Chapter 13: Distributed Databases

Modern Database Management

6th Edition

Jeffrey A. Hoffer, Mary B. Prescott, Fred R. McFadden

Definitions

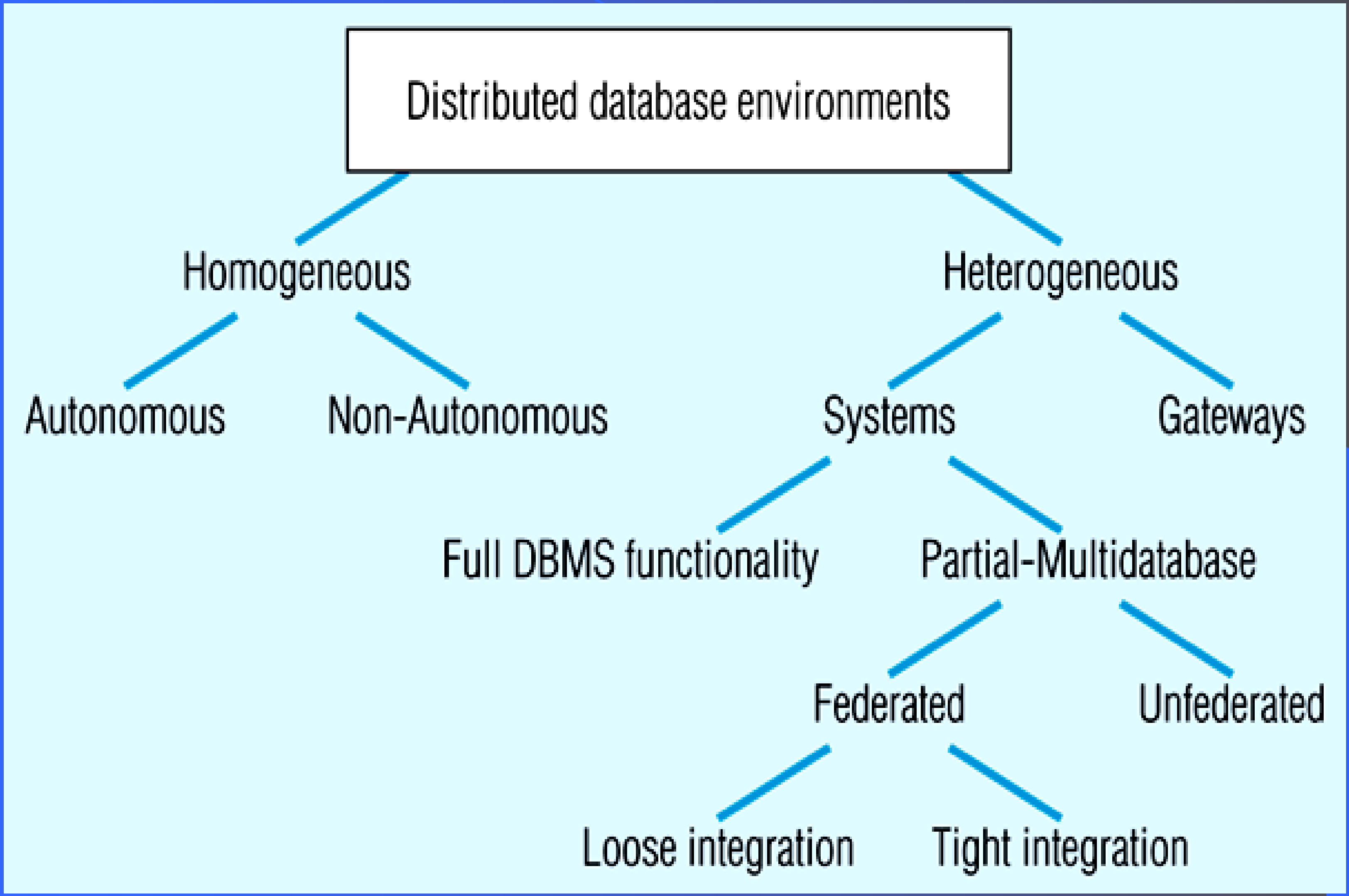
- **Distributed Database:** A *single logical database* that is spread physically across computers in multiple locations that are connected by a data communications link
- **Decentralized Database:** A collection of *independent* databases on non-networked computers

They are NOT the same thing!

Reasons for Distributed Database

- Business unit autonomy and distribution
- Data sharing
- Data communication costs
- Data communication reliability and costs
- Multiple application vendors
- Database recovery
- Transaction and analytic processing

Figure 13-1 -- Distributed database environments (adapted from Bell and Grimson, 1992)



Distributed Database Options

- Homogeneous - Same DBMS at each node
  - Autonomous - Independent DBMSs
  - Non-autonomous - Central , coordinating DBMS
  - Easy to manage, difficult to enforce
- Heterogeneous - Different DBMSs at different nodes
  - Systems – with full or partial DBMS functionality
  - Gateways - Simple paths are created to other databases without the benefits of one logical database
  - Difficult to manage, preferred by independent organizations

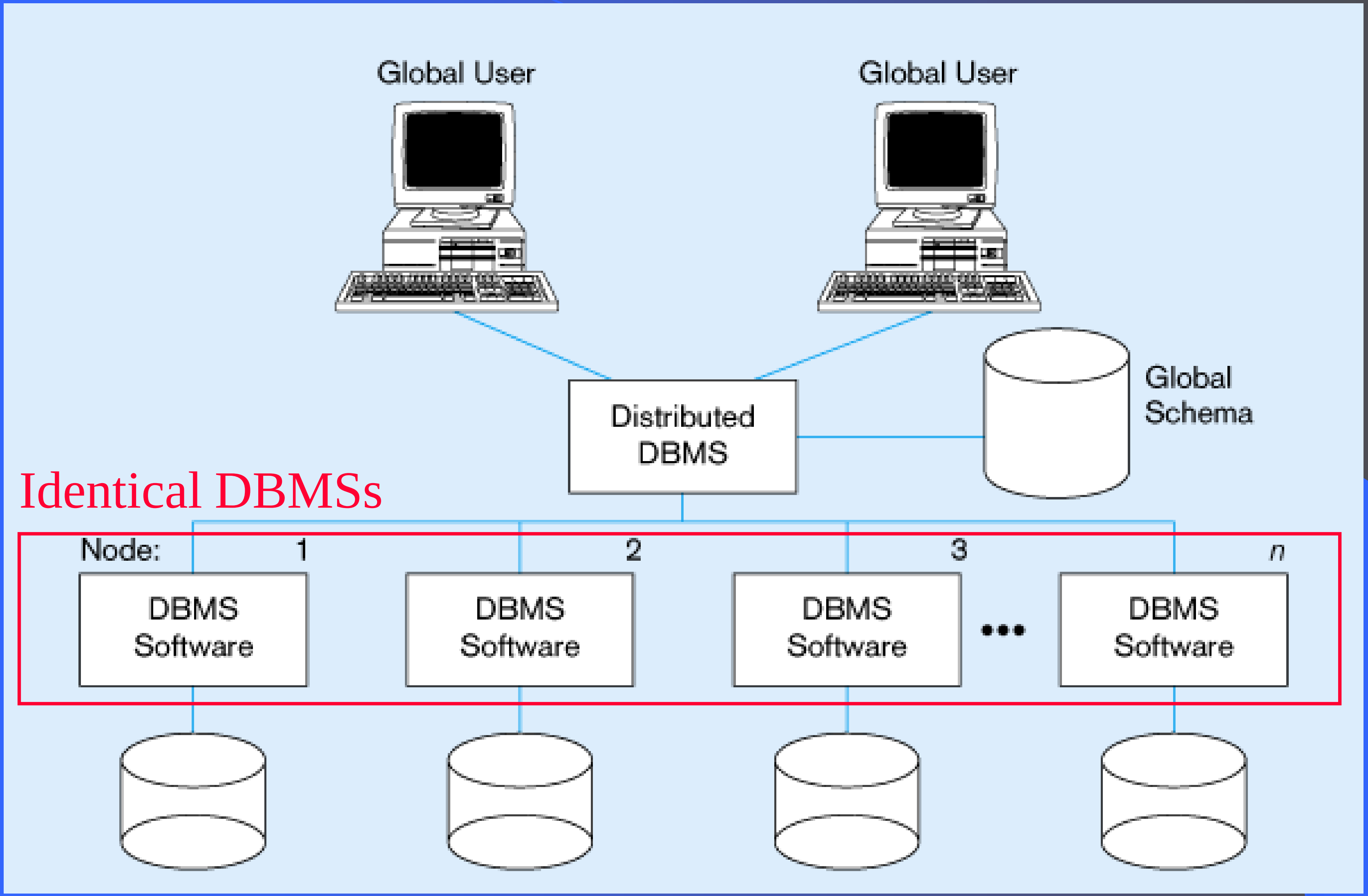
Distributed Database Options

- **Systems** - Supports some or all functionality of one logical database
  - Full DBMS Functionality - All dist. DB functions
  - Partial-Multi-database - Some dist. DB functions
    - Federated - Supports local databases for unique data requests
      - Loose Integration - Local dbs have their own schemas
      - Tight Integration - Local dbs use common schema
    - Unfederated - Requires all access to go through a central, coordinating module

Homogeneous, Non-Autonomous Database

- Data is distributed across all the nodes
- Same DBMS at each node
- All data is managed by the distributed DBMS (no exclusively local data)
- All access is through one, global schema
- The global schema is the *union* of all the local schema

Figure 13-2 – Homogeneous Database

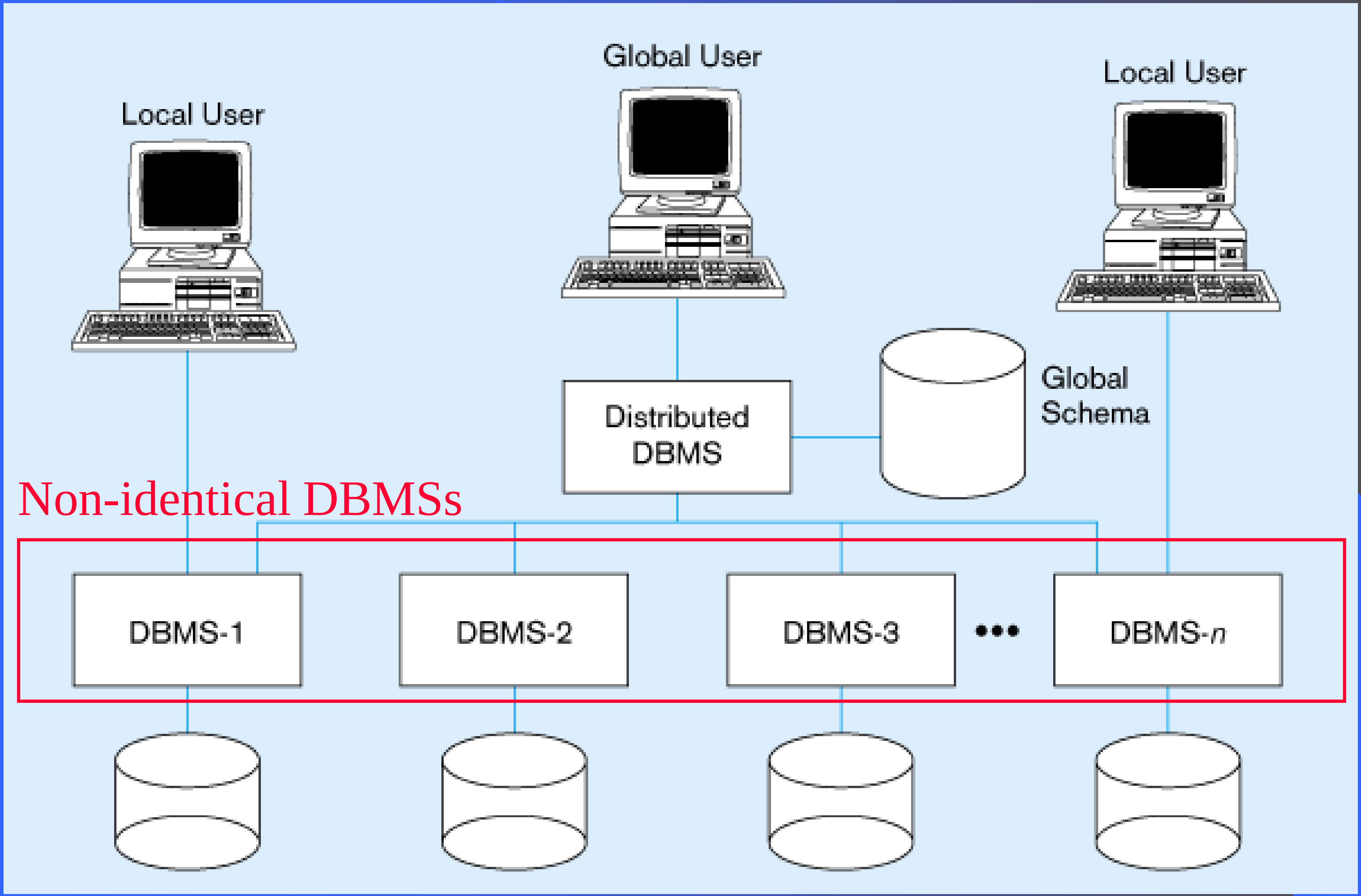


Source: adapted from Bell and Grimson, 1992.

Typical Heterogeneous Environment

- Data distributed across all the nodes
- Different DBMSs may be used at each node
- Local access is done using the local DBMS and schema
- Remote access is done using the global schema

Figure 13-3 –Typical Heterogeneous Environment



Source: adapted from Bell and Grimson, 1992.

Major Objectives

- Location Transparency
  – User does not have to know the location of the data.
  – Data requests automatically forwarded to appropriate sites
- Local Autonomy
  – Local site can operate with its database when network connections fail
  – Each site controls its own data, security, logging, recovery

Significant Trade-Offs

- Synchronous Distributed Database
  - All copies of the same data are always identical
  - Data updates are immediately applied to all copies throughout network
  - Good for data integrity
  - High overhead → slow response times
- Asynchronous Distributed Database
  - Some data inconsistency is tolerated
  - Data update propagation is delayed
  - Lower data integrity
  - Less overhead → faster response time

NOTE: all this assumes replicated data (to be discussed later)

Advantages of Distributed Database over Centralized Databases

- Increased reliability/availability
- Local control over data
- Modular growth
- Lower communication costs
- Faster response for certain queries

Disadvantages of Distributed Database compared to Centralized databases

- Software cost and complexity
- Processing overhead
- Data integrity exposure
- Slower response for certain queries

Options for Distributing a Database

- Data replication
  - Copies of data distributed to different sites
- Horizontal partitioning
  - Different rows of a table distributed to different sites
- Vertical partitioning
  - Different columns of a table distributed to different sites
- Combinations of the above

Data Replication

- Advantages -
  - Reliability
  - Fast response
  - May avoid complicated distributed transaction integrity routines (if replicated data is refreshed at scheduled intervals)
  - De-couples nodes (transactions proceed even if some nodes are down)
  - Reduced network traffic at prime time (if updates can be delayed)

Data Replication

Disadvantages -

- Additional requirements for storage space
- Additional time for update operations
- Complexity and cost of updating
- Integrity exposure of getting incorrect data if replicated data is not updated simultaneously

Therefore, better when used for non-volatile (read-only) data

Types of Data Replication

- Push Replication –
  - updating site sends changes to other sites
- Pull Replication –
  - receiving sites control when update messages will be processed

Types of Push Replication

Snapshot Replication -
- Changes periodically sent to master site
- Master collects updates in log
- Full or differential (incremental) snapshots
- Dynamic vs. shared update ownership

Near Real-Time Replication -
- Broadcast update orders without requiring confirmation
- Done through use of triggers
- Update messages stored in message queue until processed by receiving site

Issues for Data Replication

- Data timeliness – high tolerance for out-of-date data may be required
- DBMS capabilities – if DBMS cannot support multi-node queries, replication may be necessary
- Performance implications – refreshing may cause performance problems for busy nodes
- Network heterogeneity – complicates replication
- Network communication capabilities – complete refreshes place heavy demand on telecommunications

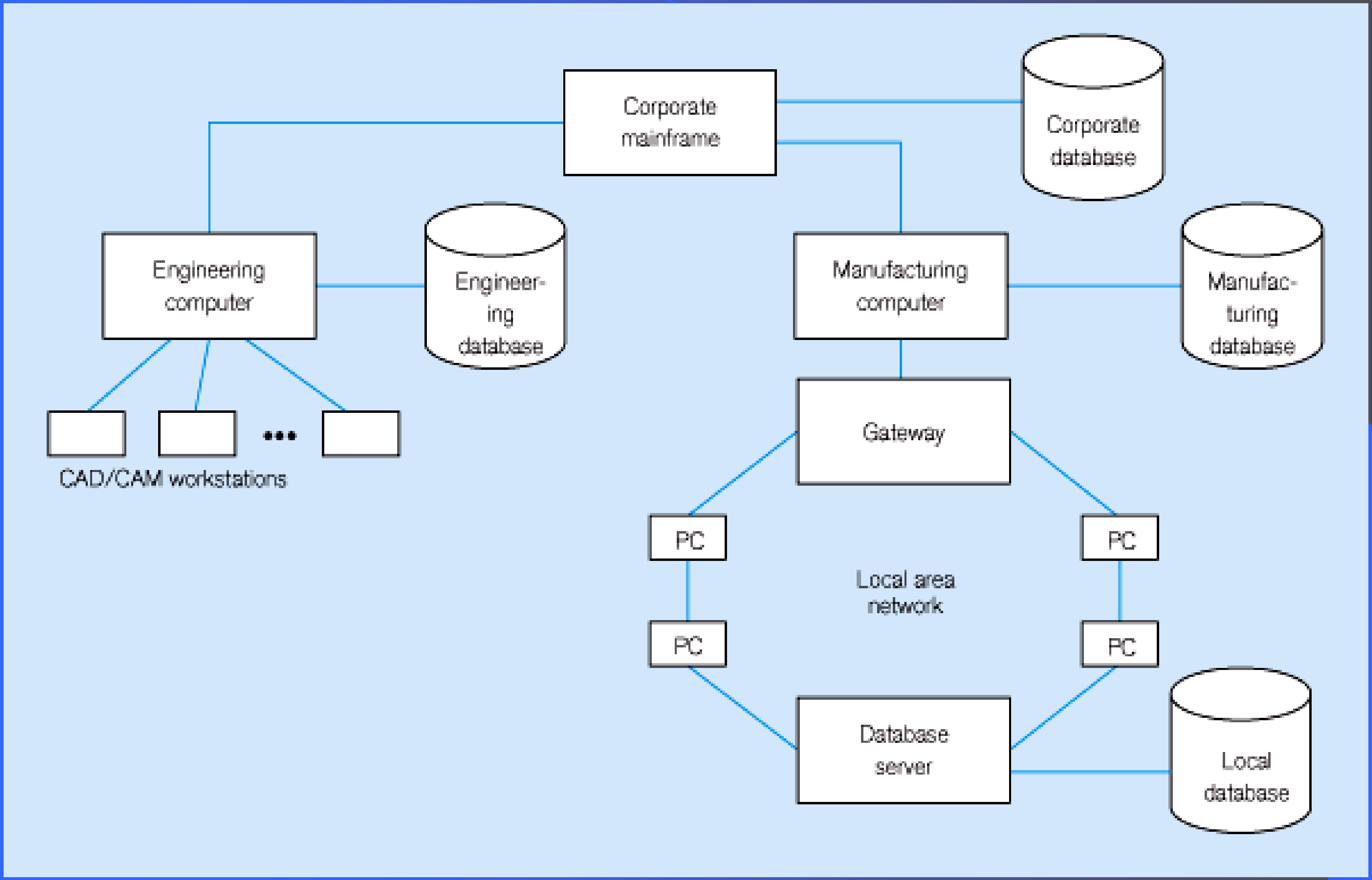
Horizontal Partitioning

- Different rows of a table at different sites
- Advantages -
  - Data stored close to where it is used → efficiency
  - Local access optimization → better performance
  - Only relevant data is available → security
  - Unions across partitions → ease of query
- Disadvantages
  - Accessing data across partitions → inconsistent access speed
  - No data replication → backup vulnerability

Vertical Partitioning

- Different columns of a table at different sites
- Advantages and disadvantages are the same as for horizontal partitioning except that combining data across partitions is more difficult because it requires joins (instead of unions)

Figure 13-6
Distributed processing system for a manufacturing company



Five Distributed Database Organizations

- Centralized database, distributed access
- Replication with periodic snapshot update
- Replication with near real-time synchronization of updates
- Partitioned, one logical database
- Partitioned, independent, non-integrated segments

Factors in Choice of Distributed Strategy

- Funding, autonomy, security
- Site data referencing patterns
- Growth and expansion needs
- Technological capabilities
- Costs of managing complex technologies
- Need for reliable service

See table 13-1

Table 13-1: Distributed Design Strategies

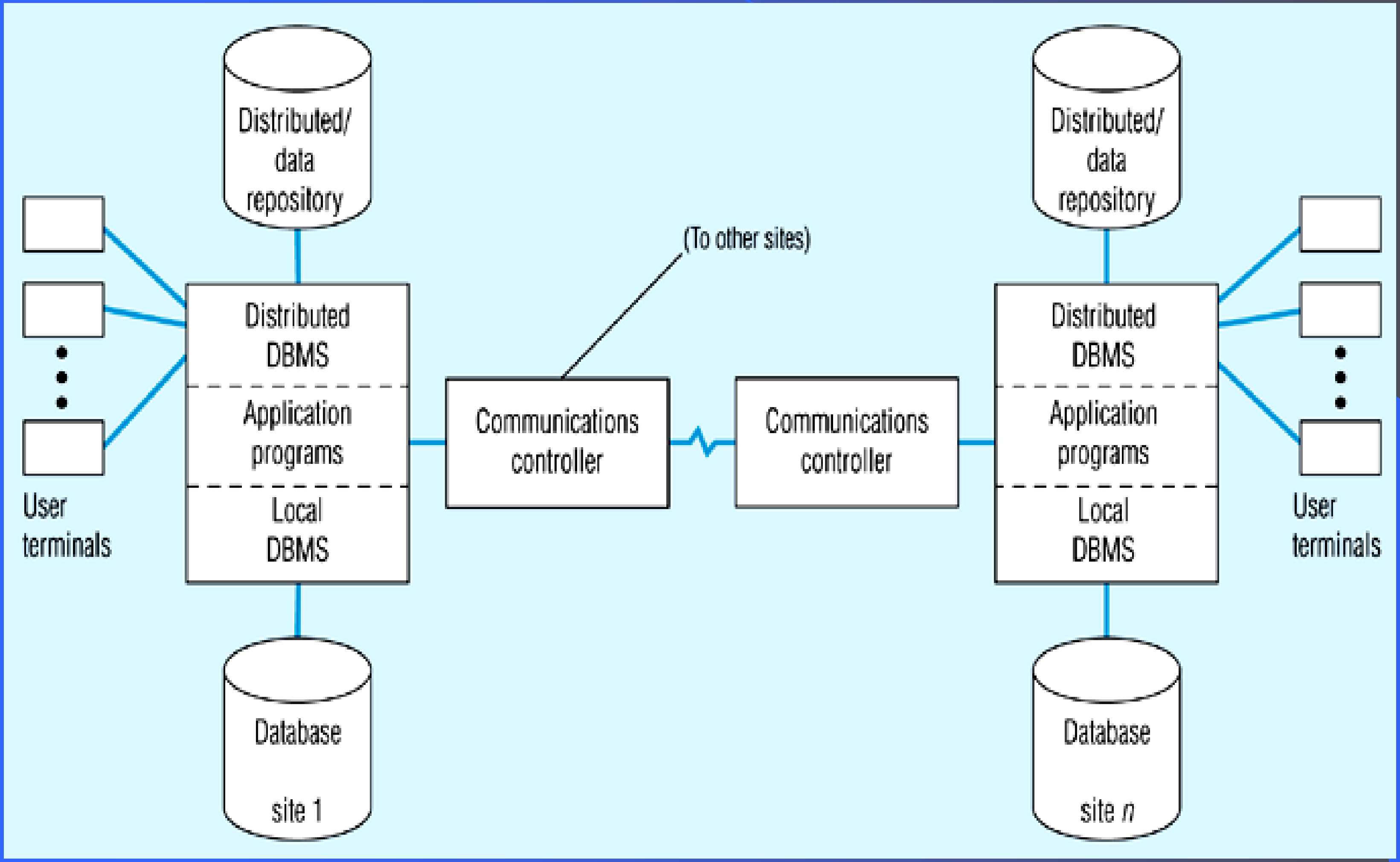
Table 13-1 Comparison of Distributed Database Design Strategies

Strategy	*Reliability*	*Expandability*	*Communications Overhead*	*Manageability*	*Data Consistency*
Centralized	**POOR:** Highly dependent on central server	**POOR:** Limitations are barriers to performance	**VERY HIGH:** High traffic to one site	**VERY GOOD:** One, monolithic site requires little coordination	**EXCELLENT:** All users always have same data
Replicated with snapshots	**GOOD:** Redundancy and tolerated delays	**VERY GOOD:** Cost of additional copies may be less than linear	**LOW to MEDIUM:** Not constant, but periodic snapshots can cause bursts of network traffic	**VERY GOOD:** Each copy is like every other one	**MEDIUM:** Fine as long as delays are tolerated by business needs
Synchronized replication	**EXCELLENT:** Redundancy and minimal delays	**VERY GOOD:** Cost of additional copies may be low and synchronization work only linear	**MEDIUM:** Messages are constant, but some delays are tolerated	**MEDIUM:** Collisions add some complexity to manageability	**MEDIUM to VERY GOOD:** Close to precise consistency
Integrated partitions	**VERY GOOD:** Effective use of partitioning and redundancy	**VERY GOOD:** New nodes get only data they need without changes in overall database design	**LOW to MEDIUM:** Most queries are local but queries which require data from multiple sites can cause a temporary load	**DIFFICULT:** Especially difficult for queries that need data from distributed tables, and updates must be tightly coordinated	**VERY POOR:** Considerable effort, and inconsistencies not tolerated
Decentralized with independent partitions	**GOOD:** Depends on only local database availability	**GOOD:** New sites independent of existing ones	**LOW:** Little if any need to pass data or queries across the network (if one exists)	**VERY GOOD:** Easy for each site, until there is a need to share data across sites	**LOW:** No guarantees of consistency, in fact pretty sure of inconsistency

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

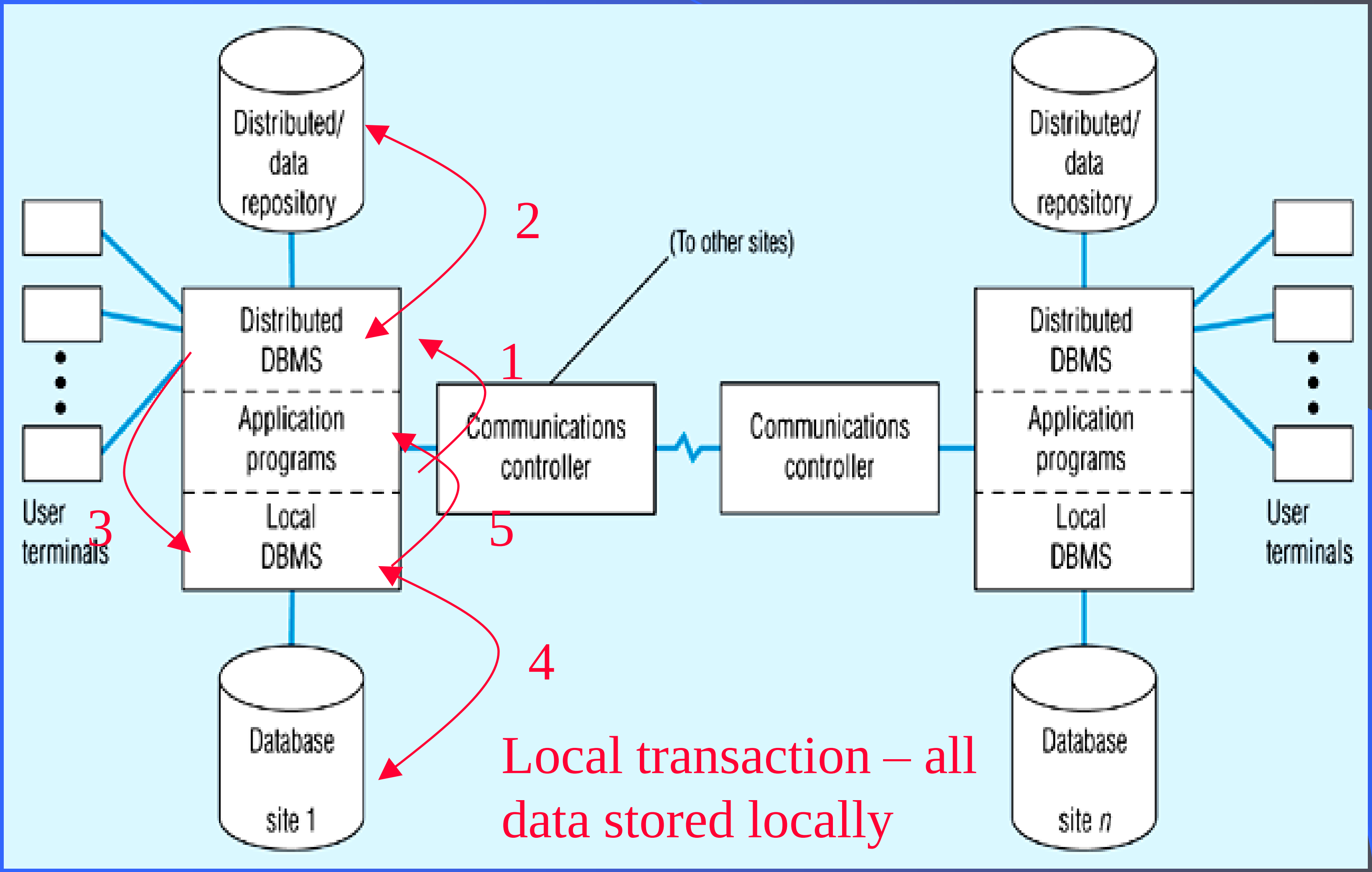
Figure 13-10 – Distributed DBMS architecture



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

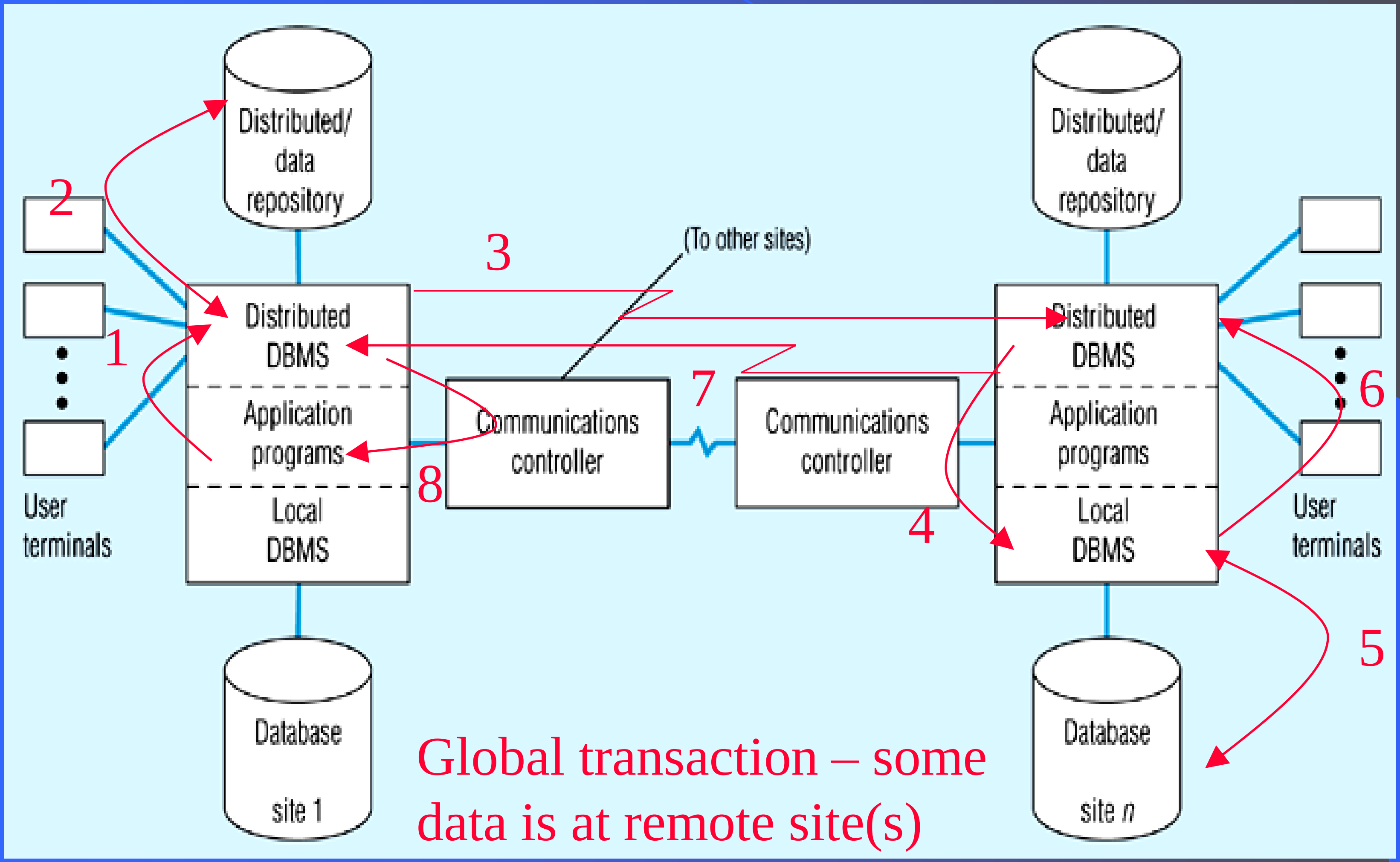
Figure 13-10 – Distributed DBMS Architecture showing Local Transaction Steps



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

Figure 13-10 – Distributed DBMS architecture showing global transaction steps



Distributed DBMS Transparency Objectives

- Location Transparency
  - User/application does not need to know where data resides
- Replication Transparency
  - User/application does not need to know about duplication
- Failure Transparency
  - Either all or none of the actions of a transaction are committed
  - Each site has a transaction manager
    - Logs transactions and before and after images
    - Concurrency control scheme to ensure data integrity
  - Requires special *commit protocol*

Two-Phase Commit

Prepare Phase

– Coordinator receives a commit request
– Coordinator instructs all resource managers to get ready to "go either way" on the transaction. Each resource manager writes all updates from that transaction to its own physical log
– Coordinator receives replies from all resource managers. If all are ok, it writes commit to its own log; if not then it writes rollback to its log

Two-Phase Commit

Commit Phase

- Coordinator then informs each resource manager of its decision and broadcasts a message to either commit or rollback (abort). If the message is commit, then each resource manager transfers the update from its log to its database
- A failure during the commit phase puts a transaction "in limbo." This has to be tested for and handled with timeouts or polling

Concurrency Control

- Concurrency Transparency
  - Design goal for distributed database
- Timestamping
  - Concurrency control mechanism
  - Alternative to locks in distributed databases

Query Optimization

In a query involving a multi-site join and, possibly, a distributed database with replicated files, the distributed DBMS must decide where to access the data and how to proceed with the join. Three step process:

1. ***Query decomposition*** - rewritten and simplified
2. ***Data localization*** - query fragmented so that fragments reference data at only one site
3. ***Global optimization*** -
   - Order in which to execute query fragments
   - Data movement between sites
   - Where parts of the query will be executed

Evolution of Distributed DBMS

- "Unit of Work" - All of a transaction's steps.
- Remote Unit of Work
  – SQL statements originated at one location can be executed as a single unit of work on a single remote DBMS

Evolution of Distributed DBMS

- Distributed Unit of Work
  - Different statements in a unit of work may refer to different remote sites
  - All databases in a single SQL statement must be at a single site
- Distributed Request
  - A single SQL statement may refer to tables in more than one remote site
  - May not support replication transparency or failure transparency