EXTERNAL SORTING

GOALS

- Why Sorting is important
- Difference between sorting in memory and on disk
- How external merge-sort works
- B+ trees for sorting

WHY SORTING

WHEN DOES A DBMS SORT DATA?

- Users request data sorted
- Bulk loading a B+tree (first step)
- Eliminating duplicates
- Join algorithms that use sorted data

Building block in query evaluation

THE PROBLEM

How do we sort 100Gb of data with 2Gb of RAM

B+TREES FOR SORTING

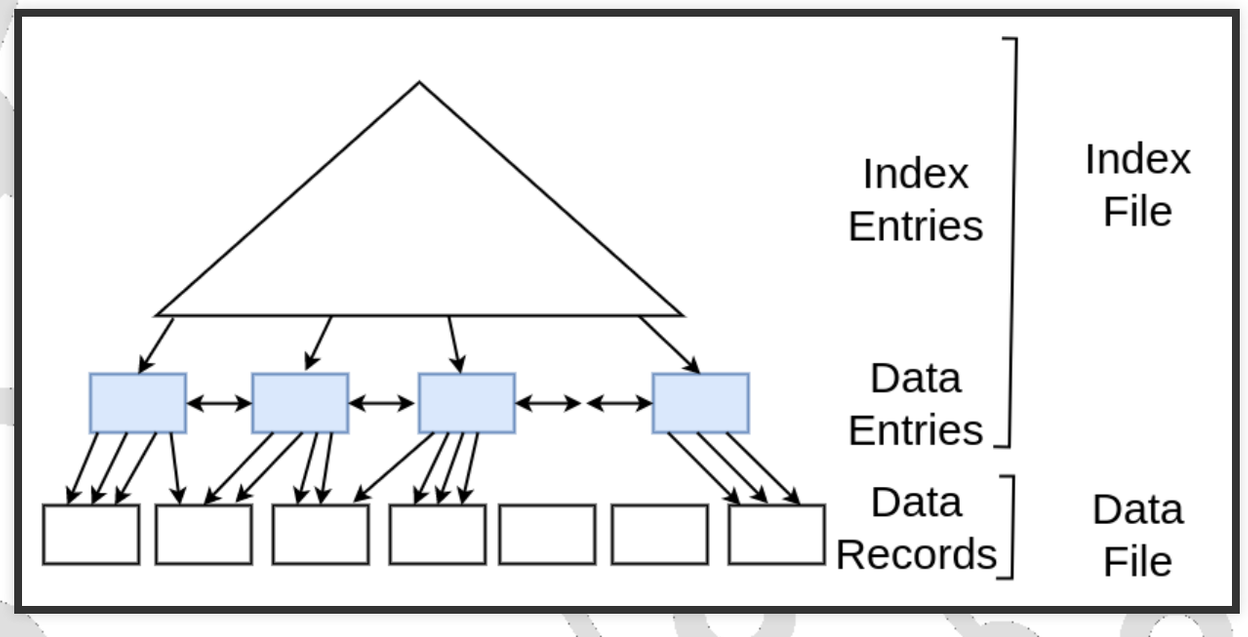
USING B+ TREES FOR SORTING

Scenario: Table to be sorted has B+ tree index on sorting column(s).

Idea: Can retrieve records in order by traversing leaf pages.

Is this a good idea?

CLUSTERED INDEX

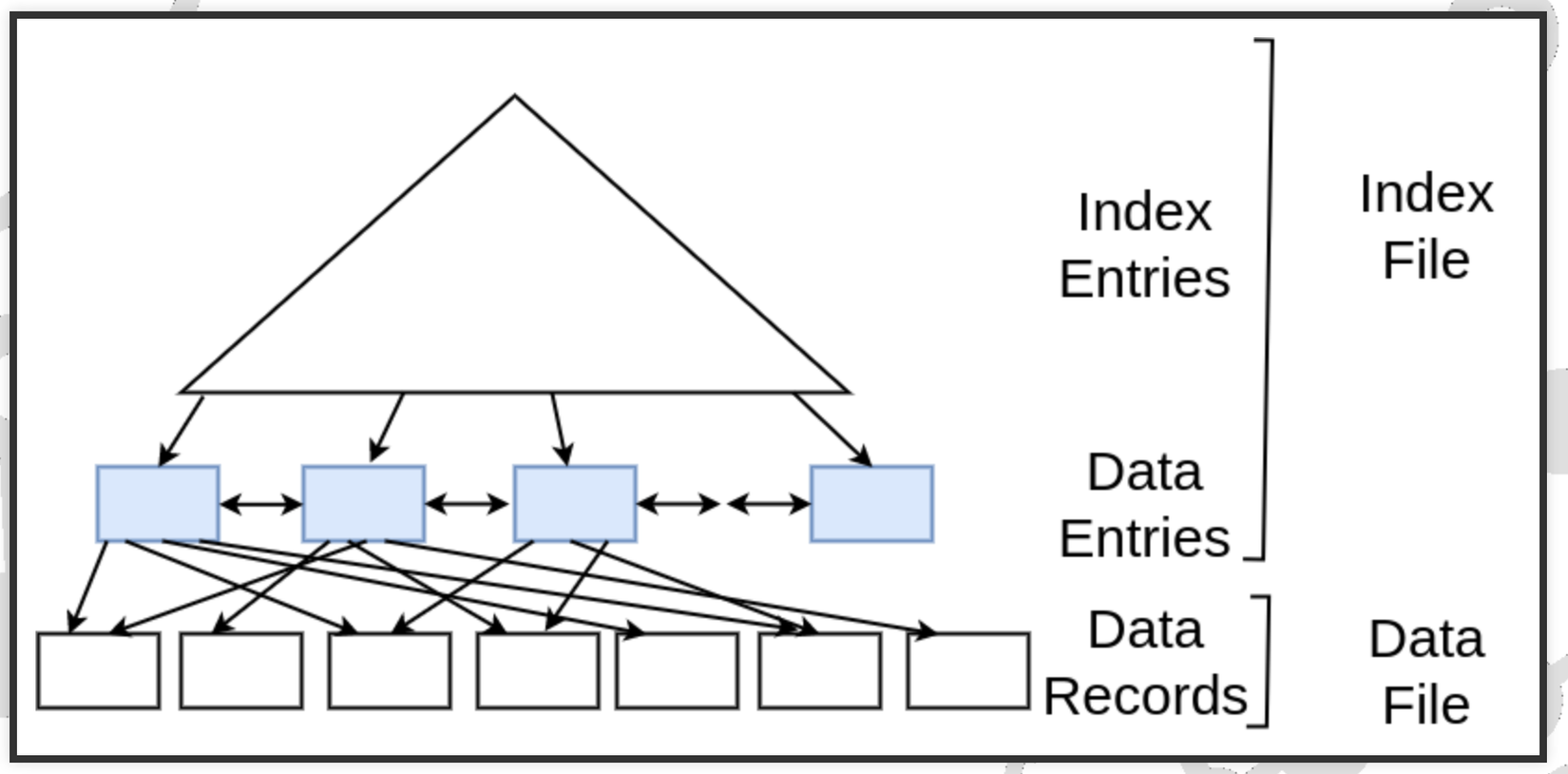


What is the cost?

- root to the left-most leaf, then retrieve all leaf pages (Alternative 1)
- Data entry is <key, rid> (Alternative 2)
  - Additional cost of retrieving data records: each page fetched just once.

Always better than external sorting!

UNCLUSTERED INDEX



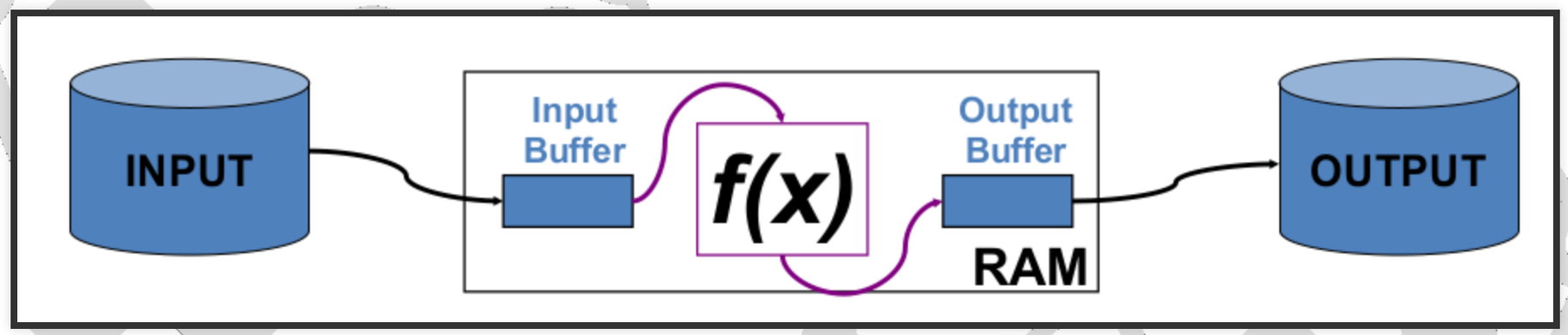
Each data entry contains *rid* of a data record.

In general, **one I/O per data record!**

EXTERNAL DATA PROCESSING 101

How is external data processing performed?

Streaming data through RAM: an important method for sorting & other DB operations



EXTERNAL DATA PROCESSING 101

Simple Case

- Compute f(x) for each record, write out the result
- Read a page from INPUT to Input Buffer
- Write f(x) for each item to Output Buffer
- When Input Buffer is consumed, read another page
- When Output Buffer fills, write it to OUTPUT

EXTERNAL DATA PROCESSING 101

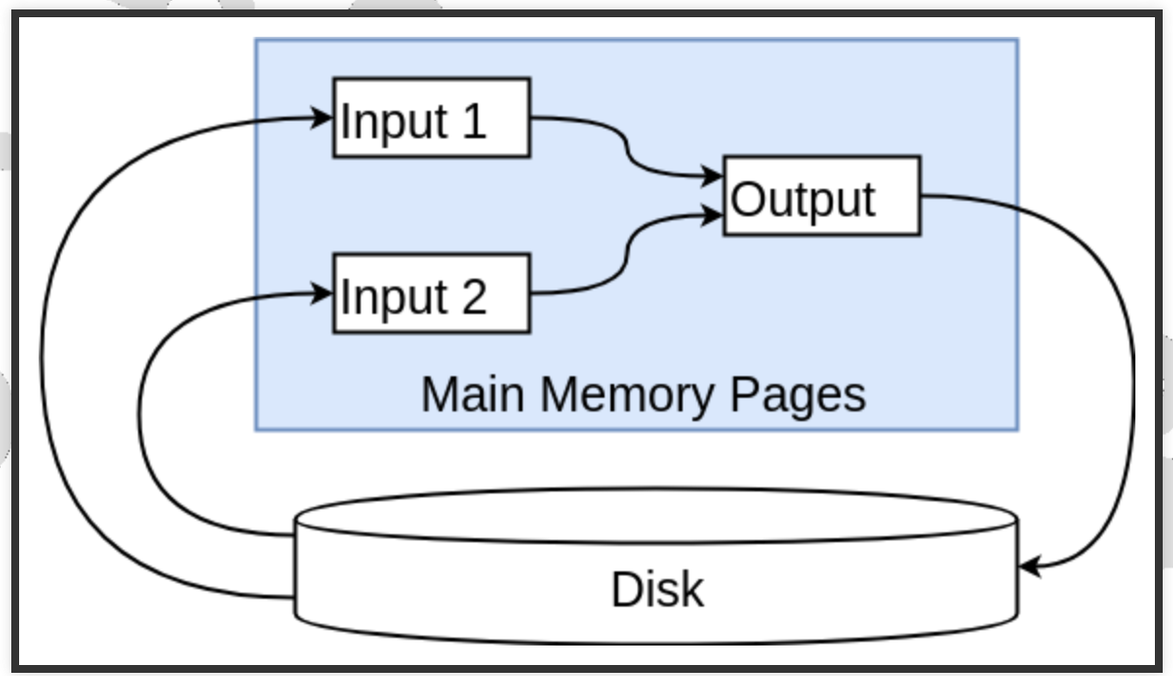
Reads and Writes might not be coordinated

- E.g., if f() is Compress(), you read many pages per write.
- E.g., if f() is DeCompress(), you write many pages per read.

SIMPLE TWO-WAY MERGE SORT

Simple Algorithm to help us understand external sorting

Uses only 3 memory pages



RUN (FORREST RUN)

Break file up into smaller files that fit into memory, sort each of the files using in memory sort algorithm.

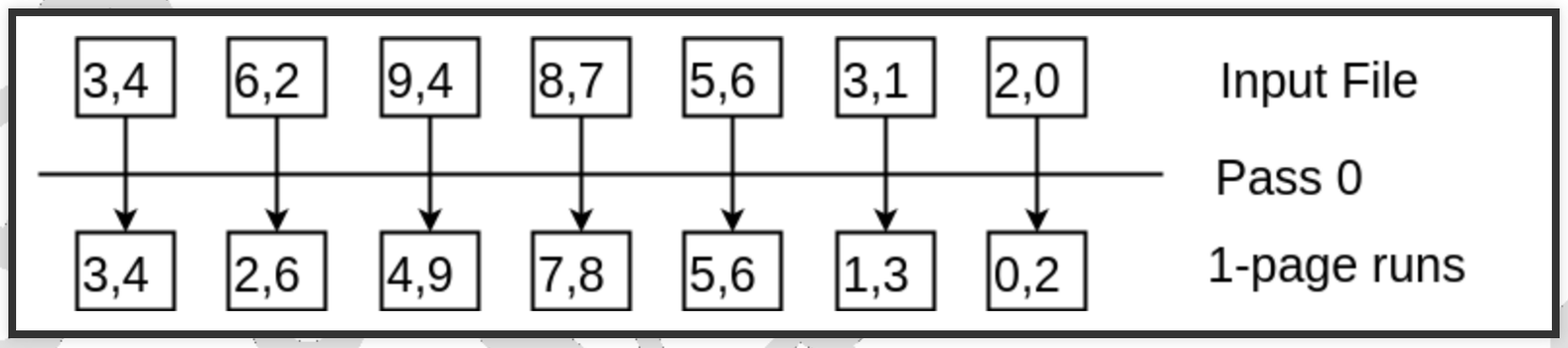
- **Run:** Each intermediate sorted subfile

Iterate all data from original file in a **pass**

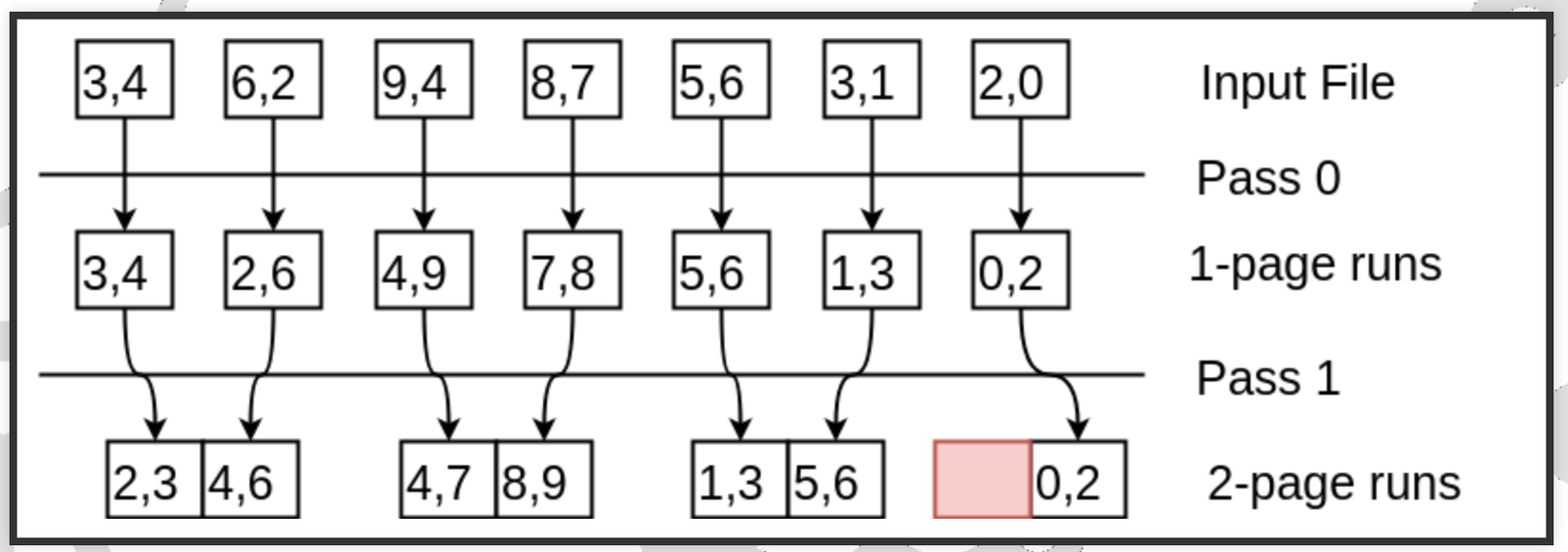
- **Divide and conquer:** sort sub-files and merge

2-WAY-EXT-SORT ALGORITHM - EXAMPLE

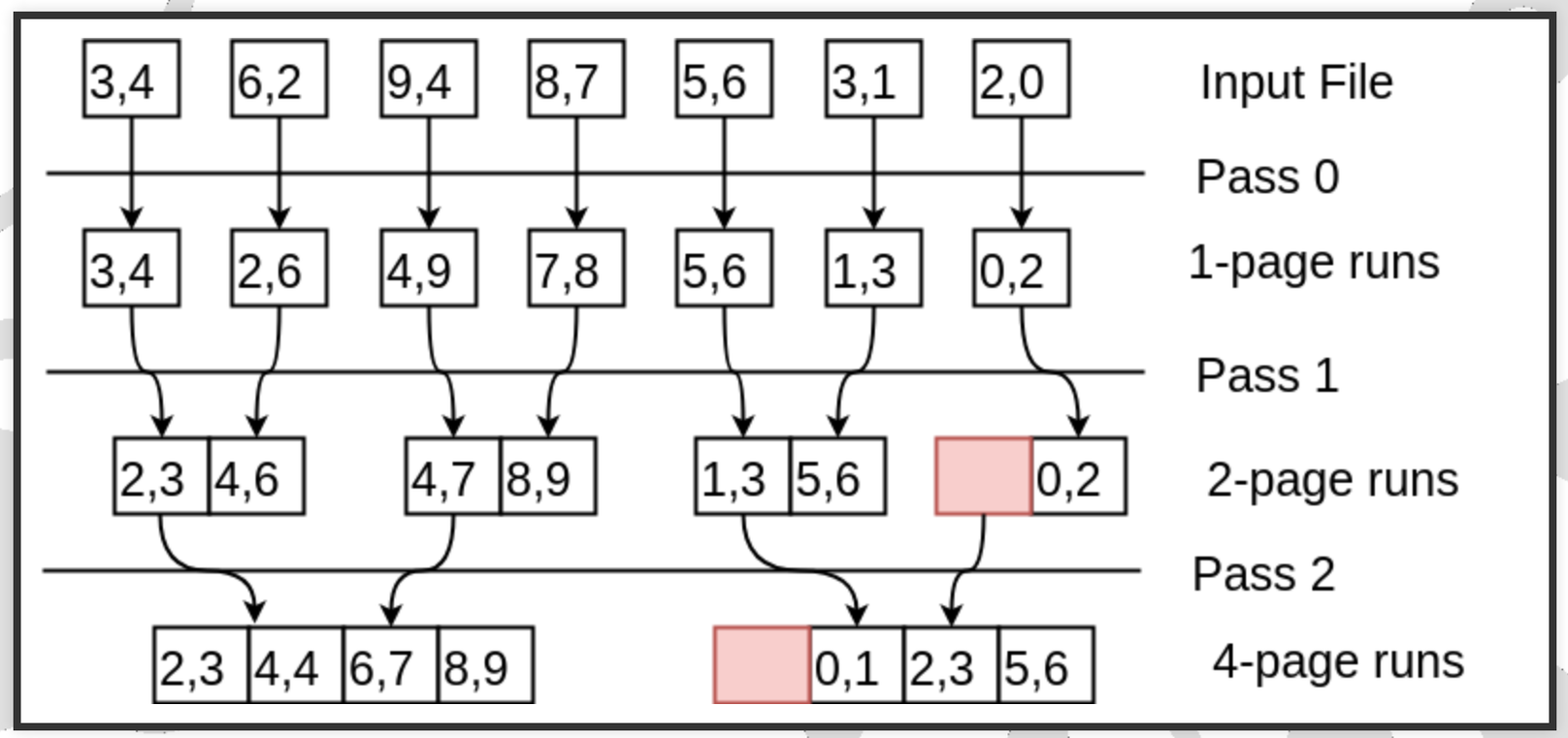
2-WAY-EXT-SORT ALGORITHM - EXAMPLE



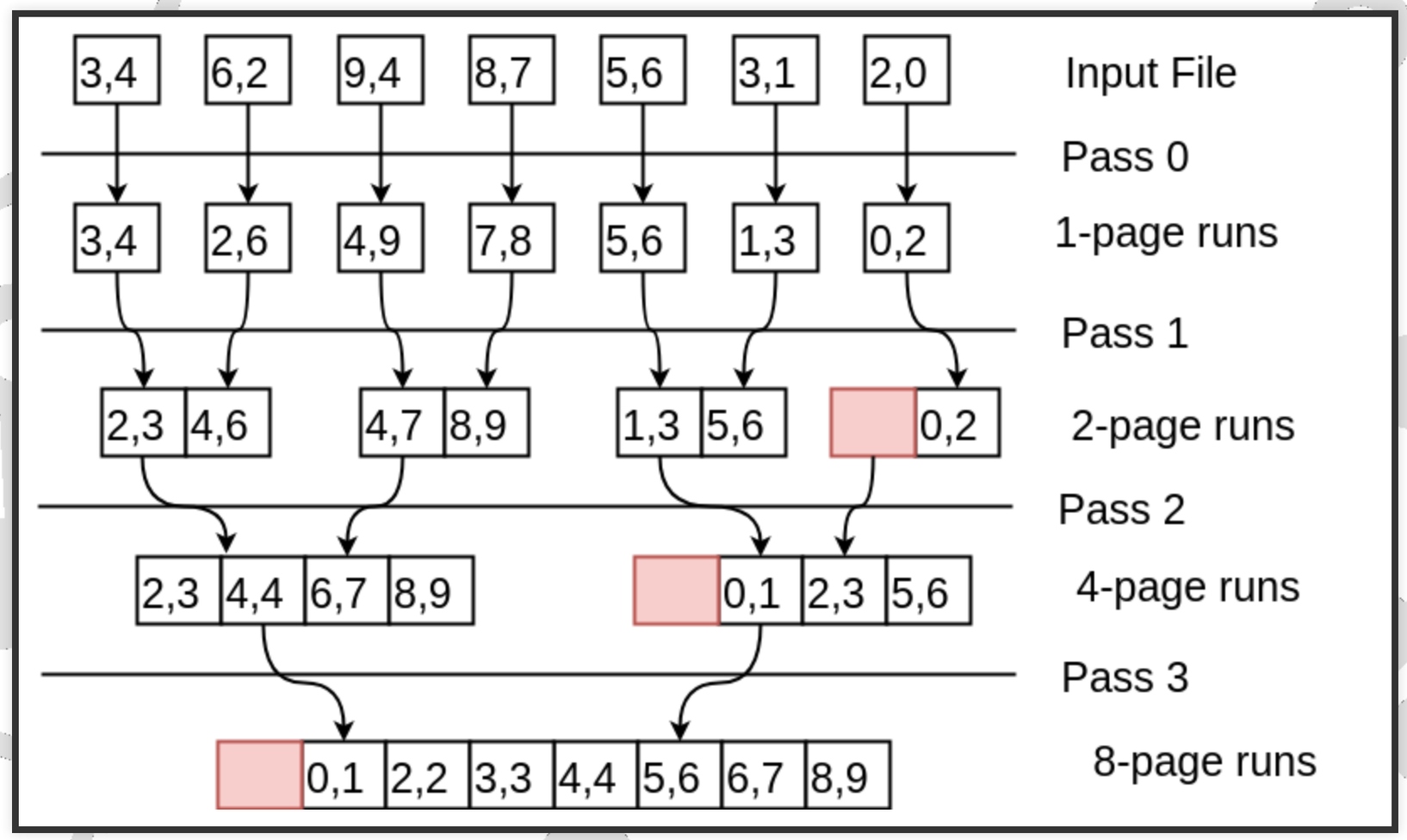
2-WAY-EXT-SORT ALGORITHM - EXAMPLE



2-WAY-EXT-SORT ALGORITHM - EXAMPLE



2-WAY-EXT-SORT ALGORITHM - EXAMPLE



HOW MANY RUNS

If the number of pages in the input is 2^k for some k then:

- Pass 0 produces 2^k sorted runs of 1 page each
- Pass 1 produces 2^{k-1} sorted runs of 2 pages each
- Pass 2 produces 2^{k-2} sorted runs of 4 pages each …
- Pass k produces 1 sorted run of 2^k pages

COST OF A PASS

In each pass

- Read every page in file
- Process it
- Write it out

So 2 disk I/O per page per pass

NUMBER OF PASSES

Number of passes

$$\lceil \log_2(N) \rceil + 1$$

Where N is the number of pages in the file

COST OF A SORT

Total cost

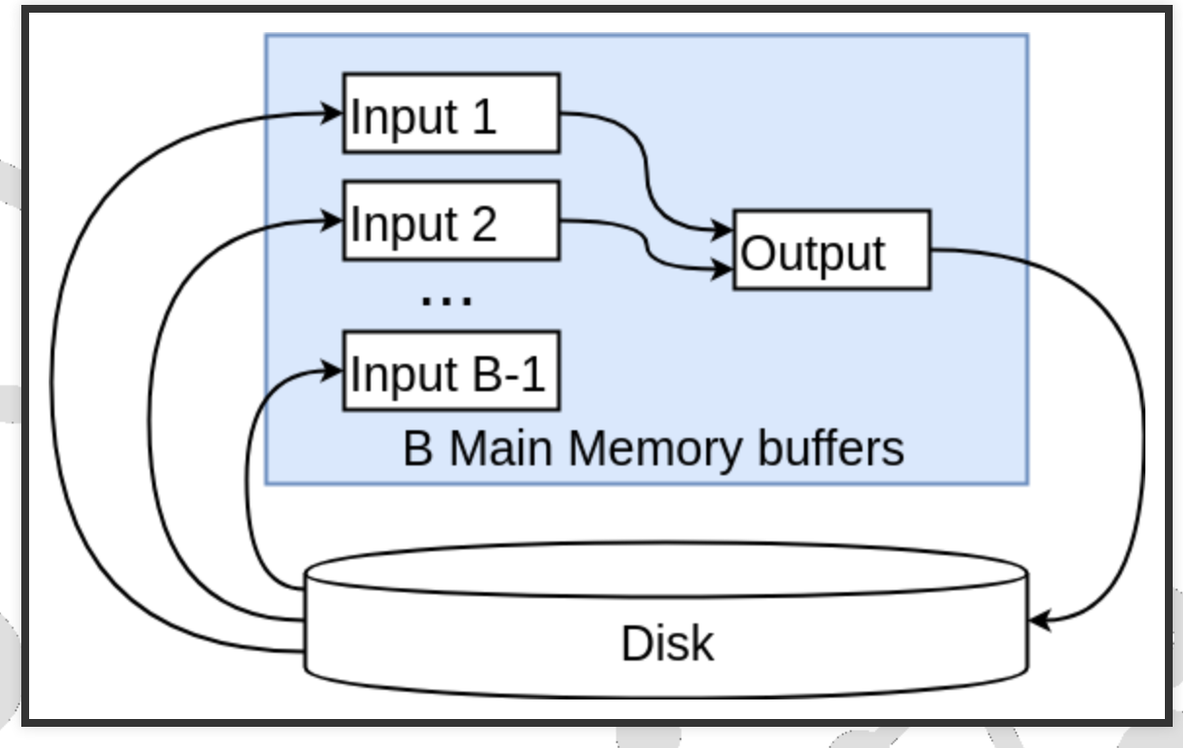
$2N(\lceil \log_2(N) \rceil + 1)$ I/Os

From example: N=7, $2*7(\lceil \log_2(7) \rceil + 1) = 56$

EXTERNAL MERGE SORT

More than 3 buffer pages. How can we utilize them?

GENERAL EXTERNAL MERGE SORT

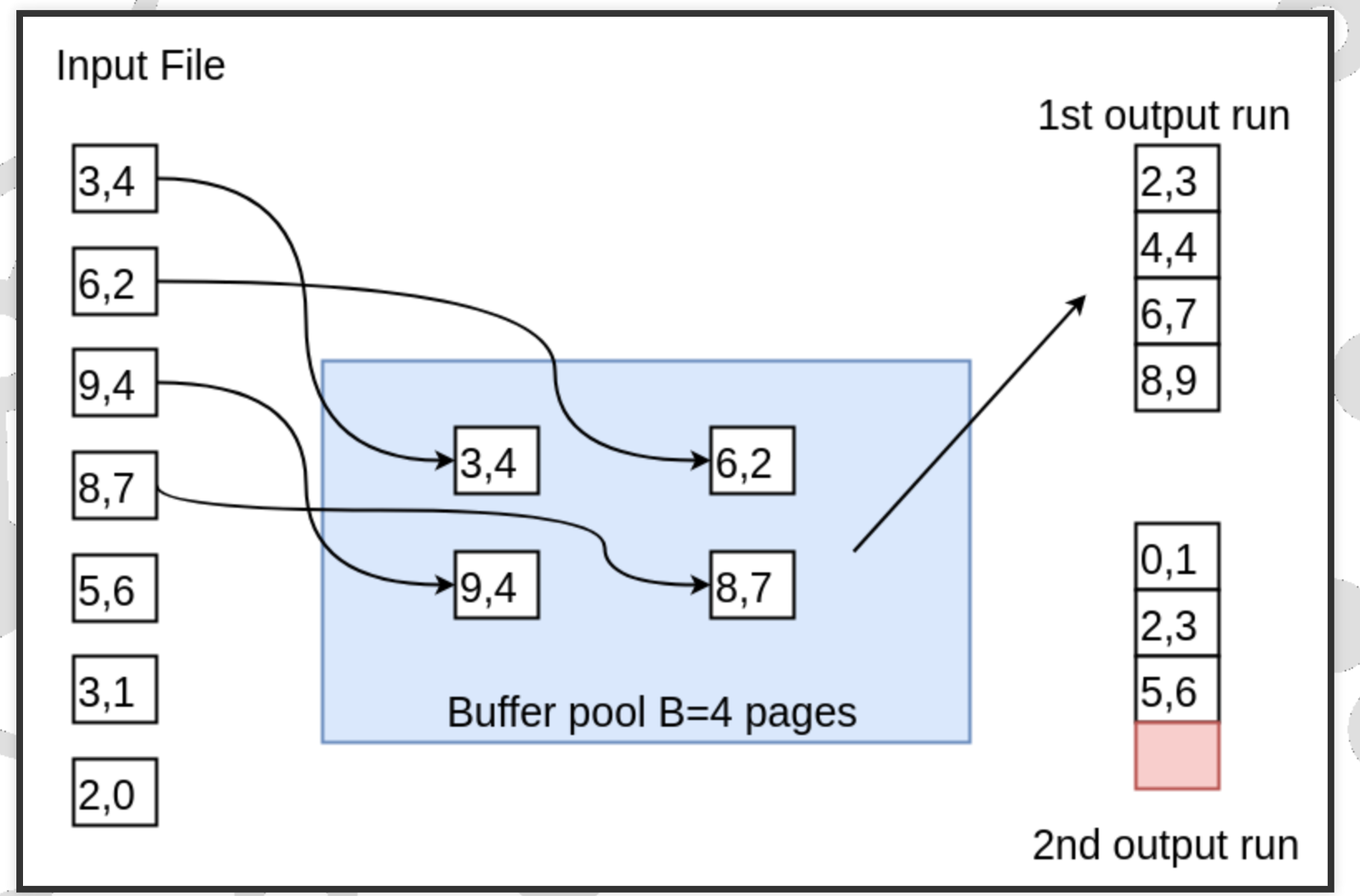


GENERAL EXTERNAL MERGE SORT

To sort a file with N pages using B buffer pages:

- Pass 0: use B buffer pages.
  - Produce ⌈ N / B ⌉ sorted runs of B pages each.
- Pass 1, 2, …, etc.: merge B-1 runs.

EXTERNAL MERGE SORT



EXTERNAL MERGE SORT

```
function extsort(file)
  // Given file on disk, sort it using B buffer pages
  // Produce runs that are B pages long: Pass 0
  Read B pages into memory, sort them, write out a run
  // Merge B-1 runs at a time to produce longer runs
  // Do so until only 1 run exist
  while the number of runs at end of previous pass > 1
    // Pass i= 1, 2, ...
    while there are runs to be merged from previous pass
      Choose next B-1 runs (from previous pass)
      Read each run into an input buffer, 1 page at a time
      Merge the runs and write the output to output buffer
      force output to disk one page at a time
```

COST OF EXTERNAL MERGE SORT

Cost = 2 * N * (# of passes)

With B buffer pages to sort N page file, number of passes:

$$1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$$

CAN WE REDUCE THE NUMBER OF PASSES OF EXTERNAL MERGE SORT?

#passes depends on

1. fan-in during the merge processes (decided by the #buffer pages)
2. the number of runs at the leaf level (produced by pass 0)

CAN WE PRODUCE RUNS LONGER THAN THE SIZE OF RAM?

MINIMIZING THE NUMBER OF RUNS

In initial pass, we can refill an empty page and continue a run, as long as we have items for it.

Tournament Sort aka "Heapsort" / "Replacement Sort"

TOURNAMENT SORT

- Use 1 buffer page as the input buffer and 1 buffer page for as the output buffer
- Maintain 2 Heap structure in the remaining B-2 pages (H1, H2)
- Read B-2 pages of records and insert them into H1
- While there still records left
  - get the “min” record *m* from H1 and send to the output buffer
  - If H1 is empty then start a new run and put all the records in H2 to H1
  - read another record *r* from the input buffer
  - if $r < m$ then put it in H2, otherwise put it in H1
- Finish the current run by outputing all the records in H1
- If there is sth left in H2, output them as a new run.

WHAT IS THE AVERAGE LENGTH OF A RUN IN HEAPSORT?

B-2 pages are used for the heaps

(B-2) + 1/2 (B-2) + 1/4 (B-2) + 1/8 (B-2) + … ≈ 2(B-2)

So we can double the length of initial runs

MINIMIZING I/O COST VS NUMBER OF I/OS

I/O FOR EXTERNAL MERGE SORT

Do I/O a page at a time – Not one I/O per record

In fact, reading a block *(chunk)* of pages sequentially is cheaper than reading them one at a time!

Suggests we should make each buffer (input/output) be a **block** of pages.

- But this will reduce fan-in during merge passes!
- In practice, most files still sorted in 2-3 passes.

BLOCKED I/O

Read/Write in unit of **b** pages

b pages allocated as output buffer

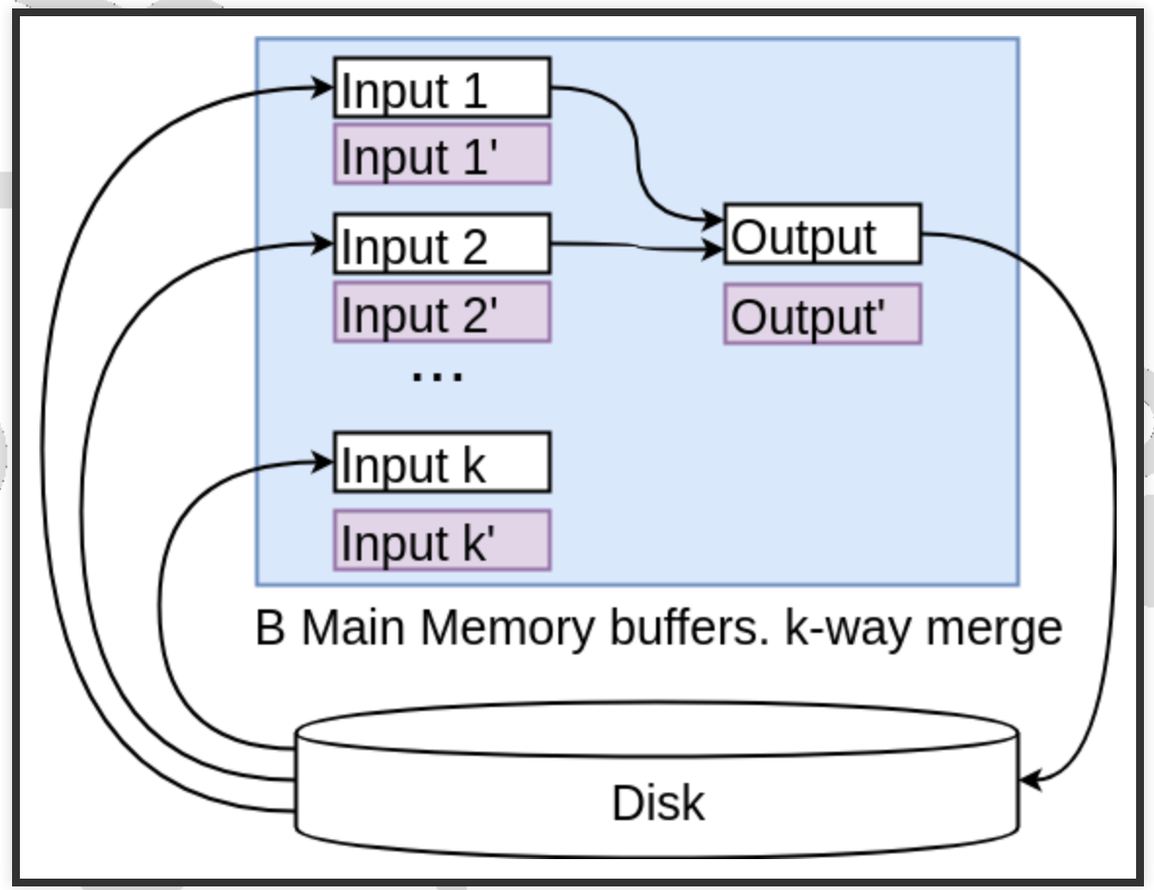
Number of runs could be merged in one pass F = ⌈ (B-b)/b ⌉

With heapsort, the number of initial runs N2 = ⌈ N/2B ⌉

The number of passes: ⌈ $\log_F$ N2 ⌉ + 1

DOUBLE BUFFERING

To reduce wait time for I/O request to complete, can prefetch into "shadow block".



NUMBER OF PASSES OF EXTERNAL SORT

N	B=3	B=5	B=9	B=17	B=129	B=257
100	7	4	3	2	1	1
1,000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000	17	9	6	5	3	3
1,000,000	20	10	7	5	3	3
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4

SUMMARY

- External sorting is important
- External merge sort minimizes disk I/O cost:
  - Pass 0: Produces sorted runs of size B (# buffer pages). Later passes: merge runs.
  - # of runs merged at a time depends on B, and block size.
  - Larger block size means less I/O cost per page.
  - Larger block size means smaller # runs merged.
  - In practice, # of passes rarely more than 2 or 3.
- Clustered B+ tree is good for sorting; unclustered tree is usually very bad.

QUESTIONS