5. Distributed Query Processing

Chapter 7

Overview of Query Processing

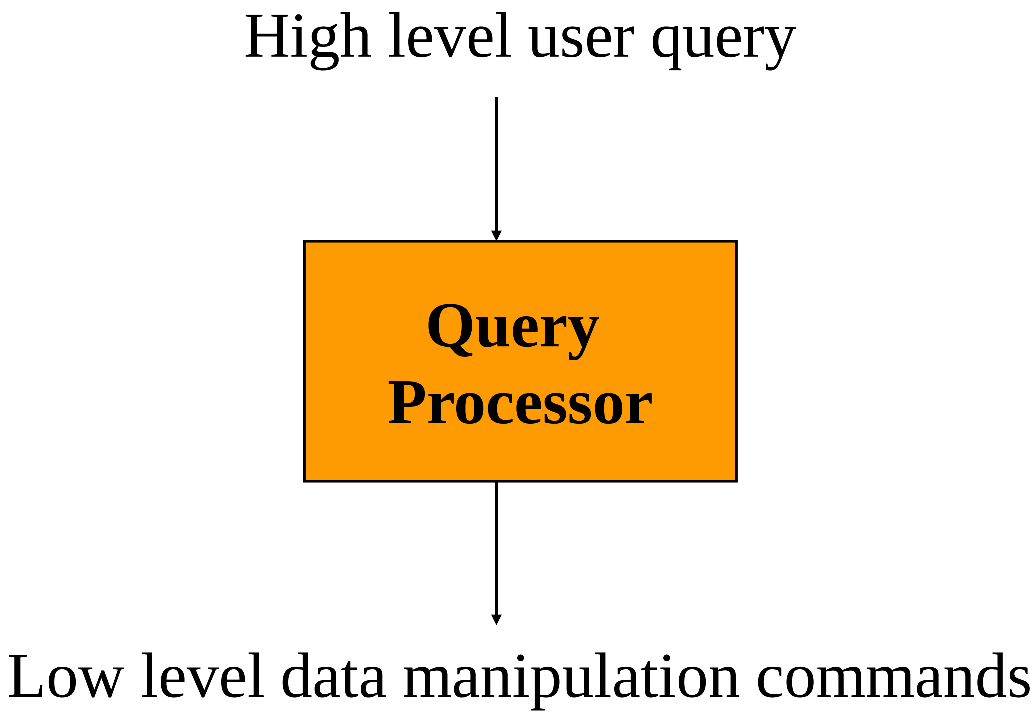
Chapter 8

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.*

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

Query Language – Domain Calculas

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

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

- Example:

  { x, y | EMP(x, y, “Manager") }

Variables are position sensitive!

Query Language SQL

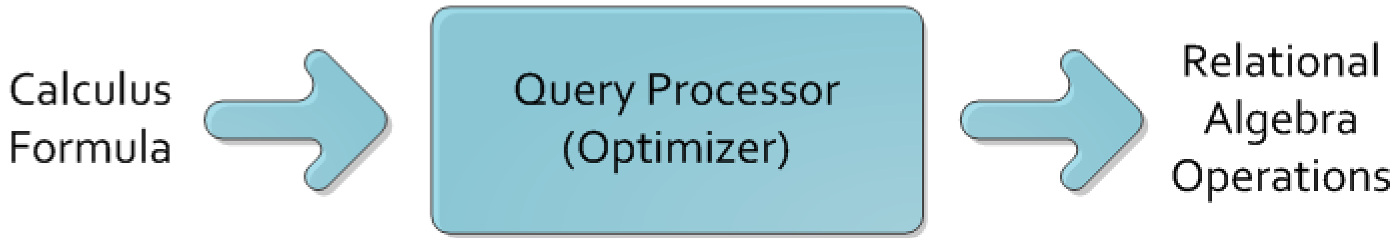
- SQL is a tuple calculus language.

```
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"
```

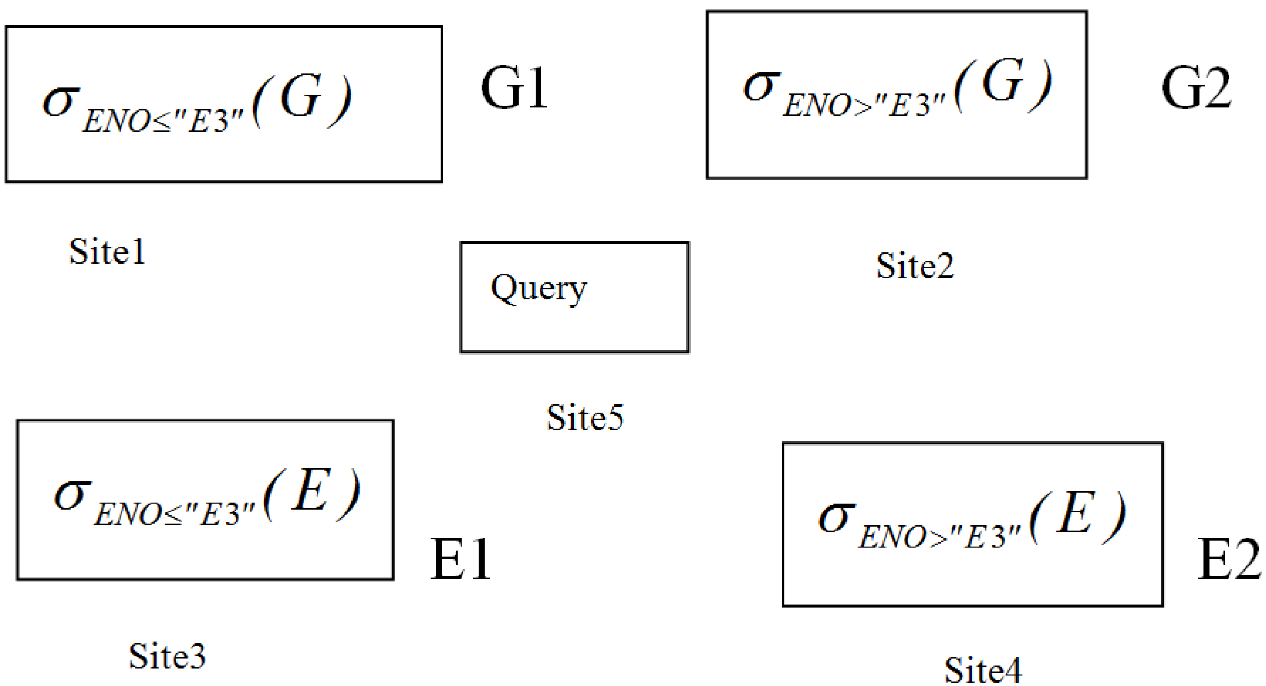
- Strategy 1: $\pi_{ENAME}\left(\sigma_{TITLE="manager" \wedge E.ENO=G.ENO}\left(E \times G\right)\right)$

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

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.



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

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

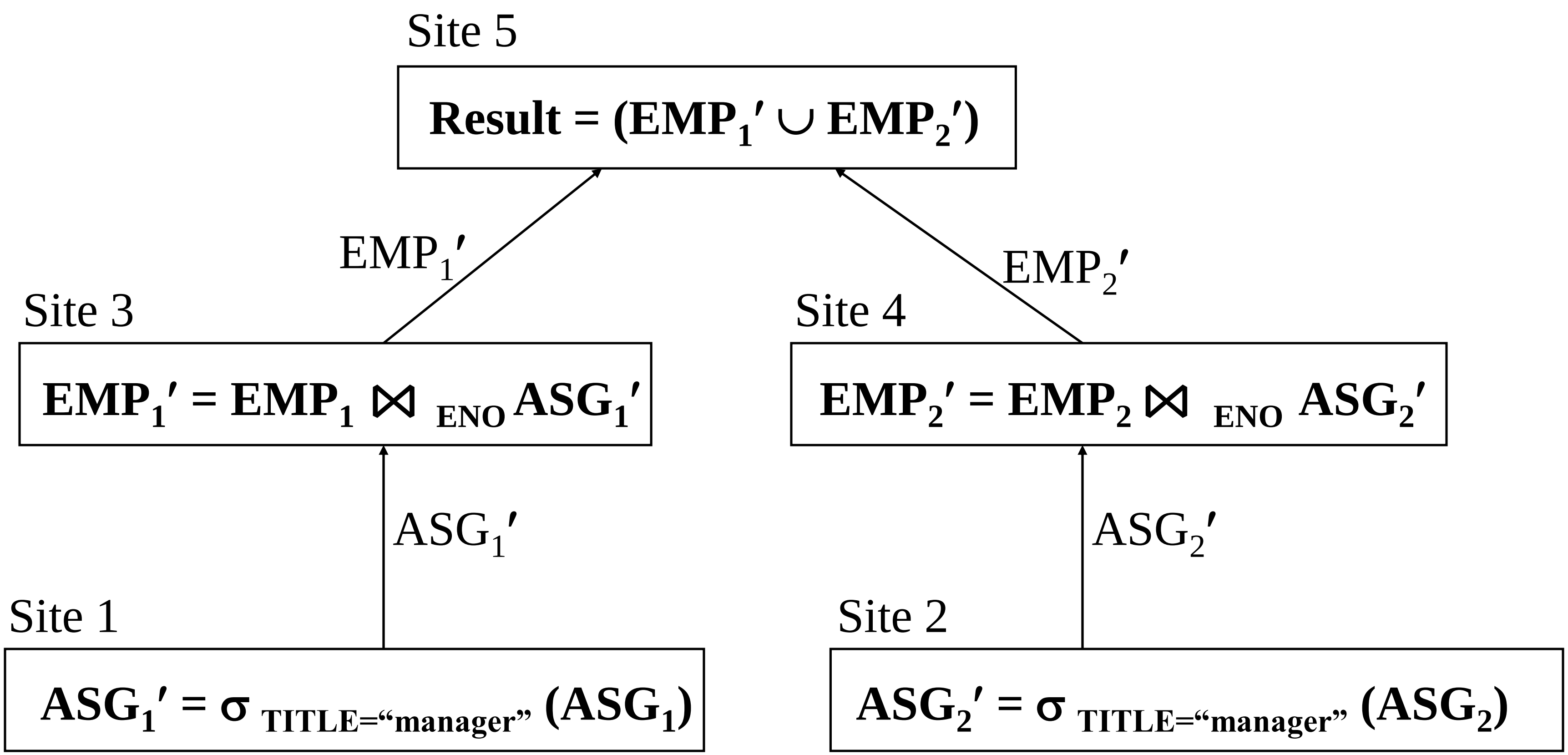
ASG_1 ASG_2 EMP_1 EMP_2

Site 1 Site 2 Site 3 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

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

- 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*, $-$	$O(n\log n)$
Cartesian-Product X	$O(n^2)$

What is O-Notation？

- Algorithm Analysis Basics
- O-Notation：

  Intuition, Definition, Manipulation, and Limitations

Algorithm Analysis Basics

- Goal: measure the efficiency of algorithms.
  - How much time and space does an algorithm use when it processes an input?
- Question: how to characterize and measure the performance of an algorithm?
- Why not measure execution time of an algorithm?
  - Varying significantly with different computers, programming languages, and compilers

Resource Consumption Patterns

- Observation: algorithms usually take more time/ space as the size of the problem grows
  - A problem size *n* can be
    - the length of a list that an algorithm searches;
    - the number of nodes in a tree that an algorithm prints;
    - the number of items in an array that an algorithm sorts, etc.

Array Size *n*	Computer1	Computer2
125	12.5	2.8
250	49.3	11.0
500	195.8	43.4
1000	780.3	172.9
2000	3114.9	690.5

Table 1: SelectionSort algorithm running times example in milliseconds on two types of computers.

Measurement Example

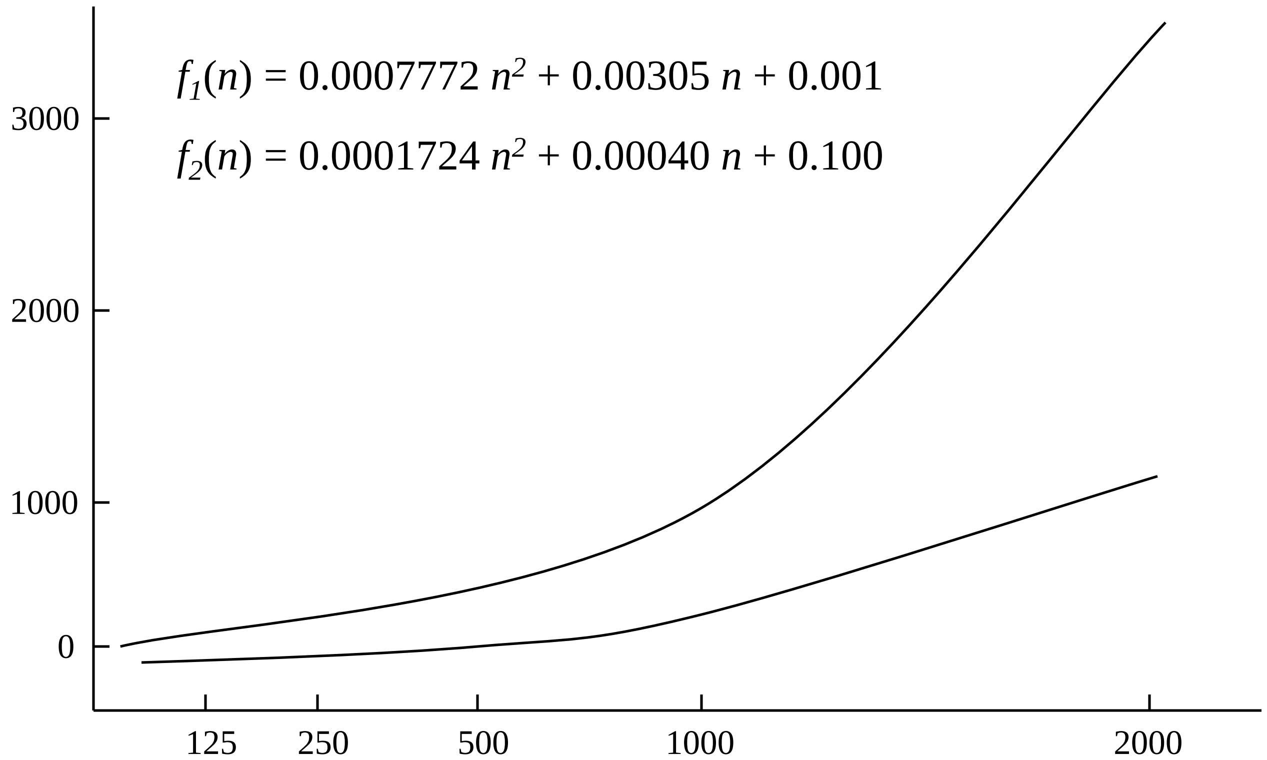
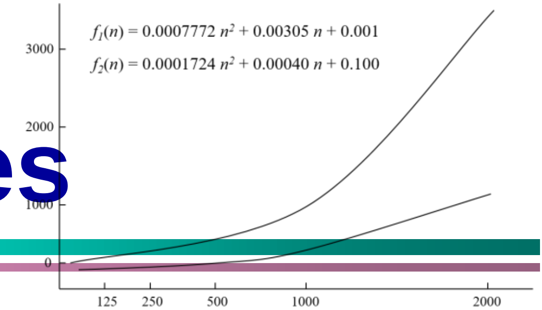


Figure 1: Two curves fitting the data in Table 1.



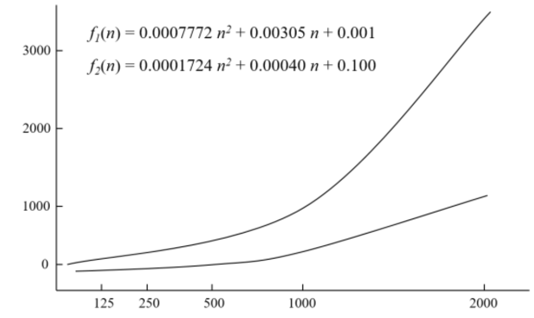
Measurement Example Notes

- The quadratic functions (the highest power is 2) shown are the only good fit for the data points gathered.
- The curves differ in their coefficients, but both are quadratic in terms of the size *n*.
- If continuing gathering execution times using different computers, languages, and compilers, we'll discover that the amount of time the algorithm consumes is best approximated by a quadratic curve of the form $f(n) = an^2 + bn + c$.

The *shape* of the curve depends only on the algorithm used.

Measurement Example Generalization

- Each curve expresses an algorithm's resource consumption in terms of the problem size.
- Each algorithm is associated with a family of similarly shaped curves.
- Running times for different algorithms fall into different complexity classes, characterized by different families of curves.



Ignore coefficients and concentrate on the *kind* of function.

Simplification of Complexity Functions

- Ignore the less terms and focus on the dominant one.
  - The dominant term is the one that grows faster when *n* grows

$f(n) = an^2+bn+c$ where a=0.0001724, b=0.0004, c=0.1			
n	f(n)	an^2	n-term as % of total
125	2.8	2.7	94.7
250	11.0	10.8	98.2
500	43.4	43.1	99.3
1000	172.9	172.4	99.7
2000	690.5	689.6	99.9

Table 2: Percentage contribution of n^2 term to the total.

- Ignore the coefficient of the dominant term

Intuitionof O-Notation

- Two simplification rules lead to the O-notation
  - e.g., the O-notation for the quadratic function, $f(n)=an^2+bn+c$, is $O(n^2)$.

f(n)	O(f(n))
$0.3n^2 + 20n + 512$	
$0.0001n^4 + 10000n^2$	
$42\log_2 n$	
$7n\log_{10} n + 2n - 12$	

O-notation gives us a language to describe algorithmic complexity

Common Complexity Classes

Adjective Name	O-Notation
Constant	$O(1)$
Logarithmic	$O(\log n)$
Linear	$O(n)$
n log n	$O(n \log n)$
Quadratic	$O(n^2)$
Cubic	$O(n^3)$
Exponential	$O(2^n)$
Exponential	$O(10^n)$

Running Times Example

	2 (2^1)	16 (2^4)	256 (2^8)	1024 (2^{10})	1048576 (2^{20})
1	1 μs	1 μs	1 μs	1 μs	1 μs
$\log_2 n$	1 μs	4 μs	8 μs	10 μs	20 μs
n	2 μs	16 μs	256 μs	1.02 ms	1.05 s
$n \log_2 n$	2 μs	64 μs	2.05 ms	10.2 ms	21 s
n^2	4 μs	25.6 μs	65.5 ms	1.05 s	1.8 wks
n^3	8 μs	4.1 ms	16.8 s	17.9 min	36559 yrs
2^n	4 μs	65.5 msec	$3.7*10^{63}$ yrs	$5.7*10^{294}$ yrs	$2.1*10^{315639}$ yrs

Estimated lifetime of the sun: $5*10^9$ yrs !

1 μs = 10^{-6} s	1 s = one second	1 wk = 604800 s
1 ms = 10^{-3} s	1 min = 60 s	1 yr = 31557600 s

Observations from Running Time Example

- For problems of small size (e.g.,$n \leq 16$), the complexity class of f(n) does not matter much.
- For problems of medium size (e.g., n=1024), algorithms that are no more complex than n^2 are still useful.
- For problems of large size (e.g., n=1048576), the difference between $n\log_2 n$ and n^2 algorithms is huge.
- Exponential algorithms (e.g. 2^n) tends to take a disastrously long time for all but small problems.

Change of Observation Angle

Table 5: Size of the largest problem that an algorithm can solve if solution is computed in time ≤ T at 1 microseconds per step.

$f(n)$ \ T	1 min	1 hr	1 day	1 wk	1 yr
n	$6*10^{7}$	$3.6*10^{9}$	$8.64*10^{10}$	$6.05*10^{11}$	$3.15*10^{13}$
$n \log_2 n$	$2.8*10^{6}$	$1.3*10^{8}$	$2.75*10^{9}$	$1.77*10^{10}$	$7.97*10^{11}$
n^2	$7.75*10^{3}$	$6.0*10^{4}$	$2.94*10^{5}$	$7.78*10^{5}$	$5.62*10^{6}$
n^3	$3.91*10^{2}$	$1.53*10^{3}$	$4.42*10^{3}$	$8.46*10^{3}$	$3.16*10^{4}$
2^n	25	31	36	39	44
10^n	7	9	10	11	13

Algorithms of Different Complexity

- Constant time algorithms O(1)
  - Take no more than a *fixed* amount of time to run regardless of the problem size.
  - e.g., choose and print a single random array item A[i] in an array A[0:n-1].
- Linear time algorithms O(n)
  - Run in time proportional to the problem size.
  - e.g., search for occurrences of a give word in a document.

Algorithms of Different Complexity (*cont.*)

- Exponential time algorithms O(2^n)
  - Not practical to use for any but small problems.
  - e.g., traveling salesperson problem solution, computing moves in game-playing situations.

- Quadratic and cubic time algorithms O(n^2) / O(n^3)
  - e.g., computation on an $n*n$ matrix, or in 3-dimensional space.

Efficiency of List Implementation

List Operation	Sequential Representation	Linked representation
Find the length	O(1)	O(n)
Insert a new first item	O(n)	O(1)
Delete the last item	O(1)	O(n)
Replace the i^{th} item	O(1)	O(n)
Delete the i^{th} item	O(n)	O(n)

Best, Worst, and Average Cases

- For some algorithms, different input data of a given size requires different amounts of time.
  - e.g., to sequential search for an item *s* in an array
    - best case: *s* is the first element in the array
    - worst case: *s* is the last element in the array
    - average case: need to go halfway on average to find *s*
- While average time seems to be the fairest measure, it may be difficult to determine.
- In real-time situations, the worst case time is important.

Formal Definition of O-Notation

$f(n)$ is $O(g(n))$ if and only if
there exists a positive constant K and n_0,
such that $|f(n)| \le K\,|g(n)|$ for all $n \ge n_0$.

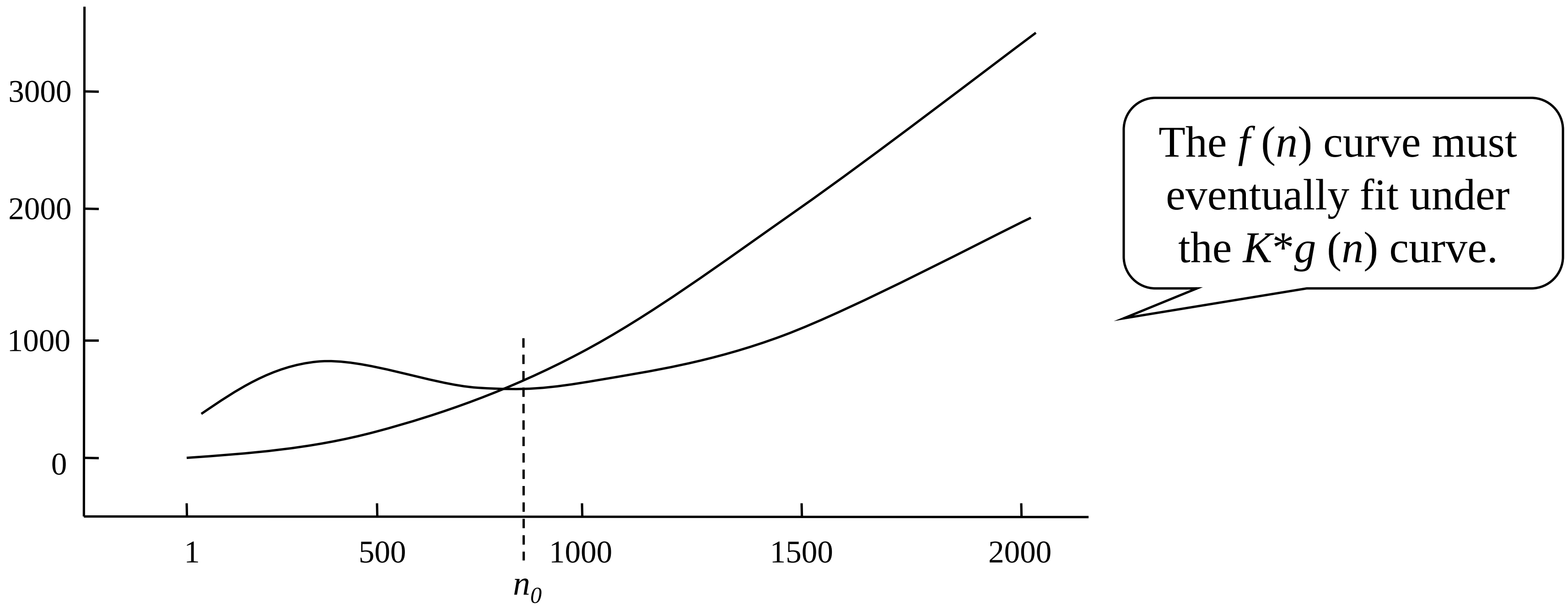


Figure 2: Graphical meaning of O-notation

Formal Proof of O-Notation

- The two simplification rules can be proven using the definition.

e.g., $f(n) = \dfrac{3n(n+1)}{2}$ is $O(n^2)$.

Proof: Let $K=3$, $n_0=1$, $g(n) = n^2$.

Substitute in the formal O-notation definition, we have

$|f(n)| \leq K\,|g(n)|$ for all $n \geq n_0$, because

$$\frac{3n(n+1)}{2} \leq 3n^2 \quad (n \geq 1)$$

Practical Shortcuts for Manipulating O-Notation

- An easy way to determine the O-notation for f(n).
  - Separate f(n) into a dominant term and lesser terms:

    f(n) = (dominant term) ± (lesser terms)

  - Throw away the lesser terms:

    O(f(n)) = O(dominant term ~~± lesser terms~~)

    = O(dominant term)

  - Ignore and drop coefficients involved.

e.g., $O(6n^3 - 15n^2 + 3nlogn) = O(6n^3) = O(n^3)$

$$\boxed{O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(10^n)}$$

Remarks on O-Notation

- When writing O-notation, never use the bases of logarithms.
  - e.g., O(logn), O(n logn), but not $O(\log_2 n)$, $O(n \log_2 n)$.
  - **Reason**: changing bases only involves multiplying by new constants (e.g., $\log_{10} n = 0.3010 * \log_2 n$); and O-notation ignores constants of proportionality.

- Always use O(1) to denote the constant O-notation.
  - **Reason:** for an algorithm running in a number of steps f(n) which is always less than *K*, we have $f(n) \leq K * 1$, where $g(n) = 1$.

Remarks on O-Notation (*cont.*)

- When determining the O-notation for f(n), try to
  - make the bound as tight as possible;
    - e.g., if "$f(n)$ is $O(n)$", it is also $O(n^2)$, $O(n^3)$, …, but we use only the tightest upper bound.
  - make the function as simple as possible.
    - Let $g(n)$ be a single term with a coefficient of one, e.g., we say "$f(n)$ is $O(n^2)$", rather than "$f(n)$ is $O(3n^2 + n\log n)$".

When O-Notation Can't Be Trusted?

- Comparisons based on O-notation only apply to large problem sizes.
  - For small problem sizes, the constants in the running time equations dominate the observed running times.
  - e.g., to find an item in an ordered list,
    - *sequential search, binary search*, and *interpolation search* are O(n), O(logn), and O(log logn), respectively.
    - when tables contain items -- 1) <u>over 500</u>, *interpolation search* is the fastest; 2) <u>between 20 and 500</u>, *binary search* is the fastest; 3) <u>up to 20</u>, *sequential search* is the fastest.
- The "measurement and tuning" method is useful to identify optimal solutions to small-sized problems.

Summary

- It is important to analyze the performance of algorithms and data structures.
- The problem of measurement:
  - the hard performance measures (e.g., execution time) vary significantly with different computers, programming languages, and compilers.
- The O-notation solution:
  - compare performance based on resource consumption patterns.
  - intuition, definition, manipulation, and limitations.

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

$R \bowtie S$

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.

$$R \ltimes S$$

Layers of Query Processing

Calculus Query on Distributed Relations

CONTROL SITE

QUERY DECOMPOSITION ← GLOBAL SCHEMA

Algebra Query on Distributed Relations

DATA LOCALIZATION ← FRAGMENT SCHEMA

Fragment Query

GLOBAL OPTIMIZATION ← STATISTICS ON FRAGMENTS

Optimized Fragment Query
With Communication Operations

LOCAL SITE

LOCAL OPTIMIZATION ← LOCAL SCHEMA

Optimized Local Queries

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:

$$
\begin{aligned}
& EMP.ENO = ASG.ENO \\
& \wedge ASG.JNO = "J1" \\
& \wedge (DUR = 12 \vee DUR = 24)
\end{aligned}
$$

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*

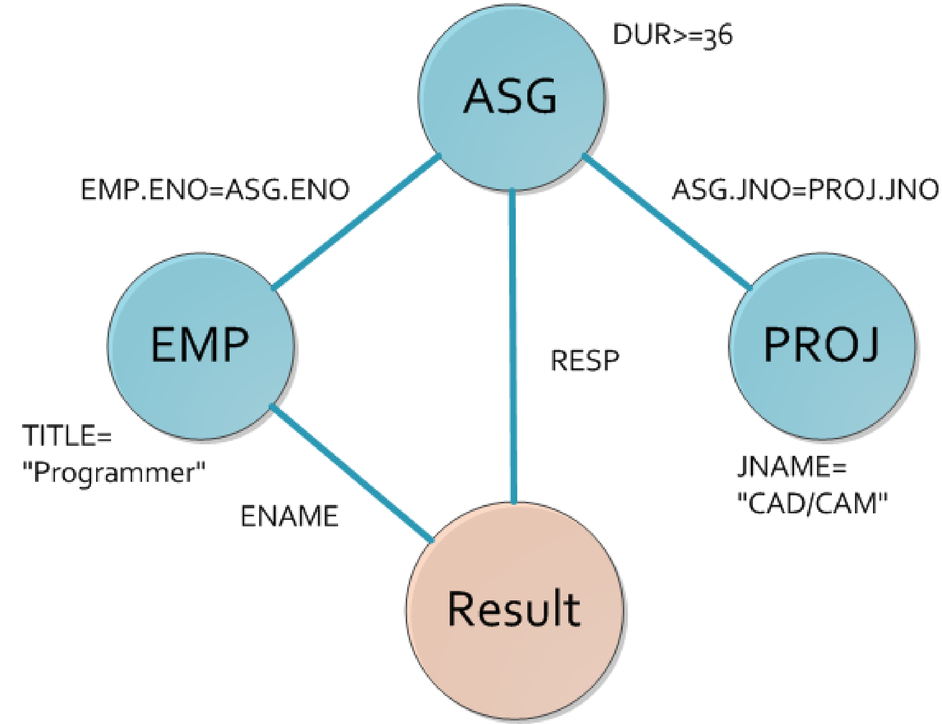
Query Graph

- 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.

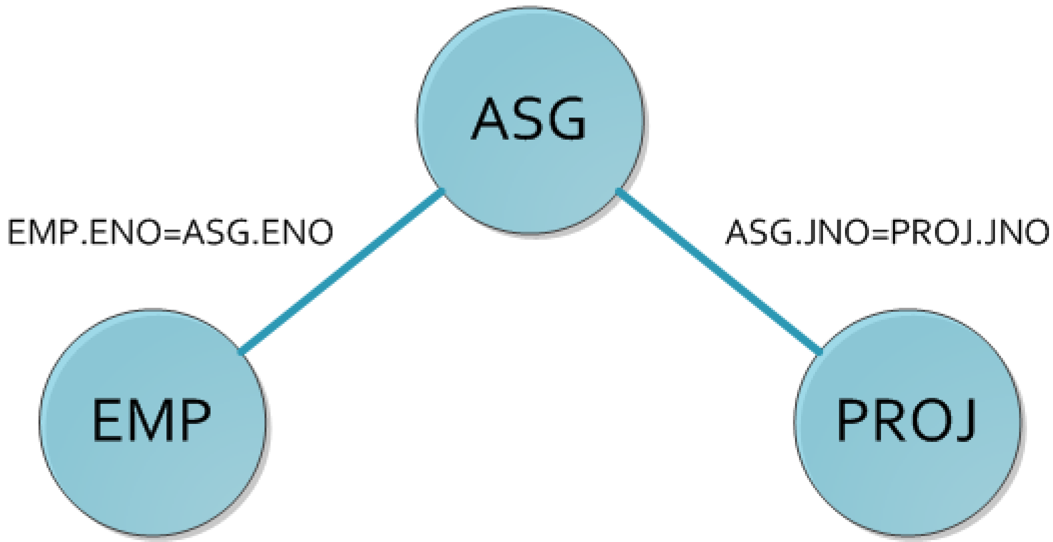
Query Graph Example

```
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="Programmer"
```



Join Graph Example 1

A subgraph of query graph for join operation.



Tool of Analysis

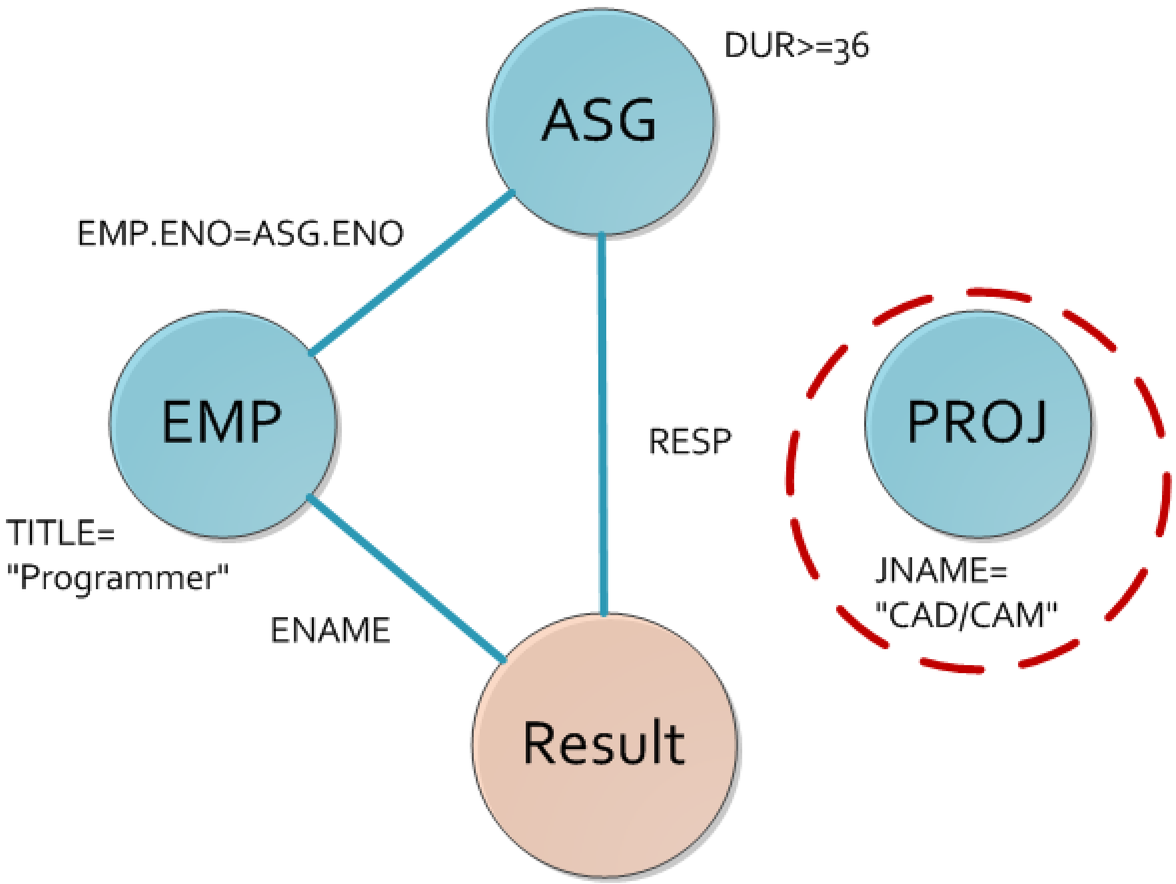
- A conjunctive query without negation is semantically incorrect if its query graph is NOT connected!

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"
```

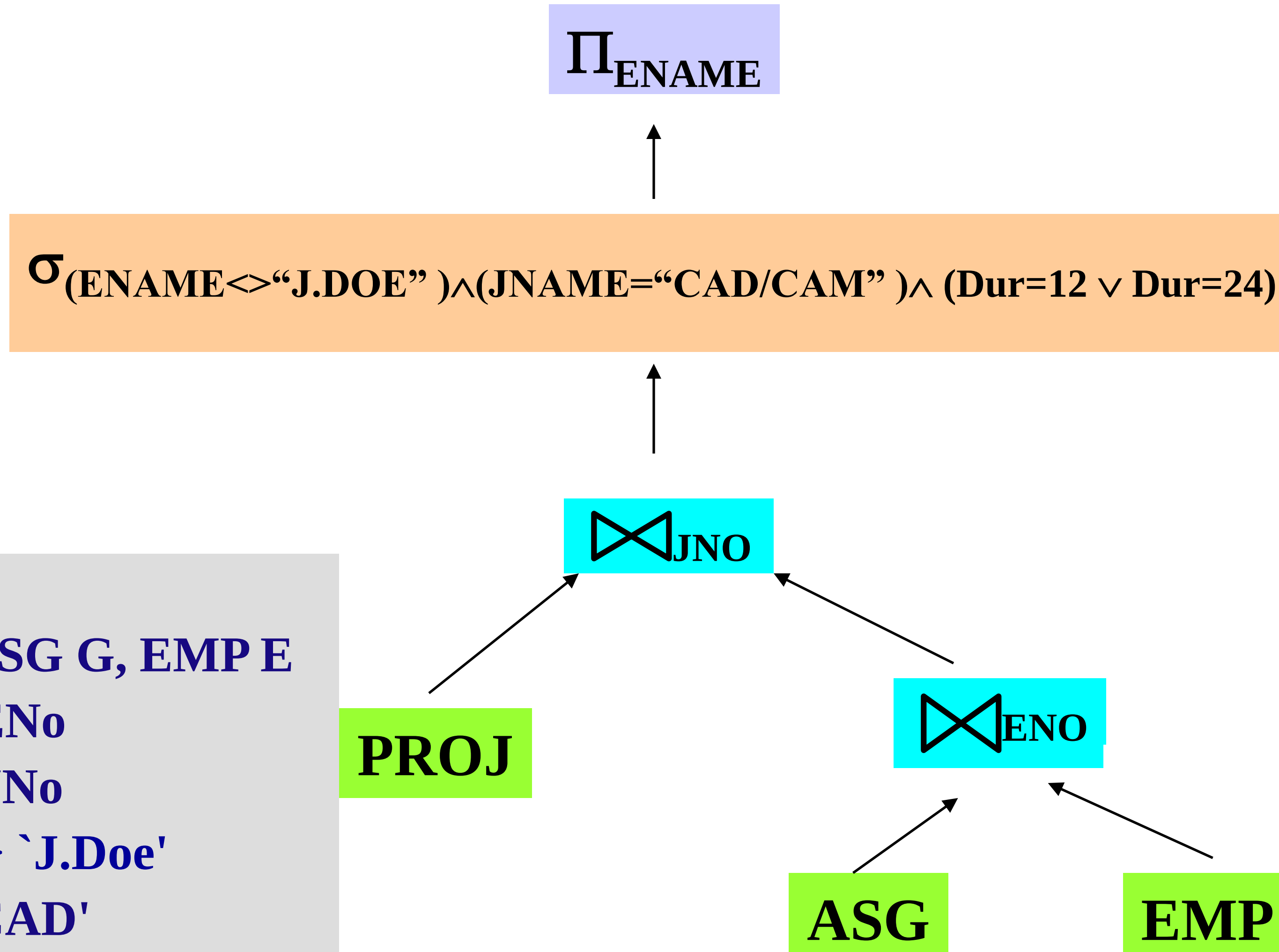
4) Rewriting

- 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:

  $$\mathbf{R \times S \Leftrightarrow S \times R}$$

  $$\mathbf{R \bowtie S \Leftrightarrow S \bowtie R}$$

  $$\mathbf{R \cup S \Leftrightarrow S \cup R}$$

- Associativity of binary operations:

  $$\mathbf{(R \times S) \times T \Leftrightarrow R \times (S \times T)}$$

  $$\mathbf{(R \bowtie S) \bowtie T \Leftrightarrow R \bowtie (S \bowtie T)}$$

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

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

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

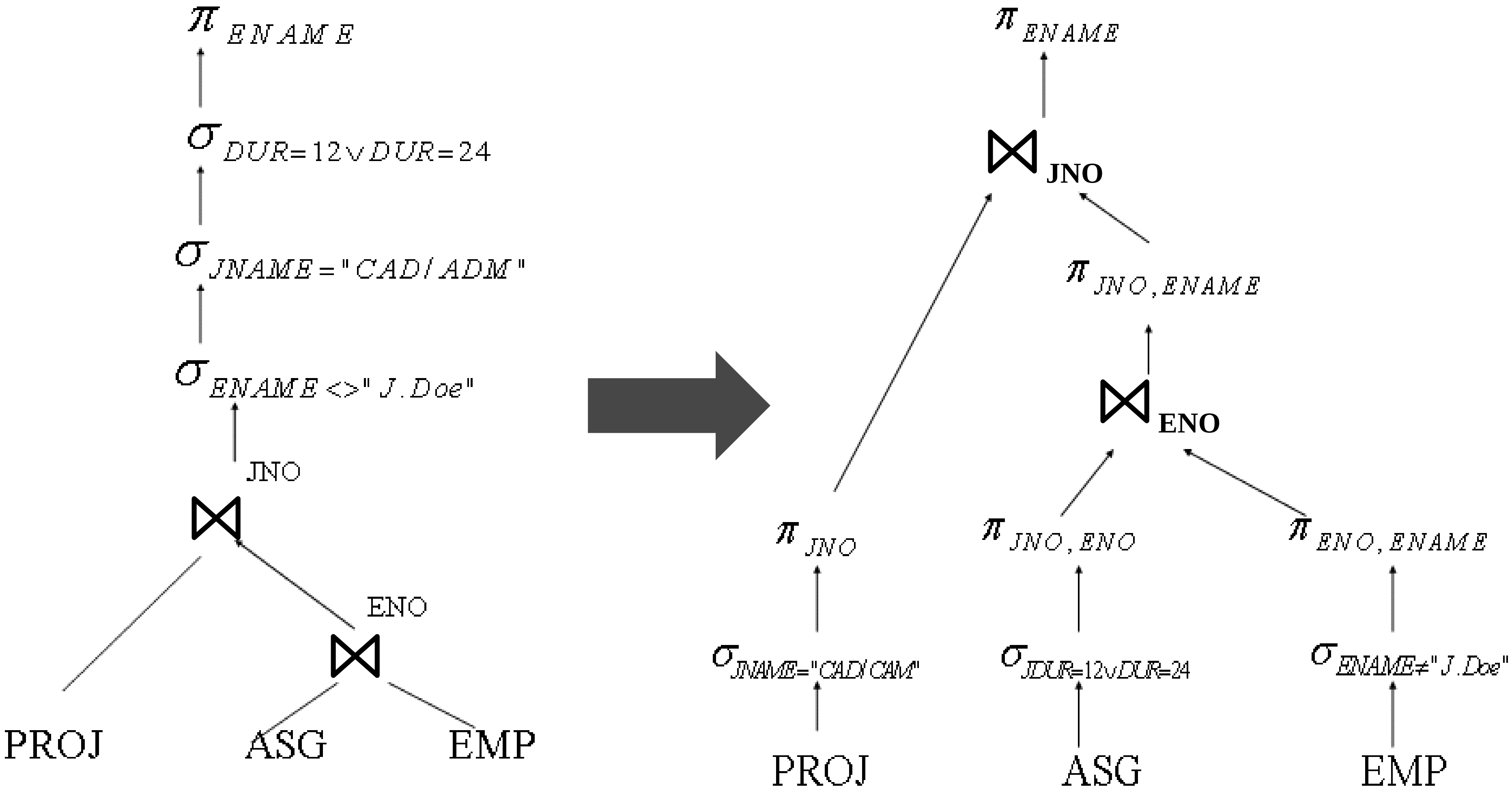
$$\Pi_C(R \bowtie S) \Leftrightarrow \Pi_C(R) \bowtie \Pi_C(S)$$

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

How to use transformation rules to optimize?

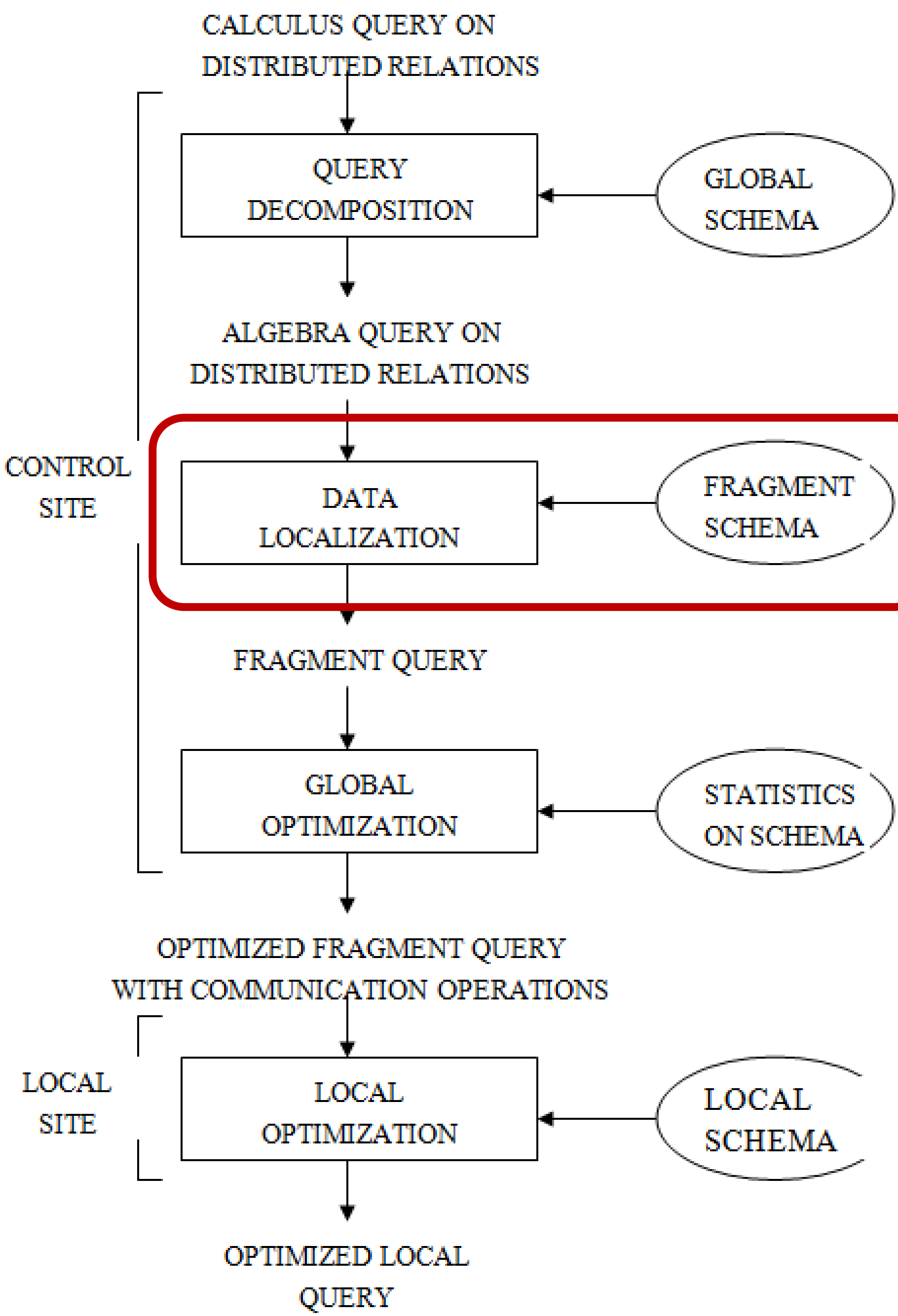
- Unary operations on the same relation may be grouped to access the same relation once
- Unary operations may be commuted with binary operations, so that they may be performed first to reduce the size of intermediate relations
- Binary operations may be reordered

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.



Reduction with Selection for PHF

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

EMP is fragmented into

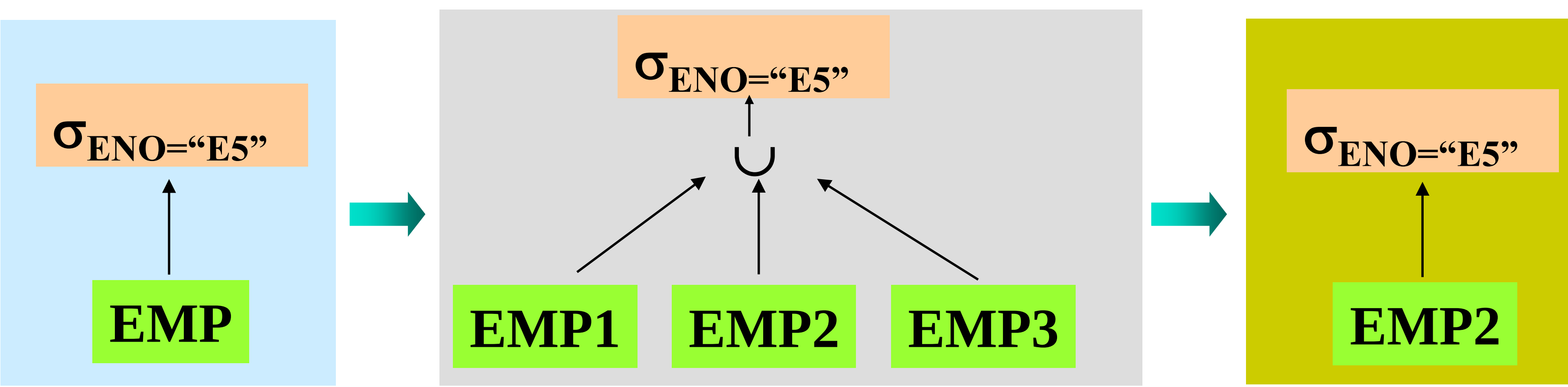
EMP1 = $\sigma_{ENO \leq \text{"E3"}}(EMP)$

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

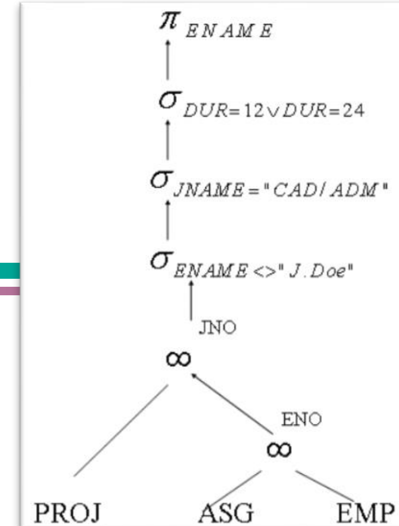
EMP3 = $\sigma_{ENO > \text{"E6"}}(EMP)$

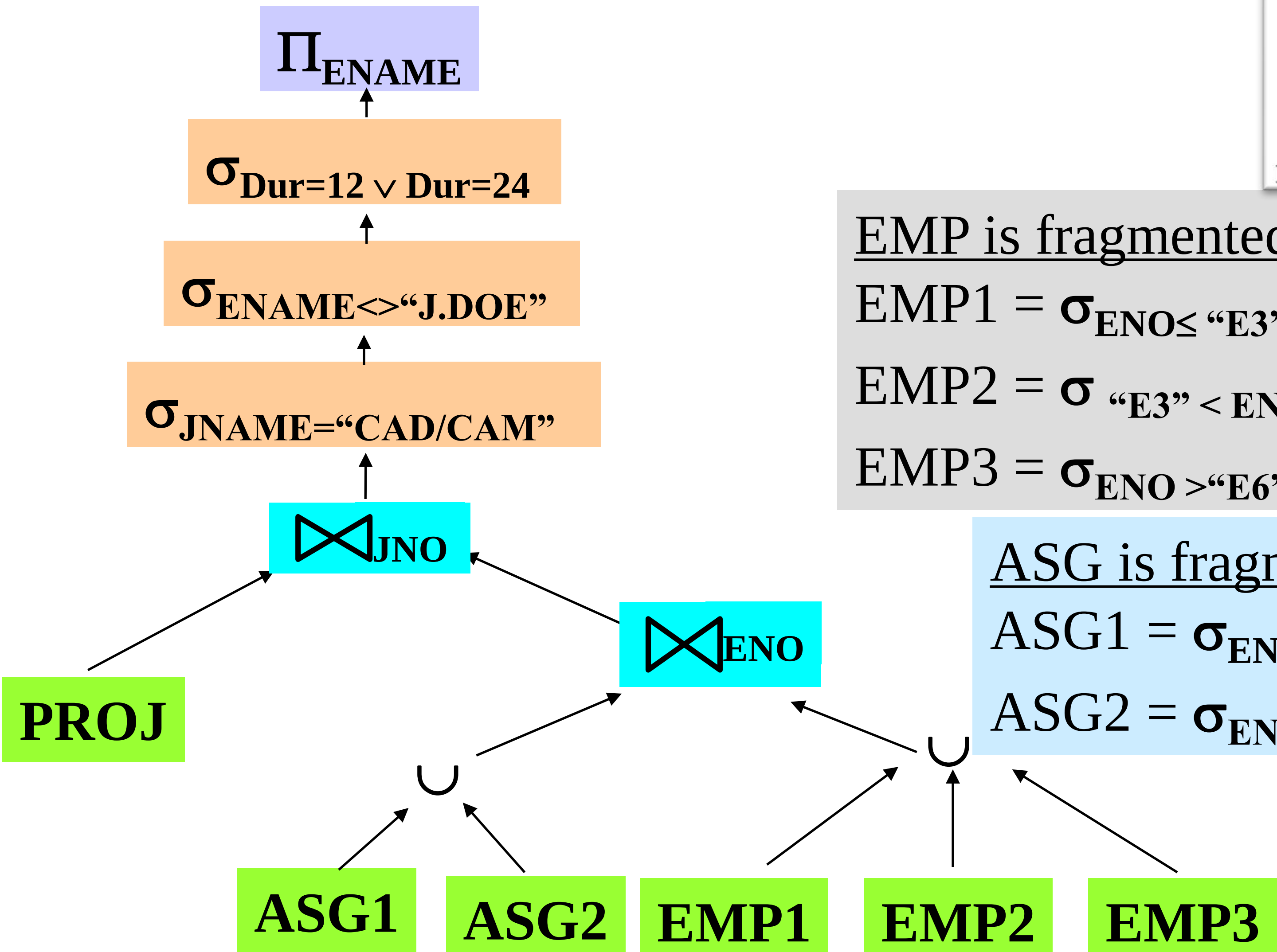
Given Relation R, $F_R = \{R_1, R_2, \ldots, R_n\}$ where $R_j = \sigma_{p_j}(R)$

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



Data Localization - Example





<u>EMP is fragmented into</u>

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

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

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

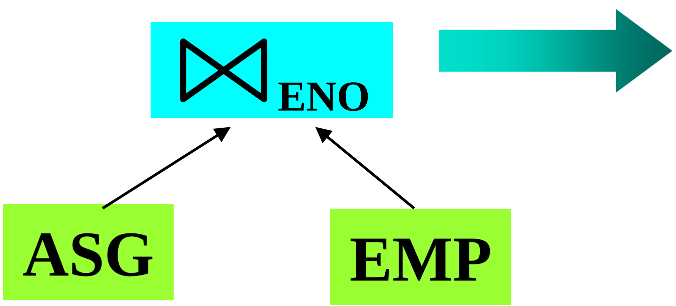
<u>ASG is fragmented into</u>

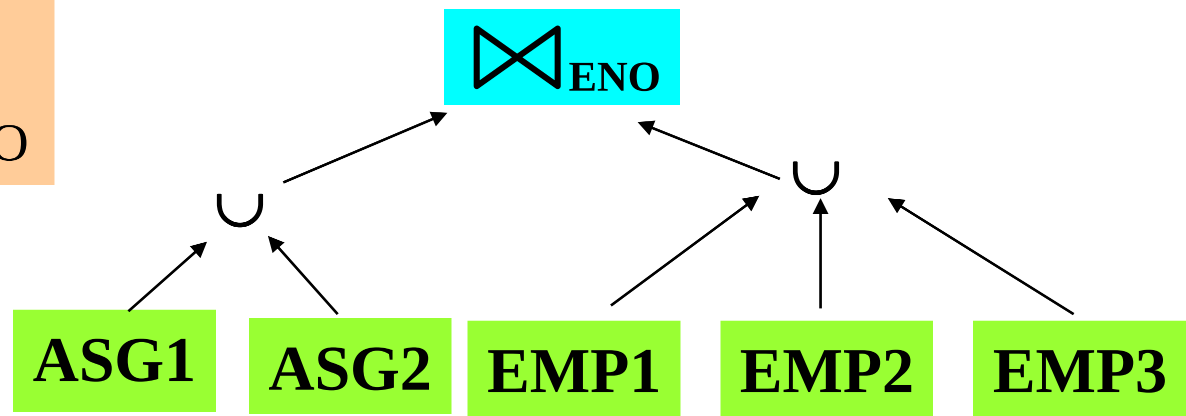
ASG1 = $\sigma_{ENO \leq \text{“E3”}}$ (ASG)

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

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)

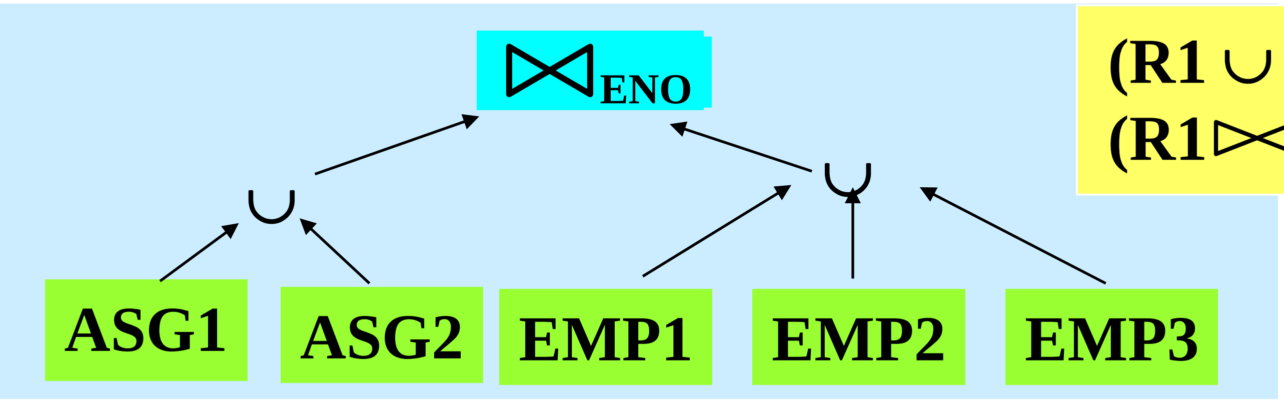
EMP is fragmented into

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

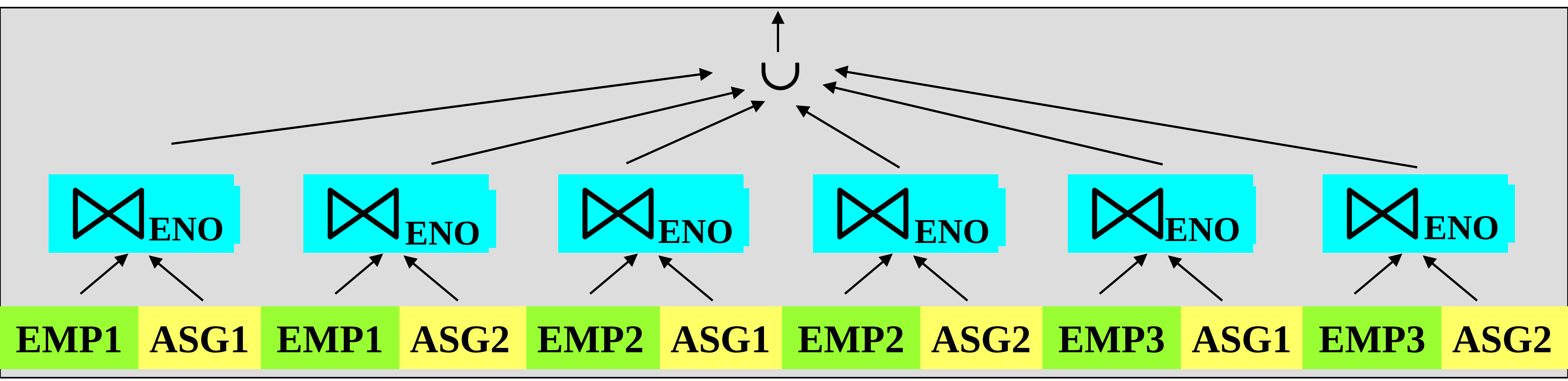
EMP2 = $\sigma_{\text{“E3”} < ENO \leq \text{“E6”}}$ (EMP)

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

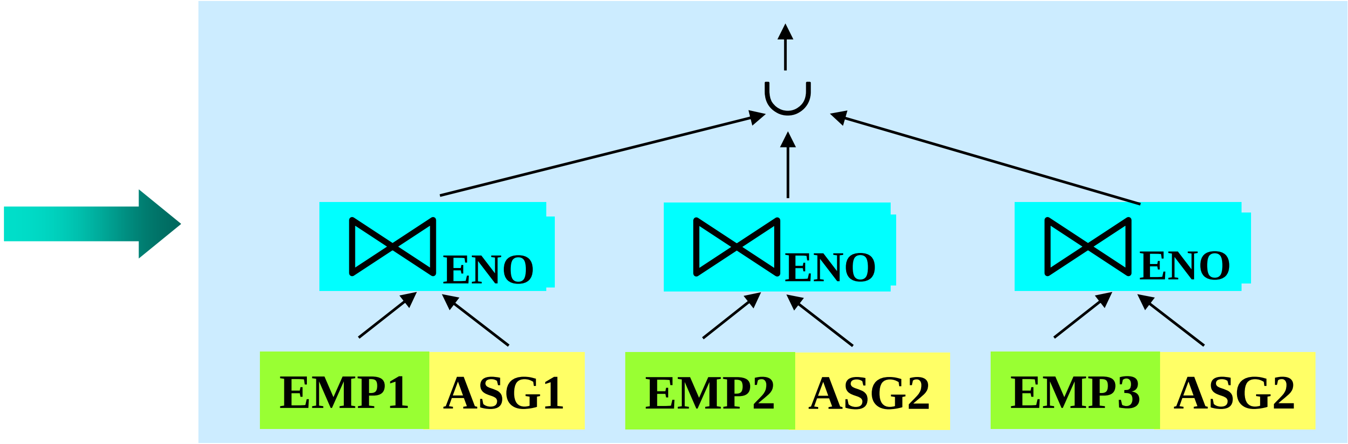
EMP2 = $\sigma_{\text{“E3”} < ENO \le \text{“E6”}}(EMP)$

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

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

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

Reduction with Join for PHF (II)



Given $R_i = \sigma_{p_i}(R)$ and $R_j = \sigma_{p_j}(R)$

$R_i \bowtie Rj = \varnothing$ 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

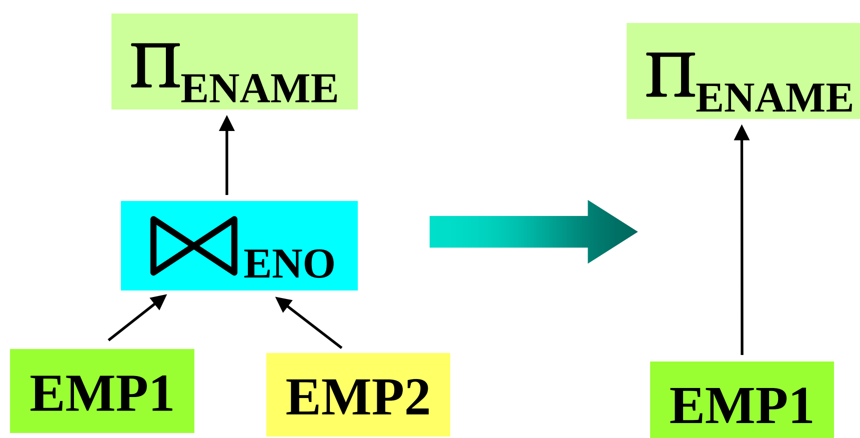
Relation R defined over attributes A = {A1, A2, …, An} vertically fragmented as $R_i = \Pi_{A'}(R)$ where $A' \subseteq A$

$\Pi_D(R_i)$ 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

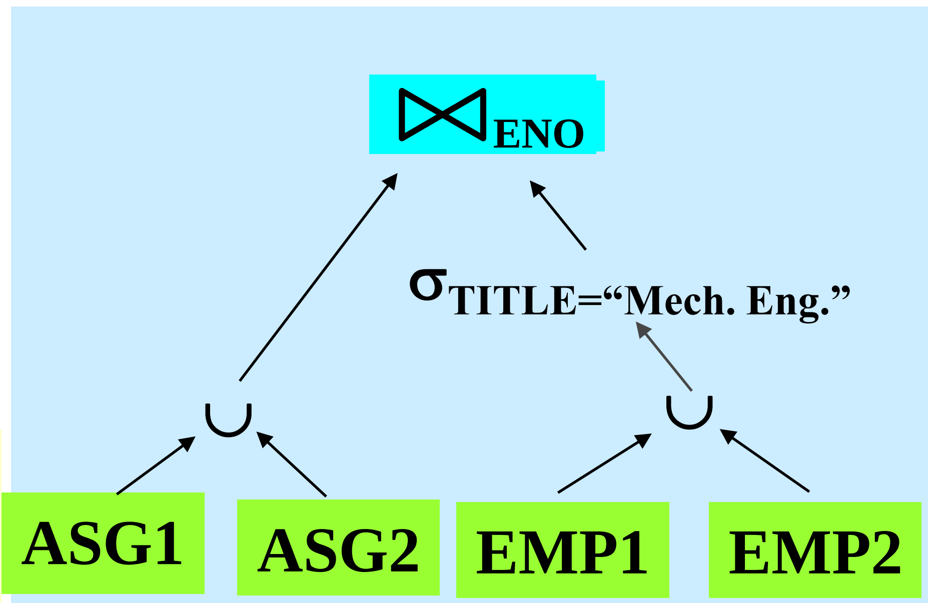
EMP1: $\sigma_{TITLE=\text{“Programmer”}}(EMP)$

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

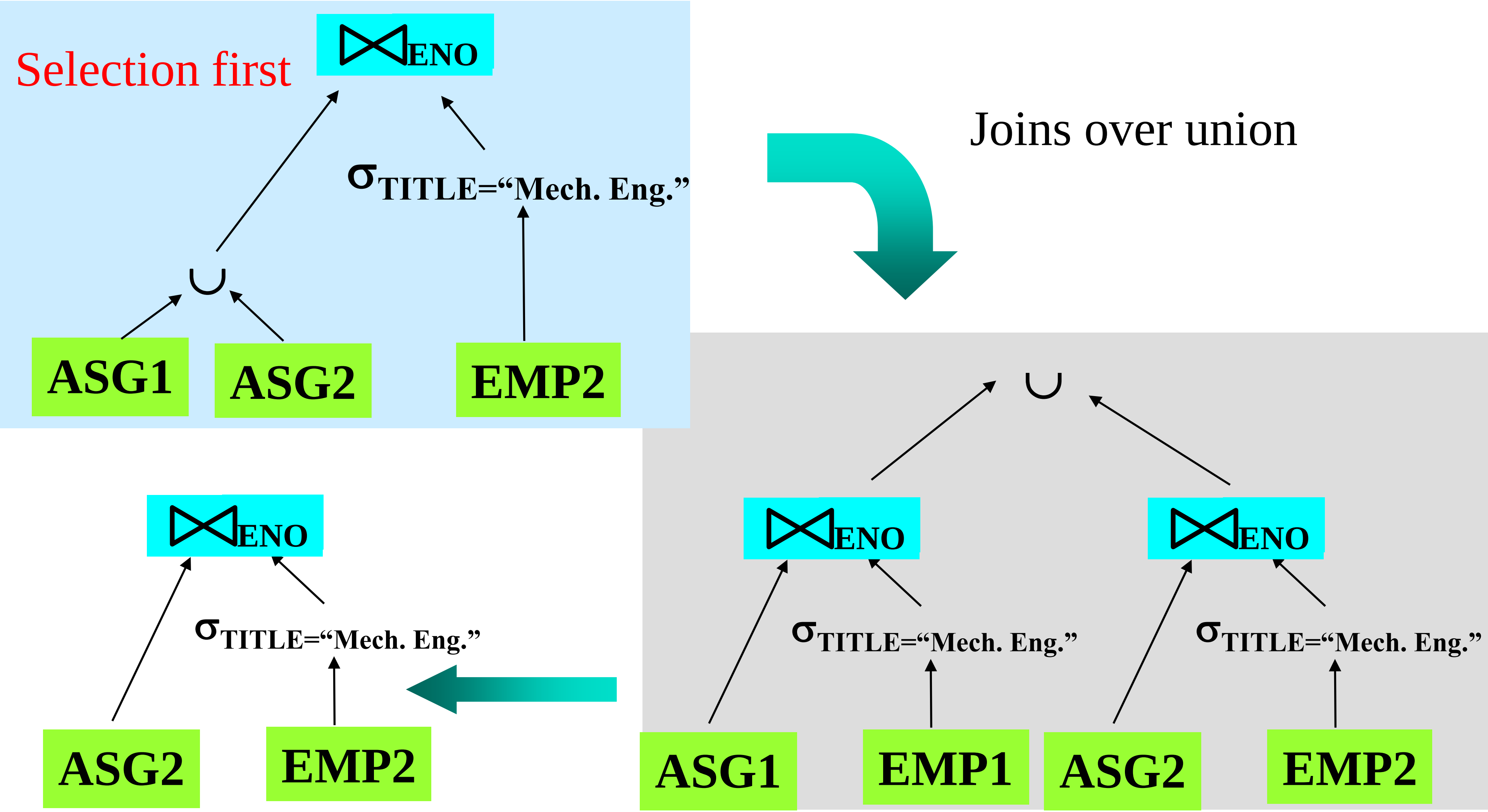
ASG1: $ASG \ltimes_{ENO} EMP1$

ASG2: $ASG \ltimes_{ENO} EMP2$

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

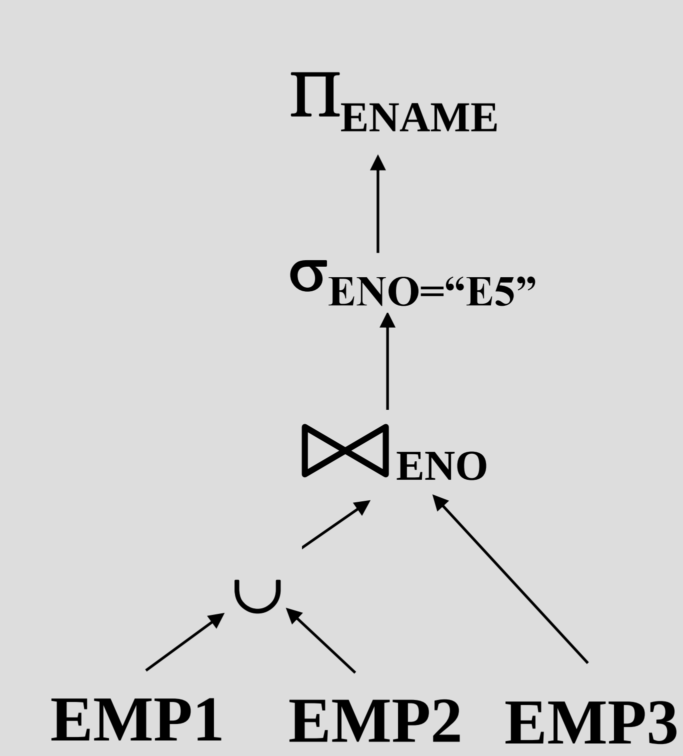
EMP1 = $\sigma_{ENO \leq \text{“E4”}}(\Pi_{ENO,ENAME}(EMP))$

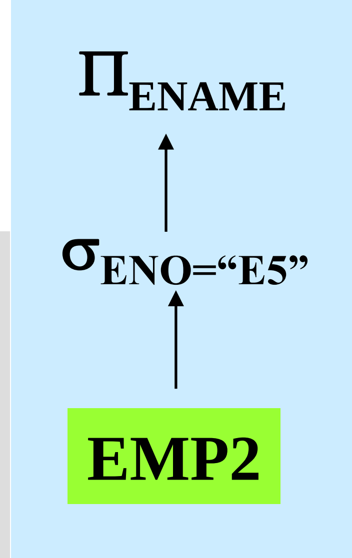
EMP2 = $\sigma_{ENO > \text{“E4”}}(\Pi_{ENO,ENAME}(EMP))$

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

QUERY:

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





Question & Answer