Introduction to Database Systems
CSE 444

Lecture 18: 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?

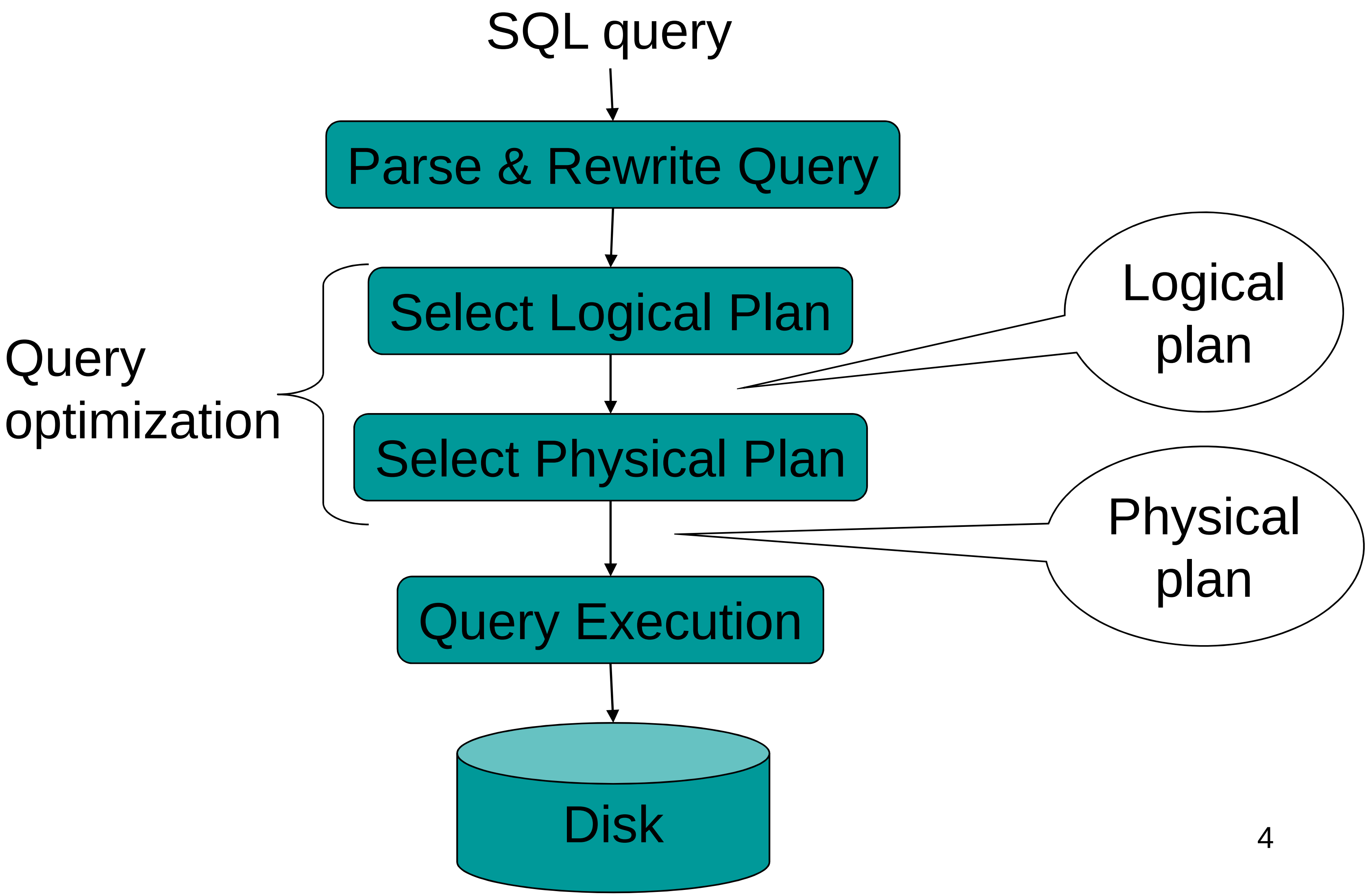
- Lecture 15-16: Data storage, indexing, physical tuning
- Lecture 17: Relational algebra
- Lecture 18: Overview of query processing steps
  - Includes a description of how queries are executed
- Lecture 19: Operator algorithms
- Lecture 20: Overview of query optimization

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 intro to next lecture on operator algorithms

Query Evaluation Steps



Example Database Schema

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

View: Suppliers in Seattle

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

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

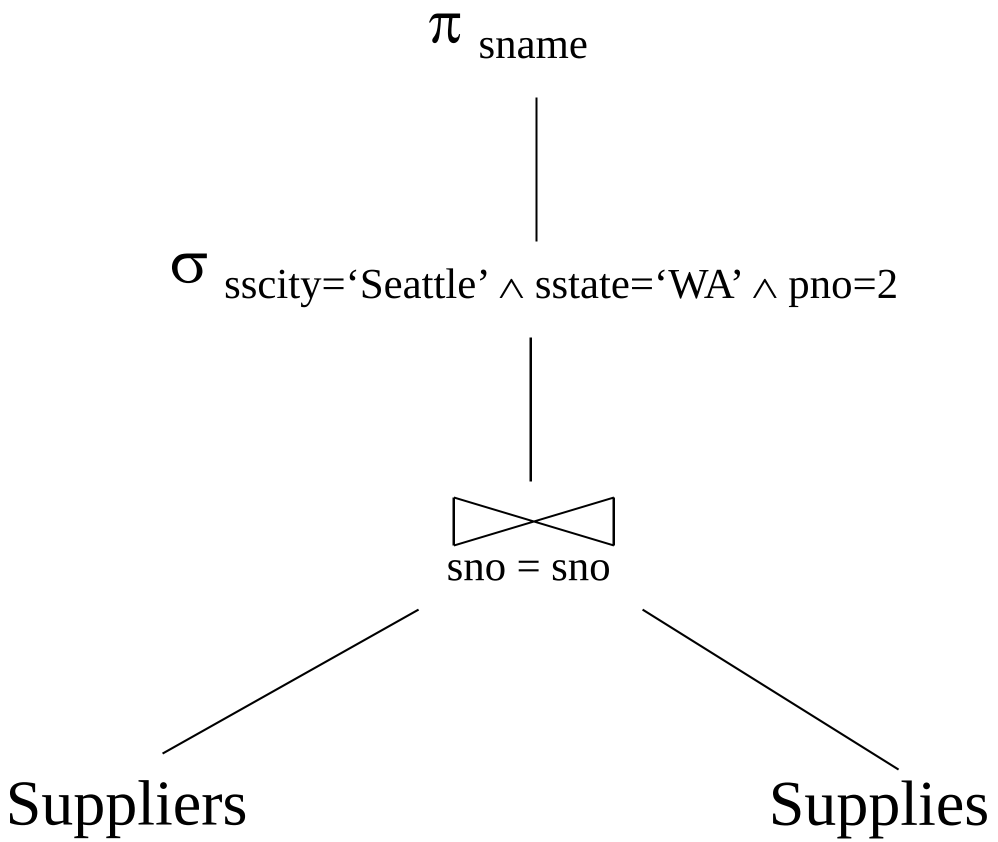
- **Step 3: Query optimization**
  - Find an efficient query plan for executing the query

- 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



π_{sname}

$\sigma_{\text{sscity='Seattle'} \wedge \text{sstate='WA'} \wedge \text{pno=2}}$

$\bowtie_{\text{sno = sno}}$

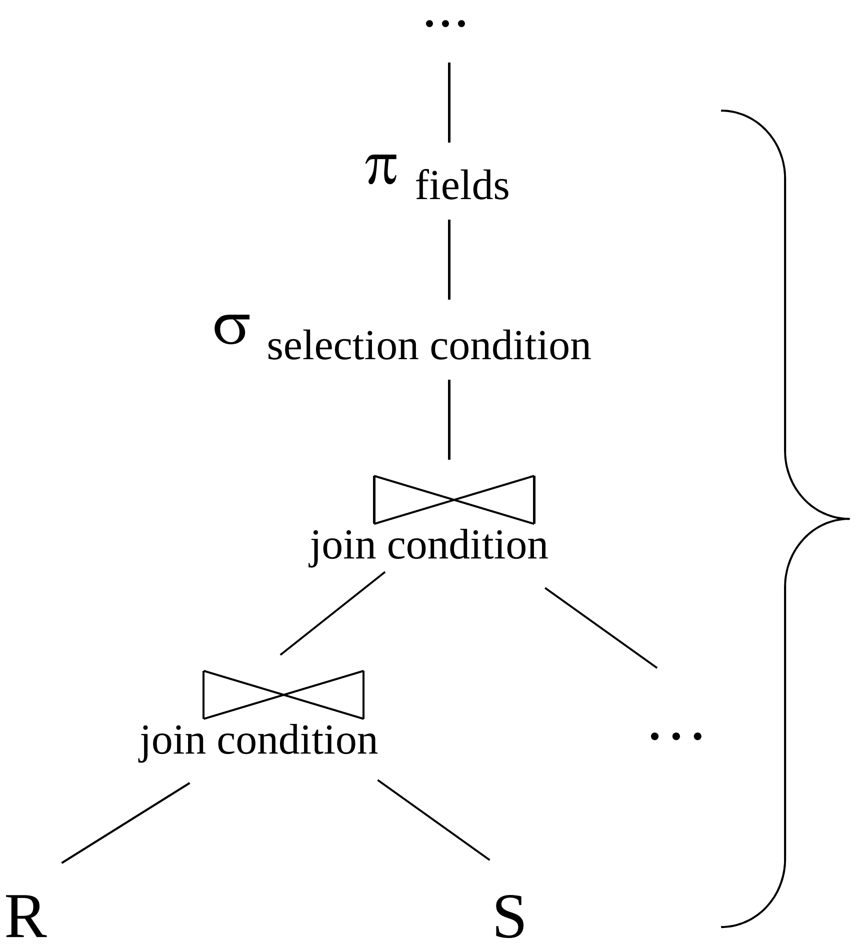
Suppliers

Supplies

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)



…

π fields

σ selection condition

$\bowtie$ join condition

$\bowtie$ join condition

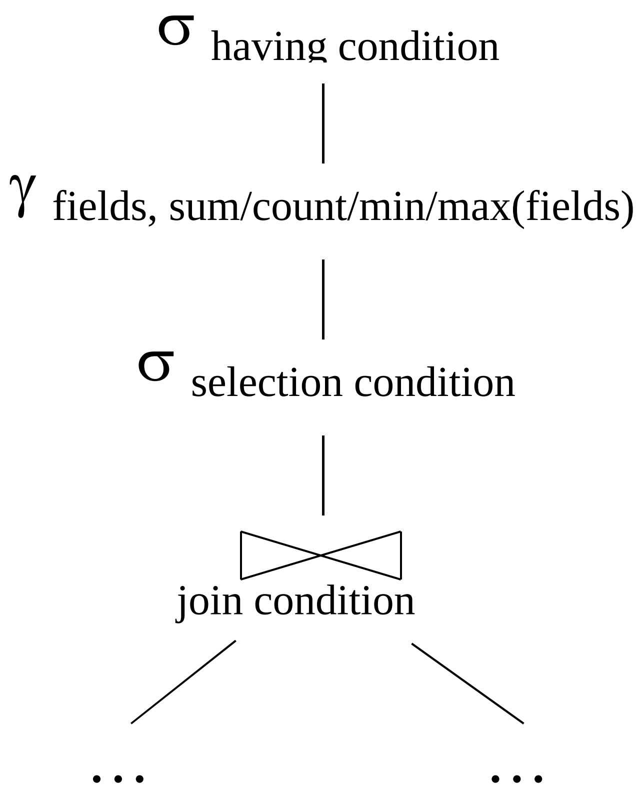
…

R

S

SELECT-PROJECT-JOIN Query

Typical Plan For Block (2/2)



$\sigma_{\text{having condition}}$

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

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

$\bowtie$ join condition

… …

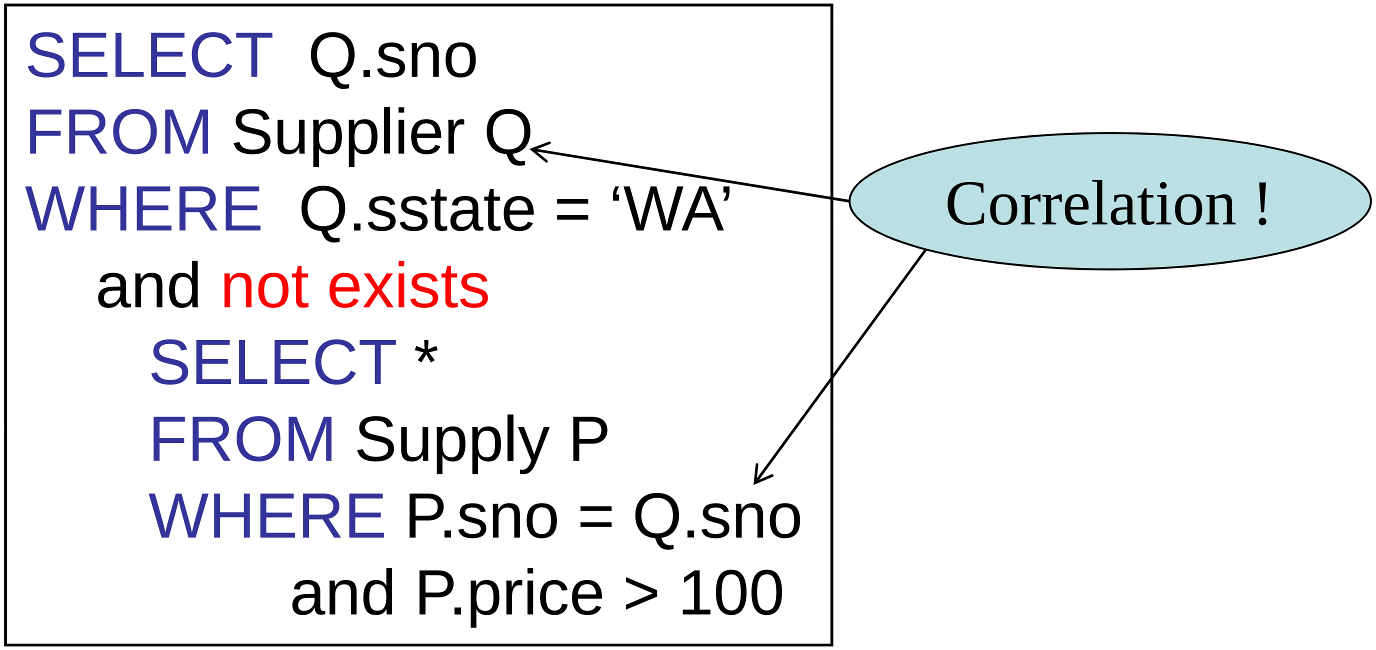
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

How about Subqueries?

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = ‘WA’
    and not exists
       SELECT *
       FROM Supply P
       WHERE P.sno = Q.sno
               and P.price > 100
```

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

How about Subqueries?



```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = ‘WA’
    and not exists
       SELECT *
       FROM Supply P
       WHERE P.sno = Q.sno
               and P.price > 100
```

Correlation !

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

How about Subqueries?

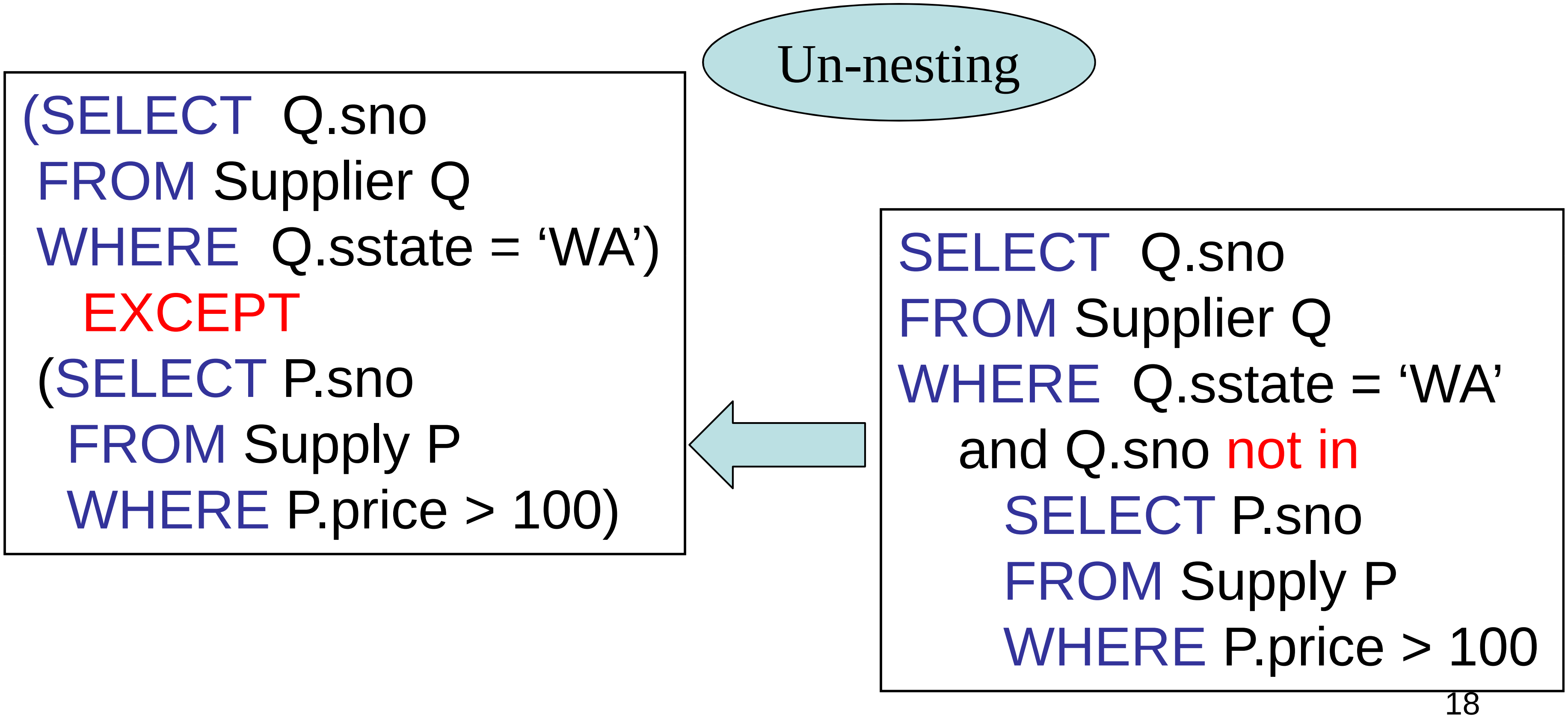
```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = ‘WA’
    and not exists
       SELECT *
       FROM Supply P
       WHERE P.sno = Q.sno
               and P.price > 100
```

De-Correlation

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = ‘WA’
    and Q.sno not in
       SELECT P.sno
       FROM Supply P
       WHERE P.price > 100
```

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

How about Subqueries?



```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = ‘WA’)
    EXCEPT
 (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

Un-nesting

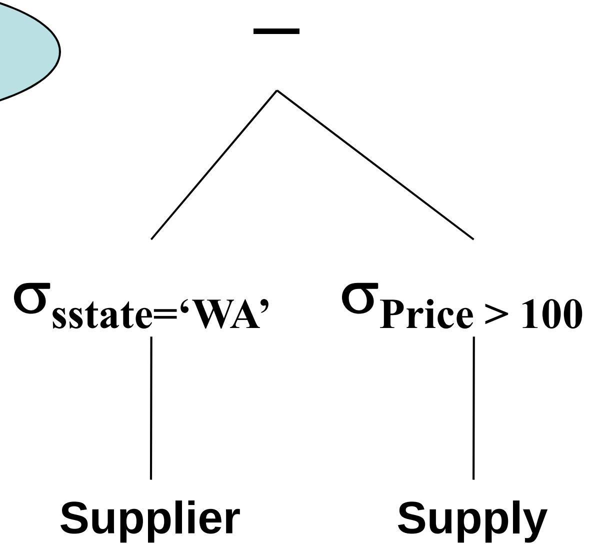
```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = ‘WA’
    and Q.sno not in
       SELECT P.sno
       FROM Supply P
       WHERE P.price > 100
```

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

How about Subqueries?

```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = ‘WA’)
    EXCEPT
 (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

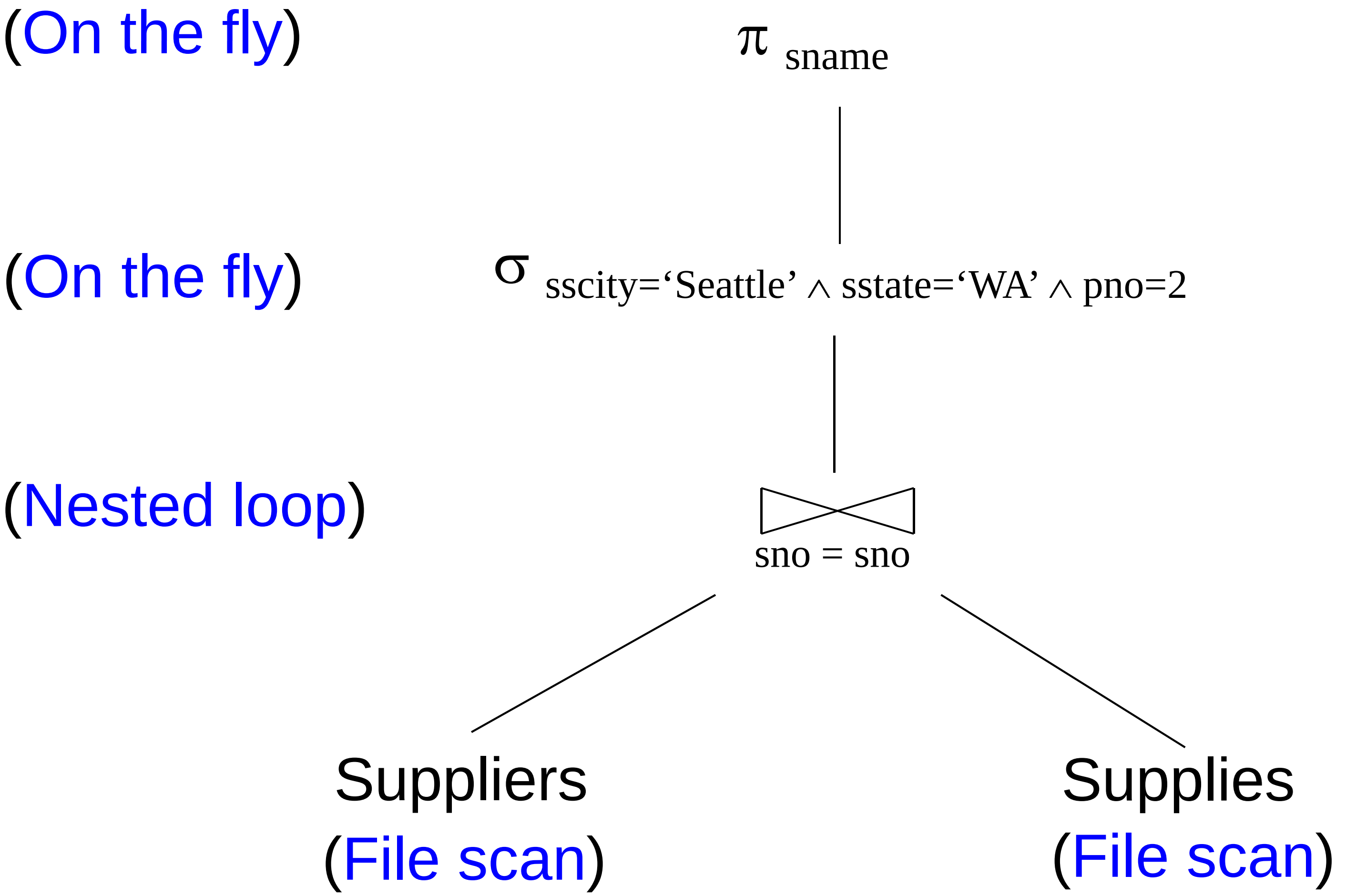
Finally…



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



(On the fly) π_{sname}

(On the fly) $\sigma_{\text{sscity='Seattle'} \wedge \text{sstate='WA'} \wedge \text{pno=2}}$

(Nested loop) $\bowtie_{\text{sno = sno}}$

Suppliers (File scan)

Supplies (File scan)

Final Step in Query Processing

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

- Approach:
  - One thread per query
  - Iterator interface
  - Pipelined execution, or
  - Intermediate result materialization

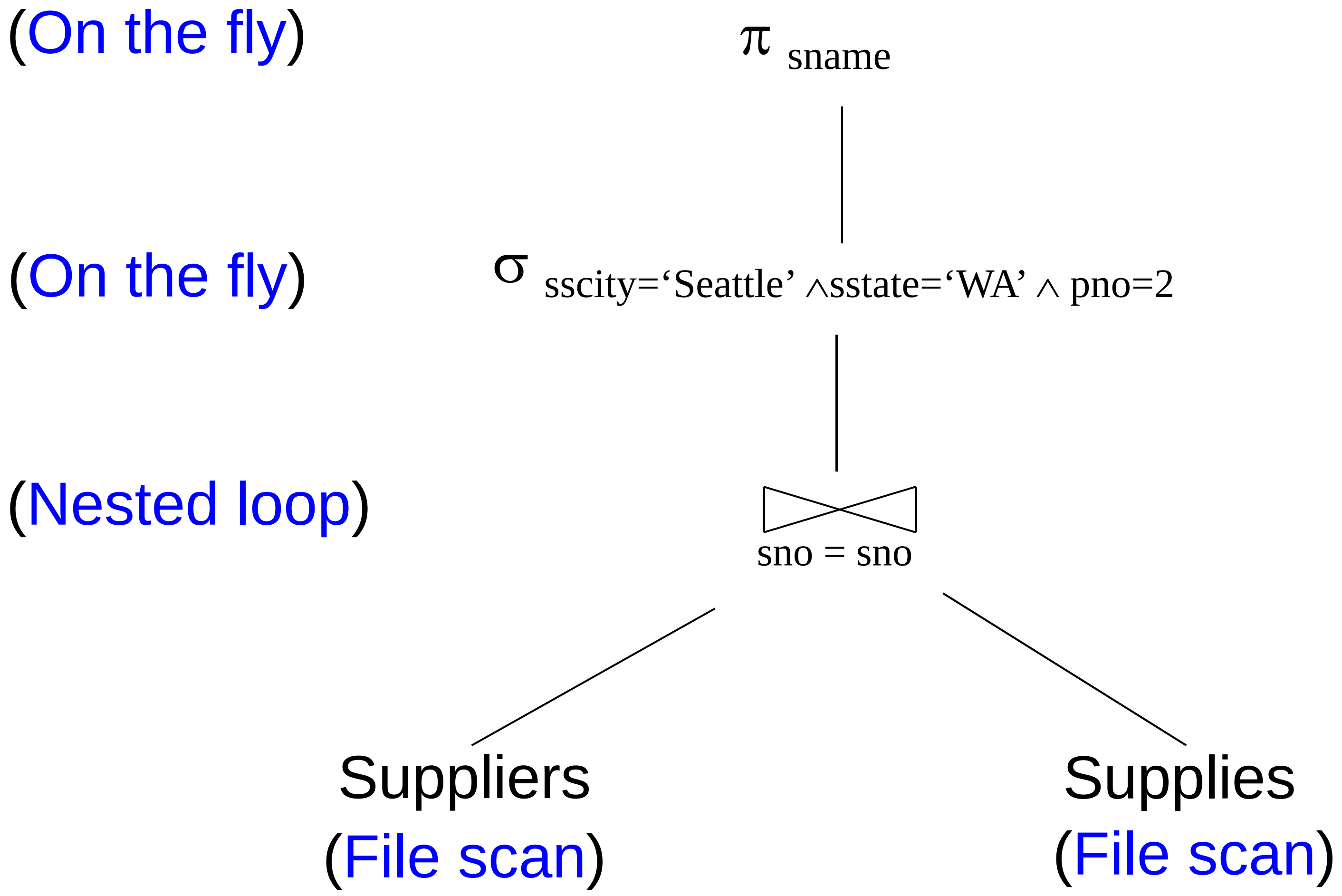
Iterator Interface

- 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



(On the fly) π_{sname}

(On the fly) $\sigma_{\text{sscity='Seattle'} \wedge \text{sstate='WA'} \wedge \text{pno=2}}$

(Nested loop) $\bowtie_{\text{sno = sno}}$

Suppliers (File scan)

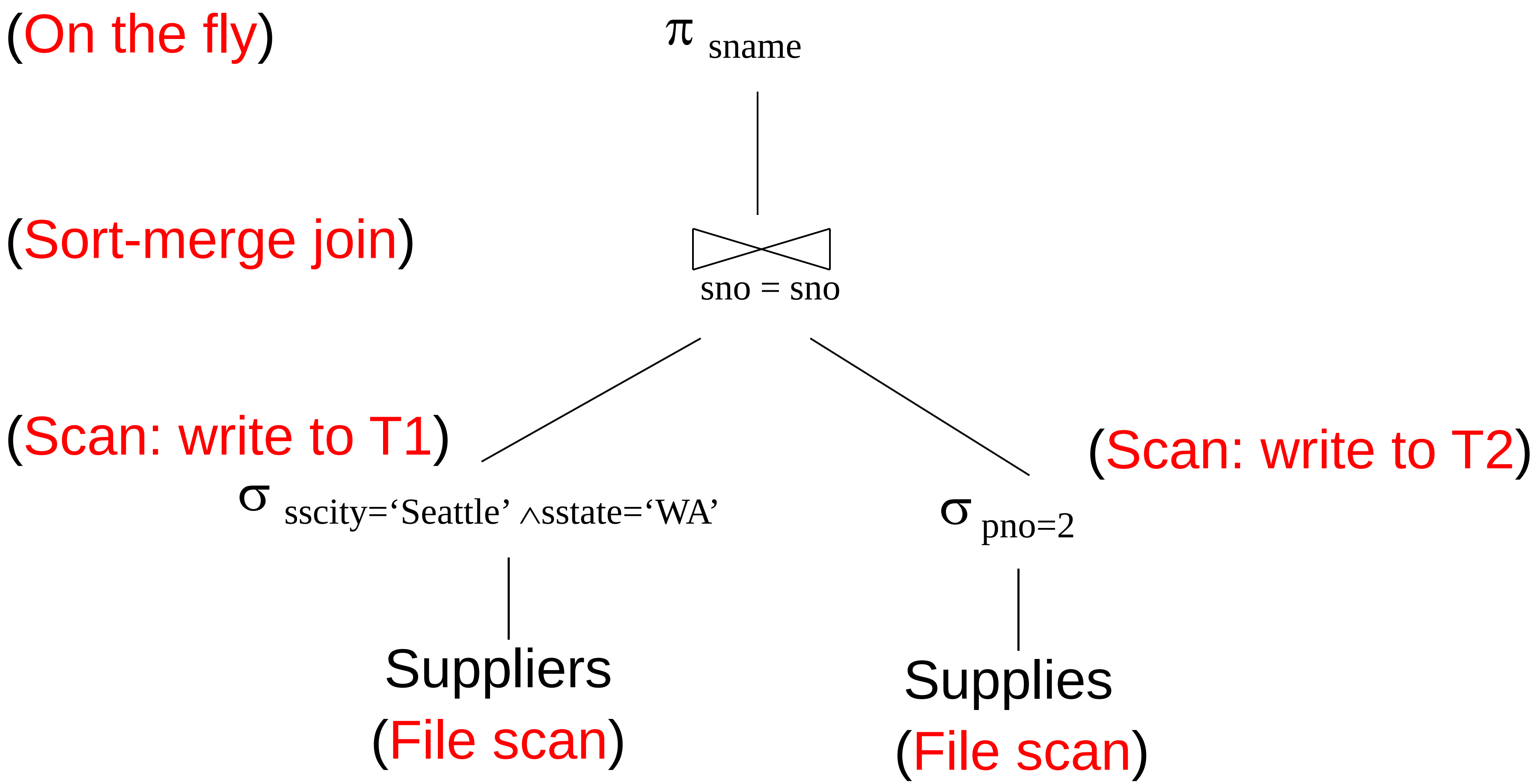
Supplies (File scan)

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



(On the fly)

π_{sname}

(Sort-merge join)

$\bowtie_{\text{sno = sno}}$

(Scan: write to T1)

$\sigma_{\text{sscity='Seattle'} \wedge \text{sstate='WA'}}$

(Scan: write to T2)

$\sigma_{\text{pno=2}}$

Suppliers

(File scan)

Supplies

(File scan)

Coming Next…

- Algorithms for physical operator implementations
- Finding a good query plan. How?