Introduction to Database Systems CSE 414

Lectures 9-10: Relational Algebra

Announcements

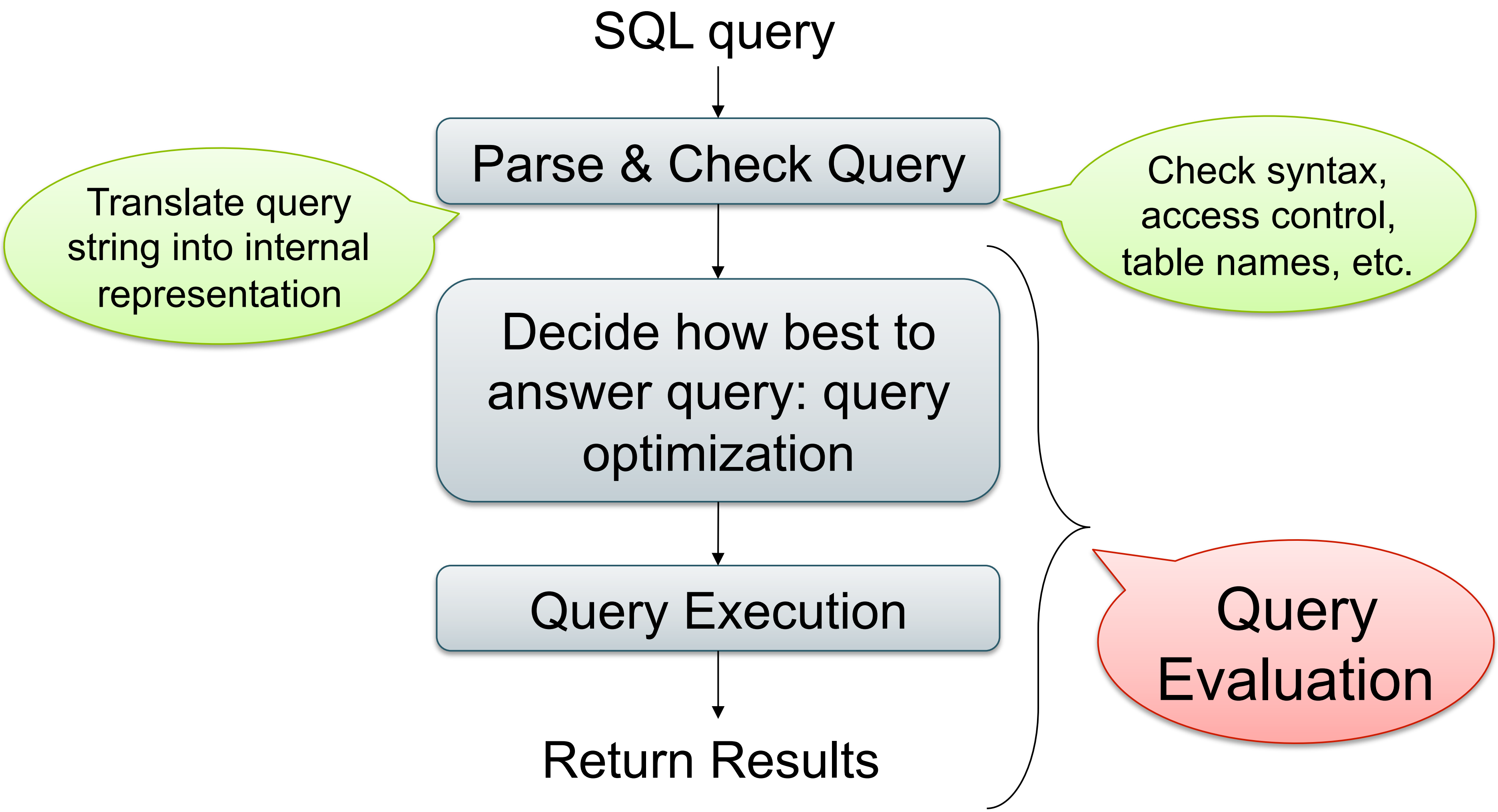
- Webquiz 3 due Monday night, 11 pm
- HW3 due Wednesday night, 11 pm
  – Log on to your Azure account by now
- Today's lecture: secs. 2.4 and 5.1

Where We Are

- Motivation for using a DBMS for managing data
- SQL, SQL, SQL
  - Declaring the schema for our data (CREATE TABLE)
  - Inserting data one row at a time or in bulk (INSERT/.import)
  - Modifying the schema and updating the data (ALTER/UPDATE)
  - Querying the data (SELECT)
  - Tuning queries (CREATE INDEX)

- Next step: More knowledge of how DBMSs work
  - Client-server architecture
  - Relational algebra and query execution

Query Evaluation Steps



The WHAT and the HOW

- SQL = WHAT we want to get form the data
- Relational Algebra = HOW to get the data we want
- The passage from WHAT to HOW is called query optimization

Overview: SQL = WHAT

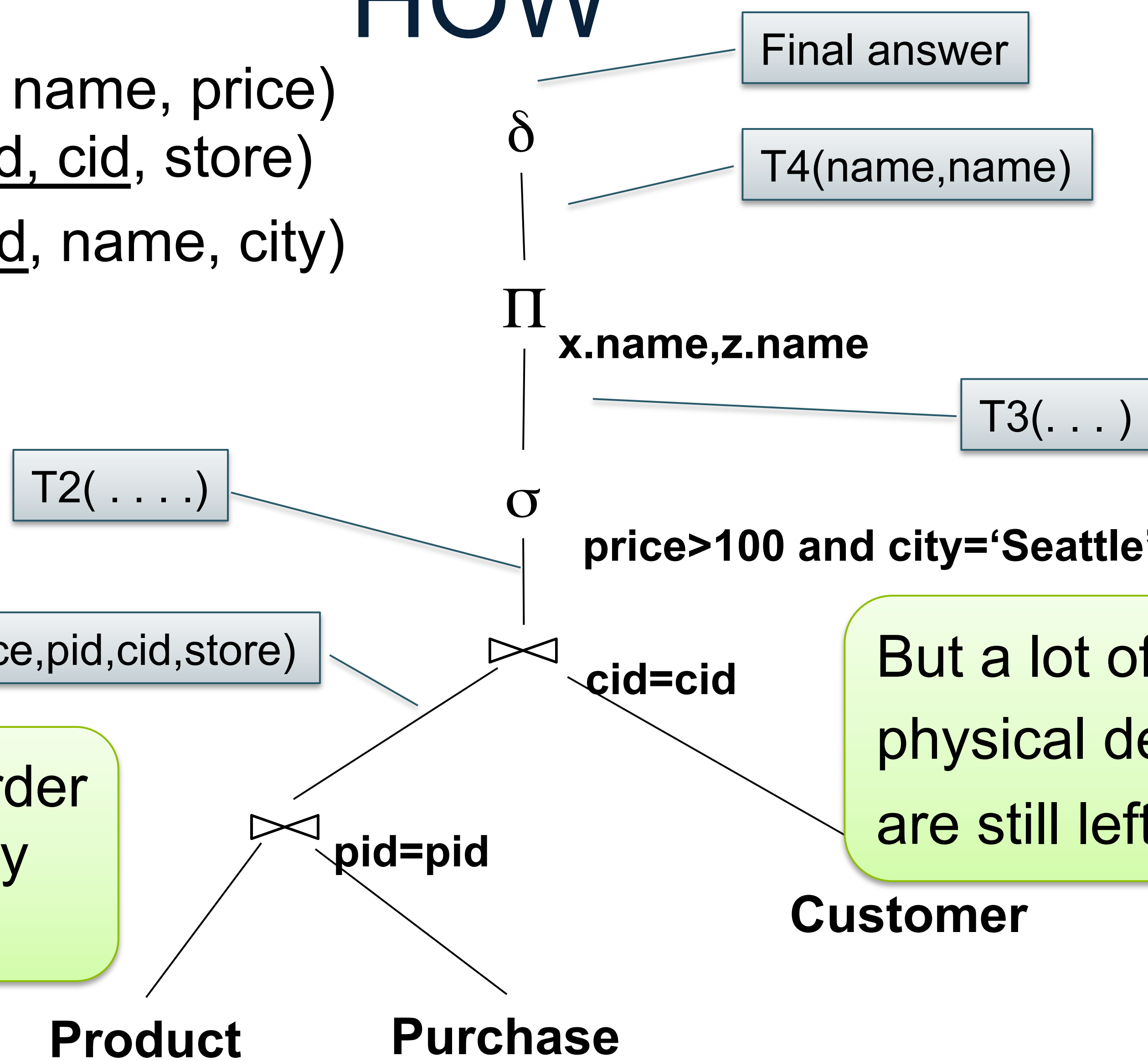
Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

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

It's clear WHAT we want, unclear HOW to get it

Overview: Relational Algebra = HOW

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



Execution order is now clearly specified

But a lot of physical details are still left open!

Relational Algebra

Sets v.s. Bags

- Sets: {a,b,c}, {a,d,e,f}, { }, . . .
- Bags: {a, a, b, c}, {b, b, b, b, b}, . . .

Relational Algebra has two semantics:

- Set semantics = standard Relational Algebra
- Bag semantics = extended Relational Algebra

DB systems implement bag semantics (Why?)

Relational Algebra Operators

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

RA

Extended RA

Union and Difference

$R1 \cup R2$

$R1 - R2$

What do they mean over bags ?

What about Intersection ?

- Derived operator using minus

$$\boxed{R1 \cap R2 = R1 - (R1 - R2)}$$

- Derived using join (will explain later)

$$\boxed{R1 \cap R2 = R1 \bowtie R2}$$

Selection

- Returns all tuples which satisfy a condition

$$\boxed{\sigma_c(R)}$$

- Examples
  - $\sigma_{\text{Salary > 40000}}$ (Employee)
  - $\sigma_{\text{name = "Smith"}}$ (Employee)
- The condition c can be =, <, ≤, >, ≥, <>

Employee

SSN	Name	Salary
1234545	John	20000
5423341	Smith	60000
4352342	Fred	50000

$\sigma_{\text{Salary > 40000}}$ (Employee)

SSN	Name	Salary
5423341	Smith	60000
4352342	Fred	50000

Projection

- Eliminates columns

$$\boxed{\Pi_{A1,\ldots,An}(R)}$$

- Example: project social-security number and names:
  - $\Pi_{\text{SSN, Name}}$ (Employee)
  - Answer(SSN, Name)

Different semantics over sets or bags! Why?

Employee

SSN	Name	Salary
1234545	John	20000
5423341	John	60000
4352342	John	20000

$\Pi_{\text{Name,Salary}}$ (Employee)

Name	Salary
John	20000
John	60000
John	20000

Bag semantics

Name	Salary
John	20000
John	60000

Set semantics

Which is more efficient?

Composing RA Operators

Patient

no	name	zip	disease
1	p1	98125	flu
2	p2	98125	heart
3	p3	98120	lung
4	p4	98120	heart

$\pi_{zip,disease}$(Patient)

zip	disease
98125	flu
98125	heart
98120	lung
98120	heart

$\sigma_{disease=\text{'heart'}}$(Patient)

no	name	zip	disease
2	p2	98125	heart
4	p4	98120	heart

π_{zip} ($\sigma_{disease=\text{'heart'}}$(Patient))

zip
98120
98125

Cartesian Product

- Each tuple in R1 with each tuple in R2

$$\boxed{R1 \times R2}$$

- Rare in practice; mainly used to express joins

Cross-Product Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependent

EmpSSN	DepName
999999999	Emily
777777777	Joe

Employee $\times$ Dependent

Name	SSN	EmpSSN	DepName
John	999999999	999999999	Emily
John	999999999	777777777	Joe
Tony	777777777	999999999	Emily
Tony	777777777	777777777	Joe

Renaming

- Changes the schema, not the instance

$$\boxed{\rho_{B1,\ldots,Bn}(R)}$$

- Example:
  - $\rho_{N,\,S}(\text{Employee}) \rightarrow \text{Answer}(N, S)$

Not really used by systems, but needed on paper

Natural Join

$$\boxed{R1 \bowtie R2}$$

- Meaning: $R1 \bowtie R2 = \Pi_A(\sigma(R1 \times R2))$
- Where:
  - Selection σ checks equality of all common attributes
  - Projection eliminates duplicate common attributes

Natural Join Example

R

A	B
X	Y
X	Z
Y	Z
Z	V

S

B	C
Z	U
V	W
Z	V

R ⋈ S =

$\Pi_{ABC}(\sigma_{R.B=S.B}(R \times S))$

A	B	C
X	Z	U
X	Z	V
Y	Z	U
Y	Z	V
Z	V	W

Natural Join Example 2

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

$P \bowtie V$

age	zip	disease	name
54	98125	heart	p1
20	98120	flu	p2

Natural Join

- Given schemas R(A, B, C, D), S(A, C, E), what is the schema of R $\bowtie$ S ?
- Given R(A, B, C), S(D, E), what is R $\bowtie$ S ?
- Given R(A, B), S(A, B), what is R $\bowtie$ S ?

Theta Join

- A join that involves a predicate

$$\boxed{R1 \bowtie_\theta R2 \;=\; \sigma_\theta\,(R1 \times R2)}$$

- Here θ can be any condition
- For our voters/disease example:

$$P \bowtie_{P.zip = V.zip \text{ and } P.age < V.age + 5 \text{ and } P.age > V.age - 5} V$$

Equijoin

- A theta join where θ is an equality

$$\boxed{R1 \bowtie_{A=B} R2 \quad = \quad \sigma_{A=B} (R1 \times R2)}$$

- This is by far the most used variant of join in practice

Equijoin Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

$P \bowtie_{P.age=V.age} V$

age	P.zip	disease	name	V.zip
54	98125	heart	p1	98125
20	98120	flu	p2	98120

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**: $R \bowtie S = \pi_A(\sigma_\theta(R \times S))$
  - Equijoin
  - Equality on **all** fields with same name in R and in S

More Joins

- **Outer join**
  – Include tuples with no matches in the output
  – Use NULL values for missing attributes
- 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

$P ⟕ J$

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

Some Examples

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

Q2: Name of supplier of parts with size greater than 10

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

Q3: Name of supplier of red parts or parts with size greater than 10

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

From SQL to RA

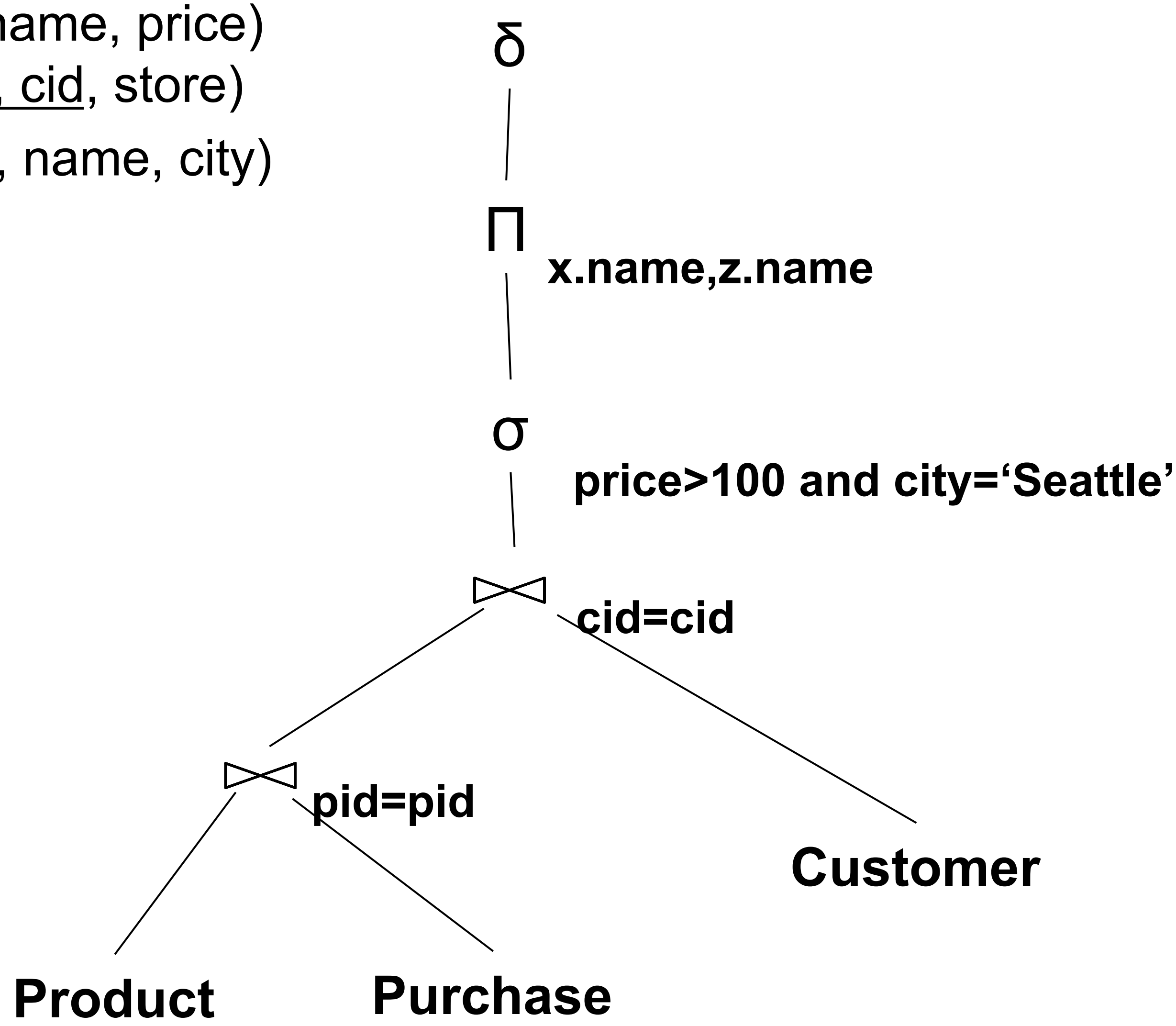
From SQL to RA

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

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

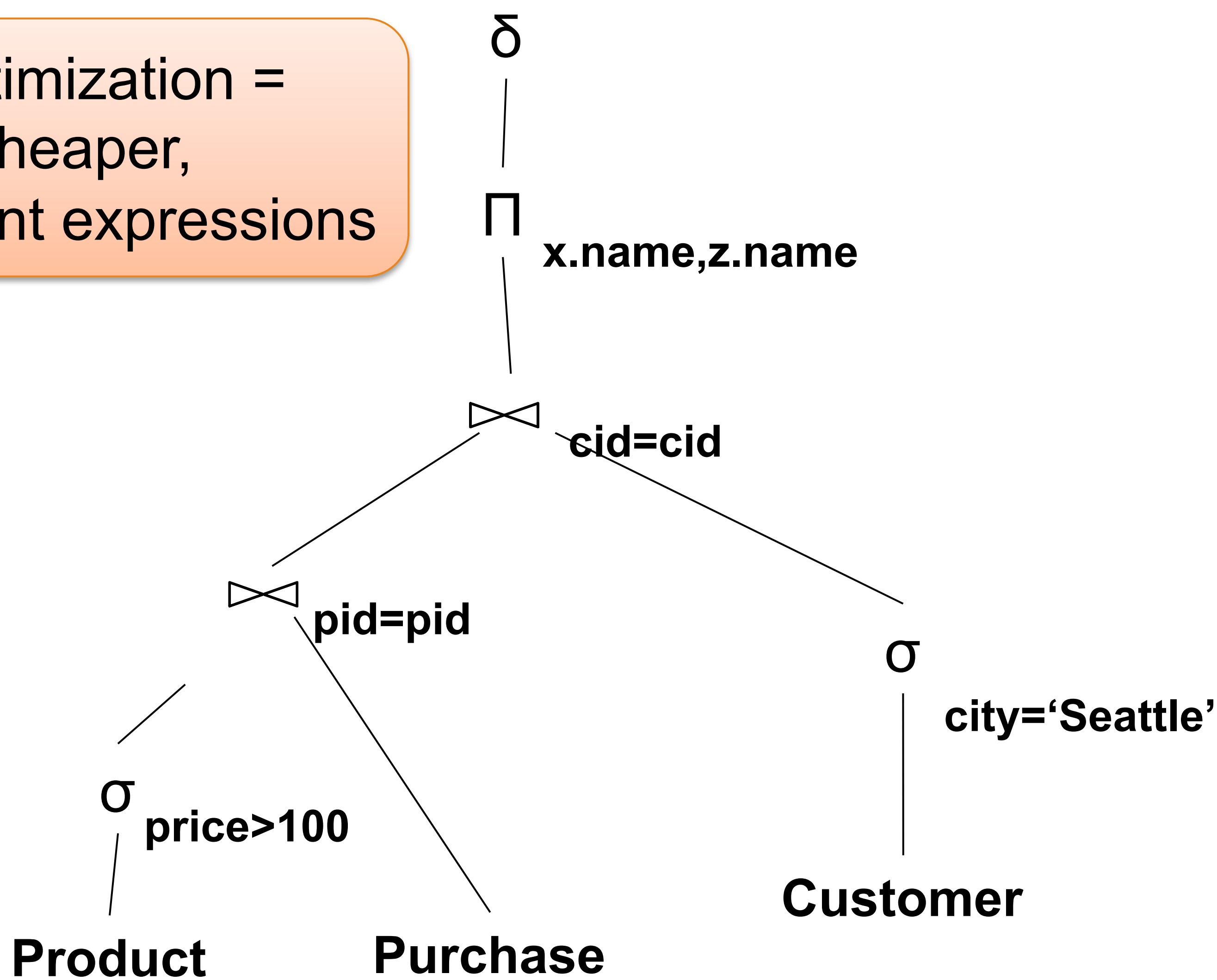
From SQL to RA

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



An Equivalent Expression

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)

$\sigma_{p > 100}$

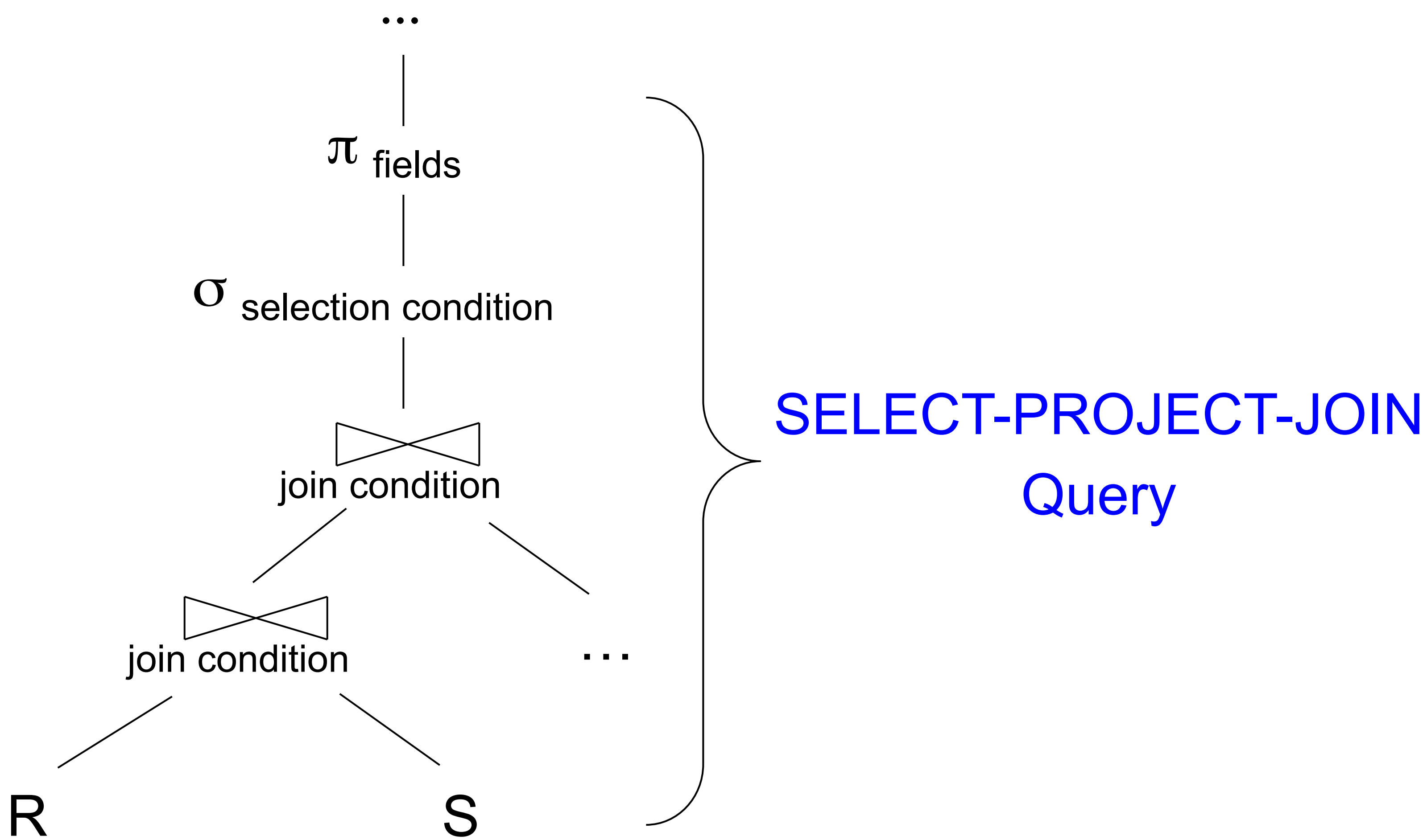
T1(city,p,c)

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

sales(product, city, price)

T1, T2, T3 = temporary tables

Typical Plan for Block (1/2)



Typical Plan For Block (2/2)

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

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

π_{fields}

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

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

… …

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

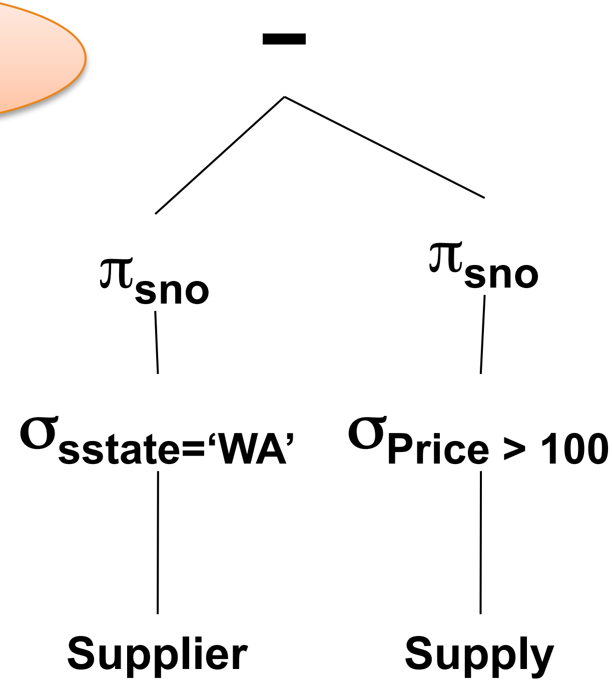
```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
    and Q.sno not in
       (SELECT P.sno
        FROM Supply P
        WHERE P.price > 100)
```

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

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

Example

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

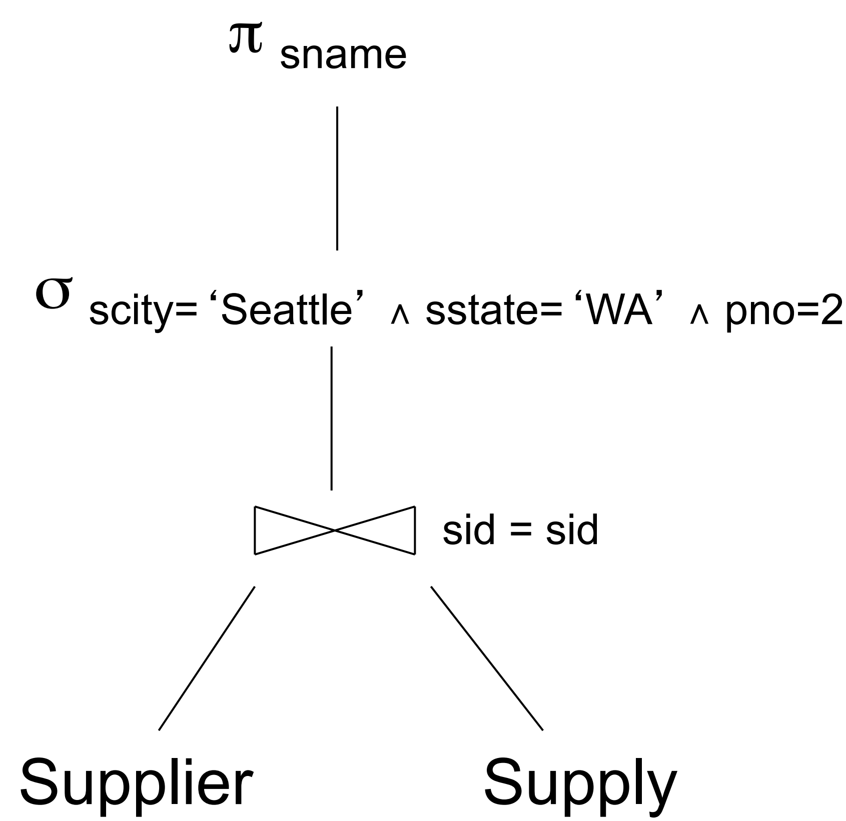
Supply(<u>sid, pno</u>, quantity)

Relational Algebra

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

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Relational Algebra



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

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Physical Query Plan 1

(On the fly) π_{sname}

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

(Block-nested loop) $\bowtie_{\text{sid = sid}}$

Supplier (File scan)

Supply (File scan)

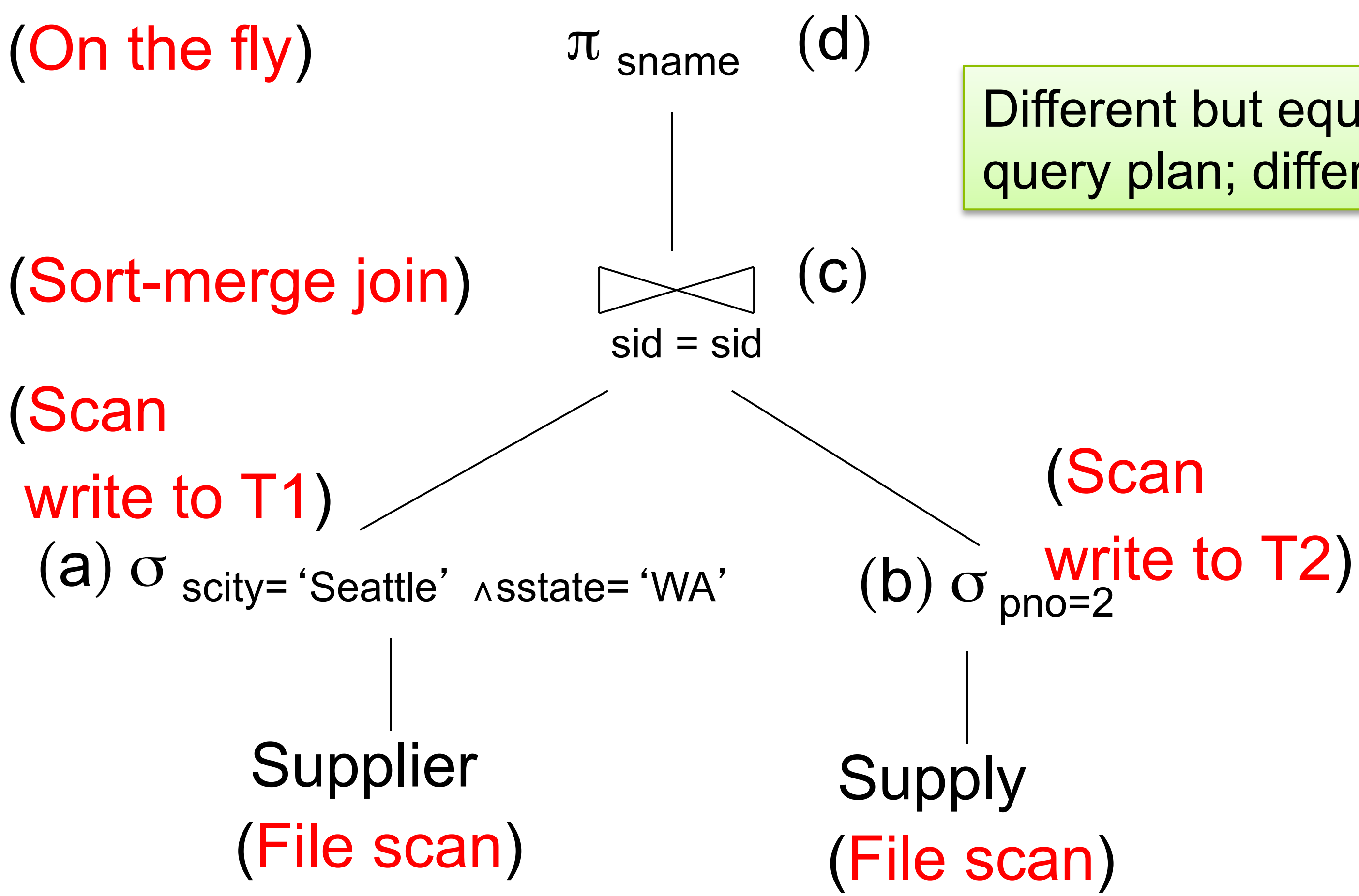
A physical query plan is a logical query plan annotated with physical implementation details

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

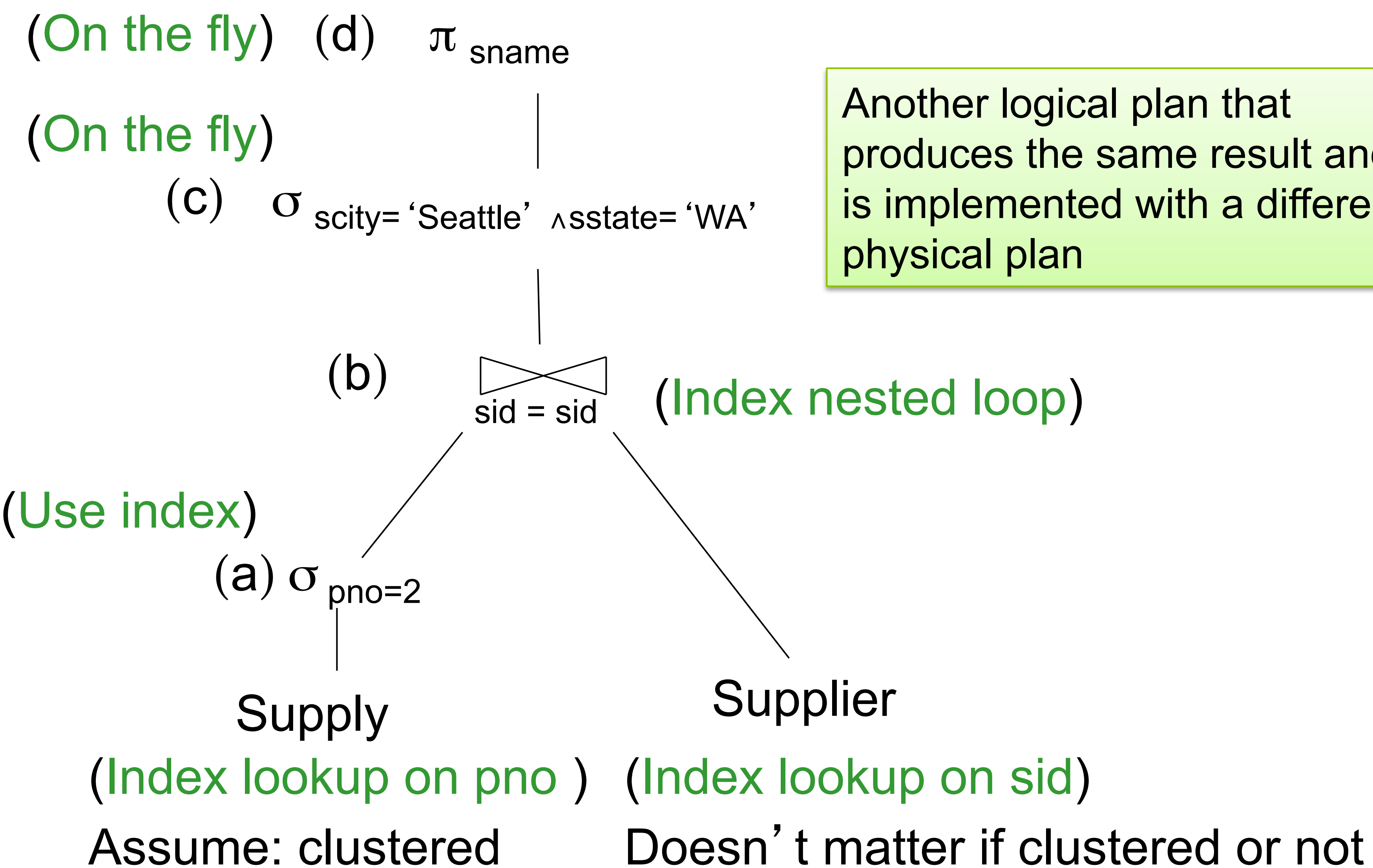
Physical Query Plan 2

Different but equivalent logical query plan; different physical plan



Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Query Plan 3



Another logical plan that produces the same result and is implemented with a different physical plan

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