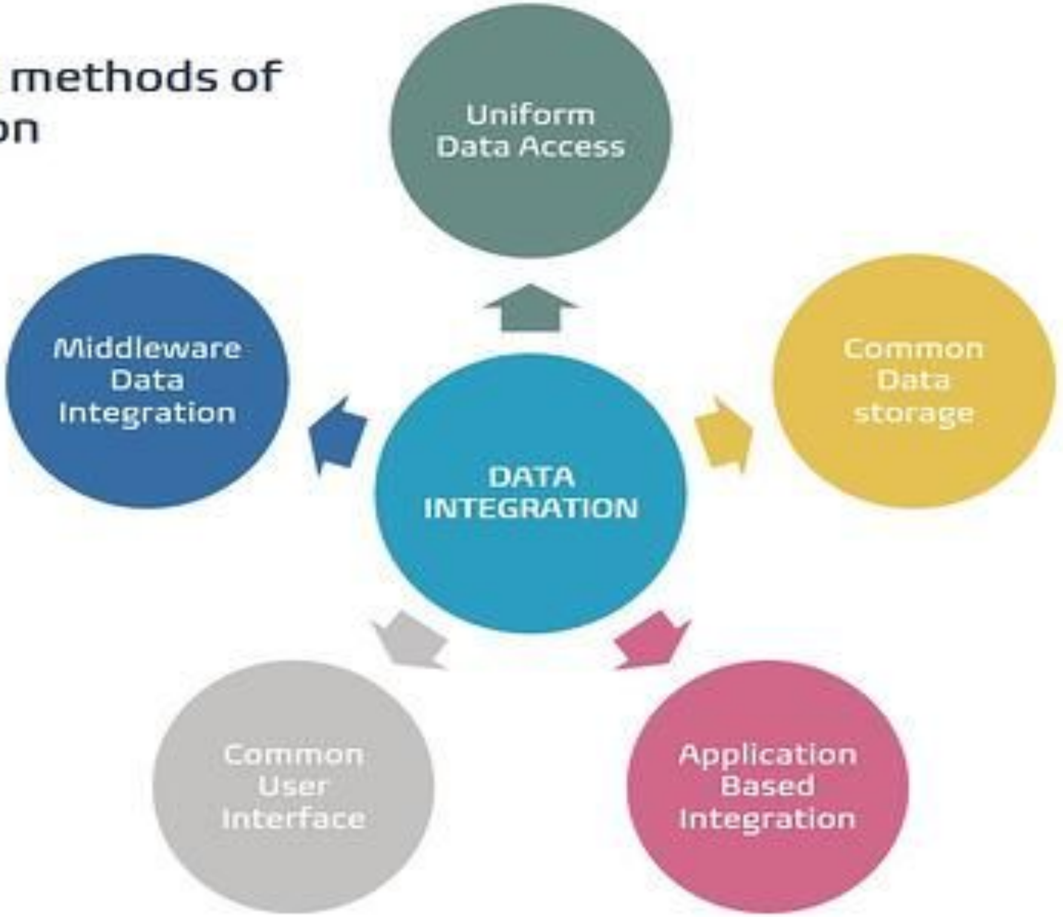
DATABASE INTEGRATION AND MANAGEMENT

Unit 5

Data Integration

- Data Integration is the process of combining data from different sources to help data managers and executives analyze it and make smarter business decisions. This process involves a person or system locating, retrieving, cleaning, and presenting the data.
- Data integration is a combination of technical and business processes used to combine different data from disparate sources in order to answer important questions. This process generally supports the analytic processing of data by aligning, combining, and presenting each data store to an end-user.
- Data integration allows organizations to better understand and retain their customers, support collaboration between departments, reduce project timelines with automated development, and maintain security and compliance.
- Data Integration process, the client sends a request to the master server for data. The master server then intakes the needed data from internal and external sources. The data is extracted from the sources, then consolidated into a single, cohesive data set. This is served back to the client for use.

Most common methods of data integration



Types of data integration

1. **Manual data integration**: Data managers must manually conduct all phases of the integration, from retrieval to presentation.
2. **Middleware data integration**: Middleware, a type of software, facilitates communication between legacy systems and updated ones to expedite integration.
3. **Application-based integration**: Software applications locate, retrieve, and integrate data by making data from different sources and systems compatible with one another.
4. **Uniform access integration**: A technique that retrieves and uniformly displays data, but leaves it in its original source.
5. **Common storage integration**: An approach that retrieves and uniformly displays the data, but also makes a copy of the data and stores it.

Manual data integration

Manual data integration occurs when a data manager oversees all aspects of the integration — usually by writing custom code. That means connecting the different data sources, collecting the data, and cleaning it, etc., without automation.

Some of the benefits are:

- Reduced cost: This technique requires little maintenance and typically only integrates a small number of data sources.
- Greater freedom: The user has total control over the integration.

Some of the cons are:

- Less access: A developer or manager must manually orchestrate each integration.
- Difficulty scaling: Scaling for larger projects requires manually changing the code for each integration, and that takes time.
- Greater room for error: A manager and/or analyst must handle the data at each stage.

This strategy is best for one-time instances, but it quickly becomes untenable for complex or recurring integrations because it is a very tedious, manual process. Everything from data collection, to cleaning, to presentation is done by hand, and those processes take time and resources.

Middleware data integration

Middleware is software that connects applications and transfers data between them and databases. It’s especially handy when a business is integrating stubborn legacy systems with newer ones, as middleware can act as an interpreter between these systems.

Some of the benefits are:

- **Better data streaming**: The software conducts the integration automatically and in the same way each time.
- **Easier access between systems**: The software is coded to facilitate communication between the systems in a network.

Some of the cons are:

- **Less access**: The middleware needs to be deployed and maintained by a developer with technical knowledge.
- **Limited functionality**: Middleware can only work with certain systems.

For businesses integrating legacy systems with more modern systems, middleware is ideal, but it’s mostly a communications tool and has limited capabilities for data analytics.

Application-based integration

In this approach, software applications do all the work. They locate, retrieve, clean, and integrate data from disparate sources. This compatibility makes it easy for data to move from one source to the other.

Some of the benefits include:

- **Simplified processes**: One application does all the work automatically.
- **Easier information exchange**: The application allows systems and departments to transfer information seamlessly.
- **Fewer resources are used**: Because much of the process is automated, managers and/or analysts can pursue other projects.

Some of the cons include:

- **Limited access**: This technique requires special, technical knowledge and a data manager and/or analyst to oversee application deployment and maintenance.
- **Inconsistent results**: The approach is unstandardized and varies from businesses offering this as a service.
- **Complicated setup**: Designing the application(s) to work seamlessly across departments requires developers, managers, and/or analysts with technical knowledge.
- **Difficult data management**: Accessing different systems can lead to compromised data integrity.

Sometimes this approach is called enterprise application integrity, because it’s common in enterprises working in hybrid cloud environments. These businesses need to work with multiple data sources — on-premises and in the cloud. This approach optimizes data and workflows between these environments

Uniform access integration

This technique accesses data from even more disparate sets and presents it uniformly. It does this while allowing the data to stay in its original location.

Some of the advantages are:

- **Lower storage requirements**: There is no need to create a separate place to store data.
- **Easier data access:** This approach works well with multiple systems and data sources.
- **Simplified view of data**: This technique creates a uniform appearance of data for the end user.

Some of the difficulties are:

- **Data integrity challenges**: Accessing so many sources can lead to compromising data integrity.
- **Strained systems**: Data host systems are not usually designed to handle the amount and frequency of data requests in this process.

For businesses needing to access multiple, disparate systems, this is an optimal approach. If the data request isn’t too burdensome for the host system, this approach can yield insights without the cost of creating a backup or copy of the data.

Common storage integration

This approach is similar to uniform access, except it involves creating and storing a copy of the data in a data warehouse. This leads to more versatility in the ways businesses can manipulate data, making it one of the most popular forms of data integration.

Some of the benefits include:

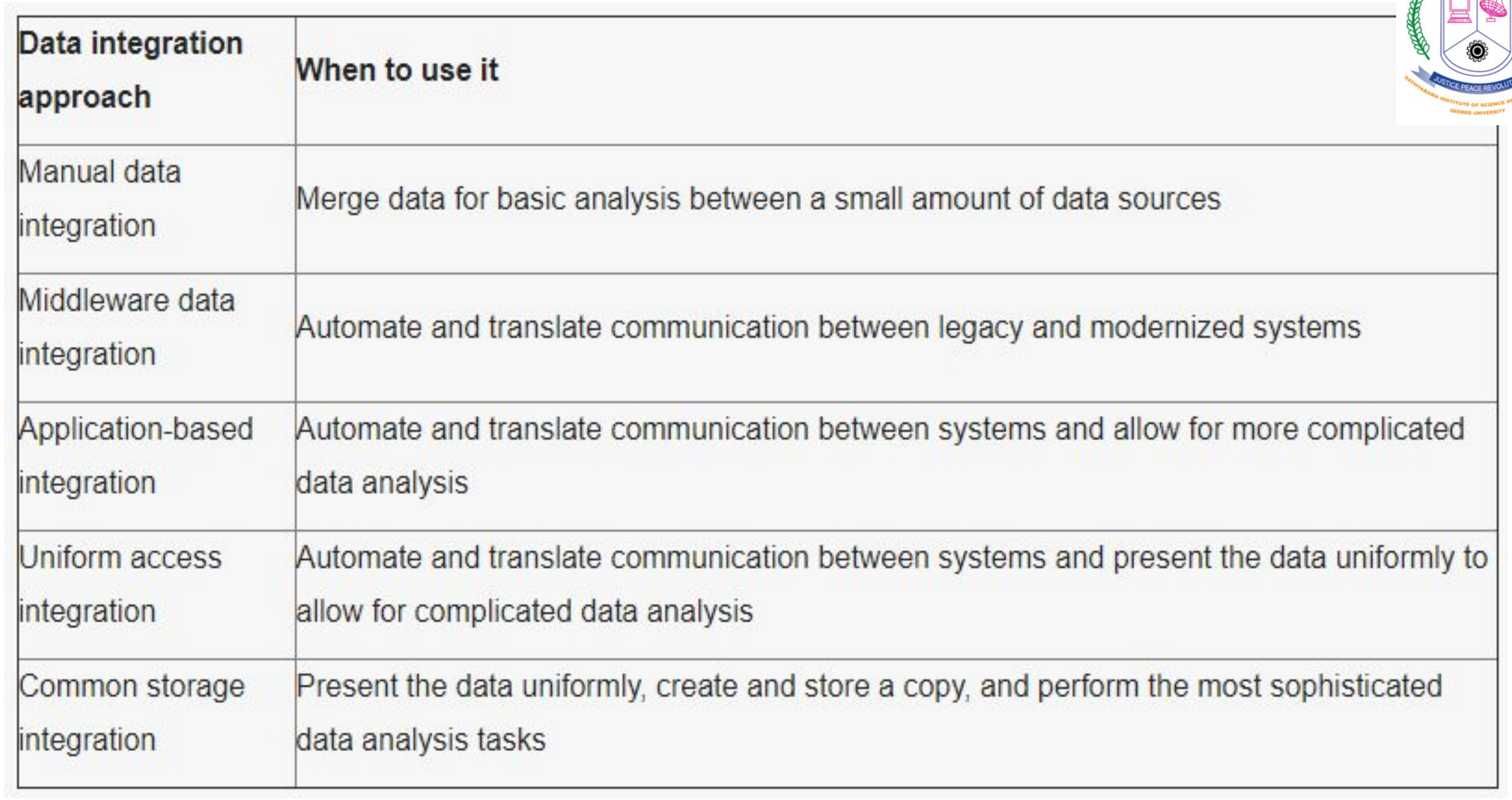
- **Reduced burden:** The host system isn’t constantly handling data queries.
- **Increased data version management control**: Accessing data from one source, versus multiple disparate sources, leads to better data integrity.
- **Cleaner data appearance**: The stored copy of data allows managers and/or analysts to run numerous queries while maintaining uniformity in the data’s appearance.
- **Enhanced data analytics**: Maintaining a stored copy allows manager and/or analysts to run more sophisticated queries without worrying about compromised data integrity.

Common storage integration

Some of the cons include:

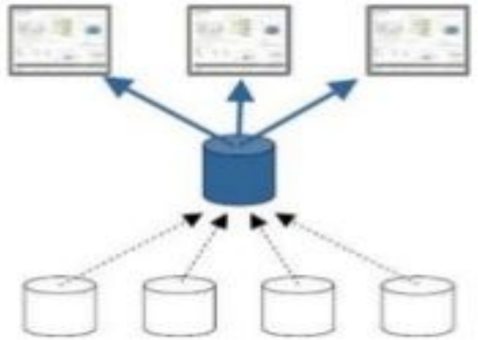
- **Increased storage costs**: Creating a copy of the data means finding and paying for a place to store it.
- **Higher maintenance costs**: Orchestrating this approach requires technical experts to set up the integration, oversee, and maintain it.

Common storage is the most sophisticated integration approach. If businesses have the resources, this is almost certainly the best approach, because it allows for the most sophisticated queries. That sophistication can lead to deeper insights.



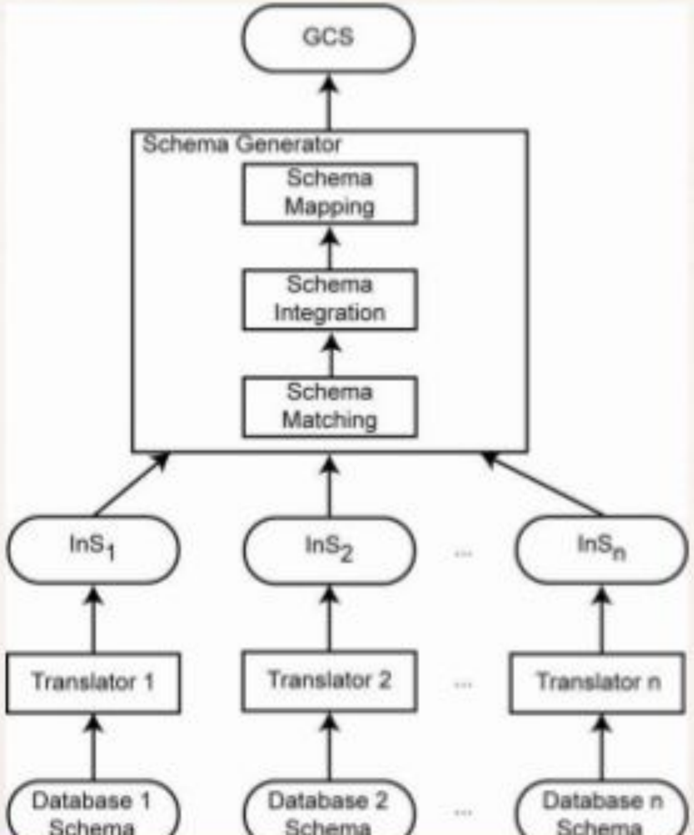
Data integration approach	When to use it
Manual data integration	Merge data for basic analysis between a small amount of data sources
Middleware data integration	Automate and translate communication between legacy and modernized systems
Application-based integration	Automate and translate communication between systems and allow for more complicated data analysis
Uniform access integration	Automate and translate communication between systems and present the data uniformly to allow for complicated data analysis
Common storage integration	Present the data uniformly, create and store a copy, and perform the most sophisticated data analysis tasks

Reasons for Integration



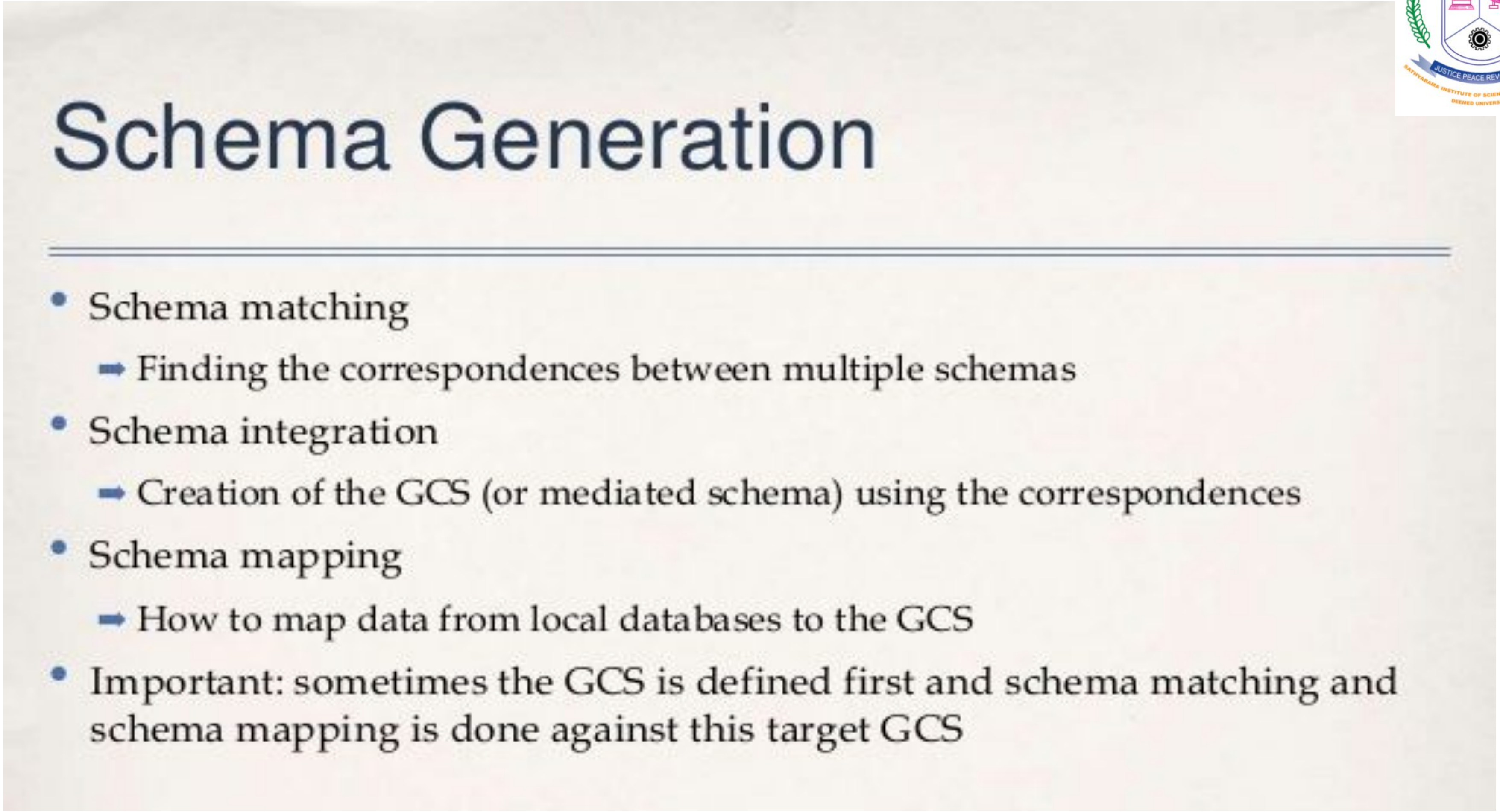
First, *given a* ***set of existing information systems****, an integrated view can be created to* ***facilitate information access*** *and reuse through a* ***single information access point***

Database Integration Process



Database Integration Issues

- Schema translation
  - ➡ Component database schemas translated to a common intermediate canonical representation
- Schema generation
  - ➡ Intermediate schemas are used to create a global conceptual schema



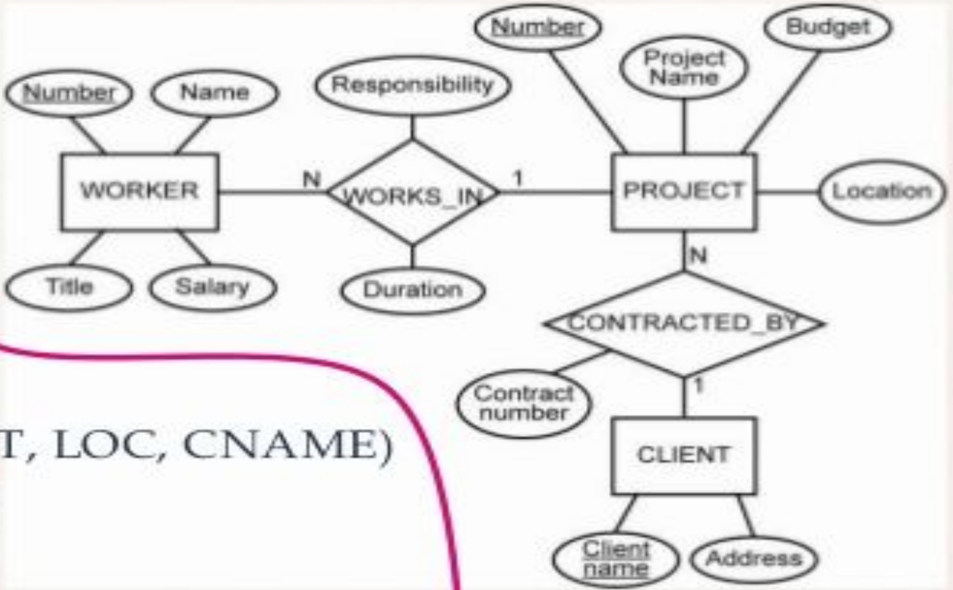
Schema Generation

- Schema matching
  - ➡ Finding the correspondences between multiple schemas
- Schema integration
  - ➡ Creation of the GCS (or mediated schema) using the correspondences
- Schema mapping
  - ➡ How to map data from local databases to the GCS
- Important: sometimes the GCS is defined first and schema matching and schema mapping is done against this target GCS

Running Example

E-R Model

Relational



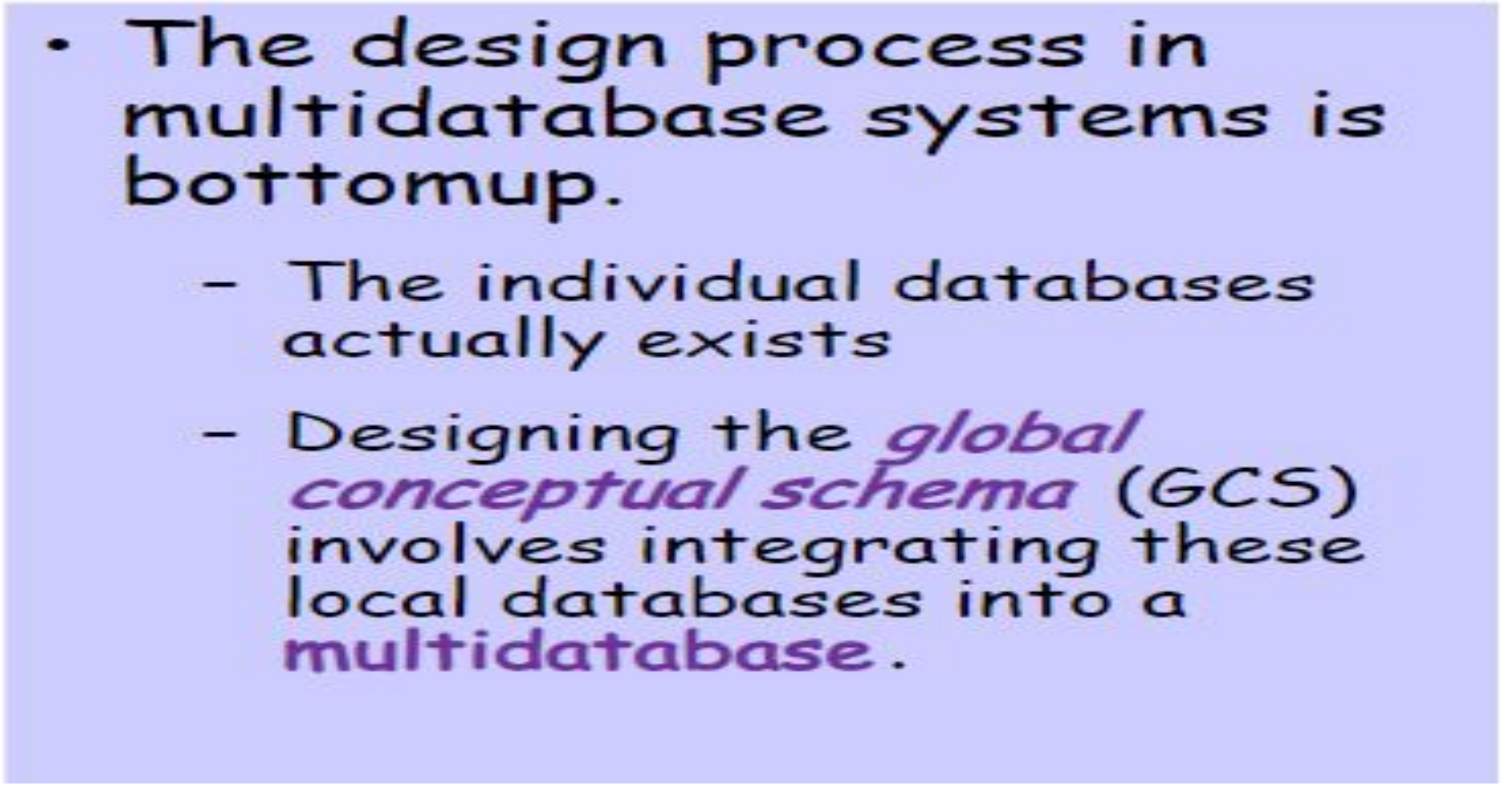
EMP(<u>ENO</u>, ENAME, TITLE)
PROJ(<u>PNO</u>, PNAME, BUDGET, LOC, CNAME)
ASG(<u>ENO, PNO</u>, RESP, DUR)
PAY(<u>TITLE</u>, SAL)

Schema Integration

Schema integration follows the translation process and generates the GCS by integrating the intermediate schemas.

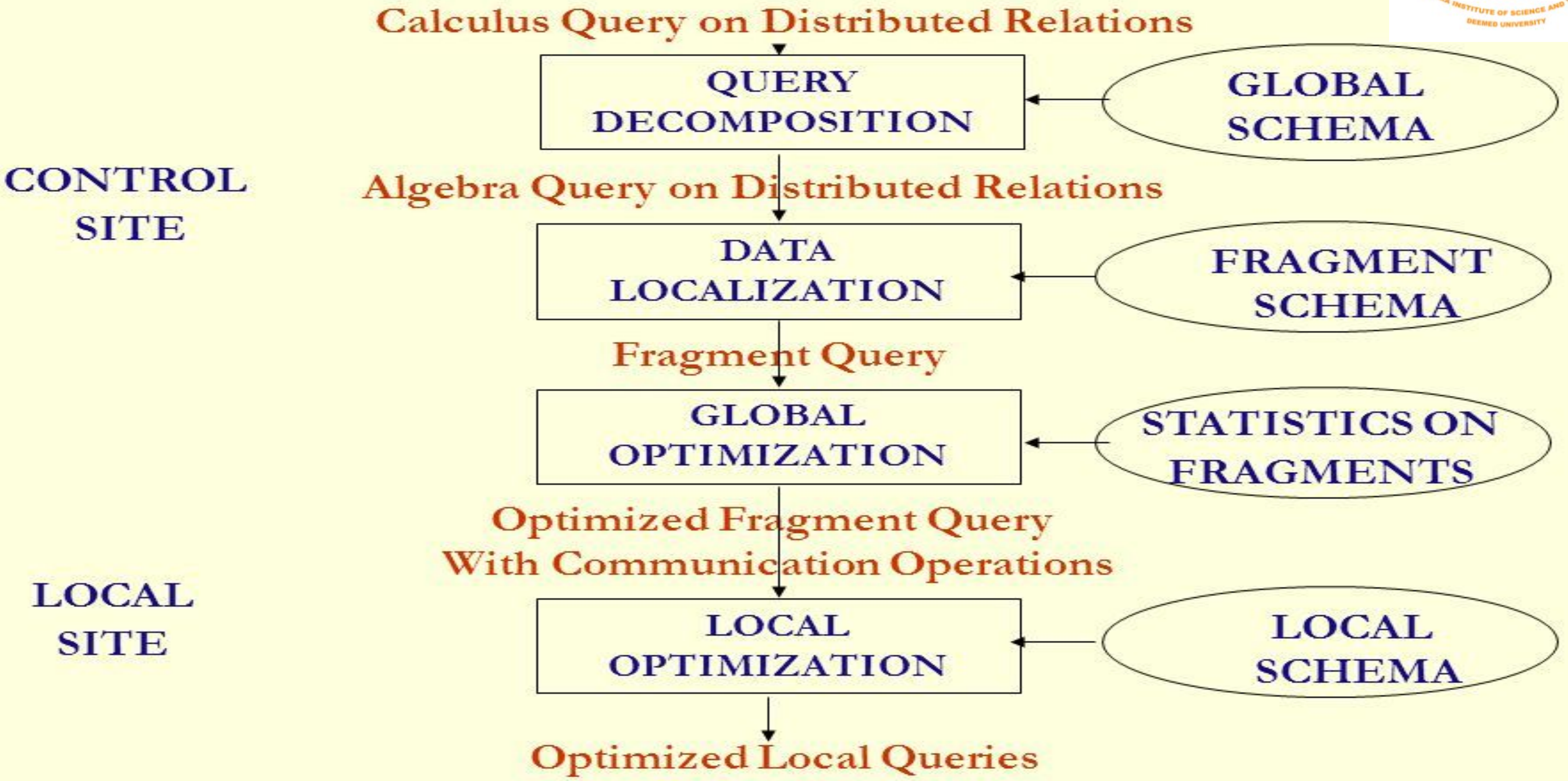
- Identify the components of a database which are related to one another.
  - Two components can be related as (1) equivalent, (2) one contained in the other one, (3) overlapped, or (4) disjoint.
- Select the best representation for the GCS.
- Integrate the components of each intermediate schema.

Scheme Translation



- The design process in multidatabase systems is bottomup.
  - The individual databases actually exists
  - Designing the *global conceptual schema* (GCS) involves integrating these local databases into a **multidatabase**.

Layers of Query Processing



Issues in Multidatabase Query Processing

- In distributed DBMSs query processors have to deal only with data distribution across multiple sites.
- In a distributed multidatabase environment data is distributed not only across sites but also across multiple databases.
- This difference increases the parties involved in processing the query from two in the distributed DBMS to three in the distributed multidatabase environment.

Query Optimization and Execution

- There are three main problems of query optimization in multidatabase systems
  I. Heterogeneous Cost Modeling
  II. Heterogeneous Query Optimization
  III. Adaptive Query Processing

Heterogeneous Cost Modeling

- The global cost function definition and the associated problem of obtaining cost-related information from component DBMSs is the most studied of the three problems.
- Primarily interested in determining the cost of the lower levels of a query execution tree

This corresponds to the parts of the query executed at component DBMSs Three approaches:

I. Black Box Approach

II. Customized Approach

III. Dynamic Approach

Black Box Approach

- This approach treats each component DBMS as a block box, running some test queries against it and then determines the necessary cost information
- Cost functions are expressed logically rather than on the basis of physical characteristics.
- The cost function for component DBMSs is expressed as:

Cost = initialization cost + cost to find qualifying tuples + cost to process selected tuples

- The individual terms of this formula will differ for different operators.
- The difficulty is determining the coefficients is the formula since they will change with different component DBMSs.
- The major drawback of this approach is that the cost model is common for all component DBMSs and may not capture their individual specifics.
- So, it might fail to accurately estimate the cost of a query executed at a component DBMS that exposes unforeseen behavior.

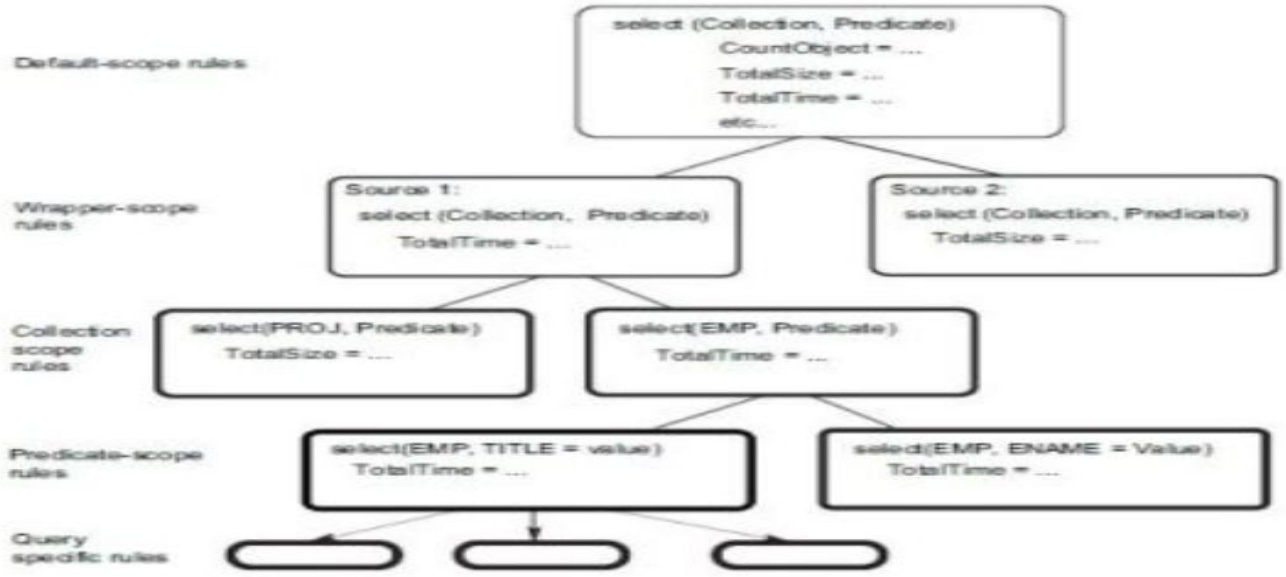
Customized Approach

- Customized Approach Uses previous knowledge about the component DBMSs and their external characteristics to subjectively determine the cost information
- The basis is that the query processors of the component DBMSs are too different to be represented by a unique cost model
- Assumes that the ability to accurately estimate the cost of local subqueries will improve global optimization.
- Provides framework to integrate the component DBMSs cost model into the mediator query optimizer.
- Extends the wrapper interface such that the mediator gets some specific cost information from each of the wrappers
- The wrapper developer is free to provide a cost model, partially or entirely
- This poses a challenge to integrate this cost description into the mediator query optimizer.
- Two main solutions:
- One: Provide the logic within the wrapper to compute three cost estimates
- I. The time to initiate the query process and receive the first result item (reset_cost)
- II. The time to get the next item (advance_cost)
- III. The result cardinality
- This makes the total query cost: Total_access_cost = reset_cost +(cardinality – 1) * advance_cost

Customized Approach

- Two: Is to use a hierarchical generic cost model
- This is where each node represents a cost rule that associates a query pattern with a cost function for various cost parameters.
- The five cost rules are:
  1. Default-scope rules
  2. Wrapper-scope rules
  3. Collection scope rules
  4. Predicate-scope rules
  5. Query specific rules

Customized Approach



Dynamic Approach

- This monitors the run-time behavior of component DBMSs and dynamically collects the cost information
- In most cases execution environment factors are constantly changing
- Three classes of environmental factors based on dynamicity:

I. First class (frequently changing, every second to every minute) are CPU load, I/O throughput and available memory

II. Second class (slowly changing, every hour to every day) are DBMS configuration parameters, physical data organization on disk and database schema.

III. Third class (almost stable, every month to every year) are DBMS type, database location, and CPU speed

Dynamic Approach

- One approach is to extend the sampling method and consider user queries as new samples for environments where network connection, data storage and available memory change over time.
- Query response time is measured to adjust the cost model parameters at run time for subsequent queries.
- This avoids the overhead of processing queries periodically
- However, it still requires heavy computation to solve the cost model equations and does not guarantee that the cost model's precision improves over time
- Qualitative is a better solution.
- This defines are system contention level as a combined effect of frequently changing factors on query cost.
- System contention level is divided into discrete categories:
    - High
    - Medium
    - Low
    - No system contention

Dynamic Approach

- These distinctions allow for defining a multi-category cost model that provides accurate cost estimates all while the dynamic factors vary.
- Cost model is initially calibrated using probing queries
- Current system contention level is computed over time
- Assumes query executions are short, so environmental factors remain pretty constant during query execution

Heterogeneous Query Optimization

One component DBMS may support only simple select operations while another may support complex queries involving join and aggregate.

Depending on how the wrappers export such capabilities, query processing at the mediator level can be more or less complex.

1 ---Query based: The wrappers support the same query capability, e. g. , a subset of SQL, which is translated to the capability of the component DBMS.

2 ---Operator based: The wrappers export the capabilities of the component DBMSs through compositions of relational operators.

Thus, there is more flexibility in defining the level of functionality between the mediator, any functionality that may not be supported by component DBMSs (e. g. , join) will need to be implemented at the mediator.

Query-based Approach

DBMSs appear homogeneous to the mediator, one approach is to use a distributed cost-based query optimization algorithm with a heterogeneous cost model extensions are needed to convert the distributed execution plan into subqueries

Hybrid two-step optimization technique

1. static plan is produced by a centralized cost-based query optimizer

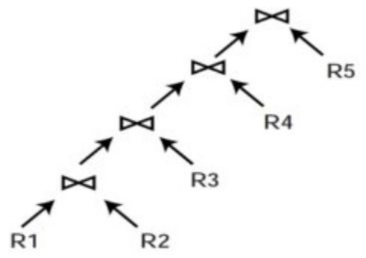
2. At startup time, an execution plan is produced by carrying out site selection and allocating the subqueries to the sites.

Cost-based query optimizer

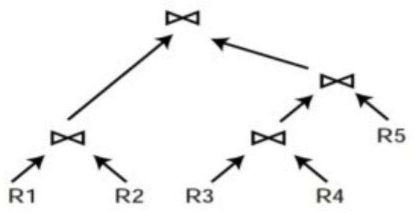
- first generate a left linear join tree, and then convert it to a bushy tree.
- A hybrid algorithm that concurrently performs a bottom-up and top-down sweep of the left linear join execution tree, transforming it, step-by-step, to a bushy tree.
- The algorithm has two pointers
    - Bottom UAN (Upper Anchor Nodes) Is set to the grandparent of the leftmost root node
    - Top UAN Is set to the root For each UAN the algorithm selects a lower anchor node (LAN)

The LAN is chosen such that its right child subtree's response time is close to the corresponding UAN's right child subtree's response time.

This helps in keeping the transformed bushy tree balanced, which reduces the response time.

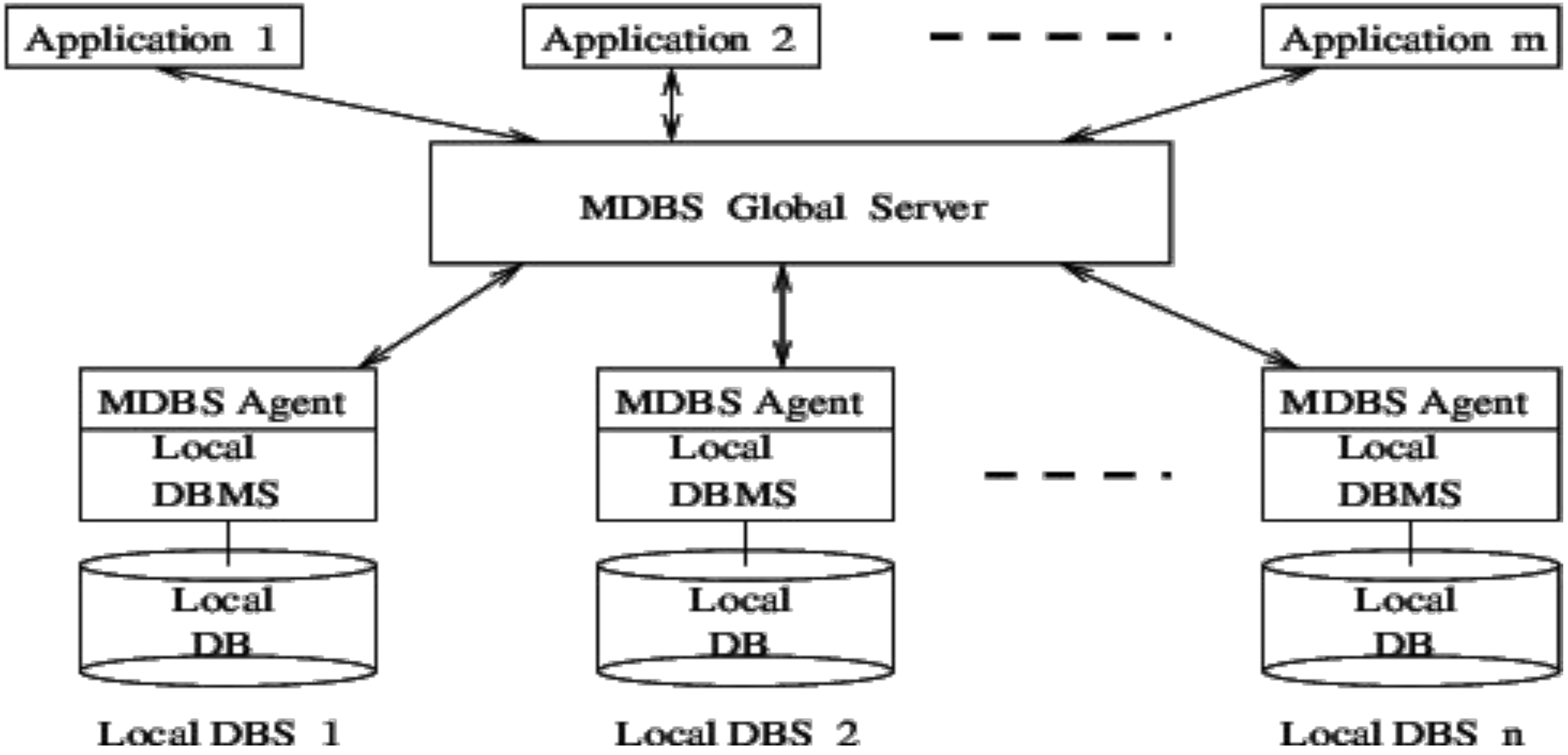


(a) Left Linear Join Tree



(b) Bushy Join Tree

Multidatabase Architecture



Multidatabase

- A multidatabase system (MDBS) is a facility that allows users access to data located in multiple autonomous database management systems (DBMSs).
- In such a system,*global transactions* are executed under the control of the MDBS. Independently,*local transactions* are executed under the control of the local DBMSs.
- Each local DBMS integrated by the MDBS may employ a different transaction management scheme.
- In addition, each local DBMS has complete control over all transactions (global and local) executing at its site, including the ability to abort at any point any of the transactions executing at its site.
- Typically, no design or internal DBMS structure changes are allowed in order to accommodate the MDBS.
- Furthermore, the local DBMSs may not be aware of each other and, as a consequence, cannot coordinate their actions.
- Thus, traditional techniques for ensuring transaction atomicity and consistency in homogeneous distributed database systems may not be appropriate for an MDBS environment.

Recovery in Multidatabase Systems

- In some cases, a single transaction, called a **multidatabase transaction**, may require access to multiple databases.
- These databases may even be stored on different types of DBMSs; for example, some DBMSs may be relational, whereas others are object-oriented, hierarchical, or network DBMSs.
- In such a case, each DBMS involved in the multidatabase transaction may have its own recovery technique and transaction manager separate from those of the other DBMSs.
- This situation is somewhat similar to the case of a distributed database management system , where parts of the database reside at different sites that are connected by a communication network.
- To maintain the atomicity of a multidatabase transaction, it is necessary to have a two-level recovery mechanism.
- A **global recovery manager**, or **coordinator**, is needed to maintain information needed for recovery, in addition to the local recov-ery managers and the information they maintain (log, tables).

Recovery in Multidatabase Systems

The coordinator usu-ally follows a protocol called the **two-phase commit protocol**, whose two phases can be stated as follows:

Phase 1. When all participating databases signal the coordinator that the part of the multidatabase transaction involving each has concluded, the coordinator sends a message *prepare for commit* to each participant to get ready for committing the transaction. Each participating database receiving that message will force-write all log records and needed information for local recovery to disk and then send a *ready to commit* or *OK* signal to the coordinator. If the force-writing to disk fails or the local transaction cannot commit for some reason, the participating database sends a *cannot commit* or *not OK* signal to the coordinator. If the coordinator does not receive a reply from the database within a certain time out interval, it assumes a *not OK* response.

Phase 2. If *all* participating databases reply *OK*, and the coordinator's vote is also *OK*, the transaction is successful, and the coordinator sends a *commit* signal for the transaction to the participating databases. Because all the local effects of the transaction and information needed for local recovery have been recorded in the logs of the participating databases, recovery from fail-ure is now possible. Each participating database completes transaction com-mit by writing a [commit] entry for the transaction in the log and permanently updating the database if needed. On the other hand, if one or more of the participating databases or the coordinator have a *not OK* response, the transaction has failed, and the coordinator sends a message to *roll back* or UNDO the local effect of the transaction to each participating database. This is done by undoing the transaction operations, using the log.

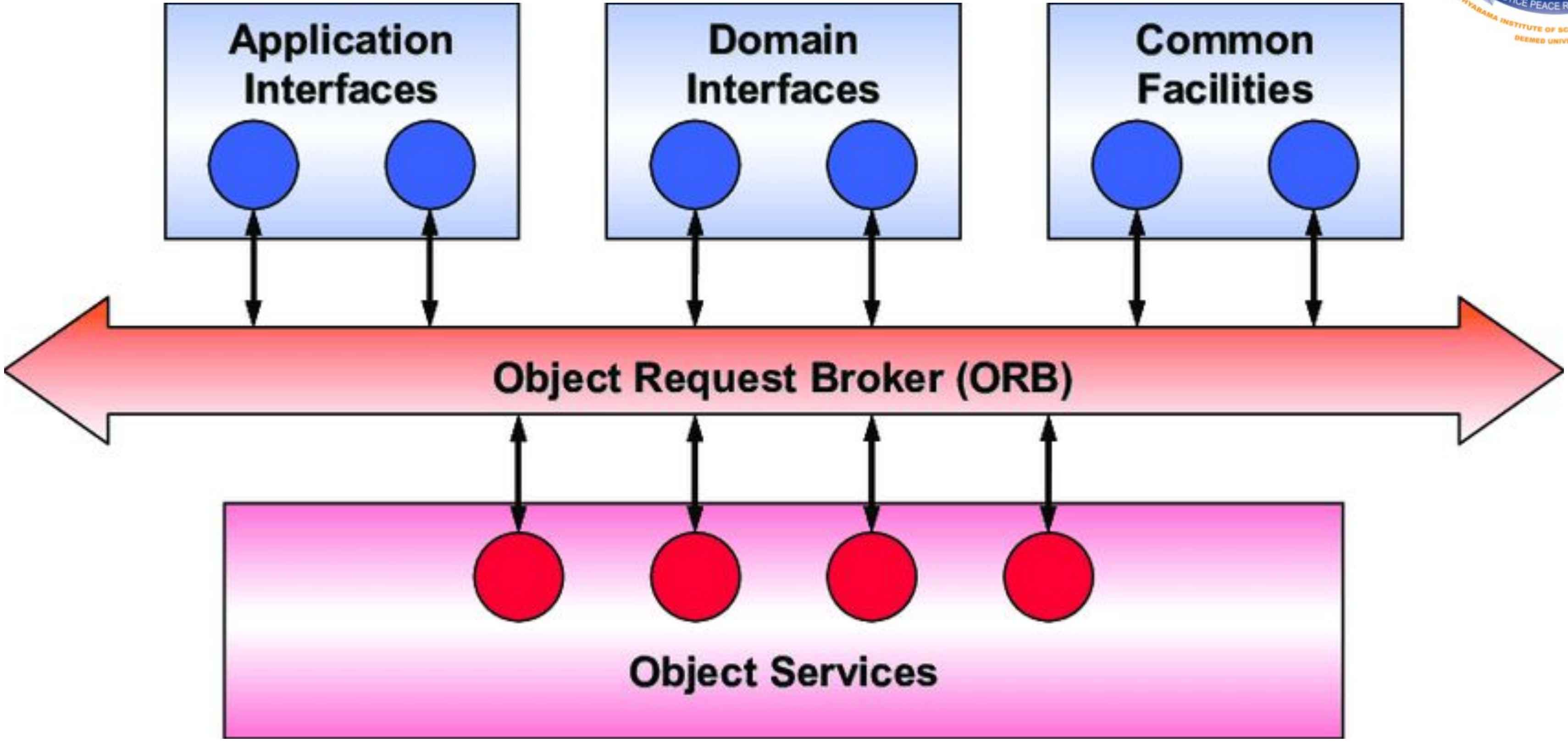
Recovery in Multidatabase Systems

The net effect of the two-phase commit protocol is that either all participating data-bases commit the effect of the transaction or none of them do.

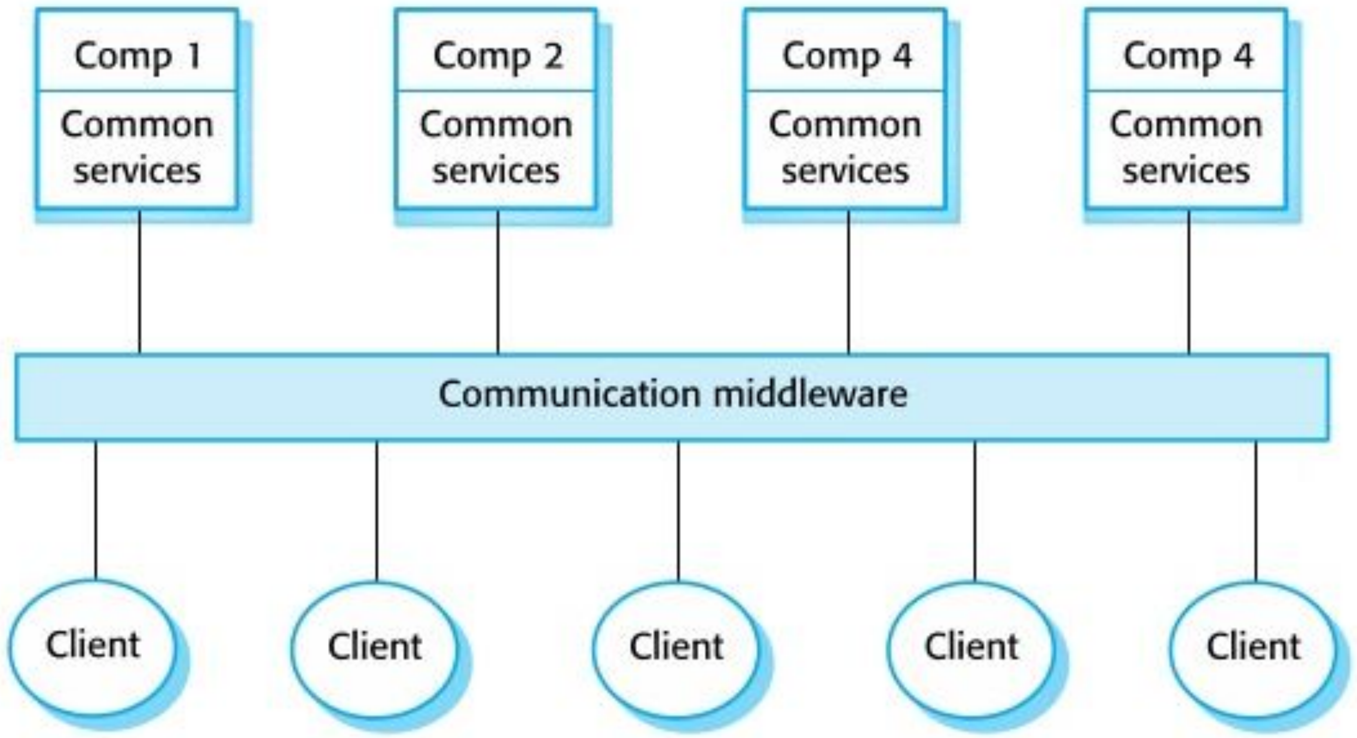
In case any of the participants—or the coordinator—fails, it is always possible to recover to a state where either the transaction is committed or it is rolled back.

A failure during or before Phase 1 usually requires the transaction to be rolled back, whereas a failure during Phase 2 means that a successful transaction can recover and commit.

Object Management Architecture



Distributed Component Model



Comp 1
Common services
Comp 2
Common services
Comp 4
Common services
Comp 4
Common services
Communication middleware
Client
Client
Client
Client
Client