Bancos de Dados I
Organização de Arquivos

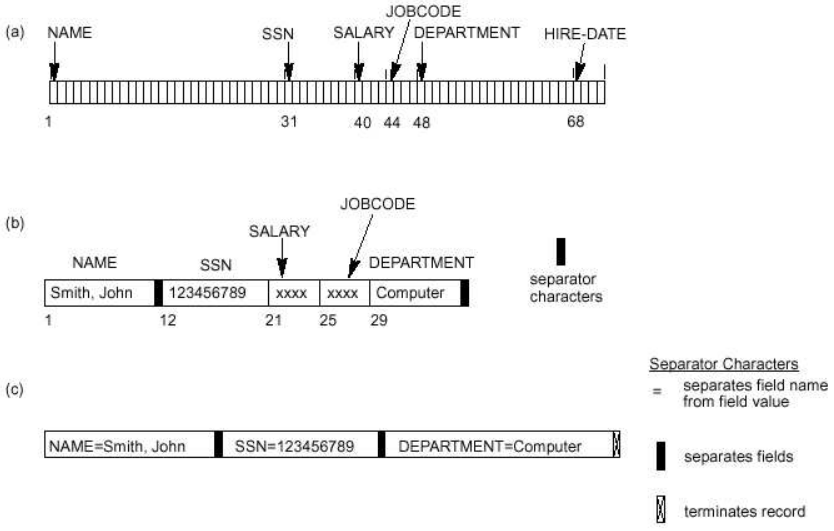
Prof. Altigran Soares da Silva
IComp/UFAM

V2018.1

Arquivos

- Arquivos: Sequências de registros
- Registros
  - Tamanho fixo: Todos os registros possuem o mesmo tamanho exatamente;
  - Tamanho variável: Um ou mais campos têm tamanho variável
    - Campos com múltiplos valores (campos repetidos);
    - Campos opcionais;
    - Registros de tipos diferentes;

Registros: Fixo x Variável



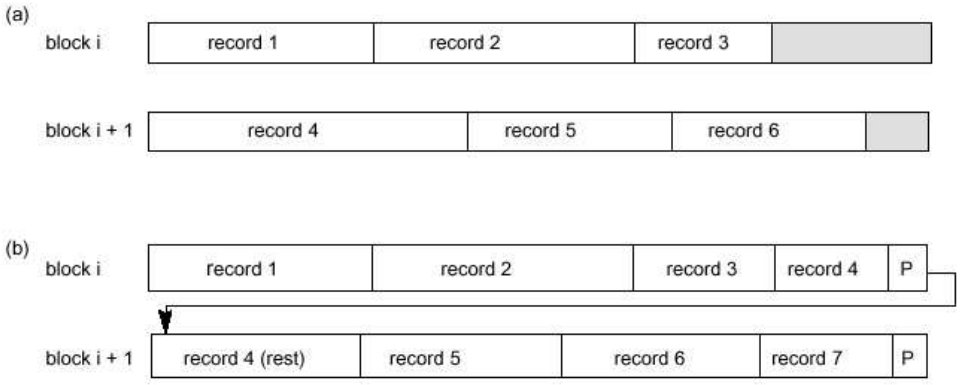
Arquivos

- Sistema Operacional/Sistema de Arquivos:
  - Fornece ao SGBD os serviços básicos par manipulação de arquivos.
  - O SGBD utiliza este serviços para prover outras facilidades (nível lógico).
- Registros:
  - Campos ou itens;
  - Tipo/Formato de Registros;
  - Tipo de dados de cada campo;

Arquivos e Blocos

- Blocos: Unidades de transferência da memória secundária para memória principal;
- Alocação de registros de um arquivo são feitas em blocos do disco;
- Modos de alocação: registros nem sempre cabem perfeitamente em um bloco;
- Fator de Bloco
- Alocação espalhada vs. Alocação não espalhada

Blocos: Espalhada vs Não-Espalhada



Organização de Arquivos

- Registros Não-Ordenados
- Registros Ordenados
- Organização por Hashing

Arquivos Não-Ordenados

- Chamados de Heap Files
- Novos registros inseridos sempre no final do arquivo.
  - Portanto é bem eficiente
- A busca por registros é linear.
  - Exige a leitura da metade dos blocos em média.
  - Alto custo.
- Remoção lógica
  - marca de remoção; reorganização periódica

Arquivos Não-Ordenados

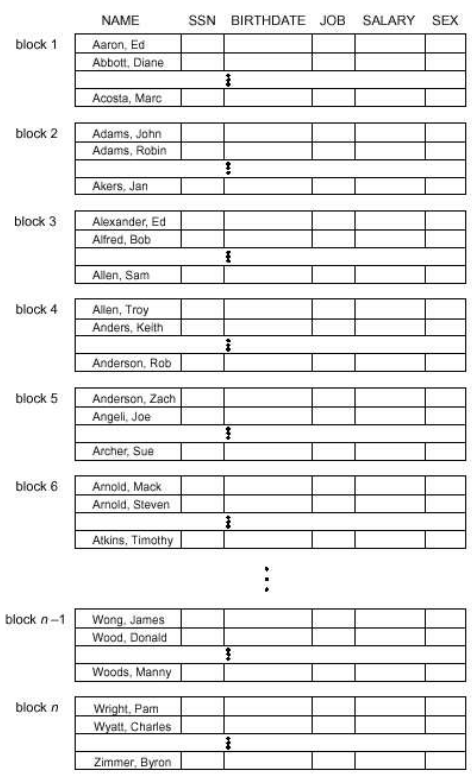
- A leitura dos registros em uma ordem em particular exige reordenação dos registros
- Se o número de registros é grande, a ordenação é em memória secundária (ordenação externa).

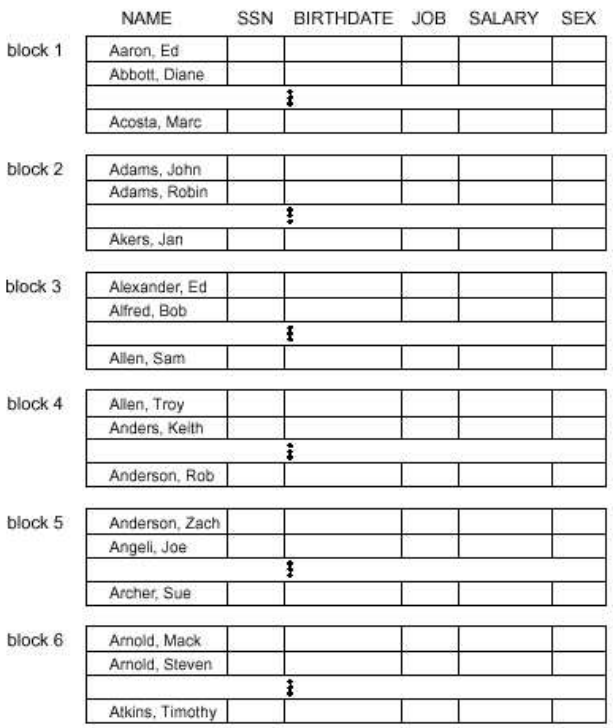
Arquivos Não-Ordenados

- Arquivo Relativo :
  - Registros de tamanho fixo;
  - Alocação não espalhada e contígua;
  - Fácil localização de registros;

Arquivos Ordenados

- Chamados Arquivos Sequenciais.
- São mantidos ordenados por um *campo de ordenação.*
- Se o campo de ordenação é uma chave:
  - Chave de Ordenação





Arquivos Ordenados (2)

- Busca binária pode ser usada para buscar registros pelo campo de ordenação.
  - Isso requer a carga de $\log_2$ blocos
  - Busca por outro campos é linear
- Varredura (scan) dos registros na ordem do campo de ordenação é bastante eficiente.

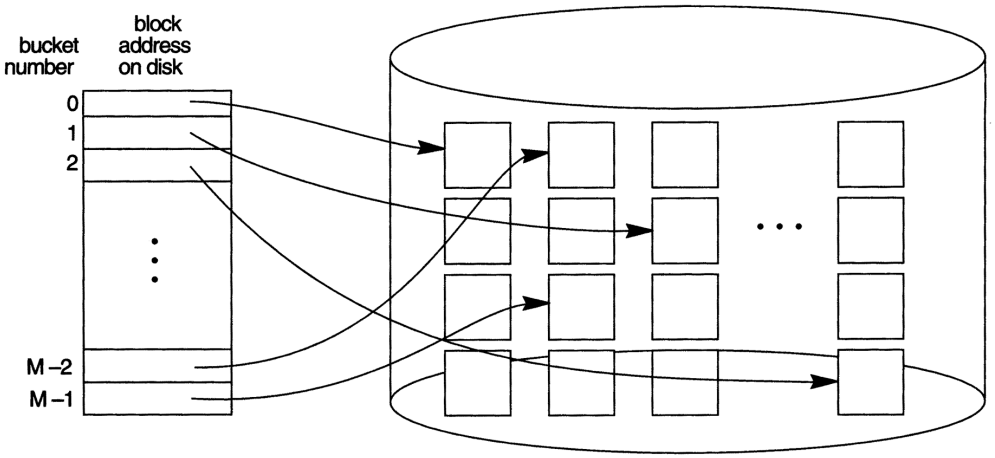
Arquivos Ordenados (3)

- Inserção tem alto custo.
  - Os registros devem ser inseridos na ordem correta.
  - É comum manter um arquivo auxiliar – chamado de overflow ou de transação – somente para receber novas inserções. De tempos em tempos o arquivo principal é reconstruído a partir dele.
  - Outra solução é deixar espaço livre nos blocos para futuras inserções
- Remoção: lógica com reconciliação física periódica. O arquivo é reconstruído.

Arquivos Hashing

- Os blocos do arquivo são divididos em M buckets de igual tamanho $b_0,b_1,b_2,\ldots,b_{M-1}$
  - Tipicamente: um bucket = B blocos
- Um dos campos é usado como o argumento da função hash – campo de hash
- O registro com chave K é armazenado no bucket i=h(K), onde h é uma função de hashing

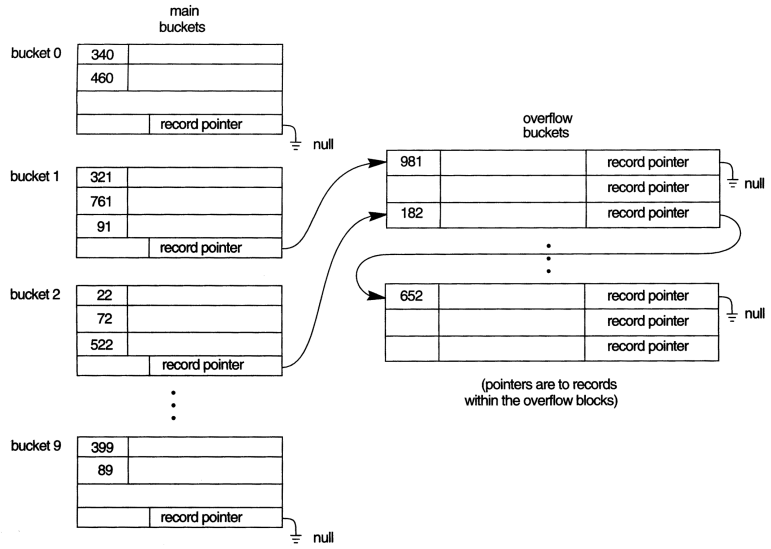
Arquivos Hashing



Colisões

- Colisões ocorrem quando um novo registro deve ser colocado em um bucket que já esteja cheio.
  - Pode-se usar um arquivo de overflow
  - Registros que geram o mesmo código de hash podem ser encadeados neste arquivo.

Colisões



Tratando Overflow

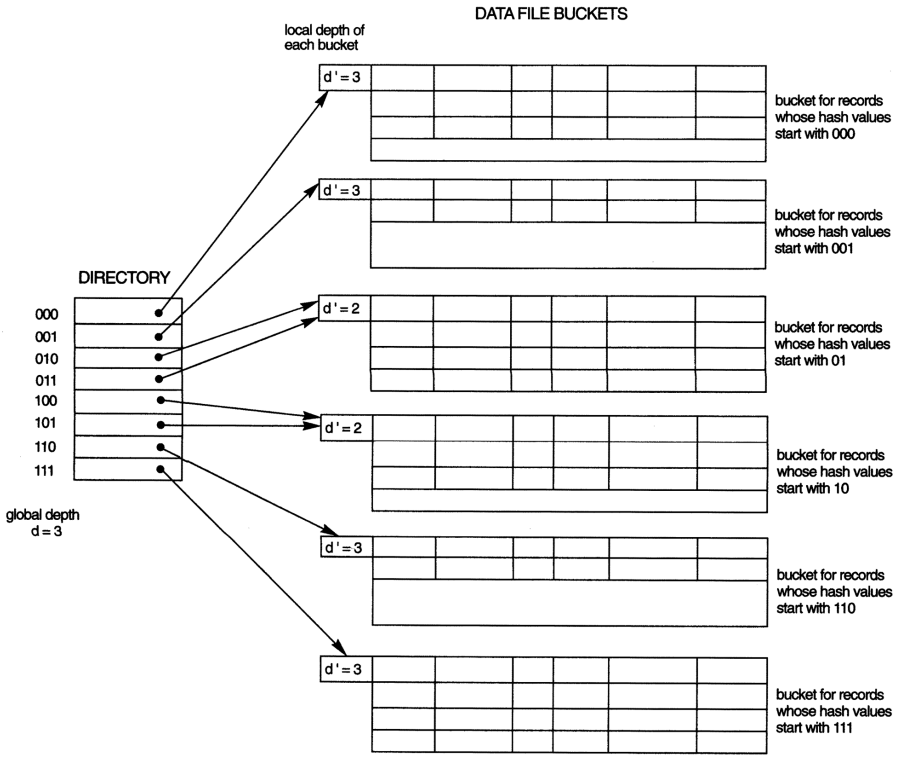
- Para reduzir a ocorrência de overflow deve-se manter os blocos 70-80% ocupados.
- A função de hash h deve distribuir os registros de forma uniforme entre os buckets
- Principais desvantagens:
  - Há um número fixo de blocos que podem ser alocados no arquivo de dados.
  - Um número grande de registros de overflow degrada a performance
  - Acessos ordenados pela chave de hash são muito ineficientes

Operações

- Busca usando o campo de hash é muito eficiente. Número de blocos lidos é O(1)
- Inserção é também muito eficiente, principalmente se não houver overflow
- Busca baseada em outro campo exige busca sequencial no arquivo
- Varredura ordenada é ineficiente, até mesmo com o campo de hashing

Hashing Extensível

- Diretório: Tabela com 2^d entradas. Cada entrada aponta para um bucket;
  - d =profundidade global;
- Os d primeiros bits do resultado do Hashing determinam em que bucket será guardado o registro;
- Cada bucket pode conter até m registros que possuem os $d_0 \leq d$ primeiros bits iguais.
  - d_0 = Profundidade Local;



Hashing Extensível

- Um novo bucket é criado quando há colisões. A profundidade do bucket é aumentada de um e os registros são divididos de acordo com essa nova profundidade;
- Quando d_0 se torna maior que d, d é incrementado de um e o tamanho do diretório é dobrado;

Hashing Extensível

- Quando um bucket fica vazio ou quando dois buckets podem ser unidos em um só, um bucket é removido e d_0 é decrementado;
- Quando todos os d_0 são menores que d, d é decrementado de um e o tamanho do diretório cai pela metade

Ordenação Externa

Modelo de Computação Baseada em E/S

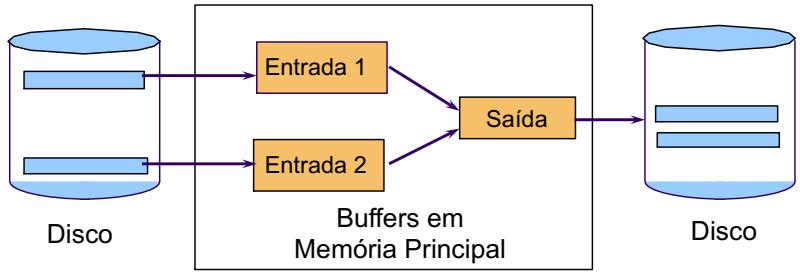
- Para algoritmos em memória, a principal preocupação é com tempo de CPU
- Em BDs o tempo é dominado pelos custos de E/S
- Consequência: alguns algoritmos precisam ser re-projetados
- Exemplo: ordenação

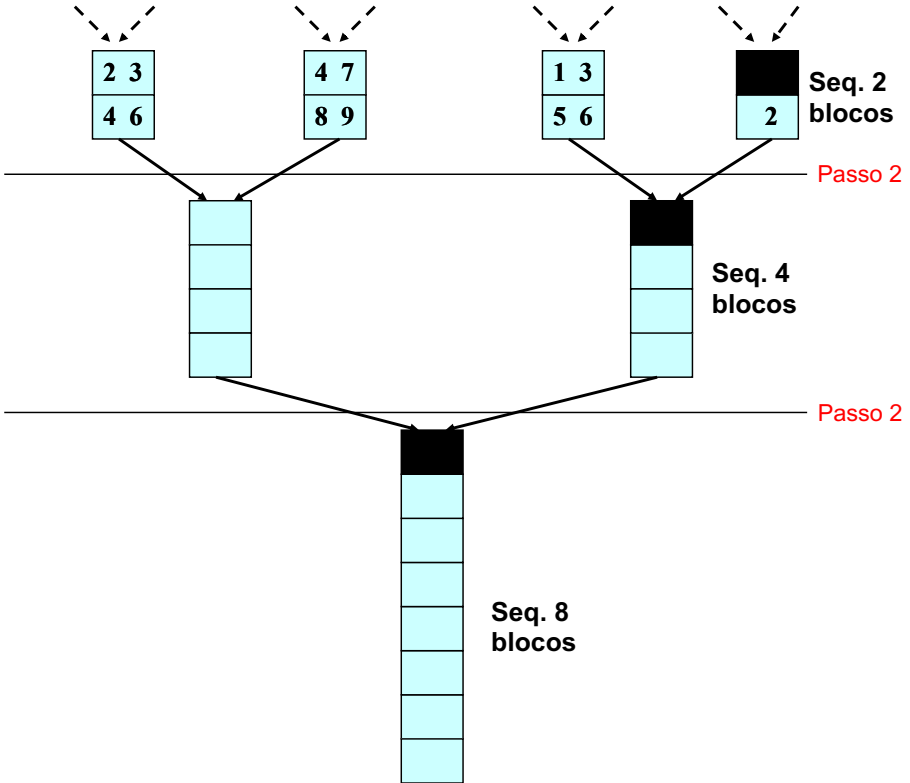
Exemplo: Ordenação Externa

- Problema: ordenar 1Gb de dados com 1Mb de RAM
- Aplicações em BDs:
  - Ordenar resultados de consultas (ORDER BY)
  - Operações de agrupamento
  - Primeiro passo no método sort-merge para implementar junções
  - Remoção de duplicatas
  - Carga de massa de dados em índices (B^+-Tree)

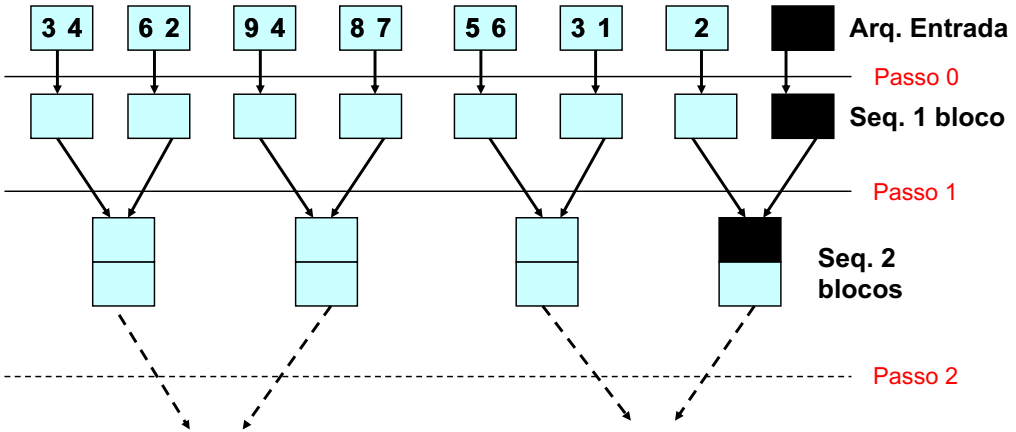
Merge-Sort Externo de 2 vias

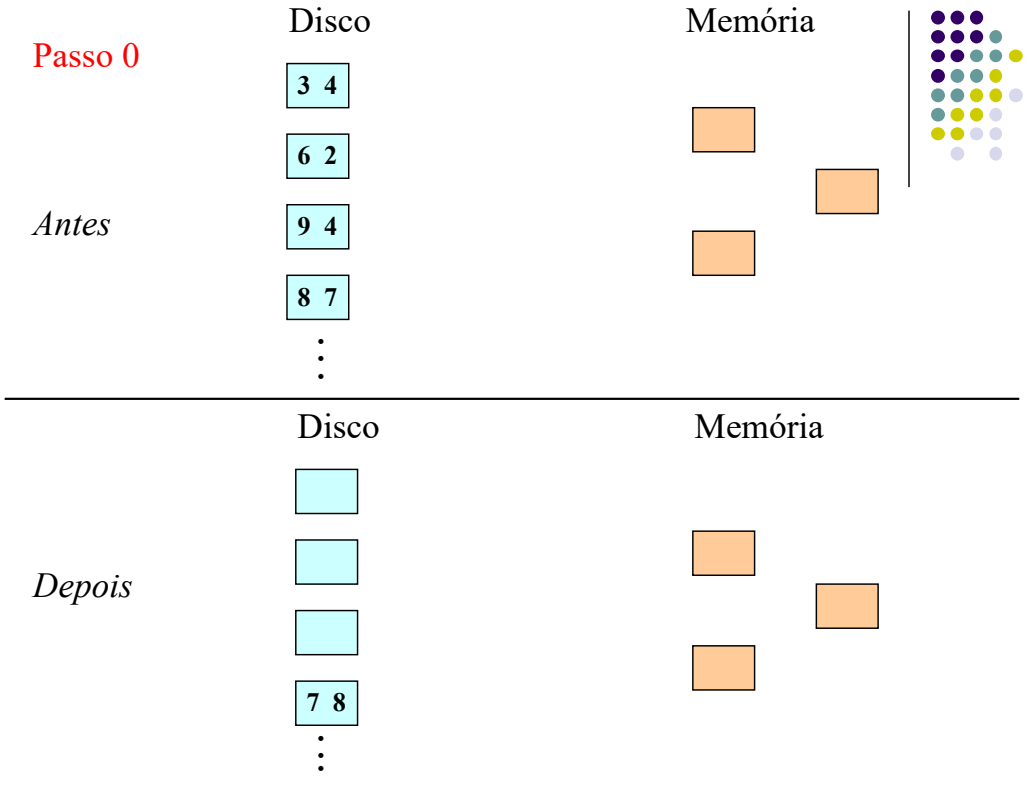
- Passo 1:
  - Ler um bloco, ordená-lo e escrevê-lo
  - Apenas um buffer é usado
- Passos 2, 3, …, etc.:
  - Três buffers são usados





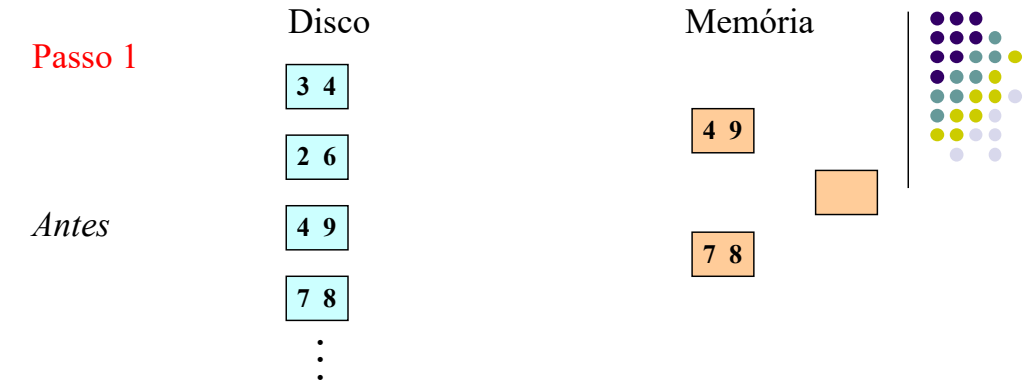
Merge-Sort Externo de 2 vias





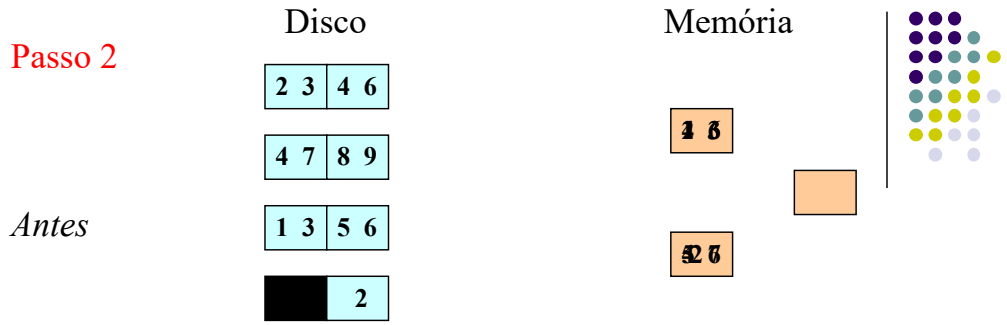
Passo 1

Antes

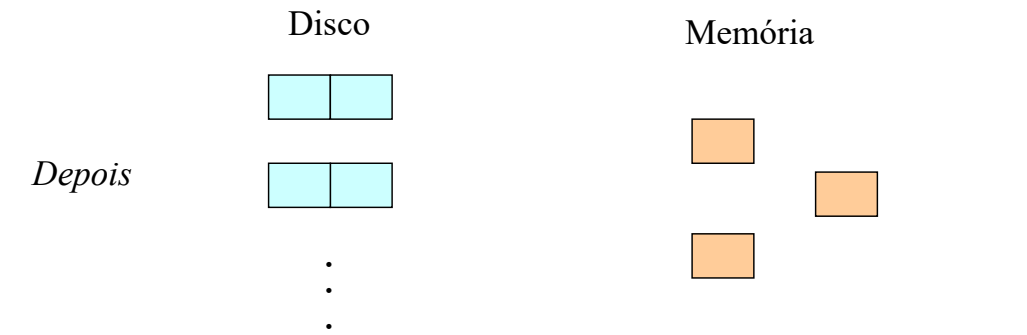


Passo 2

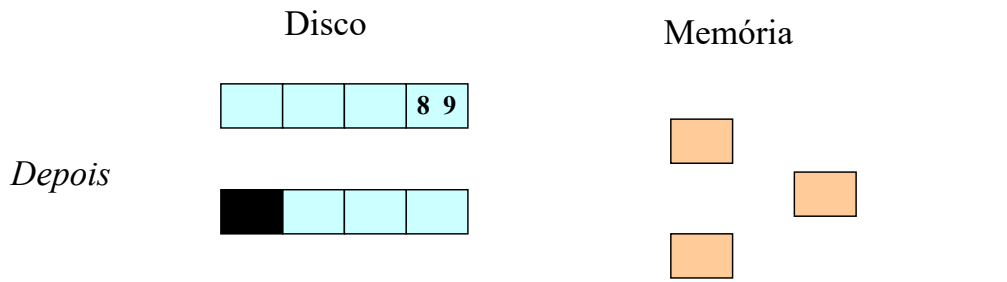
Antes



Depois

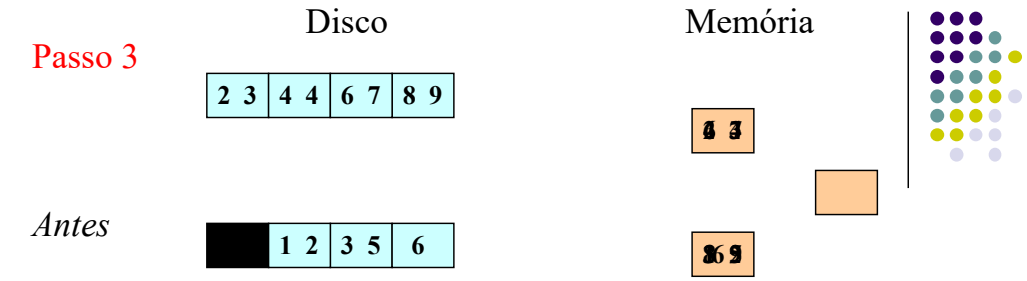


Depois

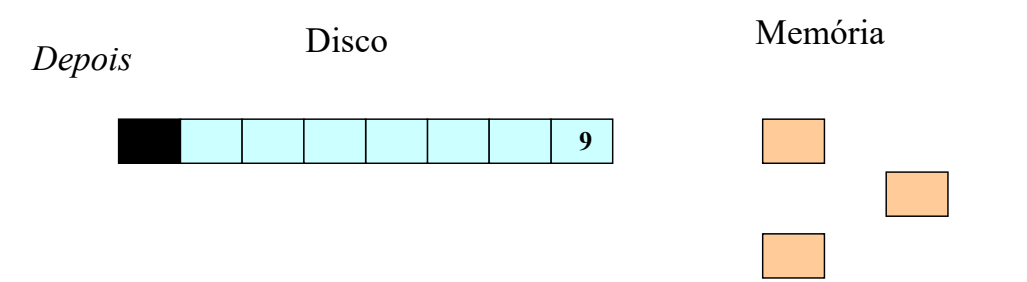


Passo 3

Antes



Depois



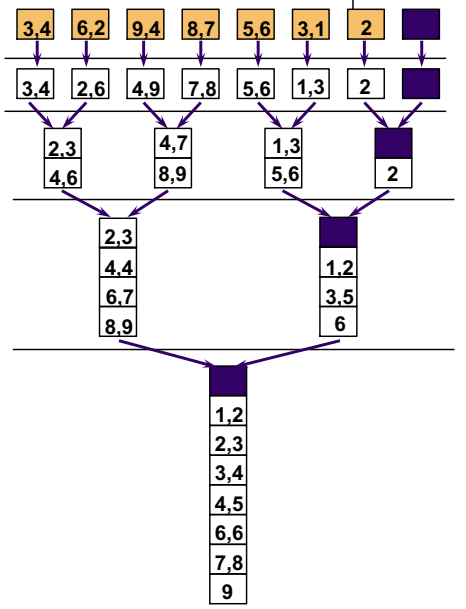
Merge-Sort Externo de 2 vias

- A cada passo ocorrem
  - 1 acesso de leitura
  - 1 acesso de escrita
- Com um arquivo de N blocos temos que o número de passos é:

$$= \lceil \log_2 N \rceil + 1$$

- Assim o custo total é:

$$2N\left(\lceil \log_2 N \rceil + 1\right)$$

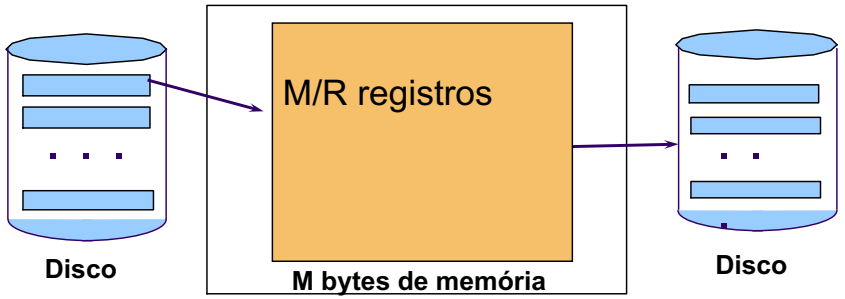


Merge-Sort Externo Multi-Vias

- Para aproveitar melhor a memória, podemos aumentar o número de buffers processados a cada passo
- Isso é usado para melhorar a performance
- Considere
  - B: Tamanho do bloco
  - M: Tamanho da memória disponível
  - N: Número de registros no arquivo
  - R: Tamanho de um registro

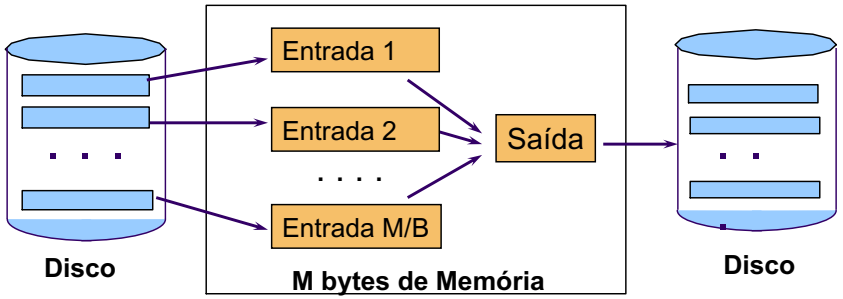
Merge-Sort Externo Multi-Vias

- Passo 1:
  - Carregar M bytes na memória e ordenar
  - Resultado: sequências de M/R registros



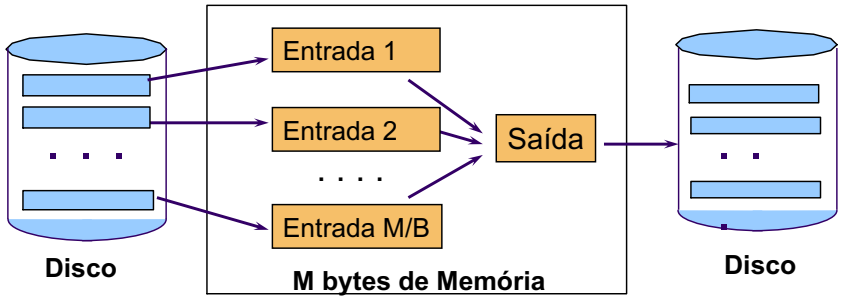
Merge-Sort Externo Multi-Vias

- Passo 2
  - Intercalar M/B – 1 sequências gerando novas sequências
  - Resultado: sequências de M/R (M/B – 1) registros



Merge-Sort Externo Multi-Vias

- Passo 3
  - Intercalar M/B – 1 sequências gerando novas sequências
  - Resultado: sequências de M/R $(M/B – 1)^2$ registros



Merge-Sort Externo Multi-Vias

- Número de passos: $1+\lceil \log_{M/B-1} \lceil NR/M \rceil \rceil$

- Exemplo
  - B = 4KB, M = 64MB, R = 0,1KB
  - Passo 1:
    - Sequências de tamanho M/R = 640.000
    - Resulta em 640.000 registros ordenados
  - Passo 2:
    - Sequências maiores por um fator de M/B – 1 = 16.000
    - Resulta em 10.240.000.000 = 10^{10} registros
  - Passo 3:
    - Sequências maiores por um fator de M/B – 1 = 16.000
    - Resulta em 10^{14} registros
  - Quantidade significativa de dados
  - Pode-se ordenar qualquer coisa em 2 ou 3 passos !