Database System
(SW5)

9. Query Processing and Optimization

Tiantian Liu
Department of Computer Science
Aalborg University
Fall 2025

Motivation and Learning Goals

AALBORG UNIVERSITY

- Motivation
  - Understanding the basics of query processing and query optimization is the foundation of database tuning
- Learning Goals
  - Understand the basic join algorithms
  - Understand the basics of heuristic (logical) query optimization
  - Understand the basics of physical query optimization

Agenda

AALBORG UNIVERSITY

- Basic Steps in Query Processing
- Sorting
- Join Strategies
- Cost-based Query Optimization
- Logical Query Optimization

Evaluation of A SQL Statement

- The clauses are specified in the following order.
  - SELECT column(s)
  - FROM table list
  - WHERE condition
  - GROUP BY grouping column(s)
  - HAVING group condition

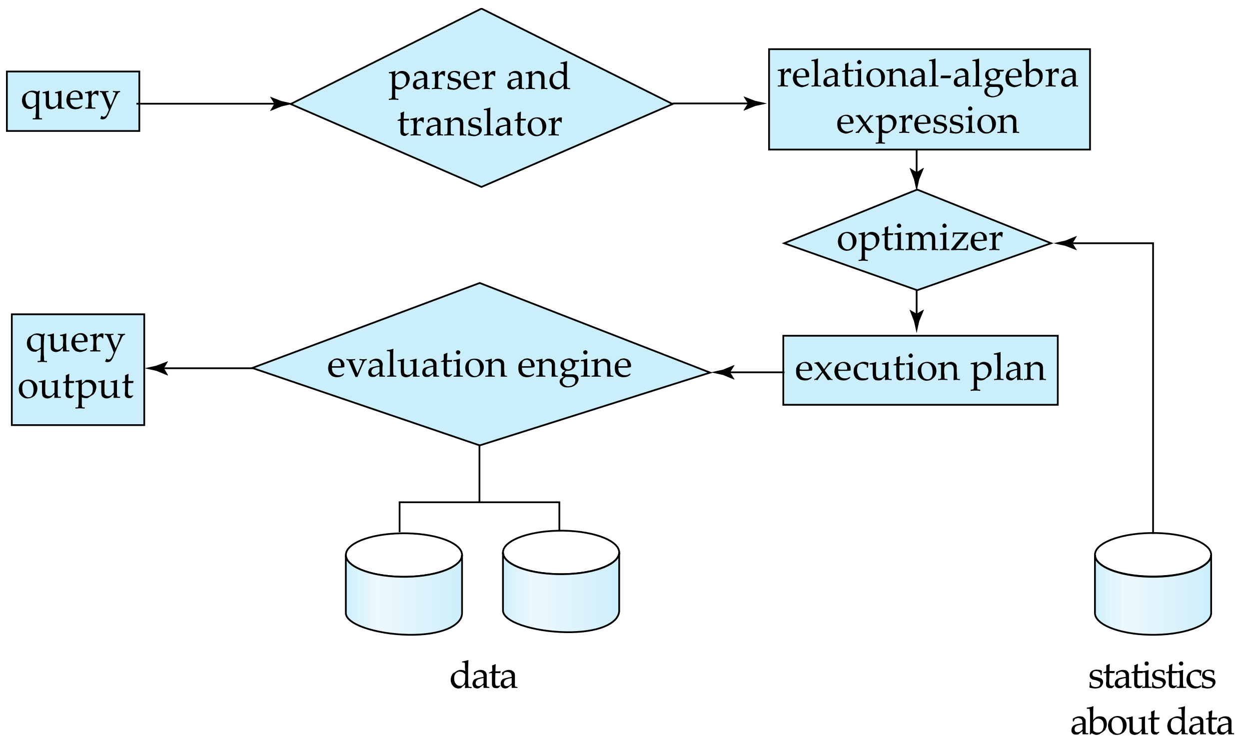
Evaluation of A SQL Statement

- But the query is evaluated in a different order
  - Cartesian product of tables in the FROM clause
  - Predicates in the WHERE clause
  - GROUP BY clause
  - Predicate in the HAVING clause (to eliminate groups)
  - Projection on columns enumerated in the SELECT clause

```
SELECT column(s)
FROM table list
WHERE condition
GROUP BY grouping column(s)
HAVING group condition
```

Basic Steps in Query Processing

- Parsing and translation
- Optimization
- Evaluation



Agenda

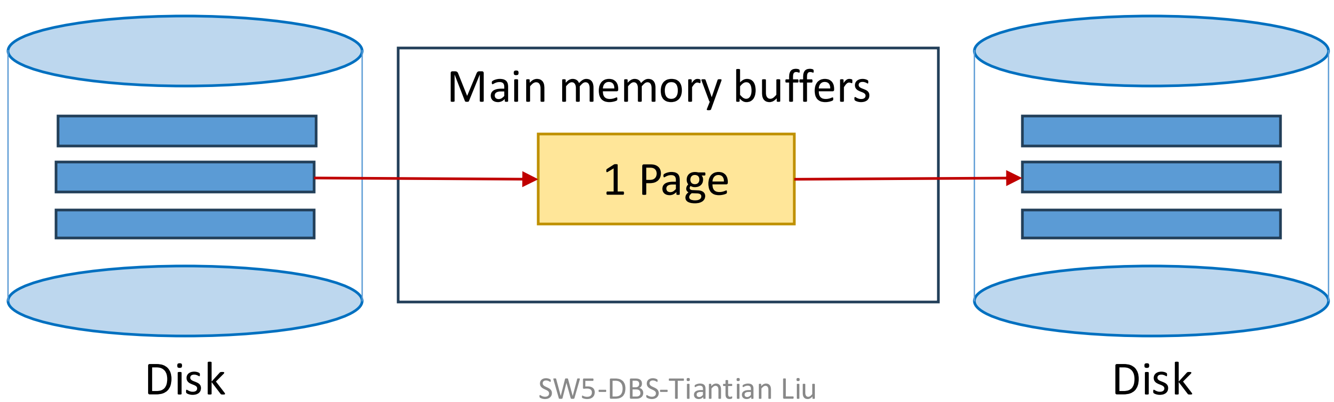
- Basic Steps in Query Processing
- Sorting
- Join Strategies
- Cost-based Query Optimization
- Logical Query Optimization

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

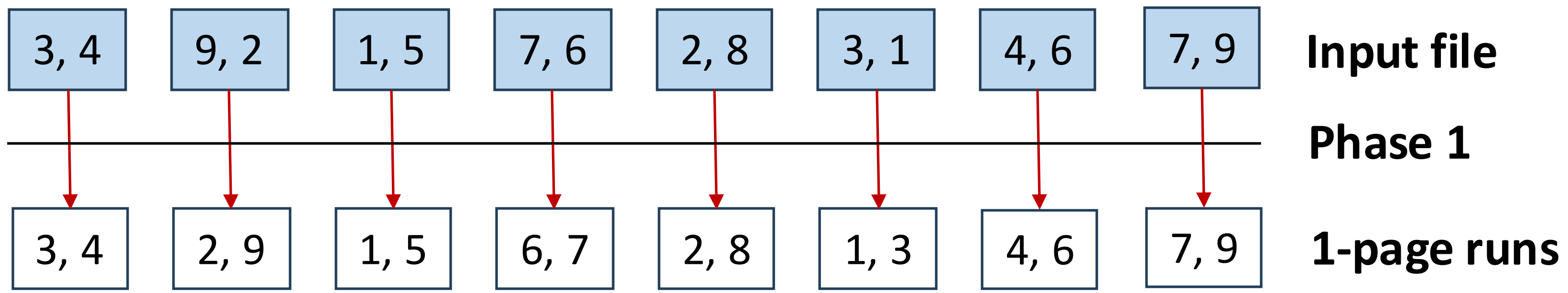
2-Way External Merge Sort: Phase 1

- 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



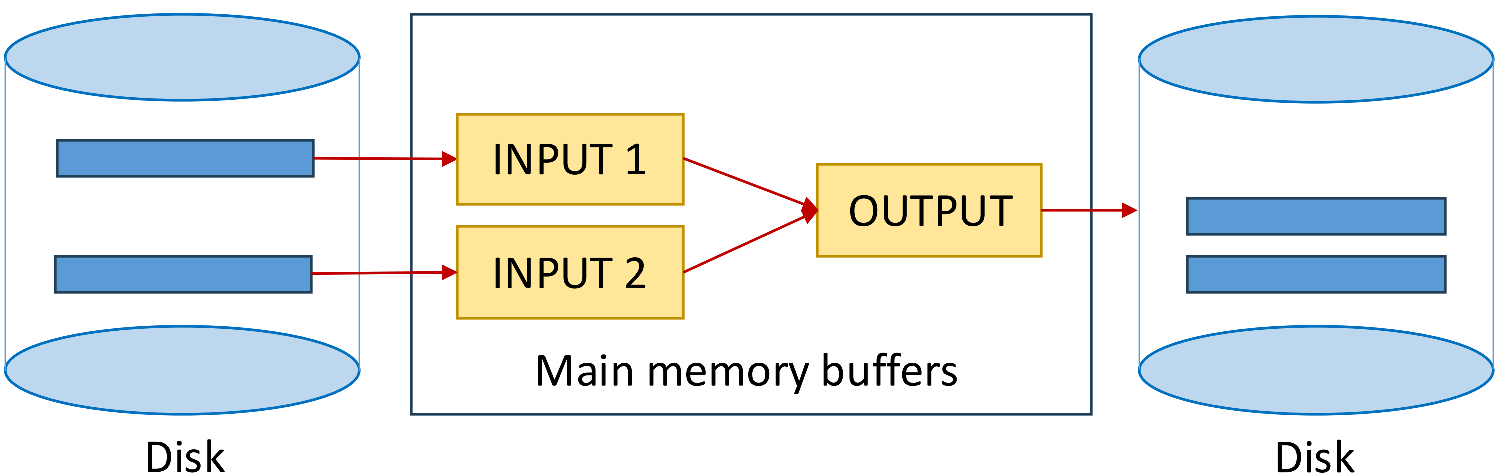
Phase 1: Example

- Assume input file with N data pages of size M
- What is the cost of Phase 1?
  - In terms of # I/O? 2N

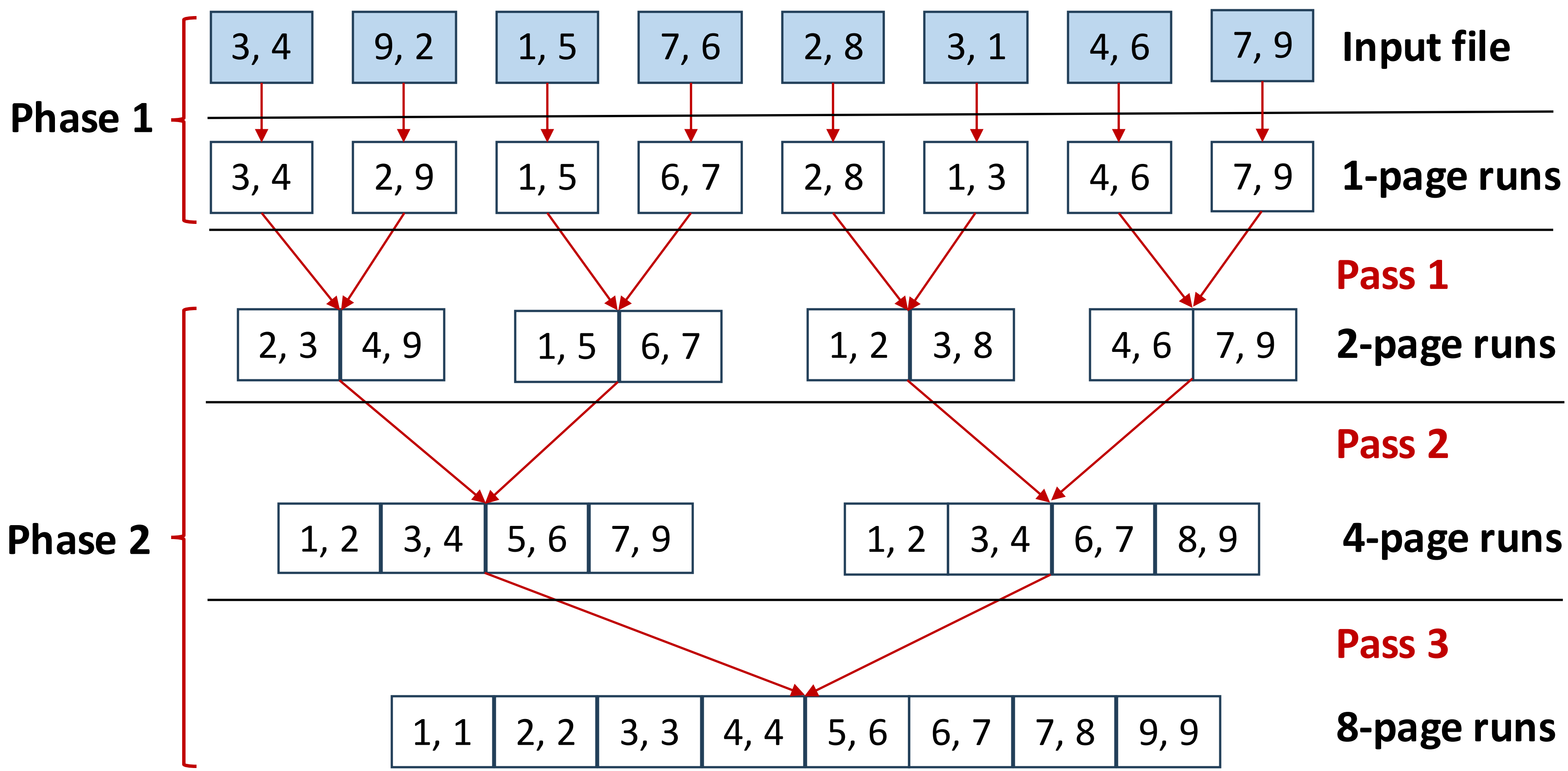


2-Way External Merge Sort: Phase 2

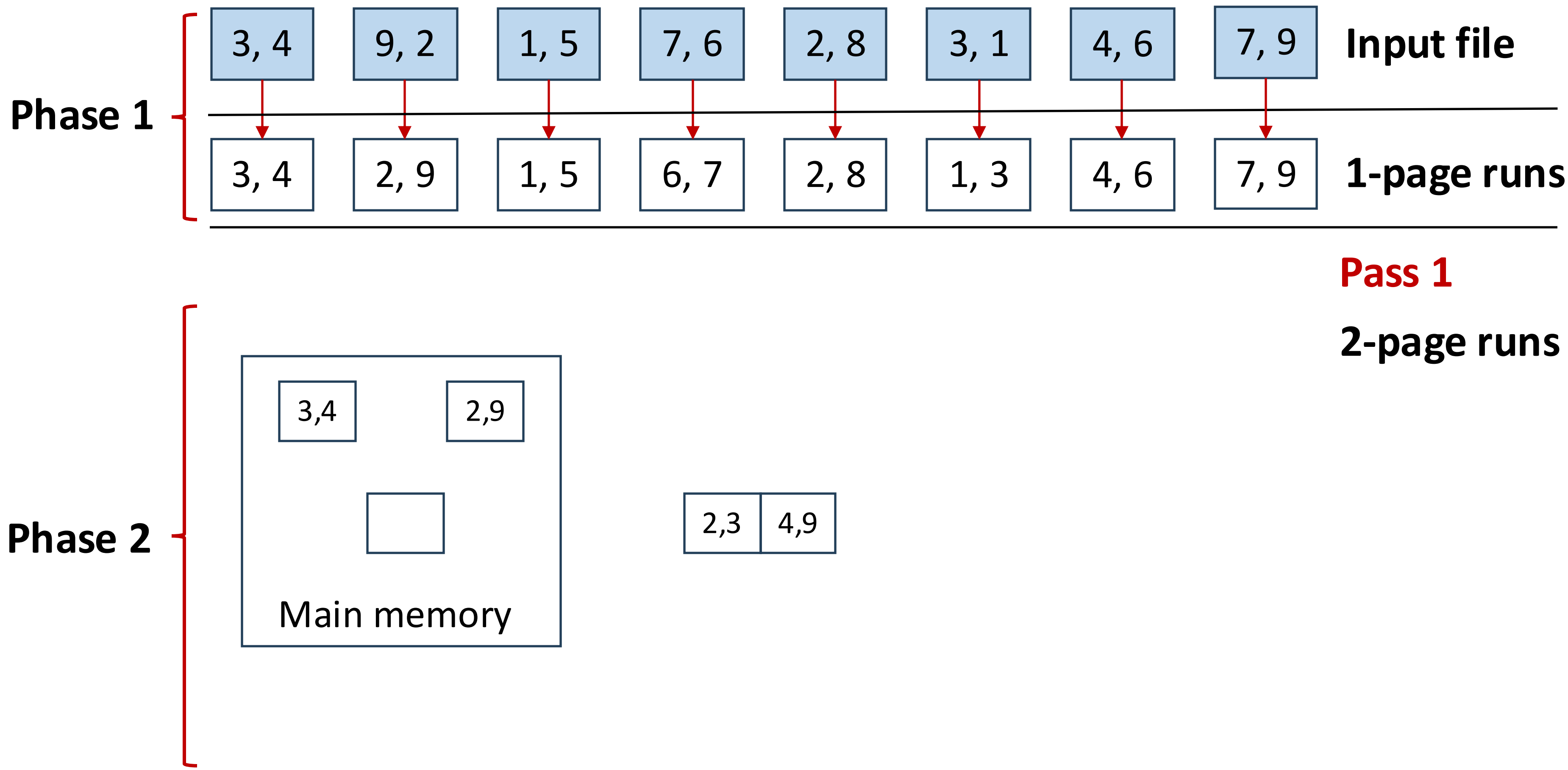
- Make multiple passes to merge runs
- Pass 1: Merge two runs of length 1 (page)
- Pass 2: Merge two runs of length 2 (pages)
- … until 1 run of length N
- Three buffer pages used



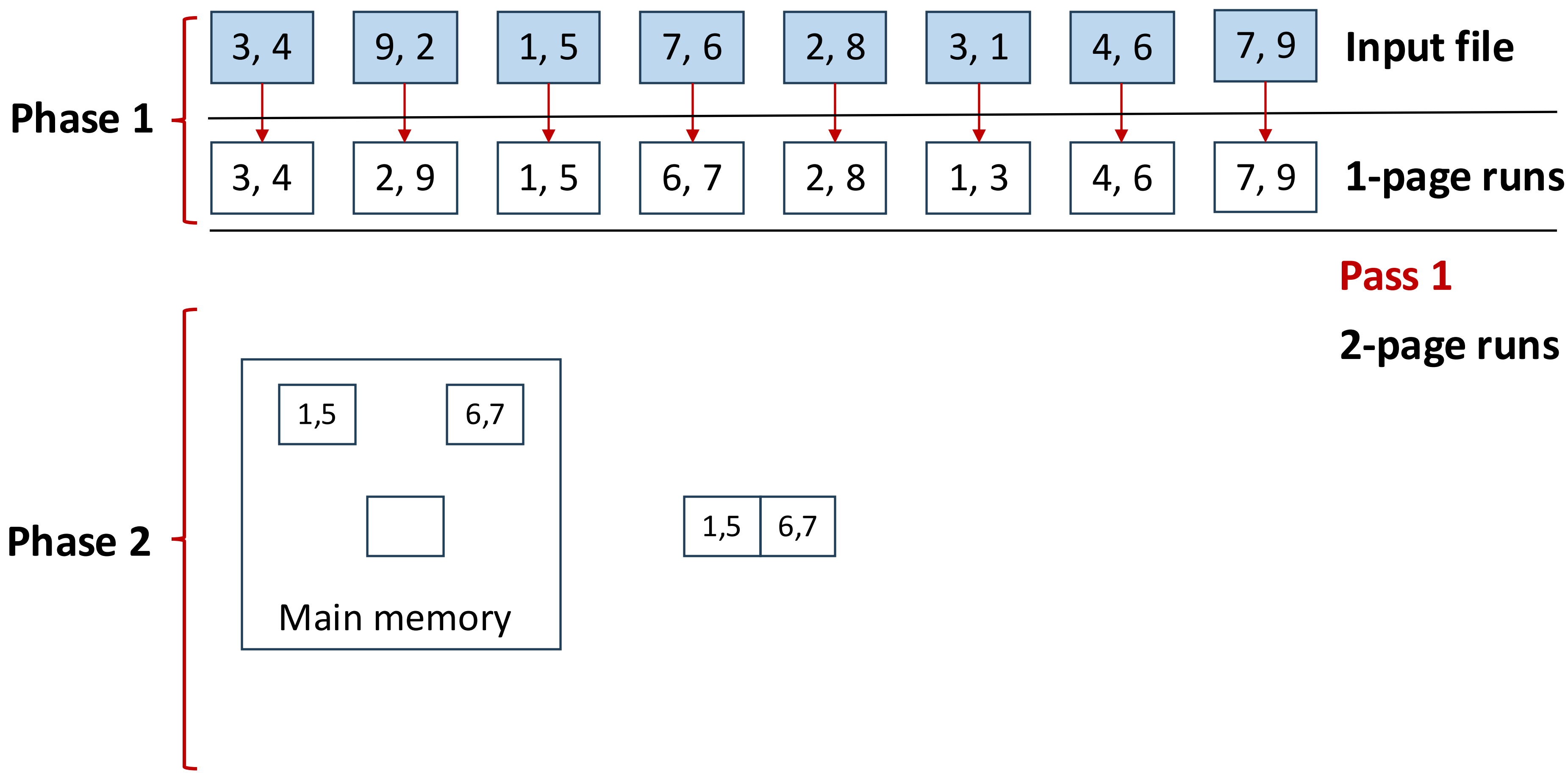
Phase 2: Example



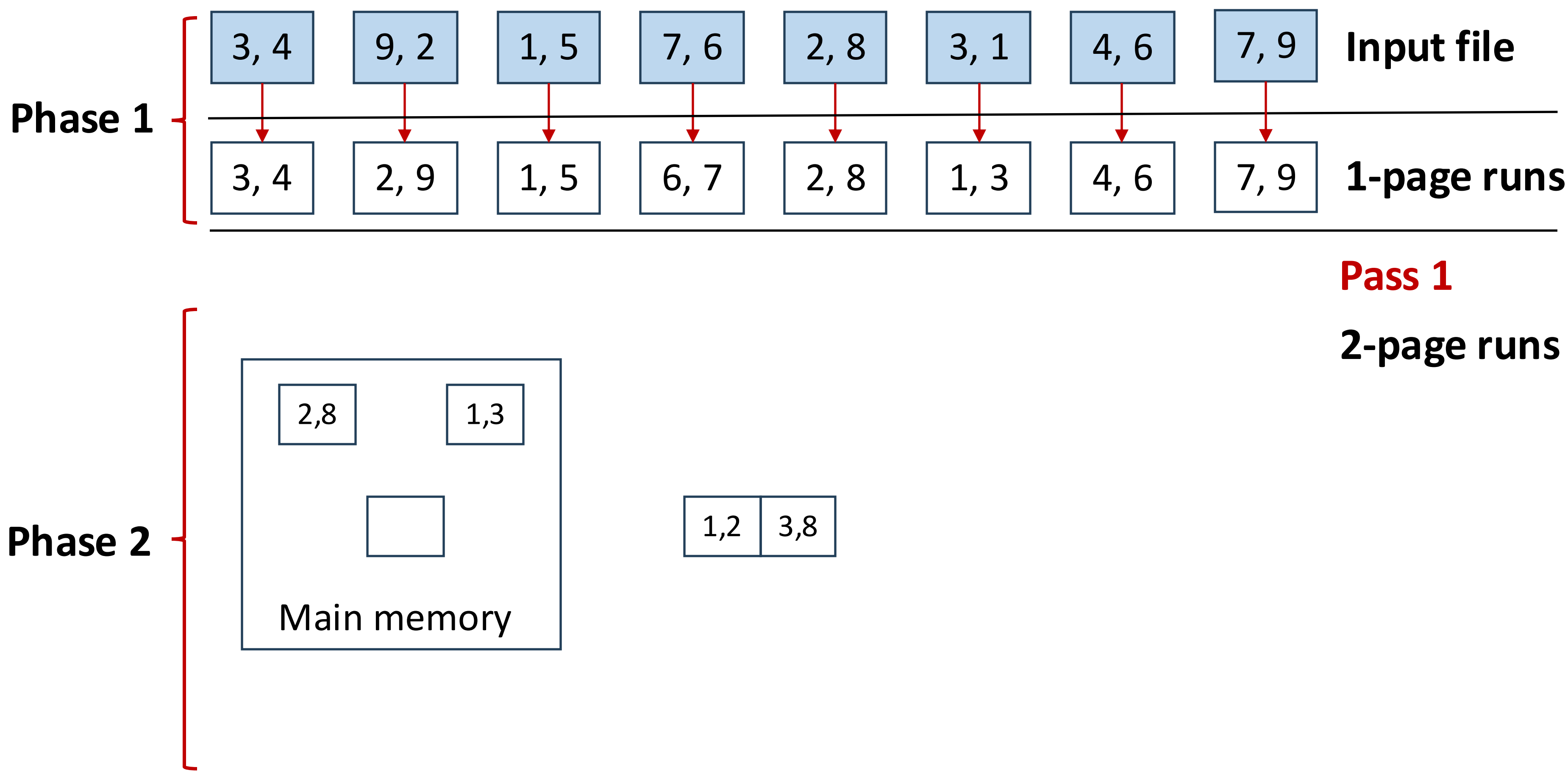
Phase 2: Example



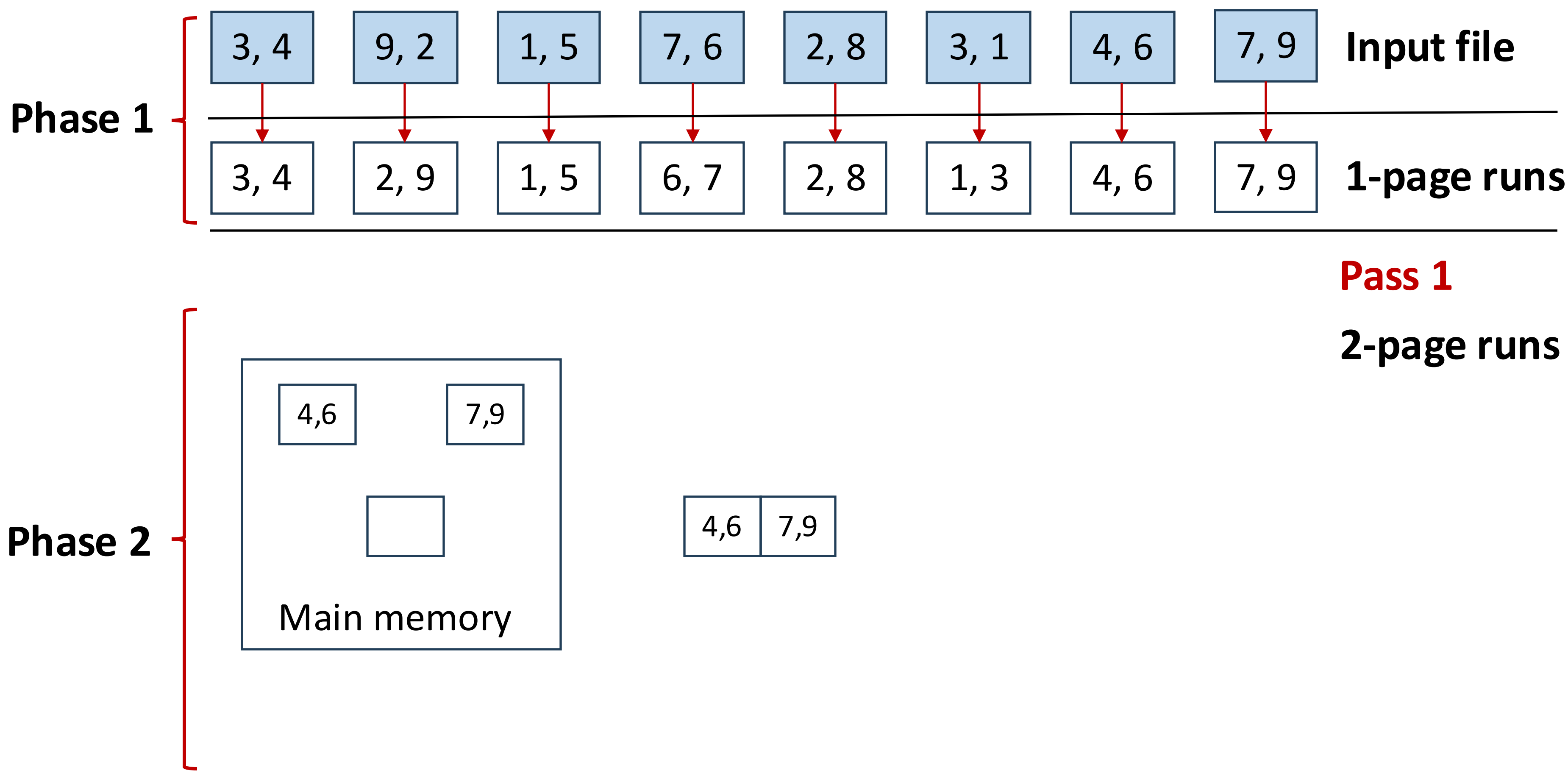
Phase 2: Example



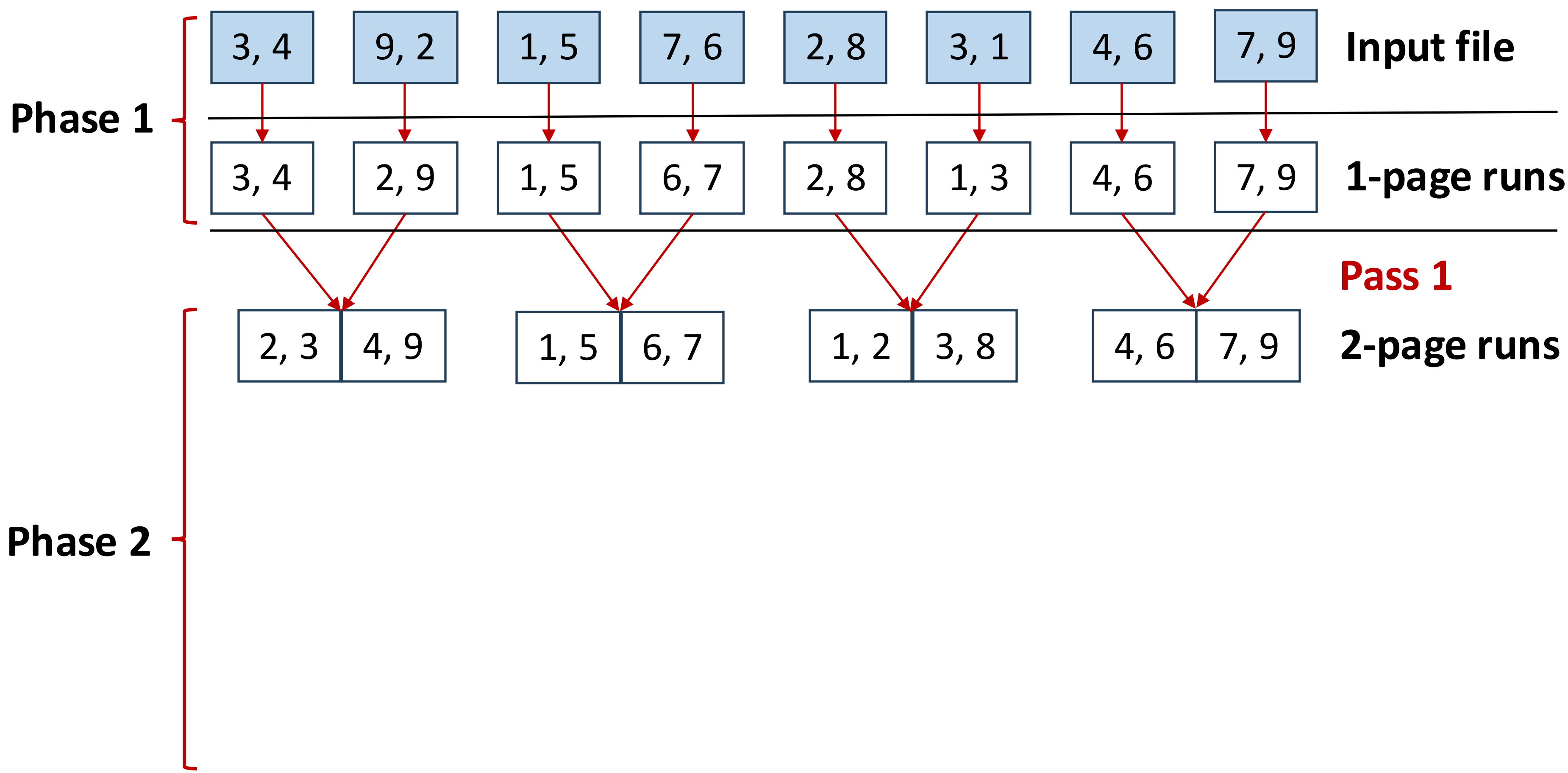
Phase 2: Example



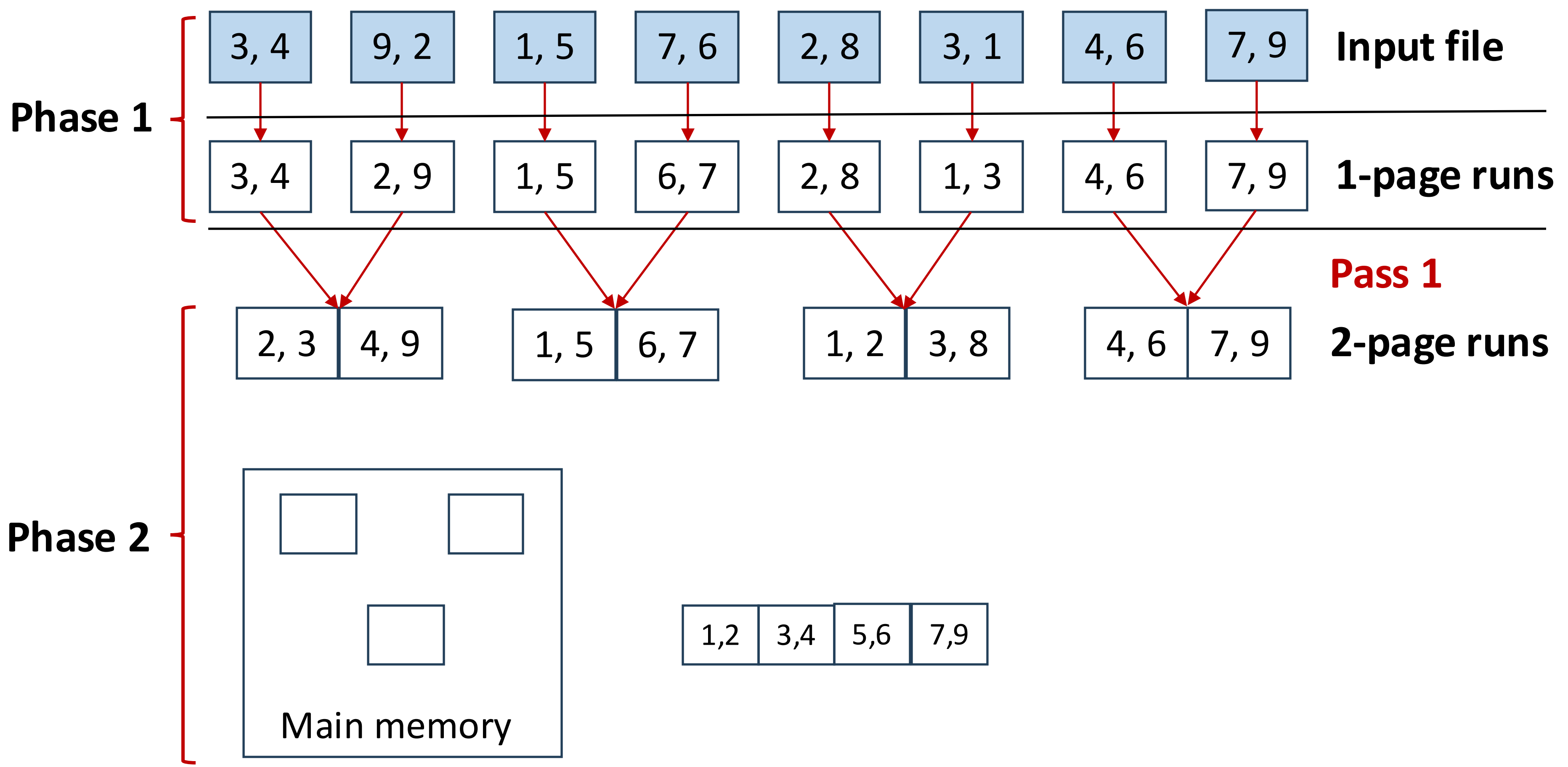
Phase 2: Example



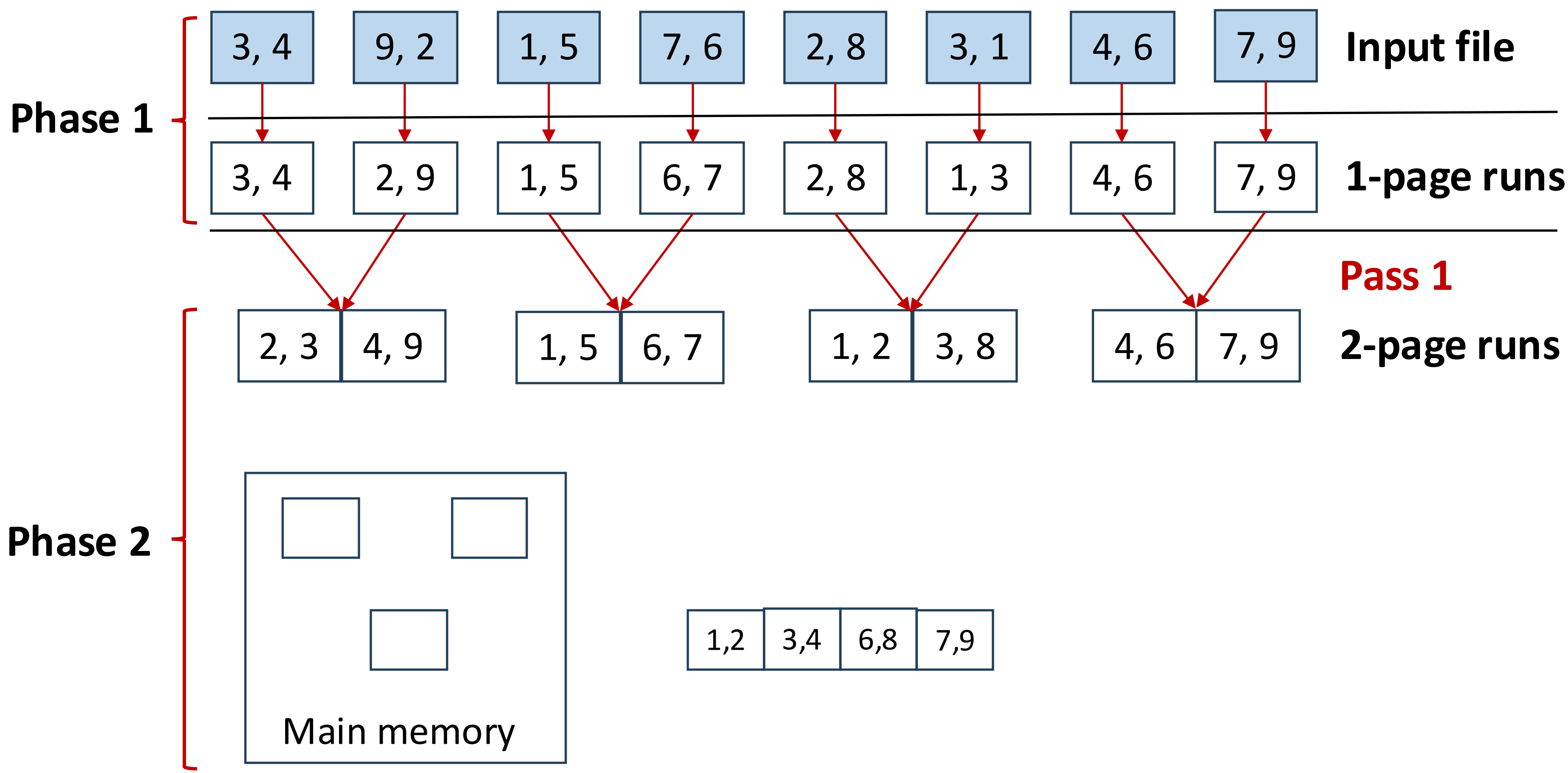
Phase 2: Example



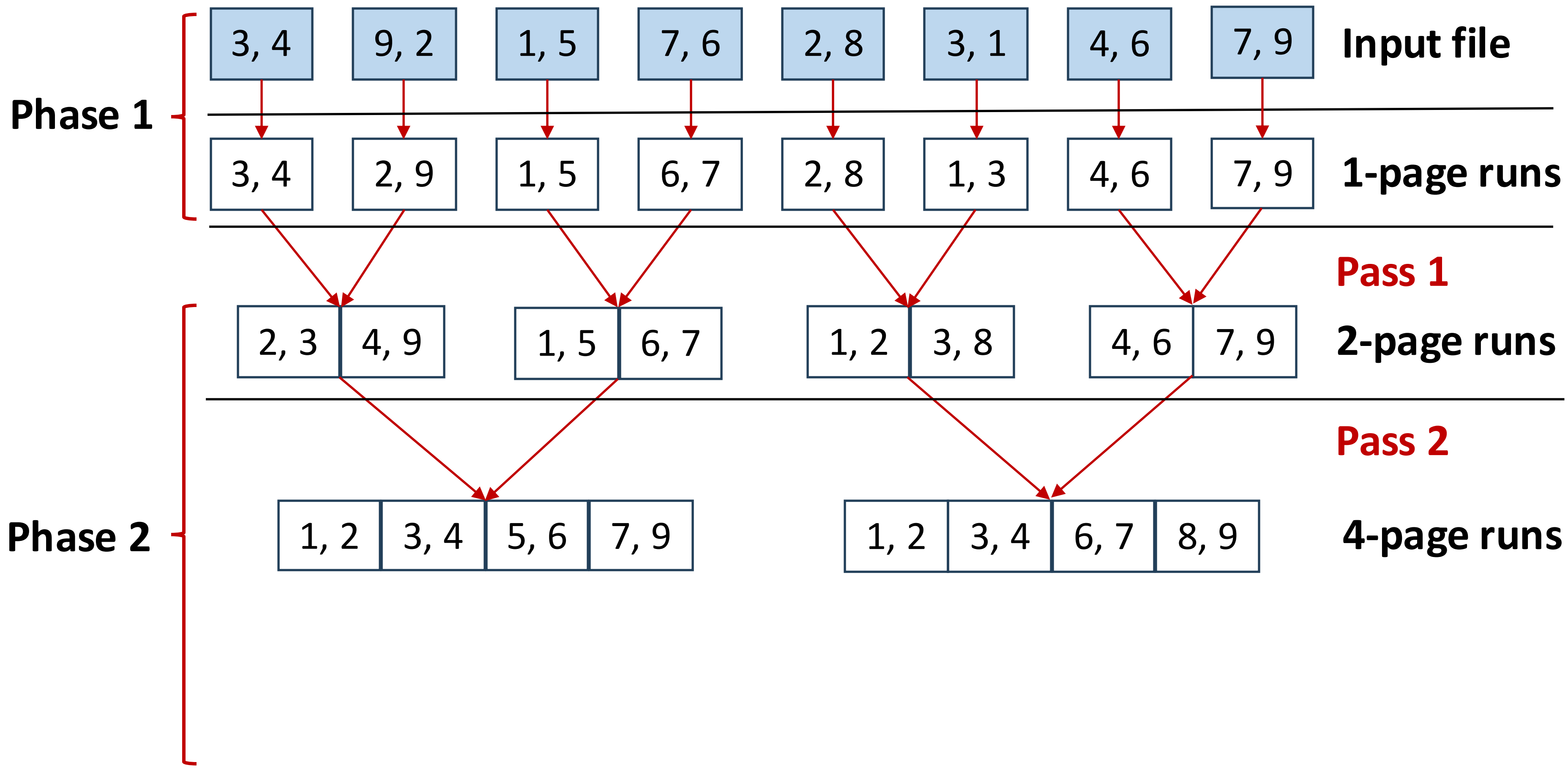
Phase 2: Example



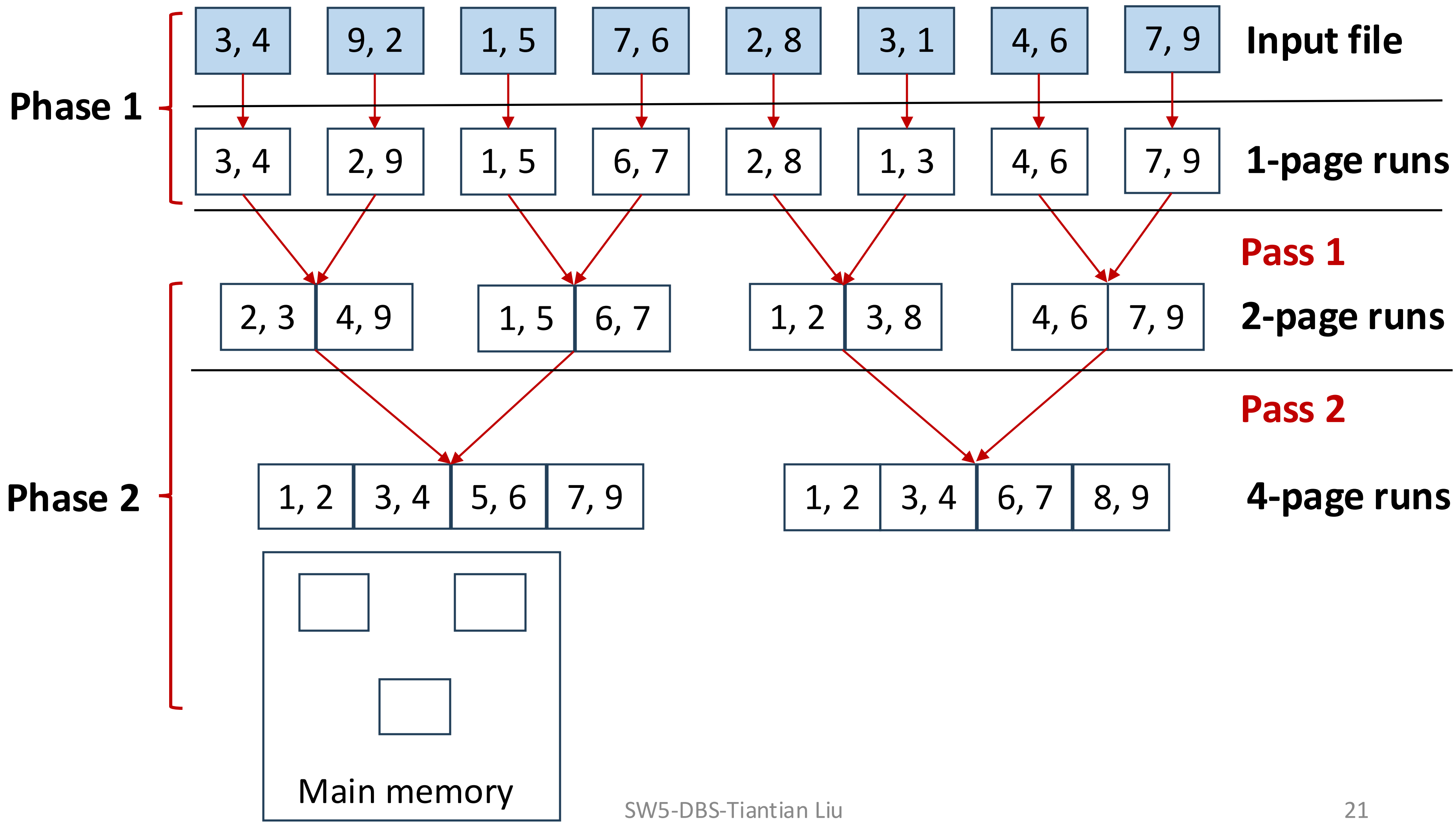
Phase 2: Example



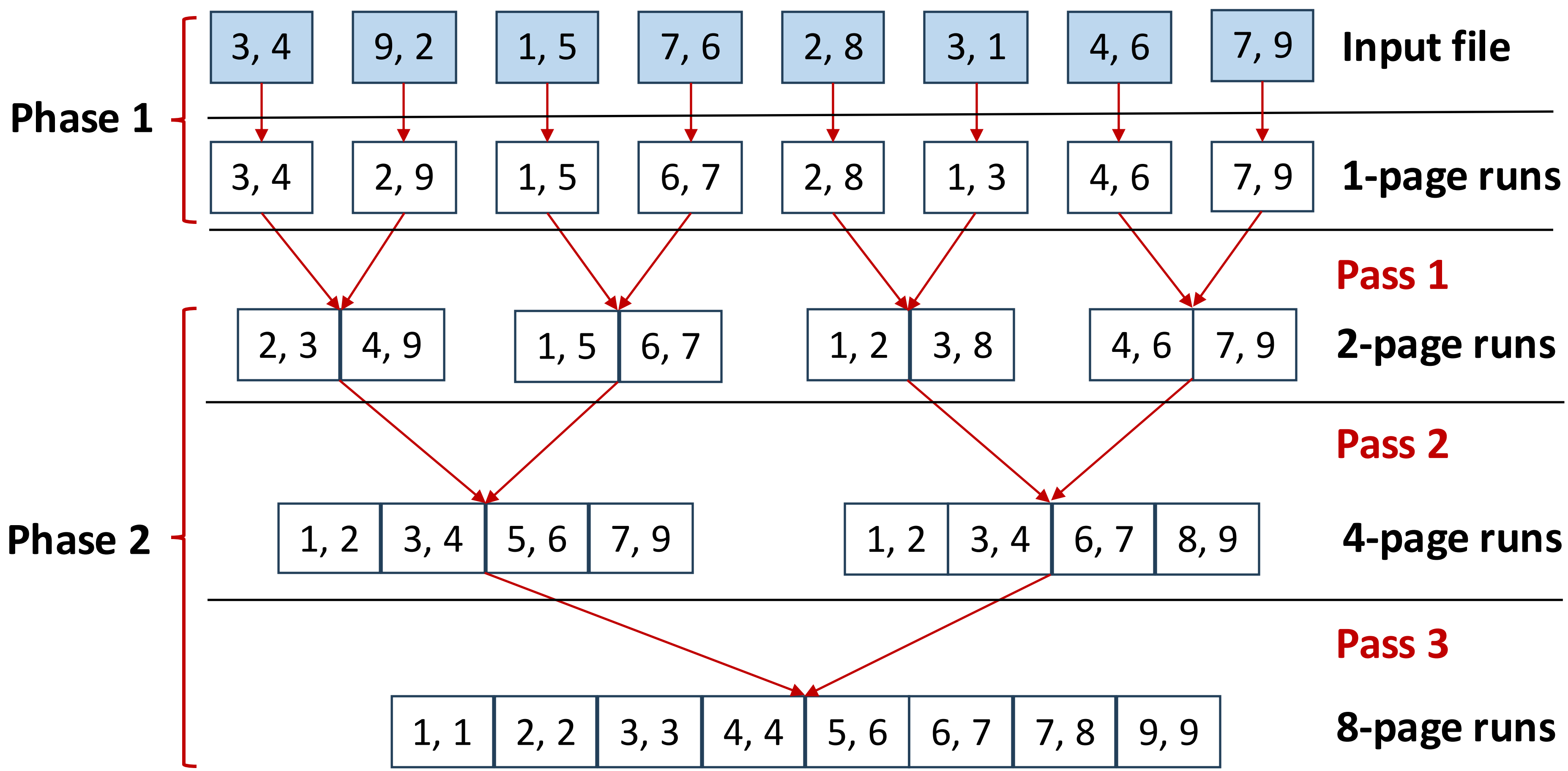
Phase 2: Example



Phase 2: Example



Phase 2: Example



2-Way External Merge Sort: Analysis

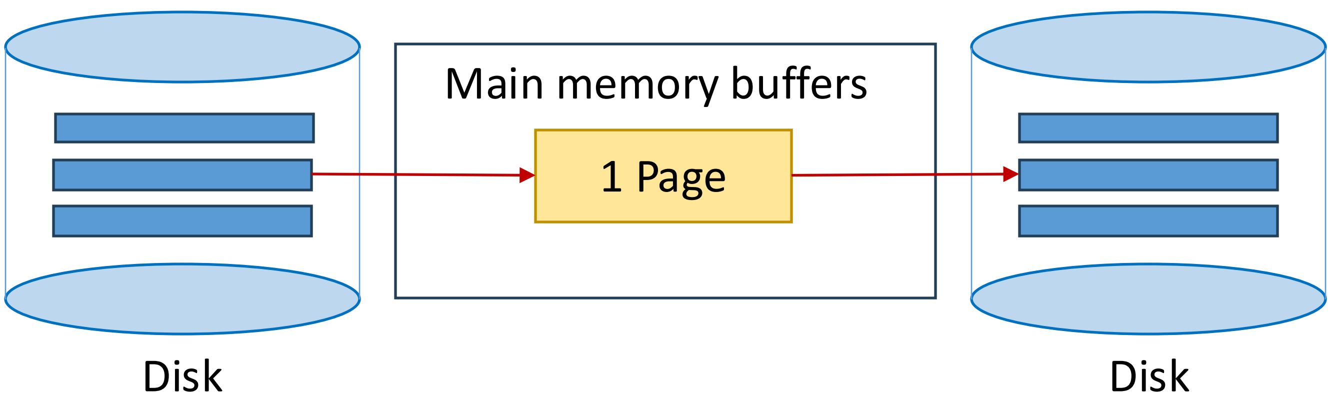
- Total I/O cost for sorting file with N pages
- Cost of Phase 1: 2N (read + write each page)
- Cost of Phase 2: $2N \times \lceil \log_2 N \rceil$
  - Number of passes in Phase 2: $\lceil \log_2 N \rceil$
  - Cost of each pass in Phase 2: 2N (Each pass we read + write each page)
- Total cost: $2N \times \lceil \log_2 N \rceil + 2N = 2N \times (\lceil \log_2 N \rceil + 1)$

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

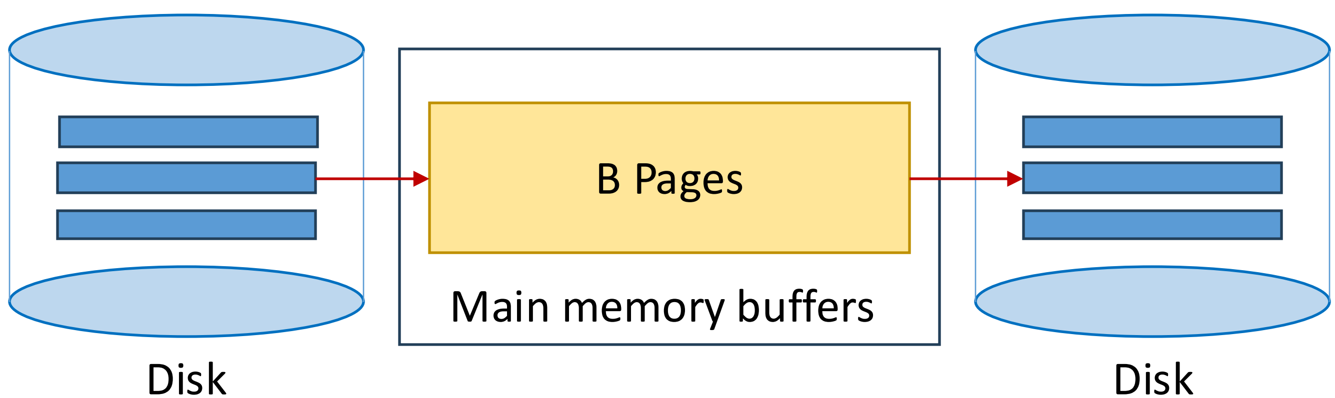
2-Way External Merge Sort

- 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?



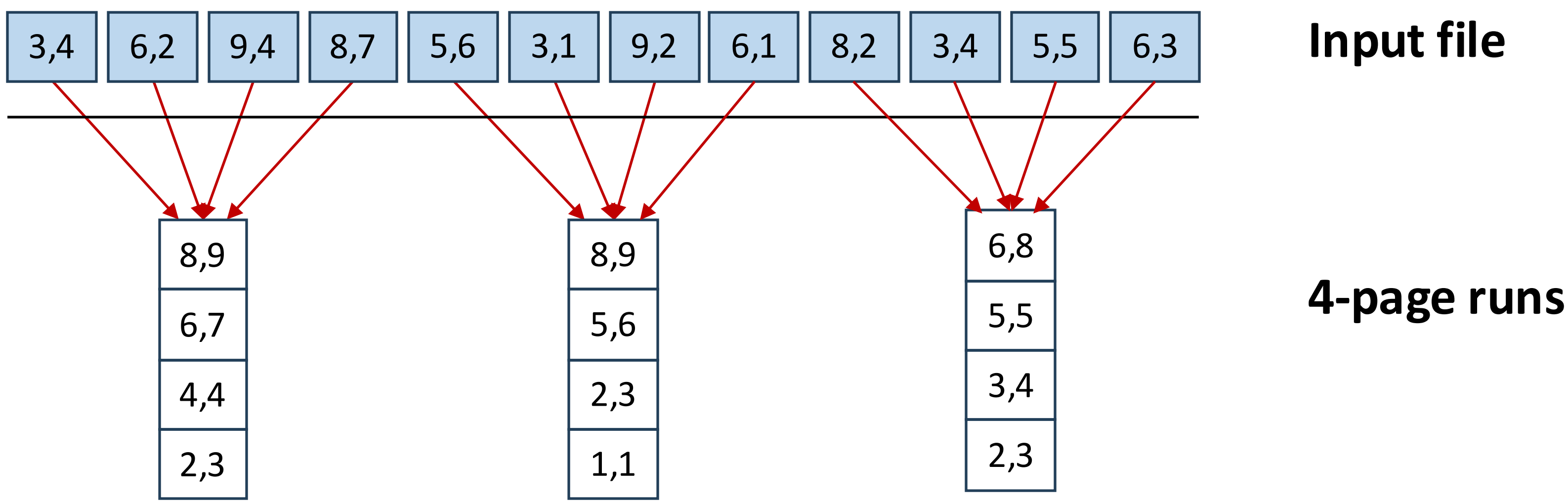
Multi-Way External Merge Sort

- 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 = $\lceil N/B \rceil$
- Cost of Phase 1 = 2N



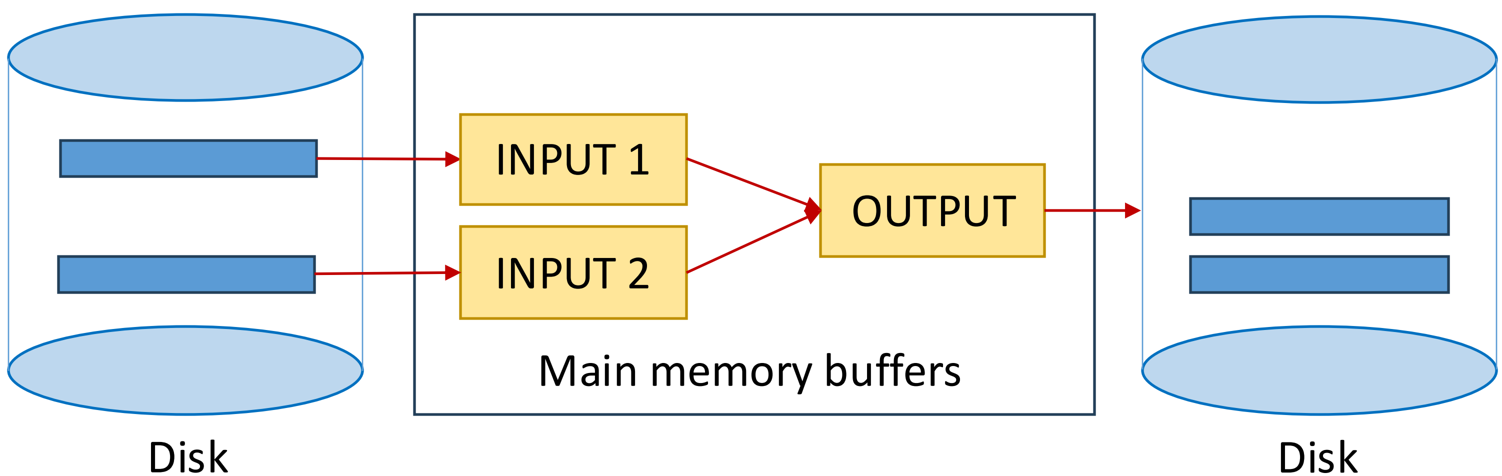
Multi-Way External Merge Sort

- #buffer pages B = 4



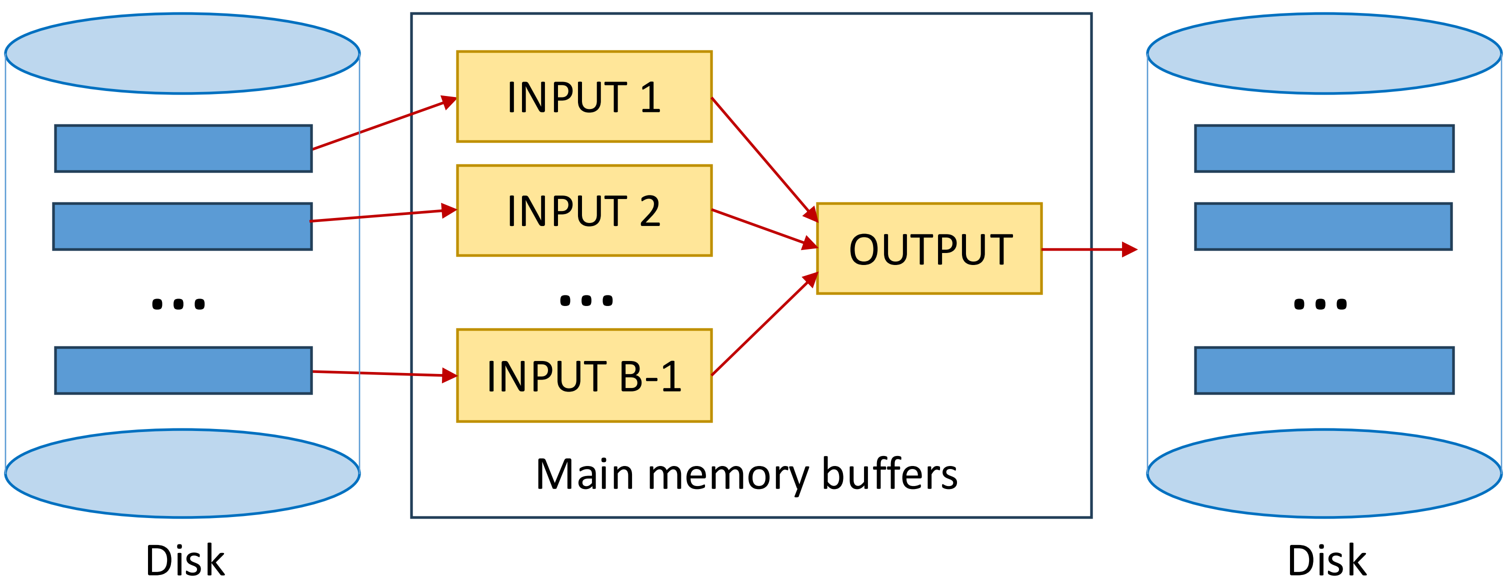
2-Way External Merge Sort: Phase 2

- 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?



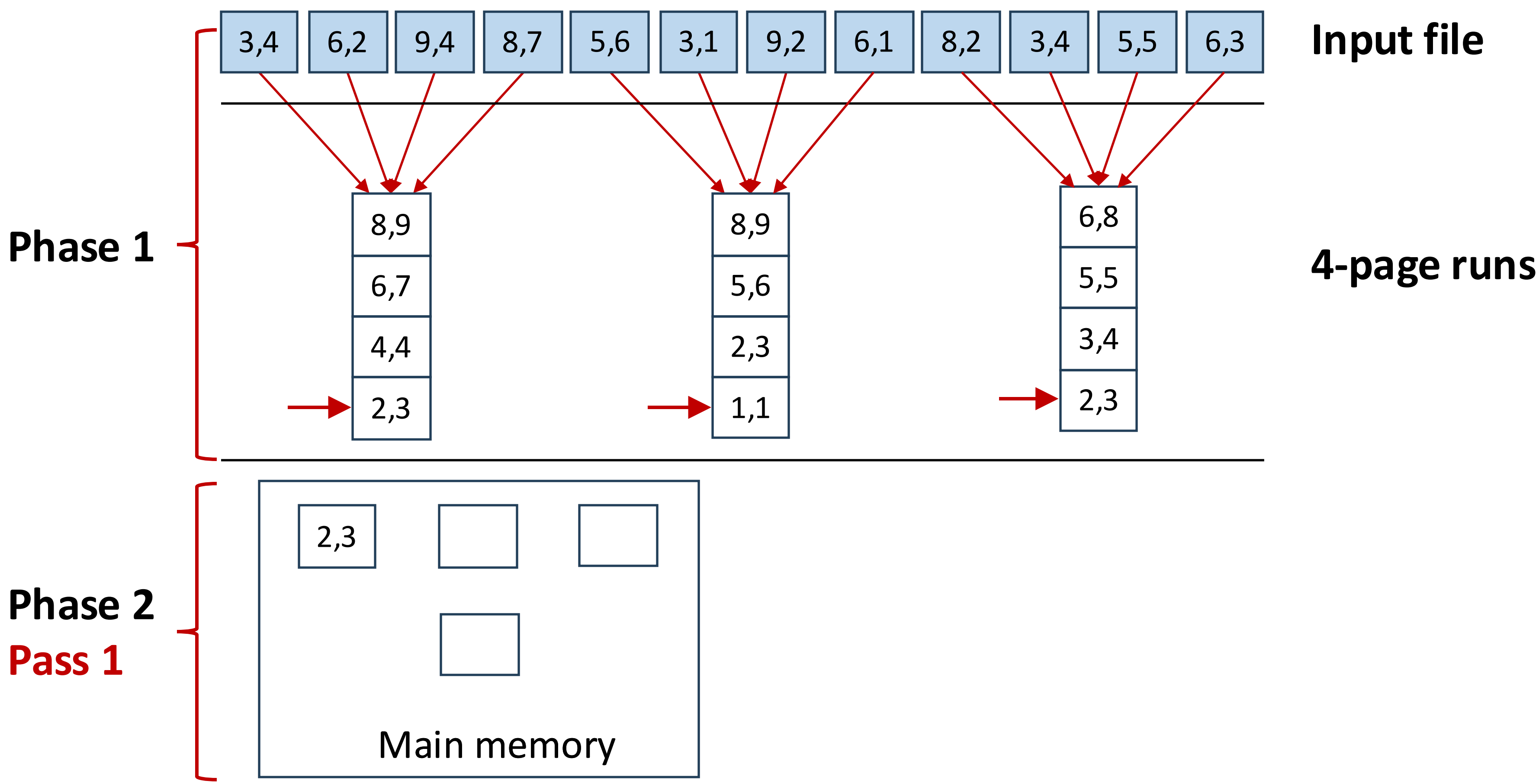
Multi-Way External Merge Sort

- 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



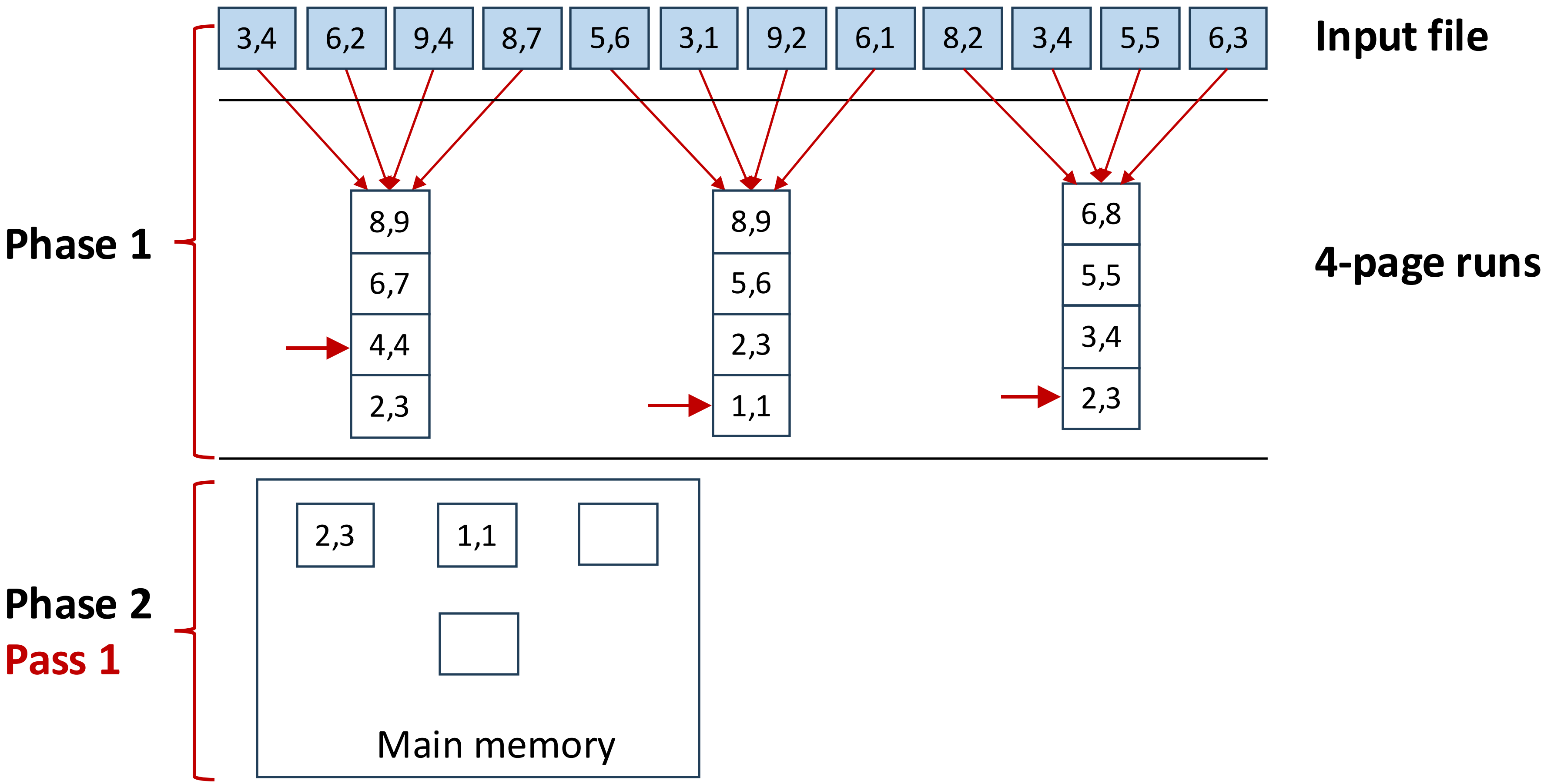
Multi-Way External Merge Sort: Phase 2

- #buffer pages B = 4



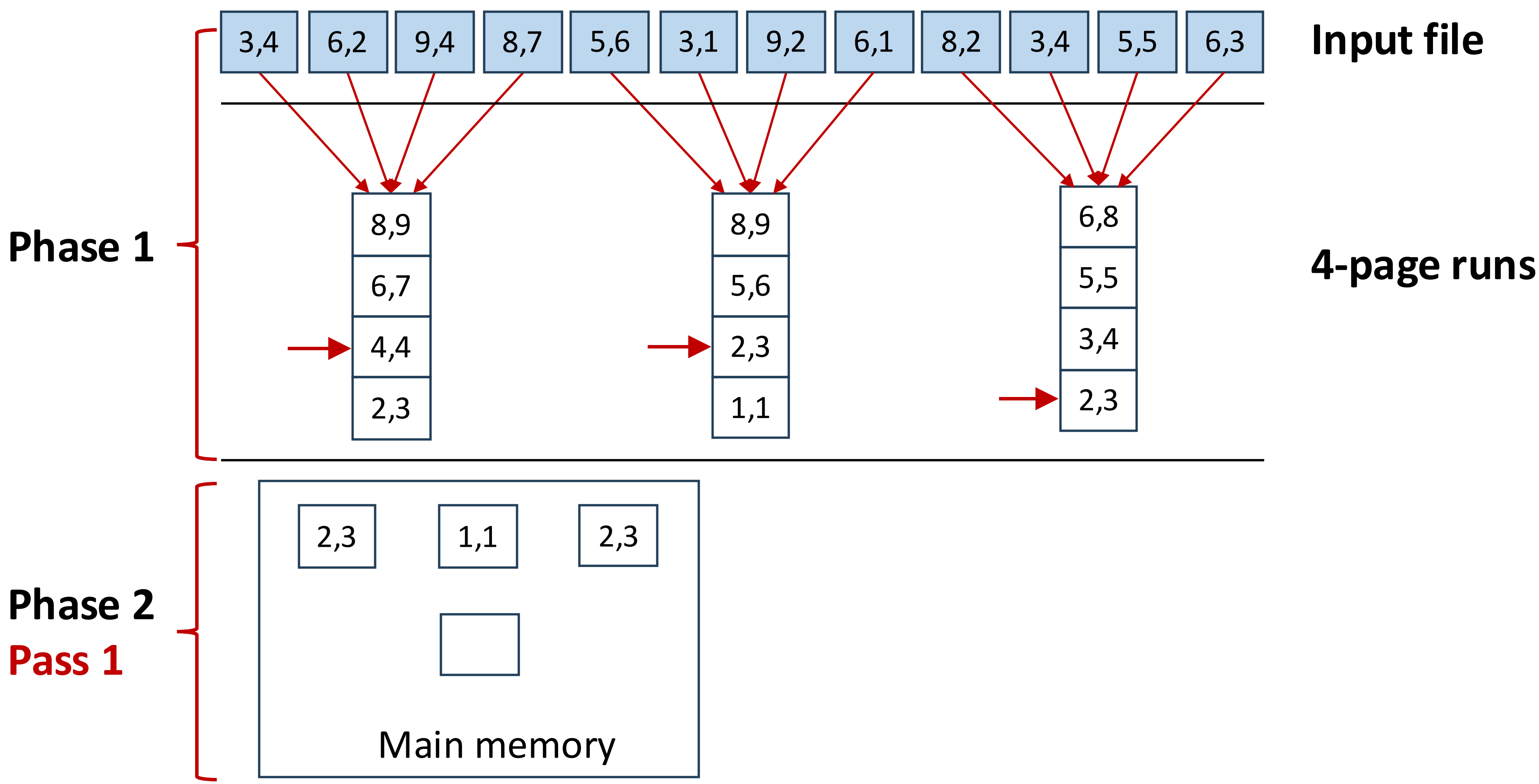
Multi-Way External Merge Sort: Phase 2

- #buffer pages B = 4



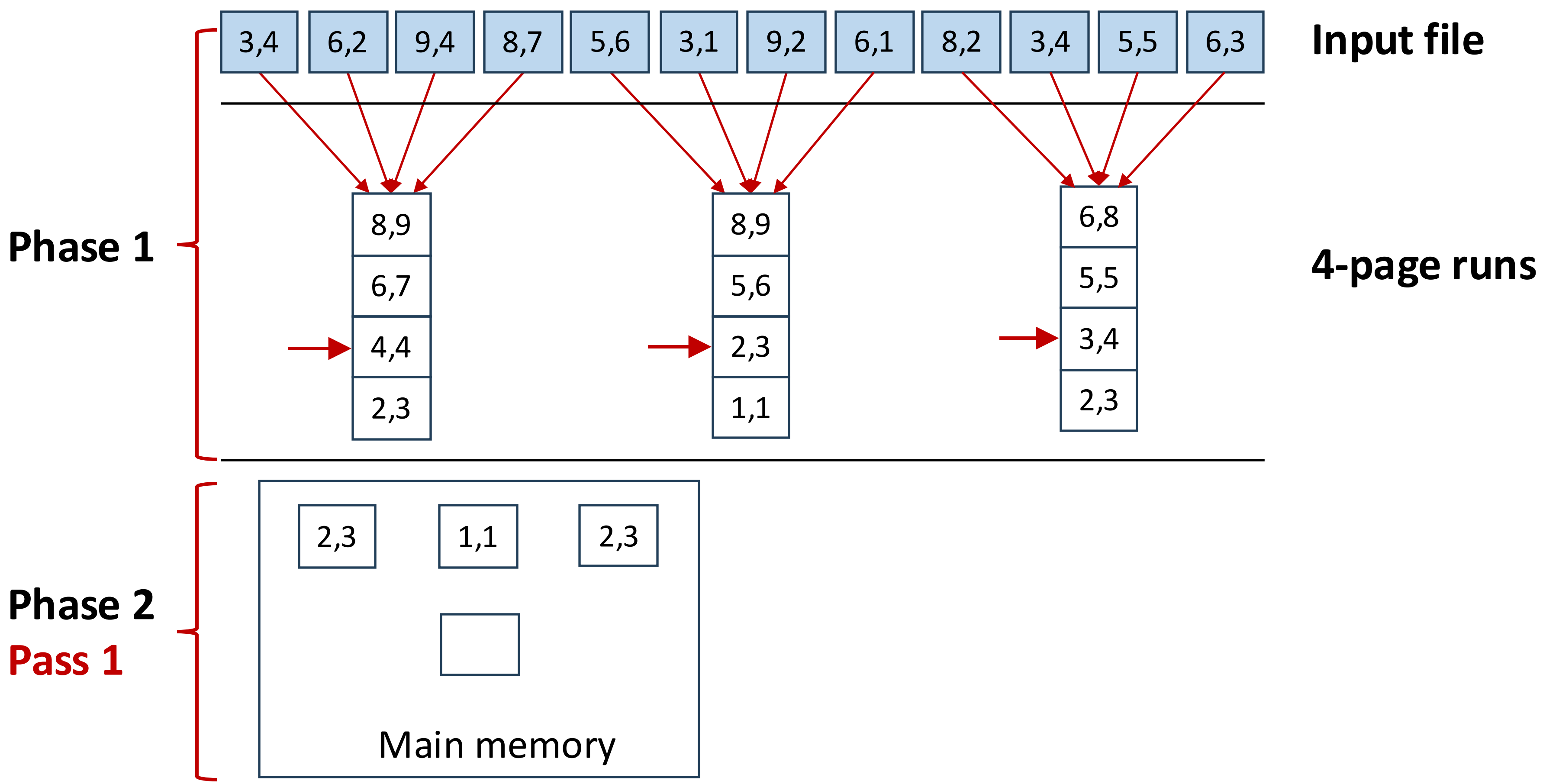
Multi-Way External Merge Sort: Phase 2

- #buffer pages B = 4



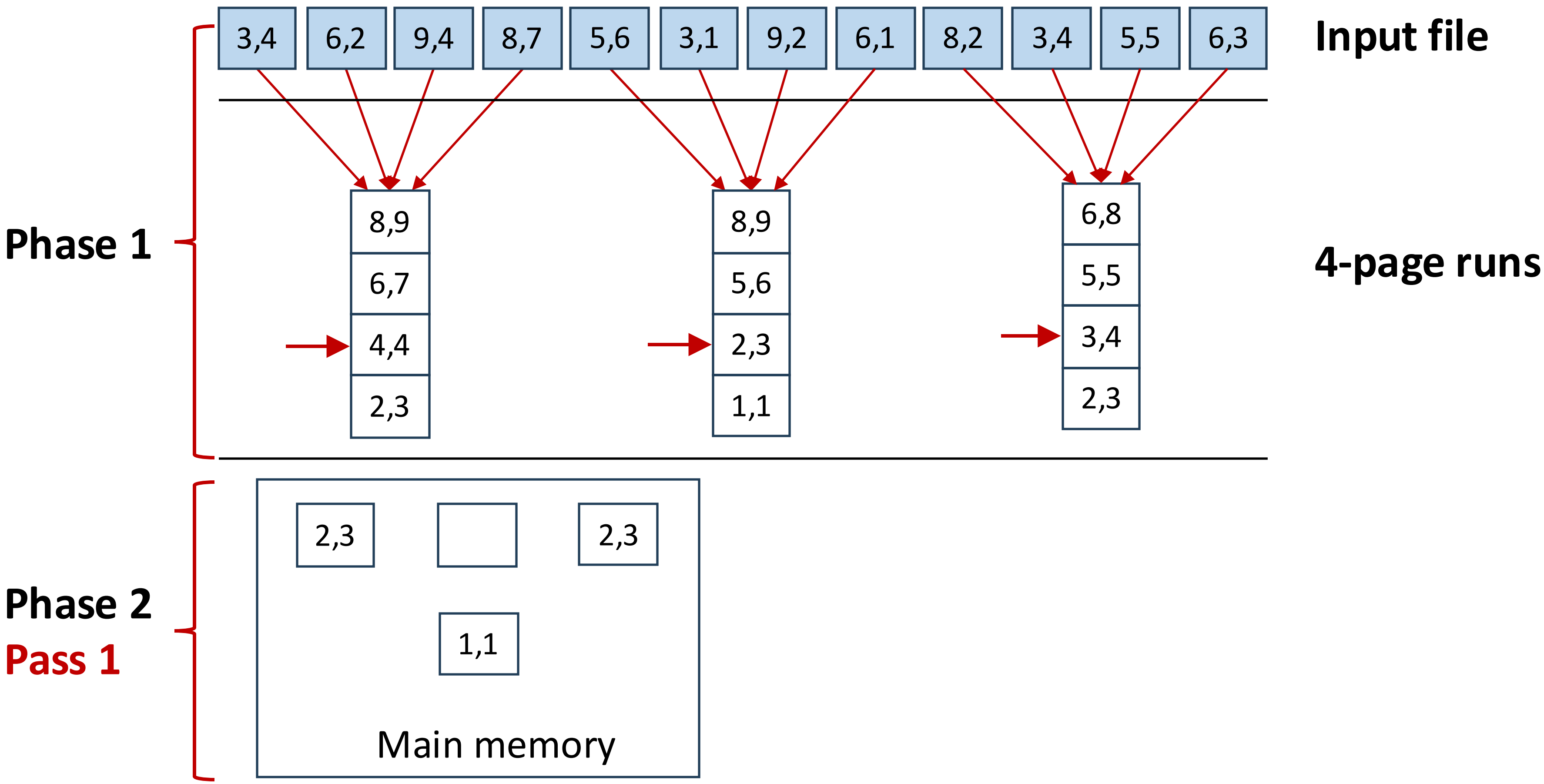
Multi-Way External Merge Sort: Phase 2

- #buffer pages B = 4



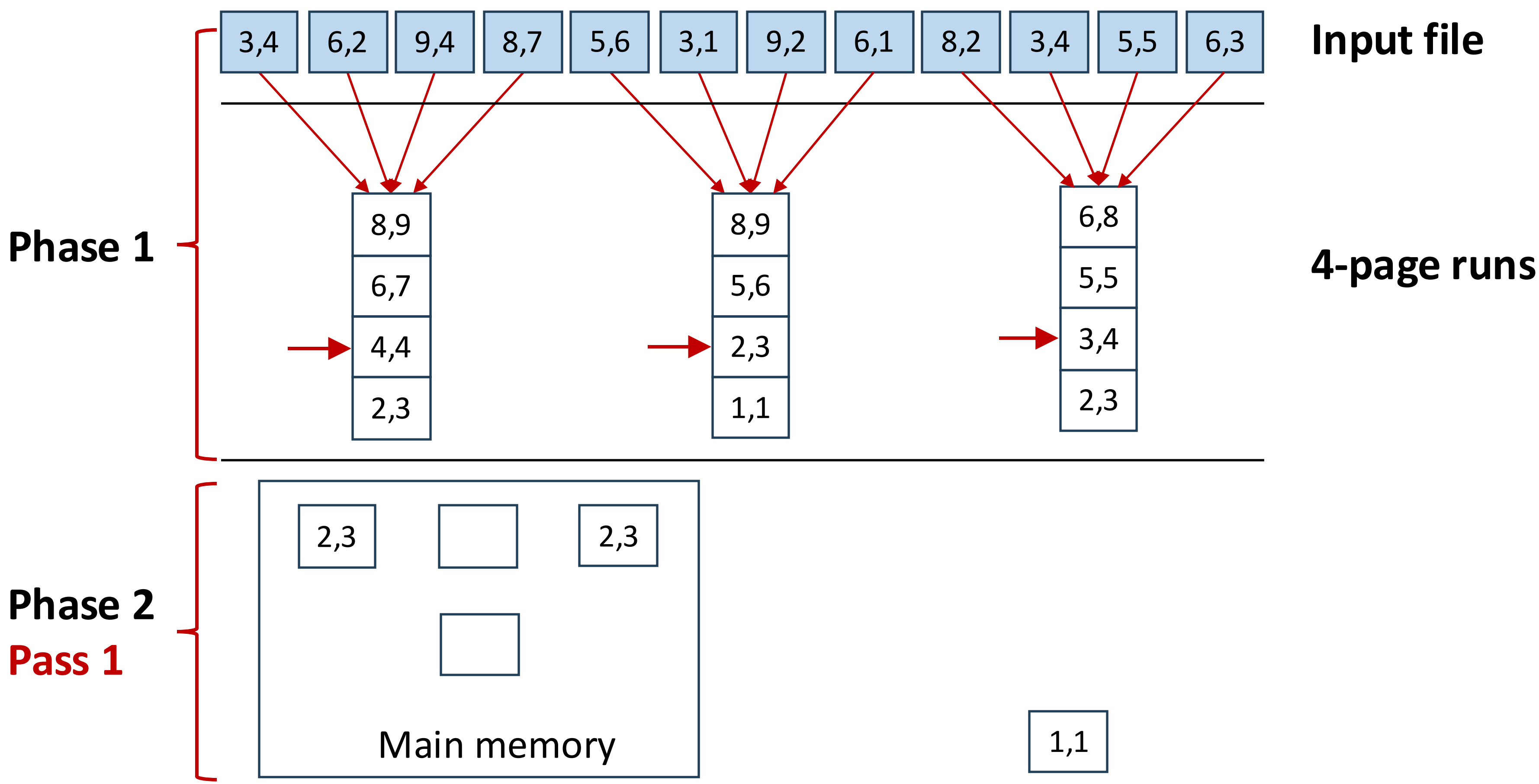
Multi-Way External Merge Sort: Phase 2

- #buffer pages B = 4



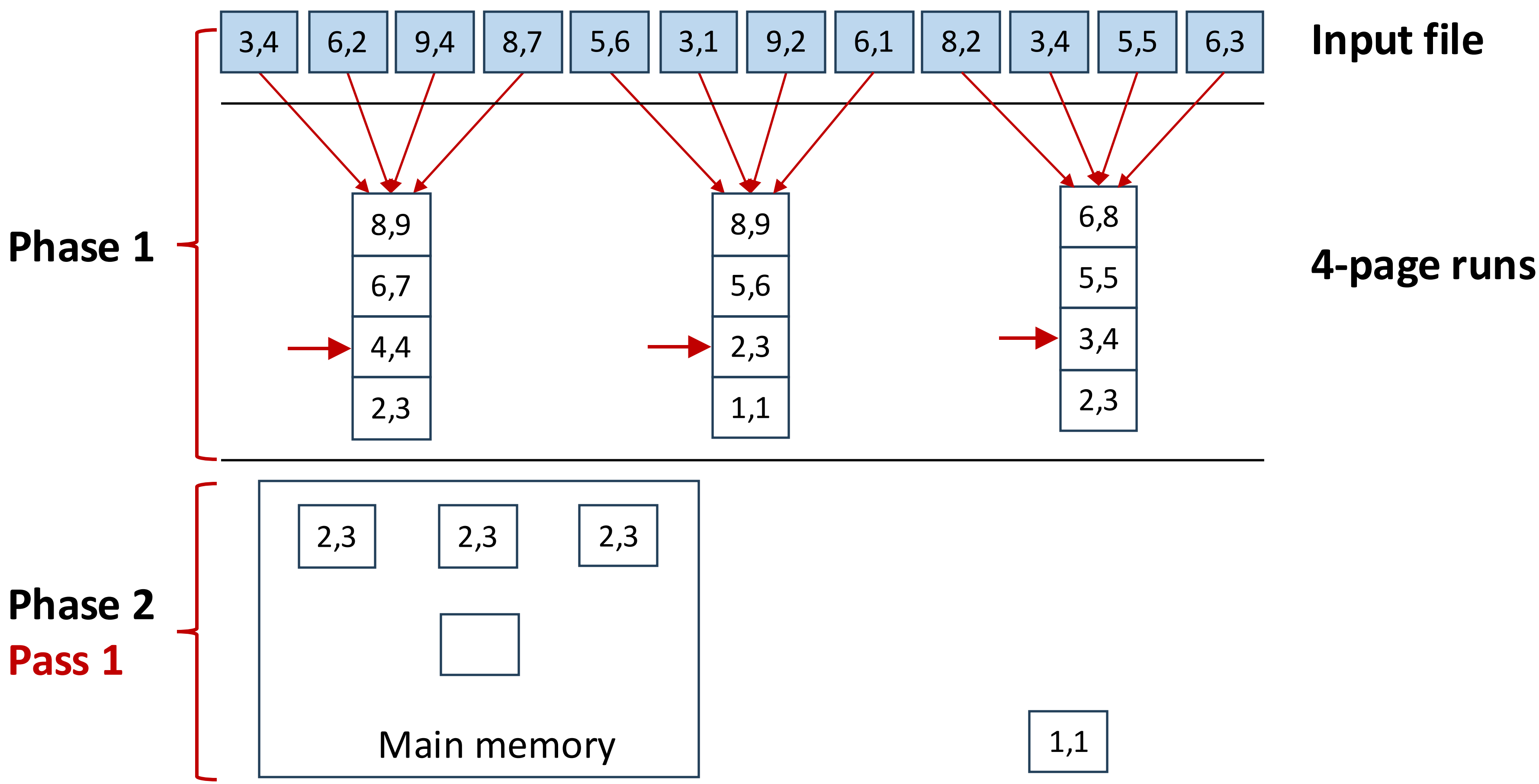
Multi-Way External Merge Sort: Phase 2

- #buffer pages B = 4



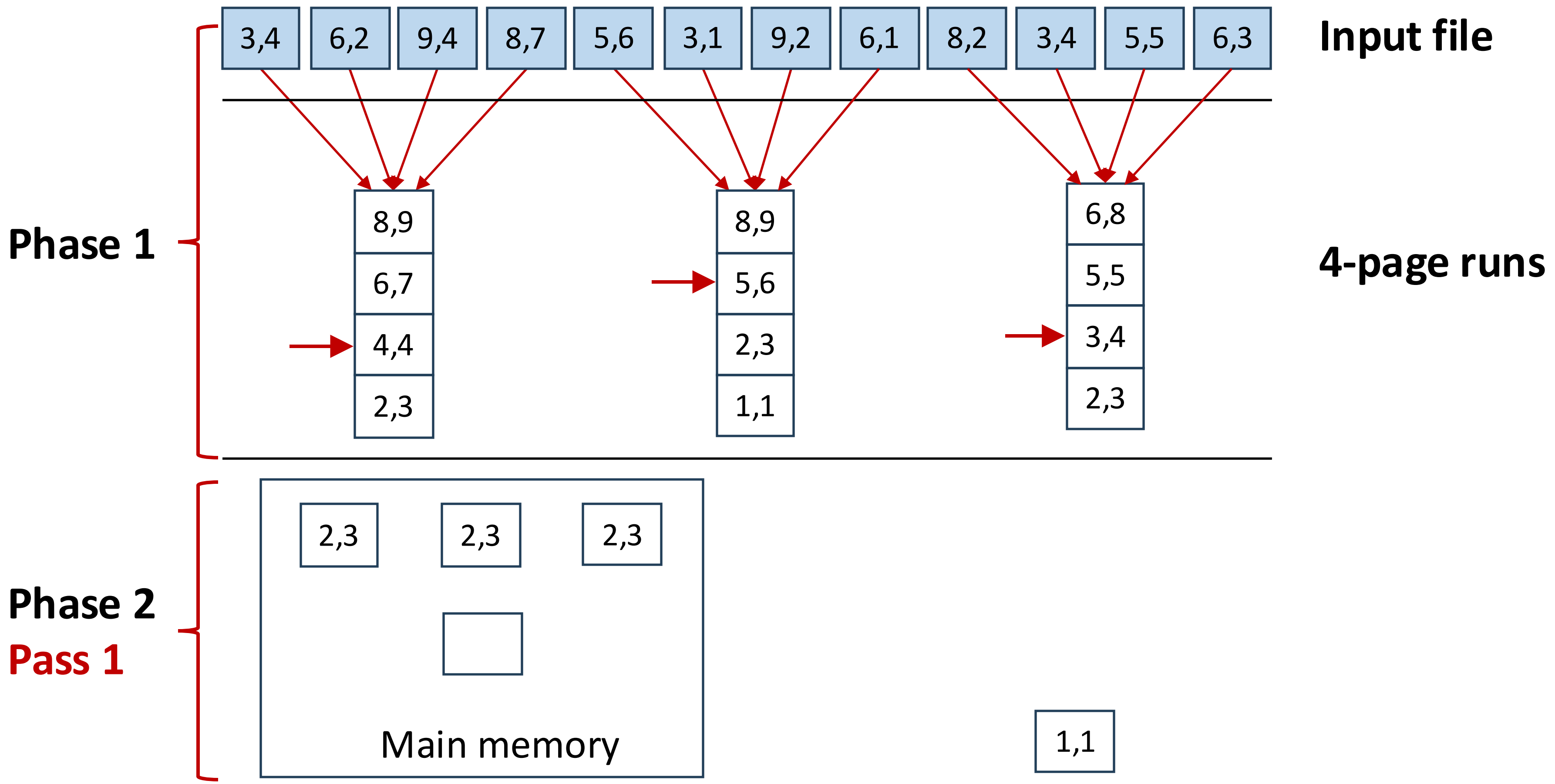
Multi-Way External Merge Sort: Phase 2

- #buffer pages B = 4



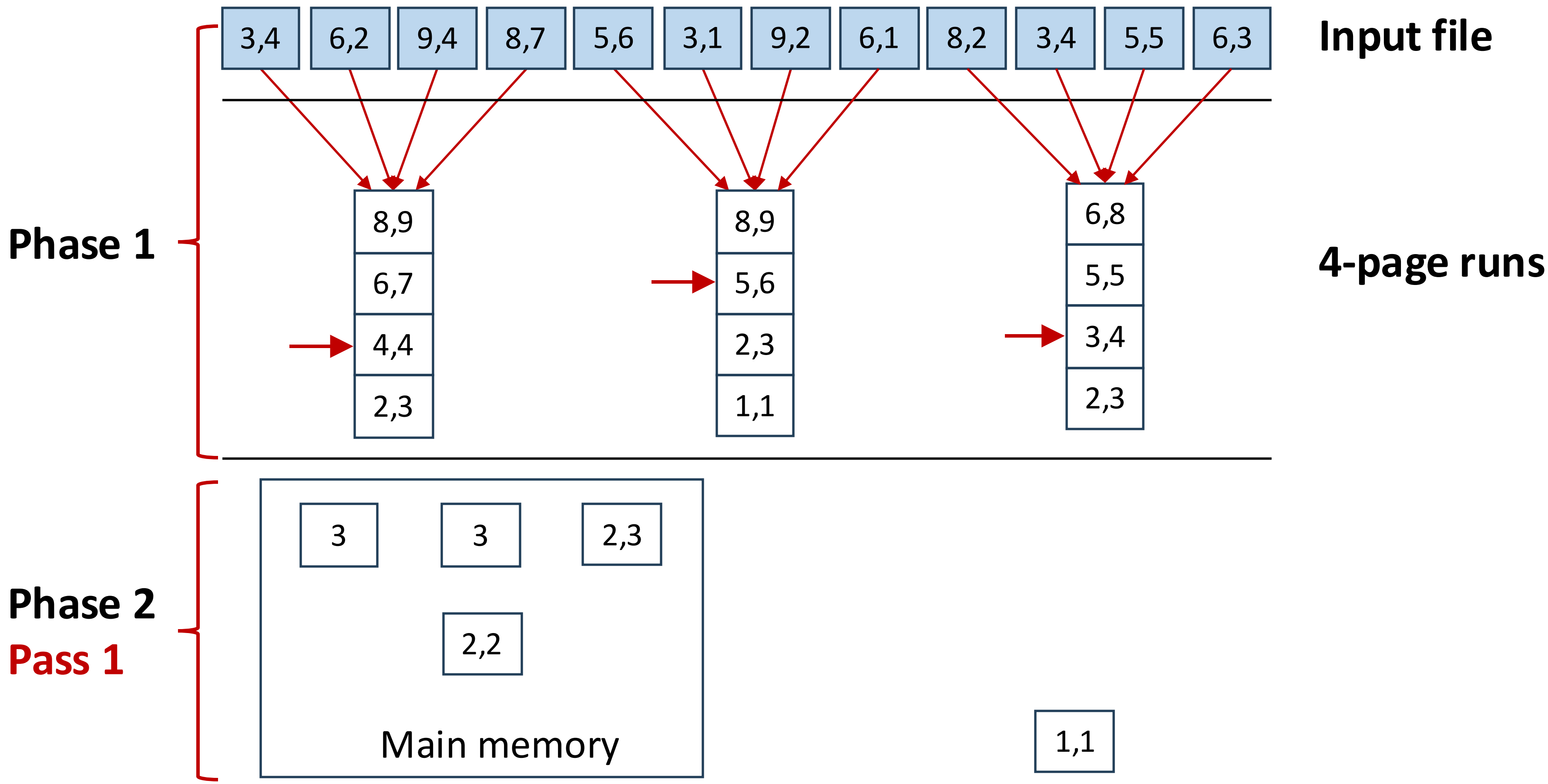
Multi-Way External Merge Sort: Phase 2

- #buffer pages B = 4



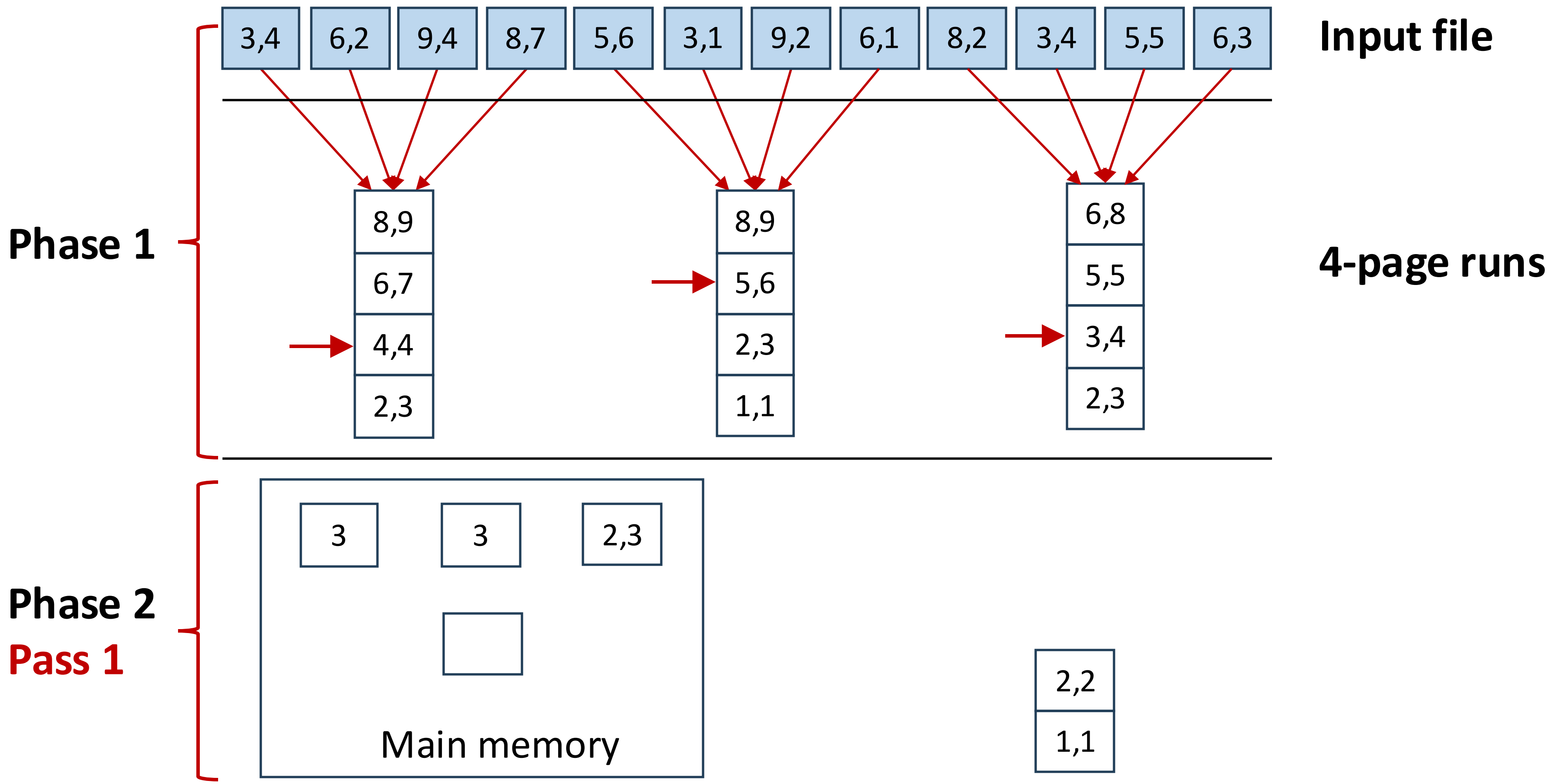
Multi-Way External Merge Sort: Phase 2

- #buffer pages B = 4



Multi-Way External Merge Sort: Phase 2

- #buffer pages B = 4



Multi-Way External Merge Sort: Analysis

- Total I/O cost for sorting file with N pages
- Cost of Phase 1: 2N (read + write each page)
- Cost of Phase 2: $2N \times \lceil \log_{B-1} \lceil N/B \rceil \rceil$
  - Number of passes in Phase 2: $\lceil \log_{B-1} \lceil N/B \rceil \rceil$
    - If #passes in Phase 2 is P, then: $B(B-1)^P = N$
    - $P = \lceil \log_{B-1} \lceil N/B \rceil \rceil$
  - Cost of each pass in Phase 2: 2N (Each pass we read + write each page)
- **Total cost:** $2N \times \lceil \log_{B-1} \lceil N/B \rceil \rceil + 2N$

$$= 2N \times (\lceil \log_{B-1} \lceil N/B \rceil \rceil + 1)$$

Exercise 1

Let relation *r* have the following properties: *r* has 40,000 tuples and 40 tuples of *r* fit on one page.

Assume B pages of memory and B ≤ 500

Q: Estimate the I/O cost required for Multi-Way External Merge Sort of *r*

Exercise 1--Solution

Let relation *r* have the following properties: *r* has 40,000 tuples and 40 tuples of *r* fit on one page.

Assume B pages of memory and B ≤ 500

Q: Estimate the I/O cost required for Multi-Way External Merge Sort of *r*

N = 40,000/40 = 1000

Cost = $2N \times (\lceil \log_{B-1} \lceil N/B \rceil \rceil + 1) = 2000 \times (\lceil \log_{B-1} \lceil 1000/B \rceil \rceil + 1)$

Agenda

AALBORG UNIVERSITY

- Basic Steps in Query Processing
- Sorting
- Join Strategies
- Cost-based Query Optimization
- Logical Query Optimization

Joins

- Joins are very common

```
select *
from instructor i, teaches t
where t.ID = i.ID;
```

- Join techniques we will cover:
  - Simple nested-loop join
  - Page-oriented nested-loop join
  - Block nested-loop join
  - Sort-merge join
  - Hash join

Simple Nested-Loop Join

```
foreach tuple r in R do
        foreach tuple s in S do
                if r.ID == s.ID then add < r, s > to result
```

- Suppose R has 1000 pages, S has 500 pages, and each page (of R and S) has p = 100 tuples?
- Cost = $|R| + (p*|R|)*|S|$ = 1000 + 100*1000*500 IOs
- What if smaller relation (S) was “outer”?

Page-Oriented Nested-Loop Join

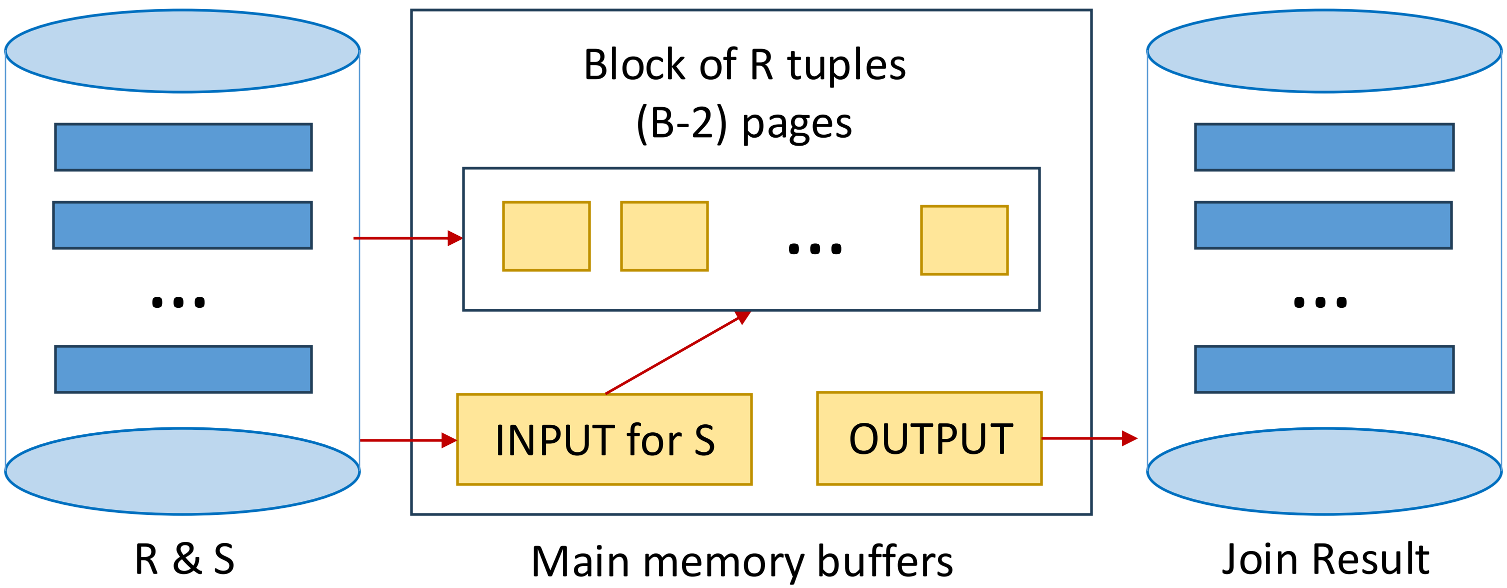
AALBORG UNIVERSITY

```
foreach page p_r in R do
    foreach page p_s in S do
        foreach tuple r in R do
            foreach tuple s in S do
                if r.ID == s.ID then add < r, s > to result
```

- Suppose R has 1000 pages, S has 500 pages, and each page (of R and S) has 100 tuples?
- Cost = |R| + |R|*|S| = 1000 + 1000*500 IOs
- Much better than naïve per-tuple approach!
- The trick is to reduce the # complete reads of the inner table

Block Nested-Loop Join

- Page-oriented NL doesn't exploit extra buffers :(
- Idea to use memory efficiently:



Examples of Block Nested-Loop Join

- Say we have B = 100+2 memory buffers
- Join cost = |outer| + (#outer blocks * |inner|)
  - #outer blocks = |outer| / (B - 2)
- With R as outer (|R| = 1000):
  - Scanning R costs |R| = 1000 IO's (done in 10 blocks)
  - Per block of R, we scan S; costs (|R|/(B - 2)) * |S| = 10*500 I/Os
  - Total = |R| + (|R|/(B - 2)) * |S| = 1000 + 10*500.
- With S as outer (|S| = 500):
  - Scanning S costs 500 IO's (done in 5 blocks)
  - Per block of S, we scan R; costs (|S|/(B - 2)) * |R| = 5*1000 IO's
  - Total = |S| + (|S|/(B - 2)) * |R| = 500 + 5*1000.

Summary of Nested-Loop Join

AALBORG UNIVERSITY

- Simple nested-loop join
  - Cost = $|R| + (p_r * |R|) * |S|$ (R as outer)
  - Cost = $|S| + (p_s * |S|) * |R|$ (S as outer)
- Page-oriented nested-loop join
  - Cost = $|R| + |R| * |S|$ (R as outer)
  - Cost = $|S| + |S| * |R|$ (S as outer)
- Block nested-loop join
  - Cost = $|R| + (|R|/(B - 2)) * |S|$ (R as outer)
  - Cost = $|S| + (|S|/(B - 2)) * |R|$ (S as outer)
- $|R|$ is the page number of relation R, p_r is the tuple number in each page of R
- $|S|$ is the page number of relation S, p_s is the tuple number in each page of S
- B is the page number of memory

Exercise 2

AALBORG UNIVERSITY

- Let relations r_1 and r_2 have the following properties: r_1 has 40,000 tuples, r_2 has 50,000 tuples, 40 tuples of r_1 fit on one page, and 25 tuples of r_2 fit on one page.
- Assume B pages of memory and B = 100 + 2.

- Q: Estimate the I/O cost required using each of the following join strategies for $r_1 \bowtie r_2$:
  1) Simple nested-loop join.
  2) Page-oriented nested-loop join.
  3) Block nested-loop join.

Exercise 2--Solution

AALBORG UNIVERSITY

- Let relations r_1 and r_2 have the following properties: r_1 has 40,000 tuples, r_2 has 50,000 tuples, 40 tuples of r_1 fit on one page, and 25 tuples of r_2 fit on one page.
- Assume B pages of memory and B = 100 + 2.
- Q: Estimate the I/O cost required using each of the following join strategies for $r_1 \bowtie r_2$:

  1) Simple nested-loop join.

  $p_1 = 40$, $p_2 = 25$

  $N_1 = r_1/p_1 = 40{,}000/40 = 1000$, $N_2 = r_2/p_2 = 50{,}000/25 = 2000$

  Cost 1 = $|N_1| + p_1 * |N_1| * |N_2| = 1000 + 40{,}000 * 2000$ (r_1 is the outer)

  Cost 2 = $|N_2| + p_2 * |N_2| * |N_1| = 2000 + 50{,}000 * 1000$ (r_2 is the outer)

Exercise 2--Solution

- Let relations r_1 and r_2 have the following properties: r_1 has 40,000 tuples, r_2 has 50,000 tuples, 40 tuples of r_1 fit on one page, and 25 tuples of r_2 fit on one page.
- Assume B pages of memory and B = 100 + 2.
- Q: Estimate the I/O cost required using each of the following join strategies for $r_1 \bowtie r_2$:

  2) Page-oriented nested-loop join.

  $p_1 = 40$, $p_2 = 25$

  $N_1 = r_1/p_1 = 40{,}000/40 = 1000$, $N_2 = r_2/p_2 = 50{,}000/25 = 2000$

  Cost 1 = $|N_1| + |N_1| * |N_2| = 1000 + 1000 * 2000$ (r_1 is the outer)

  Cost 2 = $|N_2| + |N_2| * |N_1| = 2000 + 2000 * 1000$ (r_2 is the outer)

Exercise 2--Solution

- Let relations r_1 and r_2 have the following properties: r_1 has 40,000 tuples, r_2 has 50,000 tuples, 40 tuples of r_1 fit on one page, and 25 tuples of r_2 fit on one page.
- Assume B pages of memory and B = 100 + 2.
- Q: Estimate the I/O cost required using each of the following join strategies for $r_1 \bowtie r_2$:

  3) Block nested-loop join.

  $p_1 = 40$, $p_2 = 25$

  $N_1 = r_1/p_1 = 40{,}000/40 = 1000$, $N_2 = r_2/p_2 = 50{,}000/25 = 2000$

  Cost 1 = $|N_1| + (|N_1|/(B-2)) * |N_2|$

  $= 1000 + (1000/100) * 2000$ (r_1 is the outer)

  Cost 2 = $|N_2| + (|N_2|/(B-2)) * |N_1|$

  $= 2000 + (2000/100) * 1000$ (r_2 is the outer)

Sort-Merge Join

```
select t.course_id
from instructor i, teaches t
where i.name = 'Brandt' and t.ID = i.ID;
```

- 1. Sort *instructor i* on join attribute
- 2. Sort *teaches t* on join attribute
- 3. Scan sorted-I and sorted-t in tandem, to find matches

ID	*name*
10101	Srinivasan
12121	Wu
15151	Mozart
83821	Brandt
98345	Kim

instructor

$\bowtie$

ID	*course_id*
10101	CS-101
10101	CS-122
15151	MU-199
15151	MU-201
76766	BIO-101
83821	CS-190
83821	CS-221

teaches

$\rightarrow$

ID	*name*	*course_id*

Sort-Merge Join

AALBORG UNIVERSITY

select *t.course_id*
from *instructor i, teaches t*
where *t.name = 'Brandt'* **and** *t.ID = i.ID;*

- 1. Sort *instructor i* on join attribute
- 2. Sort *teaches t* on join attribute
- 3. Scan sorted-i and sorted-t in tandem, to find matches

ID	*name*
10101	Srinivasan
12121	Wu
15151	Mozart
83821	Brandt
98345	Kim

instructor

$\bowtie$

ID	*course_id*
10101	CS-101
10101	CS-122
15151	MU-199
15151	MU-201
76766	BIO-101
83821	CS-190
83821	CS-221

teaches

$\rightarrow$

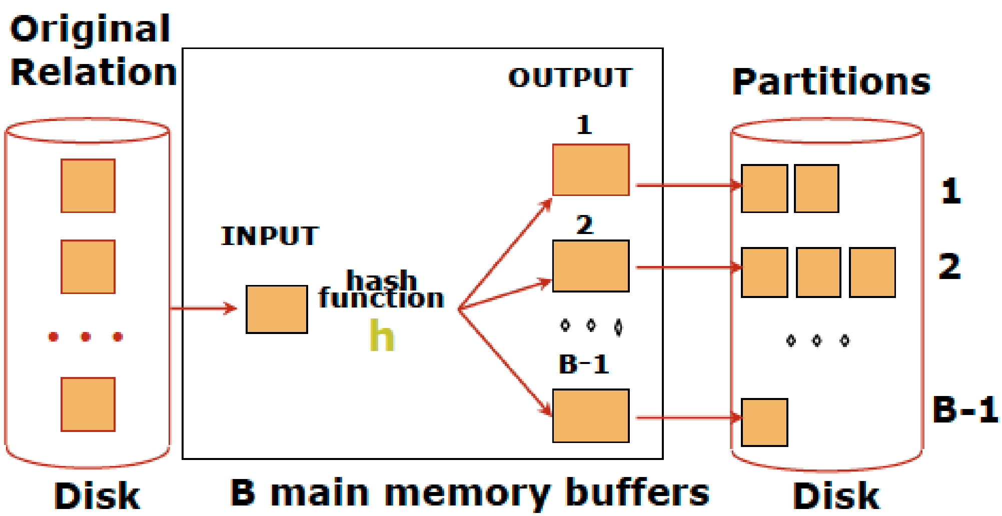
ID	*name*	*course_id*
10101	Srinivasan	CS-101
10101	Srinivasan	CS-122
15151	Mozart	MU-199
15151	Mozart	MU-201
83821	Brandt	CS-190
83821	Brandt	CS-221

Cost of Sort-Merge Join

- Cost: Sort R + Sort S + (|R|+|S|)
- An important refinement:
  - If it has enough memory (|R|/B + |S|/B < B), can do the join during that the pass of sort :
  - Cost = 3(|R| + |S|)
- Sort-merge join an especially good choice if:
  - one or both inputs are already sorted on join attribute(s)
  - output is required to be sorted on join attribute(s)

Hash Join

- Partition both relations using hash function: R tuples in partition i will only match S tuples in partition i.
- R join S = (R_1 join S_1) U (R_2 join S_2) U … U (R_{B-1} join S_{B-1})



Hash Join

ID	*name*
10101	Srinivasan
12121	Wu
15151	Mozart

instructor$_1$

$\bowtie$

ID	*course_id*
10101	CS-101
10101	CS-122
15151	MU-199
15151	MU-201

teaches$_1$

$\rightarrow$

ID	*name*	*course_id*
10101	Srinivasan	CS-101
10101	Srinivasan	CS-122
15151	Mozart	MU-199
15151	Mozart	MU-201

result$_1$

ID	*name*
83821	Brandt
98345	Kim

instructor$_2$

$\bowtie$

ID	*course_id*
76766	BIO-101
83821	CS-190
83821	CS-221

teaches$_2$

$\rightarrow$

ID	*name*	*course_id*
83821	Brandt	CS-190
83821	Brandt	CS-221

result$_2$

result = result$_1$ U result$_2$

Hash Join

- Partition both relations using hash function: R tuples in partition i will only match S tuples in partition
- Partitioning phase:
  - read + write both relations
  - 2(|R|+|S|) I/Os
- Matching phase:
  - read both relations
  - |R|+|S| I/Os
- 2-pass hash join cost = 3(|R|+|S|)

Exercise 3

- Let relations r_1 and r_2 have the following properties: r_1 has 40,000 tuples, r_2 has 50,000 tuples, 40 tuples of r_1 fit on one page, and 25 tuples of r_2 fit on one page.
- Assume B pages of memory and B <= 500.

- Q: Estimate the I/O cost required using each of the following join strategies for $r_1 \bowtie r_2$:
  1) Sort-merge join.
  2) Hash join.

Exercise 3--Solution

AALBORG UNIVERSITY

- Let relations r_1 and r_2 have the following properties: r_1 has 40,000 tuples, r_2 has 50,000 tuples, 40 tuples of r_1 fit on one page, and 25 tuples of r_2 fit on one page.
- Assume B pages of memory and B <= 500.
- Q: Estimate the I/O cost required using each of the following join strategies for $r_1 \bowtie r_2$:

  1) Sort-merge join.

  $p_1 = 40$, $p_2 = 25$

  $N_1 = r_1/p_1 = 40{,}000/40 = 1000$, $N_2 = r_2/p_2 = 50{,}000/25 = 2000$

  If $N_1/B + N_2/B < B$

  $Cost = 3(N_1 + N_2) = 3 * 3000 = 9000$

Exercise 3--Solution

- Let relations r_1 and r_2 have the following properties: r_1 has 40,000 tuples, r_2 has 50,000 tuples, 40 tuples of r_1 fit on one page, and 25 tuples of r_2 fit on one page.
- Assume B pages of memory and B <= 500.
- Q: Estimate the I/O cost required using each of the following join strategies for $r_1 \bowtie r_2$:

  1) Sort-merge join.

Otherwise,

Sort r_1: $cost1 = 2N_1 \times (\lceil \log_{B-1} \lceil N_1/B \rceil \rceil + 1) = 2000 \times (\lceil \log_{B-1} \lceil 1000/B \rceil \rceil + 1)$

Sort r_2: $cost2 = 2N_2 \times (\lceil \log_{B-1} \lceil N_2/B \rceil \rceil + 1) = 4000 \times (\lceil \log_{B-1} \lceil 2000/B \rceil \rceil + 1)$

$Cost = cost1 + cost2 + N_1 + N_2 = 3000 + cost1 + cost2$

Exercise 3--Solution

- Let relations r_1 and r_2 have the following properties: r_1 has 40,000 tuples, r_2 has 50,000 tuples, 40 tuples of r_1 fit on one page, and 25 tuples of r_2 fit on one page.
- Assume B pages of memory and B <= 500.
- Q: Estimate the I/O cost required using each of the following join strategies for $r_1 \bowtie r_2$:

  2) Hash join.

  $p_1 = 40$, $p_2 = 25$

  $N_1 = r_1/p_1 = 40{,}000/40 = 1000$, $N_2 = r_2/p_2 = 50{,}000/25 = 2000$

  Cost $= 3(N_1 + N_2) = 3 * 3000 = 9000$

Agenda

- Basic Steps in Query Processing
- Sorting
- Join Strategies
- Cost-based Query Optimization
- Logical Query Optimization

Measures of Query Costs

- Total elapsed time for answering a query (response time)
- Many factors contribute to response time
  - Disk access
  - CPU costs
  - Network communication
  - Query load
  - Parallel processing
- Disk access most dominant
  - Block access time: seek time, rotation time
  - Transfer time

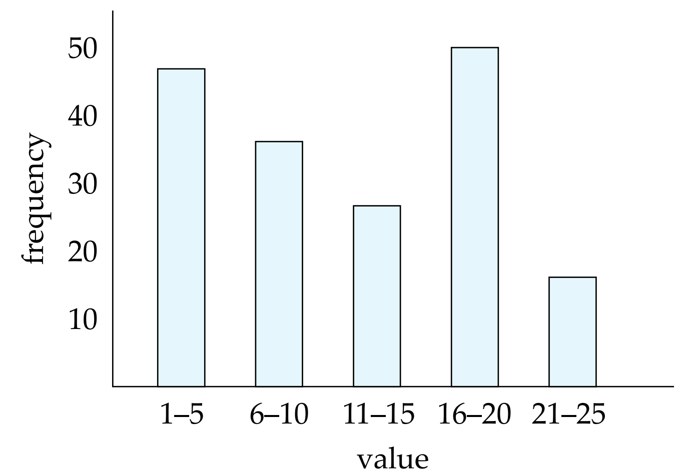
Statistical Information

- n_r: number of tuples in a relation *r.*
- b_r: number of blocks containing tuples of *r.*
- f_r: blocking factor of *r* — i.e., the number of tuples of *r* that fit into one block.
- *V(A, r):* number of distinct values that appear in *r* for attribute *A;* same as the size of $\prod_A(r)$.
- If tuples of *r* are stored together physically in a file, then:

$$b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$$

Histograms

- Histogram on attribute *age* of relation *person*
- **Equi-width** histograms
- **Equi-depth** histograms break up range such that each range has (approximately) the same number of tuples
- Many databases also store *n* **most-frequent values** and their counts
  - Histogram is built on remaining values only



Selection Size Estimation

- Let c denote the estimated number of tuples satisfying the condition.
- $\sigma_{A=v}(r)$
  - c = n_r / $V(A,r)$: number of records that will satisfy the selection
- $\sigma_{A \leq v}(r)$ (case of $\sigma_{A \geq v}(r)$ is symmetric)
  - If min(A, r) and max(A, r) are available in catalog
    - c = 0 if v < min(A, r)
    - c = n_r, *if* v $\geq$ max(A, r)
    - c = $n_r . \dfrac{v - \min(A,r)}{\max(A,r) - \min(A,r)}$

Example

Given a relation $r(A, B, C)$ with n_r = 5000 and $V(B, r)$ = 100.

1) Estimate the size of the selection operation $\sigma_{B=3}(r)$.

c = $n_r/V(B, r)$ = 5000/100 = 50

2) Assume the range of values for an attribute A is [10, 60] and the values are uniformly distributed. Estimate the size of the selection operation $\sigma_{A<20}(r)$.

$$c = n_r . \frac{v - \min(A, r)}{\max(A, r) - \min(A, r)}$$

= 5000 * (20 - 10)/(60 - 10) = 5000 * 10/50 = 1000

Agenda

AALBORG UNIVERSITY

- Basic Steps in Query Processing
- Sorting
- Join Strategies
- Cost-based Query Optimization
- Logical Query Optimization

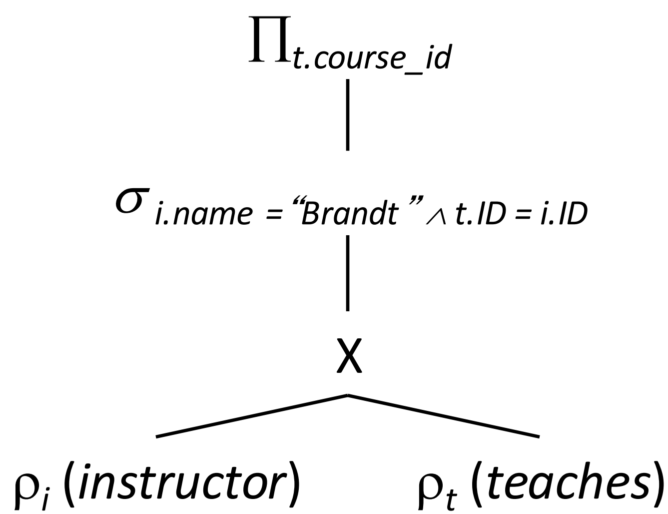
Query Optimization

- A relational algebra expression may have many equivalent expressions
  - E.g., $\sigma_{salary<75000}(\prod_{salary}(instructor))$ is equivalent to $\prod_{salary}(\sigma_{salary<75000}(instructor))$
- Annotated expression specifying detailed evaluation strategy is called an **evaluation-plan**
- It is not the task of the user to write queries "efficiently", it is the task of the query optimizer to find a way to execute them efficiently!

Parsing a Query into An Initial Query Plan

- *instructor*(*ID, name, dept_name, salary*)
- *teaches*(*ID, course_id, semester, year*)

```
select t.course_id
from instructor i, teaches t
where i.name = 'Brandt' and t.ID = i.ID;
```

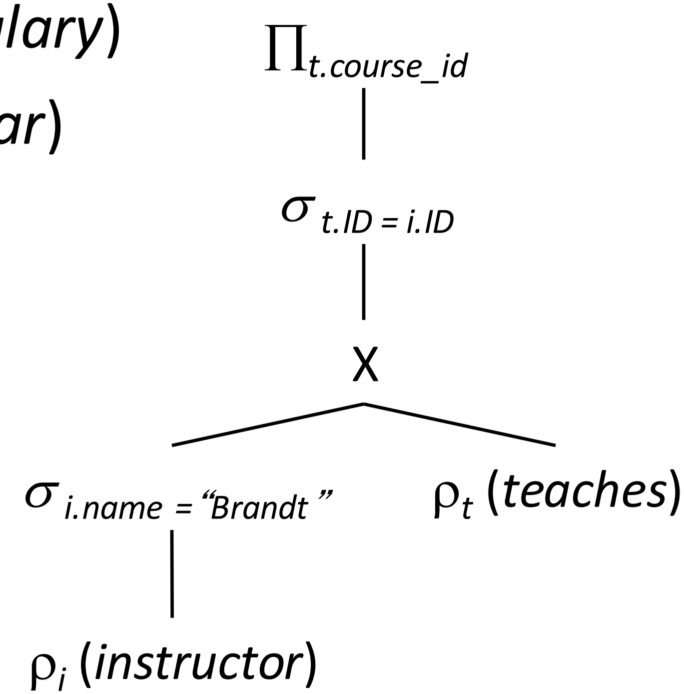


$$\Pi_{t.course\_id}(\sigma_{i.name\ =\ \text{“Brandt”} \wedge t.ID = i.ID}(\rho_i(instructor) \times \rho_t(teaches)))$$

Alternative Query Plan

- *instructor*(*ID, name, dept_name, salary*)
- *teaches*(*ID, course_id, semester, year*)

```
select t.course_id
from instructor i, teaches t
where i.name = 'Brandt' and t.ID = i.ID;
```



$$\Pi_{t.course\_id}(\sigma_{t.ID = i.ID}((\sigma_{i.name = \text{"Brandt"}}\rho_i(instructor)) \times \rho_t(teaches)))$$

Equivalence Rules

- 1. Conjunctive selection operations can be deconstructed into a sequence of individual selections.

$$\sigma_{\theta_1 \wedge \theta_2}(E) \equiv \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

- 2. Selection operations are commutative.

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) \equiv \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

- 3. Only the last in a sequence of projection operations is needed, the others can be omitted.

$$\prod_{L_1}(\prod_{L_2}(\ldots(\prod_{L_n}(E))\ldots)) \equiv \prod_{L_1}(E) \text{ where } L_1 \subseteq L_2 \ldots \subseteq L_n$$

Equivalence Rules (Cont.)

- 4. Selections can be combined with Cartesian products and theta joins.

$$\sigma_\theta (E_1 \times E_2) \equiv E_1 \bowtie_\theta E_2$$

- 5. Theta-join operations (and natural joins) are commutative.

$$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$$

Phases of Logical Query Optimization

- 1. Break up conjunctive selection predicates and push selections down
- 2. Introduce joins by combining selections and Cartesian products
- 3. Determine join order
  - Heuristic: execute joins with input from selections before executing other joins
- 4. Introduce and push down projections

Example

- For relations $r(A, B)$, $s(C, D)$ consider the following expression, please use LOGICAL QUERY OPTIMIZATION to optimize it.
  - $\Pi_A \sigma_{A=5 \wedge B<D}(r \times s)$

1. Break up selection predicate and pushdown
   - $\Pi_A \sigma_{B<D}(\sigma_{A=5}(r) \times s)$ $\quad\quad \sigma_{\theta_1 \wedge \theta_2}(E) \equiv \sigma_{\theta_1}(\sigma_{\theta_2}(E))$
2. Replace Cartesian Product
   - $\Pi_A(\sigma_{A=5}(r) \bowtie_{B<D} s)$ $\quad\quad \sigma_\theta(E_1 \times E_2) \equiv E_1 \bowtie_\theta E_2$
3. Projection Pushdown
   - $\Pi_A(\sigma_{A=5}(r) \bowtie_{B<D} \Pi_D(s))$

Summary

- Basic Steps in Query Processing
- Sorting
- Join Strategies
- Cost-based Query Optimization
- Logical Query Optimization

Next Lecture

- Transaction and Concurrency Control
  - Understanding the transaction concept
  - Understanding serializability
  - Understand and use lock-based concurrency control
  - Understand and use two-phase locking