BITS Pilani
Hyderabad Campus

BITS Pilani

Dr. Manik Gupta
Associate Professor
Department of CSIS

innovate achieve lead

BITS Pilani
Hyderabad Campus

Distributed Data Systems(CS G544)
Lecture 17-19

Thursday , 26th Sept 2024

Lecture Recap

- Query Decomposition
- Data Localization

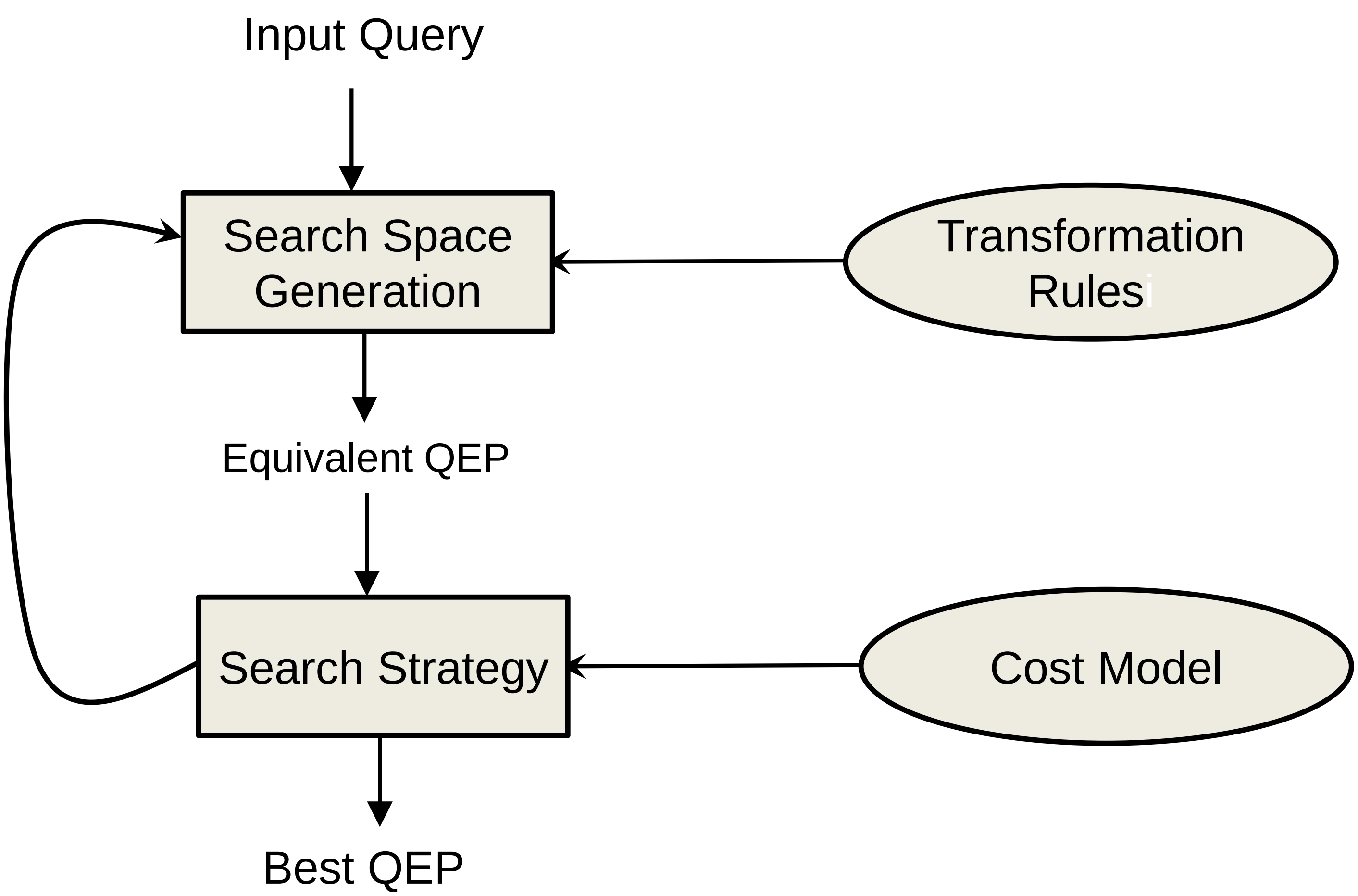
Query Optimization

- Query optimization refers to the process of producing a **query execution plan** (QEP) which represents an execution strategy for the query.
- The selected plan minimizes the objective cost function.
- A **query optimizer** is the software module that performs query optimization.

Query Optimizer

- The Query Optimizer usually has three components:
  - A **search space**: is the set of alternative execution plans to represent the input query.
  - The **cost model**: to predict the cost of a given execution plan. To be accurate, the cost model must have good knowledge about the **distributed execution environment.**
  - The **search strategy**: explores the search space and selects the best plan using the cost model. It defines which plans are examined and in which order.

Query Optimization Process



Search Space

- The query execution plans are typically extracted by means of **operator trees**, which define the order in which the operations are executed.
- For a given query the search space can be defined as the set of **equivalent operator trees** that can be reduced using the **transformation rules**.

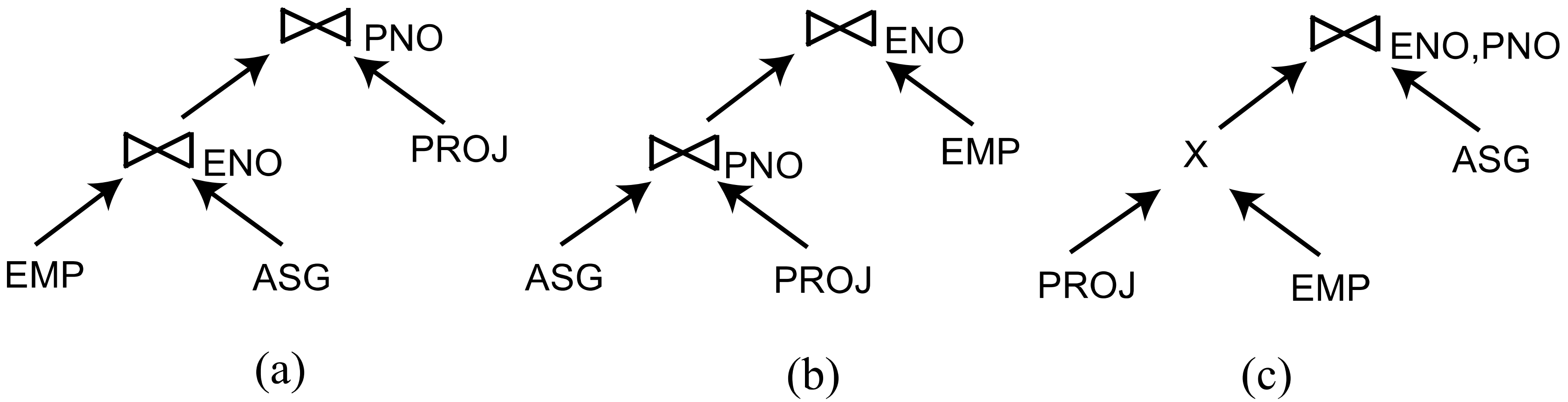
Example- Equivalent Join Trees

Select ename, resp

from EMP, ASG, PROJ

where EMP.eno=ASG.eno

AND ASG.pno= PROJ.pno



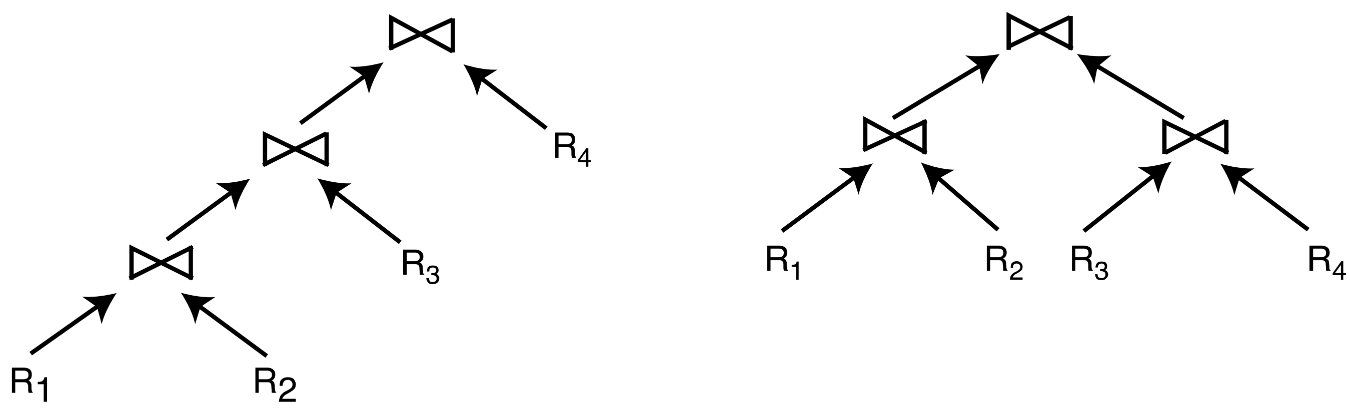
- Three equivalent join trees for the query, which are obtained by exploiting the associativity of binary operators.
- Each of these join trees can be assigned a cost based on the estimated cost of each operator.
- Join tree (c) which starts with a Cartesian product may have a much higher cost than the other join trees.

Search space

- Query optimizers typically restrict the size of the search space they consider. The first restriction is to use **heuristics**.
  - The most common heuristic is to perform selection and projection when accessing base relations.
  - Another common heuristic is to avoid Cartesian products that are not required by the query.
- Another important restriction is with respect to the shape of the join tree.
  - Two kinds of join trees are usually distinguished: linear versus bushy trees

Search space

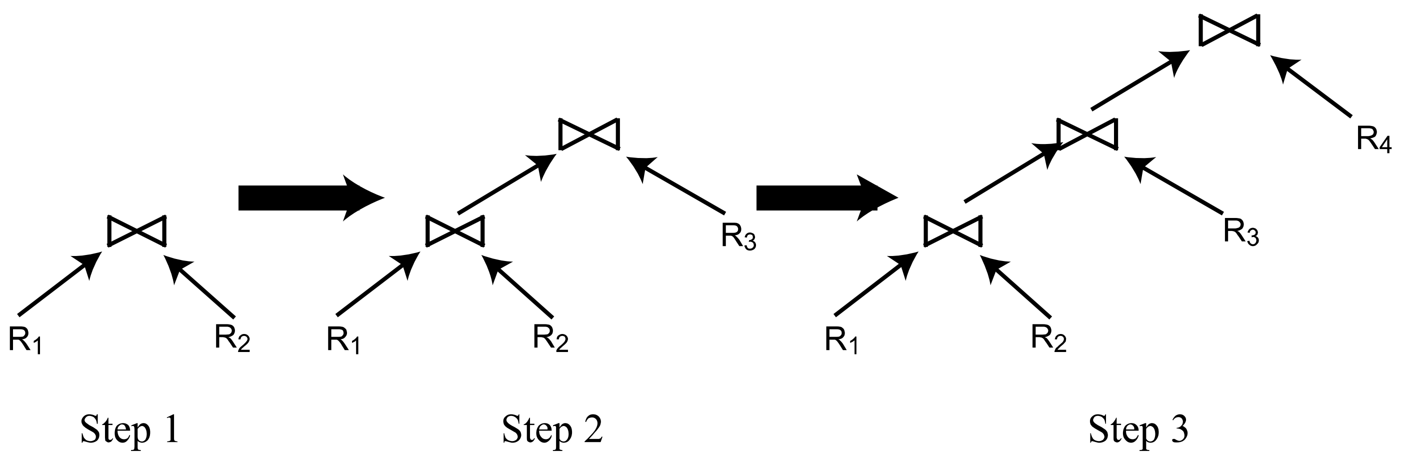
- A *linear tree* is a tree such that at least one operand of each operator node is a base relation.
- A *bushy tree* is more general and may have operators with no base relations as operands (i.e., both operands are intermediate relations).



(a) linear join tree (b) bushy join tree

Search strategy

- The most popular search strategy used by query optimizers is **dynamic programming**, which is **deterministic**.
- Deterministic strategies proceed by **building plans**, starting from base relations, joining one more relation at each step until complete plans are obtained



Search strategy

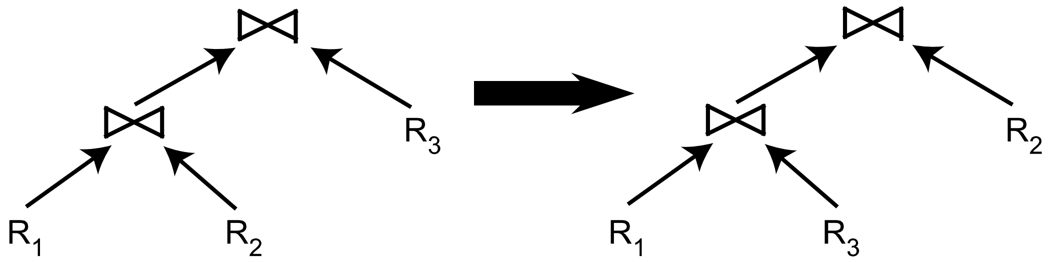
- Dynamic programming builds all possible plans, breadth-first, before it chooses the “best” plan. To reduce the optimization cost, partial plans that are not likely to lead to the optimal plan are **pruned** (i.e., discarded) as soon as possible.
- By contrast, another deterministic strategy, the **greedy algorithm**, builds only **one plan, depth-first**.

Dynamic programming approach

- Dynamic programming approach becomes too expensive when the number of relations is greater than 5 or 6.
- For more complex queries, **randomized strategies** have been proposed, which reduce the optimization complexity but do not guarantee the best of all plans.

Randomized strategies

- First, one or more **start plans** are built by a greedy strategy.
- Then, the algorithm tries to improve the start plan by **visiting its neighbors**. A neighbor is obtained by applying a random transformation to a plan.
- An example of a typical transformation consists in **exchanging two randomly chosen operand relations** of the plan.



Distributed Cost Model

- An optimizer's cost model includes cost functions to predict the cost of operators, statistics and base data and formulas to evaluate the **sizes of the intermediate results.**
- The cost is in terms of **execution time**, so a cost function represents the execution time of a query.

Cost function

– The cost of a distributed execution strategy can be expressed with respect to either the total time or the response time.
  - **Total time** = sum of all the time components.
  - **Response time** = time elapsed from the initiation to the completion of the query.

Total time

$(T_{CPU} * \#insts) + (T_{I/O} * \#I/Os) + (T_{MSG} * \#msgs) + (T_{TR} * \#bytes)$

T_{CPU} = time of a CPU instruction

$T_{I/O}$ = time of disk I/O

T_{MSG} = fixed time of initiating and receiving a message

T_{TR} = time to transmit a data unit from one site to another

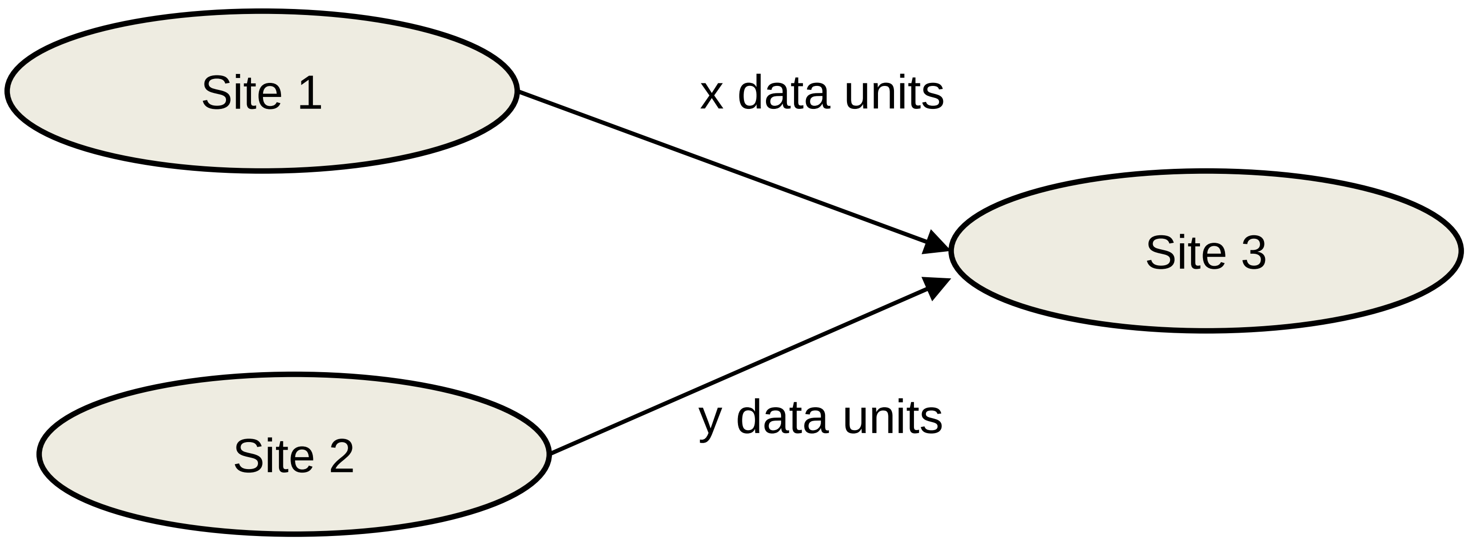
Costs are generally expressed in time units which can be translated into other units.

Response time

$(T_{CPU} * seq\_\#insts) + (T_{I/O} * seq\_\#I/Os) +$
$(T_{MSG} * seq\_\#msgs) + (T_{TR} * seq\_\#bytes)$

where seq_#x, in which x can be instructions (insts), I/O, msgs or bytes, is the **maximum number of x which must be done sequentially** for the execution of the query.

Example data transfer for a query



Total time= $2 * T_{MSG} + T_{TR} * (x+y)$

Response time= $\max \{T_{MSG} + T_{TR} * x, T_{MSG} + T_{TR} * y\}$
// since the transfer can be done in parallel.

Database Statistics

- Primary cost factor: size of intermediate relations
  - Need to estimate their sizes
  - Estimation is based on **statistical information about the base relations** and **formulas to predict the cardinalities of the results of the relational operations**
- Make statistics precise $\Rightarrow$ more costly to maintain

Statistics

For each relation $R[A_1, A_2, \ldots, A_n]$ fragmented as $R_1, \ldots, R_r$

- length of each attribute: $length(A_i)$
- the number of distinct values for each attribute in each fragment: $card(\Pi_{A_i} R_j)$
- maximum and minimum values in the domain of each attribute: $min(A_i)$, $max(A_i)$
- the cardinalities of each domain: $card(dom[A_i])$
- the number of tuples in each fragment: $card(R_j)$
- **join selectivity factor** for pair of relations is the proportion of tuples participating in the join, denoted by SF, of relations R and S is a real number between 0 and 1.

$$SF_{\bowtie}(R,S) = \frac{card(R \bowtie S)}{card(R) * card(S)}$$

0.5 corresponds to a very large joined relation; and 0.001 corresponds to a small one.

- Size of intermediate relations $size(R) = card(R) \times length(R)$

Intermediate Relation Sizes

Cardinality of Selection

$card(\sigma_F(R)) = SF_S(F) \times card(R)$

where, ***selectivity factor*** of an operation, the proportion of tuples of an operand relation that participate in the result of that operation, is denoted SF_{OP}, where *OP* denotes the operation.

Projection

$card(\Pi_A(R))=card(R)$

Cartesian Product

$card(R \times S) = card(R) * card(S)$

Union

upper bound: $card(R \cup S) = card(R) + card(S)$

lower bound: $card(R \cup S) = max\{card(R), card(S)\}$

Set Difference

upper bound: $card(R–S) = card(R)$

lower bound: 0

Intermediate Relation Size

Join

– Special case: *A* is a key of *R* and *B* is a foreign key of *S*

$$card(R \bowtie_{A=B} S) = card(S)$$

Because each tuple of *S* matched with atmost one tuple of *R*

– More general:

$$card(R \bowtie S) = SF_J * card(R) * card(S)$$

Semijoin

$$card(R \ltimes_A S) = SF_{\ltimes}(S.A) * card(R)$$

Selectivity factor of attribute A of S, denoted $SF_{\ltimes}(S.A)$

where

$$SF_{\ltimes}(R \ltimes_A S) = SF_{\ltimes}(S.A) = \frac{card(\prod_A(S))}{card(dom[A])}$$

Using Histograms for Selectivity Estimation

- An effective solution to accurately capture data distributions is to use histograms.
- Today, most commercial DBMS optimizers support histograms as part of their cost model.
- Exercise: Find out how histograms can be used to accurately estimate the selectivity of selection operations?

Centralized Query Optimization

- We discuss popular query optimization techniques for centralized systems.
  - First, a distributed query is translated into local queries, each of which is processed in a centralized way.
  - Second, distributed query optimization techniques are often extensions of centralized techniques.
  - Finally, centralized query optimization is a simpler problem; the minimization of communication costs makes distributed query optimization more complex.
- Two approaches:
  - INGRES Algorithm
  - System R Algorithm

INGRES Algorithm

- INGRES uses **dynamic query optimization**.
- This algorithm recursively breaks up the calculus query into smaller pieces.
- A query is decomposed into a sequence of queries having a unique relation in common.
- Then each **monorelation query** is processed by one-variable query processor (OVQP).
- OVQP optimizes the access method to a single relation by selecting best suitable access method (based on the predicate).
- This uses techniques like **detachment and substitution.**

Dynamic Algorithm– Decomposition

- Replace a relation query q by a series of n subqueries

  $q_1 \rightarrow q_2 \rightarrow \ldots \rightarrow q_n$

  where q_i uses the result of q_{i-1}.
- **Detachment**
  - Query q decomposed into $q' \rightarrow q''$ where q' and q'' have a common relation which is the result of q'
- **Tuple substitution**
  - Replace the value of each tuple with actual values and simplify the query

$$q(R_1, R_2, \ldots, R_n) \text{ is replaced by } \{q'(t_{1i}, R_2, R_3, \ldots, R_n), t_{1i} \in R_1\}$$

Example

Names of employees working on CAD/CAM project

```
q1: SELECT   EMP.ENAME
    FROM     EMP, ASG, PROJ
    WHERE    EMP.ENO=ASG.ENO
    AND      ASG.PNO=PROJ.PNO
    AND      PROJ.PNAME="CAD/CAM"
```

$\Downarrow$

```
q11: SELECT  PROJ.PNO INTO JVAR
     FROM    PROJ
     WHERE   PROJ.PNAME="CAD/CAM"

q':  SELECT  EMP.ENAME
     FROM    EMP,ASG,JVAR
     WHERE   EMP.ENO=ASG.ENO
     AND     ASG.PNO=JVAR.PNO
```

- What are the different steps involved in INGRES algorithm?
- What is the purpose of this step?

Example (cont'd)

```
q':    SELECT   EMP.ENAME
       FROM     EMP,ASG,JVAR
       WHERE    EMP.ENO=ASG.ENO
       AND      ASG.PNO=JVAR.PNO
```

$$\Downarrow$$

```
q12:   SELECT   ASG.ENO INTO GVAR
       FROM     ASG,JVAR
       WHERE    ASG.PNO=JVAR.PNO

q13:   SELECT   EMP.ENAME
       FROM     EMP,GVAR
       WHERE    EMP.ENO=GVAR.ENO
```

– What is the purpose of this step?
– What is the problem?
– What will happen next?

Example

- q_{11} is a mono-variable query
- q_{12} and q_{13} is subject to tuple substitution
- Assume `GVAR` has two tuples only: ⟨`E1`⟩ and ⟨`E2`⟩hence, substitution of GVAR generates two one-relation subqueries.

Then q_{13} becomes

q_{131}:
```
SELECT    EMP.ENAME
FROM      EMP
WHERE     EMP.ENO="E1"
```

q_{132}:
```
SELECT    EMP.ENAME
FROM      EMP
WHERE     EMP.ENO="E2"
```

Summary of INGRES

- The algorithm works recursively until there remain no more mono relation queries to be processed.
  - It consists of applying the selections and projections as soon as possible by **detachment**.
  - The irreducible queries that remain after detachment must be processed by tuple **substitution**.

System R Algorithm

- The System R (centralized) query optimization algorithm performs **static query optimization** based on "**exhaustive search**" of the solution space and a cost function (I/O cost + CPU cost)
  - Input: relational algebra tree
  - Output: optimal relational algebra tree
  - **Dynamic programming** technique is applied to reduce the number of alternative plans

System R Algorithm

- The optimization algorithm consists of two major steps.
  - Simple (i.e., mono-relation) queries are executed according to the best access path
  - Execute joins
    - Determine the possible ordering of joins
    - Determine the cost of each ordering
    - Choose the join ordering with minimal cost

System R Algorithm

For the join of two relations,

- the relation whose tuples are read first is called the **external**,
- while the other, whose tuples are found according to the values obtained from the external relation, is called the **internal** relation.

- Efficient access to inner relation is crucial
  - Important decision to determine the cheapest access path to the internal relation.

System R Algorithm

- For joins, two alternative algorithms:
- Nested loops

for each tuple of *external* relation (cardinality n_1)

  for each tuple of *internal* relation (cardinality n_2)

    join two tuples if the join predicate is true

  end

End

Complexity: $n_1 * n_2$

- Merge join
  - sort relations on join attribute
  - merge relations
  - Complexity: $n_1 + n_2$ if relations are previously sorted and equijoin

Algorithm

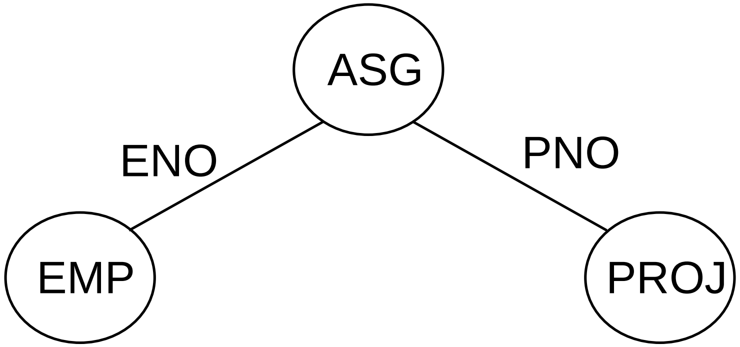
- Two loops:
  - First uses the best single relation access method to each relation in the query
  - Second examines all possible permutations of join orders and selects the best access strategy for the query.
- The permutations are produced by dynamic construction of a tree of alternative strategies.

System R – Example

Names of employees working on the CAD/CAM project

Assume

- EMP has an index on ENO,
- ASG has an index on PNO,
- PROJ has an index on PNO and an index on PNAME



- What are the steps in System R execution?

System R – Example (cont'd)

- Choose the best access paths to each relation

  EMP: sequential scan (no selection on EMP)

  ASG: sequential scan (no selection on ASG)

  PROJ: index on PNAME (there is a selection on PROJ based on PNAME)

- Determine the best join ordering

  EMP ⋈ ASG ⋈ PROJ

  ASG ⋈ PROJ ⋈ EMP

  PROJ ⋈ ASG ⋈ EMP

  ASG ⋈ EMP ⋈ PROJ

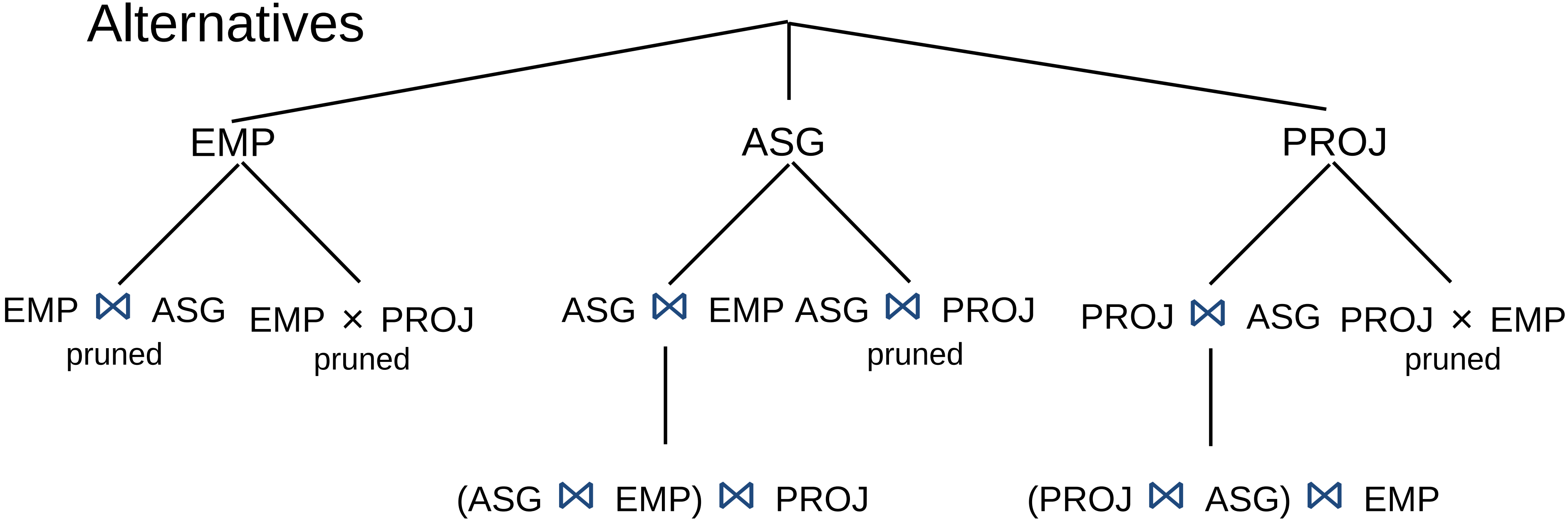
  EMP × PROJ ⋈ ASG

  PRO × EMP ⋈ ASG

  Select the best ordering based on the join costs evaluated according to the two methods

System R – Example (cont'd)

Alternatives



- First level of the tree indicates the best single-relation access method.
- Second level indicates the best join method with any other relation.
- Best total join order is given by one of
  - ((ASG ⋈ EMP) ⋈ PROJ)
  - ((PROJ ⋈ ASG) ⋈ EMP)

Assume that (EMP ▷◁ ASG) and (ASG ▷◁ PROJ) have a cost higher than (ASG ▷◁ EMP) and (PROJ ▷◁ ASG), respectively.

System R – Example (cont'd)

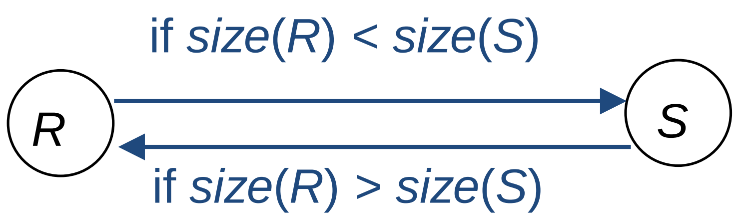
- ((PROJ ⋈ ASG) ⋈ EMP)) has a useful index on the select attribute and direct access to the joining tuples of ASG and EMP
- Therefore, chose it with the following access methods:
  - select PROJ using index on PNAME
  - then join with ASG using index on PNO
  - then join with EMP using index on ENO

Join Ordering in Distributed Queries

- Ordering joins is an important aspect of centralized query optimization.
- Join ordering in a distributed context is even more important since **joins between fragments may increase the communication time.**
- Two basic approaches exist to order joins in distributed queries.
  - One tries to **optimize the ordering of joins** directly
  - whereas the other **replaces joins by combinations of semijoins** in order to minimize communication costs.

Join Ordering

- The objective of the join-ordering algorithm is to transmit smaller operands.
- Let us first concentrate on the simpler problem of operand transfer in a single join.

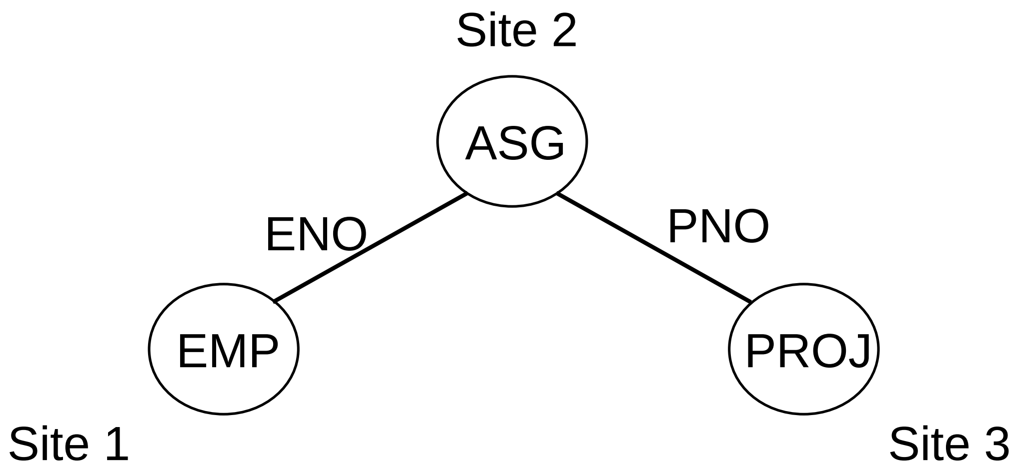


- Problem more difficult in case of multiple relations because of too many alternatives.
  - Compute the cost of all alternatives and select the best one.
    - Necessary to compute the size of intermediate relations which is difficult.

Join Ordering – Example

Consider

PROJ $\bowtie_{PNO}$ ASG $\bowtie_{ENO}$ EMP



Join Ordering – Example

Execution alternatives:

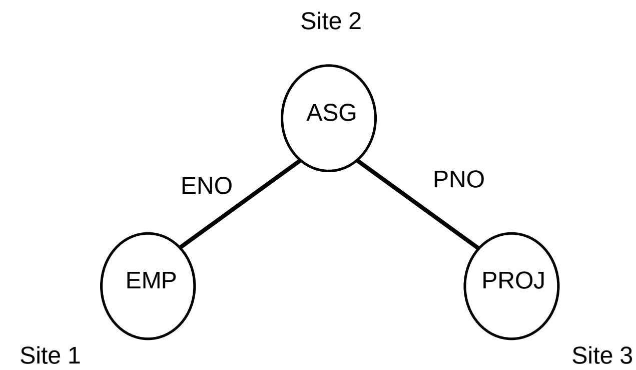
1. EMP → Site 2
   Site 2 computes EMP'=EMP $\bowtie$ ASG
   EMP' → Site 3
   Site 3 computes EMP' $\bowtie$ PROJ

2. ASG → Site 1
   Site 1 computes EMP'=EMP $\bowtie$ ASG
   EMP' → Site 3
   Site 3 computes EMP' $\bowtie$ PROJ

3. ASG → Site 3
   Site 3 computes ASG'=ASG $\bowtie$ PROJ
   ASG' → Site 1
   Site 1 computes ASG' ▷◁ EMP

4. PROJ → Site 2
   Site 2 computes PROJ'=PROJ $\bowtie$ ASG
   PROJ' → Site 1
   Site 1 computes PROJ' $\bowtie$ EMP

5. EMP → Site 2
   PROJ → Site 2
   Site 2 computes EMP $\bowtie$ PROJ $\bowtie$ ASG



Semijoin Algorithms

- The main shortcoming of the join approach is that entire operand relations must be transferred between sites. The semijoin acts as a **size reducer** for a relation much as a selection does.
- Consider join of two relations *R* and *S* over attribute *A*, stored at sites 1 and 2, respectively - can be computed by replacing one or both operand relations by a semijoin with the other relation
- Alternatives:
  - Perform the join $R \bowtie_A S$
  - Perform one of the semijoin equivalents

$$
\begin{aligned}
R \bowtie_A S \quad &\Leftrightarrow \quad (R \ltimes_A S) \bowtie_A S \\
&\Leftrightarrow \quad R \bowtie_A (S \ltimes_A R) \\
&\Leftrightarrow \quad (R \ltimes_A S) \bowtie_A (S \ltimes_A R)
\end{aligned}
$$

Semijoin Algorithms

- The use of the semijoin is beneficial if the **cost to produce and send it to the other site is less** than the cost of sending the whole operand relation and of doing the actual join.

Semijoin Algorithms

- Perform the join
  - send R to Site 2
  - Site 2 computes $R \bowtie_A S$
- Consider semijoin $(R \ltimes_A S) \bowtie_A S$
  - $S' = \Pi_A(S)$
  - $S' \rightarrow$ Site 1
  - Site 1 computes $R' = R \ltimes_A S'$
  - $R' \rightarrow$ Site 2
  - Site 2 computes $R' \bowtie_A S$

  - When will semijoin be better?

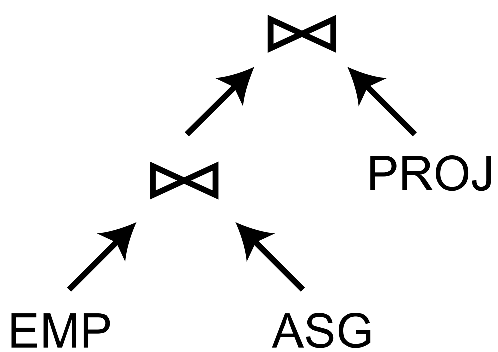
Semijoin Algorithms

- Semijoin is better if
  - **size(Π_A(S)) + size(R $\ltimes_A$ S)) < size(R)**

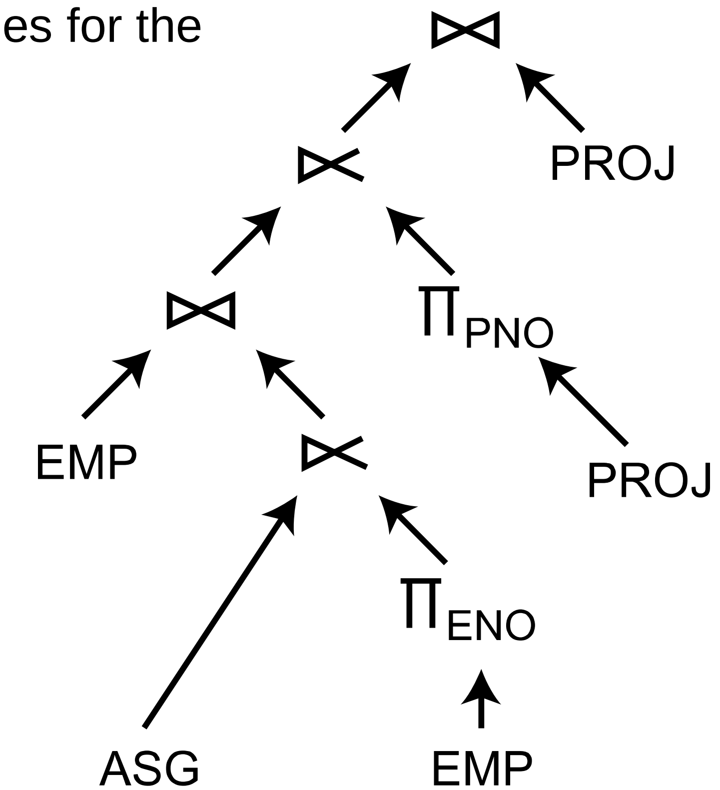
- The semijoin approach is better if the semijoin acts as a **sufficient reducer**, that is, if a few tuples of *R* participate in the join.
- The join approach is better if almost all tuples of *R* participate in the join, because the semijoin approach requires an additional transfer of a projection on the join attribute.

Join versus Semijoin

Equivalent pair of join and semijoin strategies for the query



(a) Join approach



(b) Semijoin approach

Join versus Semijoin

- The join of two relations, EMP $\bowtie$ ASG is done by sending one relation, ASG, to the site of the other one, EMP, to complete the join locally.
- When a semijoin is used, however, the transfer of relation ASG is avoided. Instead, it is replaced by the **transfer of the join attribute values of relation EMP to the site of relation ASG**, followed by the **transfer of the matching tuples of relation ASG to the site of relation EMP**, where the join is completed.
- If the join attribute length is smaller than the length of an entire tuple and the **semijoin has good selectivity**, then the semijoin approach can result in significant savings in communication time.
- Using semijoins may well **increase the local processing time**, since one of the two joined relations must be accessed twice.

Exercise 1

Apply the dynamic query optimization algorithm to the query, and illustrate the successive detachments and substitutions by giving the monorelation subqueries generated.

```
SELECT ENAME, PNAME
FROM EMP, ASG, PROJ
WHERE DUR > 12
AND EMP.ENO = ASG.ENO
AND (TITLE = "Elect. Eng."
OR ASG.PNO < "P3")
AND ASG.PNO = PROJ.PNO
```

Solution

After detachment of the selection on ASG, this query is replaced by q_1 followed by q_2, where GVAR is an intermediate relation.

q_1:
```
SELECT ENO, PNO INTO GVAR
FROM   ASG
WHERE  DUR > 12
```

q_2:
```
SELECT ENAME, PNAME
FROM   EMP, GVAR, PROJ
WHERE  EMP.ENO=GVAR.ENO
AND    (TITLE = "Elect. Eng."
OR     ASG.PNO < "P3")
AND    PROJ.PNO=GVAR.PNO
```

Solution (contd.)

Query q_1 is monorelation and can be preformed by the OVQP. Query q_2 can be further detached into q_{21} followed by q_{22}, where EGVAR is an intermediate relation.

q_{21}:
```
SELECT ENAME, PNO INTO EGVAR
FROM   EMP, GVAR
WHERE  EMP.ENO=GVAR.ENO
AND    (TITLE = "Elect. Eng."
OR     ASG.PNO < "P3")
```

q_{22}:
```
SELECT ENAME, PNAME
FROM   EGVAR, PROJ
WHERE  EGVAR.PNO=PROJ.PNO
```

Queries q_{21} and q_{22} are irreducible and must be processed by tuple substitution.

Lecture Summary

- Query Optimization

Thanks…

- Next Lecture
  - Distributed Query Optimization

- Questions??