Query Evaluation Techniques for large DB

Original slides by Daniela Stasa

Modified by Rachel Pottinger

Discussion: What do you view as the goal of a survey paper (adapted from Canvas)

- Who should a survey paper be written for?
- What sort of things should a good survey paper cover? E.g., details of the topic being surveyed, how the topic is used.
- What balance of the above issues makes sense?

Purpose

- To survey efficient algorithms and software architectures of *query execution engines* for executing *complex queries* over *large databases*

Query execution engine

- What is it?
  - Collection of query execution operators and mechanisms for operator communication and synchronization
  - Query execution engine defines the space of possible plans that can be chosen by query optimizer.

Some of the techniques discussed

- Algorithms and their execution costs
- Sorting versus hashing
- Parallelism
- Resource allocation
- Scheduling issues
- Performance-enhancement techniques
- And more …

Some notes

- On the context
  - While many of the techniques were developed for relational database systems most are applicable to any data mode that allows queries over sets and lists.
- Type of queries
  - Discusses only read-only queries but mostly applicable to updates.

Physical Algebra

- Taken as a whole, the query processing algorithms form an algebra which we call physical algebra of a database system

Physical vs. Logical Algebra (1/2)

- Equivalent but different
- Logical algebra: related to data model and defines what queries can be expressed in data model
- Physical algebra: system specific

Physical vs. Logical Algebra (2/2)

- Specific algorithms and therefore cost functions are associated only with physical operators not logical algebra operators
- Mapping logical to physical non–trivial:
  - It involves algorithm choices
  - Logical and physical operators not directly mapped
  - Some operators in physical algebra may implement multiple logical operators
  - etc

Observations

- Entire query plan executed within a single process
- Operators produce an item at a time on request
- Items never wait in a temporary file or buffer (pipelining)
- Efficient in time-space-product memory cost
- Iterators can schedule any type of trees including bushy trees
- No operator affected by the complexity of the whole plan

Discussion (pairs)

- There are many issues that could be covered by either the OS or the database. Break into groups and discuss some of these issues. For each issue, what are the pros and the cons of handling it in the database?

Sorting &Hashing

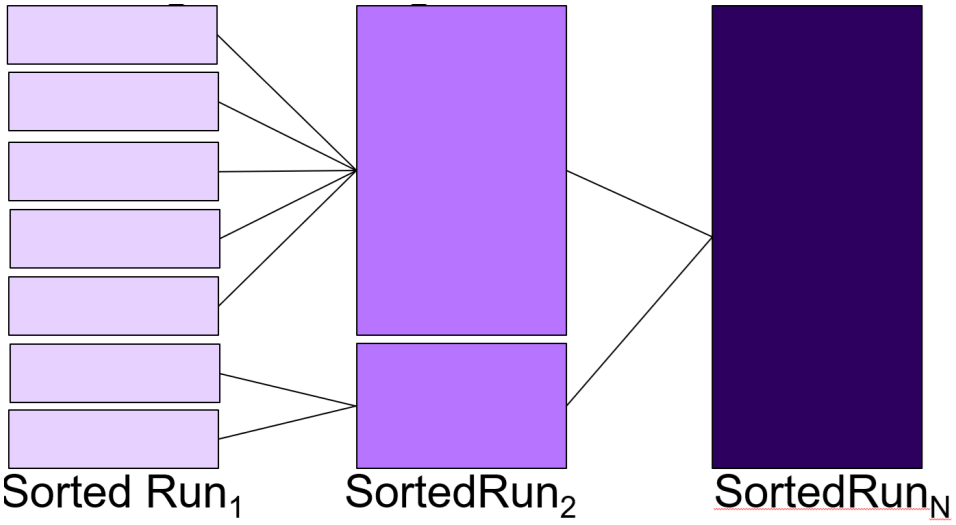
- The purpose of many query-processing algorithms is to perform some kind of matching,
  - i.e., bringing items that are “alike” together and performing some operation on them.
- There are two basic approaches used for this purpose:
  - sorting
  - and hashing.
- These are the basis for many join algorithms

More on Sorting

- For sorting large data sets there are two distinct sub-algorithm :
  - One for sorting within main memory
  - One for managing subsets of the data set on the disk.
- For practical reasons, e.g., ensuring that a run fits into main memory, the disk management algorithm typically uses physical dividing and logical combining (merging).
- A point of practical importance is the fan-in or degree of merging, but this is a parameter rather than a defining algorithm property.

Overall Sort Merge Plan

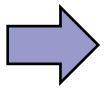
- Sort things in amounts that can fit into memory at a time (standard sorting algorithms, e.g., quick sort)
  - This creates a **sorted run**
- Merge the sorted runs into larger runs



Extremely tiny external Mergesort example: Setup

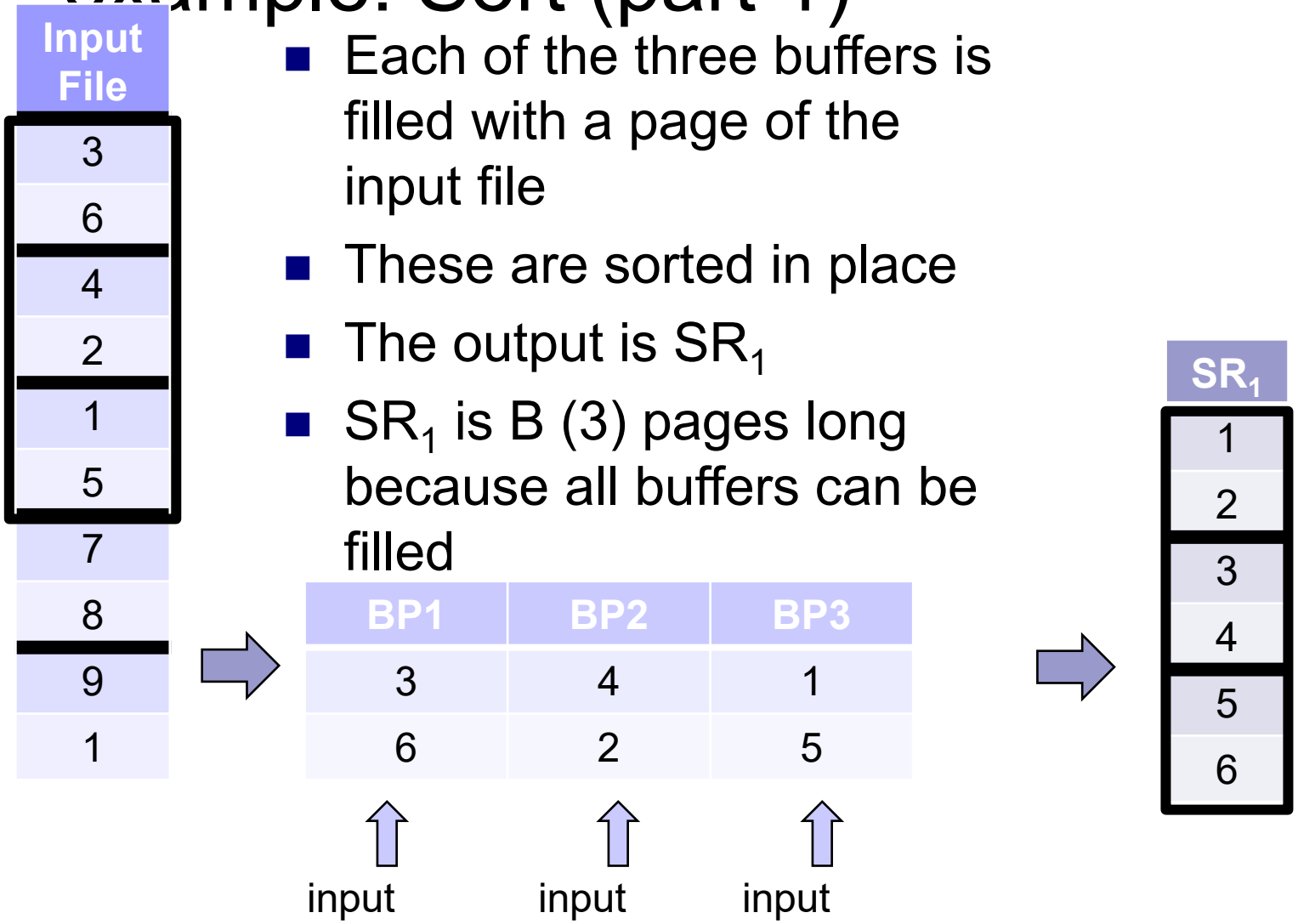
- Inputs:
  - 3 buffers
  - Each page can hold two integers
  - ←This unsorted input file

Input File
3
6
4
2
1
5
7
8
1
9



BP1	BP2	BP3

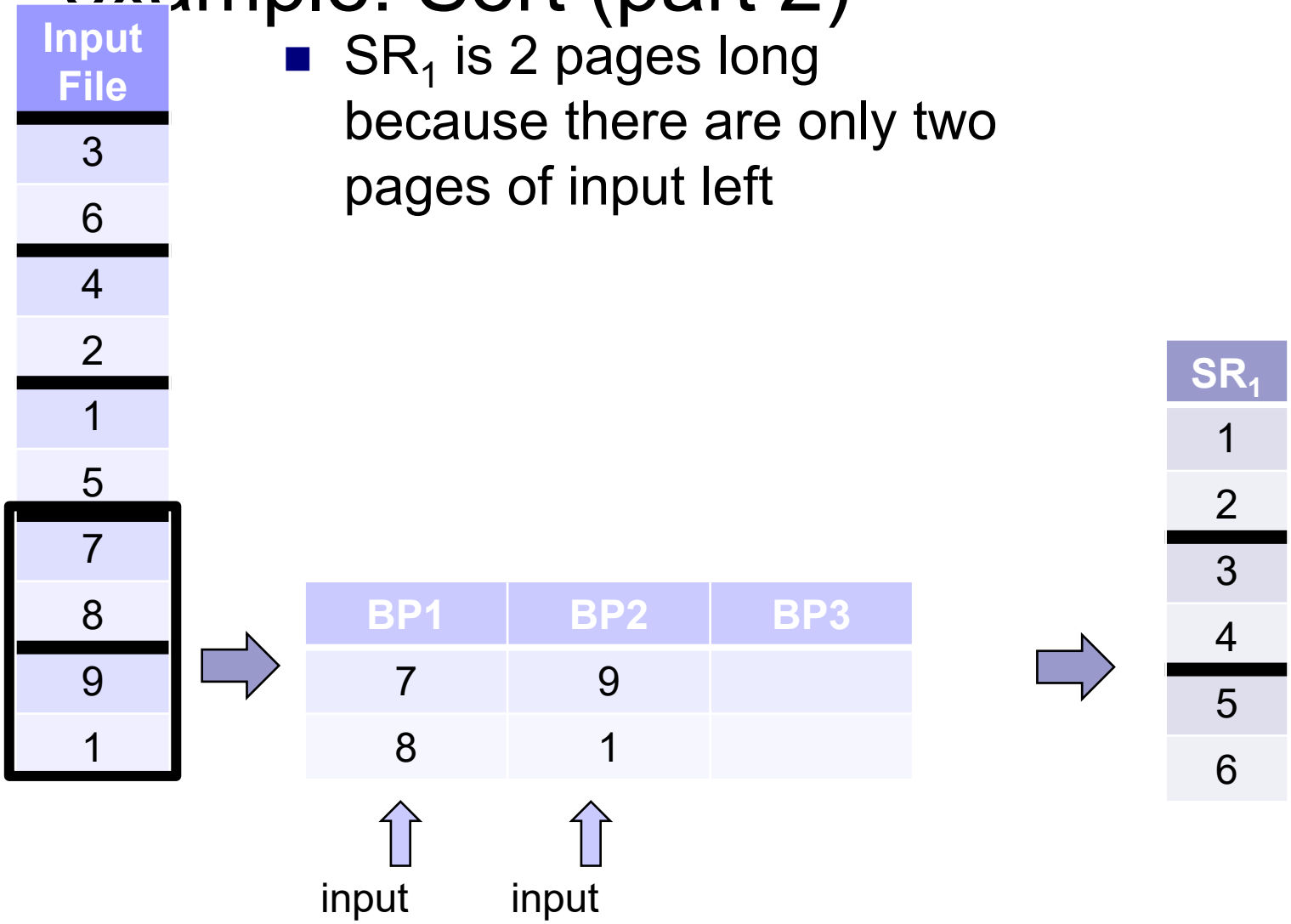
Extremely tiny external Mergesort example: Sort (part 1)



- Each of the three buffers is filled with a page of the input file
- These are sorted in place
- The output is SR_1
- SR_1 is B (3) pages long because all buffers can be filled

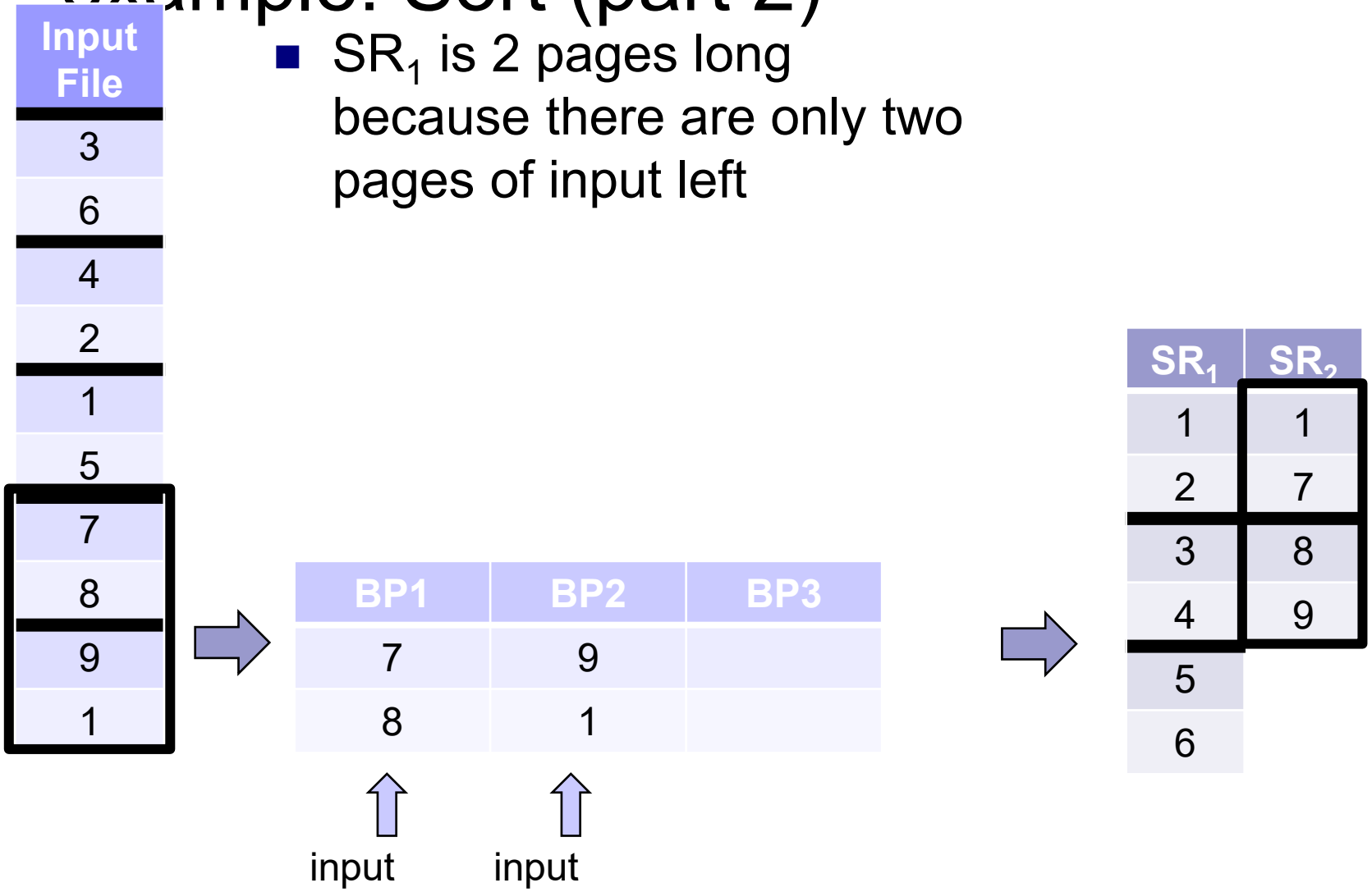
Extremely tiny external Mergesort example: Sort (part 2)

- SR_1 is 2 pages long because there are only two pages of input left

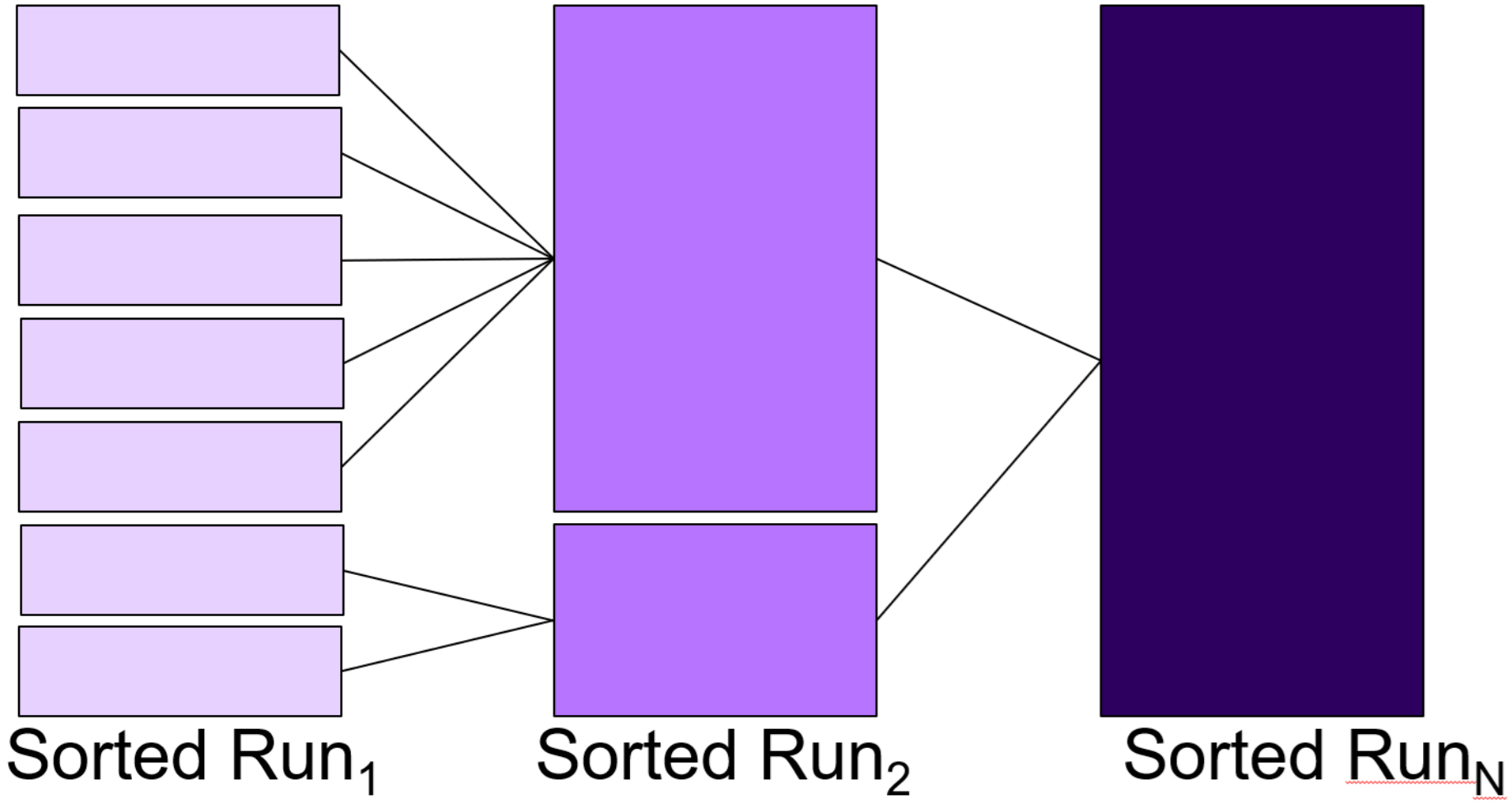


Extremely tiny external Mergesort example: Sort (part 2)

- SR_1 is 2 pages long because there are only two pages of input left



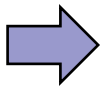
Merging Phase



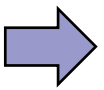
Extremely tiny external Mergesort example: Pass 1: Merge 1 (part 1)

- Take as input each of the sorted runs
- BP3 is kept for output
- BP3 fills up with the first two entries: 1, 1, which are then written to SR'

SR_1	SR_2
1	1
2	7
3	8
4	9
5	
6	



BP1	BP2	BP3
1	1	1
2	7	1

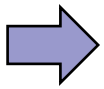


SR'
1
1

Extremely tiny external Mergesort example: Pass 1: Merge 1 (part 2)

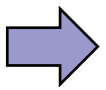
- The output buffer is now empty again.
- But we have used up the first page of SR_1, so the second page is loaded into BP1.
- The output buffer fills up with 2,3, and is written to SR'

SR_1	SR_2
1	1
2	7
3	8
4	9
5	
6	



BP1	BP2	BP3
~~1~~	~~1~~	2
2	7	

input input output

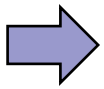


SR'
1
1

Extremely tiny external Mergesort example: Pass 1: Merge 1 (part 3)

- At the same time, the last page of SR_1 is read into BP1
- Again, this fills up BP3 and is written to SR'

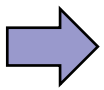
SR_1	SR_2
1	1
2	7
3	8
4	9
5	
6	



BP1	BP2	BP3
3	~~1~~	2
4	7	3



input input output

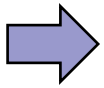


SR'
1
1
2
3

Extremely tiny external Mergesort example: Pass 1: Merge 1 (part 4)

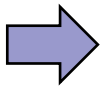
- At this point we are done reading SR_1. The contents of BP2 are appended to SR'

SR_1	SR_2
1	1
2	7
3	8
4	9
5	
6	



BP1	BP2	BP3
~~3~~	~~1~~	4
4	7	

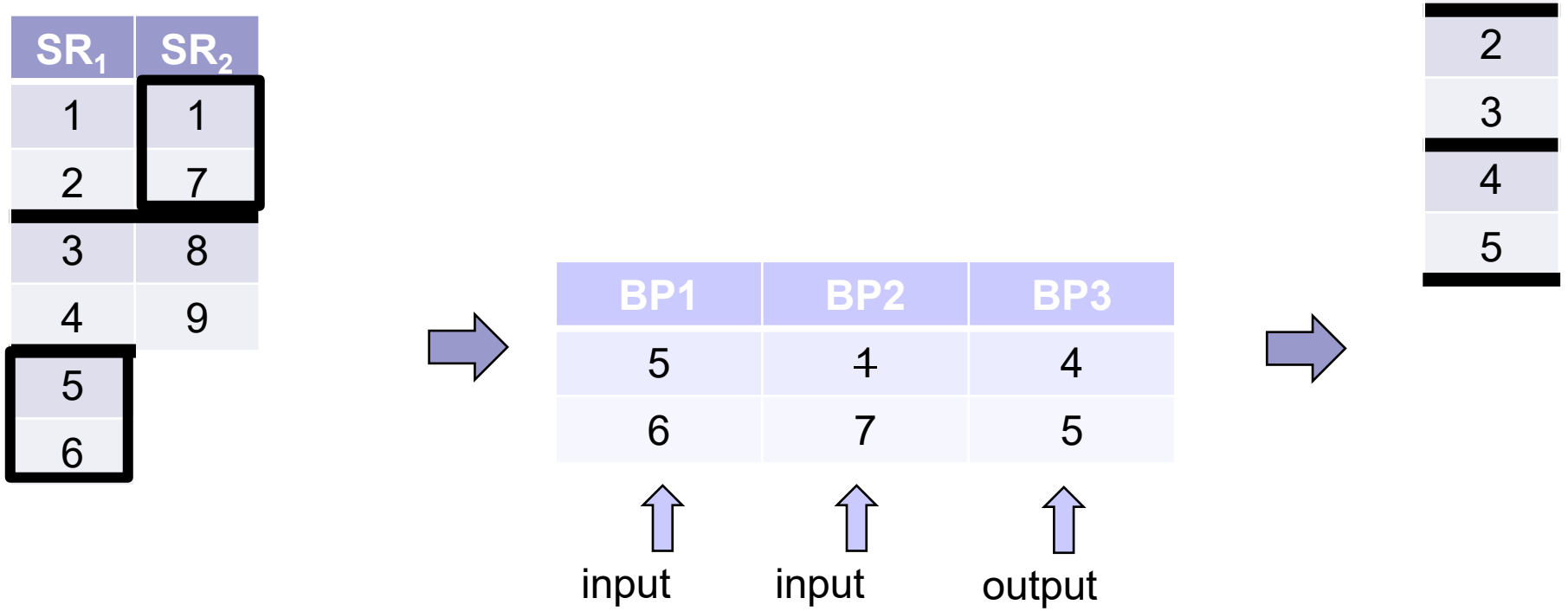
input input output



SR'
1
1
2
3

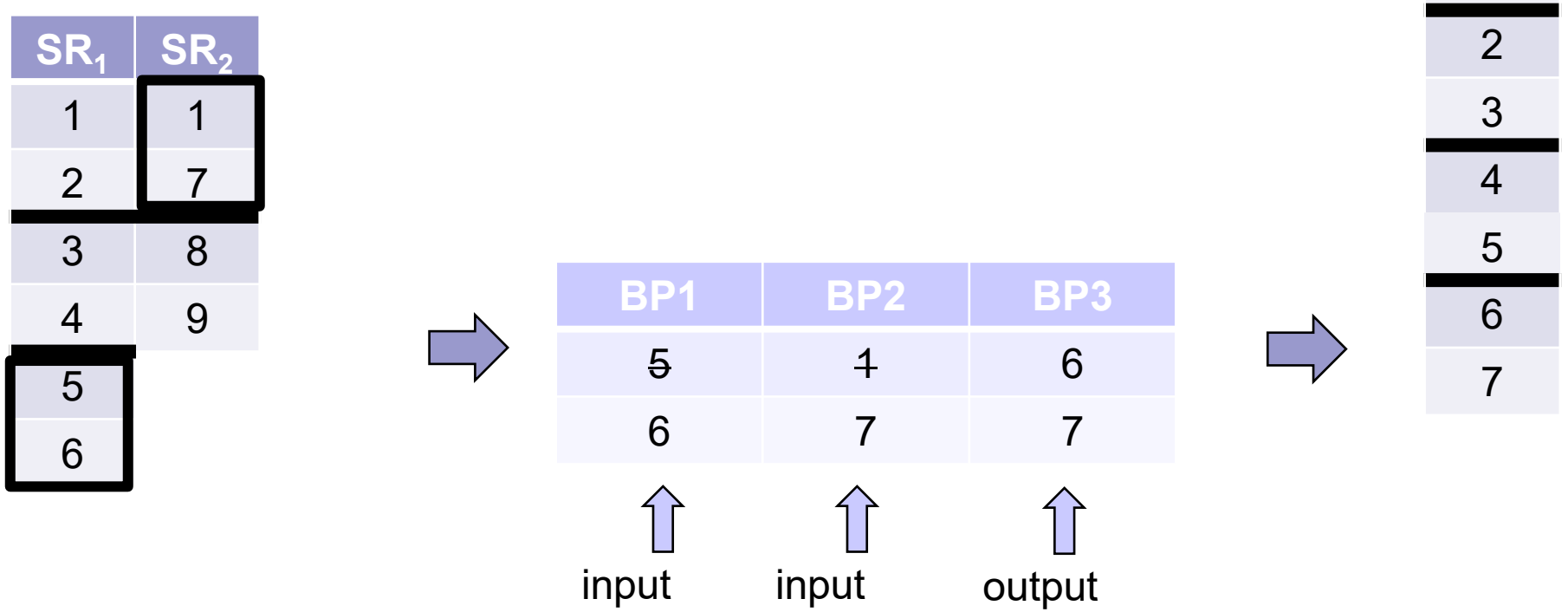
Extremely tiny external Mergesort example: Pass 1: Merge 1 (part 5)

- Finally, we read the last page of SR_2 into BP2, which are then appended to SR'



Extremely tiny external Mergesort example: Pass 1: Merge 1 (part 5)

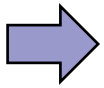
- Finally, we read the last page of SR_2 into BP2, which are then appended to SR'



Extremely tiny external Mergesort example: Pass 1: Merge 1 (part 5)

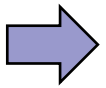
- Finally, we read the last page of SR_2 into BP2, which are then appended to SR'

SR_1	SR_2
1	1
2	7
3	8
4	9
5	
6	



BP1	BP2	BP3
	8	8
	9	9

input input output



SR'
1
1
2
3
4
5
6
7
8
9

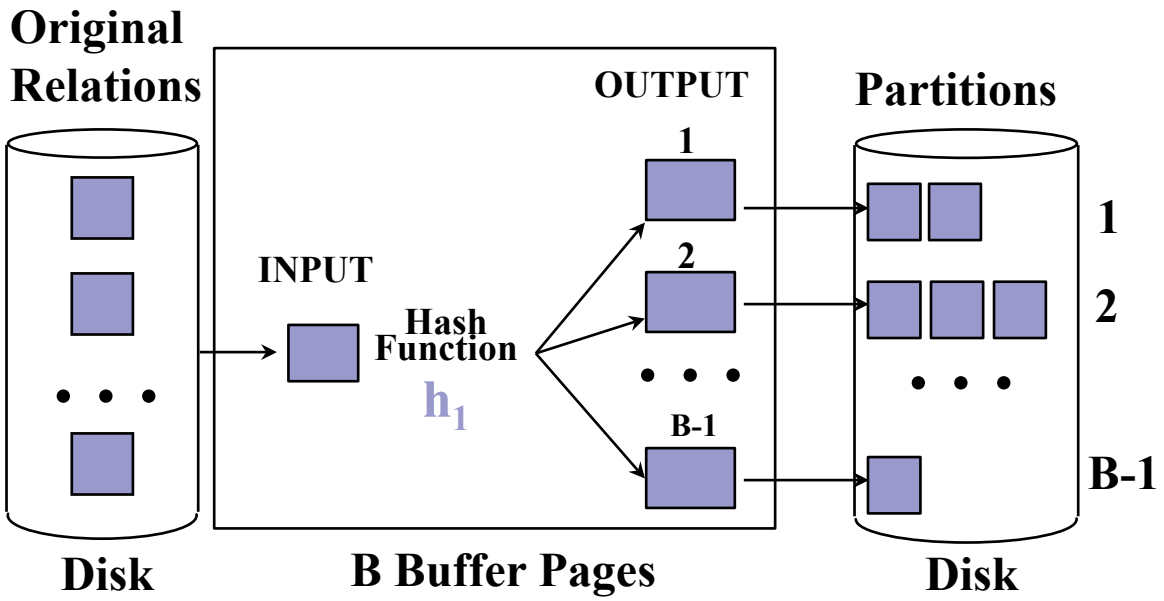
Hashing

- Alternative to sorting
- Expected complexity of hashing algorithms is O(N) rather than O(N log N) as for sorting.
- Hash-based query processing algorithms use an in-memory hash table of database objects to perform their matching task.

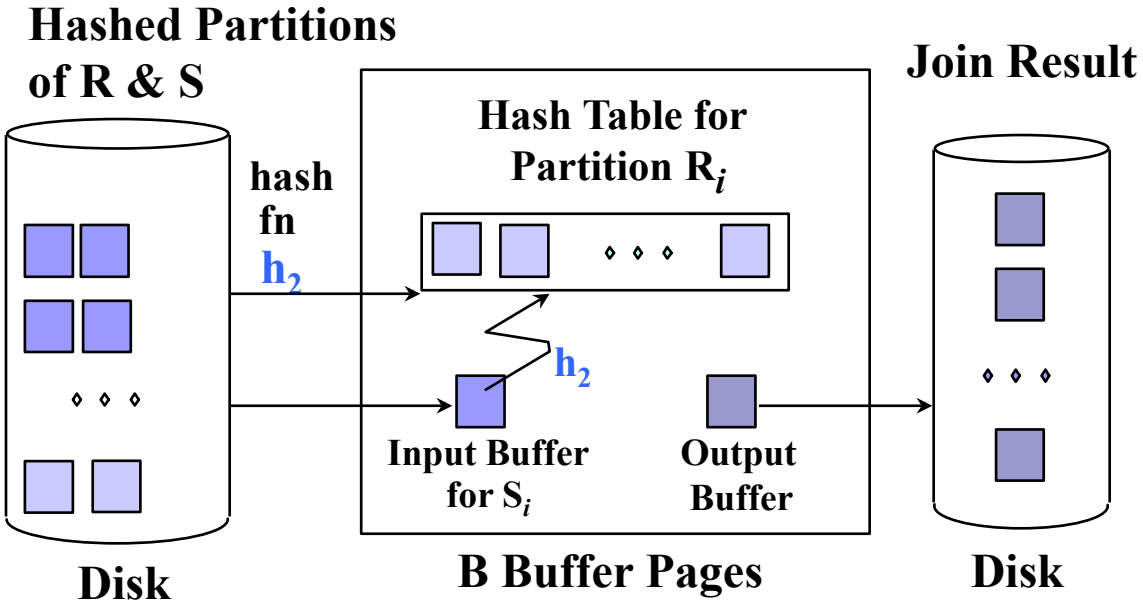
Hash Join : Partition Phase

Partition *both* relations using hash function h_1. During the join phase, the R tuples in partition *i* can only match tuples in S in partition *i*.

If partitions fit into memory, you're done. If not, hash again.

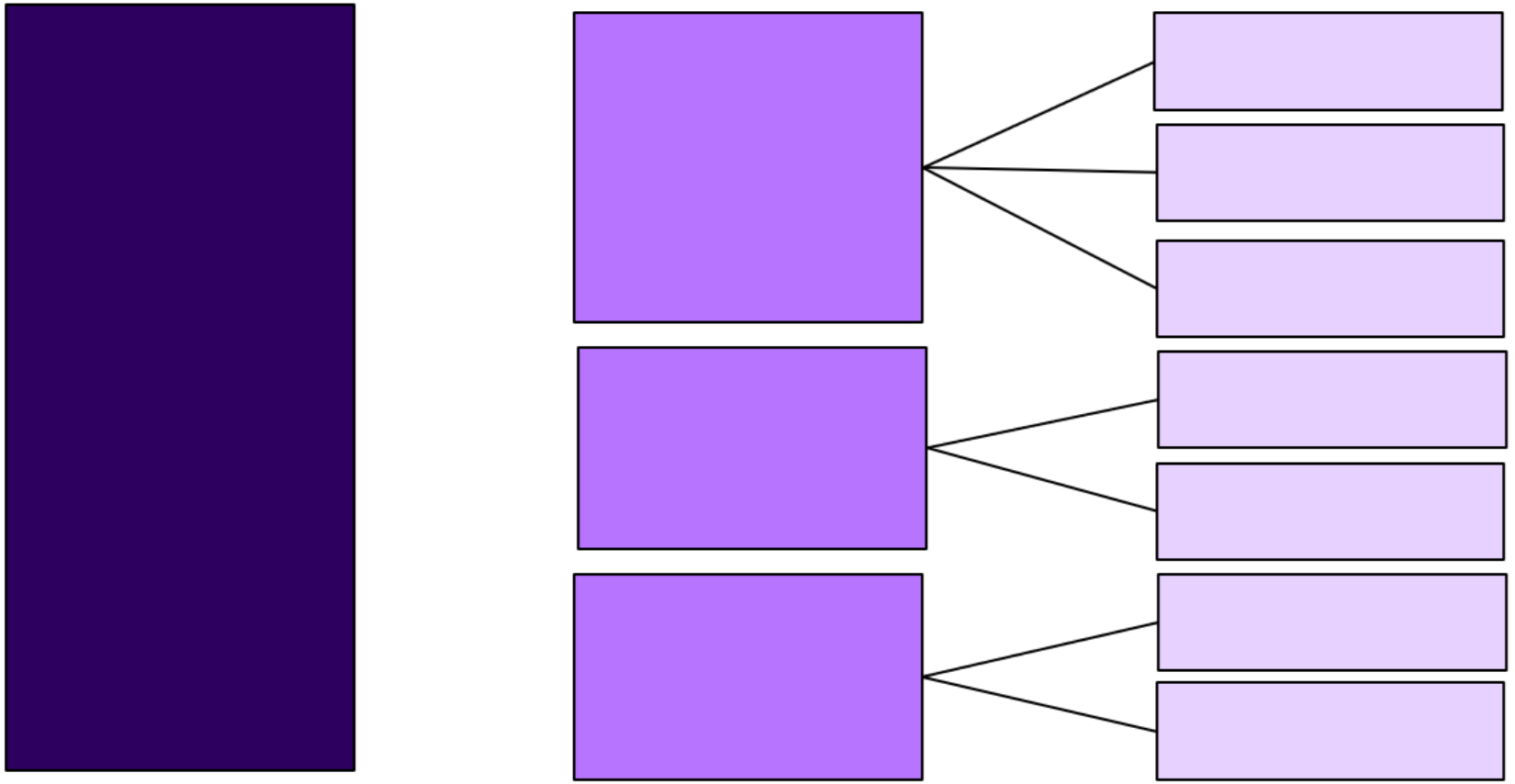


Hash Join: Join Phase



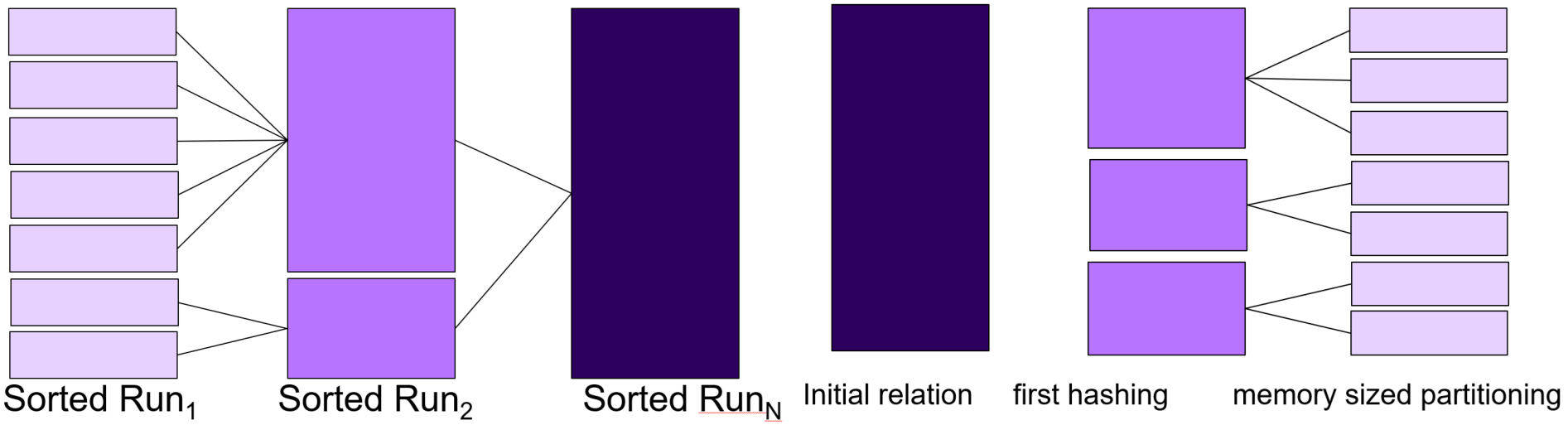
Read in a partition R_i of R, and hash it using h_2 ($\neq h_1$). Then, scan (and hash using h_2) matching partition S_i of S, searching for matches.

If a partition doesn't fit into memory, rehash with a different function



Initial relation first hashing memory sized partitioning

Duality of sorting and hashing



Sorting

Hashing

Discussion

- Does the number of generally used joins seem large or small to you? Why?
- Are you surprised by any of the joins that are used?

CONCLUSION:

The choice of Hash based or Sort based should be based on relative sizes of inputs and the danger of performance loss due to skewed data or hash value distribution.

Indices/indexes

- Goal:
  - To reduce the number of accesses to secondary storage
- How?
  - By employing search techniques in the form of indices (sometimes, also materialized views, but not in this paper)
  - Indices map key or attribute values to locator information with which database objects can be retrieved.

Some Index Structures:

- Clustered & Un-clustered
  - Clustered: order or organization of index entries determines order of items on disk.
- Sparse & Dense
  - Sparse: Indices do not contain an entry for each data item in the primary file, but only one entry for each page of the primary file;
  - Dense: there are same number of entries in index as there are items in primary file.
  - Non-clustering indices must always be dense

BINARY MATCHING OPERATIONS

- Relational join most prominent binary matching operation (others: intersection, union, etc)
- Set operations such as intersection and difference needed for any data model
- Most commercial db systems as of 1993 used only nested loops and merge-join. As per research done for SystemR, these two were supposed to be most efficient.
- SystemR researchers did not consider Hash join algorithms, which are today considered even better in performance.

NESTED-LOOPS JOIN ALGORITHMS: simple elegance

- For each item in one input, scan entire other input to find matches.
- Performance is really poor, because inner input is scanned often. (paper points this out)
- Tricks to improve performance include:
  - larger input should be the outer one.
  - if possible, use an index on the attribute to be matched in the inner input.
  - Inner input can be scanned once for each ‘page’ of outer input.

MERGE-JOIN ALGORITHMS

- Requires both inputs sorted on the join attribute
- Requires keeping track of interesting orderings
- Hybrid join (used by IBM for DB2), uses elements from index nested-loop joins and merge join, and techniques joining sorted lists on index leaf entries.

HASH JOIN ALGORITHMS

- Based on in-memory hash table on one input (smaller one, called ‘build input’), and probing this table using items from the other input (called ‘probe input’).
- Very fast if build input fits into memory, regardless of size of probe input.
- overflow avoidance methods needed for larger build inputs.
- both inputs partitioned using same partitioning function. Final join result formed by concatenating join results of pairs of partitioning files.
- Recursive partitioning may be used for both inputs
- More effective when the two input sizes are very different (smaller being the build input).

Discussion

- Who should write survey papers? Grad students? Senior researchers? Why? What are the benefits and disadvantages to having people from different groups write them?