Relational Operators: Join

CS 564 - Spring 2025

ACKs: *Jeff Naughton, Jignesh Patel, AnHai Doan*

JOIN OPERATOR

Algorithms for equijoin:

```
SELECT *
FROM   R, S
WHERE  R.a = S.a
```

Why can't we compute it as Cartesian product?

JOIN ALGORITHMS

Algorithms for equijoin:

- nested loop join
- **block nested loop join**
- index nested loop join
- block index nested loop join
- **sort merge join**
- **hash join**

NESTED LOOP JOIN

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

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

join the tuples on P_R with the tuples in P_S

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

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

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

NESTED LOOP JOIN

- Which relation should be the outer relation in the loop?
  - The smaller of the two relations
- How many buffer frames do we need?
  - 3 frames suffice!

BLOCK NESTED LOOP JOIN

Assume B buffer pages

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

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

BLOCK NESTED LOOP JOIN

What happens if **R** fits in memory?

- the I/O cost is only $M_R + M_S$!

To increase CPU efficiency, we can build an in-memory hash table for each block

- the key of the hash table is the join attribute
- the cost becomes $M_R + M_S \cdot \left\lceil \frac{f \cdot M_R}{B-2} \right\rceil$ where f is the fudge factor

fudge factor: the factor by which storing increases because of an underlying data structure

NLJ VS BNLJ

Example:

- M_R = 500 pages, M_S = 1000 pages
- B = 12

NLJ cost = $500 + 500 \cdot 1{,}000 = \mathbf{500,500}$

BNLJ cost = $500 + \frac{500 \cdot 1{,}000}{12-2} = \mathbf{50{,}500}$

The difference in I/O cost in an order of magnitude!

INDEX NESTED LOOP JOIN

S has an index on the join attribute

for each page P_R in **R**

    for each tuple r in P_R

        probe the index of **S** to retrieve matching tuples

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

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

BLOCK INDEX NESTED LOOP JOIN

for each block of *B-2* pages in **R**
    sort the tuples in the block
    for each tuple *r* in the block
        probe the index of **S** to retrieve matching tuples

- Why do we need to sort here?

SORT MERGE JOIN

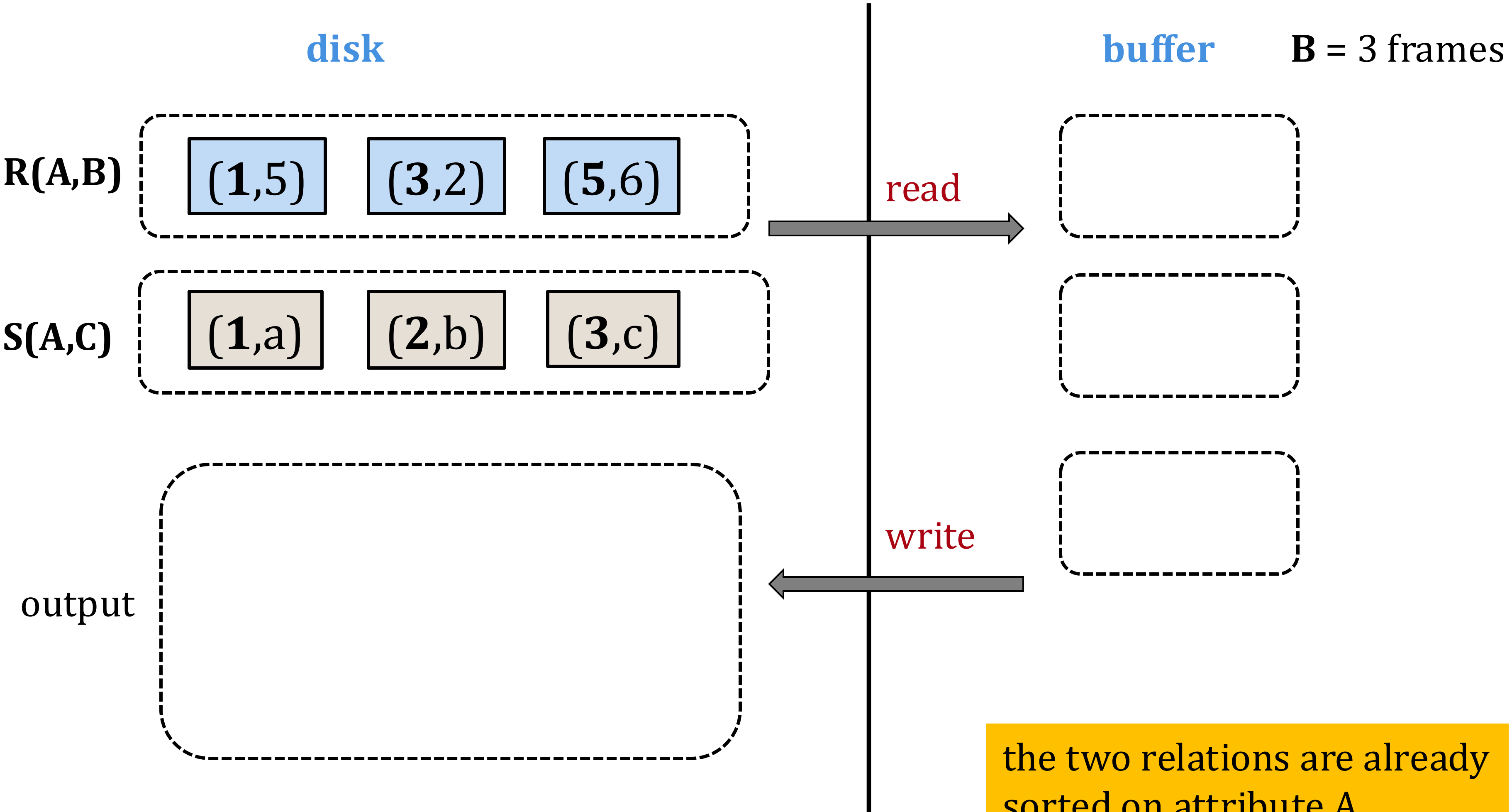
SORT MERGE JOIN: BASIC VERSION

The basic version:

- sort **R** and **S** on the join attribute using external merge sort
- read the sorted relations in the buffer and merge

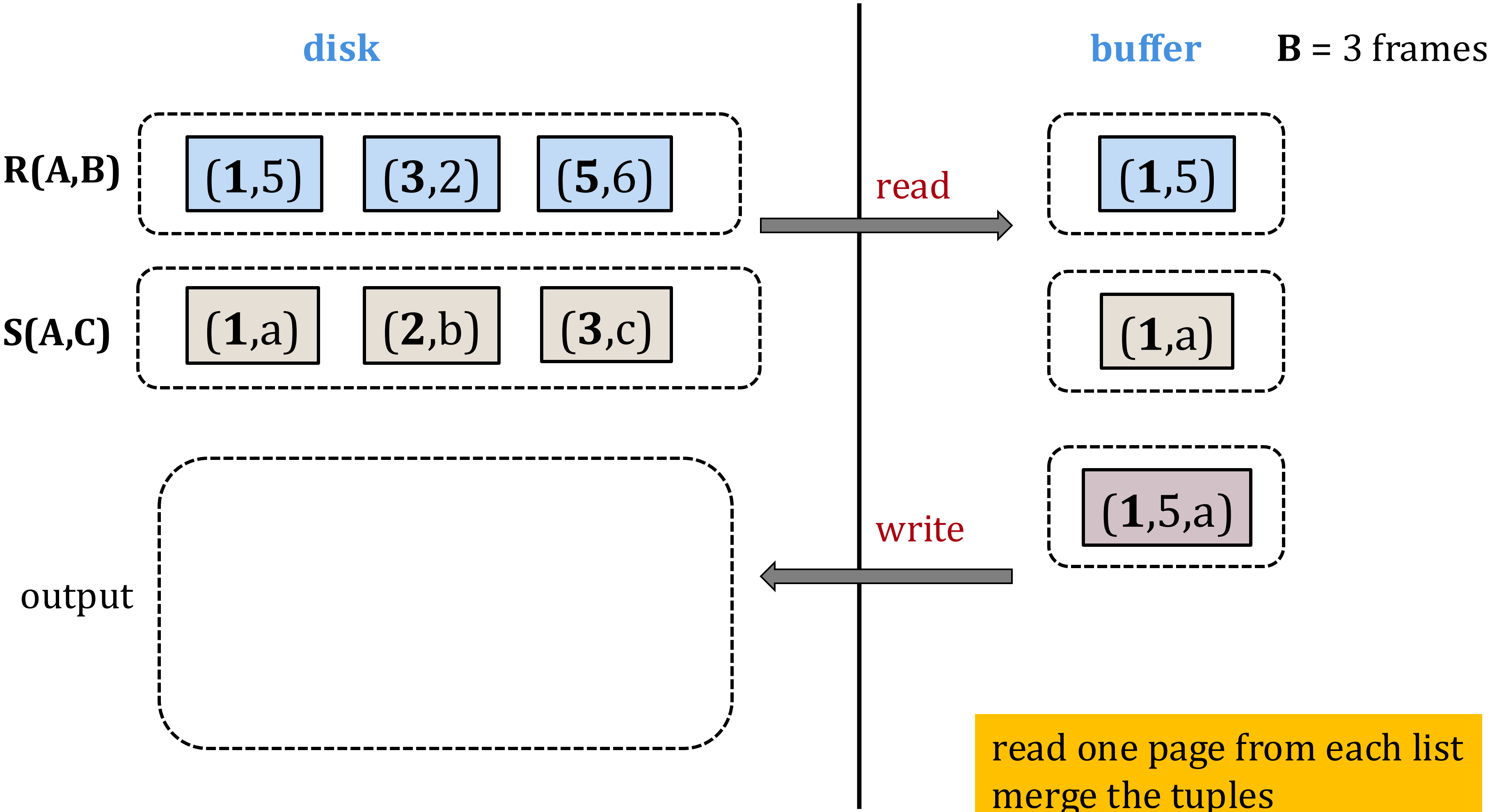
If **R**, **S** are already sorted on the join attribute we can skip the first step!

READ AND MERGE



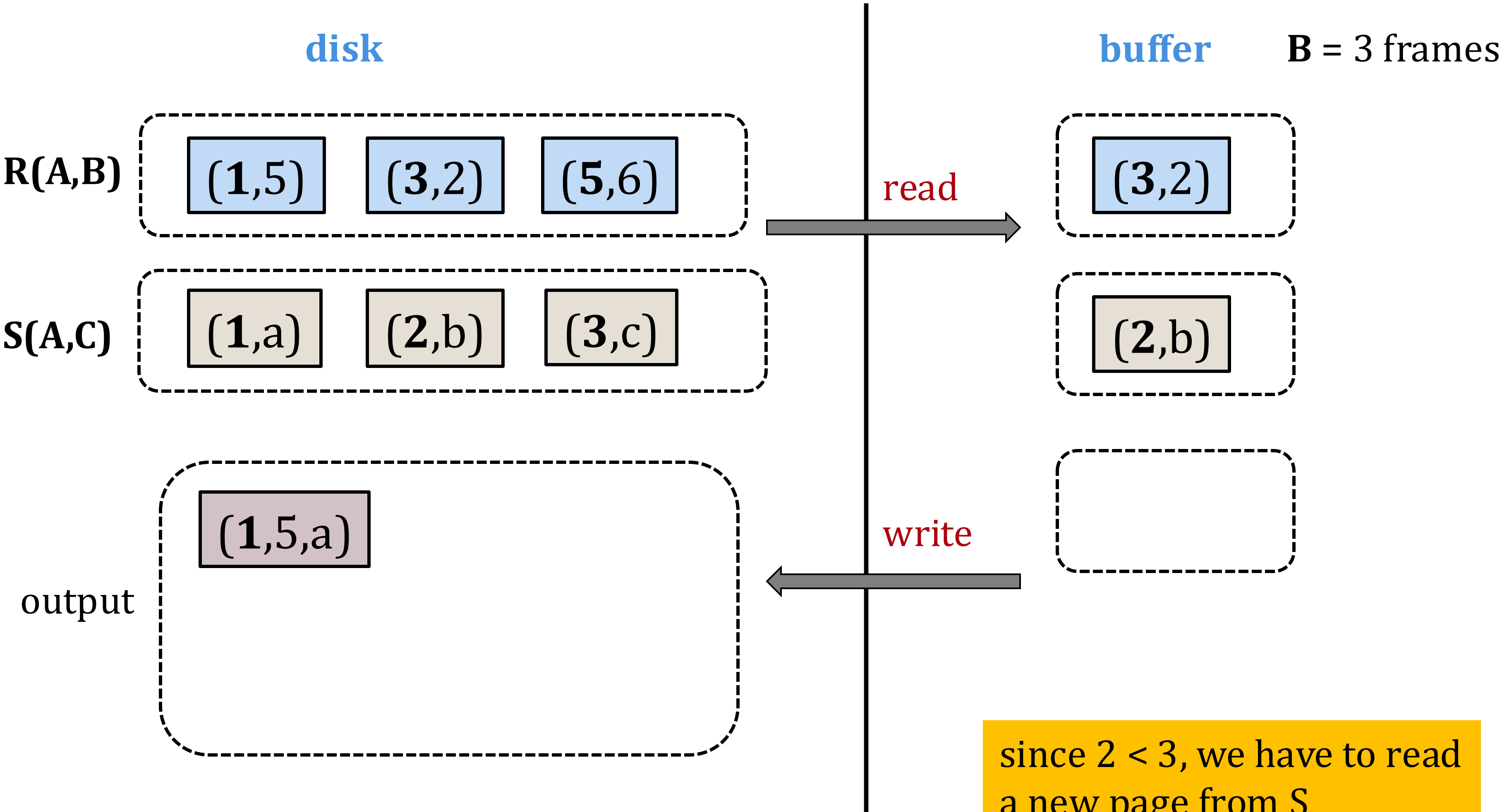
the two relations are already sorted on attribute A

READ AND MERGE



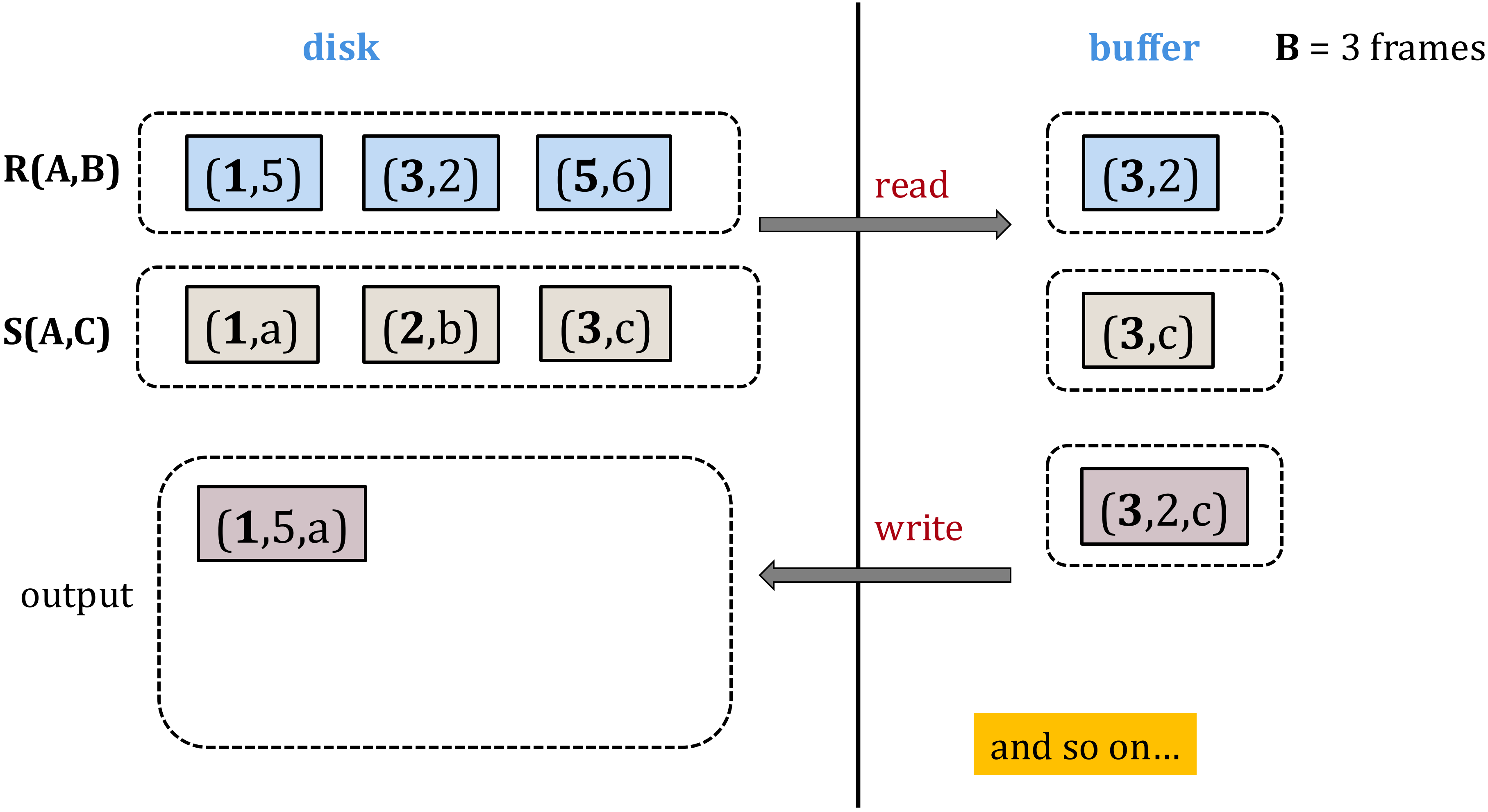
read one page from each list
merge the tuples

READ AND MERGE

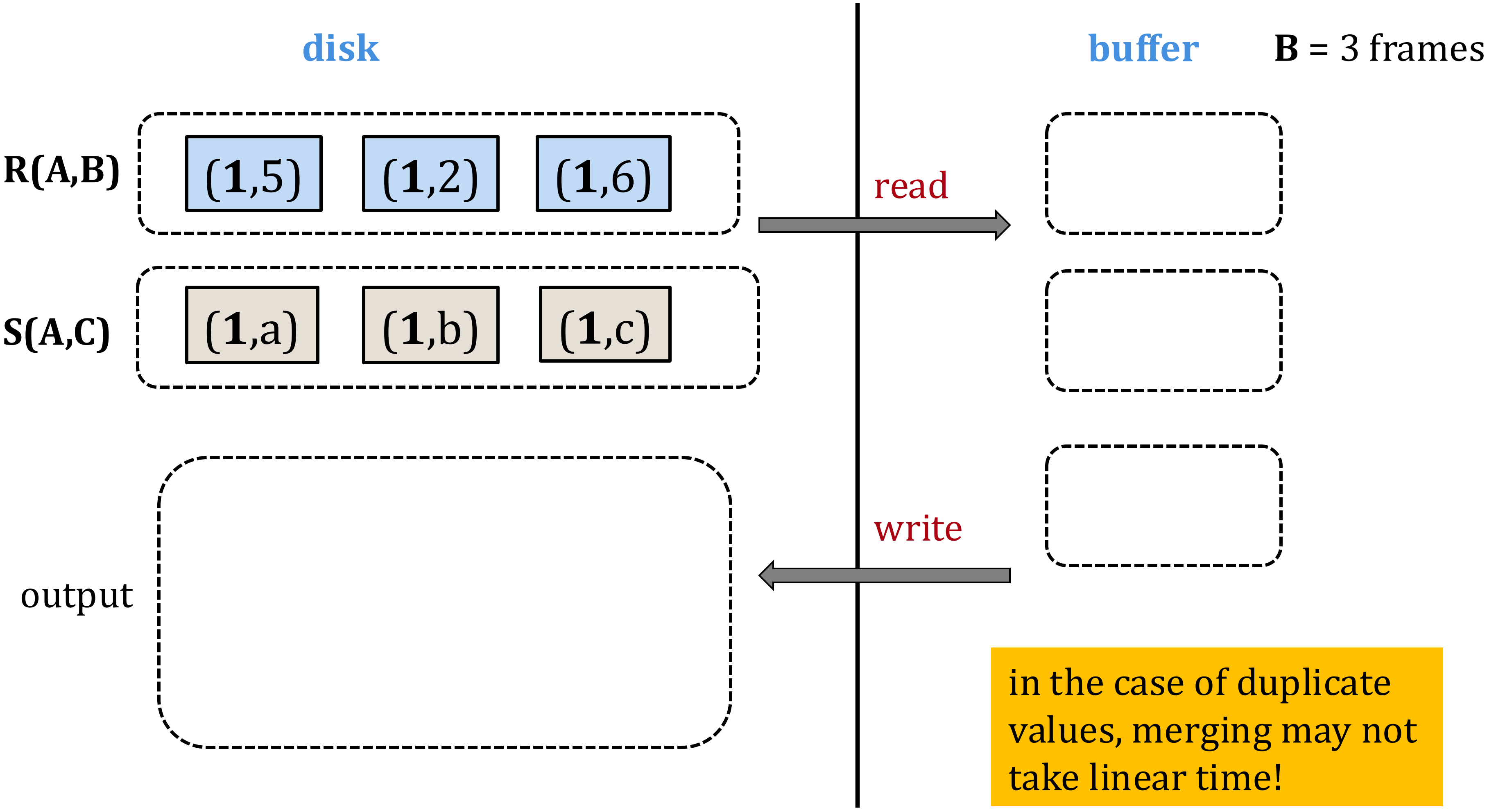


since 2 < 3, we have to read a new page from S

READ AND MERGE

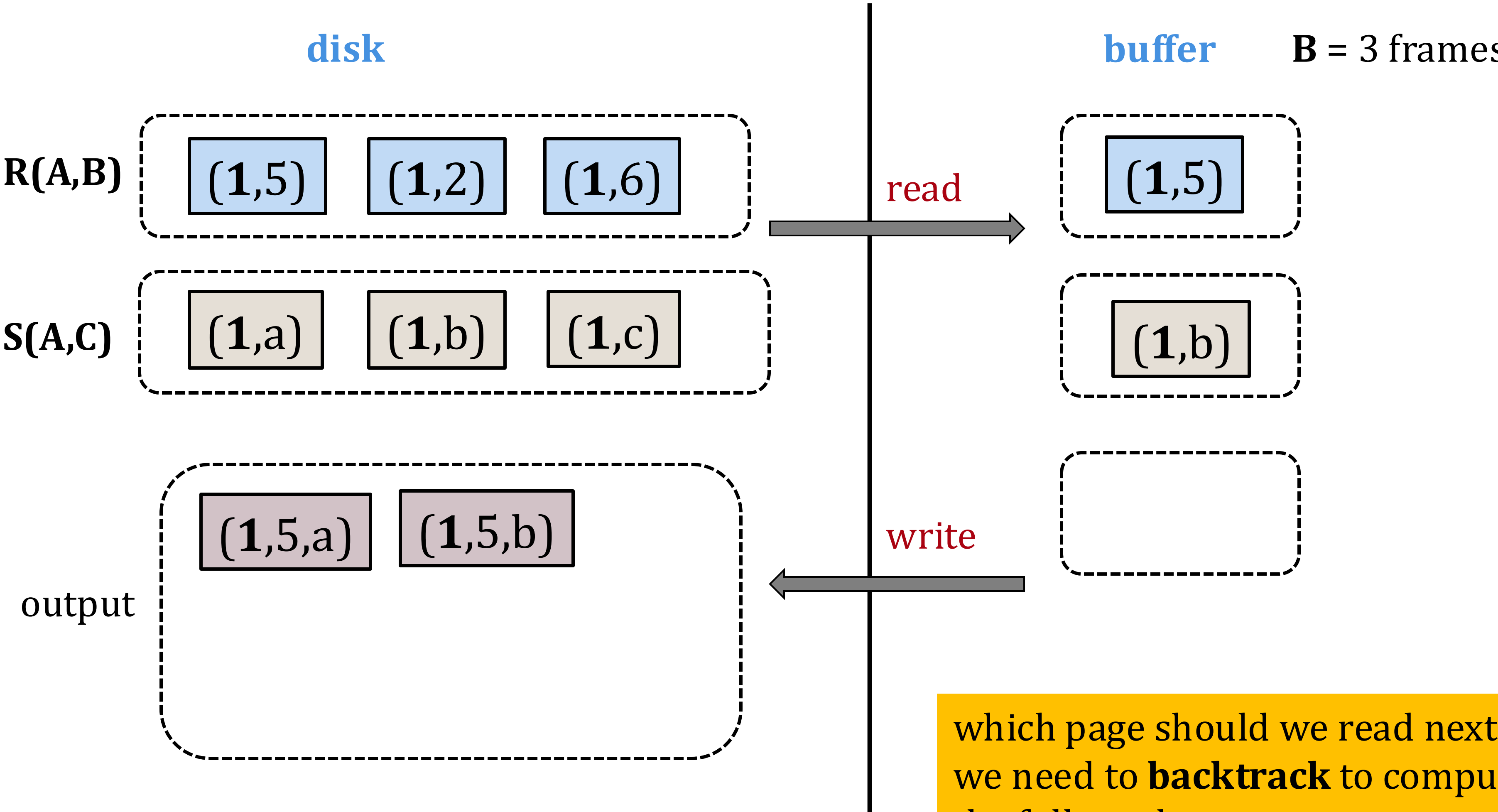


SORTING WITH DUPLICATES



in the case of duplicate values, merging may not take linear time!

SORTING WITH DUPLICATES



which page should we read next? we need to **backtrack** to compute the full result

SMJ: I/O COST

- If there is no backup, the I/O cost of read + merge is only $M_R + M_S$
- If there is backtracking, in the worst case the I/O cost could be $M_R \cdot (M_S - 1)$
  - this happens when there is a *single* join value

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

SORT MERGE JOIN: OPTIMIZED

- Generate sorted runs of size $\sim 2B$ for **R** and **S**
- Merge the sorted runs for **R** and **S**
  - while merging check for the join condition and output the join tuples

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

But how much memory do we need for this to happen?

SMJ: MEMORY ANALYSIS

- In the first phase, we create runs of length ~2B
- Hence, the number of runs is $\frac{M_R+M_S}{2B}$
- To perform a *k*-way merge, we need *k+1* buffer pages, so:

$$\frac{M_R+M_S}{2B} \leq B - 1 \text{ or } B(B-1) \geq (M_R + M_S)/2$$

If $B(B-1) \geq (M_R + M_S)/2$, then SMJ has I/O cost $\sim 3(M_R + M_S)$

HASH JOIN

HASH FUNCTION REFRESHER

- We will use a hash function *h* to map values of the join attribute (A) into buckets [1, B-1]
- A tuple *t* is then hashed to bucket *h(t.A)*

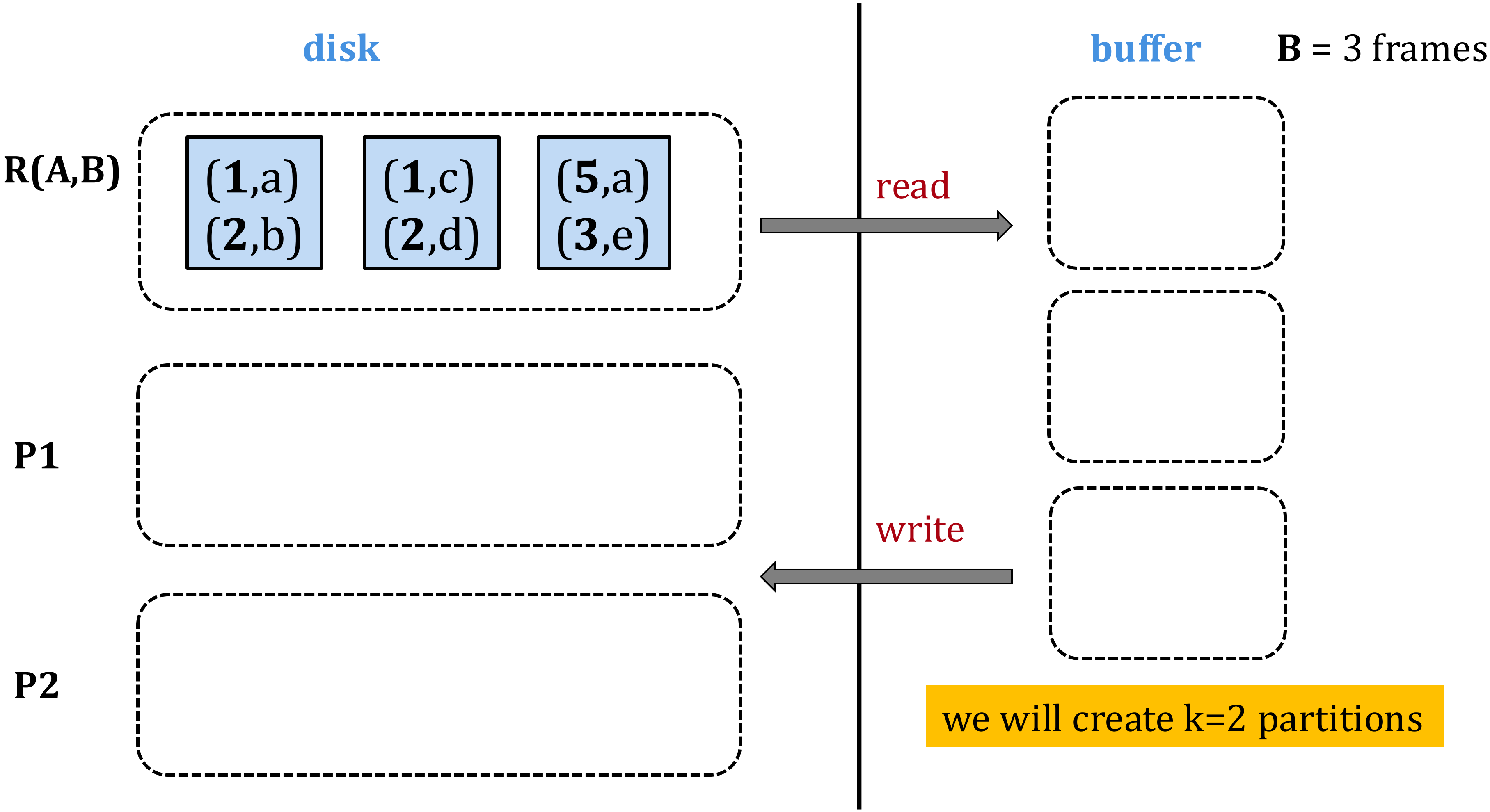
- A hash **collision** occurs when x != y but h(x) = h(y)
- It can never happen that x = y and h(x) != h(y)

HASH JOIN: OVERVIEW

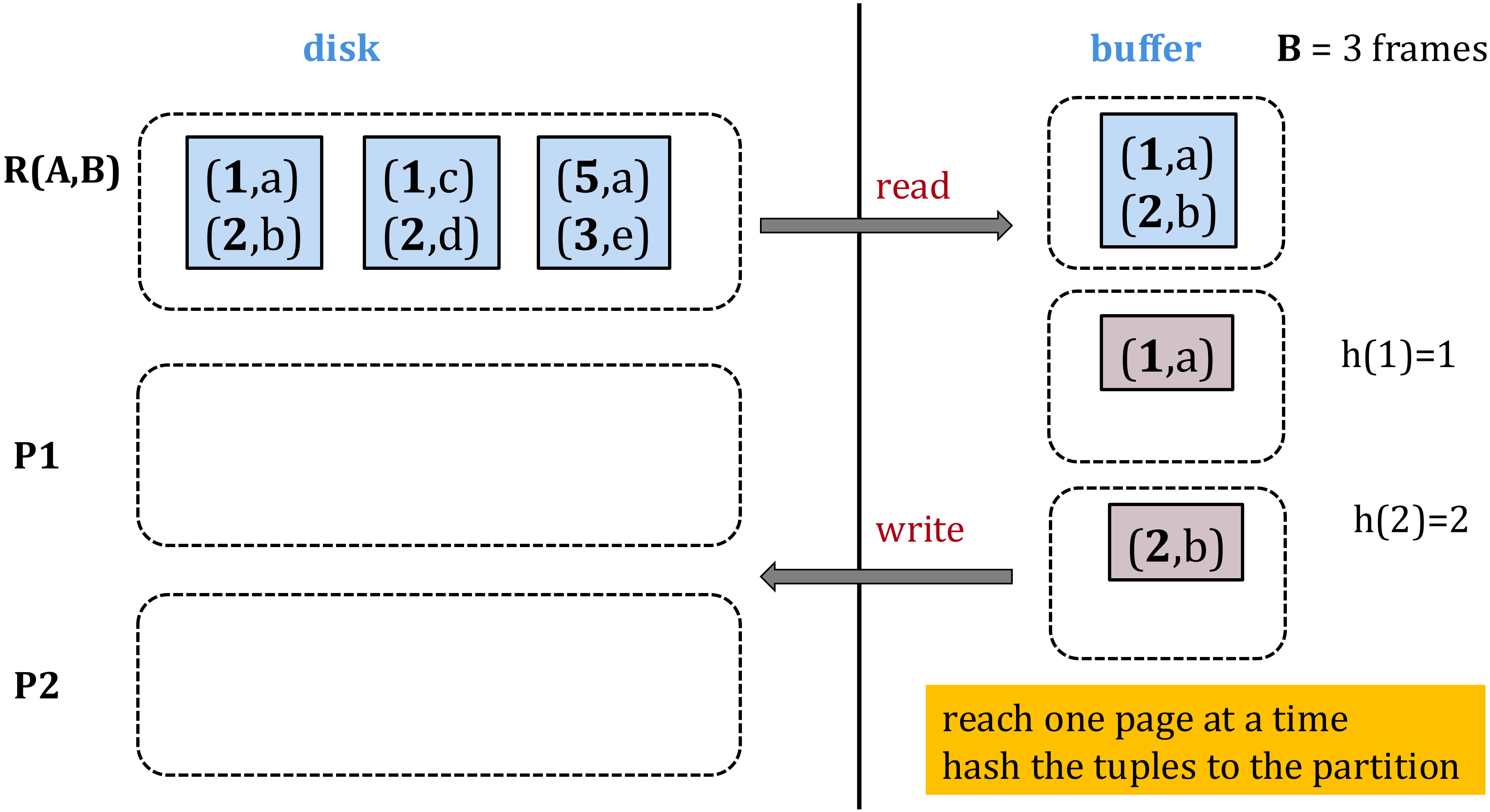
Start with a hash function h on the join attribute

- **Partition phase:** partition **R** and **S** into k partitions using h
- **Matching phase:** join each partition of **R** with the corresponding (same hash value) partition of **S** using BNLJ

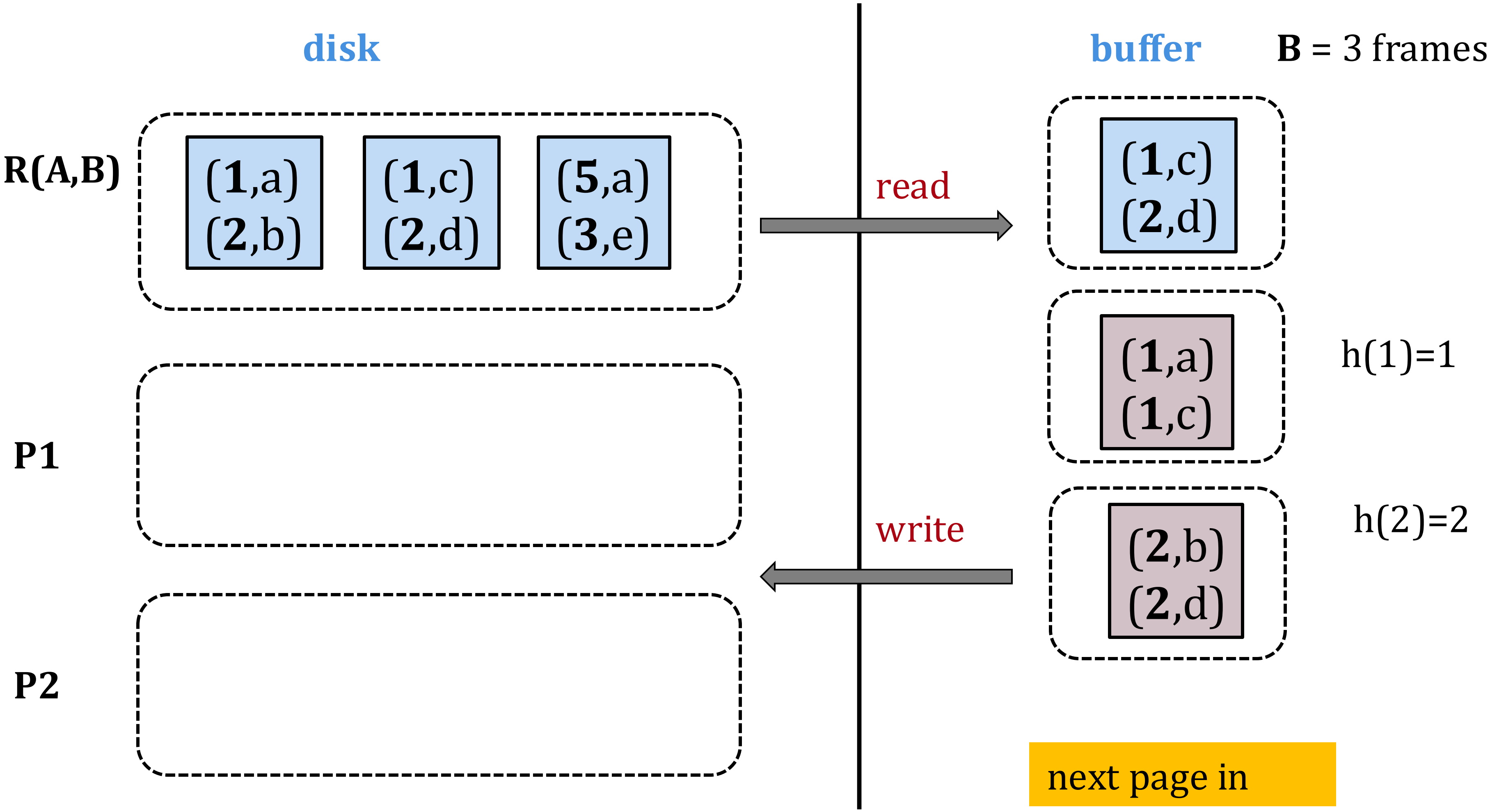
PARTITION PHASE



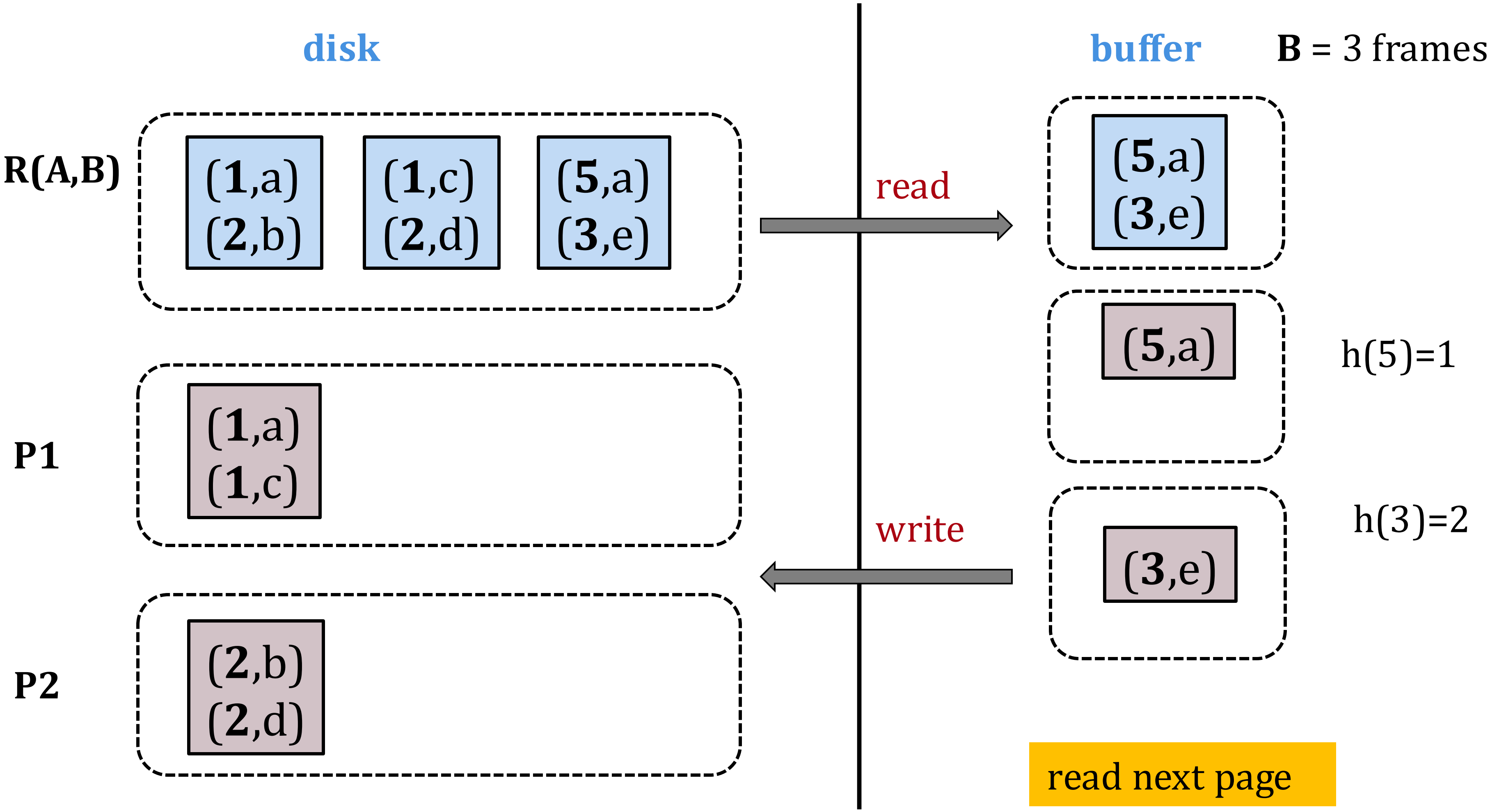
PARTITION PHASE



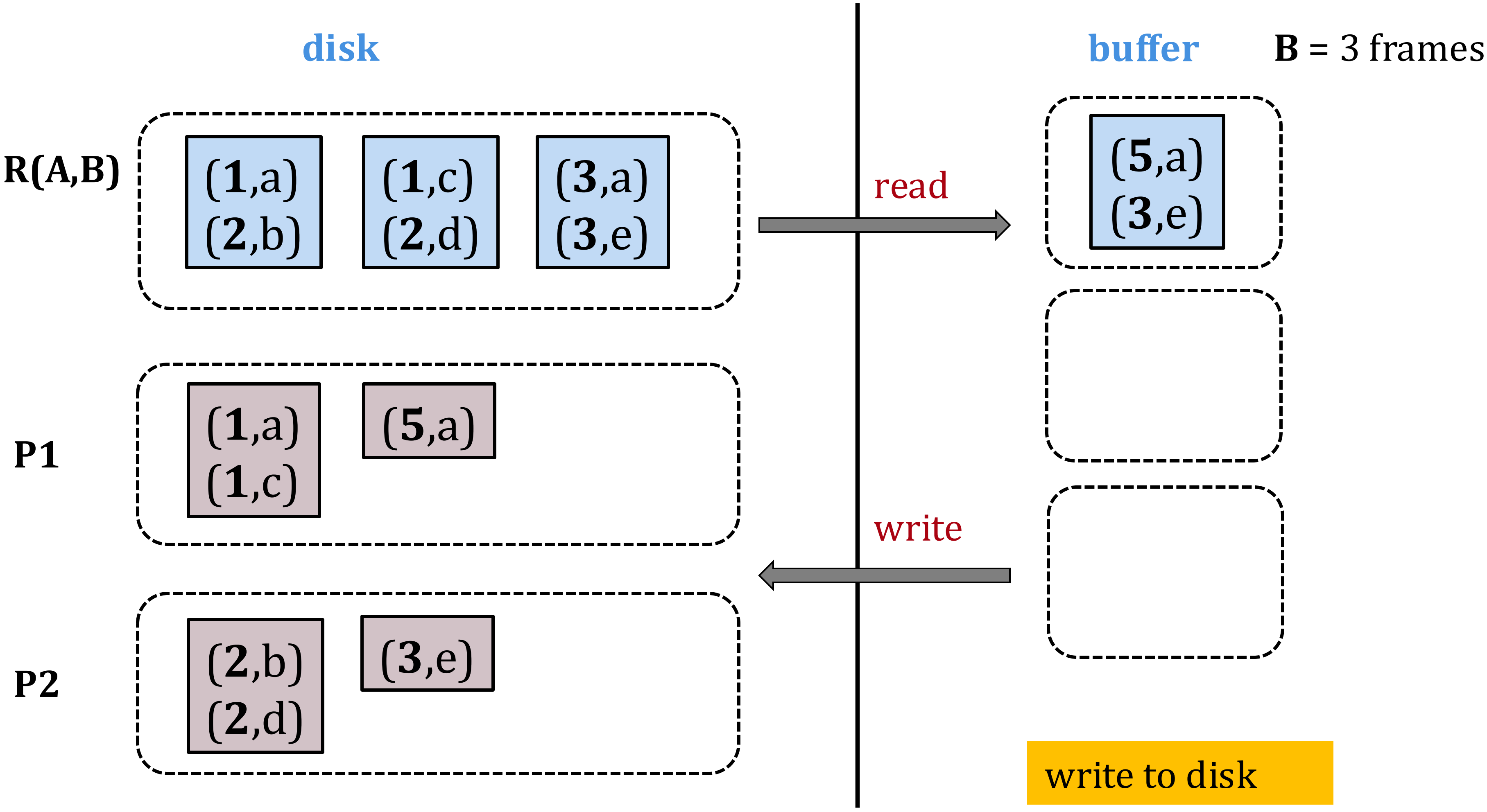
PARTITION PHASE



PARTITION PHASE



PARTITION PHASE



BUCKET SIZE

- We can create up to k = B-1 partitions in one pass
- How big are the buckets we create?
  - Ideally, each bucket has ~ M/(B-1) pages
  - but hash collisions can occur!
  - or we may have many duplicate values on the join attribute (skew)

- In the matching phase, we join two buckets from R, S with the same hash value
  - We want to do this in linear time using BNLJ, so we must guarantee that each bucket from one of the two relations is at most B-1 pages

HJ: I/O COST

- Suppose $M_R \le M_S$
- The partition phase gives buckets of size $\sim \mathrm{M_R/B}$
- To make BNLJ run in 2 passes we need to make sure that:

$$\frac{M_R}{B-1} \le B - 2 \text{ or } (\mathrm{B}-2)(B-1) \ge M_R$$

If $(\mathrm{B}-2)(B-1) \ge M_R$, then HJ has I/O cost $\sim 3(M_R + M_S)$

- If $M_R \le B - 2$, then HJ needs only one pass!

HJ/BNLJ COMPARISON

Suppose $M_R \leq M_S$

$$\text{BNLJ cost} = M_R + M_S \cdot \left\lceil \frac{M_R}{B-2} \right\rceil$$

- If $M_R \leq B - 2$, HJ and BNLJ both run in 1 pass and have the same cost
- If $B - 2 < M_R \leq 3(B - 2)$, BNLJ is better!
- For other values, it depends on M_R, M_S

SMJ/BNLJ COMPARISON

Suppose $M_R \leq M_S$

- To do a 2-pass, SMJ needs $B(B-1) \geq (M_R + M_S)/2$
  - the I/O cost is: $3(M_R + M_S)$

- To do a 2-pass, HJ needs $(B-2)(B-1) \geq M_R$
  - the I/O cost is: $3(M_R + M_S)$

GENERAL JOIN CONDITIONS

Equalities over multiple attributes

- e.g., *R.sid=S.sid* **and** *R.rname=S.sname*
- for Index Nested Loop Join
  - index on <*sid, sname*>
  - index on *sid* or *sname*
- for SMJ and HJ, we can sort or hash using the combination of join attributes

GENERAL JOIN CONDITIONS

Inequality conditions

- e.g., *R.rname* < *S.sname*
- For BINL, we need (clustered) B+ tree index
- SMJ and HJ are not applicable
- BNLJ can be always applied