Introduction to Data Management CSE 344

Lecture 9: Relational Algebra and Query Evaluation

Today

- Relational algebra
- Physical plans and query evaluation

Relational Algebra Operators

- Union ∪, intersection ∩, difference -
- Selection σ
- Projection π
- Cartesian product X, join ⋈
- Rename ρ
- Duplicate elimination δ
- Grouping and aggregation γ
- Sorting τ

RA

Extended RA

All operators take in 1 or more relations as inputs and return another relation

Join Summary

- **Theta-join**: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
  - Join of R and S with a join condition θ
  - Cross-product followed by selection θ
- **Equijoin**: $R \bowtie_{\theta} S = \pi_A (\sigma_{\theta} (R \times S))$
  - Join condition θ consists only of equalities
  - Projection π_A drops all redundant attributes
- **Natural join**: $R \bowtie S = \pi_A (\sigma_{\theta} (R \times S))$
  - Equality on **all** fields with same name in R and in S
  - Projection π_A drops all redundant attributes

So Which Join Is It ?

When we write $R \bowtie S$ we usually mean an equijoin, but we often omit the equality predicate when it is clear from the context

More Joins

- **Outer join**
  - Include tuples with no matches in the output
  - Use NULL values for missing attributes
  - Does not eliminate duplicate columns

- Variants
  - Left outer join
  - Right outer join
  - Full outer join

Outer Join Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu
33	98120	lung

AnnonJob J

job	age	zip
lawyer	54	98125
cashier	20	98120

$P ⟕ J$

P.age	P.zip	disease	job	J.age	J.zip
54	98125	heart	lawyer	54	98125
20	98120	flu	cashier	20	98120
33	98120	lung	null	null	null

Some Examples

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

Name of supplier of parts with size greater than 10

$\pi_{sname}(\text{Supplier} \bowtie (\text{Supply} \bowtie (\sigma_{psize>10} (\text{Part})))$

Name of supplier of red parts or parts with size greater than 10

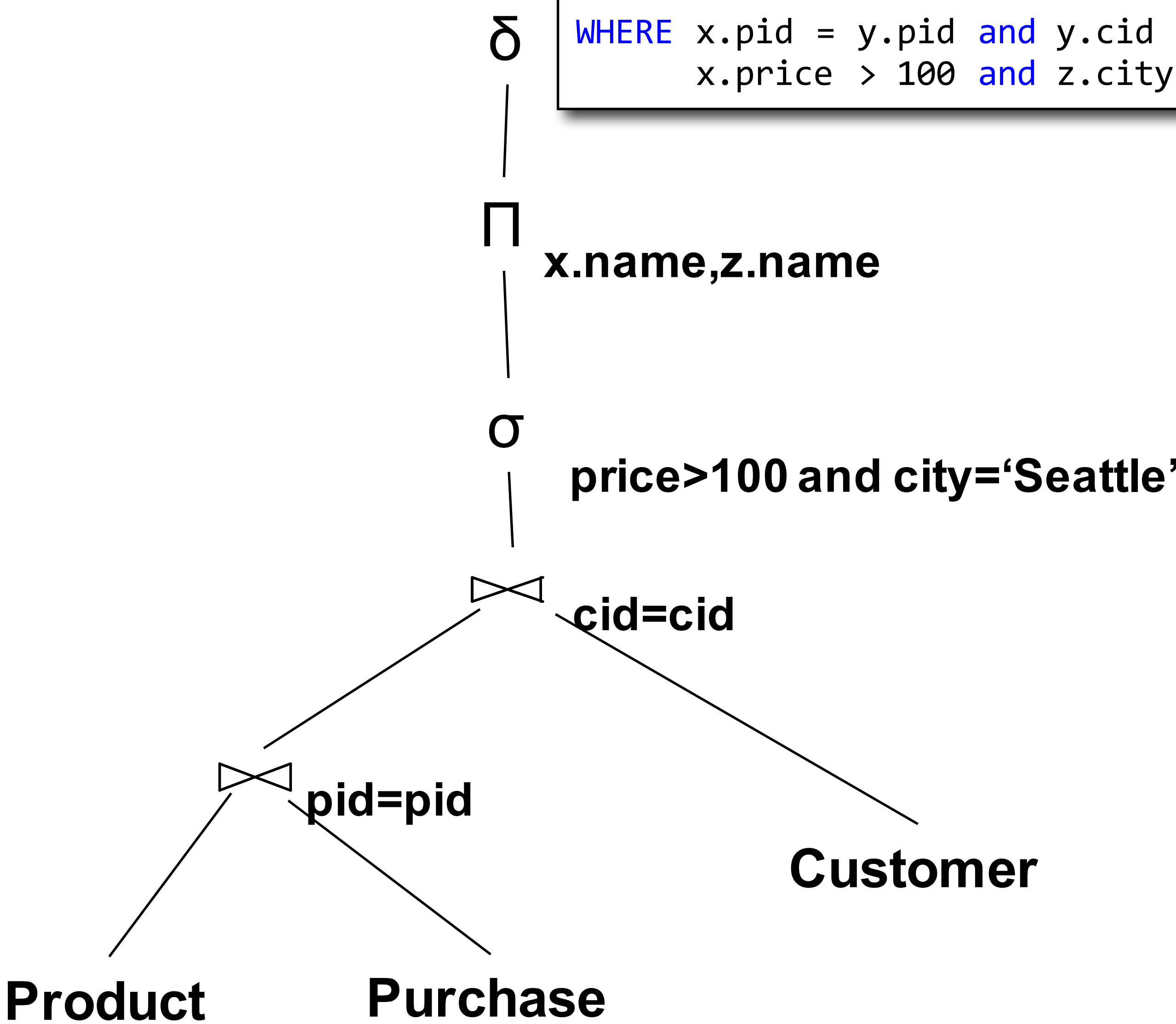
$\pi_{sname}(\text{Supplier} \bowtie (\text{Supply} \bowtie (\sigma_{psize>10} (\text{Part}) \cup \sigma_{pcolor=\text{'red'}} (\text{Part}))))$

Can be represented as trees as well (as seen from lecture 7)

```
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'
```

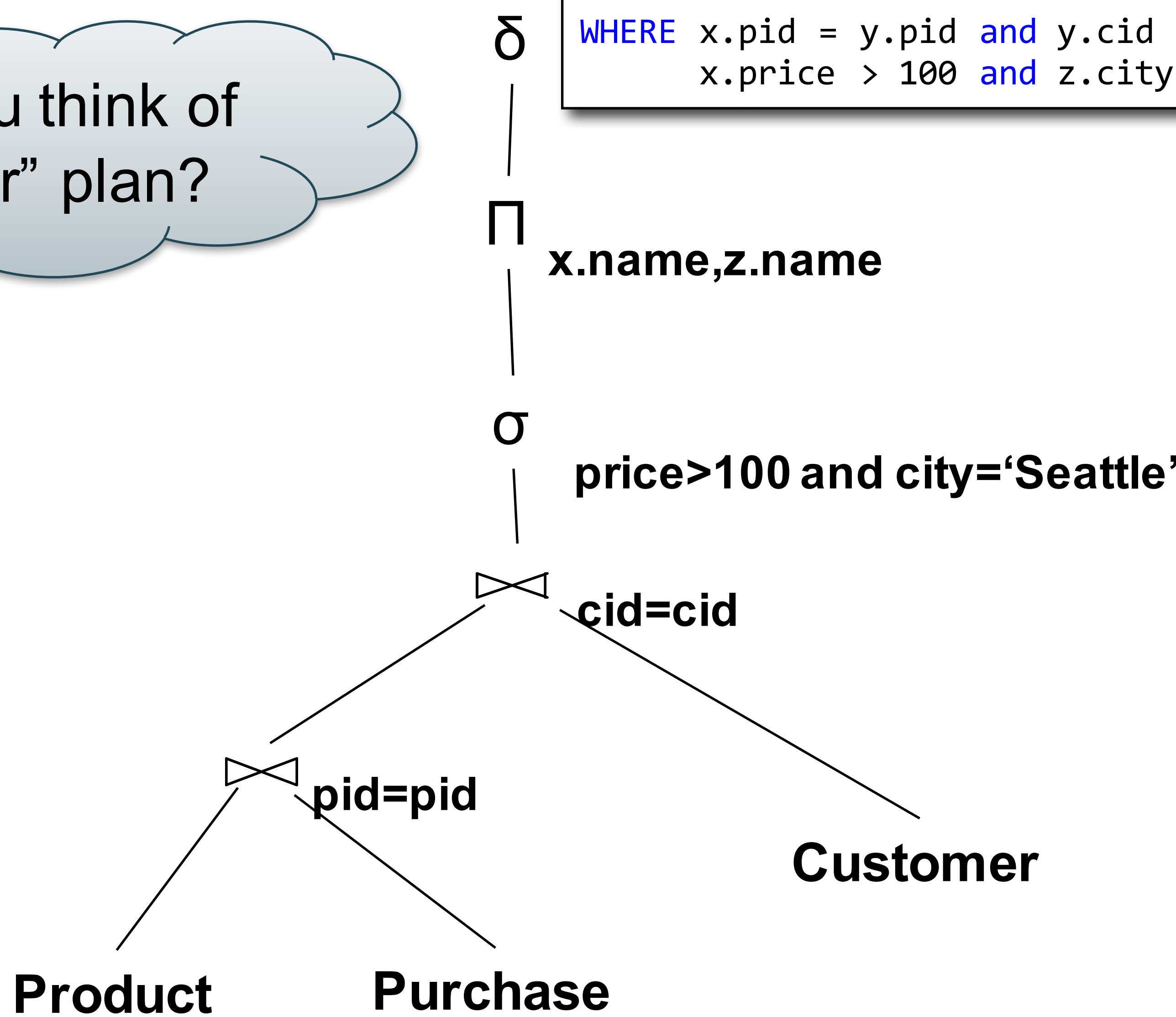


```
Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)
```

From SQL to RA

```sql
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?



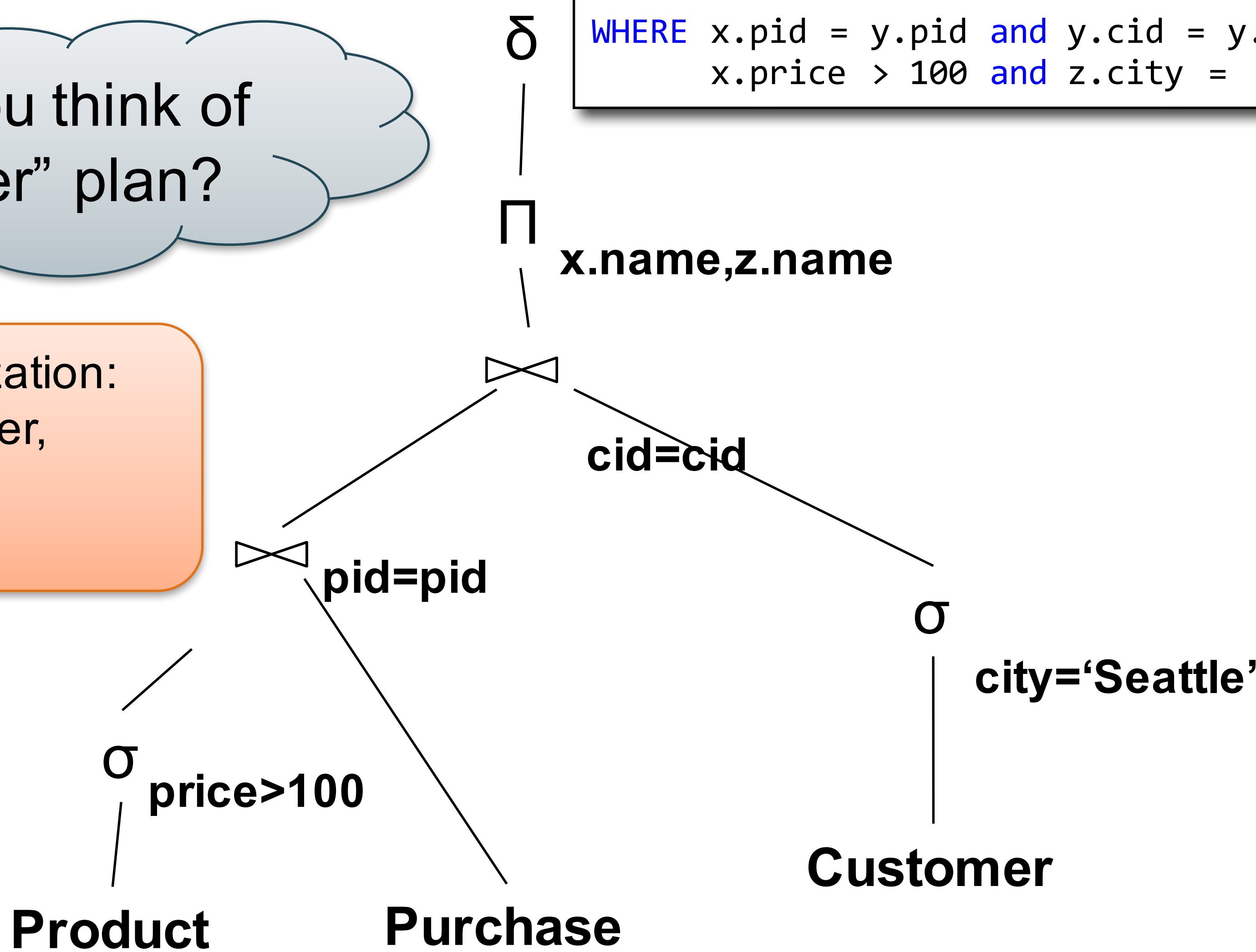
Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

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

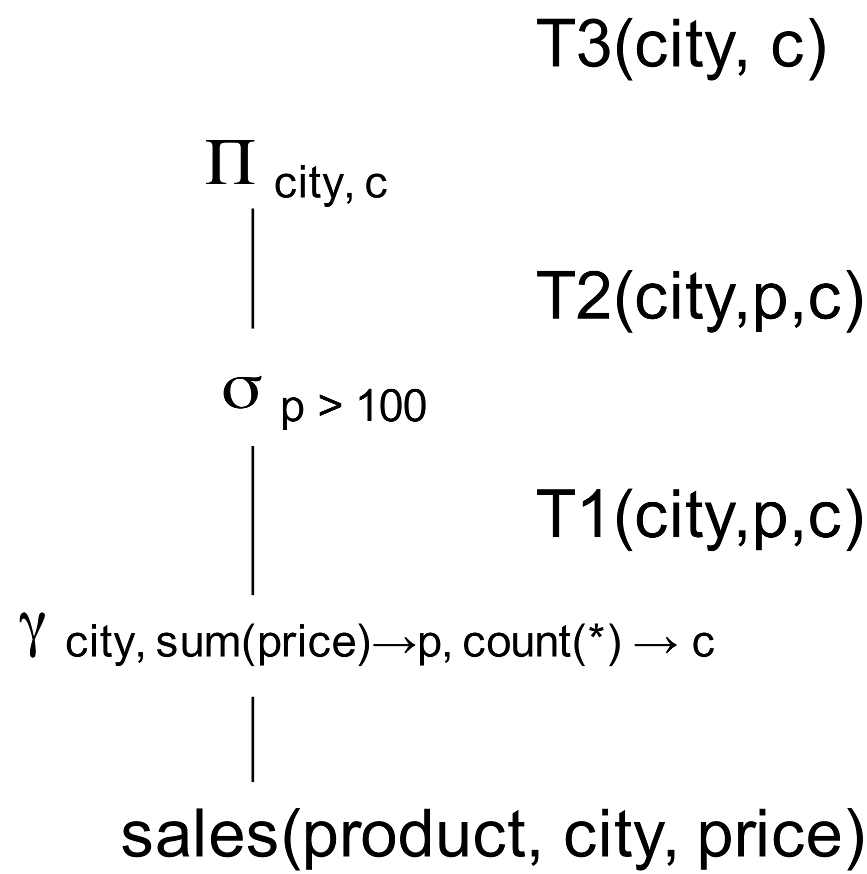


Extended RA: Operators on Bags

- Duplicate elimination δ
- Grouping γ
  - Takes in relation and a list of grouping operations (e.g., aggregates). Returns a new relation.
- Sorting τ
  - Takes in a relation, a list of attributes to sort on, and an order. Returns a new relation.

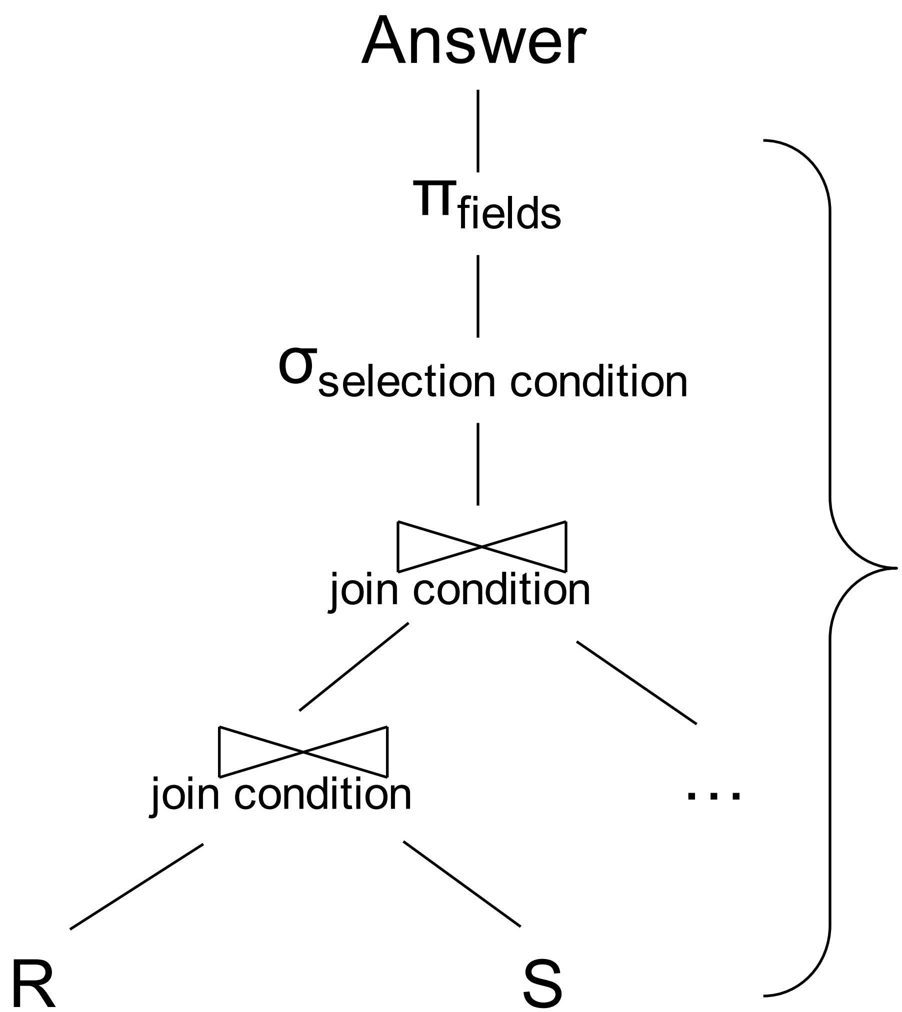
Using Extended RA Operators

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



T1, T2, T3 = temporary tables

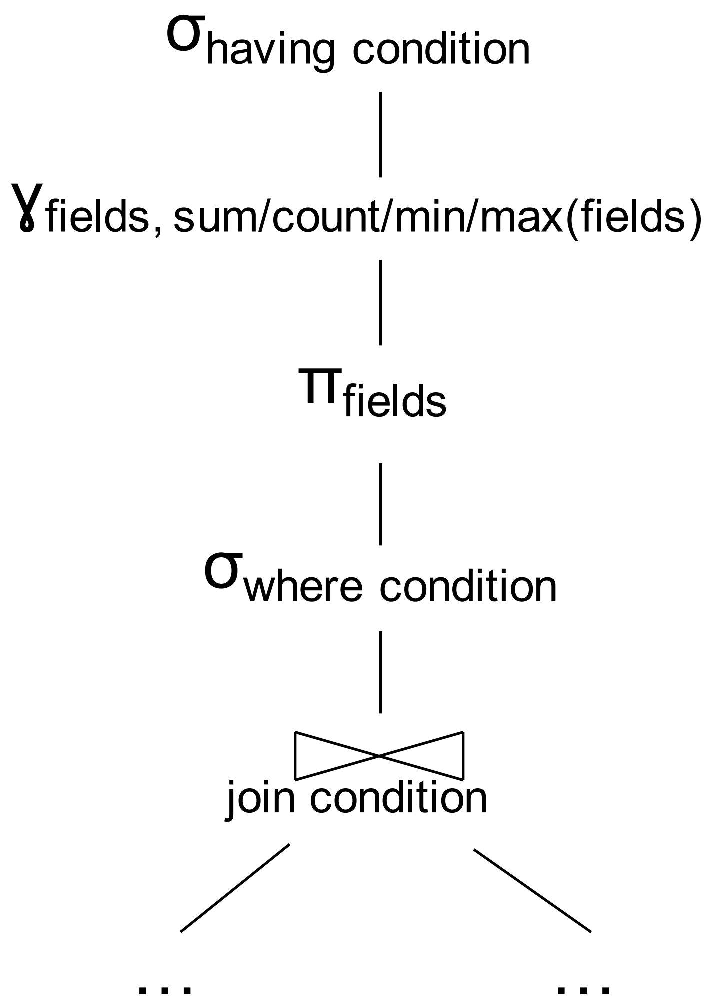
Typical Plan for a Query (1/2)




```
SELECT fields
FROM R, S, …
WHERE condition
```

SELECT-PROJECT-JOIN Query

Typical Plan for a Query (1/2)




```
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?

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

EXCEPT = set difference

Un-nesting

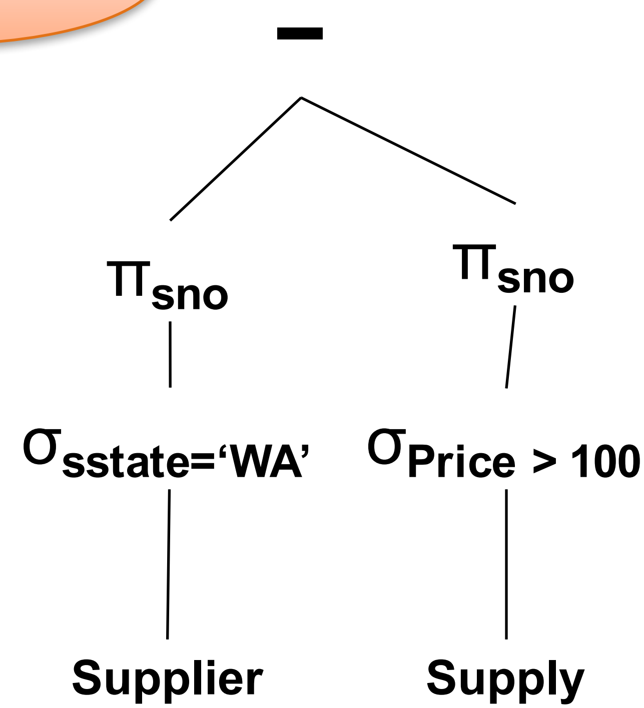
```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
  and Q.sno not in
   (SELECT P.sno
    FROM Supply P
    WHERE P.price > 100)
```

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

How about Subqueries?

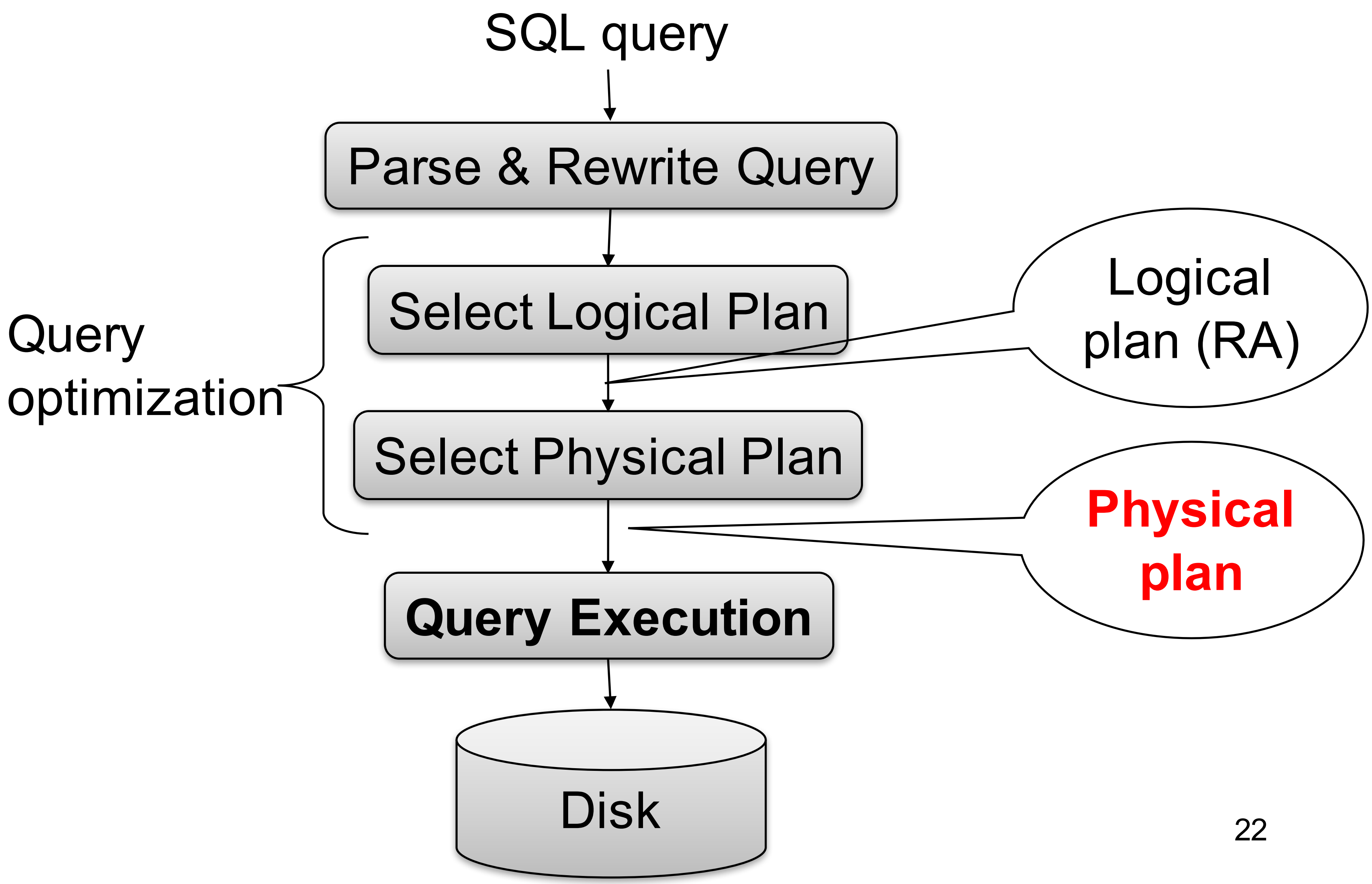
```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = 'WA')
    EXCEPT
 (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

Finally…



From Logical RA Plans to Physical Plans

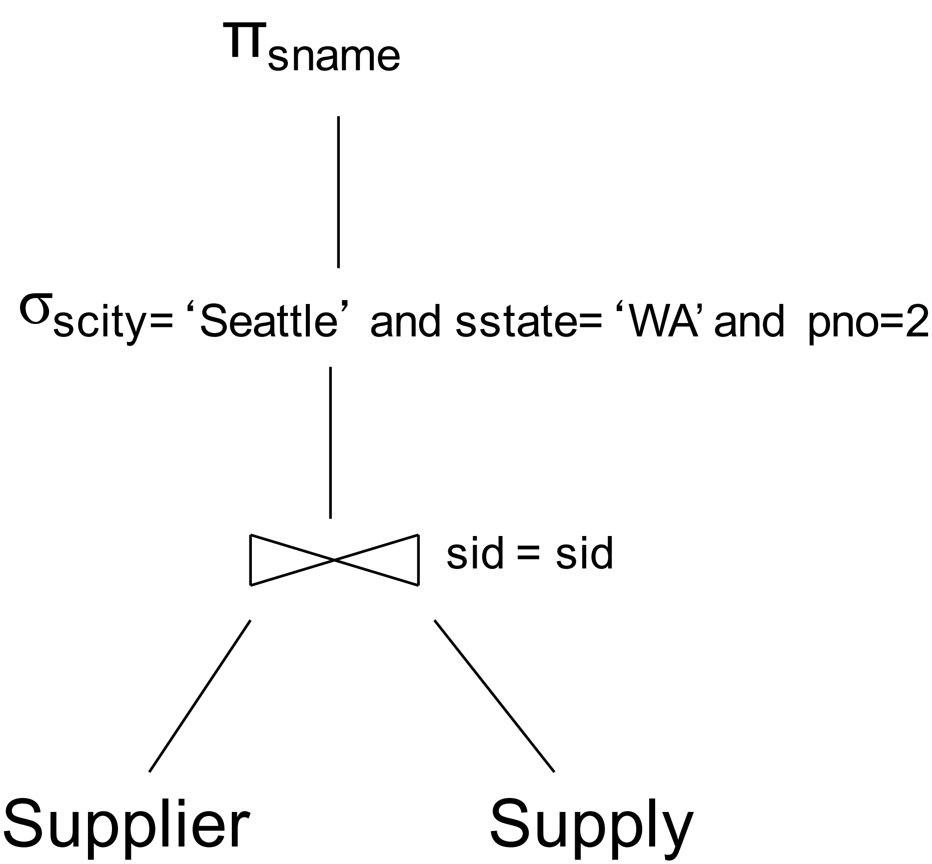
Query Evaluation Steps Review



```
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’
```

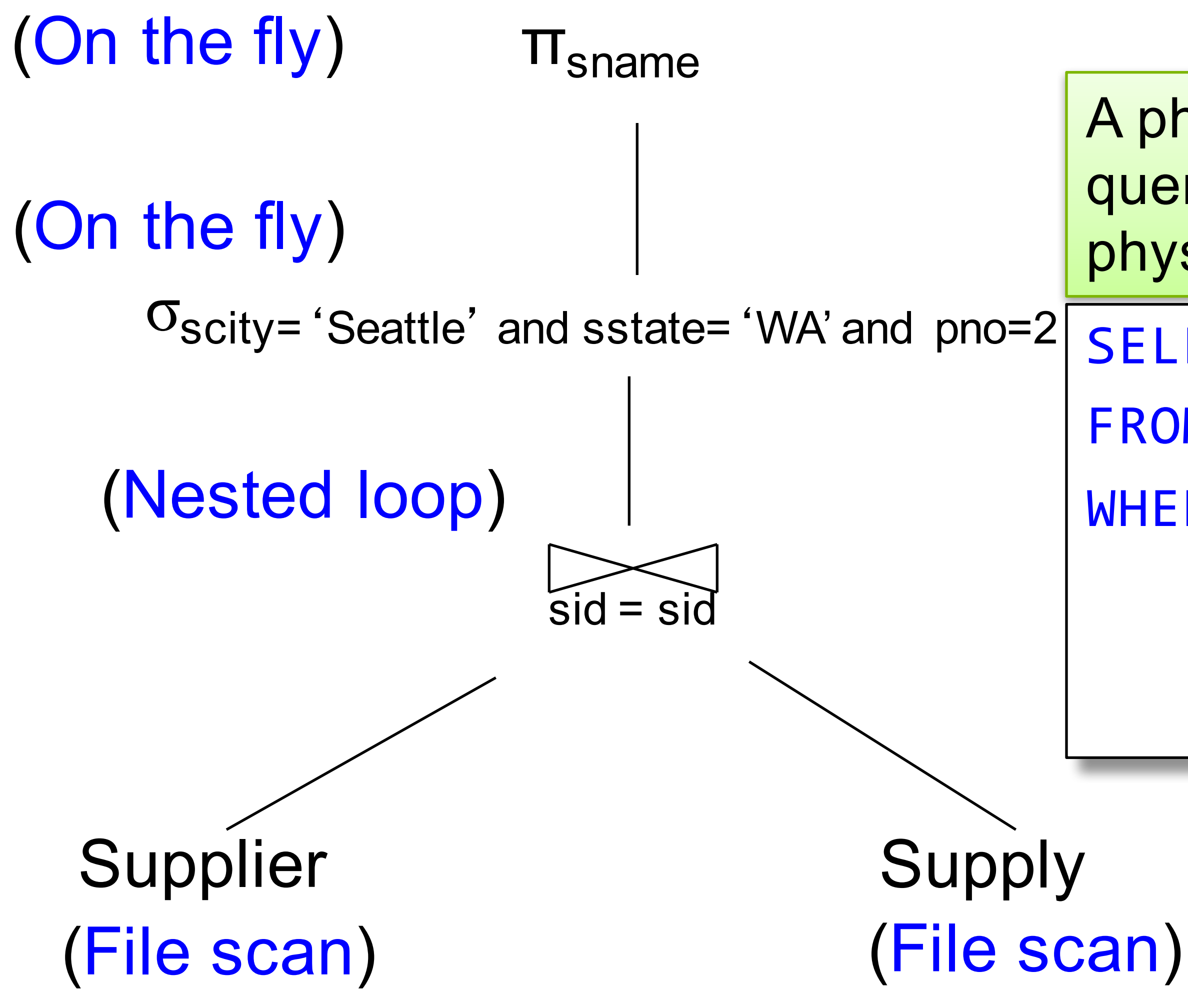


Relational algebra expression is also called the “logical query plan”

Supplier(<u>sid</u>, sname, scity, sstate)

Supply(<u>sid, pno</u>, quantity)

Physical Query Plan 1



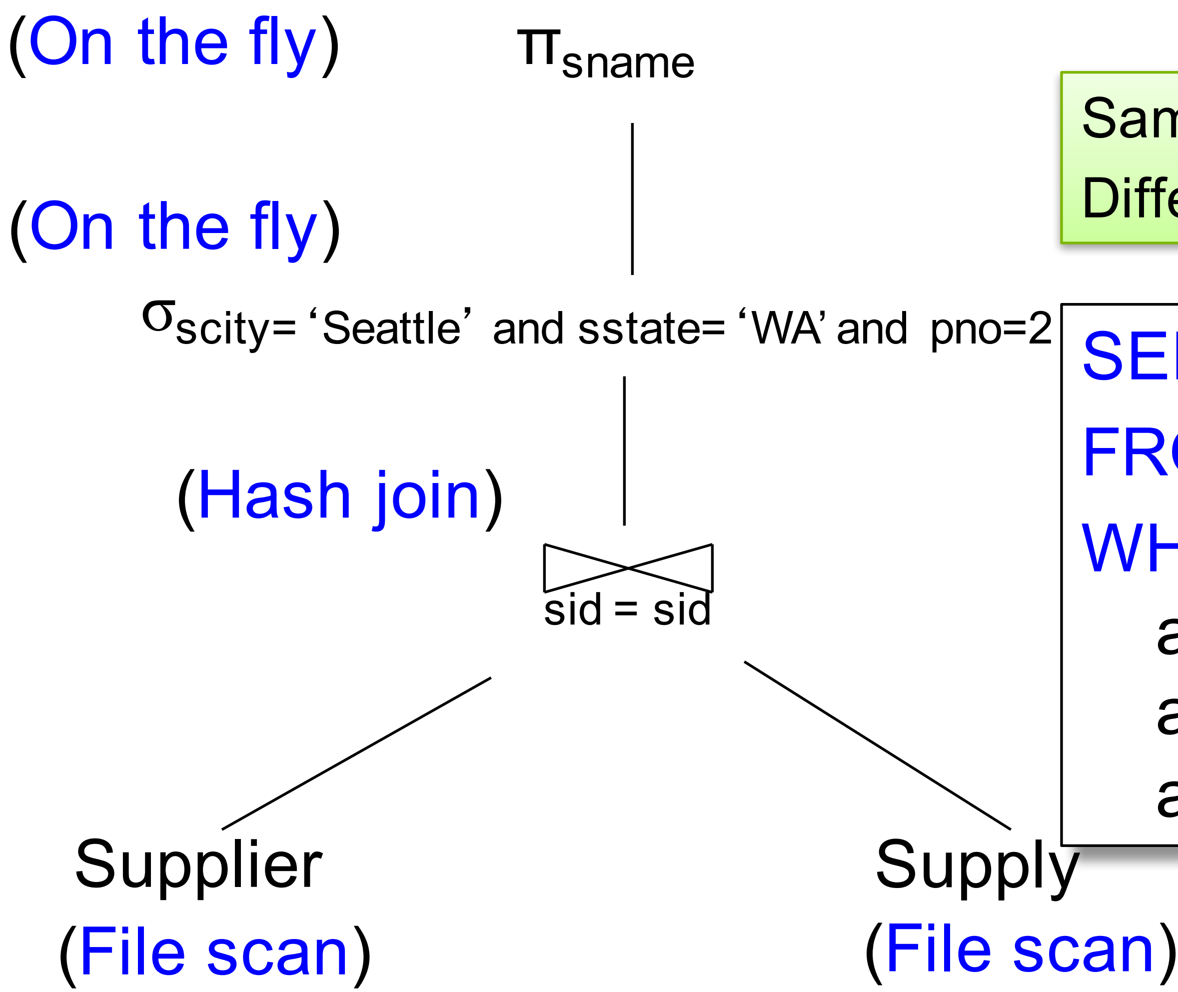
A physical query plan is a logical query plan annotated with physical implementation details

```sql
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'
```

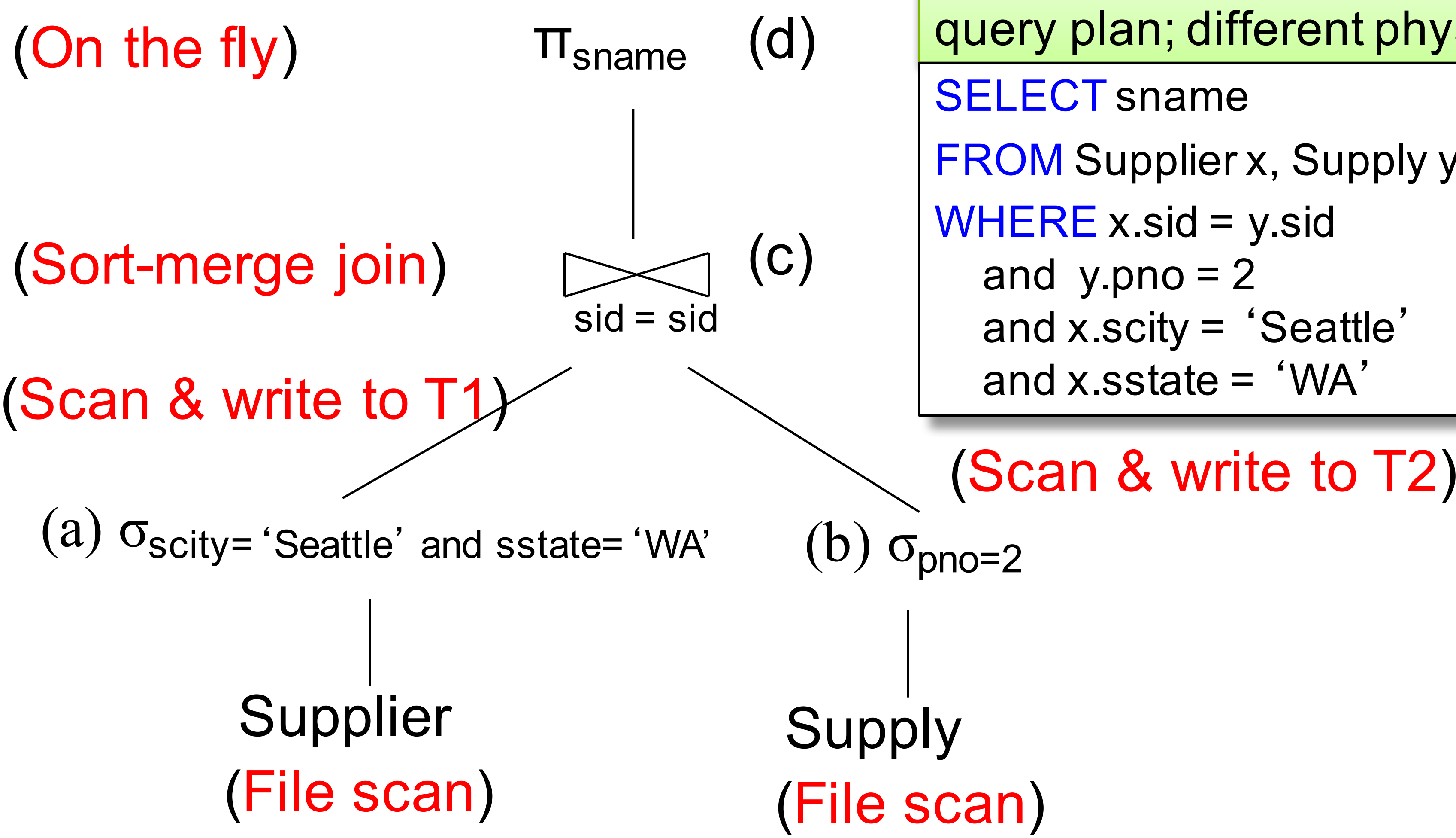
Supplier(sid, sname, scity, sstate)

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

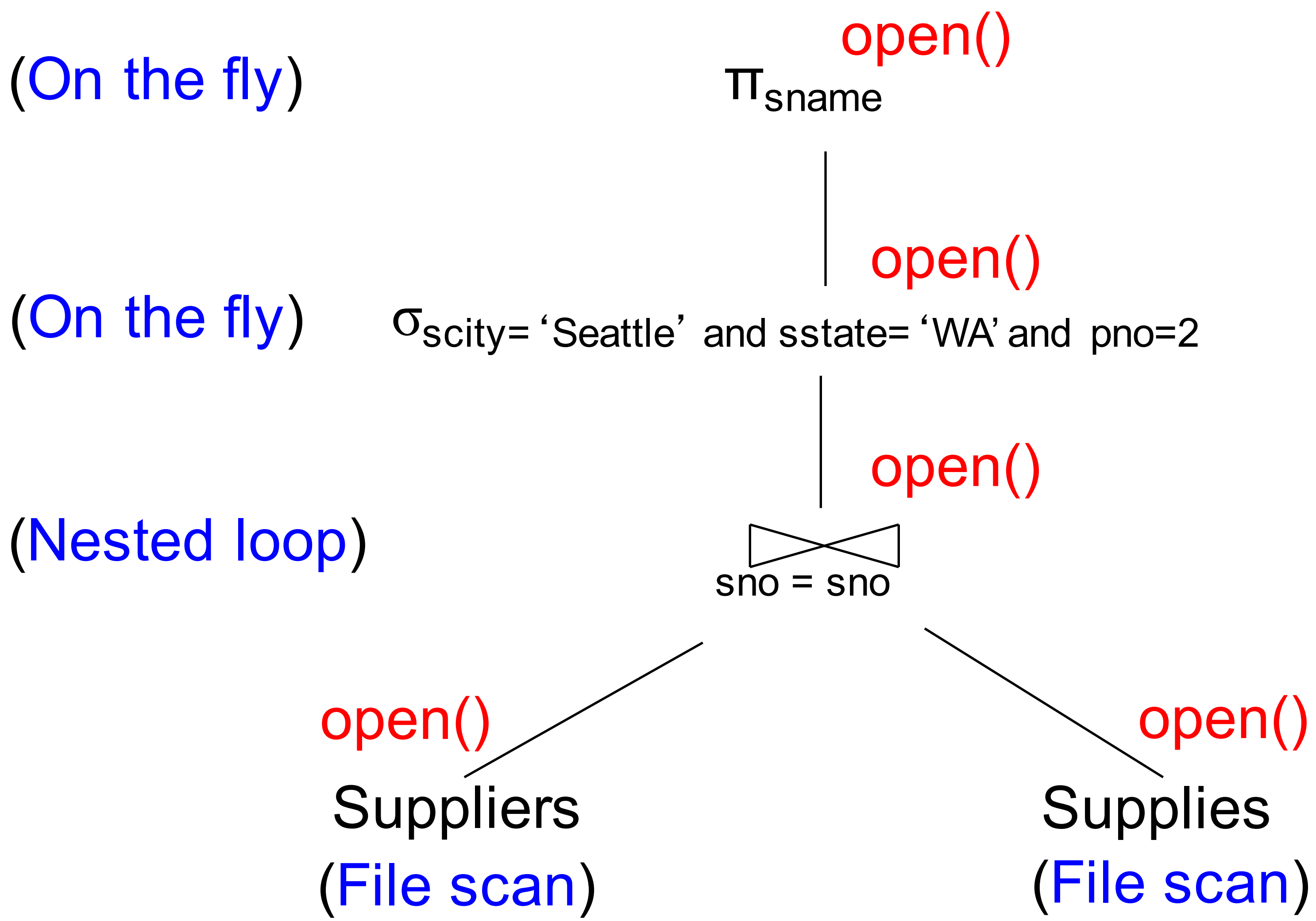
- For each SQL query… many logical plans
- For each logical plan… many physical plans
- How do find a fast physical plan?
  – Will discuss in a few lectures
  – First we need to understand how query operators are implemented

Query Execution

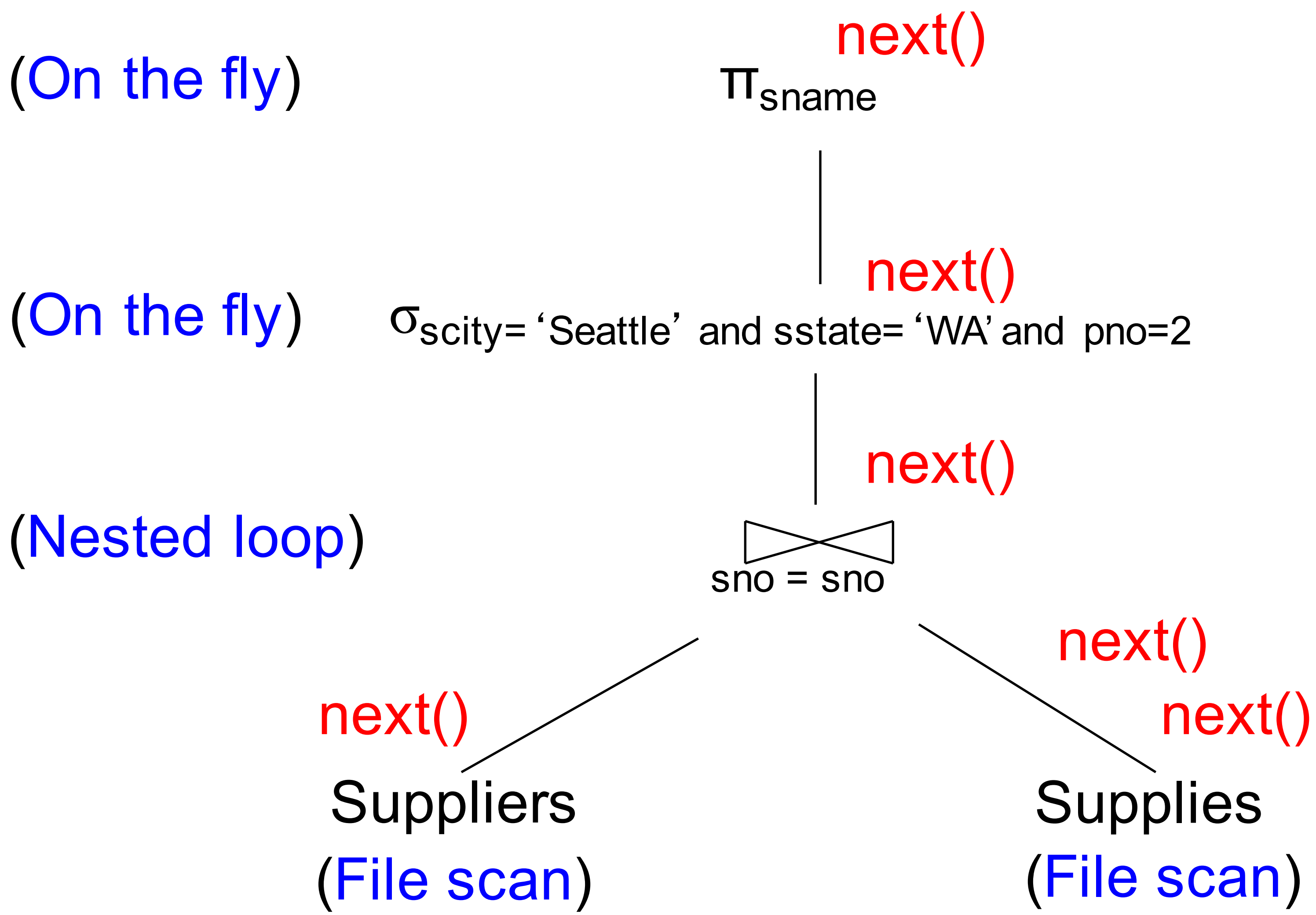
Iterator Interface for Query Operators

- **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

Query Execution Bottom Line

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

Physical Data Independence

- Applications are insulated from changes in physical storage details
- SQL and relational algebra facilitate physical data independence
  - Both languages input and output relations
  - Can choose different implementations for operators