CSE544: Principles of Database Systems

Query Optimization

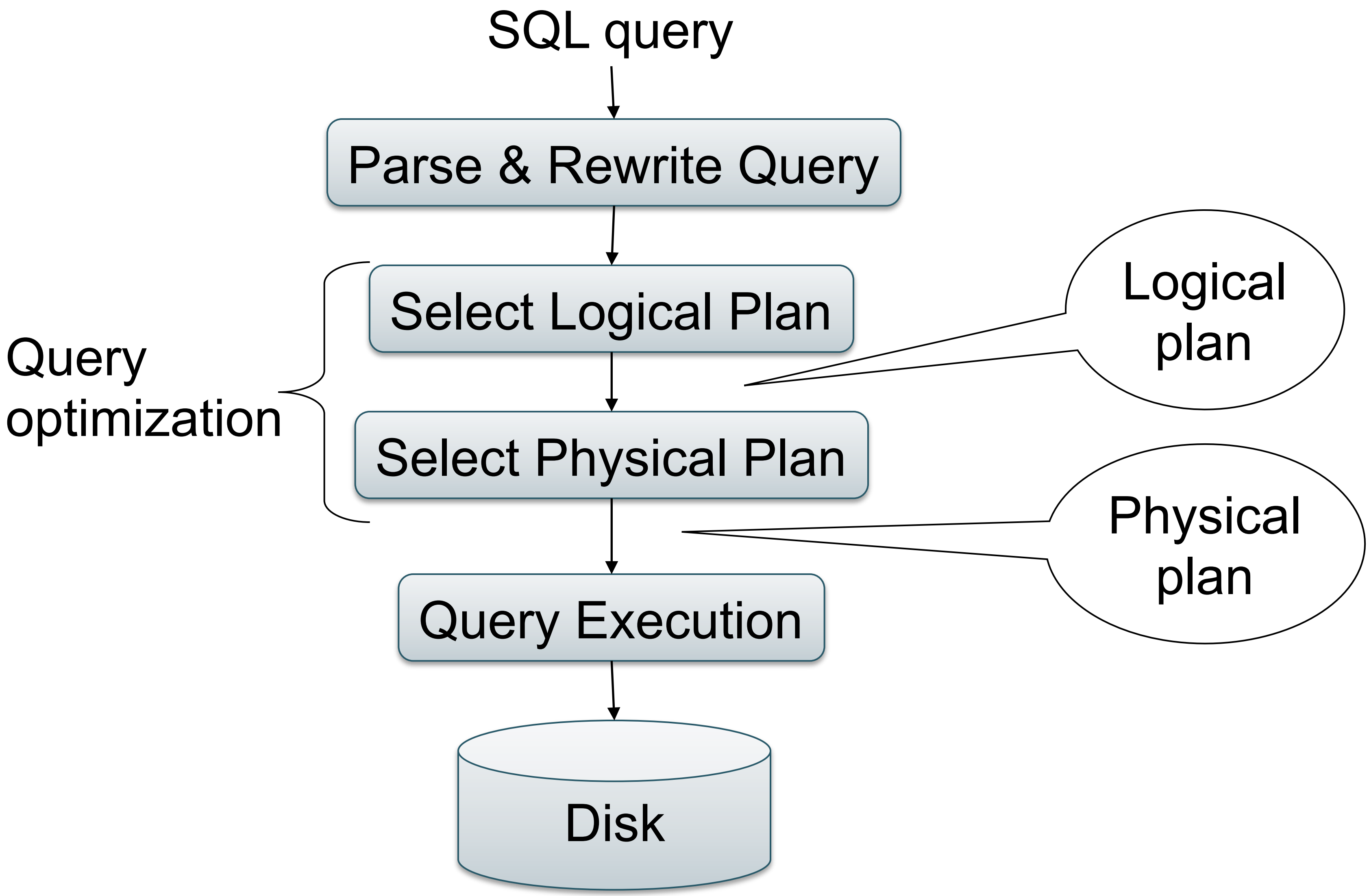
Announcements

- Project proposals due on Sunday, April 22
- Paper review for Wednesday
- Homework 2:
  - Questions for Part A → Paris
  - Questions for Part B → Dan
  - Questions for Part C → you

Outline

- Finish Query Execution
- Chapter 15 in the textbook

Steps of the Query Processor



Query Execution: Final Thoughts

Index Based Selection

Recall IMDB; assume indexes on Movie.id, Movie.year

```
SELECT *
FROM Movie
WHERE id = '12345'
```

```
SELECT *
FROM Movie
WHERE year = '1995'
```

B(Movie) = 10k
T(Movie) = 1M

What is your estimate of the I/O cost ?

Index Based Selection

Recall IMDB; assume indexes on Movie.id, Movie.year

```
SELECT *
FROM Movie
WHERE id = '12345'
```

B(Movie) = 10k
T(Movie) = 1M

Answer: 1

```
SELECT *
FROM Movie
WHERE year = '1995'
```

Answer:

- Clustered index → 10k/100 =100
- Unclustered index → 1M/100 =10k

assuming ≈ 100 years=V(Movie,year)

Cost formula for Index Based Selection

Selection on equality: $\sigma_{A=v}(R)$

- Clustered index on A: B(R)/V(R,A)
- Unclustered index : T(R)/V(R,A)

Rule of thumb:
don't build unclustered indexes when V(R,A) is small !

Index Based Join

- $R \bowtie_{A=B} S$
- Assume S has an index on B

```
for each tuple r in R do
    fetch tuples s in S using the index S(B)
    output (r,s)
```

Cost formula for Index Based Join

Cost of $R \bowtie_{A=B} S$:

- If index is clustered: $B(R) + T(R)B(S)/V(S,B)$
- If unclustered: $B(R) + T(R)T(S)/V(S,B)$

Summary of Query Execution Algorithms

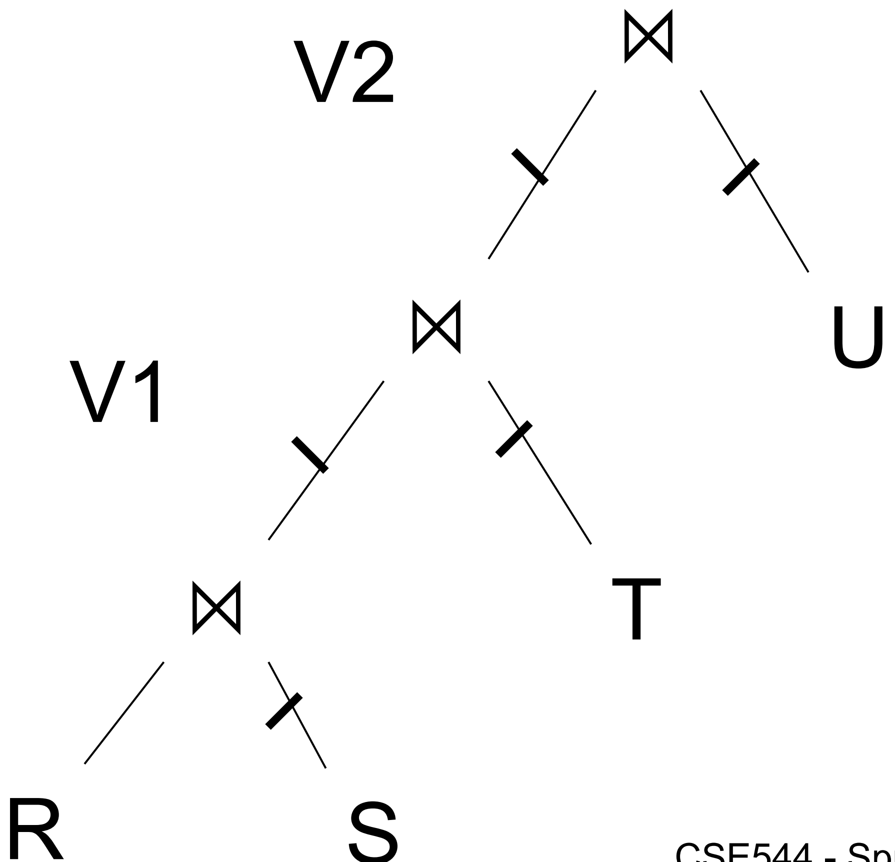
- Join $\bowtie$; Group+aggregate γ
  - Hash-based algorithms
  - Merge-sort based algorithms
  - Cost = $3B(R)+3B(S)$
- Join $R \bowtie_{A=B} S$:
  - Nested Loop join: cost = $B(R) + T(R)*B(S)$
  - Block nested loop join: cost = $B(R) + B(R)*B(S)/M$
  - Index based:
    - Clustered: cost = $B(R) + T(R)*B(S)/V(S,B)$
    - Unclustered: cost = $B(R) + T(R)*T(S)/V(S,B)$

Combining Operators

Two options:

- Materialize intermediate results
- Pipeline intermediate results

Materialize




```
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

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

Assume we do main-memory hash-join

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

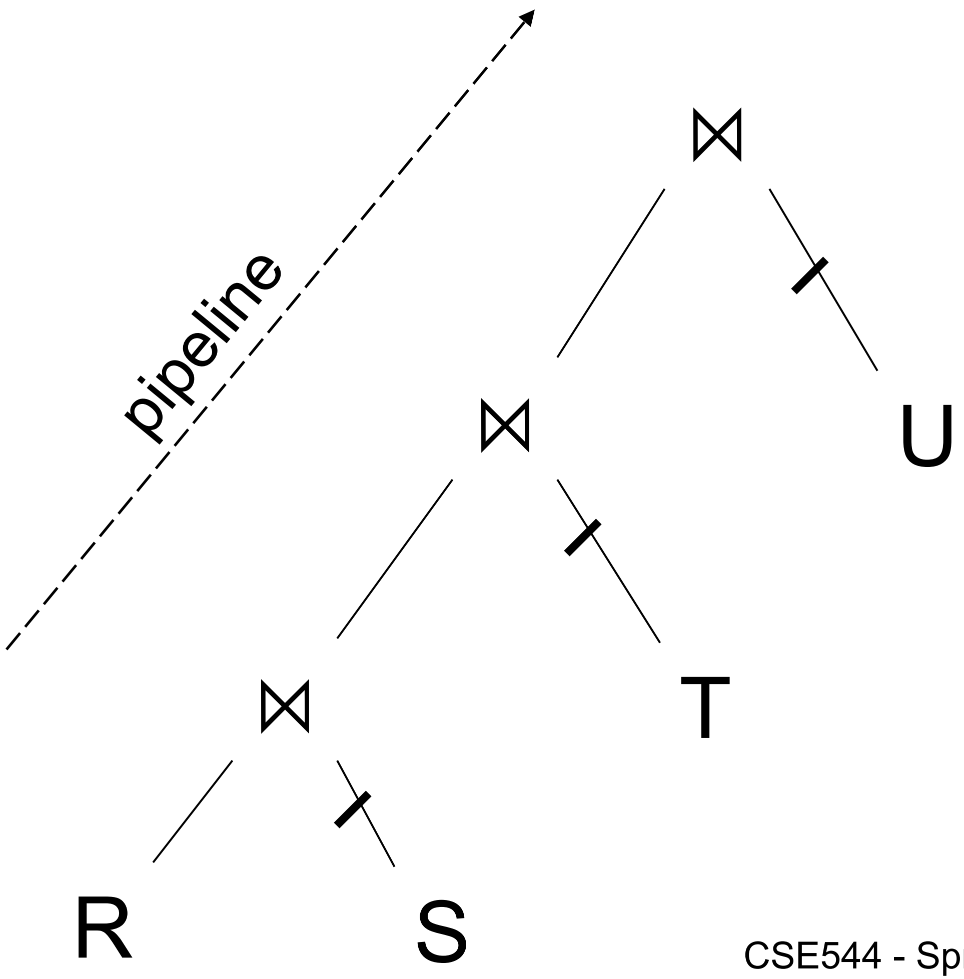
Materialize

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

Assume we do main-memory hash-join

- What is the total cost of the plan ?
  - Cost = B(R)+B(S)+B(T)+B(U)+2B(V1)+2B(V2)
- How much main memory do we need ?
  - M = max(B(S), B(T), B(U))

Pipeline




```
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

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

Assume we do main-memory hash-join

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

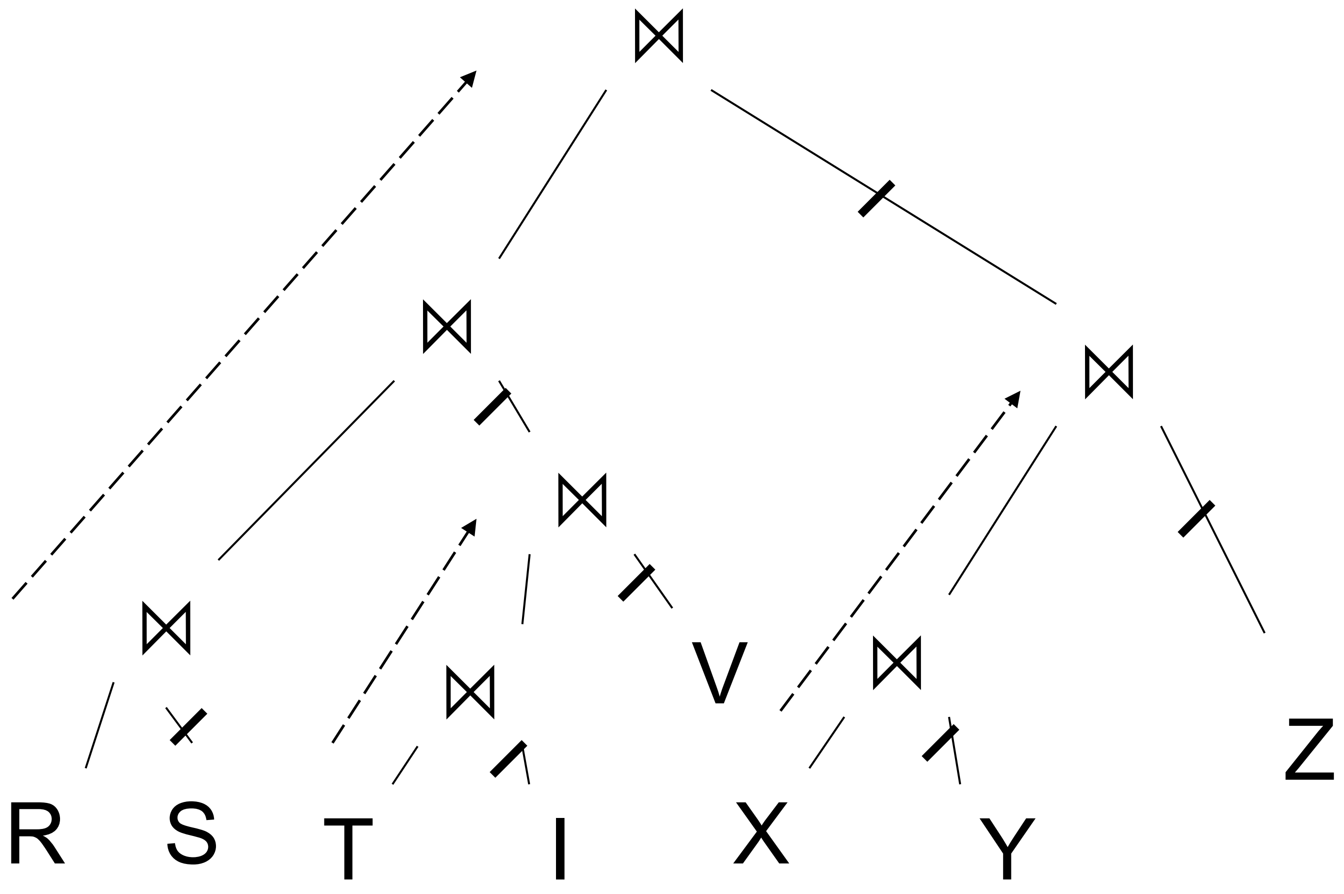
Pipeline

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

Assume we do main-memory hash-join

- What is the total cost of the plan ?
  - Cost = B(R)+B(S)+B(T)+B(U)~~+2B(V1)+2B(V2)~~
- How much main memory do we need ?
  - M = B(S) + B(T) + B(U) (~~max(B(S), B(T), B(U))~~)

Pipeline in Bushy Trees



Query Optimization

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

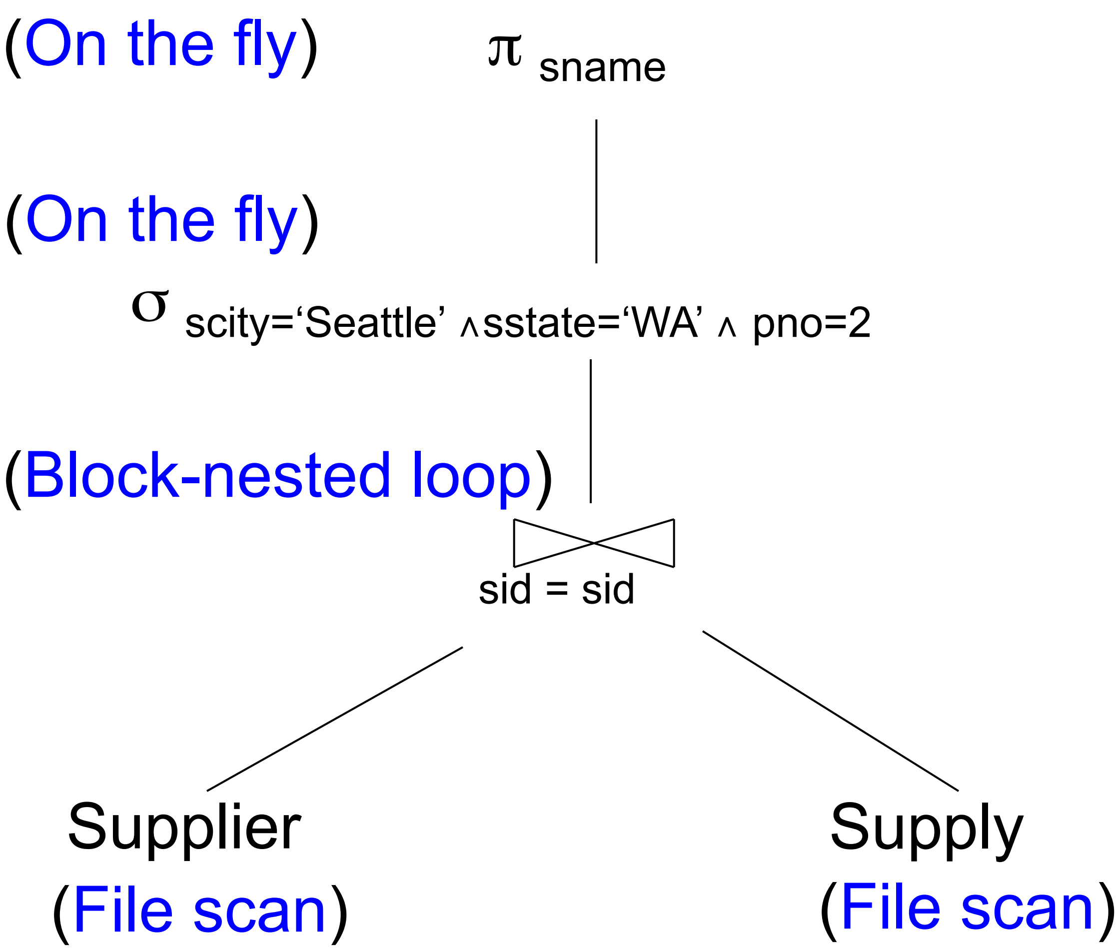
T(Supplier) = 1000
T(Supply) = 10,000

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



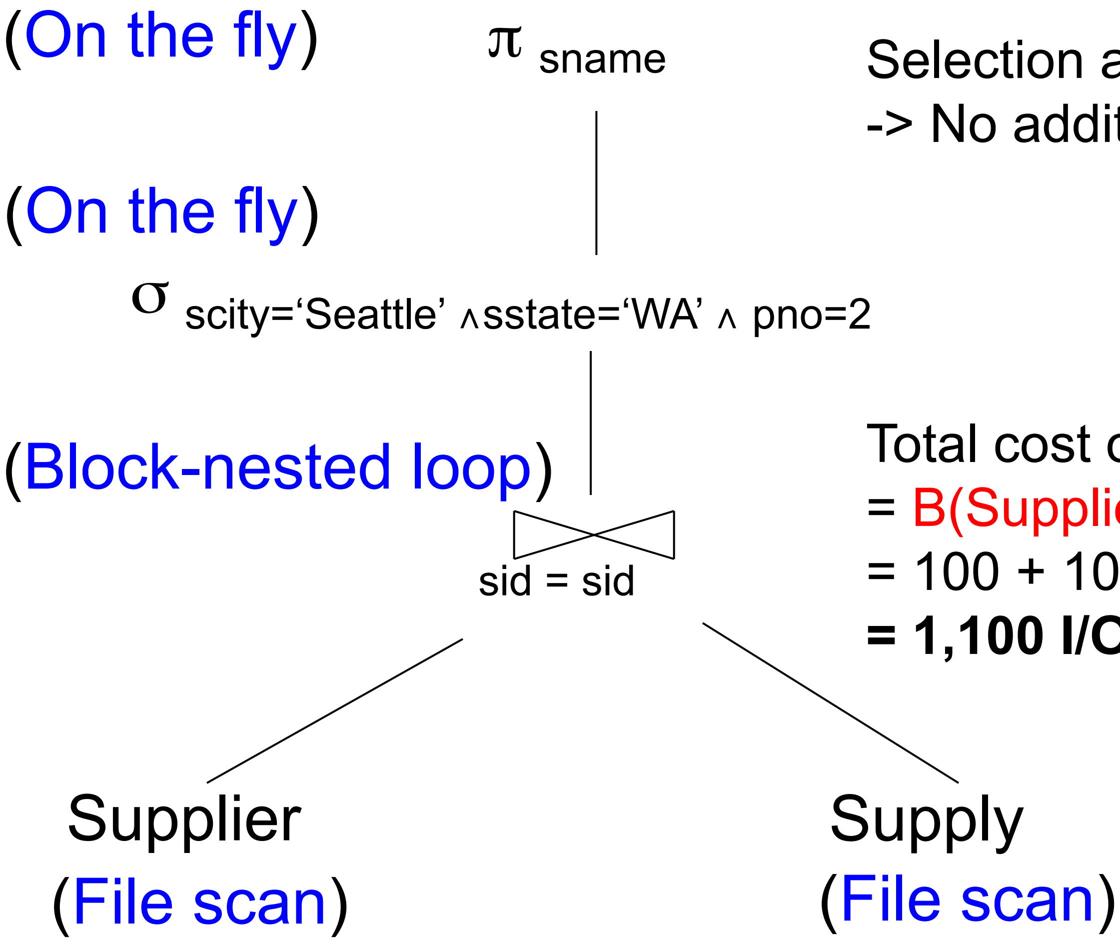
T(Supplier) = 1000
T(Supply) = 10,000

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

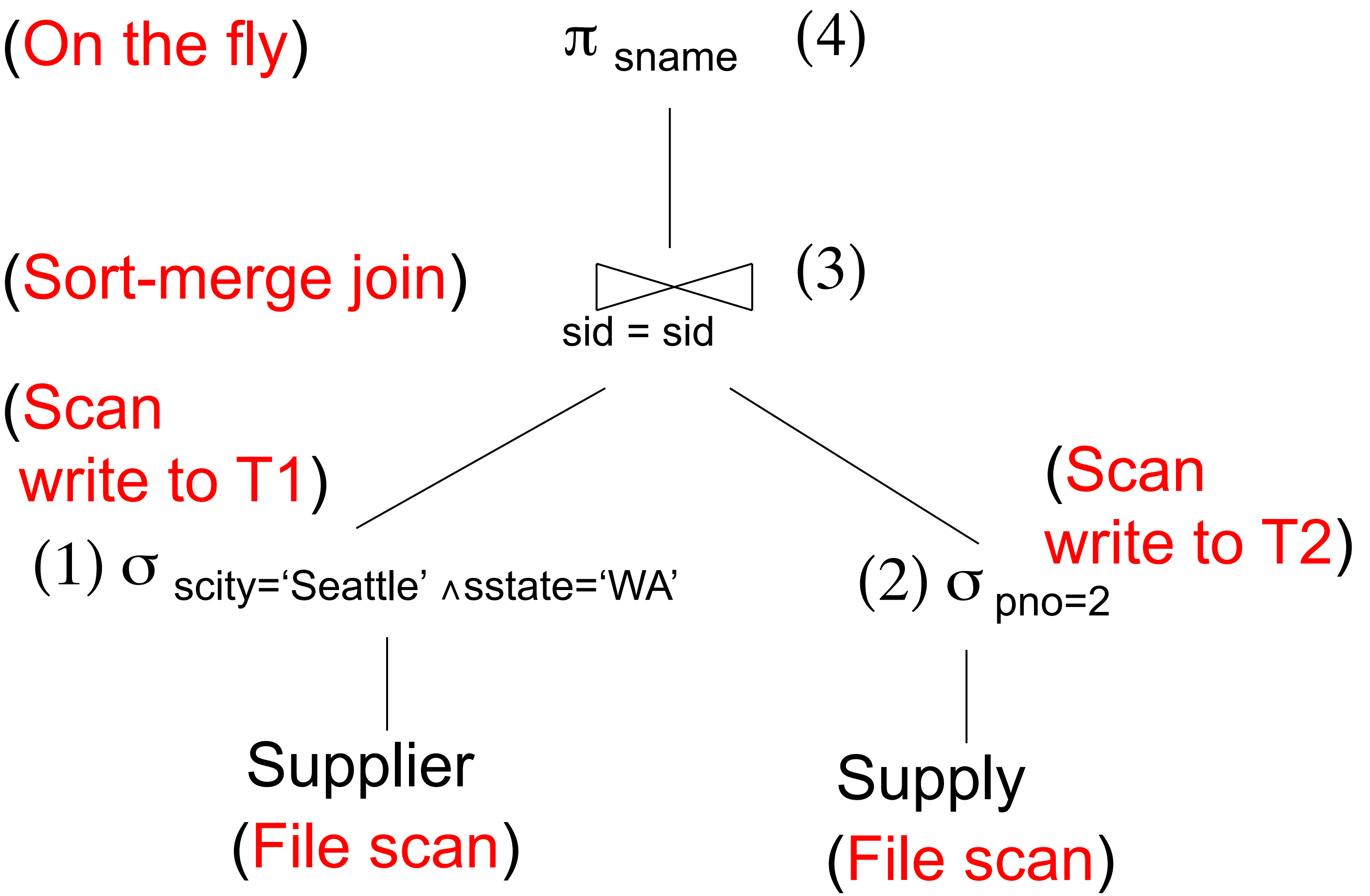
T(Supplier) = 1000
T(Supply) = 10,000

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

(On the fly) (4) π sname

(Sort-merge join) (3) ⋈ sid = sid

(Scan write to T1) (1) σ scity='Seattle' ∧sstate='WA'

(Scan write to T2) (2) σ pno=2

Supplier (File scan)

Supply (File scan)

Total cost=
B(Supplier)+B(Supplier)/V(Supplier,scity)/V(Supplier,sstate)
+ B(Supply) + B(Supply)/V(Supplier,pno) + [merge join]
= 100 + 100 * 1/20 * 1/10 (1)
+ 100 + 100 * 1/2500 (2)
+ 2 (3)
+ 0 (4)
Total cost ≈ **204 I/Os**

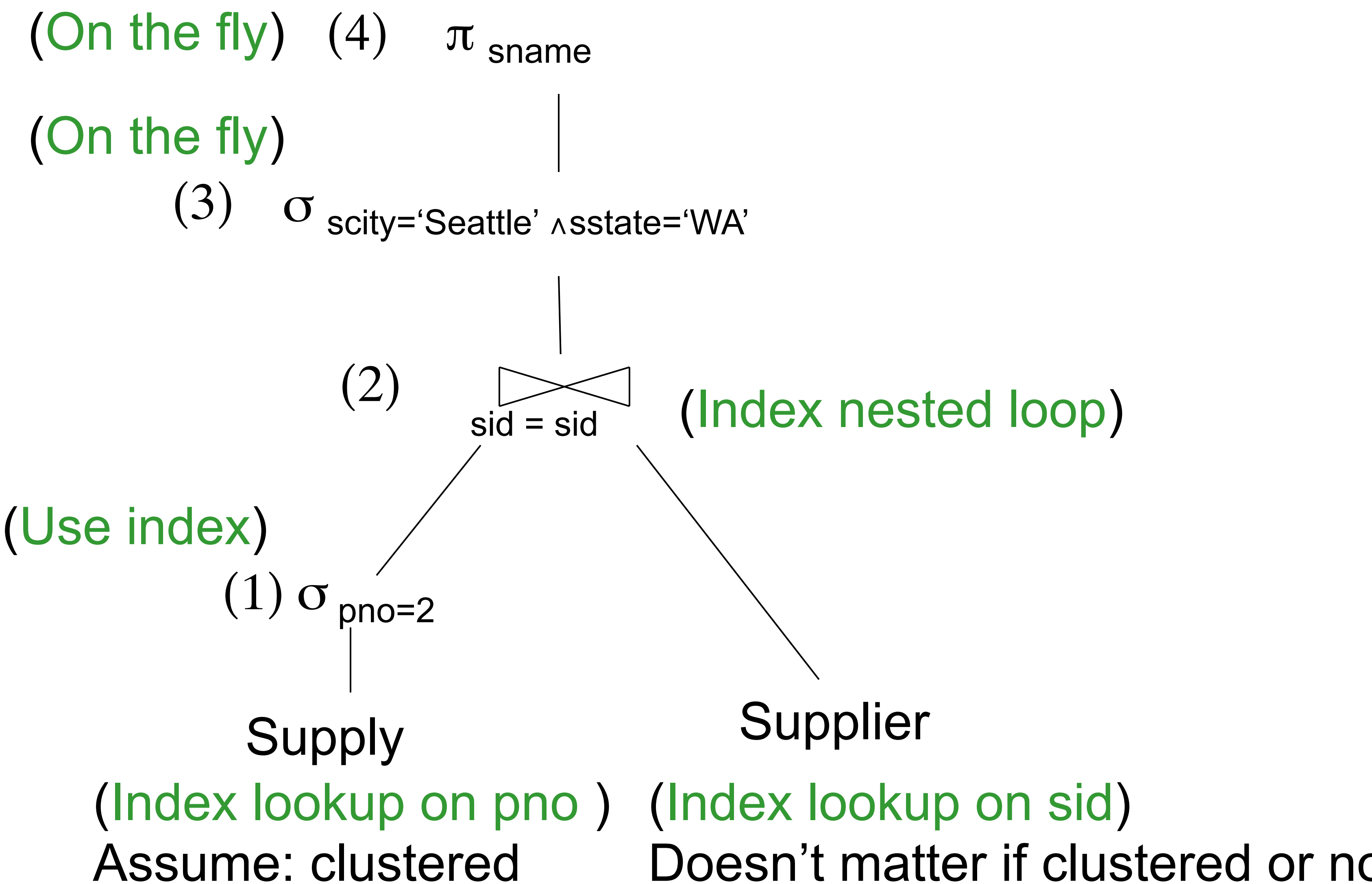
T(Supplier) = 1000
T(Supply) = 10,000

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



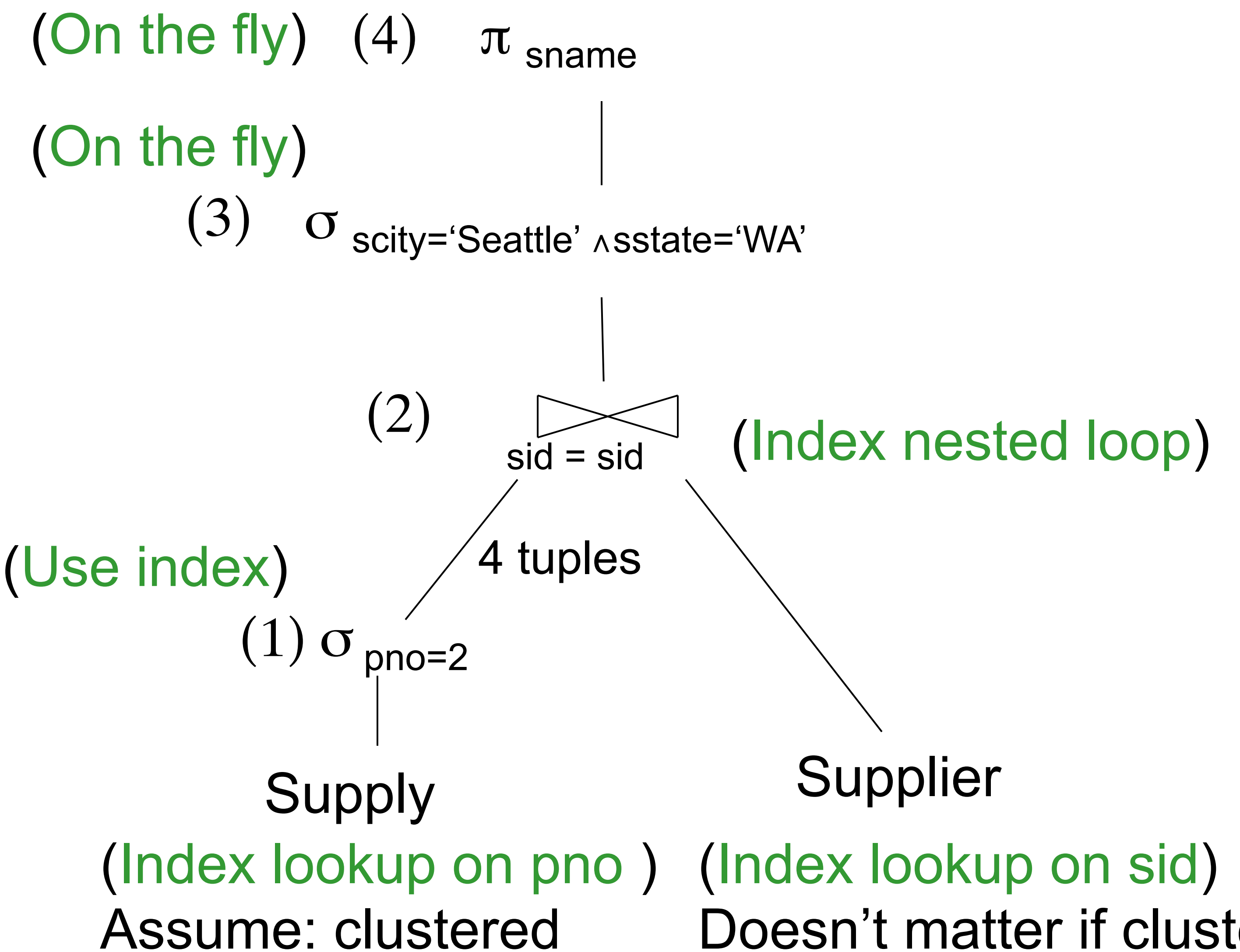
T(Supplier) = 1000
T(Supply) = 10,000

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



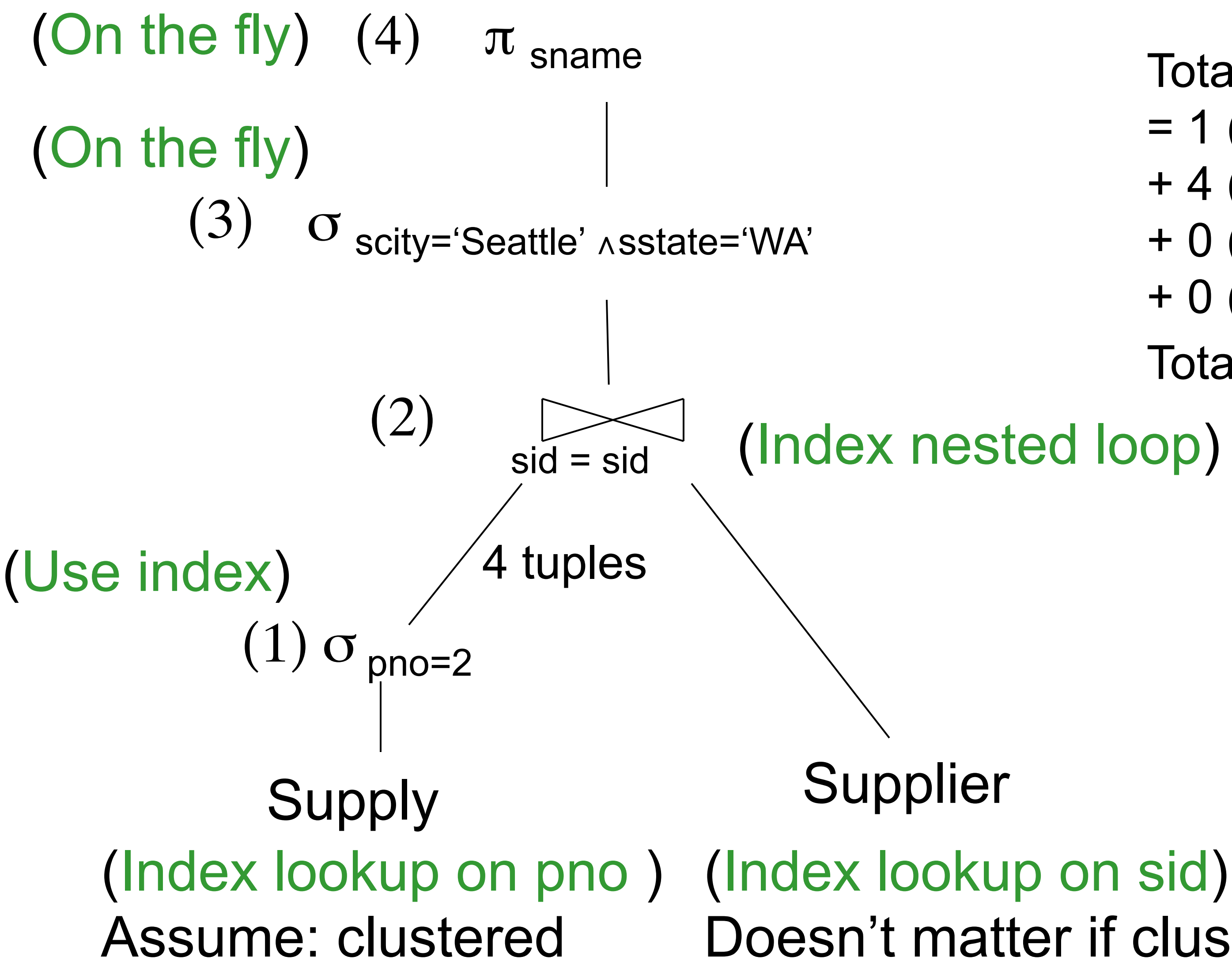
T(Supplier) = 1000
T(Supply) = 10,000

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

More Lessons

[Chaudhuri]

3. The plan depends a lot on the statistics of the selection predicates

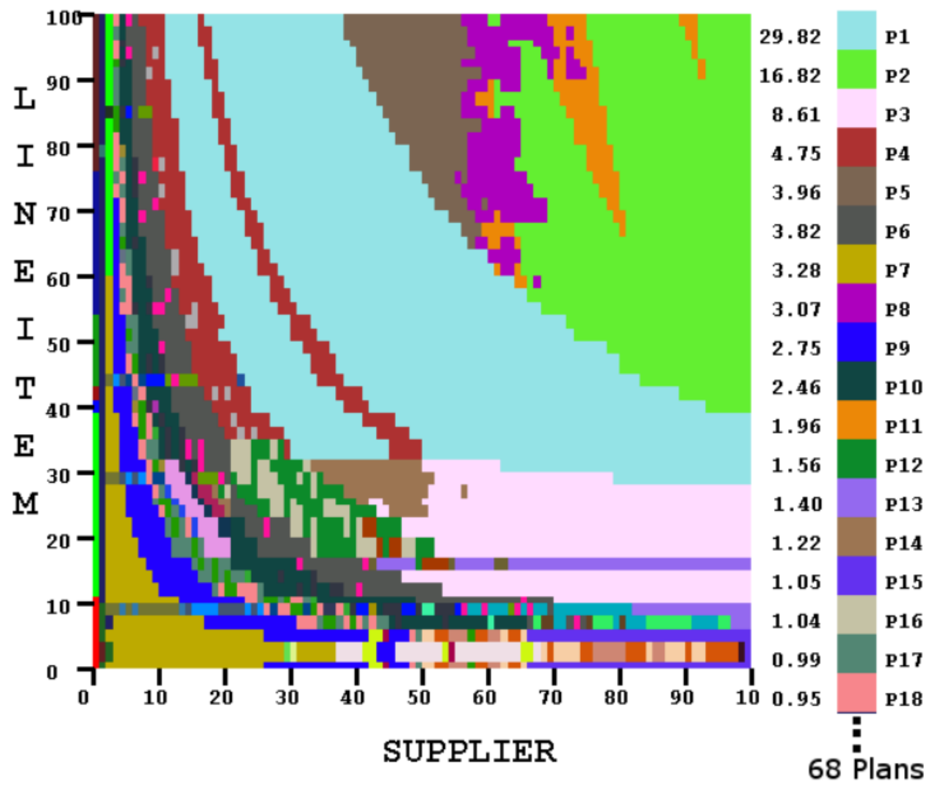


Figure 1: Plan diagram for TPC-H Query 8

The "prepare" statement must choose a plan without knowing the actual predicate values.
Discuss the *Anatomy* paper

Query Optimization

<u>Three major components</u>:

1. Search space

2. Plan enumeration algorithms

3. Cardinality and cost estimation

History of Query Optimization

- First query optimizer: System R, IBM,1979
- It had all three components in place
- You will see often references to System R
- See Section 15.6 in the book

1. Search Space

1. Search Space

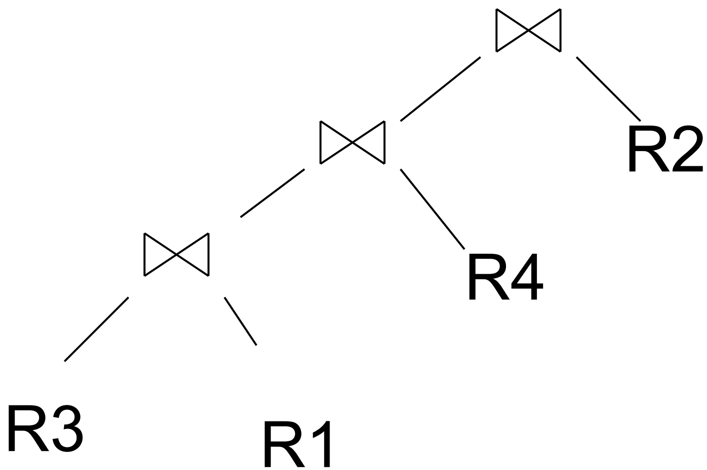
- This is the set of all alternative plans that are considered by the optimizer
- Defined by the set of *algebraic laws* and the *set of plans* used by the optimizer

Relational Algebra Laws: Joins

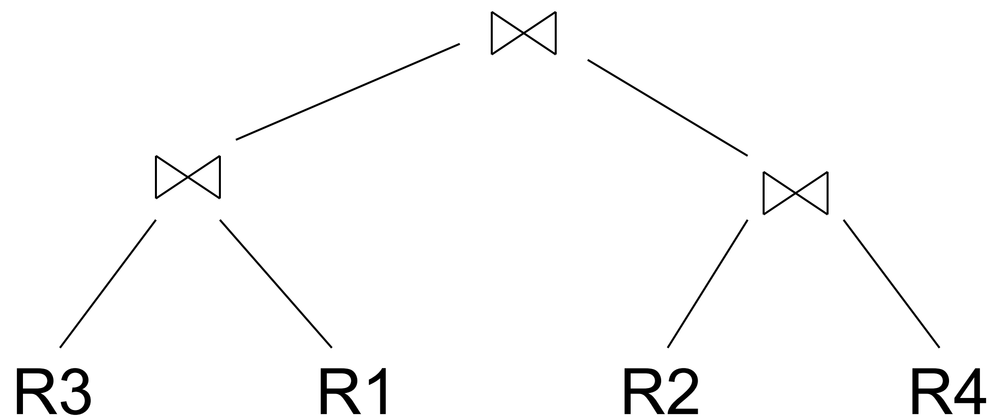
Commutativity :	$R \bowtie S = S \bowtie R$
Associativity:	$R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$
Distributivity:	$R \bowtie (S \cup T) = (R \bowtie S) \cup (R \bowtie T)$

Outer joins get more complicated

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

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

$\sigma_{F=3} (R \bowtie_{D=E} S) =$?

$\sigma_{A=5 \text{ AND } G=9} (R \bowtie_{D=E} S) =$?

Relational Algebra Laws: Selections

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

$$\sigma_{F=3}(R \bowtie_{D=E} S) = R \bowtie_{D=E} (\sigma_{F=3}(S))$$

$$\sigma_{A=5 \text{ AND } G=9}(R \bowtie_{D=E} S) = \sigma_{A=5}(R) \bowtie_{D=E} \sigma_{G=9}(S)$$

Group-by and Join

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

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

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.

Laws Involving Constraints

Foreign key

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

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

Laws Involving Constraints

Foreign key

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

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

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

Why such queries occur

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
WHERE price < 100
```

Law of Semijoins

Recall the definition of a semijoin:

- $R \ltimes S = \Pi_{A1,\ldots,An} (R \bowtie S)$
- 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 (my plan is to discuss them in the next lecture)
- Read pp. 747 in the textbook

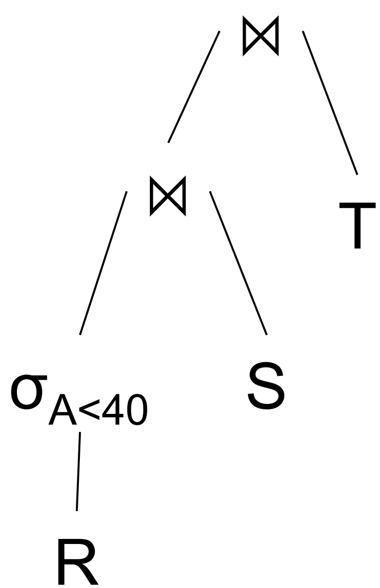
Pruning the Search Space

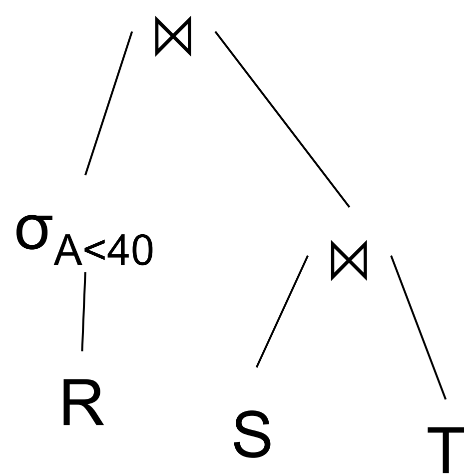
- Prune entire sets of plans that are unpromising
- The choice of *partial plans* influences how effective we can prune

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
```





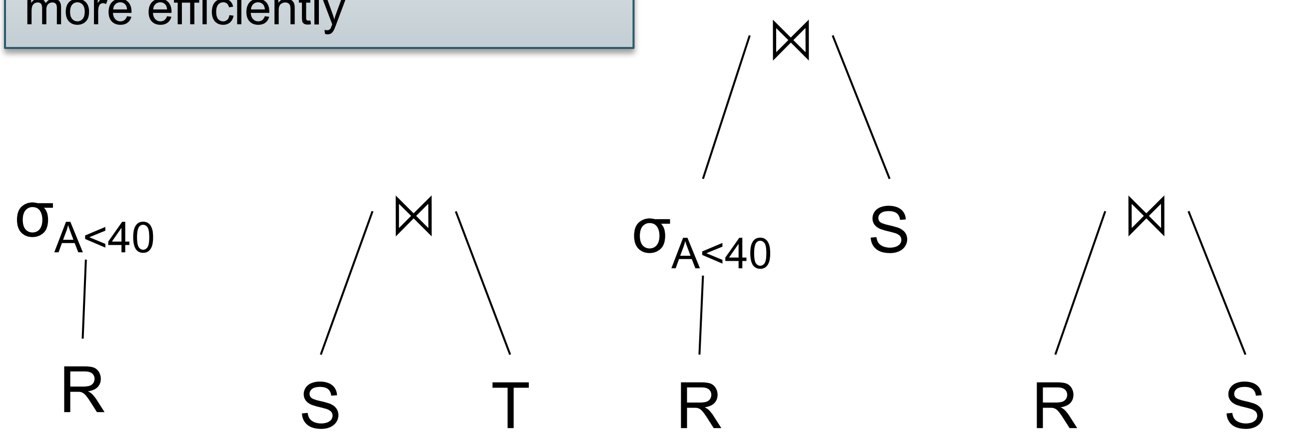
If the algorithm enumerates ***complete*** plans, then it is difficult to prune out unpromising sets of plans.

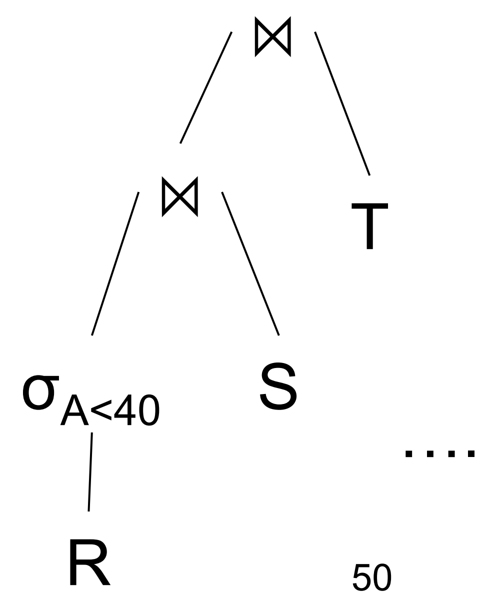
Bottom-up Partial Plans

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
```

If the algorithm enumerates partial bottom-up plans, then pruning can be done more efficiently



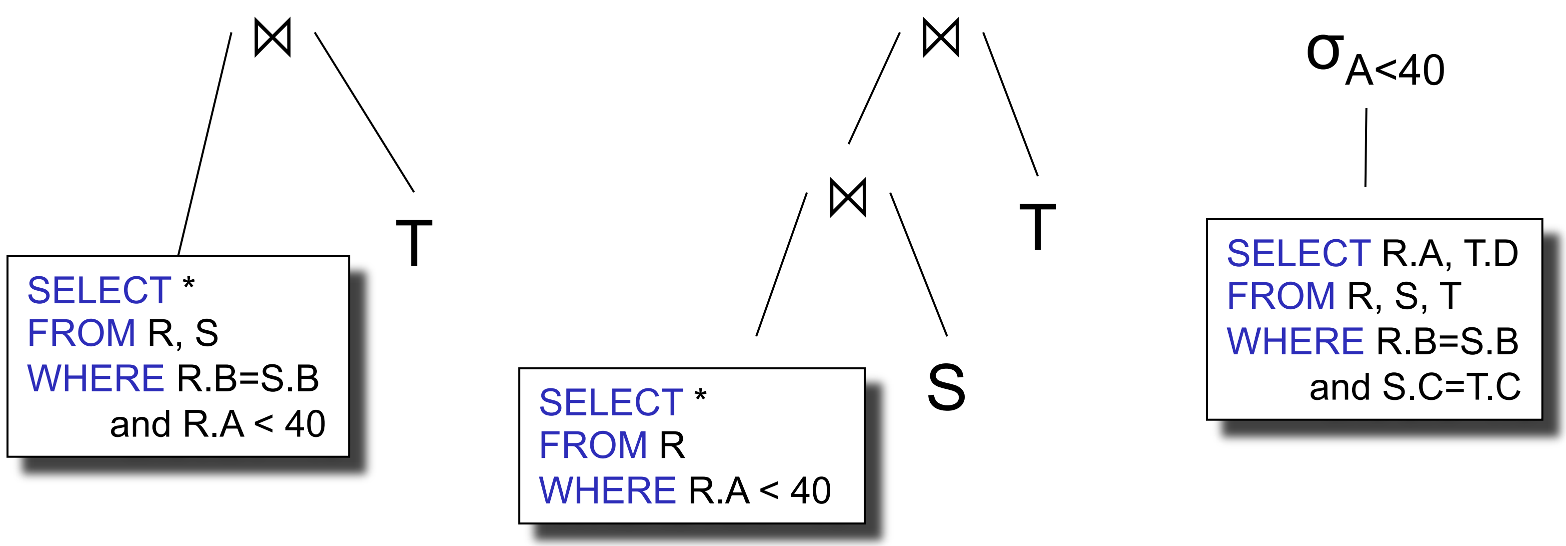


Top-down Partial Plans

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
```

Same here.



…..

Query Optimization

<u>Three major components</u>:

1. Search space
2. Algorithm for enumerating query plans
3. Cardinality and cost estimation

2. Algorithm for enumerating query plans

2. Plan Enumeration Algorithms

- System R
  - *Join reordering* – dynamic programming
  - *Access path selection*
  - Bottom-up; simple; limited
- Modern database optimizers
  - Rule-based: database of rules (x 100s)
  - Dynamic programming
  - Top-down; complex; extensible

We won't discuss them. See book for some more details

Access Path Selection

Supplier(sid,sname,scategory,scity,sstate)

B(Supplier) = 10k
T(Supplier) = 1M

$\sigma_{scategory = \text{'organic'} \wedge scity=\text{'Seattle'}}$ (Supplier)

V(Supplier,city) = 1000
V(Supplier,scategory)=100

Clustered index on scity
Unclustered index on (scategory,scity)

Access plan options:

- Table scan: cost = ?
- Index scan on scity: cost = ?
- Index scan on scategory,scity: cost = ?

Access Path Selection

Supplier(sid,sname,scategory,scity,sstate)

B(Supplier) = 10k
T(Supplier) = 1M

$\sigma_{\text{scategory = 'organic'} \wedge \text{scity='Seattle'}}$ (Supplier)

V(Supplier,city) = 1000
V(Supplier,scategory)=100

Clustered index on scity
Unclustered index on (scategory,scity)

Access plan options:

- Table scan: cost = 10k = 10k
- Index scan on scity: cost = 10k/1000 = 10
- Index scan on scategory,scity: cost = 1M/1000*100 = 10

Query Optimization

Three major components:

1. Search space
2. Algorithm for enumerating query plans
3. Cardinality and cost estimation

Next lecture