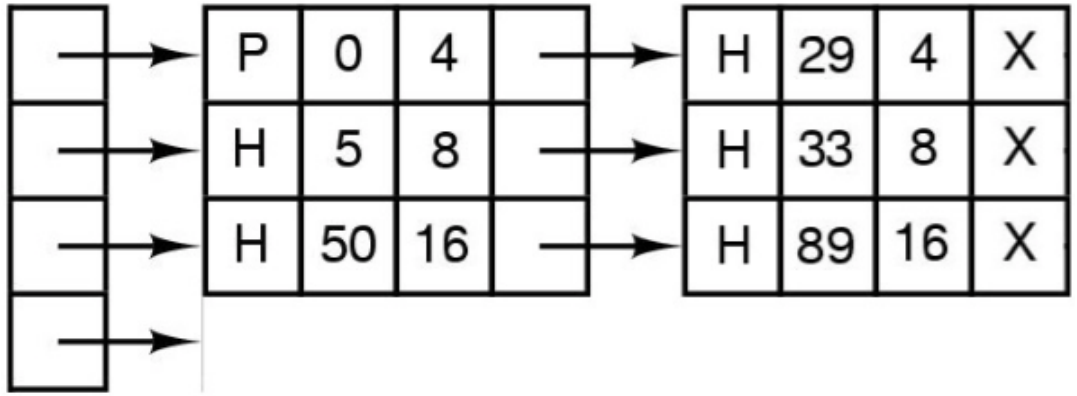
20kbytes (Q)

Sentido em que a memória é percorrida



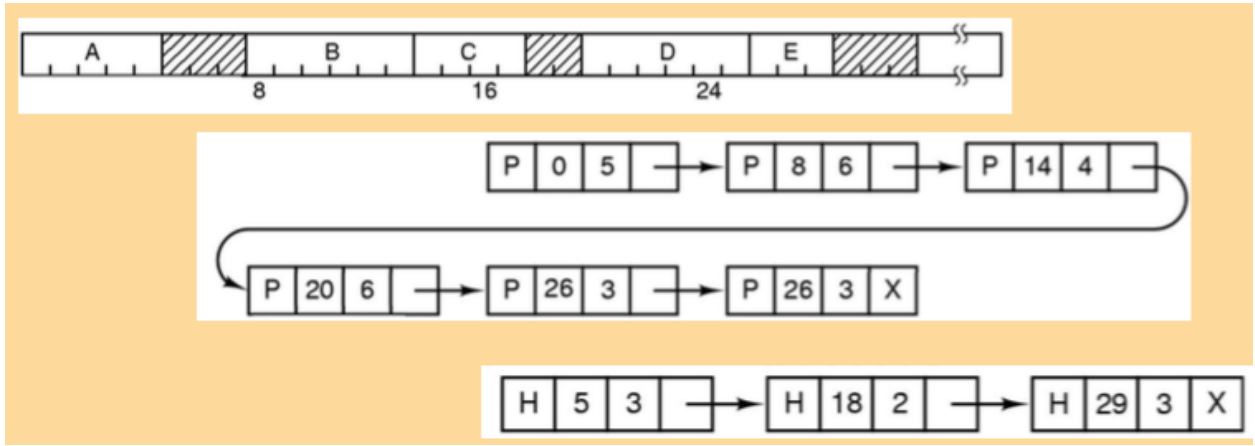
Swapping
Técnicas para alocação dinâmica de memória

- Os algoritmos de alocação são:
  - Quick fit
    - Mantém listas separadas para alguns dos tamanhos mais comumente requisitados
      - Uma para cada tamanho
    - Problema quando processo é desalocado:
      - Encontrar os vizinhos ao buraco que ele deixou, para união



Swapping
Técnicas para alocação dinâmica de memória

- Algoritmos de alocação
  - Todos podem ser melhorados se mantivermos buracos e processos em listas separadas
    - Há ganho de desempenho
    - Ainda assim, quando um processo é retirado da memória, mesclar buracos é de alta complexidade.



Swapping
Técnicas para alocação dinâmica de memória

- Principais Conseqüências dos algoritmos:
  - Melhor escolha: deixa o menor resto, porém após um longo processamento poderá deixar “buracos” muito pequenos para serem úteis.
  - Pior escolha: deixa o maior espaço após cada alocação, mas tende a espalhar as porções não utilizadas sobre áreas não contínuas de memória e, portanto, pode tornar difícil alocar grandes jobs.
  - Primeira escolha: tende a ser um meio termo entre a melhor e a pior escolha, com a característica adicional de fazer com que os espaços vazios migrem para o final da memória.

Memória virtual

- O tamanho da memória cresce rapidamente
  - O tamanho do software cresce muito mais rápido
  - Há necessidade de rodar programas grandes demais para a memória
- Programas fazem referência à memória
  - Ex:
    MOV REG,1000
  - Não necessariamente esse é o endereço real.

Memória virtual

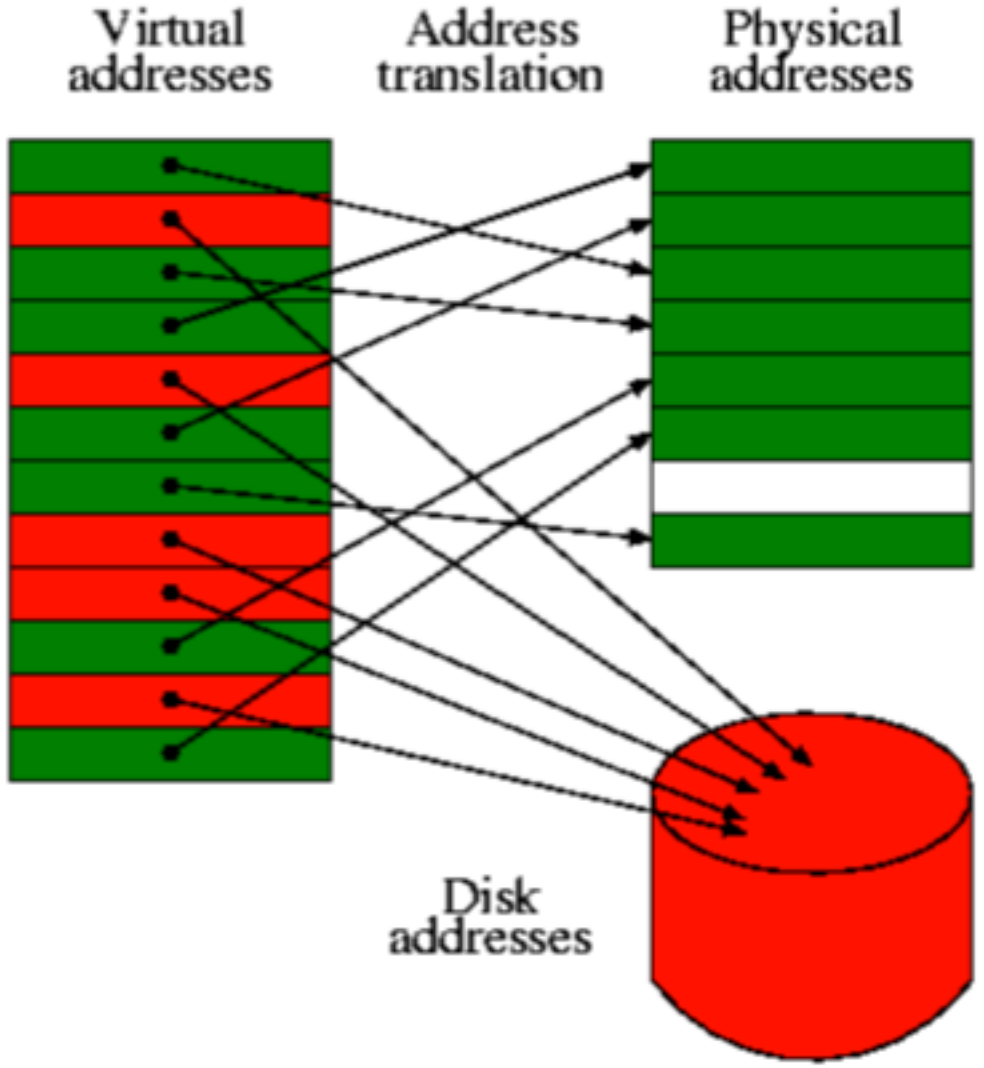
- Os endereços físicos (reais) podem ser gerados de muitas maneiras:
  - Mapeamento direto
  - Usando registradores base
  - etc
- Endereços referenciados pelos programas são chamados de virtuais
  - Na ausência de memória virtual, o endereço virtual bate com o físico
  - Com memória virtual, é tratado pela MMU

Memória virtual

- Espaço de Endereçamento Físico de um processo:
  - Formado por todos os endereços físicos/reais aceitos pela memória principal (RAM);
- Espaço de Endereçamento Virtual de um processo:
  - Formado por todos os endereços virtuais que esse processo pode gerar;

Memória virtual

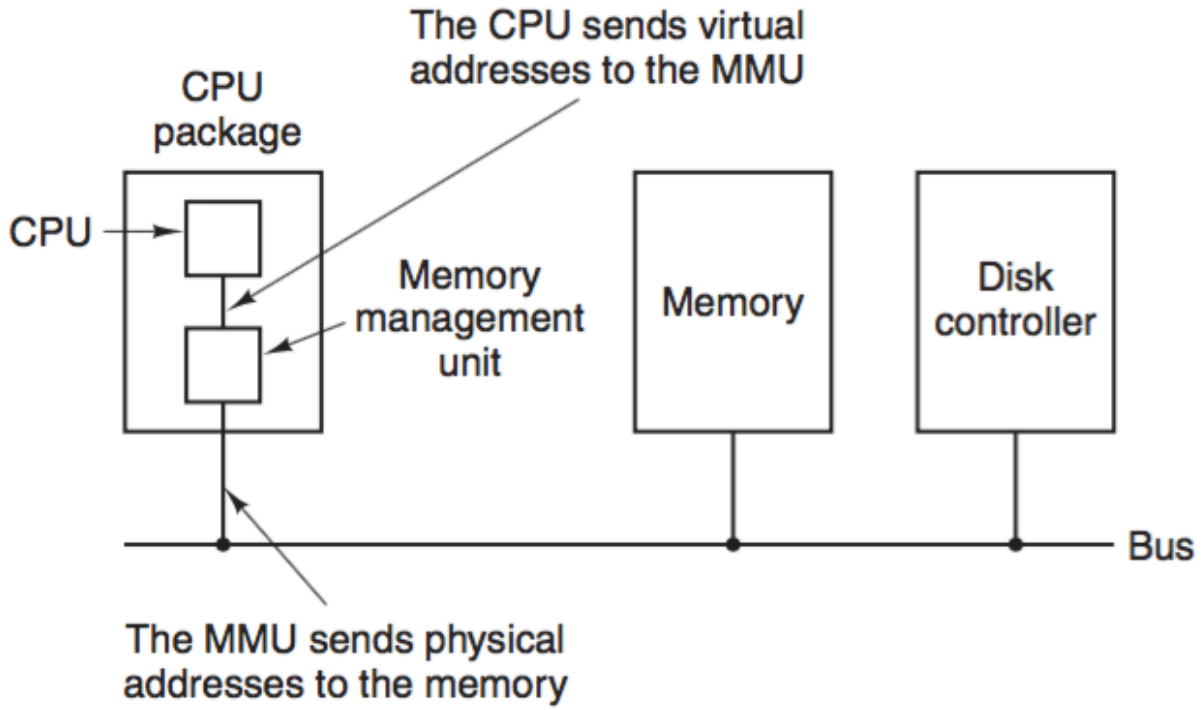
- Um processo em Memória Virtual faz referência a endereços virtuais e não a endereço reais de memória RAM;
- No momento da execução de uma instrução, o endereço virtual é traduzido para um endereço real, pois a CPU manipula apenas endereços reais da memória RAM → MAPEAMENTO;



Universidade Federal de Mato Grosso
Campus Rondonópolis

Memória virtual

- A MMU mapeia o endereço virtual à memória física



Memória virtual

- Memória virtual é então uma técnica para troca de processos na memória
  - Programas maiores que a memória eram divididos em pedaços menores chamados overlays
    - Solução adotada nos anos 60
- Quando um programa iniciava, tudo que é carregado na memória é o gerenciador de overlay
    - Imediatamente carrega e roda o overlay 0
  - Vantagem: expansão da memória principal;
  - Desvantagem: custo muito alto;

Memória virtual

- Memória virtual é uma técnica para troca de processos na memória
  - Sistema operacional era responsável por:
    - Realizar o chaveamento desses pedaços entre a memória principal e o disco;
  - Cabia ao programador dividir o programa em overlays;
- Memória Virtual automatiza a divisão em overlays
  - Cada programa tem seu próprio espaço de endereços, que é quebrado em porções → as páginas

Memória virtual

- Páginas
  - Quando o programa referencia uma parte de seu endereço que está na memória, o hardware faz o mapeamento na hora
  - Se, contudo, essa parte não estiver na memória:
    - O S.O. traz do disco o pedaço que falta
    - Executa novamente a instrução que falhou
    - Enquanto isso, o escalonador pode colocar outro processo para rodar
- Memória virtual é semelhante ao swap
  - Swap aloca espaço para todo o processo, MV não

Memória virtual

- Técnicas de MV:
  - Paginação:
    - Blocos de tamanho fixo (endereços contínuos) chamados de páginas;
    - SO mantém uma fila de todas as páginas;
    - Endereços Virtuais formam o espaço de endereçamento virtual;
    - O espaço de endereçamento virtual é dividido em páginas virtuais;
    - Mapeamento entre endereços reais e virtuais (MMU);

Memória virtual

- Técnicas de MV
  - Segmentação:
  - Blocos de tamanho arbitrário chamados segmentos;
- Arquitetura (hardware) tem que possibilitar a implementação tanto da paginação quanto da segmentação;