# H12

MANCHESTER
1824
The University of Manchester

# DBMS Architectures

Fundamentals of Databases

Alvaro A A Fernandes, SCS, UoM

[COMP23111 2014-2015 Lecture 12 of 12 ]

## Acknowledgements

- These slides are adaptations (mostly minor, but some major) of material authored and made available to instructors by **A. Silberschatz, H. F. Korth, and S. Sudarshan** to accompany their textbook **Database System Concepts, 6th Edition, McGraw-Hill, 2006, 978-0-07-352332-3**.
- Copyright remains with them, whom I thank.
- Any errors are my responsibility.

# In Previous Lectures

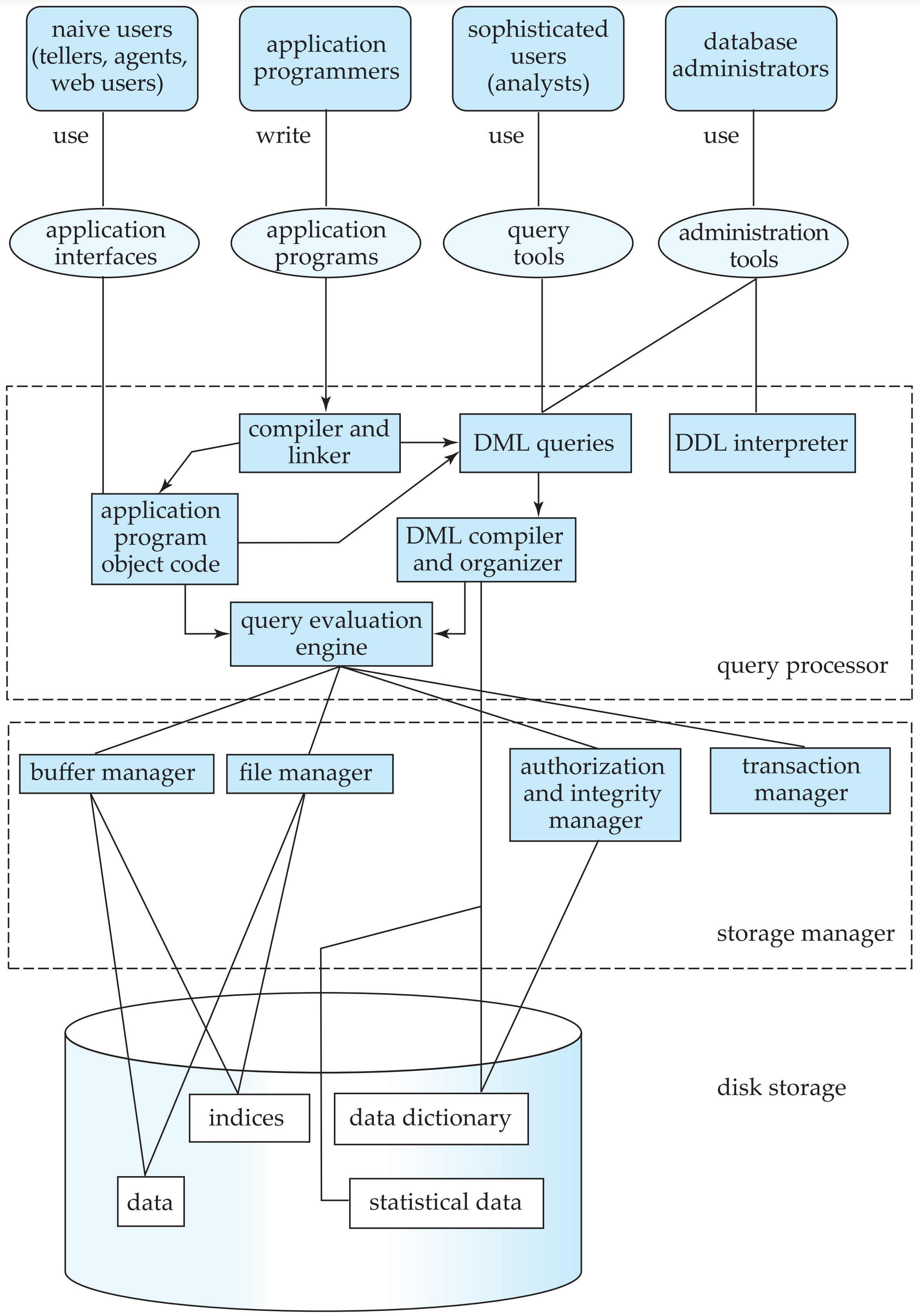
- We learned that data is an enterprise asset of such importance that specialized software systems, called DBMSs, are crucial to manage it well and that using files alone won't do.
- We learned the importance of separating concerns and of adopting different levels of abstraction in designing and implementing databases.
- We learned how data models lead to schemas and instances that enable a logical view of the data.
- We learned how to capture data requirements in the form of an (E)ER conceptual model.
- We learned how to derived from this conceptual model a logical one in the relational case, which is directly implementable in a DBMS.
- We learned how SQL has been extended to Turing-completeness and how views and trigger enrich our capability to capture and enforce domain semantics.
- We learned how DBMSs use the notion of transactions to manage concurrency and recovery.
- We learned how file organizations and indexed enable a logical-to-physical abstraction level at the storage layer too.

## In This Lecture

- What is the internal architecture of a classical DBMS like?
  - What are the main subsystems?
  - What are the main classes of users of DBMSs?
- What are the main kinds of database architecture in use today?
  - How are client-server designs realized in DBMS architecture?
  - What kinds of server are there?
  - What parallelization and distribution approaches are there for DBMSs?
  - How do DBMSs scale?

What is the internal architecture of a classical DBMS like? What are the main subsystems? What are the main classes of users?

# DBMS Architecture: Components/Interactions

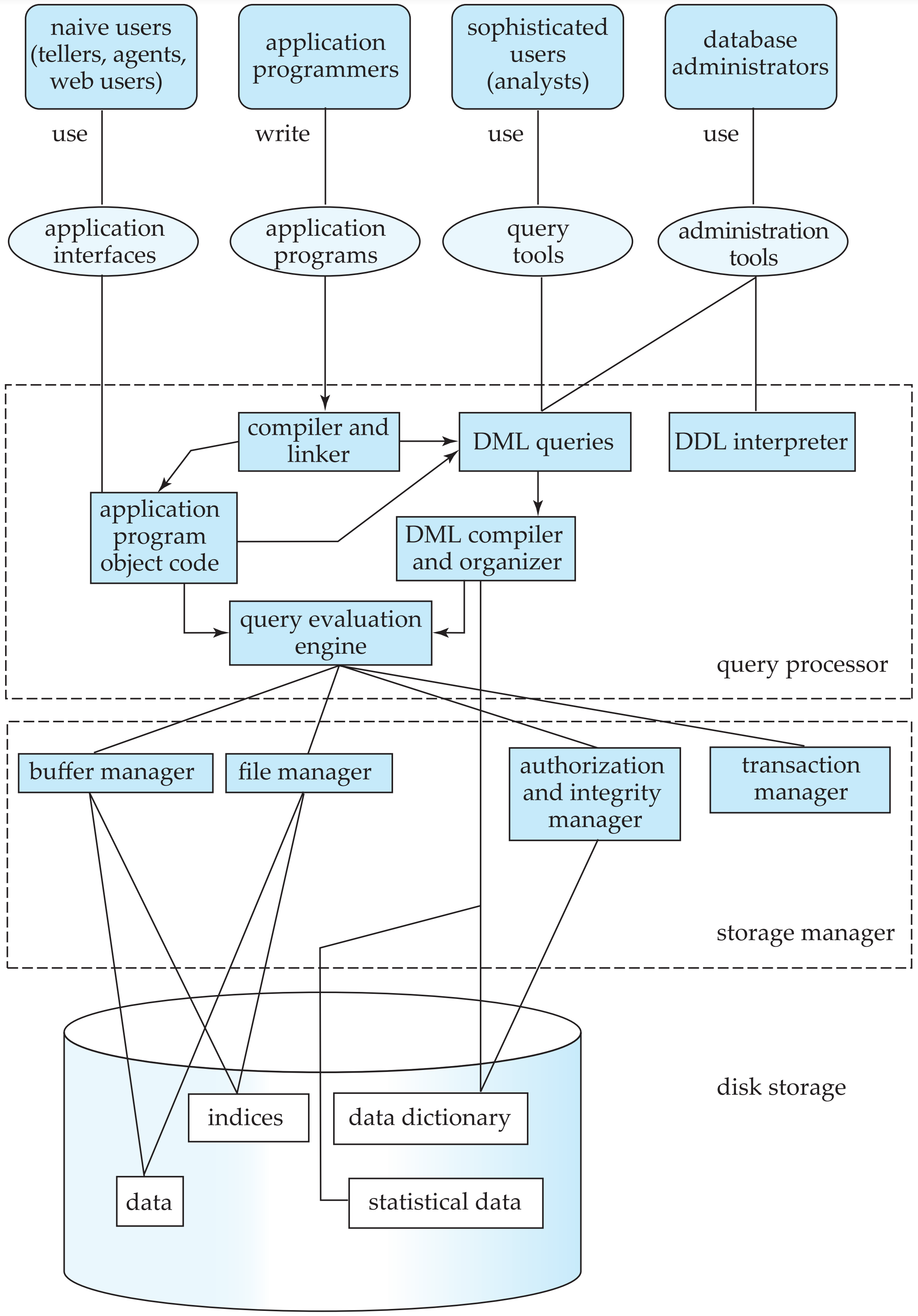


# Main DBMS Subsystems :: Storage Management

- Storage manager is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.

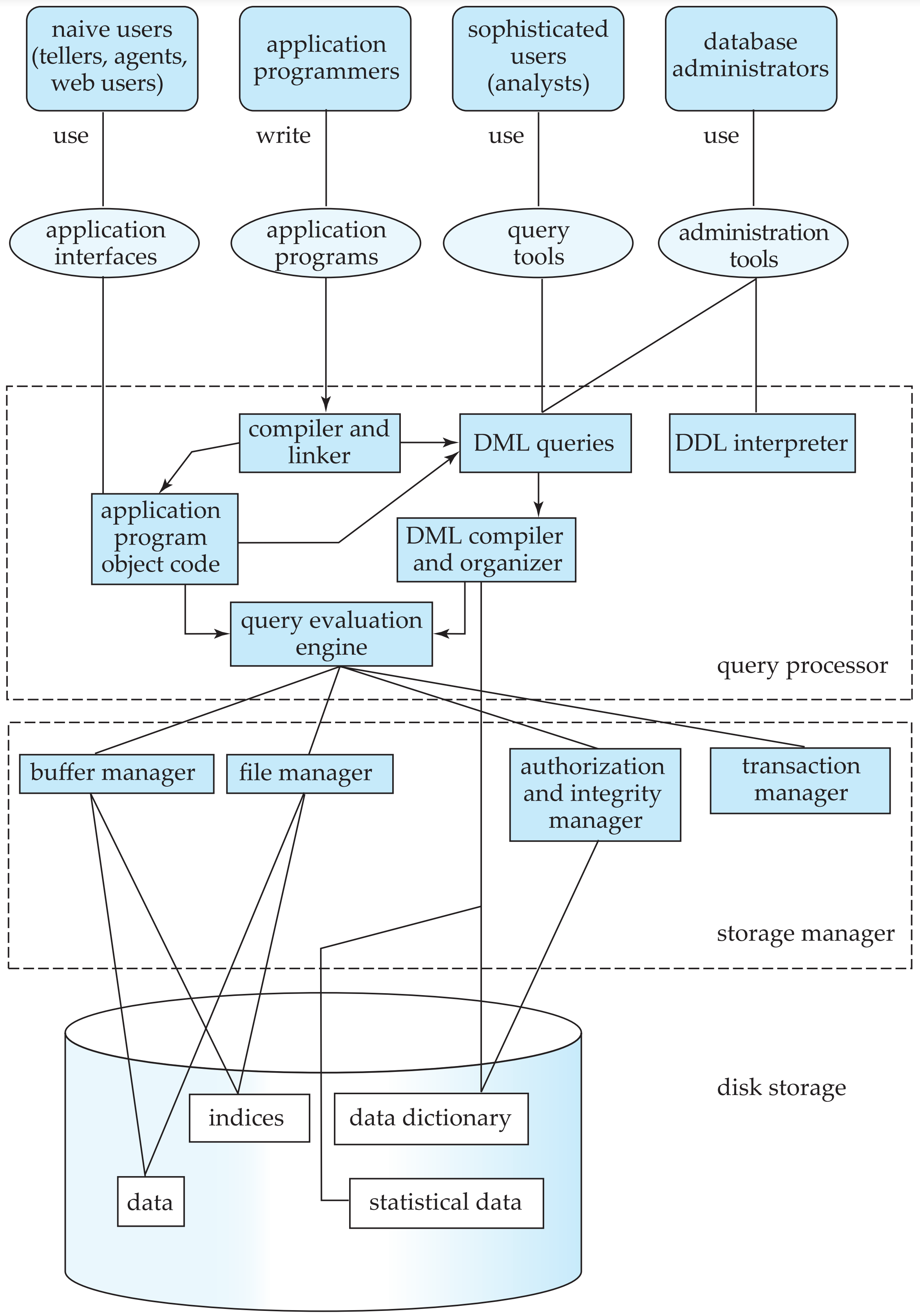
- The storage manager is responsible to the following tasks:
  - Interaction with the file manager
  - Efficient storing, retrieving and updating of data

- Design/implementation issues:
  - Storage access
  - File organization
  - Indexing and hashing



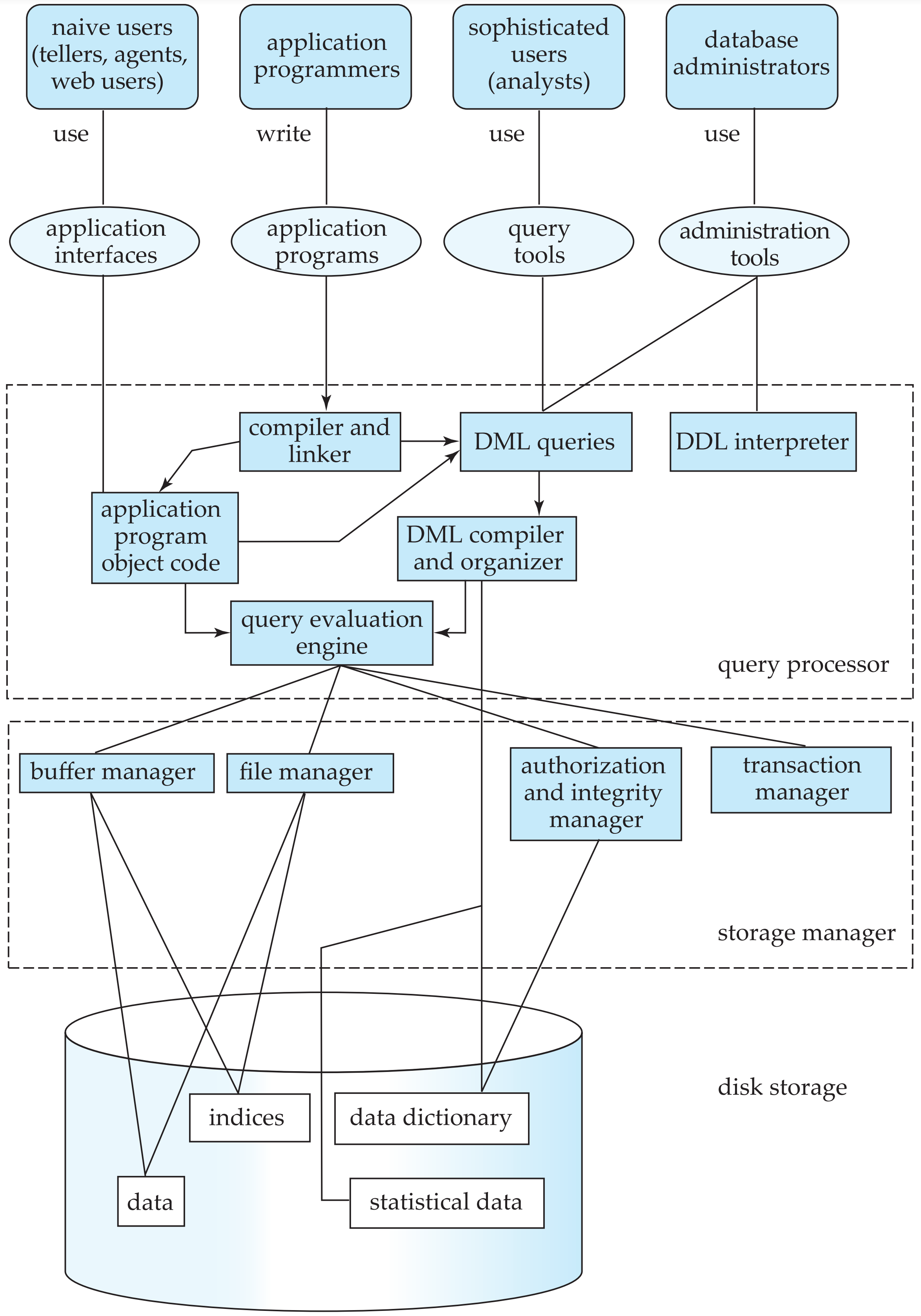
# Main DBMS Subsystems :: Transaction Management

- What if the system fails?
- A transaction is a collection of operations that performs a single logical function in a database application
- Transaction-management component ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.



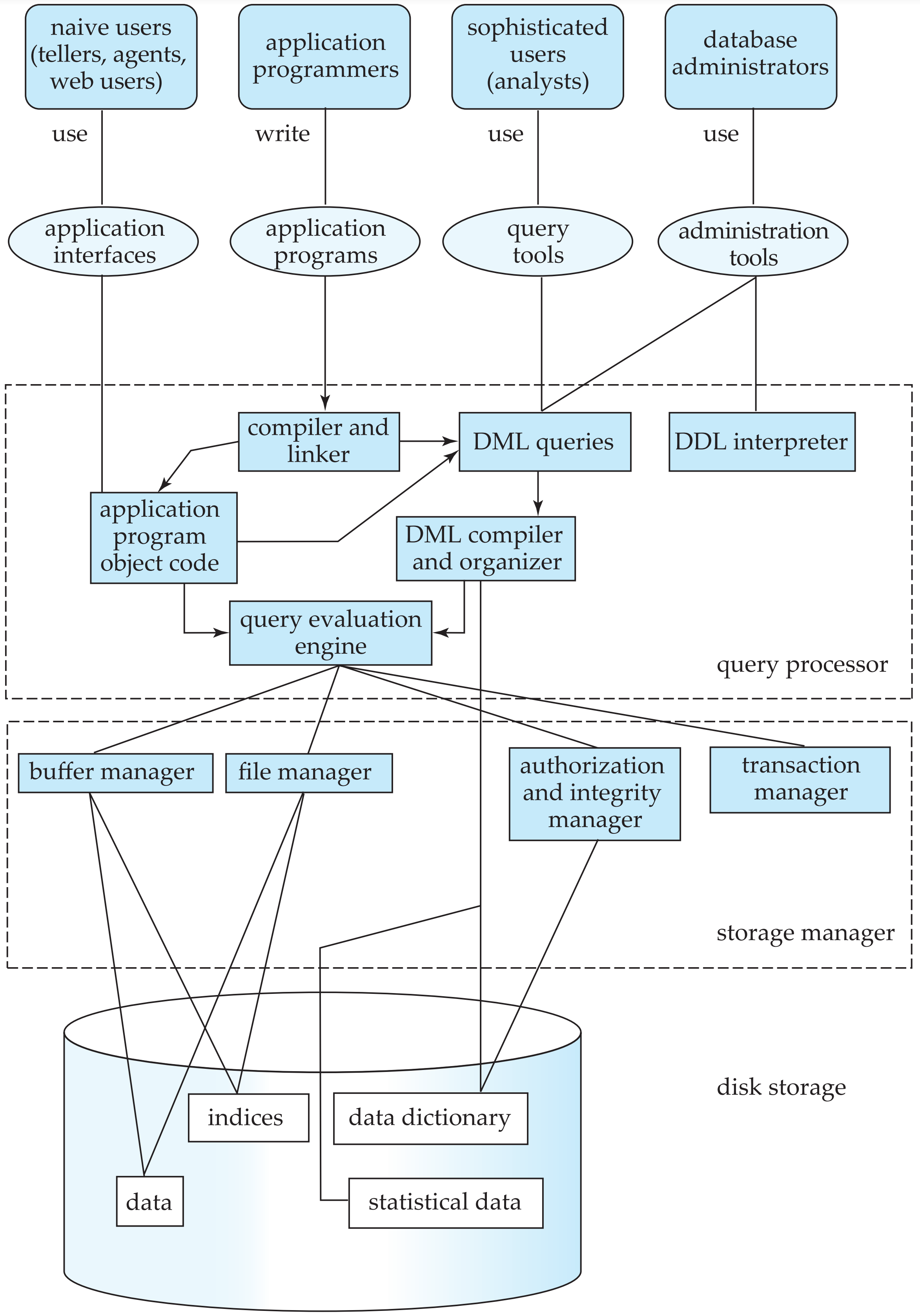
# Main DBMS Subsystems :: Concurrency Controller

- What if more than one user is concurrently updating the same data?
- Concurrency controller ensures that the interaction among the concurrent transactions do not compromise the consistency of the database.



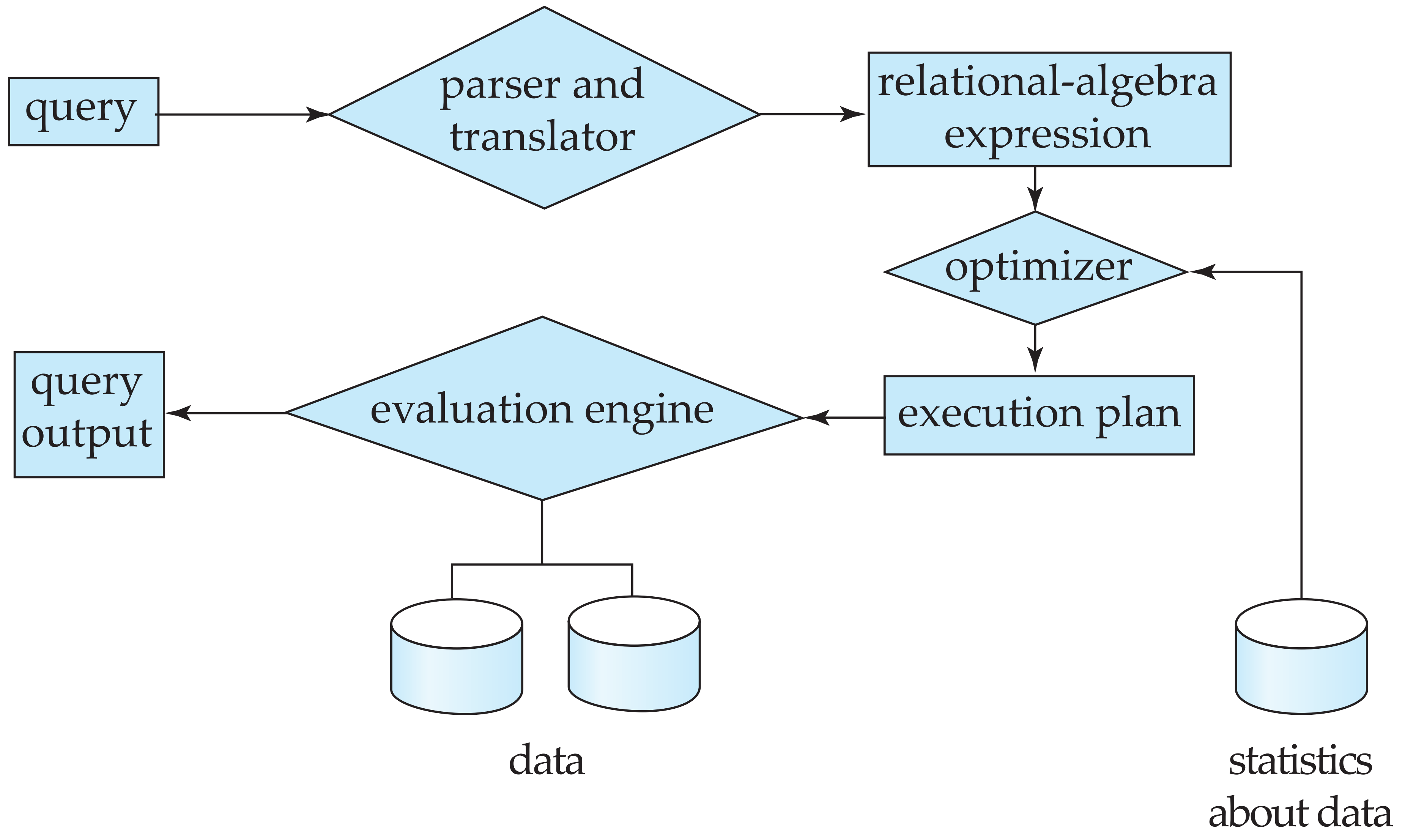
# Main DBMS Subsystems: Query Processor

- Here, by query is meant DDL, QL and DML expressions
- Code in the database languages is compiled and installed in the DBMS.
- Application programs in general-purpose languages can invoke compiled queries or pass queries for evaluation using the libraries and APIs exposed by the DBMS
- The HCI interfaces invoke bits of application that interact with the DBMS.



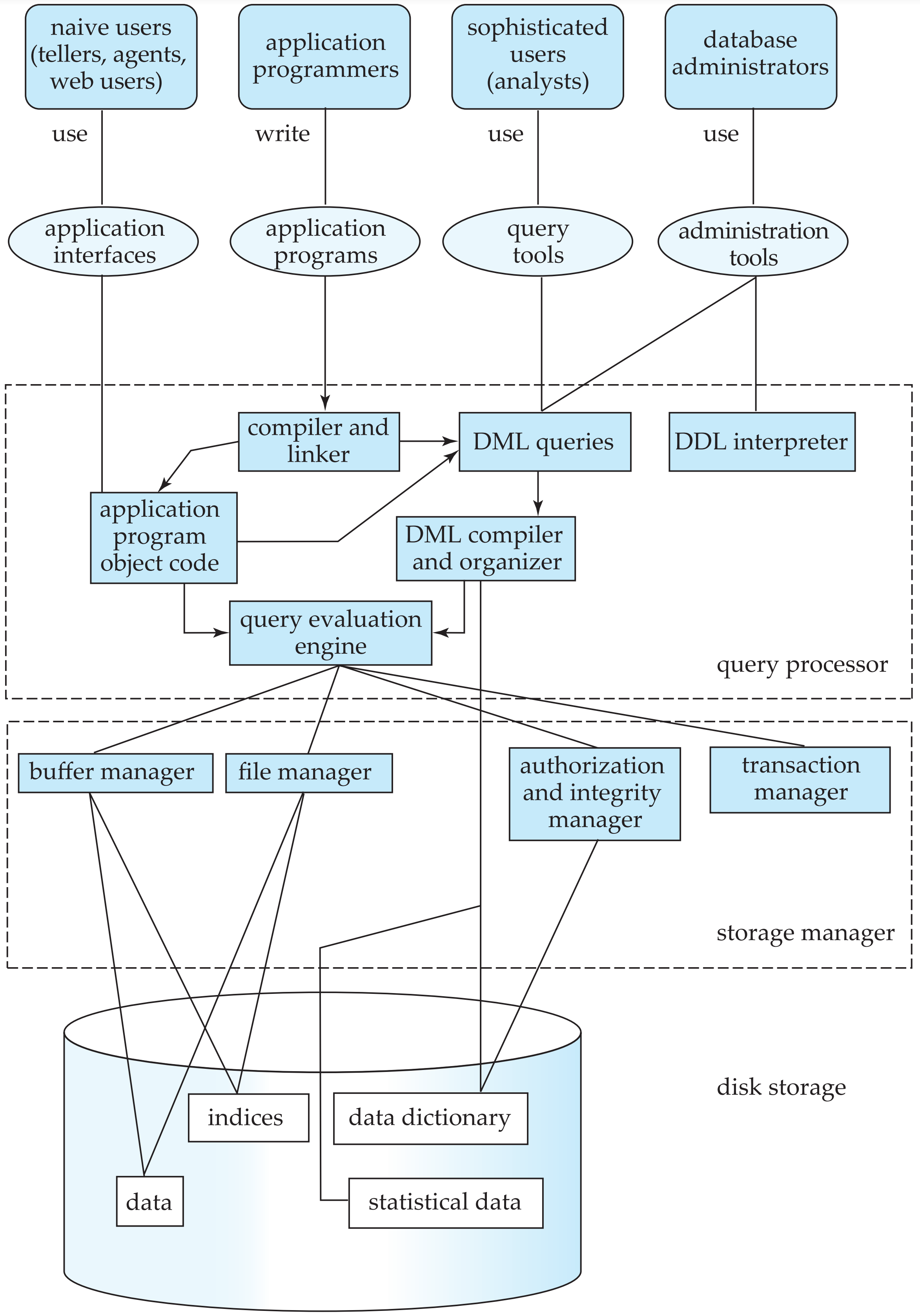
# Main Phases of Query Processing

- Parsing and translation
- Optimization
- Evaluation



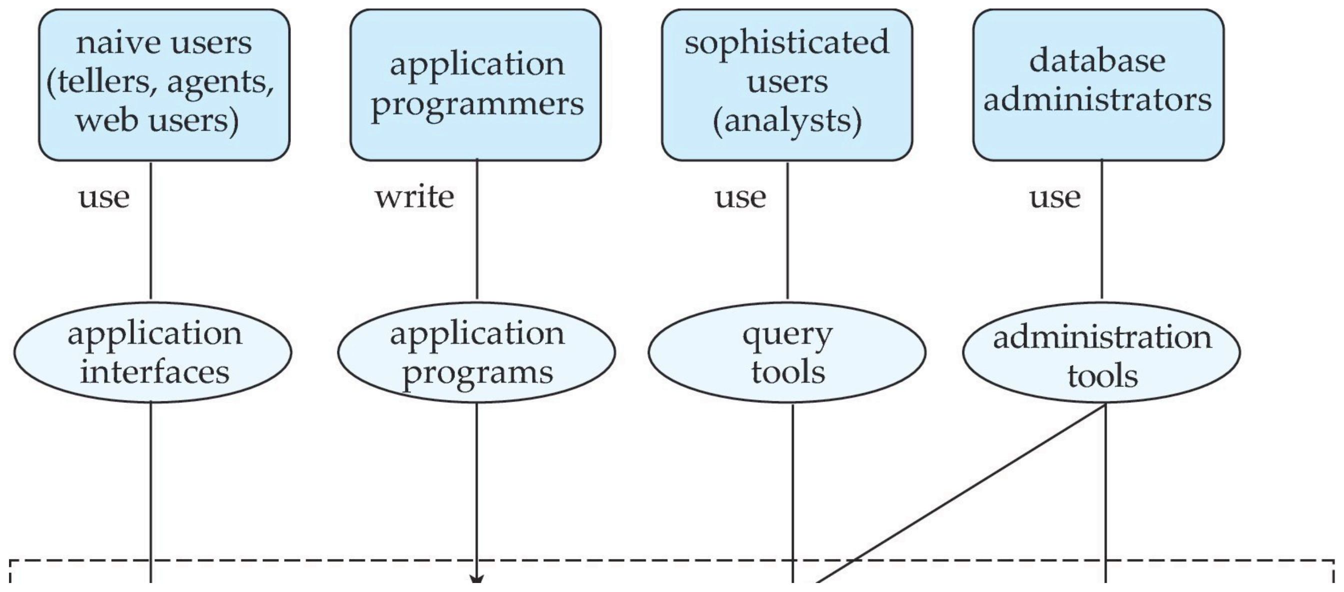
# Inside the Query Processor: Query Optimizer

- There are alternative ways of evaluating a given query
    - Many equivalent logical expressions
    - Many algorithms for each logical operation with different associated costs
- Cost difference between a good and a bad way of evaluating a query can be enormous
- Need to estimate the cost of operations
    - Depends critically on statistical information about relations which the database must maintain
    - Need to estimate statistics for intermediate results to compute cost of complex expressions



## DBMS Users: Roles

- Database administrators assign privileges to users and monitor performance.
- They continually study the statistics and revise configuration parameters to fine-tune performance.
- Sophisticated users (e.g., database designers) use CLI to implement and maintain logical models.
- Application programmers write software that uses APIs over the database, including HCI interfaces for naïve users.
- Naïve users rely on HCI interfaces for both back-room and customer-facing tasks.



Database

MANCHESTER
1824
The University of Manchester

What are the main kinds of database architecture in use today?

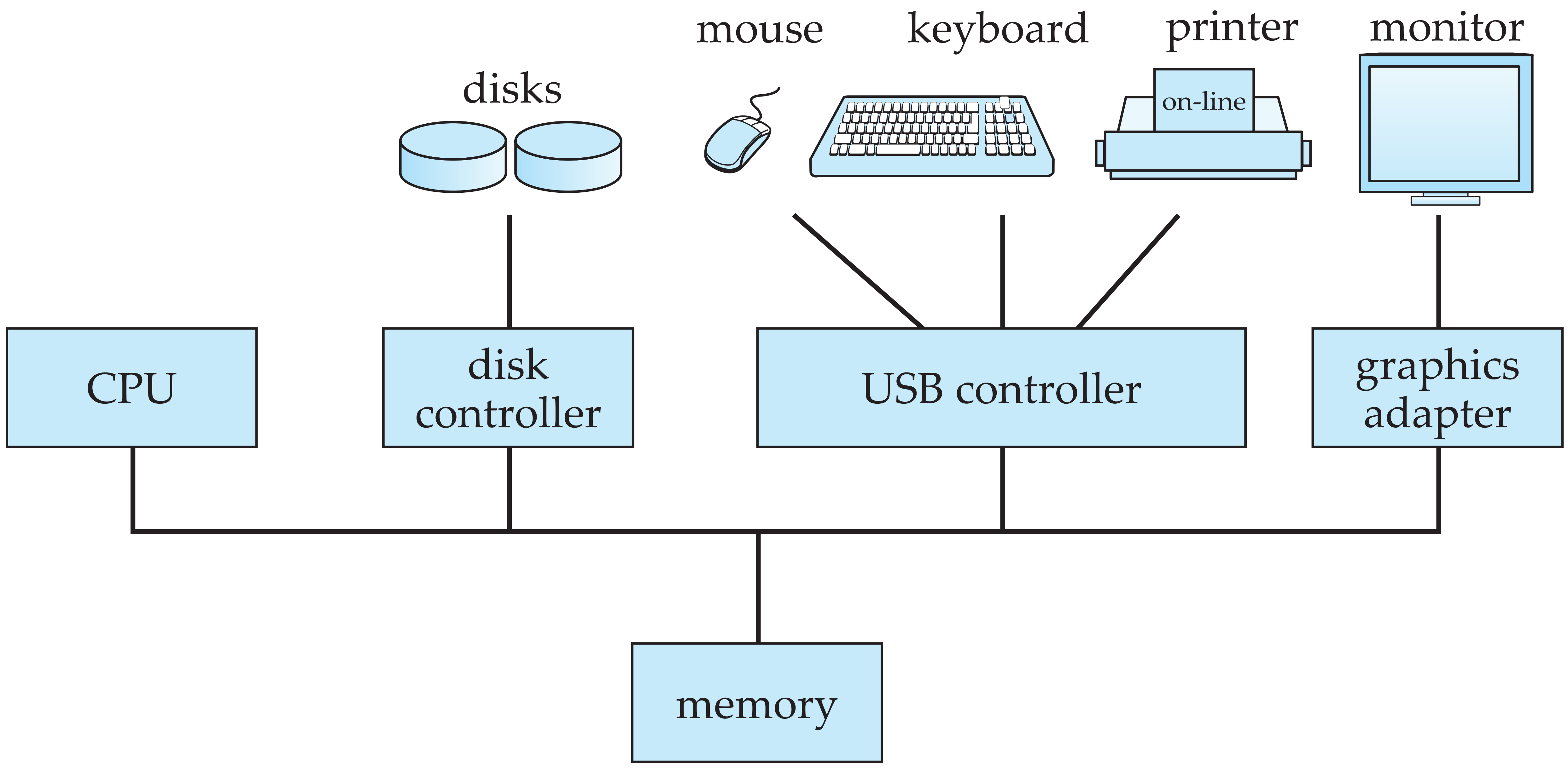
## Database Architecture: Centralized to Distributed

- The architecture of a database systems is greatly influenced by the underlying computer system on which the database is running:
  - Centralized
  - Client-server
  - Parallel (multi-processor)
  - Distributed

## Centralized Systems

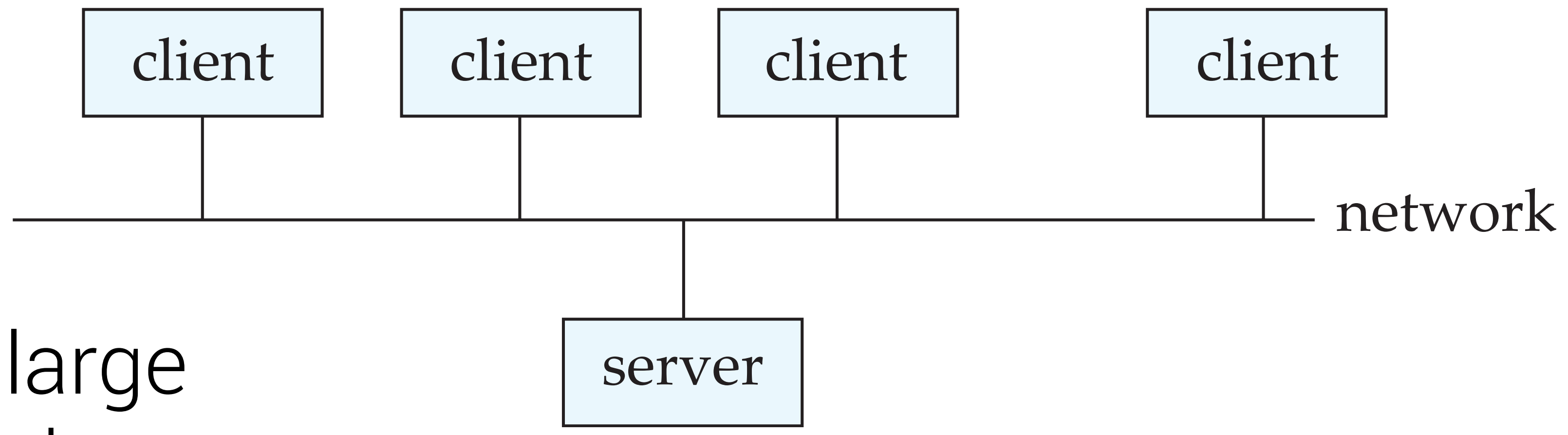
- Run on a single computer system and do not need to interact with other computer systems.
- General-purpose computer systems: one to a few CPUs and a number of device controllers that are connected through a common bus that provides access to shared memory
- Single-user systems (e.g., personal computer or workstation): desk-top unit, logically for single user, logically one CPU and one secondary mass storage unit; the OS need not usually worry about more than one user at a time
- Multi-user systems: logically and physically many disks, much more memory, multiple CPUs, and a multi-user OS to serve a large number of users who are connected via terminals, workstations, PCs

## Centralized Systems



## Client-Server Systems

## Client-Server Systems

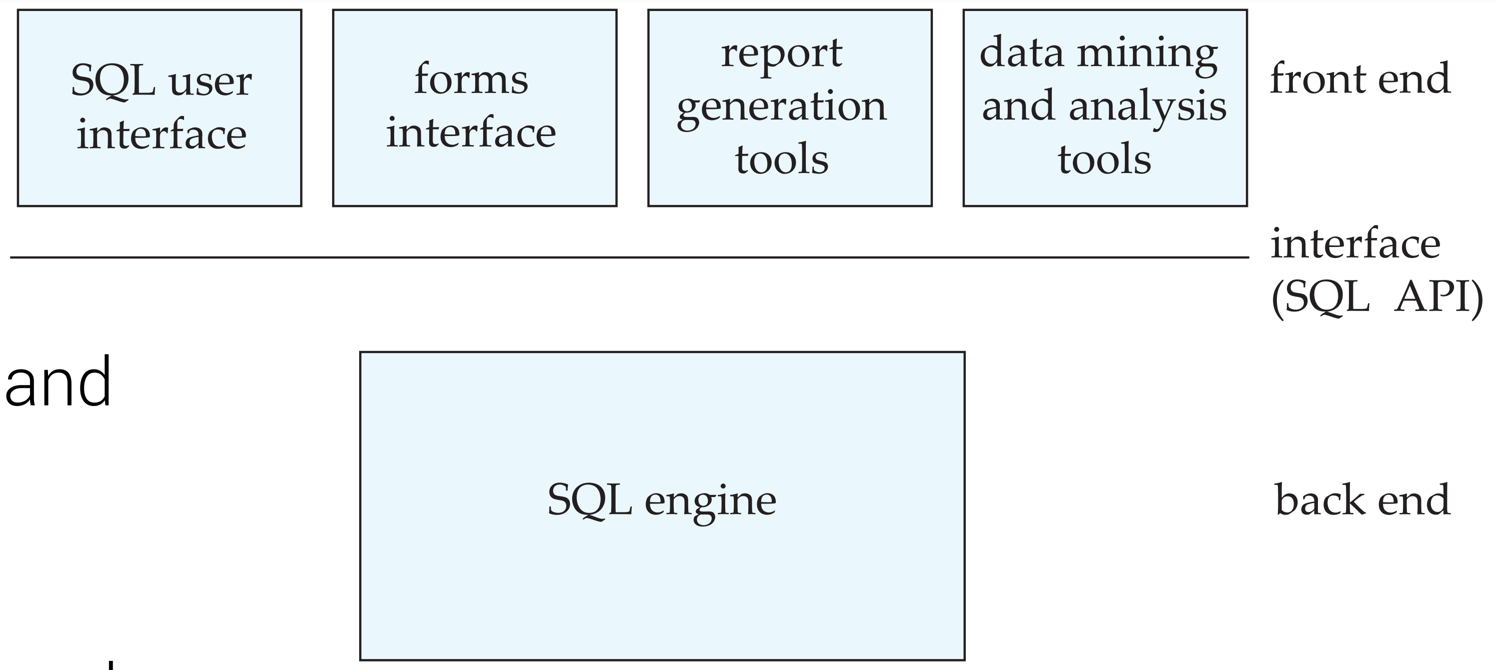


- Server systems are large multi-user systems that aim to satisfy requests generated at many client systems, each (logically) a single-user system.

- There is an asymmetry assumption that the server is many times better resourced than any client.

# 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
  - directly through a SQL engine (i.e., a query processor)
  - indirectly through an API.



## Client-Server Systems

- Some advantages of replacing mainframes (i.e., very large centralized computer systems) with networks of workstations or personal computers (i.e., single-user systems) acting as clients by way of a connection to back-end server machines (i.e., multi-user systems) are:
  - more functionality for the same cost,
  - flexibility in locating resources and expanding facilities,
  - better user interfaces,
  - easier maintenance.

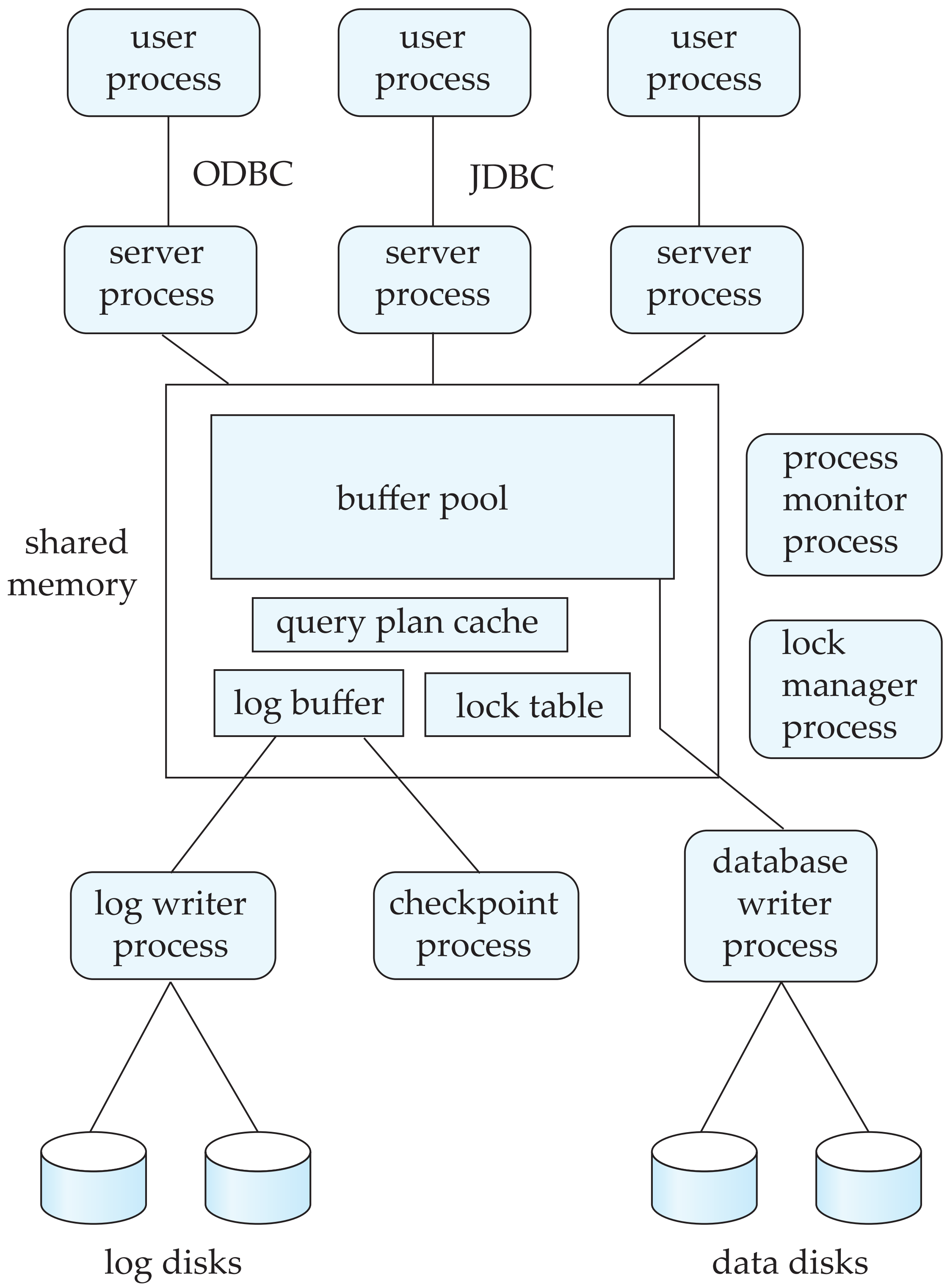
## Types of Server

- Server systems can be broadly categorized into two kinds:
  - transaction servers, which are widely used in relational database systems
  - data servers, which are often used in object-based stores

The University of Manchester

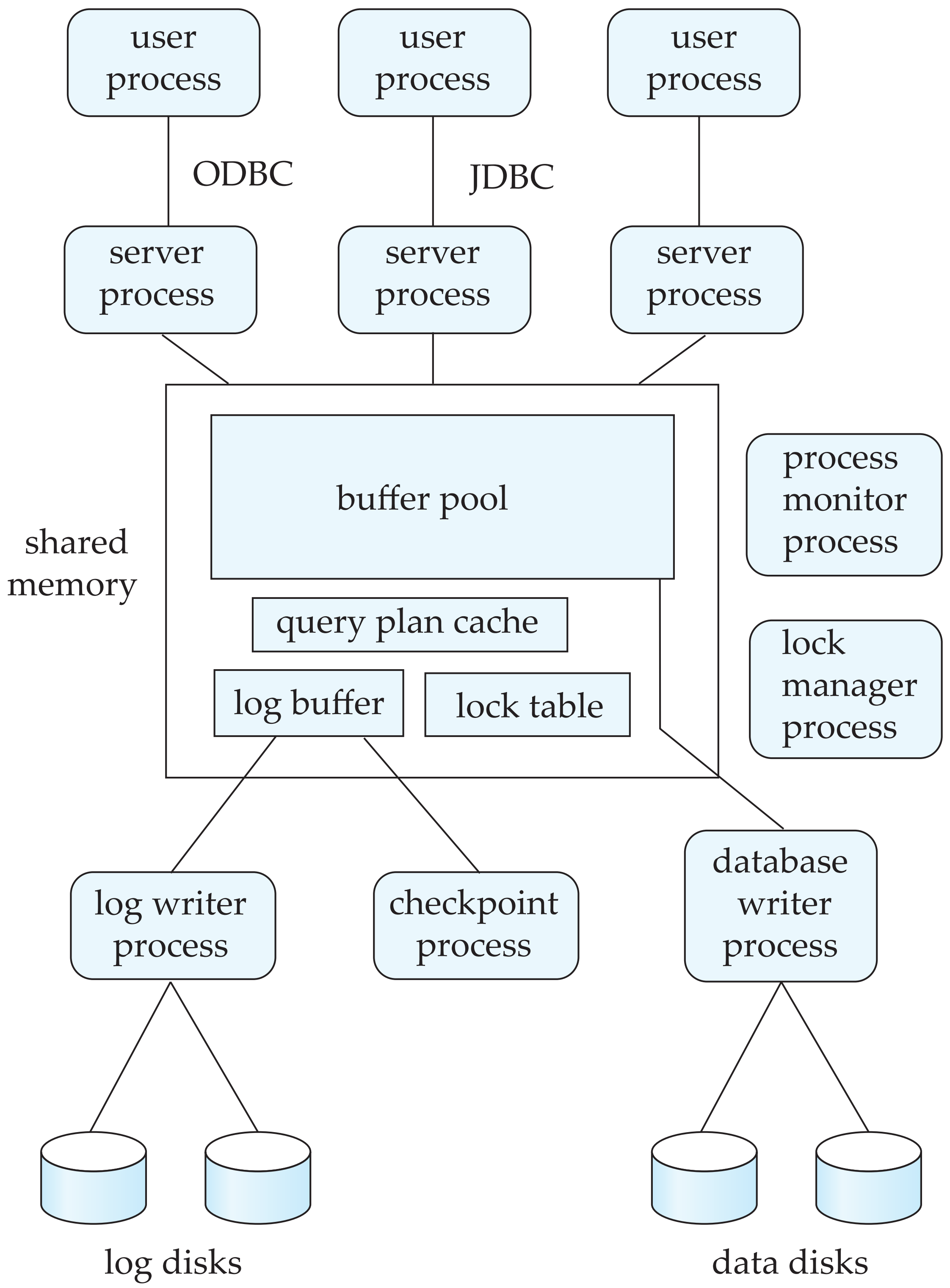
## Transaction Servers

# Transaction Servers :: Communication



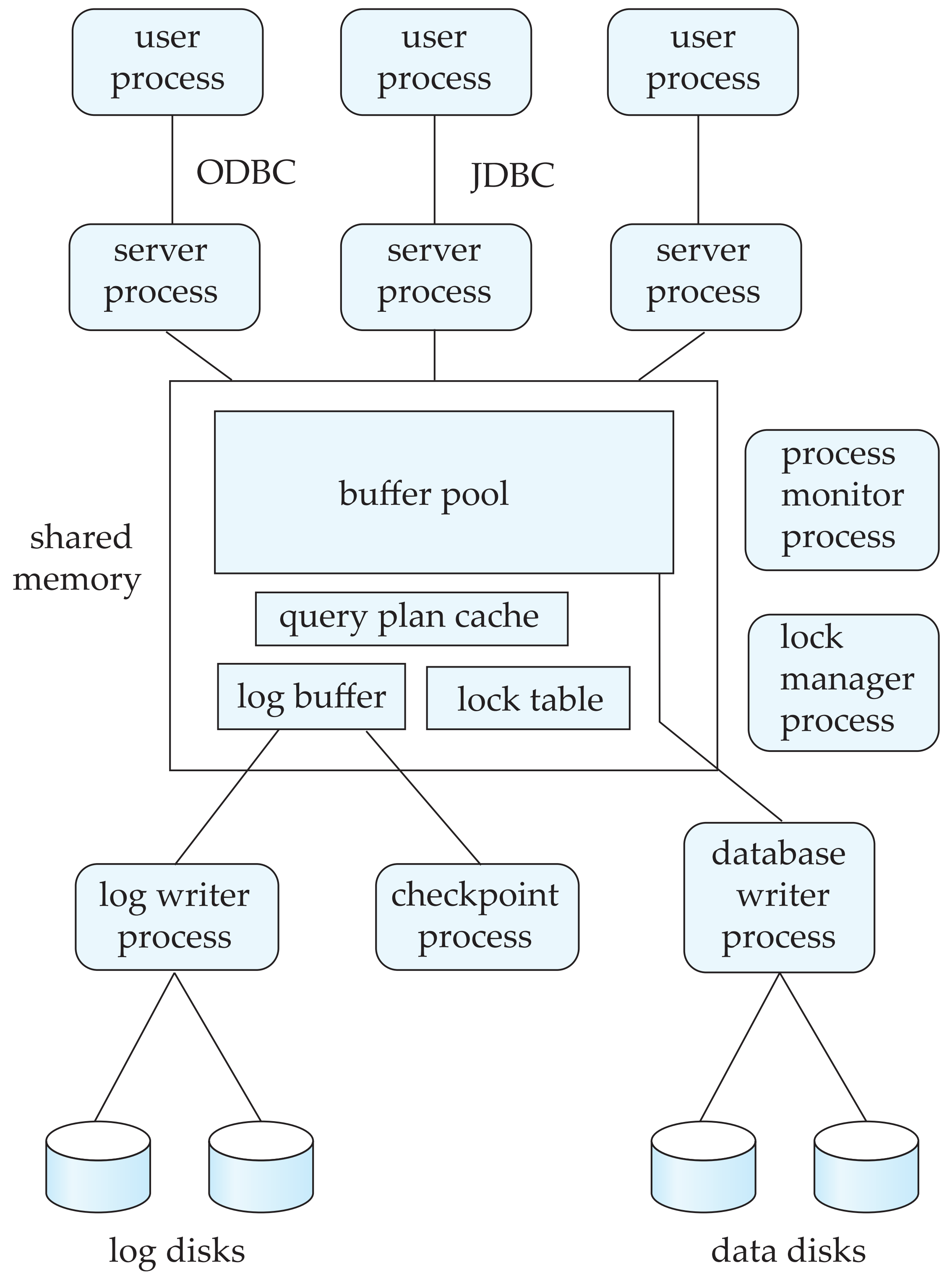
- Also called query server or SQL server systems
  - Clients send requests to the server
  - Transactions are executed at the server
  - Results are shipped back to the client.

# Transaction Servers :: Mechanisms



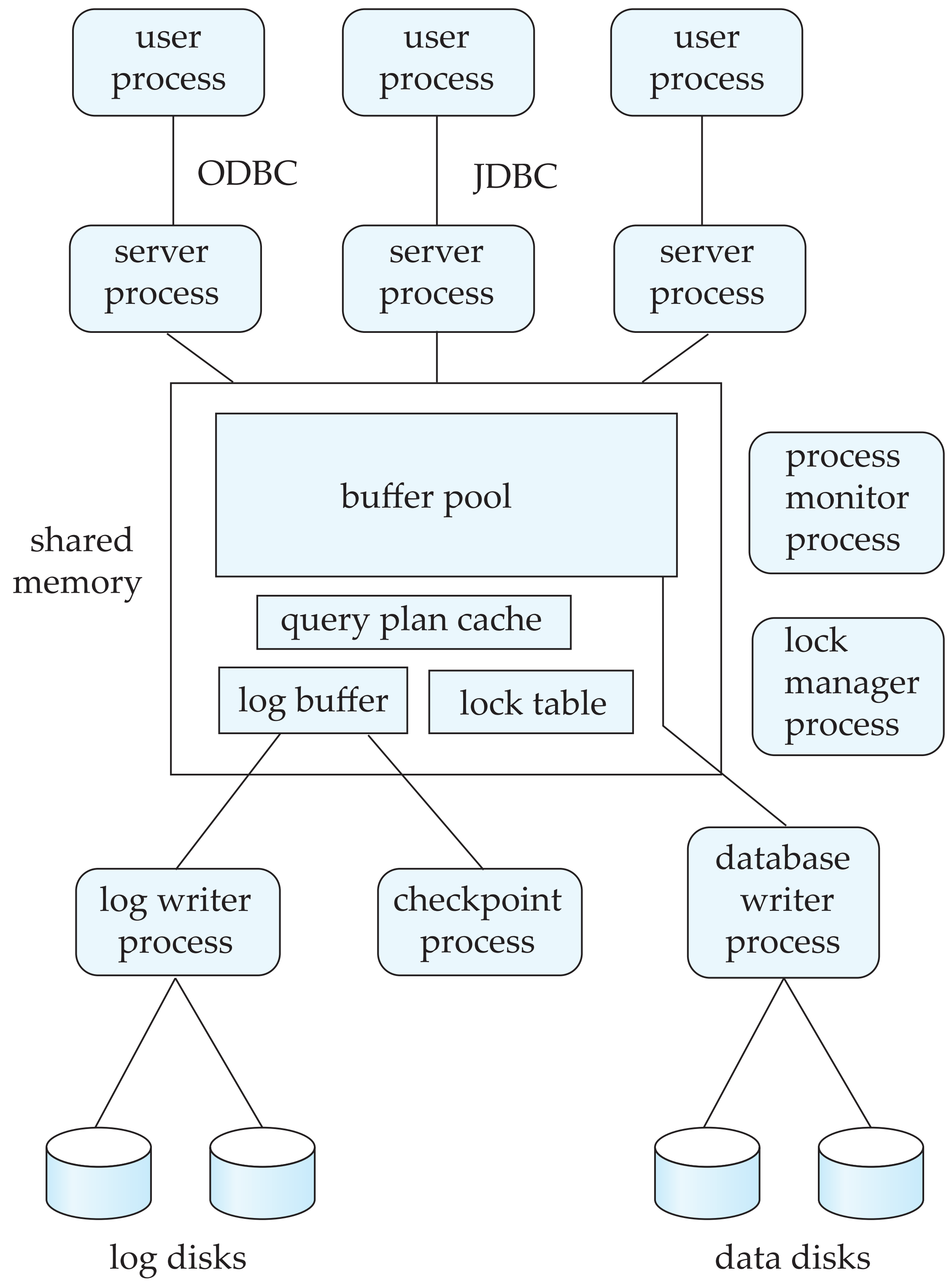
- Requests are specified in SQL, and communicated to the server through a remote procedure call (RPC) mechanism.
- Transactional RPC allows many RPC calls to form a transaction.
- Open Database Connectivity (ODBC) is a C-language API for connecting to a server, sending SQL requests, and receiving results.
- JDBC is Java-based ODBC.

## Transaction Servers :: Querying Processes



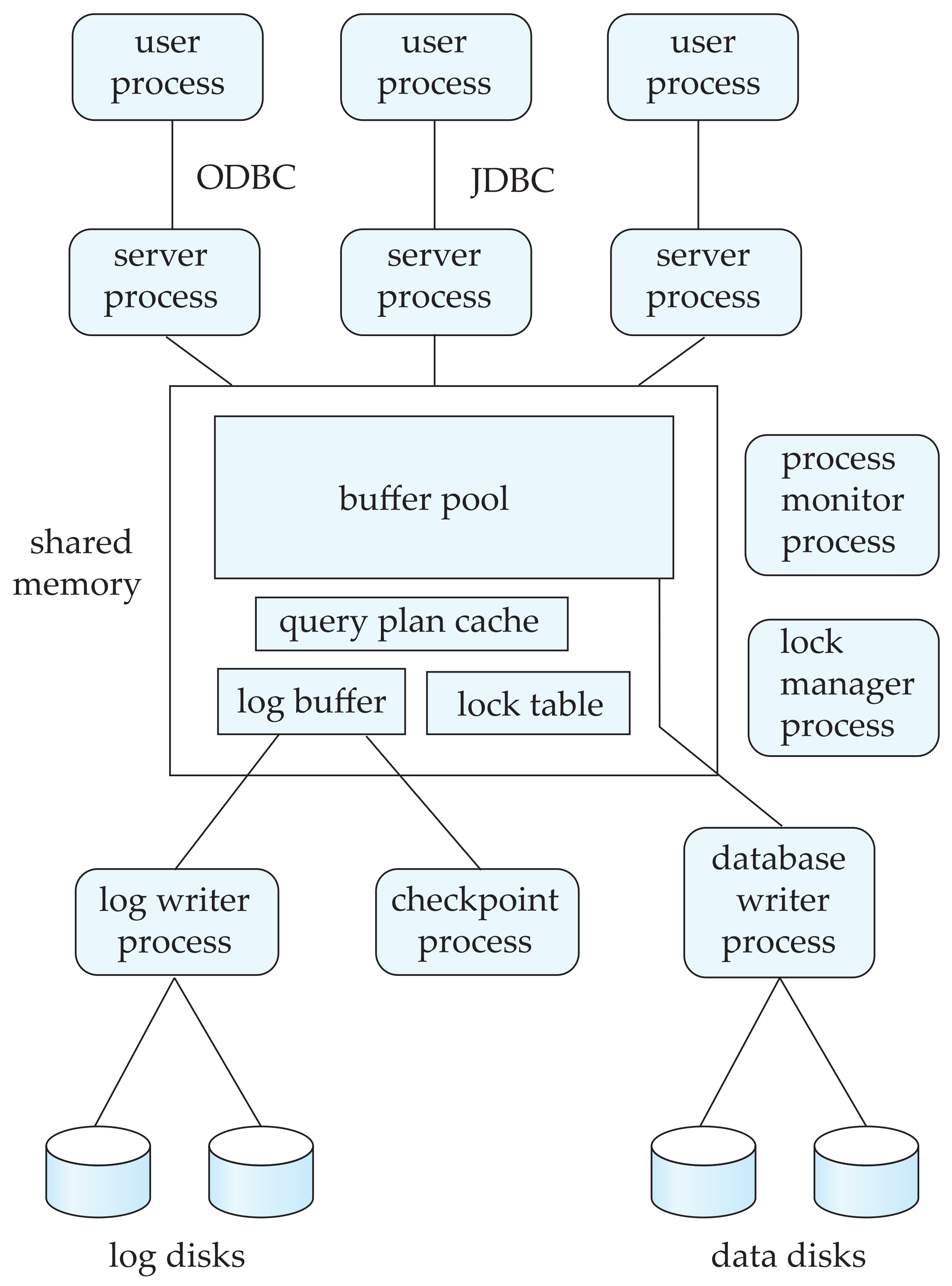
- In a transaction server, many processes cooperate to implement the database architecture we saw earlier.
- Queries are handled by server processes (i.e., these instantiate query processing functionality)
- These are typically multithreaded, typically massively so

# Transaction Servers :: Management Processes



- Other processes control access to shared memory:
  - lock manager for concurrent access
  - database writer for traffic between volatile and persistent storage
  - process monitor, checkpoint and log-writer for transactional semantics (e.g., commits and aborts/roll-backs, recovery, etc.)
- There are many trade-offs at play here: all the helper processes (i.e., lock manager, buffer writers/off-loaders, etc.) ensure semantic correctness but risk being performance drains

## Transaction Servers :: Data Structure Management



- Shared memory contains shared data, i.e.:
  - Buffer pool
  - Lock table
  - Log buffer
  - Cached query plans (reused if same query submitted again)
- All database processes can access shared memory
- To ensure that no two processes are accessing the same data structure at the same time, databases systems implement some form of mutual exclusion

MANCHESTER
1824
The University of Manchester

## Data Servers

## Data Servers, Classically

- Used in high-speed LANs, in cases where
  - The clients are comparable in processing power to the server
  - The tasks to be executed are compute-intensive.
- Data is shipped to clients for processing, and then shipped results back to the server.
- This architecture requires full back-end functionality at the clients.
- Often used in object-based systems

## Data Servers, Recently

- More recently, so-called NoSQL systems focus on massive replication of essentially equipotent systems

- Specialization leads to data servers for, say, images or sound, to be separate from classical retrieval

- Note that in the NoSQL world, there no queries in the sense we understand them here.

## Parallel Systems

## Parallel Systems

- Parallel database systems consist of multiple processors and multiple disks connected by a fast interconnection network.
- A coarse-grain-parallel machine consists of a small number of powerful processors
- A massively-parallel or fine-grain-parallel machine utilizes thousands of smaller processors.

## Parallel Systems

- Two main performance measures:
  - Throughput: the number of tasks that can be completed in a given time interval
  - Response time: the amount of time it takes to complete a single task from the time it is submitted
- The goal is to improve performance as we increase the degree of parallelism (i.e., the number of resources deployed in parallel)

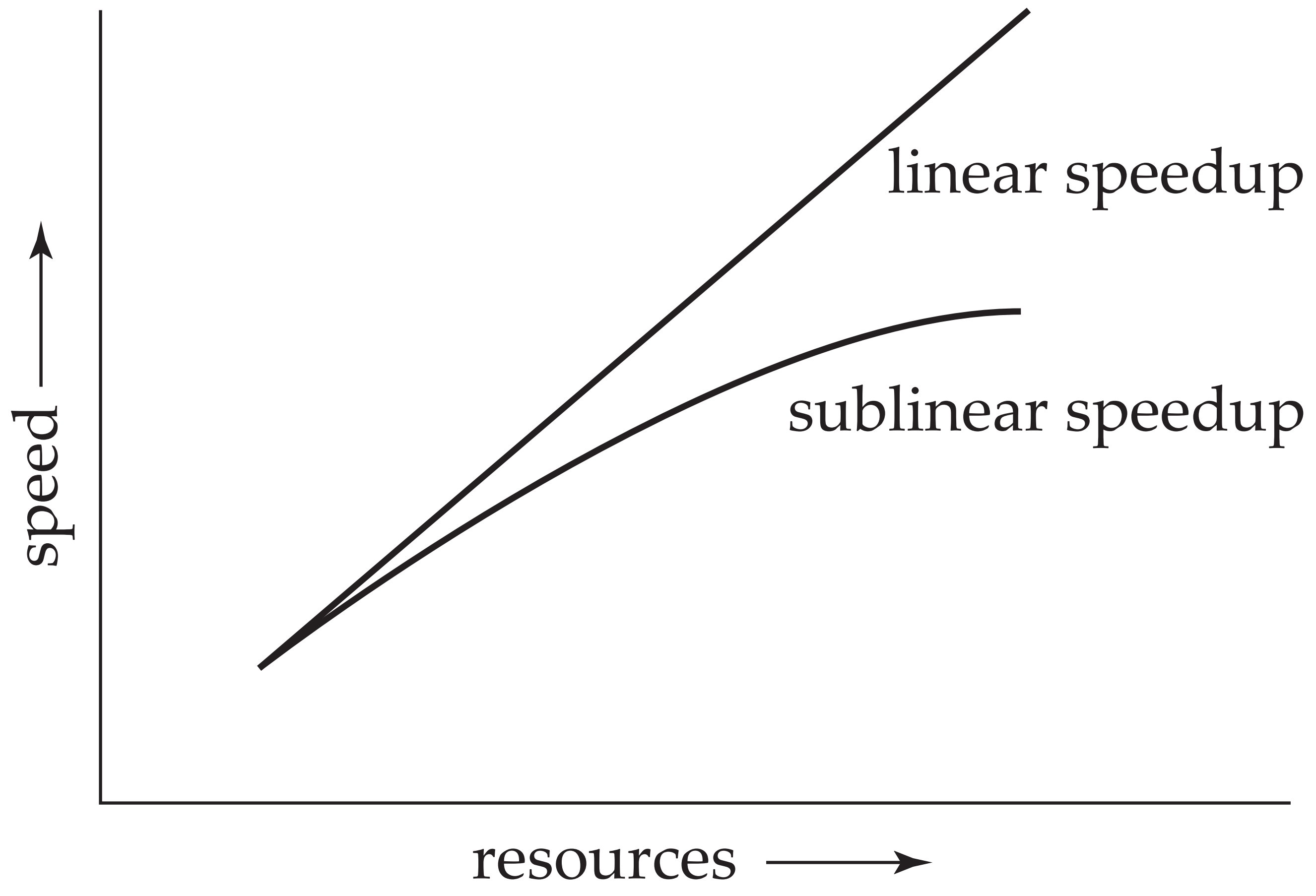
## Speed-Up

|P| = p
|S| = 3

|P| = p
|L| = 4.|S|

- Assume a problem of a given size |P| = *p* that is executing on a small system of size |S| is given to a larger system |L| = *n*.|S|
- The speed-up goal is to then solve P in time inversely proportional to *n*, i.e., the growth in resources
- The problem size remains constant

# Speed-Up



$|P| = p$ $\qquad$ $|P| = p$

$|S| = 3$ $\qquad$ $|L| = 4.|S|$

$T_S = 1$ $\qquad$ $T_L = 0.25$

$T_S/T_L = 4 \rightarrow$ linear

- Speed-up is defined as

  $$SpUP = T_S/T_L$$

  i.e., the elapsed-time $T_S$ to solve P in S divided by the elapsed-time $T_L$ to solve P in L where $|L| = n.|S|$, $n>1$

- Speed-up is linear if SpUP = $n$
- Sub-linear speed-up means that the full investment on more resources is not fully recouped in elapsed-time reduction

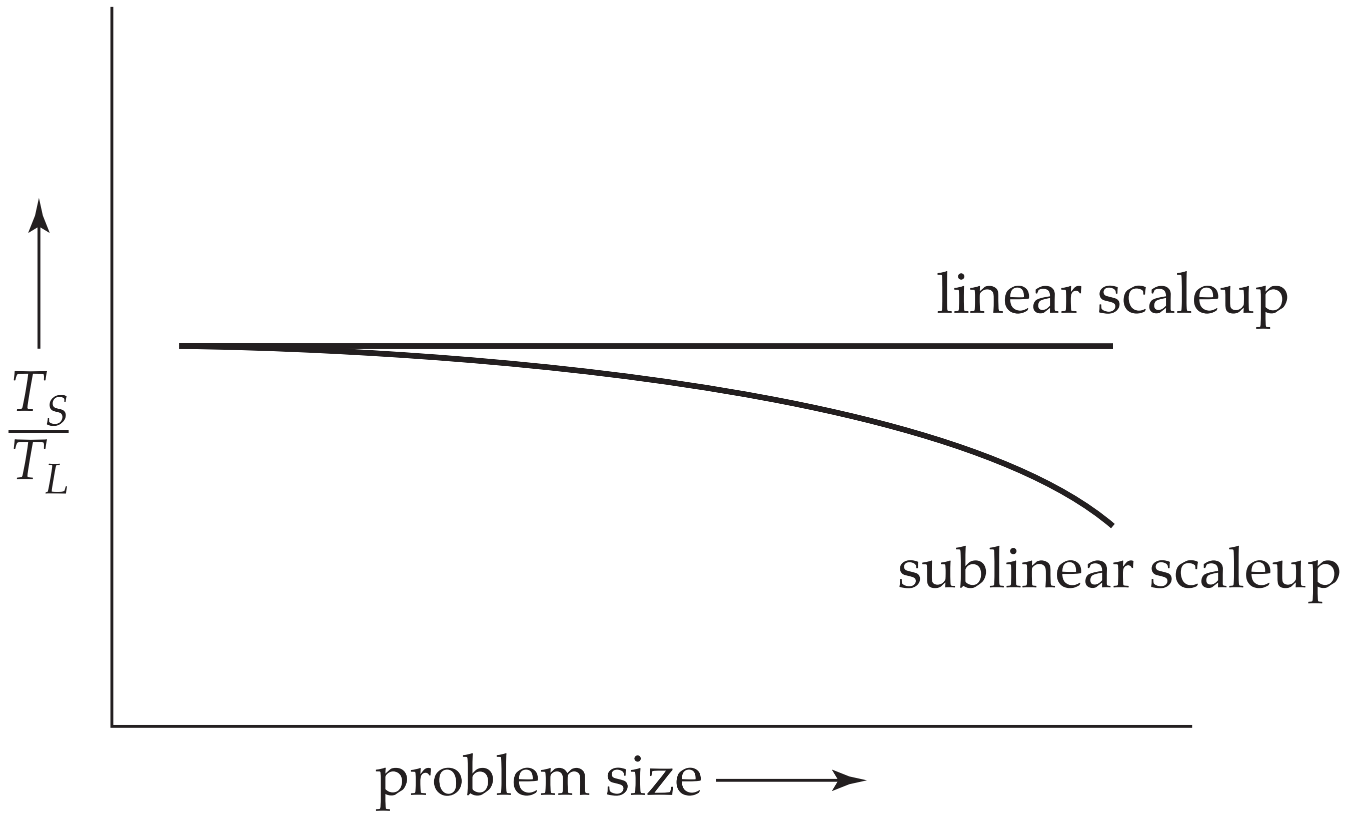
## Scale-Up

$|P| = p$ $\quad$ $|P'| = 4.p$

$|S| = 3$ $\quad$ $|L| = 4.|S|$

- Assume a problem of a given size |P| that is executing on a small system of size |S|.
- Now, assume that faced with an increase of *k*, *k*>1 in the size of the problem, we increase the system *k*-times
- The goal here is to solve the larger problem in the same time as before.
- Both the problem size and the system size grow.

## Scale-Up



$|P| = p$ $\quad |P'| = 4.p$

$|S| = 3$ $\quad |L| = 4.|S|$

$T_S = 2$ $\quad T_L = 2$

$T_S/T_L = 1 \rightarrow$ linear

- Scale-up is defined as

  $$ScUP = T_S/T_L$$

  i. e., the elapsed-time $T_S$ to solve P in S divided by the elapsed-time $T_L$ to solve P′ in L where $|P'| = k.|P|$ and $|L| = k.|S|$, $k>1$

- Scale-up is linear if ScUP = 1
- Sub-linear scale-up means that the full investment on more resources is not fully recouped in elapsed-time stability

## Batch Scale-UP

- A single large job; typical of most decision support queries and scientific simulation.
- Use an N-times larger computer on N-times larger problem.

## Transaction Scale-Up

- Numerous small queries submitted by independent users to a shared database; typical transaction processing and timesharing systems.
- N-times as many users submitting requests (hence, N-times as many requests) to an N-times larger database, on an N-times larger computer.
- Well-suited to parallel execution.

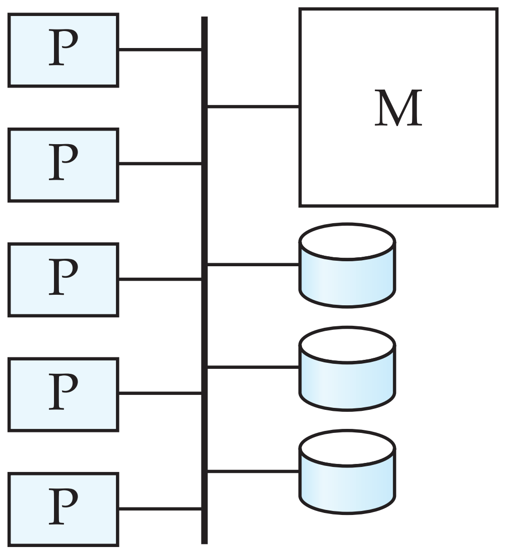
## Speed-Up and Scale-Up Limiting Factors

- Speed-up and scale-up are often sublinear due to:
    - Start-up costs: Cost of starting up multiple processes may dominate computation time, if the degree of parallelism is high.
    - Interference: Processes accessing shared resources (e.g., system bus, disks, or locks) compete with each other, thus spending time waiting on other processes, rather than performing useful work.
    - Skew: Increasing the degree of parallelism increases the variance in service times of parallel-executing tasks. Overall execution time determined by slowest of parallel-executing tasks.

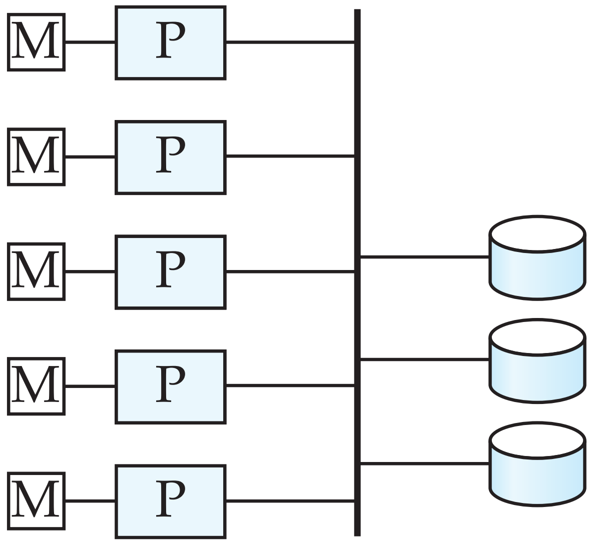
## Parallel Database Architectures

- Shared memory -- processors share a common memory
- Shared disk -- processors share a common disk (array)
- Shared nothing -- processors share neither a common memory nor common disk (array)
- Hierarchical -- hybrid of the above architectures

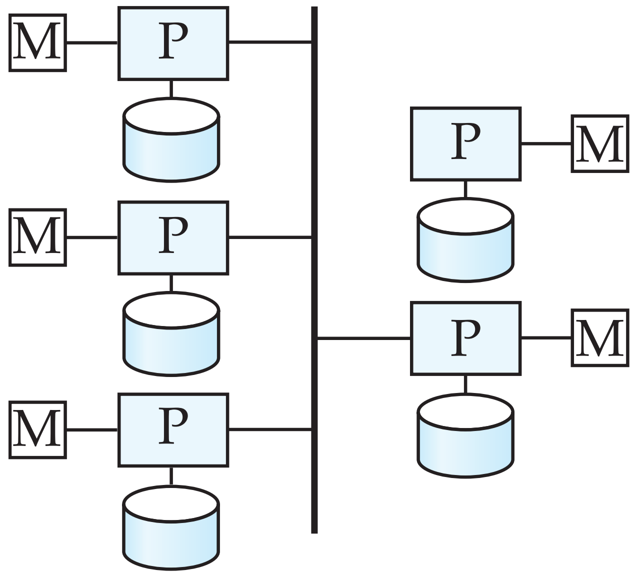
# Parallel Database Architectures



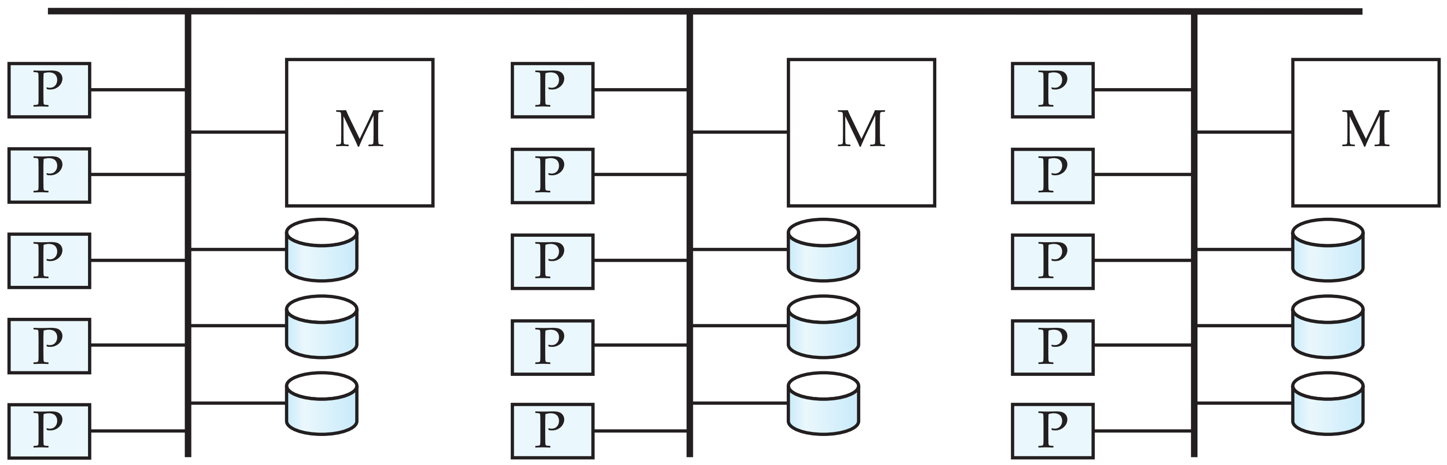
(a) shared memory



(b) shared disk



(c) shared nothing



(d) hierarchical

## Shared-Memory Architectures

- Processors and disks have access to a common memory, typically via a bus or through an interconnection network.
- Extremely efficient communication between processors — data in shared memory can be accessed by any processor without having to move it using software.

## Shared-Memory Architectures

- Downside – architecture is not scalable beyond 32 or 64 processors since the bus or the interconnection network becomes a bottleneck
- Widely used for lower degrees of parallelism (4 to 8).

## Shared-Disk Architectures

- All processors can directly access all disks via an interconnection network, but the processors have private memories.

- The memory bus is not a bottleneck

- Architecture provides a degree of fault-tolerance — if a processor fails, the other processors can take over its tasks since the database is resident on disks that are accessible from all processors.

## Shared-Disk Architectures

- Downside: bottleneck now occurs at interconnection to the disk subsystem.
- Shared-disk systems can scale to a somewhat larger number of processors, but communication between processors is slower.
- Examples: IBM Sysplex and DEC clusters (now part of Compaq) running Rdb (now Oracle Rdb) were early commercial users

## Shared-Nothing Architectures

- Node consists of a processor, memory, and one or more disks. Processors at one node communicate with another processor at another node using an interconnection network. A node functions as the server for the data on the disk or disks the node owns.

- Data accessed from local disks (and local memory accesses) do not pass through interconnection network, thereby minimizing the interference of resource sharing.

## Shared-Nothing Architectures

- Shared-nothing multiprocessors can be scaled up to thousands of processors without interference.
- Main drawback: cost of communication and non-local disk access; sending data involves software interaction at both ends.
- Examples: Teradata, Tandem, Oracle-n CUBE

## Hierarchical Architectures

- Combines characteristics of shared-memory, shared-disk, and shared-nothing architectures.
- Top level is a shared-nothing architecture – nodes connected by an interconnection network, and do not share disks or memory with each other.
- Each node of the system could be a shared-memory system with a few processors.

## Hierarchical Architectures

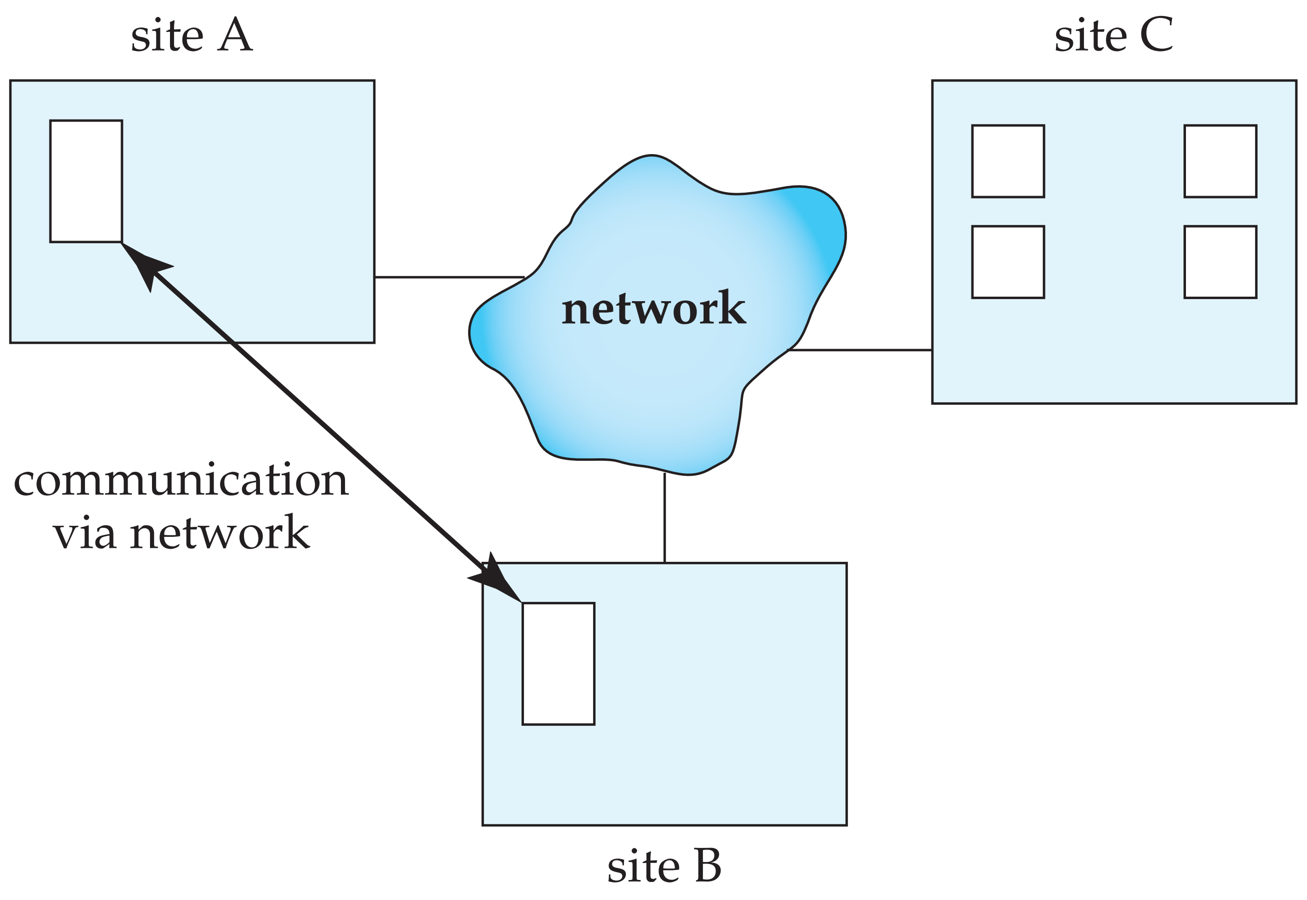
- Alternatively, each node could be a shared-disk system, and each of the systems sharing a set of disks could be a shared-memory system.
- Reduce the complexity of programming such systems by distributed virtual-memory architectures
- Also called non-uniform memory architecture (NUMA)

MANCHESTER
1824
The University of Manchester

## Distributed Systems

# Distributed Systems

- Data spread over multiple machines (also referred to as sites or nodes).
- Network interconnects the machines
- Data shared by users on multiple machines



## Distributed Databases :: Homogeneous

- Same software/schema on all sites, data may be partitioned among sites
- Goal: provide a view of a single database, hiding details of distribution

## Distributed Databases :: Heterogeneous

- Different software/schema on different sites
- Goal: integrate existing databases to provide useful functionality

## Distributed Databases :: Local v. Global

- A local transaction accesses data in the single site at which the transaction was initiated.
- A global transaction either accesses data in a site different from the one at which the transaction was initiated or accesses data in several different sites.

## Trade-Offs in Distributed Systems

- Sharing data – users at one site able to access the data residing at some other sites.

- Autonomy – each site is able to retain a degree of control over data stored locally.

- Higher system availability through redundancy — data can be replicated at remote sites, and system can function even if a site fails.

## Trade-Offs in Distributed Systems

- Disadvantage: added complexity required to ensure proper coordination among sites.
  - Software development cost.
  - Greater potential for bugs.
  - Increased processing overhead.

## Implementation Issues in Distributed Databases

- Atomicity needed even for transactions that update data at multiple sites
- The two-phase commit protocol (2PC) is used to ensure atomicity
    - Basic idea: each site executes transaction until just before commit, and the leaves final decision to a coordinator
    - Each site must follow decision of coordinator, even if there is a failure while waiting for coordinator's decision

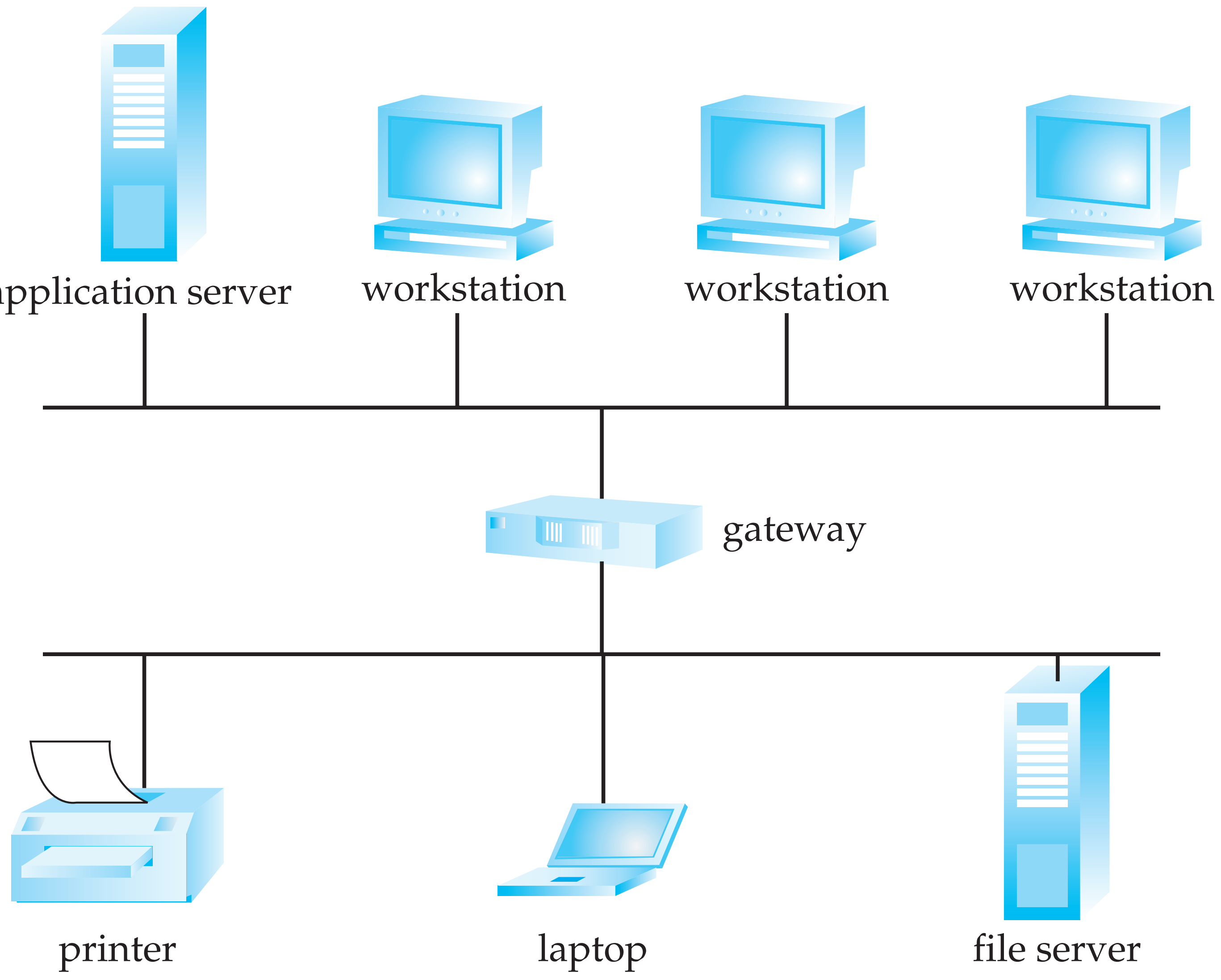
## Implementation Issues in Distributed Databases

- 2PC is not always appropriate: other transaction models based on persistent messaging, and workflows, are also used
- Distributed concurrency control (and deadlock detection) required
- Data items may be replicated to improve data availability
- (More details later in this course unit)

# Types of Network

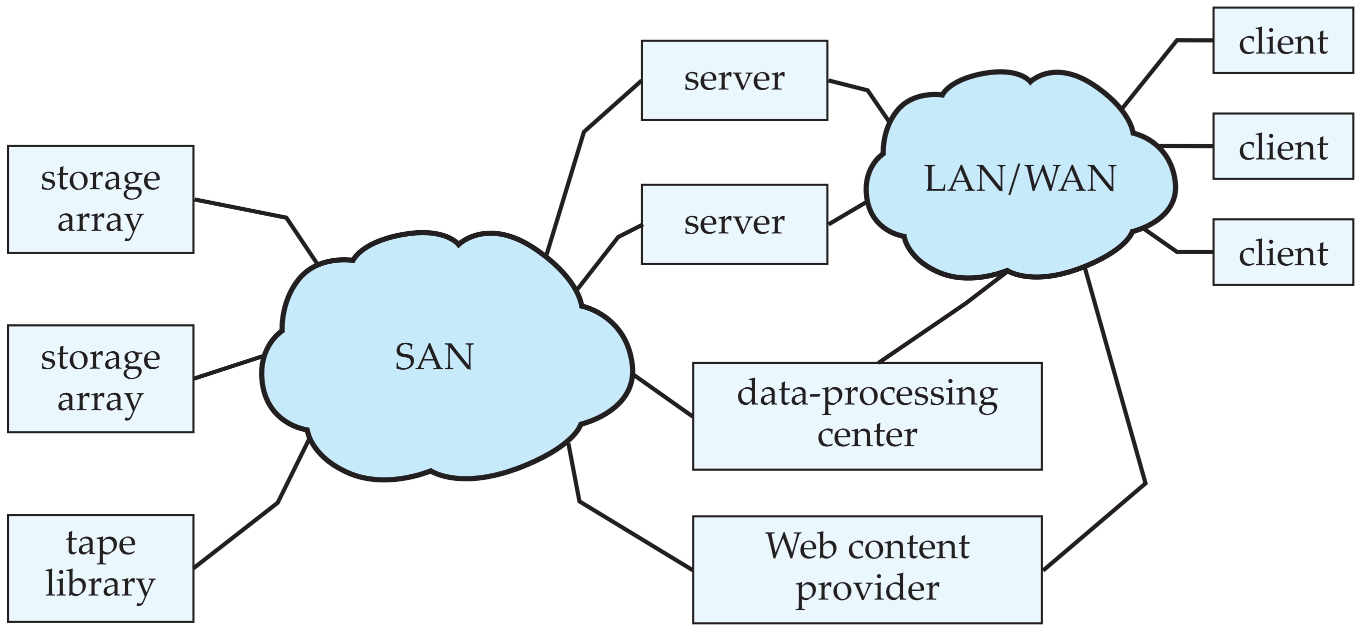
- Local-area networks (LANs) – composed of processors that are distributed over small geographical areas, such as a single building or a few adjacent buildings.

- Wide-area networks (WANs) – composed of processors distributed over a large geographical area.



# Types of Network

- WANs with continuous connection (e.g., the Internet) are needed for implementing distributed database systems
- Groupware applications such as Lotus notes can work on WANs with discontinuous connection:
  - Data is replicated.
  - Updates are propagated to replicas periodically.
  - Copies of data may be updated independently.
  - Non-serializable executions can thus result. Resolution is application dependent.



## Summary

## DBMS Components, Subsystems, Users, and Large-Scale Deployment Architectures

- The main components of DBMSs are the query processor and the storage manager.
- The query processor (QP) parses, translates and optimizes a query to generate an evaluation plan, which when executed by the query evaluation engine, produces the query results.
- In so doing the QP relies on the storage manager to regulate access to buffers in primary memory and to manage its movement from and to secondary memory.
- Important subsystems also include the concurrency and the transaction managers.
- The main classes of users of DBMSs are causal/occasional users, sophisticated users, application programmers and administrators.
- They rely on tools and APIs provided/exposed by the DBMS.
- The main kinds of database architectures used today are client-server, parallel and distributed.
- Each of these have variants (e.g., finely- or coarsely-grained parallel, WAN- or LAN-distributed).

## The End

- Thanks for attending the lectures.
- I hope you have enjoyed the course.