CSE544
Query Optimization

Tuesday-Thursday,

February 8th-10th, 2011

Outline

- Chapter 15 in the textbook

Query Optimization Algorithm

- Enumerate alternative plans
- Compute estimated cost of each plan
  - Compute number of I/Os
  - Compute CPU cost
- Choose plan with lowest cost
  - This is called cost-based optimization

Example

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

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

- Some statistics
  - T(Supplier) = 1000 records
  - T(Supply) = 10,000 records
  - B(Supplier) = 100 pages
  - B(Supply) = 100 pages
  - V(Supplier,scity) = 20, V(Supplier,state) = 10
  - V(Supply,pno) = 2,500
  - Both relations are clustered
- M = 10

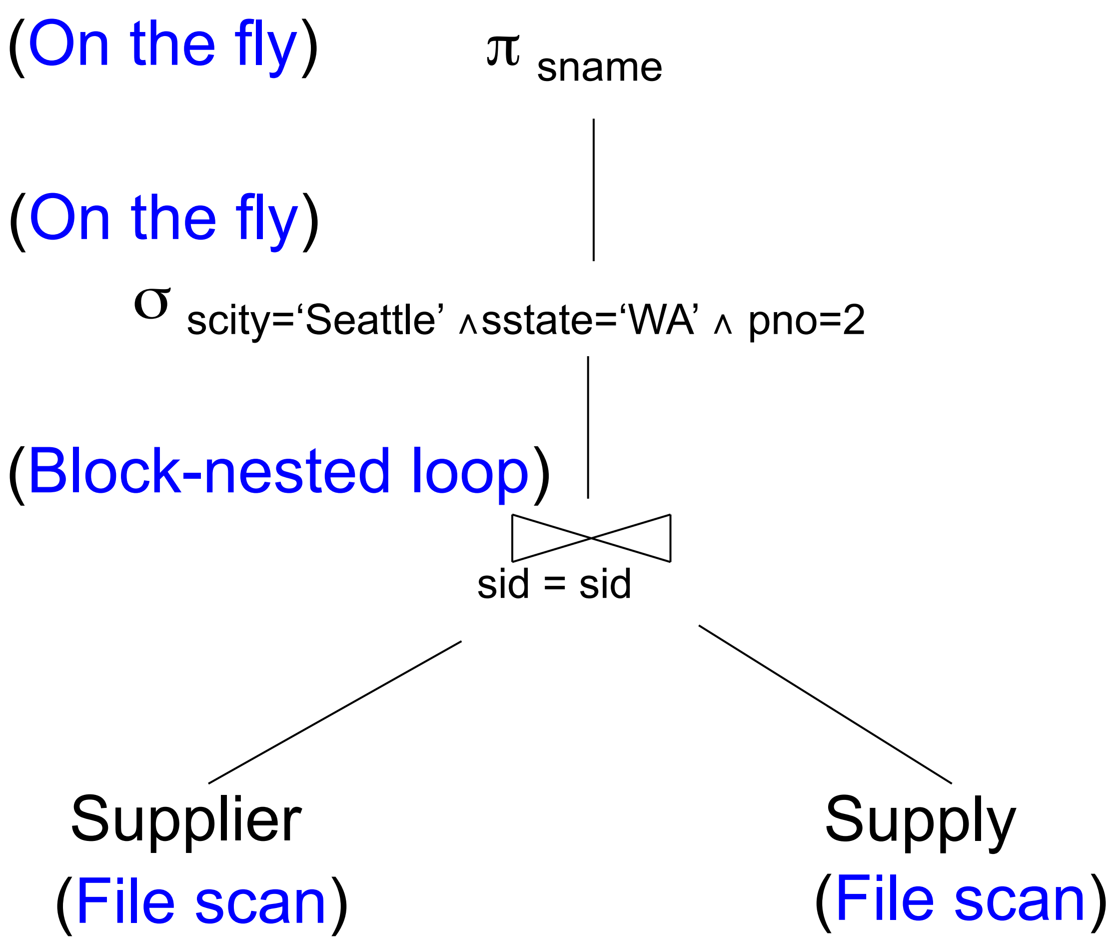
T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 10

Physical Query Plan 1



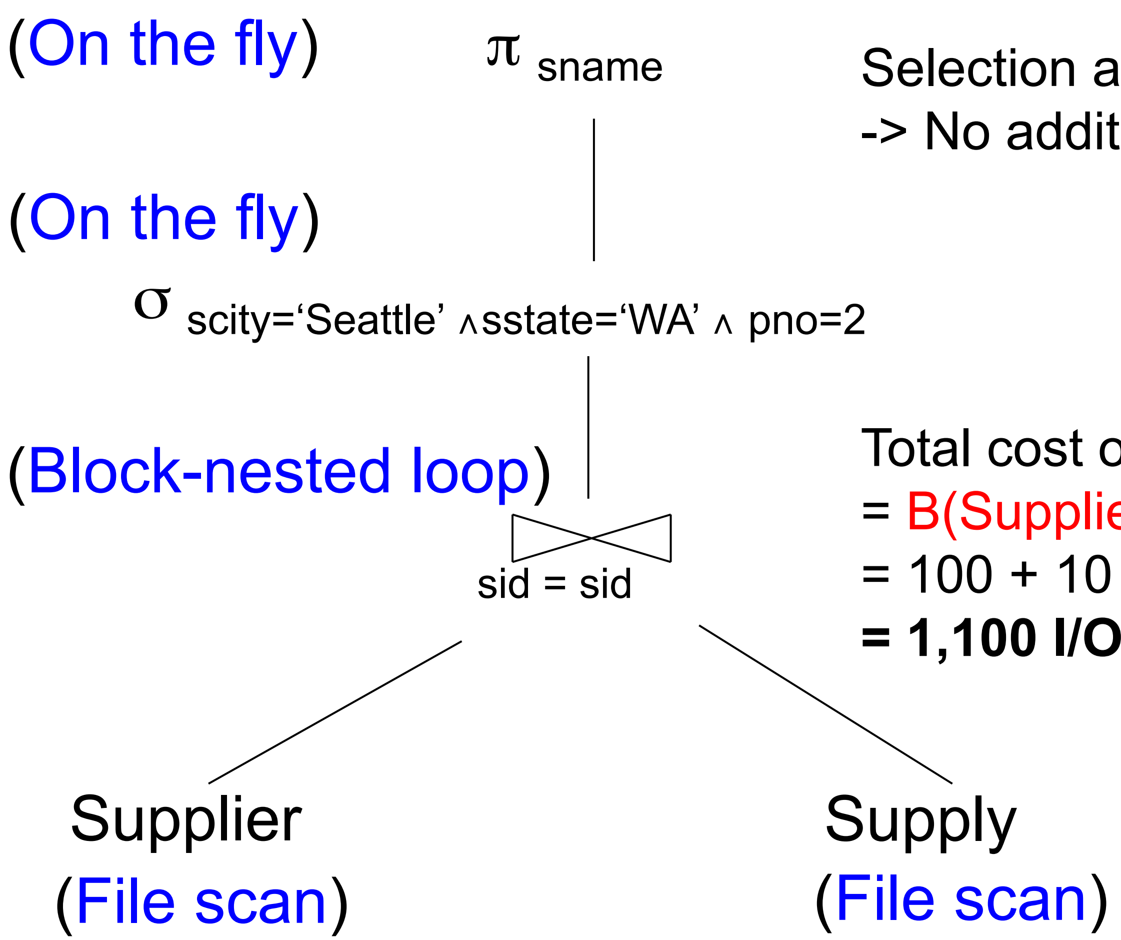
T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 10

Physical Query Plan 1



Selection and project on-the-fly
-> No additional cost.

Total cost of plan is thus cost of join:
= B(Supplier)+B(Supplier)*B(Supply)/M
= 100 + 10 * 100
= 1,100 I/Os

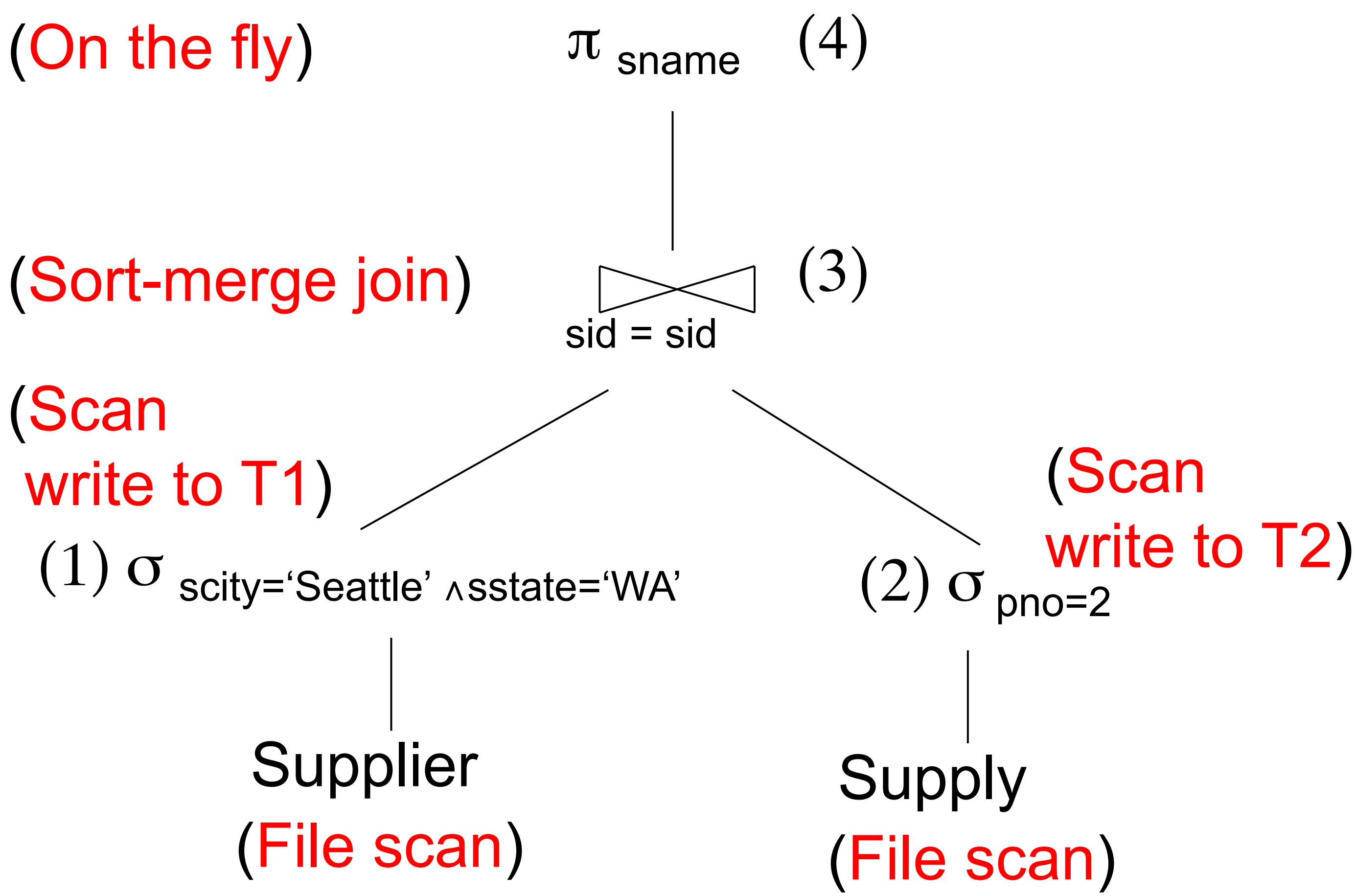
T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 10

Physical Query Plan 2



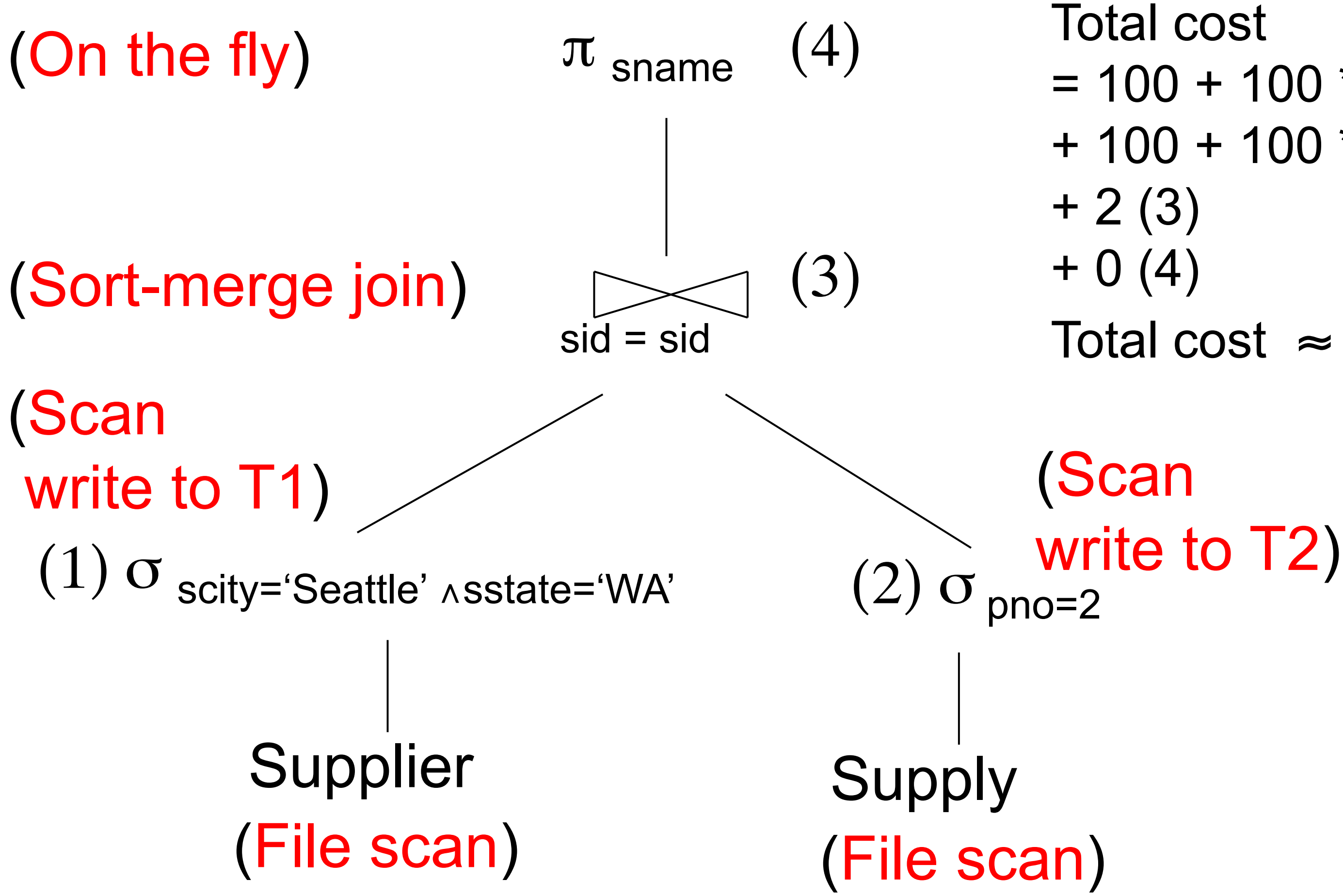
T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 10

Physical Query Plan 2



Total cost
= 100 + 100 * 1/20 * 1/10 (1)
+ 100 + 100 * 1/2500 (2)
+ 2 (3)
+ 0 (4)
Total cost ≈ **204 I/Os**

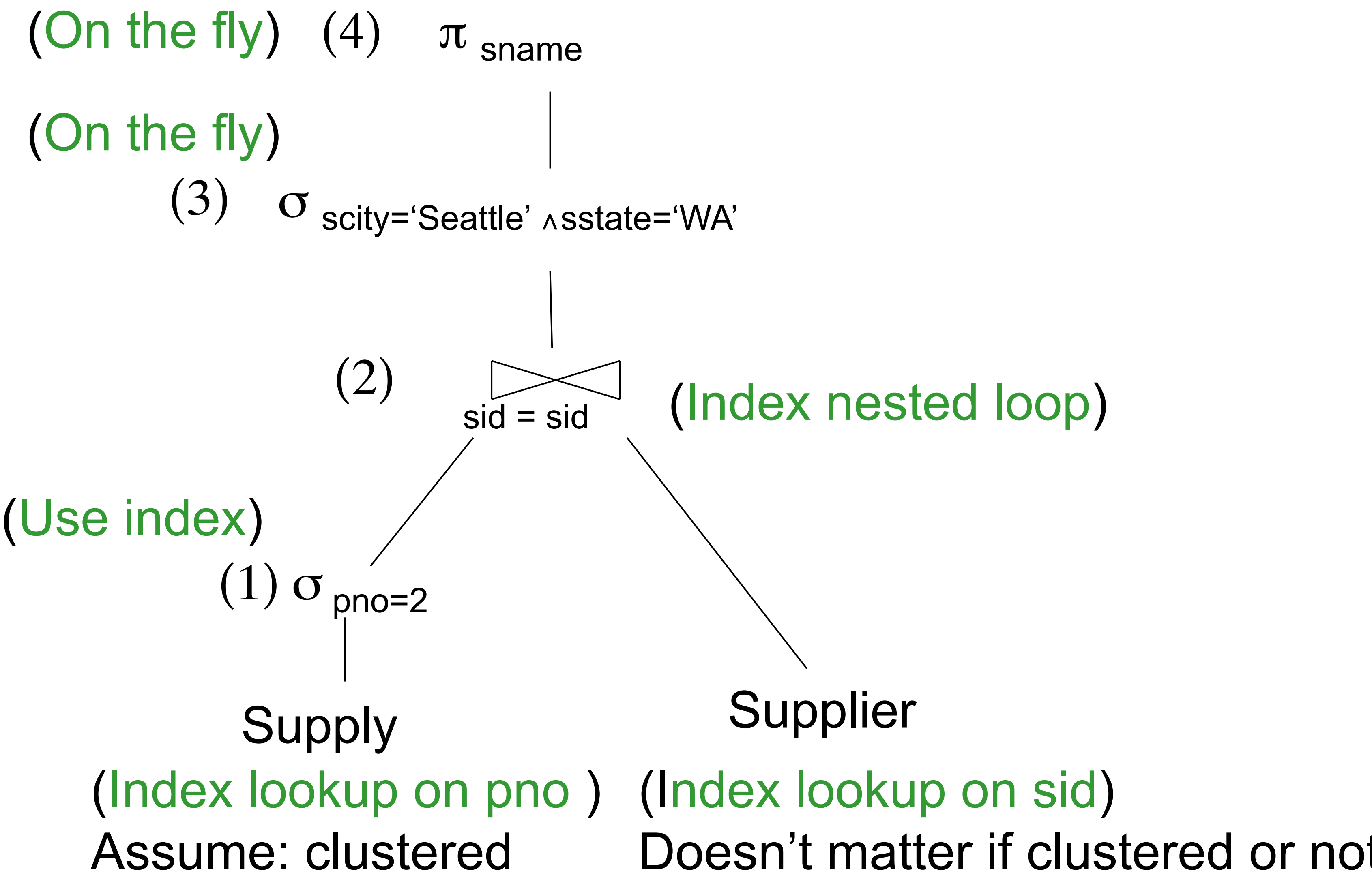
T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 10

Physical Query Plan 3



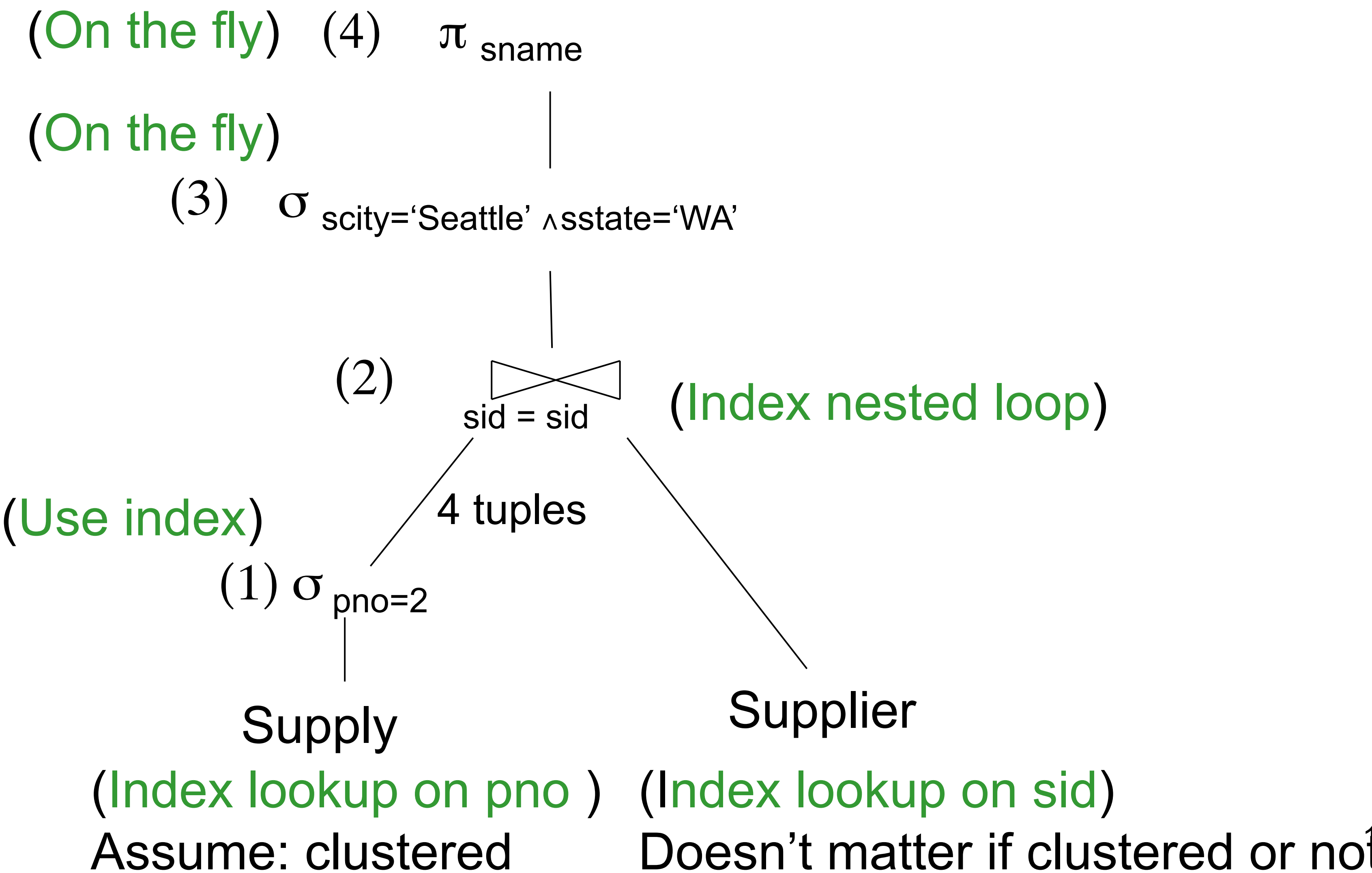
T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 10

Physical Query Plan 3



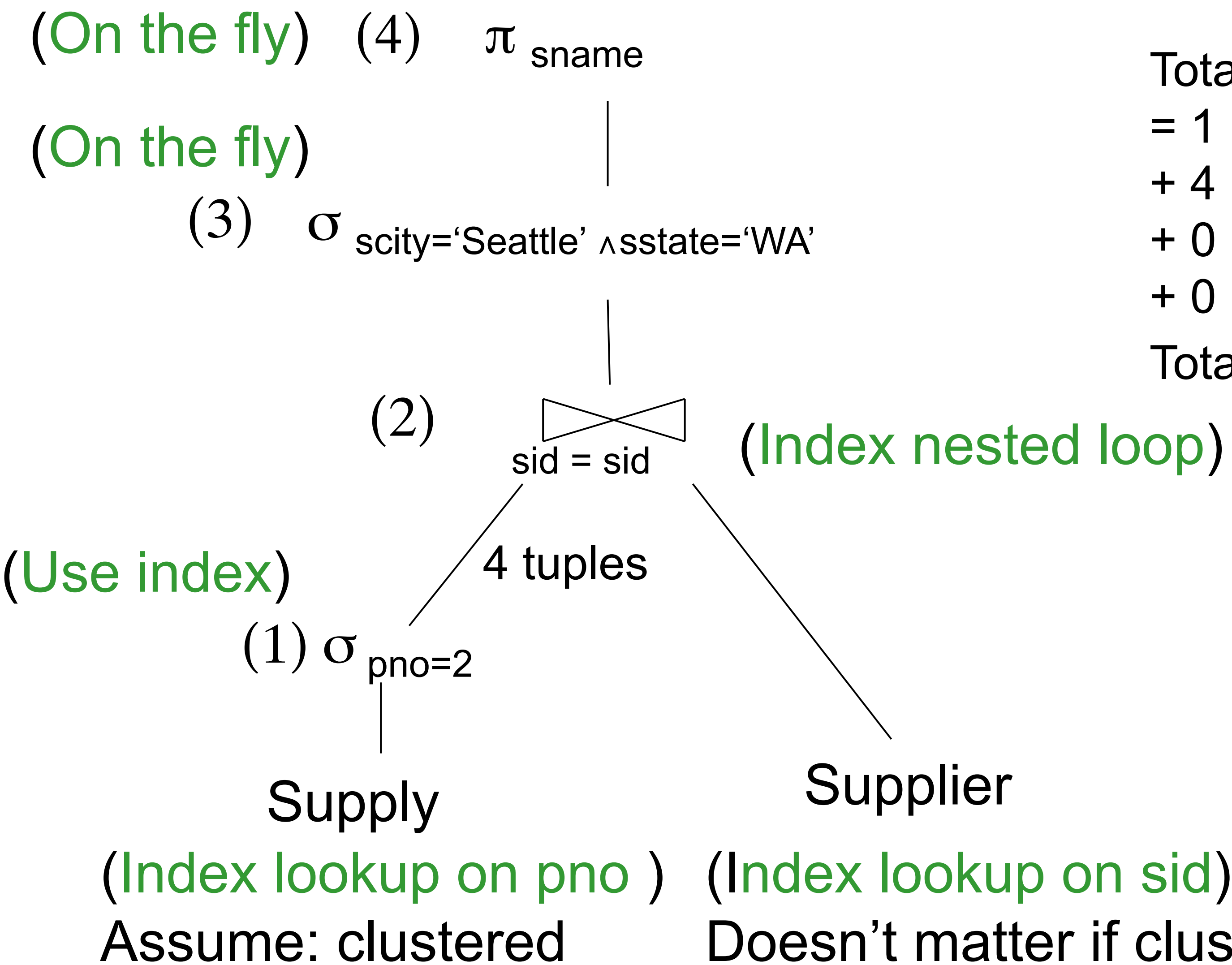
T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 10

Physical Query Plan 3



Total cost
= 1 (1)
+ 4 (2)
+ 0 (3)
+ 0 (3)
Total cost ≈ **5 I/Os**

Simplifications

- In the previous examples, we assumed that all index pages were in memory
- When this is not the case, we need to add the cost of fetching index pages from disk

Lessons

1. Need to consider several physical plan
   – even for one, simple logical plan

2. No plan is best in general
   – need to have ***statistics*** over the data
   – the B's, the T's, the V's

The Contract of the Optimizer

- High-quality execution plans for all queries,
- While taking relatively small optimization time, and
- With limited additional input such as histograms.

Query Optimization

Three major components:

1. Search space

2. Algorithm for enumerating query plans

3. Cardinality and cost estimation

History of Query Optimization

- First query optimizer was for System R, from IBM, in 1979
- It had all three components in place, and defined the architecture of query optimizers for years to come
- You will see often references to System R
- Read Section 15.6 in the book

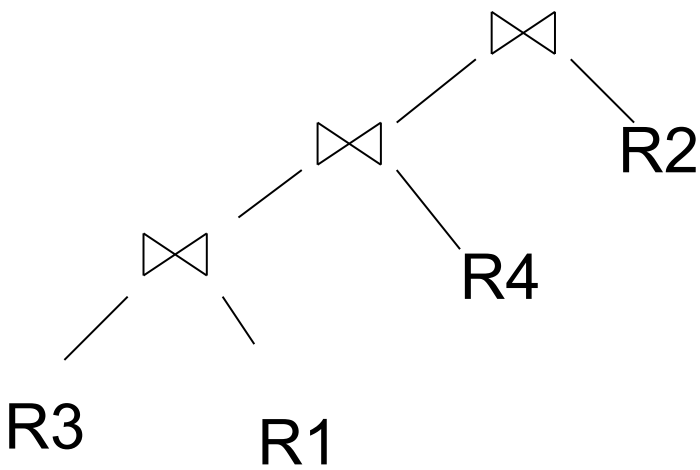
1. Search Space

- This is the set of all alternative plans that are considered by the optimizer

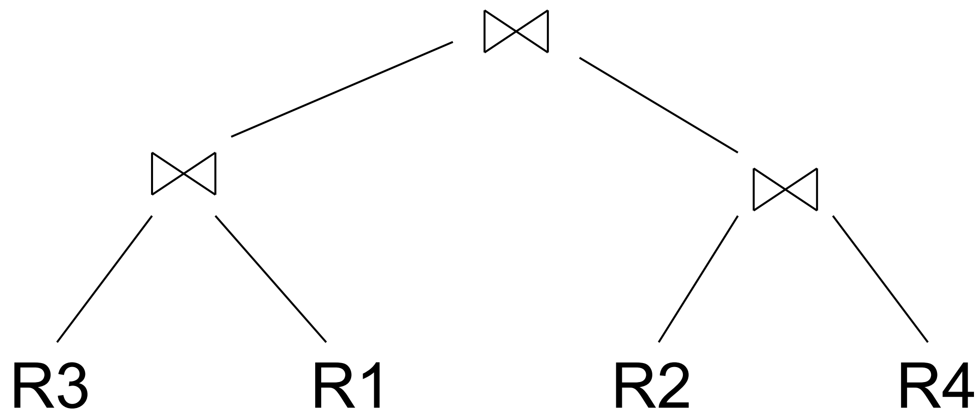
- Defined by the set of *algebraic laws* and the *set of plans* used by the optimizer

- Will discuss these laws next

Left-Deep Plans and Bushy Plans



Left-deep plan



Bushy plan

System R considered only left deep plans, and so do some optimizers today

Relational Algebra Laws

- Selections
  - Commutative: $\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R))$
  - Cascading: $\sigma_{c1 \wedge c2}(R) = \sigma_{c2}(\sigma_{c1}(R))$
- Projections
- Joins
  - Commutativity : $R \bowtie S = S \bowtie R$
  - Associativity: $R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$
  - Distributivity: $R \bowtie (S \cup T) = (R \bowtie S) \cup (R \bowtie T)$
  - Outer joins get more complicated

Example

Which plan is more efficient ?
$R \bowtie (S \bowtie T)$ or $(R \bowtie S) \bowtie T$?

- Assumptions:
  - Every join selectivity is 10%
    - That is: $T(R \bowtie S) = 0.1 * T(R) * T(S)$ etc.
  - B(R)=100, B(S) = 50, B(T)=500
  - All joins are main memory joins
  - All intermediate results are materialized

Example

- Example: R(A, B, C, D), S(E, F, G)
  - $\sigma_{F=3}(R \bowtie_{D=E} S) =$?
  - $\sigma_{A=5 \text{ AND } G=9}(R \bowtie_{D=E} S) =$?

Simple Laws

$\Pi_M(R \bowtie S) = \Pi_M(\Pi_P(R) \bowtie \Pi_Q(S))$
$\Pi_M(\Pi_N(R)) = \Pi_M(R)$ /* note that $M \subseteq N$ */

- Example R(A,B,C,D), S(E, F, G)

  $\Pi_{A,B,G}(R \bowtie_{D=E} S) = \Pi_{?}(\Pi_{?}(R) \bowtie_{D=E} \Pi_{?}(S))$

Laws for Group-by and Join

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

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

"Semantic Optimizations" = Laws that use a Constraint

Product(pid, pname, price, cid)
Company(cid, cname, city, state)

Foreign key

$$\Pi_{pid,\ price}(\text{Product} \bowtie_{cid=cid} \text{Company}) = \Pi_{pid,\ price}(\text{Product})$$

Need a second constraint for this law to hold. Which ?

Example

Foreign key

Product(pid, pname, price, cid)
Company(cid, cname, city, state)

```
CREATE VIEW CheapProductCompany
    SELECT *
    FROM Product x, Company y
    WHERE x.cid = y.cid and x.price < 100
```

```
SELECT pname, price
FROM CheapProductCompany
```

⇒

```
SELECT pname, price
FROM Product
```

Law of Semijoins

Recall the definition of a semijoin:

- $R \ltimes S = \Pi_{A1,\ldots,An} (R \bowtie S)$
- Where the schemas are:
  - Input: R(A1,…An), S(B1,…,Bm)
  - Output: T(A1,…,An)
- The law of semijoins is:

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

Laws with Semijoins

- Very important in parallel databases
- Often combined with Bloom Filters (next lecture)
- Read pp. 747 in the textbook

Semijoin Reducer

- Given a query:

$$Q = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$$

- A *semijoin reducer* for Q is

$$R_{i1} = R_{i1} \ltimes R_{j1}$$
$$R_{i2} = R_{i2} \ltimes R_{j2}$$
$$\ldots\ldots$$
$$R_{ip} = R_{ip} \ltimes R_{jp}$$

such that the query is equivalent to:

$$Q = R_{k1} \bowtie R_{k2} \bowtie \ldots \bowtie R_{kn}$$

- A *full reducer* is such that no dangling tuples remain

Example

- 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:

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

Why Would We Do This ?

- Large attributes:

$$Q = R(A, B, D, E, F, \ldots) \bowtie S(B, C, M, K, L, \ldots)$$

- Expensive side computations

$$Q = \gamma_{A,B,count(*)} R(A,B,D) \bowtie \sigma_{C=value}(S(B,C))$$

$$R_1(A,B,D) = R(A,B,D) \ltimes \sigma_{C=value}(S(B,C))$$
$$Q = \gamma_{A,B,count(*)} R_1(A,B,D) \bowtie \sigma_{C=value}(S(B,C))$$

Semijoin Reducer

- 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:

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

Are there dangling tuples ?

Semijoin Reducer

- Example:

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

- A full semijoin reducer is:

$$R_1(A,B) = R(A,B) \ltimes S(B,C)$$
$$S_1(B,C) = S(B,C) \ltimes R_1(A,B)$$

- The rewritten query is:

$$Q :- R_1(A,B) \bowtie S_1(B,C)$$

No more dangling tuples

Semijoin Reducer

- More complex example:

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

- A full reducer is:

$$S'(B,C) := S(B,C) \ltimes R(A,B)$$
$$T'(C,D,E) := T(C,D,E) \ltimes S(B,C)$$
$$S''(B,C) := S'(B,C) \ltimes T'(C,D,E)$$
$$R'(A,B) := R(A,B) \ltimes S''(B,C)$$

$$Q = R'(A,B) \bowtie S''(B,C) \bowtie T'(C,D,E)$$

Semijoin Reducer

- Example:

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

- Doesn’t have a full reducer (we can reduce forever)

Theorem a query has a full reducer iff it is “acyclic”
[*Database Theory*, by Abiteboul, Hull, Vianu]

Example with Semijoins

Emp(<u>eid</u>, ename, sal, did)
Dept(<u>did</u>, dname, budget)
DeptAvgSal(did, avgsal) /* view */

[Chaudhuri'98]

View:

```
CREATE VIEW DepAvgSal As (
        SELECT E.did, Avg(E.Sal) AS avgsal
        FROM Emp E
        GROUP BY E.did)
```

Query:

```
SELECT E.eid, E.sal
FROM Emp E, Dept D, DepAvgSal V
WHERE E.did = D.did AND E.did = V.did
        AND E.age < 30 AND D.budget > 100k
        AND E.sal > V.avgsal
```

Goal: compute only the necessary part of the view

Example with Semijoins

[Chaudhuri'98]

Emp(<u>eid</u>, ename, sal, did)
Dept(<u>did</u>, dname, budget)
DeptAvgSal(did, avgsal) /* view */

New view uses a reducer:

```
CREATE VIEW LimitedAvgSal As (
        SELECT E.did, Avg(E.Sal) AS avgsal
        FROM Emp E, Dept D
        WHERE E.did = D.did AND D.buget > 100k
        GROUP BY E.did)
```

New query:

```
SELECT E.eid, E.sal
FROM Emp E, Dept D, LimitedAvgSal V
WHERE E.did = D.did AND E.did = V.did
        AND E.age < 30 AND D.budget > 100k
        AND E.sal > V.avgsal
```

Example with Semijoins

Emp(eid, ename, sal, did)
Dept(did, dname, budget)
DeptAvgSal(did, avgsal) /* view */

[Chaudhuri'98]

Full reducer:

```
CREATE VIEW PartialResult AS
        (SELECT E.eid, E.sal, E.did
        FROM Emp E, Dept D
        WHERE E.did=D.did AND E.age < 30
        AND D.budget > 100k)

CREATE VIEW Filter AS
        (SELECT DISTINCT P.did FROM PartialResult P)

CREATE VIEW LimitedAvgSal AS
        (SELECT E.did, Avg(E.Sal) AS avgsal
        FROM Emp E, Filter F
        WHERE E.did = F.did GROUP BY E.did)
```

Example with Semijoins

New query:

```
SELECT P.eid, P.sal
FROM PartialResult P, LimitedDepAvgSal V
WHERE P.did = V.did AND P.sal > V.avgsal
```

Pruning the Search Space

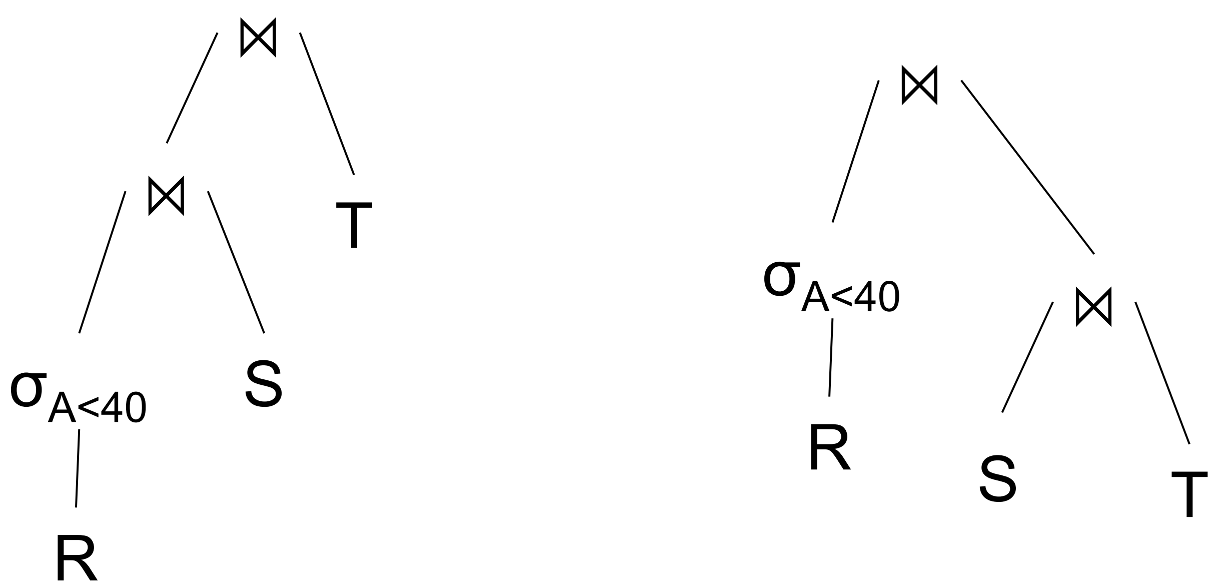
- Prune entire sets of plans that are unpromising

- The choice of *partial plans* influences how effective we can prune

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
```



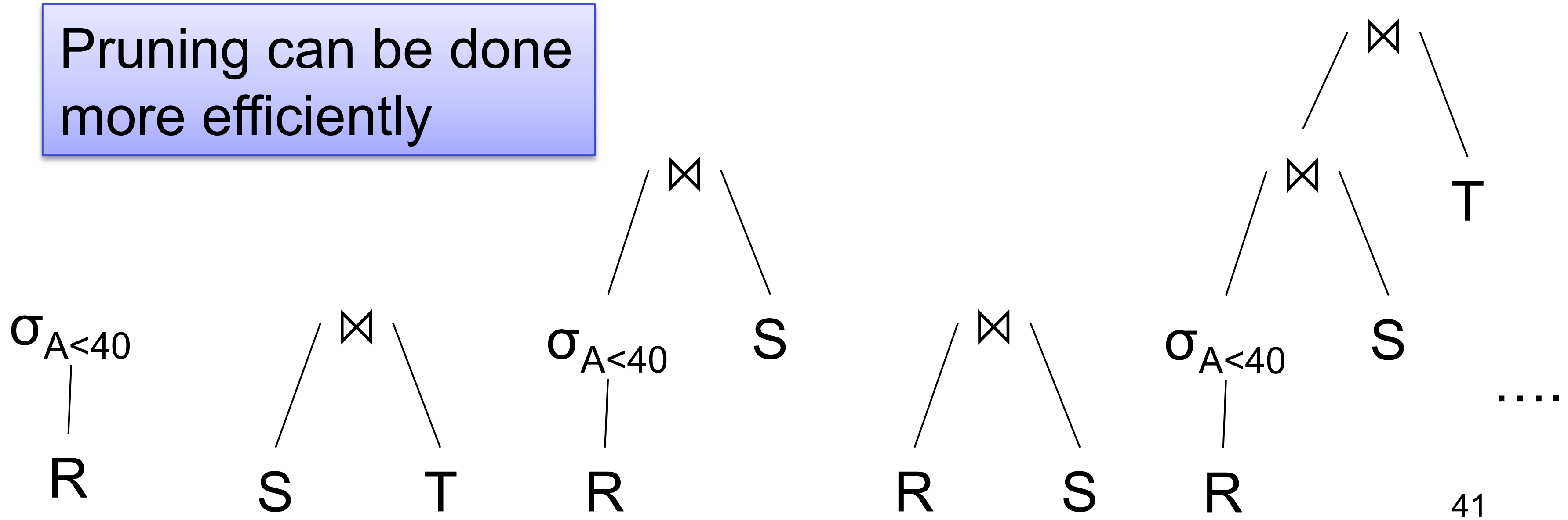
Pruning is difficult here.

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
```

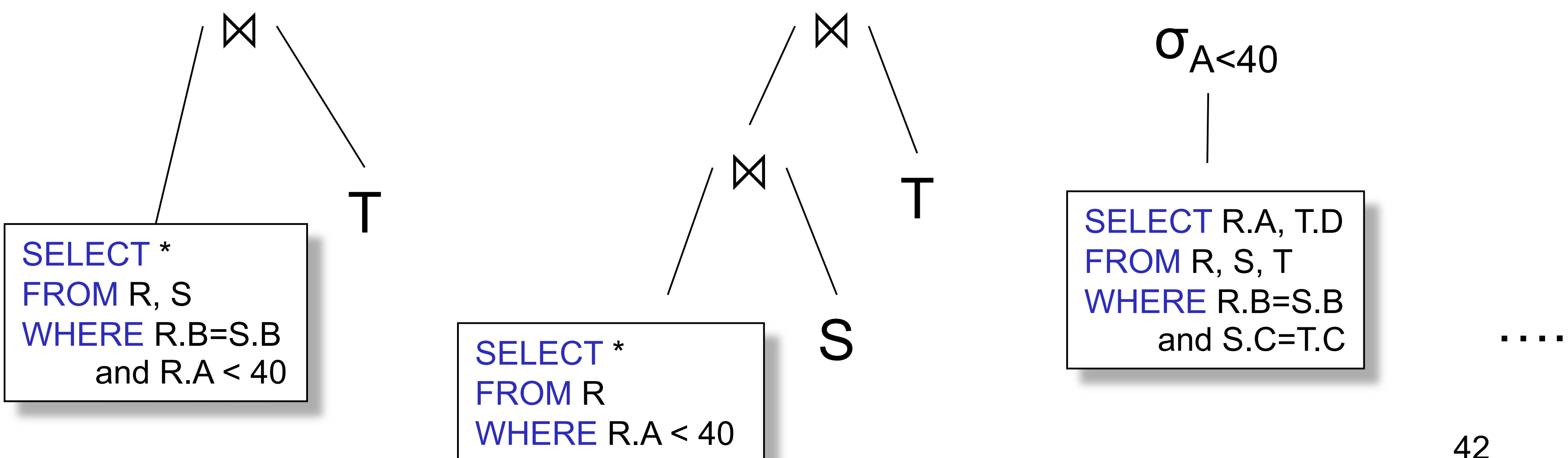
Pruning can be done more efficiently



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
```



Query Optimization

Three major components:

1. Search space

2. Algorithm for enumerating query plans

3. Cardinality and cost estimation

2. Plan Enumeration Algorithms

- System R (in class)
  - *Join reordering* – dynamic programming
  - *Access path selection*
  - Bottom-up; simple; limited
- Modern database optimizers (will not discuss)
  - Rule-based: database of rules (x 100s)
  - Dynamic programming
  - Top-down; complex; extensible

Join Reordering

System R [1979]

- Push all selections down (=early) in the query plan
- Pull all projections up (=late) in the query plan
- What remains are joins:

```
SELECT list
FROM   R1, …, Rn
WHERE  cond1 AND cond2 AND . . . AND condk
```

```
SELECT list
FROM   R1, …, Rn
WHERE  cond1 AND cond2 AND . . . AND condk
```

Join Reordering

Dynamic programming

- For each subquery Q ⊆{R1, …, Rn}, compute the optimal join order for Q
- Store results in a table: 2^n-1 entries
  - Often much fewer entries

```
SELECT list
FROM   R1, …, Rn
WHERE cond1 AND cond2 AND . . . AND condk
```

Join Reordering

Step 1: For each $\{R_i\}$ do:

- Initialize the table entry for $\{R_i\}$ with the cheapest access path for R_i

Step 2: For each subset $Q \subseteq \{R_1, \ldots, R_n\}$ do:

- For every partition $Q = Q' \cup Q''$
- Lookup optimal plan for Q' and for Q'' in the table
- Compute the cost of the plan $Q' \bowtie Q''$
- Store the cheapest plan $Q' \bowtie Q''$ in table entry for Q

Reducing the Search Space

Restriction 1: only left linear trees (no bushy)

Restriction 2: no trees with cartesian product

R(A,B) ⋈ S(B,C) ⋈ T(C,D)

Plan: (R(A,B)⋈T(C,D)) ⋈ S(B,C)
has a cartesian product.
Most query optimizers will not consider it

Access Path Selection

- **Access path**: a way to retrieve tuples from a table
  - A file scan
  - An index *plus* a matching selection condition

- Index matches selection condition if it can be used to retrieve just tuples that satisfy the condition
  - Example: Supplier(sid,sname,scity,sstate)
  - B+-tree index on (scity,sstate)
    - matches scity='Seattle'
    - does not match sid=3, does not match sstate='WA'

Access Path Selection

- Supplier(sid,sname,scity,sstate)
- Selection condition: sid > 300 ∧ scity='Seattle'
- Indexes: B+-tree on sid and B+-tree on scity

Access Path Selection

- Supplier(sid,sname,scity,sstate)
- Selection condition: sid > 300 ∧ scity='Seattle'
- Indexes: B+-tree on sid and B+-tree on scity
- Which access path should we use?

Access Path Selection

- Supplier(sid,sname,scity,sstate)
- Selection condition: sid > 300 ∧ scity='Seattle'
- Indexes: B+-tree on sid and B+-tree on scity
- Which access path should we use?
- We should pick the **most selective** access path

Access Path Selectivity

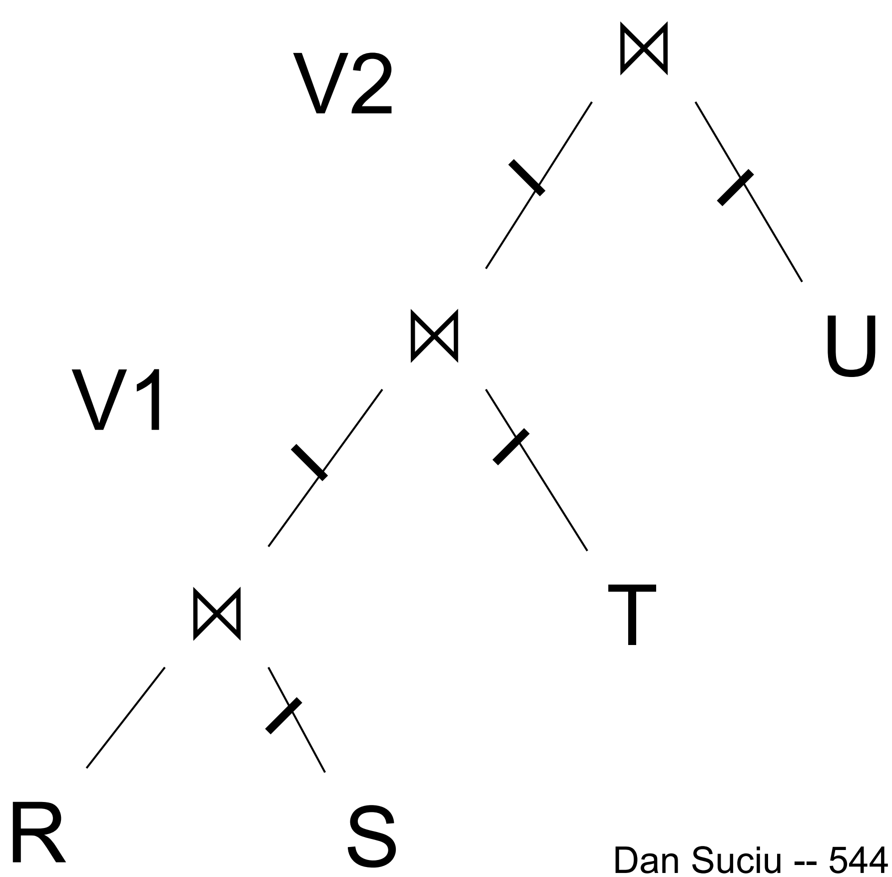
- **Access path selectivity is the number of pages retrieved if we use this access path**
  - Most selective retrieves fewest pages

- As we saw earlier, **for equality predicates**
  - Selection on equality: $\sigma_{a=v}(R)$
  - V(R, a) = # of distinct values of attribute a
  - 1/V(R,a) is thus the reduction factor
  - Clustered index on a: cost B(R)/V(R,a)
  - Unclustered index on a: cost T(R)/V(R,a)
  - (we are ignoring I/O cost of index pages for simplicity)

Other Decisions for the Optimization Algorithm

- How much memory to allocate to each operator
- Pipeline or materialize (next)

Materialize Intermediate Results Between Operators




```
HashTable ← S
repeat    read(R, x)
          y ← join(HashTable, x)
          write(V1, y)

HashTable ← T
repeat    read(V1, y)
          z ← join(HashTable, y)
          write(V2, z)

HashTable ← U
repeat    read(V2, z)
          u ← join(HashTable, z)
          write(Answer, u)
```

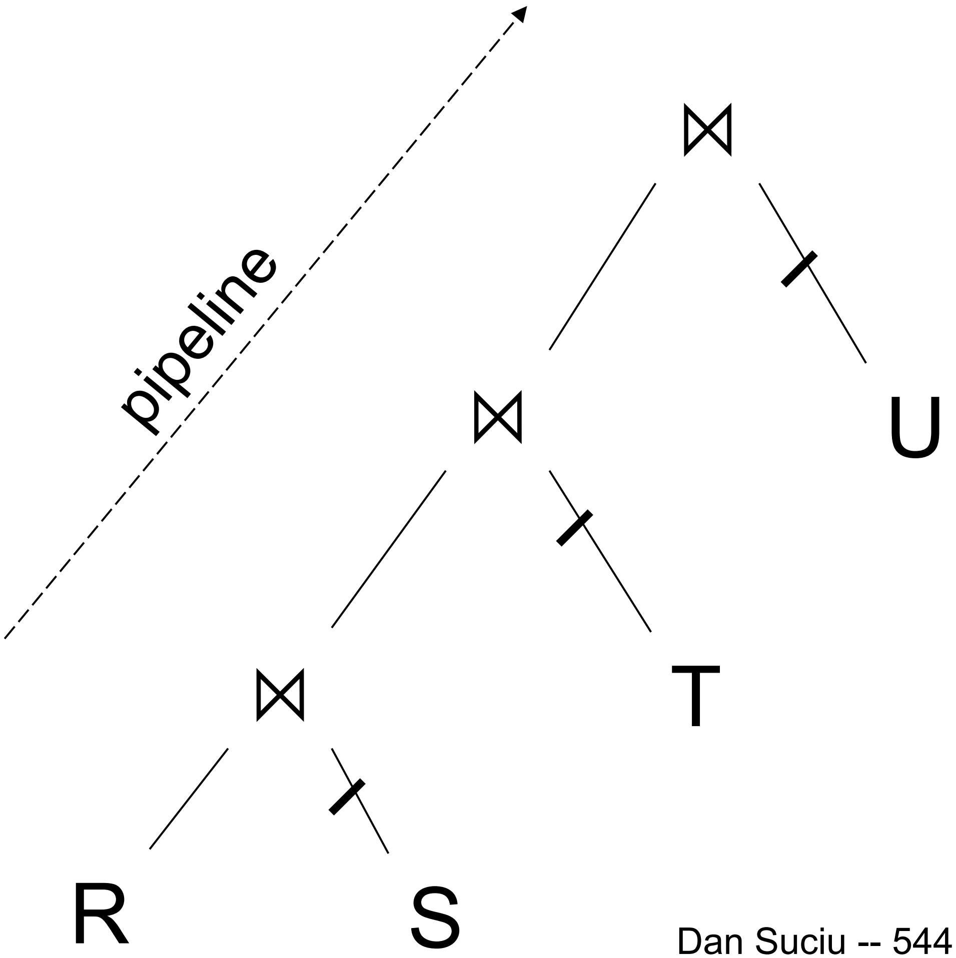
Materialize Intermediate Results Between Operators

Question in class

Given B(R), B(S), B(T), B(U)

- What is the total cost of the plan ?
  - Cost =
- How much main memory do we need ?
  - M =

Pipeline Between Operators




```
HashTable1 ← S
HashTable2 ← T
HashTable3 ← U
repeat   read(R, x)
         y ← join(HashTable1, x)
         z ← join(HashTable2, y)
         u ← join(HashTable3, z)
         write(Answer, u)
```

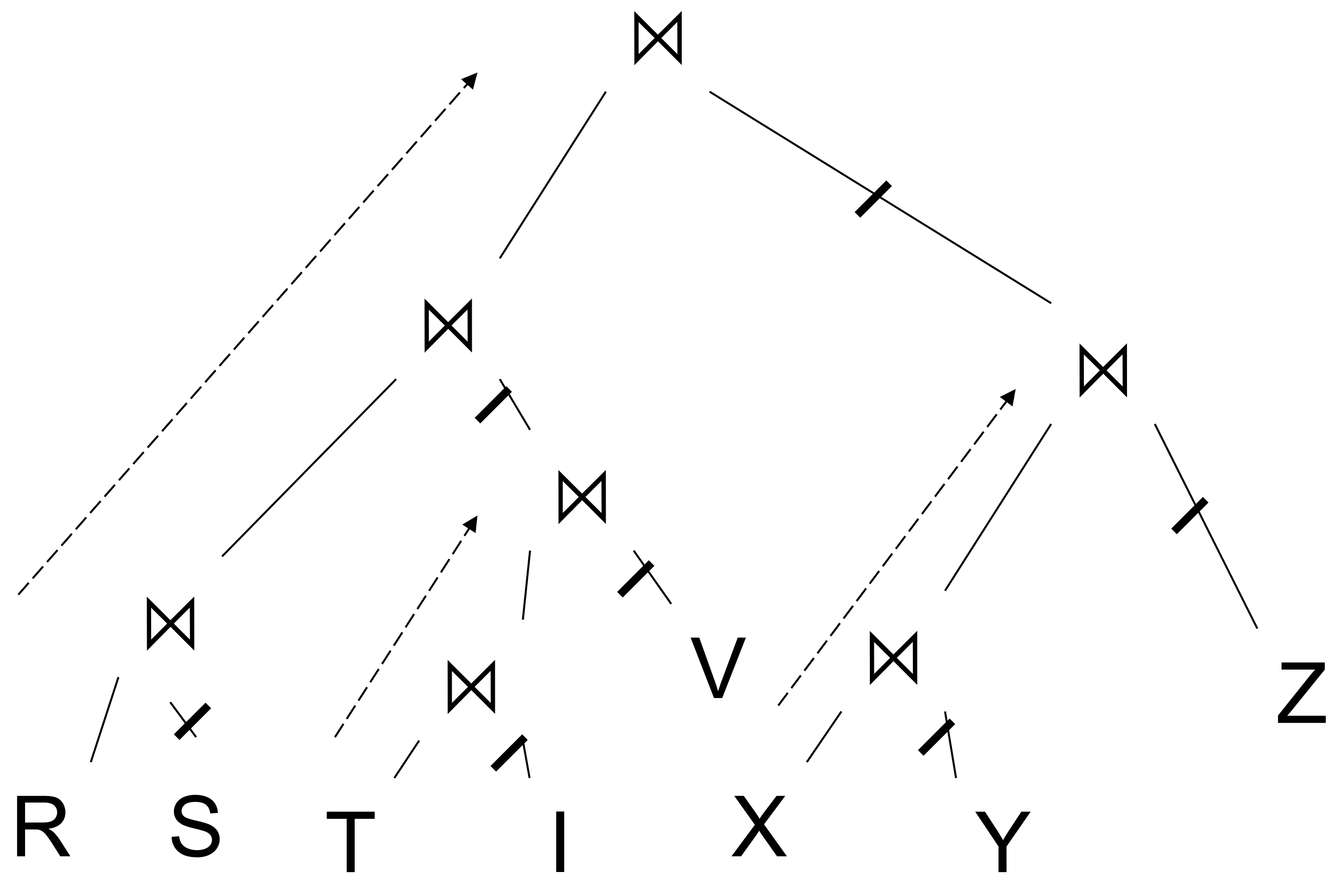
Pipeline Between Operators

Question in class

Given B(R), B(S), B(T), B(U)

- What is the total cost of the plan ?
  - Cost =
- How much main memory do we need ?
  - M =

Pipeline in Bushy Trees



Query Optimization

<u>Three major components</u>:

1. Search space

2. Algorithm for enumerating query plans

3. Cardinality and cost estimation