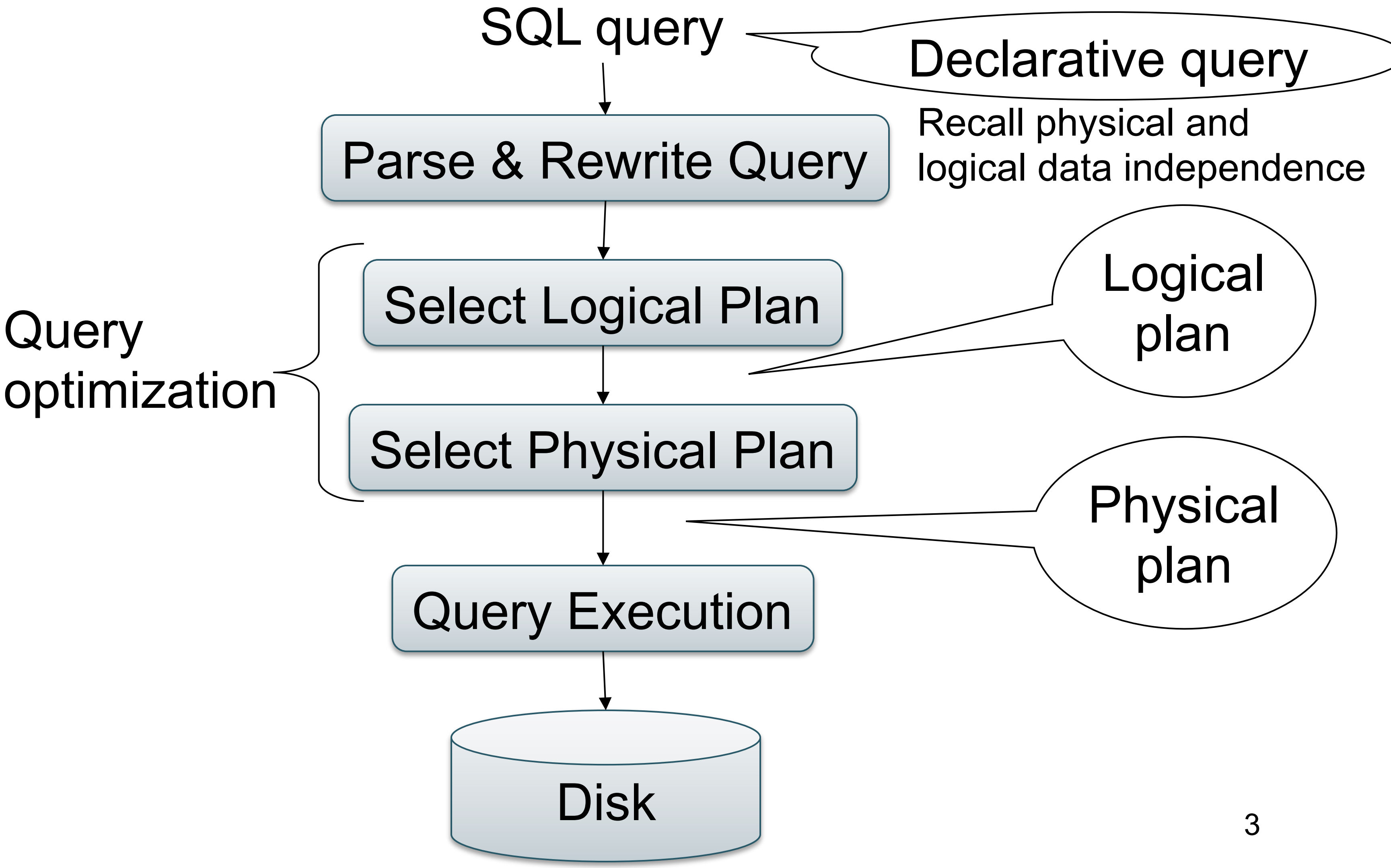
CSE 544
Principles of Database Management Systems

Lectures 9 -10: Query optimization

Announcements

- HW3 (SimpleDB) is due next Friday!
- Reading assignment was due today

Query Optimization Motivation



SQL query

Declarative query

Recall physical and
logical data independence

Parse & Rewrite Query

Query
optimization

Select Logical Plan

Logical
plan

Select Physical Plan

Physical
plan

Query Execution

Disk

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

Discussion of the Paper

- Query parsing/authorization
- Query rewriting:
  - Is salary < 75k and salary > 100k implausible?
  - What is *semantic optimization*?
- Query optimizer
  - Will discuss in detail...
  - What is query re-optimization?
    Predictable performance (IBM) v.s. self-tuning (Microsoft)
  - What is the "halloween problem"?
- Query execution
  - What are BP-tuples v.s. M-tuples? What is the *pin-count?*
- Access methods: will discuss

Query Optimization

Three major components:

1. Cardinality and cost estimation
2. Search space
3. Plan enumeration algorithms

Estimating Cost of a Query Plan

Goal: compute the cost of an entire physical query plan

- We already know how to compute the cost of each physical operator if we knew the T(R) and B(R) for each of its arguments

- Goal: estimate T(R) for each intermediate result R
  B(R) can be derived from T(R)

Statistics on Base Data

- Collected information for each database relation
  - Number of tuples (cardinality) T(R)
  - Number of physical pages B(R), clustering info
  - Indexes, number of keys in the index V(R,a)
  - Statistical information on attributes
    - Min value, max value, number distinct values
    - Histograms
  - Correlations between columns (hard)

- Collection approach: periodic, using sampling

Size Estimation

Projection: output size same as input size

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

Selection: the size decreases by *selectivity factor* θ

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

Selectivity Factors

- A = c /* $\sigma_{A=c}(R)$ */
  - Selectivity = 1/V(R,A)
- A < c /* $\sigma_{A<c}(R)$*/
  - Selectivity = (c - min(R, A))/(max(R,A) - min(R,A))
- c1 < A < c2 /* $\sigma_{c1<A<c2}(R)$*/
  - Selectivity = (c2 – c1)/(max(R,A) - min(R,A))
- Multiple predicates: assume independence

Estimating Result Sizes

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

- Take product of cardinalities of relations 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

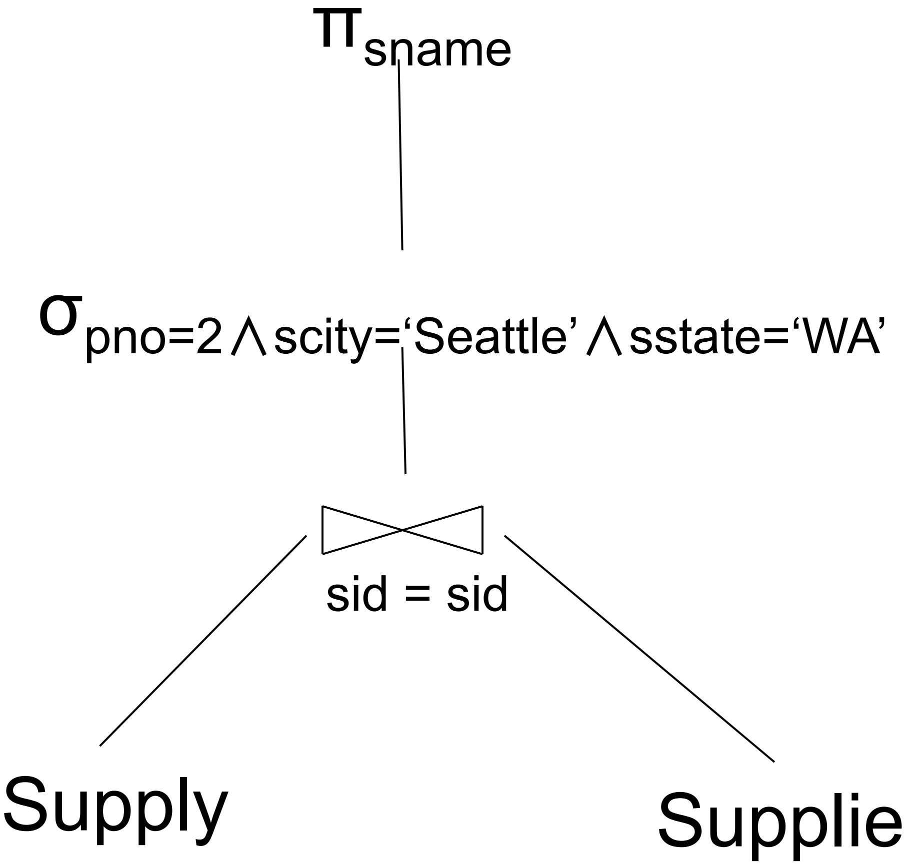
- Estimate cardinality in a bottom-up fashion
  - Cardinality is the size of a relation (nb of tuples)
  - Compute size of *all* intermediate relations in plan

- Estimate cost by using the estimated cardinalities

- Extensive example next...

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

Logical Query Plan 1



π_{sname}

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

$\bowtie_{\text{sid = sid}}$

Supply

Supplier

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

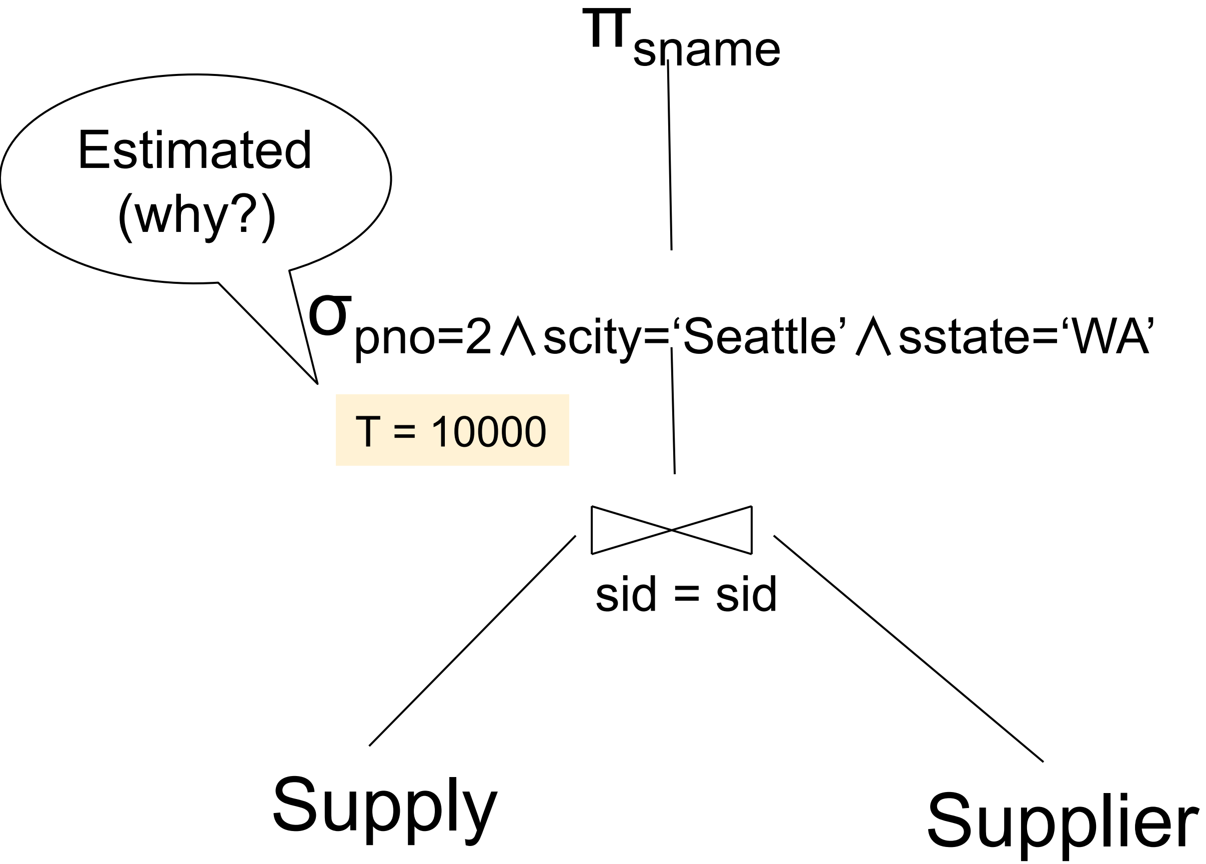
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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



π_{sname}

Estimated (why?)

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

T = 10000

$\bowtie_{sid = sid}$

Supply

Supplier

```
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

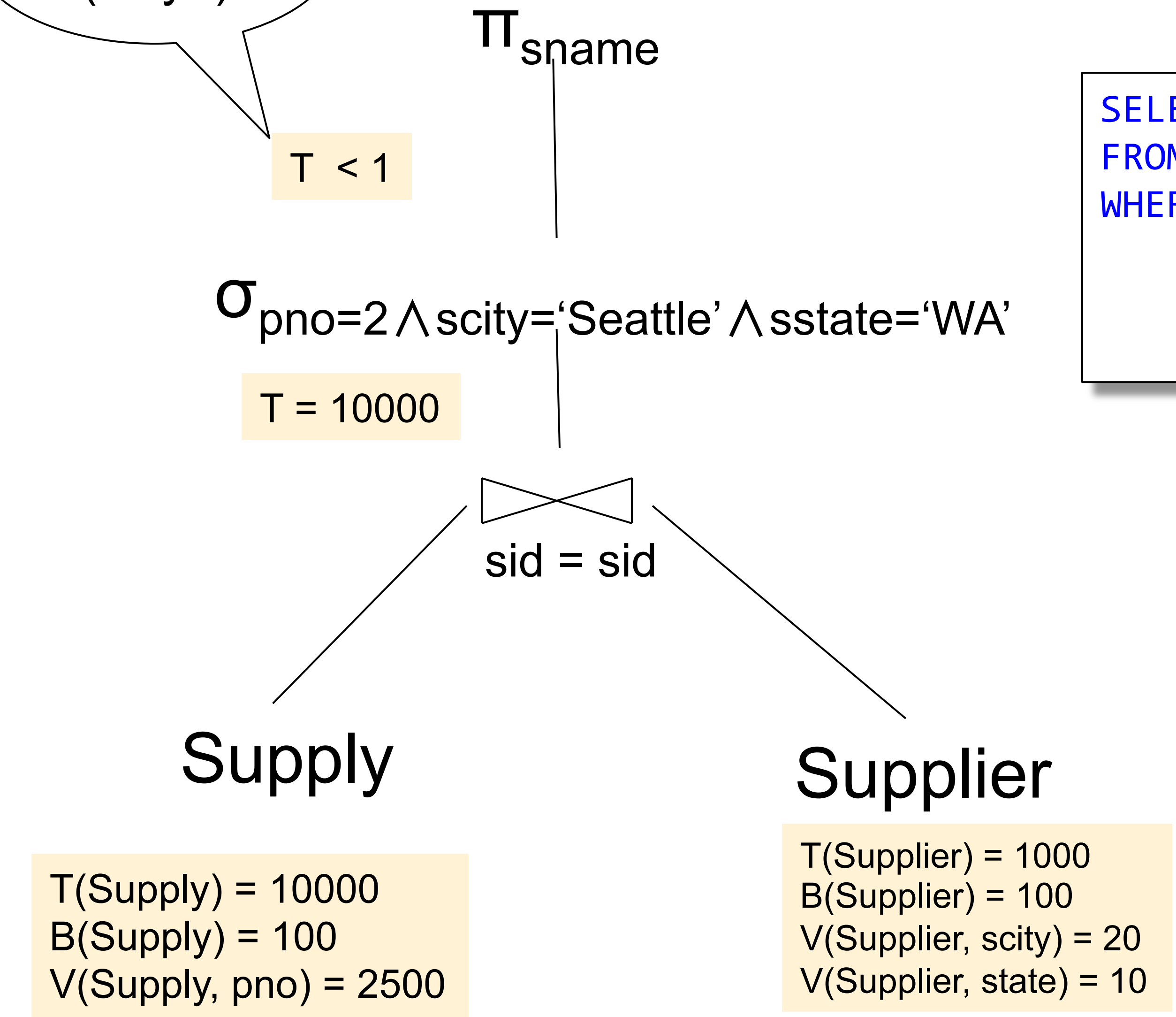
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

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

Logical Query Plan 1

Estimated (why?)



π_{sname}

T < 1

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

T = 10000

$\bowtie_{sid = sid}$

Supply

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

Supplier

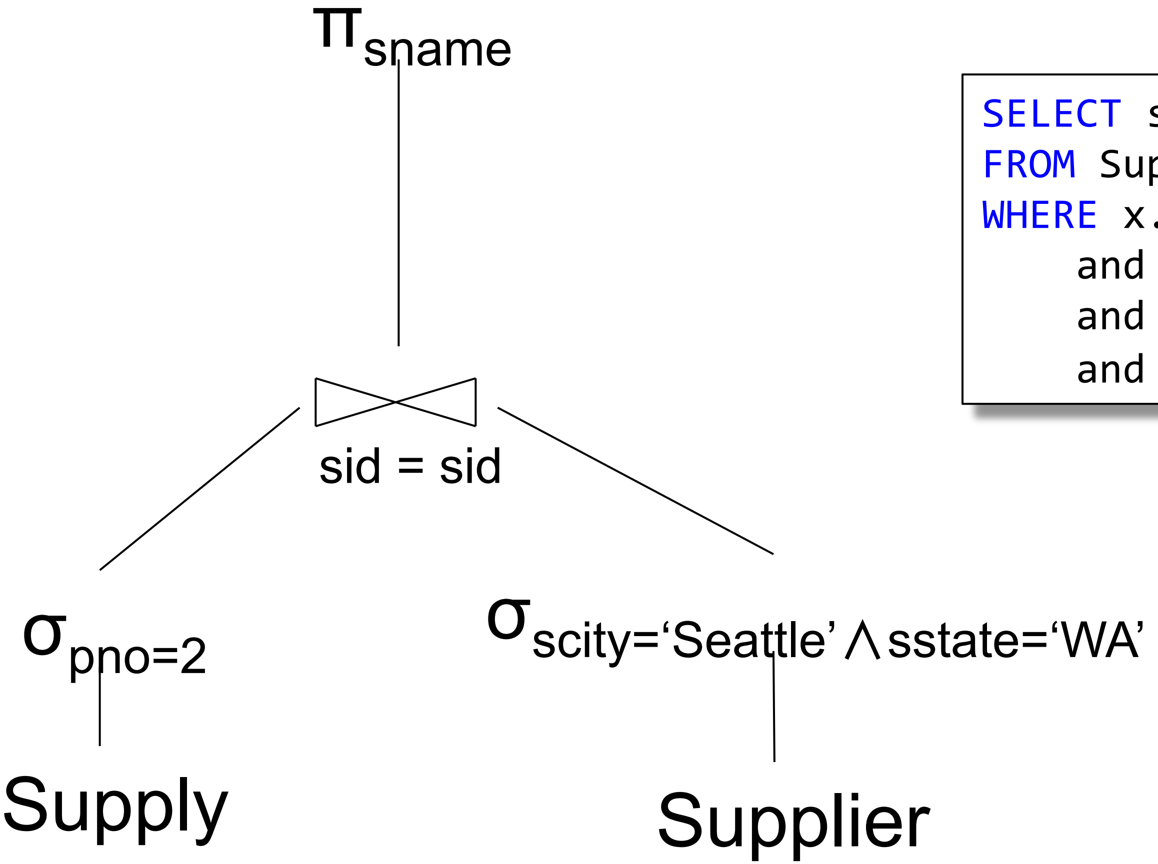
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

```
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(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, 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’
```

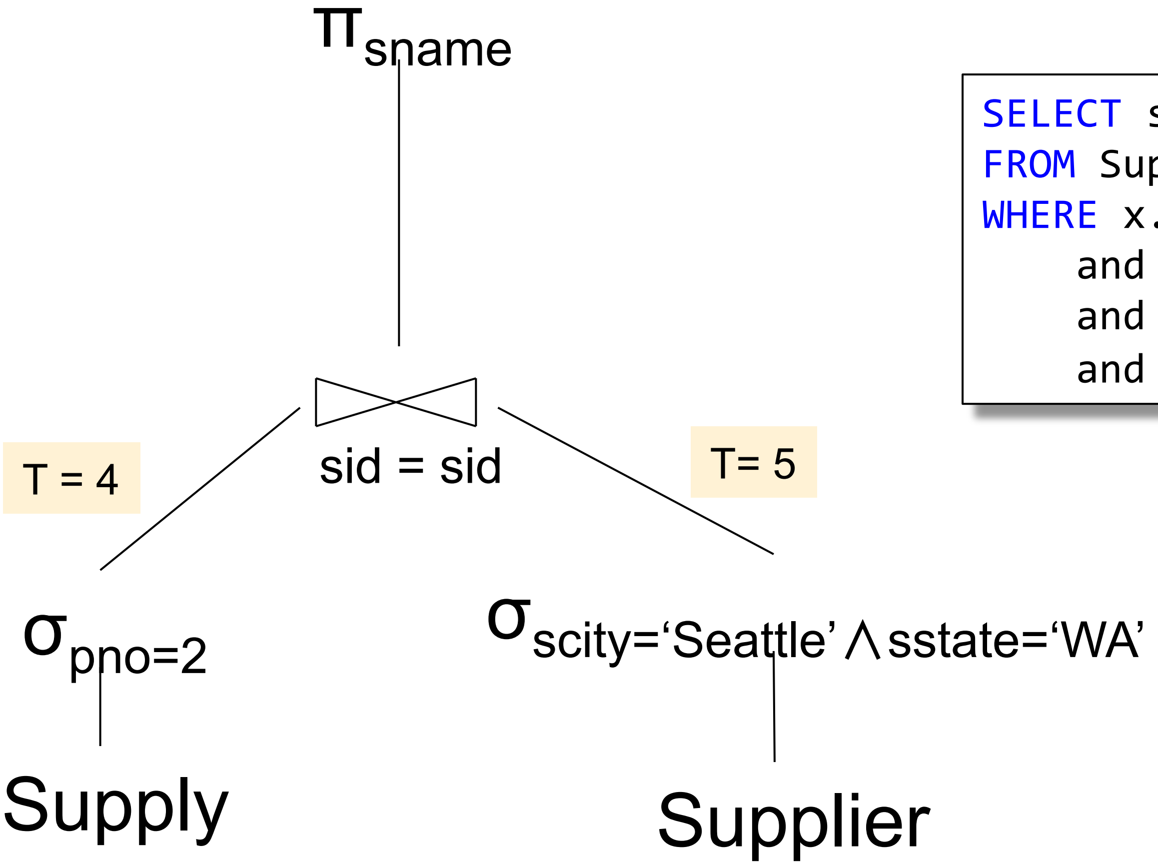
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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’
```

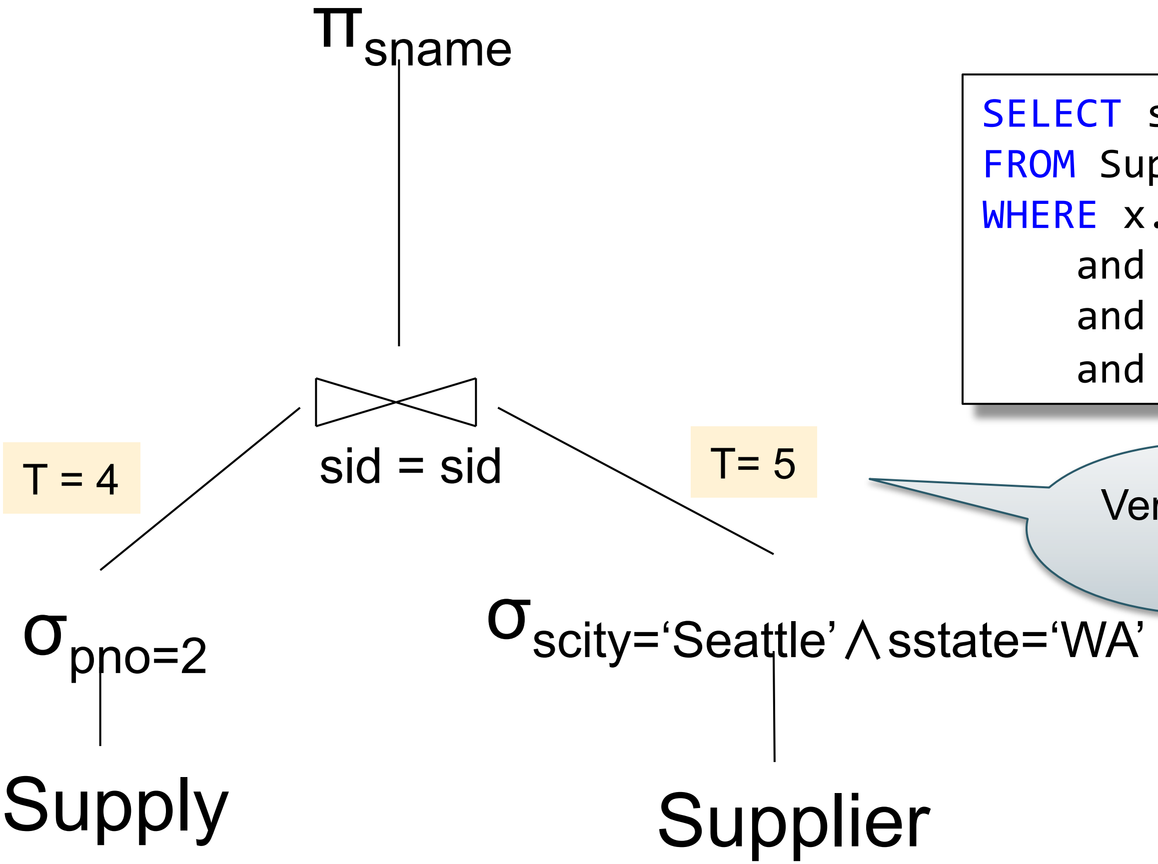
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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



π_{sname}

$\bowtie_{sid = sid}$

T = 4

T= 5

$\sigma_{pno=2}$

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

Supply

Supplier

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

Very wrong!
Why?

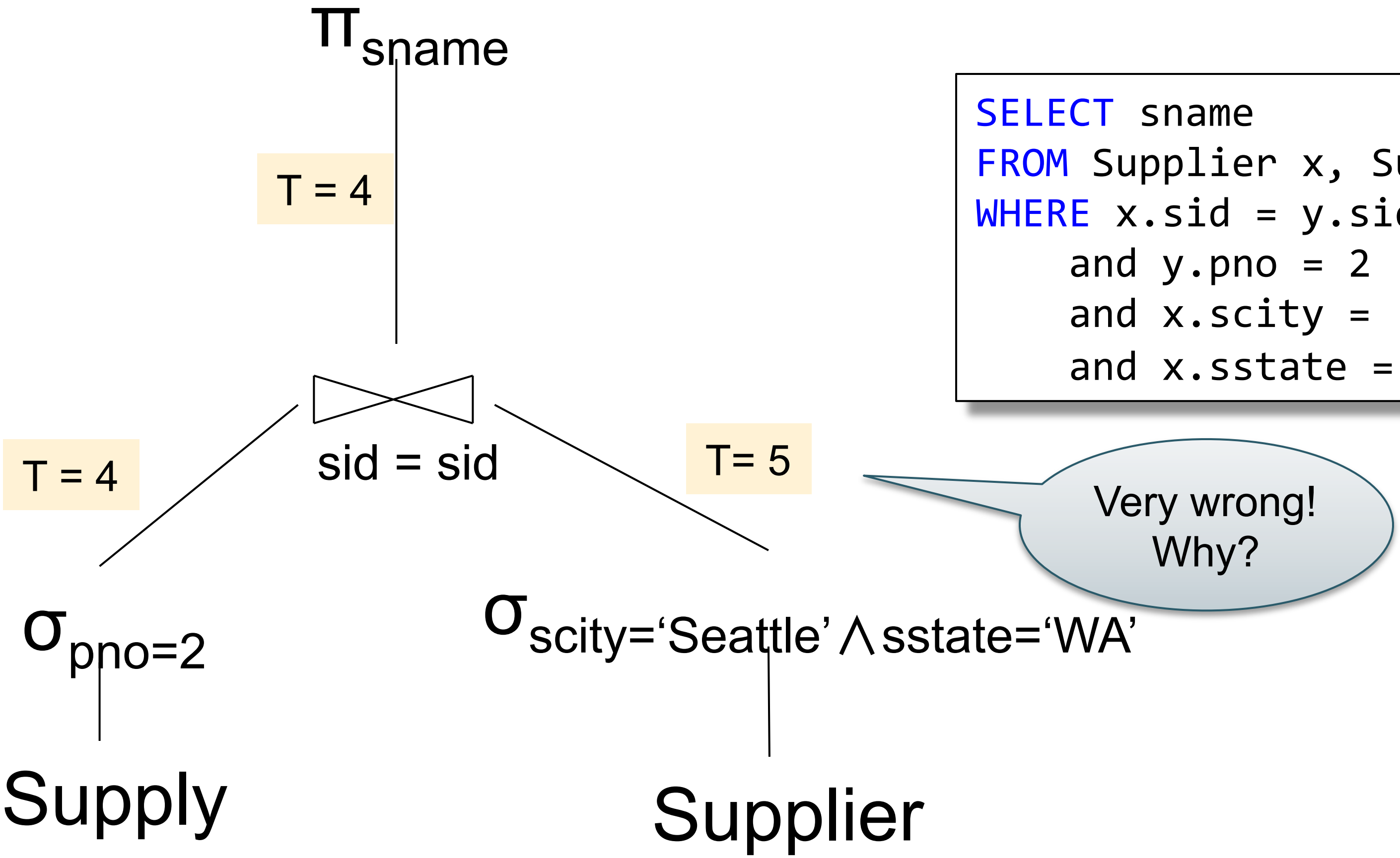
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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



π_{sname}

T = 4

$\bowtie$ sid = sid

T = 4

T= 5

Very wrong! Why?

$\sigma_{pno=2}$

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

Supply

Supplier

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

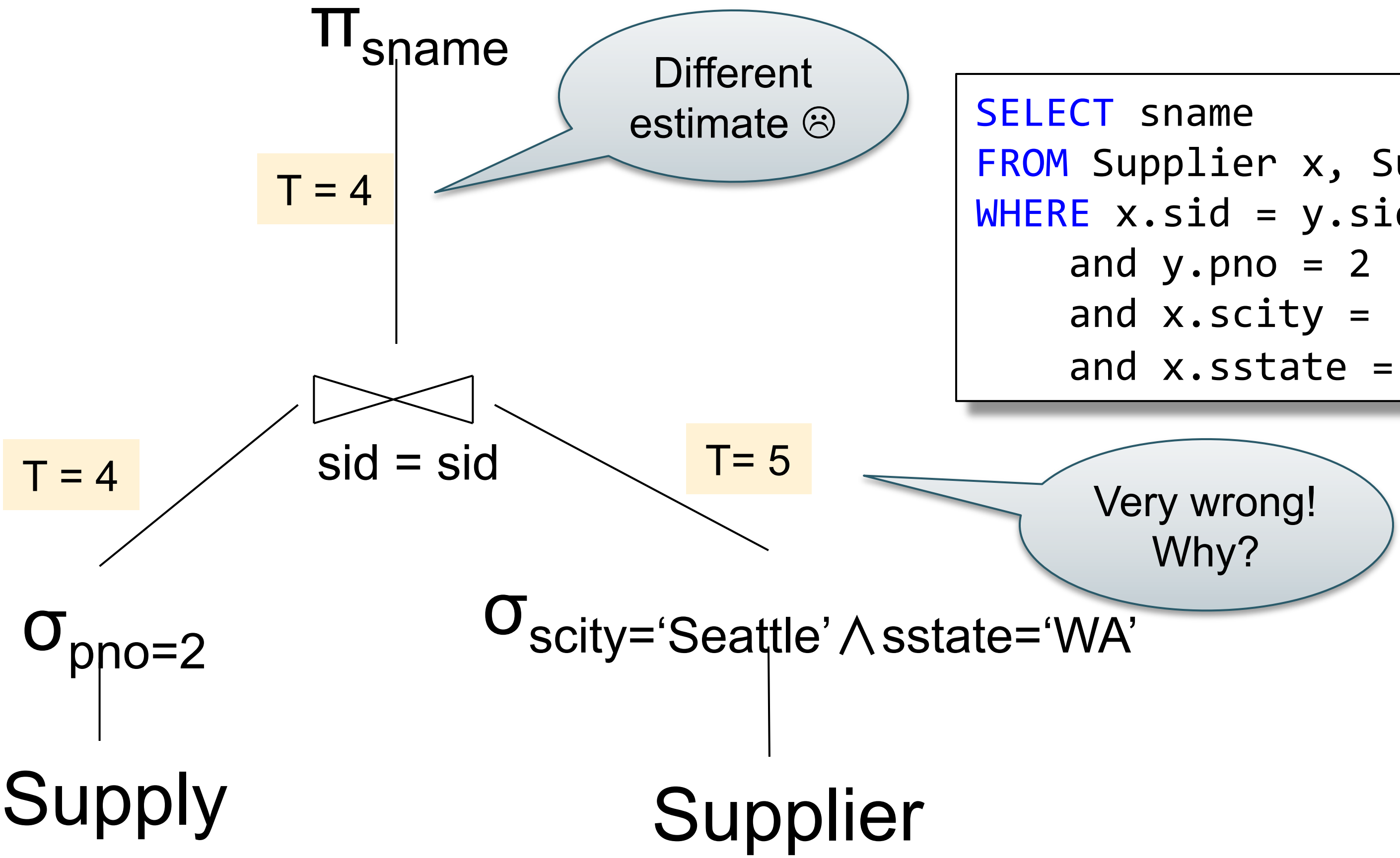
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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’
```

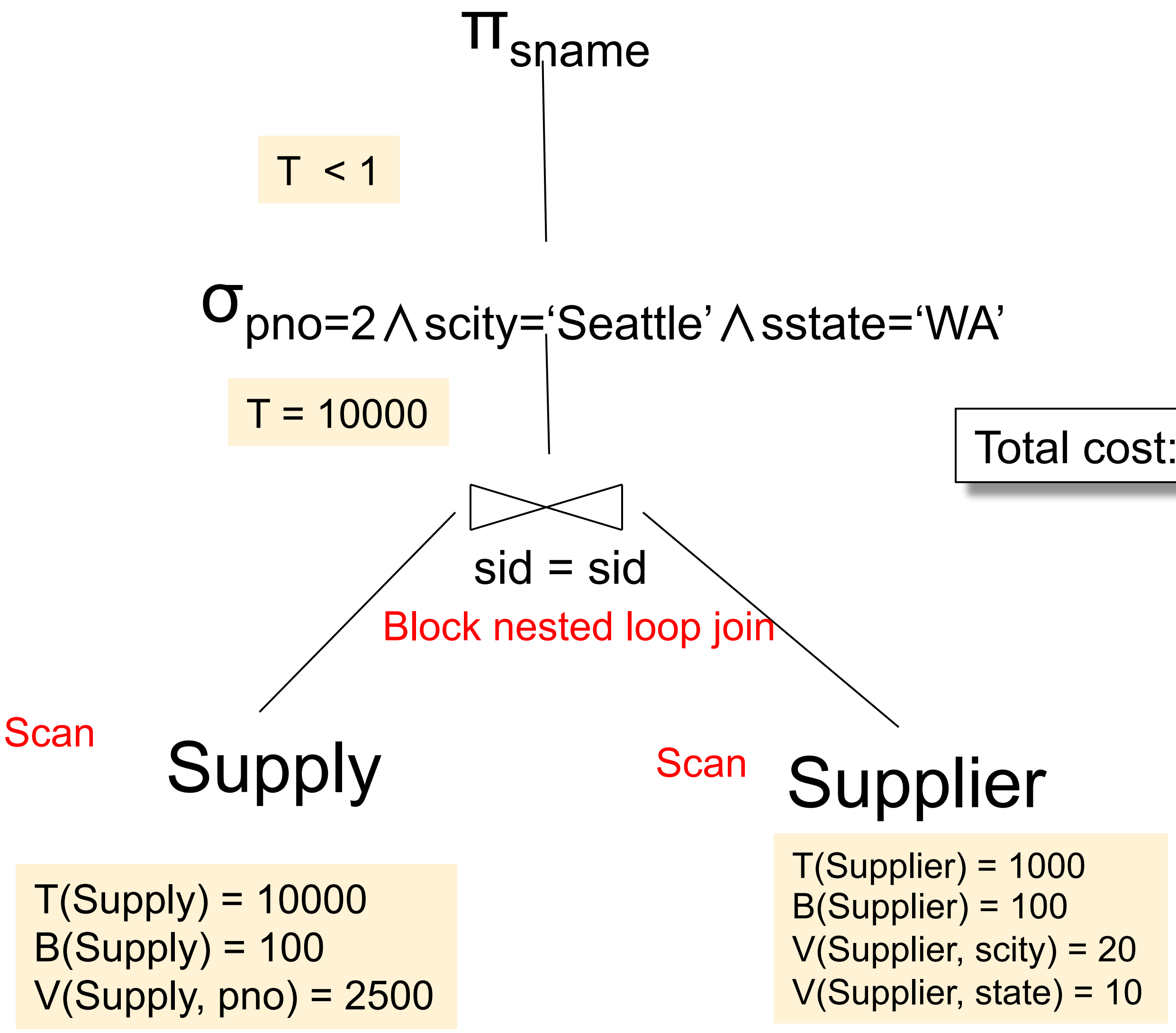
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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



π_{sname}

T < 1

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

T = 10000

$\bowtie$ sid = sid

Block nested loop join

Scan Supply

Scan Supplier

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

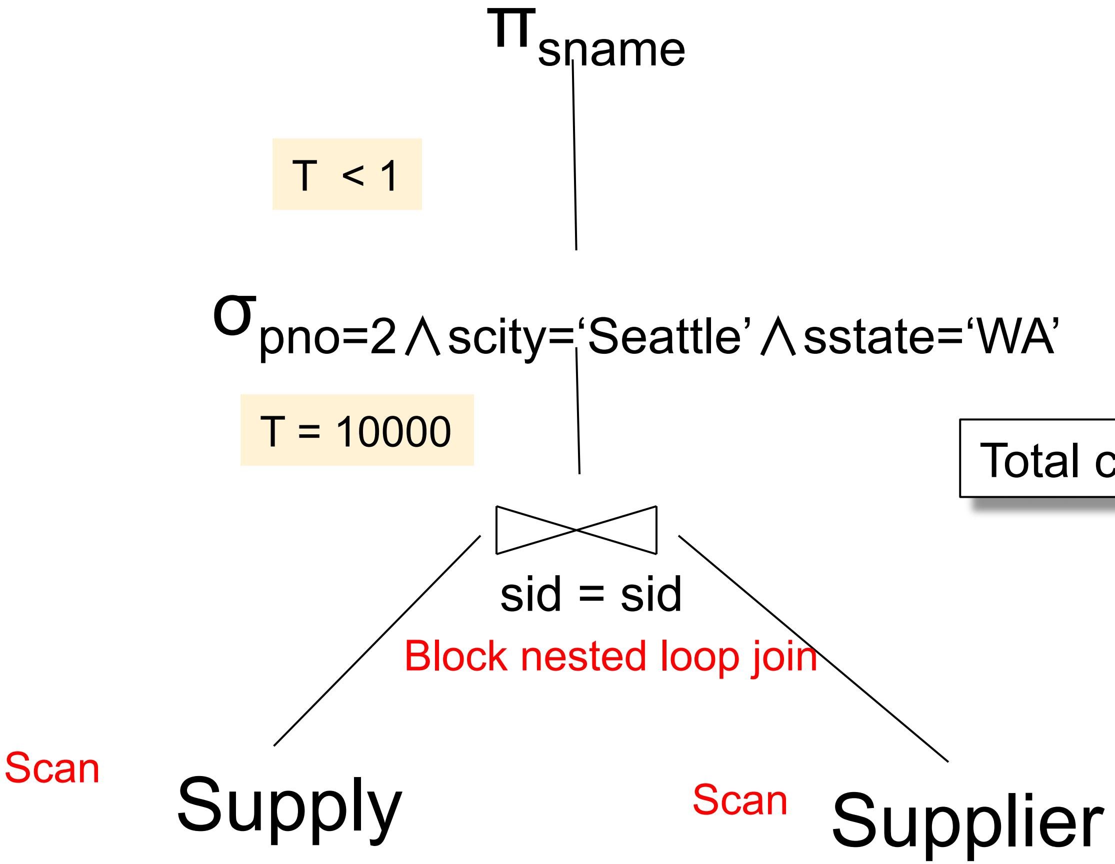
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

Total cost:

M=11

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

Physical Plan 1



π_{sname}

T < 1

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

T = 10000

$\bowtie$ sid = sid

Block nested loop join

Scan Supply

Scan Supplier

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

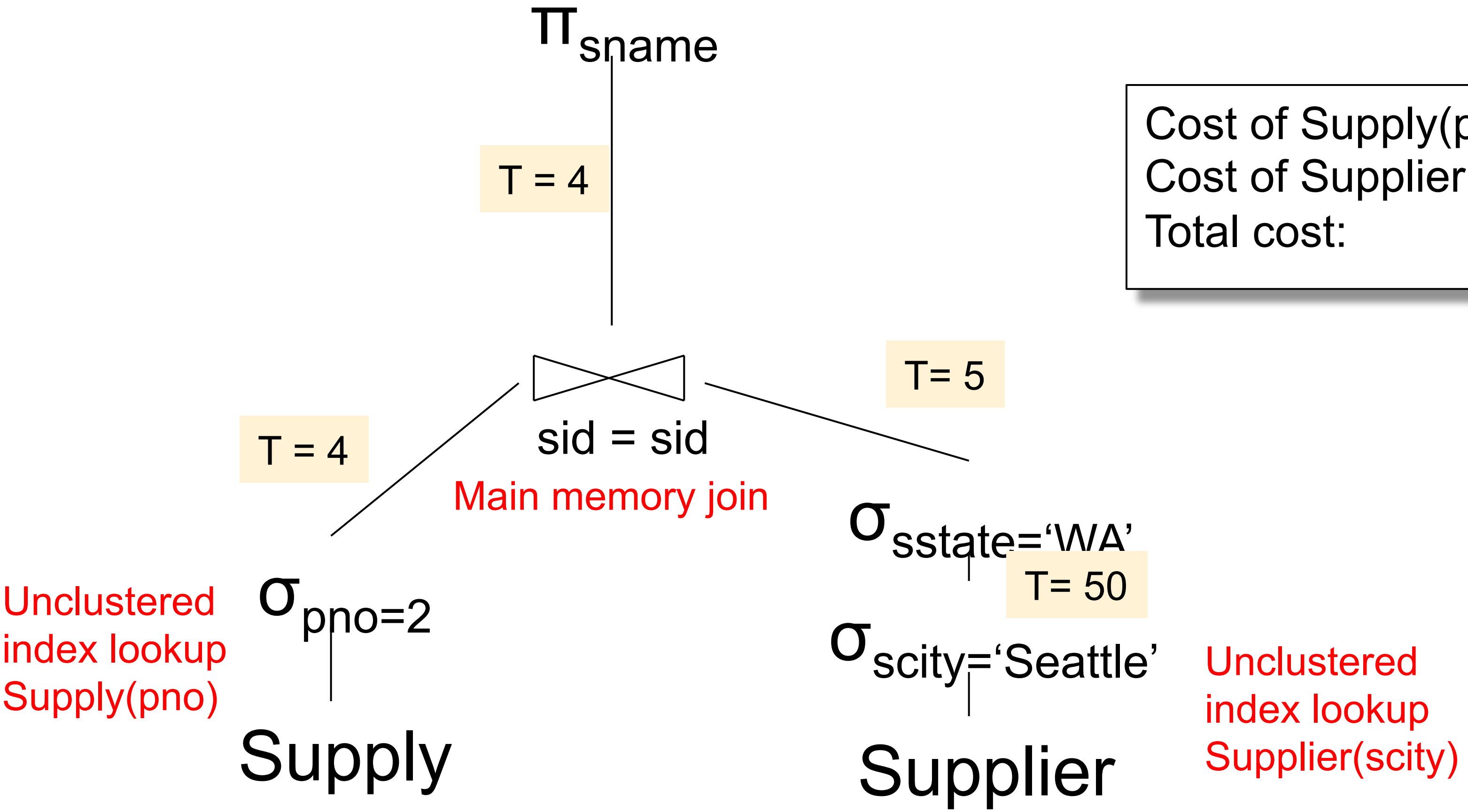
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

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

M=11

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

Physical Plan 2



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

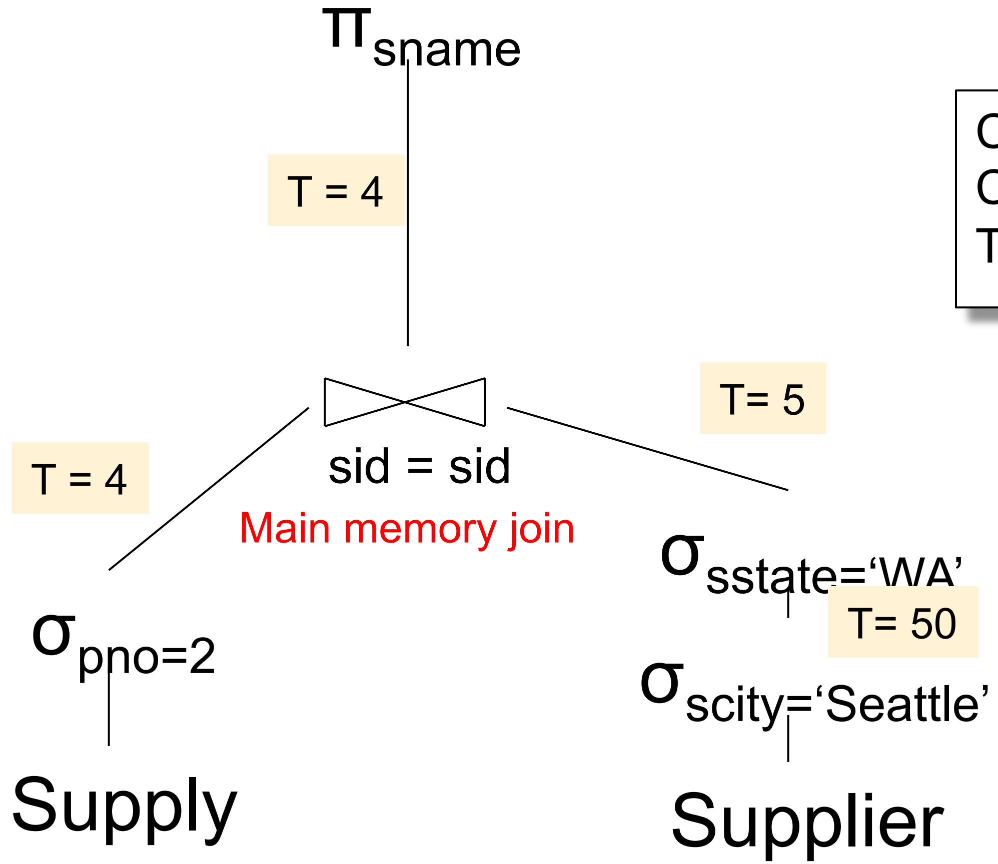
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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

M=11

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

Physical Plan 2



π_{sname}

T = 4

$\bowtie$ sid = sid

Main memory join

T = 4

T= 5

$\sigma_{pno=2}$

Unclustered index lookup Supply(pno)

Supply

$\sigma_{sstate=\text{'WA'}}$

T= 50

$\sigma_{scity=\text{'Seattle'}}$

Unclustered index lookup Supplier(scity)

Supplier

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

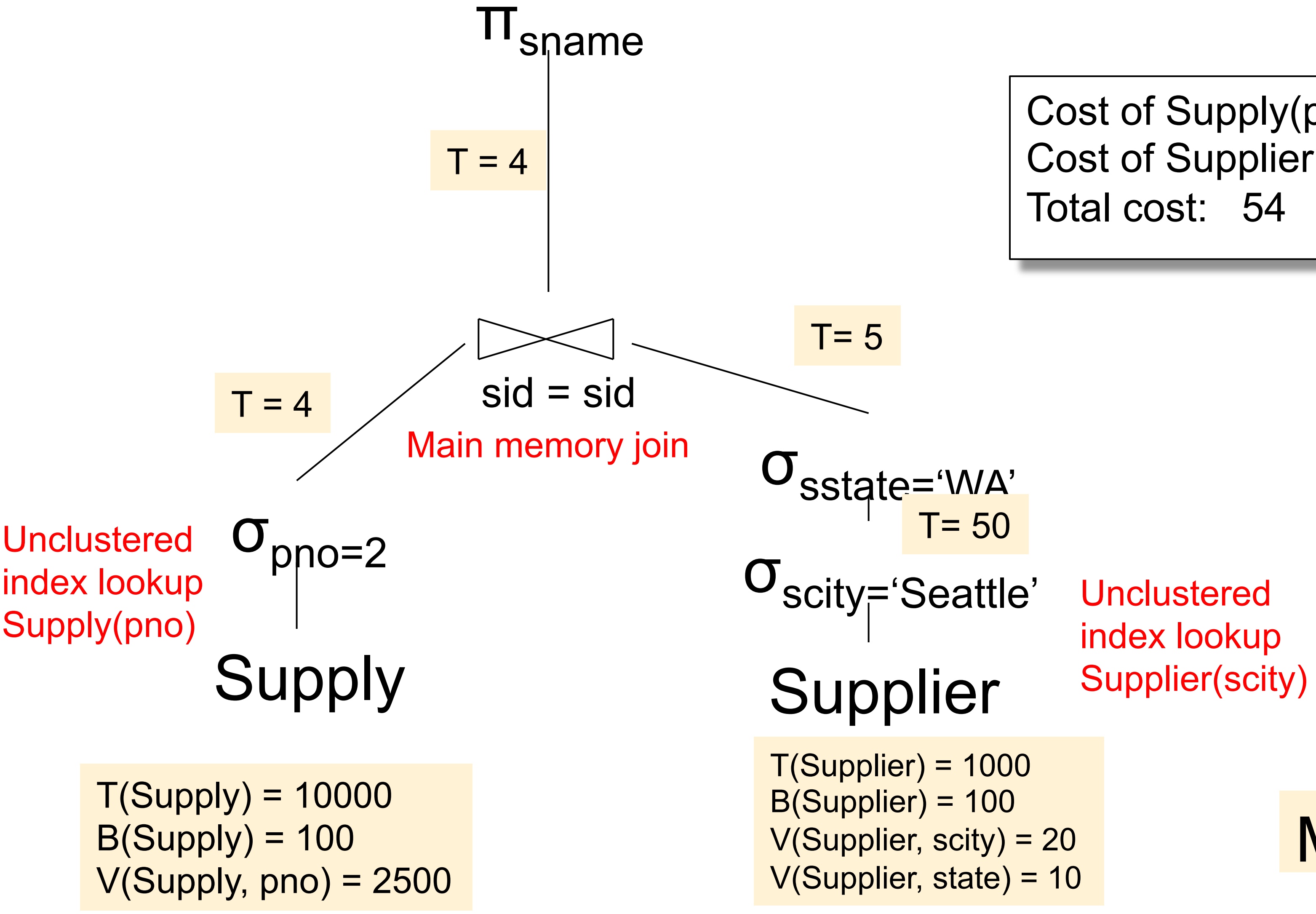
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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



π_{sname}

T = 4

$\bowtie$ sid = sid

Main memory join

T = 4

T= 5

$\sigma_{pno=2}$

Unclustered index lookup Supply(pno)

Supply

$\sigma_{sstate='WA'}$

T= 50

$\sigma_{scity='Seattle'}$

Unclustered index lookup Supplier(scity)

Supplier

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

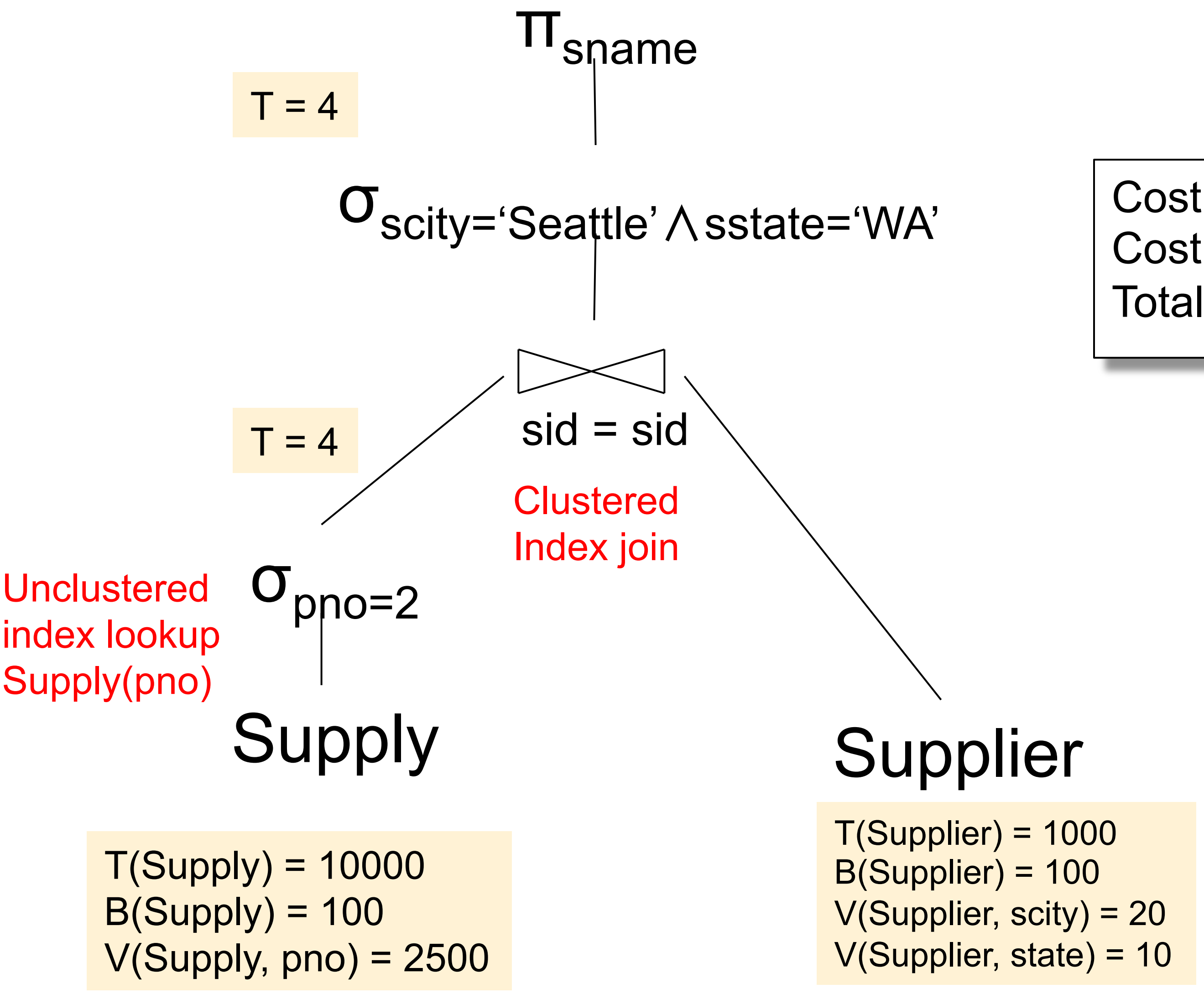
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

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

M=11

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

Physical Plan 3



π_{sname}

T = 4

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

$\bowtie$ sid = sid

Clustered
Index join

T = 4

$\sigma_{pno=2}$

Unclustered
index lookup
Supply(pno)

Supply

Supplier

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

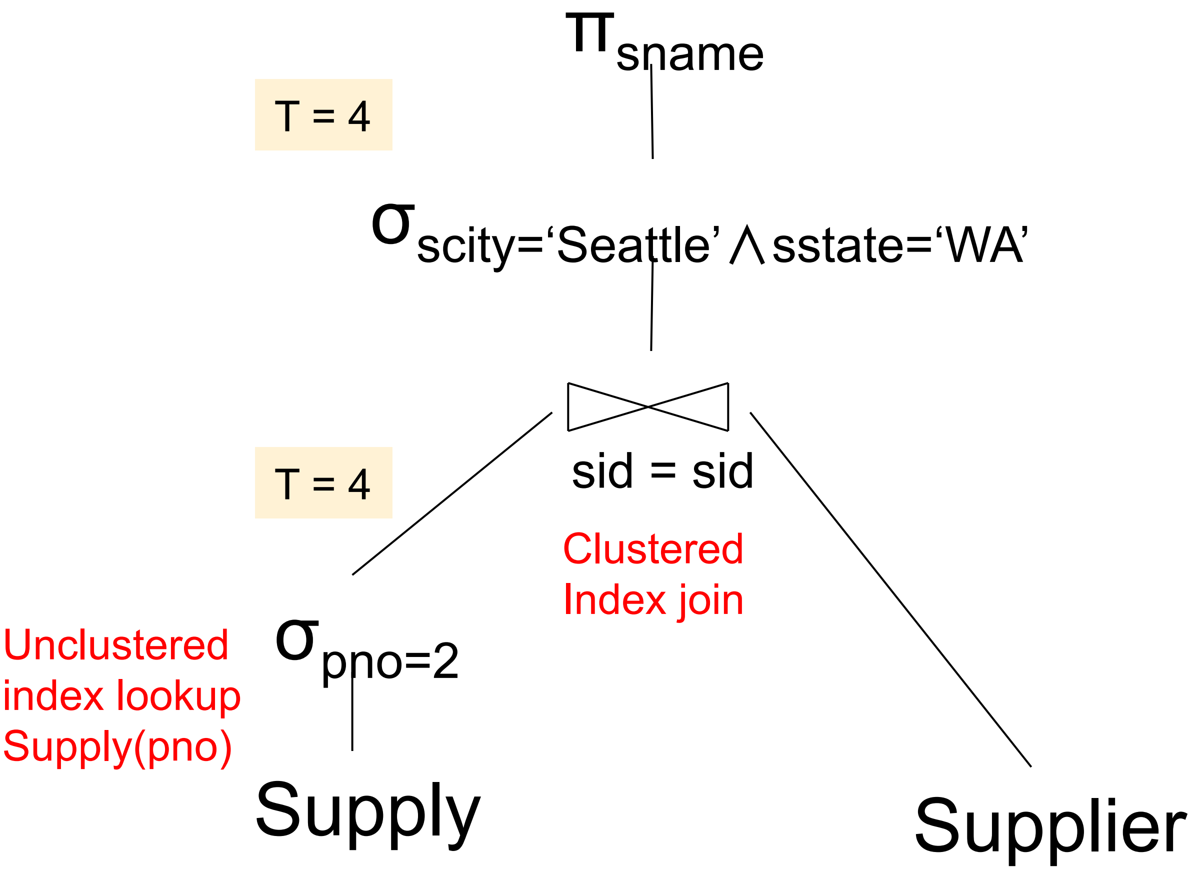
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

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

M=11

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

Physical Plan 3



π_{sname}

T = 4

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

$\bowtie$ sid = sid

Clustered Index join

T = 4

$\sigma_{pno=2}$

Unclustered index lookup Supply(pno)

Supply

Supplier

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

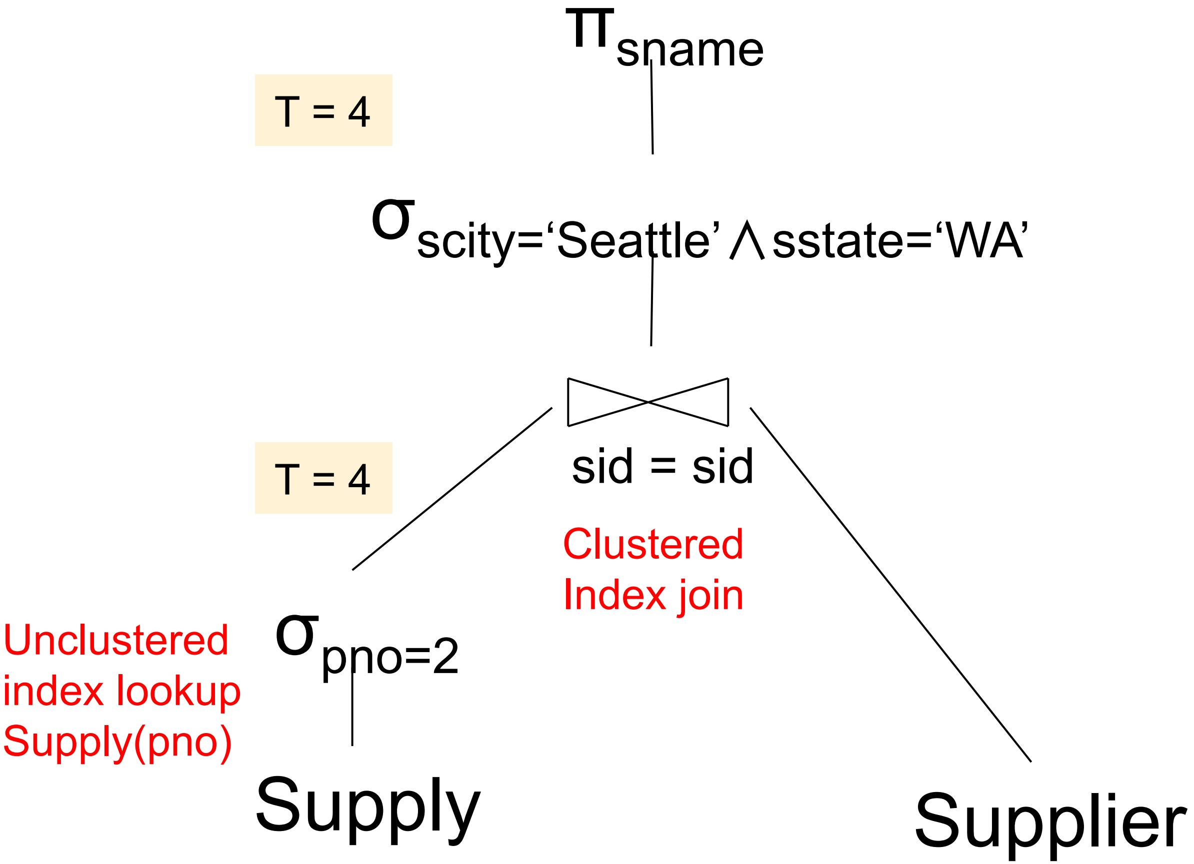
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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



π_{sname}

T = 4

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

$\bowtie$

sid = sid

Clustered
Index join

T = 4

Unclustered
index lookup
Supply(pno)

$\sigma_{pno=2}$

Supply

Supplier

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

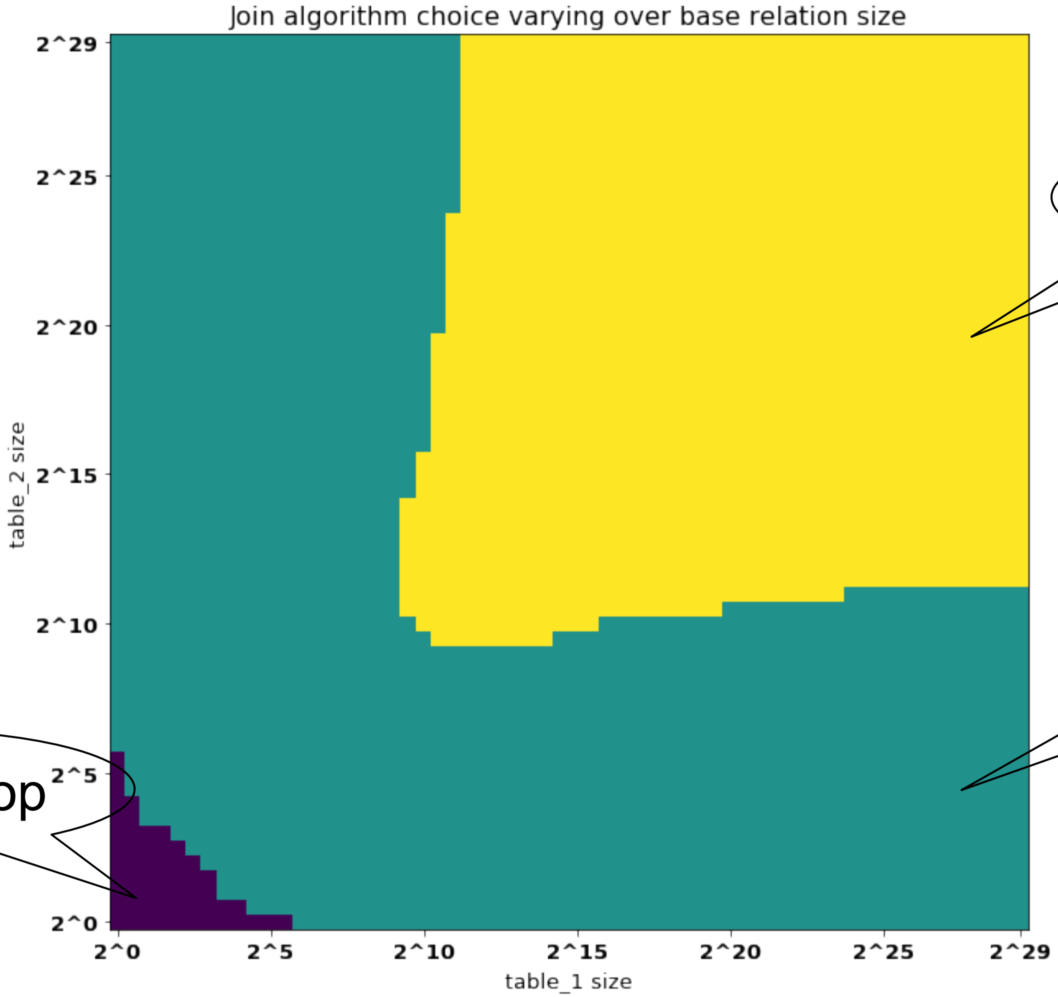
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

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

M=11

R⋈S in Postgres

Courtesy of Walter Cai



Merge join

Hash join

Nested loop

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(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) = ?

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

Discussion

- Small number of buckets
  - Hundreds, or thousands, but not more
  - WHY ?
- *Not* updated during database update, but recomputed periodically
  - WHY ?
- Multidimensional histograms rarely used
  - WHY ?

Query Optimization

Three major components:

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 paper)

Access Path Selection

Supplier(sid,sname,scity,sstate)
Selection condition: sid > 300 ∧ 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 ∧ 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)

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

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.

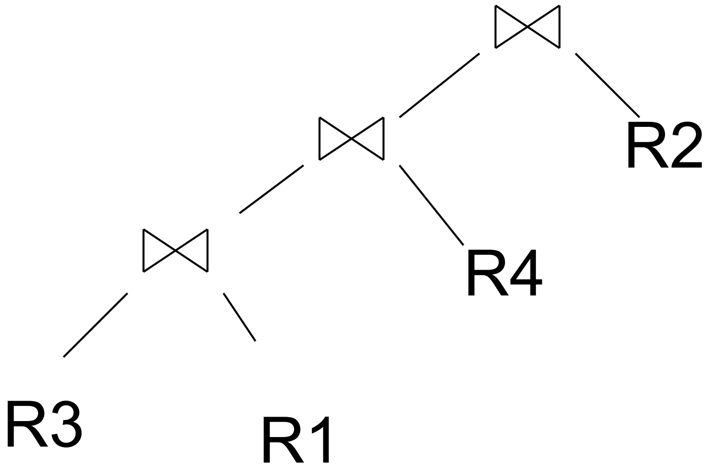
Search Space Challenges

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

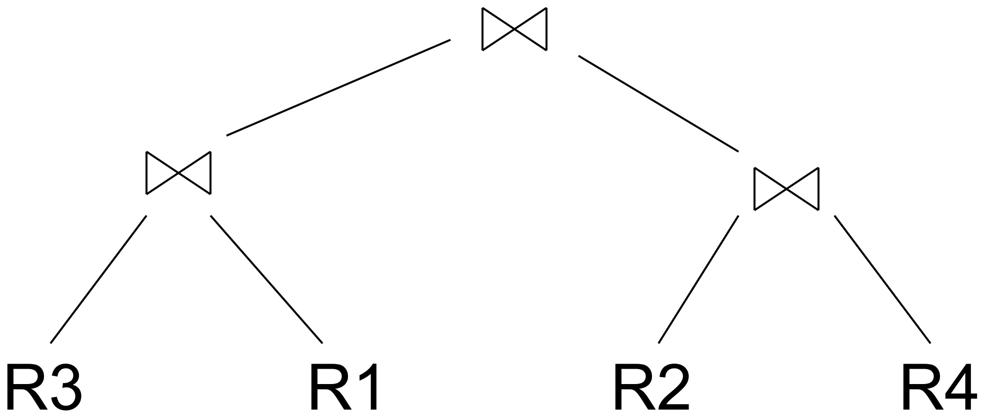
- Cannot consider ALL plans
- Want a search space that includes low-cost plans

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

Left-Deep Plans and Bushy Plans



Left-deep plan



Bushy plan

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

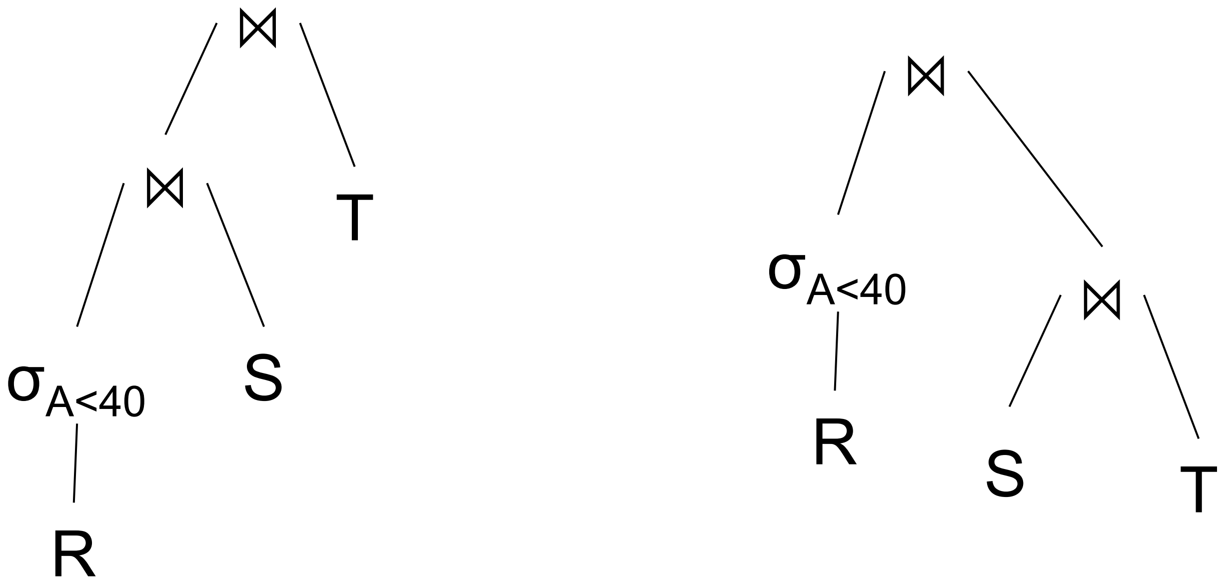
Three Approaches to Search Space Enumeration

- 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
```



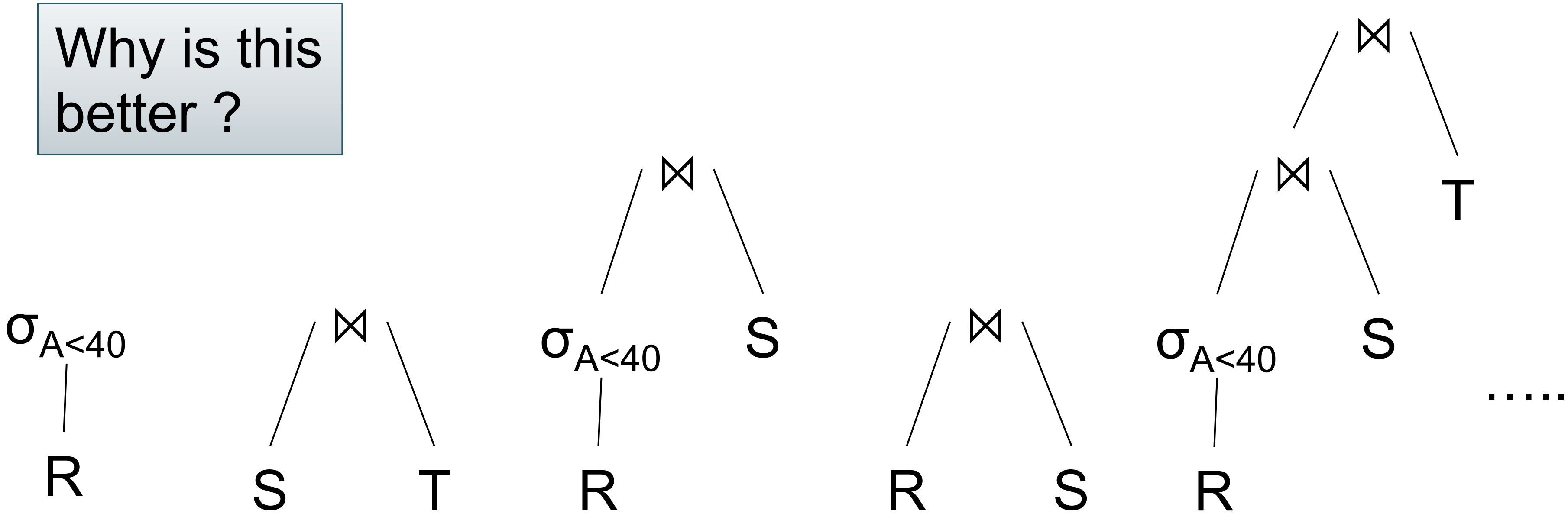
Why is this search space inefficient ?

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
```

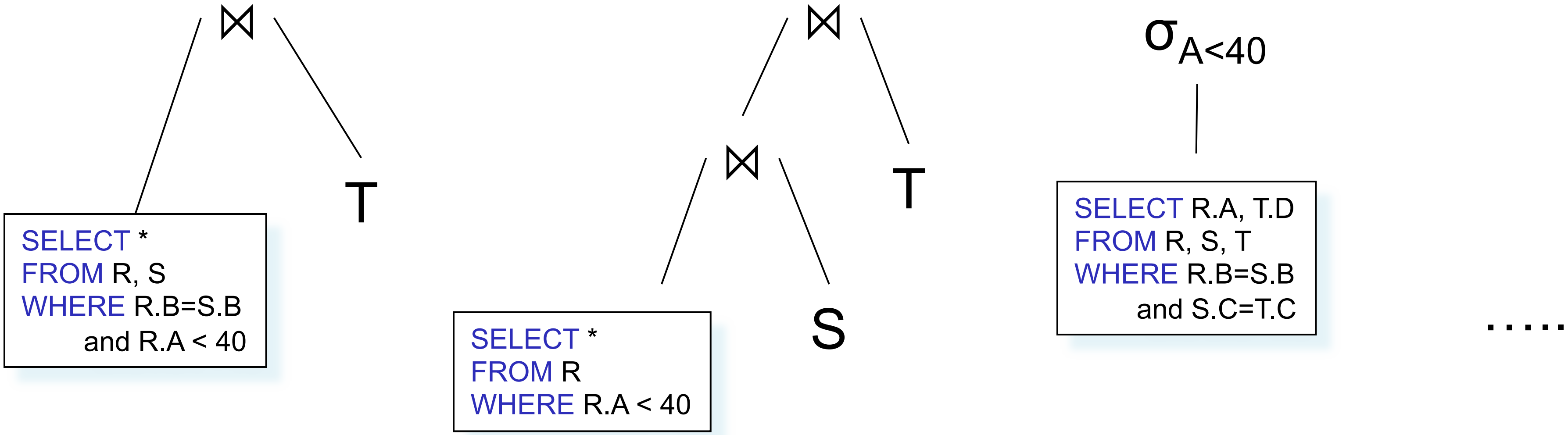
Why is this better ?



Top-down Partial Plans

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

```sql
SELECT *
FROM R, S, T
WHERE R.B=S.B and S.C=T.C and R.A<40
```



$\bowtie$

```sql
SELECT *
FROM R, S
WHERE R.B=S.B
      and R.A < 40
```

T

$\bowtie$

$\bowtie$

```sql
SELECT *
FROM R
WHERE R.A < 40
```

S

T

$\sigma_{A<40}$

```sql
SELECT R.A, T.D
FROM R, S, T
WHERE R.B=S.B
      and S.C=T.C
```

…..

Why is this best for rewrite rules?

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 $Q \subseteq \{R1, \ldots, Rn\}$ 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)