Laboratório de Pesquisa em Redes e Multimídia

Gerência de Memória

Aspectos de Projeto

Universidade Federal do Espírito Santo
Departamento de Informática

Políticas de Busca de Páginas de um Processo

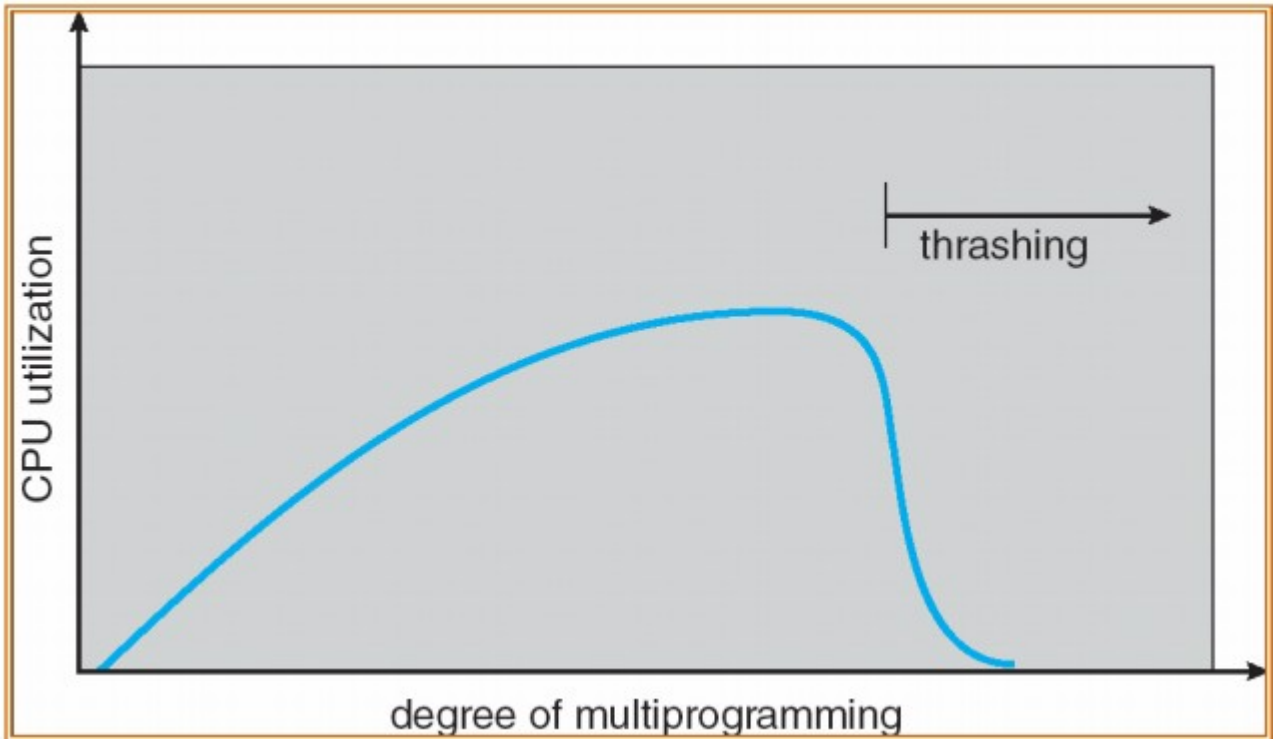
- Determina em que instante uma página deve ser trazida para memória principal
  - O objetivo é minimizar o número de faltas de página
- **Paginação por demanda**
  - No modo mais puro de paginação, os processos são iniciados sem qualquer página na memória
  - Quando a CPU tenta buscar a 1a instrução, 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***)

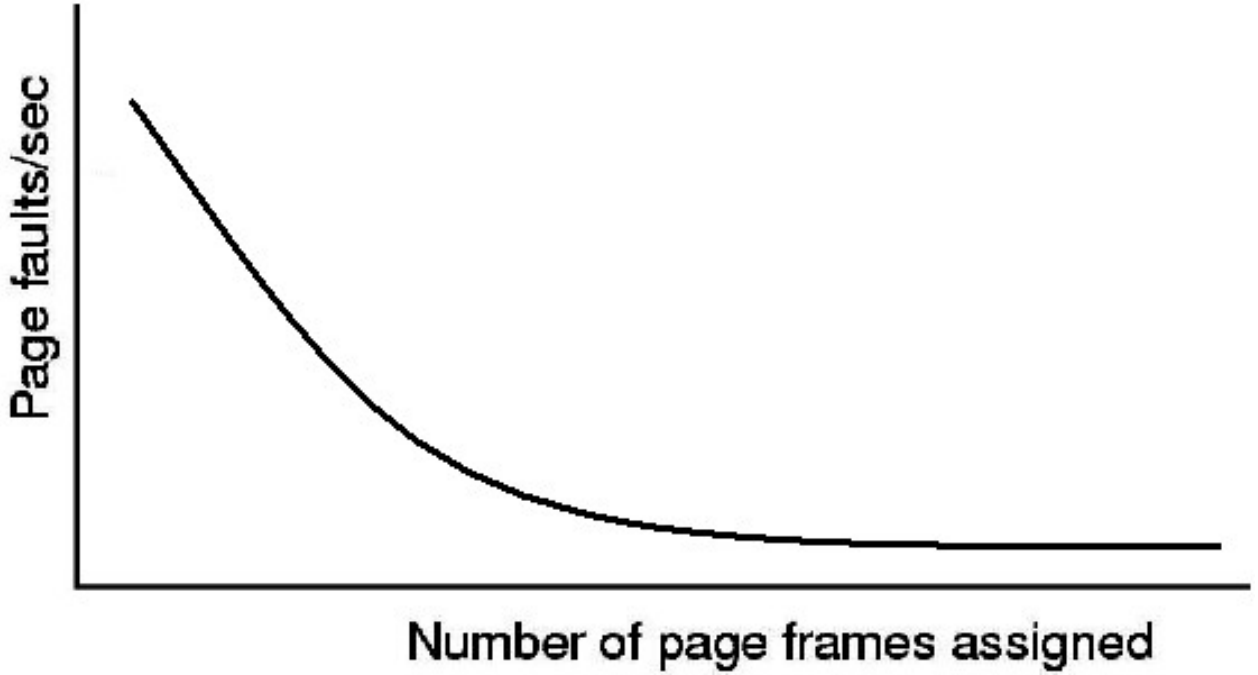
Working Set (2)

- *Trashing!!*

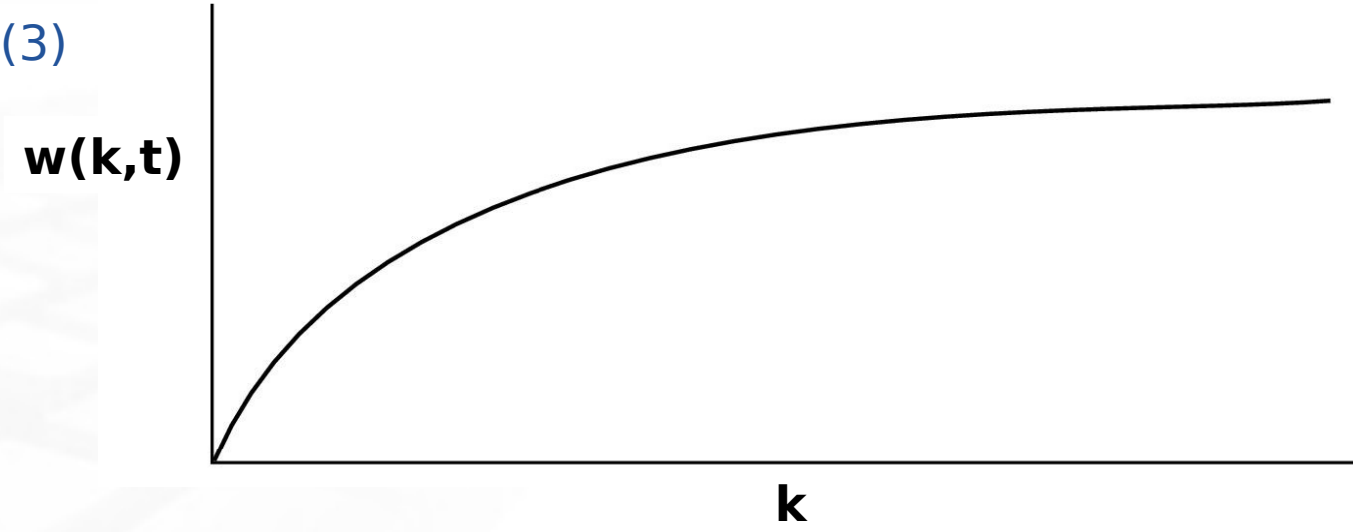


Working Set (2)

- Como prevenir o *Trashing*?

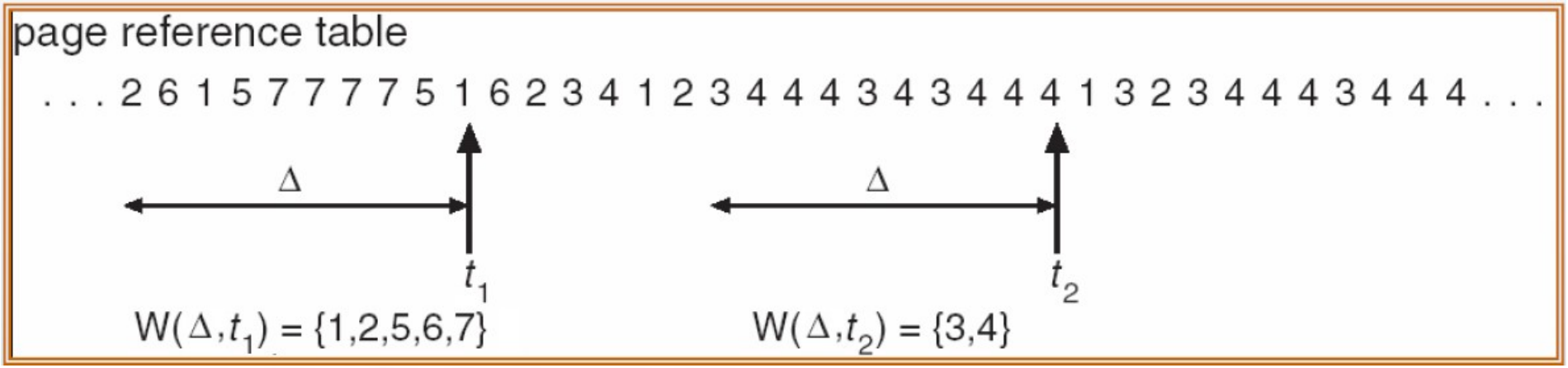


Working Set (3)



- 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(τ,t)**] retorna a quantidade de páginas do Working Set no instante *t.*

Working Set (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

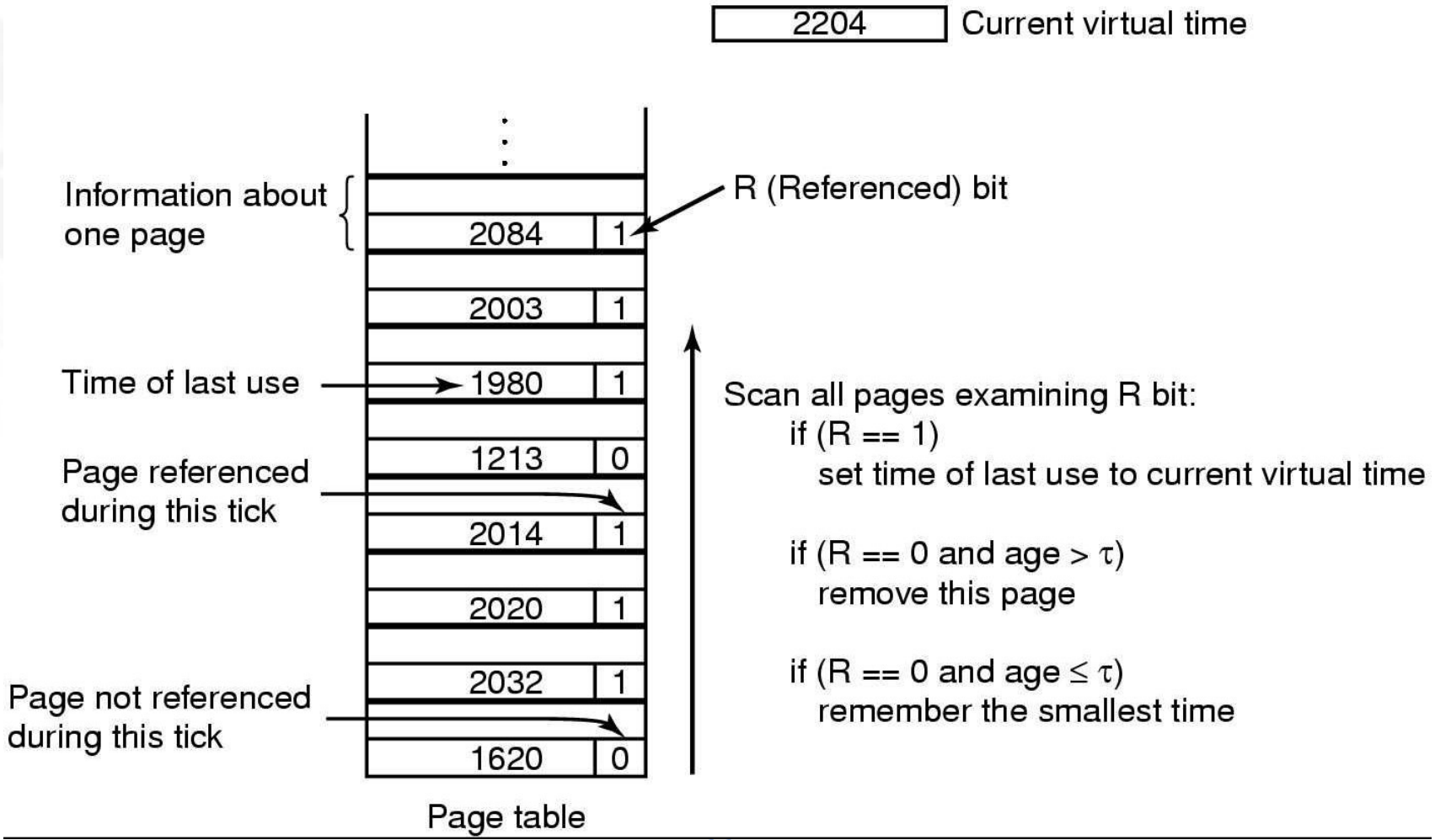
Working Set (6)

- 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

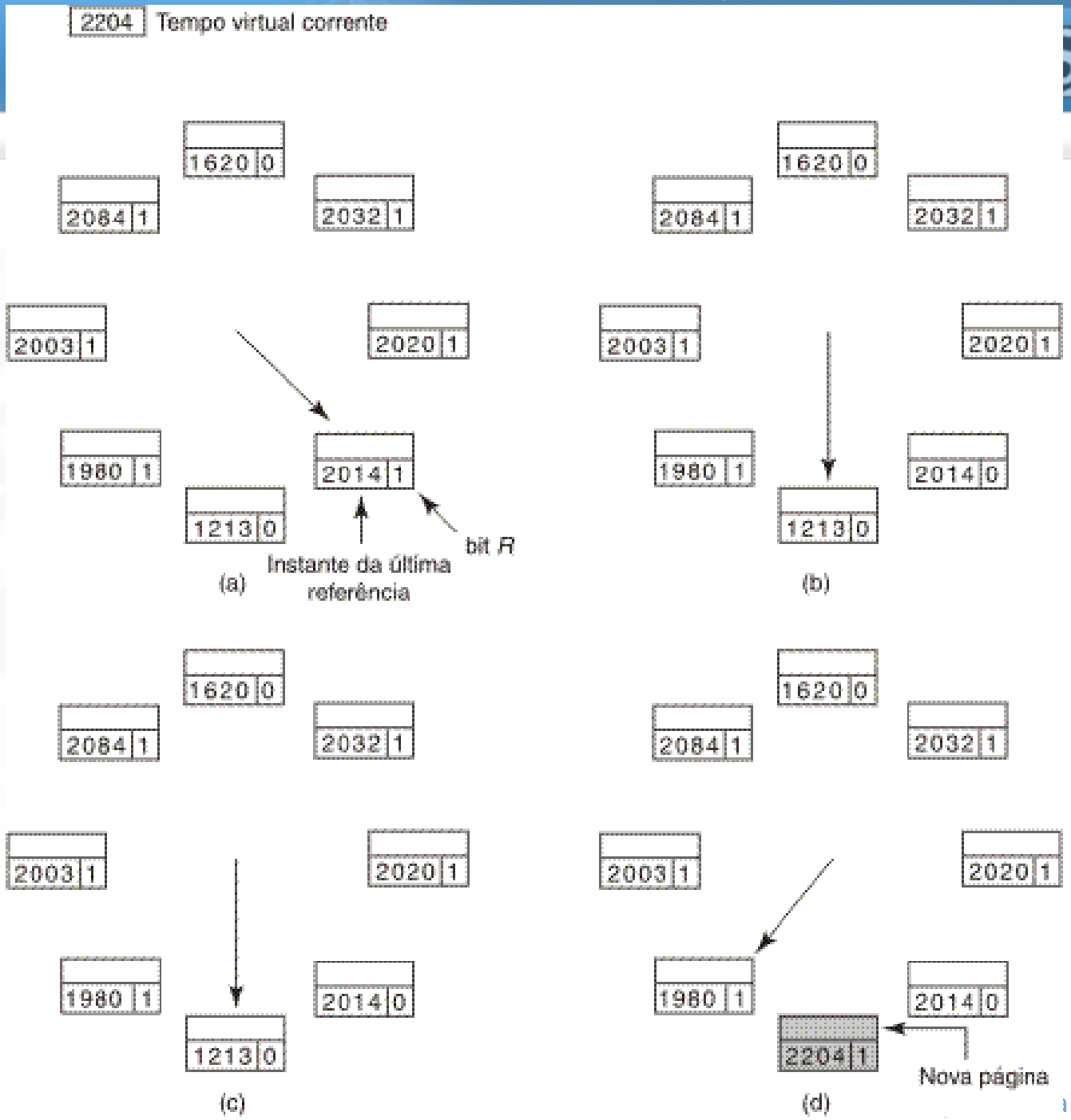
Working Set (8)



Algoritmo *WSClock* (1)

- 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)
- À medida que as páginas são carregadas em memória, elas são inseridas na lista circular
- O algoritmo é executado quando precisa-se “liberar” molduras
  - Isto pode ocorrer quando ocorrem page faults no processo em questão ou mesmo em outros processos (política Global de alocação de molduras)
  - Troca-se a primeira página (eventualmente, libera-se mais páginas) a partir da posição do ponteiro na lista que tenha R=0 e cuja idade supera τ
    - Na verdade, verifica-se se a pag. está limpa (i.e. se ela ñ foi modificada). Caso ela tenha M=1, é escalonada uma escrita dessa pag. no disco e pula-se p/ a próxima página da lista circular.

Algoritmo *WSClock* (2)



Resumo dos Algoritmos

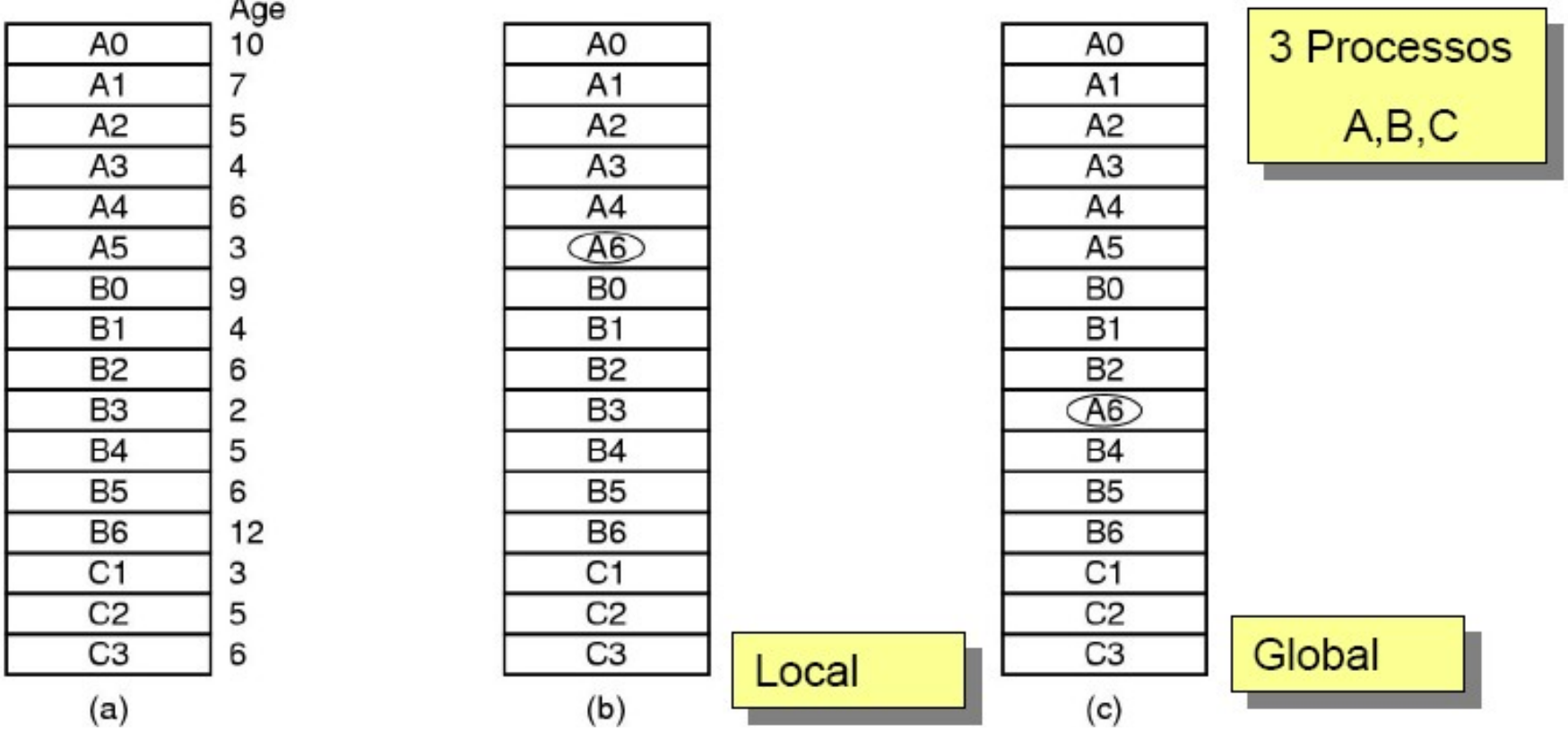
Ótimo	Não é possivel(referência)
NRU	Fácil de implementar; Pouco eficiente
FIFO	Pode retirar páginas importantes
Segunda Chance	Melhorias ao FIFO
Relógio	Implementação eficiente do SC; Realista
LRU	Excelente, dificil de implementar (HW)
NFU	Fraca aproximação do LRU
Aging	Eficiente que se aproxima do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

Considerações no Projeto de Sistemas de Paginação

- Política de alocação: Local x Global
- Anomalia de Belady
- 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?



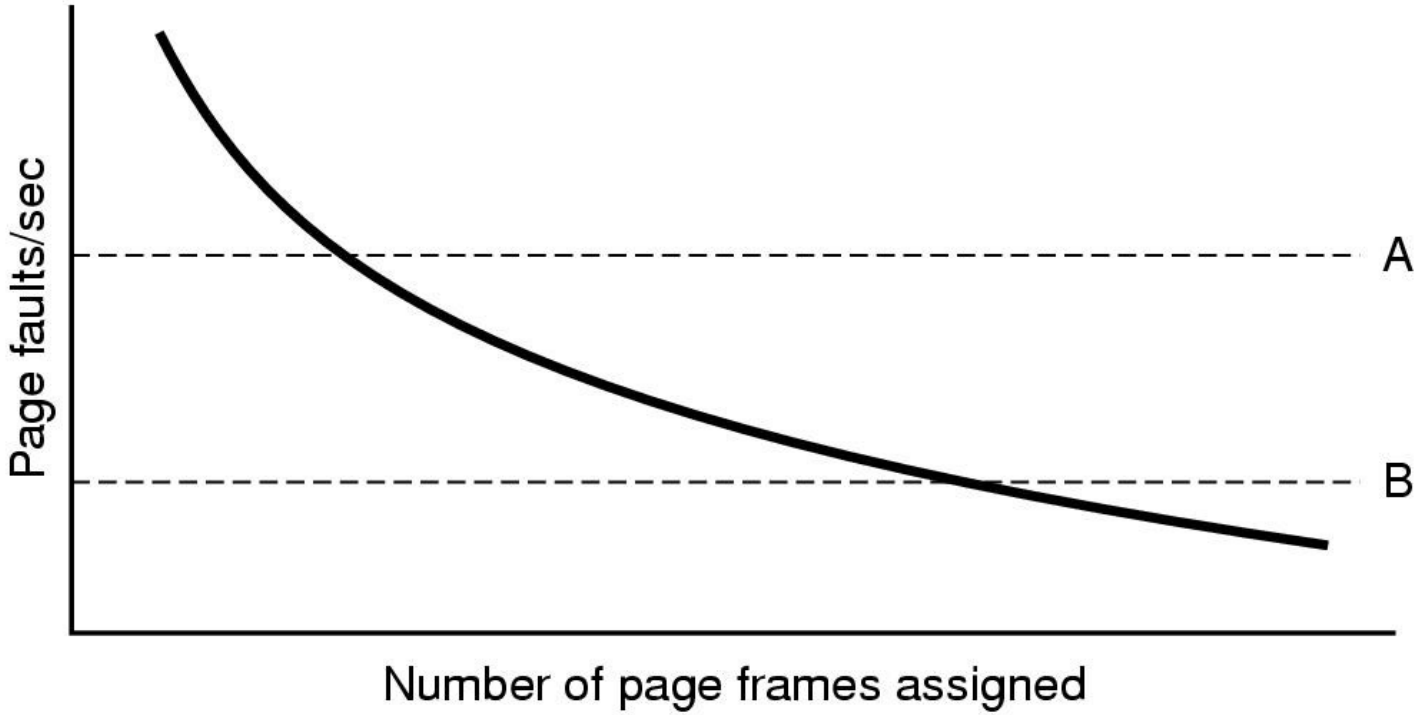
Política de alocação: Local x Global (2)

- 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 ultima referencia” 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
  - Usa-se em combinação com algum algoritmo de substituição de página

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?

- **Belady et al definiram um modelo (“Modelo de Pilha”) para mostrar se algoritmos apresentam a anomalia**

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	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

Controle de Carga

- Mesmo com paginação, swaping 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

$\frac{s}{p} . e$ -> tamanho aproximado da tabela de páginas

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

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

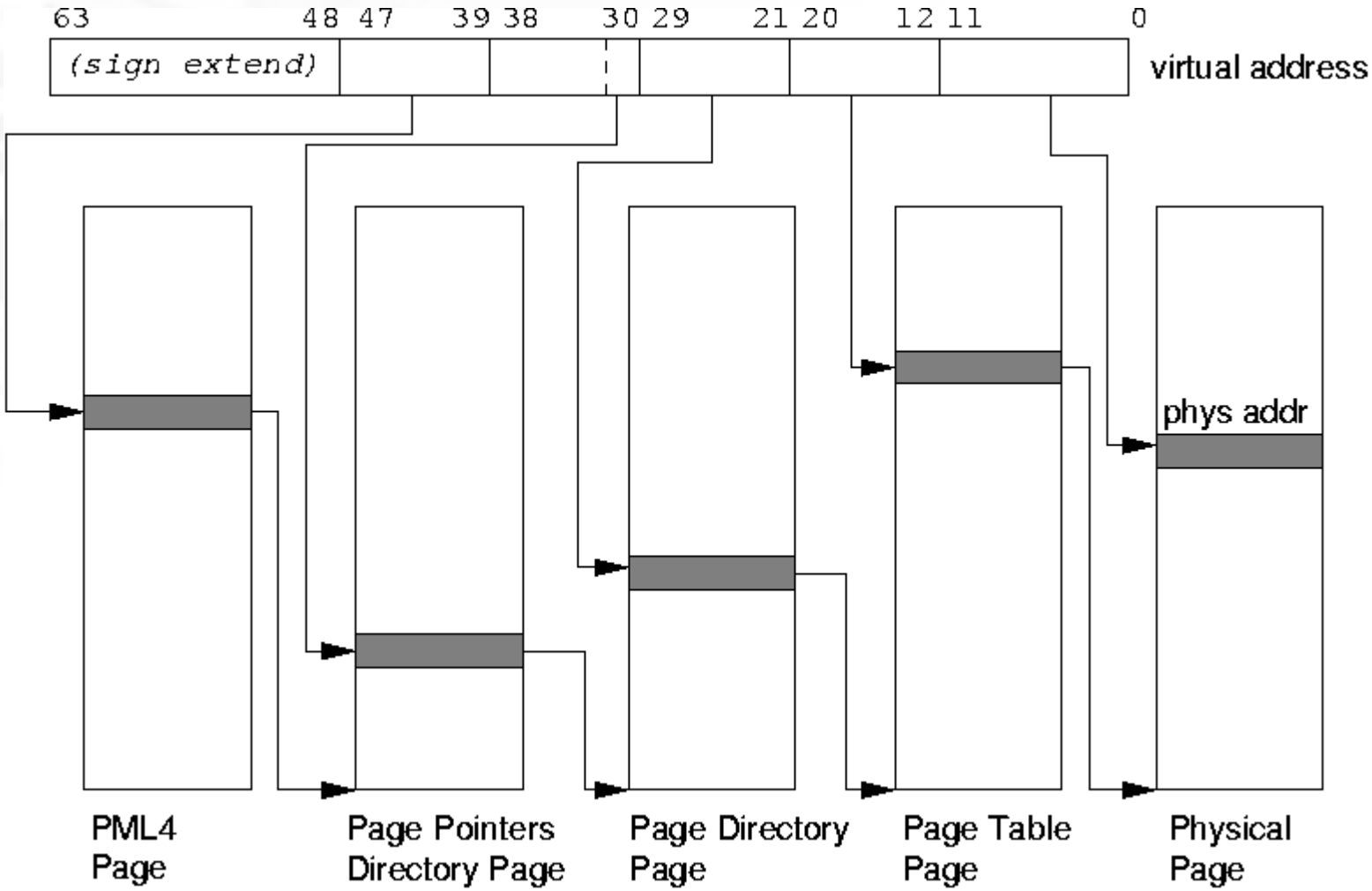
Derivando em relação a **p**: o tamanho ótimo será: $p=\sqrt{2.s.e}$

Tamanho de Páginas (3)

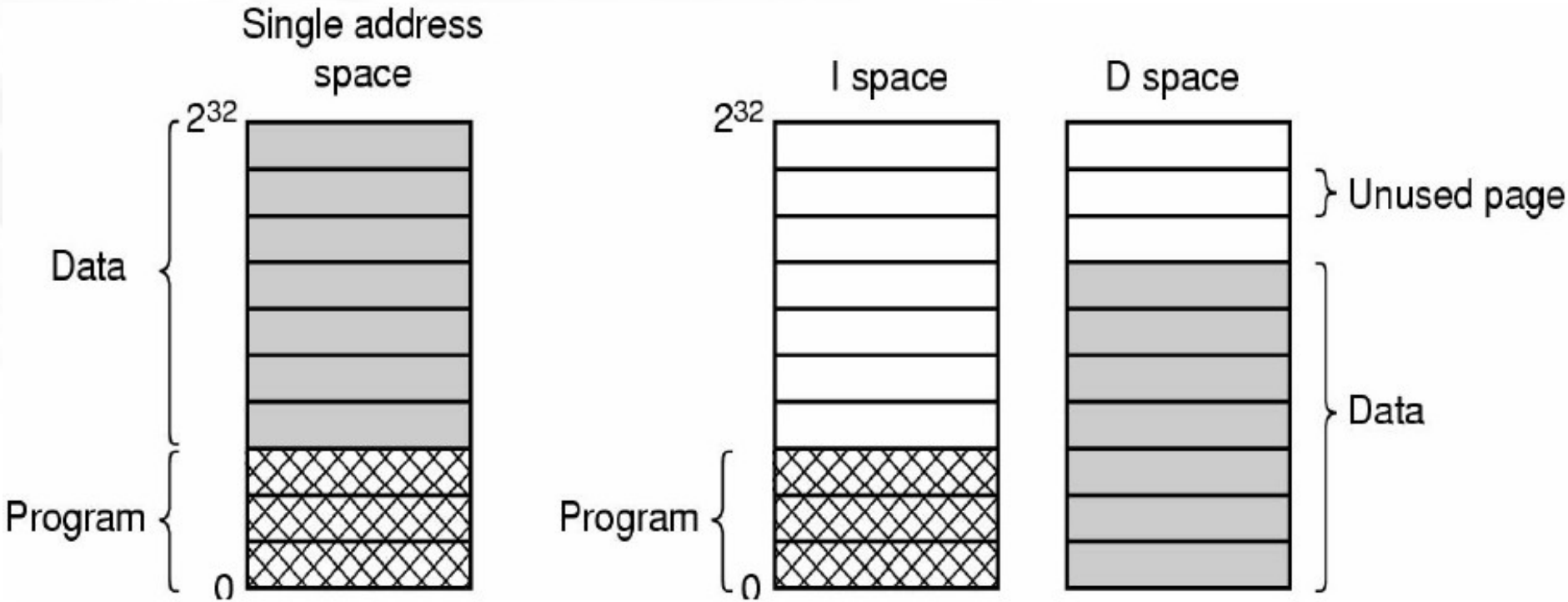
- Solução de compromisso: permitir páginas de tamanhos diversos para código e dados
- Tamanhos de páginas variam muito, de 64 bytes a 4 Mbytes
  - Pentium (... x86) permite selecionar página de 4 K ou 4 Mbytes
  - Motorola MC88200
    - páginas de 4 Kbytes para programas de usuário
    - páginas de 512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

- **64 bits: páginas de 4kb, 2Mb and 1Gb**
- **ARM: 4kb, 64kb, e 1Mb**

Tamanho de Páginas (4)



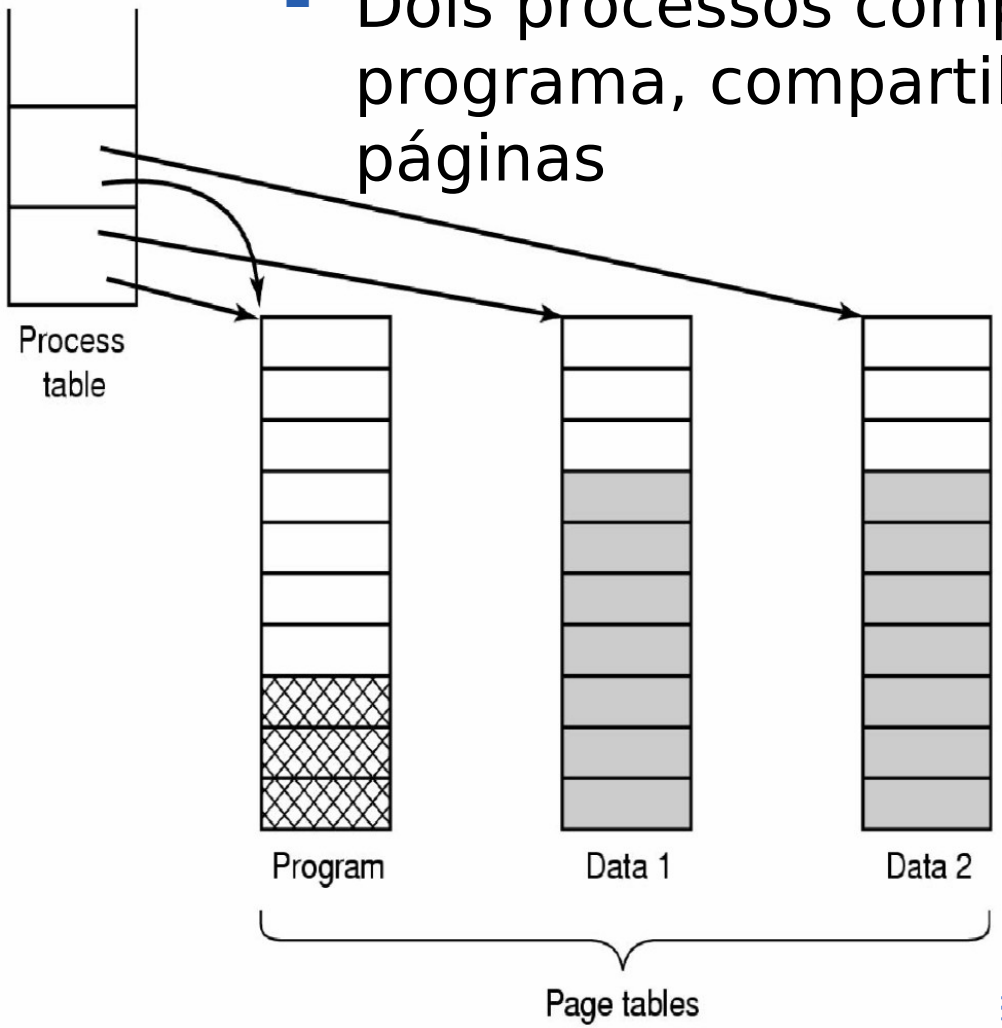
Espaços de Instruções e Dados Separados



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

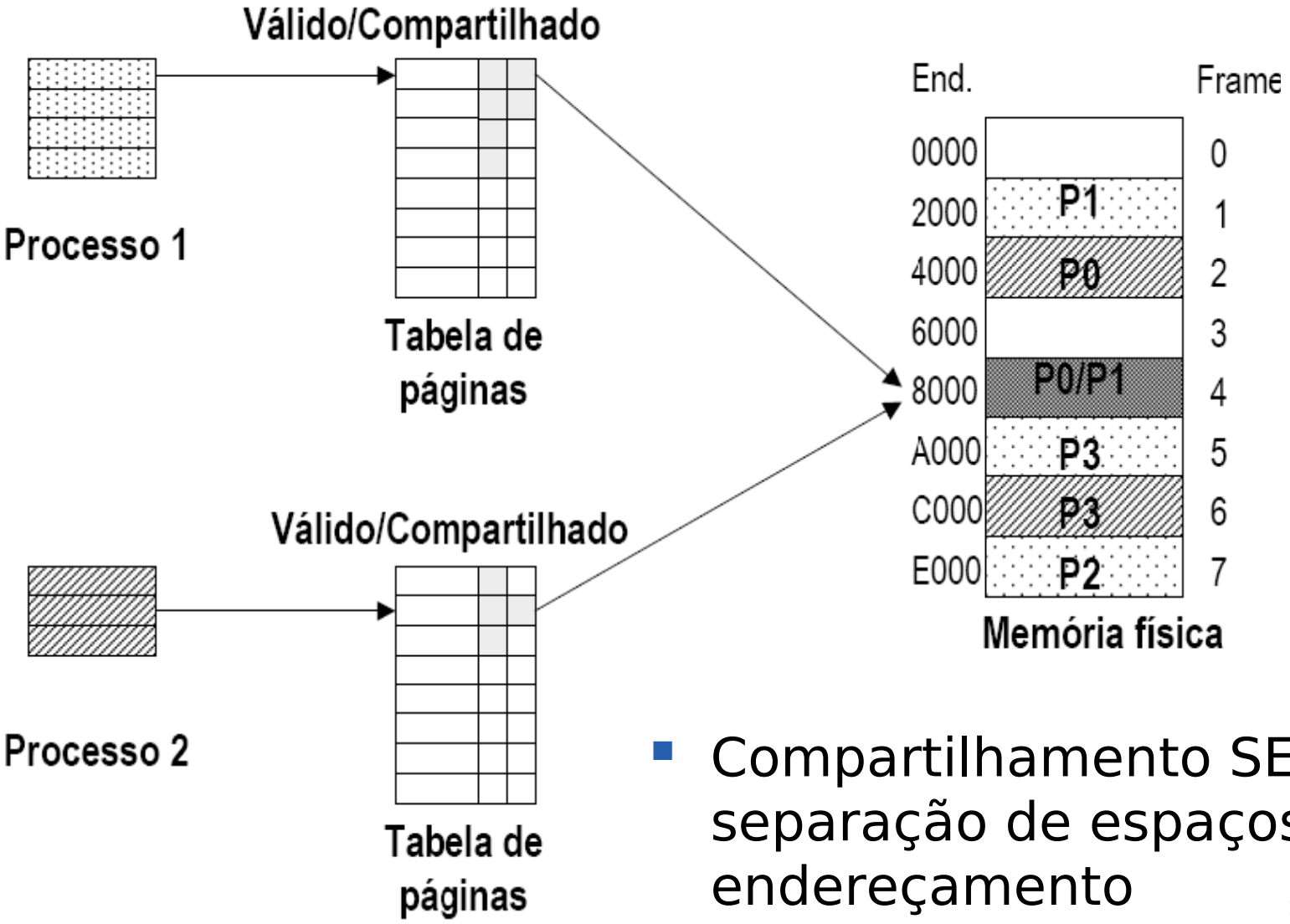
Páginas Compartilhadas (1)

- 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 K x 40 = 8000 K**
    - Se o código executável for compartilhado, serão consumidos apenas **(50 K x 40) + 150 K = 2150 K**

Referências

A. S. Tanenbaum, ''Sistemas Operacionais Modernos'', 4a. Edição, Editora Prentice-Hall, 2016.

- **Seções 4.4.8 a 4.8.10, 4.5 e 4.6**