Relational Operators: Other

CS 564- Fall 2021

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

WHAT IS THIS LECTURE ABOUT?

We will discuss algorithms for

- Projection
- Set operations
- Aggregation

PROJECTION

PROJECT OPERATOR

Simple case: `SELECT R.A, R.D`

- scan the file and for each tuple output R.A, R.D

Hard case: `SELECT DISTINCT R.A, R.D`

- project out the attributes
- eliminate *duplicate tuples* (this is the difficult part!)

PROJECT: SORT-BASED

Naïve algorithm:

1. scan the relation and project out the attributes
2. sort the resulting set of tuples using all remaining attributes
3. scan the sorted set by comparing only adjacent tuples and discard duplicates

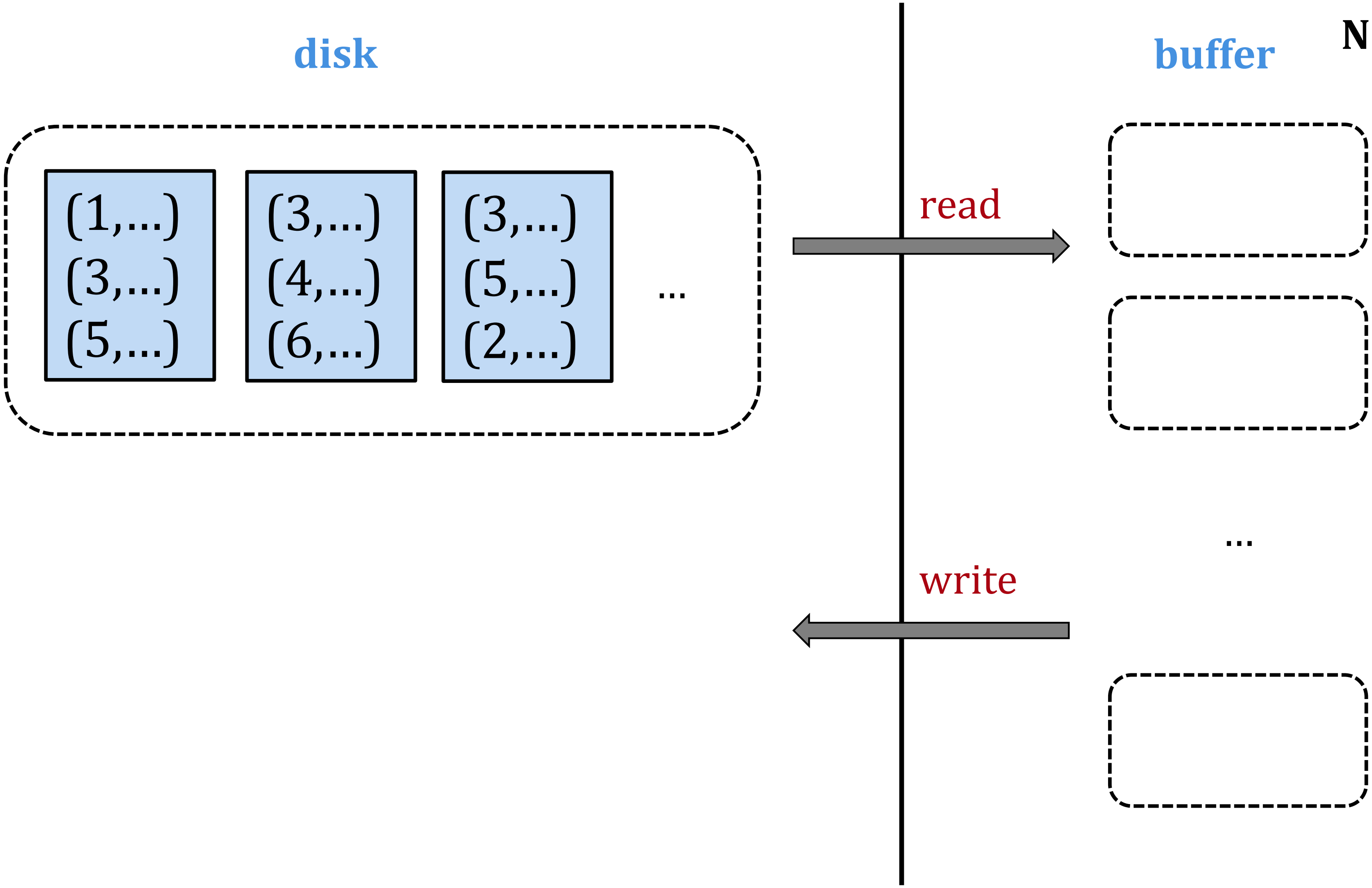
RUNNING EXAMPLE

`R(A,B,C,D,E)`

- $N = 1000$ pages
- $B = 20$ buffer pages
- Each field in the tuple has the same size
- Suppose we want to project on attribute `A`

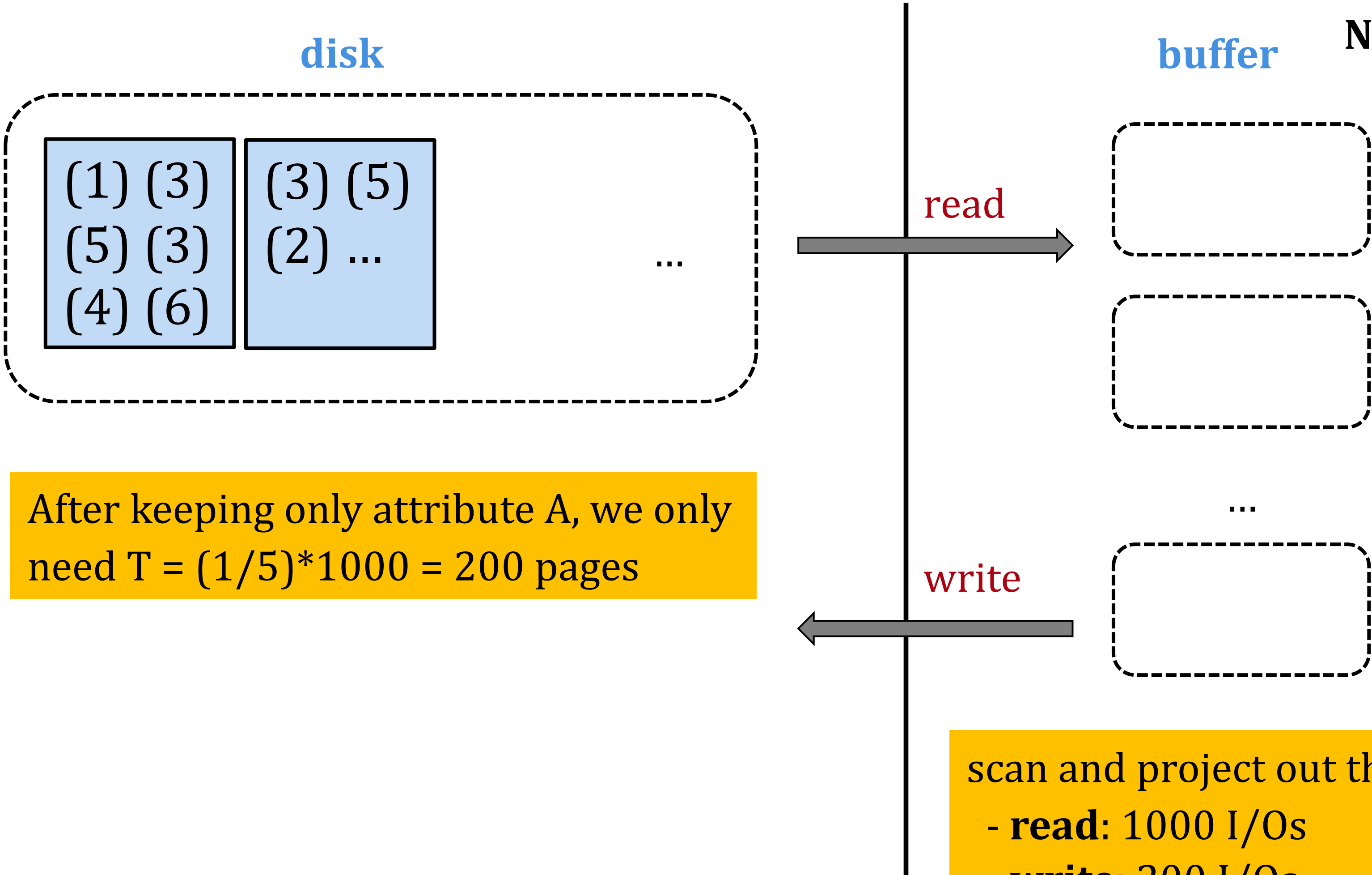
RUNNING EXAMPLE

B = 20 frames
N = 1000 pages



RUNNING EXAMPLE

B = 20 frames
N = 1000 pages

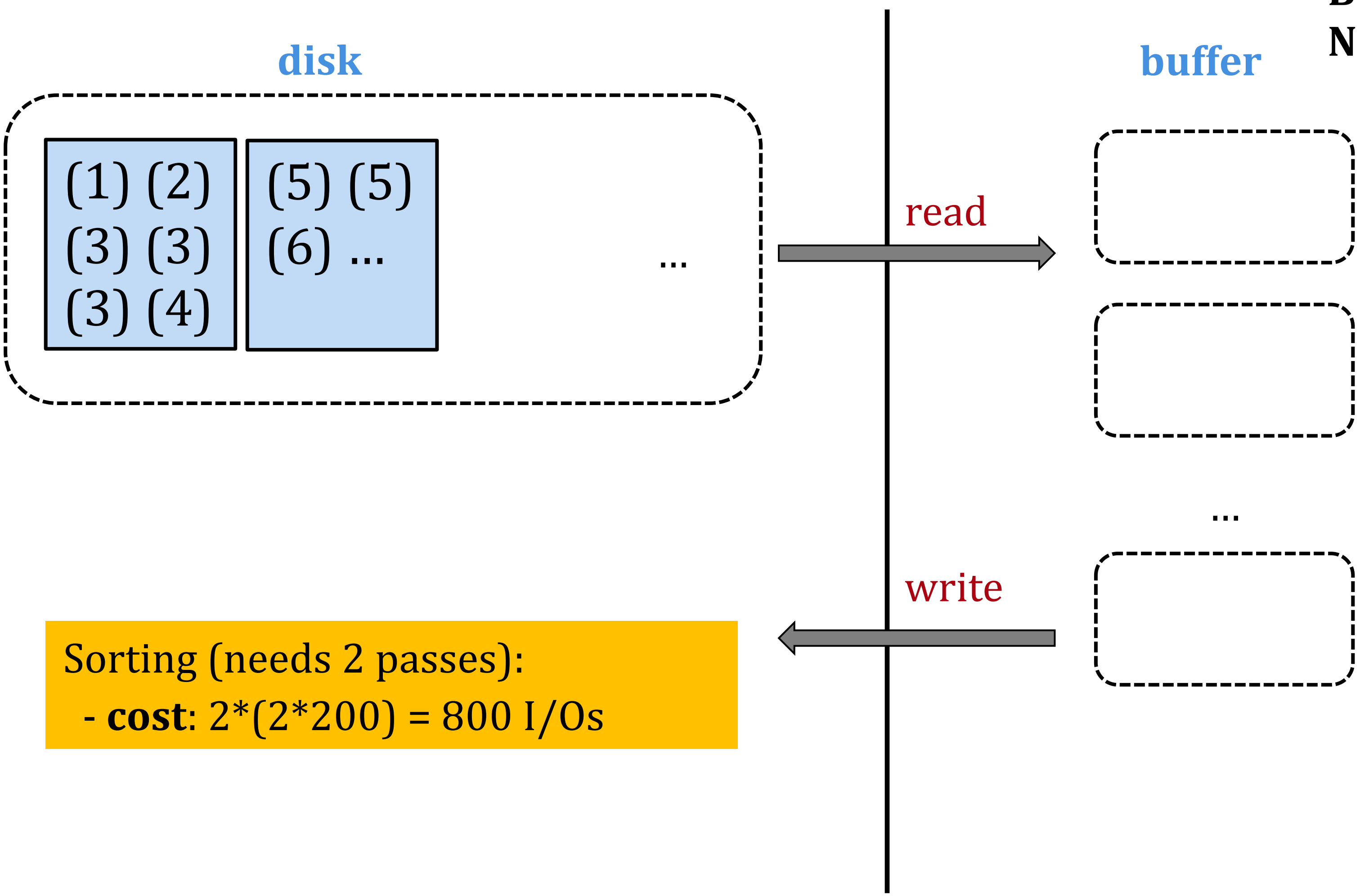


After keeping only attribute A, we only need T = (1/5)*1000 = 200 pages

scan and project out the attributes:
- **read**: 1000 I/Os
- **write**: 200 I/Os

RUNNING EXAMPLE

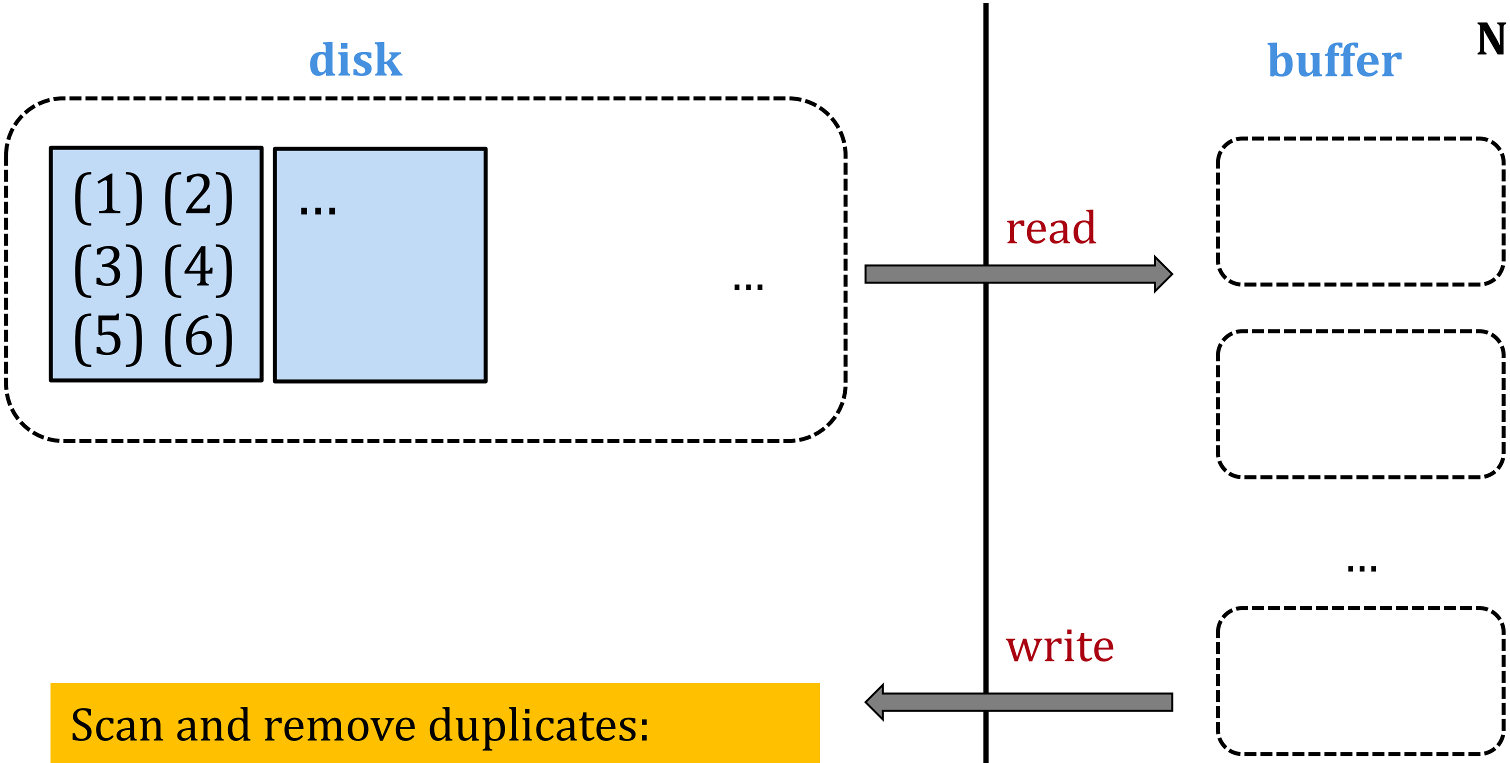
B = 20 frames
N = 1000 pages



Sorting (needs 2 passes):
- **cost**: 2*(2*200) = 800 I/Os

RUNNING EXAMPLE

B = 20 frames
N = 1000 pages



Scan and remove duplicates:
- **cost**: 200 I/Os

**Recall that we ignore the cost of writing the final output to disk*

Total cost = 2200 I/Os

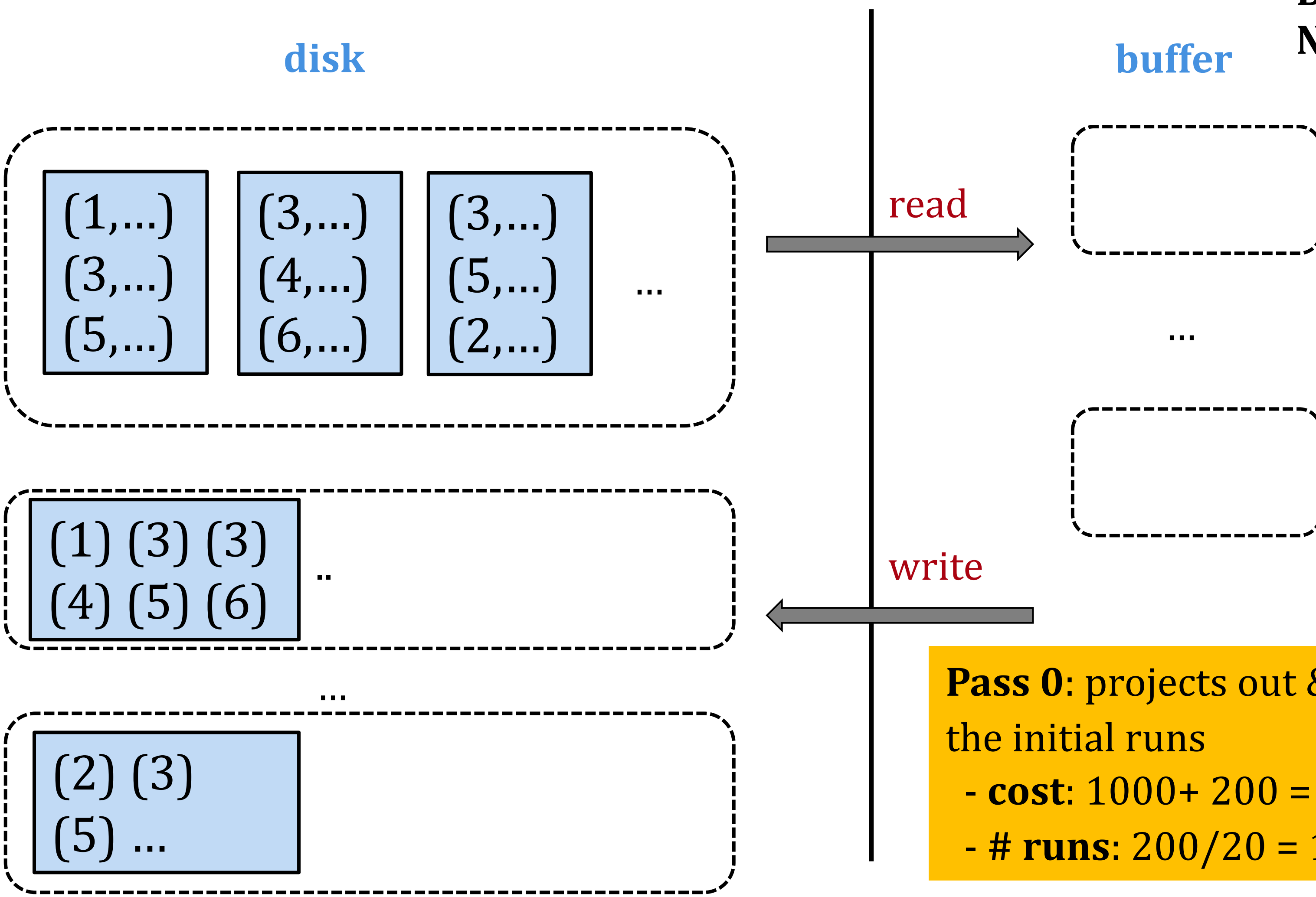
PROJECT: SORT-BASED

We can improve upon the naïve algorithm by modifying the sorting algorithm:

1. In Pass **0** of sorting, project out the attributes
2. In subsequent passes, eliminate the duplicates while merging the runs

RUNNING EXAMPLE

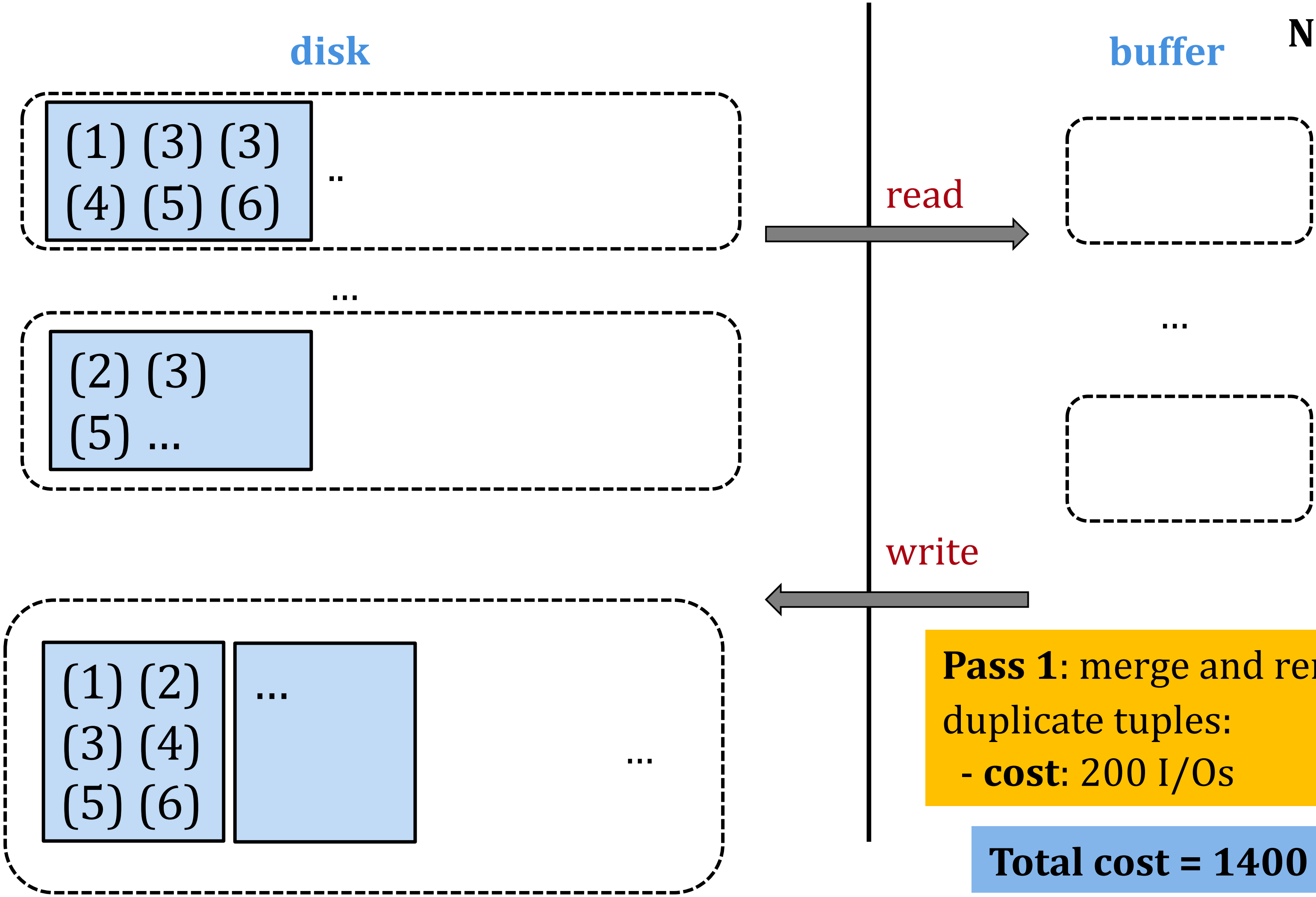
B = 20 frames
N = 1000 pages



Pass 0: projects out & creates the initial runs
- **cost**: 1000+ 200 = 1200 I/Os
- **# runs**: 200/20 = 10

RUNNING EXAMPLE

B = 20 frames
N = 1000 pages



Pass 1: merge and remove duplicate tuples:
- **cost**: 200 I/Os

Total cost = 1400 I/Os

PROJECT: HASH-BASED

2-phase algorithm:

- **partitioning**
  - project out attributes and split the input into B-1 partitions using a hash function *h*
- **duplicate elimination**
  - read each partition into memory and use an in-memory hash table (with a *different* hash function) to remove duplicates

PROJECT: HASH-BASED

When does the hash table fit in memory?

- size of a partition = $T\ /\ (B-1)$, where T is #pages after projection
- size of hash table = $f \cdot T\ /\ (B-1)$, where f is the fudge factor (typically ~ 1.2)
- So, it must be $B\ >\ f \cdot T\ /\ (B-1)$

HASH-BASED COST ANALYSIS

- T = 200 so the hash table fits in memory!
- partitioning cost = 1000 + 200 = 1200 I/Os
- duplicate elimination cost = 200 I/Os

total cost = 1400 I/Os

COMPARISON

Benefits of sort-based approach

- better handling of skew
- the output result is sorted

For large values of the buffer size B, both algorithms need only 2 passes!

PROJECT: INDEX-BASED

- Index-only scan
  - projection attributes are a subset of index attributes
  - apply projection algorithm only to index entries!
- If an *ordered index* contains all projection attributes as prefix of the search key:
  1. retrieve index data entries in order
  2. discard unwanted fields
  3. compare adjacent entries to eliminate duplicates

SET OPERATIONS

SET OPERATIONS

- Intersection is a special case of a join, so any algorithm for equijoins applies
- Union and difference are similar

UNION

- Sort-based:
  - sort both relations (on *all attributes*)
  - merge sorted relations while eliminating duplicates
- Hash-based:
  - hash-partition R and S
  - build an in-memory hash table for each partition R_i
  - probe with tuples in S_i, add to the output result if not a duplicate

SET DIFFERENCE

We want to compute R - S

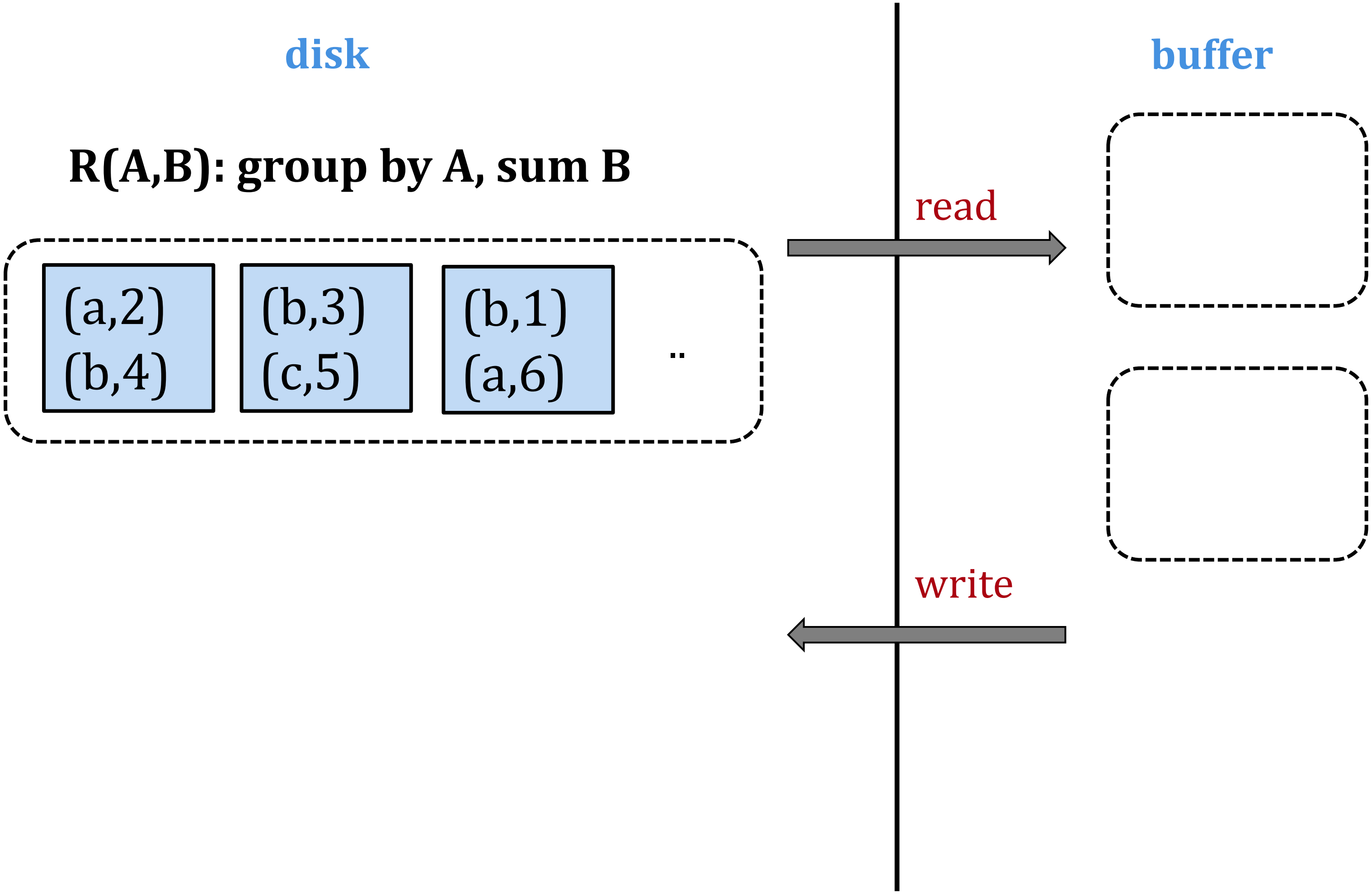
- Sort-based:
  - sort both relations (on *all attributes*)
  - merge sorted relations while eliminating tuples from R that exist in S
- Hash-based:
  - hash-partition R and S
  - build an in-memory hash table for each partition S_i
  - probe with tuples in R_i, add to the output result if the tuple does not exist in the hash table

AGGREGATION

AGGREGATION: SORTING

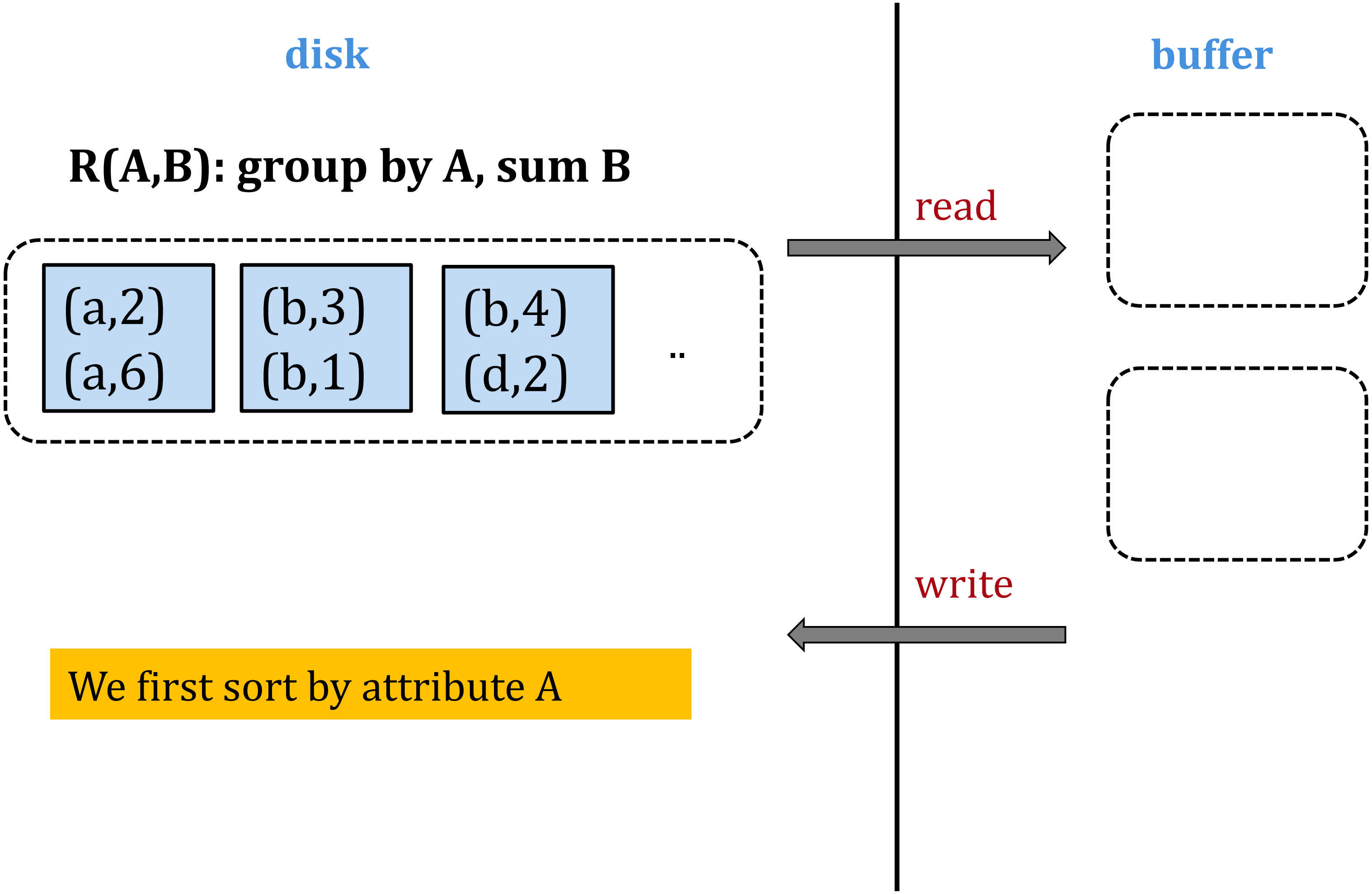
- sort on group by attributes (if any)
- scan sorted tuples, computing running aggregate
  - max/min: max/min
  - average: sum, count
- when the group by attribute changes, output aggregate result
- **cost** = sorting cost

EXAMPLE



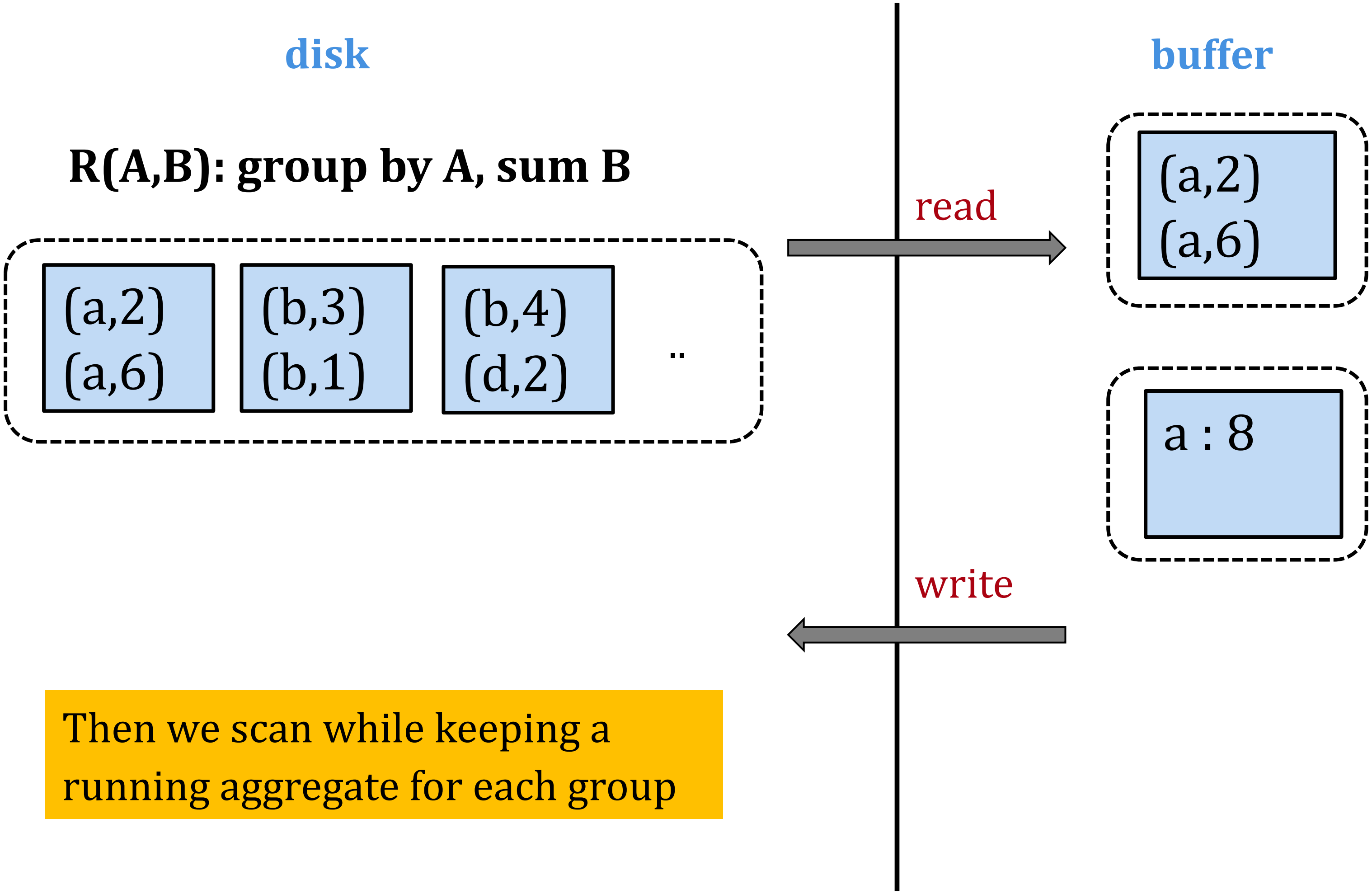
EXAMPLE

R(A,B): group by A, sum B



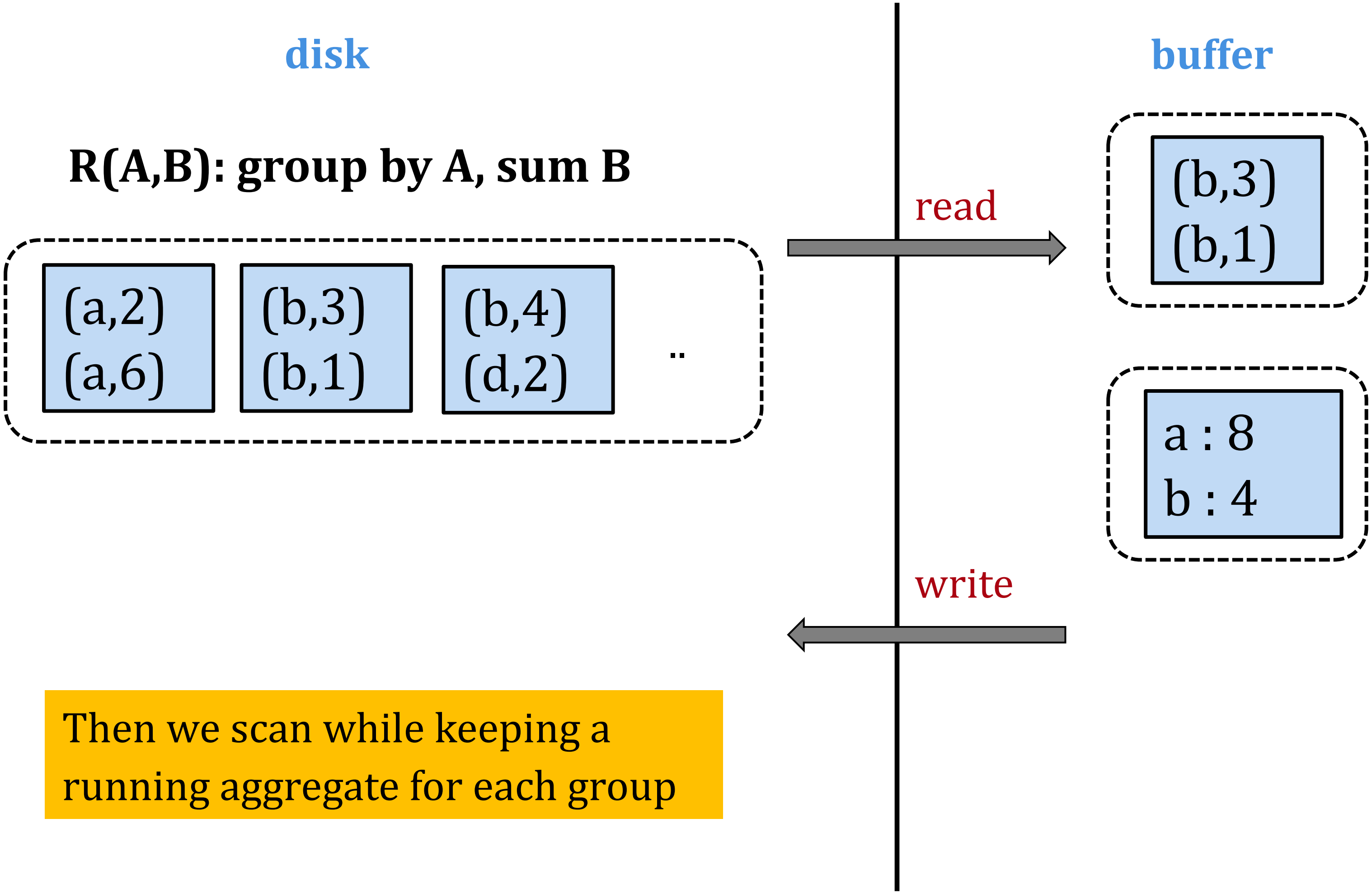
We first sort by attribute A

EXAMPLE



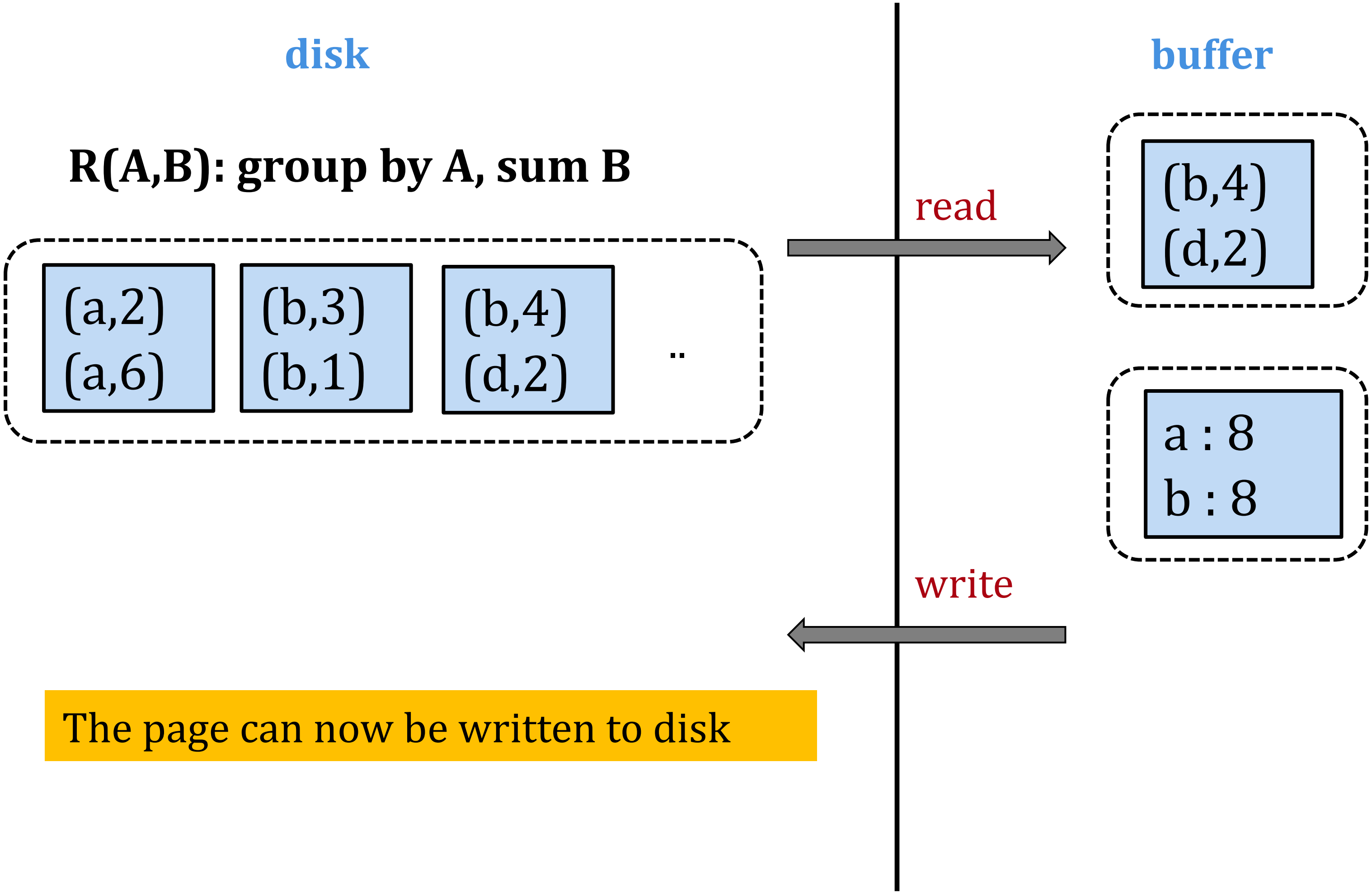
Then we scan while keeping a running aggregate for each group

EXAMPLE



Then we scan while keeping a running aggregate for each group

EXAMPLE



The page can now be written to disk

AGGREGATION: HASHING

- 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

What happens if we have too many groups?