Sistemas Operacionais

Gerenciamento de memória

Capítulos 7
Operating Systems: Internals and Design Principles
W. Stallings

O problema

- Em um ambiente multiprogramado, é necessário:
  - subdividir a memória para acomodar múltiplos processos
  - mas se poucos processos estão na memória, em boa parte do tempo estarão esperando por E/S
    - UCP sub-utilizada
  - então, deve-se alocar memória de forma eficiente ao maior número de processos

Gerenciador de Memória

Alguns requisitos do GM

Relocação

- o programador não deve se preocupar com o local onde o programa (processo) será carregado para execução
- durante a execução, o processo poderá sair da memória e retornar para um local diferente
- referências devem ser resolvidas para endereços de memória física
- p. ex. - bloqueado para suspenso

Alguns requisitos do GM

Proteção

- processos não devem poder referenciar posições de memória em outros processos sem permissão
- em virtude da relocação, não é possível testar endereços em programas
  - com suporte de *h/w*, o teste deverá ser em tempo de execução

Alguns requisitos do GM

Compartilhamento

- deve-se permitir que vários processos possam acessar a mesma porção de memória
- o mecanismo de proteção deve ter flexibilidade
  - caso por exemplo, exclusão mútua

Alguns requisitos do GM

Organização lógica

- programas são normalmente separados em módulos, que podem ser escritos e compilados separadamente
- graus diferentes de proteção podem ser atribuídos aos módulos
- compartilhamento de módulos
- manipulação de diferentes módulos de um mesmo executável pode ser melhor realizada através de segmentação

Alguns requisitos do GM

Organização física

- memória é organizada como uma hierarquia
- se um programa precisa de mais memória do que o disponível na MP, a MS deverá ser utilizada
- uso de memória *cache*
- este gerenciamento deverá ser feito de forma transparente pelo SO

Particionamento fixo

Particionamento da memória em regiões fixas

- Se partições idênticas
  - processos menores que o tamanho da partição podem ser carregados diretamente.
  - Processos maiores exigirão *overlay*.
    - programa dividido em modulos que compartilham uma mesma partição
    - modularização geralmente responsabilidade do programador

Particionamento fixo

Particionamento da memória em regiões fixas

- Se processos menores
  - fragmentação interna
  - desperdício de MP

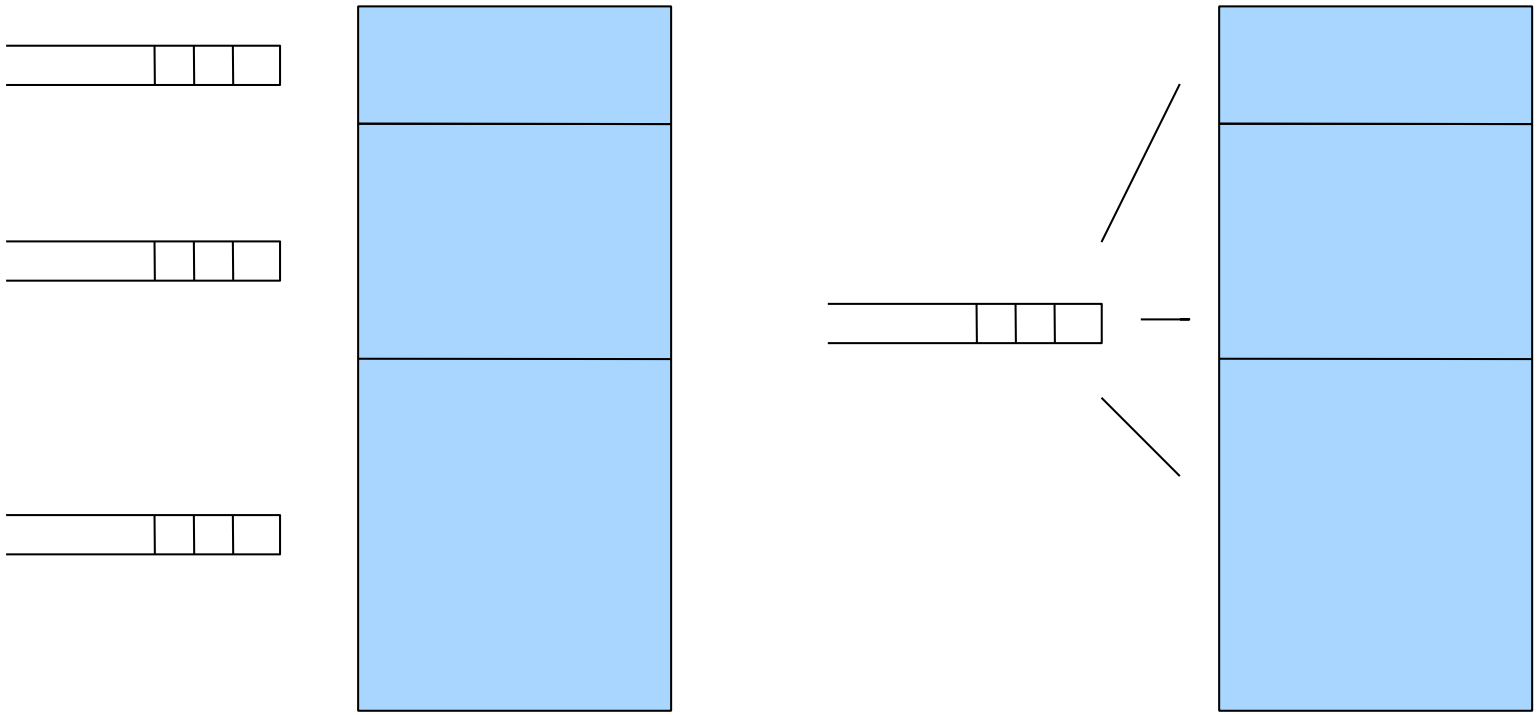
Particionamento fixo

Partições de tamanhos distintos

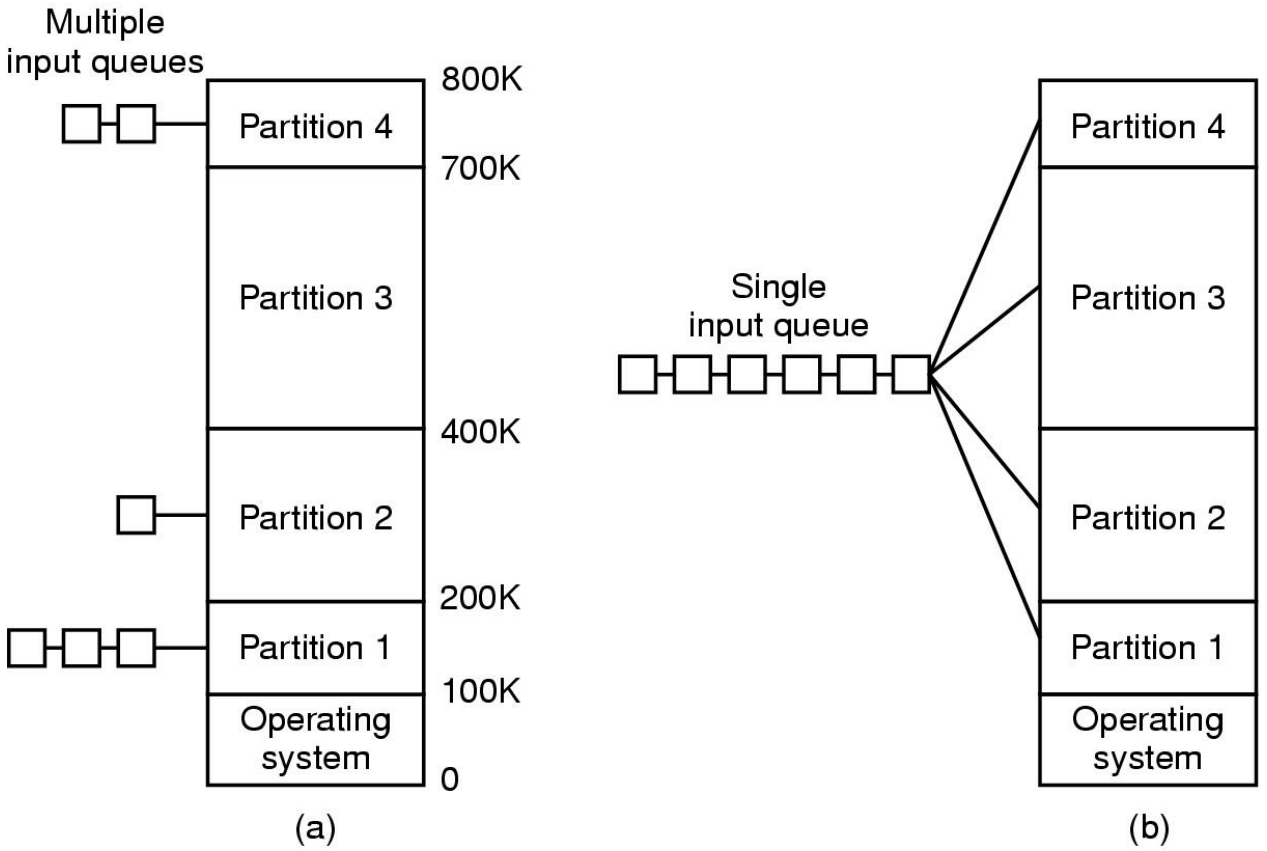
- diminuem a ineficiência (não a elimina)
- associa o processo à partição menor possível
- aumenta a sobrecarga do gerenciamento (e.g., uma fila por partição ou fila única?)

- De qualquer forma:
  - o número de partições determina o número de processos no sistema
  - processos pequenos utilizam de maneira ineficiente a memória
  - relocação

Particionamento fixo – tamanho variável

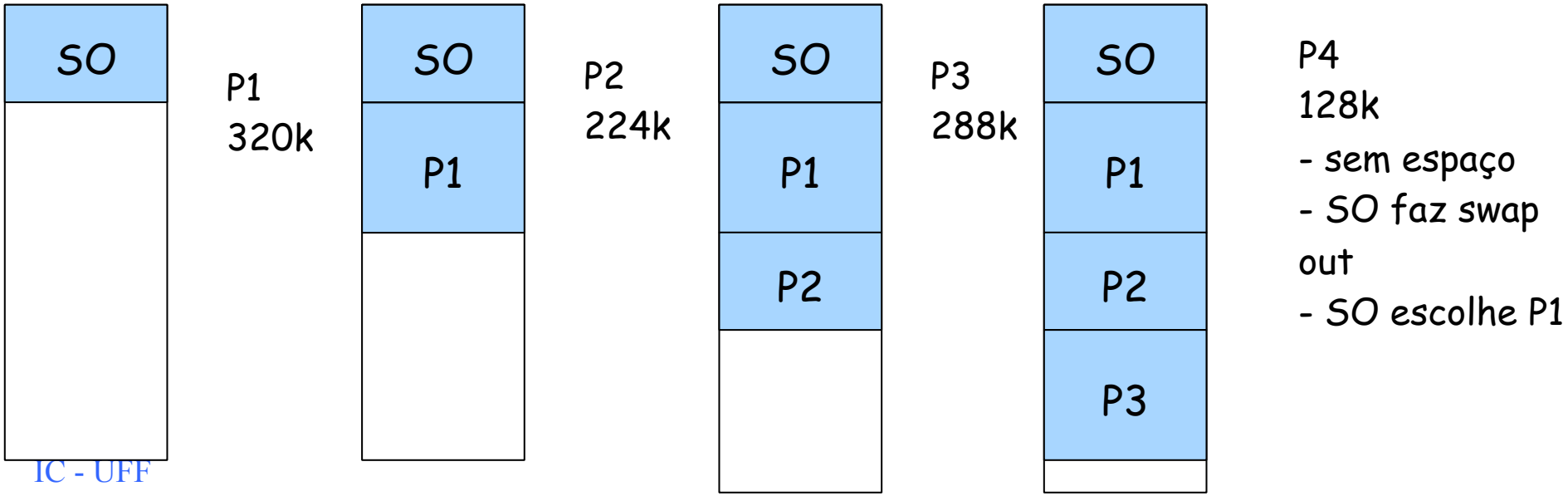


Particionamento fixo – tamanho variável

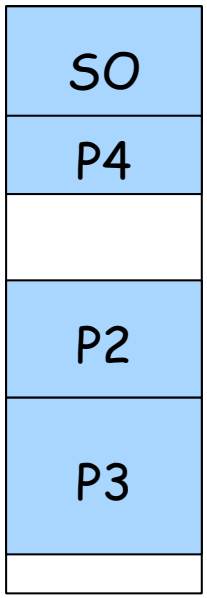


Particionamento dinâmico

- Número e tamanho das partições é variável
- Cada processo recebe a quantidade de memória que necessita
- Gerenciando buracos e processos

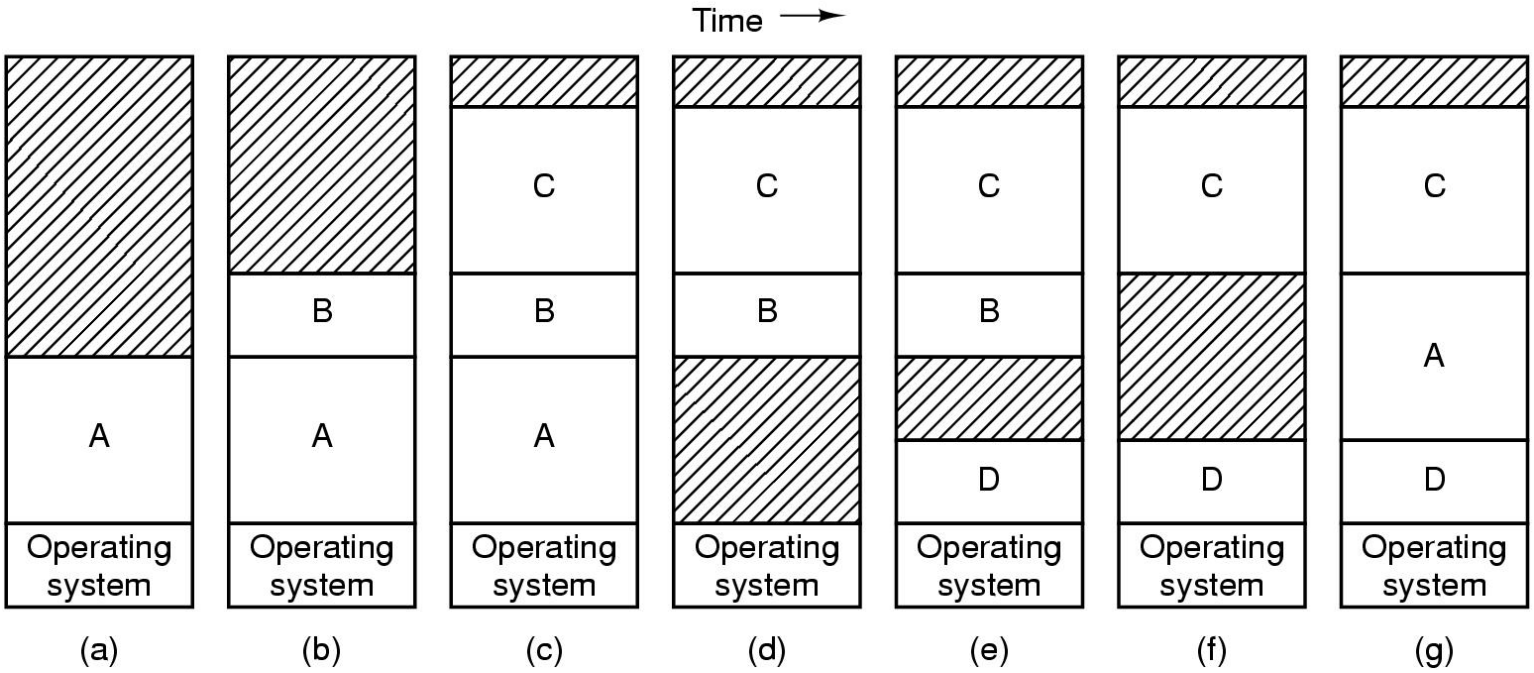


Particionamento dinâmico



- buracos começam a ser formados
- tamanhos tentem a ser pequenos
- necessidade de rearrumar os espaços

Particionamento dinâmico



Particionamento dinâmico

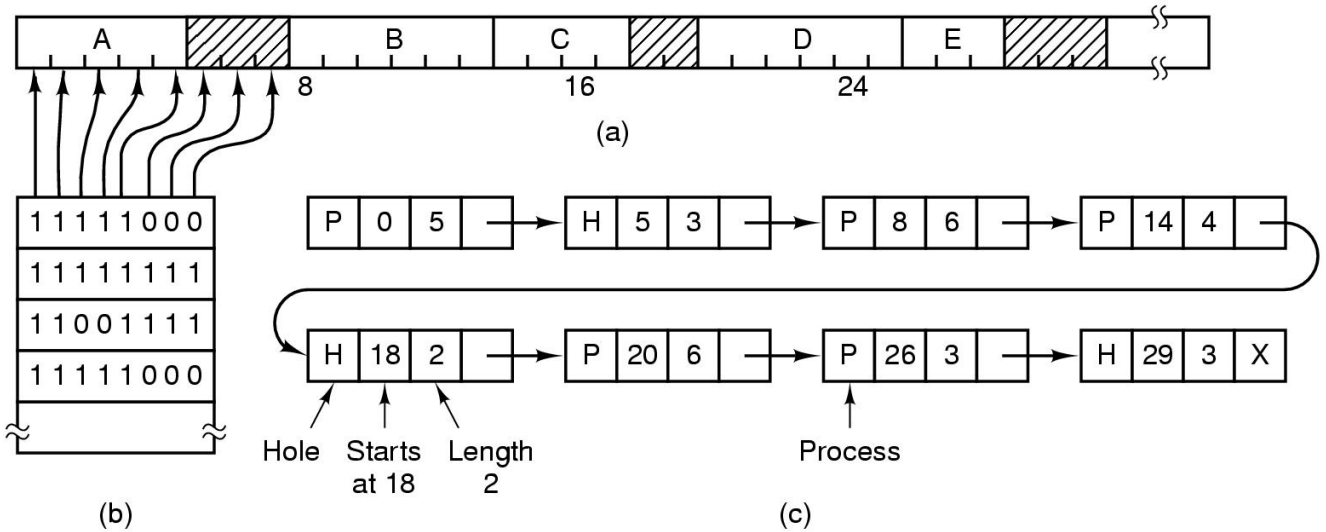
Políticas de alocação

- primeiro encaixe (o melhor) - *first fit*
  - seleciona o primeiro que encaixa, a partir do início
  - mais simples, mais rápido e melhor na maioria das vezes
- próximo encaixe (um pouco pior) – *next fit*
  - seleciona o primeiro que encaixa, a partir da última partição selecionada
  - usualmente seleciona para o final da MP
- melhor encaixe (o pior!) - *best fit*
  - de todas as partições, seleciona aquela imediatamente maior
  - mais custoso, maior grau de fragmentação externa

Particionamento dinâmico

- Buracos na memória: fragmentação externa
- Compactação: tempo perdido e relocação dinâmica (melhoria com *swapping*)
- Sobrecarga maior que método fixo
- Em qualquer caso, relocação
- Então, o que fazer?

Gerenciando buracos e processos



- Mapas de bits (b): simples; ineficiência
- Listas encadeadas (c)
  - no caso de particionamento dinamico, minimizar fragmentação externa pode ser custoso
- *Buddy System*

Sistema "Buddy"

- O espaço disponível total da memória é tratado como um único bloco de 2^U
- Se um pedido de tamanho s é tal que

$$2^{U-1} < s \leq 2^U,$$

  - aloca todo o bloco de 2^U
- Senão divide o bloco em dois 2 "buddies" iguais de 2^{U-1} cada e aloca um bloco se $2^{U-2} < s \leq 2^{U-1}$
  - A divisão continua até que o bloco encontrado seja > ou = ao s requerido.

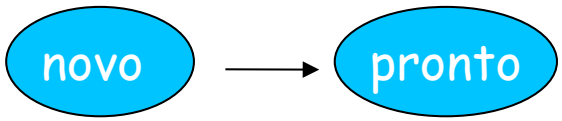
Sistema "Buddy"

- É uma tentativa de diminuir e concatenação e realocação
  - diminui fragmentação externa
  - mas pode aumentar a fragmentação interna

Realocação

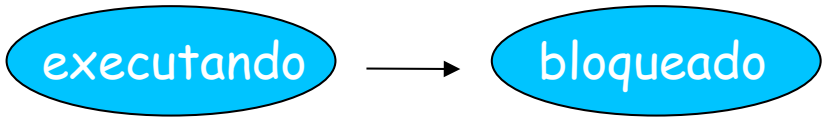
- Mapeamento de endereços virtuais em reais
- necessário, pois processos são alocados em espaço de MP dinamicamente

Ex.: processo P1



P1 em end. de MP 500

P1:



P1 passa para suspenso

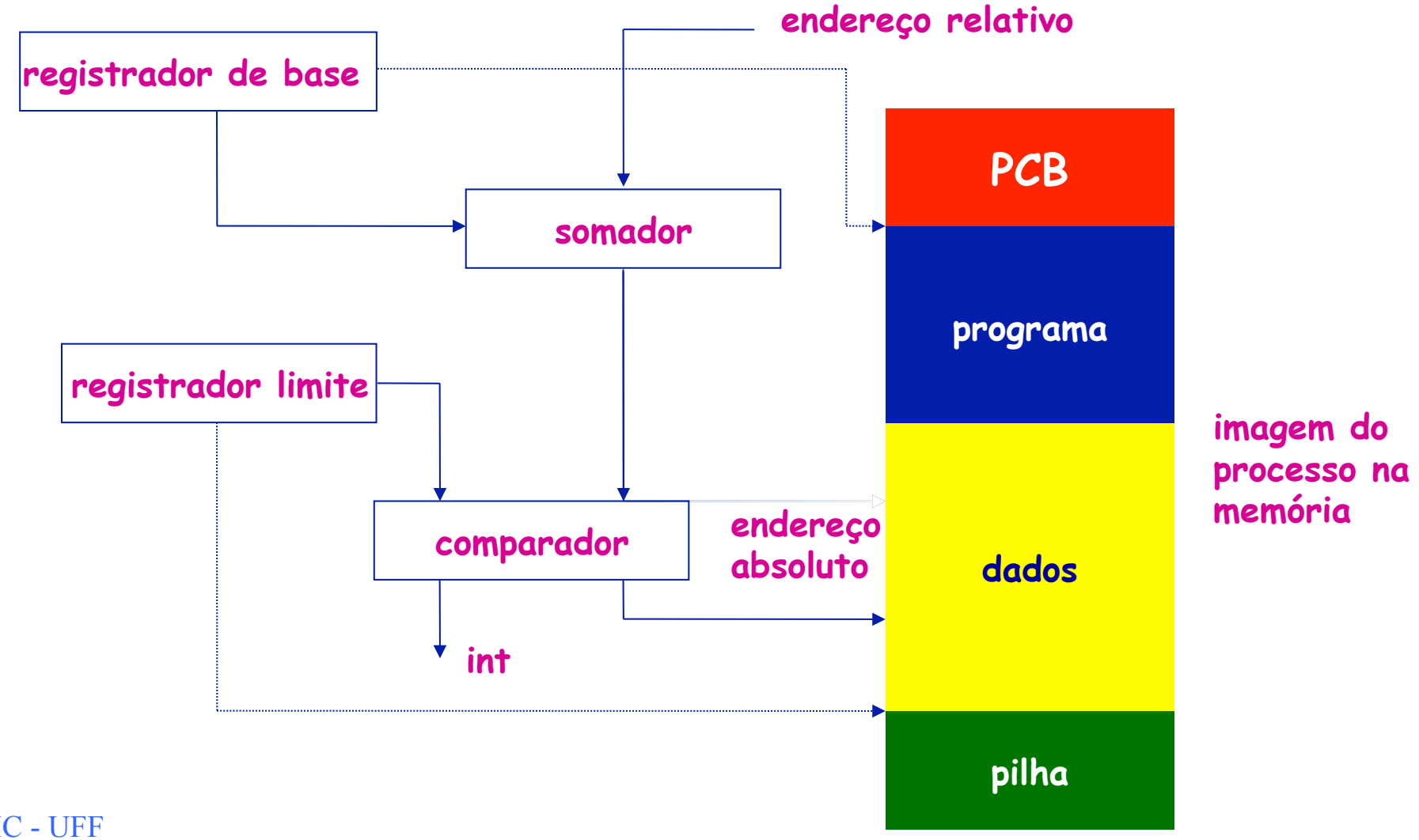
Ao voltar para MP → P1 vai para end. 1024

Realocação

- Mapeamento eficiente
  - endereço físico só calculado quando acesso a MP
  - endereços definidos: lógico, relativo, físico
- Registradores:
  - base – armazena o endereço inicial de MP do processo (quando o processo passa para executando)
  - limite – armazena endereço final do processo
- Acesso ao endereço Z no programa

```
if (Z + base <= limite)
      acesse Z+base
senão "trap"
```

Relocação


endereço relativo
registrador de base
somador
registrador limite
comparador
endereço absoluto
int
PCB
programa
dados
pilha
imagem do processo na memória

Paginação

- Problemas tanto em particionamento fixo quanto dinâmico:
  - fixo – fragmentação interna
  - dinâmico – fragmentação externa e realocação dinâmica
  - buddy – pode ter sua sobrecarga também
- Solução:
  - Processo é dividido em páginas (blocos de processos)
  - MP é dividida em quadros de mesmo tamanho

Paginação

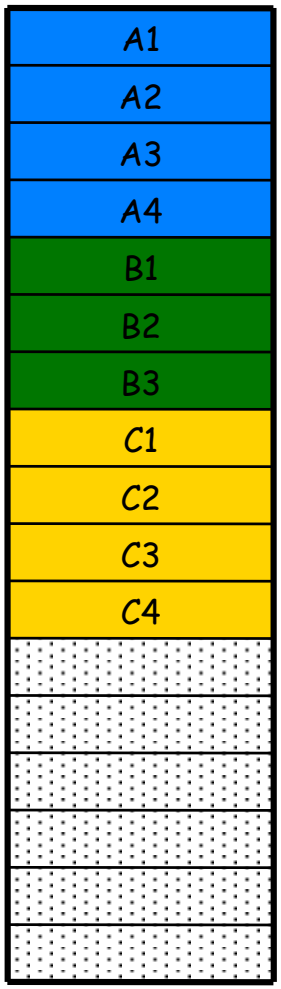
- As páginas do processo não precisam estar em quadros seguidos da mamória principal
- Páginas/quadros são de pequeno tamanho (e.g., 1K):
  - Fragmentação interna pequena
  - Elimina fragmentação externa
- No entanto, lembrando sobre realocação:
  - SO mantém uma tabela de páginas por processo

Paginação

Processos A, B, C estão prontos

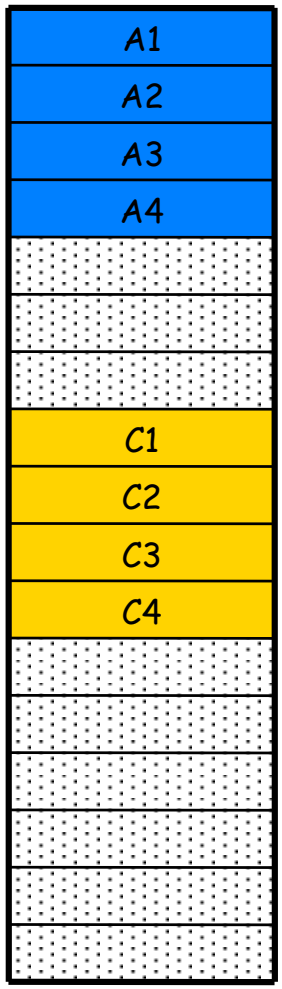
Exemplo: número de páginas/processo

A	4
B	3
C	4
D	5

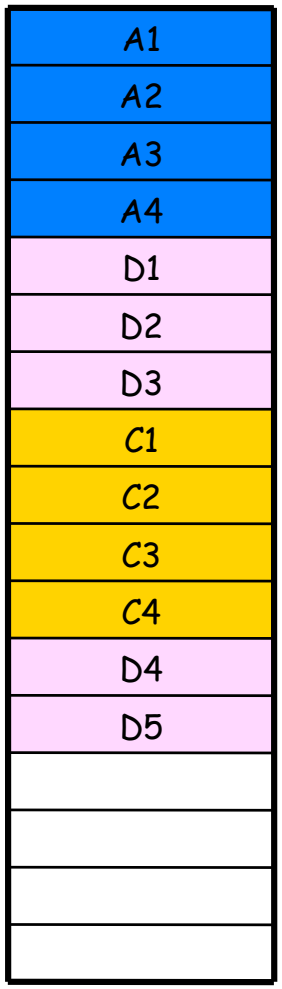


Paginação

B termina



D é submetido



Paginação

- Processo ainda é carregado completamente para a MP
  - no caso de Memória Virtual, não ocorre necessariamente)
- Processo não precisa ocupar área contígua em memória
- Endereços são gerados dinamicamente em tempo de execução
- Somente um registrador então, não é suficiente

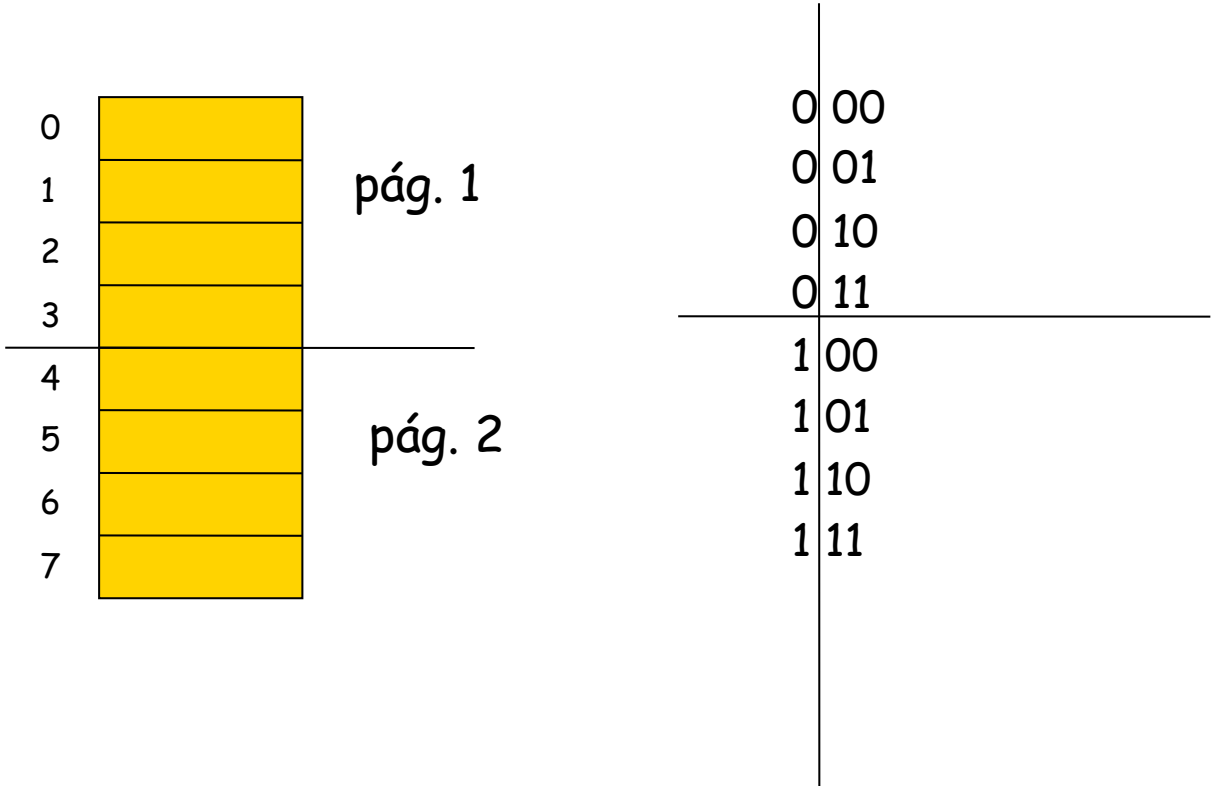
Tabela de Páginas

Paginação: suporte de *h/w*

- Cada processo tem sua tabela de páginas (TP)
  - TP: mapeamento página x quadro
  - Bit de presença
  - Bit de modificação
- Como funciona?
  - endereço lógico Z - nº da pagina + offset dentro da página

Paginação: suporte de *h/w*

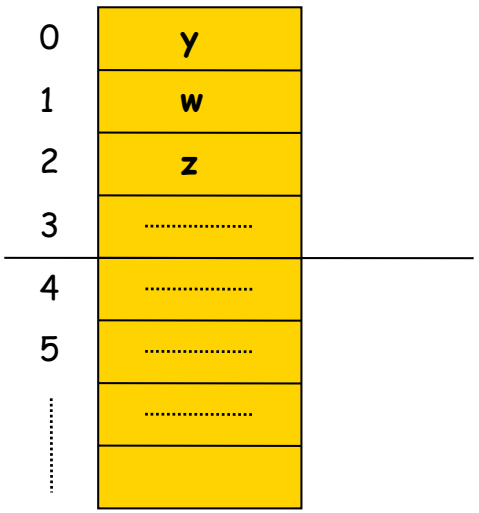
- Ex.: processo tem 8 endereços, 4 endereços por página



Paginação: suporte de *h/w*

- Ex.: endereço lógico (16 bits) – 1502
- tamanho da página 1K bytes (cada endereço armazena 1 byte)
- 1502 → 0000010111011110

Tabela de Páginas



2^{10} endereços/página → 10 bits para especificar o *offset* (478) dentro de uma página

000001 0111011110

Logo, restam 6 bits do end. lógico para especificar a página → página 1

Endereço real = 1K * w + 478 → frame tem 1K

Segmentação

- Programas são normalmente separados em módulos: unidade lógica
- Segmentos de um programa não precisam ser do mesmo tamanho
- Existe um tamanho máximo para o segmento
- Usuário tem controle
- elimina fragmentação interna (mas pode haver, externa)

Segmentação

Tabela de Segmentos

- entrada = endereço inicial na MP e tamanho do segmento
- cálculo do endereço real similar a paginação. Como?

Exercício

1) Considere um espaço de endereçamento lógico de oito páginas de 1024 palavras cada, mapeado em um memória física de 32 quadros. Quantos bits existem no endereço lógico? E no endereço físico?

2) Considere um sistema em que a memória é gerenciada por uma lista encadeada de blocos disponíveis, de diferentes tamanhos. Essa lista é definida por uma estrutura de dados contendo o endereço base do bloco, o tamanho de cada bloco e um apontador para o próximo elemento da lista. Existem dois apontadores, um para o primeiro elemento da lista, e outro para o último. Escreva o procedimento **addr = allocmem (n)**, onde **n** é o tamanho do bloco que deve ser alocado e **addr** contém, no retorno da chamada, o endereço base do bloco alocado. A técnica de alocação a ser utilizada é ***First-Fit***. Escreva também o procedimento **free(addr).**