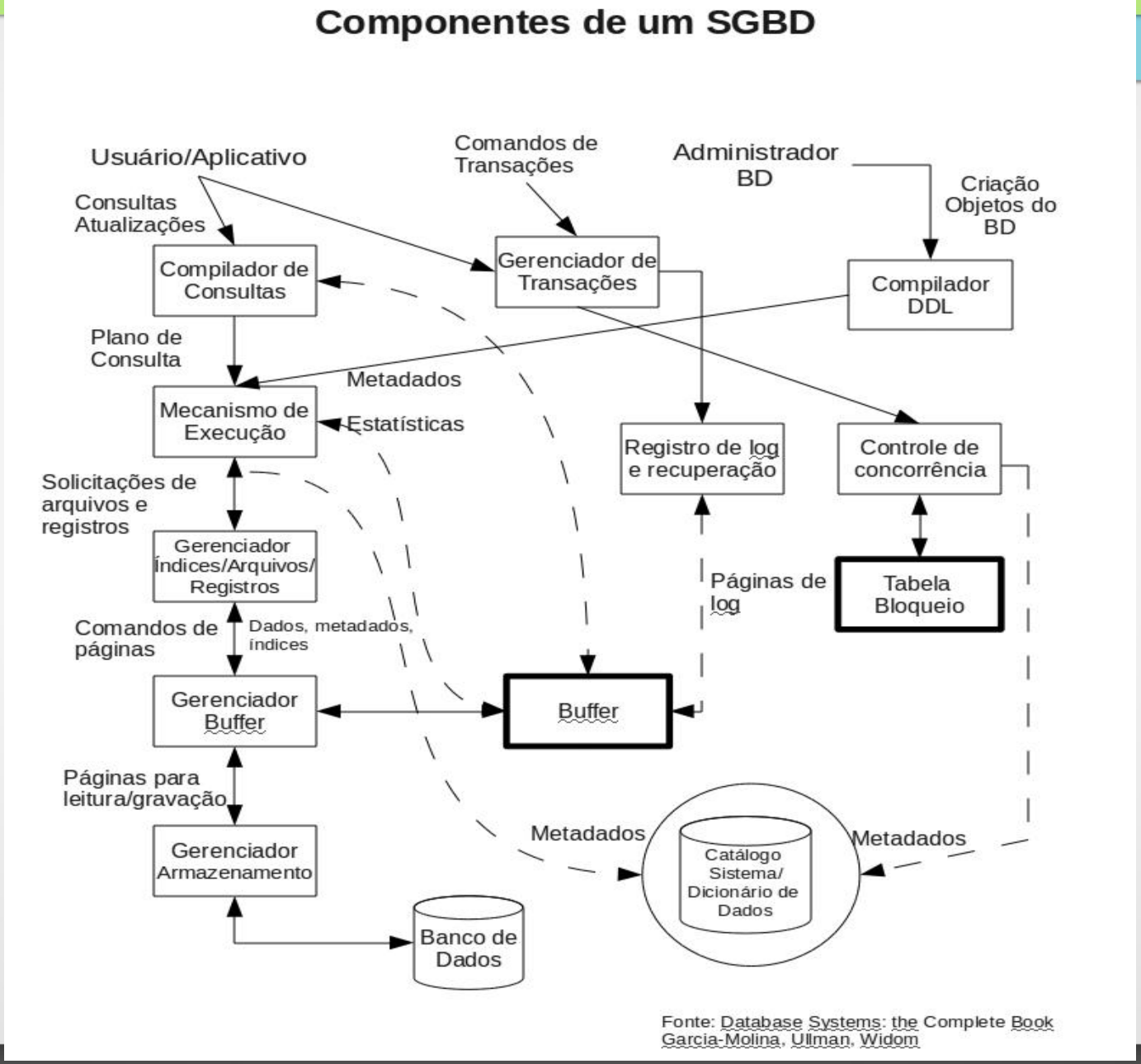
Arquitetura de SGBD Relacionais

Armazenamento: Discos e Arquivos

Cap. 8 (Vaquinhas)
Cap. 11 (Barquinho[1])
Cap. 17 (Elmasri)
Cap. 13 (The Complete Book)

Denio Duarte

Componentes SGBD

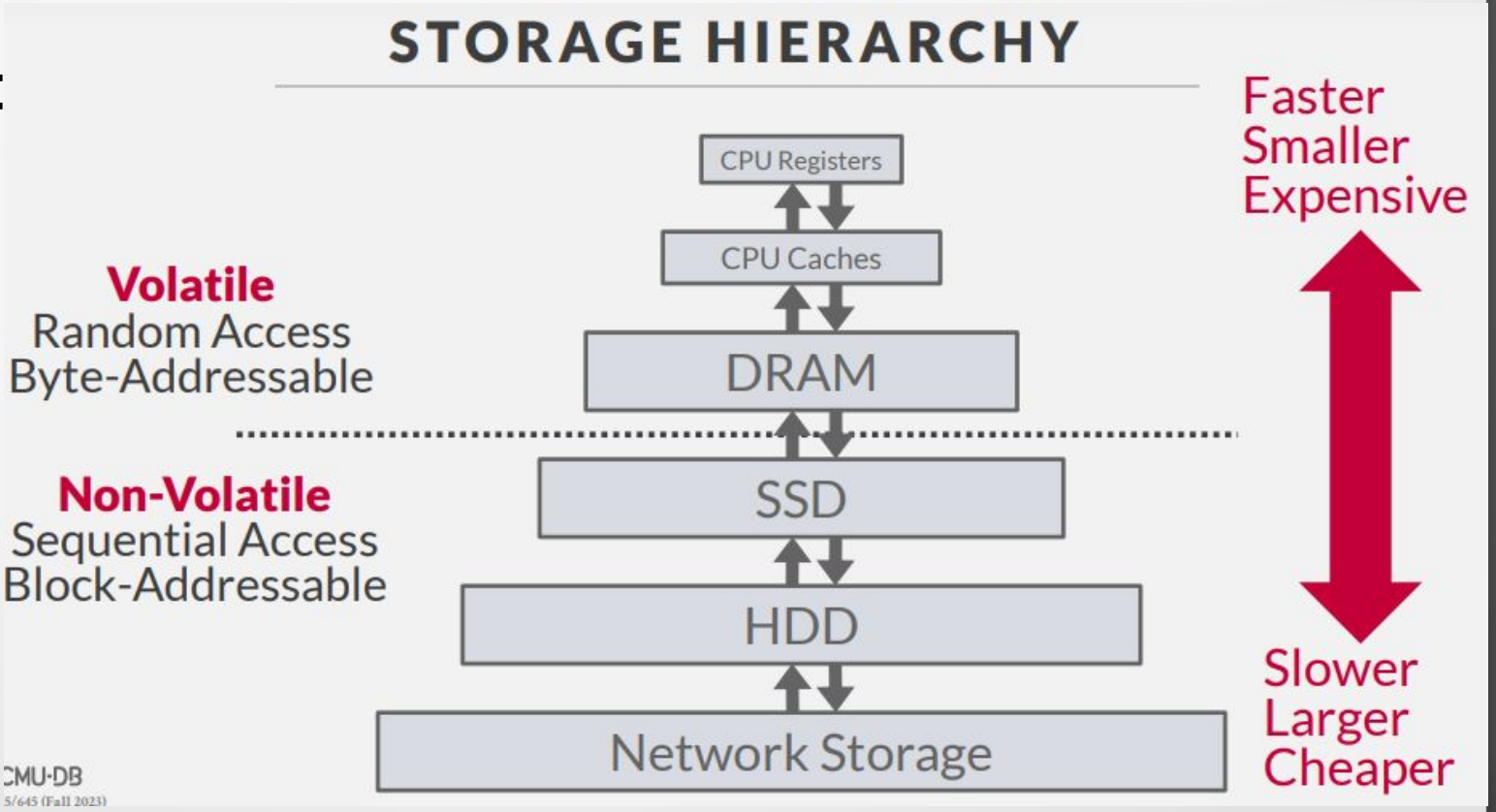


Introdução

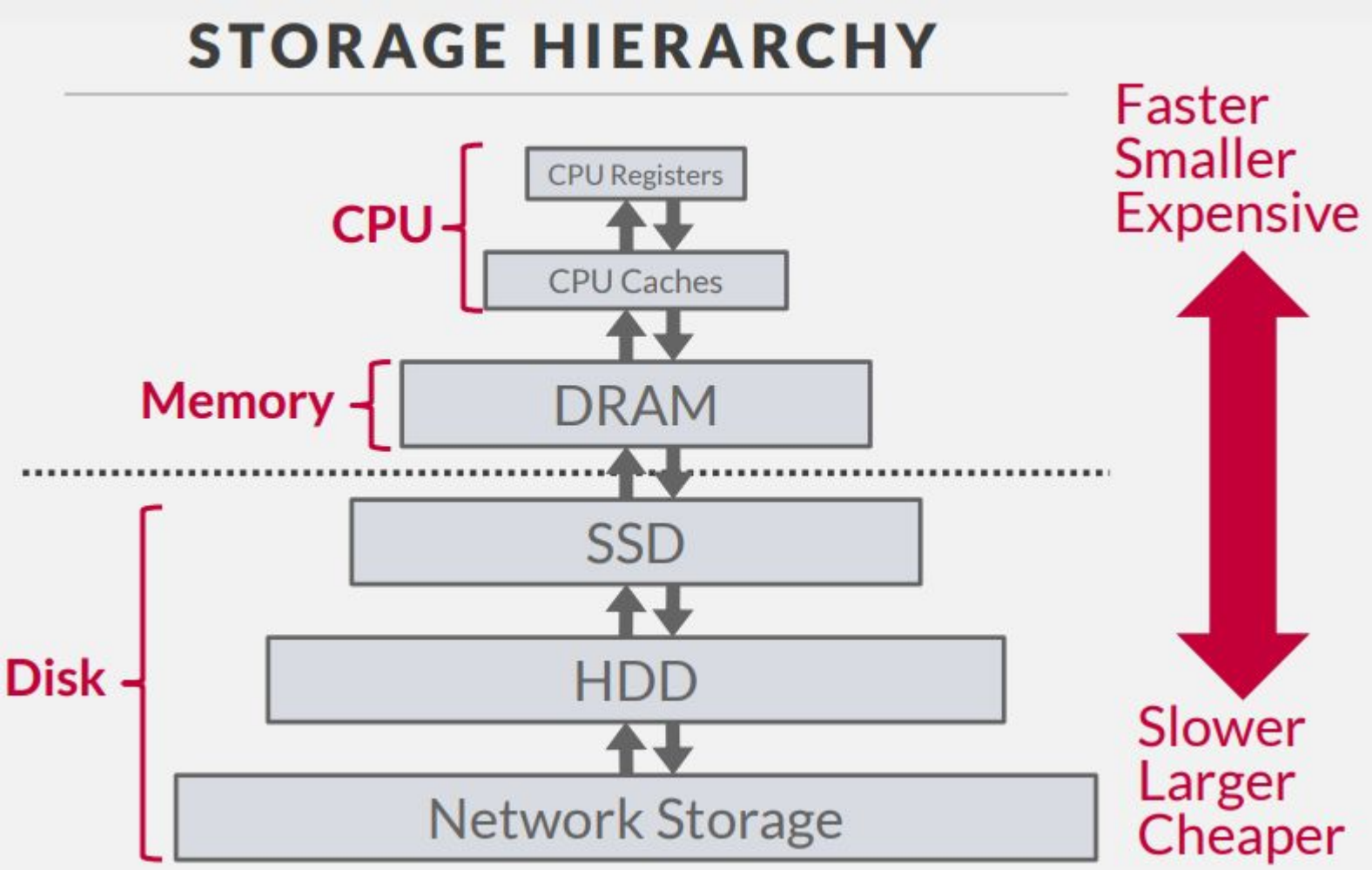
- SGBD tradicional armazena dados no disco
- Isso é que mais implica no projeto de um SGBD
  - Leitura/Read: transfere dados do disco para a memória RAM
  - Escrita/Write: transfere dados da memória RAM para o disco
  - Ambas as operações têm um custo muito alto (tempo) e devem ser planejadas cuidadosamente

Hieraquia de Armazenamento

- Custo: se um valor X compra 128Mb de RAM, com o mesmo valor compra-se 7.5Gb de disco.
- Memória RAM é volátil
- Hierarquia:

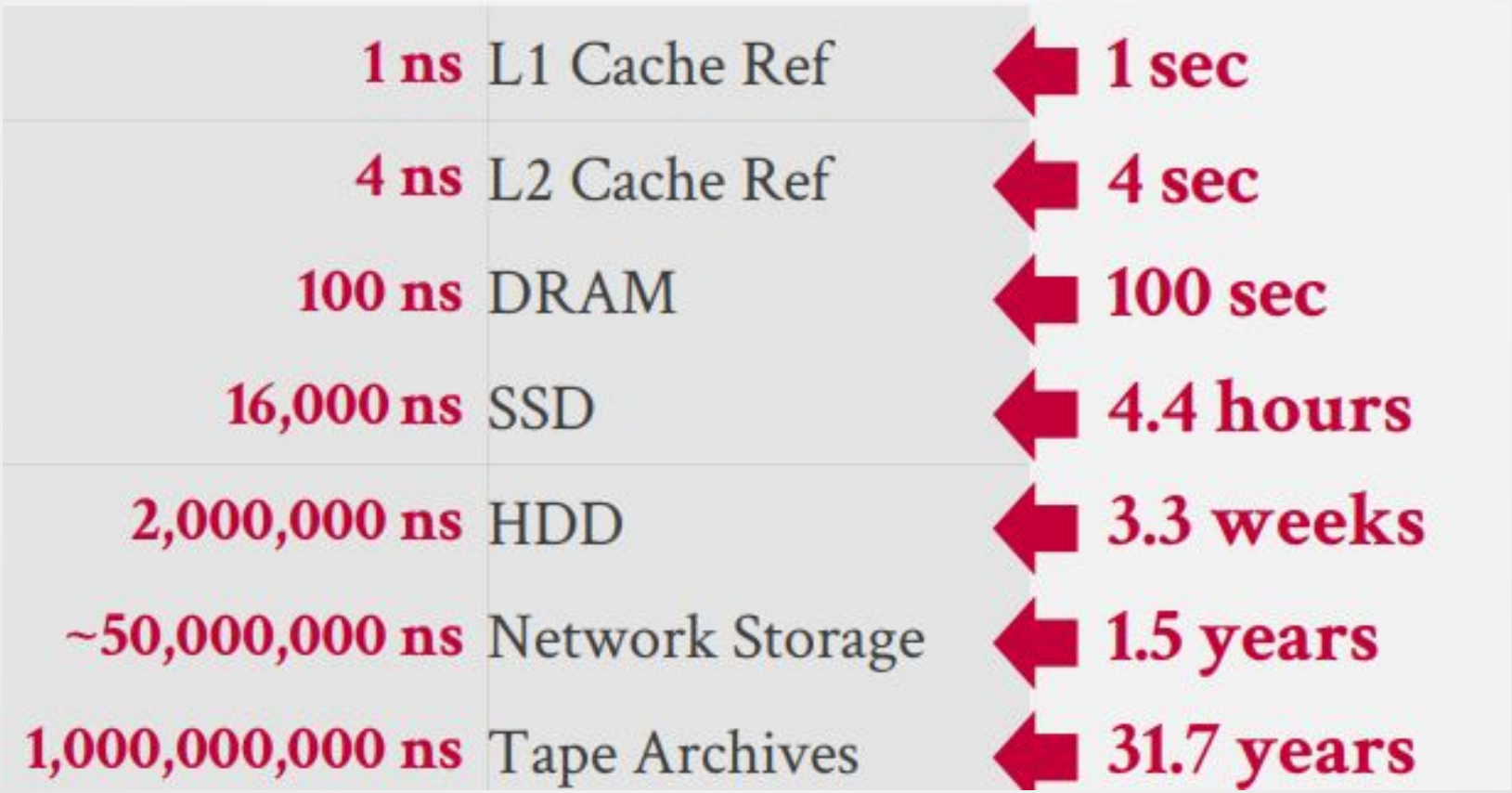


Hieraquia de Armazenamento

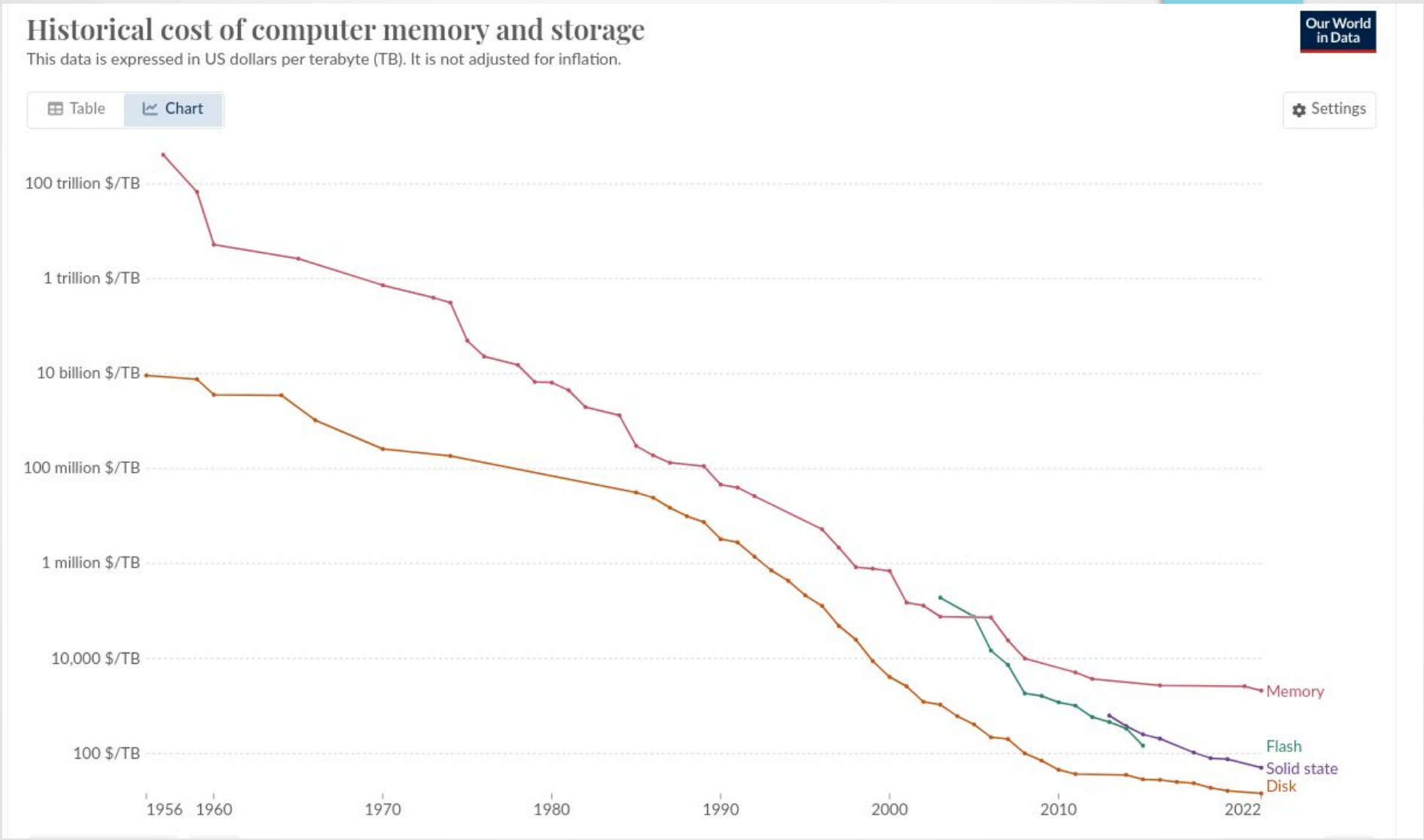


Hieraquia de Armazenamento

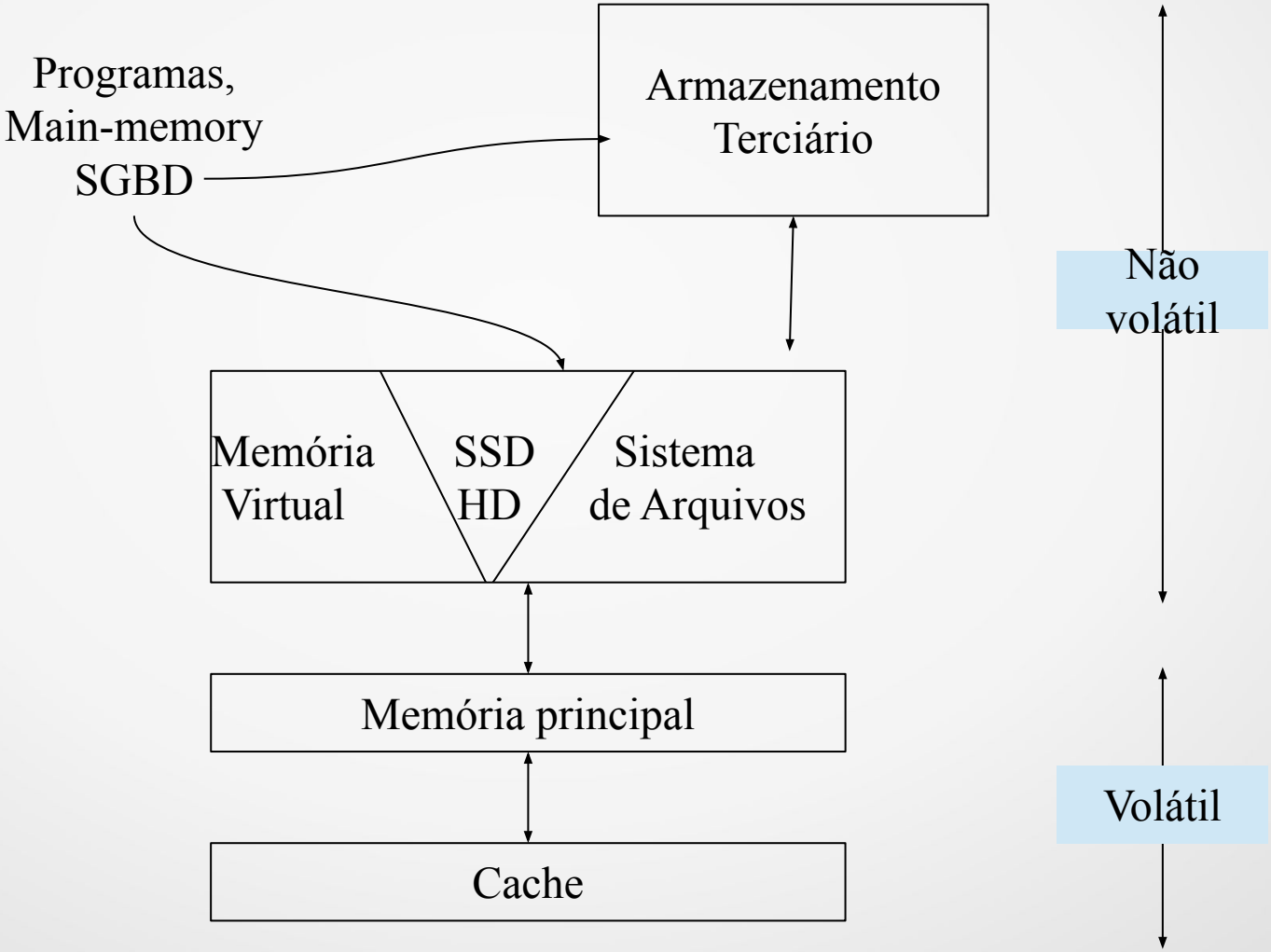
- Acesso a um bloco



Custo Hitórico (até 2022)



Arquitetura



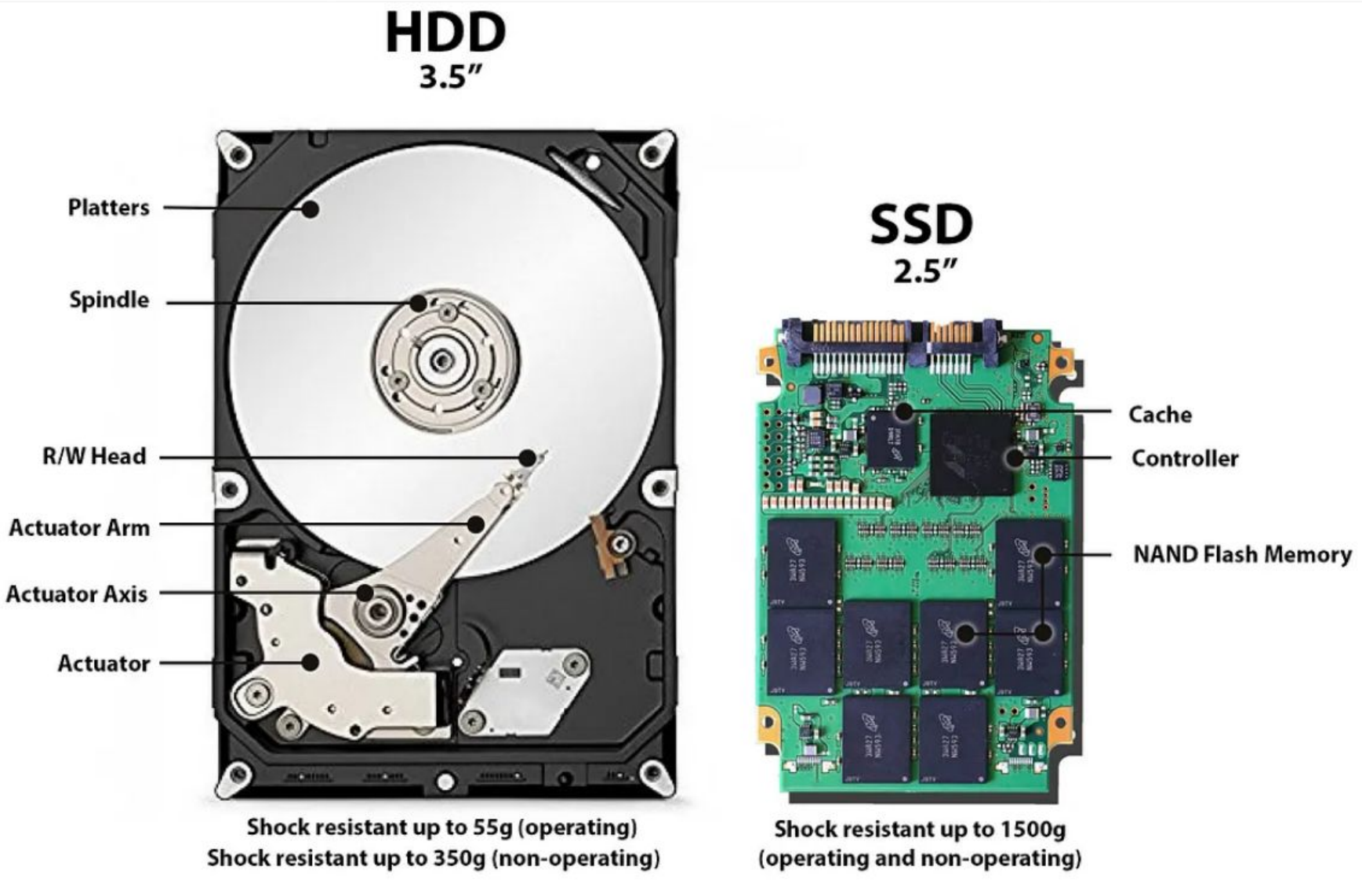
Classificação

- Velocidade que os dados são acessados
- Custo por byte
- Confiança
- Perda de dados em falha do sistema ou de energia
- Falha do dispositivo de armazenamento
- O armazenamento pode diferenciar em:
  - Volátil: perde o conteúdo quando a energia é cortada
  - Não-volátil: conteúdo persiste mesmo sem energia

Armazenamento não volátil (HD e SSD)

- Dispositivo mais utilizado para armazenamento permanente (não volátil)
- Dados são armazenados e recuperados em unidades chamadas *blocos* (ou páginas).
- Diferentemente da RAM, o tempo para recuperar um bloco no disco é mais lento.
  - Portanto, a posição relativa do bloco no disco influencia diretamente no desempenho do SGBD

Armazenamento não volátil (HD e SSD)



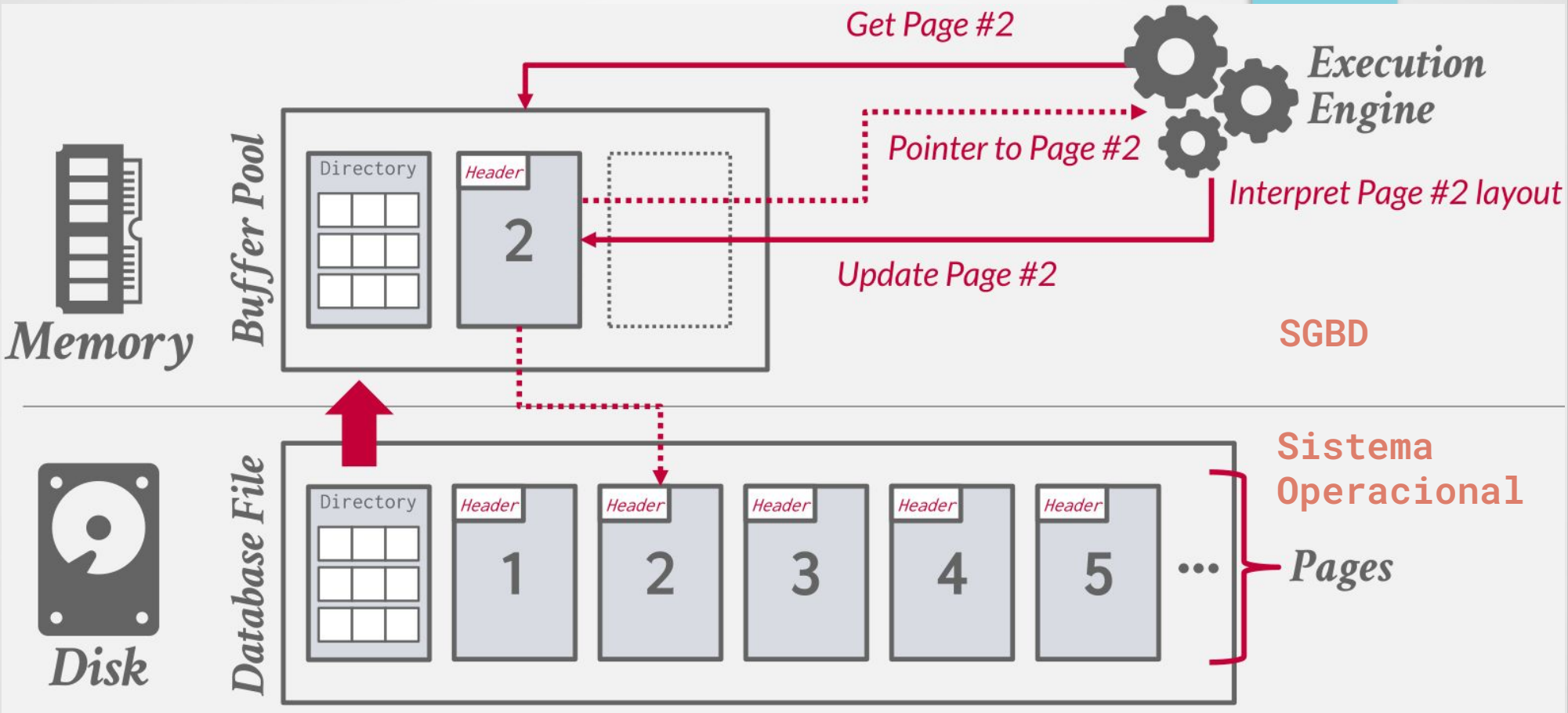
Acesso Sequencial e Randômico

- Acesso randômico em dispositivos não voláteis é quase sempre muito mais lento que o sequencial
- SGBD tenta maximizar o acesso sequencial
  - Algoritmos tentam reduzir o número de escritas em páginas randômicas, criando espaços contíguos de blocos
  - Alocar vários blocos (páginas) ao mesmo tempo é chamado uma extensão (extent)

Objetivos de Projeto

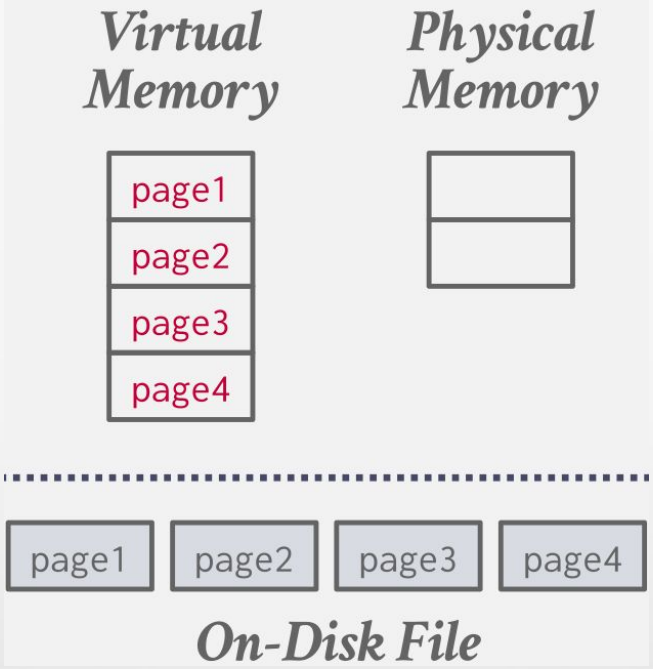
- Permitir ao SGBD gerenciar o que excede a quantidade de memória disponível
- Ler ou escrever no disco é caro
  - Gerenciar com cuidade para evitar grandes retardos e degradação do desempenho

SGBD orientado a disco

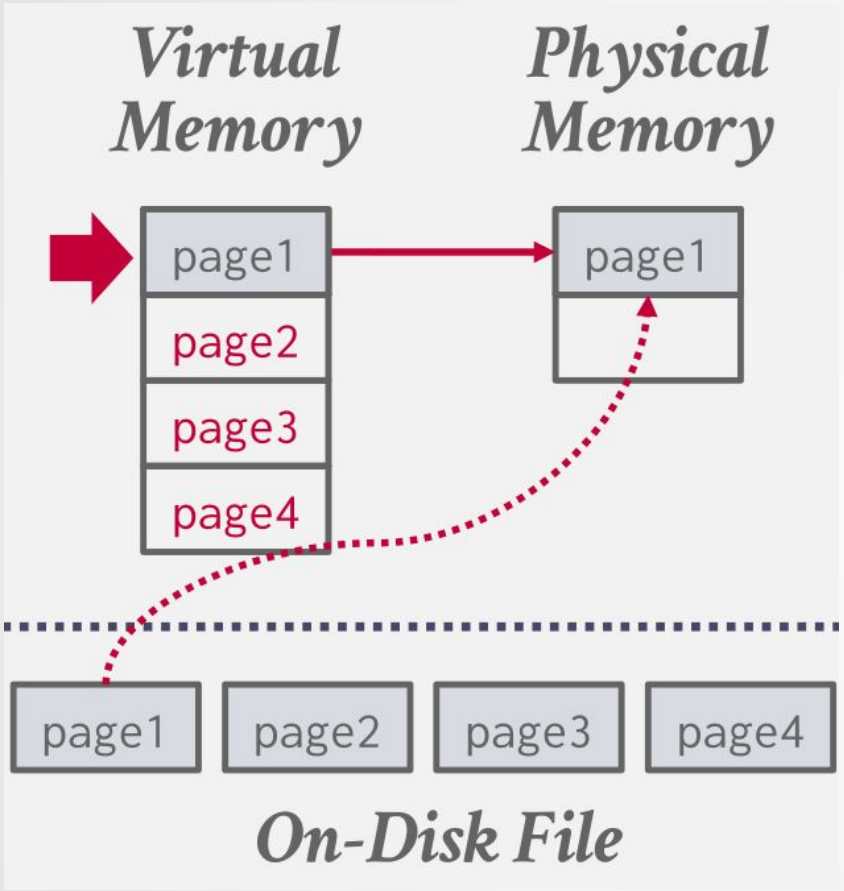


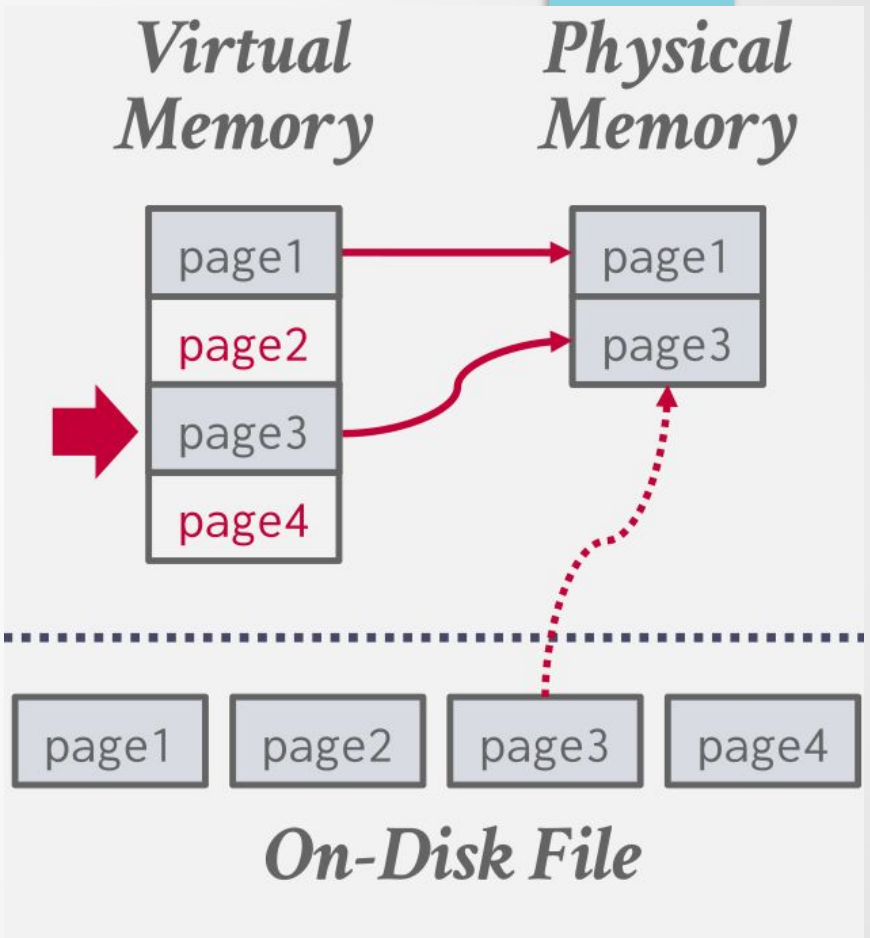
Por Que Não Usar o SO?

- SGBD pode mapear a memória do conteúdo de um arquivo para o seu espaço (buffer)
- O SO é responsável para mover os blocos (páginas) de um arquivo para dentro e fora da memória
  - SGBD não precisa ser preocupar com isso

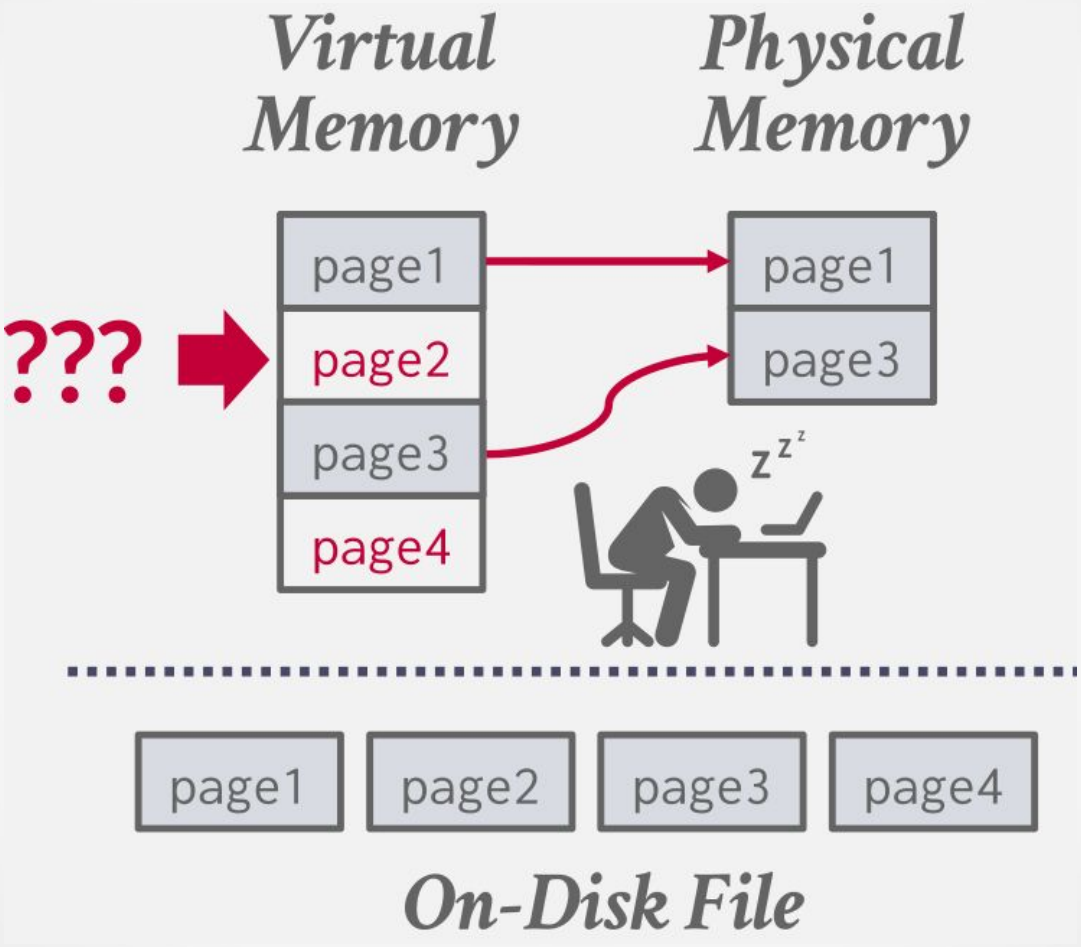


Por Que Não Usar o SO?





Por Que Não Usar o SO?



SO decide o que fazer para trazer a página 2 para a memória principal

SO Responsável: Problemas

- #1: segurança da transação
  - Páginas alteradas que não podem ainda voltar para o disco
- #2: retardos de I/O
  - SGBD não sabe quais páginas estão na memória. O SO vai retardar um processo quando a página solicitada não estiver em memória
- #3: manipulação de erros
  - Dificuldade para o SGBD validar uma dada página
- #4: desempenho
  - Diferença da estrutura dos dados do SO e do SGBD. Transações podem ser invalidadas

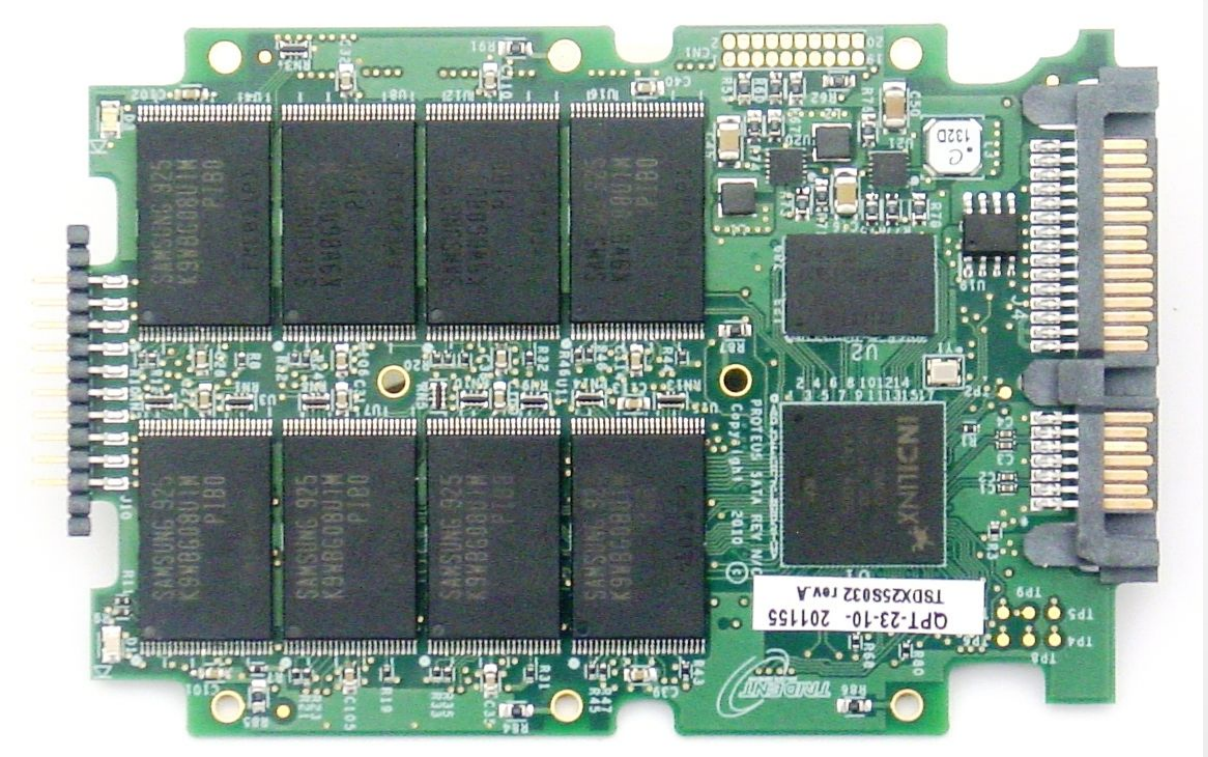
Por Que Não Usar o SO?

- SGBD (quase) sempre quer controlar as suas coisas e pode fazer a tarefa melhor que o SO
  - Descarregar páginas alteradas para o disco na ordem correta
  - Pré-buscar páginas de forma ótima
  - Política de troca de páginas no buffer
  - Escalonamento de processos/transações

O SO não é nosso amigo 😎

SSDs

- Solid State Disk (SSD)



SSD

- Solid State Drive (SSD)
  - Sem tempo de busca (seek time)
  - Sem latência de rotação (rotational latency)
- Problema
  - Custo de escrita e leitura são diferentes
  - Todos os algoritmos de acesso a disco e buffer de um SGBD devem ser adequados ao funcionamento dos SSDs.

SSD

	SLC	MLC	TLC	HDD	RAM
P/E cycles	100k	10k	5k	*	*
Bits per cell	1	2	3	*	*
Seek latency (µs)	*	*	*	9000	*
Read latency (µs)	25	50	100	2000-7000	0.04-0.1
Write latency (µs)	250	900	1500	2000-7000	0.04-0.1
Erase latency (µs)	1500	3000	5000	*	*
Notes	* metric is not applicable for that type of memory				

Visão dos Dados

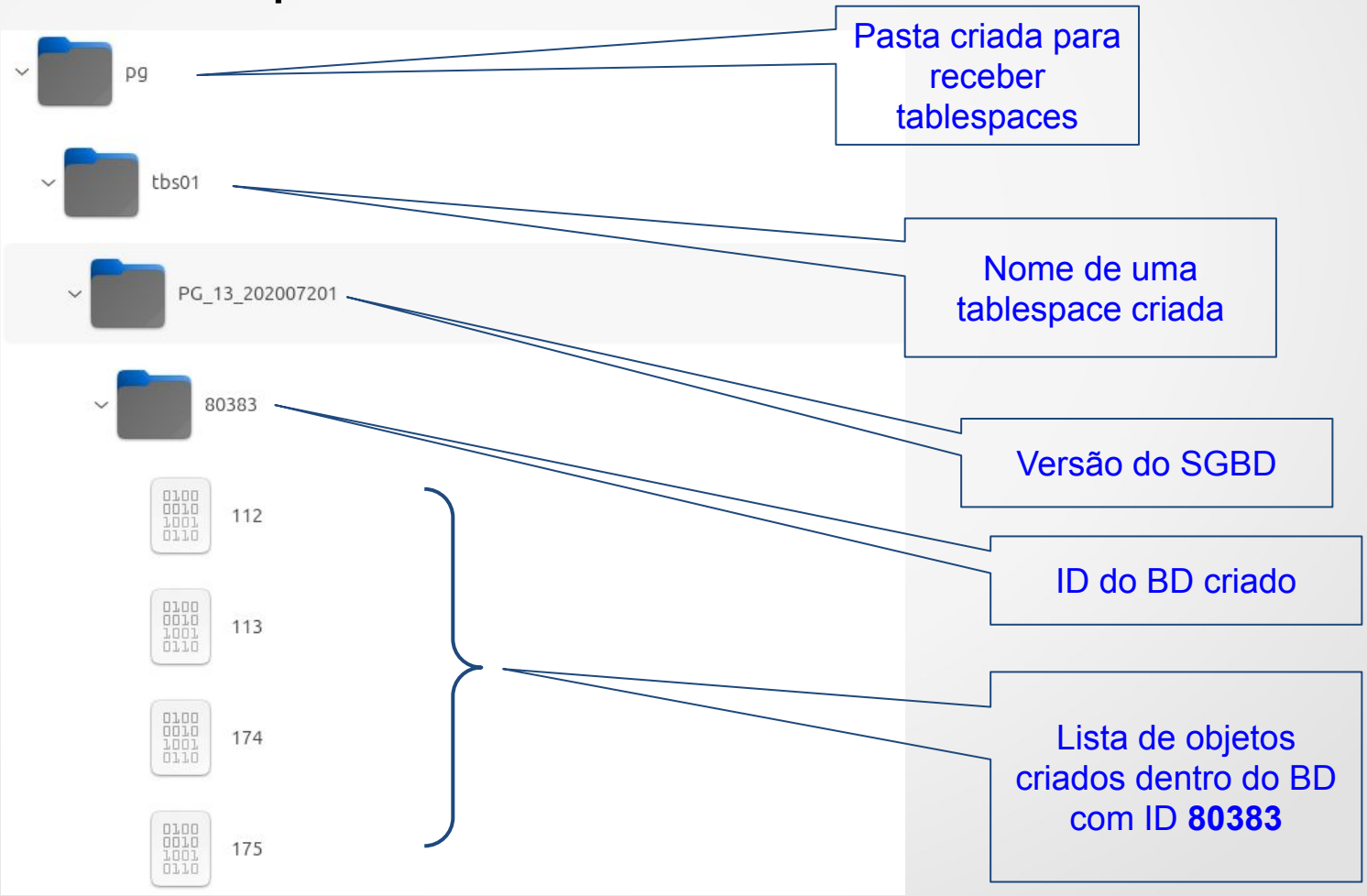
- Como o SGBD representa o banco de dados em arquivos no disco
  - SGBD: conjunto de tuplas na memória
  - Gerenciador de arquivos: um conjunto de páginas na memória
  - Gerenciador de disco: um conjunto de blocos no disco
  - Gerenciador de buffer: carga de blocos do disco para a memória ou vice-versa.

Arquivos em Disco

- O SGBD armazena um banco de dados como um ou mais arquivos no disco, tipicamente em um formato proprietário
  - O SO não “sabe” nada sonre o conteúdo destes arquivos
- O gerenciador de armazenamento é responsável por manter os arquivos de um BD.

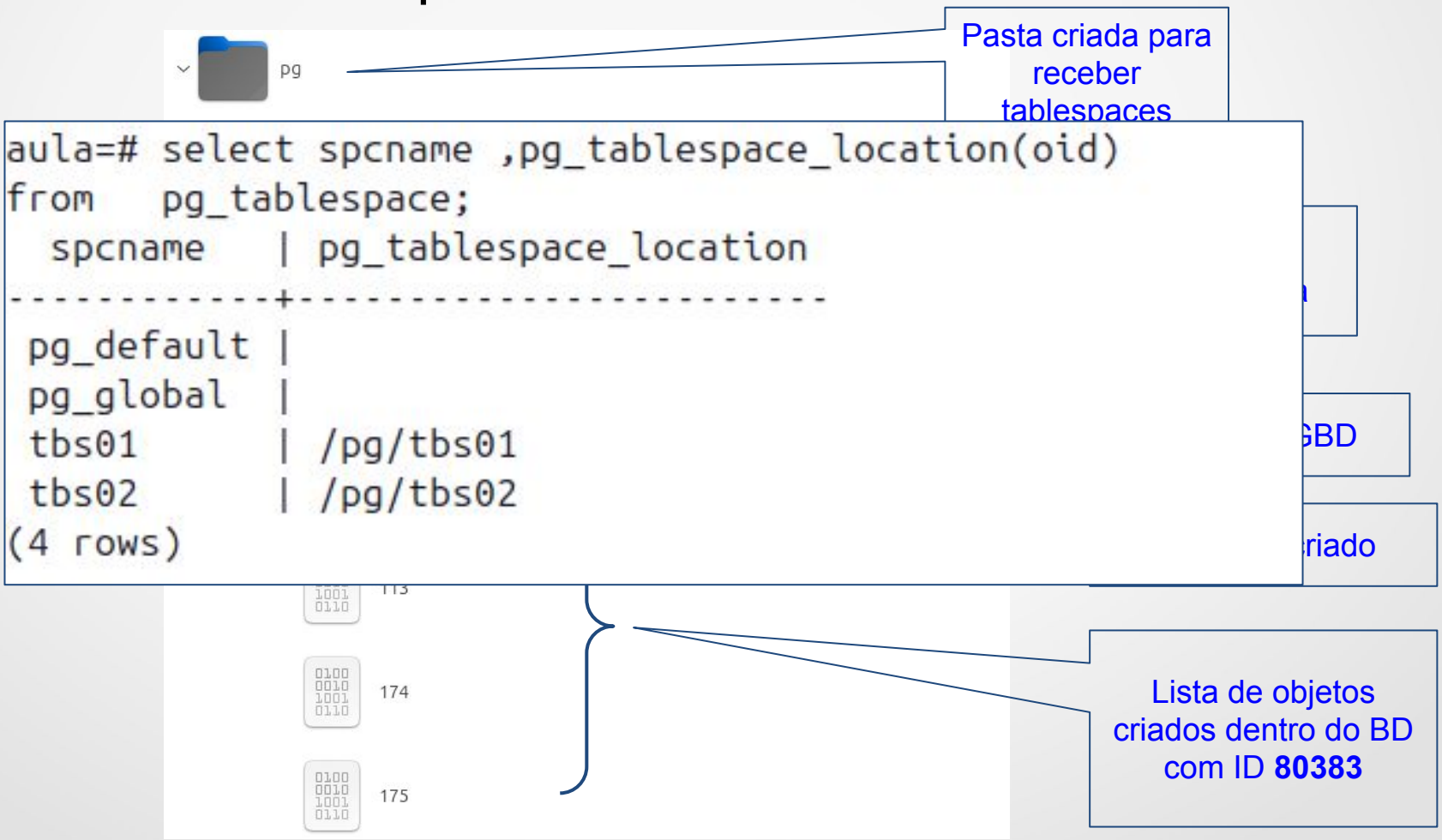
Arquivos em Disco

- PostgreSQL cria uma pasta para o banco de dados usando uma tablespace definida



Arquivos em Disco

- PostgreSQL cria uma pasta para o banco de dados usando uma tablespace definida



Gerenciador de Armazenamento

- Responsável por manter os arquivos do banco de dados
- Organiza os arquivos como uma colação de páginas (ou blocos)
  - Rastreia a escrita/leitura dos dados para as páginas
  - Rastreia o espaço disponível

Páginas

- Uma página é um bloco de dados de tamanho fixo
  - Pode conter túplas, metadados, índices, registros de logs
  - A maioria dos sistemas não mistura os tipos de páginas
  - Alguns sistemas fazem com que as páginas sejam auto-contidas
  - Cada página tem um identificador utilizado para a sua localização física

Páginas

- Existem três noções diferentes de páginas para o SGBD
  - Página no HW (4Kb)
  - Página do SO (4Kb, para 64 bits: 2Mb a 1Gb)
  - Página do banco de dados (512b a 32Kb)
- Uma página do HW é o maior bloco de dados que um dispositivo de armazenamento pode garantir escritas seguras.

Tamanhos padrão para alguns SGBD:

- 4Kb: SQLite, Oracle, DB2, RocksDB
- 8Kb: SQLServer, PostgreSQL
- 16Kb: MySQL

Páginas

- Existem três noções diferentes de páginas para o SGBD
  - Página no HW (4Kb)

```
aula=# SELECT name, setting, short_desc, extra_desc
       FROM pg_settings
       WHERE name like '%block%' or short_desc LIKE '%block%';
name            |setting |   short_desc                                  | extra_desc
----------------+--------+-----------------------------------------------+------------
 block_size     | 8192   | Shows the size of a disk block.               |
 wal_block_size | 8192   | Shows the block size in the write ahead log. |
(2 rows)
```

seguras.

Tamanhos padrão para alguns SGBD:

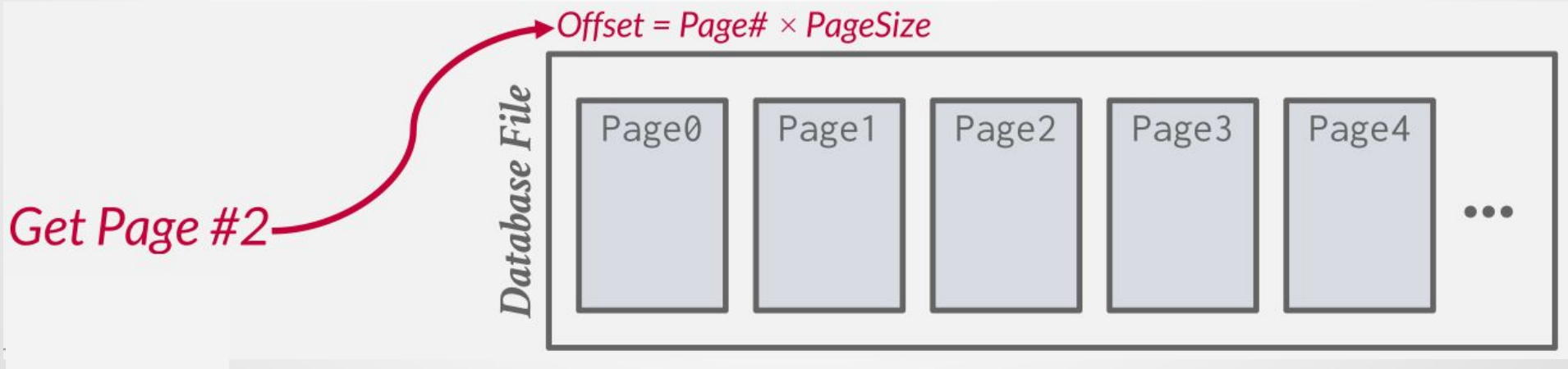
- 4Kb: SQLite, Oracle, DB2, RocksDB
- 8Kb: SQLServer, PostgreSQL
- 16Kb: MySQL

Arquitetura de Armazenamento nas Páginas

- SGBDs gerenciam páginas em arquivos no disco de diferente formas
  - **Heap file**
  - Àrvore
  - Organização sequêncial / ordenada (ISAM)
  - Organização espalhada (hashing)

Arquivos Heap

- É uma coleção desordenada de páginas com tuplas armazenadas em ordem randômica
  - São necessárias apenas quatro operações: criar (create), obter (get), escrever (write) e excluir (delete)
- Fácil de encontrar páginas se existir apenas um arquivo

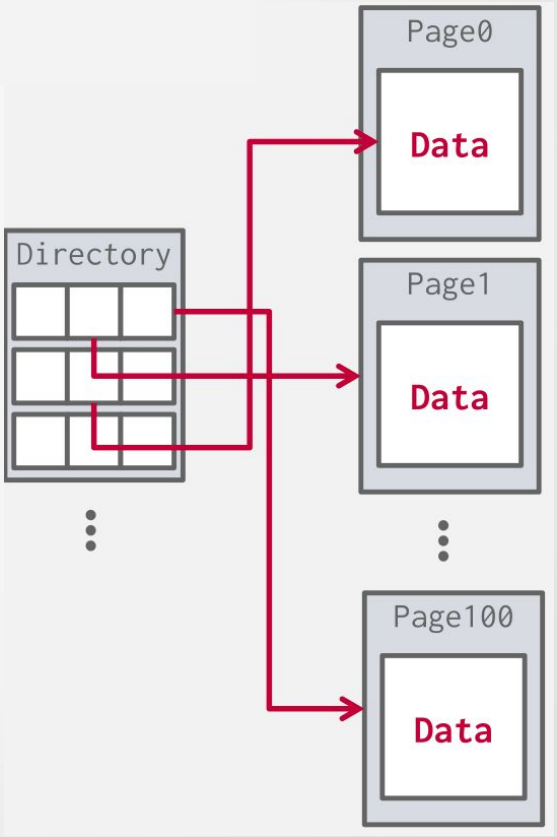


Arquivos Heap

- É uma coleção desordenada de páginas com tuplas armazenadas em ordem randômica
  - São necessárias apenas quatro operações: criar (create), obter (get), escrever (write) e excluir (delete)
- Fácil de encontrar páginas se existir apenas um arquivo
- Necessita de metadados para rastrear quais páginas existem em vários arquivos e quais têm espaços disponíveis

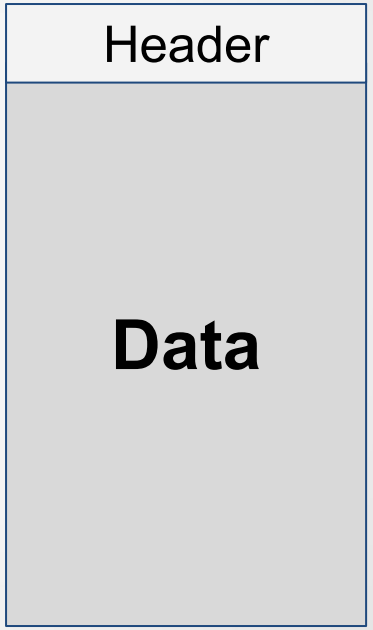
Arquivos Heap: Diretório

- São mantidas páginas especiais que rastreiam a localização das páginas de dados nos arquivos do banco
  - Deve-se garantir que o diretório esteja sincronizado com as páginas de dados
- O diretório também armazena metadados sobre as páginas disponíveis
  - Número de slots livres por página
  - Número de páginas vazias / livres



Page Header (Cabeçalho da Página)

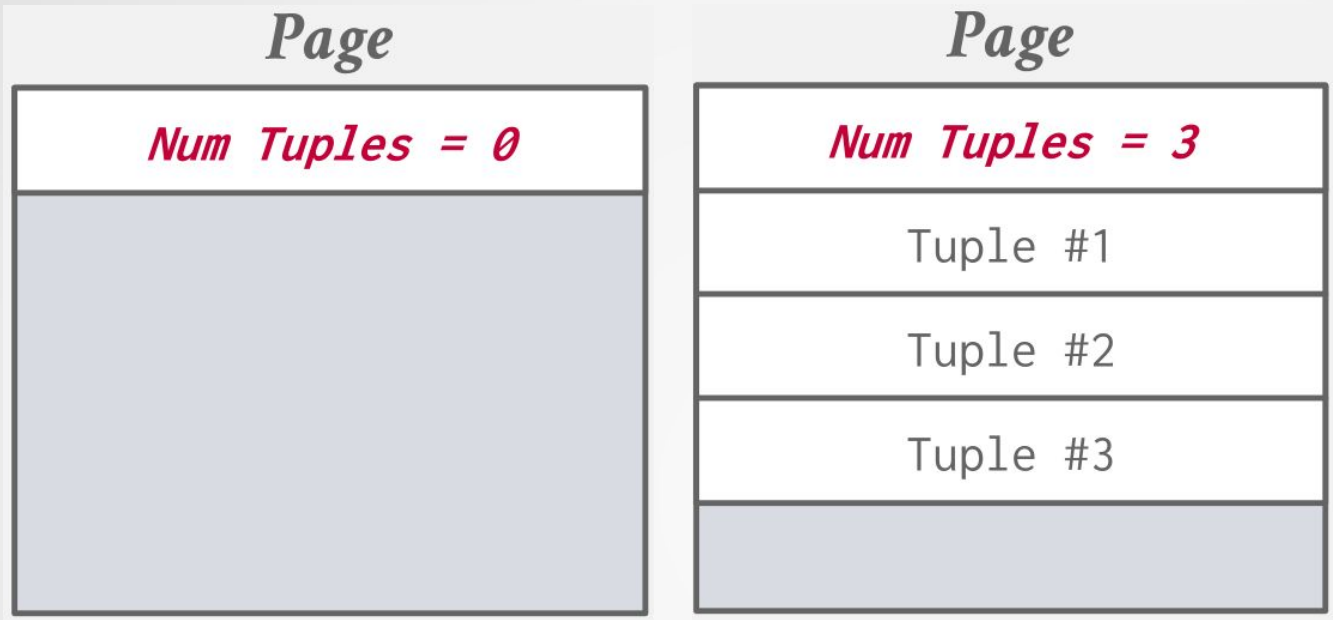
- Toda página tem um cabeçalho com metadados com o seu conteúdo
  - Tamanho da página
  - Checksum
  - Versão do SGBD
  - Transações da página
  - Dados comprimidos (metadados)
  - Informações do esquema
  - Sumarização de dados
- Alguns SGBD exigem páginas autocontidas



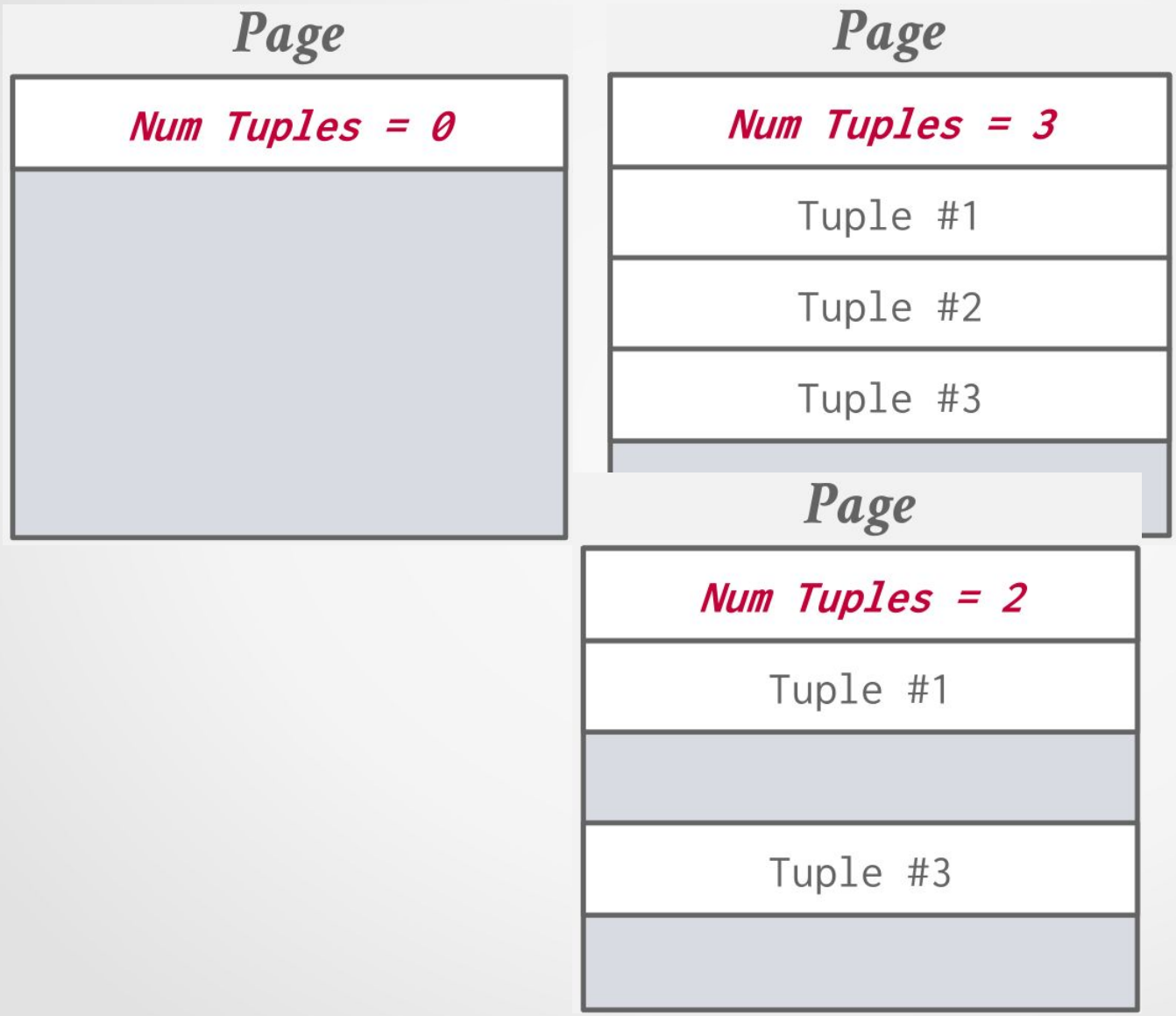
Layout da Página

- Independent da arquitetura de armazenamento da página, é necessário decidir como organizar os dados dentro da mesma
  - Armazenamento de tuplas em um modelo orientado a armazenamento de linhas
  - Armazenamento orientado a tuplas

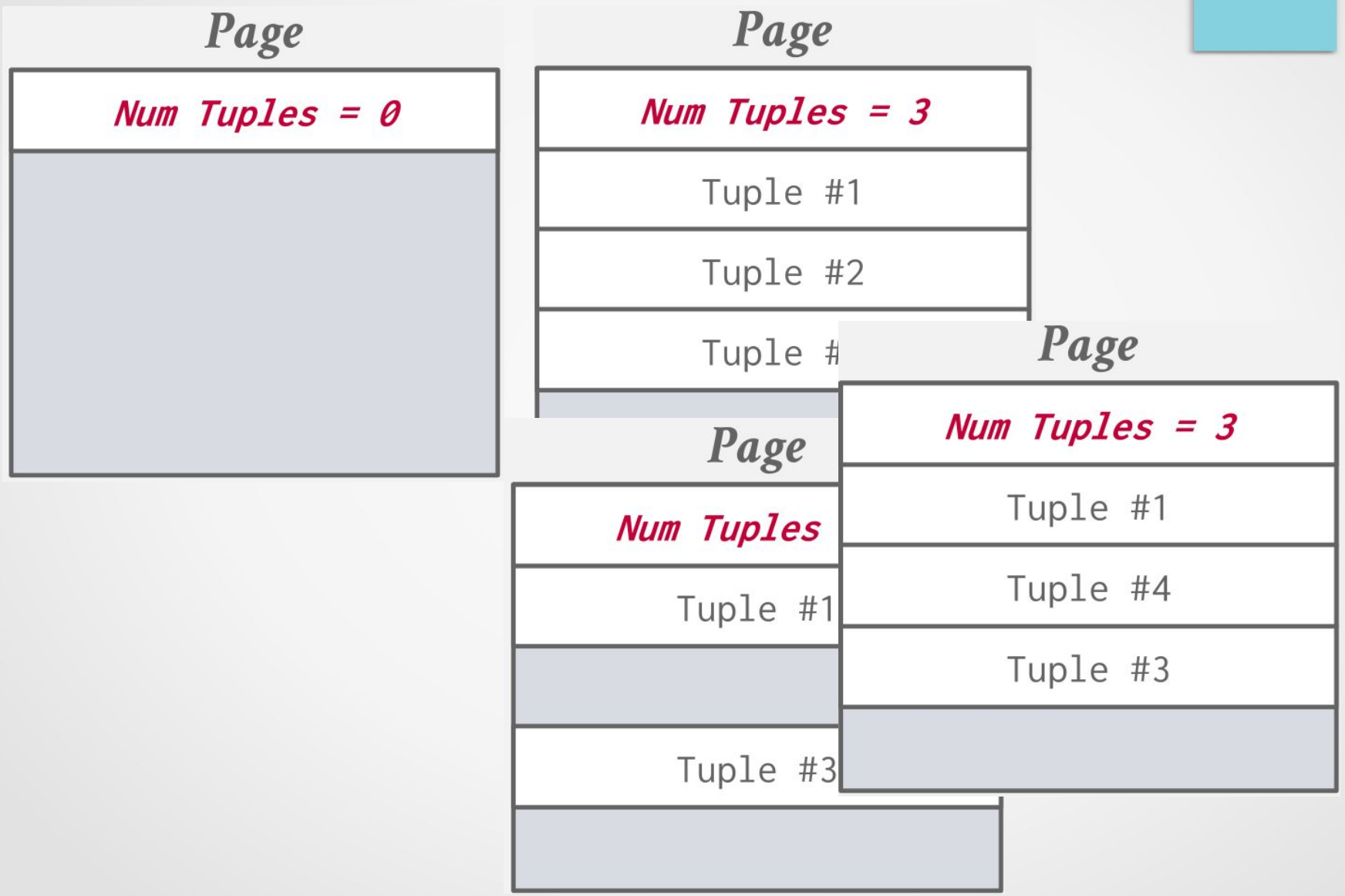
Orientado a Tuplas



Orientado a Tuplas

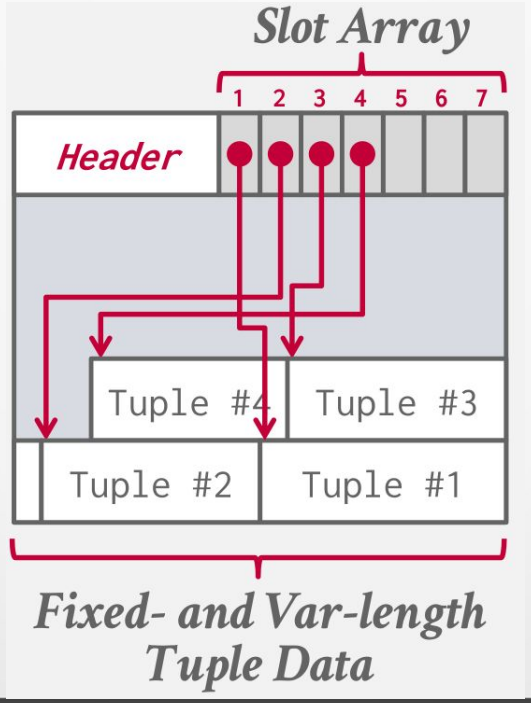


Orientado a Tuplas



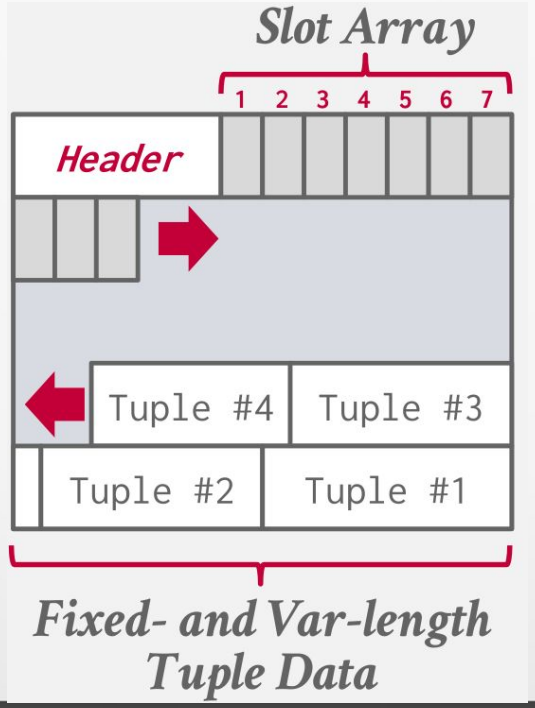
Orientado a Slots

- Uma matriz de slots mapeia slots para a posição da tupla
- O cabeçalho rastreia
  - O número de slots usados
  - O deslocamento (offset) da localização inicial do último slot usado



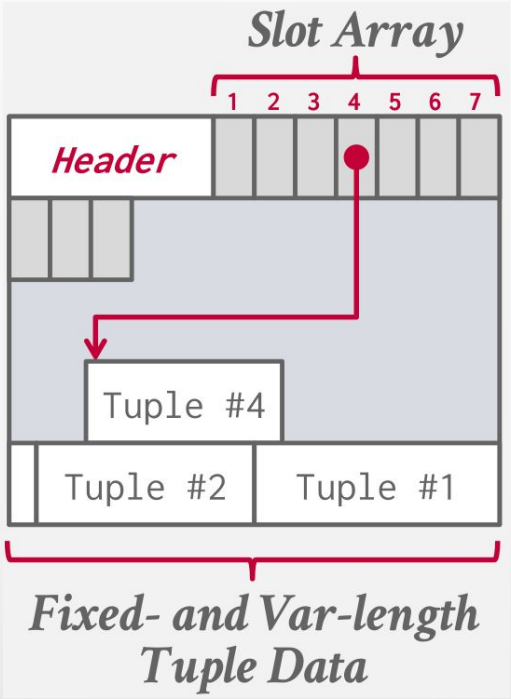
Orientado a Slots

- Uma matriz de slots mapeia slots para a posição da tupla
- O cabeçalho rastreia
  - O número de slots usados
  - O deslocamento (offset) da localização inicial do último slot usado



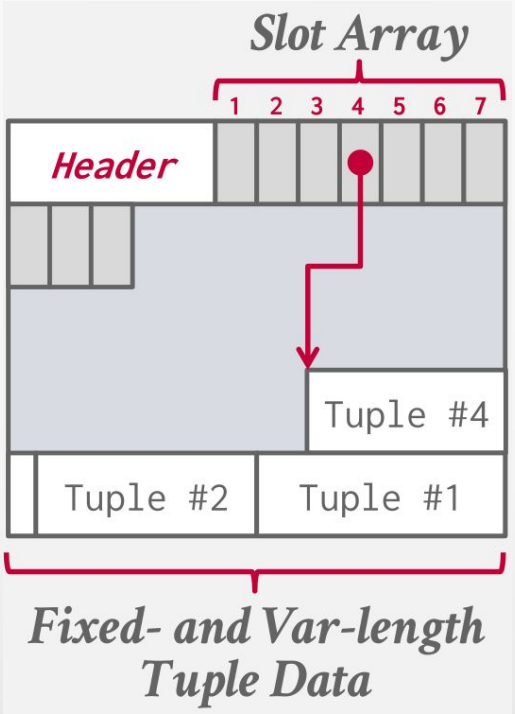
Orientado a Slots

- Uma matriz de slots mapeia slots para a posição da tupla
- O cabeçalho rastreia
  - O número de slots usados
  - O deslocamento (offset) da localização inicial do último slot usado



Orientado a Slots

- Uma matriz de slots mapeia slots para a posição da tupla
- O cabeçalho rastreia
  - O número de slots usados
  - O deslocamento (offset) da localização inicial do último slot usado



Record ID

- O SGBD associa para cada tupla lógica um identificador único que representa a localização física
  - Pode ser composto pelo IDs do arquivo e da página, e o número do slot
  - A maioria dos SGBDs calcula em tempo de execução (coletando os dados presentes nos headers)
- É tentador, mas as aplicações não devem/deveriam usá-lo para localizar uma tupla

Record ID

- O SGBD associa para cada tupla lógica um identificador únic
  - Po
    nú
  - A
    (c
- É ten -lo para

```
aula=# select * from table01 ;
 id  | val
-----+-----
 100 | aaa
 101 | bbb
 102 | ccc
(3 rows)

aula=# select ctid,* from table01 ;
 ctid  | id  | val
-------+-----+-----
 (0,1) | 100 | aaa
 (0,2) | 101 | bbb
 (0,3) | 102 | ccc
(3 rows)
```

Record ID

-
-

```
aula=# delete from table01 where id=101;
DELETE 1
aula=# select ctid,* from table01 ;
 ctid  | id  | val
-------+-----+-----
 (0,1) | 100 | aaa
 (0,3) | 102 | ccc
(2 rows)
aula=# insert into table01 (id, val) values (103,'ddd');
INSERT 0 1
aula=# select ctid,* from table01 ;
 ctid  | id  | val
-------+-----+-----
 (0,1) | 100 | aaa
 (0,3) | 102 | ccc
 (0,4) | 103 | ddd
```

Record ID

- O SGBD [illegible] identificador [illegible]

```
aula=# vacuum full table01 ;
VACUUM
aula=# select ctid,* from table01 ;
 ctid  | id  | val
-------+-----+-----
 (0,1) | 100 | aaa
 (0,2) | 102 | ccc
 (0,3) | 103 | ddd
(3 rows)
```

- [illegible]-lo para localizar uma tupla

Otimização do Acesso

- Bloco: área contígua de setores em uma memória de armazenando secundária
  - O dado é transferido entre a memória secundária e a memória principal por blocos
  - O tamanho de um bloco deve ser múltiplo do tamanho dos setores do disco.

Um bloco no hardware é de 4K

Otimização do Acesso

- Organização do arquivo: otimiza o tempo de acesso ao bloco organizando os blocos para corresponder como os dados serão acessados
  - O sistema de arquivo tenta alocar blocos contiguos (*e.g.* 8 ou 16 blocos) para um arquivo

Otimização do Acesso - Flash Memories

- As memórias flash têm suas particularidades
  - A latência da leitura é menor que a escrita
  - Quando o SGBD precisa ler dados mais que escrever
    - Ocorre o *read stall*
      - A operação de leitura é bloqueada pela da escrita

	Hard Disk	Flash SSDs				
	(HDD)	SSD-A	SSD-B	SSD-C	SSD-D	OpenSSD
Read Stalls	0%	28%	28%	36%	33%	32%

Otimização do Acesso - Flash Memories

- SGBDs devem ter algoritmos diferentes
  - HD
    - Utilizam os algoritmos tradicionais
  - SSD
    - Tentam evitar escritas desnecessárias
- Papel dos gerenciadores de buffer e de disco (HD e SSD)

Otimização do Acesso

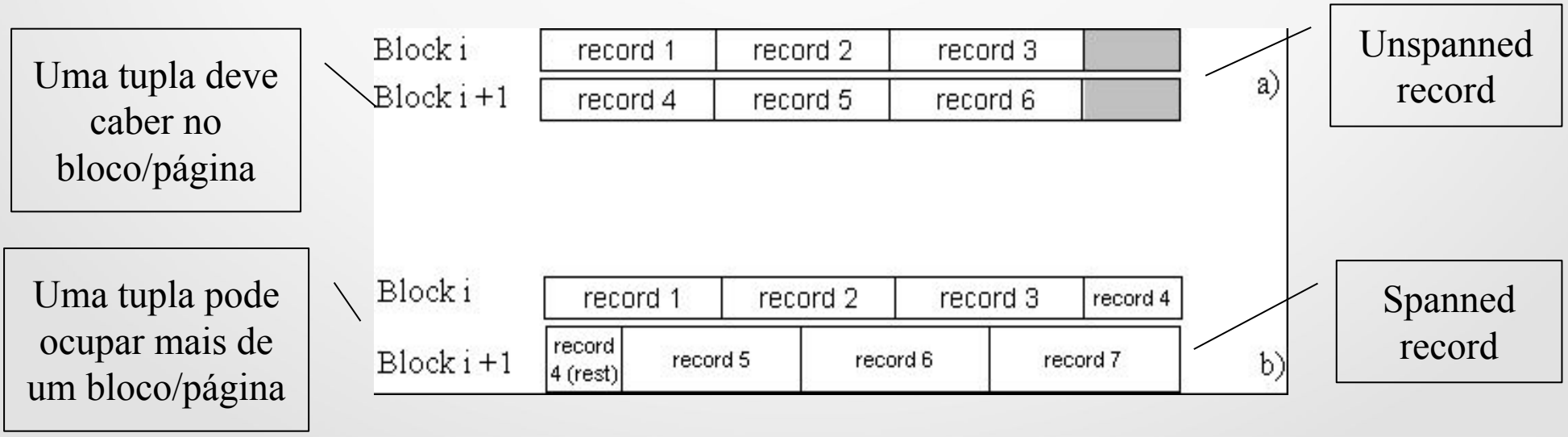
- Um arquivo do BD é particionado em unidades de tamanho fixo chamadas de blocos.
  - Unidade para alocação e transferência de dados
- O SGBD procura minimizar o número de transferência de blocos entre o disco e a memória.
  - Pode-se obter a redução através do armazenamento do máximo possível de páginas na memória

Otimização do Acesso

- Buffer: porção da memória principal que armazena cópias dos blocos do disco.
  - O buffer é organizado em páginas que, geralmente, se relacionam 1:1 com os blocos
- Gerenciador de buffer: subsistema de um SGBD responsável para gerenciar espaços no buffer

Organização Buffer

- Às vezes uma tupla/tabela não cabe em uma página do buffer
  - Isso pode ocorrer no disco
- É necessário ponteiros que apontam para qual página/bloco a continuação da tupla/tabela se encontra

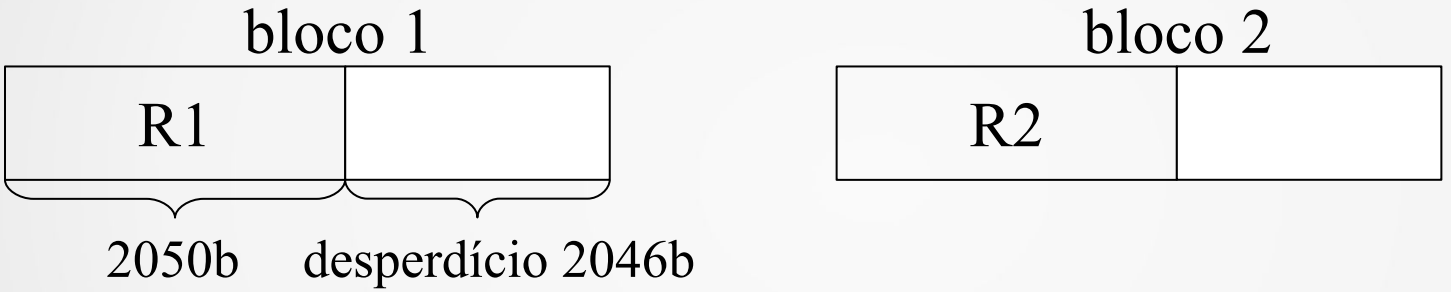


Organização Buffer

- Para arquivos de tamanho fixo (R bytes) e tamanho de bloco B > R (organização não espalhada - *unspanned*)
  - Fator de blocagem fb=Trunc(B/R)
  - Bytes desperdiçados B-(fb*R)
- Para organização spanned (espalhada) há menos desperdício de bytes (apenas para armazenar o ponteiro)

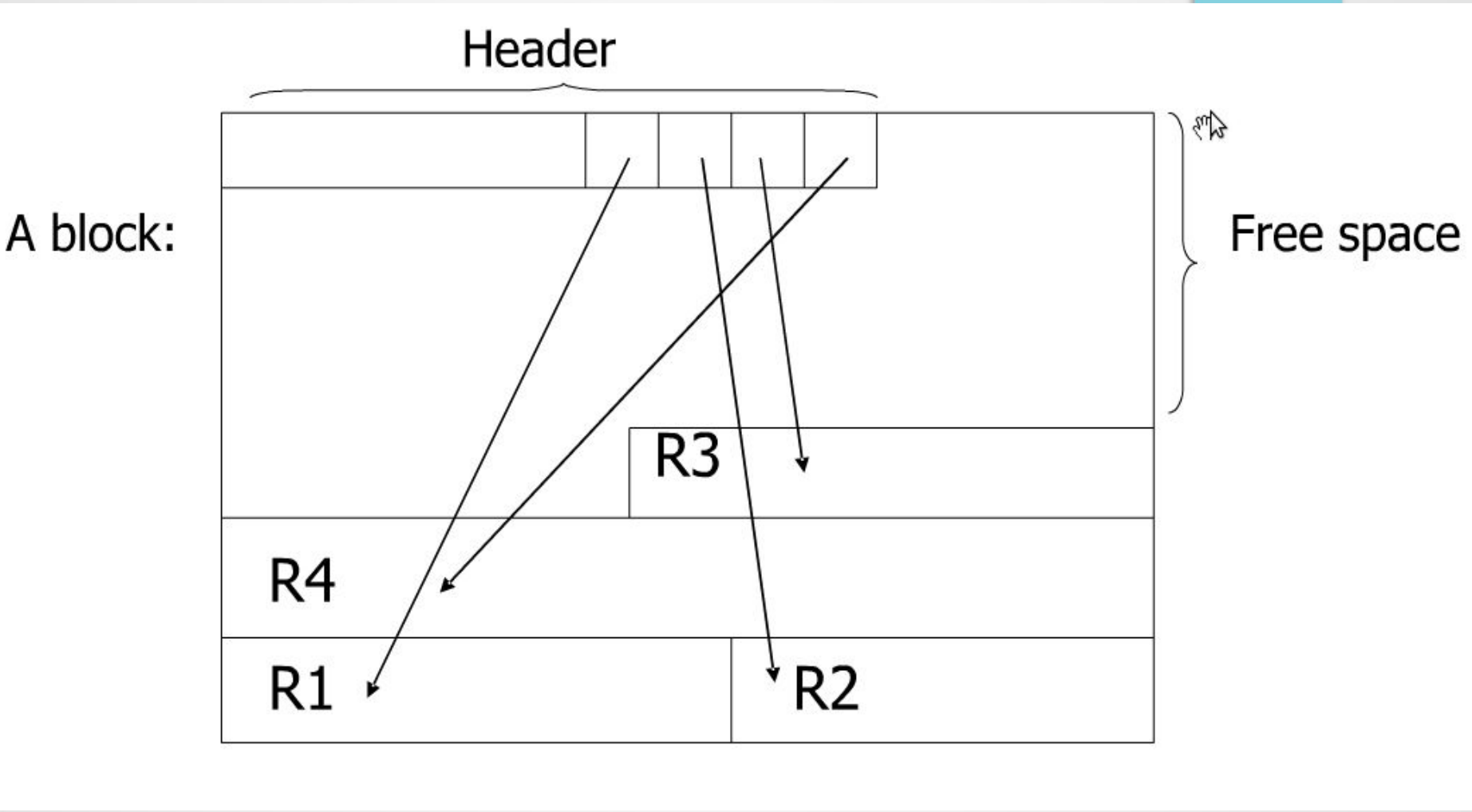
Organização Buffer

- Exemplo
  - 10^6 registros, 2.050b (fixo) e bloco=4.096



  - Total desperdício: 2.046 * 10^6= 2.046.000.000b
  - Total utilizado: 4.096 * 10^6= 4.096.000.000b

Organização Buffer



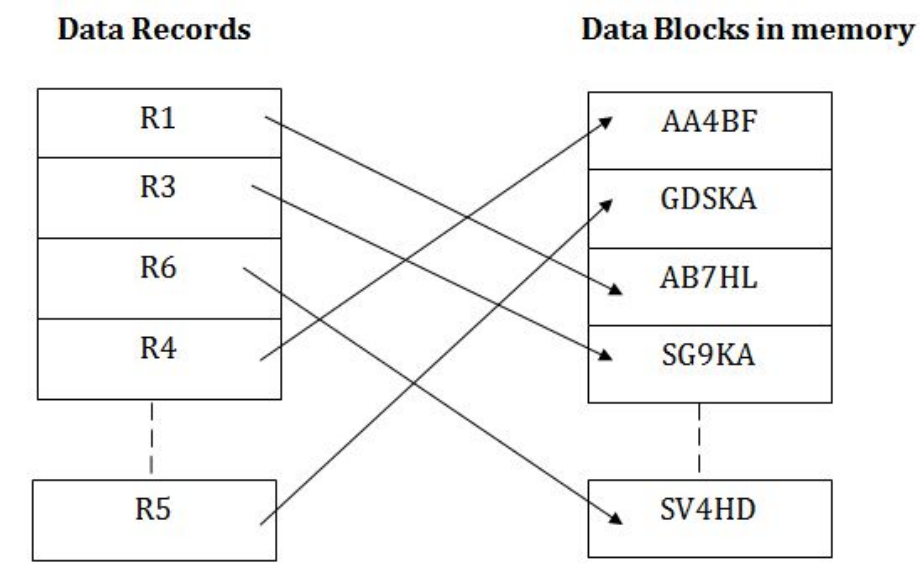
Arquivos/Tabelas

- Coleção de registros (lógicos)
- Registros são formados por *campos* ou *atributos*
- Campos possuem nome, tipo, tamanho
- Armazenados nos blocos armazenados na memória secundária

Código	Nome	Idade
00133	Fulano de Tal	25
00255	Ciclano de Tal	36
:	:	:
01034	Beltrano de Tal	18

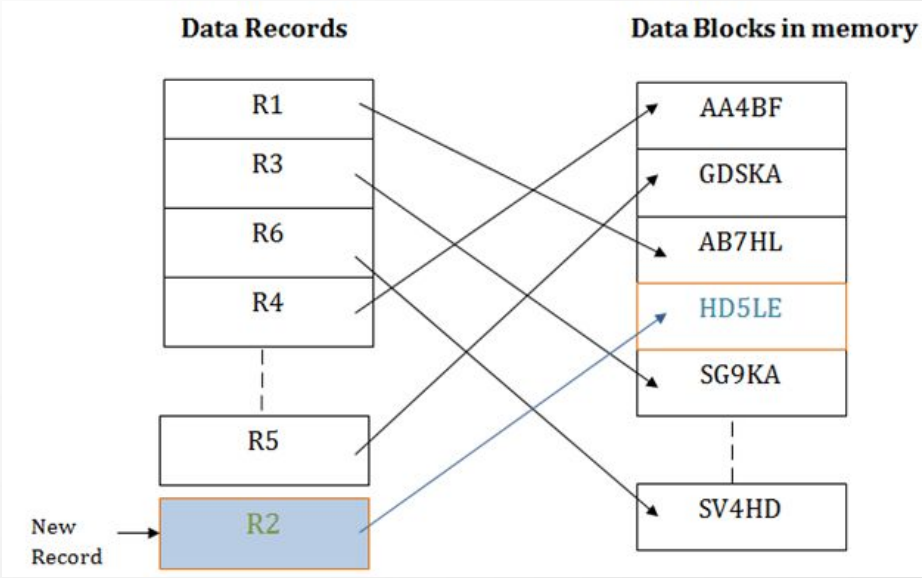
Organização de Arquivos

- Heap: um registro/uma tupla pode ser colocado em qualquer posição do arquivo onde exista espaço (registros desorganizados)
  - Forma mais simples de implementar
  - Facilita a escrita
  - Dificulta a consulta
  - Estratégia para exclusão



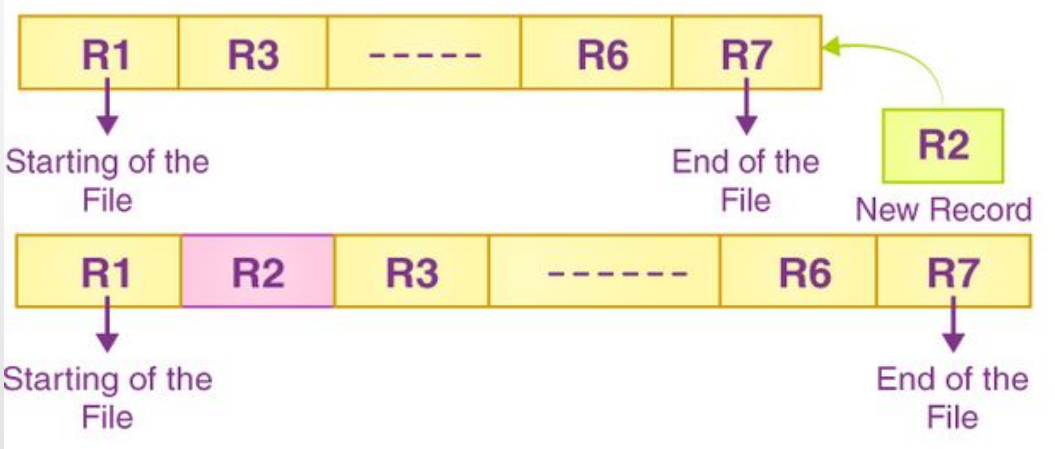
Organização de Arquivos

- Heap: um registro/uma tupla pode ser colocado em qualquer posição do arquivo onde exista espaço (registros desorganizados)
  - Forma mais simples de implementar
  - Facilita a escrita
  - Dificulta a consulta
  - Estratégia para exclusão



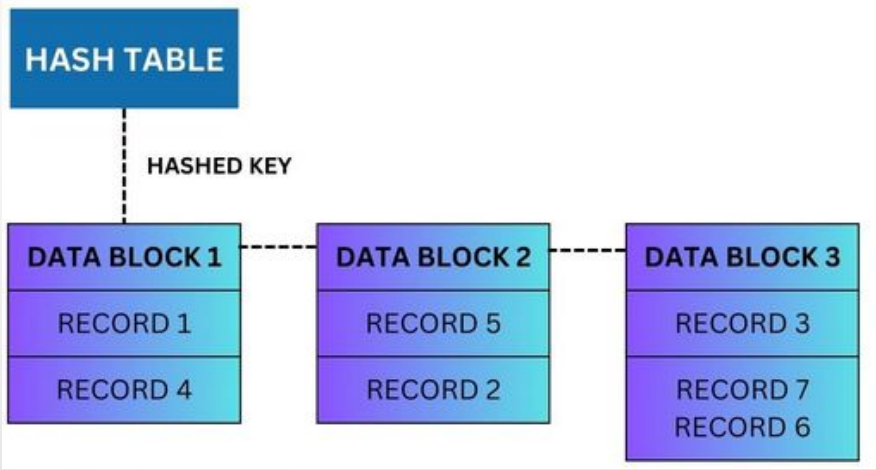
Organização de Arquivos

- Sequencial (ordenado): armazena os registros (tuplas) em uma ordem sequencial baseada em um valor (chamado chave)
  - Implementação mais complexa
    - Alocar blocos antecipadamente
  - Escrita mais complexa
    - Consulta mais simples (pela chave)



Organização de Arquivos

- Hashing: uma função de hashing calcula onde o registro/tupla será armazenado. Esta função é baseada em um atributo/chave do registro/tupla
  - Os blocos do arquivo devem ser alocados previamente
  - Escrita otimizada (caso não exista colisão)
  - Leitura otimizada pela chave e apenas igualdade



Organização de Arquivos

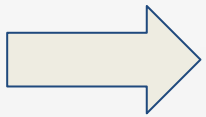
- Agrupados (clustering): registros/tuplas com a mesma chave são agrupados em blocos/páginas iguais para facilitar o acesso
  - Parecido com a estratégia ordenada mas aceita duplicações
  - E mais simples de gerenciar pois registros serão agrupados por uma determinada chave

EMPLOYEE

EMP-ID	EMP-NAME	ADDRESS	DEP-ID
1	John	Delhi	14
2	Robert	Gujarat	12
3	David	Mumbai	15
4	Amelia	Meerut	11
5	Kristen	Noida	14
6	Jackson	Delhi	13
7	Amy	Bihar	10
8	Sonoo	UP	12

DEPARTMENT

DEP-ID	DEP-NAME
10	Math
11	English
12	Java
13	Physics
14	Civil
15	Chemistry



Cluster Key

DEP-ID	DEP-NAME	EMP-ID	EMP-NAME	ADDRESS
10	Math	7	Amy	Bihar
11	English	4	Amelia	Meerut
12	Java	2	Robert	Gujarat
12		8	Sonoo	UP
13	Physics	6	Jackson	Delhi
14	Civil	1	John	Delhi
14		5	Kristen	Noida
15	Chemistry	3	David	Mumbai

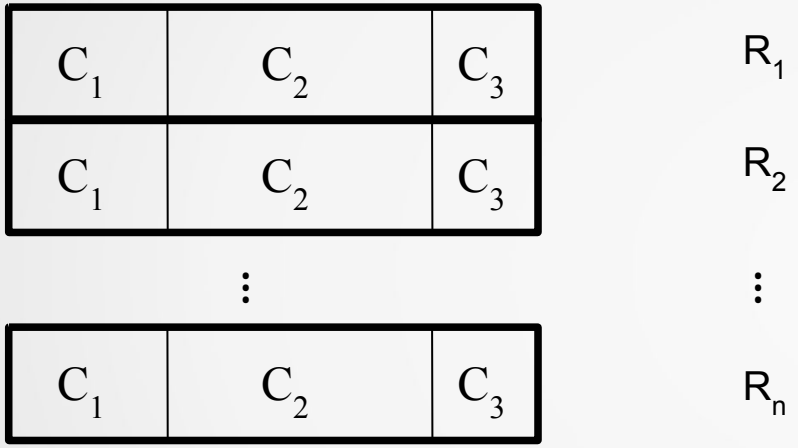
Formato Registro

- Tamanho fixo
  - Campos de tamanho fixo:
    - Número de campos fixo
    - Cada campo tem o mesmo comprimento em todos os registros.
  - Campos de tamanho variável
    - Campos têm tamanhos variáveis

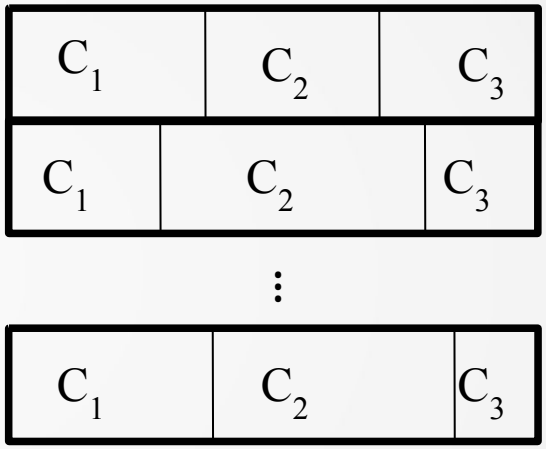
- Tamanho variável
  - Alguns campos podem ter tamanhos variáveis em diferentes registros.

Formato Registro

Tamanho fixo e campos de tamanho fixo

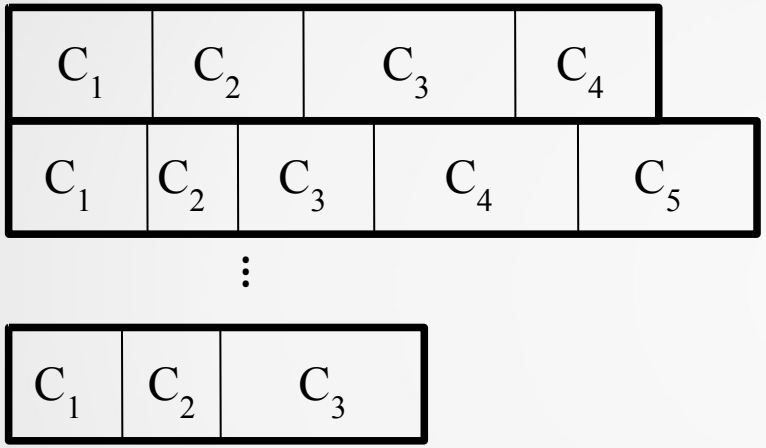


Tamanho fixo e campos de tamanho variável

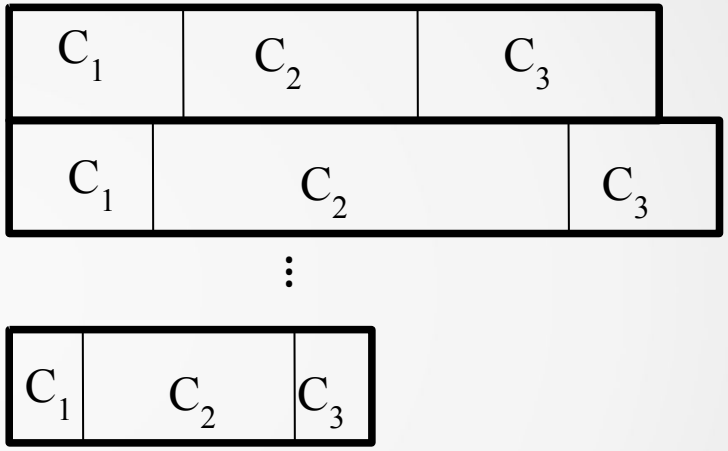


Formato Registro

Tamanho variável e número de campos variável

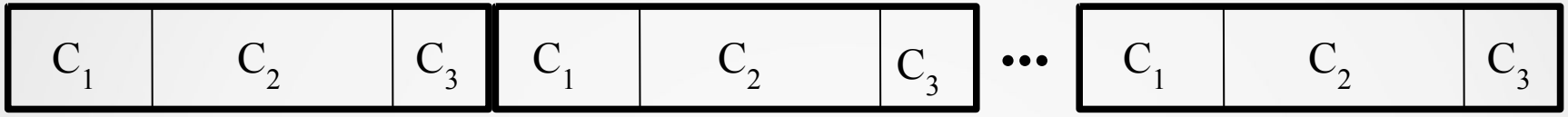


Tamanho variável e número de campos fixo de tamanho variável



Localização Registro

Tamanho fixo

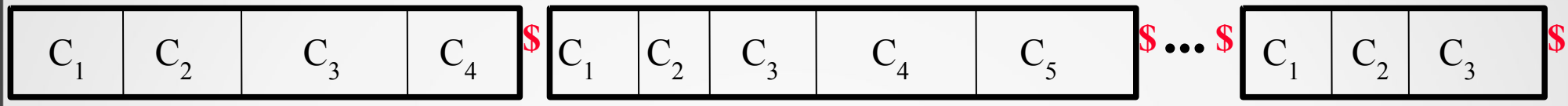


L: endereço inicial **T: Tamanho Registro** **P: Posição Registro**

Localizar um Registro L+(T*P)

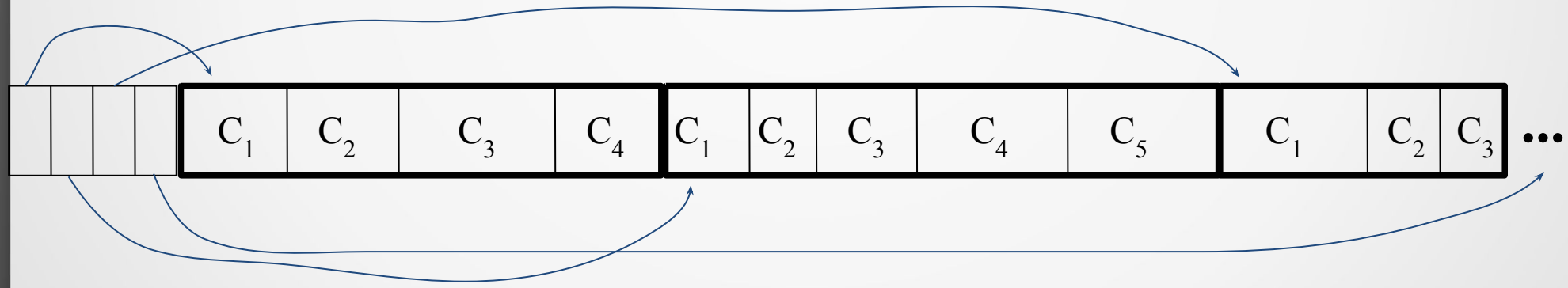
Localização Registro

Tamanho variável e número de campos variável



Registro tem uma marca de fim $

ou



Cabeçalho com ponteiros para o início de cada registro

Localização Campos

- Mesma estratégia para a localização de registros:
  - Marca de fim de campo
  - Apontadores para os campos

Representação Dados

- Quais itens que queremos armazenar?
  - Um valor de salário
  - Um nome
  - Uma data
  - Uma foto
- O que temos disponível para armazenar tais itens?
  - Bytes :(

Representação Dados

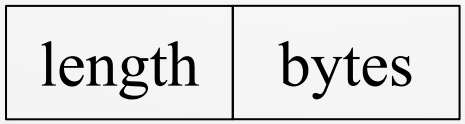
- Inteiro (short): 2 bytes
  - 36 é 00000000 00100100
- Real, ponto flutuante
  - n bytes para a mantissa e m para o expoente
- Carácter (várias opções) - ASCII
  - A : 10000001
  - a: 11000001

Representação Dados

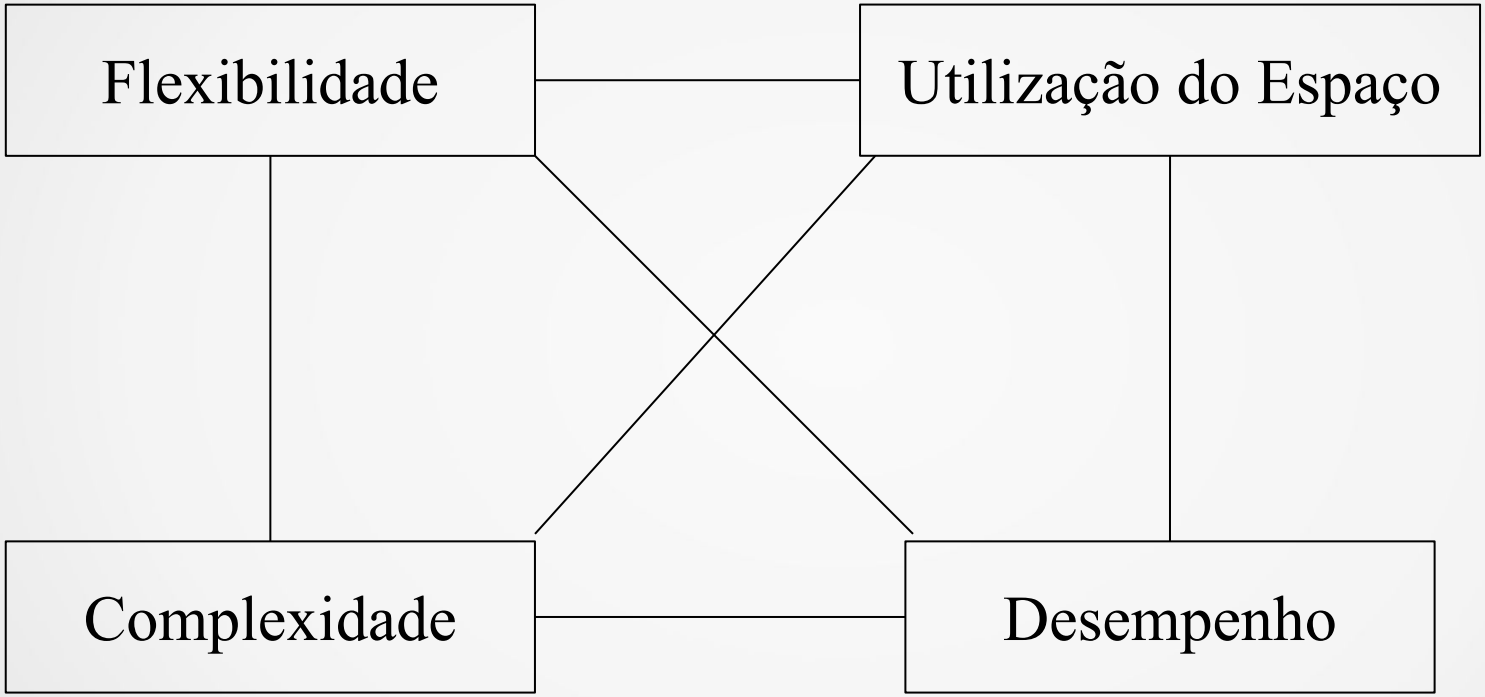
- Datas
  - Inteiro: número de dias desde 1/1/1900
  - 8 caracteres: YYYYMMDD
  - 7 caracteres: YYYYDDD
- Hora
  - Inteiro: segundos desde a meia-noite (com data)
  - Caracteres: HHMMSS

Representação Dados

- Sequência de caracteres
  - Marca de fim: L I V R O \0
  - Prefixado pelo tamanho: 5 L I V R O
  - Tamanho fixo (conhecido pelo programador)
- Coleção de bits
  - 1 (ou +) byte(s) para a quantidade de bytes (length), e os bytes



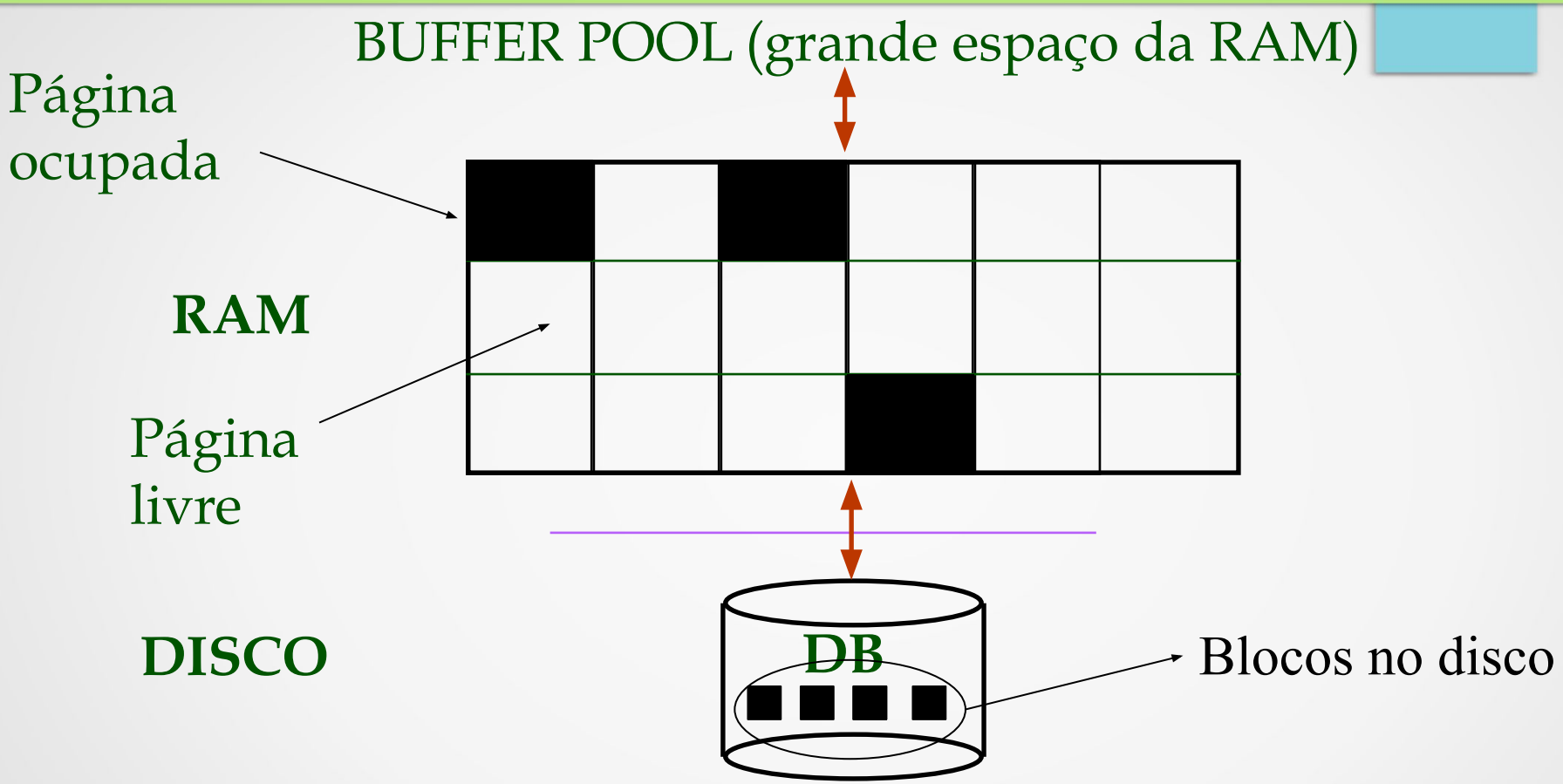
Conclusões (Rep. Dados)



Conclusões (Rep. Dados)

- Escolha de uma estratégia
  - Espaço utilizado para os dados esperados
  - Tempo esperado para
    - Buscar um dado baseado em uma chave
    - Buscar o próximo dado de uma chave
    - Inserir um registro
    - Adicionar um registro
    - Excluir um registro
    - Atualizar um registro
    - Ler todo o arquivo
    - Reorganizar o arquivo

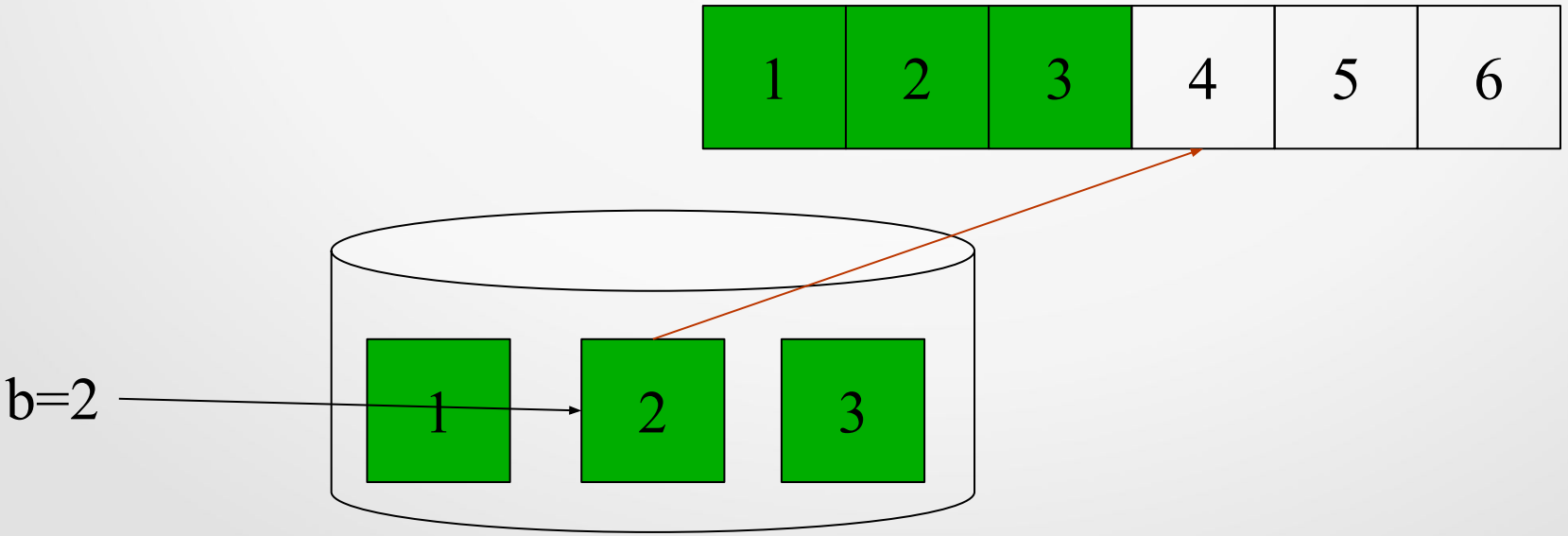
Buffer



- Dados devem estar na RAM para o SGBD trabalhar.
- Deve existir um forma de acessar os dados de forma transparente para as aplicações
- BUFFER MANAGER

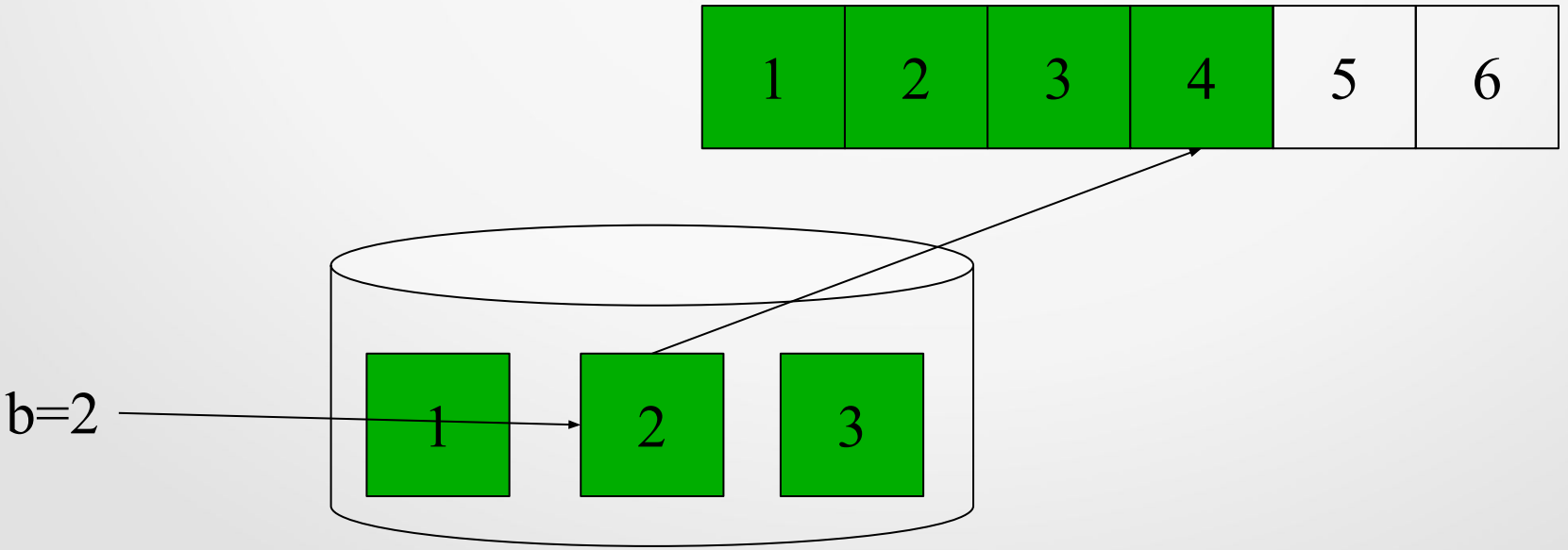
Procura de uma Página

- Operação de procura
  - Possui como entrada o número do bloco *b* do disco
  - Retorna a página *p* do buffer que corresponde ao bloco *b* do disco



Procura de uma Página

- Operação de procura
  - Possui como entrada o número do bloco *b* do disco
  - Retorna a página *p* do buffer que corresponde ao bloco *b* do disco



Procura de uma Página

- Algoritmo
  - Se o bloco **b** já está no buffer **p**, retorna **p** (economiza-se um acesso ao disco).
  - Se **b** não está no buffer, é necessário lê-lo do disco. Procura uma **p** livre, se encontrar coloque **b** na posição **p**.
  - Senão, é necessário liberar uma página **p** do buffer para colocar **b** em **p**.
    - Procura uma **p** para ser substituída (política de troca)
    - Se **p** encontrada foi modificada, reescrever **p** no disco (gerenciador de transação)
  - Transferir **b** para **p** e retornar **p** ao solicitante

Página Solicitada

- O BM possui uma tabela contendo

  <#page, #bloco, pin_count, dirty_bit>

  #page: número da página no buffer
  #bloco: número do bloco que ocupa a página
  pin_count: contador de uso da página
  dirty_bit: bit que indica se os dados da página foram modificados

Página Solicitada

- Funcionamento *pin_count* e *dirty_bit*
  - *pin_count++* é realizado a cada acesso à página e *pin_count--* quando deixa de utilizar.
  - Se *pin_count == 0* então a página pode ser liberada
  - Se um processo altera os dados de uma página, *dirty_bit* é ligado.
  - Se uma página deve ser liberada e o *dirty_bit* está ligado, a página deve ser devolvida ao disco (ao bloco correspondente)

Políticas para Troca

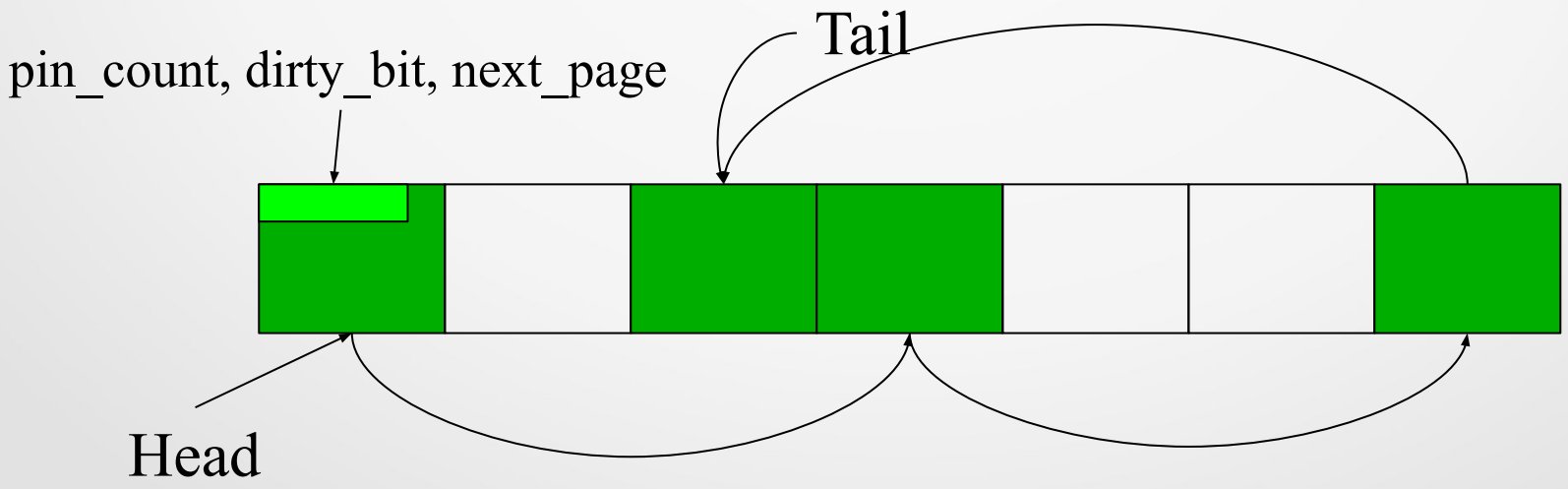
- A maioria dos SO troca o bloco de uso mais antigo (estratégia LRU – Least Recently Used)
- Ideia por trás da LRU – utilizar as páginas ocupadas mais antigas
- Frequentemente, as consultas têm padrões de acesso bem definidos (ex., varredura sequencial) e essa estratégia pode ser custosa

Políticas para Troca

- Estratégia Toss-Immediate: libera a página imediatamente após o uso
- O último bloco utilizado (MRU - Most Recently Used): a página a ser liberada é a última que foi utilizada

Políticas de Troca

- Controles do buffer
  - Bloco *Pinned* (fixado): um bloco que não pode ser reescrito de volta no disco pois está sendo acessado. Se *pin_count==0* bloco pode ser movido
  - *Dirty_bit* (bit sujo): bloco deve ser reescrito no disco caso *dirty_bit == 1*.

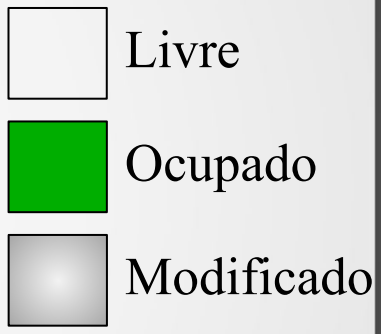


Políticas para Troca

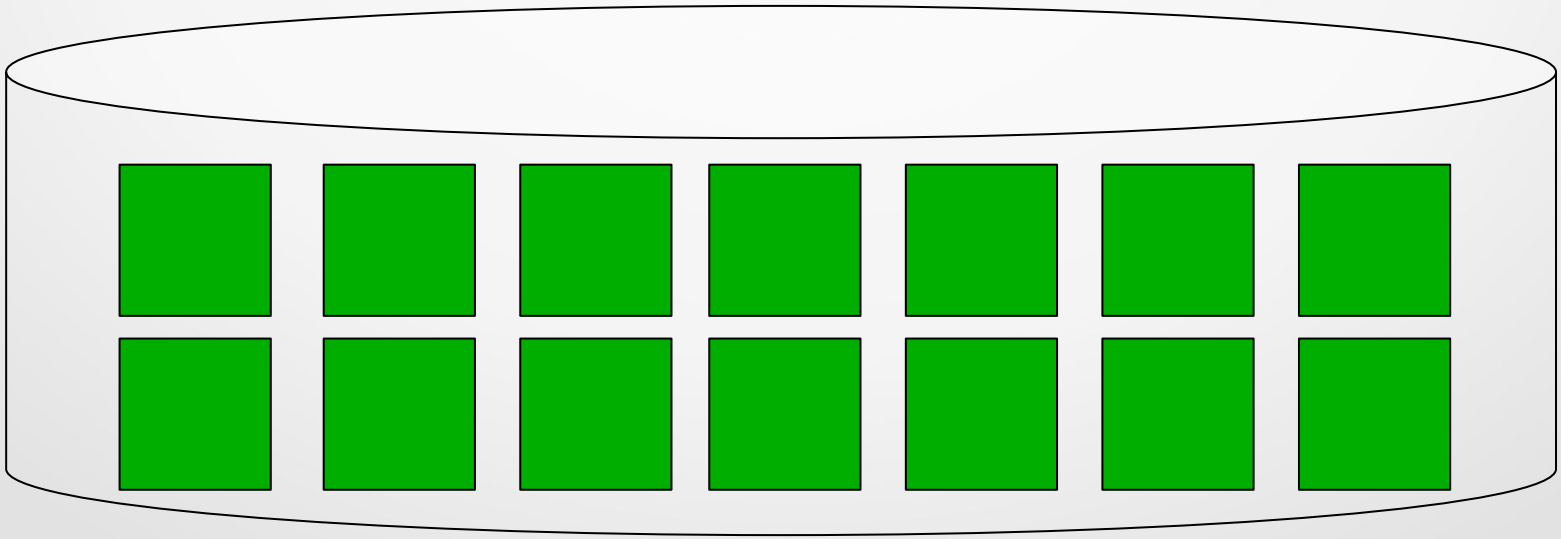
- O gerenciador de buffer pode utilizar dados estatísticos para prever qual objeto será acessado para otimizar a alocação de buffer
- Suporta a gravação forçada de bloco para o caso de recuperação após falha

Gerenciador de Buffer

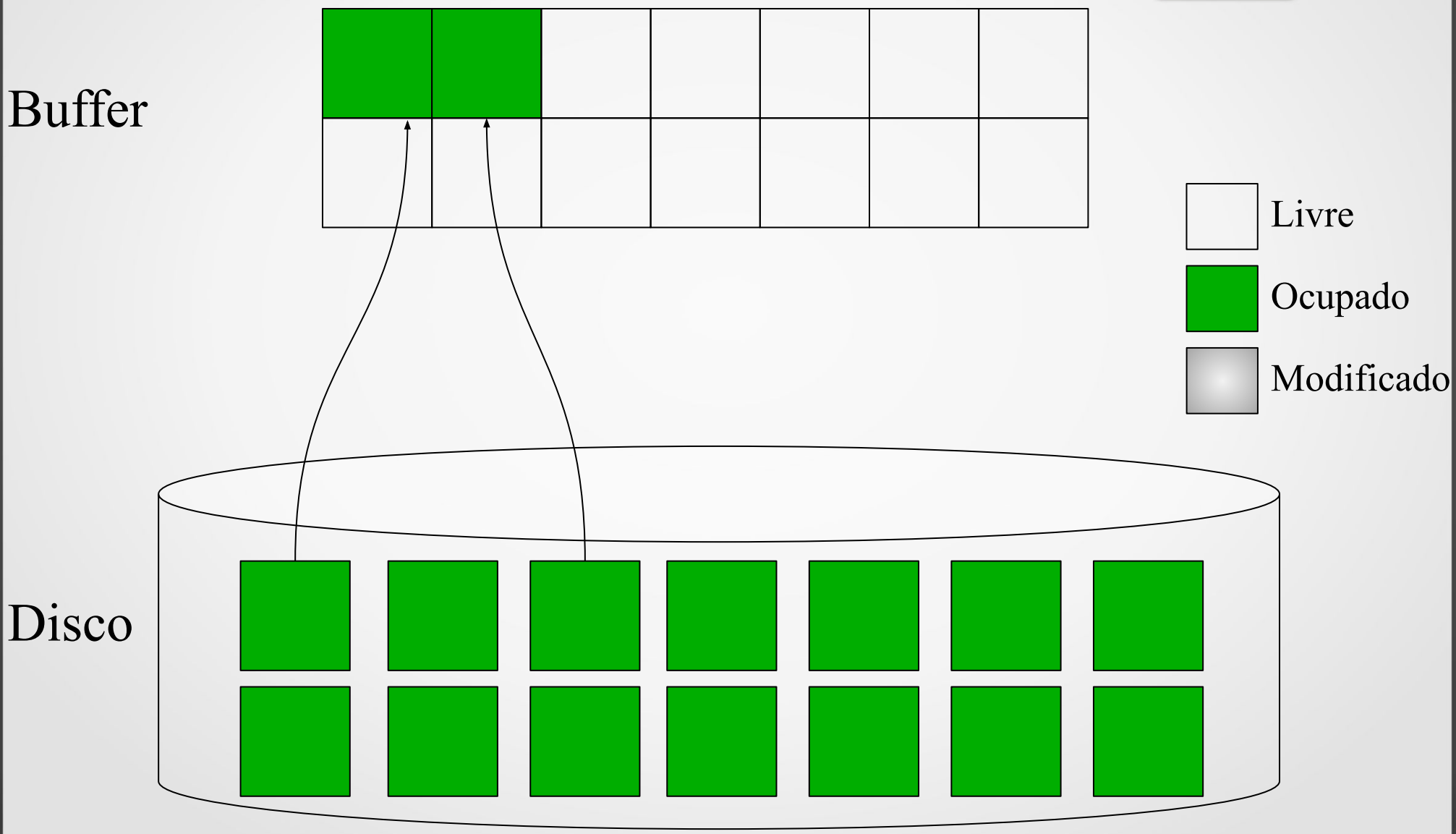
Memória
(buffer pool)



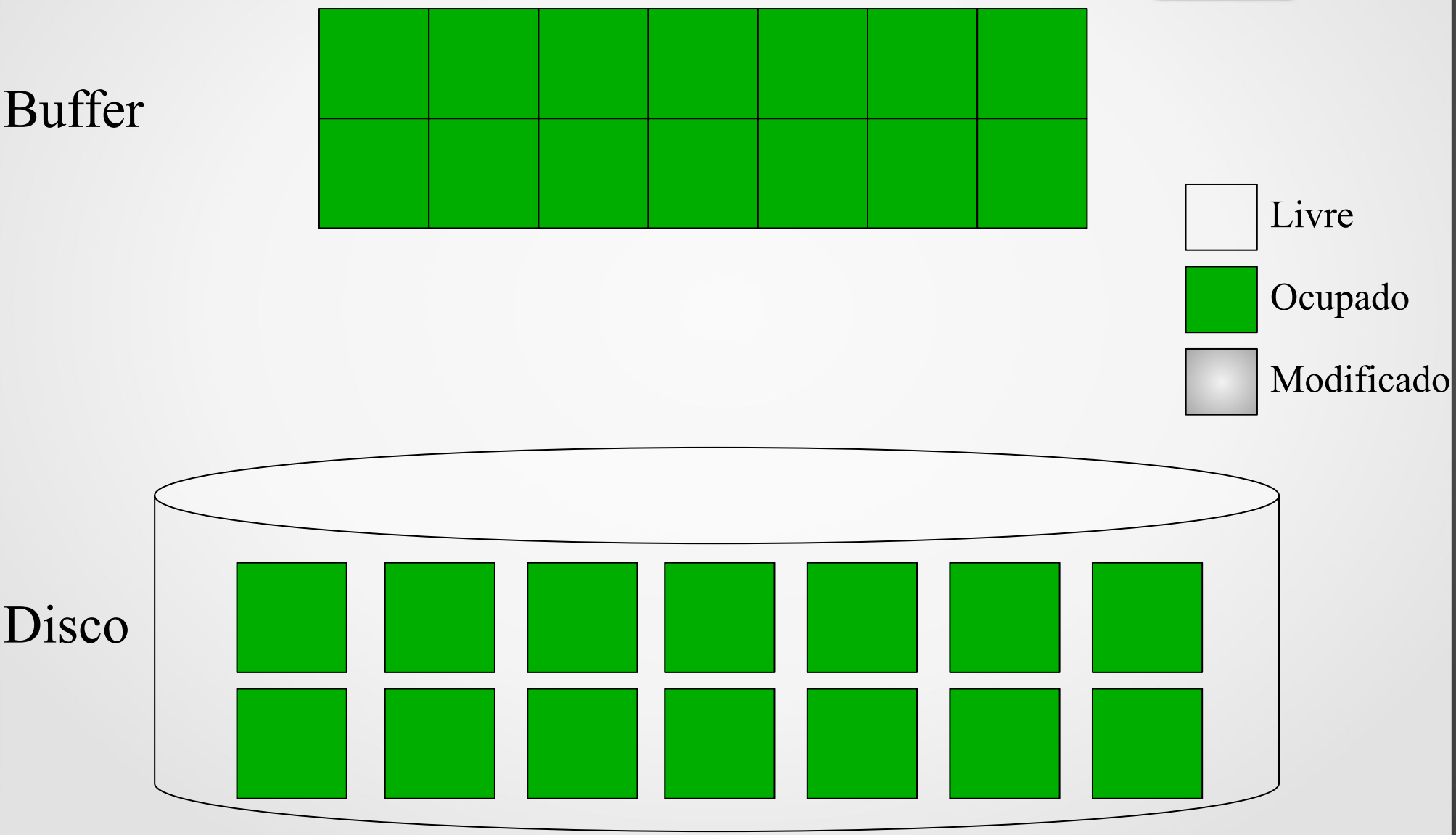
Disco



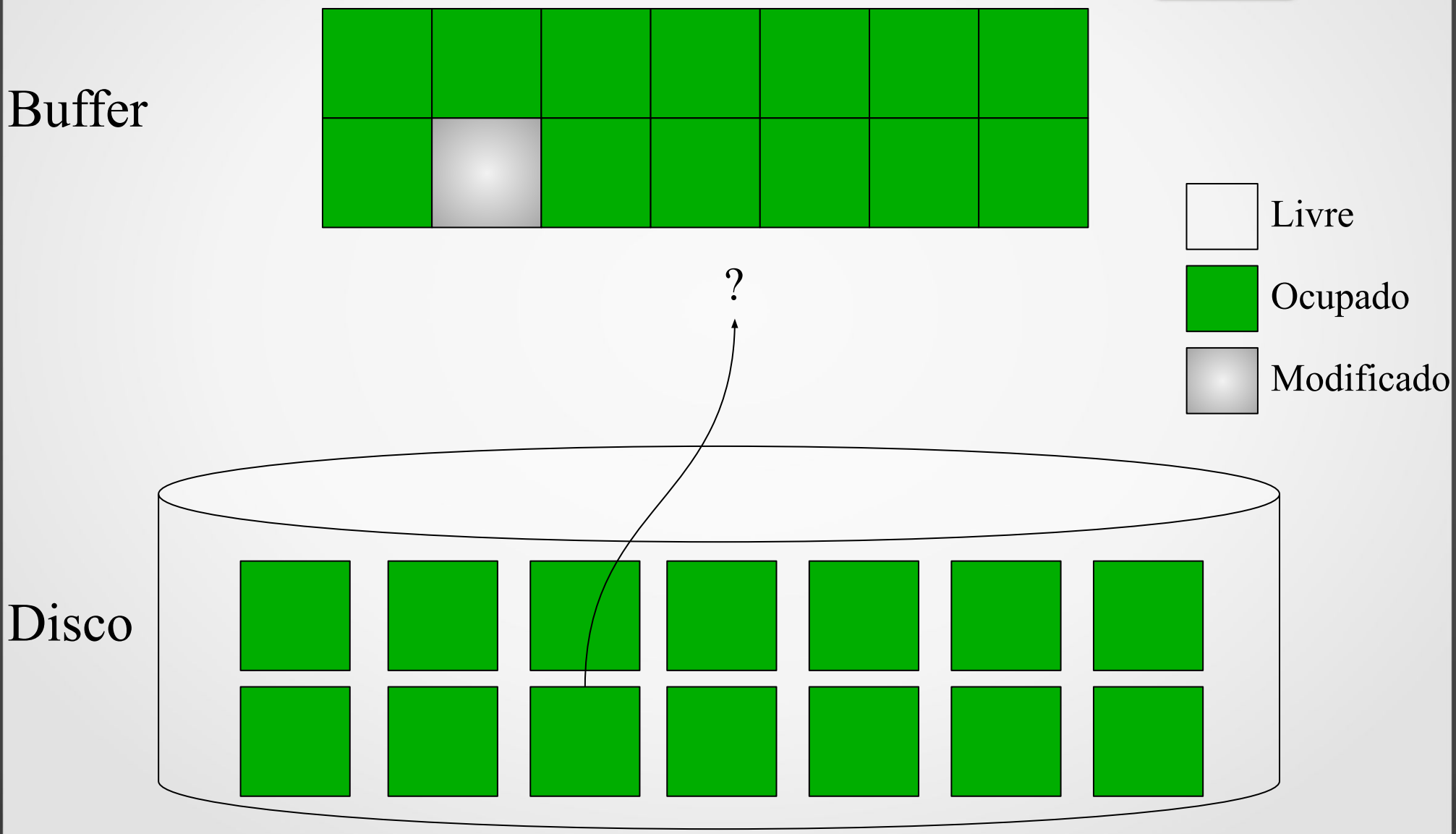
Gerenciador de Buffer



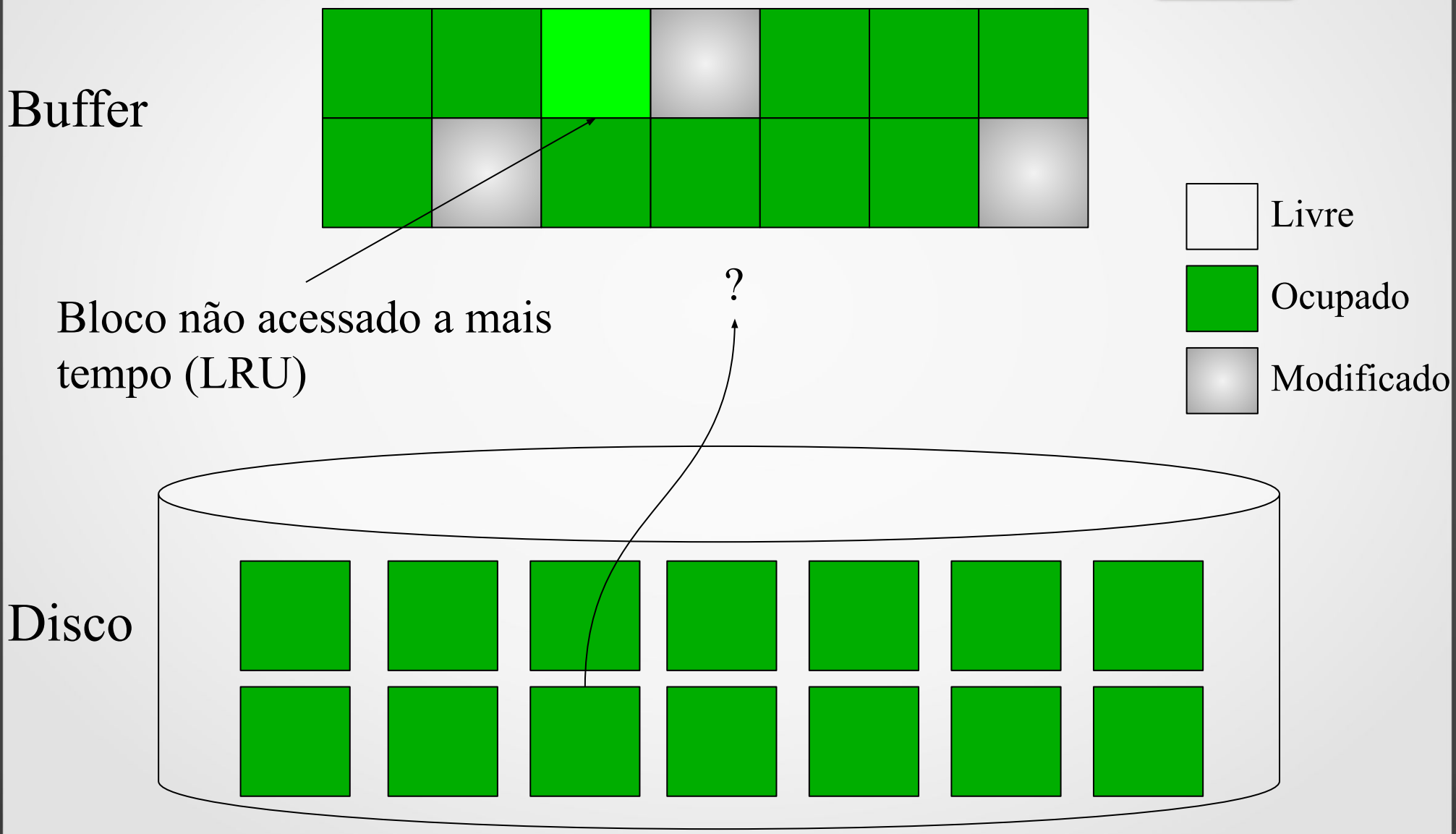
Gerenciador de Buffer



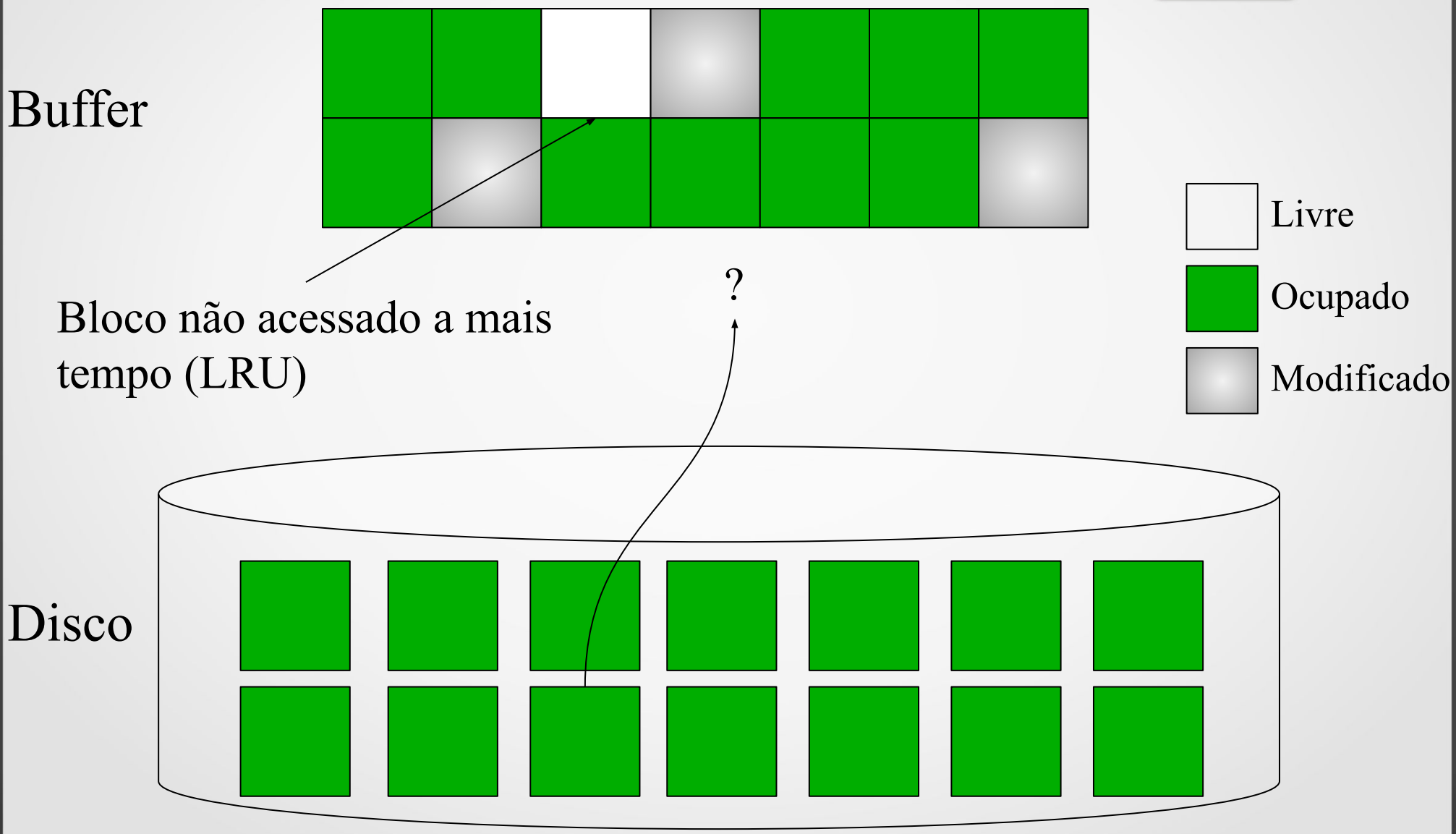
Gerenciador de Buffer



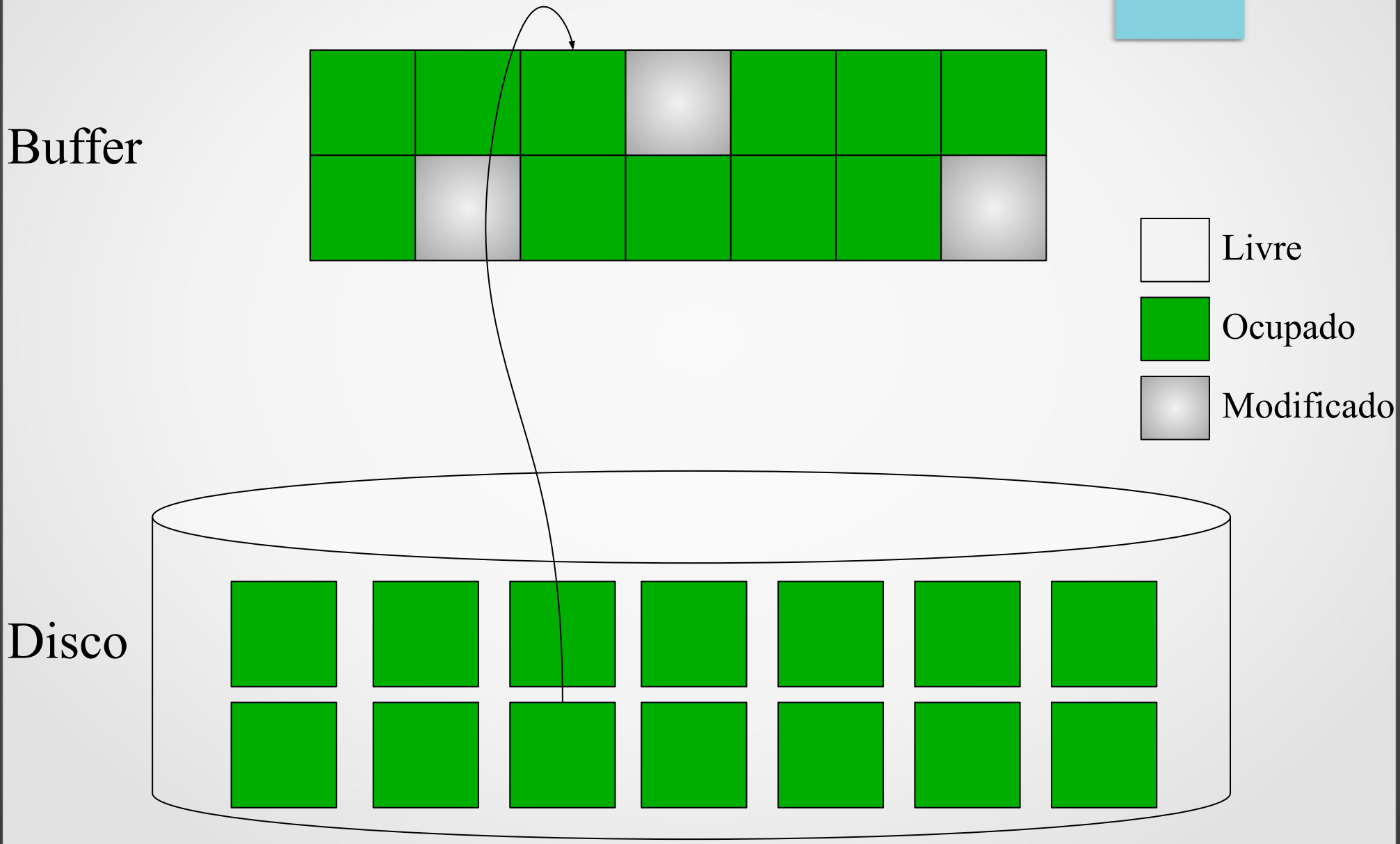
Gerenciador de Buffer



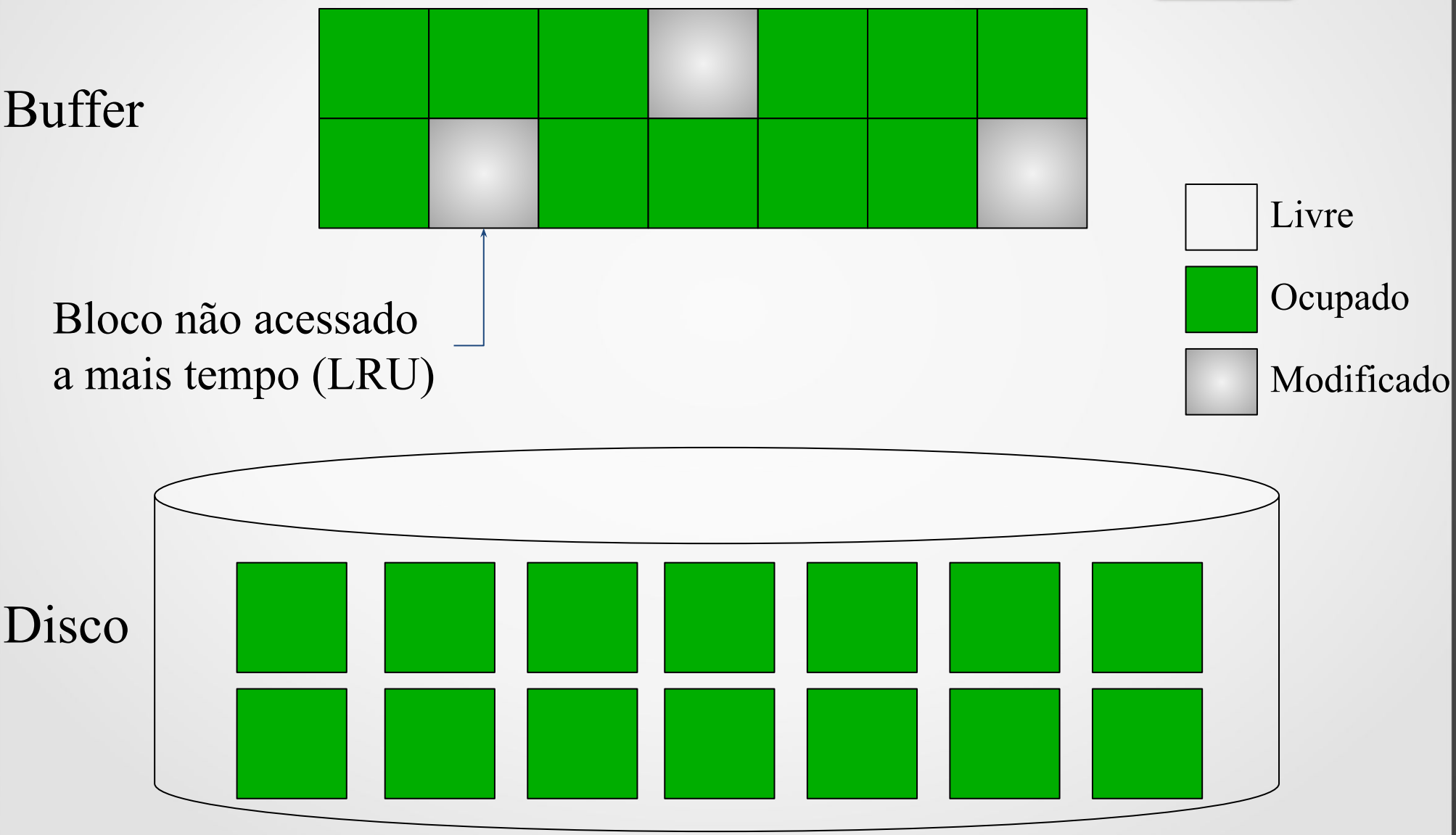
Gerenciador de Buffer



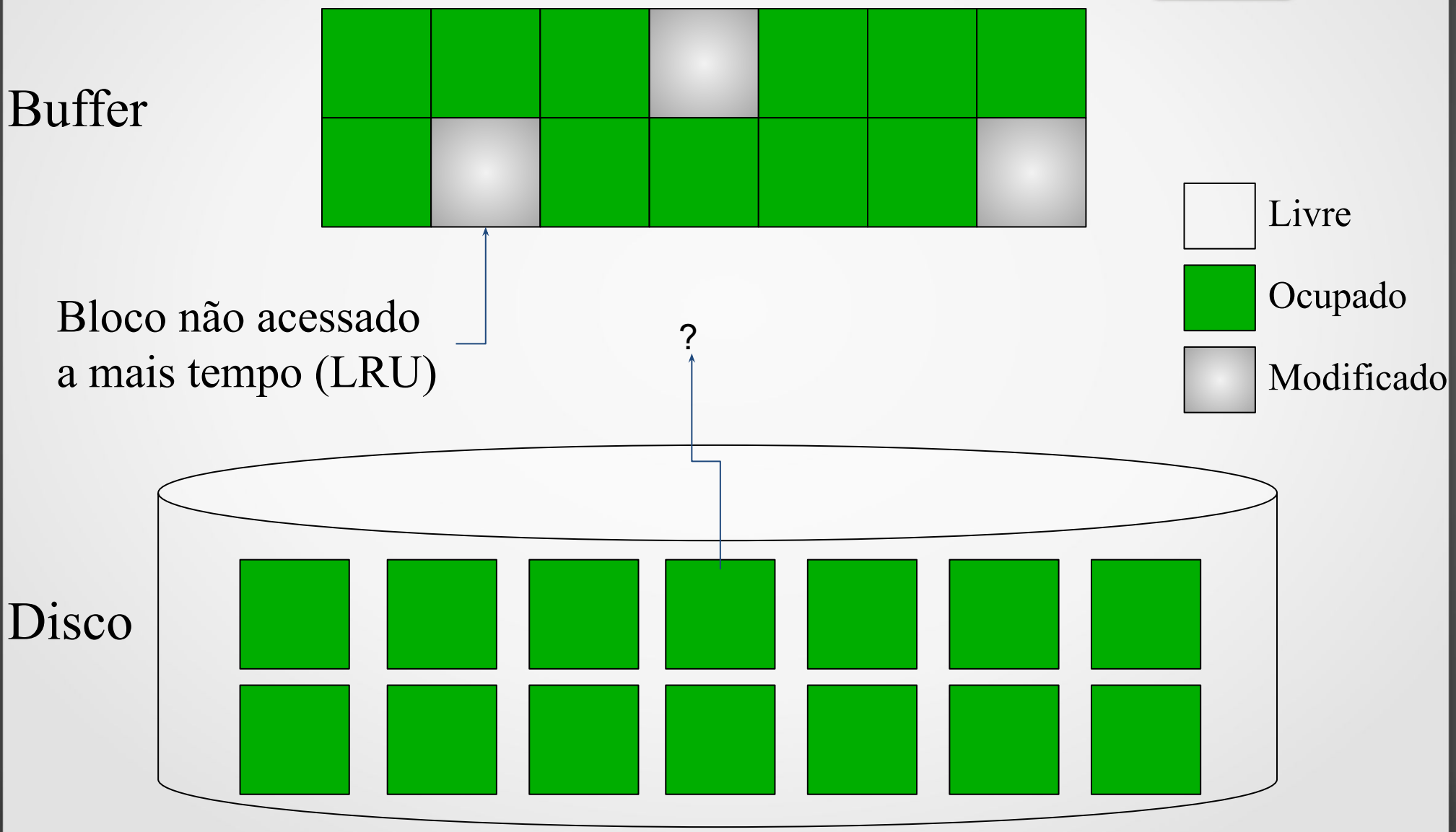
Gerenciador de Buffer



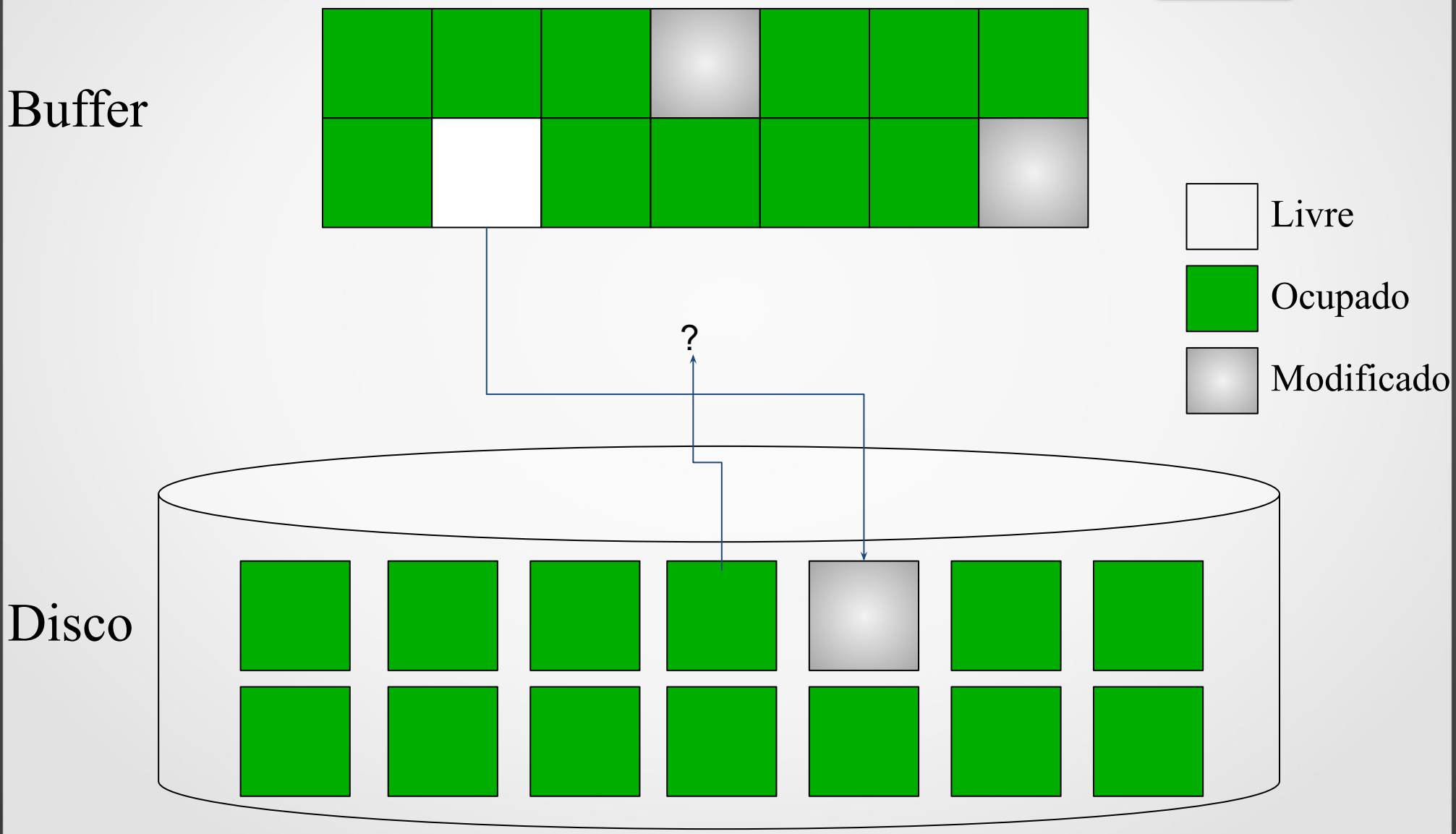
Gerenciador de Buffer



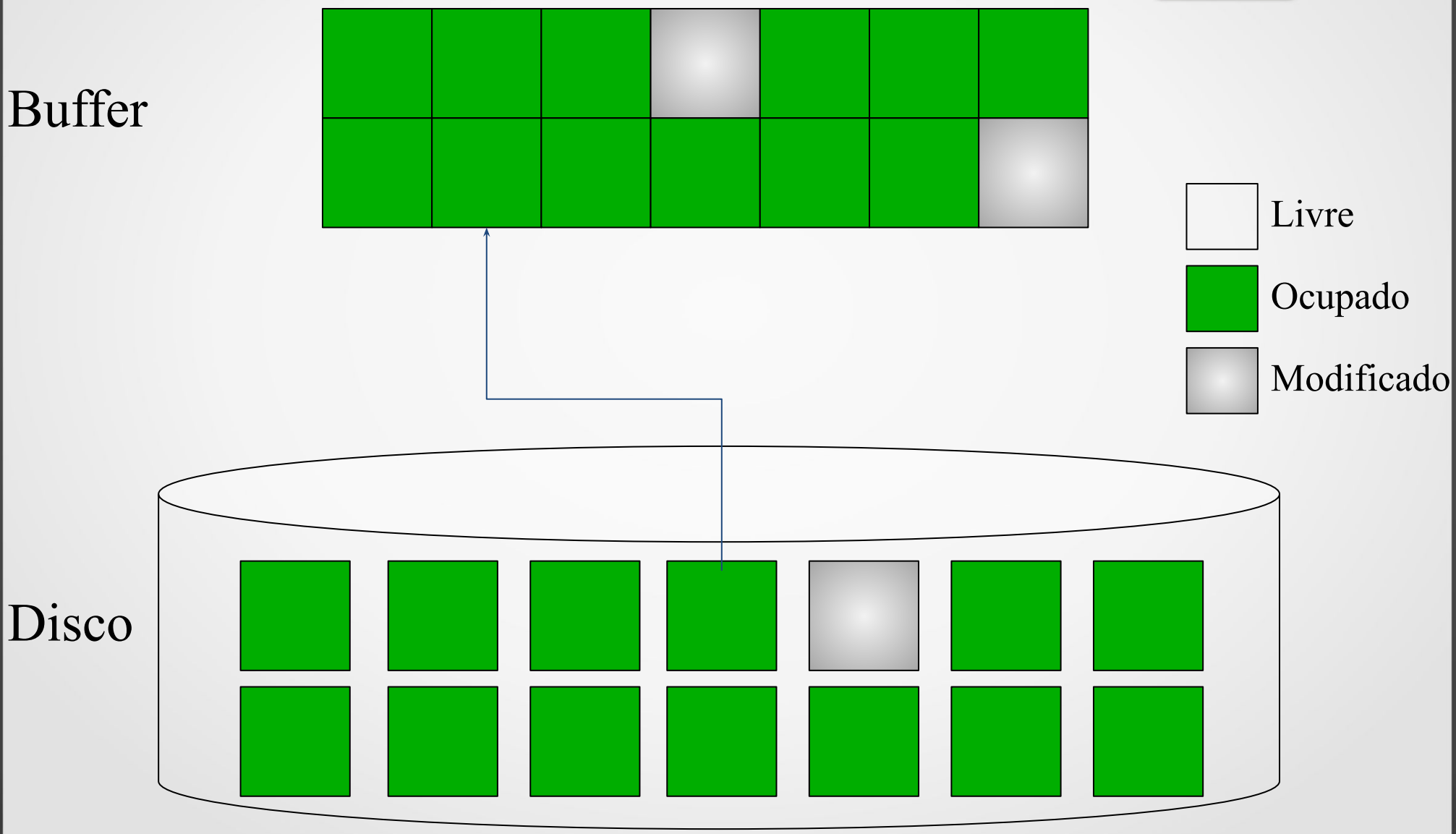
Gerenciador de Buffer



Gerenciador de Buffer

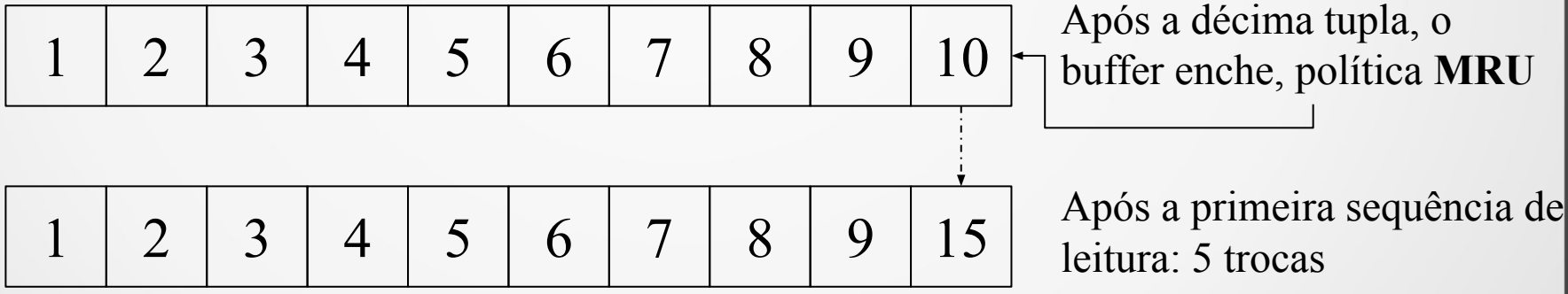


Gerenciador de Buffer



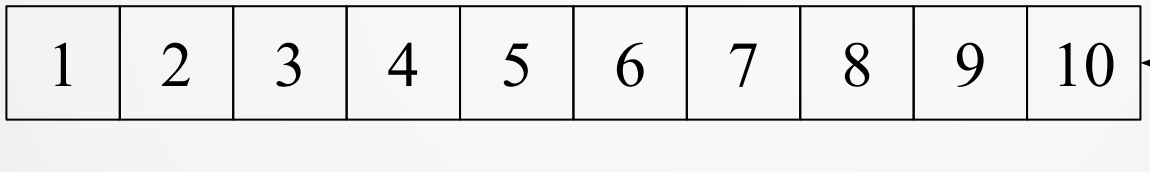
Políticas para Troca

- Suponhamos uma leitura que recupere duas vezes as tuplas 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 e 15 (e um buffer com 10 páginas e comporta uma tupla por página - inicialmente vazio)

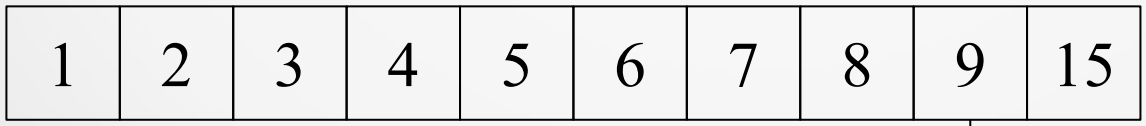


Políticas para Troca

- Suponhamos uma leitura que recupere duas vezes as tuplas 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 e 15 (e um buffer com 10 páginas e comporta uma tupla por página)

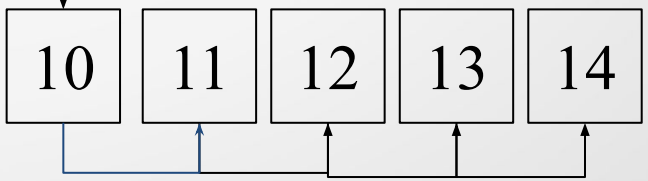


Após a décima tupla, o buffer enche, política **MRU**



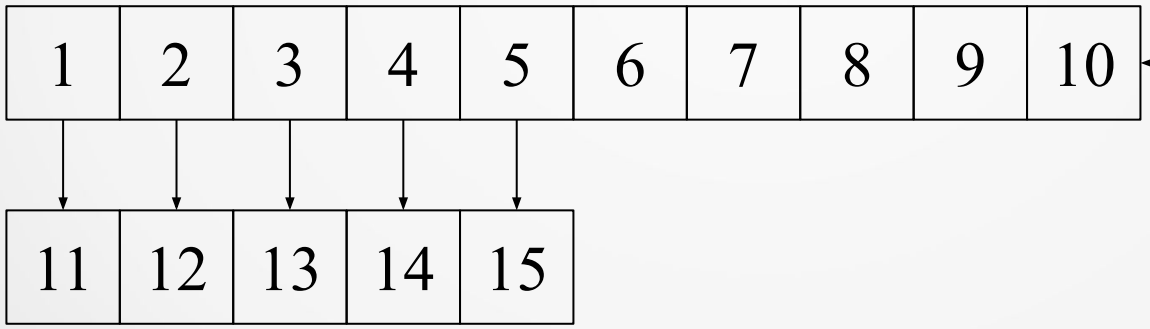
Após a primeira sequência de leitura: 5 trocas

Iniciando a segunda leitura, as tuplas de 1 a 9 estão no buffer, as trocas ocorrem na página com a tupla 9.
Mais 5 trocas.
Total **10** trocas.



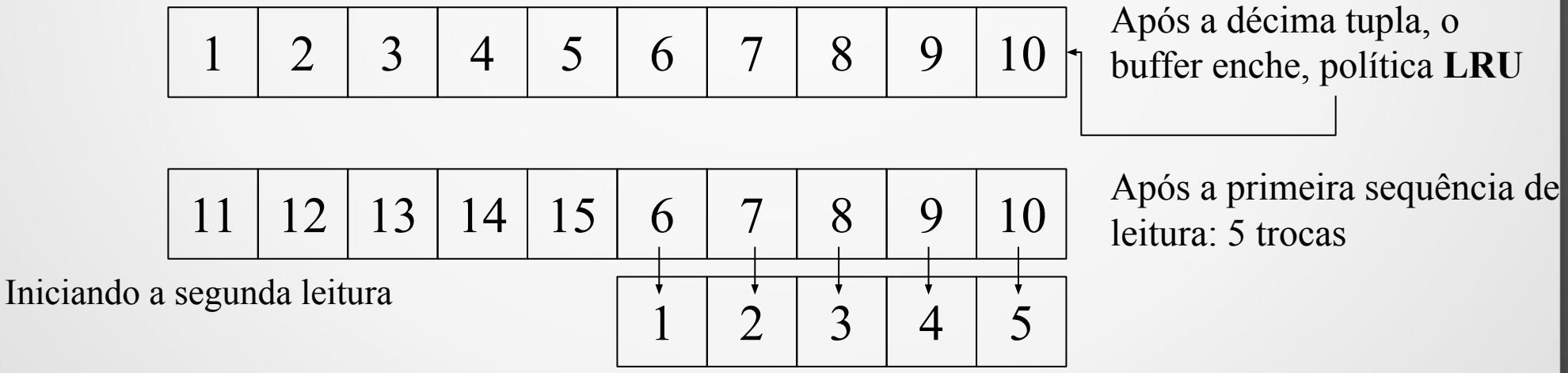
Políticas para Troca

- Suponhamos uma leitura que recupere duas vezes as tuplas 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 e 15 (e um buffer com 10 páginas e comporta uma tupla por página)



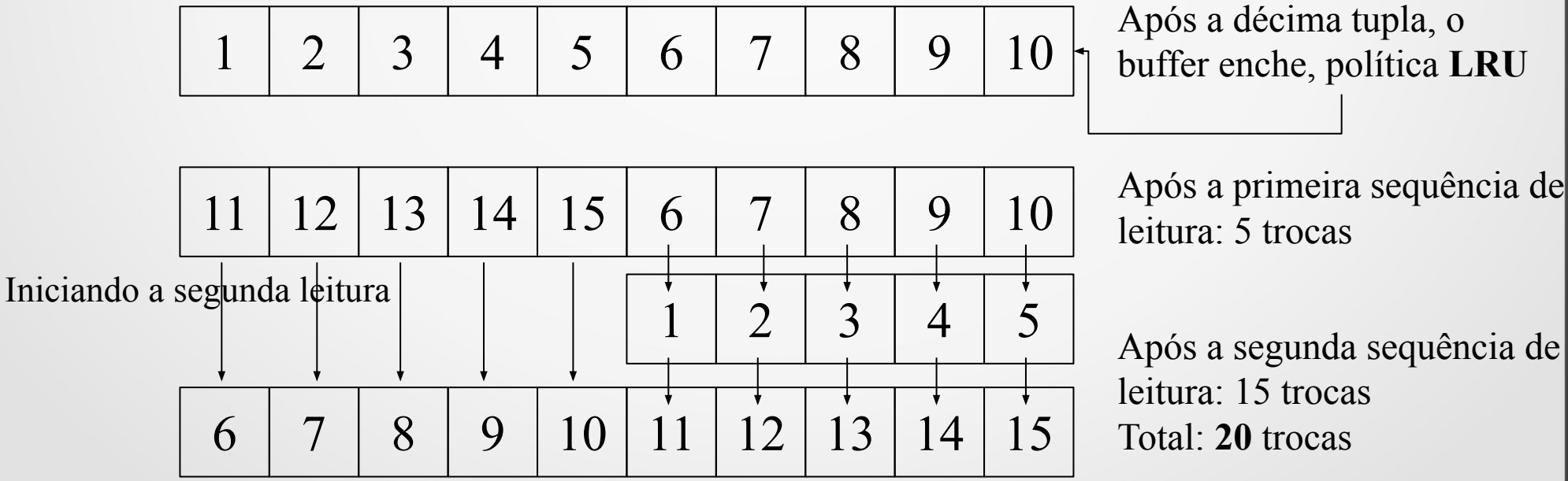
Políticas para Troca

- Suponhamos uma leitura que recupere duas vezes as tuplas 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 e 15 (e um buffer com 10 páginas e comporta uma tupla por página)



Políticas para Troca

- Suponhamos uma leitura que recupere duas vezes as tuplas 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 e 15 (e um buffer com 10 páginas e comporta uma tupla por página)



Políticas para Troca

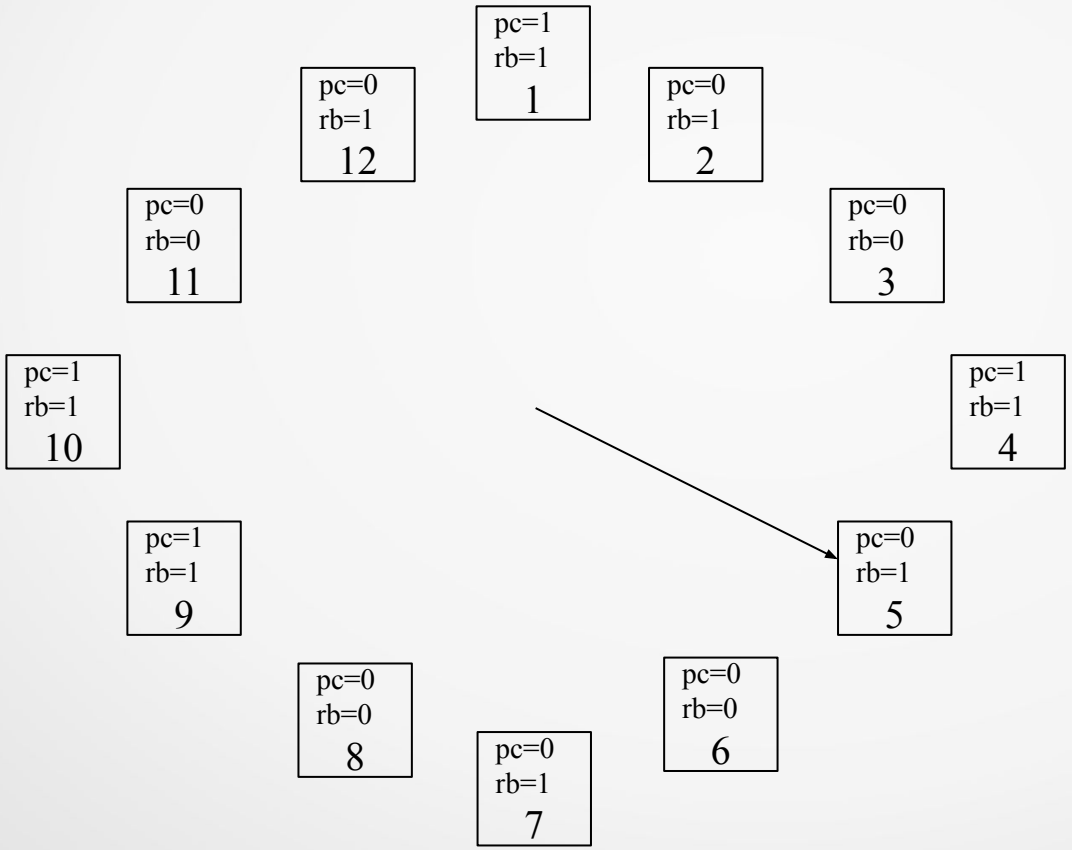
- A LRU é ainda a mais utilizada pelos SGBD, mas conforme visto no exemplo, para scan repetidos muitas páginas são trocadas.
- Uma alternativa é o algoritmo do relógio (clock algorithm)
  - uma variação do LRU mas dá uma segunda chance a página
  - Além do *pin count* é acrescentado o *referenced bit*
  - Funcionamento similar ao LRU mas ao escolher uma página, verifica o *referenced bit*, se 1 (valor inicial) se torna 0, se for 0, troca.

Políticas para Troca

- Clock algorithm (funcionamento):
  - As páginas do buffer pool são organizadas logicamente em um círculo
  - Existe um ponteiro apontando para uma das páginas e que se move no sentido horário
  - Se a página apontada possui pin_count==0 e referenced_bit==1, faz referenced_bit=0 e move para a próxima página
  - Se pin_count==0 e referenced_bit==0, página candidata a substituição

Políticas para Troca

- Clock algorithm (funcionamento):



Neste cenário:
- A página 5 terá rb ← 0
- O ponteiro move para 6 e a remove (rb==0 e pc==0)
- Os movimentos restantes fariam:
  - Página 7, rb=0
  - Página 8, escolhida
  - Página 9, nada
  - Página 10, nada
  - Página 11, escolhida
  - Página 12, rb=0
  - :

Políticas para Troca

- Clock algorithm:

```
do for each page in cycle
{
  if (pincount == 0 && refbit== 1)
    refbit==0;
  else
    if (pincount == 0 && refbit==0)
          choose this page for replacement;
} until a page is chosen;
```