5. Distributed Query Processing

Chapter 7

Overview of Query Processing

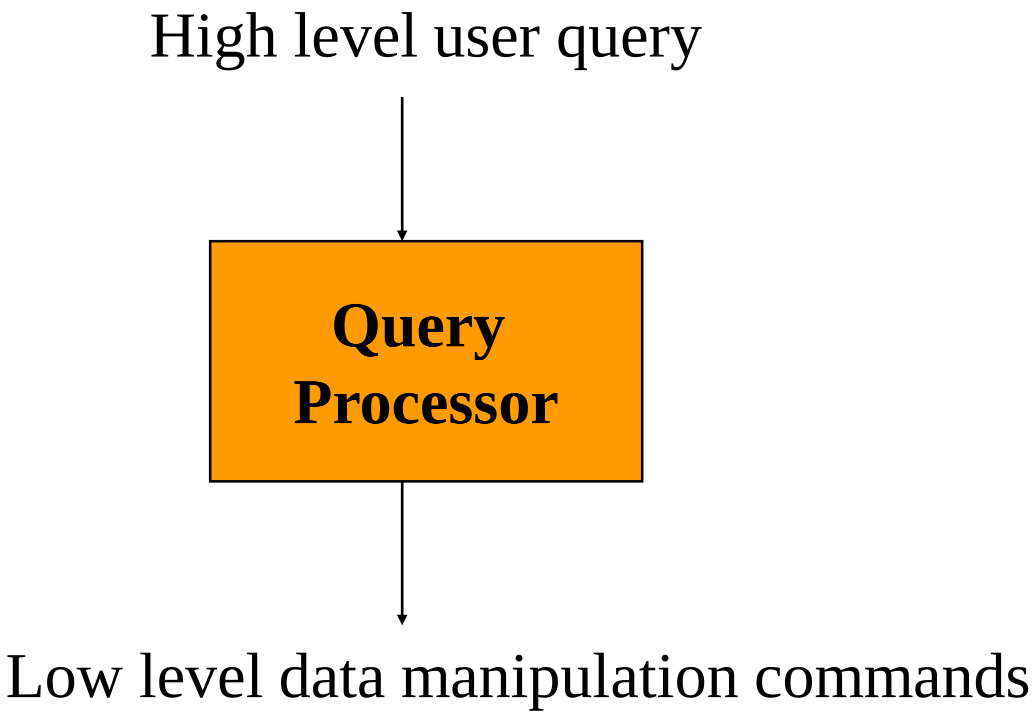
Chapter 8

Query Decomposition and Data Localization

Outline

- Overview of Query Processing (查询处理)
- Query Decomposition and Localization (查询分解与定位)

Query Processing



Query Processing Components

- Query language that is used
  - SQL (Structured Query Language)
- Query execution methodology
  - The steps that the system goes through in executing high-level (declarative) user queries
- Query optimization
  - How to determine the “best” execution plan?

Query Language – Tuple Calculus

- Tuple calculus: $\{\ t \mid F(t)\ \}$

  where t is a tuple variable, and $F(t)$ is a well formed formula

- Example:
  - *Get the numbers and names of all managers.*

    $$\left\{ t(ENO, ENAME) \mid t \in EMP \wedge t(TITLE) = "MANAGER" \right\}$$

Query Language – Domain Calculas

- Domain calculus: $\{x_1, x_2, \cdots, x_n \mid F(x_1, x_2, \cdots, x_n)\}$

  where x_i is a domain variable, and $F(x_1, x_2, \cdots, x_n)$ is a well formed formula

- Example:

  $\{ x, y \mid EMP(x, y, \text{“Manager"}) \}$

Variables are position sensitive!

Query Language SQL

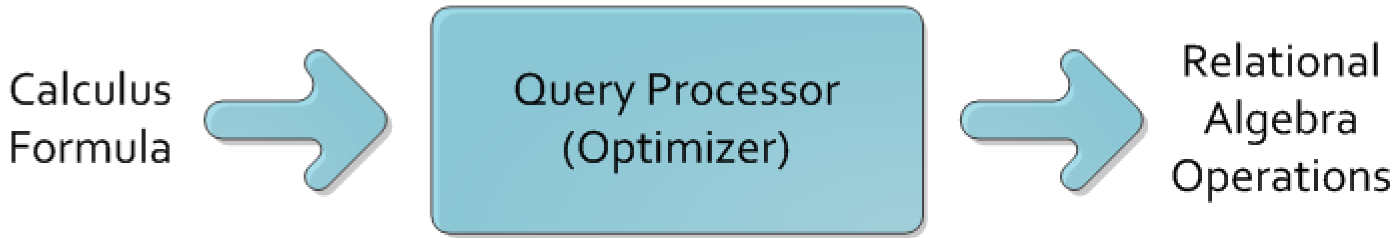
❖ SQL is a tuple calculus language.

```
SELECT ENO,ENAME
FROM   EMP
WHERE  TITLE="Programmer"
```

End user uses non-procedural (declarative) languages to express queries.

Query Processing Objectives & Problems

- Query processor transforms queries into procedural operations to access data in an optimal way.



- Distributed query processor has to deal with query decomposition and data localization.

Centralized Query Processing Alternatives

```
SELECT ENAME
FROM   EMP E, ASG G
WHERE  E.ENO=G.ENO AND TITLE="manager"
```

- Strategy 1: $\pi_{ENAME}\left(\sigma_{TITLE="manager" \wedge E.ENO=G.ENO}\left(E \times G\right)\right)$
- Strategy 2: $\pi_{ENAME}\left(E \bowtie_{ENO} \sigma_{TITLE="manager"}\left(G\right)\right)$

Which one is better?

Centralized Query Processing Alternatives (cont.)

```
SELECT ENAME
FROM   EMP E, ASG G
WHERE  E.ENO = G.ENO AND TITLE="manager"
```

- Strategy 1: $\pi_{ENAME}\left(\sigma_{TITLE=\text{"}manager\text{"} \wedge E.ENO=G.ENO}\left(E \times G\right)\right)$

- Strategy 2: $\pi_{ENAME}\left(E \bowtie_{ENO} \sigma_{TITLE=\text{"}manager\text{"}}\left(G\right)\right)$

Strategy 2 avoids Cartesian product, so is “better”.

Distributed Query Processing

- Query processor must consider the communication cost and select the best site.
- The same query example, but relation G and E are fragmented and distributed.

$\sigma_{ENO \le "E3"}(G)$ G1

Site1

$\sigma_{ENO > "E3"}(G)$ G2

Site2

Query

Site5

$\sigma_{ENO \le "E3"}(E)$ E1

Site3

$\sigma_{ENO > "E3"}(E)$ E2

Site4

Distributed Query Processing Plans

- By centralized optimization,

$$\pi_{ENAME}\left(E \bowtie_{ENO} \sigma_{TITLE="manager"}(G)\right)$$

- Two distributed query processing plans

Distributed Query Plan I

Plan I: To transport all segments to query site 5 and execute there.

Site 5

$$\textbf{Result} = (EMP_1 \cup EMP_2) \bowtie_{ENO} \sigma_{TITLE=\text{"manager"}} (ASG_1 \cup ASG_2)$$

ASG_1 — Site 1

ASG_2 — Site 2

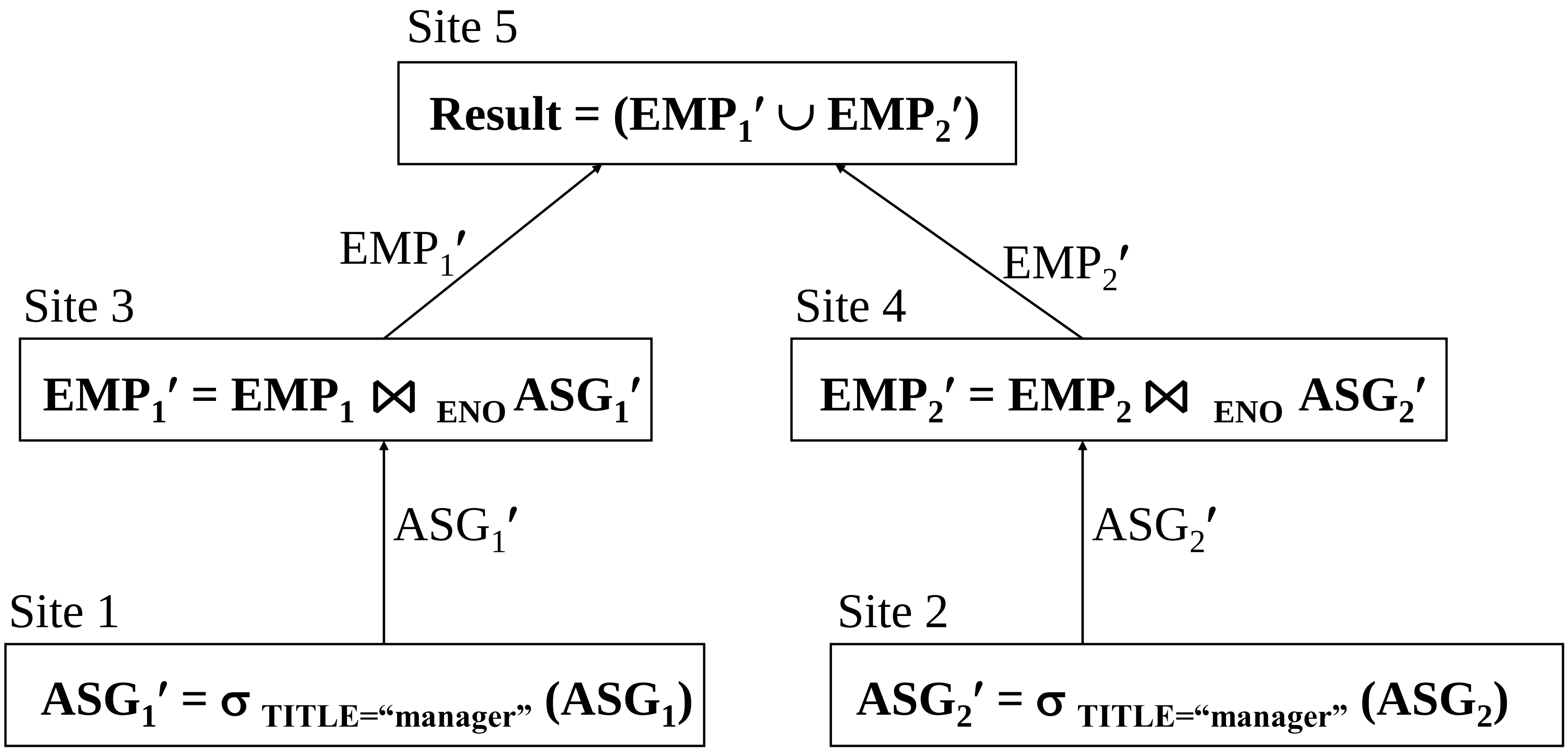
EMP_1 — Site 3

EMP_2 — Site 4

This causes too much network traffic, very costly.

Distributed Query Plan II

Plan II (Optimized): $\pi_{ENAME}\left(E \bowtie_{ENO} \sigma_{TITLE="manager"}(G)\right)$



Costs of the Two Plans

$$\pi_{ENAME}\left(E \bowtie_{ENO} \sigma_{TITLE="manager"}(G)\right)$$

- Assume
  - *size*(EMP)=400, *size*(ASG)=1000, 20 tuples with TITLE="manager"
  - tuple access cost = 1 unit; tuple transfer cost = 10 units
  - ASG and EMP are locally clustered on attribute TITLE and ENO, respectively.
- Plan 1

Transfer EMP to site 5: 400*tuple transfer cost	4000
Transfer ASG to site 5: 1000*tuple transfer cost	10000
Produce ASG′ : 1000*tuple access cost	1000
Join EMP and ASG′ : 400*20*tuple access cost	8000
Total cost	23,000

- Plan 2

Produce ASG′ : (10+10)*tuple access cost	20
Transfer ASG′ to the sites of EMP: (10+10)*tuple transfer cost	200
Produce EMP′ : (10+10)*tuple access cost	20
Transfer EMP′ to result site: (10+10)*tuple transfer cost	200
Total cost	440

Query Optimization Objectives

- Minimize a cost function

  I/O cost + CPU cost + communication cost

  - These might have different weights in different distributed environments

- Can also maximize throughout

Communication Cost

- Wide area network
  - Communication cost will dominate
    - Low bandwidth
    - Low speed
    - High protocol overhead
  - Most algorithms ignore all other cost components

- Local area network
  - Communication cost not that dominate
  - Total cost function should be considered

Complexity of Relational Algebra Operations

- Measured by cardinality ***n*** and tuples are sorted on comparison attributes

Operation	Complexity
σ, π (without duplicate elimination)	$O(n)$
π (with duplicate elimination), GROUP	$O(n\log n)$
$Join, Semijoin, Division, \cap, \cup, -$	$O(n\log n)$
Cartesian-Product X	$O(n^2)$

Types of Query Optimization

- Exhaustive search
  - Cost-based
  - Optimal
  - Combinatorial complexity in the number of relations
  - Workable for small solution spaces
- Heuristics
  - Not optimal
  - Re-group common sub-expressions
  - Perform selection and projection (σ, π) first
  - Replace a join by a series of semijoins
  - Reorder operations to reduce intermediate relation size
  - Optimize individual operations

Query Optimization Granularity

- Single query at a time
  - Cannot use common intermediate results
- Multiple queries at a time
  - Efficient if many similar queries
  - Decision space is much larger

Query Optimization Timing

- Static
  - Do it at compilation time by using statistics, appropriate for exhaustive search, optimized once, but executed many times.
  - Difficult to estimate the size of the intermediate results
  - Can amortize over many executions

- Dynamic
  - Do it at execution time, accurate about the size of the intermediate results, repeated for every execution, expensive.

Query Optimization Timing (*cont.*)

- Hybrid
  - Compile using a static algorithm
  - If the error in estimate size > threshold, re-optimize at run time

Statistics

- Relation
  - Cardinality
  - Size of a tuple
  - Fraction of tuples participating in a join with another relation
- Attributes
  - Cardinality of the domain
  - Actual number of distinct values
- Common assumptions
  - Independence between different attribute values
  - Uniform distribution of attribute values within their domain

Decision Sites

- For query optimization, it may be done by
  - **Single site** – *centralized* approach
    - Single site determines the best schedule
    - Simple
    - Need knowledge about the entire distributed database
  - **All the sites involved** – *distributed* approach
    - Cooperation among sites to determine the schedule
    - Need only local information
    - Cost of operation
  - **Hybrid** – one site makes major decision in cooperation with other sites making local decisions
    - One site determines the global schedule
    - Each site optimizes the local subqueries

Network Topology

- Wide Area Network (WAN) – point-to-point
  - Characteristics
    - Low bandwidth
    - Low speed
    - High protocol overhead
  - Communication cost will dominate; ignore all other cost factors
  - Global schedule to minimize communication cost
  - Local schedules according to centralized query optimization

Network Topology (*cont.*)

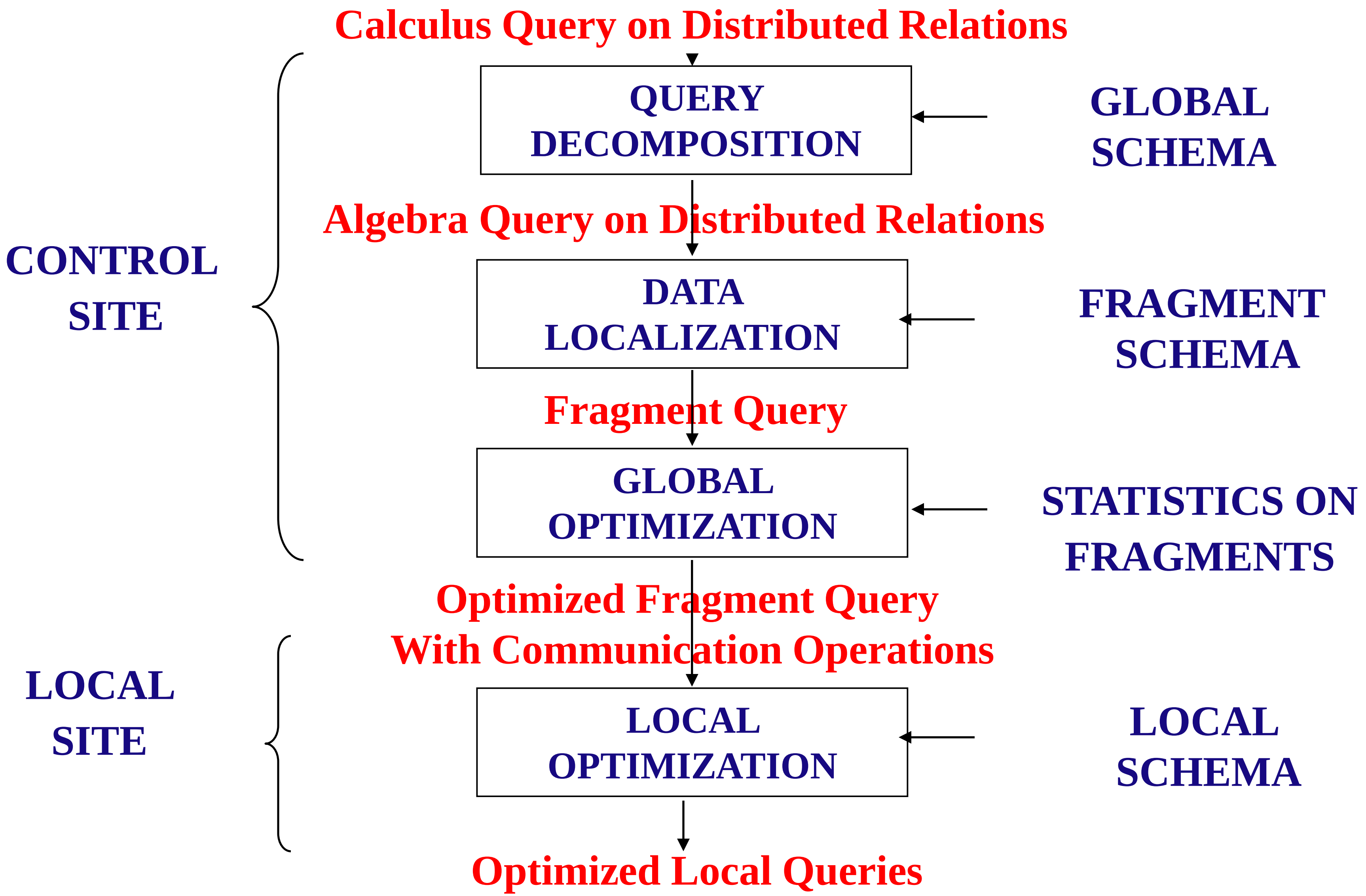
- Local Area Network (LAN)
  - Communication cost not that dominate
  - Total cost function should be considered
  - Broadcasting can be exploited
  - Special algorithms exist for star networks

Other Information to Exploit

- Using replications to minimize communication costs

- Using semijoins to reduce the size of operand relations to cut down communication costs when overhead is not significant.

Layers of Query Processing



Step 1 - Query Decomposition

- Decompose calculus query into algebra query using global conceptual schema information.

(1) normalization

(2) analysis

(3) elimination of redundancy

(4) rewriting

Step 1 - Query Decomposition (*cont.*)

1) Normalization

- The calculus query is written in a normalized form (CNF or DNF) for subsequent manipulation

2) Analysis

- To reject normalized queries for which further processing is either impossible or unnecessary (type incorrect or semantically incorrect)

3) Simplification (elimination of redundancy)

- Redundant predicates are eliminated to obtain simplified queries

4) Rewriting

- The calculus query is translated to optimal algebraic query representation
- More than one translation is possible

1) Normalization

- Lexical and syntactic analysis
  - check validity (similar to compilers)
  - check for attributes and relations
  - type checking on the qualification
- There are two possible forms of representing the predicates in query qualification
  - Conjunctive Normal Form (CNF) or Disjunctive Normal Form (DNF)
    - CNF: $(p_{11} \vee p_{12} \vee \ldots \vee p_{1n}) \wedge \ldots \wedge (p_{m1} \vee p_{m2} \vee \ldots \vee p_{mn})$
    - DNF: $(p_{11} \wedge p_{12} \wedge \ldots \wedge p_{1n}) \vee \ldots \vee (p_{m1} \wedge p_{m2} \wedge \ldots \wedge p_{mn})$
    - OR's mapped into union
    - AND's mapped into join or selection

1) Normalization (*cont.*)

- The transformation of the quantifier-free predicate is straightforward using the well-known equivalence rules for logical operations ($\wedge \vee \neg$)

$$P_1 \wedge P_2 \Leftrightarrow P_2 \wedge P_1$$

$$P_1 \vee P_2 \Leftrightarrow P_2 \vee P_1$$

$$P_1 \wedge (P_2 \wedge P_3) \Leftrightarrow (P_1 \wedge P_2) \wedge P_3$$

$$P_1 \vee (P_2 \vee P_3) \Leftrightarrow (P_1 \vee P_2) \vee P_3$$

$$P_1 \wedge (P_2 \wedge P_3) \Leftrightarrow (P_1 \wedge P_2) \wedge P_3$$

$$P_1 \vee (P_2 \wedge P_3) \Leftrightarrow (P_1 \vee P_2) \wedge (P_1 \vee P_3)$$

$$P_1 \wedge (P_2 \vee P_3) \Leftrightarrow (P_1 \wedge P_2) \vee (P_1 \wedge P_3)$$

$$\neg(P_1 \wedge P_2) \Leftrightarrow \neg P_1 \vee \neg P_2$$

$$\neg(P_1 \vee P_2) \Leftrightarrow \neg P_1 \wedge \neg P_2$$

$$\neg(\neg P_1) \Leftrightarrow P_1$$

1) Normalization (*cont.*)

- Example

```
SELECT   ENAME
FROM     EMP, ASG
WHERE    EMP.ENO=ASG.ENO AND ASG.JNO="J1"
         AND (DUR=12 OR DUR=24)
```

- The conjunctive normal form:

$$EMP.ENO = ASG.ENO$$
$$\wedge ASG.JNO = "J1"$$
$$\wedge (DUR = 12 \vee DUR = 24)$$

2) Analysis

- Objective
  - reject type incorrect or semantically incorrect queries
- Type incorrect
  - if any of its attribute or relation names is not defined in the global schema
  - if operations are applied to attributes of the wrong type

2) Analysis (*cont.*)

- Type incorrect example

```
SELECT  E#
FROM    EMP
WHERE   ENAME>200
```

! Undefined attribute

! Type mismatch

2) Analysis (*cont.*)

- Semantically incorrect
  - Components do not contribute in any way to the generation of the result
  - For only those queries that do not use disjunction ($\vee$) or negation ($\neg$), semantic correctness can be determined by using *query graph*

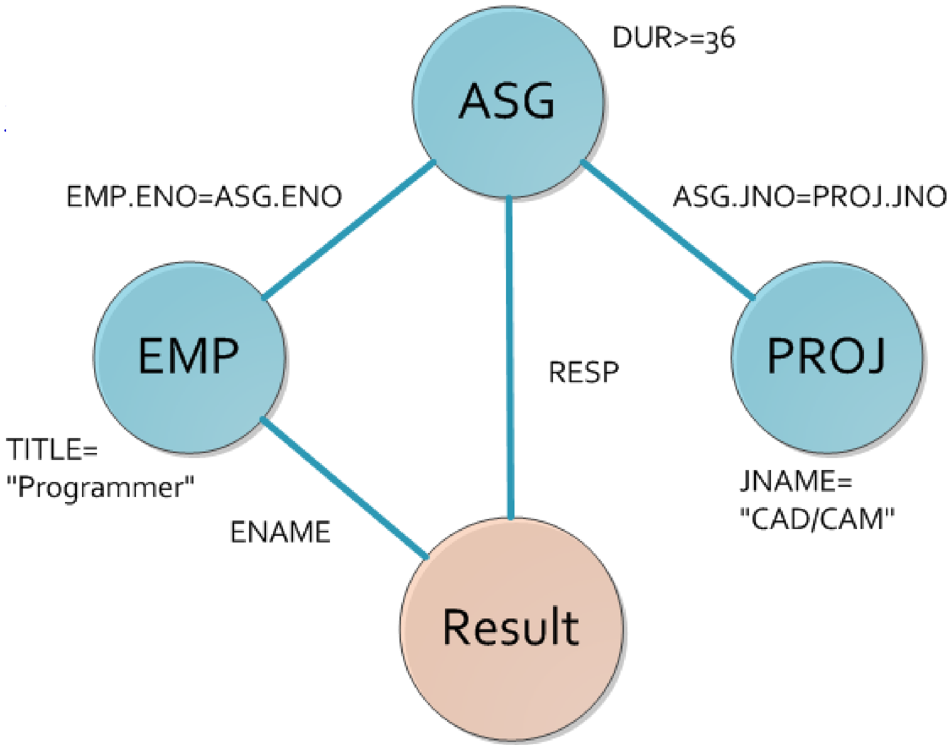
Query Graph

- Two kinds of nodes
  - One node represents the result relation
  - Other nodes represent operand relations
- Two types of edges
  - an edge to represent a join if neither of its two nodes is the result
  - an edge to represent a projection if one of its node is the result node

Nodes and edges may be labeled by predicates for selection, projection, or join.

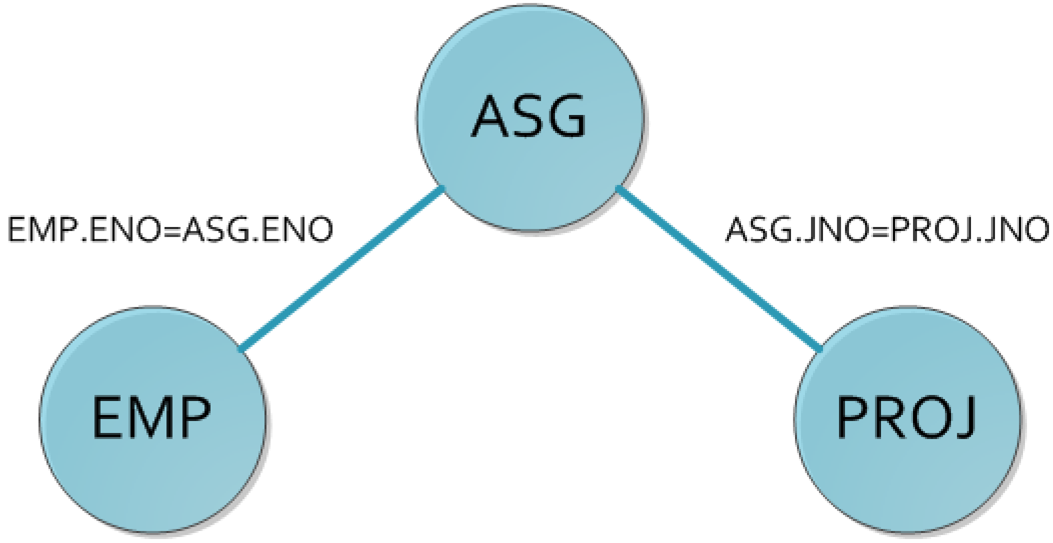
Query Graph Example

```
SELECT   ENAME, RESP
FROM     EMP,ASG,PROJ
WHERE    EMP.ENO=ASG.GNO
         AND ASG.PNO=PROJ.PNO AND PNAME="CAD/CAM"
         AND  DUR>36
         AND  TITLE="Prog
```



Join Graph Example 1

A subgraph of query graph for join operation.



Tool of Analysis

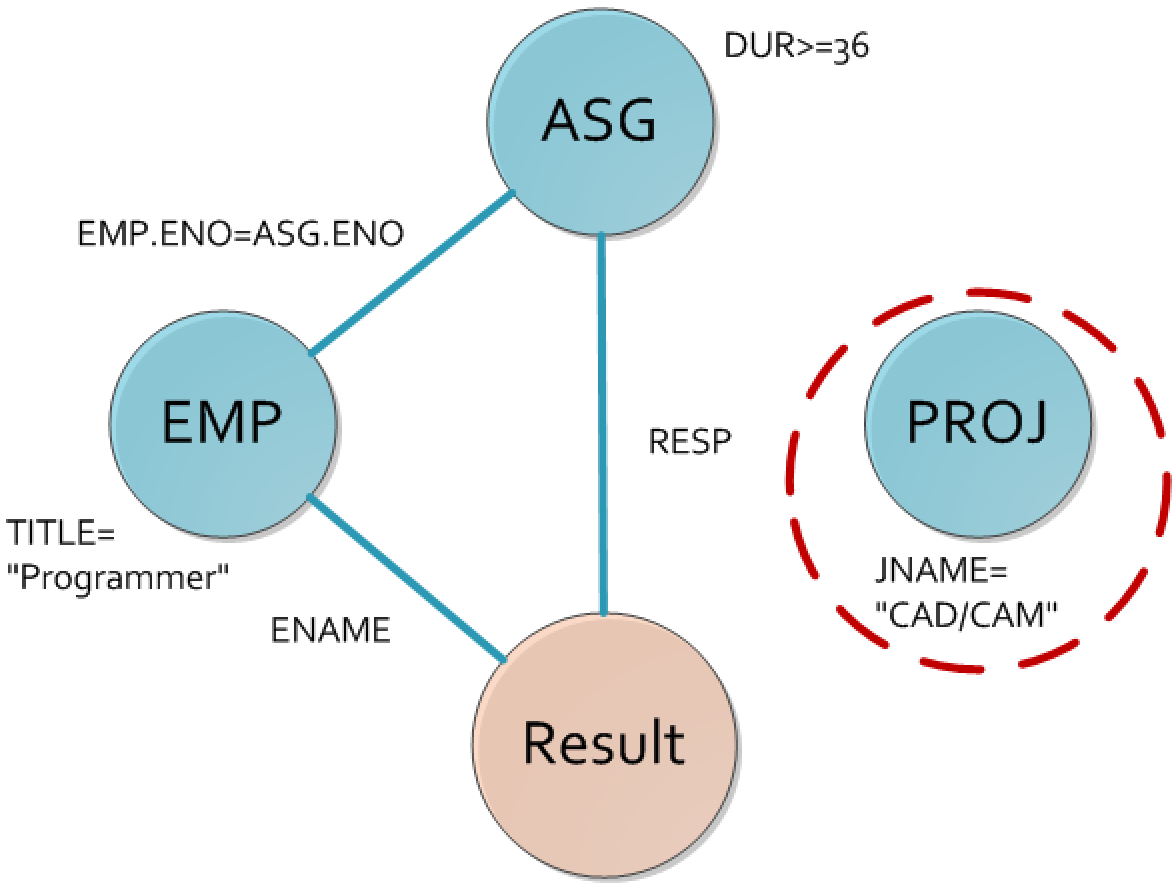
- A conjunctive query without negation is semantically incorrect if its query graph is NOT connected!

Analysis Example

- Example 2

```
SELECT      ENAME, RESP
FROM        EMP,ASG,PROJ
WHERE       EMP.ENO=ASG.GNO
    AND     ASG.PNO=PROJ.PNO
    AND     PNAME="CAD/CAM"
    AND     DUR>36
    AND     TITLE="Programmer"
```

Query Graph Example 2



3) Simplification

- Using idempotency rules to eliminate redundant predicates from WHERE clause.

$$P \wedge P \Leftrightarrow P$$

$$P \vee P \Leftrightarrow P$$

$$P \wedge true \Leftrightarrow P$$

$$P \vee false \Leftrightarrow P$$

$$P \wedge false \Leftrightarrow false$$

$$P \vee true \Leftrightarrow true$$

$$P \wedge \neg P \Leftrightarrow false$$

$$P \vee \neg P \Leftrightarrow true$$

$$P_1 \wedge (P_1 \vee P_2) \Leftrightarrow P_1$$

$$P_1 \vee (P_1 \wedge P_2) \Leftrightarrow P_1$$

Simplification Example

```
SELECT TITLE
FROM   EMP
WHERE  (NOT(TITLE=”Programmer)
        AND (TITLE=“Programmer” OR
             TITLE=“Electrical Eng.”)
        AND NOT(TITLE=“Electrical Eng.”))
        OR   ENAME=“J.Doe”
```

p1 = <TITLE = ``Programmer">
p2 = <TITLE = ``Elec. Engr">
p3 = <ENAME = ``J.Doe">

Let the query qualification is
$(\neg p1 \wedge (p1 \vee p2) \wedge \neg p2) \vee p3$

The disjunctive normal form of the query is
$= (\neg p1 \wedge p1 \wedge \neg p2) \vee (\neg p1 \wedge p2 \wedge \neg p2) \vee p3$
$= (\mathit{false} \wedge \neg p2) \vee (\neg p1 \wedge \mathit{false}) \vee p3$
$= \mathit{false} \vee \mathit{false} \vee p3$
$= p3$

Simplification Example

```
SELECT TITLE
FROM   EMP
WHERE   (NOT(TITLE=”Programmer)
         AND (TITLE=“Programmer”
         OR   TITLE=“Electrical Eng.”)
         AND NOT(TITLE=“Electrical Eng.”))
         OR  ENAME=“J.Doe”
```

is equivalent to

```
SELECT TITLE
FROM   EMP
WHERE  ENAME="J.Doe"
```

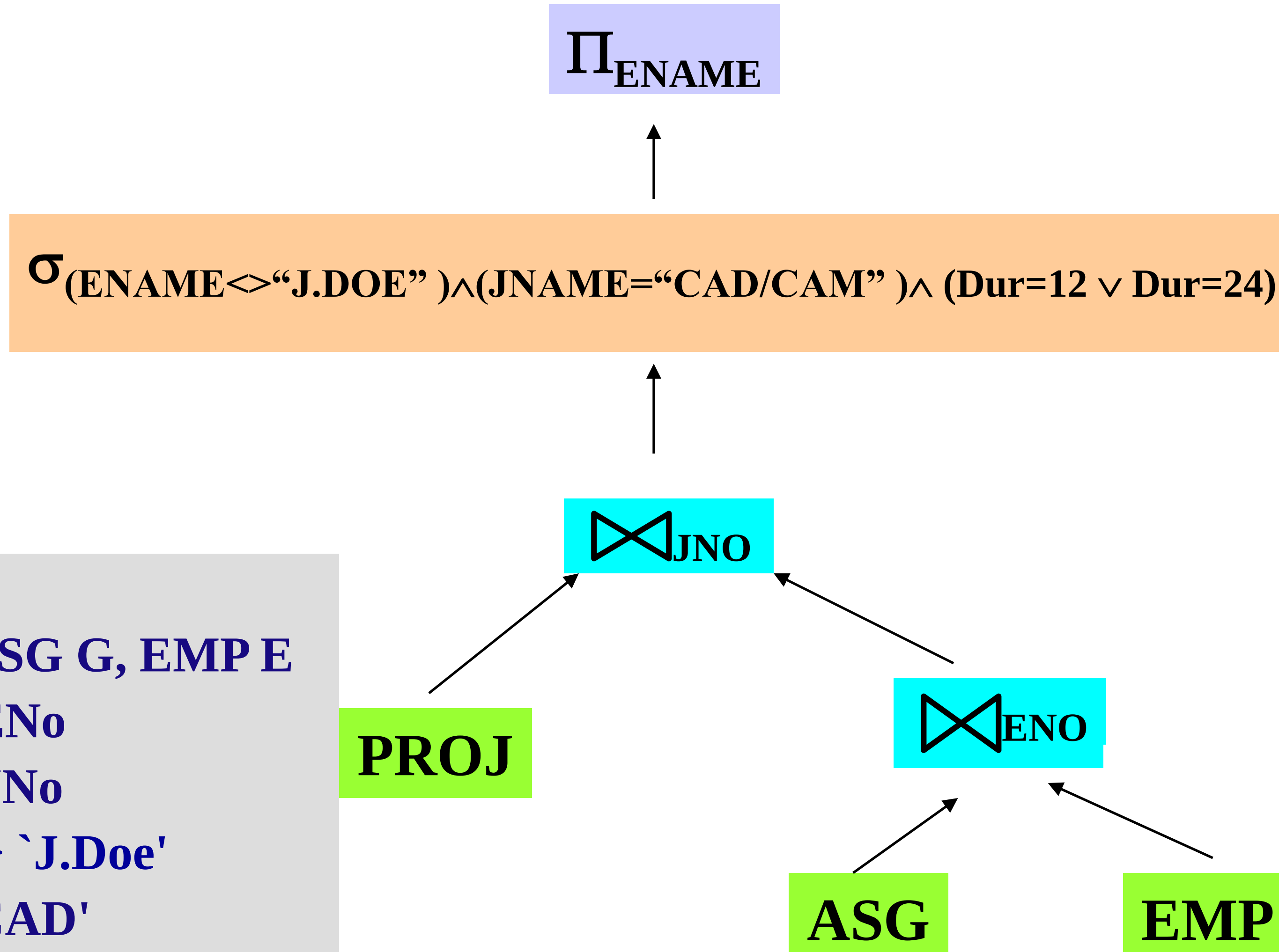
4) Rewriting

- Converting a calculus query in relational algebra
  - straightforward transformation from relational calculus to relational algebra
  - restructuring relational algebra expression to improve performance
  - making use of query trees

Relational Algebra Tree

- A tree defined by:
  - a root node representing the query result
  - leaves representing database relations
  - non-leaf nodes representing relations produced by operations
  - edges from leaves to root representing the sequences of operations

An SQL Query and Its Query Tree




```
SELECT   Ename
FROM     PROJ J, ASG G, EMP E
WHERE    G.Eno=E.ENo
   AND   G.JNo = J.JNo
   AND   ENAME <> `J.Doe'
   AND   JName = `CAD'
   AND   (Dur=12  or Dur=24)
```

How to translate an SQL query into an algebra tree?

1. Create a leaf for every relation in the FROM clause
2. Create the root as a project operation involving attributes in the SELECT clause
3. Create the operation sequence by the predicates and operators in the WHERE clause

Rewriting -- Transformation Rules (I)

- Commutativity of binary operations:

  $$R \times S \Leftrightarrow S \times R$$

  $$R \bowtie S \Leftrightarrow S \bowtie R$$

  $$R \cup S \Leftrightarrow S \cup R$$

- Associativity of binary operations:

  $$(R \times S) \times T \Leftrightarrow R \times (S \times T)$$

  $$(R \bowtie S) \bowtie T \Leftrightarrow R \bowtie (S \bowtie T)$$

- Idempotence of unary operations: grouping of projections and selections
  - $\Pi_{A'}(\Pi_{A''}(R)) \Leftrightarrow \Pi_{A'}(R)$ for $A' \subseteq A'' \subseteq A$
  - $\sigma_{p1(A1)}(\sigma_{p2(A2)}(R)) \Leftrightarrow \sigma_{p1(A1) \wedge p2(A2)}(R)$

Rewriting -- Transformation Rules (II)

- Commuting selection with projection

$$\Pi_{A1, \ldots, An}(\sigma_{p(Ap)}(R)) \Leftrightarrow \Pi_{A1, \ldots, An}(\sigma_{p(Ap)}(\Pi_{A1, \ldots, An, Ap}(R)))$$

- Commuting selection with binary operations

$$\sigma_{p(Ai)}(R \times S) \Leftrightarrow (\sigma_{p(Ai)}(R)) \times S$$

$$\sigma_{p(Ai)}(R \bowtie S) \Leftrightarrow (\sigma_{p(Ai)}(R)) \bowtie S$$

$$\sigma_{p(Ai)}(R \cup S) \Leftrightarrow \sigma_{p(Ai)}(R) \cup \sigma_{p(Ai)}(S)$$

- Commuting projection with binary operations

$$\Pi_C(R \times S) \Leftrightarrow \Pi_A(R) \times \Pi_B(S) \quad \text{where } C = A \cup B$$

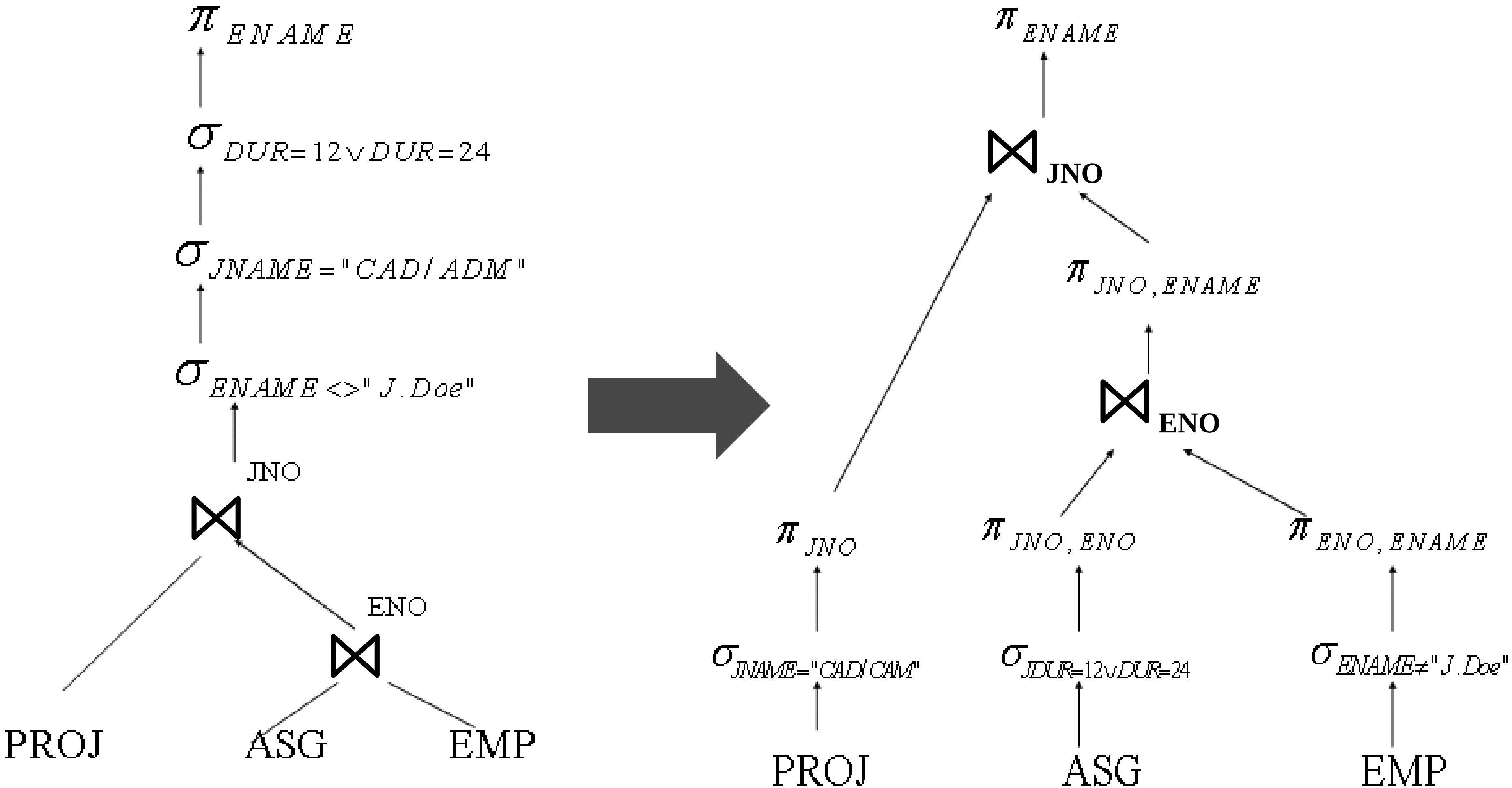
$$\Pi_C(R \bowtie S) \Leftrightarrow \Pi_C(R) \bowtie \Pi_C(S)$$

$$\Pi_C(R \cup S) \Leftrightarrow \Pi_C(R) \cup \Pi_C(S)$$

How to use transformation rules to optimize?

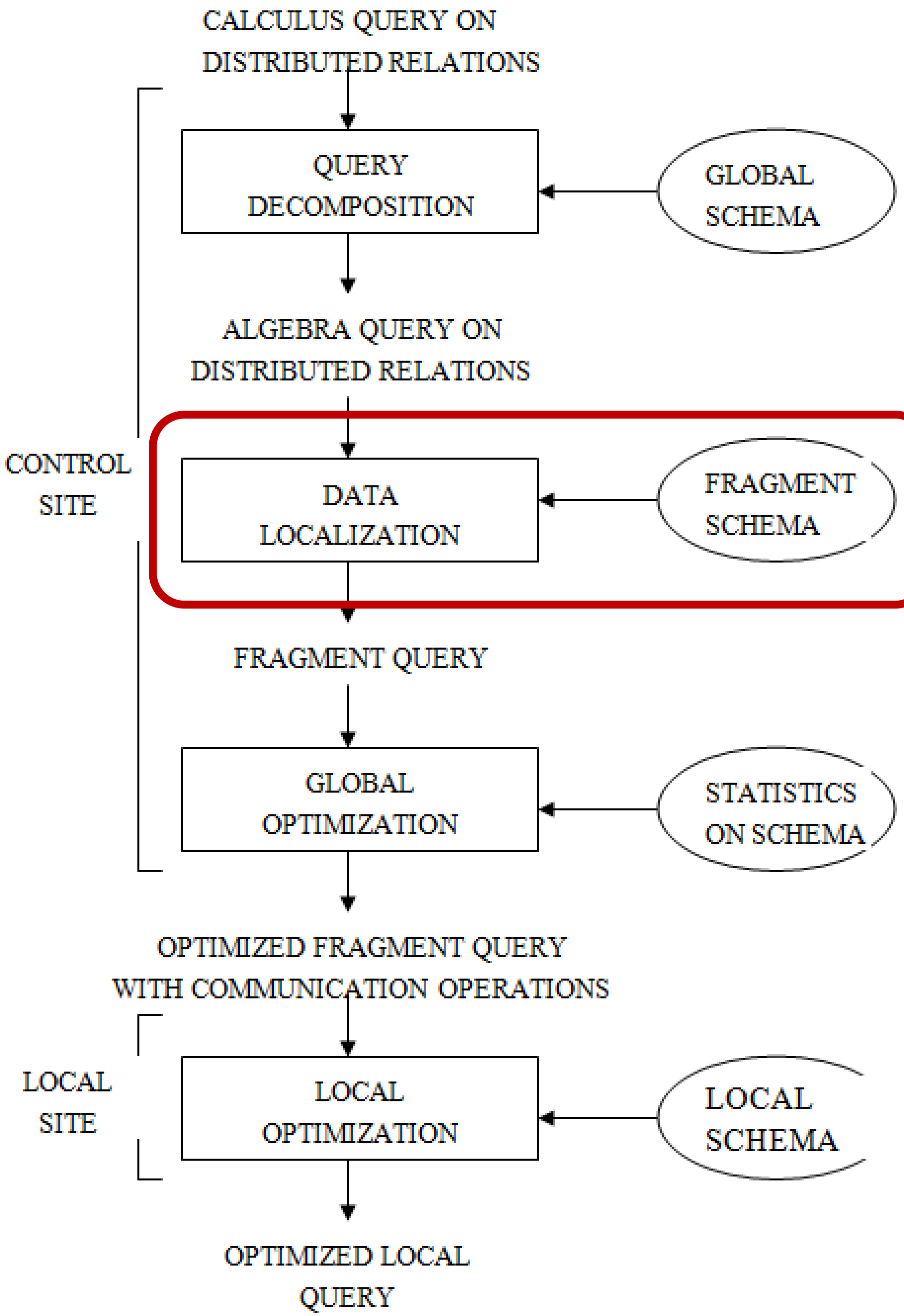
- Unary operations on the same relation may be grouped to access the same relation once
- Unary operations may be commuted with binary operations, so that they may be performed first to reduce the size of intermediate relations
- Binary operations may be reordered

Optimization of Previous Query Tree

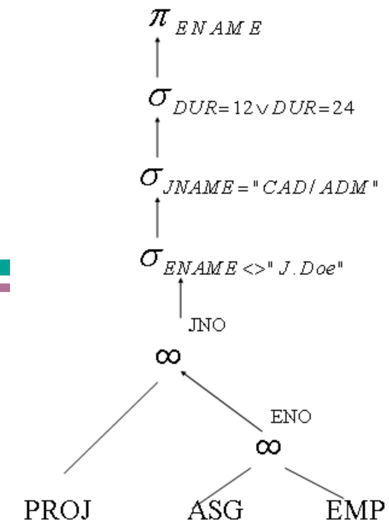


Step 2 : Data Localization

- Task : To translate a query on global relation into algebra queries on physical fragments, and optimize the query by reduction.



Data Localization - Example



EMP is fragmented into

$EMP1 = \sigma_{ENO \leq \text{“E3”}}(EMP)$

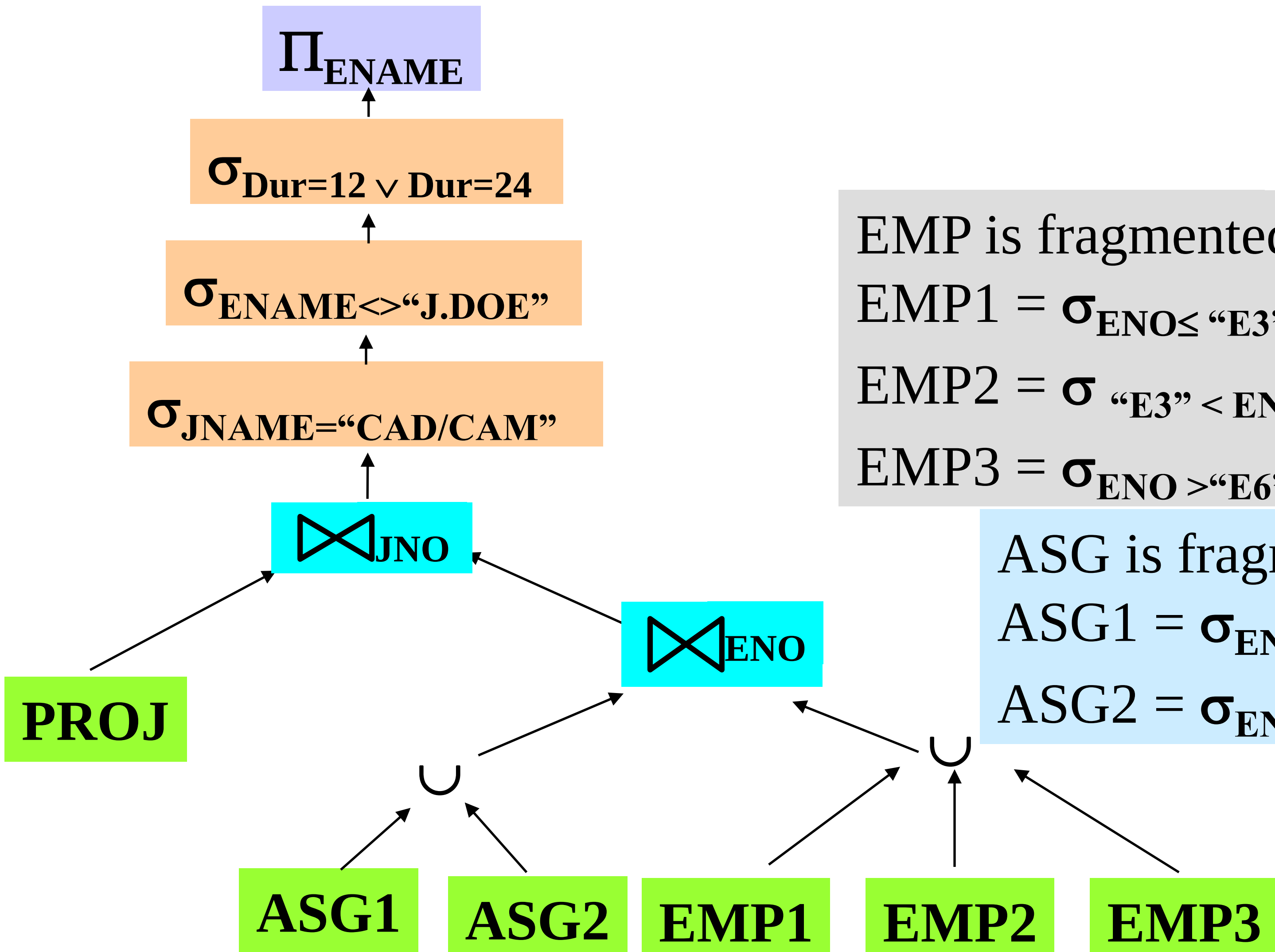
$EMP2 = \sigma_{\text{“E3”} < ENO \leq \text{“E6”}}(EMP)$

$EMP3 = \sigma_{ENO > \text{“E6”}}(EMP)$

ASG is fragmented into

$ASG1 = \sigma_{ENO \leq \text{“E3”}}(ASG)$

$ASG2 = \sigma_{ENO > \text{“E3”}}(ASG)$



Reduction with Selection for PHF

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

EMP is fragmented into

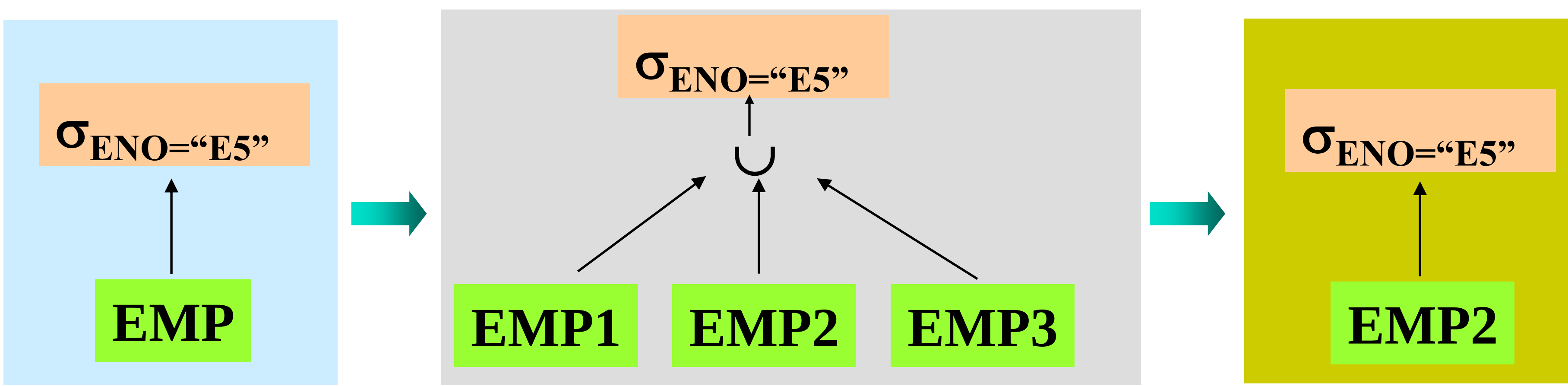
$EMP1 = \sigma_{ENO \leq \text{"E3"}}(EMP)$

$EMP2 = \sigma_{\text{"E3"} < ENO \leq \text{"E6"}}(EMP)$

$EMP3 = \sigma_{ENO > \text{"E6"}}(EMP)$

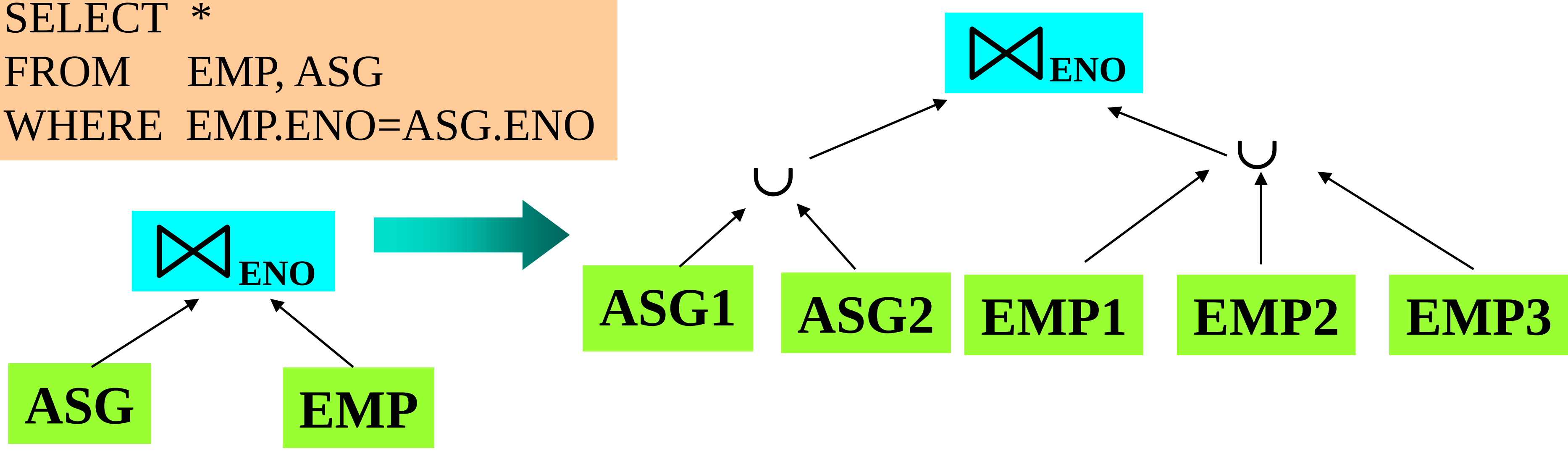
Given Relation R, $F_R = \{R_1, R_2, \ldots, R_n\}$ where $R_j = \sigma_{p_j}(R)$

$\sigma_{p_j}(R_j) = \varnothing$ if $\forall x \in R: \neg(p_i(x) \wedge p_j(x))$



Reduction with Join for PHF

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



ASG is fragmented into

ASG1 = $\sigma_{ENO \leq \text{"E3"}}(ASG)$

ASG2 = $\sigma_{ENO > \text{"E3"}}(ASG)$

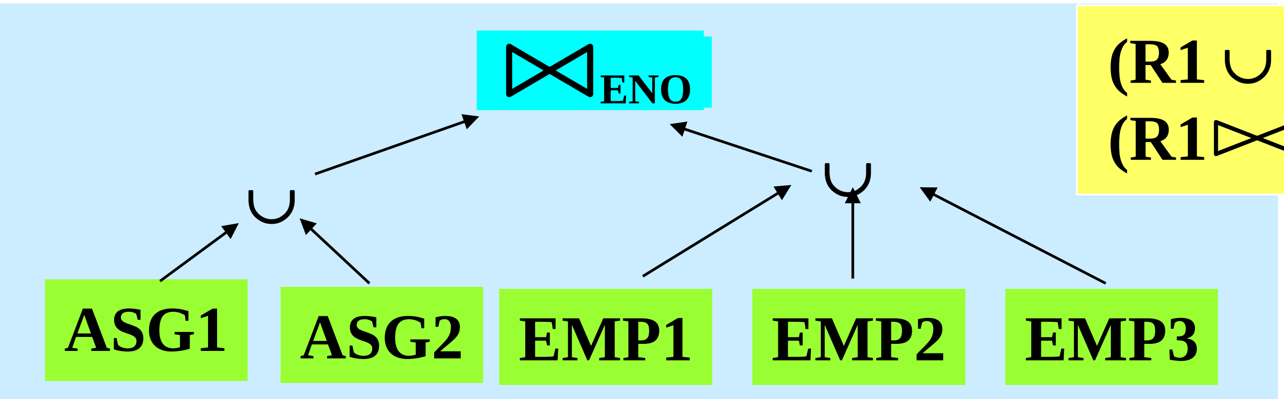
EMP is fragmented into

EMP1 = $\sigma_{ENO \leq \text{"E3"}}(EMP)$

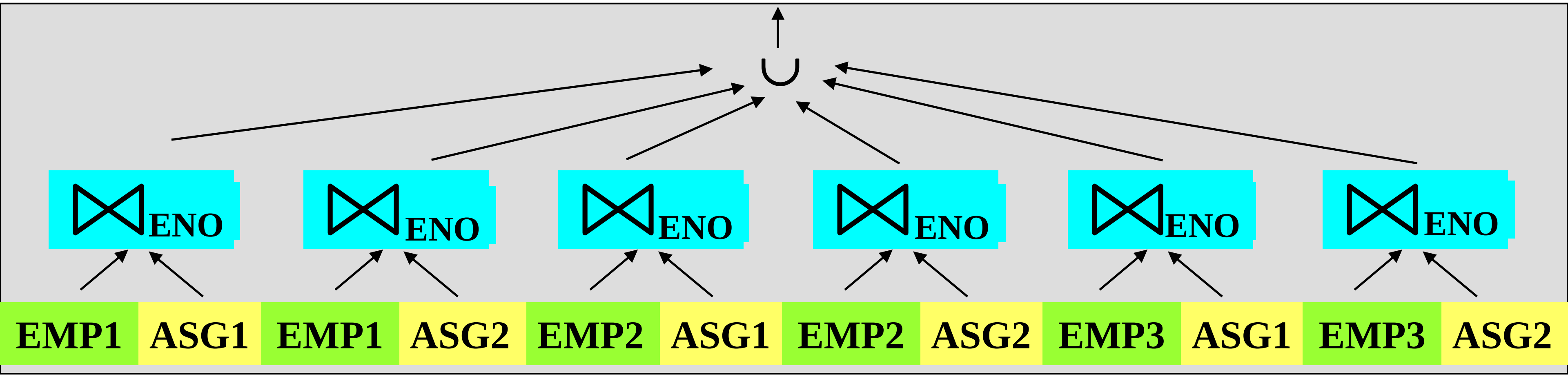
EMP2 = $\sigma_{\text{"E3"} < ENO \leq \text{"E6"}}(EMP)$

EMP3 = $\sigma_{ENO > \text{"E6"}}(EMP)$

Reduction with Join for PHF (I)



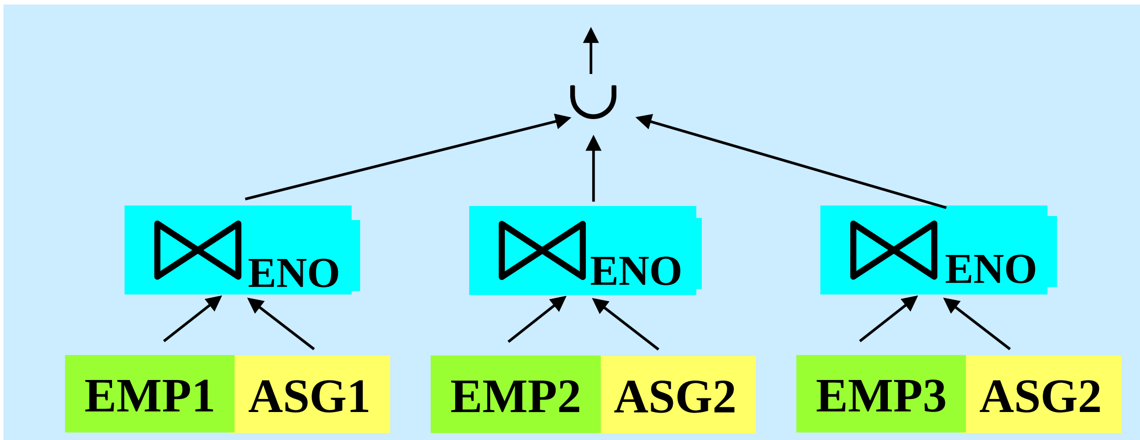
$(R1 \cup R2) \bowtie S \Leftrightarrow$
$(R1 \bowtie S) \cup (R2 \bowtie S)$



EMP1 = $\sigma_{ENO \leq \text{"E3"}}$ (EMP)
EMP2 = $\sigma_{\text{"E3"} < ENO \leq \text{"E6"}}$ (EMP)
EMP3 = $\sigma_{ENO > \text{"E6"}}$ (EMP)

ASG1 = $\sigma_{ENO \leq \text{"E3"}}$ (ASG)
ASG2 = $\sigma_{ENO > \text{"E3"}}$ (ASG)

Reduction with Join for PHF (II)



Given $R_i = \sigma_{pi}(R)$ and $R_j = \sigma_{pj}(R)$

$R_i \bowtie Rj = \emptyset$ if $\forall x \in R_i$, $\forall y \in R_j$: $\neg(p_i(x) \wedge p_j(y))$

Reduction with join

1. **Distribute join over union**
2. **Eliminate unnecessary work**

Reduction for VF

- Find useless intermediate relations

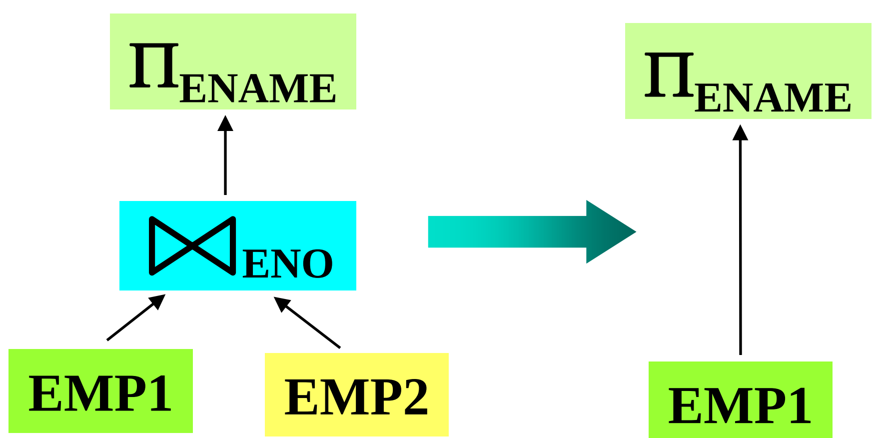
Relation R defined over attributes A = {A1, A2, …, An} vertically fragmented as $R_i = \Pi_{A'}(R)$ where $A' \subseteq A$

$\Pi_D(R_i)$ is useless if the set of projection attributes D is not in A'

EMP1= $\Pi_{ENO,ENAME}$ (EMP)

EMP2= $\Pi_{ENO,TITLE}$ (EMP)

```
SELECT   ENAME
FROM     EMP
```



Reduction for DHF

Distribute joins over union

Apply the join reduction for horizontal fragmentation

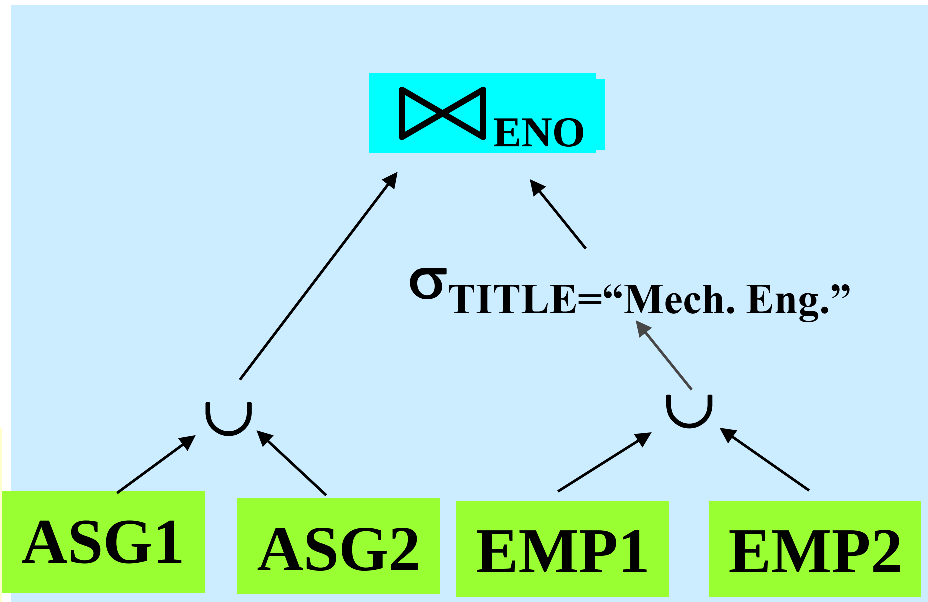
EMP1: $\sigma_{\text{TITLE=“Programmer”}}$ **(EMP)**

EMP2: $\sigma_{\text{TITLE}\neq\text{“Programmer”}}$ **(EMP)**

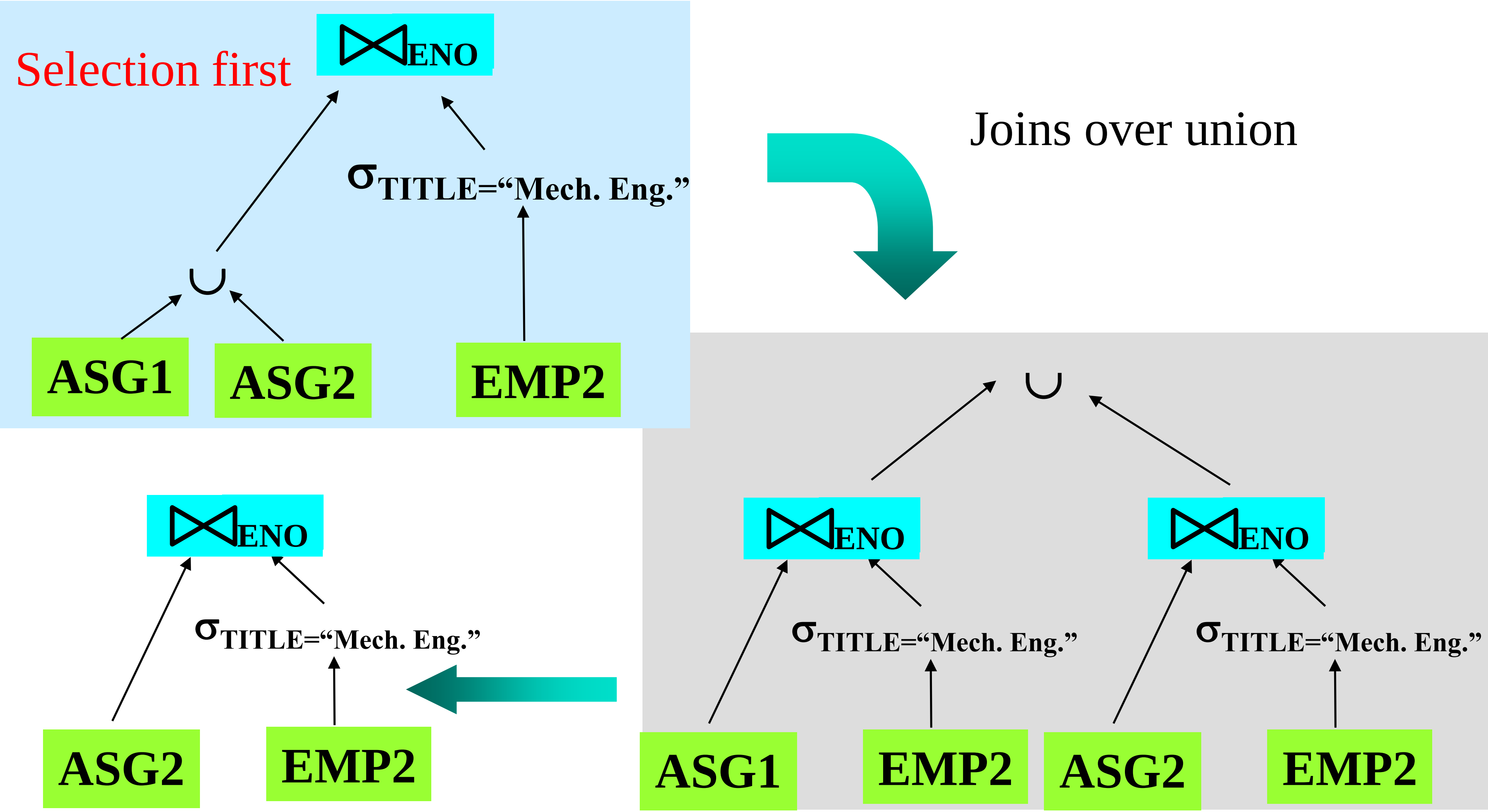
ASG1: ASG $\ltimes_{\text{ENO}}$ **EMP1**

ASG2: ASG $\ltimes_{\text{ENO}}$ **EMP2**

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



Reduction for DHF (II)



Reduction for Hybrid Fragmentation

- Combine the rules already specified
  - Remove *empty relations* generated by contradicting selection on horizontal fragments;
  - Remove *useless relations* generated by projections on vertical fragments;
  - Distribute *joins over unions* in order to isolate and remove useless joins.

Reduction for Hybrid Fragmentation - Example

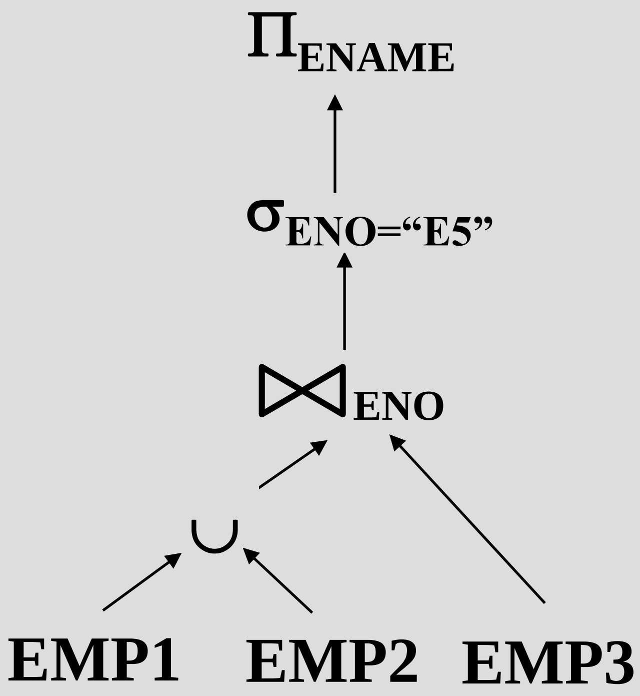
$EMP1 = \sigma_{ENO \leq \text{"E4"}} (\Pi_{ENO,ENAME} (EMP))$

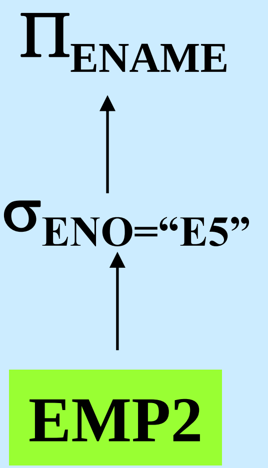
$EMP2 = \sigma_{ENO > \text{"E4"}} (\Pi_{ENO,ENAME} (EMP))$

$EMP3 = \Pi_{ENO,TITLE} (EMP)$

QUERY:

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





Question & Answer