Systems Infrastructure for Data Science

Web Science Group
Uni Freiburg
WS 2013/14

Lecture VIII: Distributed query processing and optimization

Roadmap

- Overview
- (Query Decomposition)
- Data Localization
- Query Optimization

Query Processing Recap

Declarative query specification Q → **Query Processor** → Procedural query execution plan Q'

SQL

```
SELECT   ENAME
FROM     EMP, ASG
WHERE    EMP.ENO = ASG.ENO
   AND   RESP = "Manager"
```

Relational Algebra

$$\Pi_{ENAME} (EMP \bowtie_{ENO} (\sigma_{RESP=\text{“Manager”}} (ASG)))$$

Two important requirements:

1. Correctness: Q' must be semantically equivalent to Q.
2. Efficiency: Q' must have the smallest execution cost.

Cost Metrics

- Total cost
  - processing time at all sites (CPU + I/O)
  - communication time between sites
- In WANs, communication cost usually dominates.

- Query response time
  - time elapsed for executing the query

What is the difference between total cost and query response time?
Does it change in distributed/parallel settings?

Complexity of Relational Algebra Operators

Operation	Complexity
Select Project (without duplicate elimination)	$O(n)$
Project (with duplicate elimination) Group by	$O(n*\log n)$
Join Semijoin Division Set Operators	$O(n*\log n)$
Cartesian Product	$O(n^2)$

n: relation cardinality

To reduce costs:

- The most selective operations should be performed first.
- Operations should be ordered by increasing complexity.

Query Processing in a Centralized System

Given:

EMP(ENO, ENAME, TITLE)
ASG(ENO, PNO, RESP, DUR)

Query:

Find the names of employees who are managing a project.

```
SELECT   ENAME
FROM     EMP, ASG
WHERE    EMP.ENO = ASG.ENO
   AND   RESP = "Manager"
```

Two equivalent execution plans. Which one to use?

1. $\Pi_{ENAME}(\sigma_{RESP=\text{"Manager"} \text{ AND } EMP.ENO=ASG.ENO}(EMP \times ASG))$
2. $\Pi_{ENAME}(EMP \bowtie_{ENO}(\sigma_{RESP=\text{"Manager"}}(ASG)))$ ✓

Query Processing in a Distributed System

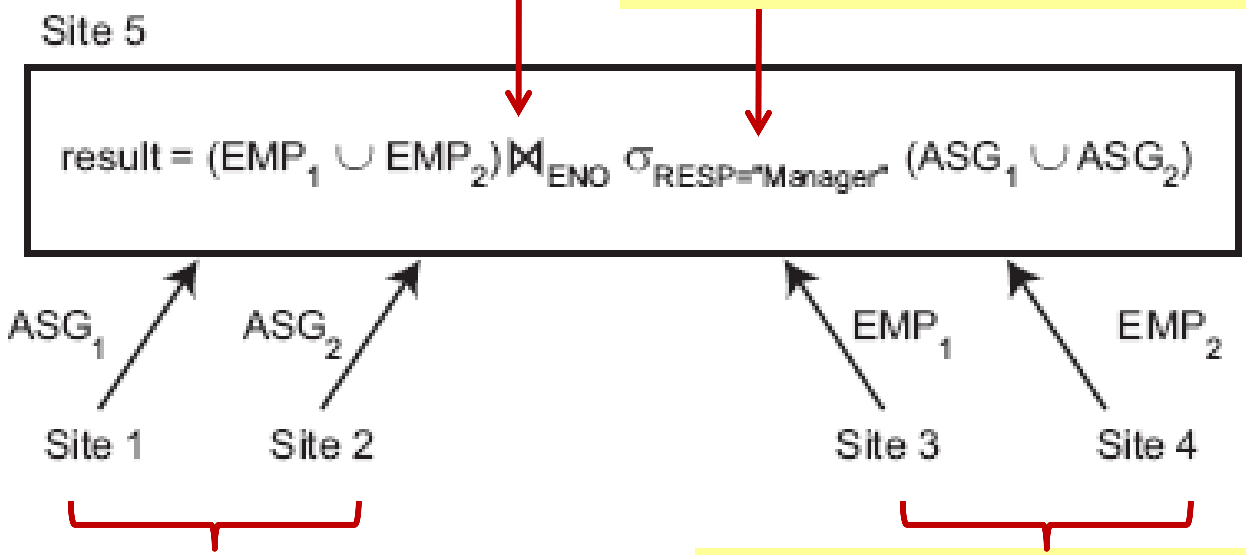
- Query: $EMP \bowtie_{ENO} (\sigma_{RESP=\text{"Manager"}} (ASG))$
- Data fragments and their allocation to sites:
  - Site1 : $ASG1 = \sigma_{ENO \leq \text{"E3"}} (ASG))$
  - Site2 : $ASG2 = \sigma_{ENO > \text{"E3"}} (ASG))$
  - Site3 : $EMP1 = \sigma_{ENO \leq \text{"E3"}} (EMP))$
  - Site4 : $EMP2 = \sigma_{ENO > \text{"E3"}} (EMP))$
  - Site5 : Result
- Assumptions:
  - size(EMP) = 400, size(ASG) = 1000, size($\sigma_{RESP=\text{"Manager"}}$ (ASG)) = 20
  - tuple access cost = 1, tuple transfer cost = 10
  - EMP locally indexed on ENO, ASG locally indexed on RESP
  - uniform data distribution across sites

Query Processing in a Distributed System

1

join = 400 * 20 * 1 = 8000

selection = 1000 * 1 = 1000



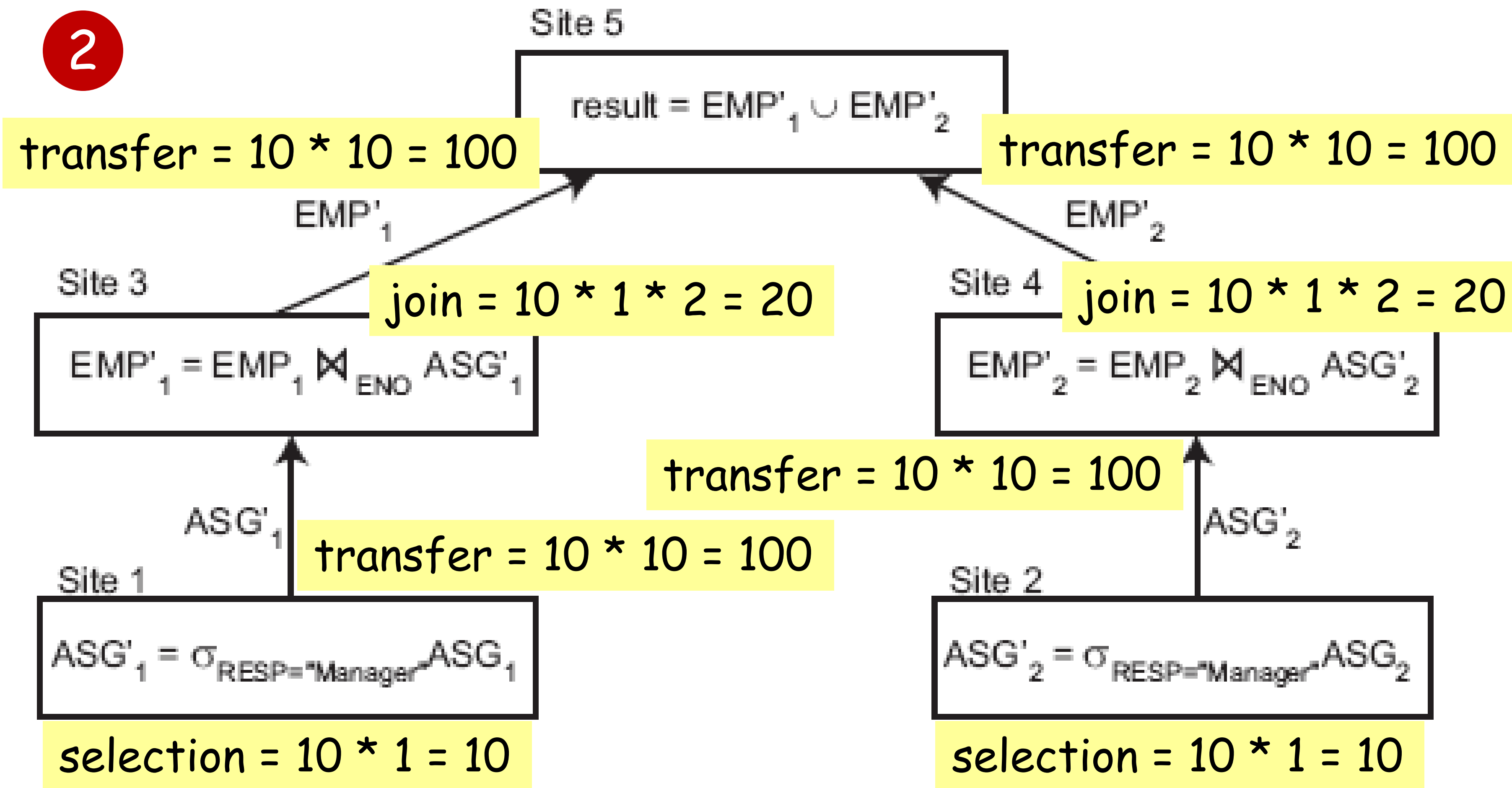
transfer = 1000 * 10 = 10000

transfer = 400 * 10 = 4000

total cost = 10000 + 4000 + 1000 + 8000 = 23000

Query Processing in a Distributed System

2



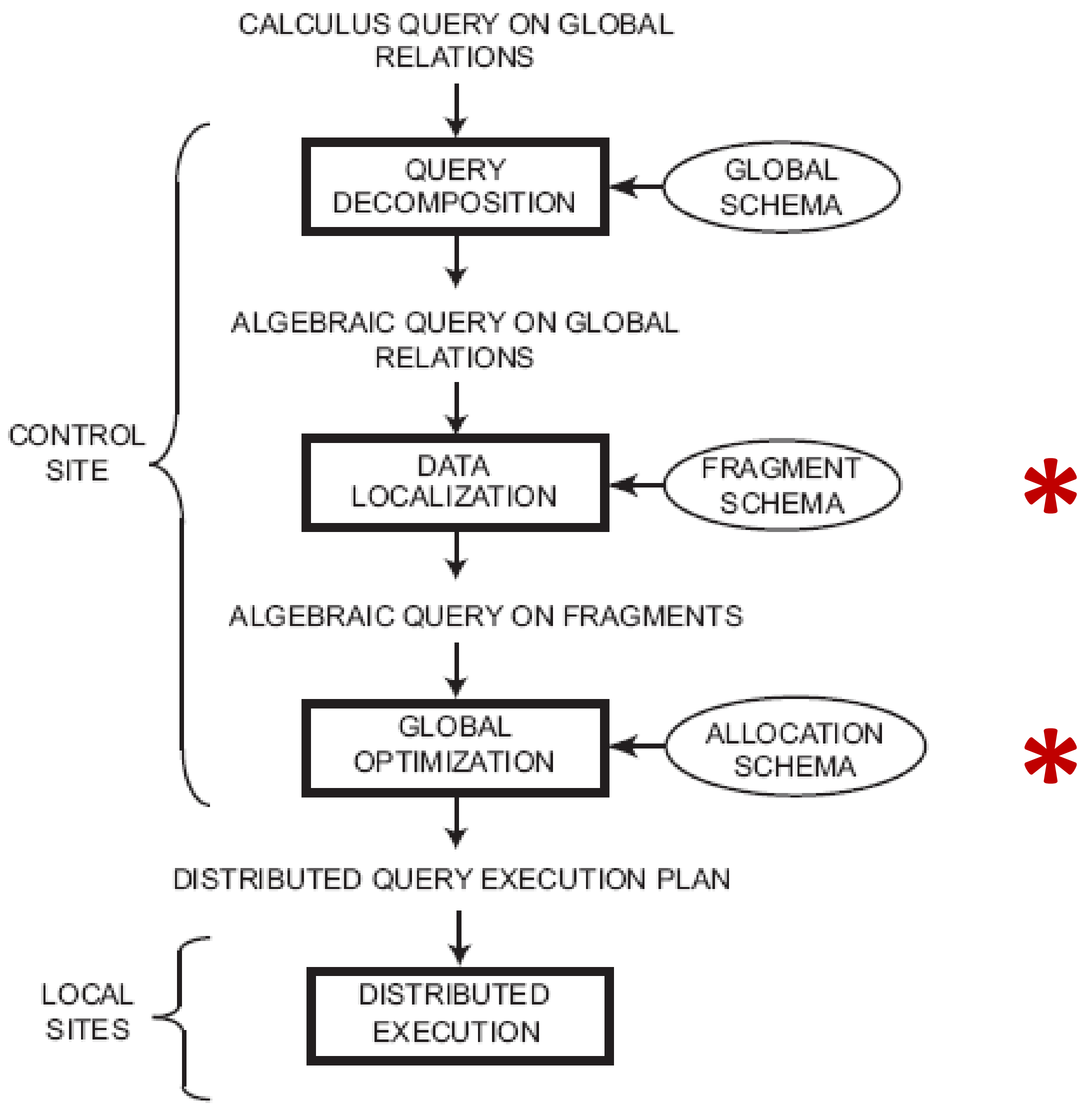
total cost = 10 + 10 + 100 + 100 + 20 + 20 + 100 + 100 = 460

General Query Optimization Issues

- Algorithmic approach:
  - Cost-based vs. Heuristics-based
- Granularity:
  - Single query at a time vs. Multi-query optimization
- Timing:
  - Static vs. Dynamic vs. Hybrid
- Statistics:
  - what to collect, accuracy, independence, uniformity
- Decision mechanism:
  - Centralized vs. Distributed vs. Hybrid
- Network topology:
  - WANs vs. LANs

Specific to distributed query processing

Distributed Query Processing



Query Decomposition

- Goal: To convert global declarative query into a correct and efficient global procedural query
- Query decomposition consists of 4 steps:
  1. Normalization
     - ➢ Transformation of query predicates into normal form
  2. Semantic Analysis
     - ➢ Detection and rejection of semantically incorrect queries
  3. Simplification
     - ➢ Elimination of redundant predicates
  4. Restructuring
     - ➢ Transformation of the query into algebraic form
- **No distribution-related processing.**

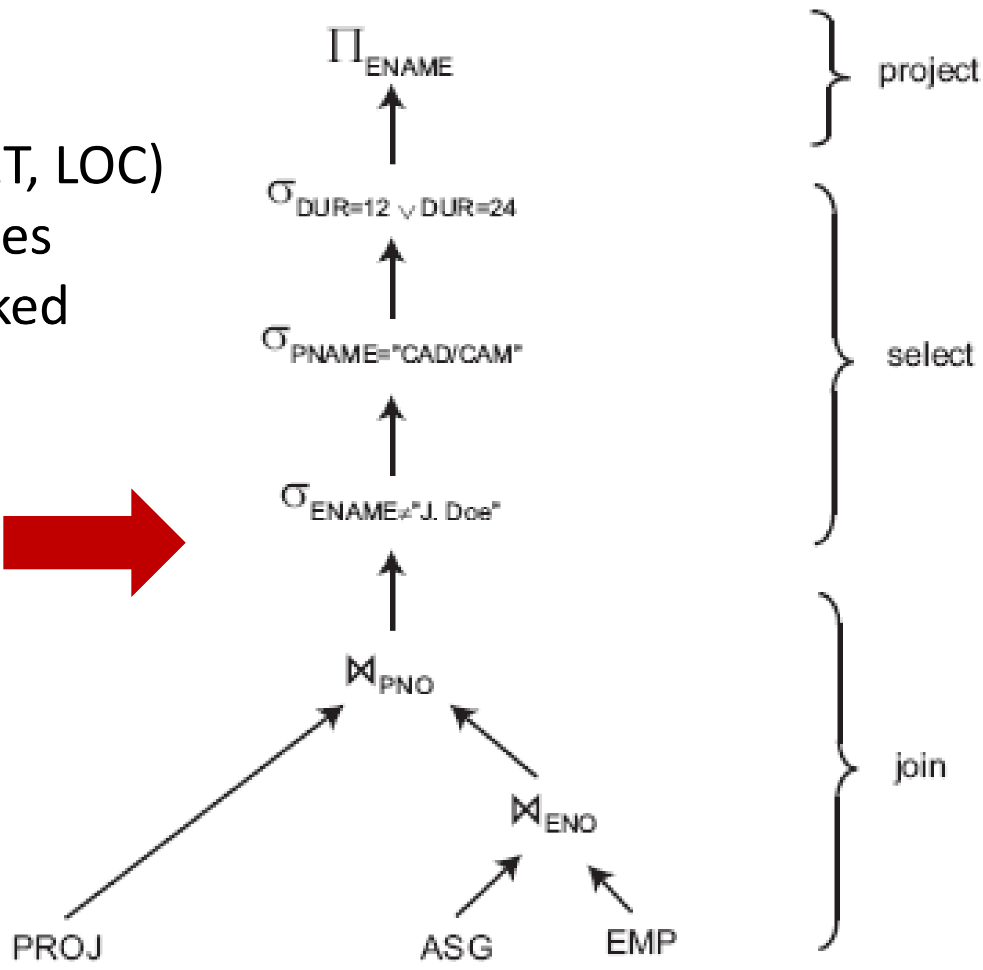
Sample Query

- Transformation of the query into algebraic form

Given: EMP(ENO, ENAME, TITLE)
ASG(ENO, PNO, RESP, DUR)
PROJ(PNO, PNAME, BUDGET, LOC)

Query: Find the names of employees other than J. Doe who worked on the CAD/CAM project for either 1 or 2 years.

```
SELECT   ENAME
FROM     EMP, ASG, PROJ
WHERE    ASG.ENO = EMP.ENO
   AND   ASG.PNO = PROJ.PNO
   AND   ENAME ≠ "J. Doe"
   AND   PROJ.PNAME = "CAD/CAM"
   AND   (DUR = 12 OR DUR = 24)
```

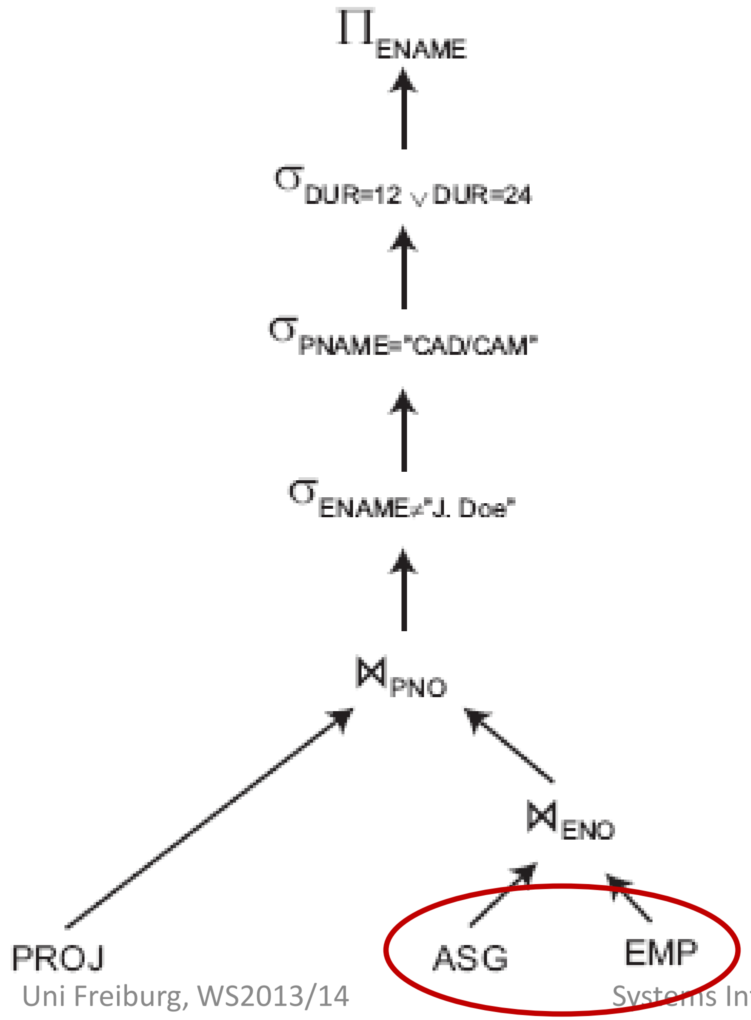


Data Localization

- Goal: To convert an algebraic query on global relations into an algebraic query on physical fragments
- General approach:
  1. Generate a localized query by substituting each global relation in the leaves of the operator tree by the appropriate subtree on fragments.
     - Union for horizontal fragments
     - Join for vertical fragments
  2. Apply reduction techniques on the localized query to generate a simpler and an optimized operator tree.

Data Localization
Example

Query plan on global relations



- EMP is fragmented as follows:

$EMP_1 = \sigma_{ENO \leq "E3"} (EMP)$

$EMP_2 = \sigma_{"E3" < ENO \leq "E6"} (EMP)$

$EMP_3 = \sigma_{ENO \geq "E6"} (EMP)$

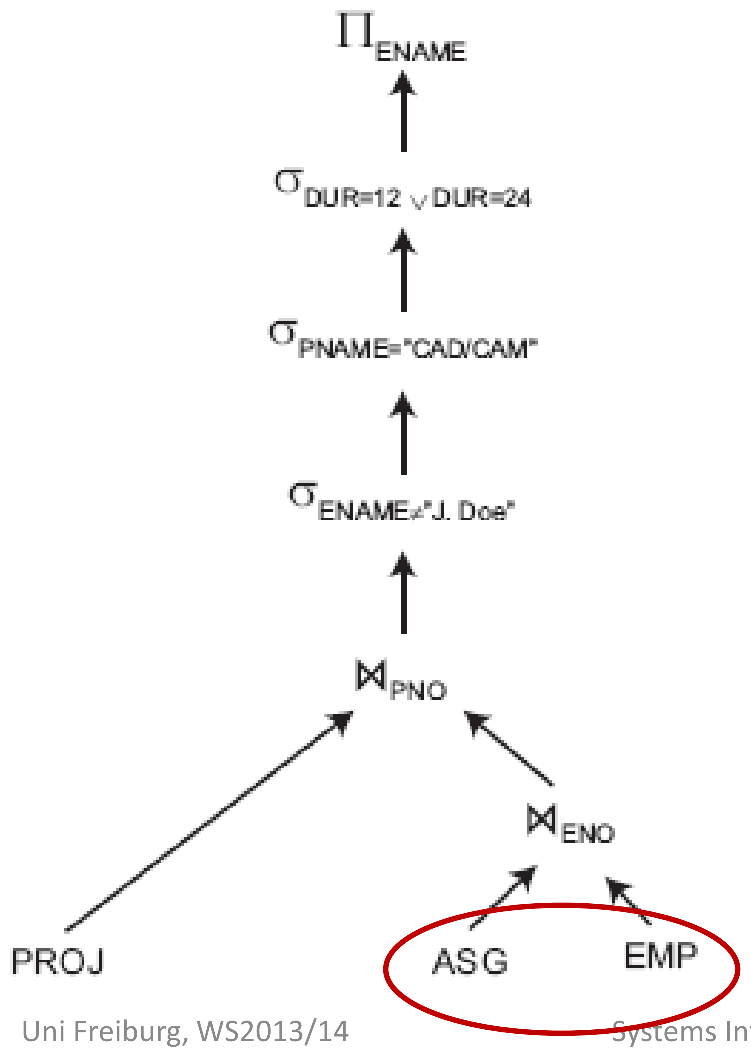
- ASG is fragmented as follows:

$ASG_1 = \sigma_{ENO \leq "E3"} (ASG)$

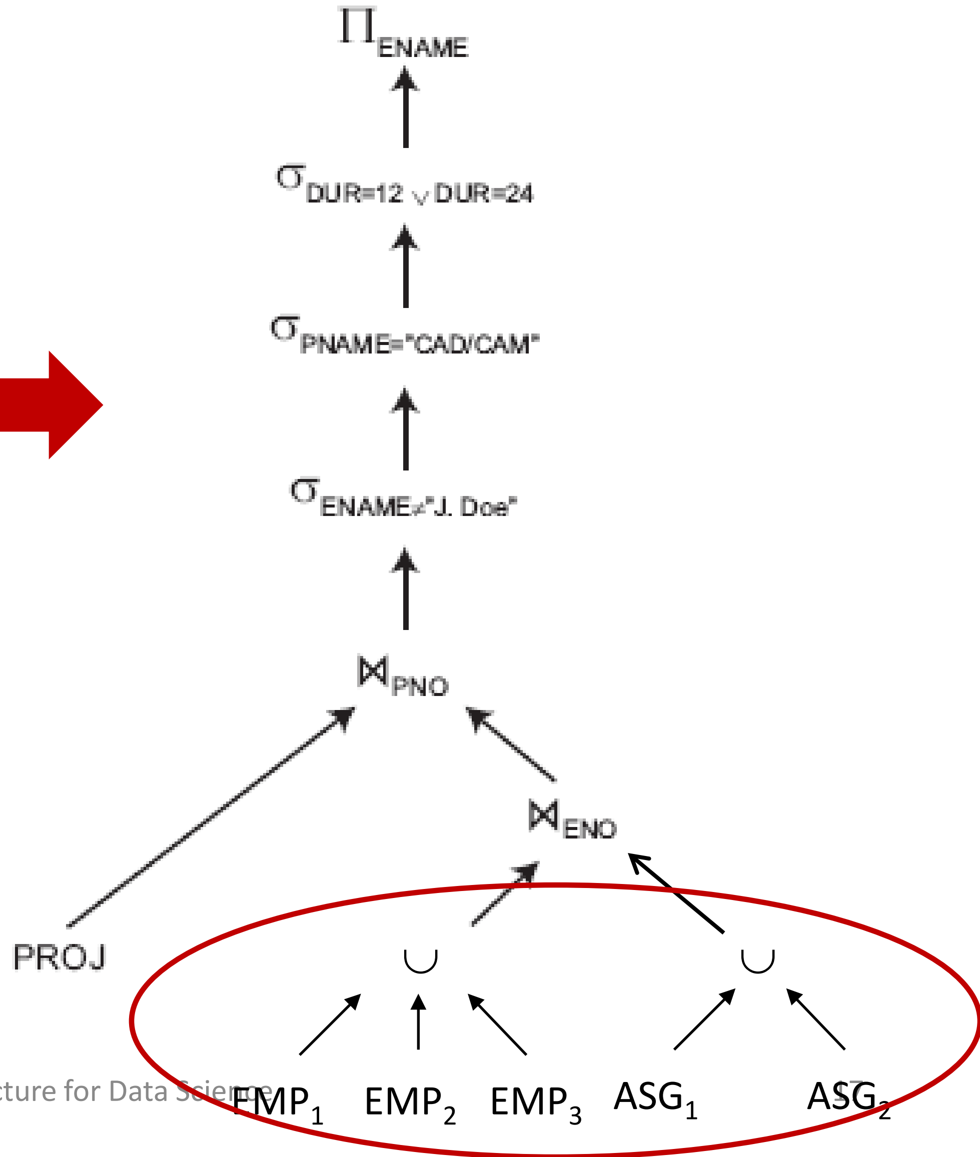
$ASG_2 = \sigma_{ENO > "E3"} (ASG)$

Data Localization Example

Query plan on global relations



Localized query plan



Data Localization

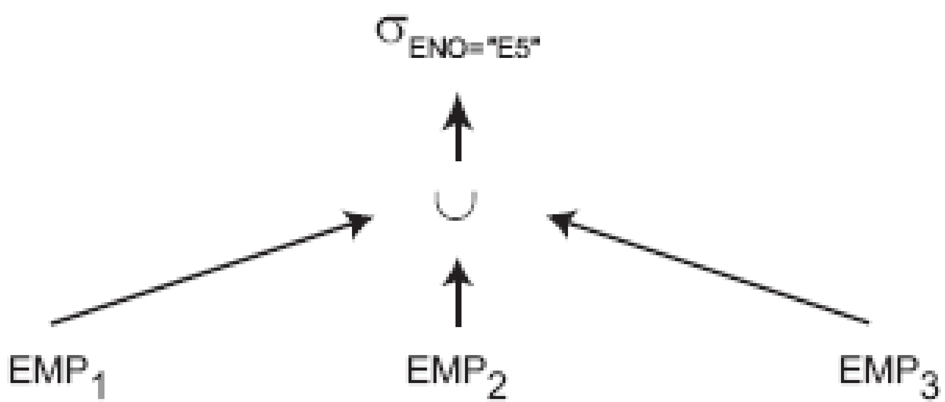
Reduction for Primary Horizontal Fragmentation

- Reduction with Selection
  - Given relation R and $F_R = \{R_1, R_2, \ldots, R_w\}$ where $R_j = \sigma_{p_j}(R)$:

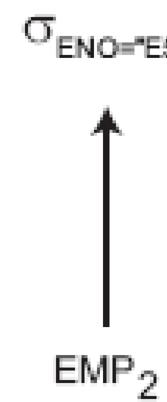
    $\sigma_{p_i}(R_j) = \phi$, if $\forall x$ in R: $\neg(p_i(x) \wedge p_j(x))$
  - Example: EMP is fragmented as before.

```
SELECT   *
FROM     EMP
WHERE    ENO = "E5"
```

Localized query



Reduced query



Data Localization

Horizontal Fragmentation

n

$\{R_1, R_2, \ldots, R_w\}$ where $R_j = \sigma_{p_j}(R)$:

$(p_i(x) \wedge p_j(x))$

ted as before.

- EMP is fragmented as follows:

$EMP_1 = \sigma_{ENO \leq \text{“E3”}}(EMP)$

$EMP_2 = \sigma_{\text{“E3”} < ENO \leq \text{“E6”}}(EMP)$

$EMP_3 = \sigma_{ENO \geq \text{“E6”}}(EMP)$

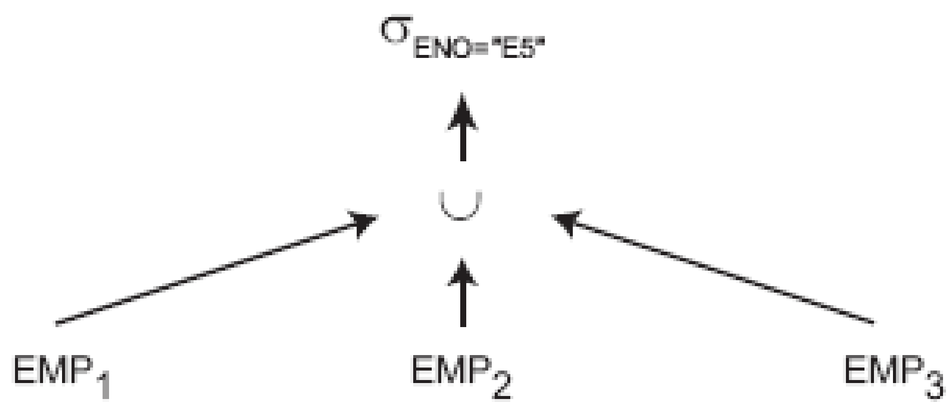
- ASG is fragmented as follows:

$ASG_1 = \sigma_{ENO \leq \text{“E3”}}(ASG)$

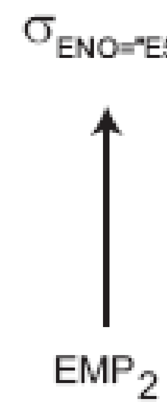
$ASG_2 = \sigma_{ENO > \text{“E3”}}(ASG)$

```
SELECT   *
FROM     EMP
WHERE    ENO = “E5”
```

Localized query



Reduced query



Data Localization

Reduction for Primary Horizontal Fragmentation

- Reduction with Join
  - Apply when fragmentation is done on the join attribute
  - Distribute Joins over Unions

    $(R_1 \cup R_2) \bowtie S \Leftrightarrow (R_1 \bowtie S) \cup (R_2 \bowtie S)$
  - Eliminate useless Joins

    $R_i \bowtie R_j = \phi$, if $\forall x$ in R_i, $\forall y$ in R_j: $\neg(p_i(x) \wedge p_j(y))$
- Example:
  - EMP and ASG are fragmented as before.

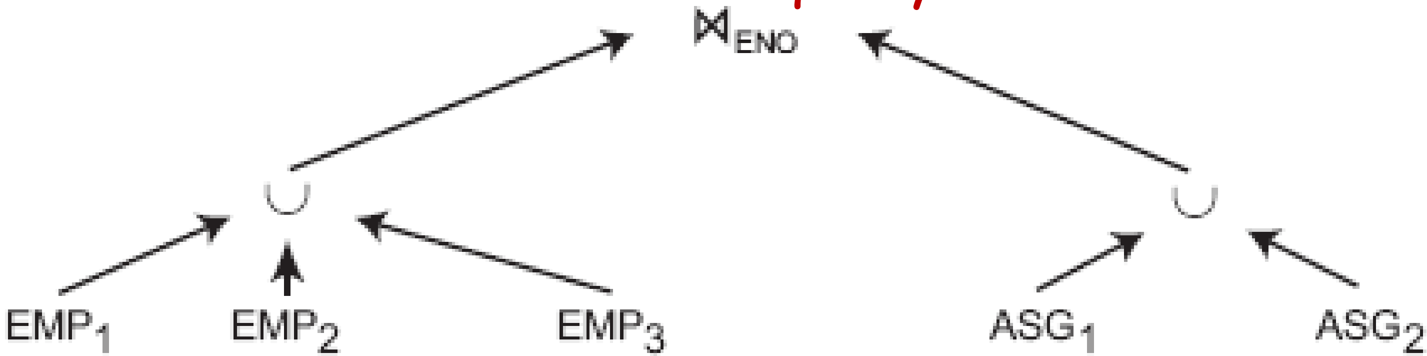
```
SELECT   *
FROM     EMP, ASG
WHERE    EMP.ENO = ASG.ENO
```

Data Localization

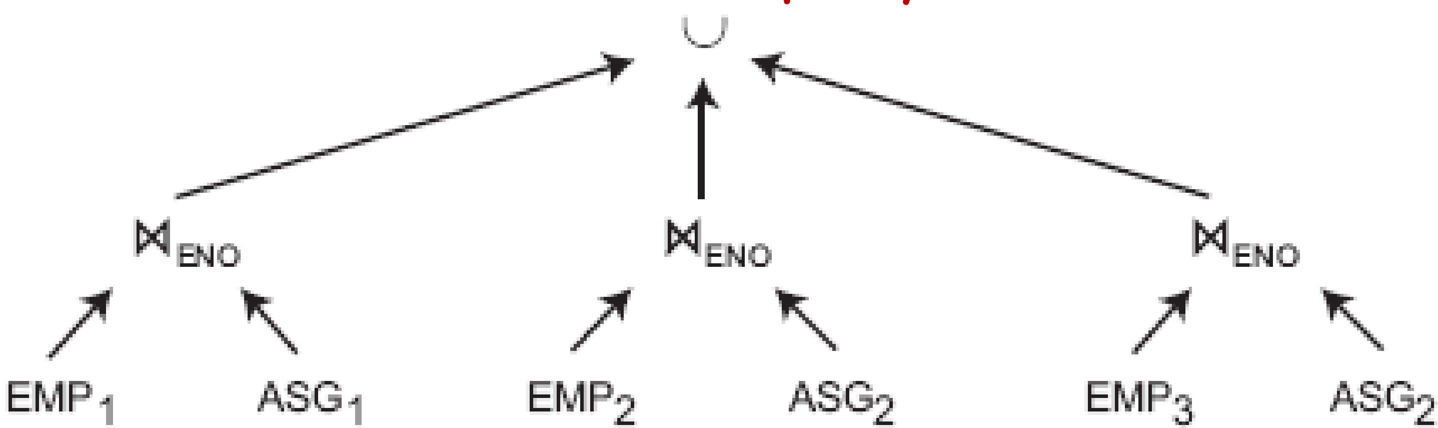
Reduction for Primary Horizontal Fragmentation

- Reduction with Join Example (cont'd):

Localized query



Reduced query

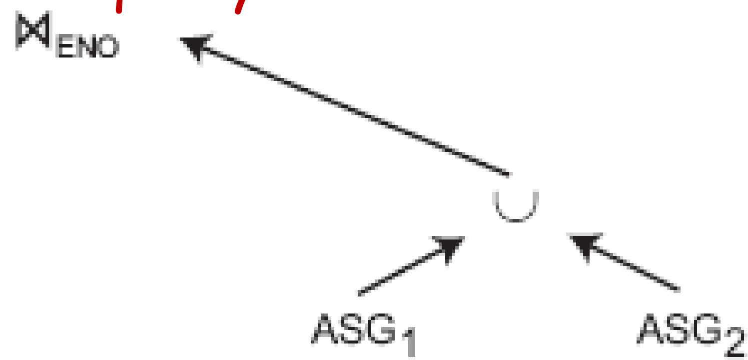


Data Localization

Horizontal Fragmentation

mple (cont'd):

ed query



- EMP is fragmented as follows:

$EMP_1 = \sigma_{ENO \leq \text{"E3"}} (EMP)$

$EMP_2 = \sigma_{\text{"E3"} < ENO \leq \text{"E6"}} (EMP)$

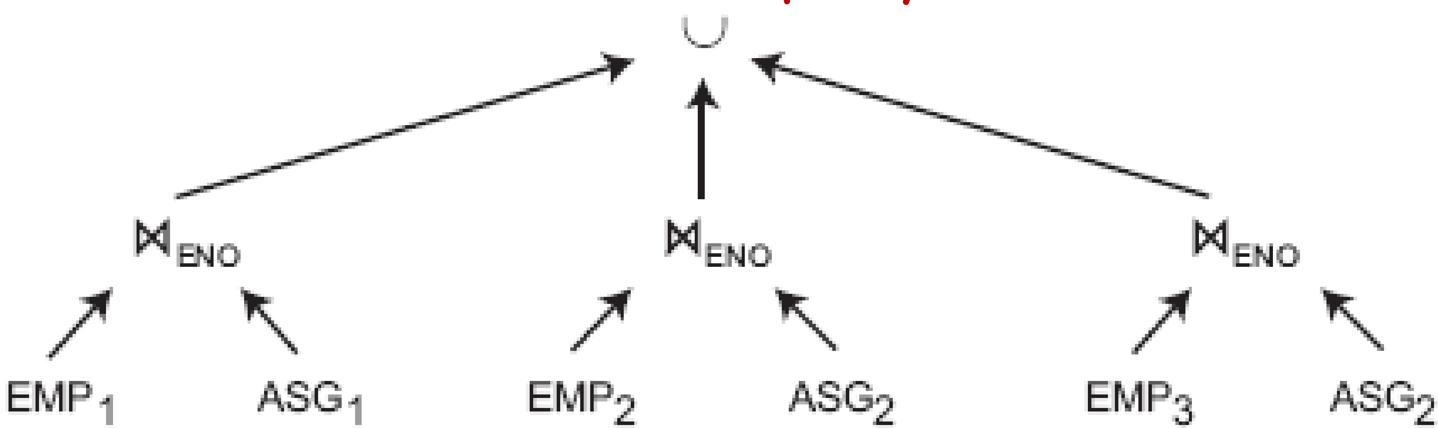
$EMP_3 = \sigma_{ENO \geq \text{"E6"}} (EMP)$

- ASG is fragmented as follows:

$ASG_1 = \sigma_{ENO \leq \text{"E3"}} (ASG)$

$ASG_2 = \sigma_{ENO > \text{"E3"}} (ASG)$

Reduced query



Data Localization
Reduction for Vertical Fragmentation

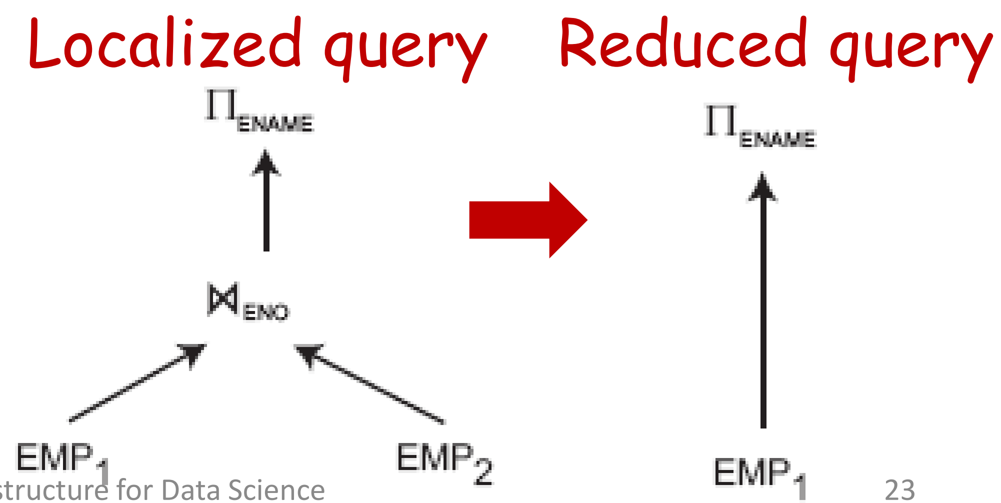
- Reduction with Projection
  - Given a relation R defined over attributes $A = \{A_1, ..., A_n\}$ and vertically fragmented as $R_i = \Pi_{A'}(R)$ where $A' \subseteq A$:

    $\Pi_{D,K}(R_i)$ is useless, if the set of projection attributes D is not in A‘.
- Example:
  - EMP is vertically fragmented as follows:

    $EMP_1 = \Pi_{ENO,ENAME}(EMP)$

    $EMP_2 = \Pi_{ENO,TITLE}(EMP)$

```
SELECT   ENAME
FROM     EMP
```



Data Localization

Reduction for Derived Horizontal Fragmentation

- Example:

ASG_1: $ASG \ltimes_{ENO} EMP_1$

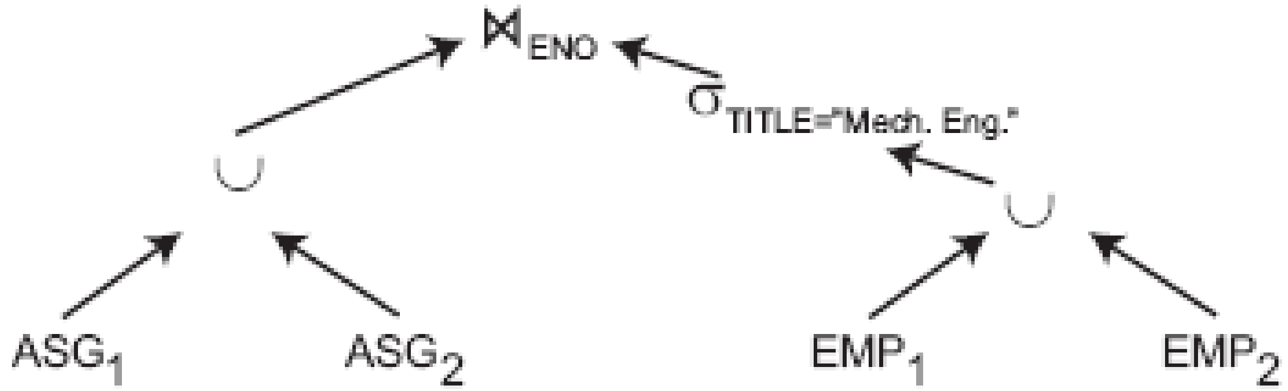
ASG_2: $ASG \ltimes_{ENO} EMP_2$

EMP_1: $\sigma_{\text{TITLE = “Programmer”}}(EMP)$

EMP_2: $\sigma_{\text{TITLE} \neq \text{“Programmer”}}(EMP)$

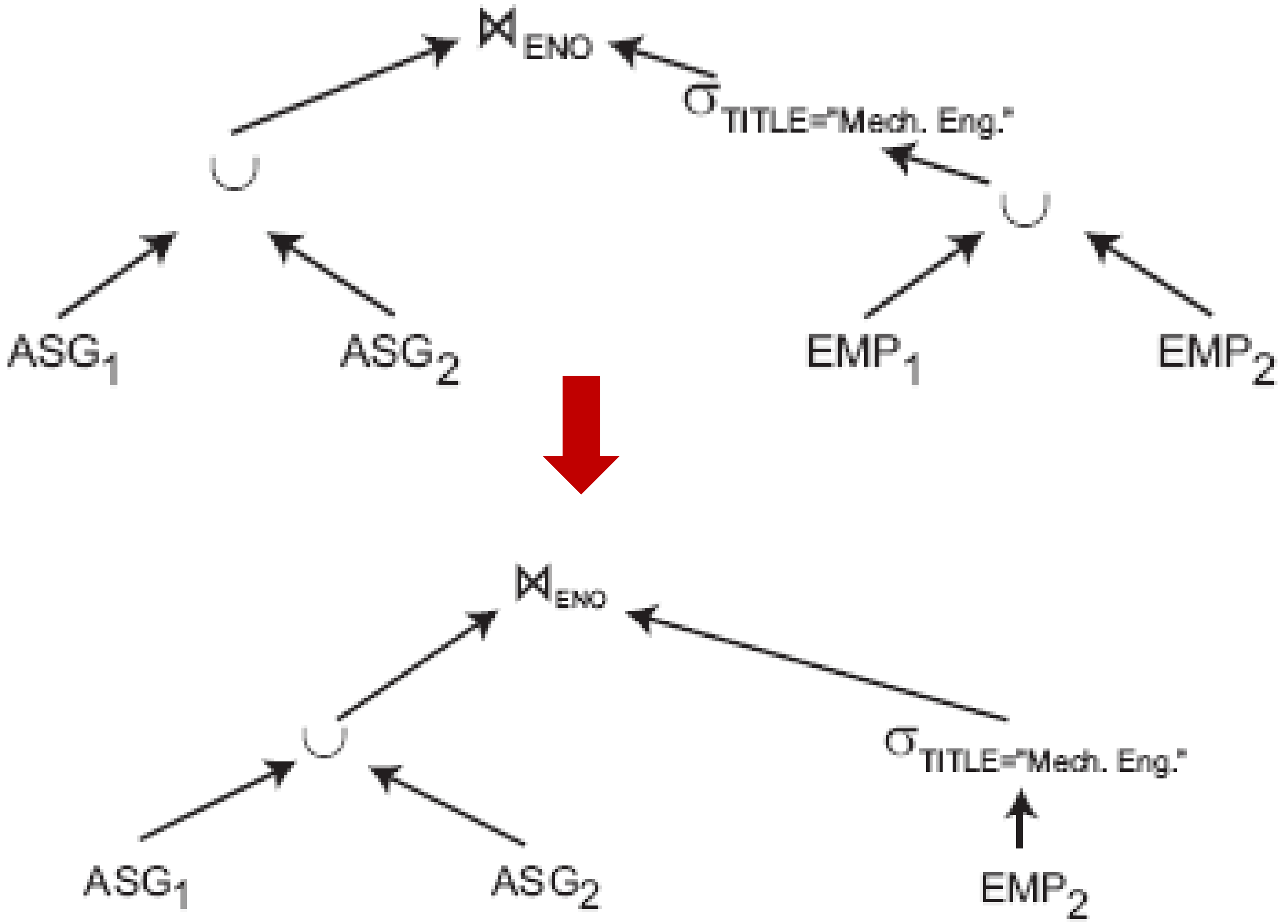
```
SELECT   *
FROM     EMP, ASG
WHERE    ASG.ENO = EMP.ENO
   AND   EMP.TITLE = “Mech. Eng.”
```

Localized query



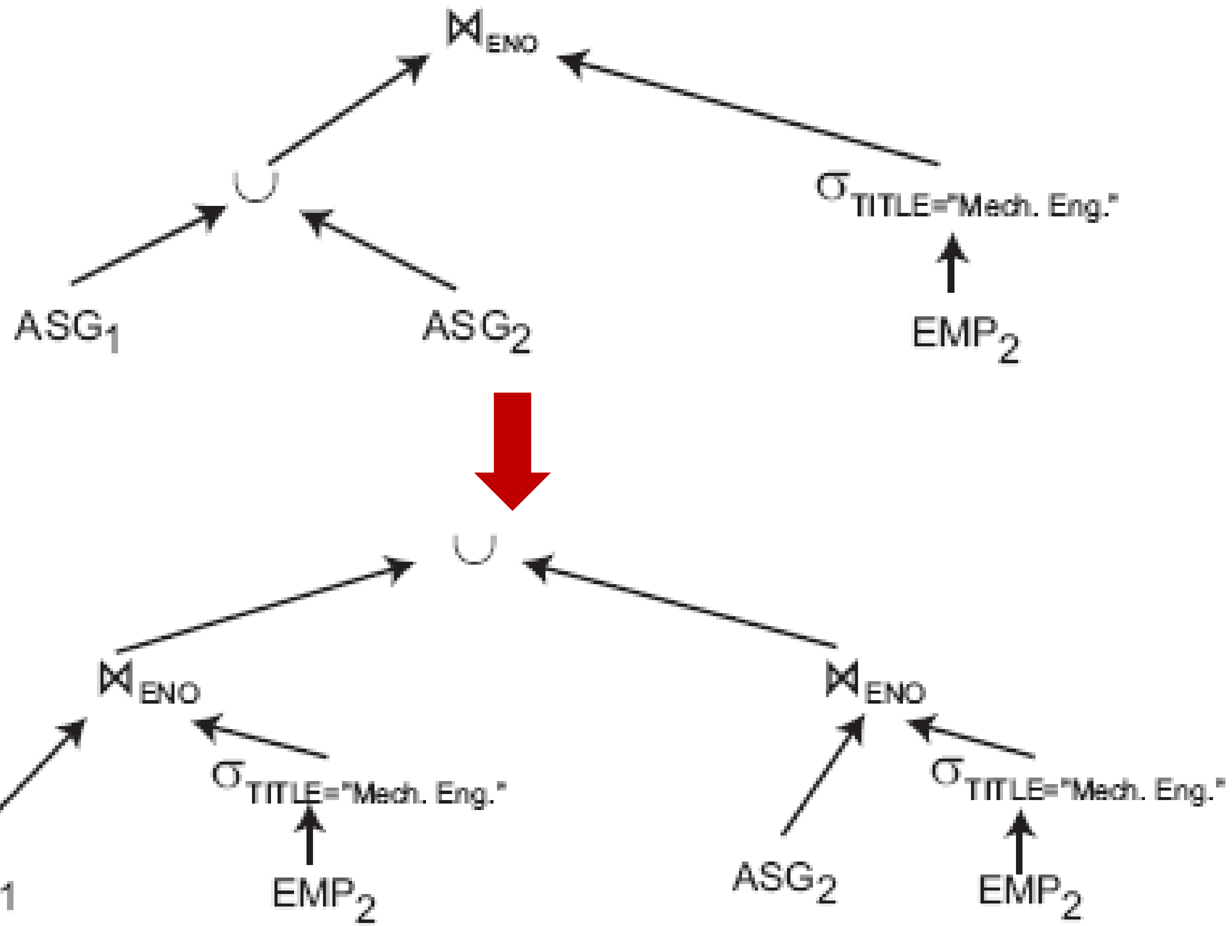
Data Localization
Reduction for Derived Horizontal Fragmentation

- Example cont'd:



Data Localization
Reduction for Derived Horizontal Fragmentation

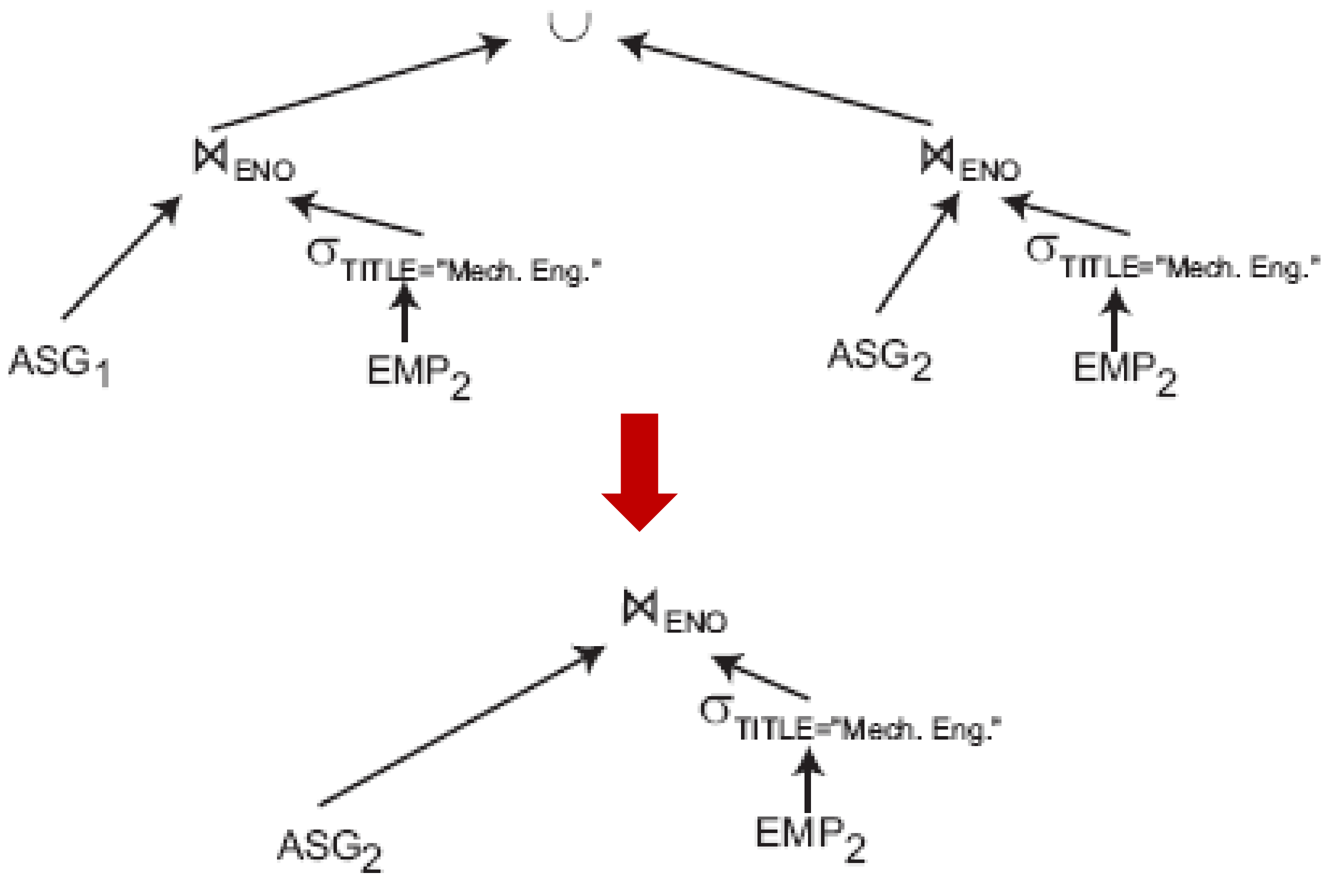
- Example cont'd:



Data Localization

Reduction for Derived Horizontal Fragmentation

- Example cont'd:



Data Localization
Reduction for Hybrid Fragmentation

- Combine all the reduction rules:
  - Remove empty relations generated by contradicting Selections on horizontal fragments.
  - Remove useless relations generated by Projections on vertical fragments.
  - Distribute Joins over Unions in order to isolate and remove useless Joins.

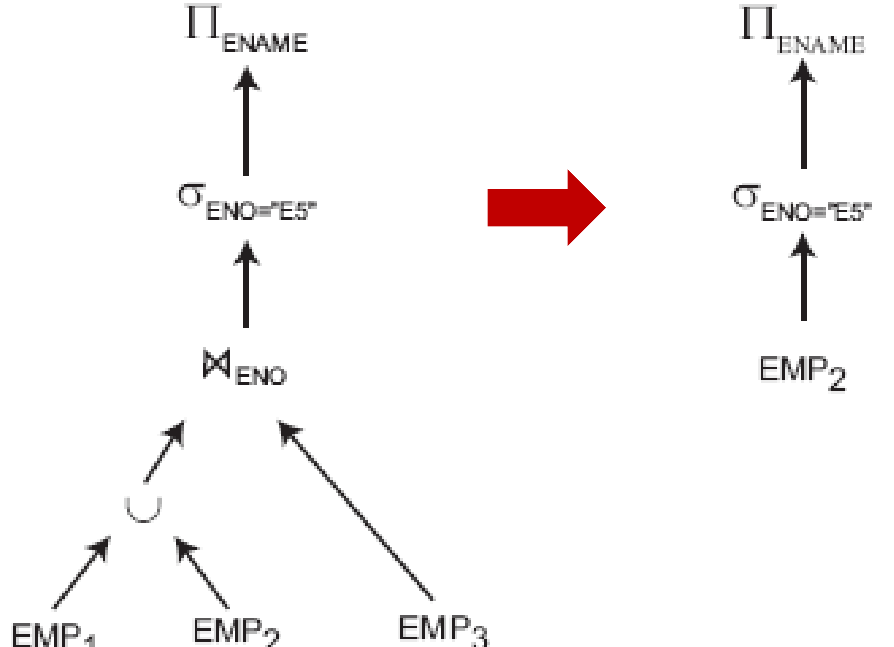
Data Localization
Reduction for Hybrid Fragmentation

- Example:

$EMP_1 = \sigma_{ENO \leq \text{"E4"}} (\Pi_{ENO,\ ENAME} (EMP))$
$EMP_2 = \sigma_{ENO > \text{"E4"}} (\Pi_{ENO,\ ENAME} (EMP))$
$EMP_3 = \Pi_{ENO,\ TITLE} (EMP)$

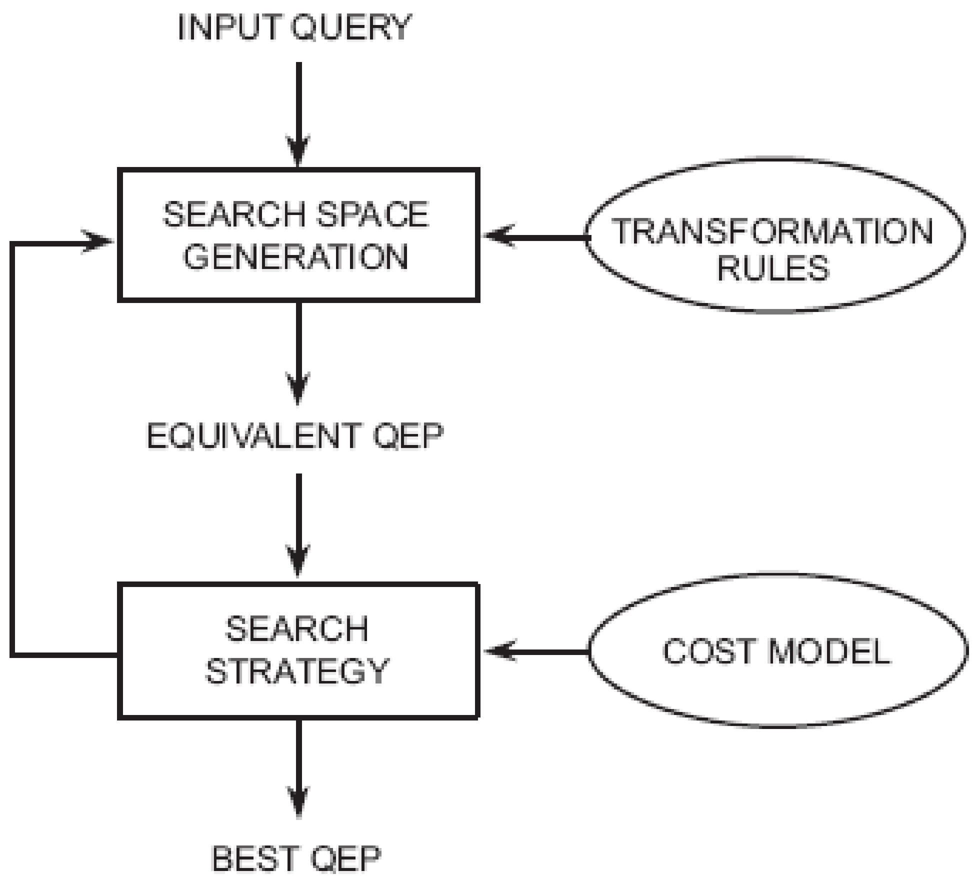
```
SELECT   ENAME
FROM     EMP
WHERE    ENO = “E5”
```

Localized query   Reduced query



Query Optimization Recap

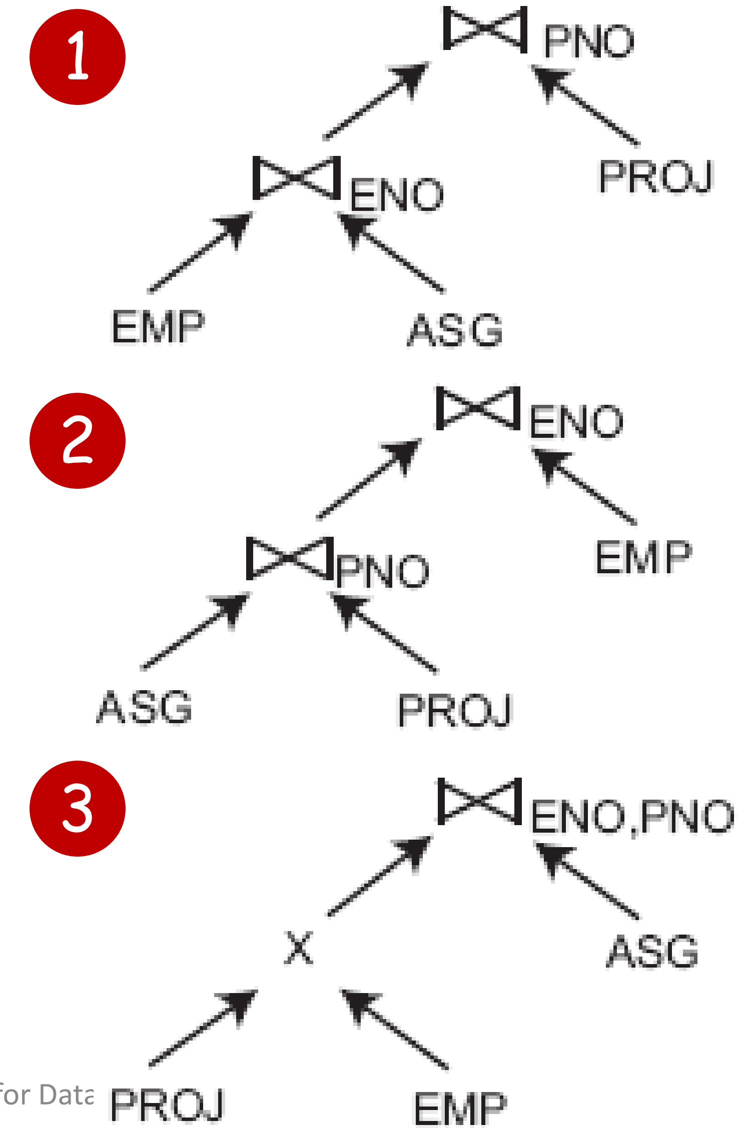
- <u>Goal:</u> To convert an algebraic query on physical fragments into an optimized query execution plan



Query Optimization
Search Space

- Search space characterized by alternative execution plans
- Focus on Join trees
- For N relations, there are O(N!) equivalent Join trees that can be obtained by applying commutativity and associativity rules
- Restrict the space w/ heuristics
- Example:

```
SELECT   ENAME, RESP
FROM     EMP, ASG, PROJ
WHERE    EMP.ENO = ASG.ENO
   AND   ASG.PNO = PROJ.PNO
```



Query Optimization
Search Strategy

- How to explore the plans in the search space
- Deterministic strategies
  - Start from base relations and build plans by adding one relation at each step
  - Dynamic programming (breadth-first approach) -> Best plan is guaranteed
  - Greedy (depth-first approach)
- Randomized strategies
  - Search for optimalities around a particular starting point
  - Trade optimization time for execution time
  - Best plan is not guaranteed
  - Simulated annealing
  - Iterative improvement

Query Optimization
Cost Model

- Cost metrics (i.e., what to optimize?)
  - Total time
  - Response time
- Database statistics (i.e., what needs to be known?)
  - Several statistics about relations, fragments, attributes need to be maintained.
  - Intermediate relation sizes/cardinalities need to be computed.
    - size(R) = cardinality(R) $*$ length(R)

Cost Model
Metrics

- Total cost = CPU cost + I/O cost + *Communication cost*

  = Unit instruction cost $*$ # of instructions

  + Unit disk I/O cost $*$ # of disk I/Os

  + *Message initiation* + *Transmission*

  - WANs: Communication cost dominates.
  - LANs: All cost are equally important.
  - To reduce total cost, cost of each component should be reduced.
- Response time is similar except that parallel components should be counted only once.
  - To reduce response time, process as many things in parallel as possible (which may actually result in higher total cost).

Centralized Query Optimization
Overview

- Static query optimization
  - Query optimization takes place at compile time, based on a cost model.
  - Example: System R [Selinger et al, IBM Almaden, 1970s]
- Dynamic query optimization
  - Query optimization and execution steps are interleaved.
  - Example: INGRES [Stonebraker et al, UC Berkeley, 1970s]
- Static-Dynamic hybrid
  - Optimized plans generated at compile time are later reoptimized at run time.

Centralized Query Optimization
System R Algorithm (Recap)

- Two main steps:
  1. For each relation R, determine the best access path.
  2. For each relation R, determine the best join ordering.
- For Joins, there are two alternative algorithms:
  1. Nested-Loop

     For each tuple of external relation R (cardinality n_1)

       For each tuple of internal relation S (cardinality n_2)

         Join two tuples if the join predicate is true

  2. Sort-Merge

     Sort R and S

     Merge R and S

System R Algorithm Example (cont'd)

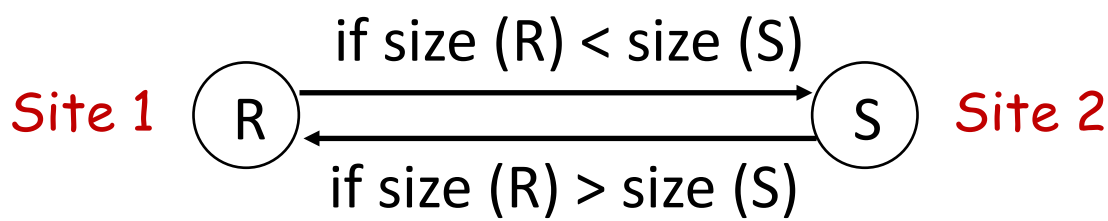
- Step 1: Determine the best access path for EMP, ASG, PROJ.
  - EMP: sequential scan (no selection)
  - ASG: sequential scan (no selection)
  - PROJ: use the index on PNAME (selection on PNAME)
- Step 2: Determine the best join ordering.
  - EMP $\bowtie$ ASG $\bowtie$ PROJ
  - ASG $\bowtie$ PROJ $\bowtie$ EMP
  - PROJ $\bowtie$ ASG $\bowtie$ EMP
  - ASG $\bowtie$ EMP $\bowtie$ PROJ
  - EMP $\times$ PROJ $\bowtie$ ASG
  - PROJ $\times$ EMP $\bowtie$ ASG

Distributed Query Optimization
Overview

- **New considerations**
  - **Join ordering in a distributed setting**
  - **Using Semijoin**
- Distributed algorithms
  - Distributed INGRES
  - Distributed System R (i.e., System R*)
  - SDD-1 based on Hill Climbing

Join Ordering in a Distributed Setting

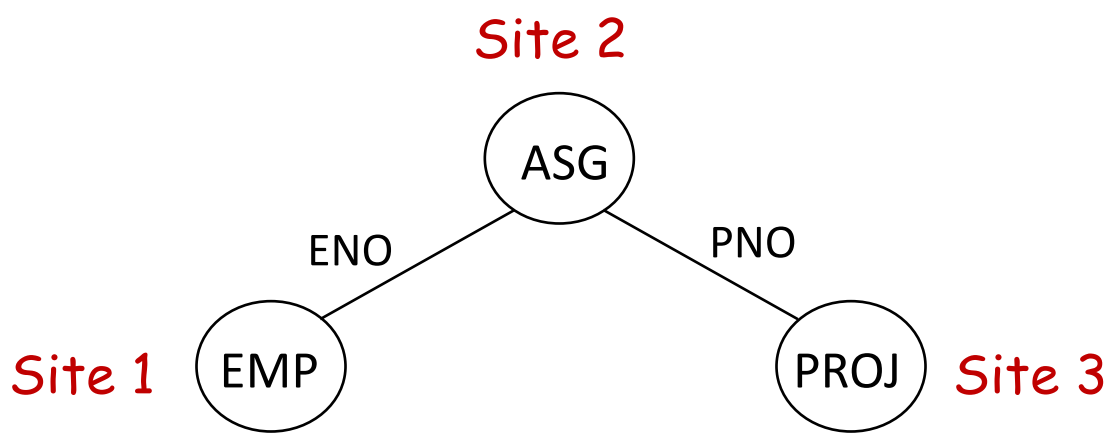
- Simplest scenario:
  - R ⋈ S, when R and S are at different sites



- When there are more than two relations, we need to worry about intermediate result sizes since these will have to be shipped between sites.

Join Ordering in a Distributed Setting
Example

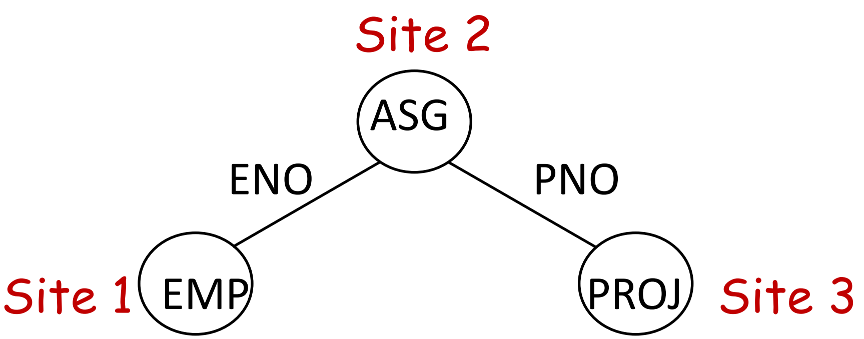
- Query:
  - PROJ $\bowtie_{PNO}$ ASG $\bowtie_{ENO}$ EMP
- Join graph:



Join Ordering in a Distributed Setting
Example (cont'd)

Alternative execution plans:

1. EMP $\rightarrow$ Site 2
   At Site 2: EMP' = EMP $\bowtie$ ASG
   EMP' $\rightarrow$ Site 3
   At Site 3: EMP' $\bowtie$ PROJ
2. ASG $\rightarrow$ Site 1
   At Site 1: EMP' = EMP $\bowtie$ ASG
   EMP' $\rightarrow$ Site 3
   At Site 3: EMP' $\bowtie$ PROJ
3. ASG $\rightarrow$ Site 3
   At Site 3: ASG' = ASG $\bowtie$ PROJ
   ASG' $\rightarrow$ Site 1
   At Site 1: ASG' $\bowtie$ EMP
4. PROJ $\rightarrow$ Site 2
   At Site 2: PROJ' = PROJ $\bowtie$ ASG
   PROJ' $\rightarrow$ Site 1
   At Site 1: PROJ' $\bowtie$ EMP
5. EMP $\rightarrow$ Site 2
   PROJ $\rightarrow$ Site 2
   At Site 2: EMP $\bowtie$ PROJ $\bowtie$ ASG



Using Semijoin

- Equivalence rules:

$$R \bowtie_A S \Leftrightarrow (R \ltimes_A S) \bowtie_A S$$

$$\Leftrightarrow R \bowtie_A (S \ltimes_A R)$$

$$\Leftrightarrow (R \ltimes_A S) \bowtie_A (S \ltimes_A R)$$

- Example: R @ Site1, S @ Site2. Assume size(R) < size(S).

1

$(R \ltimes_A S) \bowtie_A S$
At Site2: $S' = \prod_A(S)$
$S' \rightarrow$ Site 1
At Site 1: $R' = R \ltimes_A S'$
$R' \rightarrow$ Site 2
At Site 2: $R' \bowtie_A S$

2

$R \bowtie_A S$
$R \rightarrow$ Site2
At Site2: $R \bowtie_A S$

1 is better than 2 if:

$$\mathrm{size}(\Pi_A(S)) + \mathrm{size}(R \ltimes_A S') < \mathrm{size}(R)$$

Distributed Query Optimization Algorithms
A Comparative Overview

Algorithms	Opt. Timing	Objective Function	Opt. Factors	Network Topology	Semijoin	Stats	Fragments
Dist. INGRES	Dynamic	Resp. time or Total time	Msg. Size, Proc. Cost	General or Broadcast	No	1	Horizontal
R*	Static	Total time	No. Msg., Msg. Size, IO, CPU	General or Local	No	1, 2	No
SDD-1	Static	Total time	Msg. Size	General	Yes	1,3,4, 5	No

1: relation cardinality; 2: number of unique values per attribute; 3: join selectivity factor; 4: size of projection on each join attribute; 5: attribute size and tuple size

R* Algorithm
Architecture

- Master site
  - Overall coordination
  - Inter-site decisions (execution sites, fragments, data transfer methods, etc.)
- Apprentice sites
  - Local decisions (local join ordering, local access plans, etc.)

R* Algorithm
Data Transfer Alternatives

- Ship-whole
  - larger data transfer
  - smaller number of messages
  - better if relations are small
- Fetch-as-needed
  - number of messages = O(cardinality of external relation)
  - data transfer per message is minimal
  - better if relations are large and the selectivity is good

R* Algorithm
Join Strategies for R $\bowtie_A$ S

1. Move outer relation tuples to the site of the inner relation
   – Retrieve outer tuples
   – Send them to the inner relation site
   – Join them as they arrive

Total Cost = cost(retrieving qualified outer tuples)
+ # of outer tuples fetched * cost(retrieving qualified inner tuples)
+ msg. cost*(# of outer tuples fetched*avg. outer tuple size)/msg. size

R* Algorithm
Join Strategies for R $\bowtie_A$ S

2. Move inner relation to the site of outer relation
   - cannot join as they arrive; they need to be stored

Total Cost = cost(retrieving qualified outer tuples)
+ # of outer tuples fetched *
cost(retrieving matching inner tuples
from temporary storage)
+ cost(retrieving qualified inner tuples)
+ cost(storing all qualified inner tuples
in temporary storage)
+ msg. cost*(# of inner tuples fetched*avg. inner tuple size)/msg. size

R* Algorithm
Join Strategies for R $\bowtie_A$ S

3. Move both inner and outer relations to another site

Total Cost = cost(retrieving qualified outer tuples)
+ cost(retrieving qualified inner tuples)
+ cost(storing inner tuples in storage)
+ msg. cost*(# of outer tuples fetched*avg. outer tuple size)/msg. size
+ msg. cost*(# of inner tuples fetched*avg. inner tuple size)/msg. size
+ # of outer tuples fetched*cost(retrieving inner tuples from temporary storage)

R* Algorithm
Join Strategies for R $\bowtie_A$ S

4. Fetch inner tuples as needed
   - Retrieve qualified tuples at outer relation site
   - Send request containing join column value(s) for outer tuples to inner relation site
   - Retrieve matching inner tuples at inner relation site
   - Send the matching inner tuples to outer relation site
   - Join as they arrive

Total Cost = cost(retrieving qualified outer tuples)
+ msg. cost * (# of outer tuples fetched)
+ # of outer tuples fetched * (# of inner tuples fetched * avg. inner tuple size * msg. cost/msg. size)
+ # of outer tuples fetched * cost(retrieving matching inner tuples for one outer value)

Hill Climbing Algorithm

Assume join is between three relations.

Step 1: Do initial processing

Step 2: Select initial feasible solution (ES0)

- Determine the candidate result sites - sites where a relation referenced in the query exist
- Compute the cost of transferring all the other referenced relations to each candidate site
- ES0 = candidate site with minimum cost

Step 3: Determine candidate splits of ES0 into {ES1, ES2}

- ES1 consists of sending one of the relations to the other relation's site
- ES2 consists of sending the join of the relations to the final result site

Hill Climbing Algorithm (cont'd)

Step 4: Replace ES0 with the split schedule which gives

cost(ES1) + cost(local join) + cost(ES2) < cost(ES0)

Step 5: Recursively apply steps 3–4 on ES1 and ES2 until no such plans can be found

Step 6: Check for redundant transmissions in the final plan and eliminate them.

(see the example in [1])

Hill Climbing Algorithm Problems

- Greedy algorithm => determines an initial feasible solution and iteratively tries to improve it

- If there are local minima, it may not find global minima

- If the optimal schedule has a high initial cost, it won't find it, since it won't choose it as the initial feasible solution

SDD-1 Algorithm
Hill Climbing using Semijoin

Initialization

Step 1: In the execution strategy (call it ES), include all the local processing

Step 2: Reflect the effects of local processing on the database profile

Step 3: Construct a set of beneficial semijoin operations (BS) as follows :

$BS = \emptyset$

For each semijoin SJ_i

$BS \leftarrow BS \cup SJ_i$ if $cost(SJ_i) < benefit(SJ_i)$

SDD-1 Algorithm
Hill Climbing using Semijoin (cont'd)

Iterative Process

Step 4: Remove the most beneficial SJ_i from BS and append it to ES

Step 5: Modify the database profile accordingly

Step 6: Modify BS appropriately

– compute new benefit/cost values

– check if any new semijoin needs to be included in BS

Step 7: If BS ≠ Ø, go back to Step 4.

SDD-1 Algorithm
Hill Climbing using Semijoin (cont'd)

Assembly Site Selection

Step 8: Find the site where the largest amount of data resides and select it as the assembly site

Postprocessing

Step 9: For each R_i at the assembly site, find the semijoins of the type $R_i \ltimes R_j$ where the total cost of ES without this semijoin is smaller than the cost with it and remove the semijoin from ES.

Step 10: Permute the order of semijoins, if doing so would improve the total cost of ES.

(see the example in [1])

Distributed Query Processing and Optimization
Summary

- Query decomposition
  - Declarative form => Procedural form
  - Normalization, Analysis, Simplification, Restructuring
- **Data localization**
  - **Localization and reduction for different types of fragmentations**
- **Query optimization**
  - **Basic components: Search space, Search strategy, Cost model**
  - Centralized algorithms (INGRES, System R)
  - Distributed algorithms (Dist. INGRES, System R*, SDD-1)
    - **Join ordering and Semijoins**