CS 564: Database Management Systems

Lecture 19: External Sorting

Xiangyao Yu

3/6/2024

Outline

Index concurrency control

External merge

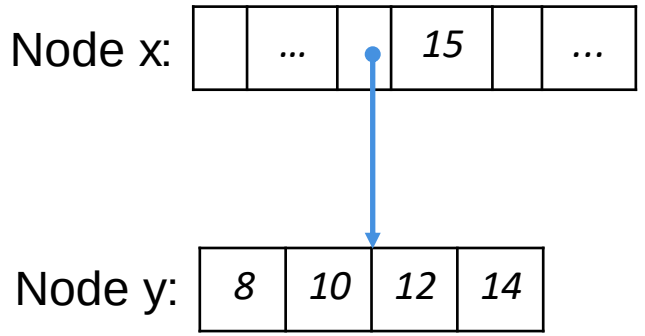
External merge-sort

- 2-way merge sort
- Multi-way merge sort

Concurrency Challenge

search(14)

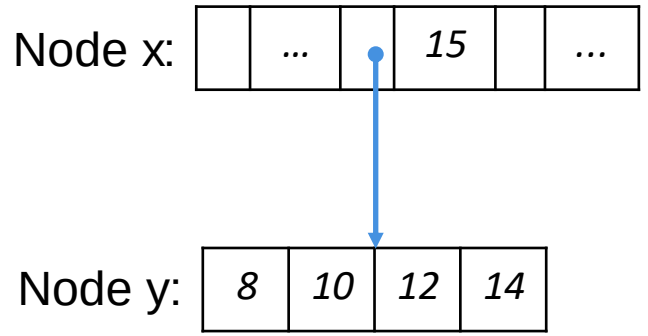
- read(x)
- get ptr tp y
- read(y)

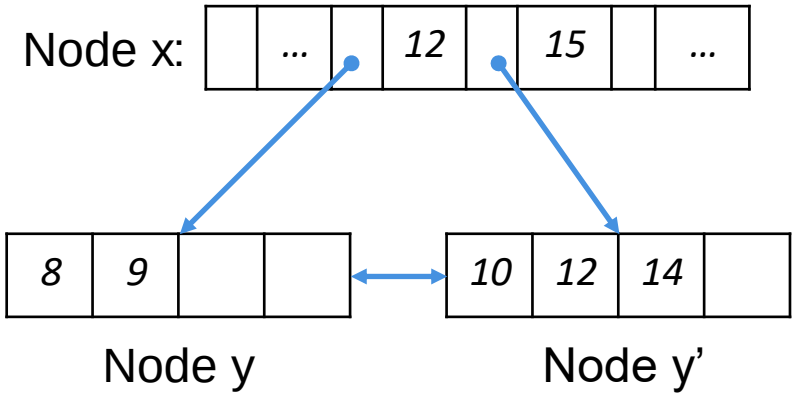


Concurrency Challenge

Insert(9):

- read(x)
- get pointer to node y
- allocate node y'
- move some entries in y to y'
- insert 12 into x
- update pointers in x





Concurrency Challenge

search(14)

read(x)

get ptr tp y

Insert(9):

read(x)

get pointer to node y

allocate node y'

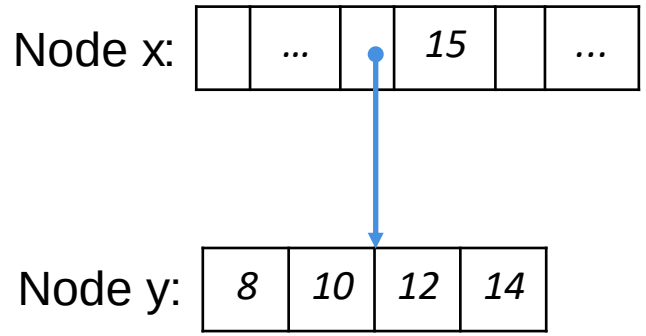
move some entries in y to y'

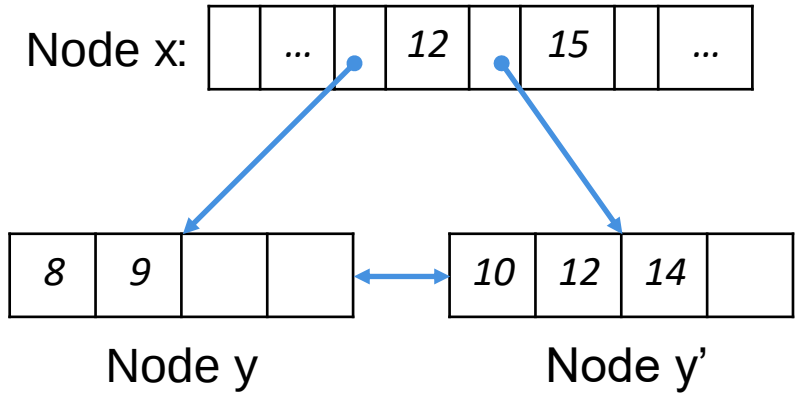
insert 12 into x

update pointers in x

read(y)

error: 14 not found!





Solution: Locking Coupling

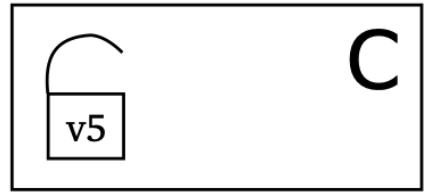
Lock coupling (aka. lock crabbing)

- lock parent
- access parent
- lock child
- if child is full:
  - split the child node
- release the parent node
- continue until reach leaf

1. lock node A
2. access node A

3. lock node B
4. unlock node A
5. access node B

6. lock node C
7. unlock node B
8. access node C
9. unlock node C



Locking Coupling – Example

search(14)

read(x)

get ptr tp y

read(y)

Insert(9):

read(x)

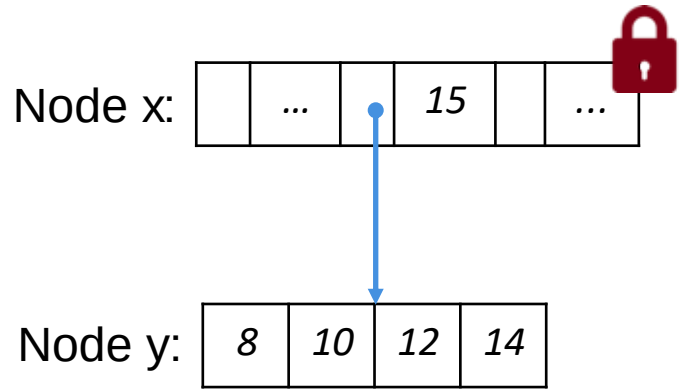
get pointer to node y

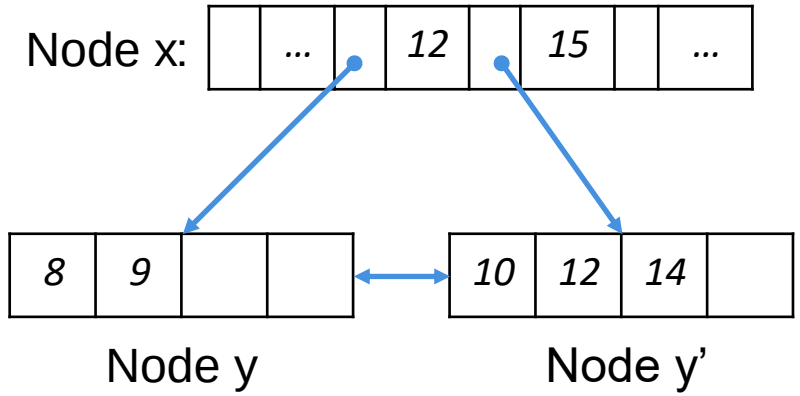
allocate node y'

move some entries in y to y'

insert 12 into x

update pointers in x





Locking Coupling – Example

search(14)

read(x)

get ptr tp y

Insert(9):

blocked by the lock

read(x)

get pointer to node y

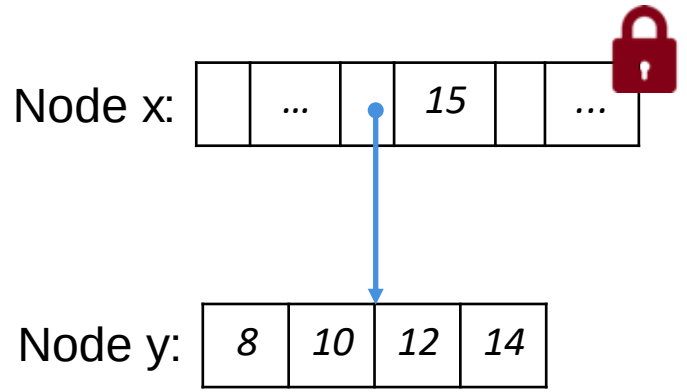
allocate node y'

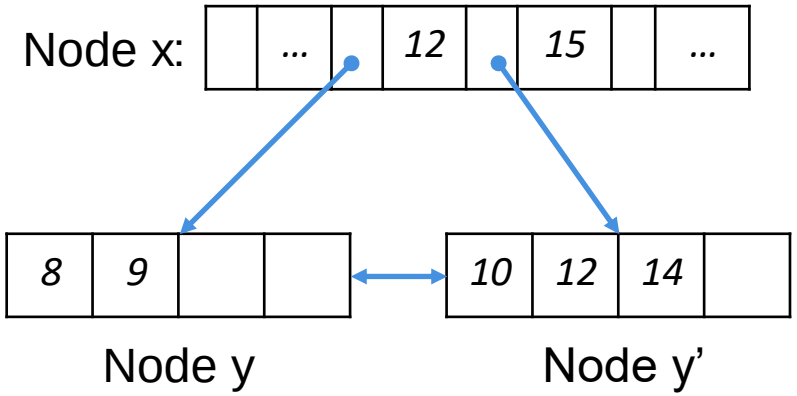
move some entries in y to y'

insert 12 into x

update pointers in x

read(y)





Locking Coupling – Example

search(14)

lock(x); read(x)

get ptr tp y

lock(y); read(y)

unlock(x); unlock(y)

Insert(9):

trylock(x) -> wait

lock(x); read(x)

get pointer to node y

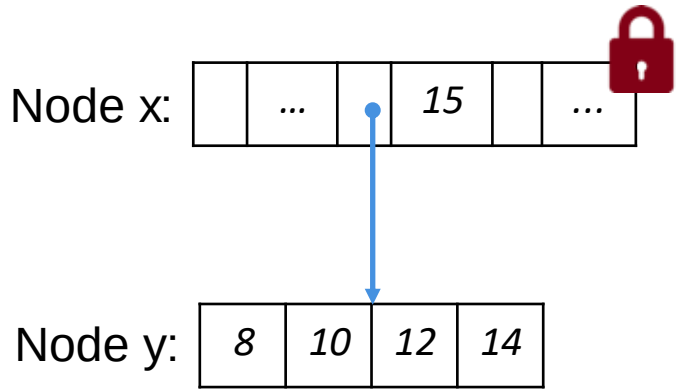
lock(y); allocate node y'

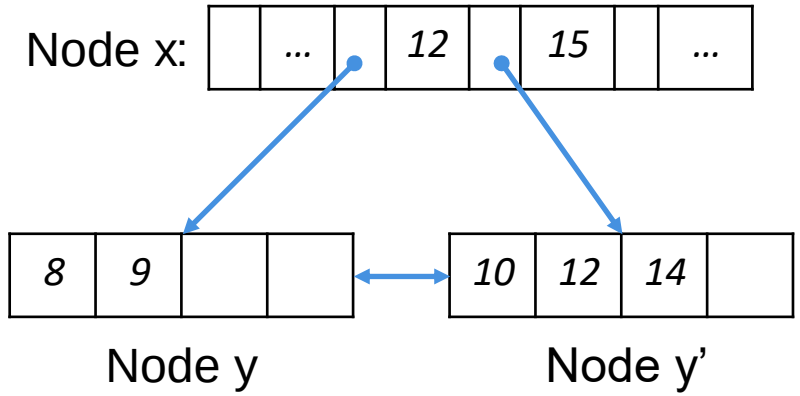
move some entries in y to y'

insert 12 into x

update pointers in x

unlock(x); unlock(y)





Outline

Index concurrency control

External merge

External merge-sort

- 2-way merge sort
- Multi-way merge sort

Why Sorting?

Users often want the data sorted (**`ORDER BY`**)

First step in bulk-loading a B+ tree

Used in duplicate elimination

The sort-merge join algorithm (later in class) involves sorting as a first step

Sorting in Databases

Why don't the standard sorting algorithms work for a database system?

- Merge sort
- Quick sort
- Heap sort

The data typically does not fit in memory!

- E.g., how do we sort 1TB of data with 8GB of RAM?

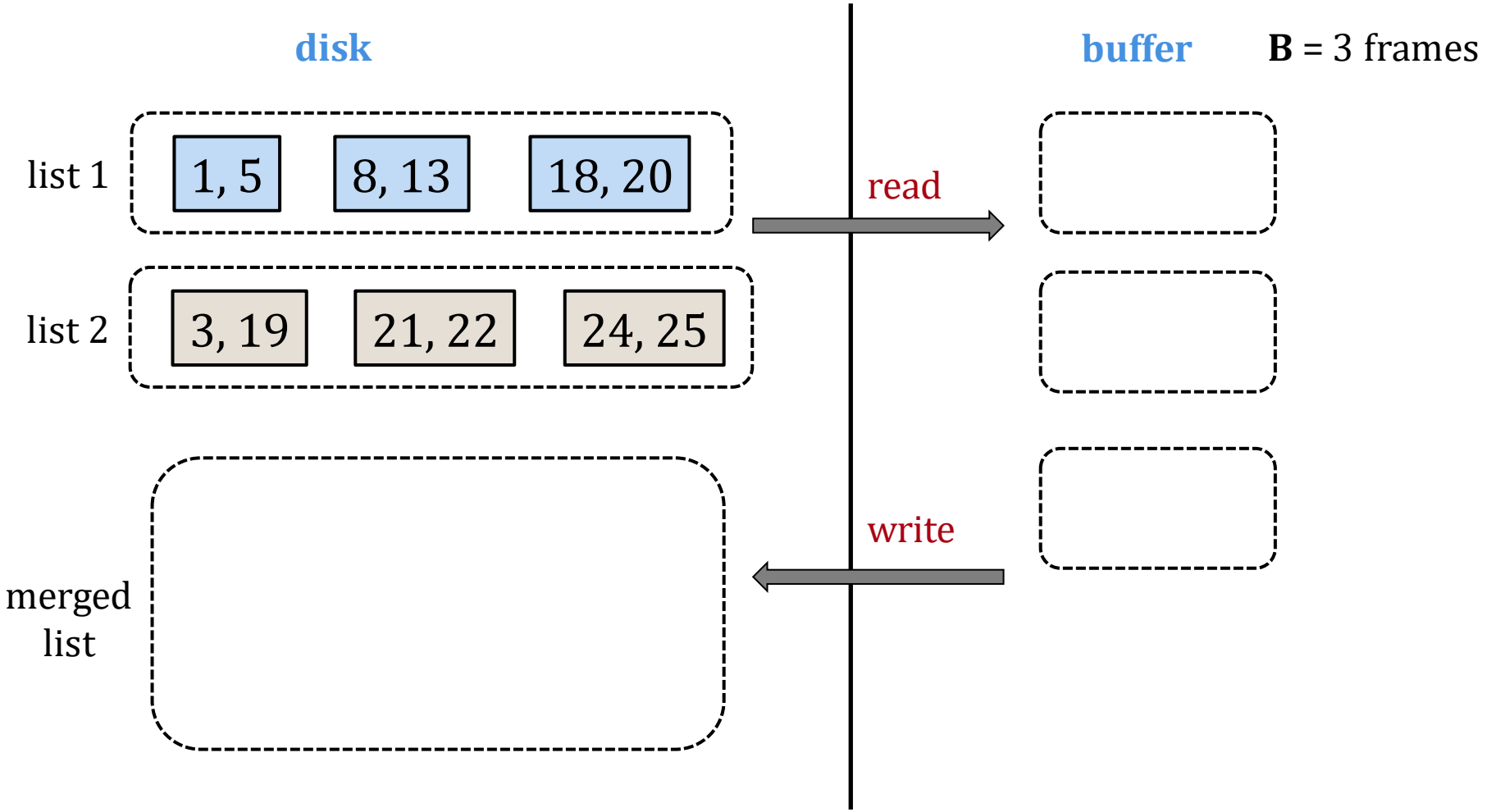
External Merge

Input: 2 sorted lists (with *M* and *N* pages*)*

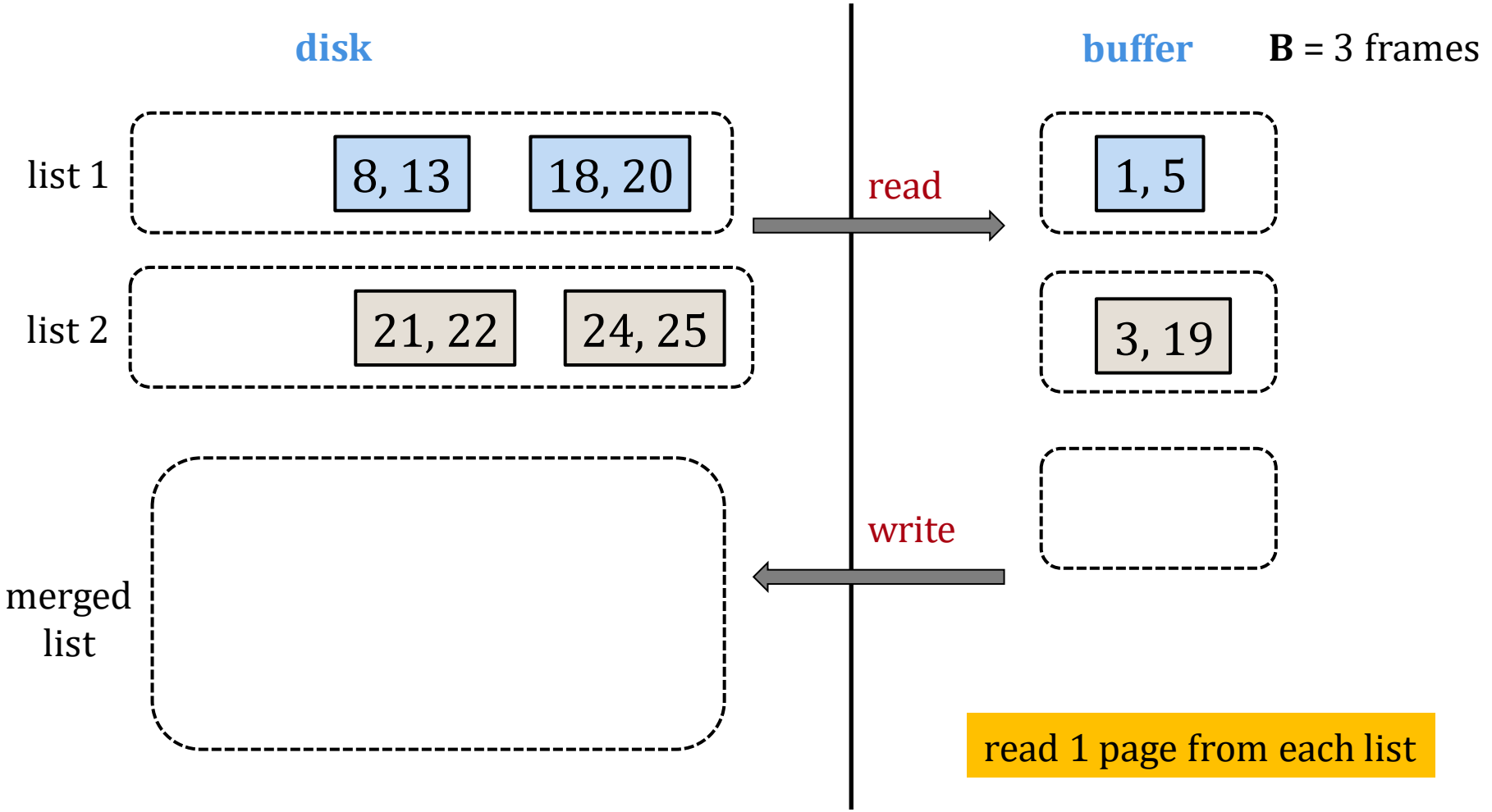
Output: 1 merged sorted list (with *M*+*N* pages)

We can efficiently (in terms of I/O) merge the two lists using a buffer of size 3 using only *2(M+N)* I/Os !

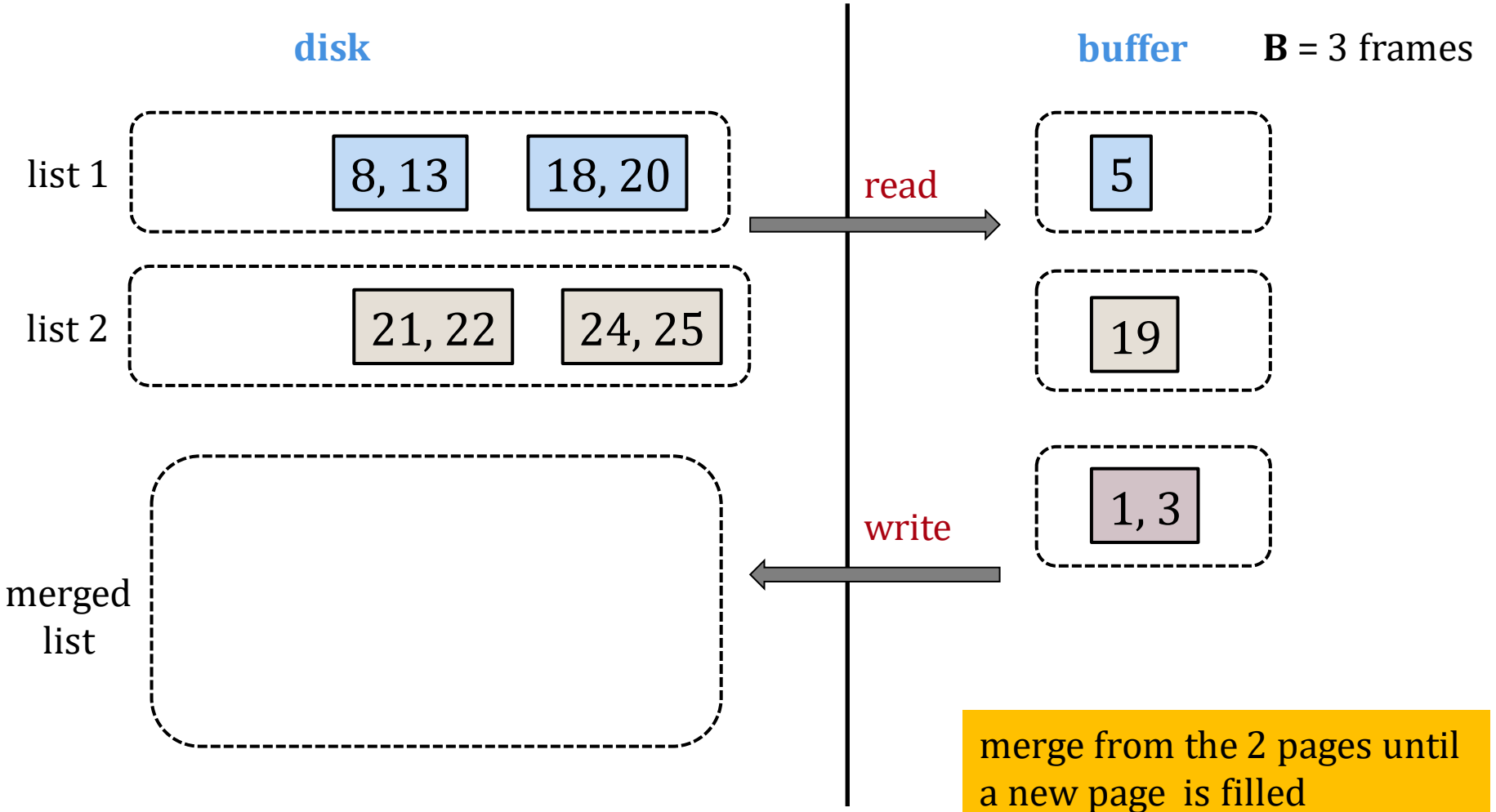
External Merge Algorithm



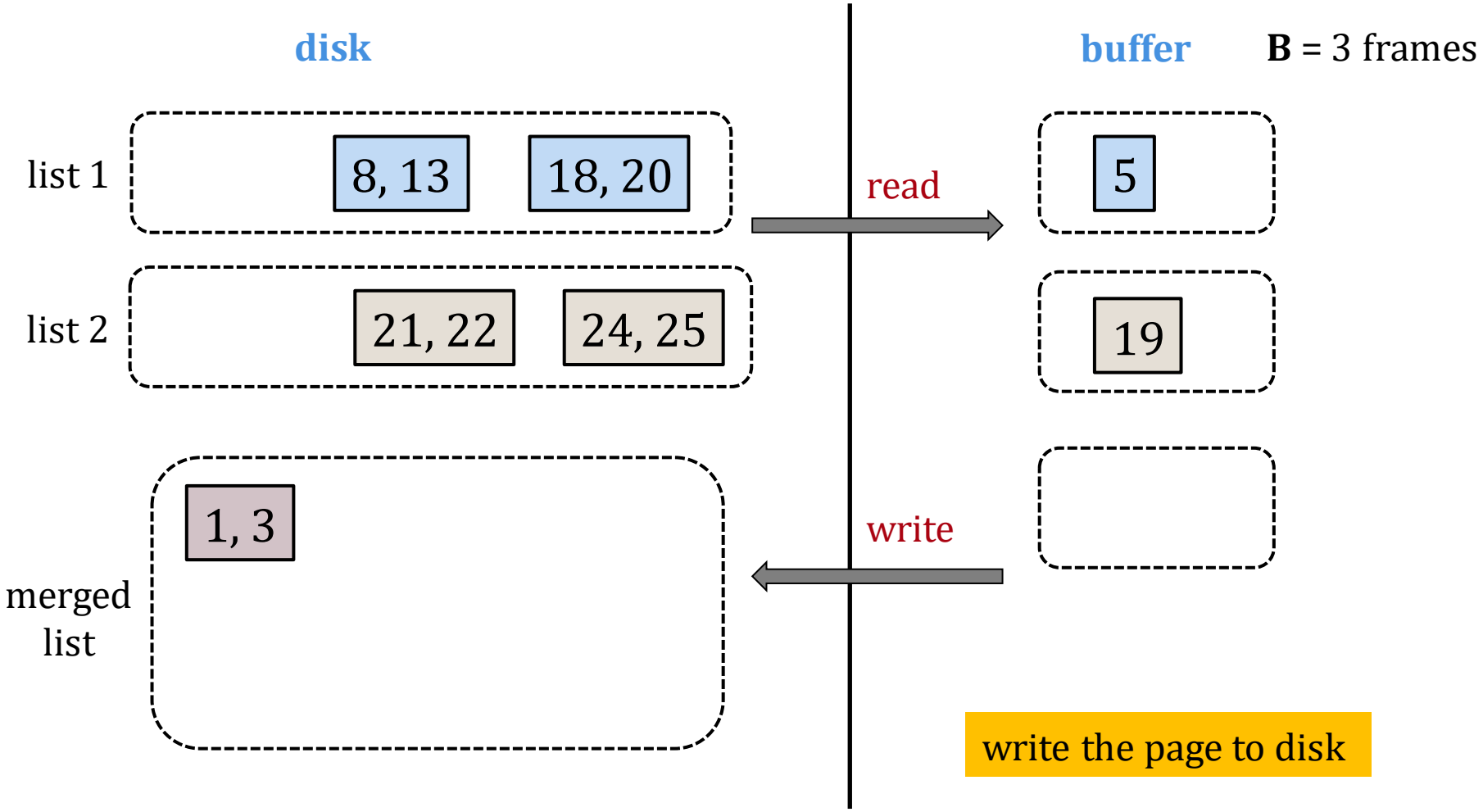
External Merge Algorithm



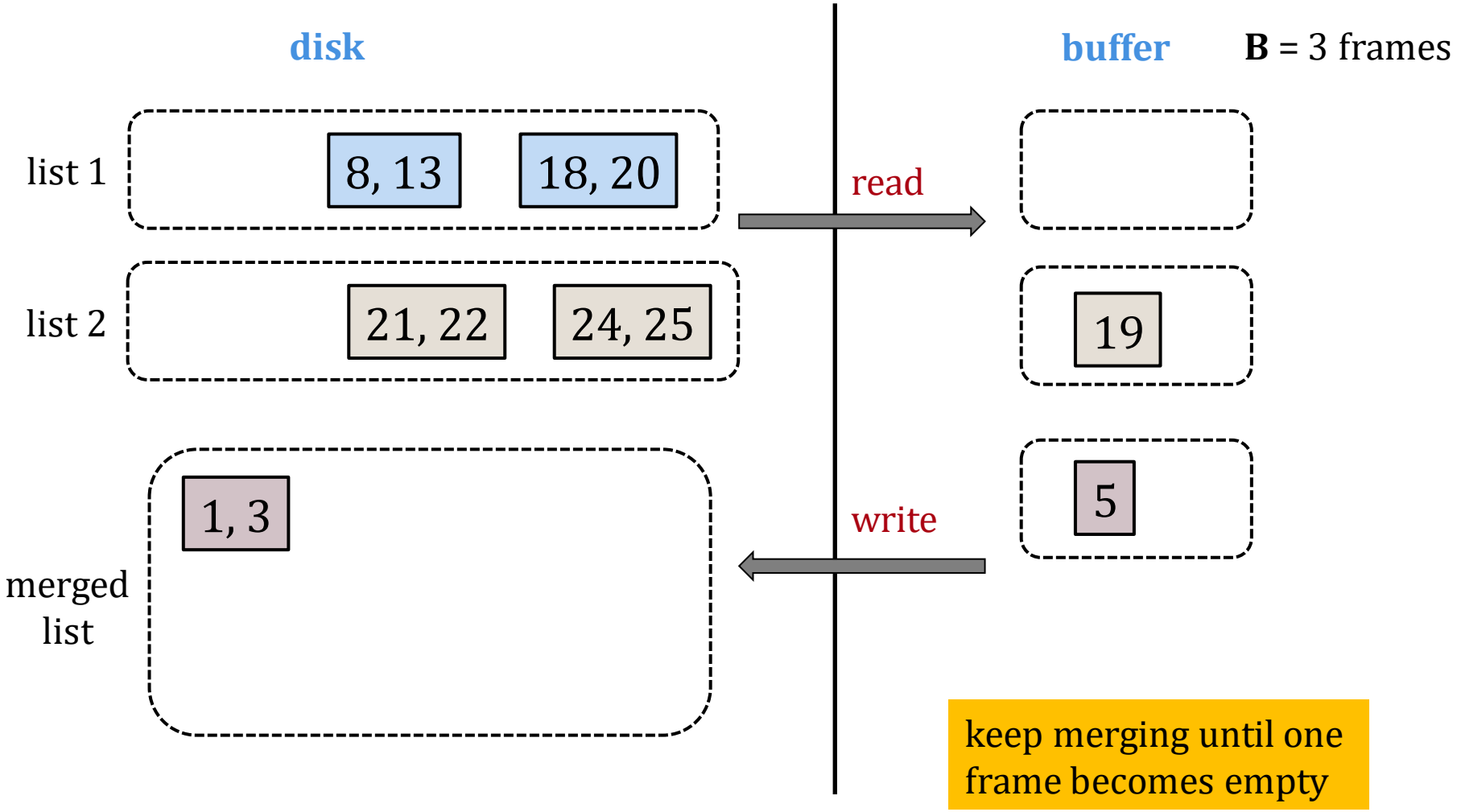
External Merge Algorithm



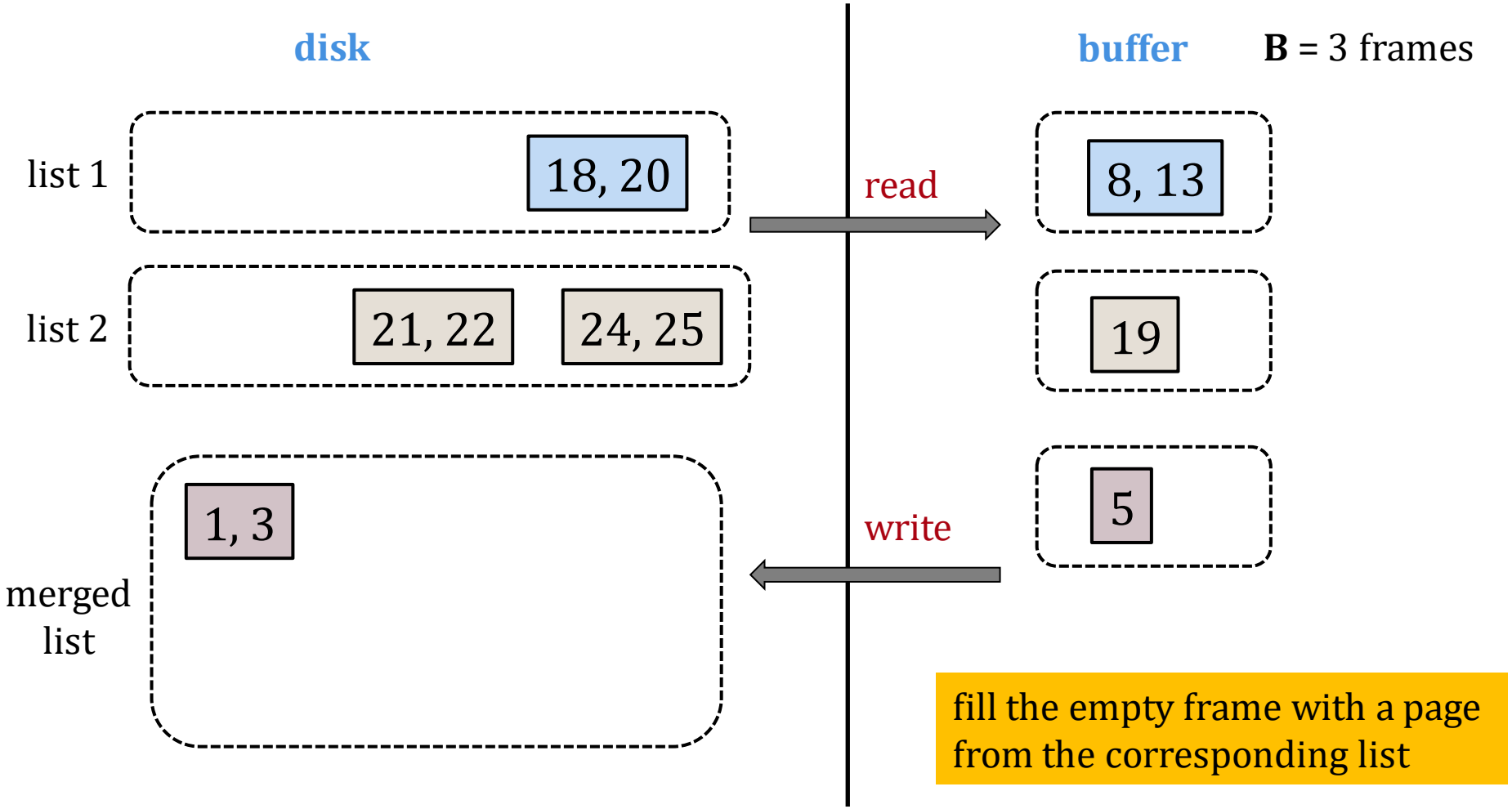
External Merge Algorithm



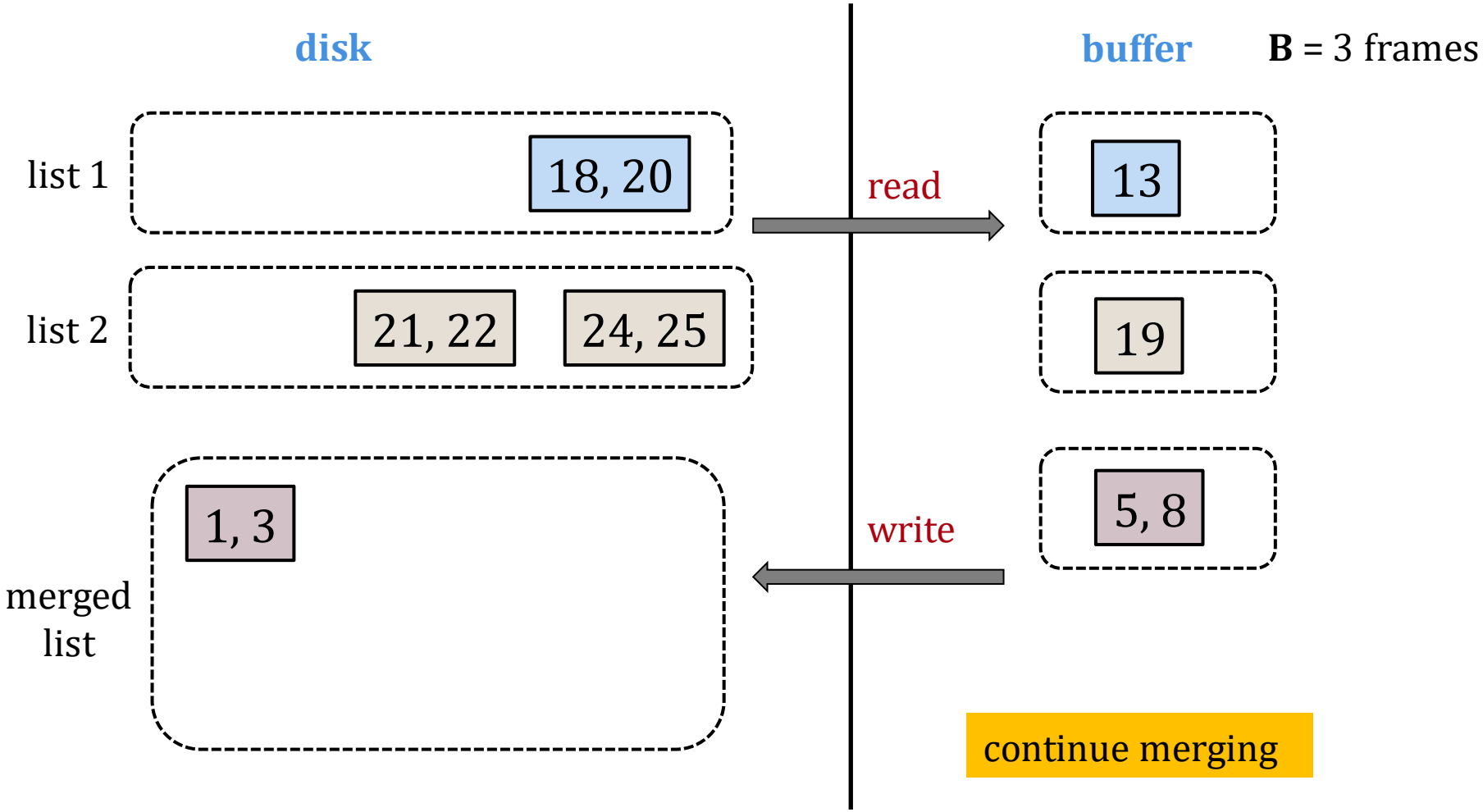
External Merge Algorithm



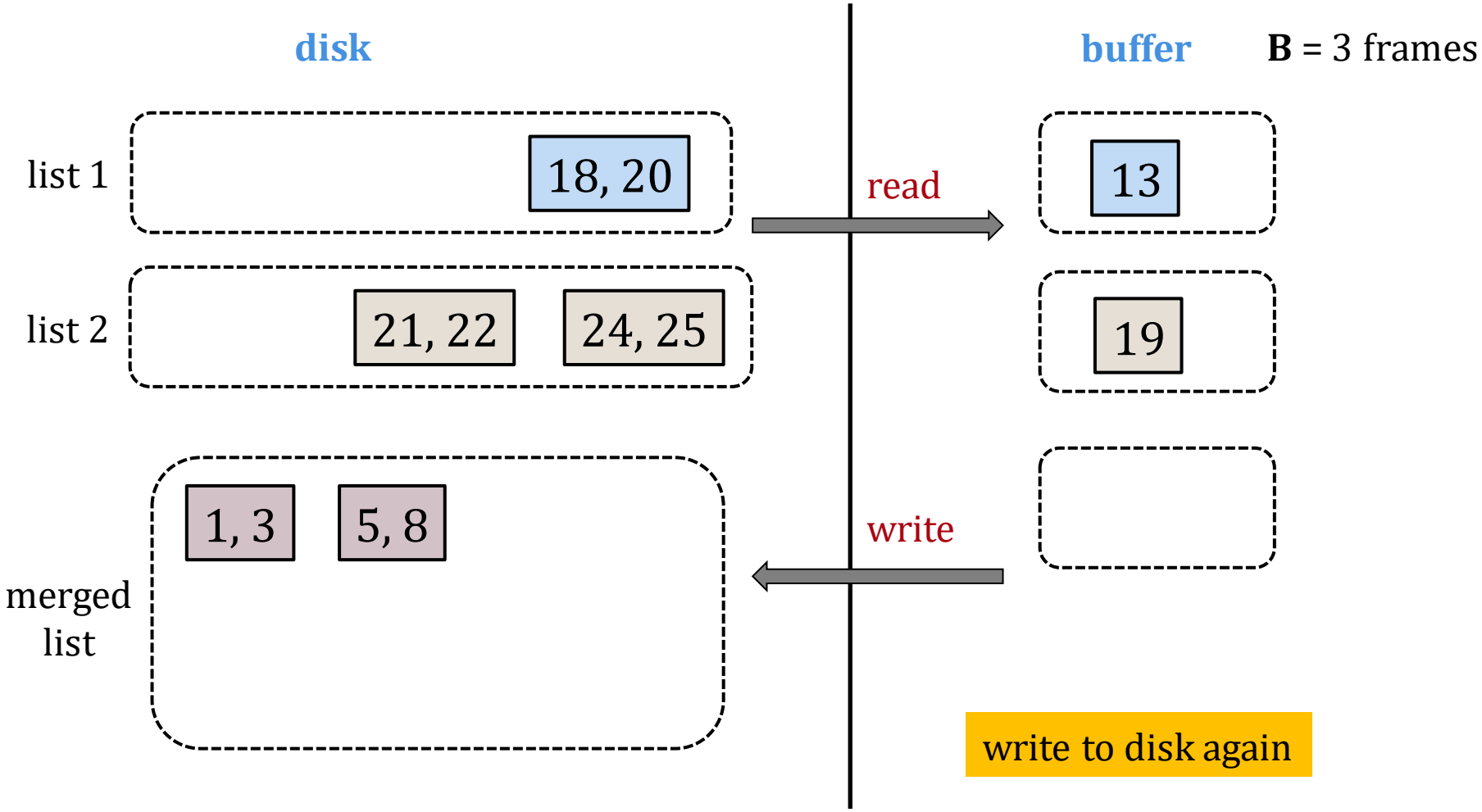
External Merge Algorithm



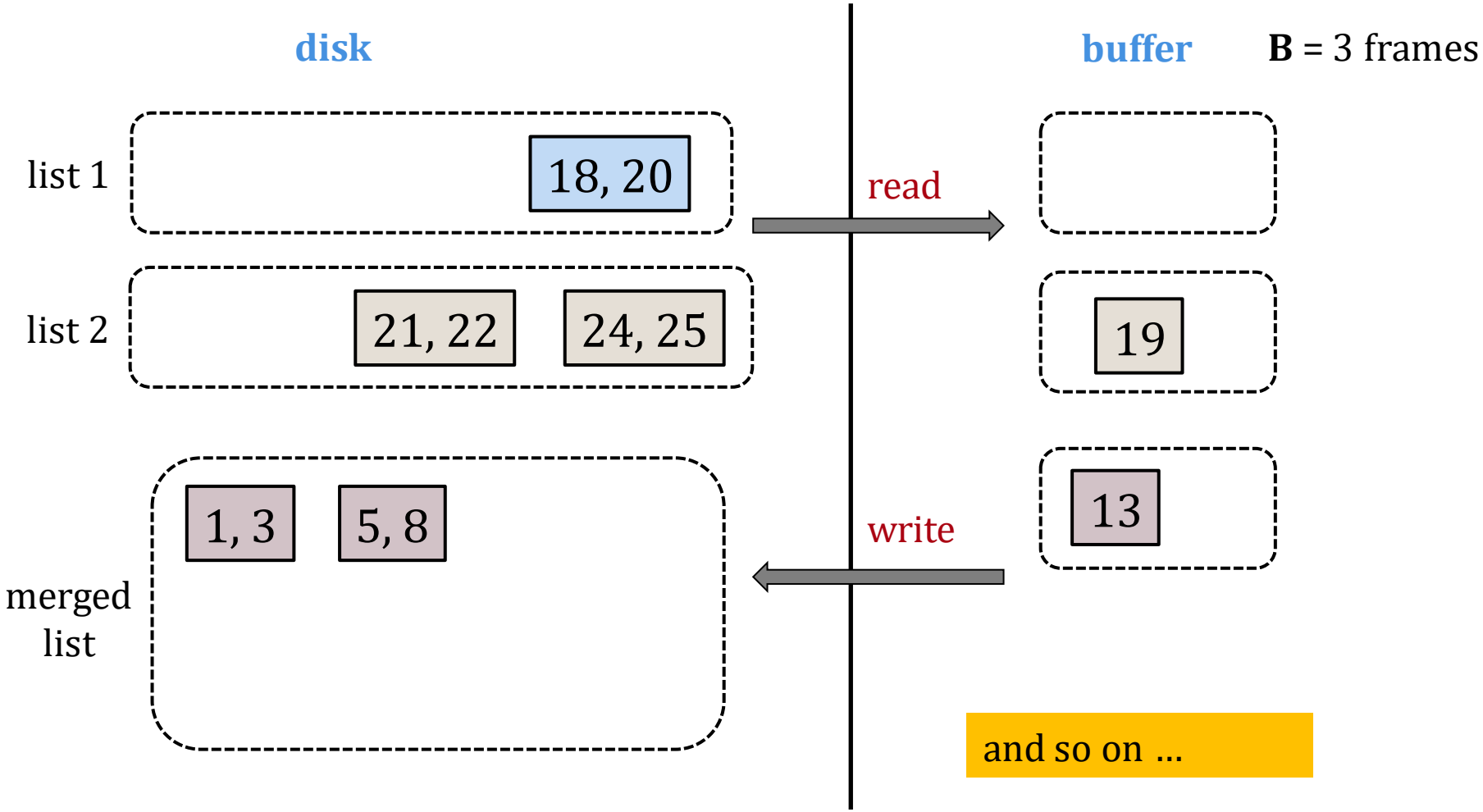
External Merge Algorithm



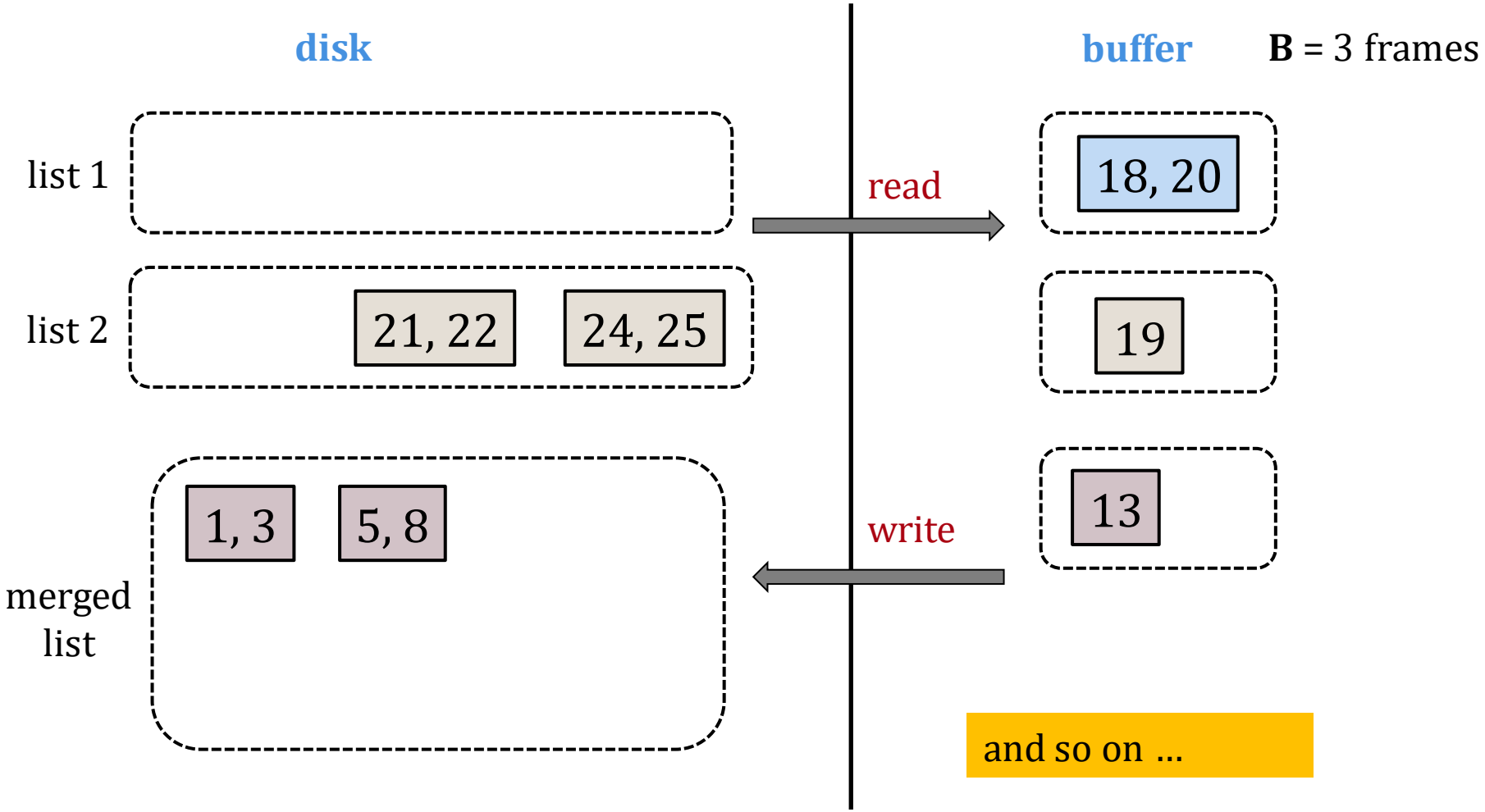
External Merge Algorithm



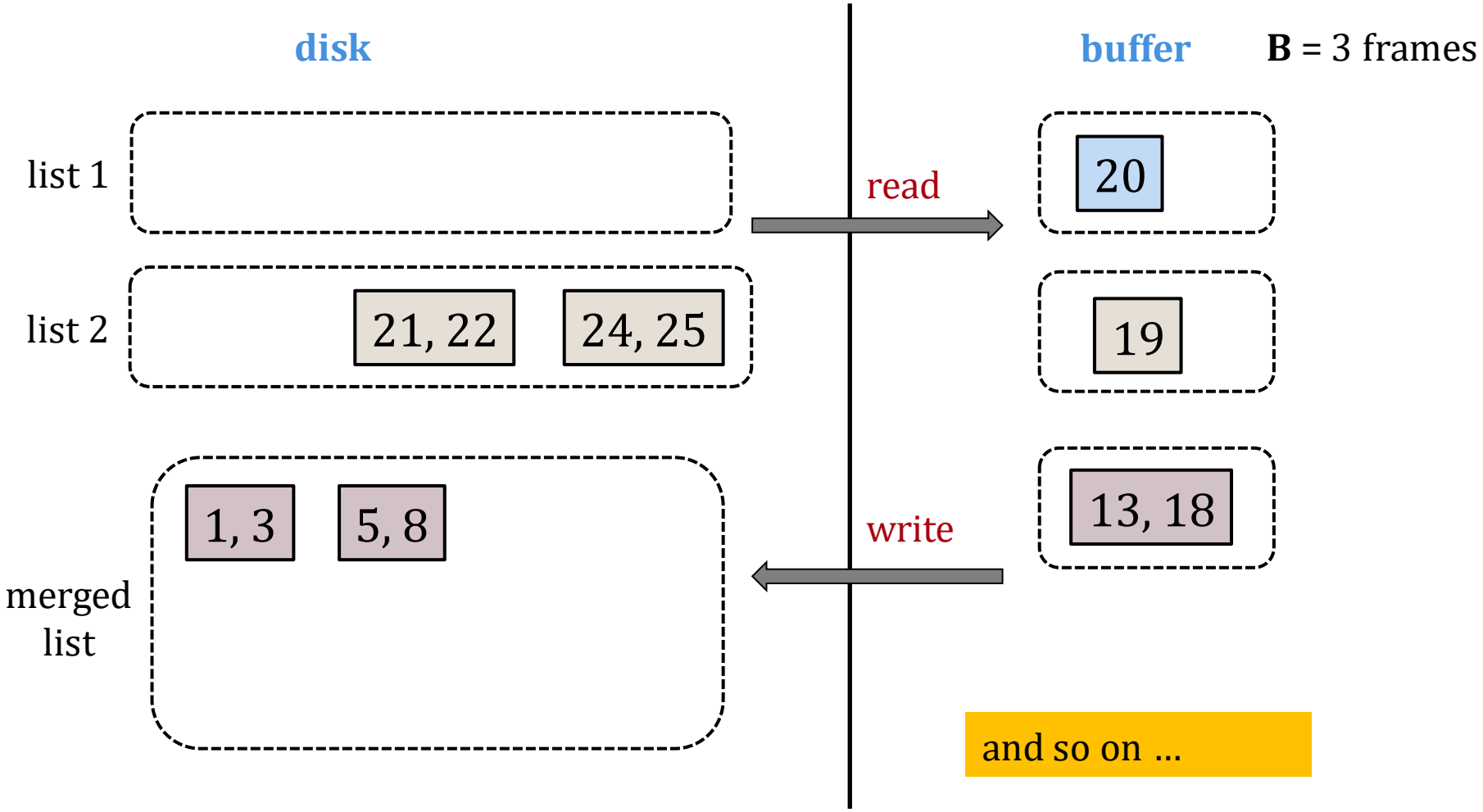
External Merge Algorithm



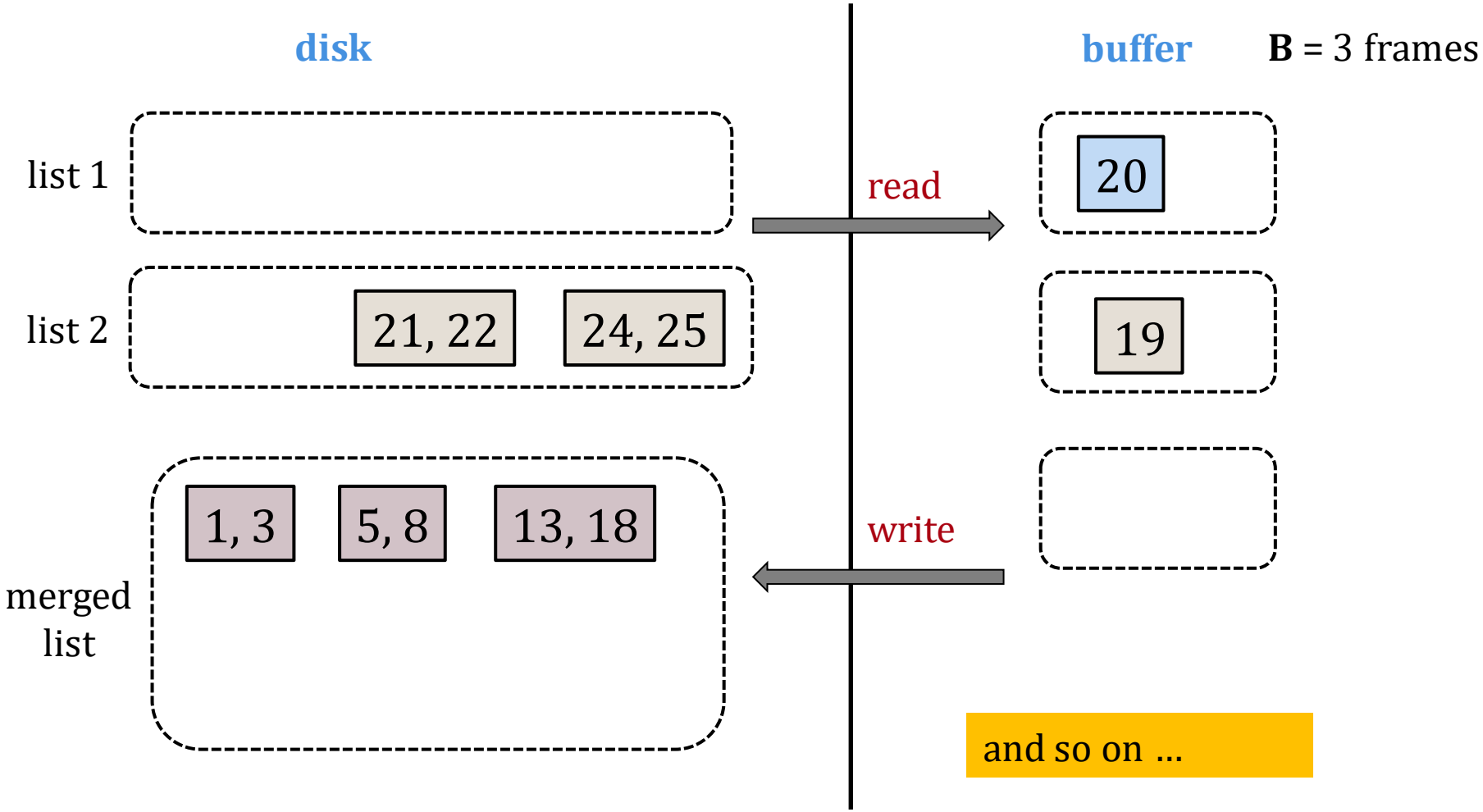
External Merge Algorithm



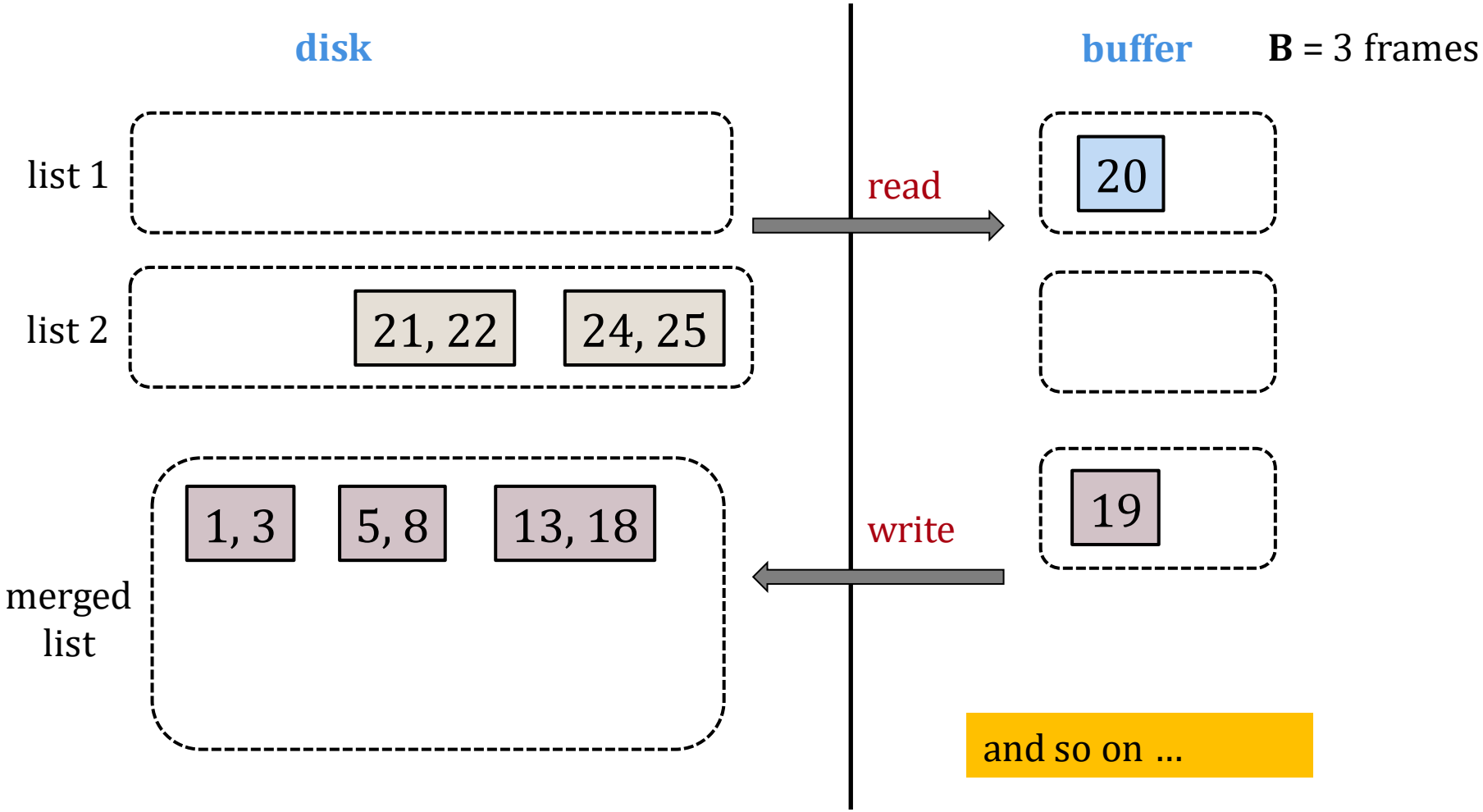
External Merge Algorithm



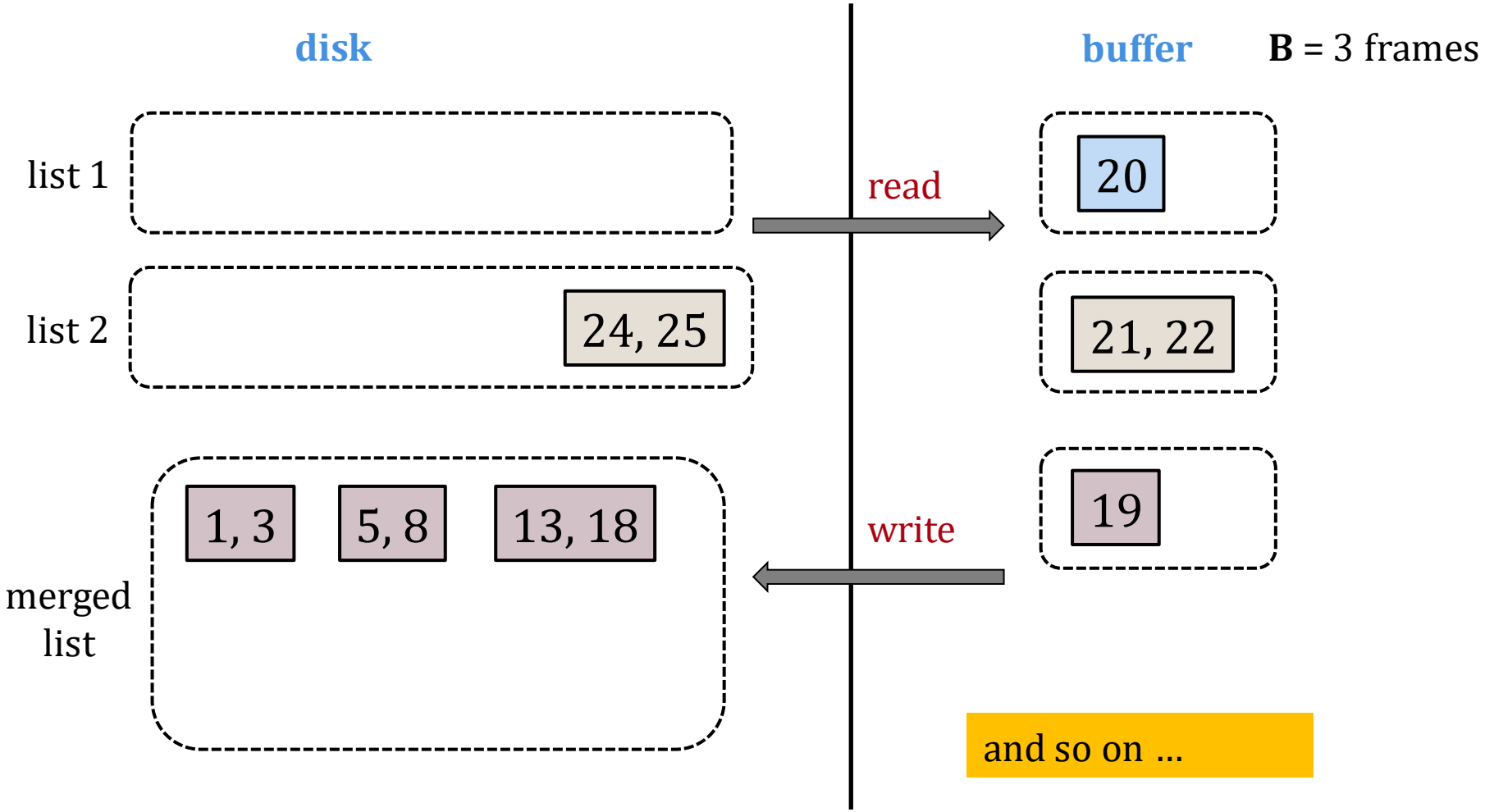
External Merge Algorithm



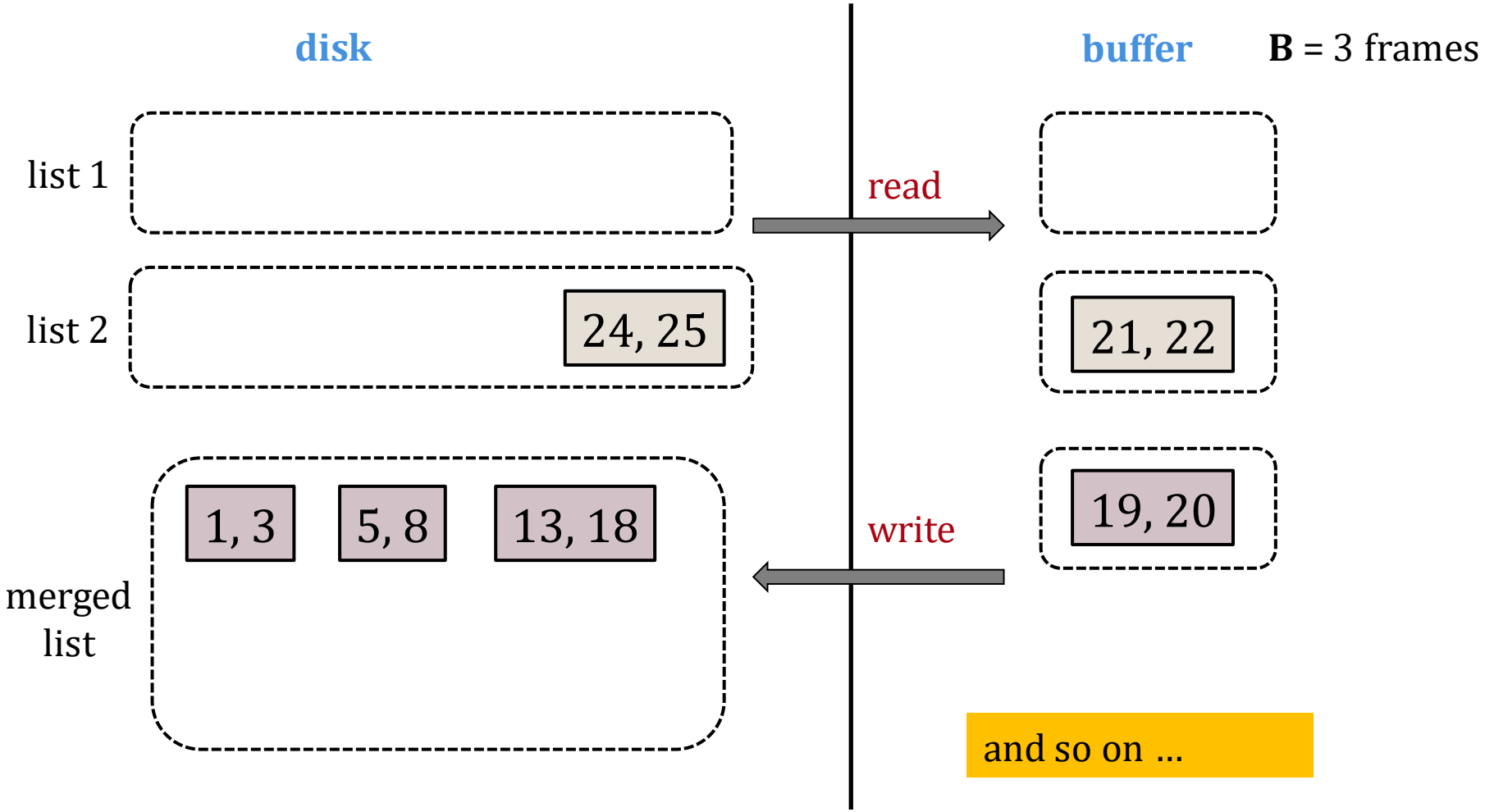
External Merge Algorithm



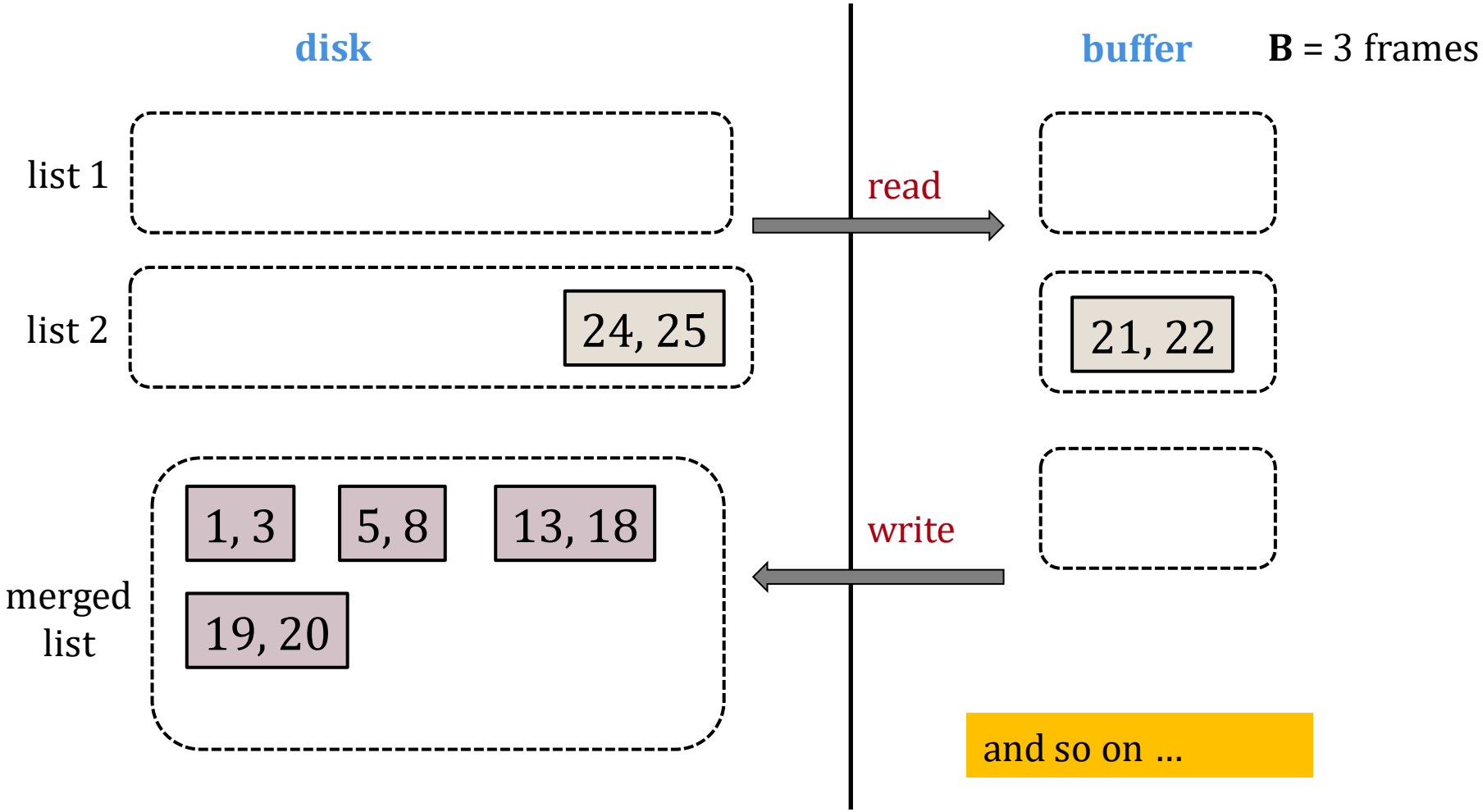
External Merge Algorithm



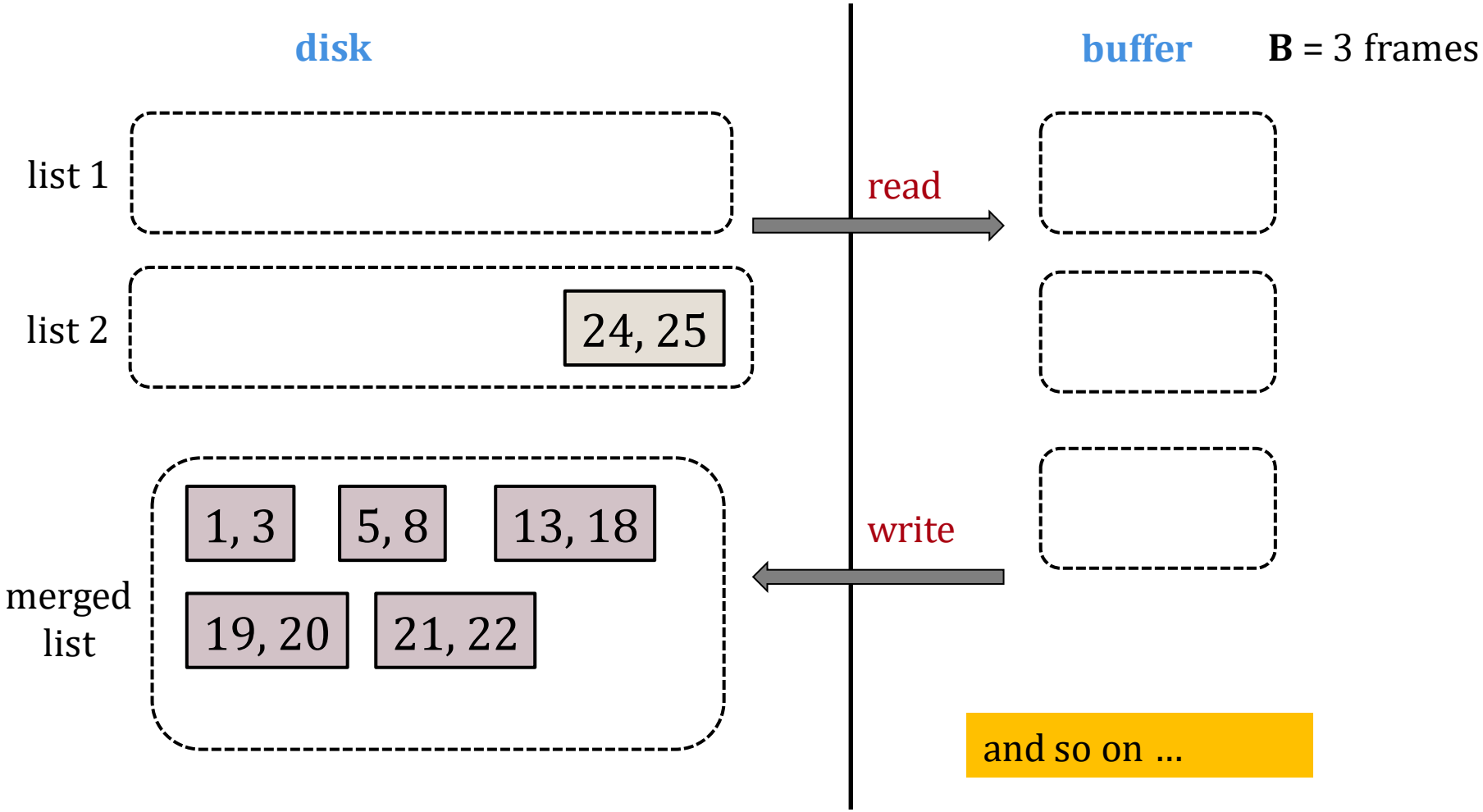
External Merge Algorithm



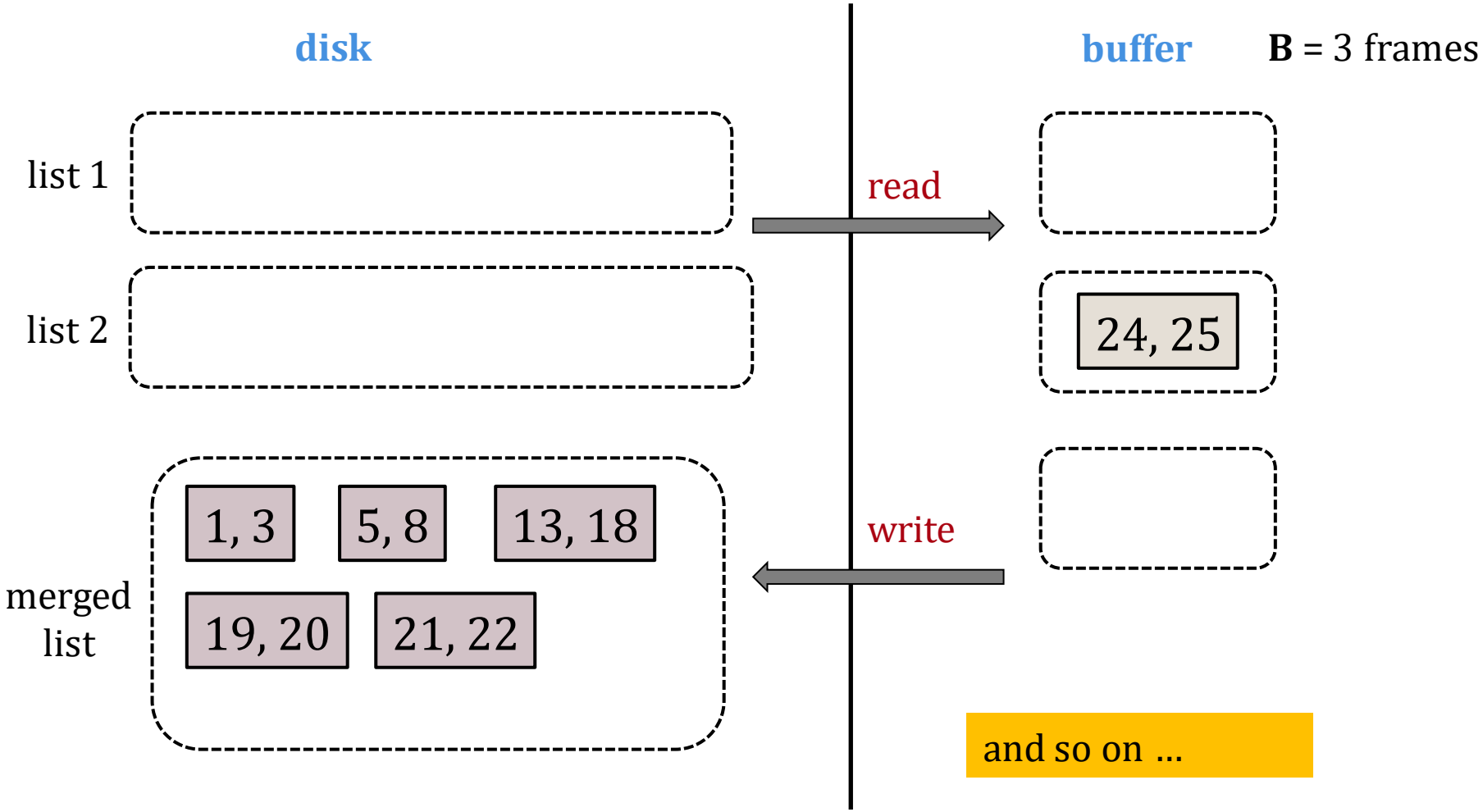
External Merge Algorithm



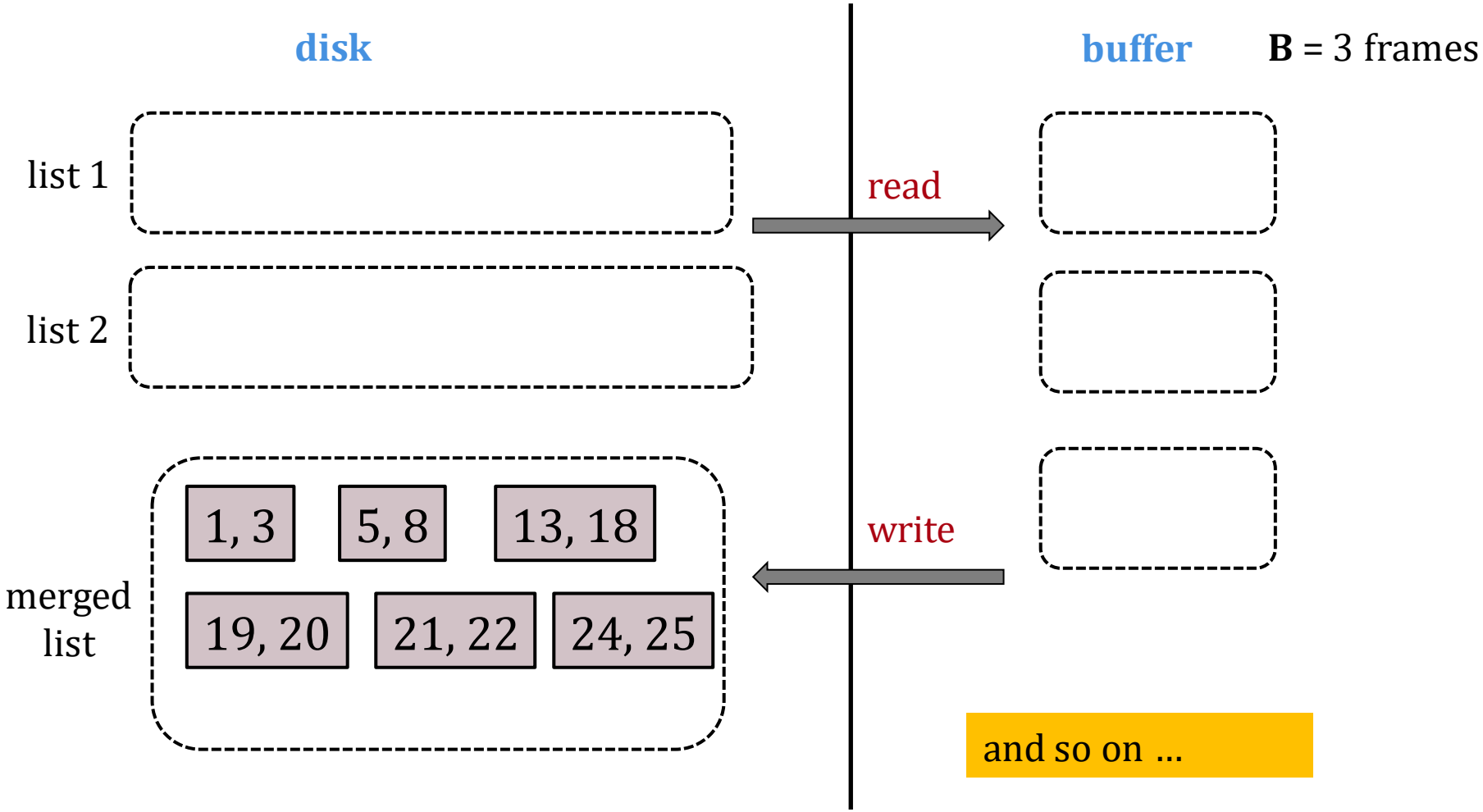
External Merge Algorithm



External Merge Algorithm



External Merge Algorithm



External Merge Cost

We can merge 2 sorted lists of *M* and *N* pages using 3 buffer frames with

I/O cost = *2 (M+N)*

When we have *B+1* buffer pages, we can merge *B* lists with the same I/O cost.

$$\text{I/O Cost} = 2 \times (N_1 + N_2 + \ldots + N_B)$$

Outline

Index concurrency control

External merge

External merge-sort

- **2-way merge sort**
- Multi-way merge sort

The External Sorting Problem

B available pages in buffer pool

A relation R of size **N** pages (where **N** > **B**)

SORTING: Output the same relation sorted on a given attribute

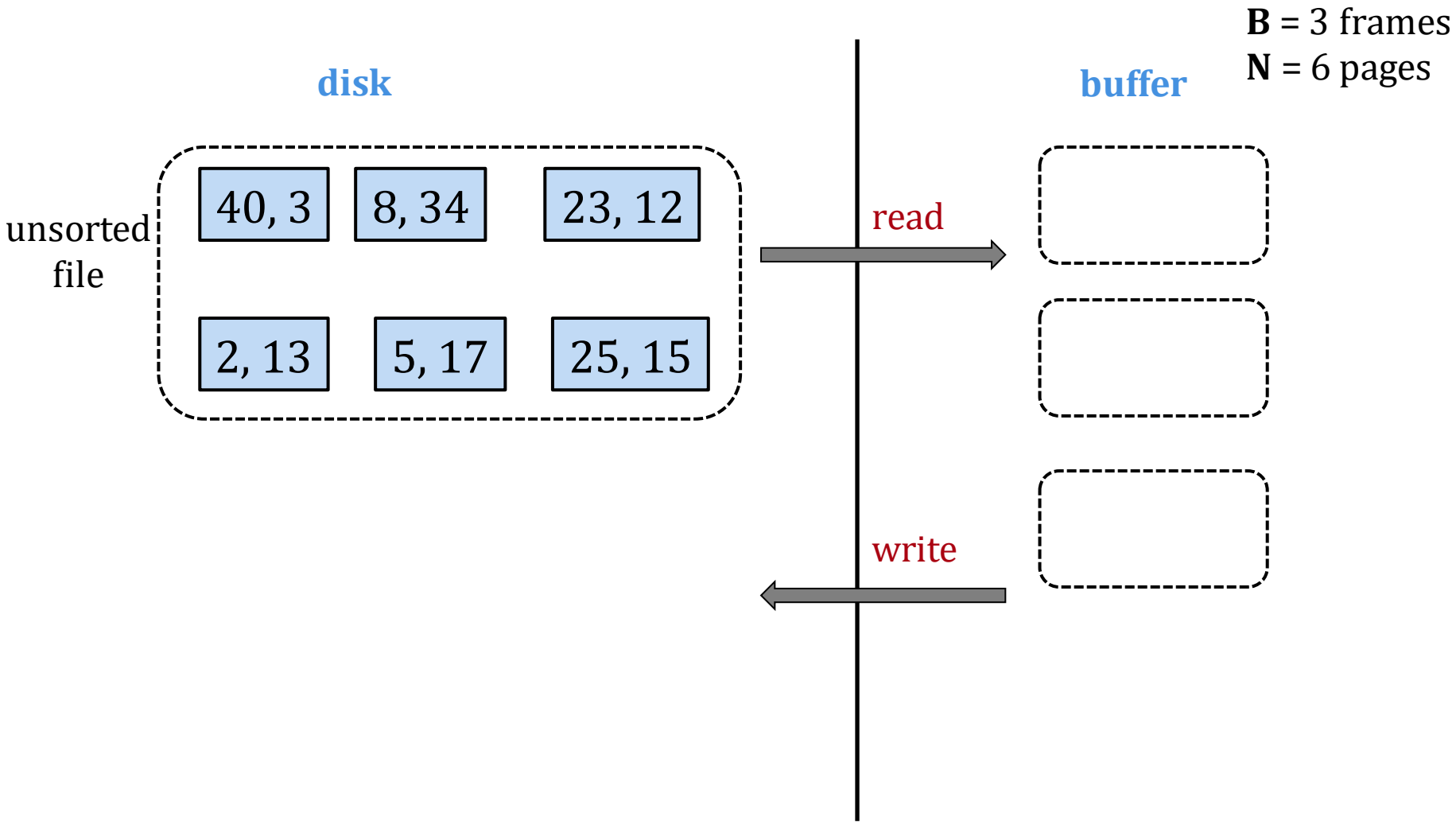
Key Idea

Split into chunks small enough to sort in memory (called runs)

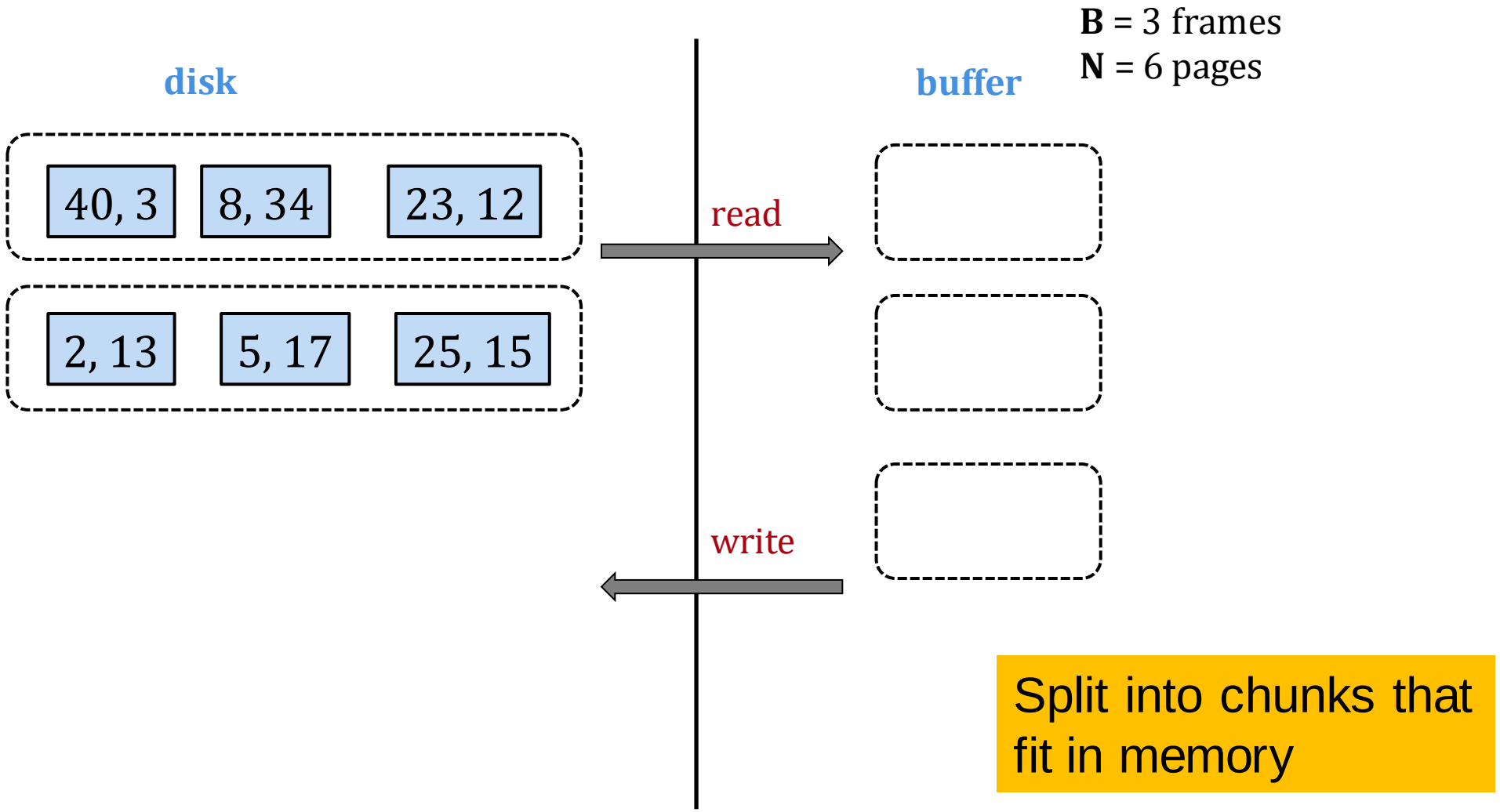
Merge groups of runs using the external merge algorithm

Keep merging the resulting runs (each time is called a pass) until left with a single sorted file

2-Way Merge Sort

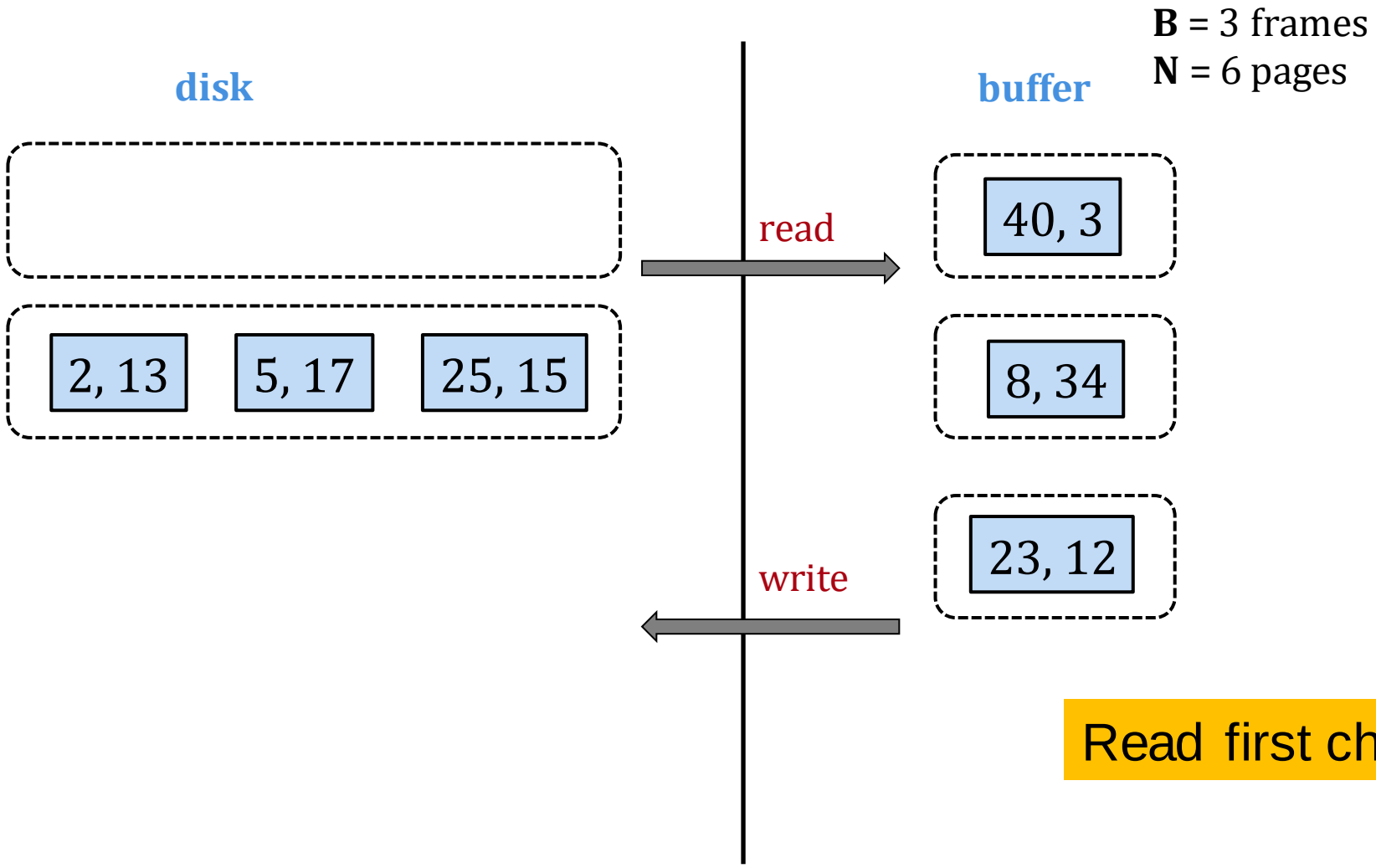


2-Way Merge Sort



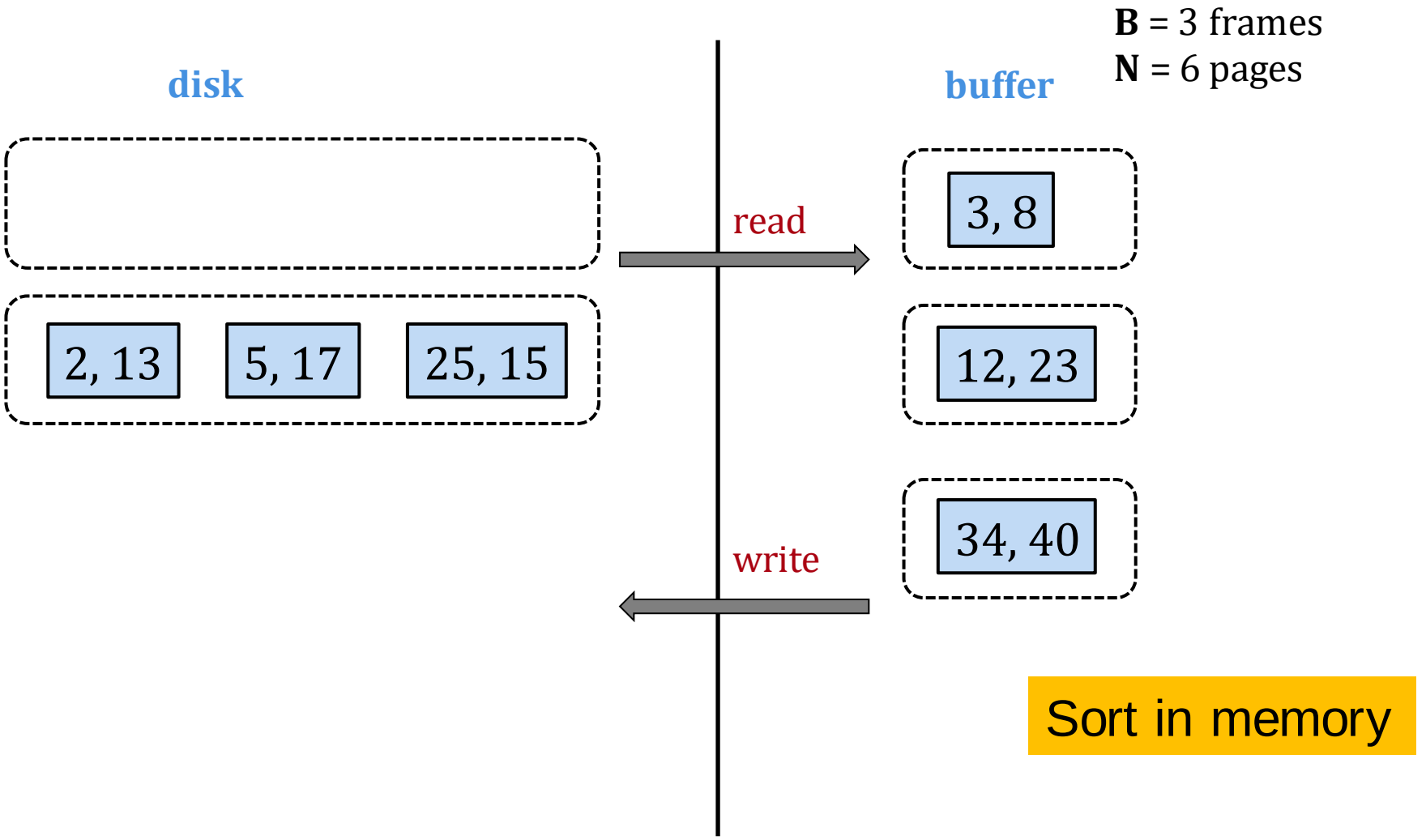
Split into chunks that fit in memory

2-Way Merge Sort

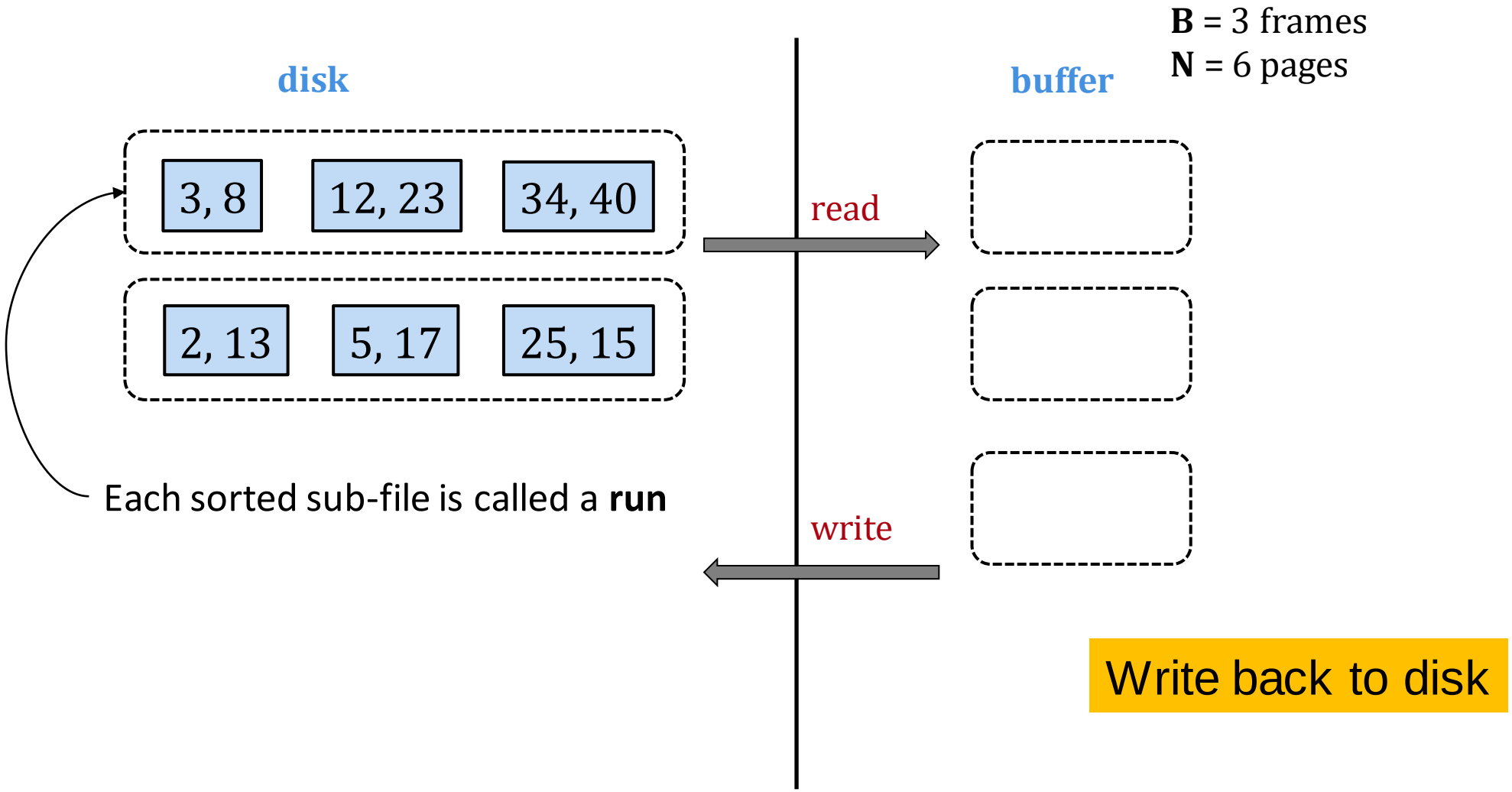


Read first chunk in memory

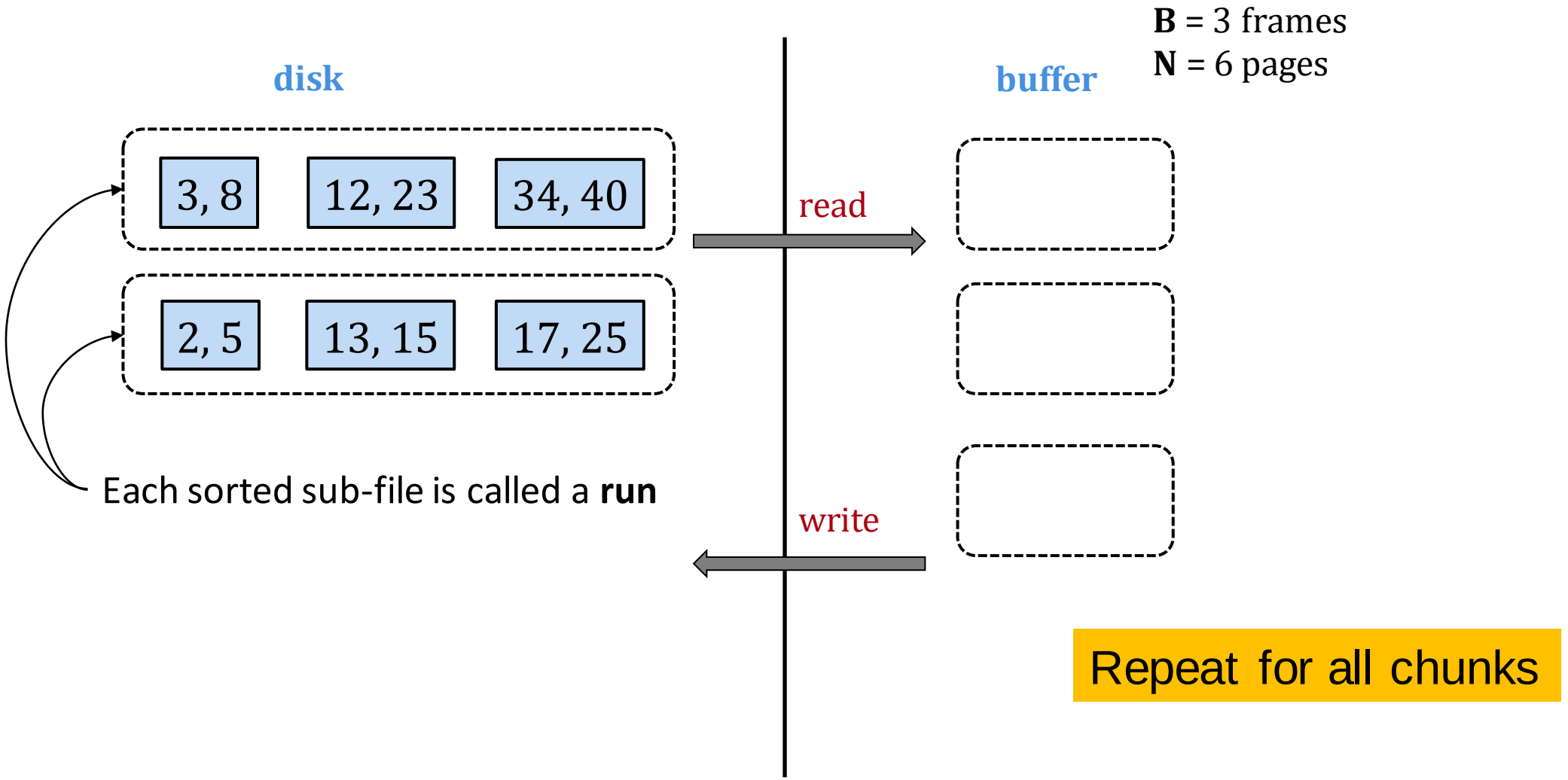
2-Way Merge Sort



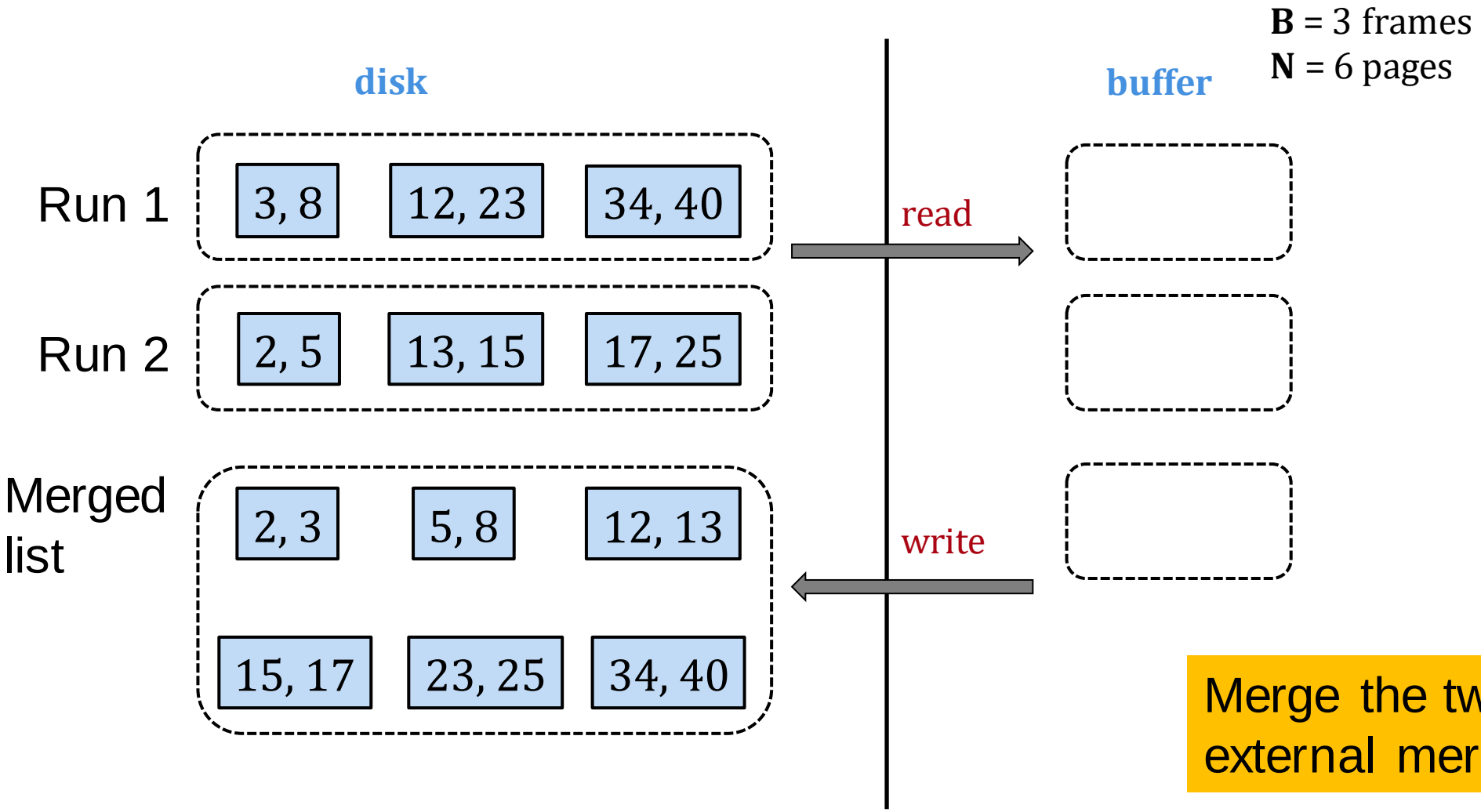
2-Way Merge Sort



2-Way Merge Sort



2-Way Merge Sort



Merge the two runs using external merge

I/O Cost

B = 3 buffer pages, **N** = 6 pages

Pass **0**: creating the first runs

- 1 read + 1 write for every page
- Total cost $= N * 2 = 12$ I/Os

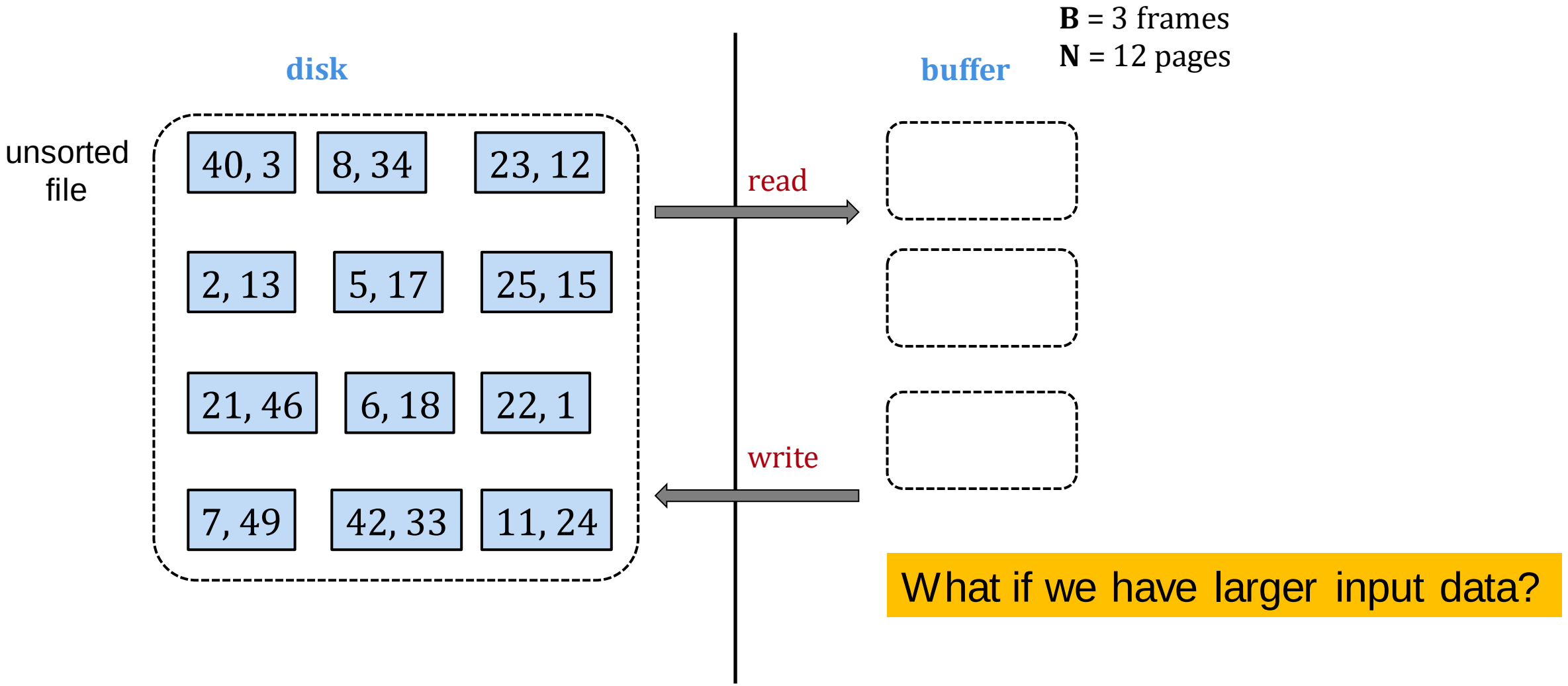
Pass **1**: external merge sort

- total cost = $N * 2 = 12$ I/Os

So **24** I/Os in total

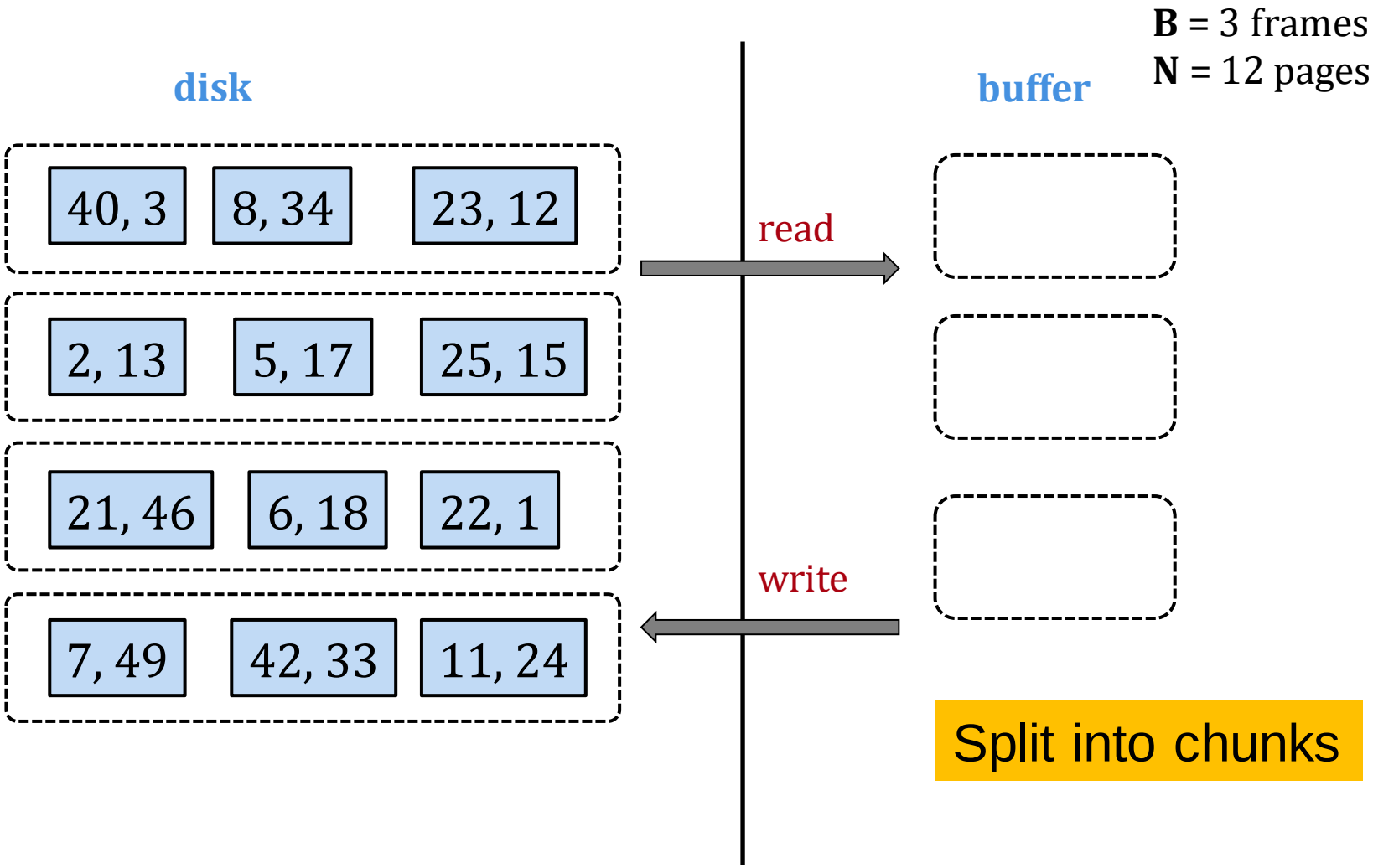
- Each page is read and written once for each pass

2-Way Merge Sort – Large Example

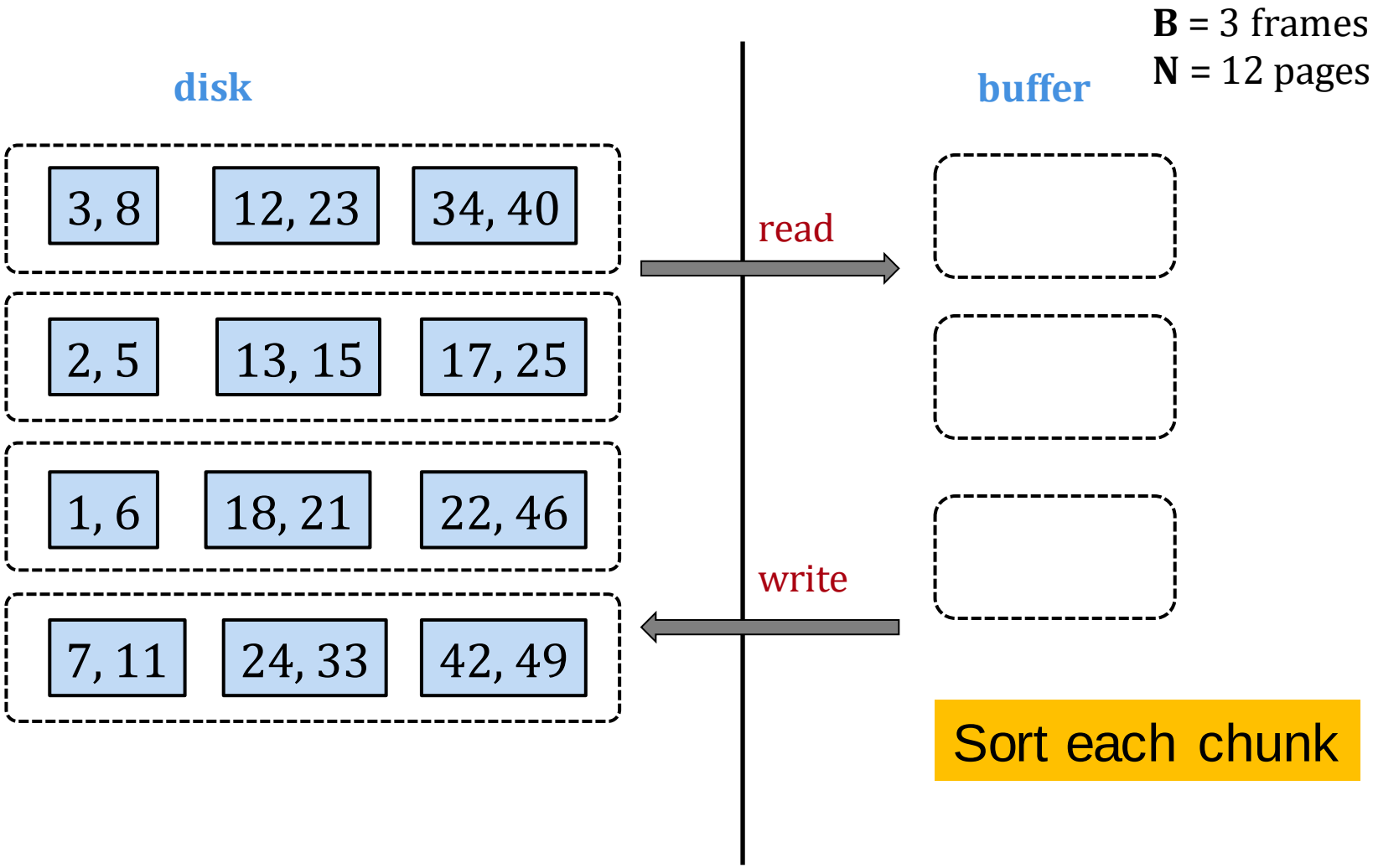


What if we have larger input data?

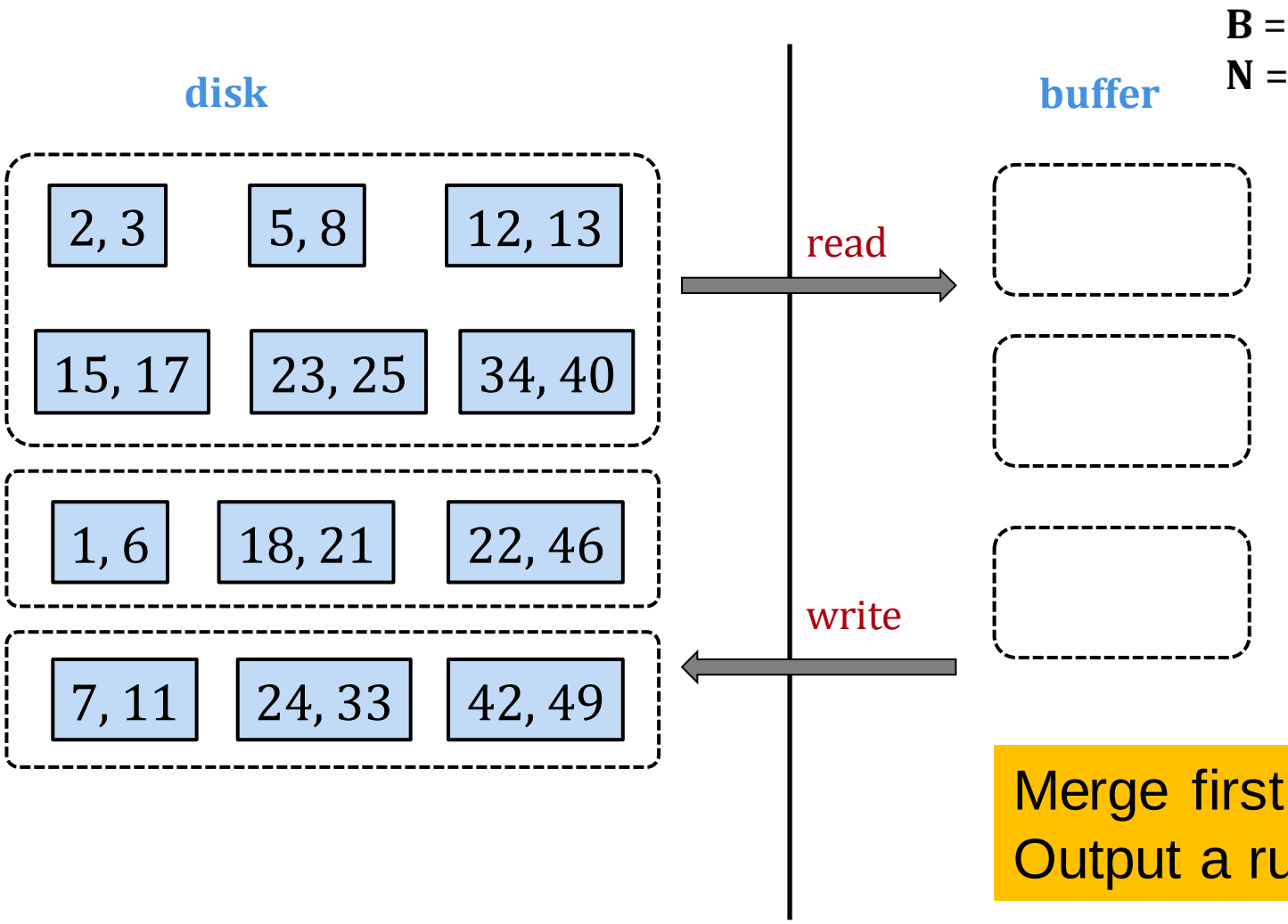
2-Way Merge Sort – Large Example



2-Way Merge Sort – Large Example

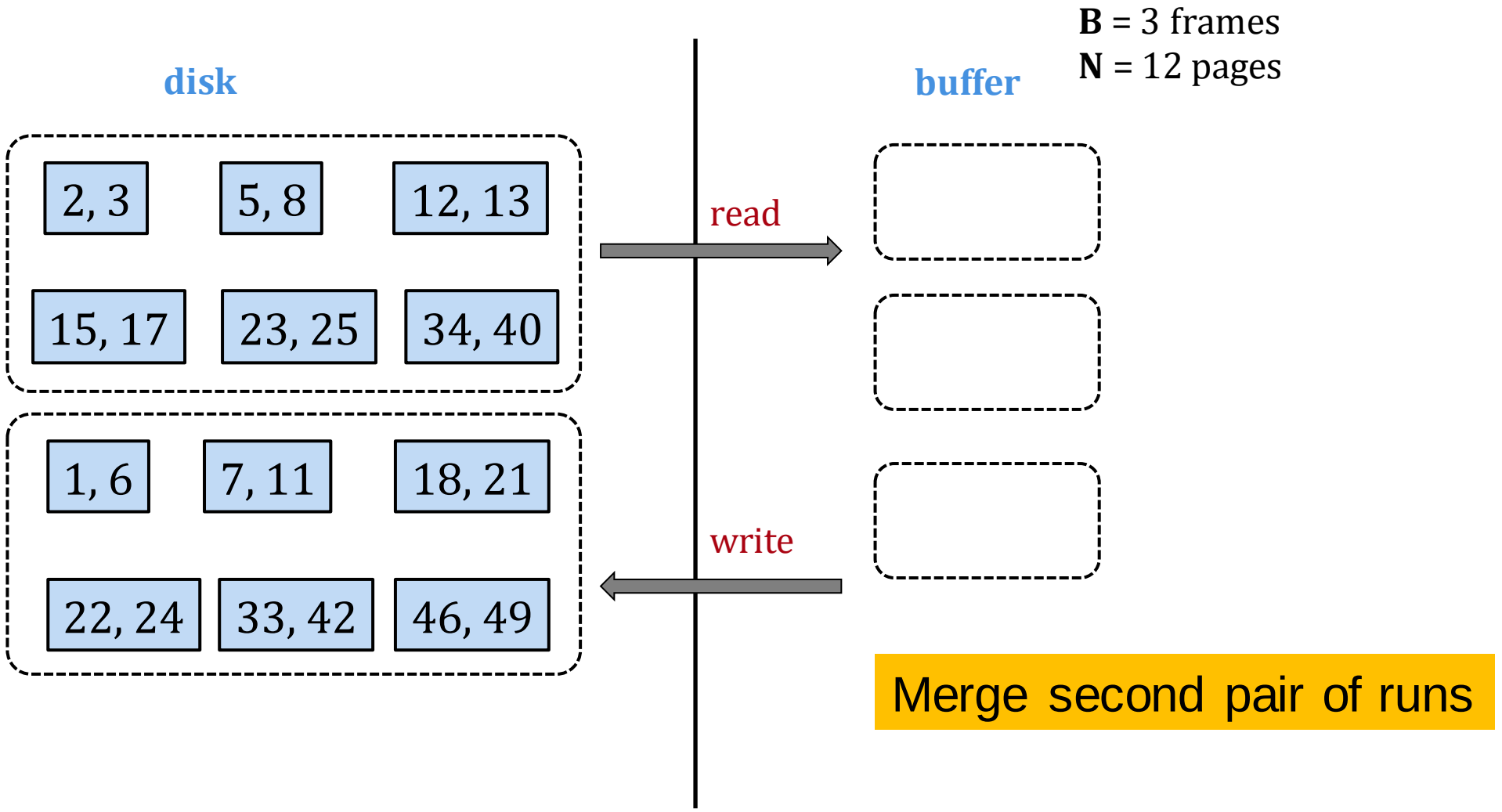


2-Way Merge Sort – Large Example

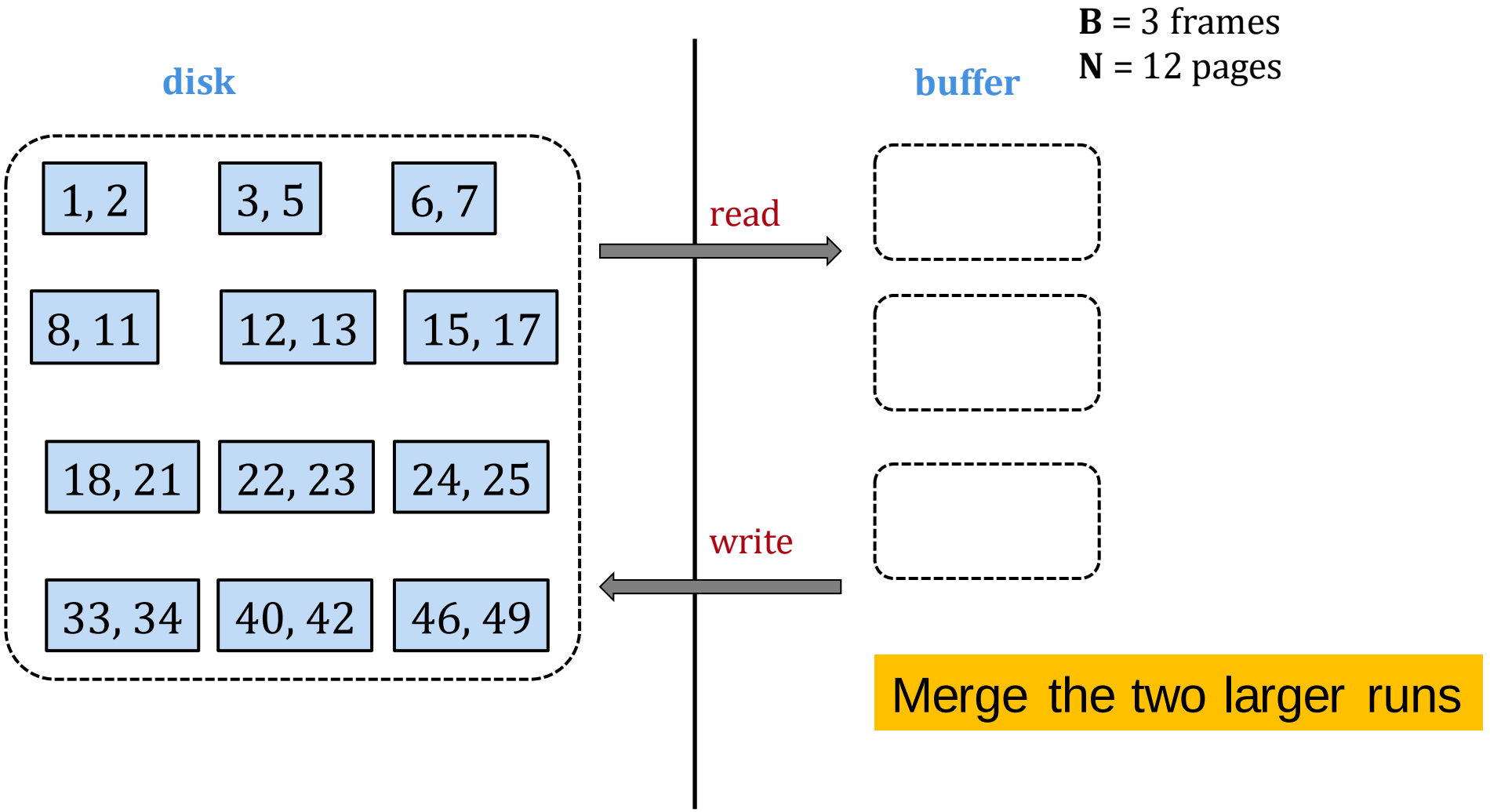


Merge first pair of runs
Output a run with twice the size

2-Way Merge Sort – Large Example



2-Way Merge Sort – Large Example



I/O Cost

B = 3 buffer pages, **N** = 12 pages

Pass **0**: creating the first runs

- 1 read + 1 write for every page
- Total cost $= N * 2 = 24$ I/Os

Pass **1**: external merge sort

- total cost = $N * 2 = 24$ I/Os

Pass **2**: external merge sort

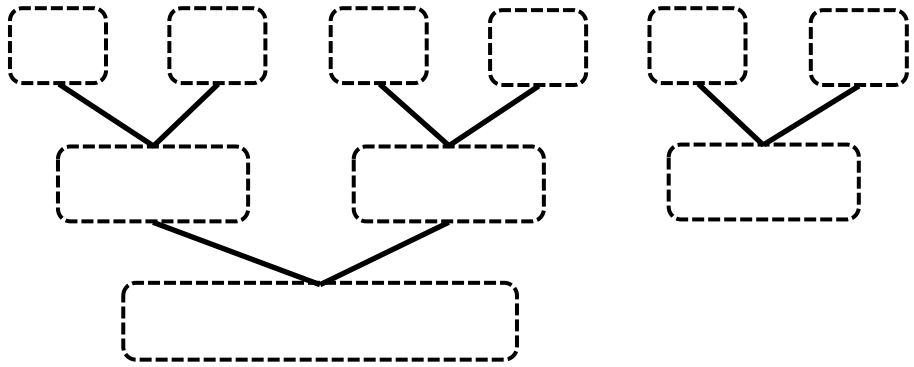
- total cost = $N * 2 = 24$ I/Os

So **72** I/Os in total

- Each page is read and written once for each pass

I/O Cost

For a merge sort with **B** = 3 buffer pages, **N** pages of input data



pass 0: N/B runs

pass 1: merge into N/B/2 runs

pass 2: merge into N/B/4 runs

We need $\left\lceil log_2 \frac{N}{B} \right\rceil + 1$ passes to sort the whole file

Each pass needs $2N$ I/Os

Total I/O cost = $\boldsymbol{2N\left(\left\lceil log_2 \frac{N}{B} \right\rceil + 1\right)}$

Can We Do Better?

The 2-way merge algorithm only uses 3 buffer pages

We can do better if we have more available memory!

Key idea: use as much of the available memory as possible in every pass (i.e., increase **B**)

- Reduce the number of passes and I/O

Outline

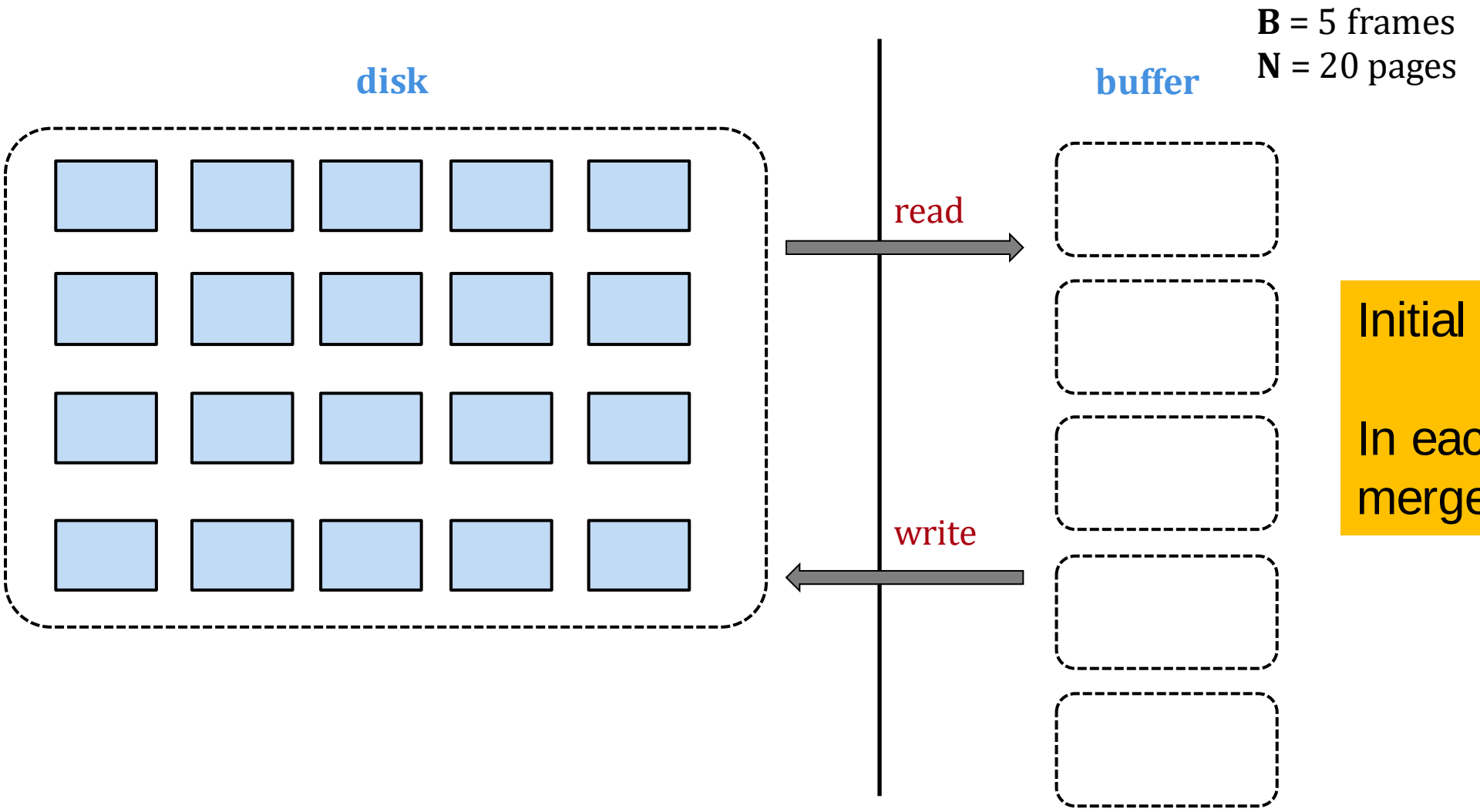
Index concurrency control

External merge

External merge-sort

- 2-way merge sort
- **Multi-way merge sort**

Multi-Way Merge Sort



Initial run length = B

In each pass, we can merge (B-1) runs

Multi-Way Merge Sort

Suppose we have $B \geq 3$ buffer pages available

Pass 0: length of each initial run = B

Pass 1: Merge (B-1) into one larger run

…

Each run incurs *2N* IOs

Total number of passes = $\left\lceil log_{B-1} \frac{N}{B} \right\rceil + 1$

Total I/O cost = $2N\left(\left\lceil log_{B-1} \frac{N}{B} \right\rceil + 1\right)$

Number of 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

Summary

Index concurrency control

External merge

External merge-sort

- 2-way merge sort
- Multi-way merge sort