INTRODUCTION

- <u>In Centralized databases</u>:

  Data is located in one place (one server)

  All DBMS functionalities are done by that server

  Enforcing ACID properties of transactions

  Concurrency control, recovery mechanisms

  Answering queries

- <u>In Distributed databases</u>:

  Data is stored in multiple places (each is running a DBMS)

  New notion of distributed transactions

  DBMS functionalities are now distributed over many machines

  Revisit how these functionalities work in distributed environment

WHY DISTRIBUTED DATABASES

- Data is too large
- Applications are by nature distributed

  Bank with many branches

  Chain of retail stores with many locations

  Library with many branches
- Get benefit of distributed and parallel processing

  Faster response time for queries

PARALLEL VS. DISTRIBUTED DATABASES

- Distributed processing usually imply parallel processing (not vise versa)

  Can have parallel processing on a single machine
- Assumptions about architecture
- Parallel Databases

-Machines are physically close to each other, e.g., same server room

-Machines connects with dedicated high-speed LANs and switches

-Communication cost is assumed to be small

-Can shared-memory, shared-disk, or shared-nothing architecture

- Distributed Databases

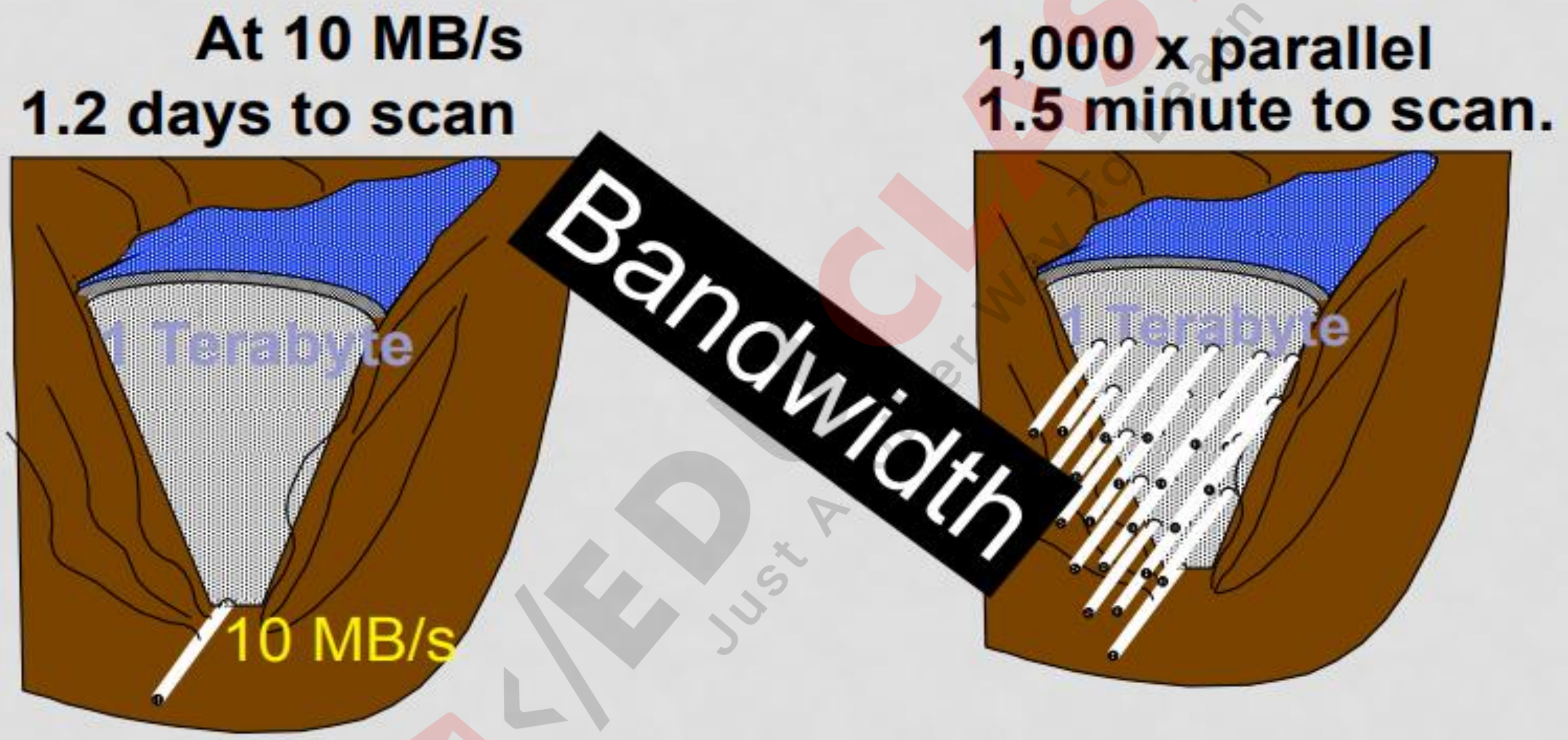
-Machines can far from each other, e.g., in different continent

-Can be connected using public-purpose network, e.g., Internet

-Communication cost and problems cannot be ignored

-Usually shared-nothing architecture

WHY PARALLEL PROCESSING



- Divide a big problem into many smaller ones to be solved in parallel
- Increase bandwidth (in our case decrease queries' response time)

Ideal advantages of parallel systems

- High performance
  - short response time or total time
  - good load balancing among processors
- High availability
  - handling failures of hardware elements
  - redundancy and consistency
- Extensibility
  - more processing and storage power can be added

High availability

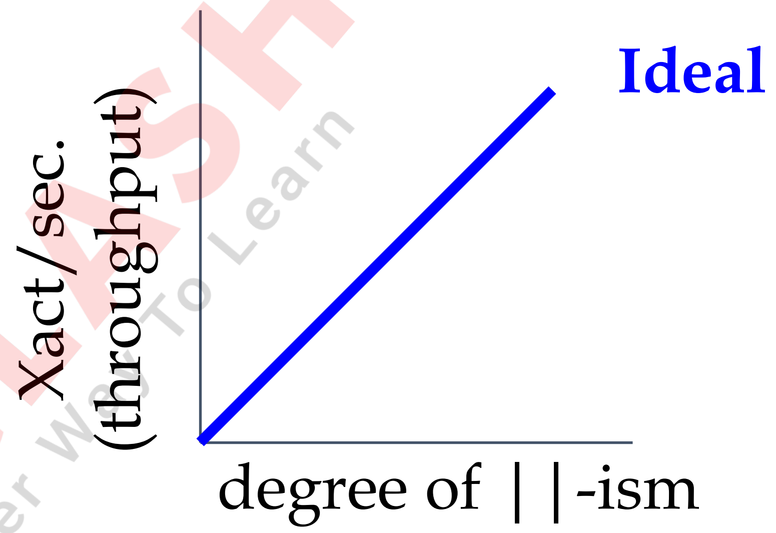
- While parallelism is motivated by performance considerations, several distinct issues motivate data distribution:
- 1. Increased Availability : If a site containing a relation goes down, the relation continues to be available if a copy is maintained at another site.
- 2. Distributed Access to Data : An organization may have branches in several cities. Although analysts may need to access data corresponding to different sites, we usually find locality in the access patterns and this locality can be exploited by distributing the data accordingly.

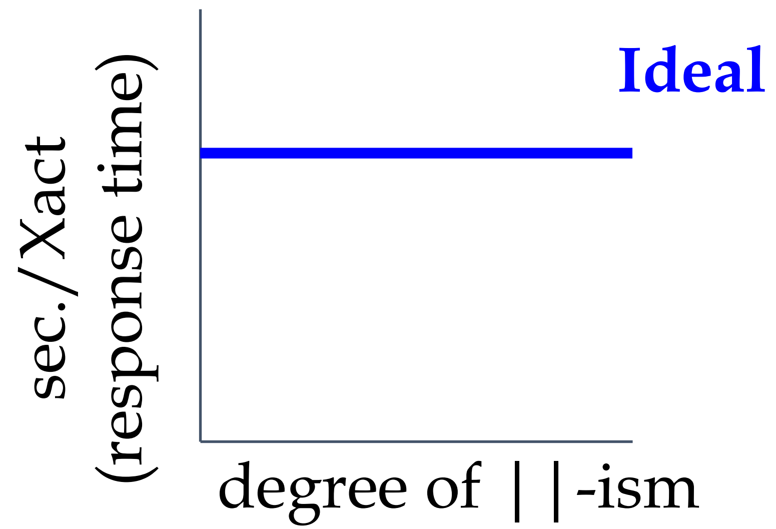
Cont…

- 3. Analysis of Distributed Data: Organizations want to examine all the data available to them, even when it is stored across multiple sites and on multiple database systems. Support for such integrated access involves many issues.

Extensibility

- Speed-Up
  - More resources means proportionally less time for given amount of data.
- Scale-Up
  - If resources increased in proportion to increase in data size, time is constant.

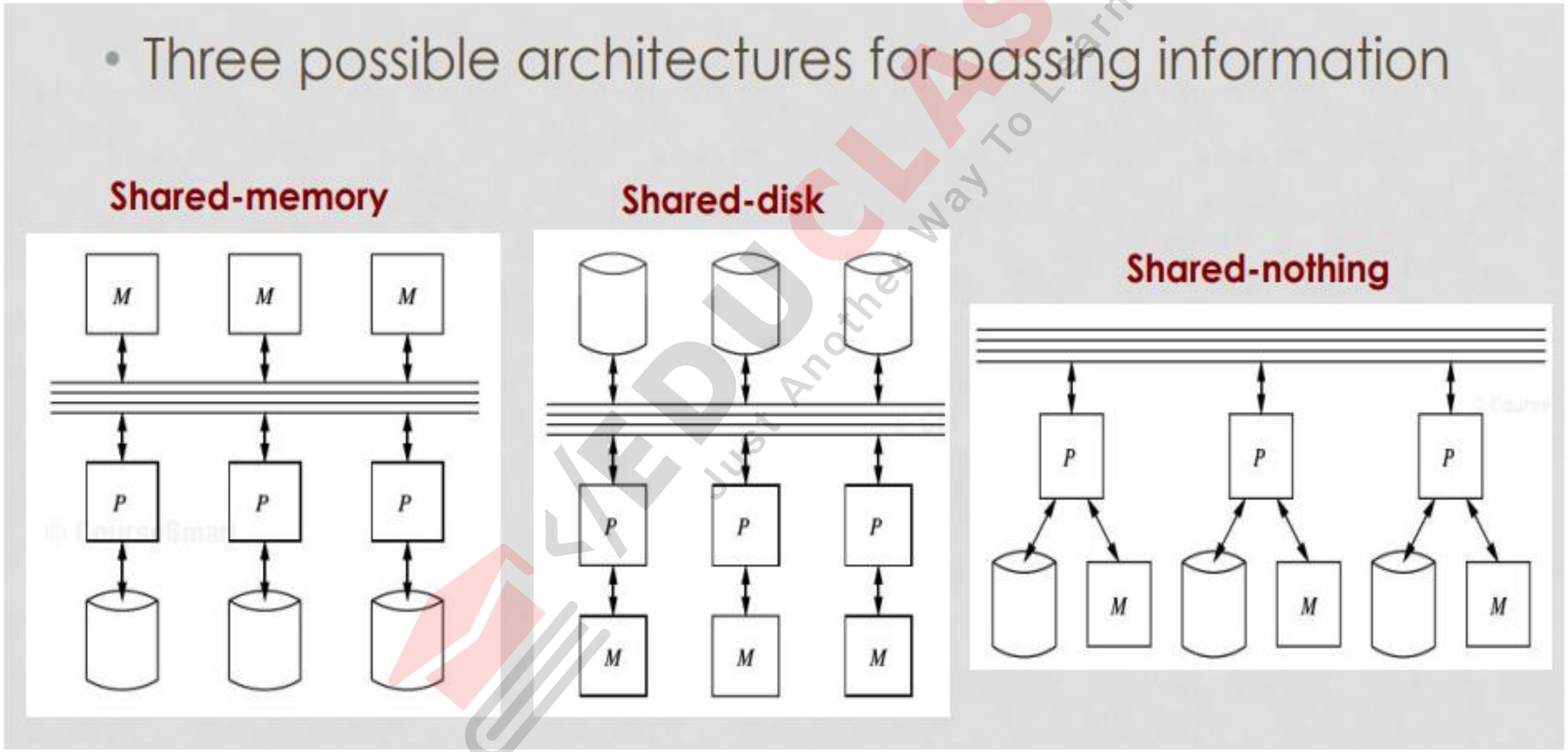




Bottlenecks

- Start-up - many processors => long start-up time
- Interference - more processors => more communications
- Data skew - it makes data distribution difficult

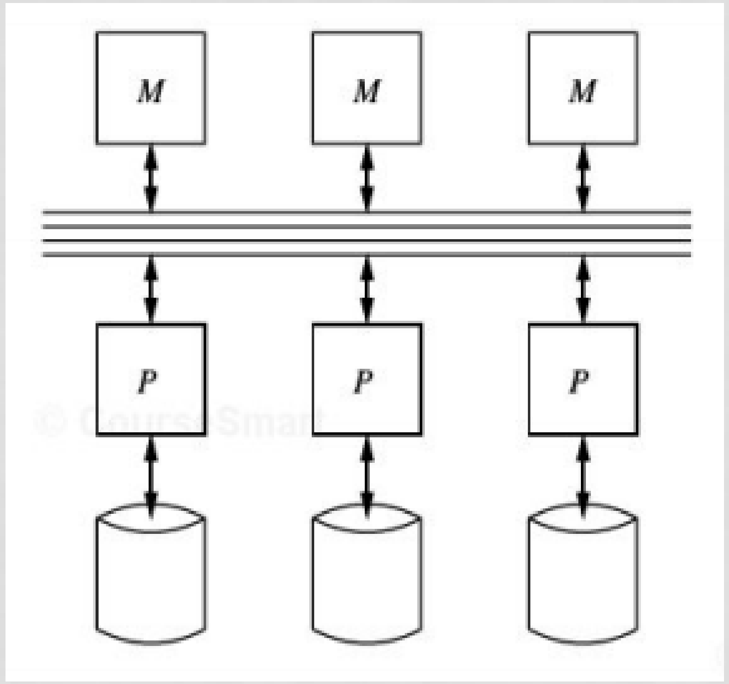
DIFFERENT ARCHITECTURE



- Three possible architectures for passing information

1- SHARED-MEMORY ARCHITECTURE

- Every processor has its own disk
- Single memory address-space for all processors
  - Reading or writing to far memory can be slightly more expensive
- Every processor can have its own local memory and cache as well



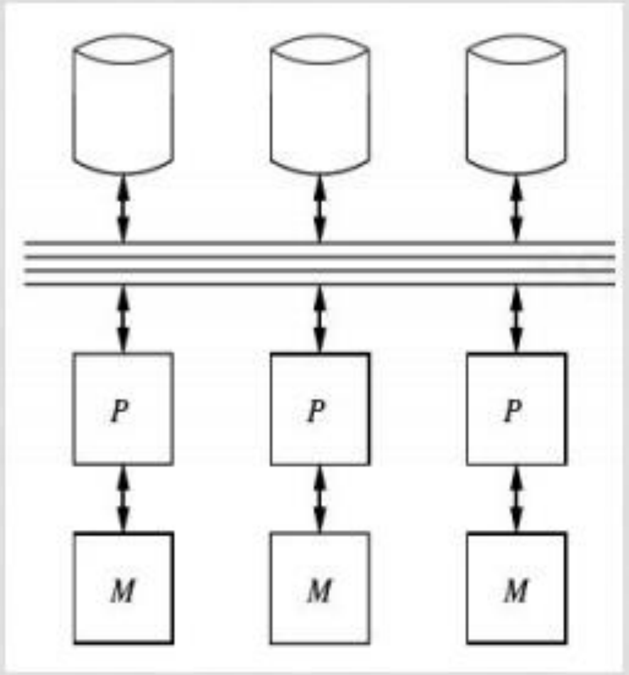
Cont….

- Characteristics
  - any CPU has access to any memory module or disk unit
- Advantages
- Simple implementation
- Establishes effective communication between processors through single memory addresses space.
- Above point leads to less communication overhead.

- Disadvantages
- Higher degree of parallelism (more number of concurrent operations in different processors) cannot be achieved due to the reason that all the processors share the same interconnection network to connect with memory. This causes Bottleneck in interconnection network (Interference), especially in the case of Bus interconnection network.
- Addition of processor would slow down the existing processors.
- Cache-coherency should be maintained. That is, if any processor tries to read the data used or modified by other processors, then we need to ensure that the data is of latest version.
- Degree of Parallelism is limited. More number of parallel processes might degrade the performance.

2- SHARED-DISK ARCHITECTURE

- Every processor has its own memory (not accessible by others)
- All machines can access all disks in the system
- Number of disks does not necessarily match the number of processors



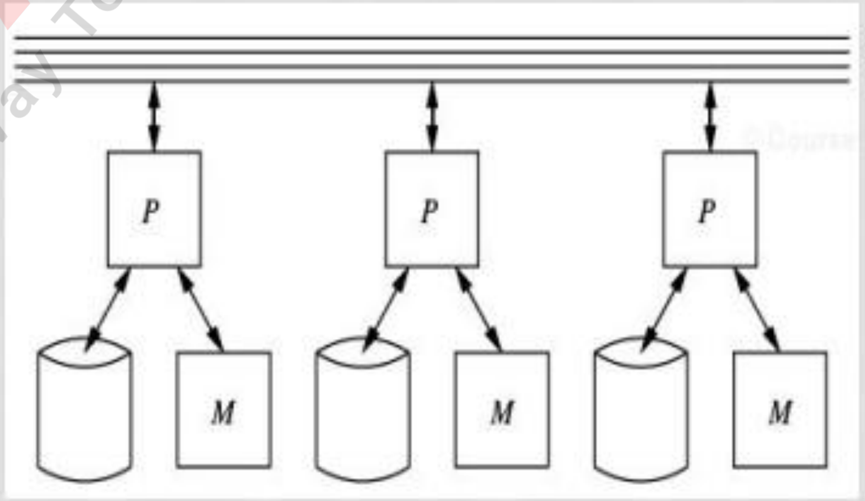
Cont….

- <u>Characteristics</u>
  - any CPU has access to any disk unit but exclusive access to its main memory

Advantages	Disadvantages
• Failure of any processors would not stop the entire system (Fault tolerance) • Interconnection to the memory is not a bottleneck. (It was bottleneck in Shared Memory architecture) • Support larger number of processors (when compared to Shared Memory architecture)	• Interconnection to the disk is bottleneck as all processors share common disk setup. • Inter-processor communication is slow. The reason is, all the processors have their own memory. Hence, the communication between processors need reading of data from other processors' memory which needs additional software support.

3- SHARED-NOTHING ARCHITECTURE

- Most common architecture nowadays
- Every machine has its own memory and disk
  - Many cheap machines (commodity hardware)
- Communication is done through high-speed network and switches
- **Usually machines can have a hierarchy**
  - Machines on same rack
  - Then racks are connected through high-speed switches



- **Scales better**
- **Easier to build**
- **Cheaper cost**

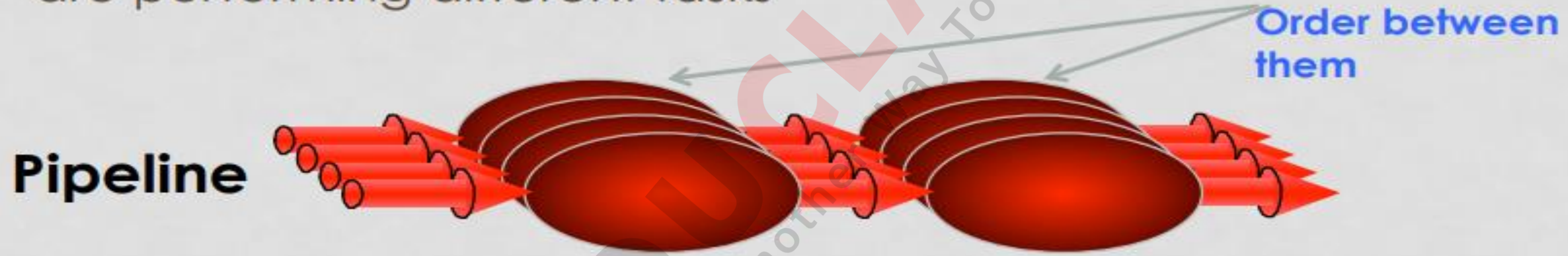
Cont..

- Characteristics
  - any CPU has only exclusive access to its main memory and disk

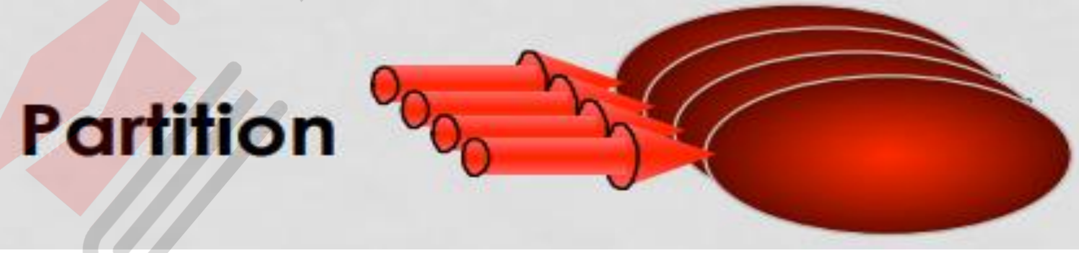
Advantages	Disadvantages
• Number of processors used here is scalable. That is, the design is flexible to add more number of computers. • Unlike in other two architectures, only the data request which cannot be answered by local processors need to be forwarded through interconnection network.	• Non-local disk accesses are costly. That is, if one server receives the request. If the required data not available, it must be routed to the server where the data is available. It is slightly complex. • Communication cost involved in transporting data among computers.

TYPES OF PARALLELISM

- **Pipeline Parallelism (Inter-operator parallelism)**
  - Ordered (or partially ordered) tasks and different machines are performing different tasks



- **Partitioned Parallelism (Intra-operator parallelism)**
  - A task divided over all machines to run in parallel

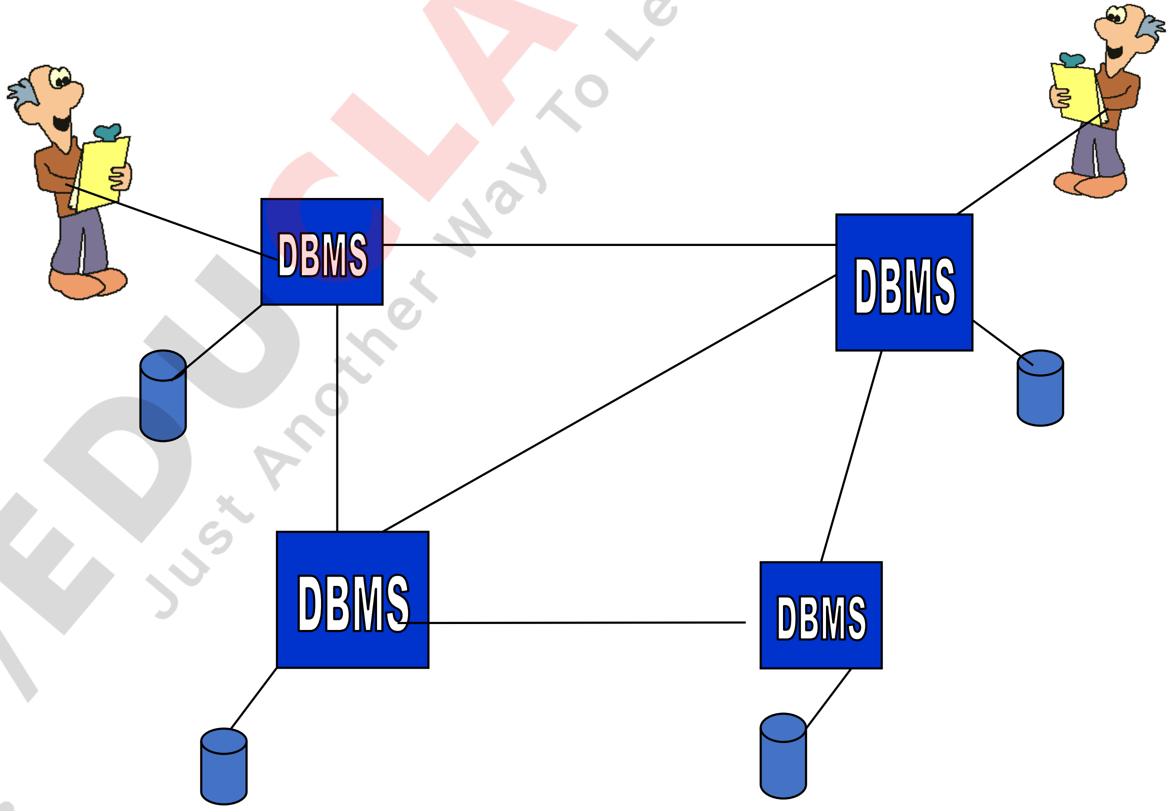


Distributed database

- **Communication Network- DBMS and Data at each node**

•Users are unaware of the distribution of the data

≡Location transparency



Distributed Databases

- Data is stored at several sites, each managed by a DBMS that can run independently.
- Following properties are desirable:
- Distributed Data Independence

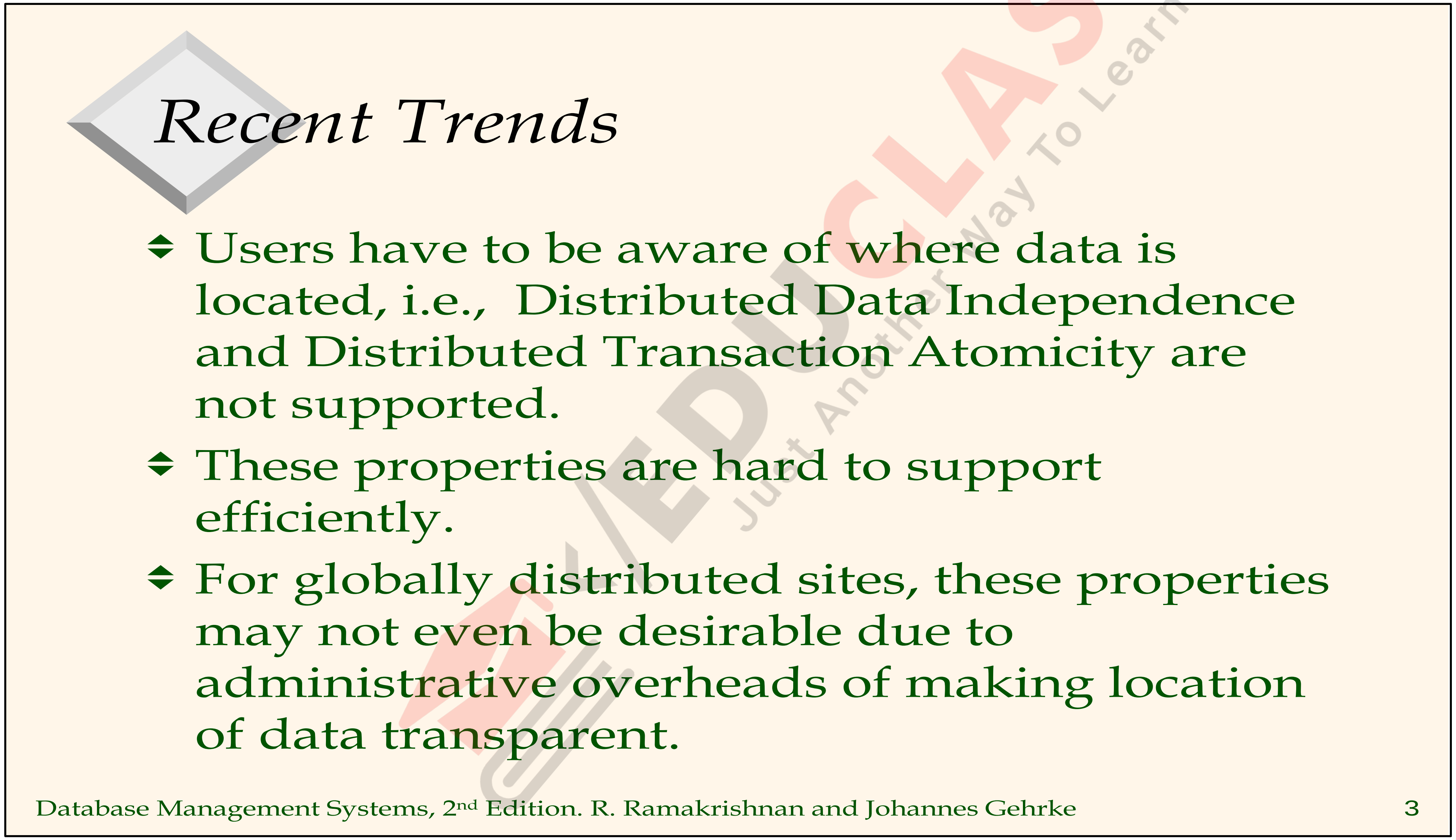
  Users should not have to know where data is located (extends Physical and Logical Data Independence principles)
- Distributed Transaction Atomicity

  Users should be able to write Xacts accessing multiple sites just like local Xacts

Distributed Databases

- ***Advantages:***
  - Reliability
  - Performance
  - Growth (incremental)
  - Local control
  - Transparency
- ***Disadvantages:***
  - Complexity of:
    - Query opt.
    - Concurrency control
    - Recovery
    - Catalog management

Distributed database



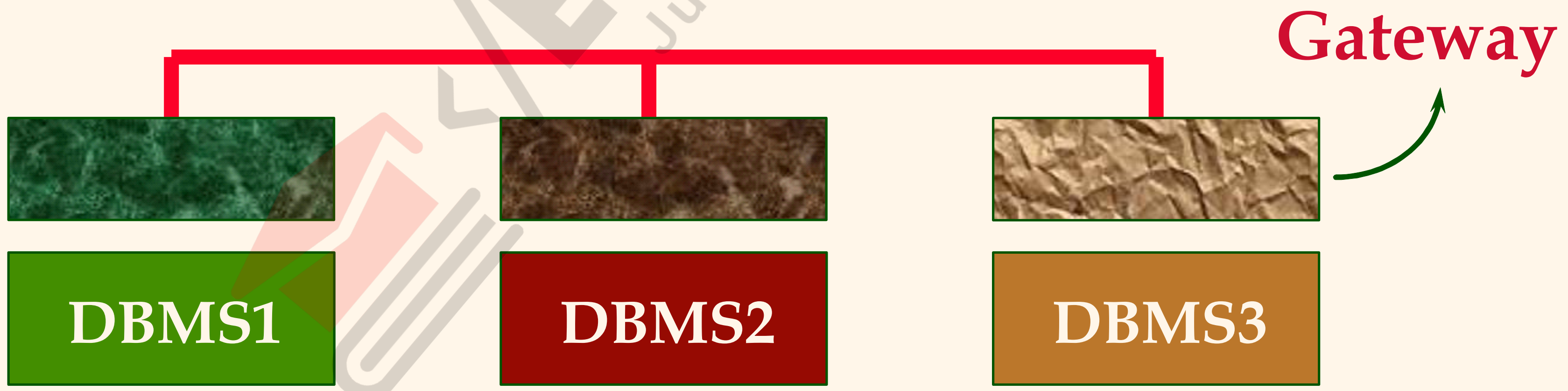
Recent Trends

- Users have to be aware of where data is located, i.e., Distributed Data Independence and Distributed Transaction Atomicity are not supported.
- These properties are hard to support efficiently.
- For globally distributed sites, these properties may not even be desirable due to administrative overheads of making location of data transparent.

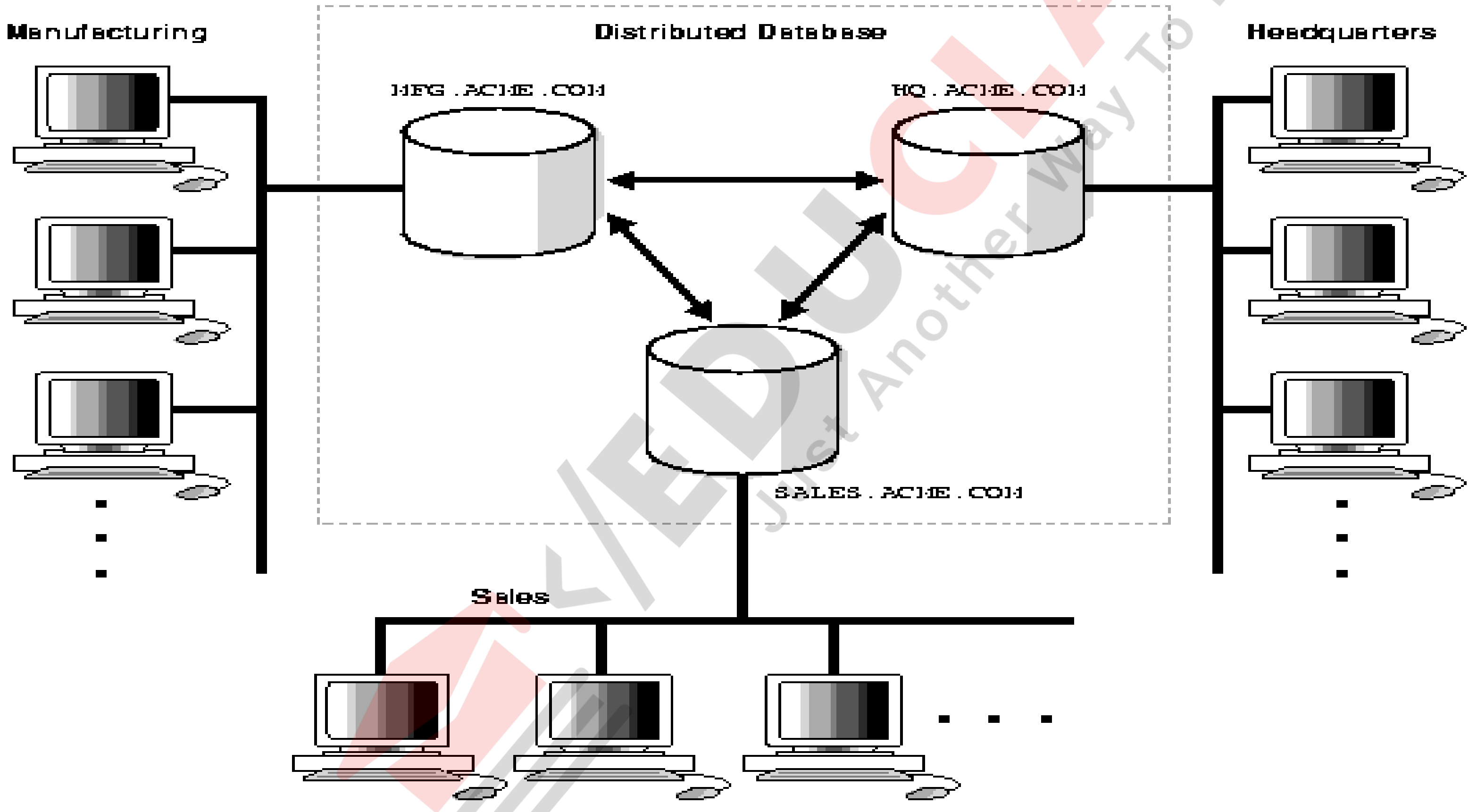
Dis

Types of Distributed Databases

- Homogeneous: Every site runs same type of DBMS.
- Heterogeneous: Different sites run different DBMSs (different RDBMSs or even non-relational DBMSs).



Homogeneous Distributed Database Systems



Cont…

- A homogenous distributed database system is a network of two or more Oracle databases that reside on one or more machines.
- A distributed system that connects three databases: hq, mfg, and sales. An application can simultaneously access or modify the data in several databases in a single distributed environment. For example, a single query from a Manufacturing client on local database mfg can retrieve joined data from the products table on the local database and the dept table on the remote hq database.
- For a client application, the location and platform of the databases are transparent. You can also create **synonyms** for remote objects in the distributed system so that users can access them with the same syntax as local objects. For example, if you are connected to database mfg but want to access data on database hq, creating a synonym on mfg for the remote dept table enables you to issue this query:
- SELECT * FROM dept;
- In this way, a distributed system gives the appearance of native data access. Users on mfg do not have to know that the data they access resides on remote databases.

Heterogenous Distributed Database System

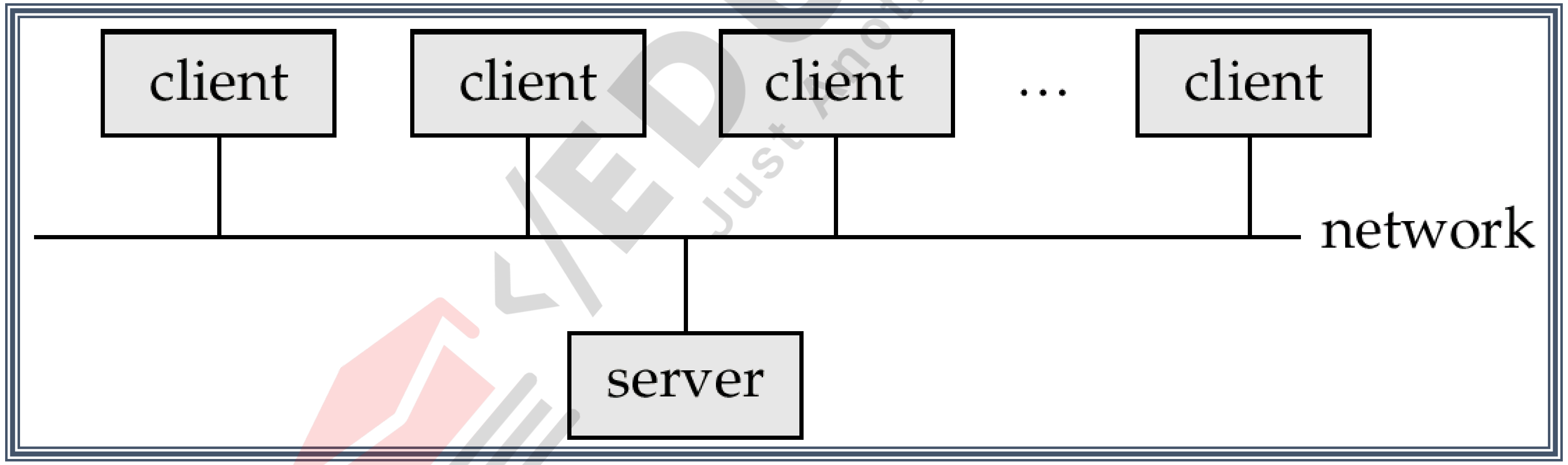
- In a heterogeneous distributed database system, at least one of the databases is a non-Oracle system. To the application, the heterogeneous distributed database system appears as a single, local, Oracle database. The local Oracle database server hides the distribution and heterogeneity of the data.
- The Oracle database server accesses the non-Oracle system using Oracle Heterogeneous Services in conjunction with an **agent**. If you access the non-Oracle data store using an Oracle Transparent Gateway, then the agent is a system-specific application. For example, if you include a Sybase database in an Oracle distributed system, then you need to obtain a Sybase-specific transparent gateway so that the Oracle databases in the system can communicate with it.
- Alternatively, you can use **generic connectivity** to access non-Oracle data stores so long as the non-Oracle system supports the ODBC or OLE DB protocols.

Distributed database Architechture

- 1. Client Server
- 2. Collaborating Server
- 3. Middleware

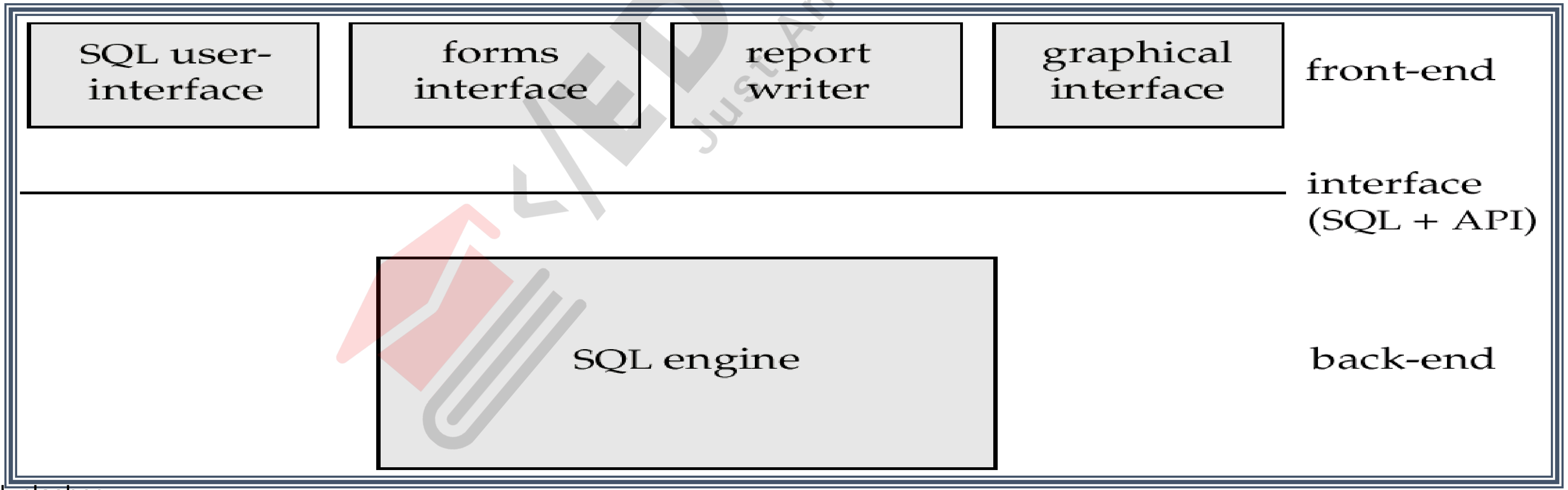
Client Server Systems

- Server systems satisfy requests generated at *m* client systems, whose general structure is shown below:



Client Server Systems

- Database functionality can be divided into:
  - **Back-end**: manages access structures, query evaluation and optimization, concurrency control and recovery.
  - **Front-end**: consists of tools such as *forms*, *report-writers*, and graphical user interface facilities.
- The interface between the front-end and the back-end is through SQL or through an application program interface.



Collaborating Server Systems

- The client server architecture does not allow a single query to span multiple servers because the client process would have to be capable of breaking such a query into appropriate sub queries to be executed at different sites and then piecing together the answers to the sub queries.
- The client process would therefore be quite complex.
- The client process capabilities would begin to overlap with the server due to which distinguishing between clients and servers becomes harder.

Collaborating Server Systems

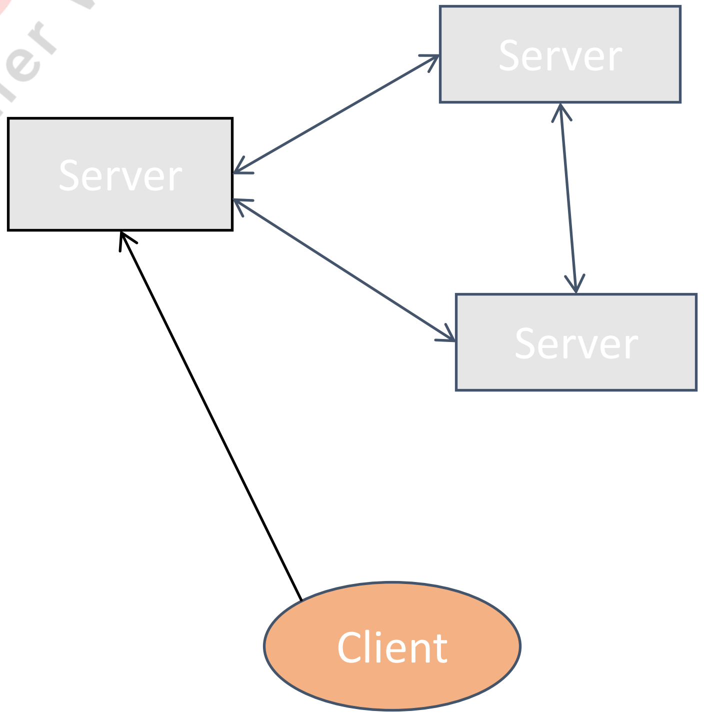
- A query can span multiple sites or servers.
- We can have database servers, each capable of running transactions

  against local data, which

  cooperatively execute

  transactions spanning

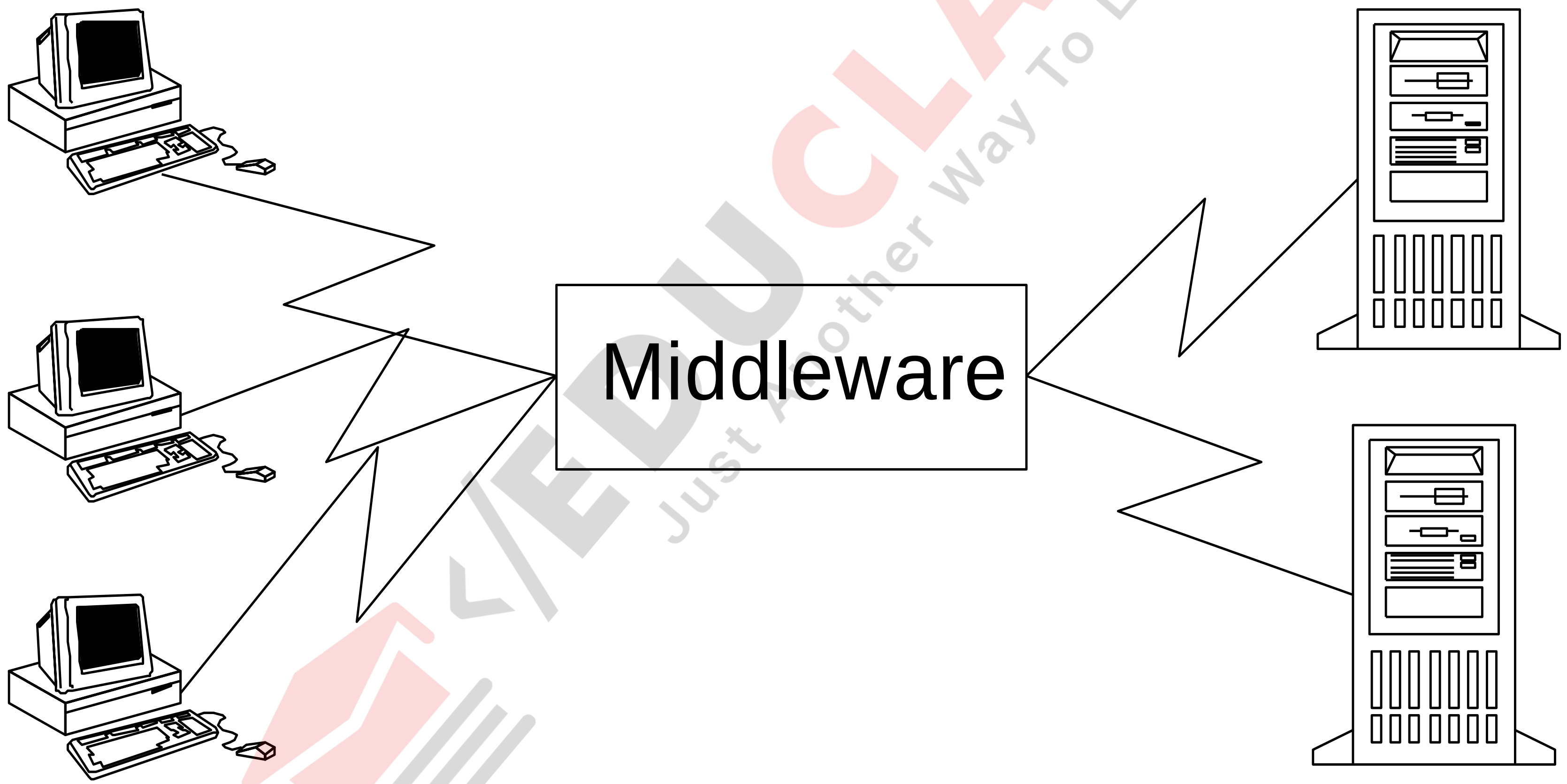
  multiple servers.



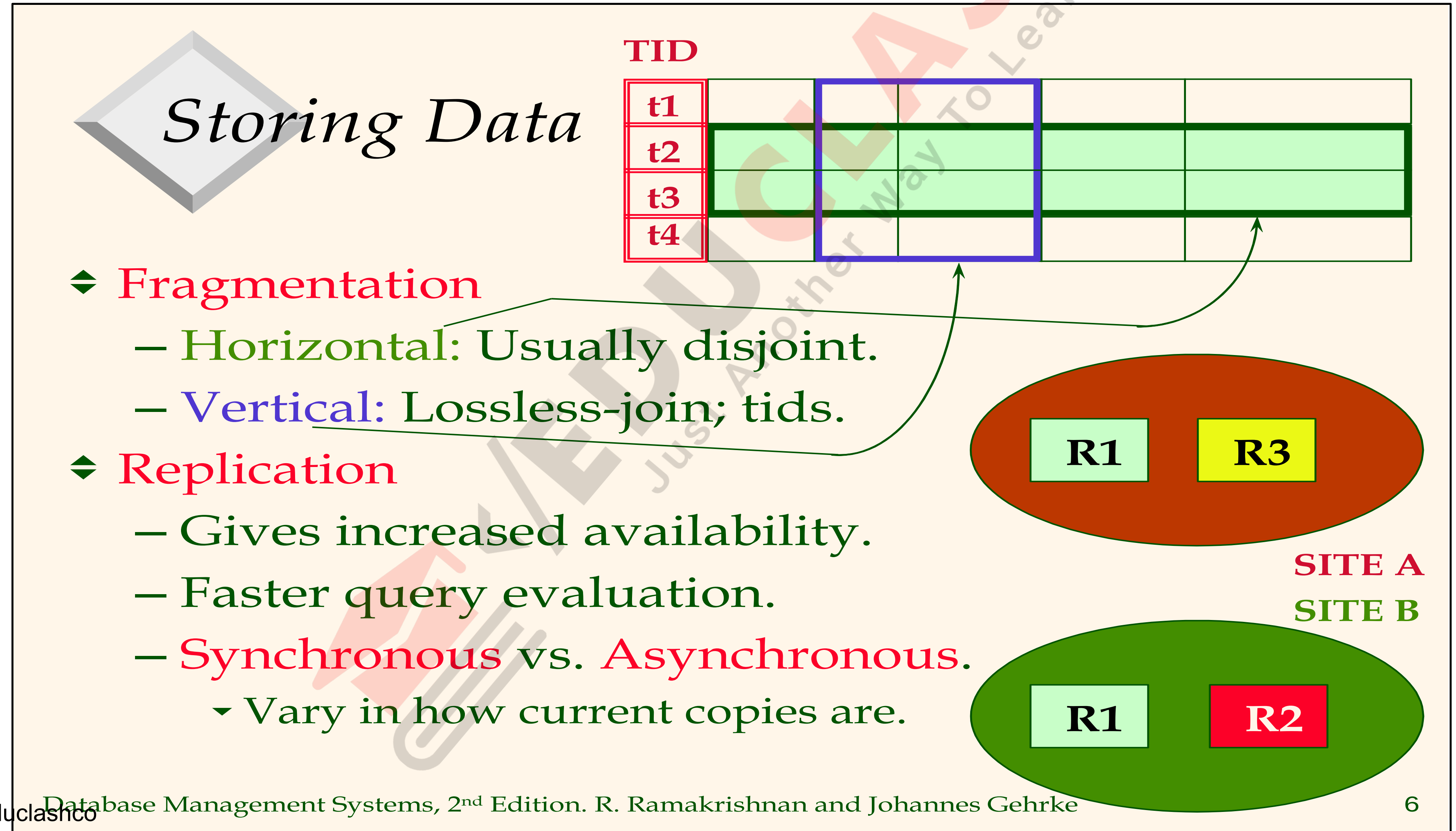
Middleware Systems

- A software component that performs process management.
- Allow clients and servers to exist on different platforms.
- One database server capable of managing queries and transactions spanning multiple servers.
- The remaining servers need to handle only local queries and transactions.
- Allows servers to efficiently process messages from a large number of clients.
- Often located on a dedicated computer.

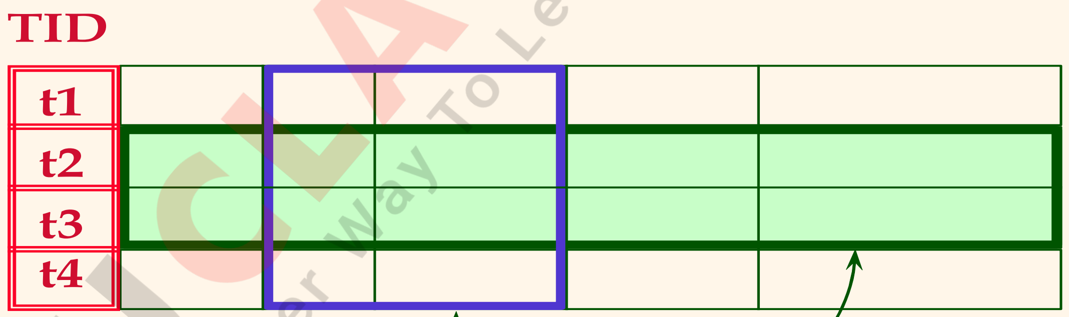
Client-Server Computing with Middleware



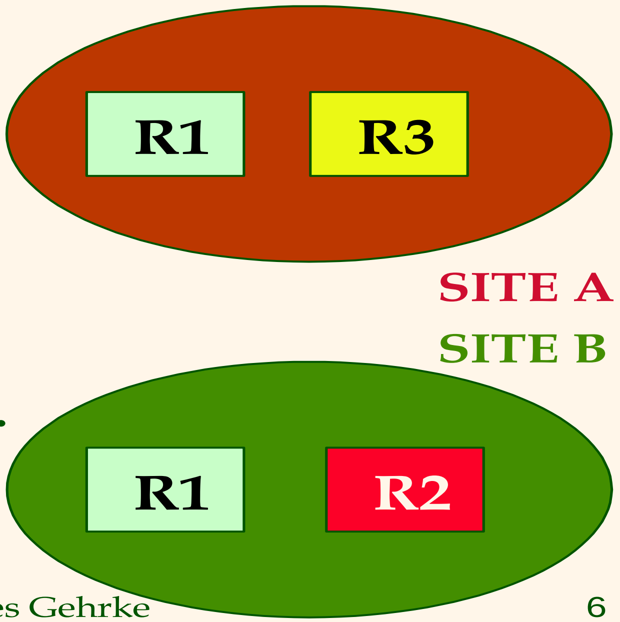
Distributed database



Storing Data



- Fragmentation
  - Horizontal: Usually disjoint.
  - Vertical: Lossless-join; tids.
- Replication
  - Gives increased availability.
  - Faster query evaluation.
  - Synchronous vs. Asynchronous.
    - Vary in how current copies are.



Fragmentation

- Horizontal – "Row-wise"
  - E.g. rows of the table make up one fragment
- Vertical –
  - E.g. columns of the table make up one fragment

ID	#Particles	Energy	Event#	Run#	Date	Time
…	…	…	…	…	…	…
10001	3	121.5	111	13120	3/1406	13:30:55.0001
10002	3	202.2	112	13120	3/1406	13:30:55.0001
10003	4	99.3	113	13120	3/1406	13:30:55.0001
10004	5	231.9	120	13120	3/1406	13:30:55.0001
10005	6	287.1	125	13120	3/1406	13:30:55.0001
10006	6	107.7	126	13120	3/1406	13:30:55.0001
10007	6	98.9	127	13120	3/1406	13:30:55.0001
10008	9	100.1	128	13120	3/1406	13:30:55.0001
…	…	…	…	…	…	…

Cont..

- When a relation is fragmentated, we must be able to recover the original relation from the fragments:
- Horizontal Fragmentation: The union of the horizontal fragments must be equal to the original relation. Fragments are usually also required to be disjoint.
- Vertical Fragmentation: The collection of vertical fragments should be a lossless-join decomposition.

Replication

- It means that we store several copies of a relation or relation fragment.
- Make synchronised or asynchronised copies of data at servers
  - Synchronised: data are always current, updates are constantly shipped between replicas
  - Asynchronised: good for read-only data

- The motivation for replication is two fold:

1. Increased Availability of Data
2. Faster Query Evaluation

Structured Data type

- A structured data type is a user defined data type with elements that are not atomic rather they are divisible and can be used either separately or as a single unit as per requirements.
- It is a form of user defined object that contains a sequence of attributes, each of which has a data type.
- An attribute is a property that helps to describe an instance of a particular type.
- for example if we want to define a structured type called address to store addresses, in which city might be one of the attributes of this structured type.

Cont..

- A structured data type can be used as the type for a column in a regular table, the type for an entire table or as an attribute of another structured type. When used as the type for a table, the table is known as typed table.
- **CREATE TYPE** statement is used to create a structured data type and DROP statement is used to delete the structured data type.
- Consider 'Emp-Dept' schema discussed previously. In this schema, table 'Emp ' is created with four column namely EmpNo, it is system generated identity column, Name contains name of the employee, Address which is used to hold the address of employee, it is a structured type column of type 'ADDRESS-T' and ProjNo which is a list that stores project number of project taken by employee and eImage that will contain images of an employee.

Cont..

- **CREATE TYPE ADDRESS-T as Row (street varchar (12), city varchar (12), state varchar (12),postal code varchar (12))**
  Now, the ‘Emp’ table is created having ‘ADDRESS-T’ data type of Address field and jpeg-image as data type of eimage as shown below:

- **CREATE TABLE EMP (Empno integer system generated, Name varchar (12),Address ADDRESS-T,Proj-no set of (varchar 12),Eimage jpeg-image);**

Structured types in SQL 1999

- Structured types available in SQL 1999 are
  ROW (f1t1,f2t2…….fntn)–
  1. It represents a row, or a tuple of fields f1,f2,…..fn of types t1,t2,..tn respectively. 'ROW' data type specifies every table as a collection of rows or every table as set of rows or multi-set of rows for example, the 'address-t' is declared as of ROW data type as shown below which contains area, city and state as its components :-
  CREATE TYPE address-t
  AS ROW (area: varchar (20),city: varchar (20), state : varchar (20))

Cont…

- 2. ARRAY [i] : It represents an array of ‘i’ items of ‘base’ type for example, the ‘objects’ field of CLIP table used an array of 10 objects, each of which is of varchar (20) type.
- A multidimensional array can not be created in SQL 99.
- An array can be used as component in ROW type as shown but not in array type :-
- **ROW (pno : integer, object : varchar (20) ARRAY [10])**

- 3. list of (base) : It represents a list of all items of ‘base’ type,
- for example,
  PROJECT (Projno : integer, Pname: varchar (25), Emp no: list of (integer))
- 4. set of (base) : It represents a set of ‘base’ type items. A set does not contain duplicate elements unlike lists otherwise it is used in the same manner as list.

- 5. bag of (base) : It represents a bag or a multi-set of base type items. Collection type or build data types are types using list of, set of, bag of and ARRAY.
- But SQL does not provide any efficient method for manipulation of these collection type objects.
- So now we will discuss how these data types can be manipulated and also discuss the operations which can be applied on these data types.

Company database

- all the tables regarding 'company' database is created by using **DDL statements** shown in below :
  CREATE TABLE PROJECT
  (Projno integer, Pname varchar (20),
  Location REF (address-t) SCOPE LOCS,
  Empno set of (integer));
  CREATE TABLE LOCS OF ADDRESS-T REF is locid system generated.
  CREATE TABLE DEPT
  (Deptno integer, Dname varchar (20),
  Dlocation REF (address-t) SCOPE LOCS,
  Projno set of (integer));
  CREATE TABLE CLIPS
  (clipno integer, Cname varchar (20),
  Objects varchar (20) ARRAY [20],
  Budget float,
  Projno integer Ctime time);

Operation on Structured Data

- **1. Operations on Arrays**
  Array is used in the same manner as in traditional RDBMS. 'Array index' method is used to return the number of elements in the array for example. Suppose we want to find those projects whose clips contain more than 10 items or objects then following query can be used :
  SELECT P.Pname, P.Projno
  FROM project P, Clip C
  WHERE CARDINALITY (C.Objects)>10 AND C.Projno = P.Projno
  The above query select project name and projectno from "PROJECT" whose clips contain more then 10 items which can calculated by using CARDINALITY operation.

Cont…

- **2. Operations on Rows**
  Row type is a collection of fields values whose each fields can be accessed by the same traditional notation.
- for example, address-t.city specify the attribute ‘city’ of the type address-t. When operation is applied on collection of rows then result obtained is also a collection of values.
- If a column or field whose type is ROW (f1t1, f2t2,…….fntn) and c1 fk gives us a list of values whose type is tk. If c1 is a set of rows or a bag of rows then c1 fk give us a set of values of type tk.
  Consider ‘Emp-Dept’ schema in which we have to find the names of those employees who resides in ‘Malviya Nagar’ of ‘New Delhi’.
  SELECT E Empno,E.Name
  FROM Emp E
  WHERE E.Address.area =’Malviya Nagar’ AND E.Address.city=’New Delhi’;

- **3. Operations on Sets and Multi-sets**
  Set and multisets are used in the traditional manner by using =,<,>,>,< comparison operators.
- An item of a set can be compared by other items using E (belongs to) relation.
- Two set objects can create a new object using U, (Union Operation).
- They can also create a new object by subtracting a set of elements from other set by using '-' (set difference operator).
- Multi-set also uses the same operations as used by the sets but the operations are applied on the number of copies of element into account.

- **4. Operations on Lists**
  List includes operations like ‘append’, ‘concatenate’, ‘head’, ‘tail’ etc. to manipulate the items of list for example, ‘concatenate’ or ‘append’ appends one list to another, ‘head’ returns the first element of list, ‘tail’ returns the list after removing the first element.

Objects, OIDS, and Reference Types

- In object database systems, data objects can be given an object identifier, which is unique in the database across time.
- All tuples stored in any table are objects and automatically assigned unique oids.
- An object oid can be used to refer to it from elsewhere in the data.
- REF types have values that are unique identifiers or oids.

Objects, OIDS, and Reference Types

- E.g a column theater of type REF(theater_t). The scope clause specifies that items in this column are references to rows in the theater table.
- Dereferencing Reference types:
- To access the value associated with object oid we use DEREF() method which is provided along with the REF type.
- E.g. from Nowshowing table, one can access the name field of the referenced theater_t object

  Nowshowing.deref(theater).name

Object Relational DBMS	Object Oriented DBMS
The features of these DBMS include: • Support for complex data types • Powerful query languages support through SQL • Good protection of data against programming errors	The features of these DBMS include: • Supports complex data types, • Very high integration of database with the programming language, • Very good performance • But not as powerful at querying as Relational
One of the major assets here is SQL. Although, SQL is not as powerful as a Programming Language, but it is none-the less essentially a fourth generation language, thus, it provides excellent protection of data from the Programming errors	It is based on object oriented programming languages, thus, are very strong in programming, however, any error of a data type made by a programmer may effect many users
The relational model has a very rich foundation for query optimisation, which helps in reducing the time taken to execute a query.	These databases are still evolving in this direction. They have reasonable systems in place.
These databases make the querying as simple as in relational even, for complex data types and multimedia data	The querying is possible but somewhat difficult to get.
Although the strength of these DBMS is SQL, it is also one of the major weaknesses from the performance point of view in memory application	Some applications that are primarily run in the RAM and require a large number of database accesses with high performance may find such DBMS more suitable. This is because of rich programming interface provided by such DBMS. However, such applications may not support very strong query capabilities. A typical example of one such