DATABASE MANAGEMENT SYSTEM

Chapter 9
Distributed Architecture

Bikash Khadka Shah

MCSE, MSDA, OCP, RHCE, RHCVA, CCNA, CEH

9841766620 | 9801076620

Background

- A geographically dispersed organization may choose the from the following options to store its databases on:
  - Central computer (centralized system)
  - Several computers (on different locations)
  - Combination of both the above

- The second option relates to the concept of distributed databases…

What is a Distributed Database?

- It is a single logical DB that is spread physically across computers in multiple locations that are connected by a data communications network.
- The comm. network must allow the users to share data.
- The sites of a distributed system may be spread over a large (country wide/world wide) or a small (building/campus) area.
- It is centrally administered as a corporate resource while providing local flexibility and customization .

What is a Distributed Database?

- A distributed database allows faster local queries and can reduce network traffic. With these benefits comes the issue of maintaining data integrity.
- A key objective for a distributed system is that it looks like a centralized system to the user.
- The user should not need to know where a piece of data is stored physically.

Distributed and Decentralized DBs are Different !

- A decentralized database is stored on computers at multiple locations.
- However, the computers are not interconnected by network and database software that make the data appear to be in one logical database.
- Thus users at various sites cannot share data.
- A decentralized database is best regarded as a collection of independent databases.

Things that encourage use of Distributed Databases…

- Distribution and autonomy of business units.
  - The geographically distant business units of an organization can have autonomy for creating and managing their own info systems.
- Data sharing.
  - Data can be shared/consolidated across local databases on demand.
- Data communication costs and reliability.
  - DDB save from the cost of transferring data over communication network by keeping the data close to where it is needed. Local data fragments are reliable for rapid data access.

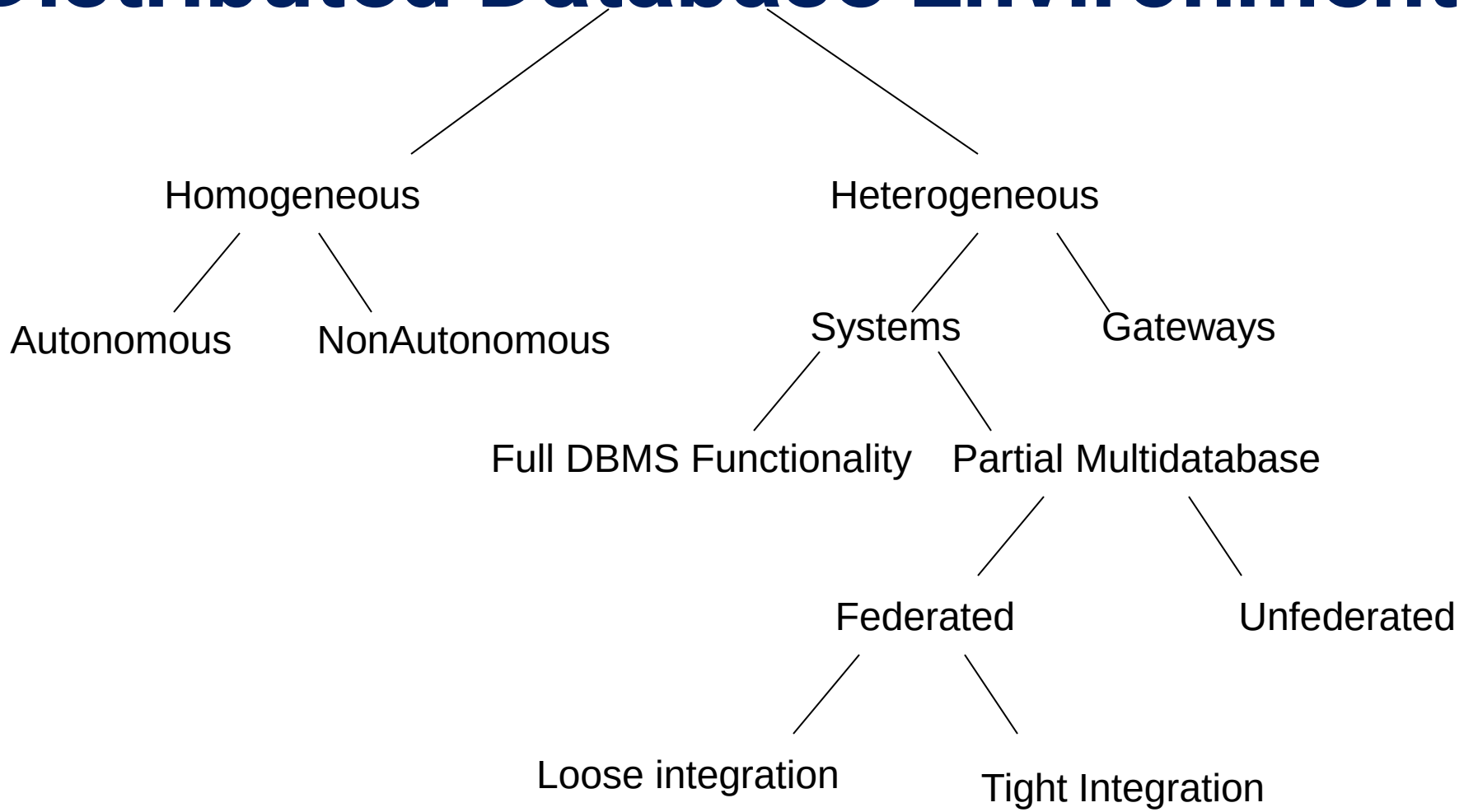
Things that encourage use of Distributed Databases…

- Multiple application vendor environment
  - Today several different vendors are producing prepackaged applications each designed to work with its own database and DBMS. DDB can provide functionality across these separate applications.
- Database recovery.
  - Replicating data across multiple sites is a natural form of distributed database. While primary site is being restored, users can access data from another site.
- Satisfying both transaction and analytical processing.
  - DDB technology can help in synchronizing data across OLTP and OLAP platforms.

Distributed Database Environments

- A distributed database requires multiple DBMSs, running at each remote site.
- The different types of distributed database environments are distinguished by:
  - The degree to which these different DBMSs cooperate and
  - whether there is a master site that coordinates requests involving data from multiple sites

Distributed Database Environments



Distributed Database Environments

- *Homogeneous*: same DBMS used at each node
  - Autonomous
    - Each DBMS works independently, passing messages back and forth to share data updates.
  - Nonautonomous
    - A central, or master, DBMS coordinates database access and update across the nodes.

Distributed Database Environments

- *Heterogeneous*: Potentially different DBMSs are used at each node.
  - **Systems:** Support all or some functionality of one logical database
    - **Full DBMS Functionality:** supports all functionality of DDB
    - **Partial Multi-database:** supports some features of DDB
      - Federated: supports local databases for unique data requests *
        - Loose Integration: many schemas exit for each local database& each local DBMS must communicate with all local schema
      - Tight Integration: One global schema exists that defines all the data across all local databases.
        Unfederated: requires all access to go through a central coordinating module
- **Gateways:** Simple paths are created to other databases, without the benefits of one logical database.

Objectives and Trade-offs

- Objectives
  - Location Transparency
    - User is unaware about the distribution of data and all data in the network appear as a single logical data base stored at one site.
    - so a user / user program using data need not know the location of data.
    - Any user request to retrieve or update data from any site is automatically forwarded by the system to the site/sites related to the processing request.
  - Local Autonomy
    - There is no reliance on central site. Data are locally owned and managed even though they are accessible from remote sites.
    - It is the capability of a site to independently administer and operate its database when connections to other nodes have failed.
    - Each site can control data , administer security, log transactions and recover when local failures occur.

Objectives and Trade-offs

- Trade-offs :
  - a significant trade-off in designing a distributed database environment is to whether to use synchronous or asynchronous distrbuted technology…
    - Synchronous distributed database
    - Synchronous distributed database

ARCHITECTURAL MODELS FOR DISTRIBUTED DBMSs - DISTRIBUTION

Distributions refers to the distributions of data. Of course, we are considering the physical distribution of data over multiple sites; the user sees the data as one logical pool.

Two alternatives:

- *client / server distribution*
- *peer-to-peer distribution (full distribution)*

ARCHITECTURAL MODELS FOR DISTRIBUTED DBMSs - DISTRIBUTION

Client / server distribution.

The client / server distribution concentrates data management duties at servers while the 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 the first attempt at distributing functionality.

Peer-to-peer 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.

ARCHITECTURAL MODELS FOR DISTRIBUTED DBMSs - HETEROGENEITY

Heterogeneity may occur in various forms in distributed systems, ranging form hardware heterogeneity and differences in networking protocols to variations in data managers.

Representing data with different modeling tools creates heterogeneity because of the inherent expressive powers and limitations of individual data models. Heterogeneity in query languages not only involves the use of completely different data access paradigms in different data models, but also covers differences in languages even when the individual systems use the same data model.

ARCHITECTURAL MODELS FOR DISTRIBUTED DBMSs - HETEROGENEITY

- Various levels (hardware, communications, operating system)
  - DBMS important one – data model, query language, transaction management algorithms
  - Representing data with different modeling tools creates heterogeneity because of the inherent expressive power and limitations of individual data models.

ARCHITECTURAL MODELS FOR DISTRIBUTED DBMSs - ALTERNATIVES

The dimensions are identified as:

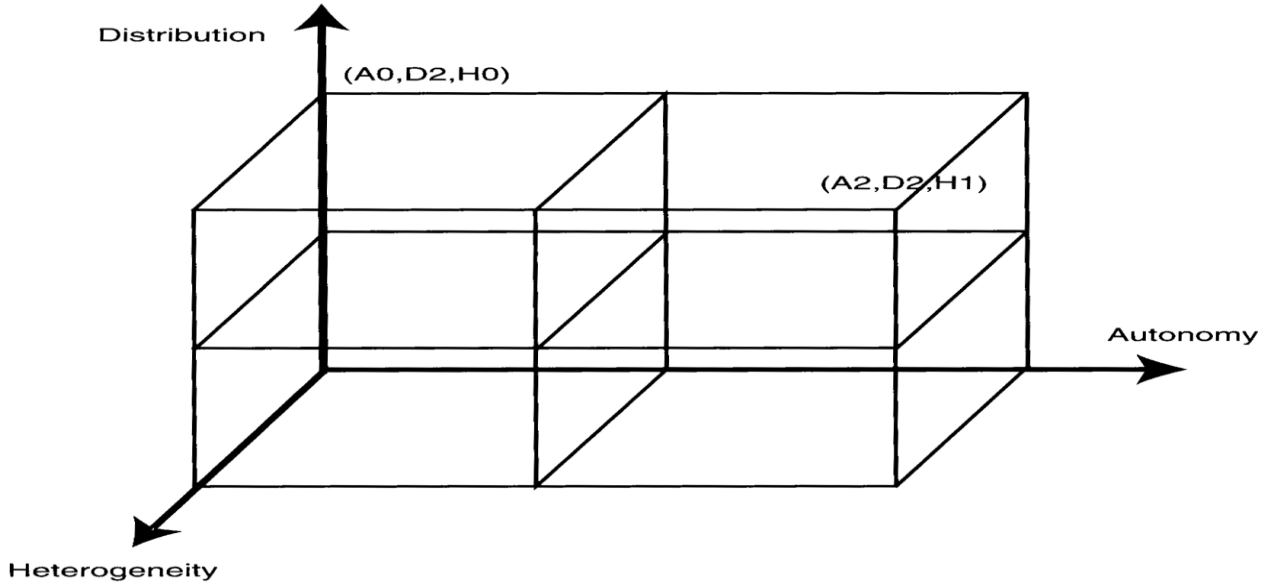
A (autonomy),

D (distribution) and

H (heterogeneity).

The alternatives along each dimension are identified by numbers as: 0, 1 or 2.

ARCHITECTURAL MODELS FOR DISTRIBUTED DBMSs



DBMS Implementation Alternatives

The systems are characterized with respect to:

(1) the autonomy of the local systems,

(2) their distribution,

(3) their heterogeneity.

ARCHITECTURAL MODELS FOR DISTRIBUTED DBMSs - ALTERNATIVES

A0 - tight integration
A1 - semiautonomous systems
A2 - total isolation

H0 - homogeneous systems
H1 - heterogeneous systems

D0 - no distribution
D1 - client / server systems
D2 - peer-to-peer systems

Autonomy

- Distribution of control (and not data) - the degree of independence
  - The local operations of the individual DBMSs are not affected by their participation in the multidatabase system
  - The manner in which individual DBMSs process queries and optimize them should not be affected by the execution of global queries
  - System consistency should not be compromised when individual DBMSs join or leave the multidatabase system

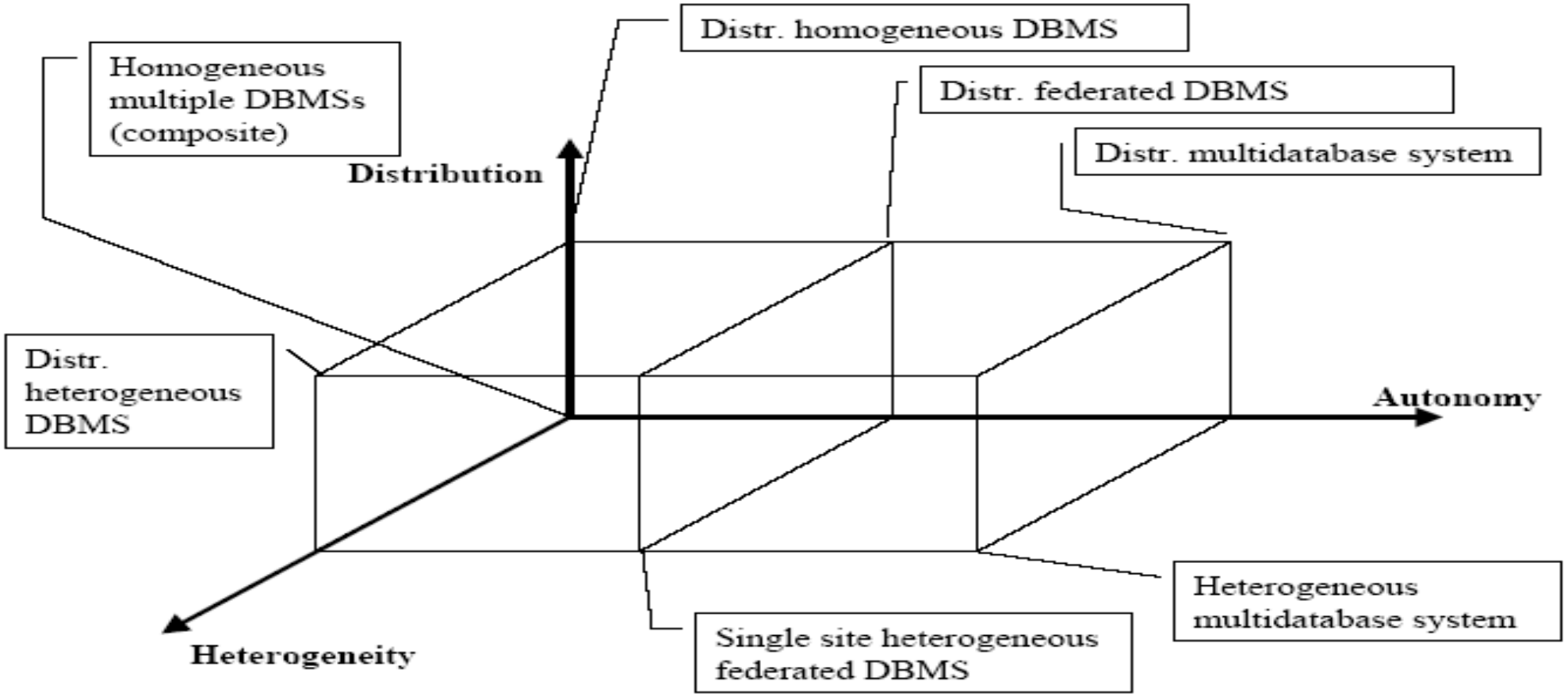
Autonomy

- On the other hand specifies the dimension of autonomy as:
  - 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.

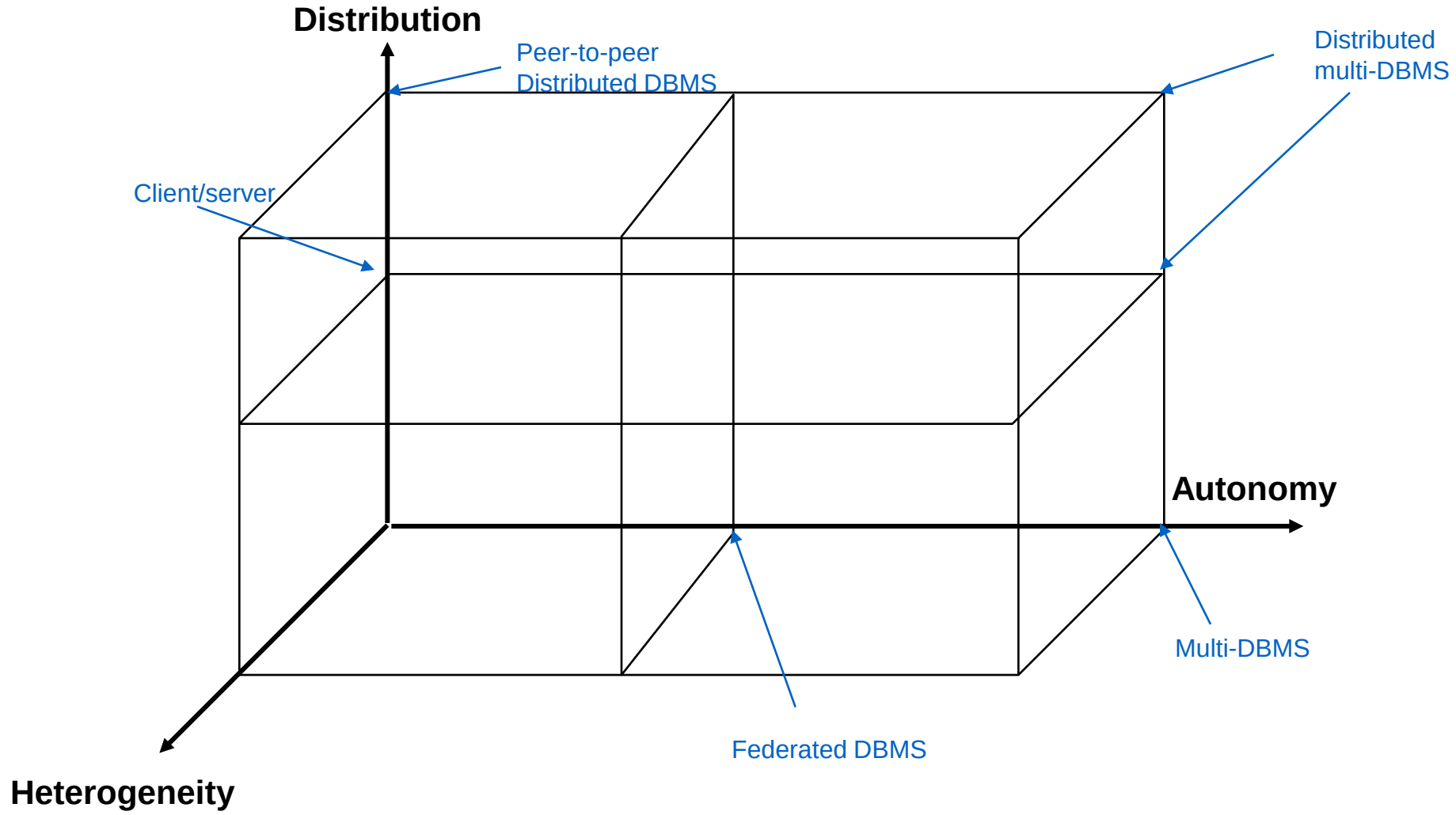
Autonomy

- Possibilities:
  - 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.
  - Semiautonomous system – consist of DBMSs that can operate independently, but have decided to participate in a federation to make their local data sharable.
  - Total isolation – the individual systems are stand-alone DBMSs, which know neither of the existence of other DBMSs nor how to communicate with them.

Architectural models for Distributed DBMSs



Alternatives in Distributed Database Systems



ARCHITECTURAL MODELS FOR DISTRIBUTED DBMSs - ALTERNATIVES

- In figure 4.3 , two alternative architectures that are focus of this book:
- (A0, D2, H0)
- (A2, D2, H1)
- Not all the architectures that are identified by this design space are meaningful.

ARCHITECTURAL MODELS FOR DISTRIBUTED DBMSs - ALTERNATIVES

(A0, D0, H0)

If there is no distribution or heterogeneity, the system is a set of multiple DBMSs that are logically integrated. Such systems can be given generic name composite systems. Not such examples but they may be suitable for shared everything multiprocessor systems.

(A0, D0, H1)

If heterogeneity is introduced, one has multiple data managers that are heterogeneous but provide an integrated view to the user.

(A0, D1, H0)

The more interesting case is where the database is distributed even though an integrated view of the data is provided to users (client / server distribution). Mentioned earlier and will discuss further.

ARCHITECTURAL MODELS FOR DISTRIBUTED DBMSs - ALTERNATIVES

(A0, D2, H0)

The same type of transparency is provided to the user in a fully distributed environment. There is no distinction among clients and servers, each site providing identical functionality.

(A1, D0, H0)

These are semiautonomous systems, which are commonly termed *federated DBMS*. The component systems in a federated environment have significant autonomy in their execution, but their participation in the federation indicate that they are willing to cooperate with other in executing user requests that access multiple databases. An example may be multiple installations of an DBMS.

ARCHITECTURAL MODELS FOR DISTRIBUTED DBMSs - ALTERNATIVES

(A1, D0, H1)

These are systems that introduce heterogeneity as well as autonomy, what we might call a *heterogeneous federated DBMS*.

(A1, D1, H1)

System of this type introduce distribution by placing component systems on different machines. They may be referred to as *distributed, heterogeneous federated DBMS*.

(A2, D0, H0)

Now we have full autonomy. These are *multidatabase systems (MDBS)*. The components have no concept of cooperation. Without heterogeneity and distribution, an MDBS is an interconnected collection of autonomous databases.

ARCHITECTURAL MODELS FOR DISTRIBUTED DBMSs - ALTERNATIVES

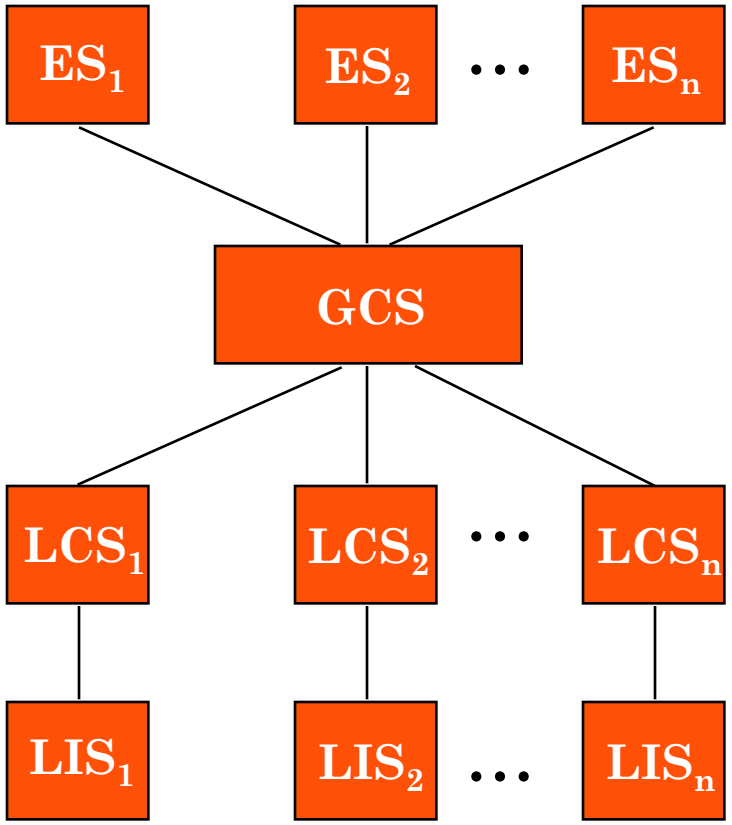
(A2, D0, H1)

These case is realistic, maybe even more so than (A1, D0, H1), in that we always want to built applications which access data from multiple storage systems with different characteristics.

(A2, D1, H1) and (A2, D2, H1)

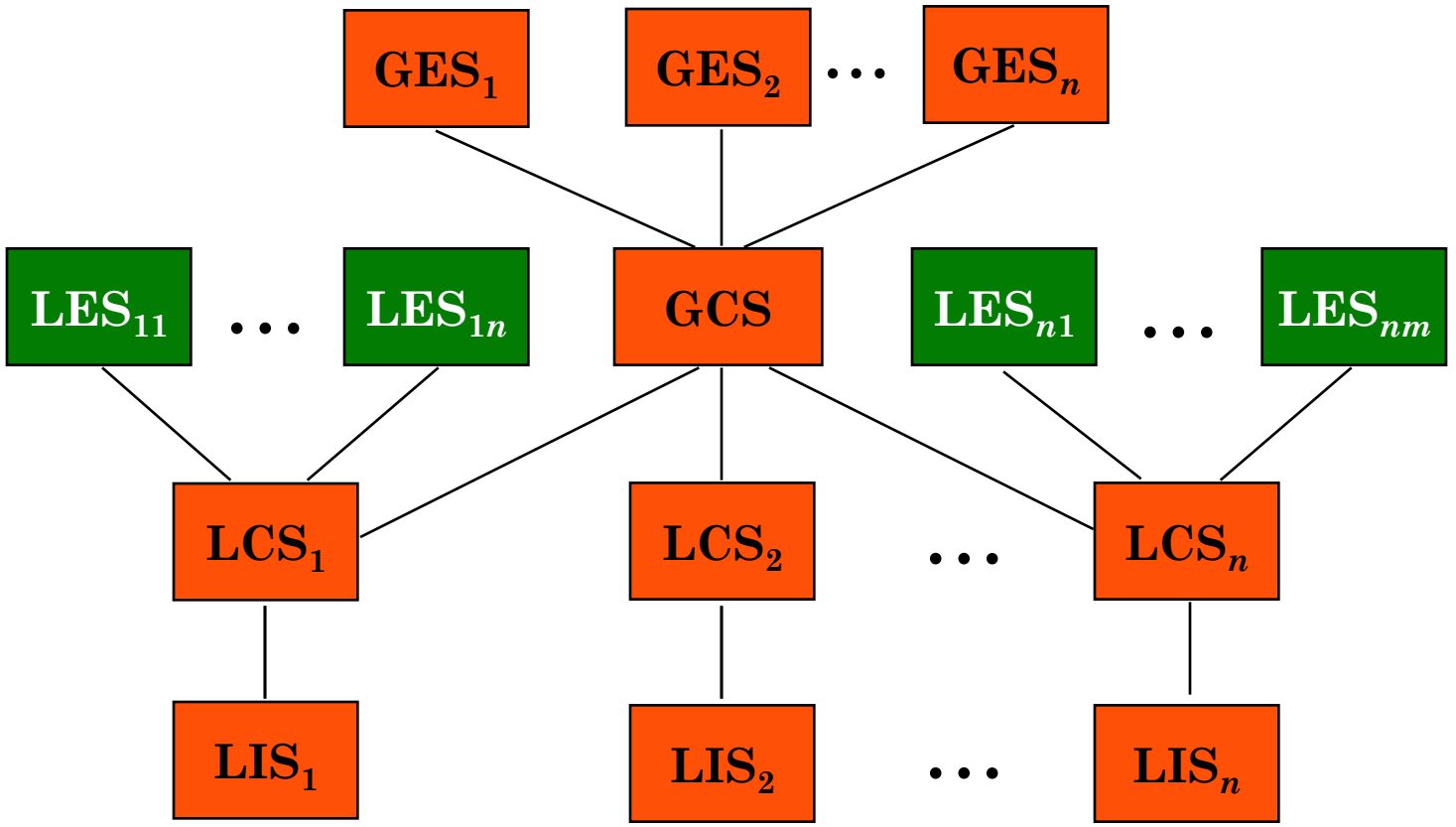
These two cases are together, because of the similarity of the problem. They both represent the case where component databases that make up the MDBS are distributed over a number of sites - we call this the *distributed MDBS*.

Data logical Distributed DBMS Architecture



ES: External Schema
GCS: Global Conceptual Schema
LCS: Local Conceptual Schema
LIS: Local Internal Schema

Datalogical Multi-DBMS Architecture

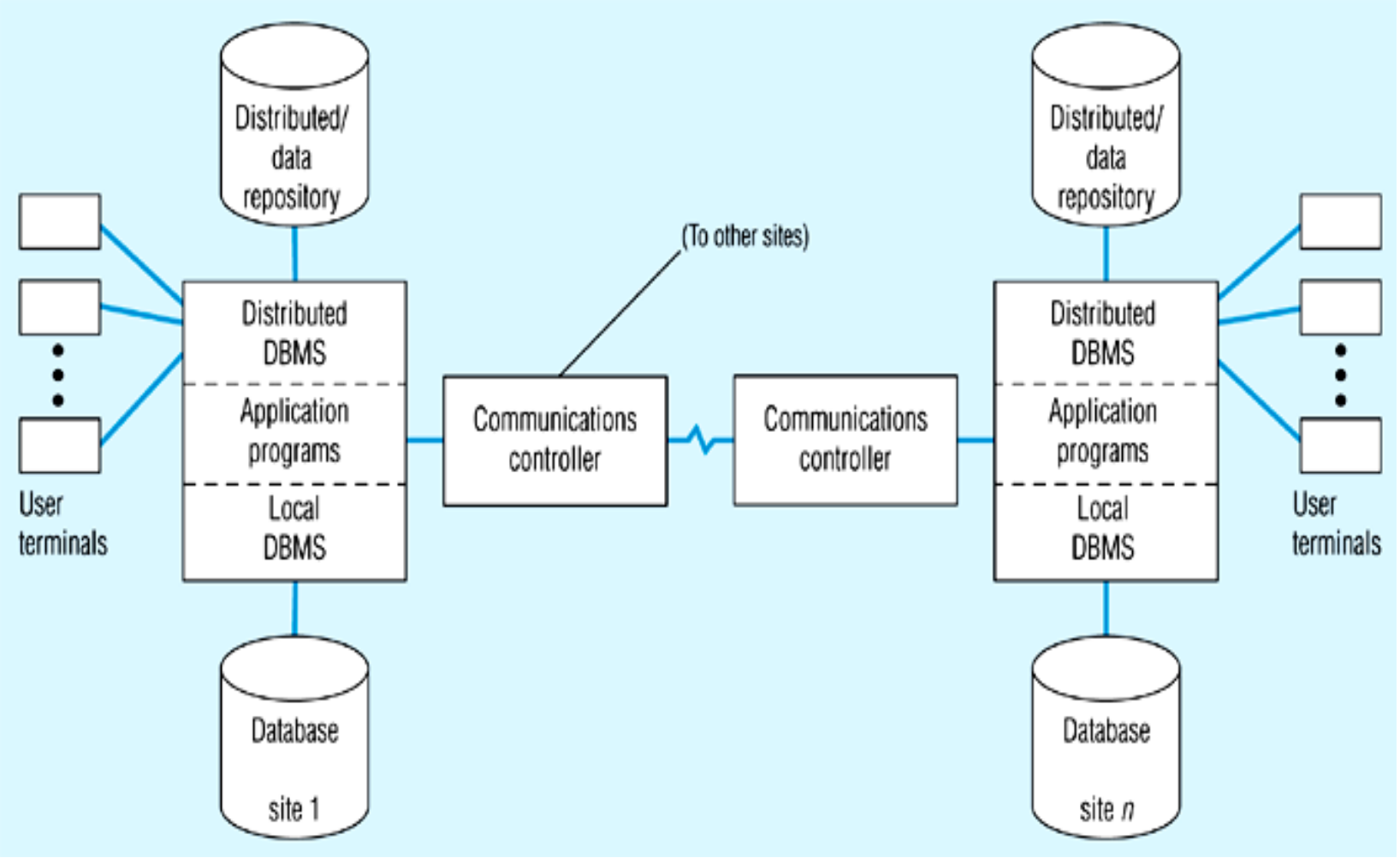


- **GES: Global External Schema**
- **LES: Local External Schema**
- LCS: Local Conceptual Schema
- LIS: Local Internal Schema

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 recovery

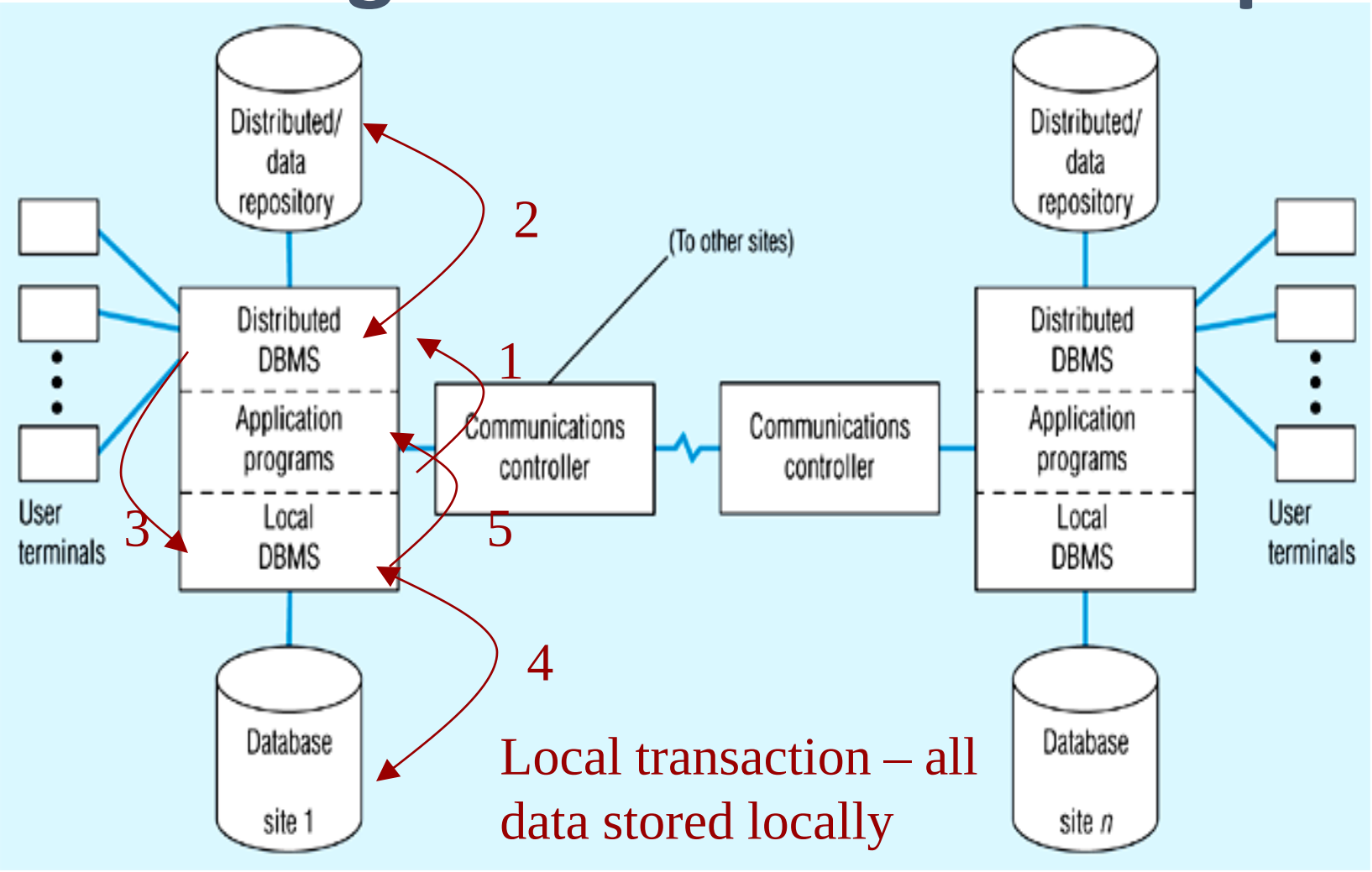
Distributed DBMS architecture



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

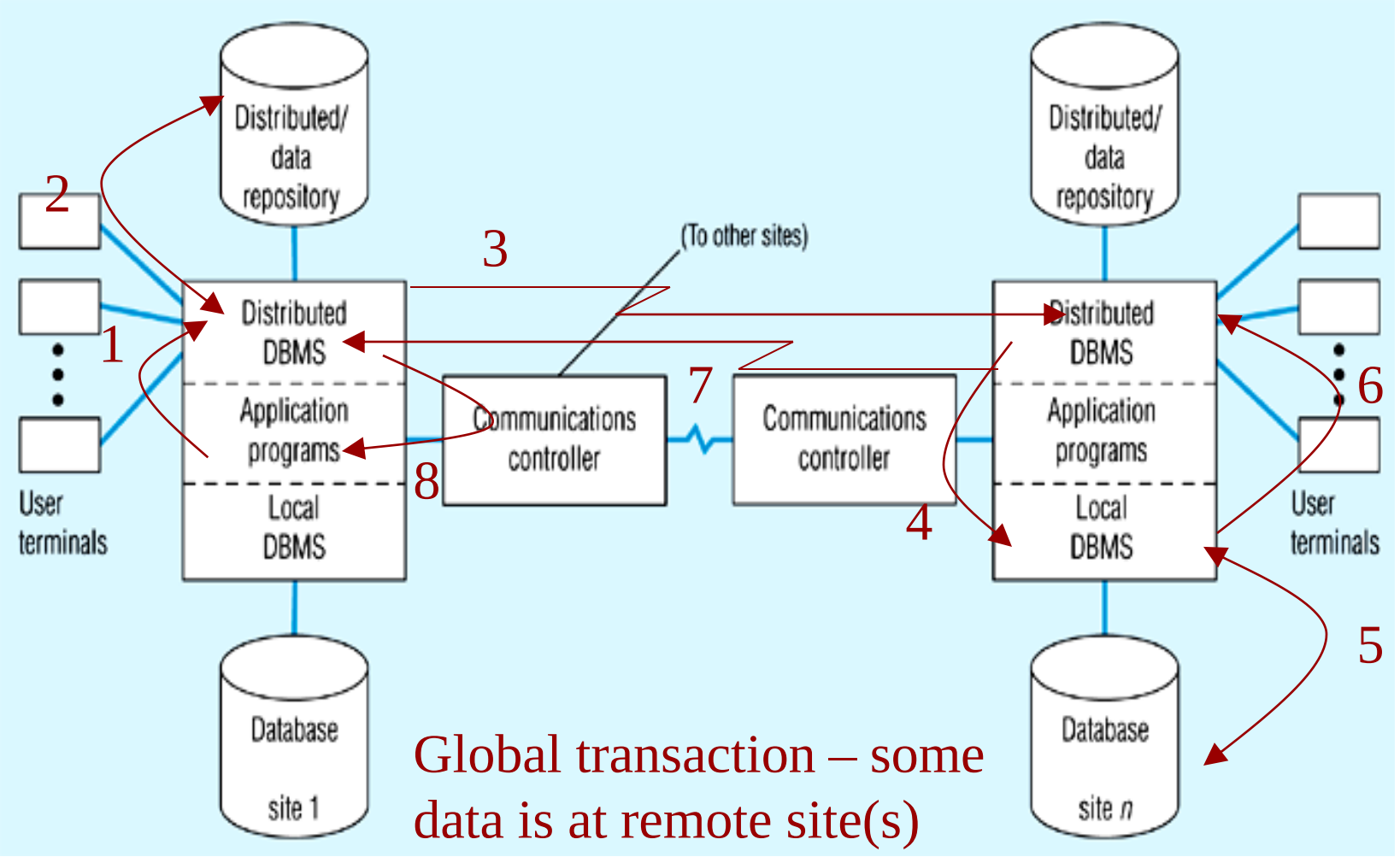
Showing local transaction steps



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

Showing global transaction steps



DISTRIBUTED DBMS ARCHITECTURE

- Client / server systems - (Ax, D1, Hy)
- Distributed databases - (A0, D2, H0)
- Multidatabase systems - (A2, Dx, Hy)

Advantages of Distributed DBs

1. Scalability
   - Horizontal scaling: Distributed databases can scale out by adding more nodes to handle increased loads, allowing for better management of large datasets and higher query throughput.
   - Elasticity: As demands increase or decrease, nodes can be added or removed dynamically, offering flexibility in resource allocation.
2. Fault Tolerance & Reliability
   - Redundancy: Data is replicated across multiple nodes, ensuring that even if one node fails, the system remains operational.
   - High availability: Distributed databases are designed to offer continuous service availability, even in the face of hardware failures or network partitions.
3. Geographical Distribution
   - Proximity to users: By distributing data across multiple geographical locations, queries can be executed closer to the user, reducing latency and improving the user experience.
   - Data localization: Certain regulations require data to be stored within specific regions; distributed databases allow compliance with these regulations while maintaining operational efficiency.

4. Load Balancing

Even distribution of queries: By distributing data across nodes, the system can balance loads more effectively, preventing any single node from becoming a bottleneck.

Improved performance: This allows for faster query processing and better performance, especially during peak loads.

5. Flexibility

Data partitioning: Distributed databases can be partitioned based on different criteria (e.g., by region, user type), allowing for more flexible data management.

No single point of failure: Since the system is distributed, it avoids centralized control or dependencies, improving overall system resilience.

6. Cost Efficiency

Commodity hardware: Distributed databases can often run on cheaper, commodity hardware, reducing overall infrastructure costs compared to centralized systems that may require high-end, specialized equipment.

7. Concurrent Access

Enhanced concurrency control: Distributed databases can handle multiple users accessing data simultaneously, improving the ability to manage large volumes of concurrent transactions without conflicts or delays.

8. Disaster Recovery

Backup and recovery: Distributed databases often have built-in mechanisms for backing up data across various locations, simplifying disaster recovery processes and reducing downtime.

Disadvantages of Distributed DBs

1. Complexity

System design and maintenance: Managing a distributed database system is far more complex than a centralized system. The need to handle replication, partitioning, synchronization, and distributed transactions increases the operational complexity.

Data consistency: Ensuring data consistency across all nodes in a distributed system can be challenging, especially in the event of network partitions or node failures.

2. Increased Latency

Network delays: Communication between geographically dispersed nodes introduces network latency, which can affect the speed of transactions, especially when consistency protocols require frequent coordination between nodes.

Longer transaction times: Distributed transactions that span multiple nodes often take longer due to the need for coordination and synchronization.

3. Data Integrity Challenges

Eventual consistency: Many distributed databases favor eventual consistency over strong consistency to achieve better performance. This means that there may be a delay before all nodes reflect the same data, potentially leading to temporary discrepancies.

Concurrency issues: Managing concurrent updates across distributed nodes can lead to issues such as data conflicts or inconsistencies.

4. Cost of Coordination

High overhead: Synchronizing and coordinating data across multiple nodes introduces significant overhead in terms of both network traffic and processing power, especially for distributed transactions that require atomicity and consistency.

Consensus algorithms: Mechanisms like Paxos or Raft, which ensure consistency and fault tolerance, come with added complexity and can slow down the system.

5. Security Risks

Broader attack surface: A distributed system with multiple nodes, often spread across different regions, increases the potential points of attack. Ensuring security at every node requires careful planning and execution.

Data transmission vulnerabilities: Data that is transmitted between nodes is vulnerable to interception or corruption unless robust encryption and security protocols are used.

6. Operational Costs

Infrastructure expenses: While distributed databases can utilize commodity hardware, the need for multiple nodes, data centers, and redundant systems can increase costs related to infrastructure, networking, and management.

Energy consumption: Operating a distributed system across multiple nodes can lead to increased energy consumption, both from hardware and cooling requirements in data centers.

7. Data Partitioning and Sharding Challenges

Skewed partitions: Inefficient partitioning or sharding of data can lead to some nodes being overloaded while others remain underutilized, which can degrade performance.

Rebalancing complexities: When nodes are added or removed, rebalancing data across nodes can be a complex and resource-intensive process.

8. Difficulty in Debugging and Monitoring

Distributed debugging: **Identifying and resolving issues in a distributed database is more challenging because failures can occur at different nodes, often leading to more complex root cause analysis.**

Monitoring and logging: **Keeping track of performance metrics, system health, and errors across distributed nodes requires sophisticated monitoring tools and careful analysis.**

9. Regulatory and Compliance Issues

Data locality restrictions: **Depending on the regulatory requirements of different regions, distributing data across borders may lead to compliance issues related to data sovereignty, privacy laws, and other legal obligations.**

Handling GDPR and similar regulations: **Compliance with regulations like GDPR can be complicated when data is spread across different jurisdictions, requiring additional safeguards.**

10. Data Loss Risks

Replication lag: **If data replication is not handled properly or there is a significant lag, there may be a risk of data loss in the event of a node failure before the data is fully replicated across the system.**