Distributed Database Management Systems

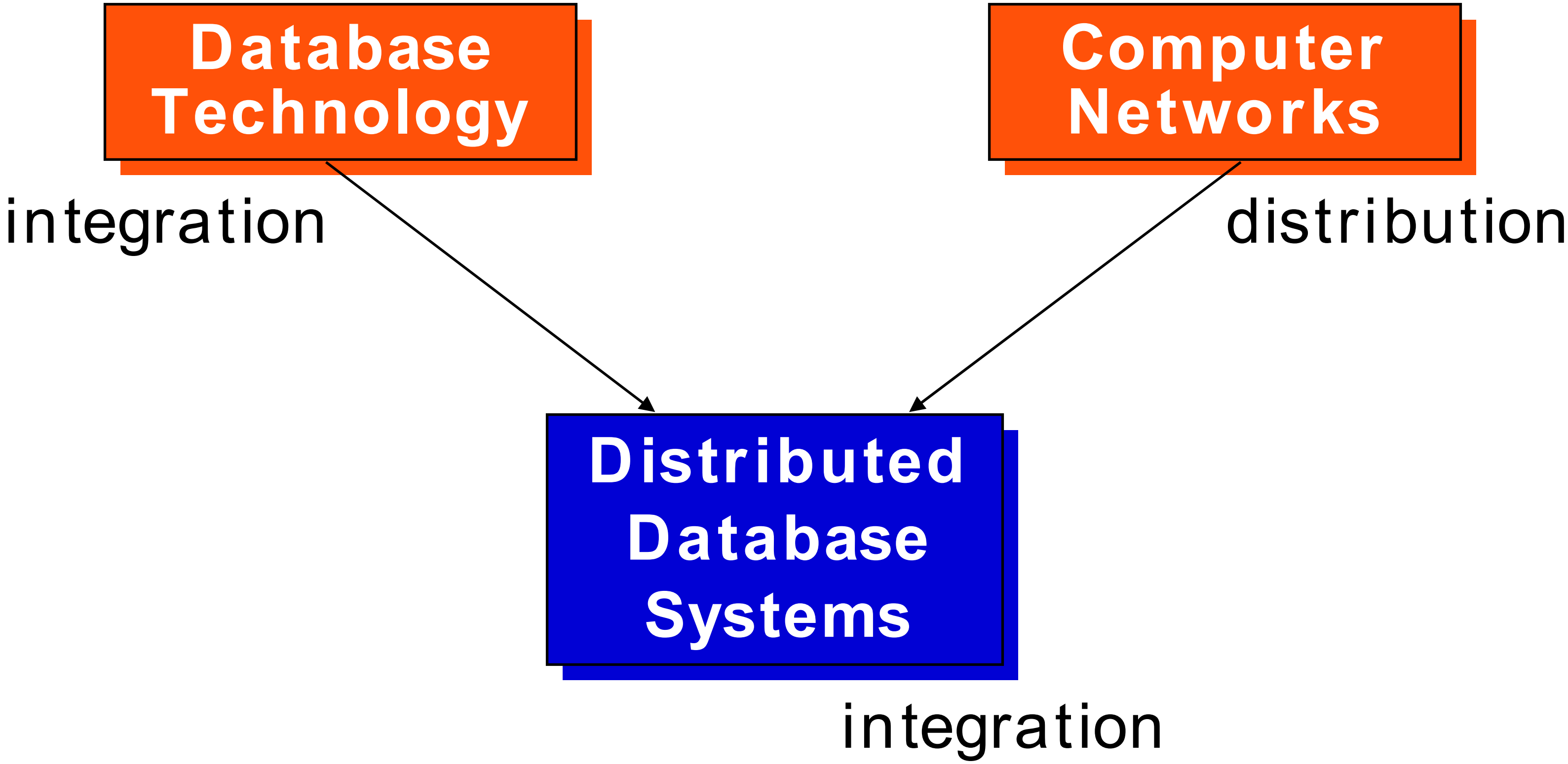
Outline

- Introduction
- Distributed DBMS Architecture
- Distributed Database Design
- Distributed Query Processing
- Distributed Concurrency Control
- Distributed Reliability Protocols

Outline

- Introduction
  - What is a distributed DBMS
  - Problems
  - Current state-of-affairs
- Distributed DBMS Architecture
- Distributed Database Design
- Distributed Query Processing
- Distributed Concurrency Control
- Distributed Reliability Protocols

Motivation



integration ≠ centralization

What is a Distributed Database System?

A distributed database (DDB) is a collection of multiple, *logically interrelated* databases distributed over a *computer network*.

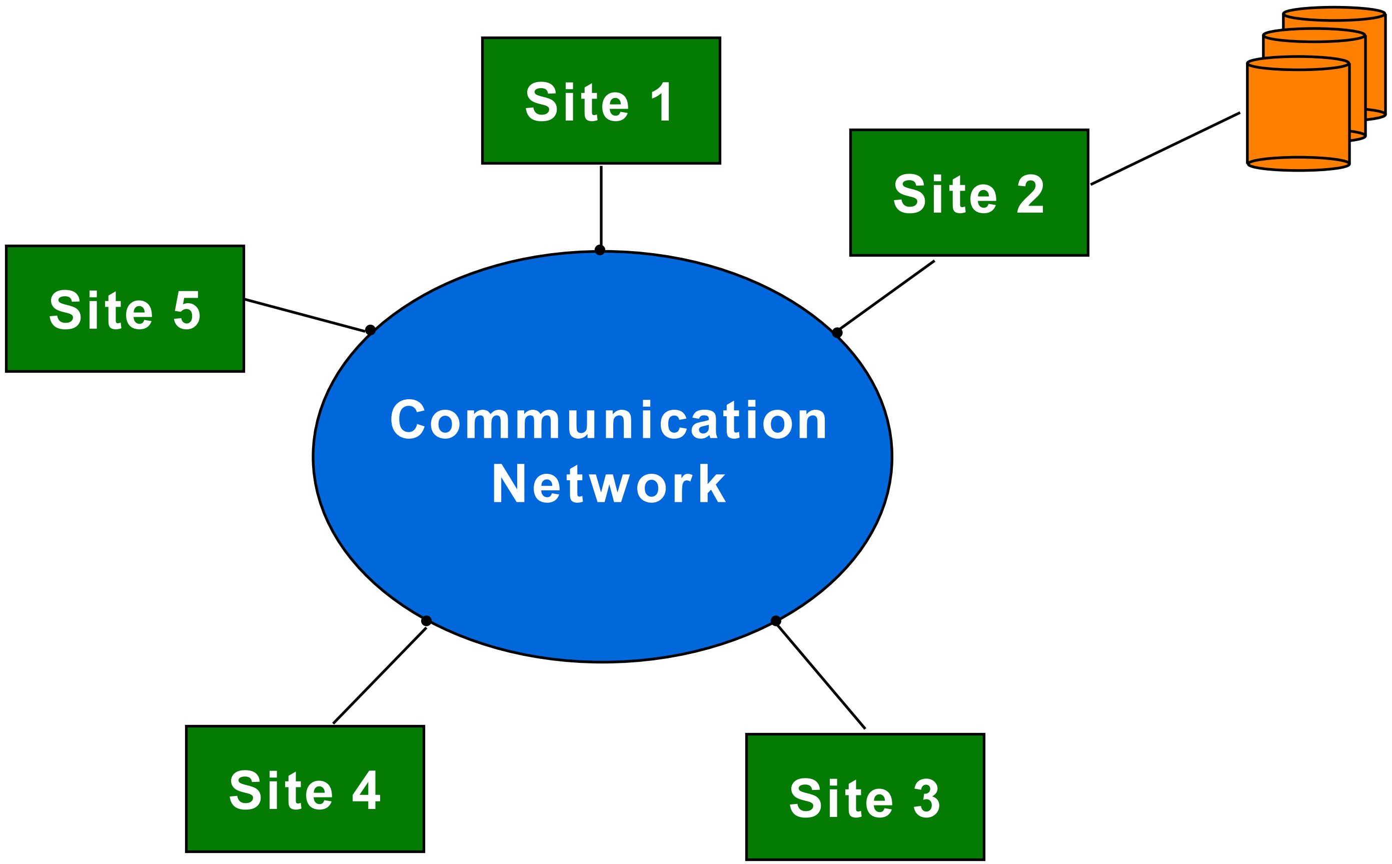
A distributed database management system (D–DBMS) is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users.

Distributed database system (DDBS) = DDB + D–DBMS

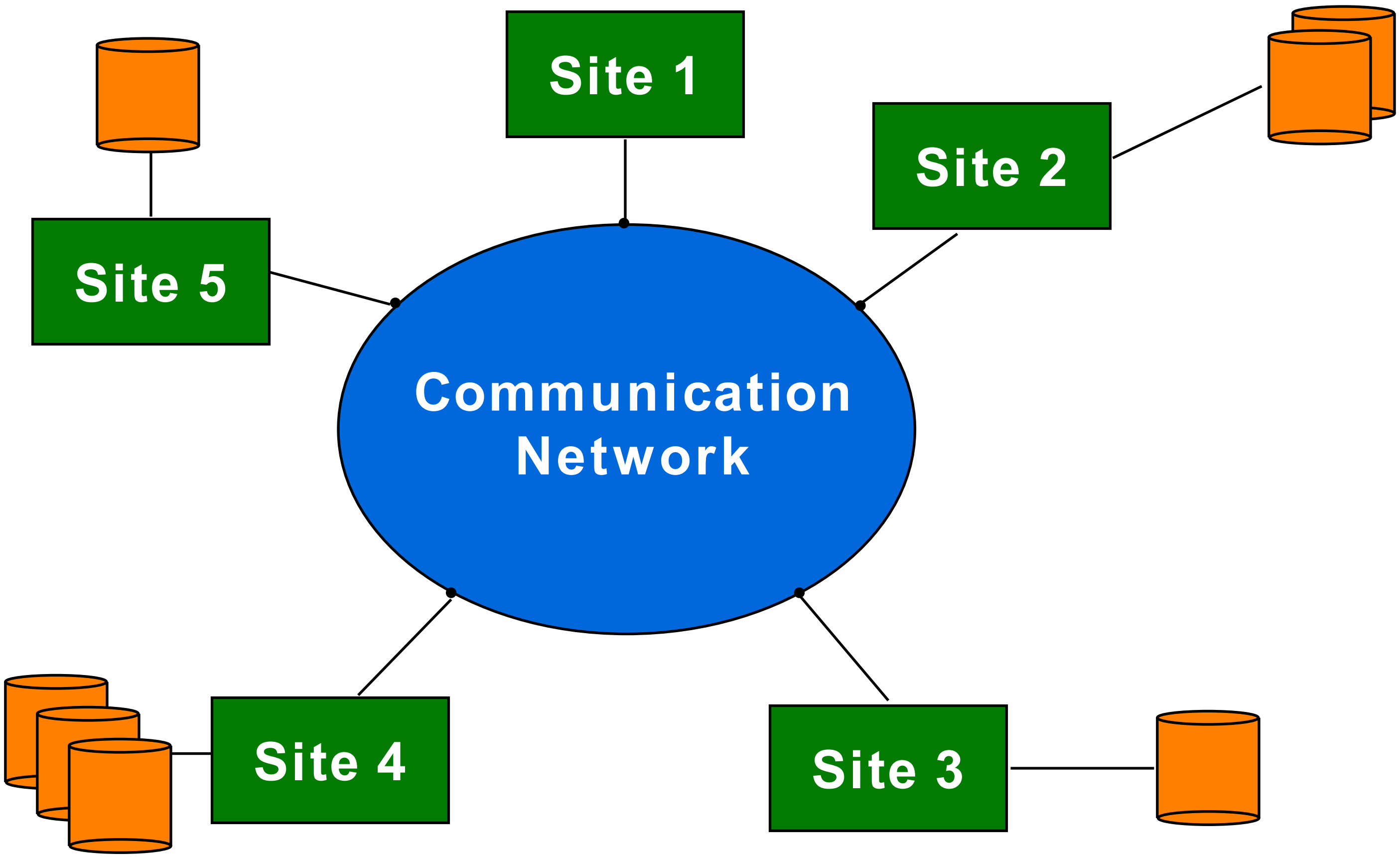
What is not a DDBS?

- A timesharing computer system
- A loosely or tightly coupled multiprocessor system
- A database system which resides at one of the nodes of a network of computers - this is a centralized database on a network node

Centralized DBMS on a Network



Distributed DBMS Environment



Implicit Assumptions

- Data stored at a number of sites ⇨ each site *logically* consists of a single processor.
- Processors at different sites are interconnected by a computer network ⇨ no multiprocessors
  - parallel database systems
- Distributed database is a database, not a collection of files ⇨ data logically related as exhibited in the users' access patterns
  - relational data model
- D-DBMS is a full-fledged DBMS
  - not remote file system, not a TP system

Distributed DBMS Promises

❶ Transparent management of distributed, fragmented, and replicated data

❷ Improved reliability/availability through distributed transactions

❸ Improved performance

❹ Easier and more economical system expansion

Transparency

- Transparency is the separation of the higher level semantics of a system from the lower level implementation issues.
- Fundamental issue is to provide

  data independence

  in the distributed environment

  - Network (distribution) transparency
  - Replication transparency
  - Fragmentation transparency
    - horizontal fragmentation: selection
    - vertical fragmentation: projection
    - hybrid

Example

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

ASG

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P5	Engineer	23
E8	P3	Manager	40

PROJ

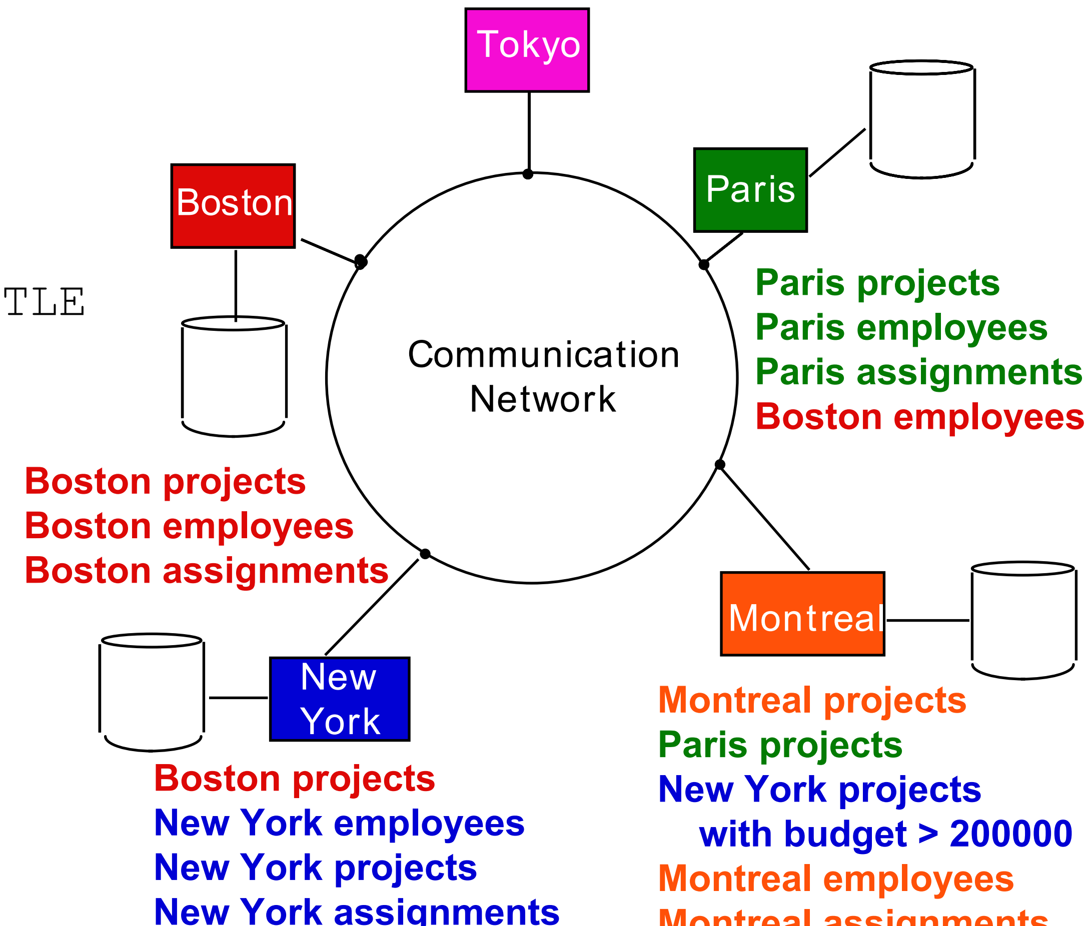
PNO	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop.	135000
P3	CAD/CAM	250000
P4	Maintenance	310000

PAY

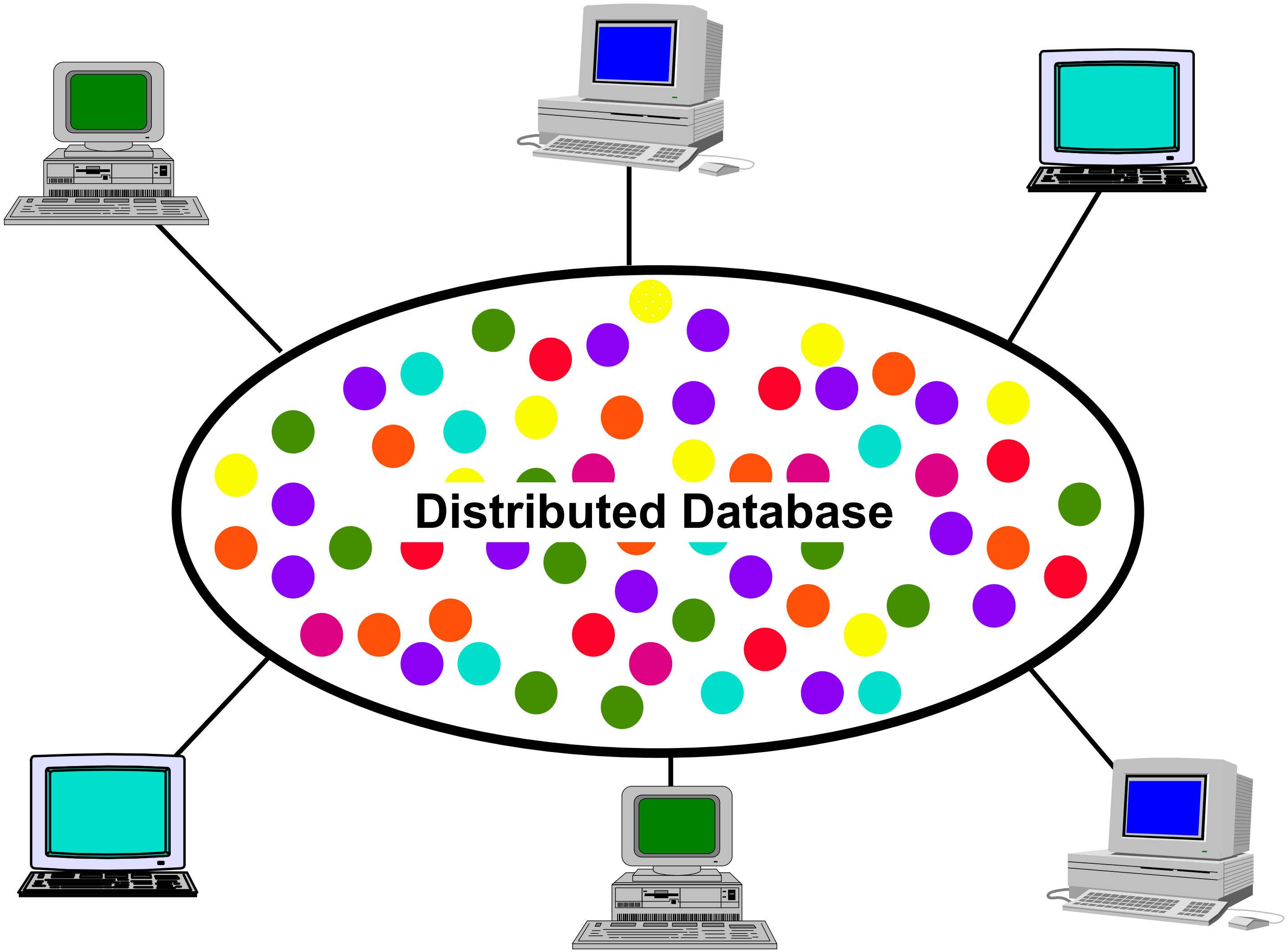
TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000
Mech. Eng.	27000
Programmer	24000

Transparent Access

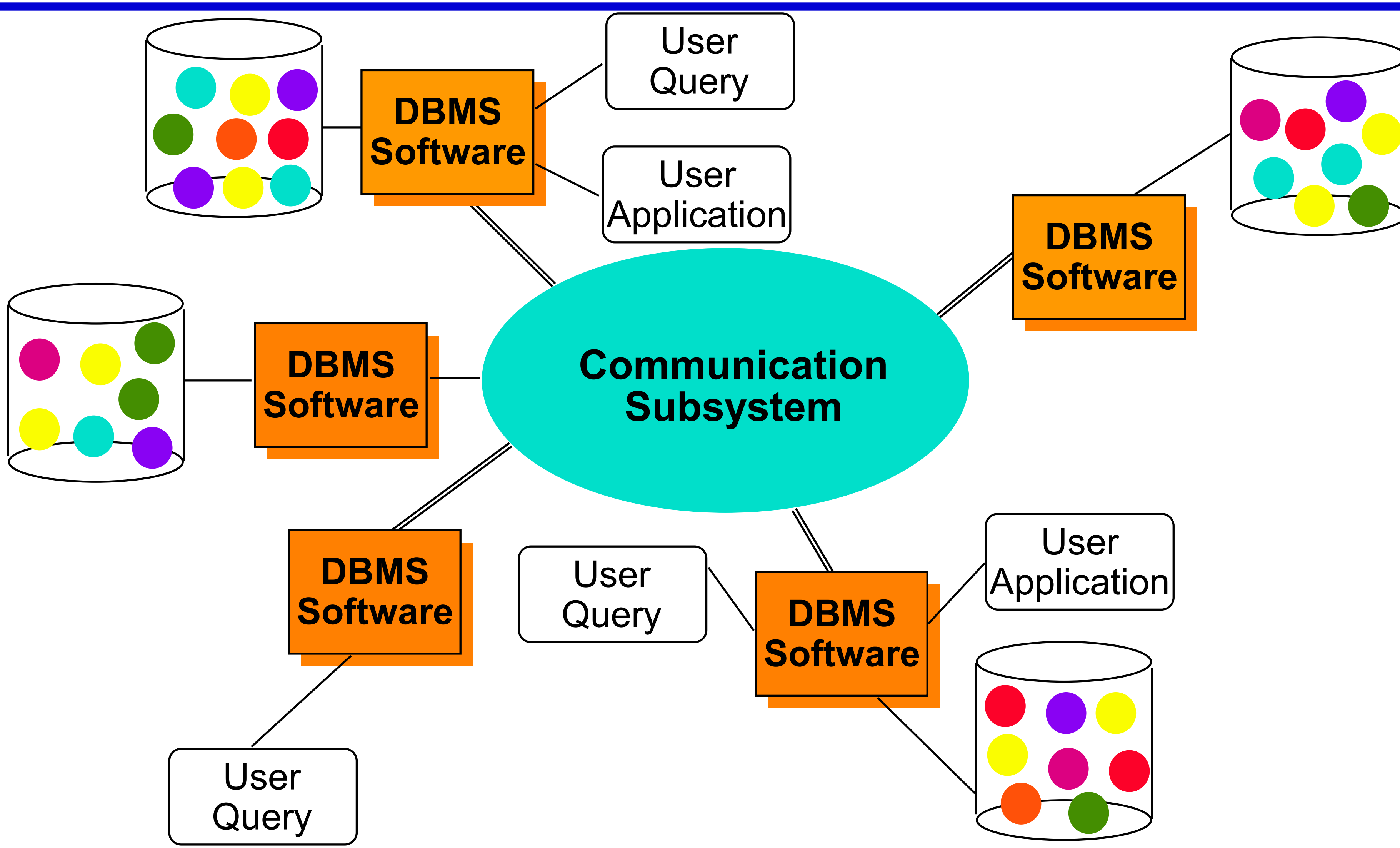
```
SELECT ENAME,SAL
FROM   EMP,ASG,PAY
WHERE  DUR > 12
AND    EMP.ENO = ASG.ENO
AND    PAY.TITLE = EMP.TITLE
```



Distributed Database – User View



Distributed DBMS - Reality



Potentially Improved Performance

- Proximity of data to its points of use
  - Requires some support for fragmentation and replication
- Parallelism in execution
  - Inter-query parallelism
  - Intra-query parallelism

Parallelism Requirements

- Have as much of the data required by *each* application at the site where the application executes
  - Full replication
- How about updates?
  - Updates to replicated data requires implementation of distributed concurrency control and commit protocols

System Expansion

- Issue is database scaling
- Emergence of microprocessor and workstation technologies
  - Demise of Grosh's law
  - Client-server model of computing
- Data communication cost vs telecommunication cost

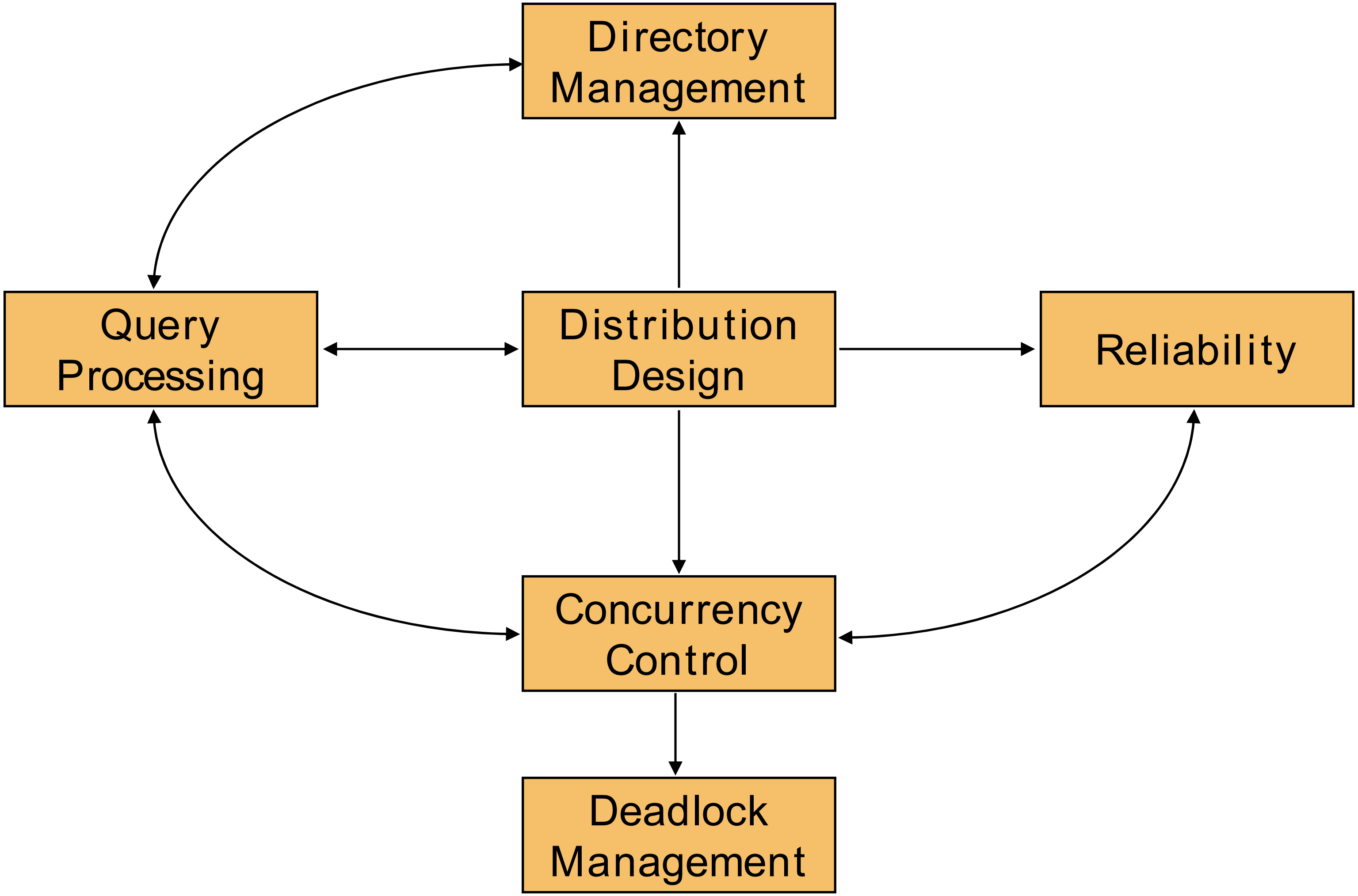
Distributed DBMS Issues

- **Distributed Database Design**
  - how to distribute the database
  - replicated & non-replicated database distribution
  - a related problem in directory management
- **Query Processing**
  - convert user transactions to data manipulation instructions
  - optimization problem
  - min{cost = data transmission + local processing}
  - general formulation is NP-hard

Distributed DBMS Issues

- **Concurrency Control**
  - synchronization of concurrent accesses
  - consistency and isolation of transactions' effects
  - deadlock management
- **Reliability**
  - how to make the system resilient to failures
  - atomicity and durability

Relationship Between Issues



Outline

- Introduction
- Distributed DBMS Architecture
- Distributed Database Design
  - Fragmentation
  - Data Placement
- Distributed Query Processing
- Distributed Concurrency Control
- Distributed Reliability Protocols

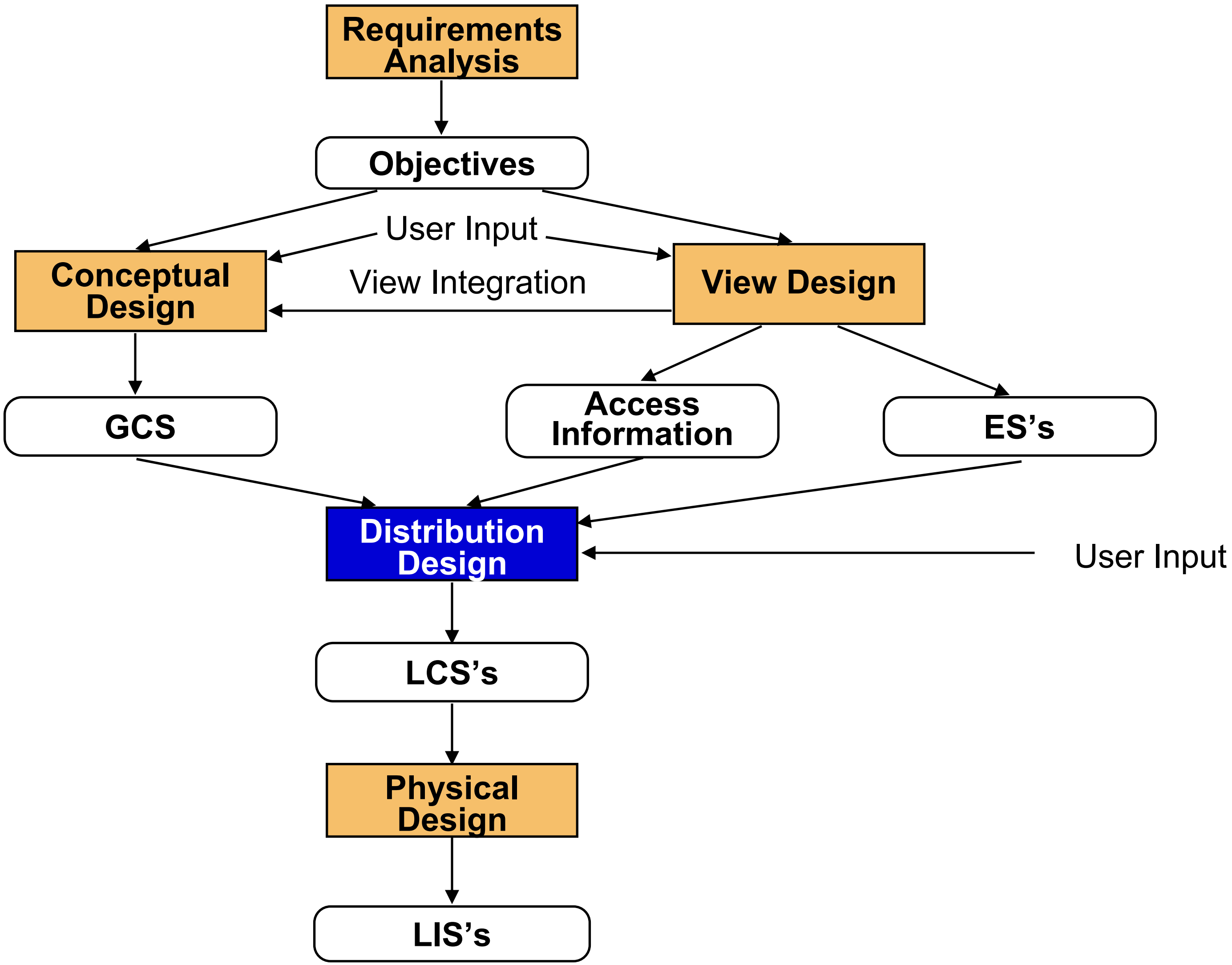
Design Problem

- In the general setting :
  - Making decisions about the placement of data and programs across the sites of a computer network as well as possibly designing the network itself.
- In Distributed DBMS, the placement of applications entails
  - placement of the distributed DBMS software; and
  - placement of the applications that run on the database

Distribution Design

- Top-down
  - mostly in designing systems from scratch
  - mostly in homogeneous systems
- Bottom-up
  - when the databases already exist at a number of sites

Top-Down Design



Distribution Design Issues

1. Why fragment at all?
2. How to fragment?
3. How much to fragment?
4. How to test correctness?
5. How to allocate?
6. Information requirements?

Fragmentation

- Can't we just distribute relations?
- What is a reasonable unit of distribution?
  - relation
    - views are subsets of relations $\Rightarrow$ locality
    - extra communication
  - fragments of relations (sub-relations)
    - concurrent execution of a number of transactions that access different portions of a relation
    - views that cannot be defined on a single fragment will require extra processing
    - semantic data control (especially integrity enforcement) more difficult

Fragmentation Alternatives – Horizontal

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

$PROJ_1$: projects with budgets less than $200,000

$PROJ_2$: projects with budgets greater than or equal to $200,000

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York

$PROJ_2$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

Fragmentation Alternatives – Vertical

$PROJ_1$: information about project budgets

$PROJ_2$: information about project names and locations

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

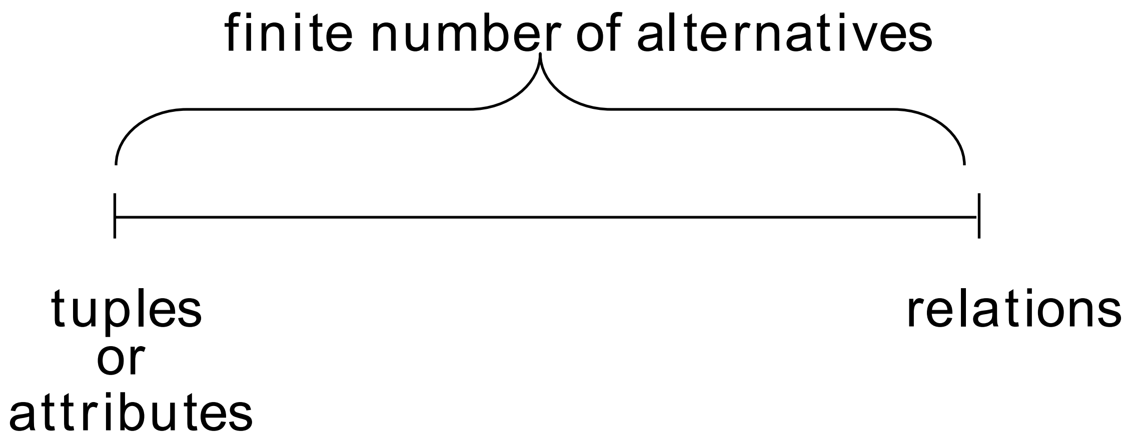
$PROJ_1$

PNO	BUDGET
P1	150000
P2	135000
P3	250000
P4	310000
P5	500000

$PROJ_2$

PNO	PNAME	LOC
P1	Instrumentation	Montreal
P2	Database Develop.	New York
P3	CAD/CAM	New York
P4	Maintenance	Paris
P5	CAD/CAM	Boston

Degree of Fragmentation



Finding the suitable level of partitioning within this range

Correctness of Fragmentation

- Completeness
  - Decomposition of relation R into fragments R_1, R_2, ..., R_n is complete iff each data item in R can also be found in some R_i
- Reconstruction
  - If relation R is decomposed into fragments R_1, R_2, ..., R_n, then there should exist some relational operator ∇ such that

$$R = \nabla_{1 \le i \le n} R_i$$

- Disjointness
  - If relation R is decomposed into fragments R_1, R_2, ..., R_n, and data item d_i is in R_j, then d_i should not be in any other fragment R_k ($k \neq j$).

Allocation Alternatives

- Non-replicated
  - partitioned : each fragment resides at only one site
- Replicated
  - fully replicated : each fragment at each site
  - partially replicated : each fragment at some of the sites
- Rule of thumb:

If $\frac{\text{read - only queries}}{\text{update queries}} \geq 1$ replication is advantageous,

otherwise replication may cause problems

Fragmentation

- Horizontal Fragmentation (HF)
  - Primary Horizontal Fragmentation (PHF)
  - Derived Horizontal Fragmentation (DHF)
- Vertical Fragmentation (VF)
- Hybrid Fragmentation (HF)

Primary Horizontal Fragmentation

Definition :

$$R_j = \sigma_{F_j}(R), \ 1 \leq j \leq w$$

where F_j is a selection formula.

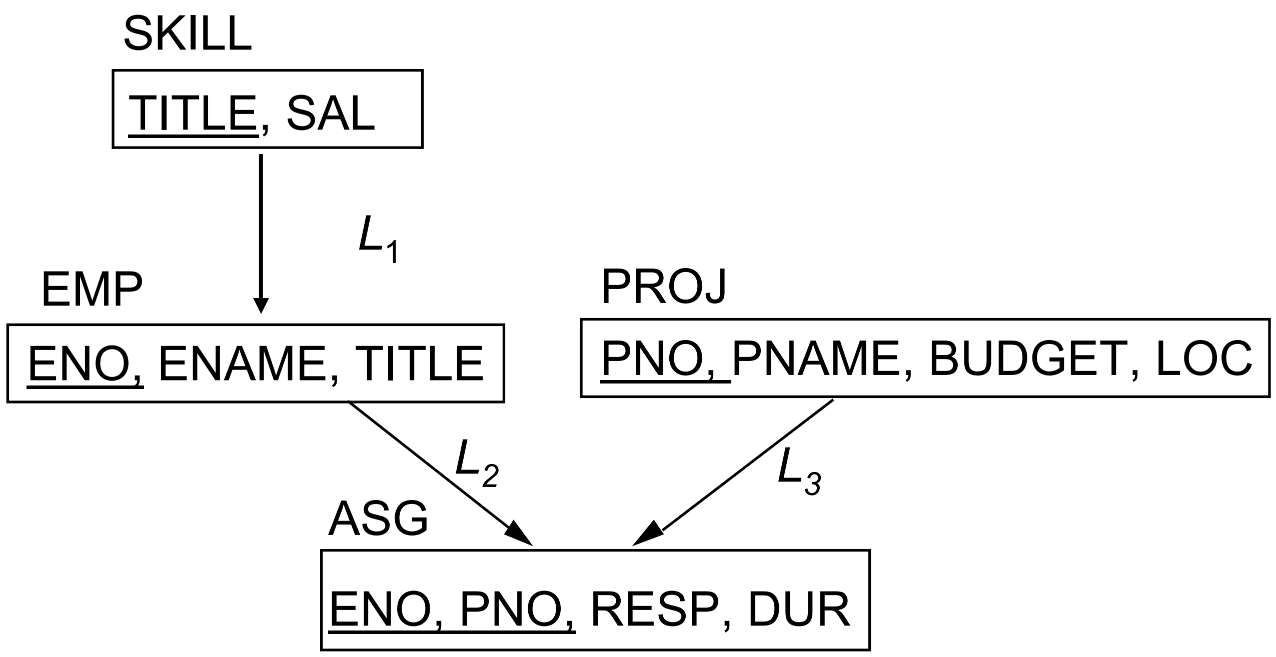
Therefore,

A horizontal fragment R_i of relation R consists of all the tuples of R which satisfy a predicate p_i.

$$\Downarrow$$

Given a set of predicates *M,* there are as many horizontal fragments of relation *R* as there are predicates.

PHF – Example



- Two candidate relations : PAY and PROJ

PHF – Example

PAY_1

TITLE	SAL
Mech. Eng.	27000
Programmer	24000

PAY_2

TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000

PHF – Example

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal

$PROJ_2$

PNO	PNAME	BUDGET	LOC
P2	Database Develop.	135000	New York

$PROJ_4$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York

$PROJ_6$

PNO	PNAME	BUDGET	LOC
P4	Maintenance	310000	Paris

PHF – Correctness

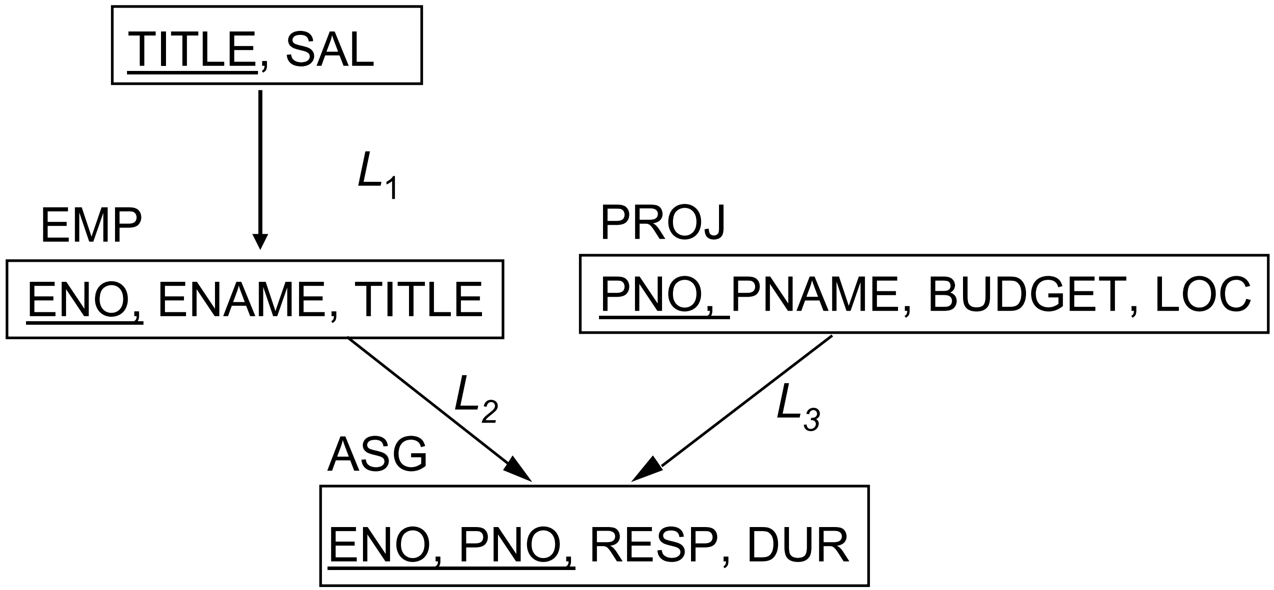
- Completeness
  - Since the set of predicates is complete and minimal, the selection predicates are complete
- Reconstruction
  - If relation R is fragmented into $F_R = \{R_1, R_2, \ldots, R_r\}$

$$R = \cup_{\forall R_i \in F_R} R_i$$

- Disjointness
  - Predicates that form the basis of fragmentation should be mutually exclusive.

Derived Horizontal Fragmentation

- Defined on a member relation of a link according to a selection operation specified on its owner.
  - Each link is an equijoin.
  - Equijoin can be implemented by means of semijoins.



DHF – Definition

Given a link L where *owner*(L)=S and *member*(L)=R, the derived horizontal fragments of R are defined as

$$R_i = R \ltimes_F S_i,\ 1 \le i \le w$$

where w is the maximum number of fragments that will be defined on R and

$$S_i = \sigma_{F_i}(S)$$

where F_i is the formula according to which the primary horizontal fragment S_i is defined.

DHF – Example

Given link L_1 where owner(L_1)=SKILL and member(L_1)=EMP

$EMP_1 = EMP \ltimes SKILL_1$

$EMP_2 = EMP \ltimes SKILL_2$

where

$SKILL_1 = \sigma_{SAL \leq 30000}(SKILL)$

$SKILL_2 = \sigma_{SAL > 30000}(SKILL)$

EMP_1

ENO	ENAME	TITLE
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E7	R. Davis	Mech. Eng.

EMP_2

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E8	J. Jones	Syst. Anal.

DHF – Correctness

- Completeness
  - Referential integrity
  - Let *R* be the member relation of a link whose owner is relation *S* which is fragmented as $F_S = \{S_1, S_2, ..., S_n\}$. Furthermore, let A be the join attribute between *R* and *S*. Then, for each tuple *t* of *R*, there should be a tuple *t'* of *S* such that

    $$t[A]=t'[A]$$
- Reconstruction
  - Same as primary horizontal fragmentation.
- Disjointness
  - Simple join graphs between the owner and the member fragments.

Vertical Fragmentation

- Has been studied within the centralized context
  - design methodology
  - physical clustering
- More difficult than horizontal, because more alternatives exist.

  Two approaches :
  - grouping
    - attributes to fragments
  - splitting
    - relation to fragments

Vertical Fragmentation

- Overlapping fragments
  - grouping
- Non-overlapping fragments
  - splitting
  - We do not consider the replicated key attributes to be overlapping.
- Advantage:
  - Easier to enforce functional dependencies (for integrity checking etc.)

VF – Correctness

A relation R, defined over attribute set A and key K, generates the vertical partitioning $F_R = \{R_1, R_2, \ldots, R_r\}$.

- Completeness
  - The following should be true for A:

$$A = \cup A_{R_i}$$

- Reconstruction
  - Reconstruction can be achieved by

$$R = \bowtie_K R_i \; \forall R_i \in F_R$$

- Disjointness
  - TID's are not considered to be overlapping since they are maintained by the system
  - Duplicated keys are not considered to be overlapping

Fragment Allocation

- Problem Statement

  Given

$F = \{F_1, F_2, \ldots, F_n\}$	fragments
$S = \{S_1, S_2, \ldots, S_m\}$	network sites
$Q = \{q_1, q_2, \ldots, q_q\}$	applications

  Find the "optimal" distribution of F to S.

- Optimality
  - Minimal cost
    - Communication + storage + processing (read & update)
    - Cost in terms of time (usually)
  - Performance

    Response time and/or throughput
  - Constraints
    - Per site constraints (storage & processing)

Allocation Model

General Form

min(Total Cost)

subject to

response time constraint

storage constraint

processing constraint

Decision Variable

$$x_{ij} = \begin{cases} 1 \text{ if fragment } F_i \text{ is stored at site } S_j \\ 0 \text{ otherwise} \end{cases}$$

Allocation Model

- Total Cost

$$\sum_{\text{all queries}} \text{query processing cost} +$$

$$\sum_{\text{all sites}} \sum_{\text{all fragments}} \text{cost of storing a fragment at a site}$$

- Storage Cost (of fragment F_j at S_k)

$$(\text{unit storage cost at } S_k) * (\text{size of } F_j) * x_{jk}$$

- Query Processing Cost (for one query)

processing component + transmission component

Allocation Model

- Query Processing Cost

  Processing component

  access cost + integrity enforcement cost + concurrency control cost

  - Access cost

    $$\sum_{\text{all sites}} \sum_{\text{all fragments}} (\text{no. of update accesses} + \text{no. of read accesses}) * x_{ij} * \text{local processing cost at a site}$$

  - Integrity enforcement and concurrency control costs
    - Can be similarly calculated

Allocation Model

- Query Processing Cost

  Transmission component

  cost of processing updates + cost of processing retrievals

  - Cost of updates

    $$\sum_{\text{all sites}} \sum_{\text{all fragments}} \text{update message cost} + \sum_{\text{all sites}} \sum_{\text{all fragments}} \text{acknowledgment cost}$$

  - Retrieval Cost

    $$\sum_{\text{all fragments}} \min_{\text{all sites}} (\text{cost of retrieval command} + \text{cost of sending back the result})$$

Allocation Model

- Constraints
  - Response Time

    execution time of query ≤ max. allowable response time for that query
  - Storage Constraint (for a site)

$$\sum_{\text{all fragments}} \text{storage requirement of a fragment at that site} \leq \text{storage capacity at that site}$$

  - Processing constraint (for a site)

$$\sum_{\text{all queries}} \text{processing load of a query at that site} \leq \text{processing capacity of that site}$$

Allocation Model

- Solution Methods
  - FAP is NP-complete
  - DAP also NP-complete
- Heuristics based on
  - single commodity warehouse location (for FAP)
  - knapsack problem
  - branch and bound techniques
  - network flow

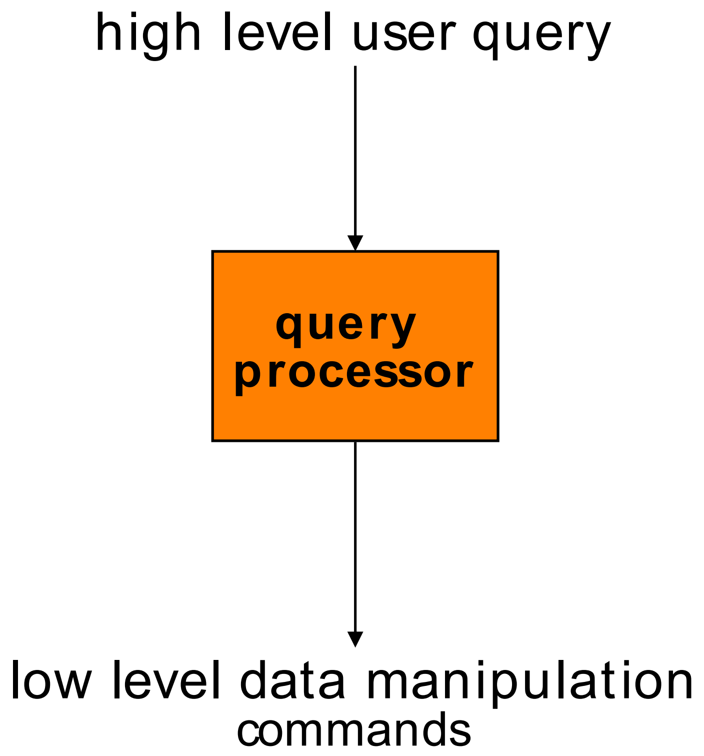
Allocation Model

- Attempts to reduce the solution space
  - assume all candidate partitionings known; select the “best” partitioning
  - ignore replication at first
  - sliding window on fragments

Outline

- Introduction
- Distributed DBMS Architecture
- Distributed Database Design
- Distributed Query Processing
  - Query Processing Methodology
  - Distributed Query Optimization
- Distributed Concurrency Control
- Distributed Reliability Protocols

Query Processing



Query Processing Components

- Query language that is used
  - SQL: "intergalactic dataspeak"
- Query execution methodology
  - The steps that one goes through in executing high-level (declarative) user queries.
- Query optimization
  - How do we determine the "best" execution plan?

Selecting Alternatives

```
SELECT  ENAME
FROM    EMP,ASG
WHERE   EMP.ENO = ASG.ENO
AND     DUR > 37
```

Strategy 1

$$\Pi_{ENAME}(\sigma_{DUR>37 \wedge EMP.ENO=ASG.ENO}(EMP \times ASG))$$

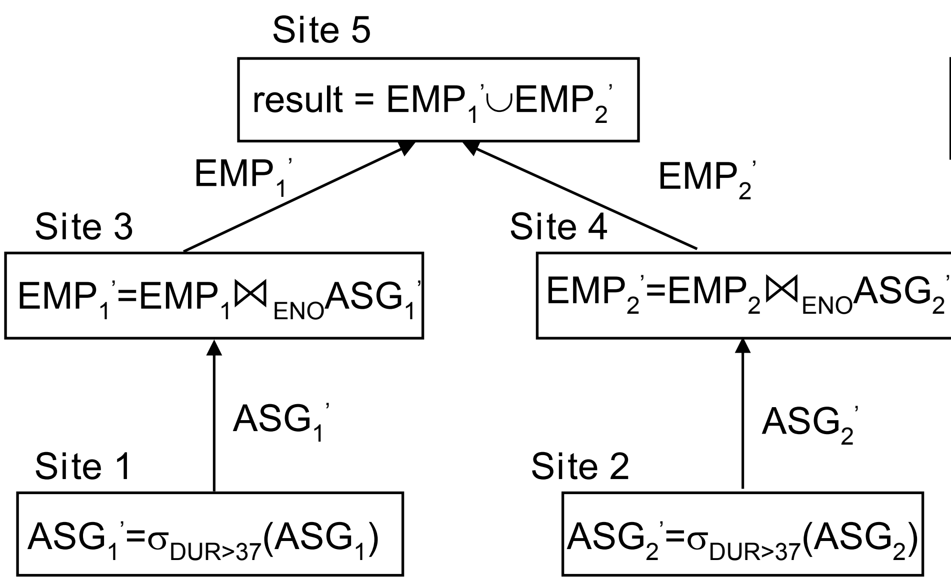
Strategy 2

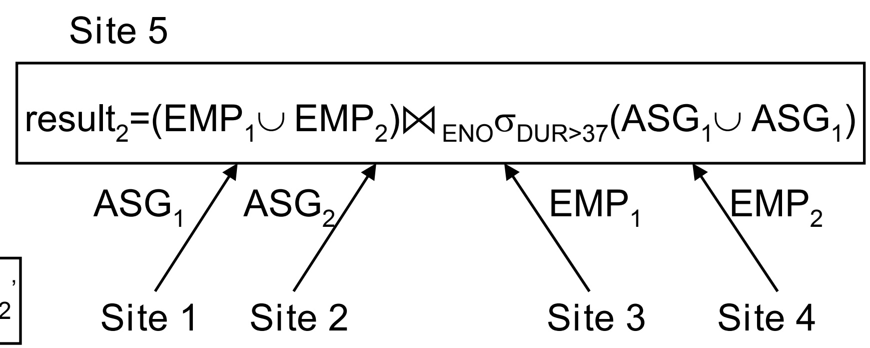
$$\Pi_{ENAME}(EMP \bowtie_{ENO} (\sigma_{DUR>37}(ASG)))$$

Strategy 2 avoids Cartesian product, so is “better”

What is the Problem?

Site 1	Site 2	Site 3	Site 4	Site 5
$ASG_1=\sigma_{ENO\leq\text{“E3”}}(ASG)$	$ASG_2=\sigma_{ENO>\text{“E3”}}(ASG)$	$EMP_1=\sigma_{ENO\leq\text{“E3”}}(EMP)$	$EMP_2=\sigma_{ENO>\text{“E3”}}(EMP)$	Result





Cost of Alternatives

- Assume:
  - *size*(EMP) = 400, *size*(ASG) = 1000
  - tuple access cost = 1 unit; tuple transfer cost = 10 units
- Strategy 1

❶	produce ASG': (10+10)∗tuple access cost	20
❷	transfer ASG' to the sites of EMP: (10+10)∗tuple transfer cost	200
❸	produce EMP': (10+10) ∗tuple access cost∗2	40
❹	transfer EMP' to result site: (10+10) ∗tuple transfer cost	200
	Total cost	460

- Strategy 2

❶	transfer EMP to site 5:400∗tuple transfer cost	4,000
❷	transfer ASG to site 5 :1000∗tuple transfer cost	10,000
❸	produce ASG':1000∗tuple access cost	1,000
❹	join EMP and ASG':400∗20∗tuple access cost	8,000
	Total cost	23,000

Query Optimization Objectives

Minimize a cost function

I/O cost + CPU cost + communication cost

These might have different weights in different distributed environments

Wide area networks

- communication cost will dominate
  - low bandwidth
  - low speed
  - high protocol overhead
- most algorithms ignore all other cost components

Local area networks

- communication cost not that dominant
- total cost function should be considered

Can also maximize throughput

Query Optimization Issues – Types of Optimizers

- Exhaustive search
  - cost-based
  - optimal
  - combinatorial complexity in the number of relations
- Heuristics
  - not optimal
  - regroup common sub-expressions
  - perform selection, projection first
  - replace a join by a series of semijoins
  - reorder operations to reduce intermediate relation size
  - optimize individual operations

Query Optimization Issues – 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 Issues – Optimization Timing

- Static
  - compilation ⇒ optimize prior to the execution
  - difficult to estimate the size of the intermediate results ⇒ error propagation
  - can amortize over many executions
  - R*
- Dynamic
  - run time optimization
  - exact information on the intermediate relation sizes
  - have to reoptimize for multiple executions
  - Distributed INGRES
- Hybrid
  - compile using a static algorithm
  - if the error in estimate sizes > threshold, reoptimize at run time
  - MERMAID

Query Optimization Issues – Statistics

- Relation
  - cardinality
  - size of a tuple
  - fraction of tuples participating in a join with another relation
- Attribute
  - cardinality of domain
  - actual number of distinct values
- Common assumptions
  - independence between different attribute values
  - uniform distribution of attribute values within their domain

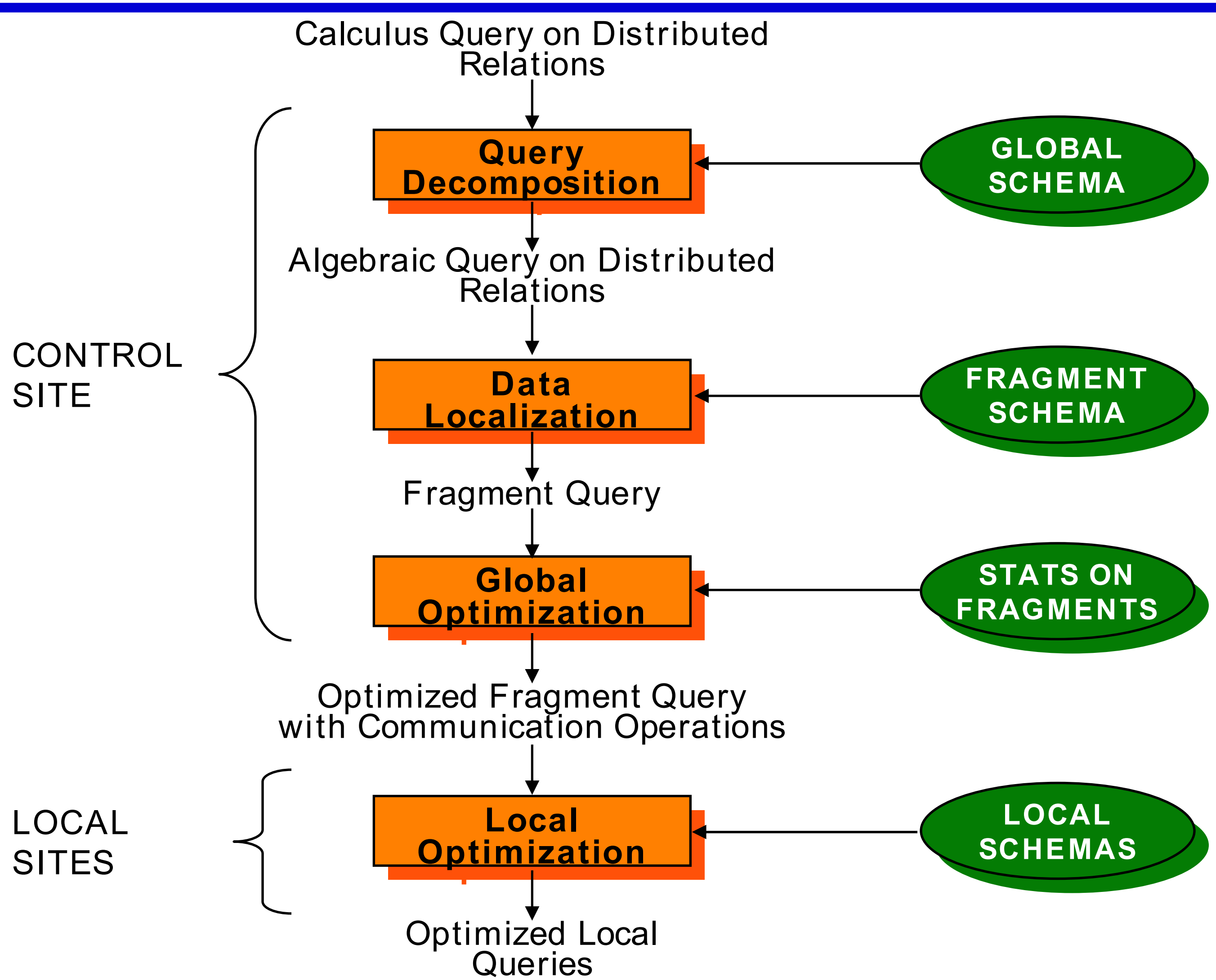
Query Optimization Issues – Decision Sites

- Centralized
  - single site determines the "best" schedule
  - simple
  - need knowledge about the entire distributed database
- Distributed
  - cooperation among sites to determine the schedule
  - need only local information
  - cost of cooperation
- Hybrid
  - one site determines the global schedule
  - each site optimizes the local subqueries

Query Optimization Issues – Network Topology

- Wide area networks (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
- Local area networks (LAN)
  - communication cost not that dominant
  - total cost function should be considered
  - broadcasting can be exploited (joins)
  - special algorithms exist for star networks

Distributed Query Processing Methodology



Step 1 – Query Decomposition

Input : Calculus query on global relations

- Normalization
  - manipulate query quantifiers and qualification
- Analysis
  - detect and reject “incorrect” queries
  - possible for only a subset of relational calculus
- Simplification
  - eliminate redundant predicates
- Restructuring
  - calculus query $\Rightarrow$ algebraic query
  - more than one translation is possible
  - use transformation rules

Normalization

- Lexical and syntactic analysis
  - check validity (similar to compilers)
  - check for attributes and relations
  - type checking on the qualification
- Put into normal form
  - Conjunctive normal form

    $$(p_{11} \lor p_{12} \lor \ldots \lor p_{1n}) \land \ldots \land (p_{m1} \lor p_{m2} \lor \ldots \lor p_{mn})$$

  - Disjunctive normal form

    $$(p_{11} \land p_{12} \land \ldots \land p_{1n}) \lor \ldots \lor (p_{m1} \land p_{m2} \land \ldots \land p_{mn})$$

  - OR's mapped into union
  - AND's mapped into join or selection

Analysis

- Refute incorrect queries
- Type incorrect
  - If any of its attribute or relation names are not defined in the global schema
  - If operations are applied to attributes of the wrong type
- Semantically incorrect
  - Components do not contribute in any way to the generation of the result
  - Only a subset of relational calculus queries can be tested for correctness
  - Those that do not contain disjunction and negation
  - To detect
    - connection graph (query graph)
    - join graph

Simplification

- Why simplify?
  - Remember the example
- How? Use transformation rules
  - elimination of redundancy
    - idempotency rules

      $p_1 \land \neg(p_1) \Leftrightarrow \text{false}$

      $p_1 \land (p_1 \lor p_2) \Leftrightarrow p_1$

      $p_1 \lor \text{false} \Leftrightarrow p_1$

      …
  - application of transitivity
  - use of integrity rules

Simplification – Example

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

$\Downarrow$

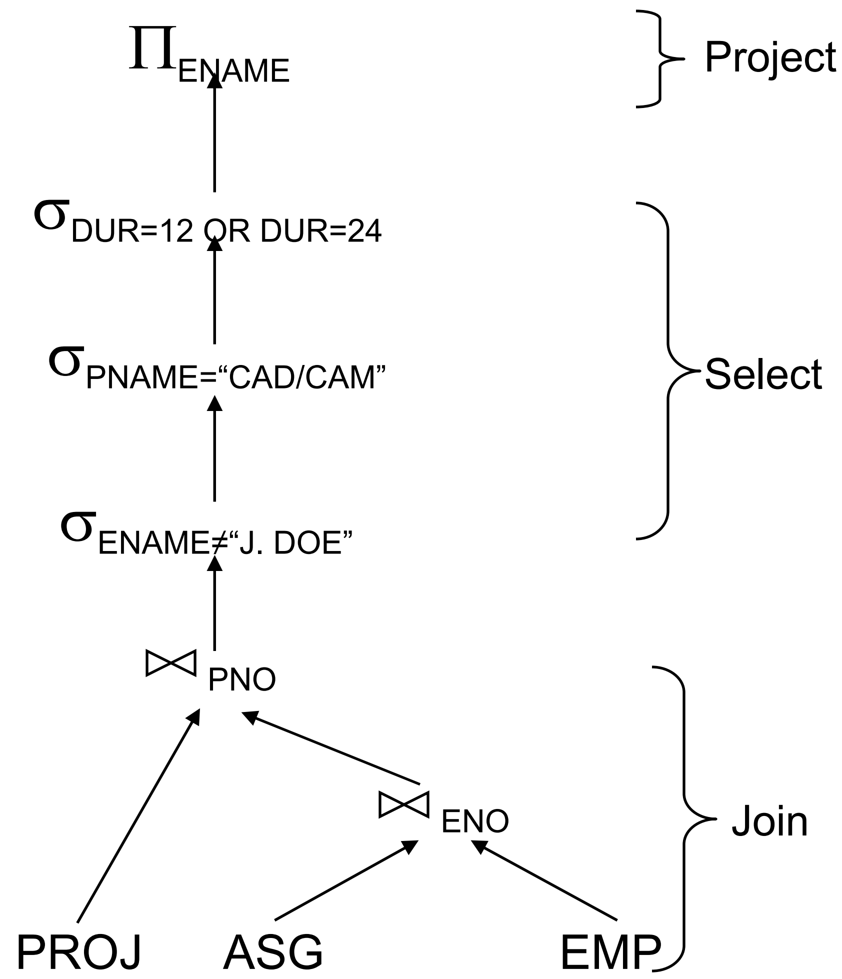
```
SELECT    TITLE
FROM      EMP
WHERE     EMP.ENAME = “J. Doe”
```

Restructuring

- Convert relational calculus to relational algebra
- Make use of query trees
- Example

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



Restructuring –Transformation Rules

- 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
  - $\Pi_{A'}(\Pi_{A'}(R)) \Leftrightarrow \Pi_{A'}(R)$
  - $\sigma_{p_1(A_1)}(\sigma_{p_2(A_2)}(R)) = \sigma_{p_1(A_1) \wedge p_2(A_2)}(R)$

  where $R[A]$ and $A' \subseteq A$, $A'' \subseteq A$ and $A' \subseteq A''$
- Commuting selection with projection

Restructuring – Transformation Rules

- Commuting selection with binary operations
  - $\sigma_{p(A)}(R \times S) \Leftrightarrow (\sigma_{p(A)}(R)) \times S$
  - $\sigma_{p(A_i)}(R \bowtie_{(A_j,B_k)} S) \Leftrightarrow (\sigma_{p(A_i)}(R)) \bowtie_{(A_j,B_k)} S$
  - $\sigma_{p(A_i)}(R \cup T) \Leftrightarrow \sigma_{p(A_i)}(R) \cup \sigma_{p(A_i)}(T)$

  where A_i belongs to R and T

- Commuting projection with binary operations
  - $\Pi_C(R \times S) \Leftrightarrow \Pi_{A'}(R) \times \Pi_{B'}(S)$
  - $\Pi_C(R \bowtie_{(A_j,B_k)} S) \Leftrightarrow \Pi_{A'}(R) \bowtie_{(A_j,B_k)} \Pi_{B'}(S)$
  - $\Pi_C(R \cup S) \Leftrightarrow \Pi_C(R) \cup \Pi_C(S)$

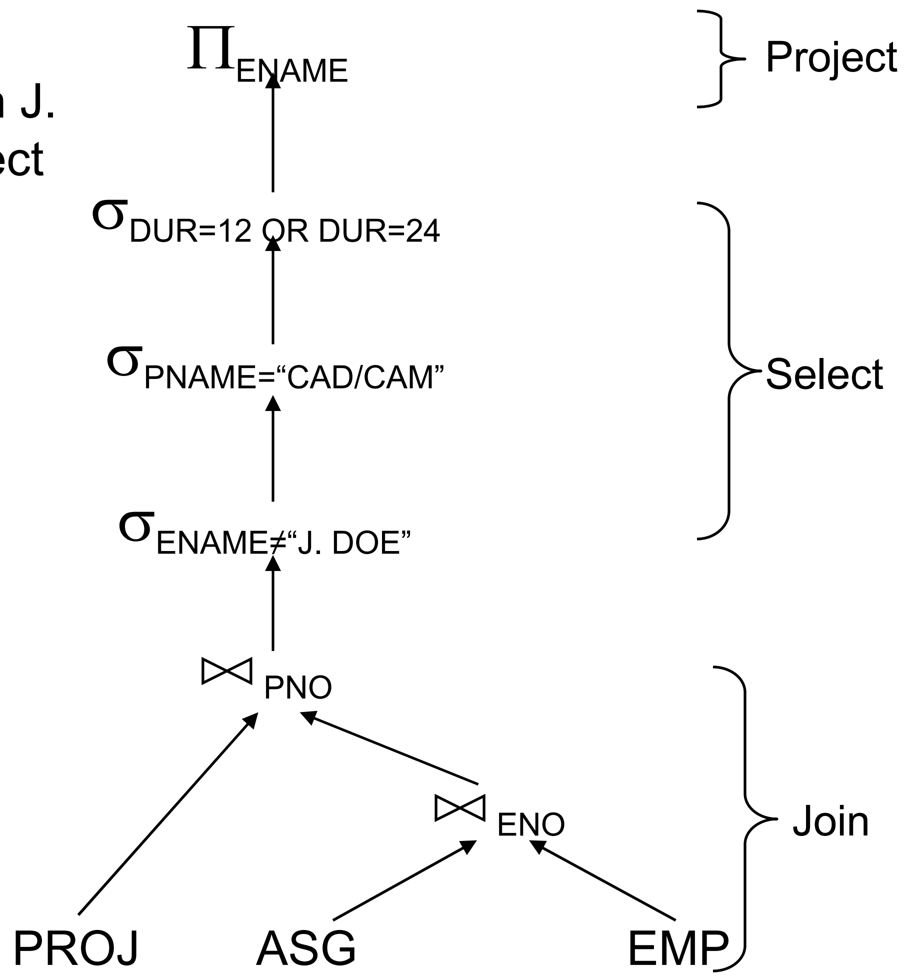
  where $R[A]$ and $S[B]$; $C = A' \cup B'$ where $A' \subseteq A$, $B' \subseteq B$

Example

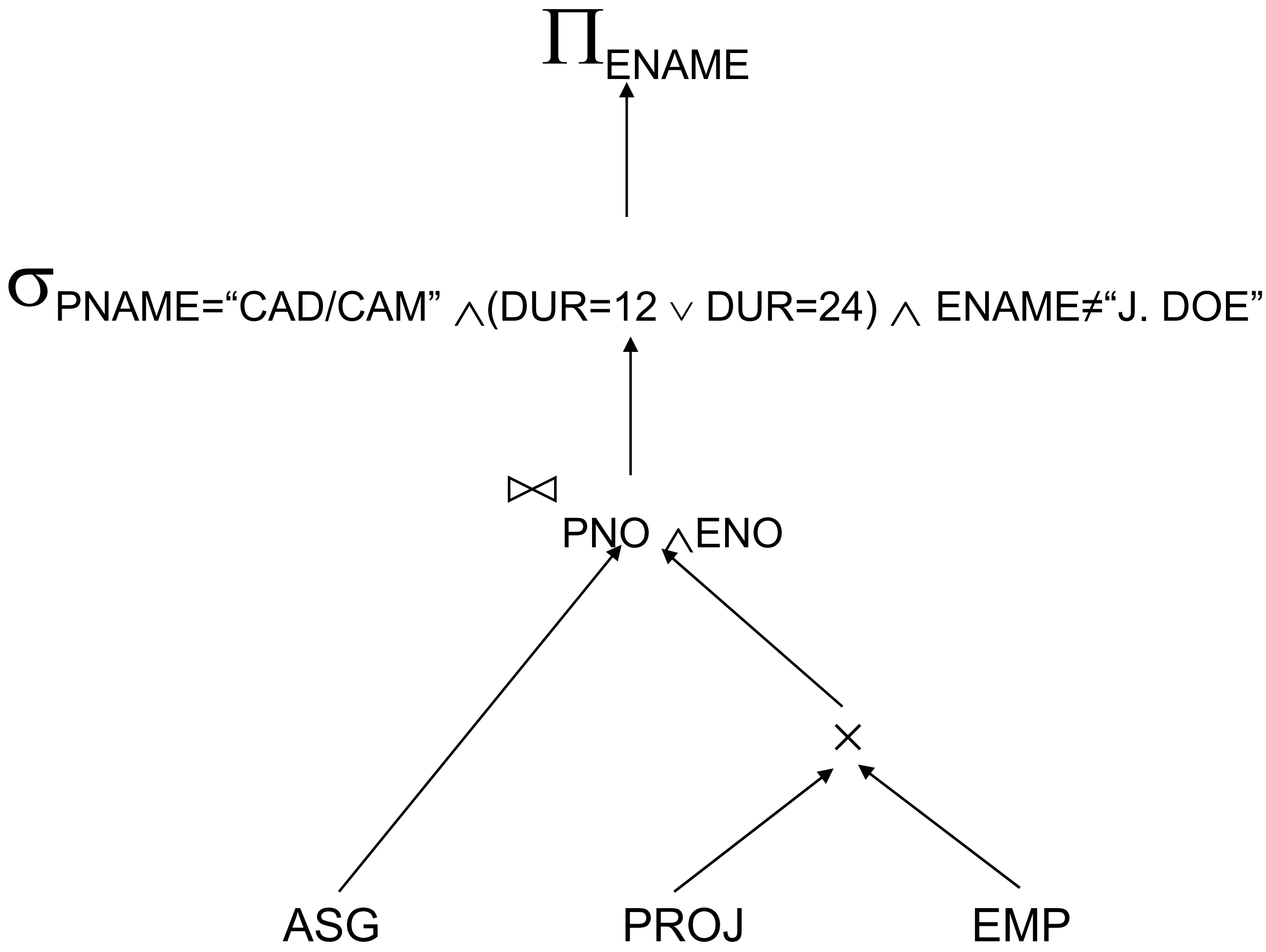
Recall the previous example:

Find the names of employees other than J. Doe who worked on the CAD/CAM project for either one or two years.

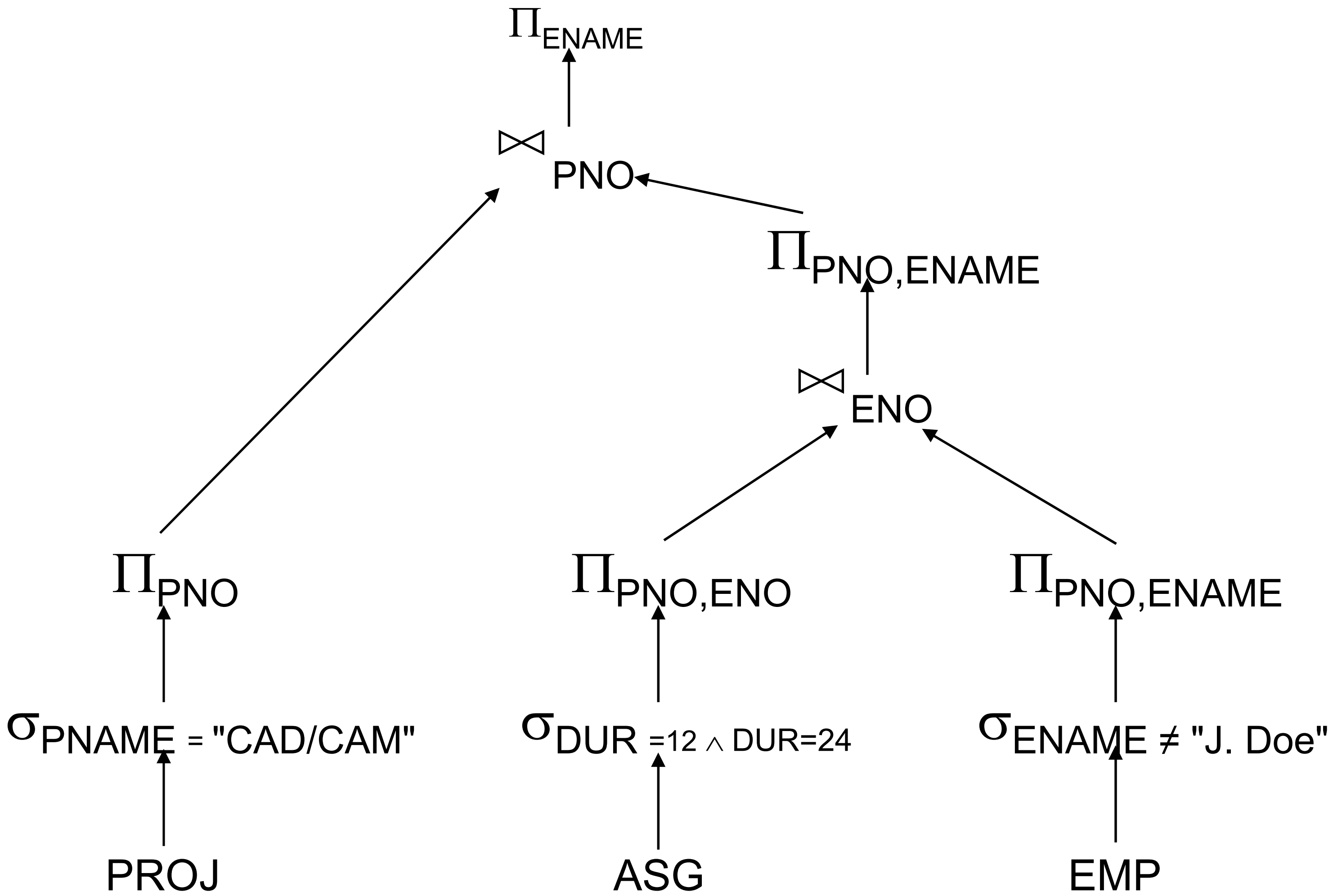
```
SELECT    ENAME
FROM      PROJ, ASG, EMP
WHERE     ASG.ENO=EMP.ENO
AND ASG.PNO=PROJ.PNO
AND ENAME≠"J. Doe"
AND PROJ.PNAME="CAD/CAM"
AND (DUR=12 OR DUR=24)
```



Equivalent Query



Restructuring



Step 2 – Data Localization

Input: Algebraic query on distributed relations

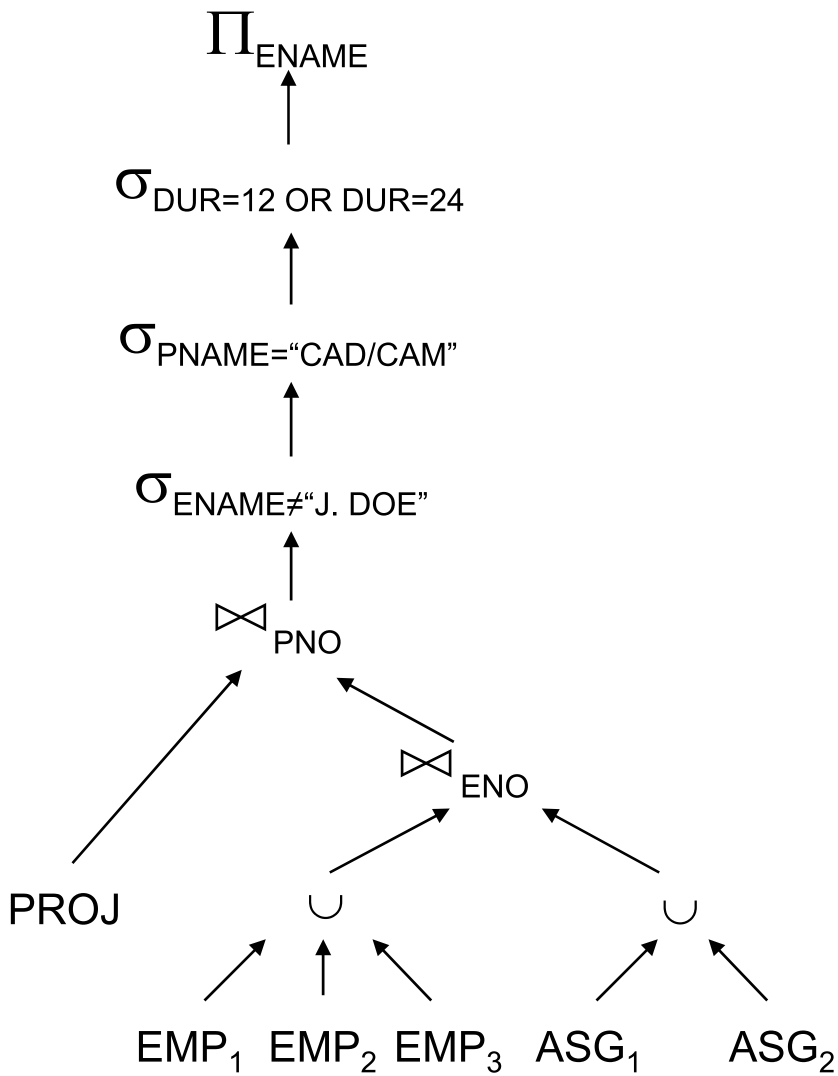
- Determine which fragments are involved
- Localization program
  - substitute for each global query its materialization program
  - optimize

Example

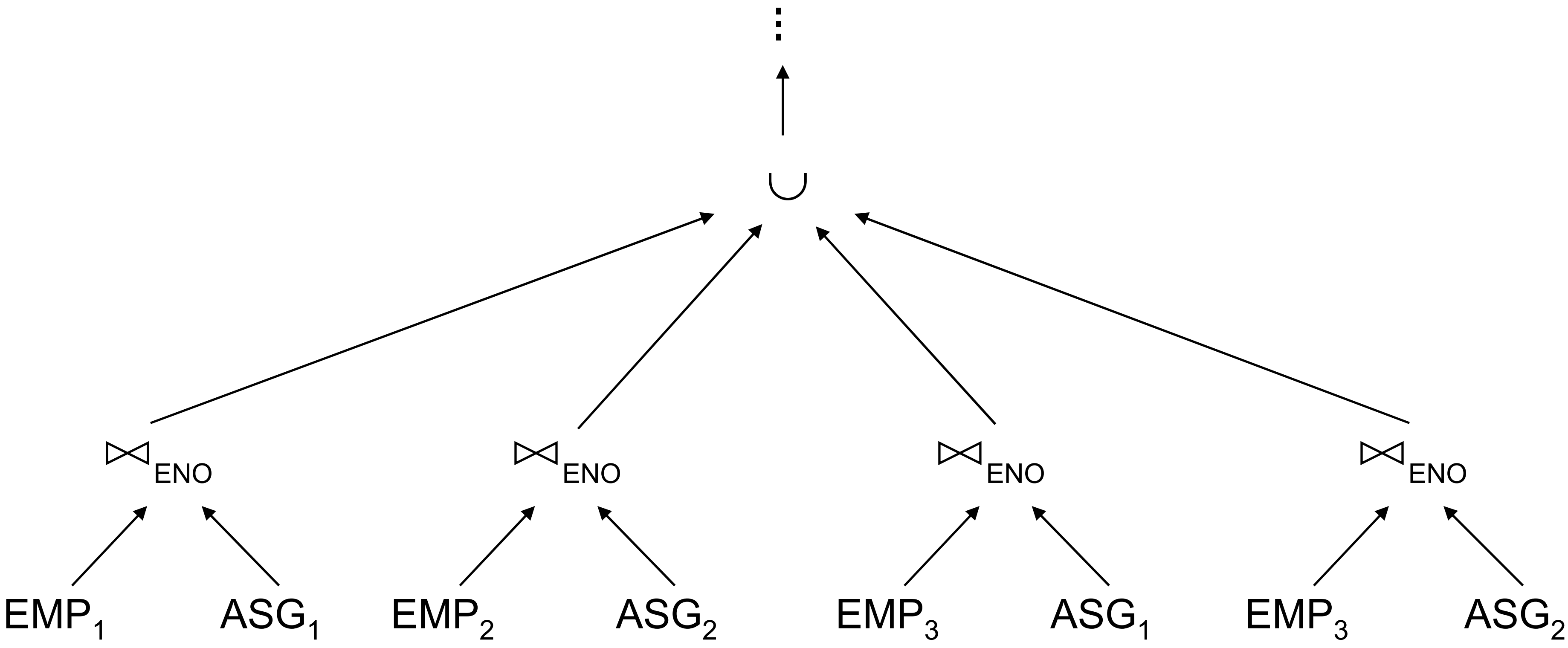
Assume

- EMP is fragmented into EMP_1, EMP_2, EMP_3 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 fragmented into ASG_1 and ASG_2 as follows:
  - $ASG_1 = \sigma_{ENO \leq \text{"E3"}}(ASG)$
  - $ASG_2 = \sigma_{ENO > \text{"E3"}}(ASG)$

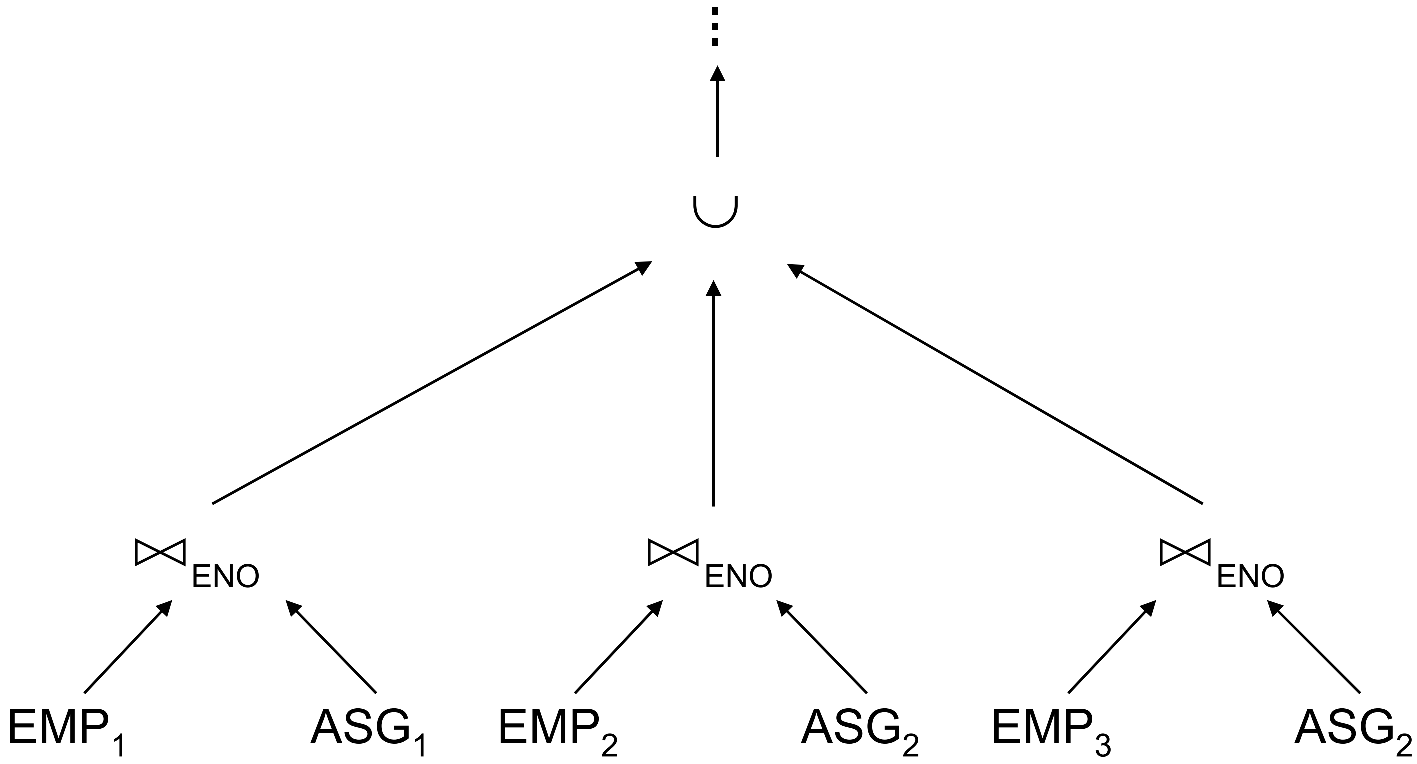
Replace EMP by $(EMP_1 \cup EMP_2 \cup EMP_3)$ and ASG by $(ASG_1 \cup ASG_2)$ in any query



Provides Parallellism



Eliminates Unnecessary Work

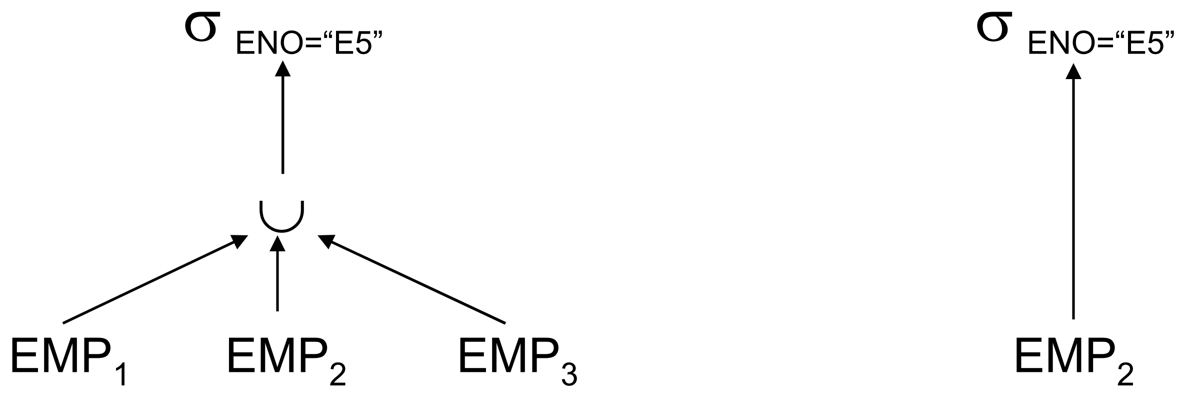


Reduction for PHF

- Reduction with selection
  - 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

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



Reduction for PHF

- Reduction with join
  - Possible if fragmentation is done on join attribute
  - Distribute join over union

    $(R_1 \cup R_2) \bowtie S \Leftrightarrow (R_1 \bowtie S) \cup (R_2 \bowtie S)$

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

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

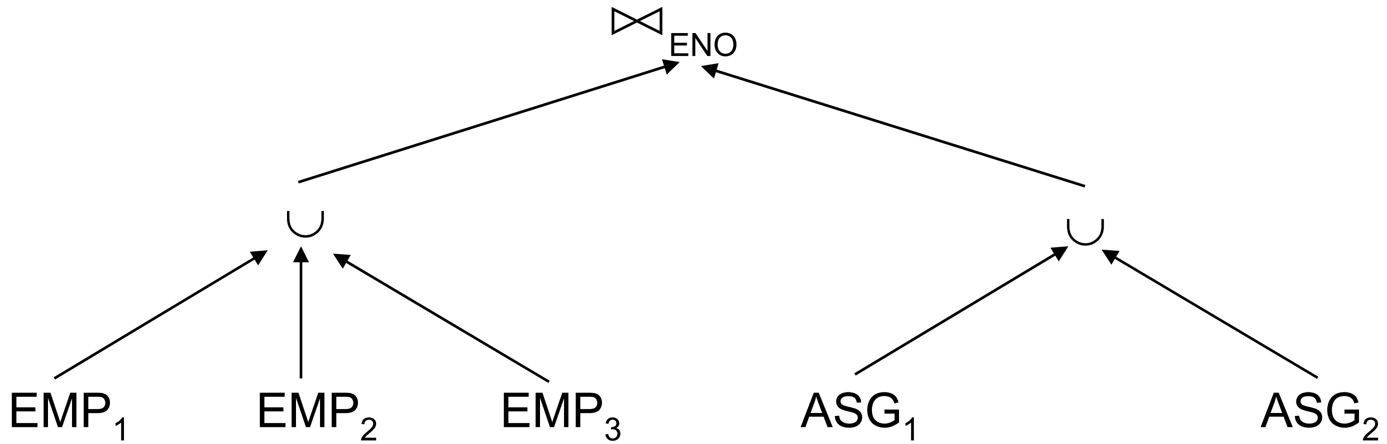
Reduction for PHF

- Reduction with join - Example
  - Assume EMP is fragmented as before and

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

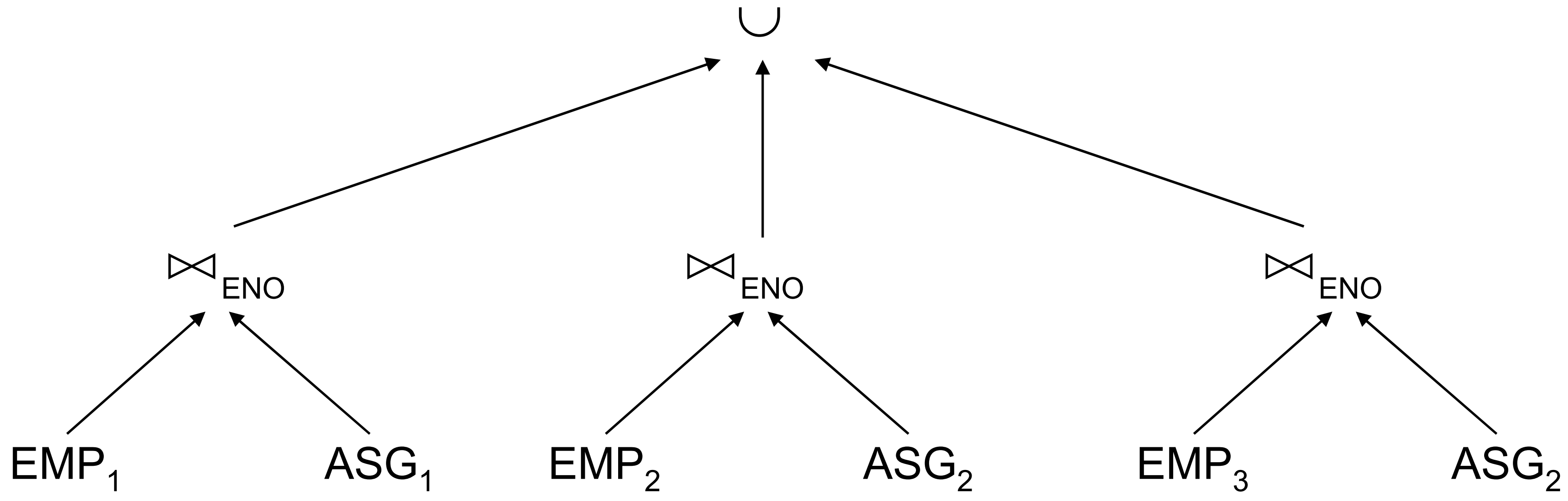
    ASG_2: $\sigma_{ENO > \text{"E3"}}(ASG)$
  - Consider the query

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



Reduction for PHF

- Reduction with join - Example
  - Distribute join over unions
  - Apply the reduction rule



Reduction for VF

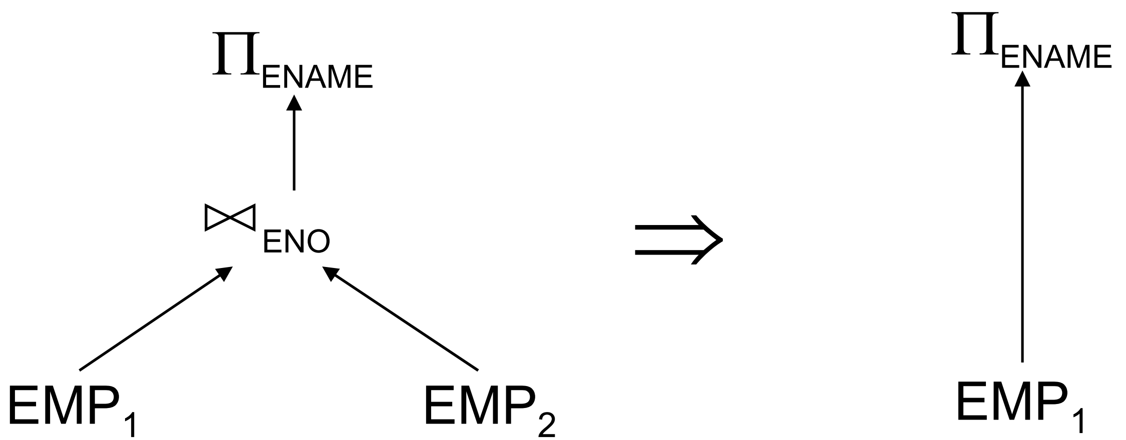
- Find useless (not empty) intermediate relations

  Relation R defined over attributes $A = \{A_1, ..., A_n\}$ 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: $EMP_1 = \Pi_{ENO,ENAME}(EMP)$; $EMP_2 = \Pi_{ENO,TITLE}(EMP)$

```
SELECT   ENAME
FROM     EMP
```



Step 3 – Global Query Optimization

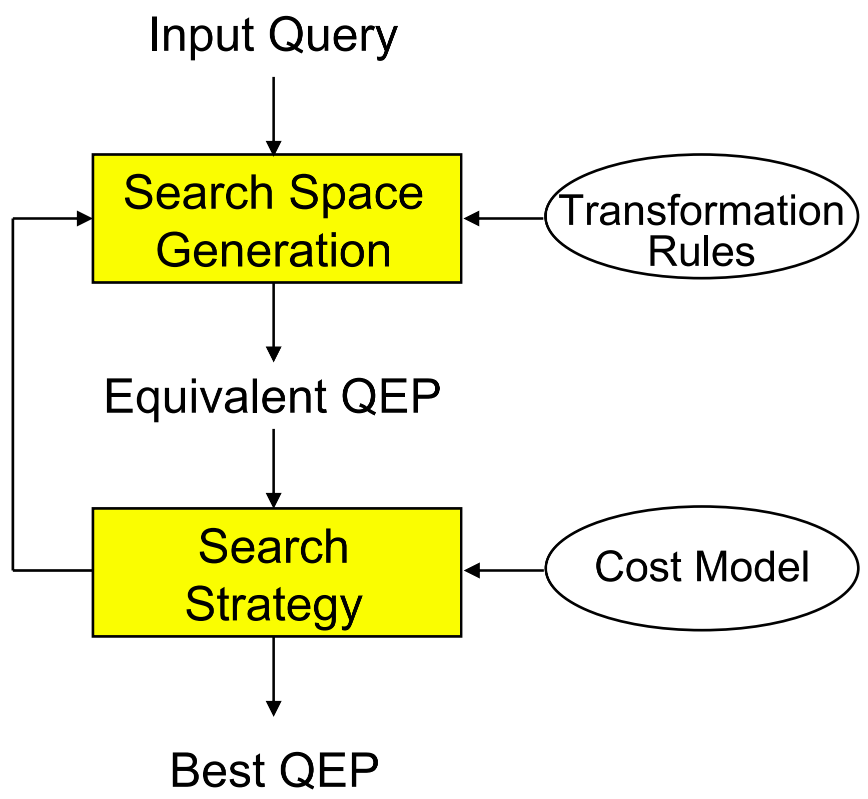
Input: Fragment query

- Find the *best* (not necessarily optimal) global schedule
  - Minimize a cost function
  - Distributed join processing
    - Bushy vs. linear trees
    - Which relation to ship where?
    - Ship-whole vs ship-as-needed
  - Decide on the use of semijoins
    - Semijoin saves on communication at the expense of more local processing.
  - Join methods
    - nested loop vs ordered joins (merge join or hash join)

Cost-Based Optimization

- Solution space
  - The set of equivalent algebra expressions (query trees).
- Cost function (in terms of time)
  - I/O cost + CPU cost + communication cost
  - These might have different weights in different distributed environments (LAN vs WAN).
  - Can also maximize throughput
- Search algorithm
  - How do we move inside the solution space?
  - Exhaustive search, heuristic algorithms (iterative improvement, simulated annealing, genetic,…)

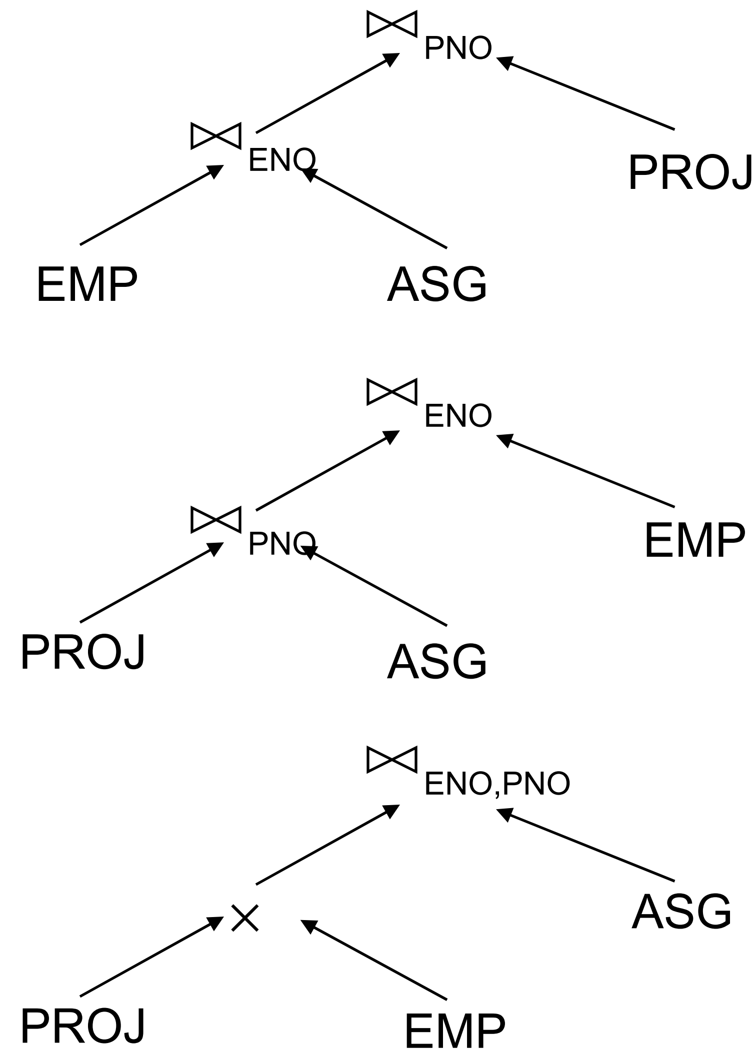
Query Optimization Process



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

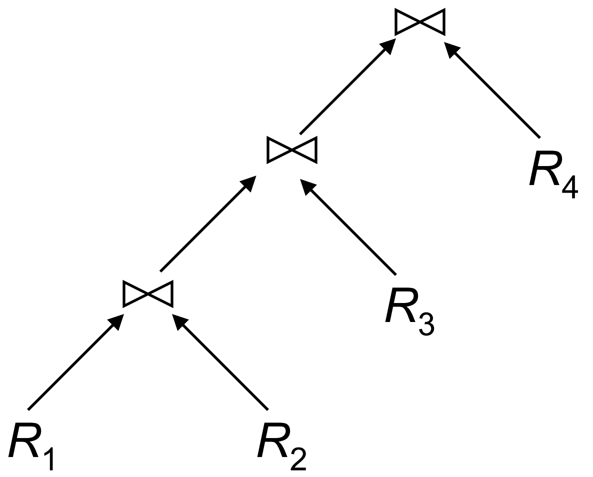
```
SELECT    ENAME,RESP
FROM      EMP, ASG, PROJ
WHERE     EMP.ENO=ASG.ENO
AND       ASG.PNO=PROJ.PNO
```



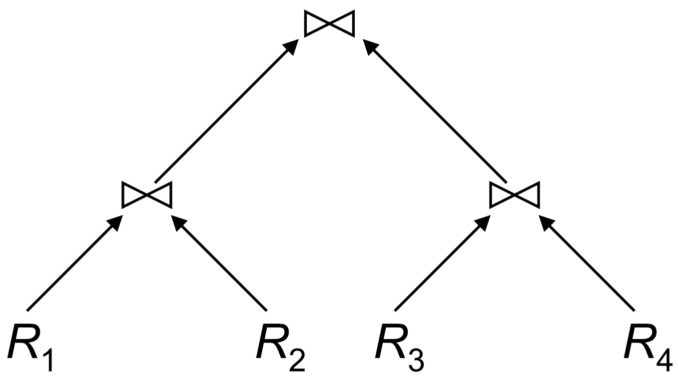
Search Space

- Restrict by means of heuristics
  - Perform unary operations before binary operations
  - …
- Restrict the shape of the join tree
  - Consider only linear trees, ignore bushy ones

Linear Join Tree



Bushy Join Tree

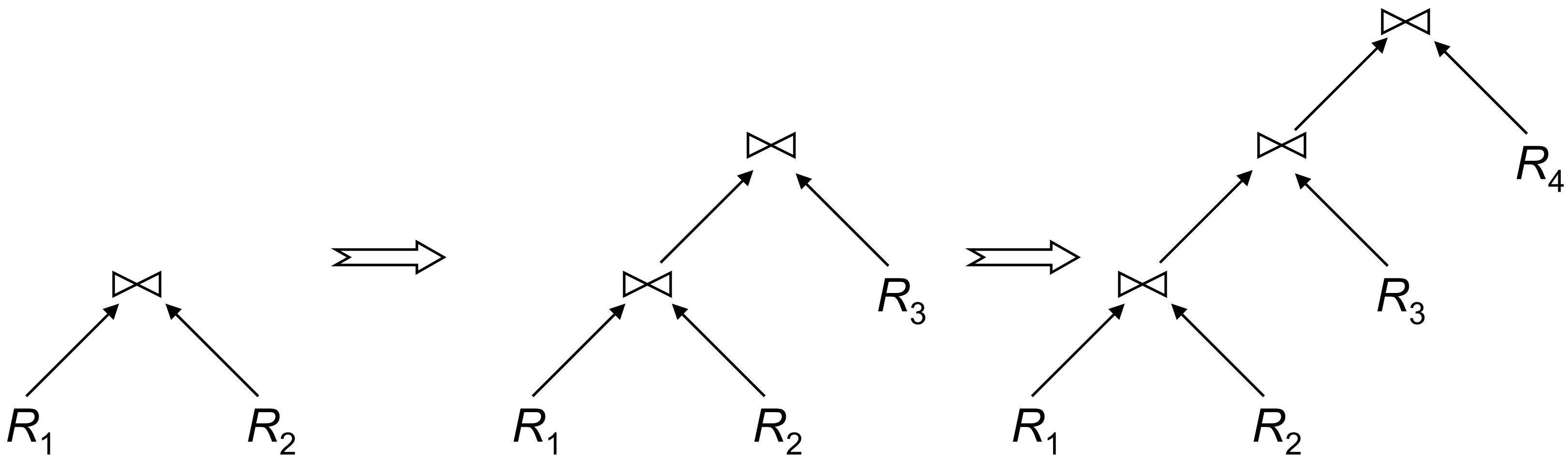


Search Strategy

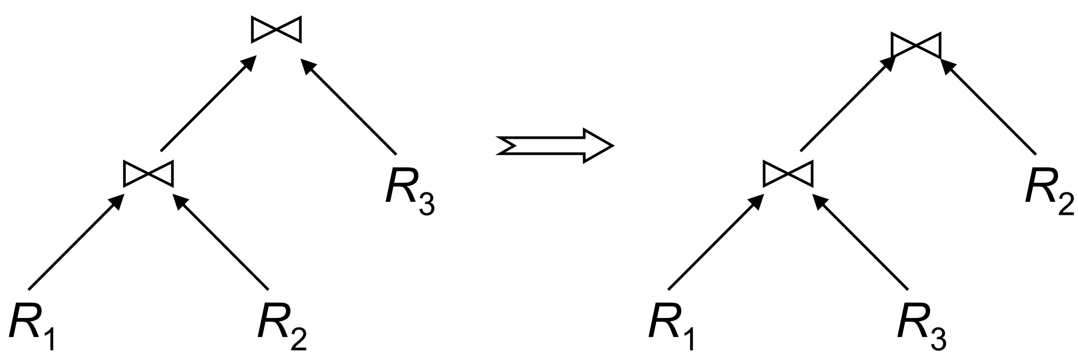
- How to “move” in the search space.
- Deterministic
  - Start from base relations and build plans by adding one relation at each step
  - Dynamic programming: breadth-first
  - Greedy: depth-first
- Randomized
  - Search for optimalities around a particular starting point
  - Trade optimization time for execution time
  - Better when > 5-6 relations
  - Simulated annealing
  - Iterative improvement

Search Strategies

- Deterministic



- Randomized



Cost Functions

- Total Time (or Total Cost)
  - Reduce each cost (in terms of time) component individually
  - Do as little of each cost component as possible
  - Optimizes the utilization of the resources

⇩

Increases system throughput

- Response Time
  - Do as many things as possible in parallel
  - May increase total time because of increased total activity

Total Cost

Summation of all cost factors

Total cost = CPU cost + I/O cost + communication cost

CPU cost = unit instruction cost $*$ no.of instructions

I/O cost = unit disk I/O cost $*$ no. of disk I/Os

communication cost = message initiation + transmission

Total Cost Factors

- Wide area network
  - message initiation and transmission costs high
  - local processing cost is low (fast mainframes or minicomputers)
  - ratio of communication to I/O costs = 20:1
- Local area networks
  - communication and local processing costs are more or less equal
  - ratio = 1:1.6

Response Time

Elapsed time between the initiation and the completion of a query

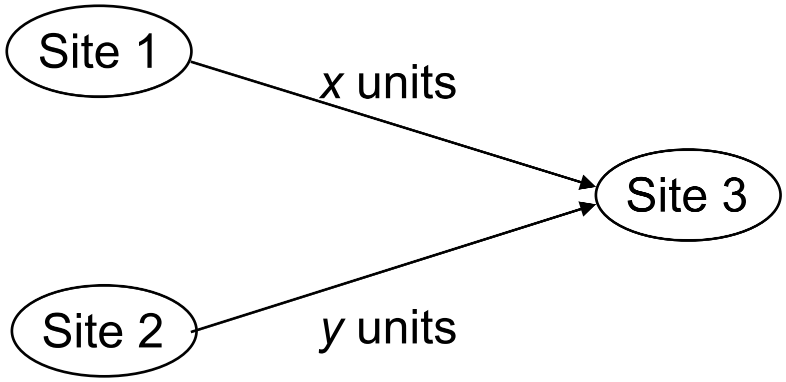
Response time = CPU time + I/O time + communication time

CPU time = unit instruction time ∗ no. of sequential instructions

I/O time = unit I/O time ∗ no. of sequential I/Os

communication time = unit msg initiation time ∗ no. of sequential msg + unit transmission time ∗ no. of sequential bytes

Example



Assume that only the communication cost is considered

Total time = 2 $*$ message initialization time + unit transmission time $*$ $(x+y)$

Response time = max {time to send x from 1 to 3, time to send y from 2 to 3}

time to send x from 1 to 3 = message initialization time + unit transmission time $*$ x

time to send y from 2 to 3 = message initialization time + unit transmission time $*$ y

Optimization Statistics

- Primary cost factor: size of intermediate relations
- Make them precise ⇒ more costly to maintain
  - For each relation $R[A_1, A_2, \ldots, A_n]$ fragmented as $R_1, \ldots, R_r$
    - length of each attribute: $length(Ai)$
    - the number of distinct values for each attribute in each fragment: $card(\prod_{A_i} R_j)$
    - 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 cardinalities of each fragment: $card(R_j)$
  - Selectivity factor of each operation for relations
    - For joins

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

Intermediate Relation Sizes

Selection

$$size(R) = card(R) * length(R)$$

$$card(\sigma_F (R)) = SF_\sigma (F) * card(R)$$

where

$$S\,F_\sigma(A = value) = \frac{1}{card(\prod_A(R))}$$

$$S\,F_\sigma(A > value) = \frac{max(A) - value}{max(A) - min(A)}$$

$$S\,F_\sigma(A < value) = \frac{value - max(A)}{max(A) - min(A)}$$

$$SF_\sigma(p(A_i) \wedge p(A_j)) = SF_\sigma(p(A_i)) * SF_\sigma(p(A_j))$$

$$SF_\sigma(p(A_i) \vee p(A_j)) = SF_\sigma(p(A_i)) + SF_\sigma(p(A_j)) - (SF_\sigma(p(A_i)) * SF_\sigma(p(A_j)))$$

$$SF_\sigma(A \in value) = SF_\sigma(A= value) * card(\{values\})$$

Intermediate Relation Sizes

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*;

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

- More general:

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

Semijoin

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

where

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

System R Algorithm

❶ Simple (i.e., mono-relation) queries are executed according to the best access path

❷ Execute joins

2.1 Determine the possible ordering of joins

2.2 Determine the cost of each ordering

2.3 Choose the join ordering with minimal cost

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

  merge relations

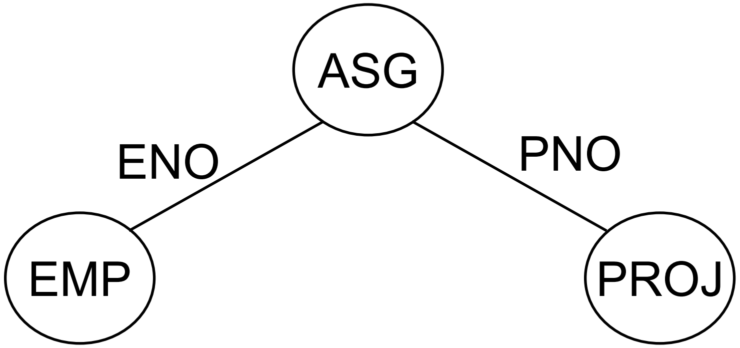
  - Complexity: $n_1 + n_2$ if relations are previously sorted and equijoin

System R Algorithm – 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



System R Example (cont'd)

❶ Choose the best access paths to each relation

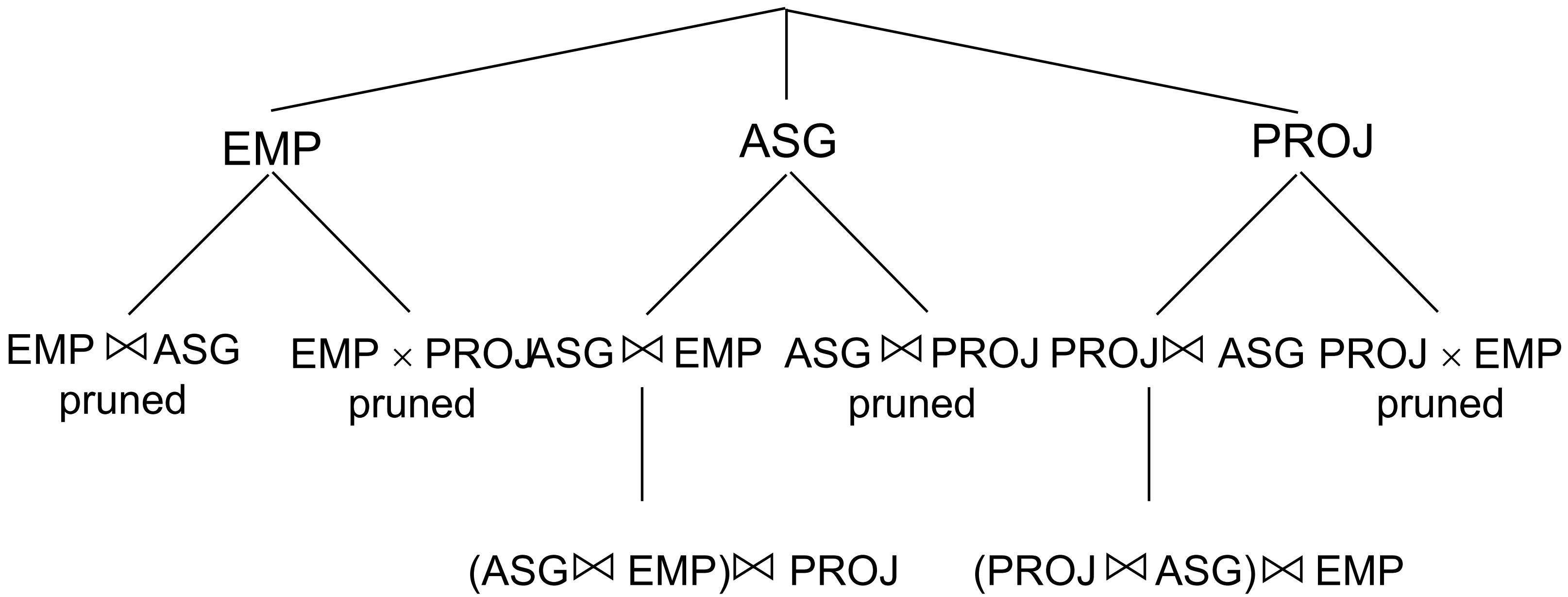
- EMP: sequential scan (no selection on EMP)
- ASG: sequential scan (no selection on ASG)
- PROJ: index on PNAME (there is a selection on PROJ based on PNAME)

❷ Determine the best join ordering

- EMP ⋈ ASG ⋈ PROJ
- ASG ⋈ PROJ ⋈ EMP
- PROJ ⋈ ASG ⋈ EMP
- ASG ⋈ EMP ⋈ PROJ
- EMP × PROJ ⋈ ASG
- PROJ × EMP ⋈ ASG
- Select the best ordering based on the join costs evaluated according to the two methods

System R Algorithm

Alternatives



Best total join order is one of

((ASG $\bowtie$ EMP) $\bowtie$ PROJ)

((PROJ $\bowtie$ ASG) $\bowtie$ EMP)

System R Algorithm

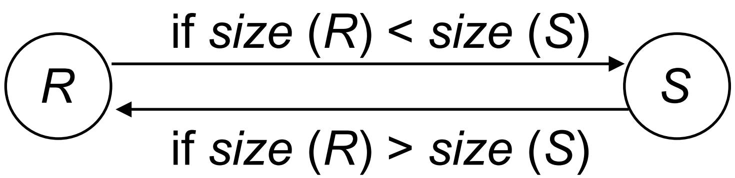
- ((PROJ $\bowtie$ ASG) $\bowtie$ EMP) has a useful index on the select attribute and direct access to the join attributes of ASG and EMP
- Therefore, chose it with the following access methods:
  - select PROJ using index on PNAME
  - then join with ASG using index on PNO
  - then join with EMP using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

Join Ordering in Fragment Queries

- Ordering joins
  - Distributed INGRES
  - System R*
- Semijoin ordering
  - SDD-1

Join Ordering

- Consider two relations only

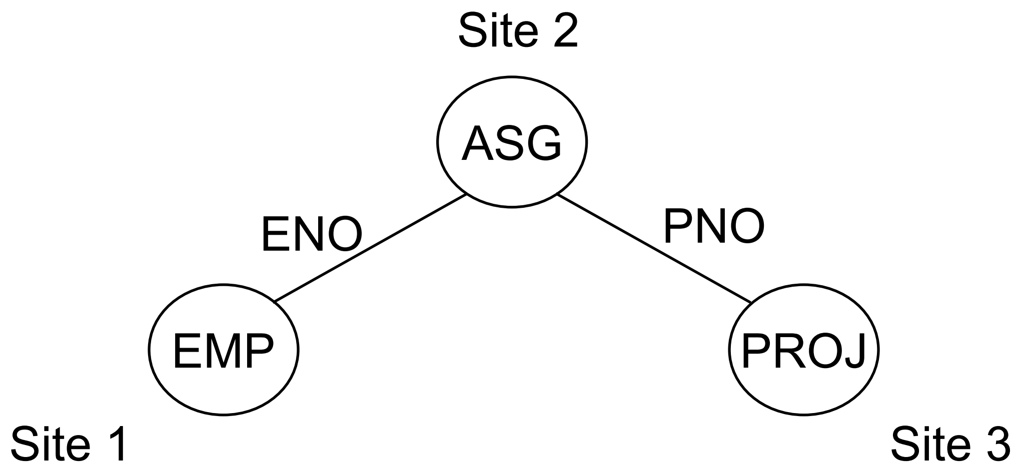


- Multiple relations more difficult because too many alternatives.
  - Compute the cost of all alternatives and select the best one.
    - Necessary to compute the size of intermediate relations which is difficult.
  - Use heuristics

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 ⋈ ASG
   EMP' → Site 3
   Site 3 computes EMP' ⋈ PROJ

2. ASG → Site 1
   Site 1 computes EMP'=EMP ⋈ ASG
   EMP' → Site 3
   Site 3 computes EMP' ⋈ PROJ

3. ASG → Site 3
   Site 3 computes ASG'=ASG ⋈ PROJ
   ASG' → Site 1
   Site 1 computes ASG' ⋈ EMP

4. PROJ → Site 2
   Site 2 computes PROJ'=PROJ ⋈ ASG
   PROJ' → Site 1
   Site 1 computes PROJ' ⋈ EMP

5. EMP → Site 2
   PROJ → Site 2
   Site 2 computes EMP ⋈ PROJ ⋈ ASG

Semijoin Algorithms

- Consider the join of two relations:
  - $R[A]$ (located at site 1)
  - $S[A]$ (located at site 2)
- Alternatives:
  1. Do the join $R \bowtie_A S$
  2. Perform one of the semijoin equivalents

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

Semijoin Algorithms

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

Semijoin is better if

$$size(\Pi_A(S)) + size(R \ltimes_A S)) < size(R)$$

Distributed Query Processing

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

- Cost function includes local processing as well as transmission
- Considers only joins
- Exhaustive search
- Compilation
- Published papers provide solutions to handling horizontal and vertical fragmentations but the implemented prototype does not

R* Algorithm

Performing joins

- 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 – Vertical Partitioning & Joins

1. Move outer relation tuples to the site of the inner relation

   (a) Retrieve outer tuples

   (b) Send them to the inner relation site

   (c) Join them as they arrive

Total Cost = cost(retrieving qualified outer tuples)

+ no. of outer tuples fetched ∗ cost(retrieving qualified inner tuples)

+ msg. cost ∗ (no. outer tuples fetched ∗ avg. outer tuple size) / msg. size

R* Algorithm – Vertical Partitioning & Joins

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)

+ no. 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 ∗ (no. of inner tuples fetched ∗ avg. inner tuple size) / msg. size

R* Algorithm – Vertical Partitioning & Joins

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 ∗ (no. of outer tuples fetched ∗ avg. outer tuple size) / msg. size

+ msg. cost ∗ (no. of inner tuples fetched ∗ avg. inner tuple size) / msg. size

+ no. of outer tuples fetched ∗ cost(retrieving inner tuples from temporary storage)

R* Algorithm – Vertical Partitioning & Joins

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

Total Cost = cost(retrieving qualified outer tuples)

+ msg. cost ∗ (no. of outer tuples fetched)

+ no. of outer tuples fetched ∗ (no. of inner tuples fetched ∗ avg. inner tuple size ∗ msg. cost / msg. size)

+ no. of outer tuples fetched ∗ cost(retrieving matching inner tuples for one outer value)

Step 4 – Local Optimization

Input: Best global execution schedule

- Select the best access path
- Use the centralized optimization techniques