ACH2024 –

Algoritmos e Estruturas de Dados II

Prof. Helton Hideraldo Biscaro
heltonhb@usp.br

Parte 2 da Disciplina:

Memória Secundária

Organização de Arquivos

Nesta aula são introduzidos conceitos sobre o modelo consequencial de processamento de arquivos

Relembrando...

- É claro que, para cada tipo de consulta, se o arquivo não estiver organizado adequadamente, é necessário uma ou mais varreduras completas (do início ao final) do arquivo
- Uma primeira idéia seria organizar um arquivo ordenando-o segundo algum campo
- Um método comum de ordenação consiste na ordenação (ou classificação) por intercalação
- A intercalação pressupõe dois arquivos ordenados por algum campo, juntando-os em um único arquivo final, também ordenado
- Em termos gerais a intercalação (*merge*) é o processo de formar uma lista de saída contendo **todos** os itens de duas (ou mais) listas de entrada

Algoritmo de Intercalação

```
procedure merge(X,Y,Z)
```

- *pré-condição:* assume dois arquivos de entrada X e Y ordenados pelo campo chave
- *pós-condição:* gera como saída o arquivo Z também ordenado por chave composto pelos registros de X e Y
- O procedimento precisa de 3 variáveis do tipo arquivo: duas variáveis de entrada para os arquivos X e Y e uma variável de saída para o arquivo Z
- No algoritmo, assume-se que os registros possuem um campo chave denominado **key**

Algoritmo de Intercalação – Pseudo Código

```
procedure merge(X,Y,Z)

 Leia registro x de X
 Leia registro y de Y
 while not X.Eof() and
       not Y.Eof() do
   if x.key < y.key then
     Escreva x em Z
     Leia x do arquivo X
   else
     Escreva y em Z
     Leia y do arquivo Y
   endif
 endwhile
```

```
   while not X.Eof() do
     Escreva x em Z
     Leia x do arquivo X
   endwhile

   while not Y.Eof() do
     Escreva y em Z
     Leia y do arquivo Y
   endwhile

end merge
```

Algoritmo de Intercalação – Pseudo Código – Variação

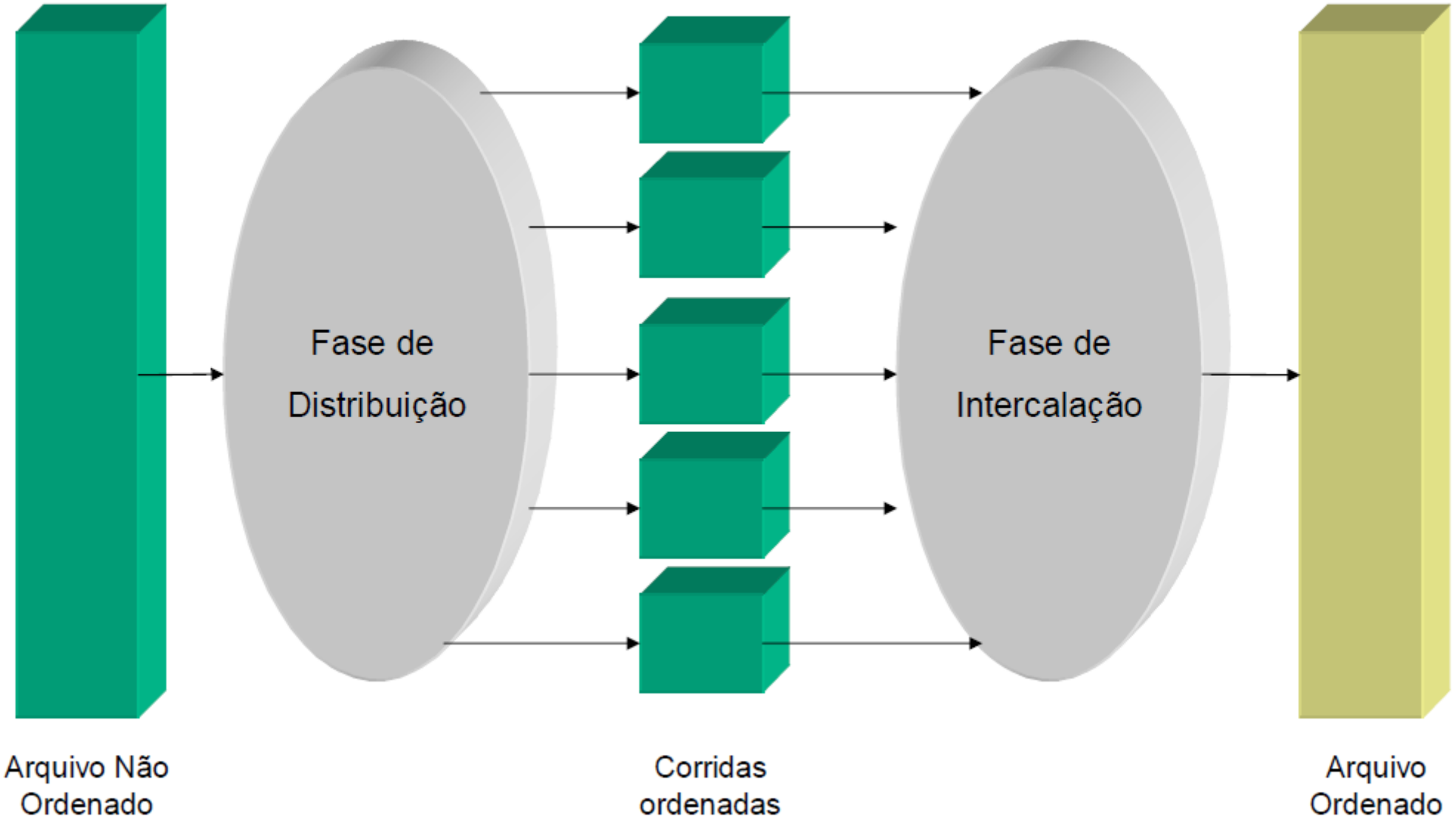
```
procedure merge(X,Y,Z)

 X.read(x);
 Y.read(y);
 acabou ← X.Eof() and Y.Eof();
 while not acabou do
   if x.key < y.key then
     Z.write(x);
     X.read(x);
   else
     Z.write(y);
     Y.read(y);
   endif
   acabou ← X.Eof() and Y.Eof();
 endwhile
```

Ordenação em Disco

- A Classificação por Intercalação é o método mais comum de ordenação nos dispositivos de memória secundária
- Esse método consiste essencialmente de duas fases distintas
  - Na primeira (distribuição), os segmentos do arquivo de entrada são classificados usando um bom método de ordenação em memória principal
    - Esses segmentos classificados, conhecidos como corridas (*runs*) estão sendo gravados na memória secundária na medida em que são gerados
  - Na segunda (intercalação), as corridas geradas na fase anterior são intercaladas até esgotar as corridas

Classificação por Intercalação



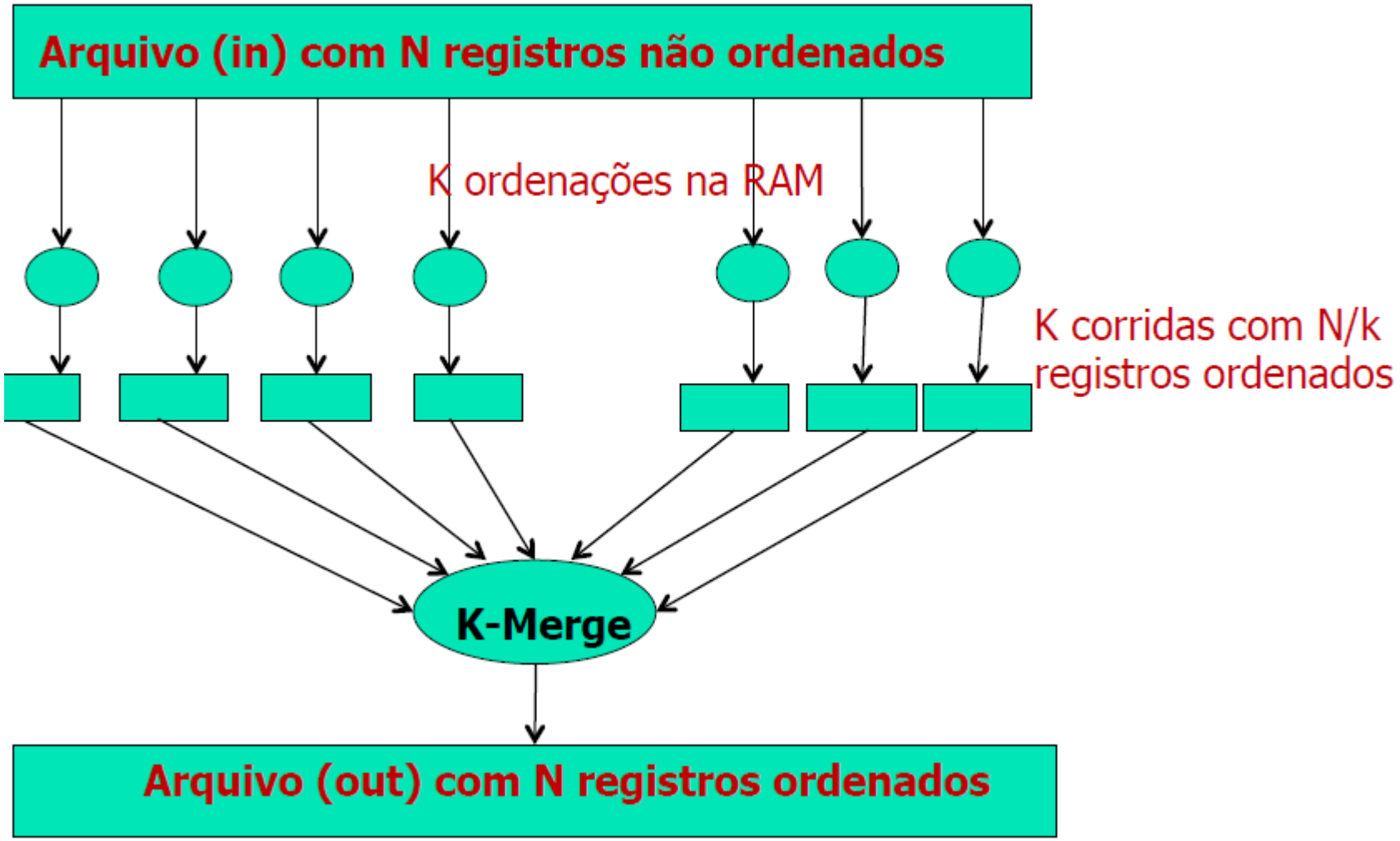
Desempenho

Os algoritmos para ordenação externa devem reduzir o número de passadas sobre o arquivo *(por que?)*

Uma boa medida de complexidade é o número de vezes que um item é lido ou escrito da/na memória externa

Bons métodos geralmente envolvem, no total, menos do que 10 passadas sobre o arquivo

Esquema Geral do Algoritmo:



Esquema Geral do Algoritmo:

- Esta solução
  - Pode ordenar arquivos realmente grandes
  - Geração das corridas envolve apenas acesso sequencial aos arquivos
  - A leitura das corridas e a escrita final também são sequenciais
  - Aplicável também a arquivos mantidos em fita, já que E/S é sequencial

K-Way Merge Sort

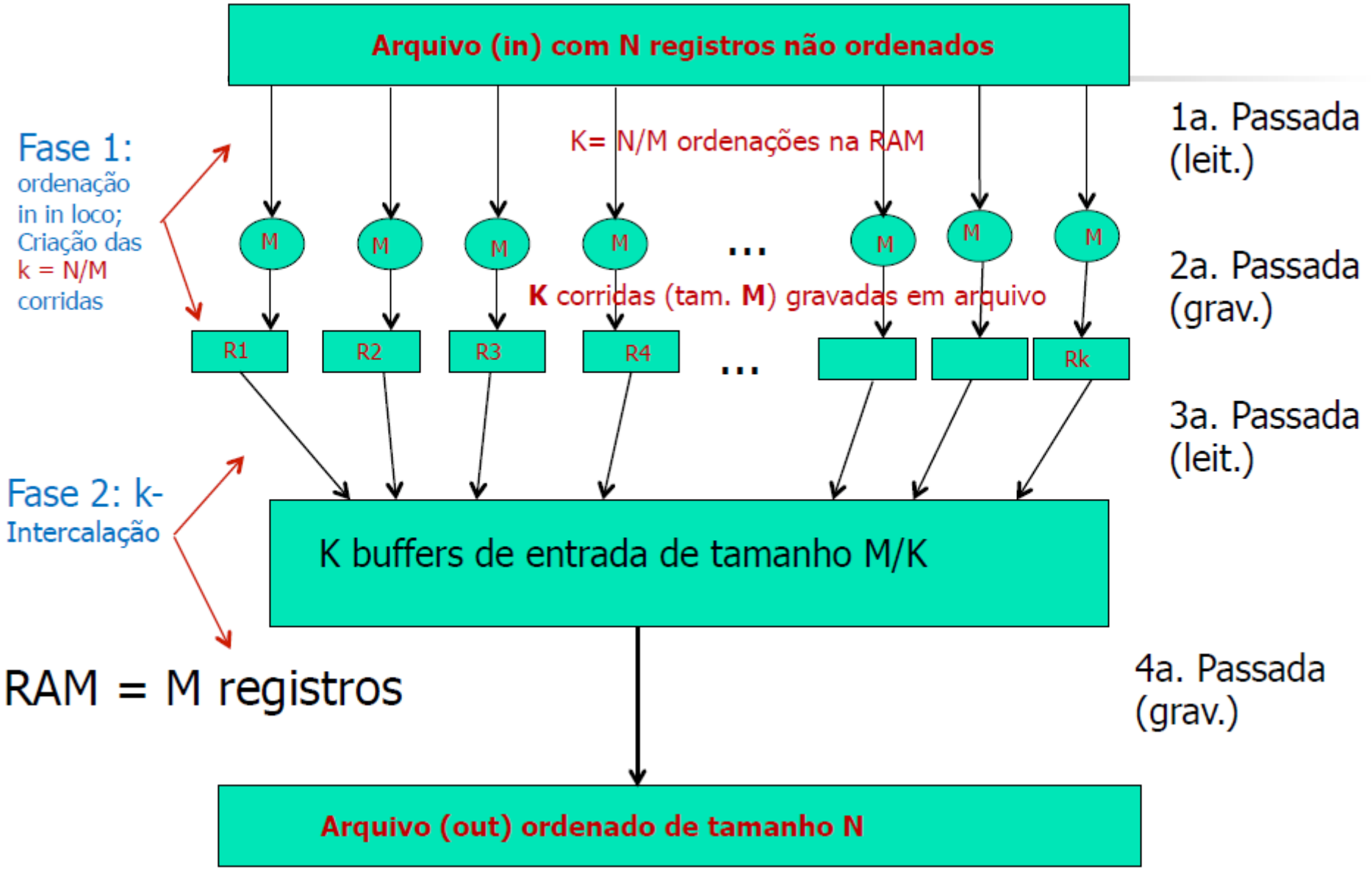
- Fase 1: Geração das Corridas
  - Segmentos do arquivo (corridas) são ordenados em memória RAM, usando algum método eficiente de ordenação interna (p.ex., quicksort ou heapsort), e gravados em disco*
  - Corridas vão sendo gravadas a medida em que são geradas

* Veja na literatura como fazer ordenação e gravação das corridas em paralelo, usando heapsort

K-Way Merge Sort

- Fase 2: Intercalação
  - As k corridas geradas na fase anterior são intercaladas (k-way), formando o arquivo ordenado, que vai sendo gravado no disco
    - Envolvem buffers de entrada – k chaves das corridas
    - E buffer(s) de saída – gravados no disco assim que completados
      - 1 buffer ou 2 buffers de saída – o que é melhor?

K-Way Merge Sort



K-Way Merge Sort - Análise

- Fase 1: Criação das Corridas:
  - 1 leitura sequencial do arquivo (N registros)
  - K=N/M gravações sequenciais de M registros

- Fase 2: Intercalação:
  - Toda a RAM é usada para buffers de entrada: abrigando K blocos de registros
  - No total, K*K seeks para ler todos os registros para a intercalação
  - Gravação dos N registros conforme ocorre a k-intercalação.

K-Way Merge Sort - Análise

- Apenas para ter um *benchmark;* não levar os números a sério!
- Supondo
  - Arquivo com 80 MB, com 800.000 registros de 100 bytes, e cada corrida com 1 MB
  - 1MB = 10.000 registros
  - Arquivo armazenado em áreas contíguas do disco (*extents*), *extents* alocados em mais de uma trilha, de tal modo que um único *rotational delay* é necessário para cada acesso
  - Características do disco
    - tempo médio para seek: 18 ms
    - atraso rotacional: 8.3 ms
    - taxa de transferência: 1229 bytes/ms
    - tamanho da trilha: 20.000 bytes

K-Way Merge Sort - Análise

- Quatro passos a serem considerados

Fase 1: Criação corridas

- Leitura dos registros, do disco para a memória, para criar as corridas
- Escrita das corridas ordenadas para o disco

Fase 2: Intercalação

- Leitura das corridas para intercalação
- Escrita do arquivo final em disco

Fase 1 - Leitura - Análise

- Lê-se 1MB de cada vez, para produzir corridas de 1 MB
- Serão 80 leituras, para formar as 80 corridas iniciais – *80-way Merge*
- O tempo de leitura de cada corrida inclui o tempo de acesso a cada bloco (seek + *rotational delay*) somado ao tempo necessário para transferir cada bloco

Fase 1 - Leitura - Análise

seek = 18ms, rot. delay = 8.3ms, total 26.3ms

Tempo total para a fase de ordenação:

80*(tempo de acesso a uma corrida) + tempo de transferência de 80MB

Acesso: 80*(seek+rot.delay= 26.3ms) = 2s

Transferência: 80 MB a 1.229 bytes/ms = 65s

Total: 67s

Fase 1 - Escrita - Análise

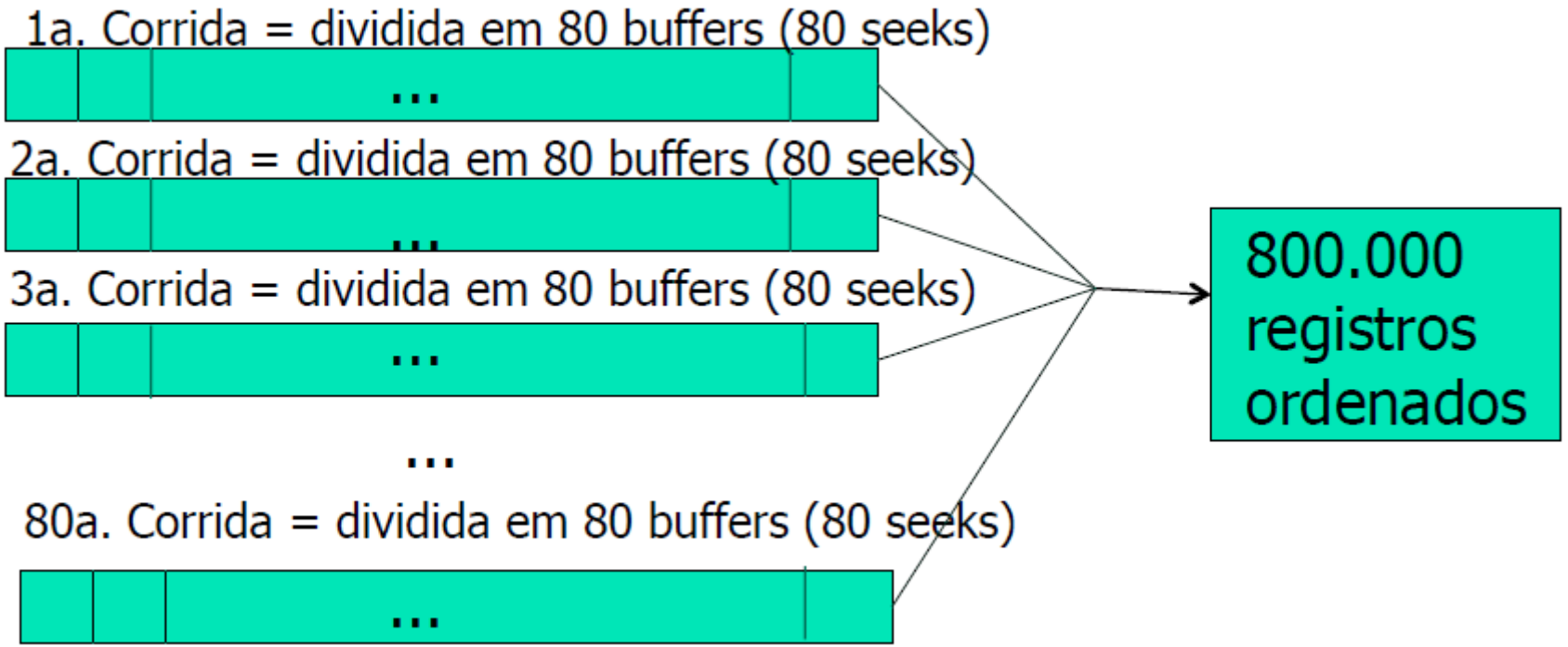
- Idem à leitura!

Serão necessários outros 67s

Fase 2 - Leirura - Análise

- 1MB de MEMÓRIA para armazenar 80 buffers de entrada
  - portanto, cada buffer armazena 1/80 de uma corrida (12.500 bytes) → cada corrida deve ser acessada 80 vezes para ser lida por completo
- 80 acessos para cada corrida X 80 corridas
  - 6.400 seeks
- considerando acesso = seek + rot. delay
  - 26.3ms X 6.400 = 168s
- Tempo para transferir 80 MB = 65s

Fase 2 - Leirura - Análise



Fase 2 - Escrita - Análise

- Precisamos saber o tamanho dos *buffers* de saída
- Nos passos 1 e 2, a MEMÓRIA funcionou como *buffer de entrada*, mas agora a MEMÓRIA está armazenando os dados a serem intercalados
- Para simplificar, assumimos que seja possível alocar 2 *buffers* de saída de 20.000 bytes, para escrita
  - dois para permitir *double buffering* (enquanto um é gravado no disco, o outro é preenchido com novos registros ordenados); 20.000 porque é o tamanho da trilha no nosso disco hipotético

Fase 2 - Escrita - Análise

- Com *buffers* de 20.000 bytes, precisaremos de 80.000.000 bytes / 20.000 bytes = 4.000 seeks
- Como tempo de seek+rot.delay = 23.6ms por seek, 4.000 seeks usam 4.000 X 26.3, e o total de 105s.
- Tempo de transferência é 65s

Tempo Total

- leitura dos registros para a memória para a criação de corridas: 67s
- escrita das corridas ordenadas para o disco: 67s
- leitura das corridas para intercalação: 168 + 65 = 233 s
- escrita do arquivo final em disco: 105 + 65 = 170 s
- tempo total do Mergesort = 537 s **(~9 minutos)**
  - Tempo dominado pelo acesso ao disco, e não pelo tempo de intercalação na RAM

Quando Usar o Merge Sort?

- Quando a ordenação deve ser feita só raramente
- Quando o arquivo não for grande demais

- Quando o Mergesort passa a ser problemático?

Arquivo Maior: 8.00.000 de Registros

- Análise - arquivo de 800 MB

- O arquivo aumenta, mas a memória não!
  - Em vez de 80 corridas iniciais, teremos 800
  - Portanto, seria necessária uma intercalação em 800-vias (k=800) no mesmo 1 MB de memória, o que implica em que a memória seja dividida em 800 buffers na fase de intercalação

Arquivo Maior: 8.00.000 de Registros

- Cada buffer comporta 1/800 de uma corrida, e cada corrida é acessada 800 vezes
- 800 corridas X 800 seeks/corrida = 640.000 seeks no total
- O tempo total agora é superior a **5 horas e 19** minutos, aproximadamente 36 vezes maior do que o arquivo de 80 MB (que é apenas 10 vezes menor do que este)

Arquivo Maior: 8.00.000 de Registros

- **Definitivamente**: necessário diminuir o tempo gasto obtendo dados na fase de intercalação

Custo de aumentar o tamanho do arquivo

- A grande diferença de tempo na intercalação dos dois arquivos (de 80 e 800 MB) é consequência da diferença nos tempos de acesso às corridas (seek e rotational delay) para intercalá-las

- Em geral, para uma intercalação em K-vias de K corridas, em que cada corrida é do tamanho da MEMÓRIA disponível, o tamanho do buffer para cada uma das corridas é de:

  (1/K) x tamanho da MEMÓRIA = (1/K) x tamanho de cada corrida
  (M/K)

Custo de aumentar o tamanho do arquivo

- Como temos K corridas gastando K seeks cada uma, a operação de intercalação requer K^2 seeks
- Medido em termos de seeks, o Mergesort é $O(K^2)$
- Como K é diretamente proporcional à N (K=N/M), o Mergesort é $O(N^2)$, em termos de seeks

Maneiras de reduzir o tempo

1. usar mais hardware (disk drives, MEMÓRIA, canais de I/O)

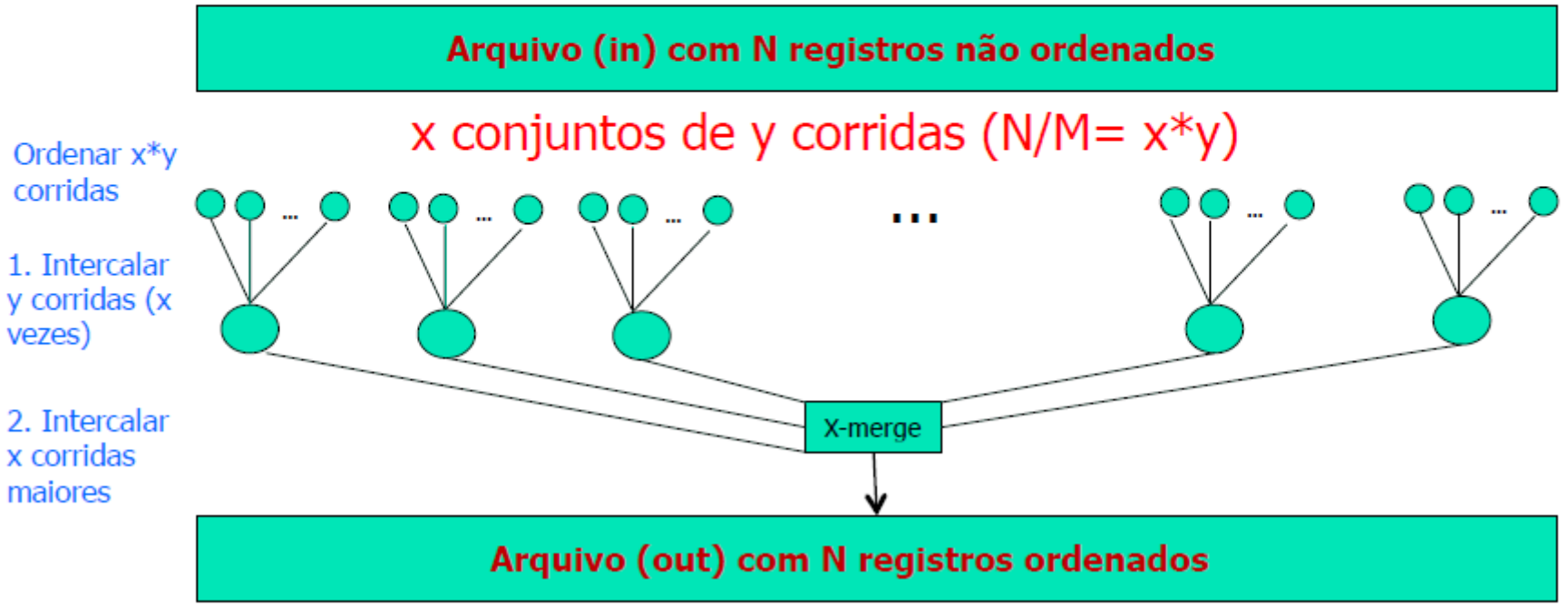
2. realizar a intercalação em mais de uma etapa, o que reduz a ordem de cada intercalação, aumentando o tamanho do buffer para cada corrida, portanto, transferindo mais registros com um único seek.

3. aumentar o tamanho das corridas iniciais (veja na bibliografia – *Replacement Selection*)

4. realizar I/O simultâneo à intercalação

Multistep Merging

- ao invés de intercalar todas as corridas simultaneamente, o grupo original é dividido em sub-grupos menores (K= x*y)



Multistep Merging

- intercalação é feita para cada sub-grupo
- para cada sub-grupo, um espaço maior é alocado para cada corrida, portanto um número menor de seeks é necessário
- uma vez completadas todas as intercalações pequenas, o segundo passo completa a intercalação de todas as corridas

Intercalação em Múltiplos passos

- No exemplo do arquivo com N=800 MB tinhamos 800 corridas com 10.000 registros cada (M=1MB). Para esse arquivo, a intercalação múltipla poderia ser realizada em dois passos:
  - primeiro, a intercalação de x=25 conjuntos de y=32 corridas cada (x*y=800)
  - depois, uma intercalação em 25-vias

Intercalação em Múltiplos passos – Análise

- Passo único visto anteriormente exige 640.000 seeks

- Para a intercalação em 2 passos, temos, no passo 1:
  - Cada intercalação em 32-vias aloca buffers que podem conter 1/32 de uma corrida. Então, serão realizados 32 X 32 = 1024 seeks
  - Então, 25 vezes a intercalação em 32-vias exige 25 X 1024 = 25.600 seeks
  - Cada corrida resultante tem 32 X 10.000 = 320.000 registros = 32 MB

Intercalação em Múltiplos passos – Análise

- No passo 2, cada uma das 25 corridas de 32 MB pode alocar 1/25 do buffer
  - portanto, cada buffer aloca 400 registros, ou seja, 1/800 corrida. Então, esse passo exige 800 seeks por corrida, num total de 25 X 800 = 20.000 seeks
- Total de seeks nos dois passos: 25.600 + 20.000 = 45.600 (contra 640.000 anteriores)

Intercalação em Múltiplos passos – Análise

- Nesse caso, cada registro é transmitido 4 vezes, em vez de duas. Portanto, gastamos mais 651s em tempo de transmissão
- Ainda, cada registro é escrito duas vezes: mais 40.000 seeks (assumindo 2 buffers de 20.000 bytes cada)
- Somando tudo isso, o tempo total de intercalação = 5.907s ~ 1hora 38 min
  - A intercalação em 800 vias consumia ~5 horas...