

# Algoritmos de Substituição de Páginas

## A falta de página força uma escolha

**P:** Qual página deve ser removida para alocar espaço para a página a ser trazida do disco para a memória?

**R:** De preferência uma que não esteja sendo usada ou que não tenha sido modificada

## CrITÉrios básicos

- Uma página modificada (que está na memória) deve, inicialmente, ser reescrita no disco  $\Rightarrow$  **atualizar página virtual**

Obs: se não tiver sido modificada é apenas sobreposta

- É aconselhável não escolher uma página que está sendo muito usada  
 $\Rightarrow$  provavelmente precisará ser trazida novamente para a memória em pouco tempo

# O Algoritmo de Substituição de Página *Ótimo*

Substitui a página que será necessária depois que se passar o maior número de instruções

- Cada página teria um rótulo com essa informação
- Remove-se a página de maior rótulo

⇒ **ótimo mas não realizável (SO não tem como saber!)**

- O rótulo poderia ser estimado...

através do registro do uso da página em execuções anteriores (simulações) do processo, guardando o uso de cada referência

⇒ **Inútil pois só serve para um conjunto específico de dados**

# O Algoritmo de Substituição de Página *Não Usada Recentemente (NUR)*

Cada página tem os bits Referenciada (R) e Modificada (M)

- ✓ Localizados em cada entrada da tabela de páginas
- ✓ Bit R é colocado em 1 quando a página é referenciada (lida ou escrita) e o bit M quando é modificada (escrita)
- ✓ Bits atualizados a cada referência à memória  $\Rightarrow$  por hardware
- ✓ Em cada interrupção de relógio o bit R é limpo (colocado em '0')
- ✓ Processo iniciado  $\Rightarrow$  bits de todas as páginas são colocados em '0'

# O Algoritmo de Substituição de Página *Não Usada Recentemente (NUR)*

As páginas são classificadas em 4 classes:

- Classe 0: não referenciada, não modificada
- Classe 1: não referenciada, modificada
- Classe 2: referenciada, não modificada
- Classe 3: referenciada, modificada

**A cada tique de relógio somente o bit R é limpo  
(classe 3 vira classe 1)**

**NUR remove, aleatoriamente, uma página da classe de ordem  
mais baixa**

# Algoritmo de Substituição de Página *Primeira a Entrar, Primeira a Sair*

- Mantém uma lista encadeada de todas as páginas
  - página mais antiga na cabeça da lista
  - página que chegou por último na memória no final da lista
- Na ocorrência de falta de página
  - página na cabeça da lista é removida
  - nova página adicionada no final da lista

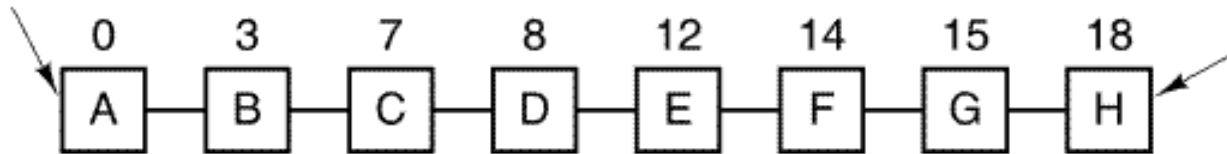
## Desvantagem:

*página há mais tempo na memória pode ser usada com muita frequência (diferentemente de processos)*

⇒ **algoritmo raramente utilizado**

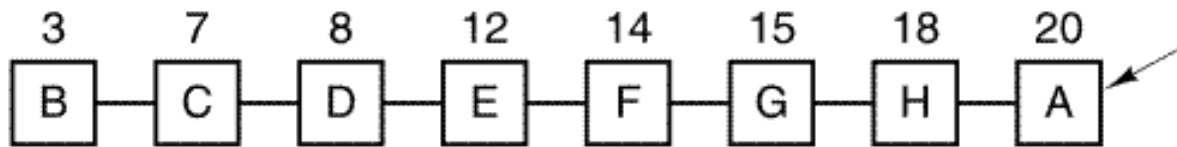
# Algoritmo de Substituição de Página *Segunda Chance (SC)*

Primeira página carregada



Página mais recentemente carregada

(a)



Página A é tratada como página mais recentemente carregada

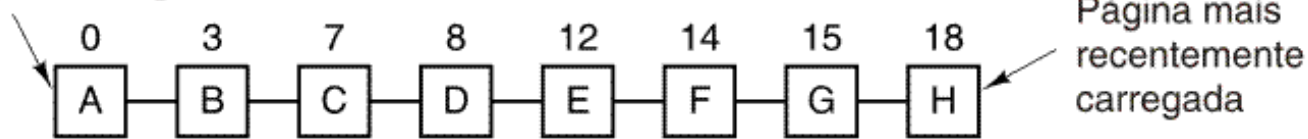
(b)

**Modificação no algoritmo FIFO que evita o problema de uma página muito usada ser substituída**

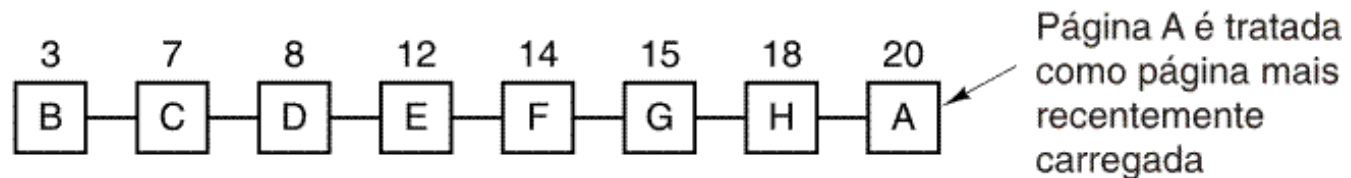
*(números = instantes de carregamento das páginas na memória)*

# Algoritmo de Substituição de Página *Segunda Chance (SC)*

Primeira página carregada



(a)



(b)

## Operação do algoritmo *segunda chance*:

- lista encadeada de páginas em ordem FIFO
- estado da lista em situação de falta de página no instante 20 se A com bit R em 1
- Se o bit *R* da página A em 1  $\Rightarrow$  é colocado em 0 e B é examinada, ...
- Transforma-se em FIFO se todas as páginas estiverem com bit  $R = 1$

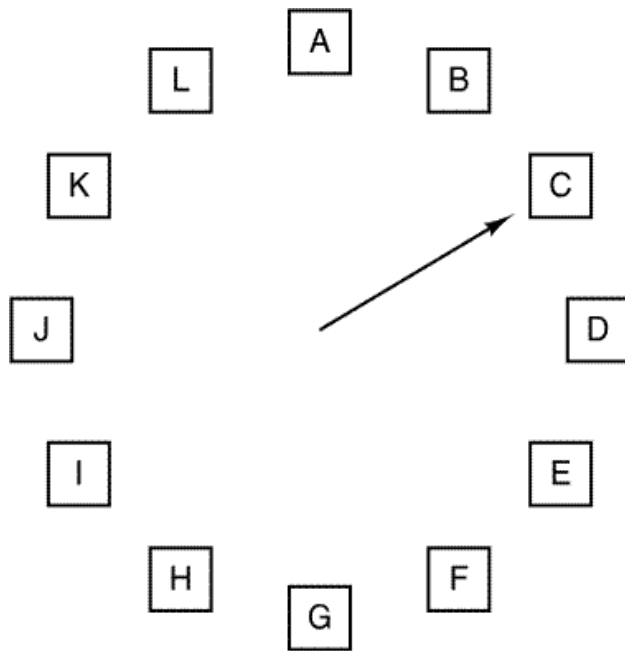
**Ineficiente, pois fica inserindo páginas no final da lista encadeada**

# Algoritmo de Substituição de Página *Relógio*

Evita a reinserção no final da lista realizada no algoritmo

**Segunda Chance**

*sempre aponta para a página mais antiga*



Quando ocorre uma falta de página,  
a página apontada é examinada.  
A atitude a ser tomada depende do bit R:  
R = 0: Retira a página,  
R = 1: Faz R = 0 e avança o ponteiro.



# Menos Recentemente Usada (LRU)

*Assume que páginas usadas recentemente logo serão usadas novamente  
(aproximação do algoritmo ótimo)*

- Retira da memória a página que há mais tempo não é usada
- Mantém uma lista encadeada de páginas

*⇒ página mais recentemente usada no início da lista, menos usada no final da lista*

Dificuldade: lista deve ser atualizada a cada referência à memória

*⇒ operação demorada, mesmo se executada em hw*

# Menos Recentemente Usada (LRU)

## Implementação em Hardware

“Recentemente” = referenciada entre uma falta de página e outra

- Existe um contador C de 64 bits e um campo de mesmo tamanho p/ cada entrada da tabela de páginas
- C é incrementado após cada instrução
- Após cada referência à memória, valor de C é copiado para a entrada da tabela de páginas correspondente à página referenciada
- Quando ocorre uma falta de página SO examina todos os contadores
- A página com o menor valor para C é retirada da memória

# LRU em Software

Problema:

Poucas máquinas têm hardware especial

Implementação em SW (alg. Não usadas frequentemente - *NFU*):

- Associa-se um contador, em SW, a cada uma das páginas
- A cada interrupção do relógio o SO percorre todas as págs na memória inspecionando o bit R;
- Bit R (0/1) é adicionado ao contador correspondente;
- Na FP, página com menor valor do contador é retirada.

# LRU em Software

## *algoritmo de envelhecimento*

Problema: *algoritmo NFU não se esquece do passado*

### **Exemplo:**

*código de um compilador de vários passos, as páginas que foram usadas intensamente no passo 1 terão um alto valor para o contador. Se o passo 1 tiver um tempo de execução muito maior que os outros passos, seus contadores serão sempre maiores que os das páginas dos passos seguintes.*

⇒ **páginas mais recentes terão contadores de menor valor**

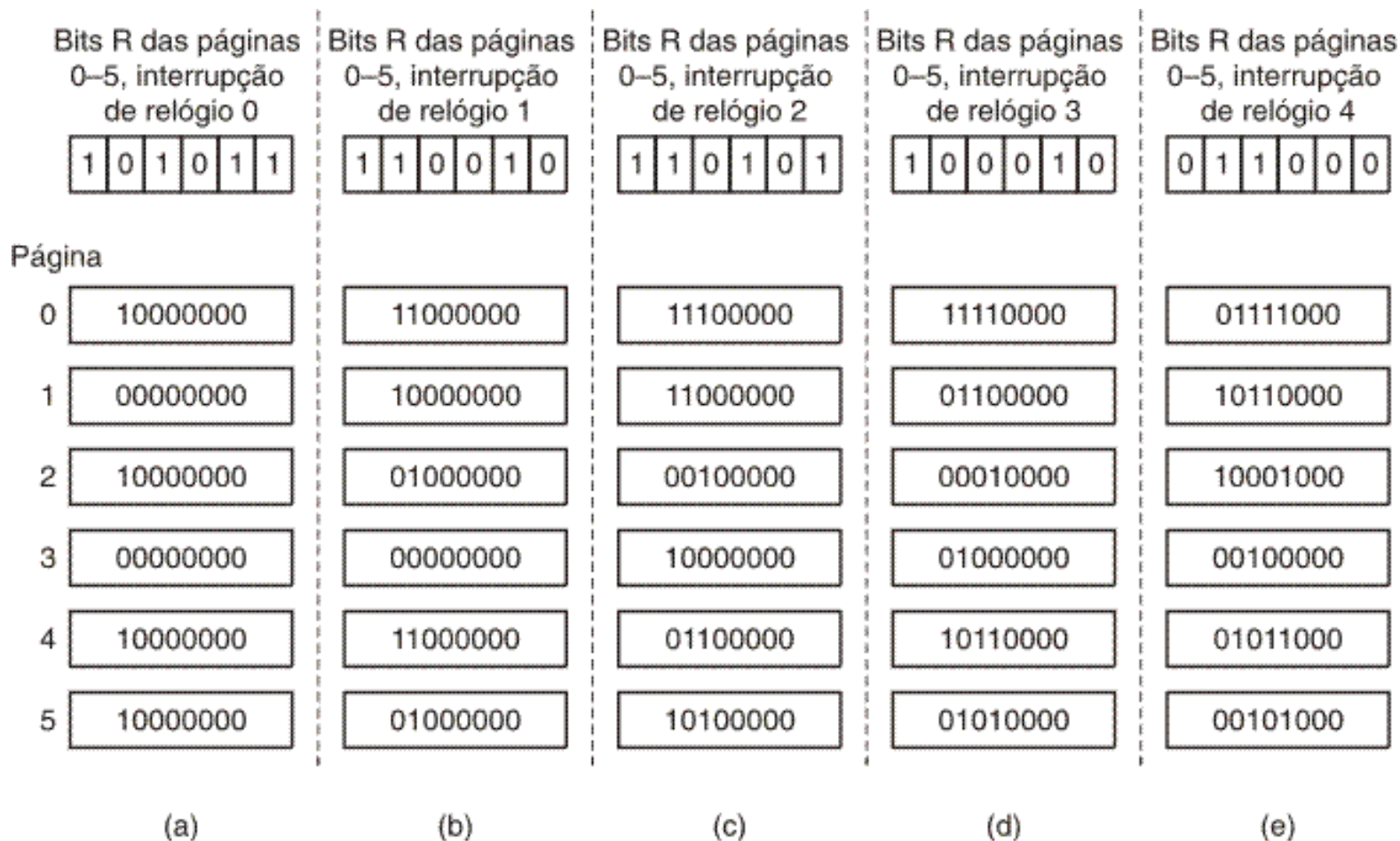
**Solução:** **algoritmo de envelhecimento**

# LRU em Software

## Algoritmo de envelhecimento

- A cada click do relógio, desloca-se à direita a sequência de bits R de cada página (normalmente, últimos 8 clicks, inicialmente zerados), inserindo-se o bit R (1 ou 0) na posição mais significativa
- Quando ocorre uma falta de página, a página que tiver o menor valor binário é retirada da memória

# Algoritmo de envelhecimento (*aging*)



- O algoritmo do envelhecimento (*aging*) simula o LRU em software
- 6 páginas com 5 tiques de relógio, (a) – (e)

# Algoritmo de envelhecimento (*aging*)

## Este algoritmo tem duas limitações:

- ✓ Ao registrar apenas um bit por intervalo de tempo, perde-se a capacidade de distinguir a ordem da referência dentro do mesmo intervalo (não se sabe qual delas foi referenciada por último e muitas páginas são referenciadas em cada tique de relógio)
- ✓ Número fixo de bits (apenas 8 bits no exemplo): limita o horizonte passado
  - ⇒ uma página pode ter sido referenciada há 9 intervalos atrás e uma outra há mil intervalos atrás.
    - Porém, páginas não utilizadas há mais de 160 ms (mais de 8 intervalos, supondo interrupções a cada 20 ms) não são importantes

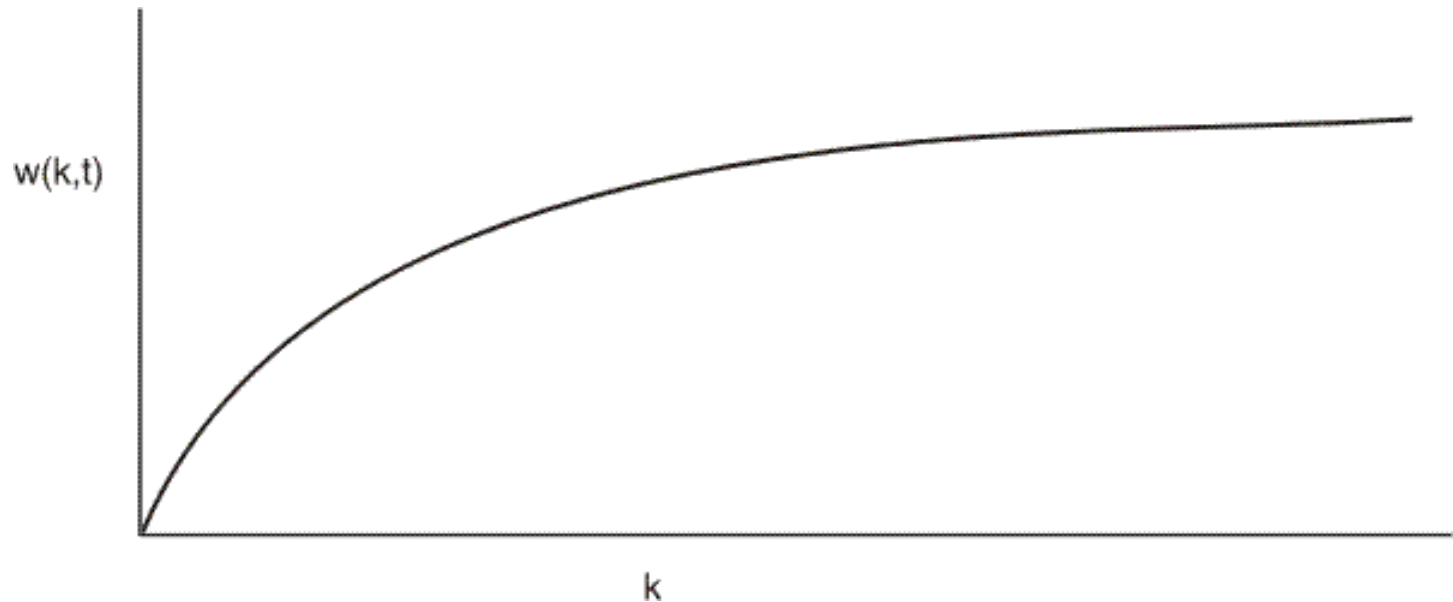
# O Algoritmo de Substituição de Página do *Conjunto de Trabalho*

## Motivação

- Normalmente a solicitação de páginas é *por demanda*  
⇒ páginas são carregadas à medida que são solicitadas
- Programa que gera muitas faltas de página é chamado de *ultrapaginado*
- Quando um processo volta para o disco (CPU é trocada para outro processo), todas as páginas podem ser retiradas da memória  
⇒ quantas faltas de página quando voltar? (uma a uma)  
⇒ **execução lenta**



# O Algoritmo de Substituição de Página do *Conjunto de Trabalho*



**Def.:** O conjunto de trabalho em um instante de tempo  $t$  é o conjunto de páginas usadas nas últimas  $k$  referências mais recentes à memória, representado por  $w(k,t)$   
*(tamanho do conjunto de trabalho no instante  $t$ )*

- $w(k,t)$  é finito e monotonicamente não decrescente (tamanho máximo de  $k$  é o espaço de endereçamento do programa)

# O Algoritmo de Substituição de Página do *Conjunto de Trabalho*

**Ideia:** gerenciar o conjunto de trabalho de cada processo para

⇒ **reduzir as faltas de páginas**

Como?

***pré-paginação:*** garantir que o conjunto já esteja na memória antes do processo ser reiniciado

- Crescimento lento da curva ( **$w(k,t)$  varia lentamente**)

⇒ possível prever quais páginas serão necessárias quando o programa for reiniciado

**Algoritmo:** *qdo ocorrer uma FP encontre uma página não pertencente ao conjunto de trabalho e a remova*

# O Algoritmo de Substituição de Página do *Conjunto de Trabalho*

## Determinação do conjunto de trabalho:

- Primeira ideia:

Usar um registrador de deslocamento de tamanho  $k$  que armazene as  $k$  últimas páginas acessadas nas últimas  $k$  referências à memória

Mas.... Manter o registrador e processá-lo a cada FP tem um *custo proibitivo*

- Solução melhor:

Conjunto de páginas referenciadas pelo processo durante os últimos  $\Gamma$  seg de uso de CPU

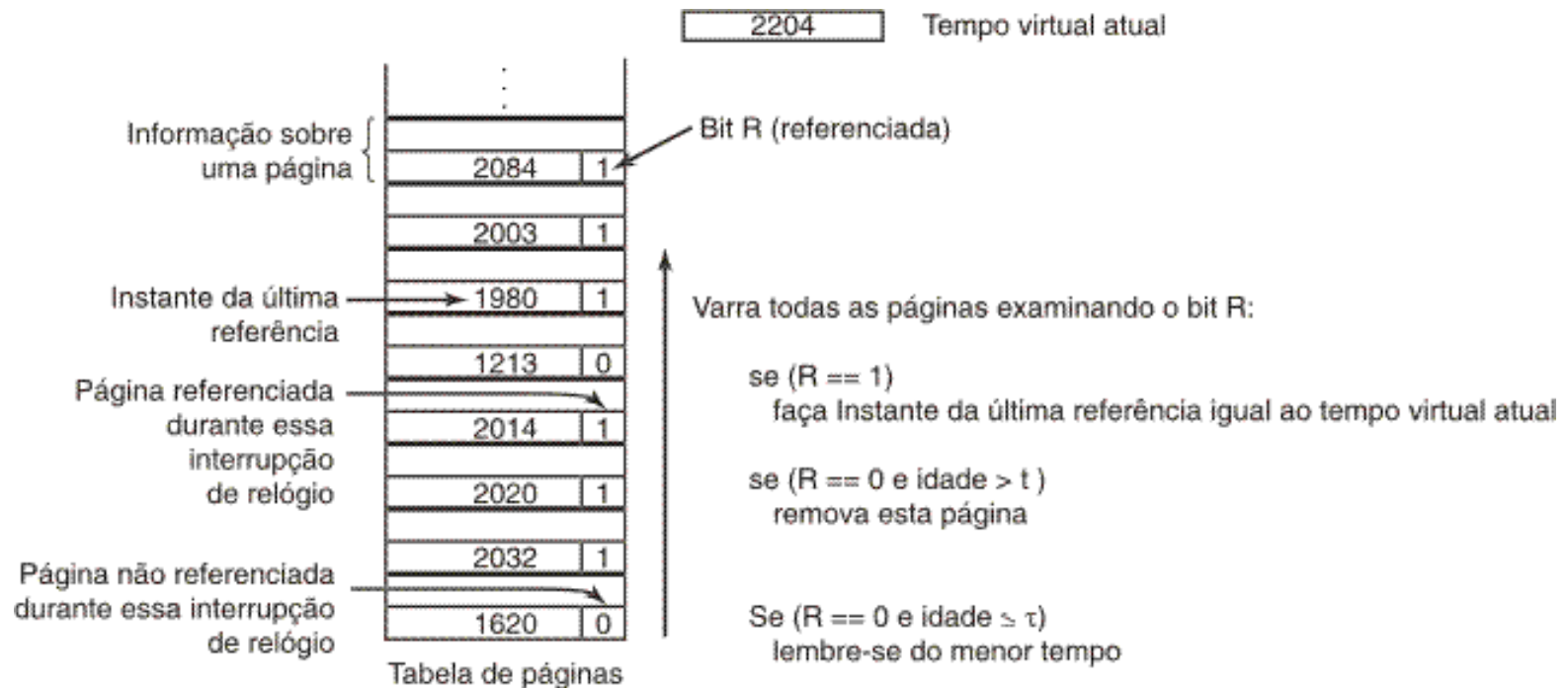
# O Algoritmo de Substituição de Página do *Conjunto de Trabalho*

Ideia do Algoritmo: encontrar uma página que não esteja no conjunto de trabalho e removê-la da memória

A cada FP, varre-se a tabela de páginas;

- Se  $R = 1$  (página referenciada desde a última interrupção do relógio), copia-se o tempo virtual atual no campo “instante de último uso”  $\Rightarrow$  **pertence a  $W(k,t)$   $\Rightarrow$  não candidata à remoção**
- Se  $R = 0$  compara-se o intervalo (tempo virtual – instante de último uso) com  $\Gamma$ ; **se intervalo  $> \Gamma \Rightarrow$  não está no conjunto de trabalho  $\Rightarrow$  página é removida** e atualiza-se as outras entradas, **senão página poupada (pertence a  $W(k,t)$ , mas é marcada como candidata).**
- Se todas pertencem a  $W(k,t)$  (todas com intervalo  $< \Gamma$ ), **página de maior idade com  $R=0$  é removida.**
- Se todas têm  $R=1$ , escolhe-se uma página “limpa” (não modificada), aleatoriamente, para remoção (se houver uma)

# O Algoritmo de Substituição de Página do *Conjunto de Trabalho*



# O Algoritmo de Substituição de Página *WSClock*

Problema com o algoritmo de conjunto de trabalho (WS):

**Custo: pesquisa em toda a tabela de páginas a cada FP**

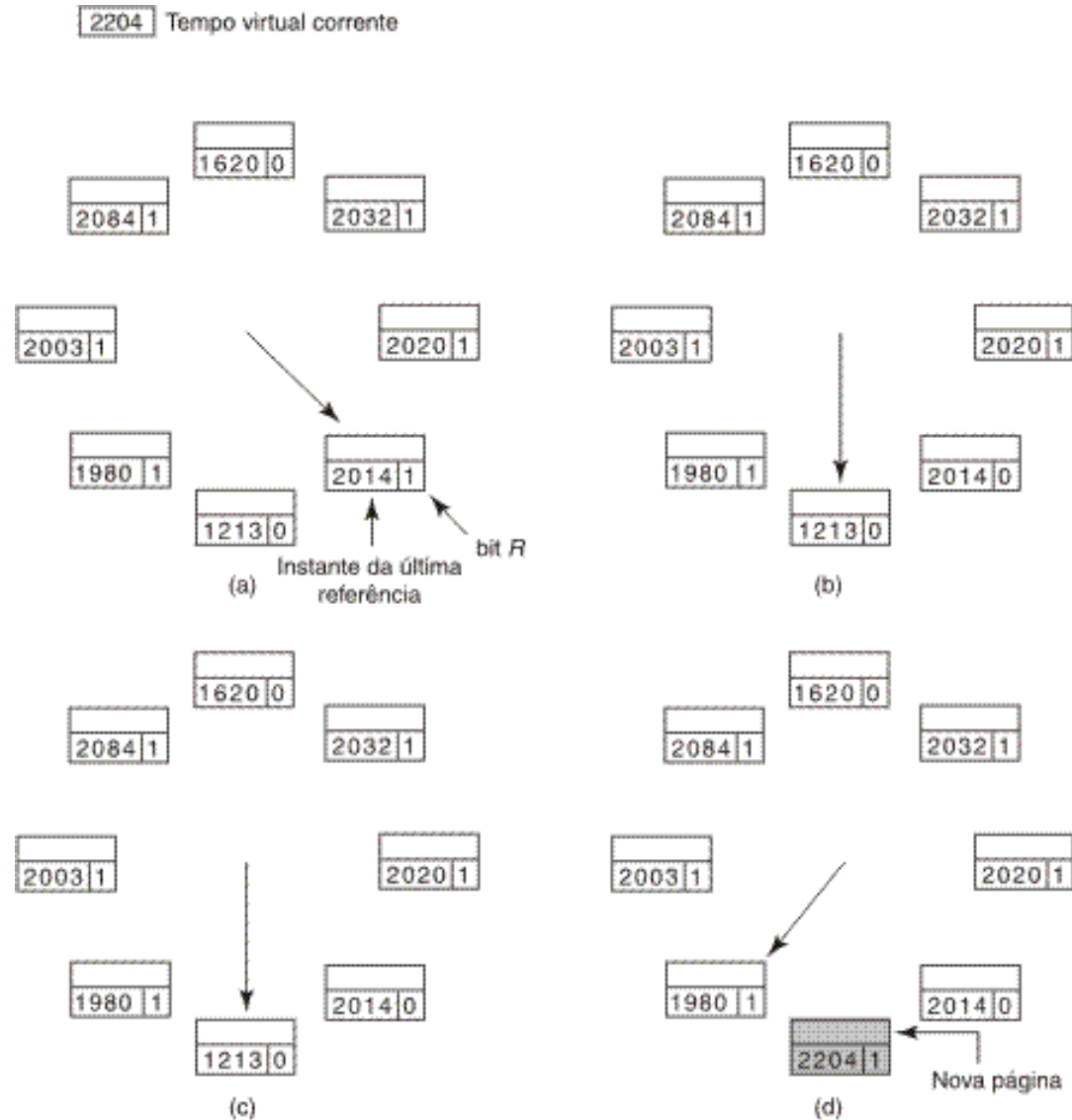
WSClock: baseado no algoritmo do relógio e melhora o desempenho do WS básico:

## *Características:*

1. utiliza uma lista circular de quadros de páginas;
2. cada entrada da lista contém o instante do último uso, o bit R e o bit M

# O Algoritmo de Substituição de Página *WSClock*

Parte da operação  
do algoritmo  
*WSClock*



# O Algoritmo de Substituição de Página *WSClock*

- lista circular: cada entrada tem o instante de último uso, bit R e bit M
- verifica-se a página apontada:
  - se  $R = 1 \Rightarrow$  **página referenciada**  $\Rightarrow$  **não candidata à remoção**.  
Bit R é colocado em 0, ponteiro avança;
  - se  $R = 0$ , verifica-se a idade:
    - Se idade  $> \Gamma$  e página limpa  
 $\Rightarrow$  **nova página colocada no lugar desta**;
    - Se idade  $> \Gamma$  e página suja  $\Rightarrow$  escrita em disco é escalonada (evita chaveamento de processo).  
 $\Rightarrow$  **Avança ponteiro**



# O Algoritmo de Substituição de Página *WSClock*

Caso o ponteiro dê uma volta completa:

1. Pelo menos uma escrita escalonada;
2. Nenhuma escrita escalonada

**Caso 1:** ponteiro continua a se mover procurando uma página limpa: em algum momento uma das escritas será realizada e a página ficará limpa. Primeira pág. limpa encontrada é removida

**Caso 2:** todas as páginas pertencem ao conjunto de trabalho; reivindicar uma página limpa para ser usada; se não existir uma página limpa, página que se está sendo apontada será escolhida e reescrita em disco

# Resumo das Características dos Algoritmos de Substituição de Página

Algoritmo	Comentário
Ótimo	Não implementável, mas útil como um padrão de desempenho
NUR (não usada recentemente)	Muito rudimentar
FIFO (primeira a entrar, primeira a sair)	Pode descartar páginas importantes
Segunda chance	Algoritmo FIFO bastante melhorado
Relógio	Realista
MRU (menos recentemente usada)	Excelente algoritmo, porém difícil de ser implementado de maneira exata
NFU (não freqüentemente usada)	Aproximação bastante rudimentar do MRU
Envelhecimento ( <i>aging</i> )	Algoritmo bastante eficiente que se aproxima bem do MRU
Conjunto de trabalho	Implementação um tanto cara
WSClock	Algoritmo bom e eficiente

## Questões de projeto:

Política de alocação de memória: *local x global*

	Idade
A0	10
A1	7
A2	5
A3	4
A4	6
A5	3
B0	9
B1	4
B2	6
B3	2
B4	5
B5	6
B6	12
C1	3
C2	5
C3	6

(a)

A0
A1
A2
A3
A4
A6
B0
B1
B2
B3
B4
B5
B6
C1
C2
C3

(b)

A0
A1
A2
A3
A4
A5
B0
B1
B2
A6
B4
B5
B6
C1
C2
C3

(c)

(a) Configuração original (b) Substituição local (c) Substituição global

Algoritmo usado: página menos recentemente usada

# Questões de projeto:

## Política de alocação – *local x global*

- ❑ Algoritmos de Substituição local alocam uma parcela fixa de memória para cada processo
  - ⇒ desperdiça molduras quando o conjunto de trabalho diminui e causa ultrapaginação quando o conjunto de trabalho aumenta
- ❑ Algoritmos de Substituição global alocam molduras de páginas aos processos em execução
  - ⇒ número de molduras varia no tempo ⇒ melhor desempenho quando o conjunto de trabalho varia no tempo

# Questões de projeto:

## Política de alocação – *local x global*

Algoritmo PFF (frequência de faltas de página):

*Tenta manter a frequência de FPs em limites aceitáveis, informando quando aumentar e quando diminuir a alocação de páginas de um processo*

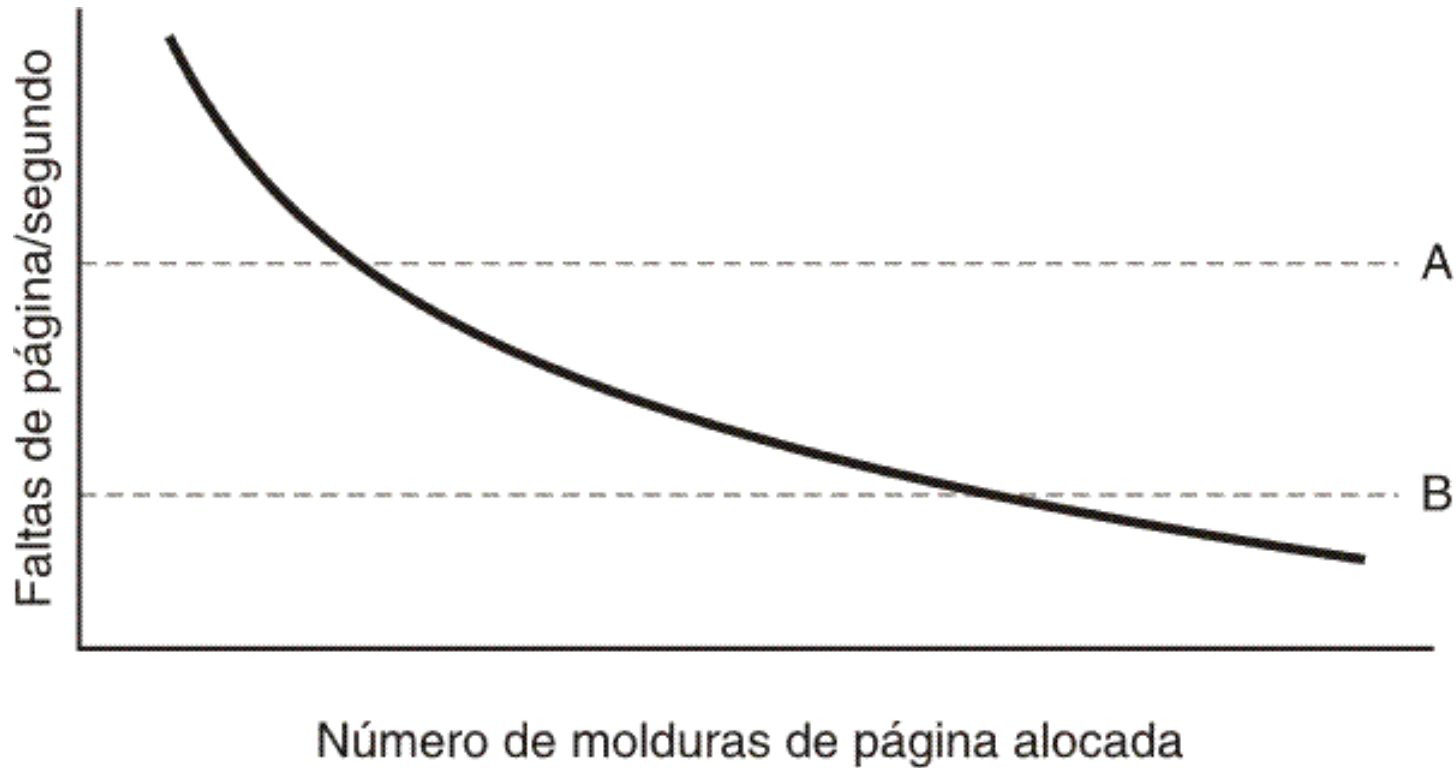
⇒ vai retirar molduras de B e ceder a A para manter a frequência de paginação para cada processo em níveis aceitáveis

**Obs:**

Maioria dos algoritmos de substituição trabalha com política de substituição global, com exceção dos que utilizam conjunto de trabalho

# Questões de projeto:

## Política de alocação – *local x global*



Ação: tirar molduras do processo B e alocar ao processo A

# Controle de Carga

Mesmo com um bom projeto, o sistema ainda pode sofrer de excessivas faltas de páginas (*ultrapaginação*)

## Exemplo:

*Quando a soma do conjunto de trabalho dos processos exceder a quantidade de memória*

## *Sintoma:*

Quando o algoritmo PFF indica que:

- **alguns processos precisam de mais memória**
- **mas nenhum processo precisa de menos memória**

# Controle de Carga

## Solução:

**Reduzir o número de processos que competem pela memória.  
Como?**

- levar alguns deles para disco e liberar as suas molduras, distribuindo-as entre outros processos sofrendo de ultrapaginação
- se a ultrapaginação continuar, levar outro para o disco, etc
- ao reconsiderar o grau de multiprogramação, deve-se, também, considerar outros fatores, não só o *tamanho dos processos* e a *taxa de paginação*, mas também:
  - *tipos dos processos (orientado a E/S ou a CPU)*

**Objetivo: CPU não ficar ociosa**



# Tamanho da Página

Pode ser determinado pelo SO mesmo quando especificado pelo HW.

**Ex: alocar pares de páginas**

*tamanho de página pequeno x tamanho de página grande*

- Vantagens

- Tamanho pequeno  $\Rightarrow$  menos fragmentação interna
- Tamanho pequeno  $\Rightarrow$  menos programas não usados na memória (menos desperdício de espaço na memória)

- Desvantagens

- Tamanho pequeno  $\Rightarrow$  programas precisam de mais páginas  
 $\Rightarrow$  **tabelas de página maiores**  
 $\Rightarrow$  **transferência de págs entre o disco e a memória leva mais tempo (ex: 64 pgs x 10 ms  $\gg$  4 pgs x 12 ms)**

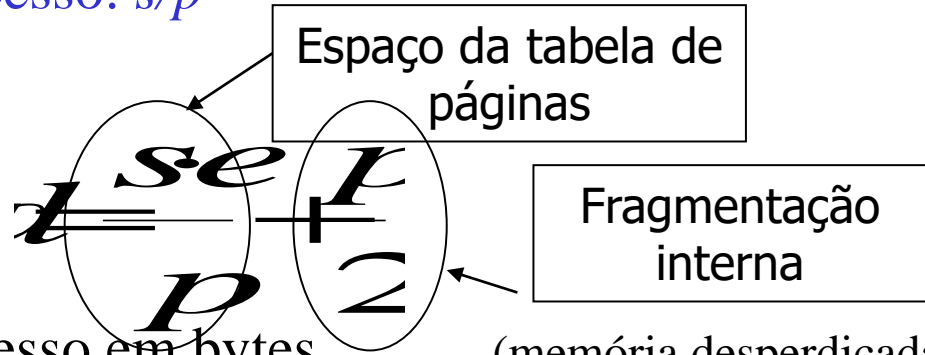
# Tamanho da Página

## Desvantagens (cont):

- Tempo para carregamento do processo aumenta quando o tamanho da página diminui
- Espaço ocupado pela tabela de páginas também aumenta número de páginas por processo:  $s/p$

Onde

*custo adicional total*

$$\frac{se}{p} + \frac{e}{2}$$


Espaço da tabela de páginas

Fragmentação interna

s = tamanho médio do processo em bytes

p = tamanho da página em bytes

e = tamanho da entrada da tabela de página

(memória desperdiçada na última página do processo, em média)

tamanho ótimo da página:

$$p = \sqrt{2se}$$