

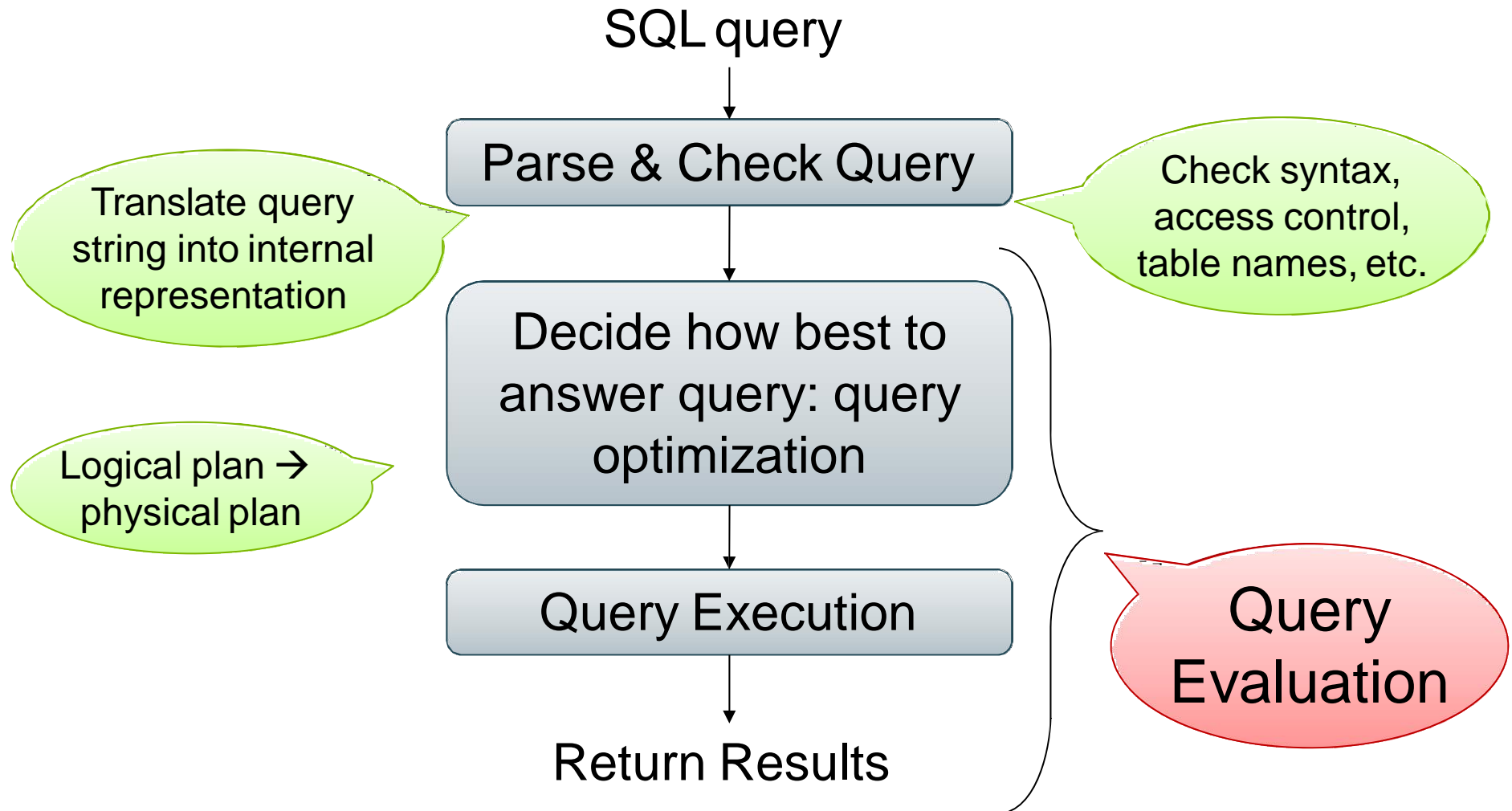
Introduction to Database Systems

Relational Algebra
(Ch. 2.4, & 5.1)

Where We Are

- Motivation for using a DBMS for managing data
- 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)
- 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 from the data
- Relational Algebra = **HOW** to get the data we want
- Move from **WHAT** to **HOW** is **query optimization**
 - SQL ~> Relational Algebra ~> Physical Plan
 - Relational Algebra = Logical Plan

Relational Algebra

Sets vs. Bags

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

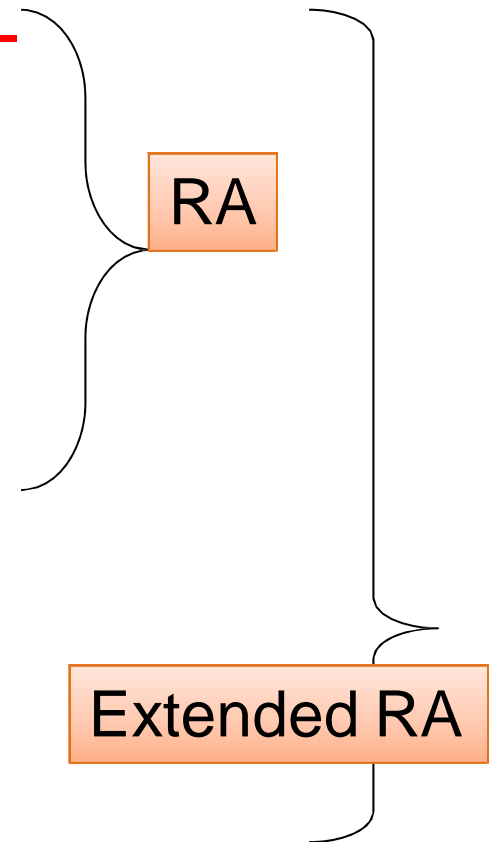
Relational Algebra has two semantics:

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

DB systems implement bag semantics (Why?)

Relational Algebra Operators

- Union \cup , intersection \cap , difference $-$
- Selection σ
- Projection π (Π)
- Cartesian product \times , join \bowtie
- Rename ρ
- Duplicate elimination δ
- Grouping and aggregation γ
- Sorting τ



Union and Difference

$$R1 \cup R2$$

$$R1 - R2$$

What do they mean over bags ?

What about Intersection ?

- Derived operator using minus

$$R1 \cap R2 = R1 - (R1 - R2)$$

- Derived using join (will explain later)

$$R1 \cap R2 = R1 \bowtie R2$$

Selection

- Returns all tuples that satisfy a condition

$$\sigma_c(R)$$

- Examples
 - $\sigma_{\text{Salary} > 40000}(\text{Employee})$
 - $\sigma_{\text{name} = \text{"Smith"}}(\text{Employee})$
- The condition c can be $=$, $<$, \leq , $>$, \geq , $<>$ combined with AND, OR, NOT

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

$$\pi_{A_1, \dots, A_n}(R)$$

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

Different semantics over sets or bags! Why?

Employee

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

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

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

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

zip	disease
98125	flu
98125	heart
98120	lung
98120	heart

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

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

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

zip	disease
98125	heart
98120	heart

Cartesian Product

- Each tuple in R1 with each tuple in R2

$$R1 \times R2$$

- 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 × 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

$$\rho_{B1, \dots, Bn} (R)$$

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

Not really used by systems, but needed on paper

Natural Join

$$R1 \bowtie R2$$

- Meaning: $R1 \bowtie R2 = \pi_A(\sigma_\theta(R1 \times R2))$
- Where:
 - Selection σ checks equality of **all common attributes** (attributes with same names)
 - Projection π 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** =

$\pi_{A,B,C}(\sigma_{R.B=S.B}(R \times S))$

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 $R \bowtie S$?
 - (A, B, C, D, E) through join on (A, C)
- Given $R(A, B, C)$, $S(D, E)$, what is $R \bowtie S$?
 - (A, B, C, D, E) through cross product
- Given $R(A, B)$, $S(A, B)$, what is $R \bowtie S$?
 - (A, B) through intersection

AnonPatient (age, zip, disease)

Voters (name, age, zip)

Theta Join

- A join that involves a predicate

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

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

$$P \bowtie_{P.zip = V.zip \text{ and } P.age \geq V.age - 1 \text{ and } P.age \leq V.age + 1} V$$

Equijoin

- A theta join where θ is an equality predicate
- 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

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

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

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:** $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
 - Join condition θ consists only of equalities
- **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
 - 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

More Examples

Supplier(sno, sname, scity, sstate)

Part(pno, pname, psize, pcolor)

Supply(sno, pno, qty, price)

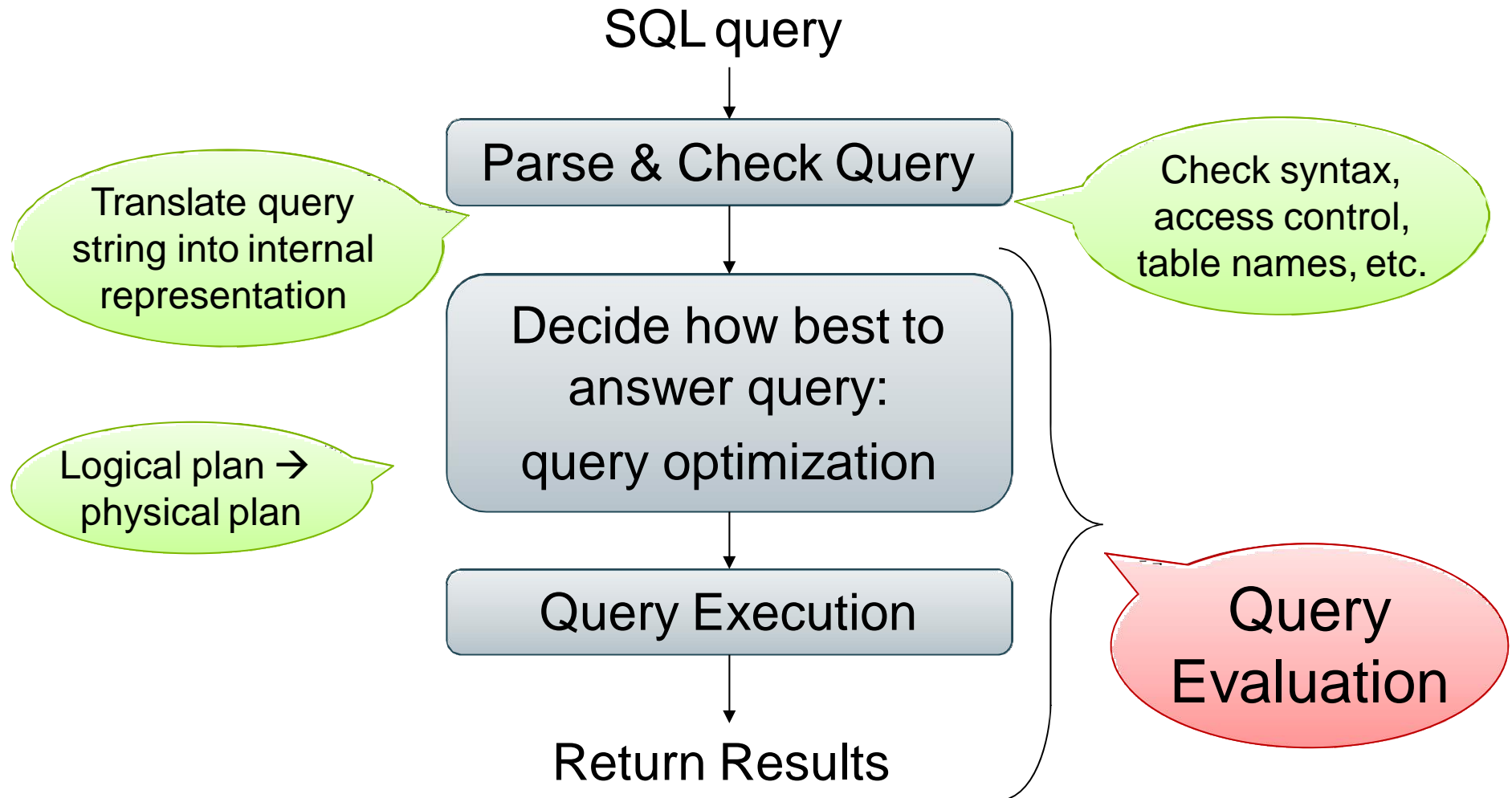
Name of supplier of parts with size greater than 10

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

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

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

Query Evaluation Steps



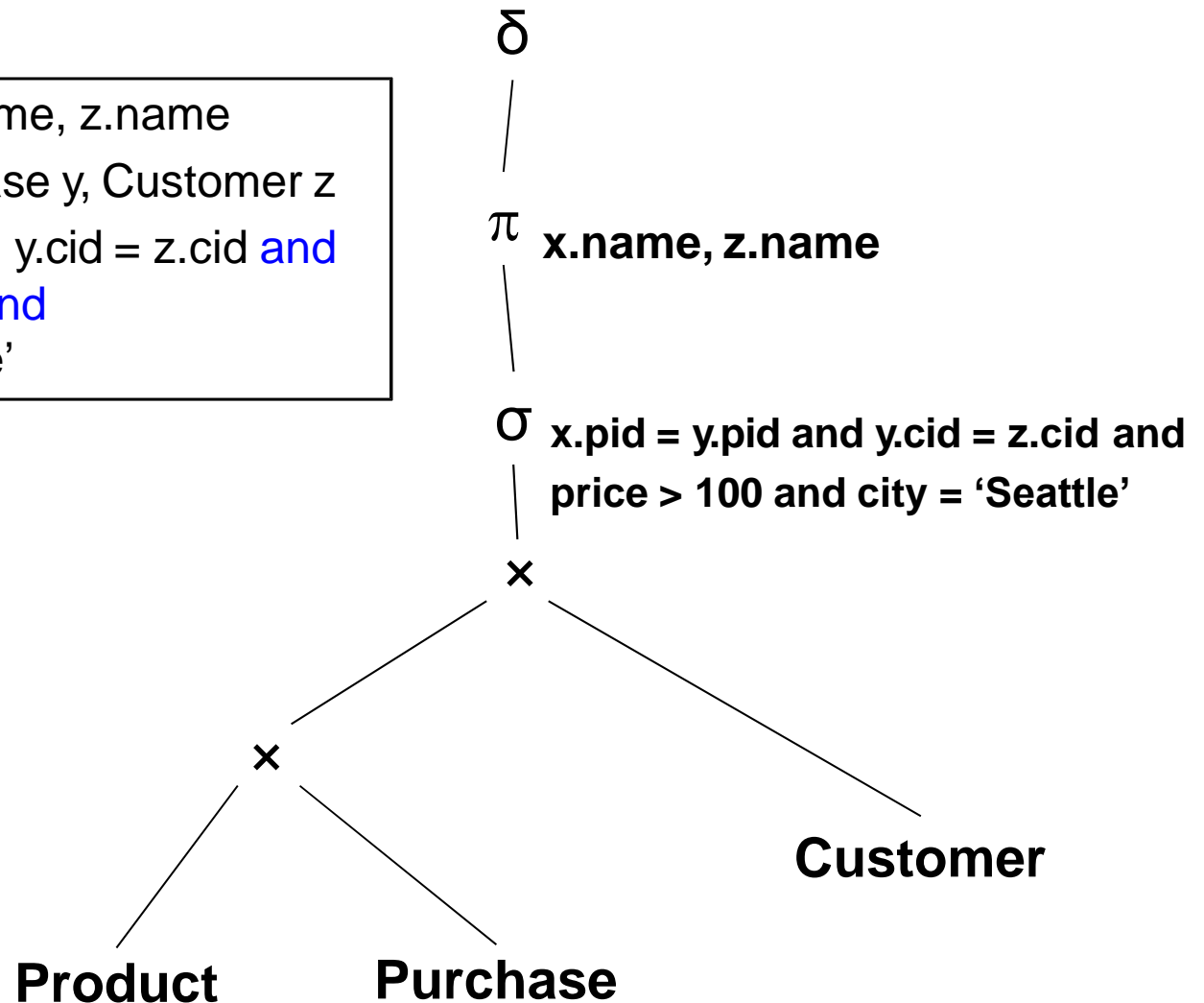
Product(pid, name, price)

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 = z.cid and
      x.price > 100 and
      z.city = 'Seattle'
```

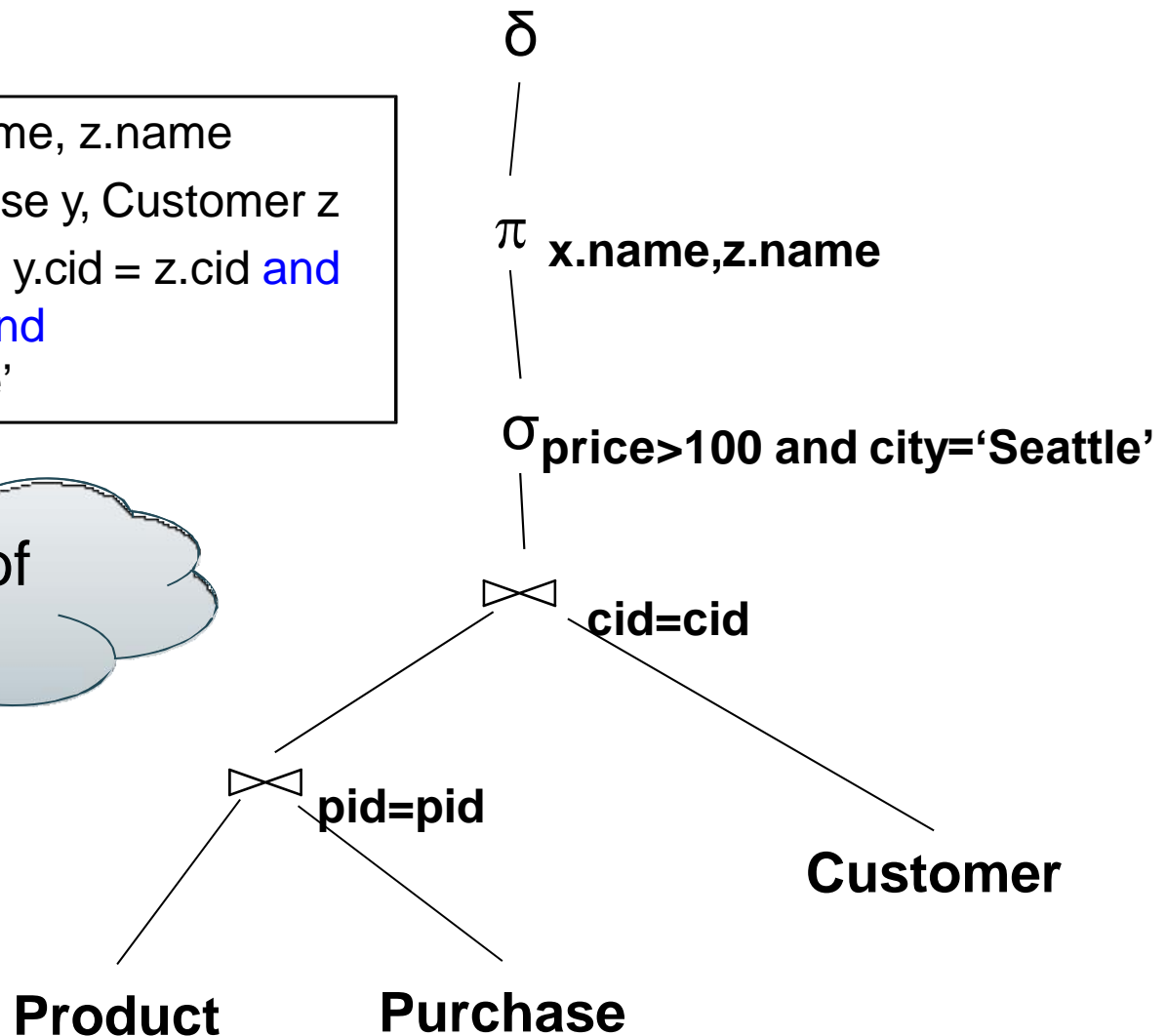


Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

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 = z.cid and
      x.price > 100 and
      z.city = 'Seattle'
```

Can you think of
another plan?



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

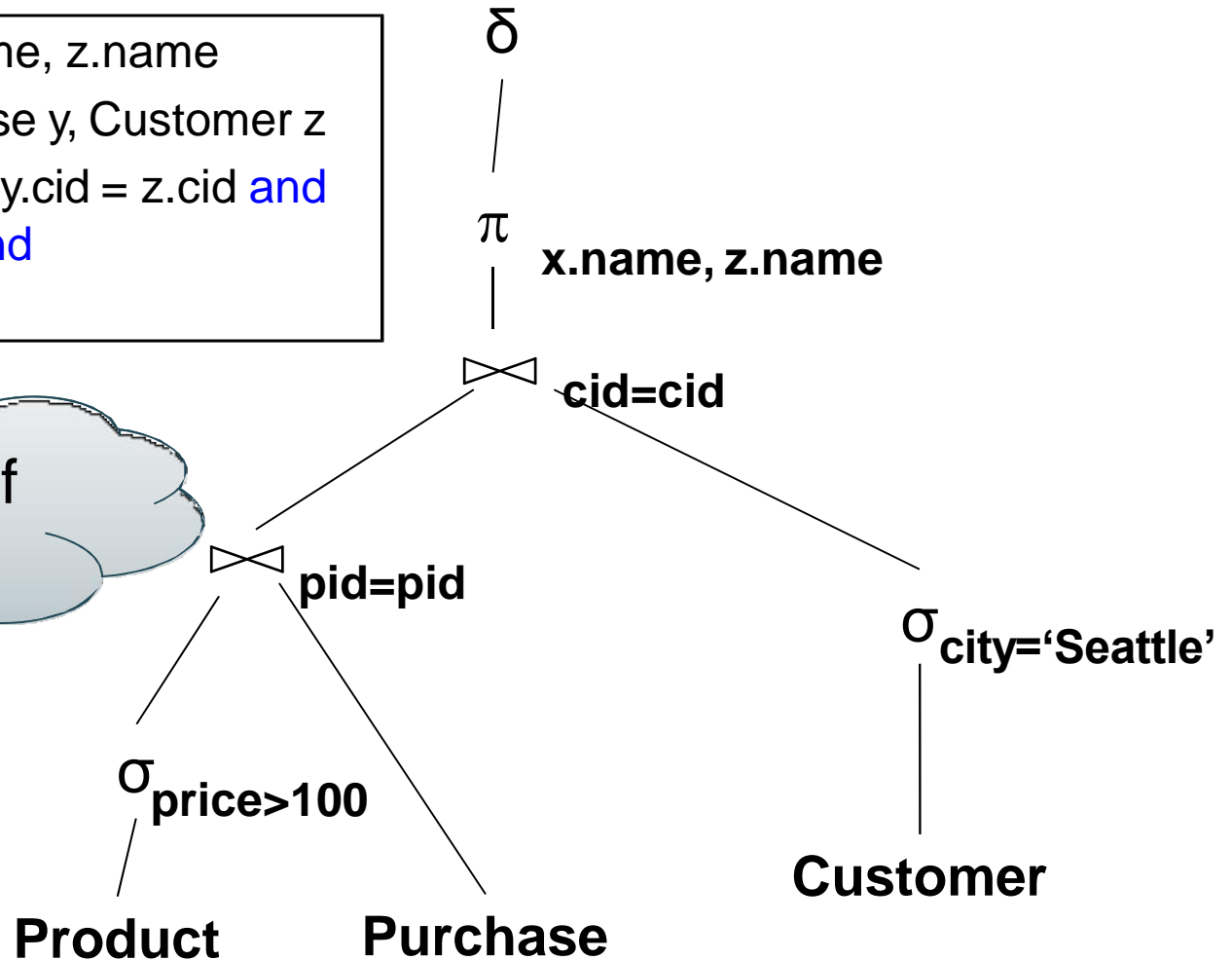
From SQL to RA

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

Can you think of
another plan?

Push selections down
the query plan!

Query optimization: find
an equivalent optimal plan



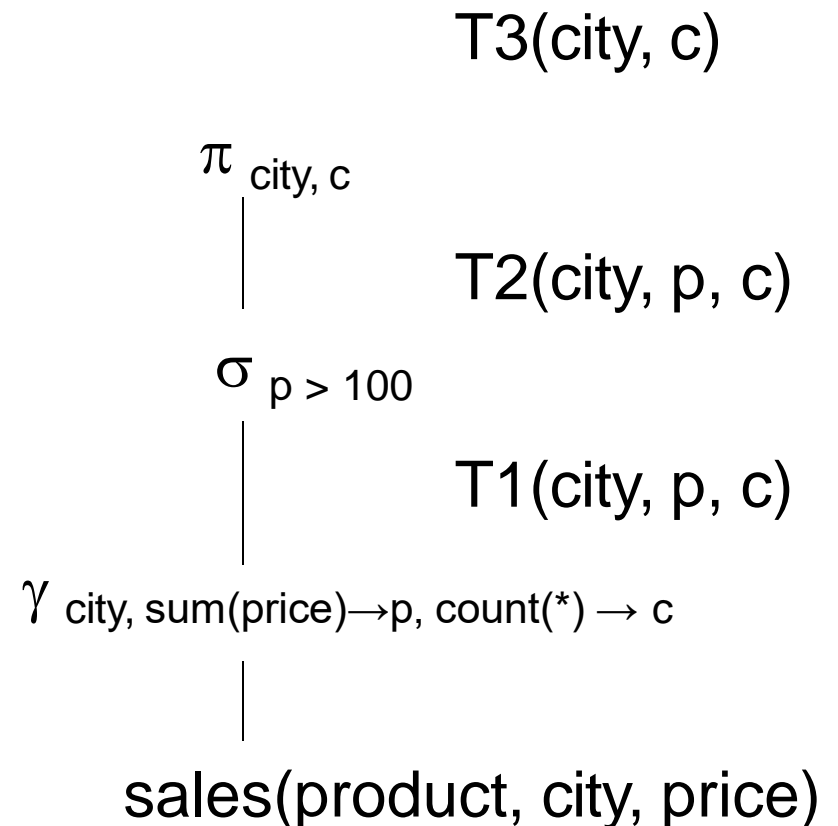
Extended RA: Operators on Bags

- Duplicate elimination δ
- Grouping & aggregation γ
- Sorting τ

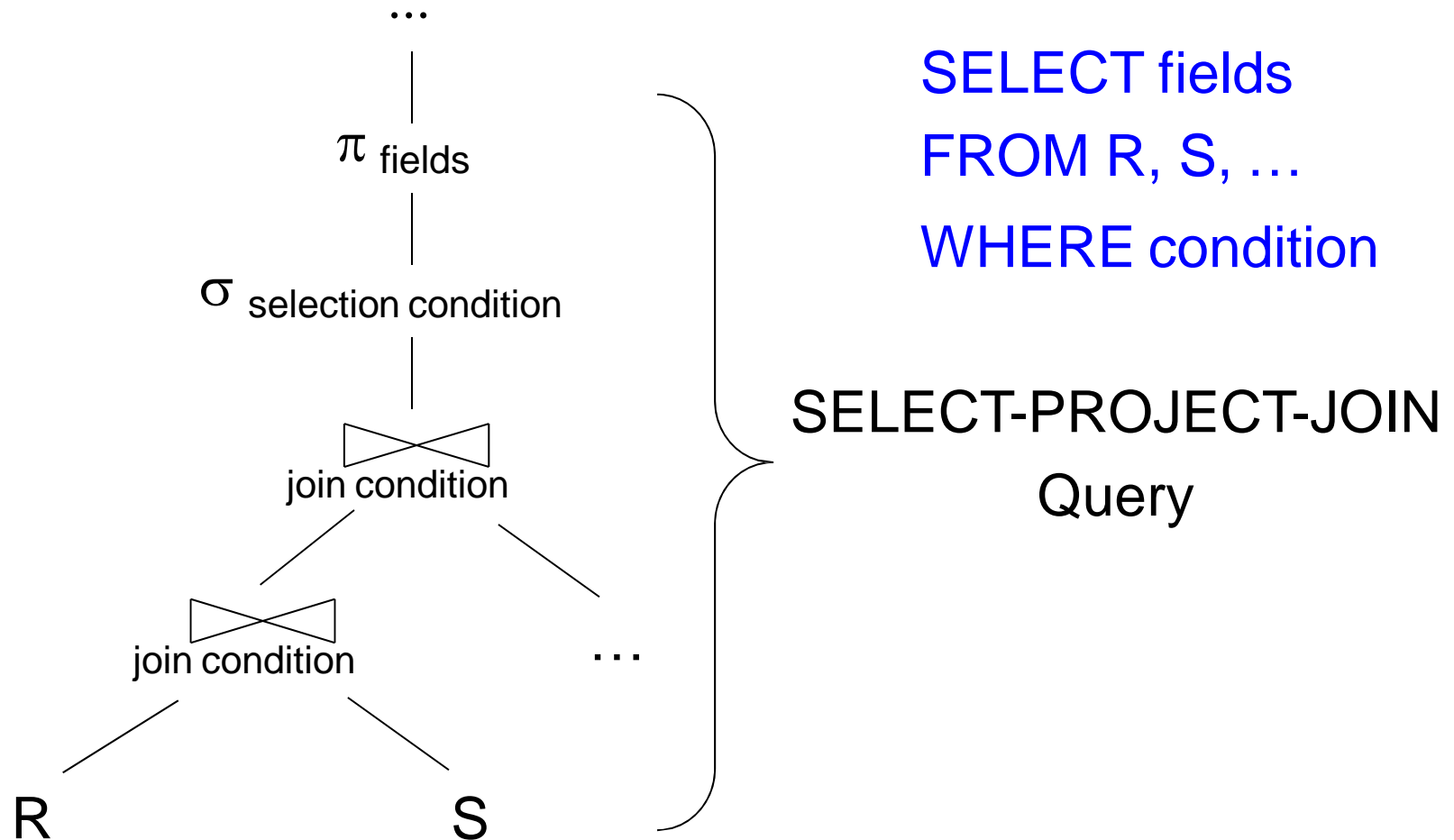
Logical Query Plan

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

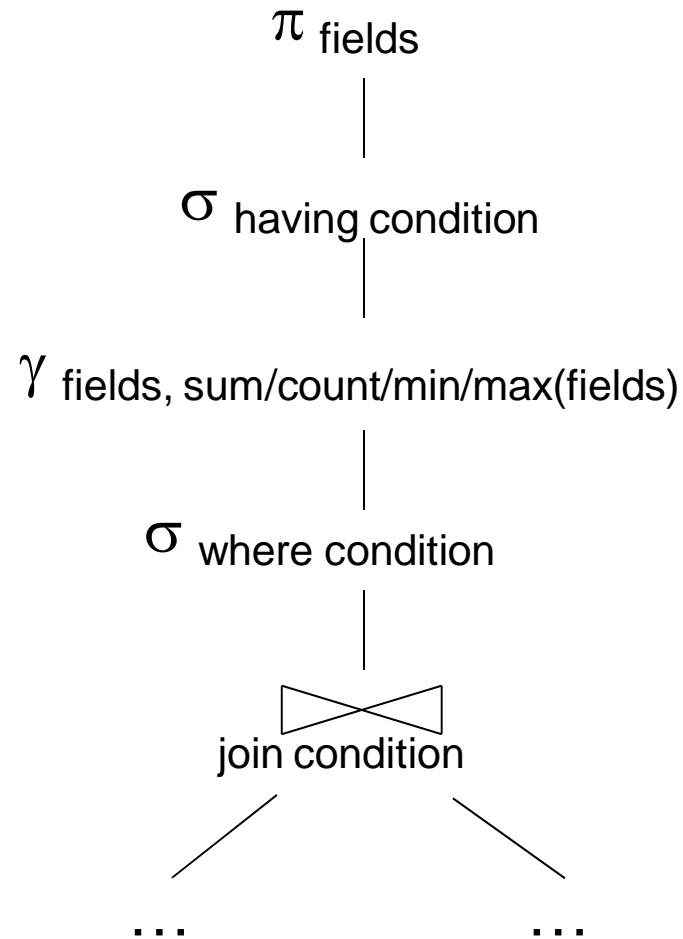
T1, T2, T3 = temporary tables



Typical Plan for Block (1/2)



Typical Plan for Block (2/2)



SELECT fields
FROM R, S, ...
WHERE condition
GROUP BY fields
HAVING condition

How about Subqueries?

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

```
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 !

How about Subqueries?

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

```
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)
```

How about Subqueries?

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

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

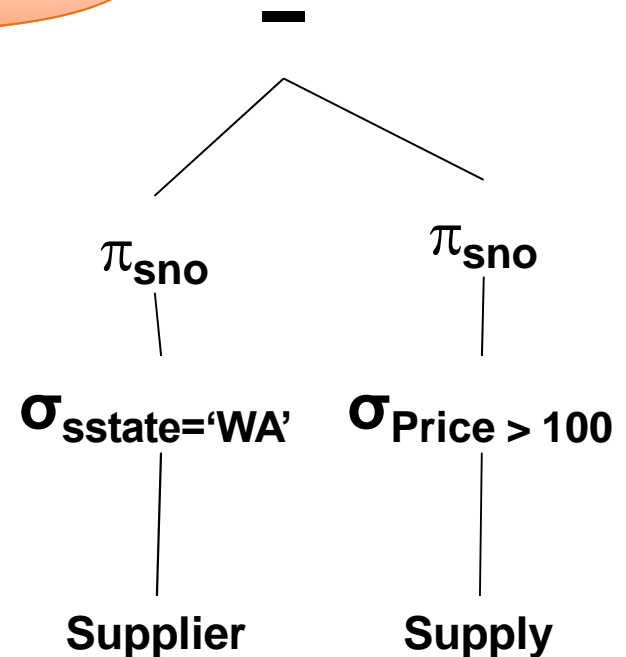
```
SELECT Q.sno  
FROM Supplier Q  
WHERE Q.sstate = 'WA'  
and Q.sno not in  
(SELECT P.sno  
FROM Supply P  
WHERE P.price > 100)
```


How about Subqueries?

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

```
(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

Physical Operators

Each of the logical operators may have one or more implementations = physical operators

Will discuss several basic physical operators, with a focus on join

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

Main Memory Algorithms

Logical operator:

Product(pid, name, price) ⋈ Purchase(pid, cid, store) pid=pid

Propose three physical operators for the join,
assuming the tables are in main memory:

1. Nested Loop Join $O(??)$
2. Merge join $O(??)$
3. Hash join $O(??)$

(note that pid is a key)

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

Main Memory Algorithms

Logical operator:

Product(pid, name, price) $\bowtie_{pid=pid}$ Purchase(pid, cid, store)

Propose three physical operators for the join, assuming the tables are in main memory:

- | | | |
|---------------------|----------|--|
| 1. Nested Loop Join | $O(n^2)$ |  two nested loops |
| 2. Merge join | $O(??)$ | |
| 3. Hash join | $O(??)$ | |

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

Main Memory Algorithms

Logical operator:

Product(pid, name, price) $\bowtie_{pid=pid}$ Purchase(pid, cid, store)

Propose three physical operators for the join, assuming the tables are in main memory:

1. Nested Loop Join $O(n^2)$
2. Merge join $O(n \log n)$
3. Hash join $O(??)$

sort both – $O(n \log n)$
merge – $O(n)$

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

Main Memory Algorithms

Logical operator:

Product(pid, name, price) ⋈ Purchase(pid, cid, store) pid=pid

Propose three physical operators for the join,
assuming the tables are in main memory:

1. Nested Loop Join $O(n^2)$
2. Merge join $O(n \log n)$
3. Hash join $O(n) \dots O(n^2)$

add n to hash – $O(n)$?
lookup n in hash – $O(n)$?

BRIEF Review of Hash Tables

Separate chaining:

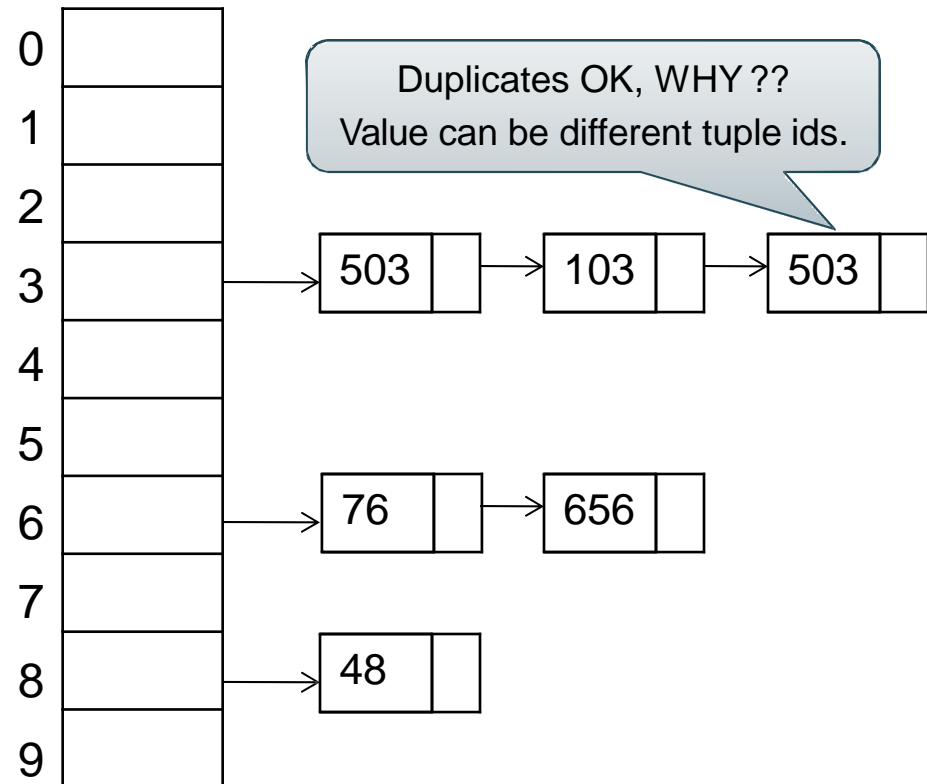
A (naïve) hash function:

$$h(x) = x \bmod 10$$

Operations:

find(103) = ??

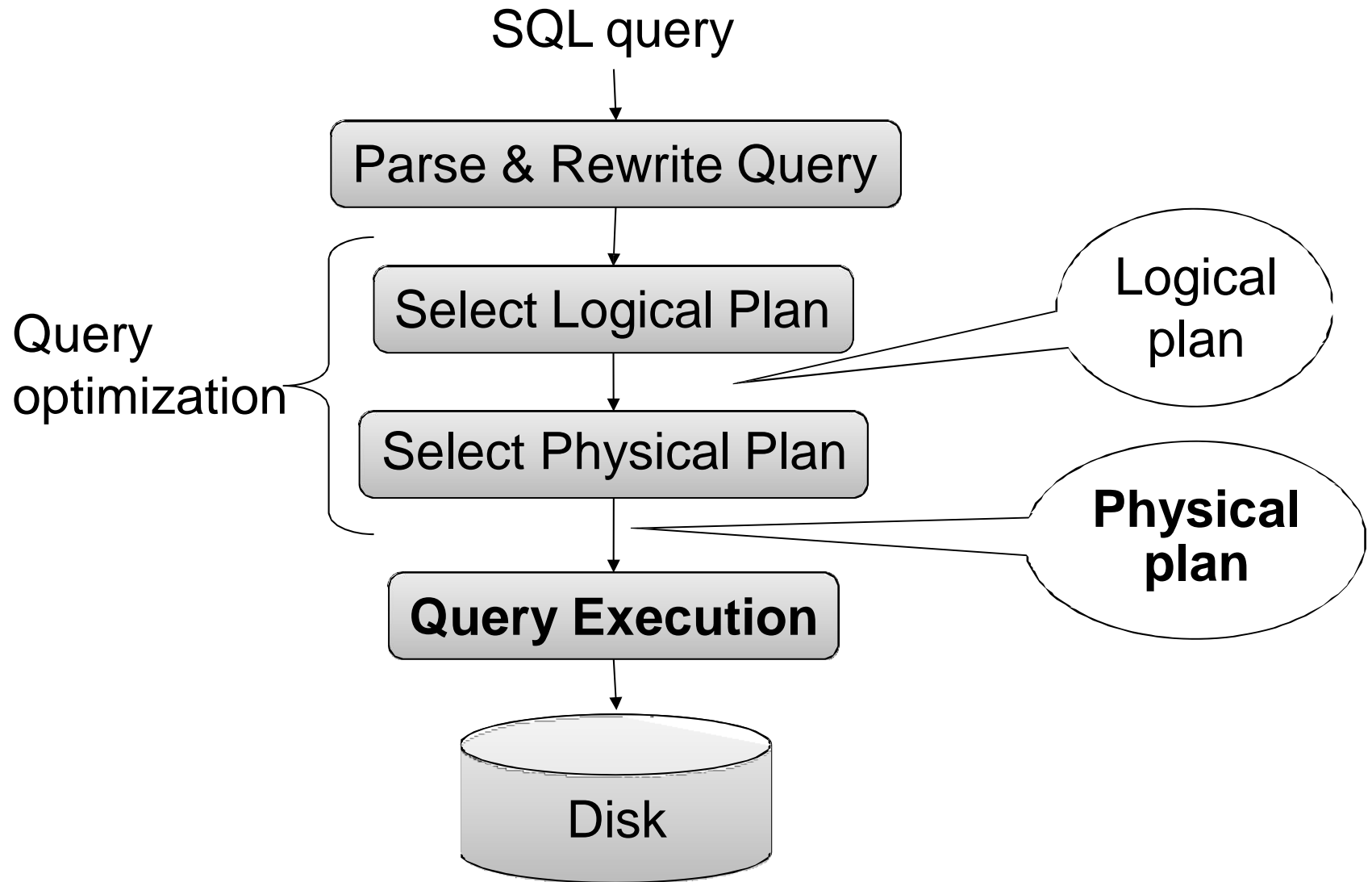
insert(488) = ??



BRIEF Review of Hash Tables

- $\text{insert}(k, v)$ = inserts a key k with value v
- Many values for one key
 - Hence, duplicate k 's are OK
- $\text{find}(k)$ = returns the **list** of all values v associated to the key k

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'
```

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'
```

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

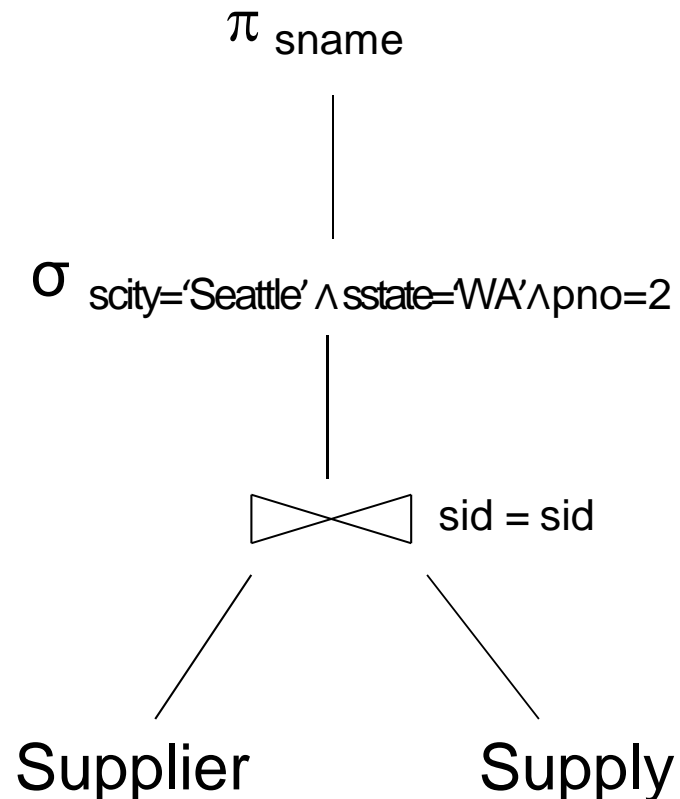
Supplier(sid, sname, scity, sstate)

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

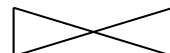
(On the fly)

π_{sname}

(On the fly)

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

(Nested loop)


sid = sid

Supplier
(File scan)

Supply
(File scan)

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

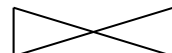
(On the fly)

π_{sname}

(On the fly)

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

(Hash join)


sid = sid

Supplier
(File scan)

Supply
(File scan)

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

(On the fly)

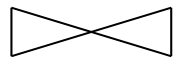
(Sort-merge join)

(Scan & write to T1)

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

Supplier
(File scan)

π_{sname}



sid = sid

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

Supply
(File scan)

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'
```

(Scan & write to T2)

Query Optimization Problem

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