Query Optimization

CSCI 4380 Database Systems

Query optimization

- Query optimization means finding the lowest cost query plan for a given query
- A query plan decides
  - Which order to execute query operations
  - Which specific implementation to use
- To find the best plan, we need to figure out the cost of individual plans

Cost estimation

- I/O cost is the dominant cost
- Cost depends on the size of relation and the operation being used
- Statistics can be kept about the databases to help with size estimation

Statistics

- TUPLES(R) - the number of tuples in R
- PAGES(R) - the number of pages R spans
- VALUES(R.A) - the number of distinct values stored for R.A
- MINVAL(R.A)/MAXVAL(R.A) - min/max value stored for R.A

Size estimation

- For any condition, we can estimate the size of the output using the given statistics
- Selectivity (s) = the percentage of tuples that will satisfy a condition
- Expected number of tuples (Exp) = total number of tuples expected in the output

Size Estimation

- To estimate the size of the output relation, we concentrate on two issues:
  - Expected number of tuples in the output
    - Keep statistics about the relations and attributes
    - Estimate the size using assumptions on the distribution of values
  - Expected size of the relation (based on the number of attributes in the result)
    - The data model defines the size of each attribute, hence the average size of a tuple with specific attributes can be measured with respect to the data model

Cardinality Estimation

- For each relation R:
  - PAGES(R) is the total number of blocks in R
  - TUPLES(R) is the total number of tuples in R
- For each attribute A of R:
  - VALUES(R.A) is the total number of distinct values stored for R.A
  - MINVAL(R.A), MAXVAL(R.A) are the minimum and maximum stored values for R.A
  - Assume values for R.A are uniformly distributed!

Cardinality Estimation

- Given a condition C, we estimate the reduction factor (reduction(C)) as the fraction of values that will pass this condition C
  - reduction(R.A=value) = 1 / VALUES(R.A)
  - Example: Given a relation R with 10,000 voters in 500 districts, if each district has equal number of voters.
  - Selecting tuples with R.district = 255, we expect 1/500 of the voters to be in this district.
  - As a result, 10,000 * 1/500 = 20 voters in this specific district (or any specific district).

Cardinality estimation

- reduction(min<R.A<max) = 1/3
- Many formulas exist. This is just an approximation.
  - Example:
  - reduction(R.district < 251) = 1/3
  - reduction(R.salary < 80K AND R.salary > 40K) = 1/3

Cardinality estimation

- reduction(C1 and C2) = reduction(C1) * reduction(C2)
- reduction(C1 or C2) = 1 - (1 - reduction(C1))*(1- reduction(C2))
- reduction(not (C1)) = 1 - reduction(C1)
  - For example, 10,000 voters in 500 districts and 2 genders (obviously)
  - R.district = 255 and R.gender='F' = 1/500 * 1/2
  - Hence, 20 voters in the specific district, and roughly half are female, so 10 female voters

Cardinality estimation

- reduction(R.A=R.B) =

  1 / max(VALUES(R.A),VALUES(R.B))

  - For example, 1,000,000 voters in 500 voting districts and 300 districts they were born in
  - reduction(R.votingdistrict = R.birthdistrict) = 1/max(500,300) = 1/500
- This is used to estimate the size of joins!
- For any query (FROM R WHERE C), the cardinality of the output is given by TUPLES(R) * reduction(C)

Size estimation

- The size of the output of a query is given by
  - The number of tuples in the output
  - The size of the tuples in the output (given the projected attributes) -> used to compute number of expected tuples per page

Example

- Suppose FROM R, S WHERE R.B < 101 AND R.A=S.A
  - TUPLES(R) = 1,000,000 and VALUES(R.A)=20,000, VALUES(R.B)=50,000
  - TUPLES(S) = 200,000 and VALUES(S.A) = 5,000
- First, how many tuples in R.B<100?
  - Reduction(R.B<101) = 1/3
  - So, we expect 1,000,000 * 1/3 tuples after selecting on R.B
  - How many distinct values of R.A after the selection on R.B? Assume the same as before, but in this case it can be at most min{1,000,000/3, 20,000} = 20,000!

Example

- How many tuples in (R.B<100) x S (cartesian product)?
  - 1,000,000/3 * 200,000
  - What is reduction factor of (R.A = S.A)?
  - Reduction(R.A=S.A) = min{ 1/20,000, 1/5,000 } = 1/20,000
- The estimated size of FROM R, S WHERE R.B < 101 AND R.A=S.A is then: 1,000,000/3 * 200,000 * 1/20,000

Histograms

- The assumption of all values and ranges being equally likely is generally not meaningful, which may result in bad estimates.
  - All ages are not equally likely, RPI students are more likely to be in the 18-22 age group.
  - All salary are not equally likely, generally the higher paid people are outliers.

Histograms

- Histograms try to find an approximation of the actual data distribution.

Relation

Tuple	Age
1	20
2	21
3	22
4	18
5	15
6	24
7	25
8	26
9	22
10	20

Histogram on age

Age Range	Number of tuples
15-18	2
19-22	5
23-26	3

Size estimation with histograms

Age Range	Number of tuples
15-18	2
19-22	5
23-26	3

- Update the size estimation by assuming that all values within a given range is equally likely.
- For example, given the histogram on the left:
  - exp(Age = 20) = 5 * 1/4
  - exp(Age = 16) = 2 * 1/4
- Of course, size estimates are more meaningful with larger relations.

Index Selectivity

- Index selectivity is an important measure of usefulness of indices:
  - The index selectivity for a selection query is the portion of the query that is answered by the index
  - Note that overall the cost of the index use must be used in understanding how useful an index is.

Index Selectivity

- Example:

  SELECT *

  FROM R

  WHERE R.A = 5 and R.B < 10 and R.C = 8

- Given an index I on R(A,B), the selectivity of the index is the selectivity of the condition R.A = 5 and R.B < 10 (as the condition on R.C cannot be checked in the index).

Index Selectivity

- Example:

  SELECT R.D

  FROM R

  WHERE R.A = 5 and R.B < 10

  An index on R(A,B) and R(A,B,D) will have the same selectivity, but the index on R(A,B,D) is more useful.

Size estimation notes

- The uniform distribution assumption is usually wrong. Then, we need better models for estimates -> histograms
- The "AND/OR" conditions are calculated as if they are independent conditions. This is usually not true. Better estimates on joint probabilities? -> too costly
- Generally, the most important issue is determining the most selective conditions.

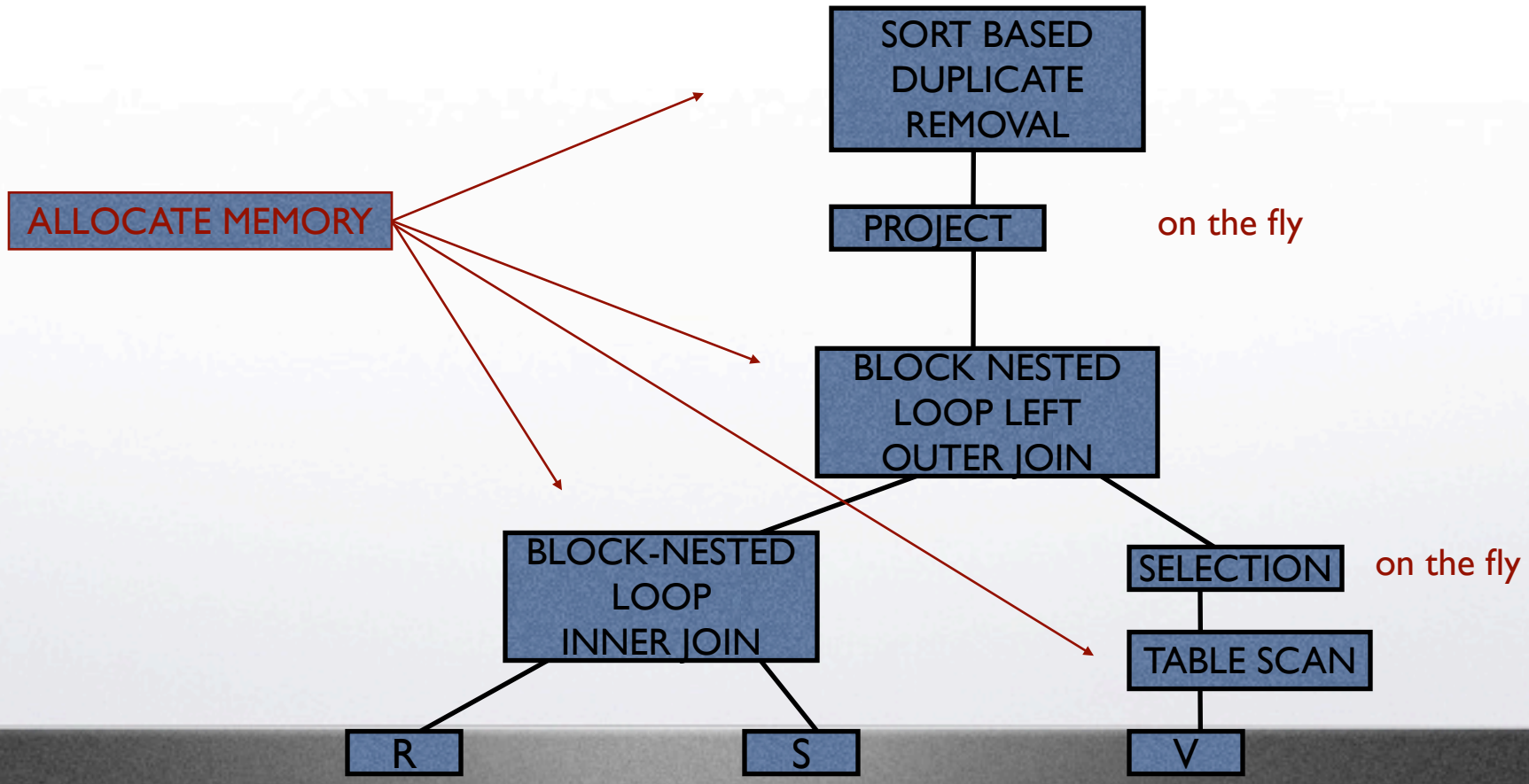
Query Optimization

- The execution of an SQL query:
  - Parse and verify the query, create an equivalent query tree where each node is a relational algebra operation and each leaf is a table from the query
  - Apply rules to alter the query tree to find equivalent queries based on algebraic equivalences
  - Generate alternate execution plans using these equivalent queries

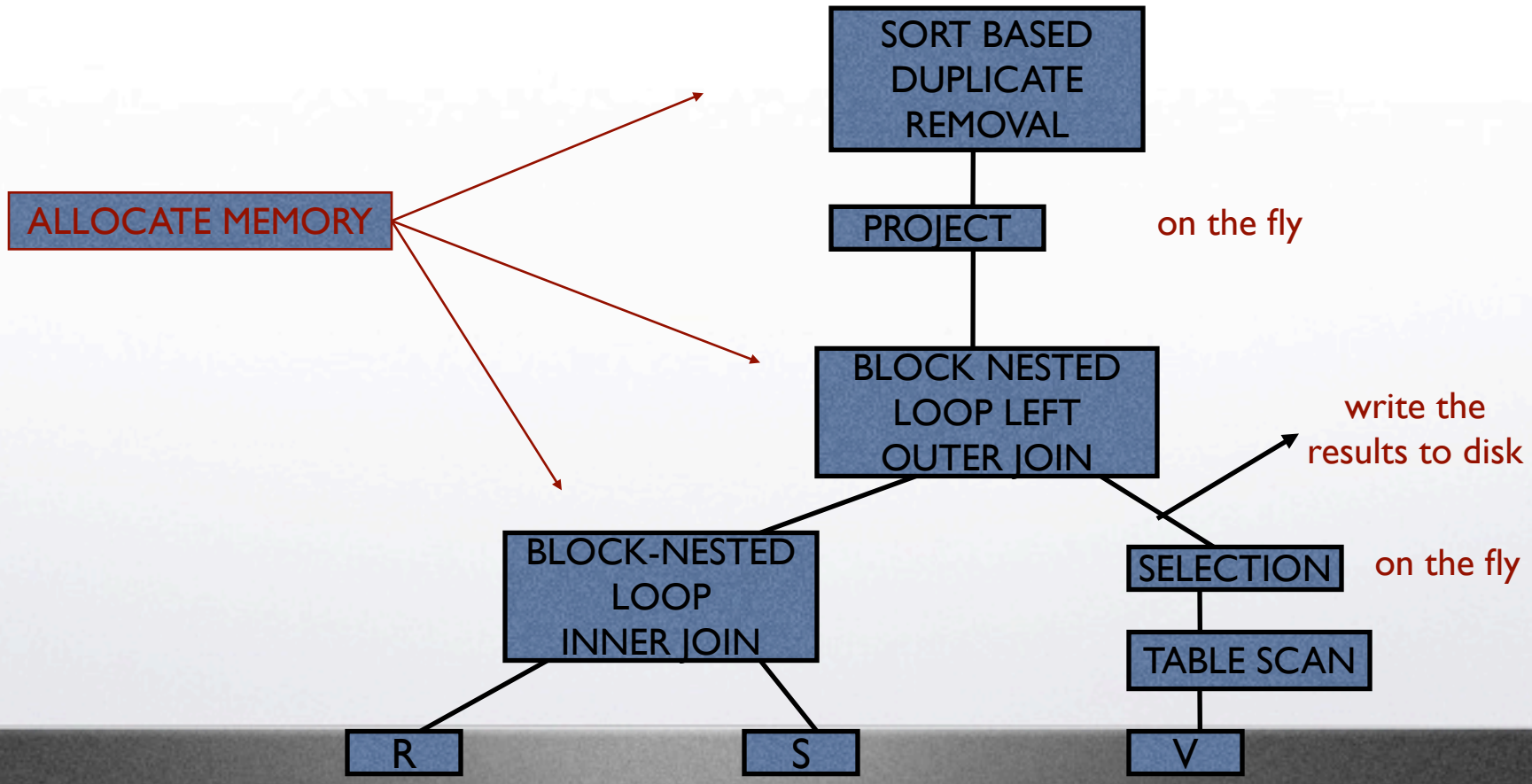
Query Optimization

- For each equivalent query, generate all possible execution plans:
  - assign implementations to operations based on available access paths
  - orders to joins
  - allocate the available memory to operations in an optimal way
  - estimate the cost of each operation based on statistics about the relations and system parameter
- Choose the lowest cost execution plan

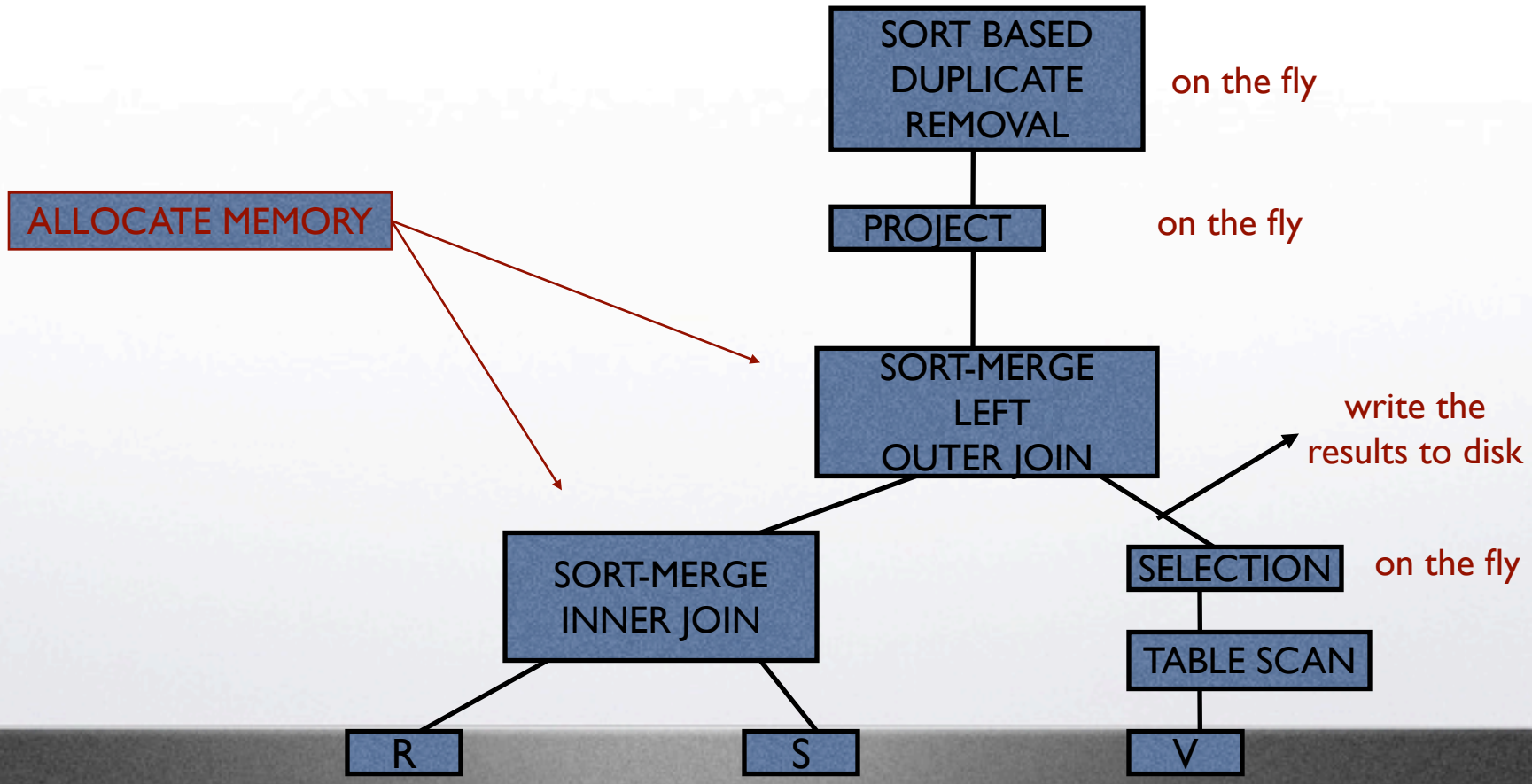
Query plan 1


SORT BASED DUPLICATE REMOVAL
ALLOCATE MEMORY
PROJECT
on the fly
BLOCK NESTED LOOP LEFT OUTER JOIN
BLOCK-NESTED LOOP INNER JOIN
SELECTION
on the fly
TABLE SCAN
R
S
V

Query plan 2


SORT BASED DUPLICATE REMOVAL
ALLOCATE MEMORY
PROJECT
on the fly
BLOCK NESTED LOOP LEFT OUTER JOIN
write the results to disk
BLOCK-NESTED LOOP INNER JOIN
SELECTION
on the fly
TABLE SCAN
R
S
V

Query plan 3

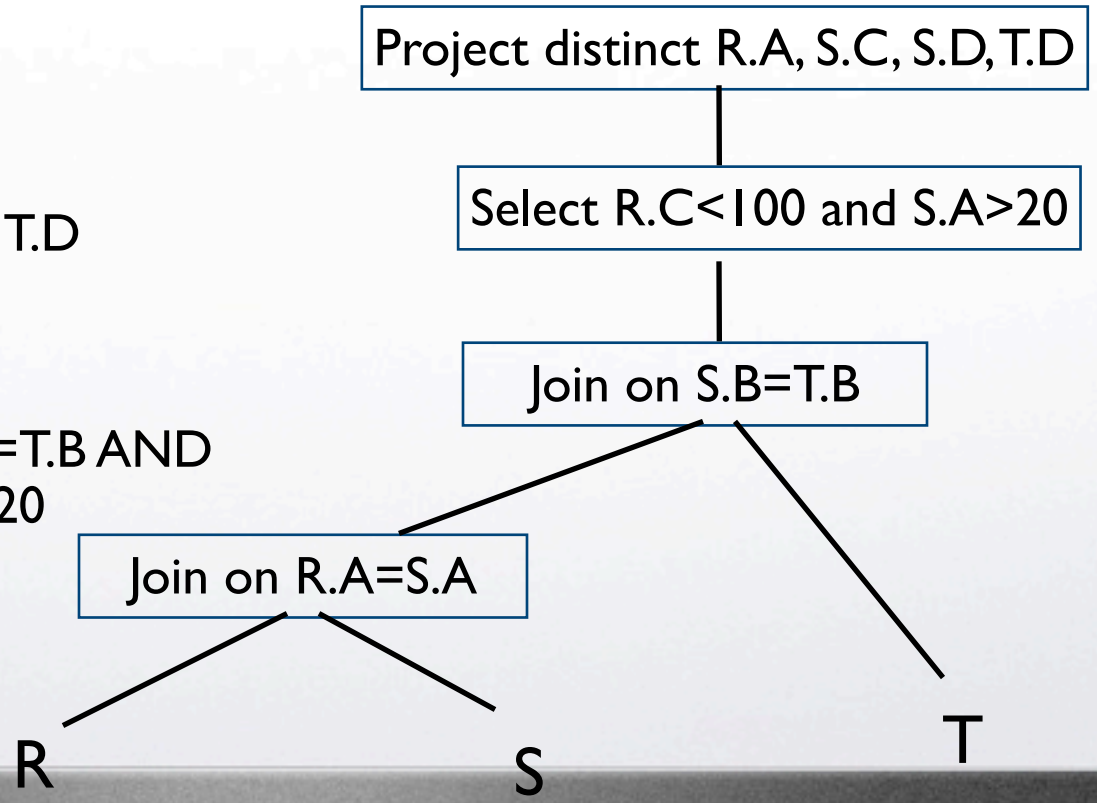

SORT BASED DUPLICATE REMOVAL
on the fly
ALLOCATE MEMORY
PROJECT
on the fly
SORT-MERGE LEFT OUTER JOIN
write the results to disk
SORT-MERGE INNER JOIN
SELECTION
on the fly
TABLE SCAN
R
S
V

Query Optimization

- The search space of an optimizer is very large
  - we'll see why when we look at join orderings
- Use heuristics to
  - favor trees that are almost always beneficial
  - remove trees that are very unlikely to be useful
  - keep trees that may be of high cost in an intermedia state but with an expected high pay off at the end

Example query tree

```
SELECT  DISTINCT
                R.A, S.C, S.D, T.D
FROM    R, S, T
WHERE  R.A=S.A and S.B=T.B AND
    R.C < 100 AND S.A > 20
```



Algebraic Equivalences

- Selections can be pushed through joins, Cartesian products

Given R(A,B,C) and S(D,E,F)

$$\sigma_{A=5 \text{ and } D>20} R \bowtie_{C=D} S$$

is equivalent to:

$$(\sigma_{A=5 \text{ and } C>20} R) \bowtie_{C=D} (\sigma_{D>20} S)$$

Algebraic Equivalences

- Selections can be joined with Cartesian products for a join condition

Given R(A,B,C) and S(D,E,F)

$$\sigma_{C=D} R \times S$$

is equivalent to:

$$R \bowtie_{C=D} S$$

Algebraic Equivalences

- Projections can be pushed through joins, Cartesian products to reduce the size of the output

Given R(A,B,C) and S(D,E,F)

$$\Pi_{B,E}(R \bowtie_{C=D} S)$$

is equivalent to:

$$\Pi_{B,E}(\Pi_{B,C}(R) \bowtie_{C=D} (\Pi_{D,E} S))$$

Algebraic Equivalences

Selection conditions can be simplified to produce equivalent conditions

Given R(A,B,C,D,E)

$$\sigma_{D=5 \text{ and } B<D \text{ and } B<10 \text{ and } D<>8} R$$

is equivalent to:

$$\sigma_{D=5 \text{ and } B<D} R$$

Algebraic Equivalences

- Nested queries can be converted to joins/outer joins under certain conditions

```
SELECT R.A, R.B
FROM R
WHERE  R.A IN (SELECT S.A FROM S)
```

Equivalent to:

```
SELECT R.A, R.B
FROM R,S
WHERE R.A=S.A
```

Under which conditions?

Algebraic Equivalences

```
SELECT Dept.name
FROM Dept
WHERE Dept.num-of-machines >=
      (SELECT COUNT(Emp.*) FROM Emp
       WHERE Dept.name= Emp.Dept-name)
```

Is this equivalent to?

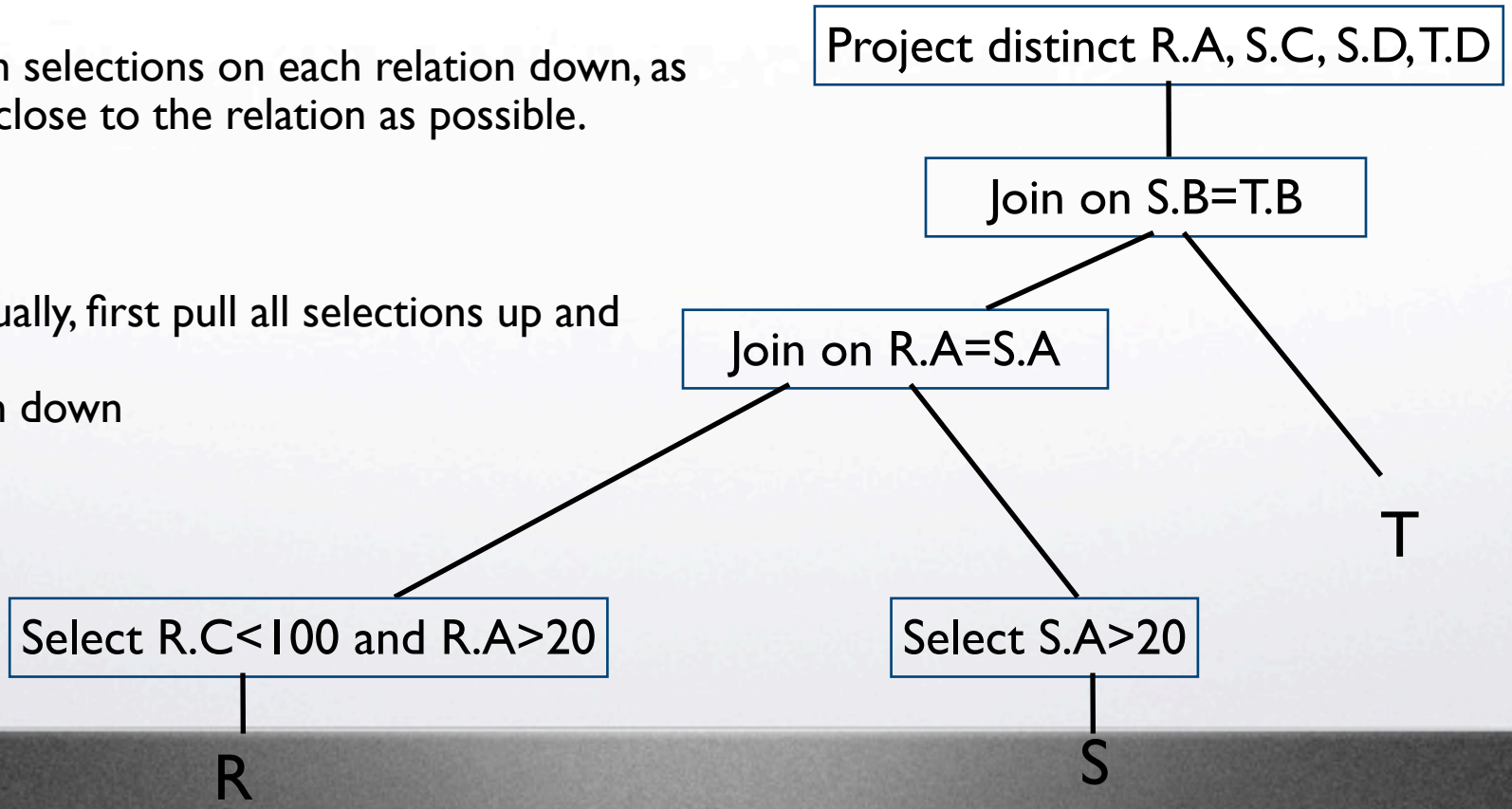
```
SELECT Dept.name
FROM Dept, Emp
WHERE Dept.name = Emp.dept-name
GROUP BY Dept.name, Dept.num-of-machines
HAVING Dept.num-of-machines >= COUNT (Emp.*)
```

Query Optimization Heuristics

Push selections on each relation down, as close to the relation as possible.

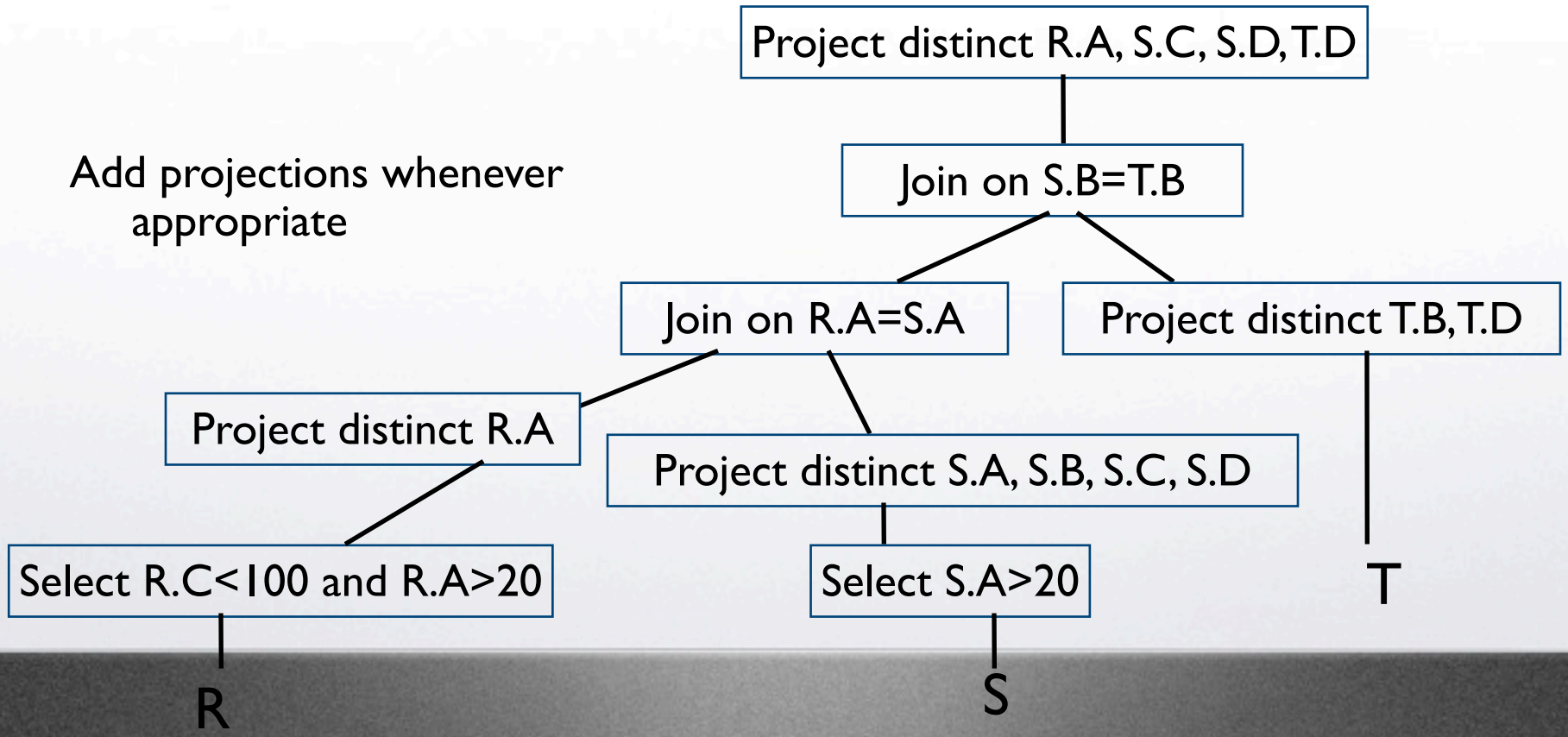
Actually, first pull all selections up and

then down



Query Optimization Heuristics

Add projections whenever appropriate



Pushing selections down

- SELECT C (R join S) = (SELECT C (R)) join S if "C" only involves attributes in R
- SELECT C1 AND C2 (R) = SELECT C1 (SELECT C2 (R)) = SELECT C2 (SELECT C1 (R))
- Selections can be pushed down the joins often to produce the size of the joined relation
- However this may not always result in a reduction in the overall cost.
  - The selection condition may not be very selective.
  - The selection may remove an access condition, sorted order that is particularly useful for the next step.

Implementation plans

- Assign implementations to logical operators given memory limitations
  - Join mapped to block-sort join, merge-sort join, etc.
  - Selection mapped to table scan or index scan, etc.
- Assign join ordering to joins

R join S join T = (R join S) join T = R join (S join T)

= (R join T) join S

For each join, inner/outer relations can be changed

- Estimate the size of each relation and cost of each operation

Implementation plans

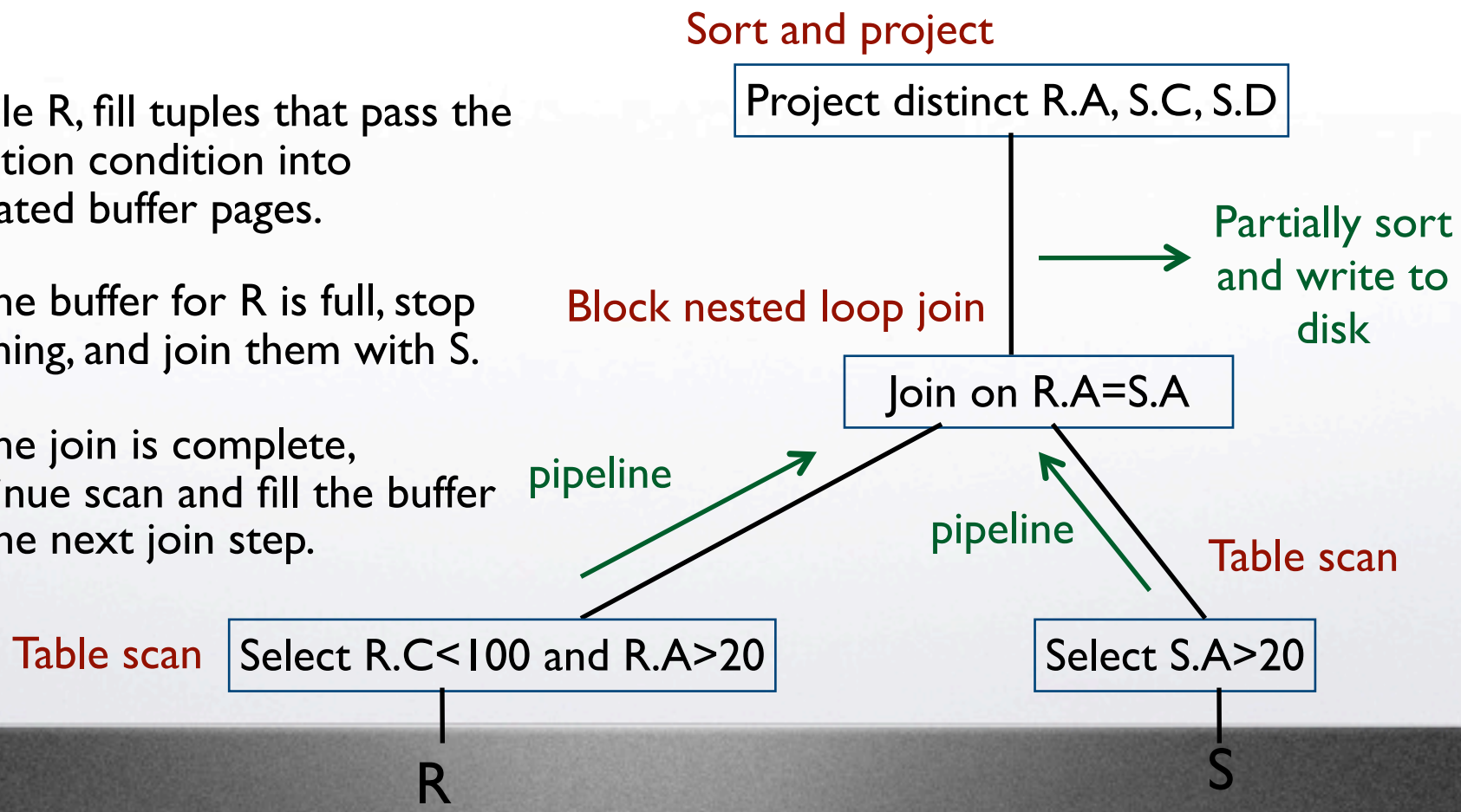
- Blocking operators require the whole relation to be present before any output can be computed
  - For example grouping, sorting, project distinct
- A non-blocking operator can be pipelined
  - As soon as a tuple is found to be in the output of an operator, it can be pipelined to the next operator
  - Hence, the output buffer for an operator serves as the input buffer of the next operator

Query Optimization

Scan table R, fill tuples that pass the selection condition into allocated buffer pages.

When the buffer for R is full, stop scanning, and join them with S.

When the join is complete, continue scan and fill the buffer for the next join step.

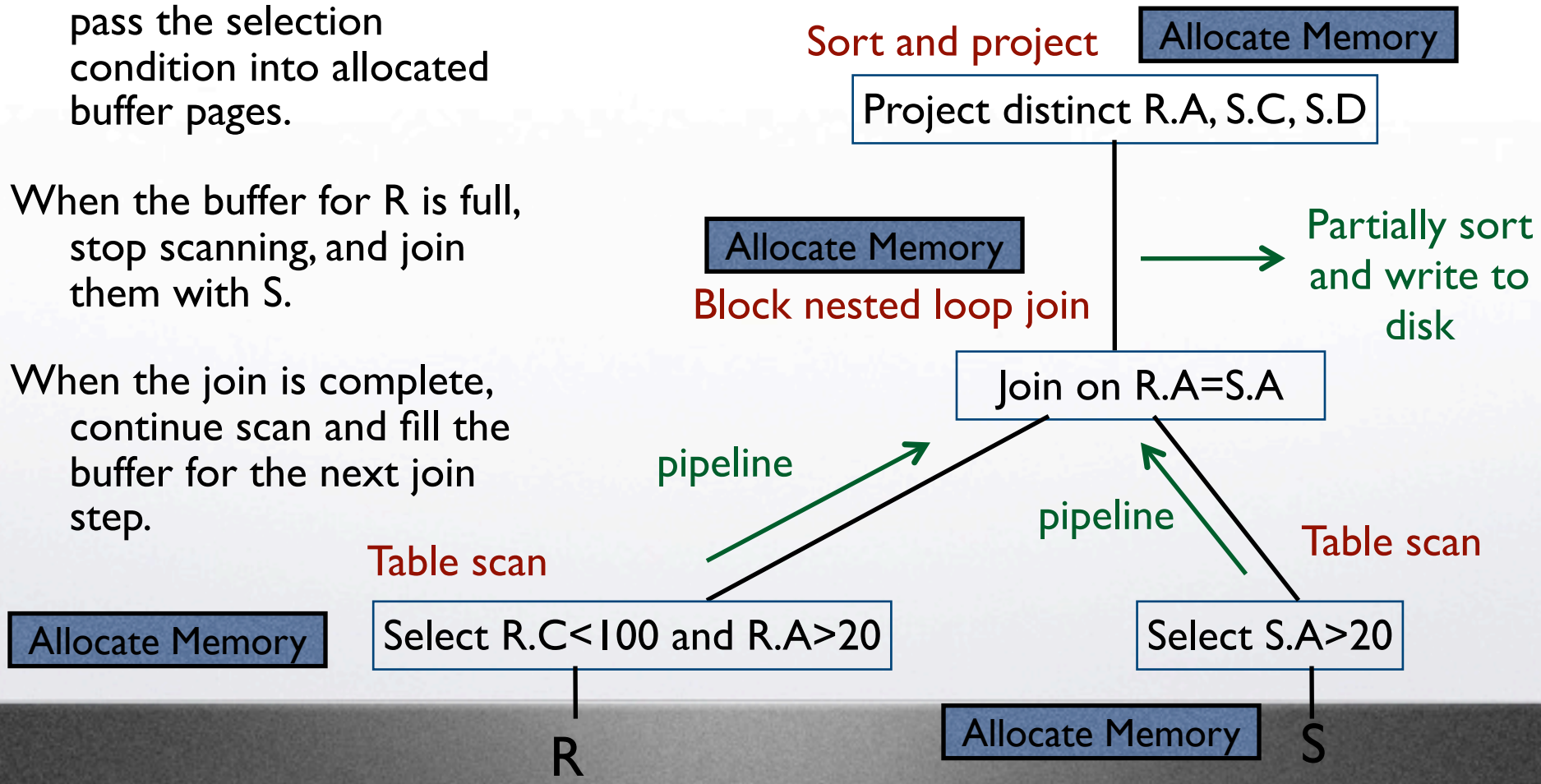


Query Optimization

Scan table R, fill tuples that pass the selection condition into allocated buffer pages.

When the buffer for R is full, stop scanning, and join them with S.

When the join is complete, continue scan and fill the buffer for the next join step.



Join ordering

Join ordering depends on the size of the output and the access paths available for each relation

Table	TUPLES	VALUES (attr A)
R	1 million	10,000
S	100,000	100,000
T	200,000	5,000

(R JOIN S ON R.A=S.A) JOIN T ON S.A=T.A

Size of R JOIN S= 1,000,000 * 100,000 * 1/100,000 = 1,000,000

Size of (R JOIN S) JOIN T= 1,000,000 * 200,000 * 1/10,000
= 20,000,000

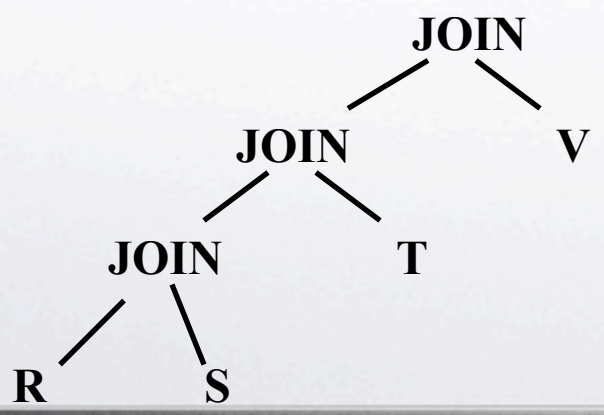
R JOIN ON R.A=S.A (S JOIN T ON S.A=T.A)

Size of S JOIN T= 100,000 * 200,000 * 1/100,000 = 200,000

Size of R JOIN (S JOIN T)= 1,000,000 * 200,000 * 1/10,000
= 20,000,000

Choosing join ordering

- The set of possible join orders is extremely large. Instead concentrate on left deep join orders
- Left join orders make it possible to pipeline the output of one join as input to the other join



To find all possible left-deep join orders

- First find all possible two way joins over the given relations, estimate the cost of the best implementation plan
- Then, find the next relation to join with the result, estimate the cost
- Remove any joins that are too costly compared to the others
- Keep enumerating all joins!