CSE544
Data Management

Lecture 11

Query Optimization – Part 1

Announcements

- Project proposal is due Friday
  - Submit via gitlab
  - If needed, update title on the spreadsheet
- Monday: Moe’s lecture on HW3
  - Don’t miss!!!
- Review 5 is due on ~~Monday~~ Wednesday
  - Advanced topic, you may skip parts

Snowflake: Recap

- SaaS = software as a service
- Other cloud services:
  - Platform as a service (PaaS): Amazon’s EC
  - Infrastructure as a service (virtual machines)
  - Software as a Service
  - Function as a Service: Amazon’s Lambda

Snowflake: Recap

Snowflake’s Data Storage:

In class:

- S3:PUT/GET/DELETE
- Table → horizontal partition in *files*
- Blobs+PAX
- Temp storage→S3

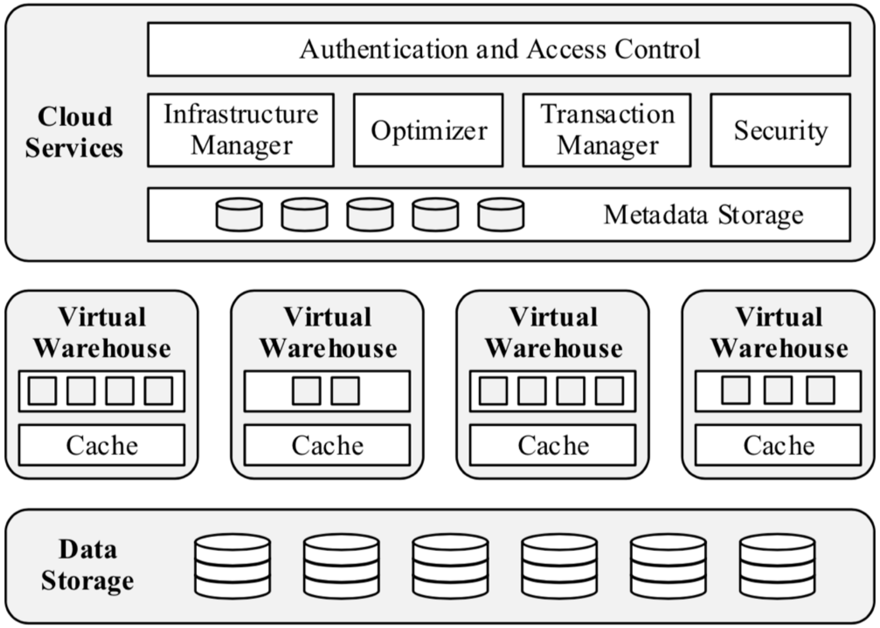


Figure 1: Multi-Cluster, Shared Data Architecture

Snowflake: Recap

- Describe Elasticity in Snowflake
  - Virtual Warehouse (VW) serves one user
  - T-Shirt sizes: X-Small … XX-Large
  - Small query may run on subset of VW
- Describe failure handling in Snowflake
  - Restart the query
  - No partial retries (like MapReduce or Spark)

Snowflake: Recap

- Column-oriented (in class)
- Vectorized (“tuple batches” – in class)
- Push-based (in class)

Snowflake: Recap

- What does Snowflake use instead of indexes?

- “Pruning”: for each file (recall: this is a horizontal partition of a table) and each attribute, it stores the min/max values in that column in that file; may skip files when not needed.

Query Optimization

Query Optimization

Three major components:

1. Search space

2. Cardinality and cost estimation

3. Plan enumeration algorithms

Search Space

Refers to the set of all possible alternate plans that the optimizers may explore

- Access Path Selection: how to use indices
- Rewrite Rules: what identities the optimizer applies
- Scheduling: pipeline or blocking

Database Statistics

- B(R) = # of blocks that store R
- T(R) = # of records in R
- V(R,A) = # of distinct values of R.A

Access Path

Access path: implements a selection $\sigma_P(R)$
(P is sometimes called *search argument* SARG)

- A file scan, or

- An index *plus* a matching selection condition

Access Path Selection

```
SELECT * FROM Supplier
WHERE sid > 300 ∧ scity='Seattle'
```

Indices:

B+-tree on sid; clustered

B+-tree on scity; unclustered

Access Path Selection

```
SELECT * FROM Supplier
WHERE sid > 300 ∧ scity='Seattle'
```

Indices:

B+-tree on sid; clustered
B+-tree on scity; unclustered

Which access path should we use?

Access Path Selection

```
SELECT * FROM Supplier
WHERE sid > 300 ∧ scity='Seattle'
```

Indices:

B+-tree on sid; clustered
B+-tree on scity; unclustered

Which access path should we use? Depends on stats

Access Path Selection

```
SELECT * FROM Supplier
WHERE sid > 300 ∧ scity='Seattle'
```

B(Supplier) = 100
T(Supplier) = 1000
V(Supplier,scity) = 20
Max(Supplier, sid) = 1000
Min(Supplier,sid) =1

Indices:

- B+-tree on sid; clustered
- B+-tree on scity; unclustered

Which access path should we use? Depends on stats

Supplier(sid, sname, scity, sstate)

Access Path Selecti

Number of distinct values of scity

```
SELECT * FROM Supplier
WHERE sid > 300 ∧ scity='Seattle'
```

B(Supplier)
T(Supplier) 1000
V(Supplier,scity) = 20
Max(Supplier, sid) = 1000
Min(Supplier,sid) =1

Indices:

- B+-tree on sid; clustered
- B+-tree on scity; unclustered

Which access path should we use? Depends on stats

Access Path Selection

```
SELECT * FROM Supplier
WHERE sid > 300 ∧ scity='Seattle'
```

B(Supplier) = 100
T(Supplier) = 1000
V(Supplier,scity) = 20
Max(Supplier, sid) = 1000
Min(Supplier,sid) =1

Indices:

- B+-tree on sid; clustered
- B+-tree on scity; unclustered

Which access path should we use? Depends on stats

Access Path Selection

```
SELECT * FROM Supplier
WHERE sid > 300 ∧ scity='Seattle'
```

B(Supplier) = 100
T(Supplier) = 1000
V(Supplier,scity) = 20
Max(Supplier, sid) = 1000
Min(Supplier,sid) =1

Indices:

B+-tree on sid; clustered
B+-tree on scity; unclustered

Which access path should we use? Depends on stats

1. Sequential scan: cost = 100

Access Path Selection

```
SELECT * FROM Supplier
WHERE sid > 300 ∧ scity='Seattle'
```

B(Supplier) = 100
T(Supplier) = 1000
V(Supplier,scity) = 20
Max(Supplier, sid) = 1000
Min(Supplier,sid) =1

Indices:

B+-tree on sid; clustered
B+-tree on scity; unclustered

Which access path should we use? Depends on stats

1. Sequential scan: cost = 100
2. Index scan on sid: cost = 7/10 * 100 = 70

Supplier(sid, sname, scity, sstate)

Access Path Selection

```
SELECT * FROM Supplier
WHERE sid > 300 ∧ scity='Seattle'
```

B(Supplier) = 100
T(Supplier) = 1000
V(Supplier,scity) = 20
Max(Supplier, sid) = 1000
Min(Supplier,sid) =1

Indices:

B+-tree on sid; clustered
B+-tree on scity; unclustered

Which access path should we use? Depends on stats

1. Sequential scan: cost = 100
2. Index scan on sid: cost = 7/10 * 100 = 70
3. Index scan on scity: cost = 1000/20 = 50

Access Path Selection

```
SELECT * FROM Supplier
WHERE sid > 300 ∧ scity='Seattle'
```

B(Supplier) = 100
T(Supplier) = 1000
V(Supplier,scity) = 20
Max(Supplier, sid) = 1000
Min(Supplier,sid) =1

Indices:

B+-tree on sid; clustered
B+-tree on scity; unclustered

Which access path should we use? Depends on stats

1. Sequential scan: cost = 100
2. Index scan on sid: cost = 7/10 * 100 = 70
3. Index scan on scity: cost = 1000/20 = 50

Optimal

Rewrite Rules

Search space is defined by the set of rewrite rules that the optimizer implements

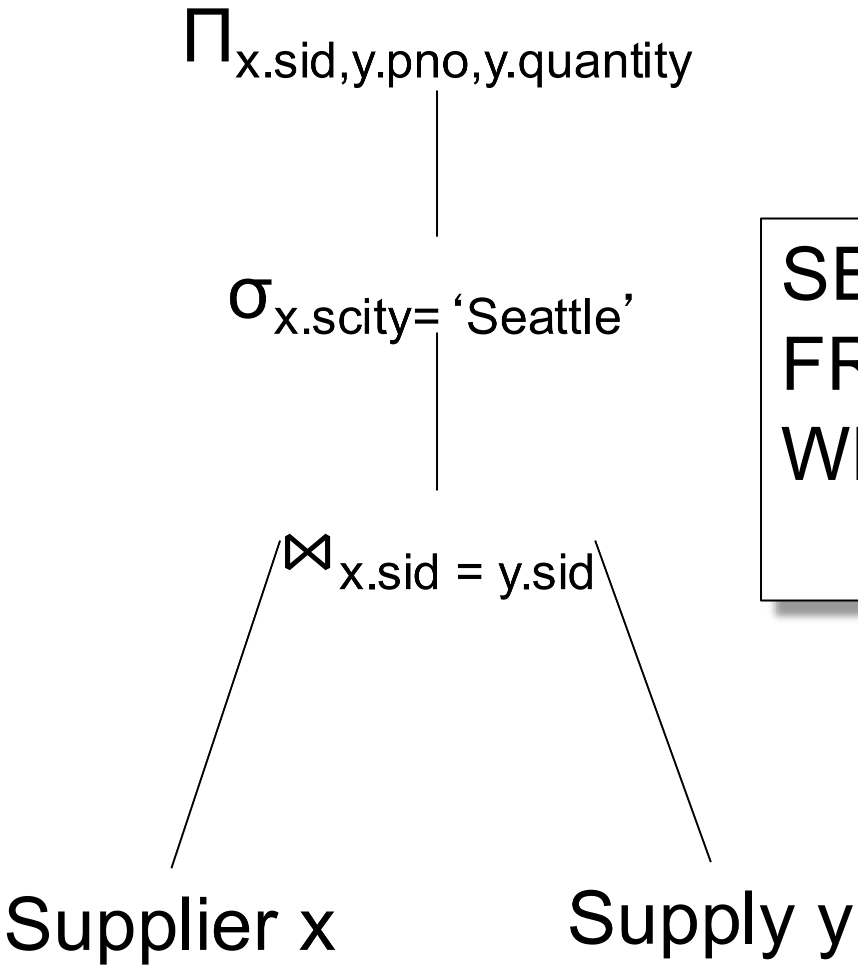
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

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)

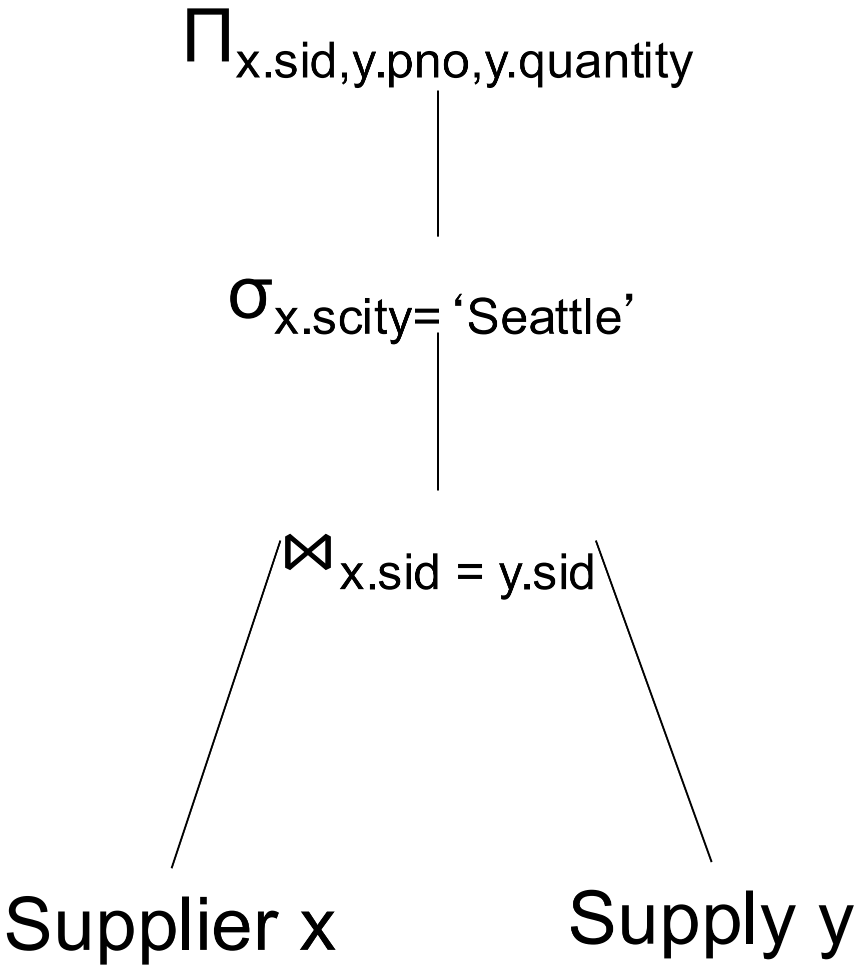
Example Optimization




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

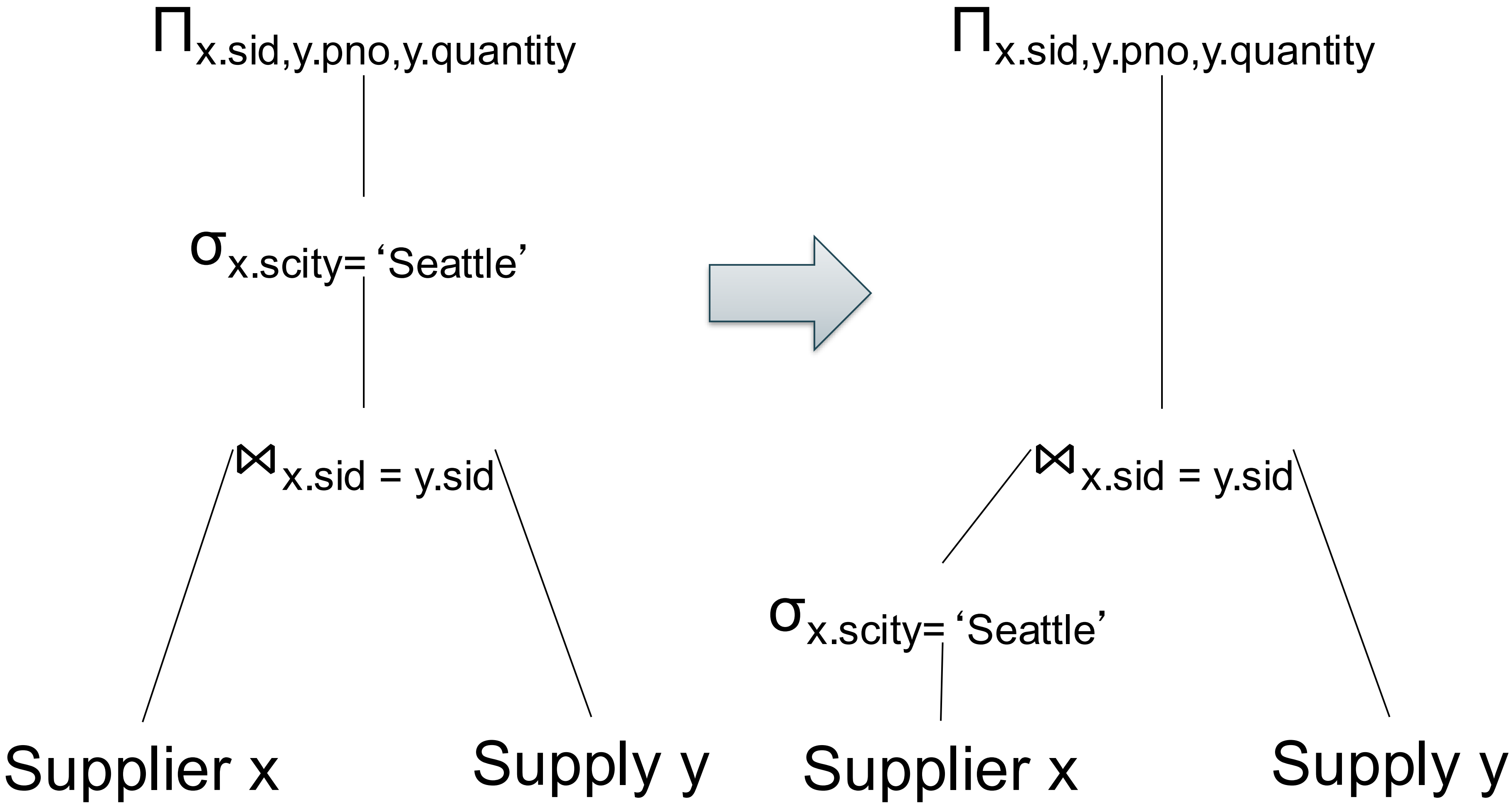
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



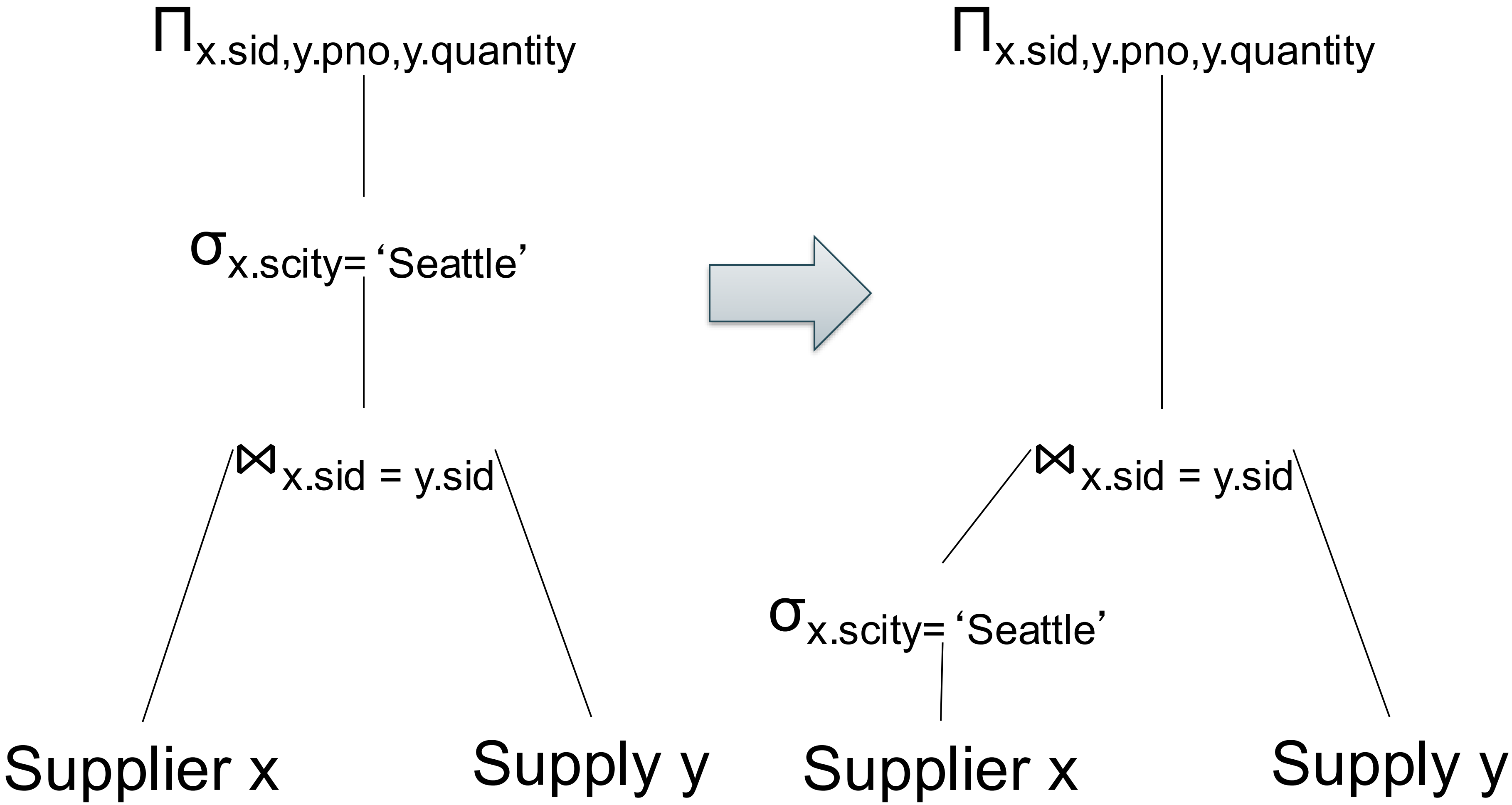
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



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

$\Pi_{x.sid,y.pno,y.quantity}$

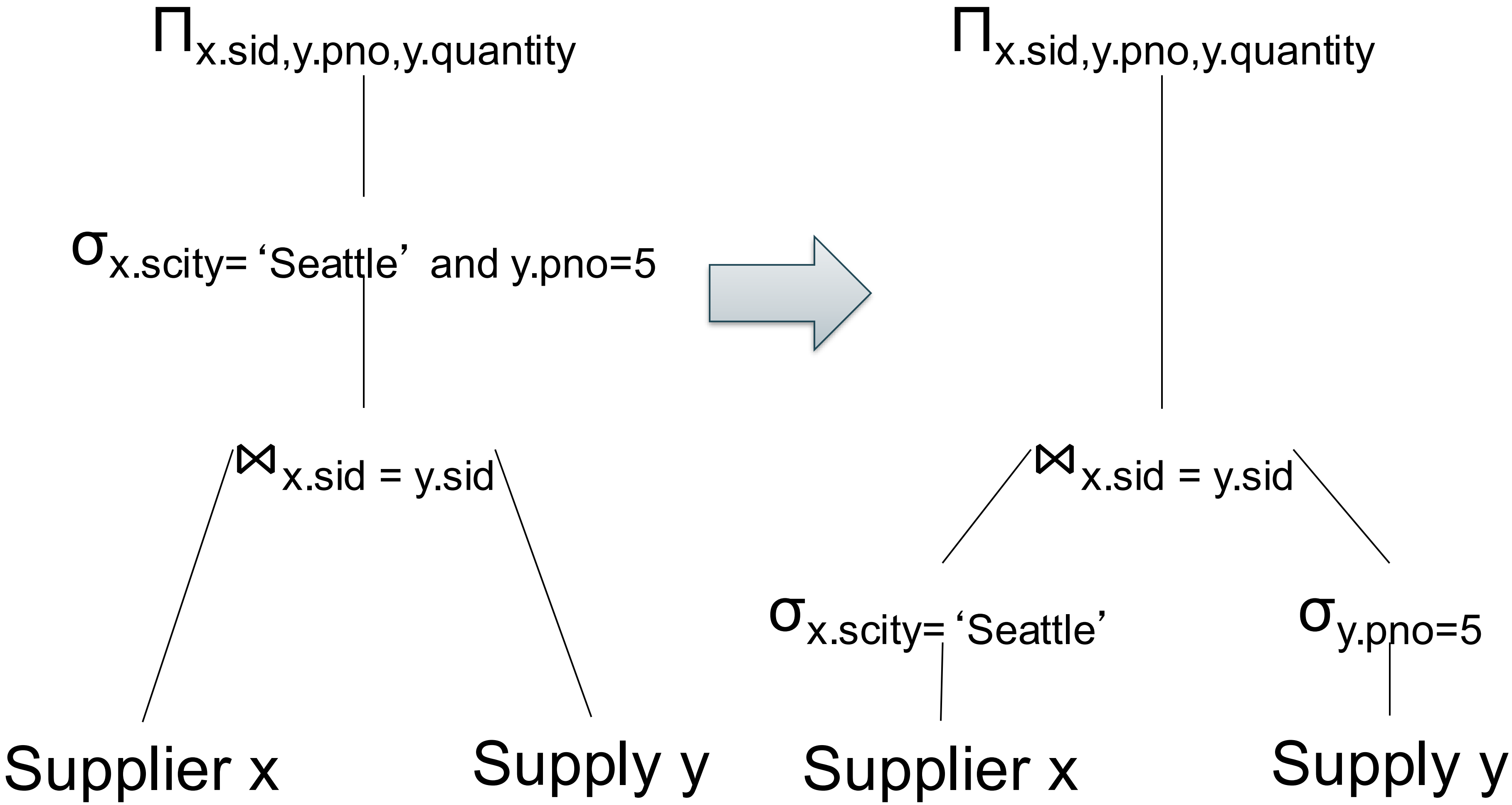
$\sigma_{x.scity=\text{'Seattle'} \text{ and } y.pno=5}$

$\bowtie_{x.sid = y.sid}$

Supplier x   Supply y

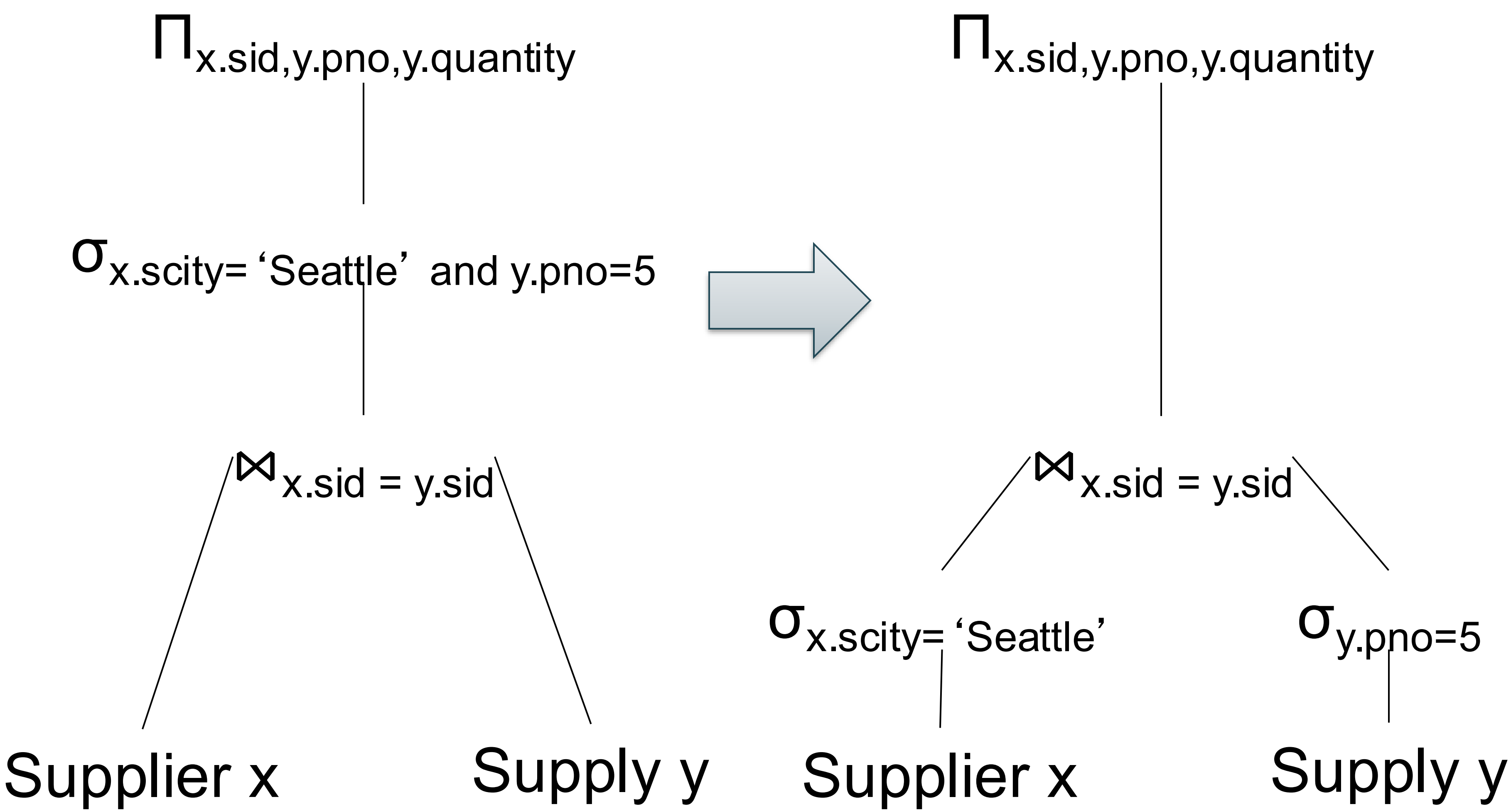
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



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

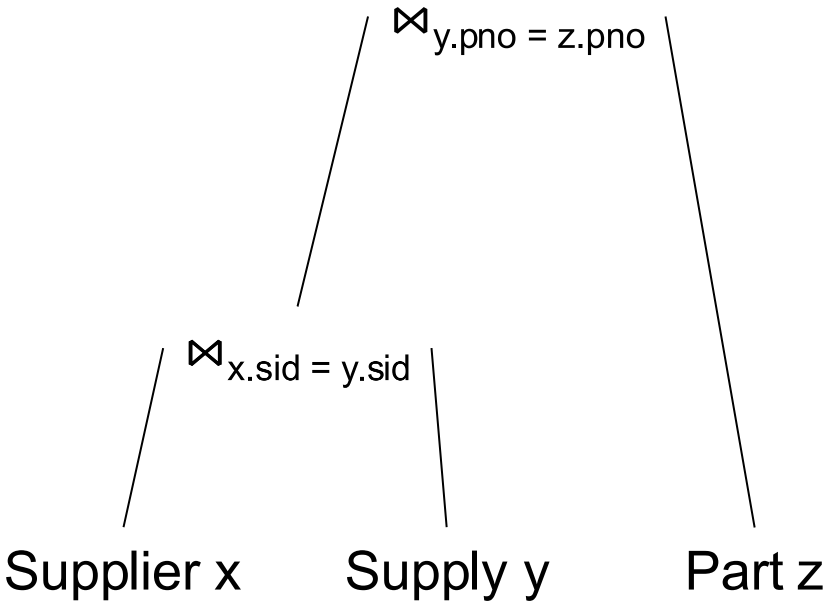
Push Selections Down



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

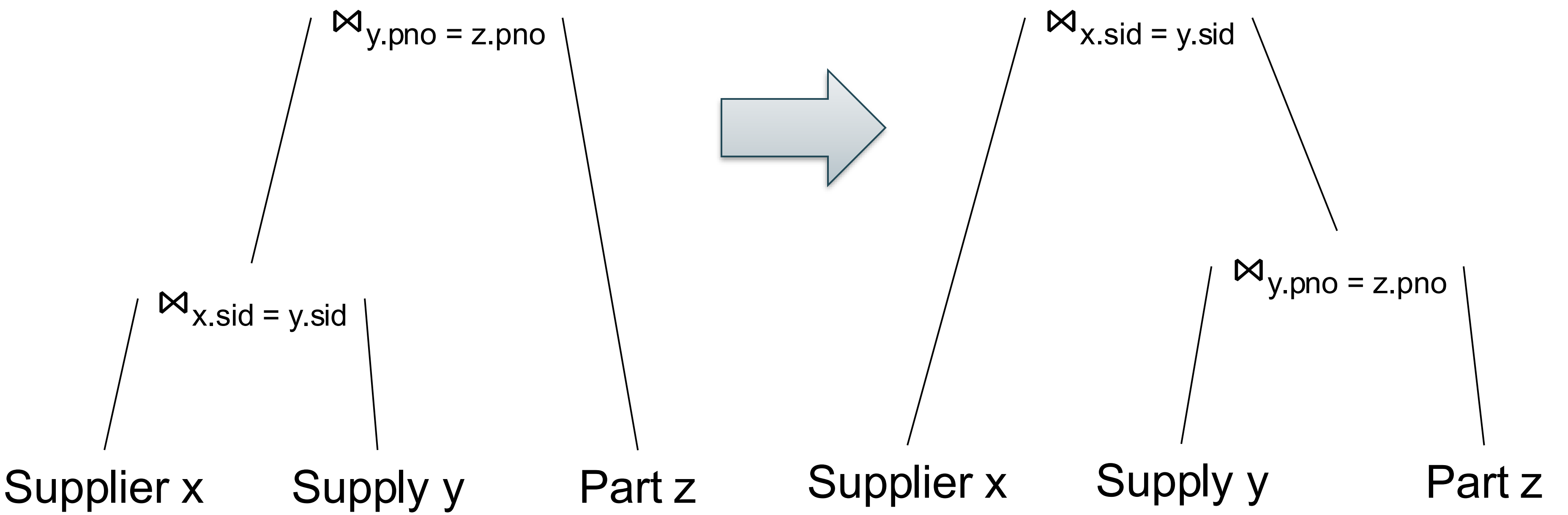
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder



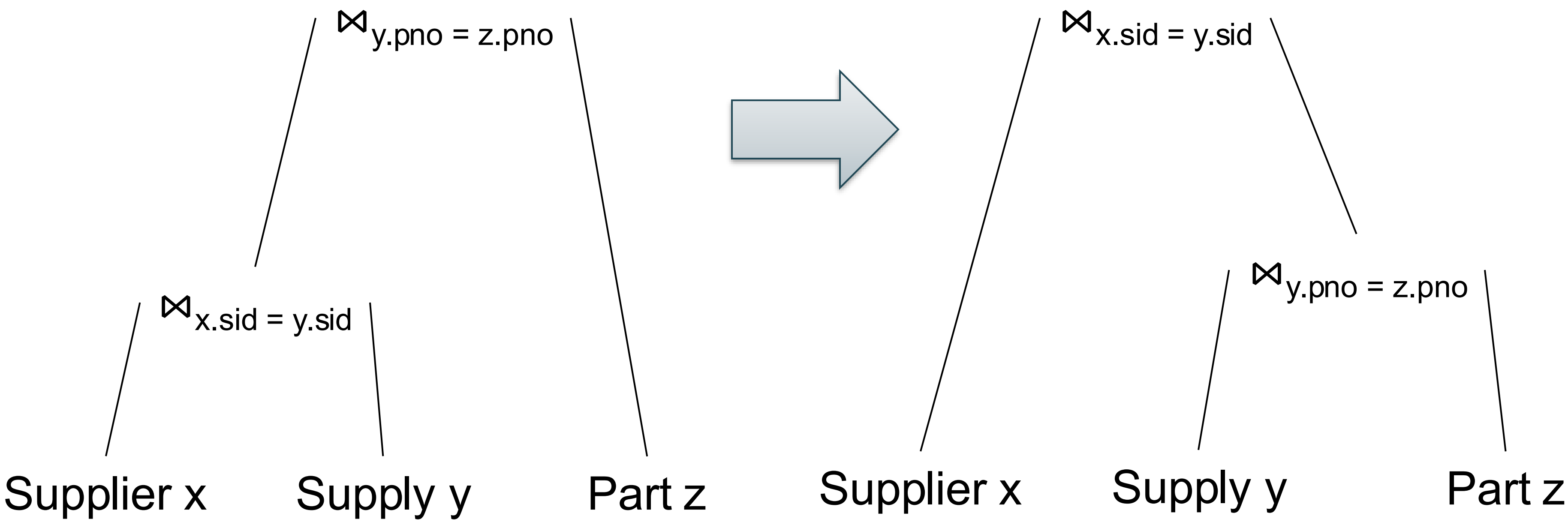
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder



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

Join Reorder



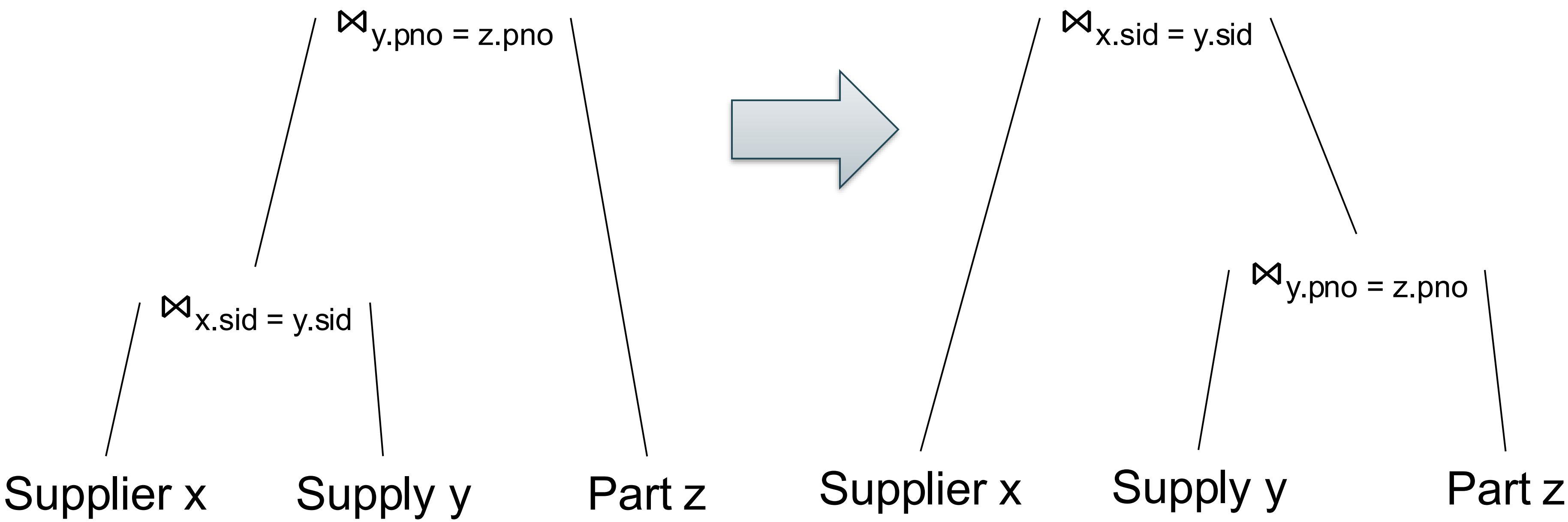
$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

$$R \bowtie S = S \bowtie R$$

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

Join Reorder

Which plan is better?



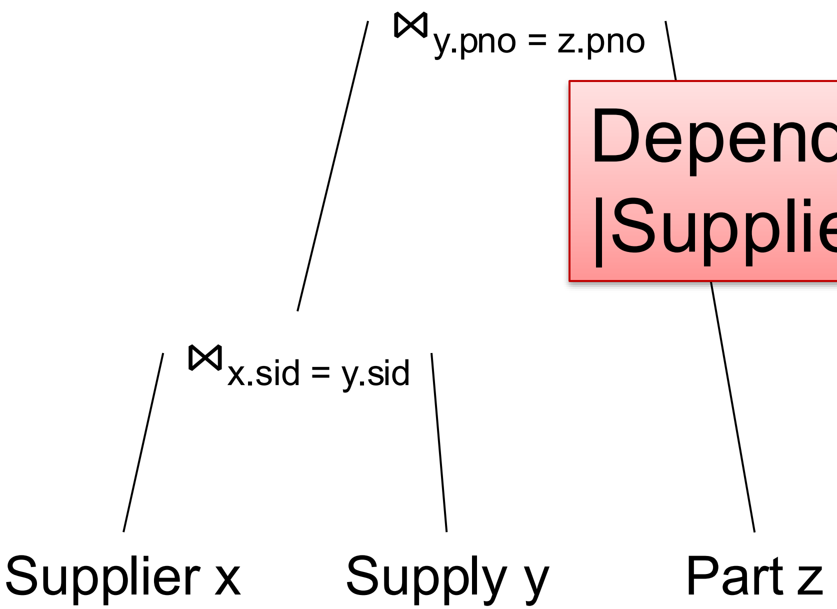
$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

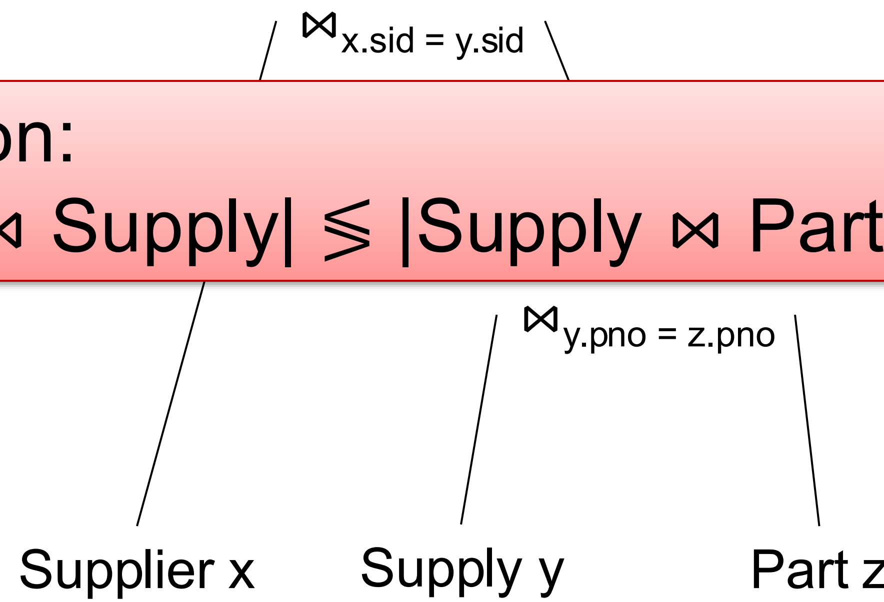
$$R \bowtie S = S \bowtie R$$

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

Join Reorder

Which plan is better?





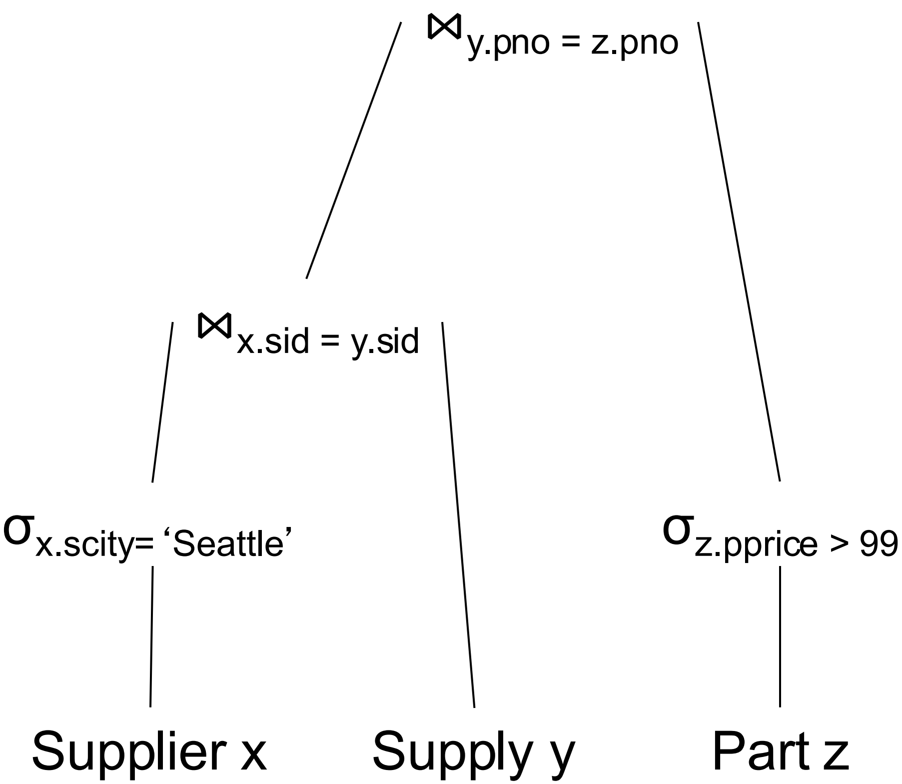
Depends on:
$|\text{Supplier} \bowtie \text{Supply}| \lesseqgtr |\text{Supply} \bowtie \text{Part}|$

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

$R \bowtie S = S \bowtie R$

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

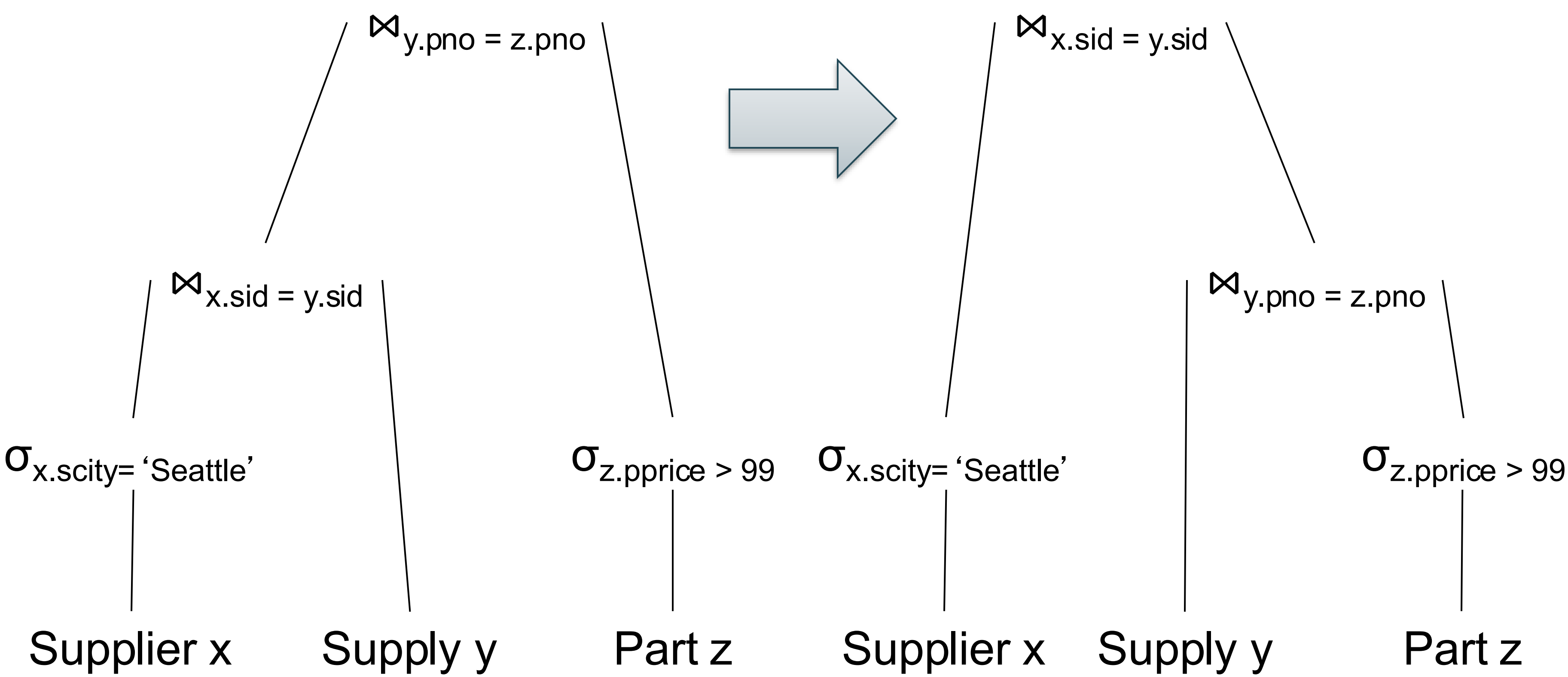
Join Reorder



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

Join Reorder

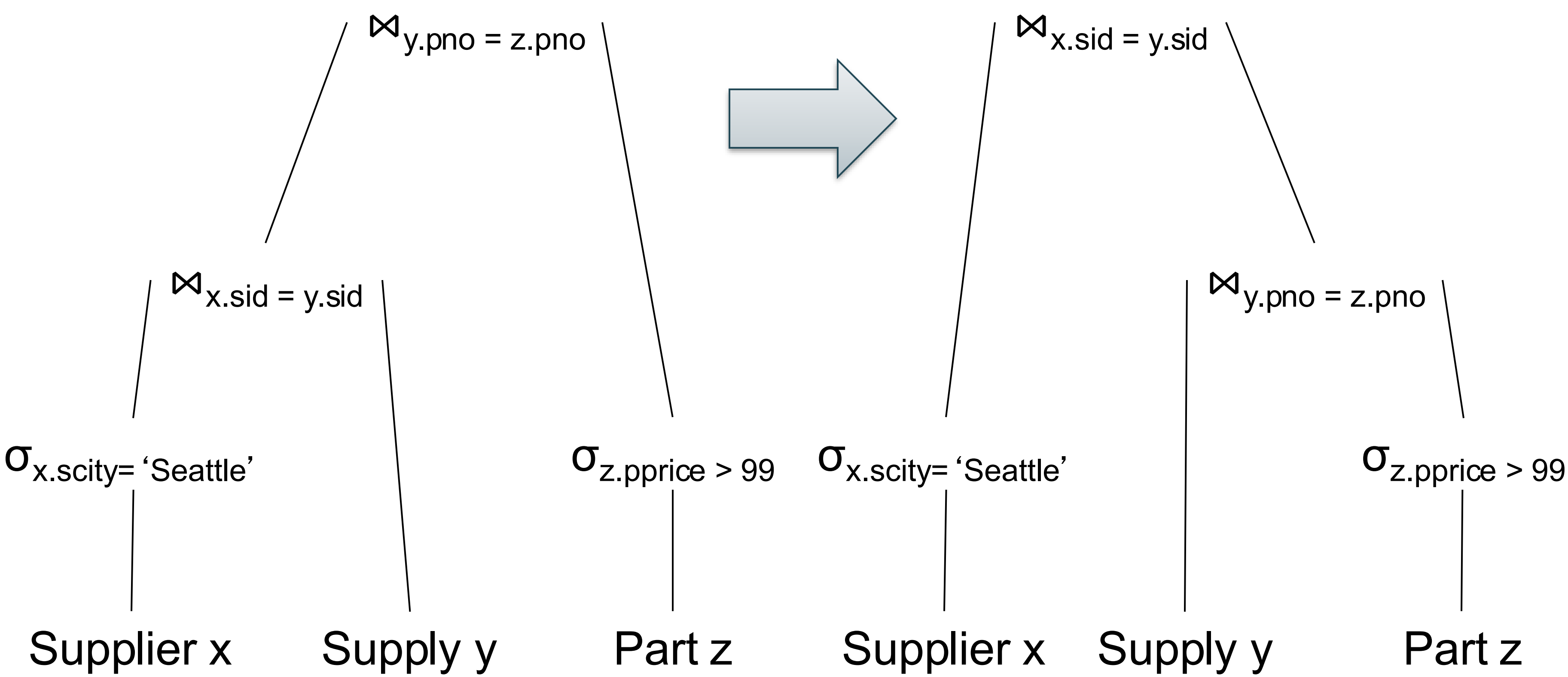
Which plan is better?



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

Join Reorder

Which plan is better?



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

Discussion

- Both rewrite rules **and** cardinality estimations are needed for optimization
- They can be developed independently

- Next: a powerful optimization rule: aggregate pushdown or better known as generalized distributivity law

Motivation

- Try this in postgres

```
select count(*) from author;
```

Answer: 2652053
Time: 0.058 s

Motivation

- Try this in postgres

```
select count(*) from author;
```

Answer: 2652053
Time: 0.058 s

```
select count(*) from publication;
```

Answer: 5120896
Time: 0.062 s

Motivation

- Try this in postgres

```
select count(*) from author;
```

Answer: 2652053
Time: 0.058 s

```
select count(*) from publication;
```

Answer: 5120896
Time: 0.062 s

```
select count(*) from author, publication;
```

Timeout!!!

Motivation

- Try this in postgres

```
select count(*) from author;
```

Answer: 2652053
Time: 0.058 s

```
select count(*) from publication;
```

Answer: 5120896
Time: 0.062 s

```
select count(*) from author, publication;
```

Timeout!!!

But 3rd query is simply the **product** of the first two!

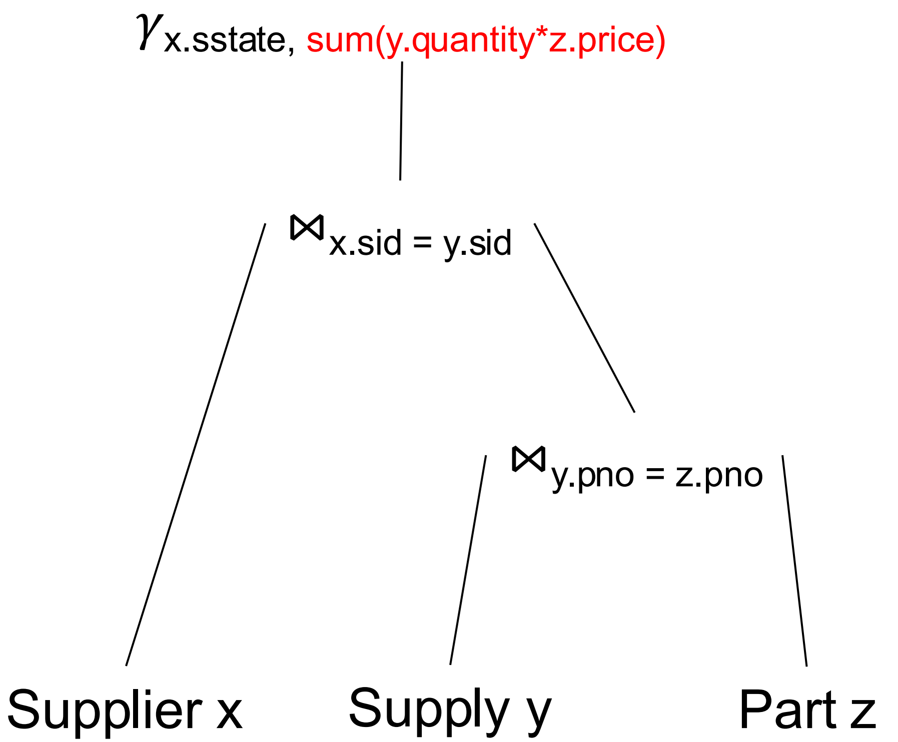
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Aggregate Push-down

```
SELECT x.sstate, sum(y.quanity*z.price)
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
GROUP BY x.sstate
```

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

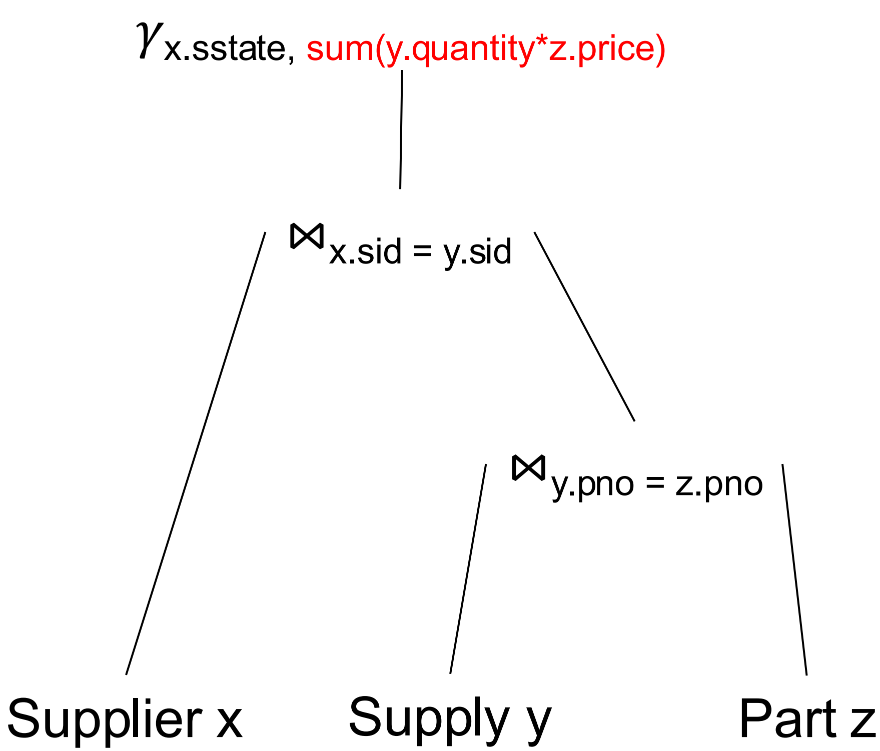
Aggregate Push-down

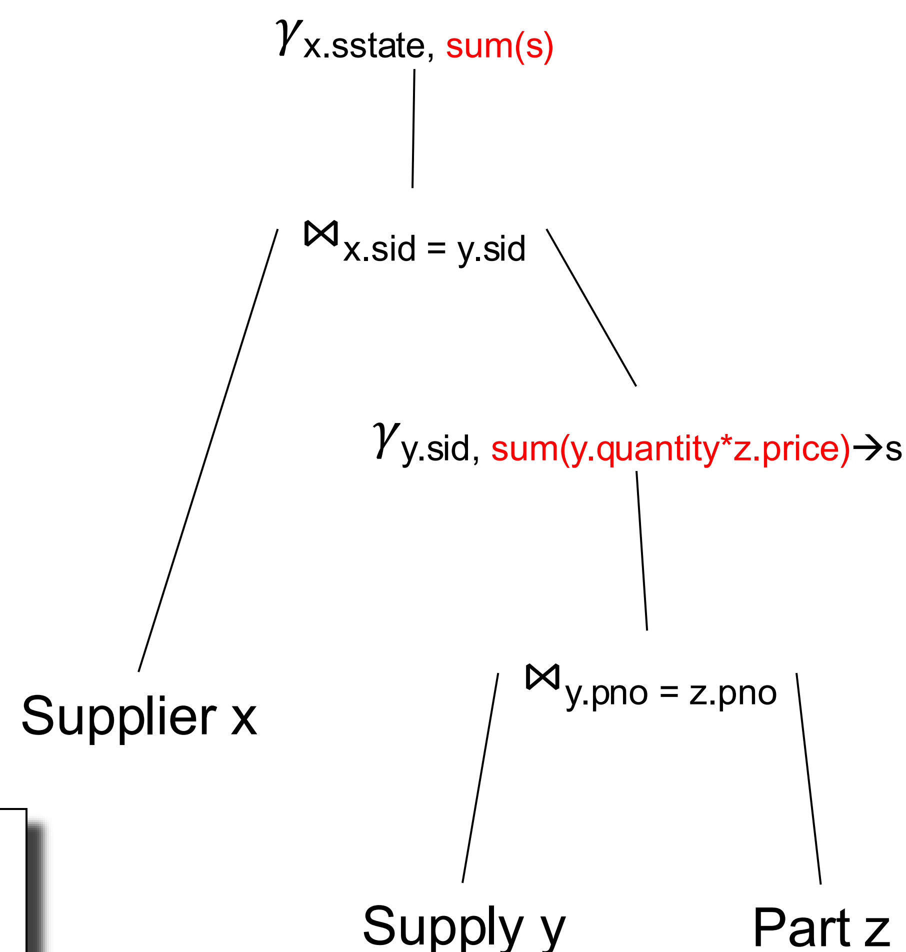



```
SELECT x.sstate, sum(y.quanity*z.price)
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
GROUP BY x.sstate
```

Aggregate Push-down

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

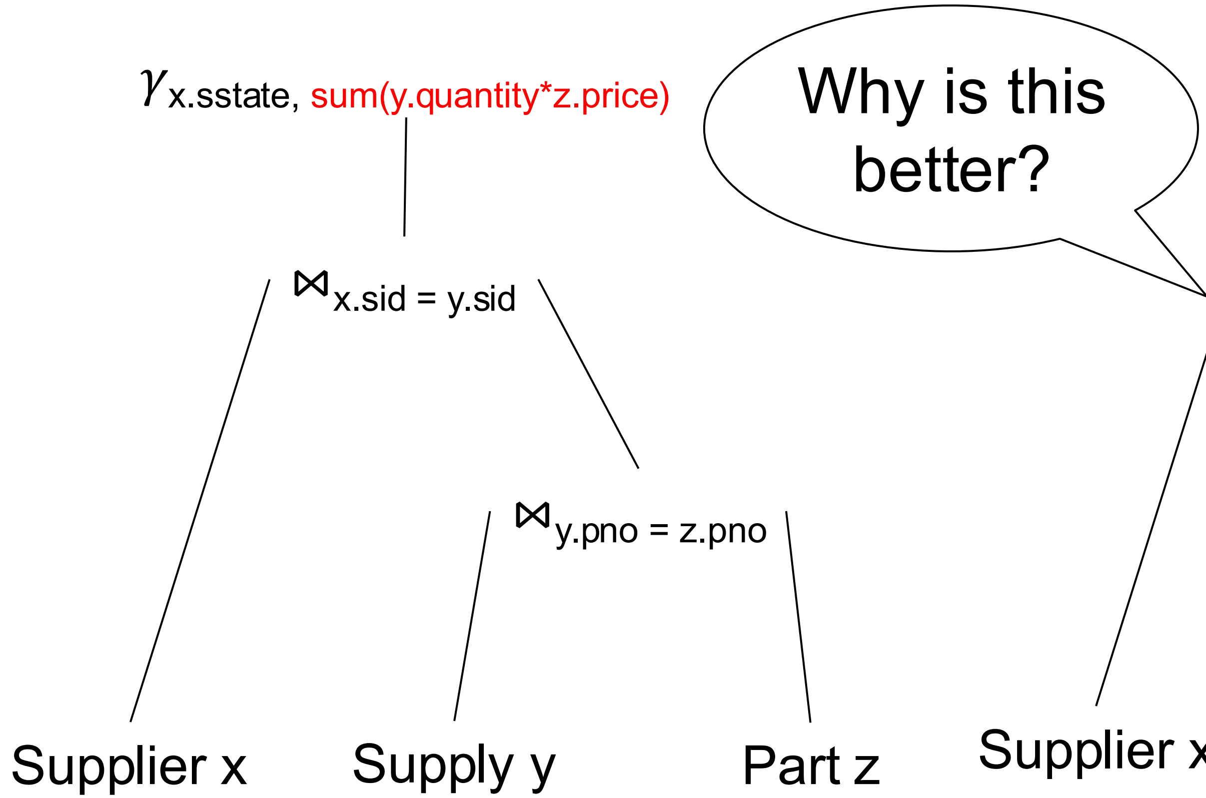


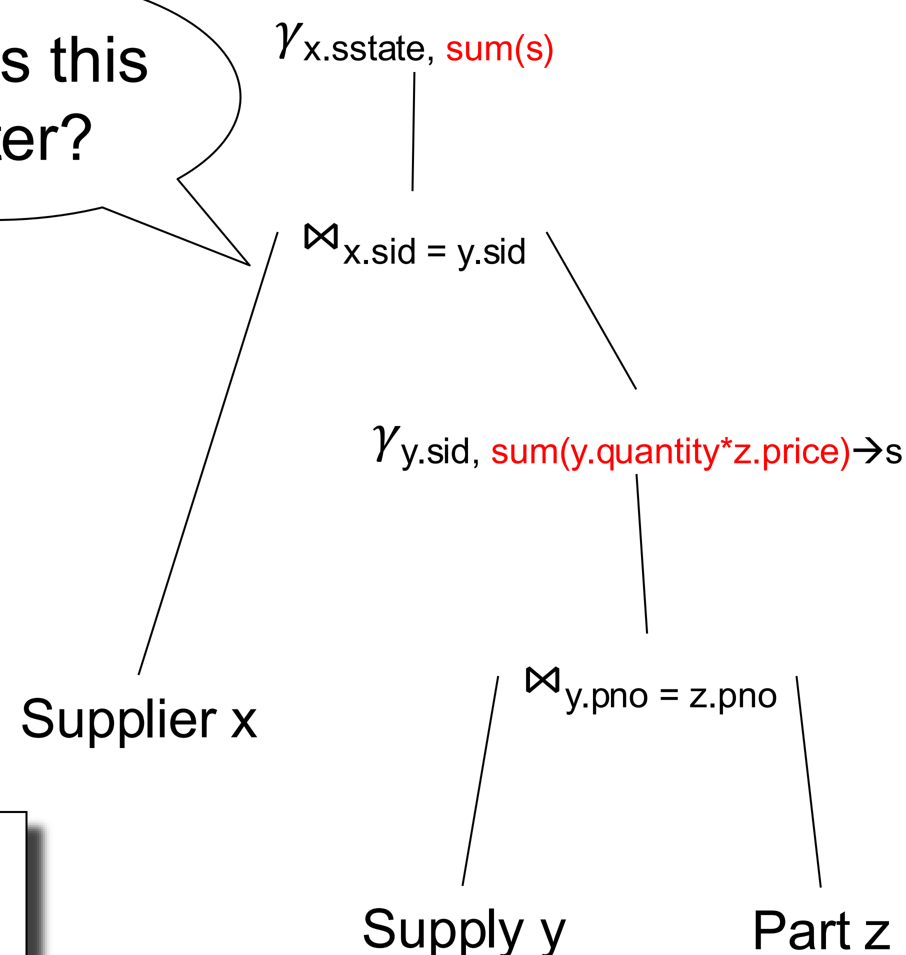



```
SELECT x.sstate, sum(y.quanity*z.price)
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
GROUP BY x.sstate
```

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

Aggregate Push-down

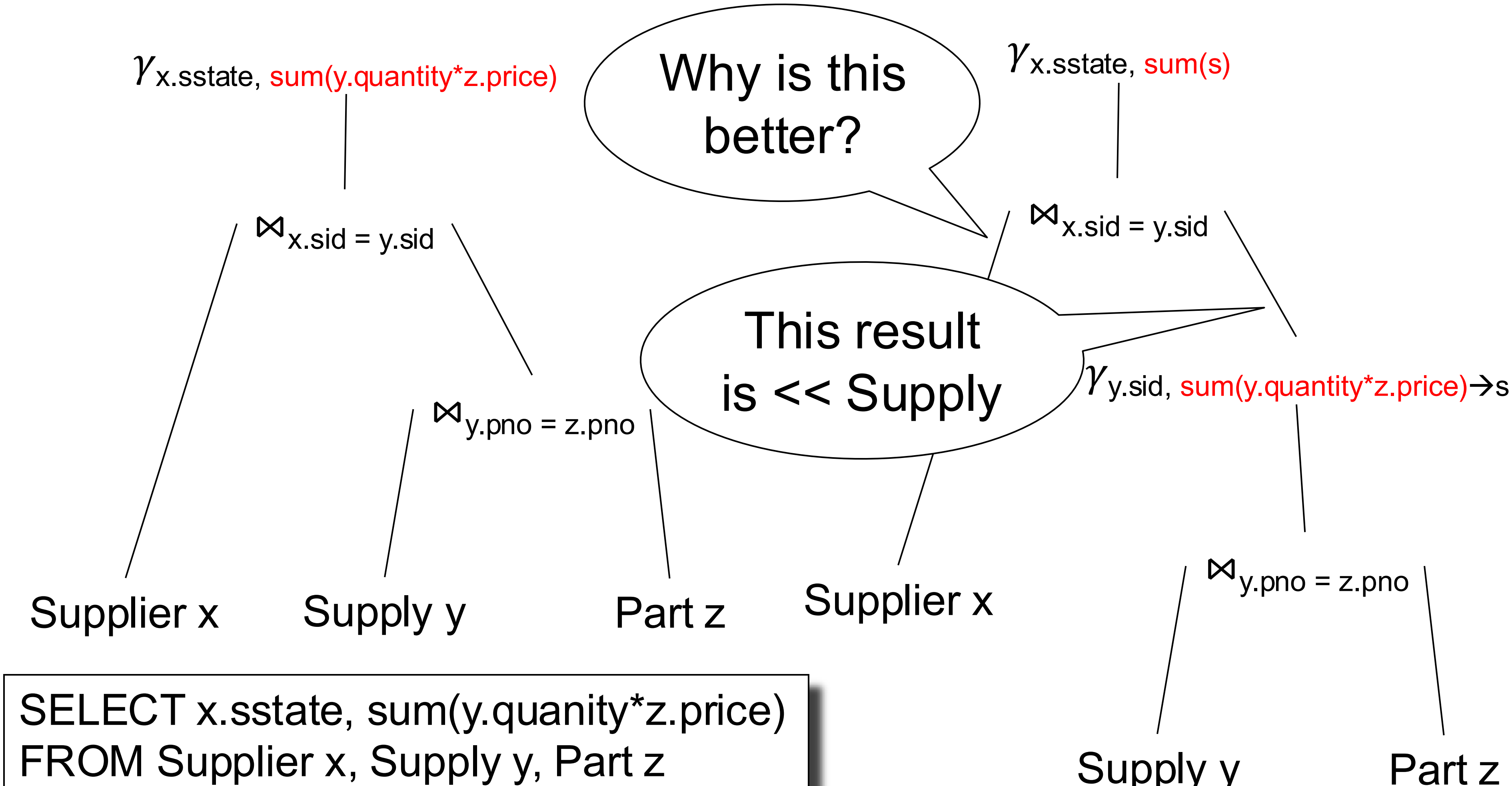





```
SELECT x.sstate, sum(y.quanity*z.price)
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
GROUP BY x.sstate
```

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

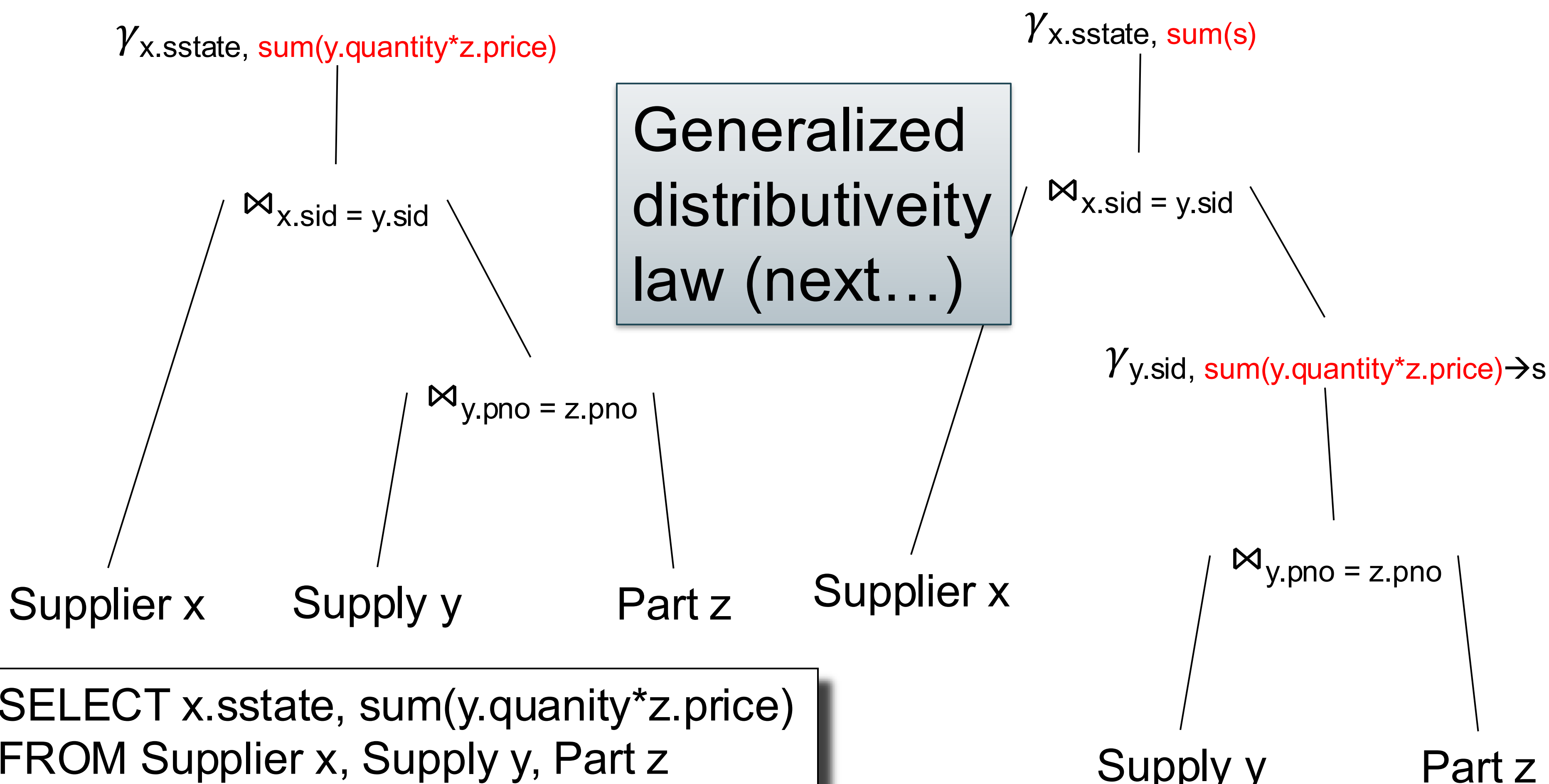
Aggregate Push-down




```
SELECT x.sstate, sum(y.quanity*z.price)
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
GROUP BY x.sstate
```

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

Aggregate Push-down

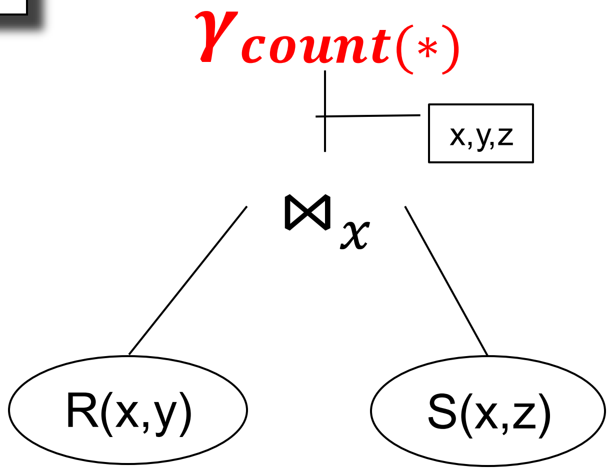


Generalized distributiveity law (next…)

```
SELECT x.sstate, sum(y.quanity*z.price)
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
GROUP BY x.sstate
```

Generalized Distributivity Law

SELECT count(*) from R, S where R.x=S.x



Generalized Distributivity Law

SELECT count(*) from R, S where R.x=S.x

R:

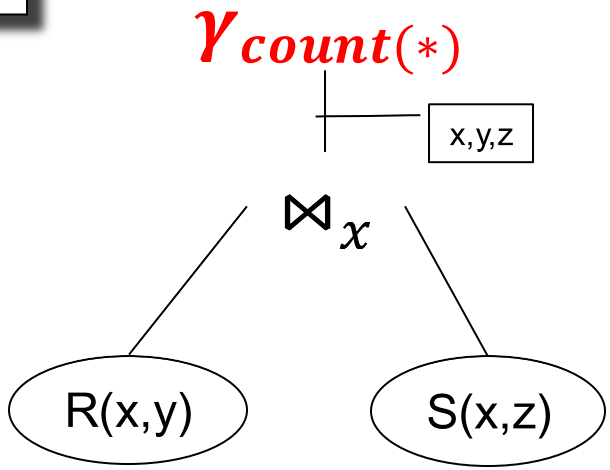
x	y
b	a
b	c
f	d
h	g

S:

x	z
b	g
b	k
h	m

Answer = 5

Runtime = ?????



Generalized Distributivity Law

SELECT count(*) from R, S where R.x=S.x

R:

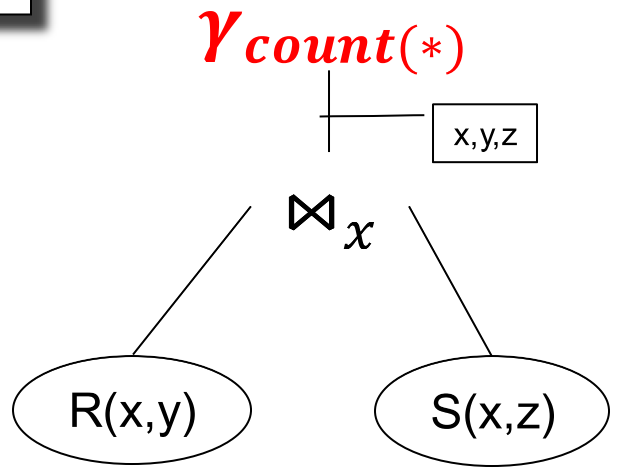
x	y
b	a
b	c
f	d
h	g

S:

x	z
b	g
b	k
h	m

Answer = 5

Runtime = $O(N^2)$



Generalized Distributivity Law

SELECT count(*) from R, S where R.x=S.x

R:

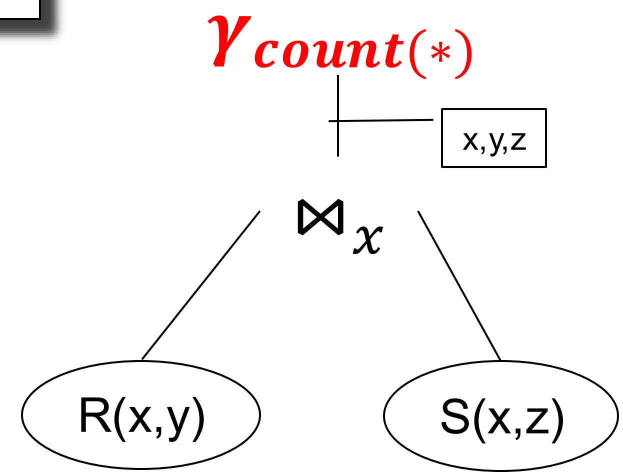
x	y
b	a
b	c
f	d
h	g

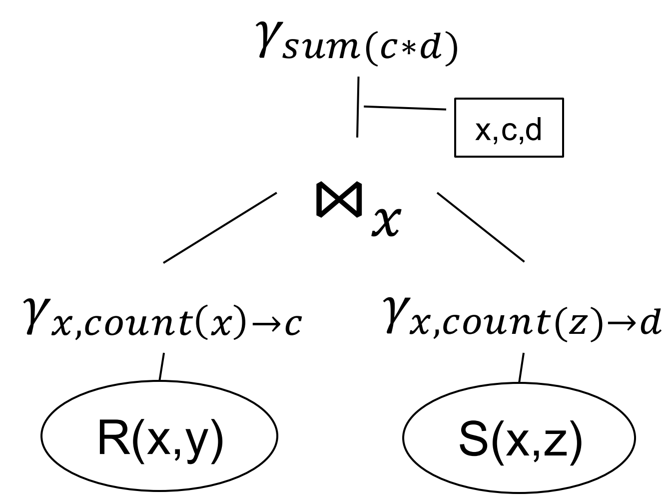
S:

x	z
b	g
b	k
h	m

Answer = 5

Runtime = $O(N^2)$





Generalized Distributivity Law

SELECT count(*) from R, S where R.x=S.x

R:

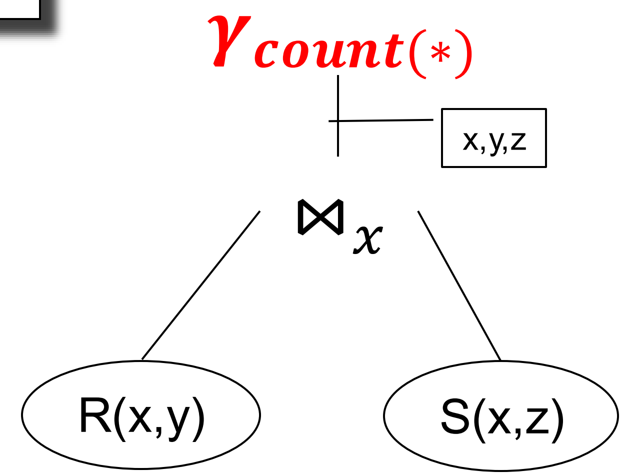
x	y
b	a
b	c
f	d
h	g

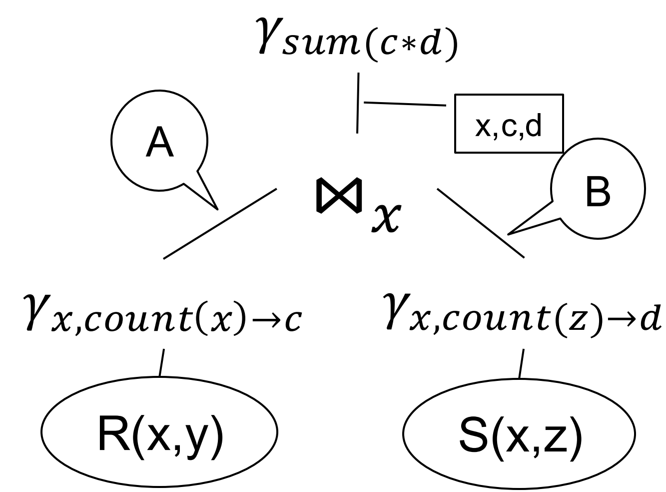
S:

x	z
b	g
b	k
h	m

Answer = 5

Runtime = $O(N^2)$





Generalized Distributivity Law

SELECT count(*) from R, S where R.x=S.x

R:

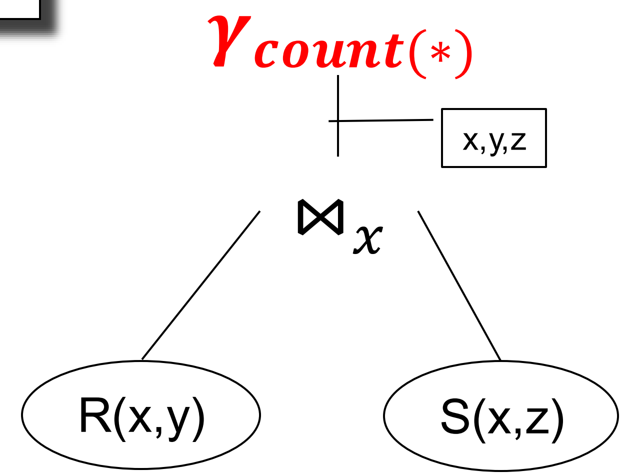
x	y
b	a
b	c
f	d
h	g

S:

x	z
b	g
b	k
h	m

Answer = 5

Runtime = O(N^2)



A:

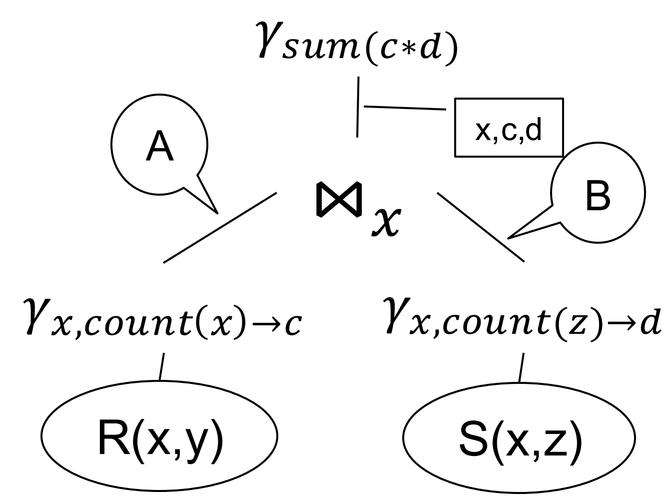
x	c
b	2
f	1
h	1

B:

x	d
b	2
h	1

A⋈B

x	c	d
b	2	2
h	1	1



Generalized Distributivity Law

SELECT count(*) from R, S where R.x=S.x

R:

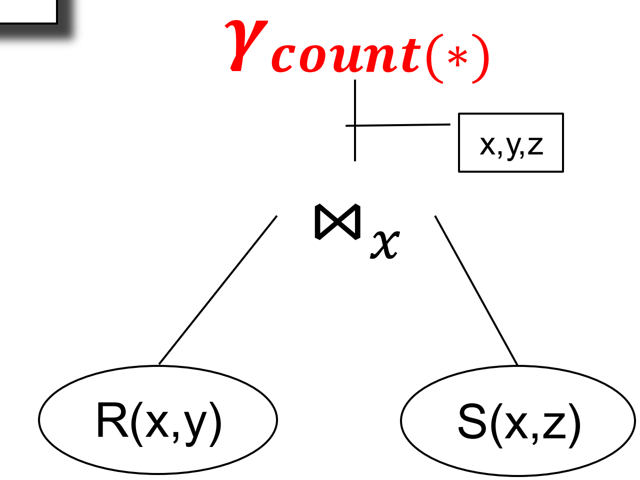
x	y
b	a
b	c
f	d
h	g

S:

x	z
b	g
b	k
h	m

Answer = 5

Runtime = O(N^2)



Runtime = ????

A:

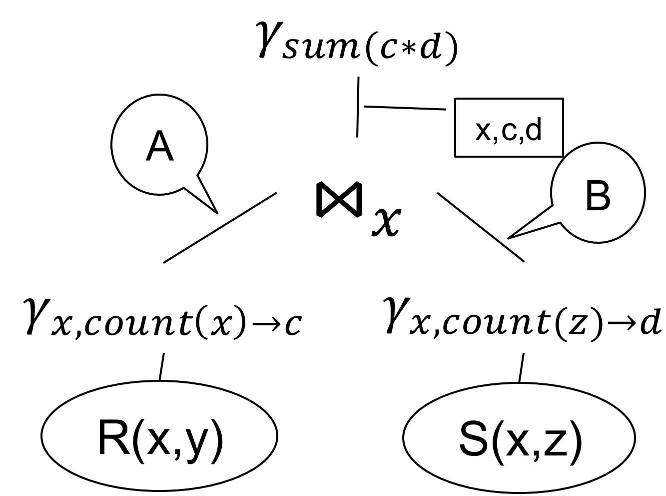
x	c
b	2
f	1
h	1

B:

x	d
b	2
h	1

A⋈B

x	c	d
b	2	2
h	1	1



Generalized Distributivity Law

SELECT count(*) from R, S where R.x=S.x

R:

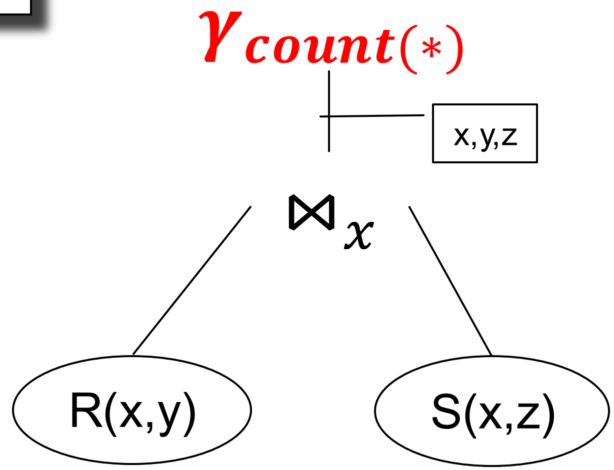
x	y
b	a
b	c
f	d
h	g

S:

x	z
b	g
b	k
h	m

Answer = 5

Runtime = $O(N^2)$



Runtime = O(N)

A:

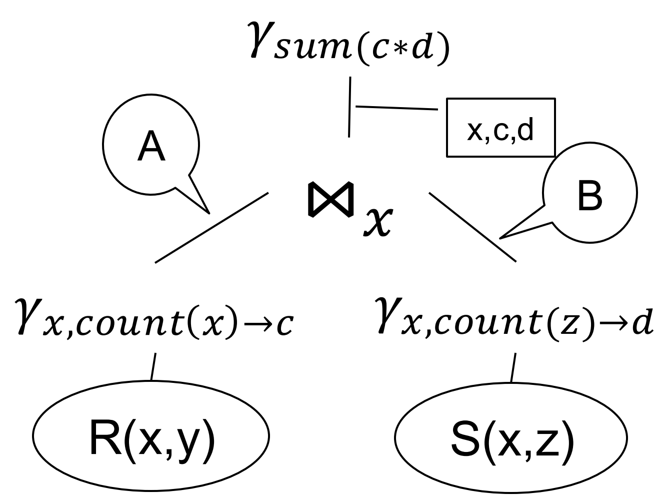
x	c
b	2
f	1
h	1

B:

x	d
b	2
h	1

A⋈B

x	c	d
b	2	2
h	1	1



Generalized Distributivity Law

$$\gamma_{Y,Z,sum(A*B*C*\cdots)}$$

$$\bowtie_X$$

$\cdots$ $\cdots$

Generalized Distributivity Law

$\gamma_{Y,Z,sum(A*B*C*\cdots)}$

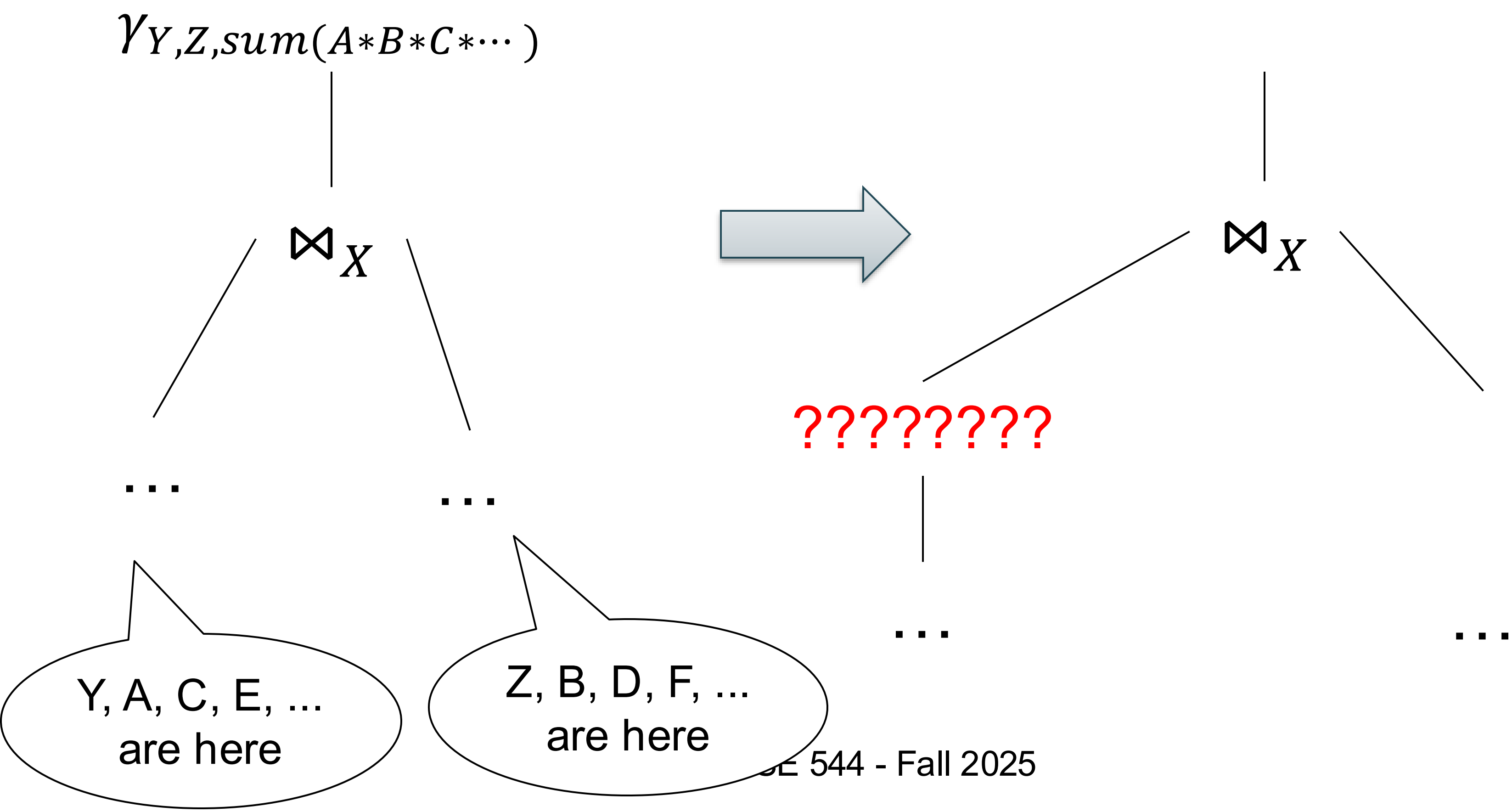
$\bowtie_X$

$\ldots$ $\ldots$

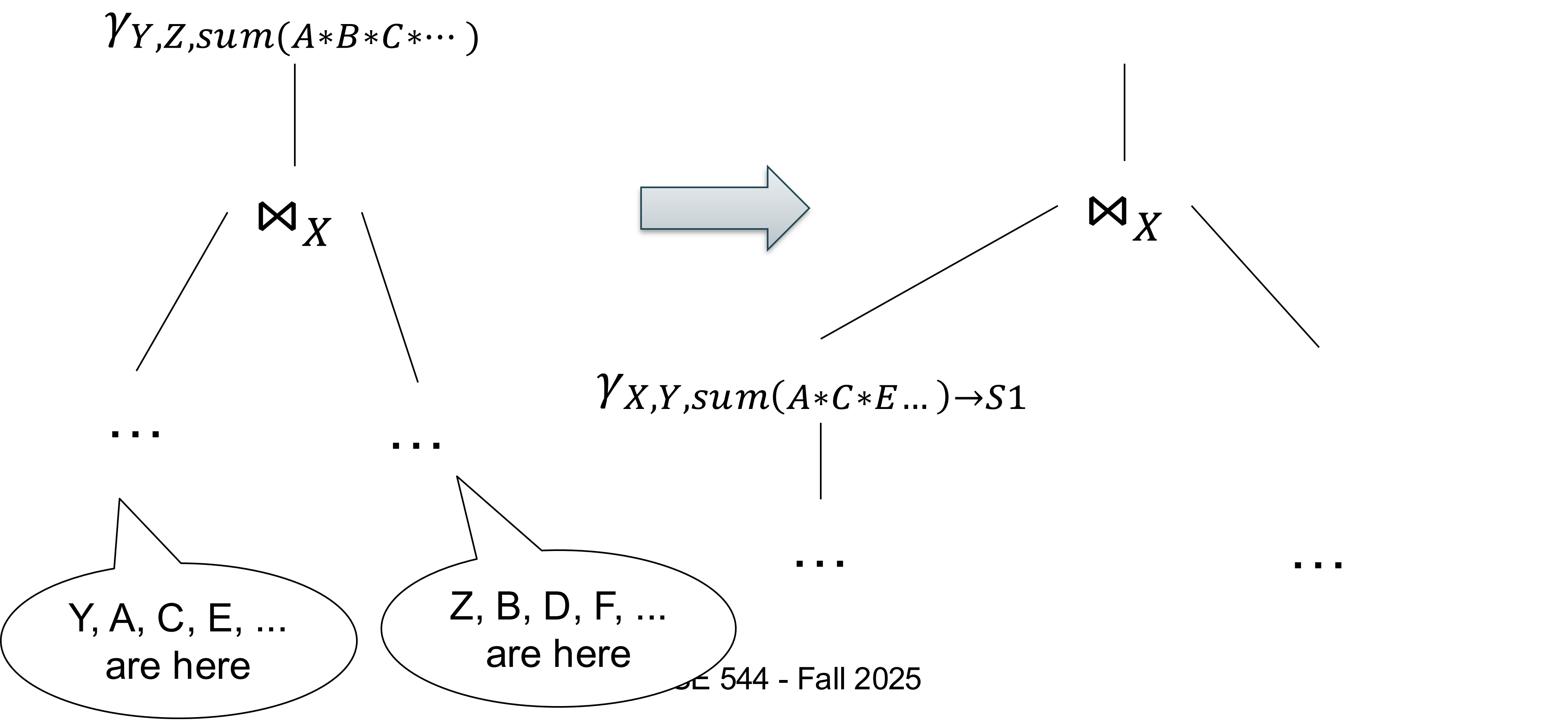
Y, A, C, E, ...
are here

Z, B, D, F, ...
are here

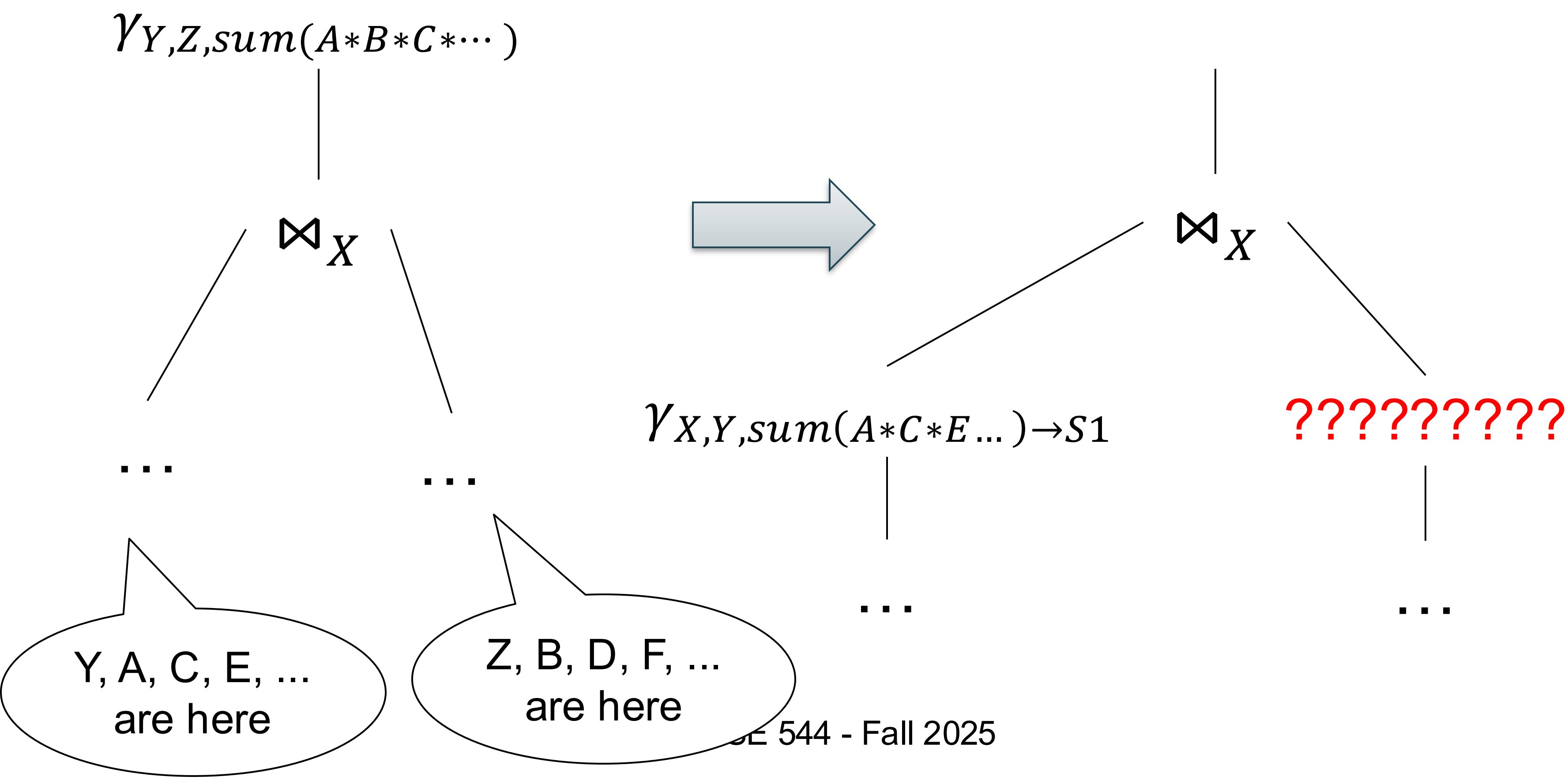
Generalized Distributivity Law



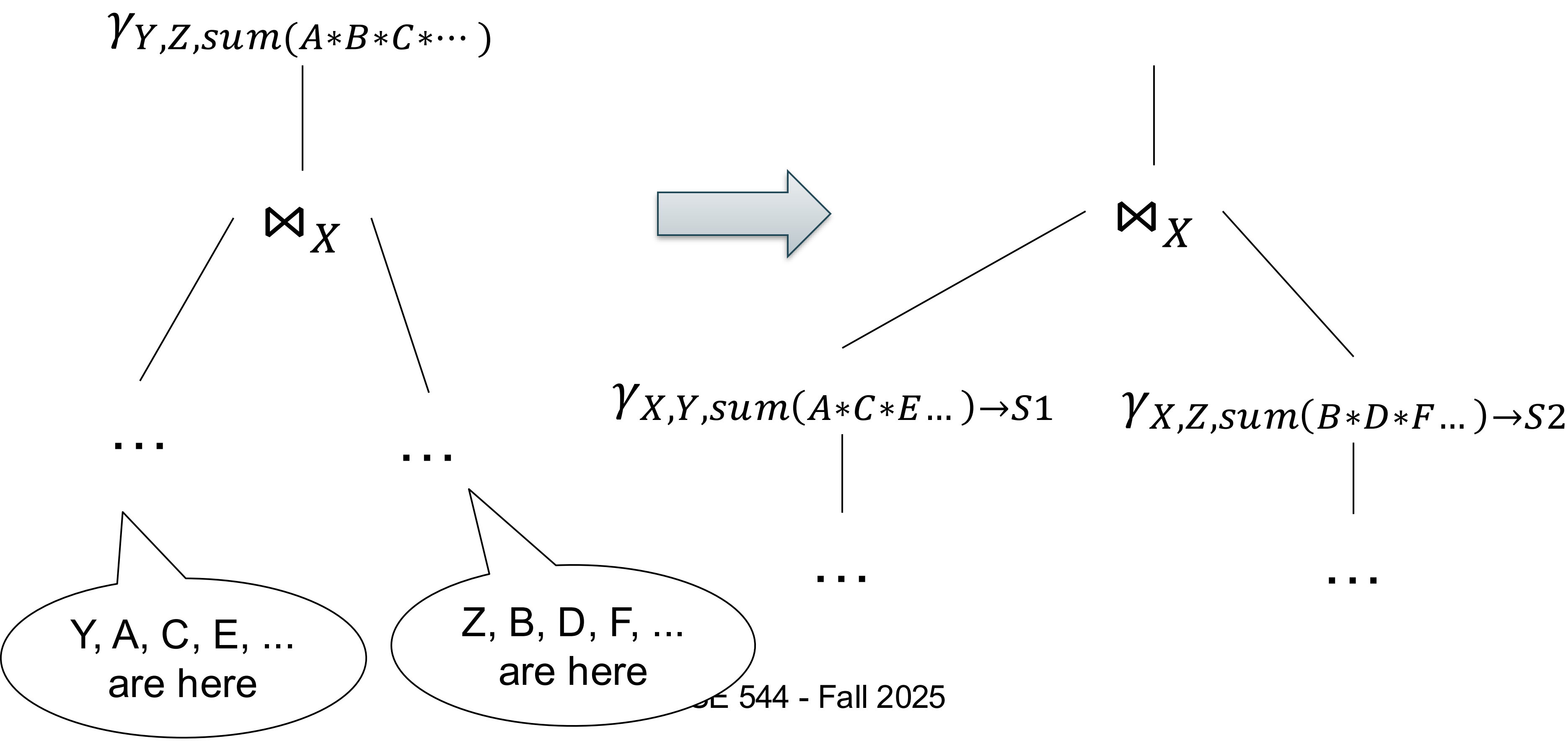
Generalized Distributivity Law



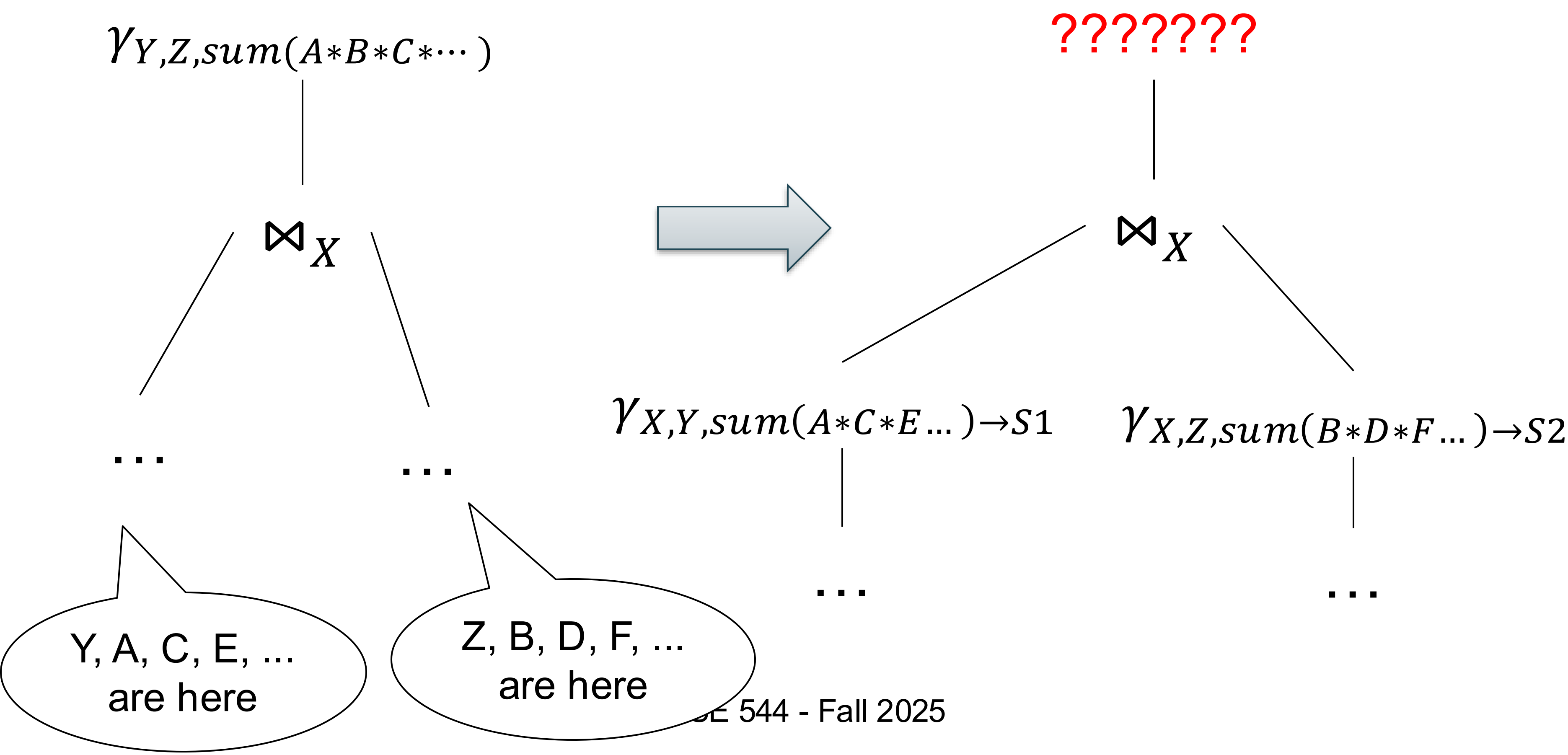
Generalized Distributivity Law



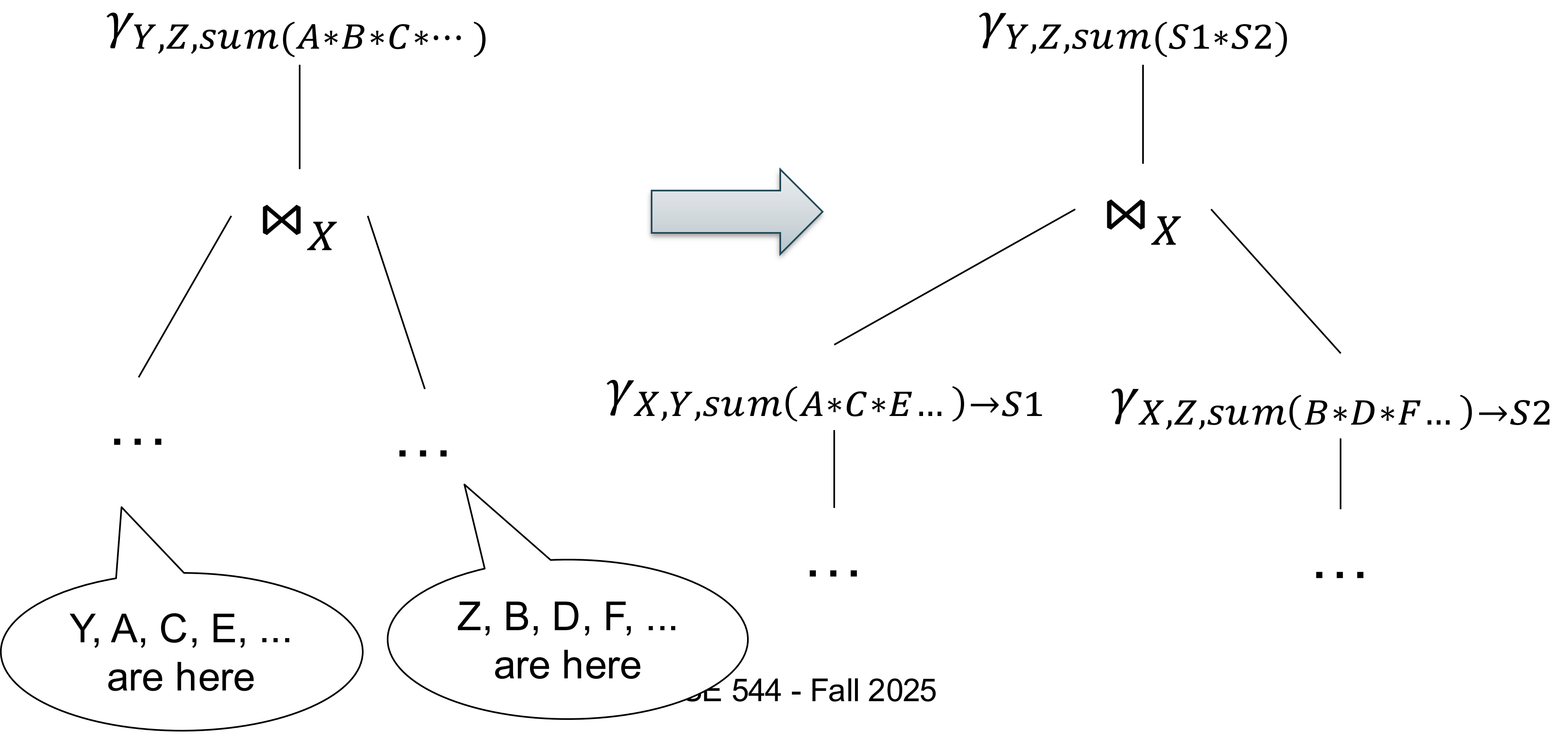
Generalized Distributivity Law



Generalized Distributivity Law



Generalized Distributivity Law



Discussion

- Most DBMS support only some subset of the generalized distributivity law
- Fun activity: check if your favorite DBMS supports it
- Next: a rewrite rule that most DBMS support

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

Key / Foreign-Key

```
Select x.pno, x.quantity
From Supply x, Supplier y
Where x.sid = y.sid
```

?

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

Key / Foreign-Key

```
Select x.pno, x.quantity
From Supply x, Supplier y
Where x.sid = y.sid
```

```
Select x.pno, x.quantity
From Supply x
```

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

Key / Foreign-Key

```
Select x.pno, x.quantity
From Supply x, Supplier y
Where x.sid = y.sid
```

```
Select x.pno, x.quantity
From Supply x
```

Only if these constraints hold:

1. Supplier.sid = key
2. Supply.sid = foreign key
3. Supply.sid NOT NULL

Summary: Rules

$$R \bowtie S = S \bowtie R$$

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

Summary: Rules

$$R \bowtie S = S \bowtie R$$

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

$$\sigma_{C_1 \wedge C_2}(R) = \sigma_{C_1}(\sigma_{C_2}(R))$$

Summary: Rules

$R \bowtie S = S \bowtie R$

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

$\sigma_{C_1 \wedge C_2}(R) = \sigma_{C_1}(\sigma_{C_2}(R))$

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

Summary: Rules

$$R \bowtie S = S \bowtie R$$

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

$$\sigma_{C_1 \wedge C_2}(R) = \sigma_{C_1}(\sigma_{C_2}(R))$$

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

$$\gamma_{X,sum(B)}\left(\gamma_{X,Y,sum(A)\rightarrow B}(R)\right) = \gamma_{X,sum(A)}(R)$$

Summary: Rules

$$R \bowtie S = S \bowtie R$$

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

In particular,
$\delta(\delta(R) = \delta(R)$

$$\sigma_{C_1 \wedge C_2}(R) = \sigma_{C_1}(\sigma_{C_2}(R))$$

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

$$\gamma_{X,sum(B)}\left(\gamma_{X,Y,sum(A)\to B}(R)\right) = \gamma_{X,sum(A)}(R)$$

Summary: Rules

$$R \bowtie S = S \bowtie R$$

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

In particular,
$\delta(\delta(R) = \delta(R)$

$$\sigma_{C_1 \wedge C_2}(R) = \sigma_{C_1}(\sigma_{C_2}(R))$$

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

$$\gamma_{X,sum(B)}\left(\gamma_{X,Y,sum(A)\to B}(R)\right) = \gamma_{X,sum(A)}(R)$$

$$\gamma_{X,sum(A*B)}(R \bowtie_{Y=Z} S) =$$
$$\gamma_{X,sum(C*D)}\left(\gamma_{X,Y,sum(A)\to C}(R) \bowtie_{Y=Z} \gamma_{X,Z,sum(B)\to D}(S)\right)$$

Summary: Rules

$R \bowtie S = S \bowtie R$

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

In particular,
$\delta(\delta(R) = \delta(R)$

$\sigma_{C_1 \wedge C_2}(R) = \sigma_{C_1}(\sigma_{C_2}(R))$

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

$\gamma_{X,sum(B)}\left(\gamma_{X,Y,sum(A)\to B}(R)\right) = \gamma_{X,sum(A)}(R)$

$\gamma_{X,sum(A*B)}(R \bowtie_{Y=Z} S) =$
$\quad \gamma_{X,sum(C*D)}(\gamma_{X,Y,sum(A)\to C}(R) \bowtie_{Y=Z} \gamma_{X,Z,sum(B)\to D}(S))$

$\Pi_{attr(R)}(R \bowtie_{X=Y} S) = R$ if S.Y key, R.X fk not null

…

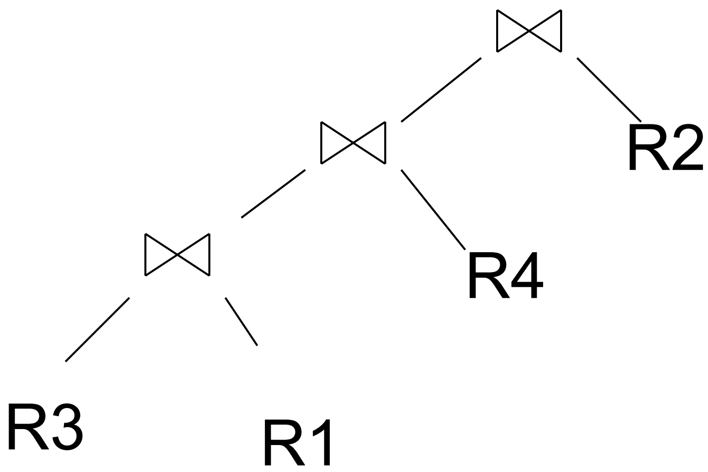
Practice

- Database optimizers typically have a database of rewrite rules
- E.g. SQL Server: about 500 rules
- Rules become complex as they need to serve specialized types of queries

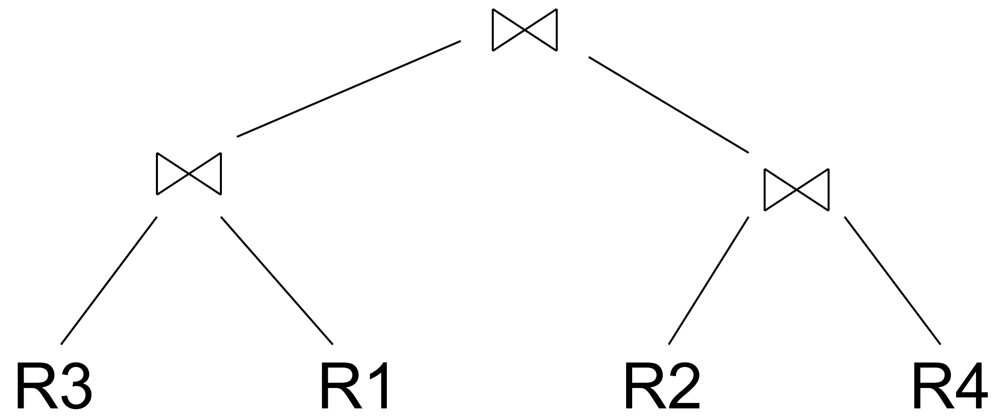
Search Space Challenges

- Search space is huge
- Typical compromises:
  - Left-deep plans
  - Plans without cartesian products

Left-Deep Plans and Bushy Plans



Left-deep plan



Bushy plan

Cartesian Product

- Plan without cartesian product:
  $$(R(A,B) \bowtie S(B,C)) \bowtie T(C,D)$$
- Plan with cartesian product
  $$\big(R(A,B) \bowtie T(C,D)\big) \bowtie S(B,C)$$
- Some optimizers avoid cartesian products

Recap: Search Space

Refers to the set of all possible alternate plans that the optimizers may explore

- Access Path Selection: how to use indices
- Rewrite Rules: what identities the optimizer applies
- Scheduling: pipeline/blocking next week

Recap: Query Optimization

<u>Three major components</u>:

1. Search space

Discussed today
(less scheduling)

2. Cardinality and cost estimation

Continued
next week

3. Plan enumeration algorithms