Distributed Databases

Chapter 1: Introduction

Johann Gamper

- Syllabus
- Data Independence and Distributed Data Processing
- Definition of Distributed databases
- Promises of Distributed Databases
- Technical Problems to be Studied
- Conclusion

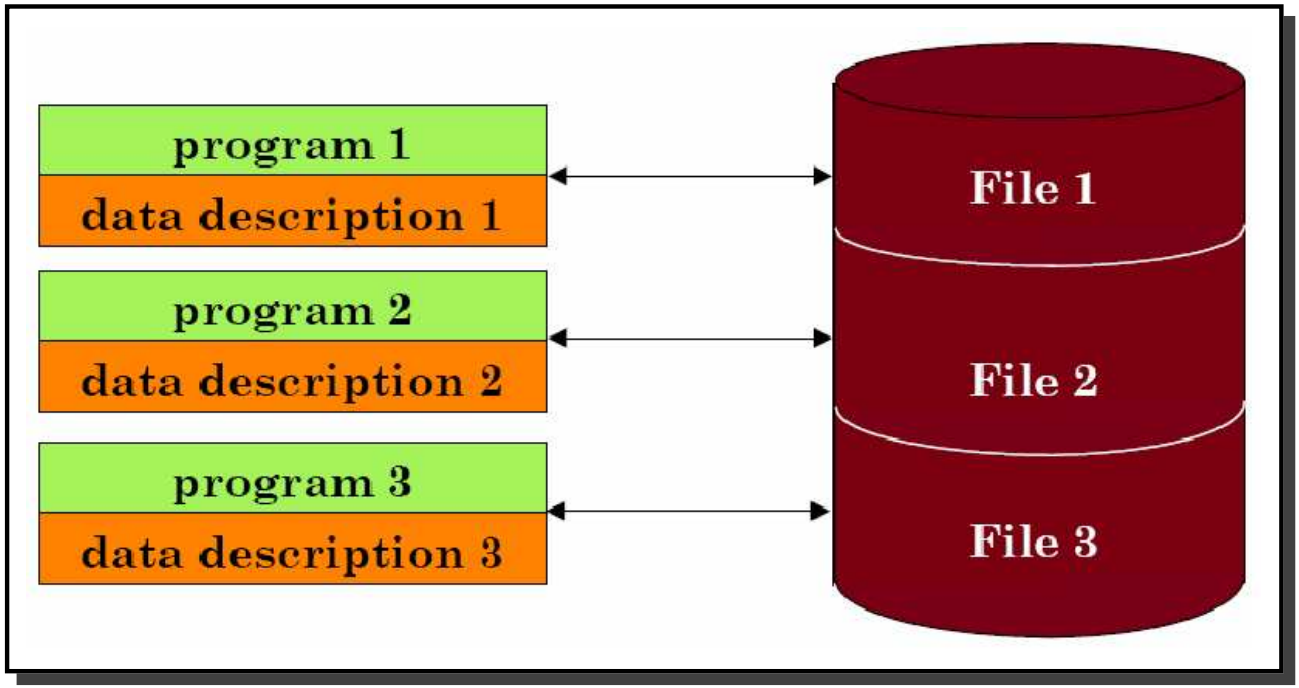
Acknowledgements: I am indebted to Arturas Mazeika for providing me his slides of this course.

Syllabus

- Introduction
- Distributed DBMS Architecture
- Distributed Database Design
- Query Processing
- Transaction Management
- Distributed Concurrency Control
- Distributed DBMS Reliability
- Parallel Database Systems

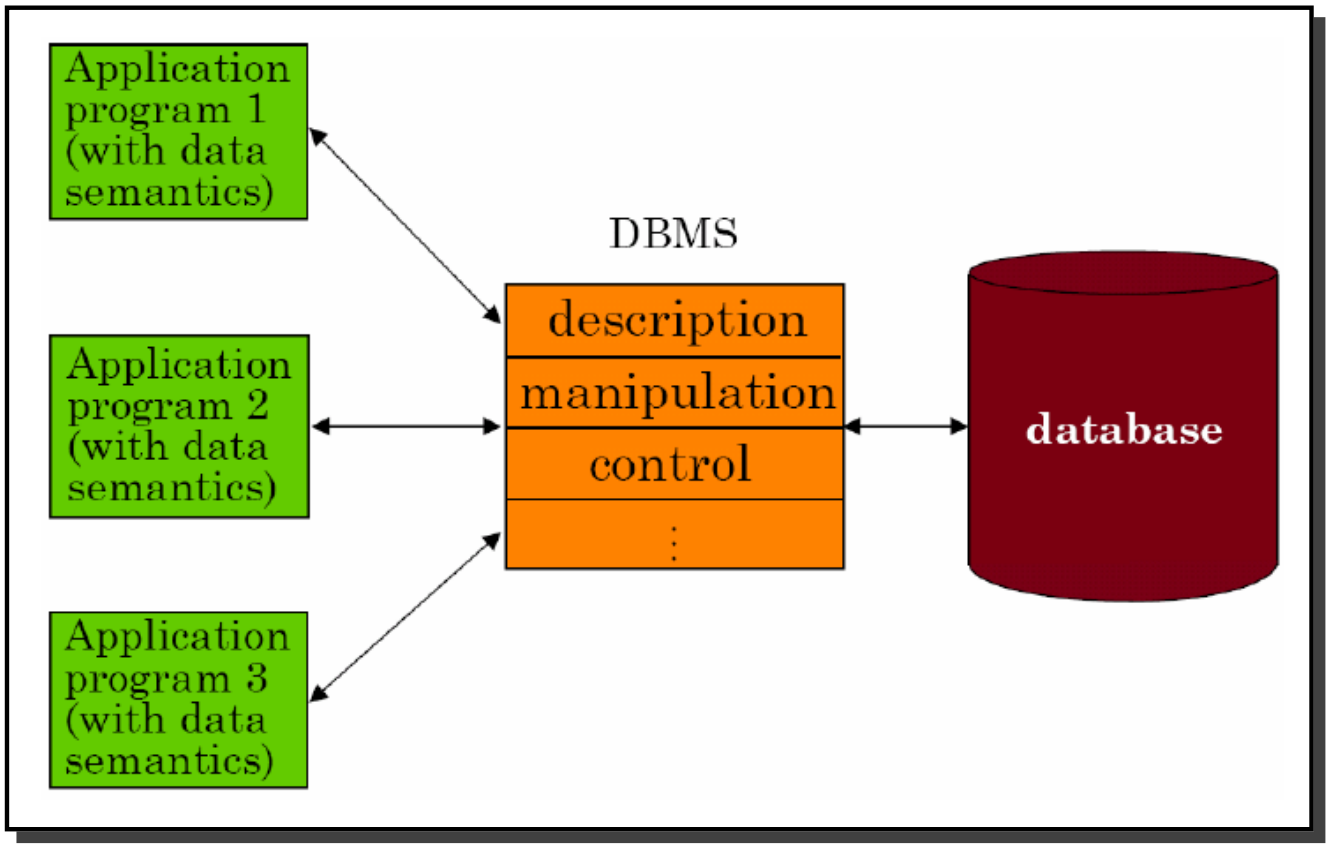
Data Independence

- In the old days, programs stored data in regular files
- Each program has to maintain its own data
  - huge overhead
  - error-prone



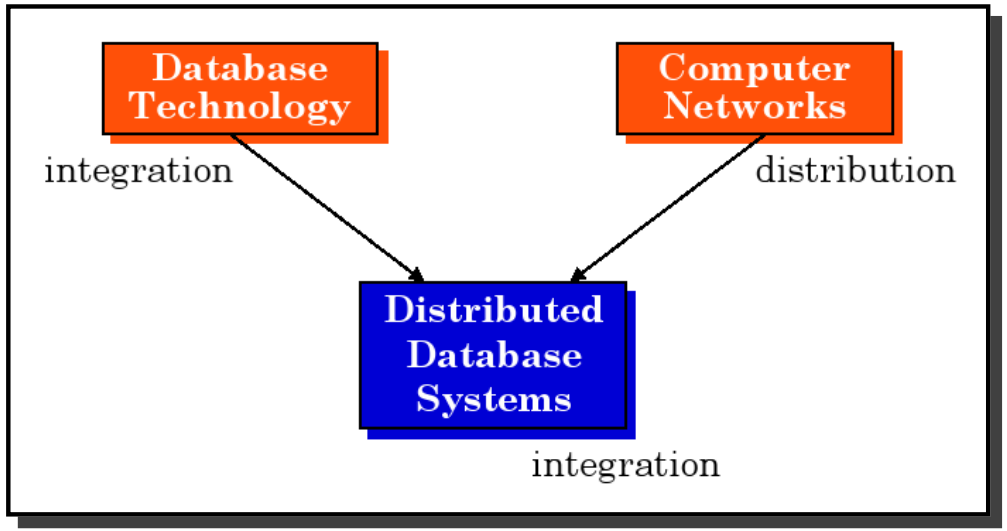
Data Independence . . .

- The development of **DBMS** helped to fully achieve data independence (transparency)
- Provide **centralized** and controlled data maintenance and access
- Application is immune to physical and logical file organization



Data Independence . . .

- Distributed database system is the union of what appear to be two diametrically opposed approaches to data processing: **database systems** and **computer network**
  - Computer networks promote a mode of work that goes against centralization
- Key issues to understand this combination
  - The most important objective of DB technology is **integration** not centralization
  - Integration is possible without centralization, i.e., integration of databases and networking does not mean centralization (in fact quite opposite)
- Goal of distributed database systems: achieve data integration **and** data distribution transparency



Distributed Computing/Data Processing

- A **distributed computing system** is a collection of autonomous processing elements that are interconnected by a computer network. The elements cooperate in order to perform the assigned task.
- The term “distributed” is very broadly used. The exact meaning of the word depends on the context.
- Synonymous terms:
  - distributed function
  - distributed data processing
  - multiprocessors/multicomputers
  - satellite processing
  - back-end processing
  - dedicated/special purpose computers
  - timeshared systems
  - functionally modular systems

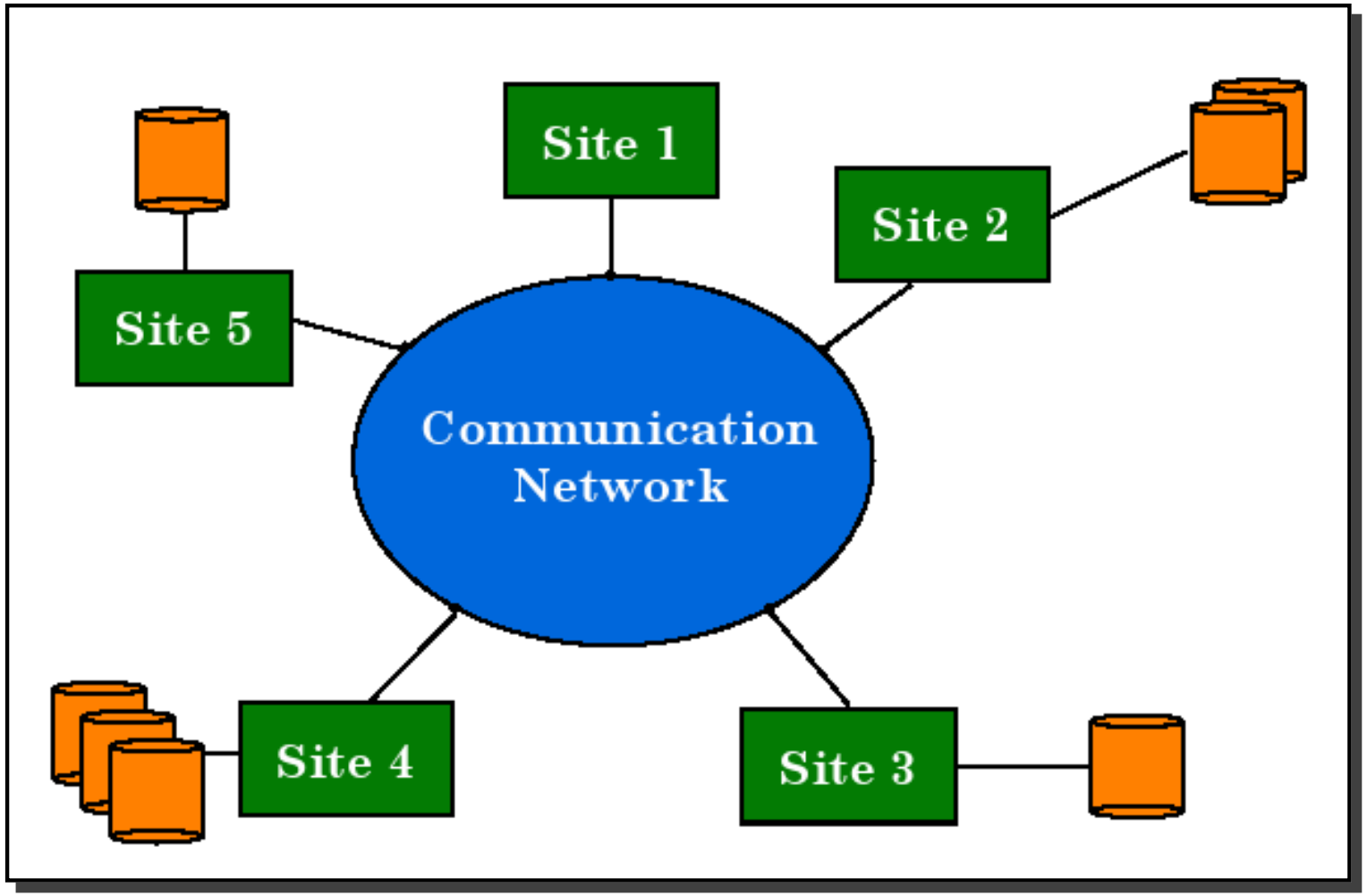
Distributed Computing/Data Processing . . .

- What can be distributed?
  - Processing logic
  - Functions
  - Data
  - Control
- Classification of distributed systems with respect to various criteria
  - Degree of coupling, i.e., how closely the processing elements are connected
    - e.g., measured as ratio of amount of data exchanged to amount of local processing
    - weak coupling, strong coupling
  - Interconnection structure
    - point-to-point connection between processing elements
    - common interconnection channel
  - Synchronization
    - synchronous
    - asynchronous

Definition of DDB and DDBMS

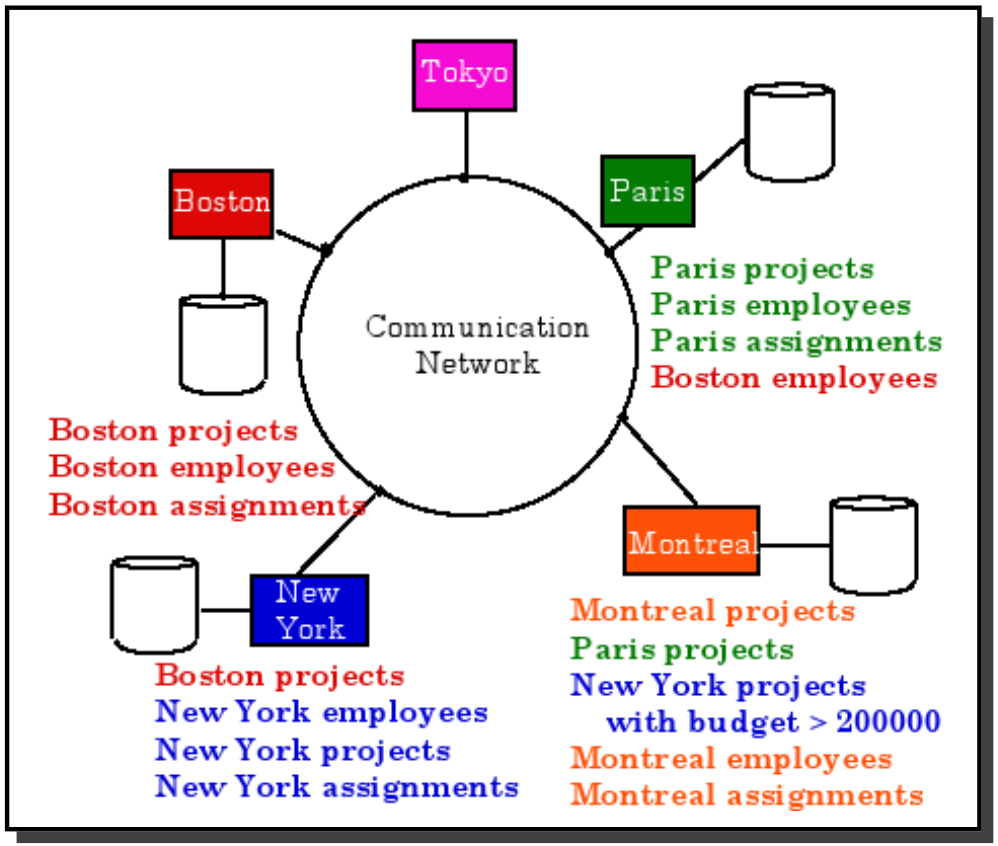
- A **distributed database** (DDB) is a collection of multiple, logically interrelated databases distributed over a computer network
- A **distributed database management system** (DDBMS) is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users
- The terms DDBMS and DDBS are often used interchangeably
- Implicit assumptions
  - Data stored at a number of sites each site logically consists of a single processor
  - Processors at different sites are interconnected by a computer network (we do not consider multiprocessors in DDBMS, cf. parallel systems)
  - DDBS is a database, not a collection of files (cf. relational data model). Placement and query of data is impacted by the access patterns of the user
  - DDBMS is a collections of DBMSs (not a remote file system)

Definition of DDB and DDBMS . . .



Definition of DDB and DDBMS ...

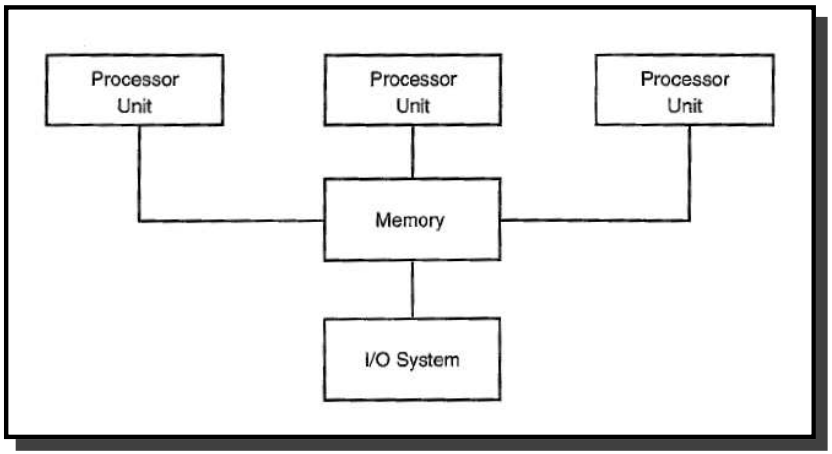
- **Example:** Database consists of 3 relations `employees`, `projects`, and `assignment` which are partitioned and stored at different sites (fragmentation).



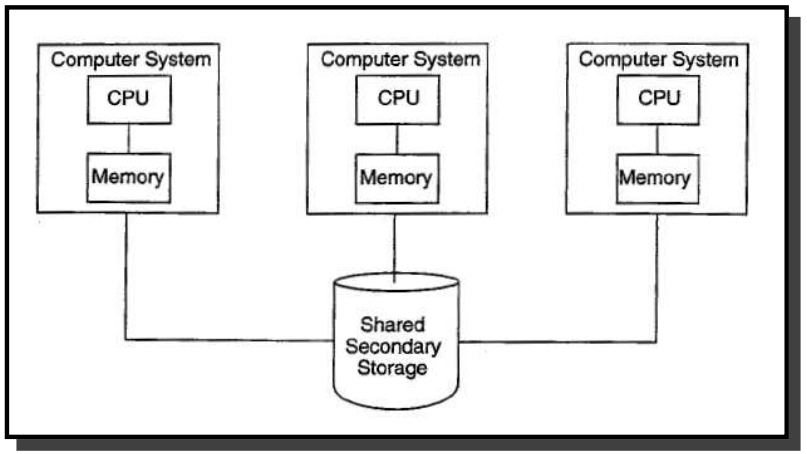
- What are the problems with queries, transactions, concurrency, and reliability?

What is not a DDBS?

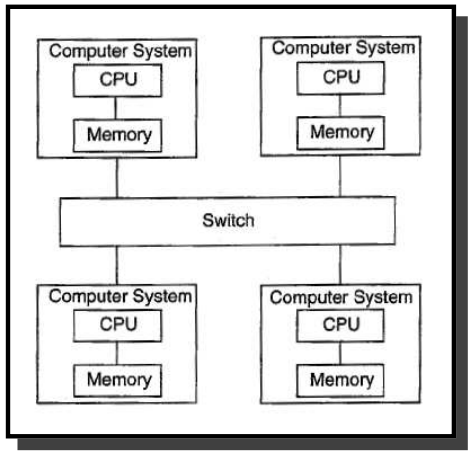
- The following systems are parallel database systems and are quite different from (though related to) distributed DB systems



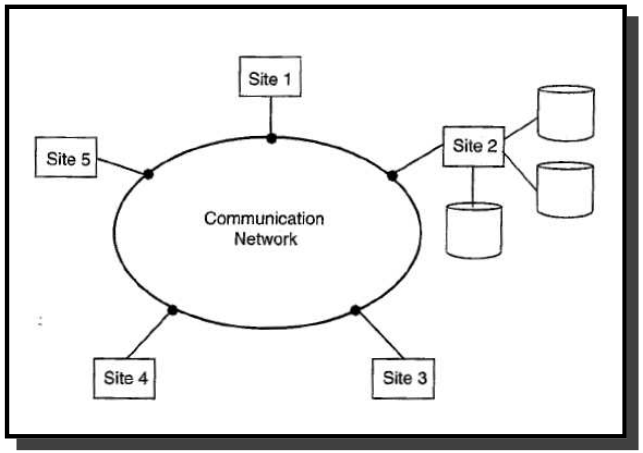
Shared Memory



Shared Disk



Shared Nothing



Central Databases

Applications

- Manufacturing, especially multi-plant manufacturing
- Military command and control
- Airlines
- Hotel chains
- Any organization which has a decentralized organization structure

Promises of DDBSs

Distributed Database Systems deliver the following advantages:

- Higher reliability
- Improved performance
- Easier system expansion
- Transparency of distributed and replicated data

Promises of DDBSs . . .

Higher reliability

- Replication of components
- No single points of failure
- e.g., a broken communication link or processing element does not bring down the entire system
- Distributed transaction processing guarantees the consistency of the database and concurrency

Promises of DDBSs . . .

Improved performance

- Proximity of data to its points of use
  - Reduces remote access delays
  - Requires some support for fragmentation and replication
- Parallelism in execution
  - Inter-query parallelism
  - Intra-query parallelism
- Update and read-only queries influence the design of DDBSs substantially
  - If mostly read-only access is required, as much as possible of the data should be replicated
  - Writing becomes more complicated with replicated data

Promises of DDBSs . . .

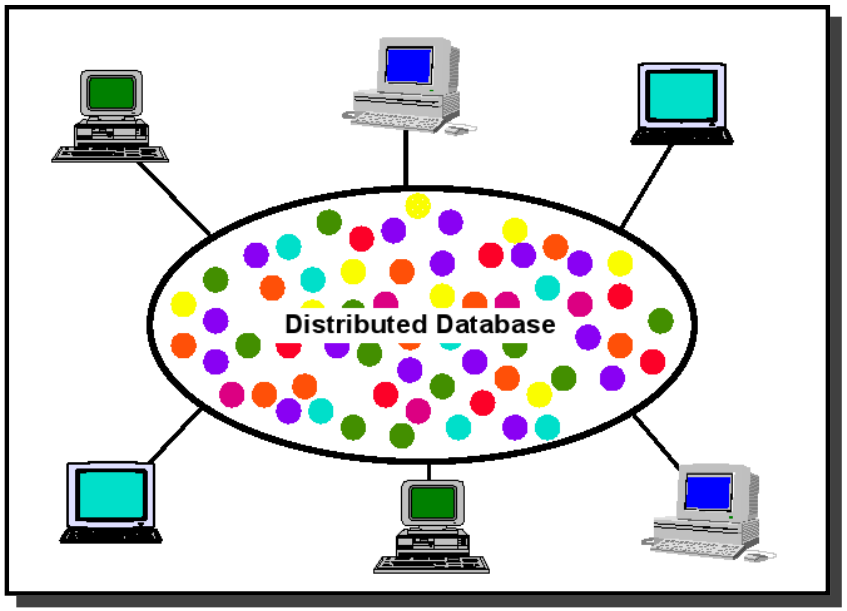
Easier system expansion

- Issue is database scaling
- Emergence of microprocessor and workstation technologies
  - Network of workstations much cheaper than a single mainframe computer
- Data communication cost versus telecommunication cost
- Increasing database size

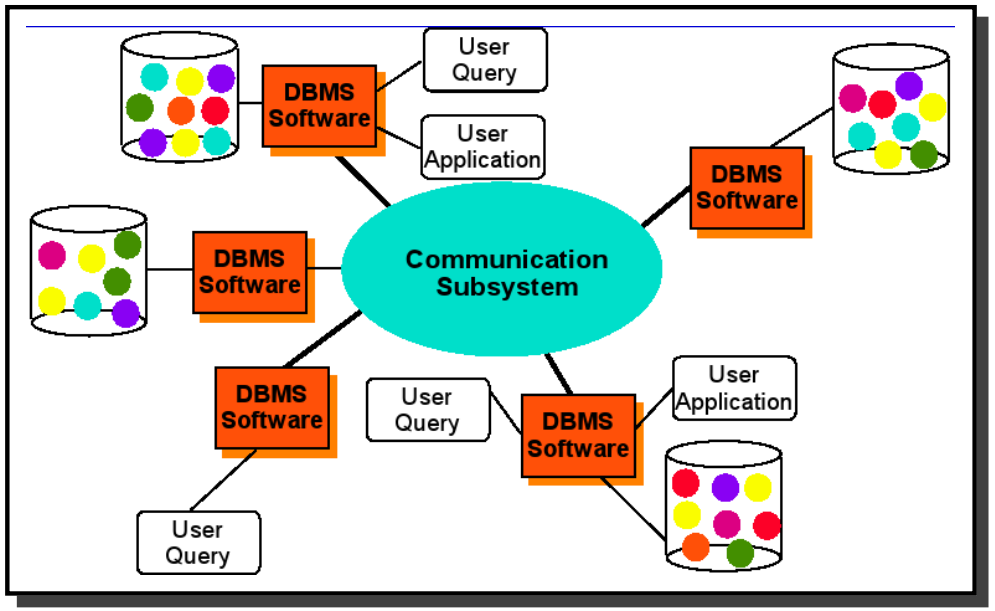
Promises of DDBSs . . .

Transparency

- Refers to the separation of the higher-level semantics of the system from the lower-level implementation issues
- A transparent system “hides” the implementation details from the users.
- A fully transparent DBMS provides high-level support for the development of complex applications.



(a) User wants to see one database



(b) Programmer sees many databases

Promises of DDBSs . . .

Various forms of **transparency** can be distingushed for DDBMSs:

- Network transparency (also called distribution transparency)
  - Location transparency
  - Naming transparency
- Replication transparency
- Fragmentation transparency
- Transaction transparency
  - Concurrency transparency
  - Failure transparency
- Performance transparency

Promises of DDBSs . . .

- **Network/Distribution transparency** allows a user to perceive a DDBS as a single, logical entity
- The user is protected from the operational details of the network (or even does not know about the existence of the network)
- The user does not need to know the location of data items and a command used to perform a task is independent from the location of the data and the site the task is performed (**location transparency**)
- A unique name is provided for each object in the database (**naming transparency**)
  - In absence of this, users are required to embed the location name as part of an identifier

Promises of DDBSs . . .

Different ways to ensure naming transparency:

- Solution 1: Create a central name server; however, this results in
  - loss of some local autonomy
  - central site may become a bottleneck
  - low availability (if the central site fails remaining sites cannot create new objects)
- Solution 2: Prefix object with identifier of site that created it
  - e.g., branch created at site S1 might be named S1.BRANCH
  - Also need to identify each fragment and its copies
  - e.g., copy 2 of fragment 3 of Branch created at site S1 might be referred to as S1.BRANCH.F3.C2
- An approach that resolves these problems uses aliases for each database object
  - Thus, S1.BRANCH.F3.C2 might be known as local branch by user at site S1
  - DDBMS has task of mapping an alias to appropriate database object

Promises of DDBSs . . .

- **Replication transparency** ensures that the user is not involved in the managment of copies of some data
- The user should even not be aware about the existence of replicas, rather should work as if there exists a single copy of the data
- Replication of data is needed for various reasons
  - e.g., increased efficiency for read-only data access

Promises of DDBSs . . .

- **Fragmentation transparency** ensures that the user is not aware of and is not involved in the fragmentation of the data
- The user is not involved in finding query processing strategies over fragments or formulating queries over fragments
  - The evaluation of a query that is specified over an entire relation but now has to be performed on top of the fragments requires an appropriate query evaluation strategy
- Fragmentation is commonly done for reasons of performance, availability, and reliability
- Two fragmentation alternatives
  - Horizontal fragmentation: divide a relation into a subsets of tuples
  - Vertical fragmentation: divide a relation by columns

Promises of DDBSs . . .

- **Transaction transparency** ensures that all distributed transactions maintain integrity and consistency of the DDB and support concurrency
- Each distributed transaction is divided into a number of sub-transactions (a sub-transaction for each site that has relevant data) that concurrently access data at different locations
- DDBMS must ensure the indivisibility of both the global transaction and each of the sub-transactions
- Can be further divided into
  - Concurrency transparency
  - Failure transparency

Promises of DDBSs . . .

- **Concurrency transparency** guarantees that transactions must execute independently and are logically consistent, i.e., executing a set of transactions in parallel gives the same result as if the transactions were executed in some arbitrary serial order.

- Same fundamental principles as for centralized DBMS, but more complicated to realize:
  - DDBMS must ensure that global and local transactions do not interfere with each other
  - DDBMS must ensure consistency of all sub-transactions of global transaction

- Replication makes concurrency even more complicated
  - If a copy of a replicated data item is updated, update must be propagated to all copies
  - Option 1: Propagate changes as part of original transaction, making it an atomic operation; however, if one site holding a copy is not reachable, then the transaction is delayed until the site is reachable.
  - Option 2: Limit update propagation to only those sites currently available; remaining sites are updated when they become available again.
  - Option 3: Allow updates to copies to happen asynchronously, sometime after the original update; delay in regaining consistency may range from a few seconds to several hours

Promises of DDBSs . . .

- **Failure transparency**: DDBMS must ensure atomicity and durability of the global transaction, i.e., the sub-transactions of the global transaction either all commit or all abort.
- Thus, DDBMS must synchronize global transaction to ensure that all sub-transactions have completed successfully before recording a final COMMIT for the global transaction
- The solution should be robust in presence of site and network failures

Promises of DDBSs . . .

- **Performance transparency**: DDBMS must perform as if it were a centralized DBMS
  - DDBMS should not suffer any performance degradation due to the distributed architecture
  - DDBMS should determine most cost-effective strategy to execute a request
- Distributed Query Processor (DQP) maps data request into an ordered sequence of operations on local databases
- DQP must consider fragmentation, replication, and allocation schemas
- DQP has to decide:
  - which fragment to access
  - which copy of a fragment to use
  - which location to use
- DQP produces execution strategy optimized with respect to some cost function
- Typically, costs associated with a distributed request include: I/O cost, CPU cost, and communication cost

Complicating Factors

- Complexity
- Cost
- Security
- Integrity control more difficult
- Lack of standards
- Lack of experience
- Database design more complex

Technical Problems to be Studied . . .

- Distributed database design
  - How to fragment the data?
  - Partitioned data vs. replicated data?
- Distributed query processing
  - Design algorithms that analyze queries and convert them into a series of data manipulation operations
  - Distribution of data, communication costs, etc. has to be considered
  - Find optimal query plans
- Distributed directory management
- Distributed concurrency control
  - Synchronization of concurrent accesses such that the integrity of the DB is maintained
  - Integrity of multiple copies of (parts of) the DB have to be considered (mutual consistency)
- Distributed deadlock management
  - Deadlock management: prevention, avoidance, detection/recovery

Technical Problems to be Studied . . .

- Reliability
  - How to make the system resilient to failures
  - Atomicity and Durability
- Heterogeneous databases
  - If there is no homogeneity among the DBs at various sites either in terms of the way data is logically structured (data model) or in terms of the access mechanisms (language), it becomes necessary to provide translation mechanisms

Conclusion

- A distributed database (DDB) is a collection of multiple, logically interrelated databases distributed over a computer network
- Data stored at a number of sites, the sites are connected by a network. DDB supports the relational model. DDB is not a remote file system
- Transparent system ‘hides’ the implementation details from the users
  - Distribution transparency
  - Network transparency
  - Transaction transparency
  - Performance transparency
- Programming a distributed database involves:
  - Distributed database design
  - Distributed query processing
  - Distributed directory management
  - Distributed concurrency control
  - Distributed deadlock management
  - Reliability