CS 564: Database Management Systems

Lecture 22: Relational Operators II

Xiangyao Yu

3/13/2024

Midterm Exam Logistics

Mid-term exam

- March 20th, Wednesday
- 2:30–3:45pm (in-class)
- Please arrive 5 min early
- Paper-based, closed-book
- Cheat sheet allowed, US letter size (8.5 × 11 inches), double-sided

Previous years' exam questions have been released

Final exam is cumulative

- You will be tested on everything you have learned so far
- More focus on the second half of class

Module B3 Query Processing

Relational operators I

Relational operators II

Query optimization I

Query optimization II

Column Store

Outline

- Join
  - **Nested loop join**
  - Block nested loop join
  - Index nested loop join
  - Sort merge join
  - Hash join
- Aggregation

Nested Loop Join – Tuple Granularity

```
for each tuple t_r in R                          #outer loop
    for each tuple t_s in S                      #inner loop
        join t_r with t_s if the join condition is satisfied
```

I/O cost = $M_R + M_S \cdot T_R$

- M_R = number of pages in **R**
- M_S = number of pages in **S**
- T_R = number of tuples in **R**

Note that we ignore the cost of writing the output to disk!

Nested Loop Join – Page Granularity

for each page P_r in **R** #outer loop

  for each page P_s in **S** #inner loop

    join tuples in P_r with tuples in P_s

Nested Loop Join – Page Granularity

```
for each page Pr in R                                   #outer loop
   for each page Ps in S                                #inner loop
      for each tuple tr in Pr
         for each tuple ts in Ps
            join tr with ts if the join condition is satisfied
```

Nested Loop Join – Page Granularity

```
for each page Pr in R                          #outer loop
    for each page Ps in S                      #inner loop
        join tuples in Pr with tuples in Ps
```

I/O cost = $M_R + M_S \cdot \boldsymbol{M_R}$

- M_R = number of pages in **R**
- M_S = number of pages in **S**
- T_R = number of tuples in **R**

Requires more CPU computation for each loop iteration, but fewer iterations

But the total CPU computation is the same

Nested Loop Join – Page Granularity

```
for each page Pr in R                              #outer loop
    for each page Ps in S                          #inner loop
        join tuples in Pr with tuples in Ps
```

I/O cost $= M_R + M_S \cdot \boldsymbol{M_R}$

- M_R = number of pages in **R**
- M_S = number of pages in **S**
- T_R = number of tuples in **R**

Which relation should be the outer relation?

- The smaller of the two relations

How many buffer frames do we need?

- 3 frames suffice!

Outline

- Join
  - Nested loop join
  - **Block nested loop join**
  - Index nested loop join
  - Sort merge join
  - Hash join
- Aggregation

Block Nested Loop Join

```
for each block of (B-2) pages in R                    #outer loop
   for each page P_S in S                             #inner loop
      join tuples in the block with tuples in P_s
```

B = number of pages in buffer pool

I/O cost = $M_R + M_S \cdot \left\lceil \frac{M_R}{B-2} \right\rceil$

– If R fits in memory, the I/O cost becomes $M_R + M_S$

Block Nested Loop Join

```
for each block of (B-2) pages in R              #outer loop
    for each page Ps in S                       #inner loop
        join tuples in the block with tuples in Ps
```

B = number of pages in buffer pool

Does blocking require more overall computation?

I/O cost = $M_R + M_S \cdot \left\lceil \frac{M_R}{B-2} \right\rceil$

– If R fits in memory, the I/O cost becomes $M_R + M_S$

Nested Loop Join – Summary

	I/O cost	Assuming M_R=500, M_S=1000, T_R=50,000, B=12
Tuple granularity	$M_R + M_S \cdot T_R$	500+5×10^7 I/Os
Page granularity	$M_R + M_S \cdot \boldsymbol{M_R}$	500,500 I/Os
Block	$M_R + M_S \cdot \left\lceil \frac{\boldsymbol{M_R}}{\boldsymbol{B-2}} \right\rceil$	50,500 I/Os

Outline

- Join
  - Nested loop join
  - Block nested loop join
  - **Index nested loop join**
  - Sort merge join
  - Hash join
- Aggregation

Index Nested Loop Join

```
for each tuple t_R in R
   probe the index of S to retrieve matching tuples
```

S has an index on the join attribute

I/O cost = $M_R + |T_R| \cdot I^*$

– I^* is the I/O cost of searching an index, and depends on the type of index and whether it is clustered or not

Outline

- Join
  - Nested loop join
  - Block nested loop join
  - Index nested loop join
  - **Sort merge join**
  - Hash join
- Aggregation

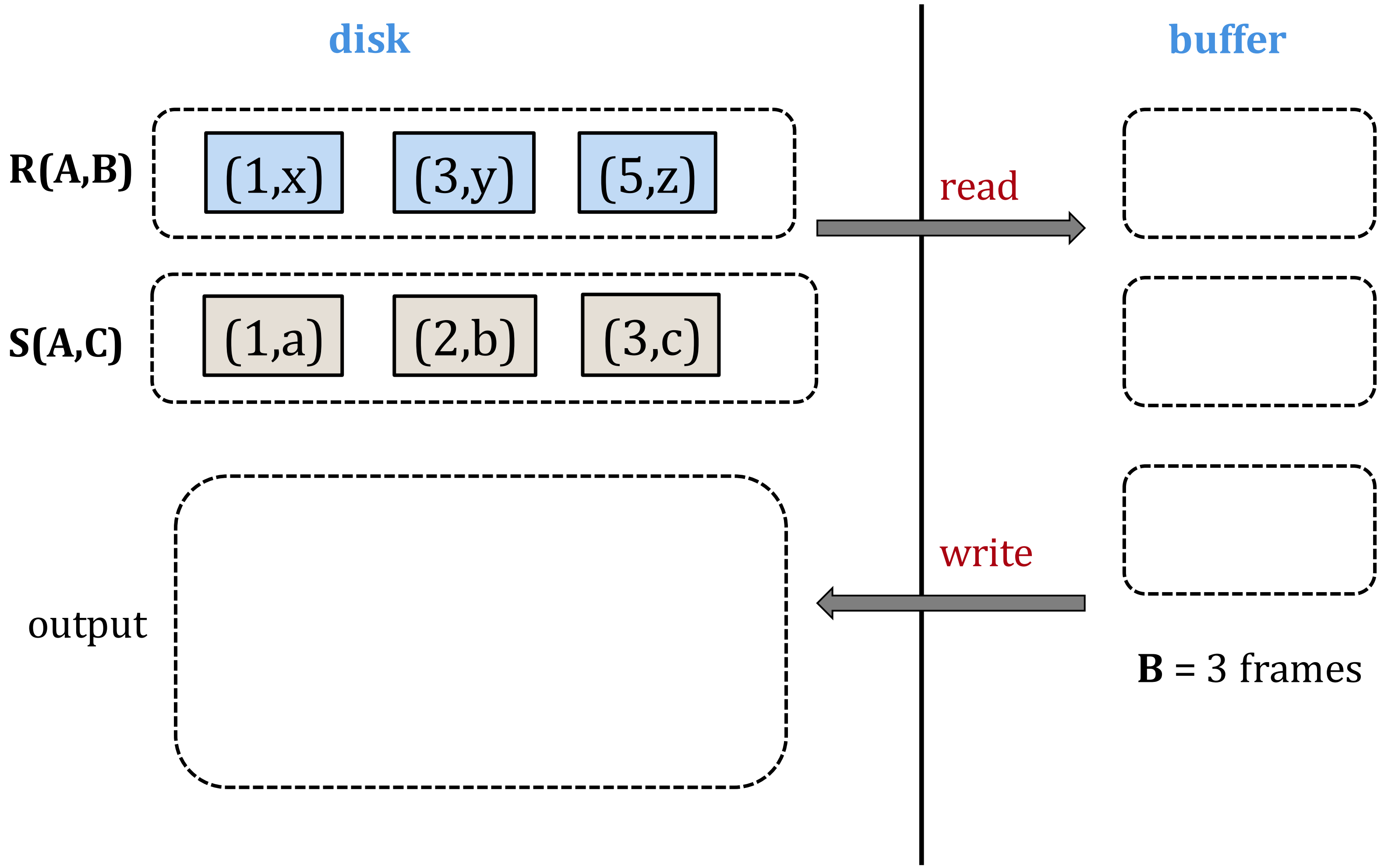
Sort Merge Join – Basic Version

Step 1: Sort **R** and **S** on the join attribute using external merge sort

Step 2: Read the sorted relations in the buffer and merge

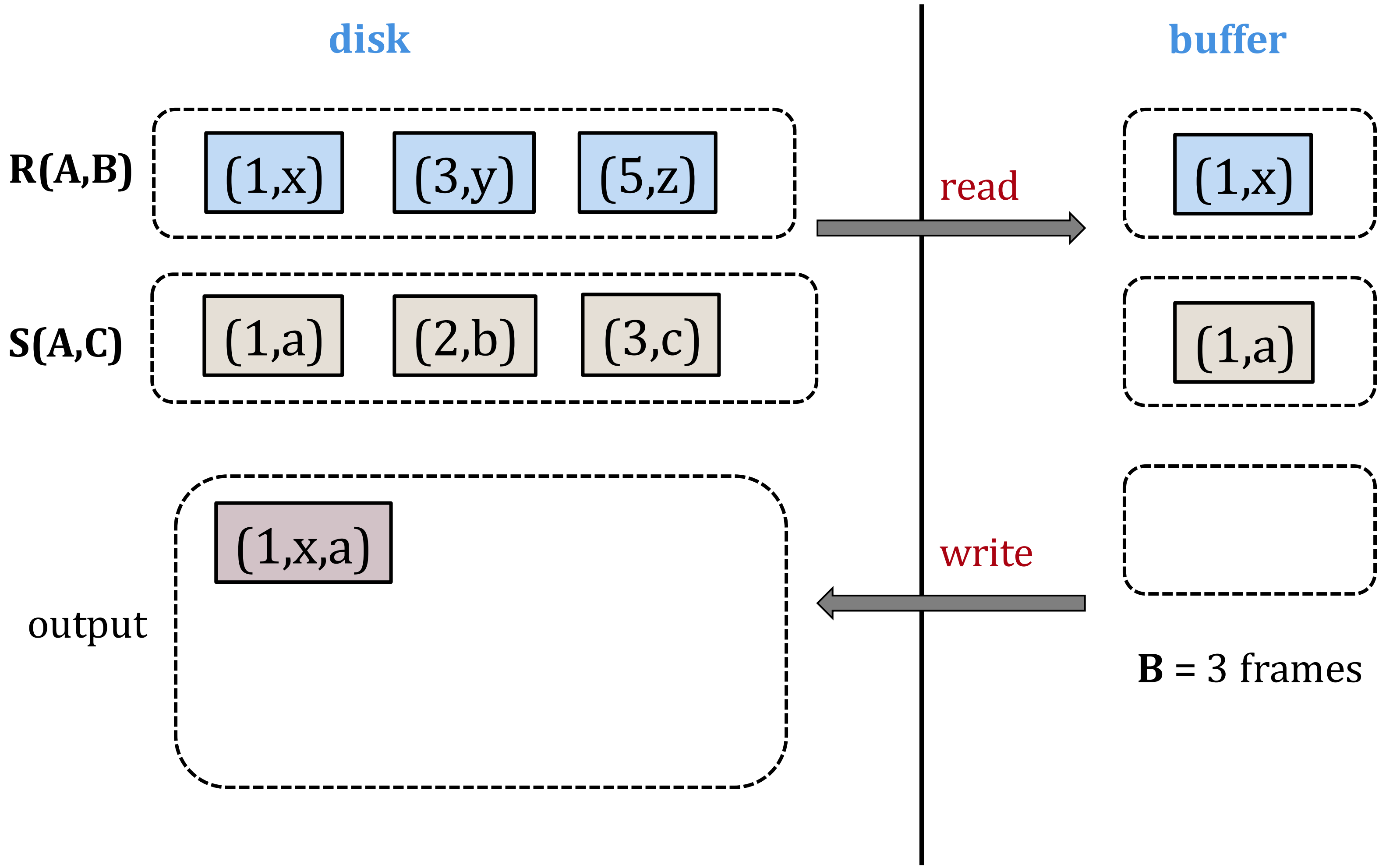
If **R** and **S** are already sorted on the join attribute, we can skip the first step

Merge Example



R and **S** are already sorted on the join attribute

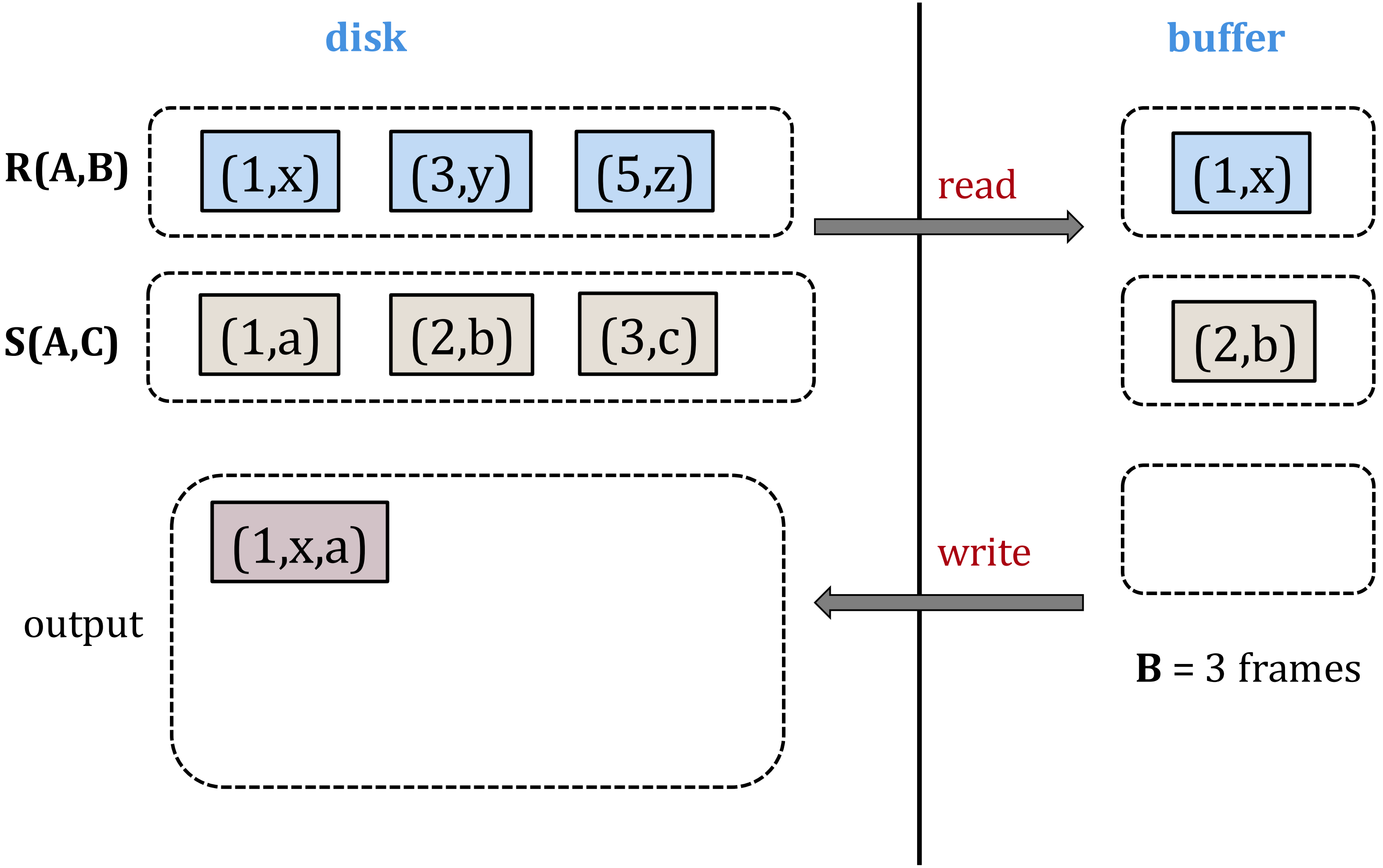
Merge Example



Read 1 page from each list merge the tuples

Matching tuples are written out

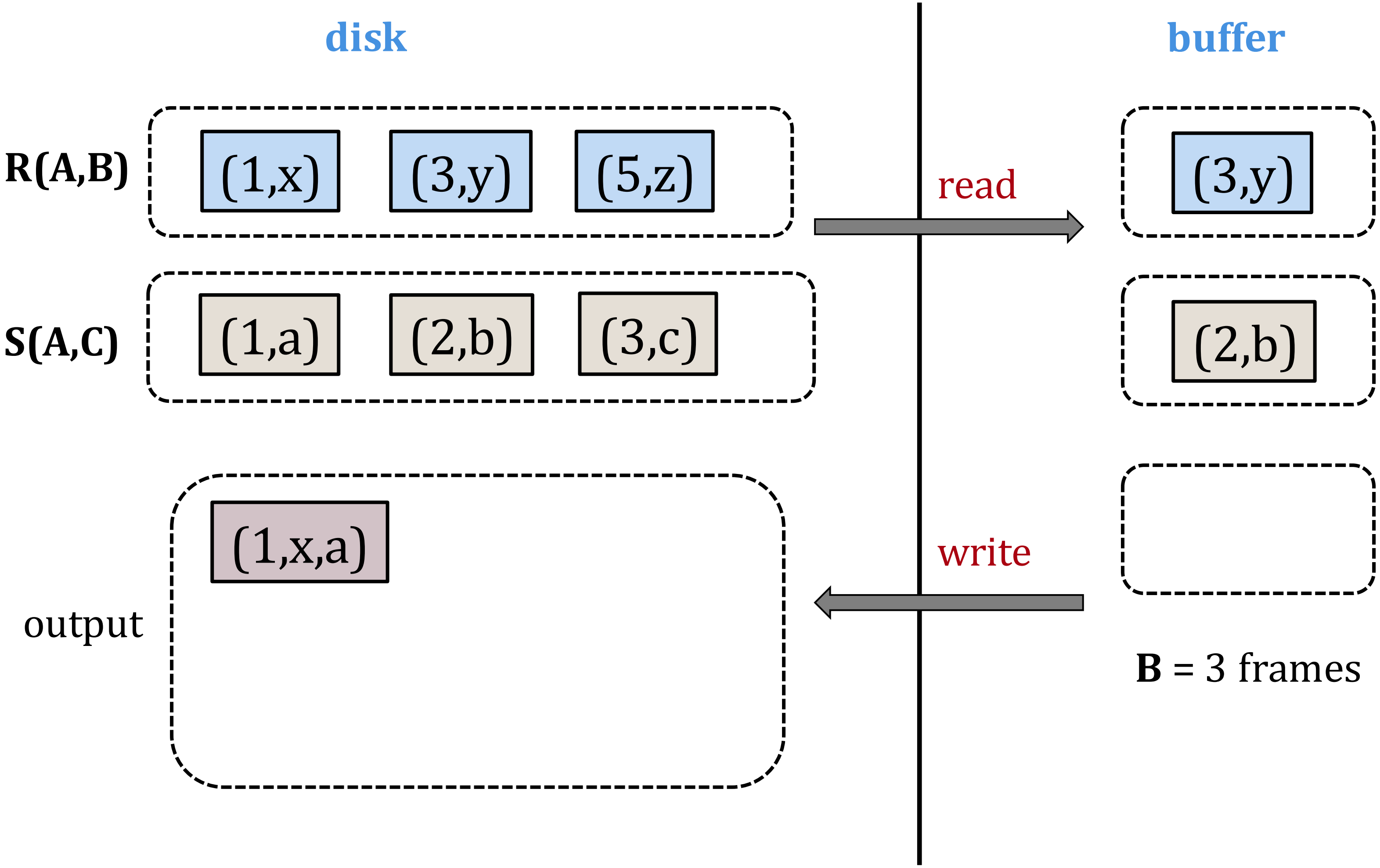
Merge Example



Read in next page

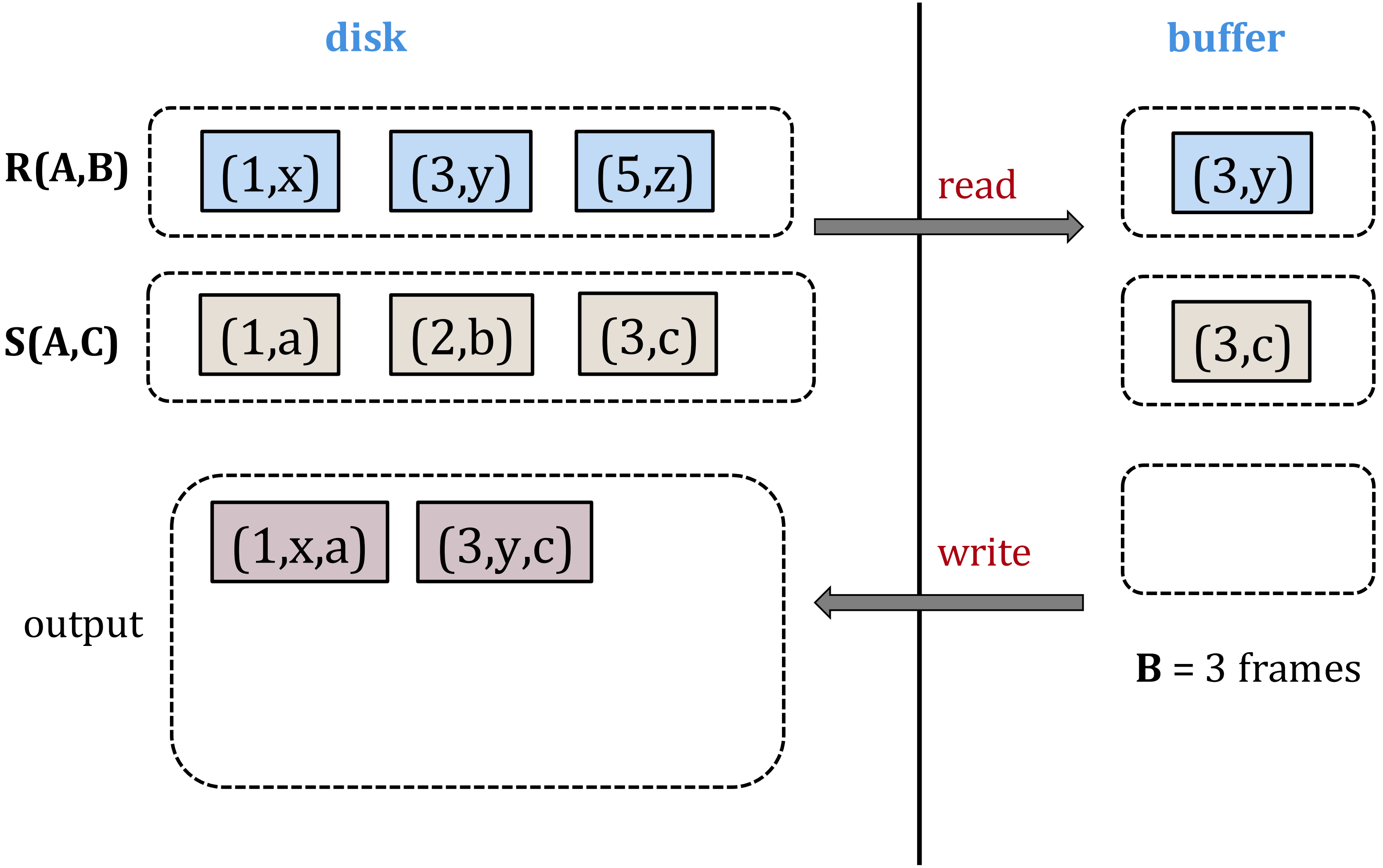
Since 1<2, next page should be read from R

Merge Example



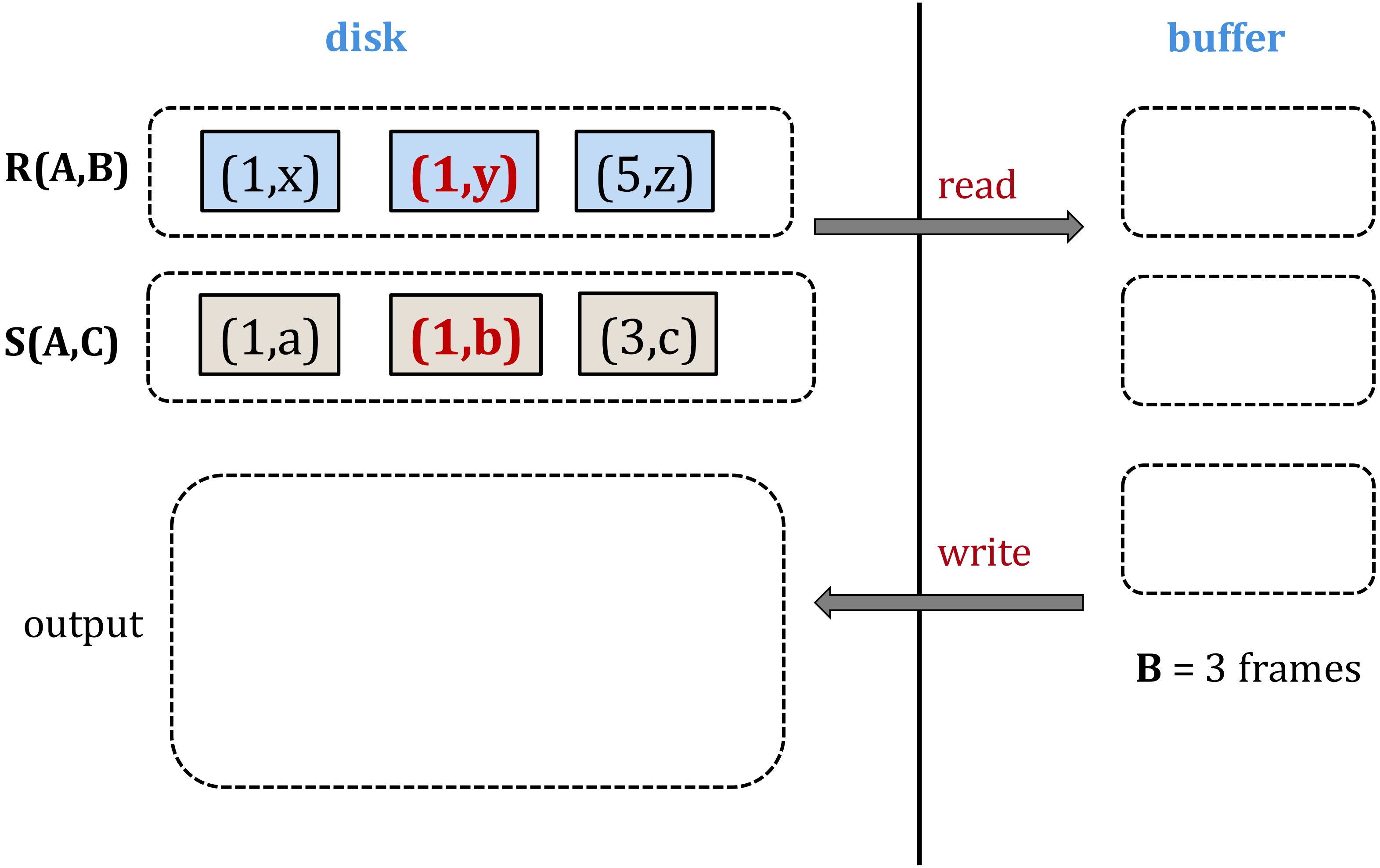
Since 2<3, next page should be read from S

Merge Example



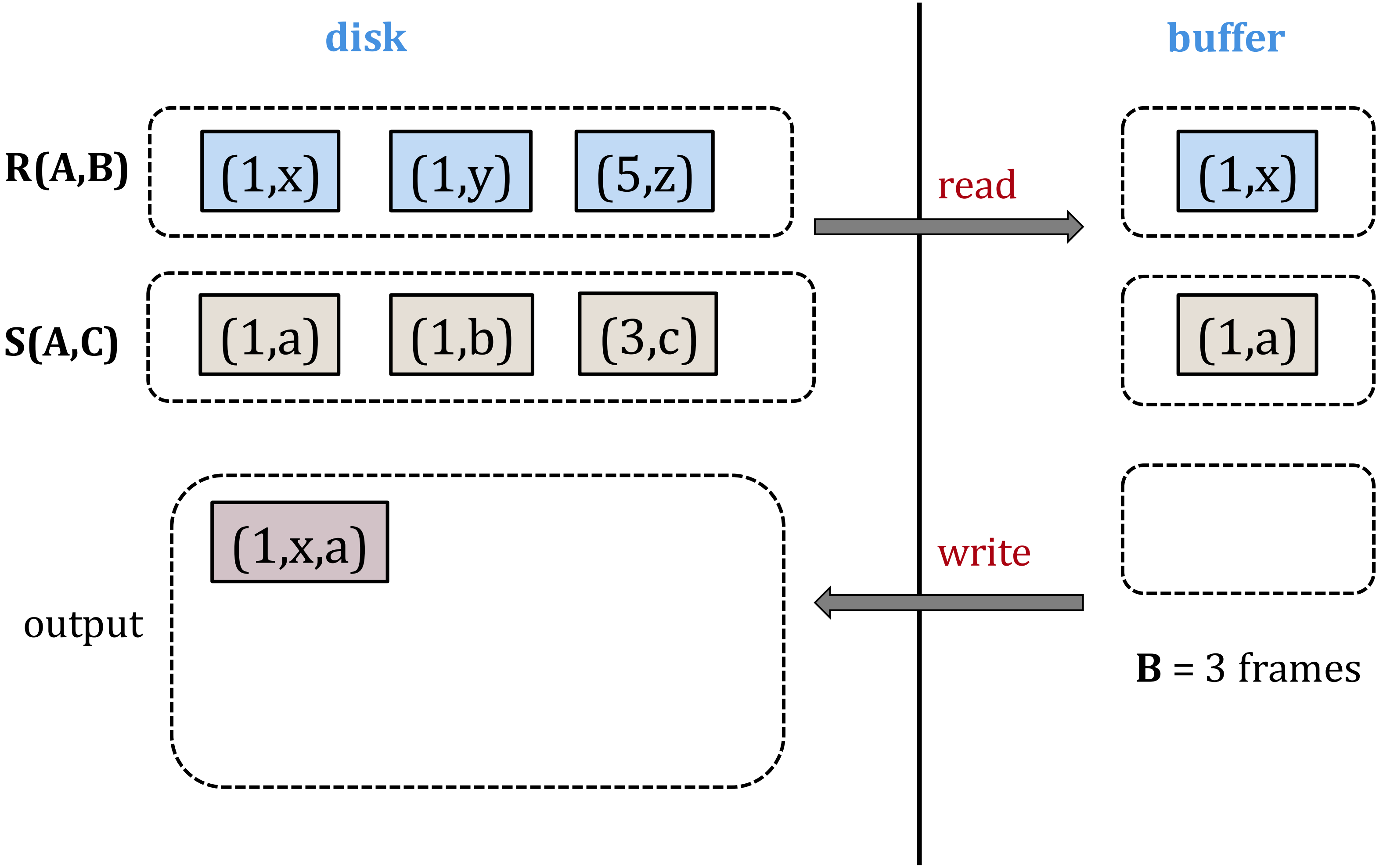
Reaching the end of relation S, algorithm terminates

Merge Example – Duplicates

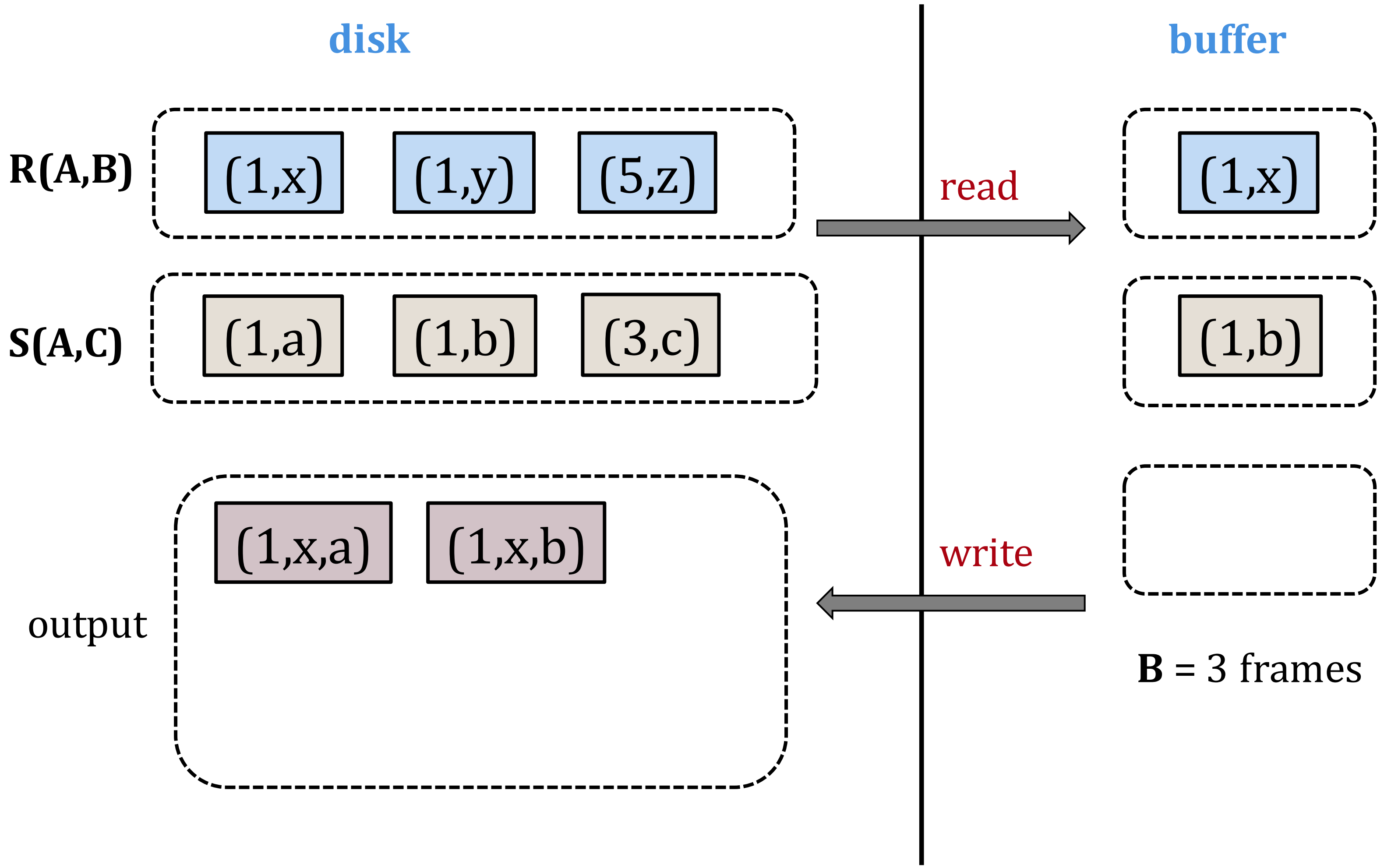


Multiple records share the same join key

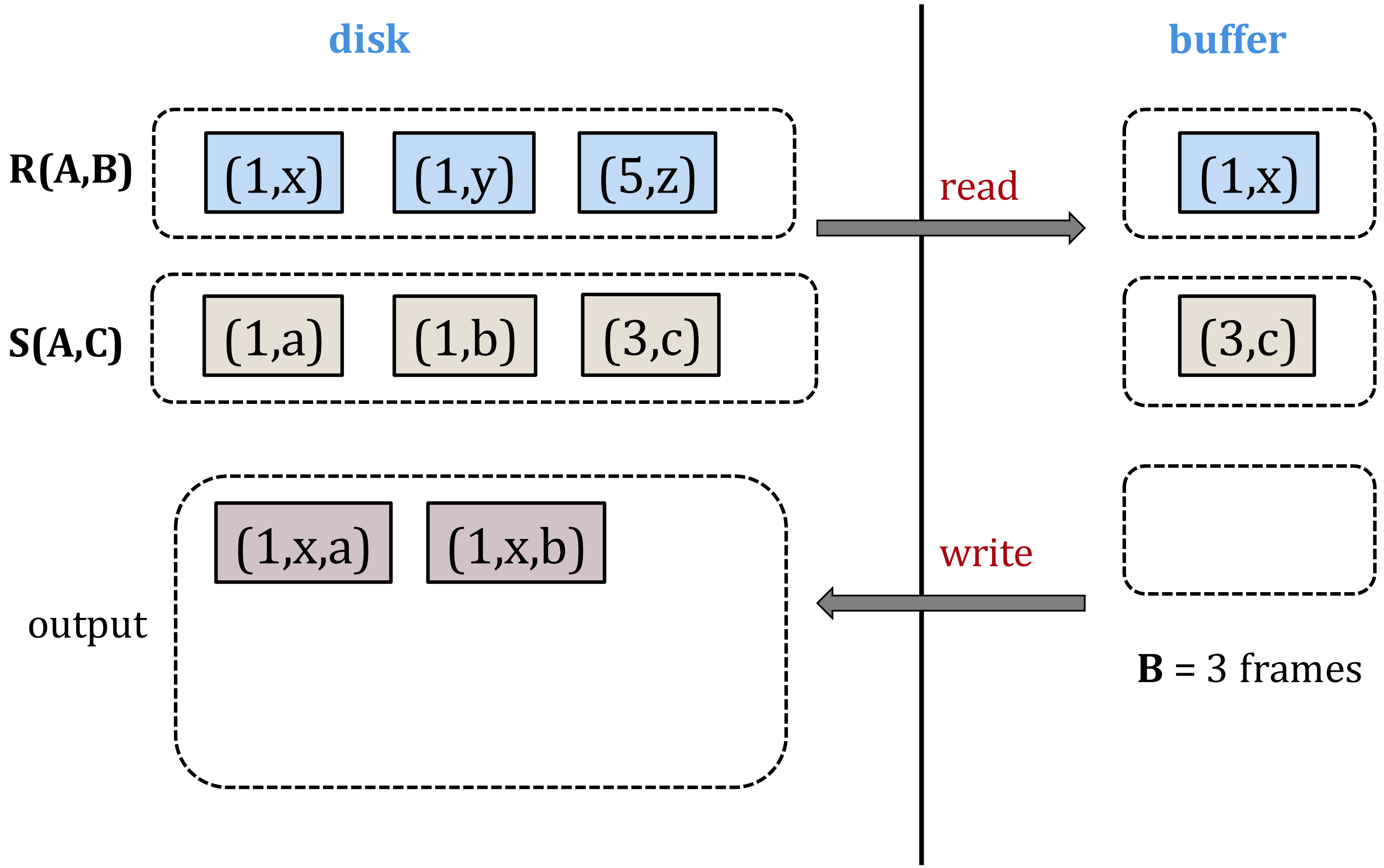
Merge Example – Duplicates



Merge Example – Duplicates

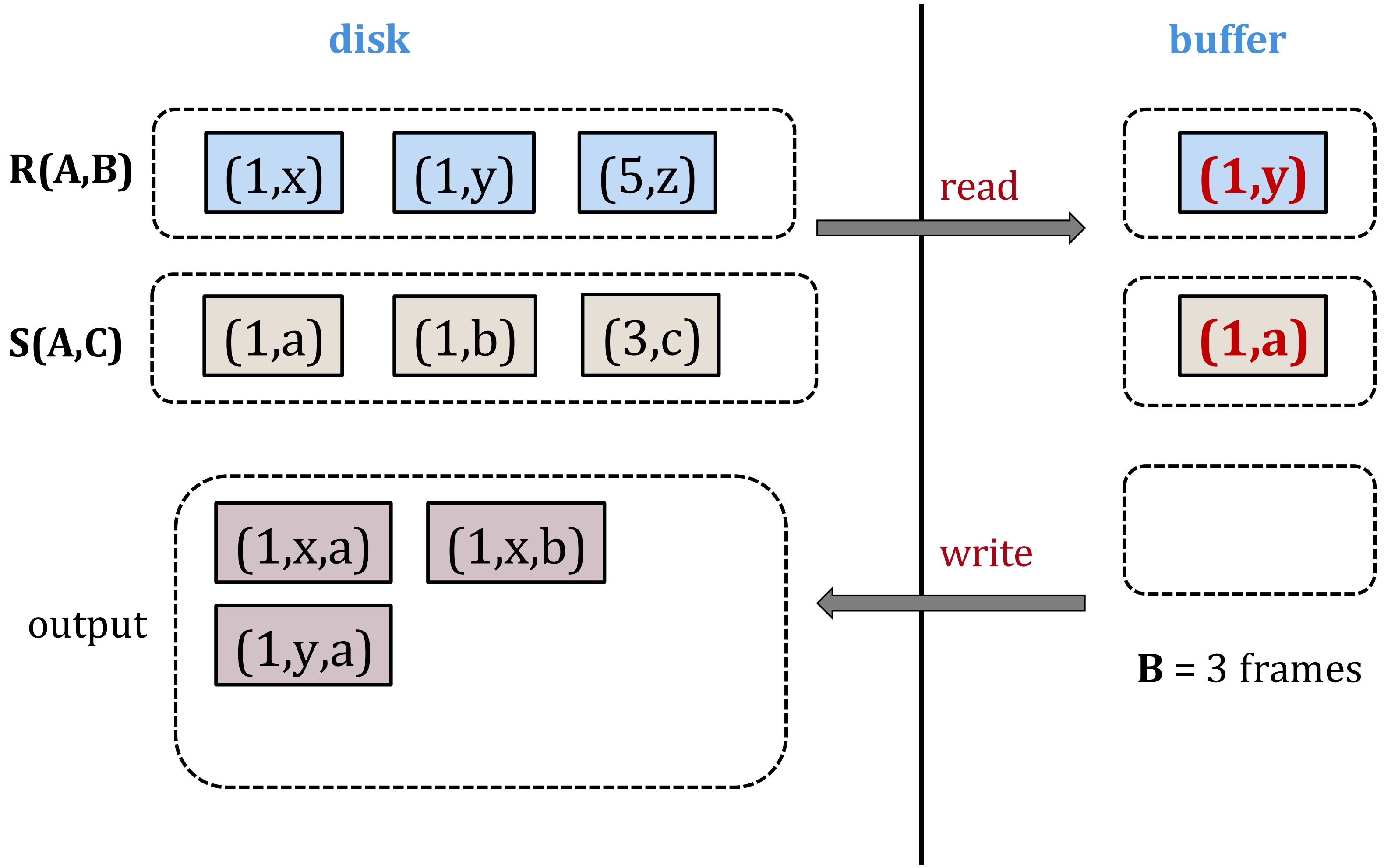


Merge Example – Duplicates



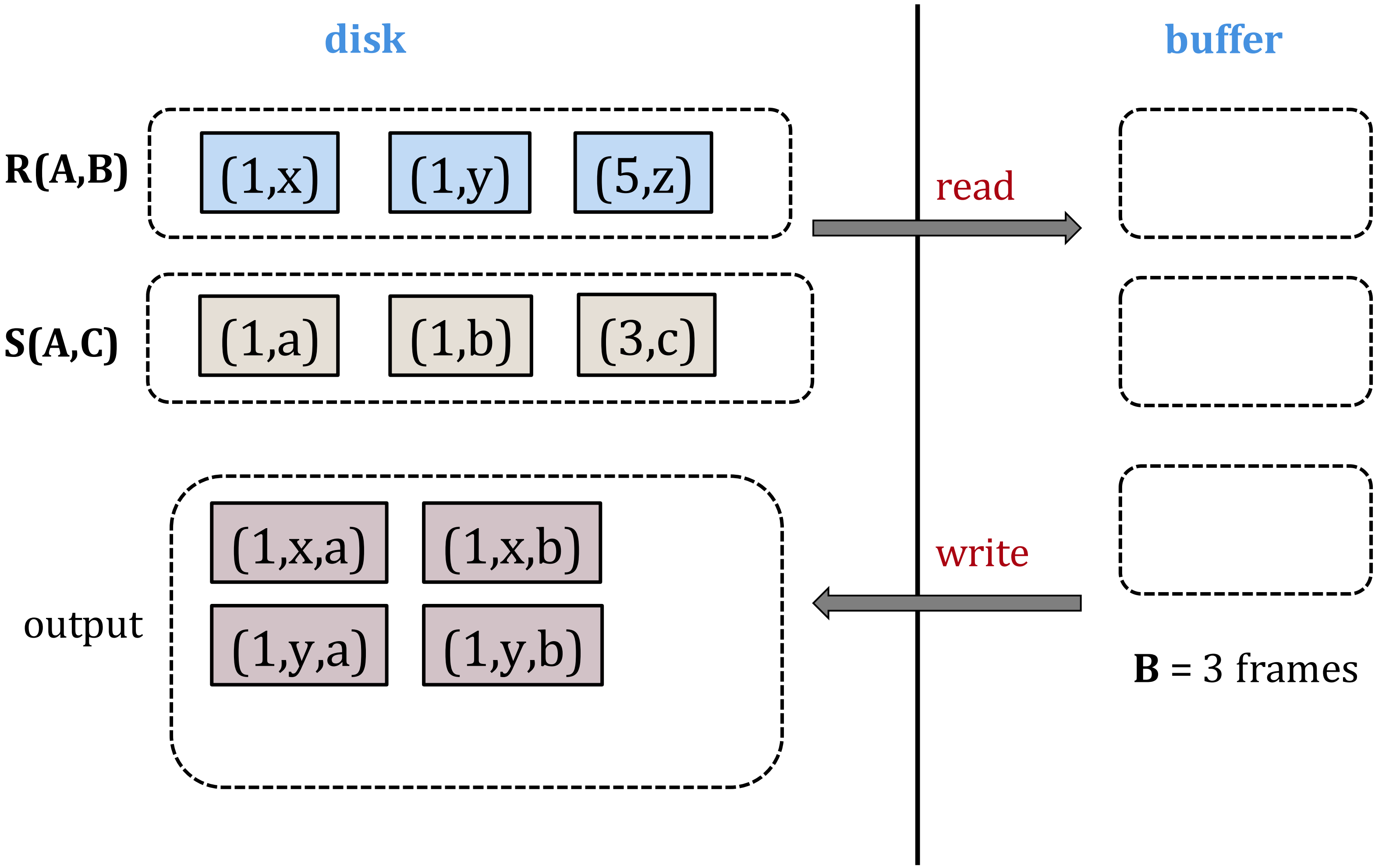
We need to **backup** to compute the full result

Merge Example – Duplicates



We need to **backup** to compute the full result

Merge Example – Duplicates



Final join output

Sort-Merge Join – I/O Cost

If there is no backup, the I/O cost of read + merge is only $M_R + M_S$

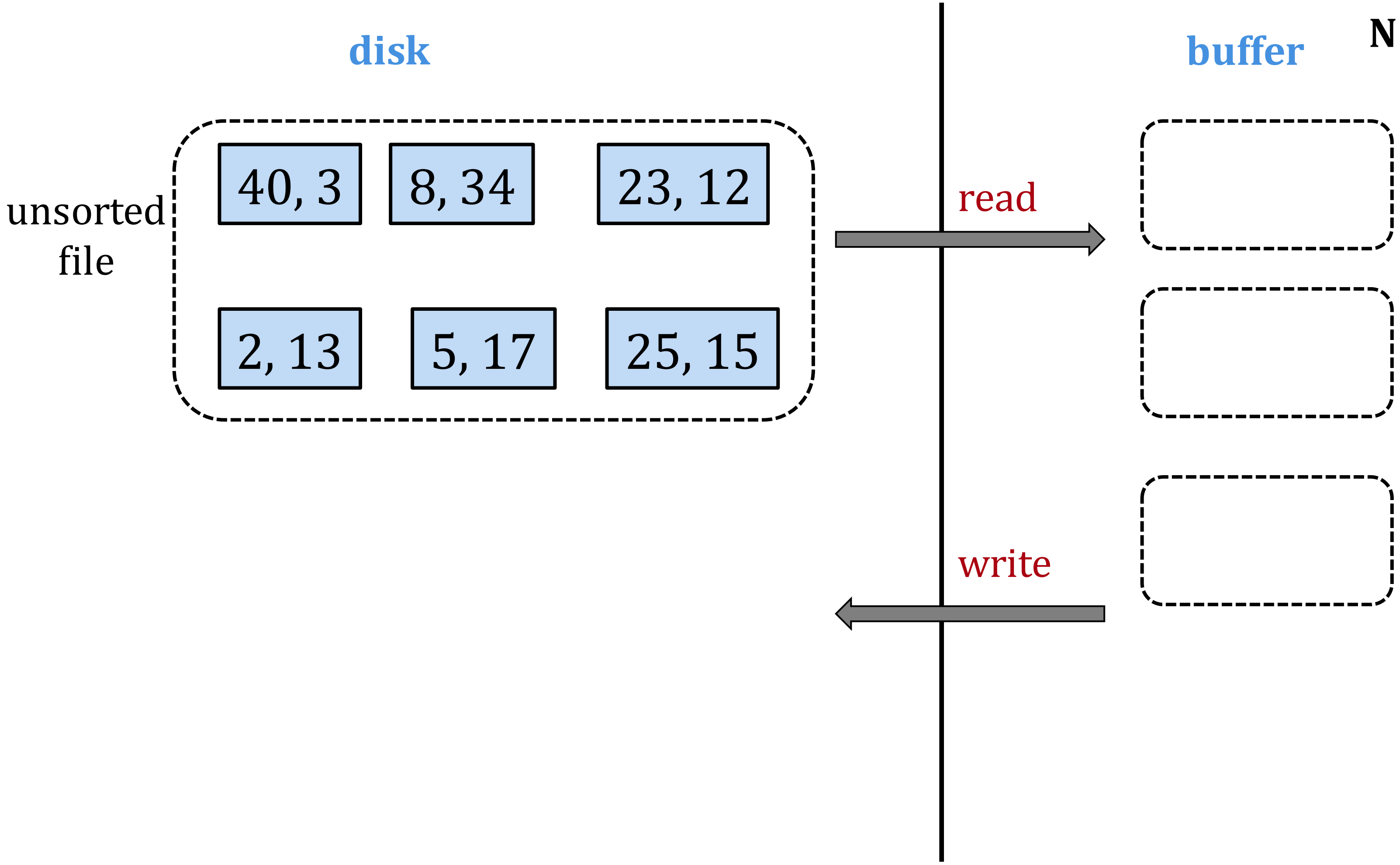
If there is backup, in the worst case the I/O cost could be $M_R \cdot M_S$

– This happens when there is a *single* join value

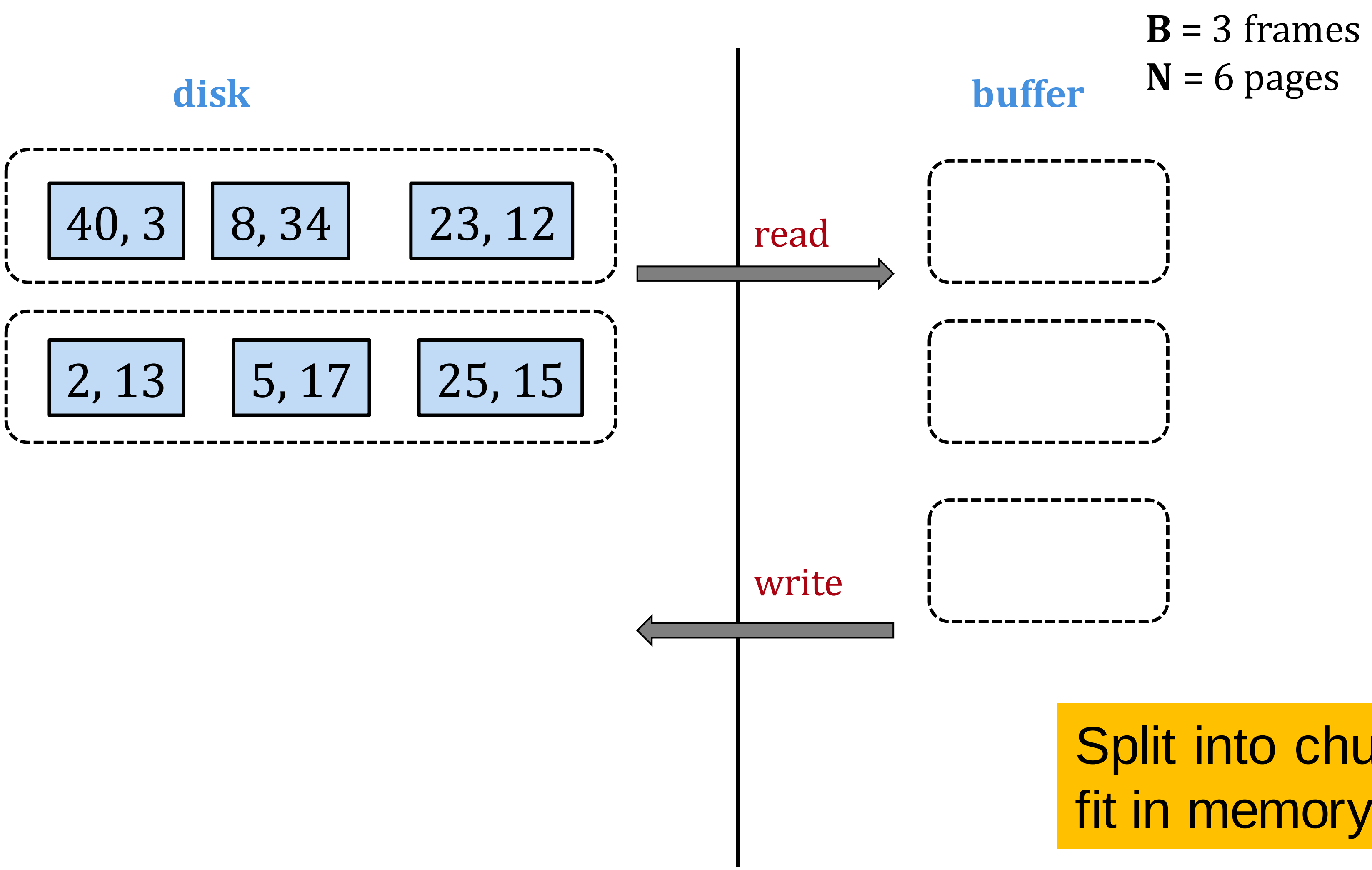
Total I/O cost ~ $sort(R) + sort(S) + M_R + M_S +$ writeJoinResults

2-Way Merge Sort

B = 3 frames
N = 6 pages



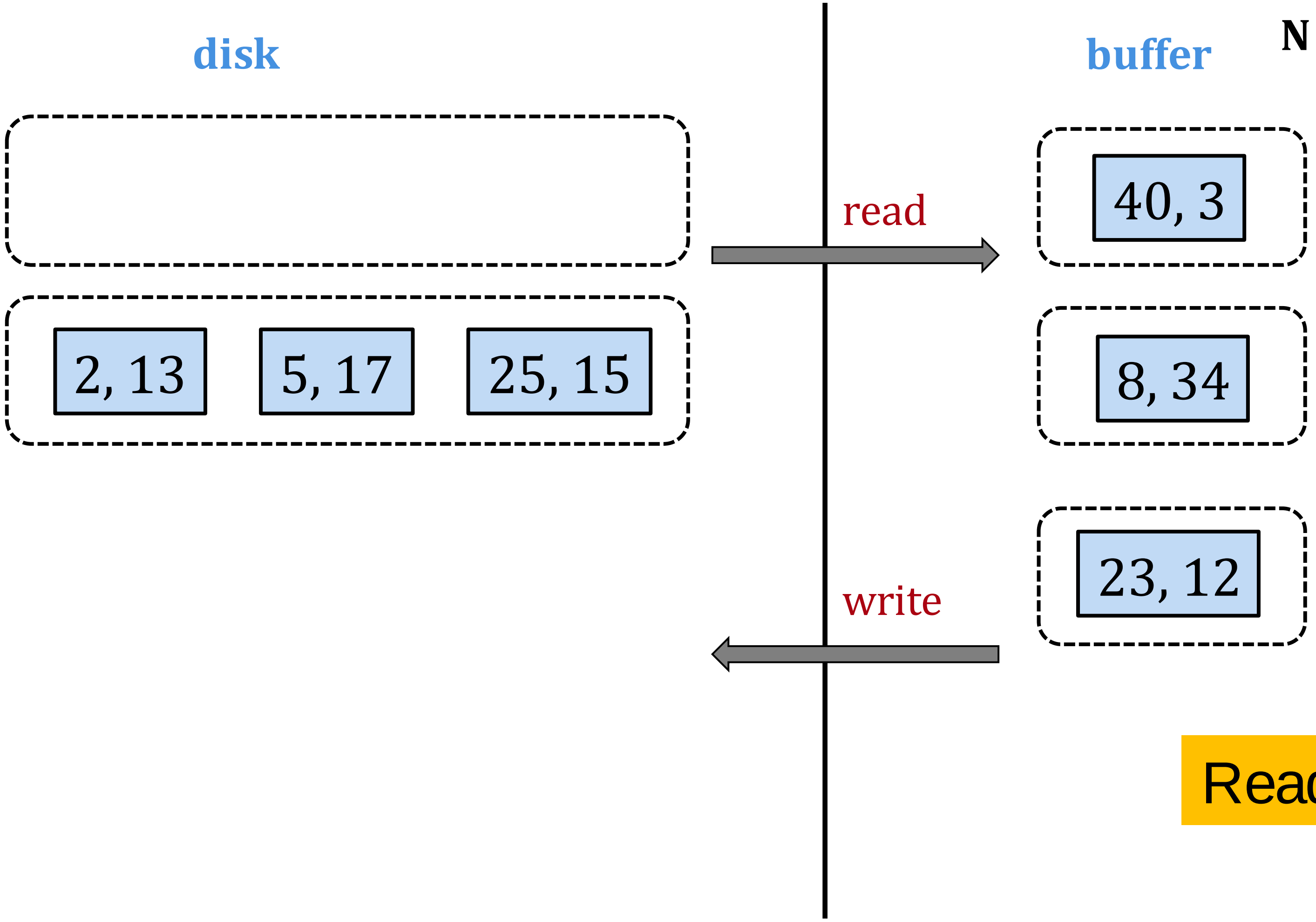
2-Way Merge Sort



Split into chunks that fit in memory

2-Way Merge Sort

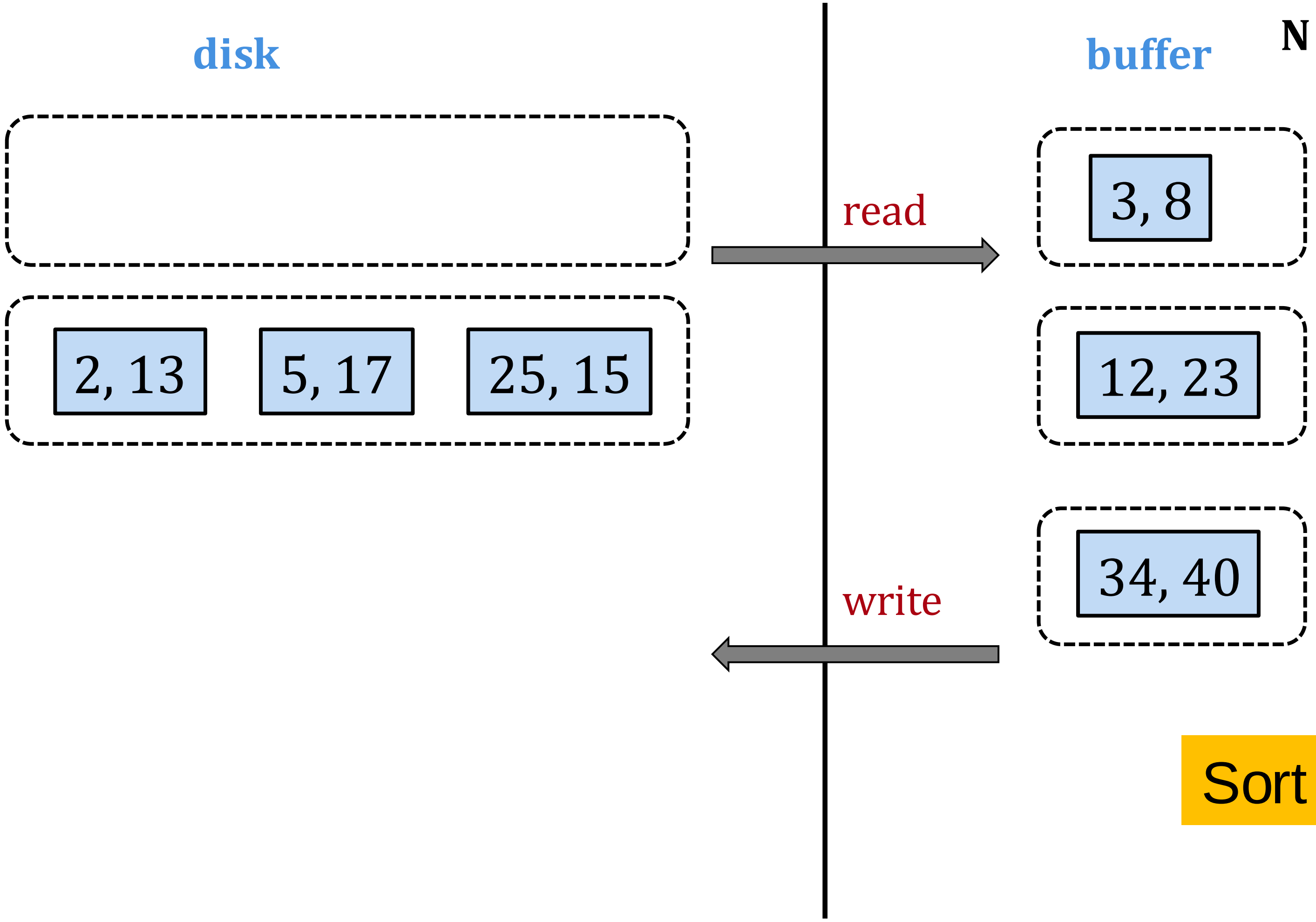
B = 3 frames
N = 6 pages



Read first chunk in memory

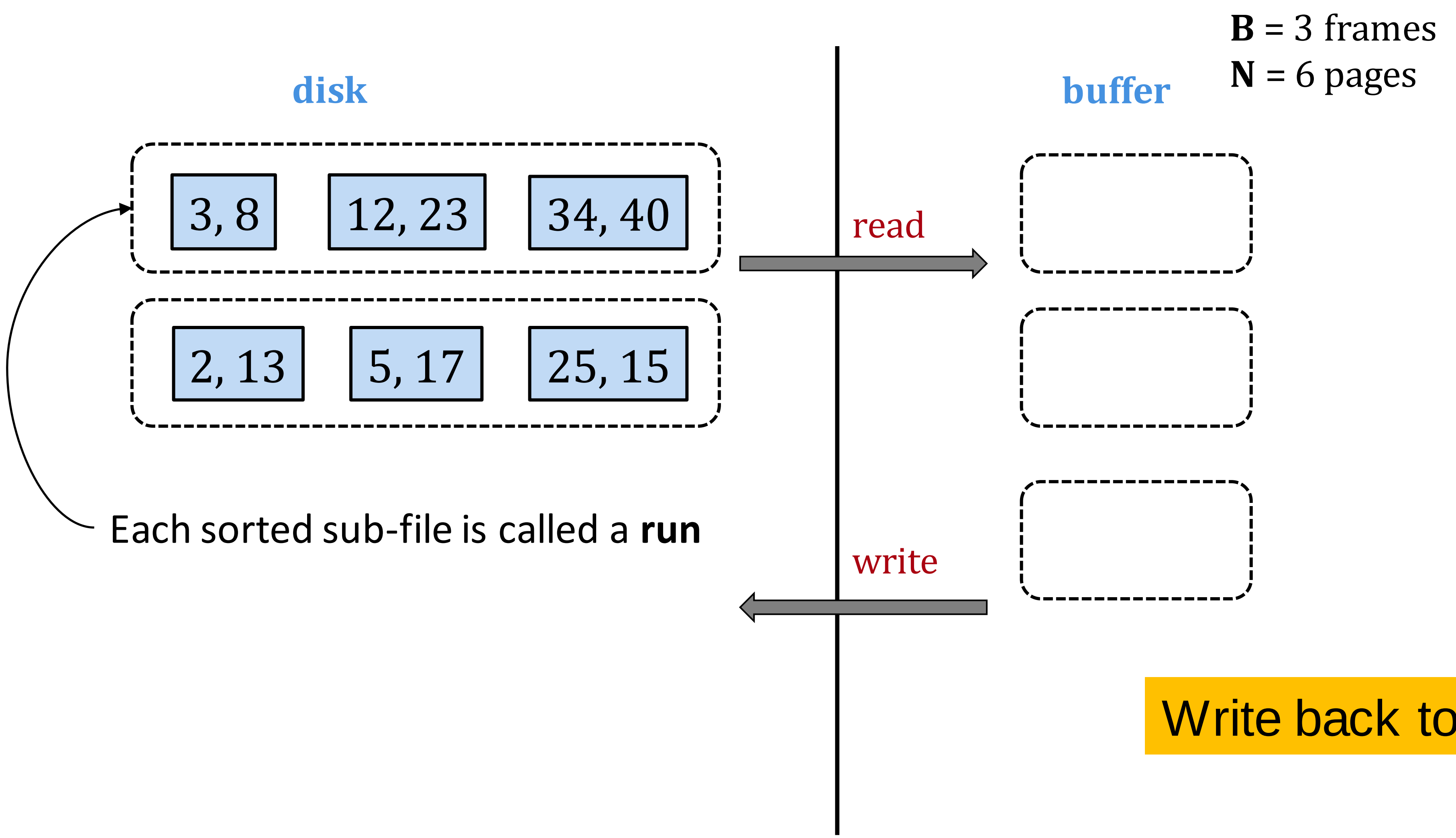
2-Way Merge Sort

B = 3 frames
N = 6 pages



Sort in memory

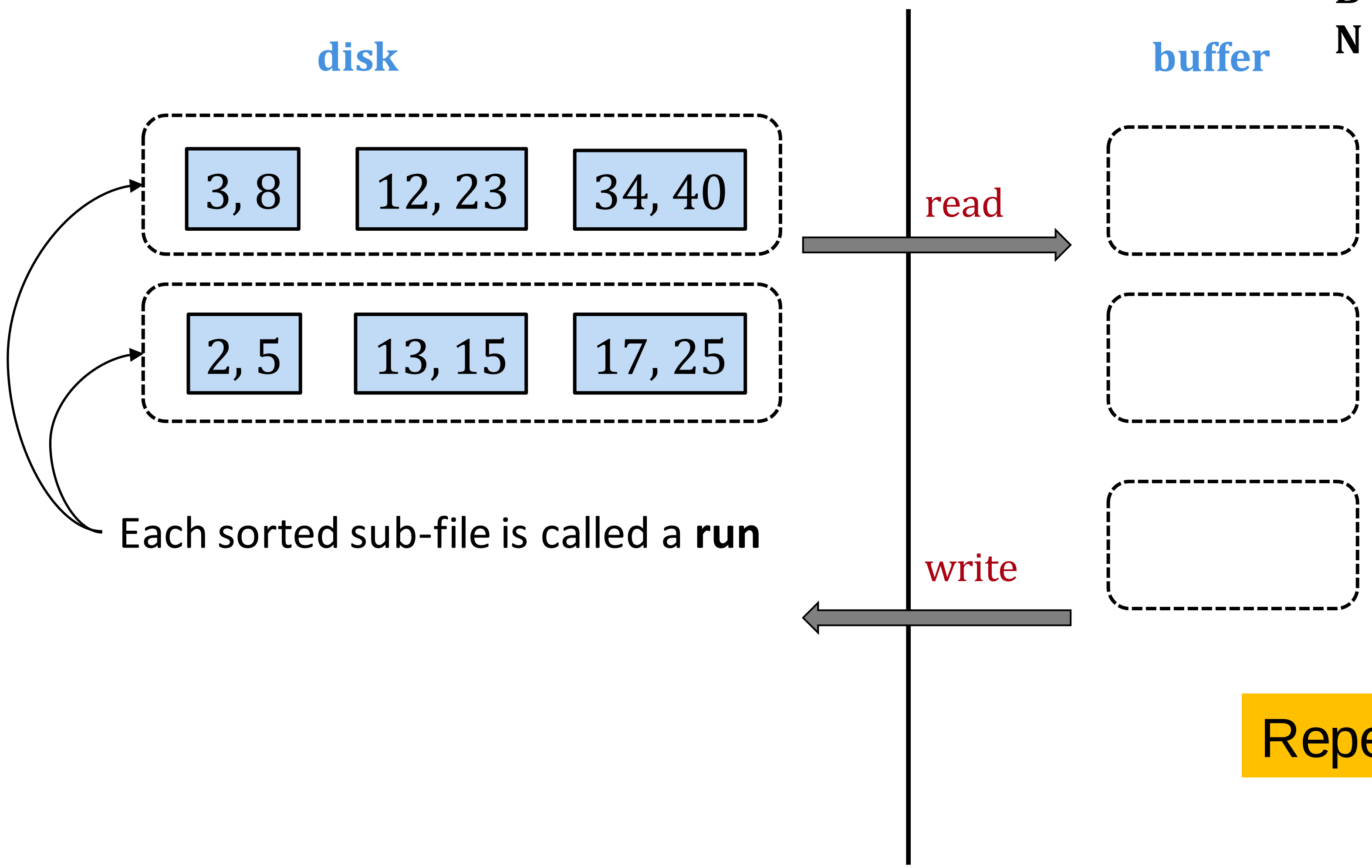
2-Way Merge Sort



Write back to disk

2-Way Merge Sort

B = 3 frames
N = 6 pages



Repeat for all chunks

Sort-Merge Join – Optimization

Generate sorted runs of for **R** and **S**

- I/O cost = $2 \times (M_R + M_S)$

Merge the sorted runs for **R** and **S**

- While merging check for the join condition and output the join tuples

I/O cost $\sim 3(M_R + M_S)$

Requires a buffer size B to be sufficiently large ($B^2 > \max(M_S, M_R)$)

Outline

Join

- Nested loop join
- Block nested loop join
- Index nested loop join
- Sort merge join
- **Hash join**

Aggregation

Hash Join

When the smaller relation (e.g., **S**) fits in memory

Build phase: Build a hash table on **S** (using hash function *h*)

Probing phase: For each tuple in R, probe the hash table for a match

Hash Join – Example

Join Key

		3
		8
		2
		4

Relation S

Join Key

1		
3		
4		
6		
8		
3		
5		
2		

Relation R

4, 8
2
3

Hash Table on S

Build phase: Build a hash table on **S** (using hash function *h*)

Hash Join – Example

Relation S

		Join Key
		3
		8
		2
		4

Relation R

Join Key		
1		
3		
4		
6		
8		
3		
5		
2		

Hash Table on S

4, 8
2
3

Build phase: Build a hash table on **S** (using hash function *h*)

Probing phase: For each tuple in R, probe the hash table for a match

Hash Join – Larger Than Memory

If neither relation fits in memory

Partition phase: Co-partition relations **R** and **S** on the join key

Build & Probing phase: For each pair of partitions, if the smaller partition fits in memory, build a hash table and probe

General Join Conditions

Equalities over multiple attributes

- e.g., *R.sid=S.sid* **and** *R.rname=S.sname*
- For Index Nested Loop Join
  - Requires matching index
- For sort-merge join or hash join, we can sort or hash using the combination of join attributes

Inequality conditions

- e.g., *R.rname* < *S.sname*
- Sort-merge join and hash join are not applicable
- Nested loop join can be always applied

Outline

Join

- Nested loop join
- Block nested loop join
- Index nested loop join
- Sort merge join
- Hash join

Aggregation

Aggregation

Sort on group by attributes (if any)

- Scan sorted tuples, computing running aggregate(max, min, sum, count)
- When the group by attribute changes, output aggregate result
- **cost** = sorting cost

Hash on group by attributes (if any)

- Hash entry = group attributes + running aggregate
- Scan tuples, probe hash table, update hash entry
- Scan hash table, and output each hash entry
- **cost** = scan relation + hashing

Summary

Join

- Nested loop join
- Block nested loop join
- Index nested loop join
- Sort merge join
- Hash join

Aggregation