DATA516/CSED516
Scalable Data Systems and Algorithms

Lecture 2 Query Execution and Optimization

Announcements

- Project teams due on Wednesday (October 18)
  - Fill out even if you are doing it solo

- Proposals due next Friday (October 27)

- HW1 is due Monday (October 23)
  - SQL Queries for Q1, Q4

Administrivia

- Paper Reading #1: Graded, released tonight
  - Address all papers, ~1 page, Avoid Copy-Paste

- Gitlab access should be resolved

- Gitlab README
  1. git remote add upstream git@gitlab.cs.washington.edu:jackkhuu/csed516-2023au.git
  2. git pull upstream main

Outline for Today

- Discuss *Goes Around* paper

- Discuss query optimization
  - Major paper to read for next time
  - We continue query optimization next time
    - (Maybe)

Discussion of the paper

- M. Stonebraker and J. Hellerstein. What Goes Around Comes Around. In "Readings in Database Systems" (aka the Red Book). 4th ed.

Data Model

- Enables a user to define the data using high-level constructs without worrying about many low-level details of how data will be stored on disk

- Examples:
  - Relational data model
  - Semi-structured data model
  - Graph data model
  - Key-value pairs data model

Data Independence

What is it?

Data Independence

What is it?

- **Physical data independence**: Applications are insulated from changes in **physical storage details**

- **Logical data independence**: Applications are insulated from changes to **logical structure of the data**

Paper Discussion

- Early data models: IMS, CODASYL

- Relational data model

- Semistructured data model

Early Proposal 1: IMS*

- What is it?

* IBM Information Management System

Early Proposal 1: IMS*

- What is it?

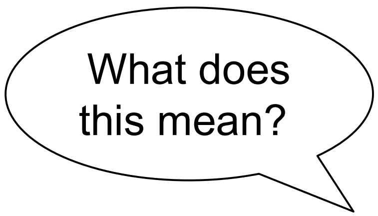
- **Hierarchical data model**

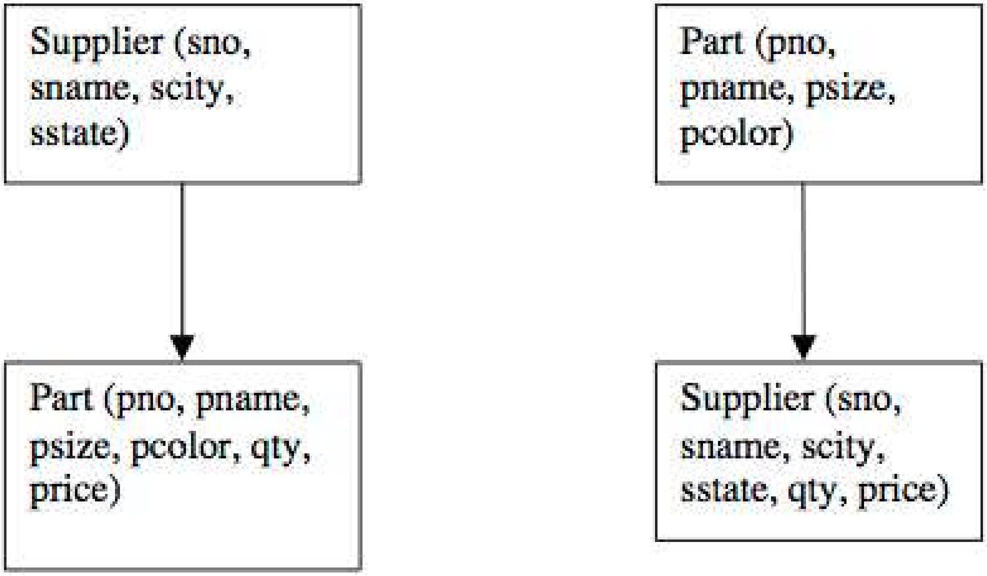
- **Record**
  - Record type, record instance
  - Each instance has a **key**
  - Arranged in a **tree**

* IBM Information Management System

IMS Example

Figure 2 from “What goes around comes around”

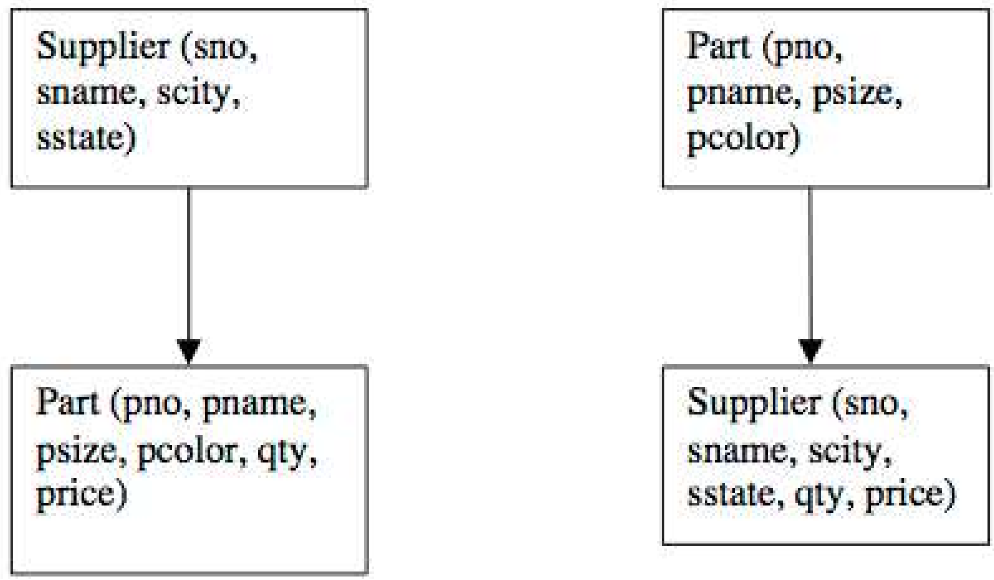




IMS Example

Figure 2 from “What goes around comes around”

What does this mean?

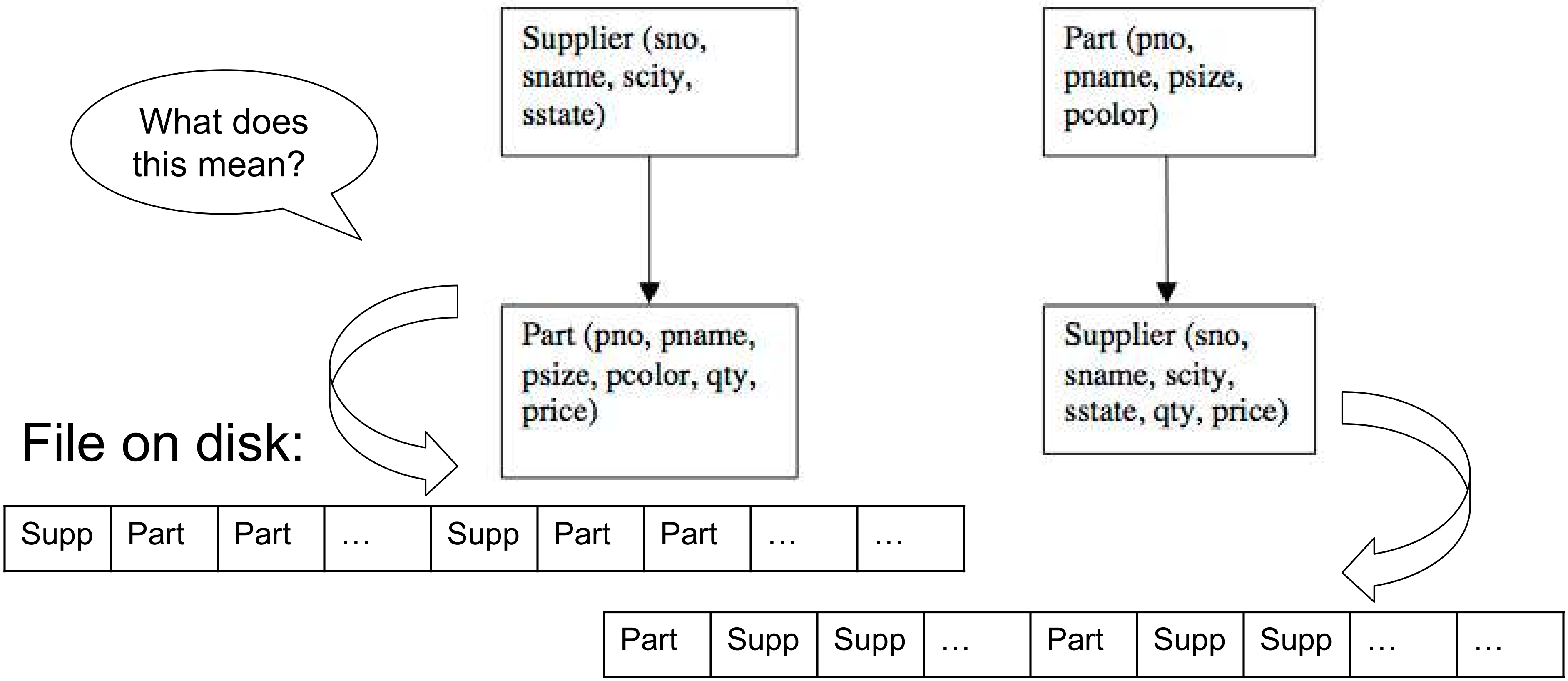


File on disk:

Supp	Part	Part	…	Supp	Part	Part	…	…

IMS Example

Figure 2 from “What goes around comes around”



IMS Limitations

IMS Limitations

- Tree-structured: redundant; existence depends on parent
- Record-at-a-time interface
- Very limited physical independence
- Some logical independence but limited

Data Manipulation Language: DL/1

How does a programmer retrieve data in IMS?

Data Manipulation Language: DL/1

How does a programmer retrieve data in IMS?

- Each record has a hierarchical sequence key (HSK)
- get_next; get_next_within_parent
- Programmers need to worry about optimization

DL/1 is a record-at-a-time language

Data storage

How is data physically stored in IMS?

Data storage

How is data physically stored in IMS?

- Root records
  - Stored sequentially (sorted on key), or
  - Indexed in a B-tree using the key of the record, or
  - Hashed using the key of the record
- Dependent records: various forms of pointers
- Selected organizations restrict DL/1 commands

Early Proposal 2: CODASYL

What is it?

Early Proposal 2: CODASYL

What is it?

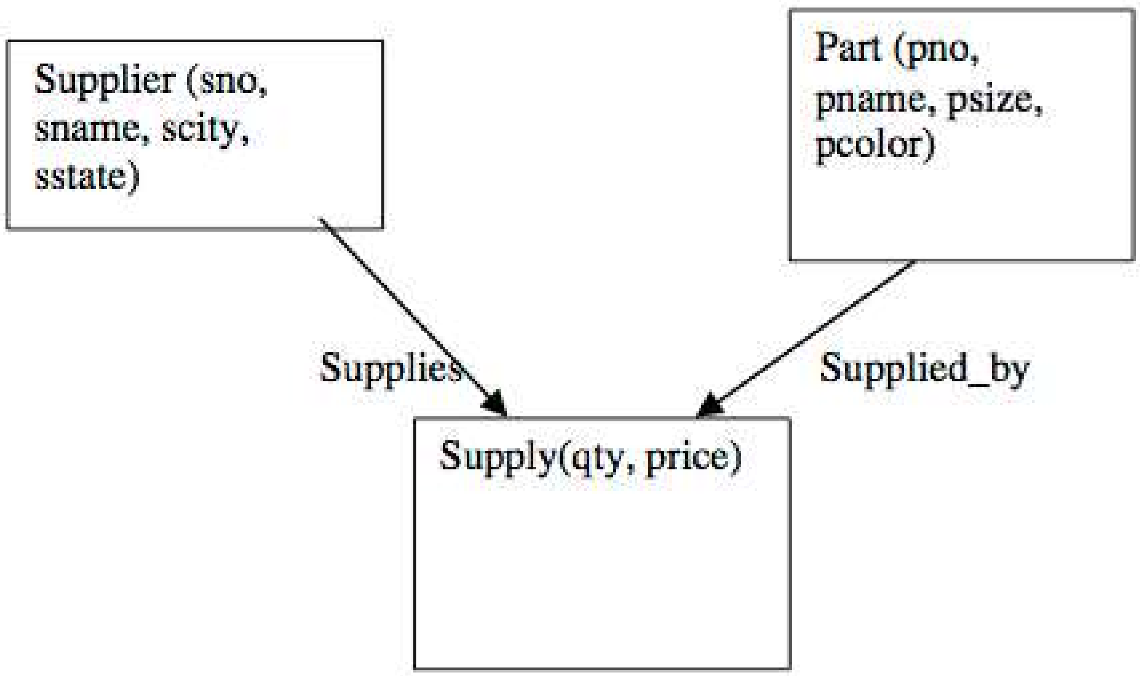
- **Networked data model**

- Organized into **network**
- Multiple parents; arcs = “sets”

- **Record-at-a-time** data manipulation language

CODASYL Example

- Figure 5 from “What goes around comes around”



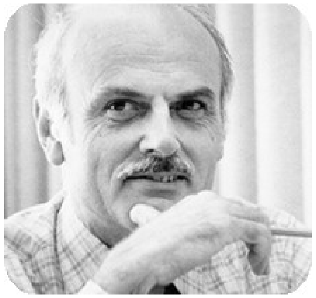
Paper Discussion

- Early data models: IMS, CODASYL
- Relational data model
- Semistructured data model

Relational Model Overview

Ted Codd 1970

- What was the motivation? What is the model?



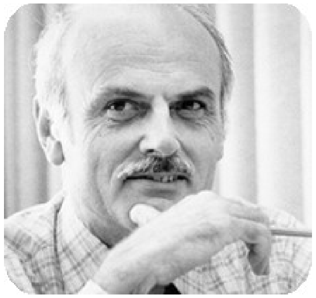
Relational Model Overview

Ted Codd 1970

- What was the motivation? What is the model?

- Logical and physical data independence

- Store data in a **simple data structure** (table)
- Access data through **set-at-a-time** language
- **No need for physical storage proposal**



Great Debate

- Pro relational
  - What were the arguments?
- Against relational
  - What were the arguments?
- How was it settled?

Great Debate

- Pro relational
  - CODASYL is too complex
  - No data independence
  - Record-at-a-time hard to optimize
  - Trees/networks not flexible enough

- Against relational
  - COBOL programmers cannot understand relational languages
  - Impossible to implement efficiently

- Ultimately settled by the market place

Recap

- Physical data independence:
  - SQL: *what* data we want
  - Optimizer: figures out *how* to get it
- Logical data independence
  - Realized in SQL through *views*

Paper Discussion

- Early data models: IMS, CODASYL
- Relational data model
- Semistructured data model

Other Data Models

- Entity-Relationship: 1970’s
  - Successful in logical database design
- Extended Relational: 1980’s
- Semantic: late 1970’s and 1980’s
- Object-oriented: late 1980’s and early 1990’s
  - Address impedance mismatch: relational dbs ←→ OO languages
  - Interesting but ultimately failed (several reasons, see references)
- Object-relational: late 1980’s and early 1990’s
  - User-defined types, ops, functions, and access methods
- Semi-structured: late 1990’s to the present

Semi-structured Data Model

- Main idea: *schema-last*

- Examples:
  - XML
  - Json
  - Protobuf

- All use a *tree data model*

XML Syntax

```
<article mdate="2011-01-11" key="journals/acta/GoodmanS83">
   <author>Nathan Goodman</author>
   <author>Oded Shmueli</author>
   <title>NP-complete Problems Simplified on Tree Schemas.</title>
   <pages>171-178</pages>
   <year>1983</year>
   <journal>Acta Inf.</journal>
   <url>db/journals/acta/acta20.html#GoodmanS83</url>
</article>
```

Semi-structured, self-describing schema

JSon

Example from: http://www.jsonexample.com/

```
myObject = {
  "first": "John",
  "last": "Doe",
  "salary": 70000,
  "registered": true,
  "interests": [ "Reading", “Biking”, "Hacking" ]
}
```

Semi-structured, self-describing schema

Discussion of Semi-structured

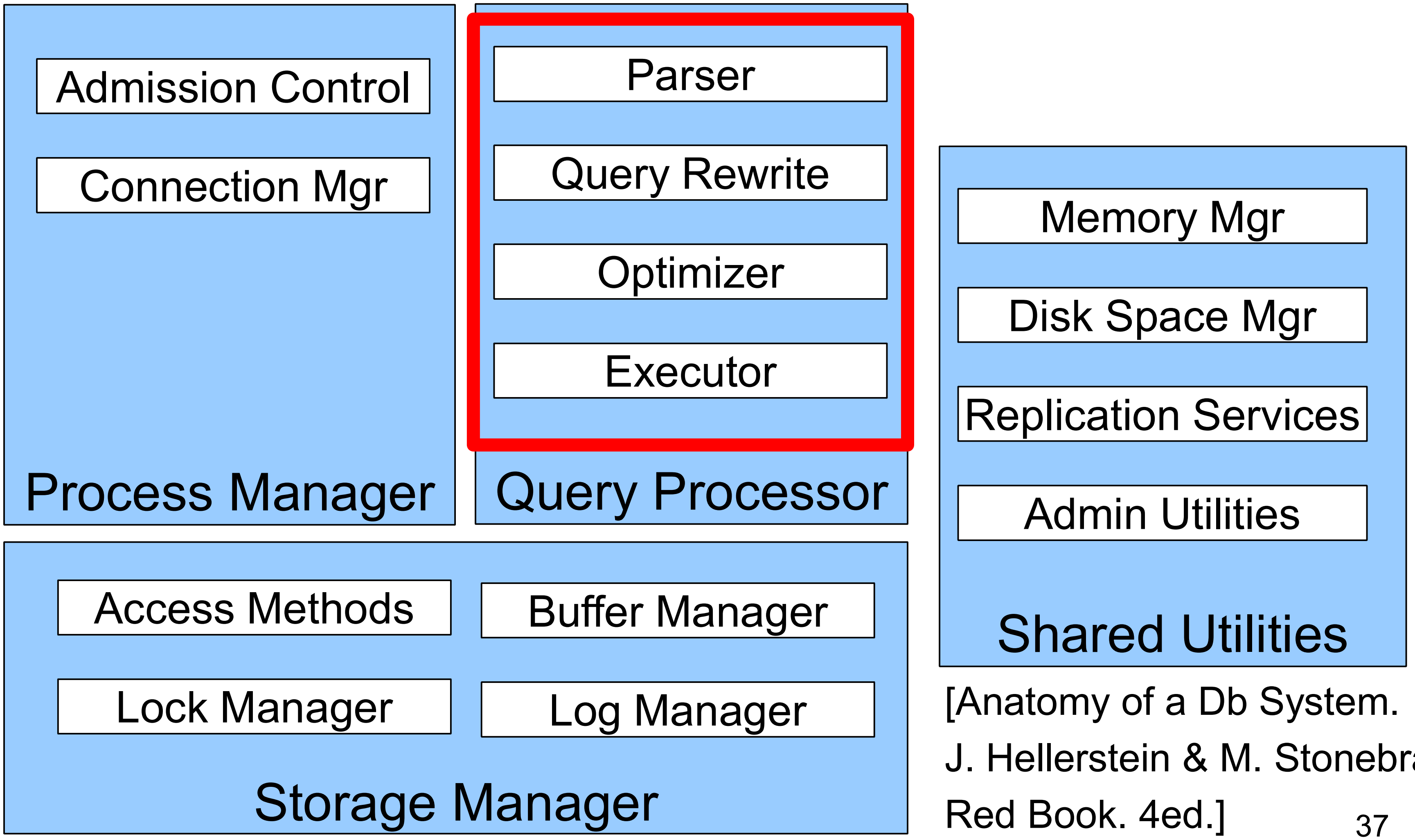
- Stonebraker (circa 1998): niche market
- Today (circa 2020): Json is common
- What changed? Data Science!

Outline for Today

- Discuss *Goes Around* paper

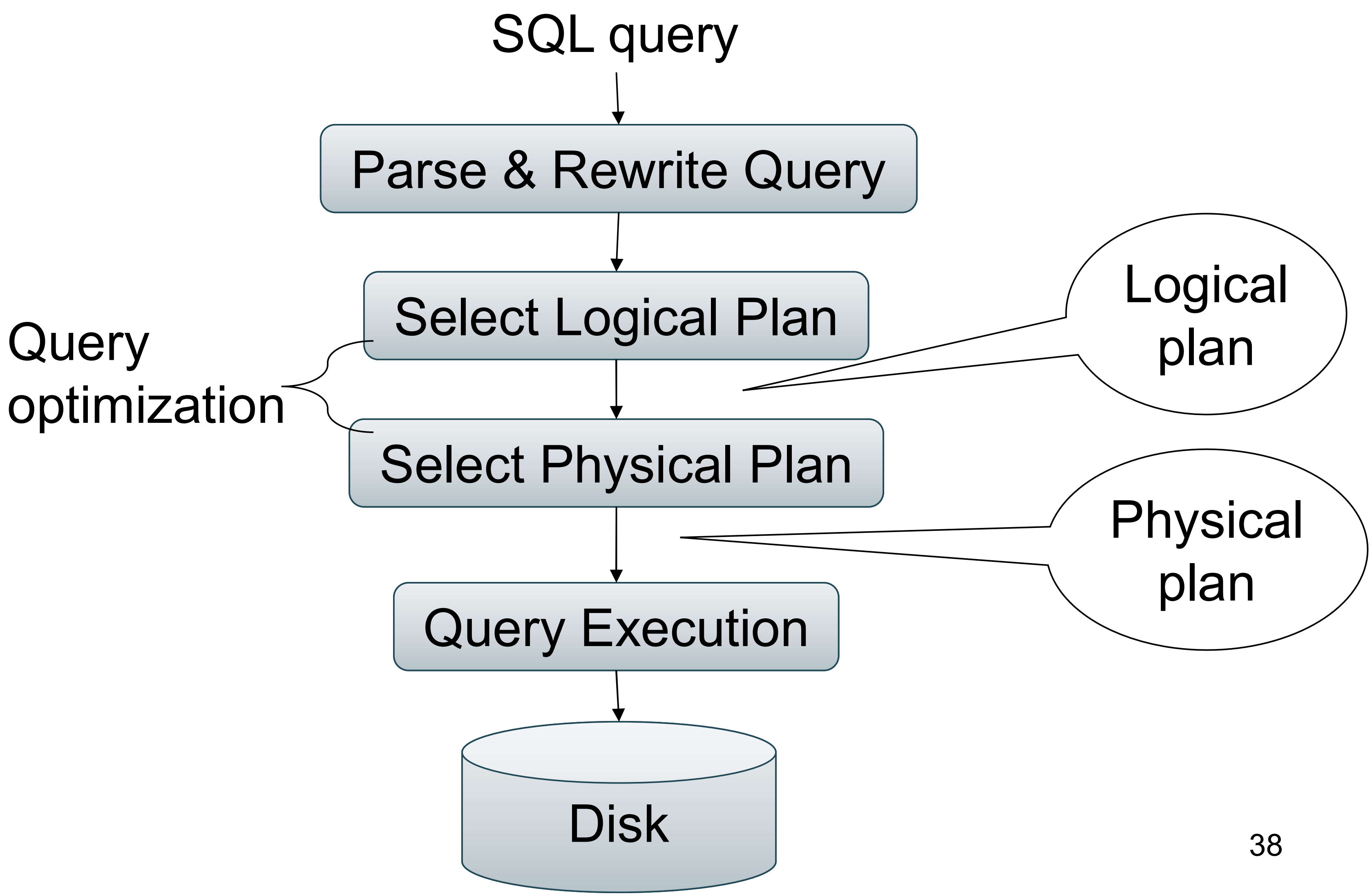
- Discuss query optimization
  - Major paper to read for next time
  - We continue query optimization next time

DBMS Architecture

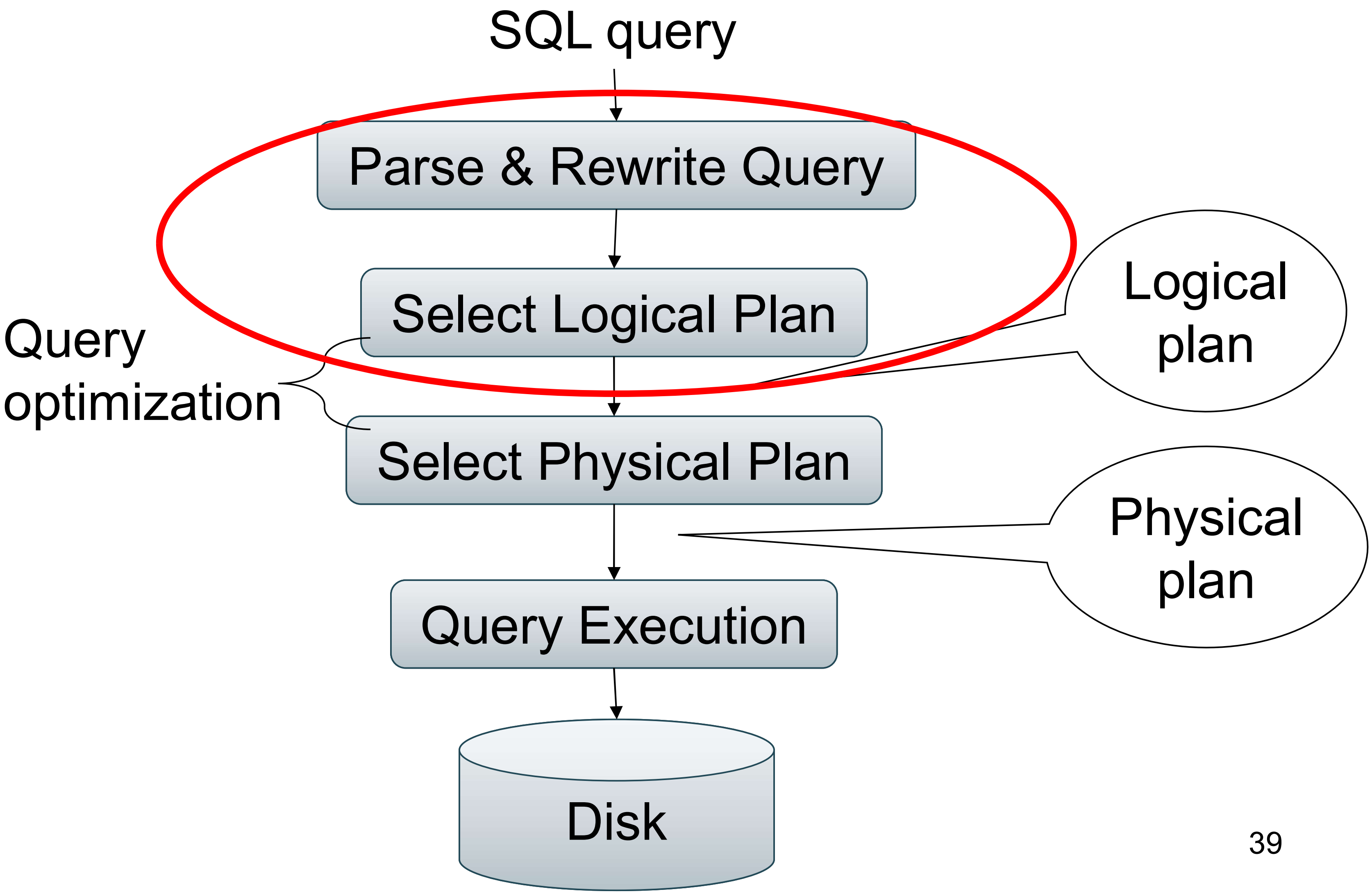


[Anatomy of a Db System.
J. Hellerstein & M. Stonebraker.
Red Book. 4ed.]

Lifecycle of a Query



Lifecycle of a Query



Relational Algebra

Relational Algebra

- A *set-at-a-time* algebra
- Inputs: relations; Output: relation
- E.g. join: R ⋈ S

RA details on next slides

Five Basic Relational Operators

- Selection: $\sigma_{\text{condition}}(S)$
- Projection: $\pi_{\text{list-of-attributes}}(S)$
- Union $(\cup)$
- Set difference $(-)$,
- Cross-product/cartesian product $(\times)$,
  Join: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$

Extended Operators of Relational Algebra

- Duplicate elimination (δ)
- Group-by/aggregate (ɣ)
- Sort operator (τ)

Supplier(sno,sname,scity,sstate)

Supply(sno,pno,price)

Part(pno,pname,psize,pcolor)

Logical Query Plan

- Is an expression in RA
- It specifies in which order to execute the operators

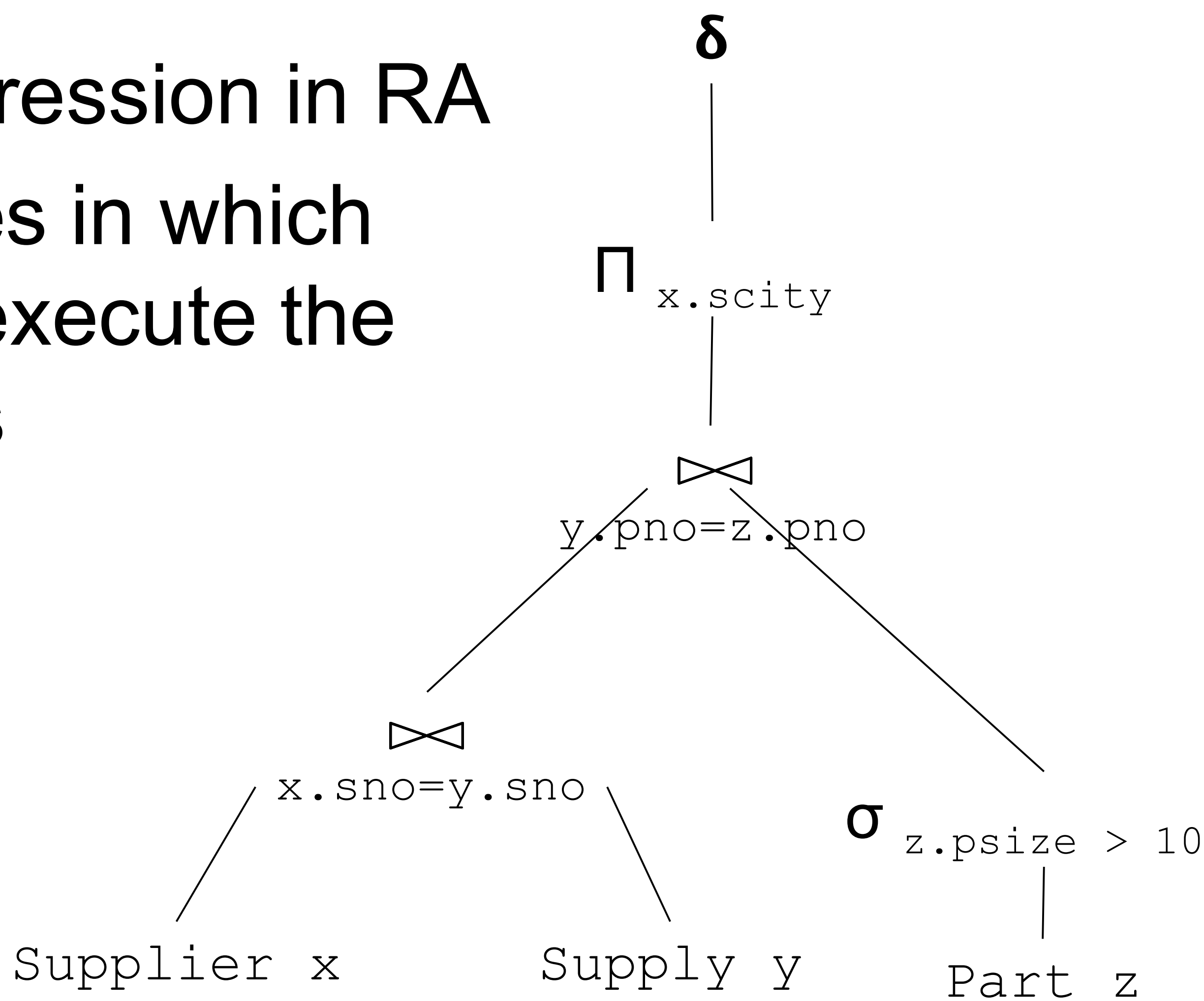
Supplier(sno,sname,scity,sstate)

Supply(sno,pno,price)

Part(pno,pname,psize,pcolor)

Logical Query Plan

- Is an expression in RA
- It specifies in which order to execute the operators



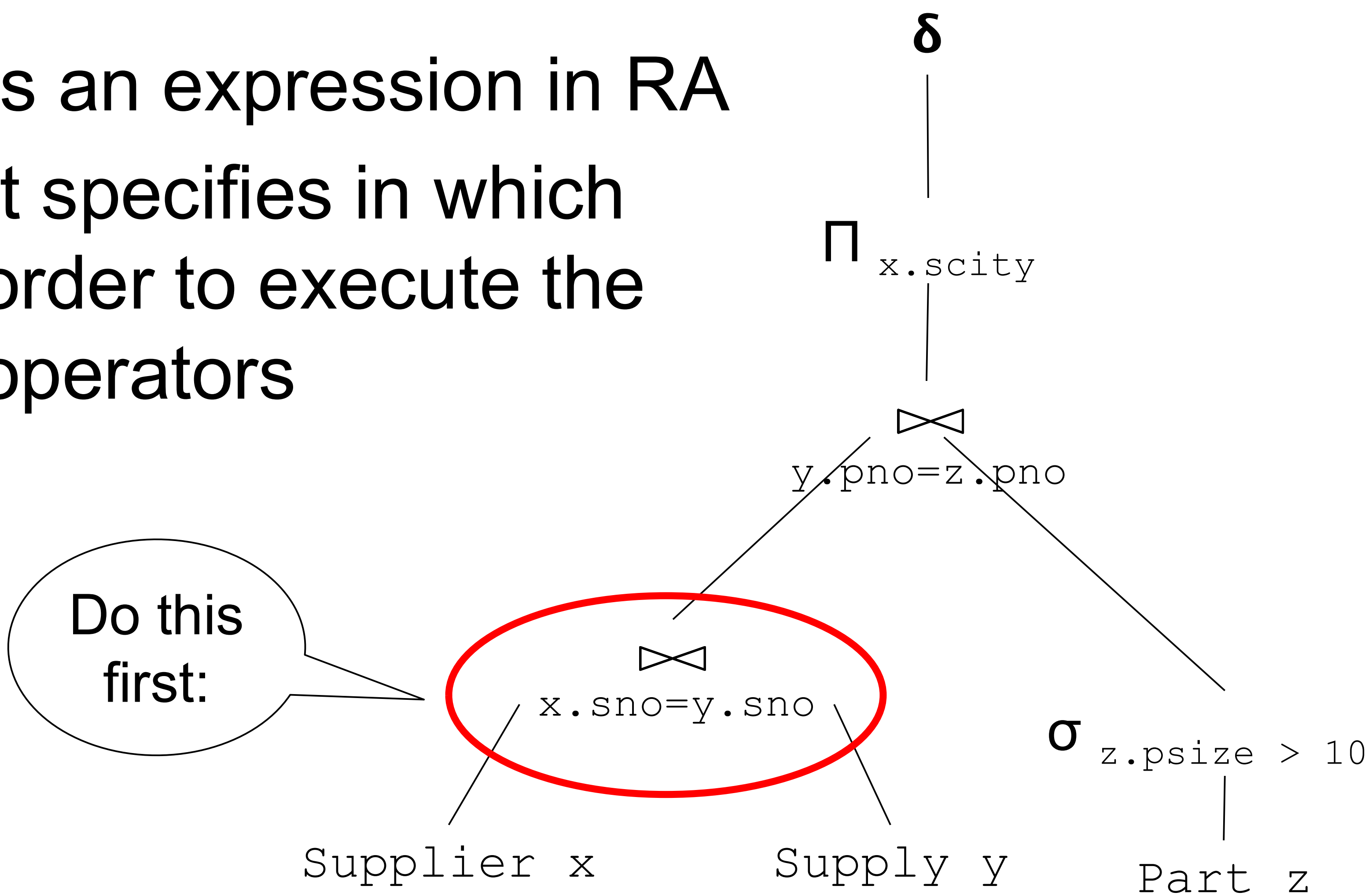
Supplier(sno,sname,scity,sstate)

Supply(sno,pno,price)

Part(pno,pname,psize,pcolor)

Logical Query Plan

- Is an expression in RA
- It specifies in which order to execute the operators



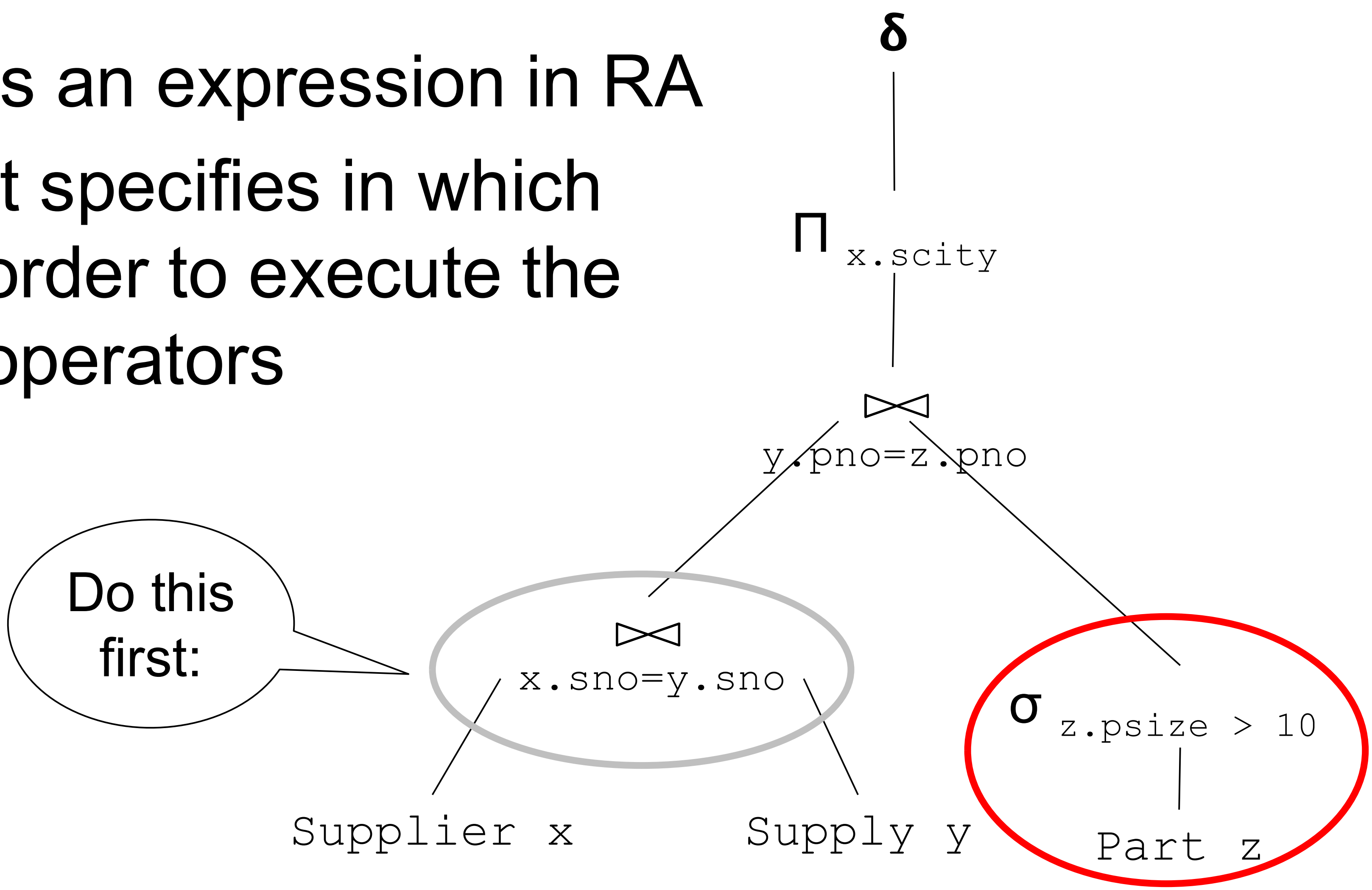
Supplier(sno,sname,scity,sstate)

Supply(sno,pno,price)

Part(pno,pname,psize,pcolor)

Logical Query Plan

- Is an expression in RA
- It specifies in which order to execute the operators



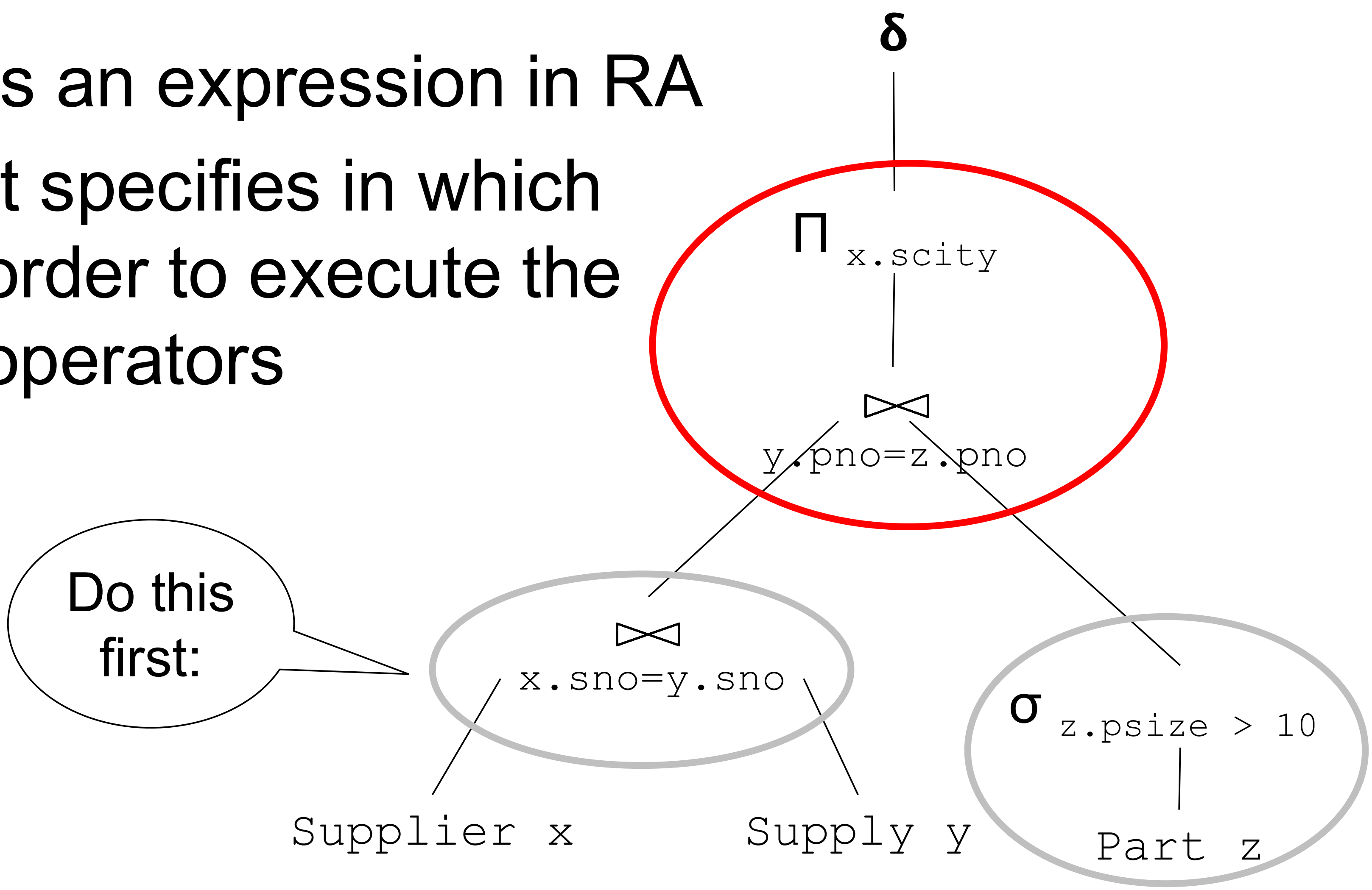
Supplier(sno,sname,scity,sstate)

Supply(sno,pno,price)

Part(pno,pname,psize,pcolor)

Logical Query Plan

- Is an expression in RA
- It specifies in which order to execute the operators



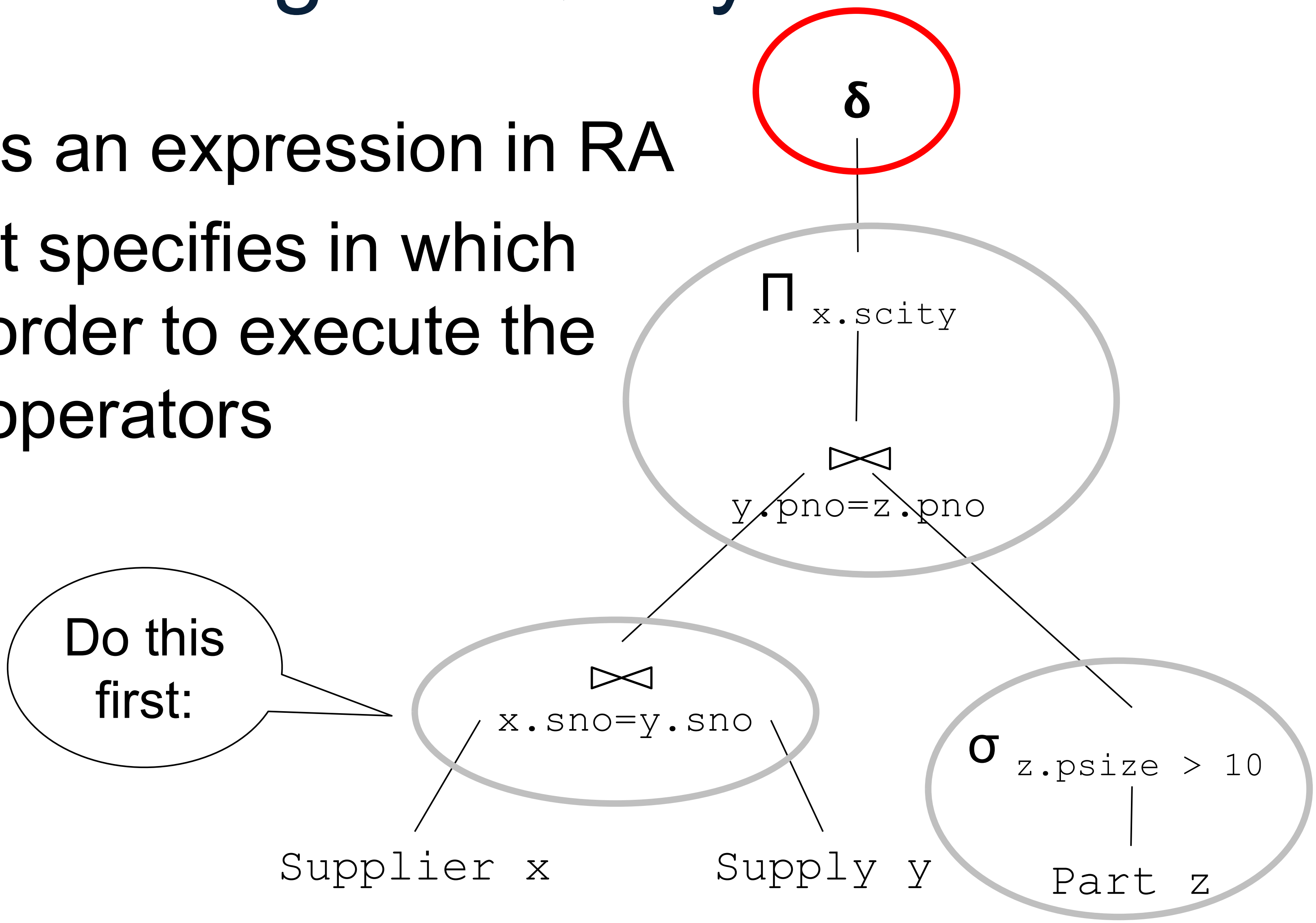
Supplier(sno,sname,scity,sstate)

Supply(sno,pno,price)

Part(pno,pname,psize,pcolor)

Logical Query Plan

- Is an expression in RA
- It specifies in which order to execute the operators



Converting SQL to RA

1. Convert FROM-WHERE to $\bowtie$ and σ

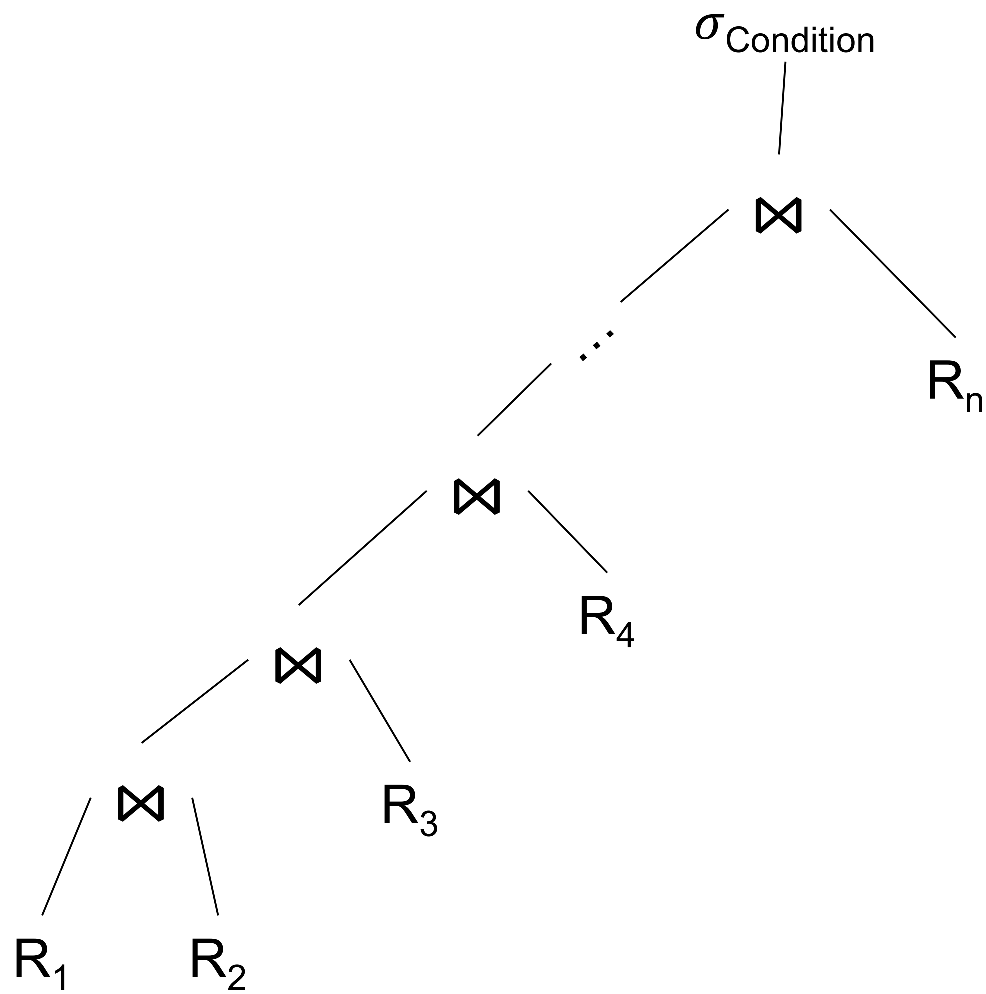
2. Convert GROUP-BY to γ

3. Convert HAVING to σ, SELECT to $\mathbf{\Pi}$

Note: Decorrelate queries (done first)

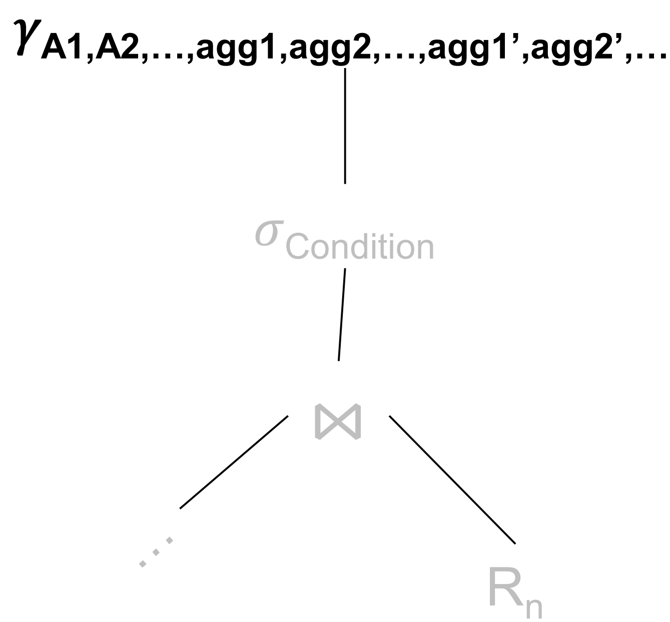
1. FROM-WHERE to ⋈-σ

SELECT …
FROM R1, R2, …
WHERE Condition
GROUP BY …
HAVING …



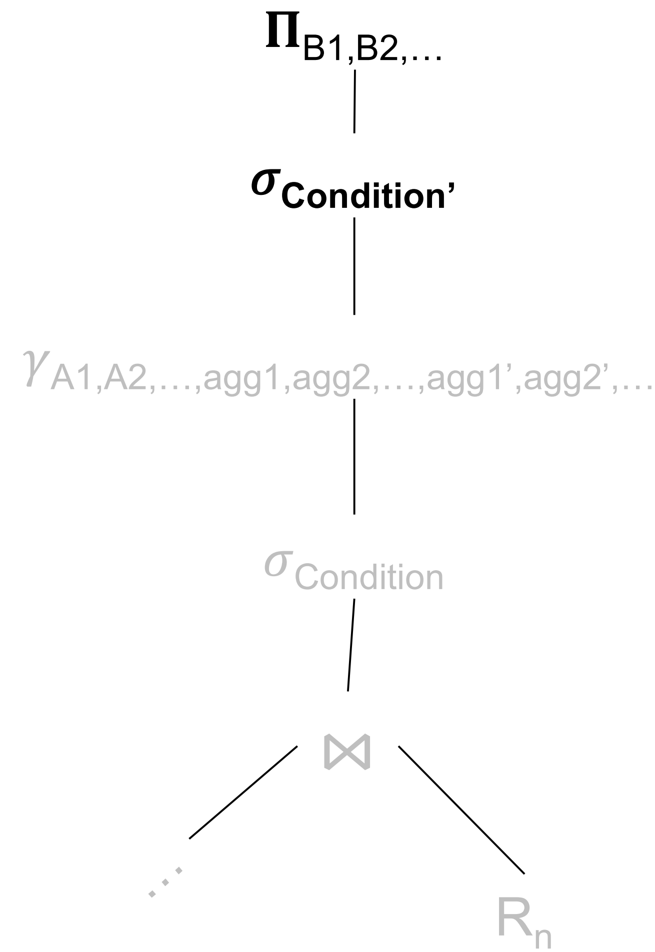
2. GROUP-BY to γ

SELECT …, agg_1,agg_2,…
FROM R1, R2, …
WHERE Condition
GROUP BY A_1, A_2,
HAVING …agg'_1, agg'_2,…



3. HAVING to σ, SELECT to Π

SELECT B1, B2,…, agg_1,…
FROM R1, R2, …
WHERE Condition
GROUP BY A_1, A_2,
HAVING Condition’



Supplier(sno,sname,scity,sstate)

Supply(sno,pno,price)

Part(pno,pname,psize,pcolor)

Example

Find max price of red products for each city that sold > 100 parts

Supplier(sno,sname,scity,sstate)

Supply(sno,pno,price)

Part(pno,pname,psize,pcolor)

Example

Find max price of red products for each city that sold > 100 parts

```
SELECT x.city, max(y.price)
FROM Supplier x, Supply y, Part z,
WHERE x.sno=y.sno and y.pno=z.pno
      and z.pcolor='red'
GROUP BY x.city
HAVING count(*) > 100
```

Supplier(sno,sname,scity,sstate)

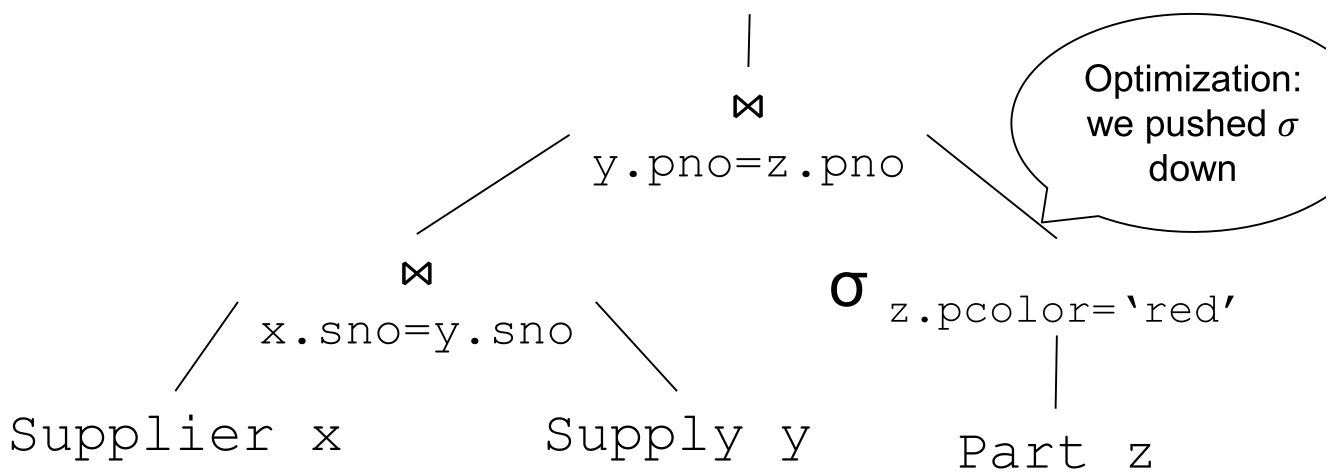
Supply(sno,pno,price)

Part(pno,pname,psize,pcolor)

Example

Find max price of red products for each city that sold > 100 parts

```
SELECT x.city, max(y.price)
FROM Supplier x, Supply y, Part z,
WHERE x.sno=y.sno and y.pno=z.pno
      and z.pcolor='red'
GROUP BY x.city
HAVING count(*) > 100
```



Supplier(sno,sname,scity,sstate)

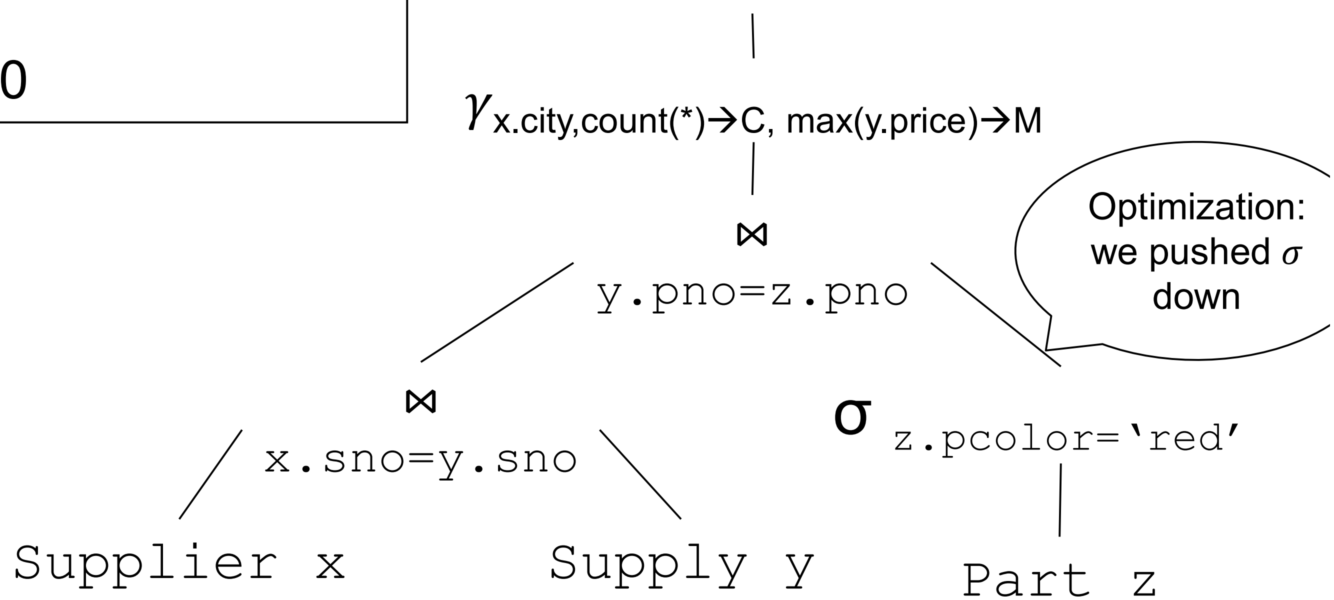
Supply(sno,pno,price)

Part(pno,pname,psize,pcolor)

Example

Find max price of red products for each city that sold > 100 parts

```
SELECT x.city, max(y.price)
FROM Supplier x, Supply y, Part z,
WHERE x.sno=y.sno and y.pno=z.pno
      and z.pcolor='red'
GROUP BY x.city
HAVING count(*) > 100
```

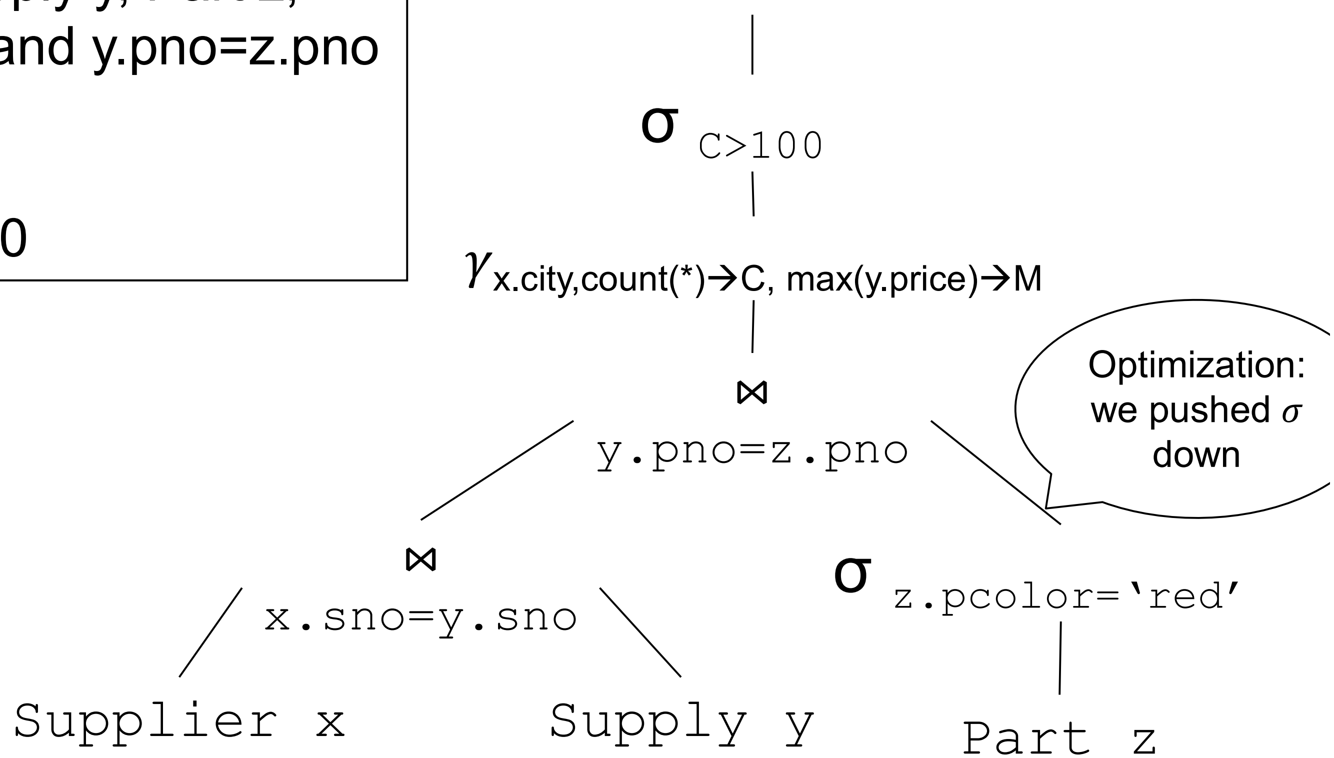


Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Example

Find max price of red products for each city that sold > 100 parts

```
SELECT x.city, max(y.price)
FROM Supplier x, Supply y, Part z,
WHERE x.sno=y.sno and y.pno=z.pno
      and z.pcolor='red'
GROUP BY x.city
HAVING count(*) > 100
```

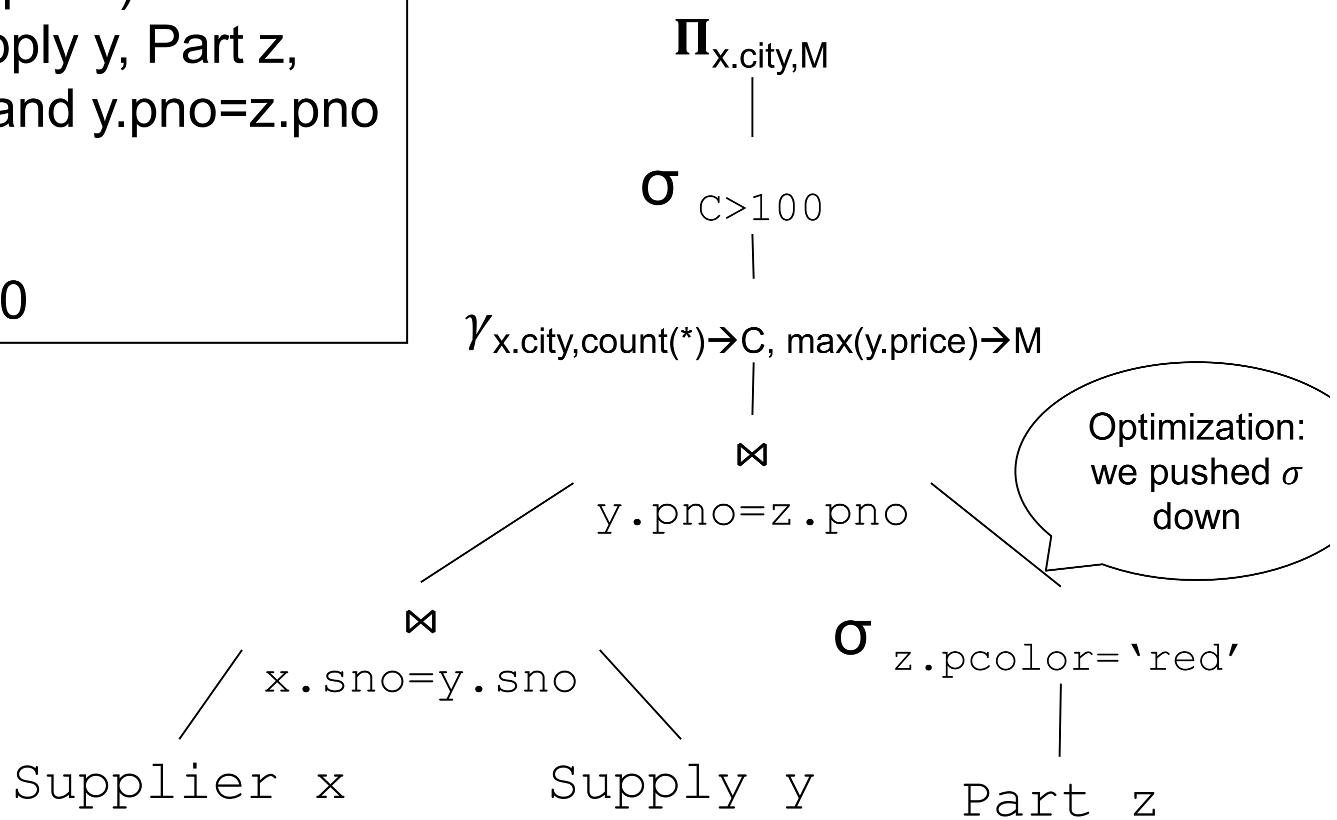


Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)
Part(pno,pname,psize,pcolor)

Example

Find max price of red products for each city that sold > 100 parts

```
SELECT x.city, max(y.price)
FROM Supplier x, Supply y, Part z,
WHERE x.sno=y.sno and y.pno=z.pno
      and z.pcolor='red'
GROUP BY x.city
HAVING count(*) > 100
```



Decorrelation

- A *correlated SQL subquery* is one that depends on a variable of outer query
- This cannot be converted to RA, because does not have variables
- Solution: decorrelation

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)

4.Decorrelation

Find all suppliers in ‘WA’ that supply only parts under $100

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)

4.Decorrelation

```
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)
```

Find all suppliers in 'WA' that supply only parts under $100

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)

4.Decorrelation

```
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)
Supply(sno,pno,price)

4.Decorrelation

```
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)
Supply(sno,pno,price)

4.Decorrelation

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)
```

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

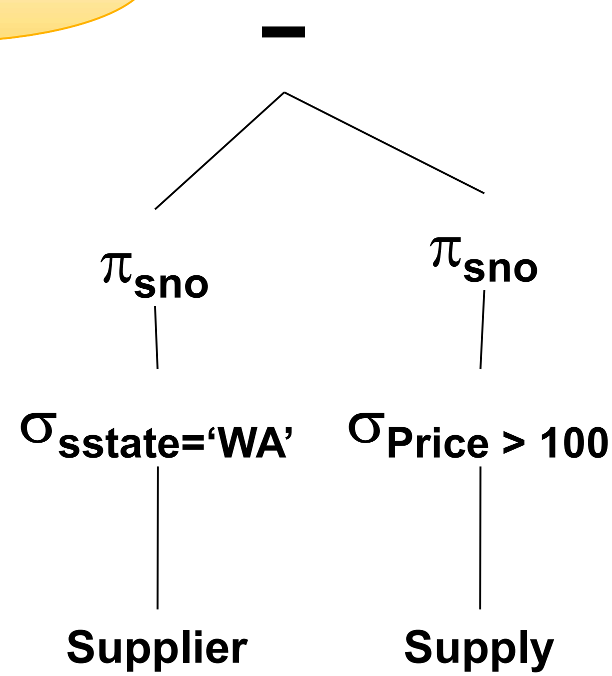
EXCEPT = set difference

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)

4.Decorrelation

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

Finally…

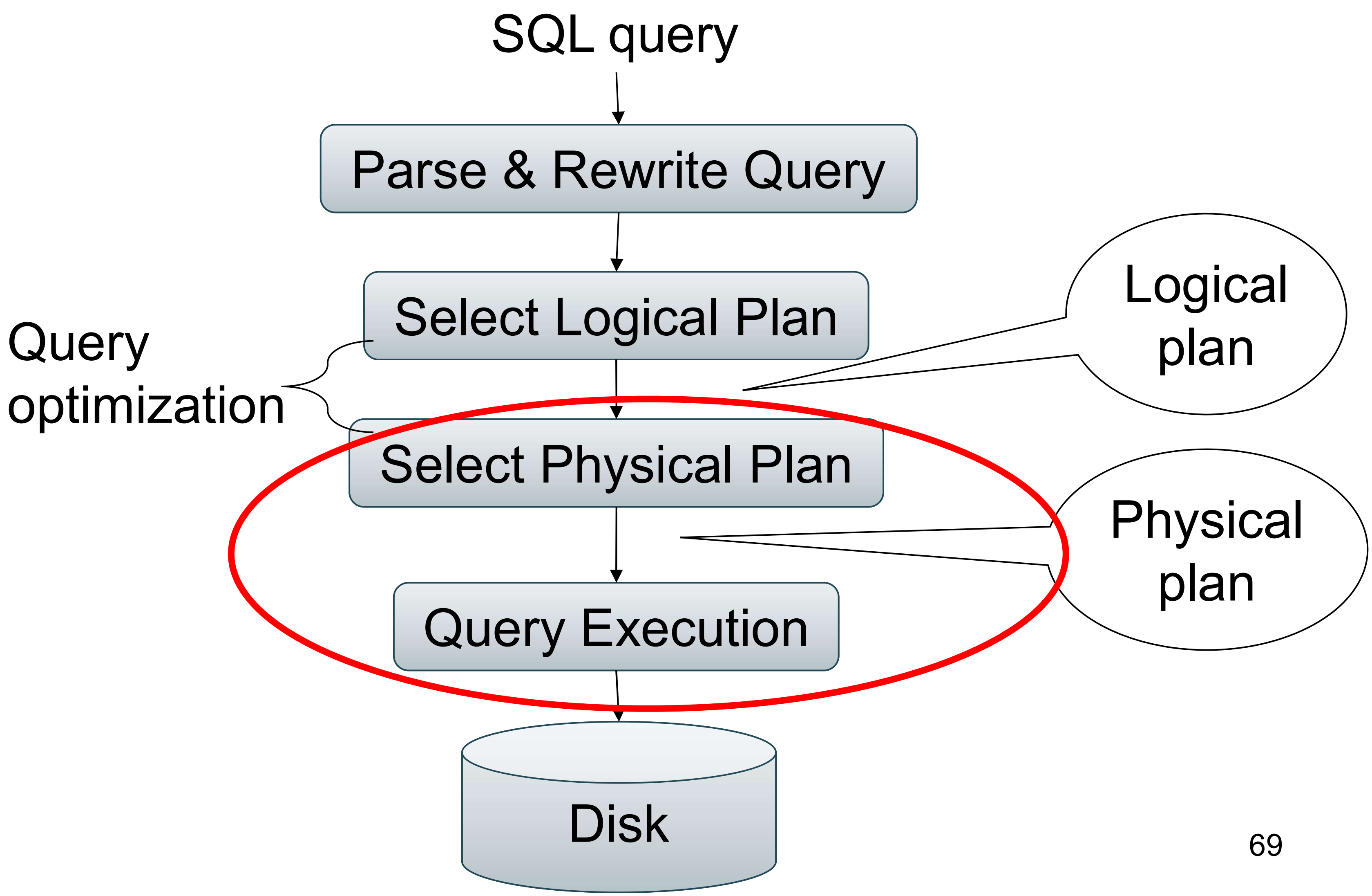


SQL to RA: Summary

- SQL query to Relational Algebra Plan, which is further optimized
- This is a *logical* plan; specifies the order
- Next: *physical* plan; specifies the implementation

Physical Operators

Lifecycle of a Query



Physical Operators

- Each logical operator in RA can be implemented in multiple ways
- An implementation: a *physical operator*

Physical Operators

- Each logical operator in RA can be implemented in multiple ways
- An implementation: a *physical operator*
  - Physical operators for σ: [in class]
  - Physical operators for ∪: [in class]

Physical Operators

- Each logical operator in RA can be implemented in multiple ways
- An implementation: a *physical operator*
  - Physical operators for σ: [in class]
  - Physical operators for $\cup$: [in class]
  - Physical operators for $\bowtie$: next slides
  - Physical operators for γ, δ: not discussed
  - Physical operators for - : not discussed

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

Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

Three algorithms:

1.
2.
3.

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

Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

Three algorithms:

1. Nested Loops
2. Hash-join
3. Merge-join

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

1. Nested Loop Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

```
for x in Supplier do
   for y in Supply do
      if x.sid = y.sid
         then output(x,y)
```

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

1. Nested Loop Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

```
for x in Supplier do
   for y in Supply do
      if x.sid = y.sid
         then output(x,y)
```

If |R|=|S|=n,
what is the runtime?

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

1. Nested Loop Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

```
for x in Supplier do
   for y in Supply do
      if x.sid = y.sid
         then output(x,y)
```

If |R|=|S|=n,
what is the runtime?

$O(n^2)$

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

"Hash Tables"

Key/value pairs; e.g. (sid, Supply)

- insert(k, v) duplicate allowed
- find(k) = returns the ***list*** of values v
- Time is O(1), but can become O(n)
- Collisions!
- Don't write your own hash function

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

2. Hash Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

Build phase

```
for x in Supplier do
    insert(x.sid, x)
```

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

2. Hash Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

Build phase

```
for x in Supplier do
      insert(x.sid, x)
```

Probe phase

```
for y in Supply do
      x = find(y.sid);
      output(x,y);
```

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

2. Hash Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

Build phase

```
for x in Supplier do
      insert(x.sid, x)

for y in Supply do
      x = find(y.sid);
      output(x,y);
```

Probe phase

If |R|=|S|=n,
what is the runtime?

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

2. Hash Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

Build phase

```
for x in Supplier do
      insert(x.sid, x)
```

Probe phase

```
for y in Supply do
      x = find(y.sid);
      output(x,y);
```

If |R|=|S|=n,
what is the runtime?

O(n)

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

2. Hash Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

Change join order

```
for y in Supply do
      insert(y.sid, y)

for x in Supplier do ??
```

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

2. Hash Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

Change join order

```
for y in Supply do
     insert(y.sid, y)

for x in Supplier do
   for y in find(x.sid) do
        output(x,y);
```

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

2. Hash Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

Change join order

```
for y in Supply do
    insert(y.sid, y)

for x in Supplier do
    for y in find(x.sid) do
        output(x,y);
```

If |R|=|S|=n,
what is the runtime?

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

2. Hash Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

Change join order

```
for y in Supply do
      insert(y.sid, y)

for x in Supplier do
    for y in find(x.sid) do
        output(x,y);
```

If |R|=|S|=n,
what is the runtime?

Can be $O(n^2)$ why?

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

2. Hash Join

Why would we change the order?

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

Change join order

```
for y in Supply do
    insert(y.sid, y)

for x in Supplier do
    for y in find(x.sid) do
        output(x,y);
```

If |R|=|S|=n,
what is the runtime?

Can be $O(n^2)$ why?

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

2. Hash Join

Why would we change the order?

When |Supply| << |Supplier|

Change join order

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

```
for y in Supply do
     insert(y.sid, y)

for x in Supplier do
    for y in find(x.sid) do
        output(x,y);
```

If |R|=|S|=n,
what is the runtime?

Can be $O(n^2)$ why?

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

3. Merge Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

```
Sort(Supplier); Sort(Supply);
x = Supplier.first();
y = Supply.first();
```

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

3. Merge Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

```
Sort(Supplier); Sort(Supply);
x = Supplier.first();
y = Supply.first();
while y != NULL do
  case:
    x.sid < y.sid: ???
    x.sid = y.sid: ???
    x.sid > y.sid: ???
```

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

3. Merge Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

```
Sort(Supplier); Sort(Supply);
x = Supplier.first();
y = Supply.first();
while y != NULL do
  case:
    x.sid < y.sid: x = x.next()
    x.sid = y.sid: ???
    x.sid > y.sid: ???
```

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

3. Merge Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

```
Sort(Supplier); Sort(Supply);
x = Supplier.first();
y = Supply.first();
while y != NULL do
  case:
    x.sid < y.sid: x = x.next()
    x.sid = y.sid: output(x,y); y = y.next();
    x.sid > y.sid: ???
```

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

3. Merge Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

```
Sort(Supplier); Sort(Supply);
x = Supplier.first();
y = Supply.first();
while y != NULL do
  case:
    x.sid < y.sid: x = x.next()
    x.sid = y.sid: output(x,y); y = y.next();
    x.sid > y.sid: y = y.next();
```

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

3. Merge Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

```
Sort(Supplier); Sort(Supply);
x = Supplier.first();
y = Supply.first();
while y != NULL do
  case:
    x.sid < y.sid: x = x.next()
    x.sid = y.sid: output(x,y); y = y.next();
    x.sid > y.sid: y = y.next();
```

If |R|=|S|=n,
what is the runtime?

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

3. Merge Join

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

```
Sort(Supplier); Sort(Supply);
x = Supplier.first();
y = Supply.first();
while y != NULL do
  case:
    x.sid < y.sid: x = x.next()
    x.sid = y.sid: output(x,y); y = y.next();
    x.sid > y.sid: y = y.next();
```

If |R|=|S|=n,
what is the runtime?

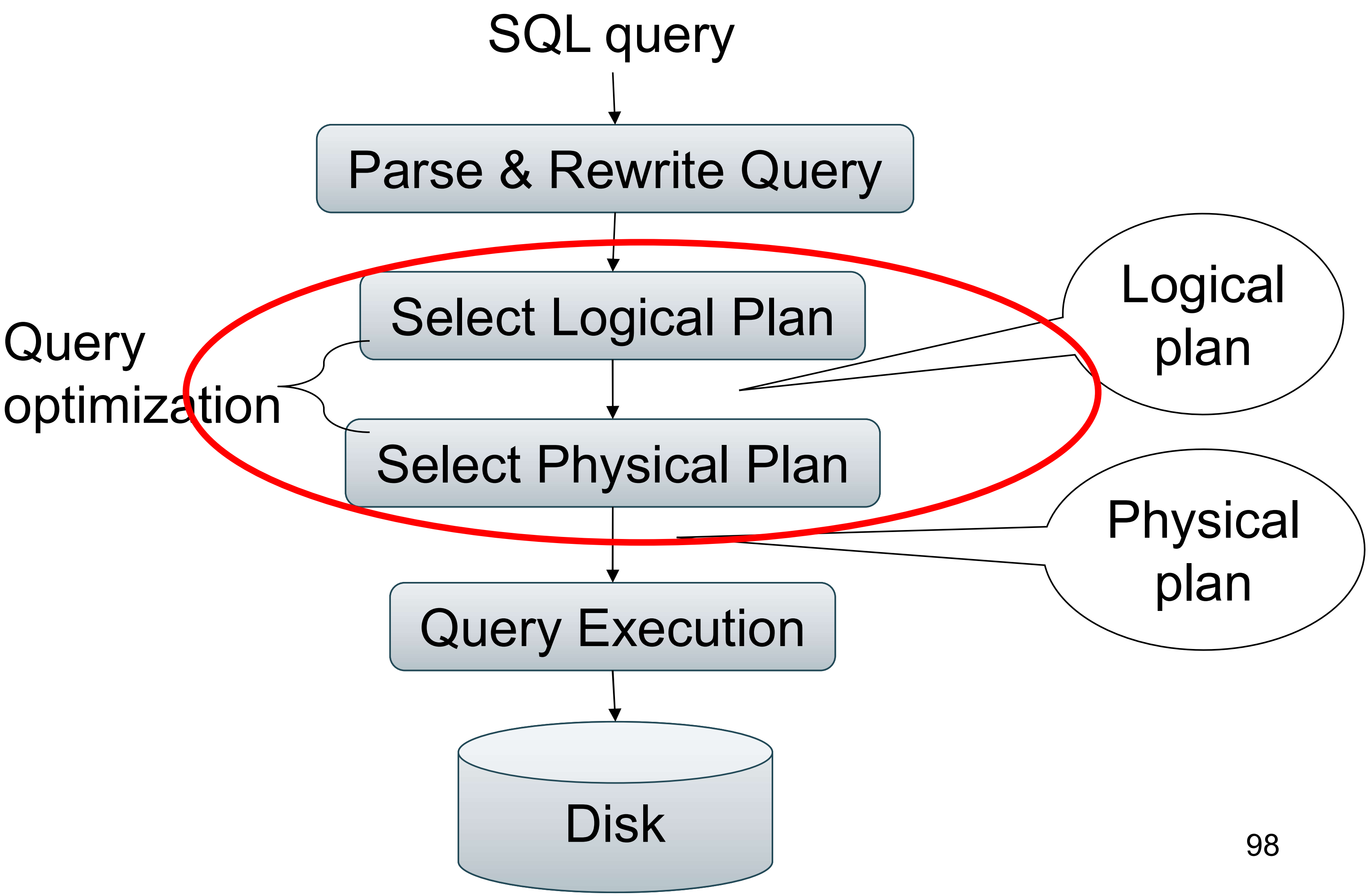
O(n log(n))
(because sorting…)

Summary of Physical Operators

- σ: on-the-fly
- $\cup$: concatenate, then apply δ
- $\bowtie$: nested-loop join, hash-join, merge-join
- γ, δ: nested-loop, hash-based, sort-based
- - : nested-loop, hash-based, sort-based

Query Optimizer

Lifecycle of a Query



Query Optimization

1. Search space

2. Cardinality and cost estimation

3. Plan enumeration algorithms (next time)

Search Space

- Search space = set of rewrite rules that the optimizer implements
- E.g. SQL Server has 400+ rules

We discuss a few basic rewrite rules next

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

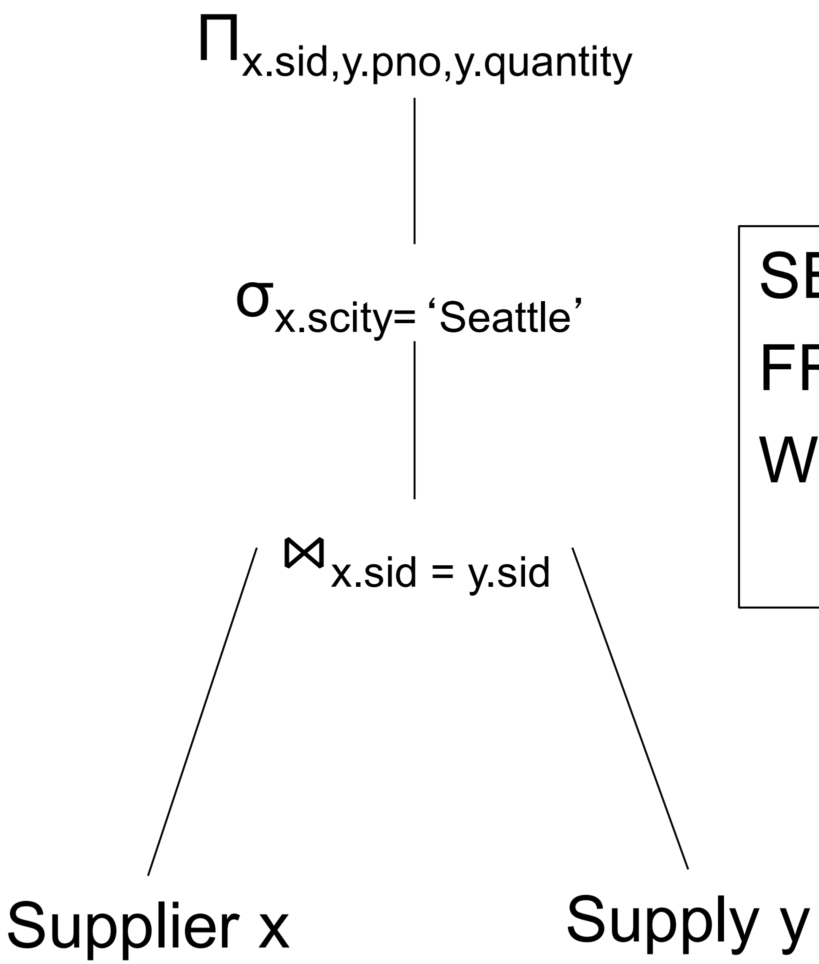
Supply(<u>sid, pno</u>, quantity)

Example Optimization

```
SELECT x.sid, y.pno, y.quantity
FROM.   Supplier x, Supply y
WHERE x.sid = y.sid
        and x.scity = ‘Seattle’
```

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

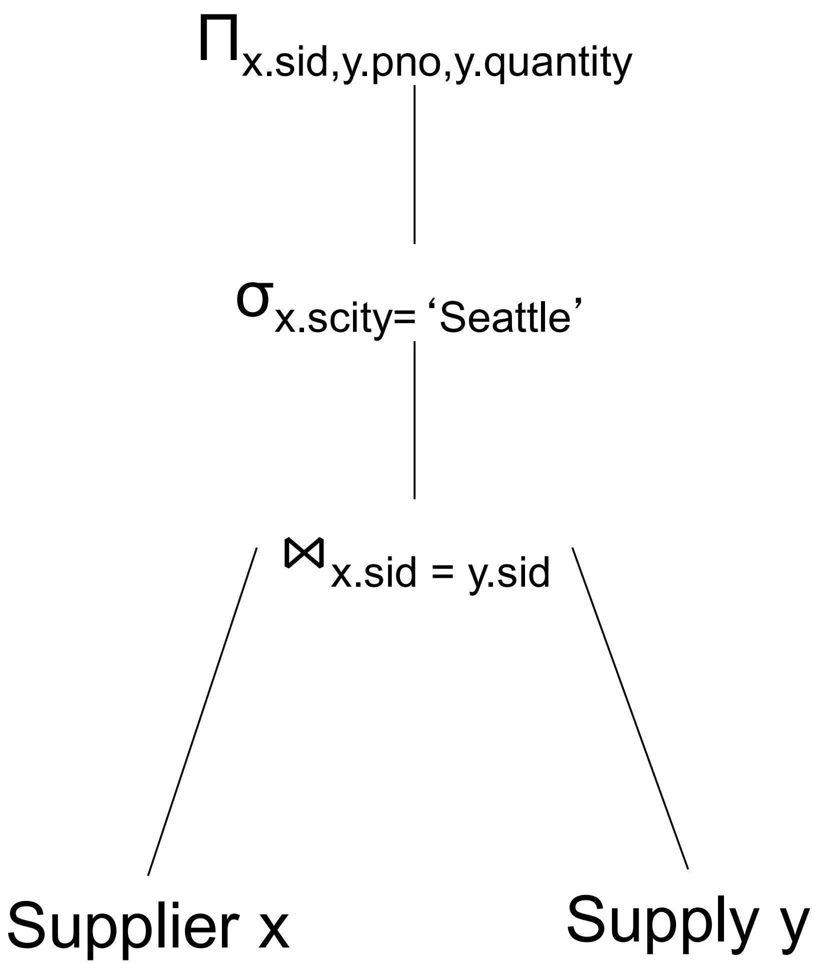
Example Optimization




```
SELECT x.sid, y.pno, y.quantity
FROM.   Supplier x, Supply y
WHERE x.sid = y.sid
      and x.scity = 'Seattle'
```

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

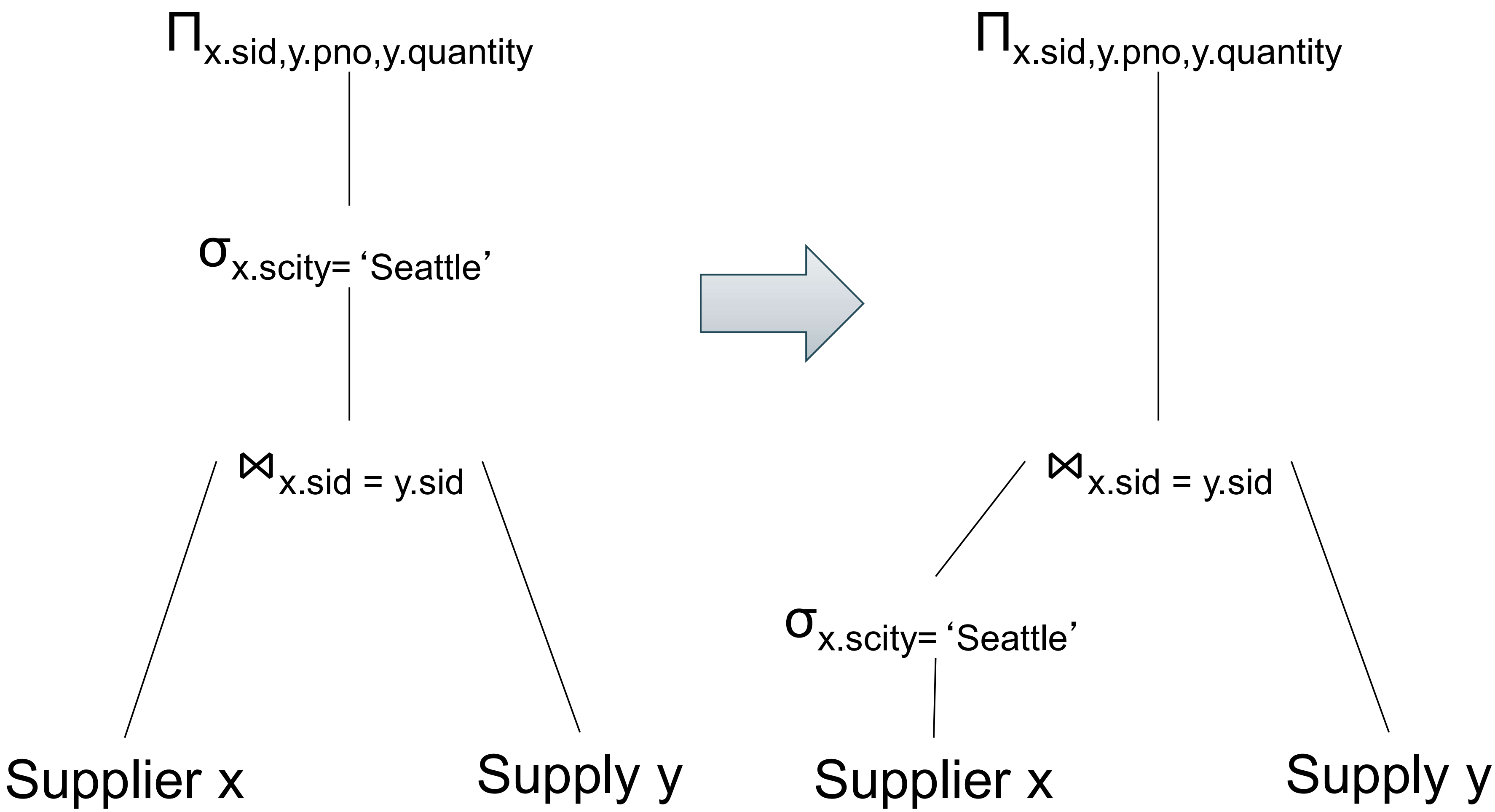
Push Selections Down



Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

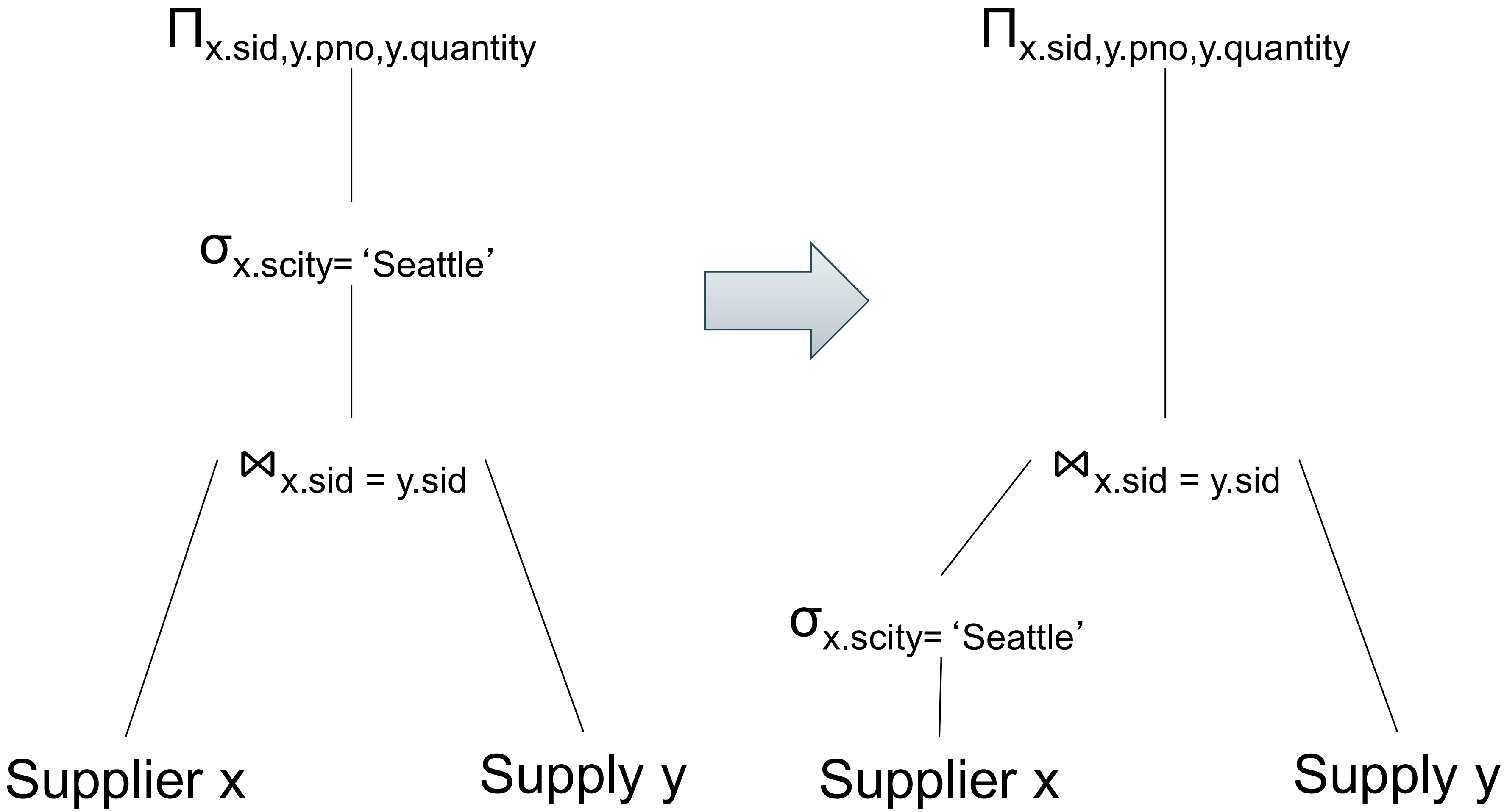
Push Selections Down



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

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

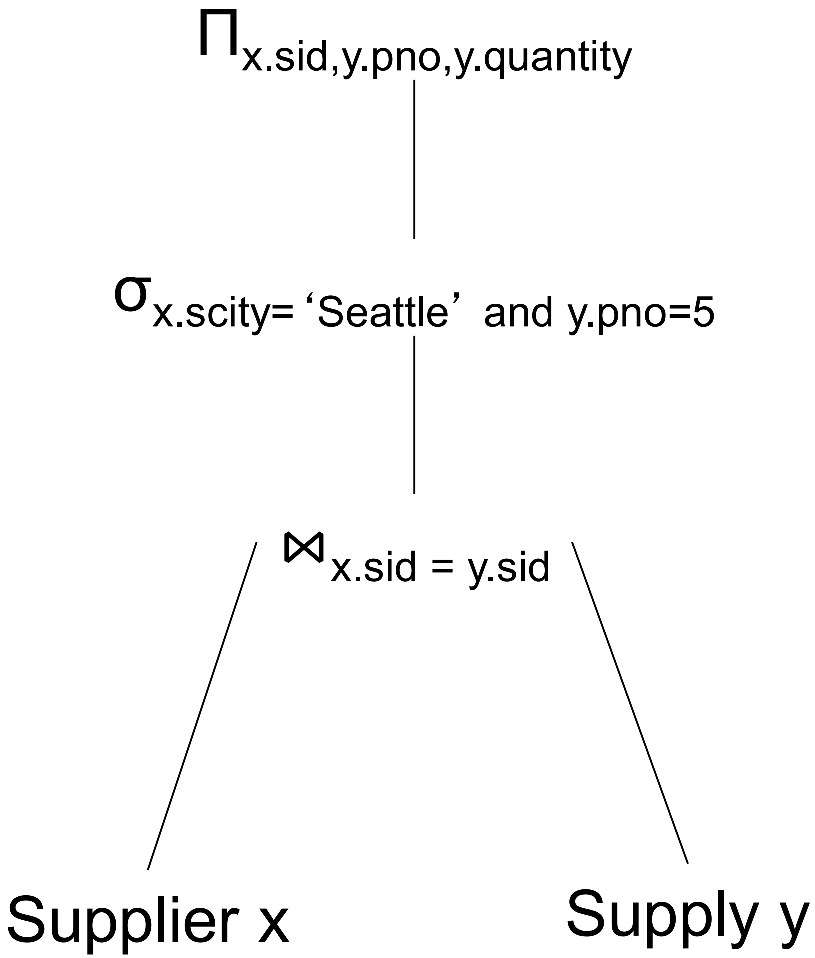
Push Selections Down



$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$ when C refers only to R

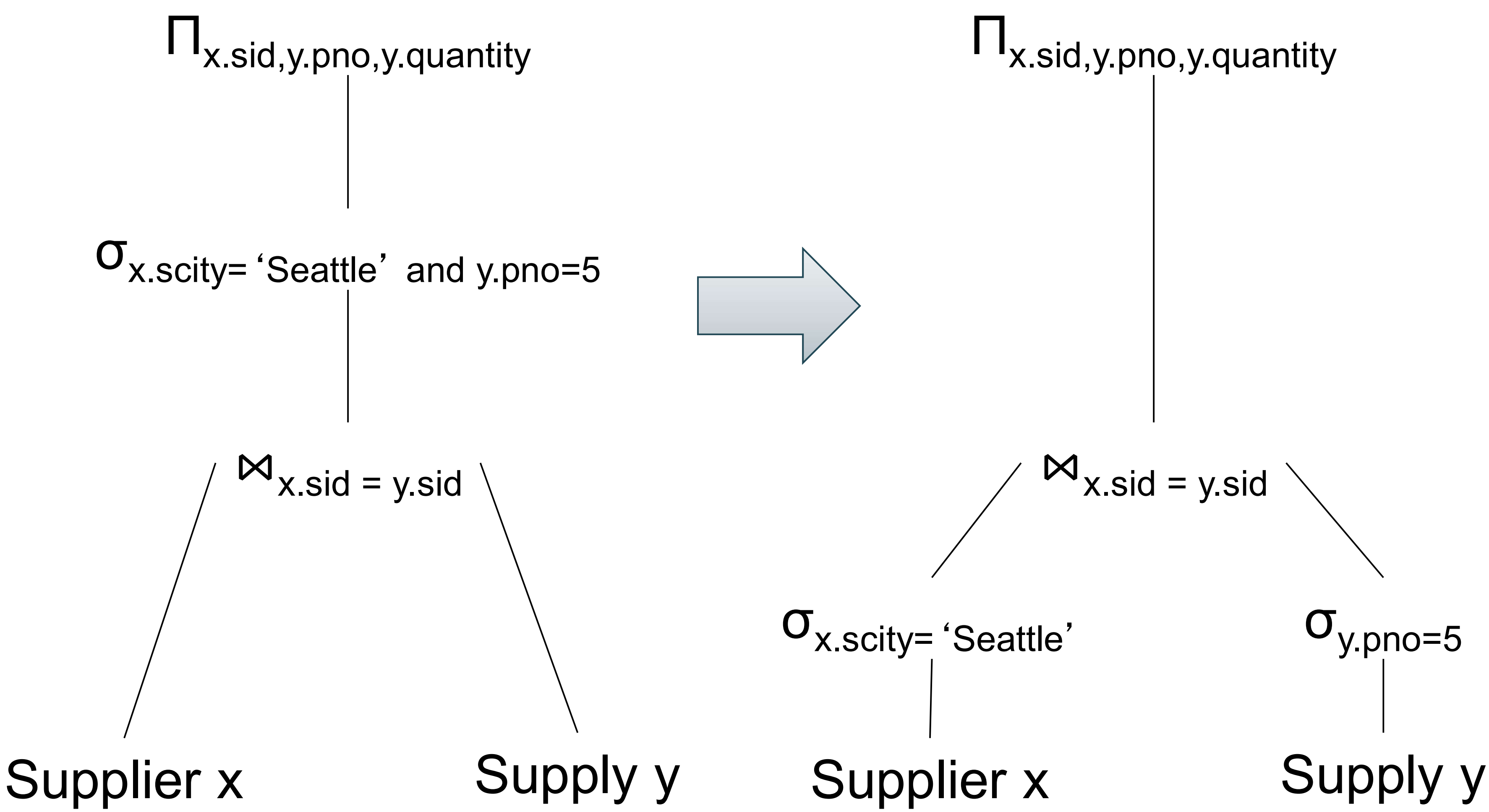
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



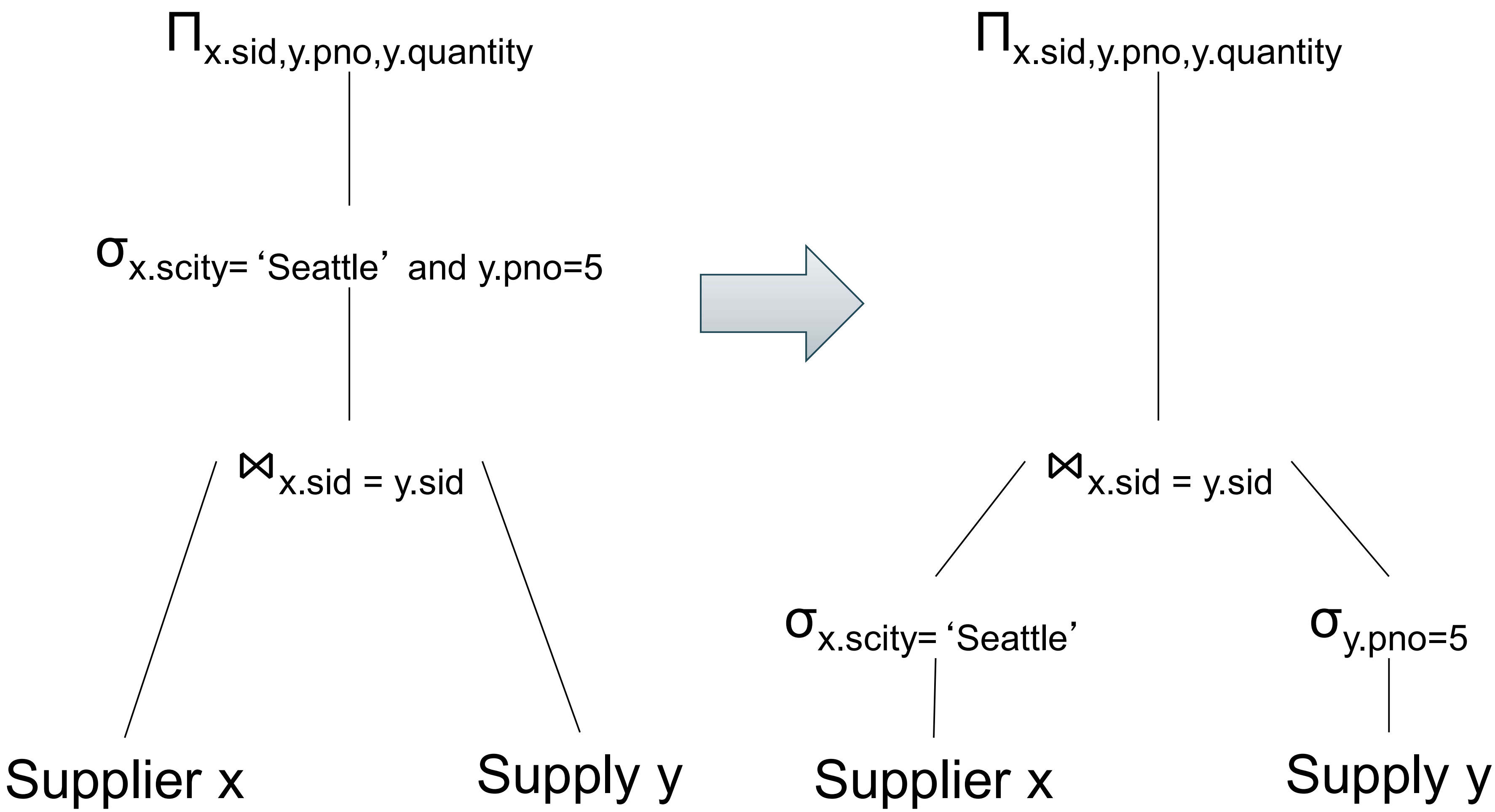
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



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

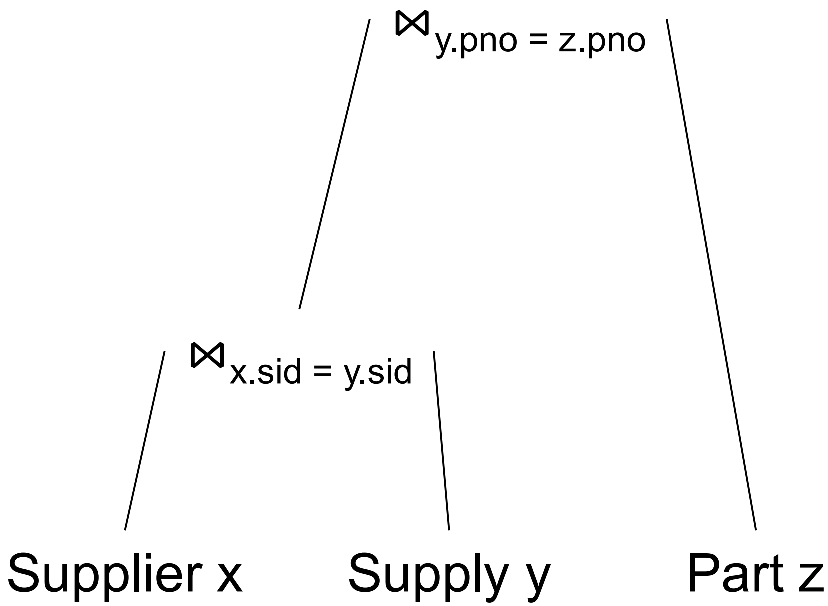
Push Selections Down



$\sigma_{C1 \text{ and } C2}(R \bowtie S) = \sigma_{C1}(\sigma_{C2}(R \bowtie S)) = \sigma_{C1}(R \bowtie \sigma_{C2}(S)) = \sigma_{C1}(R) \bowtie \sigma_{C2}(S)$

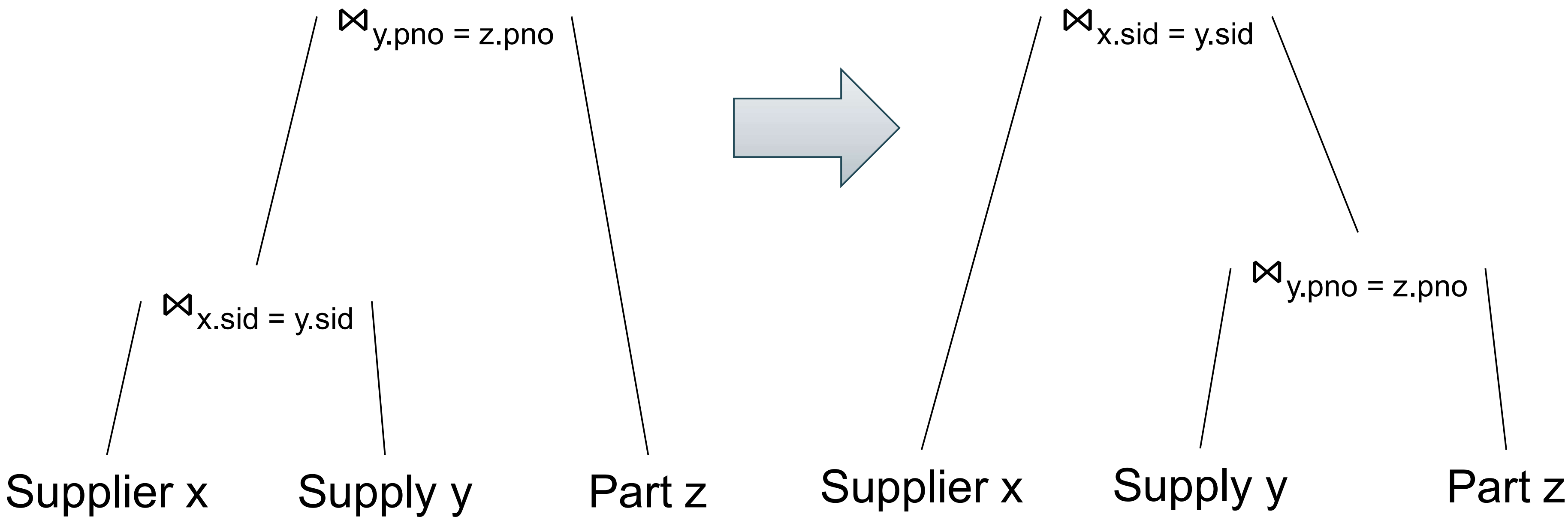
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder



Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder

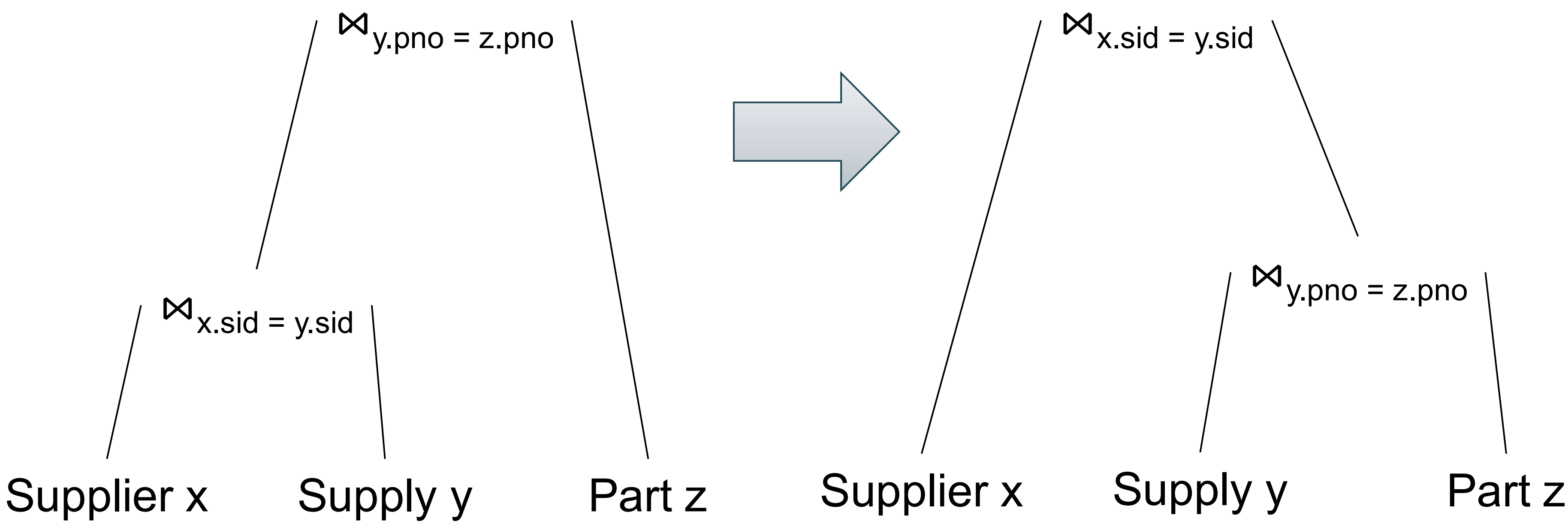


Supplier(<u>sid</u>, sname, scity, sstate)

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

Part(<u>pno</u>, pname, pprice)

Join Reorder



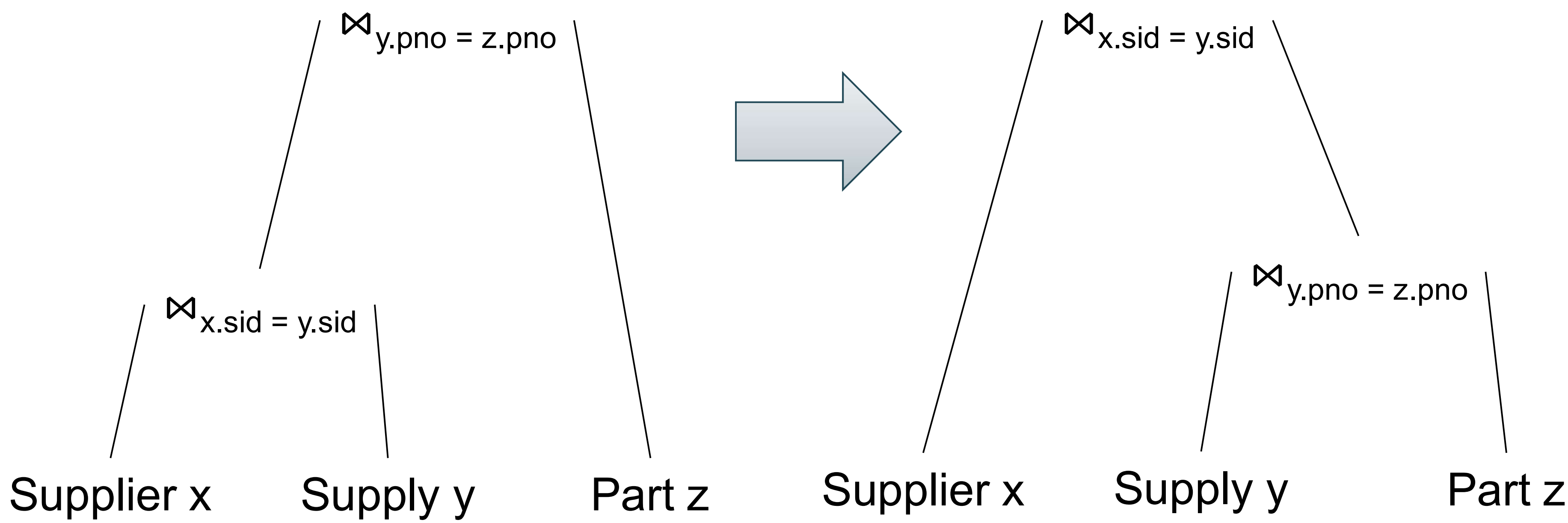
$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

$R \bowtie S = S \bowtie R$

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

Join Reorder

When is one plan better than the other?



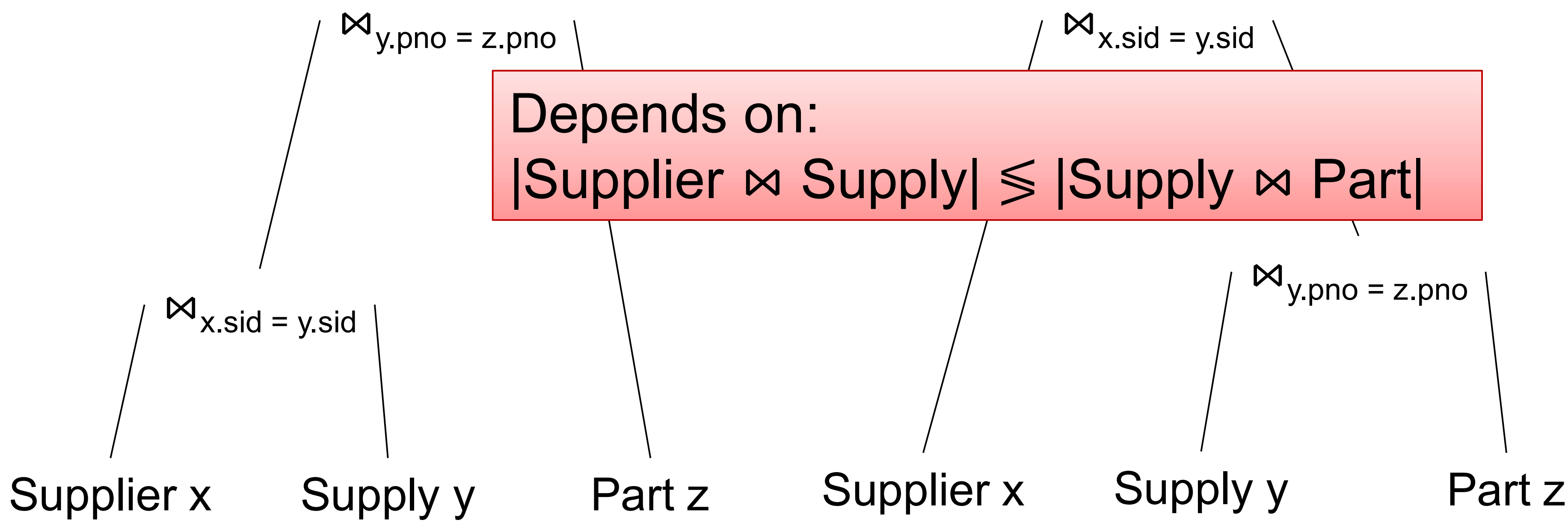
$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

$R \bowtie S = S \bowtie R$

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

Join Reorder

When is one plan better than the other?



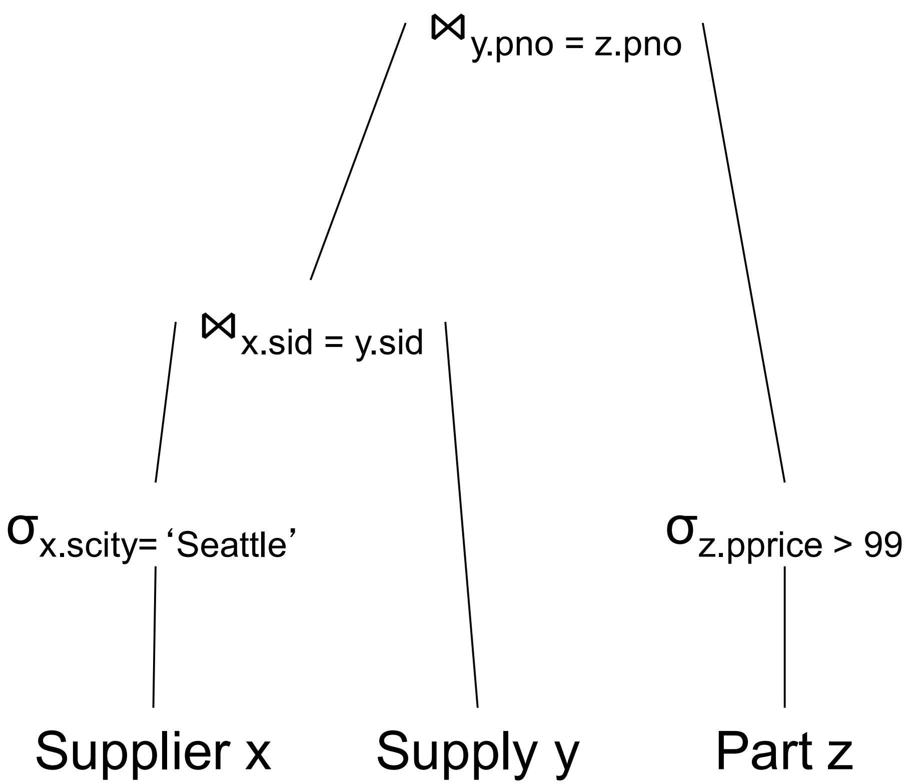
Depends on:
|Supplier ⋈ Supply| ⋚ |Supply ⋈ Part|

$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

$R \bowtie S = S \bowtie R$

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

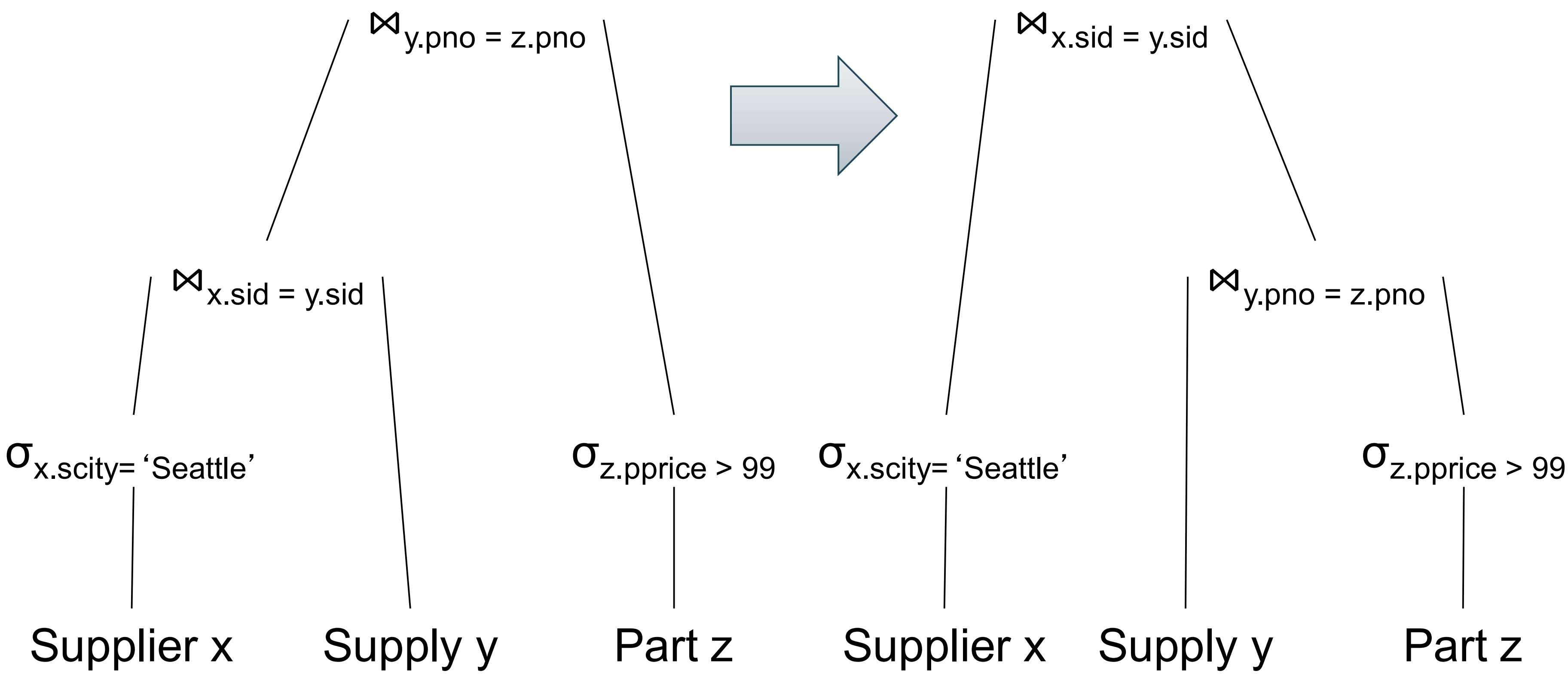
Join Reorder



Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder

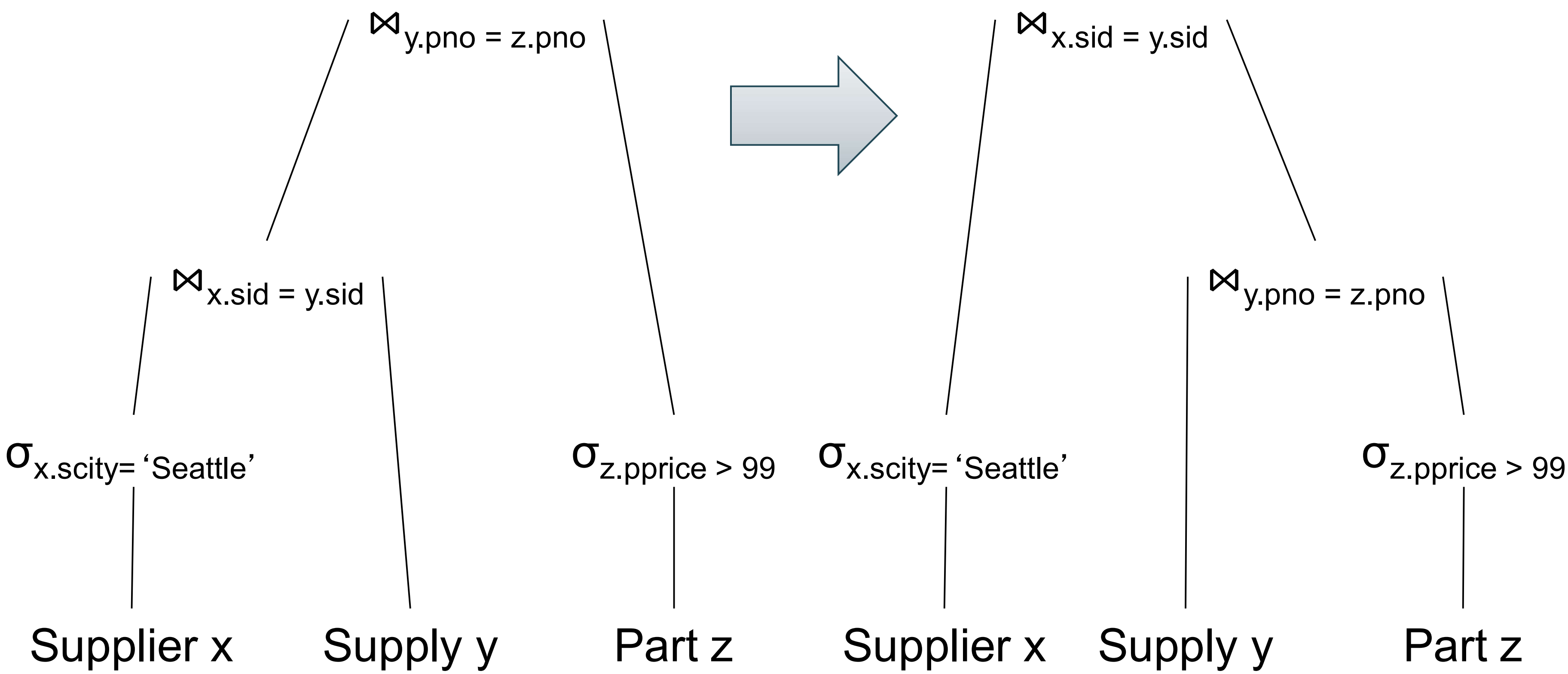
When is one plan better than the other?



Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder

When is one plan better than the other?



Lesson: need sizes of $\sigma_{x.scity=\text{'Seattle'}}$(Supplier), $\sigma_{z.pprice > 99}$(Part)

Search Space: Summary

- Large set of rewrite rules
- Generates many candidate plans
- Need to *estimate their cost*, choose best

Query Optimization

1. Search space
2. Cardinality and cost estimation
3. Plan enumeration algorithms (next time)

Cardinality Estimation

Problem: given statistics on base tables and a query, estimate size of the answer

Challenging, because:

- Need to do it very fast
- Need to use very little memory

Statistics on Base Data

- Number of tuples (cardinality) T(R)
- Number of physical pages B(R)
- Indexes, number of keys in the index V(R,a)

- Histograms: 1d or 2d (next lecture)

Computed periodically, often using sampling

Assumptions

- Uniformity
- Independence
- Containment of values
- Preservation of values

Size Estimation

Selection: size decreases by ***selectivity factor*** θ

$$T(\sigma_{pred}(R)) = \theta_{pred} * T(R)$$

Size Estimation

Selection: size decreases by *selectivity factor* θ

$T(\sigma_{pred}(R)) = \theta_{pred} * T(R)$

$T(R \bowtie_{A=B} S) = \theta_{A=B} * T(R) * T(S)$

$T(\sigma_{pred}(R)) = \theta_{pred} * T(R)$

Selectivity Factors

Uniformity assumption

Equality:

$\sigma_{A=c}(R)$

$T(\sigma_{pred}(R)) = \theta_{pred} * T(R)$

Selectivity Factors

Uniformity assumption

$\sigma_{A=c}(R)$

Equality:

- $\theta_{A=c} = 1/V(R,A)$

$T(\sigma_{pred}(R)) = \theta_{pred} * T(R)$

Selectivity Factors

Uniformity assumption

$\sigma_{A=c}(R)$

Equality:

- $\theta_{A=c} = 1/V(R,A)$

$\sigma_{c1<A<c2}(R)$

Range:

- $\theta_{c1<A<c2} = (c2 - c1)/(max(R,A) - min(R,A))$

$T(\sigma_{pred}(R)) = \theta_{pred} * T(R)$

Selectivity Factors

Uniformity assumption

$\sigma_{A=c}(R)$

Equality:

- $\theta_{A=c} = 1/V(R,A)$

$\sigma_{c1<A<c2}(R)$

Range:

- $\theta_{c1<A<c2} = (c2 - c1)/(max(R,A) - min(R,A))$

Conjunction

$\sigma_{A=c \text{ and } B=d}(R)$

$T(\sigma_{pred}(R)) = \theta_{pred} * T(R)$

Selectivity Factors

Uniformity assumption

$\sigma_{A=c}(R)$

Equality:

- $\theta_{A=c} = 1/V(R,A)$

$\sigma_{c1<A<c2}(R)$

Range:

- $\theta_{c1<A<c2} = (c2 - c1)/(max(R,A) - min(R,A))$

Conjunction

$\sigma_{A=c \text{ and } B=d}(R)$

Independence assumption

- $\theta_{pred1 \text{ and } pred2} = \theta_{pred1} * \theta_{pred2} = 1/V(R,A) * 1/V(R,B)$

$T(R \bowtie_{A=B} S) = \theta_{A=B} * T(R) * T(S)$

Selectivity Factors

$R \bowtie_{R.A=S.B} S$

Join

$T(R \bowtie_{A=B} S) = \theta_{A=B} * T(R) * T(S)$

Selectivity Factors

$R \bowtie_{R.A=S.B} S$

Join

- $\theta_{R.A=S.B} = 1/ (MAX(V(R,A), V(S,B))$

Why? Will explain next...

$T(R \bowtie_{A=B} S) = \theta_{A=B} * T(R) * T(S)$

Selectivity Factors

$R \bowtie_{R.A=S.B} S$

Containment of values: if $V(R,A) \leq V(S,B)$, then the set of A values of R is included in the set of B values of S

- Note: this indeed holds when A is a foreign key in R, and B is a key in S

$T(R \bowtie_{A=B} S) = \theta_{A=B} * T(R) * T(S)$

Selectivity Factors

$R \bowtie_{R.A=S.B} S$

Assume V(R,A) ≤ V(S,B)

- Tuple t in R joins with T(S)/V(S,B) tuples in S

$T(R \bowtie_{A=B} S) = \theta_{A=B} * T(R) * T(S)$

Selectivity Factors

$R \bowtie_{R.A=S.B} S$

Assume $V(R,A) \leq V(S,B)$

- Tuple t in R joins with $T(S)/V(S,B)$ tuples in S
- Hence $T(R \bowtie_{A=B} S) = T(R)\ T(S) / V(S,B)$

$T(R \bowtie_{A=B} S) = \theta_{A=B} * T(R) * T(S)$

Selectivity Factors

$R \bowtie_{R.A=S.B} S$

Assume $V(R,A) \leq V(S,B)$

- Tuple t in R joins with $T(S)/V(S,B)$ tuples in S
- Hence $T(R \bowtie_{A=B} S) = T(R)\ T(S) / V(S,B)$

In general:

- $T(R \bowtie_{A=B} S) = T(R)\ T(S) / \max(V(R,A),V(S,B))$
- $\theta_{R.A=S.B} = 1/ (\max(V(R,A), V(S,B))$

Final Assumption

Preservation of values:

For any other attribute C:

- $V(R \bowtie_{A=B} S, C) = V(R, C)$ or
- $V(R \bowtie_{A=B} S, C) = V(S, C)$

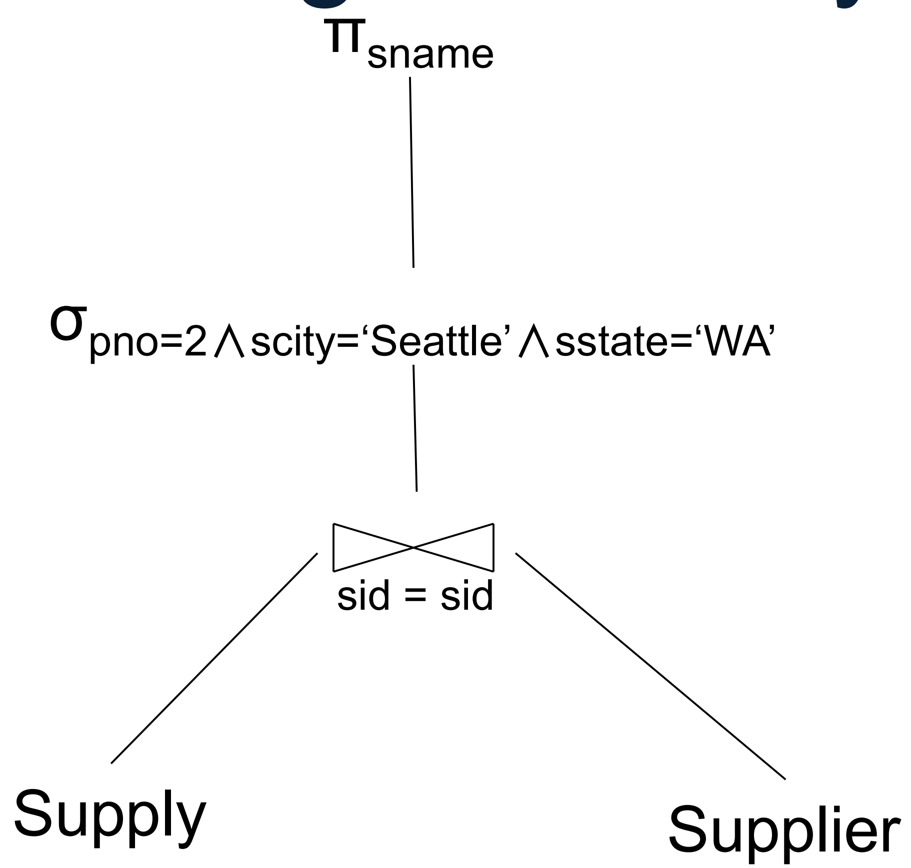
- This is needed higher up in the plan

Computing the Cost of a Plan

- Estimate cardinalities bottom-up
- Estimate cost by using estimated cardinalities
- Examples next...

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

Logical Query Plan 1




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

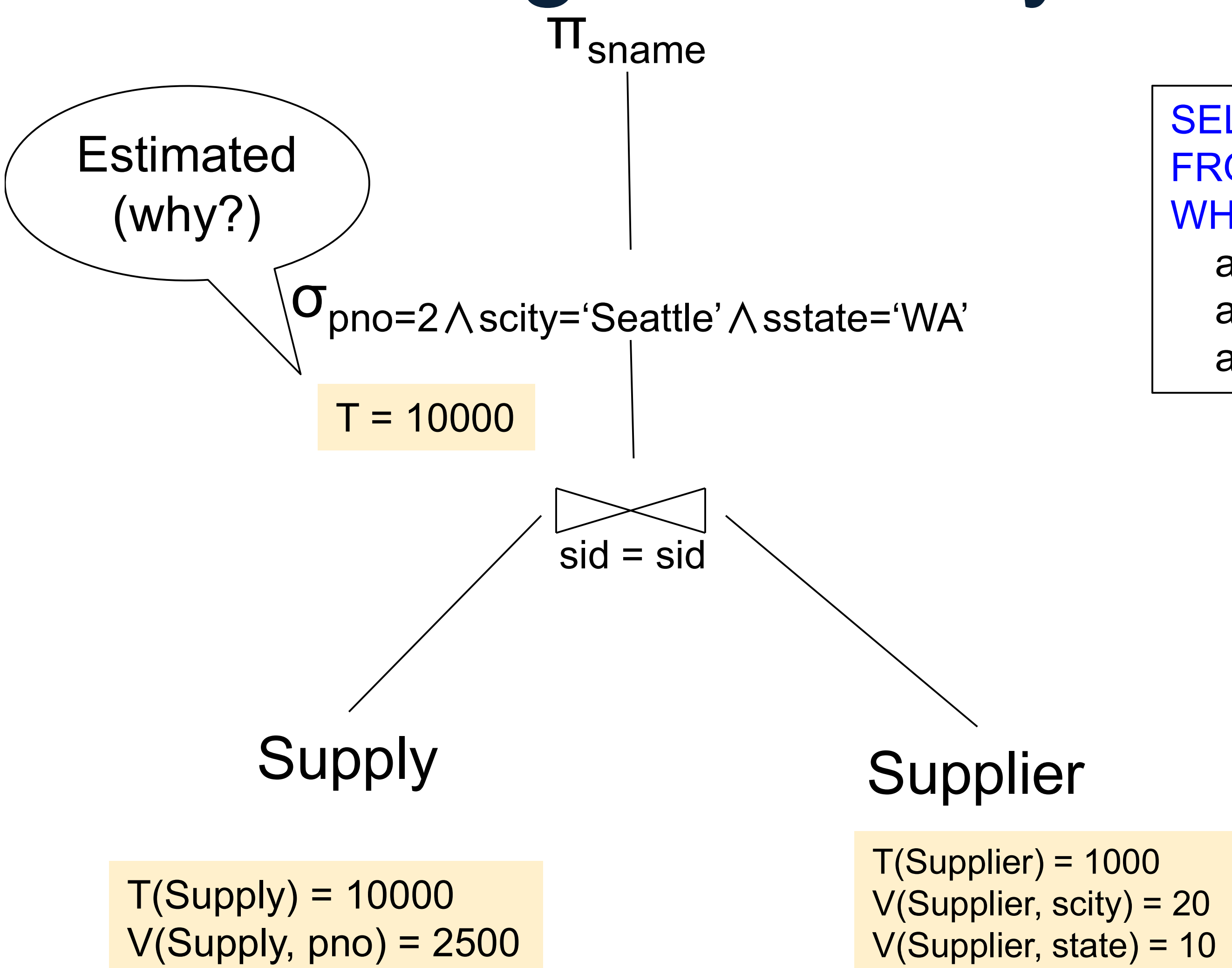
T(Supply) = 10000
V(Supply, pno) = 2500

T(Supplier) = 1000
V(Supplier, scity) = 20
V(Supplier, state) = 10

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

Logical Query Plan 1

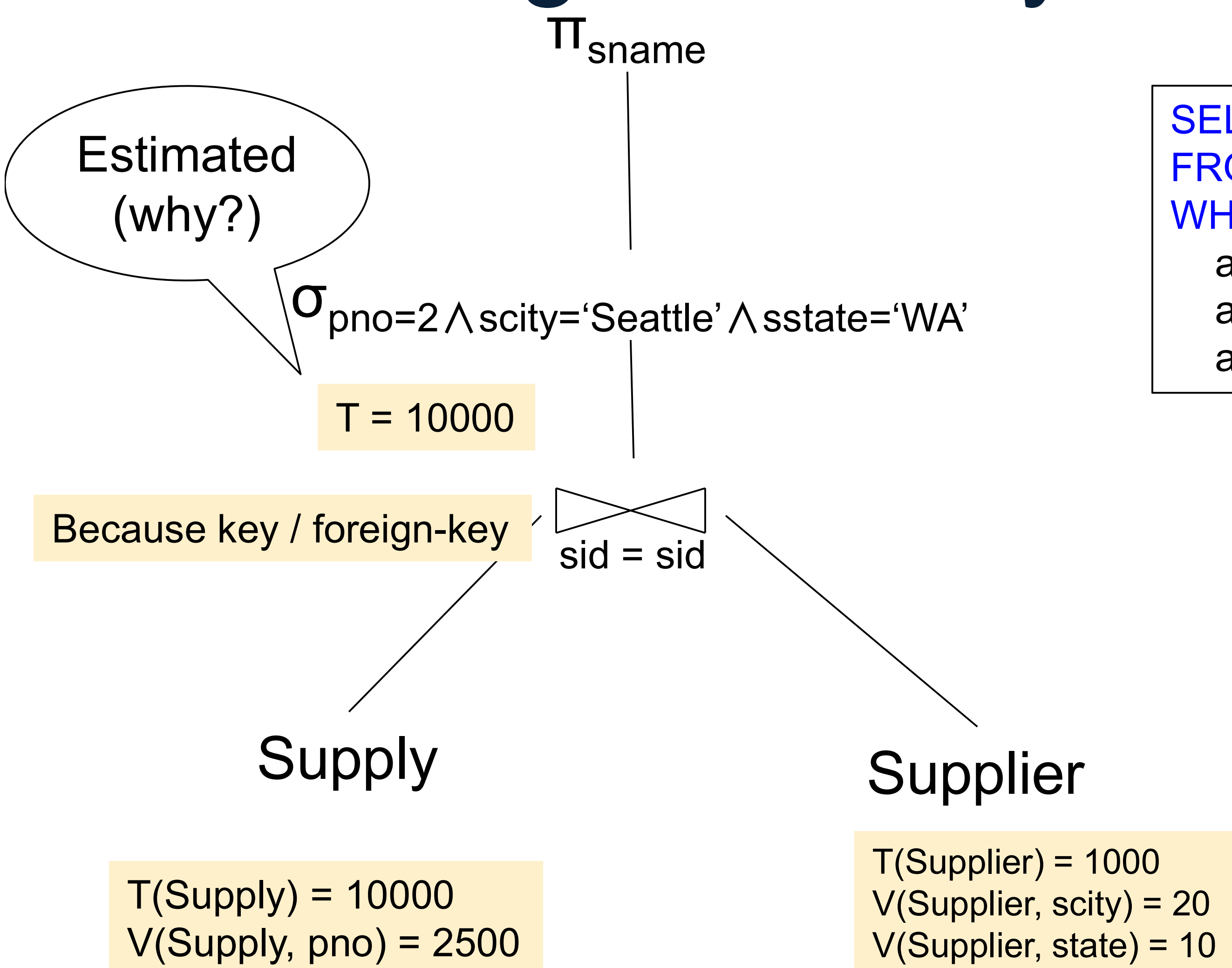
```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```



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

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

Logical Query Plan 1




```
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)

Logical Query Plan 1

π_{sname}

Estimated (why?)

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

T = 10000

$\bowtie_{sid = sid}$

Because key / foreign-key

Also: θ = 1/max(V(Supply,sid)*V(Supplier,sid) = 1/V(Supplier,sid)

Supply

T(Supply) = 10000
V(Supply, pno) = 2500

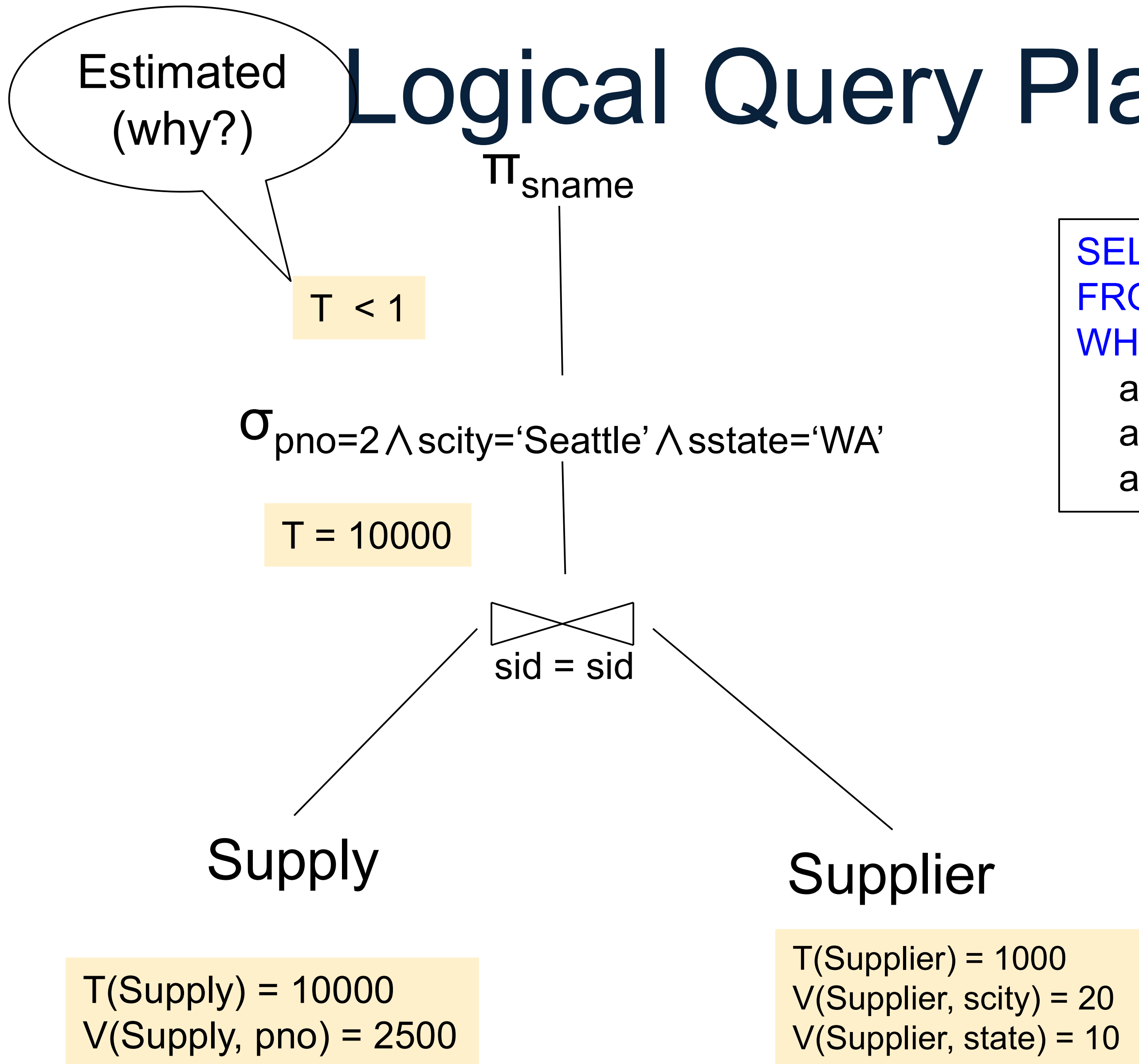
Supplier

T(Supplier) = 1000
V(Supplier, scity) = 20
V(Supplier, state) = 10

```
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)

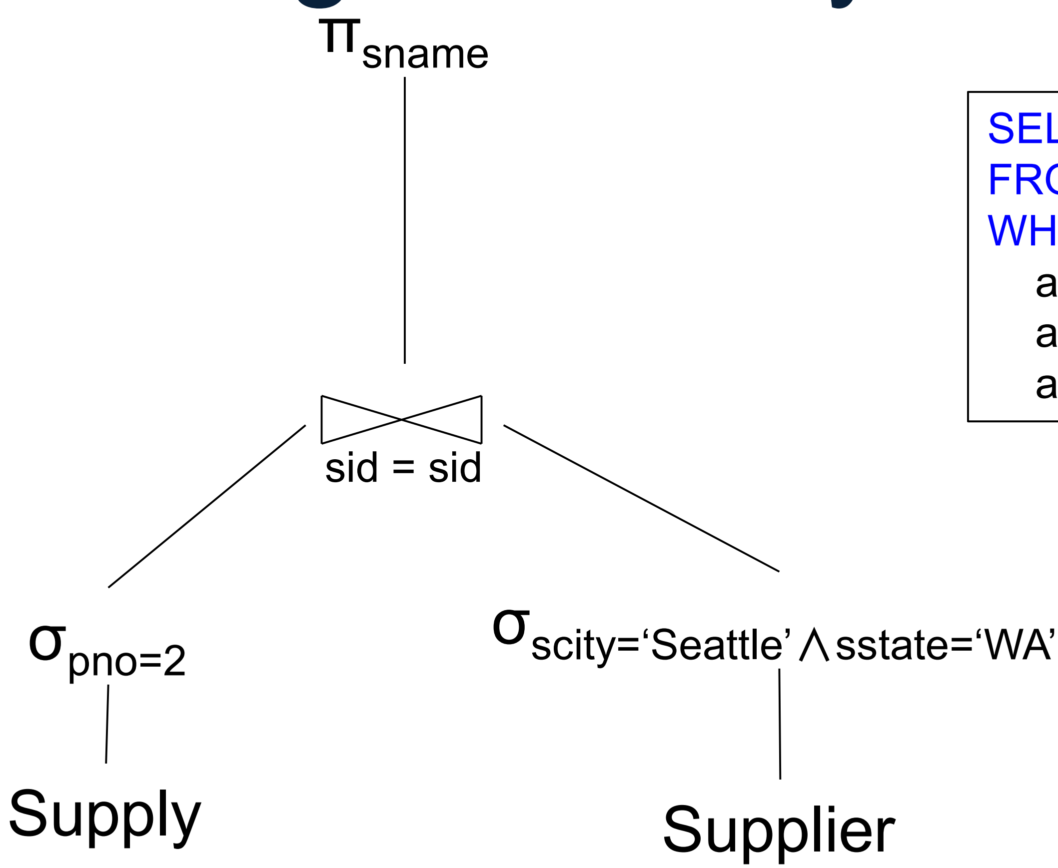
Logical Query Plan 1




```
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)

Logical Query Plan 2



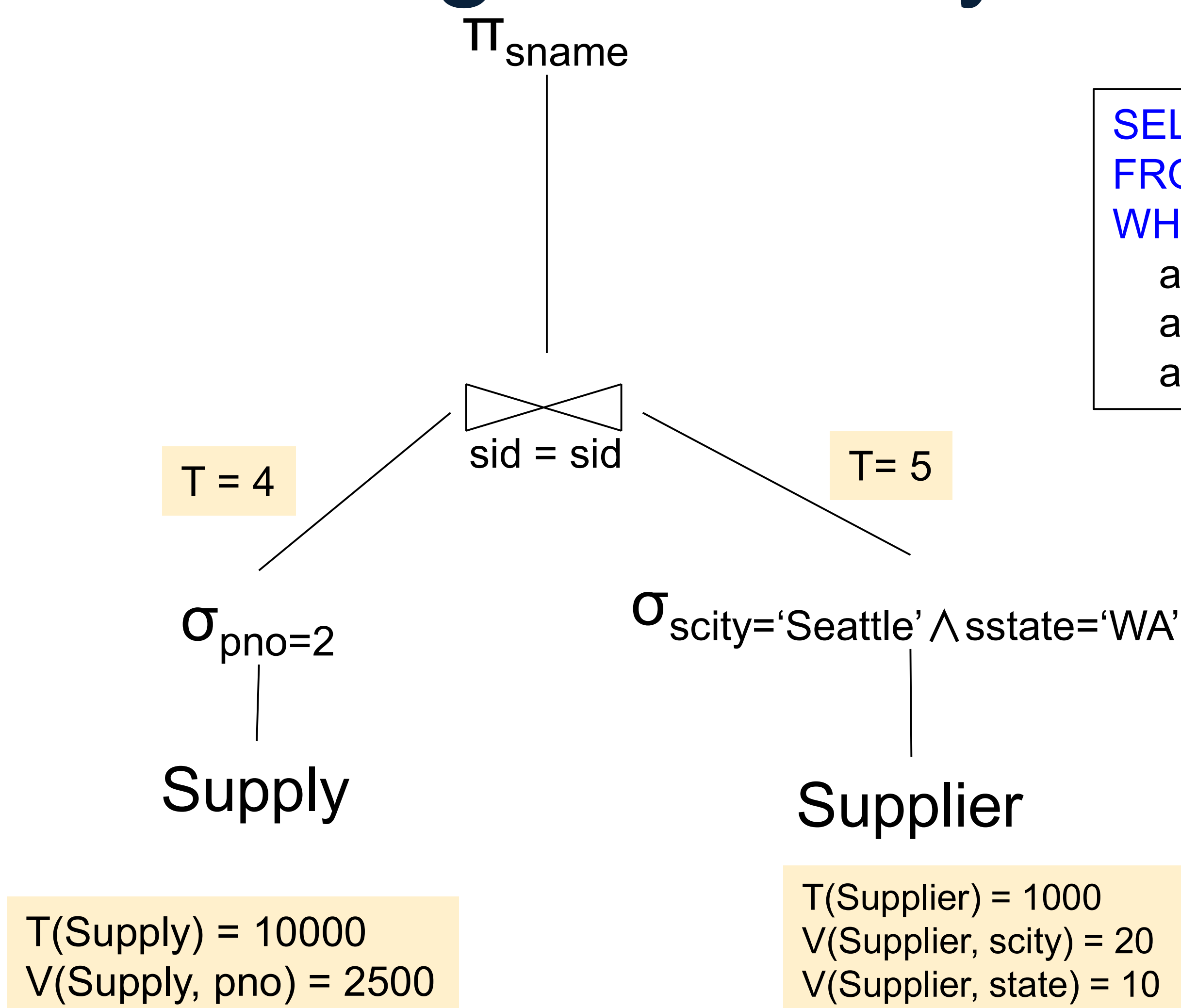

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```

T(Supply) = 10000
V(Supply, pno) = 2500

T(Supplier) = 1000
V(Supplier, scity) = 20
V(Supplier, state) = 10

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

Logical Query Plan 2

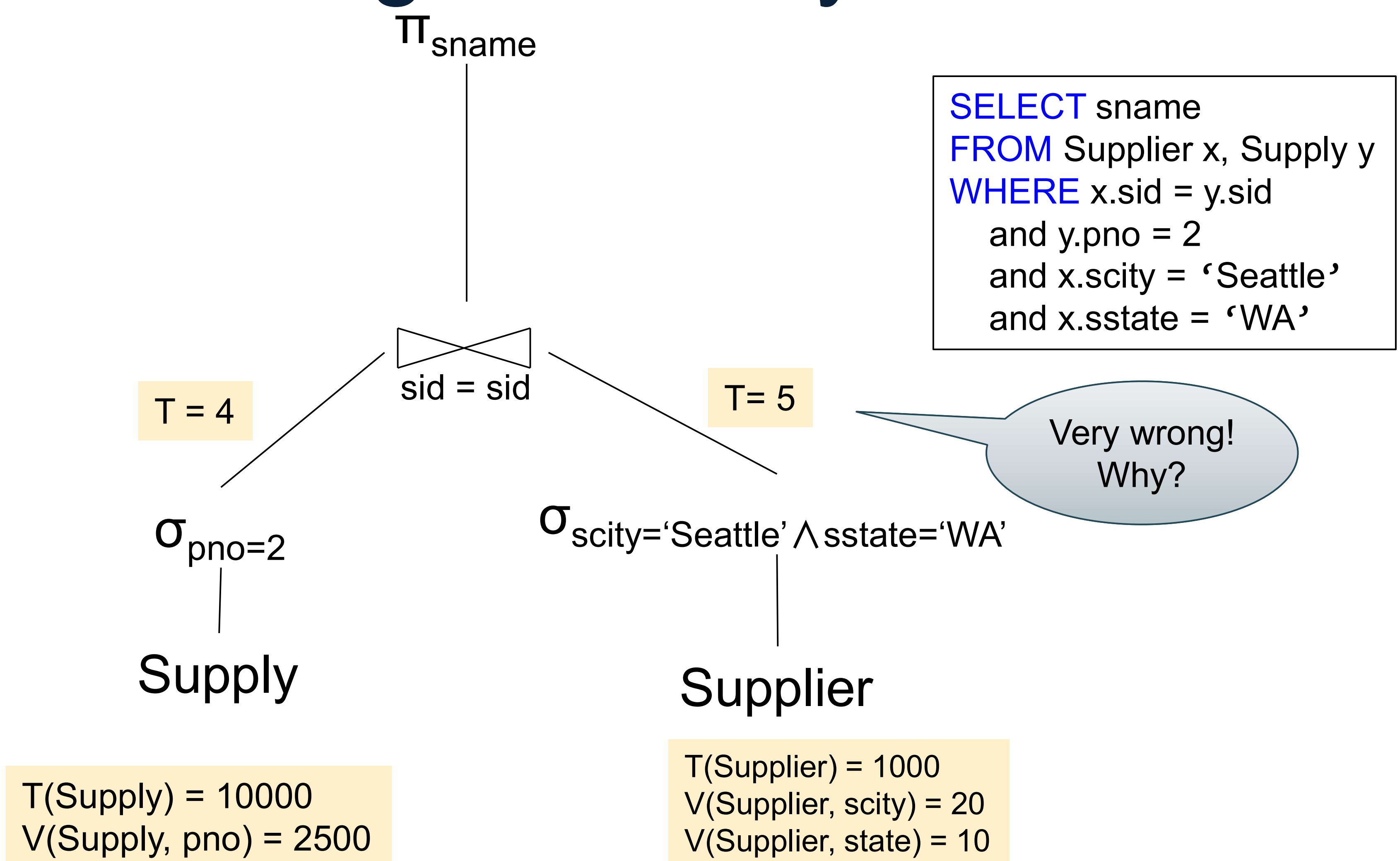



```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

Supplier(sid, sname, scity, sstate)

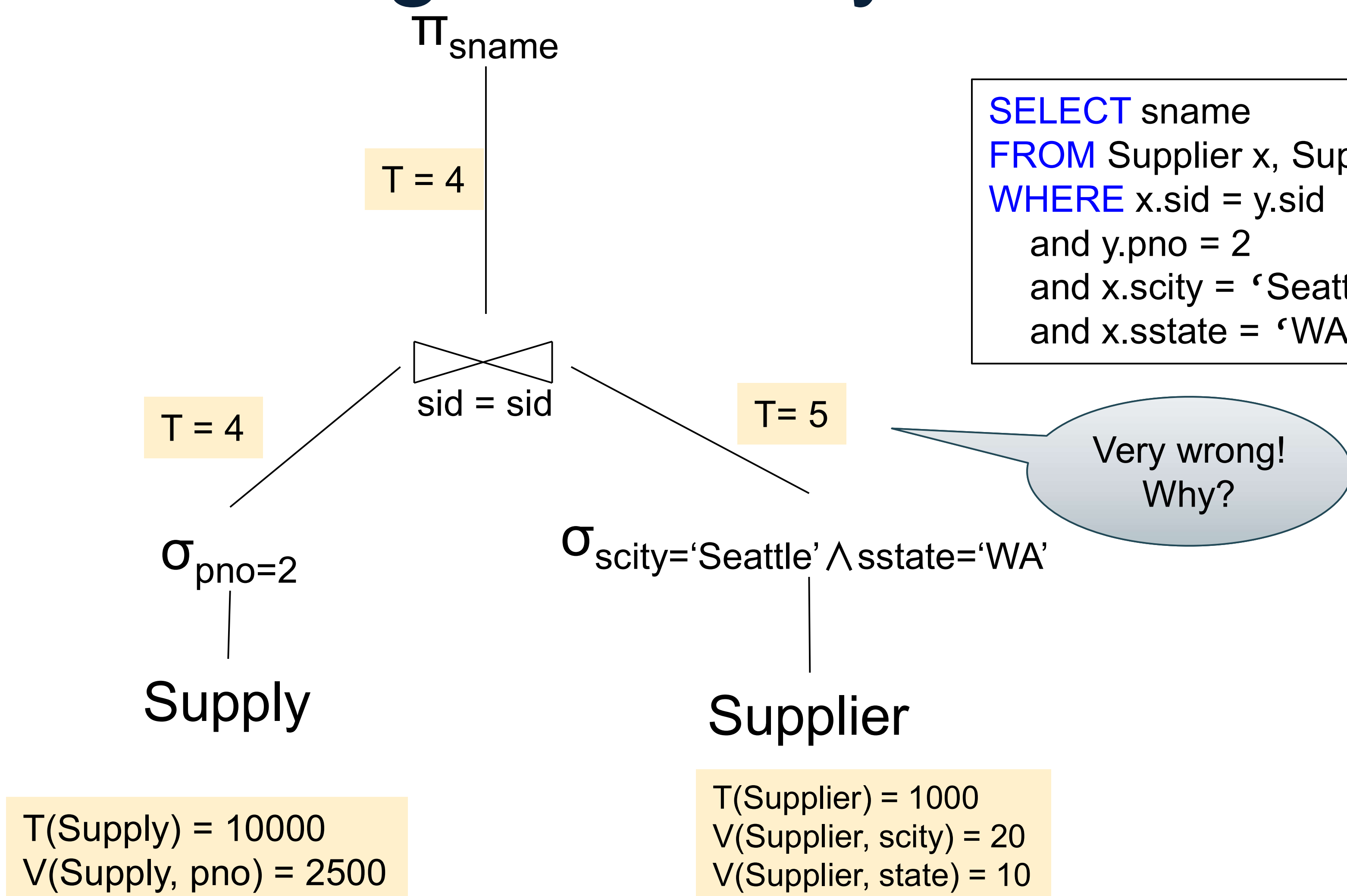
Supply(sid, pno, quantity)

Logical Query Plan 2



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

Logical Query Plan 2

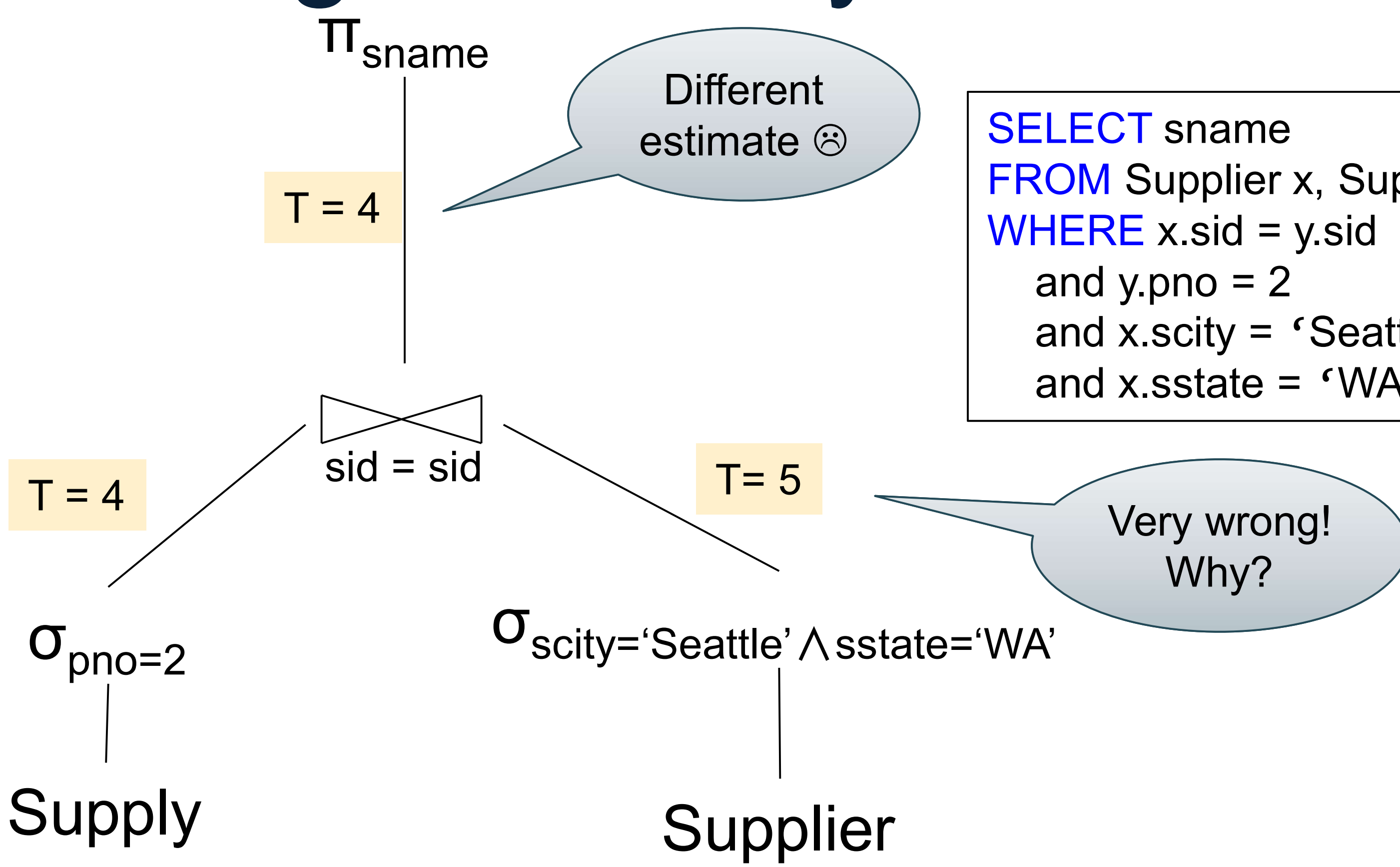



```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Logical Query Plan 2

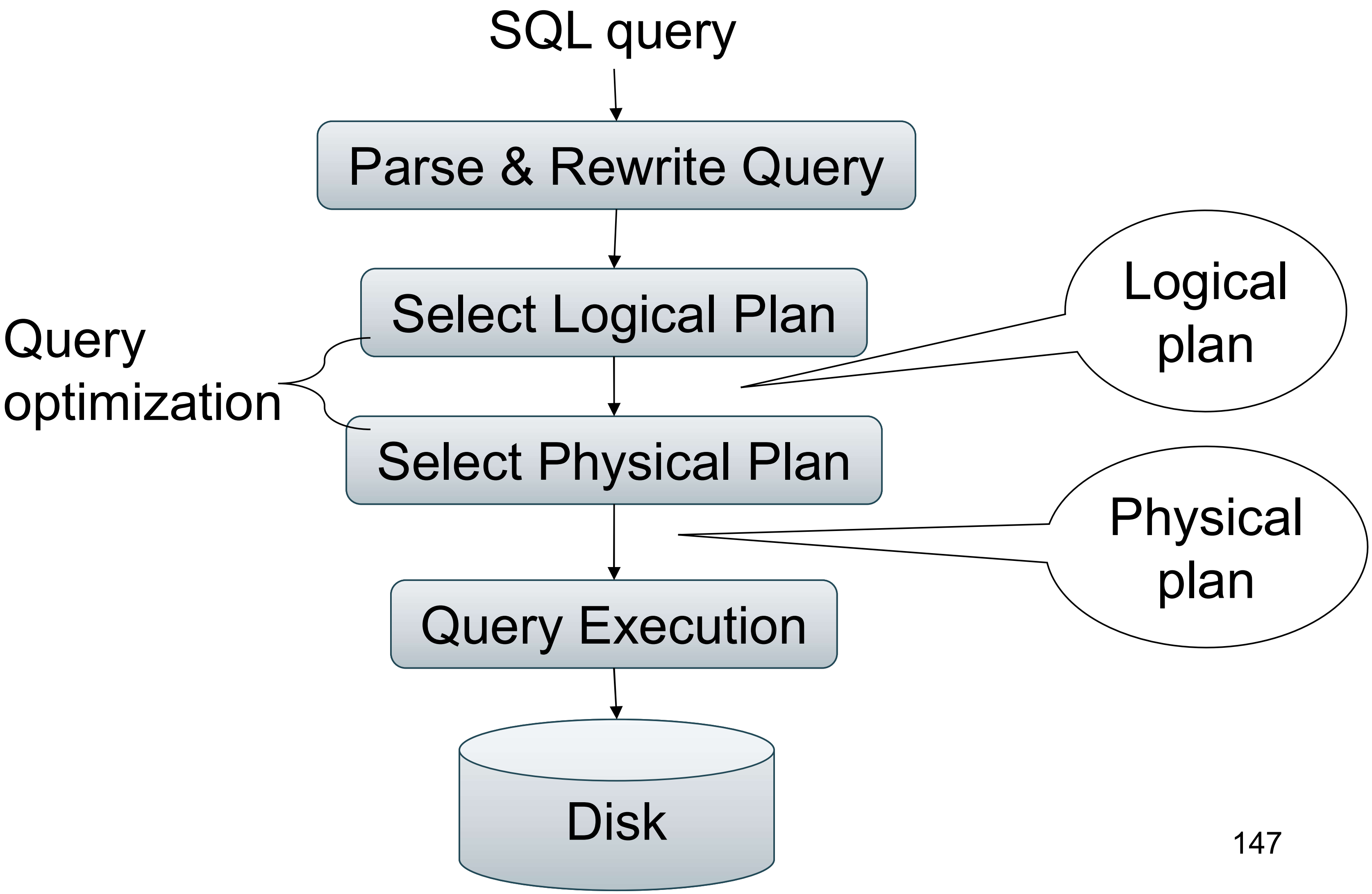



```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```

T(Supply) = 10000
V(Supply, pno) = 2500

T(Supplier) = 1000
V(Supplier, scity) = 20
V(Supplier, state) = 10

Lifecycle of a Query



Summary

- Optimizer has three components:
  - Search space
  - Cardinality and cost estimation
  - Plan enumeration algorithms (next time)
- Paper *How good are they* does a deep dive into modern optimizers
- Will continue optimizers next week