CSEP 544
Database Systems

Lecture 8: Overview of
Query Optimization
May 19, 2009

Announcements

- Homework 5 is due next week
  - How is it going?
- Homework 6 (last) to be posted soon
  - Rather short assignment, but start early in case you have questions
- Final will be *take-home*
  - Posted on June 2nd, after last lecture
  - Due by June 4th; electronic turn-in

Where We Are

- We are learning how a DBMS executes a query
- What we learned so far
  - How data is stored and indexed: lecture 6
  - Logical query plans and physical operators: lecture 7
- Today
  - How to select logical & physical query plans

Note: Today's material contains more than Chapter 15 in the textbook !

Query Optimization Goal

- For a query
  - There exists many logical and physical query plans
  - Query optimizer needs to pick a good one

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

Suppliers(sid, sname, scity, sstate)
Supplies(sid, pno, quantity)

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

Physical Query Plan 1

(On the fly) π_{sname}

(On the fly) $\sigma_{scity='Seattle' \wedge sstate='WA' \wedge pno=2}$

(Block-nested loop) $\bowtie_{sid = sid}$

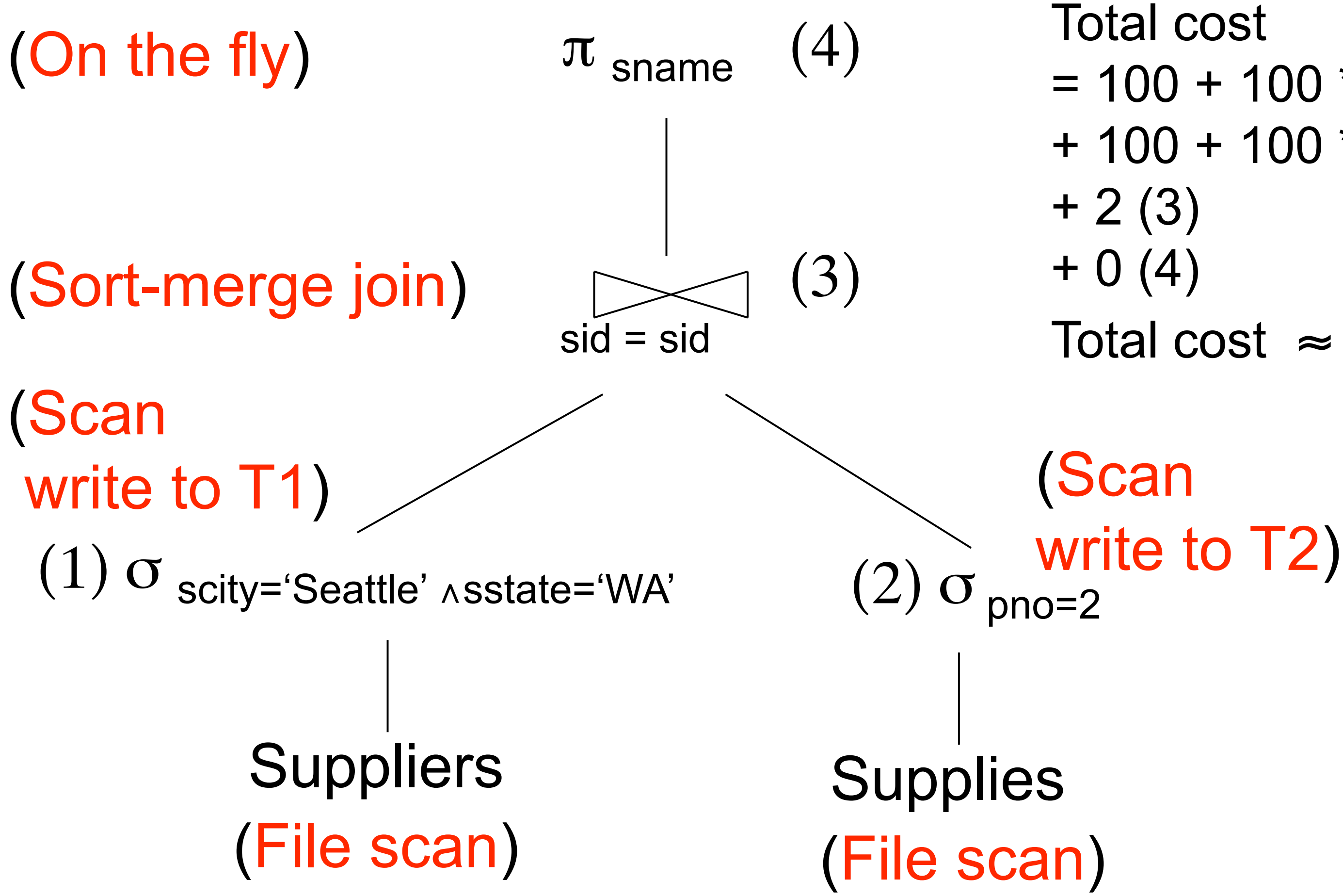
Suppliers (File scan)

Supplies (File scan)

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

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

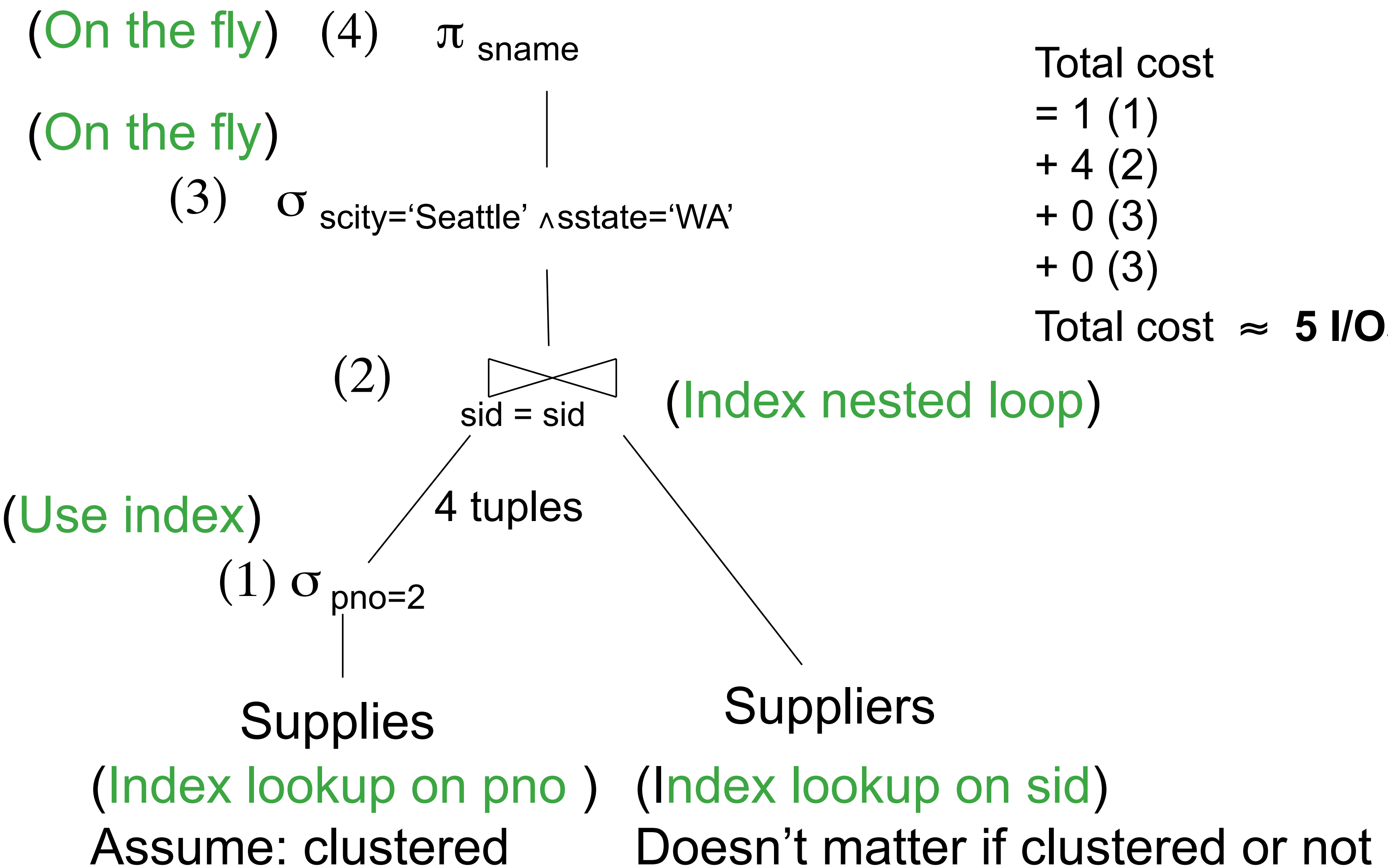
Physical Query Plan 2



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

Total cost ≈ **204 I/Os**

Physical Query Plan 3



Total cost
= 1 (1)
+ 4 (2)
+ 0 (3)
+ 0 (3)

Total cost $\approx$ **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

- Need to consider several physical plan
  - even for one, simple logical plan
- No magic "best" plan: depends on the data
- In order to make the right choice
  - need to have ***statistics*** over the data
  - the B's, the T's, the V's

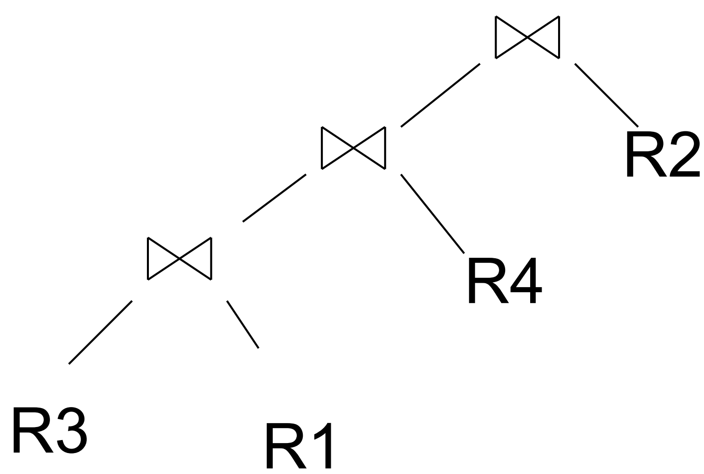
Outline

- Search space
- Algorithm for enumerating query plans
- Estimating the cost of a query plan

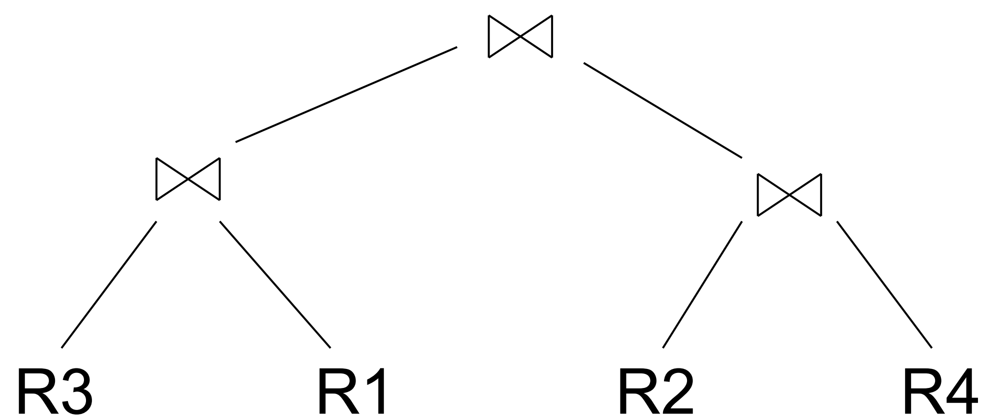
Relational Algebra Equivalences

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

Left-Deep Plans and Bushy Plans



Left-deep plan



Bushy plan

Example: Simple Algebraic Laws

- Commutative and Associative Laws

  $R \cup S = S \cup R, \;\; R \cup (S \cup T) = (R \cup S) \cup T$

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

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

- Distributive Laws

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

Example: Simple Algebraic Laws

- Laws involving selection:

  $\sigma_{C \text{ AND } C'}(R) = \sigma_C(\sigma_{C'}(R)) = \sigma_C(R) \cap \sigma_{C'}(R)$

  $\sigma_{C \text{ OR } C'}(R) = \sigma_C(R) \cup \sigma_{C'}(R)$

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

- When C involves only attributes of R

  $\sigma_C(R - S) = \sigma_C(R) - S$

  $\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$

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

Example: Simple Algebraic Laws

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

Example: Simple Algebraic Laws

- Laws involving projections

  $\Pi_M(R \bowtie S) = \Pi_M(\Pi_P(R) \bowtie \Pi_Q(S))$

  $\Pi_M(\Pi_N(R)) = \Pi_{M,N}(R)$

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

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

Example: Simple Algebraic Laws

- Laws involving grouping and aggregation:

  $\delta(\gamma_{A,\ agg(B)}(R)) = \gamma_{A,\ agg(B)}(R)$

  $\gamma_{A,\ agg(B)}(\delta(R)) = \gamma_{A,\ agg(B)}(R)$ if agg is "duplicate insensitive"

- Which of the following are "duplicate insensitive" ? sum, count, avg, min, max

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

Laws Involving Constratins

Foreign key

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

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

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

Laws with 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)

Laws with Semijoins

Semijoins: a bit of theory (see *Database Theory,* AHV)

- Given a query: $Q :- \Pi (\sigma (R_1 \bowtie R_2 \bowtie . . . \bowtie R_n))$

- A *semijoin reducer* for Q is

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

  such that the query is equivalent to:

$$Q :- \Pi (\sigma (R_{k1} \bowtie R_{k2} \bowtie . . . \bowtie R_{kn}))$$

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

Laws with Semijoins

- Example:

$$Q(A,E) :- \Pi_{A,E}(R_1(A,B) \bowtie R_2(B,C) \bowtie R_3(C,D,E))$$

- A full reducer is:

$$R_2'(B,C) := R_2(B,C) \ltimes R_1(A,B)$$
$$R_3'(C,D,E) := R_3(C,D,E) \ltimes R_2(B,C)$$
$$R_2''(B,C) := R_2'(B,C) \ltimes R_3'(C,D,E)$$
$$R_1'(A,B) := R_1(A,B) \ltimes R_2''(B,C)$$

$$Q(A,E) :- \Pi_{A,E}(R_1'(A,B) \bowtie R_2''(B,C) \bowtie R_3'(C,D,E))$$

The new tables have only the tuples necessary to compute Q(E)

Laws with Semijoins

- Example:

$$Q(E) :- R1(A,B) \ltimes R2(B,C) \ltimes R3(A,C, E)$$

- 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(eid, ename, sal, did)
Dept(did, 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

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

[Chaudhuri'98]

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
```

Search Space Challenges

- Search space is huge!
  - Many possible equivalent trees
  - Many implementations for each operator
  - Many access paths for each relation
    - File scan or index + matching selection condition

- Cannot consider ALL plans
  - Heuristics: only partial plans with “low” cost

Outline

- Search space
- Algorithms for enumerating query plans
- Estimating the cost of a query plan

Key Decisions

- When selecting a plan, some of the most important decisions include:
  - Logical plan
    - Which algebraic laws do we apply, and in which context(s) ?
    - What logical plans do we consider (left-deep, bushy ?)
  - Physical plan
    - What join algorithms to use?
    - What access paths to use (file scan or index)?

Optimizers

- Heuristic-based optimizers:
  - Apply greedily rules that always improve
    - 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

Representation of Partial Plans

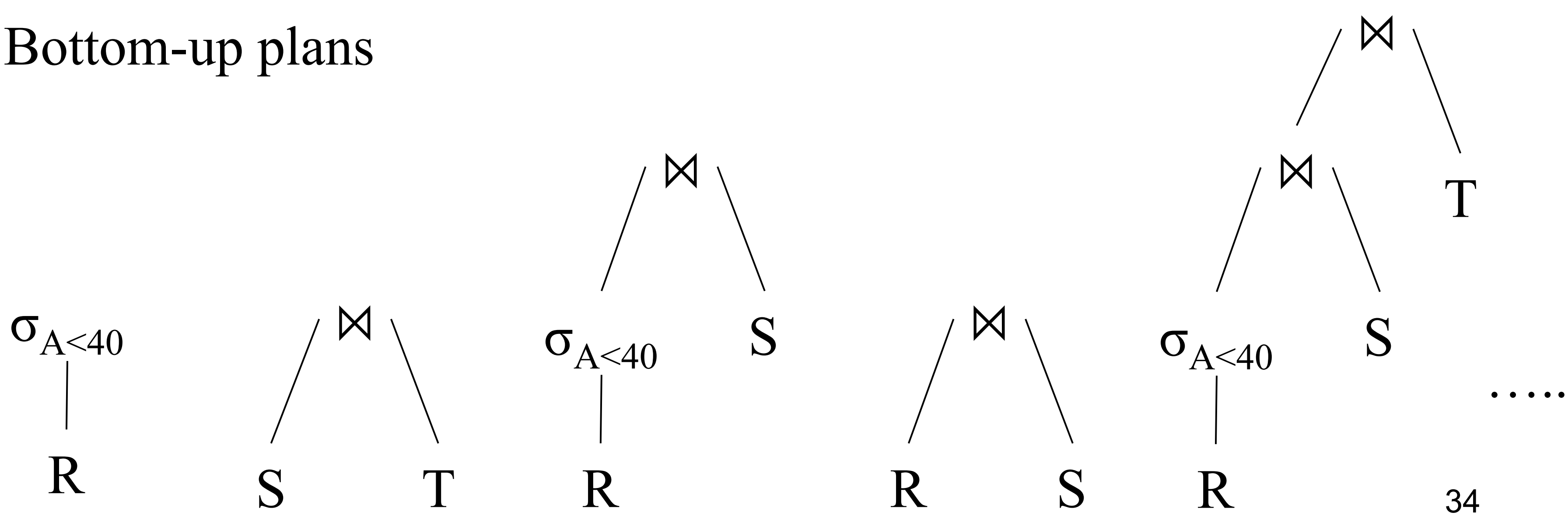
- Bottom-up optimization algorithms:
  - A partial plan is an algebra tree that computes only part of the query
- Top-down optimization algorithms:
  - A partial plan is an algebra tree whose leaves are either base relations, or queries (without a plan yet)

Examples of 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
```

Bottom-up plans

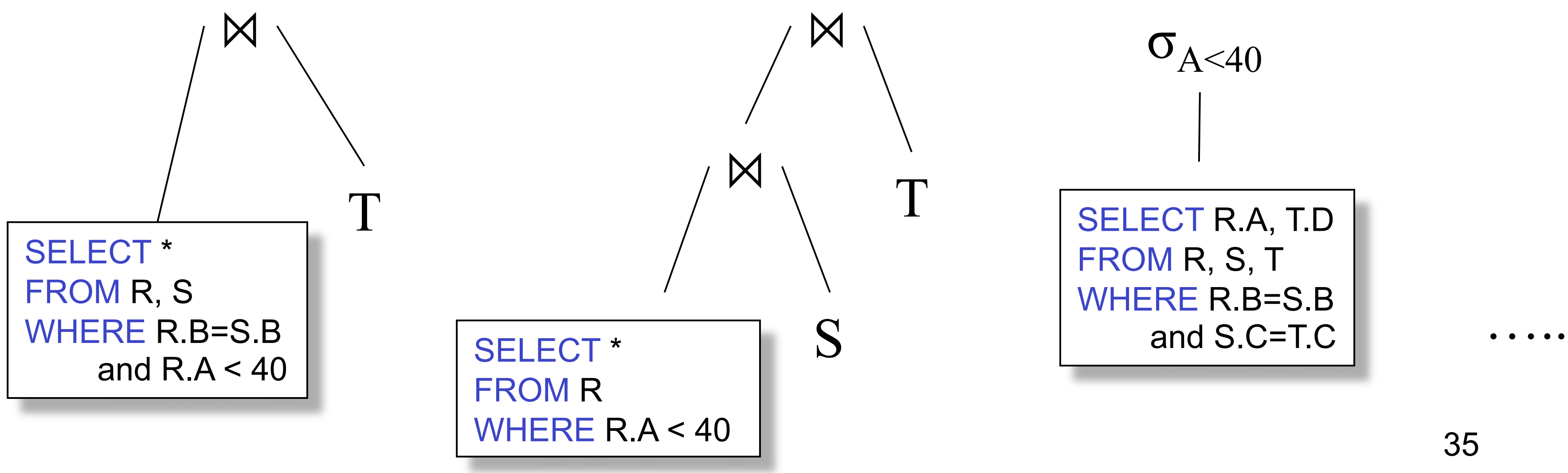


Examples of Partial Plans

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
```

Top-down plans



Plan Enumeration Algorithms

- Dynamic programming
  - Classical algorithm [1979]
  - Limited to joins: *join reordering algorithm*
  - Bottom-up
- Rule-based algorithm
  - Database of rules (=algebraic laws)
  - Usually: dynamic programming
  - Usually: top-down

Dynamic Programming

Originally proposed in System R [1979]

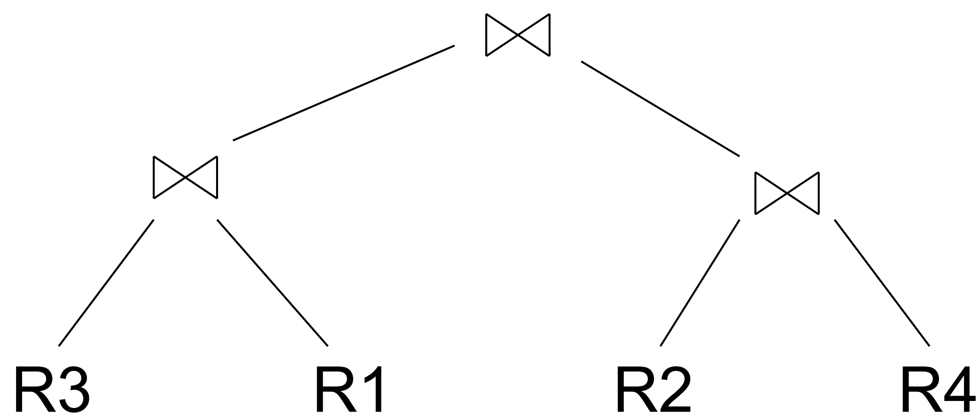
- Only handles single block queries:

```
SELECT list
FROM   R1, …, Rn
WHERE cond_1 AND cond_2 AND . . . AND cond_k
```

- Heuristics: selections down, projections up
- Dynamic programming: ***join reordering***

Join Trees

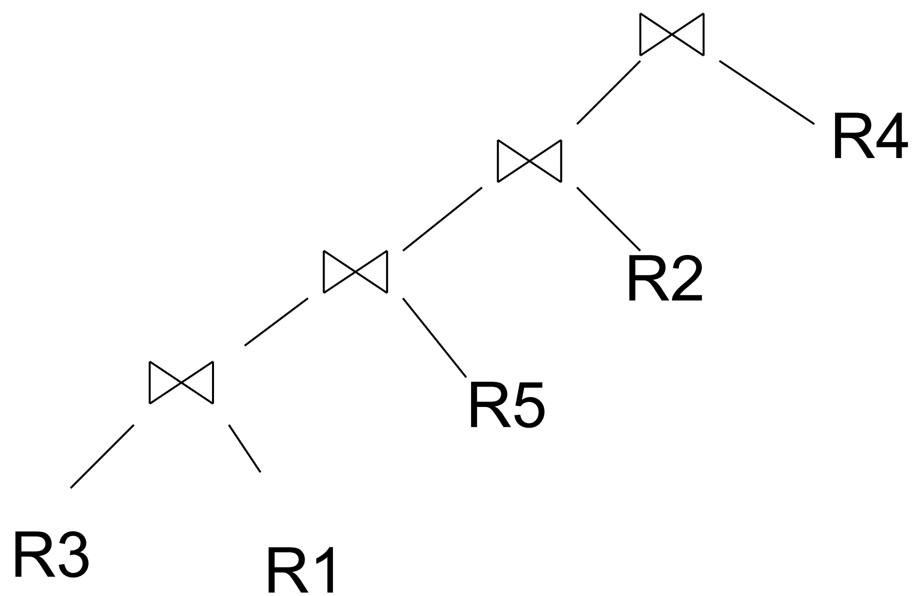
- R1 ⋈ R2 ⋈ …. ⋈ Rn
- Join tree:



- A plan = a join tree
- A partial plan = a subtree of a join tree

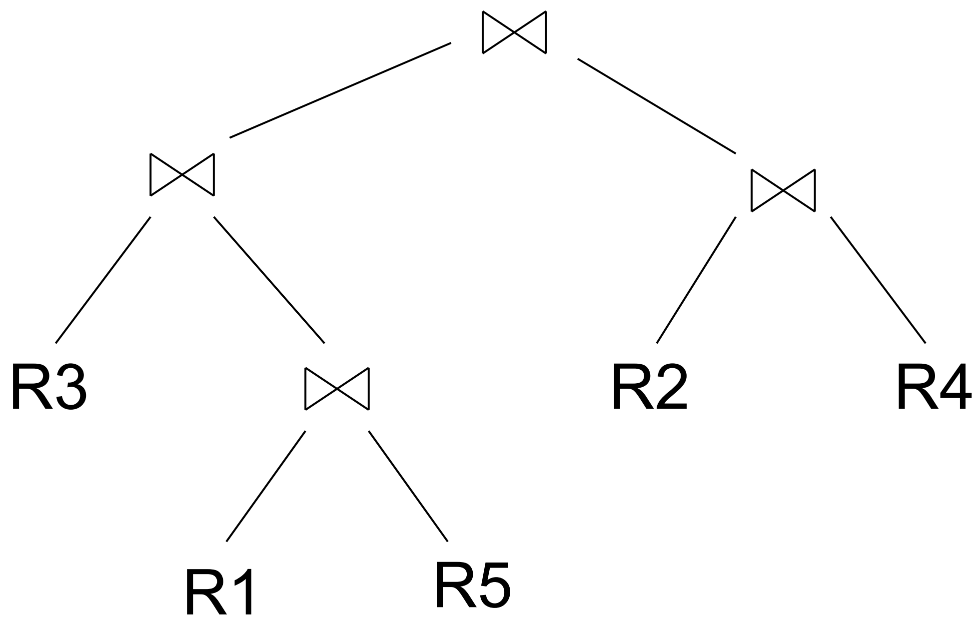
Types of Join Trees

- Left deep:



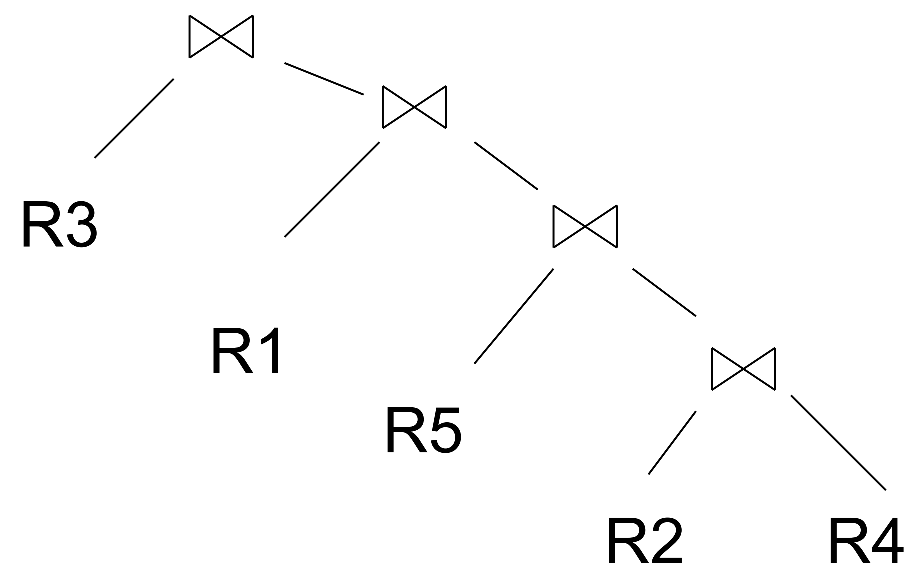
Types of Join Trees

- Bushy:



Types of Join Trees

- Right deep:



Dynamic Programming

Join ordering:

- Given: a query $R1 \bowtie R2 \bowtie \ldots \bowtie Rn$
- Find optimal order

- Assume we have a function cost() that gives us the cost of every join tree

Dynamic Programming

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

Dynamic Programming

- **Step 1**: For each $\{R_i\}$ do:
  - $Size(\{R_i\}) = B(R_i)$
  - $Plan(\{R_i\}) = R_i$
  - $Cost(\{R_i\})$ = (cost of scanning R_i)

Dynamic Programming

- **Step 2**: For each Q $\subseteq\{R_1, \ldots, R_n\}$ of cardinality i do:
  - Size(Q) = estimate it recursively
  - For every pair of subqueries Q', Q'' s.t. Q = Q' $\cup$ Q''
    compute cost(Plan(Q') $\bowtie$ Plan(Q''))
    - Cost(Q) = the smallest such cost
    - Plan(Q) = the corresponding plan

Dynamic Programming

- **Step 3**: Return Plan({R_1, …, R_n})

Example

To illustrate, we will make the following simplifications:

- $Cost(P_1 \bowtie P_2) = Cost(P_1) + Cost(P_2) +$ size(intermediate result(s))
  - Size(intermediate result(s)) =
    If P_1 = a join, then the size of the intermediate result is size(P_1), otherwise the size is 0
    Similarly for P_2
- Cost of a scan = 0

Example

- $R \bowtie S \bowtie T \bowtie U$
- Number of tuples: 2000, 5000, 3000, 1000
- Size estimation: $T(A \bowtie B) = 0.01*T(A)*T(B)$

Subquery	Size	Cost	Plan
RS			
RT			
RU			
ST			
SU			
TU			
RST			
RSU			
RTU			
STU			
RSTU			

Subquery	Size	Cost	Plan
RS	100k	0	RS
RT	60k	0	RT
RU	20k	0	RU
ST	150k	0	ST
SU	50k	0	SU
TU	30k	0	TU
RST	3M	60k	(RT)S
RSU	1M	20k	(RU)S
RTU	0.6M	20k	(RU)T
STU	1.5M	30k	(TU)S
RSTU	30M	60k+50k=110k	(RT)(SU)

Reducing the Search Space

- Left-linear trees v.s. Bushy trees
- Trees without cartesian product

Example: 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

Dynamic Programming: Summary

- Handles only join queries:
  - Selections are pushed down (i.e. early)
  - Projections are pulled up (i.e. late)

- Takes exponential time in general, BUT:
  - Left linear joins may reduce time
  - Non-cartesian products may reduce time further

Rule-Based Optimizers

- ***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

- Volcano (later SQL Sever)
- Starburst (later DB2)

Completing the Physical Query Plan

- Choose algorithm for each operator
  - How much memory do we have ?
  - Are the input operand(s) sorted ?
- Access path selection for base tables
- Decide for each intermediate result:
  - To materialize
  - To pipeline

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

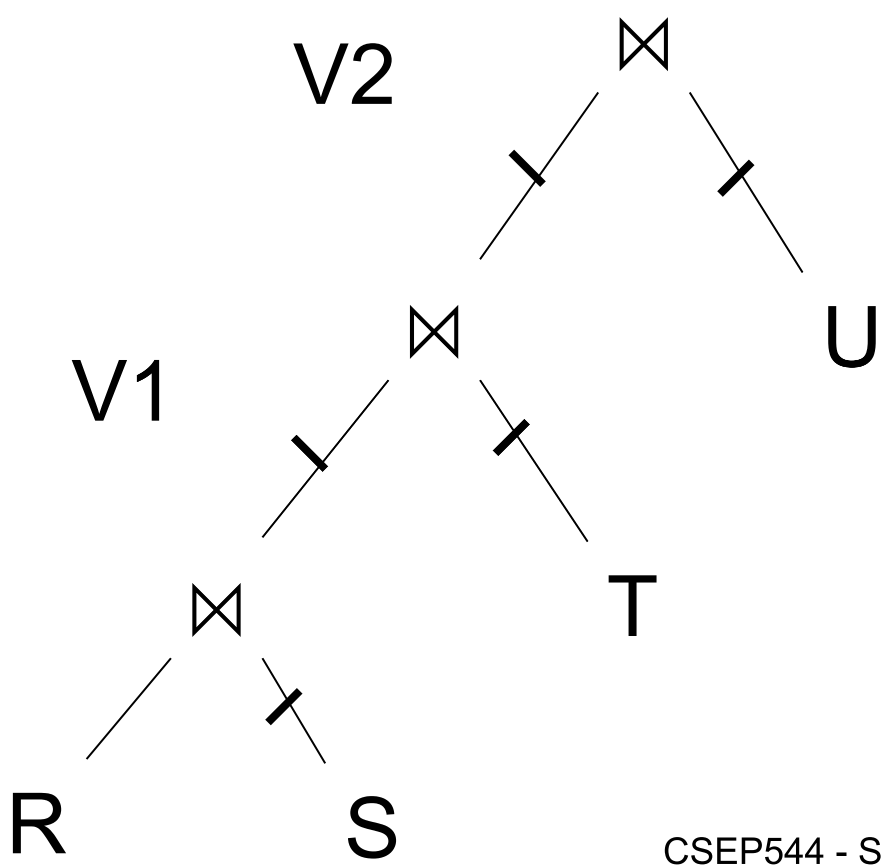
- Supplier(sid,sname,scity,sstate)
- Selection condition: sid > 300 ∧ scity='Seattle'
- Indexes: B+-tree on sid and B+-tree on scity
- Which access path should we use?
- We should pick the **most selective** access path

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)

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)
```

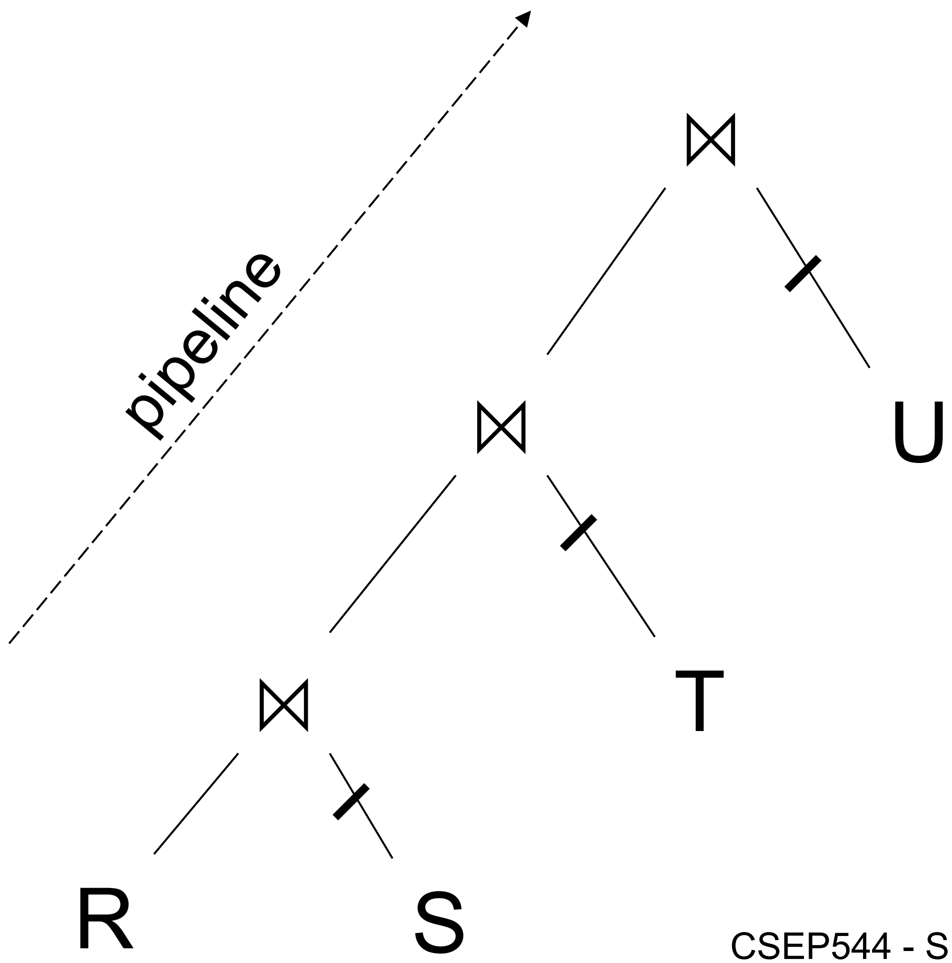
Materialize Intermediate Results Between Operators

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)
```

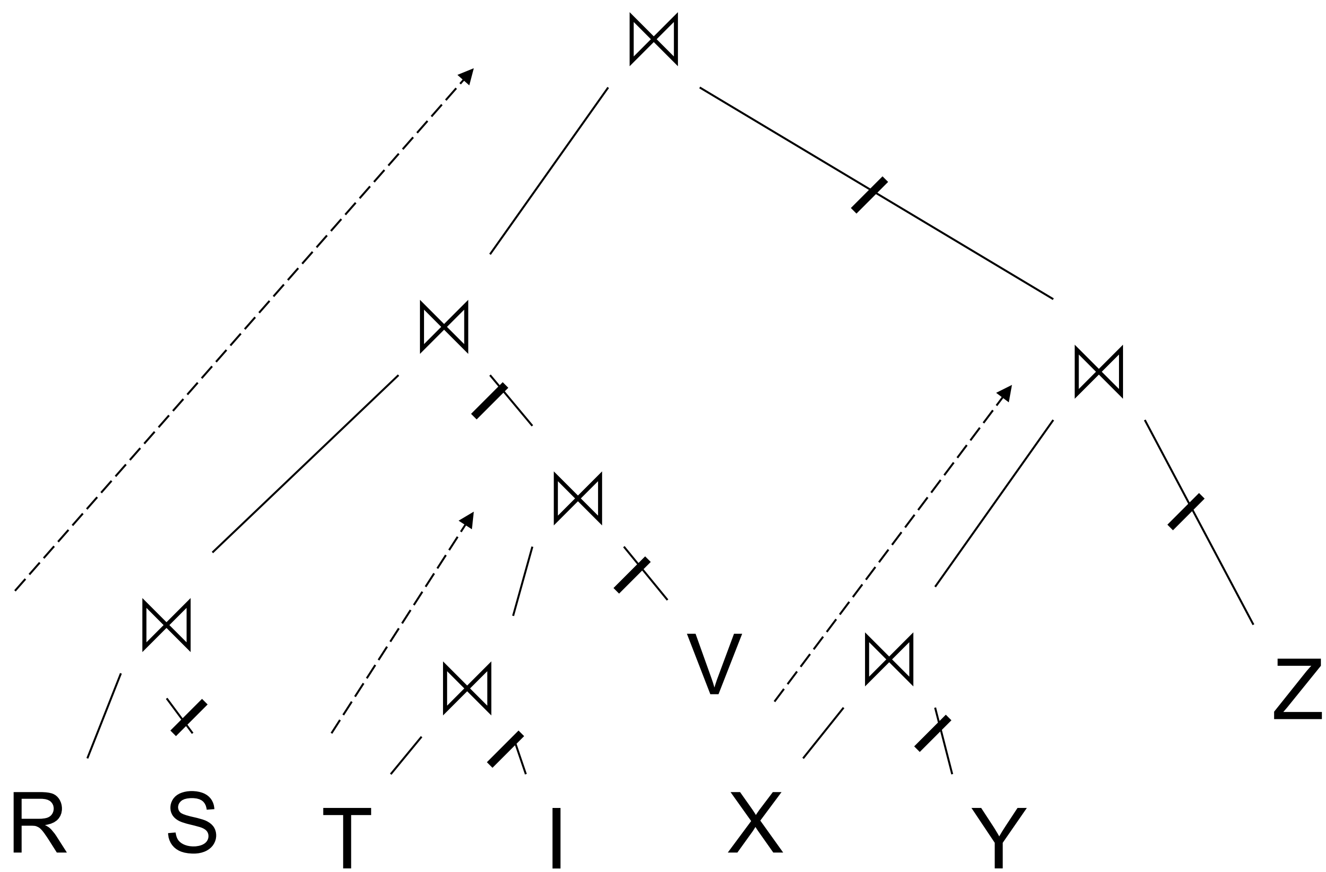
Pipeline Between Operators

Question in class

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

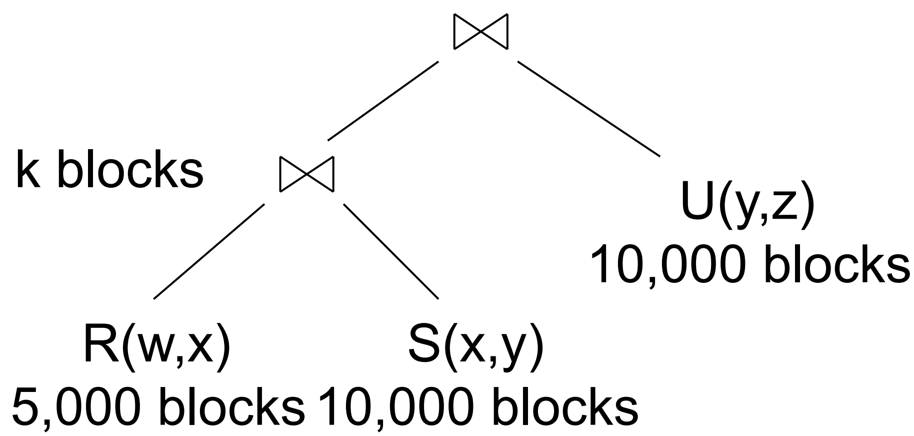
- What is the total cost of the plan ?
  - Cost =
- How much main memory do we need ?
  - M =

Pipeline in Bushy Trees



Example

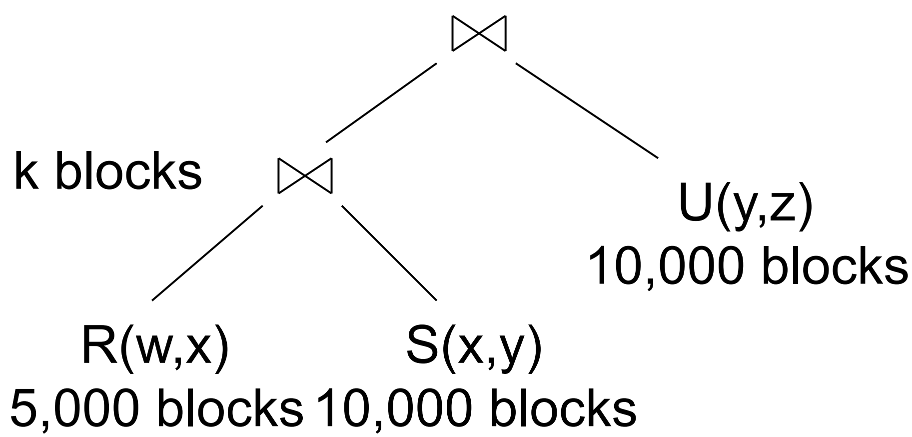
- Logical plan is:



- Main memory M = 101 buffers

Example

M = 101

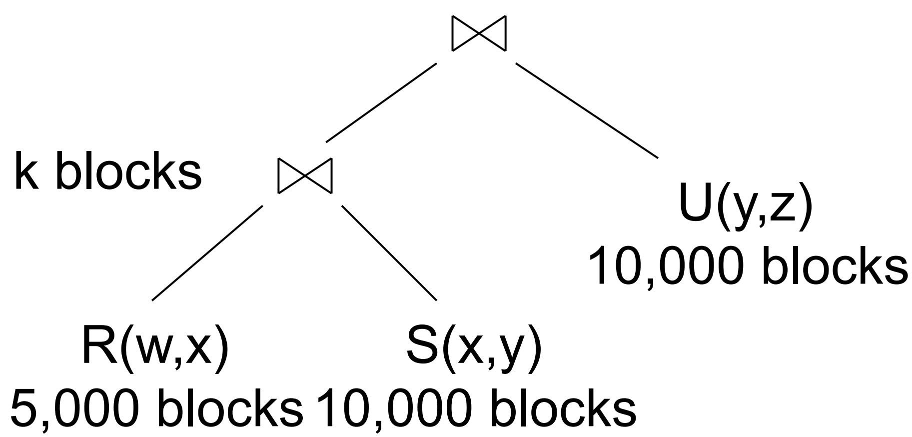


Naïve evaluation:

- 2 partitioned hash-joins
- Cost 3B(R) + 3B(S) + 4k + 3B(U) = 75000 + 4k

Example

M = 101

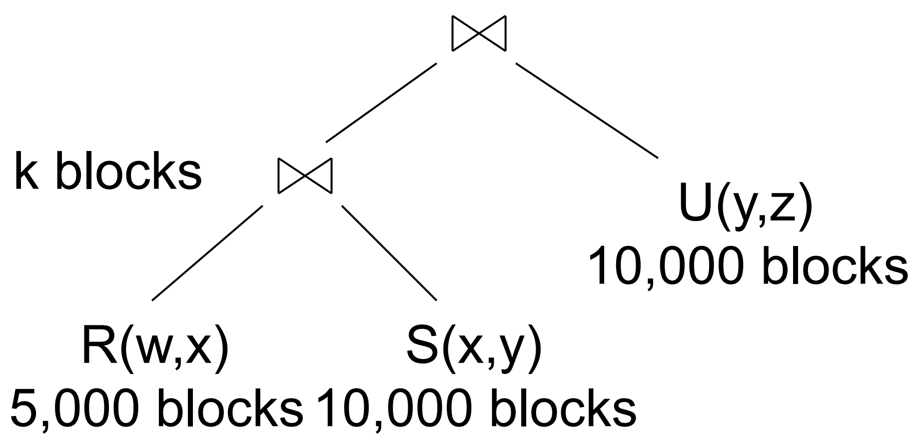


Smarter:

- Step 1: hash R on x into 100 buckets, each of 50 blocks; to disk
- Step 2: hash S on x into 100 buckets; to disk
- Step 3: read each R_i in memory (50 buffer) join with S_i (1 buffer); hash result on y into 50 buckets (50 buffers) -- here we *pipeline*
- Cost so far: 3B(R) + 3B(S)

Example

M = 101

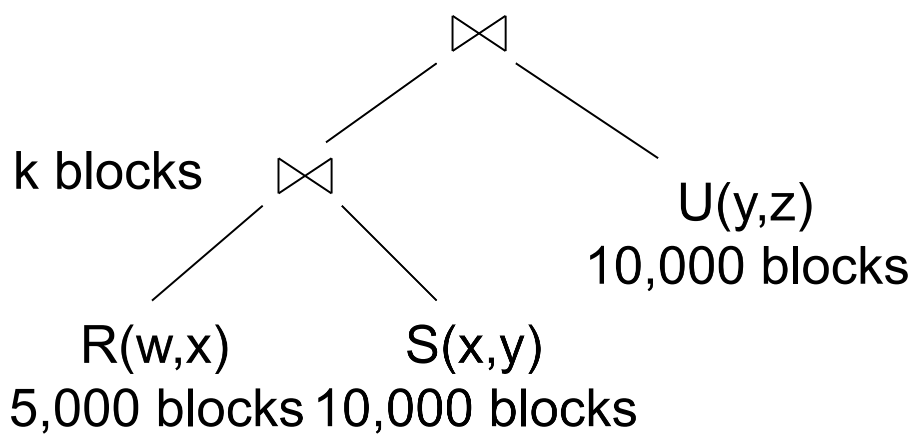


Continuing:

- How large are the 50 buckets on y ? Answer: k/50.
- If k <= 50 then keep all 50 buckets in Step 3 in memory, then:
- Step 4: read U from disk, hash on y and join with memory
- Total cost: 3B(R) + 3B(S) + B(U) = 55,000

Example

M = 101

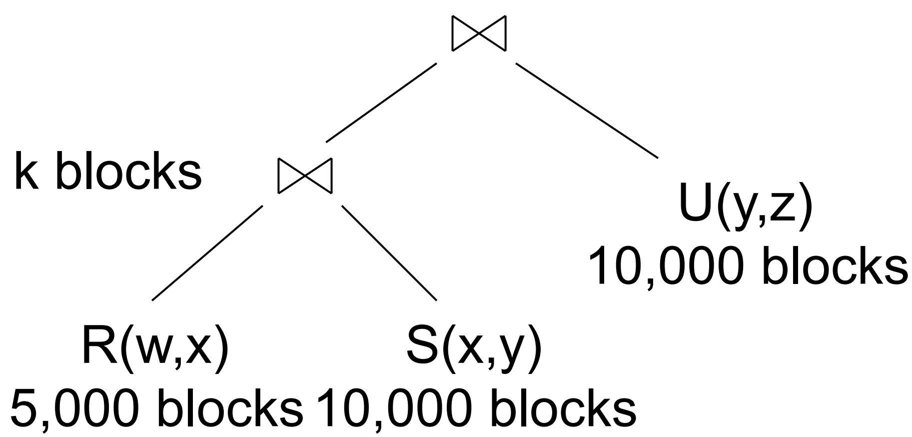


Continuing:

- If 50 < k <= 5000 then send the 50 buckets in Step 3 to disk
  - Each bucket has size k/50 <= 100
- Step 4: partition U into 50 buckets
- Step 5: read each partition and join in memory
- Total cost: 3B(R) + 3B(S) + 2k + 3B(U) = 75,000 + 2k

Example

M = 101



Continuing:

- If k > 5000 then materialize instead of pipeline
- 2 partitioned hash-joins
- Cost 3B(R) + 3B(S) + 4k + 3B(U) = 75000 + 4k

Outline

- Search space
- Algorithms for enumerating query plans
- Estimating the cost of a query plan

Computing the Cost of a Plan

- Collect statistical summaries of stored data
- Estimate size in a bottom-up fashion
- Estimate cost by using the estimated size

Statistics on Base Data

- Collected information for each relation
  - Number of tuples (cardinality)
  - Indexes, number of keys in the index
  - Number of physical pages, clustering info
  - Statistical information on attributes
    - Min value, max value, number distinct values
    - Histograms
  - Correlations between columns (hard)
- Collection approach: periodic, using sampling

Size Estimation

Estimating the size of a projection

- Easy: $T(\Pi_L(R)) = T(R)$
- This is because a projection doesn't eliminate duplicates

Size Estimation for Selection

Estimating the size of a selection

- $S = \sigma_{A=c}(R)$
  - T(S) can be anything from 0 to T(R) – V(R,A) + 1
  - Estimate: T(S) = T(R)/V(R,A)
  - When V(R,A) is not available, estimate T(S) = T(R)/10

- $S = \sigma_{A<c}(R)$
  - T(S) can be anything from 0 to T(R)
  - Estimate: T(S) = (c - Low(R, A))/(High(R,A) - Low(R,A))T(R)
  - When Low, High unavailable, estimate T(S) = T(R)/3

Size Estimation for Selection

What if we have an index on multiple attributes?

- Example selection $S=\sigma_{a=v1 \wedge b=v2}(R)$

How to compute the selectivity?

- Assume attributes are independent
- T(S) = T(R) / (V(R,a) * V(R,b))

Example

- Selection condition: **sid > 300 ∧ scity='Seattle'**
  - Index I1: B+-tree on sid clustered
  - Index I2: B+-tree on scity unclustered

- Let's assume
  - V(Supplier,scity) = 20
  - Max(Supplier, sid) = 1000, Min(Supplier,sid)=1
  - B(Supplier) = 100, T(Supplier) = 1000

- **Cost I1: B(R) * (Max-v)/(Max-Min) = 100*700/999 ≈ 70**
- **Cost I2: T(R) * 1/V(Supplier,scity) = 1000/20 = 50**

Size Estimation for Join

Estimating the size of a natural join, $R \bowtie_A S$

- When the set of A values are disjoint, then $T(R \bowtie_A S) = 0$
- When A is a key in S and a foreign key in R, then $T(R \bowtie_A S) = T(R)$
- When A has a unique value, the same in R and S, then $T(R \bowtie_A S) = T(R)\ T(S)$

Estimation seems hopelessly hard !

Size Estimation for Join

Assumptions:

- *Containment of values*: if V(R,A) <= V(S,A), then the set of A values of R is included in the set of A values of S
  - Note: this indeed holds when A is a foreign key in R, and a key in S
- *Preservation of values*: for any other attribute B, $V(R \bowtie_A S, B) = V(R, B)$ (or $V(S, B)$)

Size Estimation for Join

Assume V(R,A) <= V(S,A)

- Then each tuple t in R joins *some* tuple(s) in S
  - How many ?
  - On average T(S)/V(S,A)
  - t will contribute T(S)/V(S,A) tuples in $R \bowtie_A S$
- Hence $T(R \bowtie_A S) = T(R)\ T(S) / V(S,A)$

In general: $T(R \bowtie_A S) = T(R)\ T(S) / \max(V(R,A),V(S,A))$

Size Estimation for Join

Example:

- $T(R) = 10000$, $T(S) = 20000$
- $V(R,A) = 100$, $V(S,A) = 200$
- How large is $R \bowtie_A S$?

Answer: $T(R \bowtie_A S) = 10000\ 20000/200 = 1M$

Size Estimation for Join

Joins on more than one attribute:

- $T(R \bowtie_{A,B} S) =$

  $T(R)\ T(S)/(\max(V(R,A),V(S,A))*\max(V(R,B),V(S,B)))$

Computing Cost of an Operator

- The cost of executing an operator depends
  - On the operator implementation
  - On the input data

- We learned how to compute this in the previous lecture, so we do not repeat it here

Histograms

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

Histograms

Employee(ssn, name, salary, phone)

- Maintain a histogram on salary:

Salary:	0..20k	20k..40k	40k..60k	60k..80k	80k..100k	> 100k
Tuples	200	800	5000	12000	6500	500

- T(Employee) = 25000, but now we know the distribution

Histograms

Employee(ssn, name, salary, phone)

- Eqwidth

Salary	0..20	20..40	40..60	60..80	80..100
Tuples	2	104	9739	152	3

- Eqdepth

Salary	0..44	44..48	48..50	50..56	55..100
Tuples	1800	2000	2100	2200	1900

Example

Employee(ssn, name, salary, phone)

Salary	0..44	44..48	48..50	50..56	55..100
Tuples	1800	2000	2100	2200	1900

Estimate the size of: $S = \sigma_{\text{salary>=46 and salary<=70}}(\text{Employee})$

Example

Employee(ssn, name, salary, phone)

Salary	0..44	44..48	48..50	50..56	55..100
Tuples	1800	2000	2100	2200	1900

Estimate the size of: $S = \sigma_{salary>=46 \text{ and } salary<=70}(Employee)$

Answer: T(S) = 2000*3/4 + 2100 + 2200 + 1900*16/46

Summary of Query Optimization

- Three parts:
  - search space, algorithms, size/cost estimation
- This lecture discussed some of the issues
  - Lecture has more material than either textbook, however:
  - You won't be able to write an optimizer tomorrow !
  - There is no good text on rule-based optimizer