DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

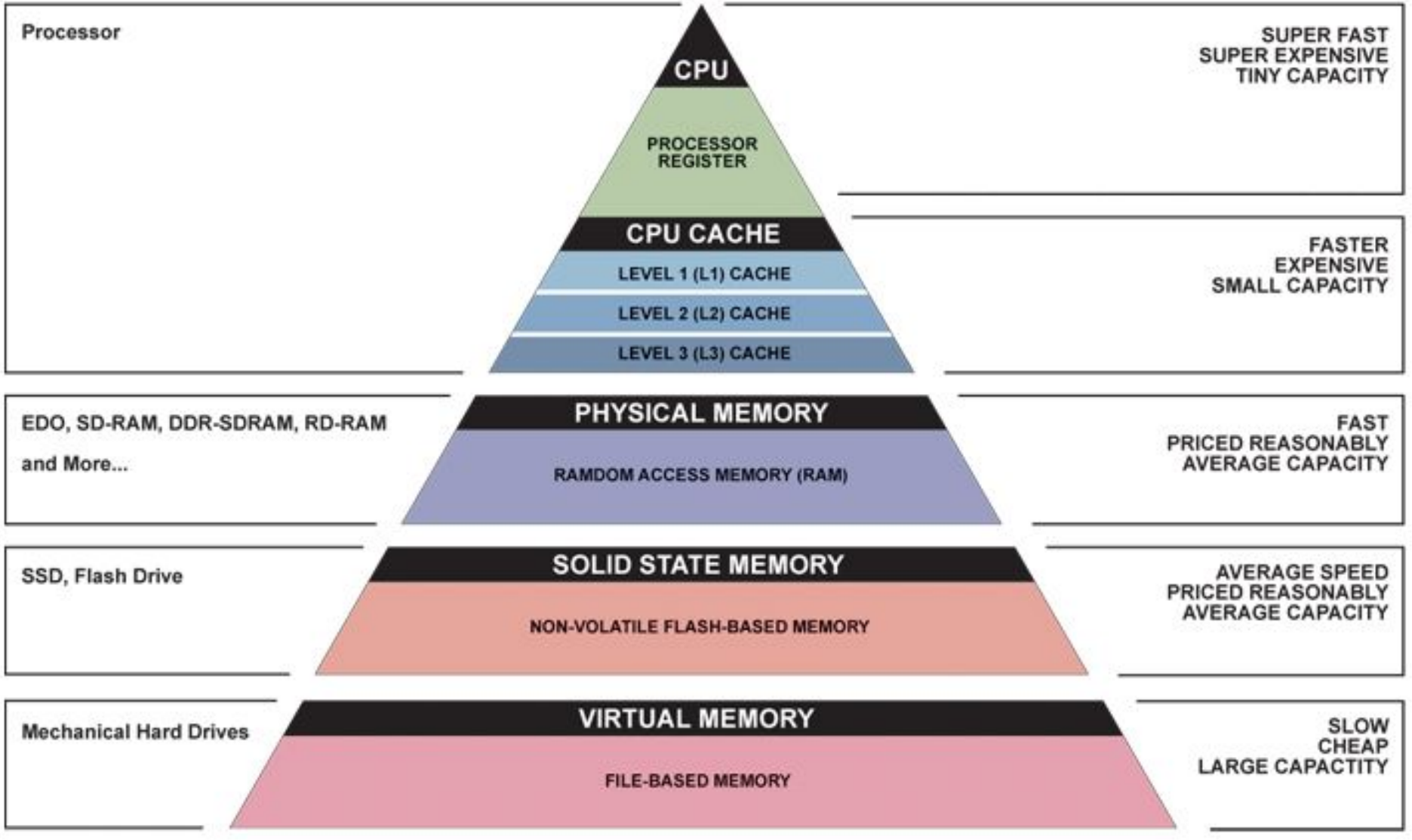
UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Estruturas de Dados

Ordenação em Memória Secundária

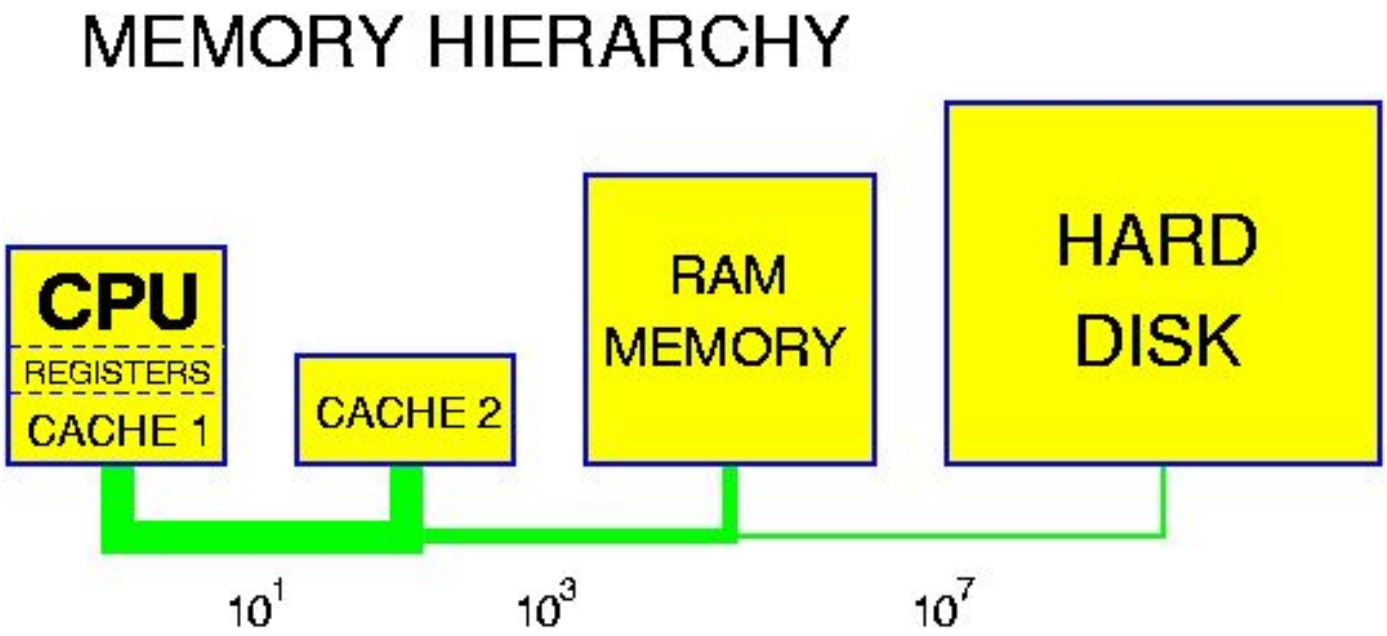
Professores: Anisio Lacerda
Lucas Ferreira
Wagner Meira Jr.
Washington Cunha

Hierarquia de Memória



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Hierarquia de Memória



Número aproximado de ciclos de CPU para acessar diferentes elementos da hierarquia de memória

Memória Interna x Memória Externa

- Os algoritmos vistos até aqui consideram que os dados podem ser armazenados na memória interna (física) do computador
  - Custo dos algoritmos é relacionado com o número de comparações e movimentações

- O que fazer quando a quantidade de dados é maior que o tamanho da memória interna?
  - Como ordenar 1TB usando uma RAM de 8GB?

Memória Interna x Memória Externa

- Nesse caso, devemos **minimizar o número de acessos à memória externa**, que têm um custo muito maior que comparações e movimentações de registros.
- É necessário trabalhar com “blocos” de dados, trazendo-os para a memória principal quando necessário e normalmente o acesso aos dados é sequencial
- Um conceito importante para isso é a **Localidade de Referência**

Analogia - Biblioteca

- Antes de utilizar um livro, você o procura na estante (memória externa), e o carrega até uma mesa (memória interna)
- Custo para achar o livro na estante é maior do que pegá-lo da mesa

“Memória externa”

“Memória interna”

- Vários livros para consultar: carregar um “bloco” de livros de uma vez para a mesa pode ser vantajoso

Localidade de Referência

- Padrões de acesso a dados observados desde os primeiros sistemas de computação:
  - **Temporal**: se um dado é acessado uma vez, há uma probabilidade grande dele ser acessado novamente num futuro próximo.
  - **Espacial**: se um dado é acessado uma vez, há uma probabilidade grande do seu vizinho ser acessado em breve
- A localidade de referência permite que os algoritmos possam trabalhar de forma eficiente utilizando blocos de dados

O Desafio do Armazenamento Secundário

- **Estruturas de dados para armazenamento secundário:**

  Estruturas de Dados para memória primária, projetadas para acesso aleatório à memória, não são eficientes para os padrões de endereçamento físico e acesso sequencial de dispositivos de armazenamento secundário.

- **Endereçamento físico:**

  Os dados são armazenados em blocos de tamanho fixo no dispositivo de armazenamento. Estruturas de dados tradicionais podem exigir a colocação de dados dispersos em vários blocos, impactando a velocidade de acesso.

- **Acesso sequencial:**

  A recuperação de dados é mais rápida quando os blocos são contíguos. O armazenamento secundário é otimizado para acesso sequencial de dados, enquanto a RAM permite acesso aleatório.

Algoritmos para Memória Externa

- Vários dos algoritmos são variações dos algoritmos para memória interna

- **Ordenação**
  - Intercalação Balanceada / Polifásica
  - Seleção por Substituição
  - Quicksort Externo

- Pesquisa
  - Acesso Sequencial Indexado
  - Árvores B
  - Árvores B*

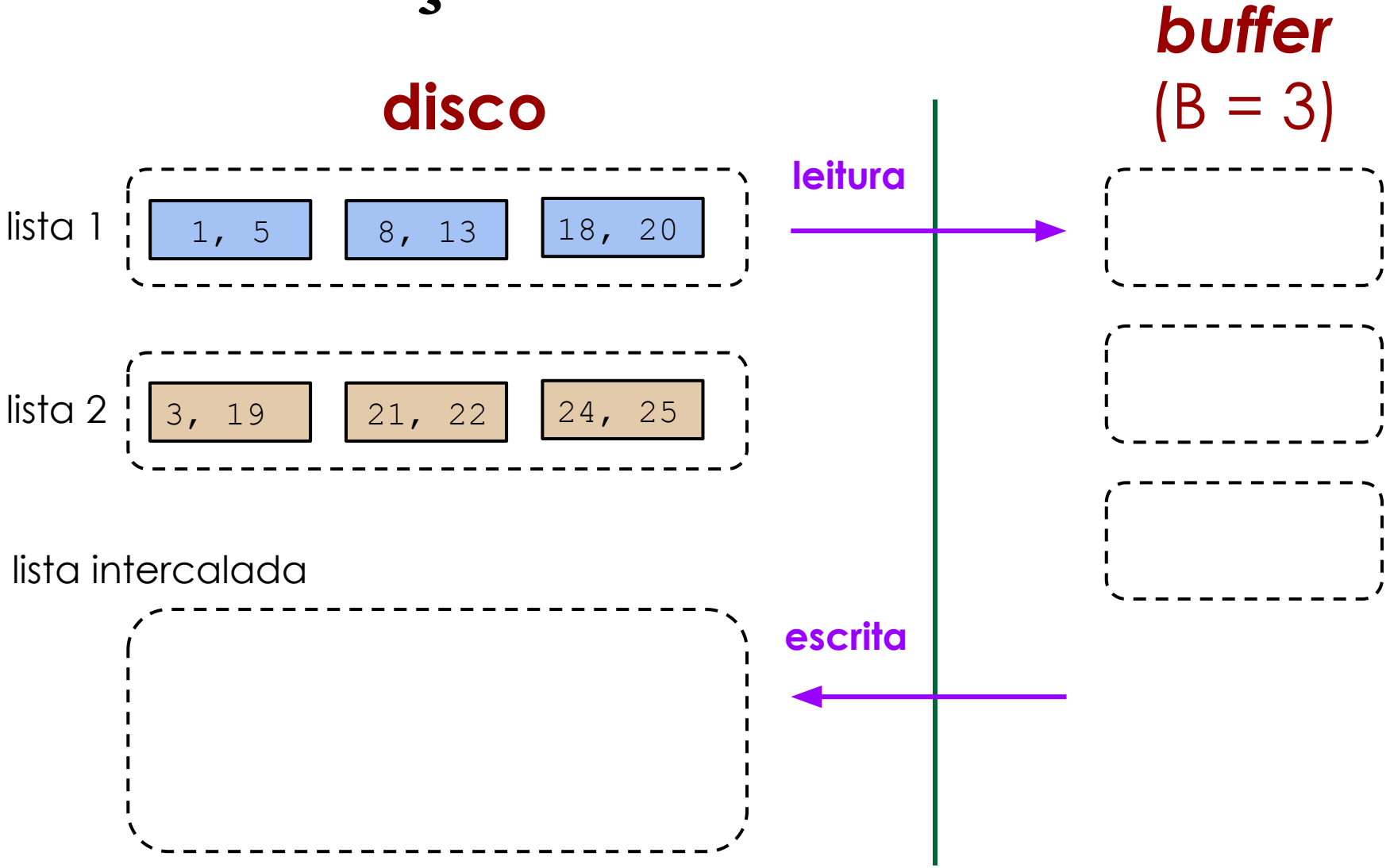
Intercalação externa

Intercalação Externa

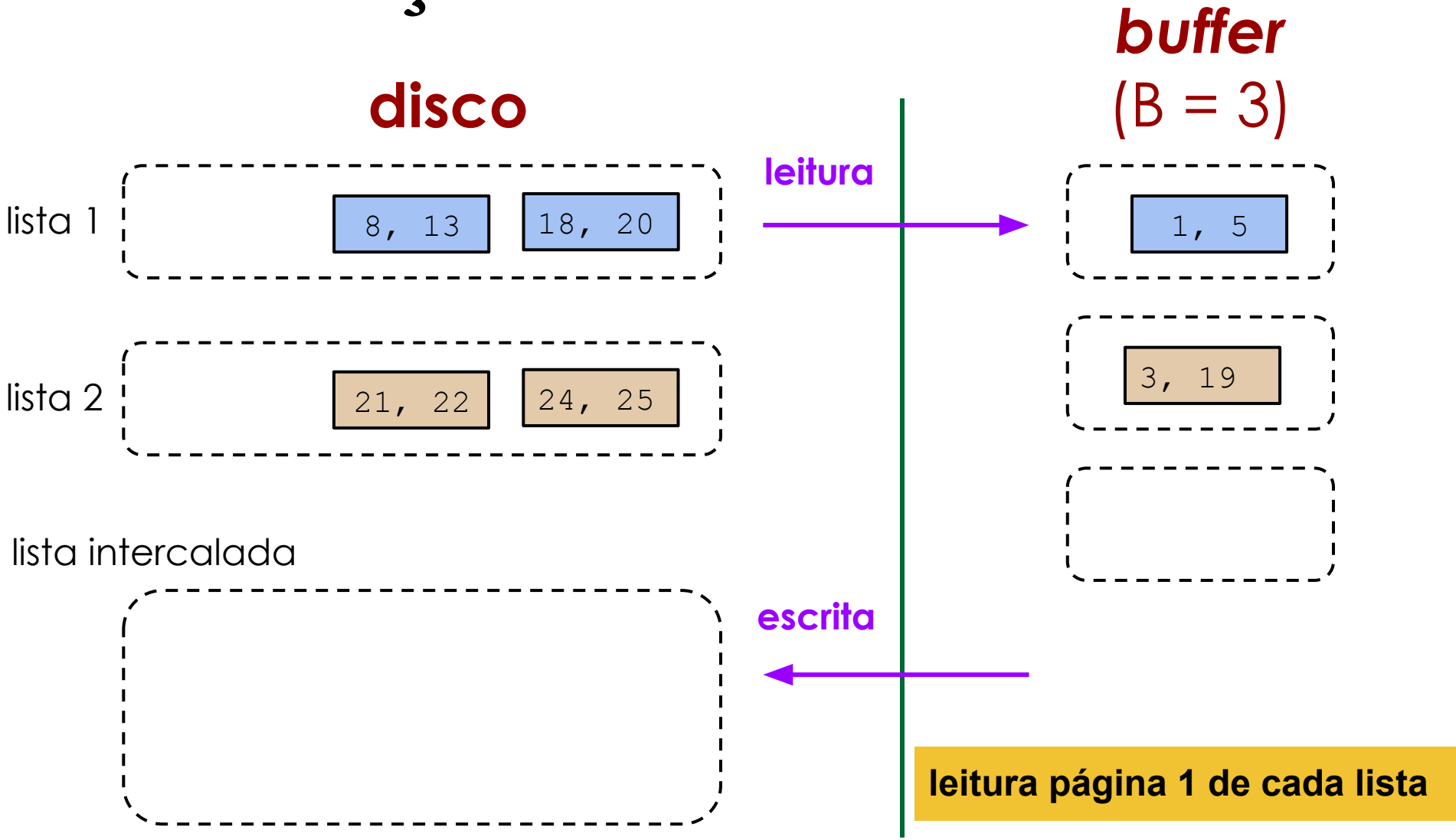
ENTRADA: 2 listas ordenadas (M e N)
SAÍDA: 1 lista ordenada (*merged*) (M+N)

Podemos intercalar eficientemente (i.e., I/O) as duas listas usando um *buffer* de tamanho pelo menos 3?

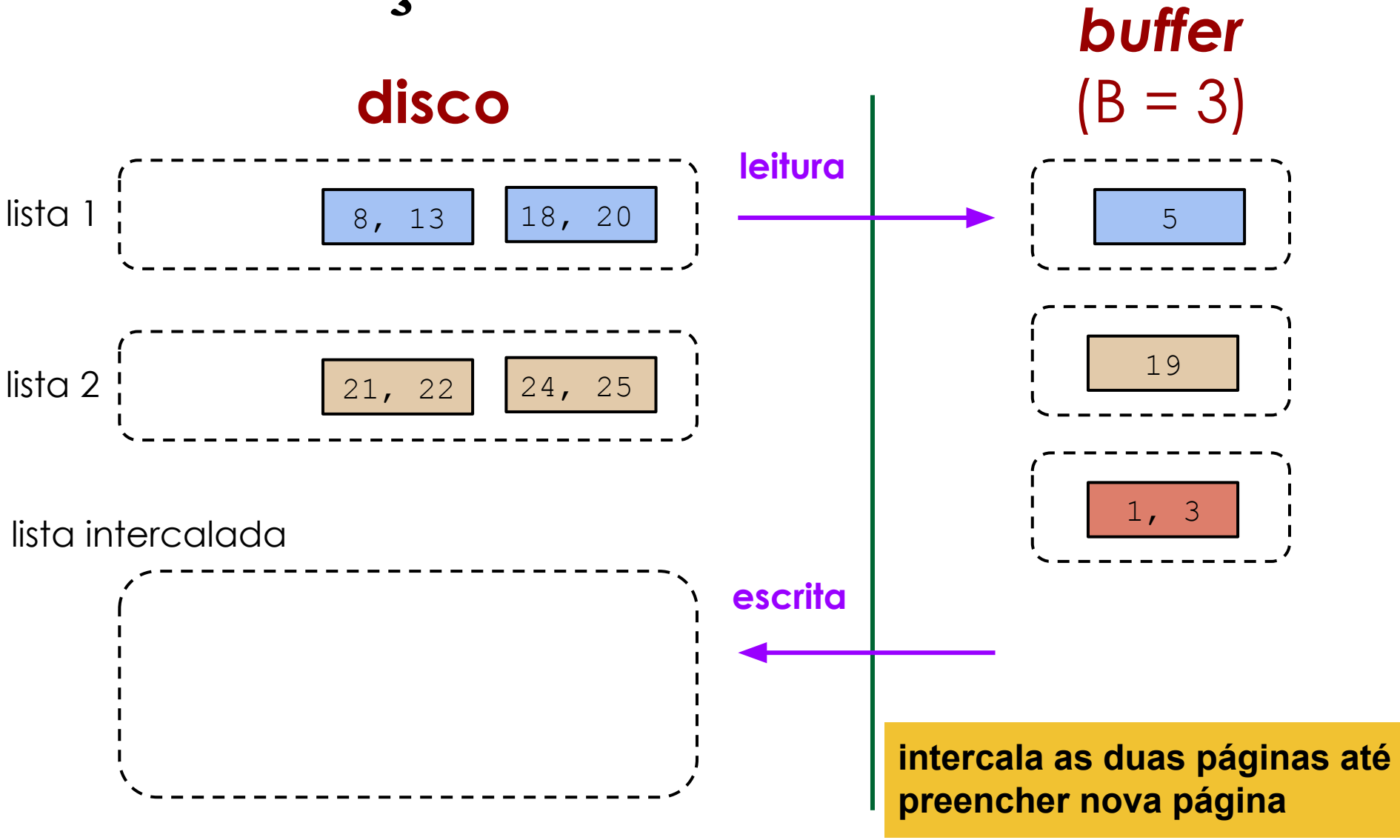
Intercalação externa



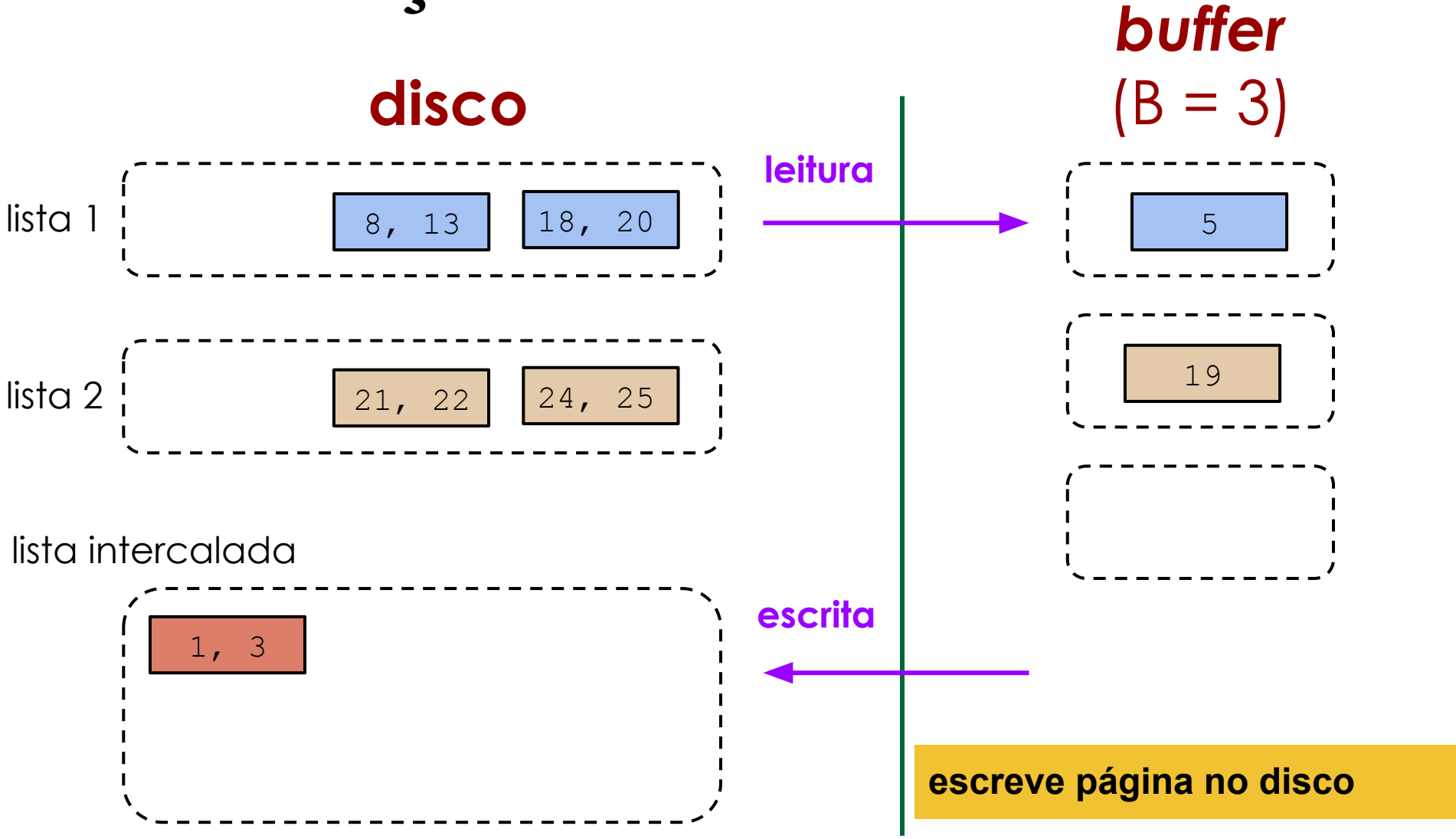
Intercalação externa



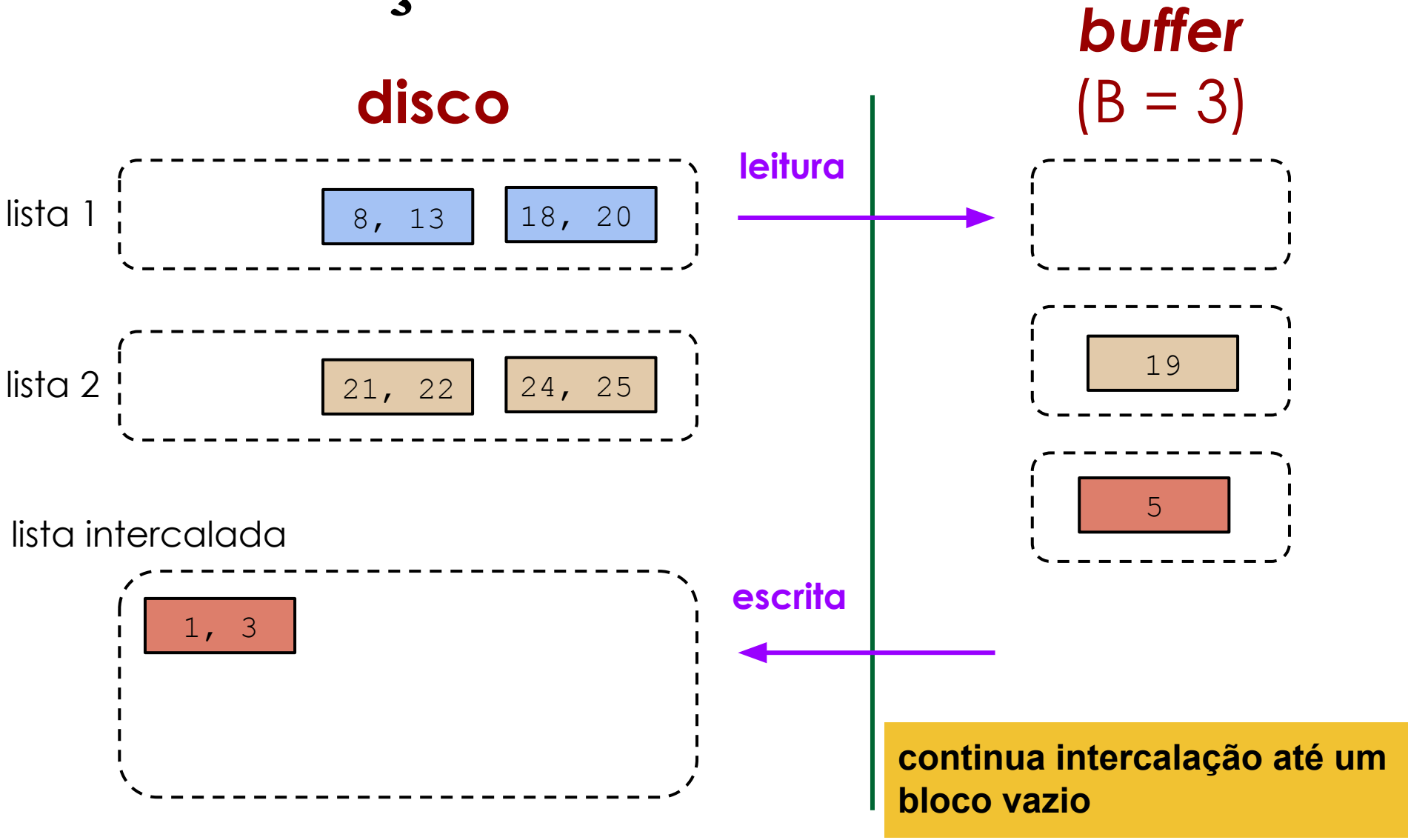
Intercalação externa



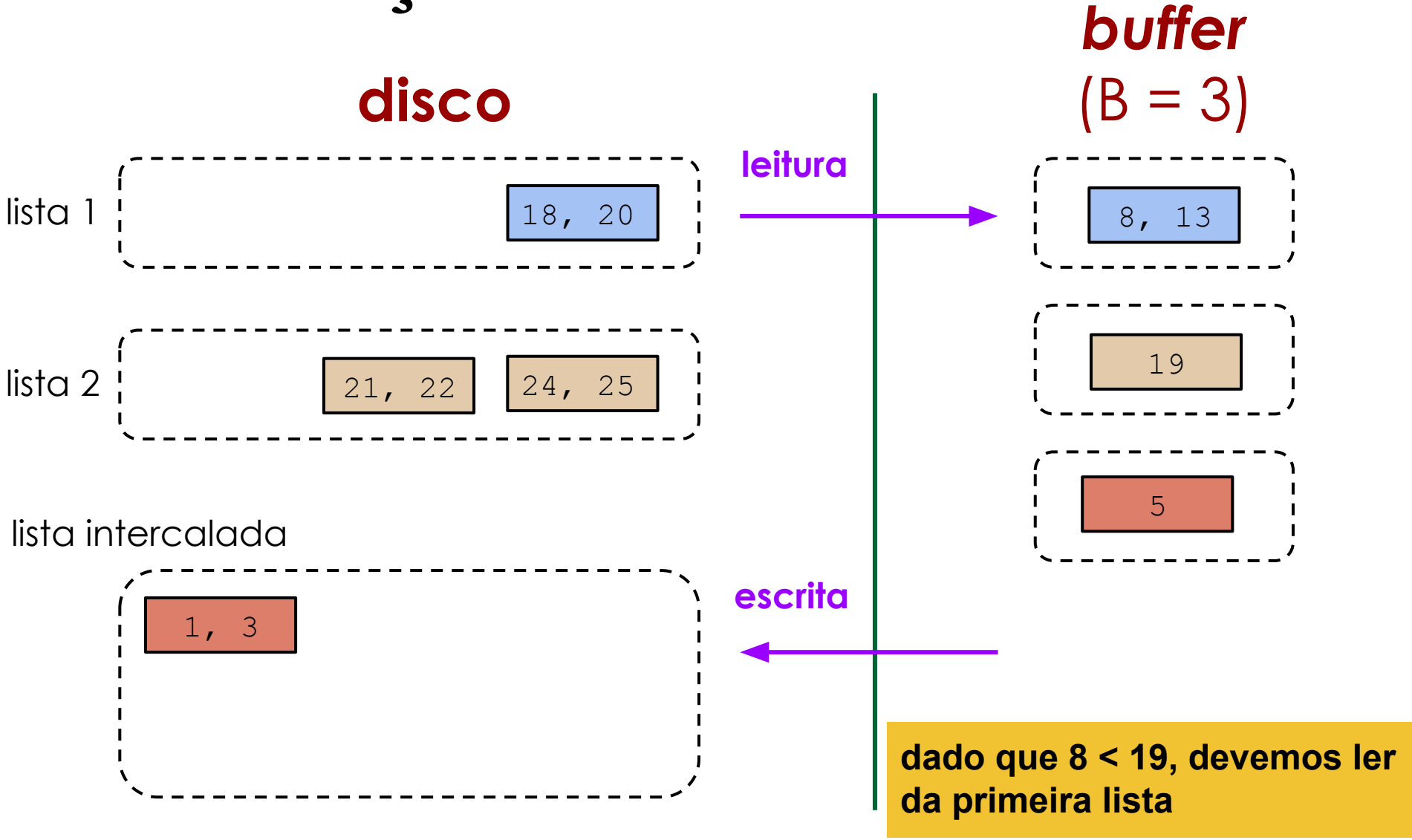
Intercalação externa



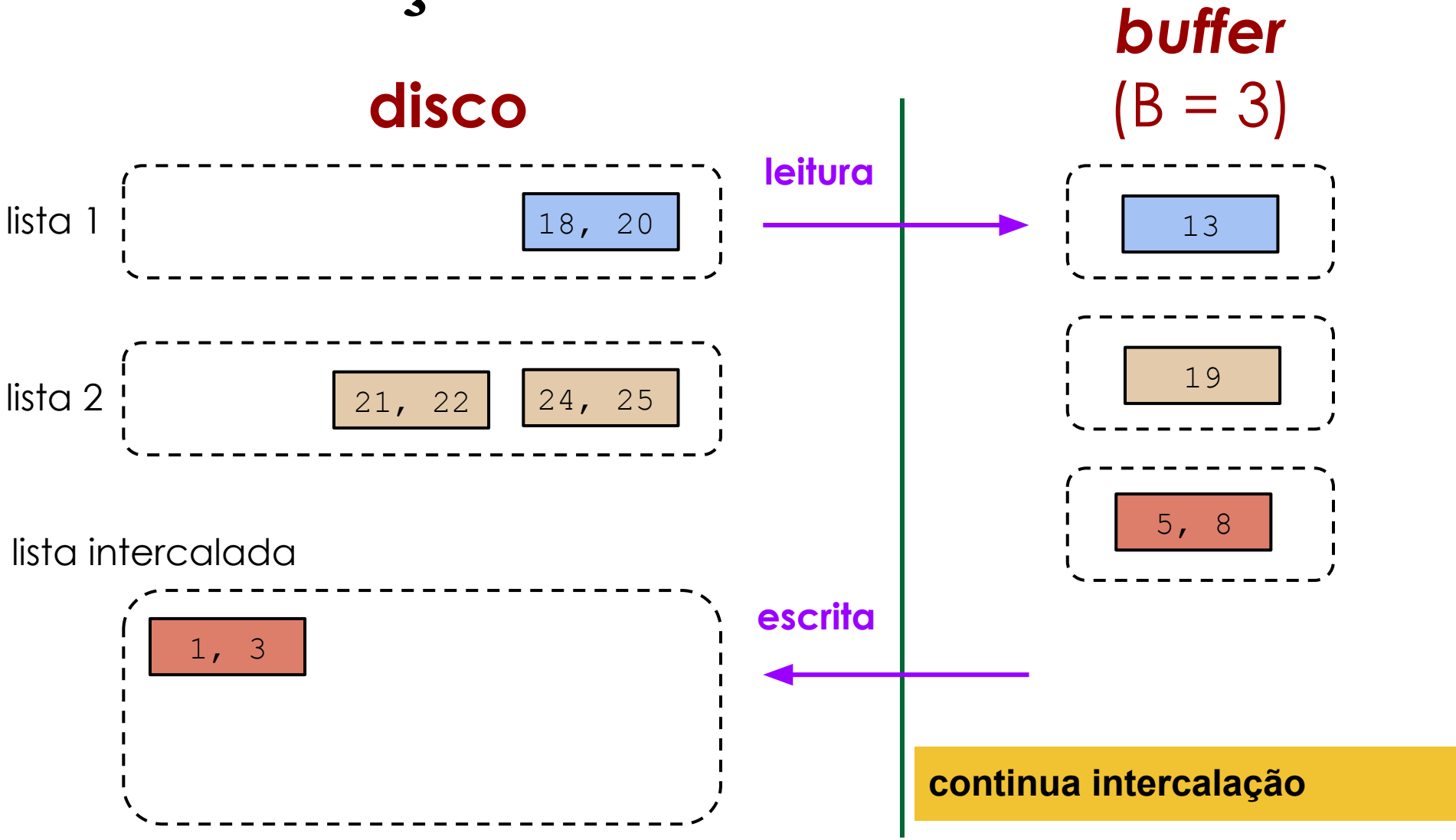
Intercalação externa



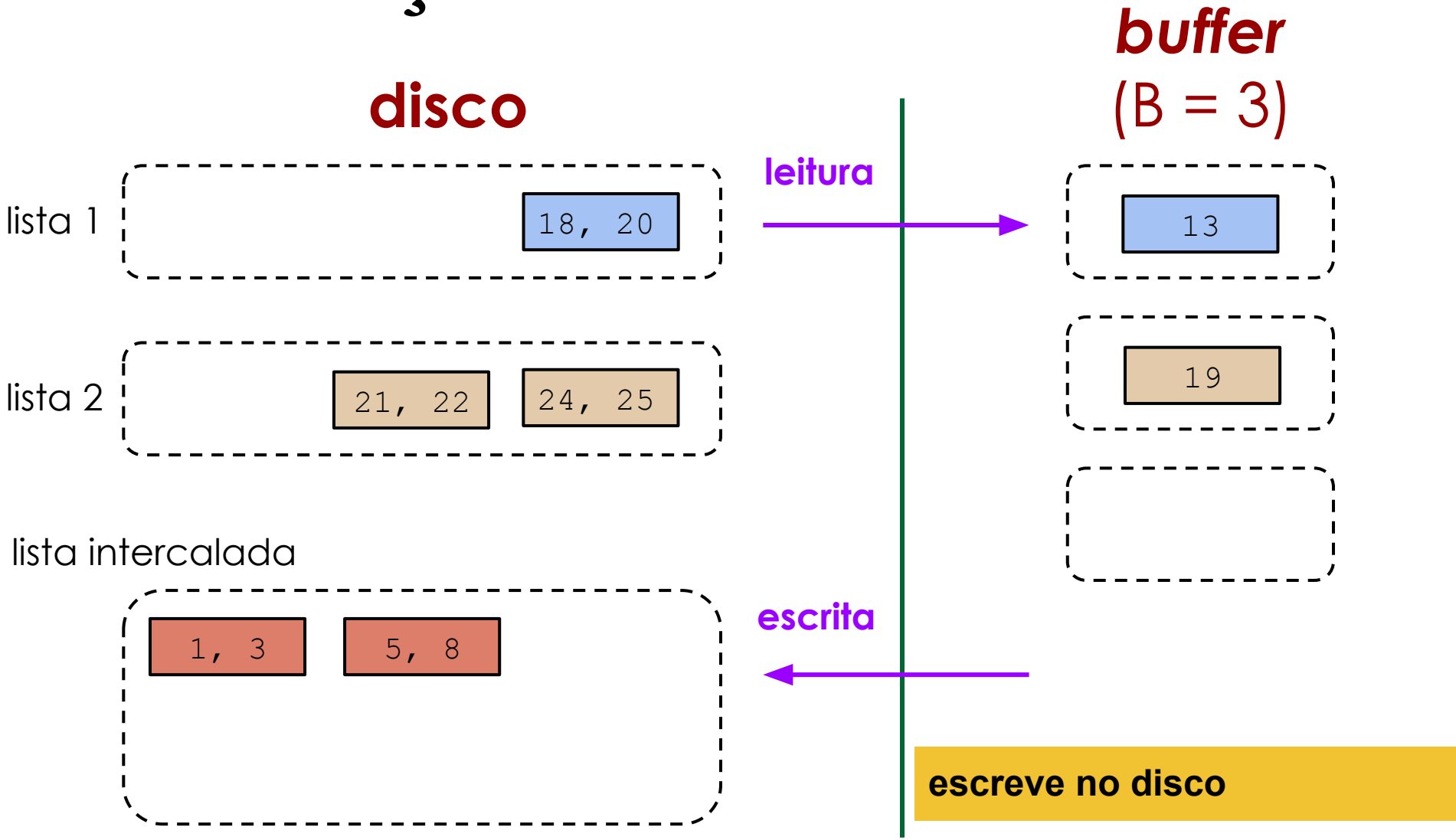
Intercalação externa



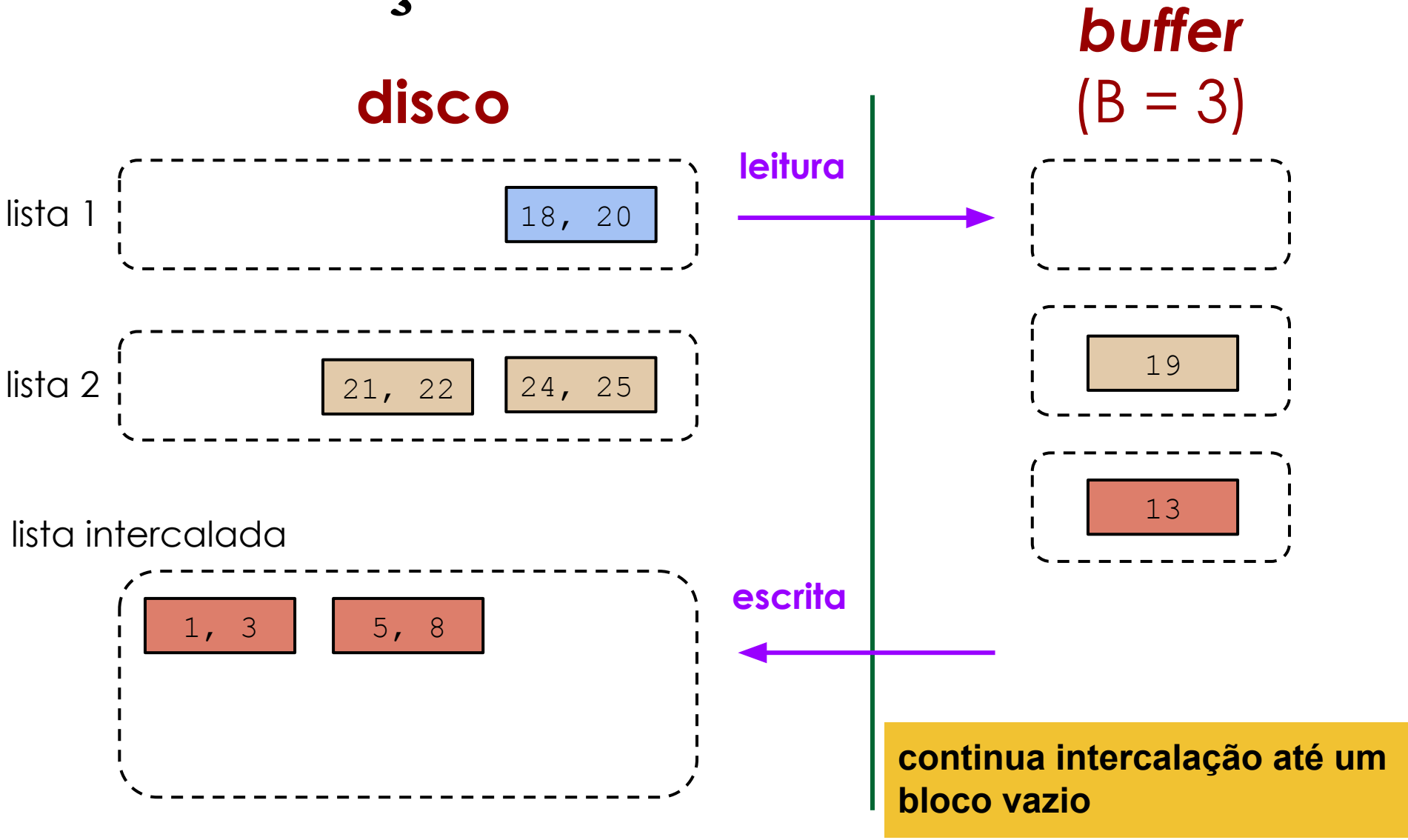
Intercalação externa



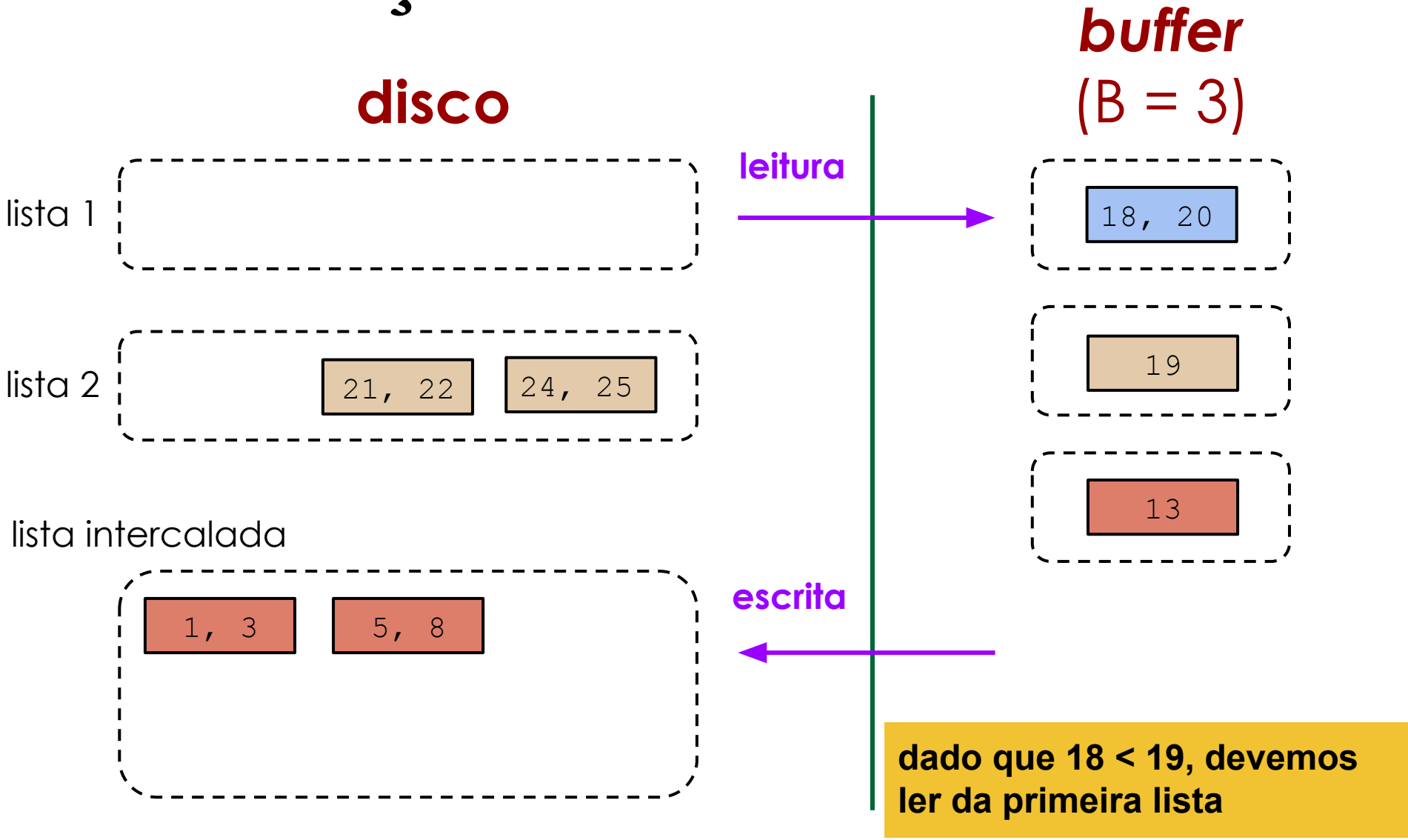
Intercalação externa



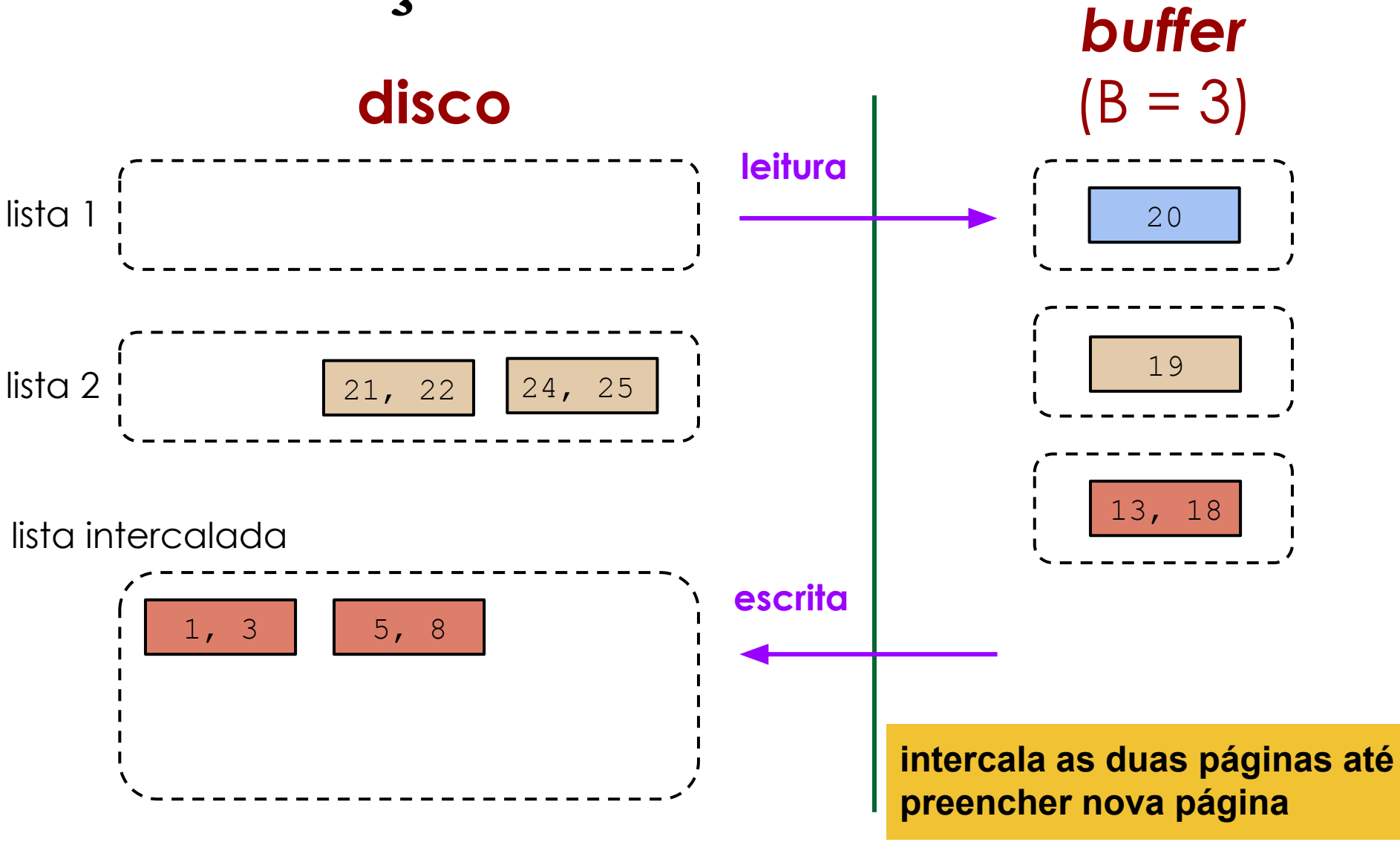
Intercalação externa



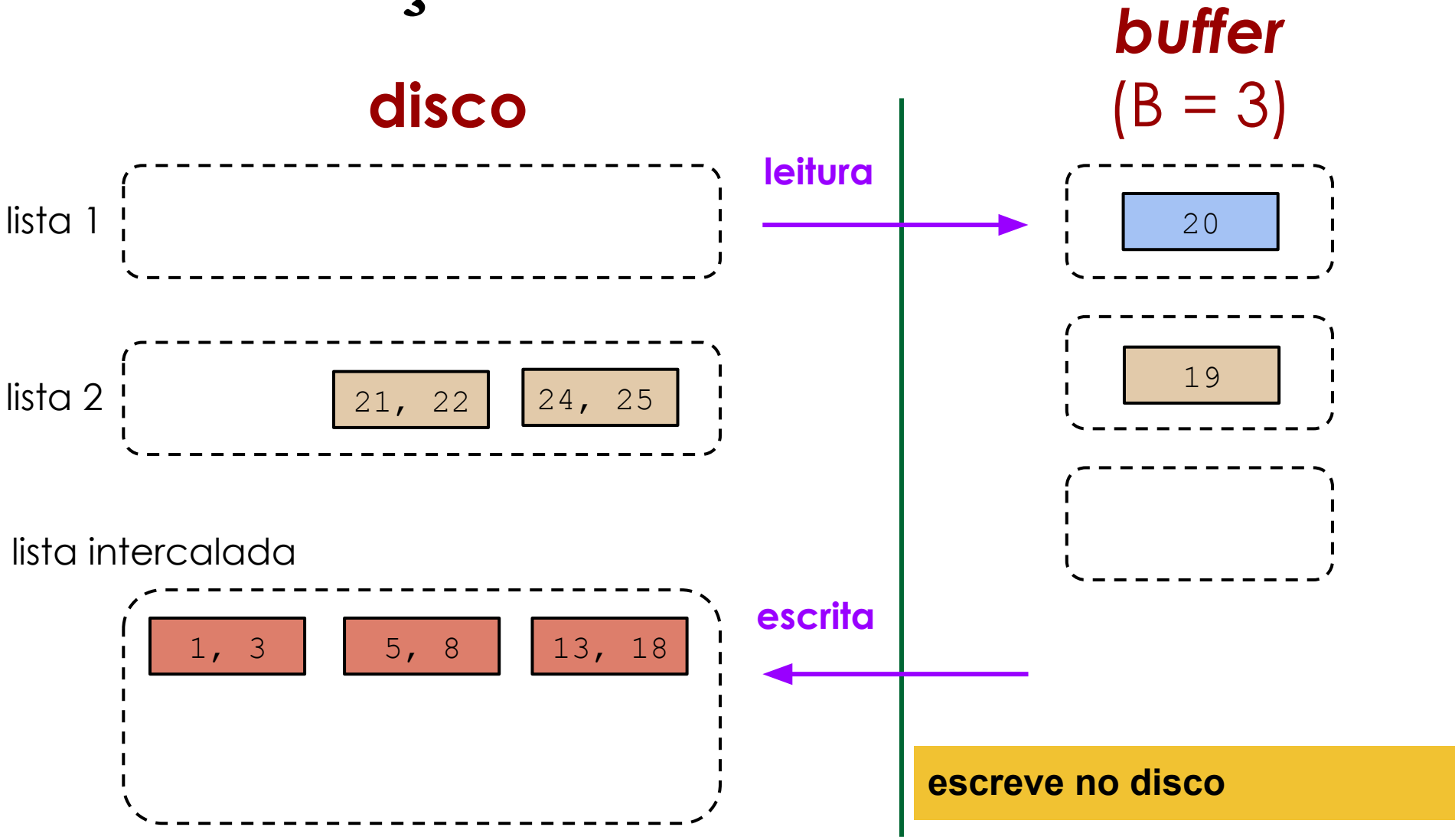
Intercalação externa



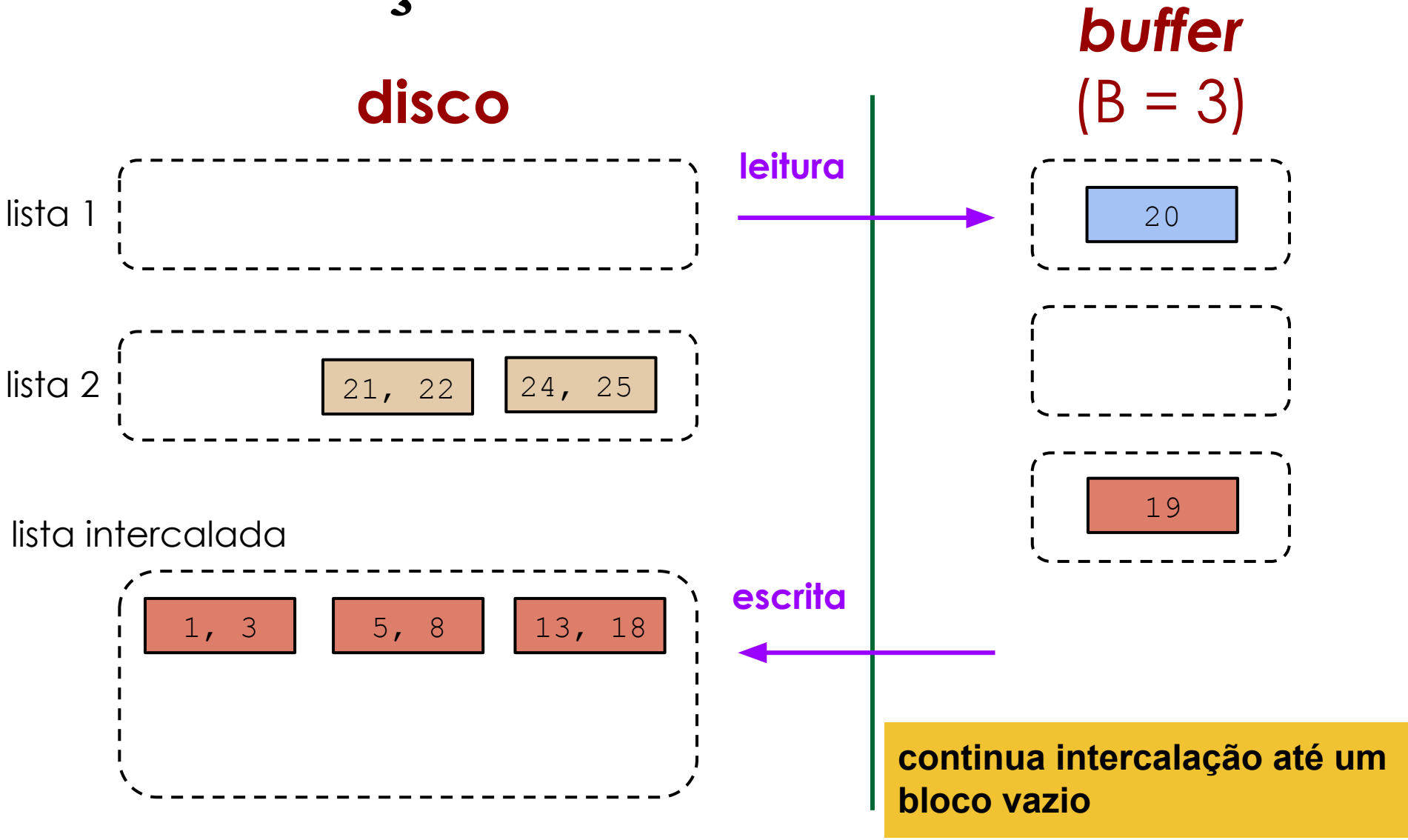
Intercalação externa



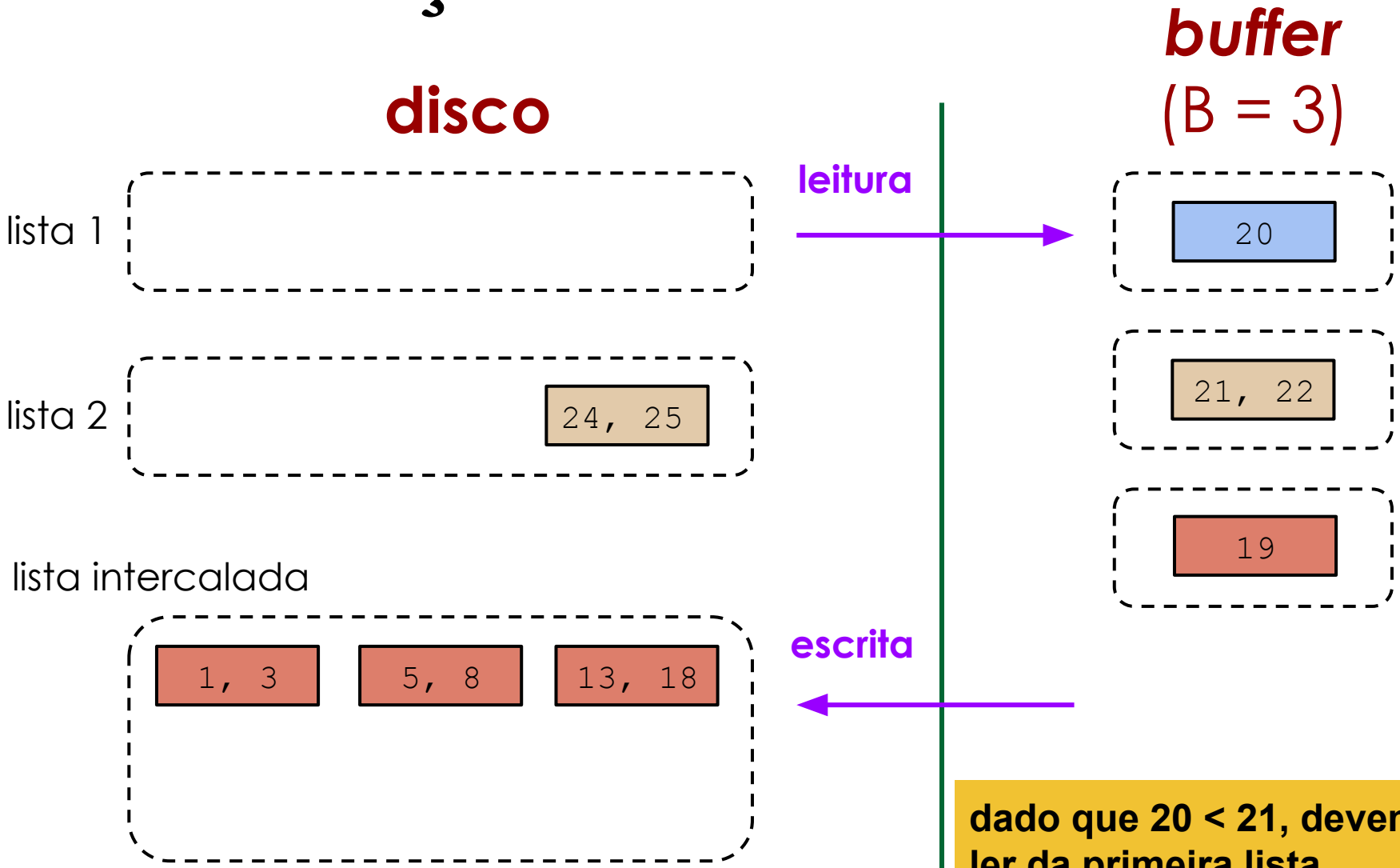
Intercalação externa



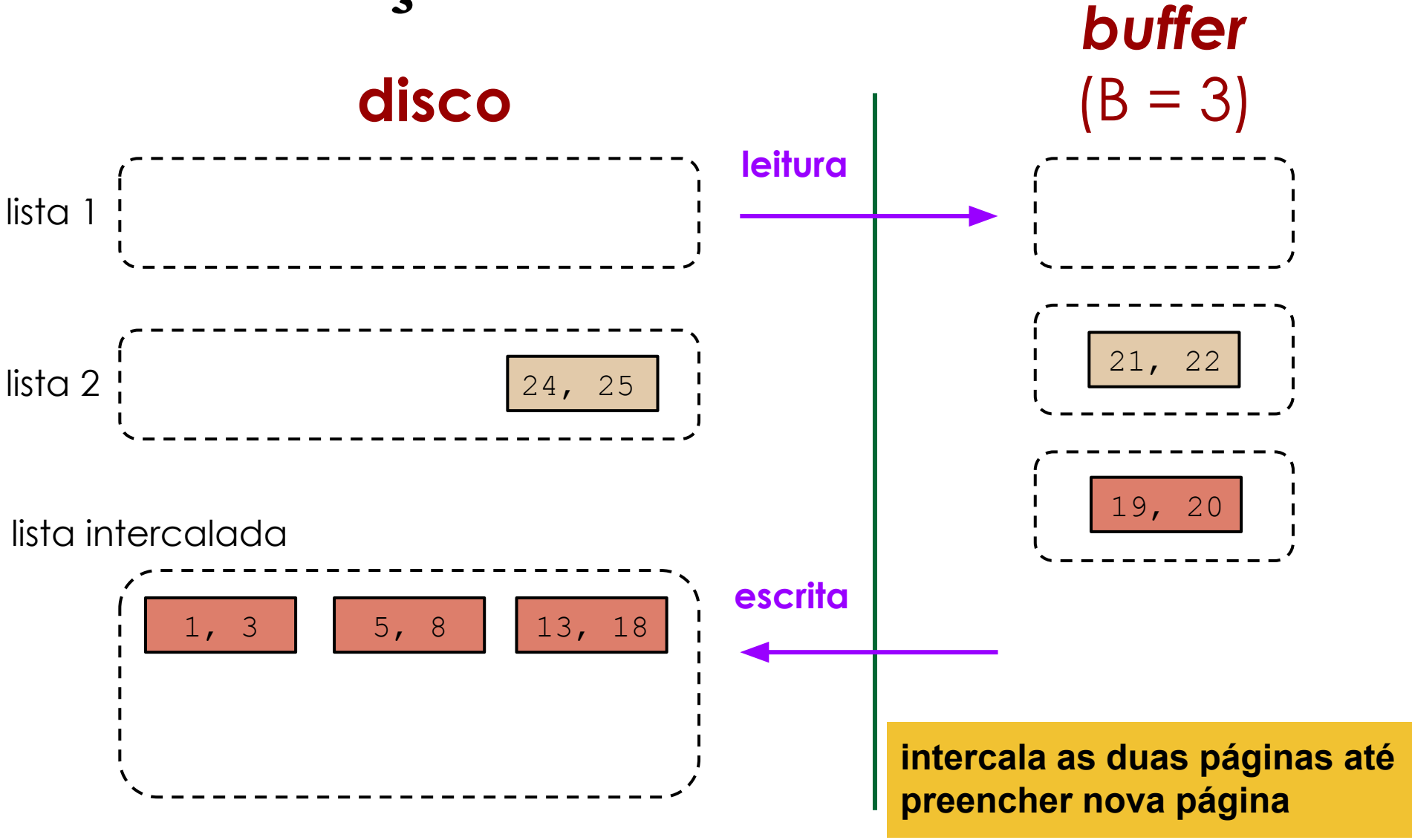
Intercalação externa



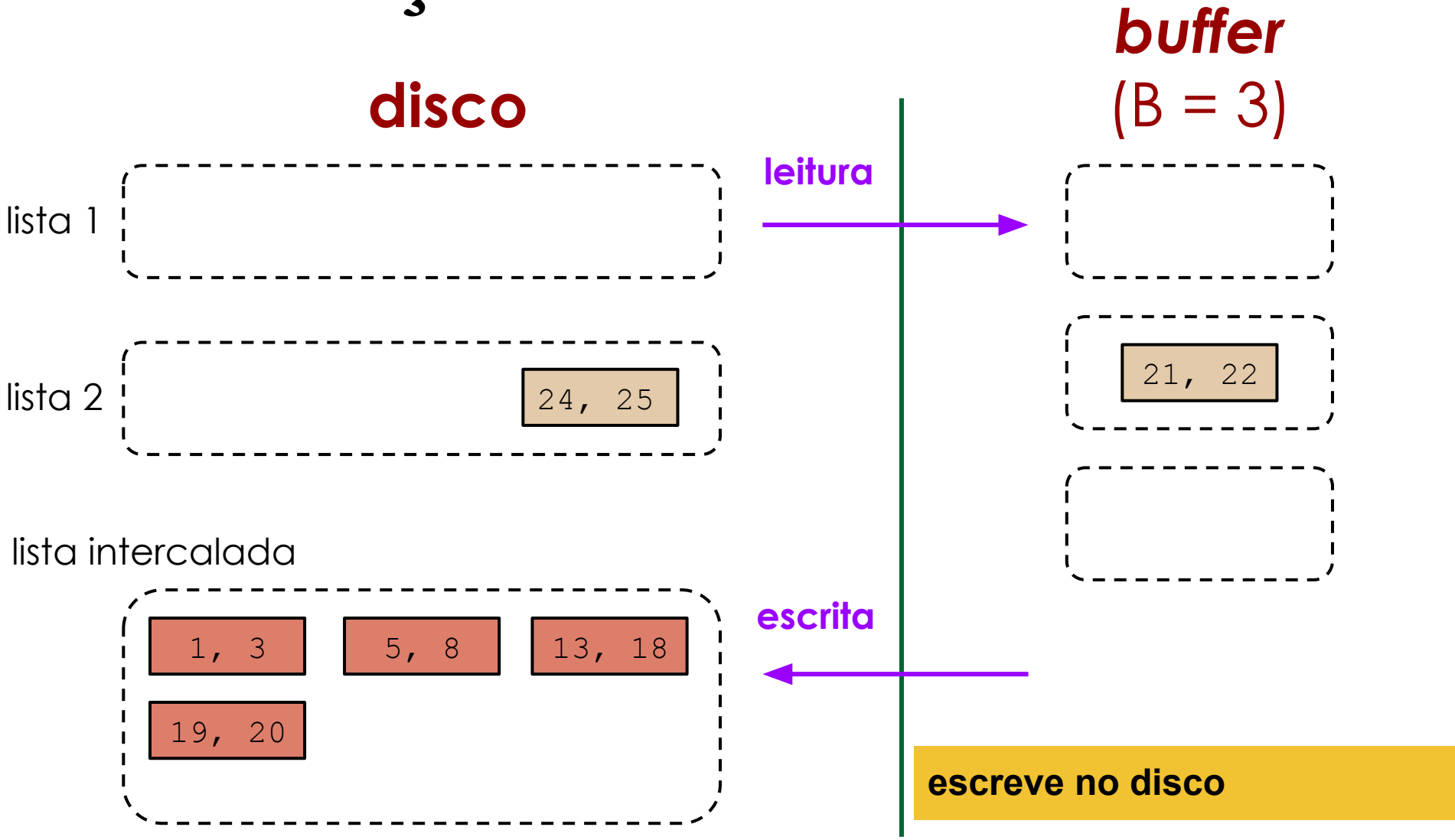
Intercalação externa



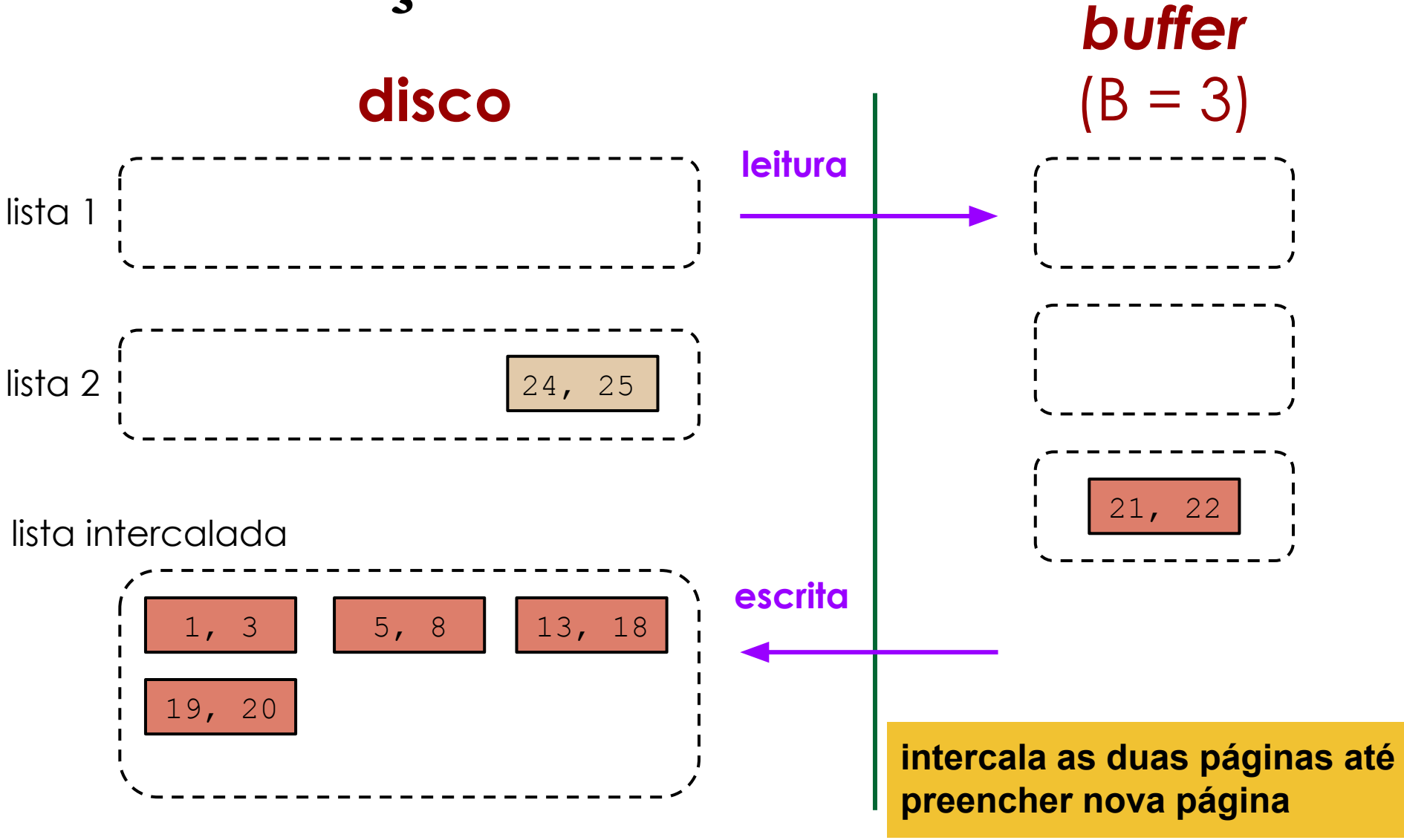
Intercalação externa



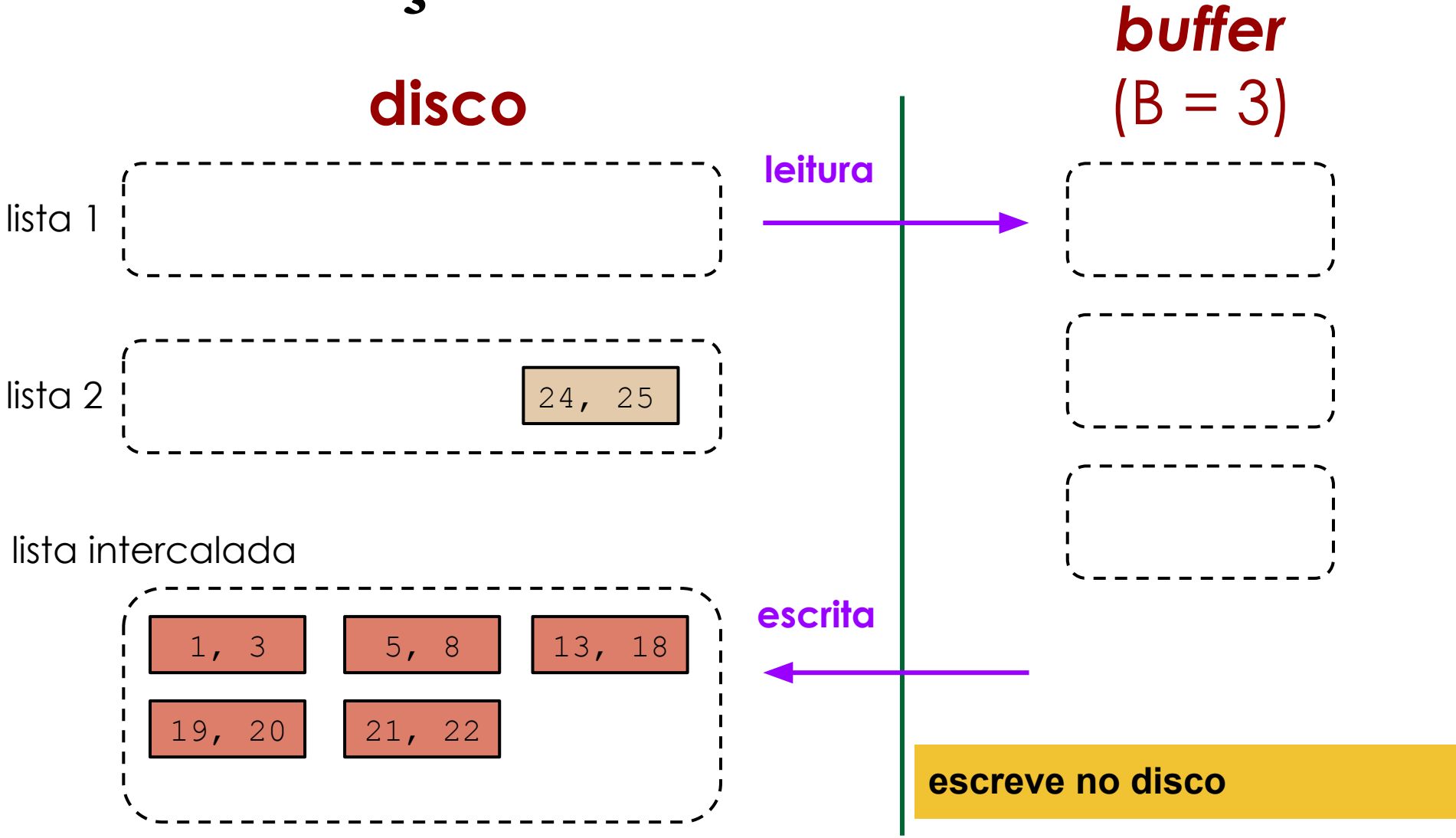
Intercalação externa



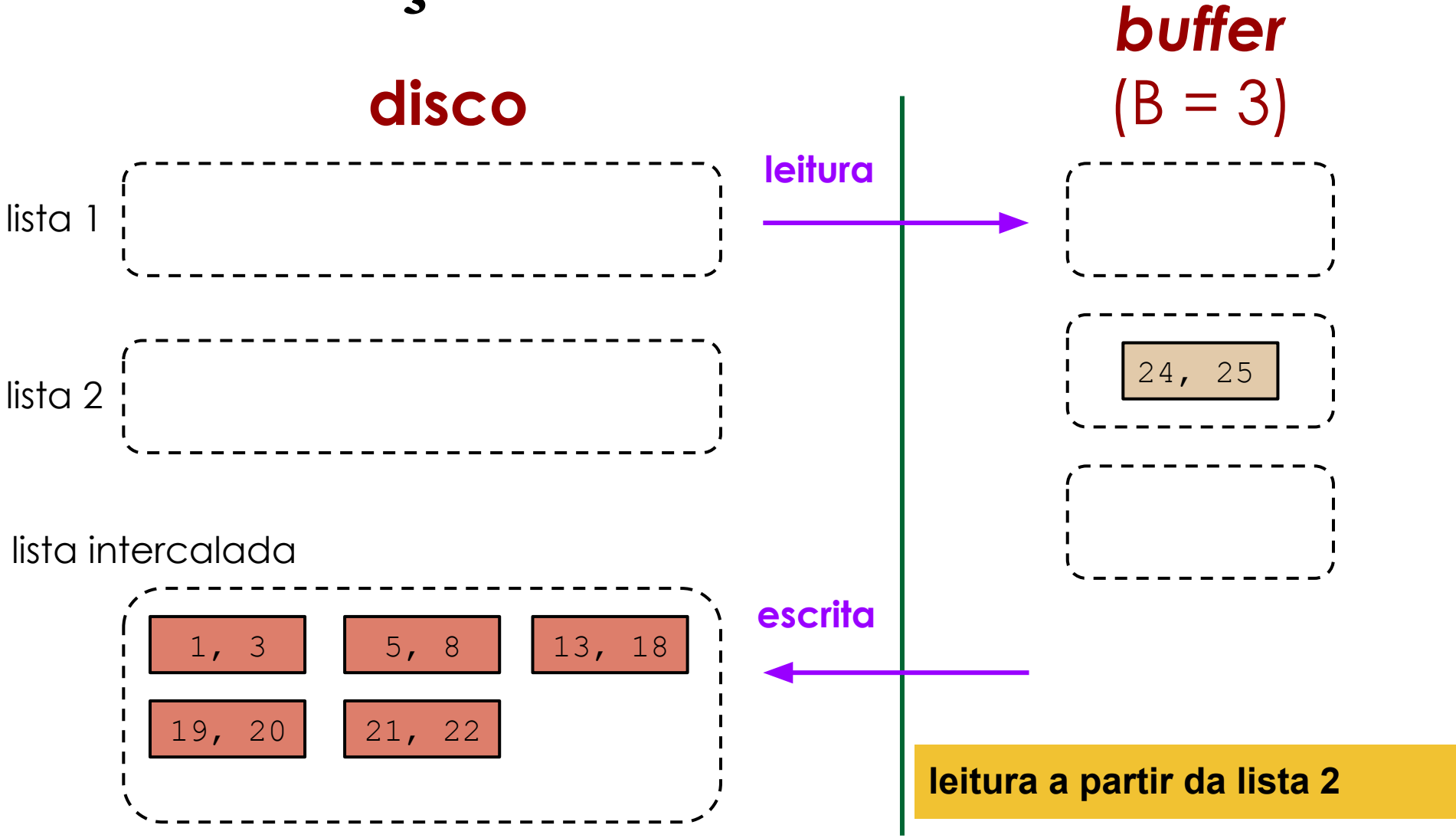
Intercalação externa



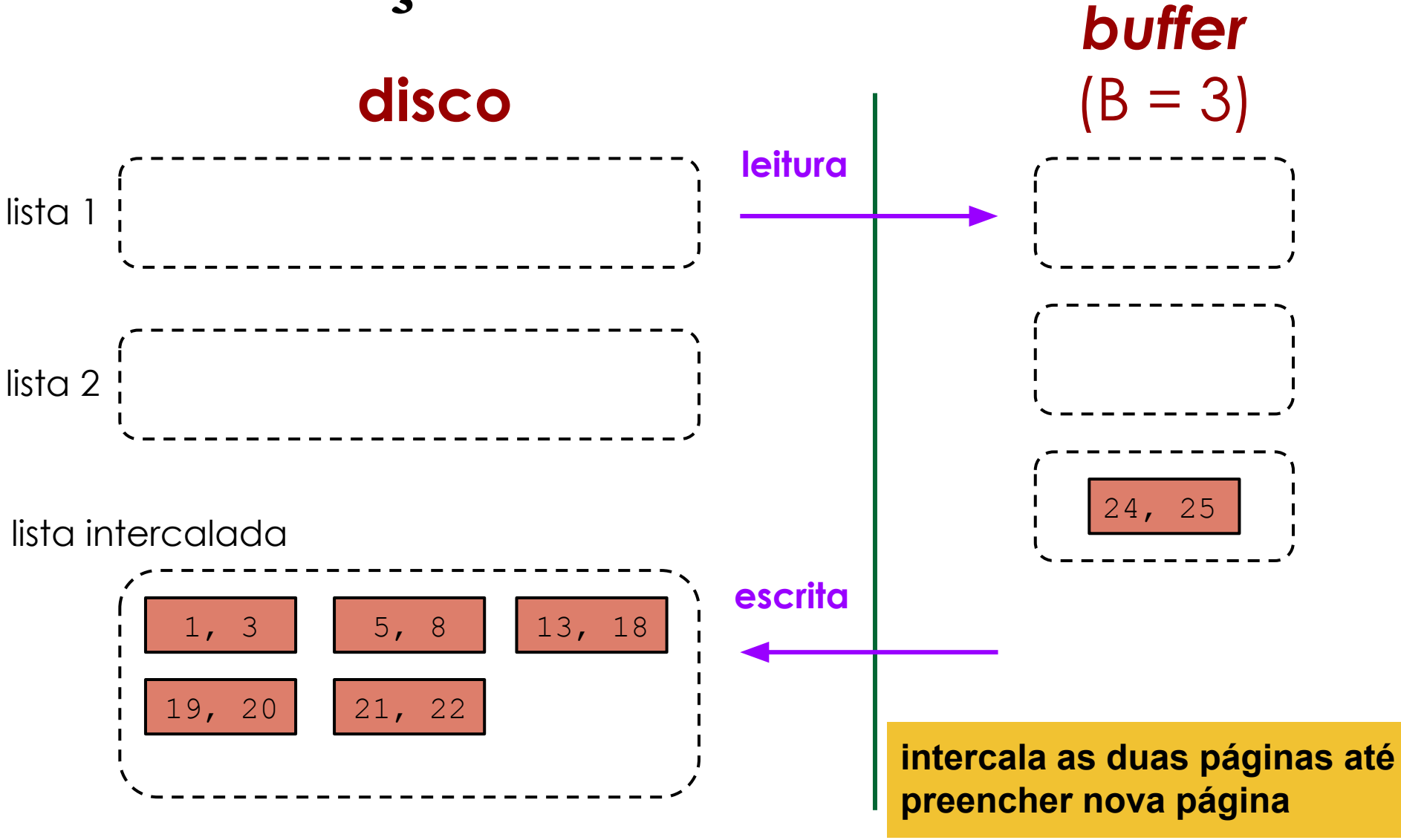
Intercalação externa



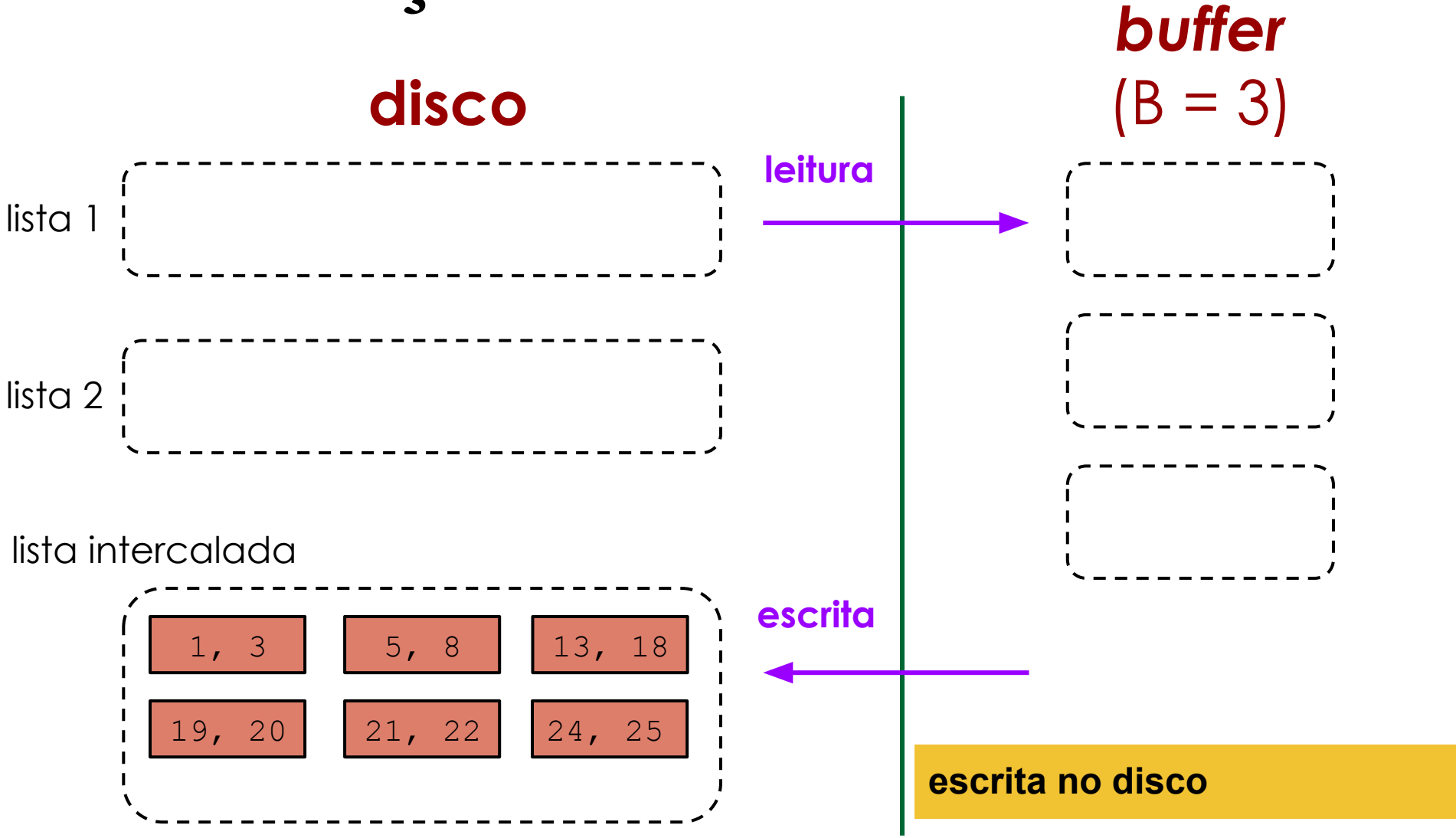
Intercalação externa



Intercalação externa



Intercalação externa



Intercalação Externa

ENTRADA: 2 listas ordenadas (M e N)
SAÍDA: 1 lista ordenada (*merged*) (M+N)

Podemos intercalar eficientemente (i.e., I/O) as duas listas usando um *buffer* de tamanho pelo menos 3?

SIM, custo de I/O: 2 (M + N)

cada página é lida/escrita uma vez

Intercalação Externa

ENTRADA: 2 listas ordenadas (M e N)
SAÍDA: 1 lista ordenada (*merged*) (M+N)

Podemos intercalar eficientemente (i.e., I/O) as duas listas usando um *buffer* de tamanho pelo menos 3?

SIM, custo de I/O: 2 (M + N)

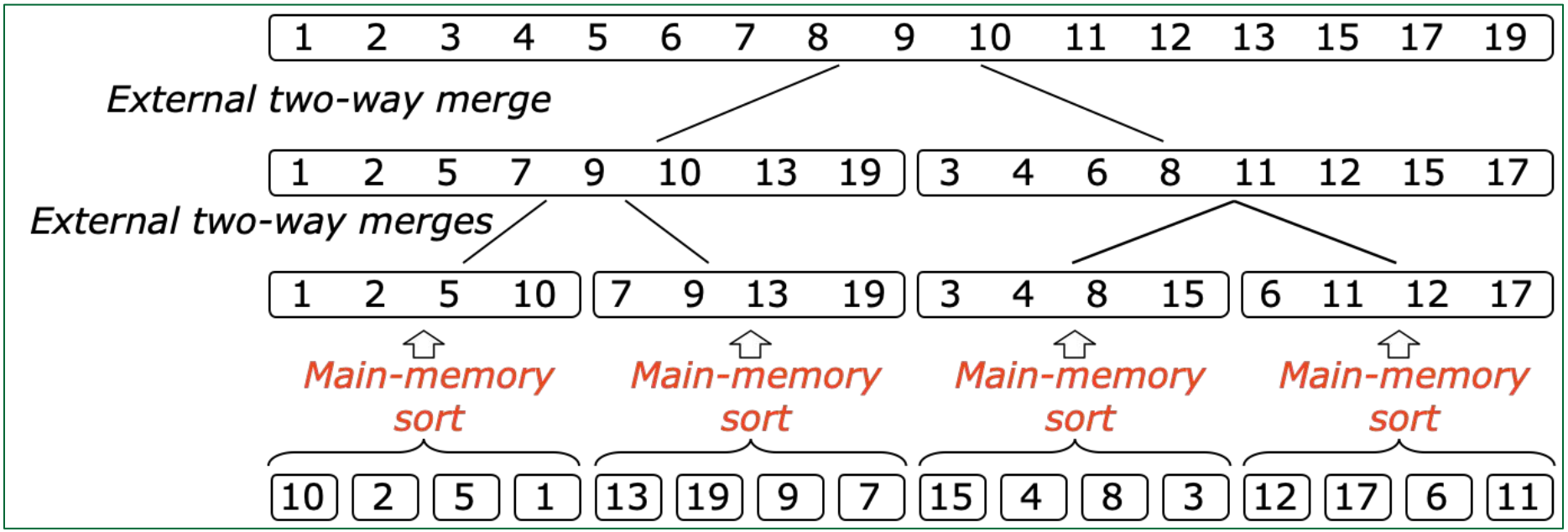
cada página é lida/escrita uma vez

Se tivermos *B+1* páginas no buffer, podemos intercalar *B* listas com o mesmo custo de I/O

Mergesort externo

Mergesort externo

Aumente o tamanho das *runs* de entrada



Mergesort externo

Entrada:

arquivo a ser ordenado X

arquivo vazio Y

Fase 1: repete até o fim do arquivo X

leia M registros de X

ordene em memória

escreva no fim do arquivo Y

Fase 2: repete enquanto existir mais de uma *run*

Apaga X

MergeAllRuns(Y, X)

X -> Y, Y -> X

Mergesort externo

MergeAllRuns(X, Y): repete até o fim de Y

chama *TwoWayMerge* para intercalar duas *runs* sequenciais de Y em uma *run*, que é escrita no fim de X

TwoWayMerge

usa 3 buckets

Ideia

Dividir o vetor de entrada em *chunks* que caibam em memória (*runs*)

Intercalar grupos de *runs* usando o algoritmo intercalação externa

Continuar a intercalação das *runs* resultantes em passos (*pass*) até termos o arquivo de entrada ordenado

WARM UP: 2-*way sort*

buffer

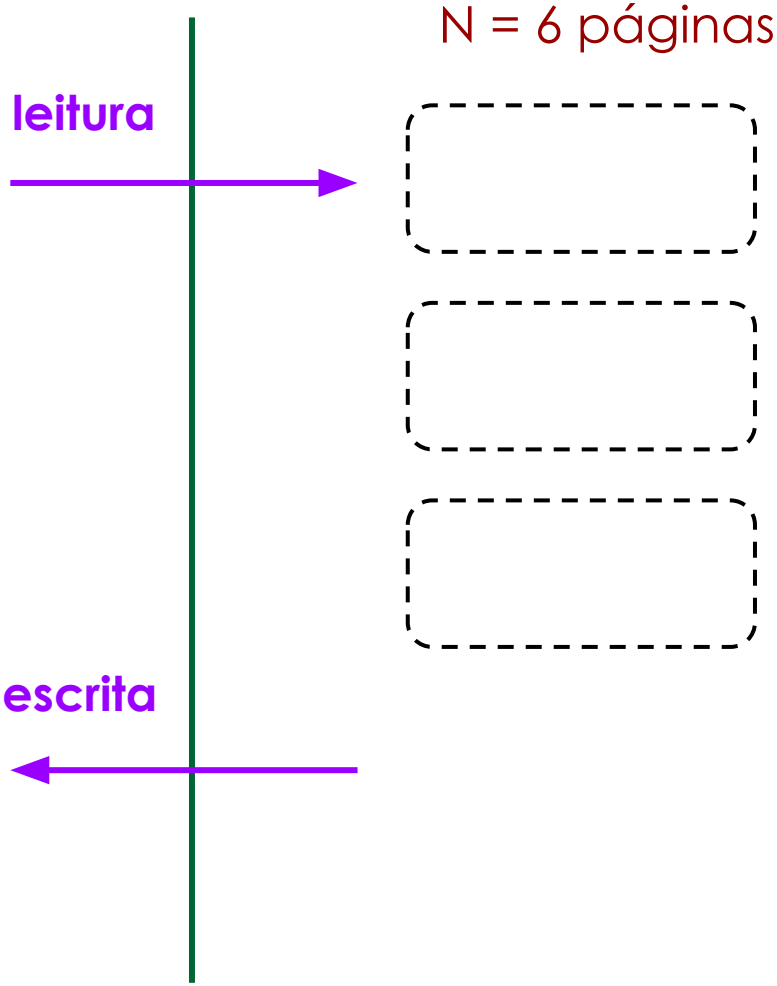
B = 3 *frames*
N = 6 páginas

disco

arquivo de entrada **não-ordenado**

40, 3, 8, 34, 23, 12,

2, 13, 5, 17, 25, 15,

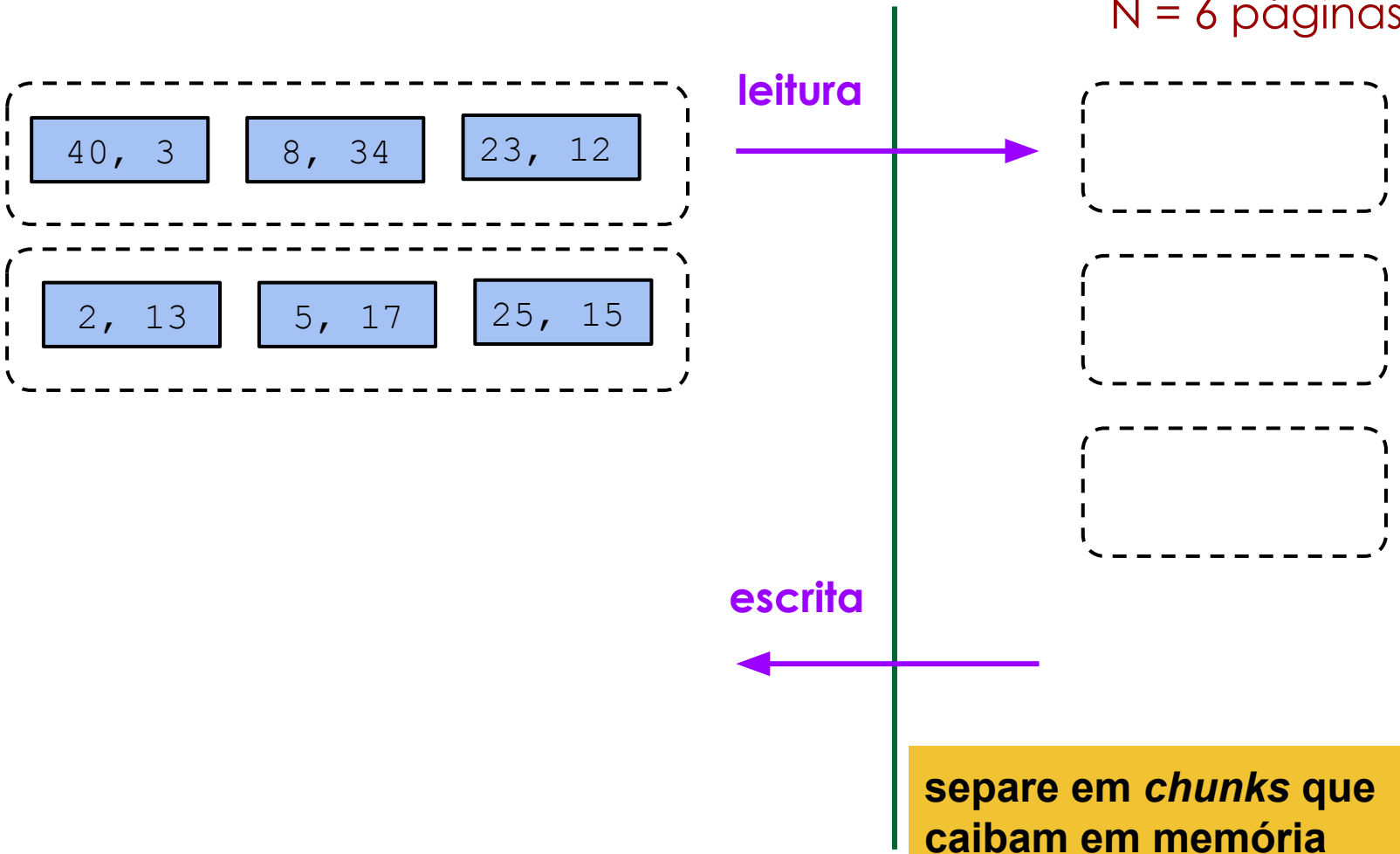


WARM UP: 2-*way sort*

buffer

B = 3 *frames*
N = 6 páginas

disco



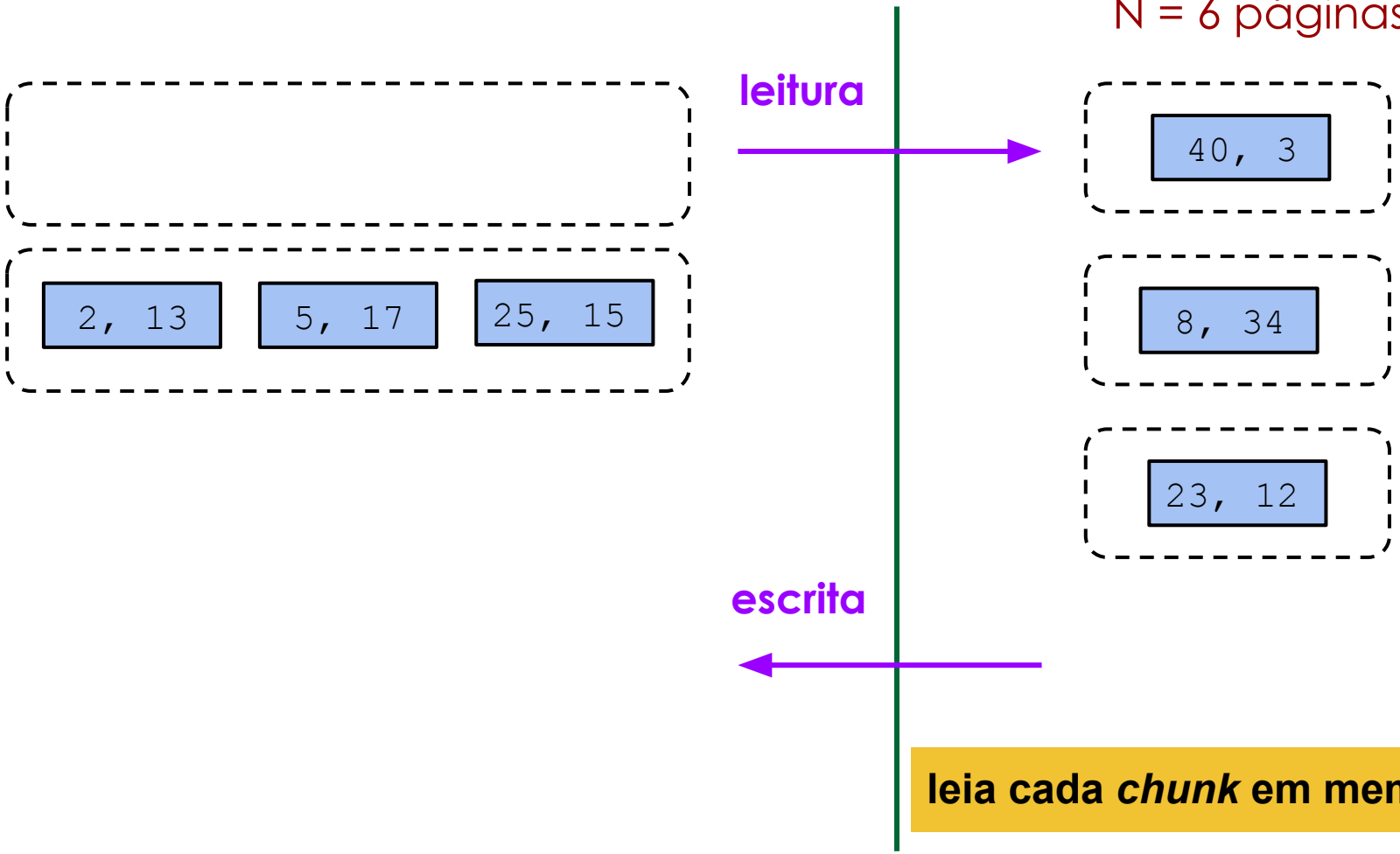
separe em *chunks* que caibam em memória

WARM UP: 2-*way sort*

buffer

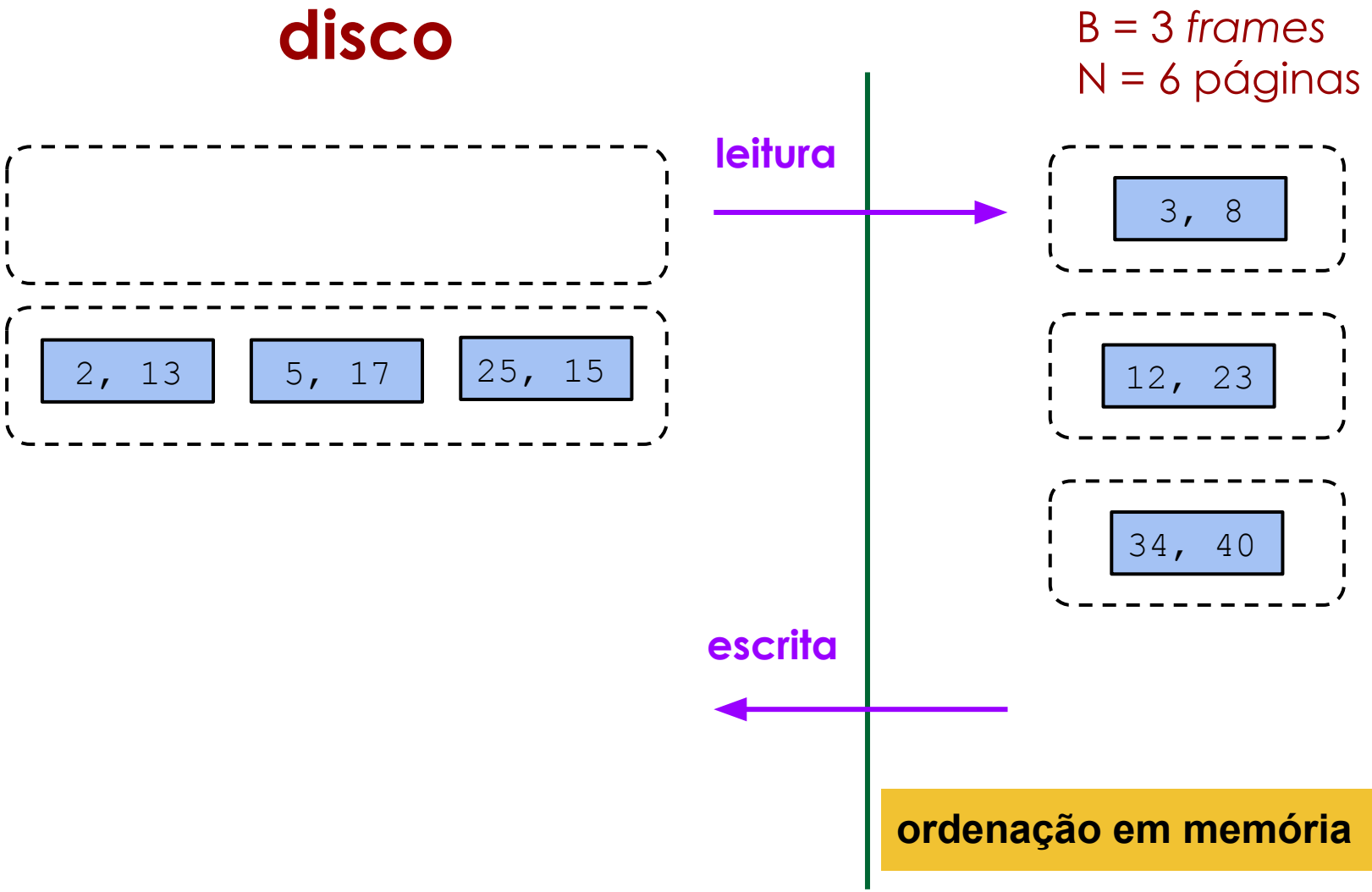
disco

B = 3 *frames*
N = 6 páginas



leia cada *chunk* em memória

WARM UP: 2-*way sort*

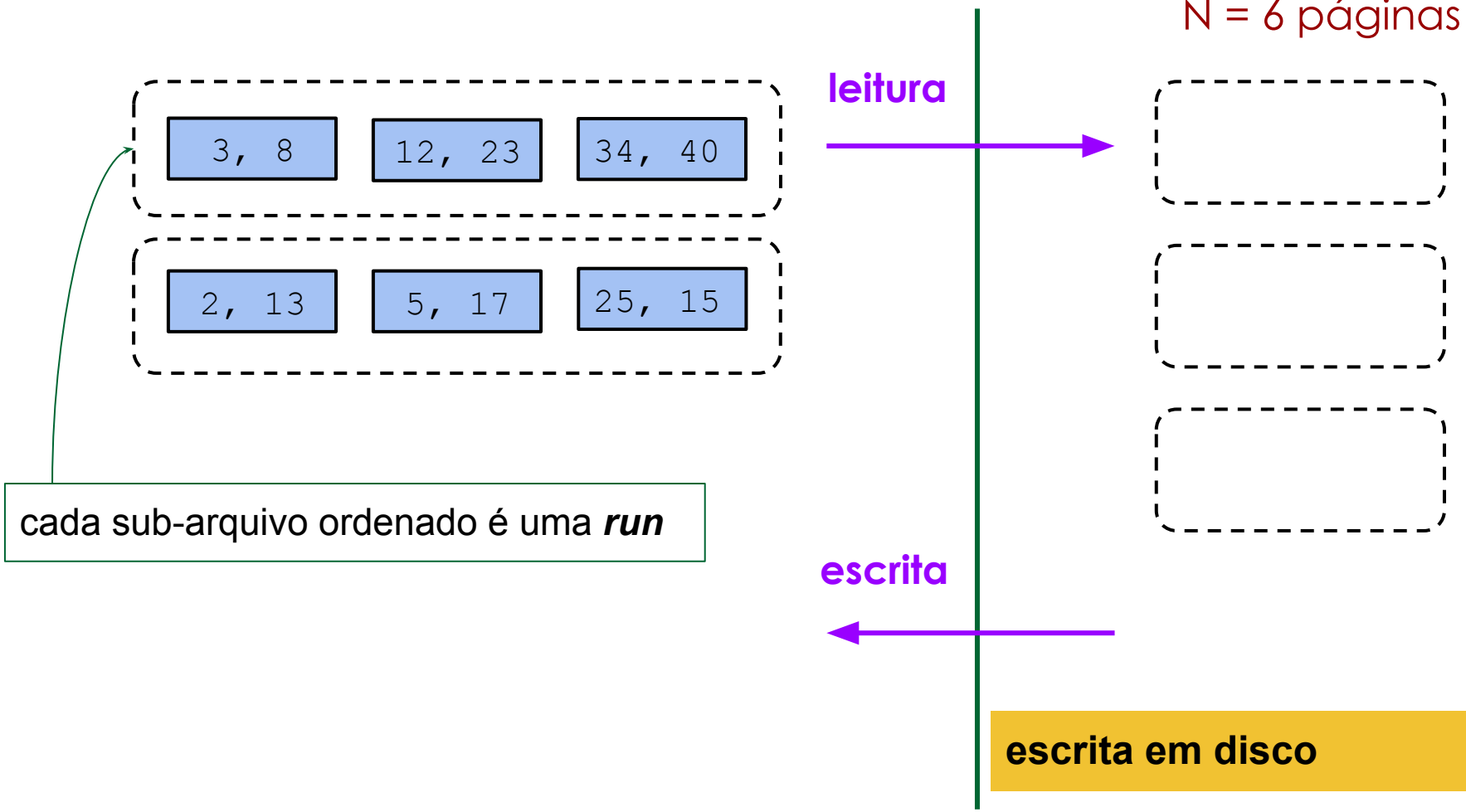


WARM UP: 2-*way sort*

buffer

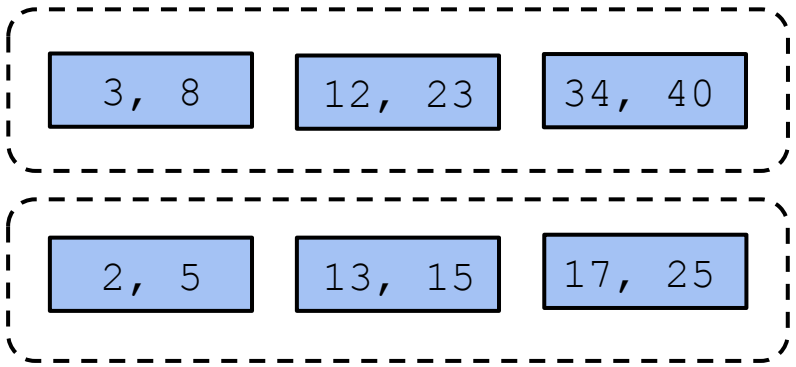
disco

B = 3 *frames*
N = 6 páginas



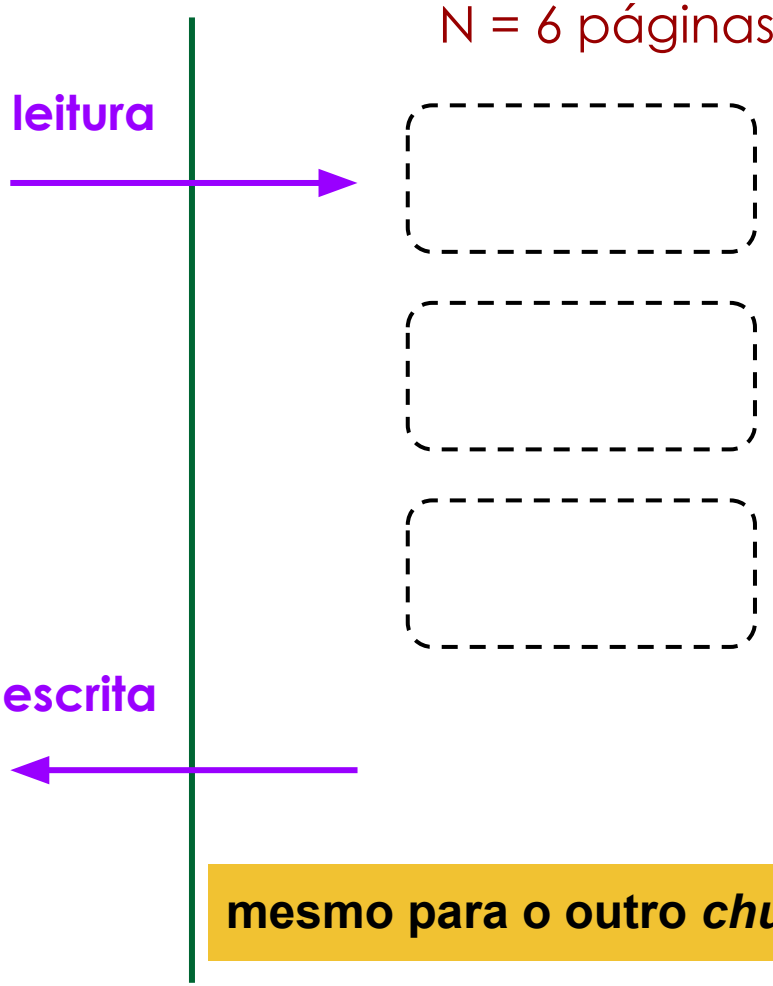
WARM UP: 2-*way sort*

disco



buffer

B = 3 *frames*
N = 6 páginas

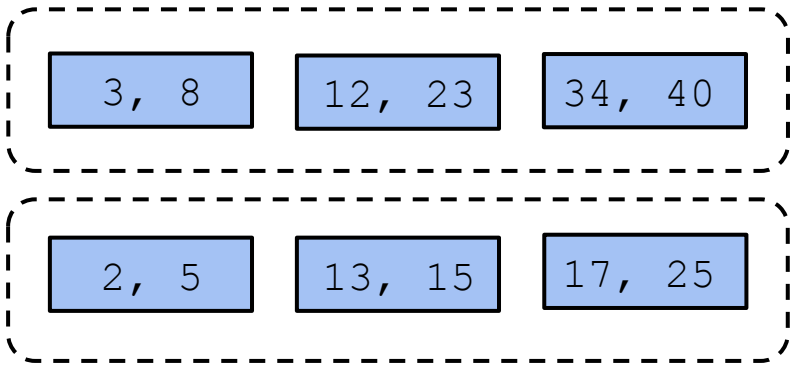


agora temos 2 ***runs***!

mesmo para o outro *chunk*

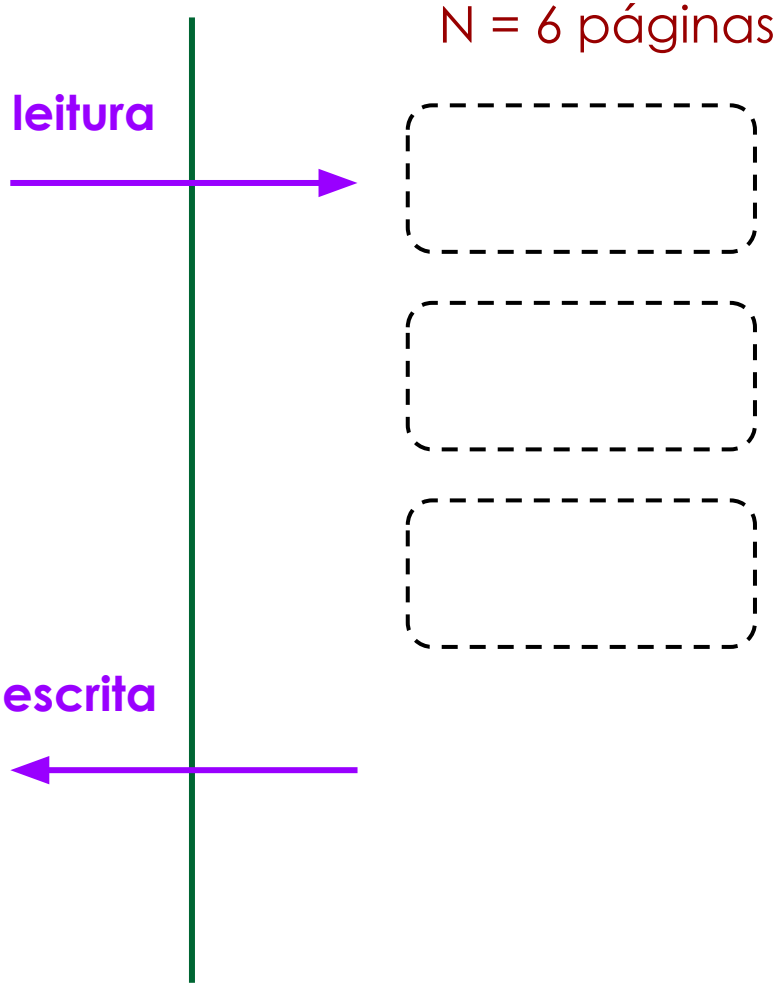
WARM UP: 2-*way sort*

disco



buffer

B = 3 *frames*
N = 6 páginas



PASSO FINAL usamos o algoritmo de intercalação externa para intercalar as 2 *runs*

Custo de I/O

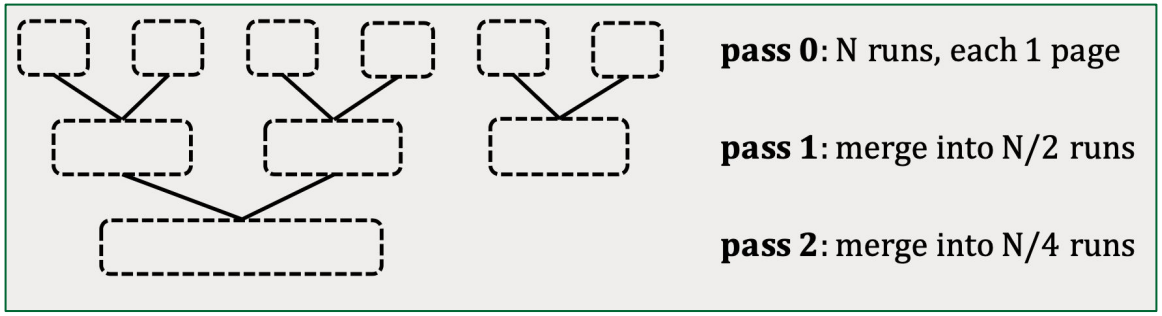
B = 3 páginas de buffer, **N** = 6 páginas

- Passo **0**: criar a primeiras *runs*
  - 1 leitura + 1 escrita para toda página
  - custo total: 6 * (1 + 1) = 12 I/O
- Passo **1**: mergesort externo
  - custo total: 2 * (3 + 3) = 12 I/O

- Logo, 24 I/O

Custo de I/O

Vamos assumir **N** *runs*, cada uma referindo-se a uma única página



Precisamos de $\lceil log_2 N \rceil + 1$ passos para ordenar todo arquivo

Cada passo precisa de 2 N I/O

$$\text{total I/O cost} = 2N(\lceil log_2 N \rceil + 1)$$

Podemos melhorar?

O algoritmo 2-*way* usa somente 3 páginas de *buffer*

E se tivermos mais memória disponível?

Ideia: utilizar o máximo de memória em cada passada, reduzindo o número de passadas, reduzimos a quantidade de operações de I/O

Intercalação de *k* sequências (*k-way*)

- $k \geq 2$ sequências ordenadas em memória externa: $S_1, S_2, \ldots, S_k$
- $|S_i| = n_i$ para $1 \leq i \leq k$
- $n = n_1 + n_2 + \ldots + n_k$
- S (inicialmente vazia) será armazenada em memória externa
- *Buffer* grande o suficiente para
  - um bloco de cada S_i
  - um bloco de S

Intercalação de *k* sequências (*k-way*)

- Seja B_i o bloco do buffer associado a S_i
- Seja B o bloco do buffer associado a S
- Sempre que B_i vazio, leia próximo bloco de S_i
- Continue removendo o menor elemento entre todos B_is para B
- Sempre que B estiver cheio, esvazie B escrevendo sequencialmente em S

Intercalação de *k* sequências (*k-way*)

Para determinar o menor valor dentre as *k* listas

O(n k) -> ineficiente

Uso de fila de prioridades (e.g., *heap*)

O(n log k)

Custo (I/O) da intercalação externa

Supondo *B* >= *3*

$$2N(\lceil log_2 N \rceil + 1) \longrightarrow 2N(\left\lceil log_2 \frac{N}{B} \right\rceil + 1) \longrightarrow 2N(\left\lceil log_{B-1} \frac{N}{B} \right\rceil + 1)$$

runs* iniciais de tamanho 1** ***3-way merge	**aumentando o tamanho das *runs* iniciais para B**	**intercalação B-1 *runs* por vez**

Número de passadas (*passes*)

N	B=3	B=17	B=257
100	7	2	1
10,000	13	4	2
1,000,000	20	5	3
10,000,000	23	6	3
100,000,000	26	7	4
1,000,000,000	30	8	4

Intercalação Polifásica

Intercalação polifásica

Em geral, o merge sort externo reduz o número de *runs* por um fator 2 (i.e., N/2)

A ideia é reduzir o número de *runs* por um fator menor que 2

Intercalação polifásica

Os blocos ordenados são distribuídos de forma desigual entre os fitas disponíveis

Uma fita é deixada livre

Em seguida, a intercalação de blocos ordenados é executada até que uma das fitas esvazie

Neste ponto, uma das fitas de saída troca de papel com a fita de entrada.

Intercalação polifásica

Blocos ordenados

fita 1:	*I N R T*	*A C E L*	*A A B C L O*
fita 2:	*A A C E N*	*A A D*	
fita 3:			

Configuração após uma intercalação-de-2-caminhos das fitas 1 e 2 para a fita 3:

fita 1:	*A A B C L O*	
fita 2:		
fita 3:	*A A C E I N N R T*	*A A A C D E L*

Intercalação polifásica

Depois da intercalação-de-2-caminhos das fitas 1 e 3 para a fita 2:

fita 1:	
fita 2:	*A A A A B C C E I L N N O R T*
fita 3:	*A A A C D E L*

Finalmente:

fita 1:	*A A A A A A A B C C C C D E E I L L N N O R T*
fita 2:	
fita 3:	

Intercalação polifásica

A implementação da intercalação polifásica é simples

A parte mais delicada está na distribuição inicial dos blocos ordenados entre as fitas

Distribuição dos blocos nas diversas etapas do exemplo:

fi ta 1	fi ta 2	fi ta 3	Total
3	2	0	5
1	0	2	3
0	1	1	2
1	0	0	1

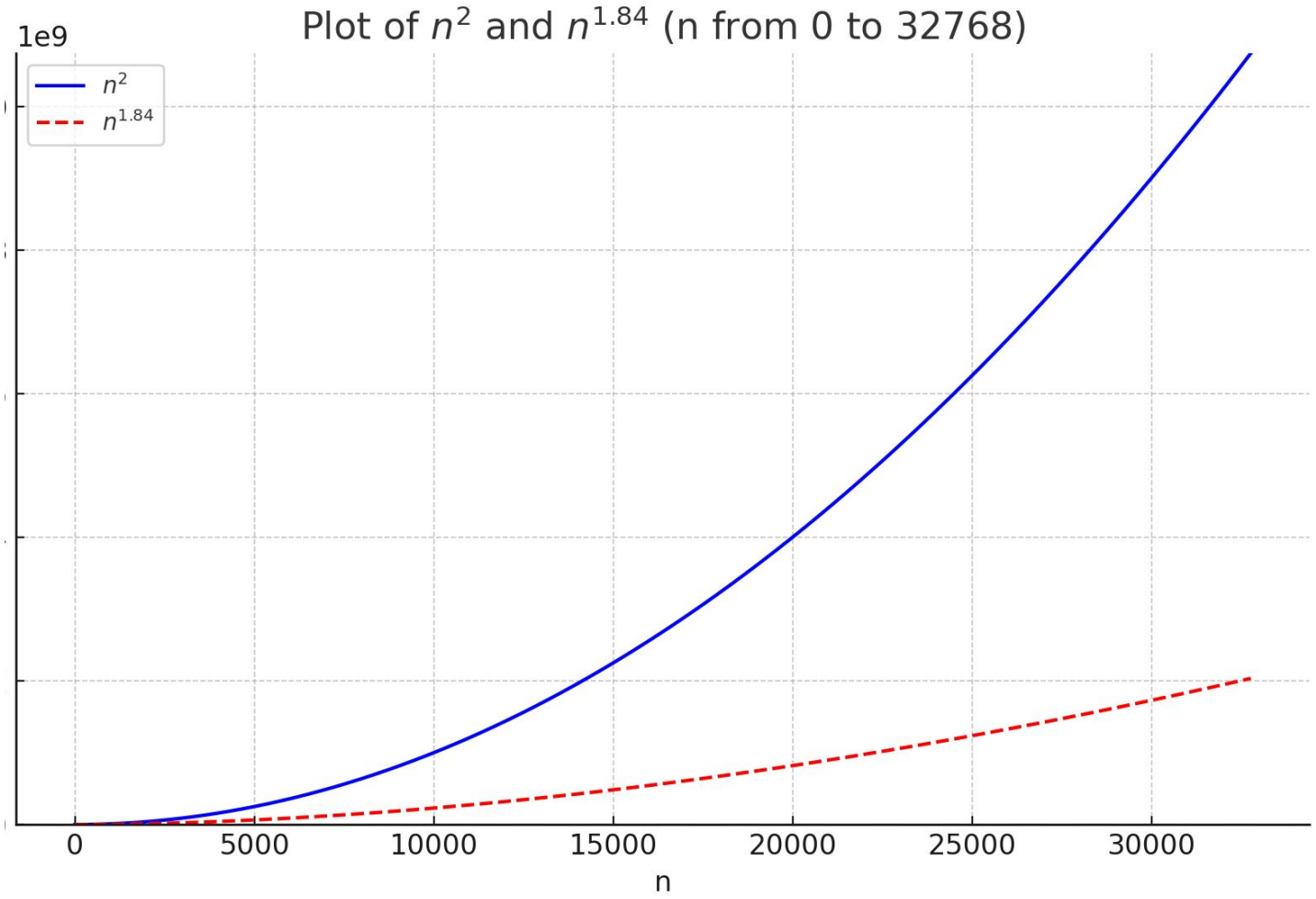
Intercalação polifásica

Para cada iteração da intercalação polifásica, o número total de runs segue um padrão similar a uma sequência de números de Fibonacci

Ex: com 4 fitas (buffers) e um dataset contendo 57 *runs*,

- o total a cada iteração é:
  - 57, 31, 17, 9, 5, 3, 1
- o fator de redução # de *runs* ~ 1.84
  - (57/31), (31/17), (17/9), (9/5), (5/3), 3/1

Intercalação polifásica



Quicksort externo

Quicksort

- Paradigma divisão e conquista
- *Quicksort interno*
  - Escolher um elemento pivot *p* da lista (ex.: rand, mediana)
  - Particionamento: reordenar a lista
    - Elementos menores que *p*: colocar antes de *p*
    - Elementos maiores que *p*: colocar depois de *p*
  - Recursivamente, ordenar as sub-listas >p e <p

Quicksort Externo

- Mesmo paradigma: divisão e conquista
- Similar ao *quicksort* interno, porém pivot é substituído por um *buffer* de tamanho *m*
  - $m = O(\log n)$, $m \geq 3$
- É um algoritmo *in situ*
  - Não precisa de memória externa adicional
- Os *n* registros a serem ordenados estão em memória externa de acesso aleatório

Quicksort Externo

- *A*: arquivo a ser ordenado
- Seja R_i o registro que se encontra na *i-ésima* posição de *A*
- Algoritmo:
  - Particionar *A* da seguinte forma:
  - $\{R_1, \ldots, R_i\} \leq \underbrace{R_{i+1} \leq R_{i+2} \leq \ldots \leq R_{j-2} \leq R_{j-1}}_{pivot} \leq \{R_j, \ldots, R_n\}$

    A_1 A_2
  - Chamar recursivamente para cada um dos subarquivos $A_1 = \{R_1, \ldots, R_i\}$ e $A_2 = \{R_j, \ldots, R_n\}$

Quicksort Externo

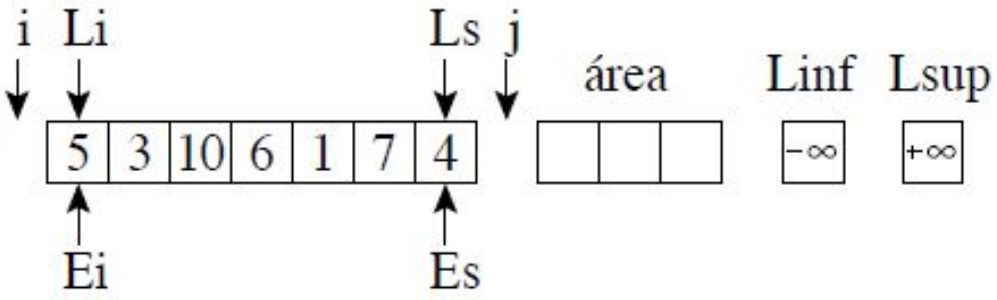
- Para o particionamento, é utilizada uma área de armazenamento na memória interna
- Tamanho da área: *TamArea* = *j* – *i* – *1*, com *TamArea* ≥ *3*
- Nas chamadas recursivas, deve-se considerar que
  - Primeiro deve ser ordenado o arquivo de menor tamanho
  - Condição para que, na média, O(log n) subarquivos tenham o processamento adiado
  - Subarquivos vazios ou com um único registro são ignorados
  - Caso o arquivo de entrada A possua no máximo *TamArea* registros, ele é ordenado em um único passo

Quicksort Externo

- Ler primeiros *m/2* e últimos *m/2* elementos de *A* para o buffer e ordená-los
- Guardar max(buffer) e min(buffer) para evitar reordenar elementos do meio que já estão escritos
- Ler *x* <- próximo elemento do começo ou do fim, alternadamente, de forma a balancear a escrita
  - Se ***x* ≤ min(buffer)**, escrever *x* no espaço liberado no começo do arquivo de entrada
  - Se ***x* ≥ max(buffer),** escrever *x* no fim do arquivo
  - Senão (**min(buffer) < x < max(buffer**), escrever min(buffer) ou max(buffer) e colocar *x* no buffer
- Ao terminar, escrever o conteúdo do buffer
- Recursivamente, ordenar(partição menor); ordenar(partição maior)

Quicksort Externo - Particionamento

- Valores das chaves R_i e R_j são denominados limite inferior (**Linf**) e limite superior (**Lsup**) da partição
  - Tais limites são inicializados com os valores $-\infty$ e $+\infty$
- A leitura de *A* é controlada por ponteiros de leitura inferior (**Li**) e superior (**Ls**)
  - A cada leitura do extremo inferior, **Li** é incrementado; a cada leitura do extremo superior, **Ls** é decrementado
- Similarmente, a escrita em A é controlada por ponteiros de escrita inferior (**Ei**) e superior (**Es**)



Quicksort Externo – Exemplo 1

Legenda:
- Lido
- Escrito

Ordenar: A = {5, 3, 10, 6, 1, 7, **4**}, m=3

1. Lê {4, 5, 7}, ordene em memória: B {4, 5, 7}, min=4, max=7
2. Lê {3} (3 ≤ min) □ escreve 3: A{3, 3, 10, 6, 1, 7, 4}
3. Lê {1} (1 ≤ min) □ escreve 1: A{3, 1, 10, 6, 1, 7, 4}
4. Lê {10} (10 ≥ max) □ escreve 10: A{3, 1, 10, 6, 1, 7, 10}
5. Lê {6} (min < 6 < max) □ escreve max (p/ balancear), coloca {6} no buffer

   A{3, 1, 10, 6, 1, 7, 10}, B{4, 5, 6}, min=4, max=6
6. Escreve buffer no arquivo: A{3, 1, 4, 5, 6, 7, 10}
7. Recursão: ordena A{3,1}; ordena A{7,10};

Quicksort Externo – Exemplo 2

Legenda:
- Lido
- Escrito

- Ordenar: A = {**5**, 3, 10, 6, 1, 7, 4}, m=3

1. Lê {5, 4, 3}, ordene em memória: B {3, 4, 5}, min=3, max=5
2. Lê {7} (7 ≥ max) □ escreve 7: A{5, 3, 10, 6, 1, 7, 7}
3. Lê {10} (10 ≥ max) □ escreve 10: A{5, 3, 10, 6, 1, 10, 7}
4. Lê {1} (1 ≤ min) □ escreve 1: A{1, 3, 10, 6, 1, 10, 7}
5. Lê {6} (6 ≥ max) □ escreve 6: A{1, 3, 10, 6, 6, 10, 7}
6. Escreve buffer no arquivo: A{1, 3, 4, 5, 6, 10, 7}
7. Recursão: ordena A{1}; ordena A{6, 10, 7};

Quicksort Externo - Análise

- Sejam:
  - *n*: número de registros no arquivo
  - *b*: tamanho do bloco de leitura e gravação do Sistema Operacional
- **Melhor caso**: $O\left(\frac{n}{b}\right)$
  - Ocorre, por exemplo, quando o arquivo de entrada já está ordenado
- **Pior caso**: $O\left(\frac{n^2}{TamArea}\right)$
  - Ocorre quando uma das partições tem o maior tamanho possível e a outra é vazia
  - À medida que *n* cresce a probabilidade de ocorrência do pior caso tende a zero
- **Caso médio**: $O\left(\frac{n}{b}\log(\frac{n}{TamArea})\right)$

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Estruturas de Dados

Ordenação em Memória Secundária

Professores: Anisio Lacerda
Lucas Ferreira
Wagner Meira Jr.
Washington Cunha