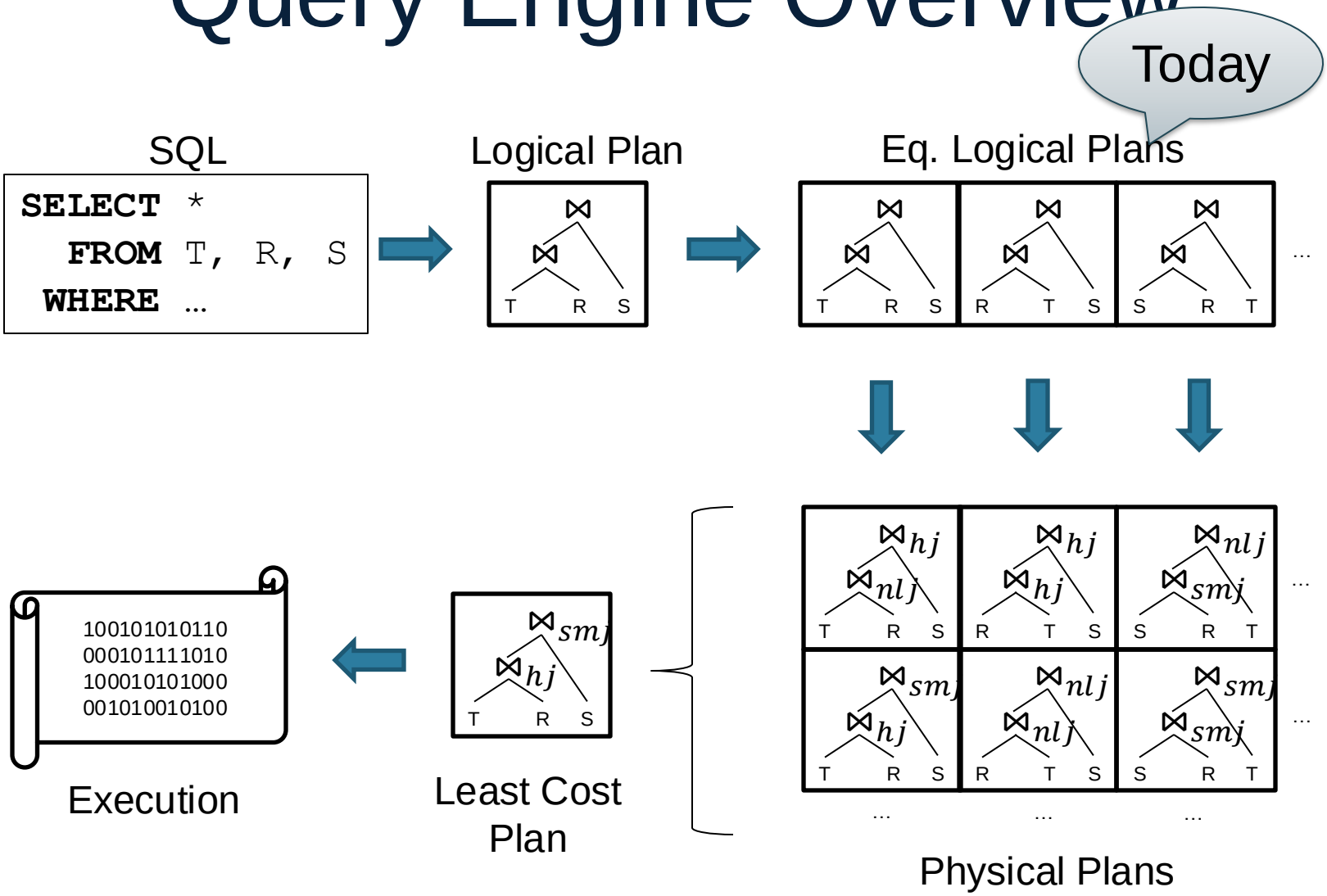
CSE544
Data Management

Lecture 6
Query Optimization

Announcements

- HW1 is graded
- P1 Project Proposal is graded
- Next week lecture moved to Friday 2/21
- HW3 is due on Sunday 2/23

Query Engine Overview



Query Optimizer

- Search Space
- Cost Estimation
- Search Algorithm

A survey that just came out

Search Space

Rewrite Rules

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

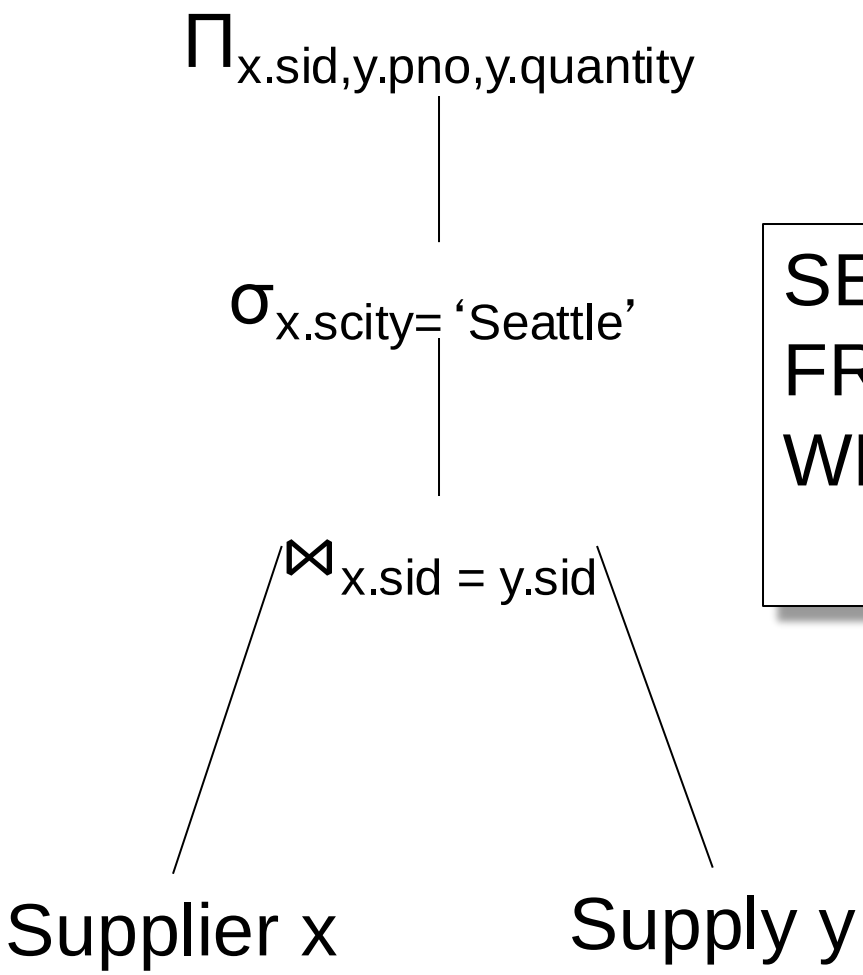
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

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)

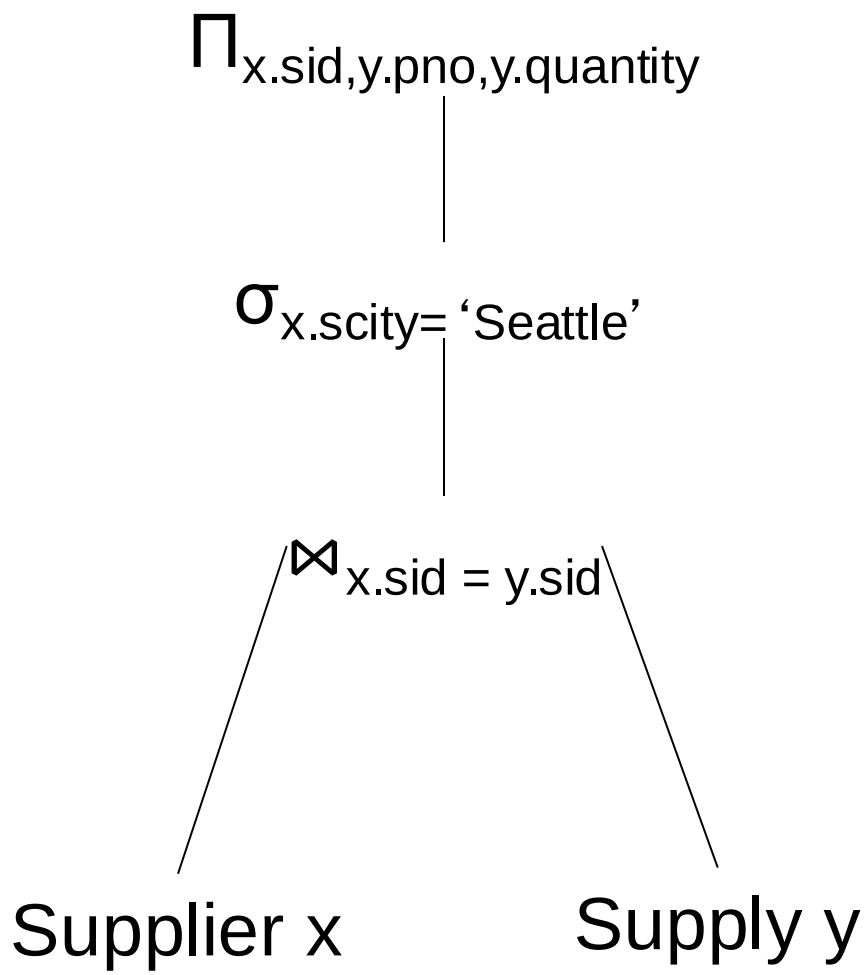
Example Optimization




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

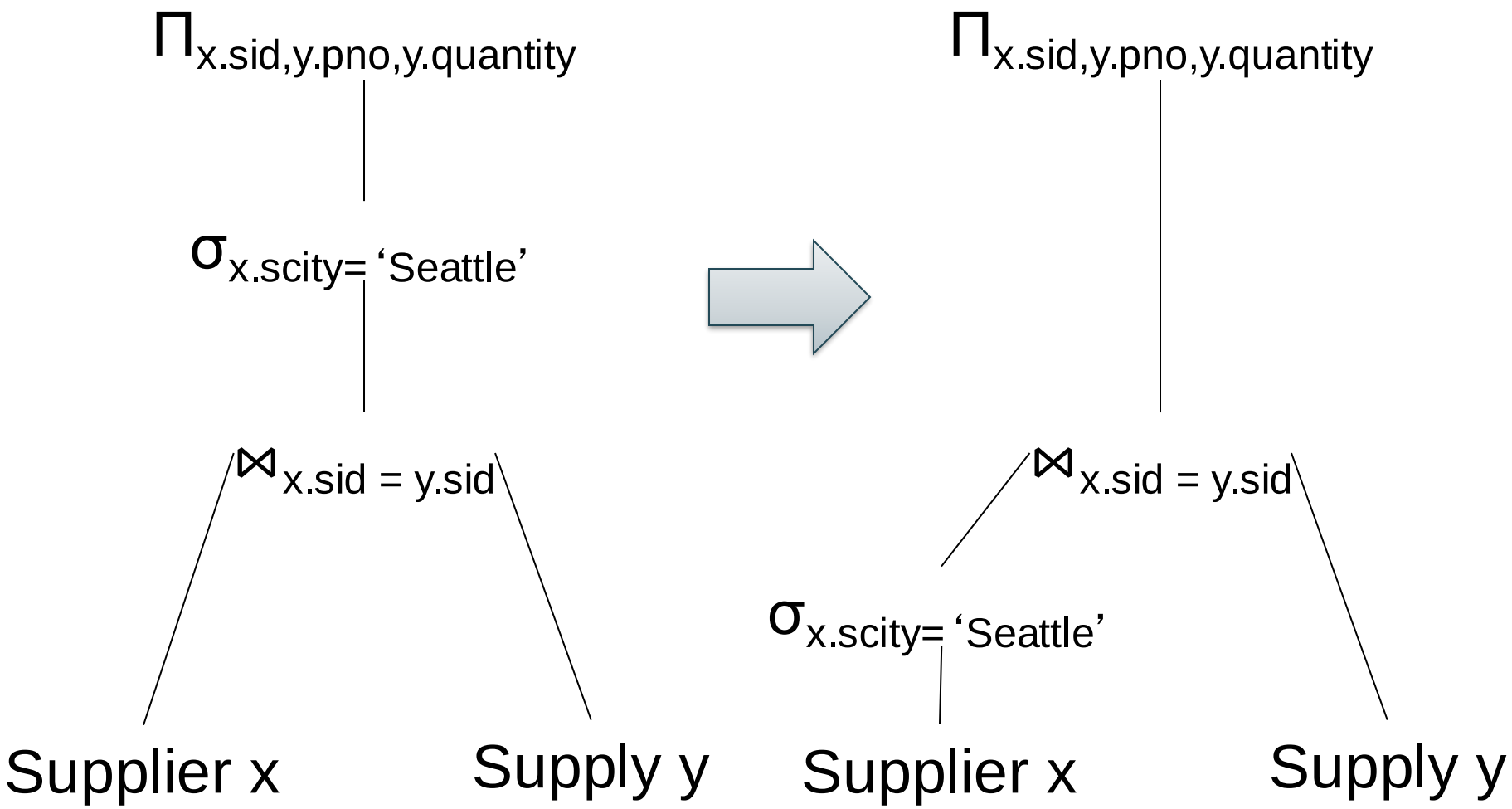
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



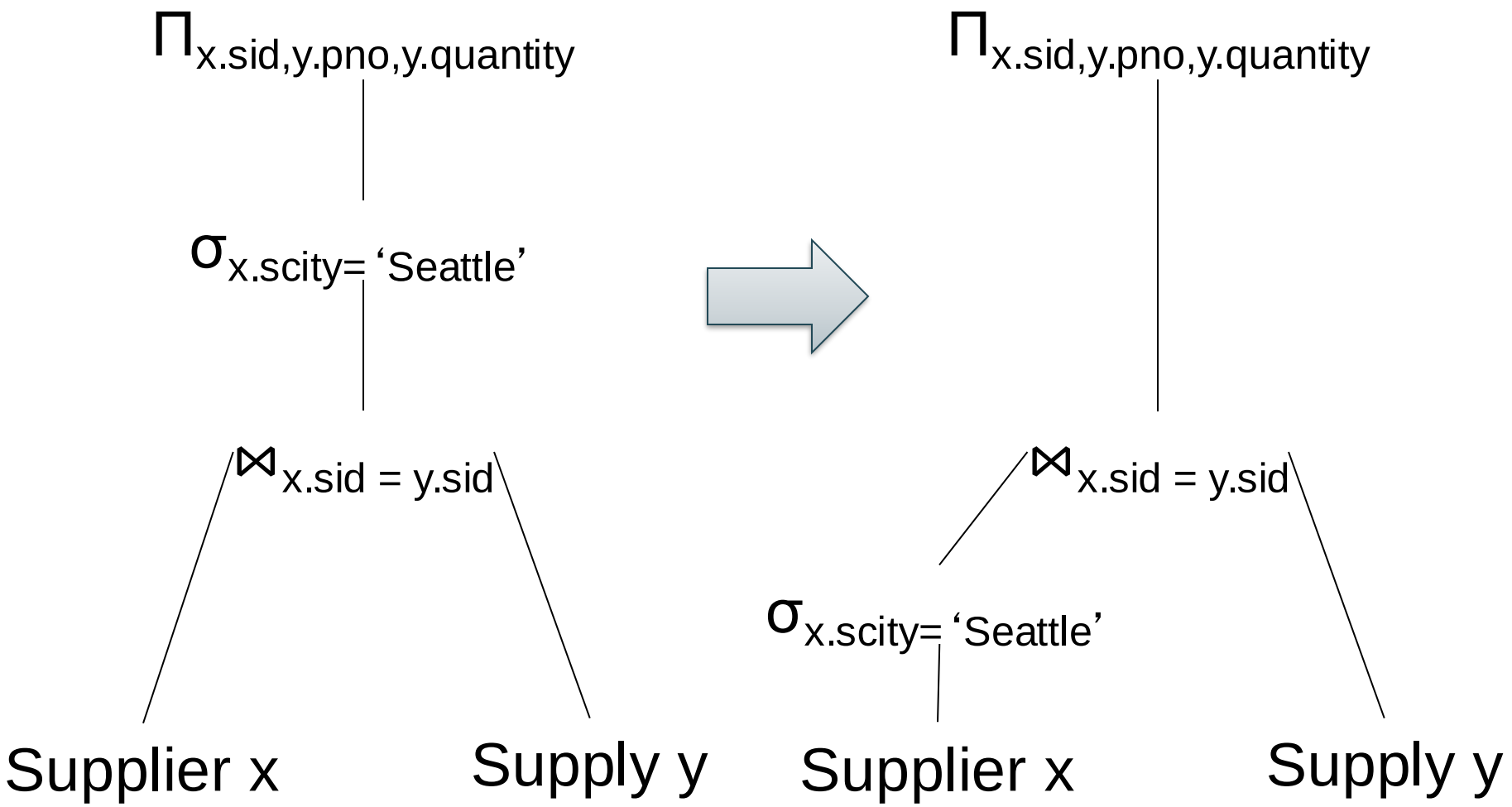
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



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

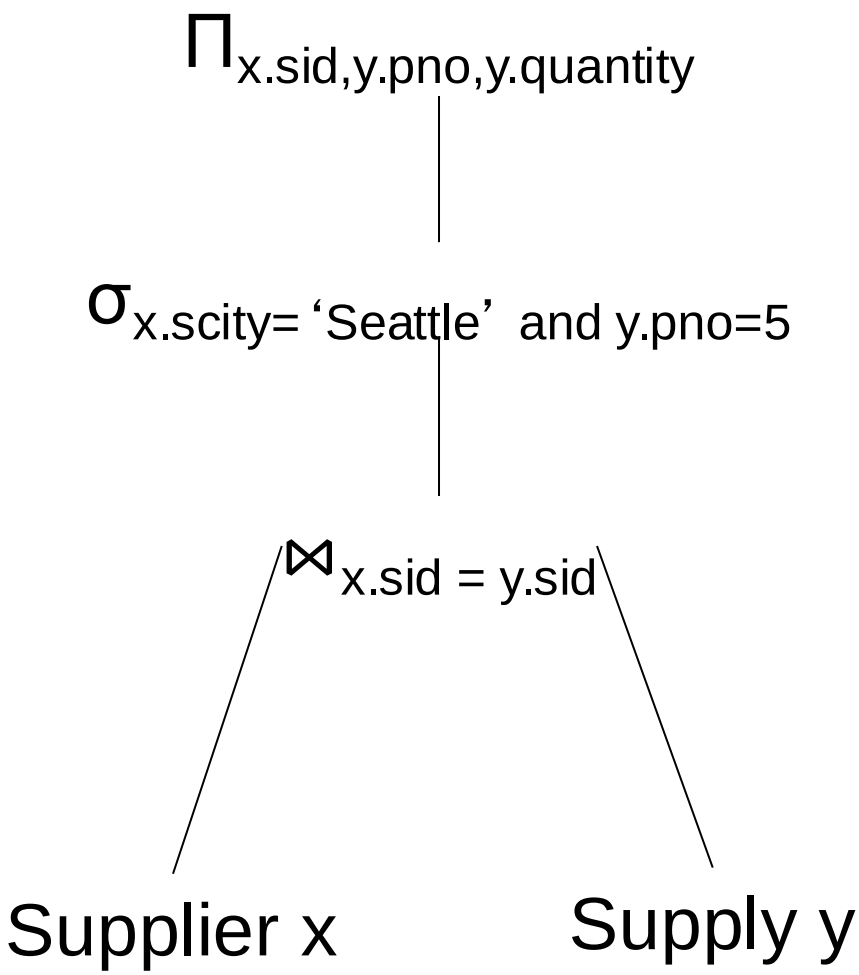
Push Selections Down



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

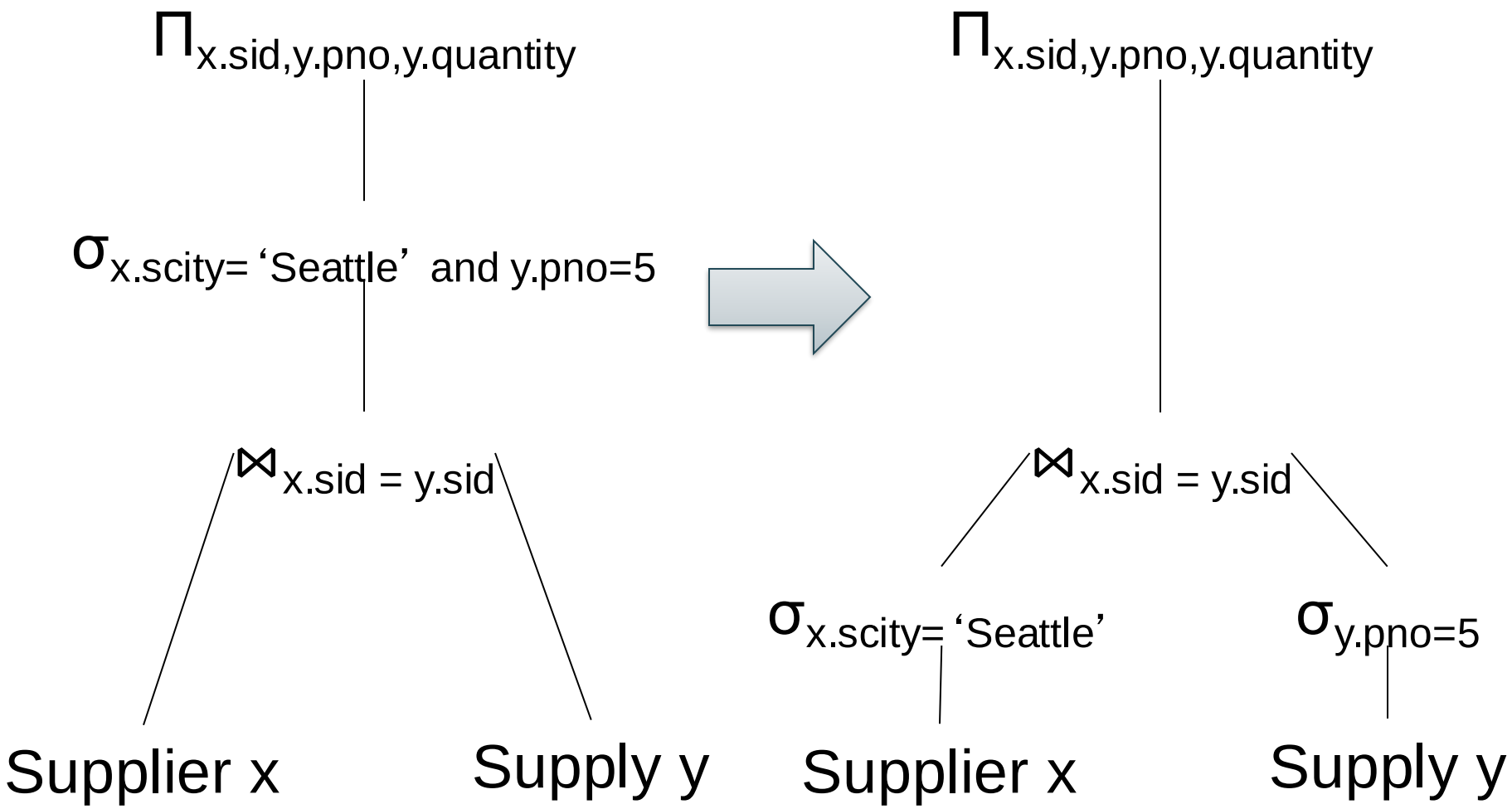
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



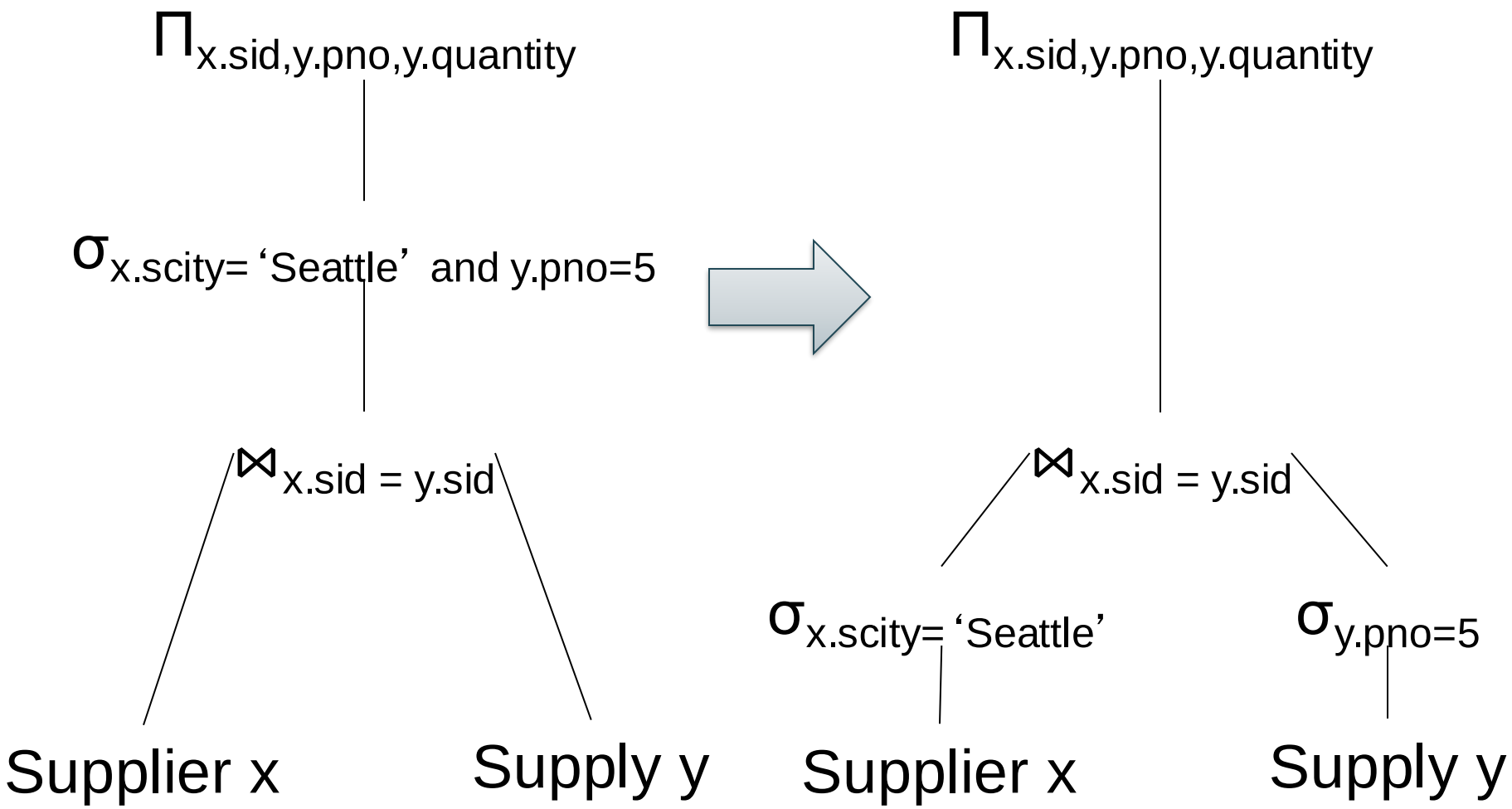
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



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

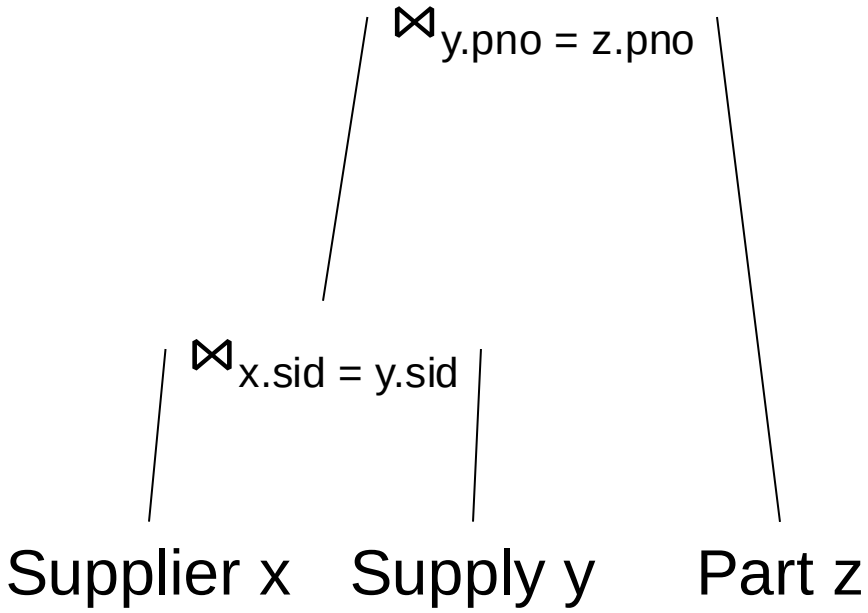
Push Selections Down



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

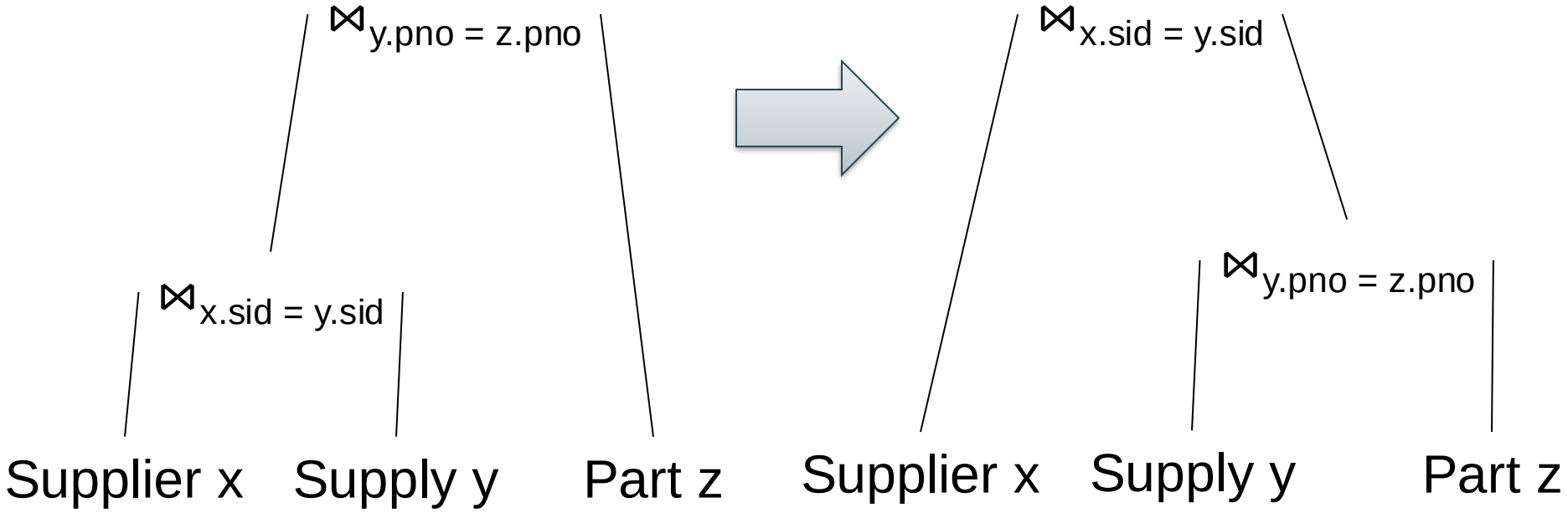
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder



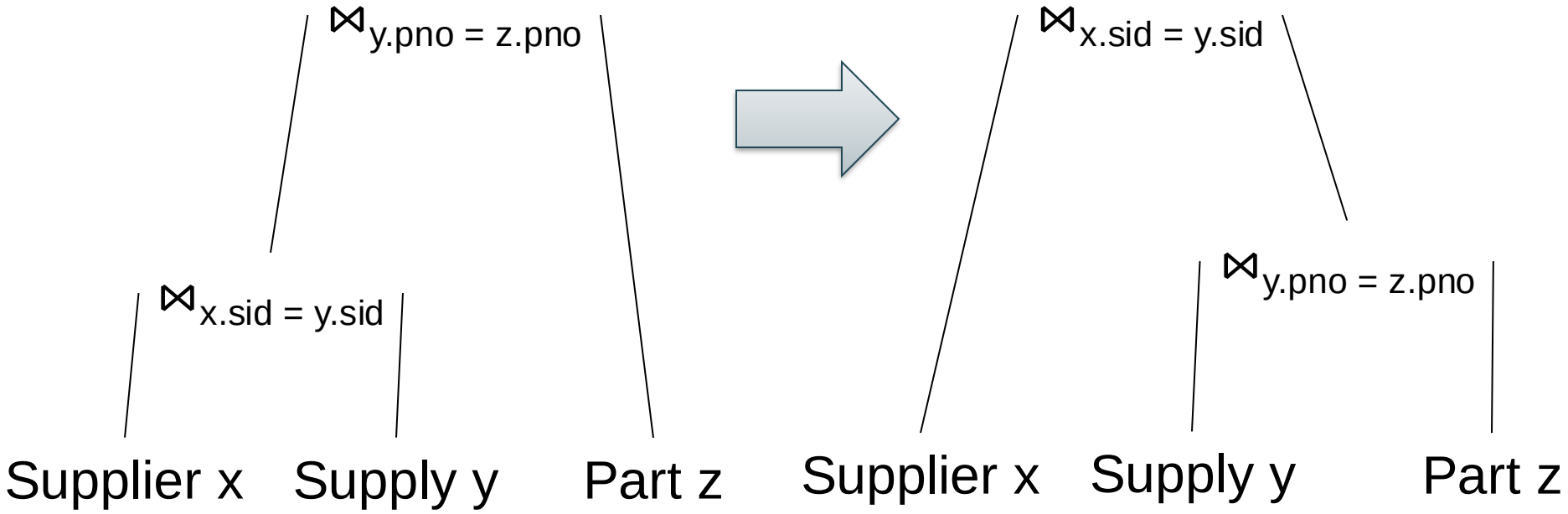
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder



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

Join Reorder



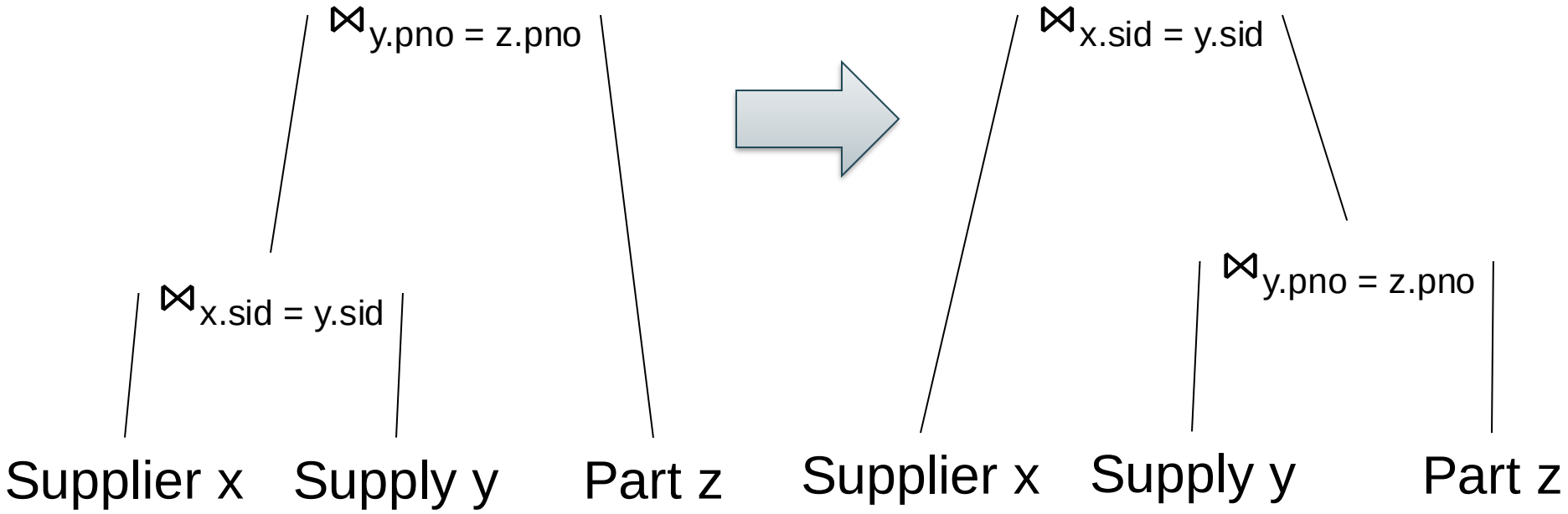
$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

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

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

Join Reorder

Which plan is better?



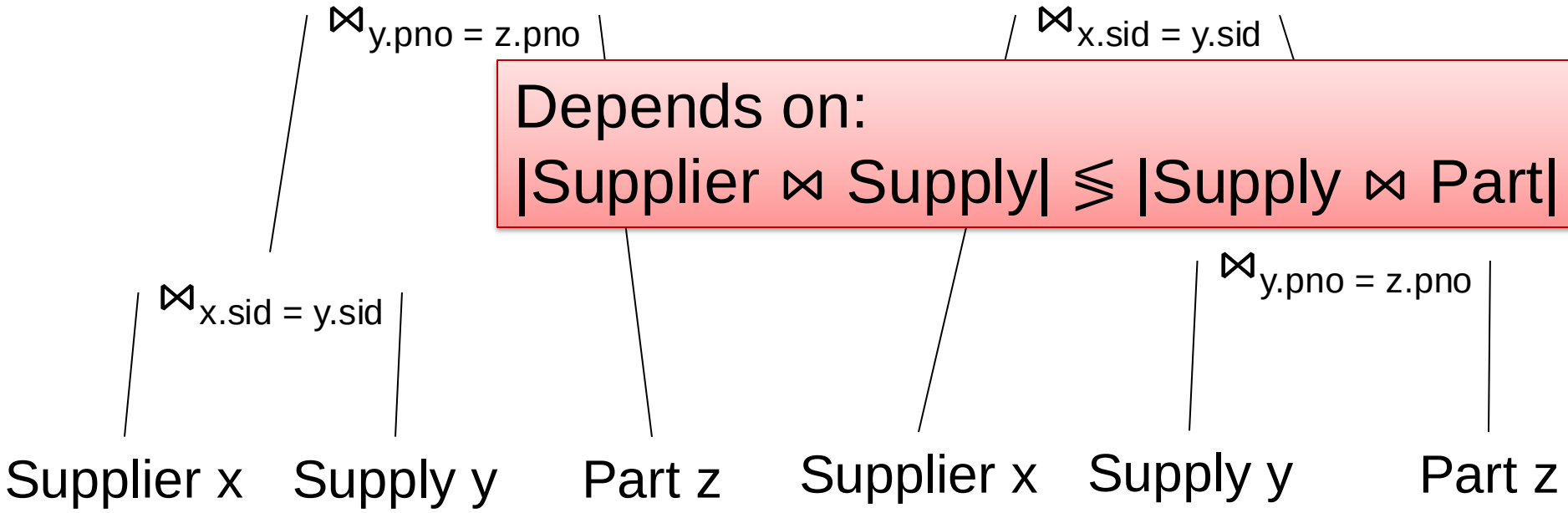
$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

$R \bowtie S = S \bowtie R$

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

Join Reorder

Which plan is better?



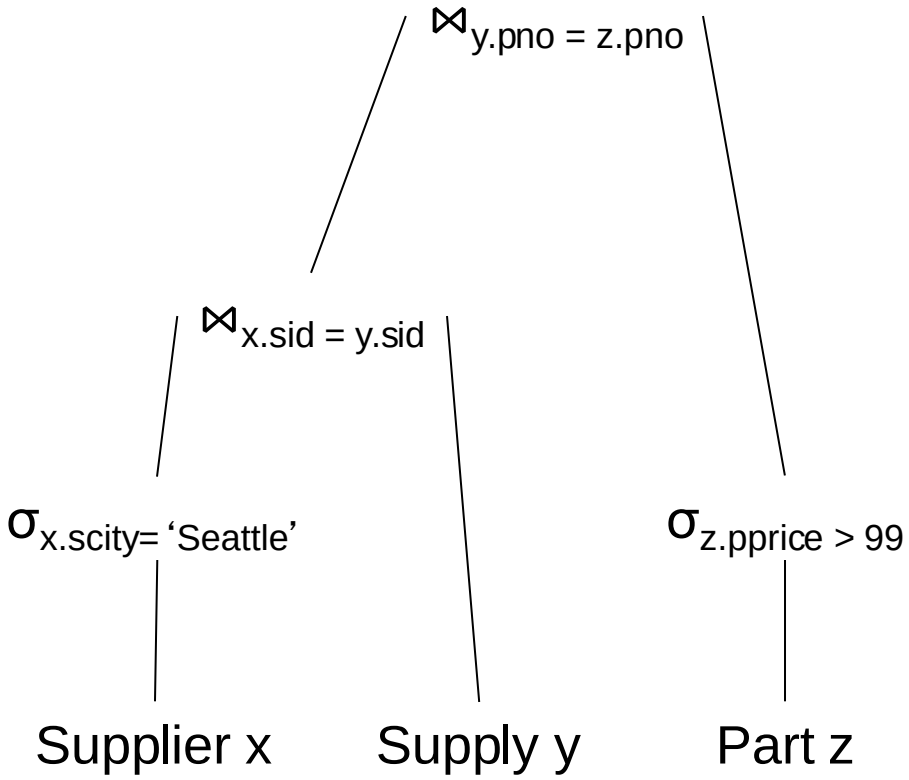
Depends on:
|Supplier ⋈ Supply| ⩽ |Supply ⋈ Part|

(R ⋈ S) ⋈ T= R ⋈ (S ⋈ T)

R ⋈ S = S ⋈ R

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

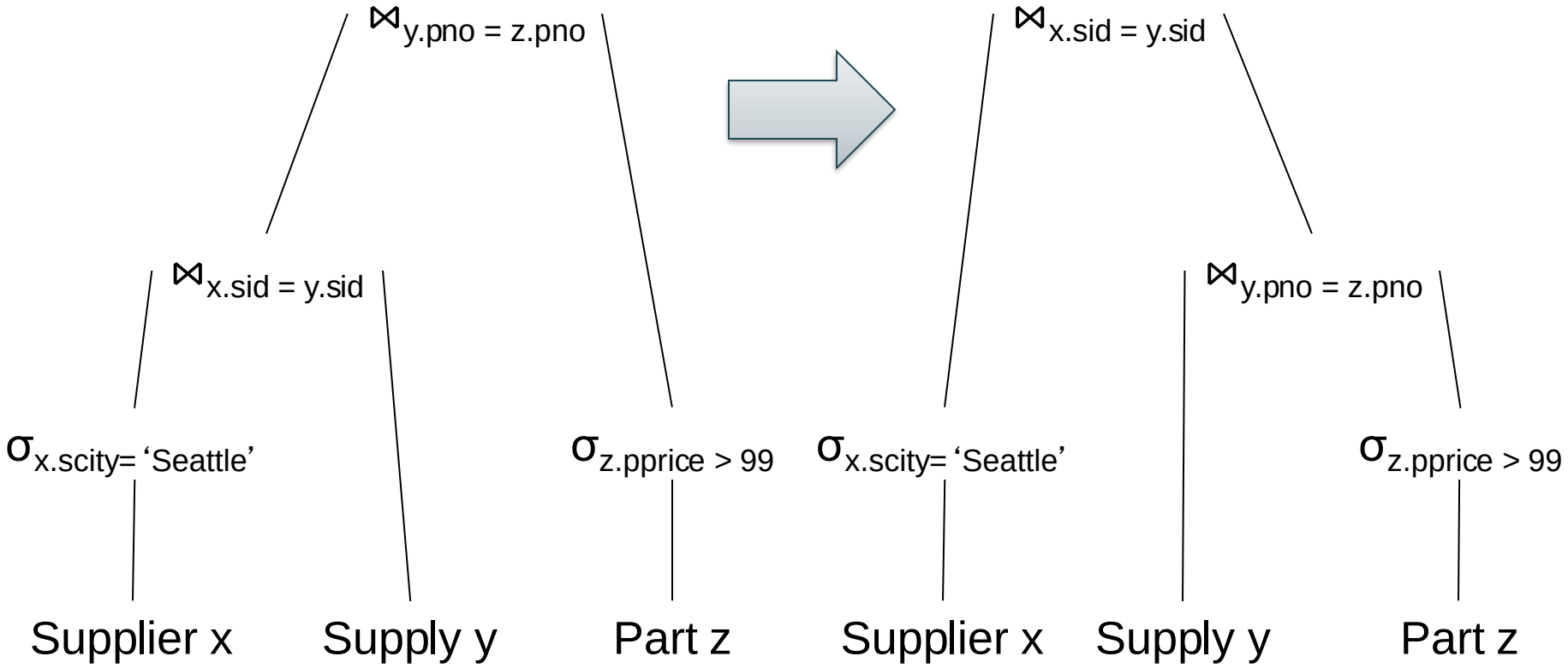
Join Reorder



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

Join Reorder

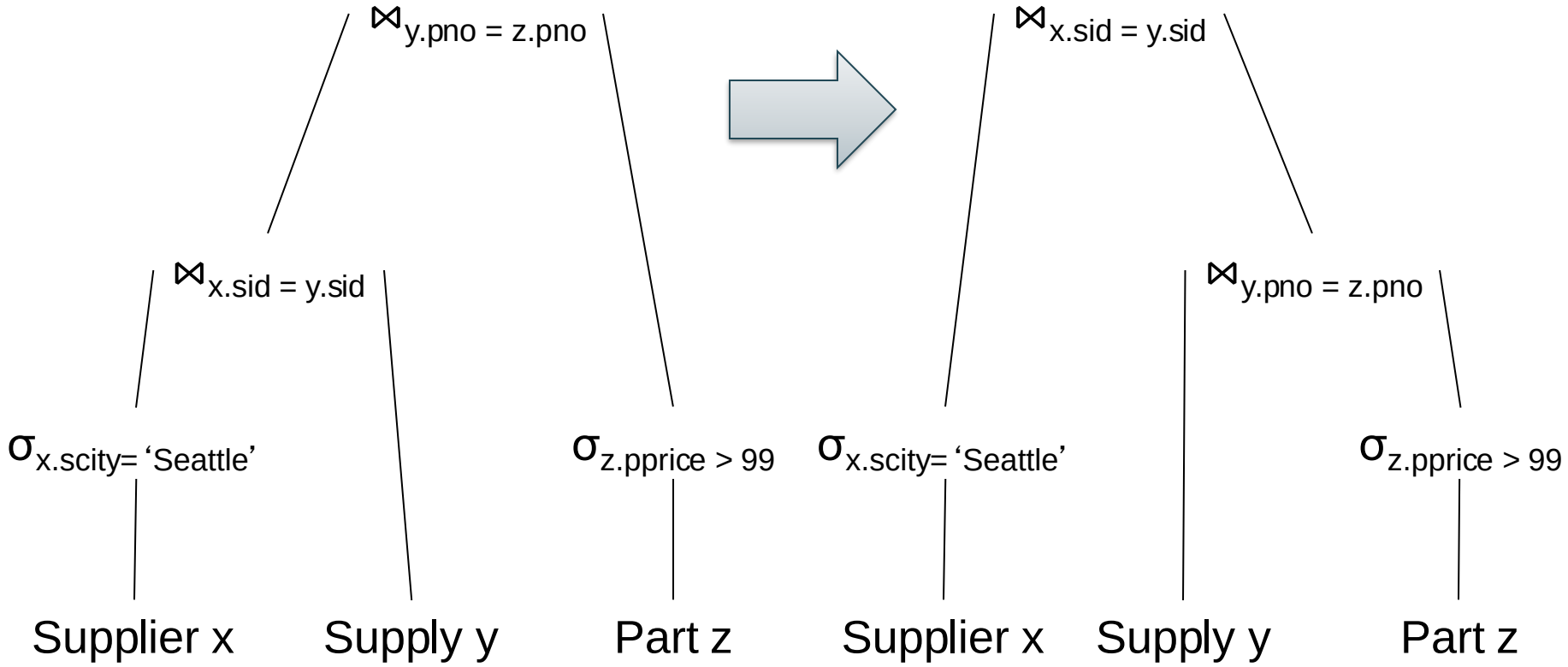
Which plan is better?



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

Join Reorder

Which plan is better?



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

Discussion

- Both rewrite rules **and** cardinality estimations are needed for optimization

- They can be developed independently

Next: aggregate pushdown rule

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!!!

The 3rd answer is simply the product of the first two!

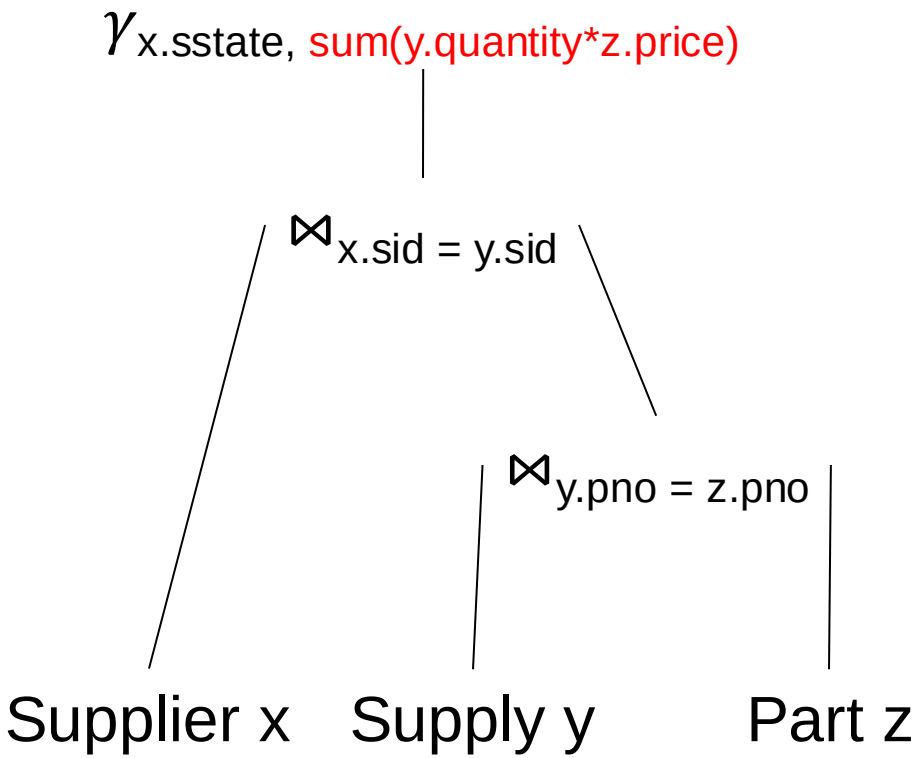
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

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)

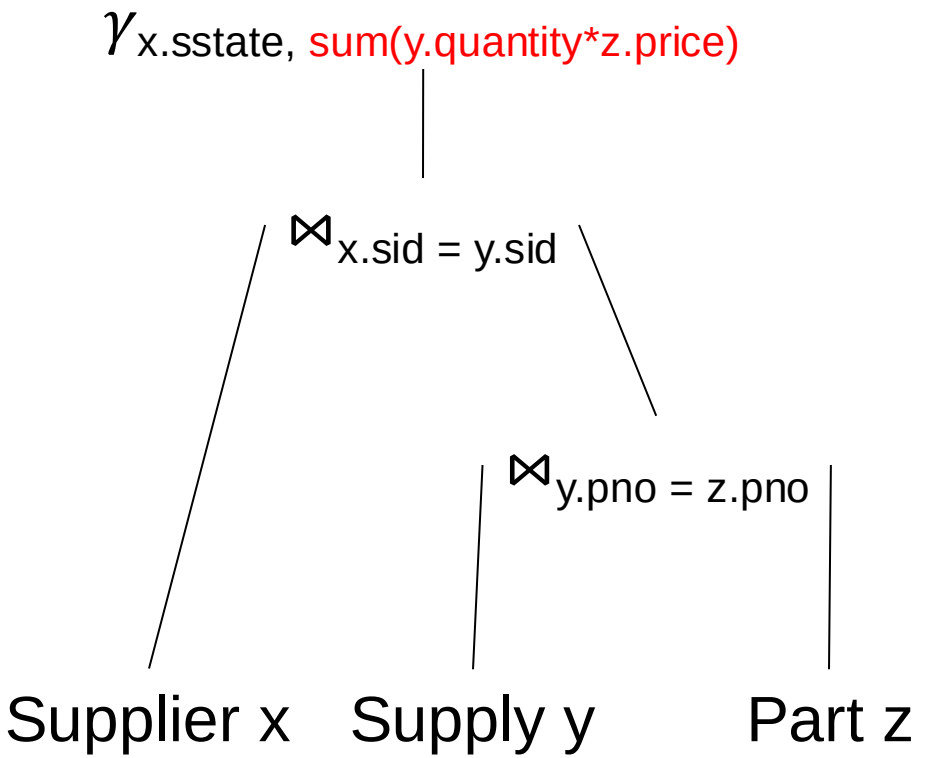
Aggregate Push-down

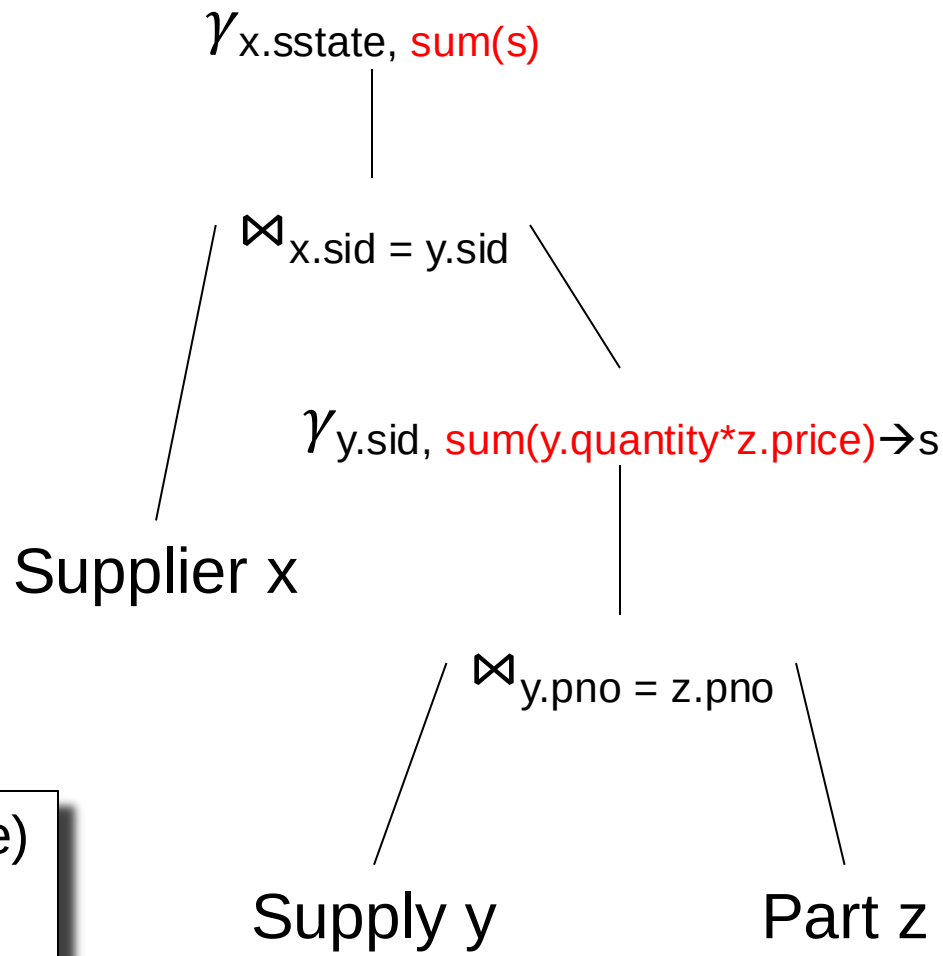



```
SELECT x.sstate, sum(y.quanity*z.price)
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
GROUP BY x.sstate
```

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

Aggregate Push-down

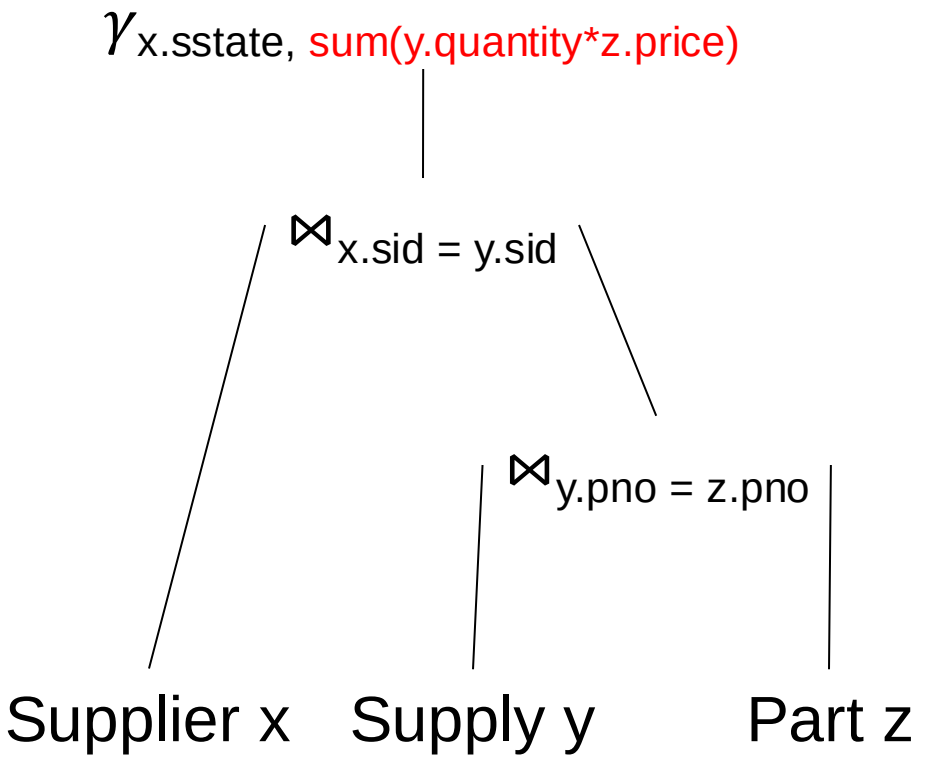


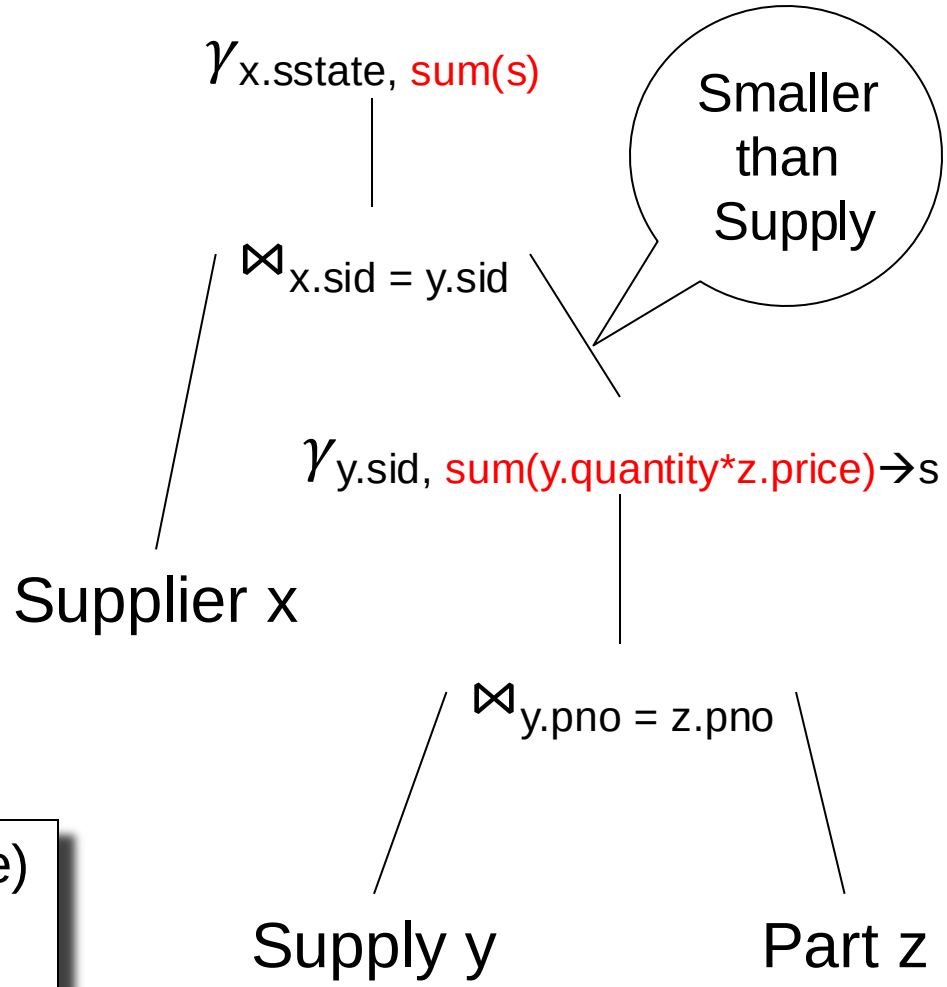



```
SELECT x.sstate, sum(y.quanity*z.price)
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
GROUP BY x.sstate
```

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

Aggregate Push-down

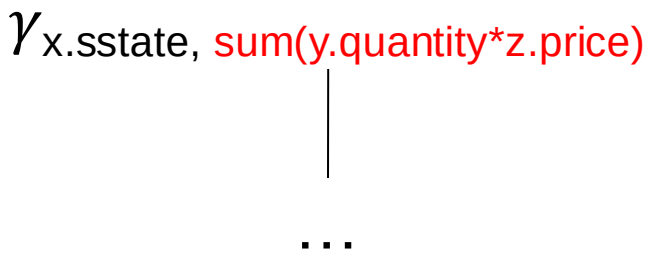


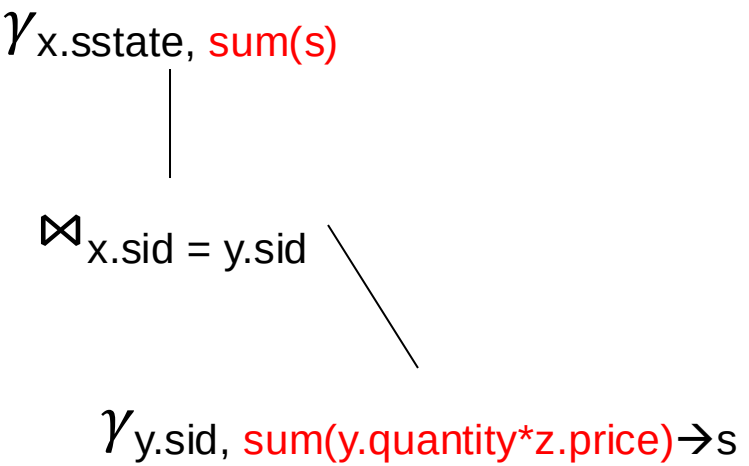


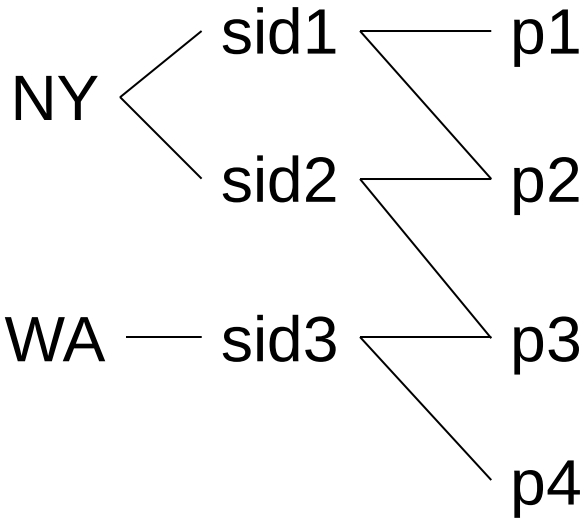

```
SELECT x.sstate, sum(y.quanity*z.price)
FROM Supplier x, Supply y, Part z
WHERE x.sid = y.sid and y.pno = z.pno
GROUP BY x.sstate
```

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

Aggregate Push-down

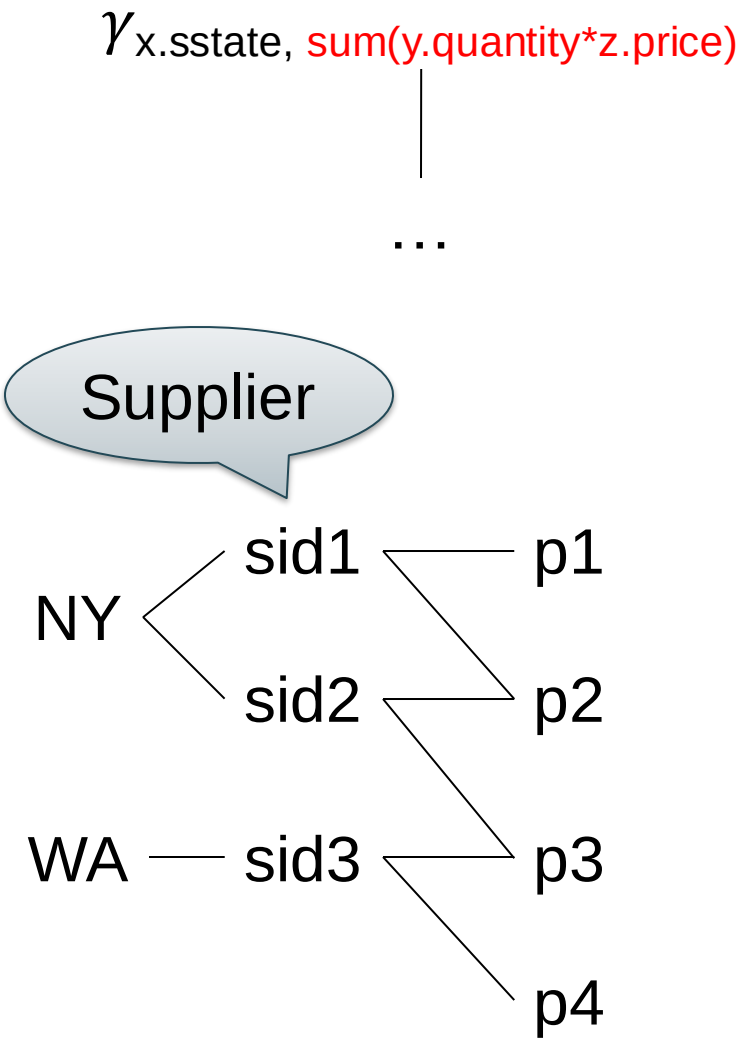


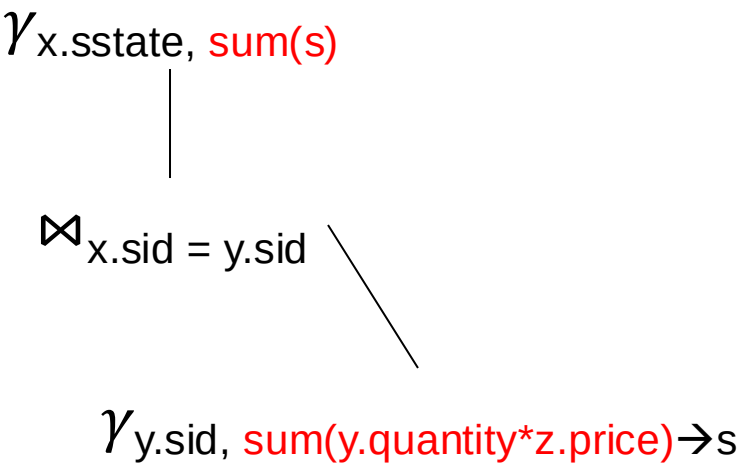




Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

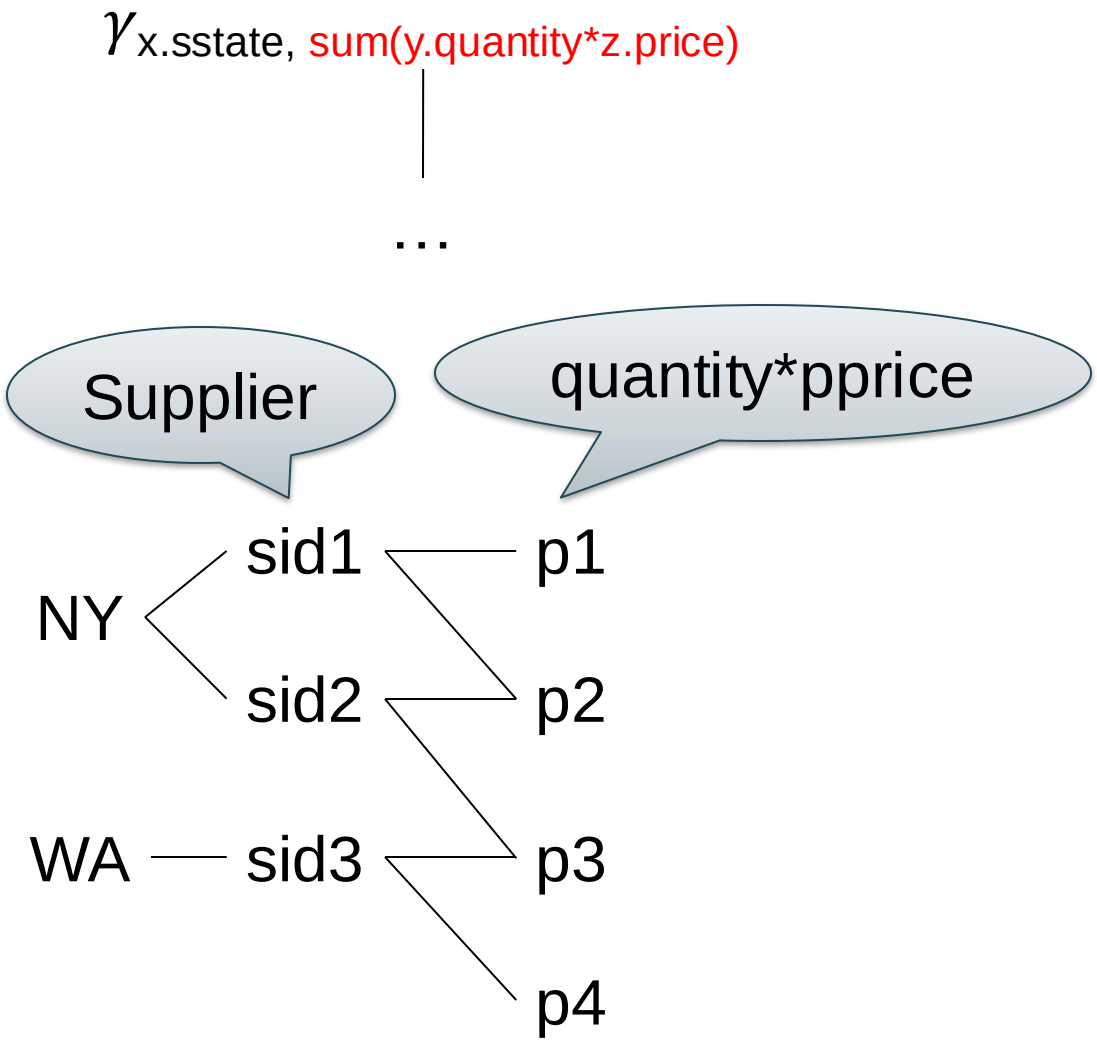
Aggregate Push-down

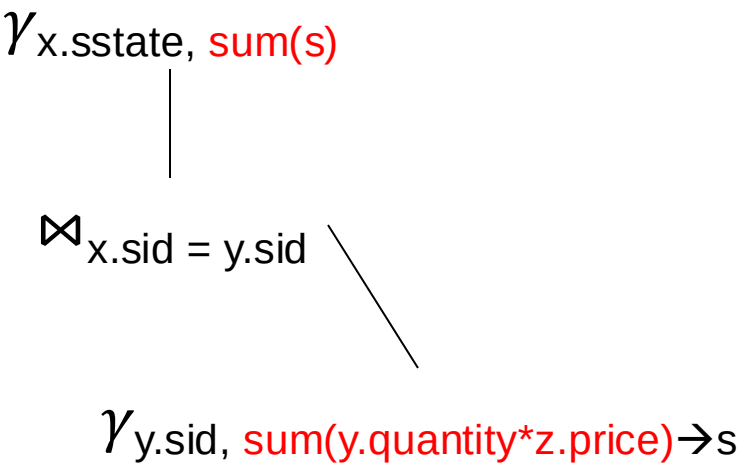




Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

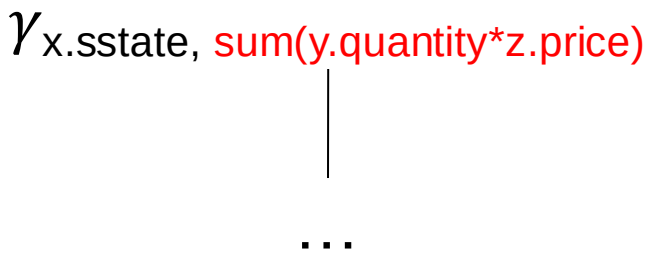
Aggregate Push-down

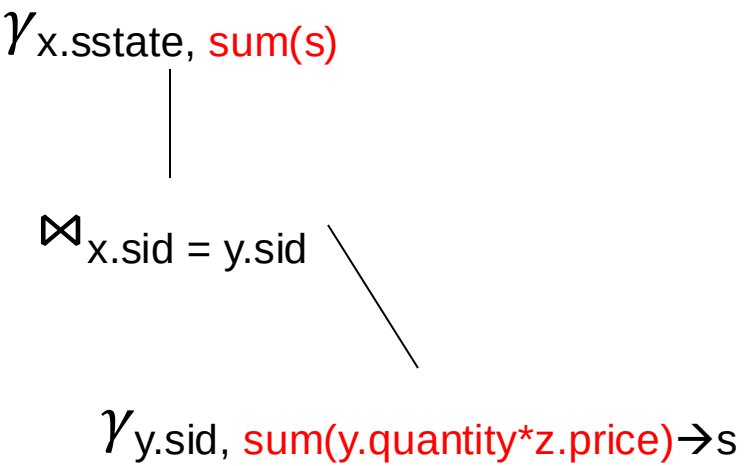


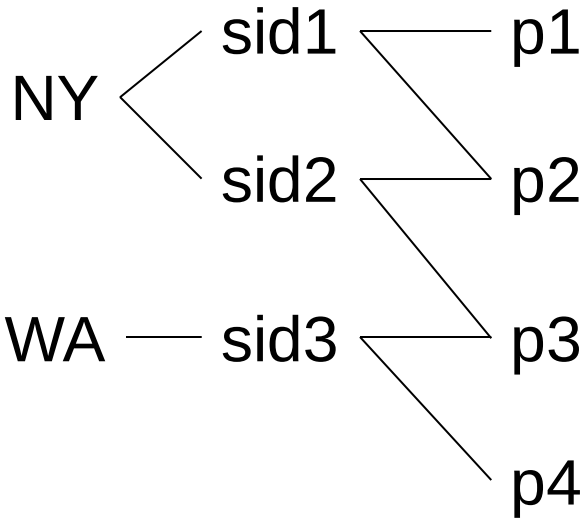


Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Aggregate Push-down

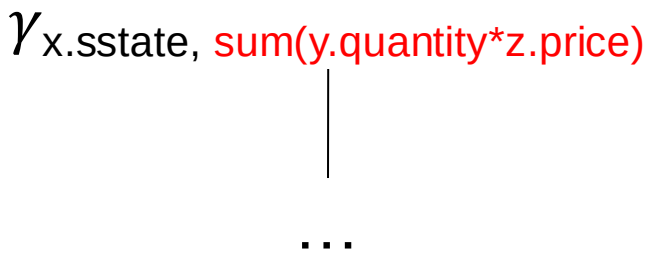


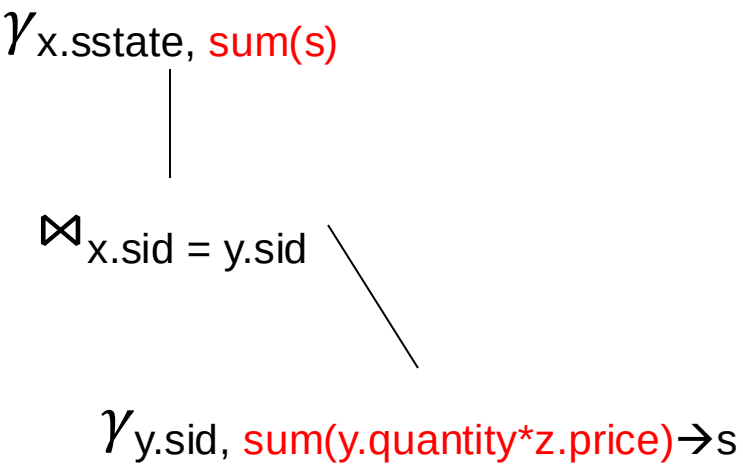


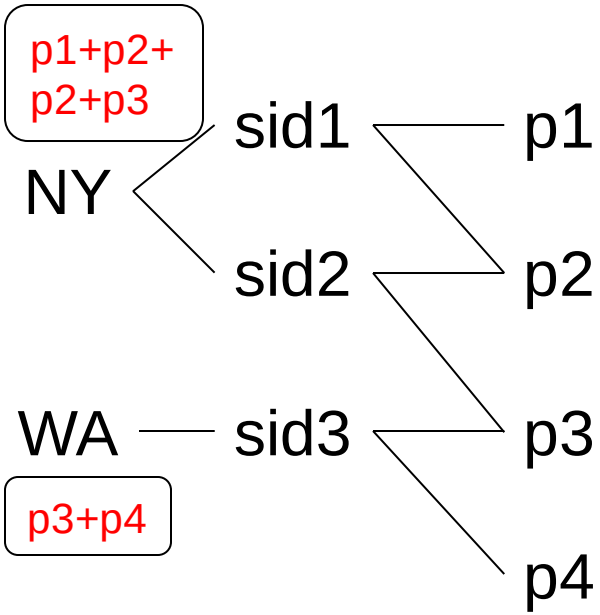


Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Aggregate Push-down

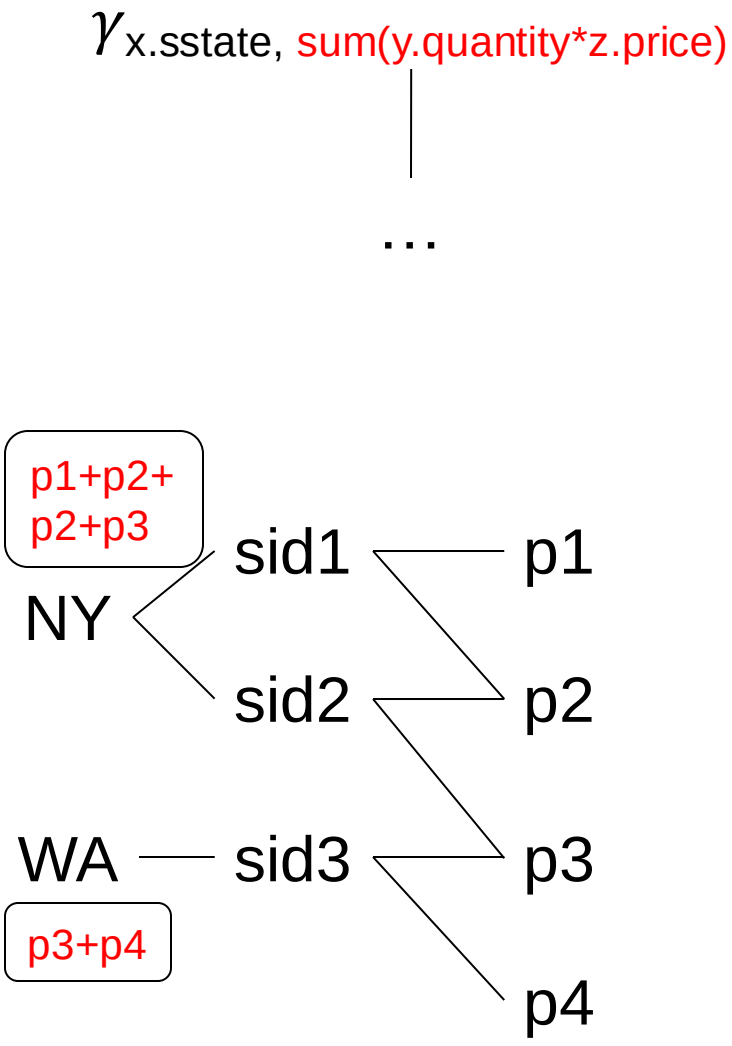


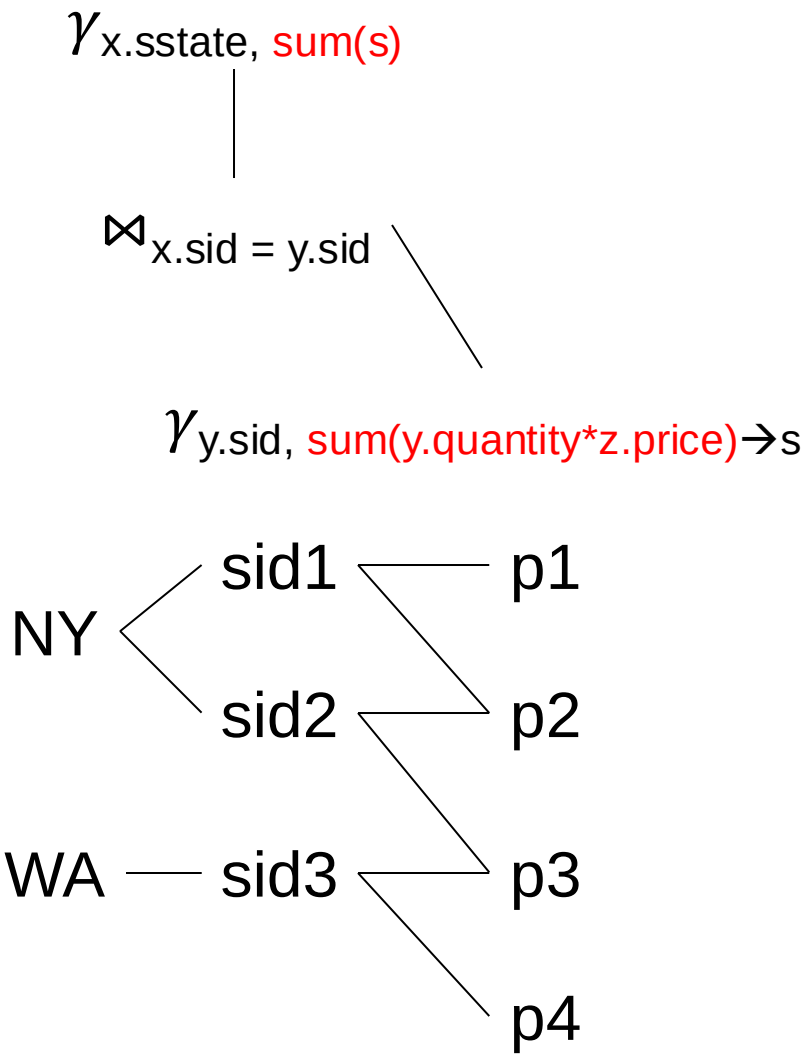




Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

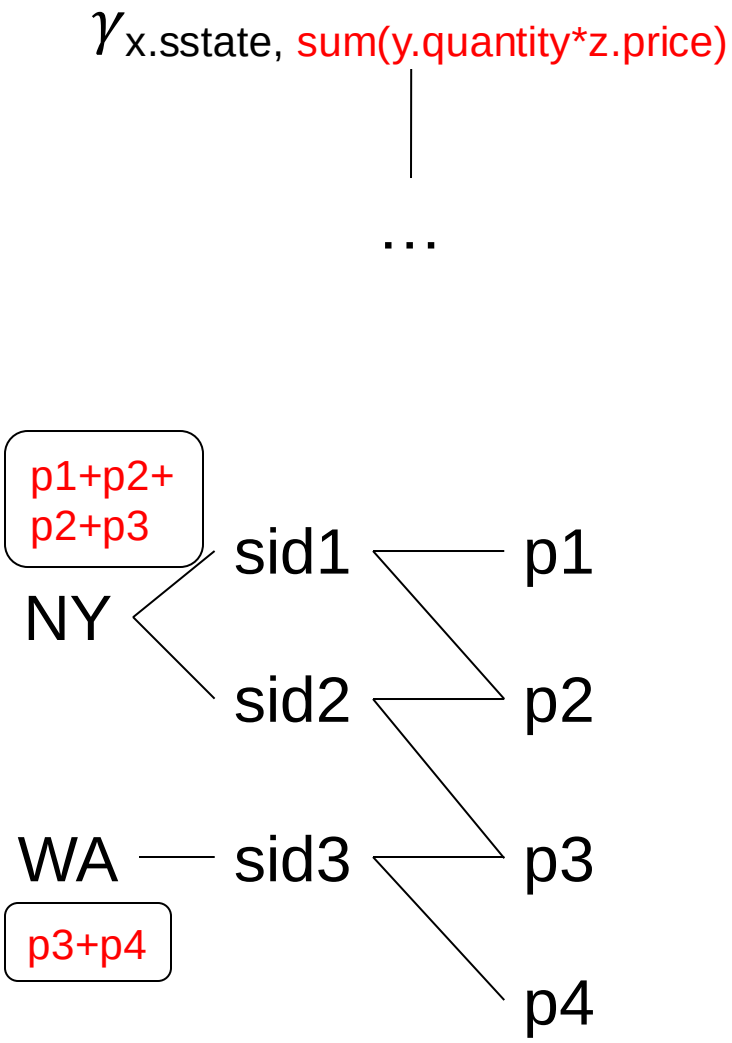
Aggregate Push-down

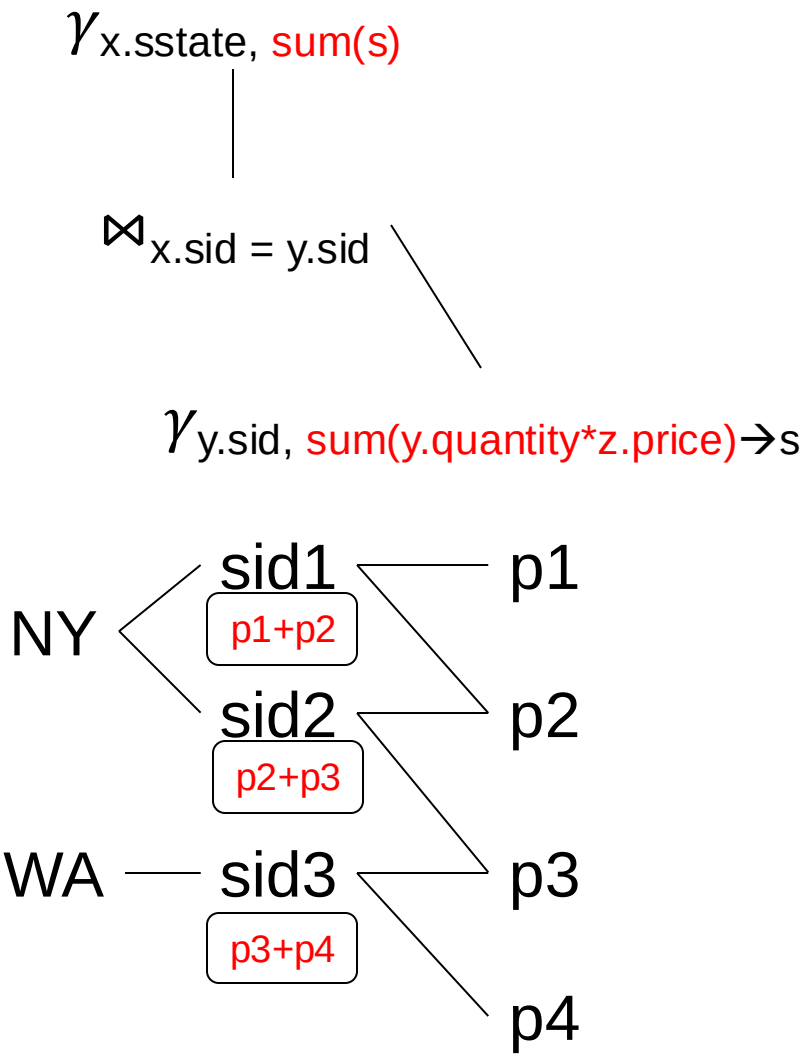




Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

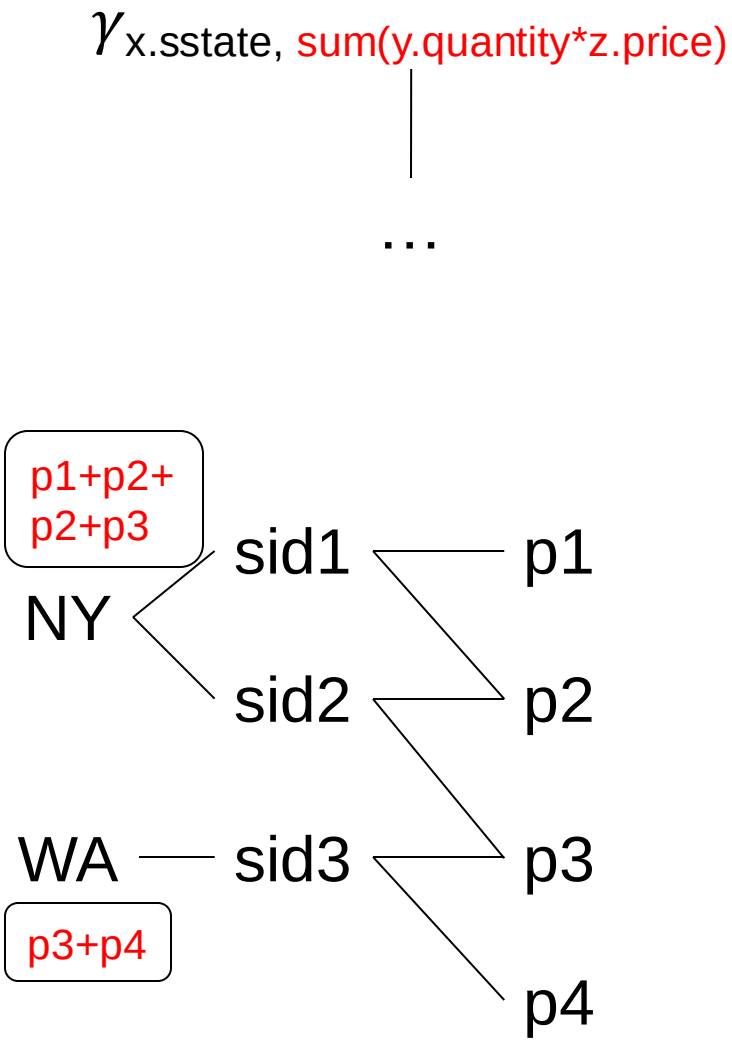
Aggregate Push-down

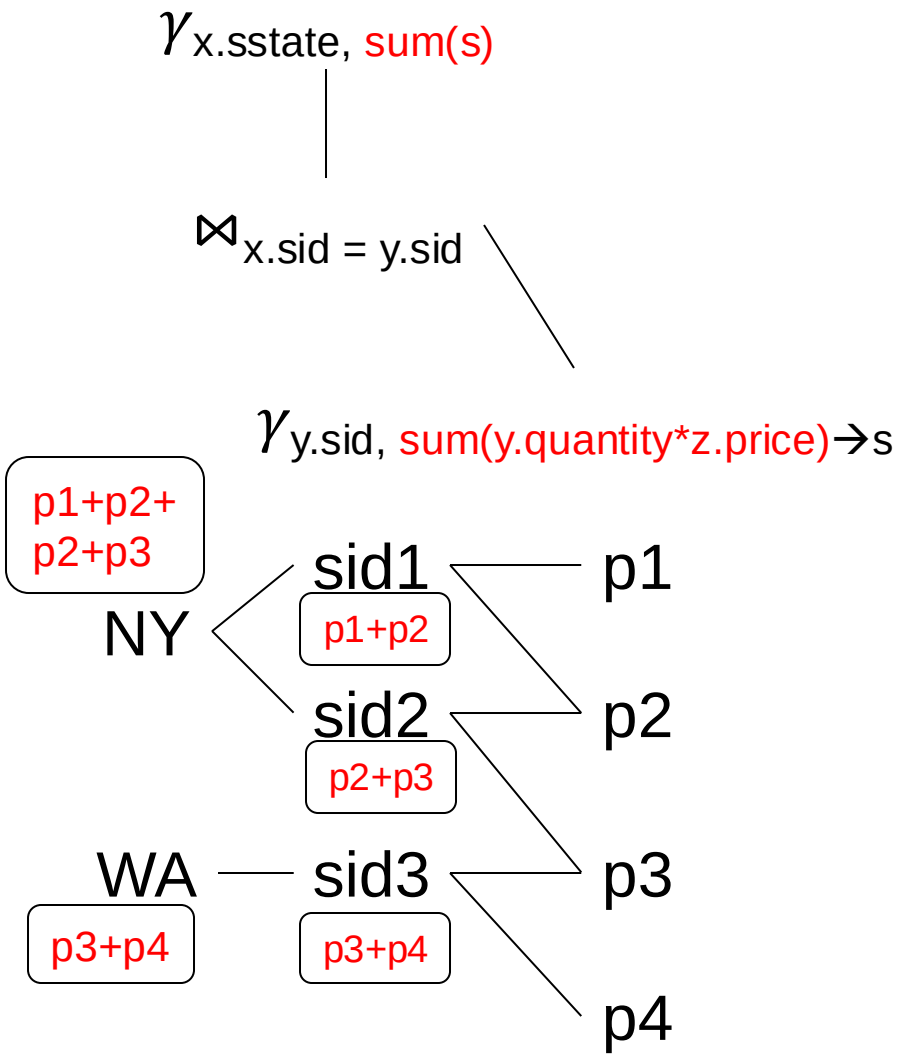




Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

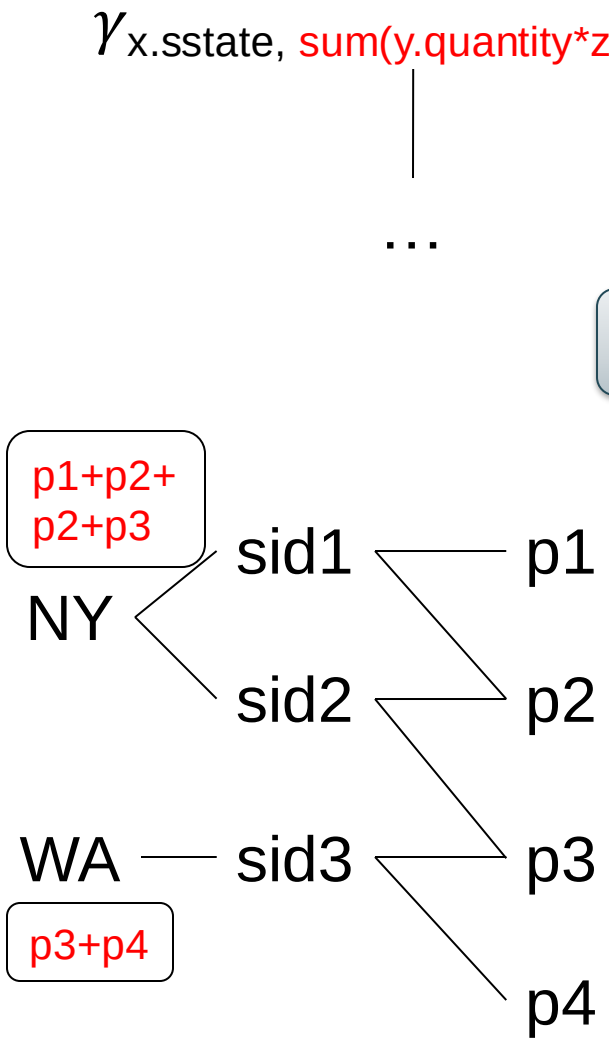
Aggregate Push-down



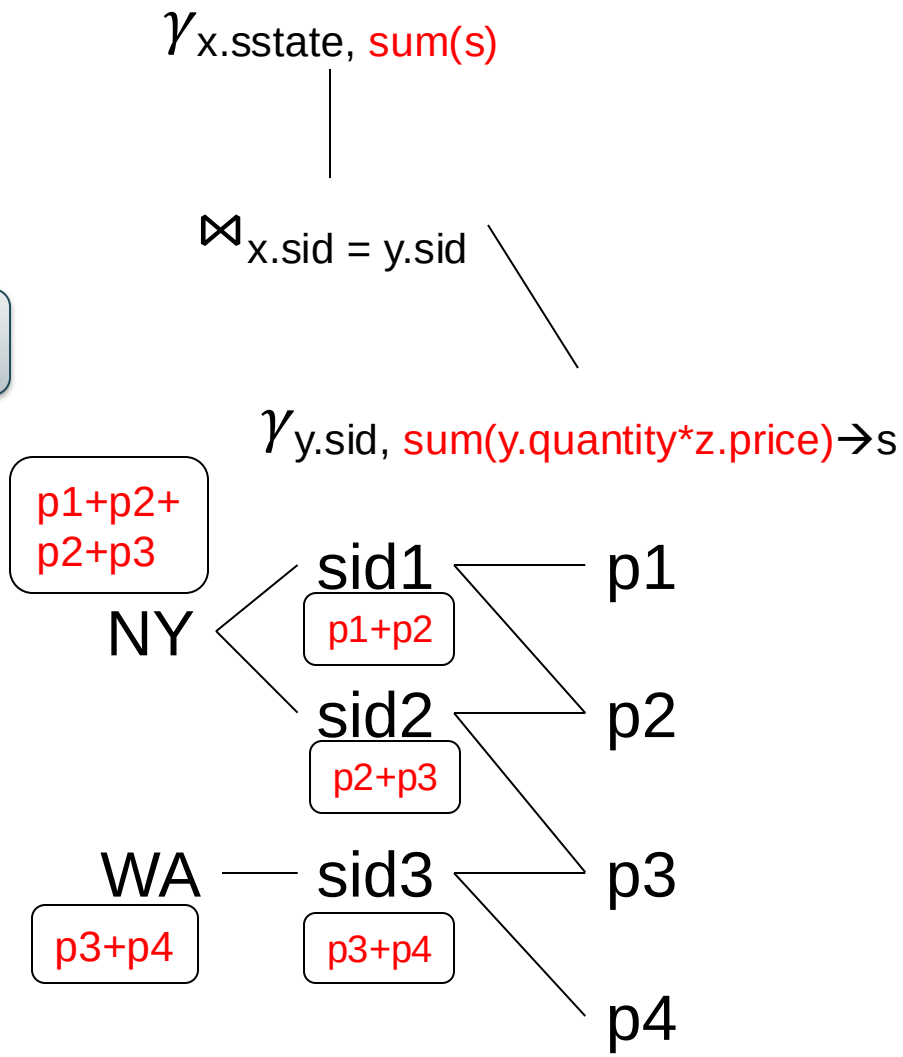


Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Aggregate Push-down



Same result



Discussion

- May need to push both left and right:

$$\gamma_{sum(R.A*S.B)}(R \bowtie_{R.X=S.Y} S) =$$
$$\gamma_{sum(U*V)}(\gamma_{X,sum(A)\rightarrow U}(R) \bowtie_{X=Y} \gamma_{Y,sum(B)\rightarrow V}(S))$$

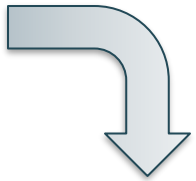
- Most DBMS support only some subset of aggregate pushdown

Next: a very useful rule

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
```

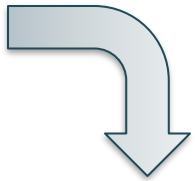


?

Supplier(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
```

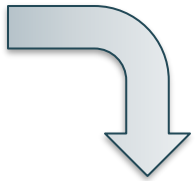


```
Select x.pno, x.quantity
From Supply x
```

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

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

Summary: Rules

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

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

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

$$\sigma_C(R \bowtie S) = R \bowtie (\sigma_C(S)) \quad \text{if C refers only to S}$$

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

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

Summary: Rules

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

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

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

$$\sigma_C(R \bowtie S) = R \bowtie (\sigma_C(S)) \quad \text{if C refers only to S}$$

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

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

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

Summary: Rules

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

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

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

$$\sigma_C(R \bowtie S) = R \bowtie (\sigma_C(S)) \qquad \text{if C refers only to S}$$

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

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

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

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

…

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

Cost Estimation

Overview

Two steps to compute the cost of a plan:

- Estimate the cardinality of each intermediate relation

Difficult

- Compute the cost of each operator, given the cardinalities

Easy

Cardinality Estimation

Maintain basis statistics on the database:

- Number of tuples (cardinality) T(R)
- Number of physical pages B(R)
- Indexes, number of keys in the index V(R,a)
- Histogram on single attribute (1d)
- Histogram on two attributes (2d)

Use them to estimate cardinality of each (sub)plan

Probability Space

SELECT * --- attrs A1, A2, …, An
FROM R_1, R_2, …, R_m
WHERE condition

Probability Space

```
SELECT *                          --- attrs A1, A2, …, An
FROM R1, R2, …, Rm
WHERE condition
```

Probability space:

- Outcomes: $R_1 \times R_2 \times \cdots \times R_m$
- Probability: $p(A_1, A_2, \ldots, A_n)$

Cardinality estimate: $Est = p(condition) \cdot T(R_1) \cdots T(R_m)$

Probability Space

```
SELECT *                          --- attrs A1, A2, …, An
FROM R1, R2, …, Rm
WHERE condition
```

Probability space:

- Outcomes: $R_1 \times R_2 \times \cdots \times R_m$
- Probability: $p(A_1, A_2, \ldots, A_n)$

Cardinality estimate: $Est = p(condition) \cdot T(R_1) \cdots T(R_m)$

Goal: compute $p(condition)$

Assumptions

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

Selectivity Factors

$p(pred)$ is called *selectivity factor* θ

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

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

Supplier(sid, sname, scity, sstate)

T(Supplier) = 100000
V(Supplier, scity) = 2000
V(Supplier, sstate) = 50

Example

```
SELECT *
FROM Supplier
WHERE scity = 'Seattle'
```

Est = ????

Supplier(sid, sname, scity, sstate)

T(Supplier) = 100000
V(Supplier, scity) = 2000
V(Supplier, sstate) = 50

Example

```
SELECT *
FROM Supplier
WHERE scity = 'Seattle'
```

Est = 1/2000 * 100000 = 50

Supplier(sid, sname, scity, sstate)

T(Supplier) = 100000
V(Supplier, scity) = 2000
V(Supplier, sstate) = 50

Example

```
SELECT *
FROM Supplier
WHERE scity = 'Seattle'
```

Est = 1/2000 * 100000 = 50

```
SELECT *
FROM Supplier
WHERE sstate = 'WA'
```

Est = 1/50 * 100000 = 2000

Supplier(sid, sname, scity, sstate)

T(Supplier) = 100000
V(Supplier, scity) = 2000
V(Supplier, sstate) = 50

Example

```
SELECT *
FROM Supplier
WHERE scity = 'Seattle'
```

Est = 1/2000 * 100000 = 50

```
SELECT *
FROM Supplier
WHERE sstate = 'WA'
```

Est = 1/50 * 100000 = 2000

```
SELECT *
FROM Supplier
WHERE scity = 'Seattle'
      and sstate = 'WA'
```

Est = ????

Supplier(sid, sname, scity, sstate)

T(Supplier) = 100000
V(Supplier, scity) = 2000
V(Supplier, sstate) = 50

Example

```
SELECT *
FROM Supplier
WHERE scity = 'Seattle'
```

Est = 1/2000 * 100000 = 50

```
SELECT *
FROM Supplier
WHERE sstate = 'WA'
```

Est = 1/50 * 100000 = 2000

Independence

```
SELECT *
FROM Supplier
WHERE scity = 'Seattle'
      and sstate = 'WA'
```

Est = 1/2000 * 1/50 * 100000
= 1

Supplier(sid, sname, scity, sstate)

T(Supplier) = 100000
V(Supplier, scity) = 2000
V(Supplier, sstate) = 50

Example

```
SELECT *
FROM Supplier
WHERE scity = 'Seattle'
```

Est = 1/2000 * 100000 = 50

```
SELECT *
FROM Supplier
WHERE sstate = 'WA'
```

Est = 1/50 * 100000 = 2000

```
SELECT *
FROM Supplier
WHERE scity = 'Seattle'
      and sstate = 'WA'
```

Independence

Est = 1/2000 * 1/50 * 100000
= 1

Very wrong

Why?

Selectivity Factors

Uniformity assumption

Equality:

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

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

Selectivity Factors

Uniformity assumption

Equality:

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

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

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

Range:

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

Selectivity Factors

Uniformity assumption

Equality:

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

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

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

Range:

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

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

Independence assumption

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

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

Selectivity Factors

Join

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

Why? Will explain next...

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

Selectivity Factors

Containment of values assumption:
if V(R,A) ≤ 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

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

Selectivity Factors

Assume V(R,A) ≤ V(S,B)

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

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

Selectivity Factors

Assume V(R,A) ≤ V(S,B)

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

In general:

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

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

T(Supplier) = 100000
V(Supplier, sid) = 100000

Example

T(Supply) = 5000000
V(Supply, sid) = 80000

```
SELECT *
FROM Supplier x, Supply y
WHERE x.sid = y.sid
```

What is the exact output size?

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

T(Supplier) = 100000
V(Supplier, sid) = 100000

Example

T(Supply) = 5000000
V(Supply, sid) = 80000

```
SELECT *
FROM Supplier x, Supply y
WHERE x.sid = y.sid
```

What is the exact output size?

T(Supply)=5000000

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

T(Supplier) = 100000
V(Supplier, sid) = 100000

Example

T(Supply) = 5000000
V(Supply, sid) = 80000

```
SELECT *
FROM Supplier x, Supply y
WHERE x.sid = y.sid
```

What is the exact output size?

T(Supply)=5000000

$$Est(Supplier \bowtie Supply) = \frac{T(Supplier)T(Supply)}{\max(V(Supplier, sid), V(Supply, sid))}$$

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

T(Supplier) = 100000
V(Supplier, sid) = 100000

Example

T(Supply) = 5000000
V(Supply, sid) = 80000

```
SELECT *
FROM Supplier x, Supply y
WHERE x.sid = y.sid
```

What is the exact output size?

T(Supply)=5000000

$$Est(Supplier \bowtie Supply) = \frac{T(Supplier)T(Supply)}{\max(V(Supplier, sid), V(Supply, sid))}$$

Always larger

Why?

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

T(Supplier) = 100000
V(Supplier, sid) = 100000

Example

T(Supply) = 5000000
V(Supply, sid) = 80000

```
SELECT *
FROM Supplier x, Supply y
WHERE x.sid = y.sid
```

What is the exact output size?

T(Supply)=5000000

$$Est(Supplier \bowtie Supply) = \frac{T(Supplier)T(Supply)}{\max(V(Supplier, sid), V(Supply, sid))}$$

Always larger

Why?

$$= \frac{T(Supplier)T(Supply)}{V(Supplier, sid)}$$

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

T(Supplier) = 100000
V(Supplier, sid) = 100000

Example

T(Supply) = 5000000
V(Supply, sid) = 80000

```
SELECT *
FROM Supplier x, Supply y
WHERE x.sid = y.sid
```

What is the exact output size?

T(Supply)=5000000

$$Est(Supplier \bowtie Supply) = \frac{T(Supplier)T(Supply)}{\max(V(Supplier, sid), V(Supply, sid))}$$

$$= \frac{T(Supplier)T(Supply)}{V(Supplier, sid)} = T(Supply)$$

Always larger

Why?

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

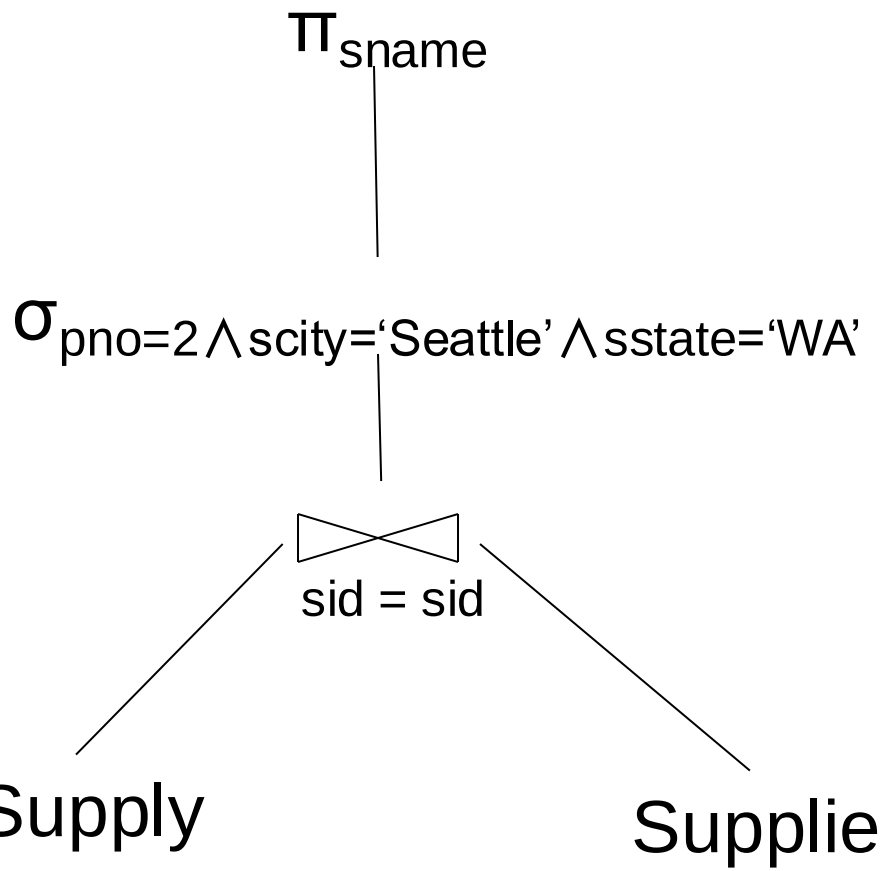
Computing the Cost of a Plan

- Estimate cardinalities bottom-up

- Estimate cost by using estimated cardinalities

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'
```

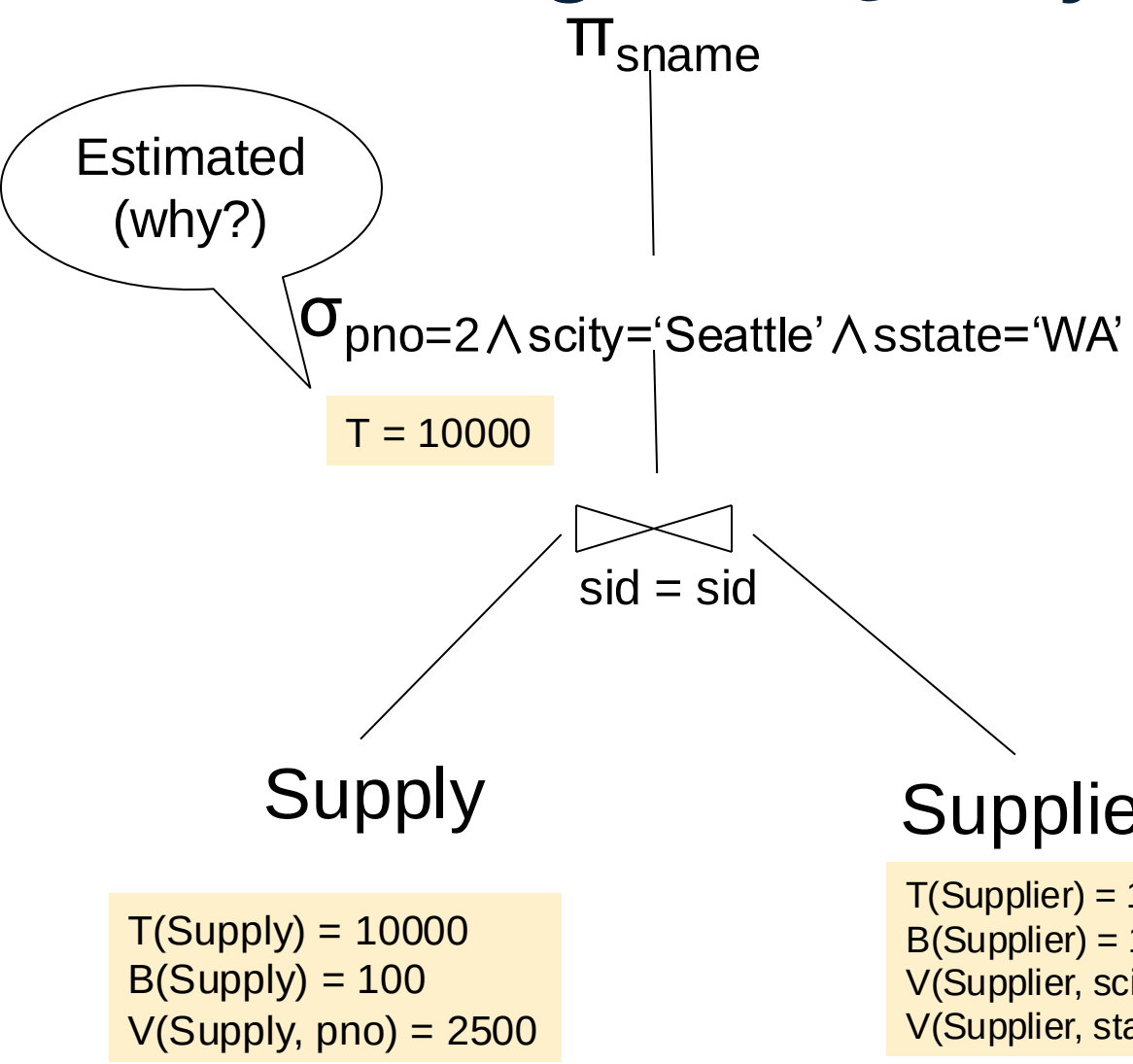
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

Logical Query Plan 1

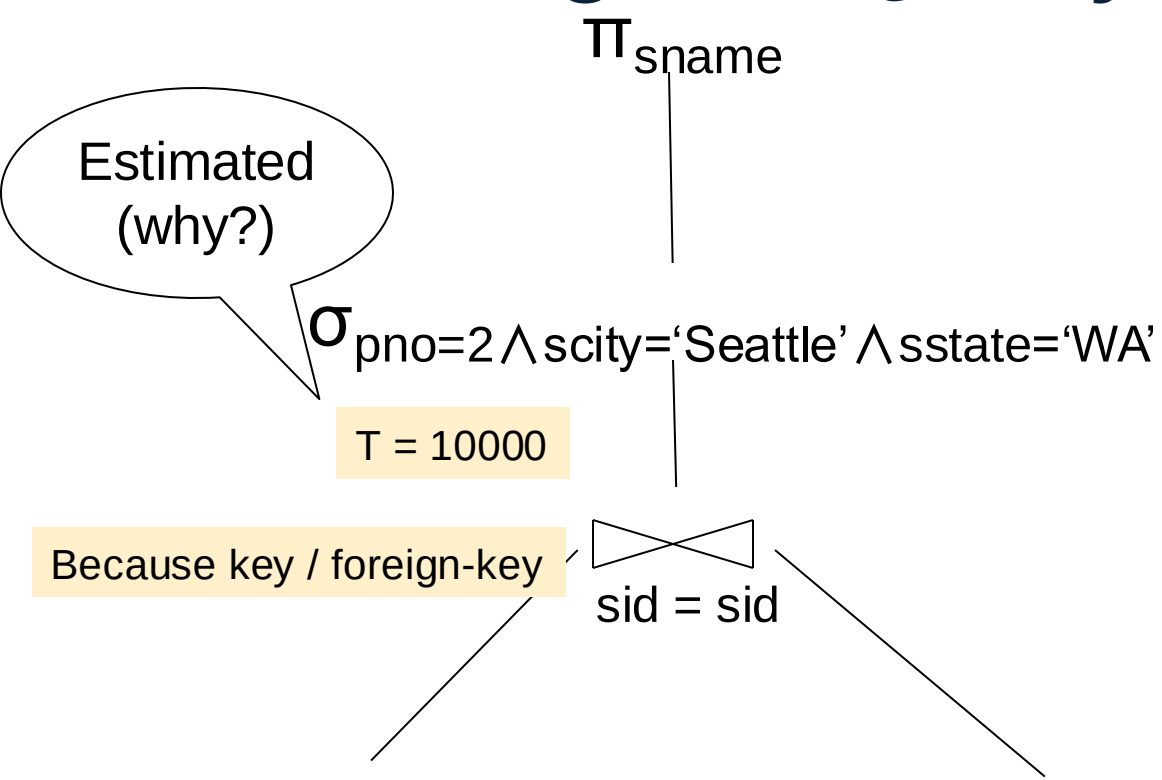



```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

M=11

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'
```

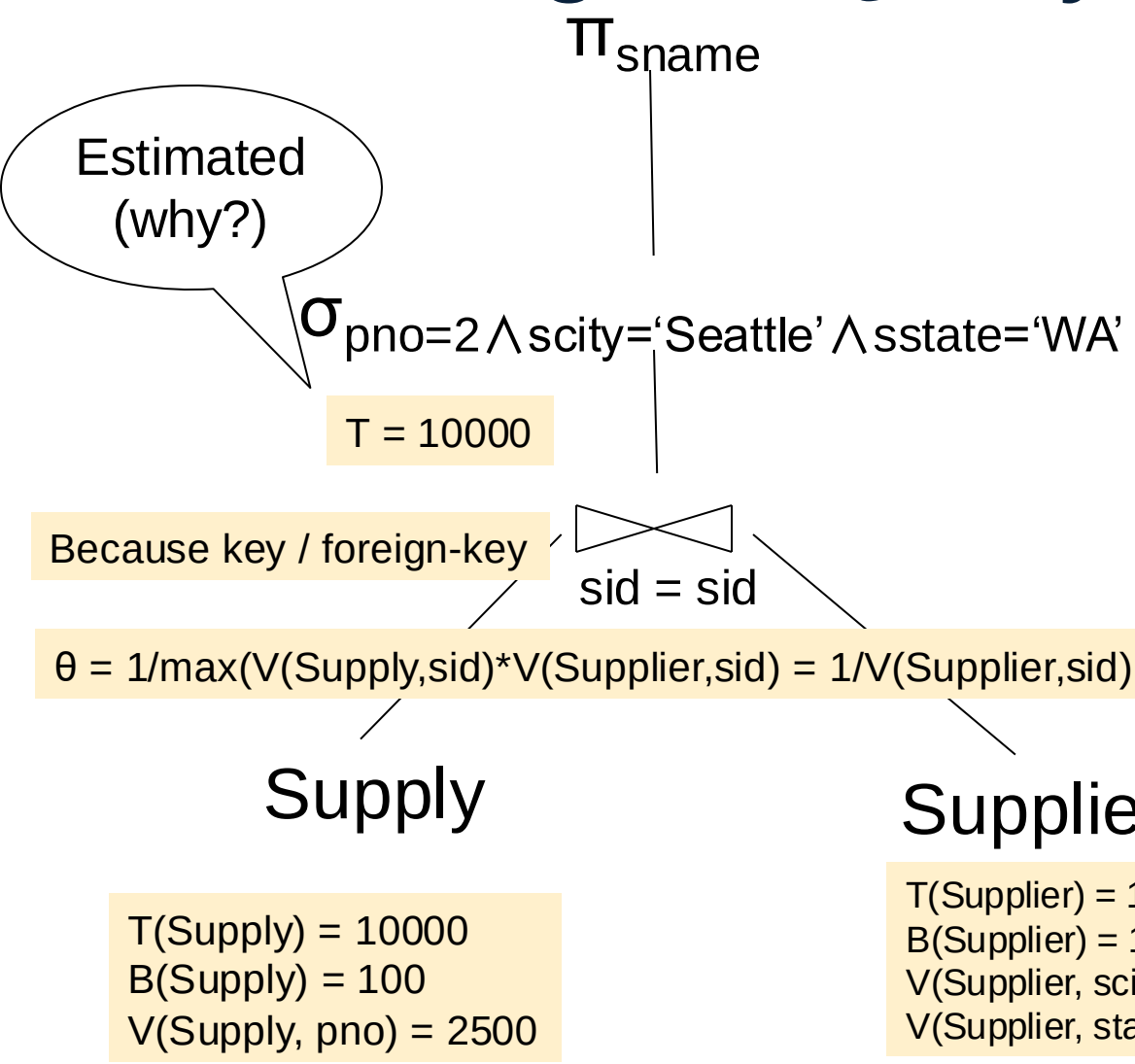
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

Logical Query Plan 1

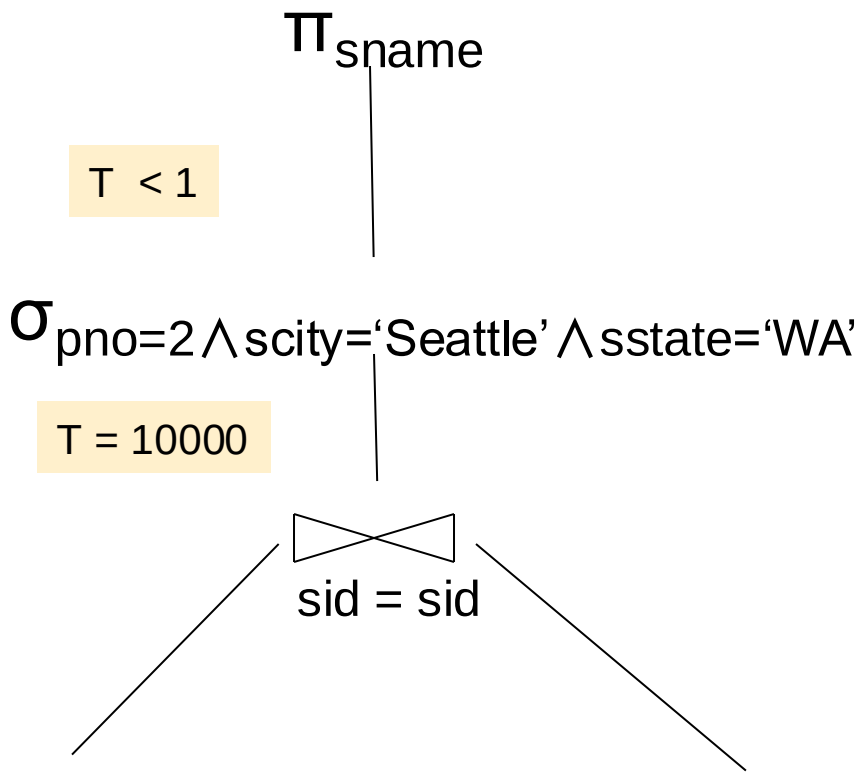



```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

M=11

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'
```

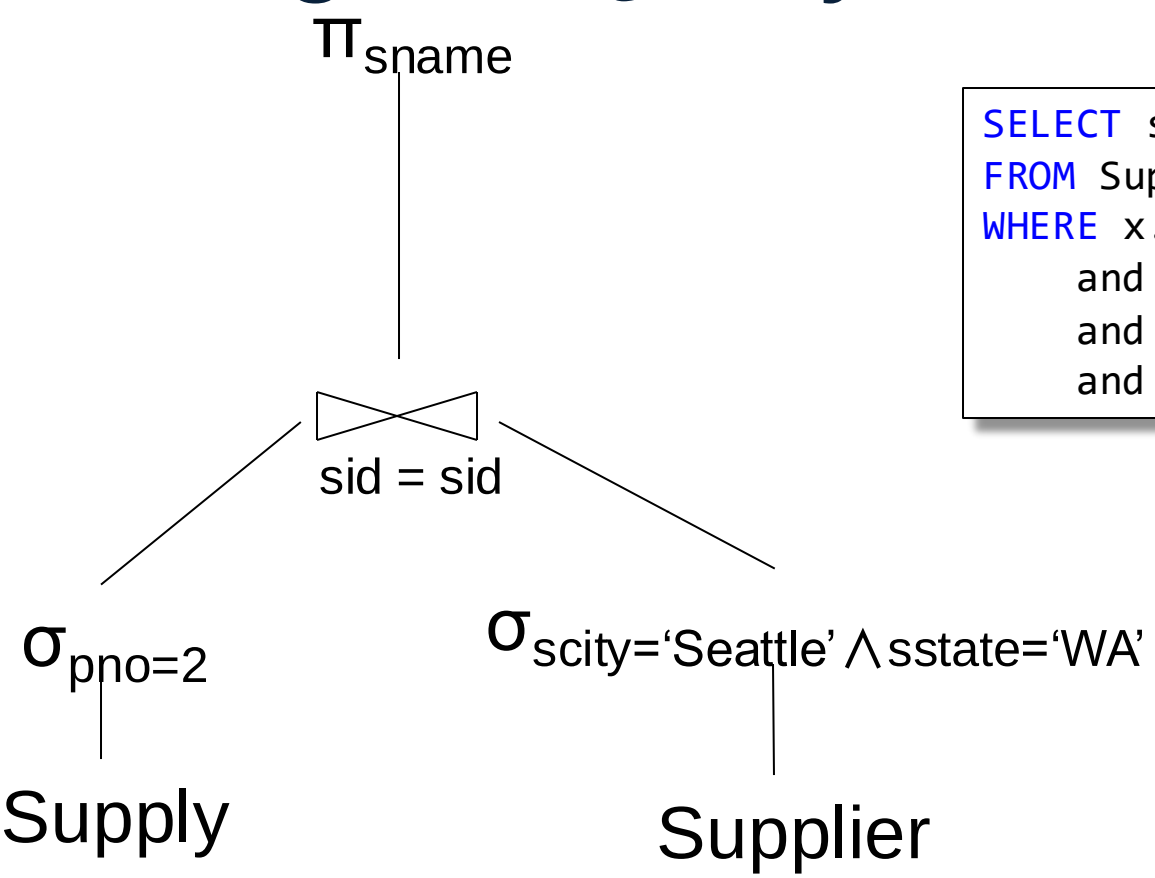
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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'
```

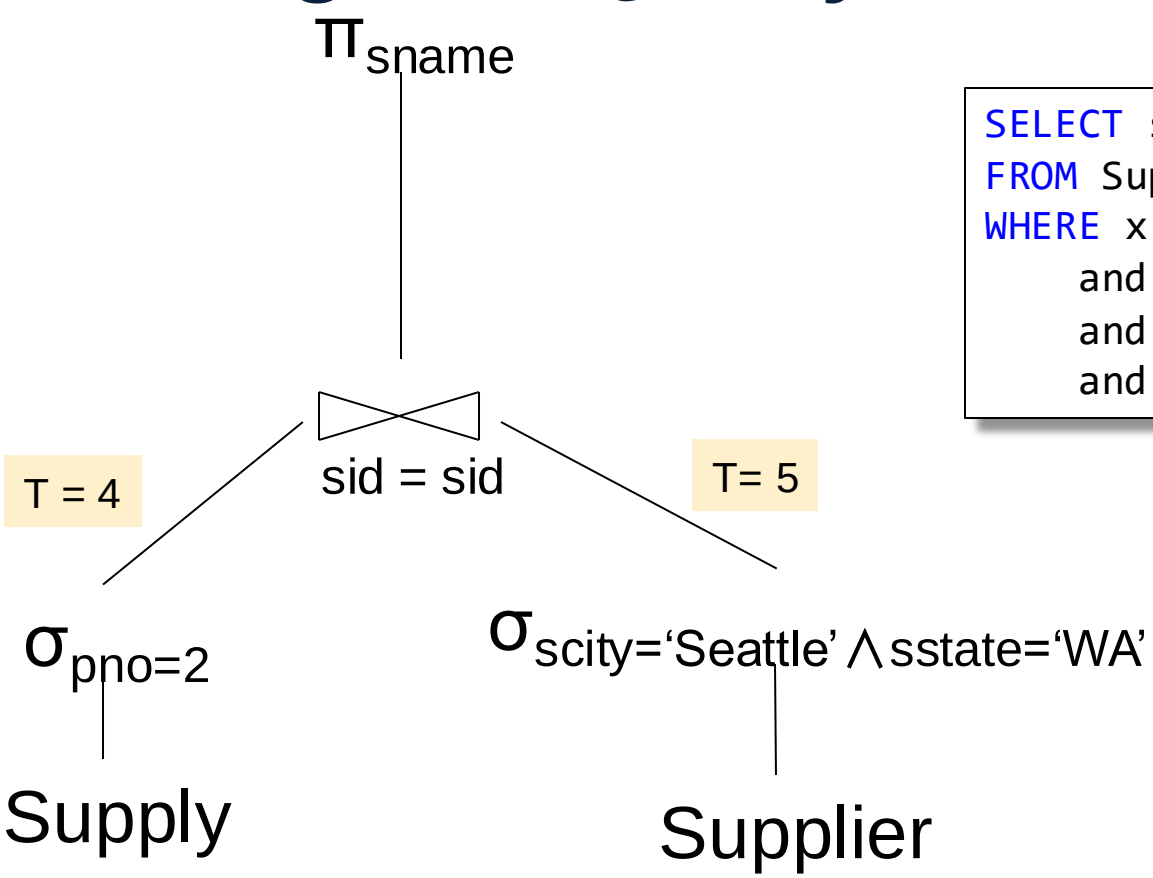
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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'
```

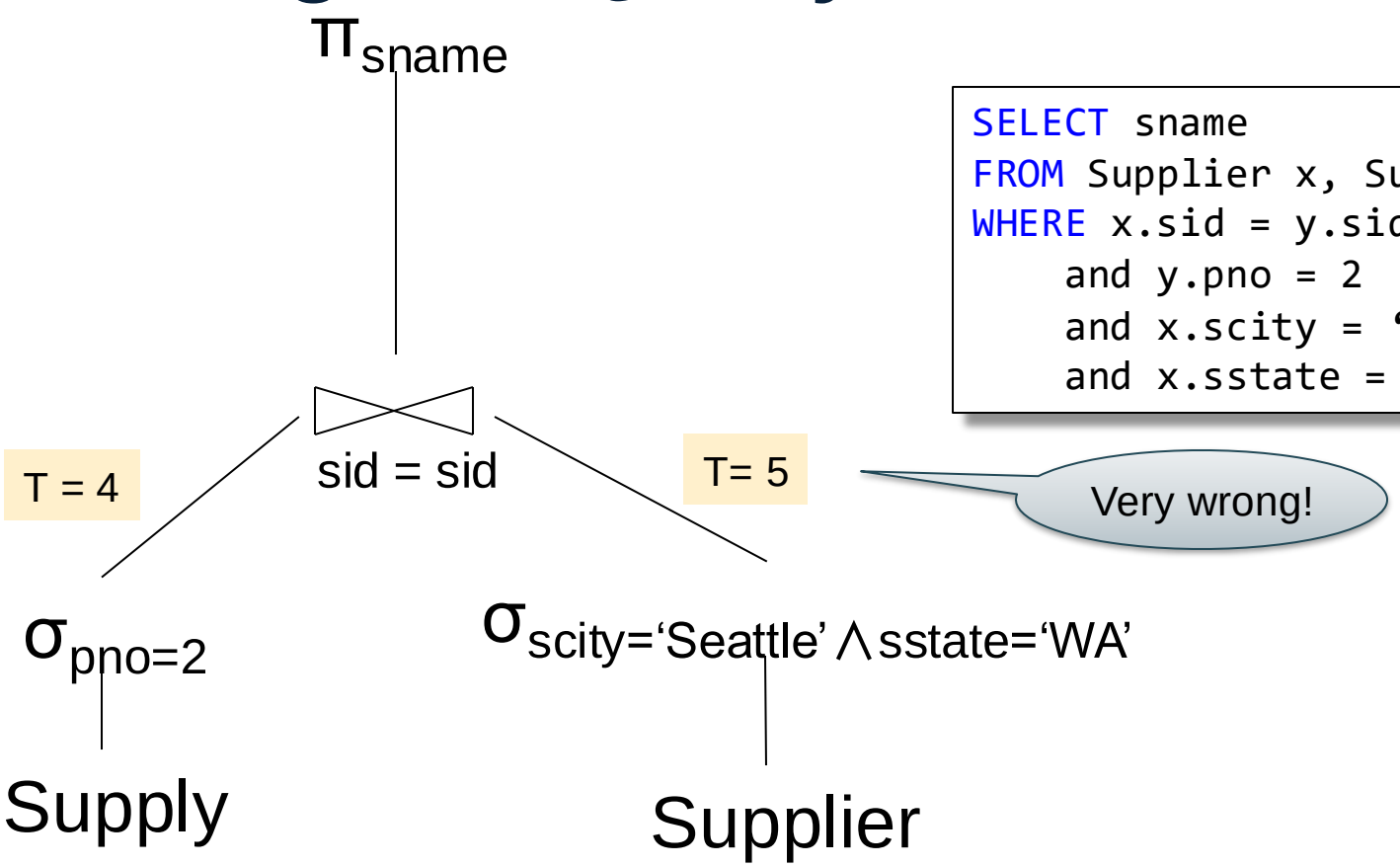
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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'
```

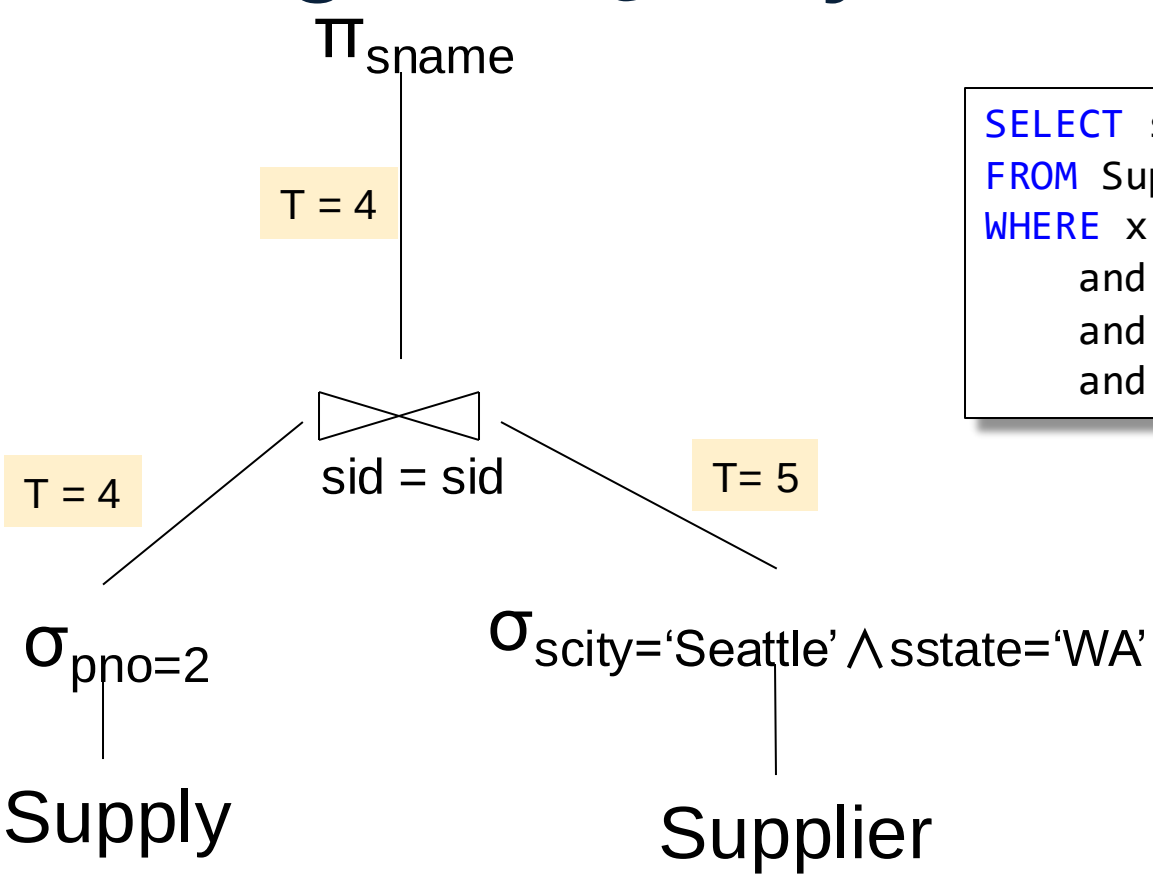
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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'
```

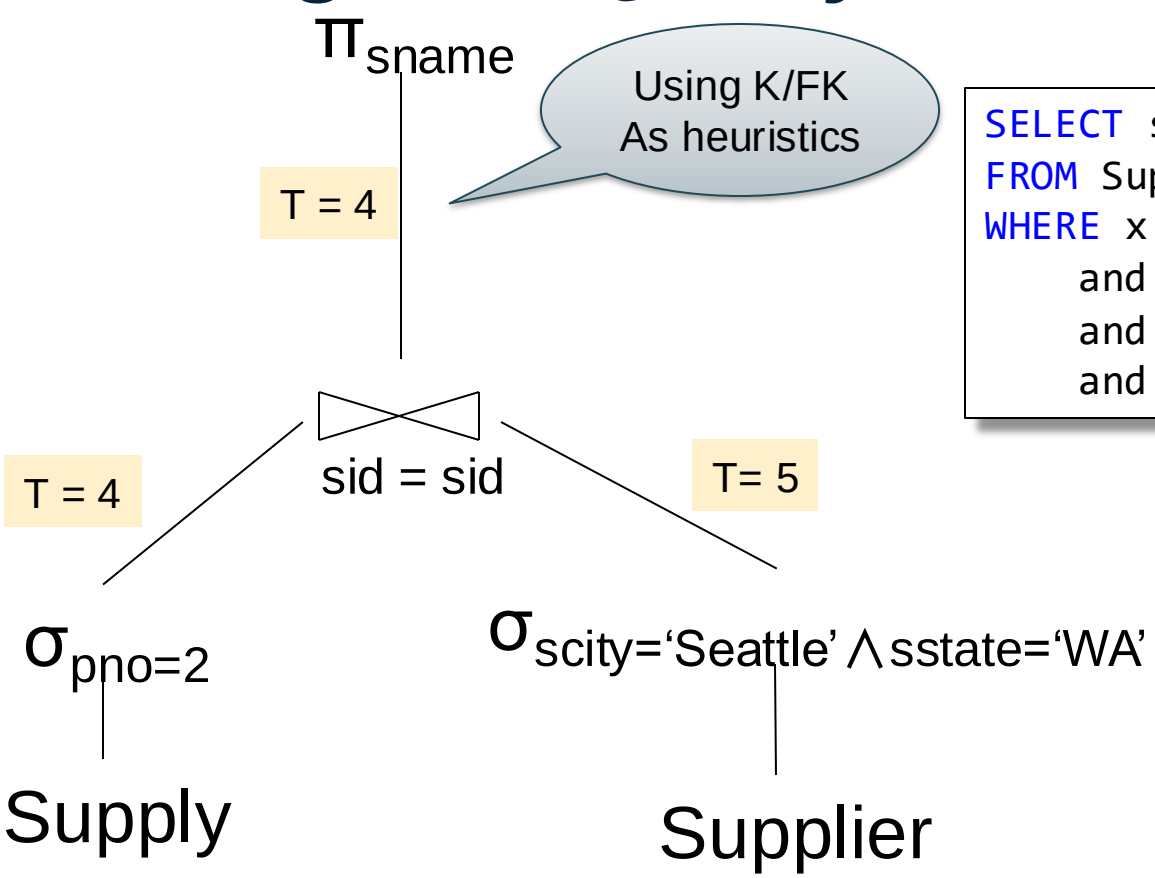
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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'
```

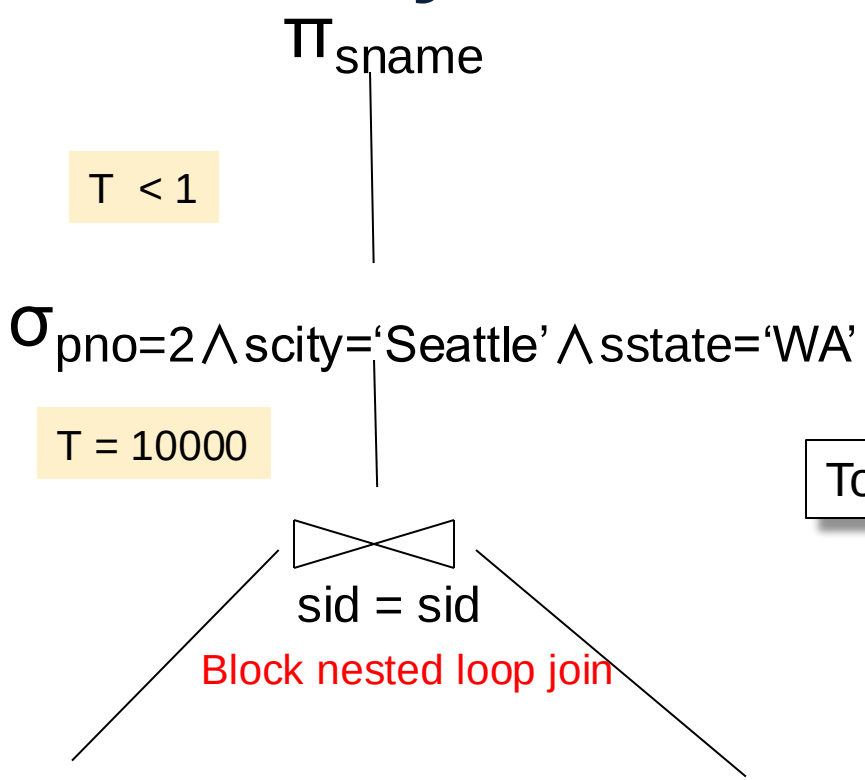
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

Physical Plan 1



Will compute only the I/O cost

Total cost:

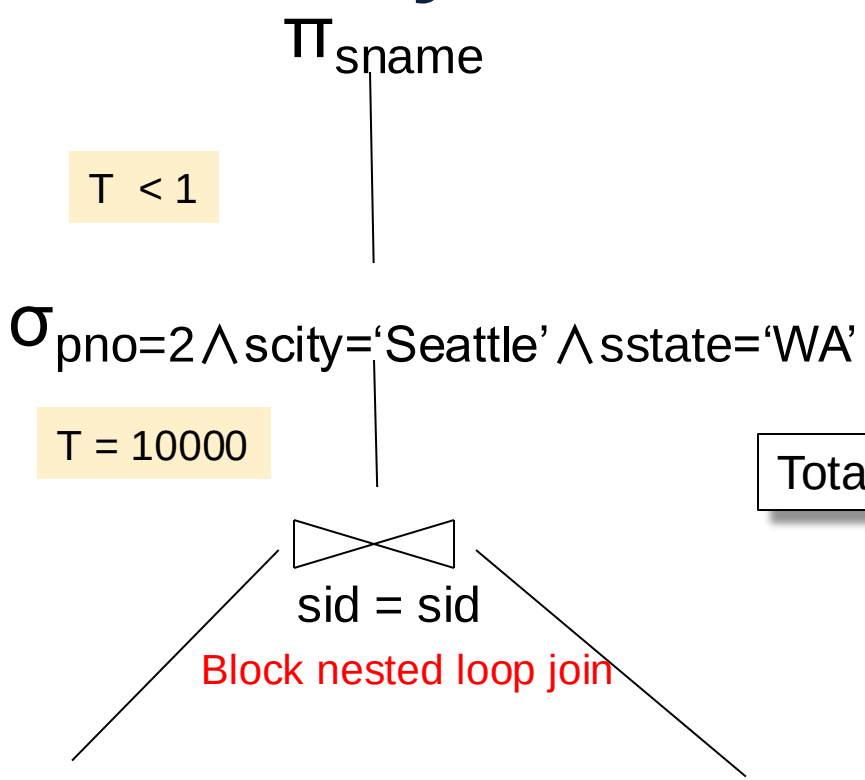
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

Physical Plan 1



Will compute only the I/O cost

Total cost: 100+100*100/10 = 1100

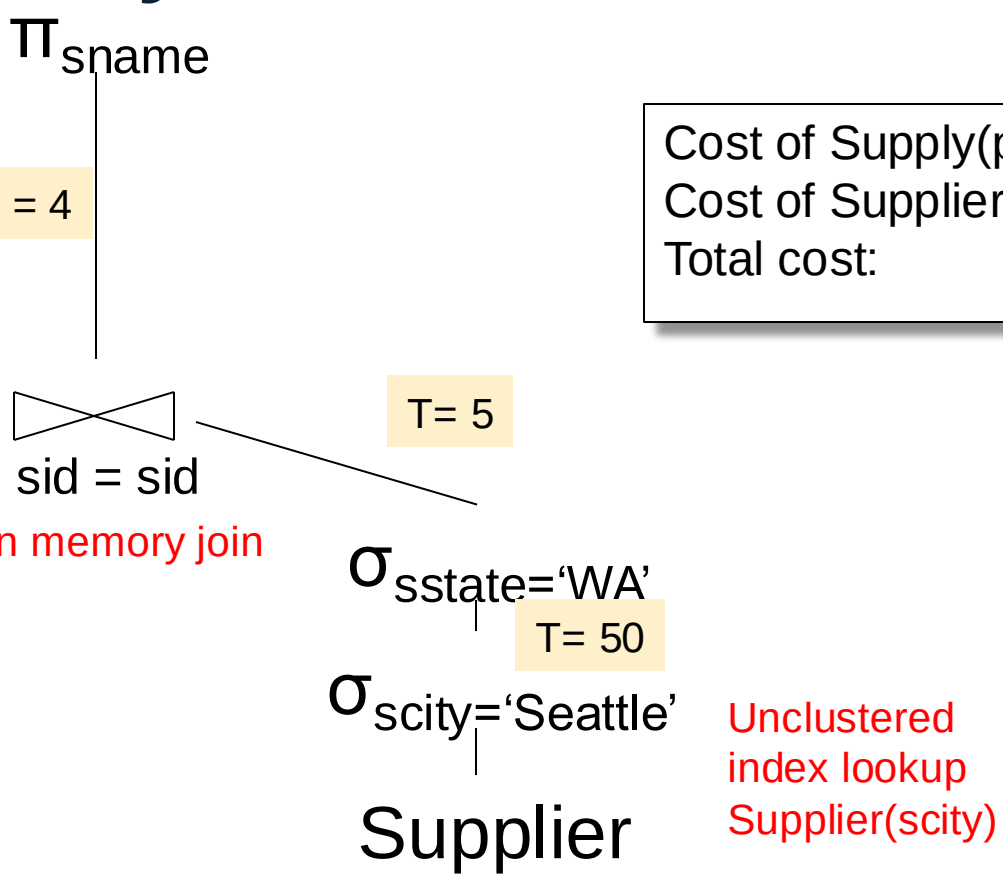
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

Physical Plan 2



Only the I/O cost

Cost of Supply(pno) =
Cost of Supplier(scity) =
Total cost:

Unclustered index lookup Supply(pno)

Unclustered index lookup Supplier(scity)

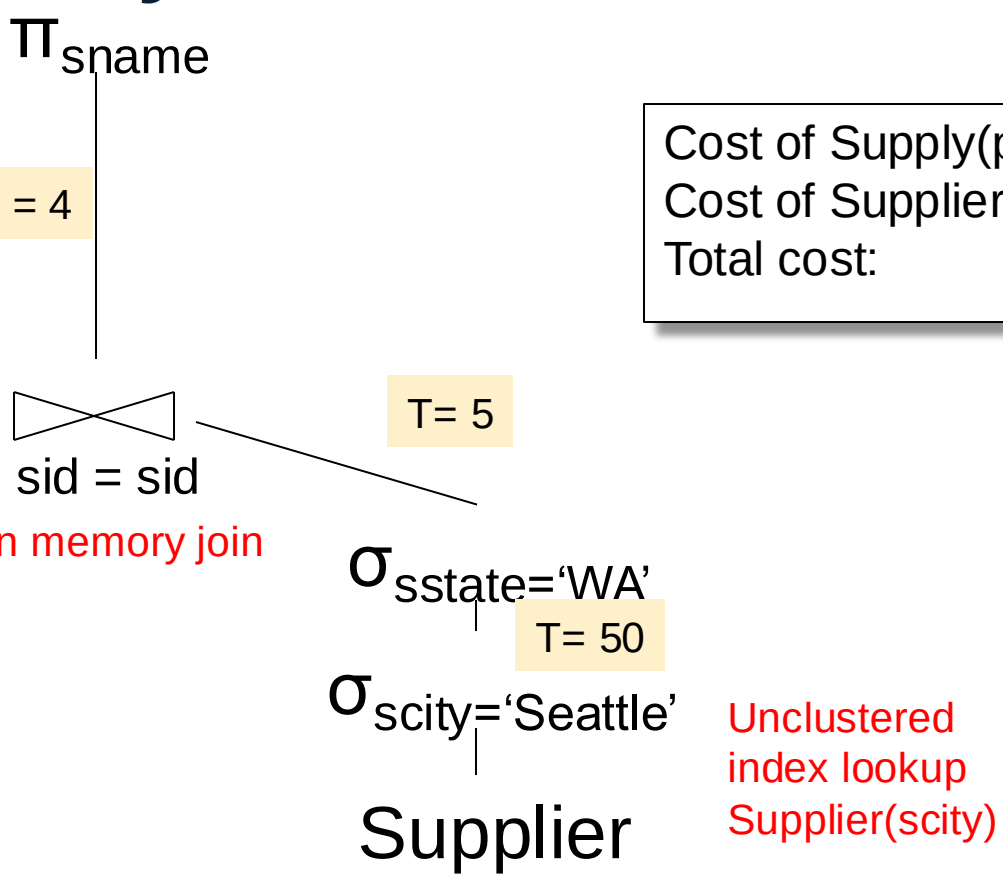
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

Physical Plan 2



Only the I/O cost

Cost of Supply(pno) = 4
Cost of Supplier(scity) =
Total cost:

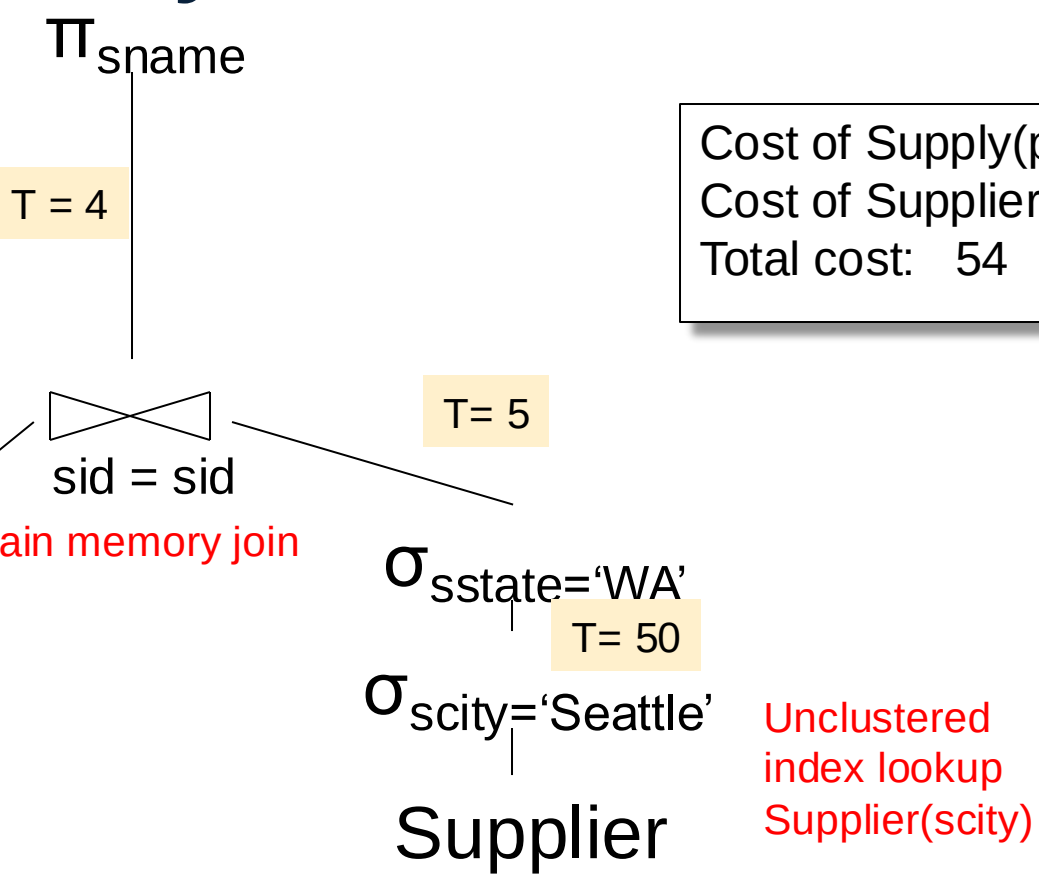
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

Physical Plan 2



Only the I/O cost

Cost of Supply(pno) = 4
Cost of Supplier(scity) = 50
Total cost: 54

Unclustered index lookup Supply(pno)

Unclustered index lookup Supplier(scity)

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

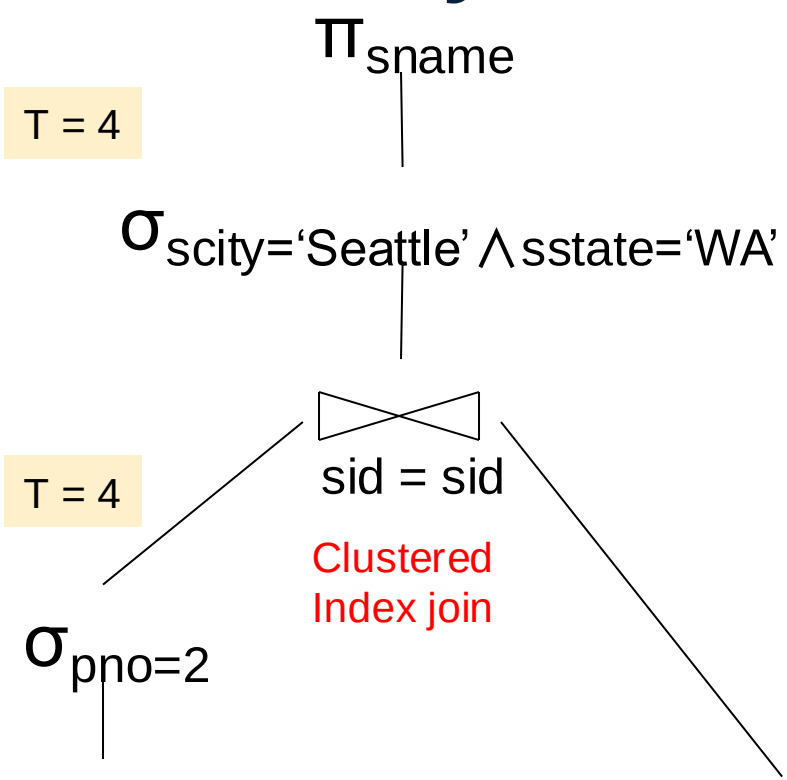
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

Physical Plan 3

Only the I/O cost



Cost of Supply(pno) =
Cost of Index join =
Total cost:

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

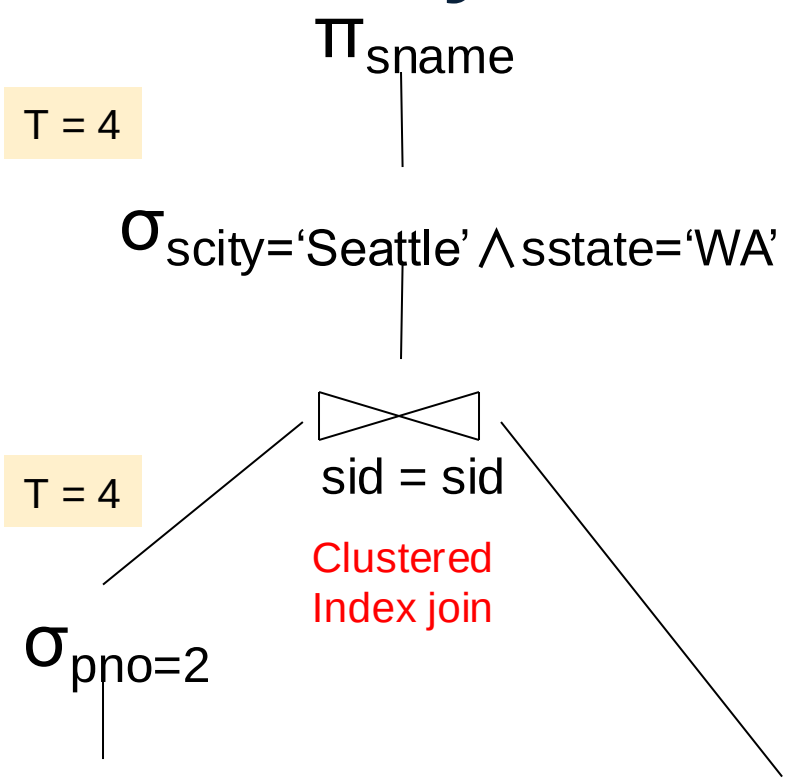
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

Physical Plan 3

Only the I/O cost



Cost of Supply(pno) = 4
Cost of Index join =
Total cost:

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

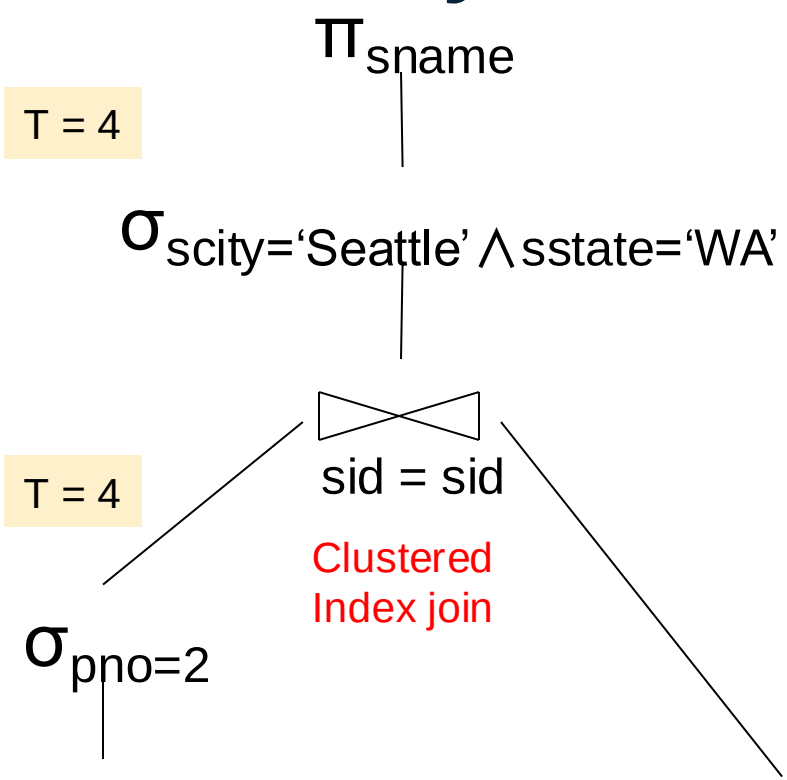
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

Physical Plan 3

Only the I/O cost



Cost of Supply(pno) = 4
Cost of Index join = 4
Total cost: 8

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Histograms

- 1d Histograms mitigate uniformity assumption
- 2d Histograms mitigate independence assumption

1d-Histograms

- Histogram on R.A refines T(R), V(R,A)
- Each bucket: T(bucket), V(bucket, A)

1d-Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50

$\sigma_{age=48}$(Empolyee) = ?

1d-Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50

$\sigma_{age=48}$(Empolyee) = ?

Estimate: T(Employee) / V(Employee,age) = 500

1d-Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50

$\sigma_{age=48}$(Empolyee) = ?

Estimate: T(Employee) / V(Employee,age) = 500

Age:	0..20	20..29	30-39	40-49	50-59	> 60
T =	200	800	5000	12000	6500	500

1d-Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50

$\sigma_{age=48}$(Empolyee) = ?

Estimate: T(Employee) / V(Employee,age) = 500

Age:	0..20	20..29	30-39	40-49	50-59	> 60
T =	200	800	5000	12000	6500	500

Assume V = 10

1d-Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50

$\sigma_{age=48}$(Empolyee) = ?

~~Estimate: T(Employee) / V(Employee,age) = 500~~

Age:	0..20	20..29	30-39	40-49	50-59	> 60
T =	200	800	5000	12000	6500	500

Assume V = 10

Estimate: 12000/10 = 1200

1d-Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50

$\sigma_{age=48}$(Empolyee) = ?

~~Estimate: T(Employee) / V(Employee,age) = 500~~

Age:	0..20	20..29	30-39	40-49	50-59	> 60
T =	200	800	5000	12000	6500	500
V =	3	10	7	6	5	4

Estimate: 12000/10 = 1200

1d-Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50

$\sigma_{age=48}$(Empolyee) = ?

~~Estimate: T(Employee) / V(Employee,age) = 500~~

Age:	0..20	20..29	30-39	40-49	50-59	> 60
T =	200	800	5000	12000	6500	500
V =	7	10	10	6	9	8

Estimate: ~~12000/10 = 1200~~ 12000/6 = 2000

Types of 1d-Histograms

- Eq-Width
- Eq-Depth
- Compressed: store outliers separately
- V-Optimal histograms

Employee(ssn, name, age)

Types of 1d-Histograms

Eq-width:

Age:	0..20	20..29	30-39	40-49	50-59	> 60
Tuples	200	800	5000	12000	6500	500

Eq-depth:

Age:	0..32	33..41	42-46	47-52	53-58	> 60
Tuples	1800	2000	2100	2200	1900	1800

Compressed: store separately highly frequent values: (48,1900)

V-Optimal Histogram

- $domain(A) = \{v_1, \ldots, v_n\}$
- Have budget of b+1 buckets for R.A:

$$-\infty < d_1 < d_2 < \cdots < d_b < \infty$$

- Choose $d_1, \ldots, d_b$ to minimize error of all queries $\sigma_{A=v}(R)$, for $v \in Domain(A)$

V-Optimal Histogram

- Error:

$$\sum_{v \in Domain(A)} \left(|\sigma_{A=v}(R)| - est_{Hist}\left(\sigma_{A=v}(R)\right) \right)^2$$

- Bucket boundaries = $\text{argmin}_{\text{Hist}}$(Error)

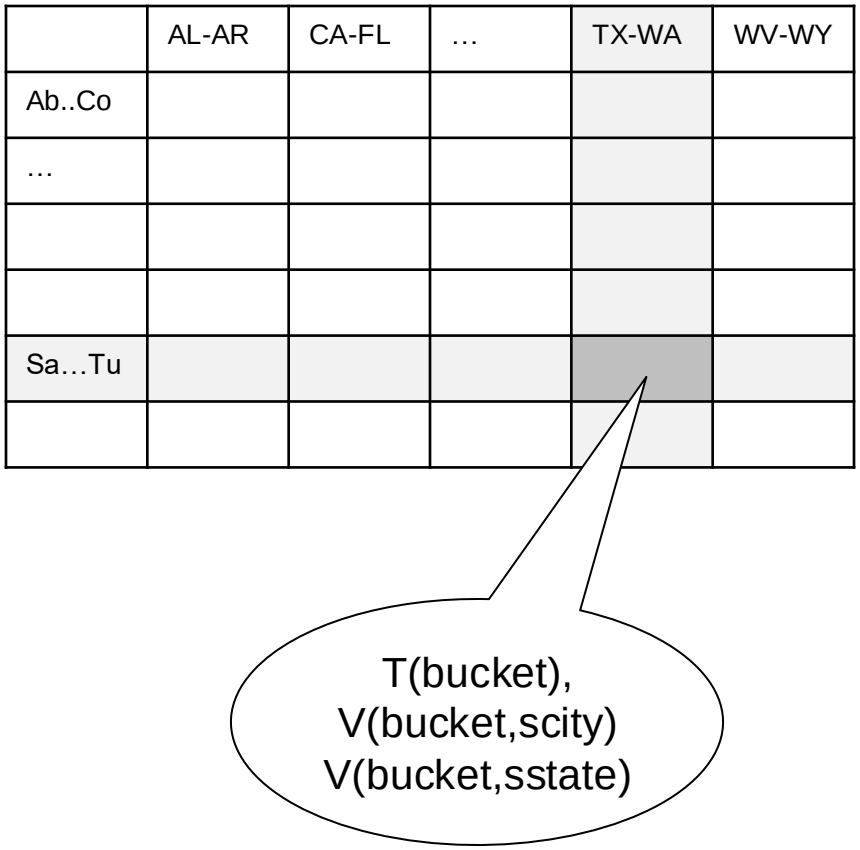
- Dynamic programming

Discussion: 1d-Histograms

- All systems support some forms
- Histograms need to be small to reside in main memory: 1000 – 10000 buckets
- Recomputed periodically, often from samples. E.g Postgres ANALYSE

2d-Histograms

```
SELECT *
FROM Supplier
WHERE scity = 'Seattle'
      and sstate = 'WA'
```



Independence only needed within the bucket

Issues with 2d-Histograms

- Limited # of dividers: $\sqrt{1000} - \sqrt{10000}$
- Too many 2d-histograms: n(n-1)/2
- Problem: how do we estimate P(A,B,C) given all 1d and 2d-histograms?

- Few systems support 2d-histograms

Paper Discussion

How Good Are Query Optimizers, Really? VLDB'2015

Questions in the paper

- How good are cardinality estimators?
- How important are they for the optimizer?
- How large does the plan space need to be?

Cardinality Estimators

- Standard database benchmark: TPC-H
- They designed a new benchmark. Why?

Cardinality Estimators

- Standard database benchmark: TPC-H
- They designed a new benchmark. Why?
- Because TPC-H is synthetically generated, unrealistically uniform

Cardinality Estimators

What type of queries are in IMDB/JOB?

Cardinality Estimators

What type of queries are in IMDB/JOB?

- For CE: select * multijoin queries
- For runtime: replace * with min Why?

Cardinality Estimators

What type of queries are in IMDB/JOB?

- For CE: select * multijoin queries
- For runtime: replace * with min Why?

- Materializing * is expensive…
- …and postgres does not push min down the plan

Single Table Estimation

	median	90th	95th	max
PostgreSQL	1.00	2.08	6.10	207
DBMS A	1.01	1.33	1.98	43.4
DBMS B	1.00	6.03	30.2	104000
DBMS C	1.06	1677	5367	20471
HyPer	1.02	4.47	8.00	2084

Table 1: Q-errors for base table selections

Single Table Estimation

What technique helped here? (conjectured)

	median	90th	95th	
PostgreSQL	1.00	2.08	6.10	207
DBMS A	1.01	1.33	1.98	43.4
DBMS B	1.00	6.03	30.2	104000
DBMS C	1.06	1677	5367	20471
HyPer	1.02	4.47	8.00	2084

Table 1: Q-errors for base table selections

Single Table Estimation

	median	90th	95th	
PostgreSQL	1.00	2.08	6.10	207
DBMS A	1.01	1.33	1.98	43.4
DBMS B	1.00	6.03	30.2	104000
DBMS C	1.06	1677	5367	20471
HyPer	1.02	4.47	8.00	2084

Table 1: Q-errors f elections

What technique helped here? (conjectured)

Sampling. E.g. Hyper: 1000 rows

Single Table Estimation

	median	90th	95th	max
PostgreSQL	1.00	2.08	6.10	207
DBMS A	1.01	1.33	1.98	43.4
DBMS B	1.00	6.03	30.2	104000
DBMS C	1.06	1677	5367	20471
HyPer	1.02	4.47	8.00	2084

Table 1: Q-errors for base table selections

Single Table Estim

Why queries still lead to poor estimates?

	median	90th	95th	max
PostgreSQL	1.00	2.08	6.10	207
DBMS A	1.01	1.33	1.98	43.4
DBMS B	1.00	6.03	30.2	104000
DBMS C	1.06	1677	5367	20471
HyPer	1.02	4.47	8.00	2084

Table 1: Q-errors for base table selections

[How good are they]

Single Table Estim

	median	90th	95th	max
PostgreSQL	1.00	2.08	6.10	207
DBMS A	1.01	1.33	1.98	43.4
DBMS B	1.00	6.03	30.2	104000
DBMS C	1.06	1677	5367	20471
HyPer	1.02	4.47	8.00	2084

Table 1: Q-errors for b ons

Why queries still lead to poor estimates?

Low selectivity: 10^{-5} – 10^{-6}

[How good are they]

Single Table Estimation

- 1d Histograms:
  - Good for single equality or range predicate
  - Poor for multiple predicates
  - Useless for LIKE
- Samples:
  - Good for multiple predicates, LIKE
  - Poor for low selectivity predicates

[How good are they]

Joins (0 to 6)

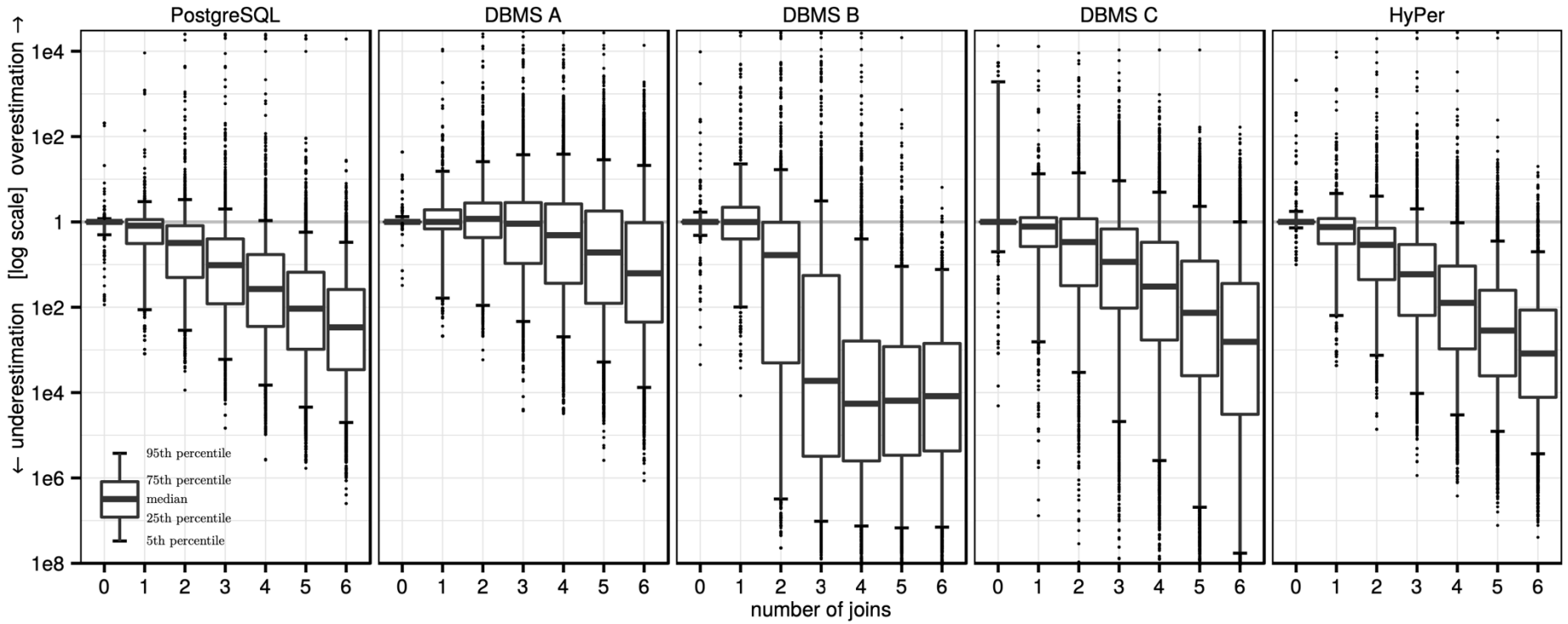


Figure 3: Quality of cardinality estimates for multi-join queries in comparison with the true cardinalities. Each boxplot summarizes the error distribution of all subexpressions with a particular size (over all queries in the workload)

[How good are they]

Joins (0 to 6)

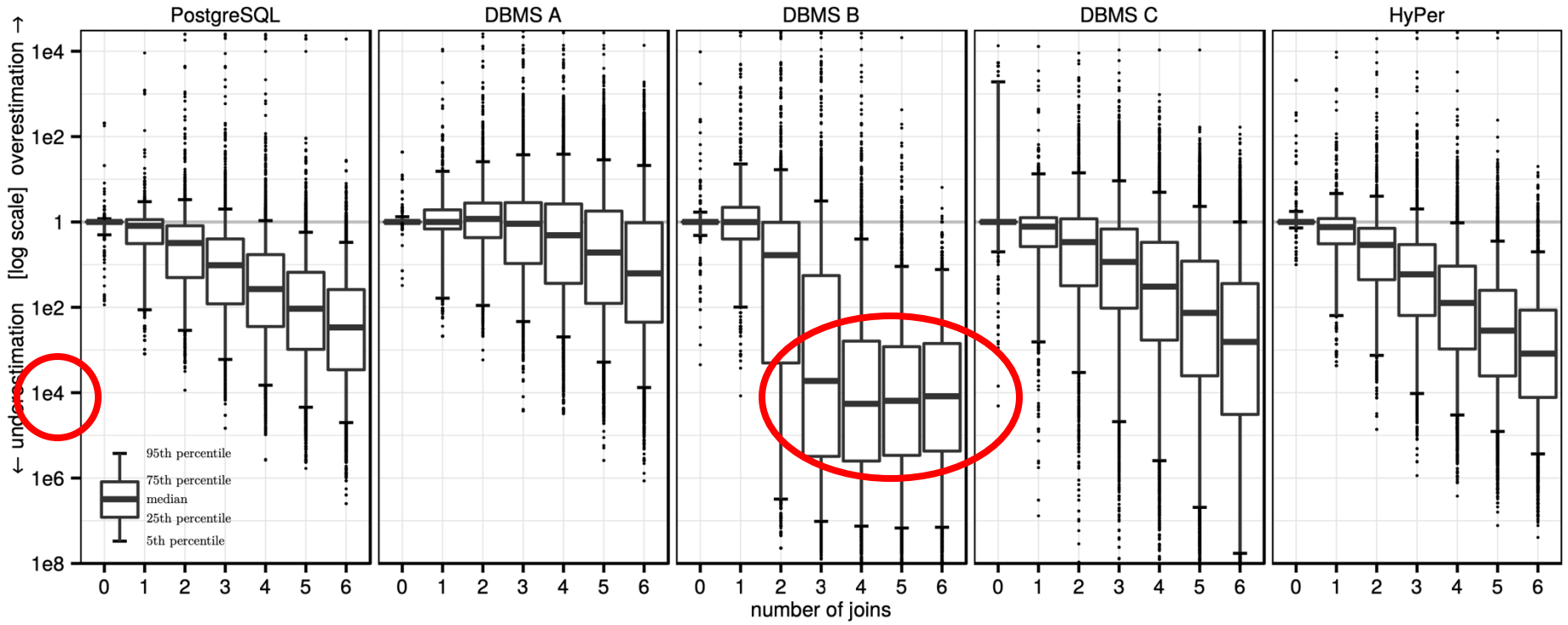


Figure 3: Quality of cardinality estimates for multi-join queries in comparison with the true cardinalities. Each boxplot summarizes the error distribution of all subexpressions with a particular size (over all queries in the workload)

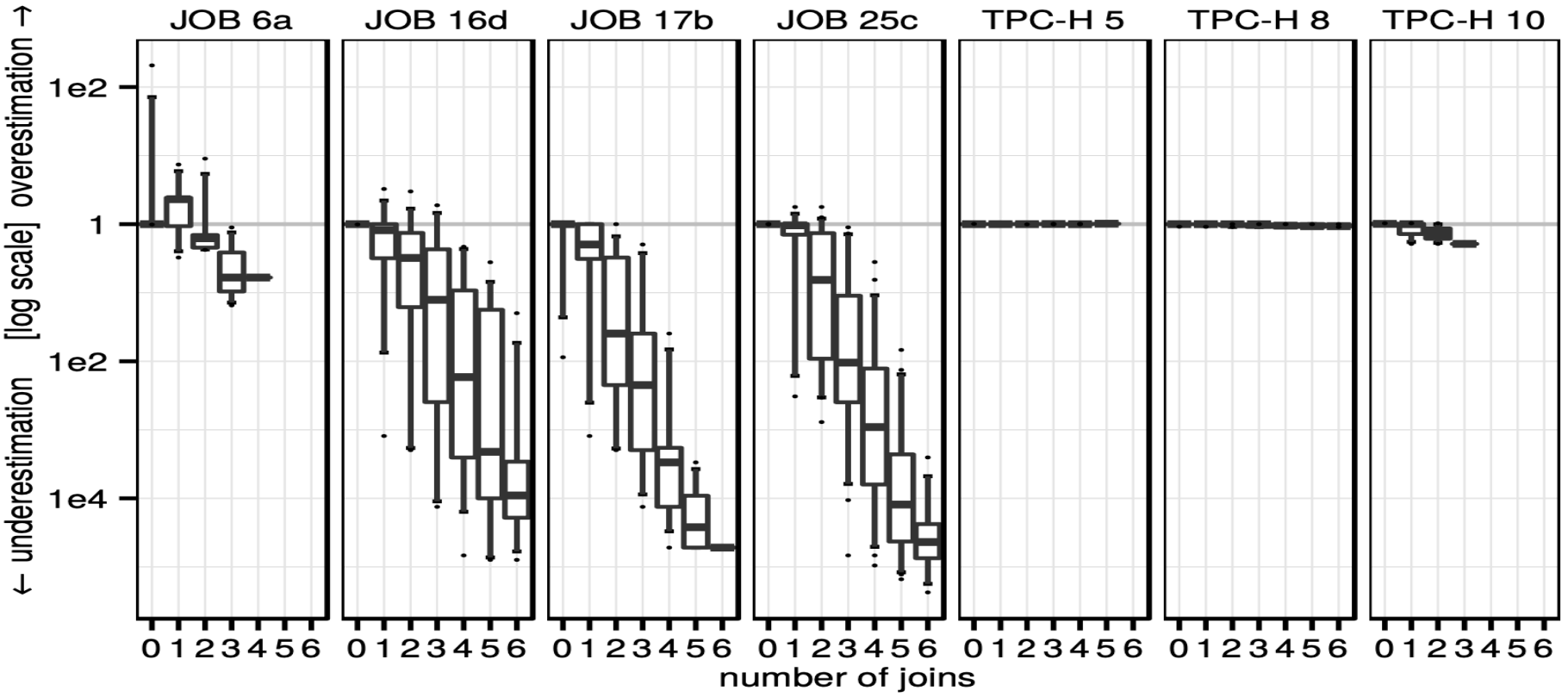
Estimation of Joins

- Error increases exponentially with the number of joins
  - This was known from [Ioannidis'91]

- Underestimate, because of positive correlations

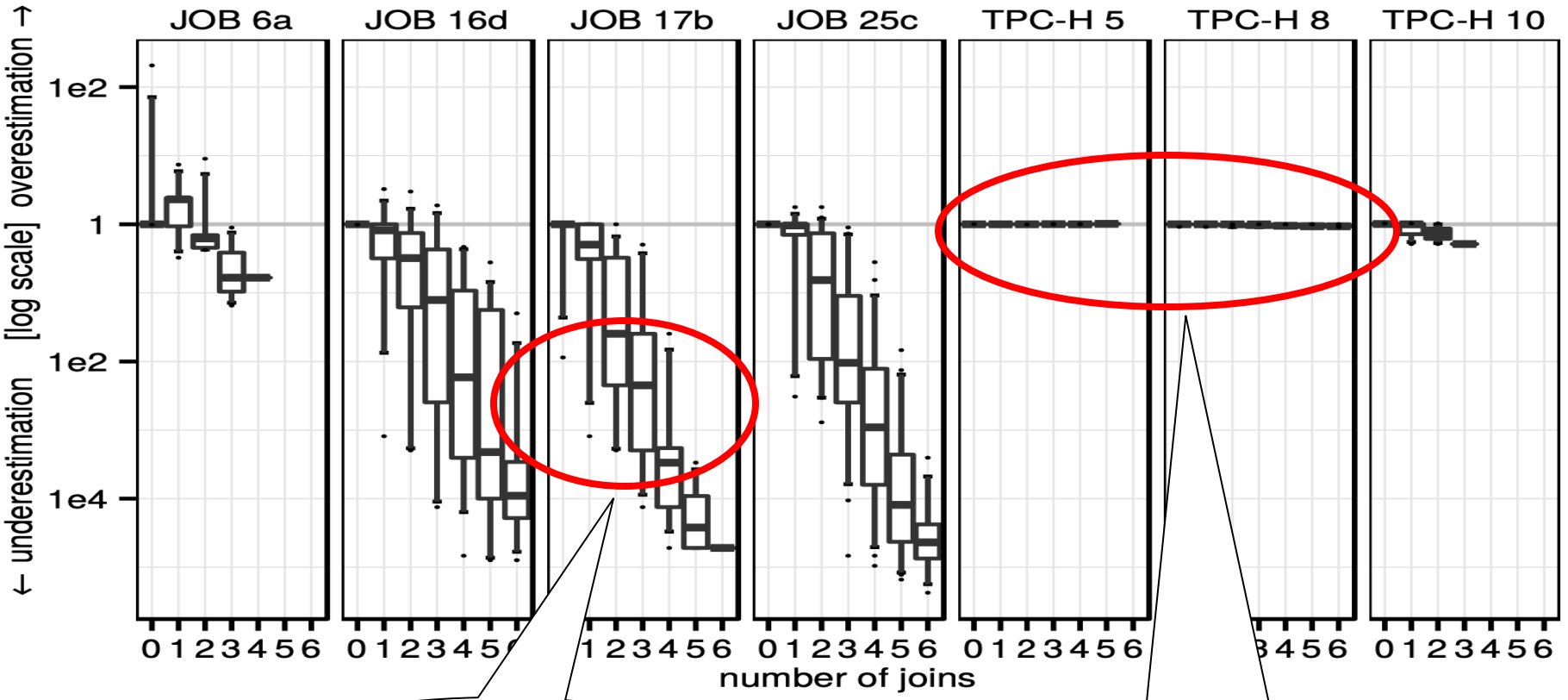
[How good are they]

TPC-H v.s. Real Data (IMDB)



[How good are they]

TPC-H v.s. Real Data (IMDB)



Impact of Mis-estimates

- Question: how much does a good/poor CE matter for the quality of a query plan
- How did they measure that?

Impact of Mis-estimates

- Question: how much does a good/poor CE matter for the quality of a query plan
- How did they measure that?
  - Inject into postgres other systems' estimates – won't discuss this
  - Inject into postgres true cardinalities; call it optimal plan, compare with regular plan
- Two configs of indexes: PK and PK+FK

Impact of Mis-estimates

PK indexes

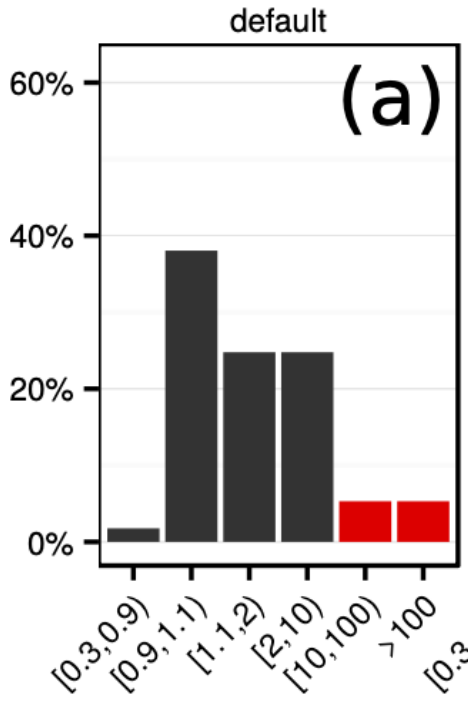


Figure 6: Slowdown of queries using PostgreSQL estimates w.r.t. using true cardinalities (primary key indexes only)

Impact of Mis-estimates

PK indexes

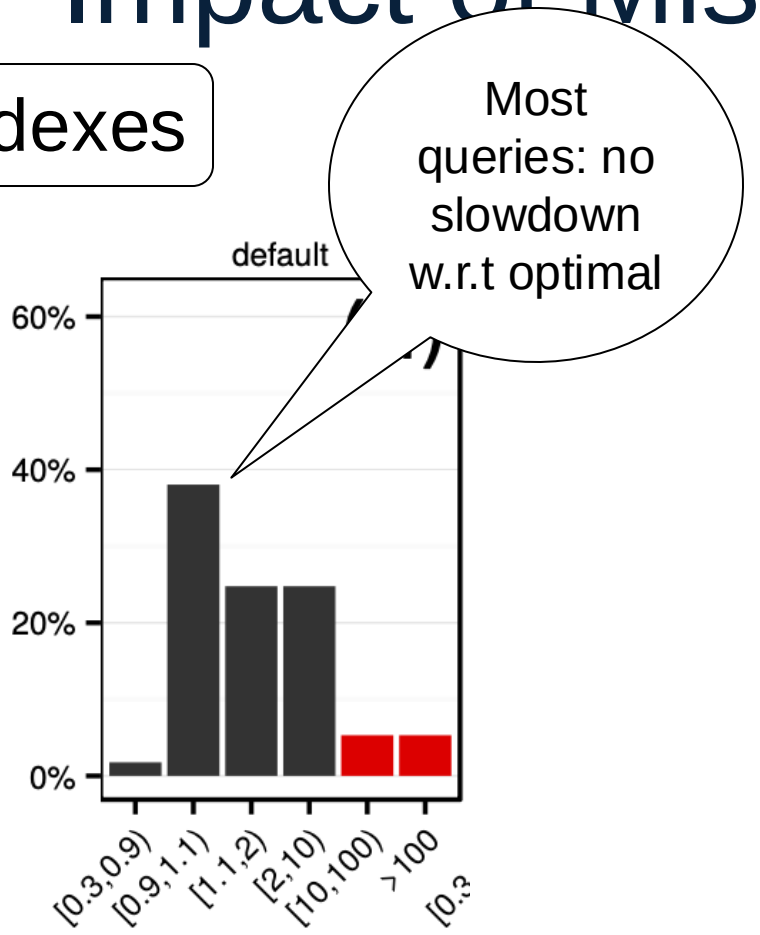


Figure 6: Slowdown of queries using PostgreSQL estimates w.r.t. using true cardinalities (primary key indexes only)

Impact of Mis-estimates

PK indexes

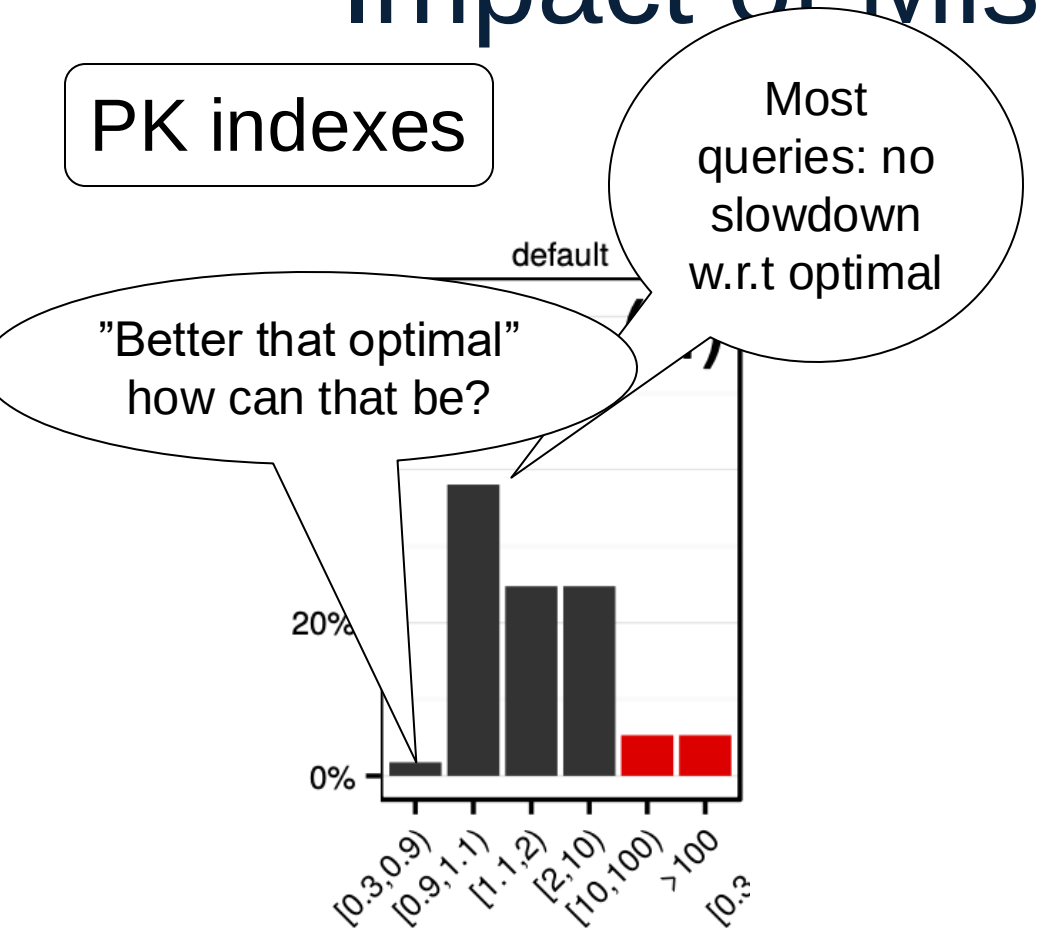


Figure 6: Slowdown of queries using PostgreSQL estimates w.r.t. using true cardinalities (primary key indexes only)

Impact of Mis-estimates

PK indexes

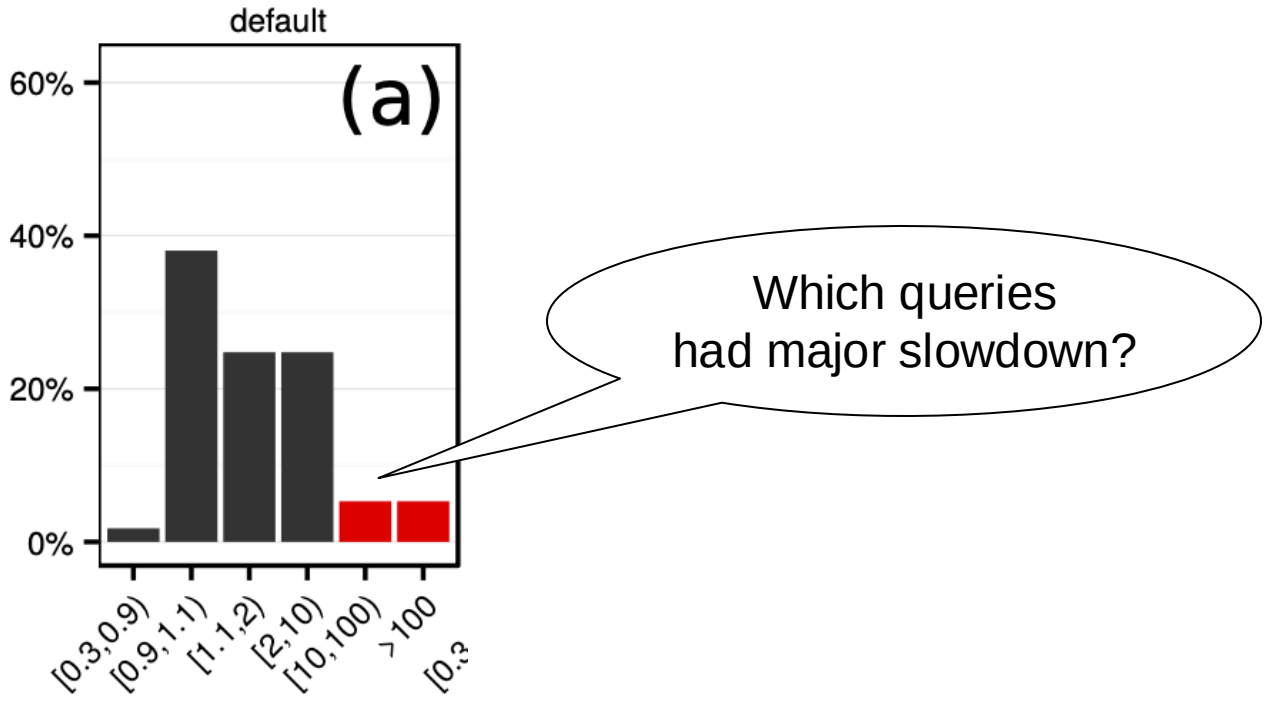


Figure 6: Slowdown of queries using PostgreSQL estimates w.r.t. using true cardinalities (primary key indexes only)

Impact of Mis-estimates

PK indexes

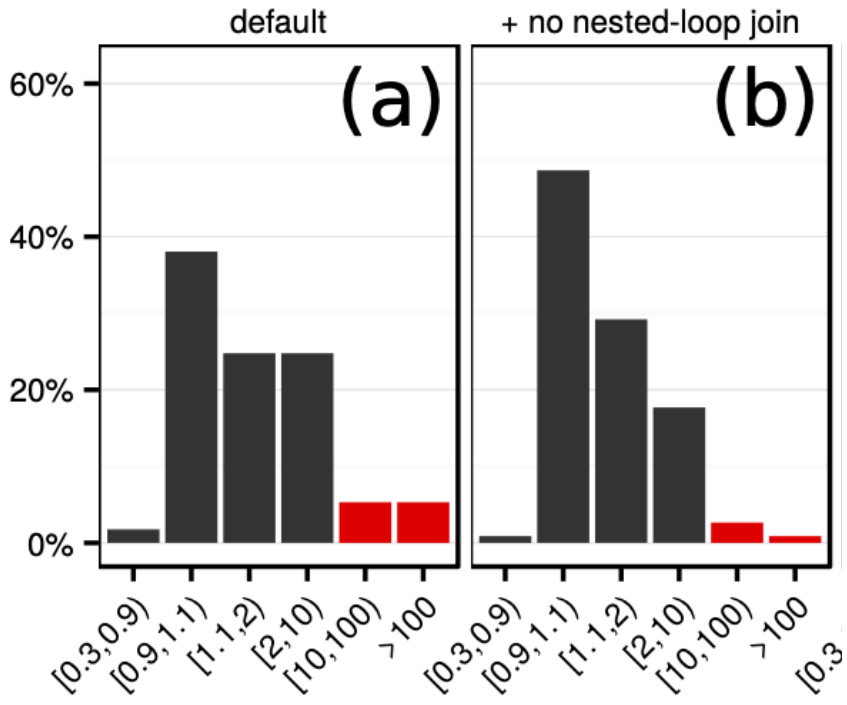
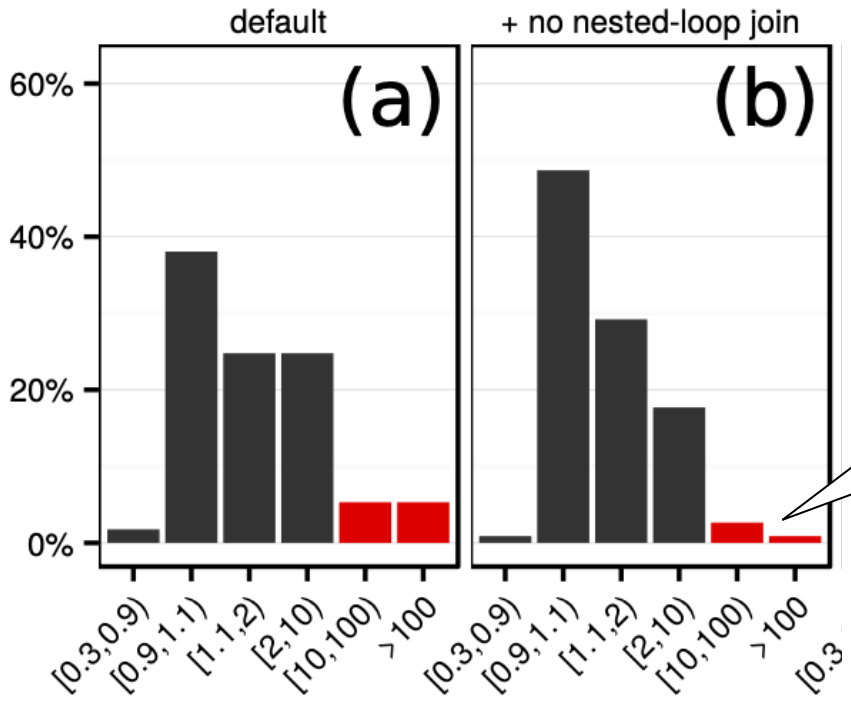


Figure 6: Slowdown of queries using PostgreSQL estimates w.r.t. using true cardinalities (primary key indexes only)

Impact of Mis-estimates

PK indexes



Still some queries significantly slower.

Why?

Figure 6: Slowdown of queries using PostgreSQL estimates w.r.t. using true cardinalities (primary key indexes only)

Impact of Mis-estimates

PK indexes

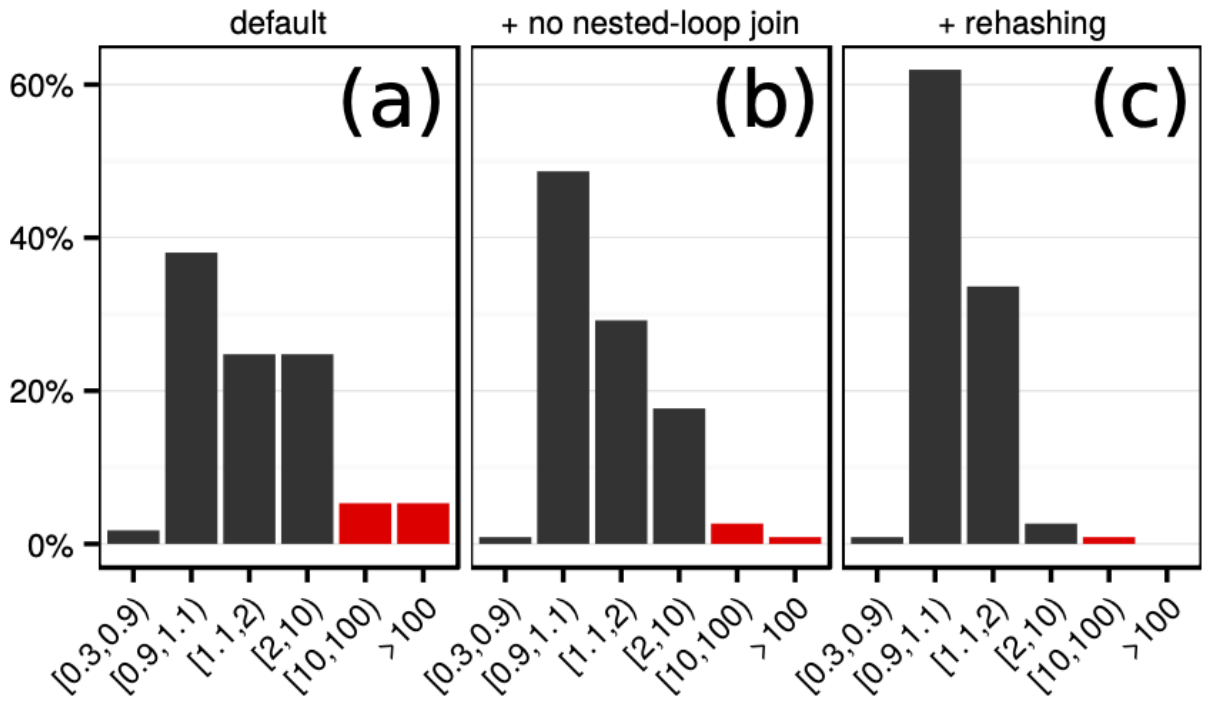


Figure 6: Slowdown of queries using PostgreSQL estimates w.r.t. using true cardinalities (primary key indexes only)

Impact of Mis-estimates

Indexes on PK only

- Low sensitivity to CE, because the “fact” table needs to be scanned anyway
- Plans most sensitive to CE errors:
  - Plans with nested-loop joins
  - Hash-table preallocation
- Discuss “robust query optimization”

Impact of Mis-estimates

FK/PK indexes

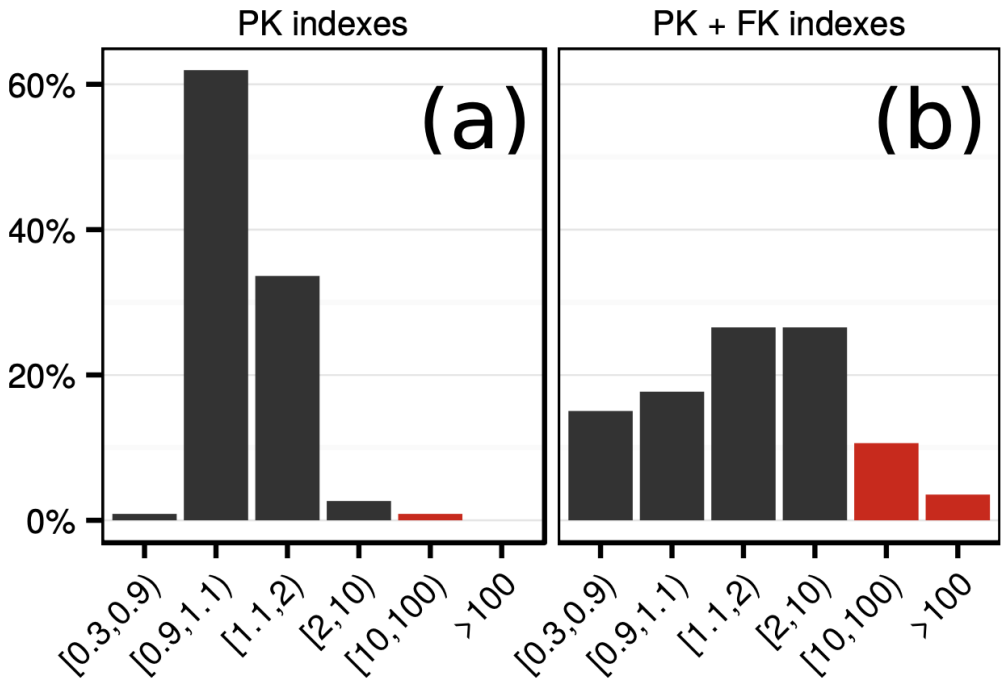


Figure 7: Slowdown of queries using PostgreSQL estimates w.r.t. using true cardinalities (different index configurations)

Impact of Mis-estimates

FK/PK indexes

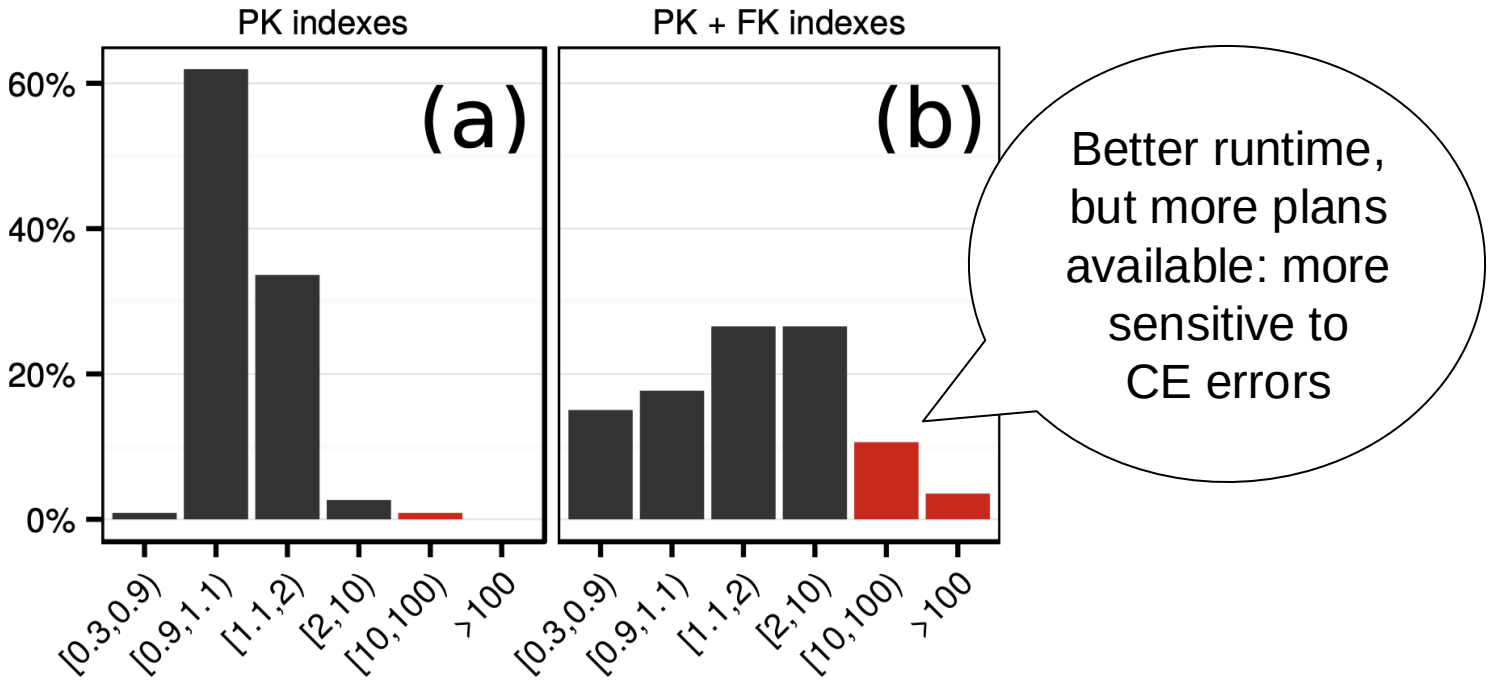


Figure 7: Slowdown of queries using PostgreSQL estimates w.r.t. using true cardinalities (different index configurations)

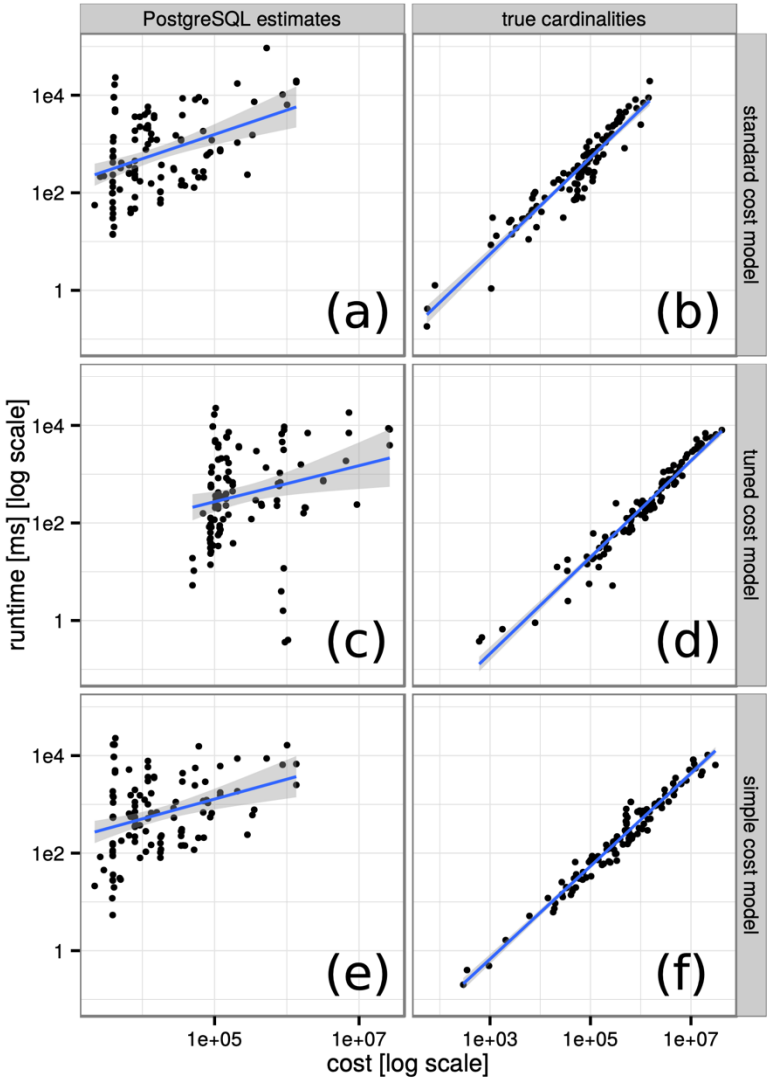
Discussion

- When PK indexes only, optimizer chooses a good plan anyway; impact of CE is limited; confirmed by others too

- When indexes on PK+FK, performance improves, but sensitivity to CE higher

[How good are they]

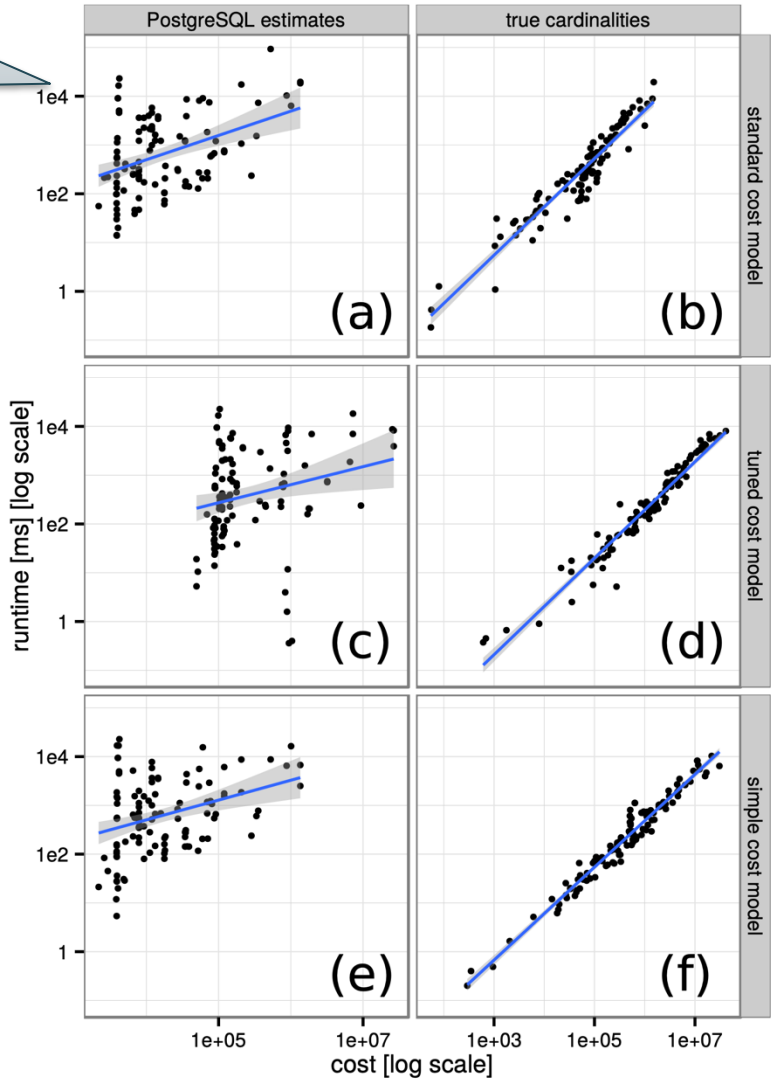
Cardinalities to Cost



[How good are they]

Cardinalities to Cost

Postgres cost

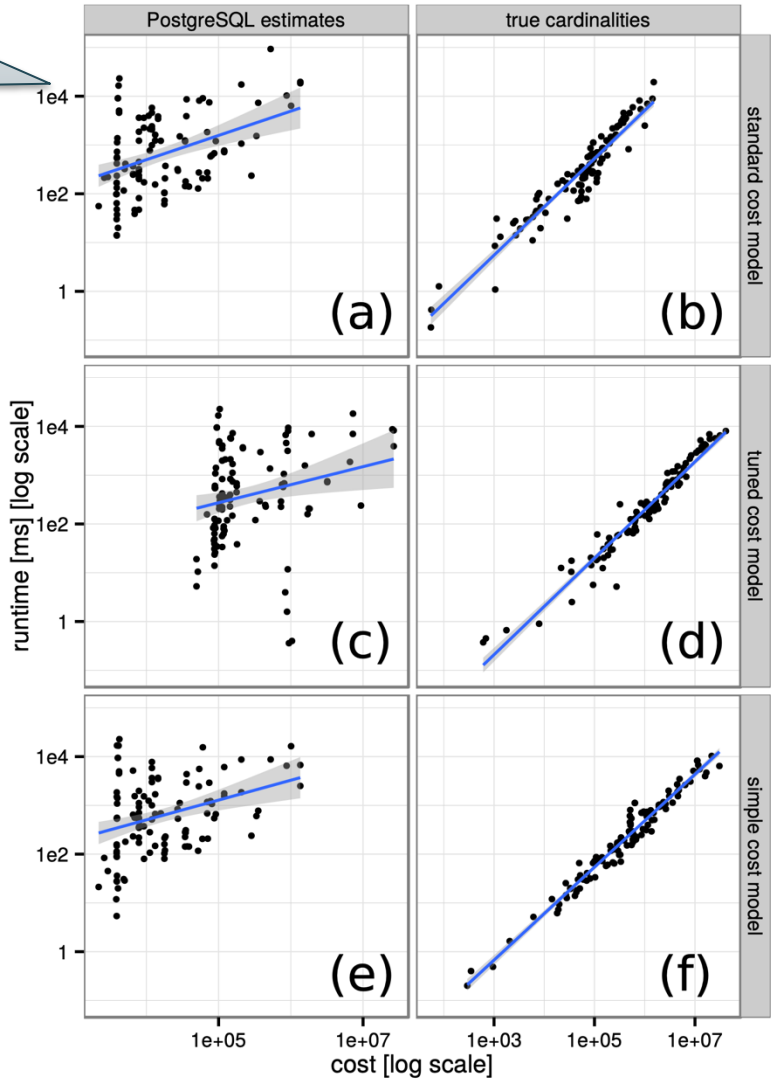


[How good are they]

Cardinalities to Cost

Postgres cost

No I/O, keep only CPU



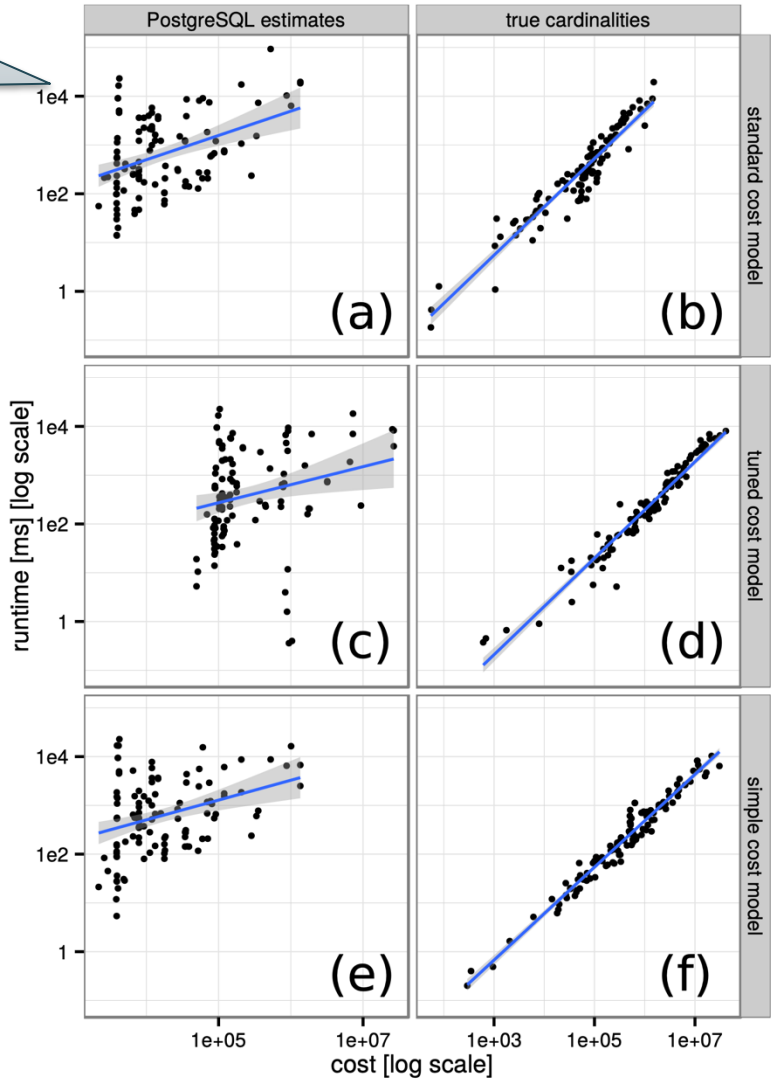
[How good are they]

Cardinalities to Cost

Postgres cost

No I/O, keep only CPU

Their own simple formula



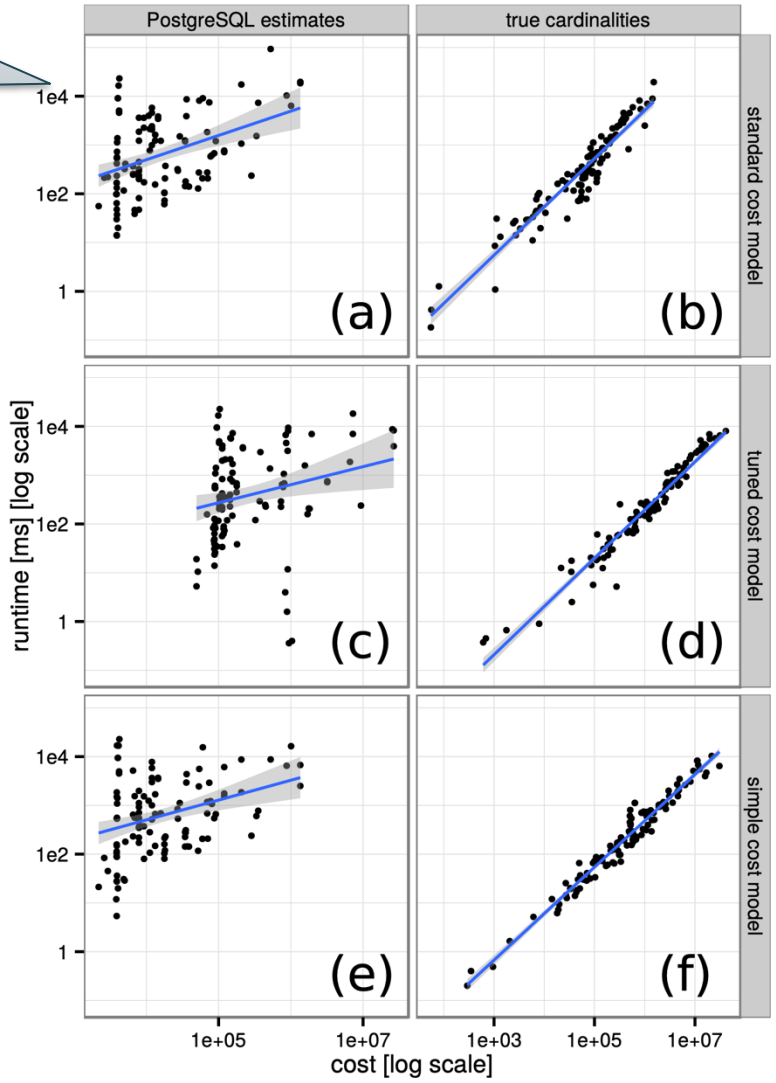
Cardinalities to Cost

- CE accounts for largest errors
- Cost models: both simple or complex are fine

Postgres cost

No I/O, keep only CPU

Their own simple formula



Plan Enumeration Algorithm

Two Types of Plan Enumeration Algorithms

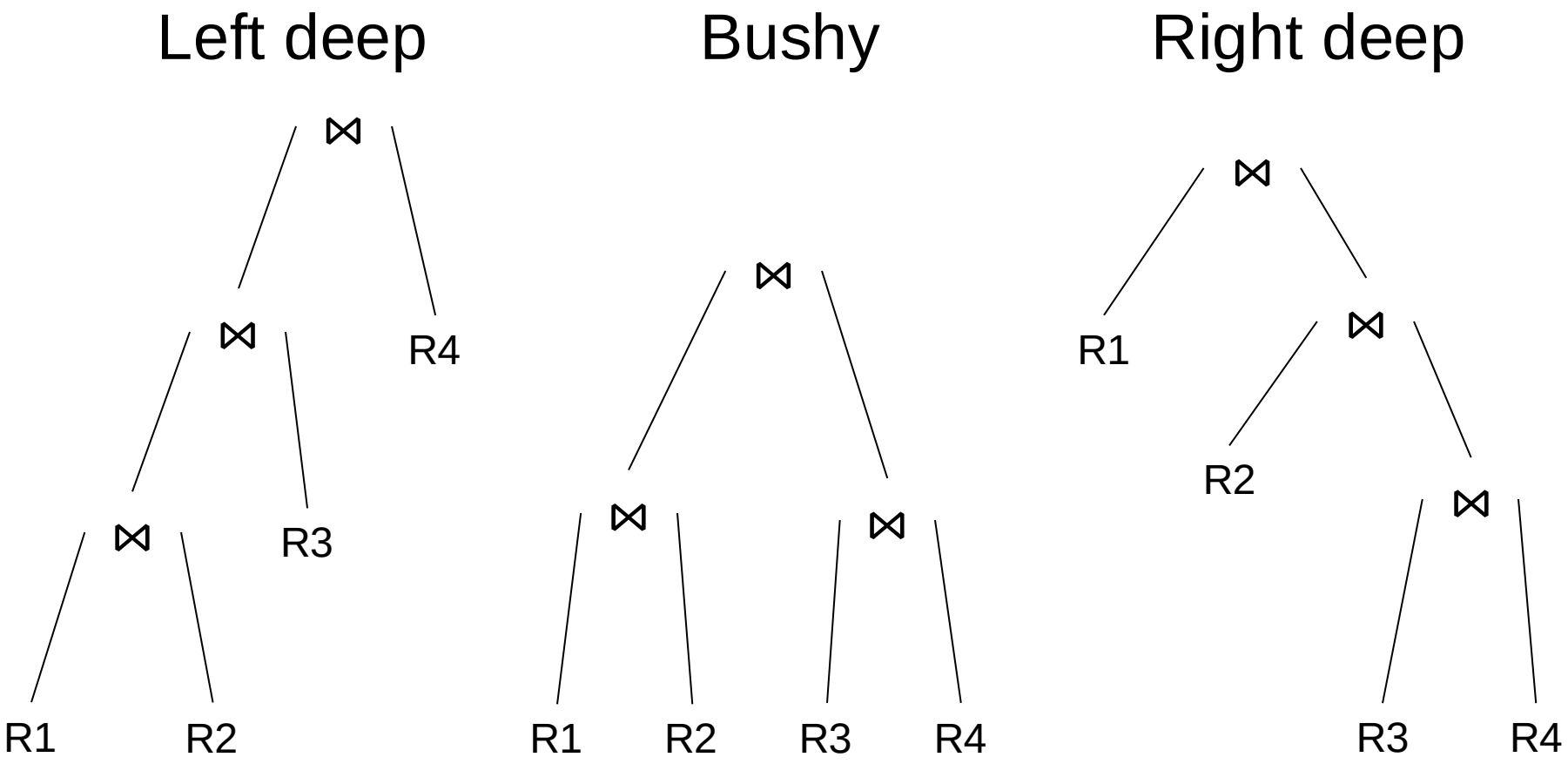
- Dynamic programming
  - Based on System R [Selinger 1979]
  - *Join reordering algorithm*

- Cascades optimizer

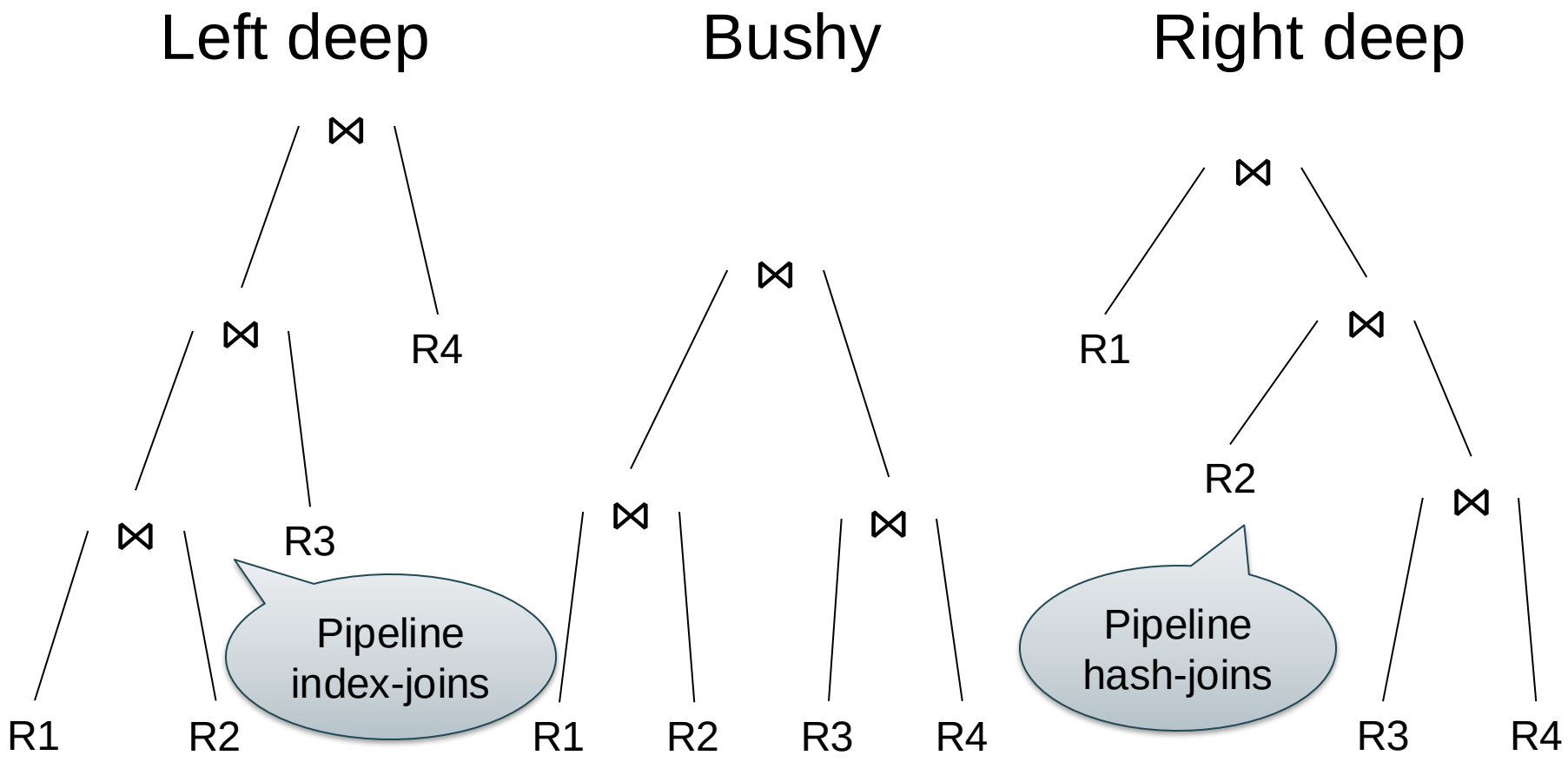
System R Optimizer

- Optimizes single-block SQL query
- Push selections down
- Goal: optimize the join order
  $(R_1 \bowtie R_2) \bowtie R_3$ vs. $R_2 \bowtie (R_1 \bowtie R_3)$
- Dynamic programming: find optimal order of any subset of relations

Types of Query Plans



Types of Query Plans



System R Optimizer

For each subquery $Q \subseteq \{R_1, \ldots, R_n\}$, compute best plan:

- Step 1: $Q = \{R_1\}, \{R_2\}, \ldots, \{R_n\}$
- Step 2: $Q = \{R_1,R_2\}, \{R_1,R_3\}, \ldots, \{R_{n-1}, R_n\}$
- …
- Step n: $Q = \{R_1, \ldots, R_n\}$

Details

Best plan for {R1, R2, R3, R4, R5}:

Details

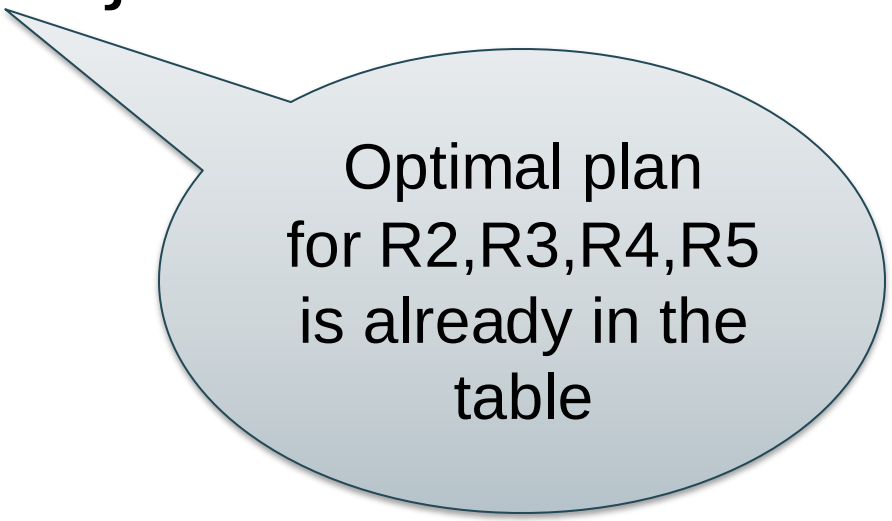
Best plan for {R1, R2, R3, R4, R5}:

- {R1,R2,R3,R4} ⋈ R5

Details

Best plan for {R1, R2, R3, R4, R5}:

- R1 ⋈ {R2,R3,R4,R5}



Details

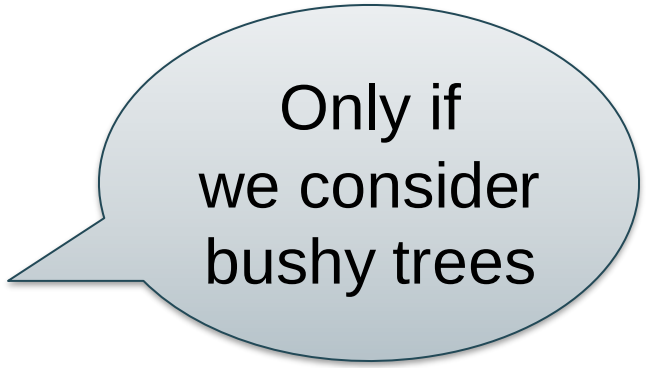
Best plan for {R1, R2, R3, R4, R5}:

- R1 ⋈ {R2,R3,R4,R5}
- R2 ⋈ {R1,R3,R4,R5}
- …

Details

Best plan for {R1, R2, R3, R4, R5}:

- R1 ⋈ {R2,R3,R4,R5}
- R2 ⋈ {R1,R3,R4,R5}
- …
- {R1,R2,R3} ⋈ {R4,R5}
- …



Details

Best plan for {R1, R2, R3, R4, R5}:

- R1 ⋈ {R2,R3,R4,R5}
- R2 ⋈ {R1,R3,R4,R5}
- …
- {R1,R2,R3} ⋈ {R4,R5}
- …

Estimate their cost, keep the minimum

Summary

For each subquery $Q \subseteq \{R_1, \ldots, R_n\}$ store:

- Estimated Size(Q)
- A best plan for Q: Plan(Q)
- The cost of that plan: Cost(Q)

One plan for each "interesting order"

Summary

Step 1: single relations $\{R_1\}$, $\{R_2\}$, …, $\{R_n\}$

- Consider all possible access paths:
  - Sequential scan, or
  - Index 1, or
  - Index 2, or
  - …
- Keep optimal plan for each "interesting order"

Summary

Step k = 2…n:

For each $Q = \{R_{i_1}, \ldots, R_{i_k}\}$

- For each j=1,…,k:
  - Consider all plans of the form $P = P_1 \bowtie P_2$
  - $Cost(P) = Cost(\bowtie) + Cost(P_1) + Cost(P_2)$
  - Keep the cheapest plan, or
  - Keep multiple plans, for “interesting orders”

Runtime: exponential in n.
Mitigated by: no cartesian products, restricted plan shapes

Importance of the Plan Space

- Do we need to explore a large space, or should we pick a plan at random?
- Do we need bushy trees, or are left-, or right-, or zigzag-trees enough?
- Do we need dynamic programming, or is greedy enough?

[How good are they]

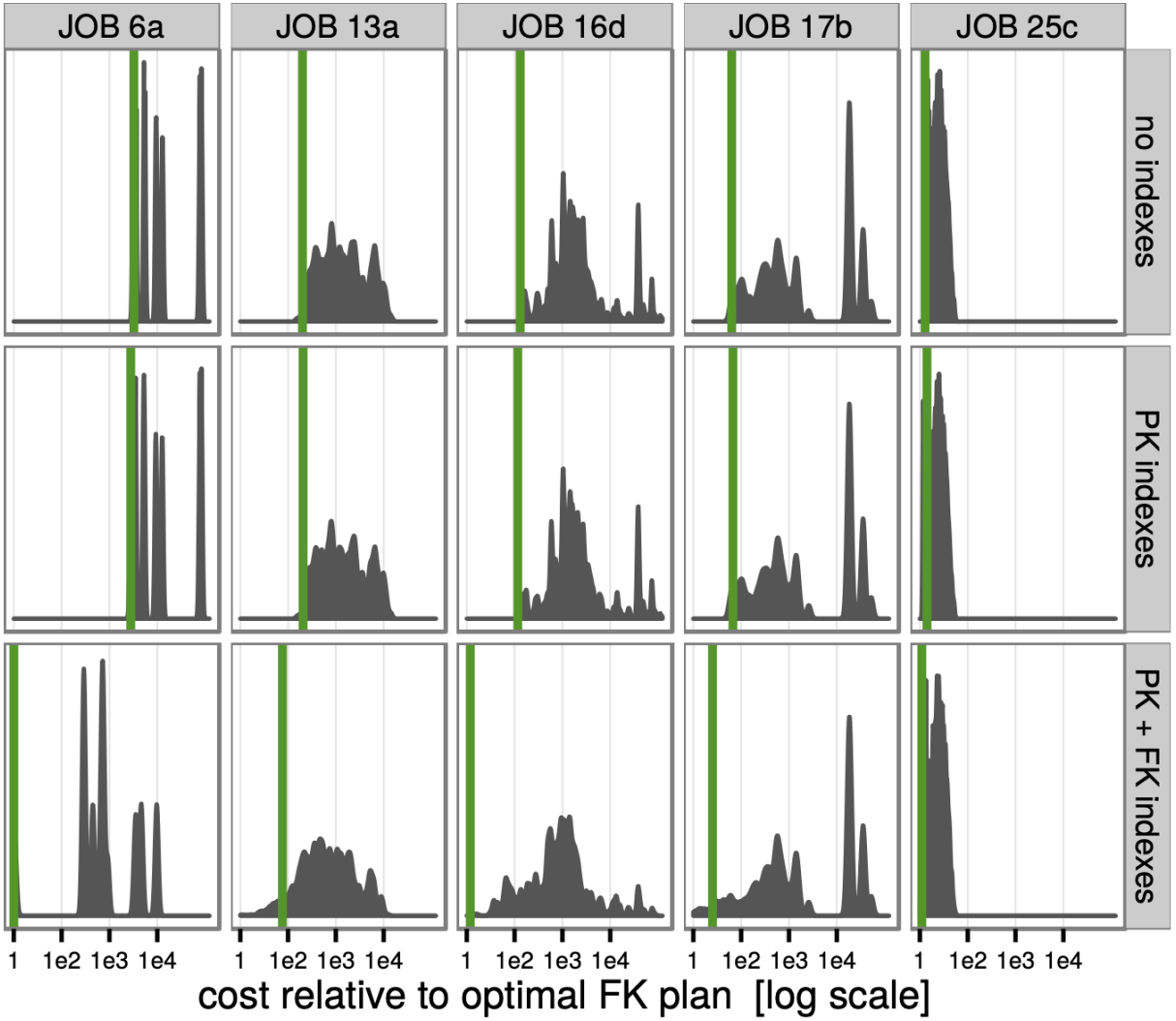


Figure 9: Cost distributions for 5 queries and different index configurations. The vertical green lines represent the cost of the optimal plan

[How good are they]

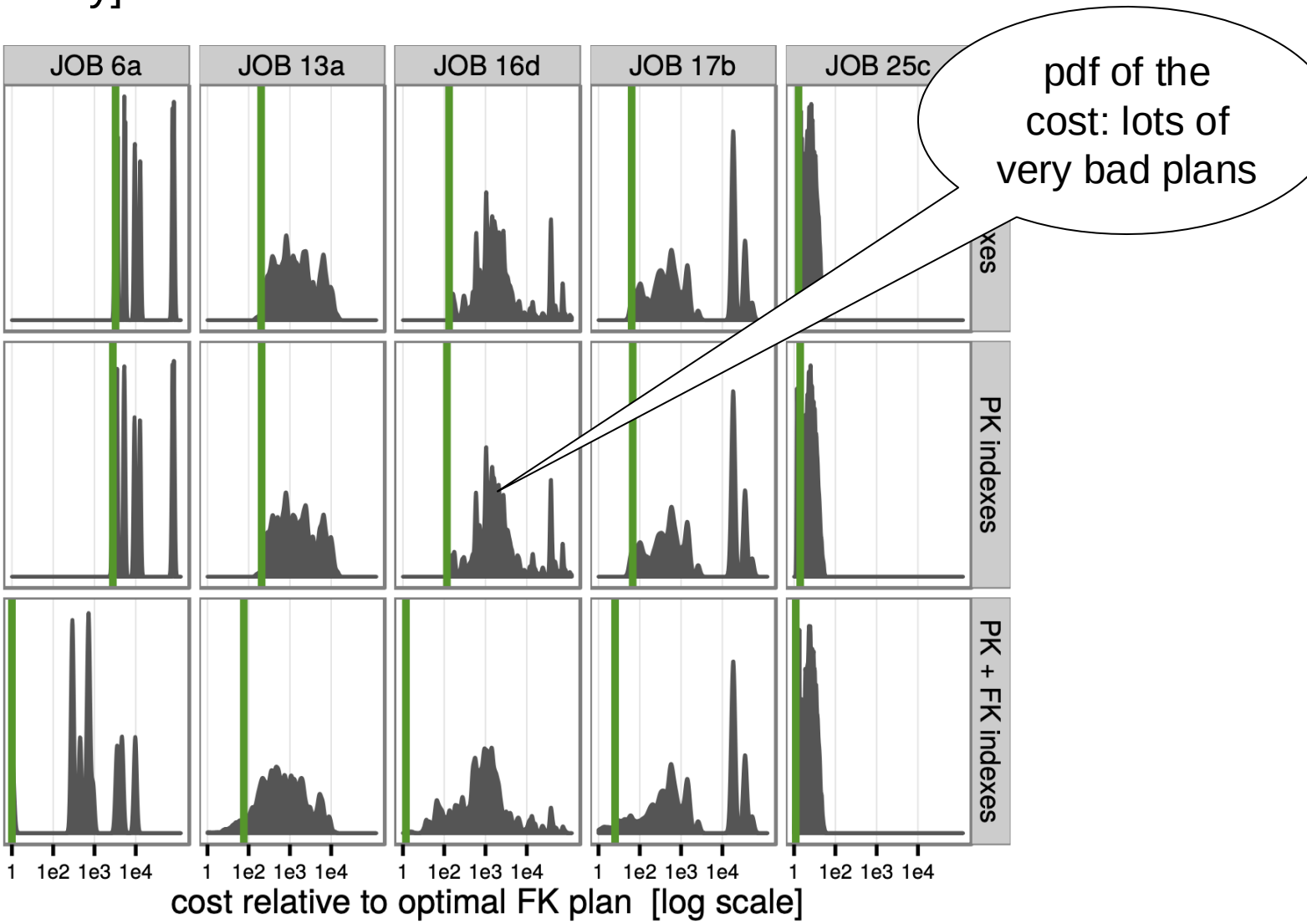


Figure 9: Cost distributions for 5 queries and different index configurations. The vertical green lines represent the cost of the optimal plan

[How good are they]

	PK indexes			PK + FK indexes		
	median	95%	max	median	95%	max
zig-zag	1.00	1.06	1.33	1.00	1.60	2.54
left-deep	1.00	1.14	1.63	1.06	2.49	4.50
right-deep	1.87	4.97	6.80	47.2	30931	738349

Table 2: Slowdown for restricted tree shapes in comparison to the optimal plan (true cardinalities)

[How good are they]

Generally, not much worse than optimal…

	PK indexes			PK + FK indexes		
	median	95%	max	median	95%	max
zig-zag	1.00	1.06	1.33	1.00	1.60	2.54
left-deep	1.00	1.14	1.63	1.06	2.49	4.50
right-deep	1.87	4.97	6.80	47.2	30931	738349

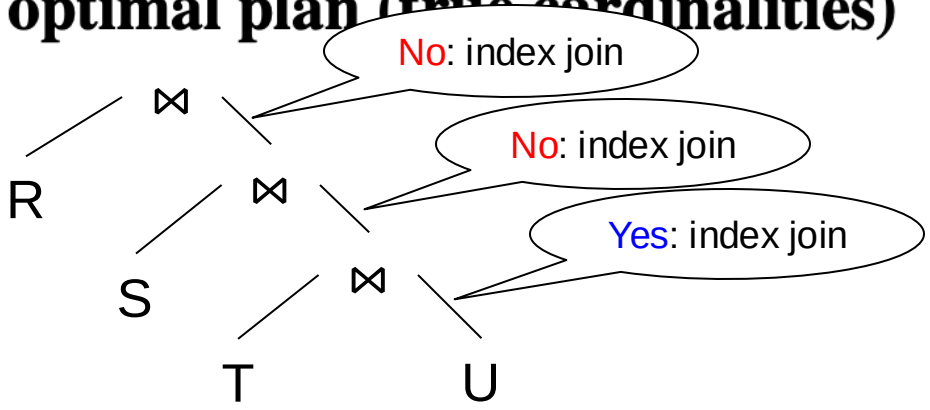
Table 2: Slowdown for restricted tree shapes in comparison to the optimal plan (true cardinalities)

[How good are they]

Generally, not much worse than optimal…

	PK indexes			PK + FK indexes		
	median	95%	max	median	95%	max
zig-zag	1.00	1.06	1.33	1.00	1.60	2.54
left-deep	1.00	1.14	1.63	1.06	2.49	4.50
right-deep	1.87	4.97	6.80	47.2	30931	738349

Table 2: Slowdown for restricted tree shapes in comparison to the optimal plan (true cardinalities)



…except here. Right-deep plans prevent index joins.

[How good are they]

	PK indexes						PK + FK indexes					
	PostgreSQL estimates			true cardinalities			PostgreSQL estimates			true cardinalities		
	median	95%	max	median	95%	max	median	95%	max	median	95%	max
Dynamic Programming	1.03	1.85	4.79	1.00	1.00	1.00	1.66	169	186367	1.00	1.00	1.00
Quickpick-1000	1.05	2.19	7.29	1.00	1.07	1.14	2.52	365	186367	1.02	4.72	32.3
Greedy Operator Ordering	1.19	2.29	2.36	1.19	1.64	1.97	2.35	169	186367	1.20	5.77	21.0

Table 3: Comparison of exhaustive dynamic programming with the Quickpick-1000 (best of 1000 random plans) and the Greedy Operator Ordering heuristics. All costs are normalized by the optimal plan of that index configuration

Cascades Optimizer

- Extends join ordering to full rewrite
- Supported by some of the most advanced DBMS today: SQL Server, Cocroach Lab; (not sure about DuckDB)
- Mostly “insider knowledge”

Cascades Optimizer

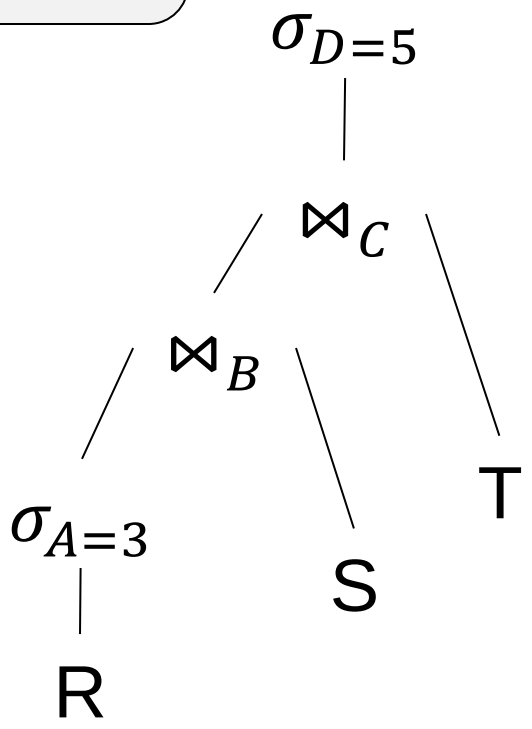
- Main idea: apply optimization rules:
  Q → Q'
- But keep both Q and Q'
- "Memo" data structure: reuses subplans

R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

Initialize Memo w/ one (naïve) plan



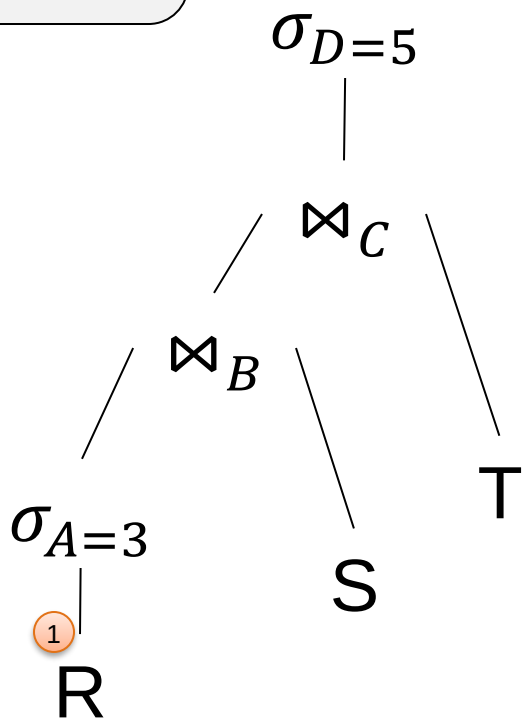
R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1 Scan R

Initialize Memo w/ one (naïve) plan



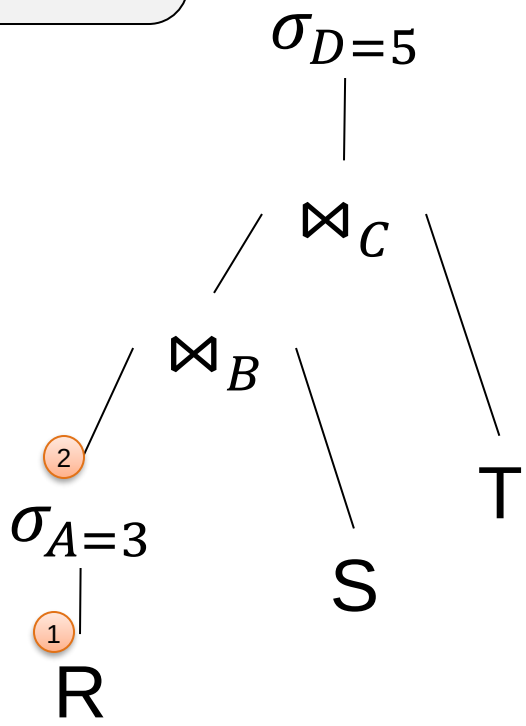
R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R
2	Select[A=3] 1

Initialize Memo w/ one (naïve) plan



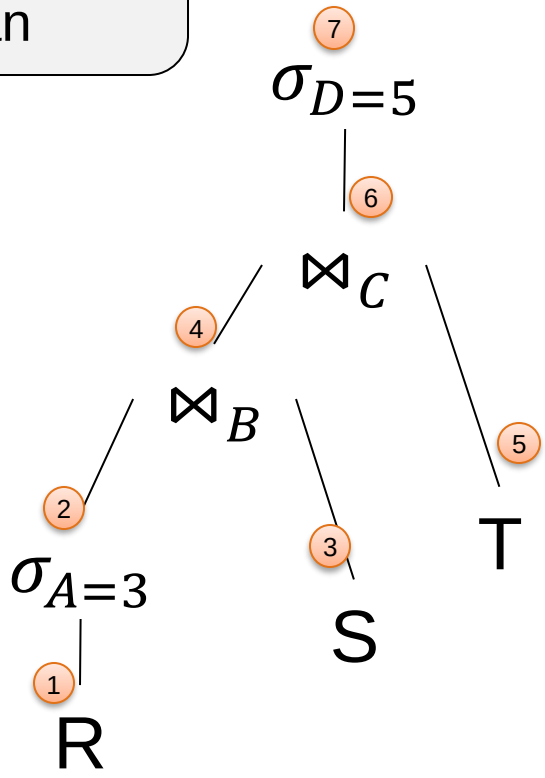
R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R
2	Select[A=3] 1
3	Scan S
4	Join[B=B] 2, 3
5	Scan T
6	Join[C=C] 4,5
7	Select[D=5] 6

Initialize Memo w/ one (naïve) plan

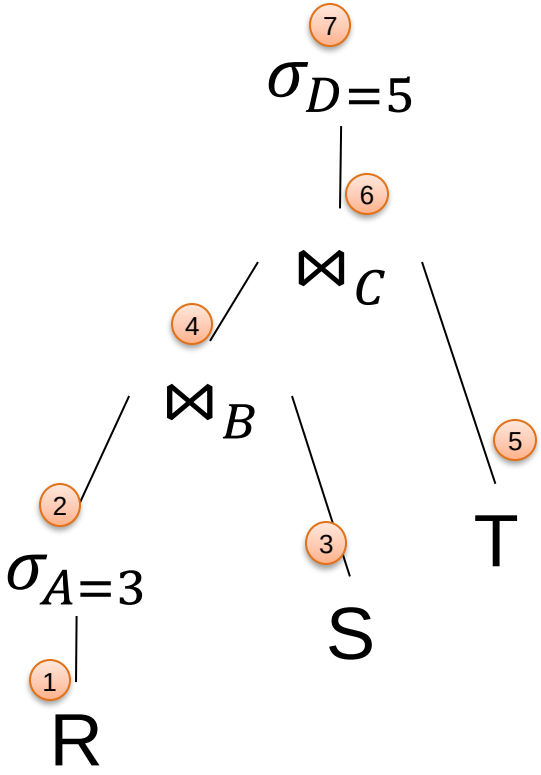


R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R
2	Select[A=3] 1
3	Scan S
4	Join[B=B] 2, 3
5	Scan T
6	Join[C=C] 4,5
7	Select[D=5] 6



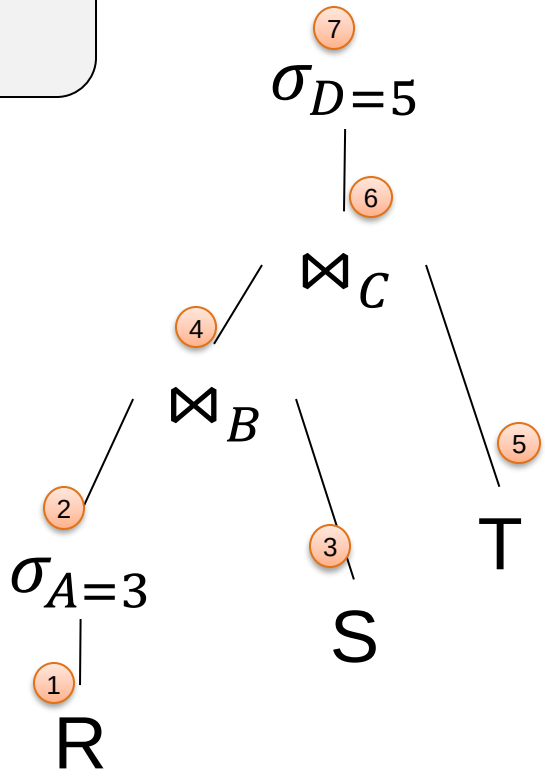
R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R
2	Select[A=3] 1
3	Scan S
4	Join[B=B] 2, 3
5	Scan T
6	Join[C=C] 4,5
7	Select[D=5] 6

Apply an optimization rule



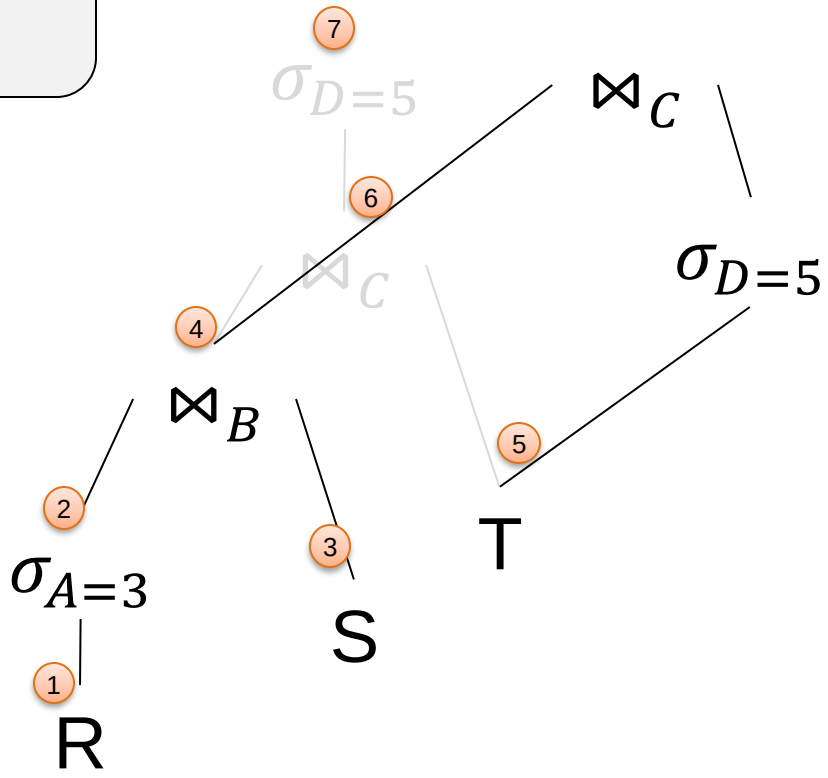
R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R
2	Select[A=3] 1
3	Scan S
4	Join[B=B] 2, 3
5	Scan T
6	Join[C=C] 4,5
7	Select[D=5] 6

Apply an optimization rule

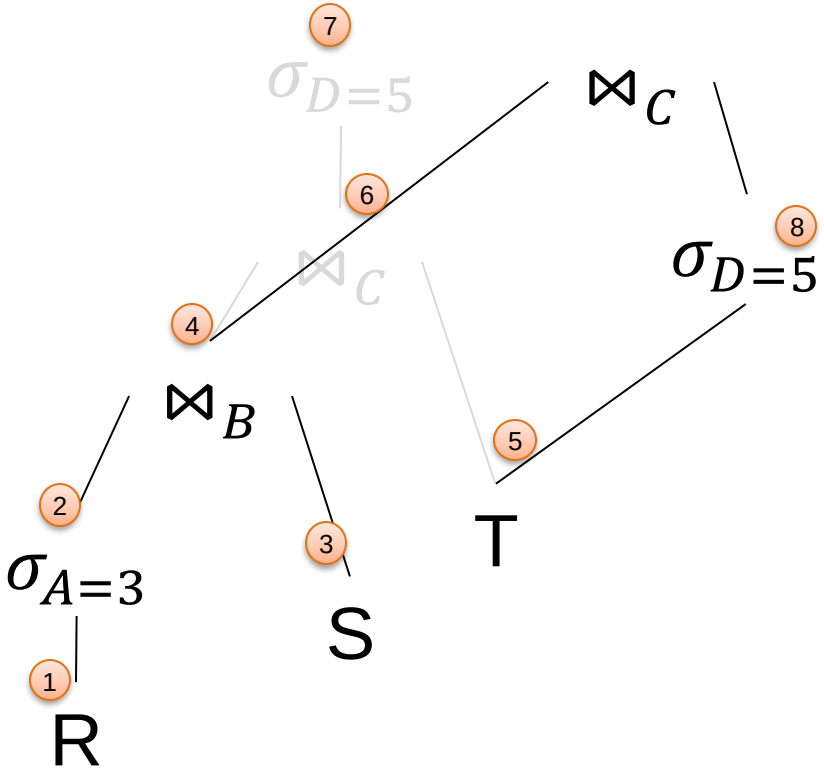


R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R
2	Select[A=3] 1
3	Scan S
4	Join[B=B] 2, 3
5	Scan T
6	Join[C=C] 4,5
7	Select[D=5] 6
8	Select[D=5] 5

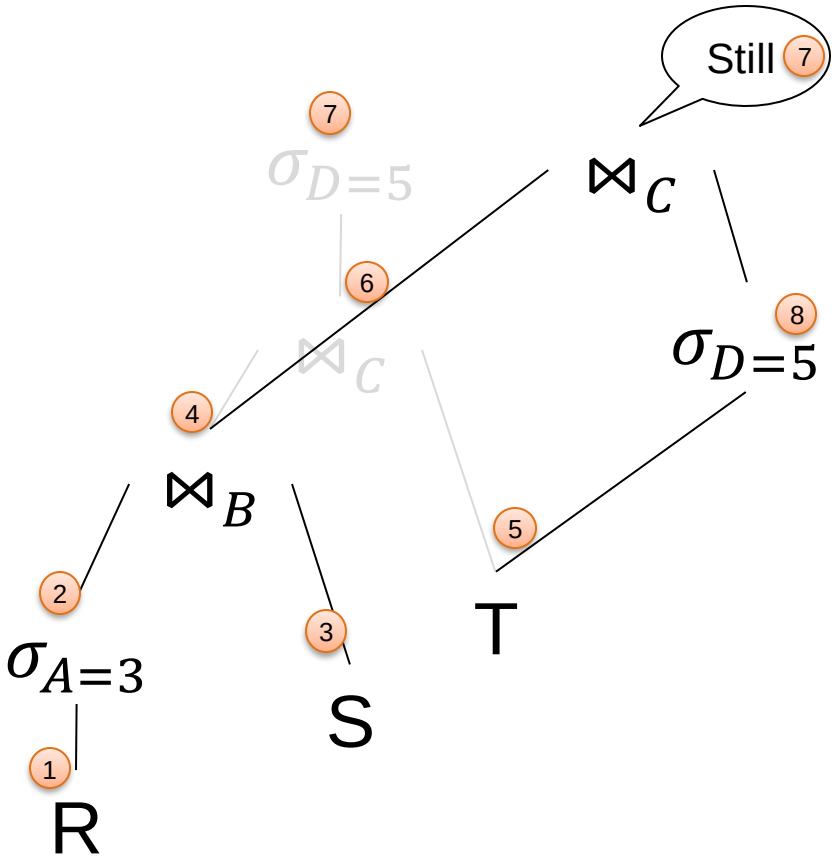


R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R	
2	Select[A=3] 1	
3	Scan S	
4	Join[B=B] 2, 3	
5	Scan T	
6	Join[C=C] 4,5	
7	Select[D=5] 6	Join[C=C] 4,8
8	Select[D=5] 5	

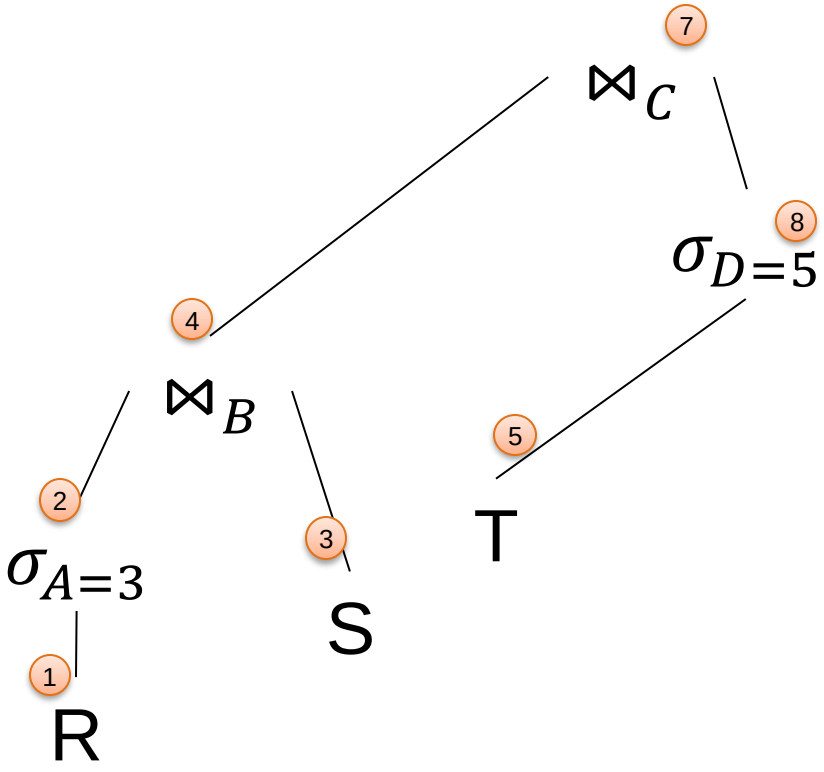


R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R	
2	Select[A=3] 1	
3	Scan S	
4	Join[B=B] 2, 3	
5	Scan T	
6	Join[C=C] 4,5	
7	Select[D=5] 6	Join[C=C] 4,8
8	Select[D=5] 5	

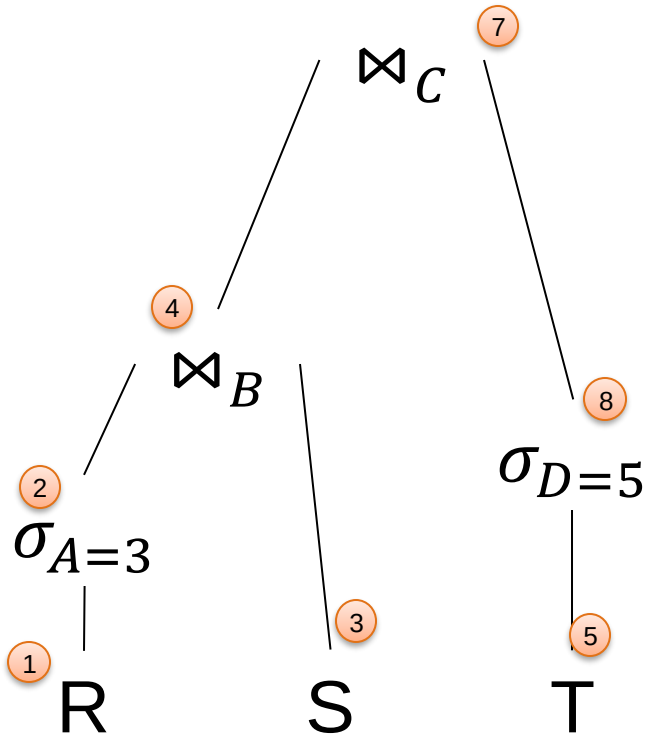


R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R
2	Select[A=3] 1
3	Scan S
4	Join[B=B] 2, 3
5	Scan T
6	Join[C=C] 4,5
7	Select[D=5] 6 Join[C=C] 4,8
8	Select[D=5] 5



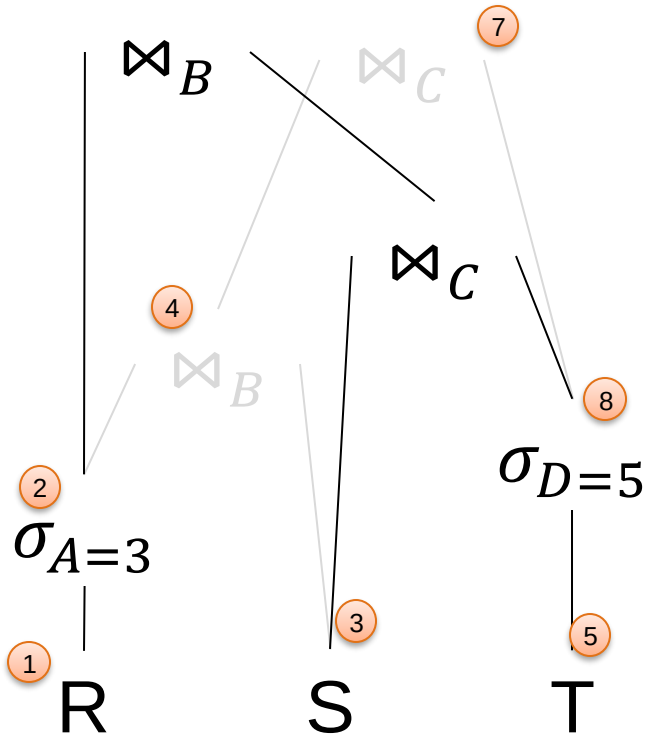
R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R	
2	Select[A=3] 1	
3	Scan S	
4	Join[B=B] 2, 3	
5	Scan T	
6	Join[C=C] 4,5	
7	Select[D=5] 6	Join[C=C] 4,8
8	Select[D=5] 5	

Apply another rule

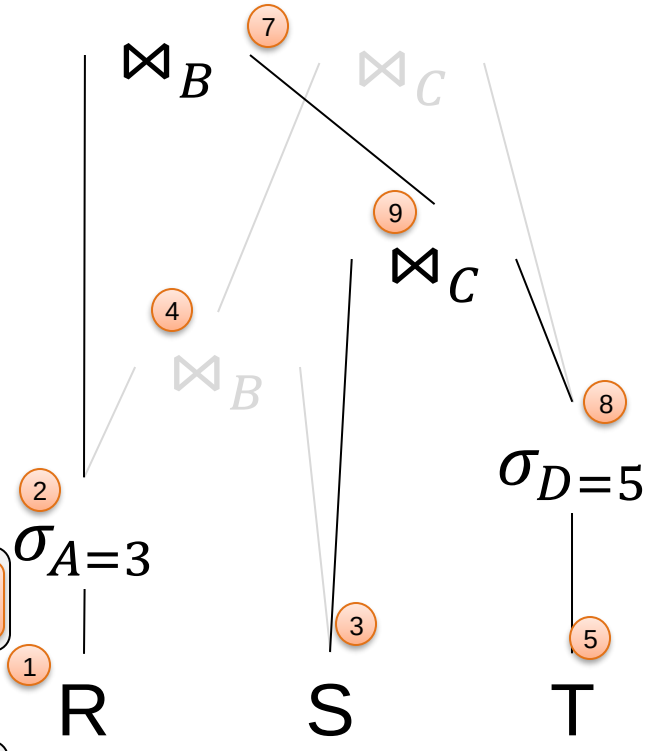


R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R
2	Select[A=3] 1
3	Scan S
4	Join[B=B] 2, 3
5	Scan T
6	Join[C=C] 4,5
7	Select[D=5] 6 Join[C=C] 4,8 Join[B=B] 2,9
8	Select[D=5] 5
9	Join[C=C] 3, 8



Conclusions

- Query optimizers: some of the most complex systems in use today

> Query optimization is not rocket science.
>
> If you fail at query optimization, they send you to build rockets.
>
> Anonymous