BITS Pilani

BITS Pilani
Hyderabad Campus

Dr. Manik Gupta
Associate Professor
Department of CSIS

BITS Pilani
Hyderabad Campus

Distributed Data Systems (CS G544) Lecture 6-8

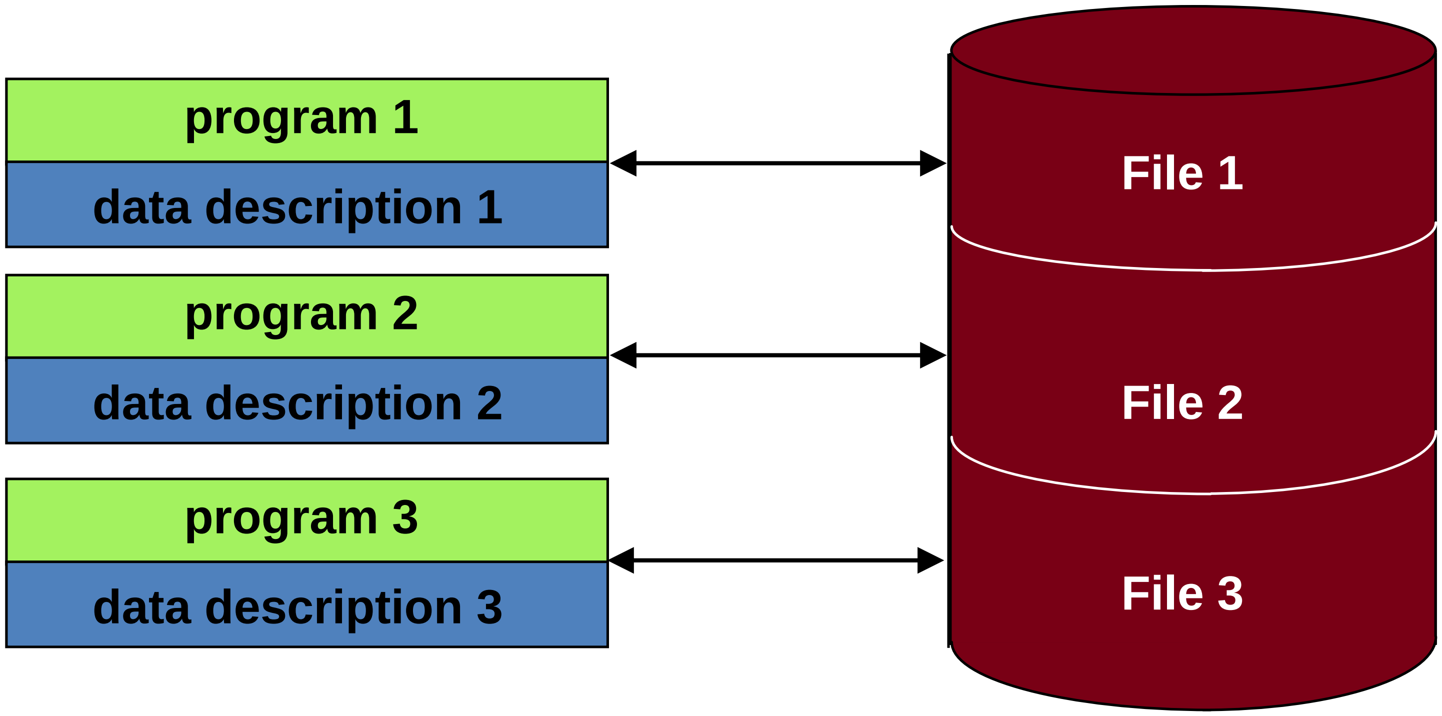
Thursday, 22nd August 2024

Lecture Recap

- Review of RDBMS

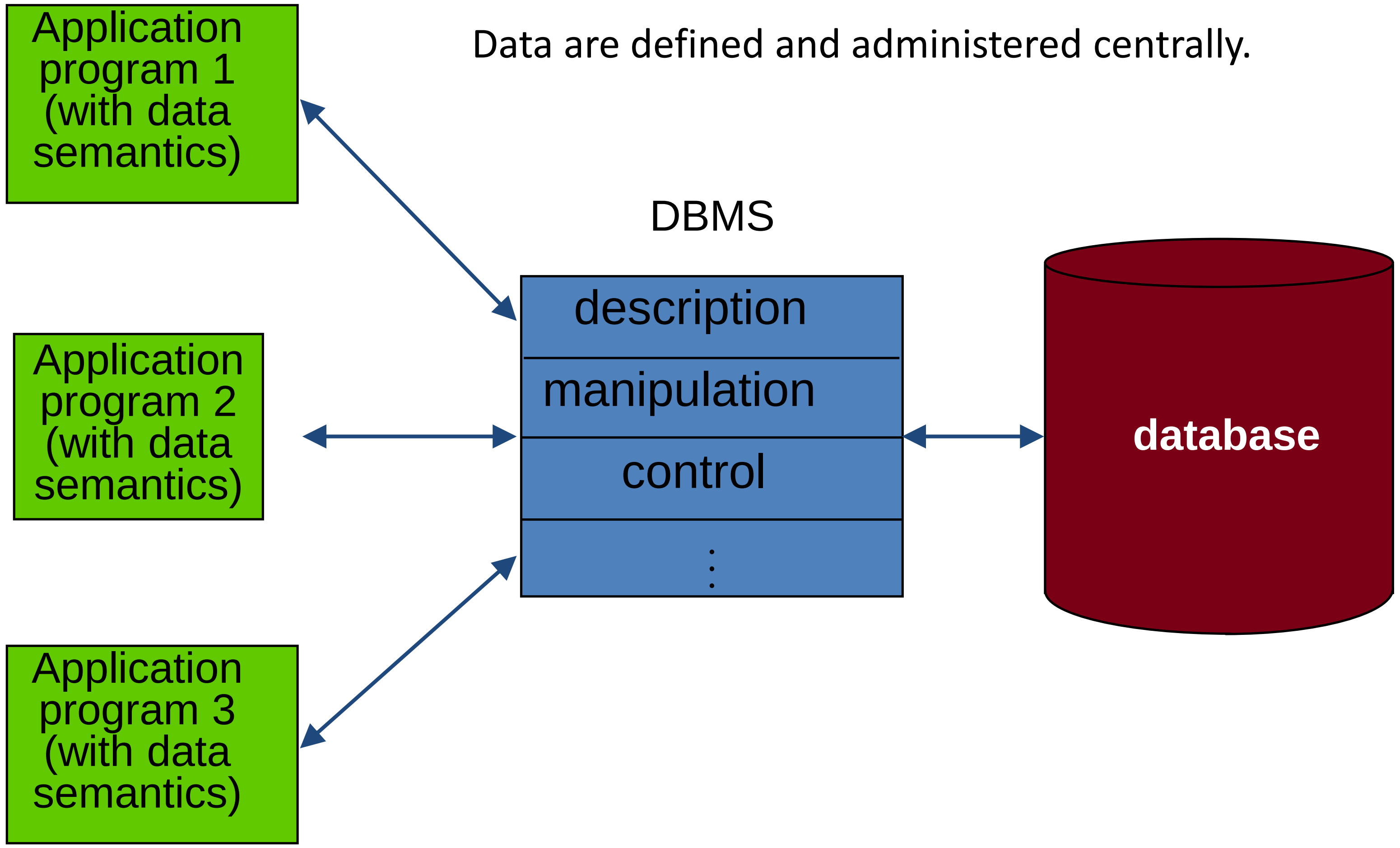
Traditional file processing

Each application defined and maintained its own data.

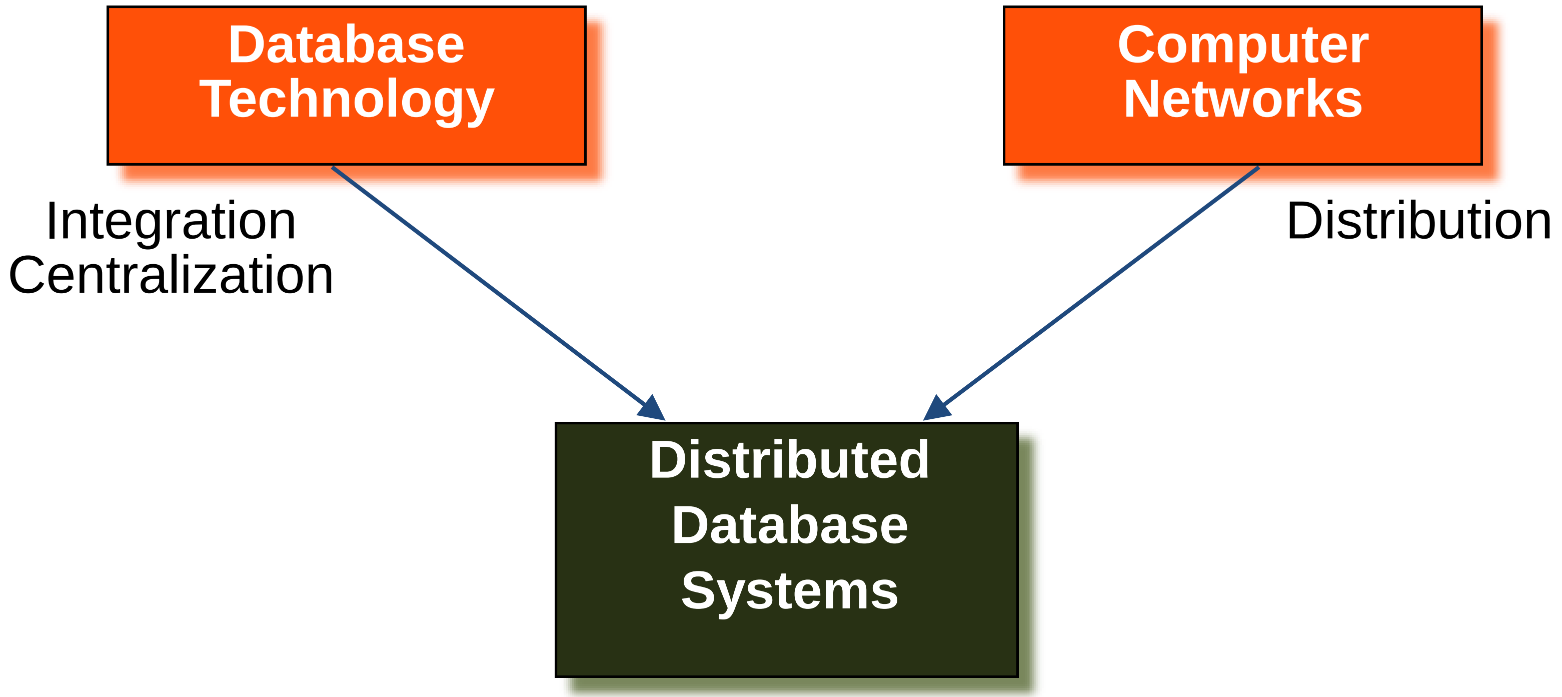


Database Management

Data are defined and administered centrally.


Application program 1 (with data semantics)
Application program 2 (with data semantics)
Application program 3 (with data semantics)
DBMS
description
manipulation
control
⋮
database

Motivation



integration ≠ centralization
Possible to achieve integration without centralization

Distributed Computing System

- A number of autonomous processing elements (not necessarily homogeneous) that are interconnected by a computer network and that cooperate in performing their assigned tasks.
- What is being distributed?
  - Processing logic or elements
  - Function
  - Data
  - Control
- Why do we distribute at all?

- From the viewpoint of distributed database systems, these modes of distribution are all necessary and important.
- Distributed database systems should also be viewed within this framework and treated as tools that could make distributed processing easier and more efficient.

What is a Distributed Database System?

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

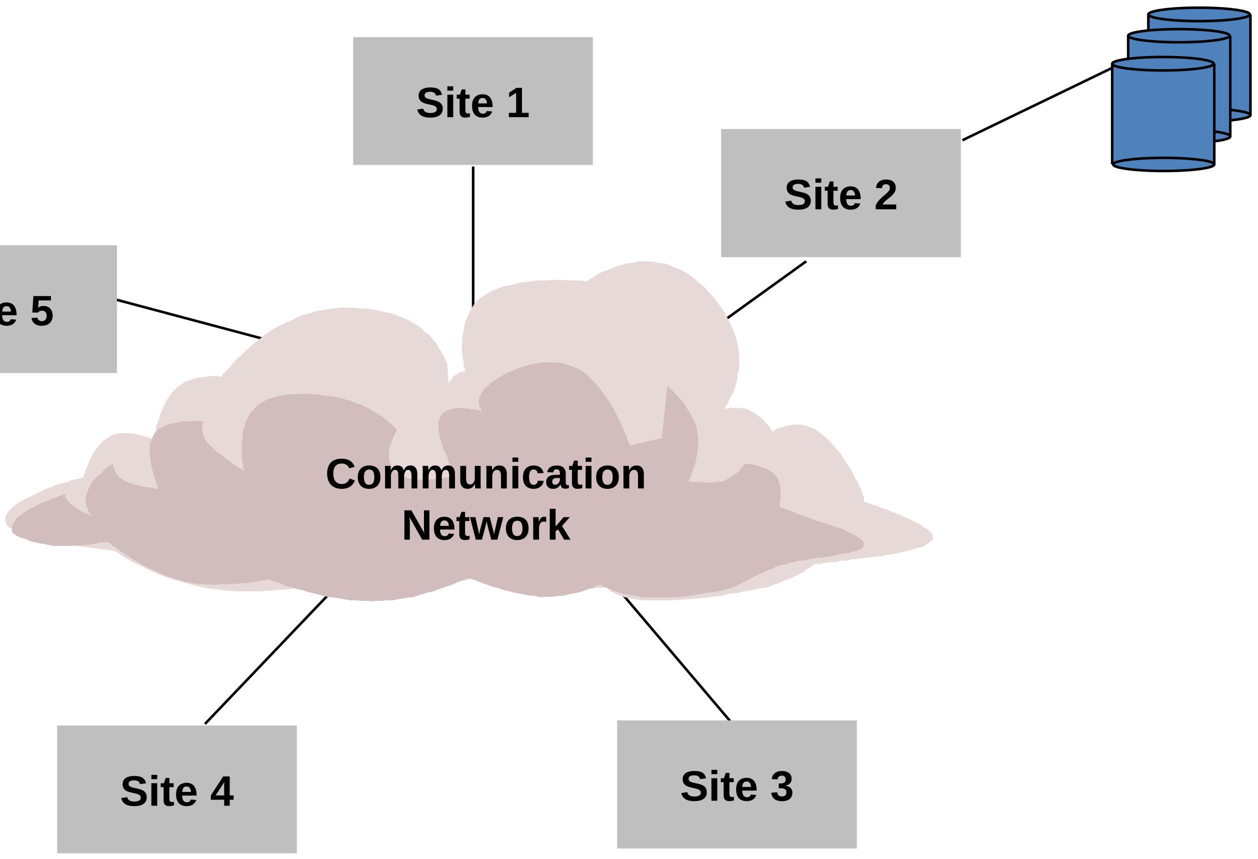
A distributed database management system (D–DBMS) is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users.

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

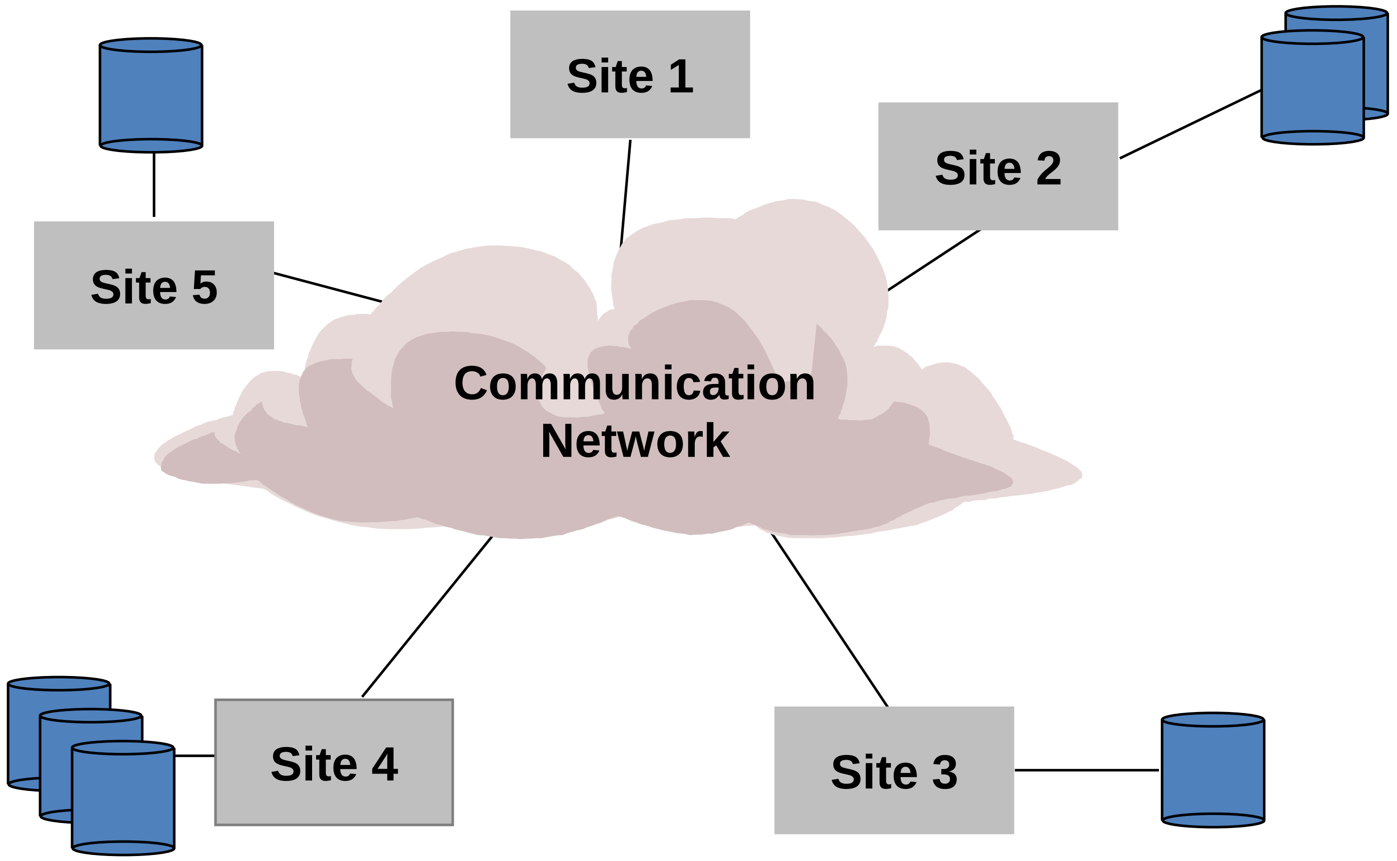
What is not a DDBS?

- Can loosely or tightly coupled multiprocessor system be considered as DDBS?
- Can a database system which resides at one of the nodes of a network of computers?
    - This is a centralized database on a network node
- Can you find out about shared everything and shared nothing architectures?

Centralized Database on a network



Distributed DBMS environment



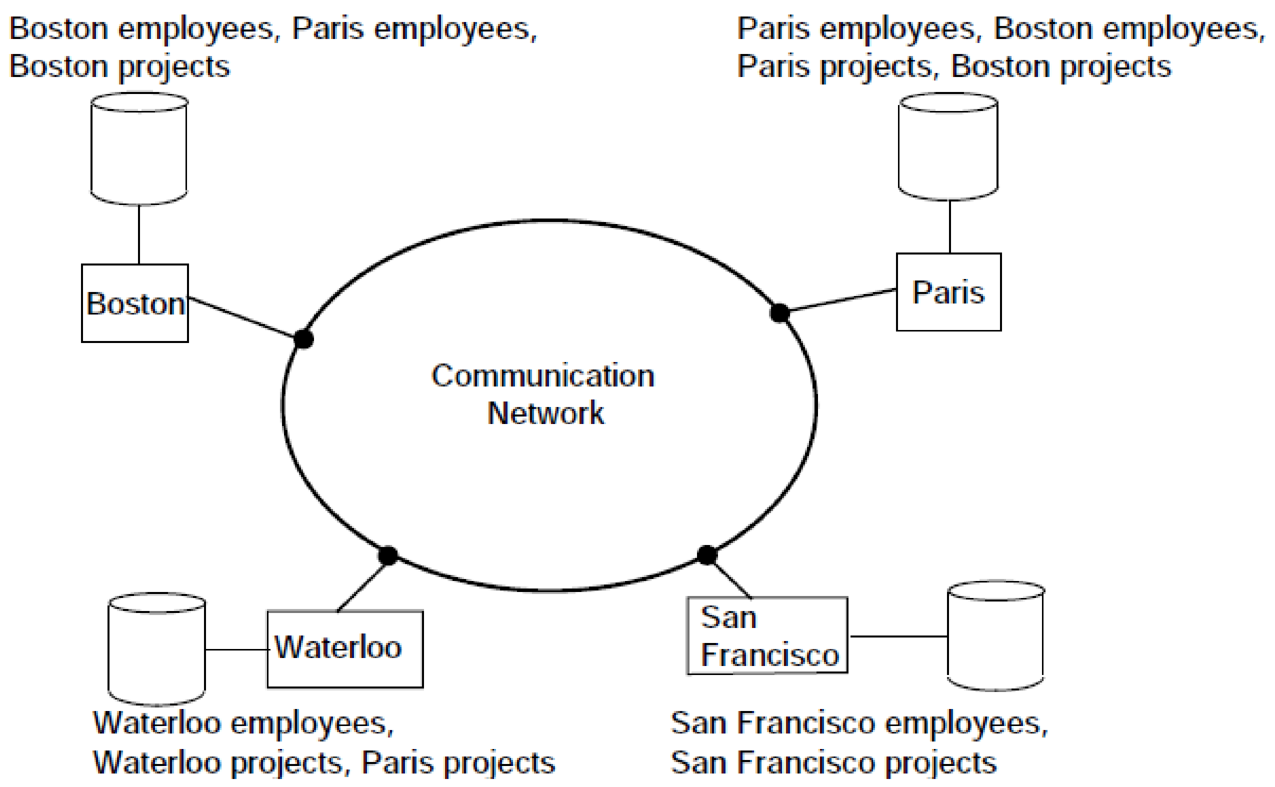
Distributed DBMS Promises

1. Transparent management of distributed, fragmented, and replicated data
2. Improved reliability/availability through distributed transactions
3. Improved performance
4. Easier and more economical system expansion

1. Transparency

- The concept of transparency extends the general idea of **hiding implementation details from end users.**
- A highly transparent system offers a lot of flexibility to the end user/application developer since it requires little or no awareness of underlying details on their part.

Transparent access



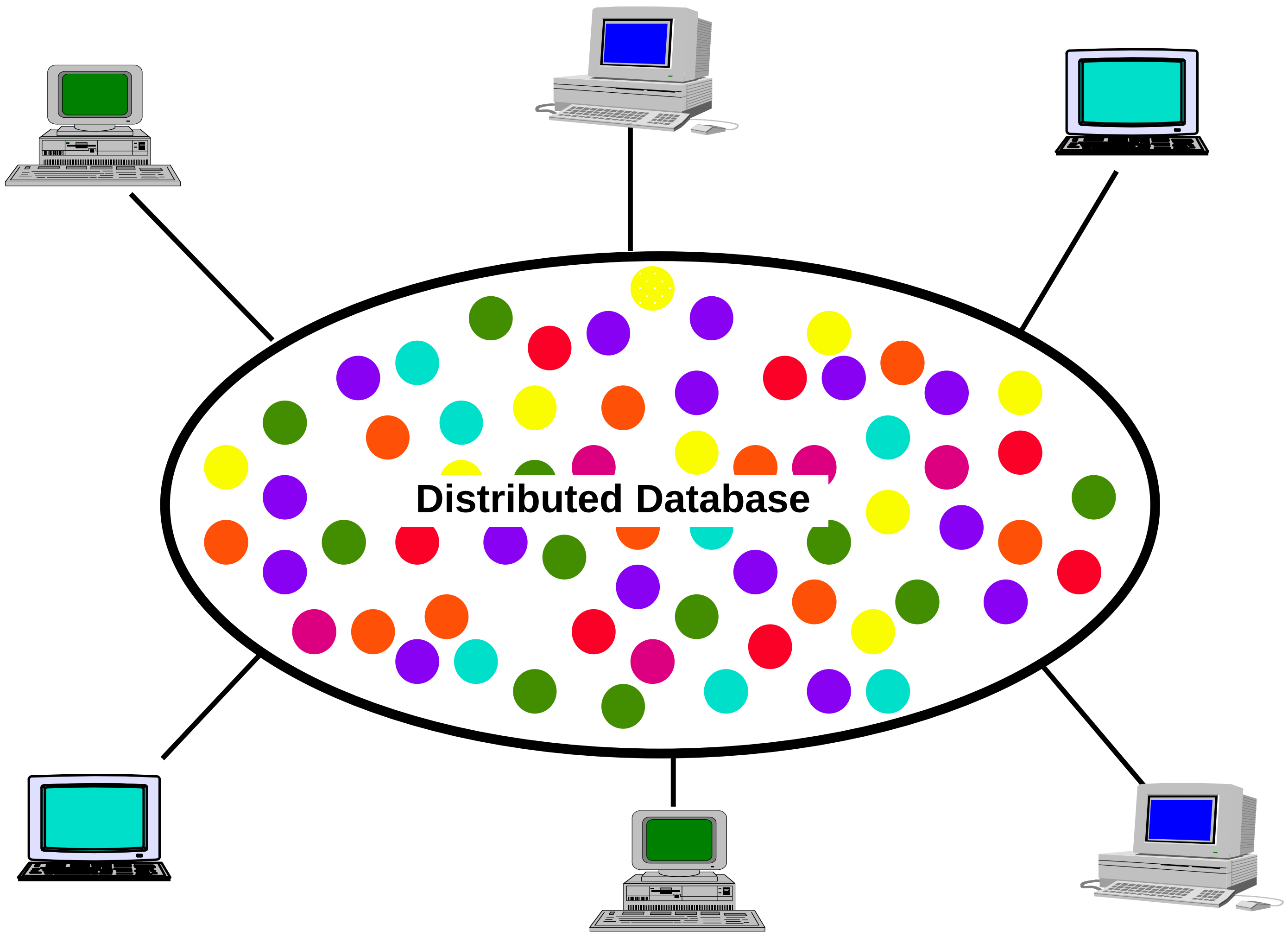
Boston employees, Paris employees, Boston projects
Paris employees, Boston employees, Paris projects, Boston projects
Boston
Paris
Communication Network
Waterloo
San Francisco
Waterloo employees, Waterloo projects, Paris projects
San Francisco employees, San Francisco projects

Find out the names and employees who worked on a project for
more than 12 months,

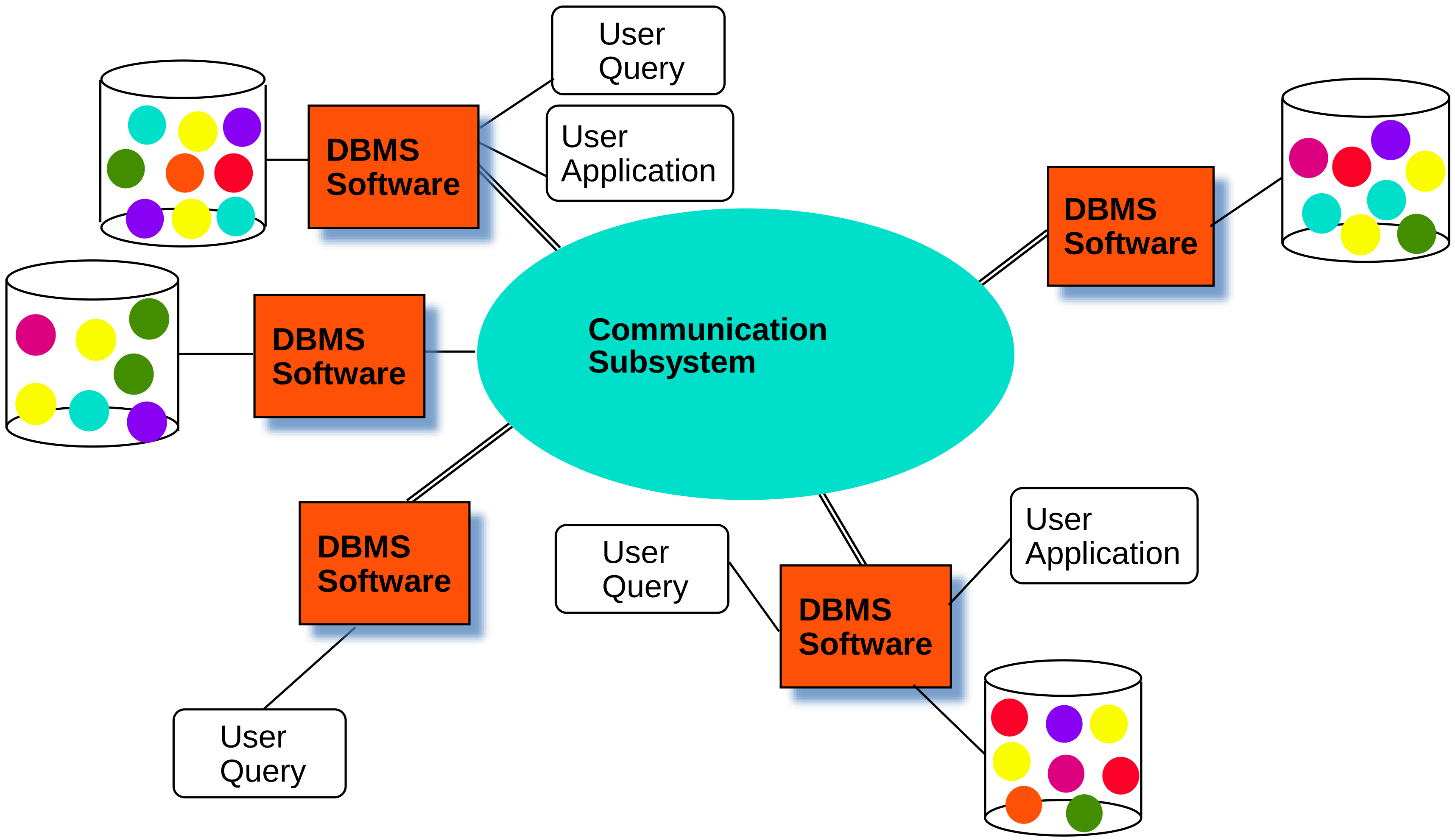
```
SELECT  ENAME, AMT
FROM    EMP, ASG, SAL
WHERE   ASG.DUR > 12
AND     EMP.ENO = ASG.ENO
AND     SAL.TITLE = EMP.TITLE
```

- Result is **distributed database** which is **fragmented and replicated.**
- **Fully transparent access** means that the users can still pose the query, without paying any attention to the fragmentation, location or replication of data, and let the system worry about resolving these issues.

Distributed Database - User View



Distributed DBMS - Reality



Types of transparency

- Data independence
- Network transparency (or distribution transparency)
  - Location transparency
  - Naming transparency
- Replication transparency
- Fragmentation transparency

Data independence

- Data definition occurs at two levels.
  - At one level the **logical structure of the data** are specified, and
  - At the other level its **physical structure**.
- The former is commonly known as the **schema definition**, whereas the latter is referred to as the **physical data description**.

Data independence

- **Logical data independence**
  - Immunity of user applications to changes in the **logical structure (i.e., schema)** of the database.
- **Physical data independence**
  - Deals with hiding the **details of the storage structure** from user applications. When a user application is written, it should not be concerned with the details of physical data organization.

Network transparency

- In a distributed database environment, there is another resource that needs to be managed in much the same manner: the network.
- The user should be protected from the **operational details of the network**; possibly even hiding the existence of the network.
- Can you think of transparency types that will enable network transparency?

Replication transparency

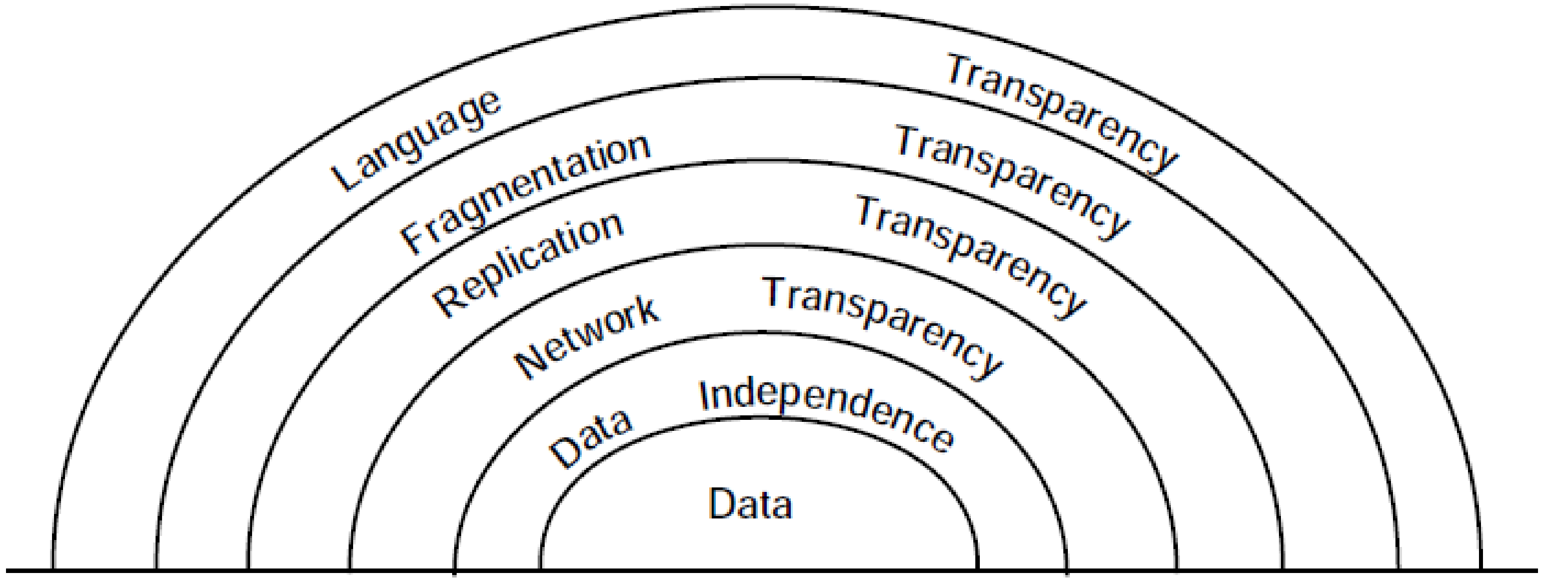
- Desirable to be able to distribute data in a replicated fashion across the machines on a network.
  - Increases the locality of reference
  - Increases the performance
  - Increases the reliability and availability

- Transparency issue is
  - whether the user should be aware of the existence of copies
  - whether the system should handle the management of copies and the user should act as if there is a single copy of the data

Fragmentation transparency

- Divide each database relation into smaller fragments and each fragment is treated as a separate database object.
- Fragmentation Alternatives
  - Vertical Fragmentation: each sub-relation is defined on a subset of attributes of the original relation.
  - Horizontal Fragmentation: each sub-relation is defined on a subset tuples of the original relation.
- Issue is to find a query processing strategy based on the fragments rather than the relations
  - A global query needs to be translated into several fragment queries.

Layers of transparency



2. Reliability through distributed transactions

- Replicated components and data should make distributed DBMS more reliable.
- **Transaction**
  - Basic unit of consistent and reliable computing, consisting of a sequence of database operations executed as an atomic action.
  - It transforms a consistent database state to another consistent database state.
- Distributed transactions provide
  - Concurrency transparency
  - Failure atomicity
- Distributed transaction support requires implementation of
  - Distributed concurrency control and reliability protocols
  - Two phase commit and distributed recovery protocols

3. Potentially improved performance

- Data is stored in close proximity to its point of use.
  - Since each site handles only a portion of the database, contention for CPU and I/O services is not that severe as for a centralized databases.
  - Localization reduces the access delays.
- Inherent parallelism of distributed systems can be exploited for inter-query and intra-query parallelism
  - Inter-query parallelism results from the ability to execute multiple queries at the same time
  - intra-query parallelism is achieved by breaking up a single query into a number of sub queries each of which is executed at a different site, accessing a different part of the distributed database.

4. Easier System Expansion

- In a distributed environment, it is much easier to accommodate **increasing database sizes**.
- Costs much less to put together a system of “smaller” computers with the equivalent power of a single big machine.

Distributed DBMS issues

Distributed Database Design

- How to distribute the database and applications
- Partitioned & replicated database distribution
- Fully replicated & partially replicated
- A related problem in directory management (A directory contains information (such as descriptions and locations) about data items in the database.)
  - Global to entire DDBS or local to each site

Distributed Query Processing

- Convert user transactions to data manipulation instructions
- Optimization problem involving decision on a strategy for executing each query over the network in most cost effective manner
  - min{cost = data transmission + distributed/local processing}

Distributed DBMS issues

Concurrency Control

- Synchronization of concurrent accesses, such that the integrity of the database is maintained
- **Consistency of multiple copies** of the database needs to be maintained.
  - The condition that requires all the values of multiple copies of every data item to converge to the same value is called **mutual consistency.**
  - These protocols can be **eager** in that they force the updates to be applied to all the replicas before the transaction completes, or they may be **lazy** so that the transaction updates one copy (called the master) from which updates are propagated to the others after the transaction completes.

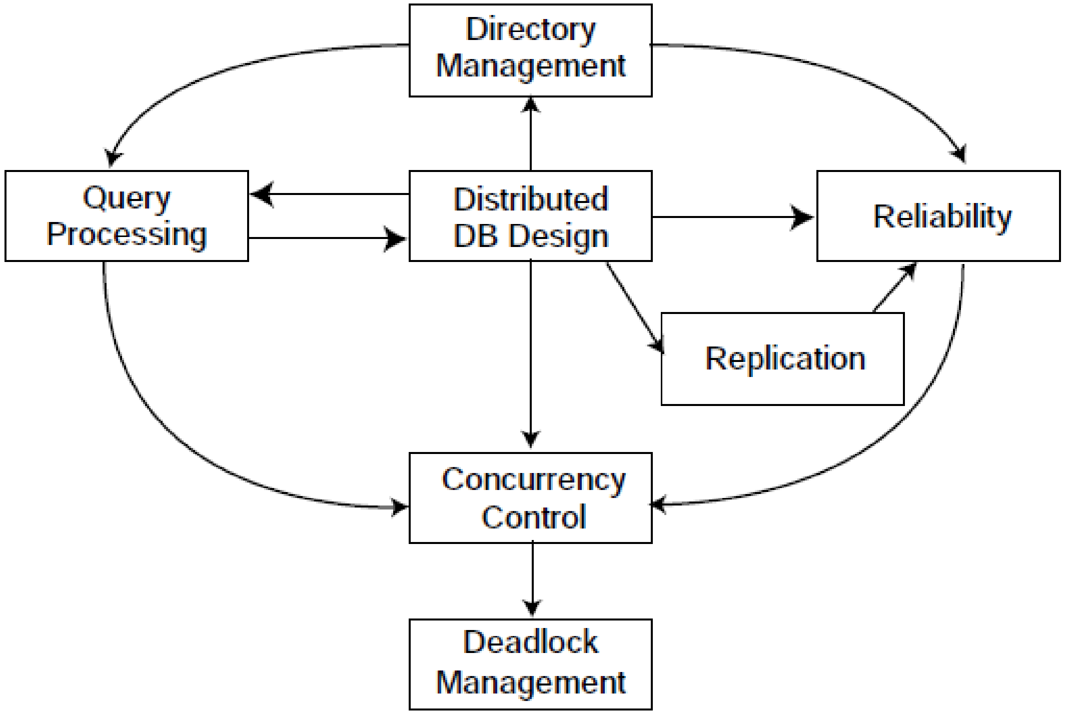
Distributed DBMS issues

Reliability

- How to make the system resilient to failure
- When a **failure occurs** and various **sites become either inoperable or inaccessible**, the databases at the operational sites remain consistent and up to date.
- Furthermore, when the computer system or network recovers from the failure, the DDBSs should be able to **recover** and bring the databases at the failed sites up-to-date.

Relationship between issues

- Study how are these issues interrelated?
  - Design of distributed databases affects directory management and query processing
  - Strong relationship among the concurrency control problem, the deadlock management problem, and reliability issues – Transaction management problem



Distributed DBMS architecture

Defines the structure of the system

- components identified
- functions of each component defined
- interrelationships and interactions between components defined

Reference architectures

Three “reference” architectures for a distributed DBMS:

- Client/server systems
- Peer-to-peer distributed DBMS
- Multidatabase systems

DBMS standardization

- Brief presentation of the “ANSI/SPARC architecture”
  - Used to define a DBMS architecture
  - Focuses on the different user classes and roles and their varying views on data.

Simplified version of the ANSI/SPARC architecture

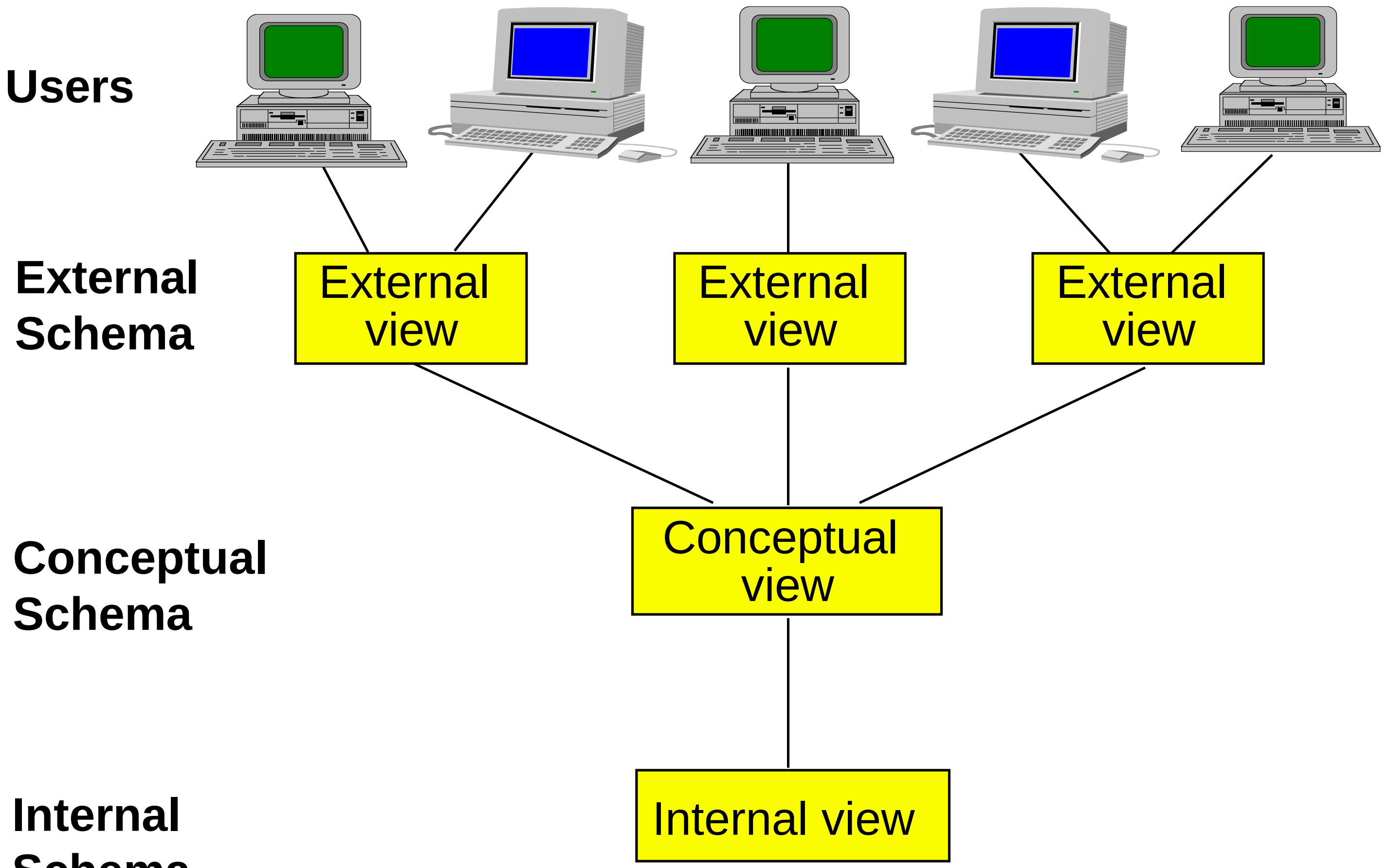
ANSI/SPARC recognizes three views of data:

- External view: user(programmer)/application view of the data.
- Conceptual view: enterprise view of the data.
- Internal view: machine/system view of the data.

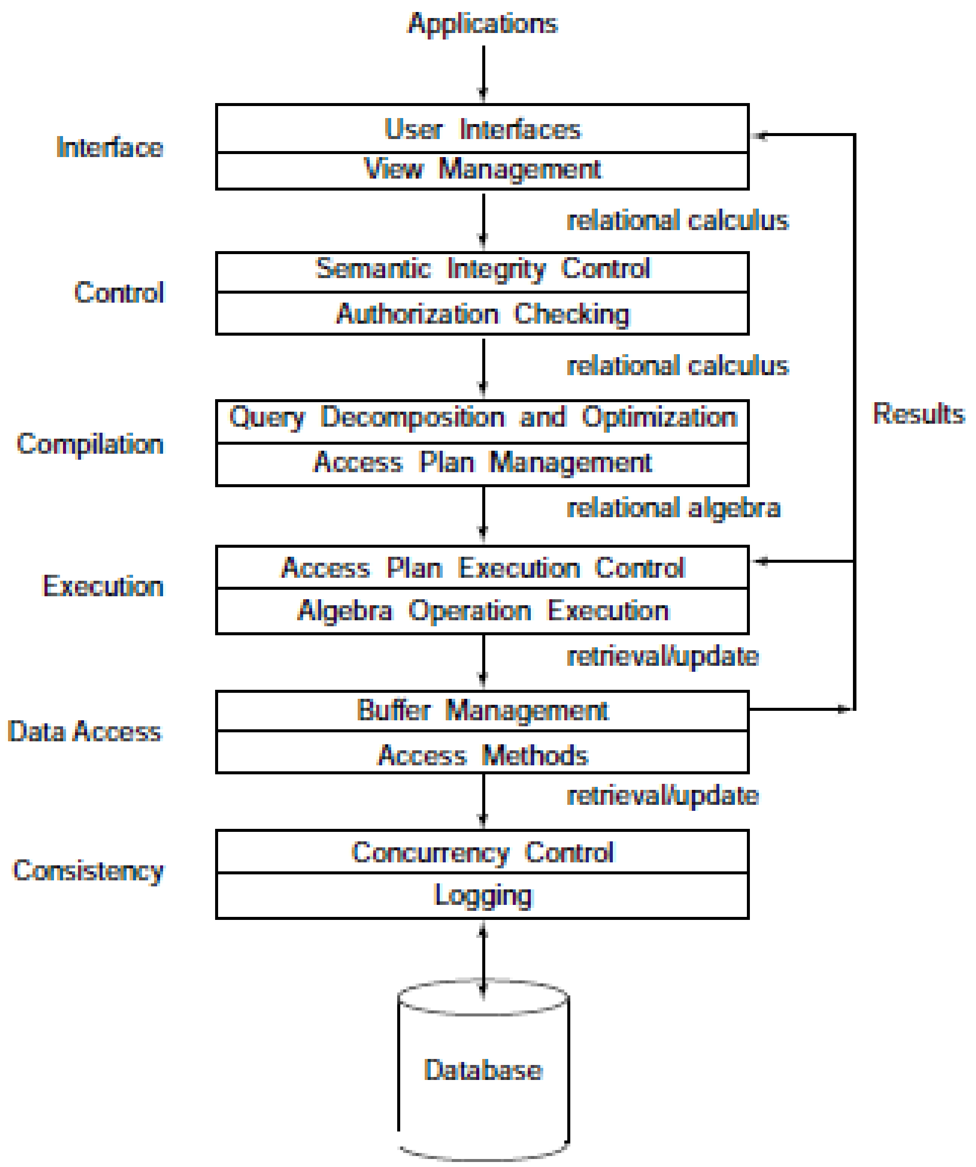
Simplified version of the ANSI/SPARC architecture

- The three-level ANSI architecture has an important place in database technology development because it clearly separates the users' external level, the database's conceptual level, and the internal storage level for designing a database.
  - **External view** - how users view the database
  - **Conceptual view** - abstract definition of the database – a "real world" view of the enterprise being modeled in the database
  - **Internal view** - physical definition and organization of data

ANSI/SPARC architecture



Functional layers of a centralized DBMS



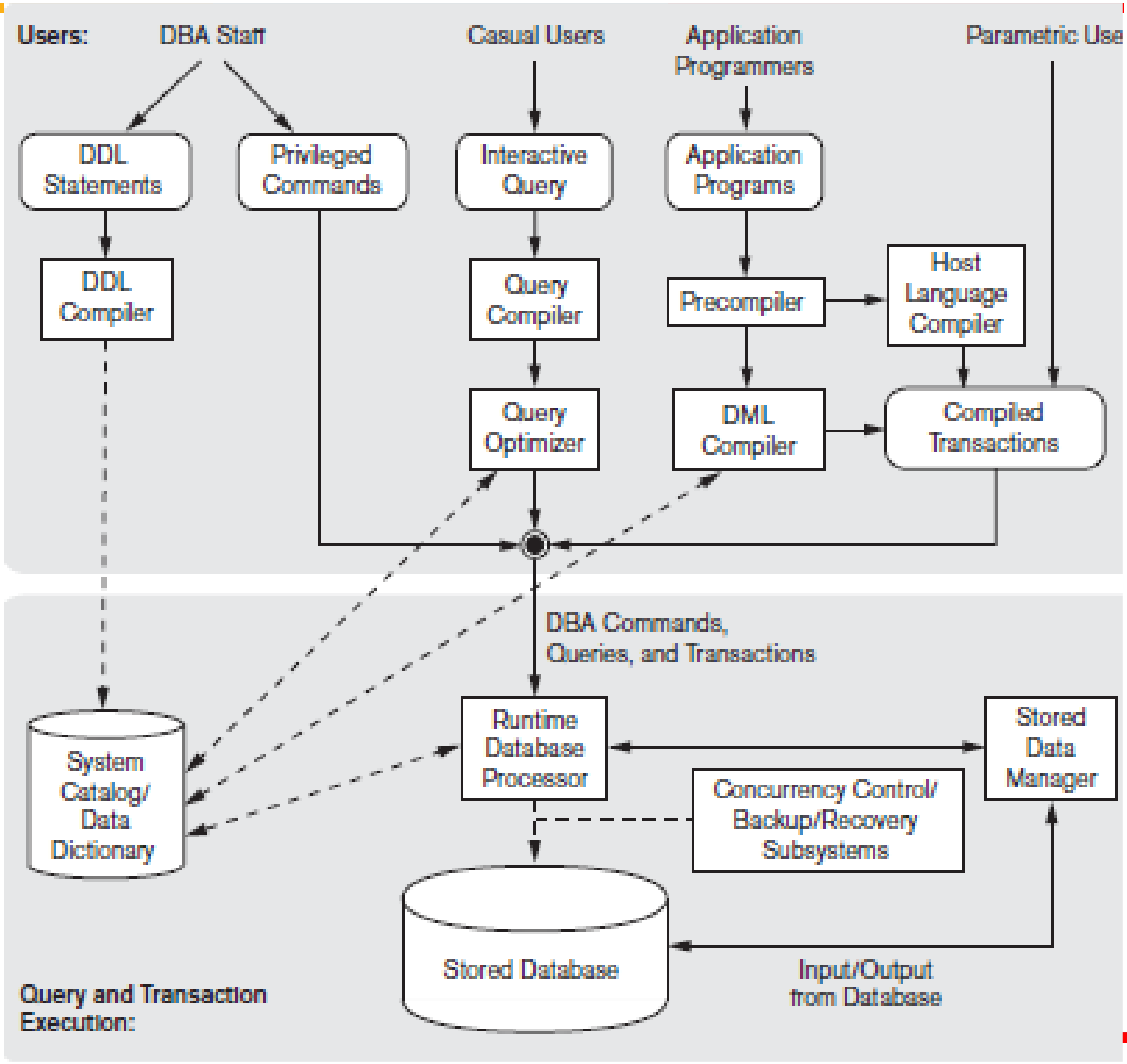
Generic centralized DBMS architecture

- A DBMS is a reentrant program shared by multiple processes (transactions), that run database programs.
- When running on a general purpose computer, a DBMS is interfaced with two other components: the communication subsystem and the operating system.
  - The **communication subsystem** permits interfacing the DBMS with other subsystems in order to communicate with applications. For example, the terminal monitor needs to communicate with the DBMS to run interactive transactions.
  - The **operating system** provides the interface between the DBMS and computer resources (processor, memory, disk drives, etc.).

Layer description

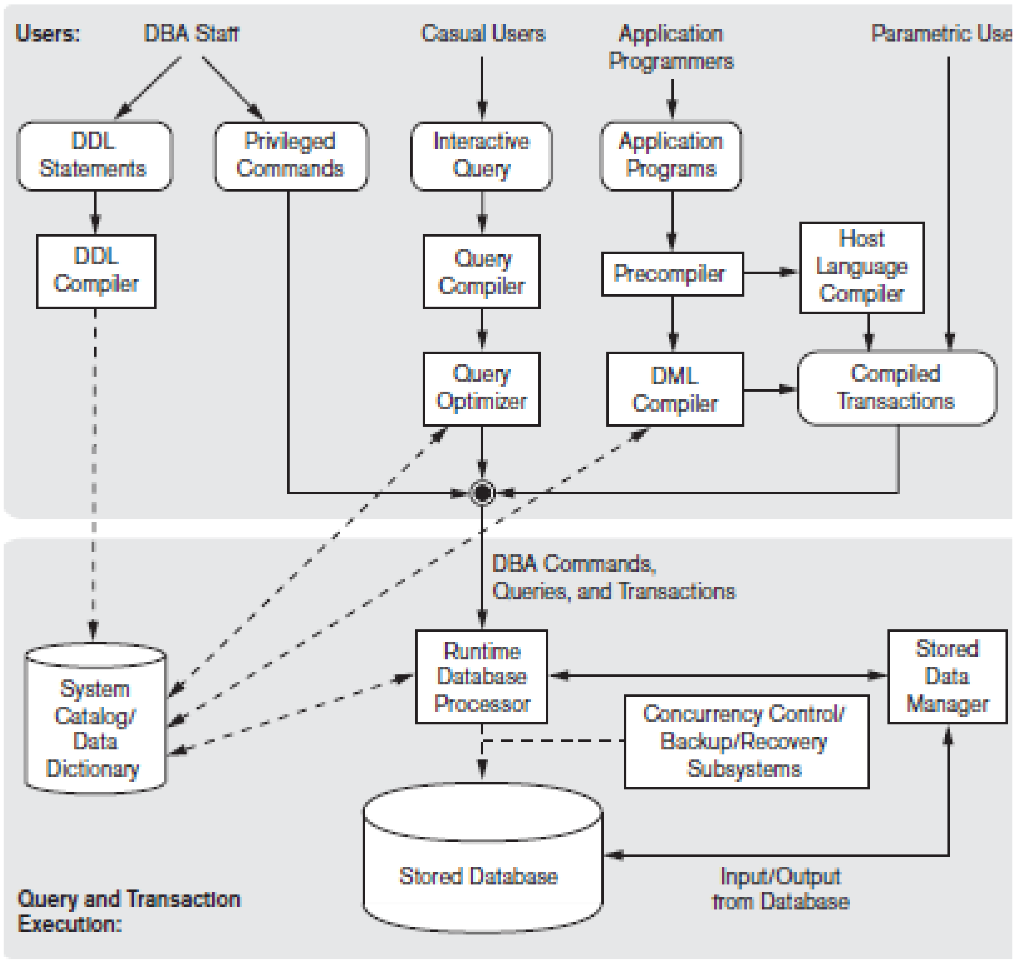
- The **interface layer** manages the interface to the applications. There can be several interfaces such as, in the case of relational DBMSs, SQL embedded in a host language, such as C.
- The **control layer** controls the query by adding semantic integrity predicates and authorization predicates.
- The **query processing** (or compilation) layer maps the query into an optimized sequence of lower-level operations.
- The **execution layer** directs the execution of the access plans, including transaction management (commit, restart) and synchronization of algebra operations.
- The **data access layer** manages the data structures that implement the files, indices, etc. It also manages the buffers by caching the most frequently accessed data.
- The **consistency layer** manages concurrency control and logging for update requests. This layer allows transaction, system, and media recovery after failure.

Discussion - What does this figure show?



Component modules of a DBMS and their interactions

- **DDL compiler** processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog.
- Queries are parsed and validated for correctness of the query syntax by a **query compiler** that compiles them into an internal form. This internal query is subjected to query optimization.
- **Query optimizer** is concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of efficient search algorithms during execution.
- **Runtime database processor** executes (1) the privileged commands, (2) the executable query plans, and (3) the canned transactions with runtime parameters.
    - It works with the **system catalog** and update it with statistics.
    - It also works with the **stored data manager**, which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory.
    - Refer to Section 2.4 Elmasri and Navathe

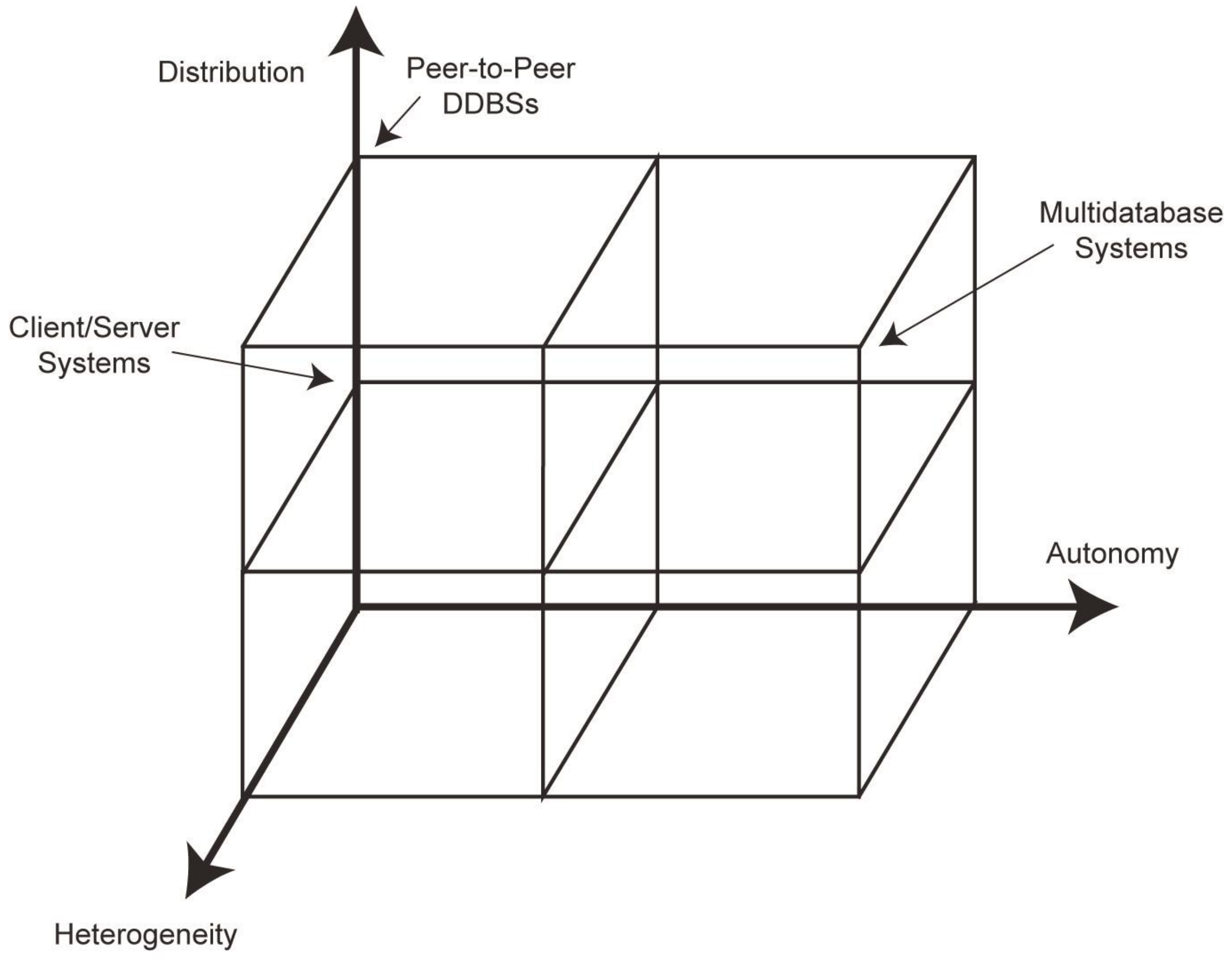


Architectural models for distributed DBMS

Systems can be characterized wrt

- Autonomy of local systems
- Their distribution
- Their heterogeneity

DBMS Implementation Alternatives



Autonomy

- *Autonomy* refers to the distribution of control, not data.
- It indicates the **degree to which individual DBMSs can operate independently.**
- Autonomy is a function of a number of factors
    - Exchange of information between component systems
    - Independent execution of transactions
    - Whether one is allowed to modify them

Dimensions of Autonomy

- **Design autonomy**: Ability of a component DBMS to decide on issues related to its own design (data models and transaction management techniques)
- **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 transactions in any manner it wants to.

Another autonomy classification

The taxonomy provides following classifications based on the autonomy :

•**Tight integration**: A single image of the entire database is available to any user who wants to share the information, which may reside in multiple databases. From users' perspective the database is **logically centralized in one database**. One DBMS controls the request of user (though multiple data managers are involved). No DBMS acts independently though it can do it.

Autonomy classification

- **Semiautonomous**: **Multiple DBMSs** that can operate independently, but have decided to participate in a federation to make their data sharable. Each of these DBMSs can **decide what data it can share with others.** They are not fully autonomous because they need to be modified to enable them to exchange information with others.
- **Total isolation**: Individual systems are **stand-alone DBMSs** which know neither of the existence of other DBMSs nor how to communicate with them. Difficult to execute global queries since there is no global over the execution of individual DBMSs.

Distribution

- Distribution dimension deals with data
  - **Client/server distribution**
    - Concentrates data management duties at servers
    - Clients focus on providing the application environment including the user interface.
    - The communication duties are shared between the client machines and servers. Client/server DBMSs represent a practical compromise to distributing functionality.
  - **Peer-to-peer distribution (or full distribution)**
    - There is no distinction of client machines versus servers.
    - Each machine has full DBMS functionality and can communicate with other machines to execute queries and transactions.

Heterogenity

- Heterogeneity may occur in various forms in distributed systems
  - Hardware heterogeneity
  - Differences in networking protocols
  - Variations in data managers
- It may also related to data models, query languages, and transaction management protocols.

Architectural Alternatives

- Considering all the three dimensions, following are the different alternatives:
  - Autonomy
    - 0 - tight integration (A0)
    - 1 - semiautonomous (A1)
    - 2 - Isolated (A2)
  - Distribution
    - 0 - No distribution (D0)
    - 1 - Client/server (D1)
    - 2 - Peer-to-peer (D2)
  - Heterogeneity
    - 0 - Homogeneous (H1)
    - 1 - Heterogeneous (H2)

Exercise – Characterise each of the below architectural alternatives

- (A0,D0,H0)
- (A0,D0,H1)
- (A0,D1,H0)
- (A0,D2,H0)
- (A1,D0,H0)
- (A1,D0,H1)
- (A1,D1,H1)
- (A2,D0,H0)
- (A2,D0,H1)
- (A2,D1,H1)/(A2,D2,H1)

Discussion – Architectural alternatives

- (A0,D0,H0) – Shared everything multiprocessor system
- (A0,D0,H1) – Integrated access to different databases on a single machine
- (A0,D1,H0) – Client server distribution
- (A0,D2,H0) – Fully distributed environment with no distinction between clients and servers

- (A1,D0,H0) – Federated DBMS
- (A1,D0,H1) – Heterogeneous federated DBMS
- (A1,D1,H1) – Distributed, Heterogeneous federated DBMS

Discussion – Architectural alternatives

- (A2,D0,H0) – Multidatabase system
- (A2,D0,H1) – Multiple storage systems with different characteristics
- (A2,D1,H1)/(A2,D2,H1) – Distributed MDBS

Architectural alternatives

Federated database system (A1,D0,H0)

- Each server is an independent and autonomous centralized DBMS that has its own local users, local transactions, and DBA, and hence has a very high degree of local autonomy.
- There is some global view or schema of the federation of databases that is shared by the applications.

Multidatabase system (A2,D0,H0)

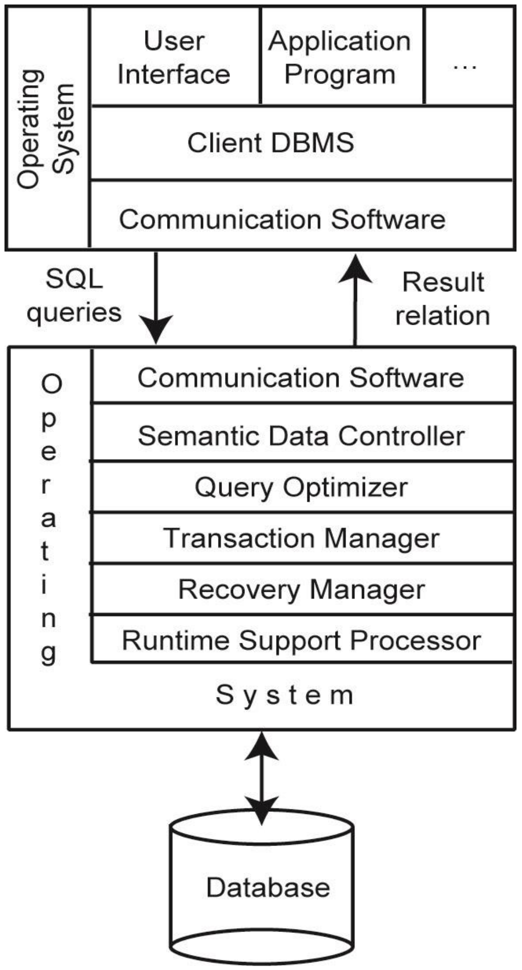
- Full local autonomy in that it does not have a global schema but interactively constructs one as needed by the application.

Distributed DBMS Architectures

- (A0,D1,H0) corresponds to client/server distributed DBMSs
- (A0,D2,H0) is a peer-to-peer distributed DBMS

Client/Server Architecture

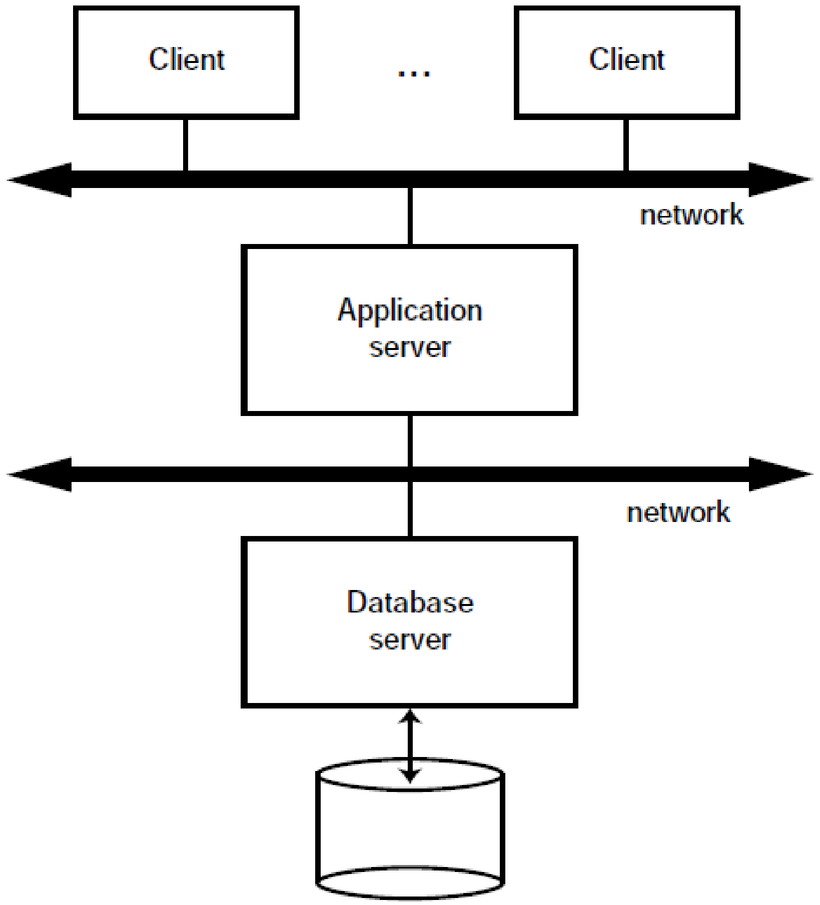
Divide functions into two classes: server functions and client functions.

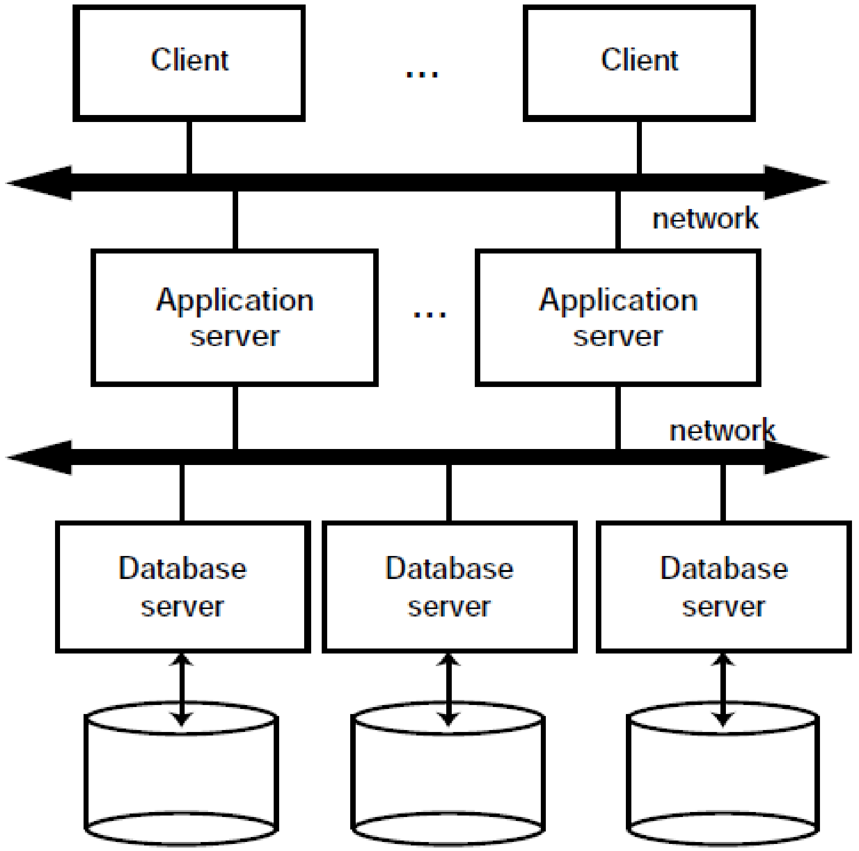


Client/Server Architecture

- In relational systems, the server does most of the data management work
  - query processing and optimization, transaction management and storage management is done at the server.
- The client, in addition to the application and the user interface, has a
  - DBMS client module that is responsible for managing the data that is cached to the client and (sometimes) managing the transaction locks that may have been cached as well.
- OS and Communication SW runs on both machines.
- Different types
  - Multiple client single server
  - Multiple client multiple server

Database server vs Distributed database server Approaches

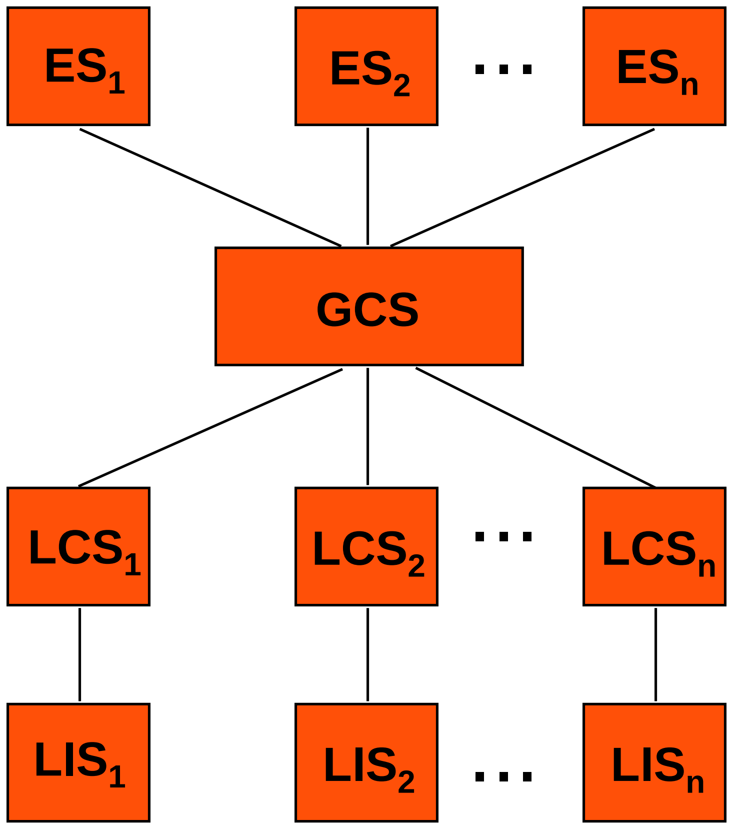




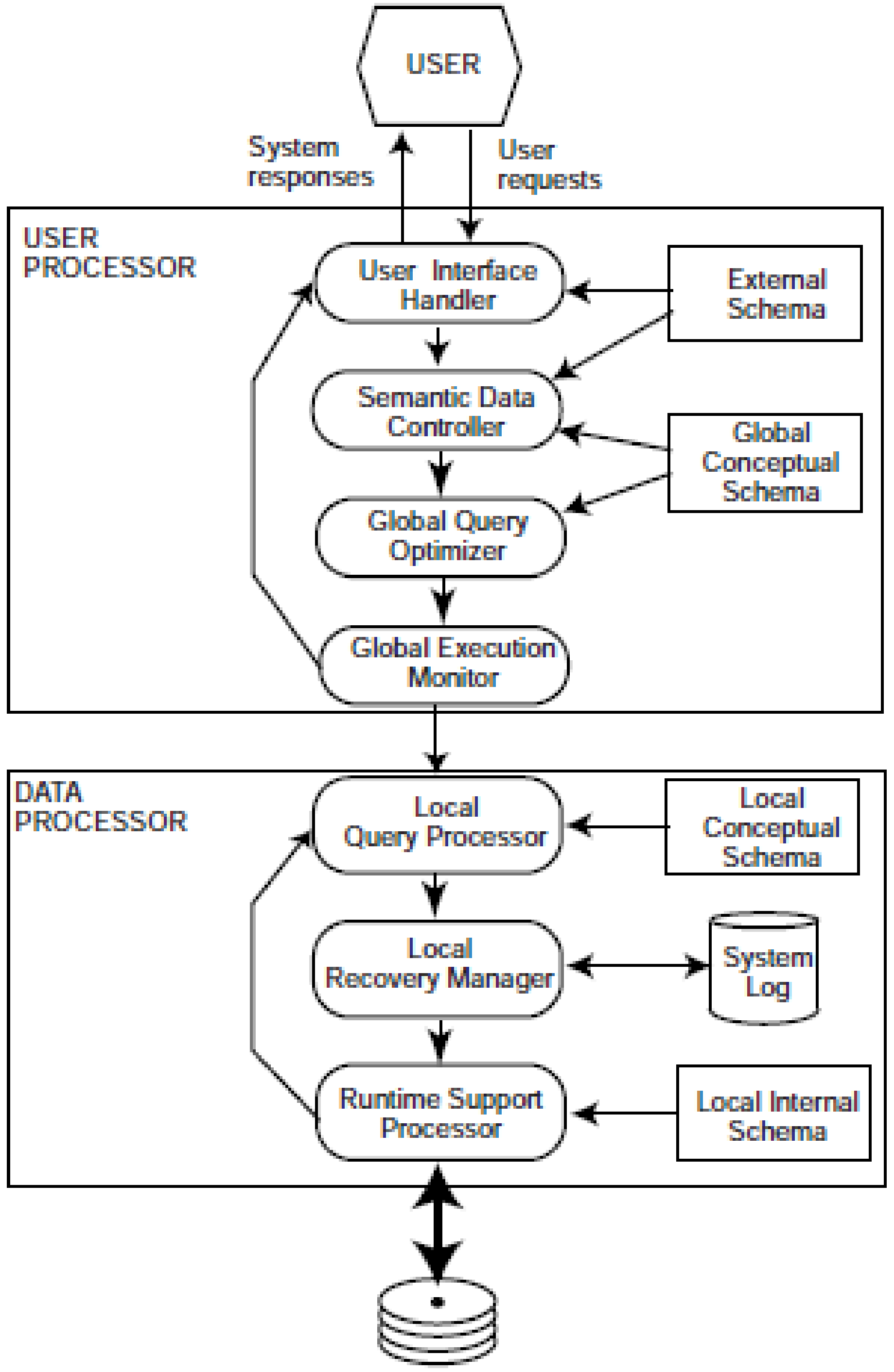
Peer to Peer distributed systems

- Physical data organization on each machine may be different. Hence we have individual internal schema definitions at each site Logical internal schema (LIS)
- The Enterprise level view of the data is given by Global Conceptual Schema (GCS). This is global because this describes the logical structure of the data for all sites.
- Due to fragmentation, each site should have a logical description of data at each site local conceptual schema (LCS).
- The GCS is union of LCSs.
- User applications and user access to database is supported by External Schemas (ES) are defined on GCS.

Distributed Database Reference Architecture



Components of a distributed DBMS



Components of a distributed DBMS

- Two important components of a DDBMS - The first handles the interaction with users, and the second one deals with the storage.
- User Processor:
  - The user interface handler is responsible for interpreting user commands as they come in, and formatting the results as it is sent to the user.
  - The semantic data controller uses the integrity constraints and authorizations that are defined as part of the global conceptual schema to check if the query can be processed. Also responsible for authorization.
  - The global query optimizer and decomposer determines an execution strategy to minimize the cost, and translates the global query into local ones using the global and local conceptual schemas and directories. Global optimizer is responsible for generating the best strategy to execute distributed joins.
  - The distributed execution monitor coordinates the distributed execution of the user queries. The execution monitor is also called as distributed transaction manager. To execute queries in a distributed fashion, the execution monitors at various sites usually do communicate with one another.

Components of a distributed DBMS

- Local Data Processor
  - The Local query optimizer, which actually acts as the access path selector, is responsible for choosing the best access path to access any data item.
  - The Local recovery manager is responsible for making sure that the local database remains consistent even when failures occur.
  - The Run-time support processor physically accesses the database according to the physical commands in the schedule generated by the query optimizer. The run-time support processor is the interface to the OS and contains the database buffer manager.
- Usually in a DDBS systems we can have both components on each machine.

Lecture Summary

Topics Covered

- Introduction to Distributed databases
- Distributed DBMS Architecture

Essential Readings

- Chapter 1,4 Tamer Ozsu

Thanks…

Next Lecture

- Distributed database design
- Fragmentation and Allocation

Questions??