CSE544
Data Management

Lectures 9-10

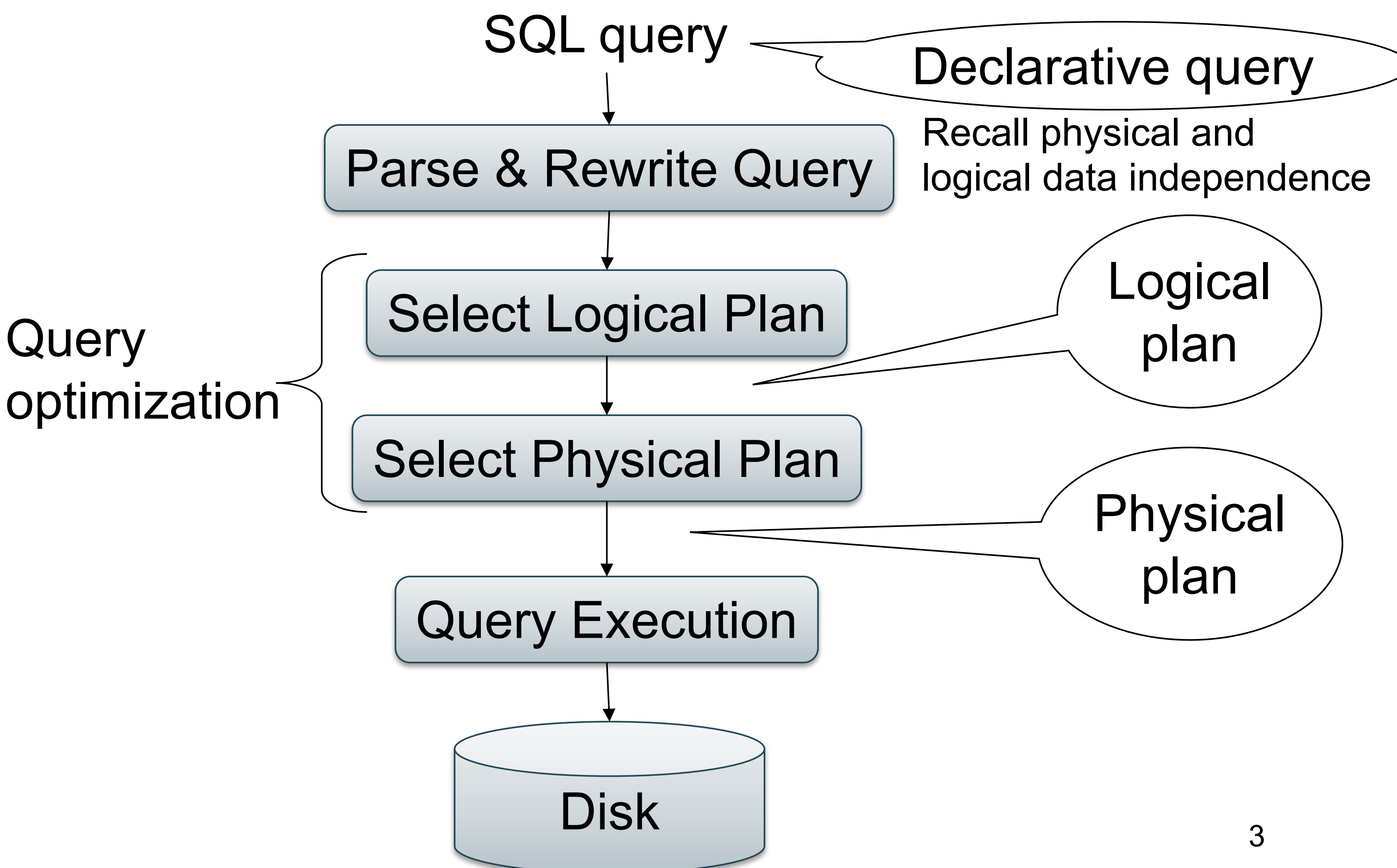
Query Optimization

Announcements

- Project meetings this Friday

- HW3 is posted, due next Friday

Query Optimization Motivation



Today

- Discuss Query Optimization

- In parallel, discuss the paper *How Good Are Query Optimizers, Really?* VLDB'2015

What We Already Know

- There exists many logical plans...
- ... and for each, there exist many physical plans
- Optimizer chooses the logical/physical plan with the smallest *estimated* cost

Query Optimization

Three major components:

1. Cardinality and cost estimation

2. Search space

3. Plan enumeration algorithms

Cost Estimation

Goal: compute cost of an entire physical plan

- We know how to compute the cost given B, T:
  - E.g. index join COST = $B(R)+T(R)B(S)/V(S,a)$

New Goal: estimate $T(R)$ for each intermediate R
"Cardinality Estimation"

Cardinality Estimation

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

Very difficult, because:

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

Statistics on Base Data

Statistics on base tables

- 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)

Computed periodically, often using sampling

Assumptions

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

Size Estimation

Projection: output size same as input size

$T(\Pi(R)) = T(R)$

Selection: size decreases by *selectivity factor* θ

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

Uniformity assumption

Selectivity Factors

- A = c /* $\sigma_{A=c}(R)$ */
  - Selectivity = 1/V(R,A)

- c1 < A < c2 /* $\sigma_{c1<A<c2}(R)$*/
  - Selectivity = (c2 – c1)/(max(R,A) - min(R,A))

Multiple predicates: *independence assumption*

- A = c and B = d /* $\sigma_{A=c \text{ and } B=d}(R)$ */
  - Selectivity = 1/V(R,A) * 1/V(R,B)

Estimating Result Sizes

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

- Take product of cardinalities of R and S
- Apply this selectivity factor:
  1/ (MAX(V(R,A), V(S,B))
- Why? Will explain next...

Assumptions

- *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

- *Preservation of values*: for any other attribute C, $V(R \bowtie_{A=B} S, C) = V(R, C)$ (or $V(S, C)$)
  - This is only needed higher up in the plan

Selectivity of $R \bowtie_{A=B} S$

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

- Each 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:

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

Computing the Cost of a Plan

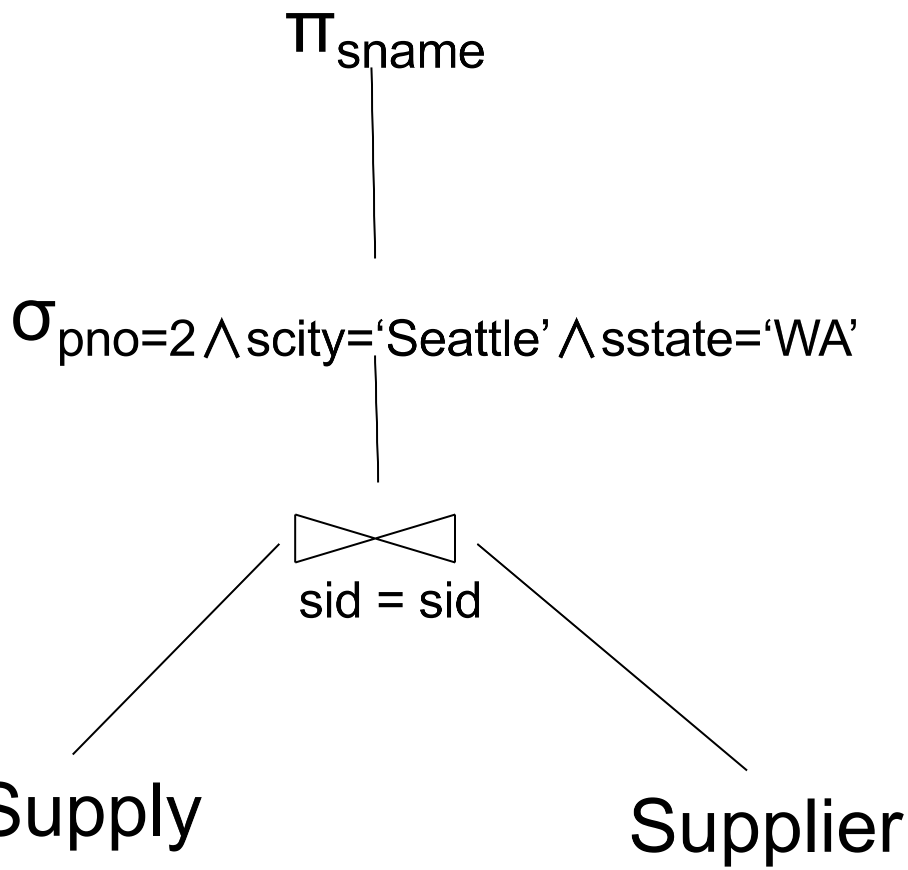
- Estimate cardinality in a bottom-up fashion
  - Cardinality is the size of a relation (nb of tuples)
  - Compute size of *all* intermediate relations in plan

- Estimate cost by using the estimated cardinalities

- Extensive example 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'
```

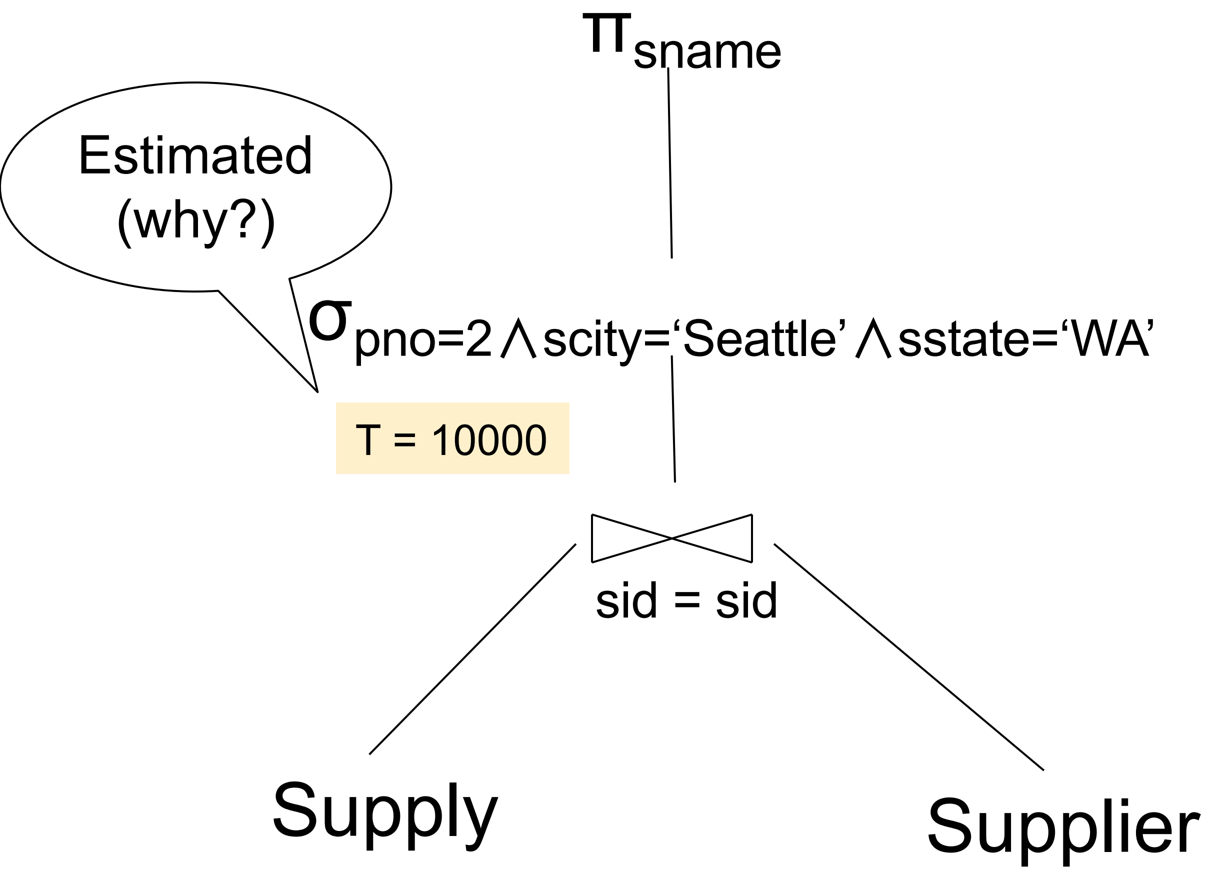
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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

M=11

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'
```

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

Estimated (why?)

π_{sname}

T < 1

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

T = 10000

$\bowtie_{sid = sid}$

Supply

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

Supplier

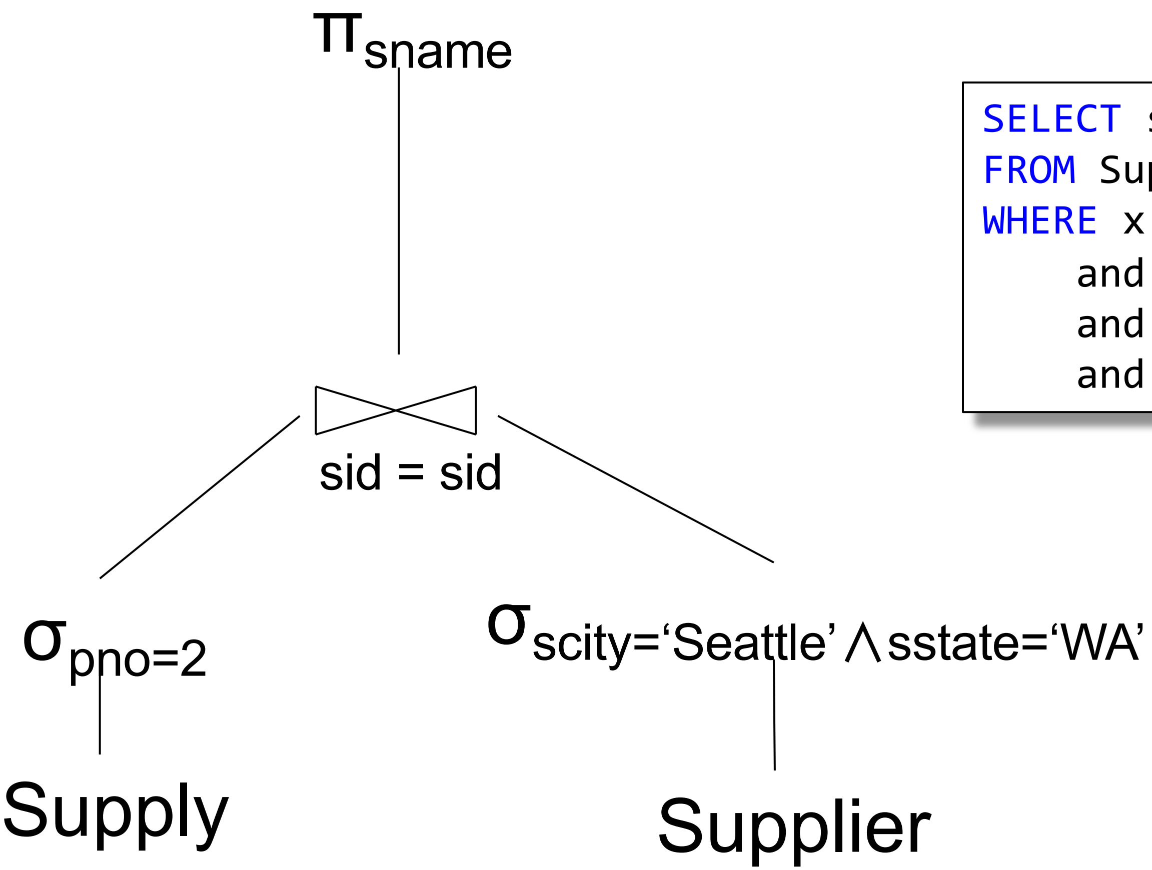
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

```sql
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 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’
```

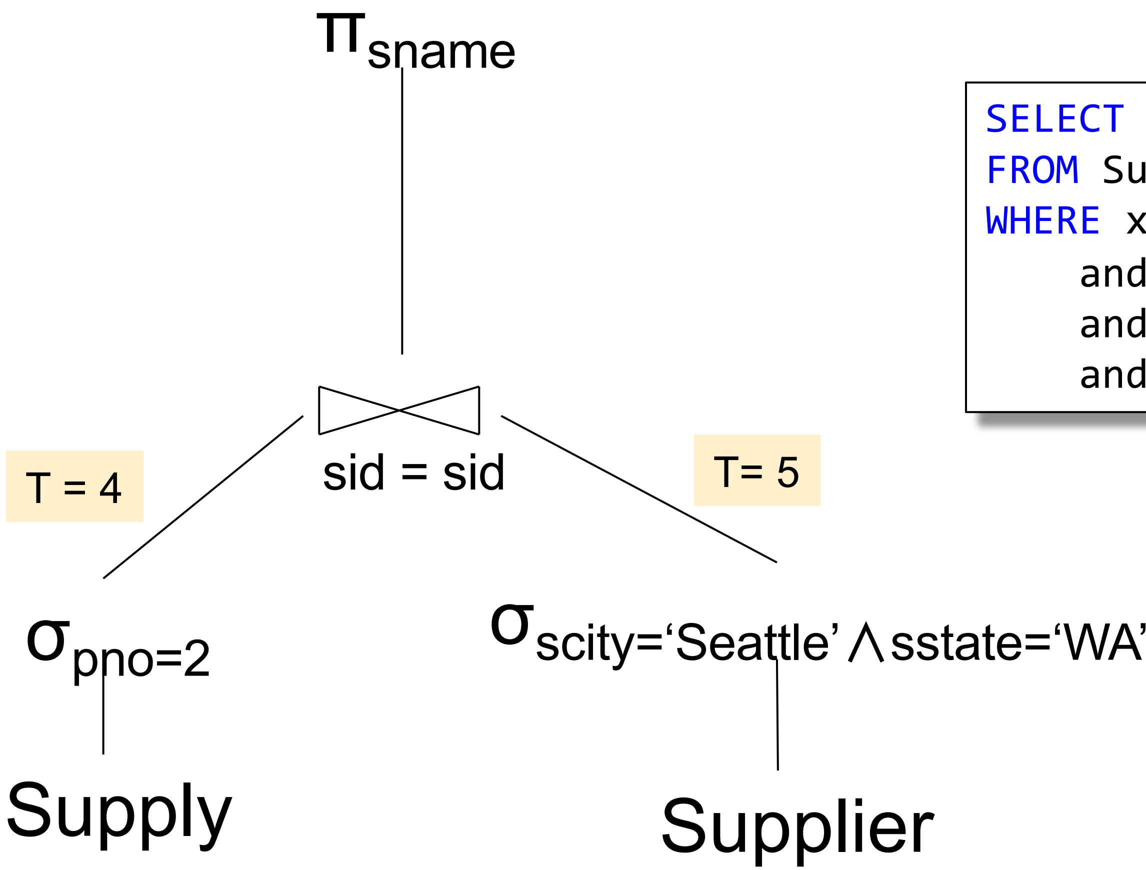
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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’
```

π_{sname}

$\bowtie_{sid = sid}$

T = 4

T= 5

Very wrong!
Why?

$\sigma_{pno=2}$

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

Supply

Supplier

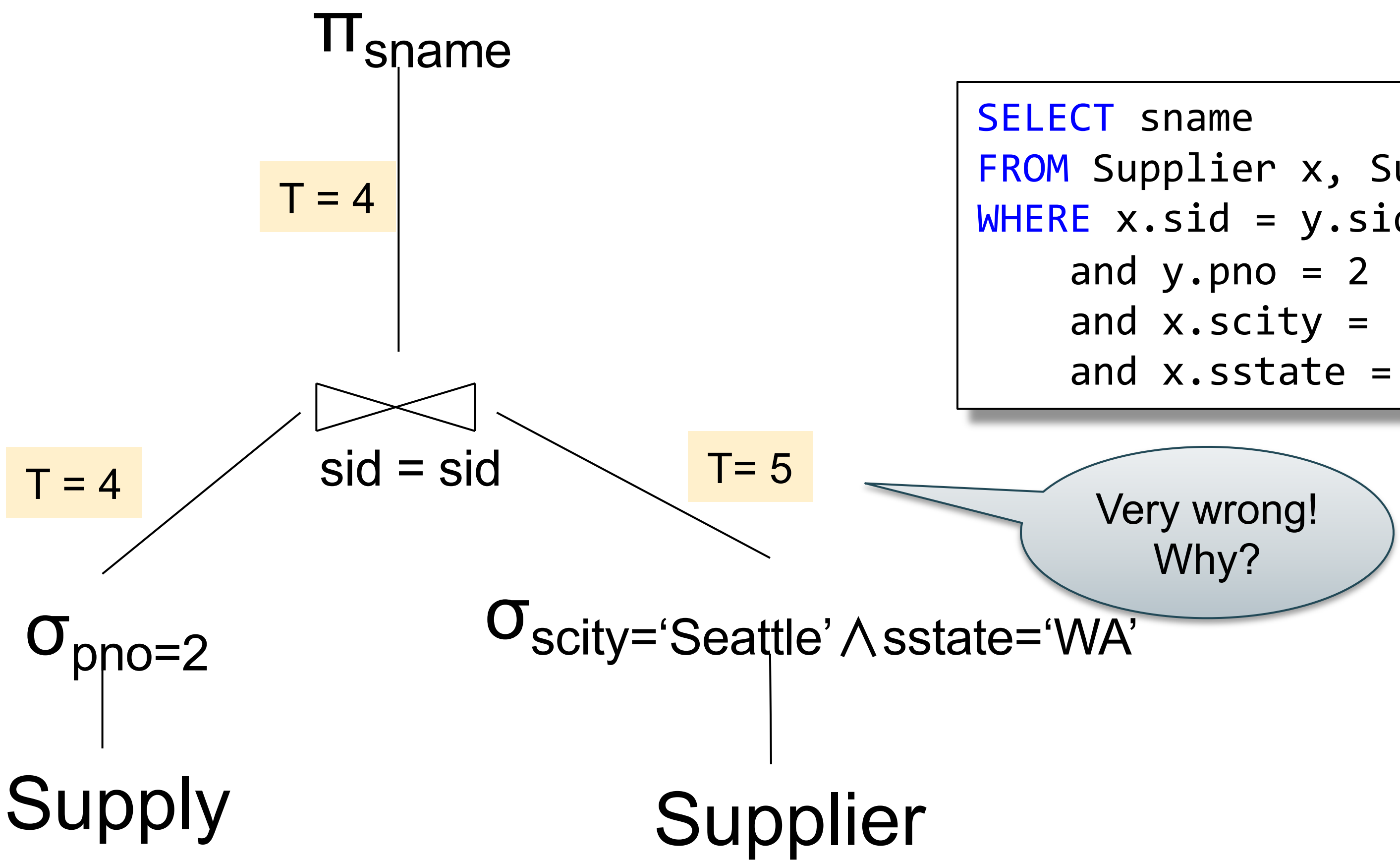
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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

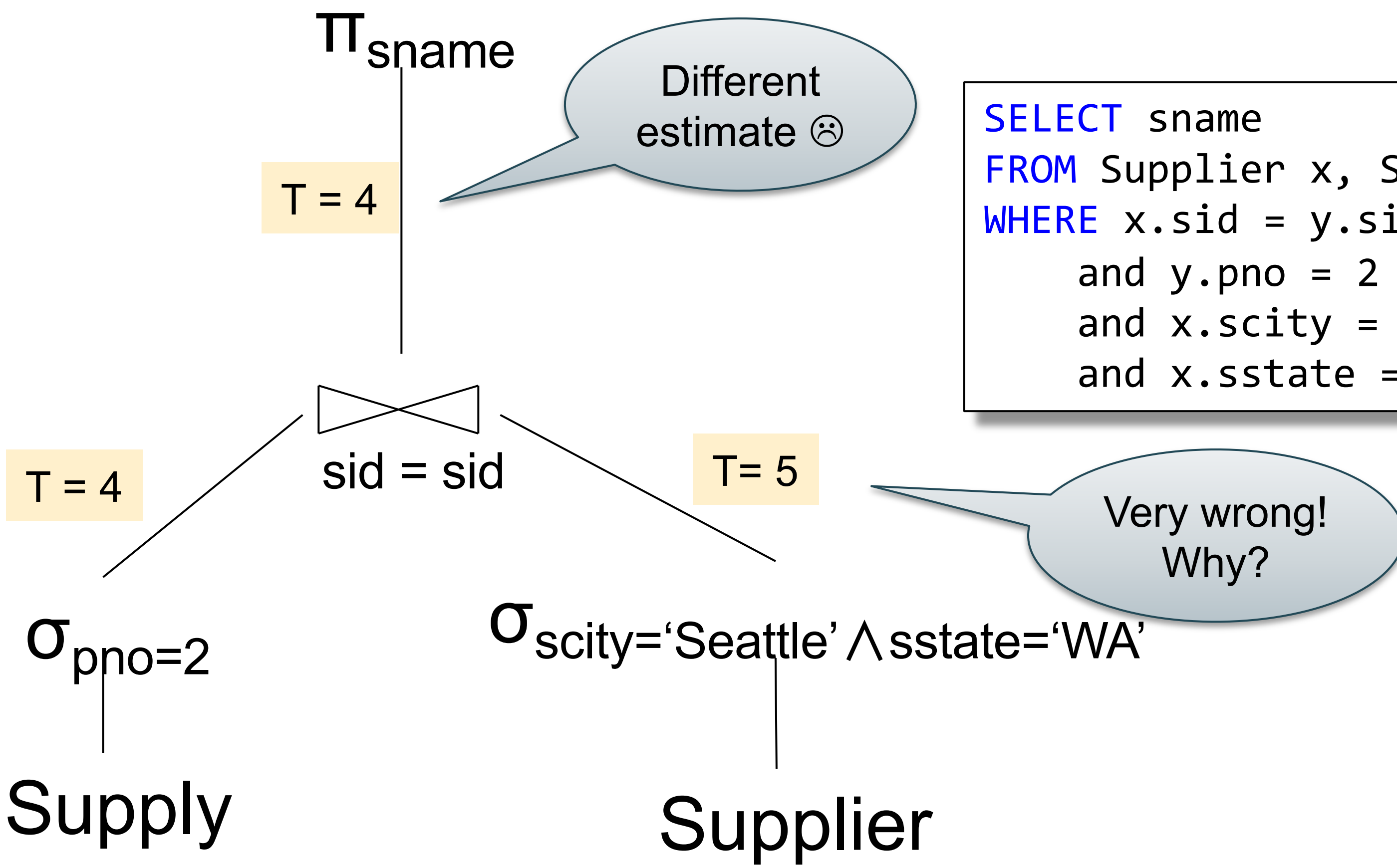
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

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'
```



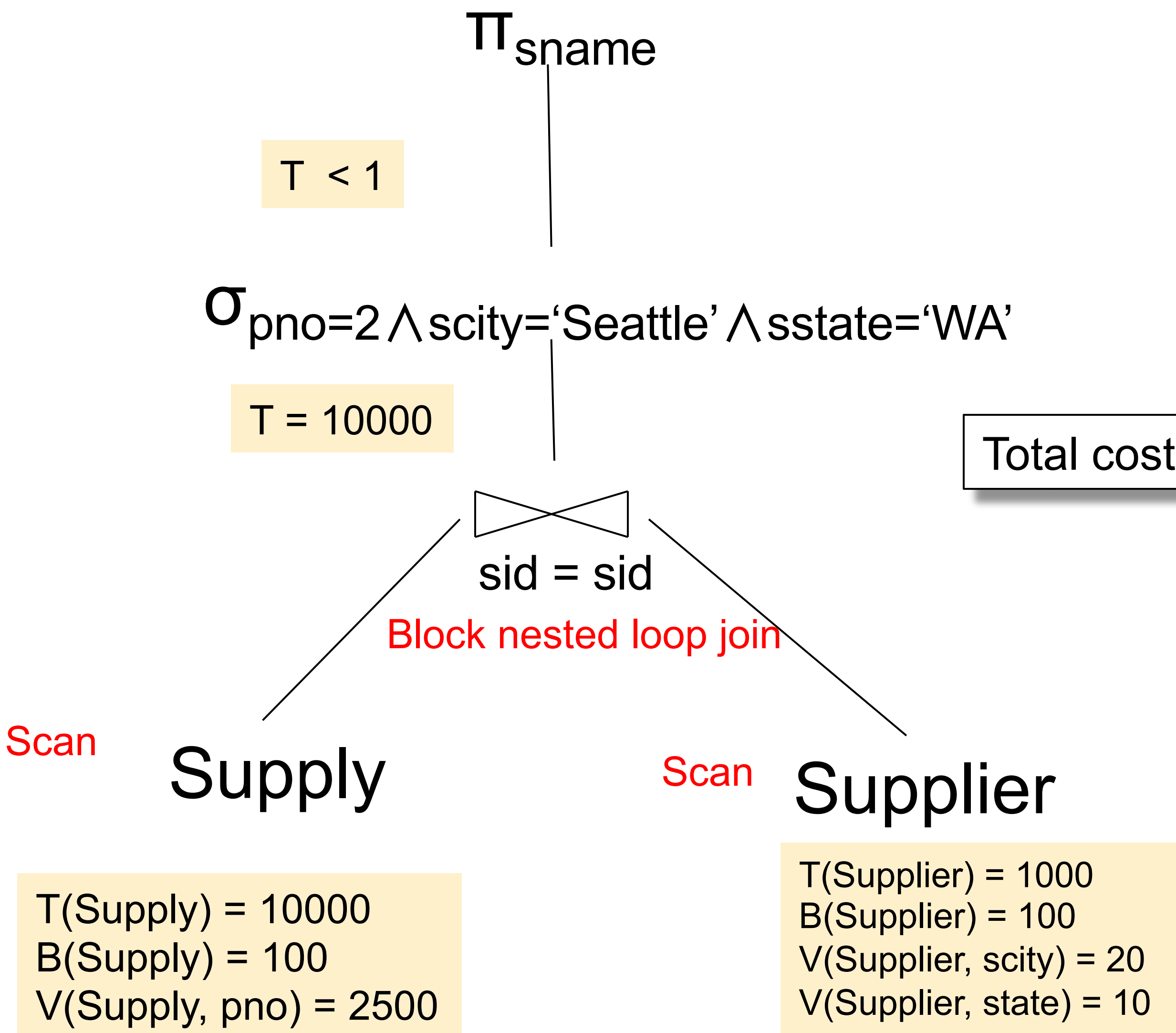
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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

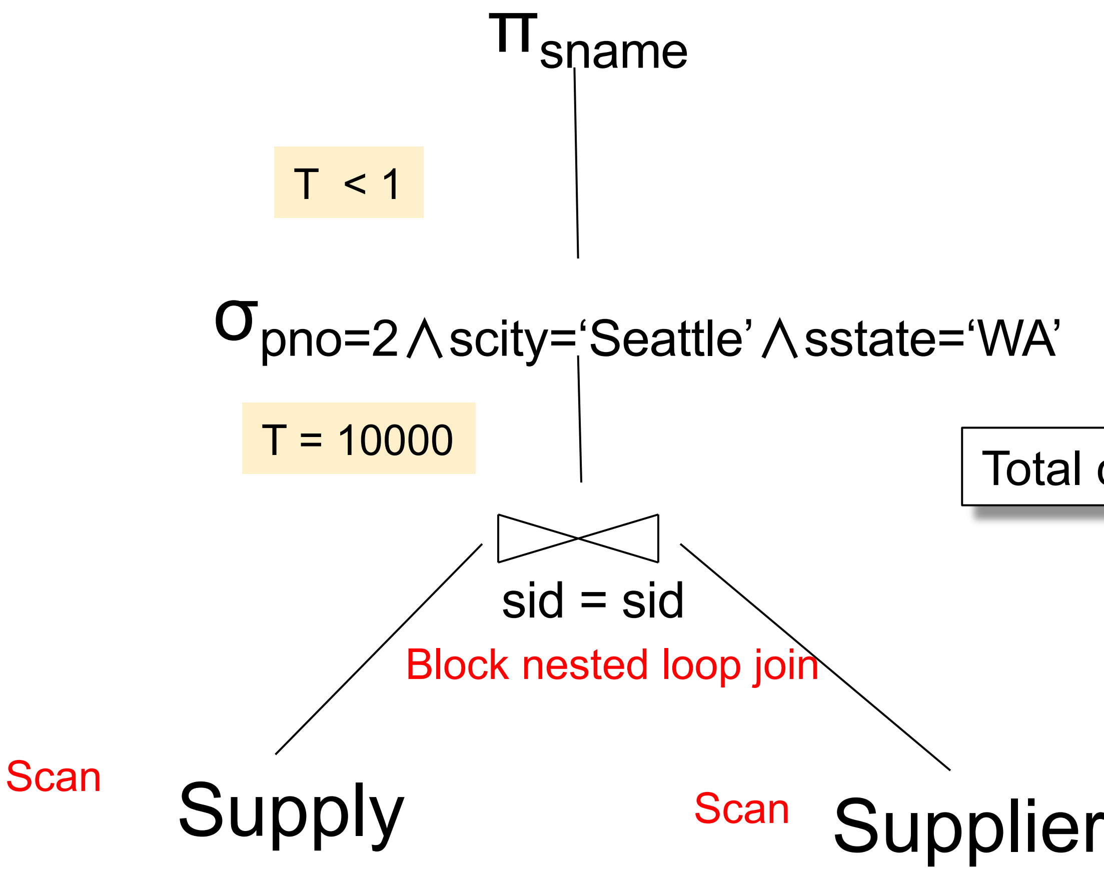


Total cost:

M=11

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

Physical Plan 1



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

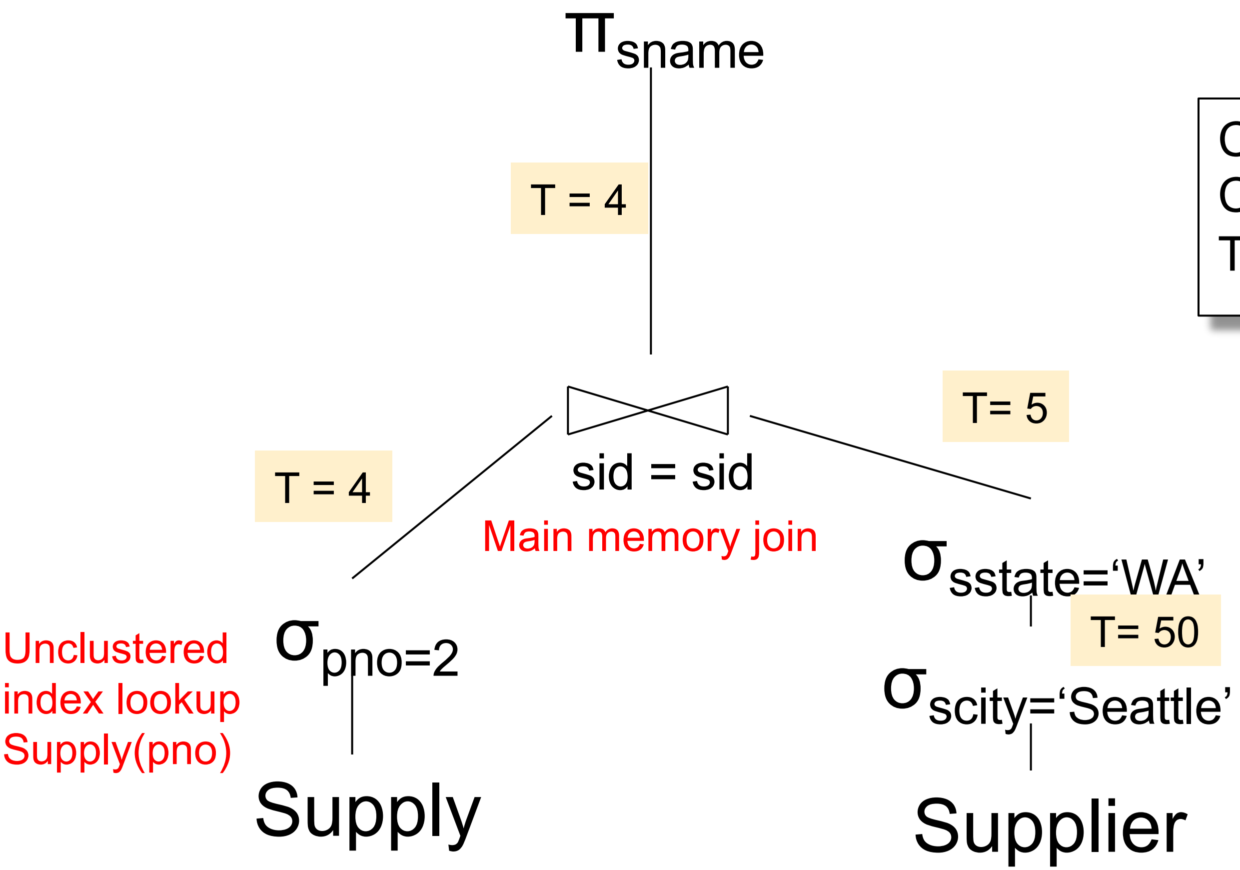
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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



Unclustered index lookup Supplier(scity)

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

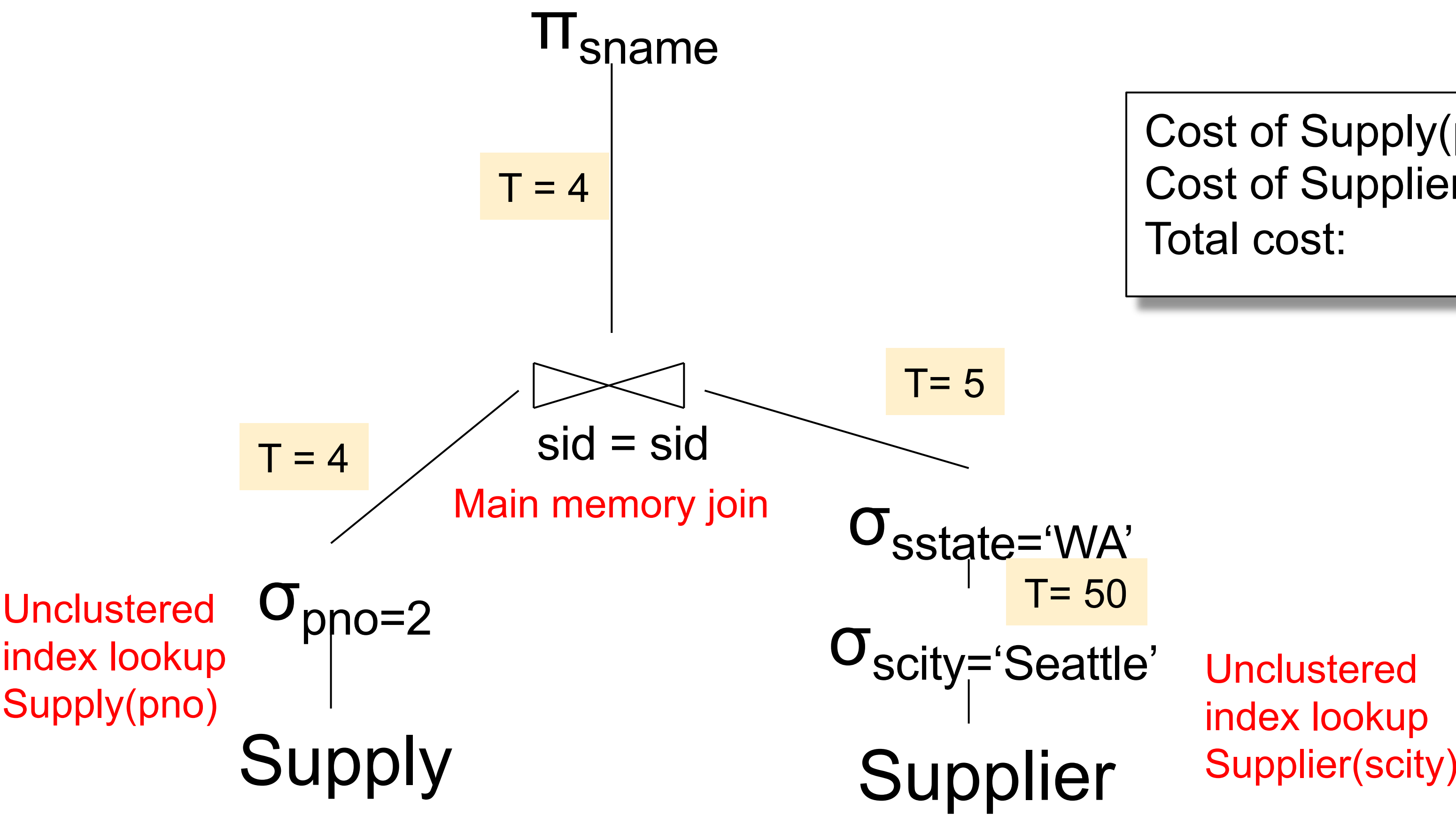
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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



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

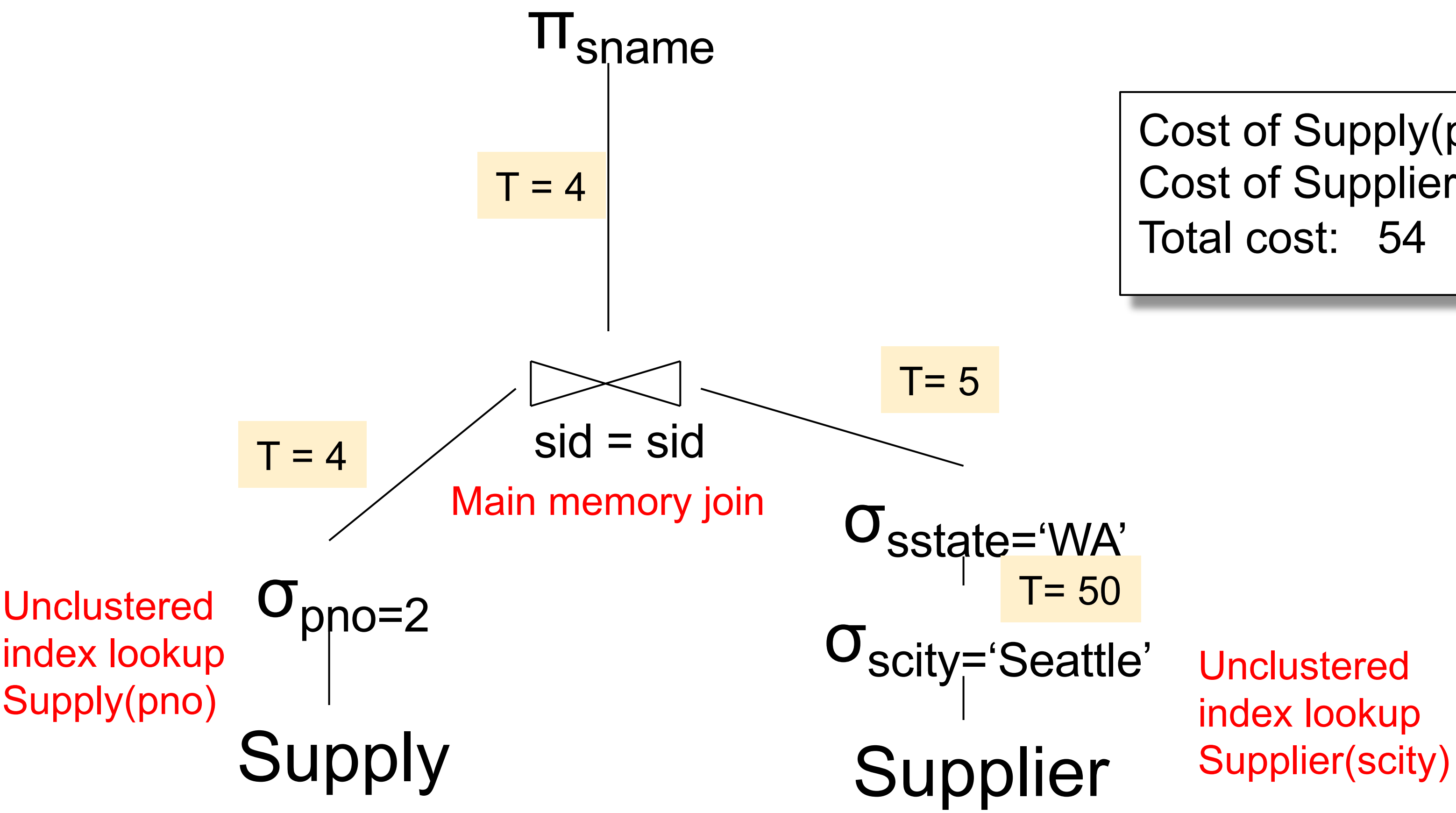
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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



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

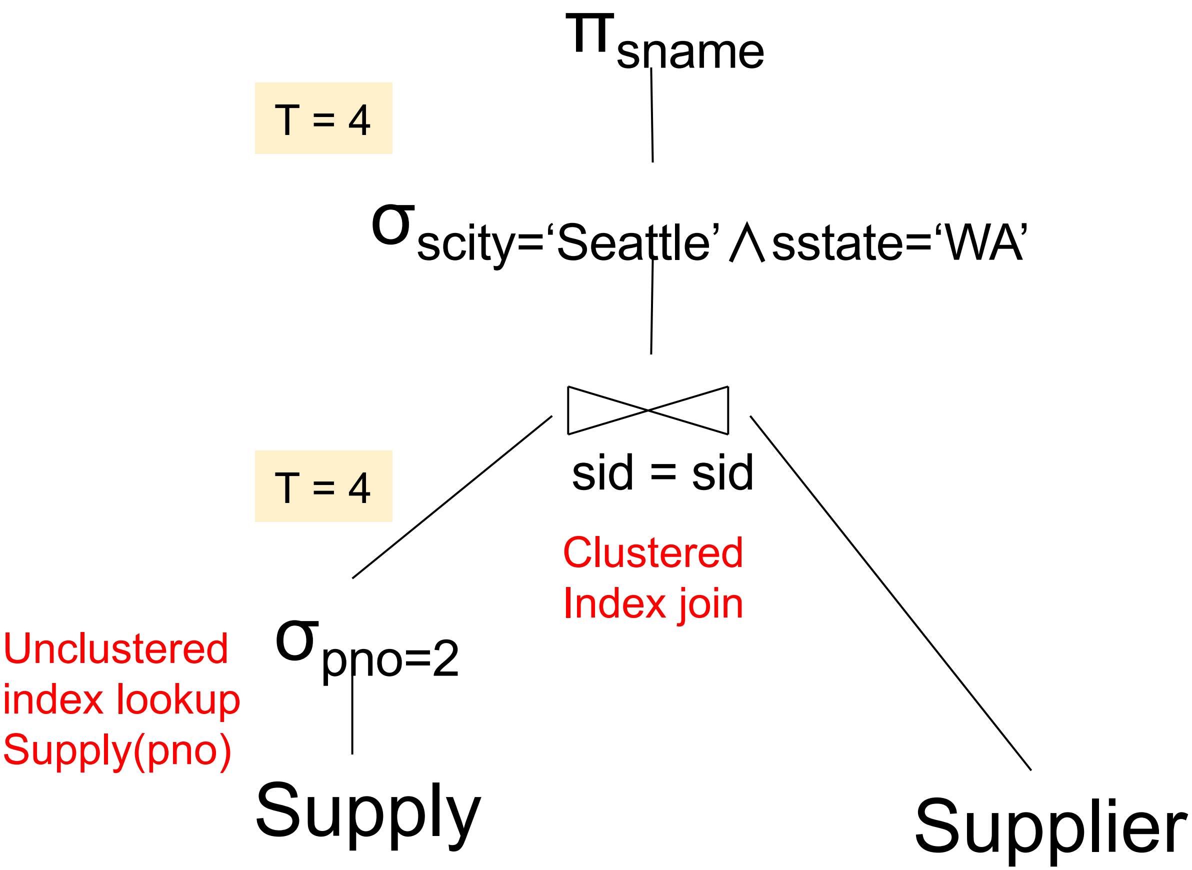
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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



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

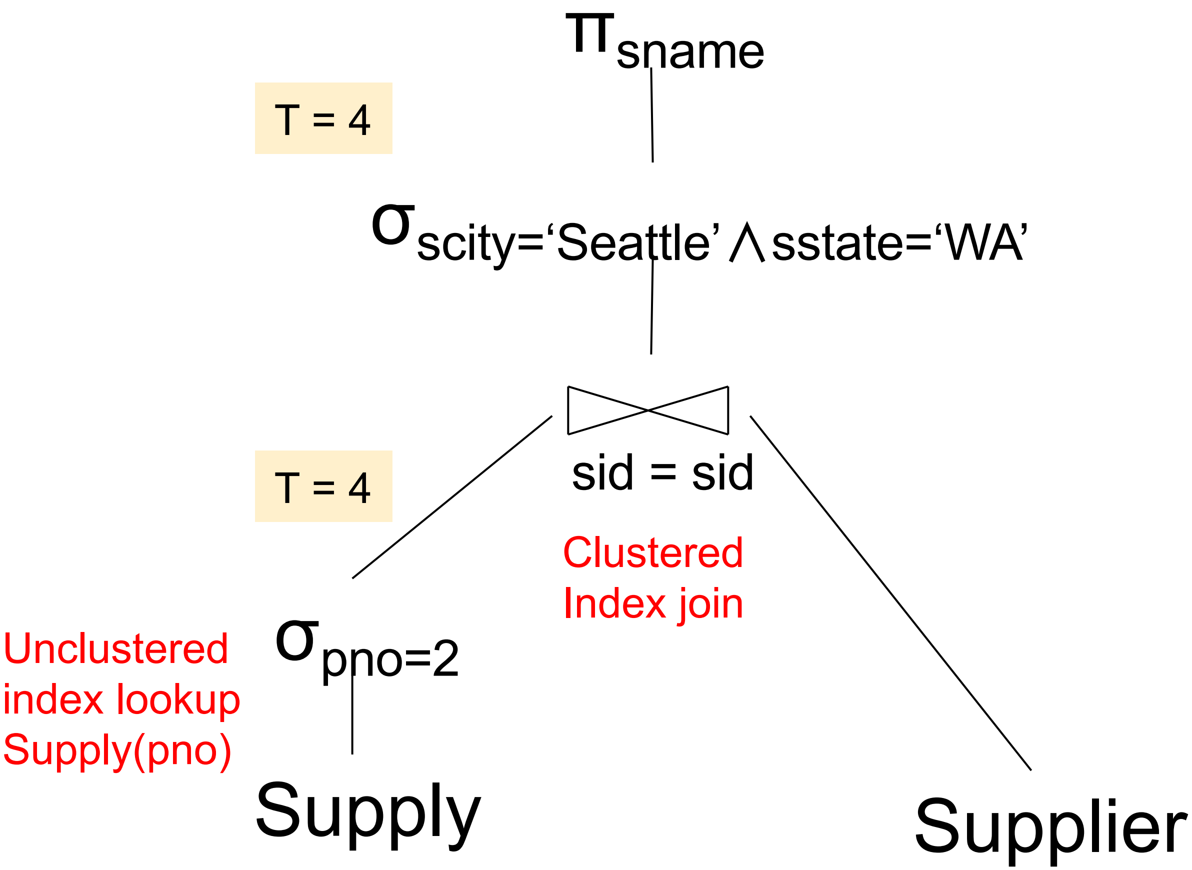
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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



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

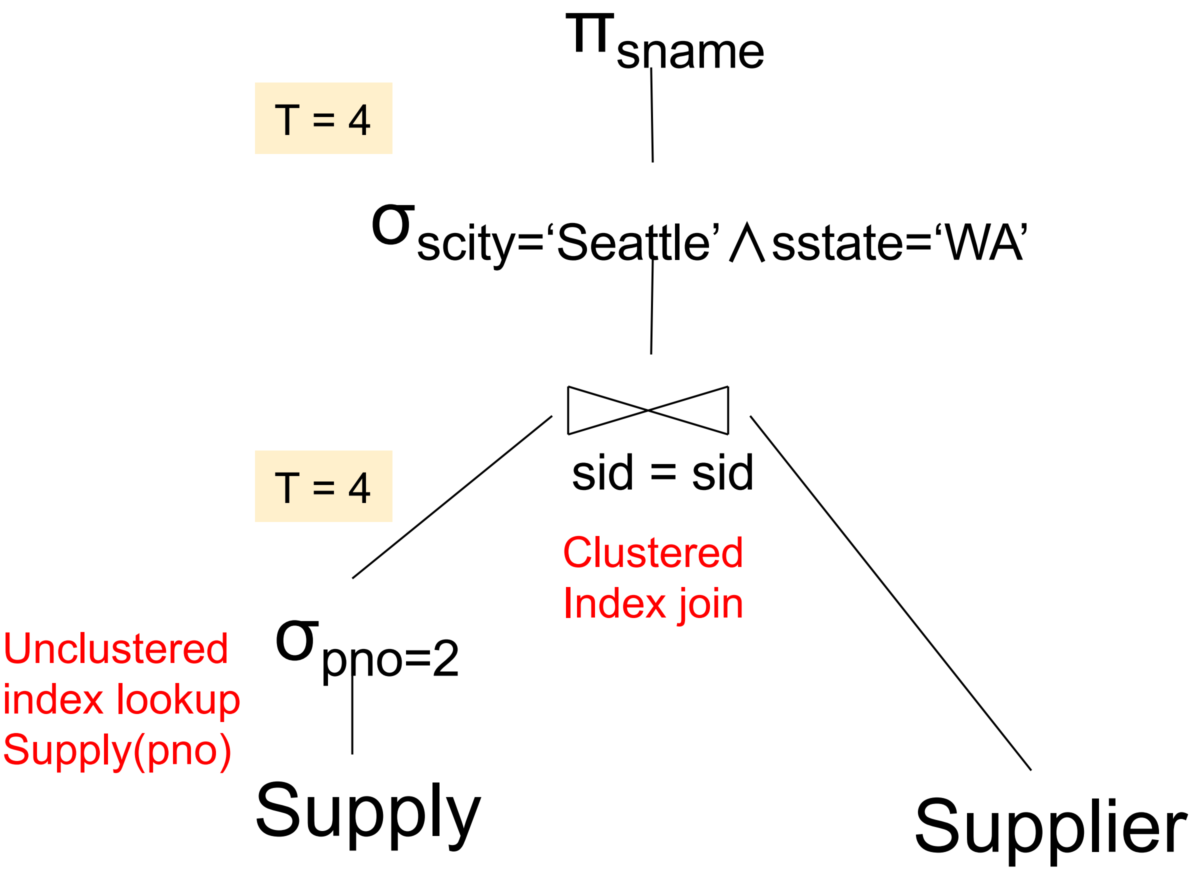
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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



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

Simplifications

- We considered only IO cost; in general we need IO+CPU
- We assumed that all index pages were in memory: sometimes we need to add the cost of fetching index pages from disk

Histograms

- Statistics on data maintained by the RDBMS
- Makes size estimation much more accurate (hence, cost estimations are more accurate)

Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50
min(age) = 19, max(age) = 68

$\sigma_{age=48}$(Empolyee) = ? $\sigma_{age>28 \text{ and } age<35}$(Empolyee) = ?

Histograms

Employee(<u>ssn</u>, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50
min(age) = 19, max(age) = 68

$\sigma_{age=48}$(Empolyee) = ? $\sigma_{age>28 \text{ and } age<35}$(Empolyee) = ?

Estimate = 25000 / 50 = 500 Estimate = 25000 * 6 / 50 = 3000

Histograms

Employee(<u>ssn</u>, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50
min(age) = 19, max(age) = 68

$\sigma_{age=48}$(Empolyee) = ? $\sigma_{age>28 \text{ and } age<35}$(Empolyee) = ?

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

Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50
min(age) = 19, max(age) = 68

$\sigma_{age=48}$(Empolyee) = ? $\sigma_{age>28 \text{ and } age<35}$(Empolyee) = ?

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

Estimate = 1200 Estimate = 1*80 + 5*500 = 2580

Types of Histograms

- How should we determine the bucket boundaries in a histogram ?

Types of Histograms

- How should we determine the bucket boundaries in a histogram ?

- Eq-Width
- Eq-Depth
- Compressed
- V-Optimal histograms

Employee(ssn, name, age)

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..20	20..29	30-39	40-49	50-59	> 60
Tuples	1800	2000	2100	2200	1900	1800

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

V-Optimal Histograms

- Defines bucket boundaries in an optimal way, to minimize the error over all point queries
- Computed rather expensively, using dynamic programming
- Modern databases systems use V-optimal histograms or some variations

Discuss the paper

- Why do they use the IMDB database instead of TPC-H?
- Do cardinality estimators typically under- or over-estimate?
- From cardinality to cost: how critical is that?

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

Discuss histograms v.s. samples

Single Table Estimation

- 1d Histograms: accurate for selection on a single equality or range predicate; poor for multiple predicates; useless for LIKE
- Samples: great for correlations, or predicates like LIKE; poor for low selectivity predicates: estimate is 0, then use ”magic constants”

Joins (0 to 6)

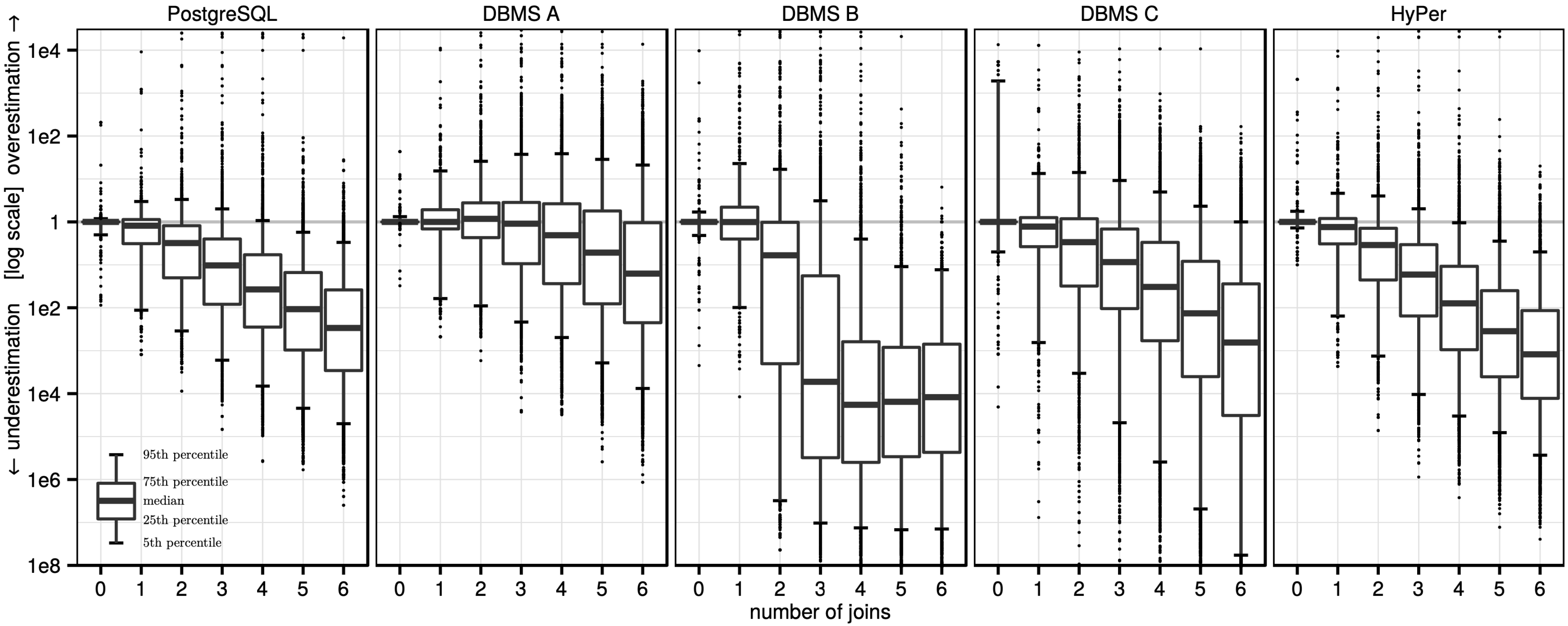
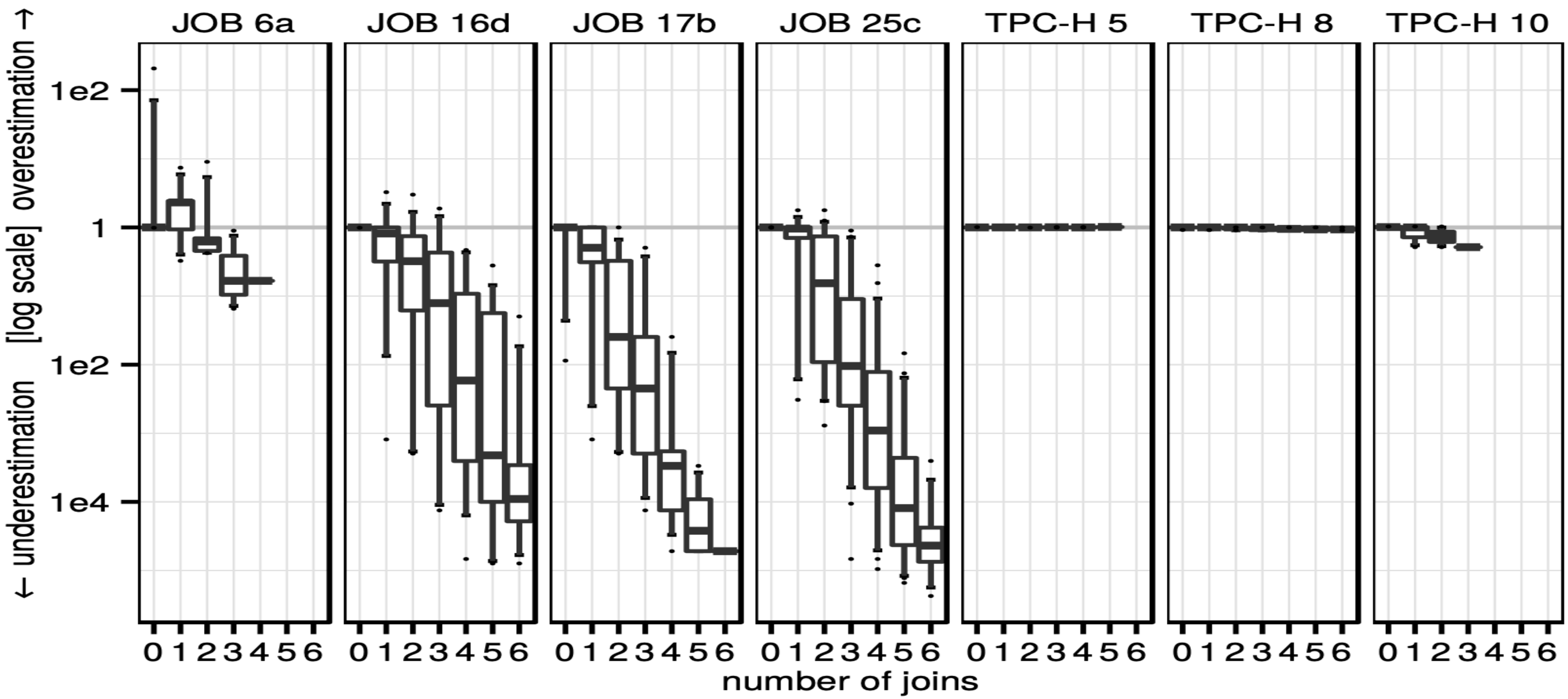


Figure 3: Quality of cardinality estimates for multi-join queries in comparison with the true cardinalities. Each boxplot summarizes the error distribution of all subexpressions with a particular size (over all queries in the workload)

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



[How good are they]

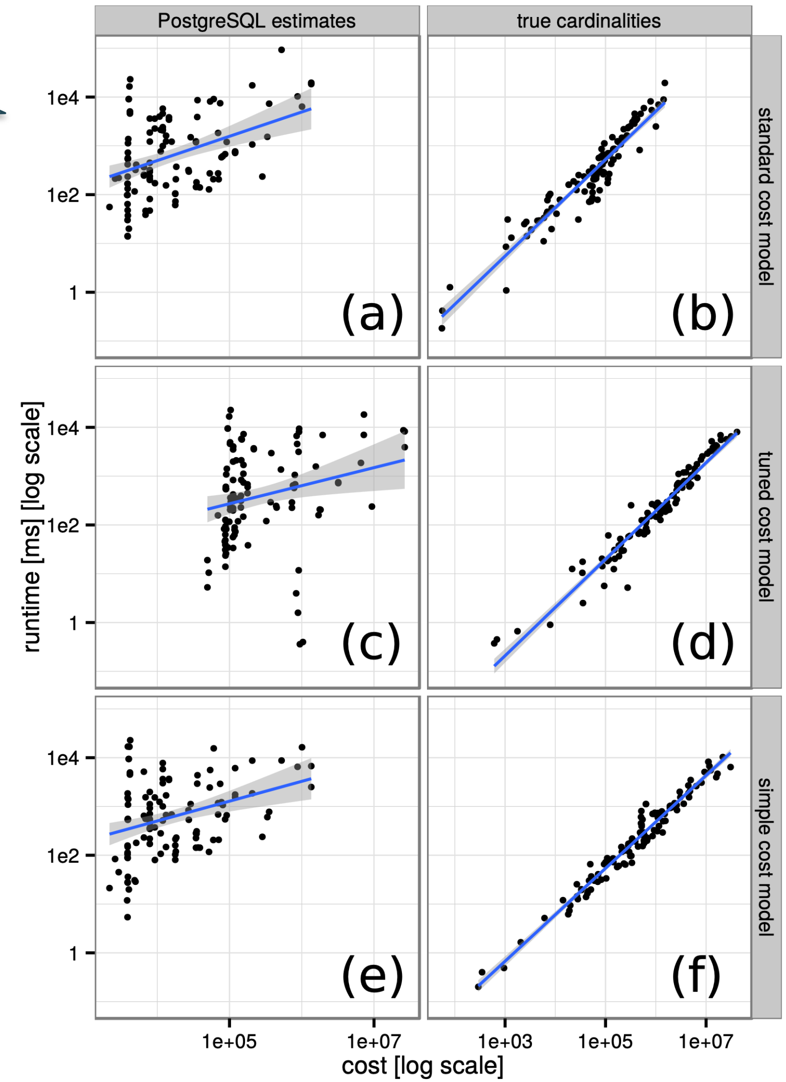
Cardinalities to Cost

- Cardinality estimation creates largest errors
- Complex or simple cost models don't differ much

Postgres cost

No I/O, keep only CPU

Their own simple formula

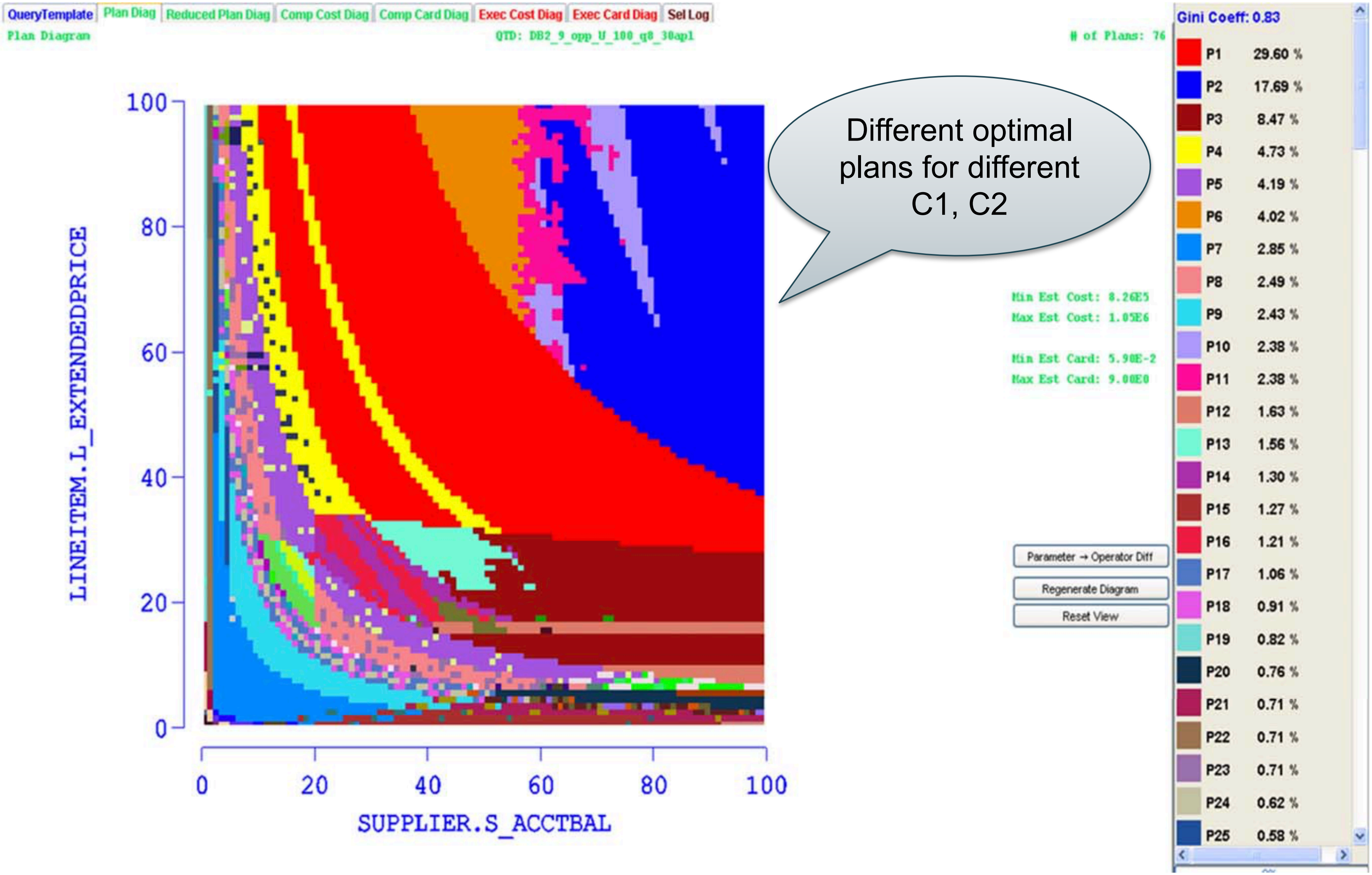


Yet Another Difficulties

- SQL Queries are often issued from applications
- Optimized once using *prepare* statement, executed often
- The constants in the query are not know until execution time: optimized plan may be suboptimal

```
select
  o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(volume)
from
 (select YEAR(o_orderdate) as o_year,
            l_extendedprice * (1 - l_discount) as volume,
                 n2.n_name as nation
  from part, supplier, lineitem, orders,
      customer, nation n1, nation n2, region
  where p_partkey = l_partkey and s_suppkey = l_suppkey
    and l_orderkey = o_orderkey and o_custkey = c_custkey
    and c_nationkey = n1.n_nationkey
    and n1.n_regionkey = r_regionkey
    and r_name = 'AMERICA'
    and s_nationkey = n2.n_nationkey
    and o_orderdate between '1995-01-01'
    and '1996-12-31'
    and p_type = 'ECONOMY ANODIZED STEEL'
   and s_acctbal ≤ C1 and l_extendedprice ≤ C2 ) as all_nations
group by o_year order by o_year
```

Optimize without knowing C1, C2


QueryTemplate
Plan Diag
Reduced Plan Diag
Comp Cost Diag
Comp Card Diag
Exec Cost Diag
Exec Card Diag
Sel Log
Plan Diagram
QTD: DB2_9_opp_U_100_q8_30apl
# of Plans: 76
LINEITEM.L_EXTENDEDPRICE
SUPPLIER.S_ACCTBAL
0
20
40
60
80
100
Different optimal plans for different C1, C2
Min Est Cost: 8.26E5
Max Est Cost: 1.05E6
Min Est Card: 5.90E-2
Max Est Card: 9.00E0
Parameter → Operator Diff
Regenerate Diagram
Reset View
Gini Coeff: 0.83
P1 29.60 %
P2 17.69 %
P3 8.47 %
P4 4.73 %
P5 4.19 %
P6 4.02 %
P7 2.85 %
P8 2.49 %
P9 2.43 %
P10 2.38 %
P11 2.38 %
P12 1.63 %
P13 1.56 %
P14 1.30 %
P15 1.27 %
P16 1.21 %
P17 1.06 %
P18 0.91 %
P19 0.82 %
P20 0.76 %
P21 0.71 %
P22 0.71 %
P23 0.71 %
P24 0.62 %
P25 0.58 %

Discussion

- Cardinality estimation = open problem
- Histograms:
  - Small number of buckets (why?)
  - Updated only periodically (why?)
  - No 2d histograms (except db2) why?
- Samples:
  - Fail for low selectivity estimates
  - Useless for joins
- Cross-join correlation – open problem

Query Optimization

<u>Three major components</u>:

1. Cardinality and cost estimation

2. Search space
   - Access path selection
   - Rewrite rules

3. Plan enumeration algorithms

Access Path

Access path: a way to retrieve tuples from a table

- A file scan, or

- An index *plus* a matching selection condition

Usually the access path implements a selection $\sigma_P(R)$, where the predicate P is called *search argument* SARG (see "architecture" paper)

Access Path Selection

Supplier(sid,sname,scity,sstate)

Selection condition: $sid > 300 \wedge scity='Seattle'$

Access Path Selection

Supplier(sid,sname,scity,sstate)

Selection condition: $sid > 300 \wedge scity=$'Seattle'

Indexes: clustered B+-tree on sid; B+-tree on scity

Access Path Selection

Supplier(sid,sname,scity,sstate)
Selection condition: sid > 300 $\wedge$ scity='Seattle'
Indexes: clustered B+-tree on sid; B+-tree on scity

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

Which access path should we use?

Access Path Selection

Supplier(sid,sname,scity,sstate)
Selection condition: sid > 300 $\wedge$ scity='Seattle'
Indexes: clustered B+-tree on sid; B+-tree on scity

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

Which access path should we use?

1. Sequential scan: cost = 100

Access Path Selection

Supplier(sid,sname,scity,sstate)
Selection condition: sid > 300 $\wedge$ scity='Seattle'
Indexes: clustered B+-tree on sid; B+-tree on scity

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

Which access path should we use?

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

Access Path Selection

Supplier(sid,sname,scity,sstate)
Selection condition: sid > 300 $\wedge$ scity='Seattle'
Indexes: clustered B+-tree on sid; B+-tree on scity

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

Which access path should we use?

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

Rewrite Rules

- The optimizer's search space is defined by the set of rewrite rules that it implements
- More rewrite rules means that more plans are being explored

Relational Algebra Laws

- Selections
  - Commutative: $\sigma_{c1}(\sigma_{c2}(R))$ same as $\sigma_{c2}(\sigma_{c1}(R))$
  - Cascading: $\sigma_{c1 \wedge c2}(R)$ same as $\sigma_{c2}(\sigma_{c1}(R))$

- Projections
  - Cascading

- Joins
  - Commutative : $R \bowtie S$ same as $S \bowtie R$
  - Associative: $R \bowtie (S \bowtie T)$ same as $(R \bowtie S) \bowtie T$

Selections and Joins

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

$\sigma_{A=v}(R(A,B) \bowtie_{B=C} S(C,D)) =$

Selections and Joins

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

$$\sigma_{A=v}(R(A,B) \bowtie_{B=C} S(C,D)) = \\ (\sigma_{A=v}(R(A,B))) \bowtie_{B=C} S(C,D)$$

The simplest optimizers use *only* this rule
Called *heuristic-based opimtizer*
In general: *cost-based optimizer*

Group-by and Join

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

$$\gamma_{A,\, sum(D)}(R(A,B) \bowtie_{B=C} S(C,D)) = \quad ?$$

Group-by and Join

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

$$\gamma_{A,\ sum(D)}(R(A,B) \bowtie_{B=C} S(C,D)) = \gamma_{A,\ sum(D)}(R(A,B) \bowtie_{B=C} (\gamma_{C,\ sum(D)} S(C,D)))$$

These are very powerful laws.
They were introduced only in the 90's.

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
```

What constraints do we need for correctness?

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
```

What constraints do we need for correctness?

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

1. Suppier.sid = key

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
```

What constraints do we need for correctness?

1. Suppier.sid = key
2. Supply.sid = foreign key

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
```

What constraints do we need for correctness?

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

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

Semi-Join Reduction

Semi-join definition:

$$\boxed{R \ltimes S = \Pi_{attr(R)}(R \bowtie S)}$$

Basic law:

$$\boxed{\Pi_{attr(R)}(R \bowtie S) = \Pi_{attr(R)}((R \ltimes S) \bowtie S)}$$

Example 1

- Example:

$$Q = R(A,B) \bowtie S(B,C)$$

Example 1

- Example:

$$\boxed{Q = R(A,B) \bowtie S(B,C)}$$

- A semijoin reducer is:

$$\boxed{R_1(A,B) = R(A,B) \ltimes S(B,C)}$$

Example 1

- Example:

$$Q = R(A,B) \bowtie S(B,C)$$

- A semijoin reducer is:

$$R_1(A,B) = R(A,B) \ltimes S(B,C)$$

- The rewritten query is:

$$Q = R_1(A,B) \bowtie S(B,C)$$

Example 2

Q(y,z,u) = R('a', y), S(y,z), T(z,u), K(u,'b')

Semi-join reducer:

Example 2

Q(y,z,u) = R('a', y), S(y,z), T(z,u), K(u,'b')

Semi-join reducer:

S'(y,z) :- S(y,z) ⋉ R('a', y)

Example 2

$Q(y,z,u) = R('a', y), S(y,z), T(z,u), K(u,'b')$

Semi-join reducer:

$S'(y,z) \text{ :- } S(y,z) \ltimes R('a', y)$
$T'(z,u) \text{ :- } T(z,u) \ltimes S'(y,z)$

Example 2

Q(y,z,u) = R('a', y), S(y,z), T(z,u), K(u,'b')

Semi-join reducer:

S'(y,z) :- S(y,z) $\ltimes$ R('a', y)
T'(z,u) :- T(z,u) $\ltimes$ S'(y,z)
K'(u) :- K(u,'b') $\ltimes$ T'(z,u)

Example 2

Q(y,z,u) = R('a', y), S(y,z), T(z,u), K(u,'b')

Semi-join reducer:

S'(y,z) :- S(y,z) ⋉ R('a', y)
T'(z,u) :- T(z,u) ⋉ S'(y,z)
K'(u) :- K(u,'b') ⋉ T'(z,u)
T''(z,u) :- T'(z,u) ⋉ K'(u)

Example 2

Q(y,z,u) = R('a', y), S(y,z), T(z,u), K(u,'b')

Semi-join reducer:

S'(y,z) :- S(y,z) $\ltimes$ R('a', y)
T'(z,u) :- T(z,u) $\ltimes$ S'(y,z)
K'(u) :- K(u,'b') $\ltimes$ T'(z,u)
T''(z,u) :- T'(z,u) $\ltimes$ K'(u)
S''(y,z) :- S'(y,z) $\ltimes$ T''(z,u)
R''(y) :- R('a',y) $\ltimes$ S''(y,z)

Example 2

$Q(y,z,u) = R(\text{'a'}, y), S(y,z), T(z,u), K(u,\text{'b'})$

Semi-join reducer:

$S'(y,z)$:- $S(y,z) \ltimes R(\text{'a'}, y)$
$T'(z,u)$:- $T(z,u) \ltimes S'(y,z)$
$K'(u)$:- $K(u,\text{'b'}) \ltimes T'(z,u)$
$T''(z,u)$:- $T'(z,u) \ltimes K'(u)$
$S''(y,z)$:- $S'(y,z) \ltimes T''(z,u)$
$R''(y)$:- $R(\text{'a'},y) \ltimes S''(y,z)$

Reduced query:

$Q(y,z,u) = R''(y), S''(y,z), T''(z,u), K''(u)$

Search Space Challenges

- Search space is huge!
  - Many possible equivalent trees (logical)
  - Many implementations for each operator (physical)
  - Many access paths for each relation (physical)

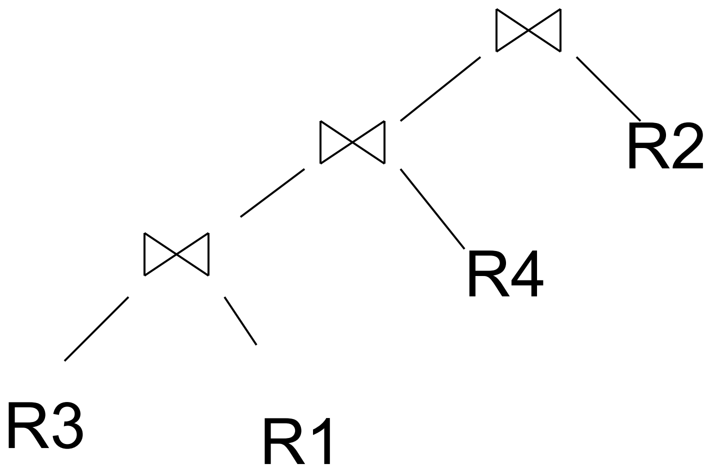
- Cannot consider ALL plans
- Want a search space that includes low-cost plans

- Typical compromises:
  - Only left-deep plans
  - Only plans without cartesian products
  - Always push selections down to the leaves

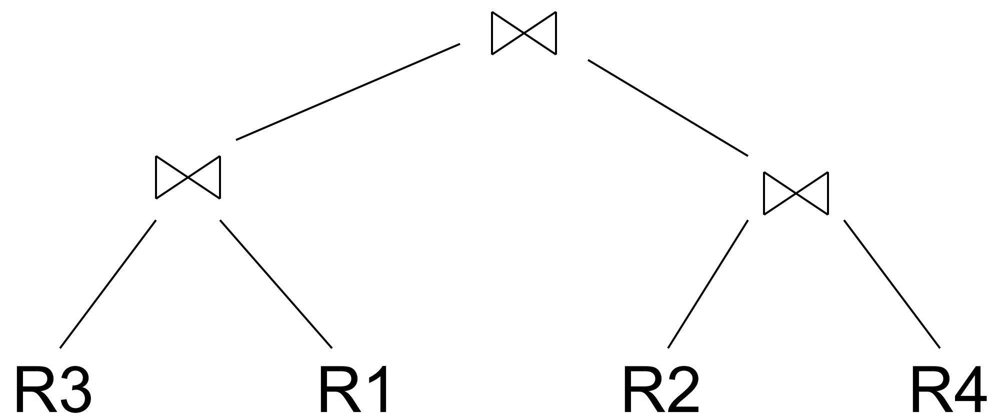
Practice

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

Left-Deep Plans and Bushy Plans



Left-deep plan



Bushy plan

[How good are they]

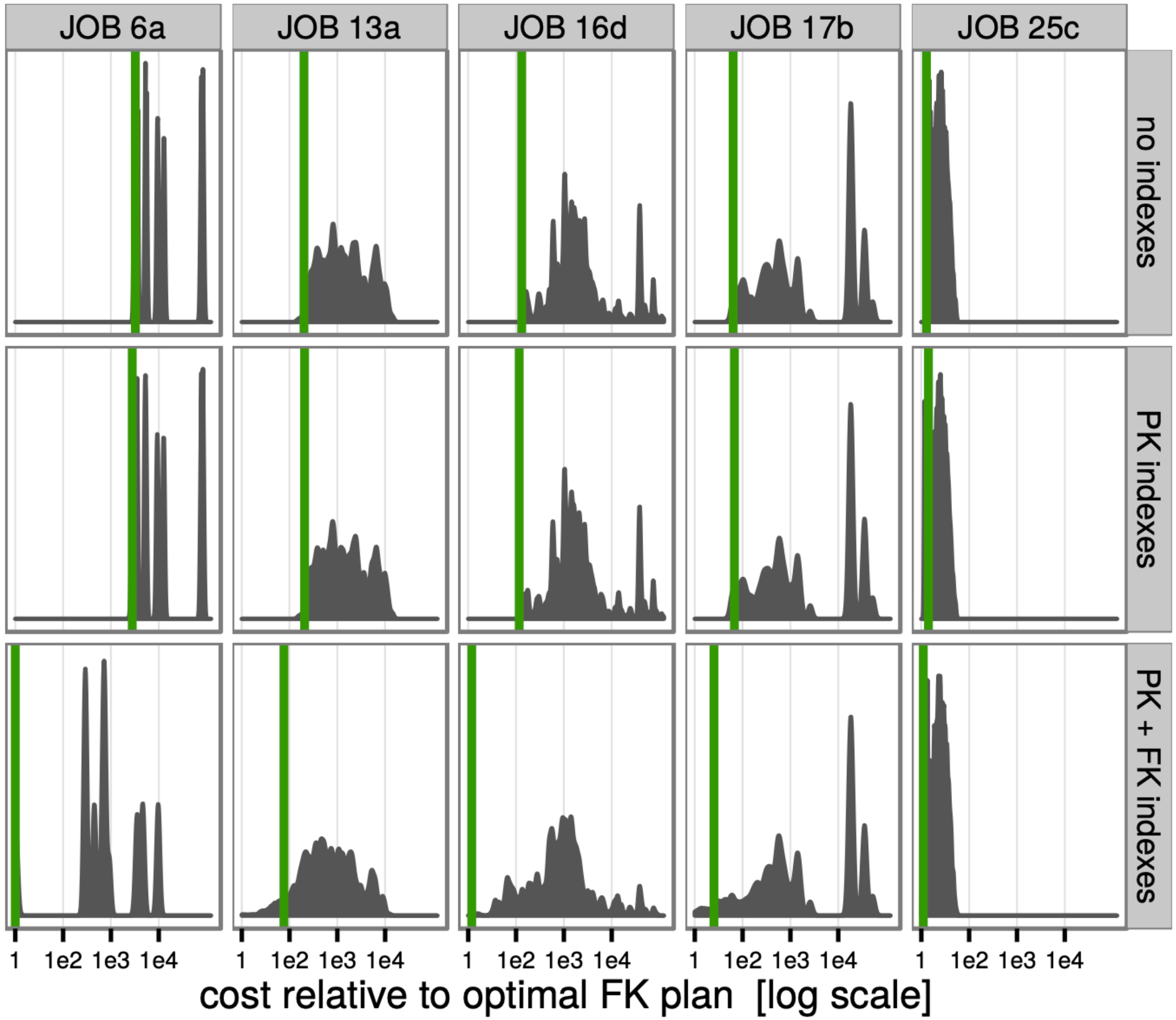


Figure 9: Cost distributions for 5 queries and different index configurations. The vertical green lines represent the cost of the optimal plan

[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)

Query Optimization

Three major components:

1. Cardinality and cost estimation

2. Search space

3. Plan enumeration algorithms

Two Types of Optimizers

- Heuristic-based optimizers:
  - Apply greedily rules that always improve plan
    - Typically: push selections down
  - Very limited: no longer used today

- Cost-based optimizers:
  - Use a cost model to estimate the cost of each plan
  - Select the “cheapest” plan
  - We focus on cost-based optimizers

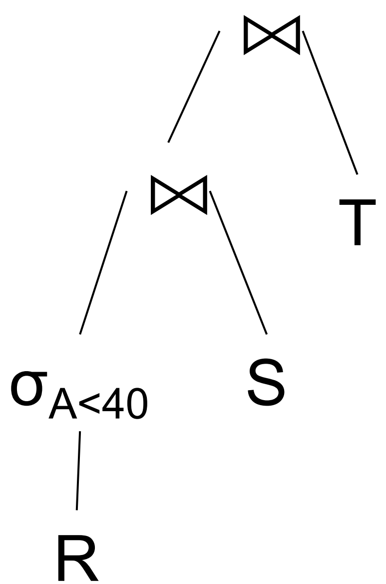
Three Approaches to Search Space Enumeration

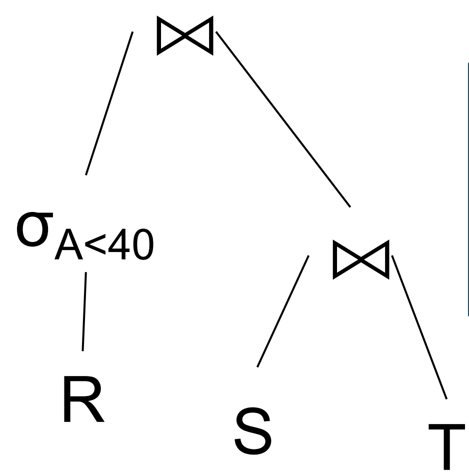
- Complete plans
- Bottom-up plans
- Top-down plans

Complete Plans

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<40
```





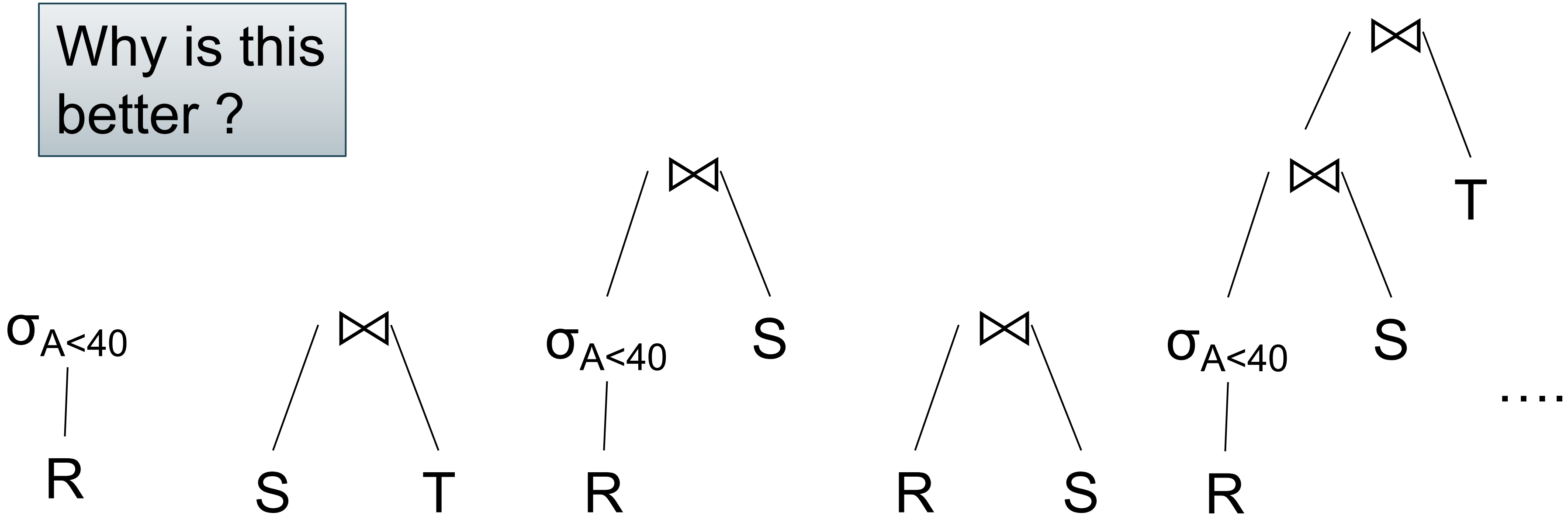
Why is this search space inefficient ?

Bottom-up Partial Plans

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<40
```

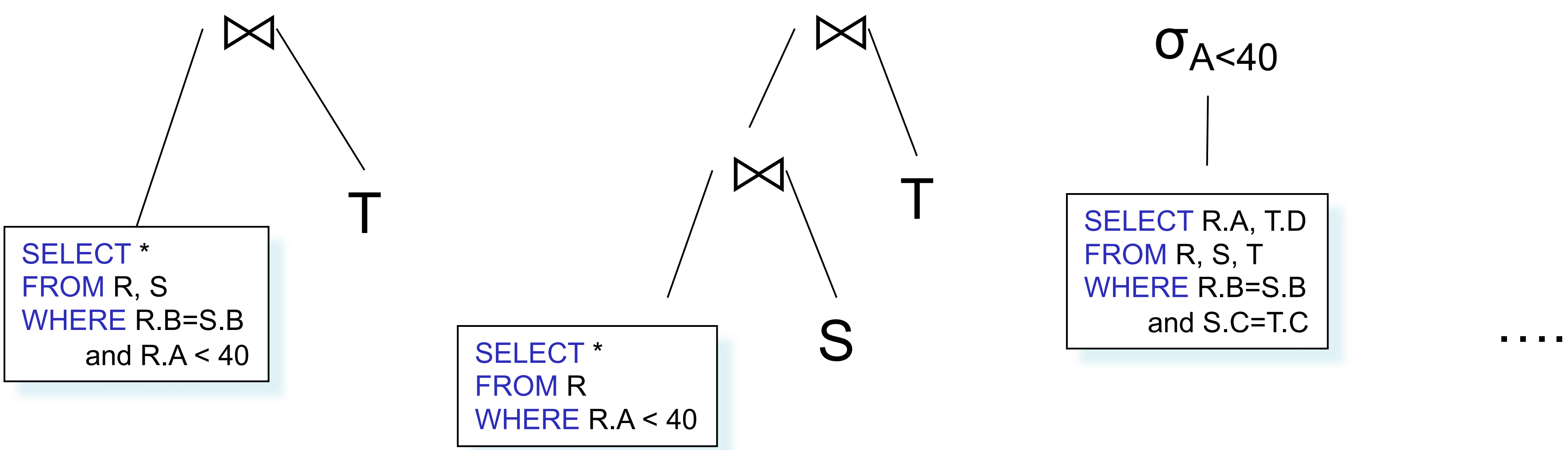
Why is this better ?



Top-down Partial Plans

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<40
```



Two Types of Plan Enumeration Algorithms

- Dynamic programming (in class)
  - Based on System R (aka Selinger) style optimizer[1979]
  - Limited to joins: *join reordering algorithm*
  - Bottom-up

- Rule-based algorithm (will not discuss)
  - Database of rules (=algebraic laws)
  - Usually: dynamic programming
  - Usually: top-down

System R Search Space (1979)

- Only left-deep plans
  - Enable dynamic programming for enumeration
  - Facilitate tuple pipelining from outer relation
- Consider plans with all “interesting orders”
- Perform cross-products after all other joins (heuristic)
- Only consider nested loop & sort-merge joins
- Consider both file scan and indexes
- Try to evaluate predicates early

System R Enumeration Algorithm

- Idea: use dynamic programming
- For each subset of {R1, …, Rn}, compute the best plan for that subset
- In increasing order of set cardinality:
  - Step 1: for {R1}, {R2}, …, {Rn}
  - Step 2: for {R1,R2}, {R1,R3}, …, {Rn-1, Rn}
  - …
  - Step n: for {R1, …, Rn}
- It is a bottom-up strategy
- A subset of {R1, …, Rn} is also called a *subquery*

Dynamic Programming Algo.

- For each subquery Q $\subseteq$ {R1, …, Rn} compute the following:
  - Size(Q)
  - A best plan for Q: Plan(Q)
  - The cost of that plan: Cost(Q)

Dynamic Programming Algo.

- **Step 1**: Enumerate all single-relation plans
  - Consider selections on attributes of relation
  - Consider all possible access paths
  - Consider attributes that are not needed
  - Compute cost for each plan
  - Keep cheapest plan per "interesting" output order

Dynamic Programming Algo.

- **Step 2**: Generate all two-relation plans
  - For each each single-relation plan from step 1
  - Consider that plan as outer relation
  - Consider every other relation as inner relation

  - Compute cost for each plan

  - Keep cheapest plan per "interesting" output order

Dynamic Programming Algo.

- **Step 3**: Generate all three-relation plans
  - For each each two-relation plan from step 2
  - Consider that plan as outer relation
  - Consider every other relation as inner relation
  - Compute cost for each plan
  - Keep cheapest plan per “interesting” output order
- **Steps 4 through n**: repeat until plan contains all the relations in the query

Commercial Query Optimizers

DB2, Informix, Microsoft SQL Server, Oracle 8

- Inspired by System R
  - Left-deep plans and dynamic programming
  - Cost-based optimization (CPU and IO)

- Go beyond System R style of optimization
  - Also consider right-deep and bushy plans (e.g., Oracle and DB2)
  - Variety of additional strategies for generating plans (e.g., DB2 and SQL Server)

Other Query Optimizers

- Randomized plan generation
  - Genetic algorithm
  - PostgreSQL uses it for queries with many joins

- Rule-based
  - ***Extensible*** collection of rules
  - Rule = Algebraic law with a direction
  - Algorithm for firing these rules
    - Generate many alternative plans, in some order
    - Prune by cost
  - Startburst (later DB2) and Volcano (later SQL Server)

[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

Query Optimization: Conclusions

- Query optimizer = critical part of DBMS
- ”Avoid a very bad plan” instead of “find the optimal plan”
- Size estimation + search space + algo
- Essential:
  - set-at-a-time language
  - order-independent

Next time: asymptotic complexity of query evaluation