



Laboratório de Pesquisa em Redes e Multimídia

A grayscale, semi-transparent image of a computer keyboard is positioned in the background, angled from the top-left towards the bottom-right. The keys are clearly visible but faded, serving as a backdrop for the text.

Gerência de Memória

Aspectos de Projeto



Universidade Federal do Espírito Santo
Departamento de Informática

Políticas de Busca de Páginas de um Processo

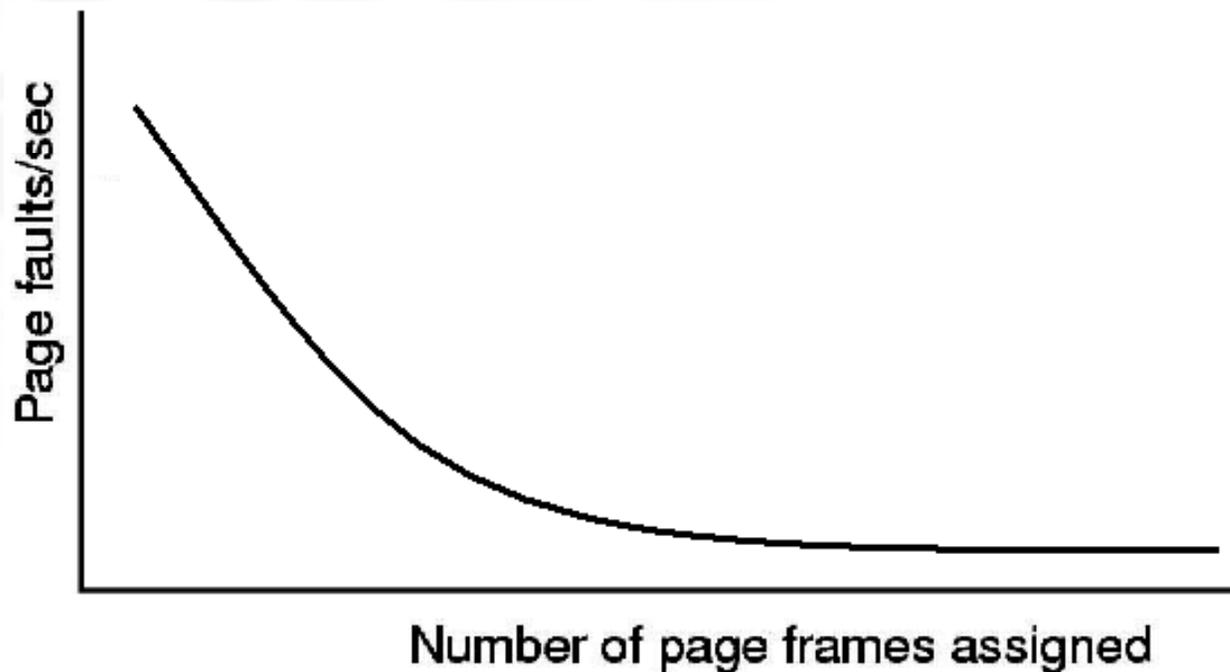
- Determina em que instante uma página deve ser trazida para memória principal
 - O objetivo é minimizar o número de faltas de página
- **Paginação por demanda**
 - No modo mais puro de paginação, os processos são iniciados sem qualquer página na memória
 - Quando a CPU tenta buscar a 1ª instrução, há um *Page fault*, forçando o S.O. a carregar a página na MP
 - À medida que *Page faults* vão ocorrendo, as demais páginas são carregadas
- **Pré-paginação** (Denning, *Working Set*)

Working Set (1)

- O conjunto de páginas que um processo está atualmente usando é denominado ***Working Set*** (espaço de trabalho)
- Verifica-se que, para intervalos de tempo razoáveis, o espaço de trabalho de um processo mantém-se constante e menor que o seu espaço de endereçamento
- Se todo o *Working Set* está presente na memória, o processo será executado com poucas *Page Fault* até passar para a próxima fase do programa, quando o *Working Set* é atualizado
 - Ex: Compilador de dois passos
- Se vários processos tiverem menos páginas em memória que o seu espaço de trabalho o sistema pode entrar em colapso (***trashing***)

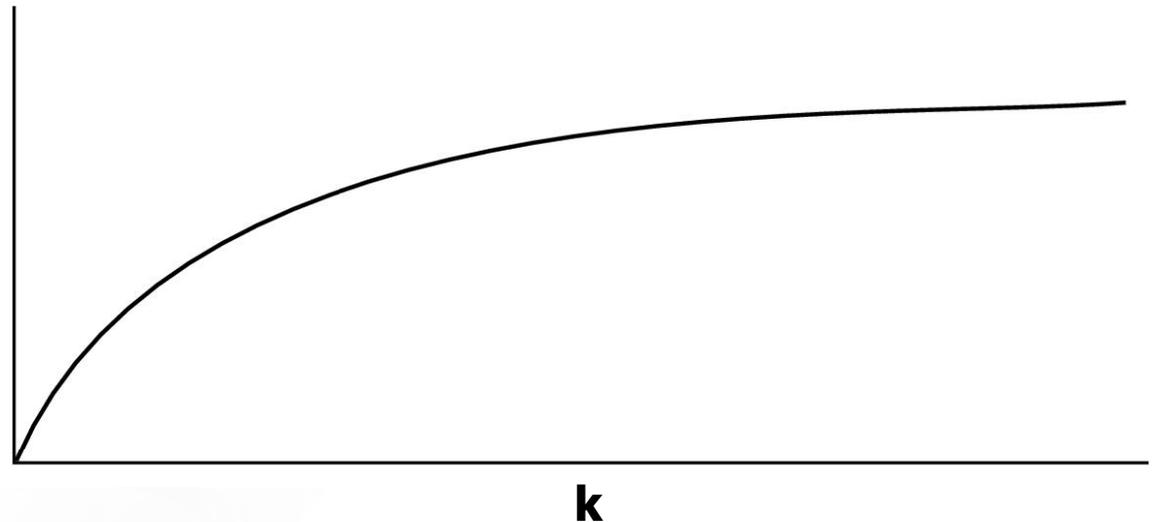
Working Set (2)

- Como prevenir o *Trashing*?



Working Set ⁽³⁾

$w(k,t)$

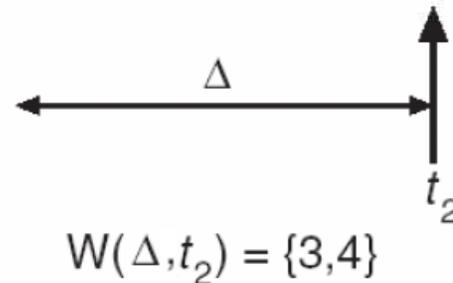
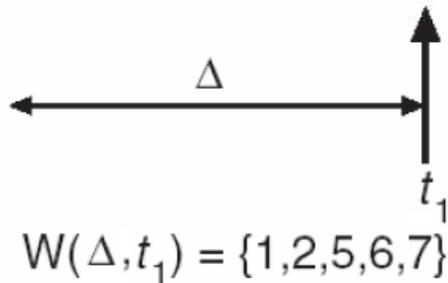


- O *Working Set* = as páginas usadas (referenciadas) pelas k referências mais recentes à memória
 - Ou aquelas usadas nos últimos τ segundos.
- A função $w(k,t)$ [ou $w(\tau,t)$] retorna a quantidade de páginas do Working Set no instante t .

Working Set (4)

page reference table

... 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 4 1 3 2 3 4 4 4 3 4 4 4 ...



Working Set (5)

- Alocação fixa:
 - cada processo recebe um número fixo de quadros
 - em caso de falta de páginas, uma das residentes é trocada
- Alocação variável: número de páginas varia durante a execução do processo
 - Utilização de valores máximo e mínimo de dimensão do espaço de trabalho para controlar a paginação
 - Estes valores devem-se adaptar dinamicamente a cada aplicação

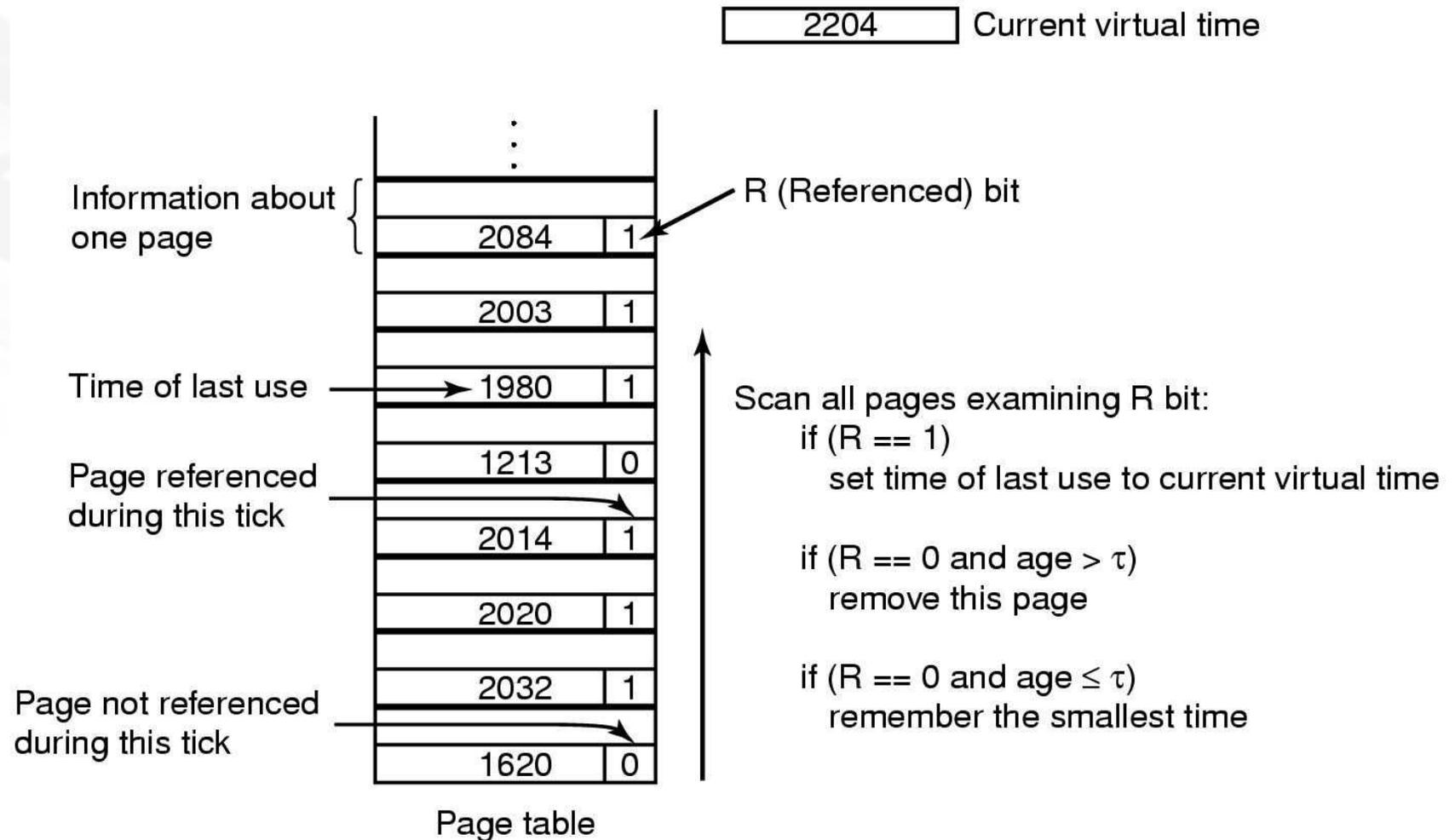
Working Set (6)

- Nos sistemas time-sharing processos estão constantemente bloqueados
- *Swapping*
 - Técnica para resolver problema de processos que aguardam por espaço livre adequado;
 - Processos não ficam mais na memória o tempo todo (são então suspensos).
 - Um processo residente na memória é levado para o disco (**Swapped-Out**), dando lugar a outro;
 - O processo Swapped-Out retorna à memória (**Swapped-In**), sem “perceber” o que ocorreu.
- Se paginação **por demanda**, 20, 50, 100... Page faults cada vez que o processo é re-carregado na MP
 - Processo fica lento, perda de tempo de CPU

Working Set (7)

- Pré-paginação
 - Carregar em memória as páginas do Working set do processo antes que o mesmo possa continuar sua execução
 - Garantimos que ocorrerá menos Page faults quando o processo for executado
- Como monitorar o Working set do processo de modo que ele esteja sempre atualizado?
 - Se a página não foi referenciada nos n clock ticks consecutivos, sai do Working set

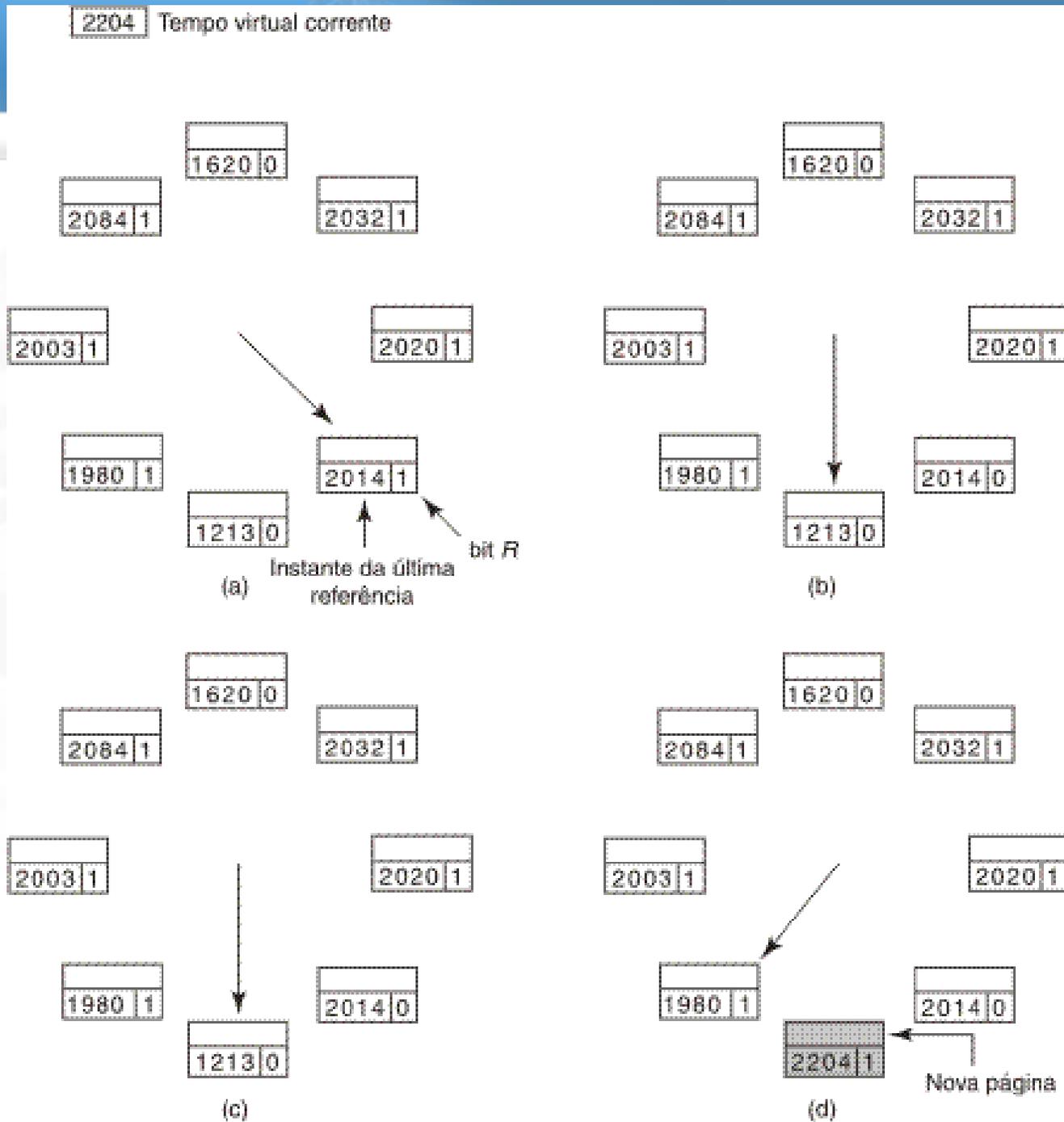
Working Set (8)



Algoritmo *WSClock* (1)

- O algoritmo básico de troca de páginas baseado no *Working set* exigiria uma varredura por toda a tabela de páginas
- No *WSClock* (*Working Set Clock*), na troca de páginas só são avaliadas as páginas presentes em uma lista circular
- Cada entrada dessa lista possui os bits R e M, além de um *timestamp* (tempo da última referência)
- Quando a primeira página é carregada, ela é inserida na lista
- Troca-se a primeira página a partir da posição do ponteiro na lista que tenha $R=0$ e cuja idade supera τ
 - Na verdade, também se a pag. está limpa (i.e. se ela ã foi modificada), se NÃO, é escalonada uma escrita dessa pag. no disco e pula-se p/ a próxima da lista. 11

Algoritmo WSClock (2)



Resumo dos Algoritmos

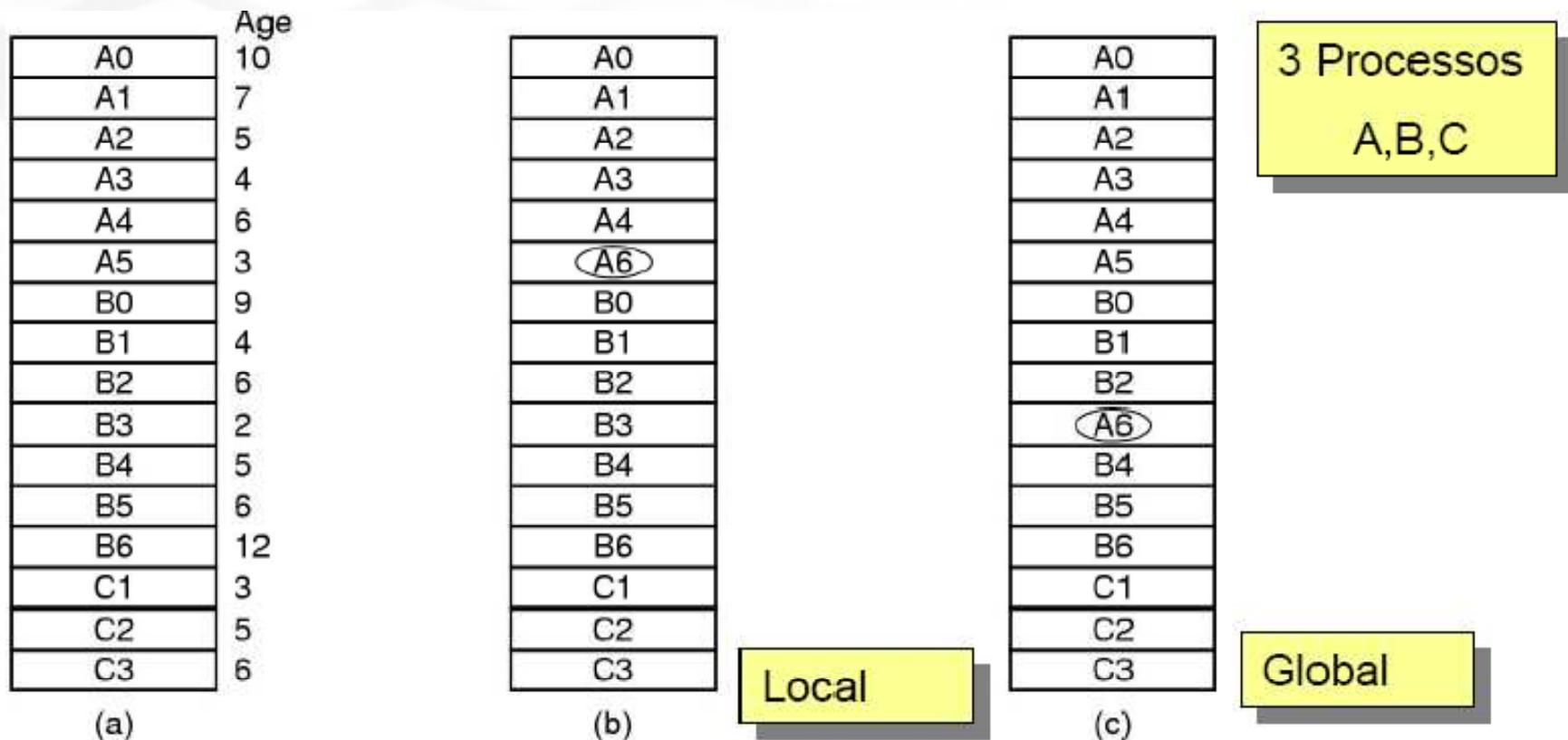
| | |
|----------------|---|
| Ótimo | Não é possível(referência) |
| NRU | Fácil de implementar; Pouco eficiente |
| FIFO | Pode retirar páginas importantes |
| Segunda Chance | Melhorias ao FIFO |
| Relógio | Implementação eficiente do SC; Realista |
| LRU | Excelente, difícil de implementar (HW) |
| NFU | Fraca aproximação do LRU |
| Aging | Eficiente que se aproxima do LRU |
| Working Set | Difícil de implementar |
| WSClock | Boa eficiência |

Considerações no Projeto de Sistemas de Paginação

- Política de alocação: Local x Global
- Anomalia de Belady
- Modelagem: Algoritmos de Pilha
- Controle de Carga
- Tamanho da página
- Espaços de Instruções e Dados Separados
- Páginas compartilhadas

Política de alocação: Local x Global (1)

- O LRU deve considerar as páginas apenas do processo que gerou o Page Fault, ou de todos os processos?



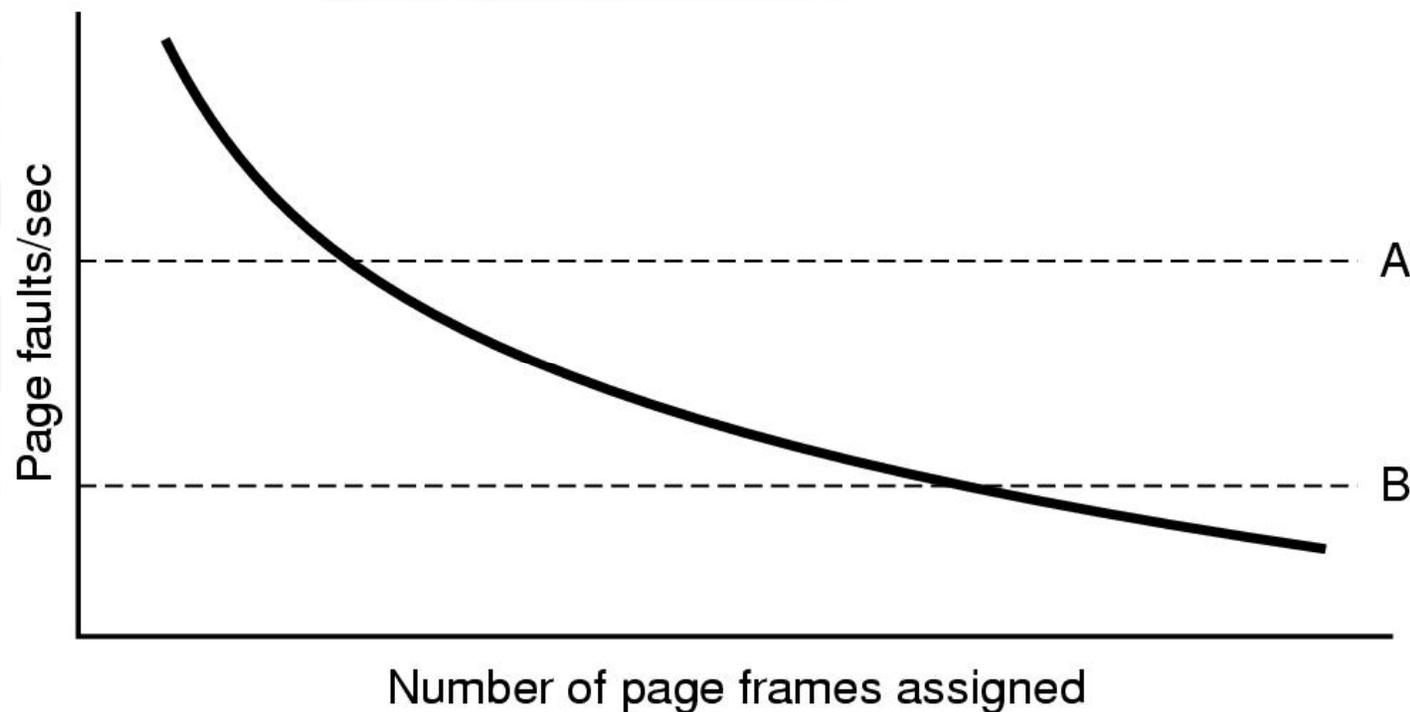
Política de alocação: Local x Global (2)

- Política LOCAL
 - Alocam uma fração fixa de memória para cada processo
- Política GLOBAL
 - Alocam molduras de páginas entre os processos em execução
 - O número de molduras alocadas para cada processo varia no tempo
- Working set varia durante a execução de um processo
 - Quando a política é local
 - Há trashing quando o tamanho do WS aumenta
 - Há desperdício quando o tamanho do WS diminui
 - Algoritmos com política global são mais adequados
 - Usa-se os bits de "tempo da última referência" para monitorar o Working Set
 - Não necessariamente evita o trashing -> o Working set pode variar de tamanho em questão de microssegundos (os bits de aging são alterados a cada interrupção de relógio)

Política de alocação: Local x Global (3)

- Outra abordagem determinar periodicamente o número de processos e dividir as molduras entre os mesmo
 - 12.416 molduras ; 10 processos => 1.241 molduras / processo
 - É justo? E se processos têm tamanho diferentes?
- Solução:
 - Alocar para cada processo um número mínimo de páginas proporcional ao tamanho do processo
 - Atualizar a alocação dinamicamente
- Algoritmo de alocação *Page Fault Frequency* (PFF)
 - Informa quando aumentar ou diminuir a alocação de molduras para um processo
 - Tenta manter a taxa de Page Fault dentro de um intervalo aceitável

Política de alocação: Local x Global (4)



- Se maior do que A, taxa muito alta
 - Deve-se alocar mais molduras
- Se menor do que B, taxa muito baixa
 - Algumas molduras podem ser eliminadas

Anomalia de Belady (1)

- Intuitivamente, quanto maior o número de molduras, menor será o número de *Page Faults*
 - Nem sempre isso será verdadeiro!
- Belady et al. descobriram um contra-exemplo para o algoritmo FIFO
 - Suponha que as páginas sejam referenciadas nesta ordem:
0 1 2 3 0 1 4 0 1 2 3 4
 - Qual será o número de Page Faults em um FIFO alocando 3 molduras para o processo? E 4 molduras?

Anomalia de Belady (2)

| | 0 | 1 | 2 | 3 | 0 | 1 | 4 | 0 | 1 | 2 | 3 | 4 | |
|--------------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Página mais nova | | 0 | 1 | 2 | 3 | 0 | 1 | 4 | 4 | 4 | 2 | 3 | 3 |
| | | | 0 | 1 | 2 | 3 | 0 | 1 | 1 | 1 | 4 | 2 | 2 |
| Página mais velha | | | | 0 | 1 | 2 | 3 | 0 | 0 | 0 | 1 | 4 | 4 |
| | | | | | | | | | | | | | |
| | | P | P | P | P | P | P | | | | P | P | |

9 Page Faults

| | 0 | 1 | 2 | 3 | 0 | 1 | 4 | 0 | 1 | 2 | 3 | 4 | |
|--------------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Página mais nova | | 0 | 1 | 2 | 3 | 3 | 3 | 4 | 0 | 1 | 2 | 3 | 4 |
| | | | 0 | 1 | 2 | 2 | 2 | 3 | 4 | 0 | 1 | 2 | 3 |
| | | | | 0 | 1 | 1 | 1 | 2 | 3 | 4 | 0 | 1 | 2 |
| Página mais velha | | | | | 0 | 0 | 0 | 1 | 2 | 3 | 4 | 0 | 1 |
| | | | | | | | | | | | | | |
| | | P | P | P | P | | | P | P | P | P | P | P |

10 Page Faults

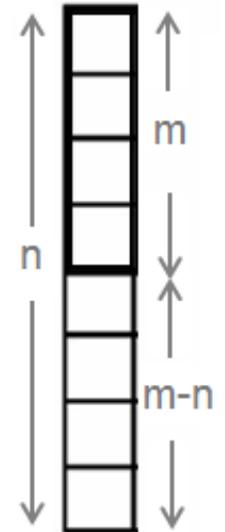
Algoritmos de Pilha (1)

- Teoria sobre algoritmos de paginação e suas propriedades
- Sabe-se que um processo gera uma seqüência de referências à memória
 - Cadeia de referências (*Reference String*)
- Sistema de Paginação caracterizado por 3 itens
 1. Cadeia de referências do processo em execução
 2. Algoritmo de substituição de páginas
 3. Número de molduras disponíveis (**m**)
- Conceitualmente, imagine um interpretador abstrato que mantém um **vetor M** que guarda o estado da memória

Algoritmos de Pilha (2)

■ Vetor M

- O vetor tem n elementos, que equivale ao número de páginas de um processo
- O vetor é dividido em duas partes
 - Parte superior, com m entradas, representando as páginas que estão atualmente na memória ($m = n^{\circ}$ de molduras)
 - Parte inferior, com $n-m$ entradas, abrangendo as páginas que já foram referenciadas 1 vez mas que foram devolvidas ao disco



- Inicialmente o vetor encontra-se vazio
- A cada referência, o interpretador verifica se a página está na memória (*i.e.* na parte superior de M)
 - Se não estiver, ocorre Page Fault.
 - Se ainda existirem molduras livres, coloca a página na memória (escrevendo a página na parte superior de M).
 - Se não há molduras livres, aplica o algoritmo de substituição de páginas. Alguma página será deslocada da parte superior do vetor para a parte inferior deste.

Algoritmos de Pilha (3)

| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reference string | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 | | |
| M | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 | |
| | | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 7 | 3 | 3 | 5 | 3 | 3 | 3 | 1 | 7 | 1 | 3 | 4 | |
| | | | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 3 | 4 | 4 | 7 | 7 | 7 | 5 | 5 | 5 | 3 | 3 | 7 | 1 | 3 | |
| | | | | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 6 | 6 | 6 | 4 | 4 | 4 | 7 | 7 | 7 | 5 | 5 | 5 | 7 | 7 | |
| | | | | | | 0 | 2 | 1 | 1 | 5 | 5 | 5 | 5 | 5 | 6 | 6 | 6 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 5 | 5 |
| | | | | | | | 0 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| | | | | | | | | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Page faults | | P | P | P | P | P | P | | P | | | | | P | | P | | | | | | | | P | | |

- Sempre que uma pag é ref. , ela é movida p/ o topo de M
- Se a pag. ref. estiver em M, as pag. acima dela serão todas deslocadas de uma posição p/ baixo
- as páginas que estão abaixo da página referenciada ã são movidas
- Uma pag. que sai da caixa em negrito e vai p/ a parte inferior corresponde a uma pag. virtual sendo removida da memória
- Sempre que uma pag ref. ã estiver no quadrante em negrito, há um PF

Algoritmos de Pilha (3)

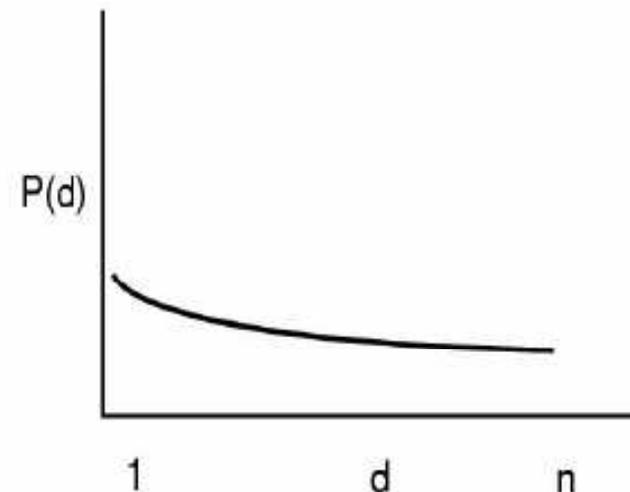
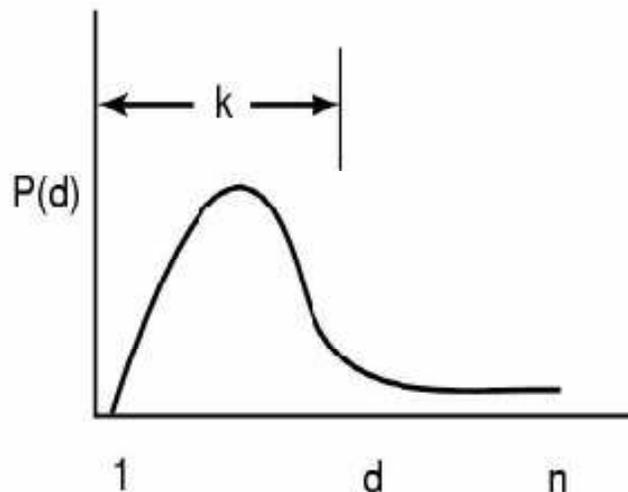
| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reference string | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 | | |
| M LRU | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 | |
| | | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 7 | 3 | 3 | 5 | 3 | 3 | 3 | 1 | 7 | 1 | 3 | 4 | |
| | | | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 3 | 4 | 4 | 7 | 7 | 7 | 5 | 5 | 5 | 3 | 3 | 7 | 1 | 3 | |
| | | | | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 6 | 6 | 6 | 4 | 4 | 4 | 7 | 7 | 7 | 5 | 5 | 5 | 7 | 7 | |
| | | | | | | 0 | 2 | 1 | 1 | 5 | 5 | 5 | 5 | 5 | 6 | 6 | 6 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 5 | 5 |
| | | | | | | | 0 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| | | | | | | | | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Page faults | | P | P | P | P | P | P | | P | | | | | P | | | P | | | | | | | P | | |
| Distance string | | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 4 | ∞ | 4 | 2 | 3 | 1 | 5 | 1 | 2 | 6 | 1 | 1 | 4 | 2 | 3 | 5 | 3 | |


 número de molduras
 índice na cadeia de referências

- Propriedade Importante: $M(m, r) \subseteq M(m+1, r)$
 - Algoritmos que apresentam esta propriedade são ditos Algoritmos de Pilha
 - LRU é um exemplo... já o FIFO não (como mostra a Anomalia de Belady)

Cadeia de Distâncias (*Distance String*)

- Para cada referência, representar a distância entre o topo da pilha e a posição onde a página referenciada se encontra em M
- A propriedade estatística da Distance String tem um grande impacto na performance do algoritmo de substituição de páginas



Funções de densidade de propabilidade para duas *Distance Strings* hipotéticas

Previendo Taxas de Page Fault (1)

- Distance String pode ser utilizada para prever o número de Page Faults para diferentes tamanhos de memória
 - Número de Page Faults $c/ 1, 2, 3 \dots n$ molduras
- O algoritmo consiste em varrer a Distance String e contabilizar o número de vezes que '1' ocorre, '2', ocorre, e assim por diante
 - C_i é o número de ocorrências de i

Previendo Taxas de Page Fault (2)

Distance string ∞ ∞ ∞ ∞ ∞ ∞ ∞ 4 ∞ 4 2 3 1 5 1 2 6 1 1 4 2 3 5 3

| | |
|----------------|---|
| $C_1 = 4$ | ← # times 1 occurs in distance string |
| $C_2 = 3$ | |
| $C_3 = 3$ | |
| $C_4 = 3$ | |
| $C_5 = 2$ | ← # times 6 occurs in distance string |
| $C_6 = 1$ | |
| $C_7 = 0$ | |
| $C_\infty = 8$ | |

(a)

| | |
|----------------|--|
| $F_1 = 19$ | ← $C_2 + C_3 + C_4 + \dots + C_\infty$ |
| $F_2 = 17$ | ← $C_3 + C_4 + C_5 + \dots + C_\infty$ |
| $F_3 = 16$ | ← $C_4 + C_5 + C_6 + \dots + C_\infty$ |
| $F_4 = 12$ | |
| $F_5 = 10$ | ← # of page faults with 5 frames |
| $F_6 = 10$ | |
| $F_7 = 8$ | |
| $F_\infty = 8$ | |

$$F_m = \sum_{k=m+1}^n C_k + C_\infty$$

(b)

Controle de Carga

- Mesmo com paginação, swapping é ainda necessário
- Determina o número de processos residentes em MP (escalonador de médio prazo)
 - Poucos processos, possibilidade de processador vazio;
 - Muitos processos, possibilidade de *trashing*
- Regra dos 50% de utilização do dispositivo de paginação (acionado por *Page fault*)
- Swapping é usado para reduzir demanda potencial por memória, em vez de reivindicar blocos para uso imediato

Tamanho de Páginas (1)

- Página de pequeno tamanho
 - tempo curto para transferência de página entre disco e memória
 - muitas páginas de diferentes programas podem estar residentes em memória
 - menor fragmentação interna
 - exige tabelas de páginas muito grandes, que ocupam espaço em memória
 - mais adequada para instruções
- Página de grande tamanho
 - Tabelas de páginas pequenas
 - Transferência de 64 páginas de 512 B pode ser mais lenta do que a transferência de 4 páginas de 8KB
 - Tempo longo para transferência de uma página entre disco e memória
 - Mais adequada para dados (gráficos exigem páginas muito grandes)

Tamanho de Páginas (2)

- Custo adicional devido à paginação

- s : tamanho médio dos processos

- p : tamanho da página em bytes

- e : tamanho de cada entrada da tabela de páginas

$s \cdot e$ -> tamanho aproximado da tabela de páginas

p

$\frac{p}{2}$ -> memória desperdiçada na última página do processo

2

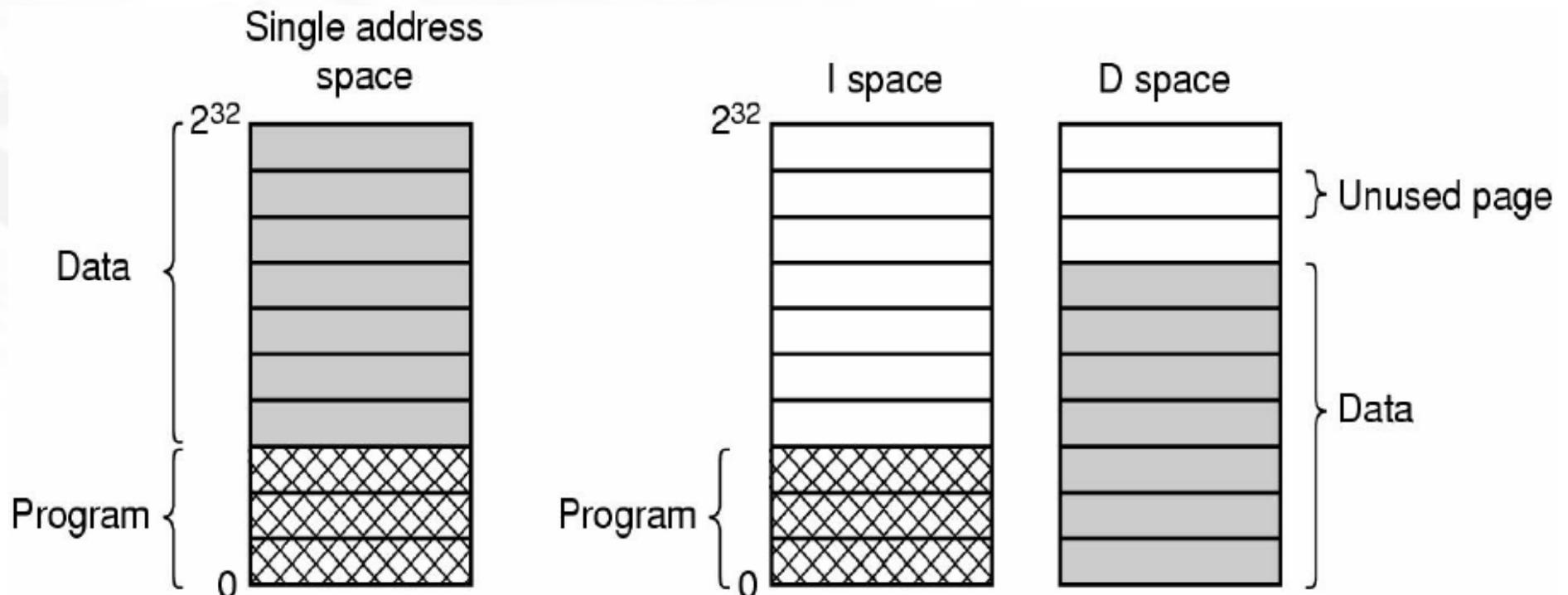
$$\text{custo adicional} = \frac{s \cdot e}{p} + \frac{p}{2}$$

Derivando em relação a p : o tamanho ótimo será: $p = \sqrt{2 \cdot s \cdot e}$

Tamanho de Páginas ⁽³⁾

- Solução de compromisso: permitir páginas de tamanhos diversos para código e dados
- Tamanhos de páginas variam muito, de 64 bytes a 4 Mbytes
 - Pentium (... x86 64) permite selecionar página de 4 K ou 4 Mbytes
 - Motorola MC88200
 - páginas de 4 Kbytes para programas de usuário
 - páginas de 512 Kbytes para programas do sistema, que devem residir sempre em memória

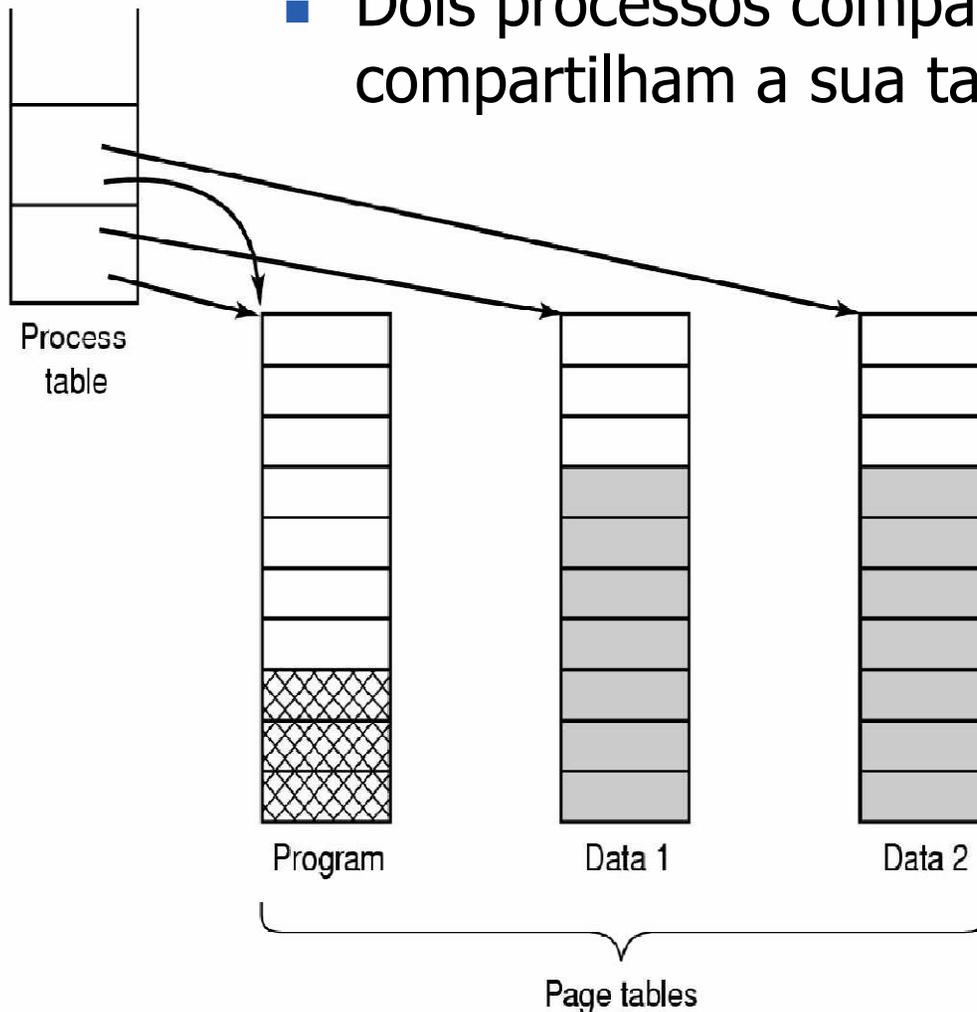
Espaços de Instruções e Dados Separados



- Duplica o espaço de endereçamento disponível
- Uma tabela de páginas para cada espaço de endereçamento

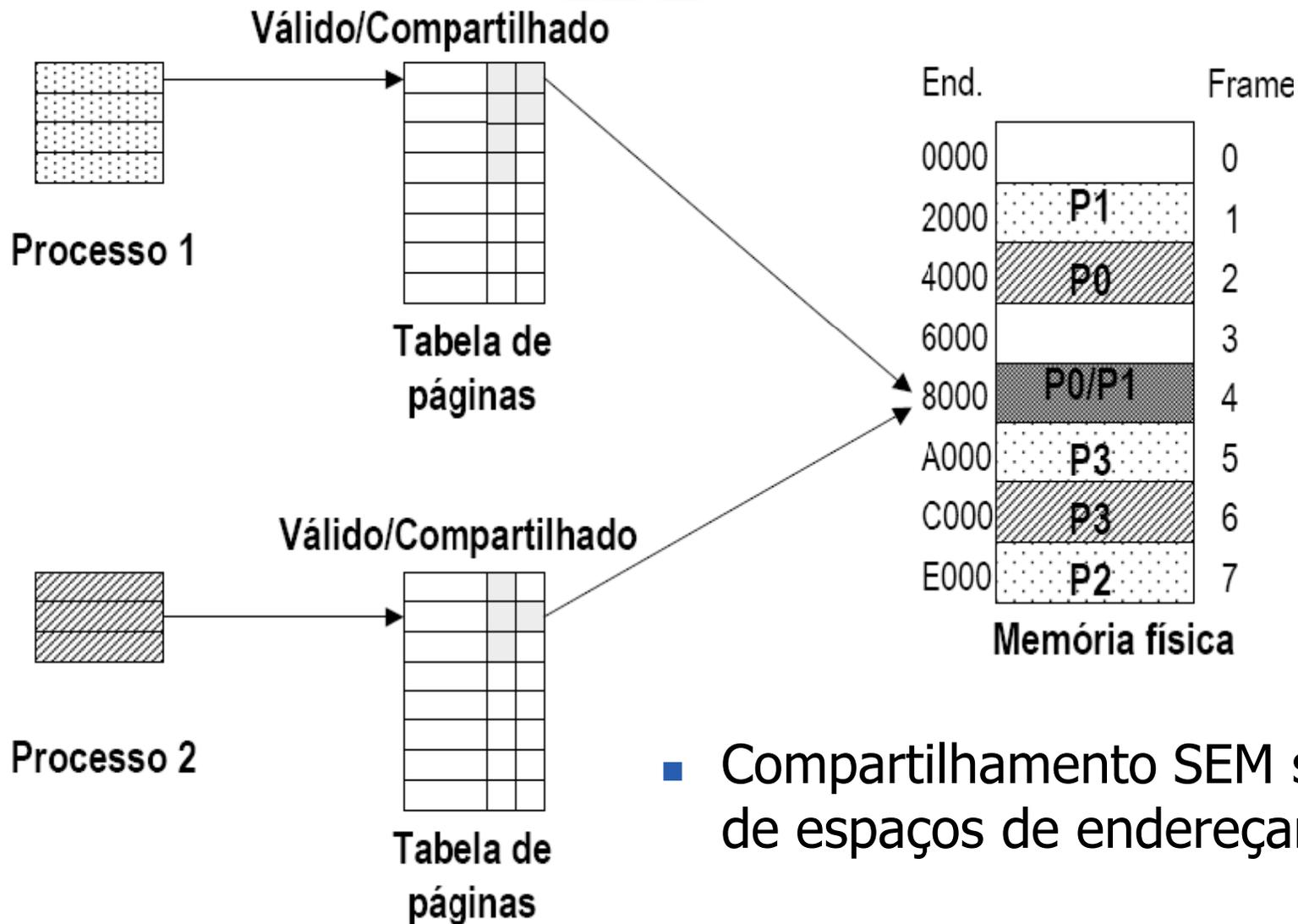
Páginas Compartilhadas (1)

- Dois processos compartilhando o mesmo programa, compartilham a sua tabela de páginas



- Mesmo não havendo dois espaços de endereçamento (Código e Dado) é possível compartilhar páginas, mas o mecanismo não é tão direto
- Usando tabelas invertidas o mecanismo é mais complicado ainda...

Páginas Compartilhadas (2)



- Compartilhamento SEM separação de espaços de endereçamento

Páginas Compartilhadas (3)

■ Código Reentrante

- Código que não modifica a si próprio, ou seja, ele nunca é modificado durante a execução
- Dois ou mais processos podem executar o mesmo código "simultaneamente"
- Exemplo:
 - Editor de texto com código reentrante de **150 K** e área de dados de **50 K**
 - **40** usuários utilizando o editor em um ambiente de tempo compartilhado, seriam necessários **$200\text{ K} \times 40 = 8000\text{ K}$**
 - Se o código executável for compartilhado, serão consumidos apenas **$(50\text{ K} \times 40) + 150\text{ K} = 2150\text{ K}$**

Referências

- **A.S. Tanenbaum, "Sistemas Operacionais Modernos", 2a. Edição, Editora Prentice-Hall, 2003.**
 - **Seções 4.4.8 a 4.8.10, 4.5 e 4.6**
- Silberschatz, P. Baer Galvin, e G. Gagne "Sistemas Operacionais com Java", 7a. Edição, Elsevier Editora / Campus, 2008. *Capítulo 10*