Distributed databases

Concepts

Distributed Database.

A logically interrelated collection of shared data (and a description of this data), physically distributed over a computer network.

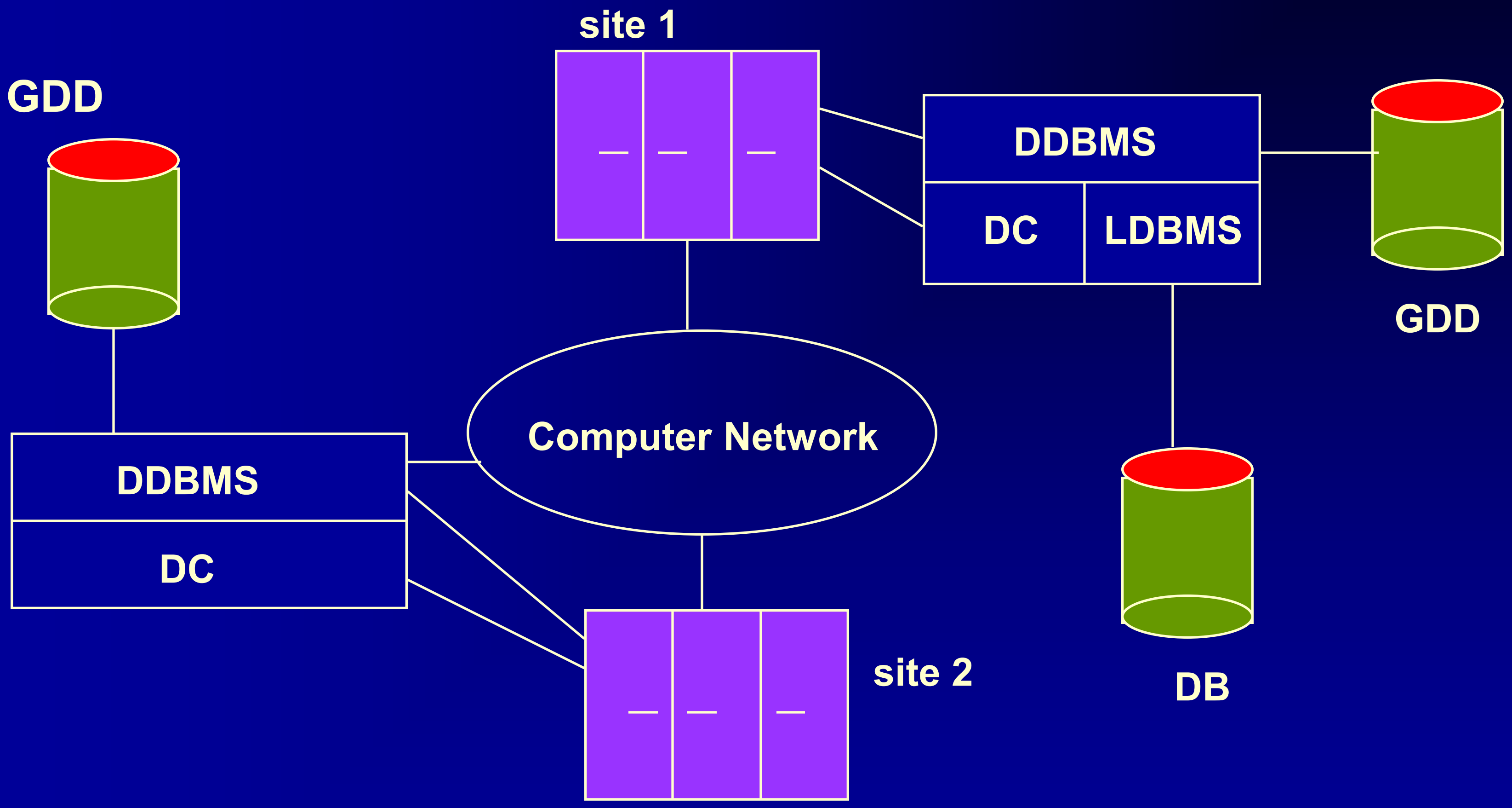
Distributed DBMS.

Software system that permits the management of the distributed database and makes the distribution transparent to users.

Concepts

- Collection of logically-related shared data.
- Data split into fragments.
- Fragments may be replicated.
- Fragments/replicas allocated to sites.
- Sites linked by a communications network.
- Data at each site is under control of a DBMS.
- DBMSs handle local applications autonomously.
- Each DBMS participates in at least one global application.

Component Architecture for a DDBMS



LDBMS : Local DBMS component
DC : Data communication component
GDD : Global Data Dictionary

The Ideal Situation

- A single application should be able to operate transparently on data that is:
  - ⇨ spread across a variety of different DBMS's
  - ⇨ running on a variety of different machines
  - ⇨ supported by a variety of different operating systems
  - ⇨ connected together by a variety of different communication networks
- The distribution can be geographical or local

Workable definition

A distributed database system consists of a collection of sites connected together via some kind of communications network, in which :

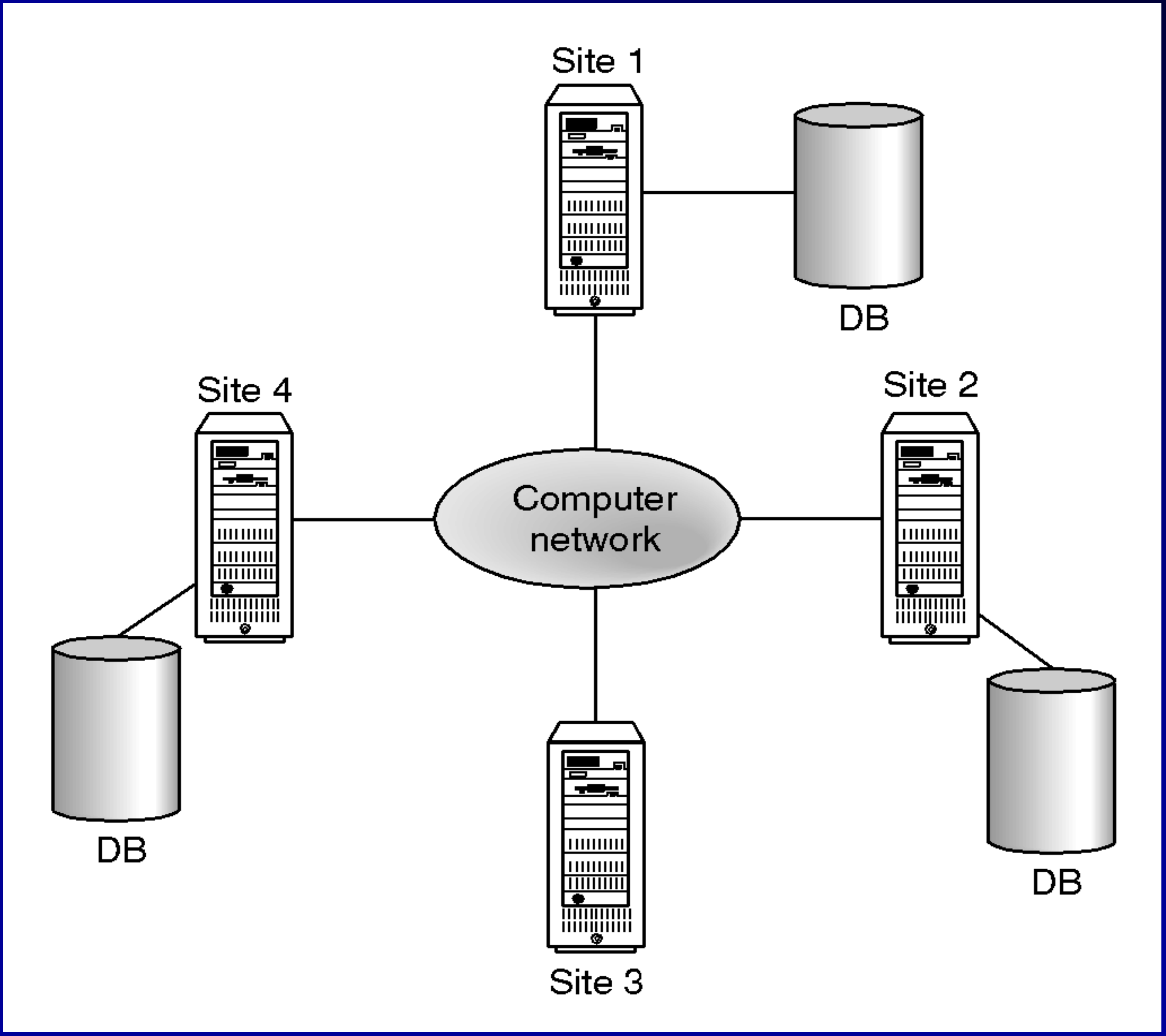
- each site is a database system site in its own right;
- the sites agree to work together, so that a user at any site can access data anywhere in the network exactly as if the data were all stored at the user's own site

It is a logical union of real databases

- It can be seen as a kind of partnership among individual local DBMS's

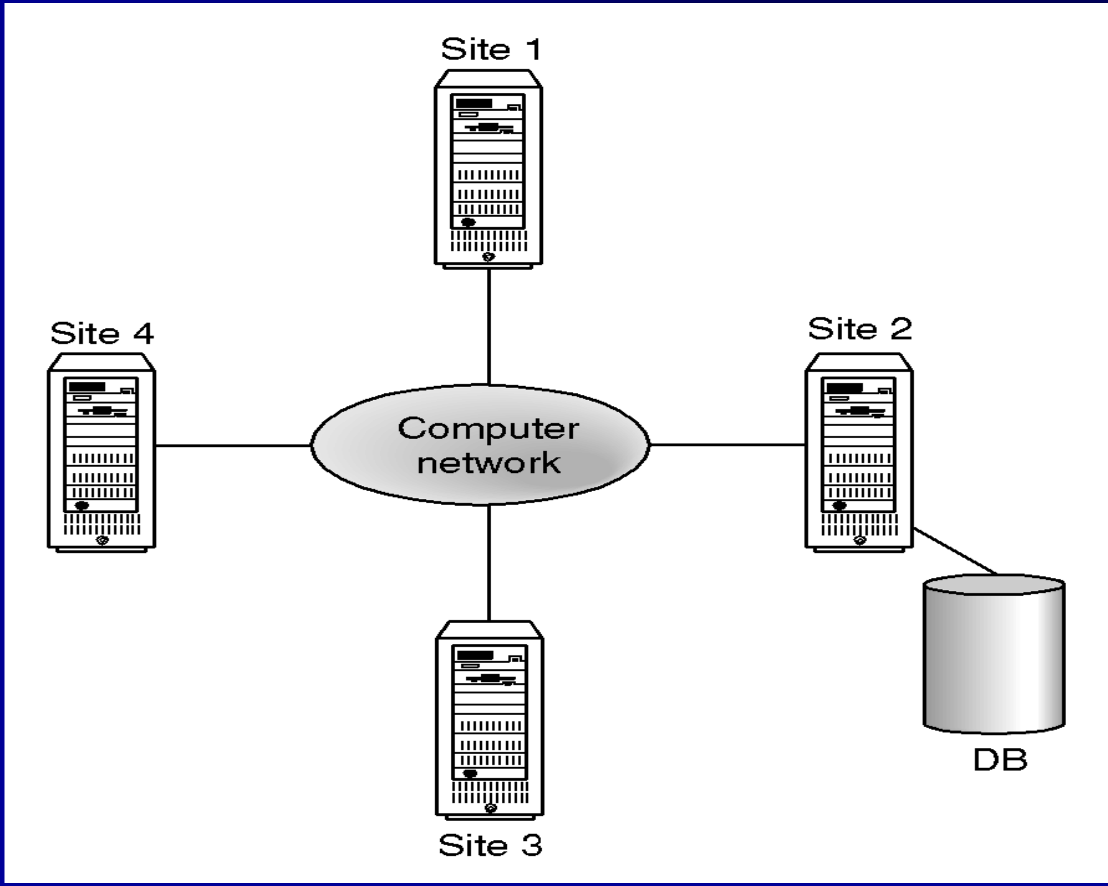
- Difference with remote access or distributed processing systems
- Temporary assumption: strict homogeneity

Distributed DBMS



Distributed Processing

- A centralized database that can be accessed over a computer network.



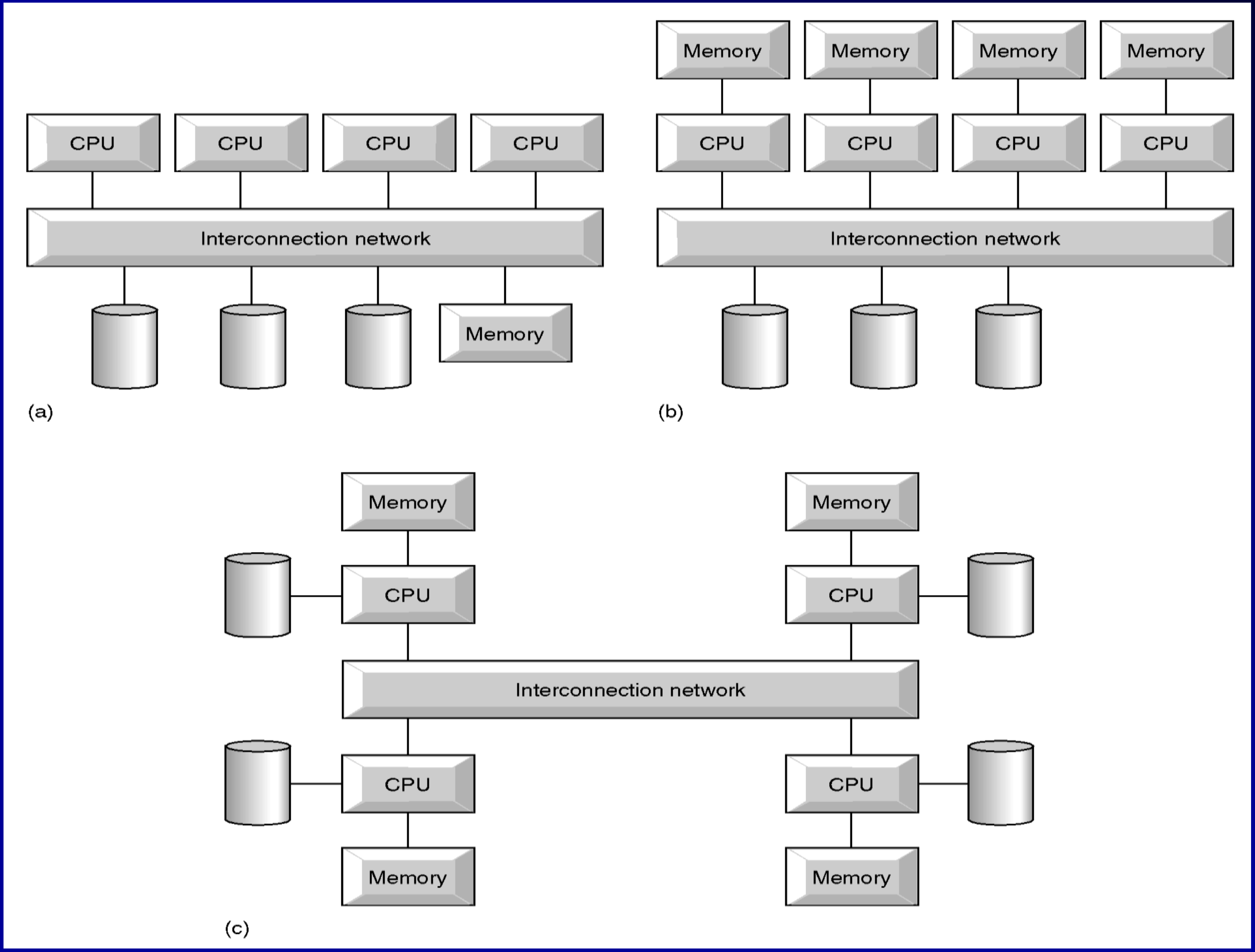
Parallel DBMS

- A DBMS running across multiple processors and disks designed to execute operations in parallel, whenever possible, to improve performance.
- Based on premise that single processor systems can no longer meet requirements for cost-effective scalability, reliability, and performance.
- Parallel DBMSs link multiple, smaller machines to achieve same throughput as single, larger machine, with greater scalability and reliability.

Parallel DBMS

- **Main architectures for parallel DBMSs are:**
  - **a: Shared memory.**
  - **b: Shared disk.**
  - **c: Shared nothing.**

Parallel DBMS



Advantages of DDBMSs

- Organizational Structure
- Shareability and Local Autonomy
- Improved Availability
- Improved Reliability
- Improved Performance
- Economics
- Modular Growth

Disadvantages of DDBMSs

- Complexity
- Cost
- Security
- Integrity Control More Difficult
- Lack of Standards
- Lack of Experience
- Database Design More Complex

Types of DDBMS

- **Homogeneous DDBMS**
- **Heterogeneous DDBMS**

Homogeneous DDBMS

- All sites use same DBMS product.
- Much easier to design and manage.
- Approach provides incremental growth and allows increased performance.

Heterogeneous DDBMS

- Sites may run different DBMS products, with possibly different underlying data models.
- Occurs when sites have implemented their own databases and integration is considered later.
- Translations required to allow for:
  - Different hardware.
  - Different DBMS products.
  - Different hardware and different DBMS products.
- Typical solution is to use gateways.

Open Database Access and Interoperability

- **Open Group has formed a Working Group to provide specifications that will create database infrastructure environment where there is:**
- **Common SQL API that allows client applications to be written that do not need to know vendor of DBMS they are accessing.**
  - ⇨ **Common database protocol that enables DBMS from one vendor to communicate directly with DBMS from another vendor without the need for a gateway.**
  - ⇨ **A common network protocol that allows communications between different DBMSs.**
- **Most ambitious goal is to find a way to enable transaction to span DBMSs from different vendors without use of a gateway.**

Multidatabase System (MDBS)

- **DDBMS in which each site maintains complete autonomy.**

- **DBMS that resides transparently on top of existing database and file systems and presents a single database to its users.**
- **Allows users to access and share data without requiring physical database integration.**
- **Non-federated MDBS (no local users) and federated MDBS (FMDBS).**

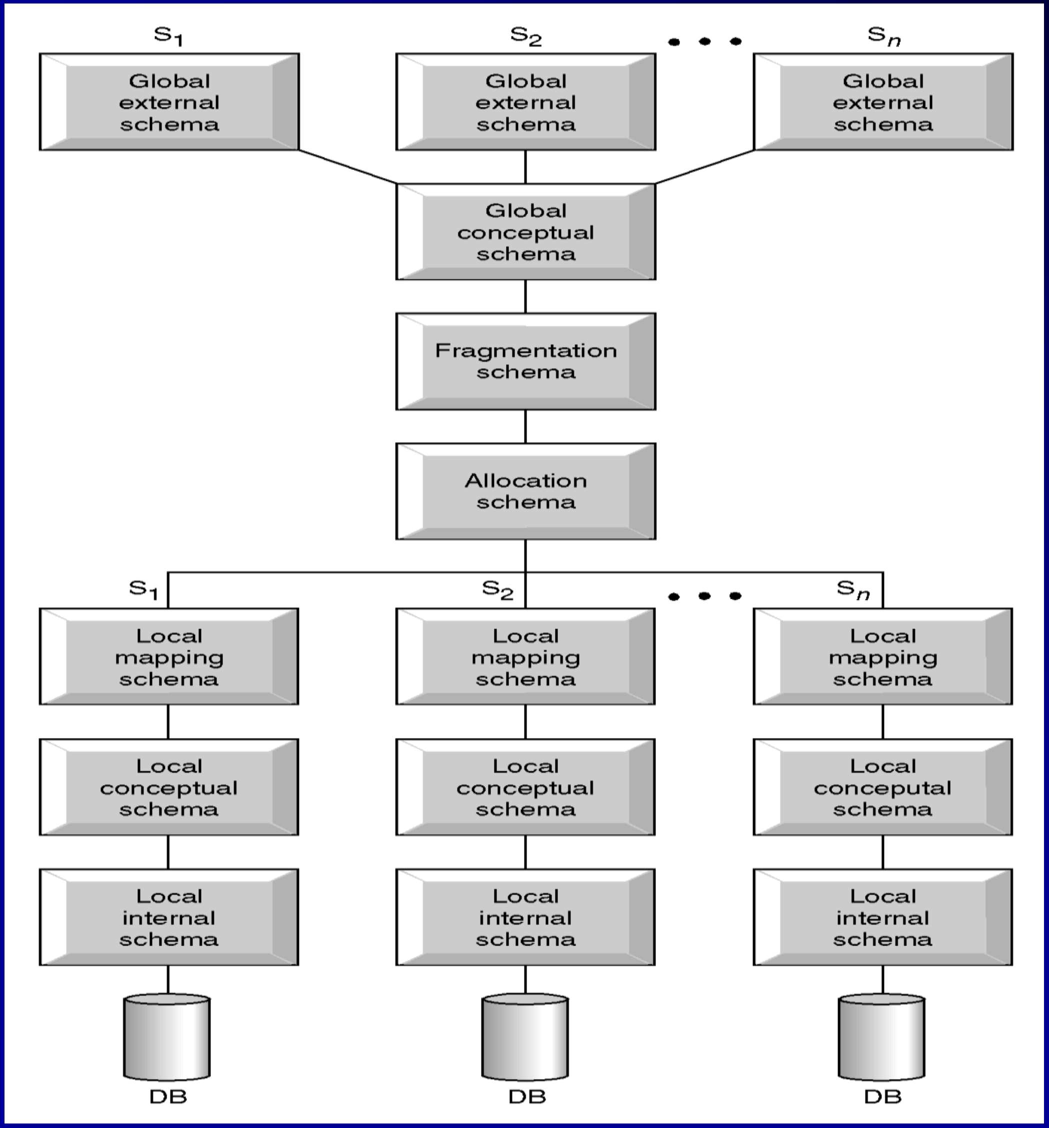
Functions of a DDBMS

- Expect DDBMS to have at least the functionality of a DBMS.
- Also to have following functionality:
  - Extended communication services.
  - Extended Data Dictionary.
  - Distributed query processing.
  - Extended concurrency control.
  - Extended recovery services.

Reference Architecture for DDBMS

- Due to diversity, no universally accepted architecture such as the ANSI/SPARC 3-level architecture.
- A reference architecture consists of:
  - ⇨ Set of global external schemas.
  - ⇨ Global conceptual schema (GCS).
  - ⇨ Fragmentation schema and allocation schema.
  - ⇨ Set of schemas for each local DBMS conforming to 3-level ANSI/SPARC .
- Some levels may be missing, depending on levels of transparency supported.

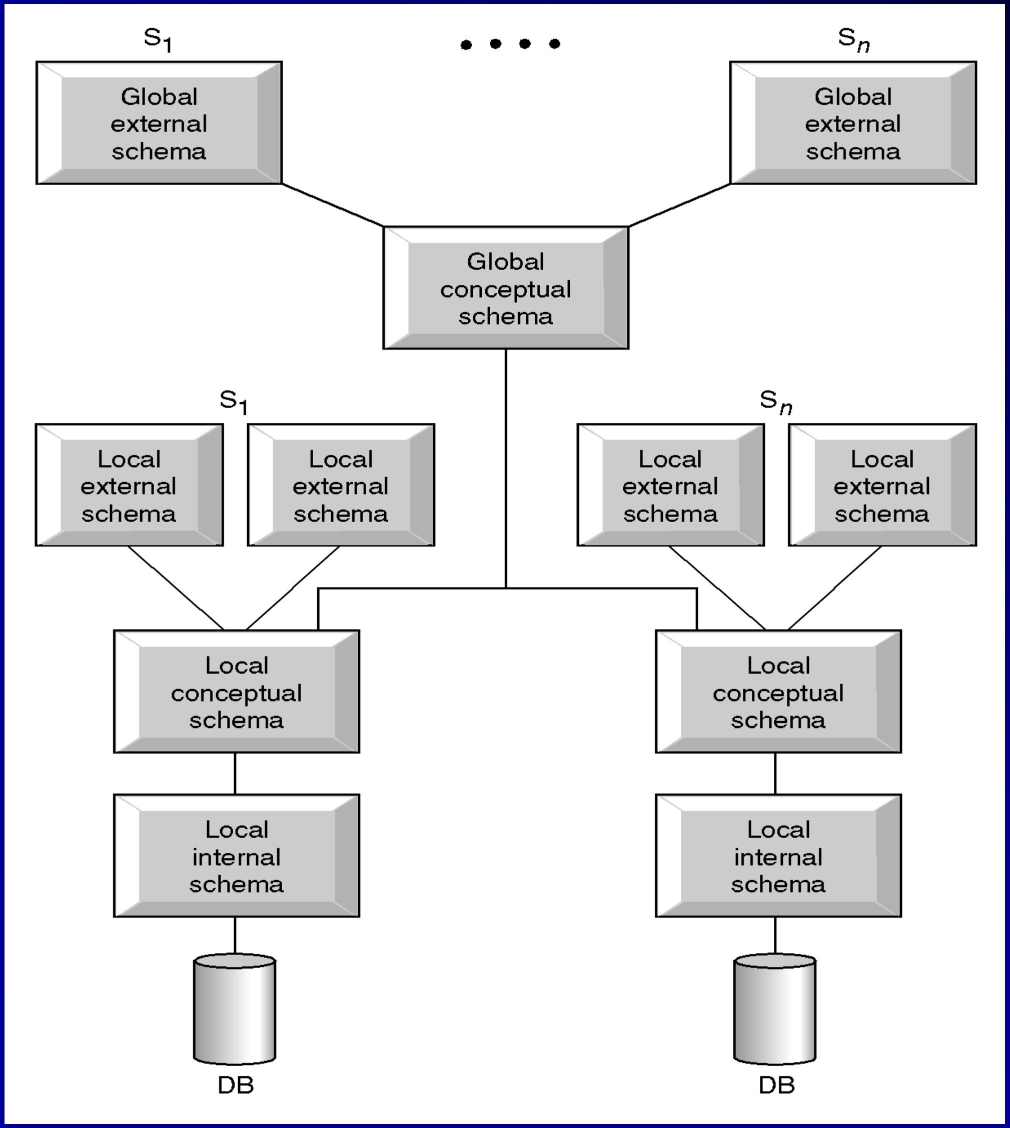
Reference Architecture for DDBMS



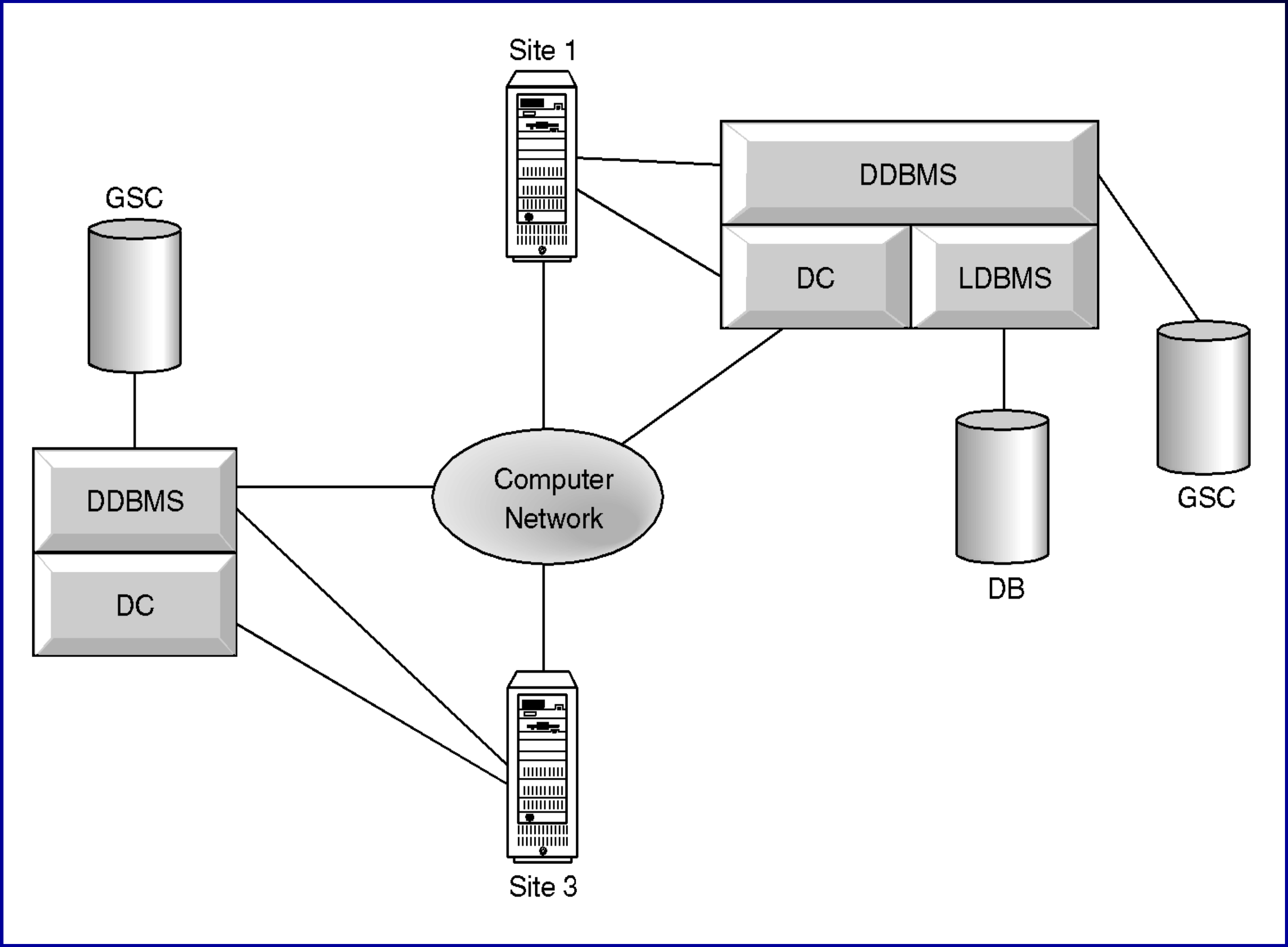
Reference Architecture for MDBS

- In DDBMS, GCS is union of all local conceptual schemas.
- In FMDBS, GCS is subset of local conceptual schemas (LCS), consisting of data that each local system agrees to share.
- GCS of tightly coupled system involves integration of either parts of LCSs or local external schemas.
- FMDBS with no GCS is called loosely coupled.

Reference Architecture for Tightly-Coupled Federated MDBS



Components of a DDBMS



Distributed Database Design

- Three key issues:
  - ⇨ Fragmentation.
  - ⇨ Allocation
  - ⇨ Replication

Distributed Database Design

- **Fragmentation**
  - ⇨ Relation may be divided into a number of sub-relations, which are then distributed.
- **Allocation**
  - ⇨ Each fragment is stored at site with "optimal" distribution.
- **Replication**
  - ⇨ Copy of fragment may be maintained at several sites.

Fragmentation

- Definition and allocation of fragments carried out strategically to achieve:
  - ⇨ Locality of Reference
  - ⇨ Improved Reliability and Availability
  - ⇨ Improved Performance
  - ⇨ Balanced Storage Capacities and Costs
  - ⇨ Minimal Communication Costs.
- Involves analyzing most important applications, based on quantitative/qualitative information.

Fragmentation

- Quantitative information may include:
  - ⇨ frequency with which an application is run;
  - ⇨ site from which an application is run;
  - ⇨ performance criteria for transactions and applications.
- Qualitative information may include transactions that are executed by application, type of access (read or write), and predicates of read operations.

Data Allocation

- **Four alternative strategies regarding placement of data:**
  - ⇨ **Centralized**
  - ⇨ **Partitioned (or Fragmented)**
  - ⇨ **Complete Replication**
  - ⇨ **Selective Replication**

Data Allocation

- **Centralized**
  - **Consists of single database and DBMS stored at one site with users distributed across the network.**
- **Partitioned**
  - **Database partitioned into disjoint fragments, each fragment assigned to one site.**

Data Allocation

- **Complete Replication**
  - ⇨ **Consists of maintaining complete copy of database at each site.**
- **Selective Replication**
  - ⇨ **Combination of partitioning, replication, and centralization.**

Comparison of Strategies for Data Distribution

Table 19.3 Comparison of strategies for data allocation.

	Locality of reference	*Reliability and availability*	*Performance*	*Storage costs*	*Communication costs*
Centralized	lowest	lowest	unsatisfactory	lowest	highest
Partitioned	high†	low for item; high for system	satisfactory†	lowest	low†
Complete replication	highest	highest	best for read	highest	high for update; low for read
Selective replication	high†	low for item; high for system	satisfactory†	average	low†

† Indicates subject to good design.

Why Fragment?

- Usage
  - ⇨ Applications work with views rather than entire relations.
- Efficiency
  - ⇨ Data is stored close to where it is most frequently used.
  - ⇨ Data that is not needed by local applications is not stored.

Why Fragment?

- Parallelism
  - With fragments as unit of distribution, transaction can be divided into several subqueries that operate on fragments.
- Security
  - Data not required by local applications is not stored and so not available to unauthorized users.

- Disadvantages
  - Performance
  - Integrity.

Correctness of Fragmentation

- Three correctness rules:
  - Completeness
  - Reconstruction
  - Disjointness.

Correctness of Fragmentation

- Completeness
  - ⇨ If relation R is decomposed into fragments R1, R2, ... Rn, each data item that can be found in R must appear in at least one fragment.
- Reconstruction
- Must be possible to define a relational operation that will reconstruct R from the fragments.
- Reconstruction for horizontal fragmentation is Union operation and Join for vertical .

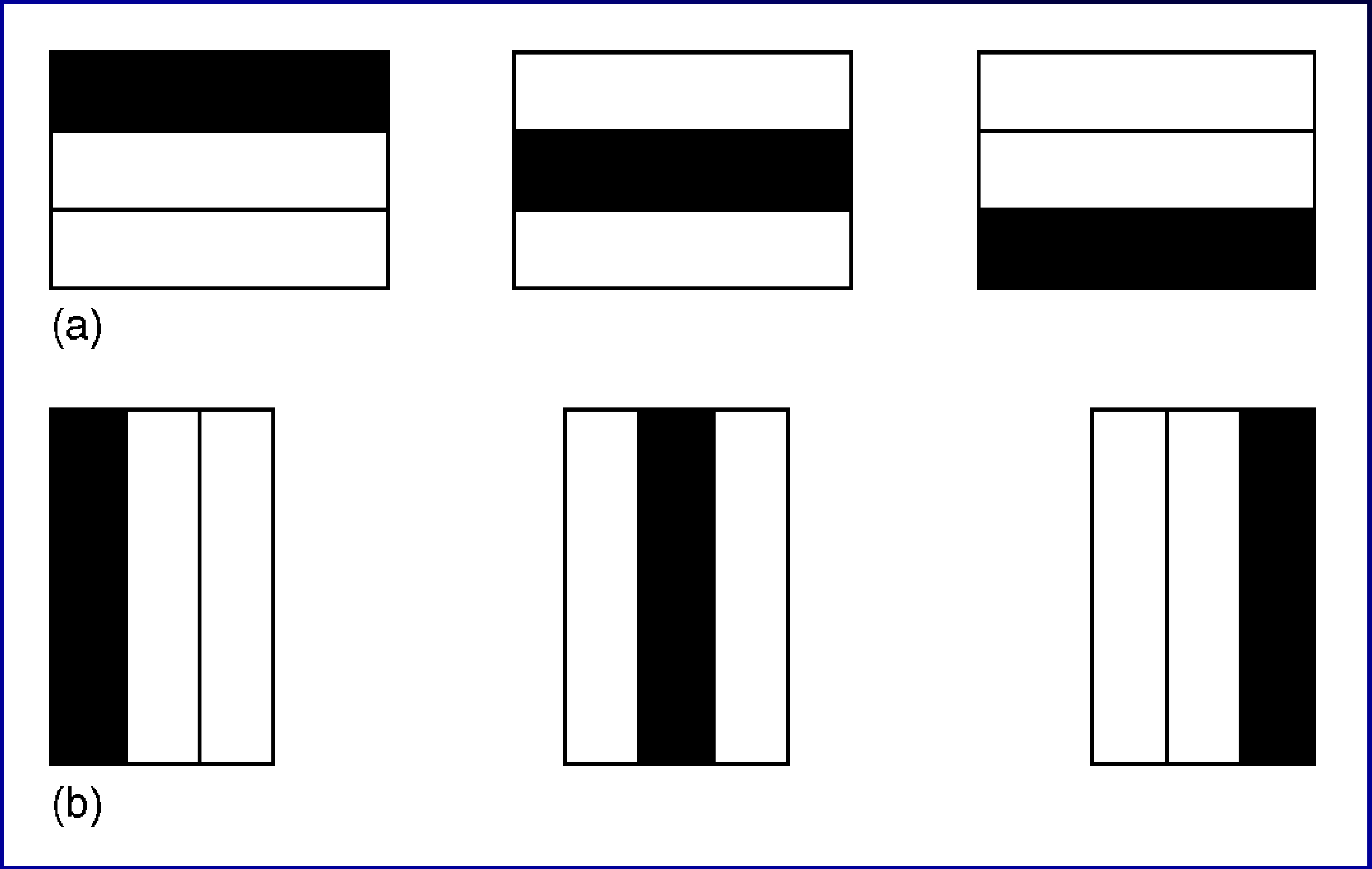
Correctness of Fragmentation

- Disjointness
- If data item di appears in fragment Ri, then it should not appear in any other fragment.
- Exception: vertical fragmentation, where primary key attributes must be repeated to allow reconstruction.
- For horizontal fragmentation, data item is a tuple
- For vertical fragmentation, data item is an attribute.

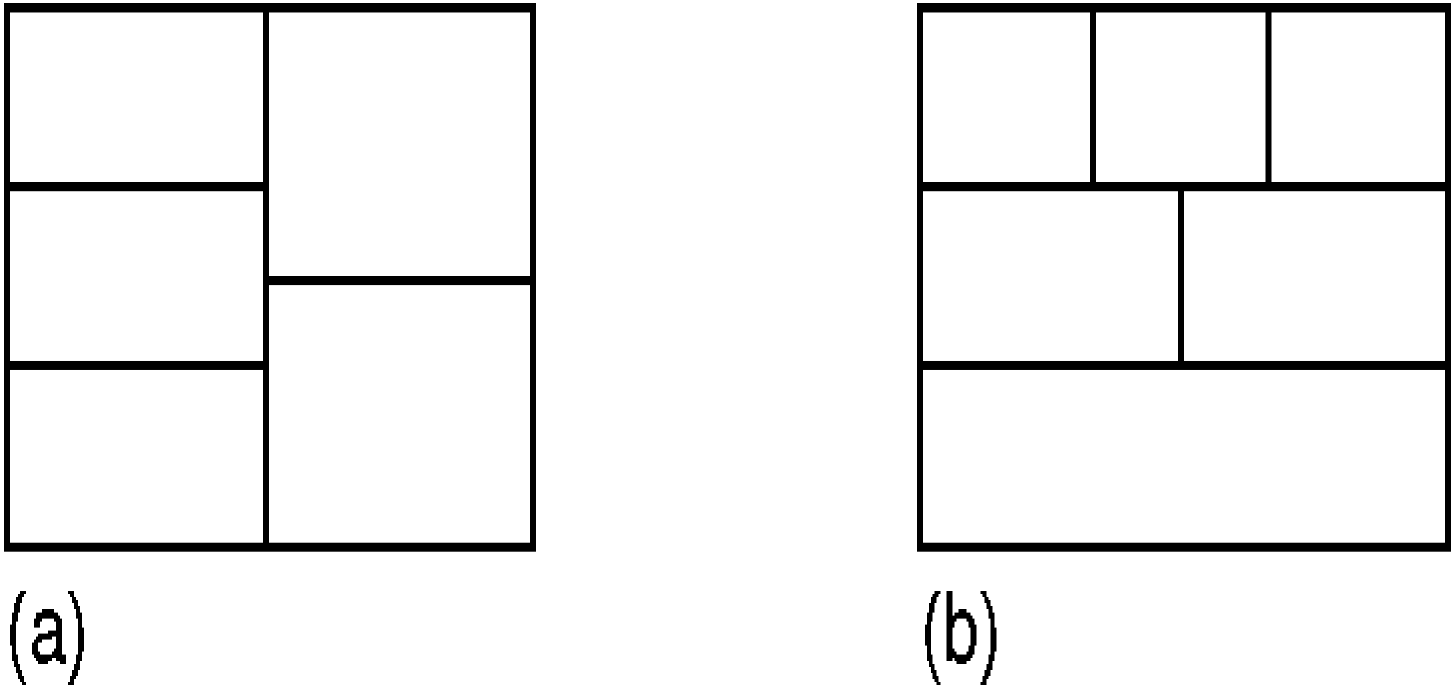
Types of Fragmentation

- Four types of fragmentation:
  - Horizontal
  - Vertical
  - Mixed
  - Derived.
- Other possibility is no fragmentation:
  - If relation is small and not updated frequently, may be better not to fragment relation.

Horizontal and Vertical Fragmentation



Mixed Fragmentation



Horizontal Fragmentation

- This strategy is determined by looking at predicates used by transactions.
- Involves finding set of minimal (complete and relevant) predicates.
- Set of predicates is complete, if and only if, any two tuples in same fragment are referenced with same probability by any application.
- Predicate is relevant if there is at least one application that accesses fragments differently.

Transparencies in a DDBMS

- **Distribution Transparency**
  - ⇨ **Fragmentation Transparency**
  - ⇨ **Location Transparency**
  - ⇨ **Replication Transparency**
  - ⇨ **Local Mapping Transparency**
  - ⇨ **Naming Transparency**

Transparencies in a DDBMS

- Transaction Transparency
  - Concurrency Transparency
  - Failure Transparency
- Performance Transparency
- DBMS Transparency

Distribution Transparency

- Distribution transparency allows user to perceive database as single, logical entity.
- If DDBMS exhibits distribution transparency, user does not need to know:
  - data is fragmented (fragmentation transparency),
  - location of data items (location transparency),
  - otherwise call this local mapping transparency.
- With replication transparency, user is unaware of replication of fragments .

Naming Transparency

- Each item in a DDB must have a unique name.
- DDBMS must ensure that no two sites create a database object with same name.
- One solution is to create central name server. However, this results in:
  - loss of some local autonomy;
  - central site may become a bottleneck;
  - low availability; if the central site fails, remaining sites cannot create any new objects.

Transaction Transparency

- Ensures that all distributed transactions maintain distributed database's integrity and consistency.
- Distributed transaction accesses data stored at more than one location.
- Each transaction is divided into number of sub-transactions, one for each site that has to be accessed.
- DDBMS must ensure the indivisibility of both the global transaction and each subtransactions.

Concurrency Transparency

- All transactions must execute independently and be logically consistent with results obtained if transactions executed one at a time, in some arbitrary serial order.
- Same fundamental principles as for centralized DBMS.
- DDBMS must ensure both global and local transactions do not interfere with each other.
- Similarly, DDBMS must ensure consistency of all sub-transactions of global transaction.

Concurrency Transparency

- Replication makes concurrency more complex.
- If a copy of a replicated data item is updated, update must be propagated to all copies.
- Could propagate changes as part of original transaction, making it an atomic operation.
- However, if one site holding copy is not reachable, then transaction is delayed until site is reachable.

Concurrency Transparency

- Could limit update propagation to only those sites currently available. Remaining sites updated when they become available again.
- Could allow updates to copies to happen asynchronously, sometime after the original update. Delay in regaining consistency may range from a few seconds to several hours.

Failure Transparency

- **DDBMS must ensure atomicity and durability of global transaction.**
- **Means ensuring that sub-transactions of global transaction either all commit or all abort.**
- **Thus, DDBMS must synchronize global transaction to ensure that all sub-transactions have completed successfully before recording a final COMMIT for global transaction.**
- **Must do this in presence of site and network failures.**

Performance Transparency

- **DDBMS must perform as if it were a centralized DBMS.**
  - ⇨ **DDBMS should not suffer any performance degradation due to distributed architecture.**
  - ⇨ **DDBMS should determine most cost-effective strategy to execute a request.**

Performance Transparency

- **Distributed Query Processor (DQP) maps data request into ordered sequence of operations on local databases.**
- **Must consider fragmentation, replication, and allocation schemas.**
- **DQP has to decide:**
  - **which fragment to access;**
  - **which copy of a fragment to use;**
  - **which location to use.**

Performance Transparency

- **DQP produces execution strategy optimized with respect to some cost function.**
- **Typically, costs associated with a distributed request include:**
  - **I/O cost;**
  - **CPU cost;**
  - **communication cost.**

Date's 12 Rules for a DDBMS

- 0. Fundamental Principle
  - To the user, a distributed system should look exactly like a non-distributed system.
- 1. Local Autonomy
- 2. No Reliance on a Central Site
- 3. Continuous Operation
- 4. Location Independence
- 5. Fragmentation Independence
- 6. Replication Independence

Date's 12 Rules for a DDBMS

- 7. Distributed Query Processing
- 8. Distributed Transaction Processing
- 9. Hardware Independence
- 10. Operating System Independence
- 11. Network Independence
- 12. Database Independence
- Last four rules are ideals.