Gerenciamento de Buffer Pool

Anderson Chaves Carniel
Profa. Dra. Cristina Dutra de Aguiar Ciferri

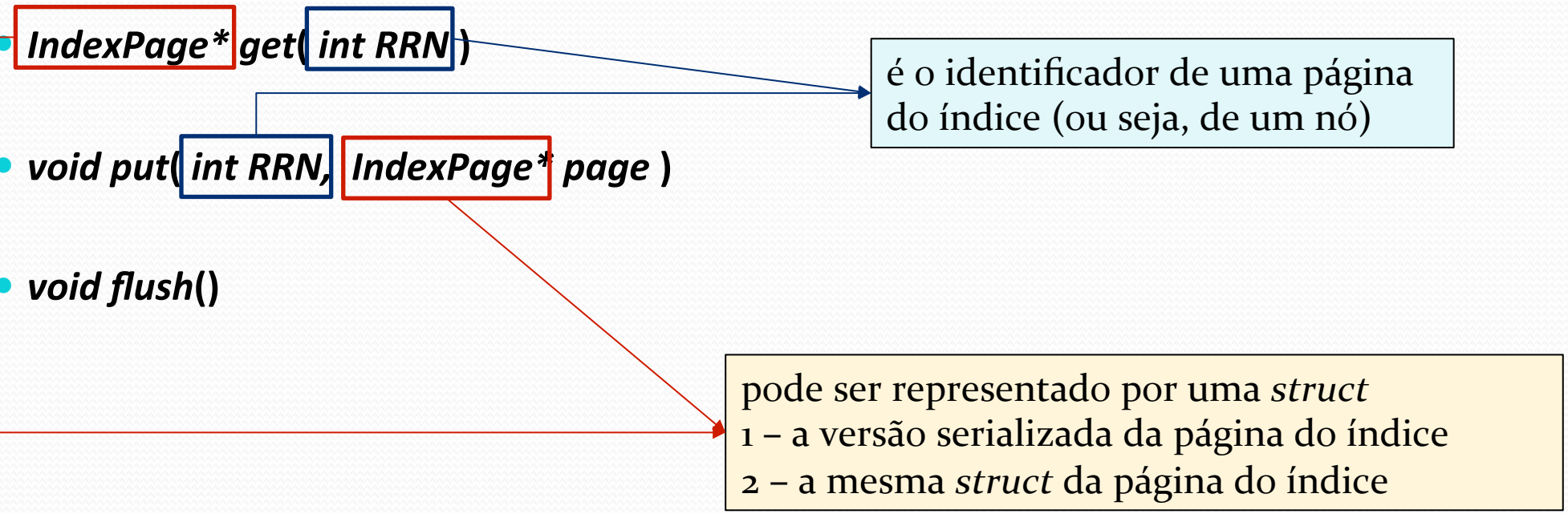
Gerenciamento de Buffer

- Mantém-se uma quantidade reservada de memória principal, chamada de **buffer**, para:
  - Diminuir o número de acessos a disco
  - Evitar alguns tipos de padrões de acesso a disco
- O uso de um buffer é altamente indicado para **melhorar o desempenho** de um índice, como a árvore B
- Em geral, busca-se manter no buffer as páginas do índice (ou seja, os nós) que são as **mais acessadas**
- Problema:
  - **Como o buffer tem tamanho limitado, qual é o critério que deve ser usado para determinar quais páginas devem ser mantidas no buffer?**

Gerenciamento de Buffer
Operações básicas

- Pode-se implementar um buffer como uma **camada de gerenciamento de dados**, a qual é responsável por recuperar e modificar as páginas do índice

- ***IndexPage**** ***get*****(** ***int RRN*** **)**
- ***void put*****(** ***int RRN*****,** ***IndexPage**** ***page*** **)**
- ***void flush*****()**



Gerenciamento de Buffer
Operações básicas

- ***IndexPage* get(int RRN)***
  - Esta função é chamada para recuperar o conteúdo de uma página do índice
  - Existem dois casos básicos:
  - 1 – se a página do índice está mantida no buffer:
    - 1º: copie o conteúdo da página requerida para uma variável auxiliar do tipo *IndexPage*, chamada *P*
    - 2º: realize **possíveis restruturações na organização do buffer** (relacionada a **política de substituição de páginas**)
    - 3º: retorne *P*
  - 2 – se a página do índice não está no buffer:
    - 1º: recupere o conteúdo da página do disco e armazena em uma variável auxiliar do tipo *IndexPage*, chamada *P*
    - 2º: insira P no buffer, chamando a função ***put***
    - 3º: retorne *P*

Gerenciamento de Buffer
Operações básicas

- ***void put(int RRN, IndexPage* page)***
  - Esta função é chamada para armazenar páginas no buffer
  - Existem dois casos básicos, que podem ser processados por uma versão *interna* do *put*:
    - ***void put(int RRN, IndexPage* page, bool modified)***

  - 1 – se a página do índice não está no buffer (logo, ela foi invocada pelo segundo caso da função *get*):
    - 1º: copie o conteúdo da página para uma variável auxiliar do tipo *IndexPage*, chamada *P*
    - 2º: se o buffer conter espaço disponível: insira *P* no espaço disponível
    - 2º: caso contrário: remova uma página *N* do buffer e insira *P*, respeitando a **política de substituição de páginas**
  - 2 – se a página do índice está no buffer (logo, ela foi modificada por alguma operação do índice):
    - 1º: recupere a página armazenada no buffer e atualize seu conteúdo
    - 2º: marque que *P* é uma página modificada
    - 3º: reorganize a estrutura interna do buffer de acordo com sua **política de substituição de páginas**

Gerenciamento de Buffer
Operações básicas

- ***void flush*()**
  - Esta função é chamada para escrever no disco todas as páginas com **modificações** presentes no *buffer*

- Outra função *interna* relacionada é:

- ***void flush*(IndexPage *page)**
  - Esta função é chamada para escrever no disco uma página que foi **modificada**
  - Sempre chamada durante a realização das trocas de páginas a serem armazenadas no *buffer*

Gerenciamento de Buffer
Políticas de substituição de páginas

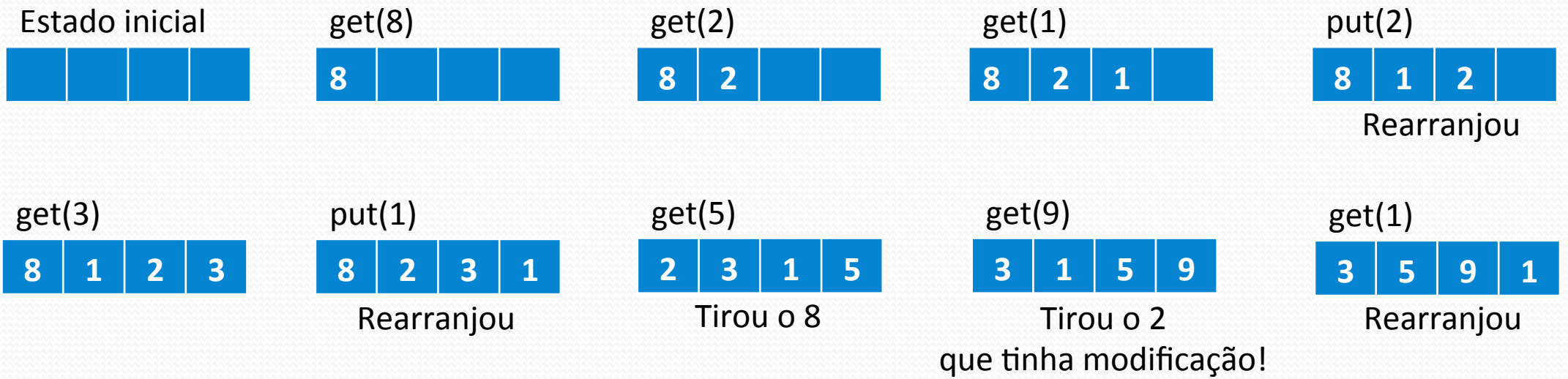
- Existem várias políticas de substituições de páginas, as quais são invocadas quando páginas precisam ser armazenadas no buffer e não há espaço disponível
- A estrutura de dados usada no buffer deve considerar a política de substituição para melhor eficiência

- ***LRU (Least Recently Used)***
- ***MRU (Most Recently Used)***
- ***LFU (Least Frequently Used)***
- ***FIFO (First-in First-out)***
- ***LIFO (Last-in First-out)***
- ***S2Q (Simplified 2 Queues)***
- ***NRU (Not-recently Used)***
- ***SCA (Second-chance Algorithm)***

Na parte 2 do trabalho, cada grupo usará uma política de buffer

LRU - *Least Recently Used*

- **Processo de escolha de página a ser substituição:** **a menos recentemente usada** (sempre localizada na cabeça da lista – a cauda da lista indica a página usada mais recentemente) – ideal para um padrão aleatório de acesso (tende a deixar páginas recentes no *buffer*)
- **Estrutura de dados básica:** lista **Capacidade:** 4 páginas (ou *frames*)
- **Requisições de páginas (*put/get*):** 8 – 2 – 1 – 2 – 3 – 1 – 5 – 9 – 1
- ***Page fault***: 6 ***Page hit***: 3



MRU - *Most Recently Used*

- **Processo de escolha de página a ser substituição:** **a mais recentemente usada** (sempre localizada na cauda da lista – a cabeça da lista indica a página menos usada) – ideal para um padrão sequencial de acesso (tende a deixar páginas antigas no *buffer*)
- **Estrutura de dados básica:** lista **Capacidade:** 4 páginas (ou *frames*)
- **Requisições de páginas (*put/get*):** 8 – 2 – 1 – 2 – 3 – 1 – 5 – 9 – 1
- ***Page fault***: 7 ***Page hit***: 2

Estado inicial

get(8)

get(2)

get(1)

put(2)

Rearranjou

get(3)

put(1)

Rearranjou

get(5)

Tirou o 1
que tinha modificação!

get(9)

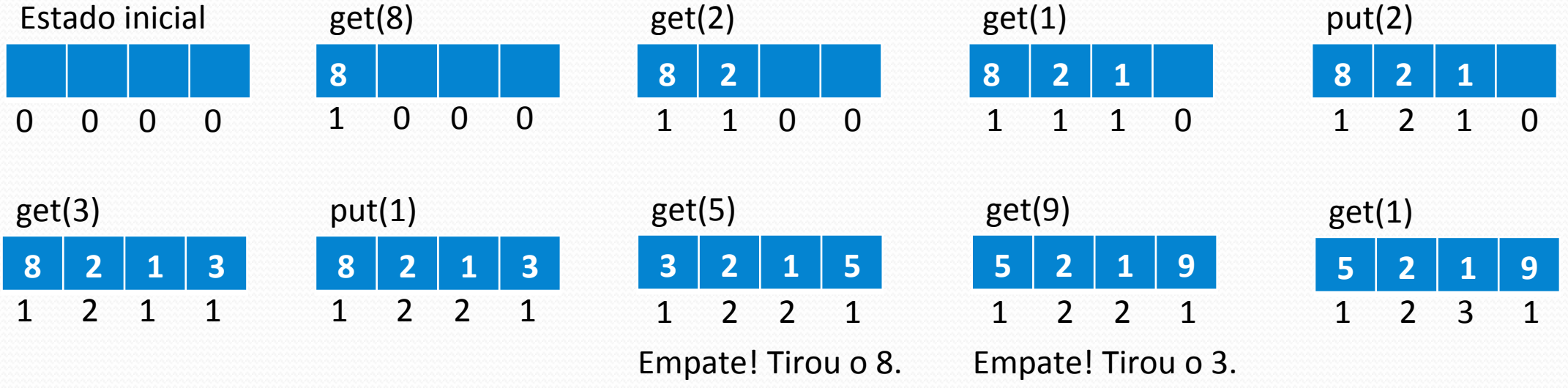
Tirou o 5

get(1)

Tirou o 9

LFU - *Least Frequently Used*

- **Processo de escolha de página a ser substituição:** **a página menos usada.** Cada página armazenada no *buffer* tem um contador de quantos acessos/modificações a página tem e então descarta-se a página com a menor contagem. Parecido com o LRU (em empates, tira-se o mais antigo – localizado mais próximo da cabeça da lista)
- **Estrutura de dados básica:** lista + contadores **Capacidade:** 4 páginas (ou *frames*)
- **Requisições de páginas (*put/get*):** 8 – 2 – 1 – 2 – 3 – 1 – 5 – 9 – 1
- ***Page fault***: 6 ***Page hit***: 3



FIFO - *First-in First-out*

- **Processo de escolha de página a ser substituição:** **a primeira página que foi acessada** (comportamento de uma fila) – bastante simples e não tão eficiente pois não considera outras informações como tempo e quantidade de acessos
- **Estrutura de dados básica:** fila **Capacidade:** 4 páginas (ou *frames*)
- **Requisições de páginas (*put/get*):** 8 – 2 – 1 – 2 – 3 – 1 – 5 – 9 – 1
- ***Page fault***: 6 ***Page hit***: 3

Estado inicial

get(8)

get(2)

get(1)

put(2)

Não troca a ordem.

get(3)

put(1)

Não troca a ordem.

get(5)

Tirou o 8,
a primeira que entrou

get(9)

Tirou o 2, com modif.,
a segunda que entrou

get(1)

LIFO - *Last-in First-out*

- **Processo de escolha de página a ser substituição:** **a última página que foi acessada e modificada** (comportamento de uma pilha) – bastante simples e não tão eficiente pois não considera outras informações como tempo e quantidade de acessos
- **Estrutura de dados básica:** pilha **Capacidade:** 4 páginas (ou *frames*)
- **Requisições de páginas (*put/get*):** 8 – 2 – 1 – 2 – 3 – 1 – 5 – 9 – 1
- ***Page fault***: 7 ***Page hit***: 2

Estado inicial

get(8)

get(2)

get(1)

put(2)

O topo agora é o 2.

get(3)

put(1)

O topo agora é o 1.

get(5)

Tirou o 1,
a última modificada

get(9)

Tirou o 5,
a última acessada

get(1)

Tirou o 9,
a última acessada

S2Q - *Simplified 2 Queues*

Processo de escolha de página a ser substituição: **usa-se 2 estruturas – uma LRU e outra FIFO.** A LRU armazena páginas frequentemente acessadas, enquanto a FIFO armazena páginas recentemente acessadas. estrutura FIFO armazena apenas os identificadores de páginas!! O conteúdo das páginas e seus identificado são armazenados na estrutura LRU, que armazena as páginas que tiveram ao menos 2 acessos. Indicado par acesso com padrão sequencial e repetitivo.

Estrutura de dados básica: lista e fila **Capacidades: LRU =** 4 páginas (ou *frames*), **FIFO =** 2 identificado

Requisições de páginas (*put/get*): 8 – 2 – 1 – 2 – 3 – 1 – 5 – 9 – 1

Page fault: 8 ***Page hit***: 1

Estado inicial

get(8)

FIFO: 8

get(2)

FIFO: 8 - 2

get(1)

FIFO: 2 - 1

put(2)

FIFO: 1

Armazena pois estava no FIFO

get(3)

FIFO: 1 - 3

put(1)

FIFO: 3

Armazena pois estava no FIFO.

get(5)

FIFO: 3 - 5

get(9)

FIFO: 5 - 9

get(1)

FIFO: 5 - 9

NRU - *Not-recently Used*

- **Processo de escolha de página a ser substituição:** **a página que não é nem acessada ou modificada.** Marca-se o tipo de operação de cada página do buffer com um valor 0 a 3: 0 = não foi modificada ou acessada, 1 = somente modificada, 2 = somente acessada, 3 = acessada e modificada. Escolhe-se uma página aleatória que tenha 0 como marcação. Caso não existam páginas, vai subindo de níveis.
- **Estrutura de dados básica:** lista **Capacidade:** 4 páginas (ou *frames*)
- **Requisições de páginas (*put/get*):** 8 – 2 – 1 – 2 – 3 – 1 – 5 – 9 – 1
- ***Page fault***: 5 ***Page hit***: 4

Estado inicial

5	7	4	2
0	0	0	0

áginas aleatórias são escolhidas.

get(8)

5	7	8	2
0	0	2	0

Tirou o 4.
Uma página aleatória sai.

get(2)

5	7	8	2
0	0	2	2

get(1)

1	7	8	2
2	0	2	2

Tirou o 5.
Uma página aleatória sai.

put(2)

1	7	8	2
2	0	2	3

get(3)

1	3	8	2
2	2	2	3

Tirou o 7.

put(1)

1	3	8	2
3	2	2	3

get(5)

1	3	5	2
3	2	2	3

Empate! Tirou o 8.

get(9)

1	9	5	2
3	2	2	3

Empate! Tirou o 3.

get(1)

1	9	5	2
3	2	2	3

SCA - *Second-chance Algorithm*

- **Processo de escolha de página a ser substituição:** **como a FIFO, mas com uma segunda chance.** Marca-se um bit (0 ou 1) em cada página armazenada no *buffer*. Quando uma página é acessada ou modificada, marca-se seu bit como 1. Se uma página a ser substituída conforme a política FIFO tiver seu bit igual a 1, muda-se o seu bit para 0 e escolhe o próximo elemento na ordem FIFO (esse processo pode se repetir até que se encontre uma página com bit 0).
- **Estrutura de dados básica:** fila **Capacidade:** 4 páginas (ou *frames*)
- **Requisições de páginas (*put/get*):** 8 – 2 – 1 – 2 – 3 – 1 – 5 – 9 – 1
- ***Page fault***: 6 ***Page hit***: 3