Distributed Systems

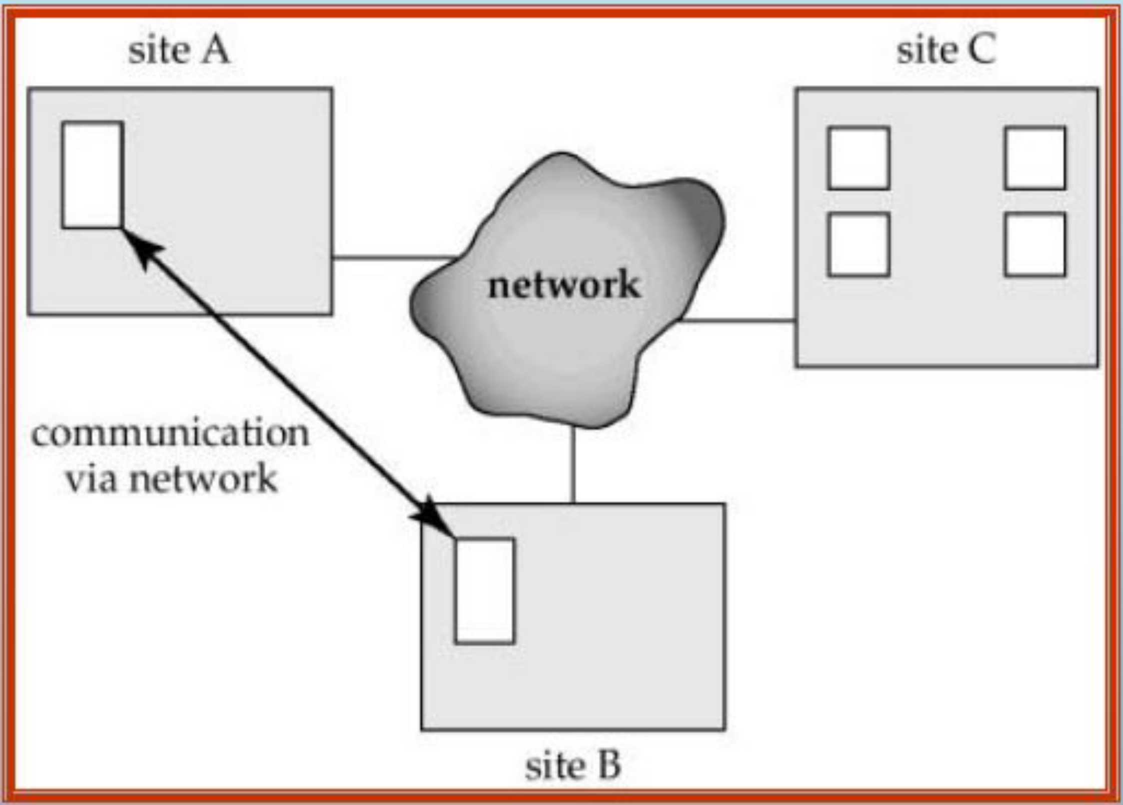
- In a distributed database system, the database is stored on several computers.
- The computers in a distributed system communicate with one another through various communication media, such as high-speed networks or telephone lines.
- They do not share main memory or disks. The computers in a distributed system may vary in size and function, ranging from workstations up to mainframe systems.

Distributed Systems

- The computers in a distributed system are referred to by a number of different names, such as sites or nodes, depending on the context in which they are mentioned.
- We mainly use the term site, to emphasize the physical distribution of these systems.

Distributed Systems

- Data spread over multiple machines (also referred to as **sites** or **nodes**.
- Network interconnects the machines
- Data shared by users on multiple machines



Need for Distributed Systems

- Distributed Nature of Organizational Units
- Need for Sharing of Data
- Support for Both OLTP and OLAP
- Database Recovery
- Support for Multiple Application Software

Distributed Databases

- Homogeneous distributed databases
  - Same software/schema on all sites, data may be partitioned among sites
  - Goal: provide a view of a single database, hiding details of distribution
- Heterogeneous distributed databases
  - Different software/schema on different sites
  - Goal: integrate existing databases to provide useful functionality
- Differentiate between *local* and *global* transactions
  - A local transaction accesses data in the *single* site at which the transaction was initiated.
  - A global transaction either accesses data in a site different from the one at which the transaction was initiated or accesses data in several different sites.

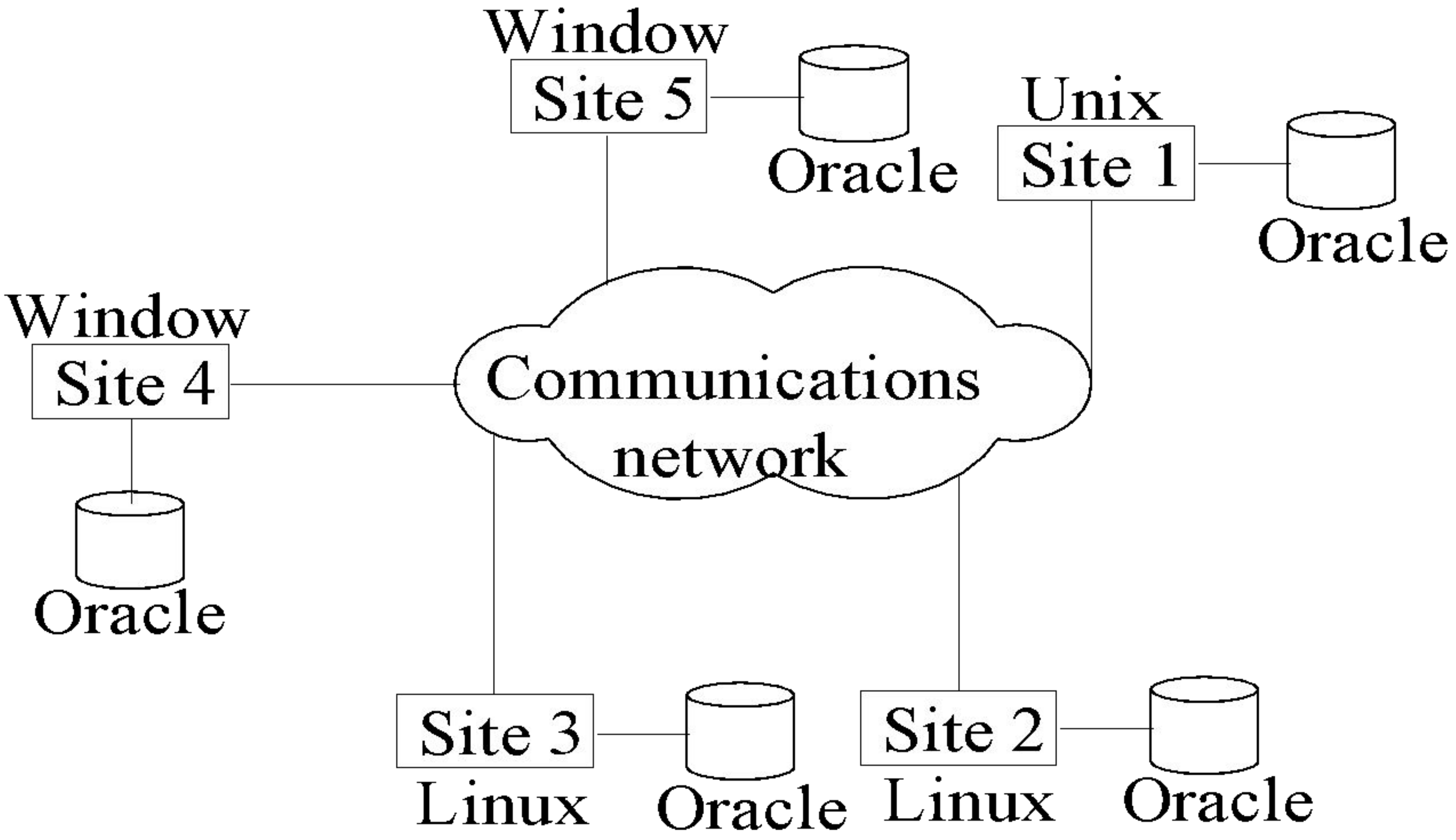
Distributed DBMS Architectures

- DDBMS architectures are generally developed depending on three parameters –
- **Distribution** – It states the physical distribution of data across the different sites.
- **Autonomy** – It indicates the distribution of control of the database system and the degree to which each constituent DBMS can operate independently.
- **Heterogeneity** – It refers to the uniformity or dissimilarity of the data models, system components and databases.

Homogeneous Distributed Database

- All sites of the database system have identical setup, i.e., same database system software.
- The underlying operating system may be different.
  - For example, all sites run Oracle or DB2, or Sybase or some other database system.
- The underlying operating systems can be a mixture of Linux, Window, Unix, etc.

Homogeneous Distributed Database



- In a homogeneous distributed database, all the sites use identical DBMS and operating systems. Its properties are −

❖ The sites use very similar software.

❖ The sites use identical DBMS or DBMS from the same vendor.

❖ Each site is aware of all other sites and cooperates with other sites to process user requests.

❖ The database is accessed through a single interface as if it is a single database.

Heterogeneous Distributed Database

In a heterogeneous distributed database, different sites have different operating systems, DBMS products and data models.

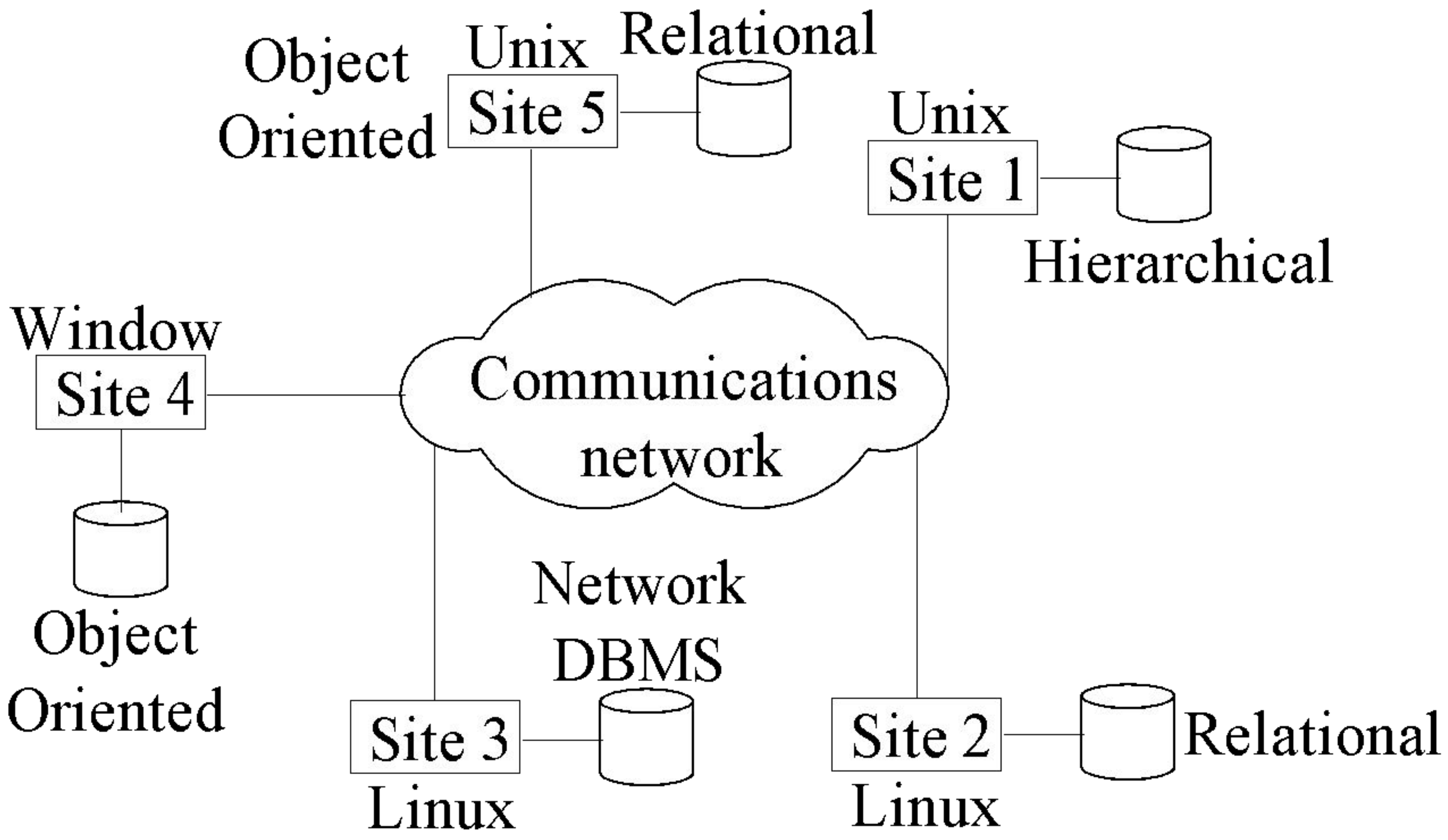
Its properties are −

- Different sites use dissimilar schemas and software.
- The system may be composed of a variety of DBMSs like relational, network, hierarchical or object oriented.
- Query processing is complex due to dissimilar schemas.
- Transaction processing is complex due to dissimilar software.
- A site may not be aware of other sites and so there is limited co-operation in processing user requests.

Heterogeneous Distributed Database

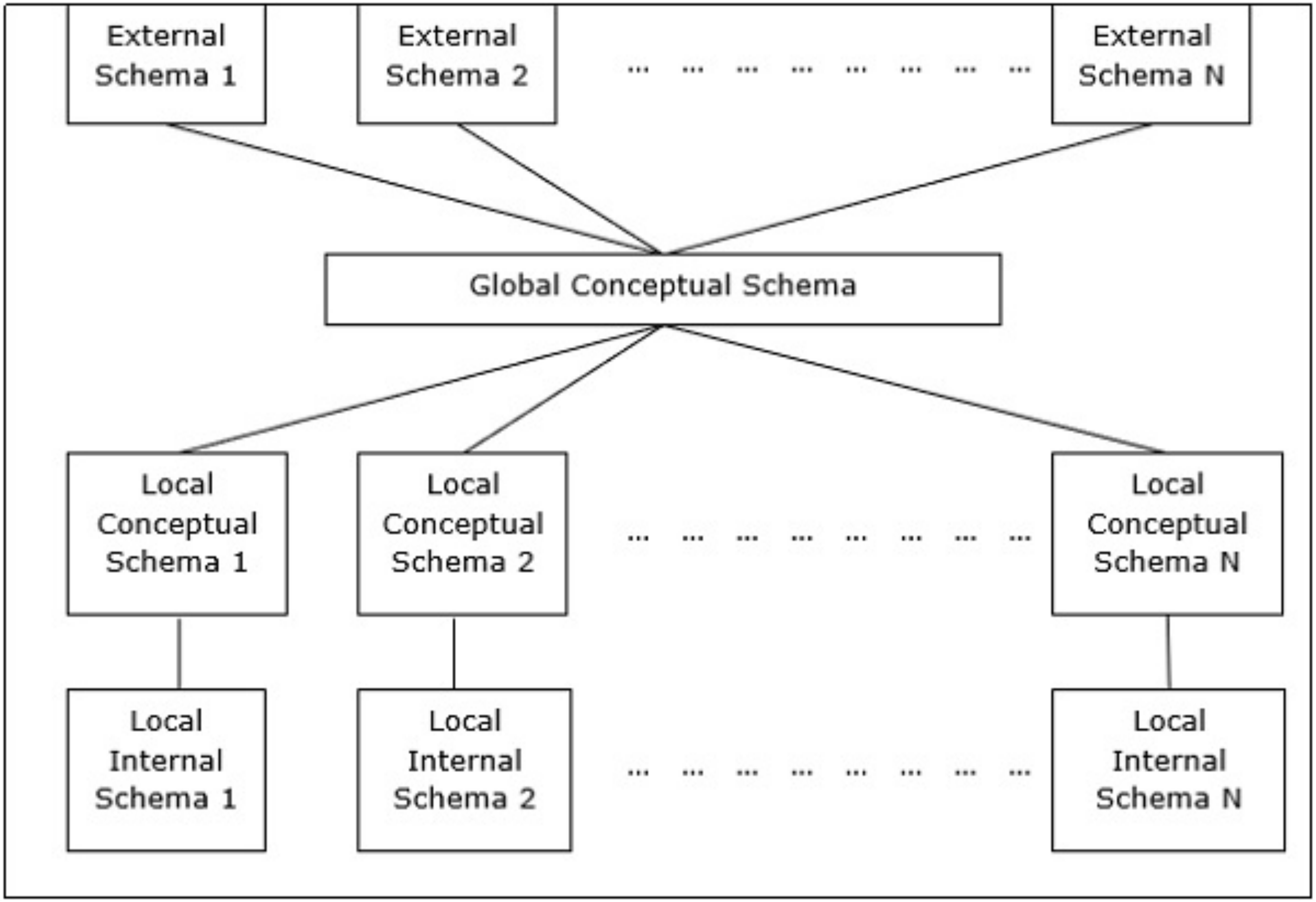
- **Federated:** Each site may run different database system but the data access is managed through a single conceptual schema. This implies that the degree of local autonomy is minimum.
- Each site must adhere to a centralized access policy. There may be a global schema.
- **Multidatabase:** There is no one conceptual global schema. For data access a schema is constructed dynamically as needed by the application software.

Heterogeneous Distributed Database



Peer- to-Peer Architecture for DDBMS

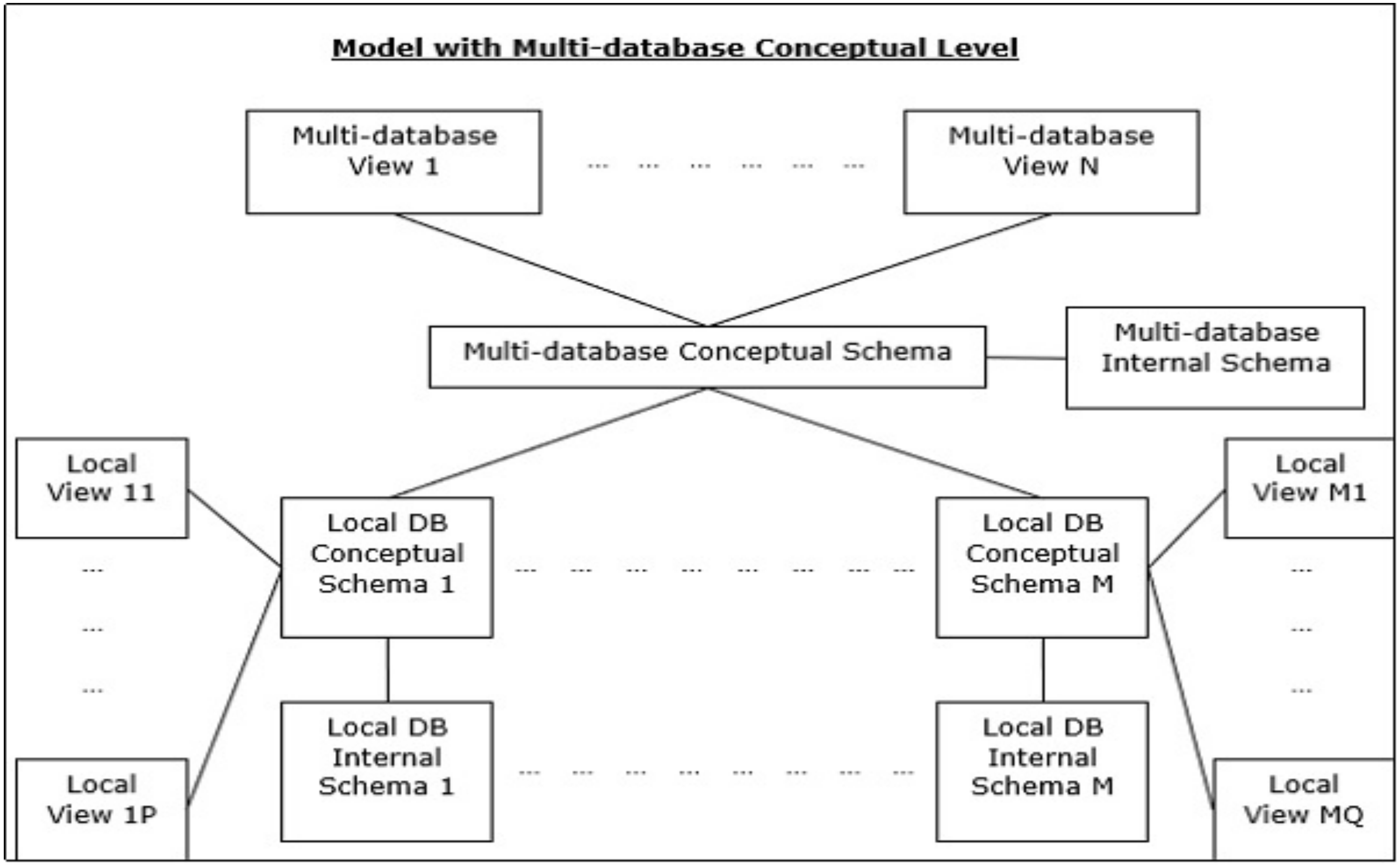
- In these systems, each peer acts both as a client and a server for imparting database services. The peers share their resource with other peers and co-ordinate their activities.
- This architecture generally has four levels of schemas −
- **Global Conceptual Schema** − Depicts the global logical view of data.
- **Local Conceptual Schema** − Depicts logical data organization at each site.
- **Local Internal Schema** − Depicts physical data organization at each site.
- **External Schema** − Depicts user view of data.


External Schema 1
External Schema 2
… … … … … … … …
External Schema N
Global Conceptual Schema
Local Conceptual Schema 1
Local Conceptual Schema 2
… … … … … … … …
Local Conceptual Schema N
Local Internal Schema 1
Local Internal Schema 2
… … … … … … … …
Local Internal Schema N

Multi - DBMS Architectures

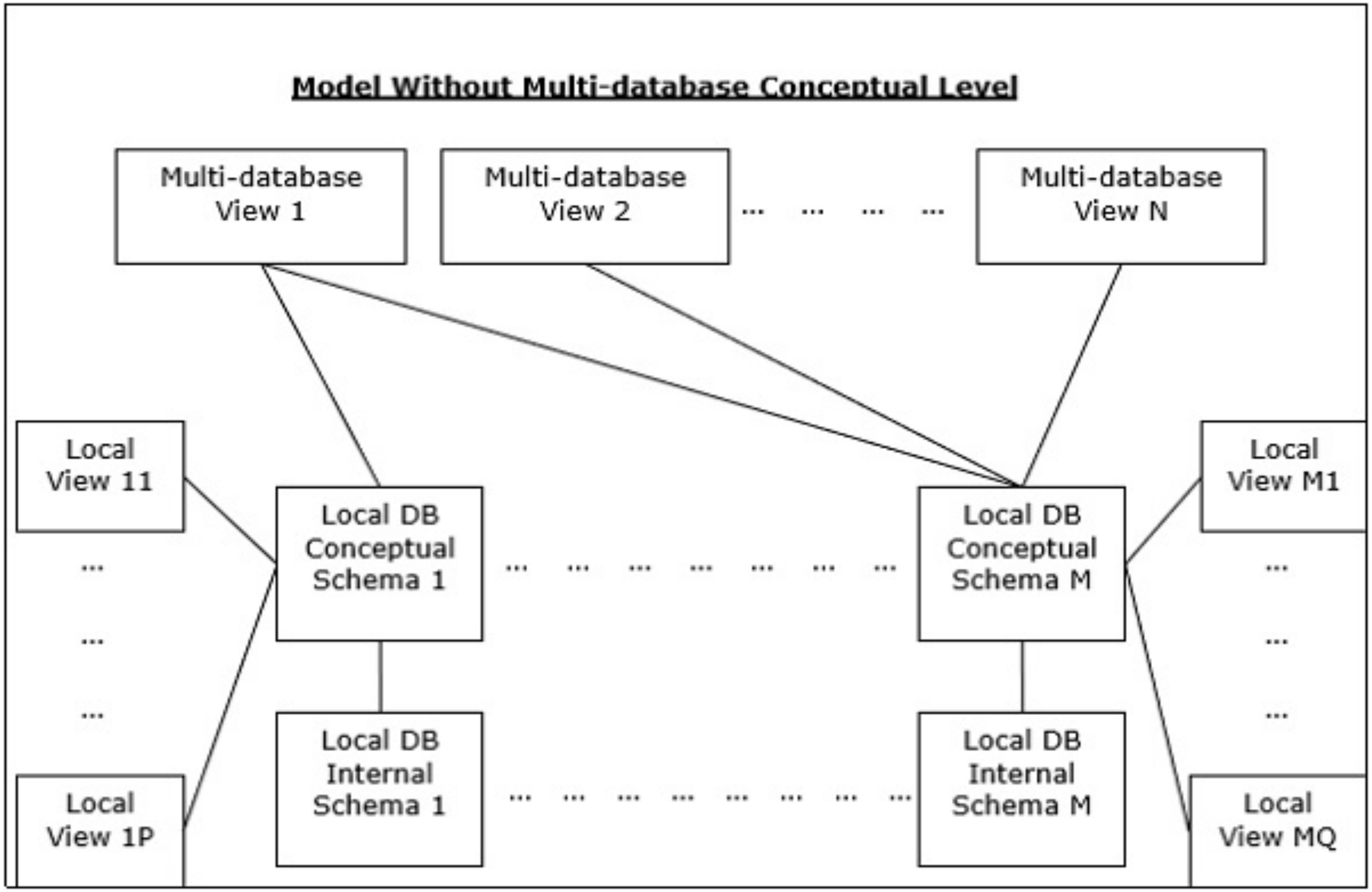
- There are two design alternatives for multi-DBMS –
- **Model with multi-database conceptual level.**
- **Model without multi-database conceptual level.**

Model with multi-database conceptual level



Model with Multi-database Conceptual Level
Multi-database View 1
Multi-database View N
Multi-database Conceptual Schema
Multi-database Internal Schema
Local View 11
Local View 1P
Local DB Conceptual Schema 1
Local DB Conceptual Schema M
Local DB Internal Schema 1
Local DB Internal Schema M
Local View M1
Local View MQ

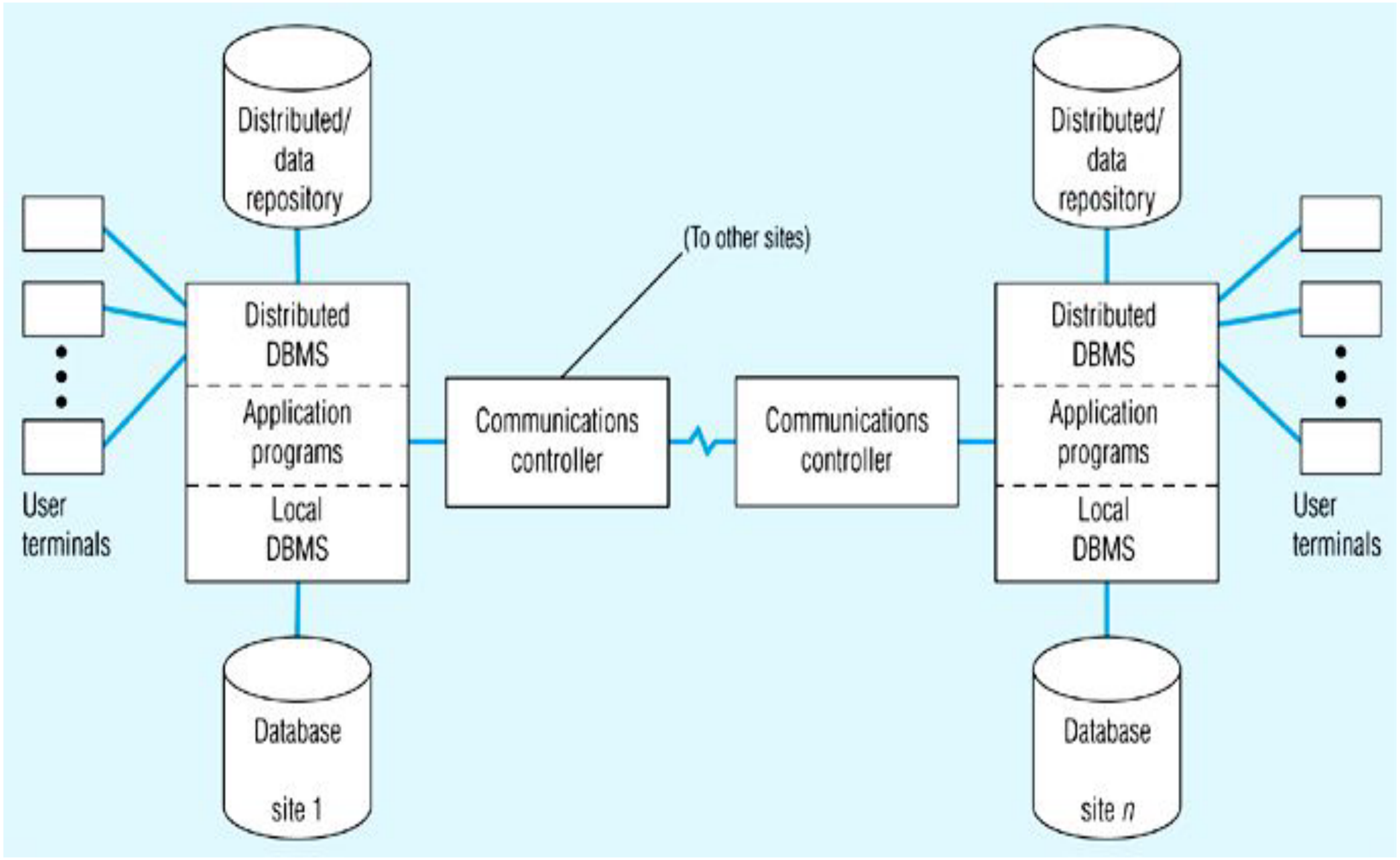
Model without multi-database conceptual level.


Model Without Multi-database Conceptual Level
Multi-database View 1
Multi-database View 2
Multi-database View N
Local View 11
Local View 1P
Local DB Conceptual Schema 1
Local DB Internal Schema 1
Local DB Conceptual Schema M
Local DB Internal Schema M
Local View M1
Local View MQ

Distributed DBMS

- **Distributed database** requires **distributed DBMS**
- Functions of a distributed DBMS:
  - Locate data with a *distributed data dictionary*
  - Determine location from which to retrieve data and process query components
  - DBMS translation between nodes with different local DBMSs (using *middleware*)
  - Data consistency (via *multiphase commit protocols*)
  - Global primary key control
  - Scalability
  - Security, concurrency, query optimization, failure

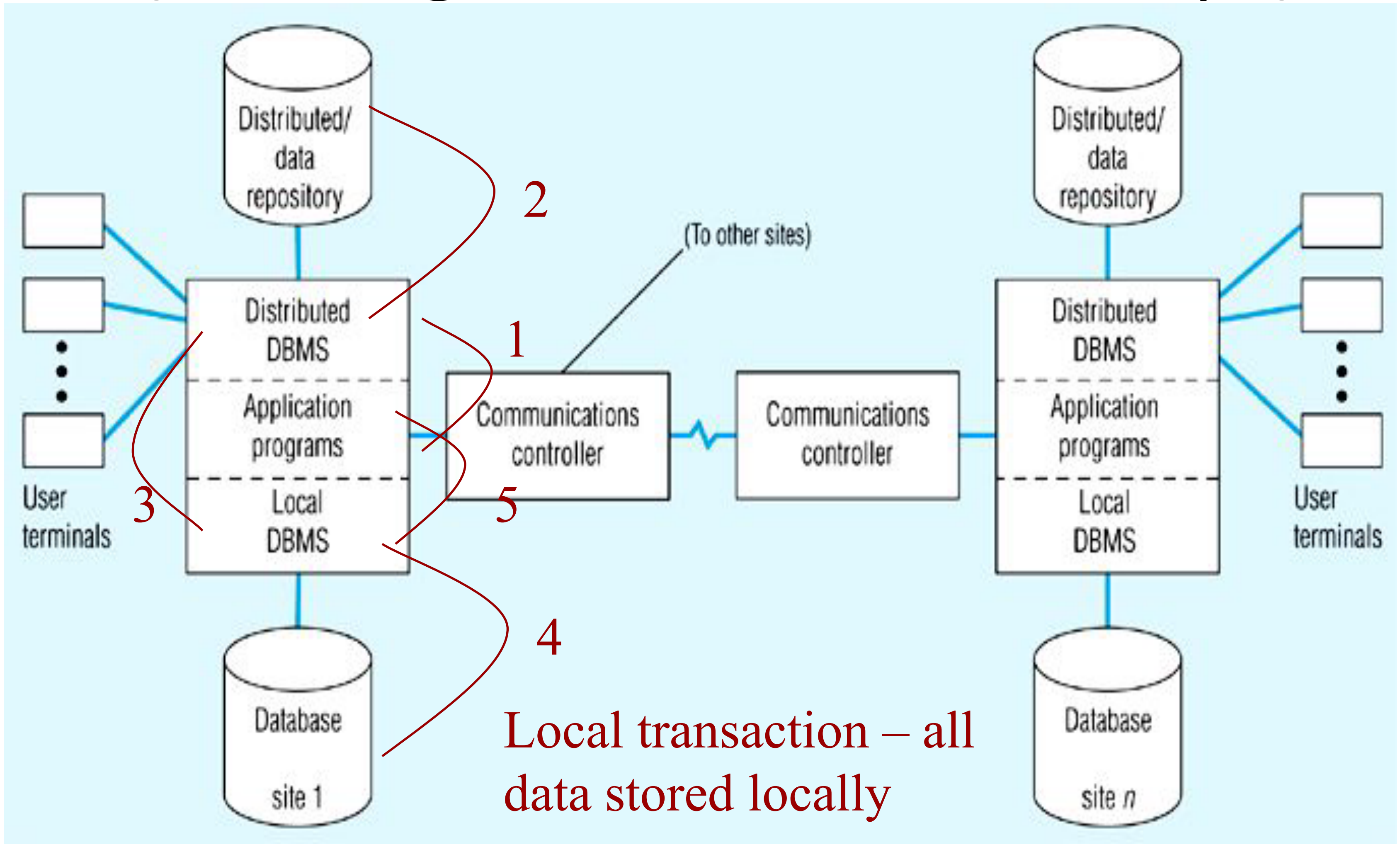
Distributed DBMS architecture



Local Transaction Steps

1. Application makes request to distributed DBMS
2. Distributed DBMS checks distributed data repository for location of data. Finds that it is local
3. Distributed DBMS sends request to local DBMS
4. Local DBMS processes request
5. Local DBMS sends results to application

Distributed DBMS Architecture (cont.) (showing local transaction steps)

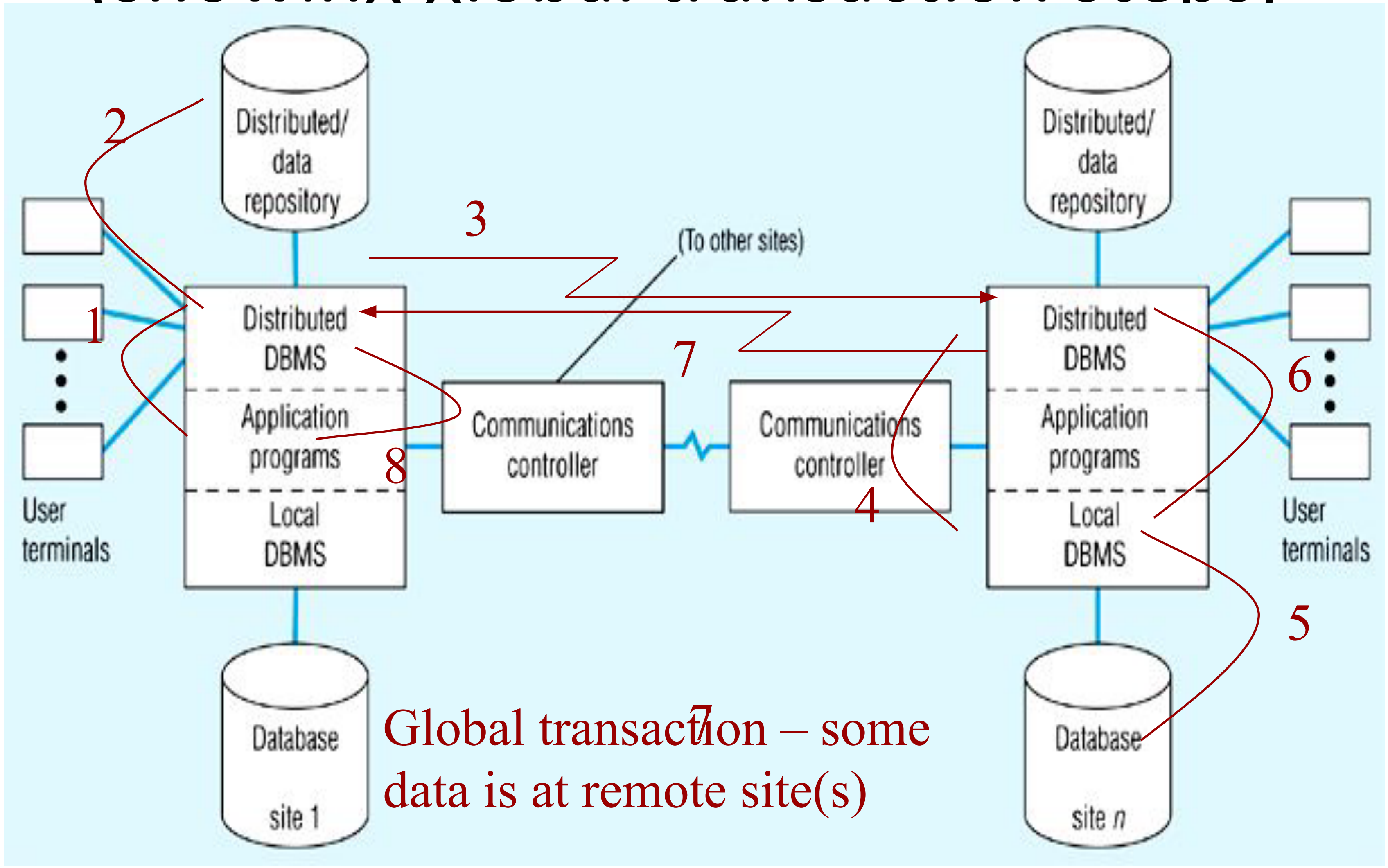


Local transaction – all data stored locally

Global Transaction Steps

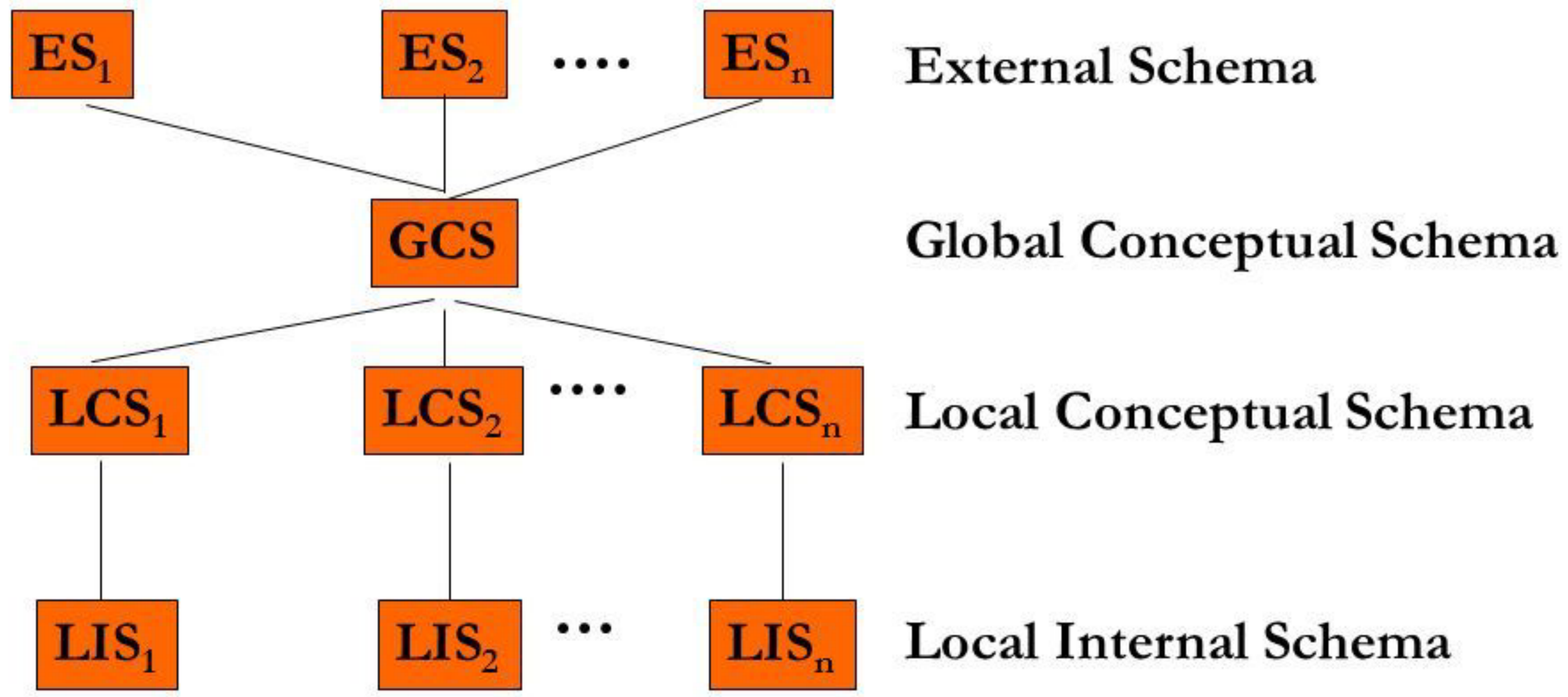
1. Application makes request to distributed DBMS
2. Distributed DBMS checks distributed data repository for location of data. Finds that it is **remote**
3. Distributed DBMS routes request to remote site
4. Distributed DBMS at remote site translates request for its local DBMS if necessary, and sends request to local DBMS
5. Local DBMS at remote site processes request
6. Local DBMS at remote site sends results to distributed DBMS at remote site
7. Remote distributed DBMS sends results back to originating site
8. Distributed DBMS at originating site sends results to application

Distributed DBMS architecture (cont.) (showing global transaction steps)



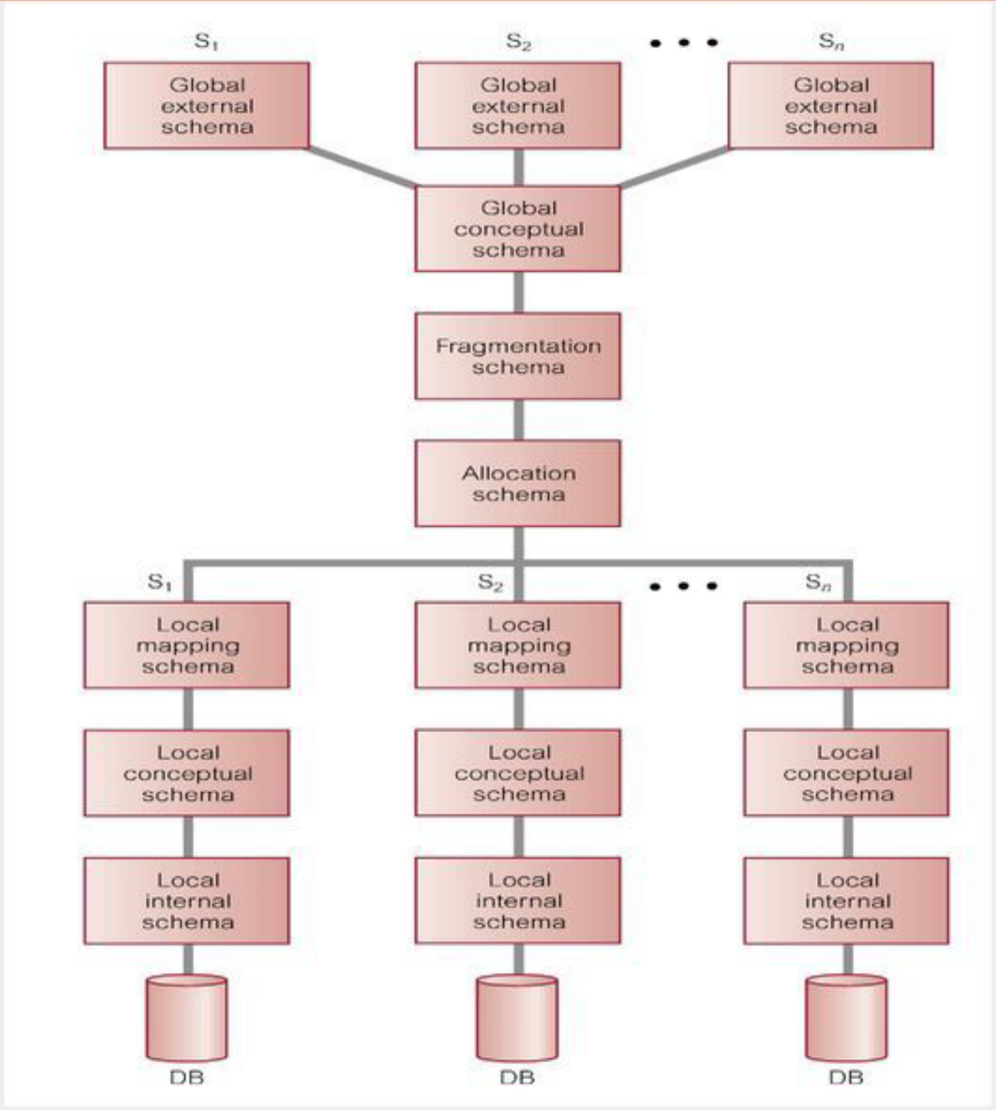
Global transaction – some data is at remote site(s)

Distributed Database Reference Architecture



It is logically integrated. Provides for the levels of transparency

Reference Architecture for DDBMS



Reference Architecture for a DDBMS

- The reference architecture shown in Figure consists of the following schemas:
  - a set of global external schemas;
  - a global conceptual schema;
  - a fragmentation schema and allocation schema;
  - a set of schemas for each local DBMS conforming to the ANSI-SPARC three-level architecture.

Global conceptual schema

- The global conceptual schema is a logical description of the whole database, as if it were not distributed.
- This level corresponds to the conceptual level of the ANSI-SPARC architecture and contains definitions of
  - entities,
  - relationships,
  - constraints,
  - Security, and
  - integrity information.
- It provides physical data independence from the distributed environment.
- The global external schemas provide logical data independence.

Fragmentation and allocation schemas

- The fragmentation schema is a description of how the data is to be logically partitioned.
- The allocation schema is a description of where the data is to be located, taking account of any replication.

Local schemas

- Each local DBMS has its own set of schemas.
- The local conceptual and local internal schemas correspond to the equivalent levels of the ANSI-SPARC architecture.
- The local mapping schema maps fragments in the allocation schema into external objects in the local database.
- It is DBMS independent and is the basis for supporting heterogeneous DBMSs.

Top-Down Design Process

- Suitable when a system is being designed from scratch
- A general method for designing centralized databases includes four phases:
  - Analyzing requirements
  - View integration and conceptual design
  - Data distribution design
  - Local physical schema design

Diagram of Top-Down Design Process

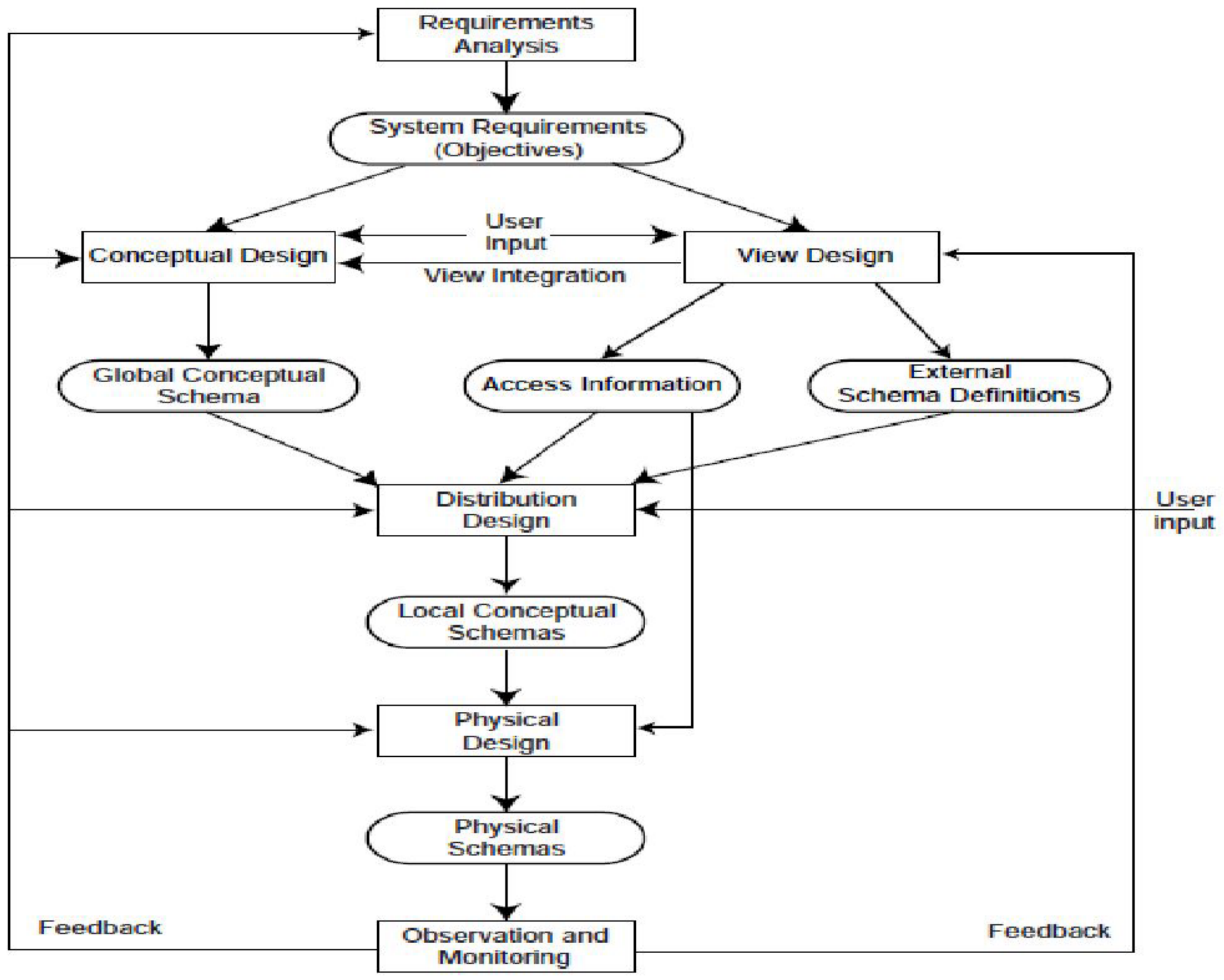


Fig. 3.2 Top-Down Design Process

- Requirement Analysis
  - Defines the environment of the system
  - Elicits both the data and processing needs of all potential DB users
- System Requirements
  - Where the final system is expected to stand?
  - Performance, Reliability, Availability, Economics, Flexibility
- The requirements document is input to two parallel activities:
  - view design
  - conceptual design

- Conceptual Design
  - Determines entity types and relationships among these Entities
  - Entity analysis:
    - determines the entities, attributes, and relationships
  - Functional analysis:
    - determines the fundamental functions with which the modeled enterprise is involved
  - The process is identical to the centralized database design
- View Design
  - Defines the interfaces for end users
  - The conceptual schema can be interpreted as being an integration of user views.

 The results of **these two steps** need to be cross-referenced to get a better understanding of which functions deal with which entities.

- In **conceptual design** and view design activities the user needs to specify the data entities and must determine the applications that will run on the database as well as statistical information about these applications.
- Statistical information includes the specification of the frequency of user applications, the volume of various information, and the like.

- The global conceptual schema (GCS) and access pattern information collected as a result of view design are inputs to the distribution design step.
- The objective at this stage, is to design the local conceptual schemas (LCSs) by distributing the entities over the sites of the distributed system.
- Rather than distributing relations, it is quite common to divide them into subrelations, called fragments, which are then distributed.

- **Distribution Design**
  - – Designs the local conceptual schema by distributing the entities over the sites of the distributed system
  - – Consists of two steps :
    - fragmentation and allocation

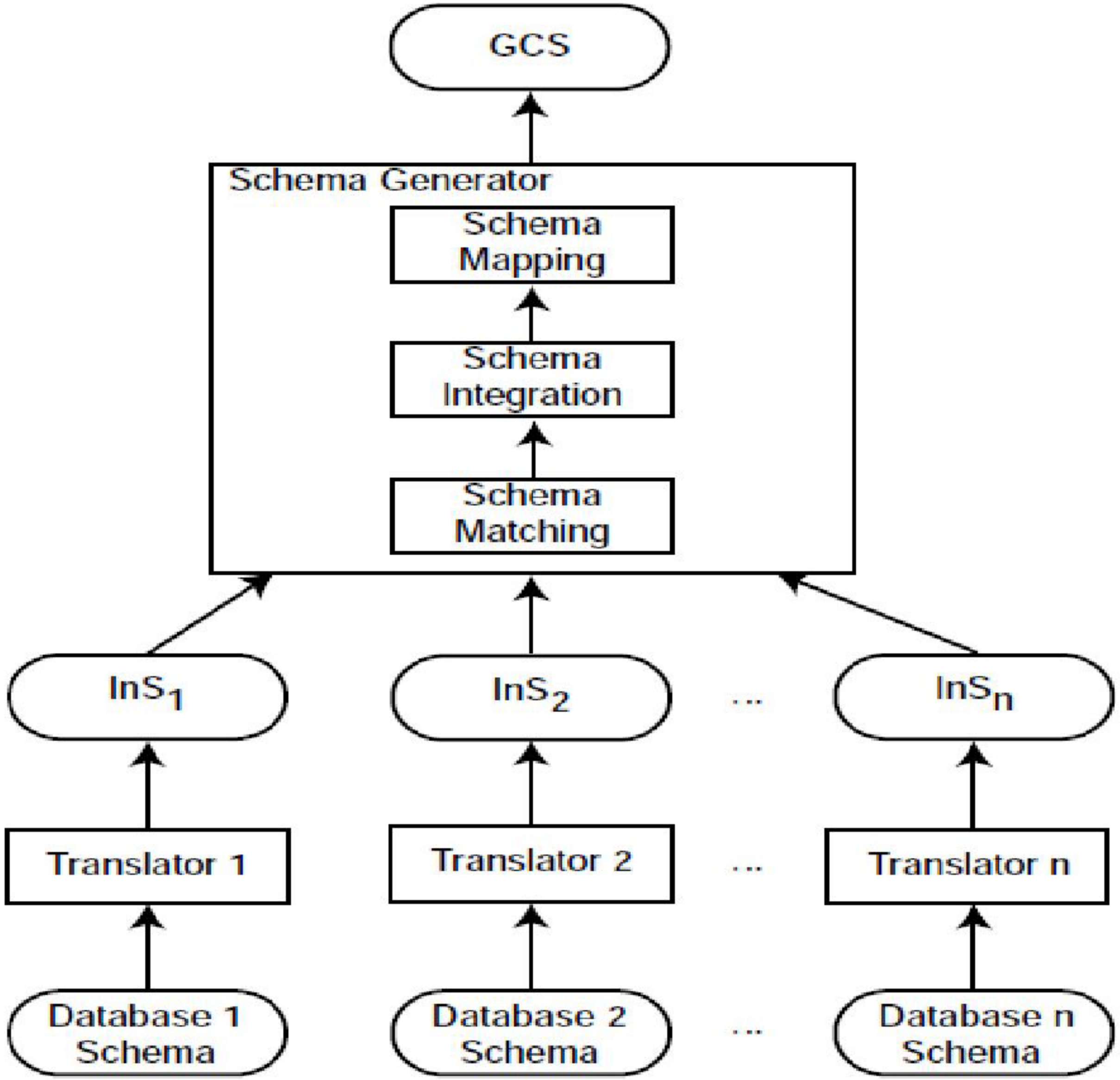
- **physical design**
  - – which maps the local conceptual schemas to the physical storage devices available at the corresponding sites.
  - – The inputs to this process are the local conceptual schema and the access pattern information about the fragments in them.

- **Observation and Monitoring**
- It is well known that design and development activity of any kind is an ongoing process requiring constant monitoring and periodic adjustment and tuning.
- We have therefore included observation and monitoring as a major activity in this process.
- Note that one does not monitor only the behavior of the database implementation but also the suitability of user views.
- The result is some form of feedback, which may result in backing up to one of the earlier steps in the design.

Bottom-Up Design

- Bottom-up design involves the process by which information from participating databases can be (physically or logically) integrated to form a single cohesive multidatabase.
- Suitable when many DBs exist, and the design task involves integrating them into one DB

Diagram of Bottom-Up Design Methodology



- Bottom-up design occurs in two general steps
  - schema translation (or simply translation)
  - schema generation

- The starting point of bottom-up design is the individual local conceptual schemas.
- The process consists of integrating local schemas into the global conceptual schema.

- In the first step, the component database schemas are translated to a common intermediate canonical representation (InS1; InS2; : : : ; InSn).
- The use of a canonical representation facilitates the translation process by reducing the number of translators that need to be written.
- Canonical representation may be
  - entity-relationship model
  - object-oriented model
  - graph
  - tree
- The choice of the canonical model is important.
- As a principle, it should be one that is sufficiently expressive to incorporate the concepts available in all the databases that will later be integrated.

- The **translation step** is necessary only if the component databases are heterogeneous and local schemas are defined using different data models.
- The translation step is delayed, providing increased flexibility for applications to access underlying data sources in a manner that is suitable for their needs.

- In the second step of bottom-up design, the intermediate schemas are used to generate a GCS.
- In some methodologies, local external (or export) schemas are considered for integration rather than full database schemas, to reflect the fact that local systems may only be willing to contribute some of their data to the multidatabase

- The schema generation process consists of the following steps:
    1. Schema matching to determine the syntactic and semantic correspondences among the translated LCS elements or between individual LCS elements and the pre-defined GCS elements.
    2. Integration of the common schema elements into a global conceptual (mediated) schema if one has not yet been defined.
    3. Schema mapping that determines how to map the elements of each LCS to the other elements of the GCS.

- It is also possible that the schema mapping step may be divided into two phases:
  - mapping constraint generation
  - transformation generation
- In the first phase, given correspondences between two schemas, a transformation function such as a query or view definition over the source schema is generated that would "populate" the target schema.
- In the second phase, an executable code is generated corresponding to this transformation function that would actually generate a target database consistent with these constraints.
- In some cases, the constraints are implicitly included in the correspondences, eliminating the need for the first phase.

Database Integration

- It involves the process by which information from participating databases can be conceptually integrated to form a single cohesive definition of a multidatabase
- It is the process of designing the global conceptual schema
- Database integration can occur in two steps
  - Schema translation
  - Schema integration

Distributed Relational Database Design

- Data Fragmentation,
- Replication and
- Allocation

Data Fragmentation

- A relation may be divided into a number of sub relations, called **fragments, which are then distributed.**

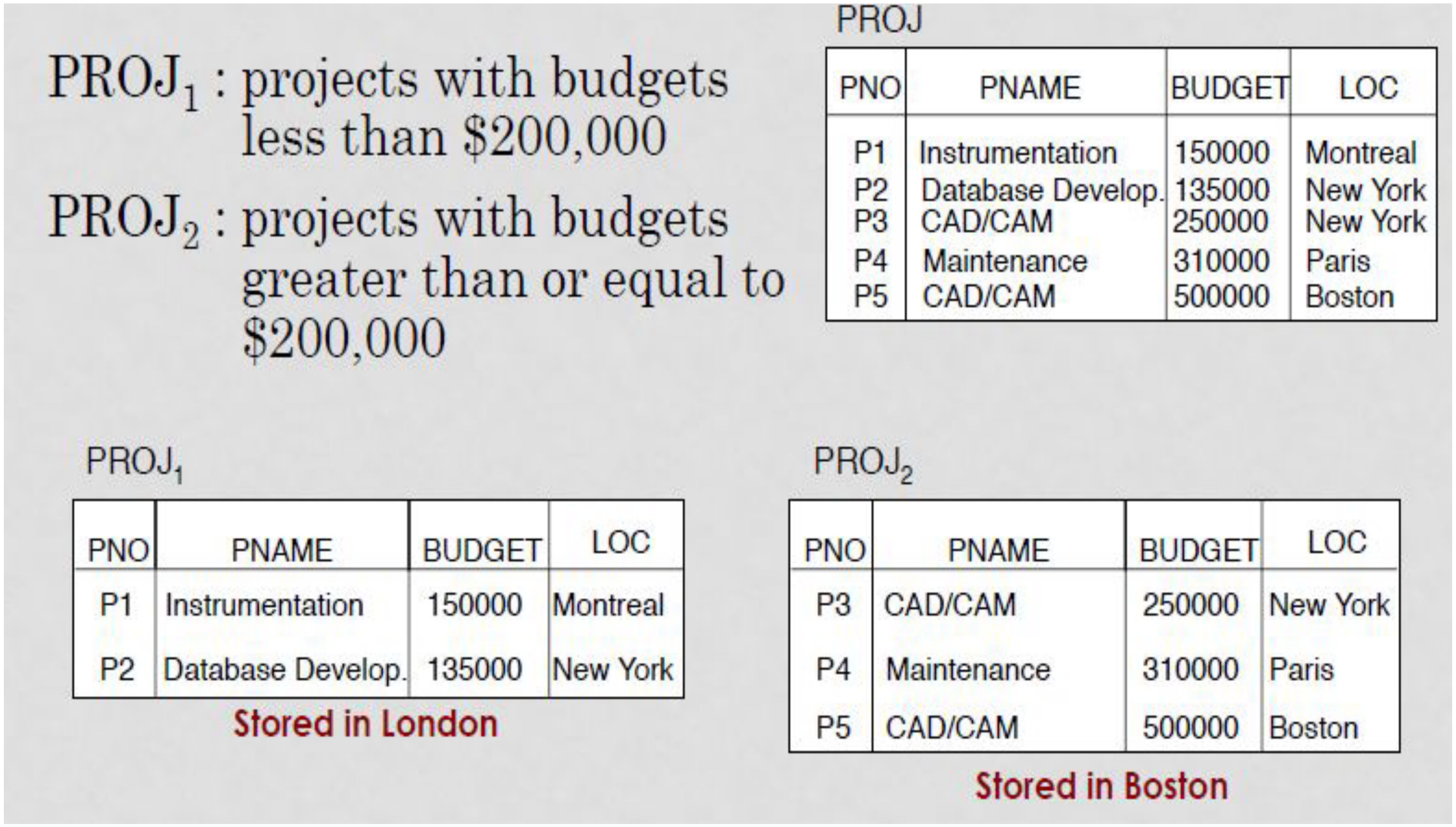
- **Fragmentation schema**
  - A definition of a set of fragments (horizontal or vertical or horizontal and vertical) that includes all attributes and tuples in the database that satisfies the condition that the whole database can be reconstructed from the fragments by applying some sequence of UNION (or OUTER JOIN) and UNION operations.

- There are two main types of fragmentation:
  - **Horizontal Fragmentation**: subsets of tuples
  - Vertical **Fragmentation :** subsets of attributes.

Data Fragmentation, Replication and Allocation

- **Horizontal fragmentation**
  - It is a horizontal subset of a relation which contain those of tuples which satisfy selection conditions.
  - Consider the Employee relation with selection condition
  - (DNO = 5).
  - All tuples satisfy this condition will create a subset which will be a horizontal fragment of Employee relation.
  - A selection condition may be composed of several conditions connected by AND or OR.
  - Derived horizontal fragmentation: It is the partitioning of a primary relation to other secondary relations which are related with Foreign keys.

HORIZONTAL



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

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

PROJ

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

$PROJ_1$

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

Stored in London

$PROJ_2$

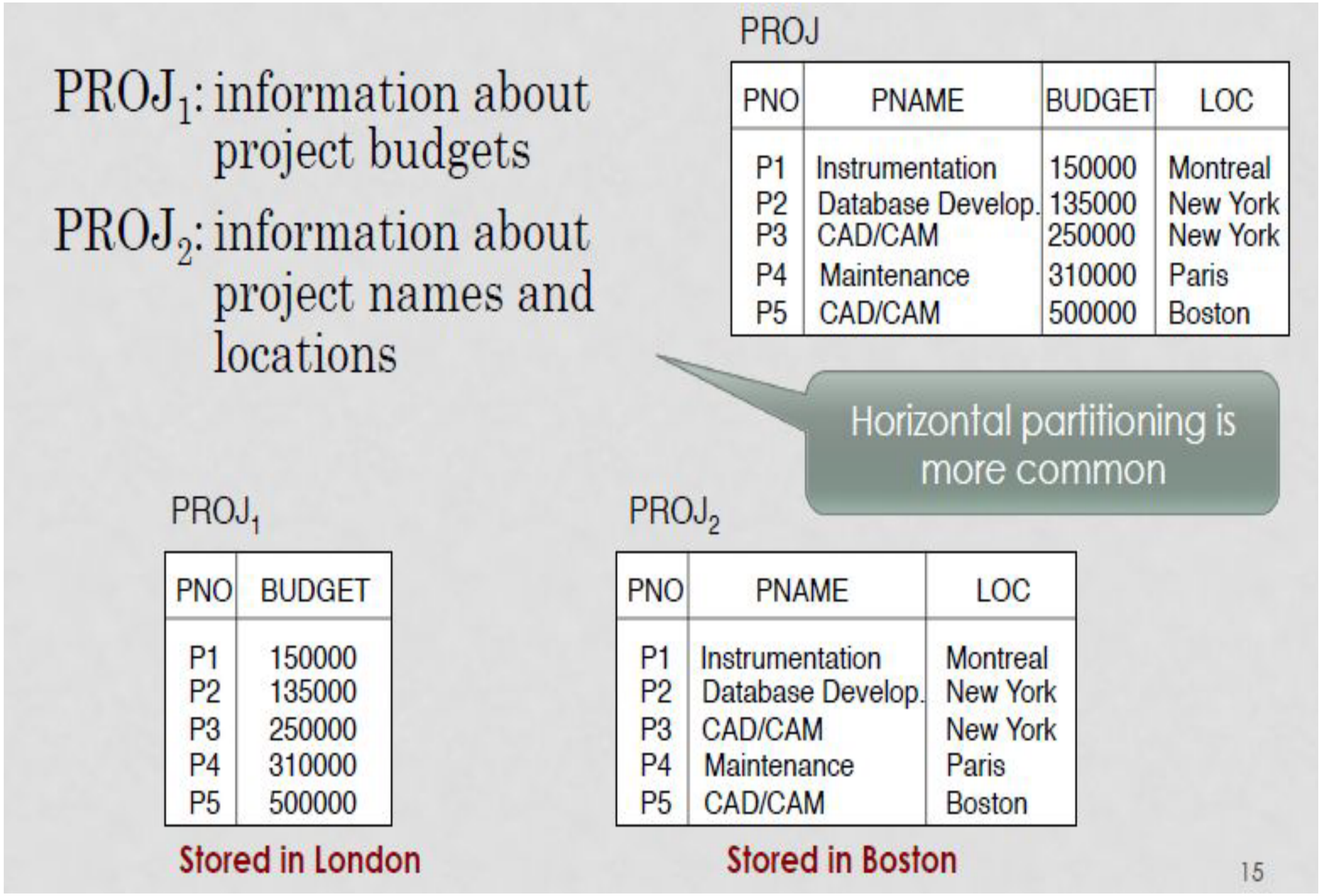
PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

Stored in Boston

Data Fragmentation, Replication and Allocation

- **Vertical fragmentation**
  - It is a subset of a relation which is created by a subset of columns. Thus a vertical fragment of a relation will contain values of selected columns. There is no selection condition used in vertical fragmentation.
  - Consider the Employee relation. A vertical fragment of can be created by keeping the values of Name, Bdate, Sex, and Address.
  - Because there is no condition for creating a vertical fragment, each fragment must include the primary key attribute of the parent relation Employee. In this way all vertical fragments of a relation are connected.

Vertical fragmentation



PROJ$_1$: information about project budgets

PROJ$_2$: information about project names and locations

PROJ

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

Horizontal partitioning is more common

PROJ$_1$

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

Stored in London

PROJ$_2$

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

Stored in Boston

15

Data Fragmentation, Replication and Allocation

Data Replication

A description of the replication of fragments is sometimes called a replication schema.

- Replication is useful in improving the availability of data.

- The most extreme case is replication of the whole database at every site in the distributed system, thus creating a fully replicated distributed database.

- This can improve availability remarkably because the system can continue to operate as long as at least one site is up.
-
- It also improves performance of retrieval for global queries, because the result of such a query can be obtained locally from anyone site; hence, a retrieval query can be processed at the local site where it is submitted, if that site includes a server module.

Data Replication conti…

- The disadvantage of full replication is that it can slow down update operations drastically, since a single logical update must be performed on every copy of the
- database to keep the copies consistent.
- This is especially true if many copies of the database exist.
- Full replication makes the concurrency control and recovery techniques more expensive than they would be if there were no replication.

Data Replication conti…

- The other extreme from full replication involves having no replication-that is, each fragment is stored at exactly one site.
- In this case all fragments must be disjoint except for the repetition of primary keys among vertical (or mixed) fragments.
- This is also called nonredundant allocation.

Data Replication conti…

- Between these two extremes, we have a wide spectrum of partial replication of the data-that is, some fragments of the database may be replicated whereas others may not.
- The number of copies of each fragment can range from one up to the total number of sites in the distributed system.
- A special case of partial replication is occurring heavily in applications where user carry partially replicated databases with them on laptops and personal digital assistants and synchronize them periodically with the server database.

Data Distribution (Data Allocation)

- **Allocation schema**
  - It describes the distribution of fragments to sites of distributed databases. It can be fully or partially replicated or can be partitioned.

- Each fragment-or each copy of a fragment-must be assigned to a particular site in the distributed system.
- This process is called data distribution (or data allocation).
- The choice of sites and the degree of replication depend on the performance and availability goals of the system and on the types and frequencies of transactions submitted at each site.
- For example, if high availability is required and transactions can be submitted at any site and if most transactions are retrieval only, a fully replicated database is a good choice.

Data Distribution (Data Allocation)

conti…

- However, if certain transactions that access particular parts of the database are mostly submitted at a particular site, the corresponding set of fragments can be allocated at that site only.
- Data that is accessed at multiple sites can be replicated at those sites.
- If many updates are performed, it may be useful to limit replication.
- Finding an optimal or even a good solution to distributed data allocation is a complex optimization problem

An Example of a Distributed Database

- Consider a banking system consisting of four branches in four different cities. Each branch has its own computer, with a database of all the accounts maintained at that branch. Each such installation is thus a site. There also exists one single site that maintains information about all the branches of the bank. Each branch maintains (among others) a relation account(Account-schema),

- **where Account-schema = (account-number, branch-name, balance)**
- **The site containing information about all the branches of the bank maintains the relation branch(Branch-schema), where**
- **Branch-schema = (branch-name, branch-city, assets)**

Concurrency control

- Since a transaction may access data items at several sites, transaction managers at several sites may need to coordinate to implement concurrency control.
- If locking is used (as is almost always the case in practice), locking can be performed locally at the sites containing accessed data items, but there is also a possibility of deadlock involving transactions originating at multiple sites.

- Therefore deadlock detection needs to be carried out across multiple sites.
- Failures are more common in distributed systems since not only may computers fail, but communication links may also fail. Replication of data items, which is the key to the continued functioning of distributed databases when failures occur, further complicates concurrency control.

- The primary disadvantage of distributed database systems is the added complexity required to ensure proper coordination among the sites. This increased complexity takes various forms:

Distributed Two-phase Locking Algorithm

- The basic principle of distributed two-phase locking is same as the basic two-phase locking protocol. However, in a distributed system there are sites designated as lock managers. A lock manager controls lock acquisition requests from transaction monitors. In order to enforce co-ordination between the lock managers in various sites, at least one site is given the authority to see all transactions and detect lock conflicts.

- Depending upon the number of sites who can detect lock conflicts, distributed two-phase locking approaches can be of three types −
- **Centralized two-phase locking** − In this approach, one site is designated as the central lock manager. All the sites in the environment know the location of the central lock manager and obtain lock from it during transactions.
- **Primary copy two-phase locking** − In this approach, a number of sites are designated as lock control centers. Each of these sites has the responsibility of managing a defined set of locks. All the sites know which lock control center is responsible for managing lock of which data table/fragment item.
- **Distributed two-phase locking** − In this approach, there are a number of lock managers, where each lock manager controls locks of data items stored at its local site. The location of the lock manager is based upon data distribution and replication.

Distributed Timestamp Concurrency Control

- In a centralized system, timestamp of any transaction is determined by the physical clock reading. But, in a distributed system, any site's local physical/logical clock readings cannot be used as global timestamps, since they are not globally unique. So, a timestamp comprises of a combination of site ID and that site's clock reading.
- For implementing timestamp ordering algorithms, each site has a scheduler that maintains a separate queue for each transaction manager. During transaction, a transaction manager sends a lock request to the site's scheduler. The scheduler puts the request to the corresponding queue in increasing timestamp order. Requests are processed from the front of the queues in the order of their timestamps, i.e. the oldest first.

Distributed Optimistic Concurrency Control Algorithm

- Distributed optimistic concurrency control algorithm extends optimistic concurrency control algorithm. For this extension, two rules are applied –
- **Rule 1** – According to this rule, a transaction must be validated locally at all sites when it executes. If a transaction is found to be invalid at any site, it is aborted. Local validation guarantees that the transaction maintains serializability at the sites where it has been executed. After a transaction passes local validation test, it is globally validated.

- **Rule 2** – According to this rule, after a transaction passes local validation test, it should be globally validated. Global validation ensures that if two conflicting transactions run together at more than one site, they should commit in the same relative order at all the sites they run together. This may require a transaction to wait for the other conflicting transaction, after validation before commit. This requirement makes the algorithm less optimistic since a transaction may not be able to commit as soon as it is validated at a site.