Database Systems II

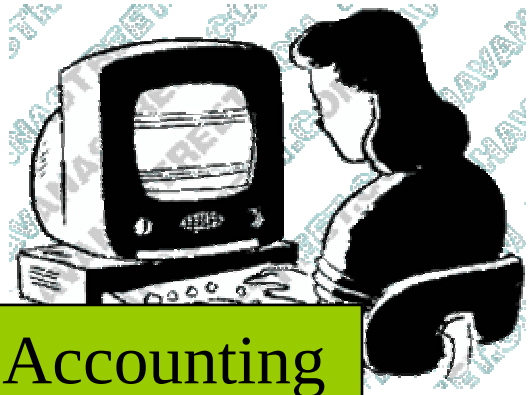
Distributed Databases

Prof. G.N.Wikramanayake

Objective & Content

- Describe distributed database architecture
- Produce designs for distributed database systems
- Recognise different categories of distributed database systems
- Infer query processing in a distributed environment

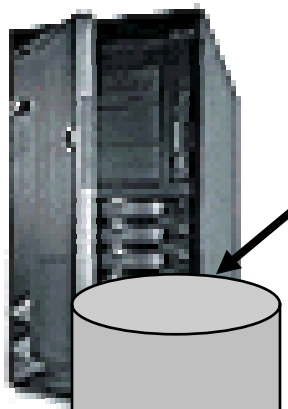
Database Environment



Accounting

Accounts Receivable

Accounts Payable



Marketing

Sales

Advertising

Corporate Database

Accounting

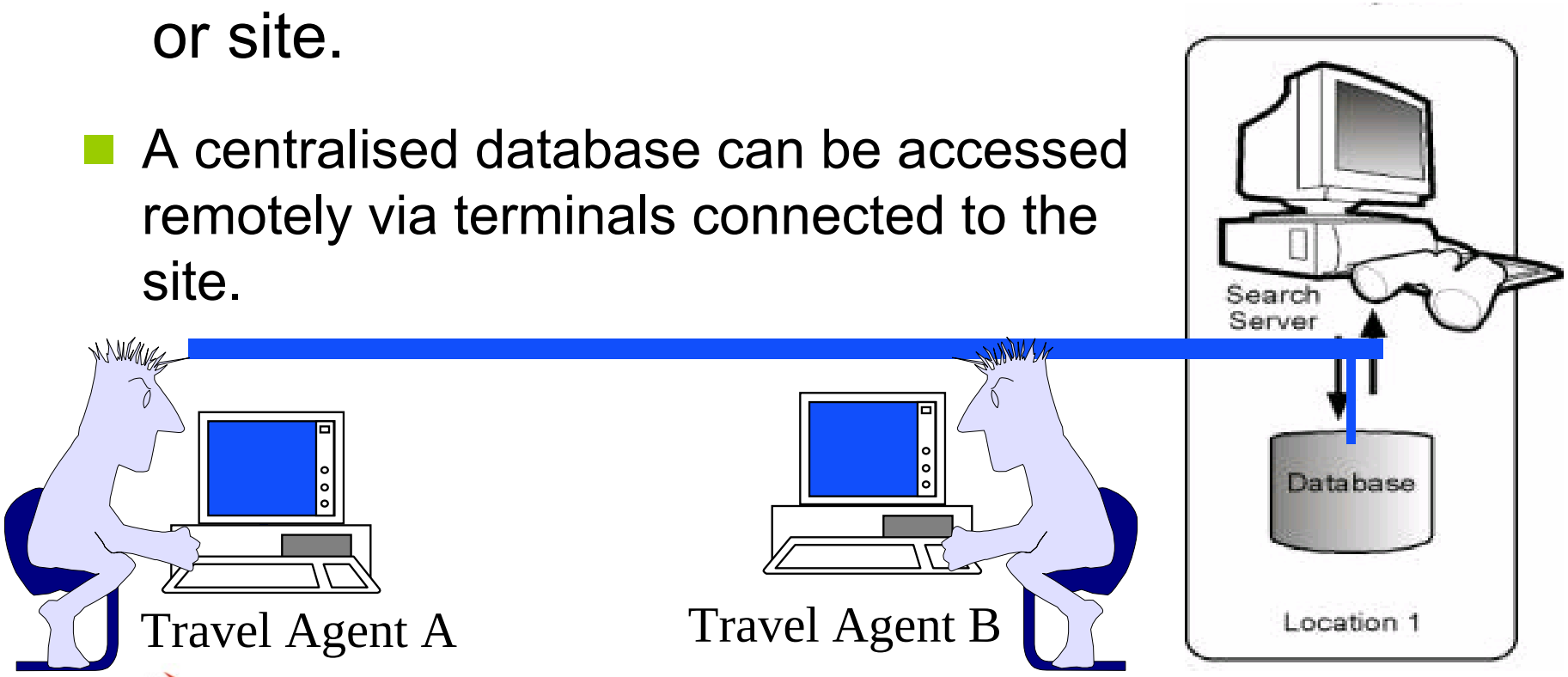
Marketing

Purchasing

Centralised DBMS

- Centralised DBMS all system components (data, DBMS software and secondary storage devices) reside at a single computer or site.

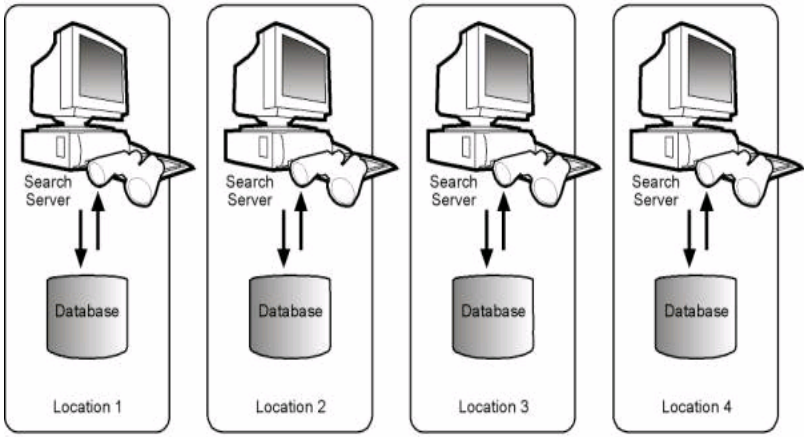
- A centralised database can be accessed remotely via terminals connected to the site.



Distributed Database

- A **distributed database** is a database that is under the control of a central database management system (DBMS) in which storage devices are not all attached to a common CPU.

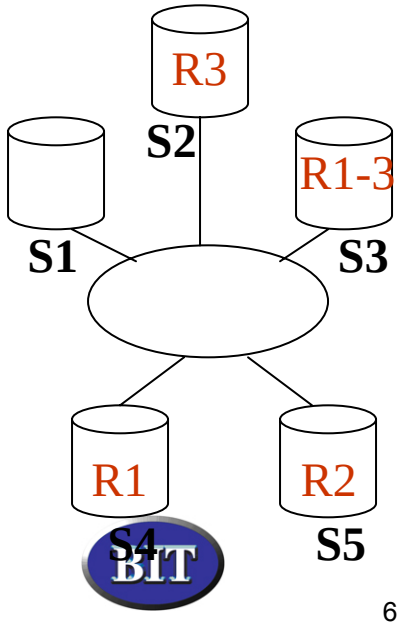
- It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers.



Distributed DBMS

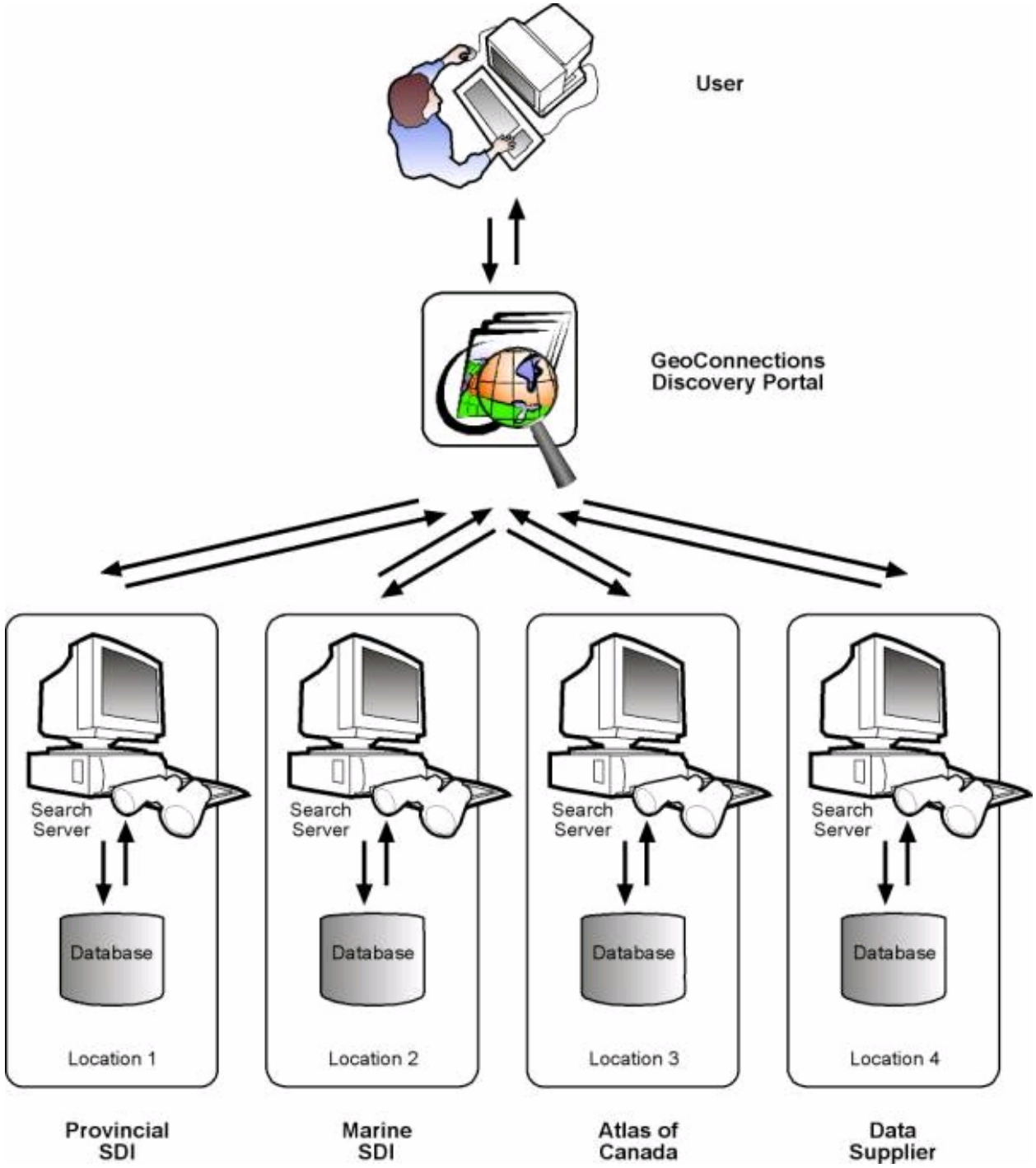
- In Distributed DBMS, each site is a database system site in its own right, but the sites have agreed to work together (if necessary).
- User at any site can access data anywhere in the network exactly as if the data were all stored at user's own site.
- E.g.

  5 Sites, 4 Databases, 1 Replicate



Distributed DBMS

A Centralised view, but distributed

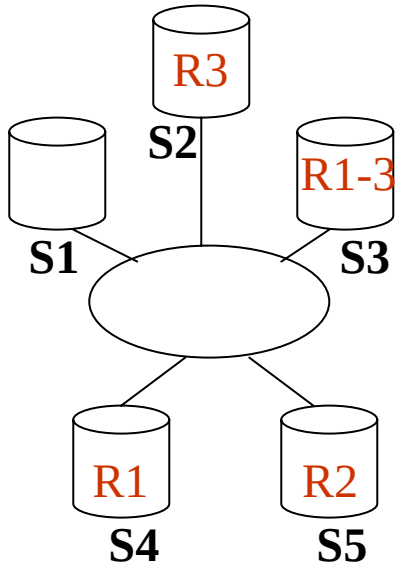


UCSC

Distributed DBMS

- A database server is the software managing a database, and a client is an application that requests information from a server.

- Each computer in a system is a node. A node in a distributed database system acts as a client, a server, or both, depending on the situation.

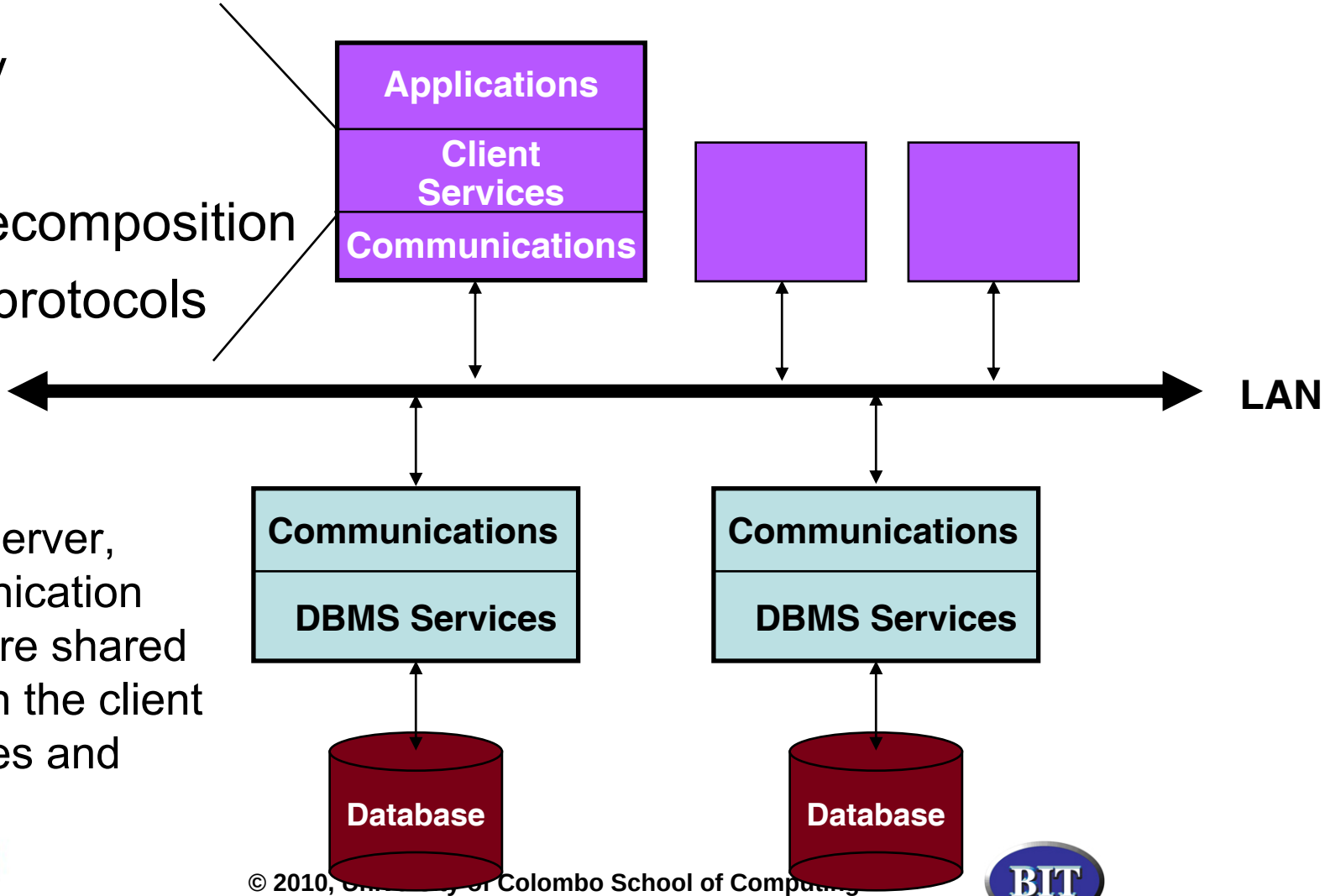


Distribution

- Deals with Physical distribution of data over multiple sites
- Three alternative architectures available
  - Client-Server, communication duties are shared between the client machines and servers.
  - Peer-to-peer systems, no distinction of client machines versus servers. Integrate all local schemas.
  - Non-distributed systems. Integrate some local schema.

Clients/Servers (Multiple)

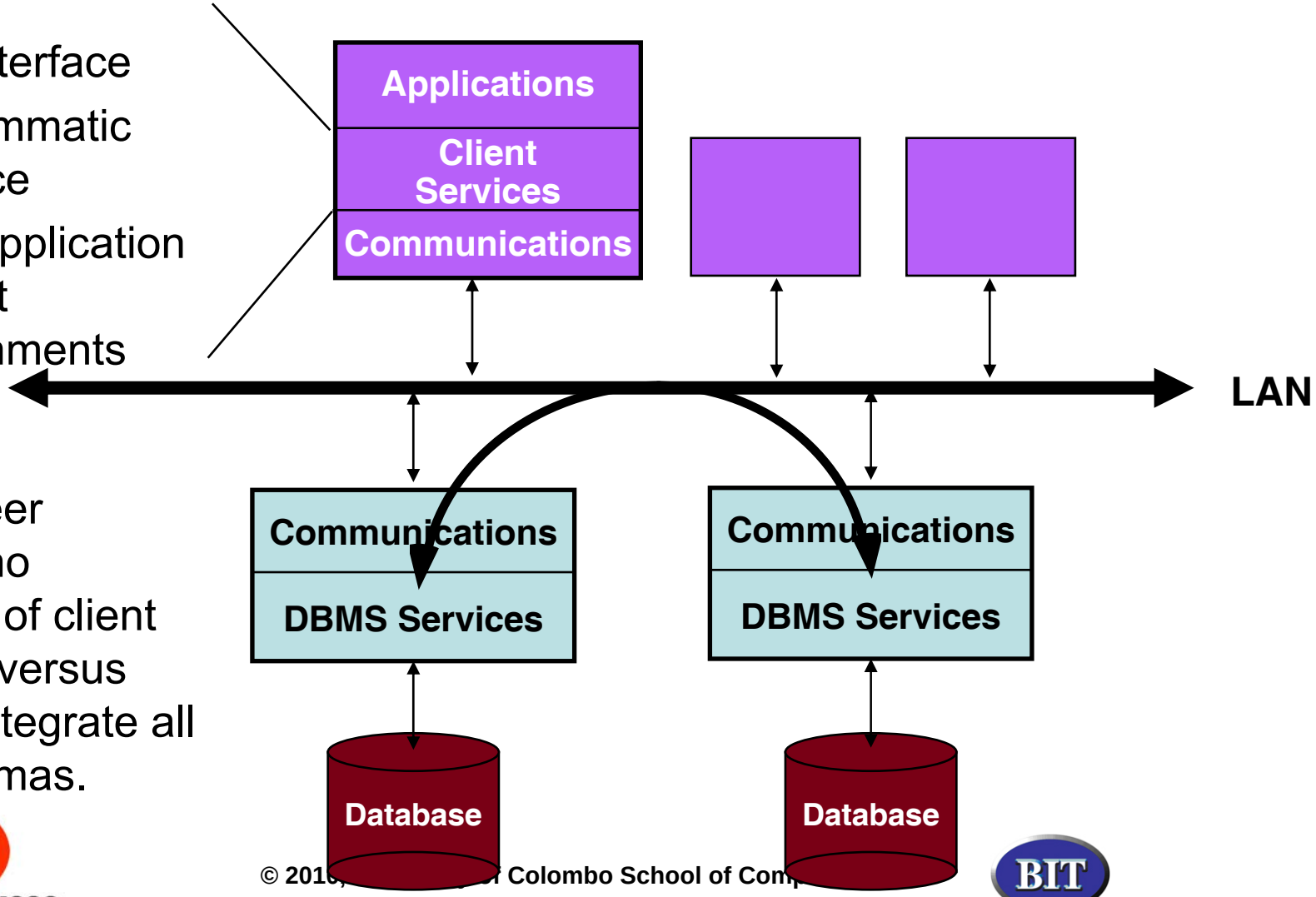
- directory
- caching
- query decomposition
- commit protocols



Client-Server, communication duties are shared between the client machines and servers.

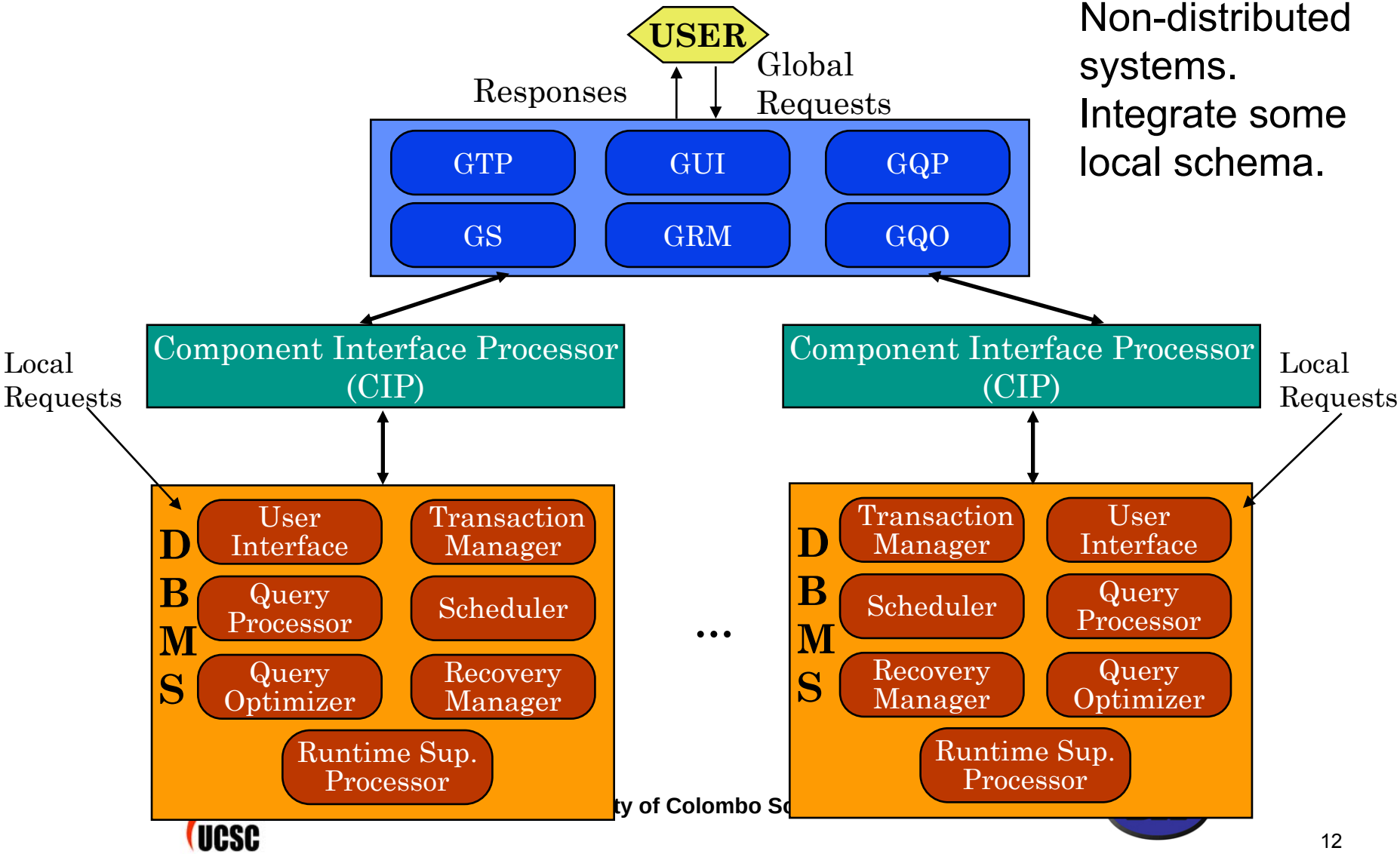
Peer-to-Peer (P2P) or Server-to-Server

- SQL interface
- programmatic interface
- other application support environments



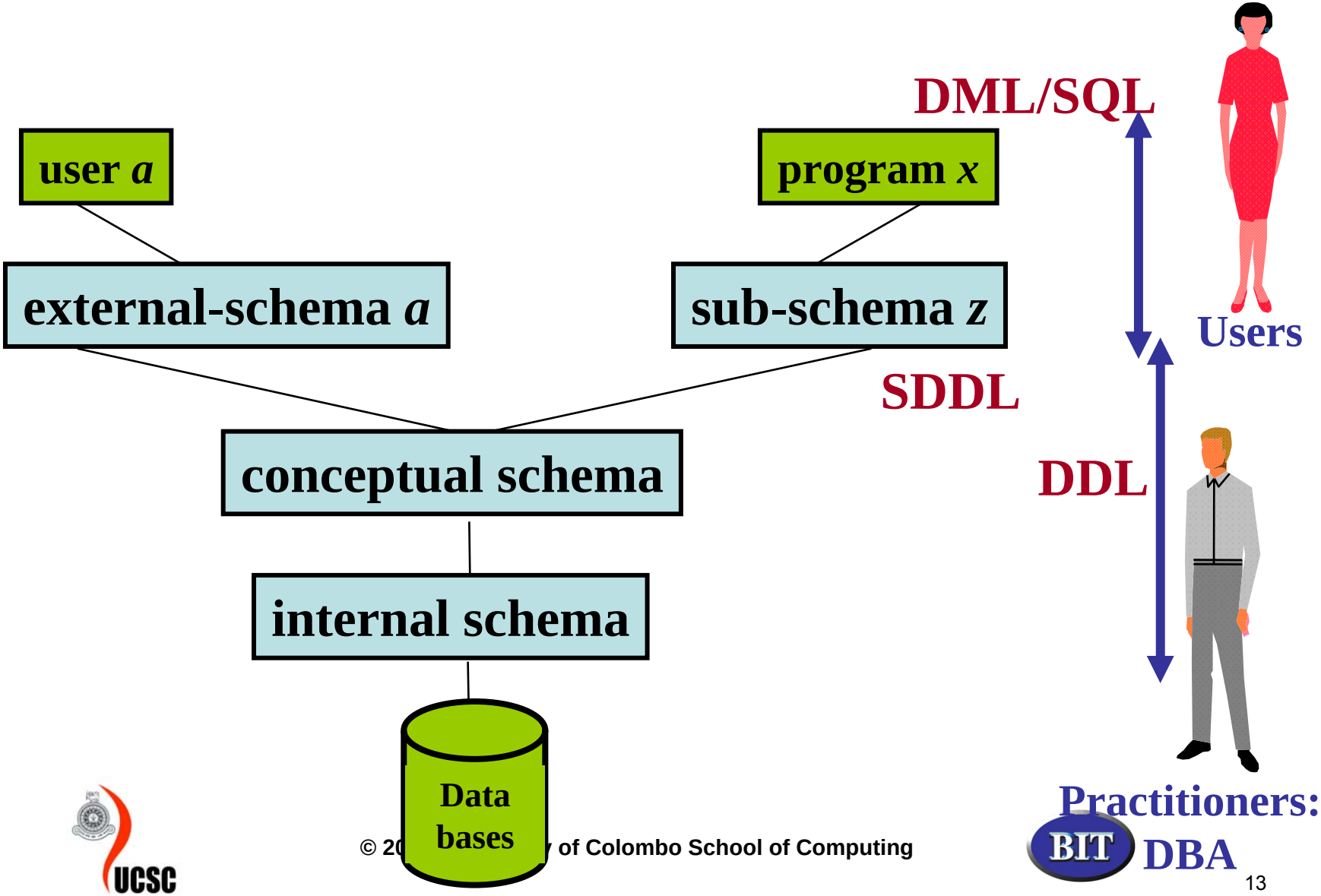
Peer-to-peer systems, no distinction of client machines versus servers. Integrate all local schemas.

Multi-DBMS

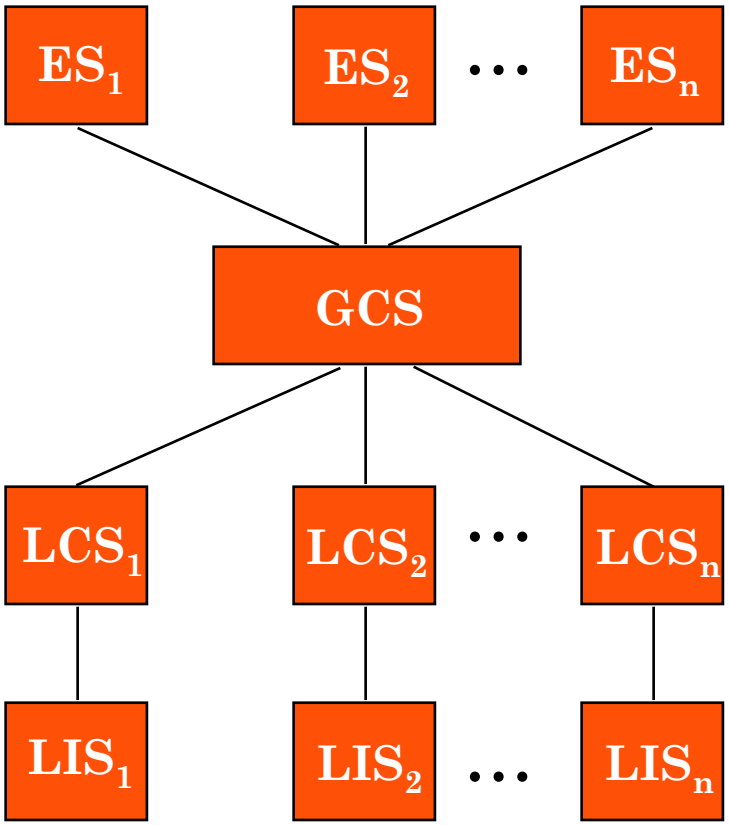


Non-distributed systems. Integrate some local schema.

3 Level Architecture



Distributed DBMS Architecture



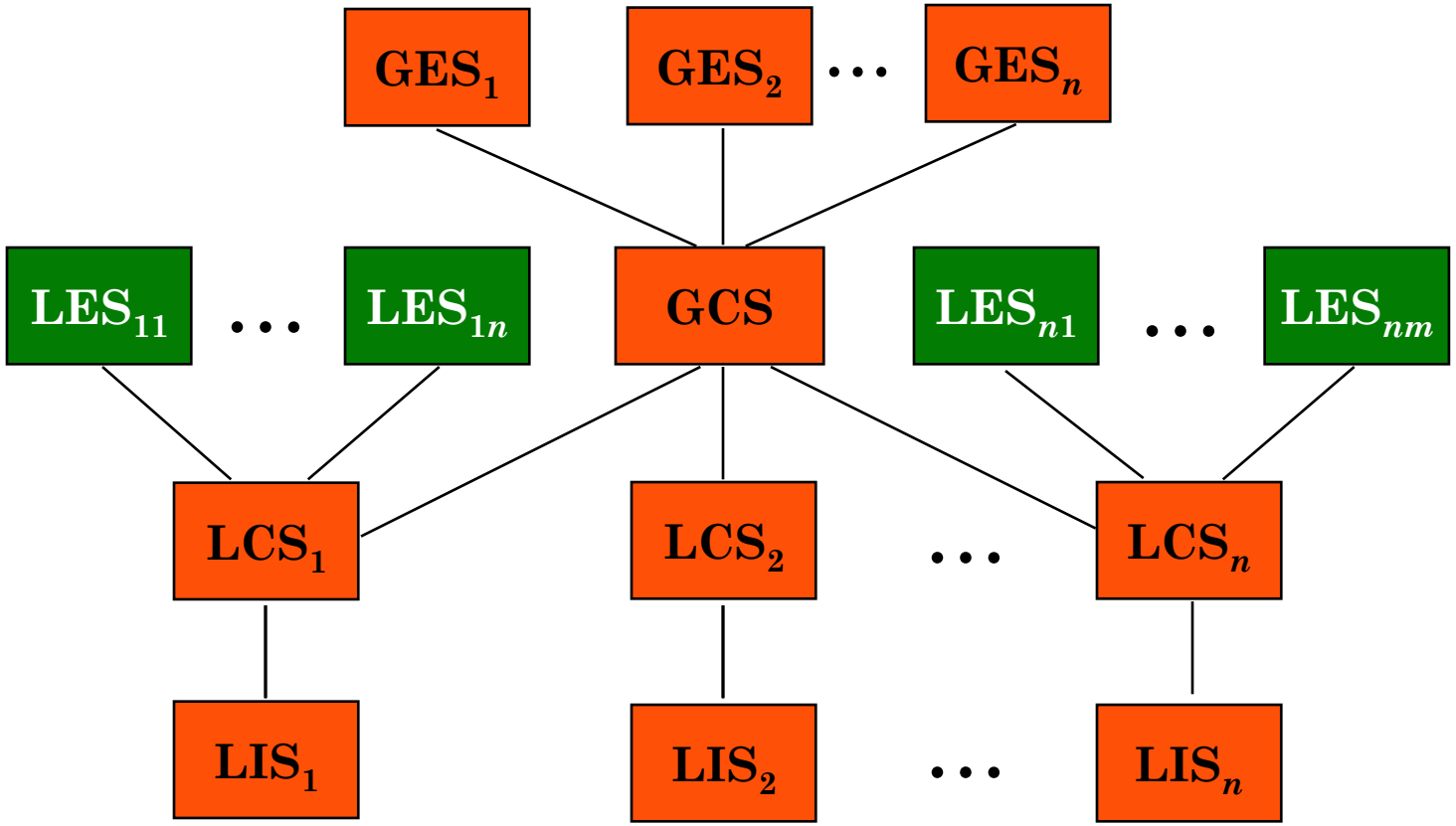
ES: External Schema

GCS: Global Conceptual Schema

LCS: Local Conceptual Schema

LIS: Local Internal Schema

Multi-DBMS Architecture

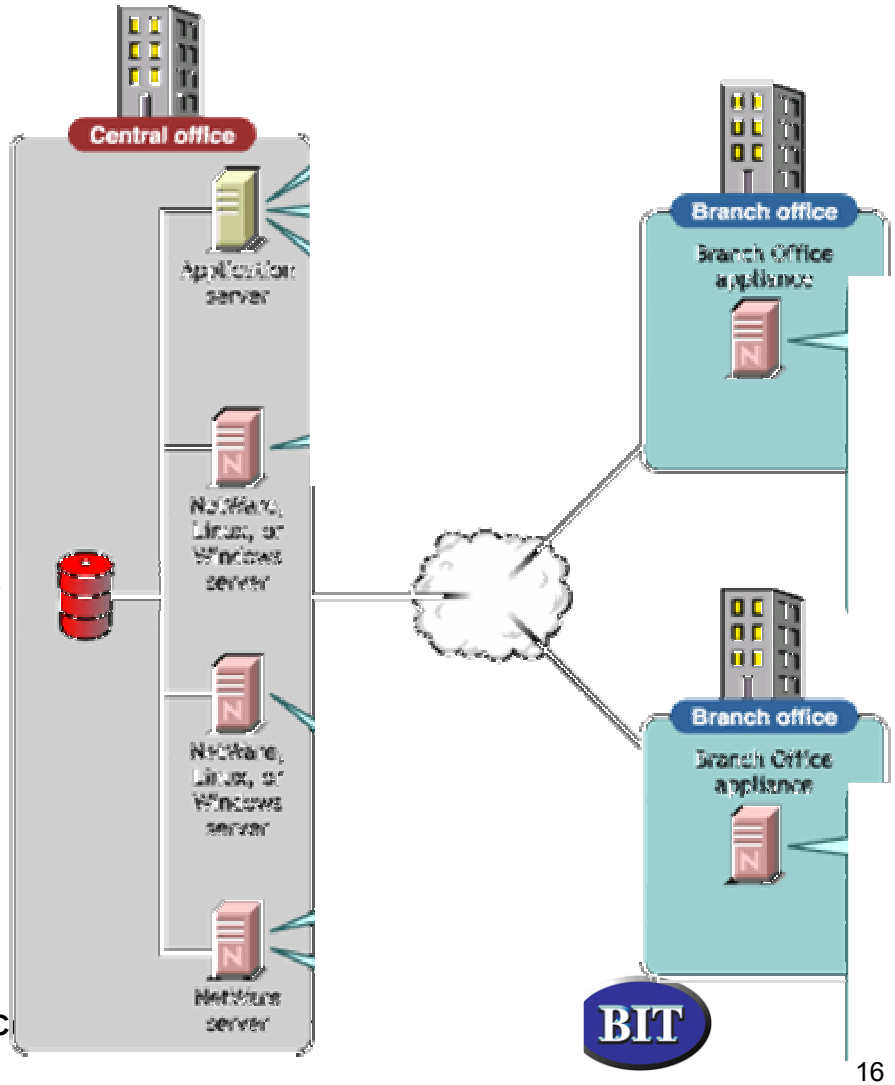


- GES: Global External Schema
- LES: Local External Schema
- LCS: Local Conceptual Schema
- LIS: Local Internal Schema

Motivation for Distributed Databases

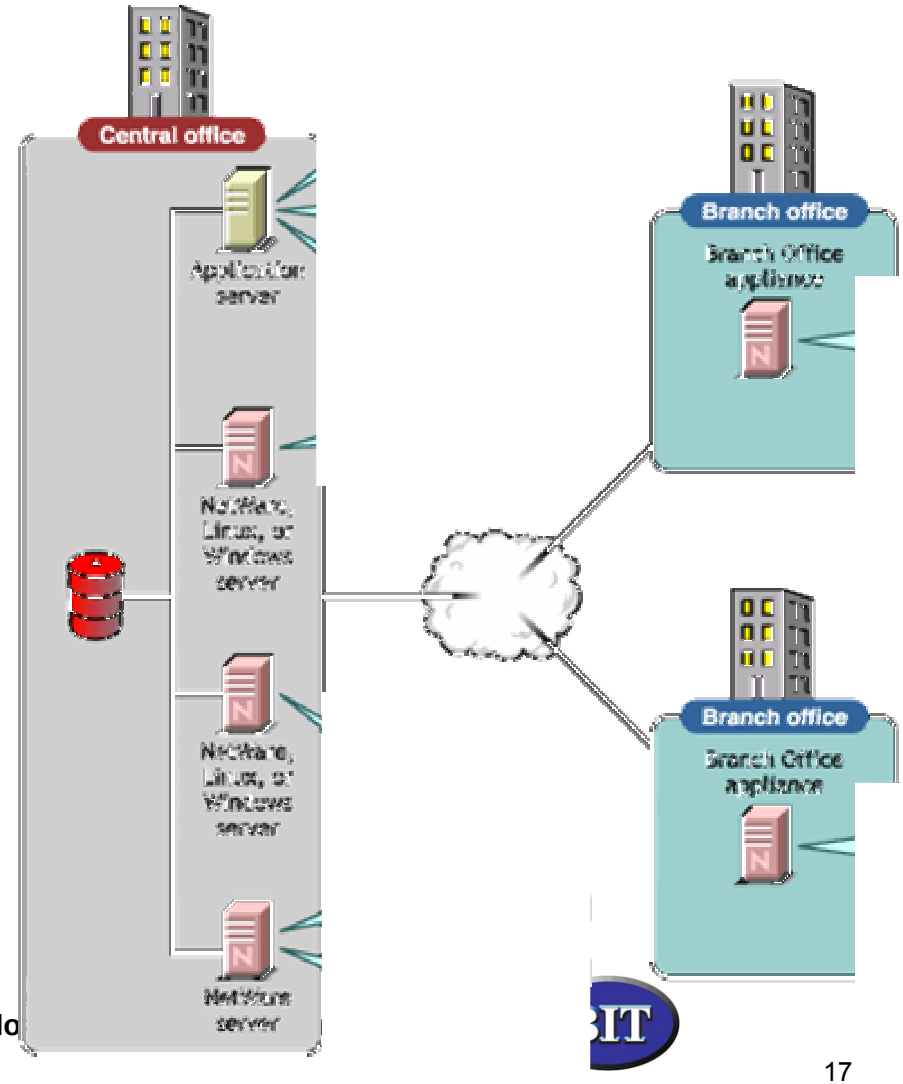
Organisational and economic reason - Many organisations are decentralised and a distributed database approach fits more naturally the structure of the organisation.

E.g. Banks.



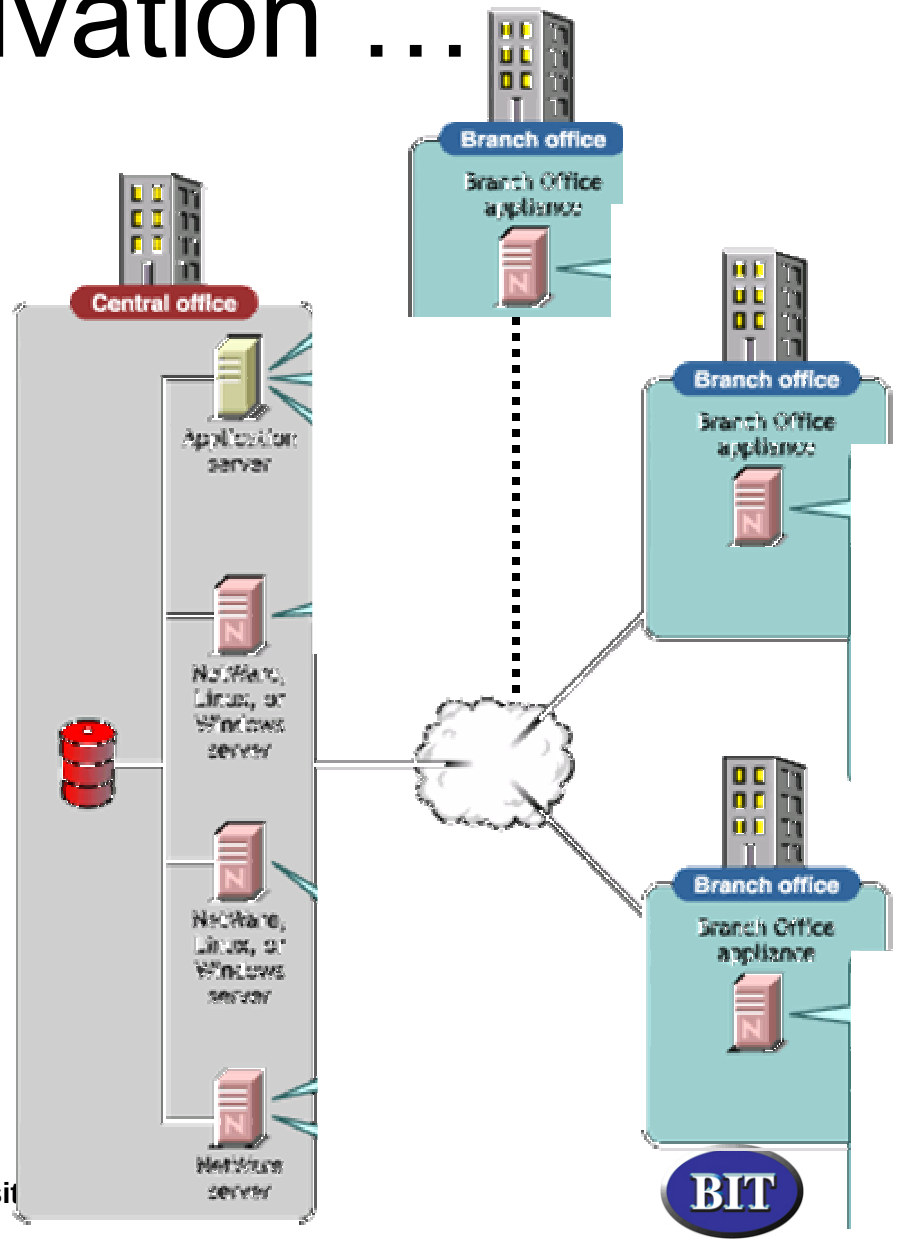
Motivation for Distributed Databases

Interconnection of existing databases - Distributed databases are a natural solution when several databases already exist in an organisation and the necessity of performing global applications arises.



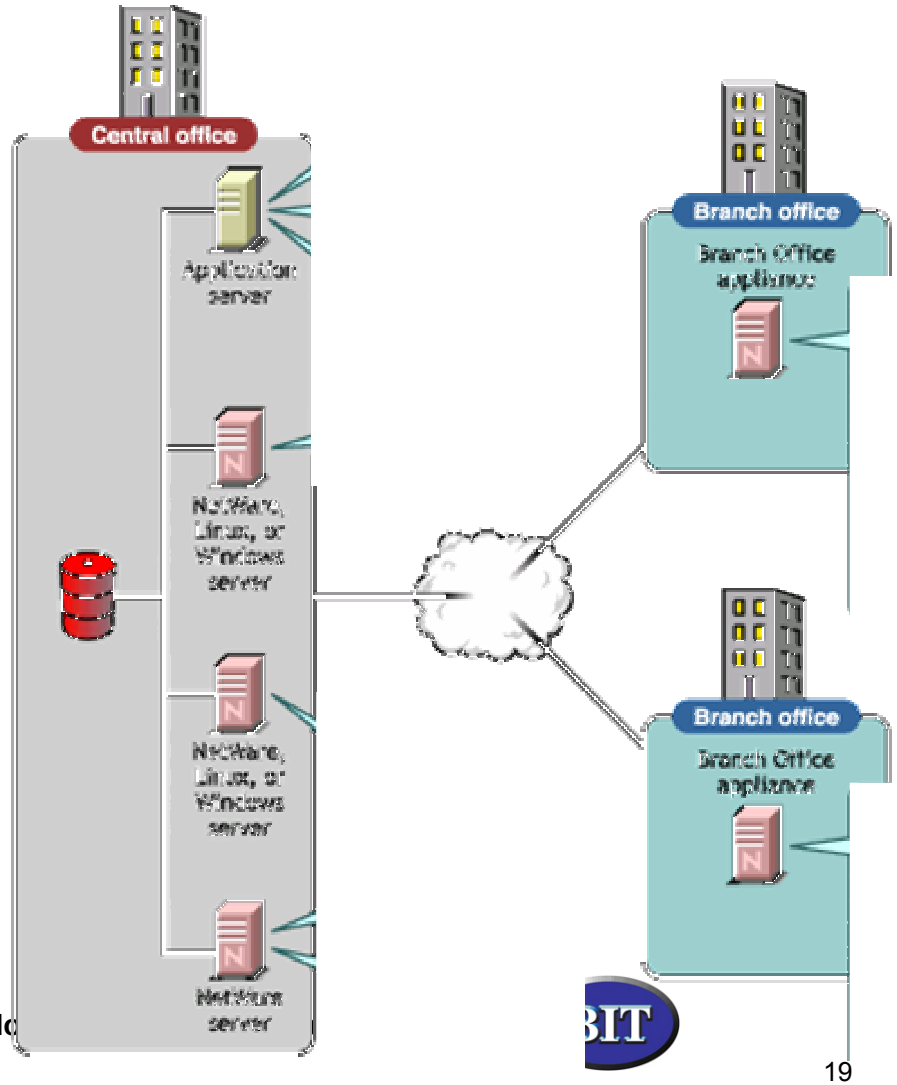
Motivation …

- Incremental growth - Supports organisational growth (new branches) in a smoother manner than with a centralised database approach.



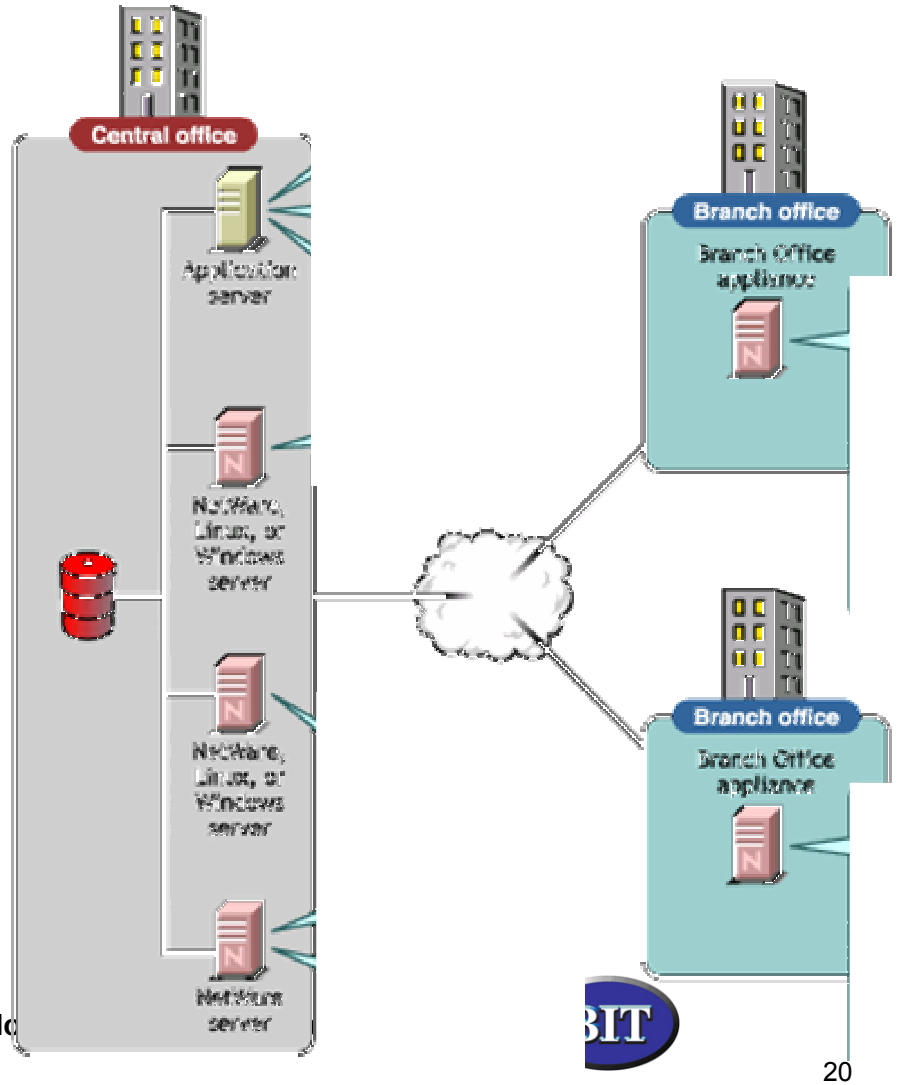
Motivation for Distributed Databases

- Allow data sharing while maintaining some measure of local control (autonomy).



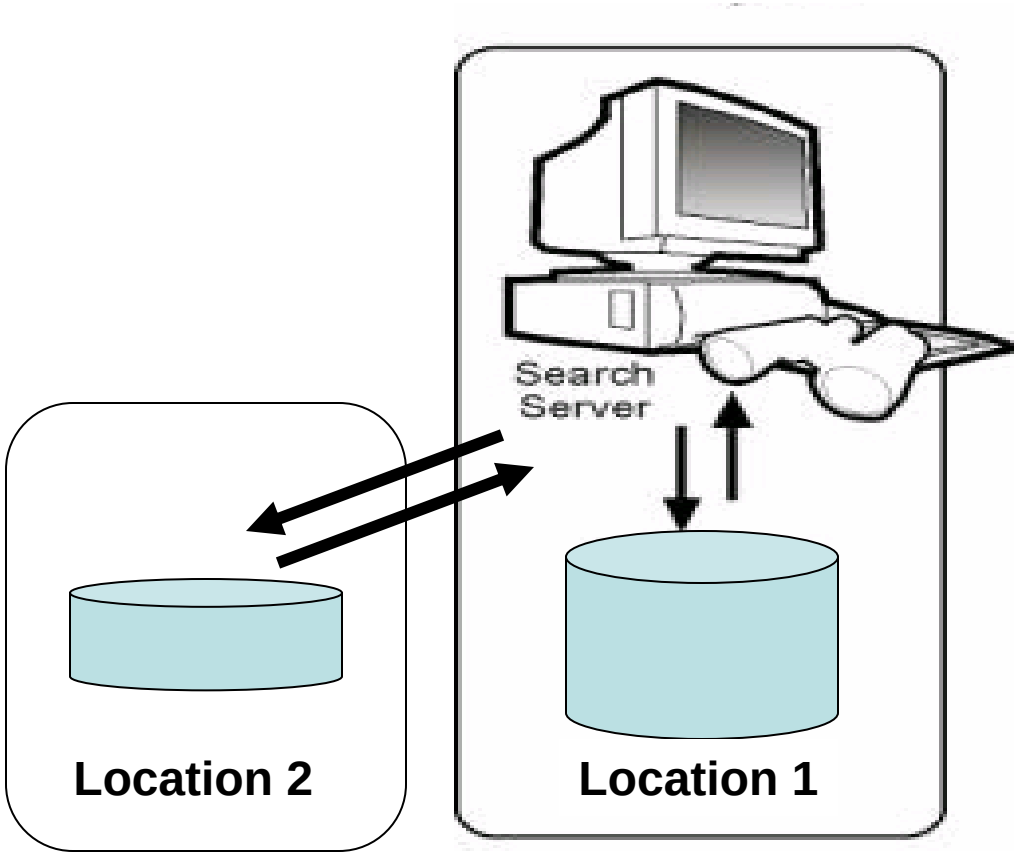
Motivation for Distributed Databases

- Reduce Communication overhead
  - This is not automatically guaranteed by distribution, but depends largely on the quality of the distributed database design.



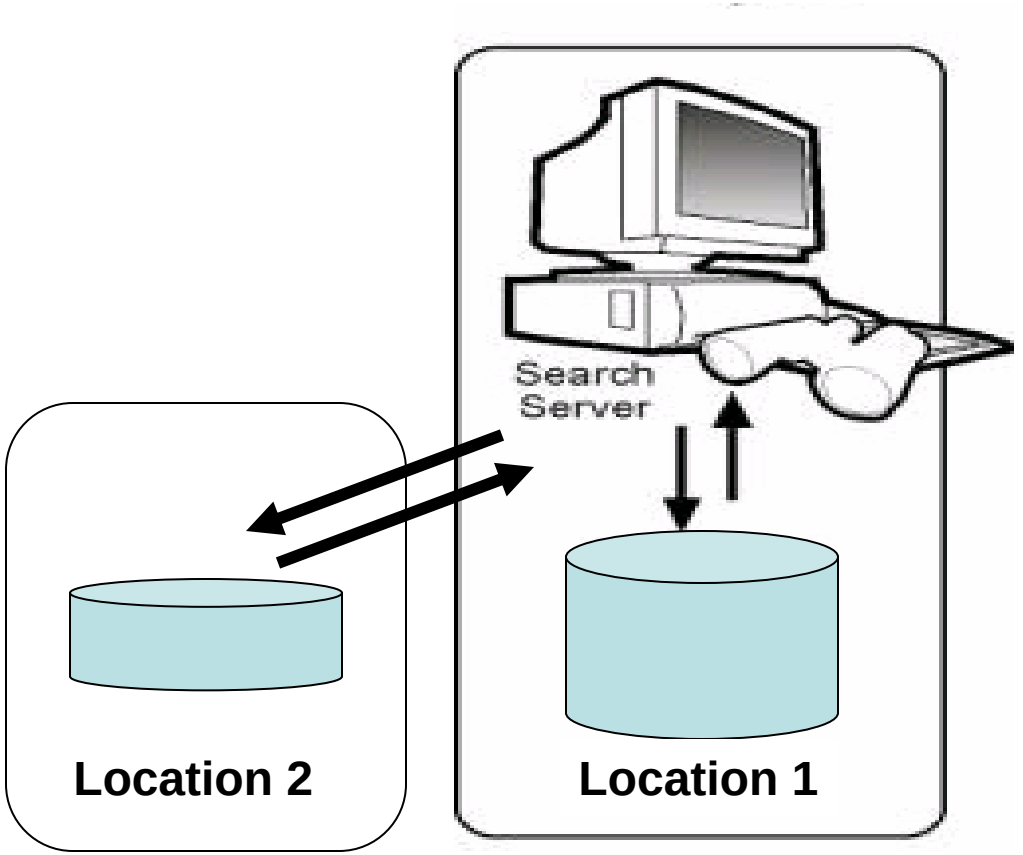
Motivation for Distributed Databases

- Performance Consideration
  - The existence of several processors results in better performance through the use of parallelism. Smaller databases exist at each site and hence, local queries and transactions are improved.



Motivation for Distributed Databases

- Increased reliability and availability
  - The use of multiple components means that higher reliability can be obtained. Also data replication can be used to increase availability of data.



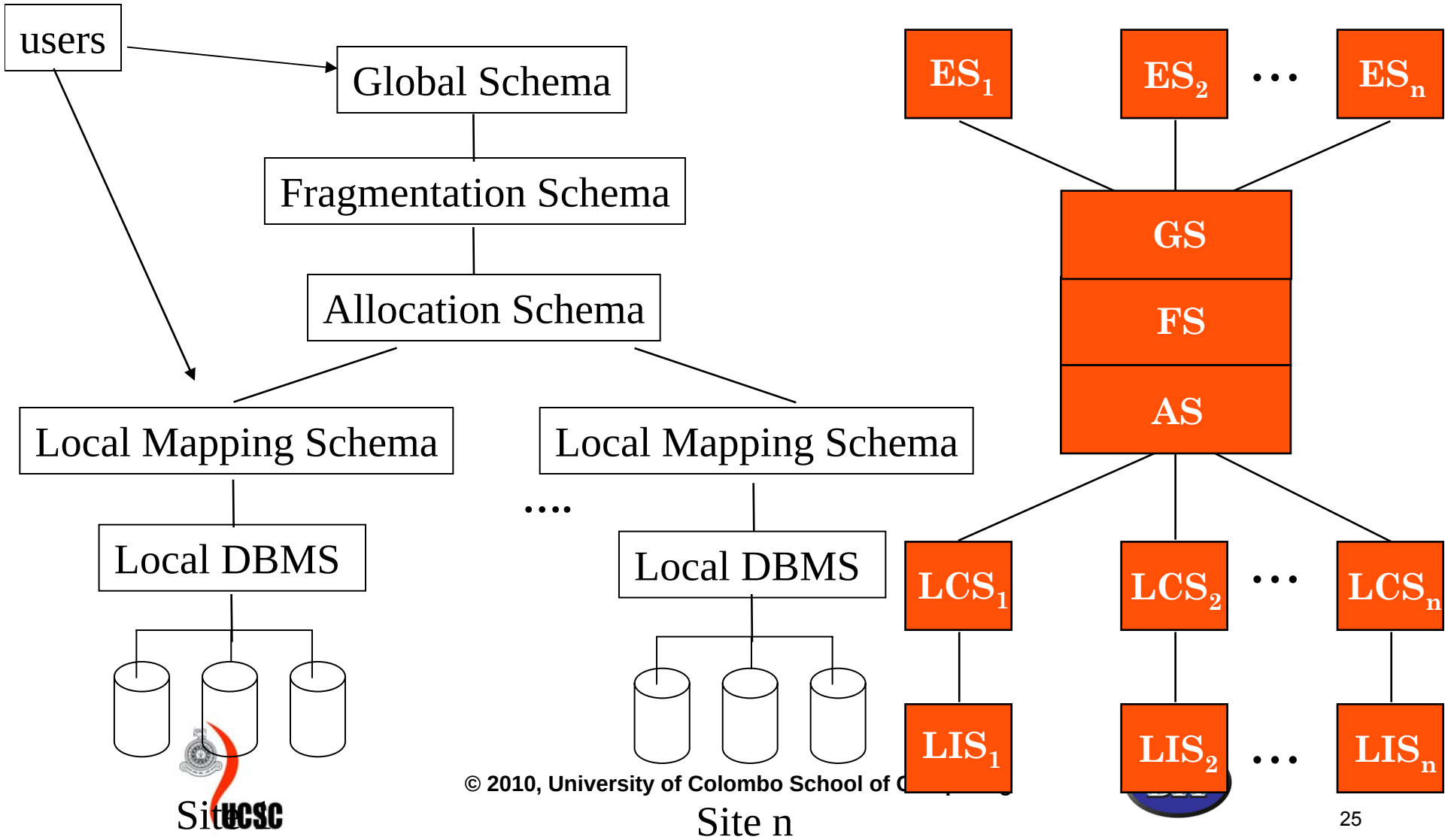
Advantages of distributed databases

- Reflects organizational structure — database fragments are located in the departments they relate to.
- Local autonomy — a department can control the data about them (as they are the ones familiar with it.)
- Improved availability — a fault in one database system will only affect one fragment, instead of the entire database.
- Improved performance — data is located near the site of greatest demand, and the database systems themselves are parallelized, allowing load on the databases to be balanced among servers. (A high load on one module of the database won't affect other modules of the database in a distributed database.)
- Economics — it costs less to create a network of smaller computers with the power of a single large computer.
- Modularity — systems can be modified, added and removed from the distributed database without affecting other modules (systems).

Disadvantages of distributed databases

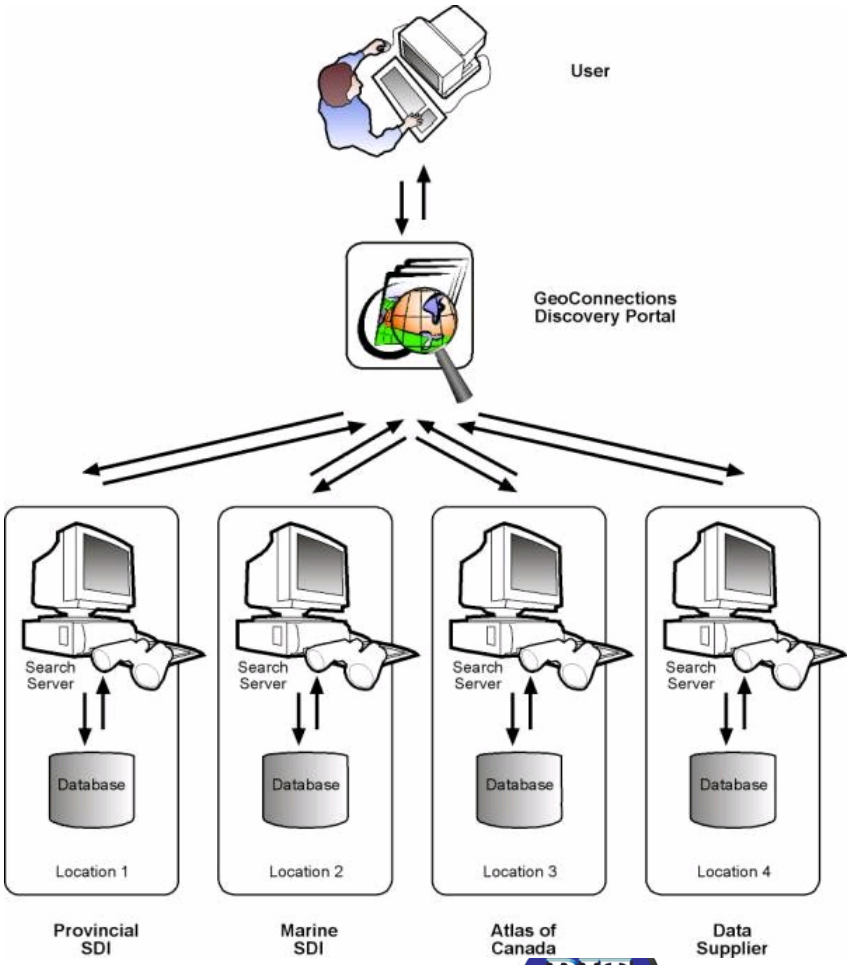
- Complexity — extra work must be done by the DBAs to ensure that the distributed nature of the system is transparent. Extra work must also be done to maintain multiple disparate systems, instead of one big one. Extra database design work must also be done to account for the disconnected nature of the database — for example, joins become prohibitively expensive when performed across multiple systems.
- Economics — increased complexity and a more extensive infrastructure means extra labour costs.
- Security — remote database fragments must be secured, and they are not centralized so the remote sites must be secured as well. The infrastructure must also be secured (e.g., by encrypting the network links between remote sites).
- Difficult to maintain integrity — in a distributed database, enforcing integrity over a network may require too much of the network's resources to be feasible.
- Inexperience — distributed databases are difficult to work with, and as a young field there is not much readily available experience on proper practice.

General Architecture for a Distributed Databases



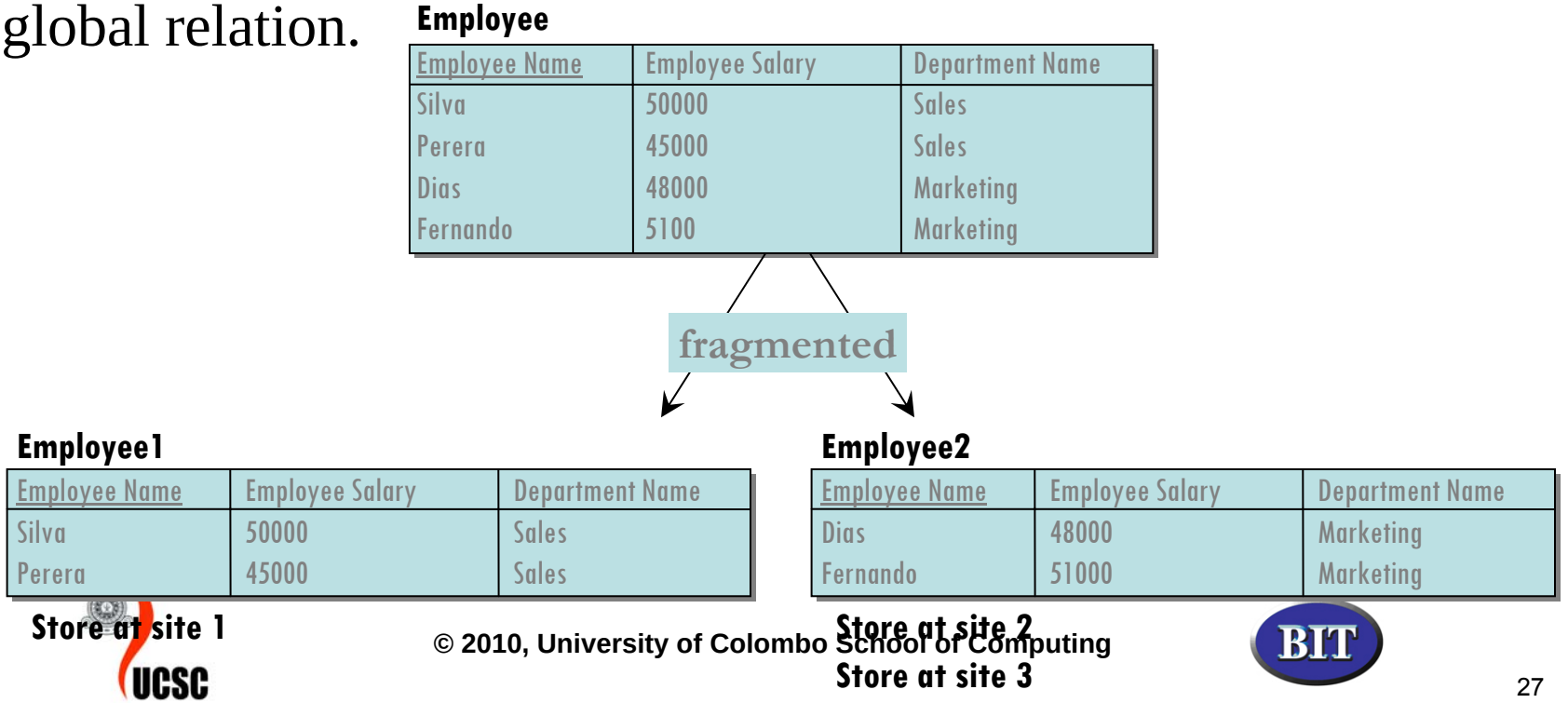
General Architecture for a Distributed Databases

Global Schema
Defines all the data contained in the distributed database as if the data were not distributed at all. It consists of a set of global relations.



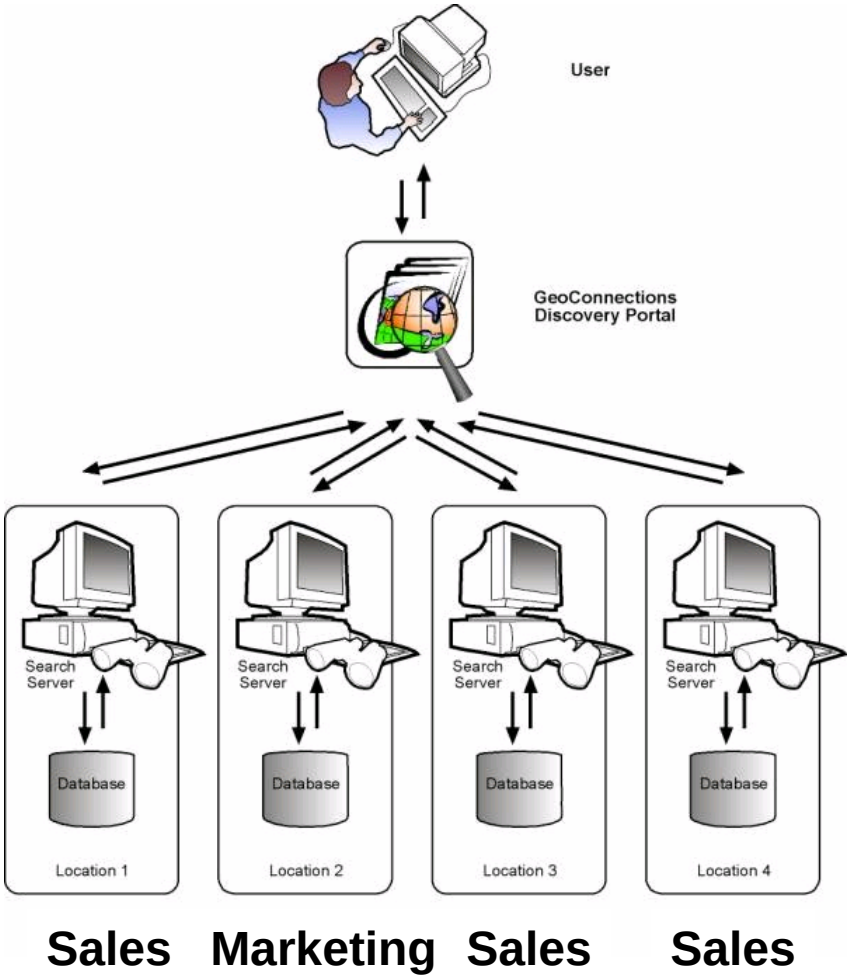
General Architecture for a Distributed Databases

Fragmentation Schema - Each global relation can be split into several non-overlapping portions called fragments. The mapping between global relations and fragments is defined in the fragmentation schema. Here, several fragments correspond to one global relation.



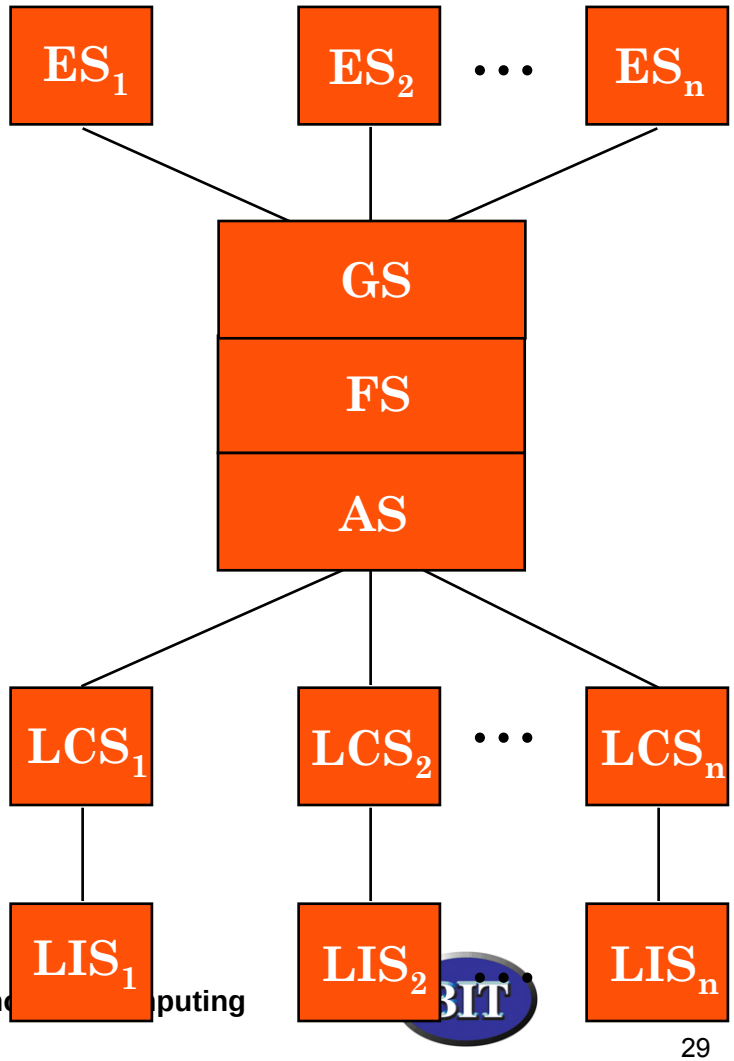
General Architecture for a Distributed Databases

Allocation Schema - Fragments are physically located at one or several sites of the network. The mapping defined in the allocation schema determines whether the distributed database is redundant or non-redundant. Each allocation corresponds to a fragment if the data is non-redundant otherwise several allocations will correspond to a fragment.



General Architecture for a Distributed Databases

Local Mapping Schemes - The top three levels are site independent. They do not depend on the data model of the local DBMS. At the lower level, it is necessary to map the objects to those, which are manipulated by the local DBMS. This mapping is called the local mapping schema.



Types of data transparency

- Data structure: “data independence” of centralized DBMSs
- Location of fragments: “location transparency”
- Existence of fragments: “fragmentation transparency”
- Replication of fragments: “replication transparency”

Transparencies

The distribution is transparent — users must be able to interact with the system as if it were one logical system. This applies to the system's performance, and methods of access amongst other things.
Transactions are transparent — each transaction must maintain database integrity across multiple databases. Transactions must also be divided into sub-transactions, each sub-transaction affecting one database system..

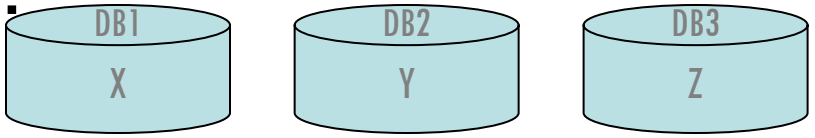
Network / Distribution / Location transparency – DBMSs presented globally to user as though a single centralised DBMS; Global DD holds location of each underlying table; DDBMS performs query decomposition & result joining without user being made aware

Distributed Execution Plan: Example

- Transaction

```
Read X;
Read Y;
Read Z;
Report X+Y+Z to user
```

- Allocation information:



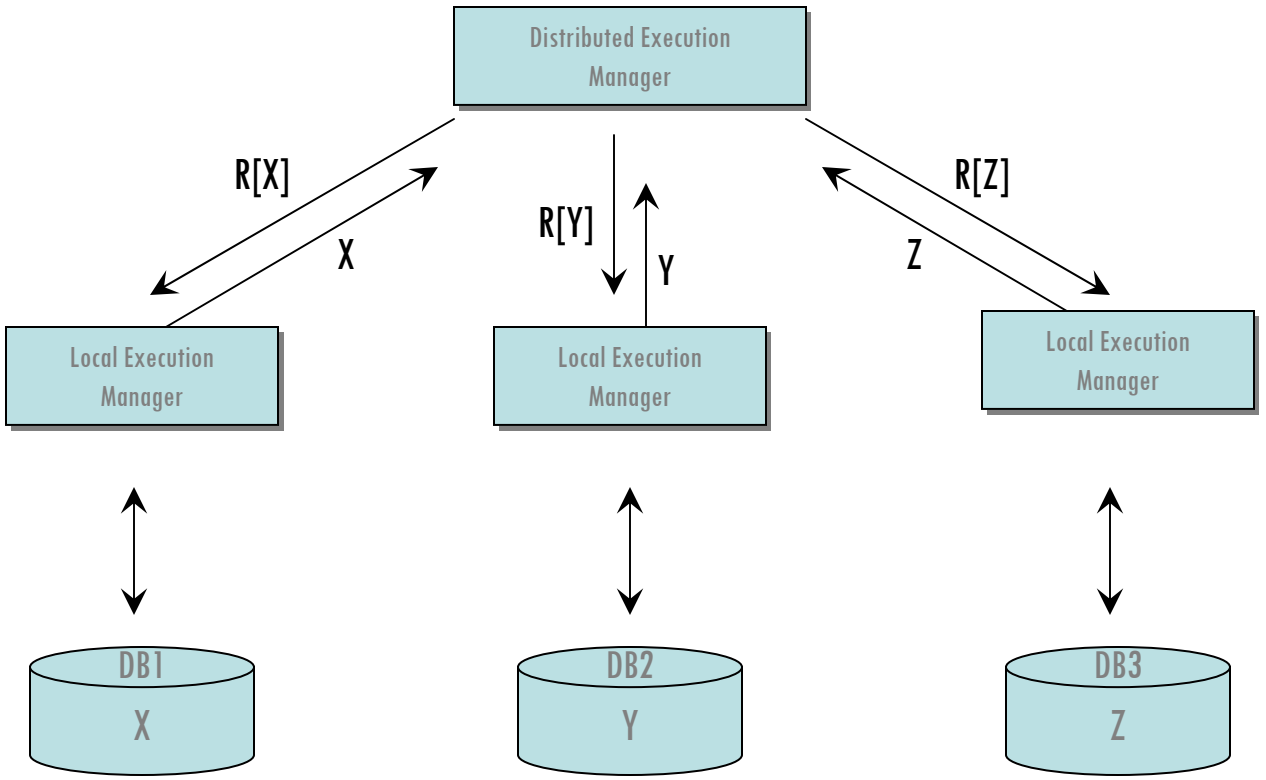
- Global execution plan

```
Read X from DB1
Read Y from DB2
Read Z from DB3
Calculate X+Y+Z
Report X+Y+Z
```

Subtransactions to be executed at different sites

Distributed Execution Plan: Example, cont.

- Global execution plan ↓



Report X+Y+Z

R[x] means Read x

Transparencies

```
SELECT Salary
FROM Employee
WHERE Name="Dias";
```

Fragmentation transparency – For access purposes tables may be split (fragmented) vertically or horizontally; Details of fragmentation kept in Global Data Dictionary; User views global DB without awareness of fragmentation

Employee

Employee Name	Employee Salary	Department Name
Silva	50000	Sales
Perera	45000	Sales
Dias	48000	Marketing
Fernando	5100	Marketing

fragmented

Employee1

Employee Name	Employee Salary	Department Name
Silva	50000	Sales
Perera	45000	Sales

Store at site 1

Employee2

Employee Name	Employee Salary	Department Name
Dias	48000	Marketing
Fernando	51000	Marketing

Store at site 2

Store at site 3

No Fragmentation
No Location
Transparencies

```
SELECT Salary
FROM Employee1 at Site1
WHERE Name="Dias"
UNION
SELECT Salary
FROM Employee2 at Site2
WHERE Name="Dias";
```

Employee

Employee Name	Employee Salary	Department Name
Silva	50000	Sales
Perera	45000	Sales
Dias	48000	Marketing
Fernando	5100	Marketing

fragmented

Employee1

Employee Name	Employee Salary	Department Name
Silva	50000	Sales
Perera	45000	Sales

Store at site 1

Employee2

Employee Name	Employee Salary	Department Name
Dias	48000	Marketing
Fernando	51000	Marketing

Store at site 2

Store at site 3

No Fragmentation Location Transparencies

```
SELECT Salary
FROM Employee1
WHERE Name="Dias"
UNION
SELECT Salary
FROM Employee2
WHERE Name="Dias";
```

Employee

Employee Name	Employee Salary	Department Name
Silva	50000	Sales
Perera	45000	Sales
Dias	48000	Marketing
Fernando	5100	Marketing

fragmented

Employee1

Employee Name	Employee Salary	Department Name
Silva	50000	Sales
Perera	45000	Sales

Store at site 1

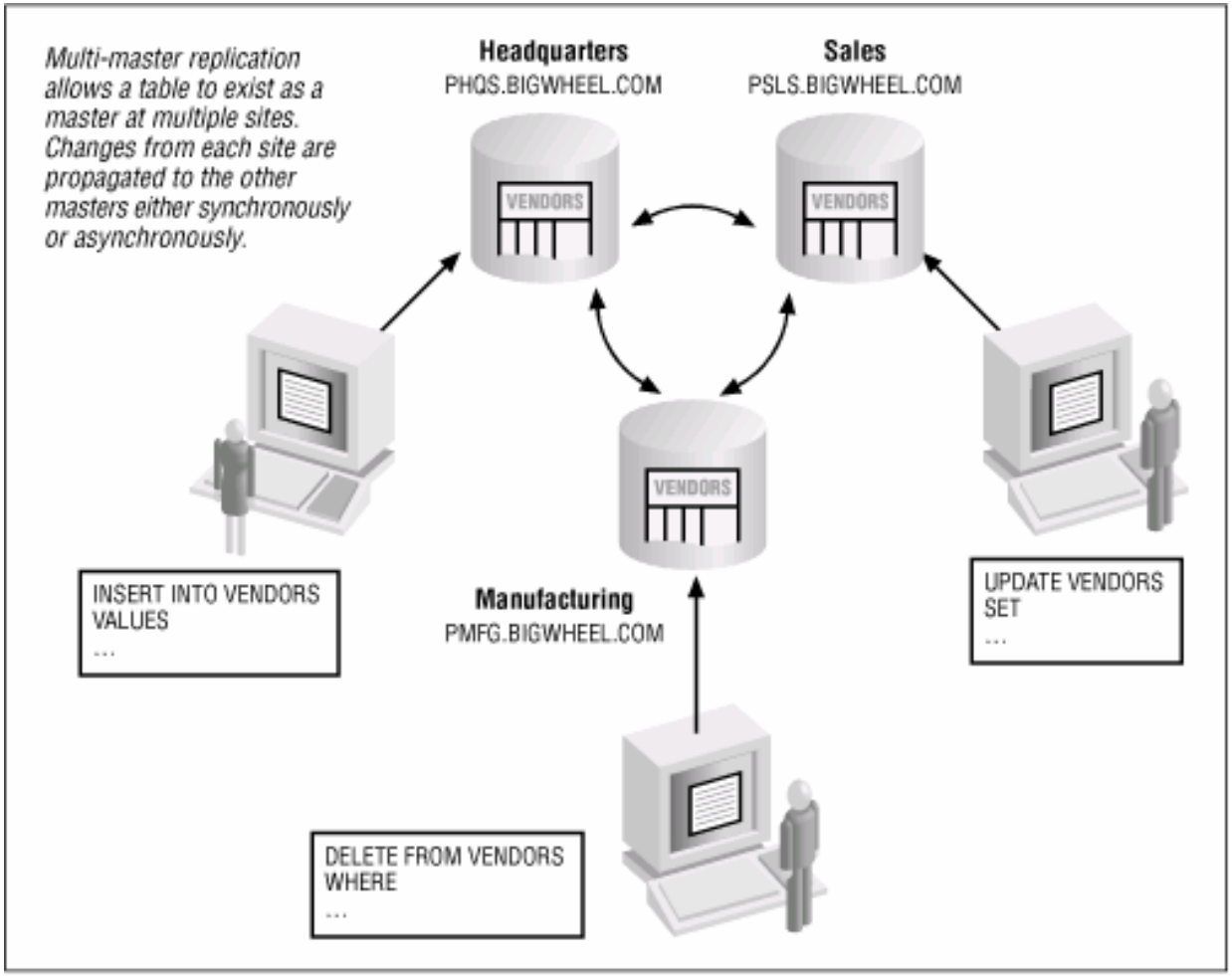
Employee2

Employee Name	Employee Salary	Department Name
Dias	48000	Marketing
Fernando	51000	Marketing

Store at site 2

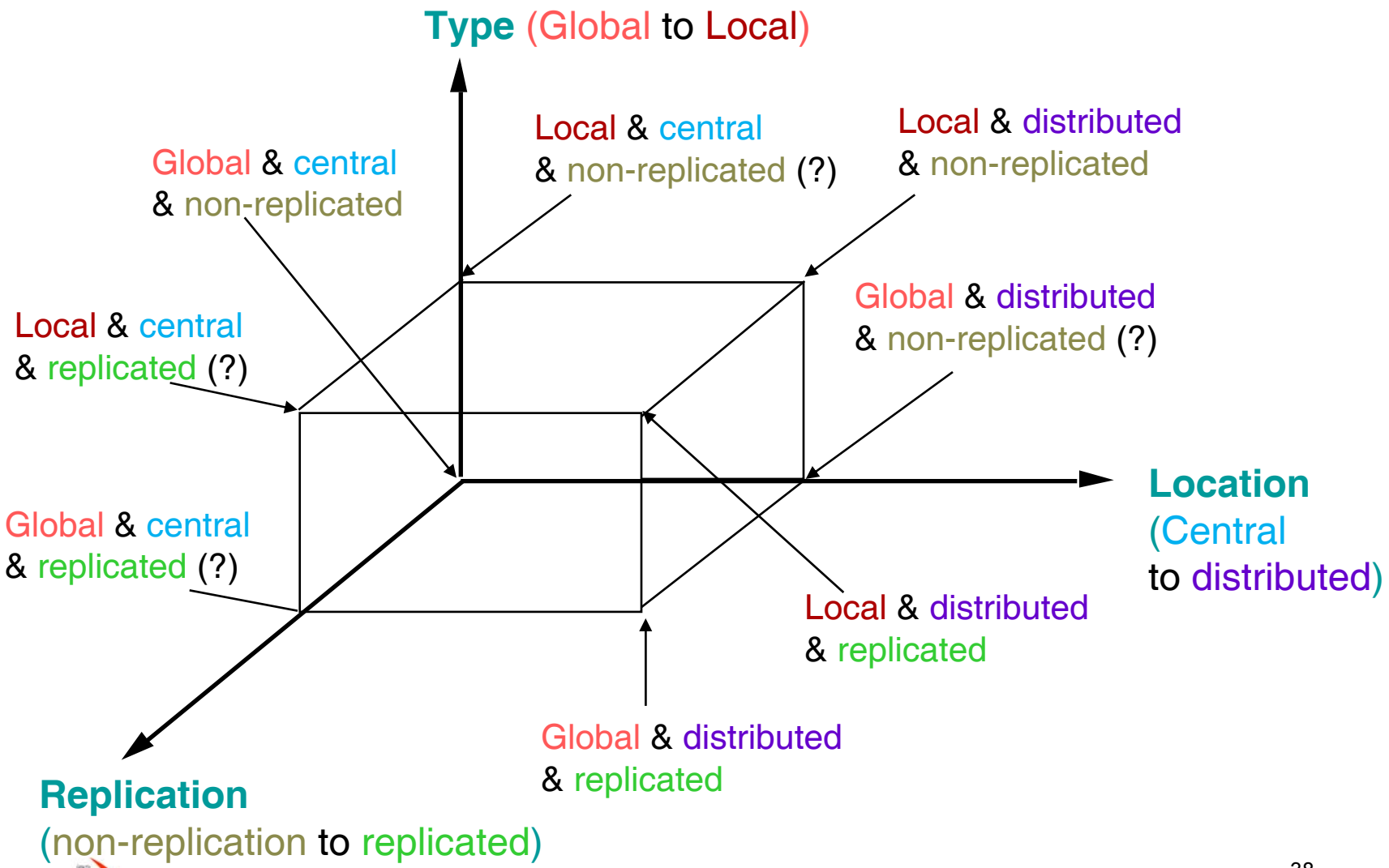
Store at site 3

Replication



Replication transparency – relations may be duplicated for local/global access requirements; DDBMS handles replication without user awareness; issue of update propagation

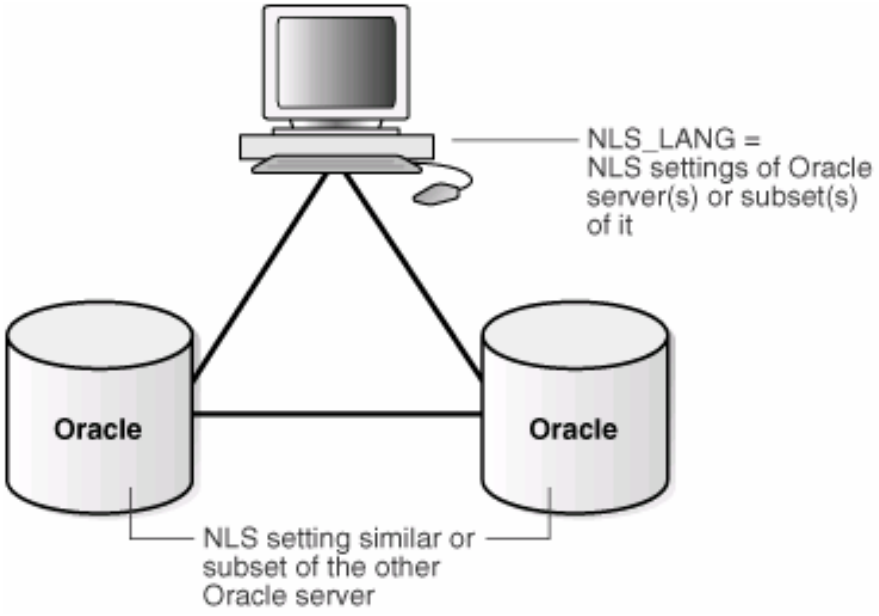
Directory Issues



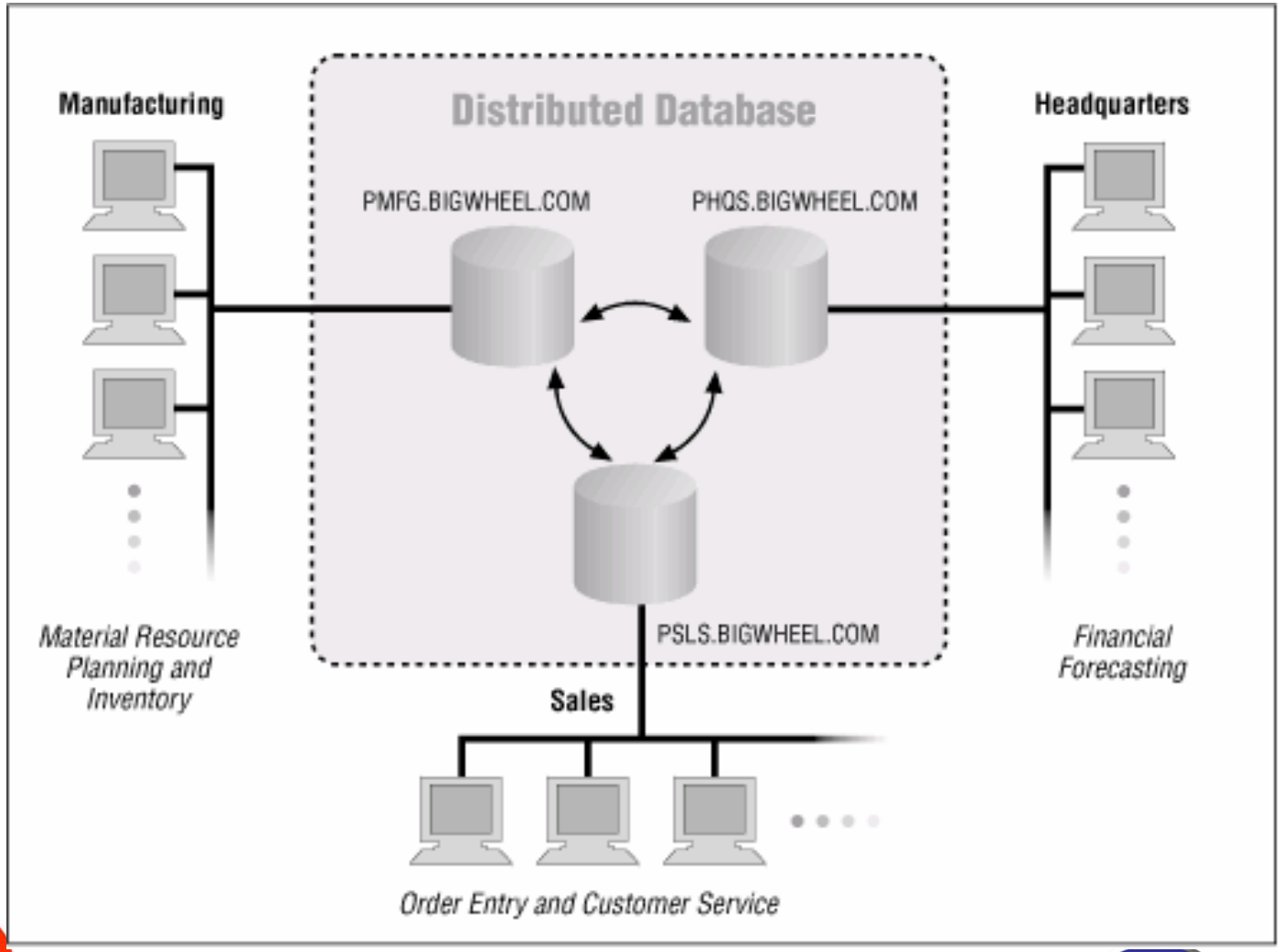
Types of DDBMS

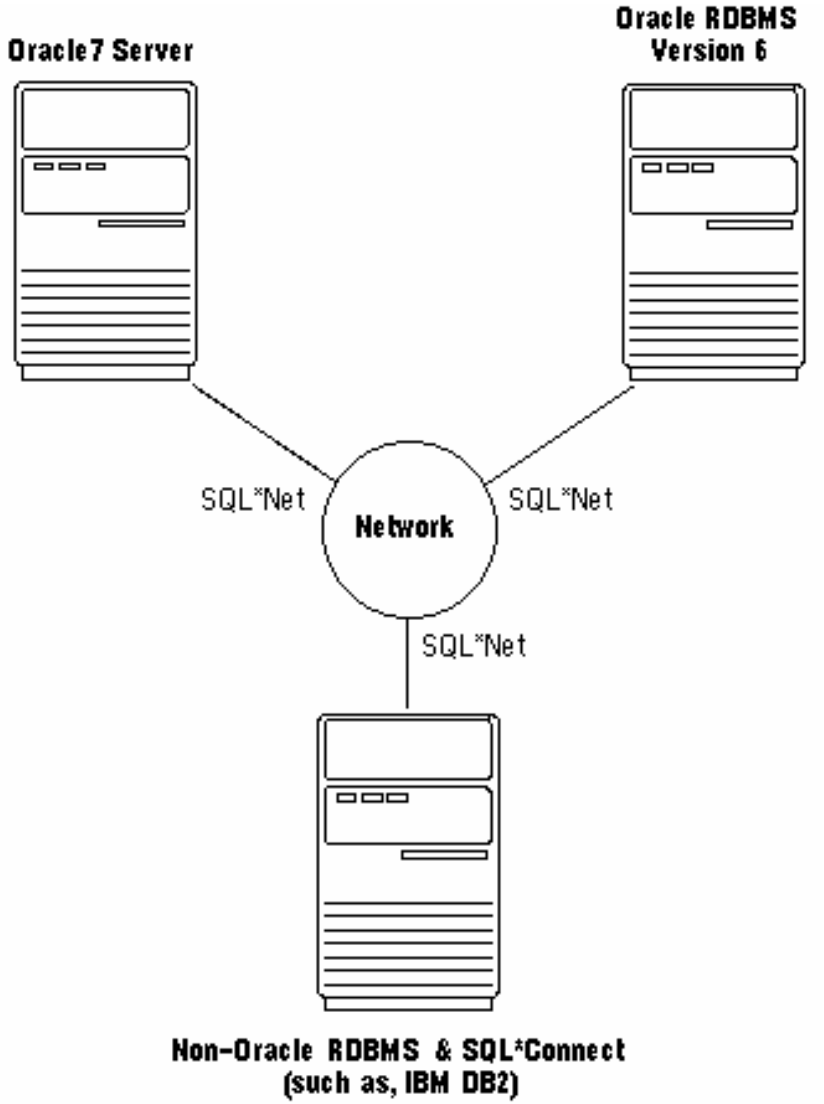
Homogeneous DDBMS - Homogeneous DDBMSs are software to integrate DBMSs geographically distributed over network; all DBMSs have same data model/query language; hardware is same manufacturer/operating system (server & user).

uses one DBMS (e.g.: MS-SQL or Oracle)



Homogeneous DDBMS





Heterogeneous DDBMS – difference in local DBMS and user software
uses multiple DBMS's (e.g.: Oracle and MS-SQL and PostgreSQL).

Autonomy

Local applications

applications which do not require data from other sites.

Global applications

applications which do require data from other sites.

Degree of local autonomy

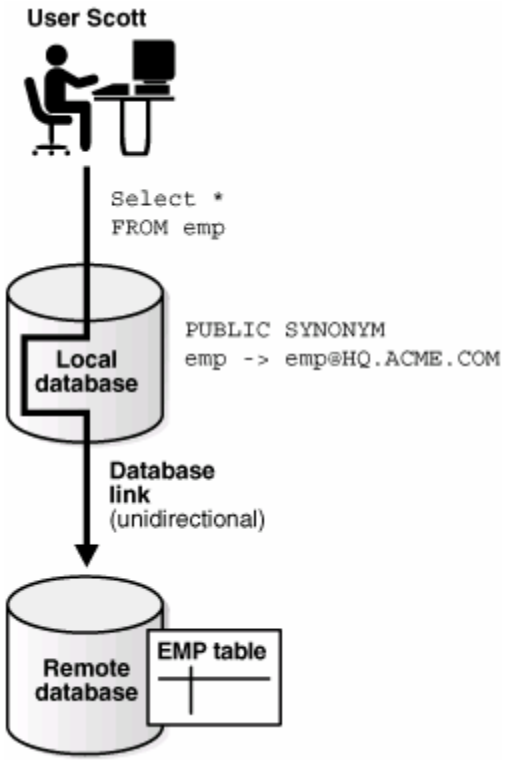
Access to DDBMS via global schema (client) – no local autonomy, or direct by local schema (server) – some degree of local autonomy.

• If single integrated schema then high degree of distribution transparency - no distribution details.

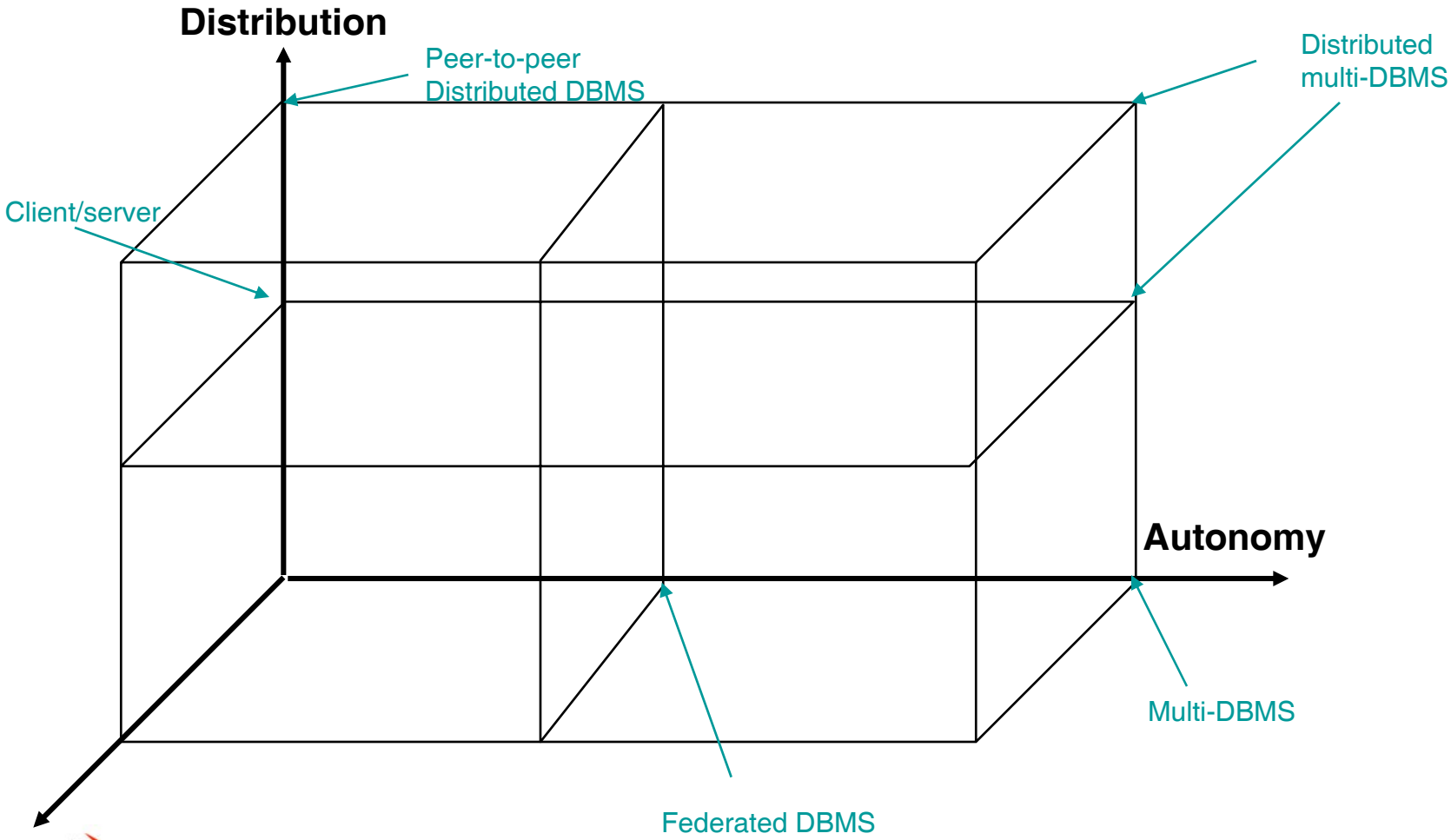
• If no integrated schema then distribution details are required to formulate queries - no distribution transparency.

Autonomy

Models without Global Conceptual Schemas
each site has DBMS of it own. Share data by defining export schemas.
Federated (semiautonomous)
Multi-database (autonomous)



Alternatives in Distributed Database Systems



Distribution Design Issues

- Why fragment at all?
- How to fragment?
- How much to fragment?
- How to test correctness?
- How to allocate?
- Information requirements?

Fragmentation

- Can't we just distribute relations?
- What is a reasonable unit of distribution?
  - relation
    - views are subsets of relations
    - 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

Design

Distributed Database Design

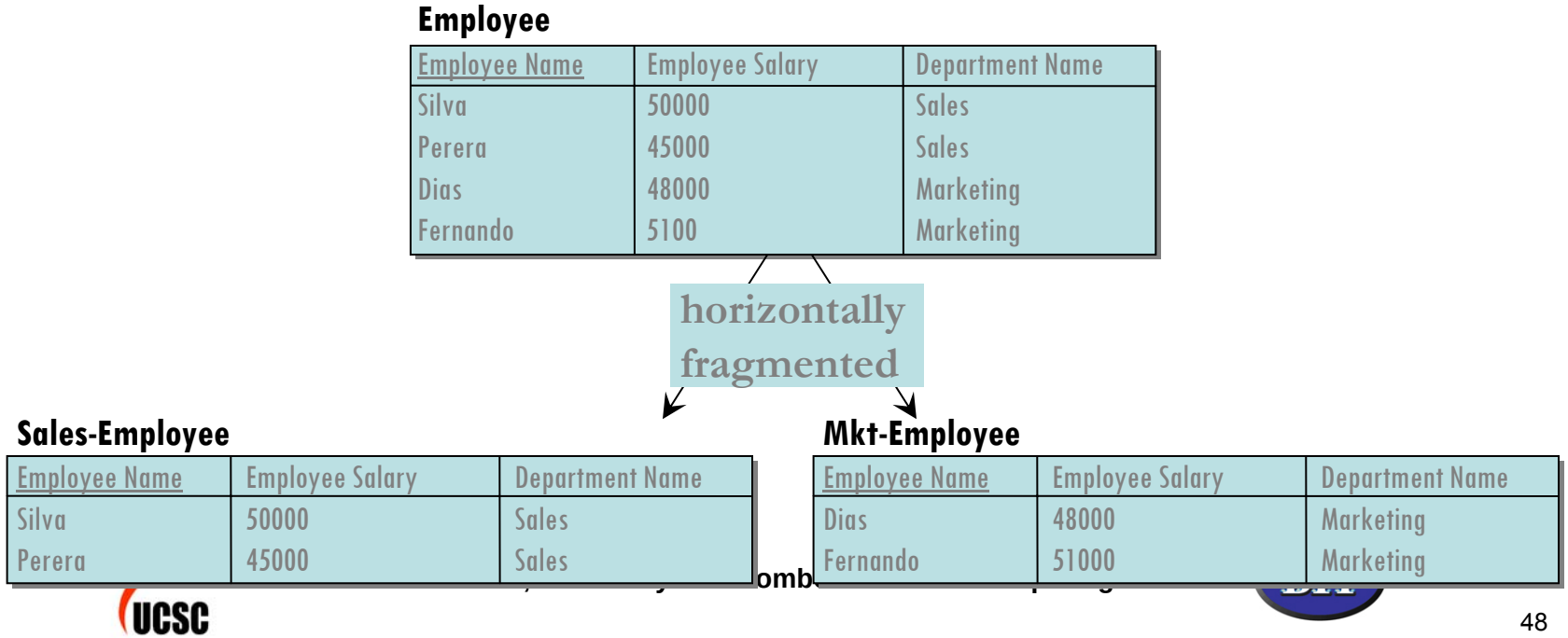
Objectives of distributed database design are separation of **data fragmentation** from **data allocation**; control of **data redundancy**; independence from local DBMSs

A database is broken into logical units called fragments and assigned for storage at various sites. Data fragmentation is partitioning data into number of disjoint subsets.

Fragmentation

Horizontal fragmentation partitions the records of a global table into subsets. A horizontal fragment keeps only certain rows of the global relation. The reconstruction is done by taking the union of all fragments.

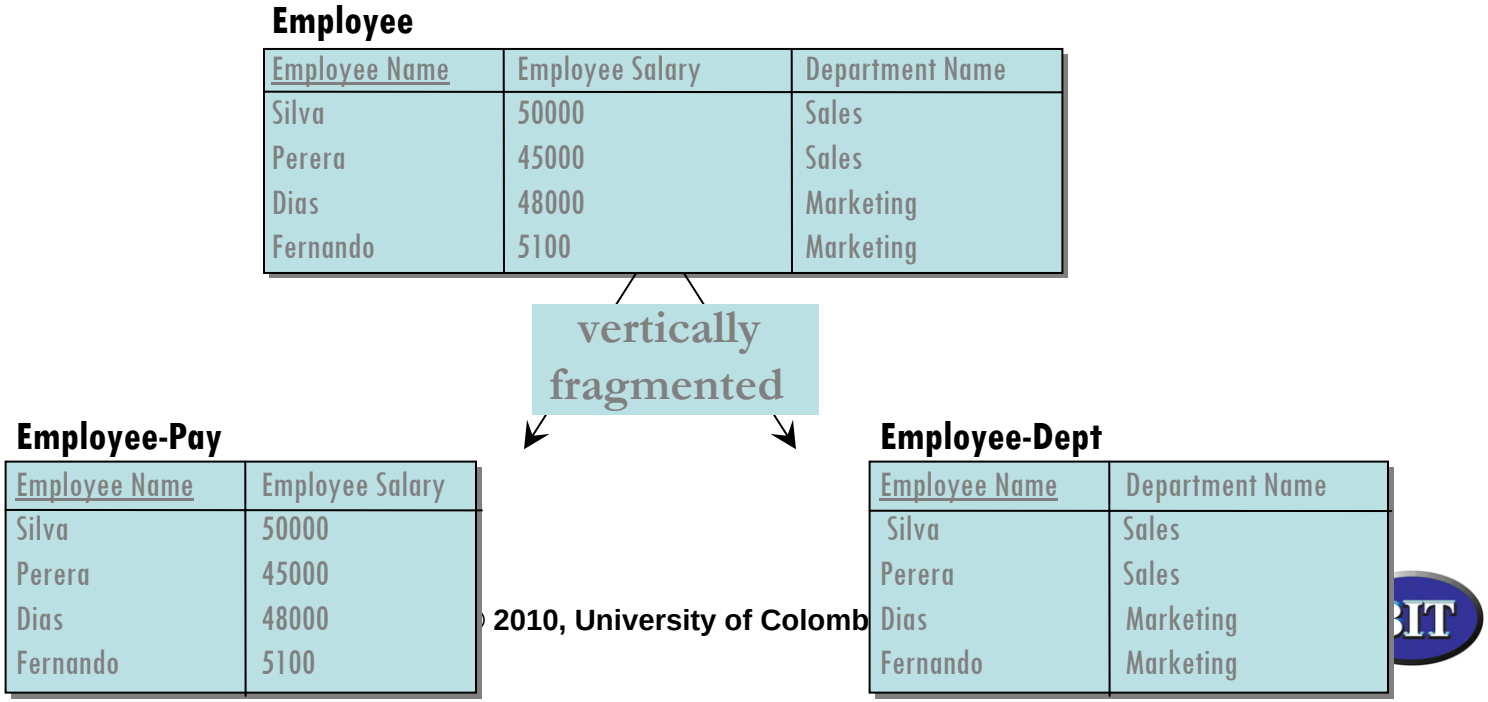
subsets of tuples (rows) from a relation (table).



Fragmentation

Vertical fragmentation subdivides the attributes of the global table into groups. A vertical fragment keeps only certain attributes of the global relation. The reconstruction is done by taking the join of all fragments using a common key.

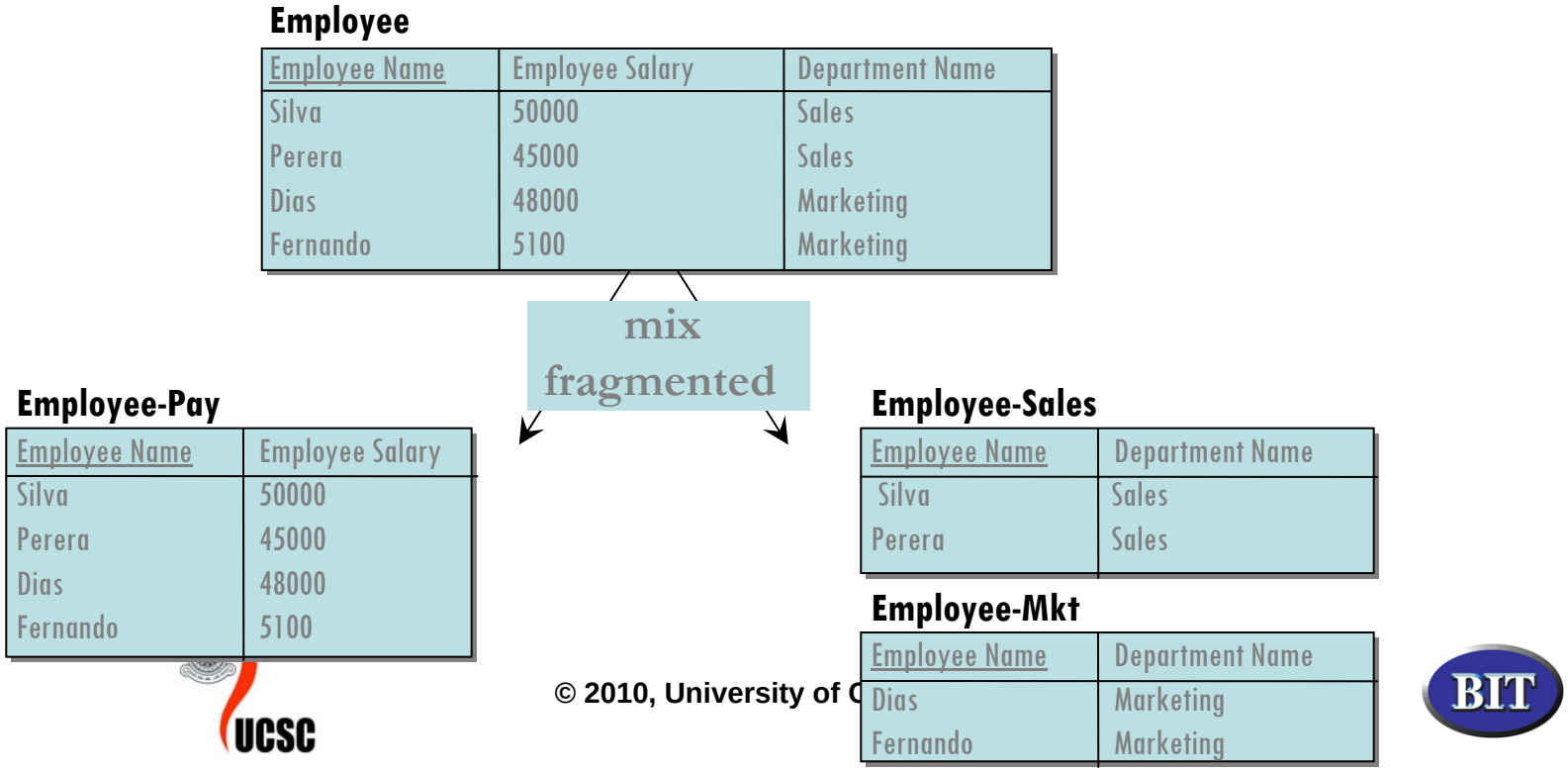
subsets of attributes (columns) from a relation (table).



Fragmentation

Mixed fragmentation is the result of the successive application of both fragmentation techniques.

a fragment which is both horizontally and vertically fragmented



Fragmentation

EMPLOYEE

EmpNo	Name	Salary	DNO
10	Perera	8000	SAL
12	De Silva	7500	MKT
22	Alwis	13000	MKT
25	Silva	12000	SAL
27	Dias	15000	MKT
30	Fernando	10000	SAL

SAL-EMP

EmpNo	Name	Salary	Dept
10	Perera	8000	SAL
25	Silva	12000	SAL
30	Fernando	10000	SAL

MKT-EMP

EmpNo	Name	Salary	DNO
12	De Silva	7500	MKT
22	Alwis	13000	MKT
27	Dias	15000	MKT

Fragmentation

EMPLOYEE

EmpNo	Name	Salary	DNO
10	Perera	8000	SAL
12	De Silva	7500	MKT
22	Alwis	13000	MKT
25	Silva	12000	SAL
27	Dias	15000	MKT
30	Fernando	10000	SAL

PER-EMP

EmpNo	Name	DNO
10	Perera	SAL
12	De Silva	MKT
22	Alwis	MKT
25	Silva	SAL
27	Dias	MKT
30	Fernando	SAL

PAY-EMP

EmpNo	Salary
10	8000
12	7500
22	13000
25	12000
27	15000
30	10000

Criteria

Criteria for Fragment Definition

Completeness Condition

All the data of the global relations must be mapped into the fragments.

Reconstruction Conditions

It must always be possible to reconstruct each global relation from its fragments

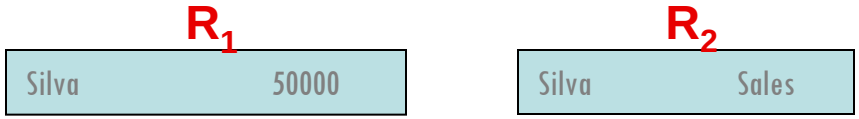
Degree of Fragmentation

- Finding the suitable level of partitioning within this range
- finite number of alternatives
  - Tuples or Attributes
  - Relations

Correctness rules of fragmentation

- Completeness

  If a relation instance R is decomposed into fragments $R_1, R_2 \ldots R_n$, each data item that can be found in R can also be found in one or more of R_i's.



- Reconstruction

  If a relation R is decomposed into fragments $R_1, R_2 \ldots R_n$, it should be possible to define a relational operator such that

  $R = \blacktriangledown R_i, \forall R_i \varepsilon F_R$,

  Please note the operator would be different for the different forms of fragmentation

$$R = R_1 \bowtie R_2$$

- Disjointness

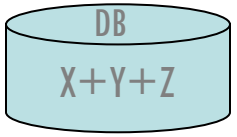
  If a relation R is horizontally decomposed into fragments $R_1, R_2 \ldots R_n$, and data item d_i is in R_j, it is not in any other fragment Rk (k != j).

Data Allocation

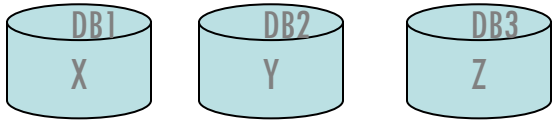
strategy is determined by the system architecture and network.

Four basic approaches:

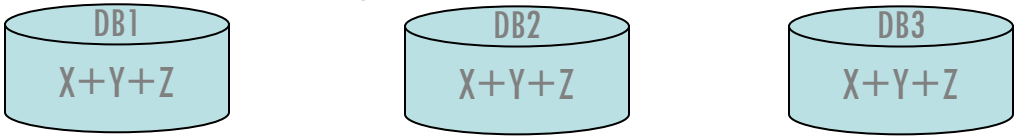
Centralised: all the data is located at a single site



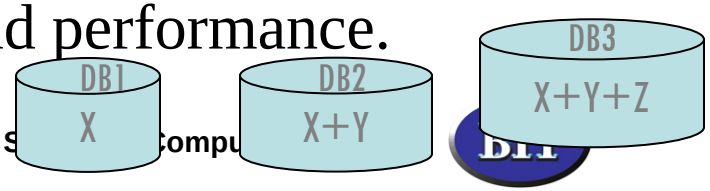
Partitioned: database is partitioned into disjoint fragments and each fragment is assigned to a particular site



Replicated: allocate a full copy of the database to each site

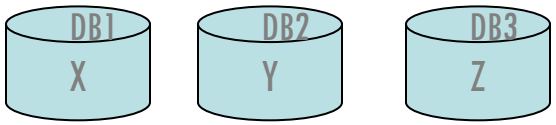


Selective Replication: partitioned data into critical and non-critical fragments and replicate the critical fragments to achieve the required level of availability and performance.



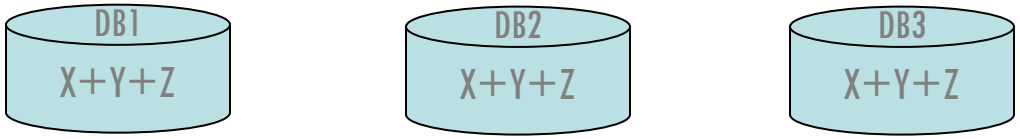
Allocation Alternatives

- Non-replicated

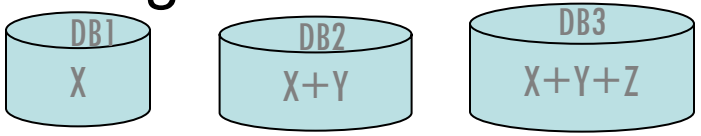


  - partitioned : each fragment resides at only one site

- Replicated



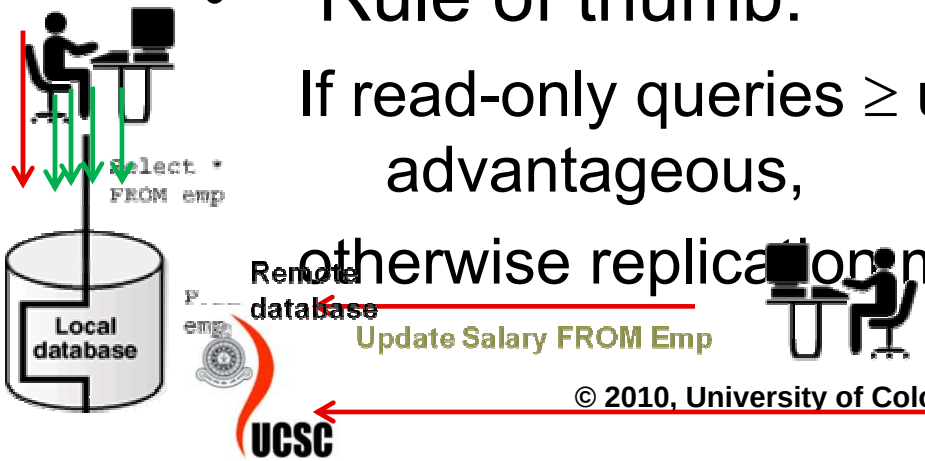
  - fully replicated : each fragment at each site
  - partially replicated : each fragment at some of the sites



- Rule of thumb:

If read-only queries ≥ update queries replication is advantageous,

otherwise replication may cause problems



Comparison of Replication Alternatives

	Full Replication	Partial Replication	Partitioning
Query Processing	Easy	Same ⟷ Difficulty	
Directory Management	Easy or nonexistent	Same ⟷ Difficulty	
Concurrency Control	Moderate	Difficult	Easy
Reliability	Very High	High	Low
Reality	Possible Application	Realistic	Possible application

Information Requirements

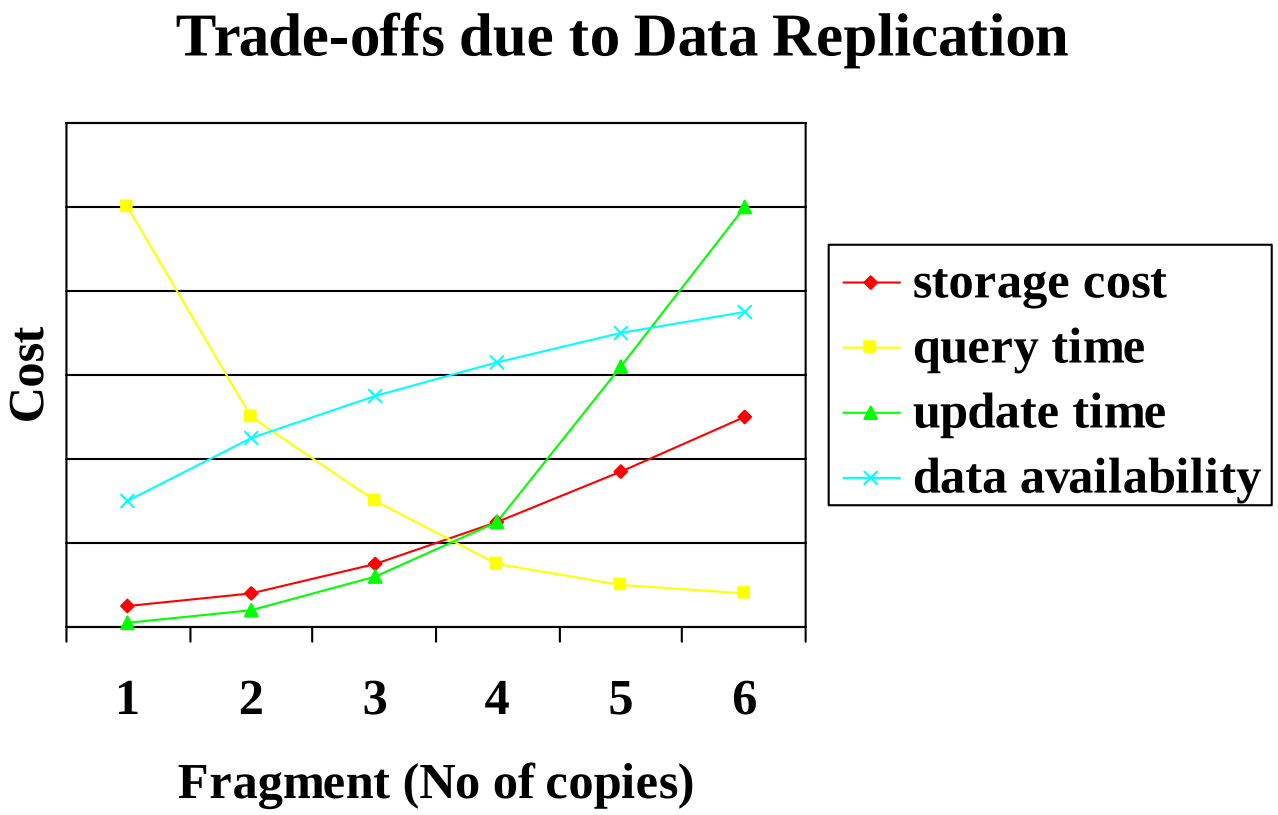
- Database information
  - selectivity of fragments
  - size of a fragment
- Application information
  - access types and numbers
  - access localities
- Communication network information
  - unit cost of storing data at a site
  - unit cost of processing at a site
- Computer system information
  - bandwidth
  - latency
  - communication overhead

Cost Benefit

Cost/benefit of the replicated database allocation strategy can be estimated in terms of storage cost, communication cost (query and update time) and data availability.

An optimal data allocation can be theoretically determined to minimise the total cost (storage+communication+local processing) subject to some response time and availability constraints.

Cost Benefit



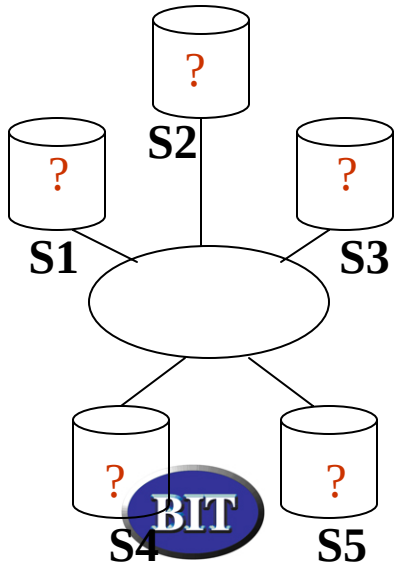
Strategies to Distribute Data

- Fragmentation
- Allocation

Employee

Employee Name	Employee Salary	Department Name
Silva	50000	Sales
Perera	45000	Sales
Dias	48000	Marketing
Fernando	51000	Marketing

- Distribution?
- Access?
- Query Processing?
- Security?



Types of Distribution Schemes

- **Round robin**

  creates even data distribution by randomly placing rows in fragments

  – The relation is scanned in any order and the i th tuple is sent to disk numbered D (i mod n). Ensure even distribution of tuples across disks.

Consider to use **round robin** when your queries perform sequential scans and you have little information about the data being stored. Also useful when your application is update intensive or when fast data loading is important.

- **Round robin**

  Advantages

  – no knowledge of the data is needed to achieve an even distribution among the fragments.

  – When column values are updated, rows are not moved to other fragments because the distribution does not depend on column values.

  Disadvantage

  – query optimiser is not able to eliminate fragments when evaluating a query.

CREATE TABLE table1 (col1 char(5), …)

FRAGMENT BY ROUND ROBIN IN dbspace1, dbspace2;

Distribution Schemes contd.

- **Expression based**

  puts related rows in the same fragment, may creates uneven distribution of data

  Logical and Relational Operators

  – Distributes contiguous attribute-value ranges to each disk. If value range is >= v(i) and < v(i+1) then place on disk i+1.

CREATE TABLE table1 (col1 char(5), …) **FRAGMENT BY EXPRESSION** col1>=1 and col1<=10 IN dbspace1, col1>10 and col1<=20 IN dbspace2, REMAINDER IN dbspace3;

CREATE TABLE table1 (col1 char(5), …) **FRAGMENT BY EXPRESSION** col1 IN (1000, 6000, 8500) IN dbspace1, col1>10 and col1<=20 IN dbspace2, REMAINDER IN dbspace3;

Advantages

- fragments may be eliminated from query scans.
- data can be segmented to support a particular archiving strategy.
- users can be granted privileges at the fragment level.
- unequal data distribution can be created to offset an unequal frequency of access.

Disadvantages

- CPU resources are required for rule evaluation. When rule is complex more CPU time is consumed.
- finding the optimum rule may be an iterative process and once found may need to be monitored.
- more administrative work than the round robin.

Consider using an **expression strategy** when:

- non-overlapping fragments on a single column can be created.
- The table is accessed with a high degree of selectivity.
- The data access is not evenly distributed.
- Overlapping fragments on single or multiple columns can be created.

Distribution Schemes contd.

- **Expression based**

  a hash function can be used to evenly distribute data across fragments, especially when the column value may not divide commonly accessed data evenly across fragments.

  Hash Functions

  – Maps each tuple to a disk location based on a hash function, whose range is (0, 1, …, n-1). If hash function returns i, then place tuple on disk i.

Advantage

- a hash expression yields an even distribution of data.
- When there is an unequal search it permits fragment elimination during query optimisation.

Disadvantage

- fragment elimination does not occur during a range search.

CREATE TABLE table1 (col1 char(5), …)

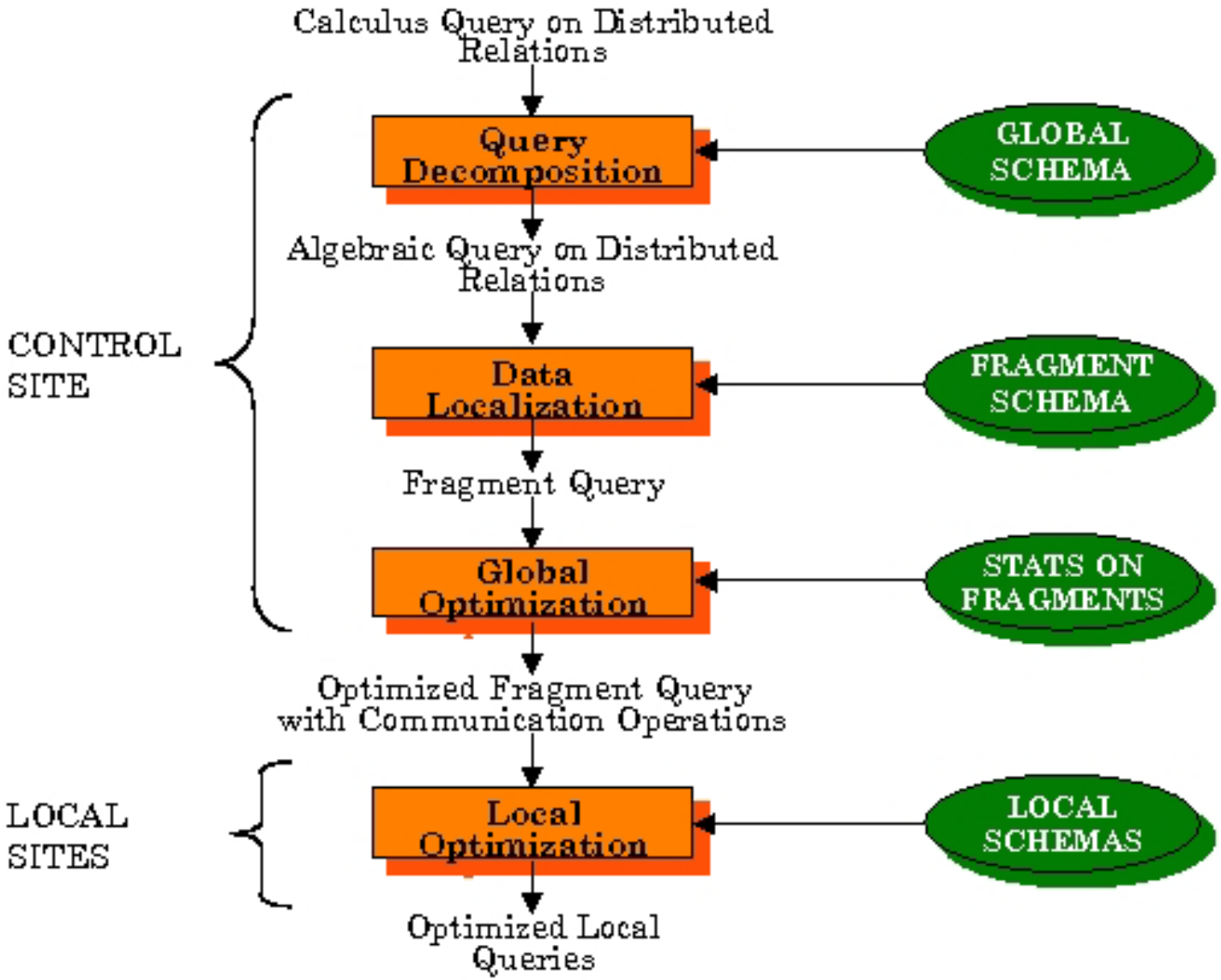
FRAGMENT BY EXPRESSION

MOD(col1,3) = 0 IN dbspace1, MOD(col1,3) = 1 IN dbspace2, MOD(col1,3) = 2 IN dbspace3;

Guidelines

- avoid REMAINDER IN clause as fragment is always checked
- distribute data so that I/O is balanced across disks. Not necessarily means even distribution.
- Keep fragment expressions simple. Complex expressions takes more CPU time to evaluate. Avoid any expression that must perform a conversion.
- Optimised data loads by placing the most frequently accessed fragment first in your fragment statement. This reduces the number of fragments to be checked.
- If a significant benefit is not expected, do not fragment the table.

Distributed Query Processing Methodology



Distributed QP

- Problem of query processing can itself be decomposed into several sub-problems corresponding to various layers.
- First two correspond to query rewriting.
- First three layers are performed by a central site using global information.
- Fourth is done by the local site.

Step 1 – Query Decomposition

Use techniques of a centralise DBMS on global relations. Calculus query is rewritten in a normalised form suitable for subsequent manipulation.

- Input : Calculus query on global relations
- Normalisation
  - manipulate query quantifiers and qualification by applying logical operator priority
- Analysis
  - detect and reject “incorrect” queries
  - possible for only a subset of relational calculus
- Simplification
  - eliminate redundant predicates
- Restructuring
  - calculus query ⇒ algebraic query
  - more than one translation is possible
  - use transformation rules

Normalisation

- 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

    $(p11 \vee p12 \vee \ldots \vee p1n) \wedge \ldots \wedge (pm1 \vee pm2 \vee \ldots \vee pmn)$
  - Disjunctive normal form

    $(p11 \wedge p12 \wedge \ldots \wedge p1n) \vee \ldots \vee (pm1 \wedge pm2 \wedge \ldots \wedge pmn)$
  - 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

Analysis – Example

SELECT ename,mgr

FROM Employee E, WorksOn W, Project P

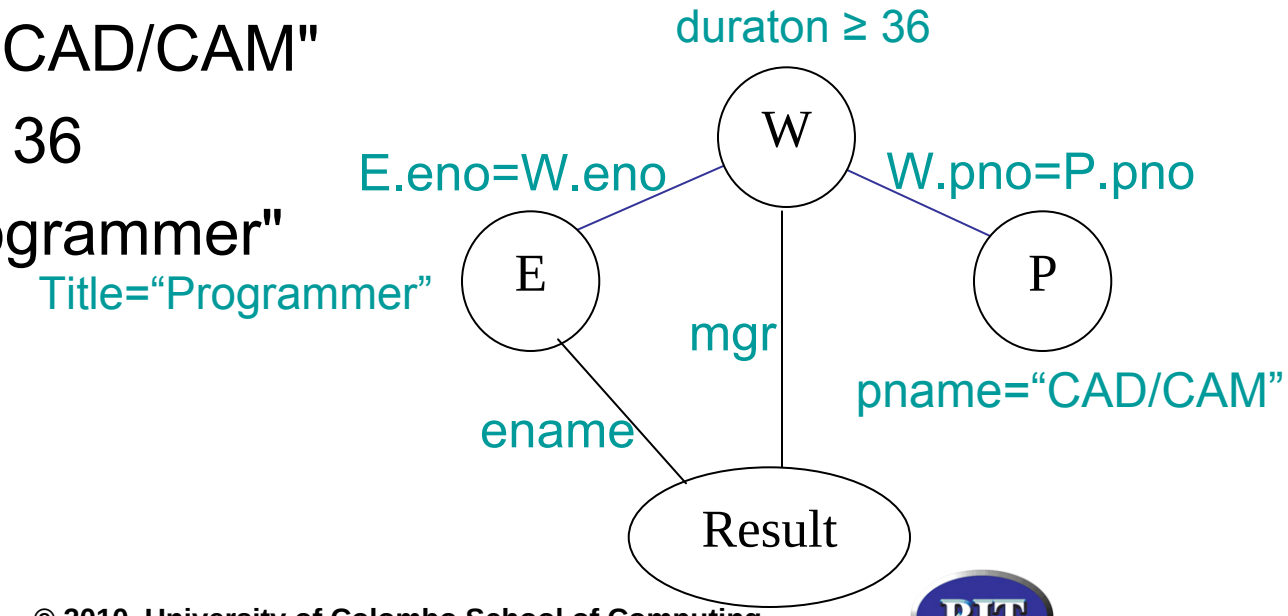
WHERE E.eno = W.eno

AND W.pno = P.pno

AND pname = "CAD/CAM"

AND duration ≥ 36

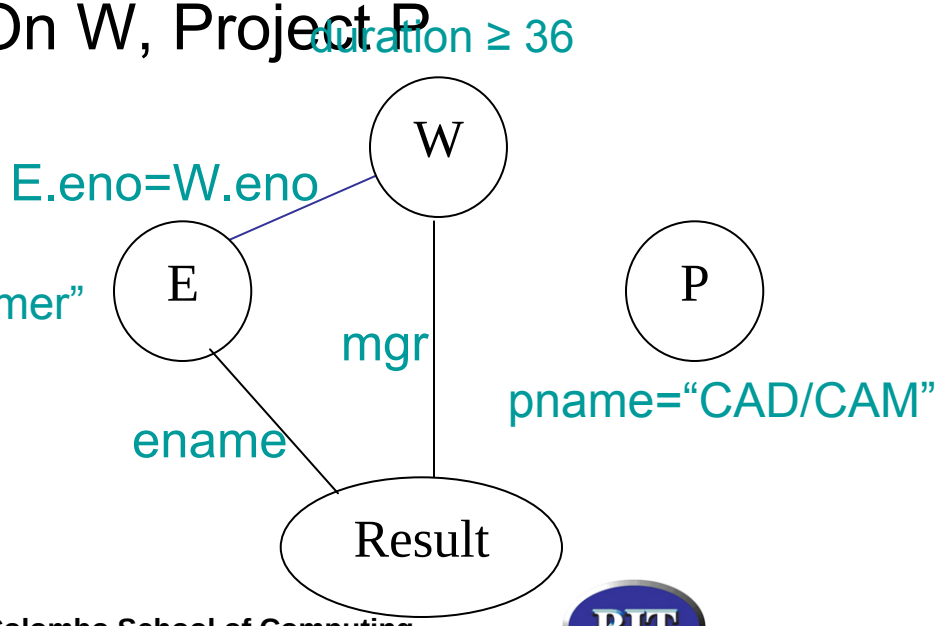
AND title = "Programmer"



Analysis

- If the query graph is not connected, the query is wrong.

SELECT ename,mgr
FROM Employee E, WorksOn W, Project P
WHERE E.eno = W.eno
AND pname = "CAD/CAM"
AND duration ≥ 36
AND title = "Programmer"

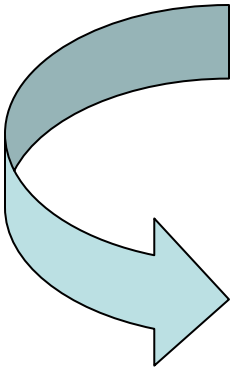


Simplification

- Why simplify?
  - Simple query vs Complex
- How? Use transformation rules
  - elimination of redundancy
  - Idem potency rules
    - $p1 \wedge \neg(p1) \Leftrightarrow$ false
    - $p1 \wedge (p1 \vee p2) \Leftrightarrow p1$
    - $p1 \vee$ false $\Leftrightarrow p1$
    - …
  - application of transitivity
  - use of integrity rules

Simplification – Example

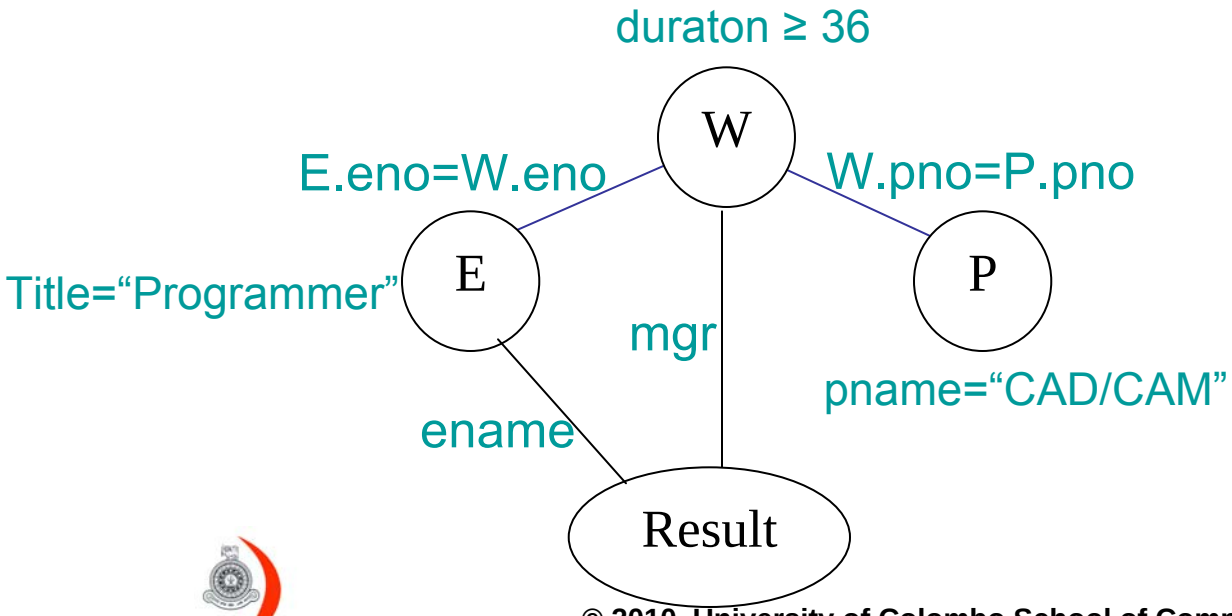
SELECT Title
FROM Employee
WHERE ENAME = “Perera”
OR (**NOT** (Title = “Programmer”)
AND (Title = “Programmer”
OR Title = “Elect. Eng.”)
AND NOT (Title = “Elect. Eng.”))

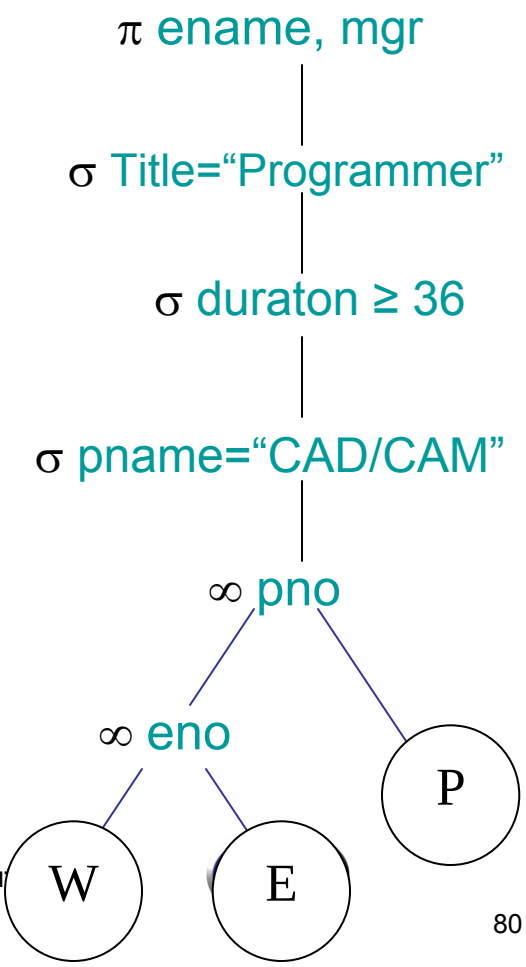


SELECT Title
FROM Employee
WHERE ENAME = “Perera”

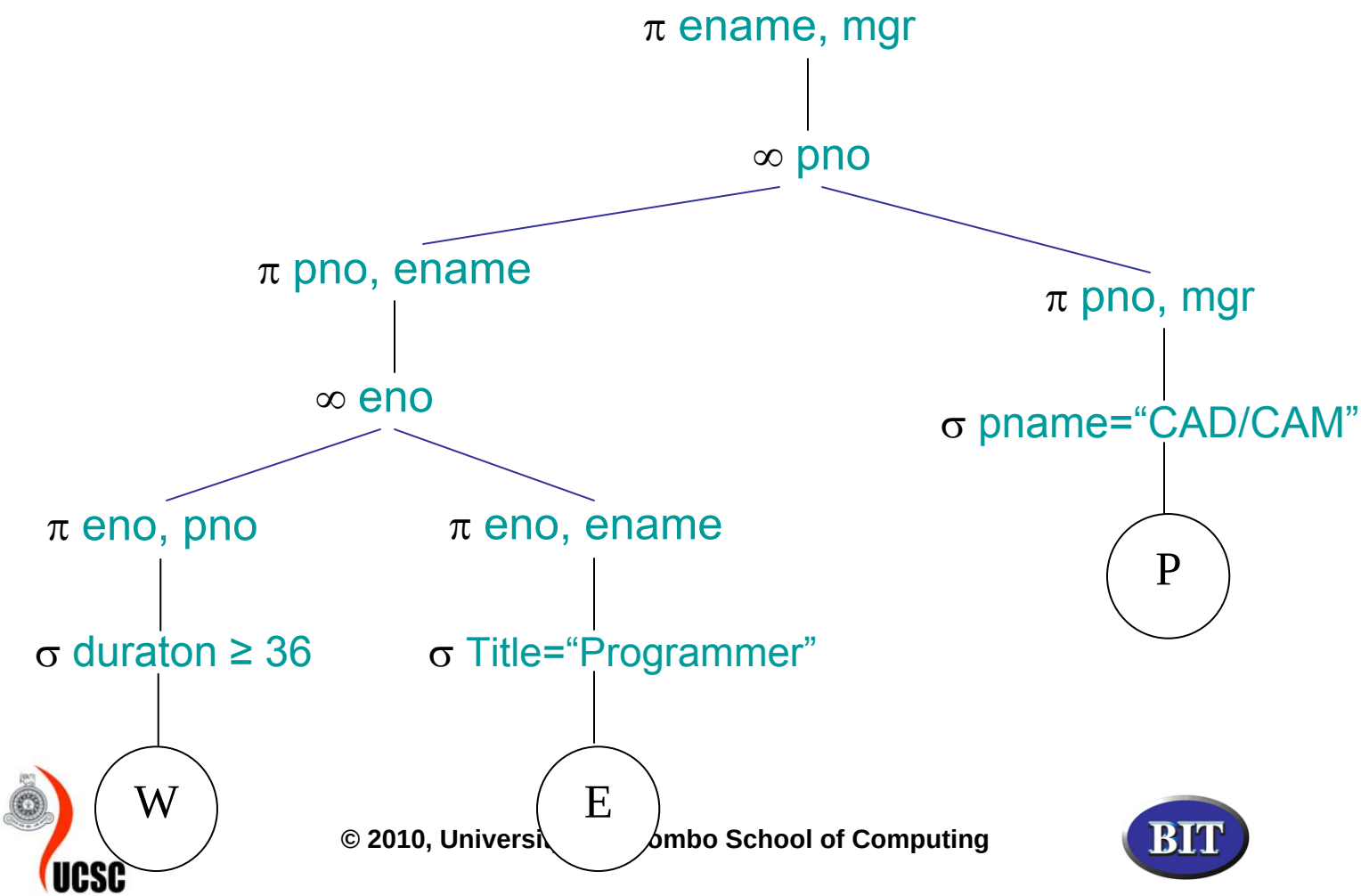
Restructuring

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

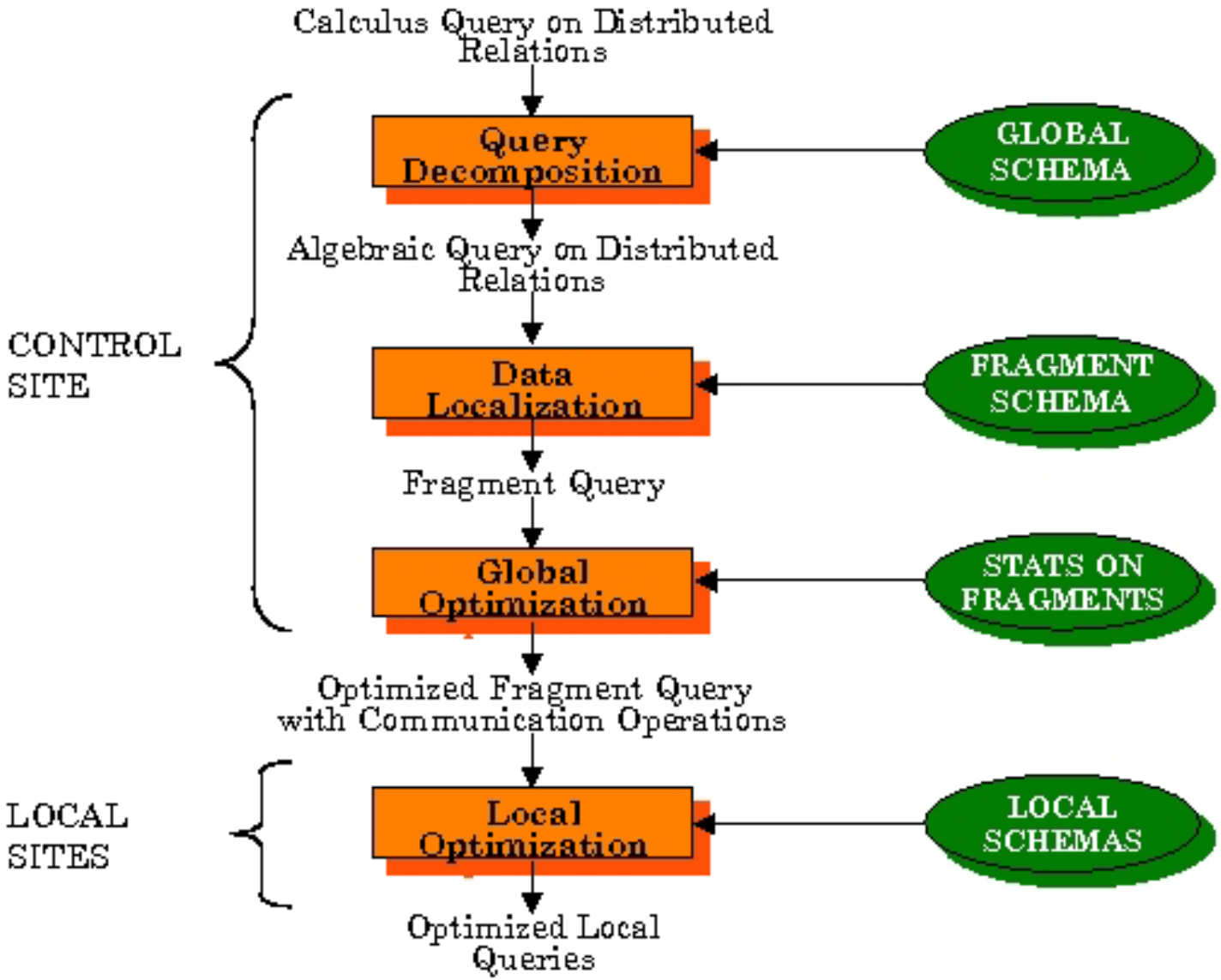




Restructuring



Distributed Query Processing Methodology



Step 2 – Data Localisation

Input: Algebraic query on distributed relations

- Determine which fragments are involved
- Localisation program
  - substitute for each global query its materialisation program
  - optimise

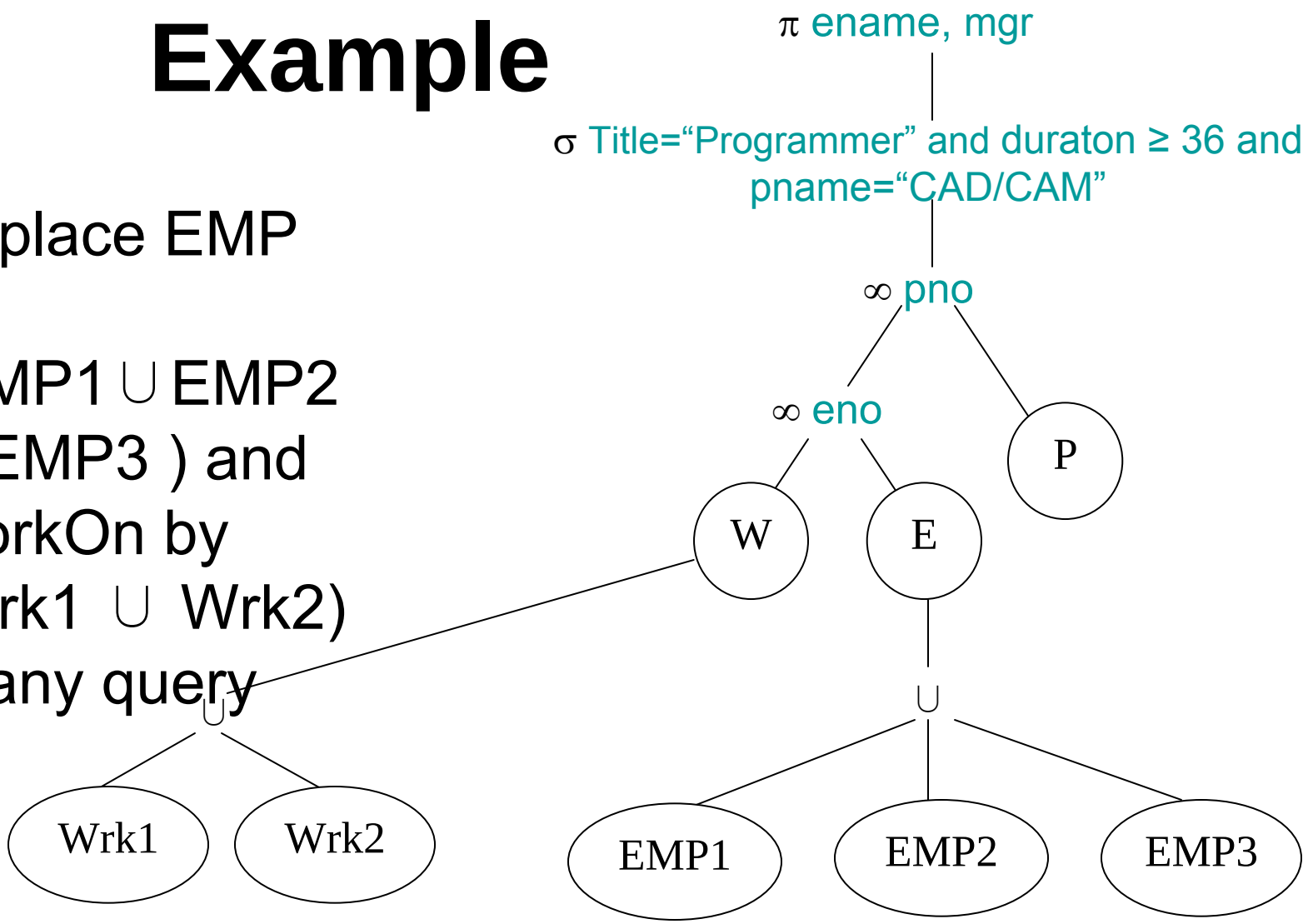
Example

Assume

- Employee is fragmented into EMP1,EMP2, EMP3 as follows:
  - EMP1 = $\sigma_{eno \leq \text{“E3”}}$(Employee)
  - EMP2 = $\sigma_{\text{“E3”} < eno \leq \text{“E6”}}$(Employee)
  - EMP3 = $\sigma_{eno \geq \text{“E6”}}$(Employee)
- WorkOn fragmented into Wrk1 and Wrk2 as follows:
  - Wrk1 = $\sigma_{eno \leq \text{“E3”}}$(WorkOn)
  - Wrk2 = $\sigma_{eno > \text{“E3”}}$(WorkOn)

Example

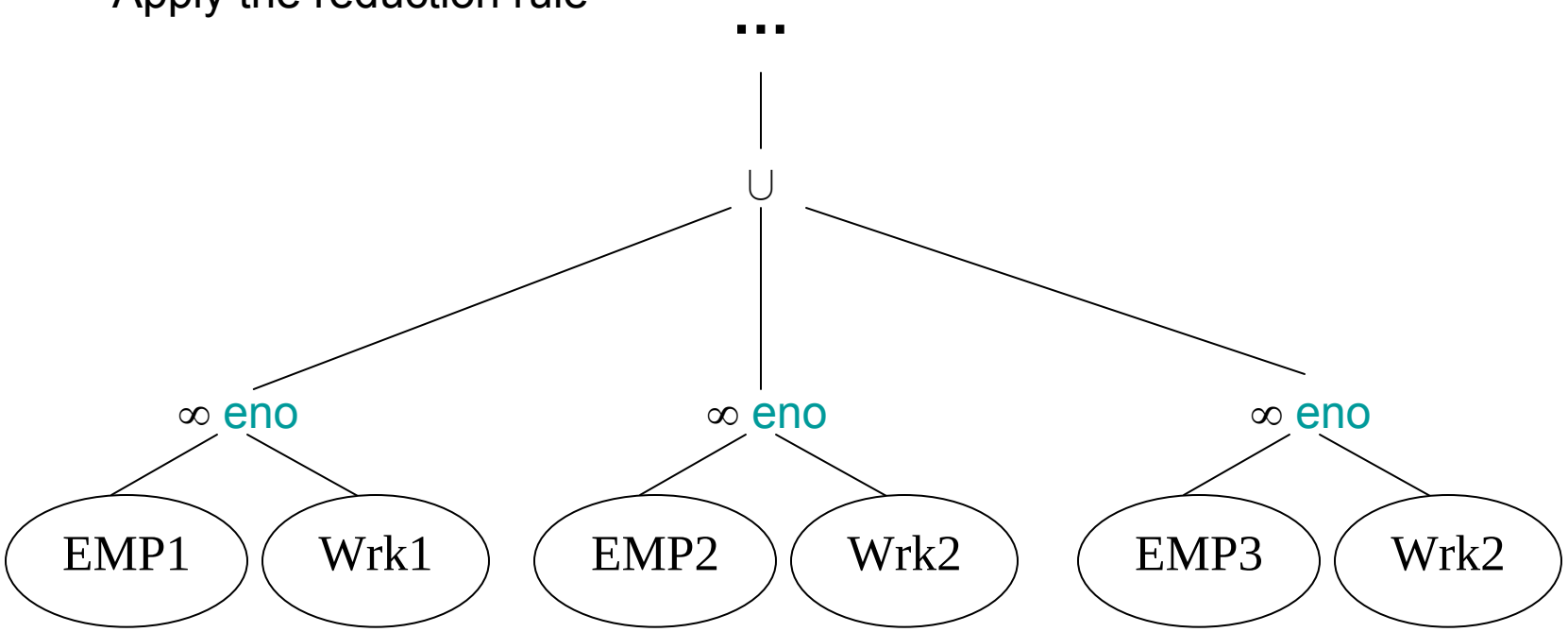
Replace EMP by (EMP1 ∪ EMP2 ∪ EMP3) and WorkOn by (Wrk1 ∪ Wrk2) in any query



Provides Parallelism
Eliminate unnecessary work

Reduction with join

- Distribute join over unions
- Apply the reduction rule



Step 3 – Global Query Optimisation

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 semi-joins
    - Semi-join saves on communication at the expense of more local processing.
  - Join methods
    - nested loop vs ordered joins (merge join or hash join)

Cost-Based Optimisation

- 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 maximise throughput
- Search algorithm
  - How do we move inside the solution space?
  - Exhaustive search, heuristic algorithms (iterative improvement, simulated annealing, genetic,…)

Search Space

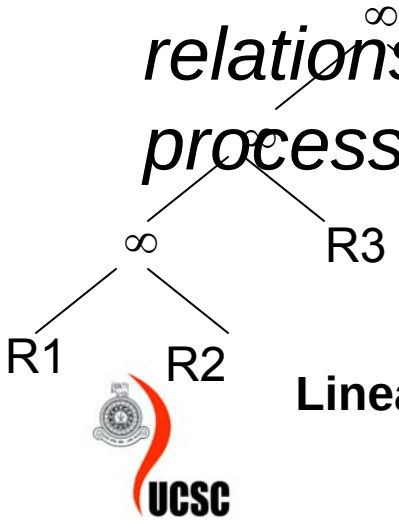
- Search space characterised by alternative execution plans
- Focus on join trees
- For *N* relations, there are O(*N*!) equivalent join trees that can be obtained by applying commutative and associative rules

SELECT ename,mgr
FROM Employee E, WorksOn W, Project P
WHERE E.eno = W.eno
AND W.pno = P.pno

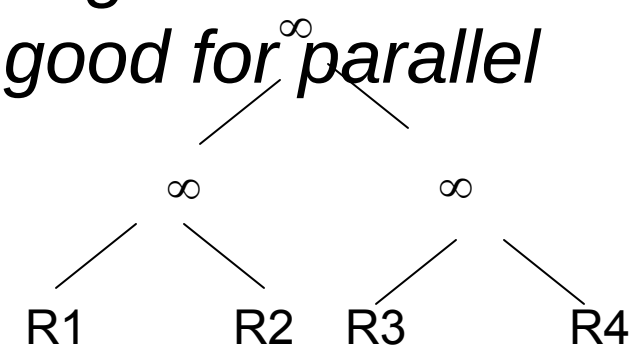
Search Space

- 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

Liner use at least one base relation at a time while bushy ends up using intermediate relations. But bushy is good for parallel processing.



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 (build all possible plans and chose best)
  - Greedy: depth-first (build only one-plan)
- Randomised
  - Search for optimality around a particular starting point
  - Trade optimisation time for execution time
  - Better when > 5-6 relations
  - Iterative improvement

Distributed Query Optimisation Problems

- Cost model
  - multiple query optimisation
  - heuristics to cut down on alternatives
- Larger set of queries
  - optimisation only on select-project-join queries
  - also need to handle complex queries (e.g., unions, disjunctions, aggregations and sorting)
- Optimisation cost vs execution cost trade-off
  - heuristics to cut down on alternatives
  - controllable search strategies
- Optimisation/re-optimisation interval
  - extent of changes in database profile before re-optimisation is necessary