

# Distributed DBMSs

Dr Na Yao

# Objectives

- Database architectures
- Concepts of distributed databases
- Functions and architecture for a DDBMS
- Distributed database design
- Fragmentation
- Levels of transparency
- Advantages and disadvantages of distributed databases


# Database architectures

- Database application programmes and the database DO NOT NEED to reside on the same computer. (usually they don't!)
- Elements:
  - *Client*: the machine running the user interface of the applications.
  - *Database server*: the machine that runs the DBMS and contains the database.
- Categories:
  1. Personal databases
  2. Two-tier databases
  3. Multi-tier databases.

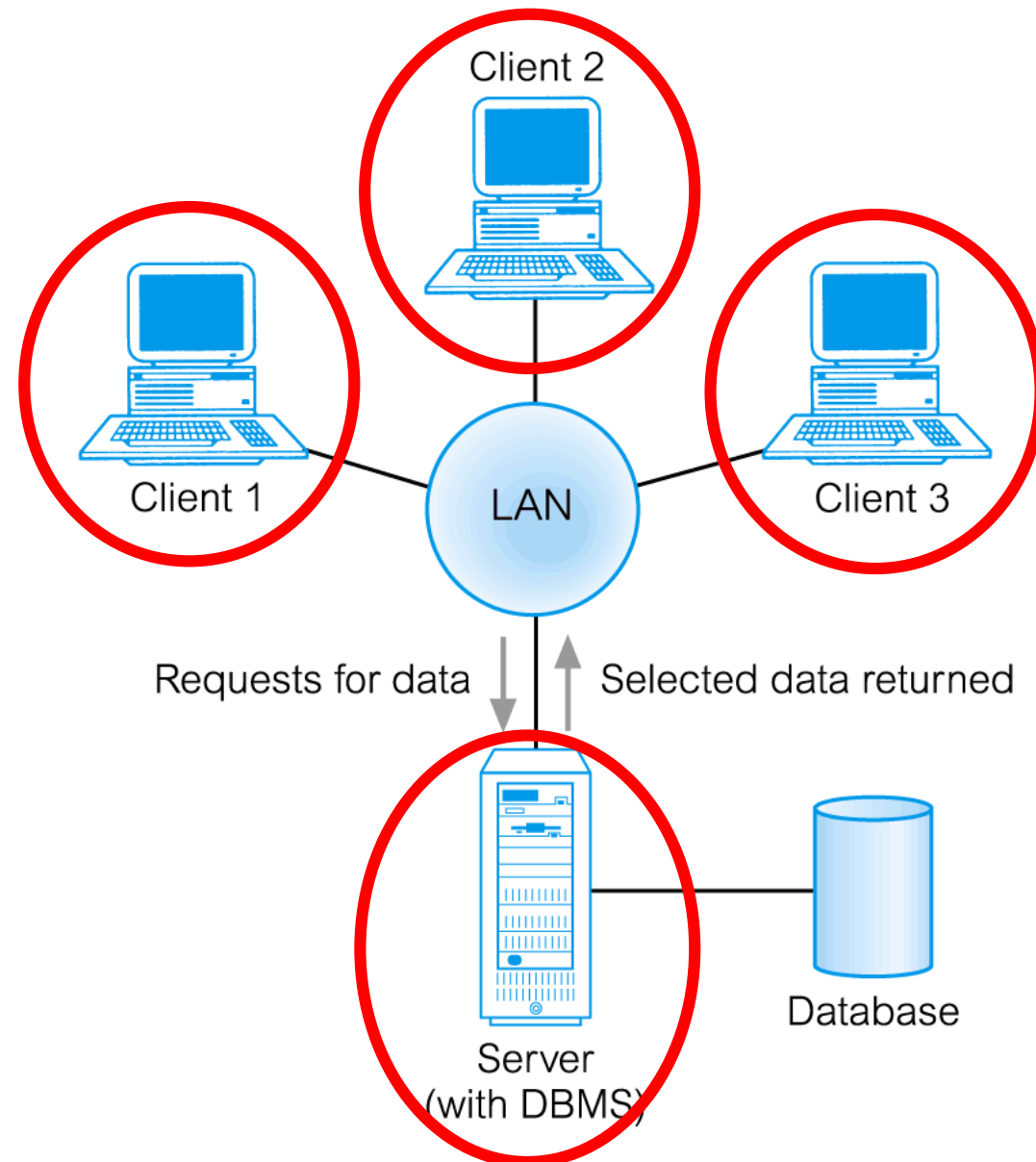
# Personal databases

- **Purpose:** allow the user to manage (store, update, delete, and retrieve) small amounts of data in an efficient manner.

 • **Pro:** can improve personal productivity

 • **Con:** the data cannot easily be shared with other users

# Two-Tier Client/Server Architecture



- Computers (*client*) connected over wired or wireless local area network (*LAN*)
- The database itself and the DBMS are stored on a central device called the database *server*, which is also connected to the network.

# Two-Tier Client/Server Architecture

## First Tier

*Client*

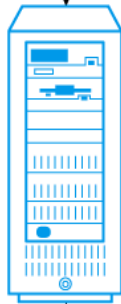


## Tasks

- User interface
- Main business and data processing logic

## Second Tier

*Database server*

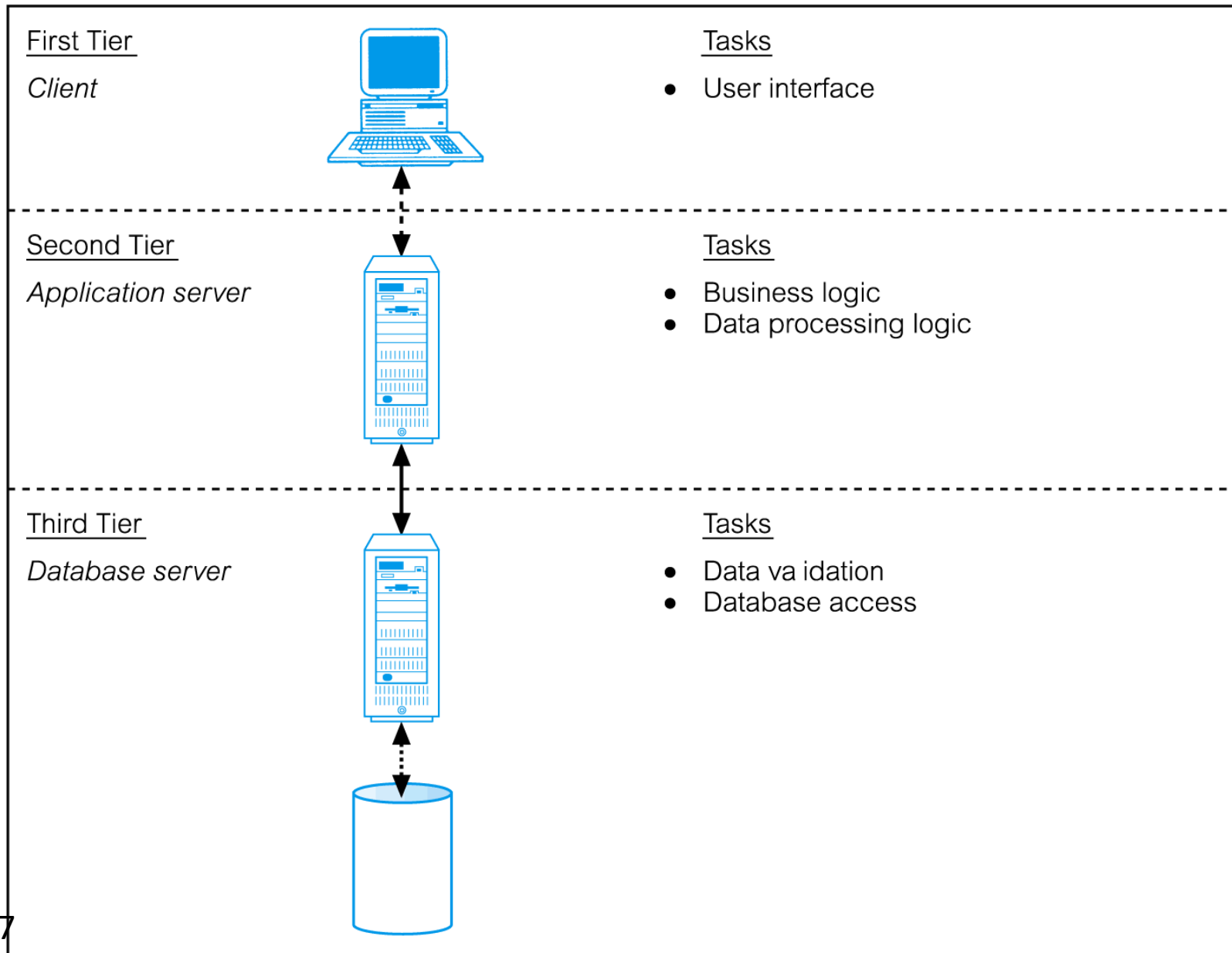


## Tasks

- Server-side validation
- Database access



# Three-Tier Client-Server Architecture



# Multi-tier Client/Server Architecture

- The user interface layer, which runs on the end-user's computer (the *client*).
- The business logic and data processing layer. This middle tier runs on a server and is often called the *application server*.
- A DBMS, which stores the data required by the middle tier. This tier may run on a separate server called the *database server*.
- The three-tier architecture can be extended to *multi*-tiers, with additional tiers added to provide more flexibility and scalability.





# Distributed Database

- **Distributed Database**

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

- **Distributed DBMS**

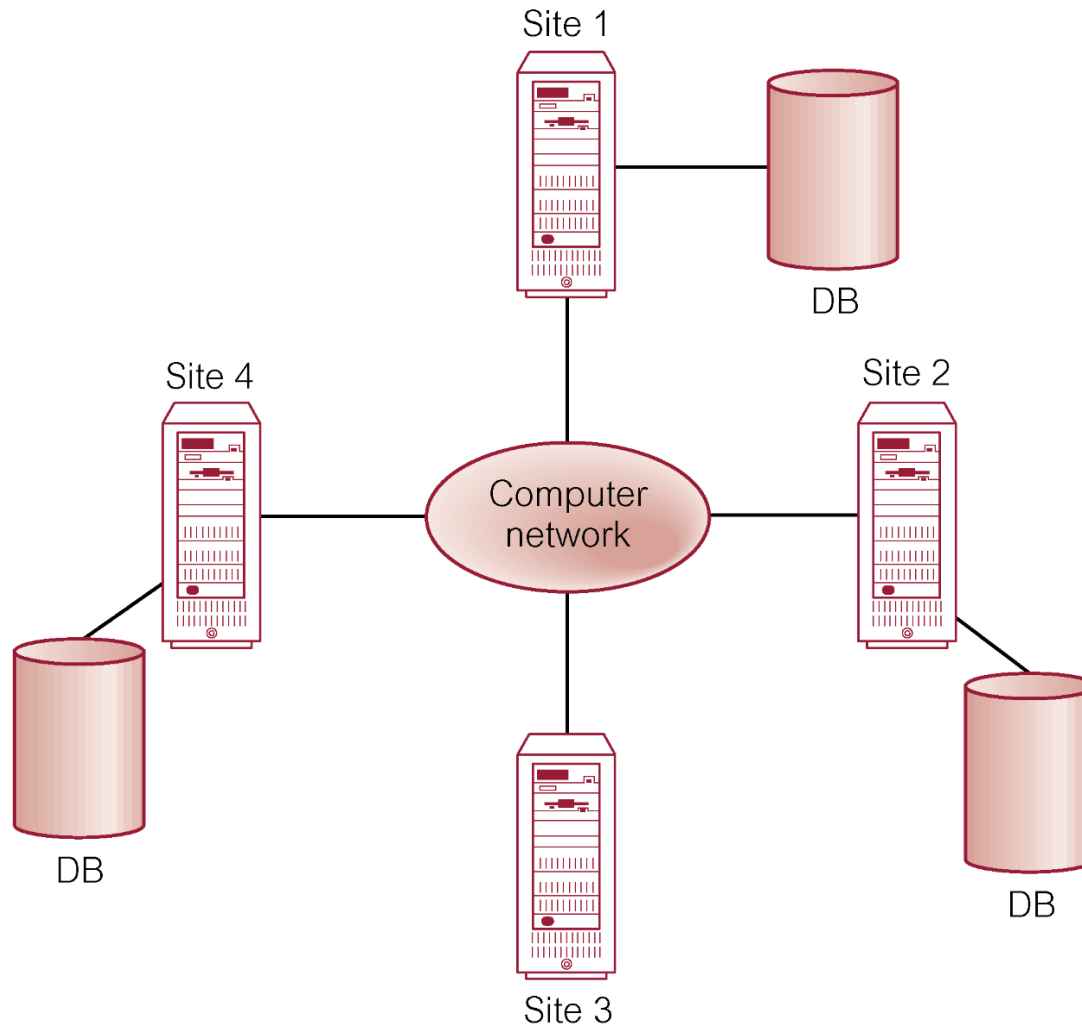
Software system that permits the management of the distributed database and makes the distribution transparent to users.



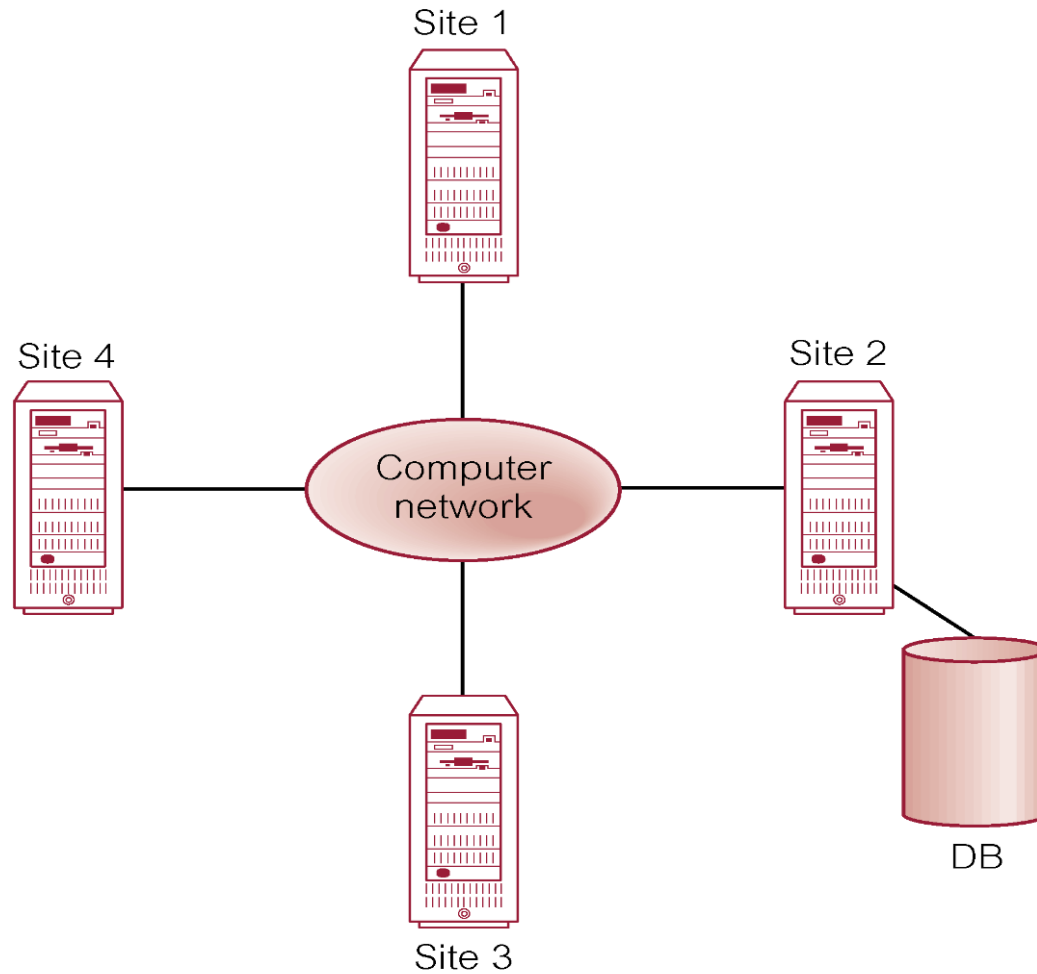
# Important facts

- 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.

# Distributed DBMS



# Distributed Processing



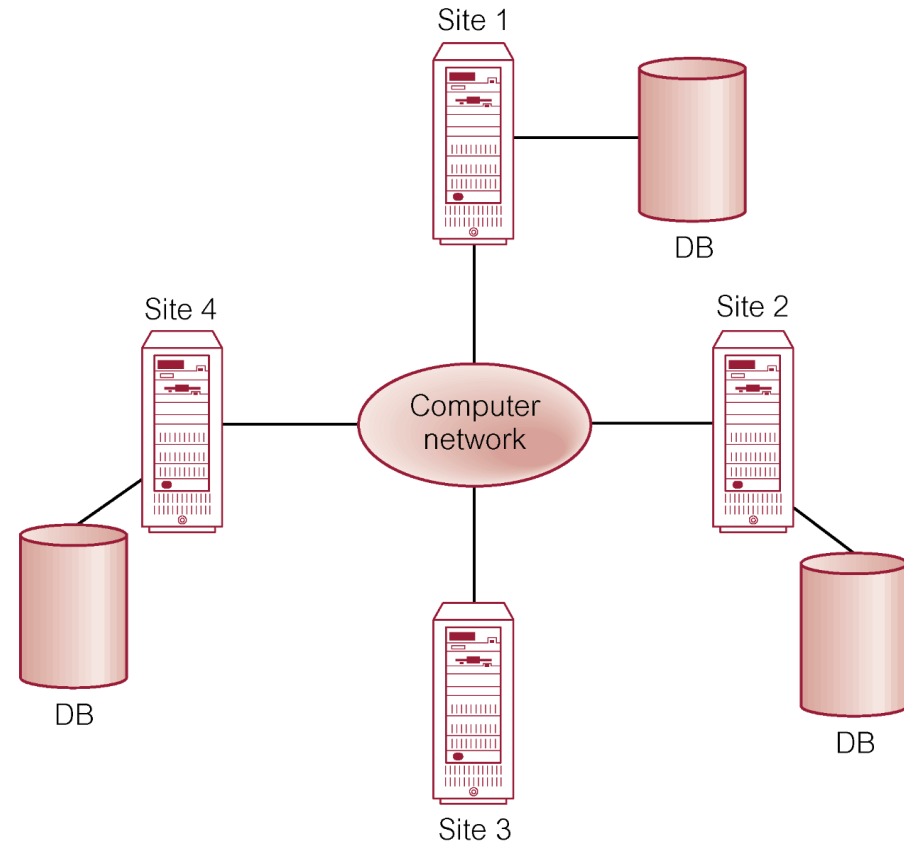
Is distributed processing the same as  
a Distributed DBMS???

**No!**

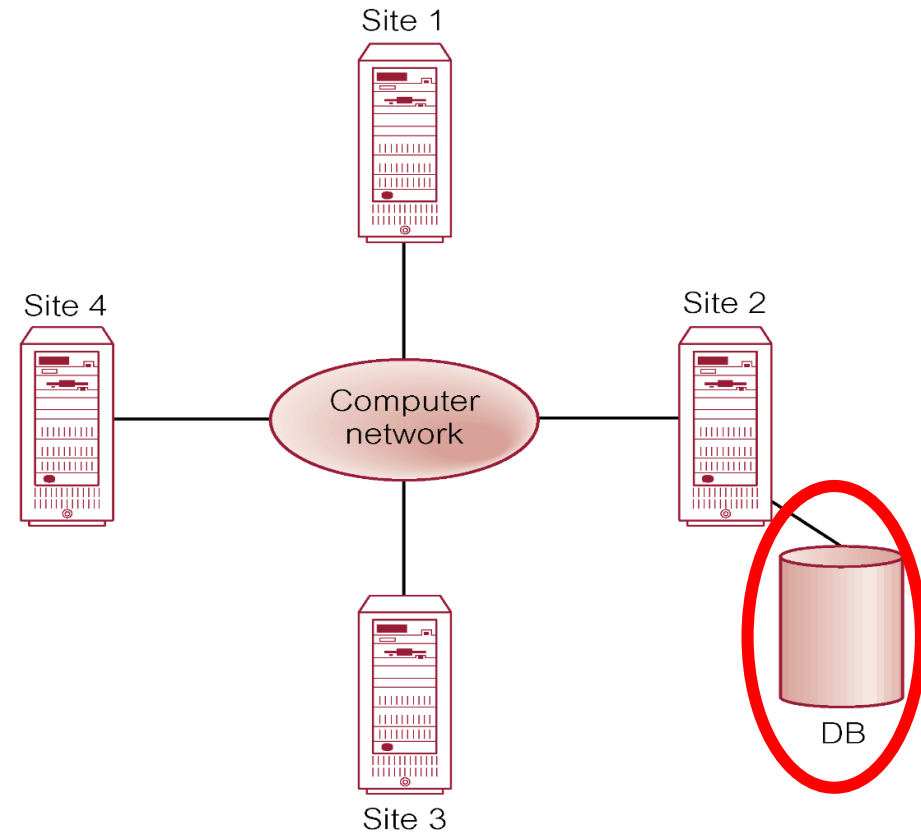
# Where is the difference?

- Distributed Processing:
  - centralized database
  - that can be accessed over a computer network
- Distributed DBMS:
  - Single logical database that is split into fragments
  - These fragments are distributed over a computer network

# Which one is the DDBMS??



**DDBMS**



**Distributed Processing**

# 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 are required to allow for:
  - Different hardware.
  - Different DBMS products.
  - Different hardware and different DBMS products.
- Solution??
  - **Gateways:** a piece of software that causes one DBMS to look like another.

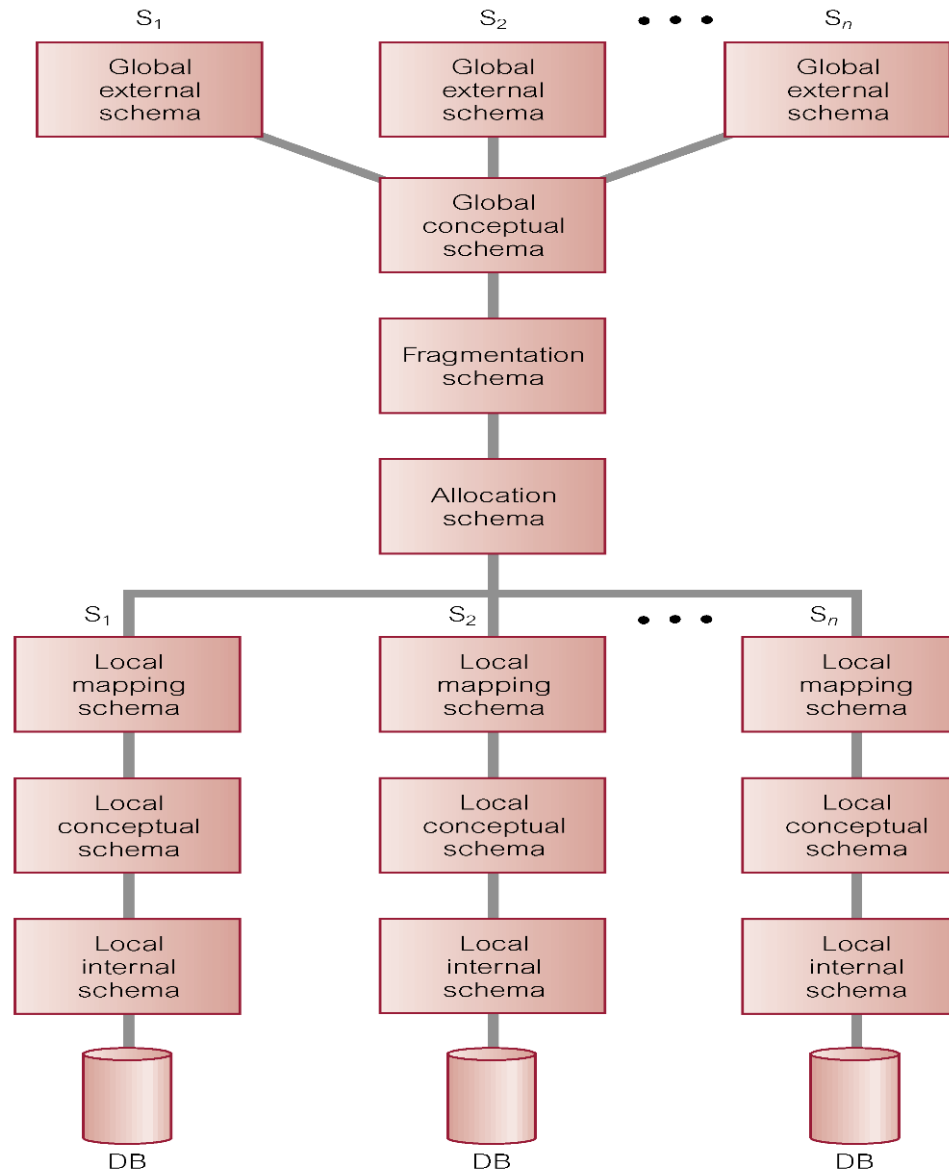
# Functions of a DDBMS

- 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 accepted architecture equivalent to 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

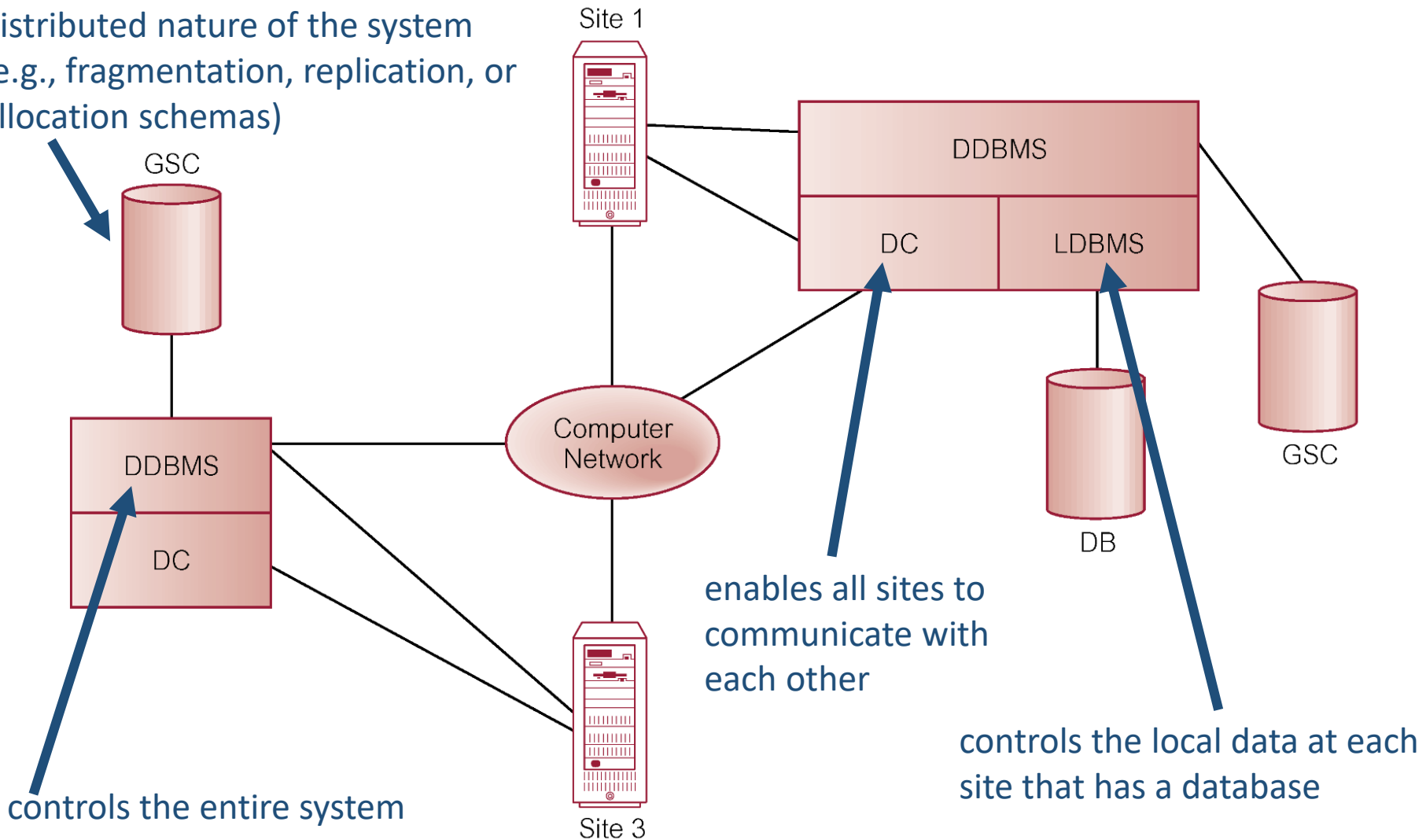


# Component Architecture for a DDBMS

1. Local DBMS (LDBMS): standard DBMS, responsible for controlling the local data at each site that has a database
2. Data Communications (DC): software enabling all sites to communicate with each other.
3. Global System Catalogue (GSC): holds information specific to the distributed nature of the system, such as the fragmentation, replication, and allocation schemas (it can be managed as a distributed database).
4. Distributed DBMS (DDBMS): the controlling unit of the entire system.

# Components of a DDBBMS

information regarding the distributed nature of the system (e.g., fragmentation, replication, or allocation schemas)



# Distributed Database Design – key issues

- 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.



# Distributed Database Design – key issues

- Fragmentation

Relation may be divided into a number of sub-relations, which are then distributed (a fragment is a logical data unit).



- Allocation

Each fragment is stored at site with “optimal” distribution.

- Replication

Copy of fragment may be maintained at several sites.

# Disadvantages of Fragmentation

-  **Performance:** the performance of global applications that require data from several fragments located at different sites may be slower.
-  **Integrity:** Integrity control may be more difficult if data and functional dependencies are fragmented and located at different sites.

# Correctness of fragmentation

- **Completeness** If a relation instance  $R$  is decomposed into fragments  $R_1, R_2, \dots, R_n$ , each data item that can be found in  $R$  must appear in at least one fragment.
- **Reconstruction** It must be possible to define a relational operation that will reconstruct the relation  $R$  from the fragments. This rule ensures that functional dependencies are preserved.
- **Disjointness** If a data item  $d_i$  appears in fragment  $R_i$ , then it should not appear in any other fragment (except in vertical fragmentation where the primary key attributes must be repeated to allow reconstruction). This rule ensures minimal data redundancy.

# Types of Fragmentation

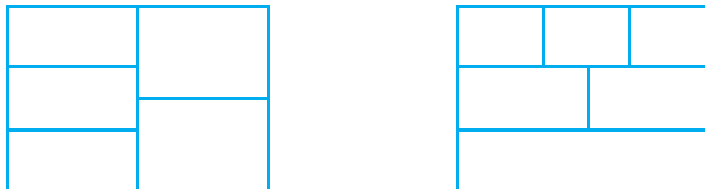
- Horizontal



- Vertical



- Mixed



# Horizontal Fragmentation

- Consists of a subset of the *tuples* of a relation.
- Defined using the Selection operation of relational algebra:  $\sigma_p(R)$   
where  $p$  is a predicate based on one or more attributes of the relation.

# Horizontal Fragmentation example

- Assuming that there are only two property types, Flat and House, the horizontal fragmentation of PropertyForRent by property type can be obtained as follows:

$$P1 = \sigma_{\text{type}='House'}(\text{PropertyForRent})$$

$$P2 = \sigma_{\text{type}='Flat'}(\text{PropertyForRent})$$

- This particular fragmentation strategy may be advantageous if there are separate applications dealing with houses and flats.

# Vertical Fragmentation

- Consists of a subset of *attributes* of a relation.
- Defined using the Projection operation of relational algebra:

$$\Pi_{a_1, \dots, a_n}(R)$$

where  $a_1, \dots, a_n$  are attributes of the relation  $R$ .

# Vertical Fragmentation example

- The *DreamHome* payroll application requires the staff number staffNo and the position, sex, DOB, and salary attributes of each member of staff; the personnel department requires the staffNo, fName, lName, and branchNo attributes.
- The vertical fragmentation of Staff for this example can be obtained as follows:

$$S1 = \Pi_{\text{staffNo, position, sex, DOB, salary}}(\text{Staff})$$

$$S2 = \Pi_{\text{staffNo, fName, lName, branchNo}}(\text{Staff})$$

- The advantage of vertical fragmentation is that the fragments can be stored at the sites that need them. In addition, performance is improved as the fragment is smaller than the original base relation.



# Mixed Fragmentation

- Consists of a horizontal fragment that is vertically fragmented, or a vertical fragment that is horizontally fragmented.
- Defined using Selection and Projection operations of relational algebra:

$$\sigma_p(\Pi_{a_1, \dots, a_n}(R)) \quad \text{or}$$

$$\Pi_{a_1, \dots, a_n}(\sigma_p(R))$$

where  $p$  is a predicate based on one or more attributes of  $R$  and  $a_1, \dots, a_n$  are attributes of  $R$ .

# Mixed Fragmentation example

- Previously we vertically fragmented Staff for the payroll and personnel departments into

$$S1 = \Pi_{\text{staffNo, position, sex, DOB, salary}}(\text{Staff})$$

$$S2 = \Pi_{\text{staffNo, fName, lName, branchNo}}(\text{Staff})$$

- We could now horizontally fragment S2 according to branch number (assume that there are only three branches):

$$S21 = \sigma_{\text{branchNo}='B003'}(S2)$$

$$S22 = \sigma_{\text{branchNo}='B005'}(S2)$$

$$S23 = \sigma_{\text{branchNo}='B007'}(S2)$$

# Strategies for Data Allocation

- **Centralized:** a single database and DBMS stored at one site with users distributed across the network – essentially distributed processing
- **Fragmented (or Partitioned):** Database partitioned into disjoint fragments, each fragment assigned to one site.
- **Complete Replication:** Consists of maintaining complete copy of database at each site.
- **Selective Replication:** Combination of fragmentation, replication, and centralization.

# Comparison of Strategies for Data Distribution

## Comparison of strategies for data allocation

|                       | Locality of reference | Reliability and availability  | Performance               | Storage costs | Communication costs           |
|-----------------------|-----------------------|-------------------------------|---------------------------|---------------|-------------------------------|
| Centralized           | Lowest                | Lowest                        | Unsatisfactory            | Lowest        | Highest                       |
| Fragmented            | High <sup>a</sup>     | Low for item; high for system | Satisfactory <sup>a</sup> | Lowest        | Low <sup>a</sup>              |
| Complete replication  | Highest               | Highest                       | Best for read             | Highest       | High for update; low for read |
| Selective replication | High <sup>a</sup>     | Low for item; high for system | Satisfactory <sup>a</sup> | Average       | Low <sup>a</sup>              |

---

<sup>a</sup> Indicates subject to good design.

# Distributed Database Design Methodology

- Normalization to produce a design for the global relations.
- Examine topology of system to determine where databases will be located.
- Analyse most important transactions and identify appropriateness of horizontal/vertical fragmentation.
- Decide which relations are not to be fragmented.
- Examine relations on 1 side of relationships and determine a suitable fragmentation schema. Relations on many sides may be suitable for horizontal derived fragmentation.
- Check for situations where either vertical or mixed fragmentation would be appropriate (that is, where transactions require access to a subset of the attributes of a relation).

# Does the user need to see the complexity of a DDMS??



Hiding DDBMS complexity, is called **transparency**

# Transparency in a DDBMS

- User doesn't want/need the details of the DDBMS implementation. Using distributed database should feel like a centralized database.
- Distribution Transparency
- Transaction Transparency
- Performance Transparency
- DBMS Transparency

# Distribution Transparency

- The database feels as a single, logical entity.
- The user ignores how
  - data is fragmented (fragmentation transparency),
  - location of data items (location transparency),
  - Replication of fragments (replication transparency)
- No distribution transparency = the user needs to know how the data is fragmented and the location of fragments (i.e., local mapping transparency). This is undesirable.





# Naming Transparency

- Each item in a DDB must have a unique name.
- DDBMS must ensure that two sites do not create a database object with the same name.
- One solution is to create central name server.

## Implications:

- 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.

# Naming Transparency, solutions

- **Alternative solution:** prefix the object with identifier of the site that created it.
  - For example, Branch created at site S1 might be named S1.BRANCH.
  - Also need to identify each fragment and its copies.
  - Thus, copy 2 of fragment 3 of Branch created at site S1 might be referred to as S1.BRANCH.F3.C2.
- Implications:  
low distribution transparency.

# Naming Transparency, more solutions

- Solve both problems: uses *aliases* for each database object.
- Thus, S1.BRANCH.F3.C2 might be known as LocalBranch by user at site S1.
- DDBMS has task of mapping an alias to appropriate database object.

# Transaction Transparency

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

# Example - Distributed Transaction

- T prints out names of all staff. Staff has different fragments in different site, namely S3, S5, S7.  
Define three subtransactions TS3, TS5, and TS7 to represent agents at sites 3, 5, and 7.

| Time  | $T_{s_3}$           | $T_{s_5}$           | $T_{s_7}$           |
|-------|---------------------|---------------------|---------------------|
| $t_1$ | begin_transaction   | begin_transaction   | begin_transaction   |
| $t_2$ | read(fName, lName)  | read(fName, lName)  | read(fName, lName)  |
| $t_3$ | print(fName, lName) | print(fName, lName) | print(fName, lName) |
| $t_4$ | end_transaction     | end_transaction     | end_transaction     |

# Concurrency Transparency

- All transactions must execute independently and be logically consistent with the results obtained if the transactions are 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 subtransactions of global transaction.

# Concurrency Transparency

- Replication makes concurrency more complex.
- If a copy of a replicated data item is updated, the 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 the transaction is delayed until site is reachable!!
- Could limit update propagation to only those sites currently available. Remaining sites updated when they become available again.

# Failure Transparency

- DDBMS must ensure atomicity and durability of global transaction.
- Means ensuring that subtransactions of global transaction either all commit or all abort.
- Thus, DDBMS must synchronize global transaction to ensure that all subtransactions 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.
- Distributed Query Processor (DQP) maps data request into ordered sequence of operations on local databases.
- DQP has to decide:
  - which fragment to access;
  - which copy of a fragment to use;
  - which location to use.

How does  
DQP  
decide?



# 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.

# Performance Transparency - Example

|                           |                           |
|---------------------------|---------------------------|
| Property(propNo, city)    | 10000 records in London   |
| Client(clientNo,maxPrice) | 100000 records in Glasgow |
| Viewing(propNo, clientNo) | 1000000 records in London |

List the properties in Aberdeen that have been viewed by clients who have a maximum price limit greater than £200,000:

```
SELECT p.propNo
FROM Property p, Client c, Viewing v
WHERE c.clientNo = v.clientNo AND
      p.propNo = v.propNo AND
      p.city='Aberdeen' AND c.maxPrice > 200000;
```

# Performance Transparency - Example

- Assume:
  - Each tuple in each relation is 100 characters long.
  - 10 renters with maximum price greater than £200,000.
  - 100 000 viewings for properties in Aberdeen.
  - Computation time negligible compared to communication time.
- Data transmission rate is 10,000 characters per second and 1 second access delay to send a message from one site to another.

# Can you calculate it?

Communication Time =

$$C_0 + (\text{no\_of\_bits\_in\_message} / \text{transmission\_rate})$$

$C_0$  is access delay to send a message from one site to another

- **Strategy 1:** Move the Client relation to London and process query there:

$$\text{Time} = 1 + (100\,000 * 100/10\,000) \cong 16.7 \text{ minutes}$$

- **Strategy 2:** Move the Property and Viewing relations to Glasgow and process query there:

$$\text{Time} = 2 + [(1\,000\,000 + 10\,000) * 100/10\,000] \cong 28 \text{ hours}$$

- **Strategy 3:** Join the Property and Viewing relations at London, select tuples for Aberdeen properties and then, for each of these tuples in turn, check at Glasgow to determine if the associated client's maxPrice > £200,000. The check for each tuple involves two messages: a query and a response.

$$\text{Time} = 100\,000 * (1 + 100/10\,000) + 100\,000 * 1 \cong 2.3 \text{ days}$$



- **Strategy 4:** Select clients with  $\text{maxPrice} > \text{£}200,000$  at Glasgow and, for each one found, check at London to see if there is a viewing involving that client and an Aberdeen property. Again, two messages are needed:

$$\text{Time} = 10 * (1 + 100/10\,000) + 10 * 1 \cong 20 \text{ seconds}$$

- **Strategy 5:** Join Property and Viewing relations at London, select Aberdeen properties, project result over propertyNo and clientNo, and move this result to Glasgow for matching with maxPrice > £200,000. For simplicity, we assume that the projected result is still 100 characters long:  
Time =  $1 + (100\ 000 * 100/10\ 000) \cong 16.7$  minutes
- **Strategy 6:** Select clients with maxPrice > £200,000 at Glasgow and move the result to London for matching with Aberdeen properties:  
Time =  $1 + (10 * 100/10\ 000) \cong 1$  second

# Performance Transparency - Example

## Comparison of distributed query processing strategies

| Strategy   | Time         |
|--|--------------|
| (1) Move Client relation to London and process query there   | 16.7 minutes |
| (2) Move Property and Viewing relations to Glasgow and process query there   | 28 hours     |
| (3) Join Property and Viewing relations at London, select tuples for Aberdeen properties, and for each of these in turn, check at Glasgow to determine if associated maxPrice > £200,000             | 2.3 days     |
| (4) Select clients with maxPrice > £200,000 at Glasgow and for each one found, check at London for a viewing involving that client and an Aberdeen property  | 20 seconds   |
| (5) Join Property and Viewing relations at London, select Aberdeen properties, and project result over propertyNo and clientNo and move this result to Glasgow for matching with maxPrice > £200,000 | 16.7 minutes |
| (6) Select clients with maxPrice > £200,000 at Glasgow and move the result to London for matching with Aberdeen properties   | 1 second     |

---

# Advantages of DDBMSs

- Reflects organizational structure
- Improved 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

# Summary

- Database architectures
- Distributed database, Distributed DBMS
- Functions and architecture for a DDBMS
- Distributed database design: fragmentation, allocation and replication
- Fragmentation
- Levels of transparency in DDBMS
  - Distribution transparency
  - Transaction transparency
  - Performance transparency
  - DBMS transparency
- Advantages and disadvantages of distributed databases