Query Optimization: Cost-Based Optimization

Abdu Alawini

University of Illinois at Urbana-Champaign

CS411: Database Systems

Leaning Objectives

After this lecture, you should be able to:

- Estimate the cost of a join
- Use the dynamic programming algorithms to determine the "best" join tree
- Estimate the size and cost of a query plan

Outline

- Introduction to Cost-Based Optimization
- Dynamic Programming Algorithm for Determining the Best Join Tree
- Completing the Physical Query Plan

Cost-based Optimizations

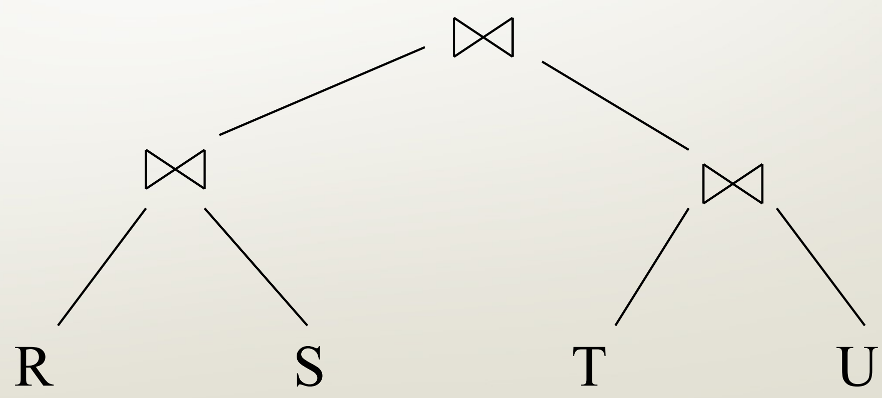
- Main idea: given a "partial query": apply algebraic laws, until estimated cost of partial plan is minimal

- Start from partial plans, build more complete plans
  - Will see in a few slides

- Problem: there are too many ways to apply the laws, hence too many partial plans

Partial Plans Focus: Join Trees

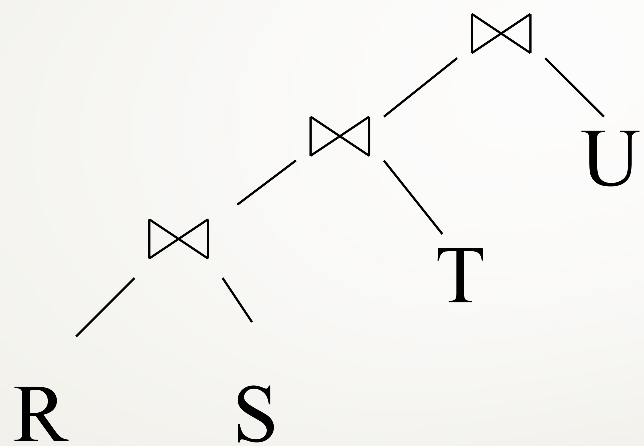
- $R1 \bowtie R2 \bowtie .... \bowtie Rn$
- Join includes: Njoin, Theta-join, cross product
- Why focus on joins?
  - Super common, very basic operation
  - Joins are costly, selections/projections can be applied inlined with other operations (e.g., scan), other operators are usually applied once at the top
- Join tree:



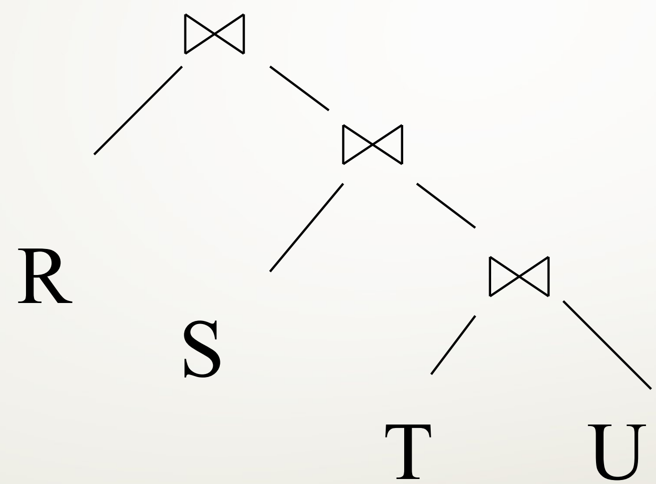
- A plan = a join tree.
- Lots of these !
- A partial plan = a subtree of a join tree

16.6

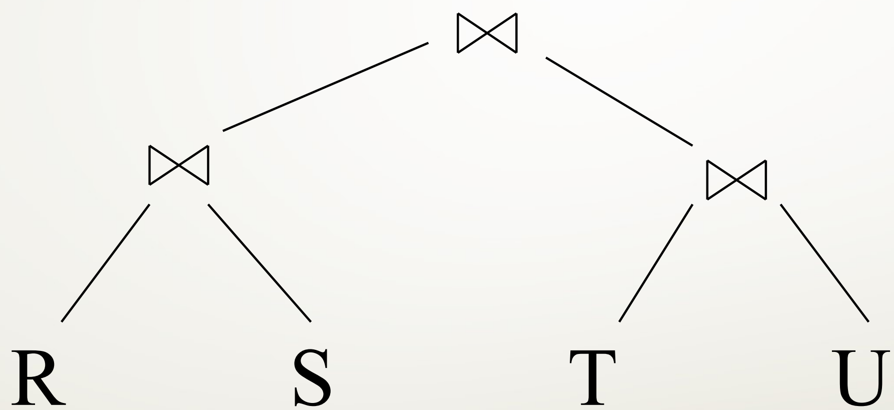
Types of Join Trees:
Left deep



Types of Join Trees: Right deep



Types of Join Trees: Bushy



Problem

- Given: a query $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$
- Assume we have a function cost() that gives us the cost of every join tree
- Find the best join tree for the query

Outline

✓ Introduction to Cost-Based Optimization
- Dynamic Programming Algorithm for Determining the Best Join Tree
- Completing the Physical Query Plan

Dynamic Programming

- Idea: 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

Dynamic Programming

- For each subset $Q \subseteq \{R1, \ldots, Rn\}$ compute
  - Size(Q): estimated size of the join of the relations in Q
  - Plan(Q): a best plan for Q – a particular join tree.
  - Cost(Q): the **least** cost of that plan: the cost is going to be the size of **intermediate tables** used by the plan.
- *Assumptions:*
  - Final table, initial Ri's are both ignored (common to all plans)
  - We'll restrict ourselves to left-deep plans (plans are just reordering of relations)
  - Recall that size of tables (B(R) + B(S)) is a lower bound on join cost.
    - the least cost of a specific join is at least equal to sum of the sizes of the two relations.

Dynamic Programming

- **Step 1**: For each {Ri} do:
  - Size({Ri}) = B(Ri)
  - Plan({Ri}) = Ri
  - Cost({Ri}) = 0. (remember: cost of intermediate tables)

- **Step 2**: For each {Ri, Rj} do:
  - Size({Ri,Rj}) = estimate of size of join $\boxed{T(R \bowtie_A S) = T(R)\,T(S) / \max(V(R,A), V(S,A))}$
  - Plan({Ri,Rj}) = either Ri $\bowtie$ Rj, or Rj $\bowtie$ Ri
    - Smaller table on the left (convention + other reasons)
  - Cost = 0. (no intermediate tables used)

Dynamic Programming

- **Step i**: For each $Q \subseteq \{R_1, \ldots, R_n\}$ of cardinality i do:
  - Compute Size(Q) (as we've seen in size estimation earlier)
  - For every pair of subsets Q', Q''
    s.t. Q = Q' U Q''
    compute cost(Q') + cost(Q'') + size(Q') + size(Q'')
    - If Q' or Q'' is a single table, then don't count its size
    - Cost(Q) = the smallest such sum
    - Plan(Q) = the corresponding plan

- **In the end,** Return Plan($\{R_1, \ldots, R_n\}$)

Dynamic Programming

- Example:
- $Cost(R5 \bowtie R7) = 0$ (no intermediate results)
- $Cost((R2 \bowtie R1) \bowtie R7)$
  $= Cost(R2 \bowtie R1) + Cost(R7) + size(R2 \bowtie R1)$
  $= size(R2 \bowtie R1)$
- $Cost\ (((R2 \bowtie R1) \bowtie R7) \bowtie R8)$
  $= Cost\ ((R2 \bowtie R1) \bowtie R7) + Cost\ (R8) + Size((R2 \bowtie R1) \bowtie R7)$
  $= Size\ (R2 \bowtie R1) + 0 + Size((R2 \bowtie R1) \bowtie R7)$

Dynamic Programming

- Relations: R, S, T, U
- Number of tuples: 2000, 5000, 3000, 1000
- Estimated sizes of values sets for the attributes in each relation:

R(a, b)	**S(b, c)**	**T(c, d)**	**U(d, a)**
V(R, a) = 100			V(U, a) = 50
V(R, b) = 200	V(S, b) = 100		
	V(S, c) = 500	V(T, c) = 20	
		V(T, d) = 50	V(U, d) = 1000

R(a, b)	S(b, c)	T(c, d)	U(d, a)
V(R, a) = 100			V(U, a) = 50
V(R, b) = 200	V(S, b) = 100		
	V(S, c) = 500	V(T, c) = 20	
		V(T, d) = 50	V(U, d) = 1K

Table	Size
R	2K
S	5K
T	3K
U	1K

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

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

R(a, b)	S(b, c)	T(c, d)	U(d, a)
V(R, a) = 100			V(U, a) = 50
V(R, b) = 200	V(S, b) = 100		
	V(S, c) = 500	V(T, c) = 20	
		V(T, d) = 50	V(U, d) = 1K

Table	Size
R	2K
S	5K
T	3K
U	1K

Subquery	Size	Cost	Plan
RS	50k	0	RS
RT	6M	0	RT
RU	20k	0	UR
ST	30k	0	TS
SU	5M	0	US
TU	3k	0	UT
RST			
RSU			
RTU			
STU			
RSTU			

R(a, b)	S(b, c)	T(c, d)	U(d, a)
V(R, a) = 100			V(U, a) = 50
V(R, b) = 200	V(S, b) = 100		
	V(S, c) = 500	V(T, c) = 20	
		V(T, d) = 50	V(U, d) = 1K

Table	Size
R	2K
S	5K
T	3K
U	1K

Subquery	Size	Cost	Plan
RS	50k	0	RS
RT	6M	0	RT
RU	20k	0	UR
ST	30k	0	TS
SU	5M	0	US
TU	3k	0	UT
RST	300K	30K	(TS)R
RSU	500K	20K	(UR)S
RTU	60K	3K	(UT)R
STU	30K	3K	(UT)S
RSTU			

R(a, b)	S(b, c)	T(c, d)	U(d, a)
V(R, a) = 100			V(U, a) = 50
V(R, b) = 200	V(S, b) = 100		
	V(S, c) = 500	V(T, c) = 20	
		V(T, d) = 50	V(U, d) = 1K

Table	Size
R	2K
S	5K
T	3K
U	1K

Subquery	Size	Cost	Plan
RS	50k	0	RS
RT	6M	0	RT
RU	20k	0	UR
ST	30k	0	TS
SU	5M	0	US
TU	3k	0	UT
RST	300K	30K	(TS)R
RSU	500K	20K	(UR)S
RTU	60K	3K	(UT)R
STU	30K	3K	(UT)S
RSTU	3K	33K	(UTS)R

Dynamic Programming

- Summary: computes optimal plans for subqueries:

Step 1: {R1}, {R2}, …, {Rn}
Step 2: {R1, R2}, {R1, R3}, …, {Rn-1, Rn}
…
Step n: {R1, …, Rn}

- In practice:
  - heuristics for Reducing the Search Space
    - Restrict to left linear / deep trees
    - Restrict to trees “without cartesian product”: R(A,B), S(B,C), T(C,D)
      (R join T) join S has a cartesian product

Outline

- ✓ Introduction to Cost-Based Optimization
- ✓ Dynamic Programming Algorithm for Determining the Best Join Tree
- Completing the Physical Query Plan

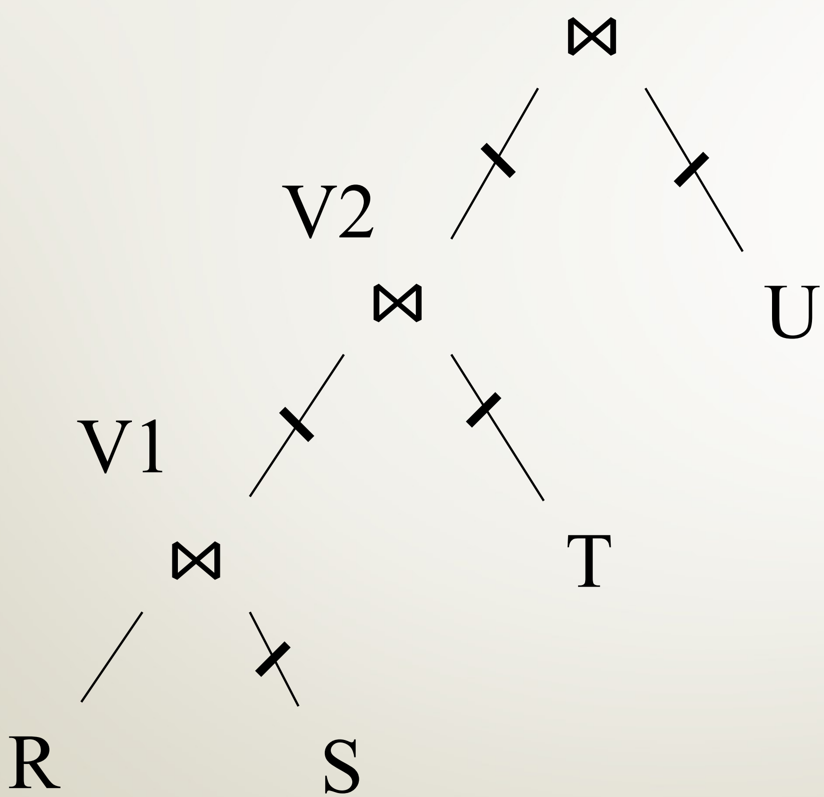
Completing the Physical Query Plan

- Choose algorithm to implement each operator
  - Need to account for more than cost:
    - How much memory do we have ?
    - Are the input operand(s) sorted ?
- Decide for each intermediate result:
  - To materialize: create entirely and store on disk
  - To pipeline: create in parts and move on to next operation; entire result may never be available at the same time, not stored on disk.

We'll now switch back to talking about actual costs for each operator as opposed to costs of intermediates

Materialize Intermediate Results Between Operators

16.7.3



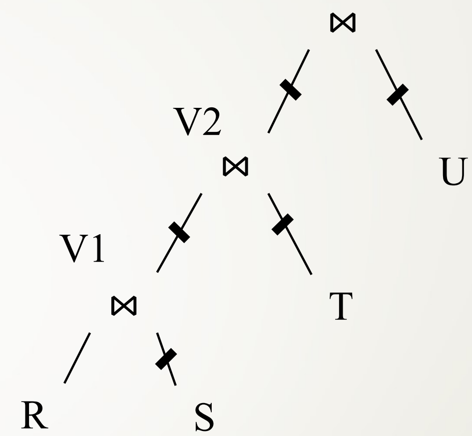

```
HashTable ← S
repeat    read(R, x)
          y ← join(HashTable, x)
          write(V1, y)

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)
```

What is the actual total cost of this physical plan?

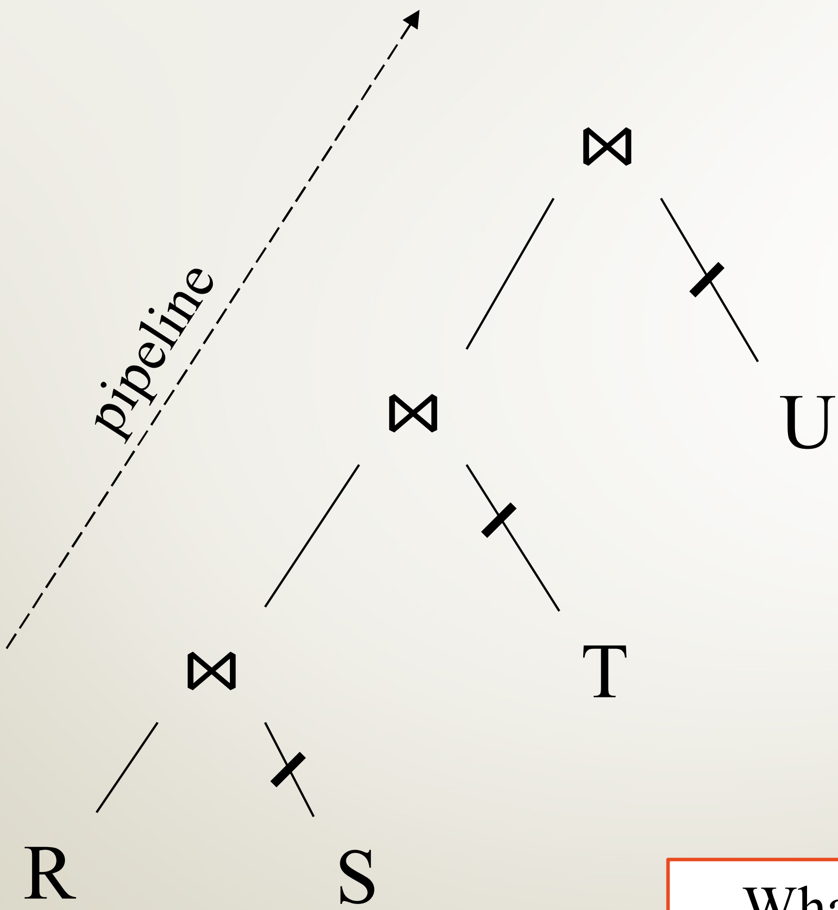
Materialize Intermediate Results Between Operators



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

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

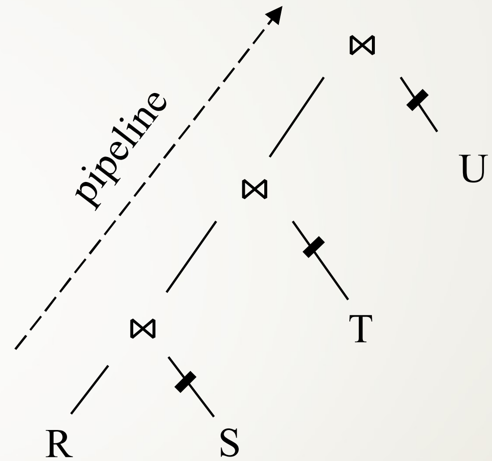
Pipeline Between Operators



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)

What is the actual total cost of this plan?

Pipeline Between Operators



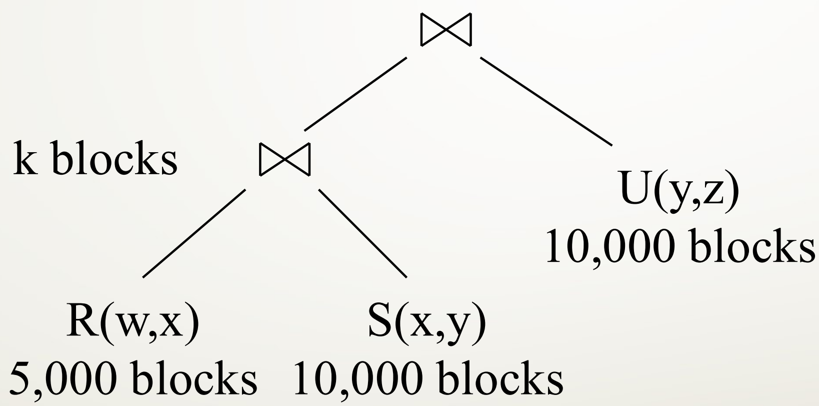
Given B(R), B(S), B(T), B(U)

- What is the actual total cost of the plan ?
  - Actual Cost = B(R) + B(S) + B(T) + B(U)
  - Cost of Intermediates is Zero
- How much main memory do we need ?
  - M = B(S) + B(T) + B(U)

Compare with materialization: more main memory but less actual cost.

Example 1

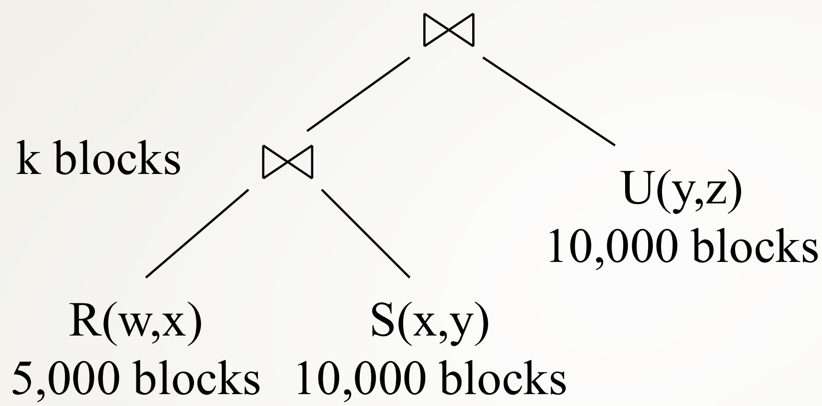
- Logical plan is:



- Main memory M = 101 buffers

Main memory
M = 101 buffers

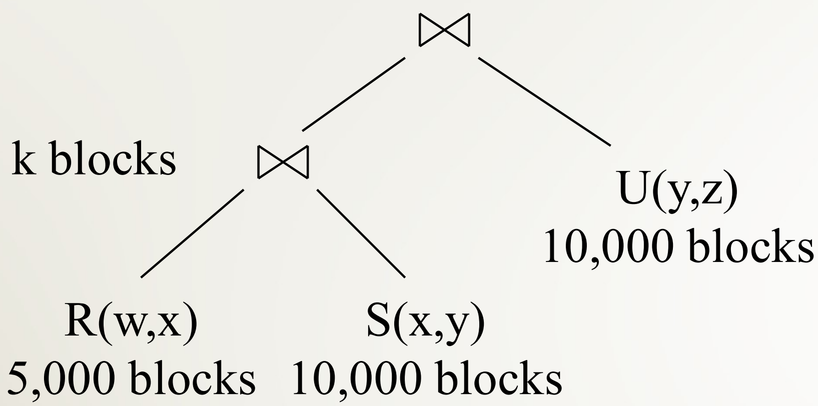
Example 1: Naïve evaluation



- 2 partitioned (2-pass) hash-joins. Materialize after computing first join (R ⋈ S).
- Cost 3B(R) + 3B(S) + k + 3k + 3B(U) = 75000 + 4k
- Memory requirement: min(B(R), B(S)) <= (M-1)(M-2) for first join and min(k, B(U)) <= (M-1)(M-2) for second join.

Main memory
M = 101 buffers

Example 1: Smarter plan



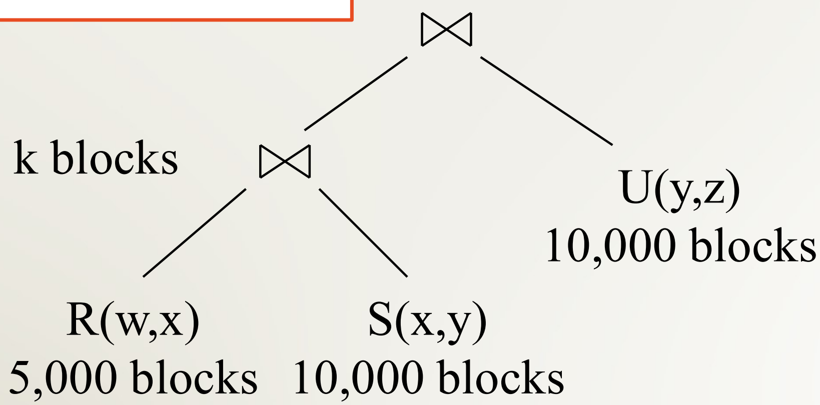
Same first step:

- Step 1: hash R on x into 100 buckets, each of 50 blocks; to disk
- Step 2: hash S on x into 100 buckets, each of 100 blocks; to disk
- Step 3: read each bucket Ri in memory (50 buffers), join with Si (1 buffer); use 50 remaining buckets to store the result (hashed on y)

- Cost so far: 3B(R) + 3B(S)

Main memory
M = 101 buffers

Example 1: K<=50



Case 1: If k <= 50 then keep all 50 buckets in Step 3 in memory, then:

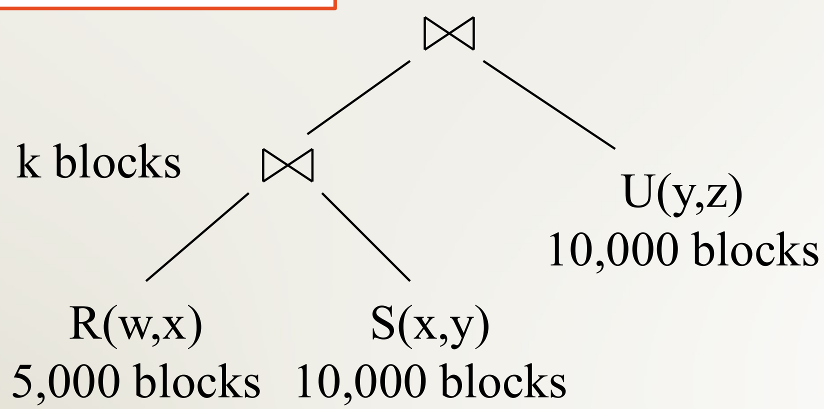
- Step 4: read U from disk (one block at a time), hash on y and join with memory
- Total cost: 3B(R) + 3B(S) + B(U) = 55,000

What if k > 50? Can we do this?

Only 50 blocks for the results of the join, so intermediate result can't be stored in memory

Main memory
M = 101 buffers

Example 1: 50 < k <= 4950



Case 2:

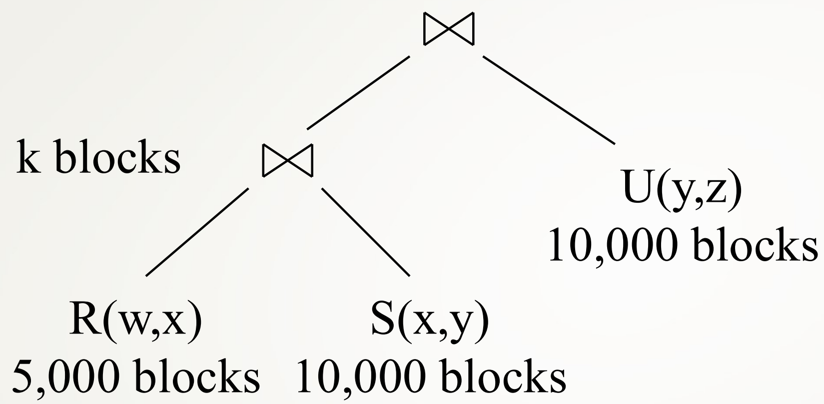
- If 50 < k <= 4950 then send the 50 buckets (hashed on y) in Step 3 to disk
  - Each bucket has size k/50 <= 99
- Step 4: partition U into 50 buckets
- Step 5: read each bucket of (R $\bowtie$ S) into memory, read corresponding bucket of U (block by block) and join in memory
- Total cost: 3B(R) + 3B(S) + 2k + 3B(U) = 75,000 + 2k

Why did we partition U into 50 buckets in Step 4?

Why do we have the <=4950 condition?

Main memory
M = 101 buffers

Example 1: k > 4950



Continuing:

- If k > 4950 then materialize instead of pipeline
- 2 partitioned hash-joins
- Cost 3B(R) + 3B(S) + 4k + 3B(U) = 75000 + 4k

Summary of Example 1

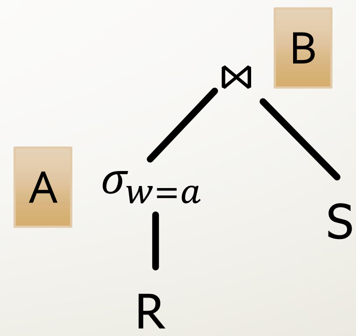
- If k <= 50, cost = 55,000
- If 50 < k <=4950, cost = 75,000 + 2k
- If k > 4950, cost = 75,000 + 4k

- Read Example 16.36.

Example 2: Cost Estimation

- Given relations R(w, $\underline{x}$) and S($\underline{x, y}$), estimate the total cost and size of the final relation.

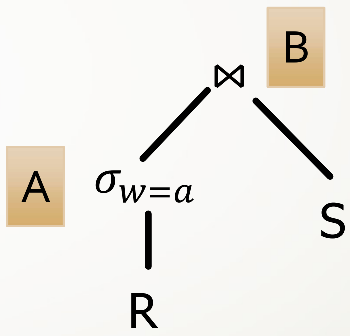
R(w, x)	S(x,y)
T(R) = 10000	T(S) = 5000
B(R) = 500	B(S) = 200
V(R, x) = 10	V(S, x) = 5
V(R, w) = 100	
Clustered index on w	Clustered index on x



- Size of A = T(R)/V(R,w) = 100 tuples.
- Cost of selection (A) = B(R)/ V(R, w) = 500/100 = 5 IOs.

Example 2: Selection A

R(w, x)	S(x,y)
T(R) = 10000	T(S) = 5000
B(R) = 500	B(S) = 200
V(R, x) = 10	V(S, x) = 5
V(R, w) = 100	
Clustered index on w	Clustered index on x



- Using the preservation of values assumption:

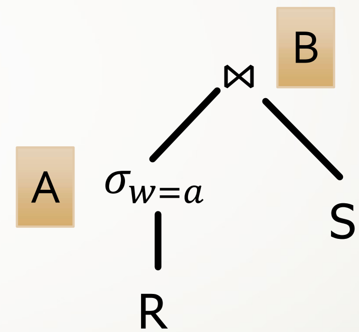
  V(A,x) = V(R, x) = 10
- B(A) = T(A)/(block size: # of tuples per block)

  B(A) = 100/(10000/500)

  = 5 blocks.

Example 2: Join B

R(w, x)	S(x,y)
T(R) = 10000	T(S) = 5000
B(R) = 500	B(S) = 200
V(R, x) = 10	V(S, x) = 5
V(R, w) = 100	
Clustered index on w	Clustered index on x



Assume an index-based nested loop join with materialization will be used in JOIN B

- Cost of join B = B(A) + T(A)*B(S)/V(S,x)

  = 5 + (100 x 200)/5 = 4005 IOs

- Size of B = T(A) * T(S)/ max(V(A,x), V(S,x))

  = (100 * 5000)/10 = 50000 tuples.

- **Total Cost = Cost ($\boldsymbol{\sigma_{w=a}}$) + B(A) + Cost (Join)** = 5+ 5 + 4005 = 4015 IOs.