COP 4710: Database Systems
Spring 2006

CHAPTER 22 – Parallel and Distributed Database Systems – Part 2

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CSB 242, 823-2790
http://www.cs.ucf.edu/courses/cop4710/spr2006

School of Electrical Engineering and Computer Science
University of Central Florida

Options for Distributing A Database

- There are four basic strategies that can be employed for distributing a database:

1. Data replication
   - Full
   - Partial

   Covered in previous section of notes

2. Horizontal fragmentation
3. Vertical fragmentation
4. Combinations of those above.
   - Replicated horizontal fragments
   - Replicated vertical fragments
   - Horizontal/vertical fragments

Horizontal Fragmentation

- With horizontal fragmentation, some of the rows of a relation (table) are put into a base relation at one site, and other rows of the relation are put into a base relation at another site.
  - Note: there is no overlapping of the rows across the various sites – this is pure fragmentation, if there were overlapping rows, we would also have replication, which falls into the last category of distributed options. This would be a more general approach, although it is also quite common.
- Horizontal fragmentation results from applying selection conditions (relational algebra selections) to relations.

Horizontal Fragmentation (cont.)

Customer Name	Branch
Kristi	Oviedo
Debbie	Maitland
Michael	Longwood
Didi	Oviedo
Tawni	Oviedo

Initial table R

Horizontal fragments based on:

$\delta_{(\text{Branch} = \text{'Oviedo'})}(R)$

Customer Name	Branch
Kristi	Oviedo
Didi	Oviedo
Tawni	Oviedo

Fragment #1

Customer Name	Branch
Debbie	Maitland
Michael	Longwood

Fragment #2

Horizontal Fragmentation (cont.)

- Horizontal fragmentation has four major advantages:

1. Efficiency – Data can be stored close to where they are used and separate from other data used by other users or applications.
2. Local optimization – Data can be stored to optimize performance for local access rather than global access.
3. Security – Data not relevant to usage at a particular site is not made available at that site.
4. Ease of querying – Combining data across horizontal fragments is easy because the rows are simply merged by unions across the fragments.

Horizontal Fragmentation (cont.)

- Horizontal fragmentation has two primary disadvantages:

  1. Inconsistent access speed – When data from several fragments are required, the access time can be significantly different from local-only data access.

  2. Backup vulnerability – Since the data is not replicated, when data at one site becomes inaccessible or damaged, usage cannot switch to another site where a copy exists; data may be lost if proper backup is not performed at each site.

- Horizontal fragmentation is typically used when an organizational function is distributed, but each site is concerned with only a subset of the entity instances (often geographically based).

Vertical Fragmentation

- With vertical fragmentation, some of the columns of a relation (table) are put into a base relation at one site, and other columns of the relation are put into a base relation at another site.
  - Note: there must be a common domain stored at each site so that the original table can be reconstructed.
- Vertical fragmentation results from applying projection operations (relational algebra projection) to relations.

Vertical Fragmentation (cont.)

Customer Name	Branch	Balance
Kristi	Oviedo	15,000
Debbie	Maitland	23,000
Michael	Longwood	4,000
Didi	Oviedo	50,000
Tawni	Oviedo	18,000

Initial table R

Vertical fragment based on: $\pi_{(name, branch)}(R)$

Customer Name	Branch
Kristi	Oviedo
Debbie	Maitland
Michael	Longwood
Didi	Oviedo
Tawni	Oviedo

Vertical fragment based on: $\pi_{(name, balance)}(R)$

Customer Name	Balance
Kristi	15,000
Debbie	23,000
Michael	4,000
Didi	50,000
Tawni	18,000

Combinations of Distribution Strategies

- To complicate matters even further, there are an almost unlimited number of combinations of distribution strategies based upon the previous cases.
- Some data may be stored centrally while others are replicated. Both horizontal and vertical fragments can be replicated.

Horizontal/Vertical Fragmentation

Customer Name	Branch	Balance
Kristi	Oviedo	15,000
Debbie	Maitland	23,000
Michael	Longwood	4,000
Didi	Oviedo	50,000
Tawni	Oviedo	18,000

Initial table R

Fragment based on: $\delta_{(branch = 'Oviedo')}(\pi_{(name, branch)}(R))$

Customer Name	Branch
Kristi	Oviedo
Didi	Oviedo
Tawni	Oviedo

Fragment based on: $\delta_{(branch <> 'Oviedo')}(\pi_{(name, branch)}(R))$

Customer Name	Branch
Debbie	Maitland
Michael	Longwood

Fragment based on: $\delta_{(balance >15000)}(R)$

Customer Name	Branch	Balance
Debbie	Maitland	23,000
Didi	Oviedo	50,000
Tawni	Oviedo	18,000

Fragment based on: $\delta_{(name = 'Krisit')}(\pi_{(name, balance)}(R))$

Customer Name	Balance
Kristi	15,000

Selecting a Distribution Strategy

- A distributed database can be organized in five unique ways:

1. Totally centralized – all data resides at one location accessed from many geographically distributed sites.
2. Partially or totally replicated (snapshot) – data is either partially or totally replicated across geographically distributed sites, with each replica periodically updated with snapshots.
3. Partially or totally replicated (real-time synchronization) – data is either partially or totally replicated across geographically distributed sites, with near real-time synchronization.
4. Fragmented (integrated) – data is into segments at different geographically distributed sites, but still within one logical database and one distributed DBMS.
5. Fragmented (nonintegrated) – data is fragmented into independent, non integrated segments spanning multiple computer systems and database software.

Selecting a Distribution Strategy

- A distributed database can be organized in five unique ways:

1. Totally centralized – all data resides at one location accessed from many geographically distributed sites.
2. Partially or totally replicated (snapshot) – data is either partially or totally replicated across geographically distributed sites, with each replica periodically updated with snapshots.
3. Partially or totally replicated (real-time synchronization) – data is either partially or totally replicated across geographically distributed sites, with near real-time synchronization.
4. Fragmented (integrated) – data is into segments at different geographically distributed sites, but still within one logical database and one distributed DBMS.
5. Fragmented (nonintegrated) – data is fragmented into independent, non integrated segments spanning multiple computer systems and database software.

Summary of Distributed 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

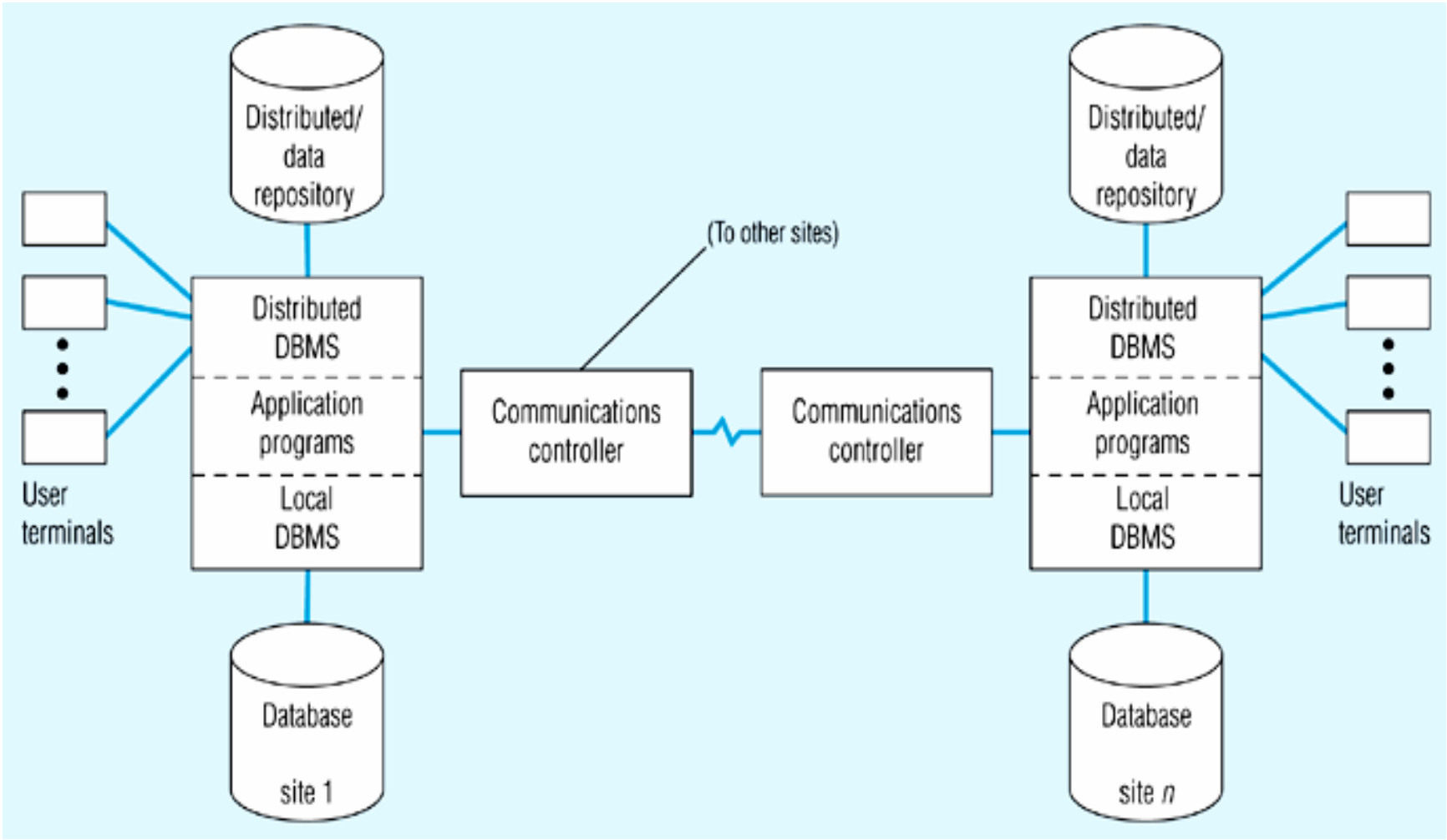
Distributed DBMS

- To have a distributed database, there must be a database management system that coordinates the access to the data at the various sites.
- Such a system is called a distributed DBMS.
- Although each site may have a DBMS managing the local database at that site, a distributed DBMS must perform the following functions for the distributed database.

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.
- Provide scalability.
- Security, concurrency, query optimization, failure recovery.

Distributed DBMS Architecture



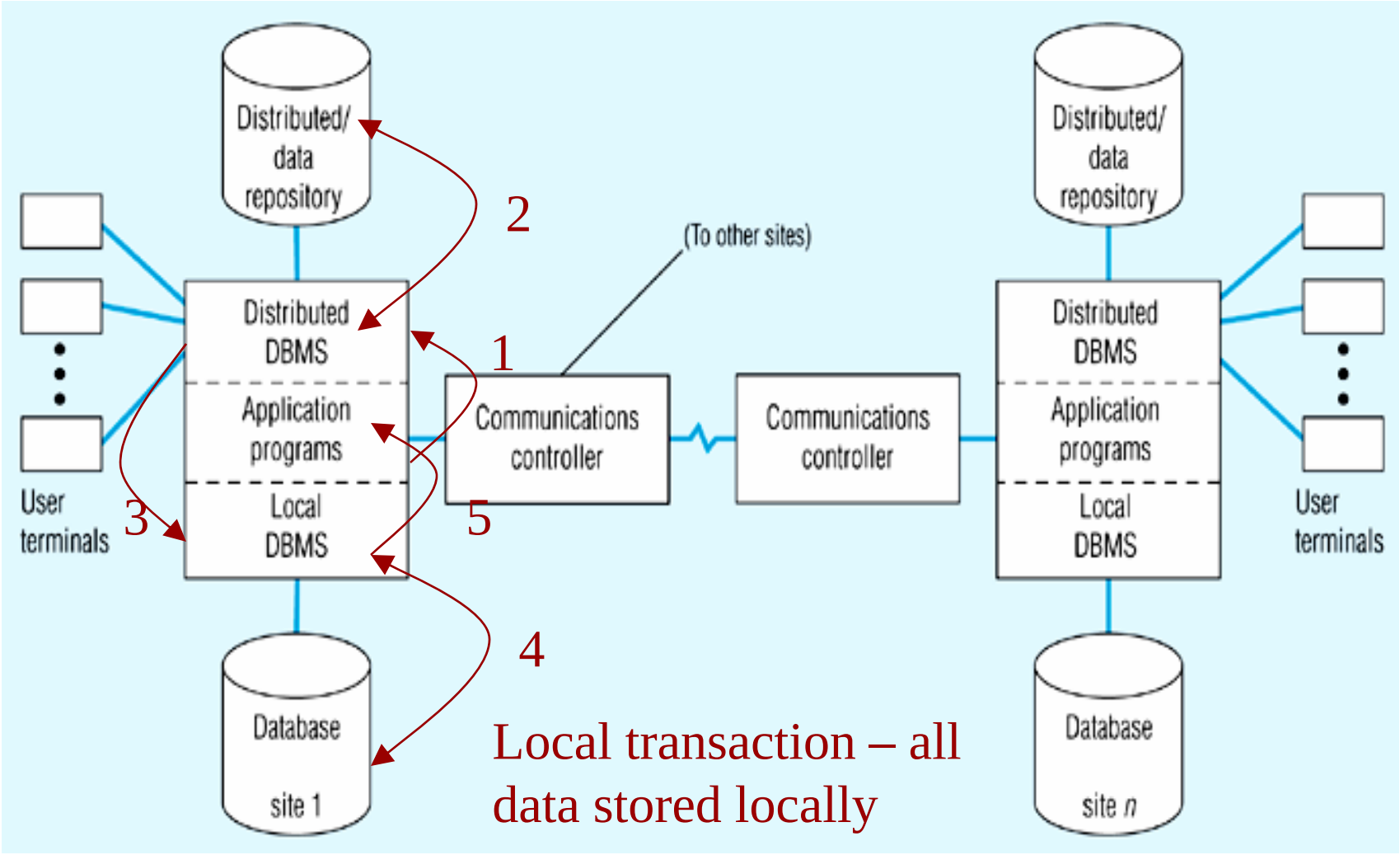
Local vs. Global Transactions

- A local transaction is one for which the required data are stored entirely at the local site.
  - The distributed DBMS passes the request to the local DBMS.
- A global transaction references data at one or more non-local sites.
  - The distributed DBMS routes the request to other sites as necessary. The distributed DBMSs at the participating sites exchange messages as needed to coordinate the processing of the transaction until it is completed (or aborted, if necessary).

Steps to Process a Local Transaction

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

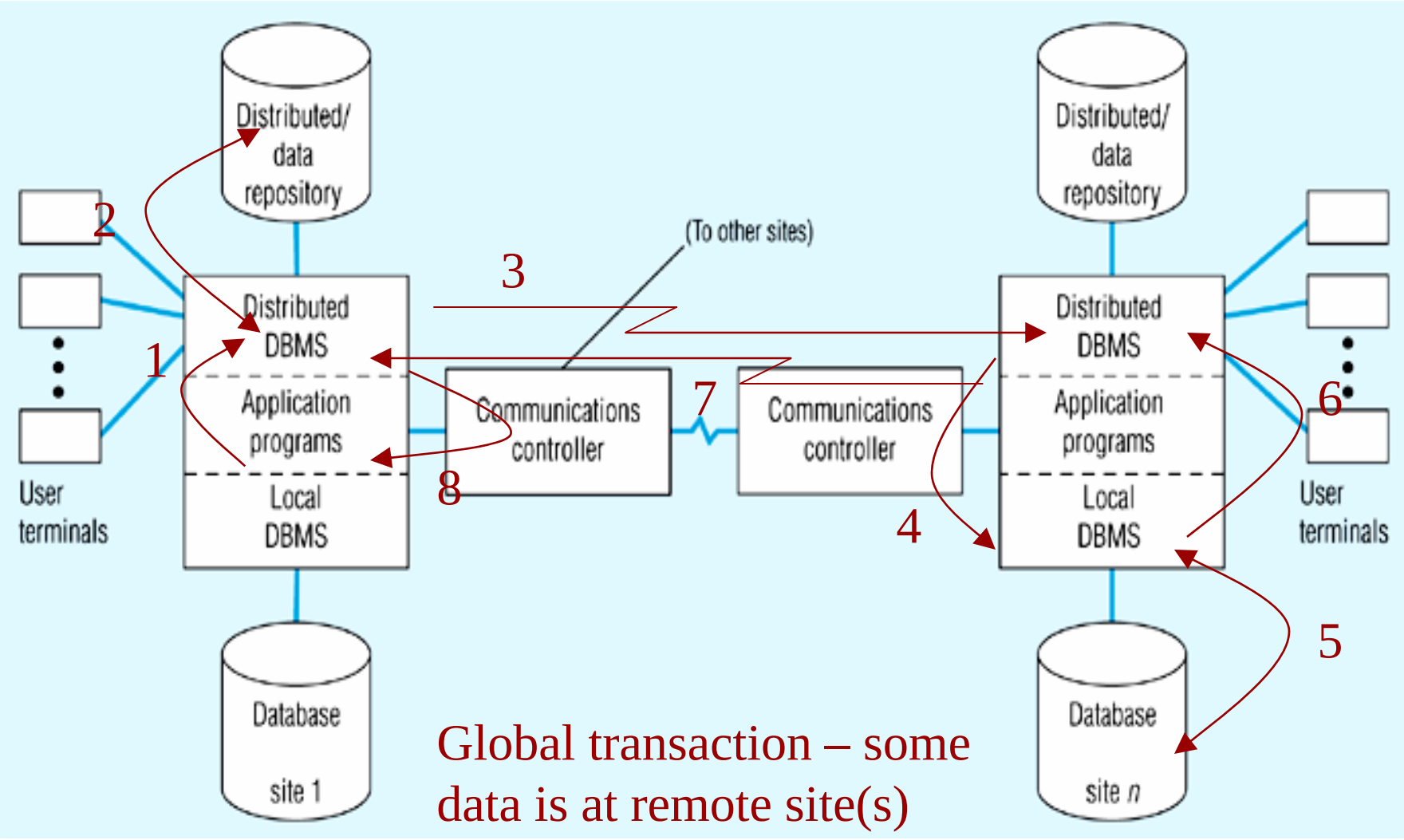
Processing a Local Transaction



Steps to Process a Global Transaction

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

Processing a Global Transaction



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 (cont.)

- **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