Sistemas Operacionais

Gestão de memória - Paginação em disco

Prof. Carlos Maziero

DInf UFPR, Curitiba PR

Abril de 2019

Conteúdo

Uso da memória

Memória é um recurso precioso:

- Programas cada vez maiores
- Mais processos na memória
- Memória RAM é cara (~ U$10 / GByte)

Contudo:

- Nem todos os processos estão ativos
- Nem todas as áreas de memória são usadas
- Discos são baratos (~ U$0,05-0,30 / GByte)

Ideia: usar um espaço em disco como **extensão** da memória

Estendendo a memória RAM

Conceito:

- Transferir partes da RAM para o disco
- Buscar o conteúdo no disco quando necessário
- Usar a MMU para tornar tudo transparente

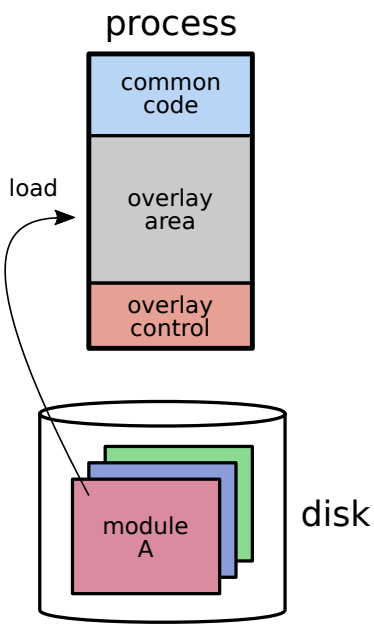
Abordagens:

- **Overlays**
- **Swapping**
- **Paging**

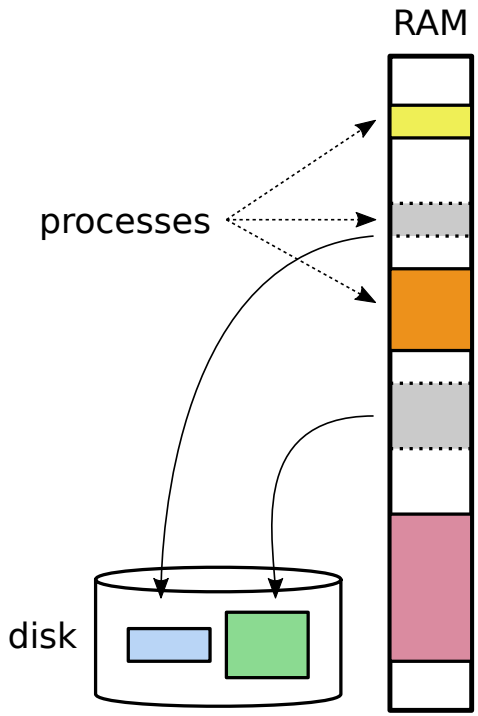
Abordagens

- **Overlays**:
  - executável dividido em módulos
  - somente um módulo é carregado na memória
  - gerência a cargo do programador
- **Swapping**:
  - processos suspensos saem da memória para o disco
  - voltam para a memória ao acordar
- **Paging**:
  - páginas pouco usadas saem da memória
  - *page faults* trazem as páginas de volta

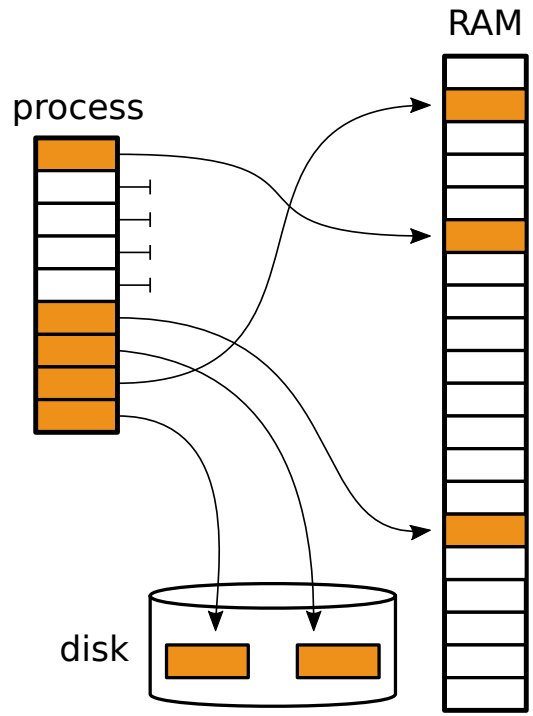
Abordagens



overlay



swapping



paging

Paginação em disco

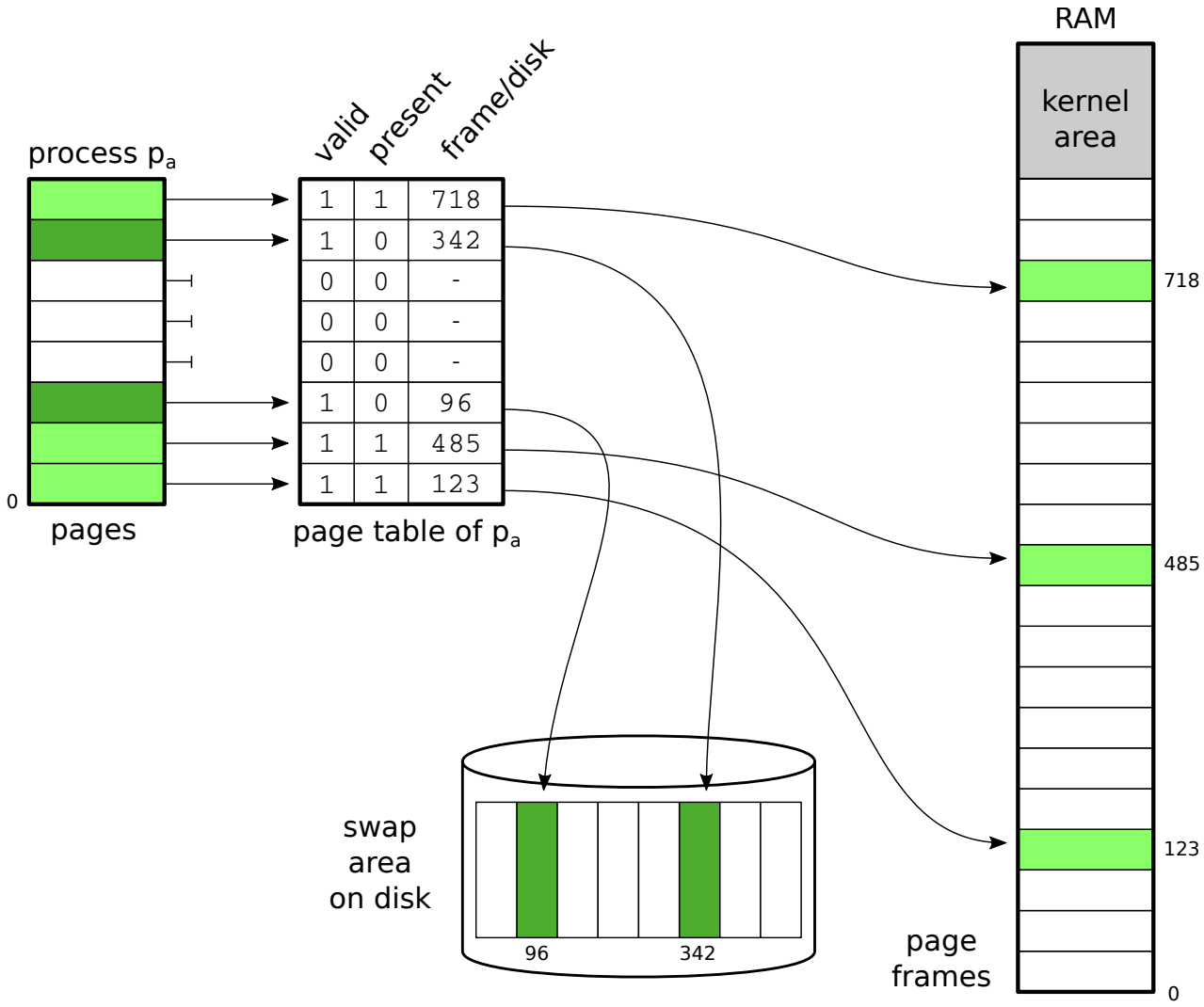
Ideia central:

- Mover páginas ociosas da RAM para o disco
- Ajustar tabela de páginas para indicar isso
- Se página for acessada, gerar *page fault*
- Trazer página de volta para a RAM

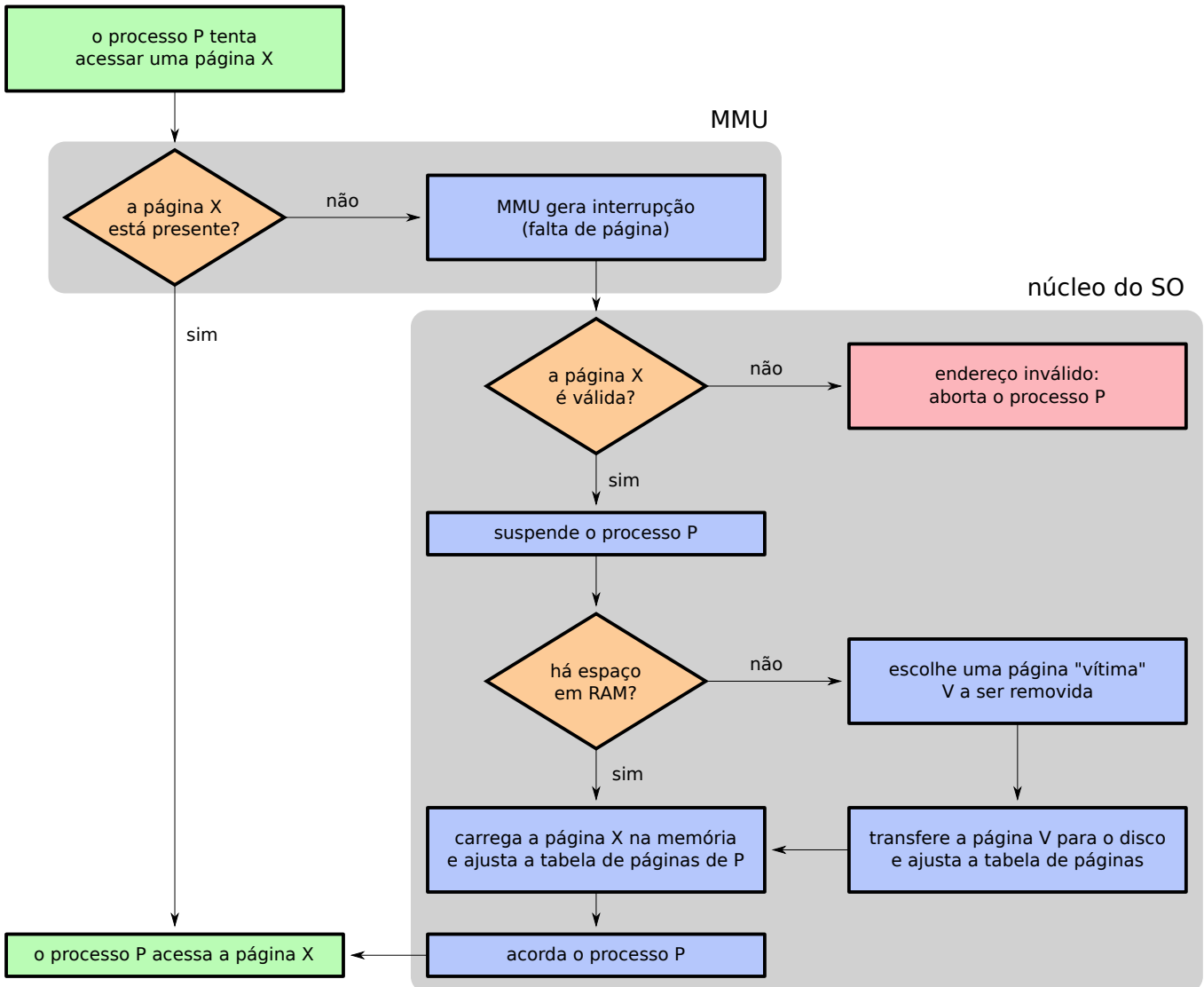
Implementação:

- Núcleo escolhe páginas ociosas
- Núcleo move as páginas entre RAM e disco
- MMU detecta acessos inválidos

Paginação em disco



Dinâmica da paginação



Desempenho da memória virtual

Problema: discos são **lentos**: $t_{disk} \gg t_{ram}$

Considerando $t_{disk} = 6ms$ e $t_{ram} = 60ns$

- uma falta de página em 1.000.000 acessos à RAM:

$$\begin{aligned} t_{medio} &= \frac{(999.999 \times 60ns) + 6ms + 60ns}{1.000.000} \\ &= 66ns \end{aligned}$$

- uma *fp* em 100.000 acessos $\longrightarrow t_{medio} = 120ns$

Frequência de faltas de páginas

Fatores que a influênciam:

- O **tamanho da memória** em relação à demanda
- A forma como os processos **acessam** a memória
- A **escolha das páginas** a tirar da memória

Escolha das páginas:

- definir páginas “vítimas” a retirar
- Algoritmos de **substituição** de páginas

Fatores a considerar

- Idade da página na memória
- Frequência de acessos à página
- Data do último acesso
- Prioridade do processo
- Conteúdo da página (código ou dados)
- Páginas especiais (*buffers*, conteúdos sensíveis)
- ...

Cadeia de referências

- Sequência de páginas acessadas durante uma execução
- Usadas para estudar algoritmos de substituição
- Cadeias reais são muito longas (milhões refs/segundo)

Cadeia usada neste texto:

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

Algoritmos clássicos

ÓTIMO

- Melhor solução teórica
- Impossível de implementar

FIFO - *First-In, First-Out*

- Solução mais simples

LRU - *Least Recently Used*

- Melhor solução implementável

Algoritmo Ótimo

Critério

Retirar a página que ficará **mais tempo** sem ser acessada.

Problema

Como prever o comportamento futuro dos processos?

$\longrightarrow$ o algoritmo ótimo **não é implementável** !

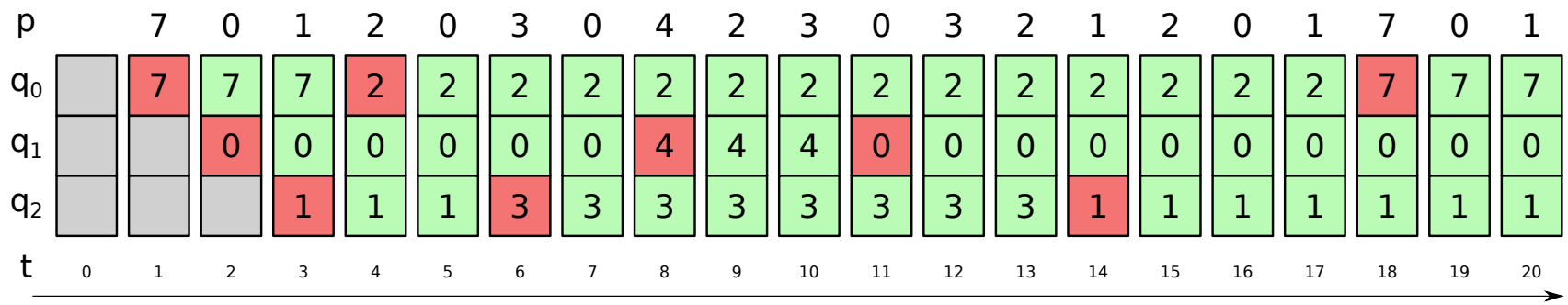
Mas ele serve como limite conceitual:

> *Se o algoritmo ótimo gera N faltas de página, nenhum algoritmo irá gerar $M < N$ faltas na mesma situação.*

Algoritmo Ótimo

t	página	quadros q_o	q_1	q_2	fp	ação realizada
0						situação inicial, quadros vazios
1	7	7			✓	p_7 é carregada em q_0
2	0	7	0		✓	p_0 é carregada em q_1
3	1	7	0	1	✓	p_1 é carregada em q_2
4	2	2	0	1	✓	p_2 substitui p_7 (só é usada em $t = 18$)
5	0	2	0	1		p_0 já está na memória
6	3	2	0	3	✓	p_3 substitui p_1
7	0	2	0	3		p_0 já está na memória
8	4	2	4	3	✓	p_4 substitui p_0
						...

Algoritmo Ótimo



O algoritmo ótimo gera 9 faltas de página.

Algoritmo FIFO

Ideia:

- Considera o tempo em que as páginas estão na memória
- Retirar da RAM as páginas mais antigas

Implementação: uma **fila de números** de páginas em RAM

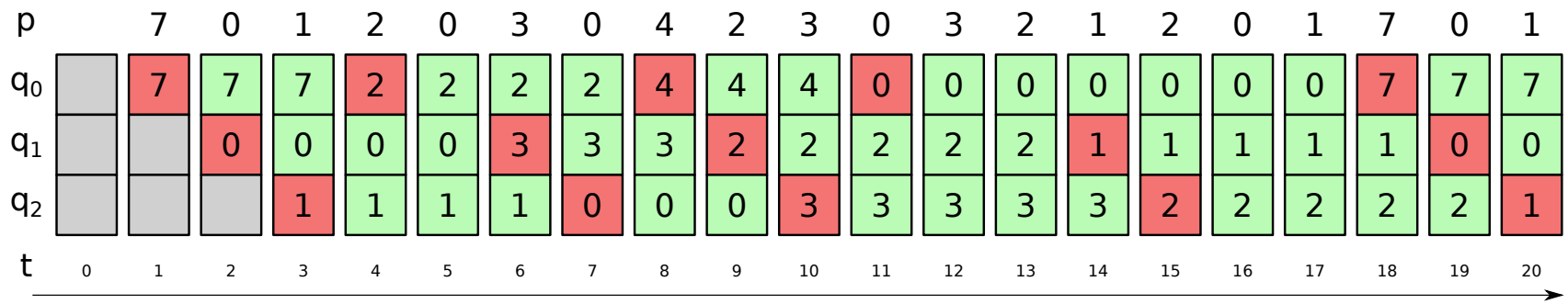
Problemas:

- Não considera o **uso** das páginas
- Não oferece bons resultados
- Processos antigos podem sofrer mais

Algoritmo FIFO

t	página	quadros q_o	q_1	q_2	fp	ação realizada
0						situação inicial, quadros vazios
1	7	7			✓	p_7 é carregada em q_0
2	0	7	0		✓	p_0 é carregada em q_1
3	1	7	0	1	✓	p_1 é carregada em q_2
4	2	2	0	1	✓	p_2 substitui p_7 (carregada em $t = 1$)
5	0	2	0	1		p_0 já está na memória
6	3	2	3	1	✓	p_3 substitui p_0
7	0	2	3	0	✓	p_1 substitui p_1
8	4	4	3	0	✓	p_4 substitui p_2
						...

Algoritmo FIFO



O algoritmo FIFO gera 15 faltas de página.

Algoritmo LRU

Localidade de referências:

- páginas com uso recente podem ser usadas em breve

Ideia:

- Retirar da RAM as páginas há mais tempo **sem uso**
- *Least Recently used*

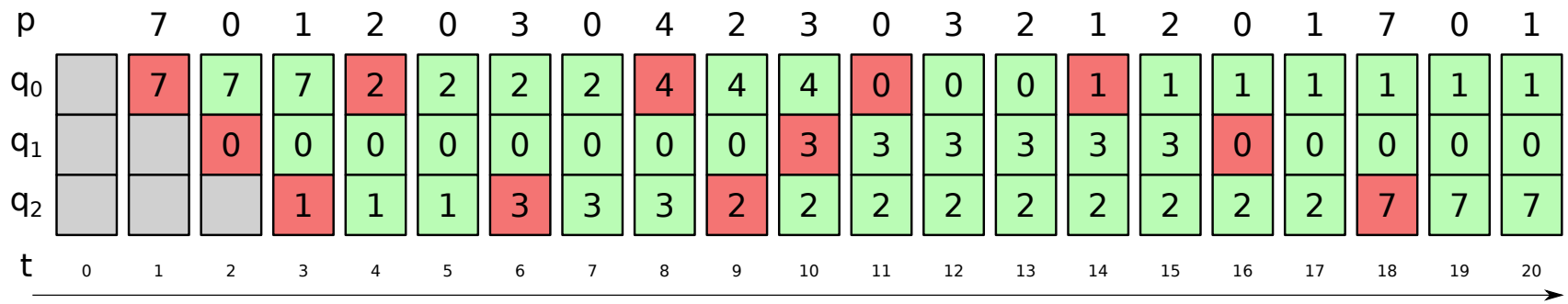
O LRU é simétrico ao OPT em relação ao tempo:

> *enquanto o OPT busca as páginas que serão acessadas "mais longe" no futuro, o algoritmo LRU busca as páginas que foram acessadas "mais longe" no passado.*

Algoritmo LRU

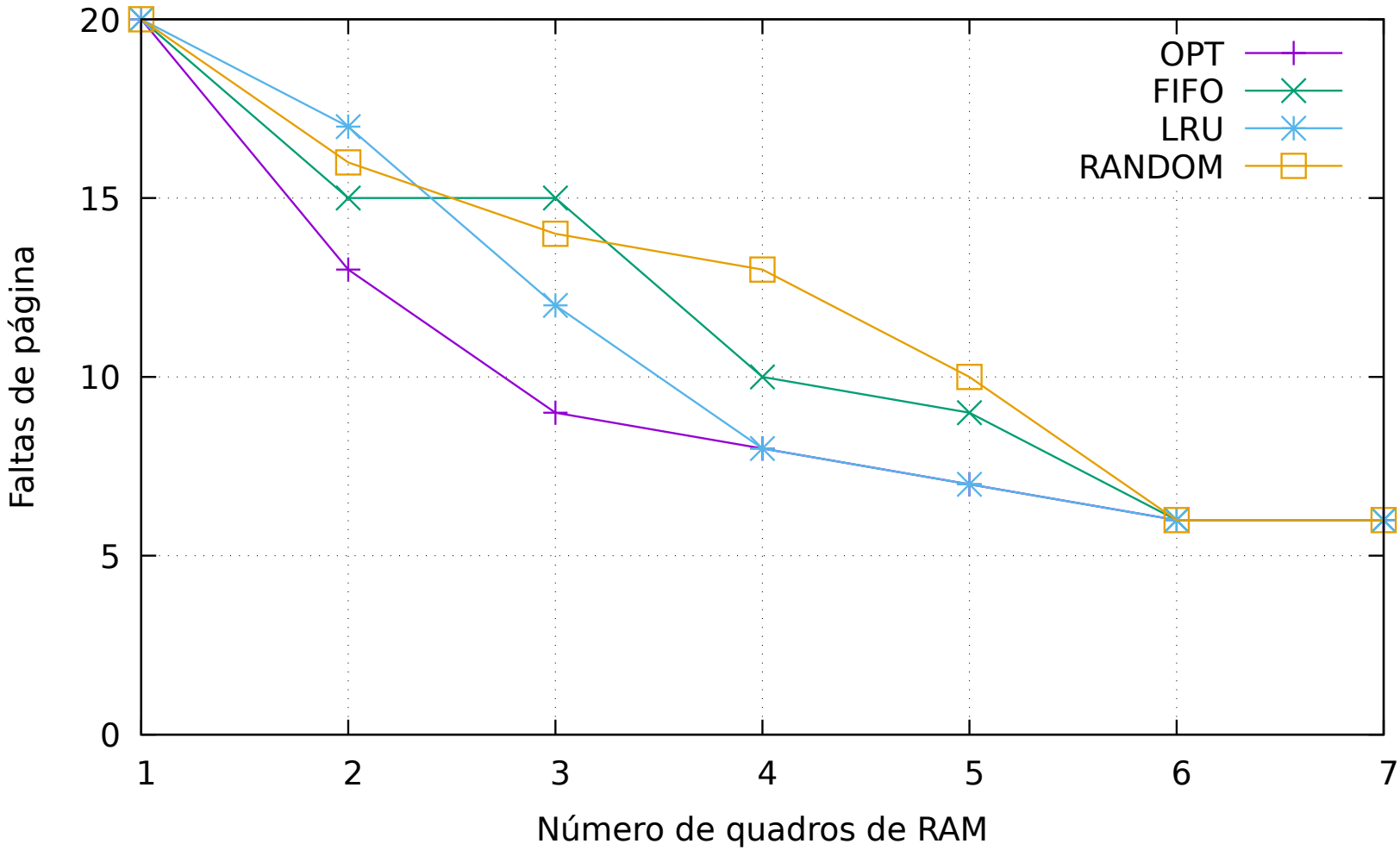
t	página	quadros q_o	q_1	q_2	fp	ação realizada
0						situação inicial, quadros vazios
1	7	7			✓	p_7 é carregada em q_0
2	0	7	0		✓	p_0 é carregada em q_1
3	1	7	0	1	✓	p_1 é carregada em q_2
4	2	2	0	1	✓	p_2 substitui p_7 (mais tempo sem uso)
5	0	2	0	1		p_0 já está na memória
6	3	2	0	3	✓	p_3 substitui p_1
7	0	2	0	3		p_0 já está na memória
8	4	4	0	3	✓	p_4 substitui p_2
						...

Algoritmo LRU

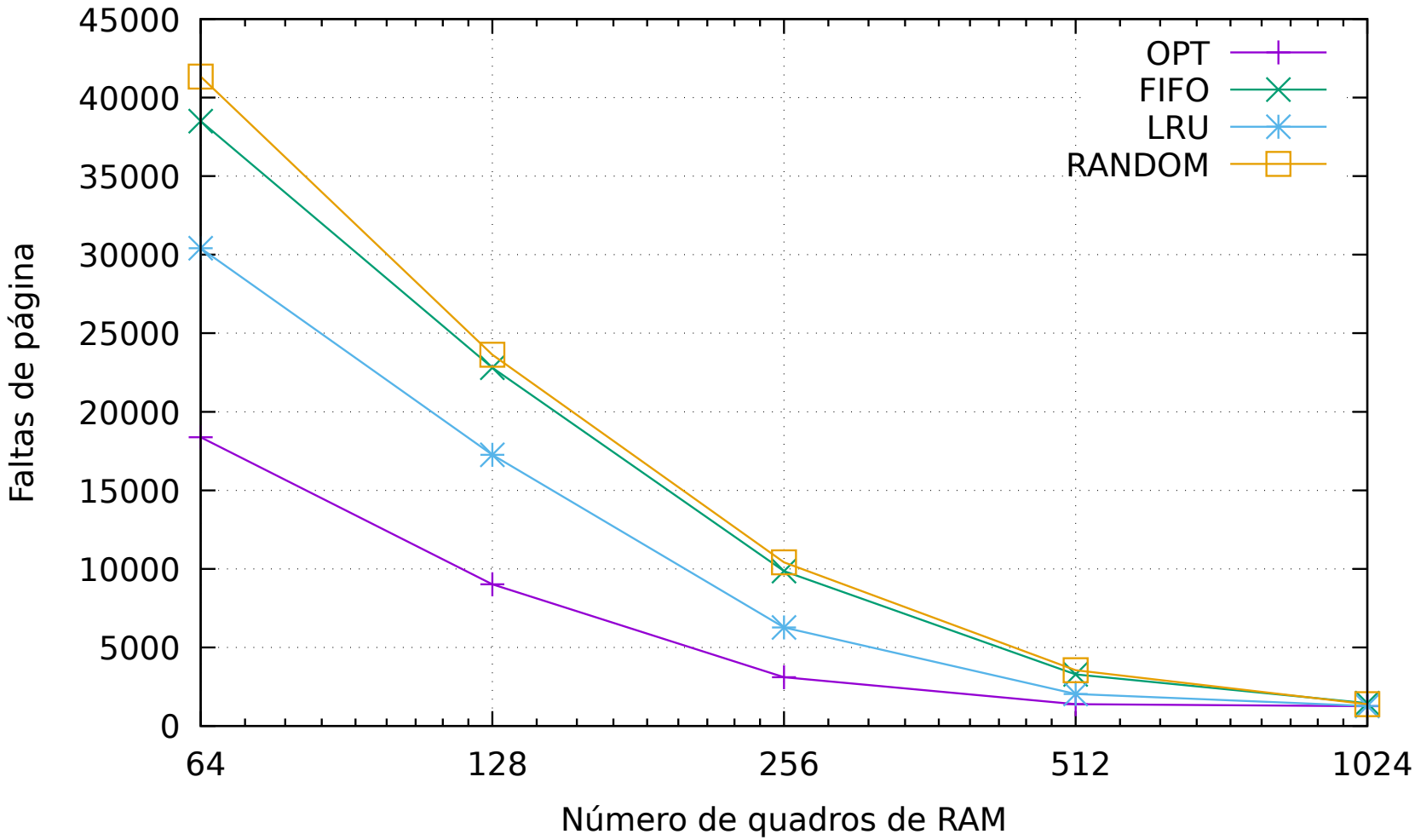


O algoritmo LRU gera 12 faltas de página.

Comparativo: string de referência



Comparativo: compilador GCC



Aproximações do algoritmo LRU

LRU completo é inviável:

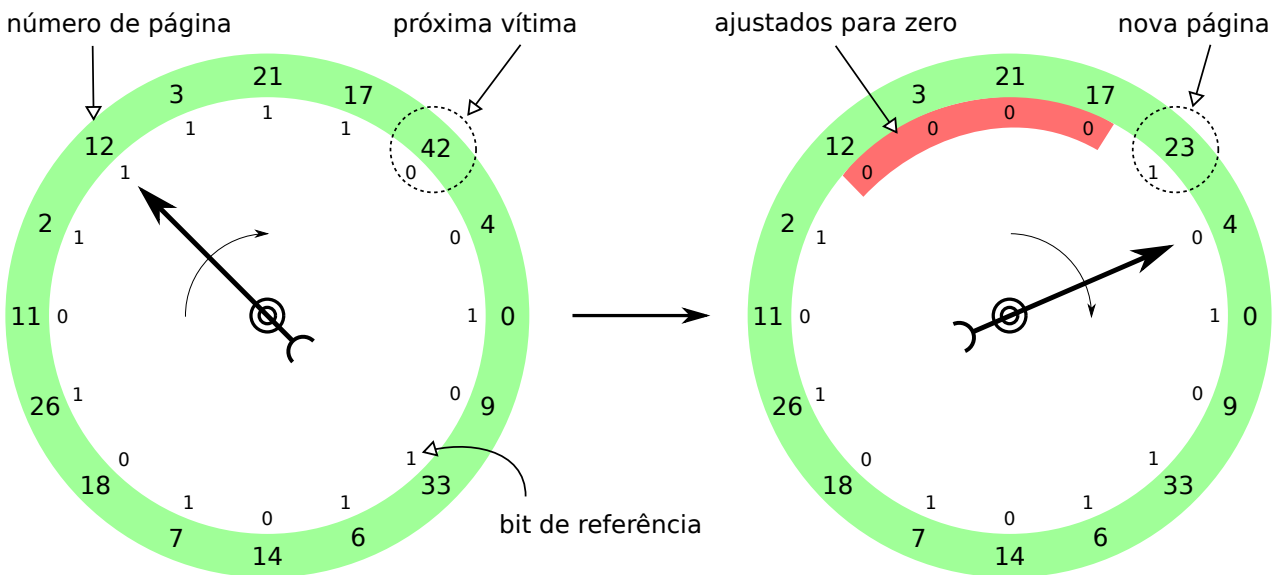
- registrar as datas de acesso às páginas a cada acesso
- encontrar a página com data de acesso mais antiga

Aproximações do LRU usadas na prática:

- Algoritmo do relógio
- *Not-Recently Used*
- Envelhecimento
- *Working sets*

Algoritmo do relógio (ou da segunda chance)

- Baseado no algoritmo FIFO
- Analisa o bit de referência de cada página candidata
- Dá “Segunda chance” às páginas usadas recentemente
- Evita substituir páginas antigas mas muito acessadas



Algoritmo *Not Recently Used* - NRU

Usa flags de referência (R) e modificação(M) das páginas.

Bits R e M defiem níveis de importância:

R	**M**	status
0	0	nem usada nem modificada; melhor escolha!
0	1	sem uso mas alterada (salvar antes de remover)
1	0	usada recentemente
1	1	usada recentemente e alterada, pior escolha!

Algoritmo: substituir primeiro páginas do nível mais baixo (00).

Algoritmo do Envelhecimento

Ideia: usar o bit R para construir *contadores de acesso*:

- Um contador inteiro com N bits para cada página
- O bit R é usado para atualizar o contador
- Periodicamente, para cada página:
  1. os bits do contador são deslocados para a direita
  2. valor do bit R é usado como novo *Most Significant Bit*
  3. O bit R é ajustado para zero
- Acessos mais recentes têm maior peso

$$\overbrace{(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)}^{contador}, R \rightarrow \overbrace{(R, b_7, b_6, b_5, b_4, b_3, b_2, b_1)}^{contador}, 0$$

Algoritmo do Envelhecimento

$$
\begin{array}{c}
\\
p_1 \\
p_2 \\
p_3 \\
p_4
\end{array}
\begin{bmatrix} R \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}
\begin{bmatrix} \textit{contadores} & \\ 0000\ 0011 & (3) \\ 0011\ 1101 & (61) \\ 1010\ 1000 & (168) \\ 1110\ 0011 & (227) \end{bmatrix}
\Longrightarrow
\begin{bmatrix} R \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}
\begin{bmatrix} \textit{contadores} & \\ 0000\ 0001 & (1) \\ 1001\ 1110 & (158) \\ 0101\ 0100 & (84) \\ 1111\ 0001 & (241) \end{bmatrix}
$$

Conjunto de trabalho

Definição:

Conjunto de páginas acessadas recentemente pelo processo.

Propriedades:

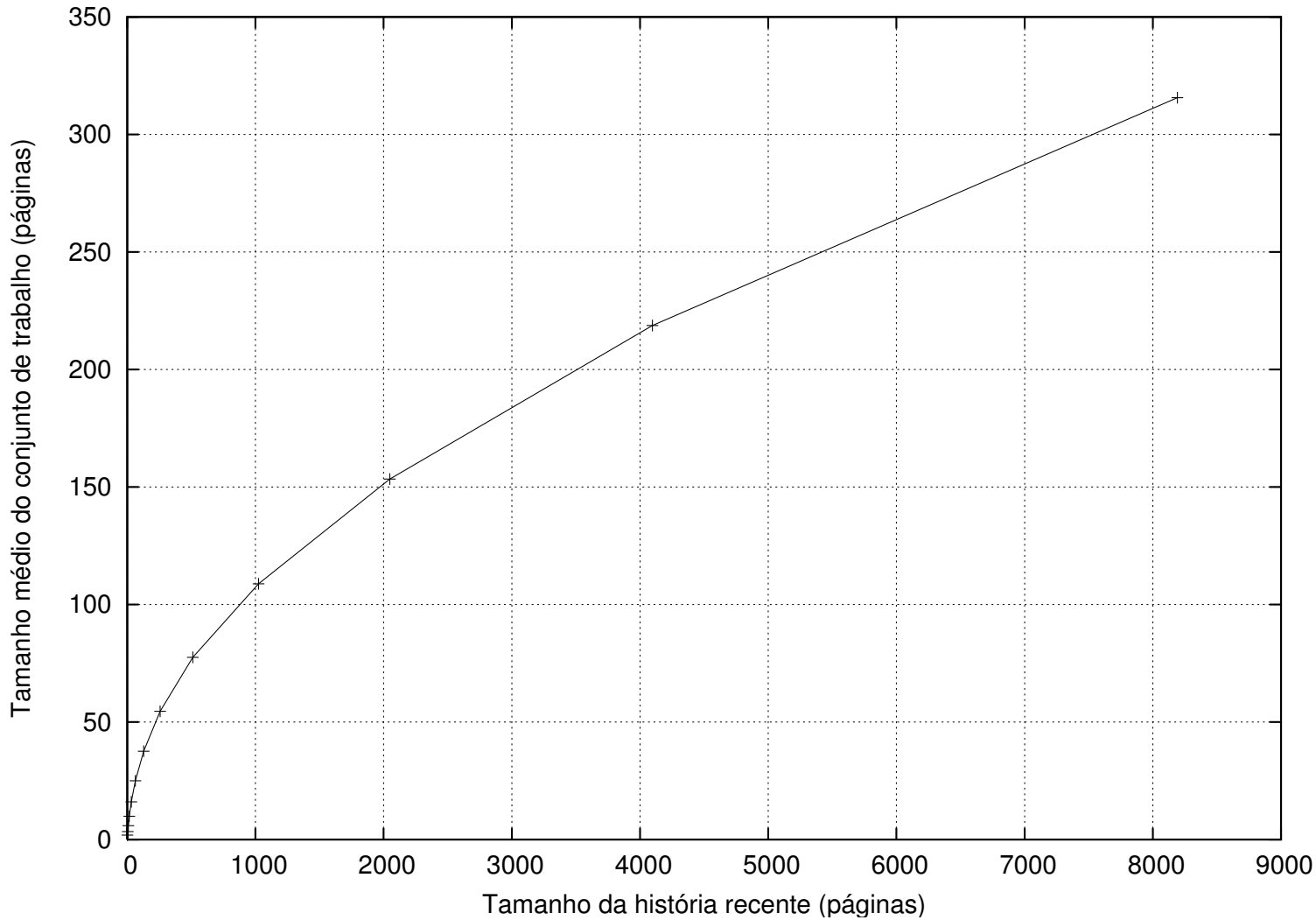
- Geralmente pequeno, devido à localidade de referências
- Dinâmico, varia com a evolução do processo
- Se o processo tiver seu conjunto de trabalho na memória, terá poucas faltas de página

Algoritmo: substituir páginas que não pertençam ao conjunto de trabalho de nenhum processo ativo (difícil de implementar)

Conjuntos de trabalho

t	**página**	history = 3	history = 4	history = 5
1	7	7	7	7
2	0	0, 7	0, 7	0, 7
3	1	0, 1, 7	0, 1, 7	0, 1, 7
4	2	0, 1, 2	0, 1, 2, 7	0, 1, 2, 7
5	0	0, 1, 2	0, 1, 2	0, 1, 2, 7
6	3	0, 2, 3	0, 1, 2, 3	0, 1, 2, 3
7	0	0, 3	0, 2, 3	0, 1, 2, 3
8	4	0, 3, 4	0, 3, 4	0, 2, 3, 4
9	2	0, 2, 4	0, 2, 3, 4	0, 2, 3, 4
10	3	2, 3, 4	0, 2, 3, 4	0, 2, 3, 4
11	0	0, 2, 3	0, 2, 3, 4	0, 2, 3, 4
12	3	0, 3	0, 2, 3	0, 2, 3, 4

Conjunto de trabalho do Gnome GThumb



Algoritmo Working Set Clock (WSClock)

Usar *Working Sets* diretamente é caro.

WSClock: aproximação de WS usando o algoritmo do relógio:

- Cada página tem uma data de último acesso $t_a(p)$
- Cada página tem uma idade $i(p) = t_{now} - t_a(p)$
- $t_a(p)$ é atualizada quando o ponteiro do relógio a visita
- As páginas têm um prazo de validade τ

Ideia do algoritmo:

- visitar as páginas usando o algoritmo do relógio
- remover as páginas não-acessadas e fora do prazo de validade

Algoritmo Working Set Clock (WSClock)

O ponteiro percorre a fila e analisa cada página p:

1. Se $R(p) = 1$ (a página foi acessada):
   - $t_a(p) = t_{now}$ (atualiza a data de acesso)
   - $R(p) = 0$ (limpa o bit de referência)
2. Se $R(p) = 0$:
   - se $i(p) \leq \tau$, a página está no conjunto de trabalho
   - se não estiver:
     - Se $M(p) = 0$ a página pode ser removida
     - Se $M(p) = 1$ agenda-se a escrita da página
3. Se não achar página com $i(p) > \tau$, $R(p) = 0$ e $M(p) = 0$:
   1. remover a página mais antiga com $M(p) = 0$ e $R(p) = 0$
   2. ... (usando NRU)

A anomalia de Belady

Intuição: quanto mais RAM, menos faltas de páginas

Problema:

- Esse comportamento **não se verifica** sempre
- Alguns algoritmos têm comportamento anômalo

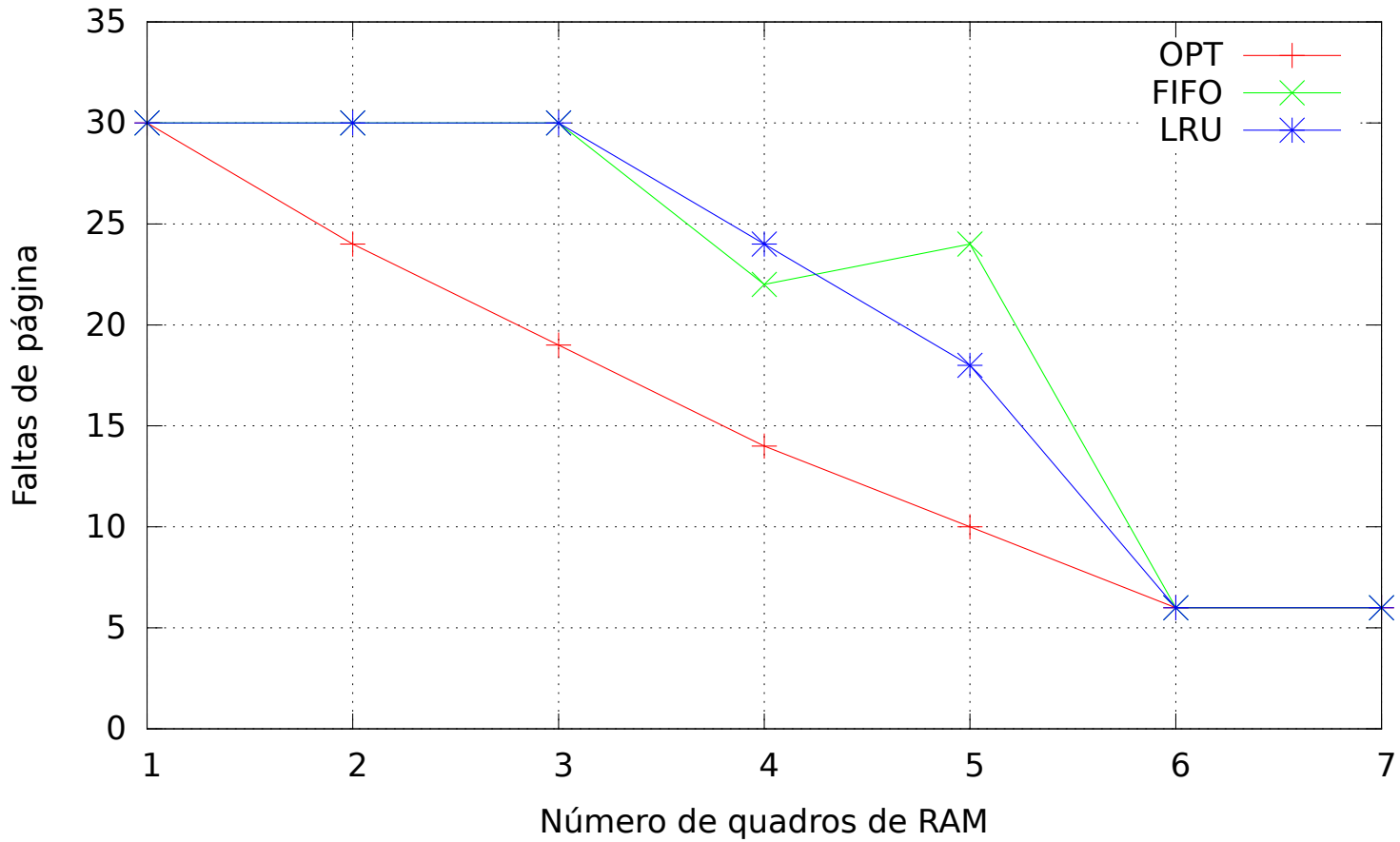
Anomalia de Belady

Ao aumentar o número de quadros de RAM, o número de faltas de página **aumenta**, ao invés de diminuir.

Exemplo de cadeia de referências:

0, 1, 2, 3, 4, 0, 1, 2, 5, 0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 0, 1, 2, 5, 0, 1, 2, 3, 4, 5

A anomalia de Belady



Thrashing

Saturação no uso da RAM:

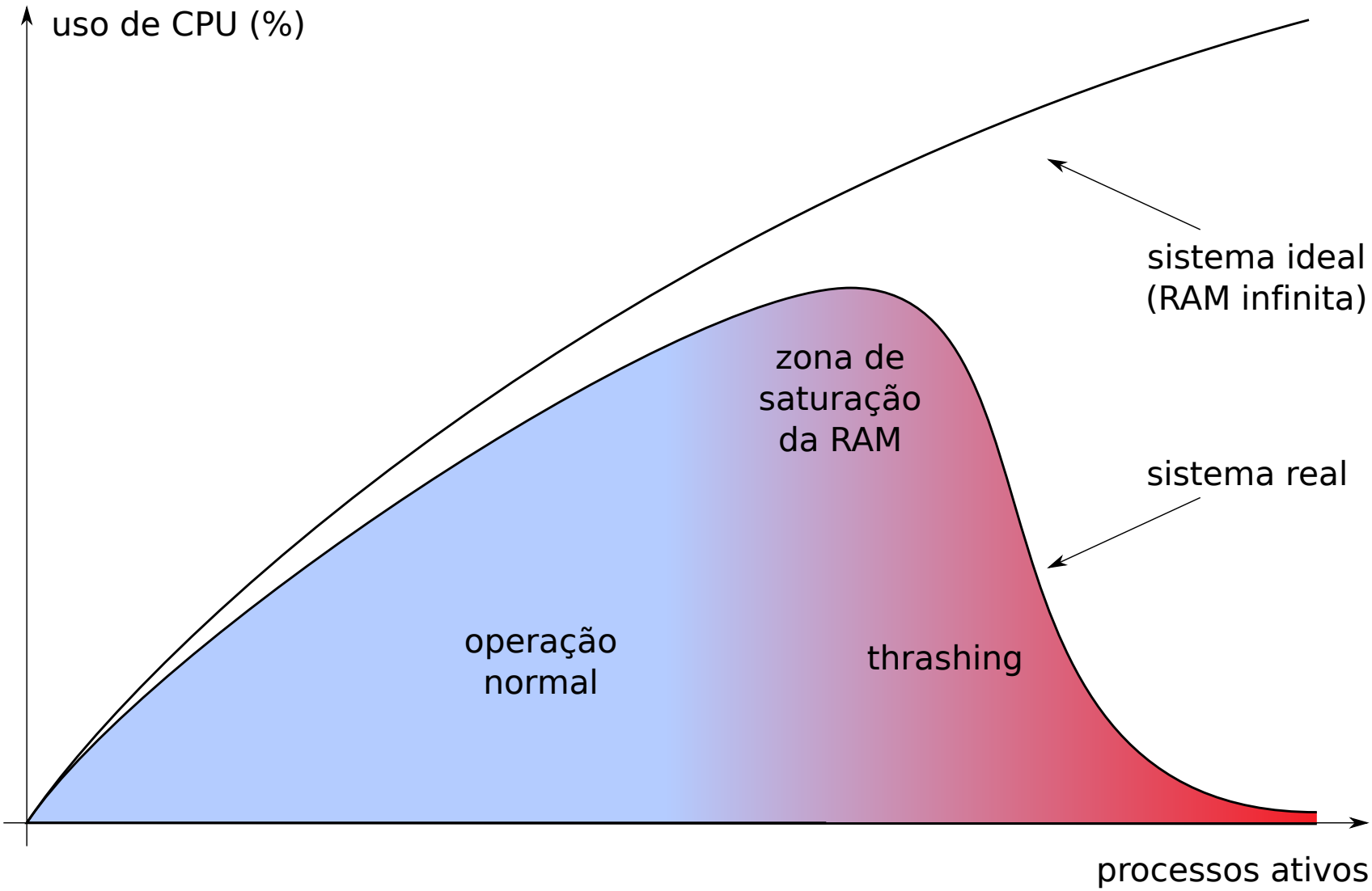
- Muitos processos ativos
- Os processos não têm RAM para seus WSets
- Cada processo gera muitas faltas de página

Consequências:

- Processos ficam suspensos a maior parte do tempo
- A paginação toma a maior parte do tempo do sistema

Solução: diminuir o número de processos ativos

Thrashing


uso de CPU (%)
sistema ideal
(RAM infinita)
zona de
saturação
da RAM
sistema real
operação
normal
thrashing
processos ativos