Outline

- Introduction
- Background
- Distributed DBMS Architecture
  - Transparencies
  - Datalogical Architecture
  - Component Architecture
- Distributed Database Design
- Semantic Data Control
- Distributed Query Processing
- Distributed Transaction Management
- Distributed Database Operating Systems
- Open Systems and Interoperability
- Parallel Database Systems
- Distributed Object Management
- Concluding Remarks

Architecture

- Defines the structure of the system
  - components identified
  - functions of each component defined
  - interrelationships and interactions between components defined

Transparency is ...

separation of the higher level semantics of a system from the lower level implementation issues.

Hide the implementation details from higher layers of the system and from the users.

Fundamental Transparency Issue

Provide

data independence

in the distributed environment

- Network (distribution) transparency
- Replication transparency
- Fragmentation transparency

Data Independence

- Immunity of the user applications to changes in the definition and/or organization of data and vice versa.
  - The only form available in centralized DBMS.
  - Logical data independence
    - Immunity of user applications to changes in the logical structure of the database.
    - Schema definition might change without affecting the user applications.
    - For example, addition of new attributes to a relation, the creation of new relations, logical reordering of attributes.
  - Physical data independence
    - Hiding the details of the storage structure from the user applications.
    - Physical data description might change without affecting the user applications.
    - For example, data might be moved from one disk volume to another; the organization of the data might change.

Network Transparency

- Existence of the network should not be noticed by user applications.
- Entails two things:
  - *Location transparency*
    - the command that is used is independent of the location of the data **and** the location where the operation is to be carried out.
    - Example : copy command in Unix (cp versus rcp)

      No location transparency
  - *Naming transparency*
    - providing a unique name for each object in the distributed system.
    - Do not embed location information into the names.

Replication Transparency

- If there are replicas of database objects, their existence should be controlled by the system not by the user.
- Tradeoff:
  - If the user is aware of the existence of replicas and is responsible for their management, then the system has to do minimal work and the performance might be better.
  - You might forget about moving data around.

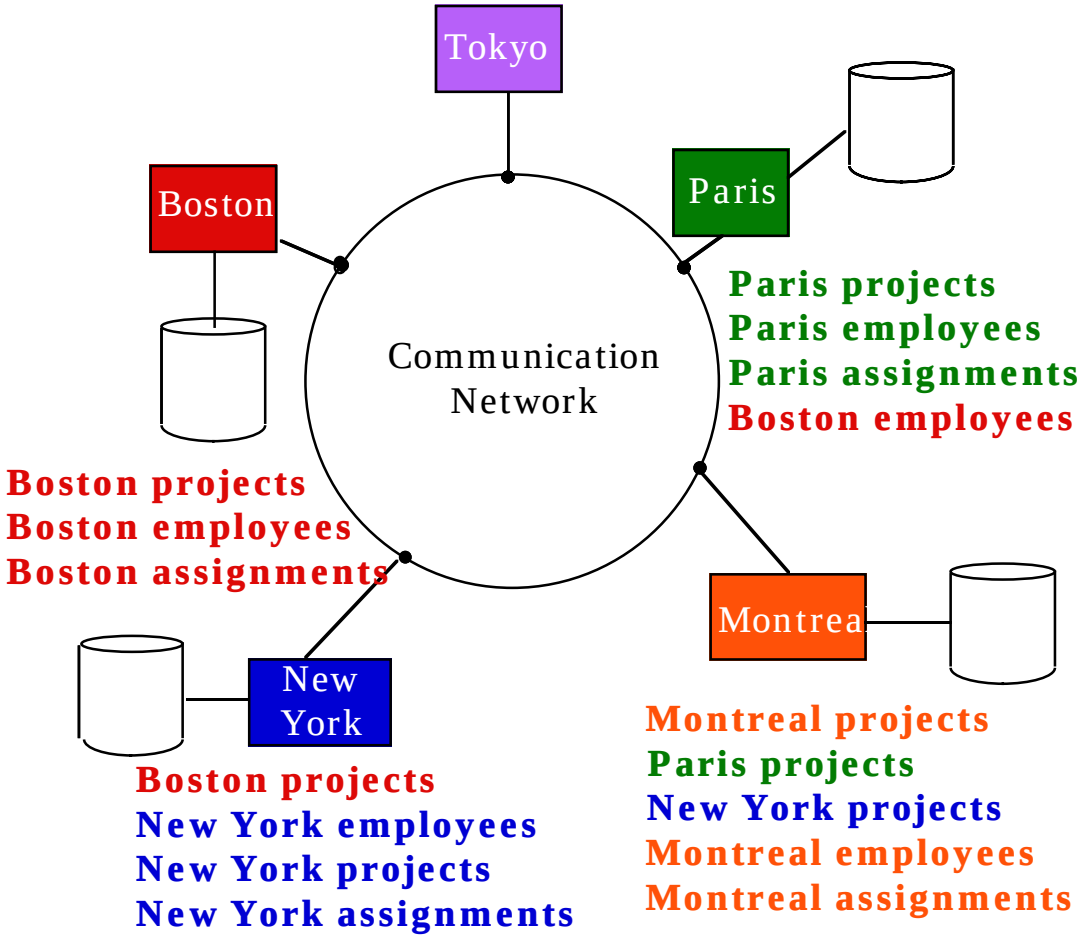
Fragmentation Transparency

- If database objects (relations) are fragmented, then the system has to handle the conversion of user queries defined on global relations to queries defined on fragments.

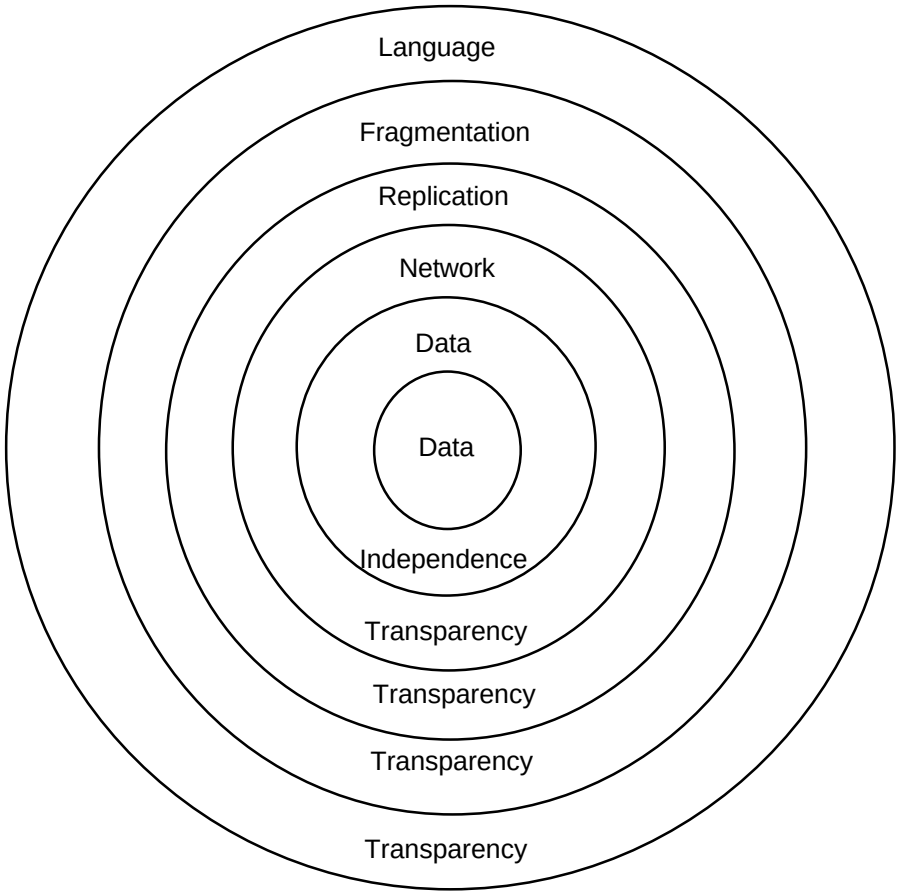
- Translation from *global queries* to *fragment queries* and putting together fragments into an answer.

Transparent Access

```
SELECT   ENAME,SAL
FROM     E,G,S
WHERE    DUR > 12
AND      E.ENO = G.ENO
AND      E.TITLE = S.TITLE
```



Layers of Transparency



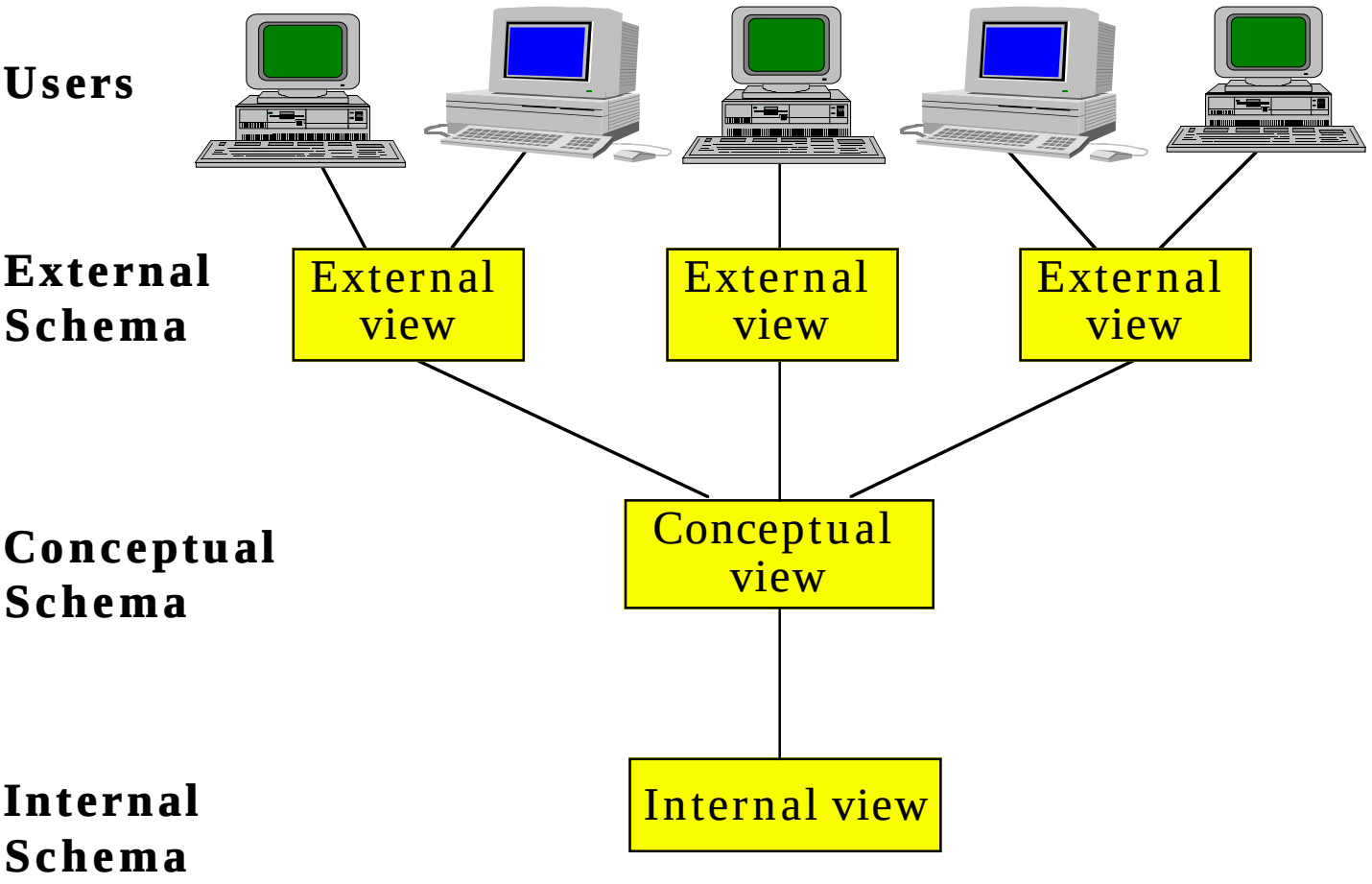
Who Is Responsible For Transparency?

- Language
- Operating system
- DBMS

} shared responsibility

- Distributed query processing concerns
- Distributed object naming concerns

ANSI/SPARC Architecture



Standardization

Reference Model

- A conceptual framework whose purpose is to divide standardization work into manageable pieces and to show at a general level how these pieces are related to one another.

Approaches

- **Component-based**
  - Components of the system are defined together with the interrelationships between components.
  - Good for design and implementation of the system.
- **Function-based**
  - Classes of users are identified together with the functionality that the system will provide for each class.
  - The objectives of the system are clearly identified. But how do you achieve these objectives?
- **Data-based**
  - Identify the different types of describing data and specify the functional units that will realize and/or use data according to these views.

Conceptual Schema Definition

RELATION E [
    KEY = {ENO}
    ATTRIBUTES = {
        ENO : **CHARACTER**(9)
        ENAME : **CHARACTER**(15)
        TITLE : **CHARACTER**(10)
    }
]
RELATION S [
    KEY = {TITLE}
    ATTRIBUTES = {
        TITLE : **CHARACTER**(10)
        SAL : **NUMERIC**(6)
    }
]

Conceptual Schema Definition

```
RELATION J [
    KEY = {JNO}
    ATTRIBUTES = {
        JNO     : CHARACTER(7)
        JNAME   : CHARACTER(20)
        BUDGET  : NUMERIC(7)
        LOC     : CHARACTER(15)
    }
]
RELATION G [
    KEY = {ENO,JNO}
    ATTRIBUTES = {
        ENO     : CHARACTER(9)
        JNO     : CHARACTER(7)
        RESP    : CHARACTER(10)
        DUR     : NUMERIC(3)
    }
]
```


Internal Schema Definition

```
RELATION E [
    KEY = {ENO}
    ATTRIBUTES = {
        ENO     : CHARACTER(9)
        ENAME   : CHARACTER(15)
        TITLE   : CHARACTER(10)
    }
]
```

```
INTERNAL_REL EMP [
    INDEX ON E# CALL EMINX
    FIELD = {
        E#      : BYTE(9)
        ENAME   : BYTE(15)
        TIT     : BYTE(10)
    }
]
```


External View Definition – Example 1

Create a BUDGET view from the PROJECT relation

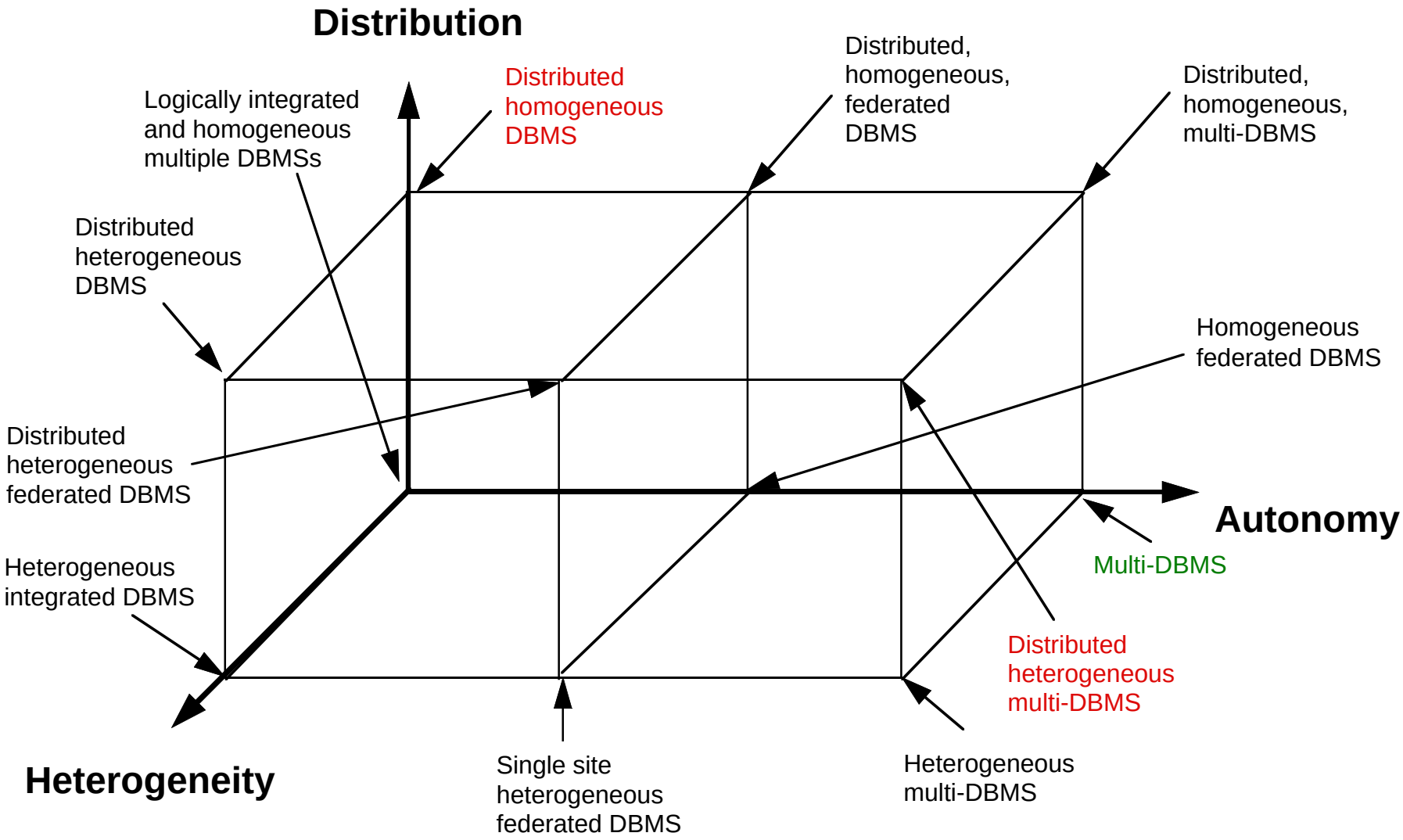
```
CREATE  VIEW    BUDGET(PNAME, BUD)
 AS     SELECT  JNAME, BUDGET
        FROM    PROJECT
```


External View Definition – Example 2

Create a Payroll view from relations EMPLOYEE and TITLE_SALARY

```
CREATE  VIEW    PAYROLL (EMP_NO, EMP_NAME, SAL)
AS      SELECT  E.ENO,E.ENAME,S.SAL
        FROM    E, S
        WHERE   E.TITLE = S.TITLE
```

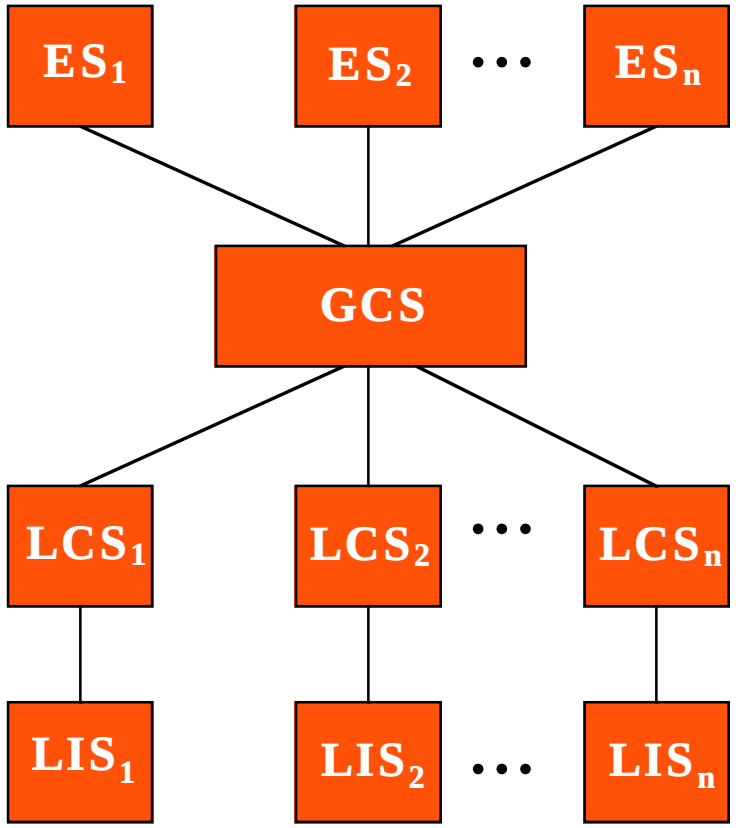

DBMS Implementation Alternatives



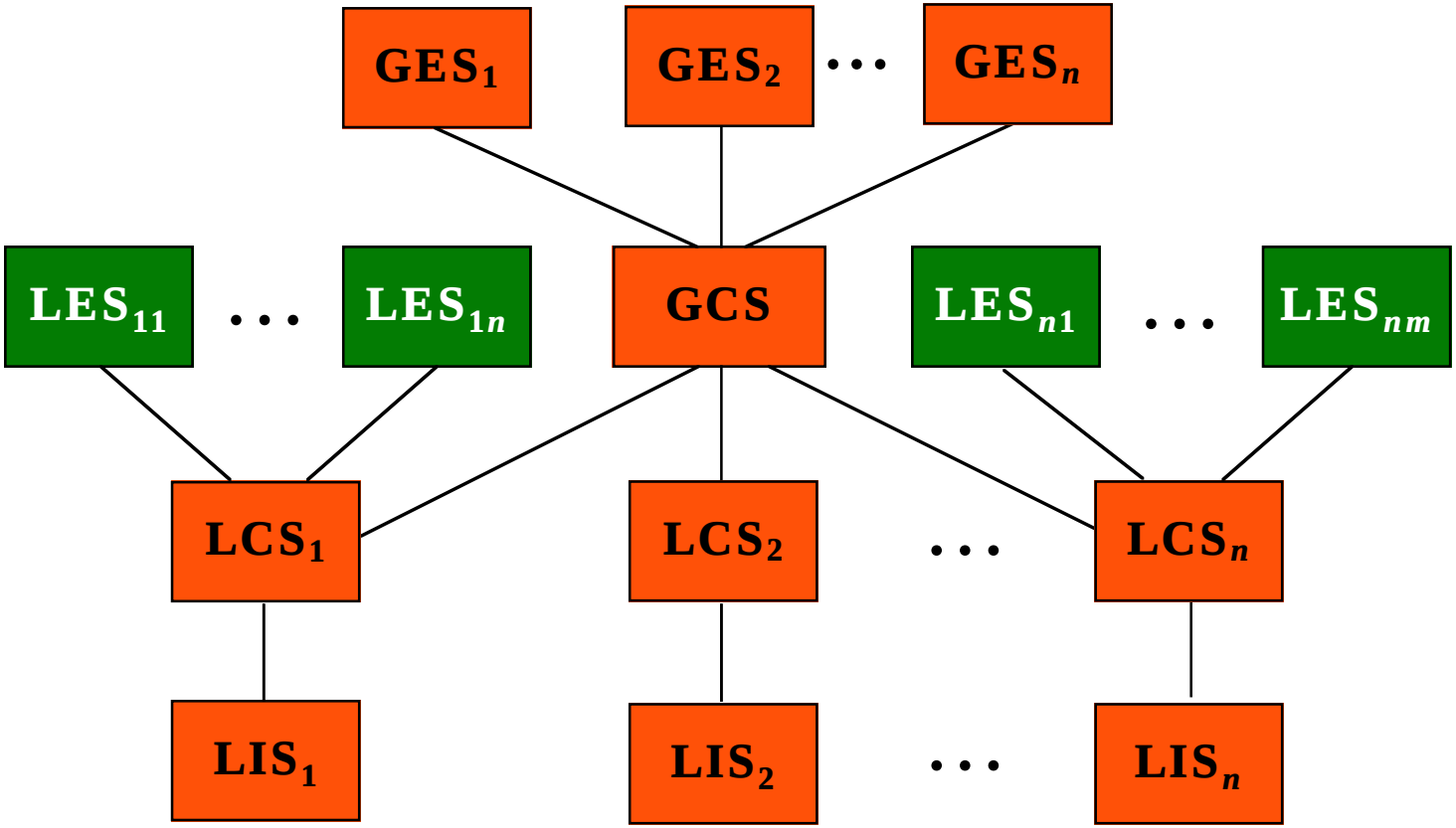
Dimensions of the Problem

- Distribution
  - Whether the components of the system are located on the same machine or not
- Heterogeneity
  - Various levels (hardware, communications, operating system)
  - DBMS important one
    - data model, query language,transaction management algorithms
- Autonomy
  - Not well understood and most troublesome
  - Various versions [Veijalainen and Pepescu-Zeletin, 1988]
    - Design autonomy: Ability of a component DBMS to decide on issues related to its own design.
    - Communication autonomy: Ability of a component DBMS to decide whether and how to communicate with other DBMSs.
    - Execution autonomy: Ability of a component DBMS to execute local operations in any manner it wants to.

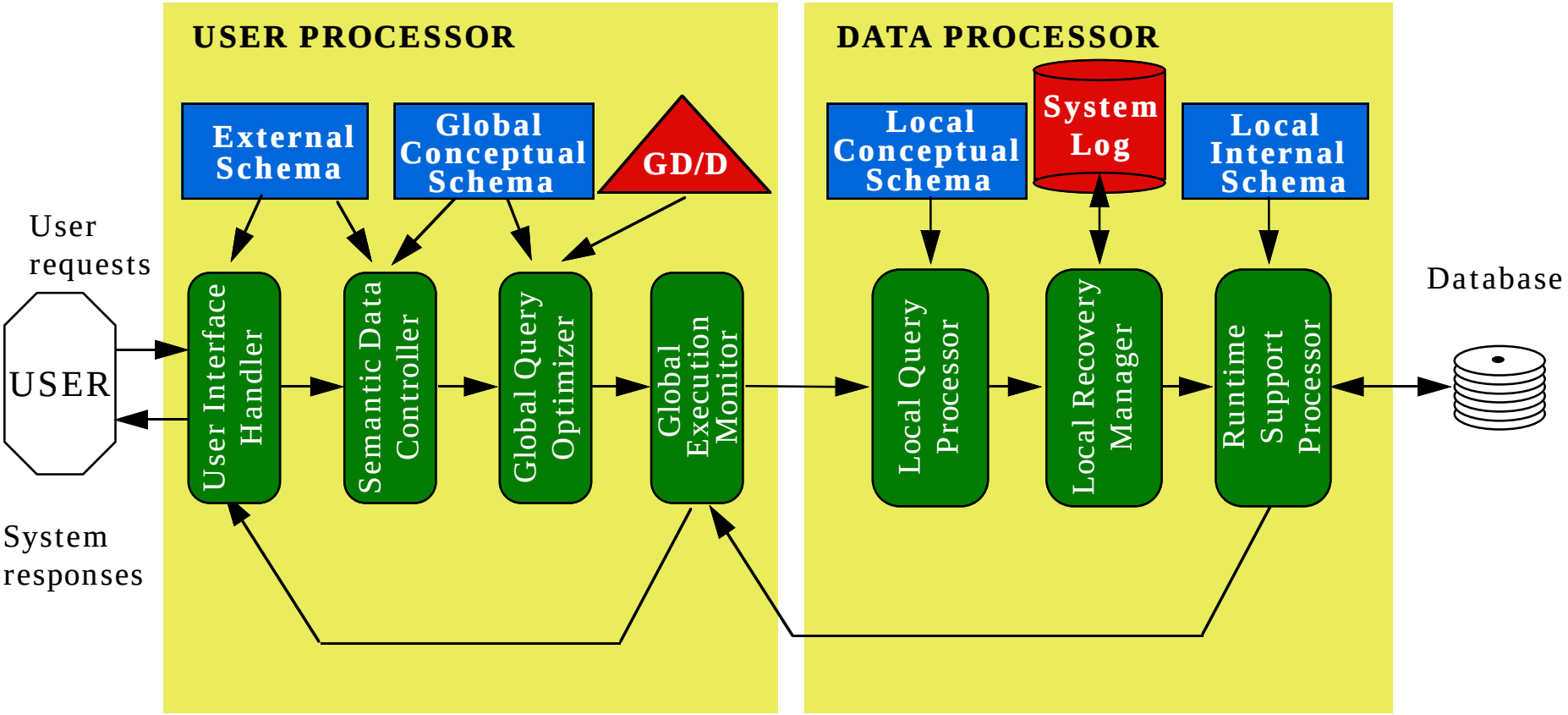
Datalogical Distributed DBMS Architecture



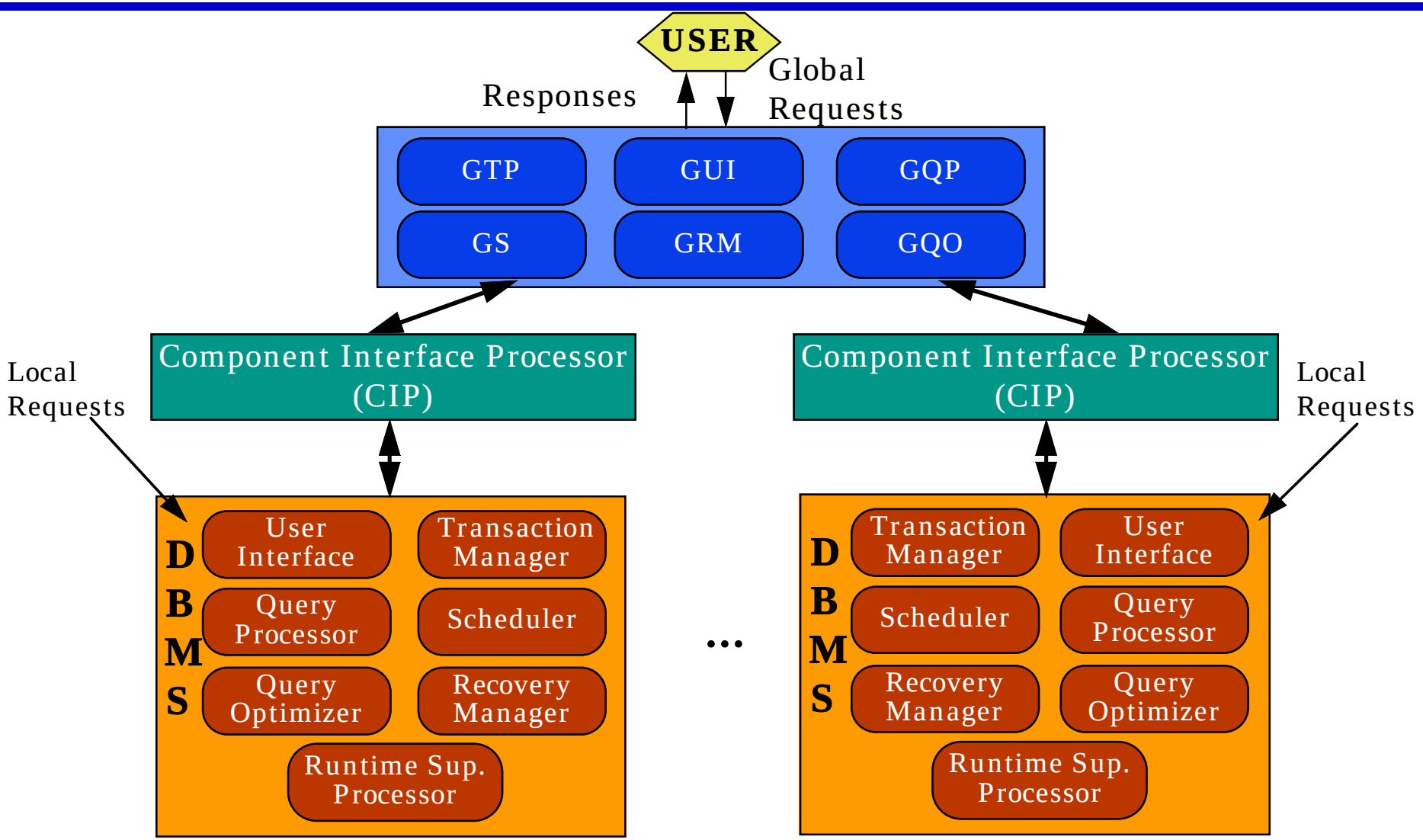
Datalogical Multi-DBMS Architecture



Distributed DBMS Component Architecture



Components of a Multi-DBMS



Directory Issues