Big Data Systems

Djellel Difallah

Spring 2023

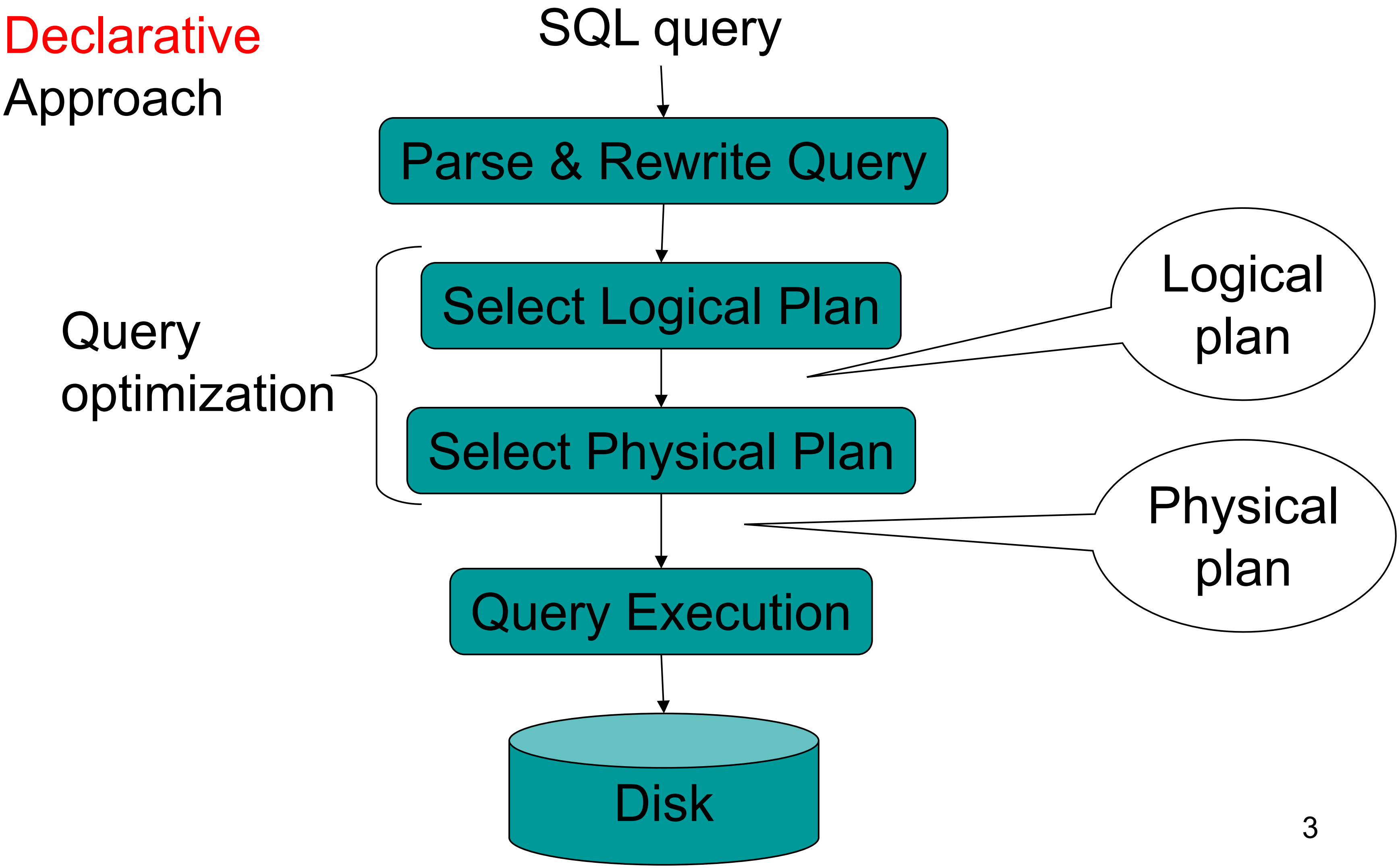
Lecture 3 – Query Execution

Outline

- **Steps involved in processing a query**
  - Logical query plan
  - Physical query plan
  - Query execution overview

- **Operator implementations**
  - One pass algorithms
  - Two-pass algorithms
  - Index-based algorithms

Query Evaluation Steps



Example Query

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
```

- Find the names of all suppliers in Seattle who supply part number 2

```
SELECT S.sname
FROM Supplier S, Supplies U
WHERE S.scity='Seattle' AND S.sstate='WA'
AND S.sno = U.sno
AND U.pno = 2;
```

Steps in Query Evaluation

- **Step 0: admission control**
  - User connects to the db with username, password
  - User sends query in text format
- **Step 1: Query parsing**
  - Parses query into an internal format
  - Performs various checks using catalog
    - Correctness, authorization, integrity constraints
- **Step 2: Query rewrite**
  - View rewriting, flattening, etc.

Continue with Query Evaluation

- **Step 3: Query optimization**
  – Find an efficient query plan for executing the query

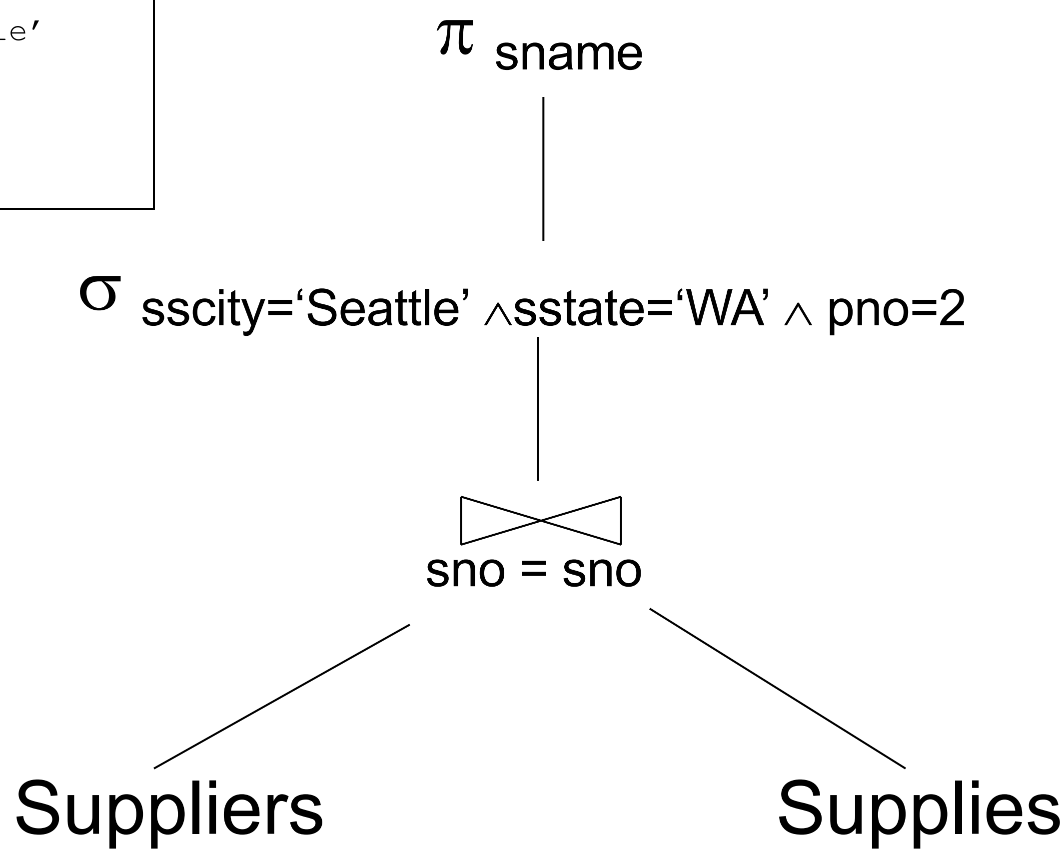
- A **query plan** is
  – **Logical query plan**: an extended relational algebra tree
  – **Physical query plan**: with additional annotations at each node
    - Access method to use for each relation
    - Implementation to use for each relational operator

Extended Algebra Operators

- Union $\cup$, intersection $\cap$, difference $-$
- Selection σ
- Projection π
- Join $\bowtie$
- Duplicate elimination δ
- Grouping and aggregation γ
- Sorting τ
- Rename ρ

Logical Query Plan

```
SELECT S.sname
FROM Supplier S, Supplies U
WHERE S.scity='Seattle'
AND S.sstate='WA'
AND S.sno = U.sno
AND U.pno = 2;
```

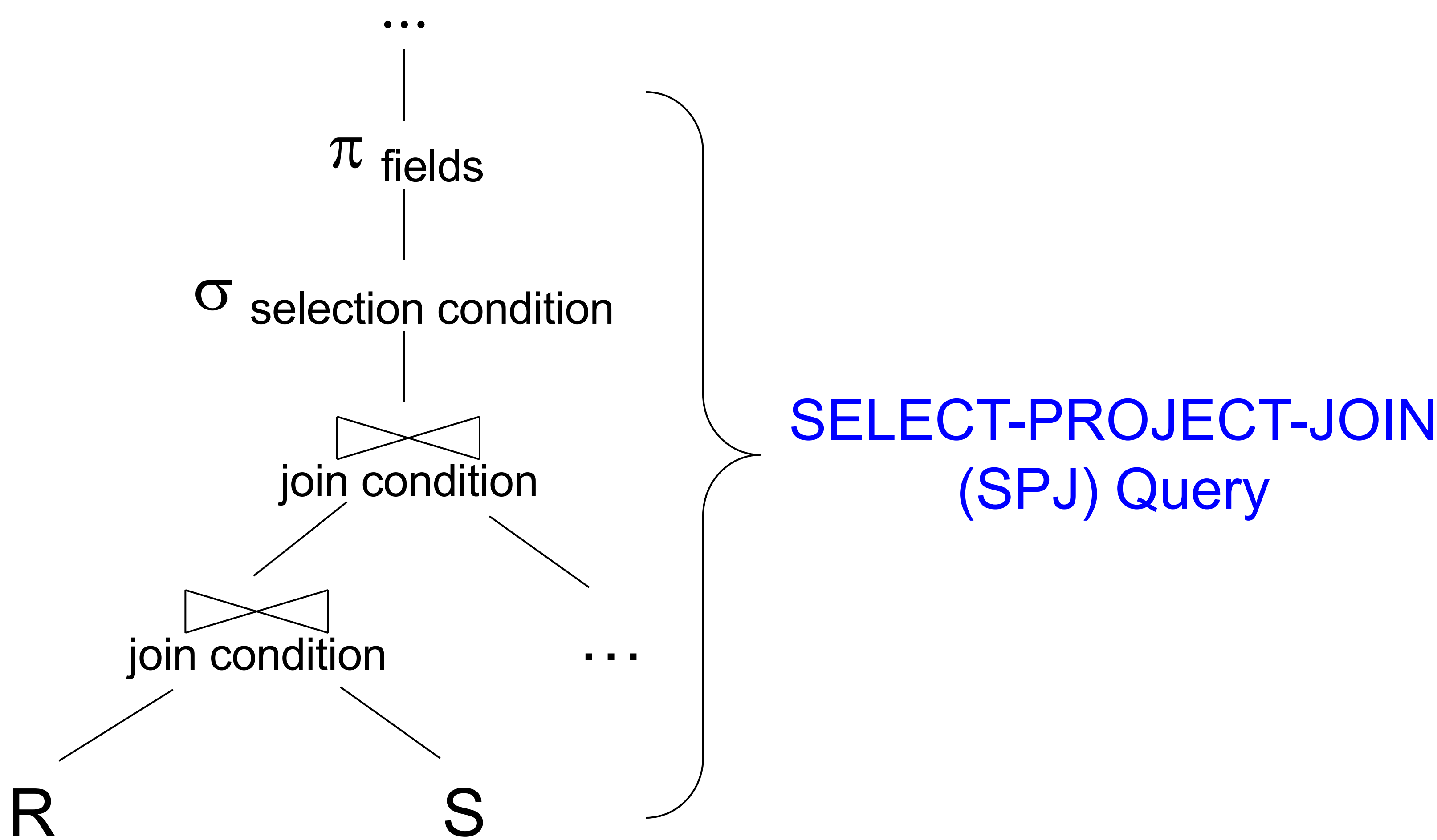


Query Block

- Most optimizers operate on individual query blocks

- A query block is a SQL query with **no nesting**
  - **Exactly one**
    - SELECT clause
    - FROM clause
  - **At most one**
    - WHERE clause
    - GROUP BY clause
    - HAVING clause

Typical Plan for Block (1/2)



Typical Plan For Block (2/2)

$\text{having}_{\text{condition}}$

$\gamma_{\text{fields, sum/count/min/max(fields)}}$

π_{fields}

$\sigma_{\text{selection condition}}$

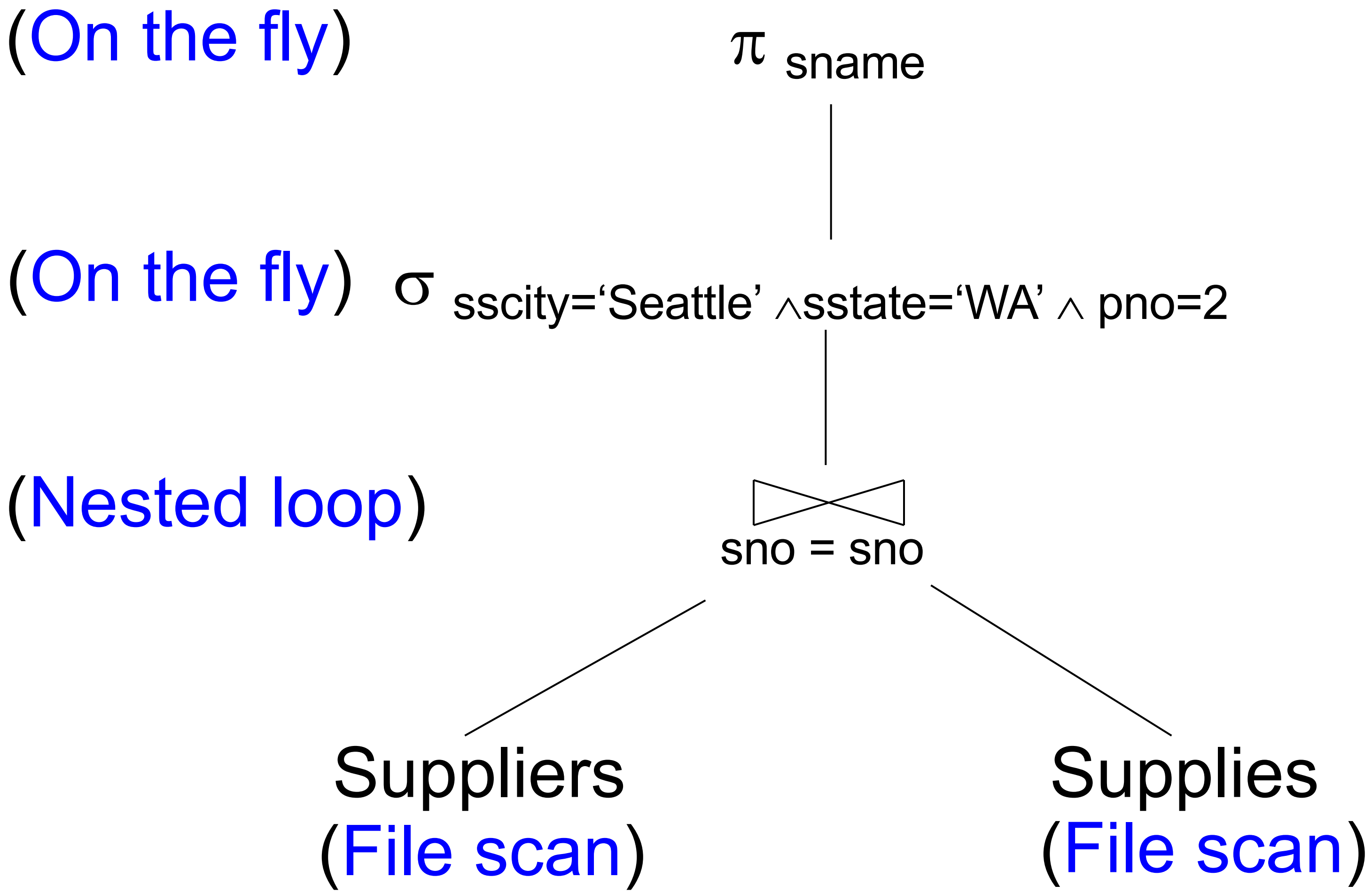
$\bowtie_{\text{join condition}}$

… …

Physical Query Plan

- Logical query plan with extra annotations
- **Access path selection** for each relation
  - Use a file scan or use an index
- **Implementation choice** for each operator
- **Scheduling decisions** for operators

Physical Query Plan



Final Step in Query Processing

- **Step 4: Query execution**
  - How to synchronize operators?
  - How to pass data between operators?

- Standard approach:
  - Record Iterator
  - Pipelined execution or Intermediate result materialization

Pipelined Execution

- Applies parent operator to tuples directly as they are produced by child operators

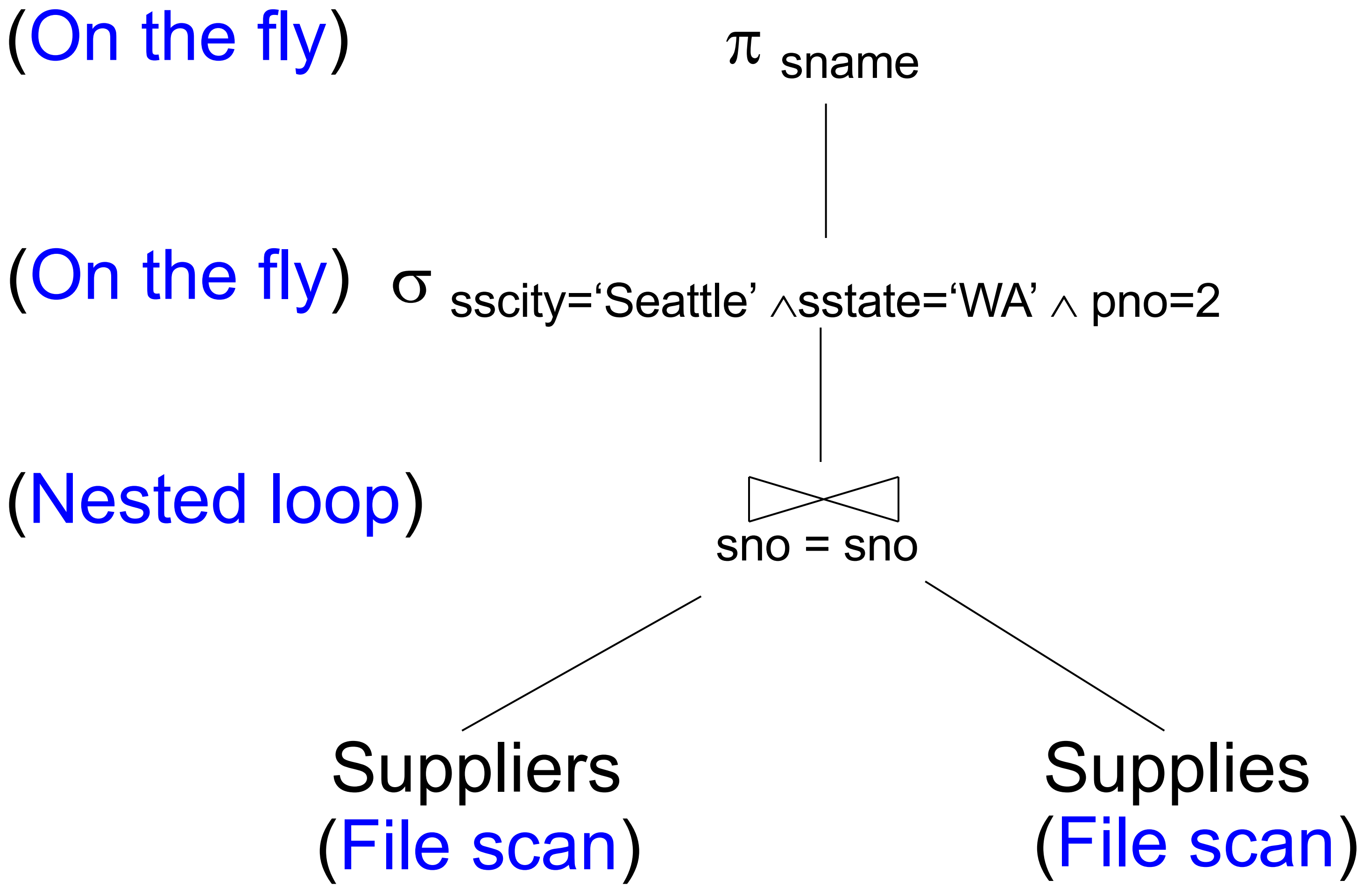
- Benefits
  - No operator synchronization issues
  - Saves cost of writing intermediate data to disk
  - Saves cost of reading intermediate data from disk
  - Good resource utilizations on single processor

- This approach is used whenever possible

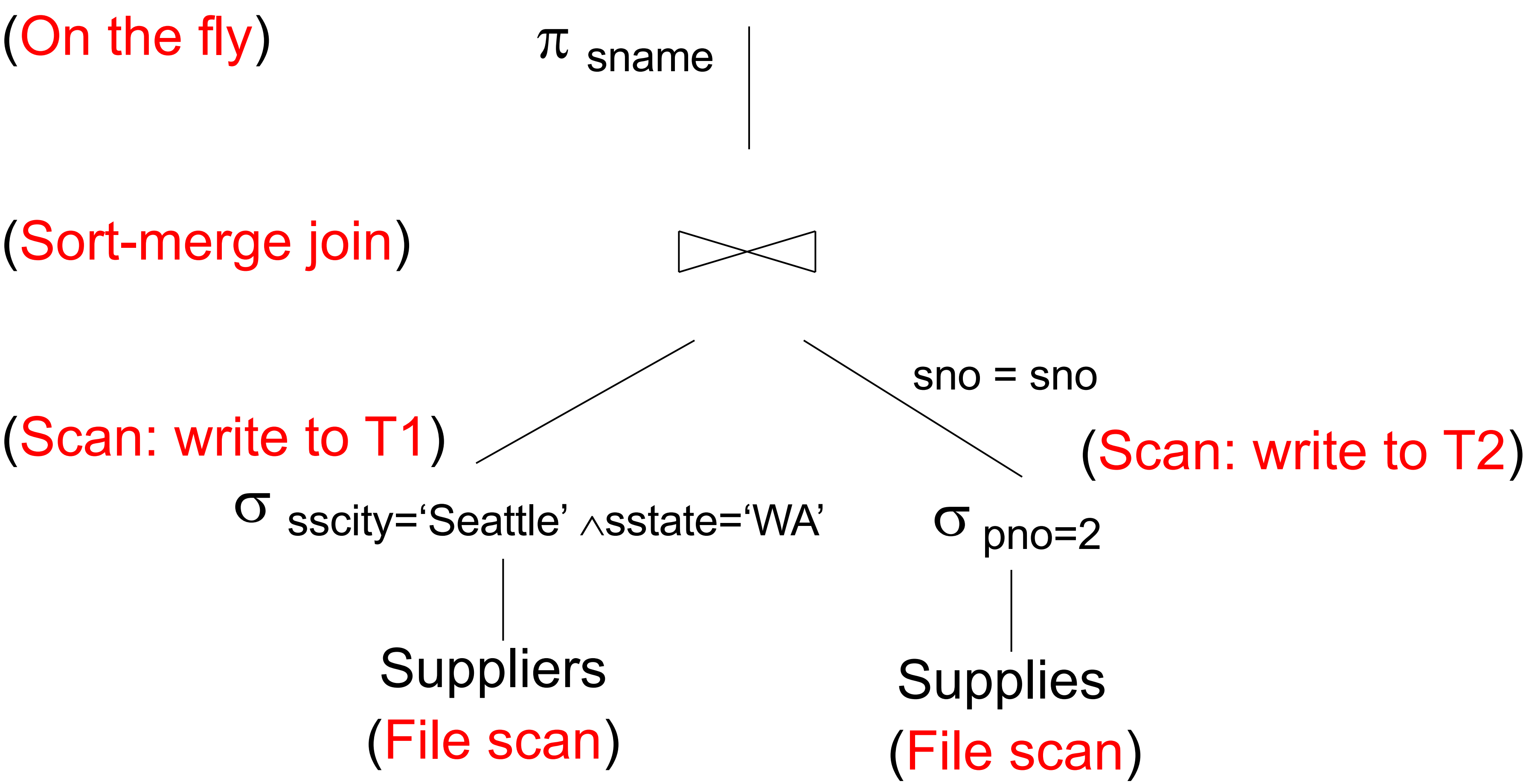
Intermediate Tuple Materialization

- Writes the results of an operator to an intermediate table on disk
- Necessary for some operator implementations
  - E.g., When operator needs to examine the same tuples multiple times

Pipelined Execution



Intermediate Tuple Materialization



Outline

- **Steps involved in processing a query**
  - Logical query plan
  - Physical query plan
  - Query execution overview

- **Operator implementations**
  - One pass algorithms
  - Two-pass algorithms
  - Index-based algorithms

Why Learn About Op Algos?

- Relevant for other Bid data systems
- Different sysmtes implement different subsets of these algorithms
- Good algorithms can greatly improve performance

Cost Parameters

- In database systems the data is on disk
- **Cost = total number of I/Os**
  - **More complex models possible (which ones?)**
  - **Different models for main-memory / distributed DBMSs (which ones?)**
- Parameters:
  - **B(R) = # of blocks (i.e., pages) for relation R**
  - **T(R) = # of tuples in relation R**
  - **V(R, a) = # of distinct values of attribute a**

Cost

- Cost of an operation = number of disk I/Os to
  - read the data
  - compute the result
- Cost of writing the result to disk is *not included*
  - Need to count it separately when applicable

Cost Parameters

- Clustered relation R:
  - $B(R) \approx T(R)$ / blockSize
- Unclustered relation R:
  - Worst case: If the system uses shared blocks
  - $B(R) \approx T(R)$
- When **a** is a primary key, V(R,a) = T(R)
- When **a** is not a primary key, V(R,a) = ?

Cost of Scanning a Table

- Scanning a table:
  - Reading all its records.

- Clustered relation:
  - Result may be unsorted: B(R)
  - Result needs to be sorted: 3B(R) (see later)

- Unclustered relation
  - Unsorted: T(R)
  - Sorted: T(R) + 2B(R)

Outline

- **Steps involved in processing a query**
  - Logical query plan
  - Physical query plan
  - Query execution overview

- **Operator implementations**
  - One pass algorithms
  - Two-pass algorithms
  - Index-based algorithms

One-pass Algorithms

Selection $\sigma(R)$, projection $\Pi(R)$

- Both are ***tuple-at-a-time*** operators
- Cost: B(R), the cost of scanning a clustered relation

Join Algorithms

- Logical operator:
  - Product(pname, cname) ⋈ Company(cname, city)

- Physical operators for the join:
  - **Hash join**
  - **Nested loop join**
  - **Sort-merge join**

Hash Join

Hash join: $R \bowtie S$

- Scan R, build buckets in main memory (M)
- Then scan S and join
- Cost: B(R) + B(S)
- One pass algorithm when one of the relations holds in memory!
  - That is: B(R) <= M

Nested Loop Joins

- Tuple-based nested loop $R \bowtie S$
- R is the outer relation, S is the inner relation

```
for each tuple r in R do
   for each tuple s in S do
      if r and s join then output (r,s)
```

- Cost: B(R) + T(R) B(S) when S is clustered
- Cost: B(R) + T(R) T(S) when S is unclustered

Page-at-a-time Refinement

```
for each page of tuples r in R do
   for each page of tuples s in S do
         for all pairs of tuples
                 if r and s join then output (r,s)
```

- Cost: B(R) + B(R)B(S) if S is clustered
- Cost: B(R) + B(R)T(S) if S is unclustered

Nested Loop Joins

- We can be cleverer

- How would you compute the join in the following cases ? What is the cost ?

  - B(R) = 1000, B(S) = 2, M = 4

  - B(R) = 1000, B(S) = 3, M = 4

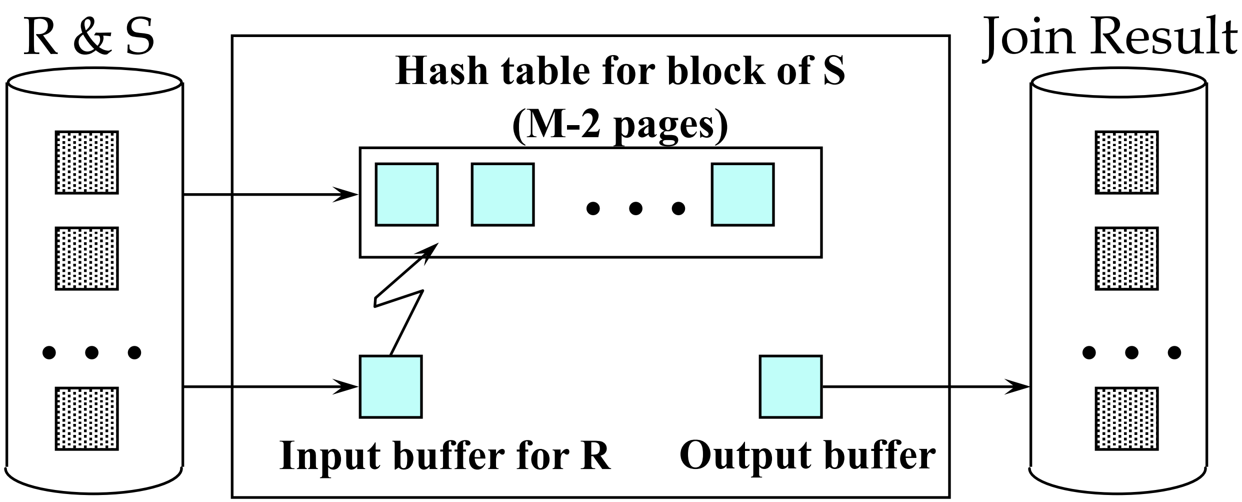
  - B(R) = 1000, B(S) = 6, M = 4

Block Nested Loop Joins

- Block Nested Loop Join
- Group of (M-2) pages of S is called a “block”

```
for each (M-2) pages ps of S do
   for each page pr of R do
         for each tuple s in ps
              for each tuple r in pr do
                   if “r and s join” then output(r,s)
```

Block Nested Loop Joins



Block Nested Loop Joins

- Cost of block-based nested loop join
  - Read S once: cost B(S)
  - Outer loop runs B(S)/(M-2) times, and each time need to read R: costs B(S)B(R)/(M-2)
  - Total cost: B(S) + B(S)B(R)/(M-2)

- Notice: it is better to iterate over the smaller relation first

Sort-Merge Join

Sort-merge join: $R \bowtie S$

- Scan R and sort in main memory
- Scan S and sort in main memory
- Merge R and S

- Cost: B(R) + B(S)
- One pass algorithm when B(S) + B(R) <= M
- Typically, this is NOT a one pass algorithm

More One-pass Algorithms

Duplicate elimination $\delta(R)$

- Need to keep tuples in memory
- When new tuple arrives, need to compare it with previously seen tuples
- Balanced search tree or hash table
- Cost: B(R)
- Assumption: $B(\delta(R)) <= M$

Even More One-pass Algorithms

Grouping:

Product(name, department, quantity)

$$\gamma_{\text{department, sum(quantity)}} \text{ (Product)}$$

How can we compute this in main memory ?

Even More One-pass Algorithms

- Grouping: $\gamma_{\text{department, sum(quantity)}}$ (R)

- Need to store all departments in memory
- Also store the sum(quantity) for each department
- Balanced search tree or hash table
- Cost: B(R)
- Assumption: number of depts fits in memory

Outline

- **Steps involved in processing a query**
  - Logical query plan
  - Physical query plan
  - Query execution overview

- **Operator implementations**
  - One pass algorithms
  - Two-pass algorithms
  - Index-based algorithms

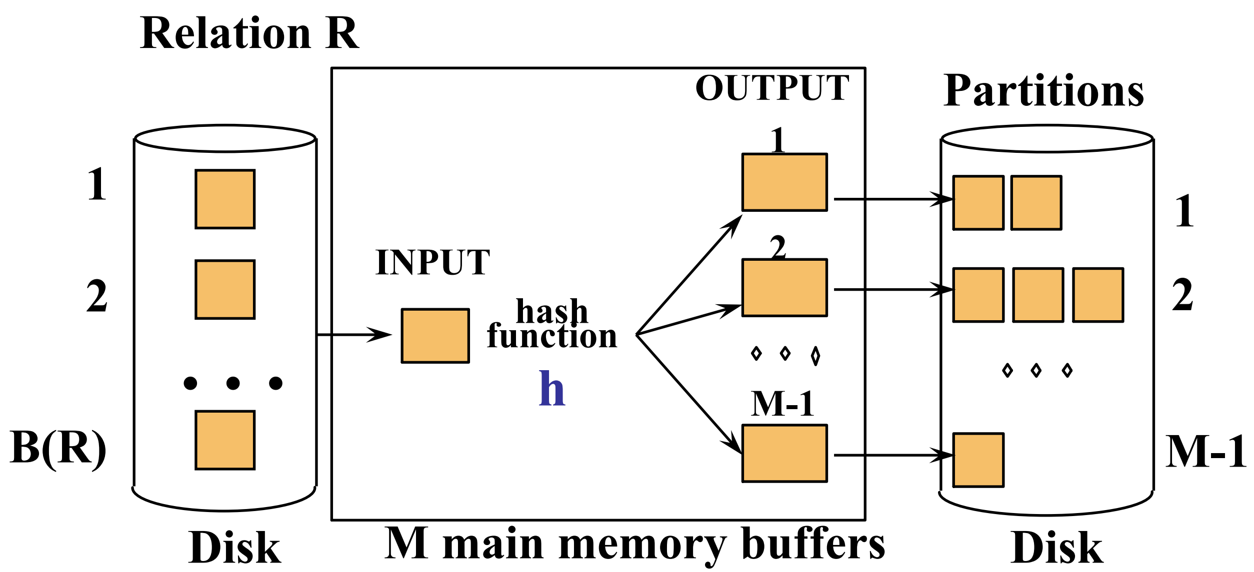
Two-Pass Algorithms

- What if data does not fit in memory?
- Need to process it in multiple passes

- Two key techniques
  - Hashing
  - Sorting
  - Write back to disk!

Two Pass Algorithms Based on Hashing

- Idea: partition a relation R into buckets, on disk
- Each bucket has size approx. B(R)/M



- Does each bucket fit in main memory ?
  - Yes if B(R)/M <= M, i.e. $B(R) <= M^2$

Hash Based Algorithms for δ

- Recall: $\delta(R) =$ duplicate elimination

- Step 1. Partition R into buckets
- Step 2. Apply δ to each bucket

- Cost: 3B(R)
- Assumption: $B(R) <= M^2$

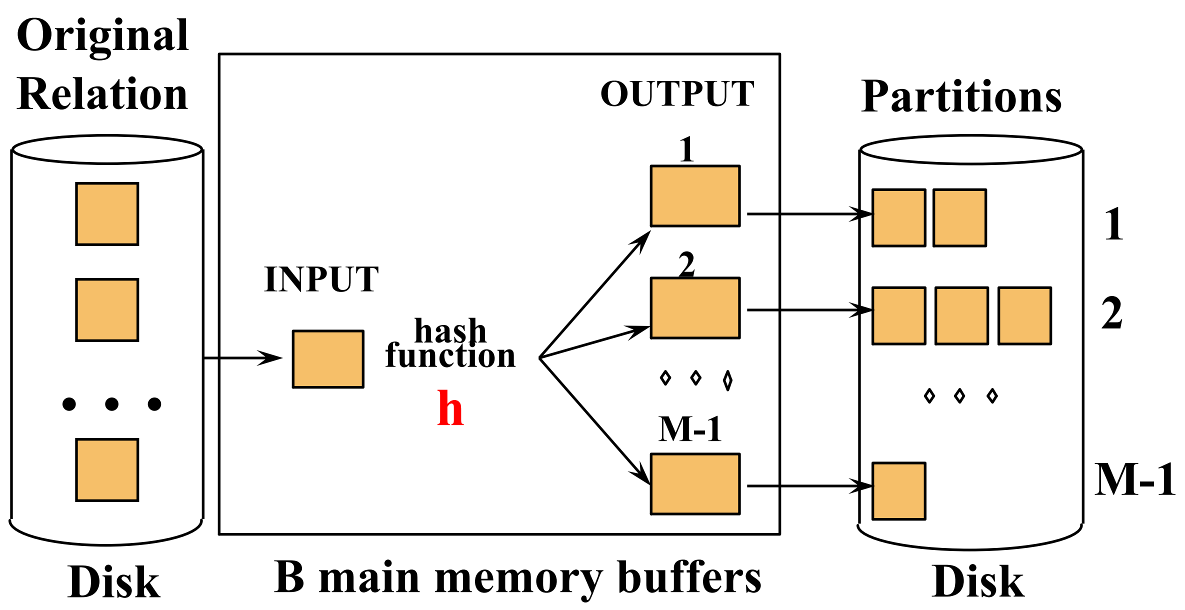
Partitioned (Grace) Hash Join

$R \bowtie S$

- Step 1:
  - Hash S into M-1 buckets
  - Send all buckets to disk
- Step 2
  - Hash R into M-1 buckets
  - Send all buckets to disk
- Step 3
  - Join every pair of buckets

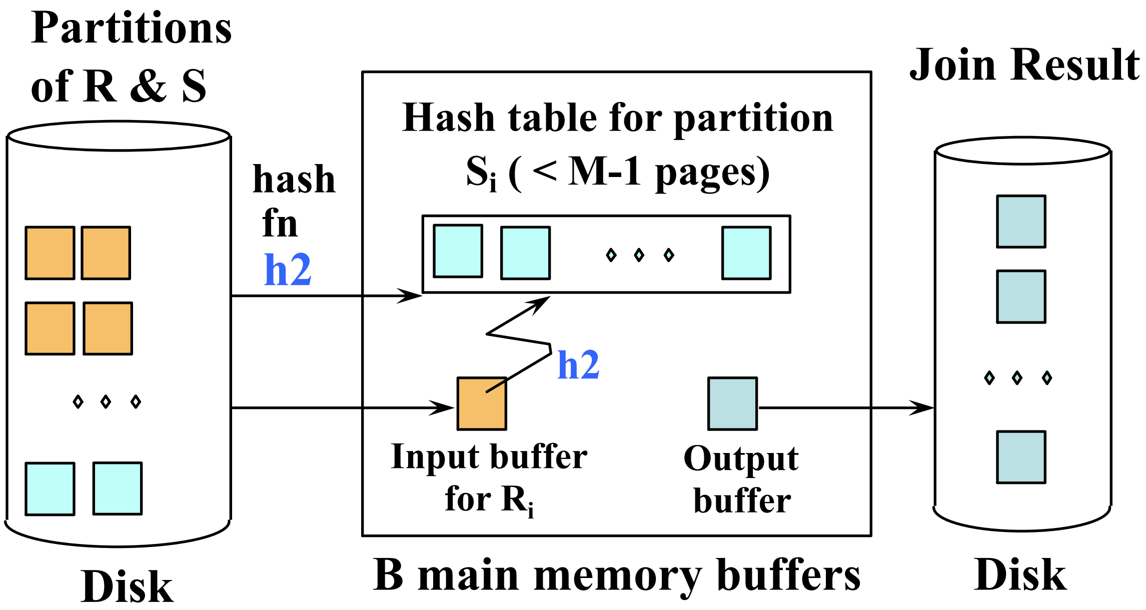
Partitioned Hash Join

- Partition both relations using hash function **h**
- R tuples in partition *i* will only match S tuples in partition *i*.



Partitioned Hash Join

- Read in partition of R, hash it using h2 (≠ h)
  - **Build phase**
- Scan matching partition of S, search for matches
  - **Probe phase**



Partitioned Hash Join

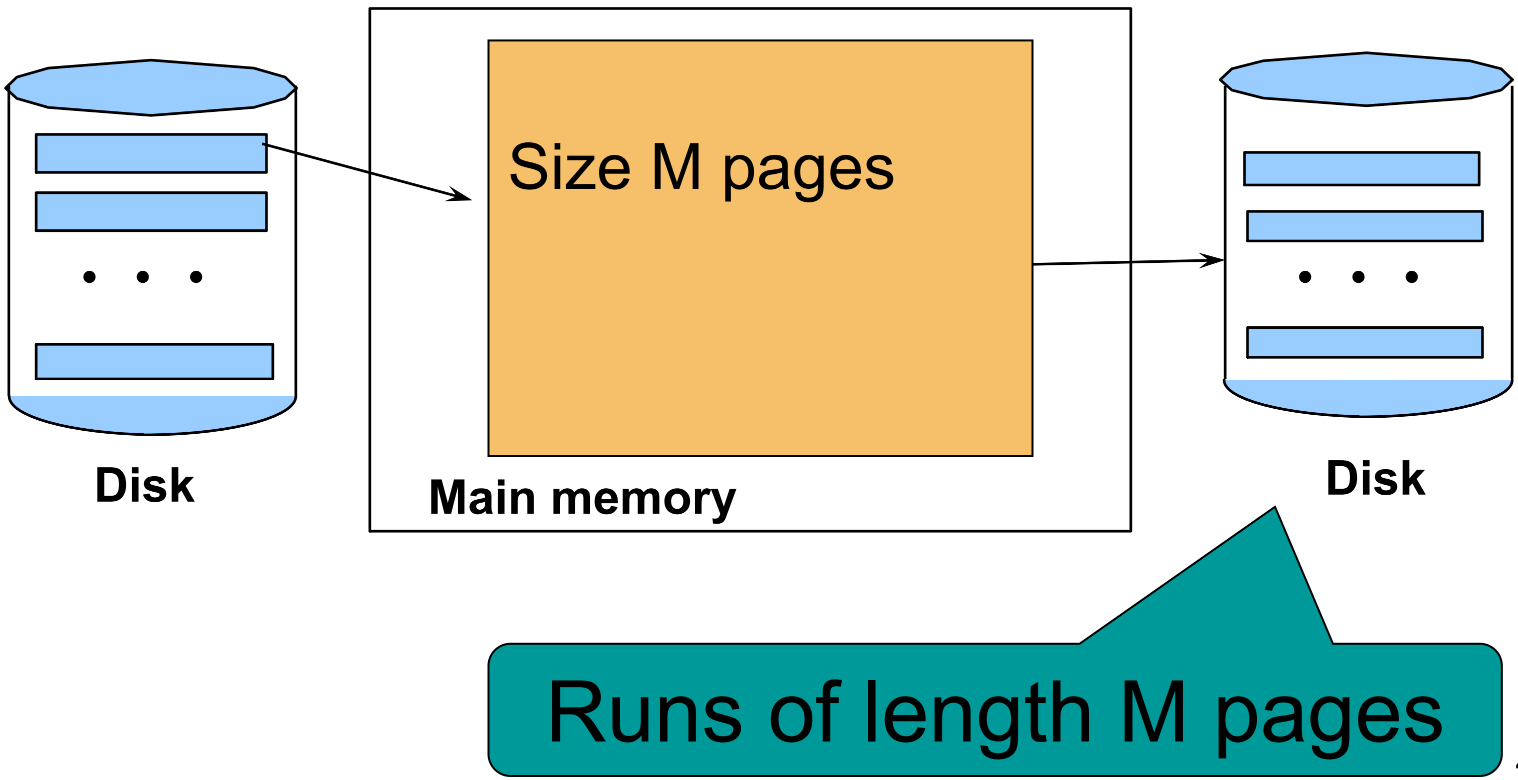
- Cost: 3B(R) + 3B(S)
- Assumption: B(R) and B(S) <= M^2

External Sorting

- Problem: Sort a file of size B with memory M
- E.g., ORDER BY in SQL queries
- Will discuss only 2-pass sorting, for when $B < M^2$

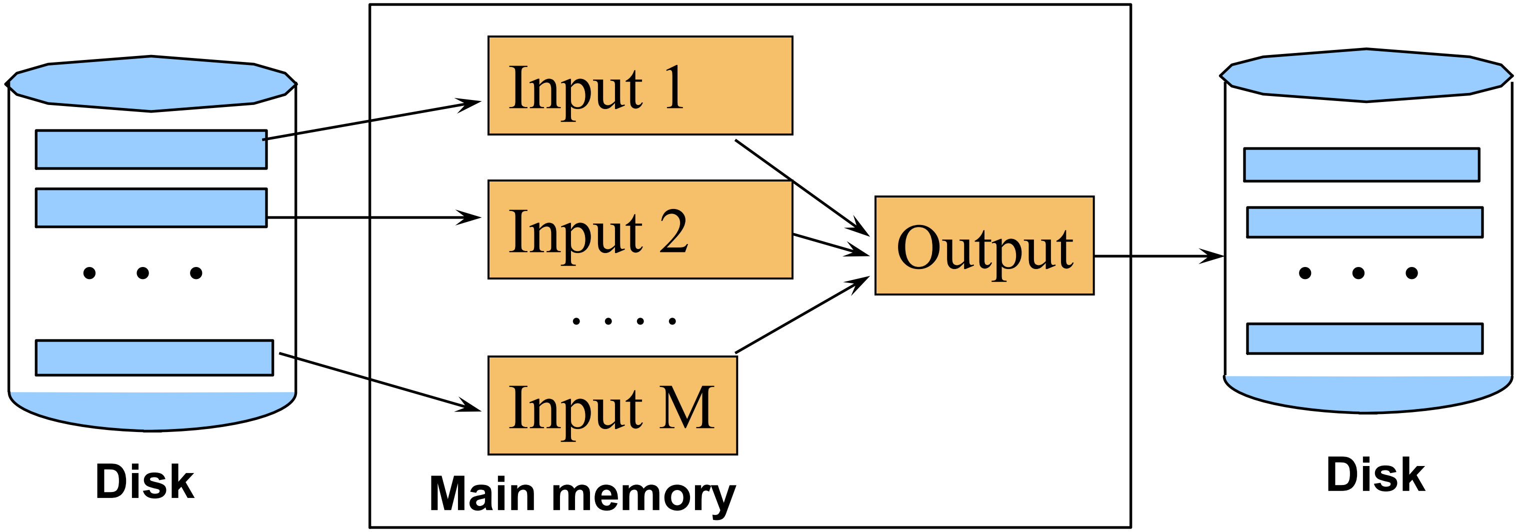
External Merge-Sort: Step 1

- Phase one: load M pages in memory, sort



External Merge-Sort: Step 2

- Merge M – 1 runs into a new run
- Output: run of length M (M – 1) $\approx M^2$



If $B(R) \le M^2$ then we are done
(slightly more complex otherwise)

External Merge-Sort

- Cost:
  - Read+write+read = 3B(R)
  - Assumption: B(R) <= M^2

- Other considerations
  - In general, a lot of optimizations are possible

Two-Pass Algorithms Based on Sorting

Duplicate elimination $\delta(R)$

- Trivial idea: sort first, then eliminate duplicates
- Step 1: sort chunks of size M, write
  - cost 2B(R)
- Step 2: merge M-1 runs, but include each tuple only once
  - cost B(R)
- Total cost: 3B(R), Assumption: $B(R) <= M^2$

Two-Pass Algorithms Based on Sorting

Grouping: $\gamma_{a,\ sum(b)}$ (R)

- Same as before: sort, then compute the sum(b) for each group of a's
- Total cost: 3B(R)
- Assumption: B(R) <= M^2

Two-Pass Algorithms Based on Sorting

Join R $\bowtie$ S

- Start by sorting both R and S on the join attribute:
  - Cost: 4B(R)+4B(S) (because need to write to disk)
- Read both relations in sorted order, match tuples
  - Cost: B(R)+B(S)
- Total cost: 5B(R)+5B(S)
- Assumption: B(R) <= M^2, B(S) <= M^2

Two-Pass Algorithms Based on Sorting

Join $R \bowtie S$

- Also, if $B(R) + B(S) <= M^2$ we can compute the join during the merge phase
- Total cost: $3B(R)+3B(S)$

Outline

- **Steps involved in processing a query**
  - Logical query plan
  - Physical query plan
  - Query execution overview
- **Operator implementations**
  - One pass algorithms
  - Two-pass algorithms
  - Index-based algorithms

Review: Access Methods

- **Heap file**
  - Scan tuples one at the time
- **Hash-based index**
  - Efficient selection on equality predicates
- **Tree-based index**
  - Efficient selection on equality or range predicates

Index Based Selection

- Selection on equality: $\sigma_{a=v}(R)$
- V(R, a) = *# of distinct values of attribute a*
- Clustered index on a: cost ~ B(R)/V(R,a)
- Unclustered index on a: cost ~ T(R)/V(R,a)
- Note: we ignored the I/O cost for the index pages

Index Based Selection

- Example:
  - B(R) = 2000
  - T(R) = 100,000
  - V(R, a) = 20
  - Compute the cost of $\sigma_{a=v}(R)$
- Index based selection
  - If index is clustered: B(R)/V(R,a) = 100
  - If index is unclustered: T(R)/V(R,a) = 5,000
- Note
  - Don't build unclustered indexes when V(R,a) is small.

Index Nested Loop Join

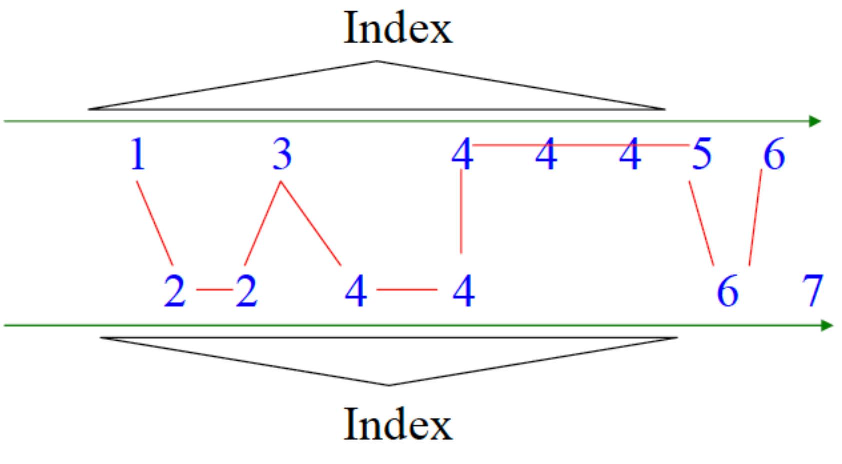
$R \bowtie S$ (Natural Join)

- Assume S has an index on the join attribute
- Iterate over R, for each tuple fetch corresponding tuple(s) from S

- Cost:
  – Assuming R is clustered
  – If index on S is clustered: $B(R) + T(R)B(S)/V(S,a)$
  – If index on S is unclustered: $B(R) + T(R)T(S)/V(S,a)$

Index Based Join

- Assume both R and S have a sorted index (B+tree) on the join attribute
- Perform a merge join (Zig-Zag join)
- Cost: B(R) + B(S)



Outline

- **Steps involved in processing a query**
  - Logical query plan
  - Physical query plan
  - Query execution overview

- **Operator implementations**
  - One pass algorithms
  - Two-pass algorithms
  - Index-based algorithms

Summary of Query Execution

- For each logical query plan
  - There exist many physical query plans
  - Each plan has a different cost
  - Cost depends on the data
- Additionally, for each query
  - There exist several logical plans
- Query optimization factors in the cost of several logical plans by using the operators and data estimates (very fast)