Introduction to Database Systems CSE 444

Lecture 19: Query Processing Overview

Where We Are

- We are learning how a DBMS executes a query
  - How come a DBMS can execute a query so fast?

- Lectures 16-17: Data storage, indexing, physical tuning
- Lecture 18: Relational algebra

- Lecture 19: Overview of query processing steps
  - Includes a description of how queries are executed

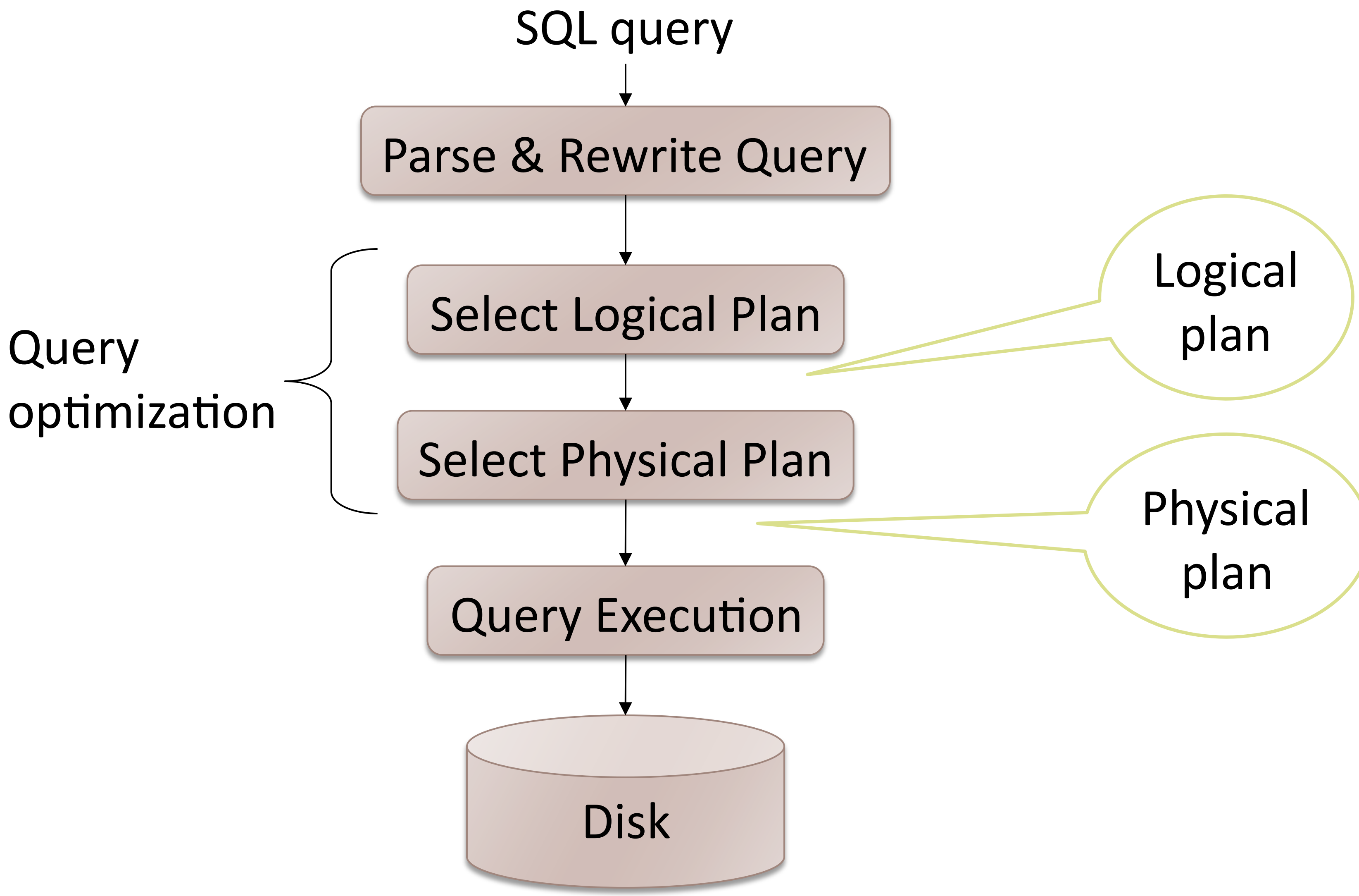
- Lecture 20: Operator algorithms
- Lectures 21-23: Overview of query optimization

Outline for Today

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

- **Readings**: Section 15.1 of the book
  - Query processing steps
  - Query execution using the iterator model
  - An introduction to next lecture on operator algos

Query Evaluation Steps



Example Database Schema

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

View: Suppliers in Seattle

```
CREATE VIEW NearbySupp AS
SELECT sno, sname
FROM Supplier
WHERE scity='Seattle' AND sstate='WA'
```


x

Example Query

Find the names of all suppliers in Seattle

who supply part number 2

```
SELECT sname
FROM  NearbySupp
WHERE sno IN ( SELECT sno
               FROM Supplies
               WHERE 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.

Rewritten Version of Our Query

Original query:

```
SELECT sname
FROM   NearbySupp
WHERE sno IN ( SELECT sno
               FROM Supplies
               WHERE pno = 2 )
```

Rewritten query:

```
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;
```

Continue with Query Evaluation

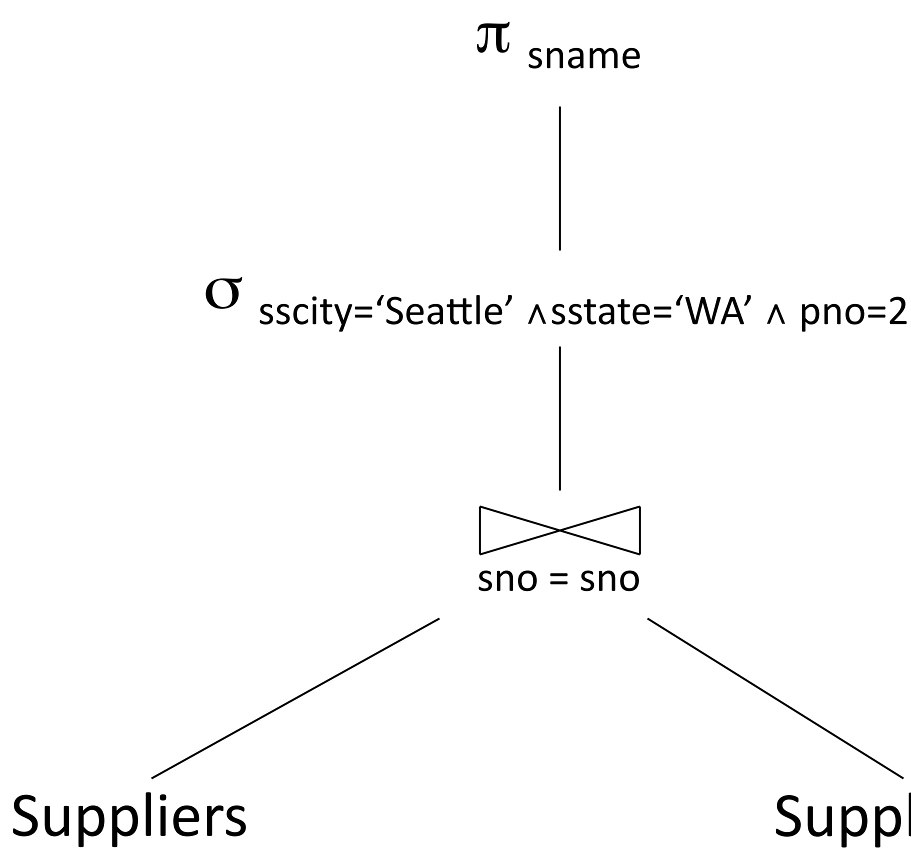
- **Step 3: Query optimization**
  - Find an efficient query plan for executing the query
  - We will spend three lectures on this topic

- **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

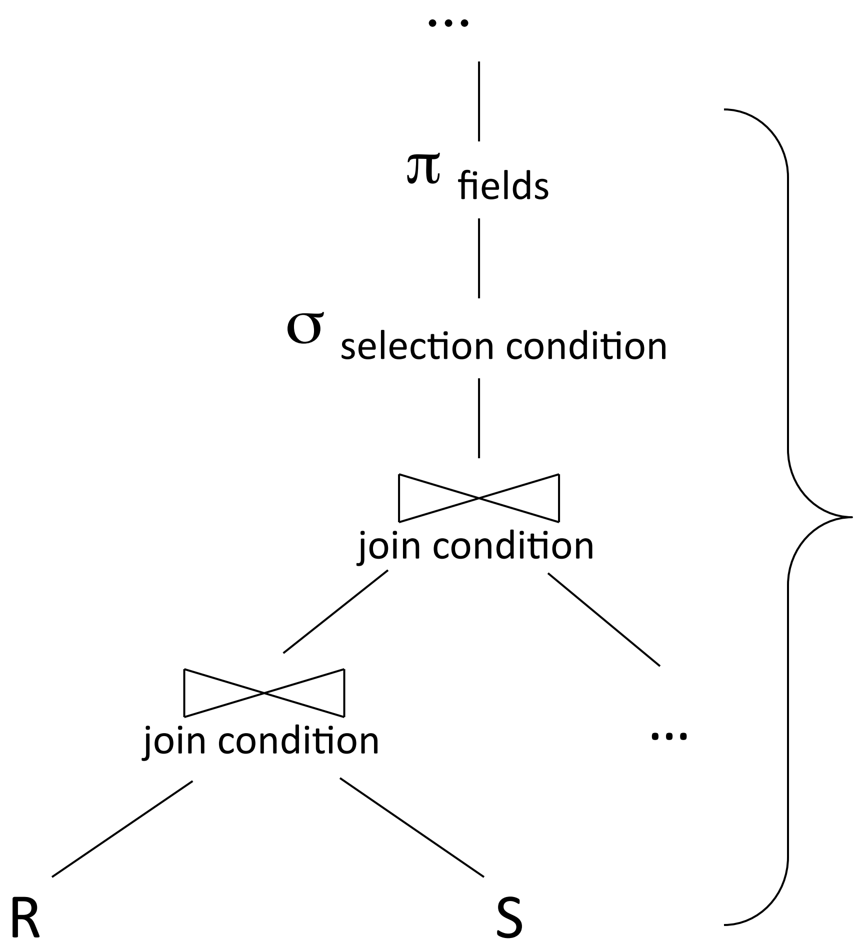
Logical Query Plan



Query Block

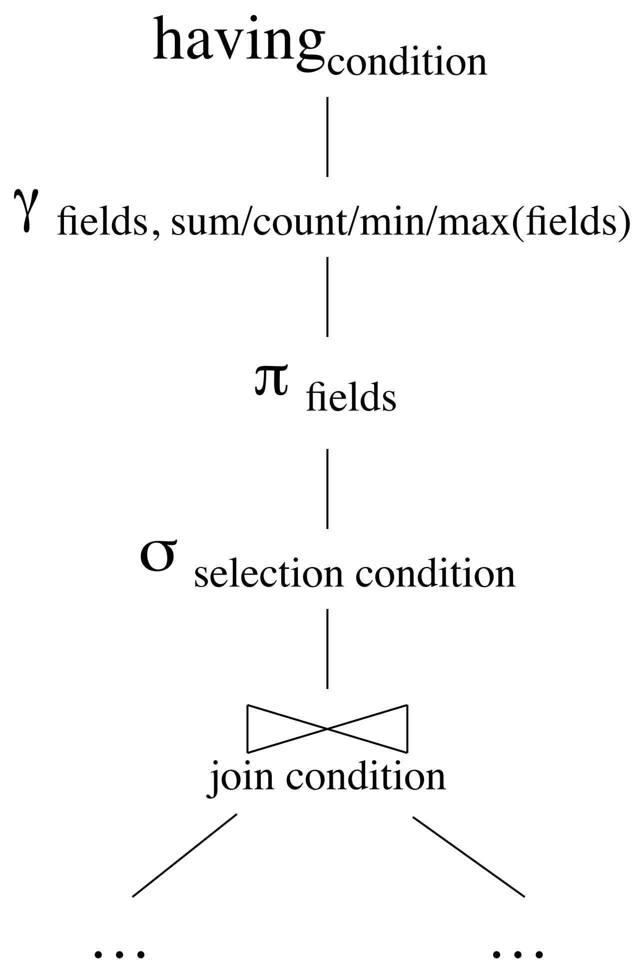
- Most optimizers operate on individual query blocks
- A query block is an SQL query with no nesting
  - Exactly one
    - SELECT clause
    - FROM clause
  - At most one
    - WHERE clause
    - GROUP BY clause
    - HAVING clause

Typical Plan for Block (1/2)



SELECT-PROJECT-JOIN Query

Typical Plan For Block (2/2)

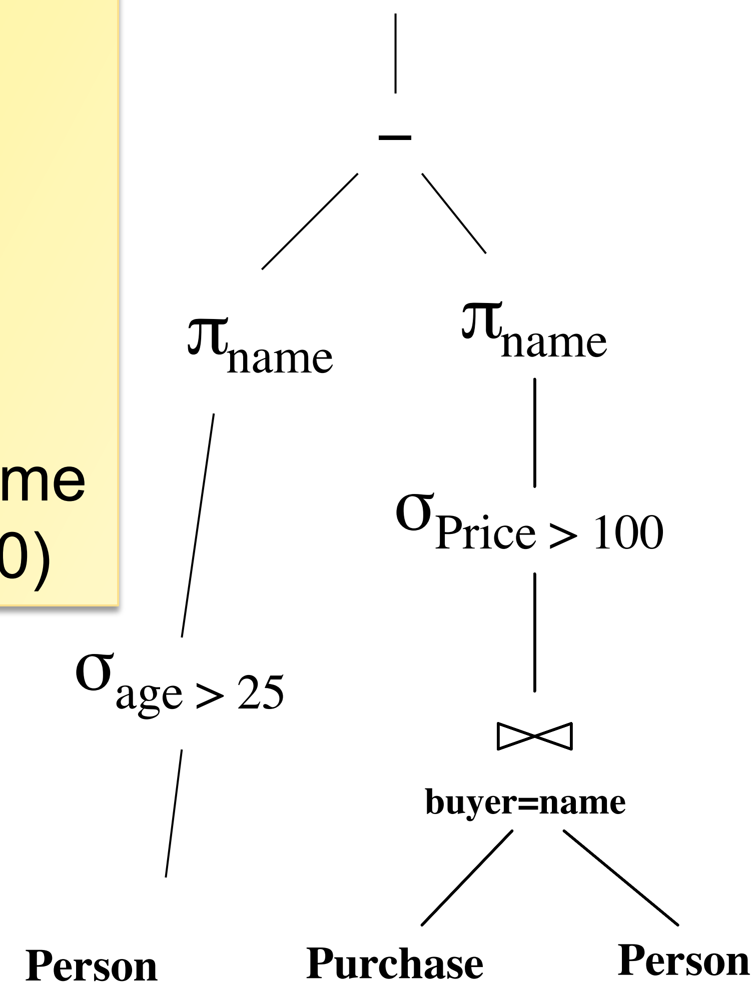


How about Subqueries?

```
SELECT  Q.name
FROM Person Q
WHERE  Q.age > 25
        AND NOT EXISTS
              (SELECT *
              FROM Purchase P
              WHERE P.buyer = Q.name
                    AND P.price > 100)
```

How about Subqueries?

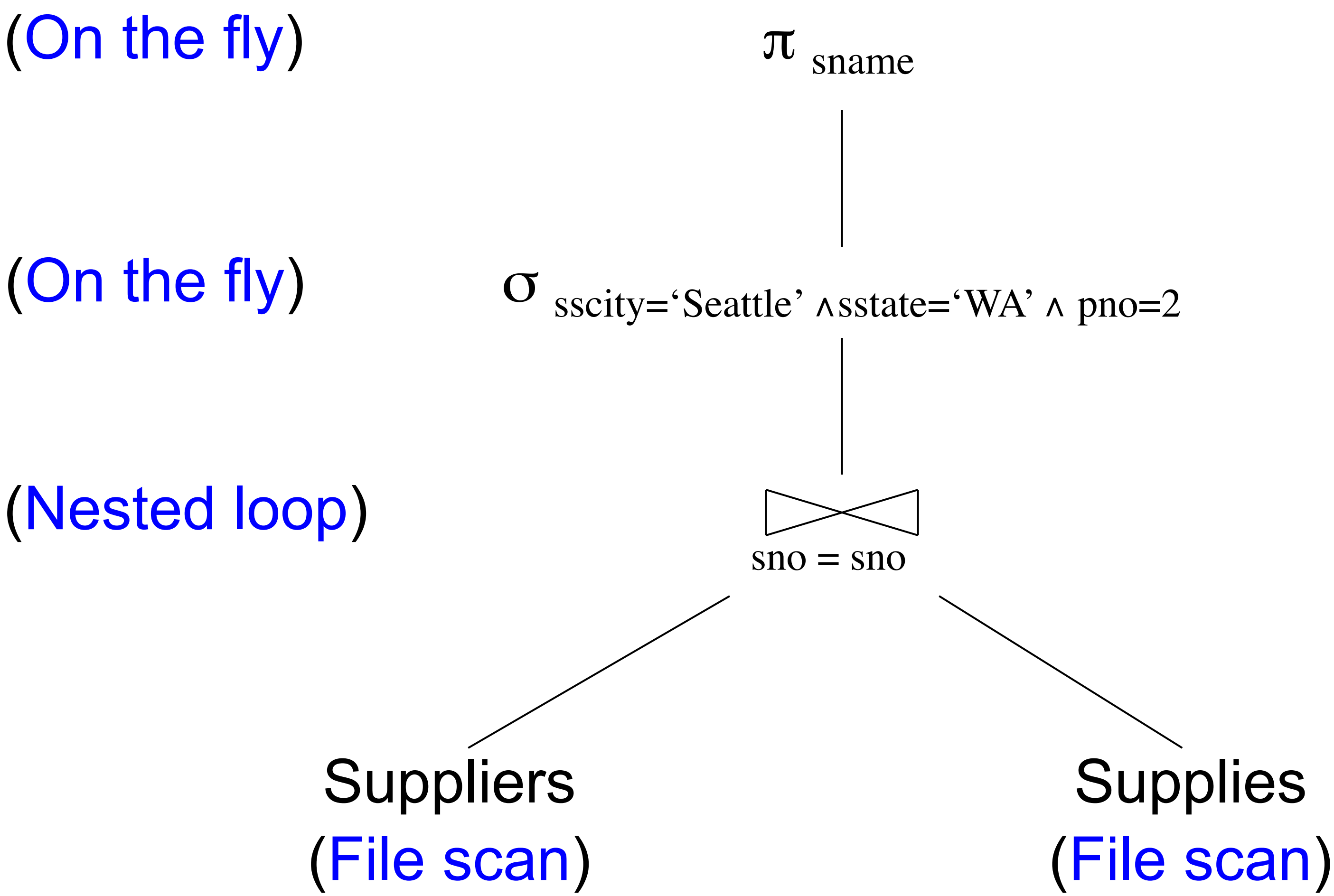
```
SELECT  Q.name
FROM Person Q
WHERE  Q.age > 25
        AND NOT EXISTS
            (SELECT *
            FROM Purchase P
            WHERE P.buyer = Q.name
                AND P.price > 100)
```



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?

- Approach:
  - Iterator interface with
    - Pipelined execution or
    - Intermediate result materialization

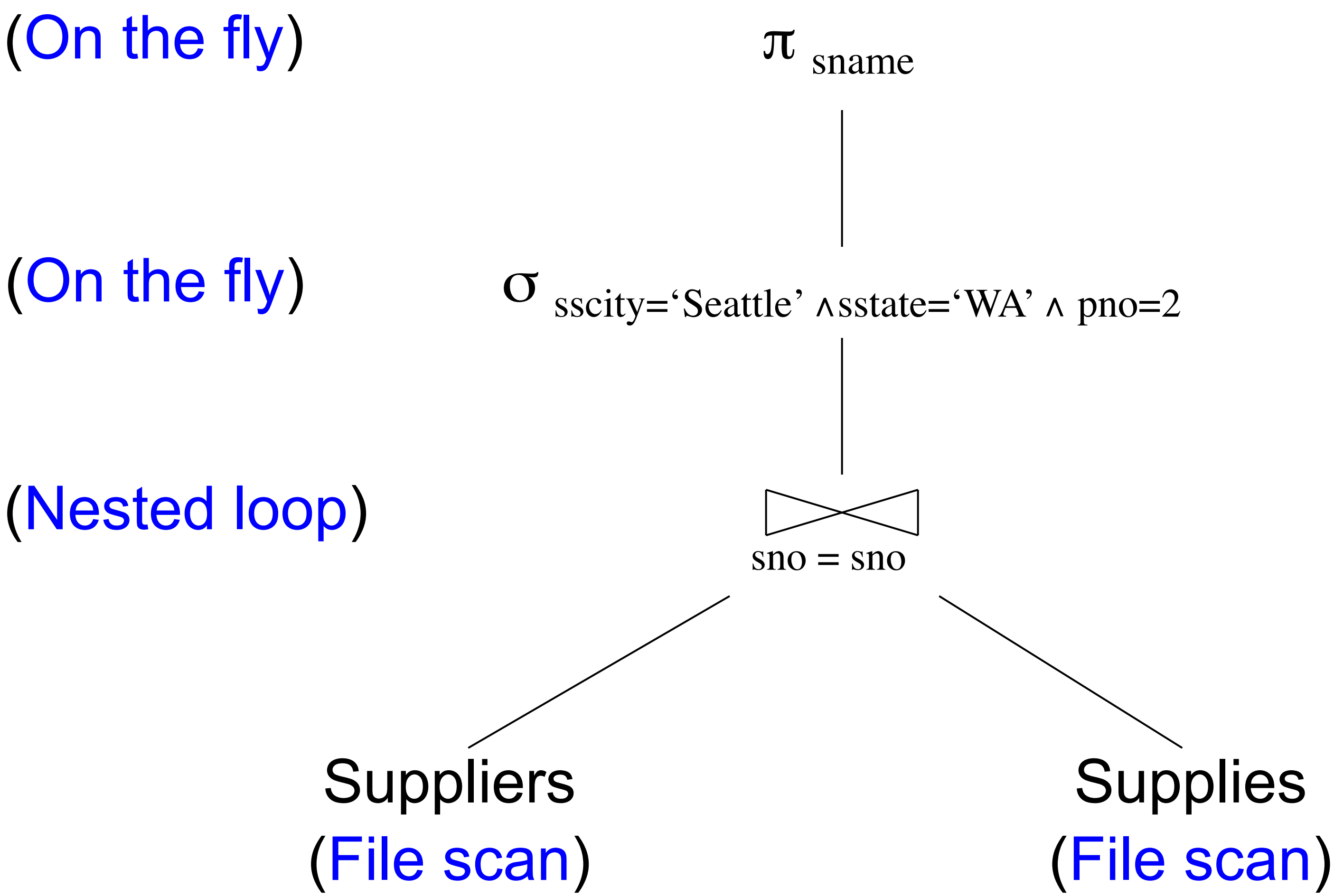
Iterator Interface

- **Each operator implements iterator interface**
- Interface has only three methods
- open()
  - Initializes operator state
  - Sets parameters such as selection condition
- get_next()
  - Operator invokes get_next() recursively on its inputs
  - Performs processing and produces an output tuple
- close(): cleans-up state

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

Pipelined Execution

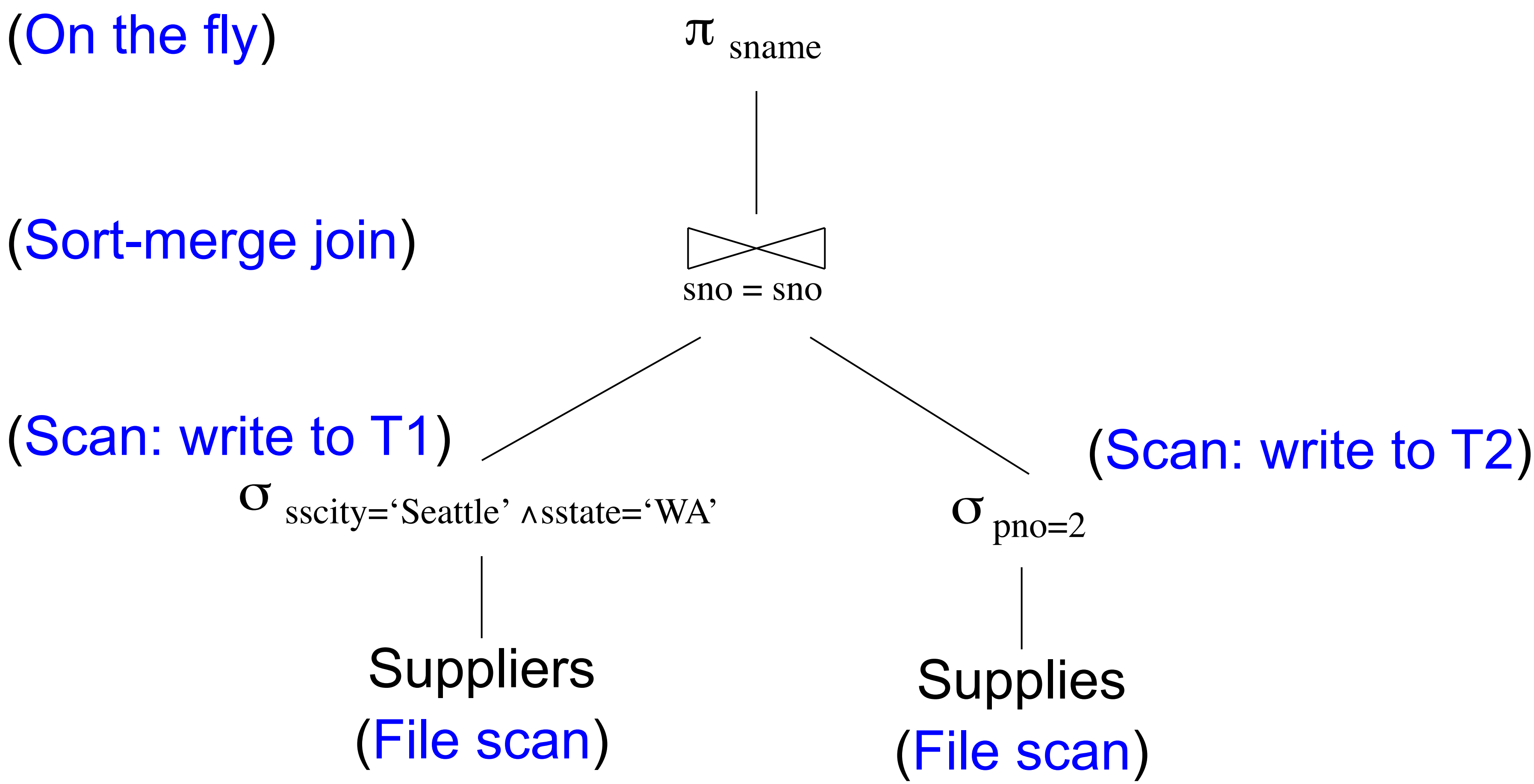


Intermediate Tuple Materialization

- Writes the results of an operator to an intermediate table on disk

- No direct benefit but
- Necessary for some operator implementations
- When operator needs to examine the same tuples multiple times

Intermediate Tuple Materialization



Next Time

- Algorithms for physical op. implementations

- How to find a good query plan?