Ordenação de Arquivos Grandes

Organização e Recuperação de Dados

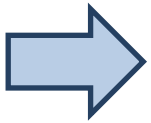
Profa. Valéria

UEM – CTC – DIN

Slides preparados com base no Cap. 7 do livro FOLK, M.J. & ZOELLICK, B. *File Structures*. 2nd Edition, Addison-Wesley Publishing Company, 1992, e nos slides disponibilizados pelo Prof. Pedro de Azevedo Berger (DCC/UnB)

Ordenação de arquivos grandes

"Arquivos grandes" são aqueles grandes demais para ordenação interna



Keysort

- Desvantagens do *keysort*
  - Depois de ordenar as chaves, existe um custo alto de *seeking* para ler e reescrever cada registro no arquivo novo
  - O tamanho do arquivo a ser ordenado ainda é limitado pelo número de pares chave/ponteiro que pode ser armazenado na RAM
    - Inviável para arquivo realmente grandes

Ordenação de arquivos grandes

- Exemplo hipotético:
  - Características do arquivo a ser ordenado:
    - 800.000 registros
    - Tamanho fixo dos registros: 100 bytes
    - Tamanho fixo da chave: 10 bytes
  - Tamanho total do arquivo: 80 MB
  - Memória disponível para a ordenação: 1 MB
  - Memória necessária apenas para as chaves: 8 MB

Usaremos valores aproximados:
1.000B = 1KB
1.000KB = 1MB
1.000MB = 1GB

Não é possível fazer *ordenação interna* nem *keysort*

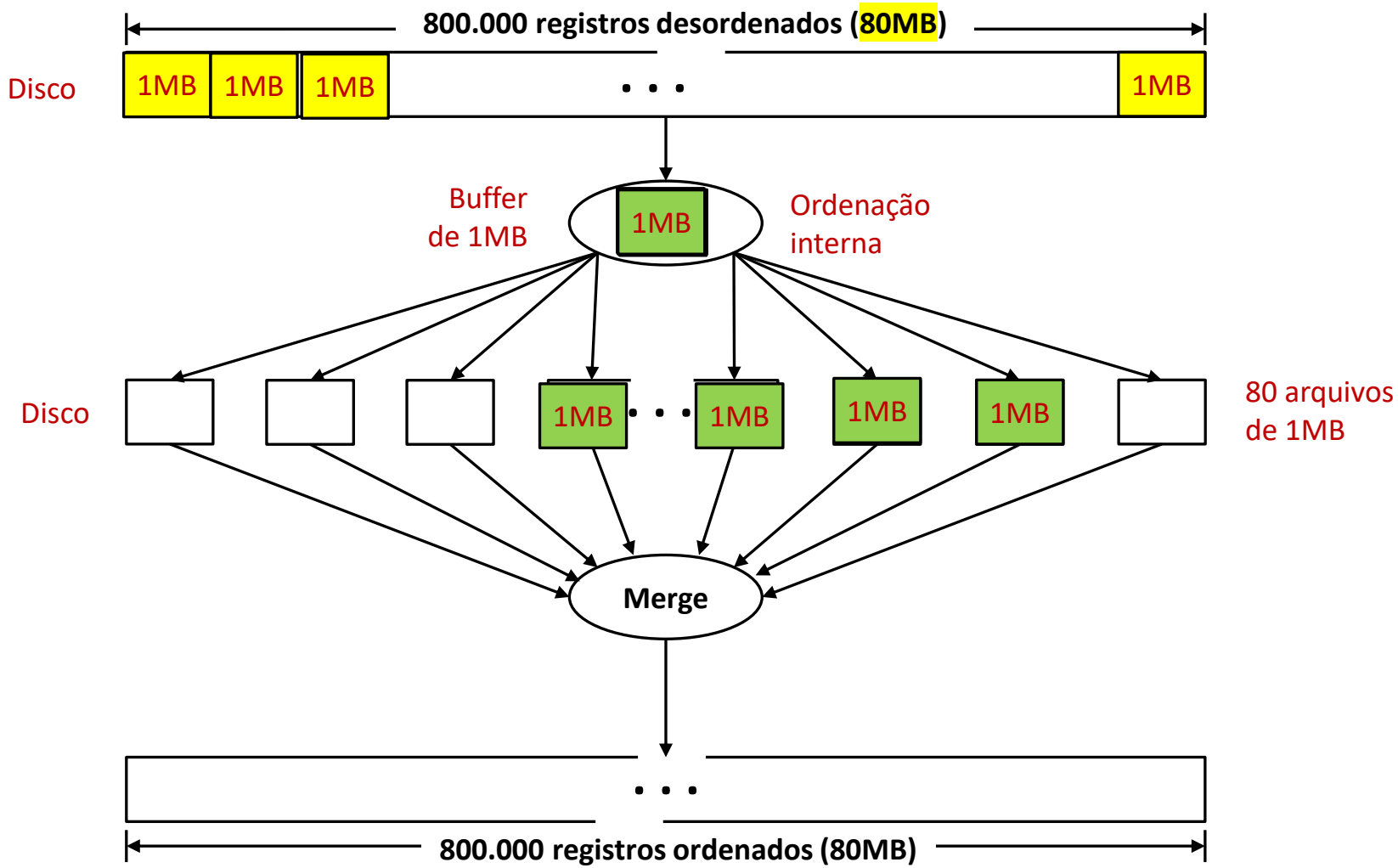
Ordenação de arquivos grandes

- *Merge sort externo*
  1) Criar arquivos menores ordenados que chamaremos de "partições"
     - Trazer o máximo de registros possíveis para a memória, fazer **ordenação interna** e **salvar em um arquivo temporário (partição)**
       - Qualquer algoritmo de ordenação interna pode ser utilizado para ordenar as partições
     - Repetir o processo até que todos os registros do arquivo original tenham sido lidos, ordenados e gravados em uma partição
  2) Fazer a intercalação dos arquivos ordenados (***K-way merge***)
     - *K* define o número de partições que serão intercaladas

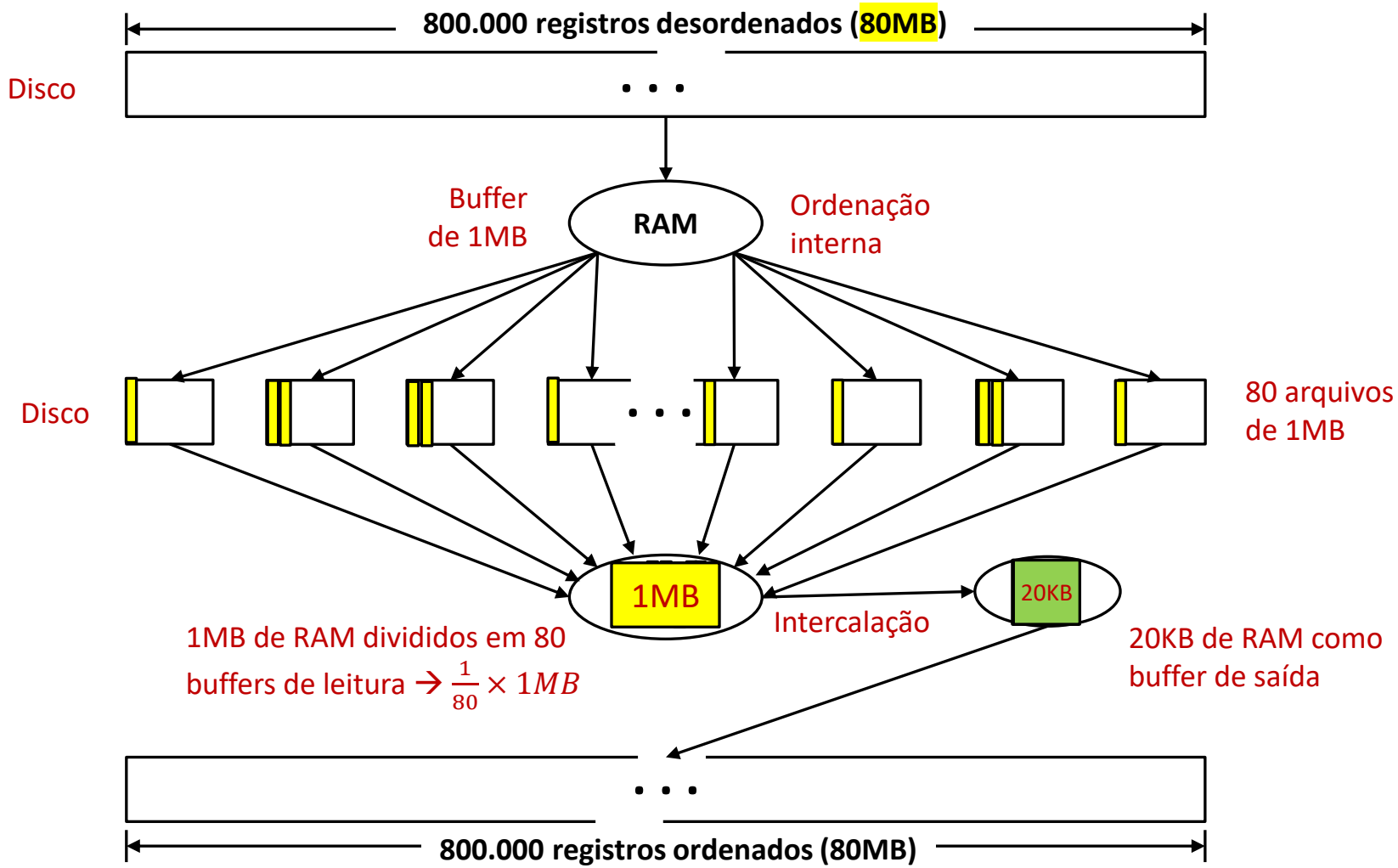
Merge sort externo

- No exemplo anterior (arquivo de 80 MB), qual seria o tamanho de cada partição?
  - Memória disponível = 1MB = 1.000.000 bytes
  - Tamanho dos registros = 100 bytes
  - Quantos registros cabem na memória disponível?
    - 10.000 registros ← tamanho de cada partição
  - Se o total de registros é 800.000, qual o número total de partições?
    - 80 partições ← total de partições
- As 80 partições estarão em 80 arquivos separados que posteriormente serão intercalados
  - ***80-way merge*** para gerar o arquivo final ordenado

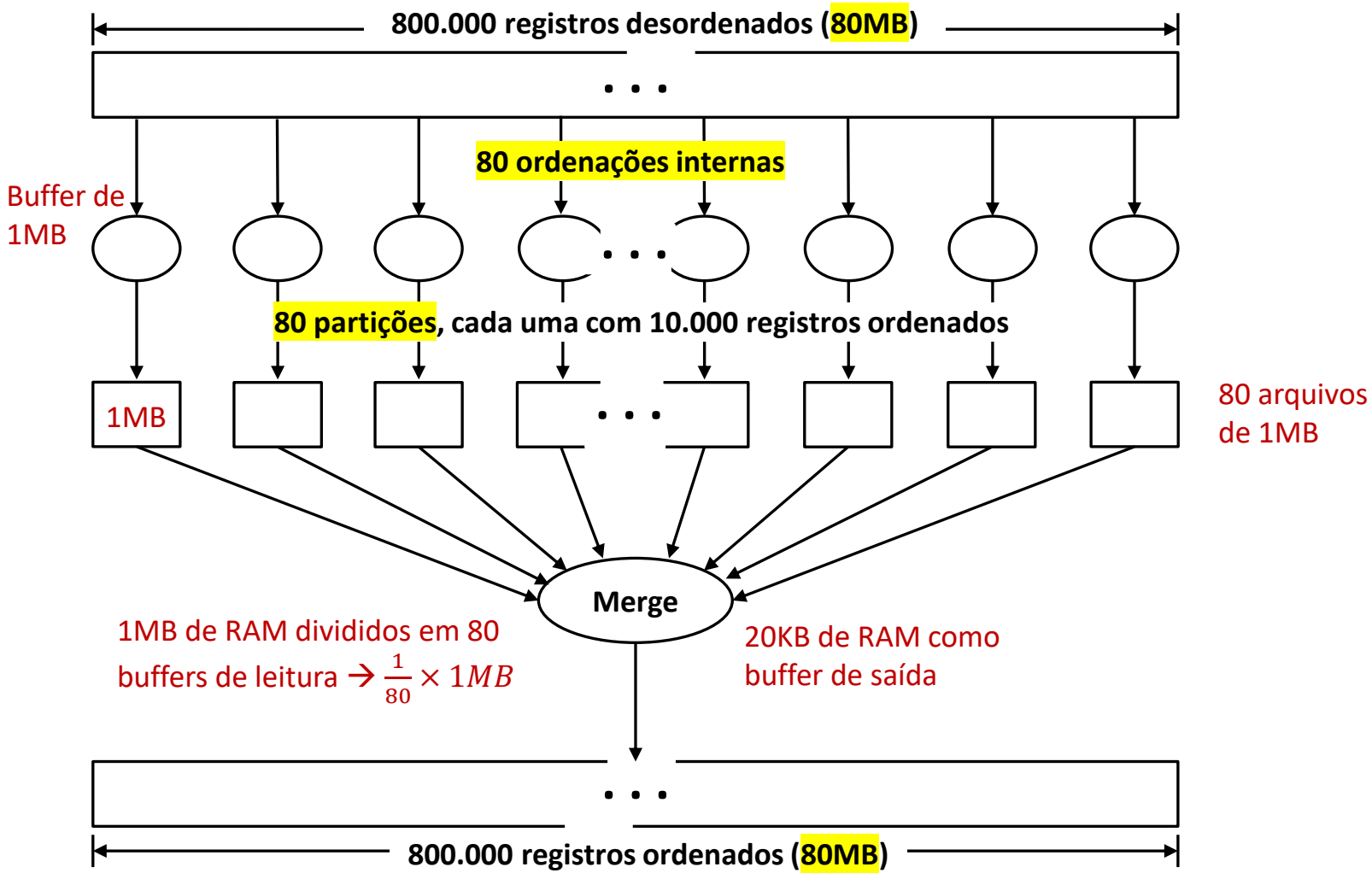
Merge sort externo



Merge sort externo



Merge sort externo



Merge sort externo

- Características do *Merge sort externo* (partições ordenadas + intercalação múltipla)
  - É extensível para arquivos de qualquer tamanho
  - A leitura do arquivo de entrada para a criação das partições é sequencial
  - A leitura das partições durante o processo de *merging* e a escrita dos registros ordenados também é sequencial
    - Teremos acesso aleatório quando alternamos a leitura entre as partições durante o processo de intercalação

Qual o custo do *merge sort* em disco?

- O maior custo da ordenação externa é devido as operações em disco – *seeks* e transferência
- No *Merge Sort*, operações de E/S são realizadas 4 vezes
  - Durante a fase de ordenação (*Sorting*)
    1. Leitura dos registros para a memória para criar as partições
    2. Escrita das partições ordenadas no disco
  - Durante a fase de intercalação (*Merging*)
    3. Leitura das partições ordenadas para a memória para realizar o *merge*
    4. Escrita do arquivo final ordenado no disco

Qual o custo do *merge sort* em disco?

1. Leitura dos registros para a memória para criar as partições
2. Escrita das partições ordenadas no disco

- Os passos 1 e 2 são feitos da seguinte forma:
  - Leia um bloco de 1MB e escreva uma partições ordenada de 1MB
    - 1 seek para a leitura e 1 seek para a escrita
  - Repita o passo anterior 80 vezes
  - Em termos de operações em disco:
    - Para leitura: 80 seeks + tempo de transferência p/ 80MB
    - Para escrita: 80 seeks + tempo de transferência p/ 80MB

Qual o custo do *merge sort* em disco?

3. Leitura das partições ordenadas para a memória para realizar o *merge*

- Para que o *merge* possa ser realizado, é preciso ler as 80 partições simultaneamente
  - Divida a memória de 1MB em 80 *buffers* de entrada
  - Cada *buffer* conterá 1/80 de uma partição
    - Cada partição será acessada 80 vezes para que seja lida por completo
    - Consideramos 1 seek para cada leitura
  - Cada uma das 80 partições será acessada 80 vezes (80*80 = 6.400)
    - Operações em disco: 6.400 seeks + tempo de transferência de 80MB

Qual o custo do *merge sort* em disco?

4. Escrita do arquivo ordenado no disco

- Para escrever o arquivo ordenado no disco, o número de *seeks* depende do tamanho do *buffer* de saída
  - Qtd de bytes no arquivo/Qtd de bytes no *buffer* de saída
    - Ex.: se o *buffer* de saída for de 20KB, serão 4.000 *seeks* (80MB/20KB)
  - A transferência novamente será de 80MB

    Obs.: A memória de 1MB está sendo usada pelo passo 3, de modo que pelo menos um *buffer* de saída adicional será necessário

Conclusão: A fase de *merging* (passos 3 e 4) é o gargalo

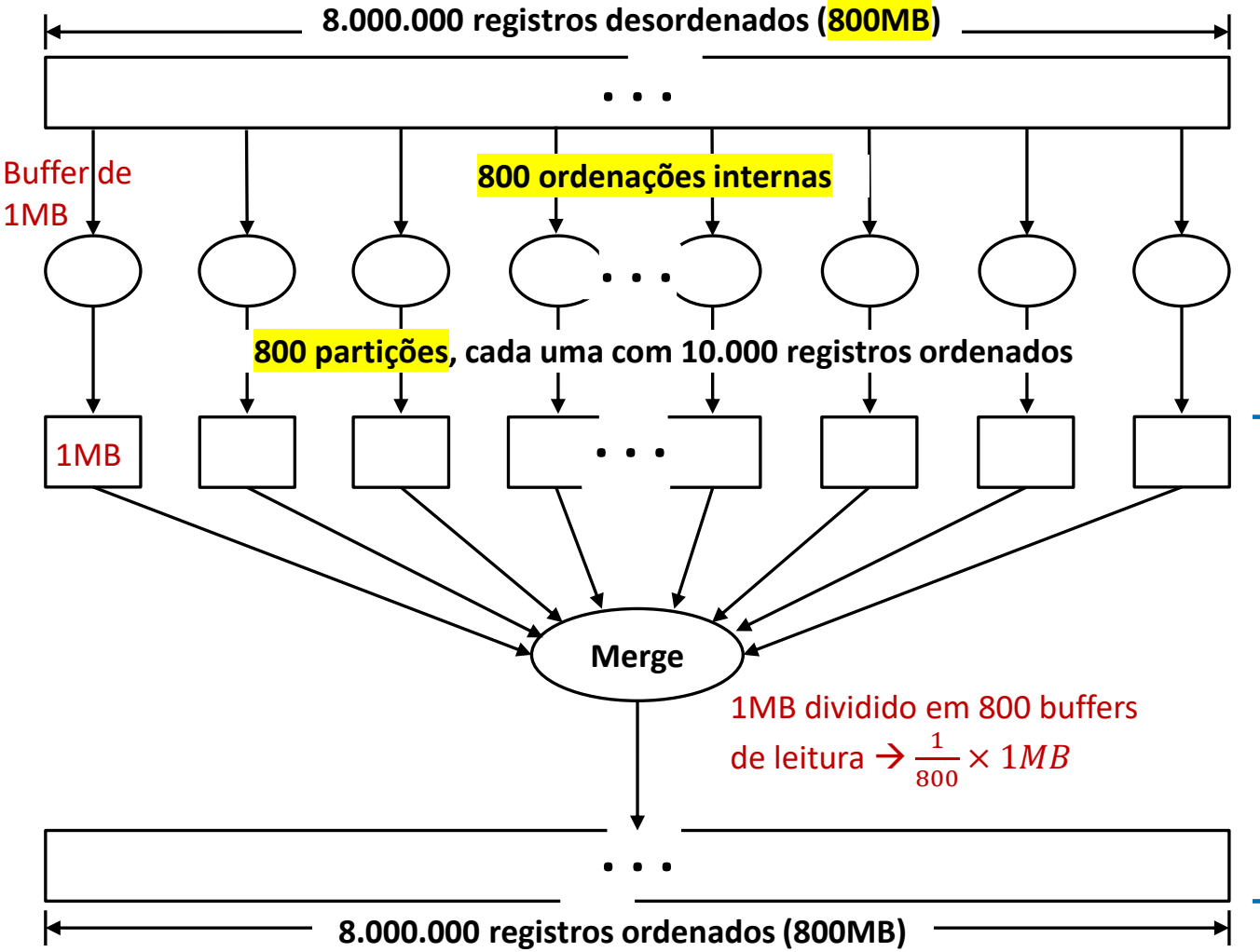
Custo do *merge sort* em disco

- E se formos ordenar um arquivo de 800MB em vez de 80MB?
  - Usando o mesmo tamanho de memória (1MB)
  - Os *seeks* aumentarão em qual proporção?

O número de ***seeks*** será **100 vezes maior** para o arquivo de 800MB

Enquanto o tamanho da entrada aumentou 10 vezes, o número de *seeks* do passo 3 aumentou 100 vezes (Teremos **680.000** *seeks* só na fase de *merging*)

Custo do *merge sort* em disco



Passo 3:
800*800 =
640.000 *seeks*

Passo 4:
800.000KB ÷ 20KB =
40.000 *seeks*

Total = 680.000 *seeks*

Custo do *merge sort* em disco

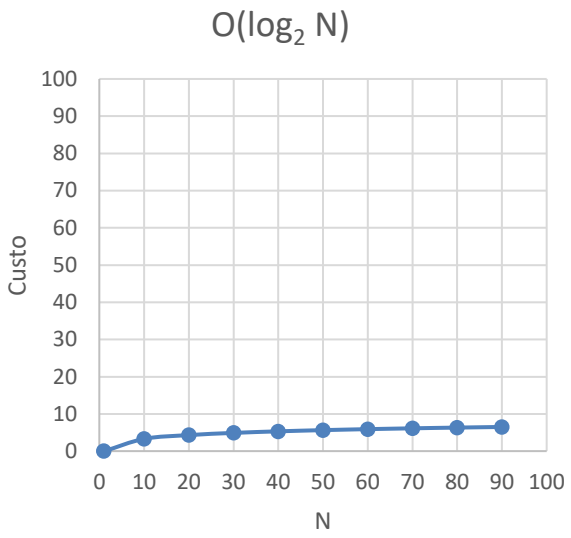
- Em geral, para um ***K-Way Merge*** com ***K* partições**, em que cada partição é do tamanho da memória disponível, o tamanho do *buffer* para cada partição é de:

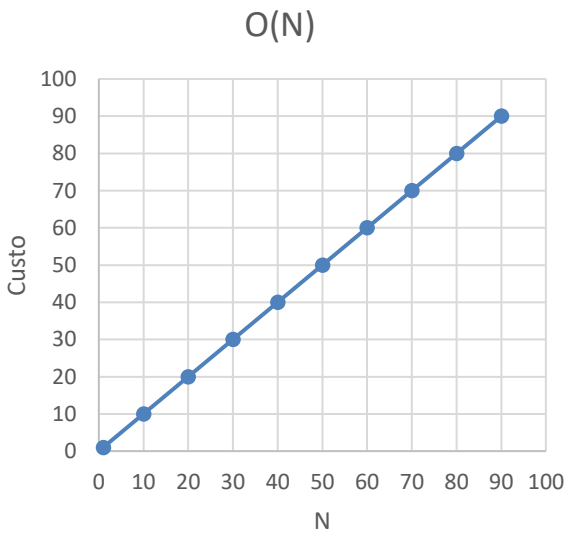
  (1/K) * tamanho da MEMÓRIA = (1/K) * tamanho de cada partição

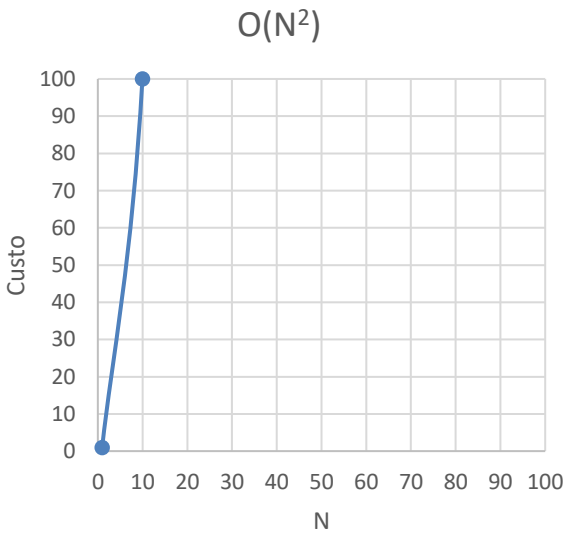
  - *K seeks* serão necessários para ler todos os registros de uma partição
- Como temos *K* partições e cada partição será lida *K* vezes, a operação de intercalação requer K^2 *seeks*
  - Assim, em termos de *seeks*, o *Merge Sort* Externo é O(K^2)
  - Mantendo o tamanho da memória fixo, temos que *K* é diretamente proporcional a *N*, então dizemos que o ***MergeSort* é O(N^2)**

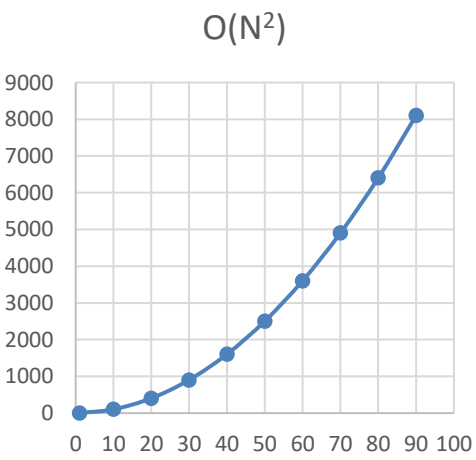
Conforme o arquivo cresce, o tempo requerido para realizar a ordenação cresce rapidamente!

Comportamento de diferentes funções de custo





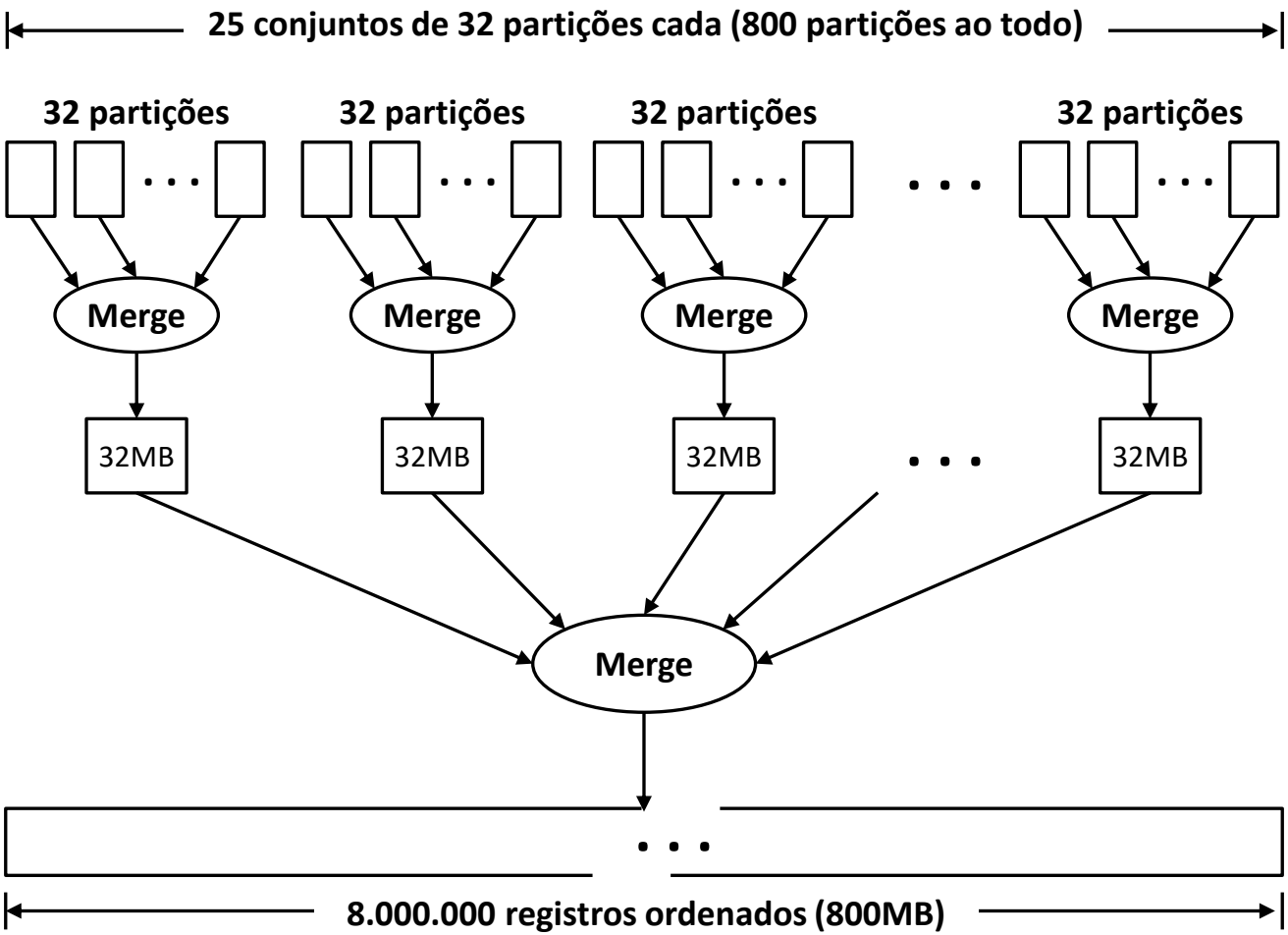




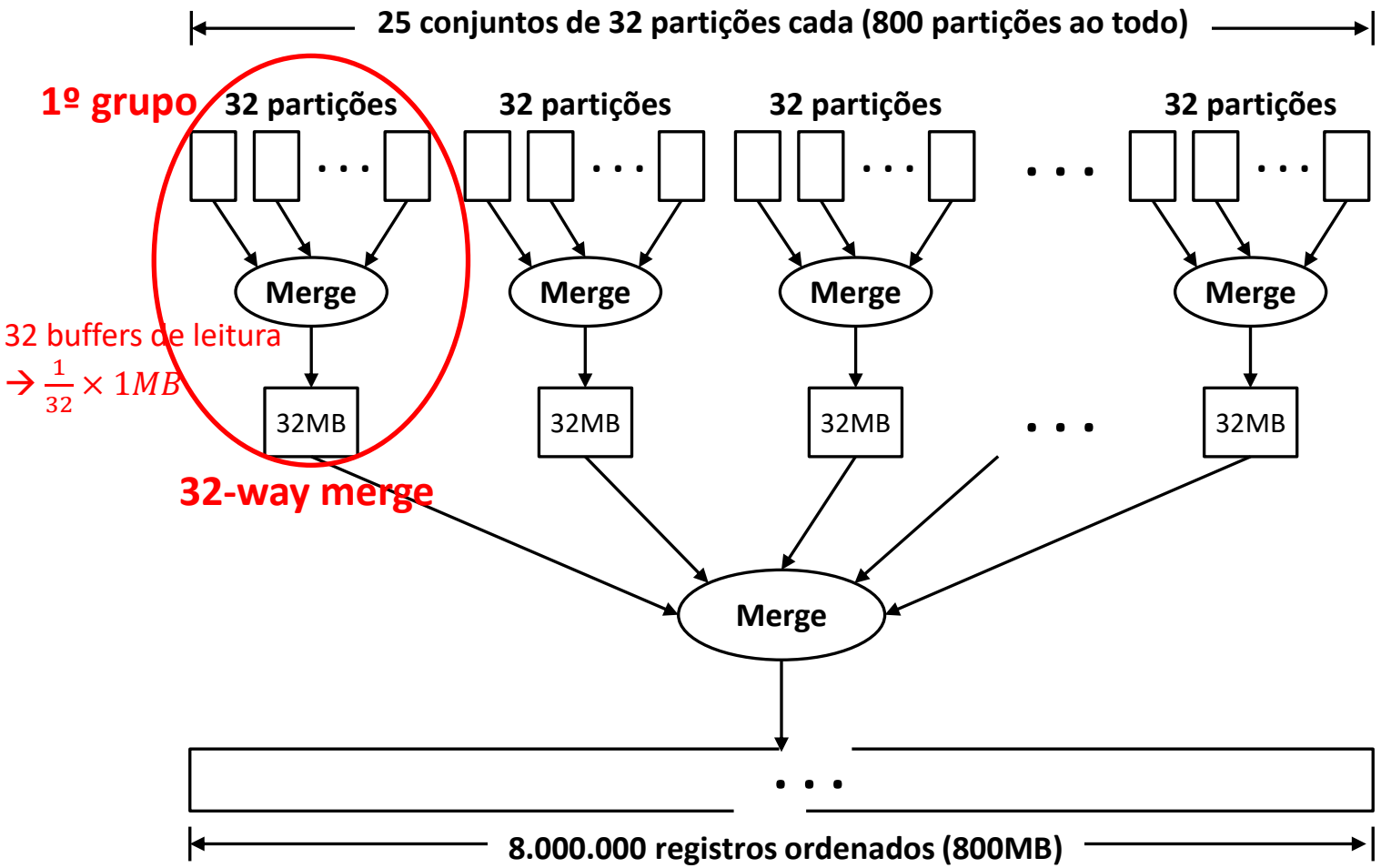
Redução do custo do *merge sort*

- Merge em múltiplos passos
  - Em vez de fazer o *merge* de todas as partições ao mesmo tempo, o grupo original de partições é dividido em subgrupos
  - Um *merge* é feito para cada subgrupo
    - As partições de um subgrupo poderão alocar um espaço maior do *buffer*, portanto, um número menor de *seeks* será realizado
  - Uma vez terminados os *merges* dos subgrupos (1º passo), o 2º passo completa o *merge* das partições resultantes do passo anterior

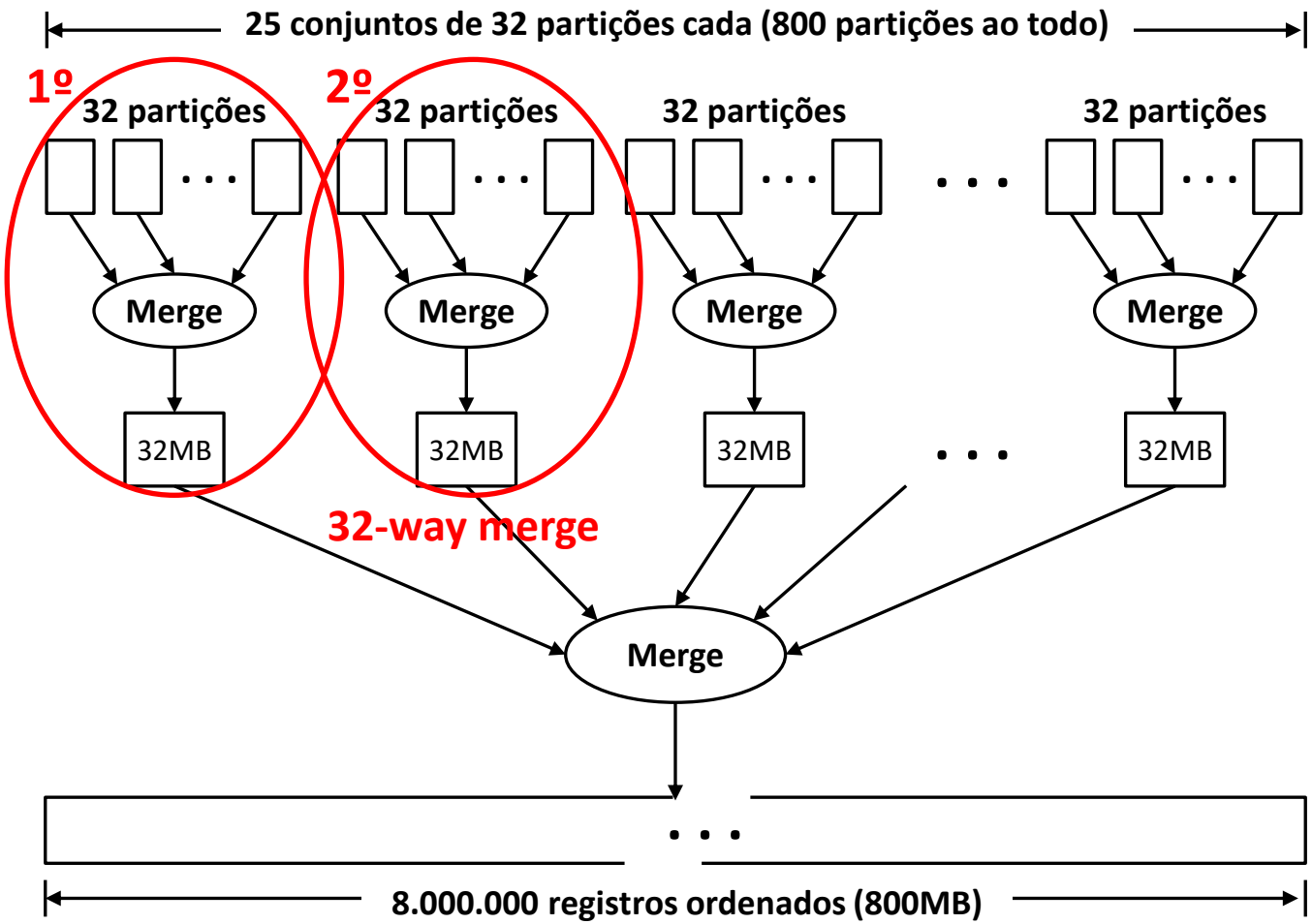
Merge 25 × 32-vias + 25-vias



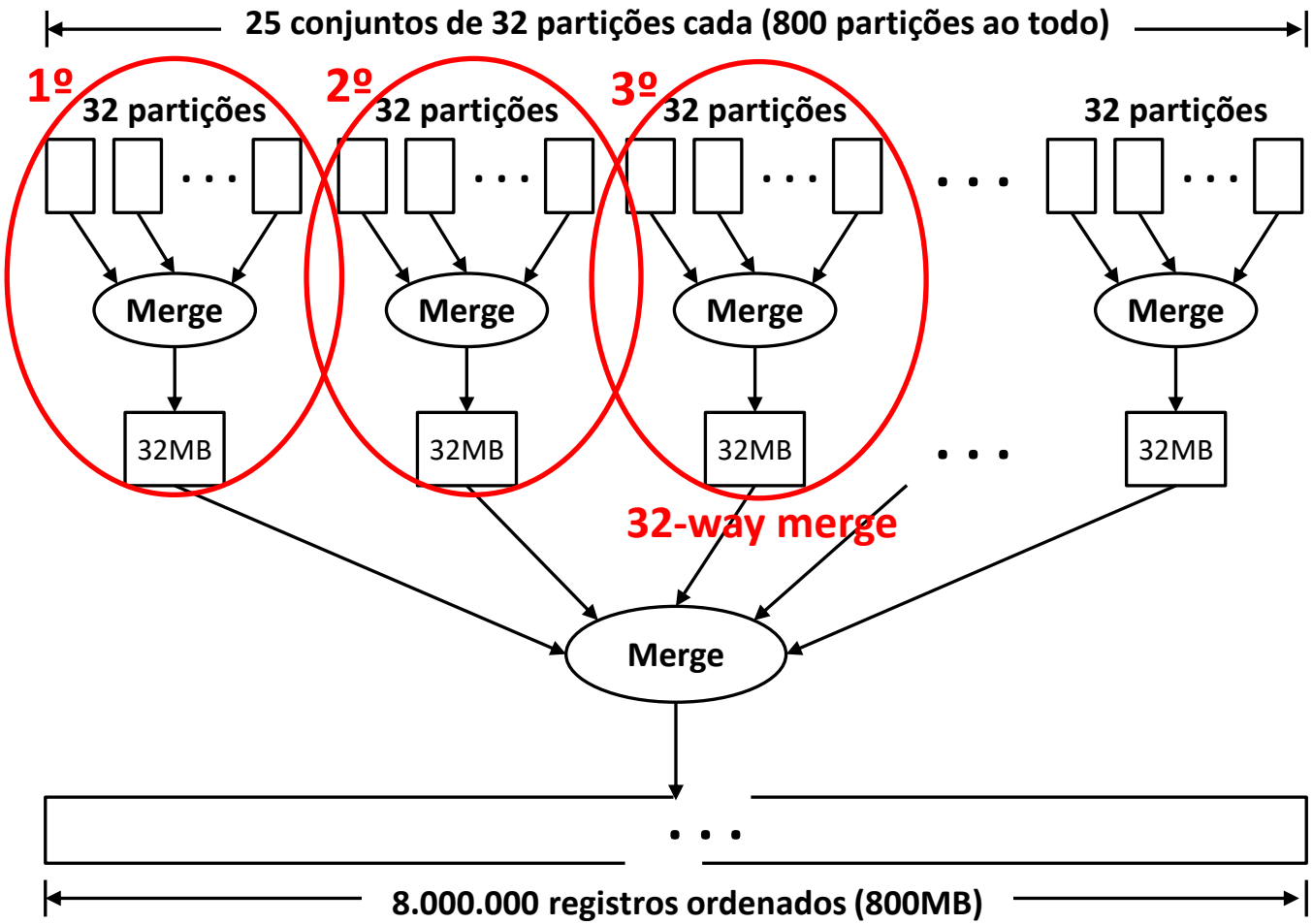
Merge 25 × 32-vias + 25-vias



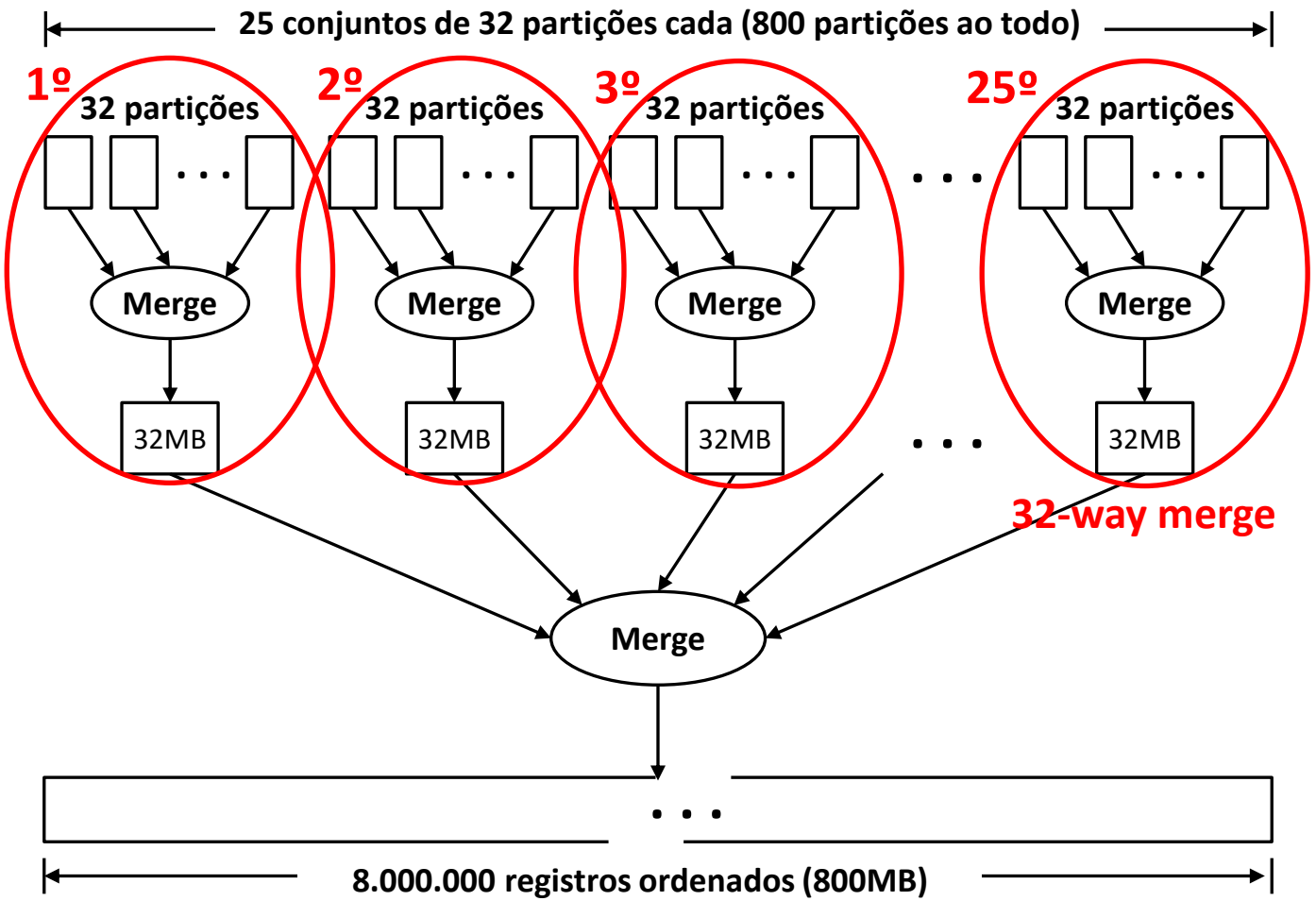
Merge 25 × 32-vias + 25-vias



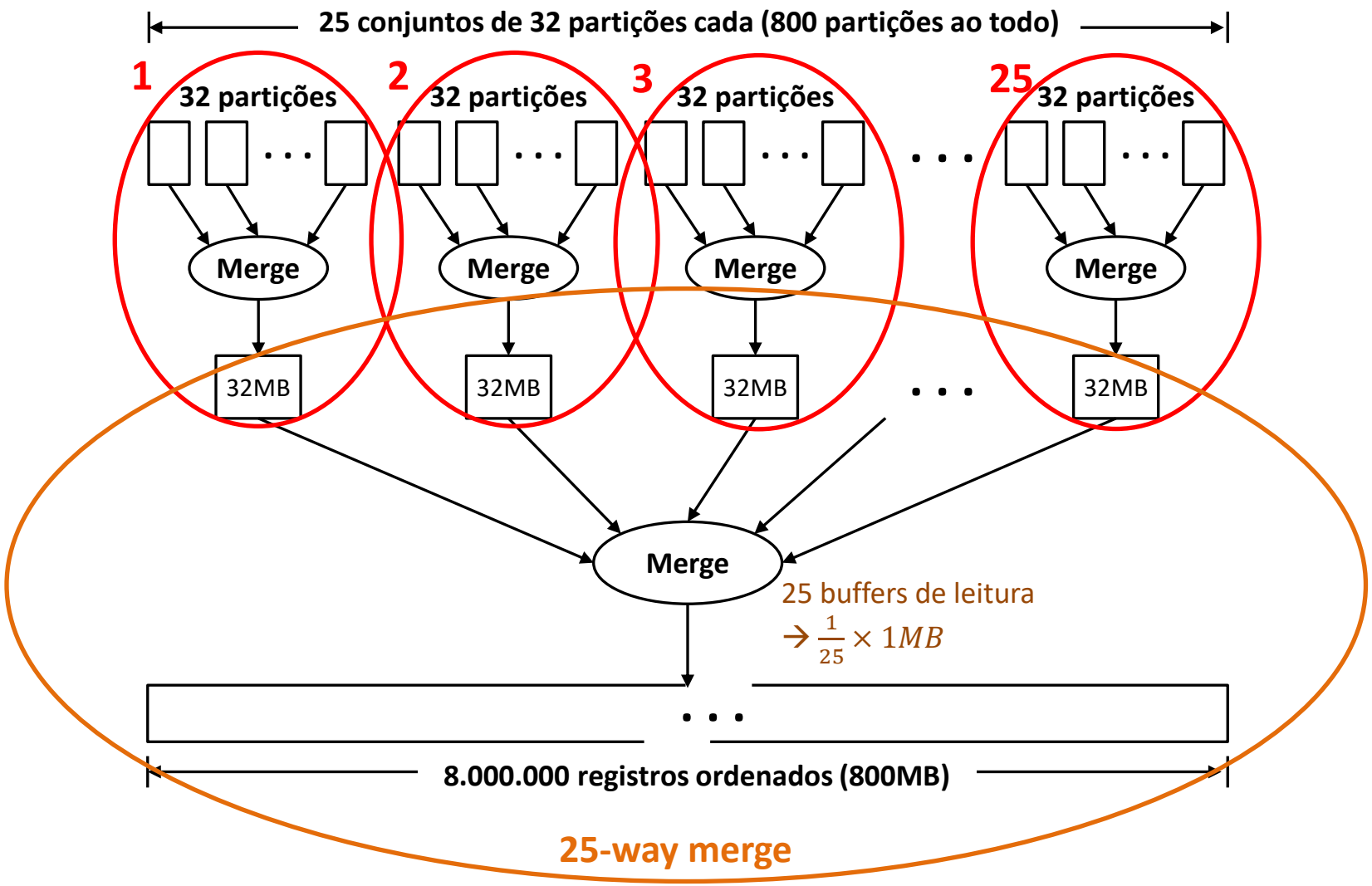
Merge 25 × 32-vias + 25-vias



Merge 25 × 32-vias + 25-vias



Merge 25 × 32-vias + 25-vias



Custo do *merge sort* em múltiplos passos

- Temos menos *seeks* na primeira passada, mas ainda há uma segunda passada → no final, compensa?
- *800-way merge* original: **640.000 *seeks* no passo 3**
- *2-step merging* (25 x 32-way merge + 25-way merge): **?**
  - 1ª passada (Passo 3-1):
    - Cada *merge* de 32-vias aloca *buffers* que podem conter 1/32 de uma partição, então serão realizados 32 *seeks* por partição
      - 32 *seeks* por partição* 32 partições = 1.024 *seeks* por subgrupo
      - 25 subgrupos * 1.024 *seeks* por subgrupo = **25.600 *seeks***
    - Cada partição resultante desse passo terá 320.000 registros (32 * 10.000 registros de 100 bytes), o que equivale a um **arquivo de 32MB**
  - 2ª passada (Passo 3-2):
    - Cada uma das **25 partições de 32MB** poderá alocar 1/25 do *buffer* (40KB). Portanto, cada *buffer* poderá conter 400 registros, o que é igual a 1/800 da partição
    - 32.000.000 bytes/40.000 bytes **= 800 *seeks* por partição**
    - 25 partições * 800 *seeks* por partição = **20.000 *seeks***
  - TOTAL = 25.600 + 20.000 = **45.600 seeks**

Merge sort em múltiplos passos

- Encontramos uma maneira de aumentar o espaço disponível no *buffer* para cada partição
- Trocamos passadas extras sobre os dados por uma diminuição no acesso aleatório
  - Aumenta o tempo de transferência, mas diminui o número de *seeks*
- Se fizermos um *merge* em 3 passos, podemos obter resultados ainda melhores?
  - Talvez não, pois temos que considerar o tempo de transferência dos dados
    - No *merge* original, os dados são transferidos 2 vezes (passos 3 e 4), enquanto no *2-step merge*, os dados são transferidos 4 vezes
  - Também é preciso considerar a escrita extra dos dados
    - 80.000 *seeks* na escrita dos 1º e 2º merge (com *buffer* de 20K)
    - No total, esse *2-step merge* custou **127.200 *seeks***

Merge sort em múltiplos passos

- Observação:
  - Vale notar que as diferenças nos valores das comparações apresentadas nos slides estão exageradas, pois não consideramos o tempo de transferência
  - Se considerarmos esse tempo, o *merge* em múltiplos passos continua ganhando, mas as diferenças em termos de tempo serão menores