Introduction to Data Management
CSE 344

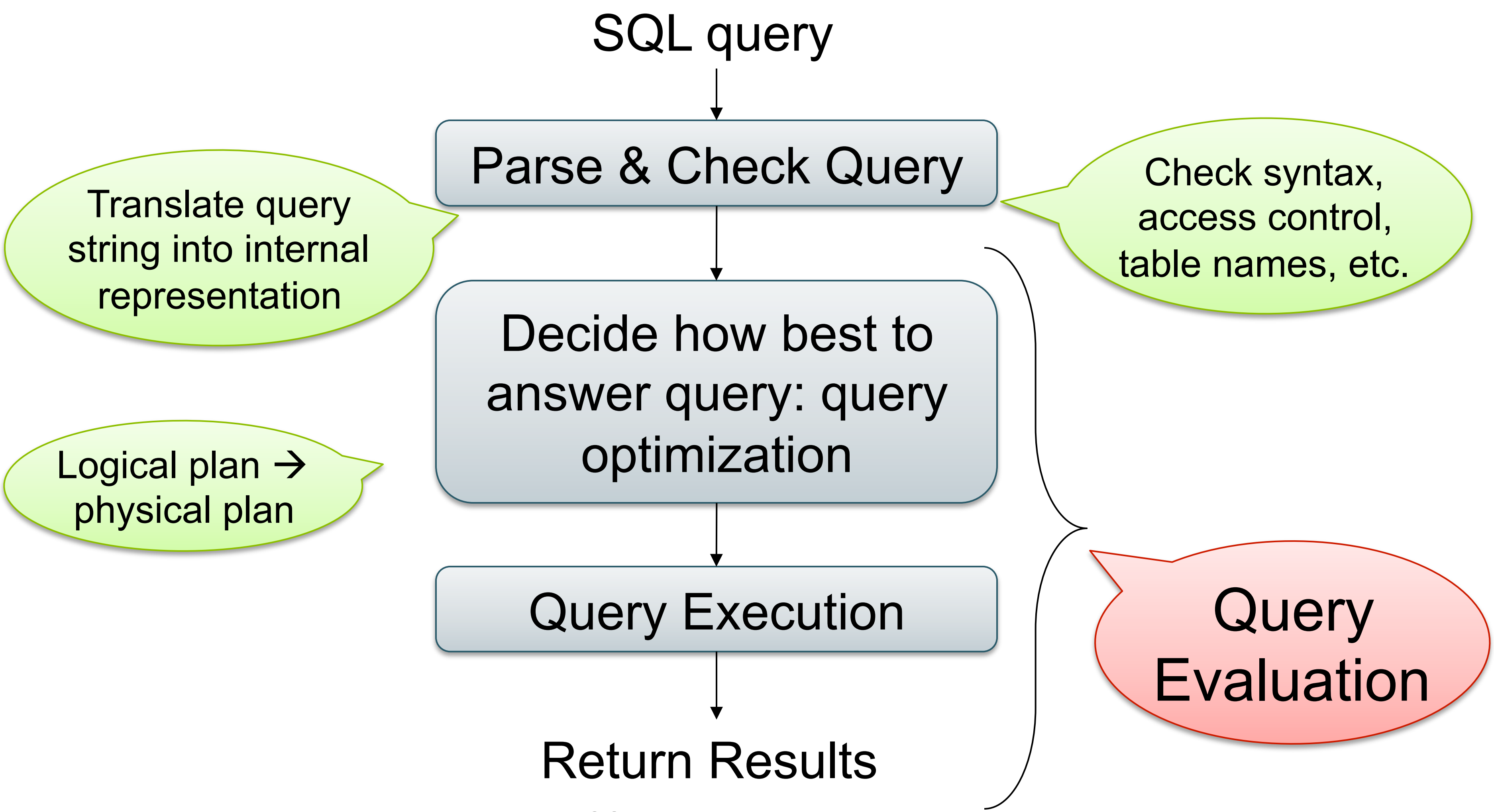
Lectures 9: Relational Algebra (part 2) and Query Evaluation

Guest lecturer: Laurel Orr

Announcements

- HW3 is due next Tuesday

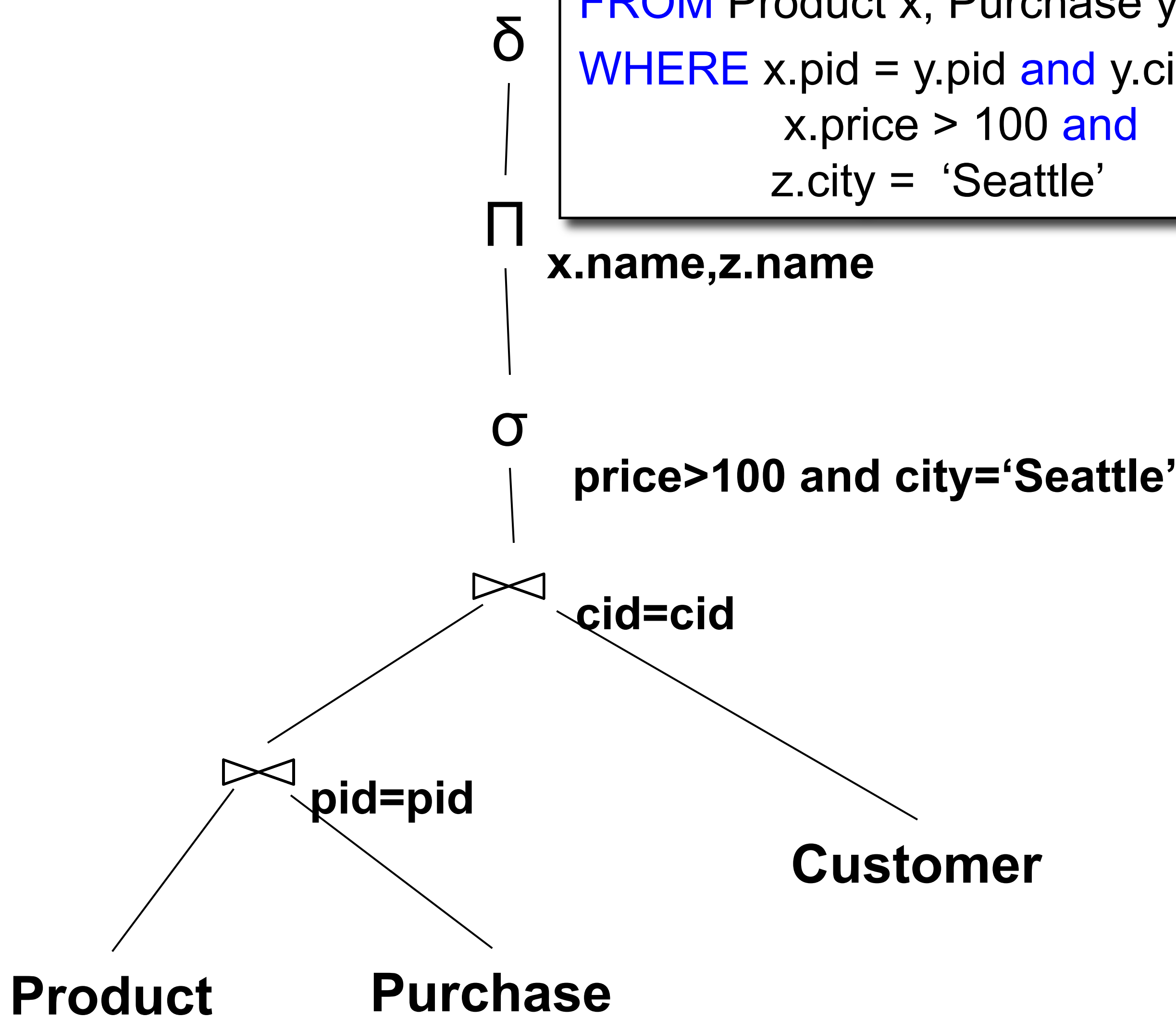
Query Evaluation Steps



Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

From SQL to RA

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
            x.price > 100 and
            z.city =  'Seattle'
```

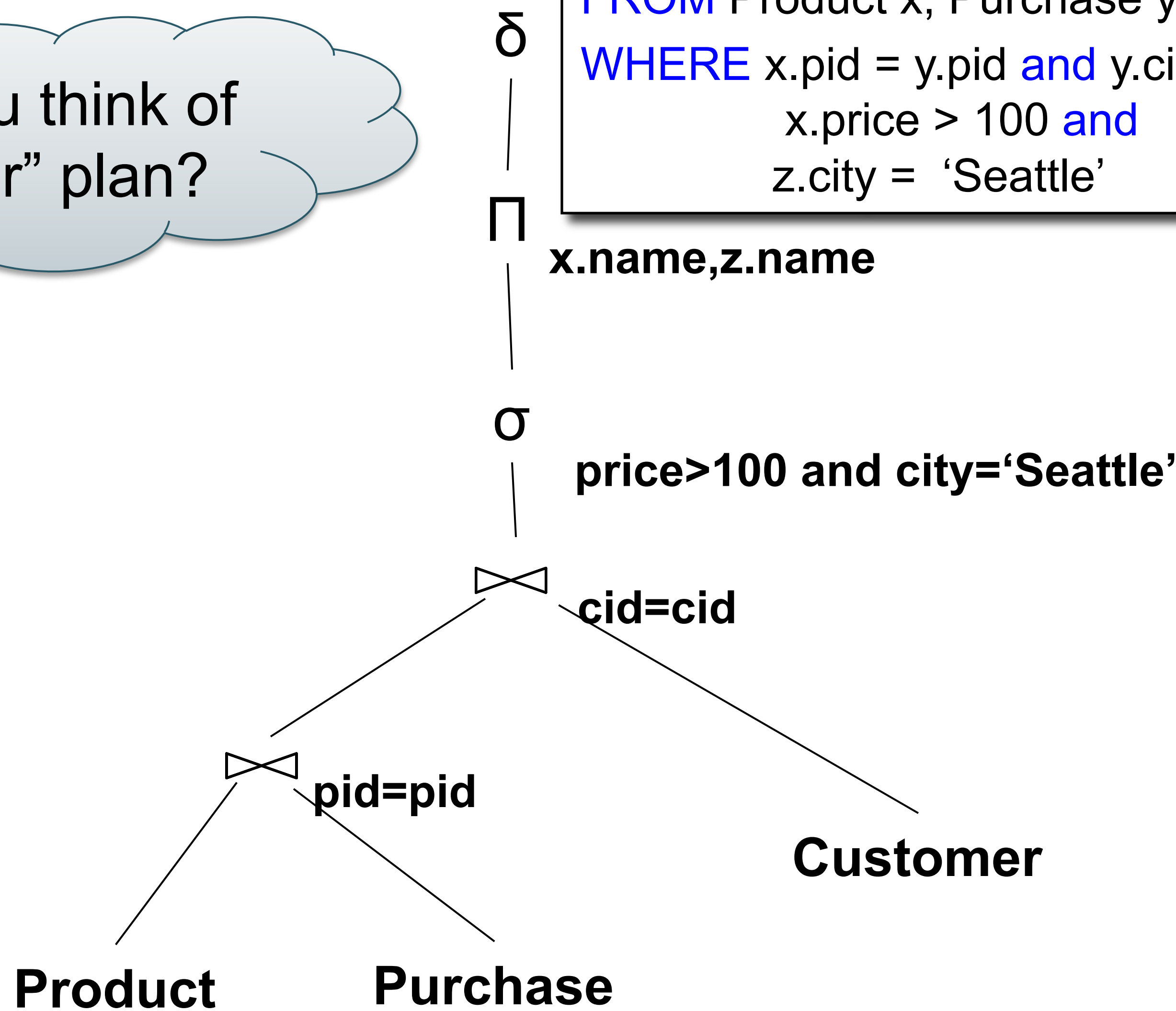


Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

From SQL to RA

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
           x.price > 100 and
           z.city =  'Seattle'
```

Can you think of a "better" plan?

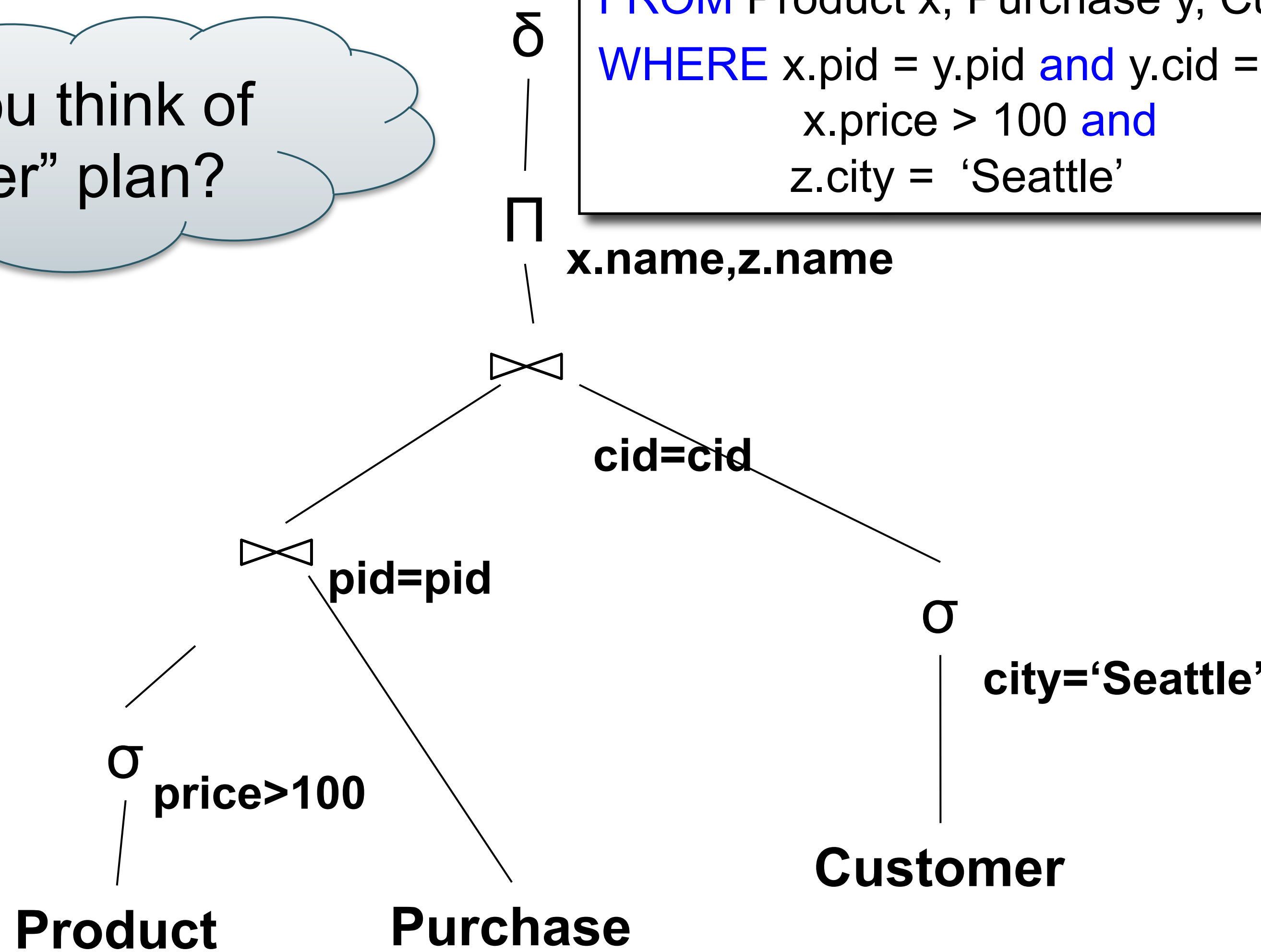


Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

Equivalent Expression

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
            x.price > 100 and
            z.city =  'Seattle'
```

Can you think of a “better” plan?



Query optimization = finding cheaper, equivalent expressions

Extended RA: Operators on Bags

- Duplicate elimination δ
- Grouping γ
- Sorting τ

Logical Query Plan

```
SELECT city, count(*)
FROM sales
GROUP BY city
HAVING sum(price) > 100
```

T3(city, c)

$\Pi_{\text{city, c}}$

T2(city,p,c)

$\sigma_{\text{p > 100}}$

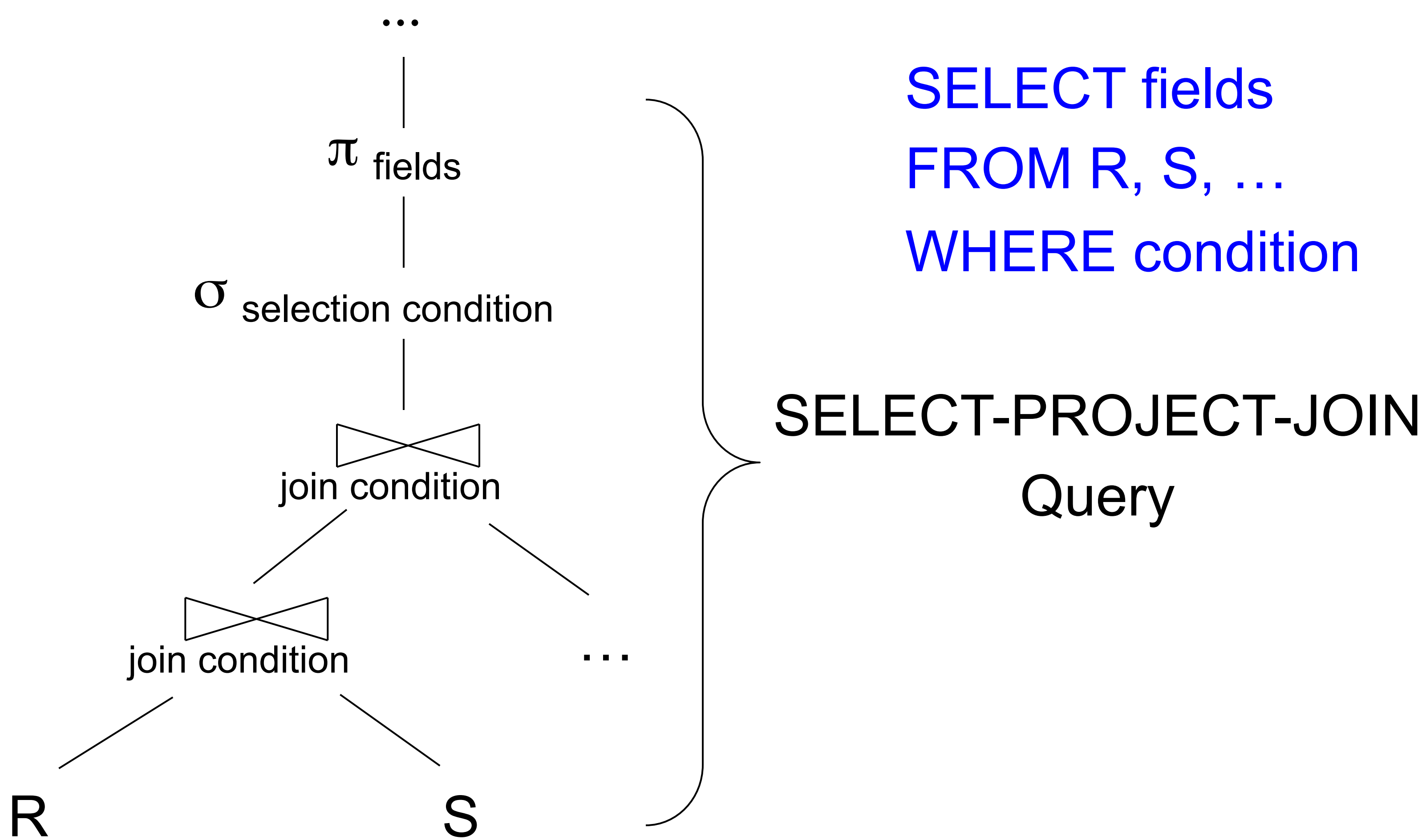
T1(city,p,c)

$\gamma_{\text{city, sum(price)} \rightarrow \text{p, count(*)} \rightarrow \text{c}}$

sales(product, city, price)

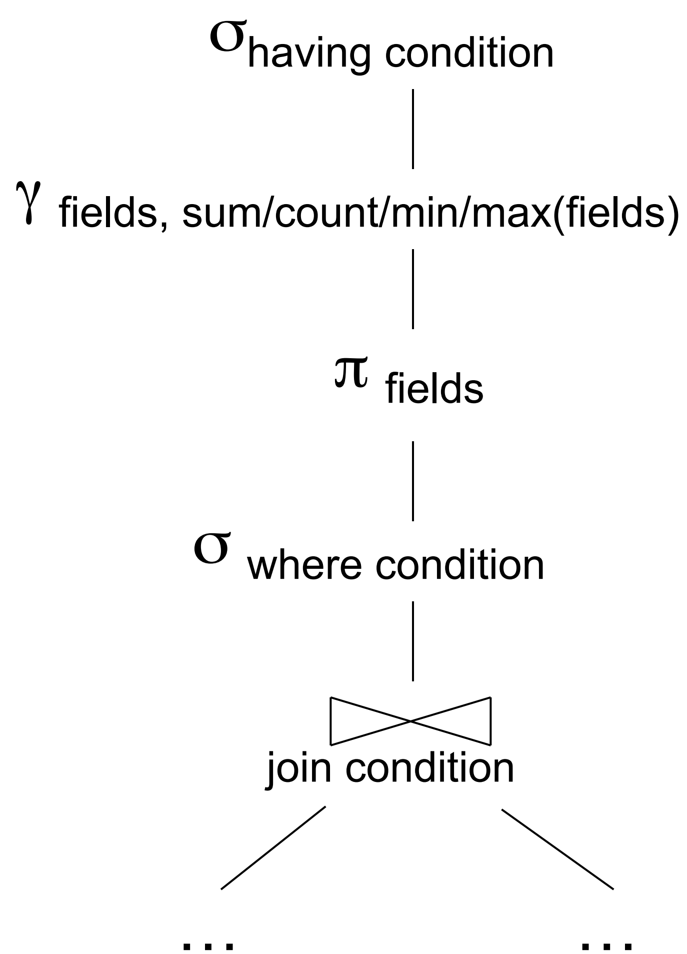
T1, T2, T3 = temporary tables

Typical Plan for Block (1/2)



SELECT fields
FROM R, S, …
WHERE condition

Typical Plan For Block (2/2)



SELECT fields
FROM R, S, …
WHERE condition
GROUP BY fields
HAVING 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?

Un-nesting

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

EXCEPT = set difference

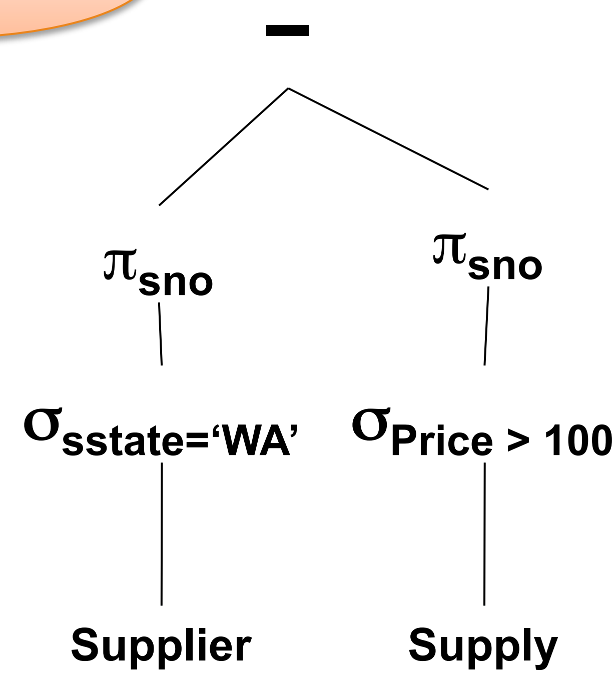
```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
    and Q.sno not in
      (SELECT P.sno
       FROM Supply P
       WHERE P.price > 100)
```

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

How about Subqueries?

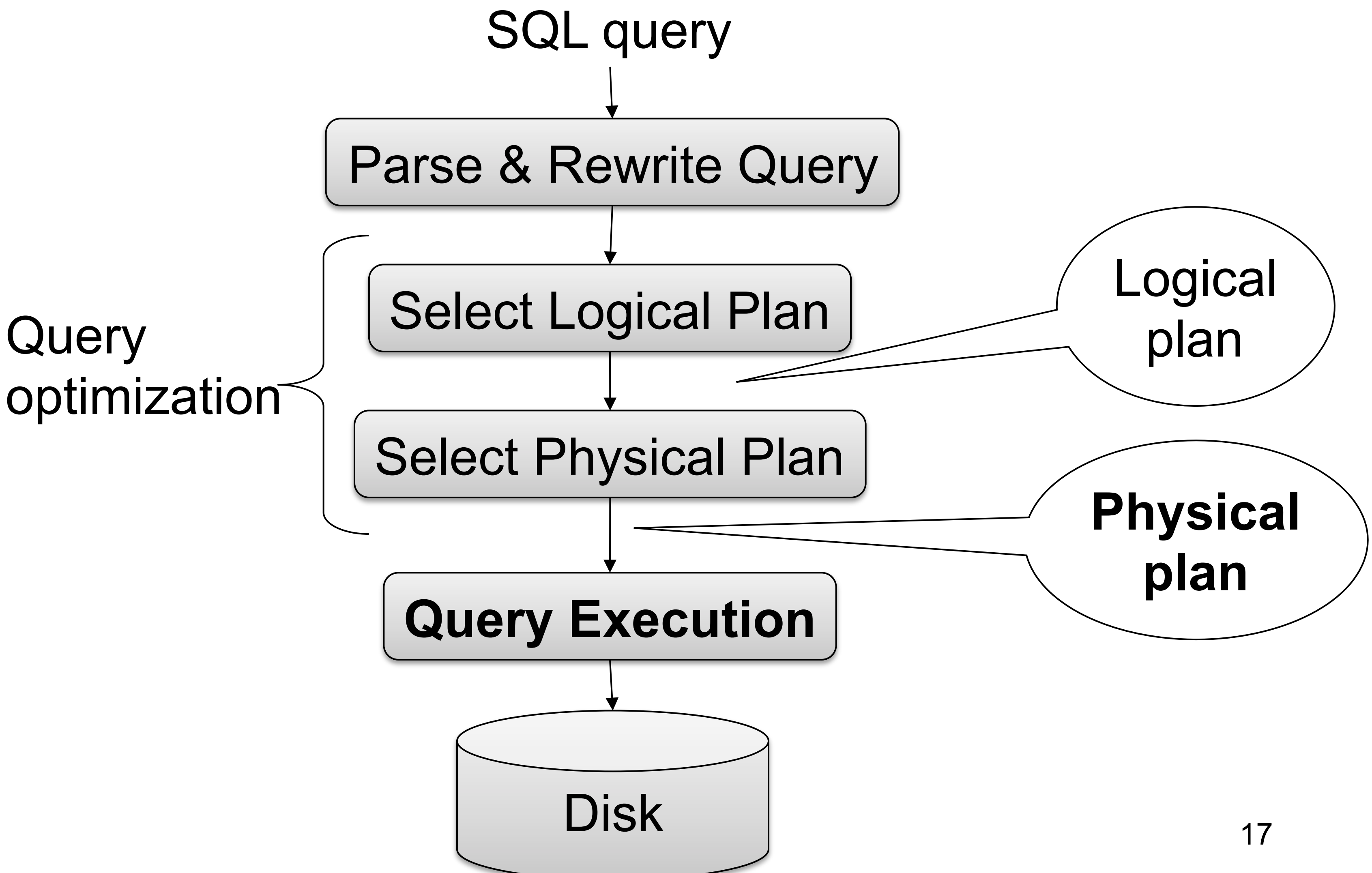
```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = 'WA')
   EXCEPT
 (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

Finally…



From Logical Plans to Physical Plans

Query Evaluation Steps Review



Relational Algebra

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and  y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```

Give a relational algebra expression for this query

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Relational Algebra

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and  y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```

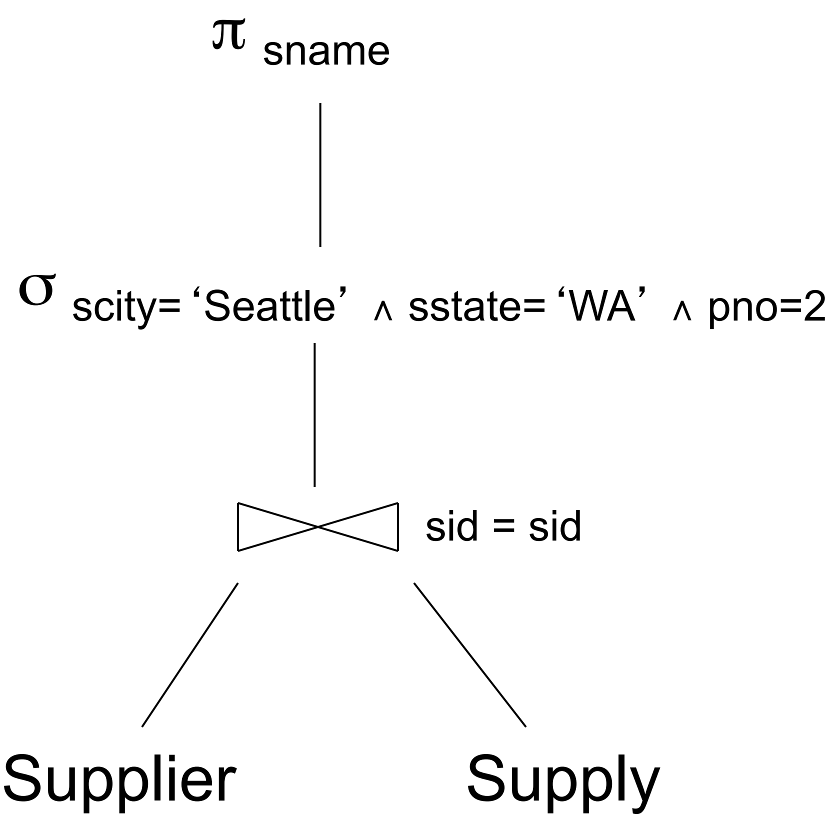
$$\pi_{\text{sname}}(\sigma_{\text{scity='Seattle'} \wedge \text{sstate='WA'} \wedge \text{pno=2}}(\text{Supplier} \bowtie_{\text{sid = sid}} \text{Supply}))$$

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Relational Algebra

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and  y.pno = 2
    and x.scity =  'Seattle'
    and x.sstate =  'WA'
```

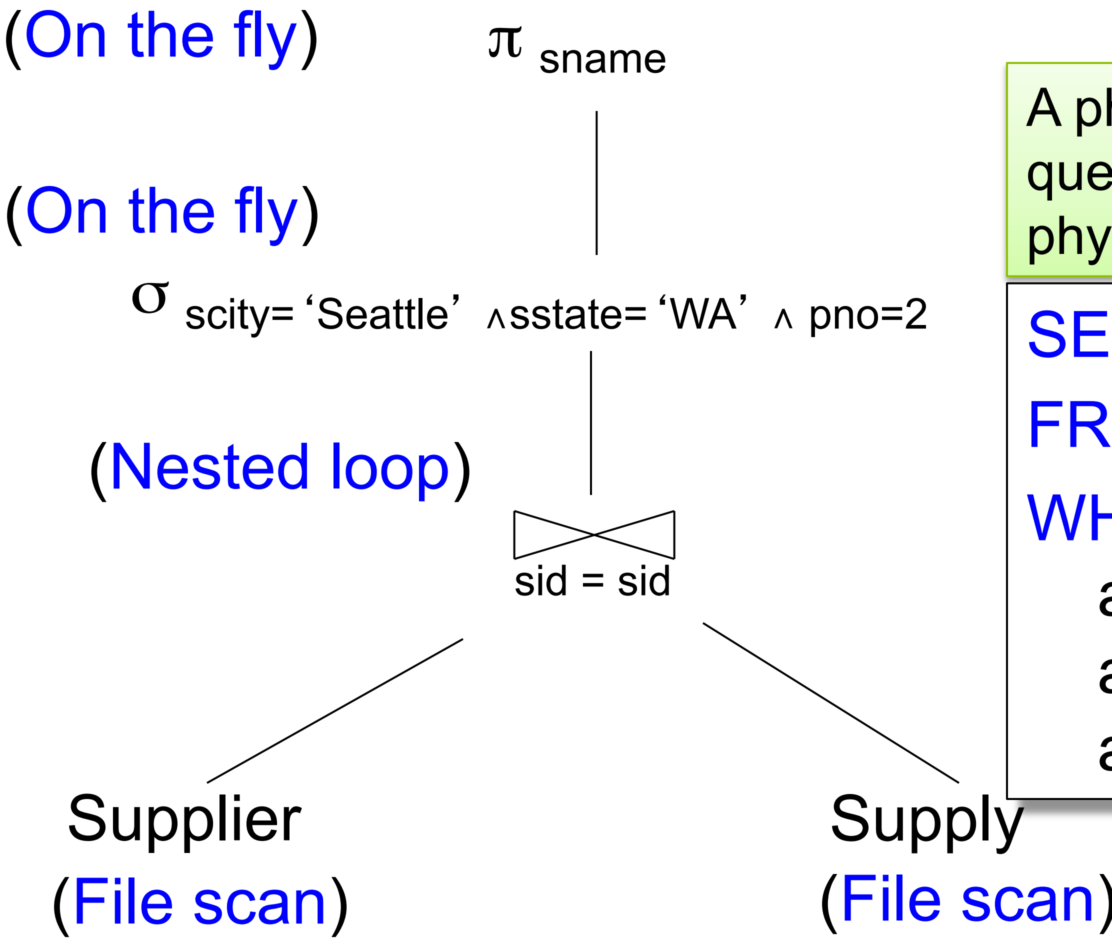


Relational algebra expression is also called the "logical query plan"

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Physical Query Plan 1



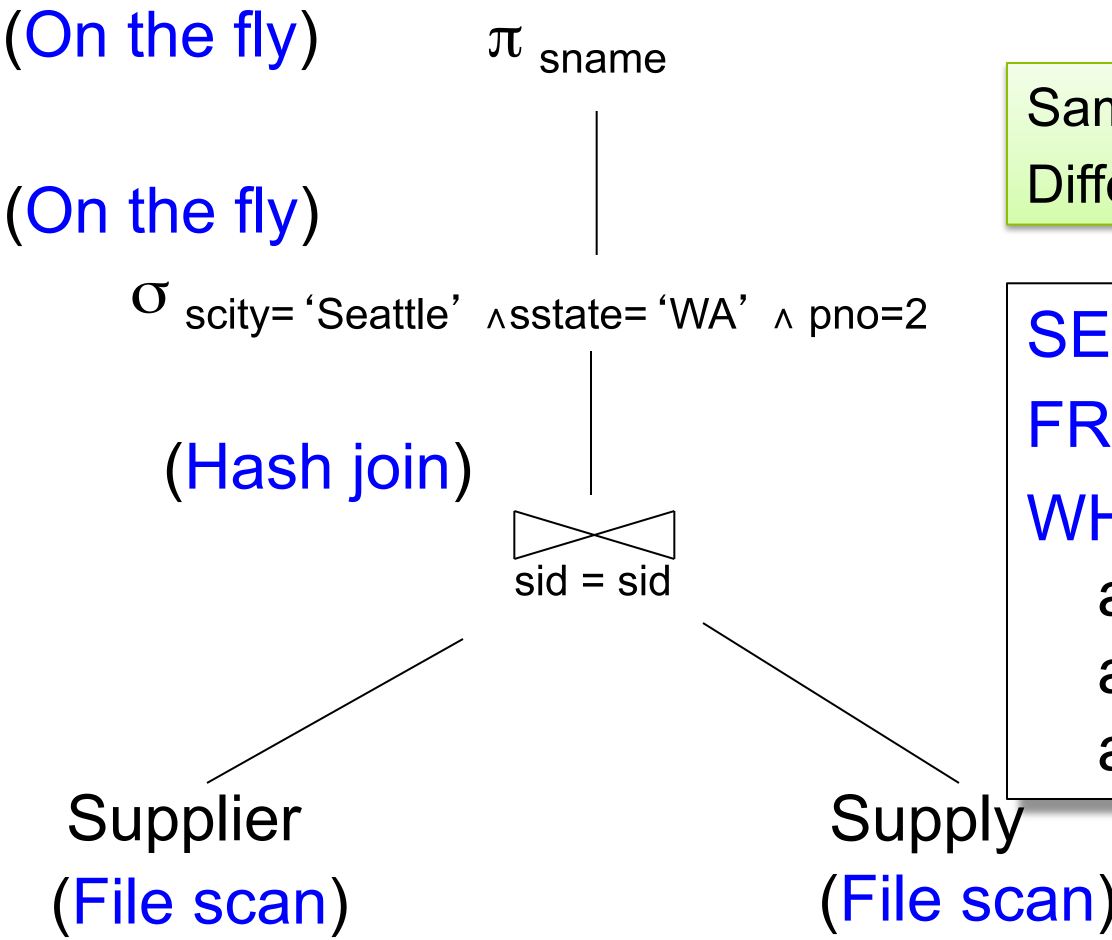
A physical query plan is a logical query plan annotated with physical implementation details

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and  y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Physical Query Plan 2



Same logical query plan

Different physical plan

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and  y.pno = 2
   and x.scity =  'Seattle'
   and x.sstate =  'WA'
```

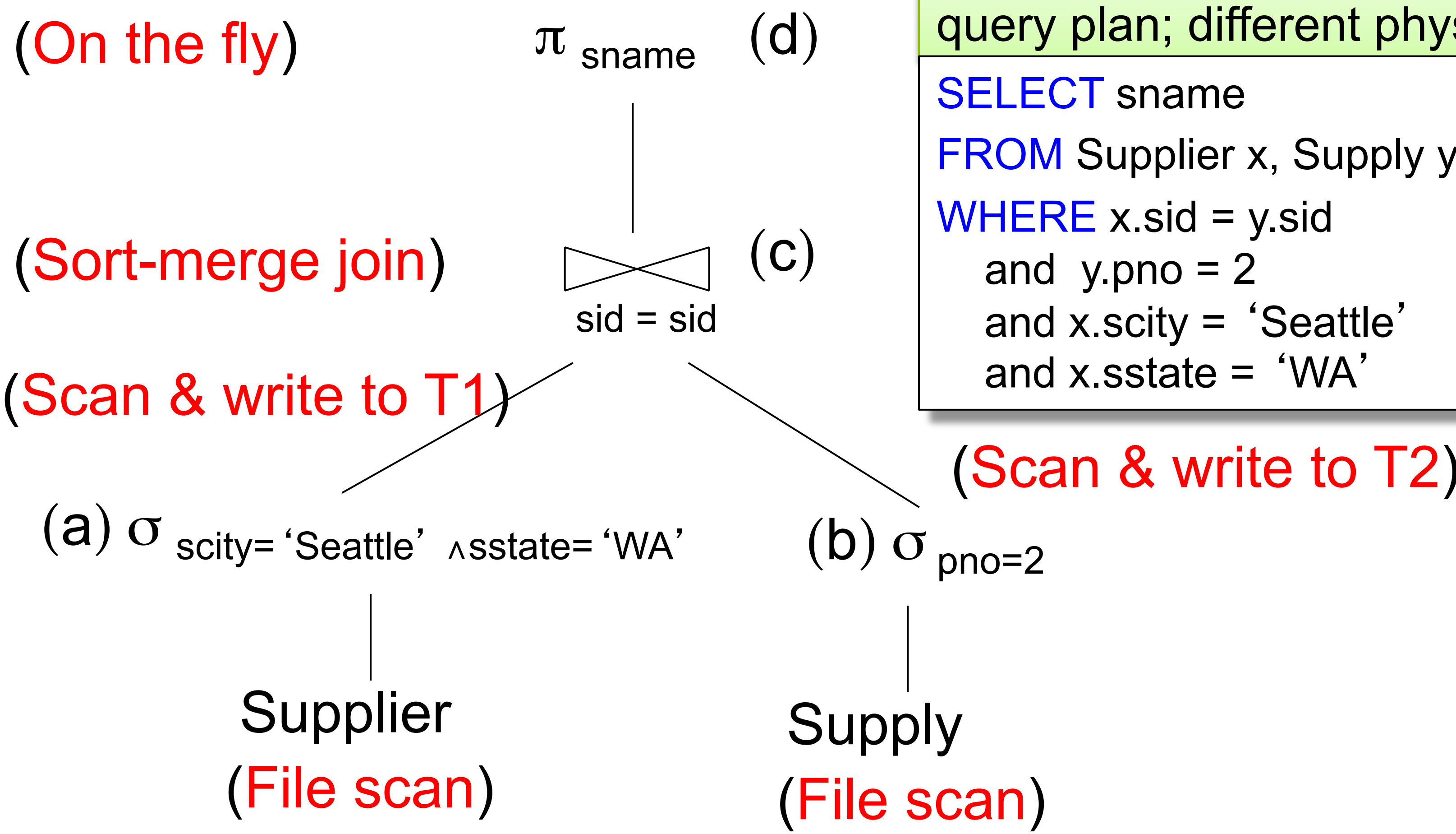
Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Physical Query Plan 3

Different but equivalent logical query plan; different physical plan

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and  y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```



Query Optimization Problem

- For each SQL query… many logical plans
- For each logical plan… many physical plans
- How do find a fast physical plan?
  - Will discuss in a few lectures

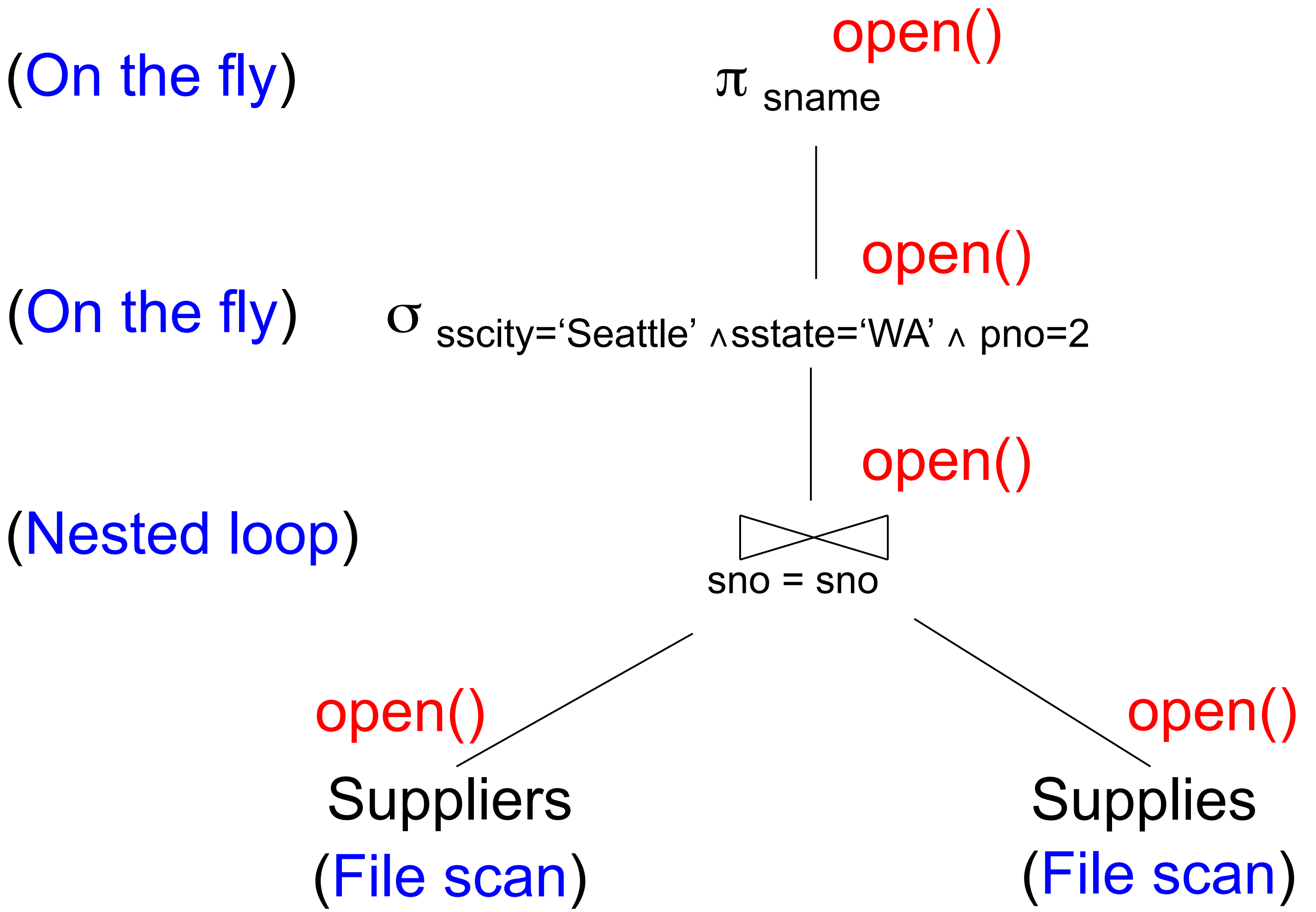
Demonstration with SQL Server Management Studio

Query Execution

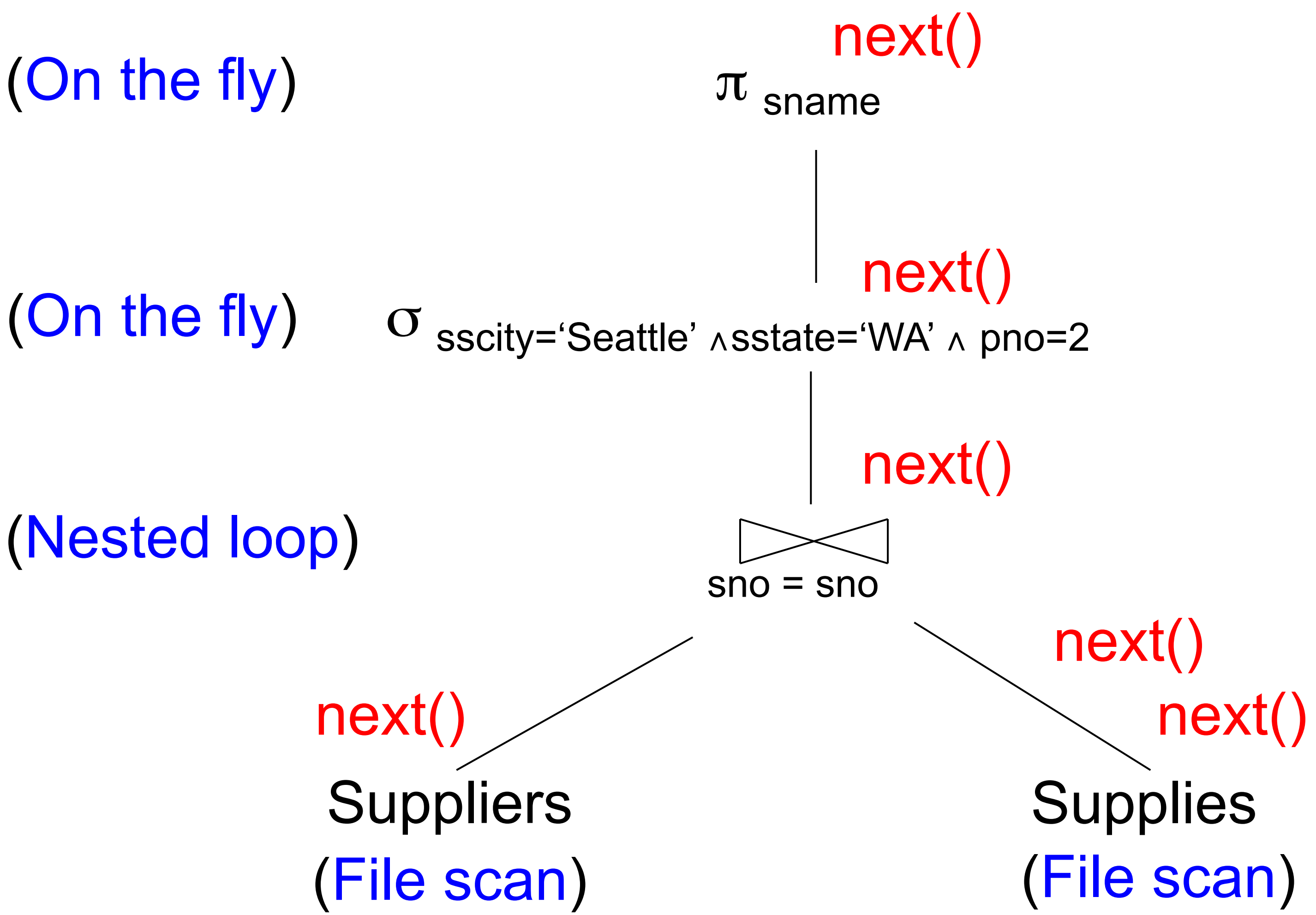
Iterator Interface

- **open()**
  - Initializes operator state
  - Sets parameters such as selection condition
- **next()**
  - Operator invokes get_next() recursively on its inputs
  - Performs processing and produces an output tuple
- **close()**: clean-up state

Pipelined Query Execution



Pipelined Query Execution



Pipelined Execution

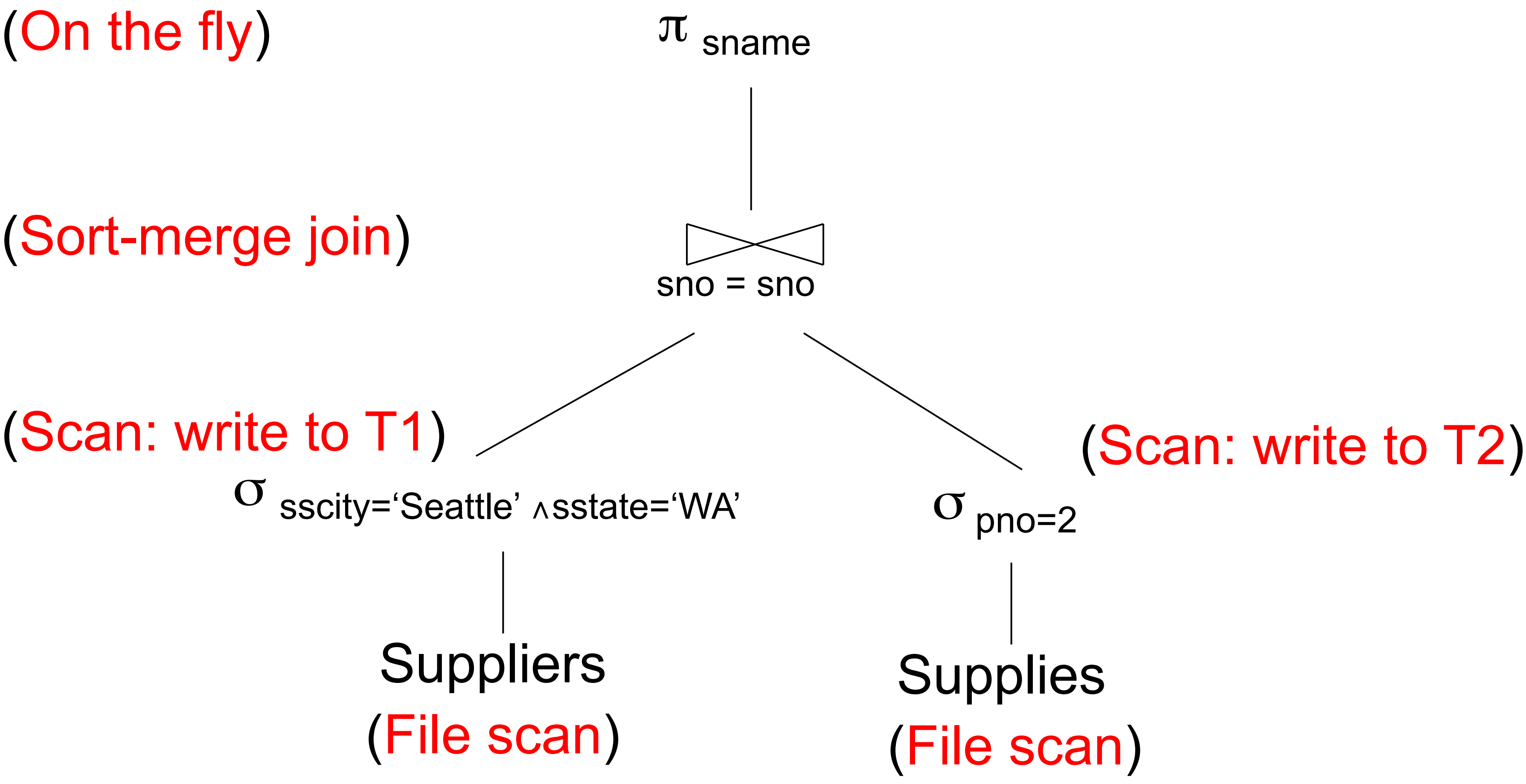
- Tuples generated by an operator are immediately sent to the parent
- Benefits:
  - No operator synchronization issues
  - No need to buffer tuples between operators
  - Saves cost of writing intermediate data to disk
  - Saves cost of reading intermediate data from disk
- This approach is used whenever possible

Intermediate Tuple Materialization

- Tuples generated by an operator are written to disk an in intermediate table

- No direct benefit
- Necessary:
  – For certain operator implementations
  – When we don't have enough memory

Intermediate Tuple Materialization



Query Execution Bottom Line

- SQL query transformed into physical plan
  - **Access path selection** for each relation
    - Scan the relation or use an index (see next lecture)
  - **Implementation choice** for each operator
    - Nested loop join, hash join, etc.
  - **Scheduling decisions** for operators
    - Pipelined execution or intermediate materialization
- Execution of the physical plan is pull-based

Physical Data Independence

- Means that applications are insulated from changes in physical storage details
  - E.g., can add/remove indexes without changing apps
  - Can do other physical tunings for performance

- SQL and relational algebra facilitate physical data independence because both languages are “set-at-a-time”: Relations as input and output