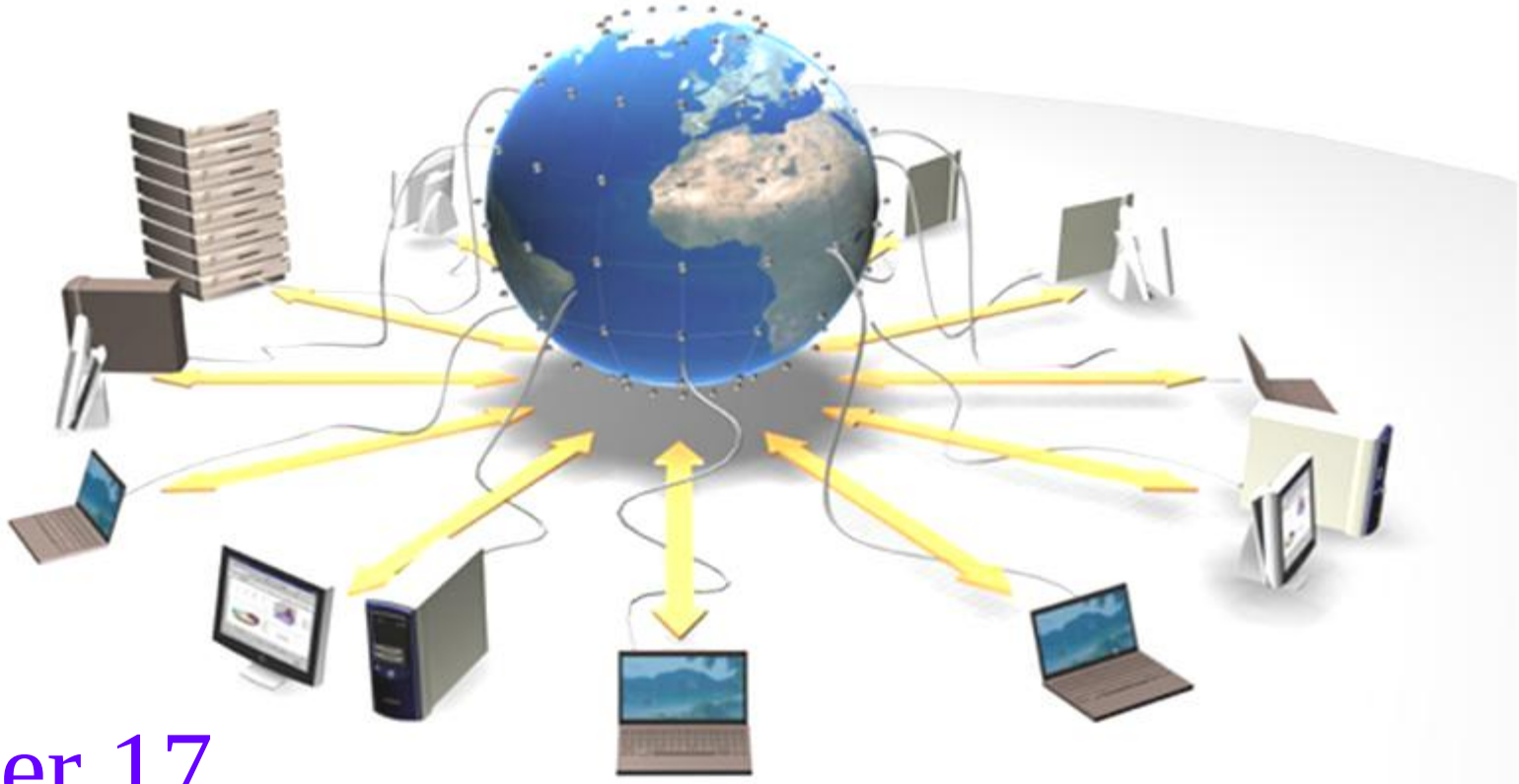
Client-Server Processing, Parallel Database Processing, *and* Distributed Databases



Chapter 17

Outline

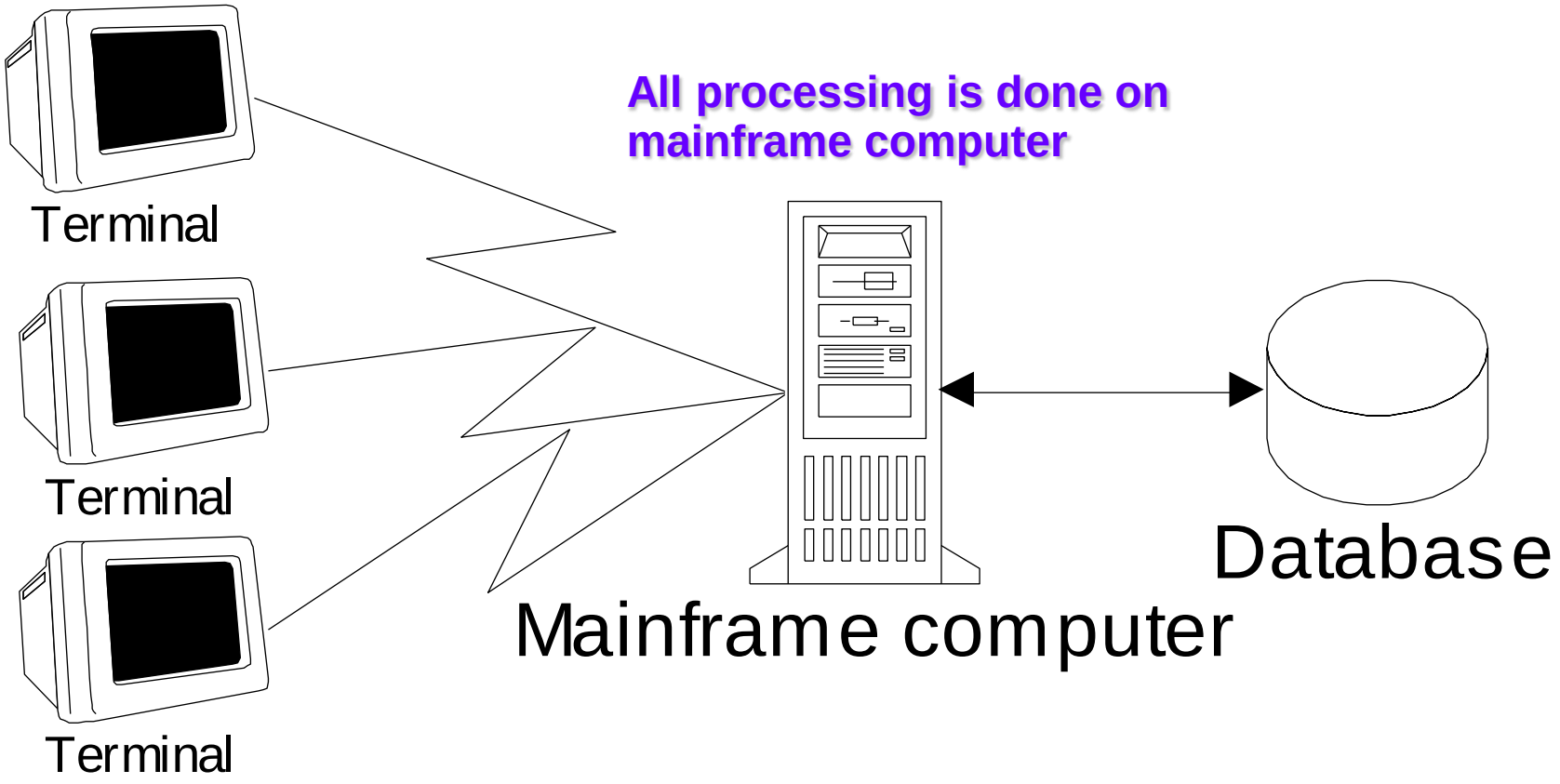
- Overview
- Client-Server Database Architectures
- Parallel Database Architectures
- Architectures for Distributed Database Management Systems
- Transparency for Distributed Database Processing
- Distributed Database Processing

Evolution of Distributed Processing and Distributed Data

- Need to share resources across a network
- Timesharing (1970s)
- Remote procedure calls (1980s)
- Client-server computing (1990s)
  - A client is a program that makes requests to a server.
  - The server performs the request and communicates the result to the client.

Timesharing Network

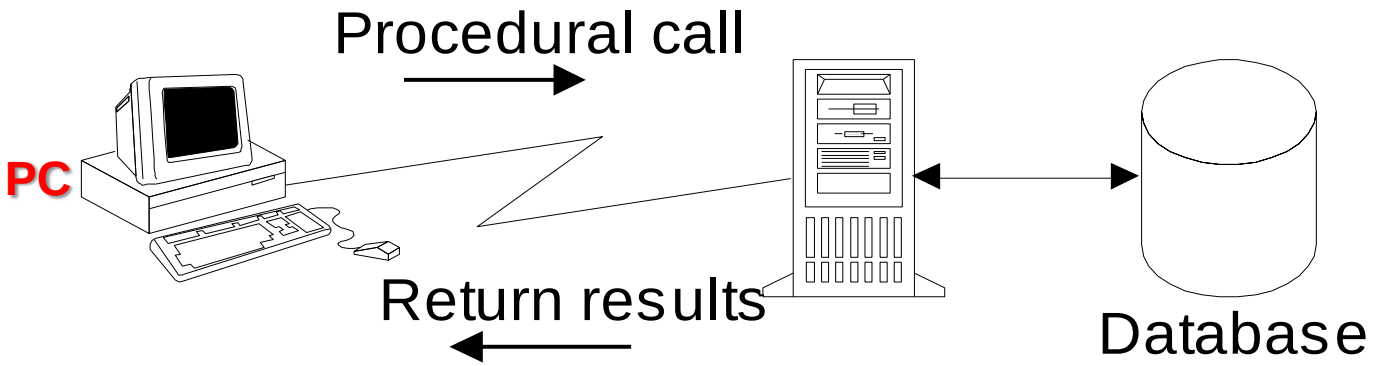
dump terminal



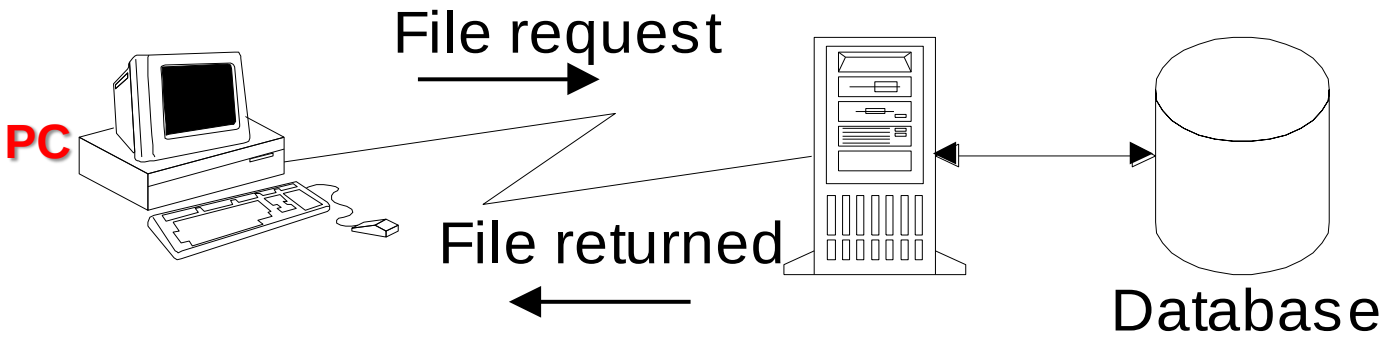
Simple Resource Sharing

simple ways to share processing and data in a network

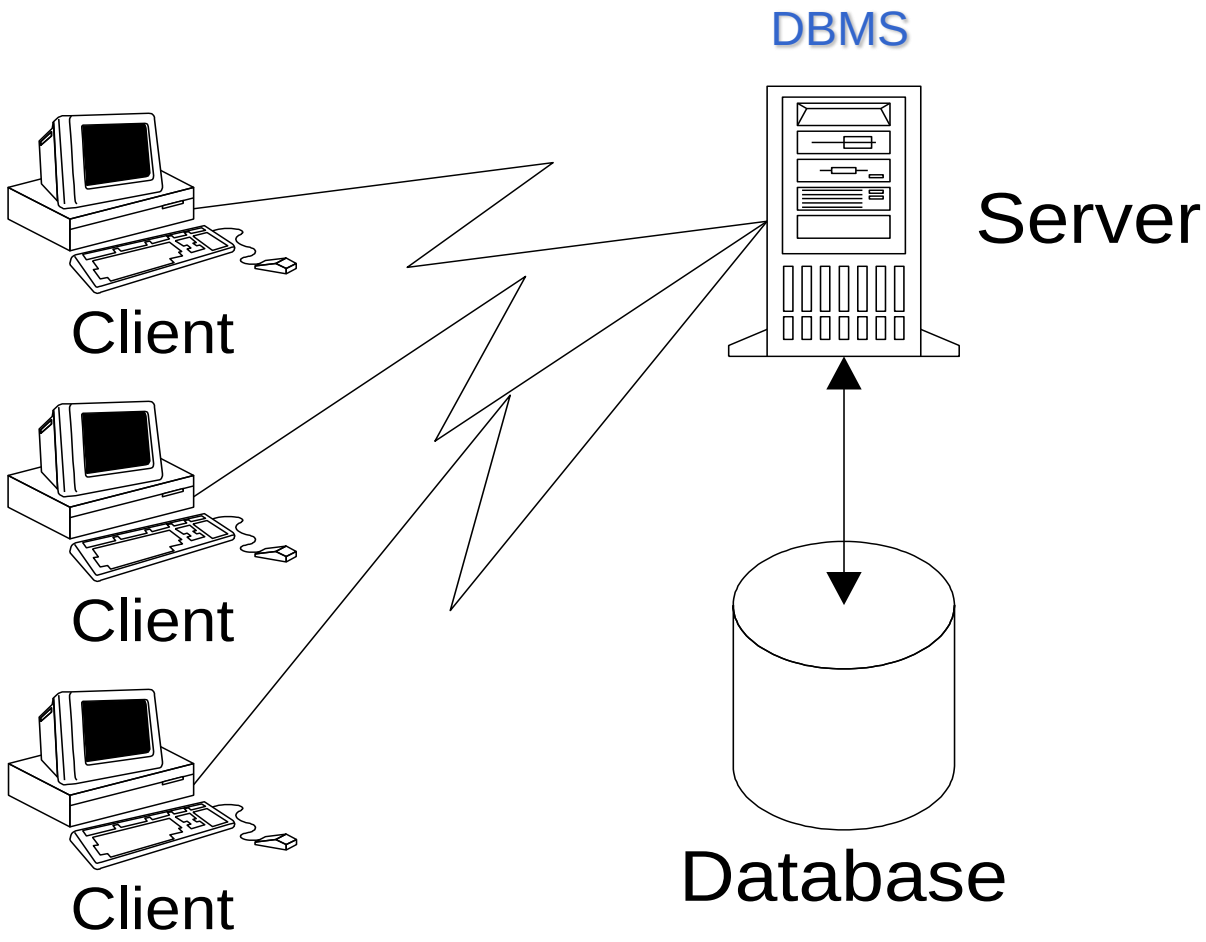
(a) Remote procedural call



(b) File sharing



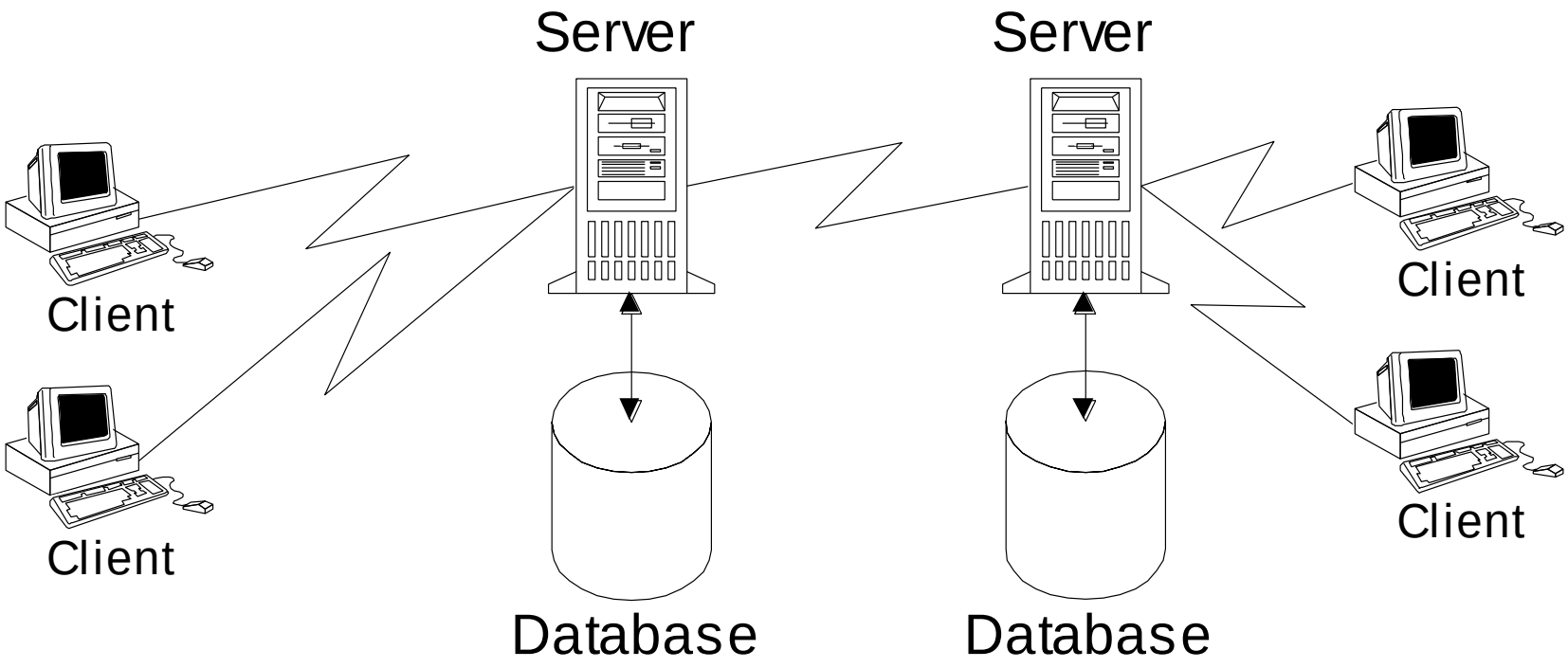
Client-Server Processing



divide the work between clients processing on PCs and a server processing on a separate computer

Distributed processing and data

client-server approach



Motivation for Client-Server Processing

- Flexibility: the ease of maintaining and adapting a system
- Scalability: the ability to support scalable growth of hardware and software capacity
- Interoperability: open standards that allow two or more systems to exchange and use software and data

Motivation for Parallel Database Processing

Multiple Processors(อาจมีคอมฯ >1) and Multiple Discs

- Scaleup: increased work that can be accomplished
- Speedup: decrease in time to complete a task
- Availability: increased accessibility of system
  - Highly available: little downtime
  - Fault-tolerant: no downtime

Motivation for Distributed Data

- **Data control:** locate data to match an organization's structure
- **Communication costs:** locate data close to data usage to lower communication cost and improve performance
- **Reliability:** increase data availability by replicating data at more than one site

Summary of Distributed Processing and Data

Technology	Advantages	Disadvantages
Client-server processing	Flexibility, interoperability, scalability	High complexity, high development cost, possible interoperability problems
Parallel database processing	Speedup, scaleup, availability, scalability for predictive performance improvements	Possible interoperability problems, high cost
Distributed databases	Local control of data, improved performance, reduced communication costs, increased reliability	High complexity, additional security concerns

Client-Server Database Architectures

- Client-Server Architecture is an arrangement of components (clients and servers) among computers connected by a network.
- A client-server architecture supports efficient processing of messages (requests for service) between clients and servers.

Design Issues

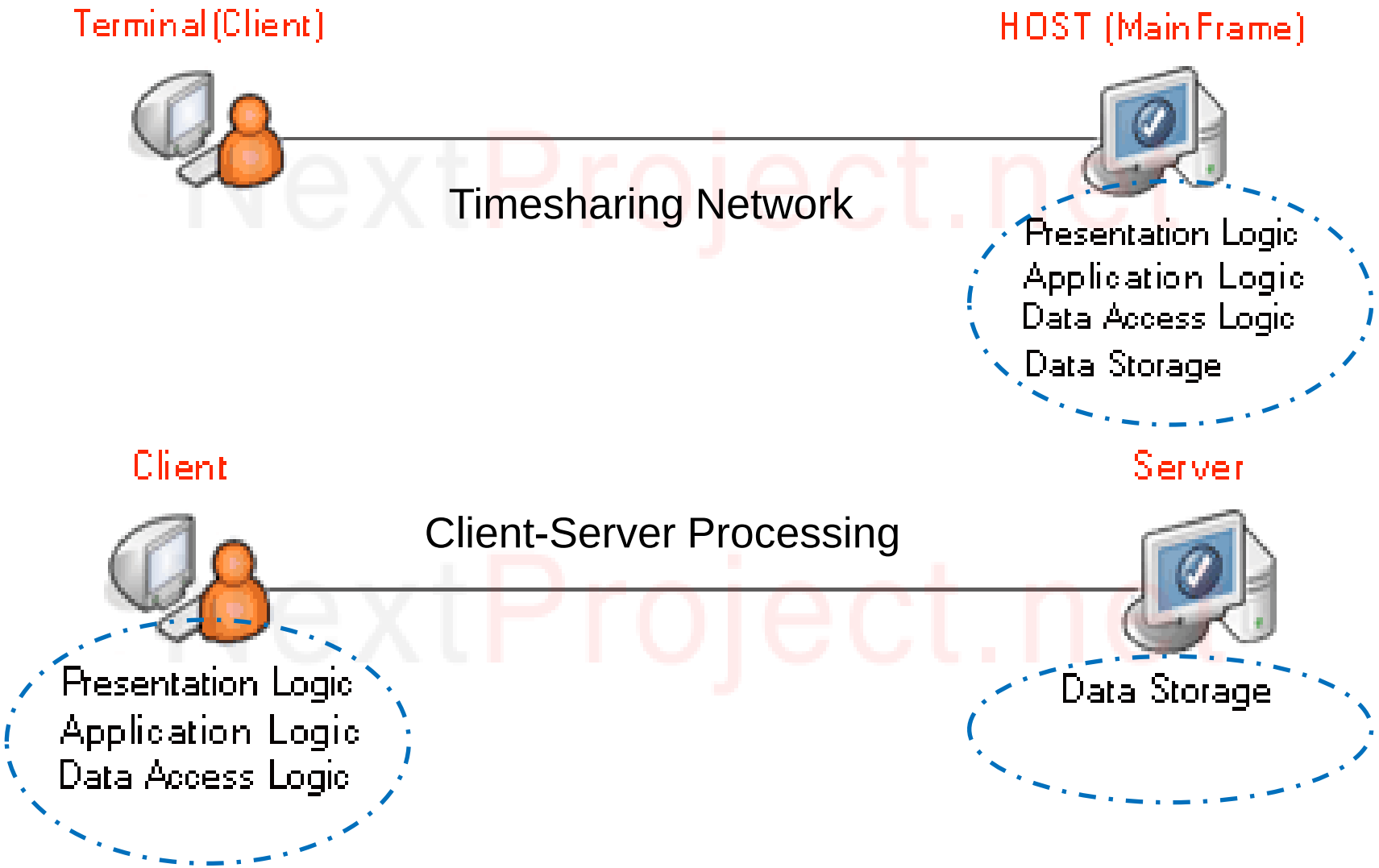
- **Division of processing:** the allocation of tasks to clients and servers.
- **Process management:** interoperability among clients and servers and efficiently processing messages between clients and servers.
- **Middleware:** software for process management

Tasks to Distribute

performed on a client or remotely on a server

- Presentation: code to maintain the graphical user interface
- Validation: code to ensure the consistency of the database and user inputs
- Business logic: code to perform business functions
- Workflow: code to ensure completion of business processes
- Data access: code to extract data to answer queries and modify a database

Tasks to Distribute

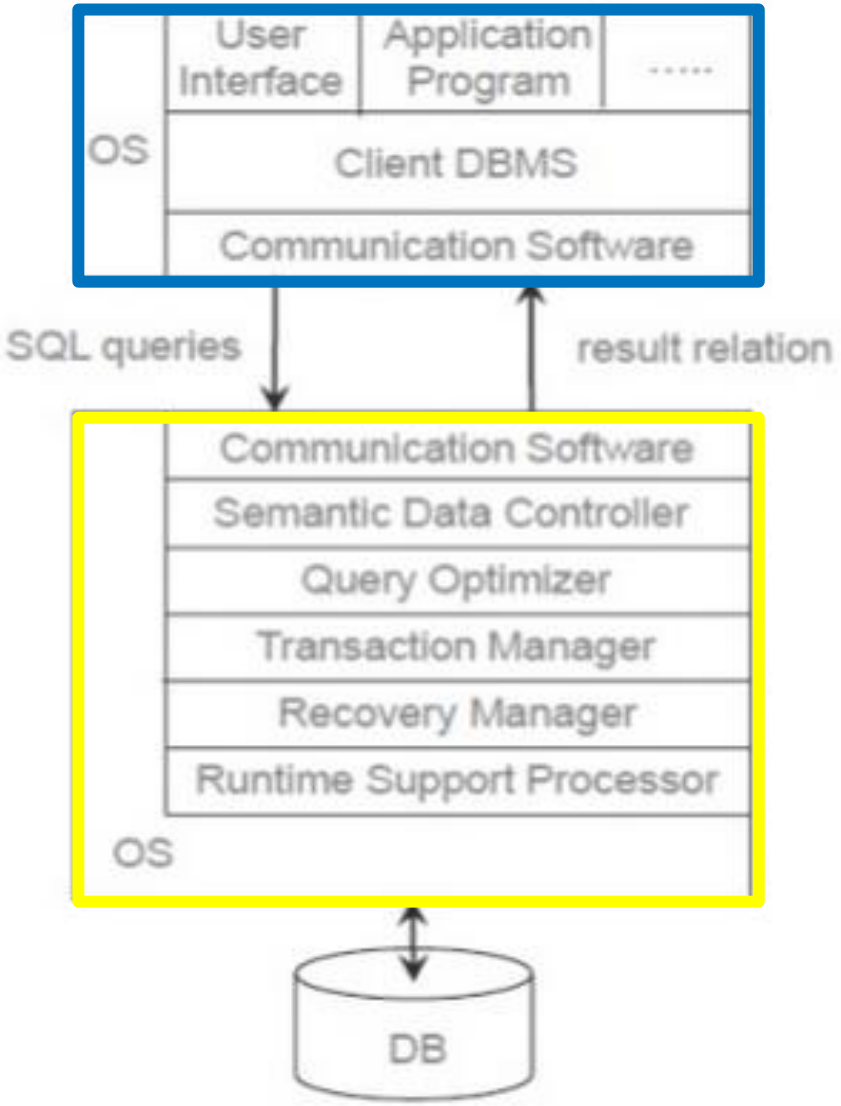


Client-Server Architecture

Architecture

Client

Server

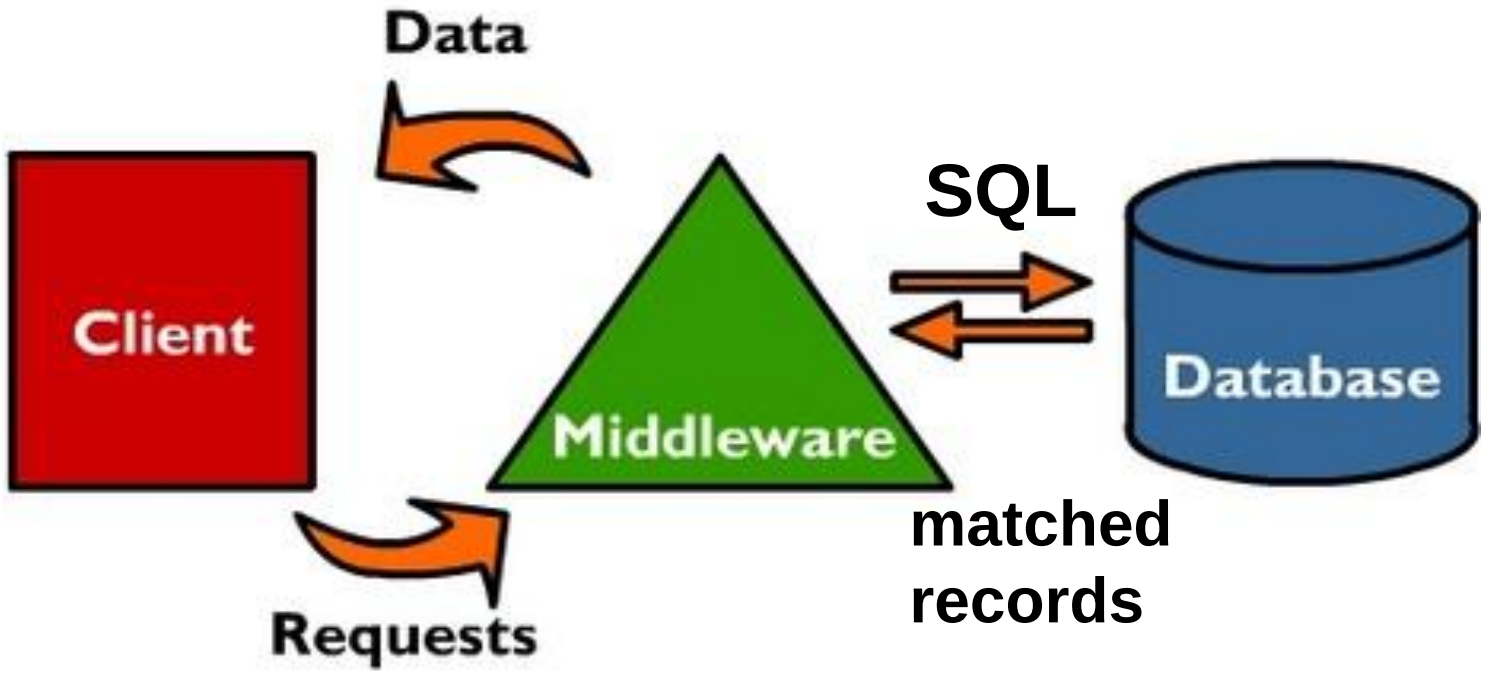


Middleware→ Interoperability

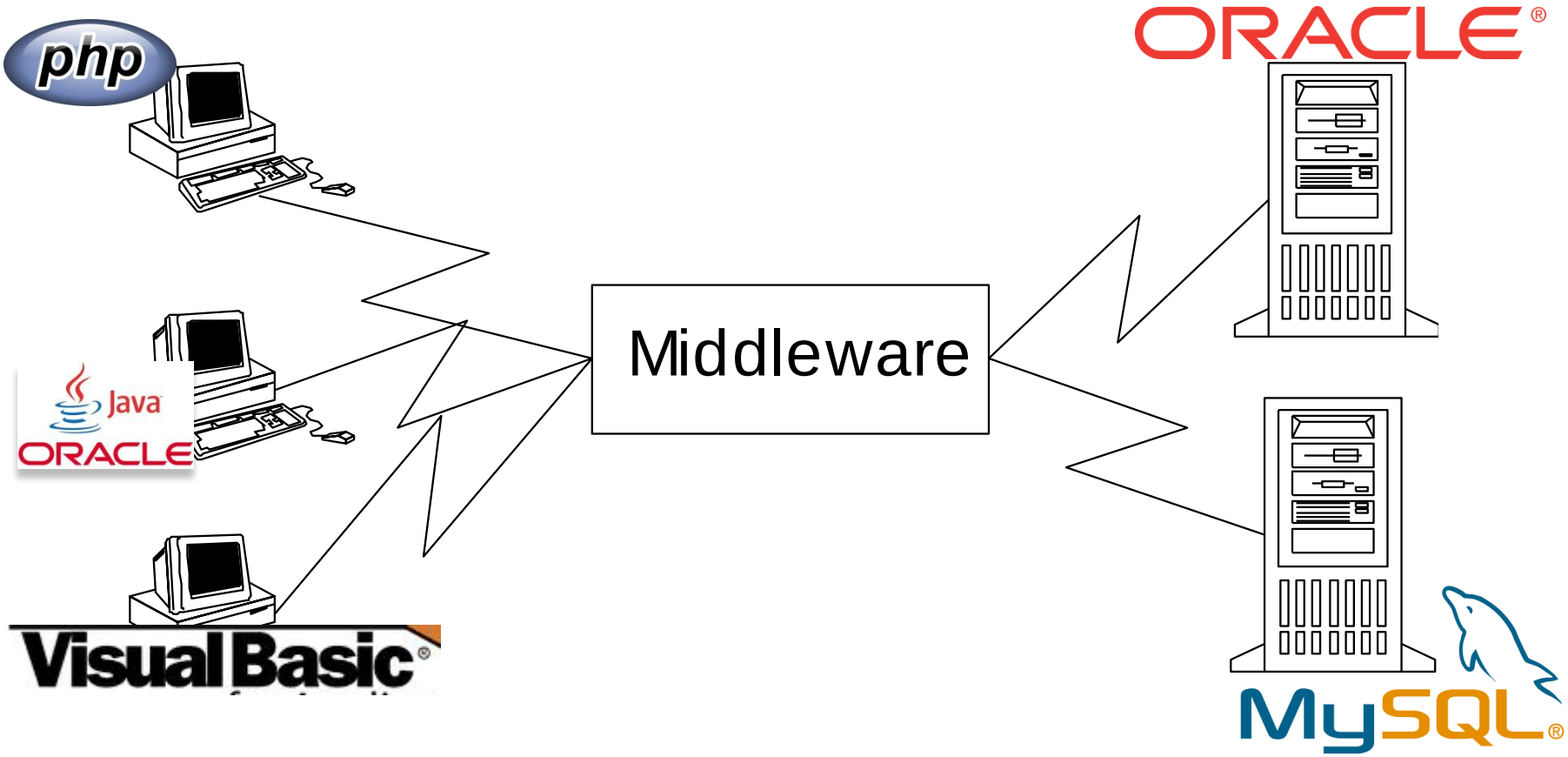
- A software component that performs ***process management***.
- Allow clients and servers to exist on ***different*** platforms.
- Allows servers to efficiently process messages from ***a large number of clients***.
- Often located on ***a dedicated computer***.

e.g., ODBC, JDBC

Middleware


Data
Client
Middleware
SQL
Database
matched
records
Requests

Client-Server Computing with Middleware

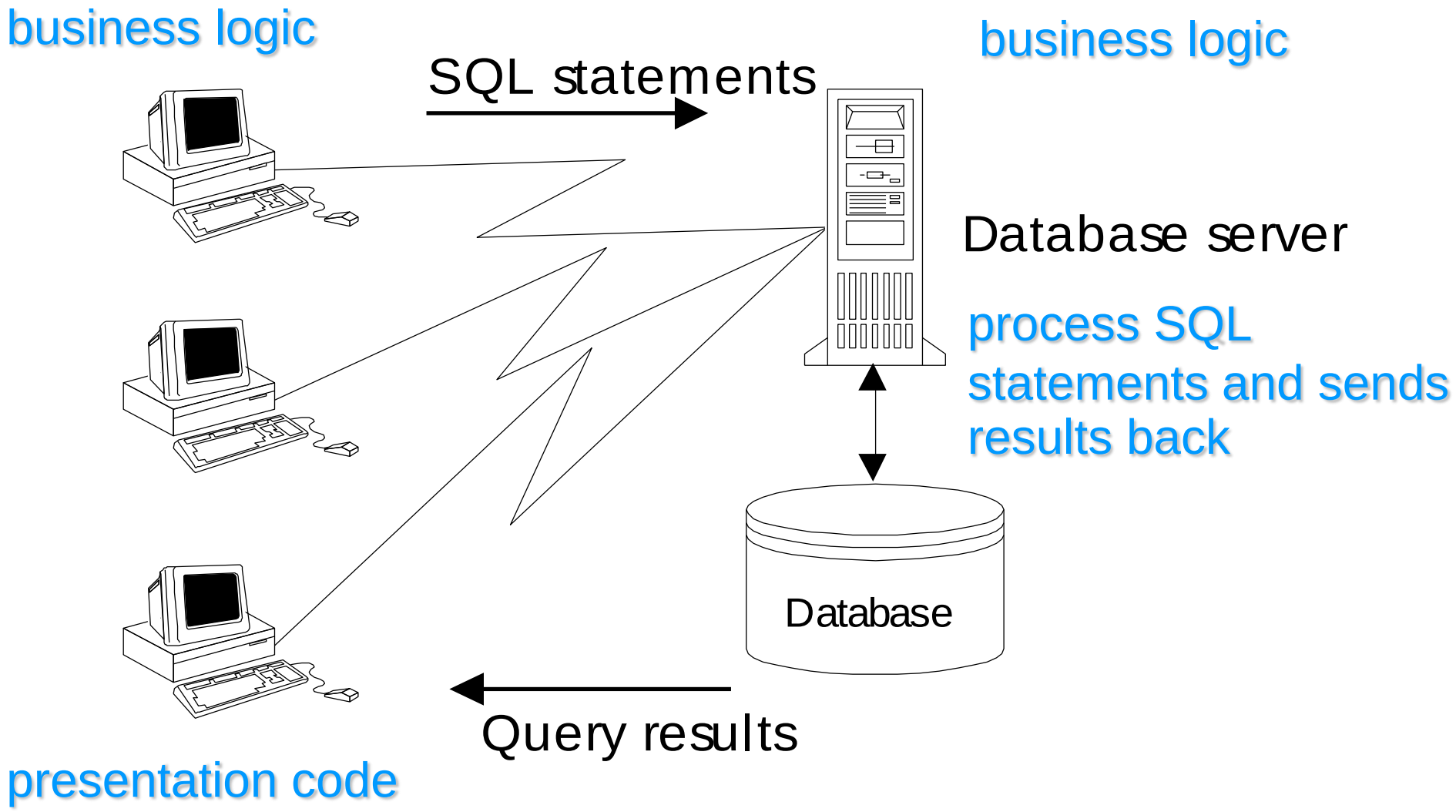


middleware allowing clients and servers to communicate without regard to the underlying platforms of the clients and the servers

Data access middleware:

- provide a uniform interface to relational and non relational data using SQL (ODBC,JDBC)
- Requests to access data from a DBMS are sent to a data access driver rather than directly to the DBMS.
- The data access driver converts the SQL statement into the SQL supported by the DBMS and then routes the request to the DBMS.
- The data access driver adds another layer of overhead between an application and a DBMS.
  - However, the data access driver supports independence between an application and the proprietary SQL supported by a DBMS vendor.

Two-Tier Architecture

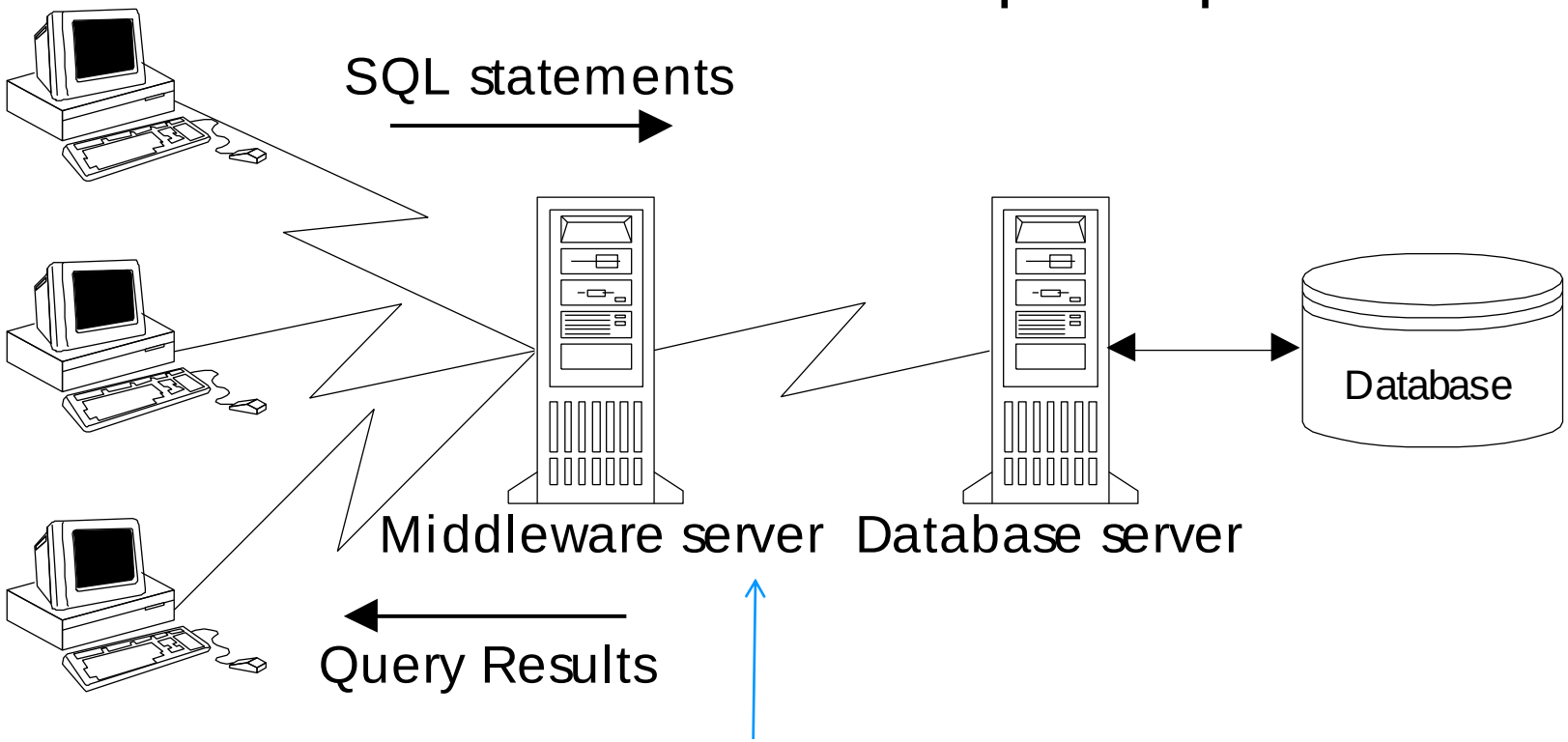


Two-Tier Architecture

- A PC client and a database server interact directly to request and transfer data.
- The PC client contains the user interface code.
- The server contains the data access logic.
- The PC client and the server share the validation and business logic.

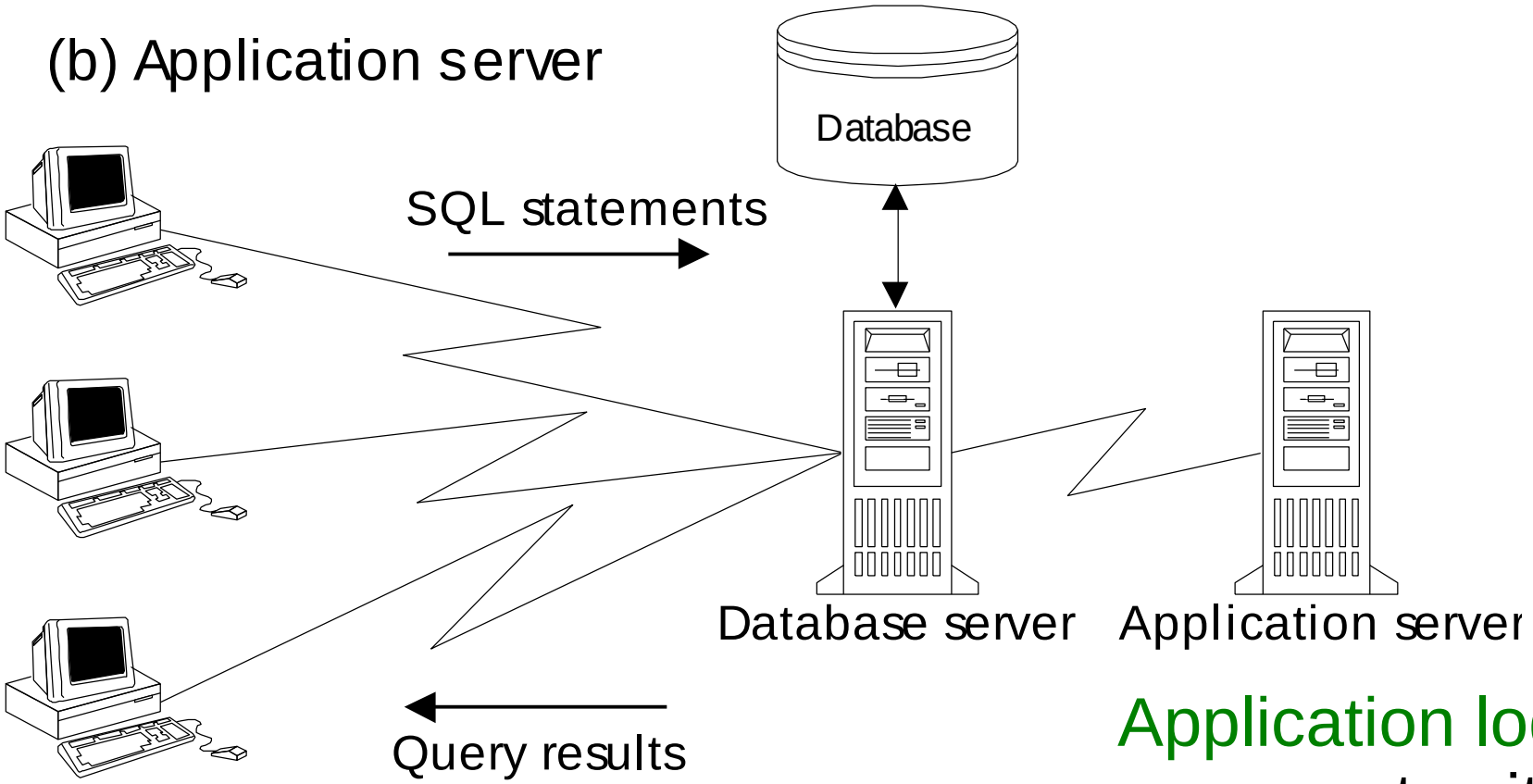
Three-Tier Architecture (Middleware Server)

improve performance



usually consists of a transaction-processing monitor or message-oriented middleware

Three-Tier Architecture (Application Server)



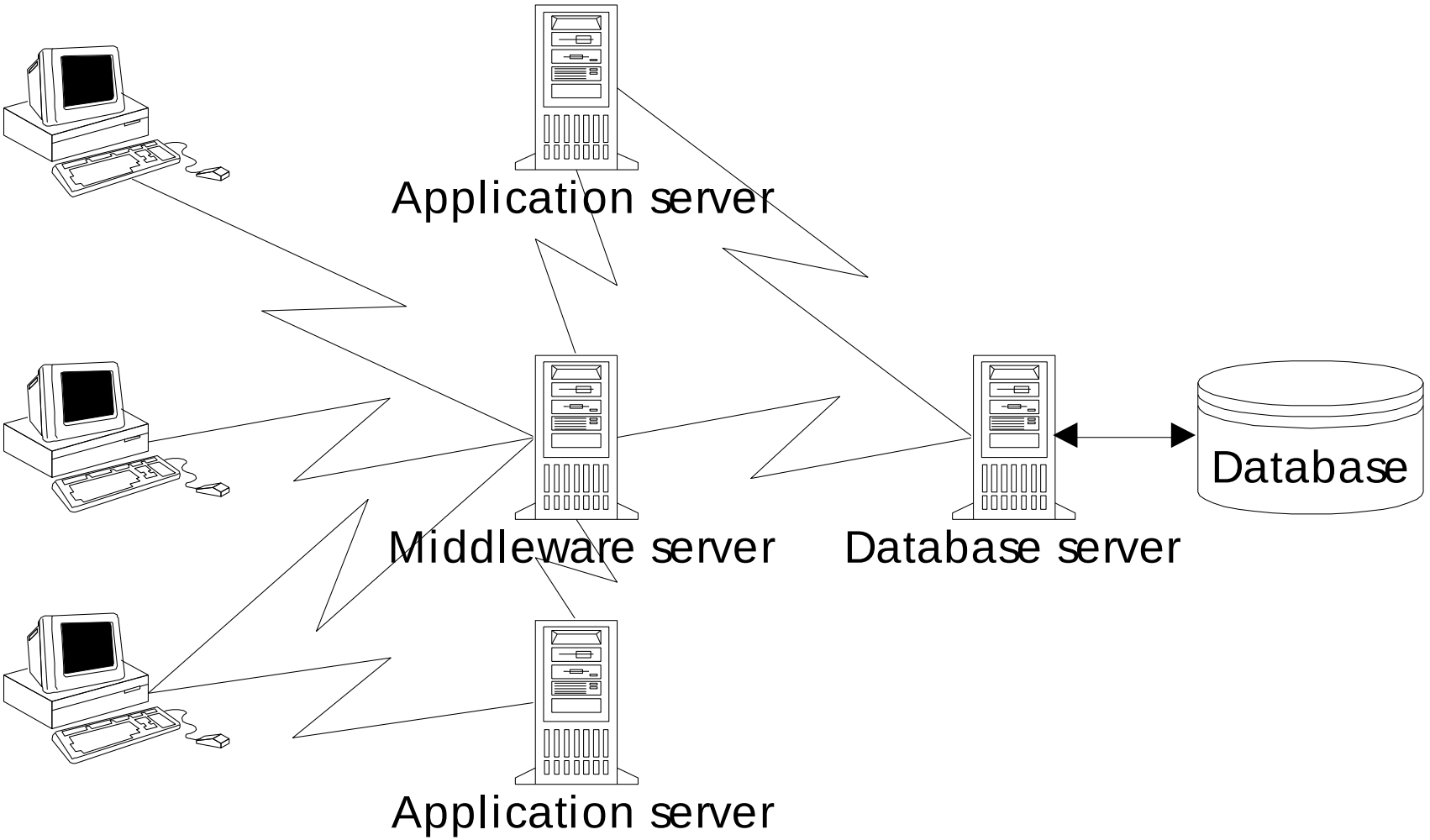
Three-Tier Architecture

- To improve performance, the three-tier architecture adds another server layer either by a middleware server or an application server.
- The *additional server software* can reside on a separate computer.
- Alternatively, the additional server software can be distributed between the database server and PC clients.

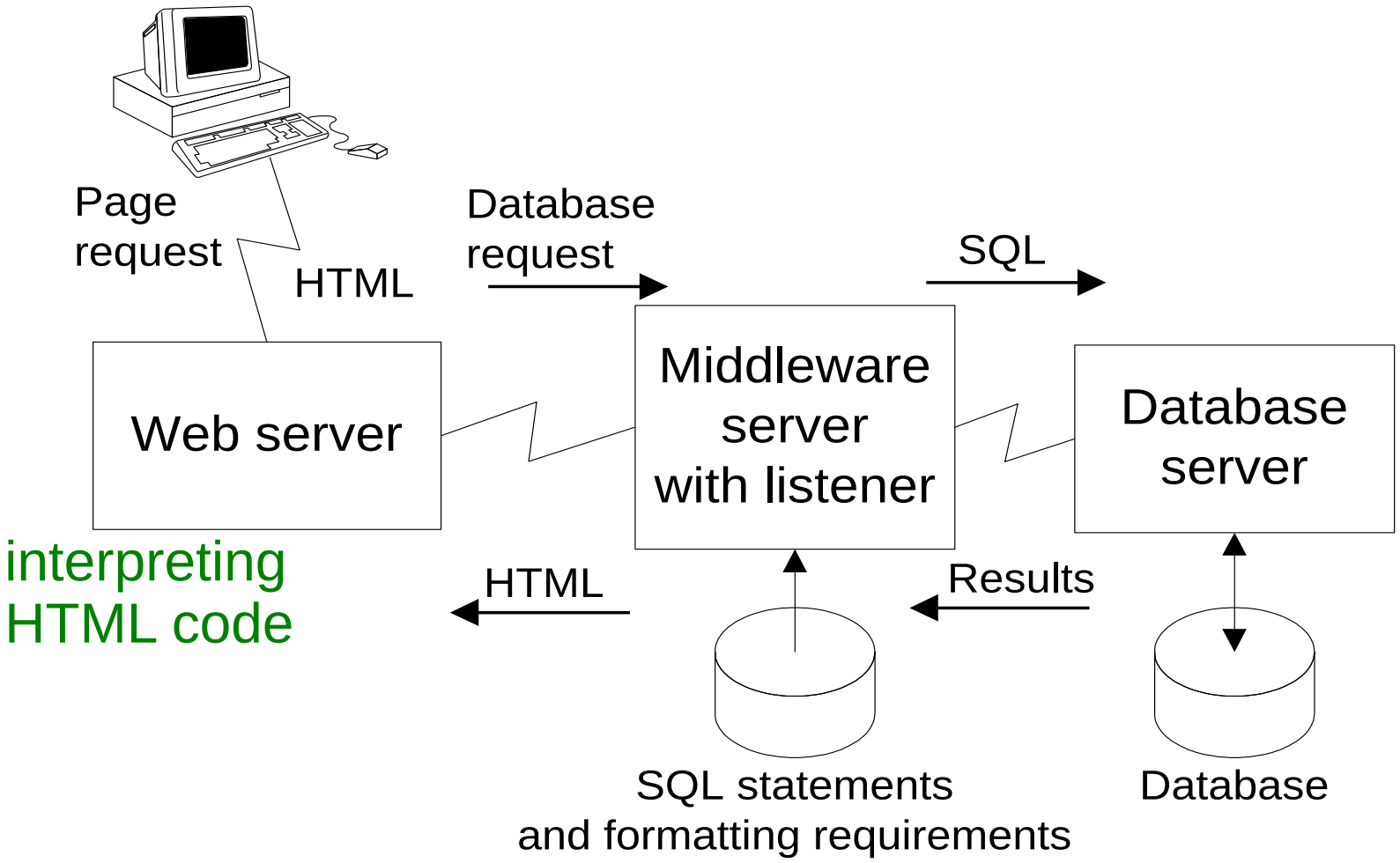
Multiple-Tier Architecture

- A client-server architecture with more than three layers: a PC client, a backend database server, an intervening middleware server, and application servers.
- Provides more flexibility on division of processing
- The application servers perform business logic and manage specialized kinds of data such as images.

Multiple-Tier Architecture



Multiple-Tier Architecture with Web Server



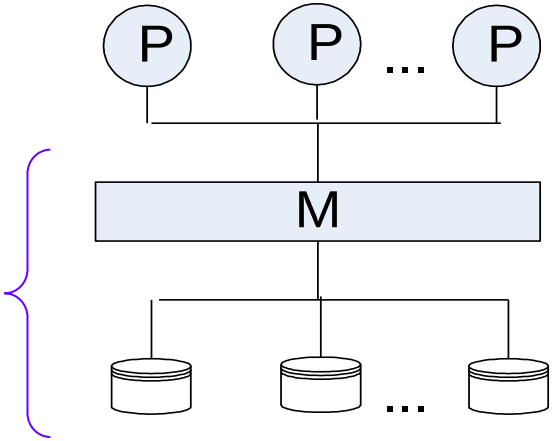
Parallel DBMS

- Uses a collection of *resources (processors, disks, and memory)* to perform work in parallel
- Divide work among resources to achieve desired performance *(scaleup and speedup)* and availability.
- Uses high speed network, operating system, and storage system
- Purchase decision involves more than parallel DBMS

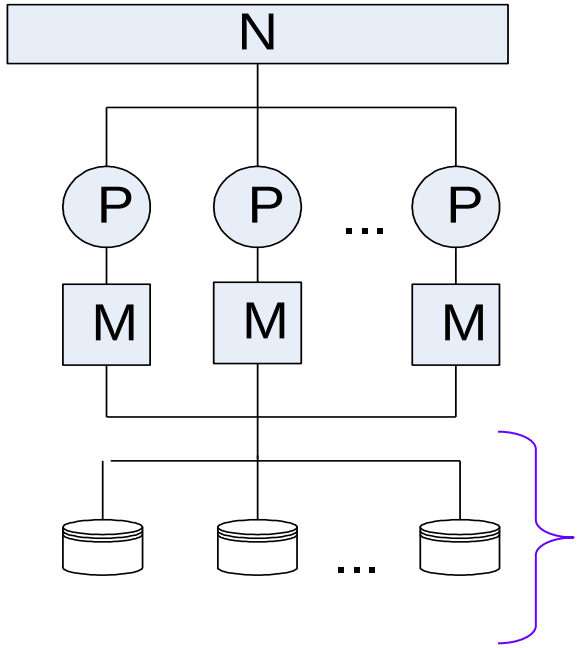
Basic Architectures

(a) SE

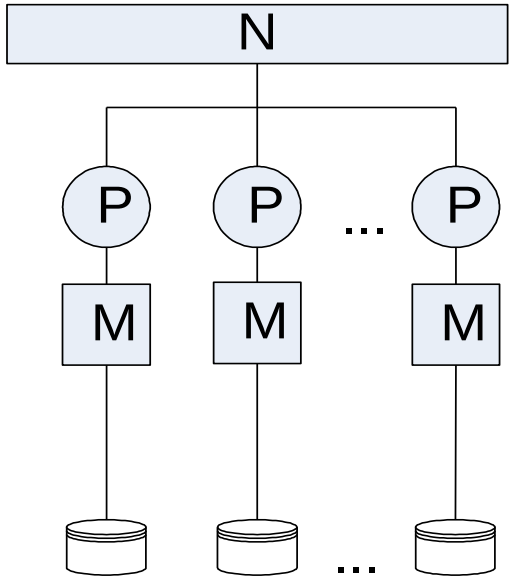
single multiprocessing computer



(b) SD



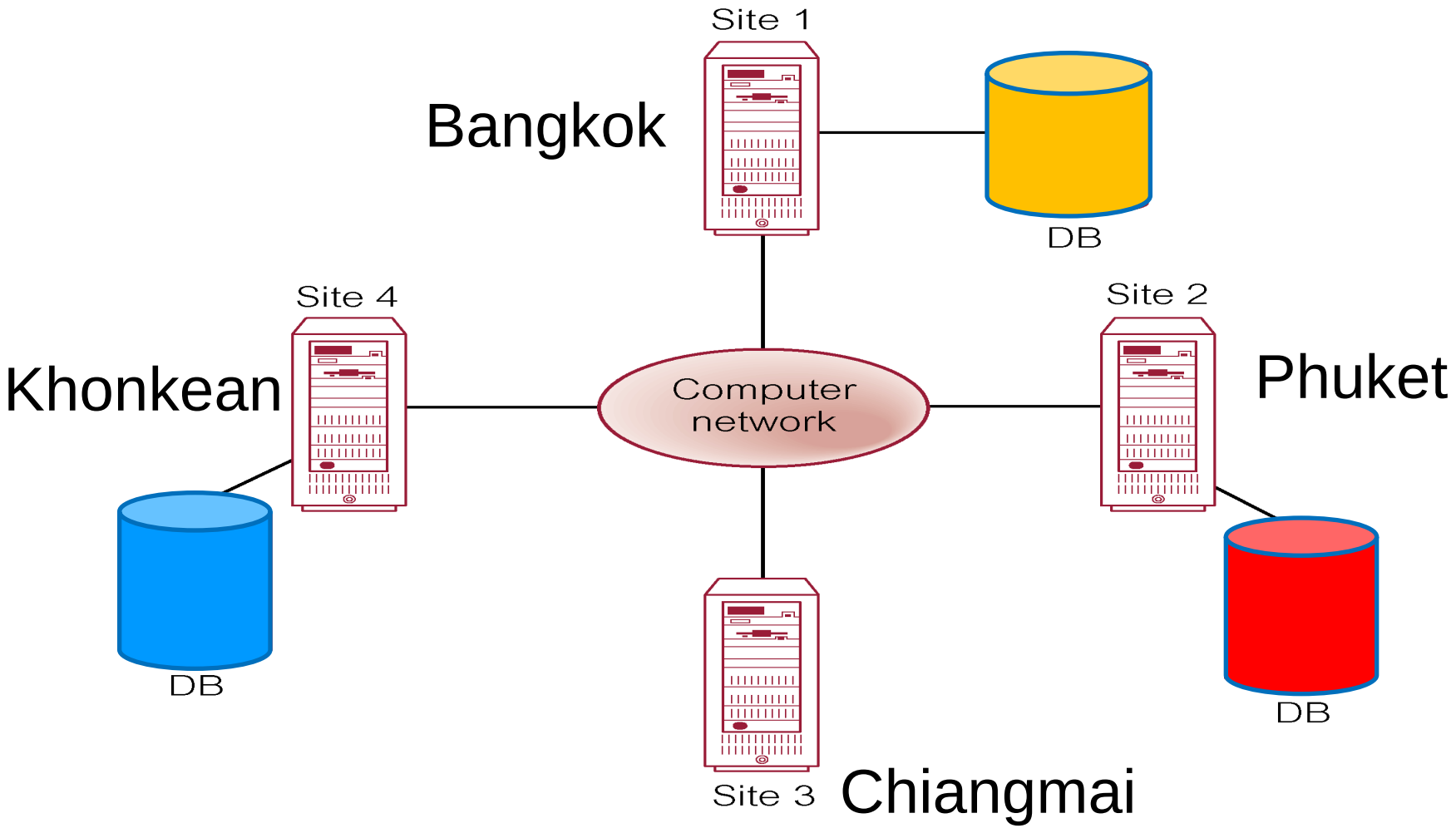
(c) SN



Legend

P: processor
M: memory
N: high-speed network
SE: shared everything
SD: shared disk
SN: shared nothing

Distributed Database

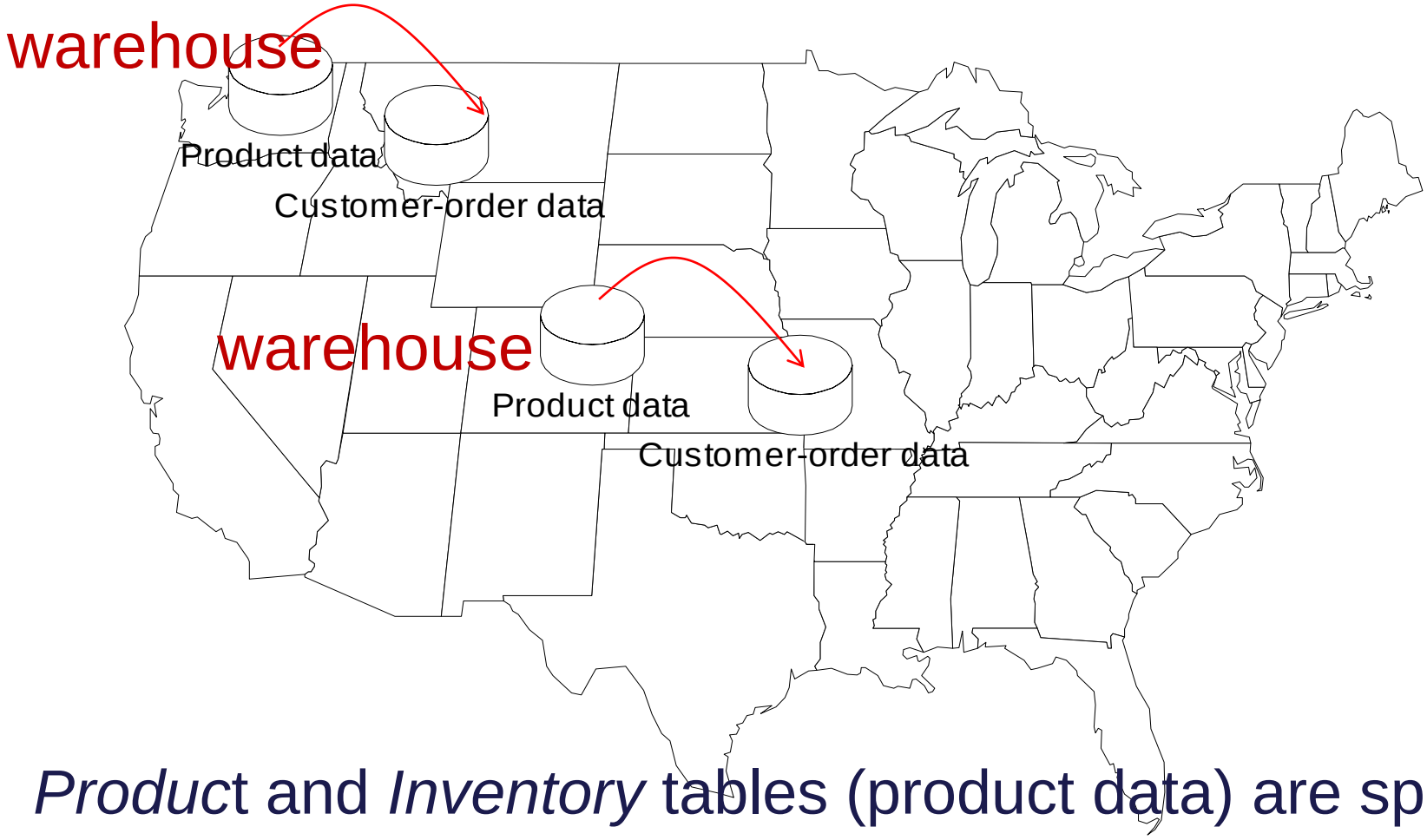


Distributed Database Architectures

- DBMSs need fundamental extensions.
- Underlying the extensions are a different component architecture and a different schema architecture.
- Component Architecture manages distributed database requests.
- Schema Architecture provides additional layers of data description (global, local).

Global Requests:

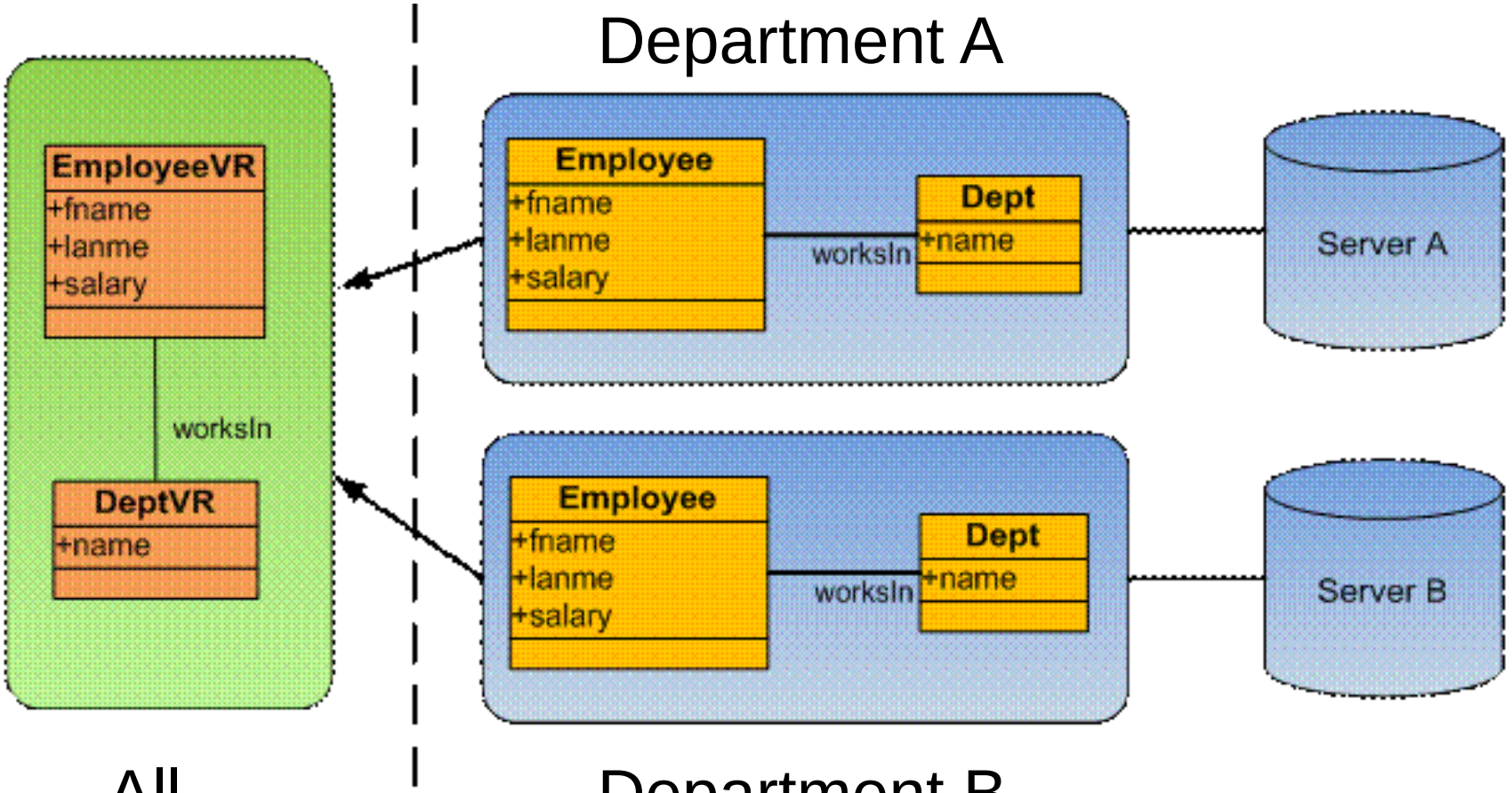
use data stored at more than one site



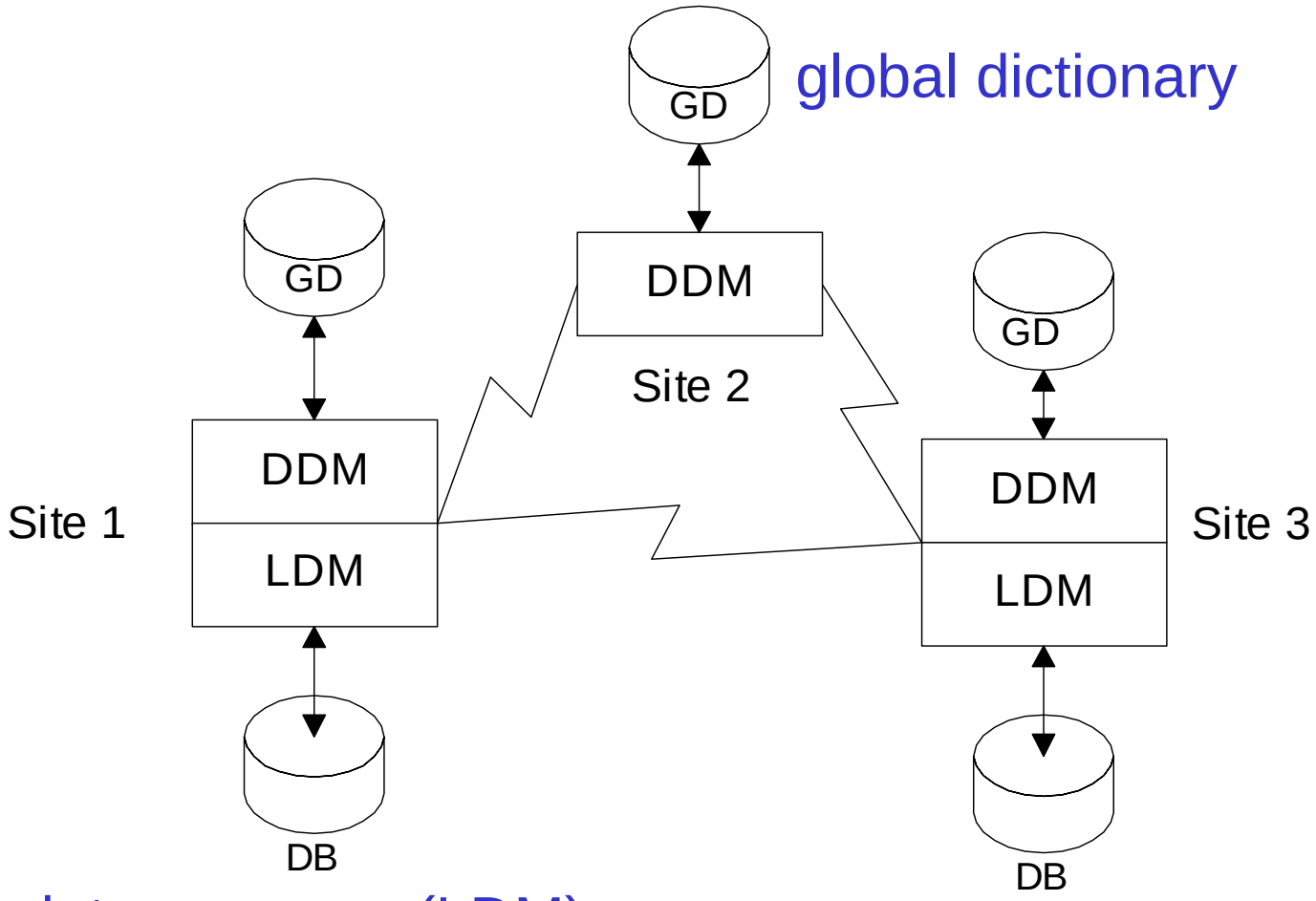
Product and *Inventory* tables (product data) are split between Seattle and Denver.

Virtual Schema

Fragmentation Schema



Components of a DDBMS



a local data manager (LDM)

distributed data manager (DDM)

Issues with DDBMS

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.

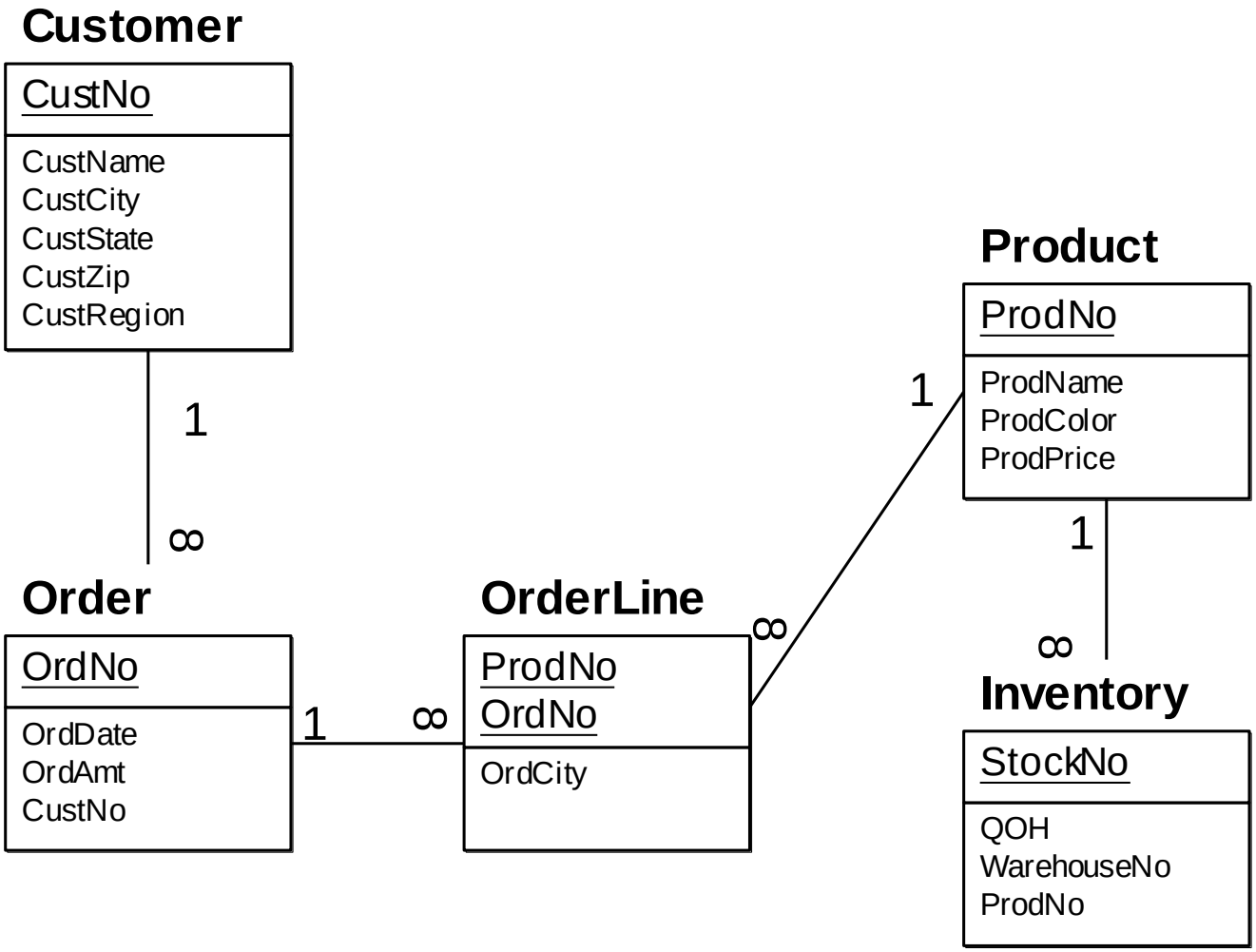
Replication

Copy of fragment may be maintained at several sites.

Distributed Database Transparency

- **Transparency** is related to data independence.
- With transparency, users can write queries with no knowledge of the distribution, and distribution changes will not cause changes to existing queries and transactions.
- Without transparency, users must reference some distribution details in queries and distribution changes can lead to changes in existing queries.

Motivating Example



Create Fragment

```
CREATE FRAGMENT Western_Customer AS
SELECT * FROM Customers
WHERE CustRegion = ‘West’
```

Vertical fragments can be defined by using project operator

Fragments Based on the *CustRegion* Column

```
CREATE FRAGMENT Western-Customers AS
  SELECT * FROM Customer WHERE CustRegion = 'West'
```

```
CREATE FRAGMENT Western-Orders AS
  SELECT Order.* FROM Order, Customer
    WHERE Order.CustNo = Customer.CustNo AND CustRegion = 'West'
```

Western Region

```
CREATE FRAGMENT Western-OrderLines AS
  SELECT OrderLine.* FROM Customer, OrderLine, Order
    WHERE OrderLine.OrdNo = Order.OrdNo
      AND Order.CustNo = Customer.CustNo AND CustRegion = 'West'
```

```
CREATE FRAGMENT Eastern-Customers AS
  SELECT * FROM Customer WHERE CustRegion = 'East'
```

```
CREATE FRAGMENT Eastern-Orders AS
  SELECT Order.* FROM Order, Customer
    WHERE Order.CustNo = Customer.CustNo AND CustRegion = 'East'
```

Eastern Region

```
CREATE FRAGMENT Eastern-OrderLines AS
 SELECT OrderLine.* FROM Customer, OrderLine, Order
    WHERE OrderLine.OrdNo = Order.OrdNo
      AND Order.CustNo = Customer.CustNo AND CustRegion = 'East'
```

Fragments Based on the *WareHouseNo* Column

```
CREATE FRAGMENT Denver-Inventory AS
  SELECT * FROM Inventory WHERE WareHouseNo = 1
```

```
CREATE FRAGMENT Seattle-Inventory AS
  SELECT * FROM Inventory WHERE WareHouseNo = 2
```

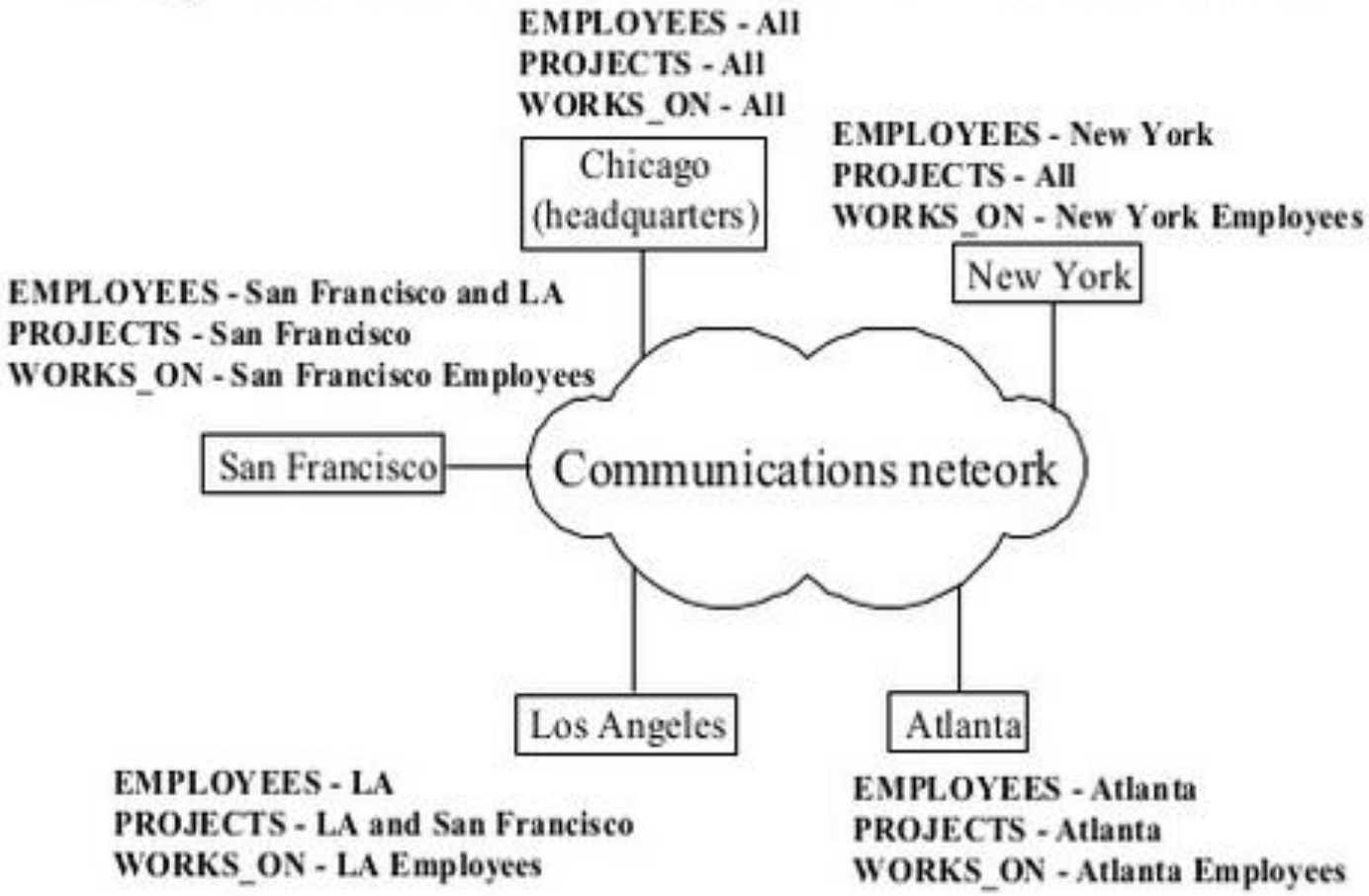
The *Product* table is not fragmented because the entire table is replicated at multiple sites: Denver & Seattle.

Distributed Database Processing

- Distributed data adds considerable complexity to
  - *query processing* and *transaction processing*.
- Distributed database processing involves
  - movement of data,
  - remote processing,
  - *site coordination*.
- Complexities can affect performance.

More example

The EMPLOYEE, PROJECT, and WORKS_ON tables may be fragmented horizontally and stored with possible replication as shown below.

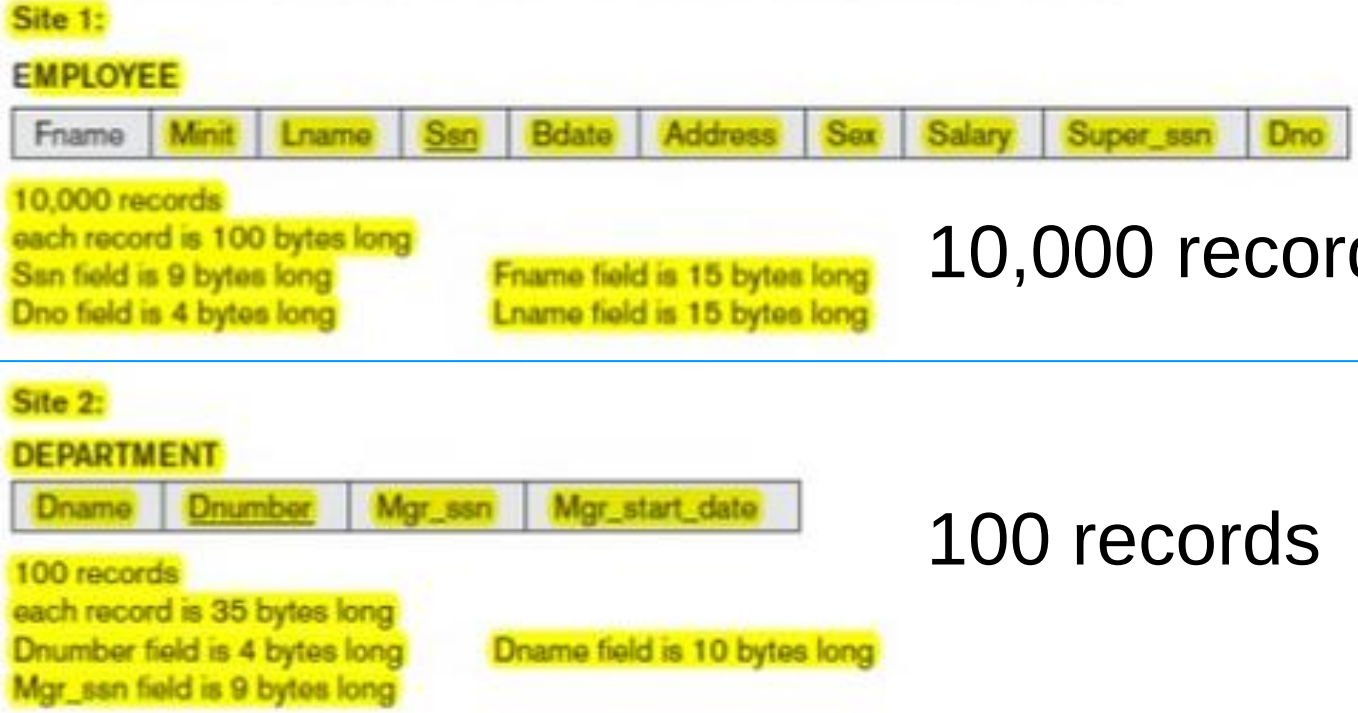


Distributed Query Processing

- Involves both local and global (inter site) optimization.
- Multiple optimization objectives
  - minimizing resources may conflict with minimizing response time
- The weighting of communication costs versus local processing costs depends on network characteristics.
- There are many more possible access plans for a distributed query.

Query Processing in Distributed Databases

Figure 25.10
Example to illustrate volume of data transferred.



10,000 records

100 records

Distributed Query Processing

Stretagies:

1. Transfer Employee and Department to site 3. Total transfer bytes = 1,000,000 + 3500 = 1,003,500 bytes.
2. Transfer Employee to site 2, execute join at site 2 and send the result to site 3. Query result size = 40 * 10,000 = 400,000 bytes. Total transfer size = 400,000 + 1,000,000 = 1,400,000 bytes.
3. Transfer Department relation to site 1, execute the join at site 1, and send the result to site 3. Total bytes transferred = 400,000 + 3500 = 403,500 bytes.

Optimization criteria: minimizing data transfer.

Preferred approach: strategy 3.

Distributed Transaction Processing

- Distributed DBMS provides concurrency and recovery transparency.
- Independently operating sites must be coordinated.
- ***New kinds of failures exist*** because of the communication network.
- New protocols are necessary.

Distributed Concurrency Control

- The simplest scheme involves ***centralized coordination.***
- Centralized coordination involves the fewest messages and the *simplest deadlock detection*.
- The number of messages can be twice as much in distributed coordination.
- ***Primary Copy Protocol*** is used to reduce overhead with locking multiple copies.

Write locks are necessary only for the primary copy.

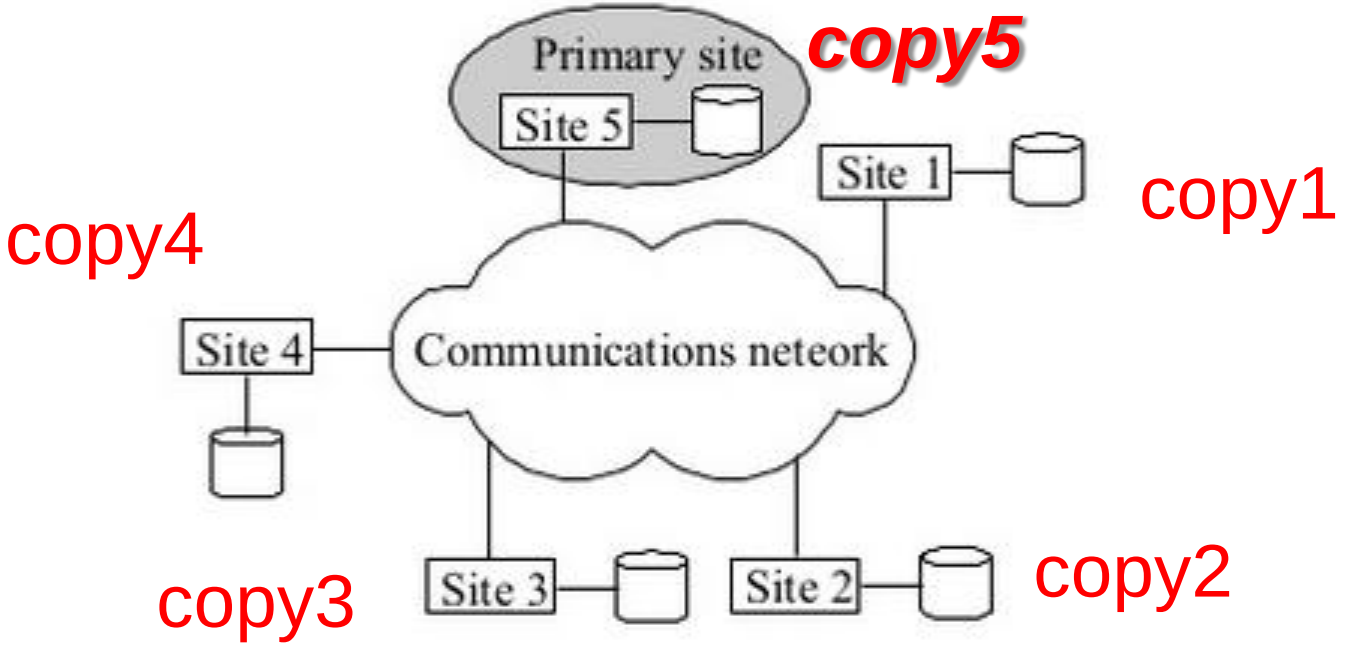
Primary Copy

- Lock managers distributed to a number of sites.
- For replicated data item, one copy is chosen as *primary copy*, others are *slave copies*
- Only need to *write-lock primary copy* of data item that is to be updated.
- Once primary copy has been updated, change can be propagated to slaves.
- Disadvantages - deadlock handling is more complex
- Advantages - lower communication costs and better performance than centralized 2PL.

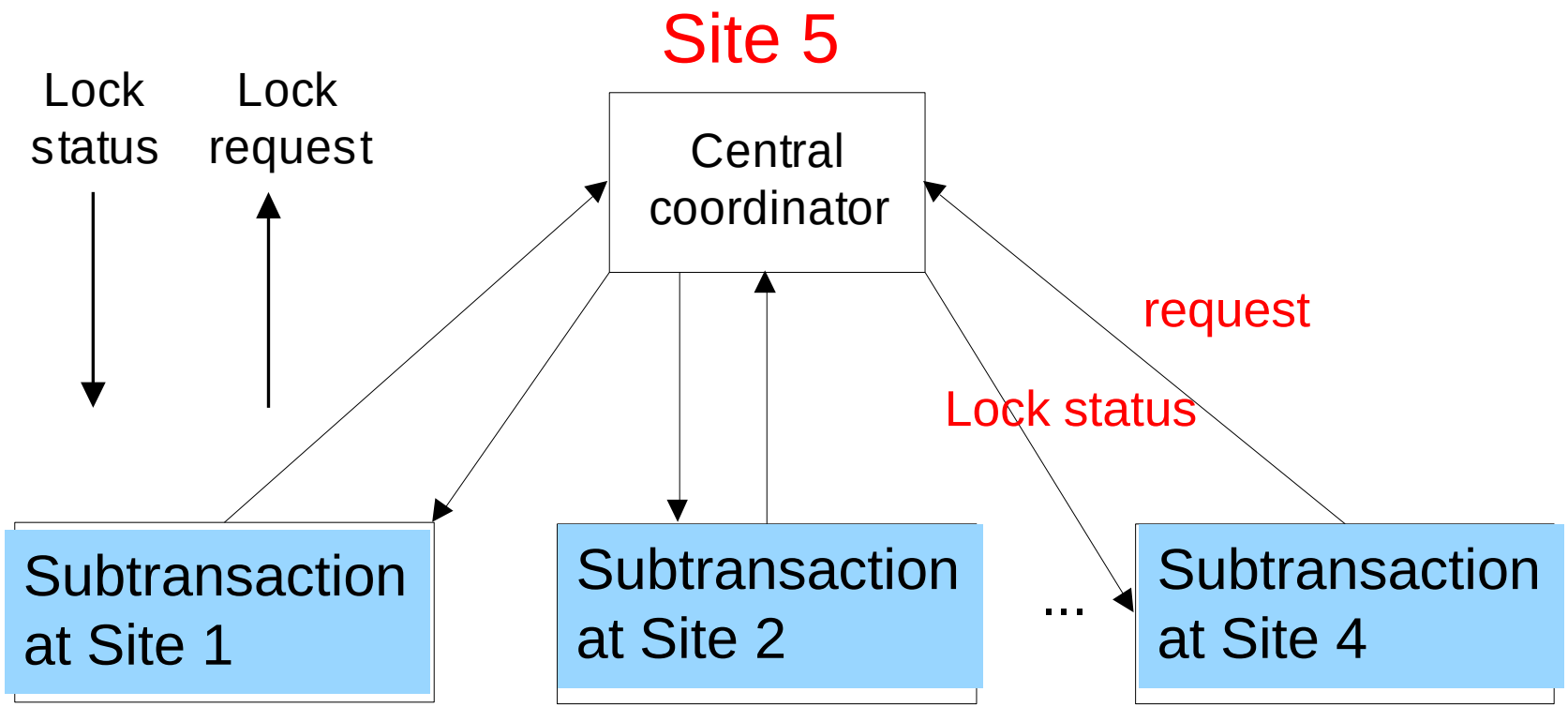
Distributed copy of data item

Primary site technique:

A single site is designated as a primary site which serves as a coordinator for transaction management.



Centralized Coordination



using the normal two phase locking rules for all sites.

Centralized Coordination

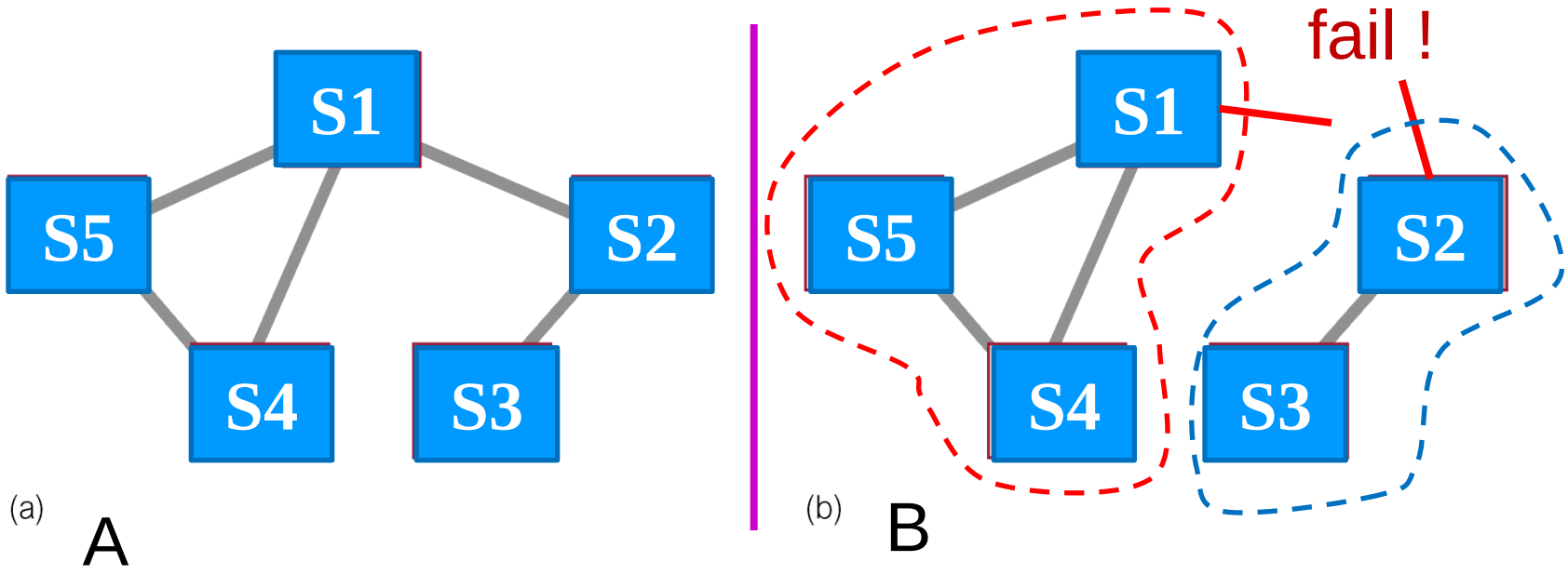
- At the beginning of a transaction, the **coordinating** site is chosen
- The transaction is divided into **sub transactions** performed at other sites.
- Each site hosting a sub transaction submits lock and release
- requests to the coordinating site using the normal **Two Phase Locking** rules.
  - guarantee serializable

Distributed Recovery Management

- Distributed DBMSs must contend with failures of communication links and sites.
- Detecting failures involves coordination among sites.
- The recovery manager must ensure that different parts of a partitioned network act in unison.

Distributed Recovery Control

- Communication failures can result in network becoming split into two or more partitions.
- May be difficult to distinguish whether communication link or site has failed.



Distributed Transaction

Transaction A

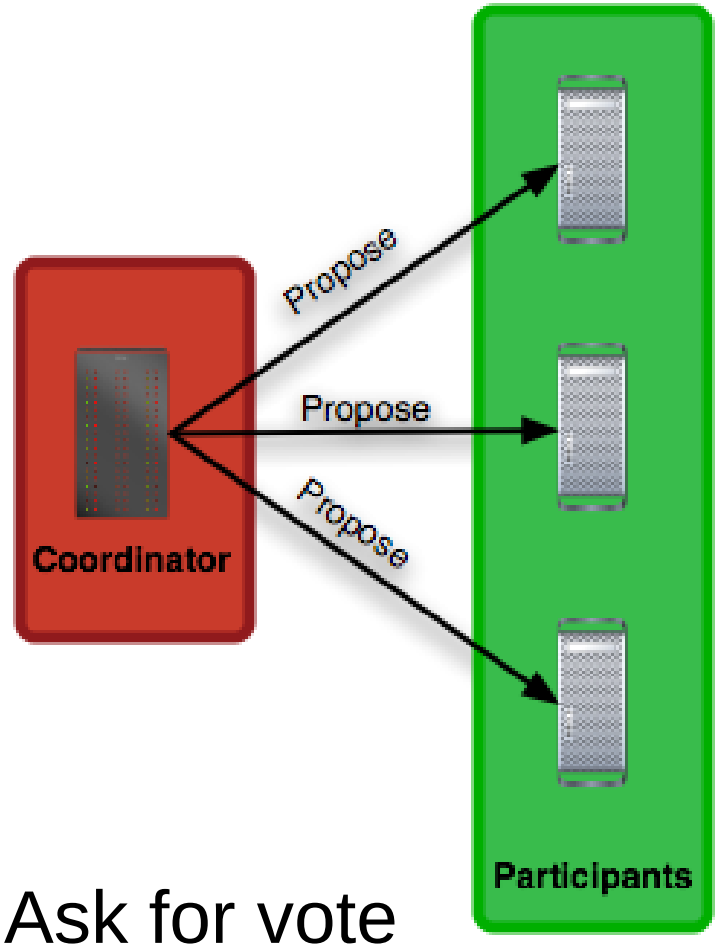
- insert customer at **site 1**
- update inventory at **site 2**
- update order at **site 3**

- To ensure atomicity, all sites need to agree on the final outcome of executing transaction A.

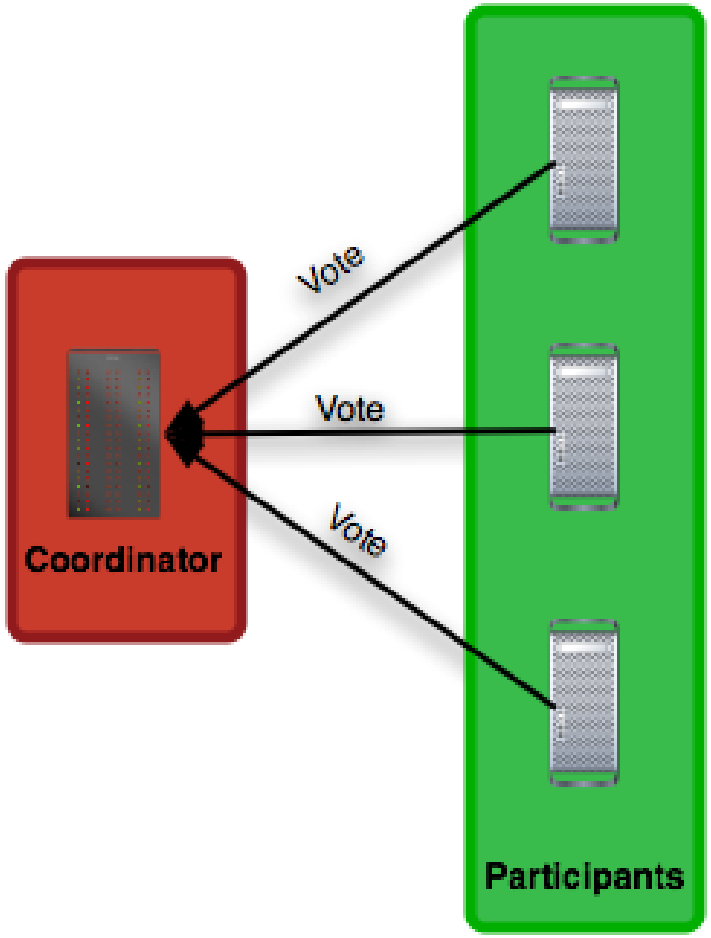
Two-Phase Commit (2PC)

- **Two phases: a *voting phase* and a *decision phase*.**
- **Coordinator asks all participants whether they are prepared to commit transaction.**
  - **If one participant votes abort, or fails to respond within a timeout period, coordinator instructs all participants to abort transaction.**
  - **If all vote commit, coordinator instructs all participants to commit.**
- **All participants must adopt global decision.**

1. Voting phase

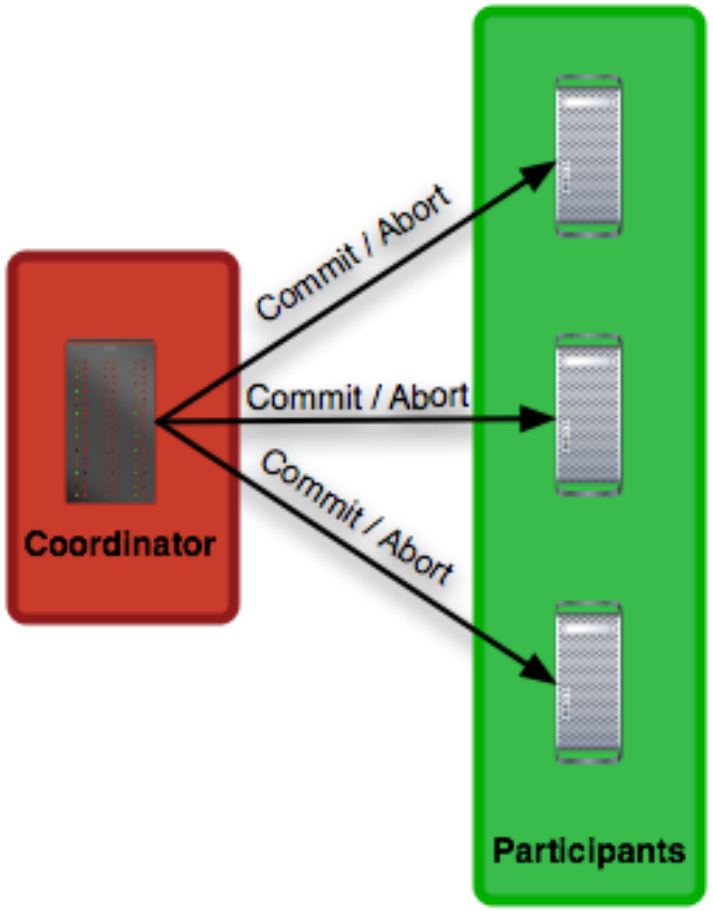


Ask for vote



If participant fails to vote, abort is assumed.

2. Decision phase



If one aborts, abort all

If all vote commit, commit all

If participant gets no vote instruction from coordinator, it can abort T.

Two-Phase Commit (2PC)

- **If participant votes abort, free to abort transaction immediately**
- **If participant votes commit, must wait for coordinator to broadcast global-commit or global-abort message.**

- **Protocol assumes each site has its own local log and can rollback or commit transaction reliably.**

Summary

- Utilizing distributed processing and data can significantly improve DBMS services but at the cost of new design challenges.
- Client-server architectures provide alternatives among cost, complexity, and benefit levels.
- Parallel database processing provides improved performance (speedup and scaleup) and availability.
- Architectures for distributed DBMSs differ in the integration of the local databases and level of data independence.