UNIVERSITY OF COPENHAGEN

Data Processing: Basic Concepts and External Sorting

ACS, Yongluan Zhou & Marcos Vaz Salles

with slides revised by Michael Kirkedal Carøe, originally from *Database Management Systems, Ramakrishnan and Gehrke*

Before the break: Read-Write Systems

- On-Line Transaction Processing (**OLTP**)
  - Process multiple, but relatively simple, application functions

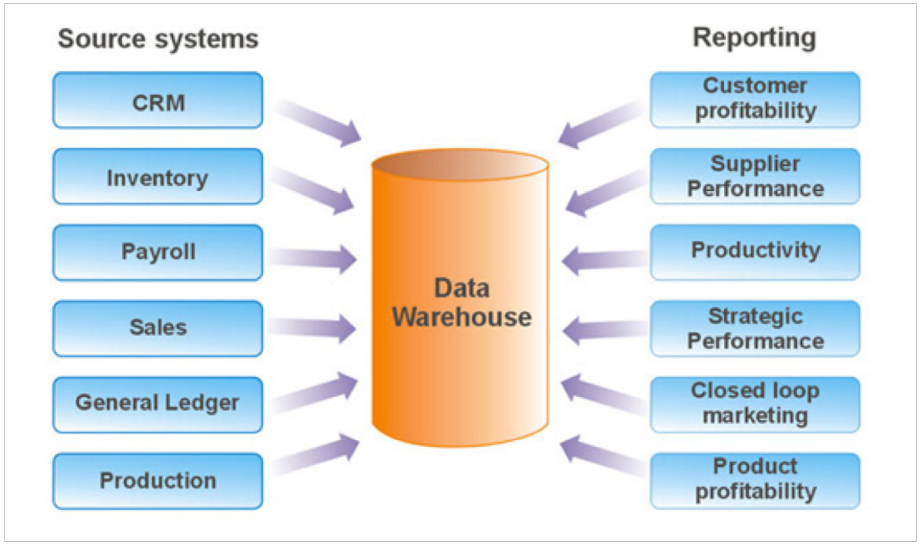
- **Examples**
  - Order processing, e.g., Amazon
  - Item buy/sell in computer games, e.g., EVE Online
  - High-performance trading
  - Updates on social networks, e.g., Facebook

Before the break: Topics of study

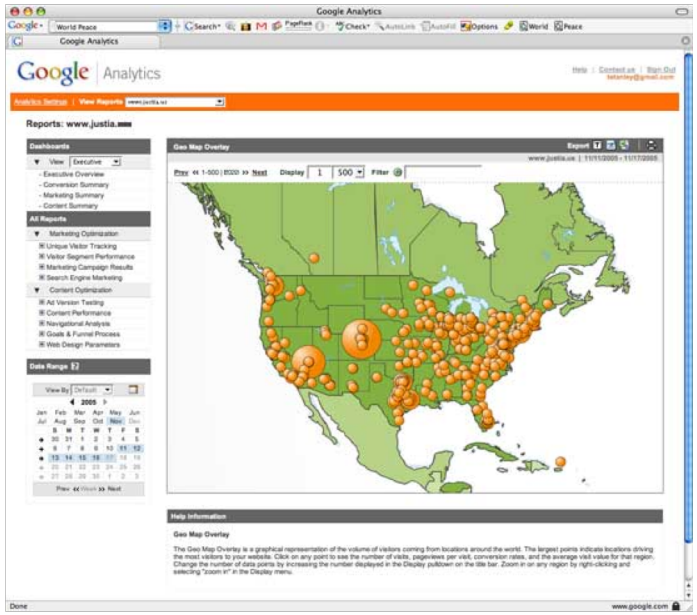
- **Property: Strong Modularity**
  - Fundamental abstractions
  - RPCs, techniques for performance
- **Properties: Atomicity and Durability**
  - Concurrency control (2PL, OCC)
  - Recovery (ARIES)
- **Property: High Availability**
  - Reliability
  - Replication
  - Communication
- **Experimental Design**

The consequence of our success: Read-Intensive Systems

- **Data warehouses**
  - Consolidating data from different sources
  - Querying and aggregation



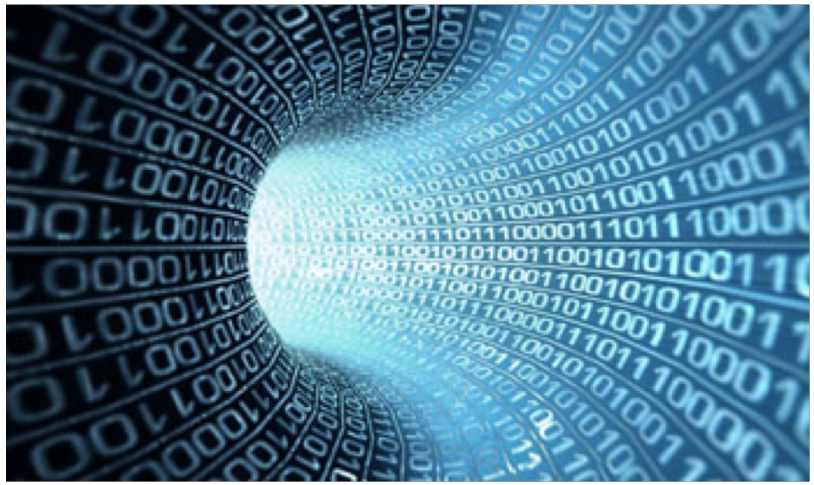
- **Analytics**
  - Deriving value from loosely structured data
  - Machine learning and data mining



- **Social media analysis, web, visualization, others**

Scalability vs. Expressiveness is fundamental trade-off

- Property: Scalability with Data Size
  - Must use I/O resources wisely
  - Hopefully linear increase in hardware leads to linear increase in performance

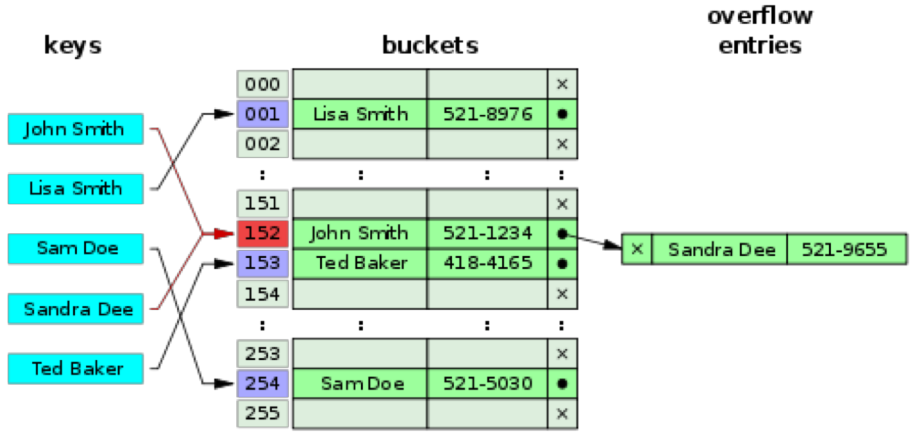


- Data Processing
  - Operators, typically relational
  - External sorting
  - Hash- and sort-based techniques for multiple operations (e.g., set operations, joins)
  - Parallelism

Do-it-yourself recap: Basic Algorithms

- We will need some algorithmic building blocks
- **Merge sort:** Explain how the following array would be sorted →

  { 3, 4, 6, 2, 9, 4, 8, 7 }
- **Hashing:** What is a hash function and a hash table?
- **B+ Trees:** What is a B+ Tree? What is the difference between a B+ Tree and a binary tree?



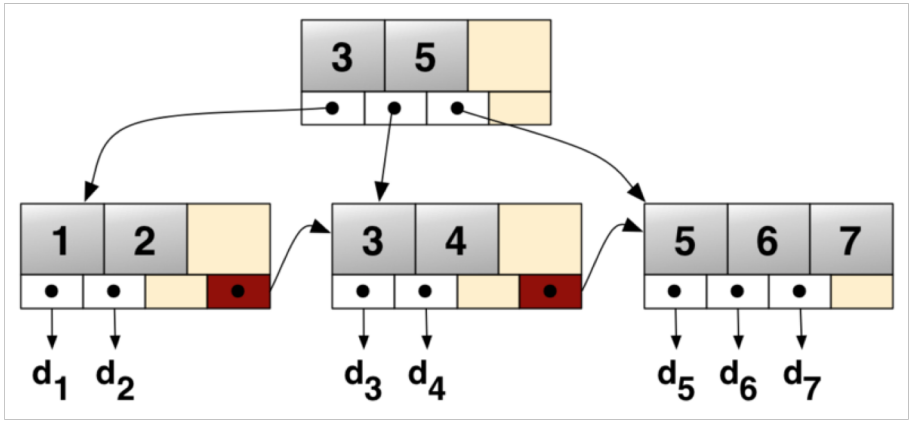
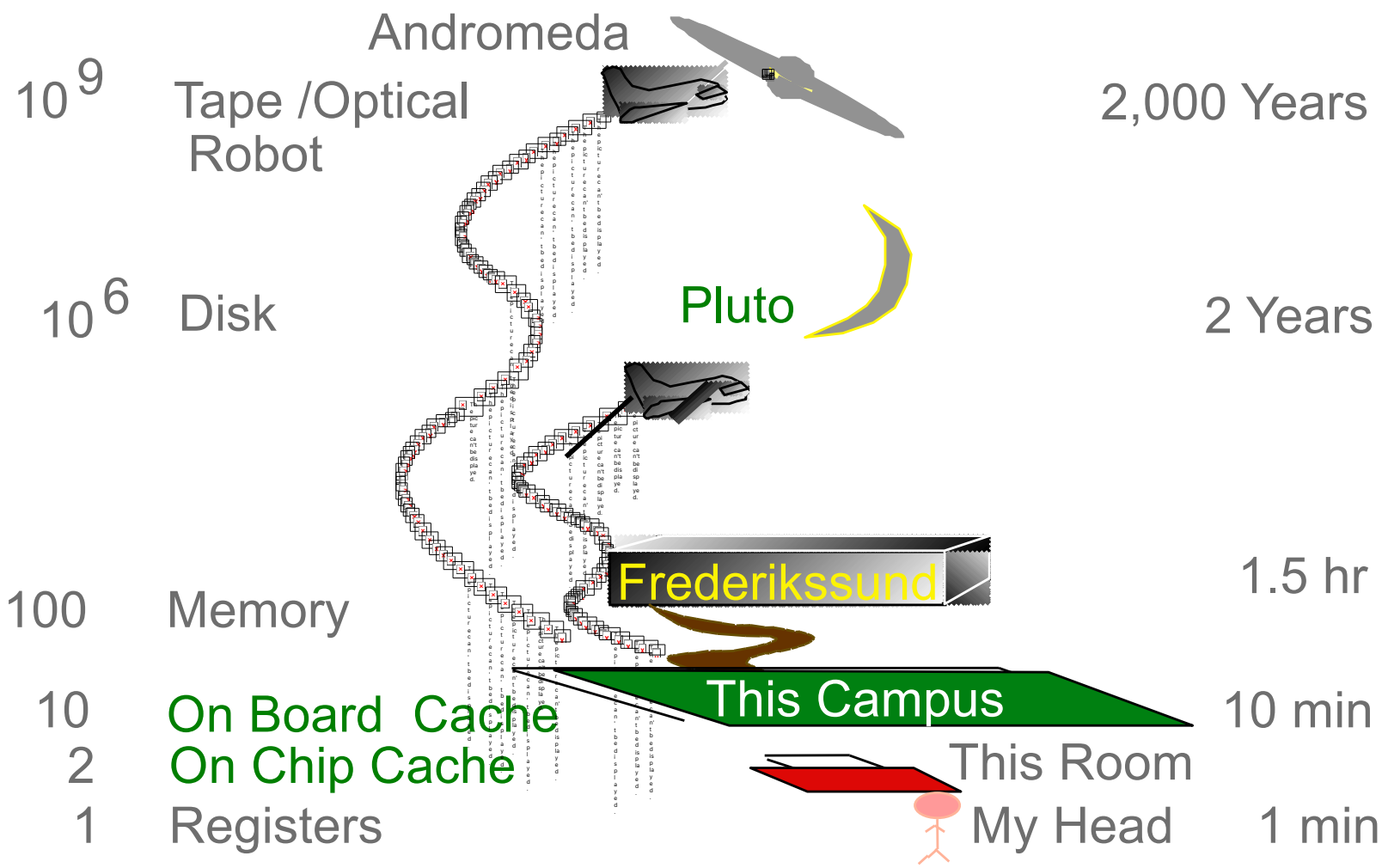


Image source: Wikipedia

What should we learn today?

- Reason about cost models and algorithmic design decisions when processing BIG data, such that I/Os and not only RAM operations are necessary
- Explain the need for an external sorting algorithm, compared to simply applying an internal memory algorithm over a two-level memory abstraction
- Explain and apply external sorting based on 2-way and multi-way sort-merge approaches
- Discuss potential optimizations to external sorting, including shadow buffering, use of indexes, and replacement selection

Storage Hierarchy



Source: Gray (partial)

Costing Algorithms over a Storage Hierarchy

- **RAM model**
  - Every basic operation takes constant time
  - Memory access, simple additions, does not matter!
  - Except it does ☺
- **I/O model**
  - Transfer data from disk in large blocks / pages
  - Count number of I/Os performed
  - **Assumption:** I/Os dominate total cost, any I/O as good as another one → not always true
- **More sophisticated cost models**
  - Create cost function which mixes CPU, memory access, I/O costs
  - Differentiate types of access patterns (sequential, random, semi-random, etc)
  - Complexity can grow very high, very quickly

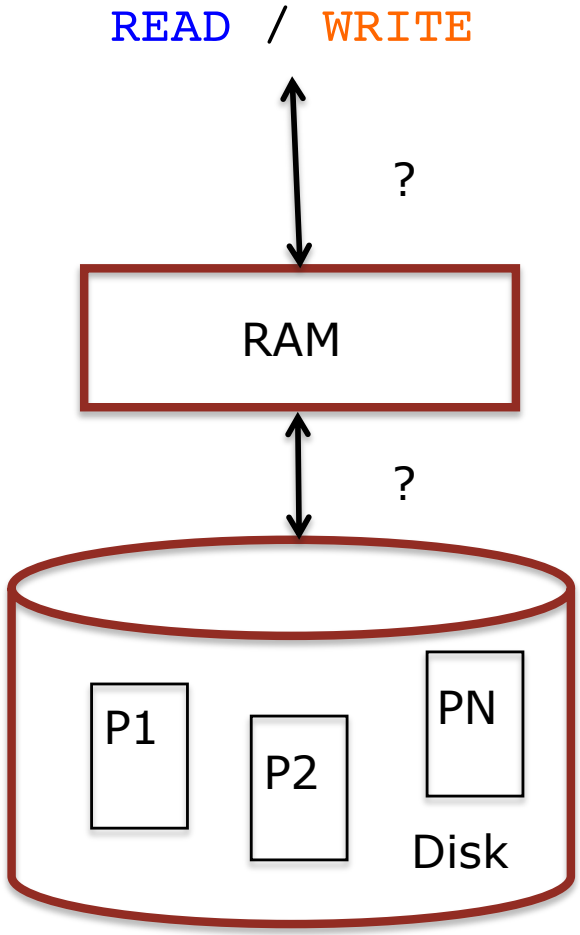
Our first external memory algorithm: Sorting

- **Why sorting?**
- **Important in data processing, relational queries**
  - Used for eliminating duplicates
    - Select DISTINCT …
  - Bulk loading B+ trees
    - Need to first sort leaf level pages
  - Data requested in sorted order
    - SELECT S.name
FROM Sailor S
ORDER BY S.age
  - Some join algorithms use sorting
    - Sort-merge join
  - Some MapReduce implementations perform sort to group keys for reducers

Sorting big data on a small machine?

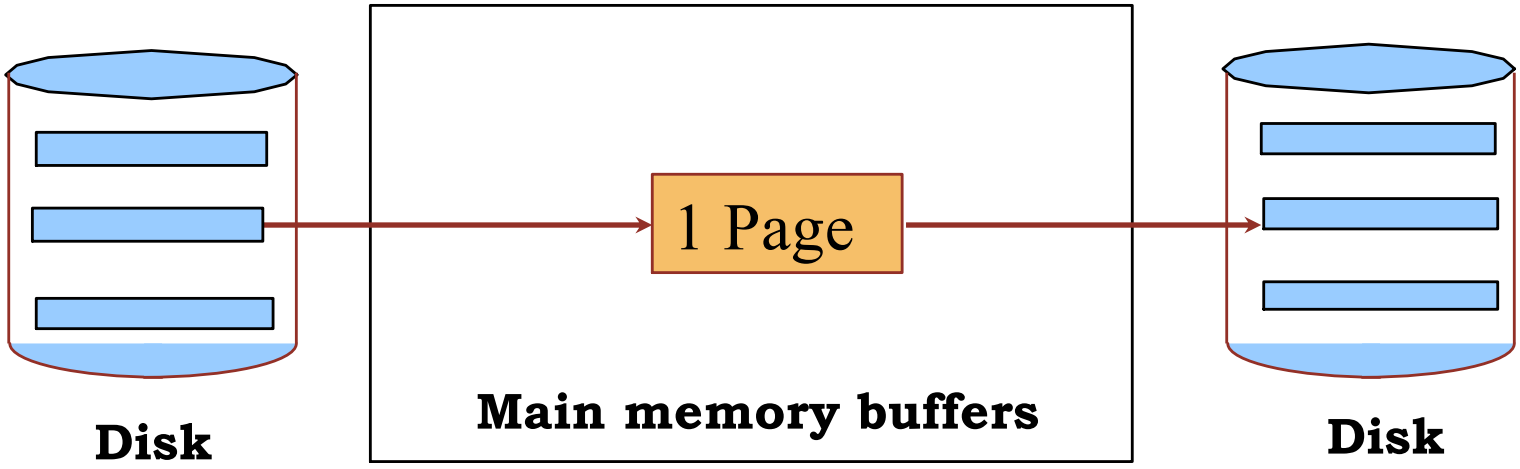
- Say, we want to sort 1 TB of data with 1 GB of RAM

- How would you do it?
- Can't we just use QuickSort and rely on two-level memory & page replacement?
  - Why / Why not?

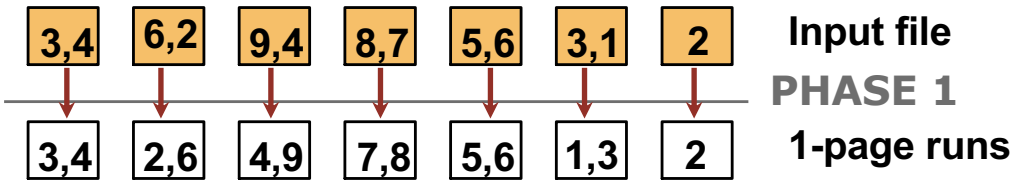


2-Way External Merge Sort: Phase 1

- Based on merge sort
  - Two phases
- Read one page at a time from disk
- Sort it in memory (e.g. quicksort)
- Write it to disk as one temporary file (called "run")
  - Given an input with N pages, Phase 1 produces N runs
- Only one buffer page used



Phase 1: Example



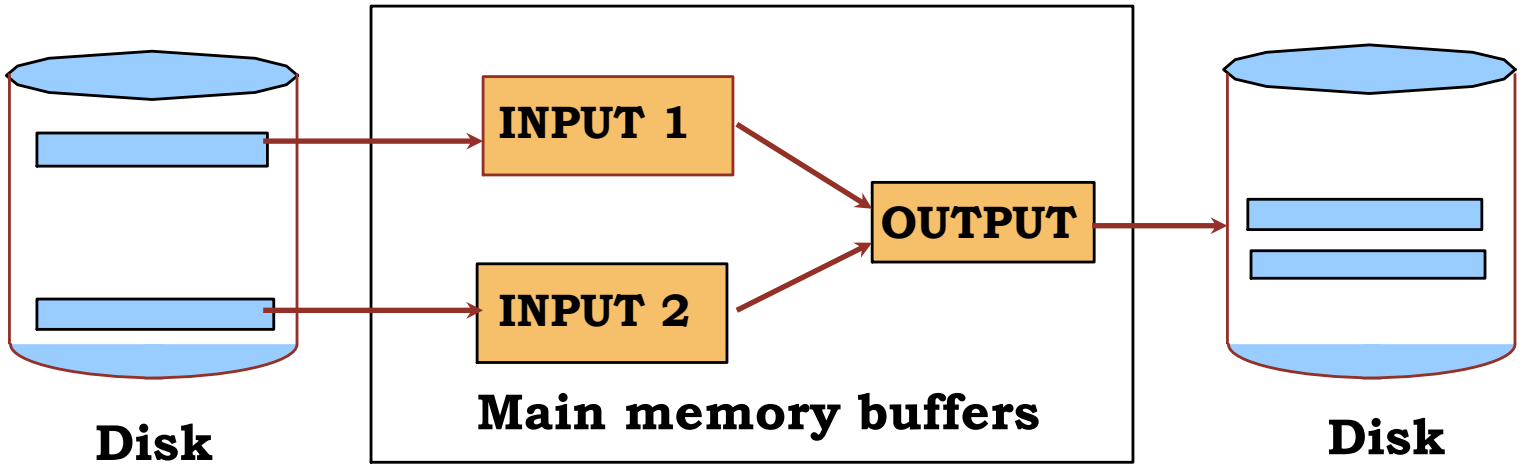
- Assume input file with N data pages of size M
- What is the cost of Phase 1?
  - in terms of # I/O? $2N$
  - in terms of computational steps? O(N M log(M))

2-Way External Merge Sort: Phase 2

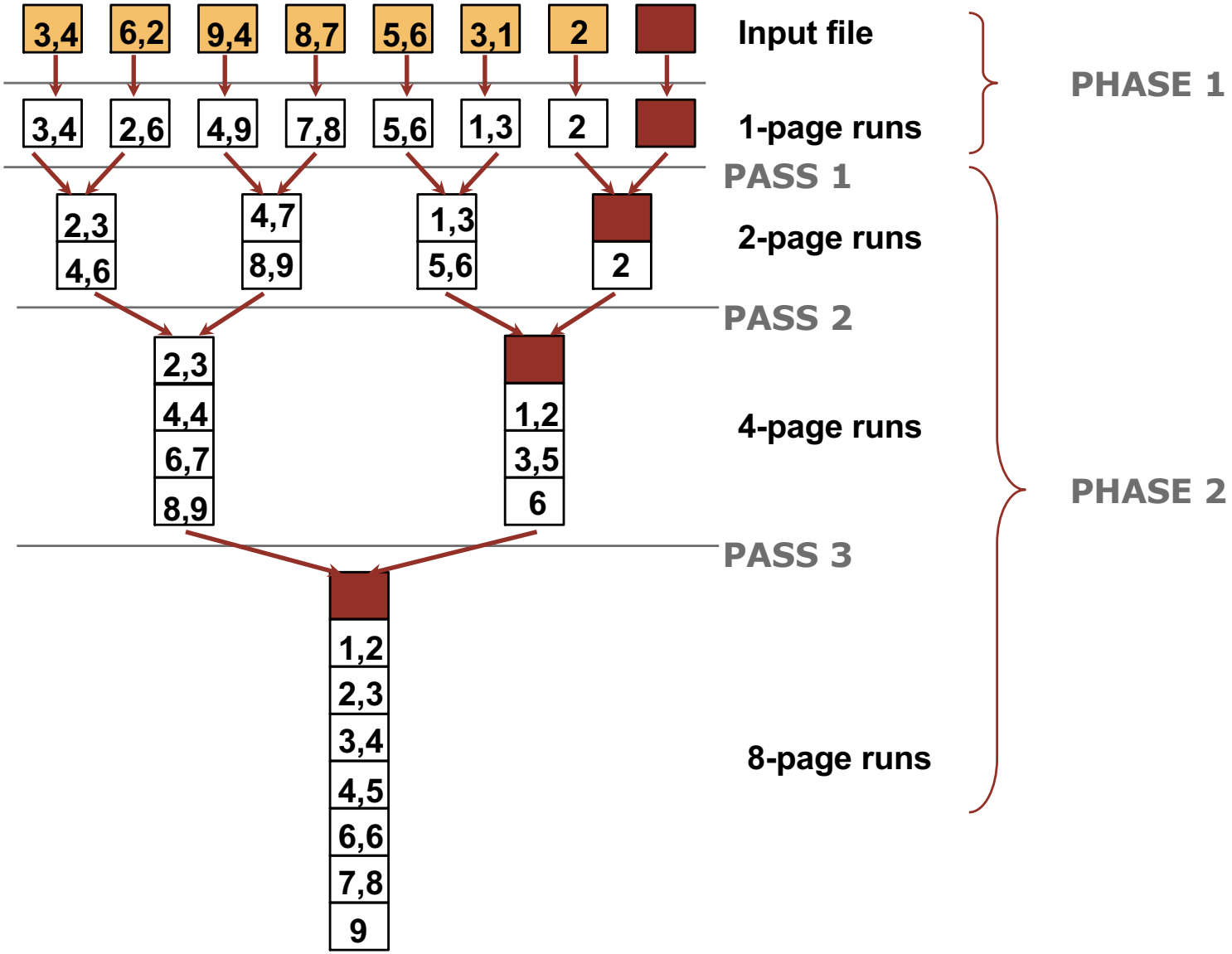
Make multiple passes to merge runs

- Pass 1: Merge two runs of length 1 (page)
- Pass 2: Merge two runs of length 2 (pages)
- … until 1 run of length N

- Three buffer pages used



2-Way External Merge Sort: Example



Two-Way External Merge Sort

- Each pass we read + write each page

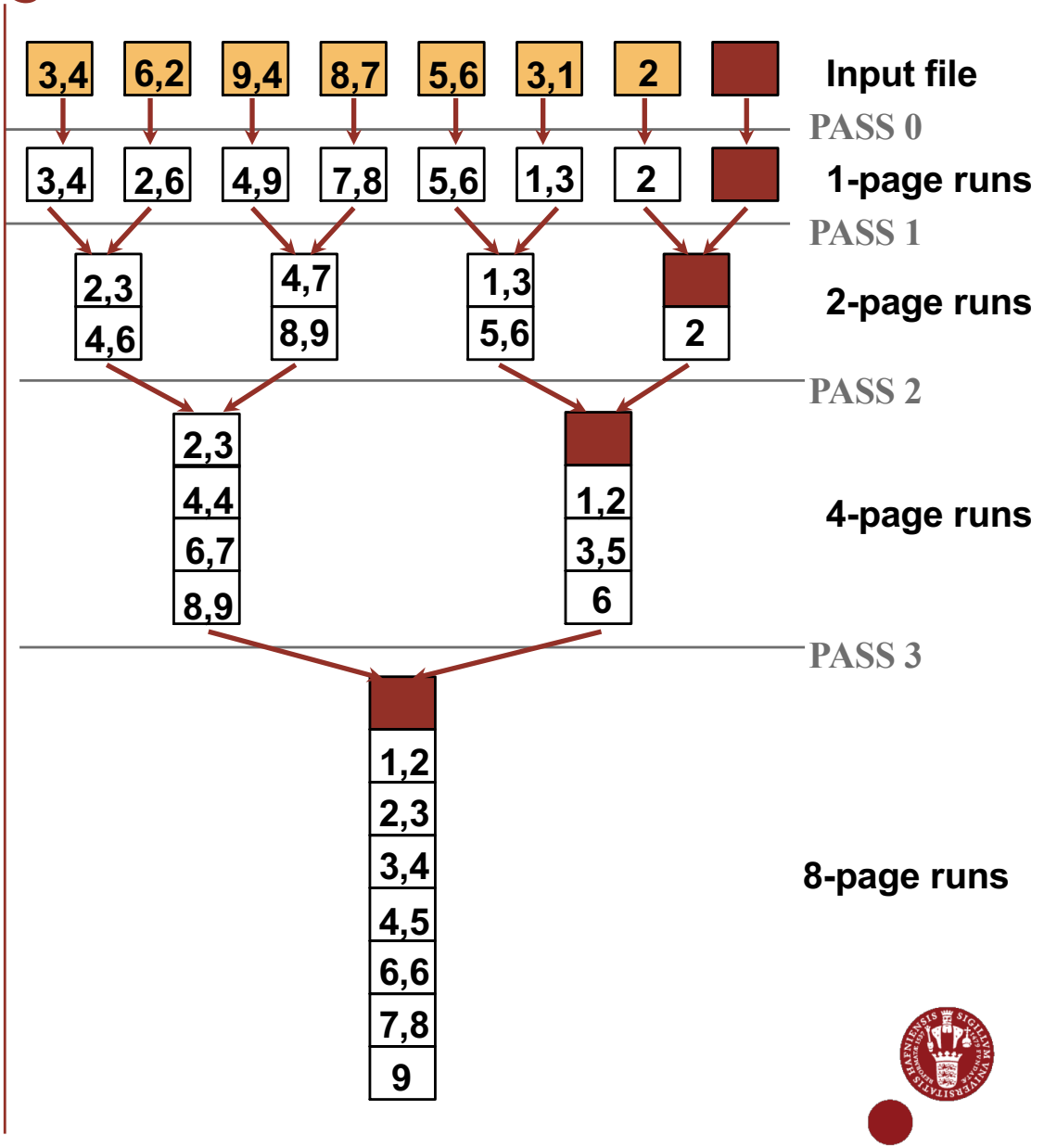
- Given N pages, the number of passes is

$$= \lceil \log_2 N \rceil + 1$$

- So total cost is:

$$2N\left(\lceil \log_2 N \rceil + 1\right)$$

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



2-Way External Merge Sort: Analysis

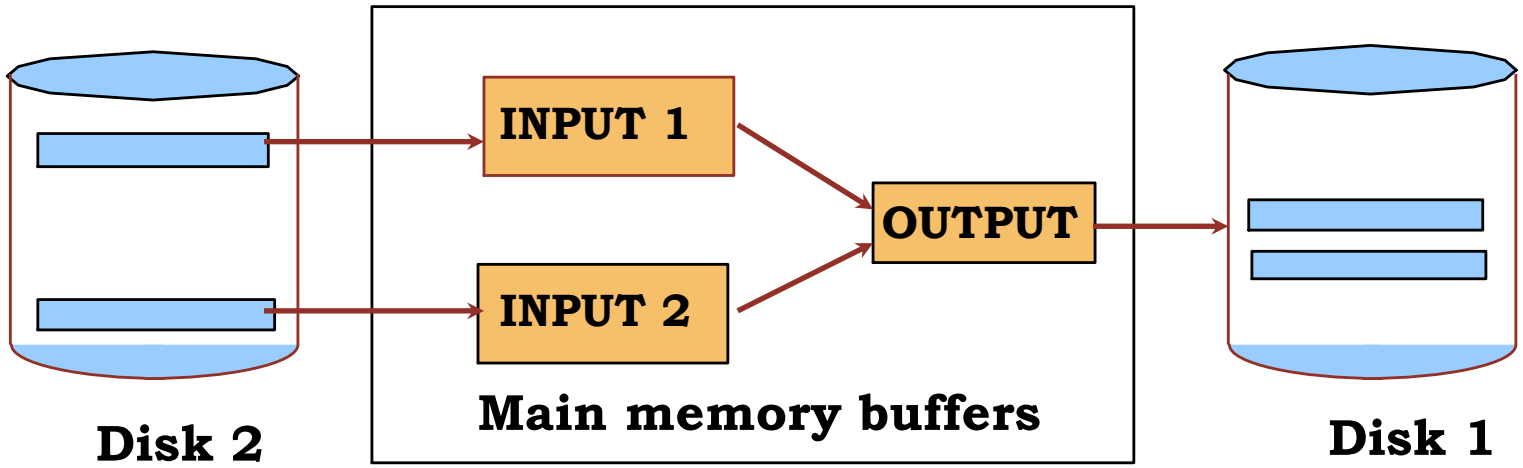
- Total I/O cost for sorting file with N pages

- Cost of Phase 1 = $2N$
- Number of passes in Phase 2 = $\lceil \log_2 N \rceil$
- Cost of each pass in Phase 2 = $2N$

- Cost of Phase 2 = $2N \times \lceil \log_2 N \rceil$

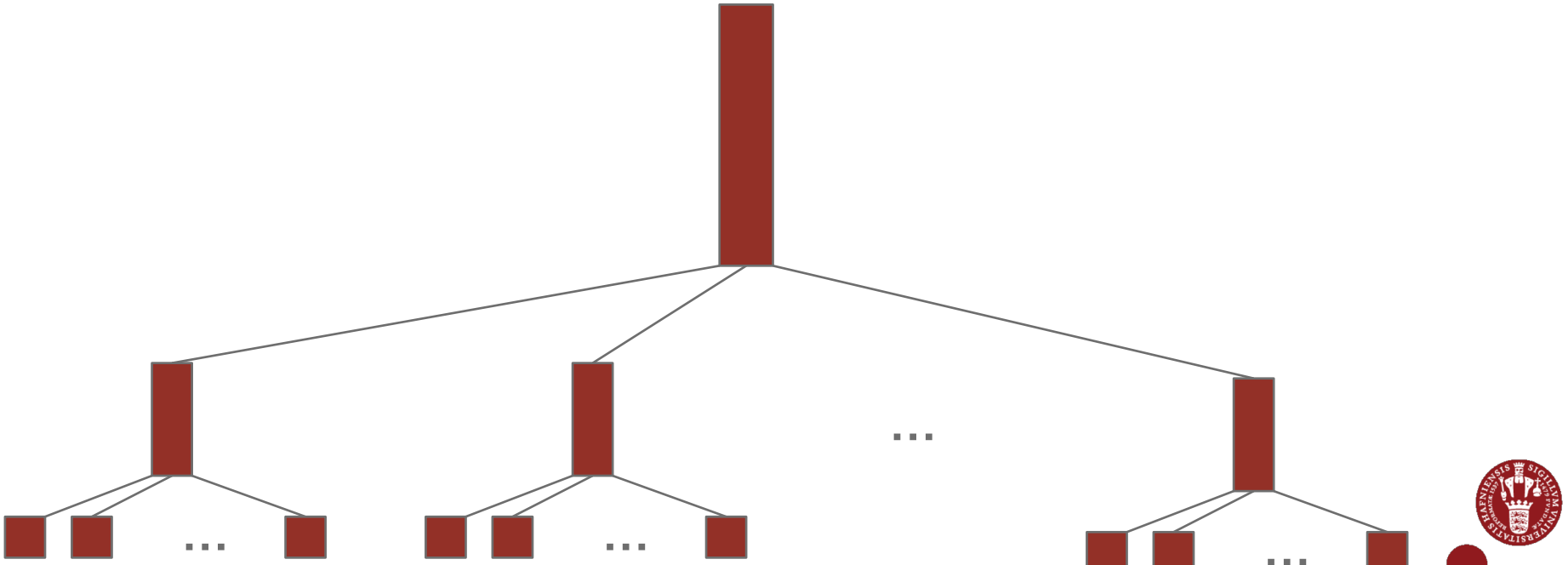
- Total cost = $2N\left(\lceil \log_2 N \rceil + 1\right)$

Questions so far?



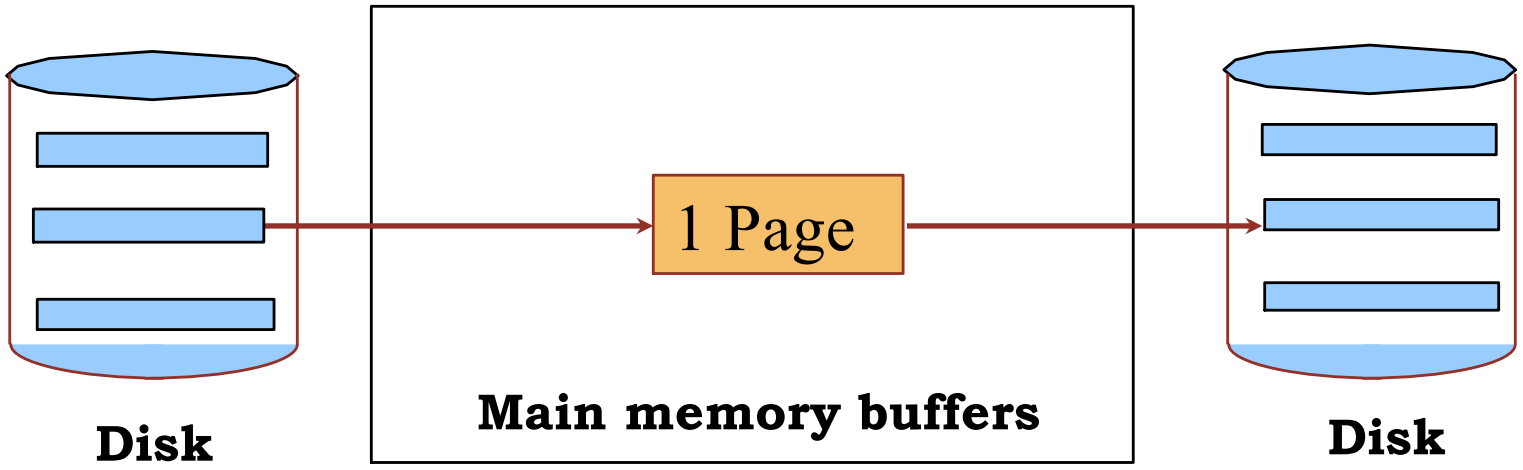
Can we do better?

- The cost depends on the #passes
- #passes depends on
  - fan-in during the merge phase
  - the number of runs produced by phase 1



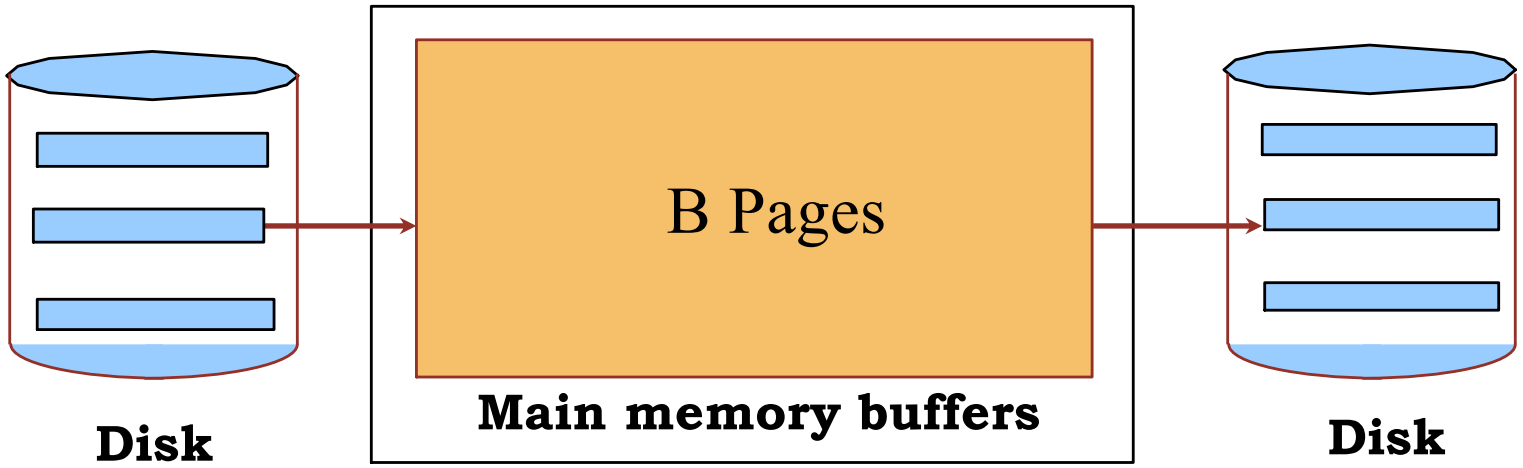
2-Way External Merge Sort

- Phase 1: Read a page at a time, sort it, write it
  - Only one buffer page used
- **How can this be modified if B buffer pages are available?**



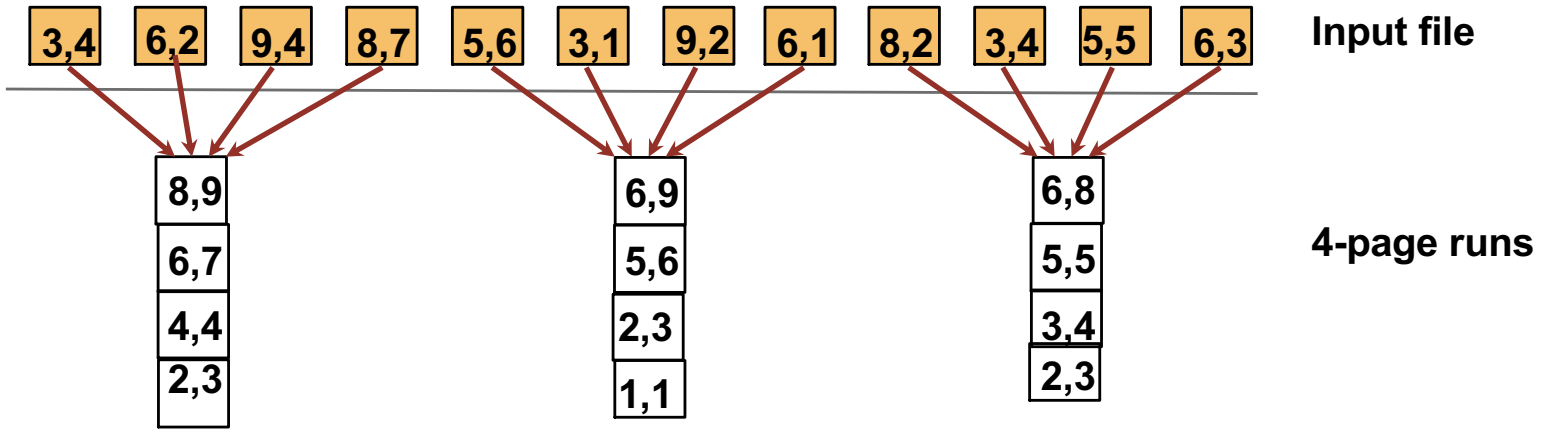
Multi-Way External Merge Sort

- Phase 1: Read B pages at a time, sort B pages in main memory, and write out B pages
- **Length of each run = B pages**
- Assuming N input pages, number of runs = N/B
- Cost of Phase 1 = $2N$



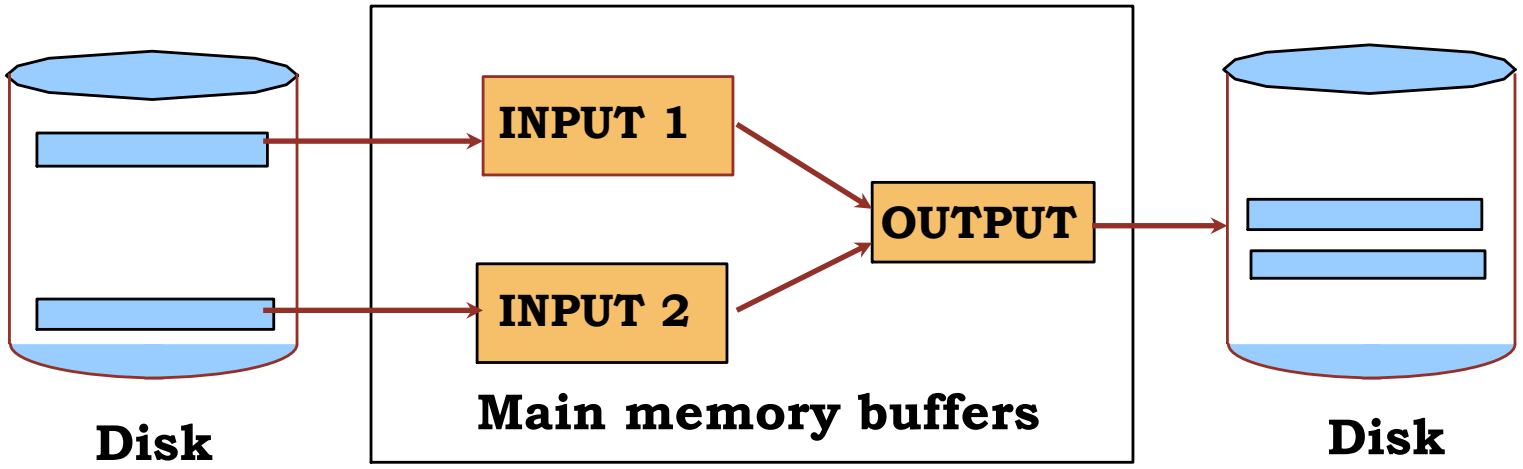
Multi-Way External Merge Sort

- # buffer pages B = 4



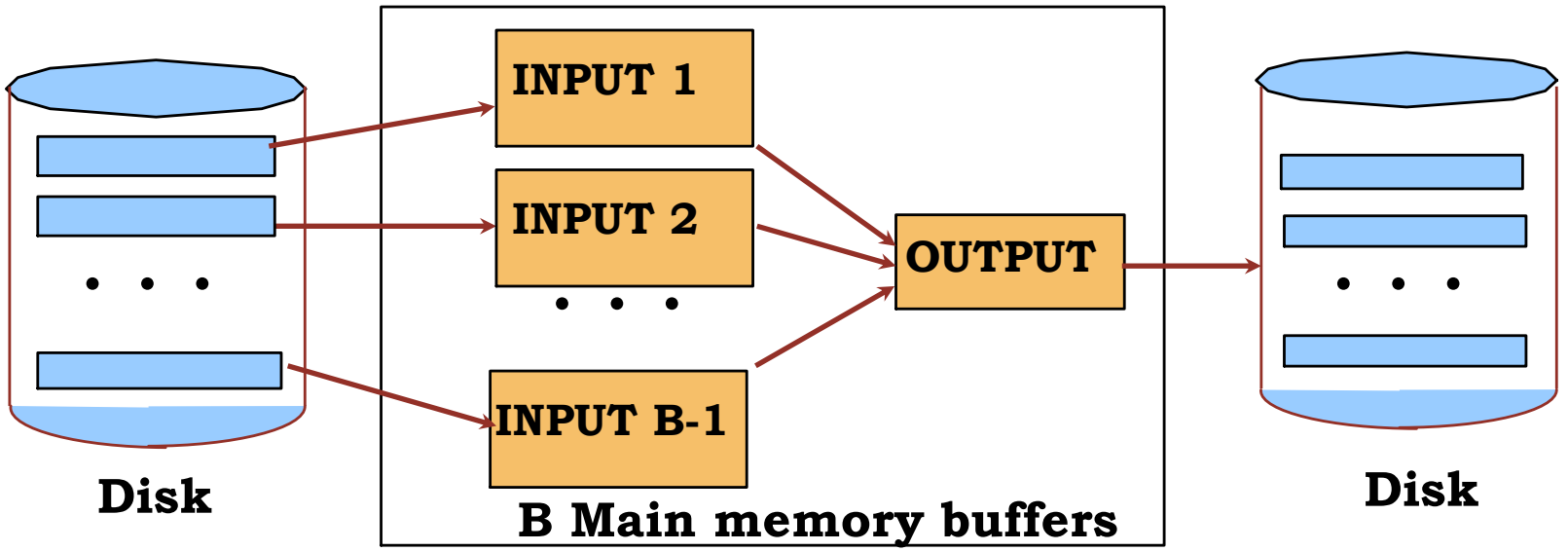
2-Way External Merge Sort

- Phase 2: Make multiple passes to merge runs
  - Pass 1: Merge two runs of length 1 (page)
  - Pass 2: Merge two runs of length 2 (pages)
  - … until 1 run of length N
  - Three buffer pages used
- **How can this be modified if B buffer pages available?**



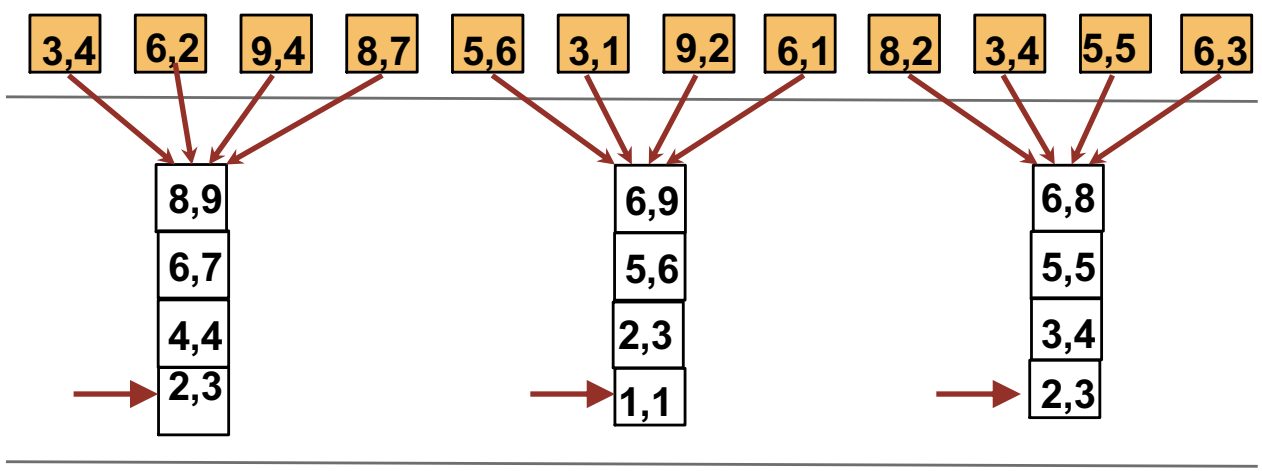
Multi-Way External Merge Sort

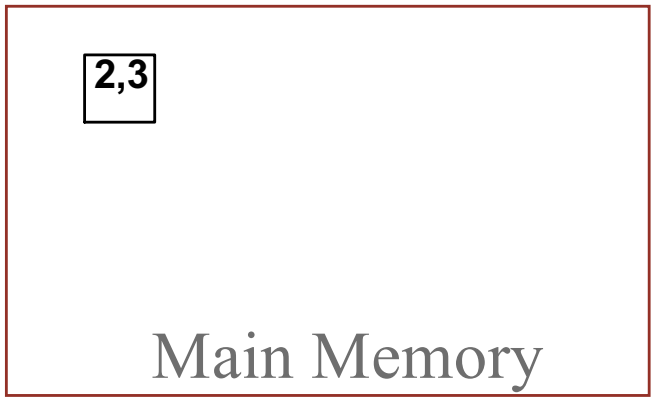
- Phase 2: Make multiple passes to merge runs
  - Pass 1: Produce runs of length B(B-1) pages
  - Pass 2: Produce runs of length $B(B-1)^2$ pages
  - …
  - Pass P: Produce runs of length $B(B-1)^P$ pages



Multi-Way External Merge Sort: Phase 2

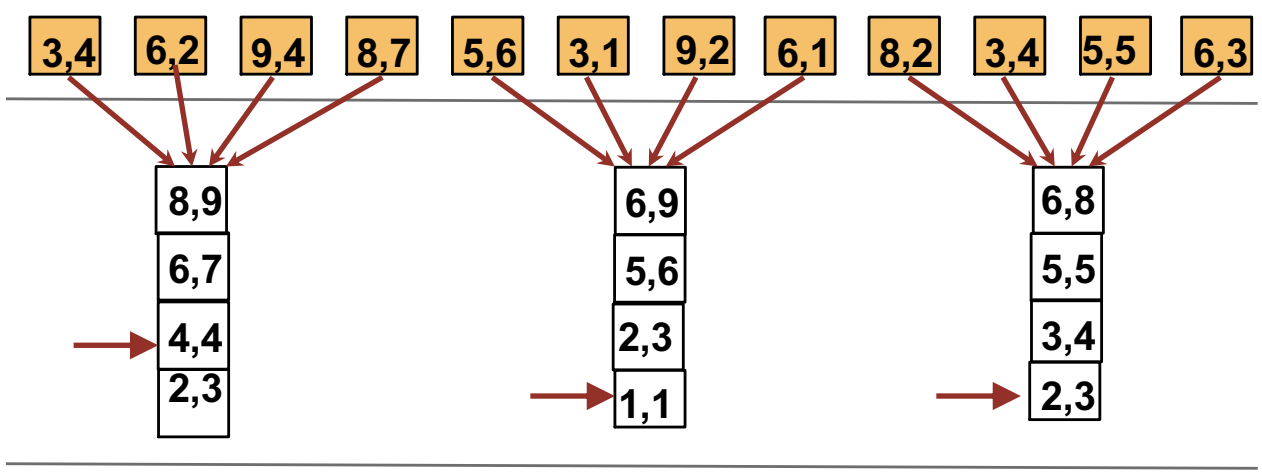
- # buffer pages B = 4

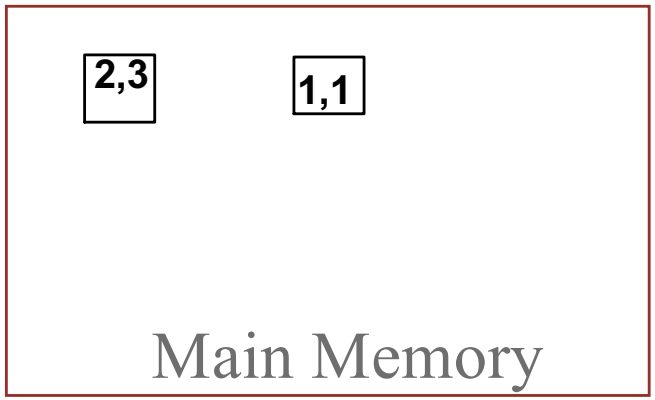




Phase 2
Pass 1

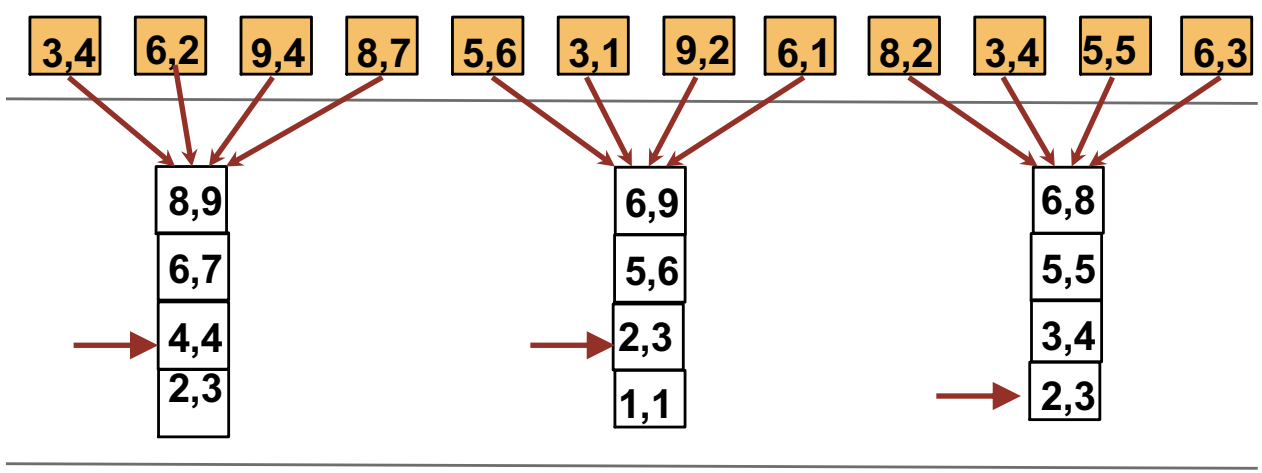
Multi-Way External Merge Sort: Phase 2

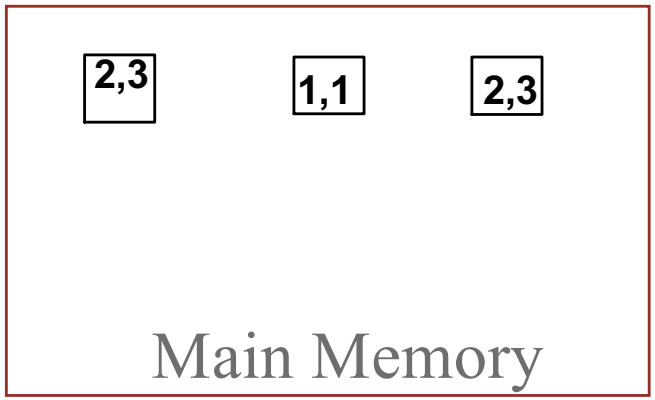




Phase 2
Pass 1

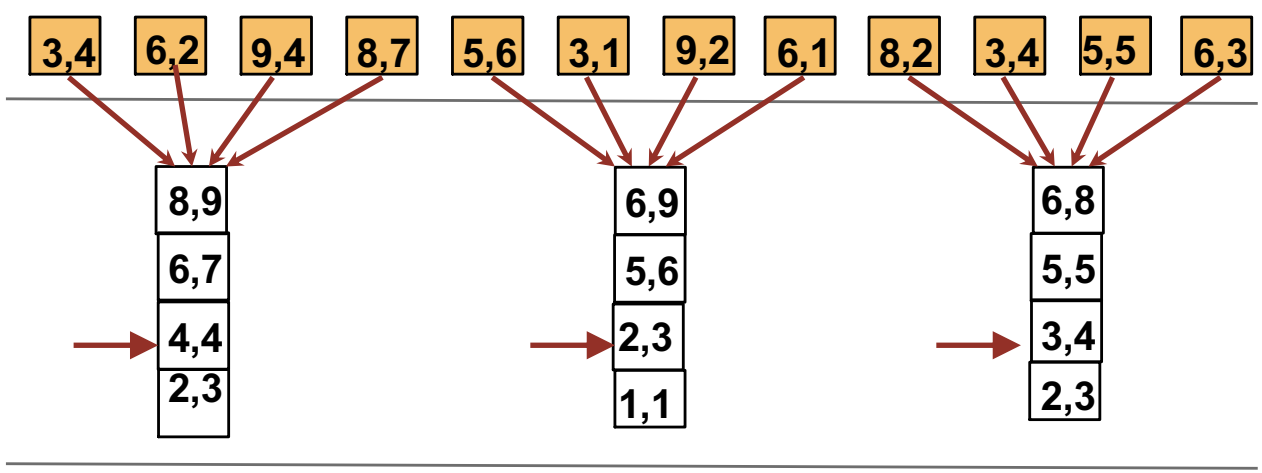
Multi-Way External Merge Sort: Phase 2

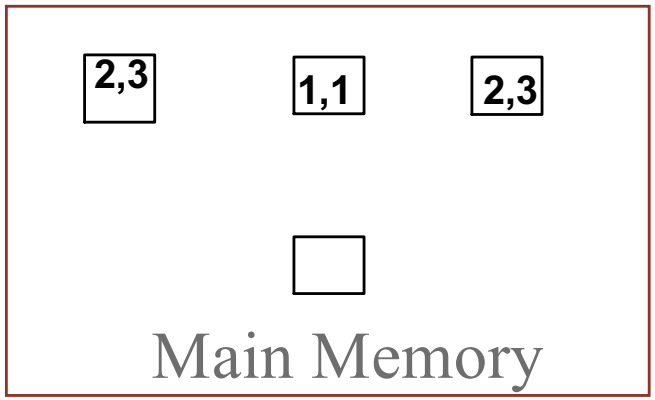




Phase 2
Pass 1

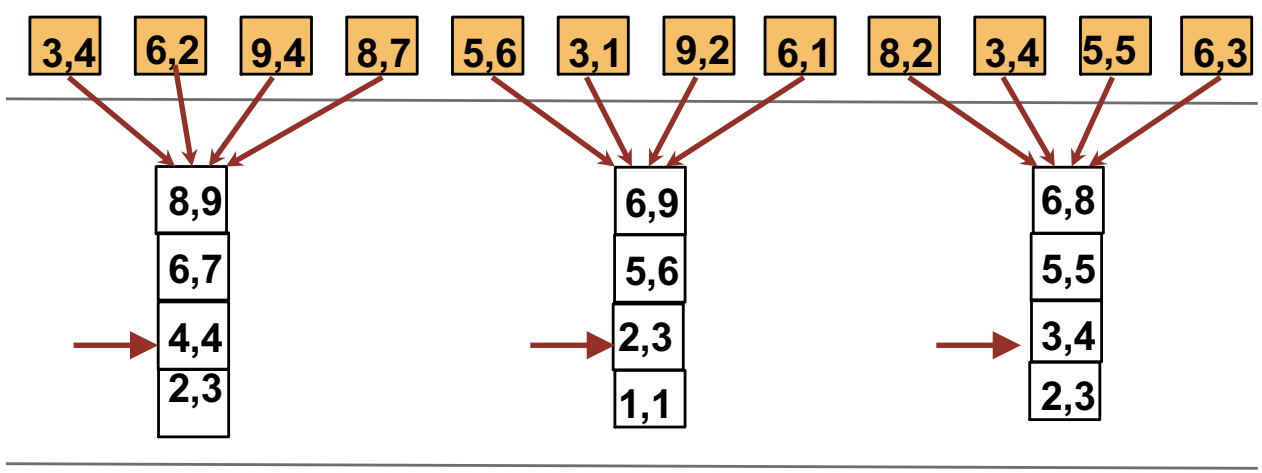
Multi-Way External Merge Sort: Phase 2

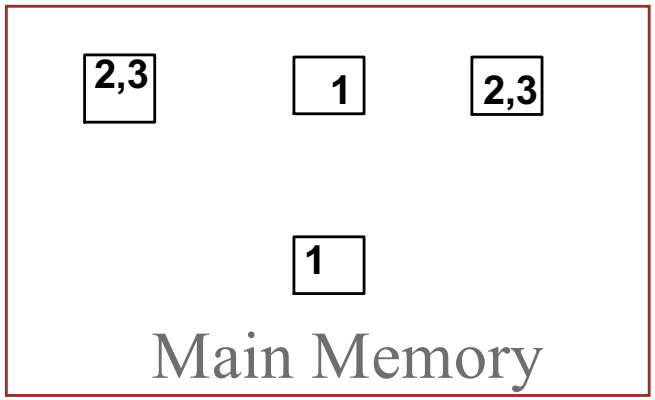




Phase 2
Pass 1

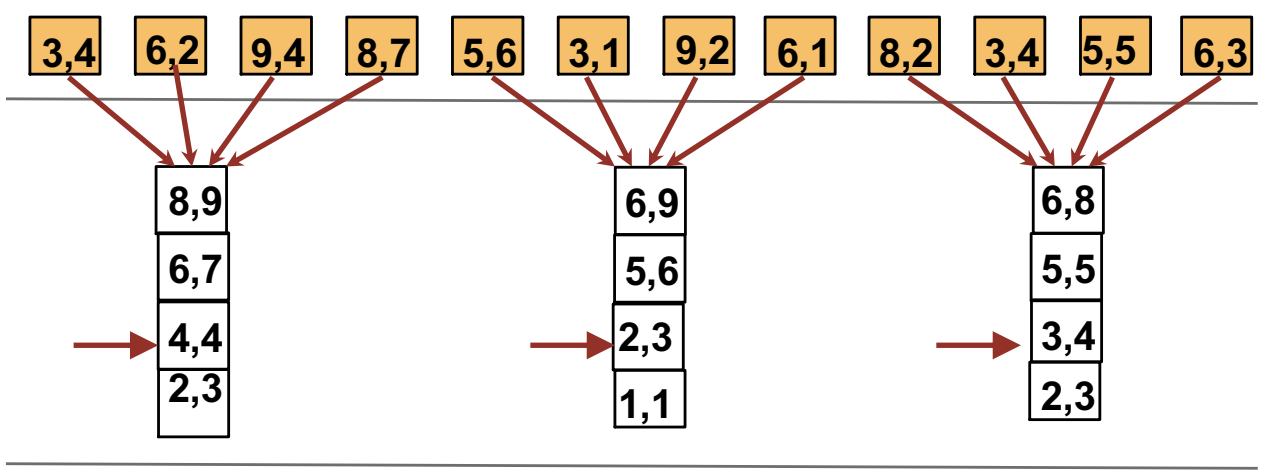
Multi-Way External Merge Sort: Phase 2

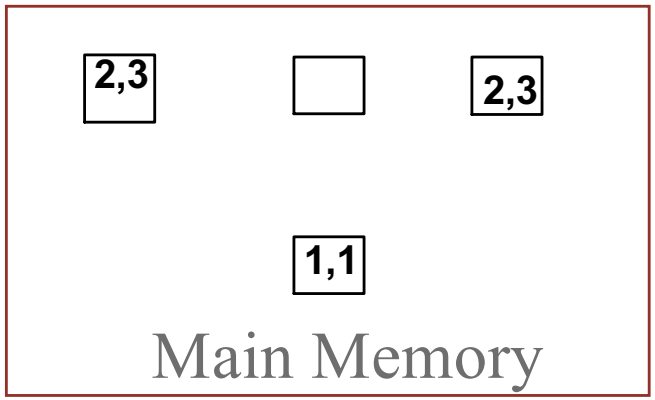




Phase 2
Pass 1

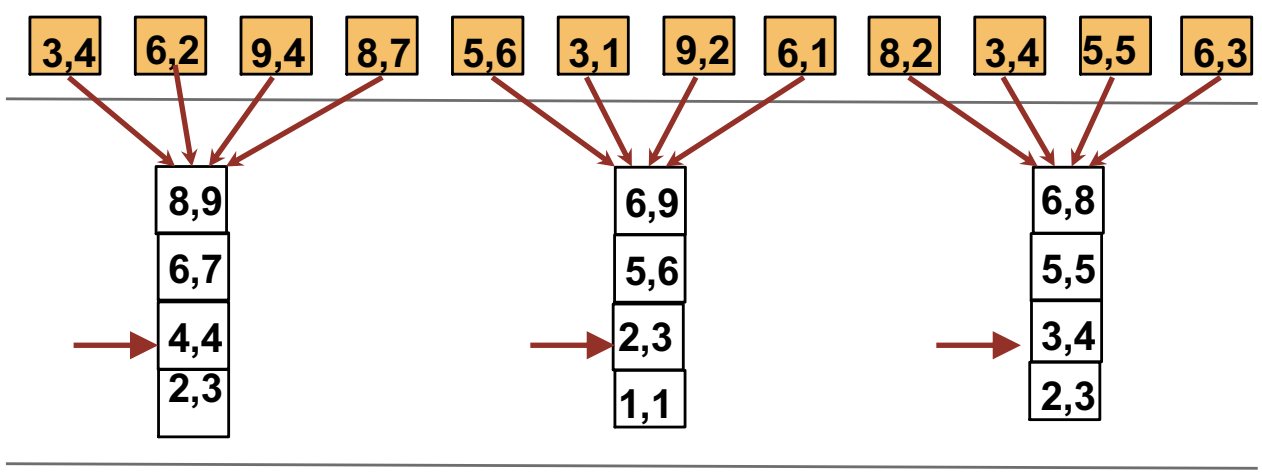
Multi-Way External Merge Sort: Phase 2

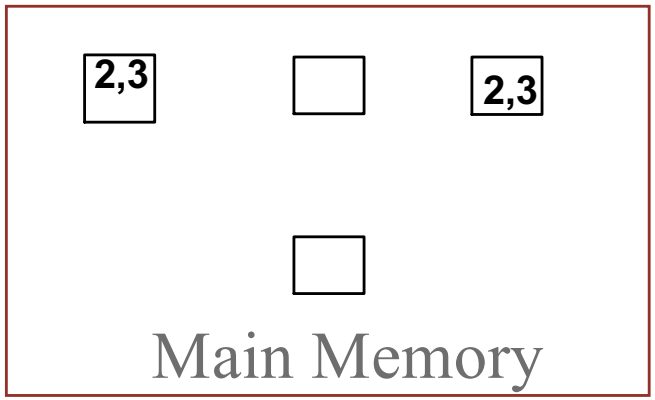




Phase 2
Pass 1

Multi-Way External Merge Sort: Phase 2

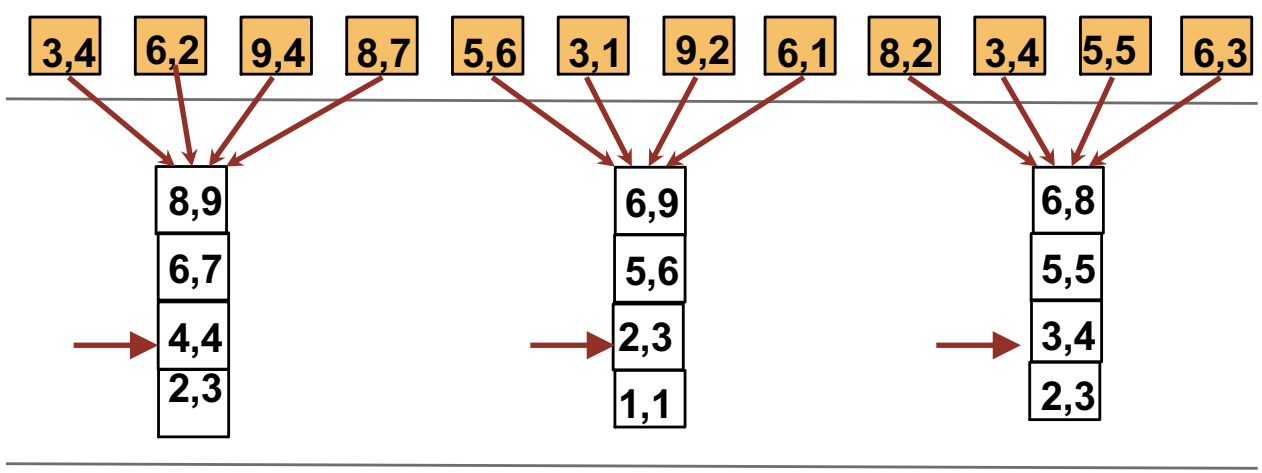


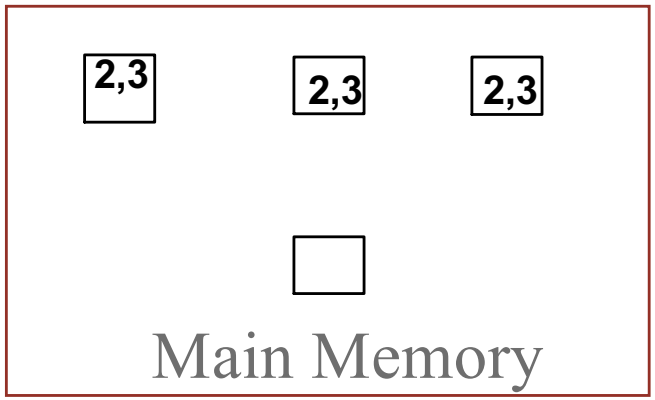


1,1

Phase 2
Pass 1

Multi-Way External Merge Sort: Phase 2

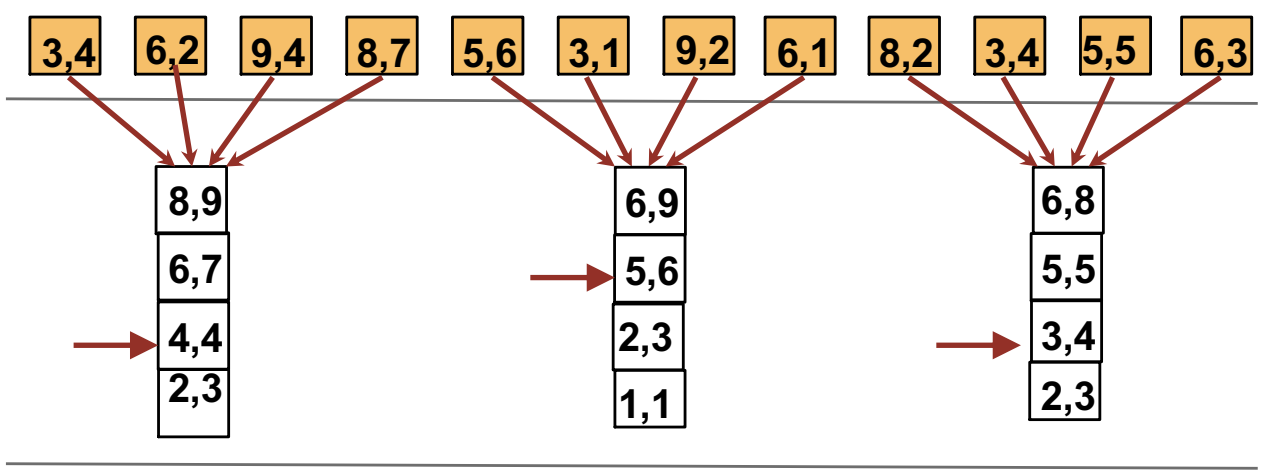


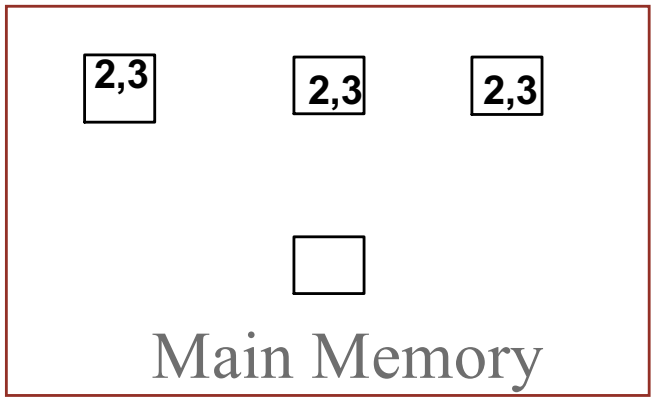


1,1

Phase 2
Pass 1

Multi-Way External Merge Sort: Phase 2

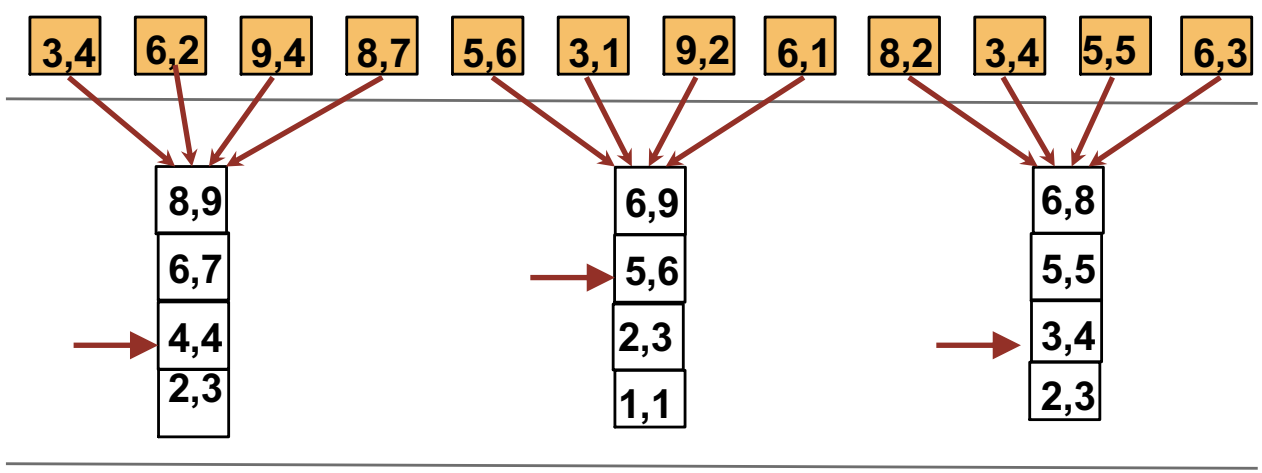


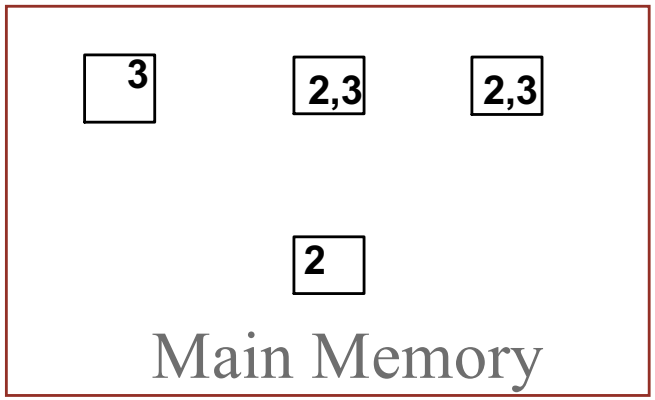


1,1

Phase 2
Pass 1

Multi-Way External Merge Sort: Phase 2

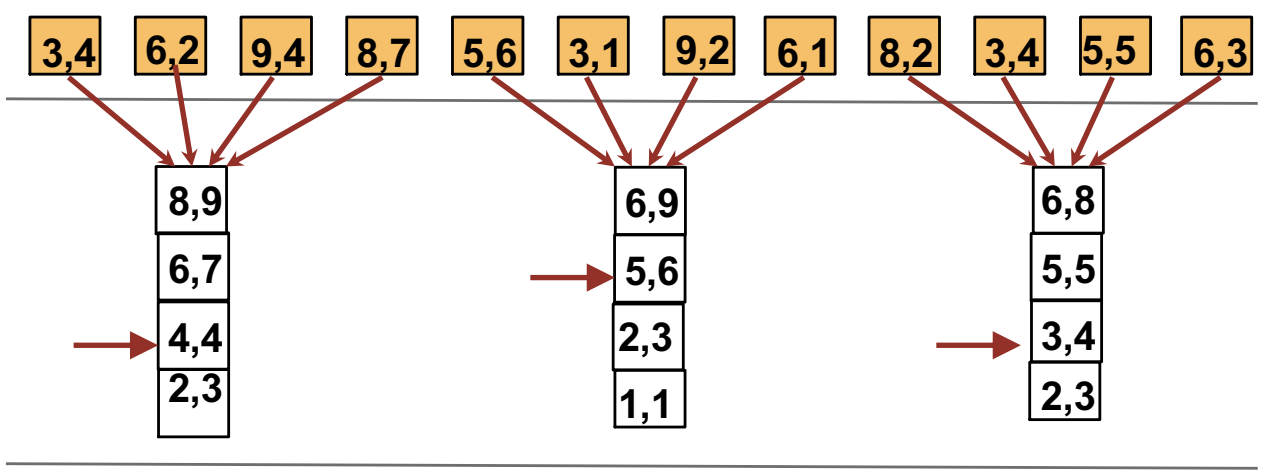


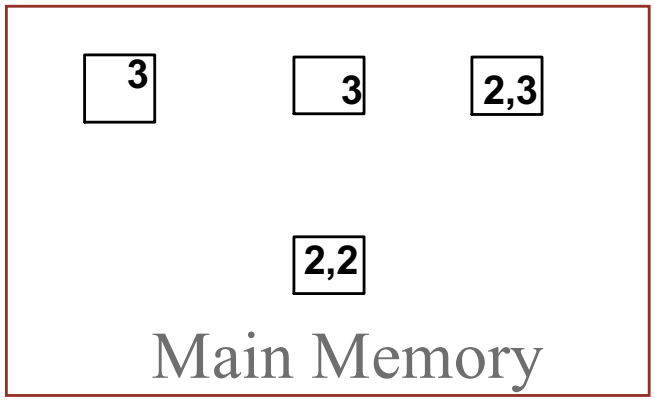


1,1

Phase 2
Pass 1

Multi-Way External Merge Sort: Phase 2

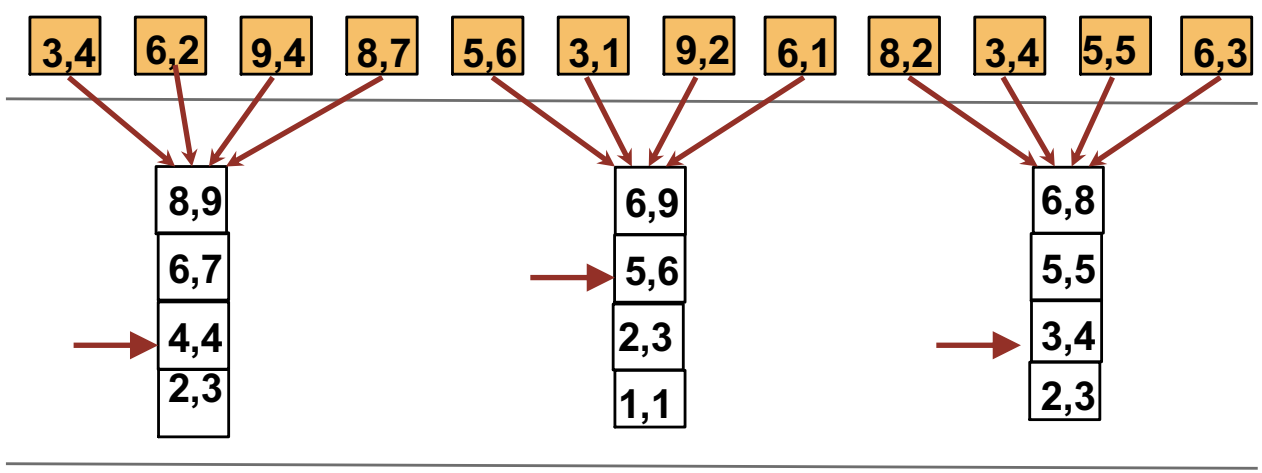


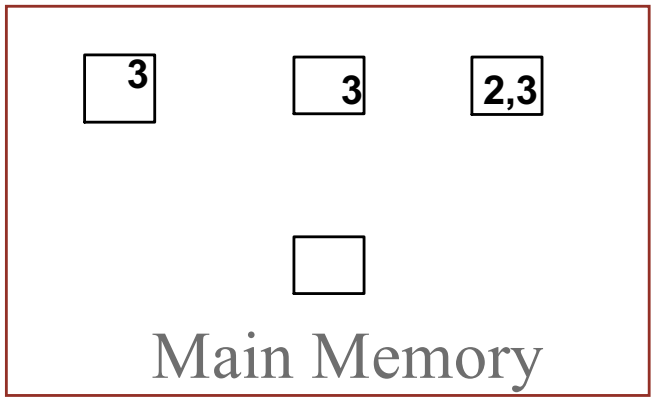


1,1

Phase 2
Pass 1

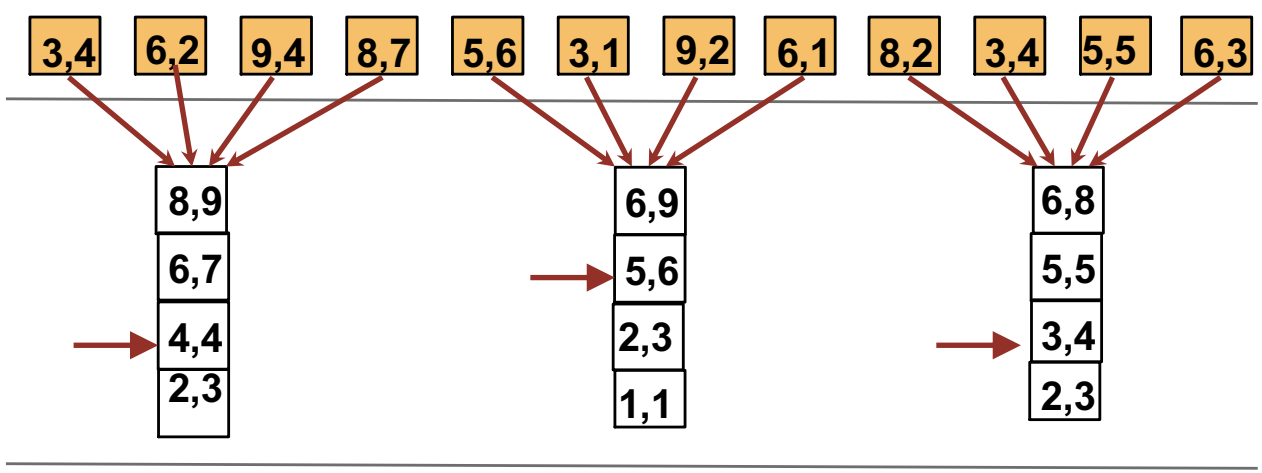
Multi-Way External Merge Sort: Phase 2

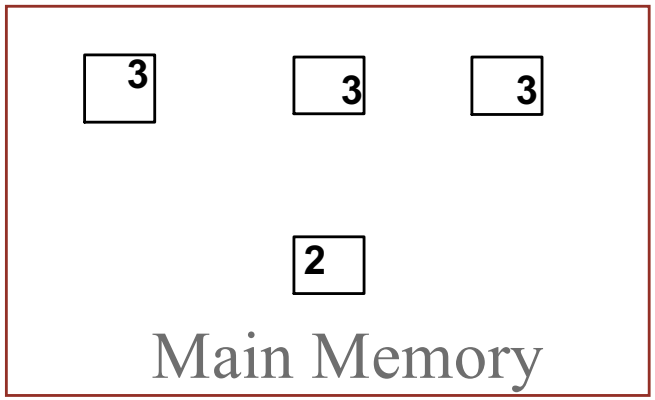




Phase 2
Pass 1

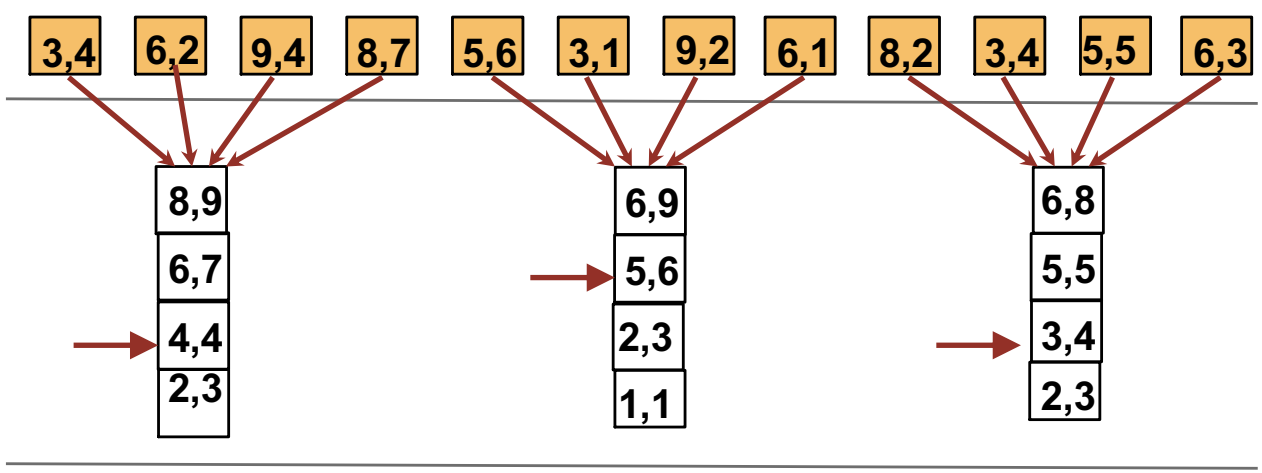
Multi-Way External Merge Sort: Phase 2

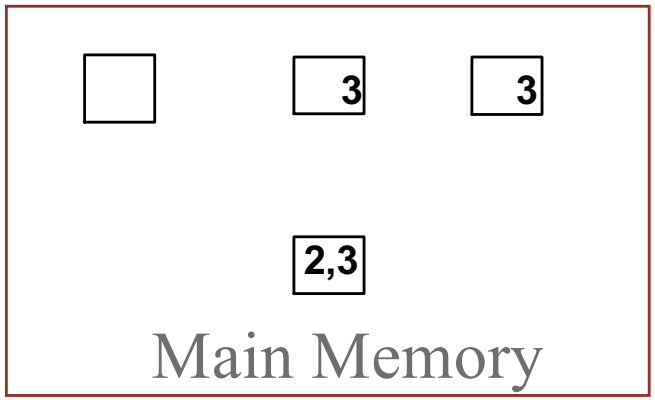




Phase 2
Pass 1

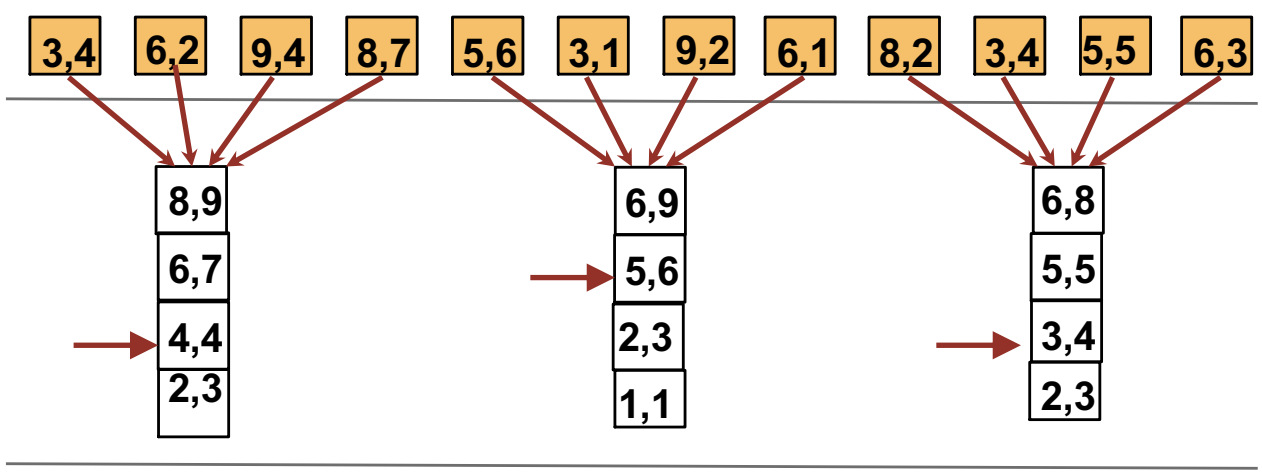
Multi-Way External Merge Sort: Phase 2

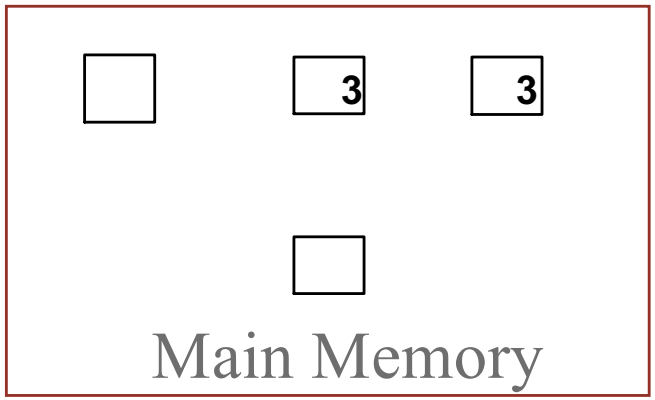




Phase 2
Pass 1

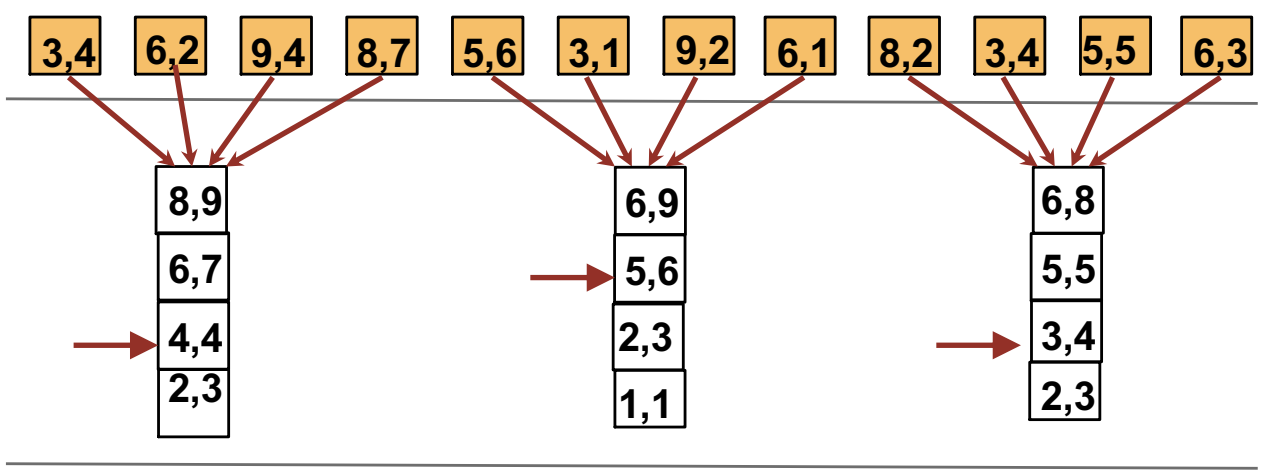
Multi-Way External Merge Sort: Phase 2

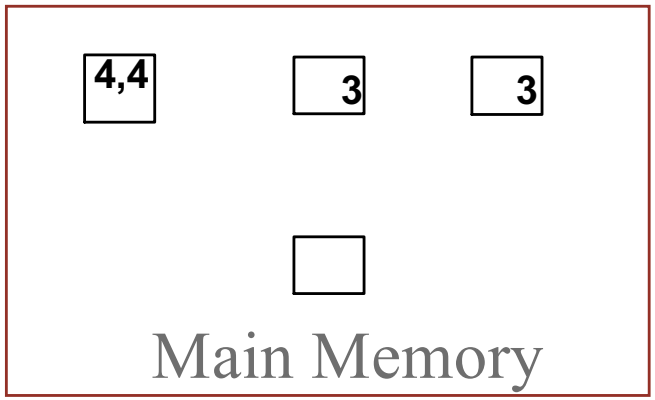




Phase 2
Pass 1

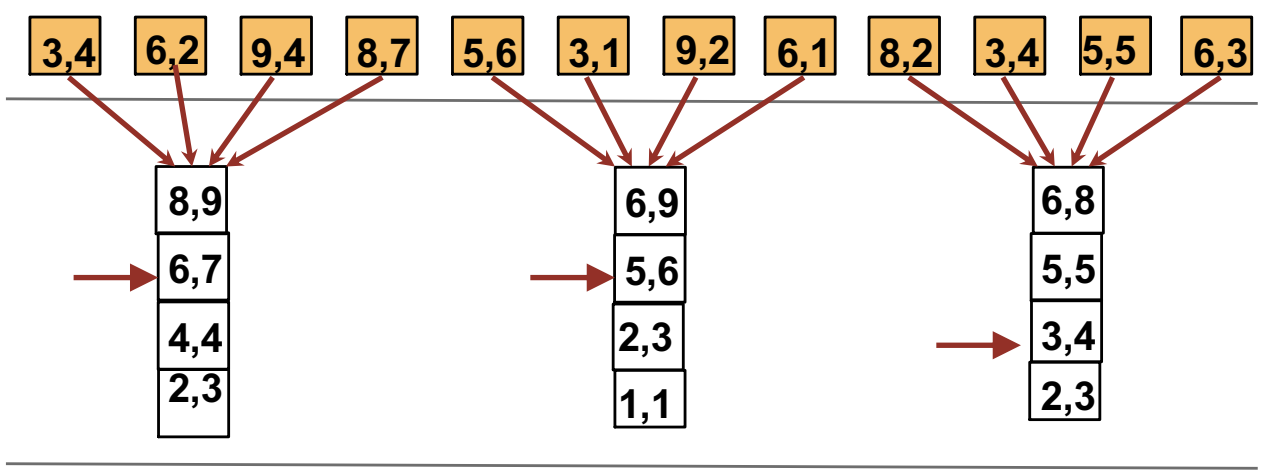
Multi-Way External Merge Sort: Phase 2

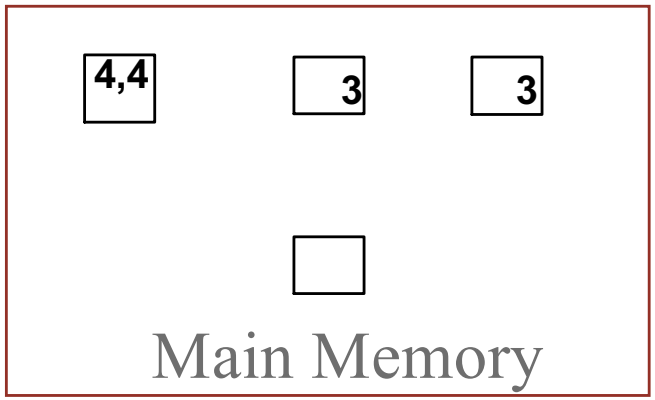




Phase 2
Pass 1

Multi-Way External Merge Sort: Phase 2





Phase 2
Pass 1

Multi-Way External Merge Sort: Analysis

Total I/O cost for sorting file with N pages

- Cost of Phase 1 = $2N$

If # passes in Phase 2 is P then: $B(B-1)^P = N$

Thus

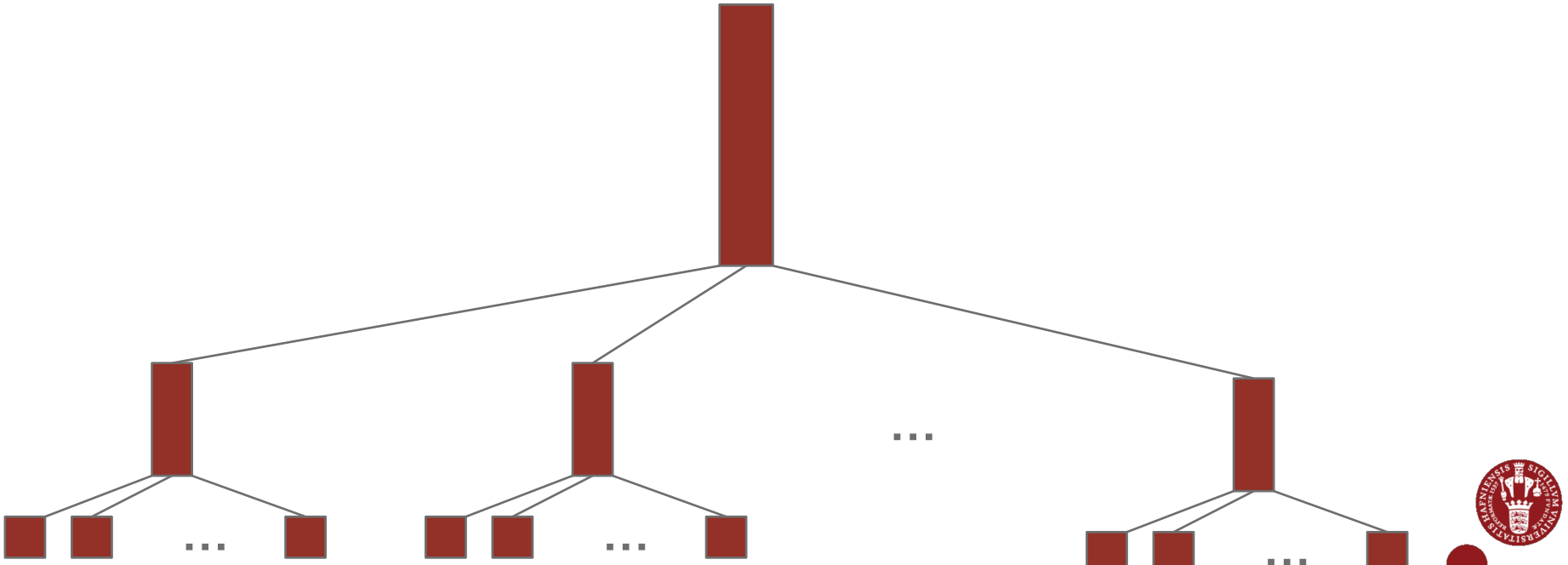
- P = $\lceil \log_{B-1} \lceil N/B \rceil \rceil$
- Cost of each pass = $2N$
- Cost of Phase 2 = $2N \times \lceil \log_{B-1} \lceil N/B \rceil \rceil$
- Total cost = $2N(\lceil \log_{B-1} \lceil N/B \rceil \rceil + 1)$
- Compared to $2N(\lceil \log_2 N \rceil + 1)$

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

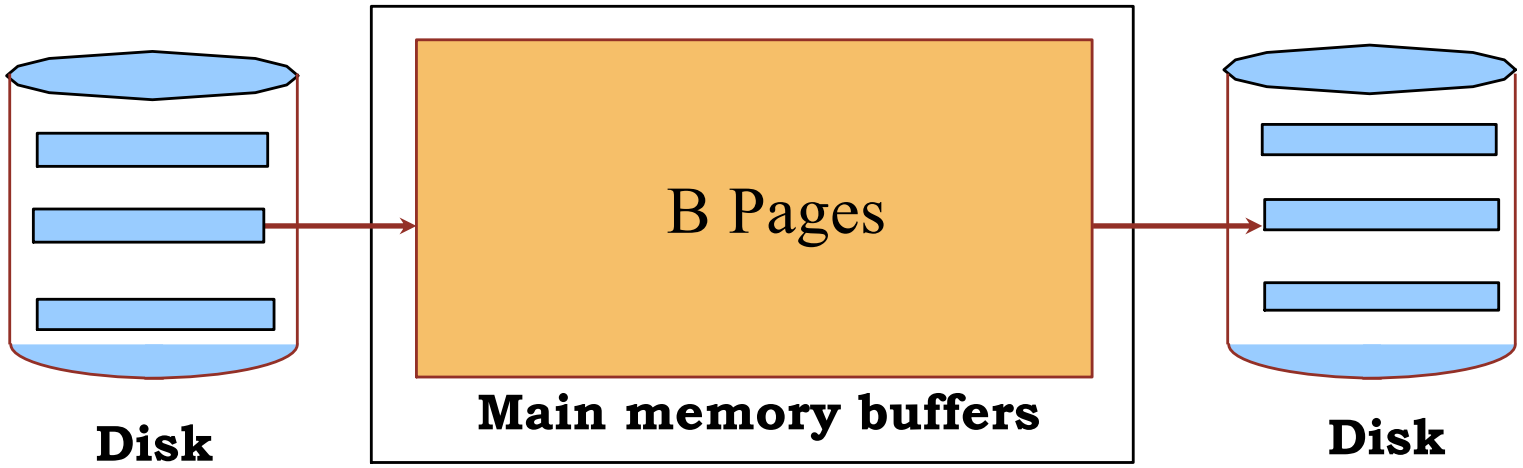
Can we do even better without purchasing more memory?

- The cost depends on the #passes
- #passes depends on
  - fan-in during the merge phase
  - **the number of runs produced by phase 1**

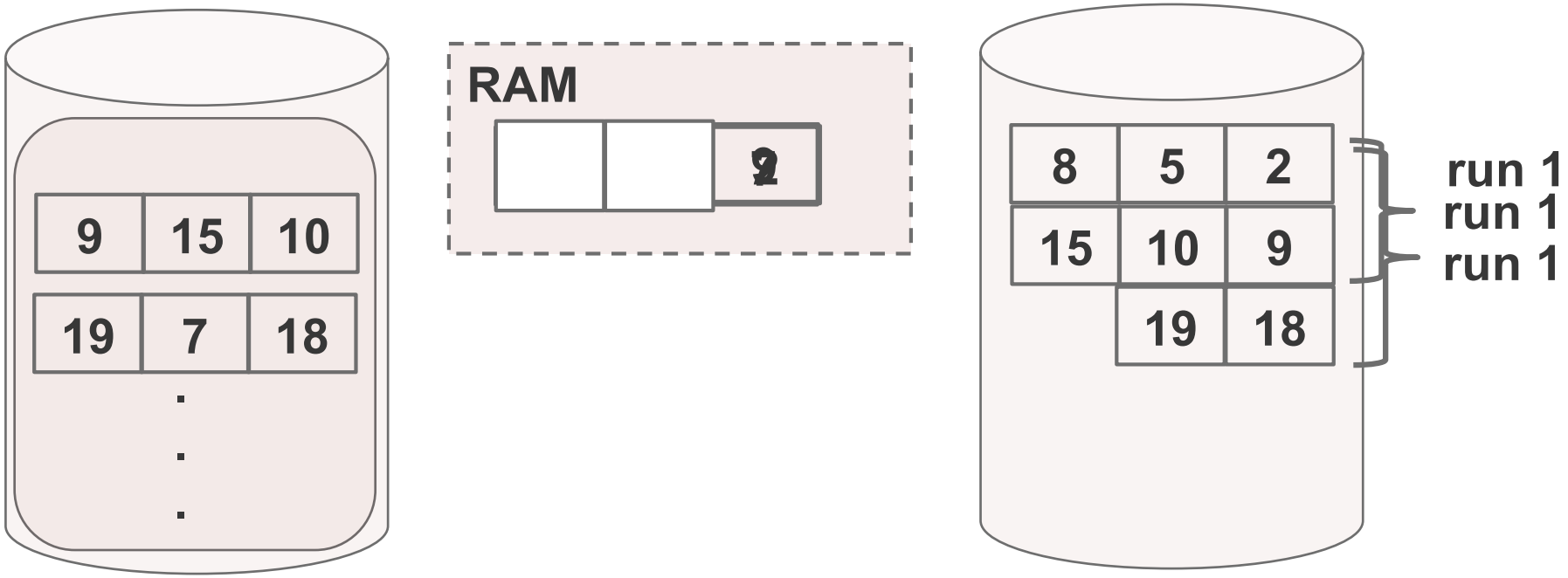


Multi-Way External Merge Sort

- Phase 1: Read B pages at a time, sort B pages in main memory, and write out B pages
- **Length of each run = B pages**
- Assuming N input pages, number of runs = N/B

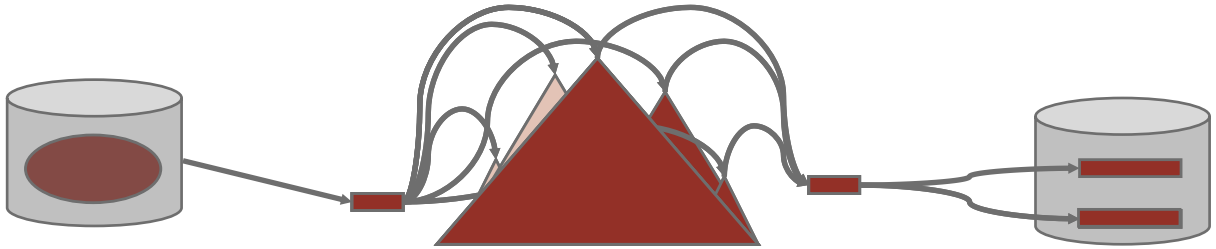


Can we produce runs longer than the size of RAM?



49

Tournament sort



- "tournament sort" (a.k.a. "heapsort", "replacement selection 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 outputting all the records in H1
  - If there is something left in H2, output them as a new run.

More on Heapsort

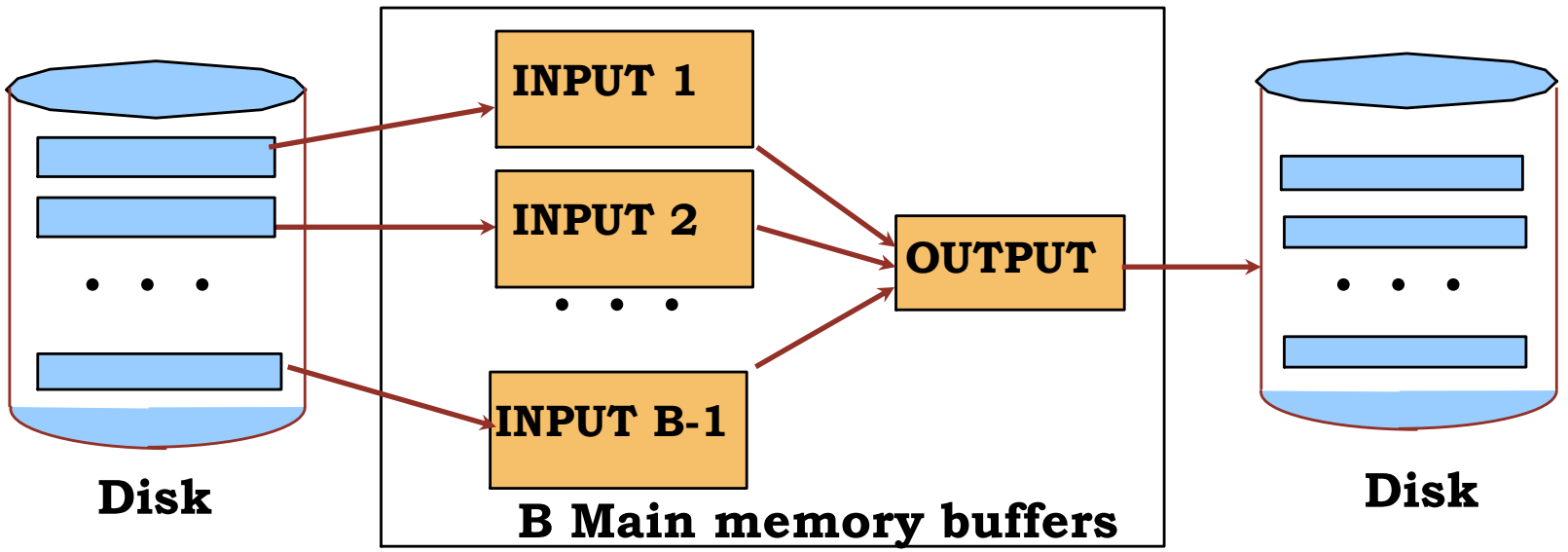
- Fact: average length of a run in heapsort is 2(B-2)
- Worst-Case:
  - min length of a run?
  - How does this arise?
- Best-Case:
  - max length of a run?
  - How does this arise?
- Quicksort is faster, but longer but fewer runs often mean fewer passes!

What is the average length of a run in heapsort?

B-2 pages are used for the heaps

$(B-2) + 1/2\,(B-2) + 1/4\,(B-2) + 1/8\,(B-2) + \ldots$
$\approx 2(B-2)$

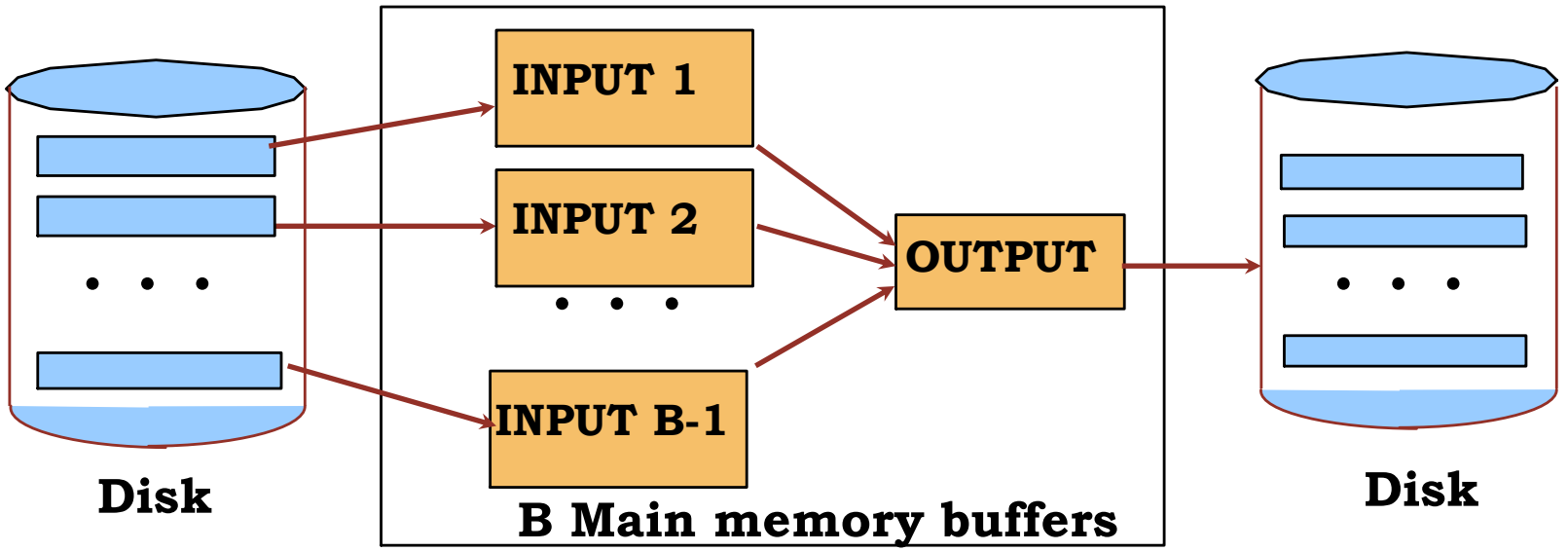
Questions so far?



External Merge Sort: Optimizations

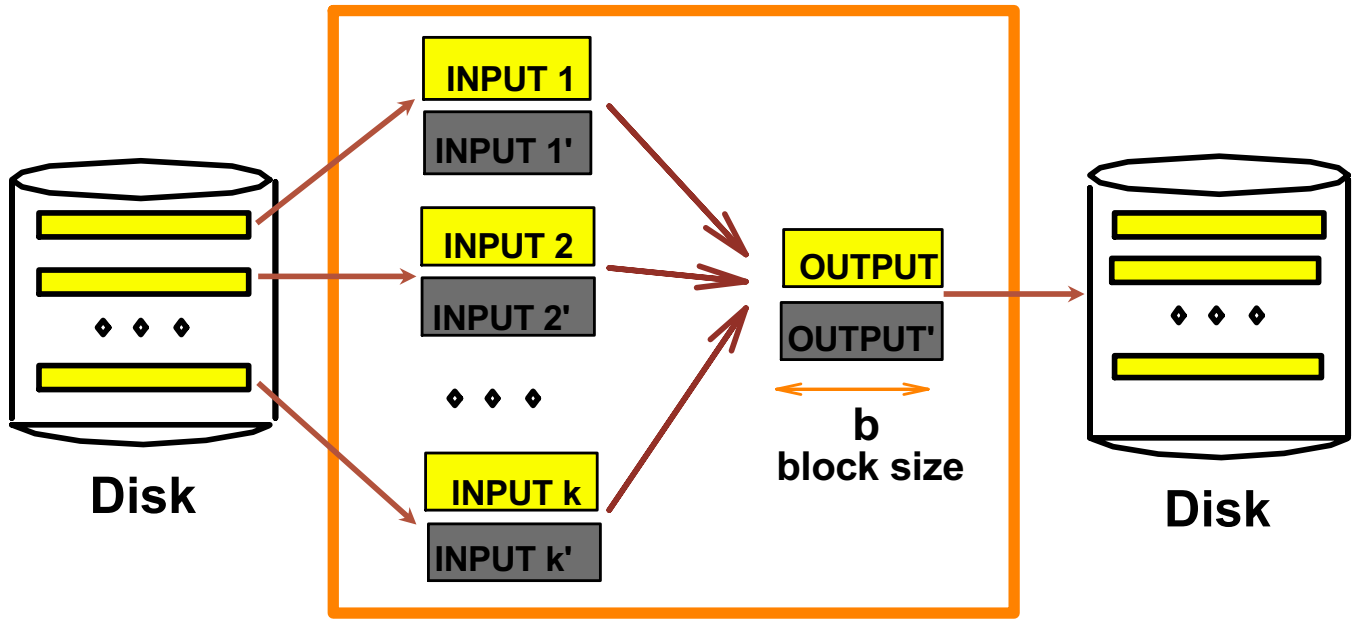
In Phase 2:

- Currently, do one page I/O at a time
- But can read/write a block of pages sequentially!
  - Make each buffer input/output a block of pages
  - Better read performance
- Possible negative impact?



External Merge Sort: Optimizations

- Double buffering: to reduce I/O wait time, *prefetch* into `shadow block'.
  - Again, potentially more passes; in practice, most files *still* sorted in 2-3 passes.



B main memory buffers, k-way merge

What have we learned so far?

- When data is much too big to fit in memory our "normal" best algorithms might not be the best.

- External sorting
  - Sorting with two (or more) disks
  - Merge sort

- Optimizations:
  - Utilize memory to the fullest
  - Use heap/replacement sort to reduce #runs
  - Read sequences of pages from disk
  - Keep disks "busy"

Using B+ Trees for Sorting

Alternative to external sorting

Scenario:

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

Idea:

- Can retrieve records in order by traversing leaf pages.

Not always a good idea?

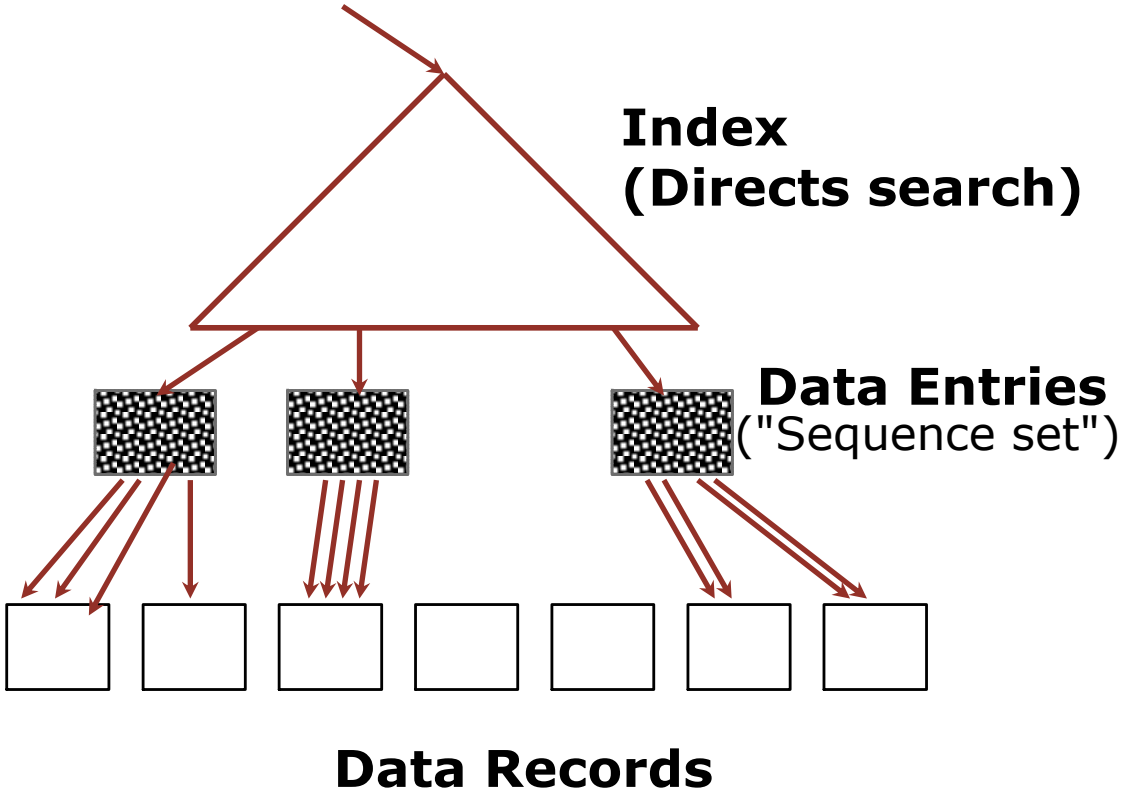
Cases to consider:

- B+ tree is clustered — Good idea!
- B+ tree is not clustered — Could be a very bad idea!

Using B+ Trees for Sorting

- **Cost:** root to the left-most leaf, then retrieve all leaf pages (direct index)

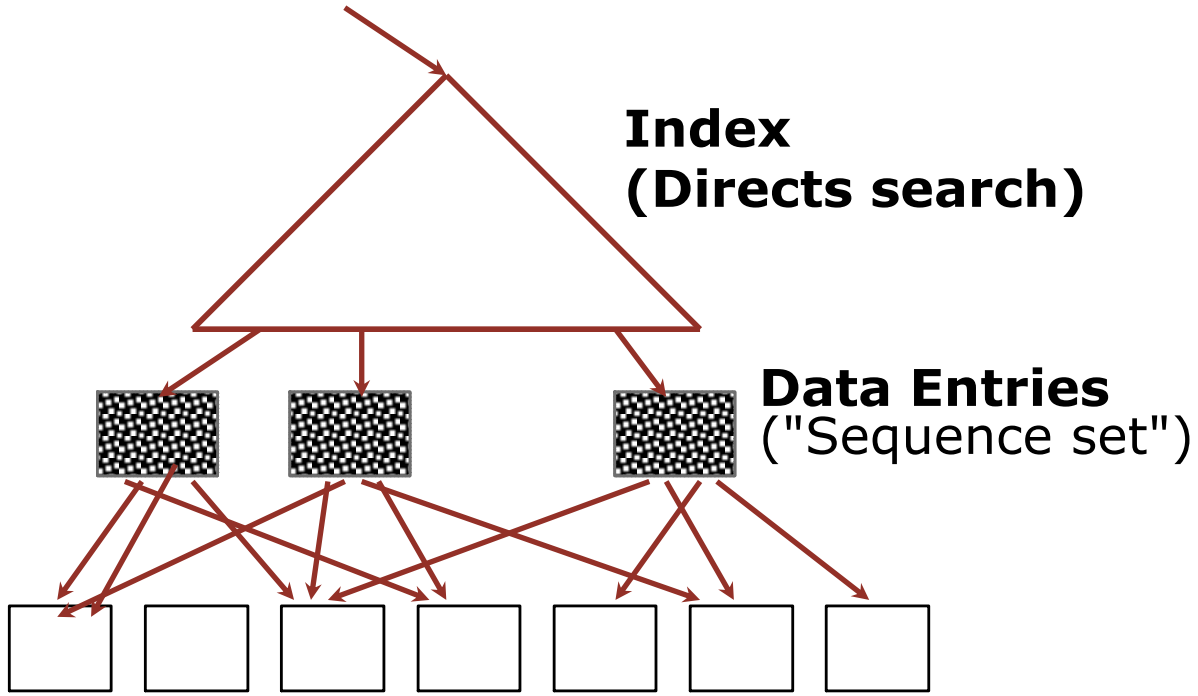
- If indirect index is used? Additional cost of retrieving data records: each page fetched just once.



☛ *Always better than external sorting!*

Using B+ Trees for Sorting

- Indirect, unclustered index: each data entry contains rid of a data record.
- In general, one I/O per data record!



Relational Operator Implementation

Implementing relational operators is challenging because:

- Relational queries are declarative
  - There is no efficient predefined strategy
- The data sets are typically very large

Relational Operators

How do we implement operators for relational operations in external memory?

- Select
- Project
- Join
- Set operations (union, intersect, except)
- Aggregation

Next time! ☺

What should we learn today?

- Reason about cost models and algorithmic design decisions when processing BIG data, such that I/Os and not only RAM operations are necessary
- Explain the need for an external sorting algorithm, compared to simply applying an internal memory algorithm over a two-level memory abstraction
- Explain and apply external sorting based on 2-way and multi-way sort-merge approaches
- Discuss potential optimizations to external sorting, including shadow buffering, use of indexes, and replacement selection