**Chapter 2: DDBMS Architecture**

* Definition of the DDBMS Architecture
* ANSI/SPARC Standard
* Global, Local, External, and Internal Schemas, Example
* DDBMS Architectures
* Components of the DDBMS

**Acknowledgements:** I am indebted to Arturas Mazeika for providing me his slides of this course.

* **Architecture:** The architecture of a system defines its structure:
  + the components of the system are identified;
  + the function of each component is specified;
  + the interrelationships and interactions among the components are defined.
* Applies both for computer systems as well as for software systems, e.g,
  + division into modules, description of modules, etc.
  + architecture of a computer
* There is a close relationship between the architecture of a system, standardisation efforts, and a reference model.
* DDBMS might be implemented as homogeneous or heterogeneous DDBMS
* **Homogeneous** DDBMS
  + All sites use same DBMS product
  + It is much easier to design and manage
  + The approach provides incremental growth and allows increased performance
* **Heterogeneous** DDBMS
  + Sites may run different DBMS products, with possibly different underlying data models
  + This occurs when sites have implemented their own databases first, and integration is considered later
  + Translations are required to allow for different hardware and/or different DBMS products
  + Typical solution is to use gateways

⇒ A common standard to implement DDBMS is needed!

* The standardization efforts in databases developed reference models of DBMS.
* **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 each other.

* A reference model can be thought of as an **idealized architectural model** of the system.
* Commercial systems might deviate from reference model, still they are useful for the standardization process
* A reference model can be described according to 3 different approaches:
  + component-based
  + function-based
  + data-based

# Components-based

* + Components of the system are defined together with the interrelationships between the components
  + Good for design and implementation of the system
  + It might be difficult to determine the functionality of the system from its components

# Function-based

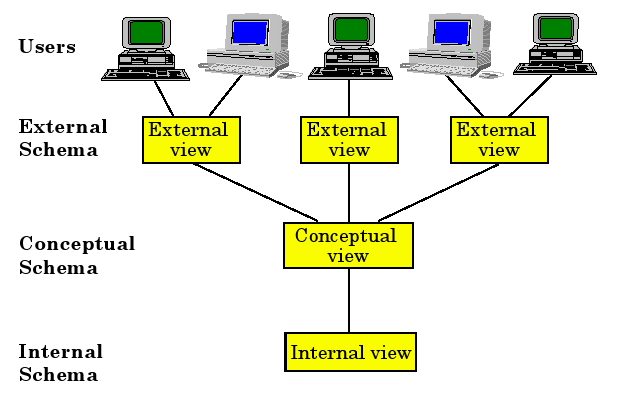
* + Classes of users are identified together with the functionality that the system will provide for each class
  + Typically a hierarchical system with clearly defined interfaces between different layers
  + The objectives of the system are clearly identified.
  + Not clear how to achieve the objectives
  + Example: ISO/OSI architecture of computer networks

# Data-based

* + Identify the different types of the data and specify the functional units that will realize and/or use data according to these views
  + Gives central importance to data (which is also the central resource of any DBMS)

→ Claimed to be the preferable choice for standardization of DBMS

* + The full architecture of the system is not clear without the description of functional modules.
  + Example: ANSI/SPARC architecture of DBMS
* The interplay among the 3 approaches is important:
  + Need to be used together to define an architectural model
  + Each brings a different point of view and serves to focus on different aspects of the model
* ANSI/SPARC architecture is based on data
* 3 views of data: external view, conceptual view, internal view
* Defines a total of 43 interfaces between these views



* Conceptual schema: Provides enterprise view of entire database

**RELATION** EMP [ **KEY** = {ENO} **ATTRIBUTES** = {

ENO : **CHARACTER**(9) ENAME: **CHARACTER**(15) TITLE: **CHARACTER**(10)

}

]

**RELATION** PAY [ **KEY** = {TITLE} **ATTRIBUTES** = {

TITLE: **CHARACTER**(10) SAL : **NUMERIC**(6)

}

]

**RELATION** PROJ [ **KEY** = {PNO} **ATTRIBUTES** = {

PNO : **CHARACTER**(7) PNAME : **CHARACTER**(20) BUDGET: **NUMERIC**(7) LOC : **CHARACTER**(15)

}

]

**RELATION** ASG [ **KEY** = {ENO,PNO} **ATTRIBUTES** = {

ENO : **CHARACTER**(9) PNO : **CHARACTER**(7) RESP: **CHARACTER**(10) DUR : **NUMERIC**(3)

}

]

* Internal schema: Describes the storage details of the relations.
  + Relation EMP is stored on an indexed file
  + Index is defined on the key attribute ENO and is called EMINX
  + A HEADER field is used that might contain flags (delete, update, etc.)

**INTERNAL REL** EMPL [

**INDEX ON** E# **CALL** EMINX

**FIELD** =

HEADER: **BYTE**(1) E# : **BYTE**(9) ENAME : **BYTE**(15)

Conceptual schema:

**RELATION** EMP [ **KEY** = {ENO} **ATTRIBUTES** = {

ENO : **CHARACTER**(9) ENAME: **CHARACTER**(15) TITLE: **CHARACTER**(10)

TIT : **BYTE**(10) ] }

]

* External view: Specifies the view of different users/applications
  + Application 1: Calculates the payroll payments for engineers

**CREATE VIEW** PAYROLL (ENO, ENAME, SAL) **AS SELECT** EMP.ENO,EMP.ENAME,PAY.SAL

**FROM** EMP, PAY

**WHERE** EMP.TITLE = PAY.TITLE

* + Application 2: Produces a report on the budget of each project

**CREATE VIEW** BUDGET(PNAME, BUD) **AS SELECT** PNAME, BUDGET

**FROM** PROJ

* Architectural Models for DDBMSs (or more generally for multiple DBMSs) can be classified along three dimensions:
  + Autonomy
  + Distribution
  + Heterogeneity
* **Autonomy**: Refers to the distribution of control (not of data) and indicates the degree to which individual DBMSs can operate independently.

Autonomy is a function of a number of factors such as whether the component systems (i.e.,

Individual DBMSs) exchange information, whether they can independently execute transactions, and whether one is allowed to modify them.

* Requirements of an autonomous system have been specified as follows

[Gligor and Popescu-Zeletin,1986]:

1. The local operations of the individual DBMSs are not affected by their participation

in the distributed system.

2. The manner in which the individual DBMSs process queries and optimize

them should not be affected by the execution of global queries that access

multiple databases.

3. System consistency or operation should not be compromised when individual

DBMSs join or leave the distributed system.

* On the other hand, the dimensions of autonomy can be specified as follows [Du

and Elmagarmid, 1989]:

* + *Design autonomy* : each individual DBMS is free to use the data models and transaction management techniques that it prefers.
  + *Communication autonomy* : each individual DBMS is free to decide what information to provide to the other DBMSs
  + *Execution autonomy* : each individual DBMS can execture the transactions that are submitted to it in any way that it wants to.
* We will use a classification that covers the important aspects of these features
  + *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 DBs);

From the users’ perspective, the data are logically integrated in one database. In these tightly-integrated systems one data manager is in control of the processing of each user request. The data managers do not typically operate as independent DBMSs

* + *Semiautonomous* systems: individual DBMSs can operate independently, but have decided to participate in a federation to make some of their local data sharable.

Each of these DBMSs determines what parts of their own database they will make accessible to users of other DBMSs. They are not fully autonomous systems because they need to be modified to enable them to exchange information with one another.

* + *Total isolation*: the individual systems are stand-alone DBMSs, which know neither of the existence of other DBMSs nor how to communicate with them; there is no global control over the execution of individual DBMSs.
* **Distribution**: Distribution Taxonomy Refers to the physical distribution of data over multiple sites.

The taxonomy identifies three alternative architectures

* + *No distribution*: No distribution of data at all
  + *Client/Server distribution*:

∗ Data management duties are concentrated on the server, while clients provide application environment/user interface. The communication duties are shared between the client machines and servers.

∗ First attempt to distribution

* + *Peer-to-peer distribution* (also called *full distribution*):

∗ No distinction between client and server machine

∗ Each machine has full DBMS functionality and can communicate with other machines to execute queries and transactions.

* **Heterogeneity**: Refers to heterogeneity of the components at various levels
  + hardware
  + communications
  + operating system
  + DB components (e.g., data model, query language, transaction management algorithms)

**Architectural Alternatives**

Identified three alternative architectures:

Dimensions are A(autonomy), D(distribution), H(heterogeneity)

A (autonomy):

0 - tight integration

1 - semiatonomous

2 - total isolation

D (distribution):

0 – no distribution

1 – client/server system

2 – peer-to-peer distribution

H (heterogeneity):

0 – homogeneous

1 - heterogeneous

* (A0,D1, H0) that corresponds to client/server distributed DBMSs,
* (A0, D2, H0) that is a peer-to-peer distributed DBMS and
* (A2, D2, H1) which represents a (peer-to-peer) distributed, heterogeneous multi database system.
* Client/Server Systems
* Peer – to- Peer Distributed Systems
* Multi Database management systems (MDBS) Architecture
* General idea: Divide the functionality into two classes:
  + server functions

∗ mainly data management, including query processing, optimization, transaction management, storage management, etc.

* + client functions

∗ might also include some data management functions (consistency checking,

transaction locks management, etc.)not just user interface

* Provides a two-level architecture
* More efficient division of work
* The communication between the clients

and the server(s) is at the level of SQL statements.

The client passes SQL queries to the server without trying to understand or optimize them. The server does most of the work and returns the result relation to the client.

* Different types of client/server architecture
  + Multiple client/single server

In this case where there is only one server which is accessed by multiple clients.

From a data management perspective, this is not

much different from centralized databases since the database is stored on only one

machine (the server) that also hosts the software to manage it.

* + Multiple client/multiple server

In this there are multiple servers.

In this case, two alternative management strategies are possible:

1. either each client manages its own connection to the appropriate server

or

1. each client knows of only its “home server” which then communicates with other servers as required.

The former approach simplifies server code, but loads the client machines with additional responsibilities. This leads to what has been called “heavy client” systems.

The latter approach, on the other hand, concentrates the data management functionality at the servers. Thus, the transparency of data access is provided at the server interface, leading to “light clients.”

From a datalogical perspective, client/server DBMSs provide the same view of

data as do peer-to-peer systems. That is, they give the user the

appearance of a logically single database, while at the physical level data may be

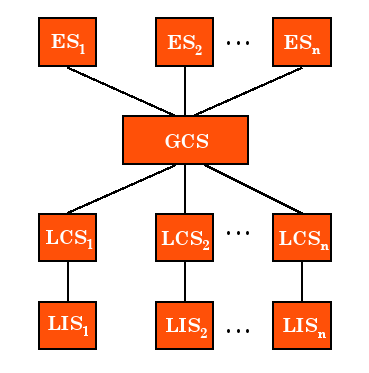
distributed.

Thus the primary distinction between client/server systems and peer-to-

peer ones is not in the level of transparency that is provided to the users and

applications, but in the architectural paradigm that is used to provide

transparency.



* *Local internal schema* (LIS)
  + Describes the local physical data or- ganization (which might be different on each machine)
* *Local conceptual schema* (LCS)
  + Describes logical data organization at each site
  + Required since the data are frag- mented and replicated
* Global conceptual schema (GCS)
  + Describes the global logical view of the data

The enterprise view of the data is described

by the global conceptual schema (GCS),

which is global because it describes the

logical structure of the data at all the sites.

* + Union of the LCSs
* *External schema* (ES)
  + Describes the user/application view on the data

Location and replication transparencies are supported by the definition of the local and global

conceptual schemas and the mapping in between. Network transparency, on the other hand, is supported by the definition of the global conceptual schema.

The user queries data irrespective of its location or of which local component of the distributed

database system will service it. As mentioned before, the distributed DBMS translates

global queries into a group of local queries, which are executed by distributed DBMS

components at different sites that communicate with one another.

The detailed components of a distributed DBMS are:

One component handles the interaction with users, and another deals with the storage.

The first major component, which we call the user processor, consists of four elements:

1. The user interface handler is responsible for interpreting user commands as

they come in, and formatting the result data as it is sent to the user.

2. The semantic data controller uses the integrity constraints and authorizations

that are defined as part of the global conceptual schema to check if the user

query can be processed.

3. The global query optimizer and decomposer determines an execution strategy

to minimize a cost function, and translates the global queries into local ones

using the global and local conceptual schemas as well as the global directory.

The global query optimizer is responsible, among other things, for generating

the best strategy to execute distributed join operations.

4. The distributed execution monitor coordinates the distributed execution of the

user request. The execution monitor is also called the distributed transaction

manager. In executing queries in a distributed fashion, the execution monitors

at various sites may, and usually do, communicate with one another.

The second major component of a distributed DBMS is the data processor and

Consists of three elements:

1. 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.

2. The local recovery manager is responsible for making sure that the local

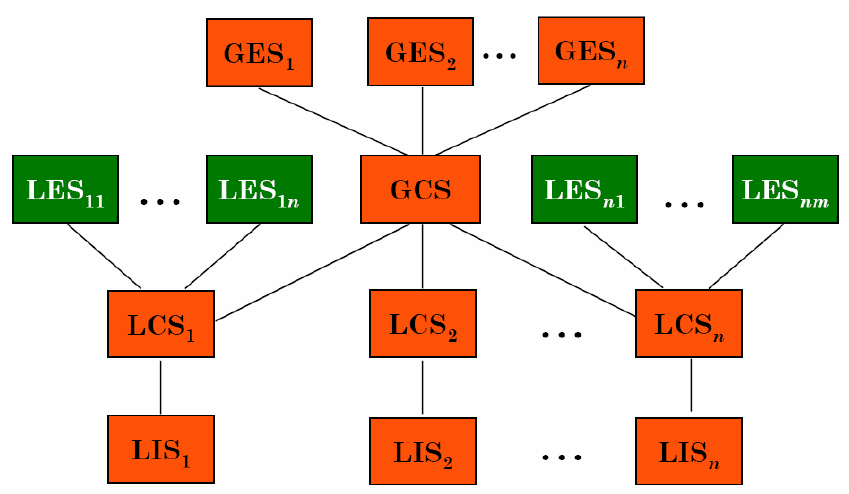
Data base remains consistent even when failures occur.

3. The run-time support processor physically accesses the database according

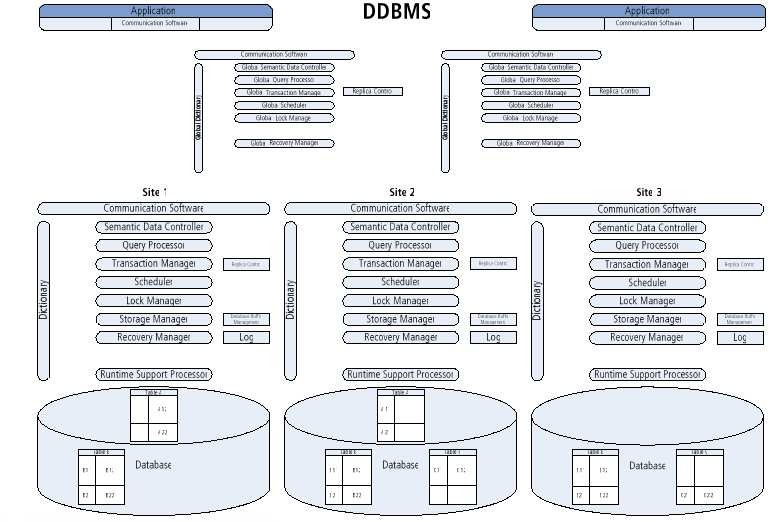
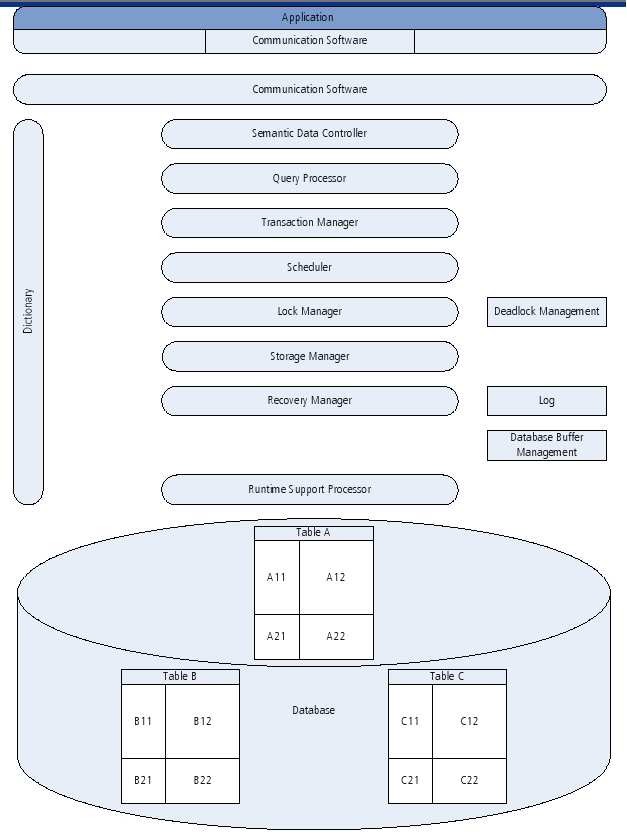
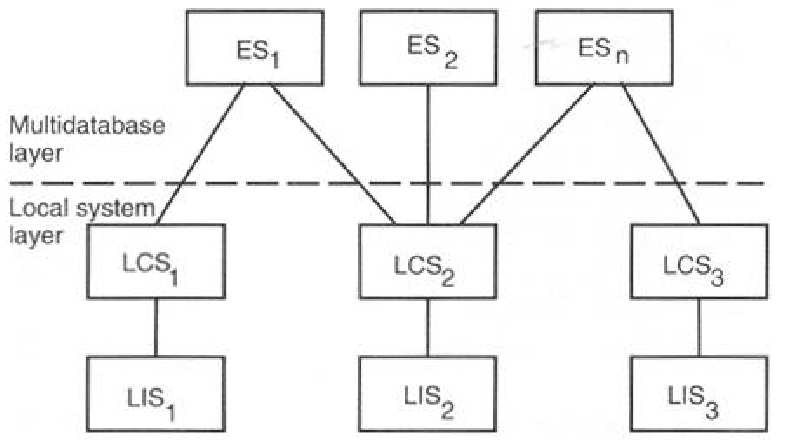
to the physical commands in the schedule generated by the query optimizer.

The run-time support processor contains the database buffer (or cache) manager, which is responsible for maintaining the main memory buffers and managing the data accesses.

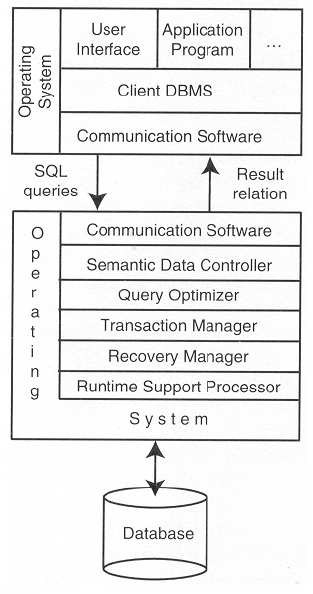
* Fundamental difference to peer-to-peer DBMS is in the definition of the global conceptual schema (GCS)
  + In a MDBMS the GCS represents only the collection of *some* of the local databases that each local DBMS want to share.
* This leads to the question, whether the GCS should even exist in a MDBMS?
* Two different architecutre models:
  + Models with a GCS
  + Models without GCS
* Model with a GCS
  + GCS is the union of parts of the LCSs
  + Local DBMS define their own views on the local DB



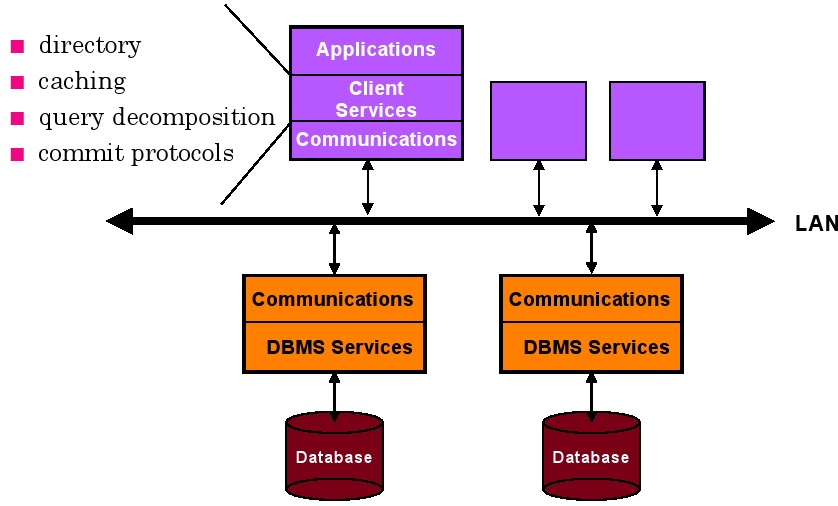
* Model without a GCS
  + The local DBMSs present to the multi-database layer the part of their local DB they are willing to share.
  + External views are defined on top of LCSs



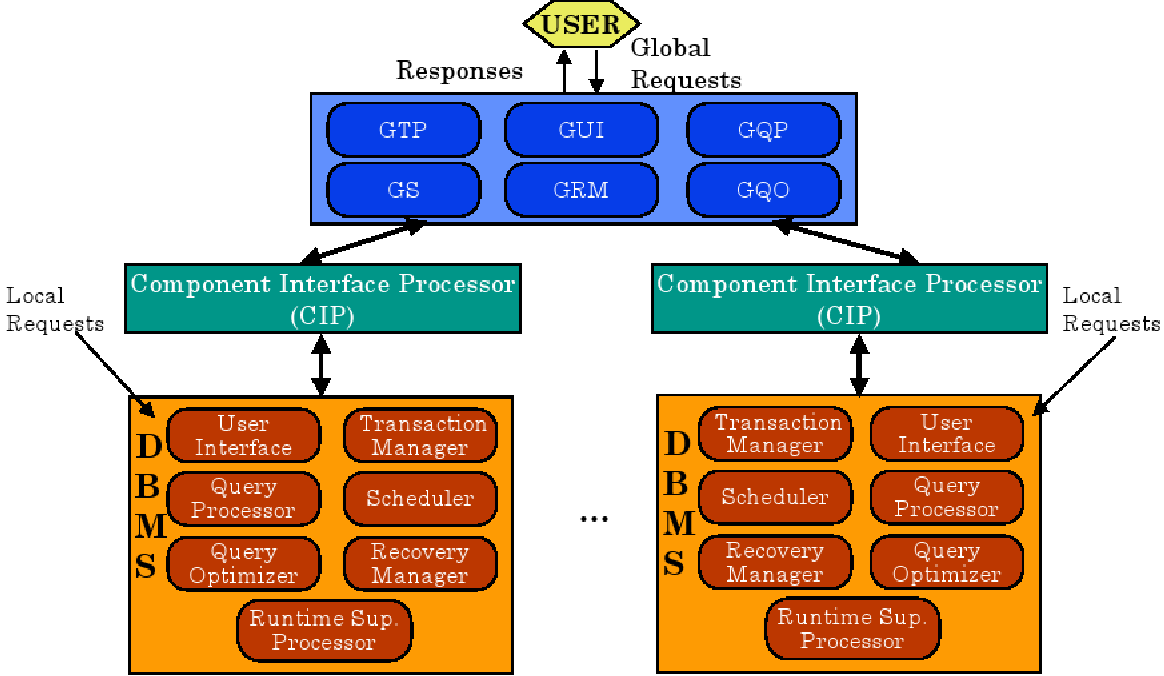
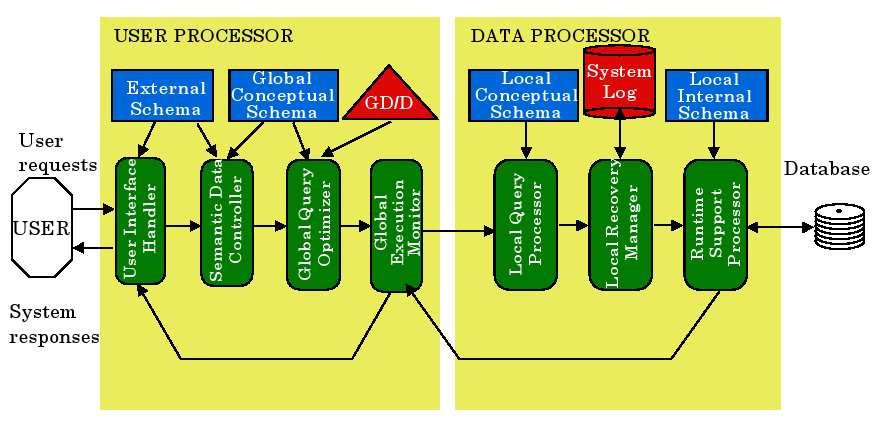
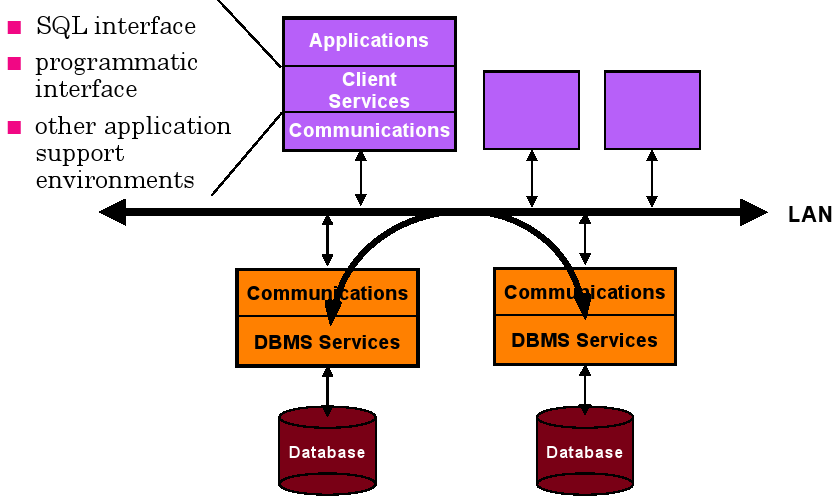
* One server, many clients



* Many servers, many clients



* Many servers, many clients



* Architecture defines the structure of the system. There are three ways to define the architecture: based on components, functions, or data
* DDBMS might be based on identical components (homogeneous systems) or different components (heterogeneous systems)
* ANSI/SPARC architecture defines external, conceptual, and internal schemas
* There are three orthogonal implementation dimensions for DDBMS: level of distribution, autonomity, and heterogeinity
* Different architectures are discussed:
  + Client-Server Systems
  + Peer-to-Peer Systems
  + Multi-DBMS