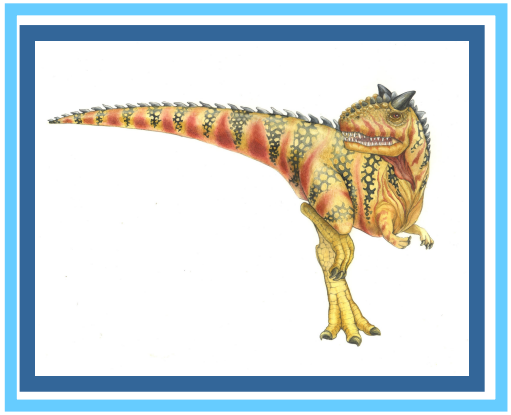
Memoria Virtual (parte 2) Algoritmos de substituicao (cont)



Algoritmo Optimal

- Trocar as paginas que nao serao usadas por maiores periodos de tempo
- Exemplo com 4 quadros

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

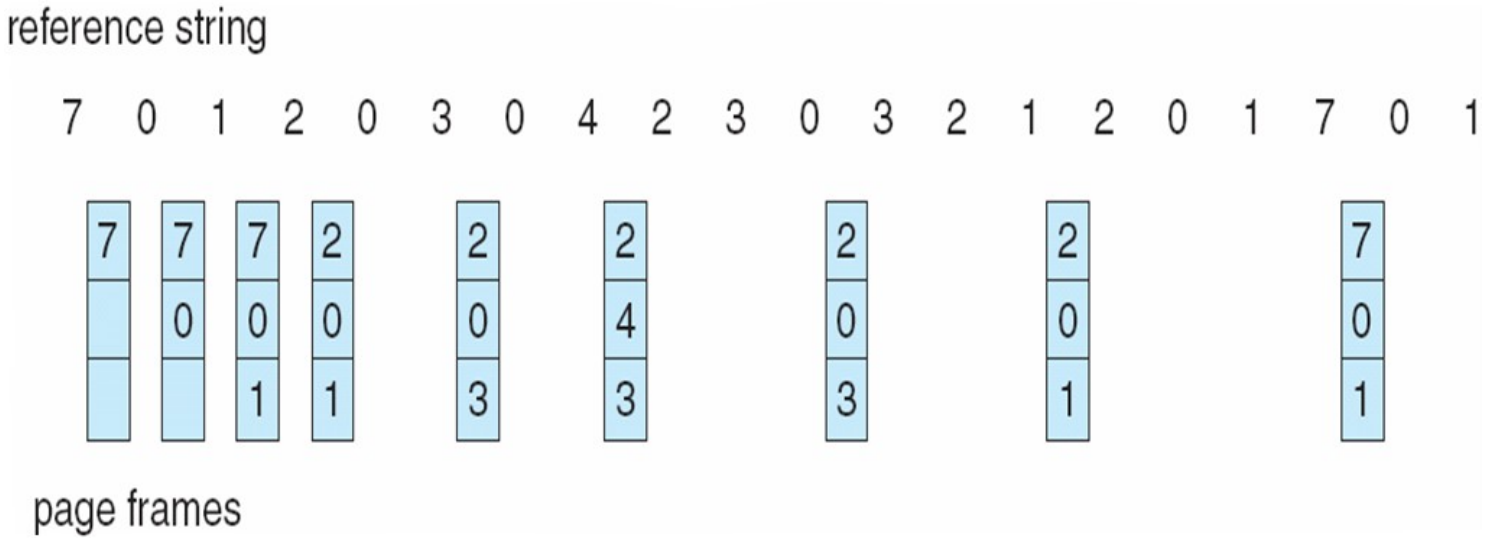
1	4
2	
3	
4	5

6 page faults

- Como sabemos isso ?

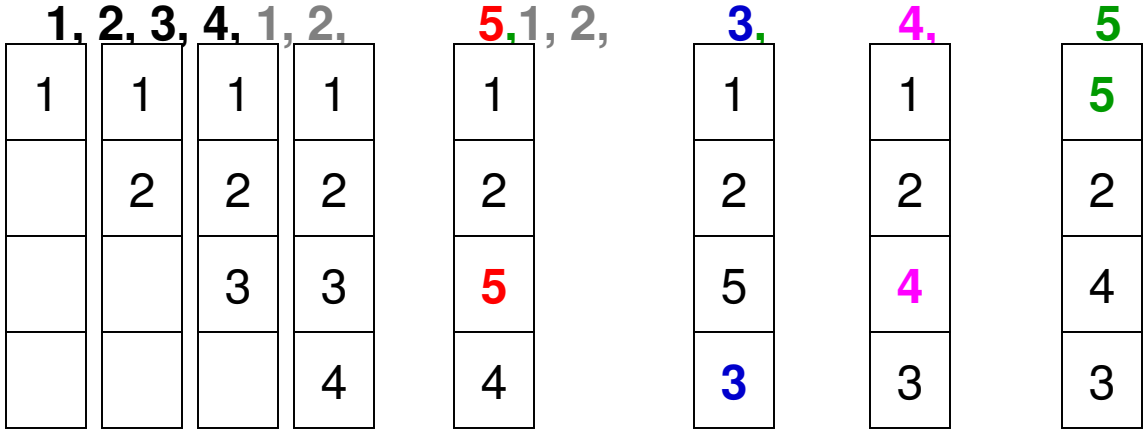
- OBS: esse algoritmo e usado para comparacao com outros
- esse eh o otimo (teorico)

Optimal Page Replacement



Algoritmo Least Recently Used (LRU)

- String de referencia: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



- Implementacao com contador
  - para cada pagina: manter um campo contador (ex: na tab. de paginas)
    - a cada referencia nessa pagina: copiar o contador de clocks nesse campo (pode ser campo na TLB)
  - quando uma pagina precisa ser substituida:
    - substituir a que tem menor valor

Exemplo de substituicao LRU

reference string

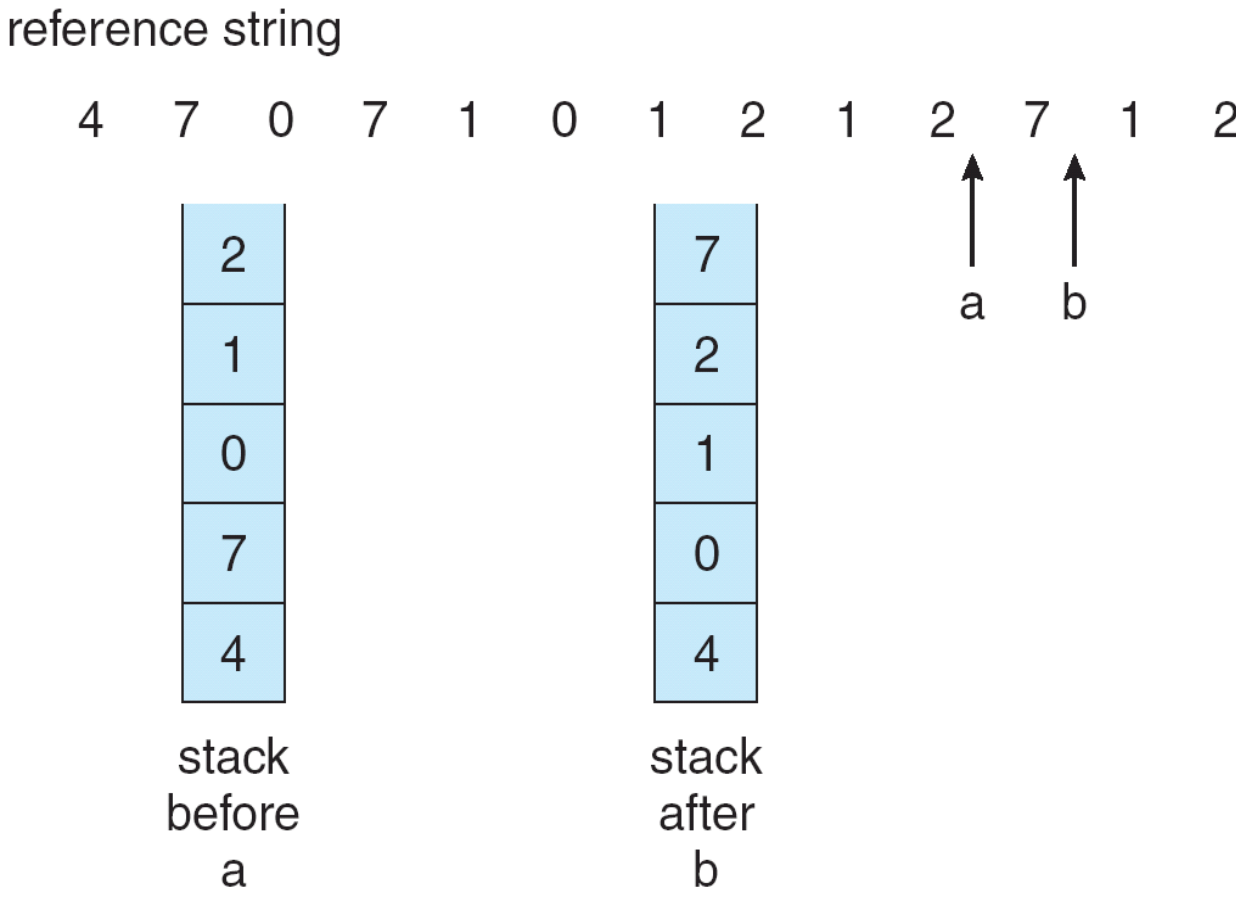
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2		4	4	4	0			1		1		1		
	0	0	0		0		0	0	3	3			3		0		0		
		1	1		3		3	2	2	2			2		2		7		

page frames

Algoritmo LRU (Cont.)

- Implementacao com pilha – manter pilha de numeros de paginas em lista duplamente encadeada:
  - Pagina referenciada:
    - move para o topo
    - requer mudanca em 6 pointers (?menos?)
  - Sem necessidade de pesquisa ==> O(1)

Uso de Pilha para manter as referencias mais recentes



Alocacao Global vs. Local

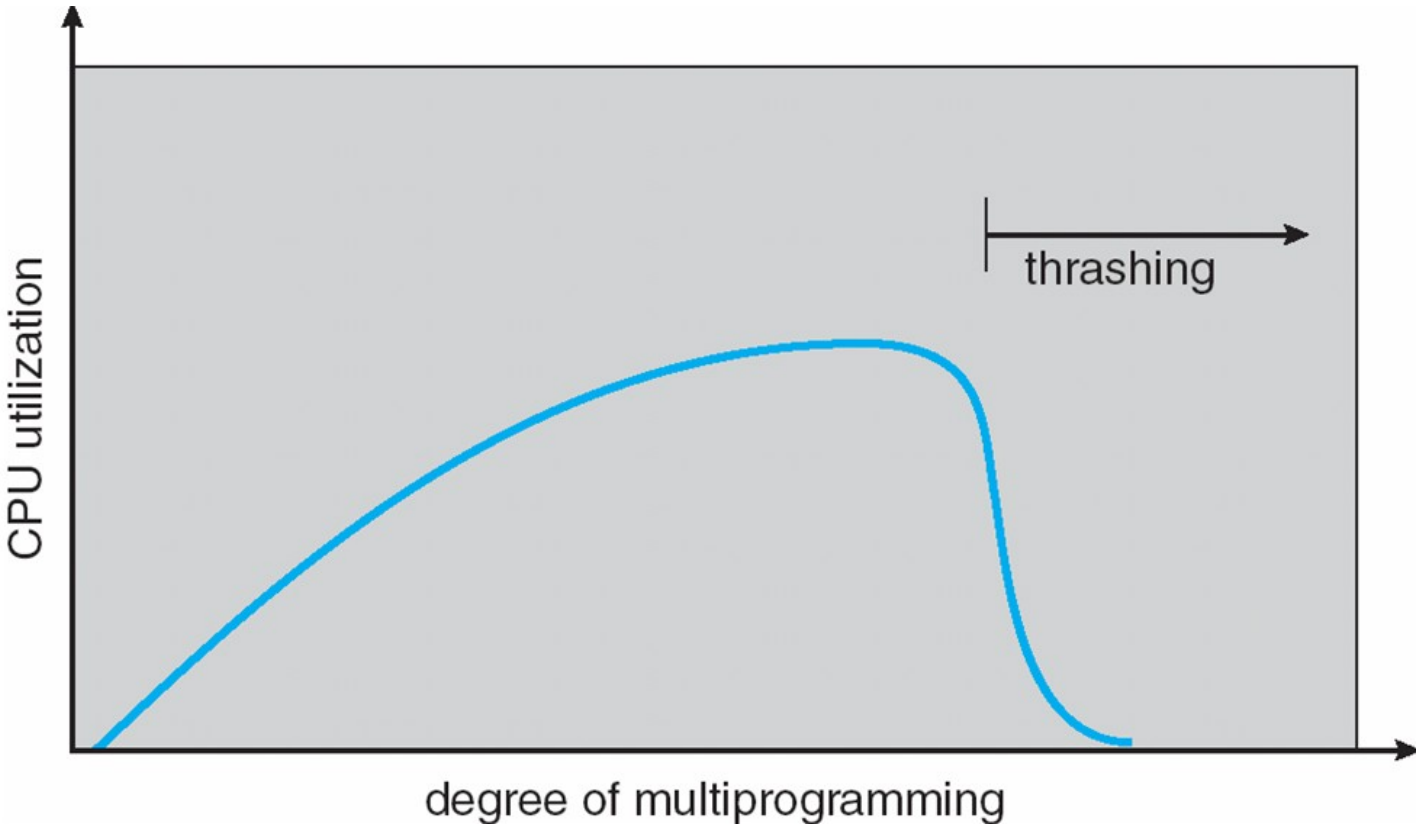
- **Substituicao (alocacao) Global:** alocar (substituir) quadros retirando de qualquer processo
- **Substituicao (alocaco) Local:** substituir somente quadros do mesmo processo gerador da falta (page fault)

Thrashing

- Se processo nao tem bastante paginas ==> aumenta taxa de page faults. Isso leva a:
  - baixa utilizacao de CPU
  - SO pensa que pode aumentar o grau de multipprogramacao
  - mais processos sao admitidos
  - problema piora
- **Thrashing** ≡ sistema ocupa grande parte do tempo fazendo swap de paginas

Thrashing (Cont.)

OBS: esse grafico mostra (apenas) que trashing eh mais provavel quando grau de multiprogramacao fica muito alto



Um **modelo** para entender o trashing:

modelo de conjuntos de trabalho (working sets)

Paginacao por Demanda e Thrashing

- Porque a paginacao por demanda apresenta bons resultados ?
  - Principio de Localidade de referencias a memoria
    (ver grafico na proxima pagina)

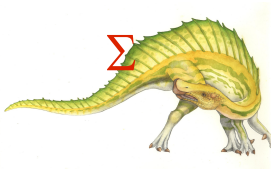
- Porque o trashing ocorre ?
- resumidamente:

  - Processo(s) migra(m) sua(s) localidade(s) de referencia
  - ==> alguma paginacao ocorre se:

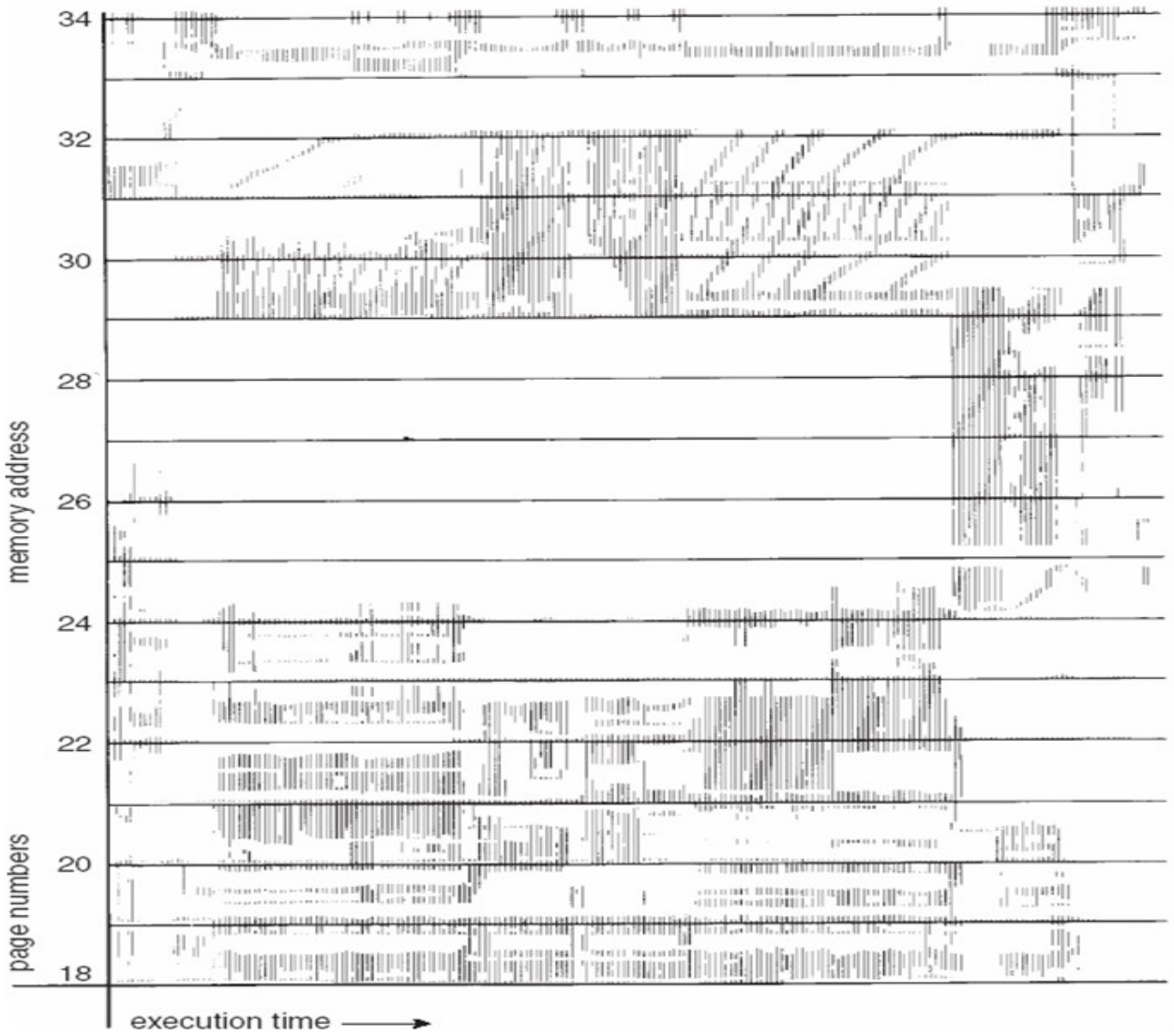
    Σ tamanho da(s) localidade(s) $>$ memoria disponivel

  - ===> MUITA paginacao ocorre se:

    tamanho da(s) localidade(s) >> memoria disponivel



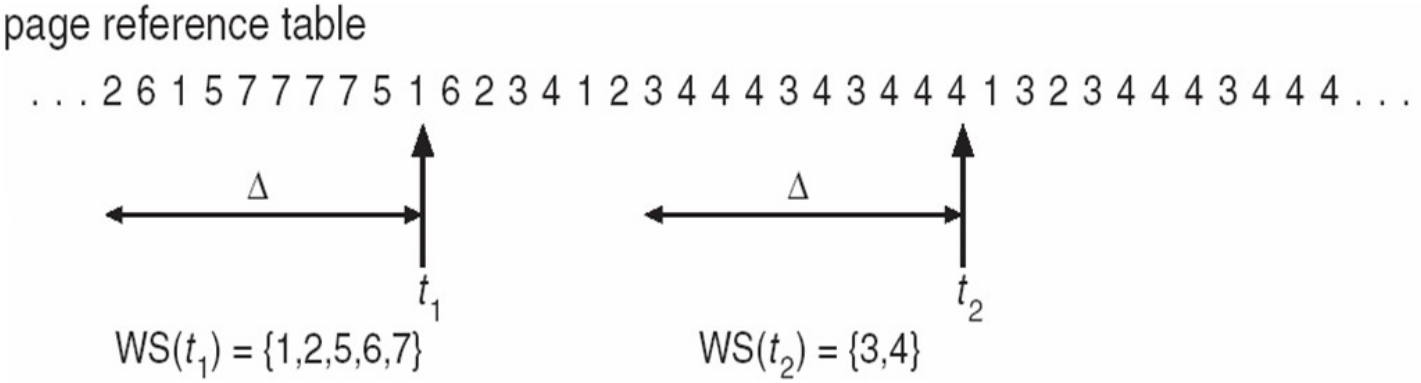
Localidades em um Padrao de referencias a Memoria



Modelo de Working-Sets (conjuntos de trabalho)

- Um **modelo** para entender o fenomeno de trashing:
- $\Delta \equiv$ working-set window $\equiv$ um numero fixo de referencias a paginas (referencias contiguas no tempo, inclusive repetidas)
  Exemplo: 10,000 acessos (memoria de instrucoes e de dados)
- WSS_i (working set size of Process P_i) =
  quantidade de paginas diferentes acessadas em Δ mais recente (varia com o tempo)
  - Se Δ muito pequeno: nao agrega toda a localidade
  - Se Δ muito grande: pega varias localidades
  - Se $\Delta = \infty \Rightarrow$ agrega a localidade de uma execucao do processo (obs: isso eh diferente de todas as localidades possiveis ao programa)
- $D = \Sigma\ WSS_i$ $\equiv$ demanda total de quadros em um Δ
- Seja m = numero de quadros disponiveis
- Se $D > m \Rightarrow$ Thrashing
- Uma politica possivel: Se $D > m$:

Working-set model



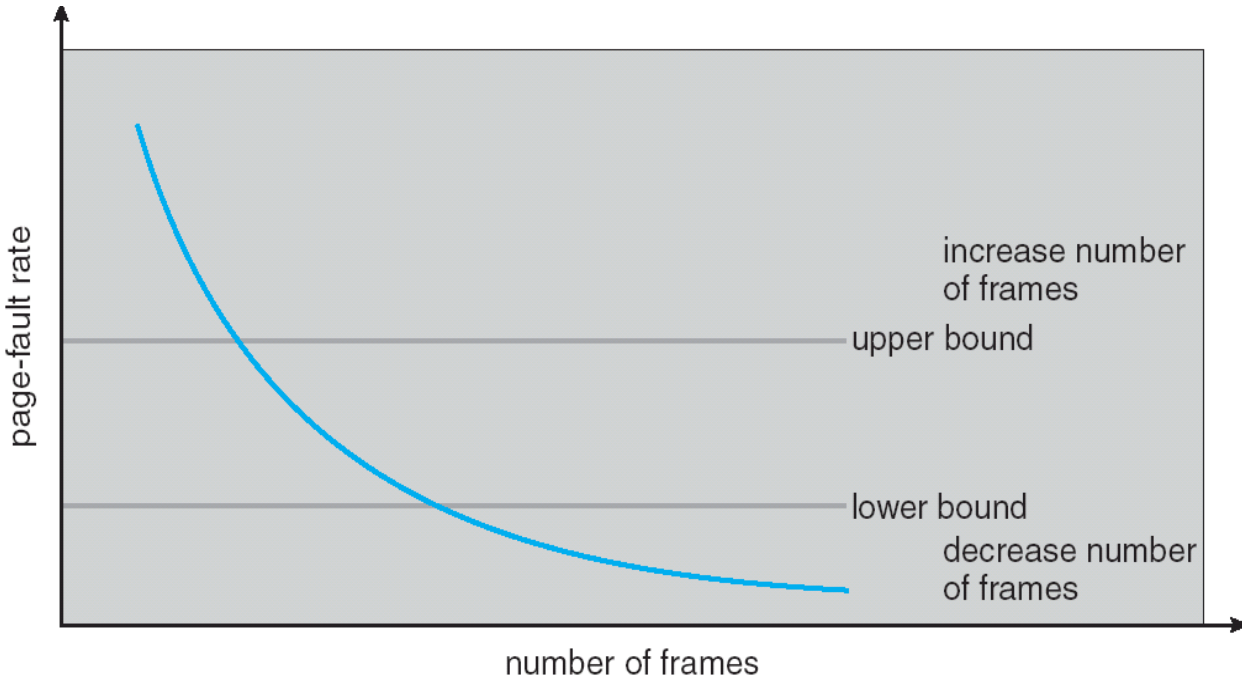
Nesse modelo: como manter informacoes sobre o(s) Working Set(s) ?

- Manter tamanhos exatos de conjuntos de trabalhos eh muito caro
- Fazer aproximacoes usando temporizador de intervalo de tempo (timer) mais bit de referencia (por pagina, em cada entrada da tabela, ou na TLB)
- Exemplo: Δ = 10,000
  - Timer interrompe a cada 5000 unidades de tempo
  - manter (em memoria separada) 2 bits para cada pagina do processo
    - isso simula um contador
  - quando temporizador dispara:
    - copiar valores par o contador correspondente e zerar bits de referencias (de todos as paginas do processo)
  - Se algum bit de referencia bits era = 1 $\Rightarrow$ page esta no conjunto de trabalho

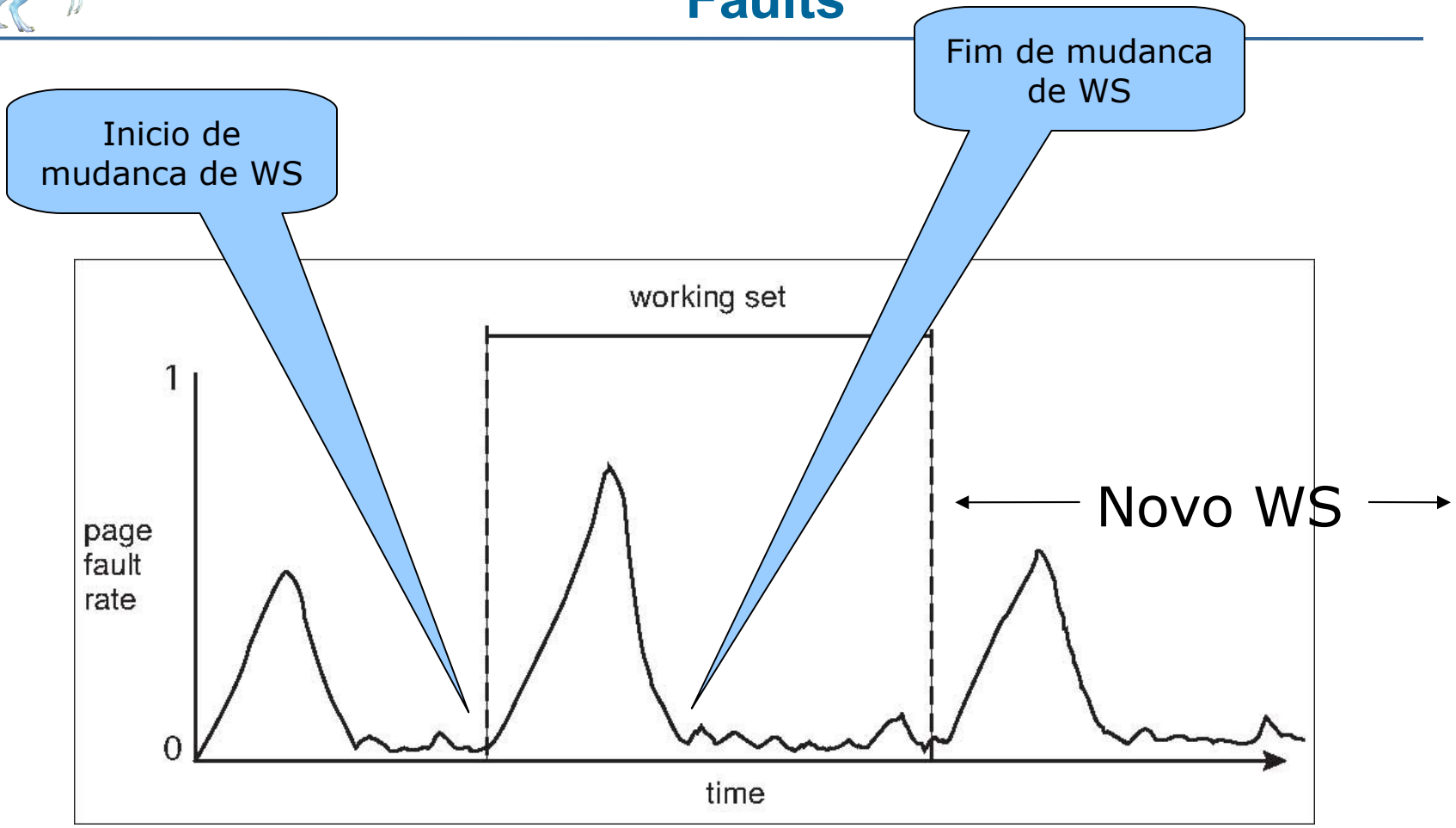
- Porque isso nao determina exatamente o conjunto de trabalho ?
- Melhoramento: usar = 10 bits/contador e interromper a cada 1000 unidades de tempo

Esquema para prevenir de trashing por controle/monitoramento da frequencia Page-Faults

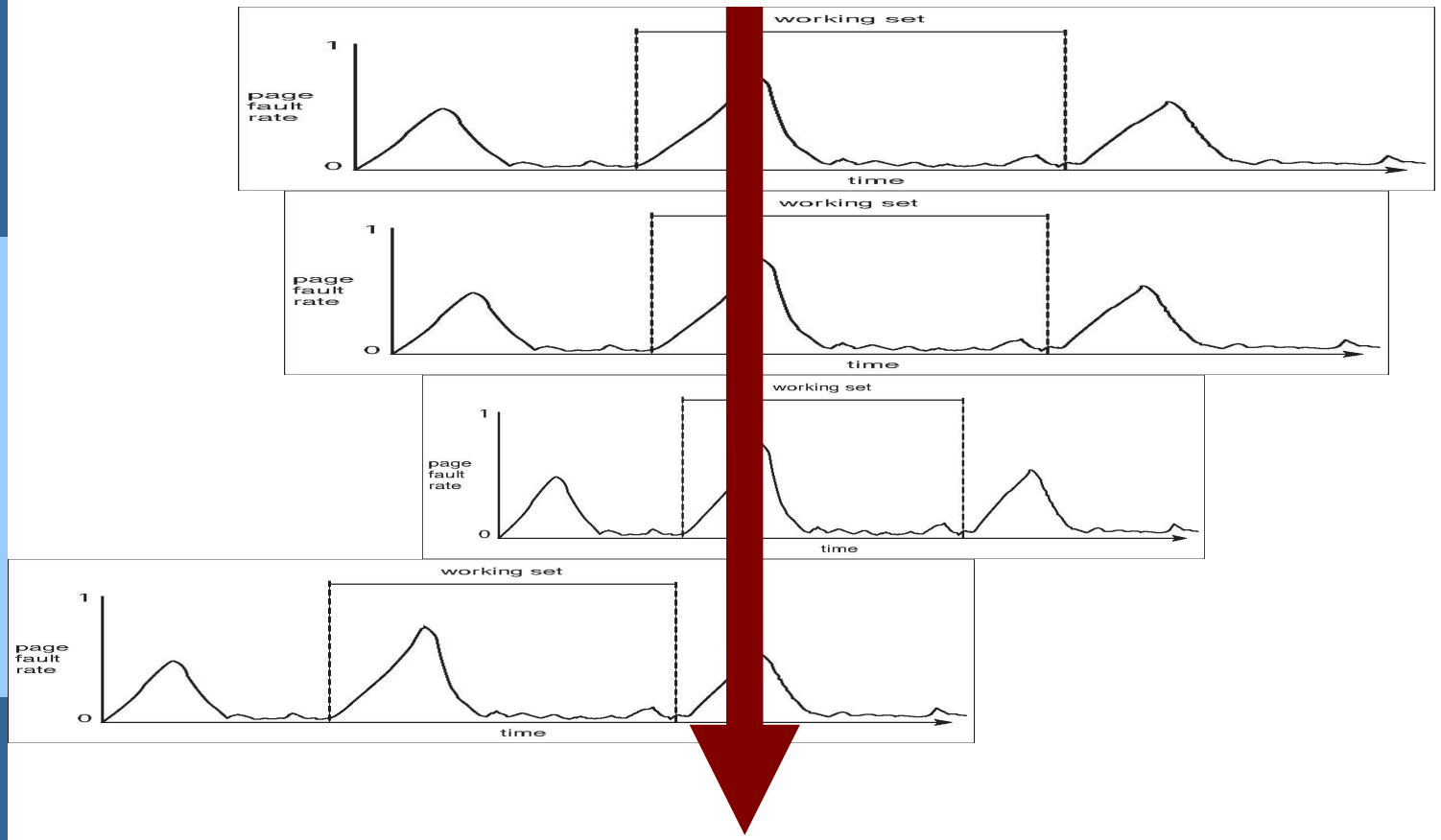
- Metodo de conjunto de trabalho + swap de processos nao eh muito efetivo
- Descrevemos aqui o esquema para prevenir trashing que eh mais efetivo:
  - monitorar a frequencia de page faults por processo
  - controlar alocacao de quadros aos processos que estao com frequencia alta (acima de limiar) ou baixa (abaixo de limiar)
    - Set freq. Pfs muito baixa: processo pode perder quadros
    - Se freq. Pfs muito alta: processo deve ganhar quadros



Relacao entre Working Set(s) e frequencia de Page Faults



Mudancas de Working Set(s), frequencia de Page Faults e Trashing



Processo

P0

P1

P2

P3

- Muita mudanca de WS
- Muita demanda por memoria
- **Muita Paginacao**

==> trashing

Outros aspectos em VM: Prepaginacao

- Prepaginacao
  - Para reduzir o grande numero de page faults no inicio da exec. de (novos) processos
  - buscar antecipadamente (pre-paginar) algumas paginas que do processo, antes mesmo de serem referenciadas
  - Mas SE:
    - paginas buscadas nao sao usadas ==> desperdicio de tempo de I/O e memoria.

Outros aspectos: – Tamanho da Pagina

- Para selecao do tamanho da pagina devemos considerar:
  - fragmentacao (interna)
  - tamanho das tabelas de paginas
  - overhead de I/O (por pagina)
  - localidade

Outros Aspectos – TLB Reach (alcance da TLB)

- Alcance da TLB - Eh a quantidade de memoria que pode ser acessada via TLB (i.e. sem TLB miss)
- TLB Reach = (TLB Size) X (Page Size)
- Idealmente: Cada working set de um processo deve ser menor que o alcance da TLB
  - Senao, mais paginacao ocorre devido a acessos na tabela de paginas
- Aumento do tamanho da pagina
  - aumenta alcance da TLB (ver formula acima!)
  - mas aumenta o custo da fragmentacao
- Uma ideia:
  - Possibilitar multiplos tamanhos de paginas

Outros aspectos – Estrutura de programas

- Exemplo:
  - `int data[128][128];`
  - Cada linha eh armazenada em uma pagina
  - Programa 1

```
for (c = 0; c < 128; c++)
    for (l = 0; l < 128; l++)
        data[l][c] = 0;
```

128 x 128 = 16,384 page faults

  - Programa 2

```
for (l = 0; l < 128; l++)
    for (c = 0; c < 128; c++)
        data[l][c] = 0;
```

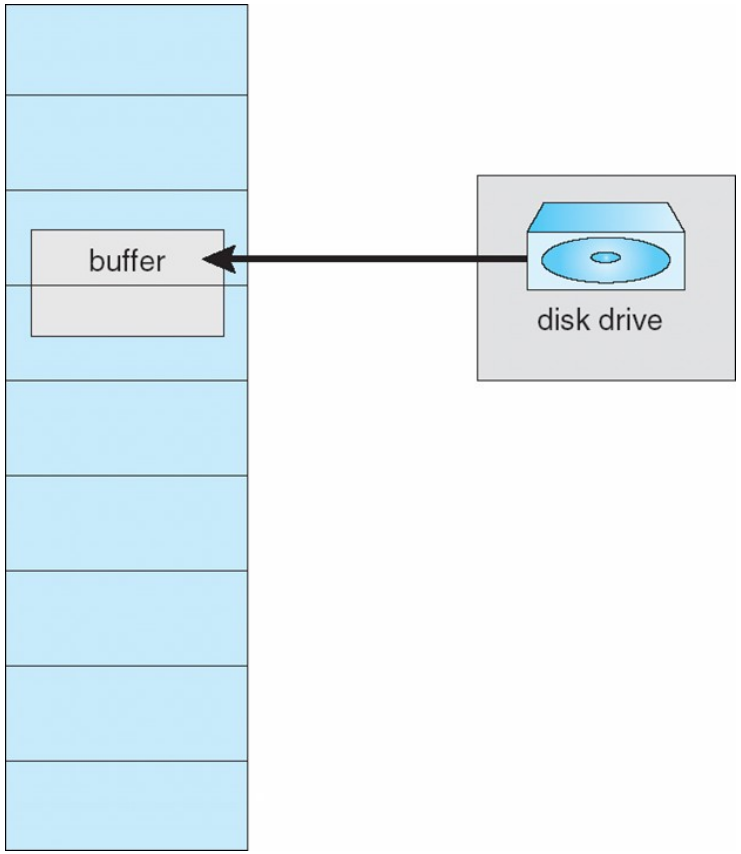
128 page faults

Outros aspectos – fixar paginas para I/O

- **I/O Interlock** – Algumas Paginas devem ser fixadas (presas, pinned) em memoria para trabalho com dispositivos de I/O

- So nao deve deixar sistema de paginacao (algoritmos de substituicao de paginas) utilizar tais paginas
- Considere o funcionamento de dispositivos de I/O, por exemplo com DMA
- Paginas que sao usadas para transferencias sao usadas pelo sistema de DMA de maneira assincrona

Paginas presas para I/O (pinned memory)



Fim Memoria Virtual