Introduction to Database Systems

2023-Fall

Database System Architectures

Outline

- Centralized Database Systems
- Server System Architectures
- Parallel Systems
- Distributed Systems
- Network Types

Centralized Database Systems

- Run on a single computer system
- **Single-user system**
  - Embedded databases
- **Multi-user systems** also known as **server systems**.
  - Service requests received from client systems
  - Multi-core systems with **coarse-grained parallelism**
    - Typically a few to tens of processor cores
    - In contrast, fine-grained parallelism uses very large number of computers

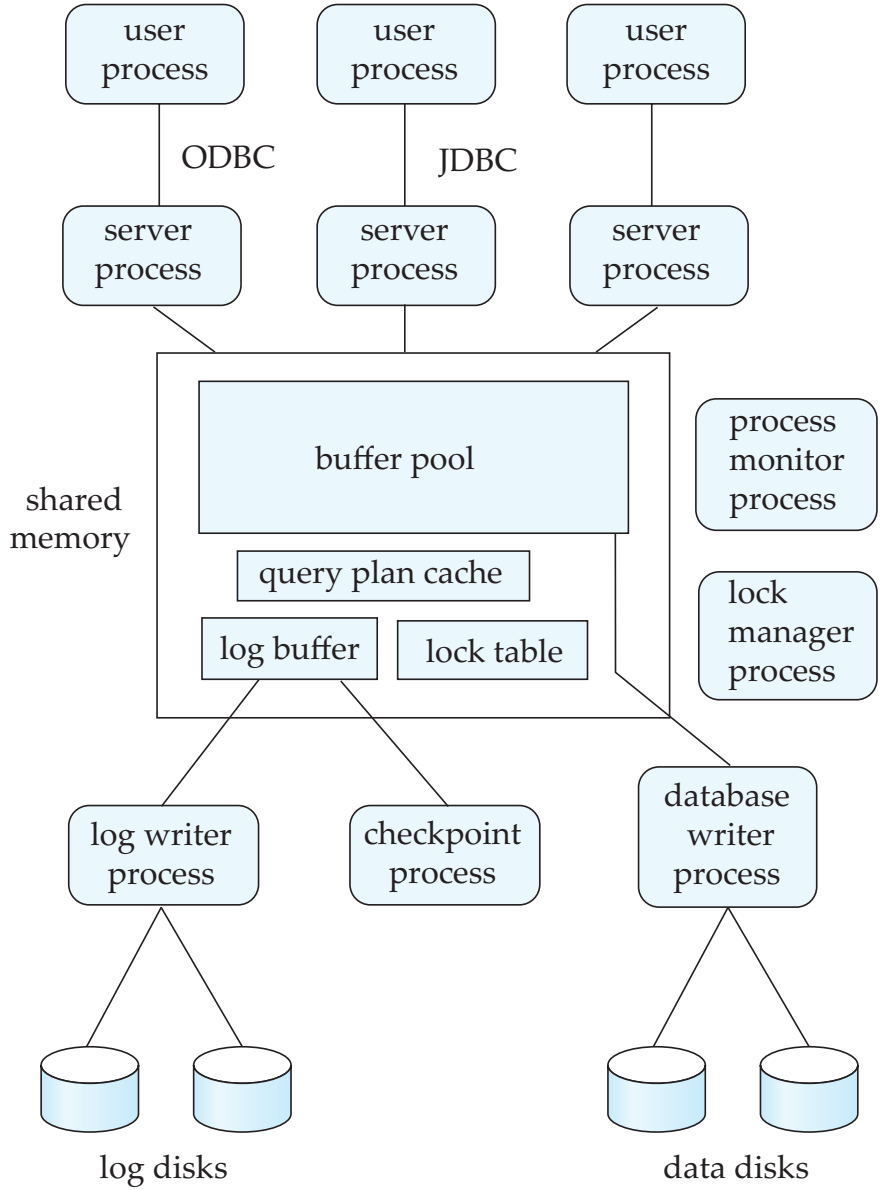
Server System Architecture

- Server systems can be broadly categorized into two kinds:
  - **transaction servers**
    - Widely used in relational database systems, and
  - **data servers**
    - Parallel data servers used to implement high-performance transaction processing systems

Transaction Servers

- Also called **query server** systems or ***SQL server*** systems
  - Clients send requests to the server
  - Transactions are executed at the server
  - Results are shipped back to the client.
- 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.
- Applications typically use ODBC/JDBC APIs to communicate with transaction servers

Transaction System Processes (Cont.)



Transaction Server Process Structure

- A typical transaction server consists of multiple processes accessing data in shared memory
- Shared memory contains shared data
  - Buffer pool
  - Lock table
  - Log buffer
  - Cached query plans (reused if same query submitted again)
- All database processes can access shared memory
- Server processes
  - These receive user queries (transactions), execute them and send results back
  - Processes may be **multithreaded**, allowing a single process to execute several user queries concurrently
  - Typically multiple multithreaded server processes

Transaction Server Process

- ***Database writer process***
  - Output modified buffer blocks to disks continually
- ***Log writer process***
  - Server processes simply add log records to log record buffer
  - Log writer process outputs log records to stable storage.
- ***Checkpoint process***
  - Performs periodic checkpoints
- ***Process monitor process***
  - Monitors other processes, and takes recovery actions if any of the other processes fail
    - E.g. aborting any transactions being executed by a server process and restarting it

Transaction System Processes (Cont.)

- ***Lock manager process***
  - To avoid overhead of inter-process communication for lock request/grant, each database process operates directly on the lock table
    - instead of sending requests to lock manager process
  - Lock manager process still used for deadlock detection
- To ensure that no two processes are accessing the same data structure at the same time, databases systems implement ***mutual exclusion*** using either
  - Atomic instructions
    - Test-And-Set
    - Compare-And-Swap (CAS)
  - Operating system semaphores
    - Higher overhead than atomic instructions

Data Servers/Data Storage Systems

- Data items are shipped to clients where processing is performed
- Updated data items written back to server
- Earlier generation of data servers would operated in units of data items, or pages containing multiple data items
- Current generation data servers (also called data storage systems) only work in units of data items
  - Commonly used data item formats include JSON, XML, or just uninterpreted binary strings

Data Servers/Storage Systems (Cont.)

- **Prefetching**
  - Prefetch items that may be used soon
- **Data caching**
  - Cache coherence
- **Lock caching**
  - Locks can be cached by client across transactions
  - Locks can be **called back** by the server
- **Adaptive lock granularity**
  - **Lock granularity escalation**
    - switch from finer granularity (e.g. tuple) lock to coarser
  - **Lock granularity de-escalation**
    - Start with coarse granularity to reduce overheads, switch to finer granularity in case of more concurrency conflict at server
    - Details in book

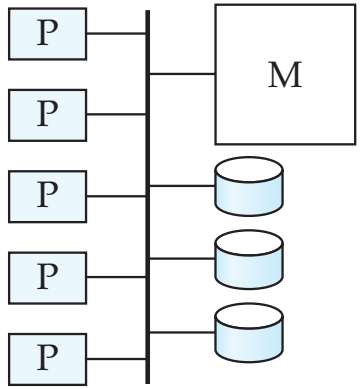
Parallel Systems

- Parallel database systems consist of multiple processors and multiple disks connected by a fast interconnection network.
- Motivation: handle workloads beyond what a single computer system can handle
- High performance **transaction processing**
  - E.g. handling user requests at web-scale
- **Decision support** on very large amounts of data
  - E.g. data gathered by large web sites/apps

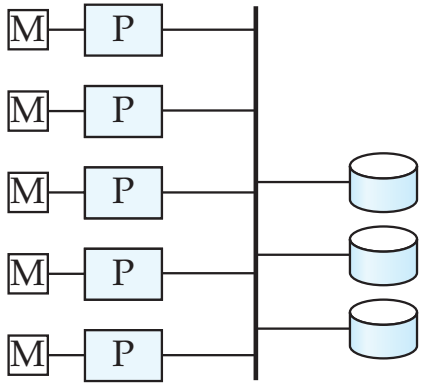
Parallel Systems (Cont.)

- 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.
  - Typically hosted in a **data center**
- 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

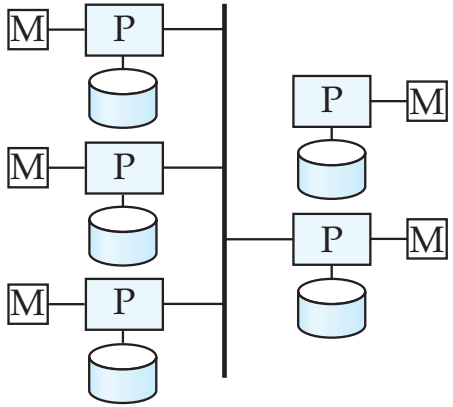
Parallel Database Architectures



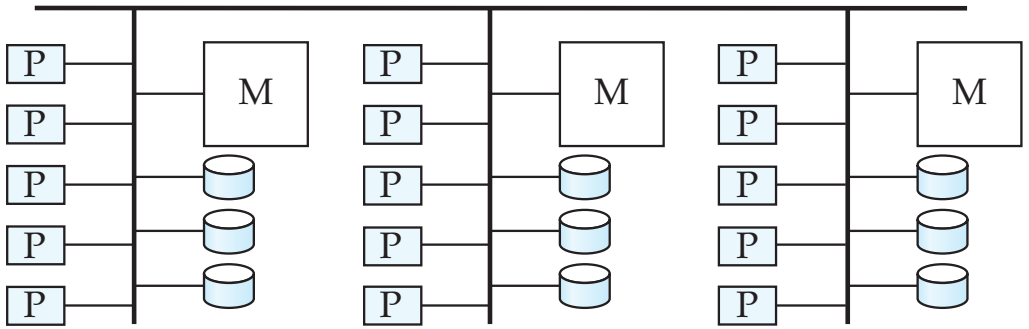
(a) shared memory



(b) shared disk

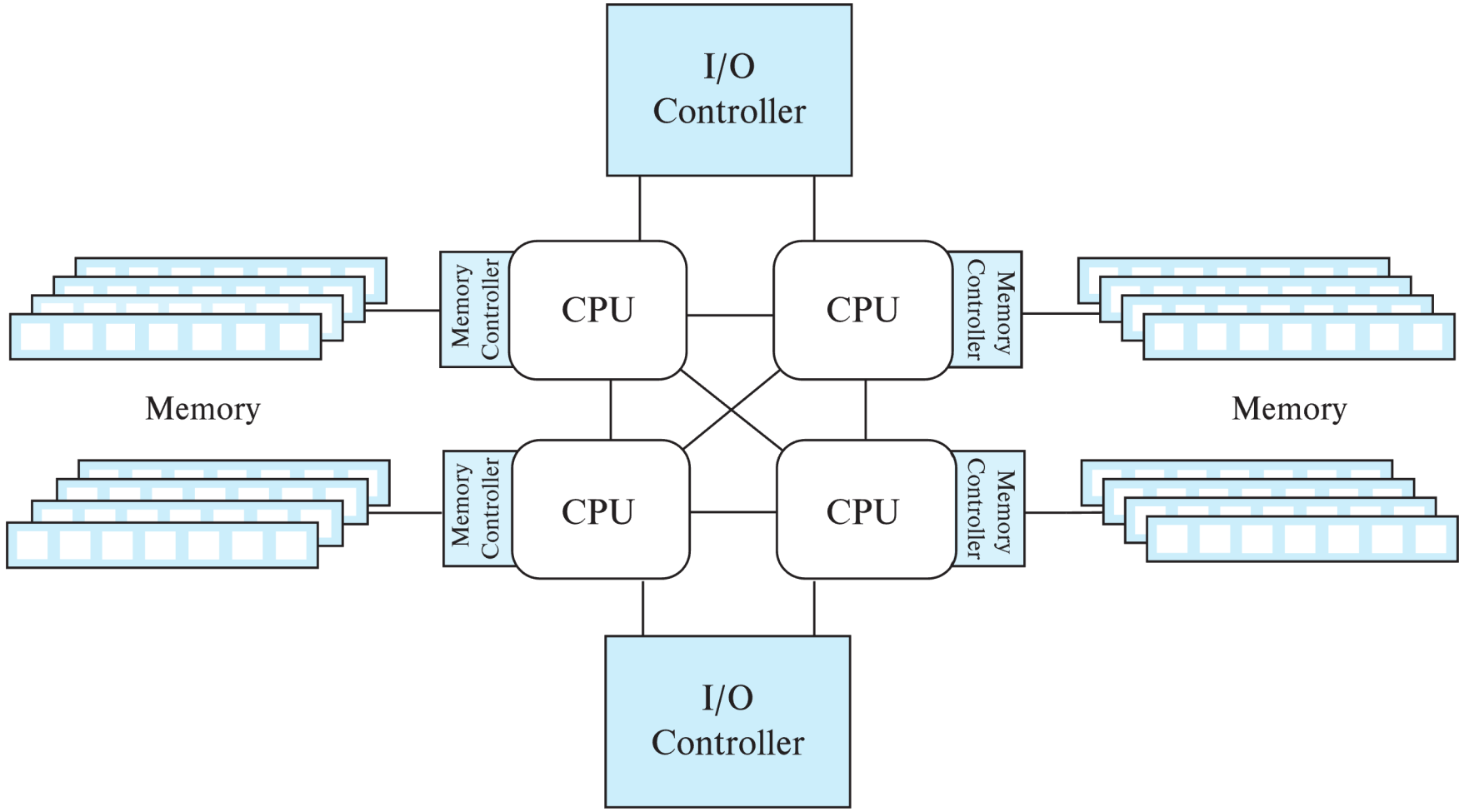


(c) shared nothing

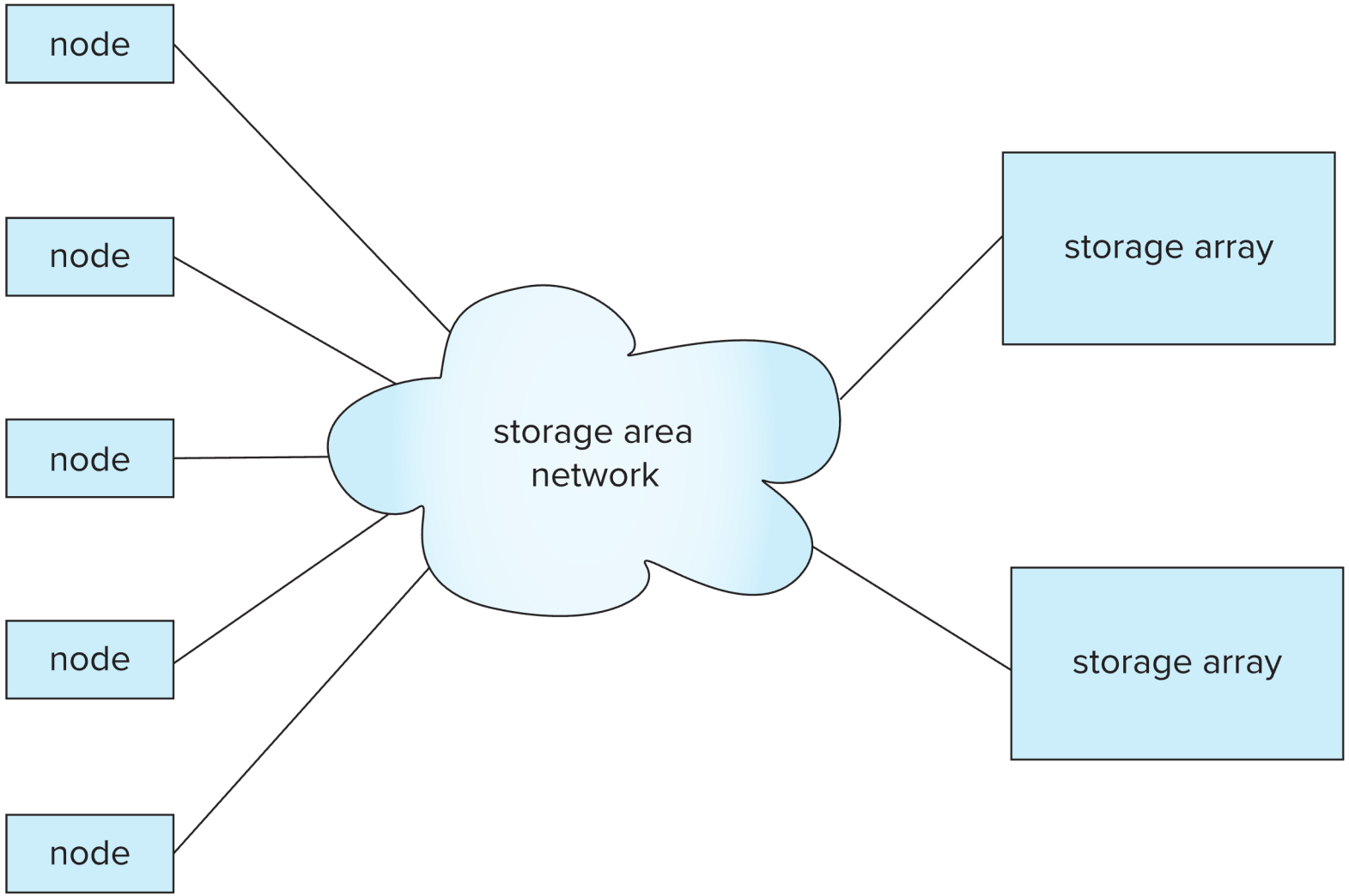


(d) hierarchical

Modern Shared Memory Architecture

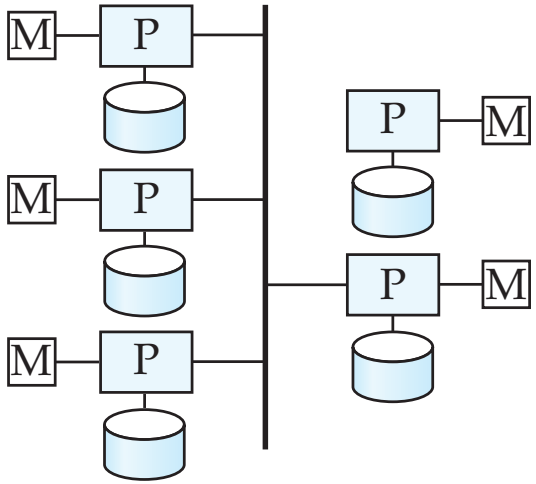


Modern Shared Disk Architectures: via Storage Area Network (SAN)



Shared Nothing

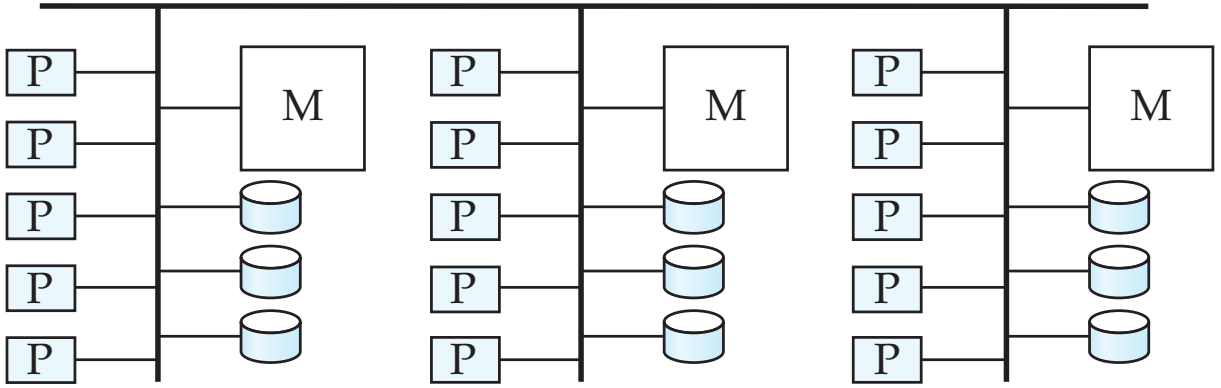
- Node consists of a processor, memory, and one or more disks
- All communication via interconnection network
- Can be scaled up to thousands of processors without interference.
- ***Main drawback***: cost of communication and non-local disk access; sending data involves software interaction at both ends.



(c) shared nothing

Hierarchical

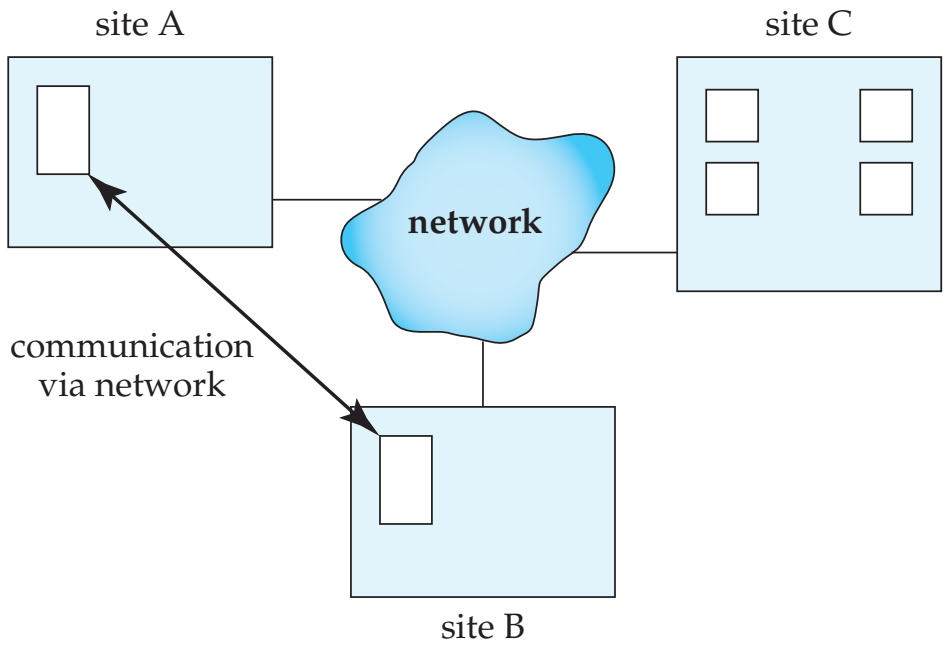
- Combines characteristics of shared-memory, shared-disk, and shared-nothing architectures.
  - Top level is a shared-nothing architecture
    - With each node of the system being a shared-memory system
  - Alternatively, top level could be a shared-disk system
    - With each node of the system being a shared-memory system



(d) hierarchical

Distributed Systems

- Data spread over multiple machines (also referred to as **sites** or **nodes**).
- **Local-area networks (LANs)**
- **Wide-area networks (WANs)**
  - Higher latency



Distributed Databases

- **Homogeneous distributed databases**
  - Same software/schema on all sites, data may be partitioned among sites
  - Goal: provide a view of a single database, hiding details of distribution
- **Heterogeneous distributed databases**
  - Different software/schema on different sites
  - Goal: integrate existing databases to provide useful functionality
- Differentiate between **local transactions** and **global transactions**
  - 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.

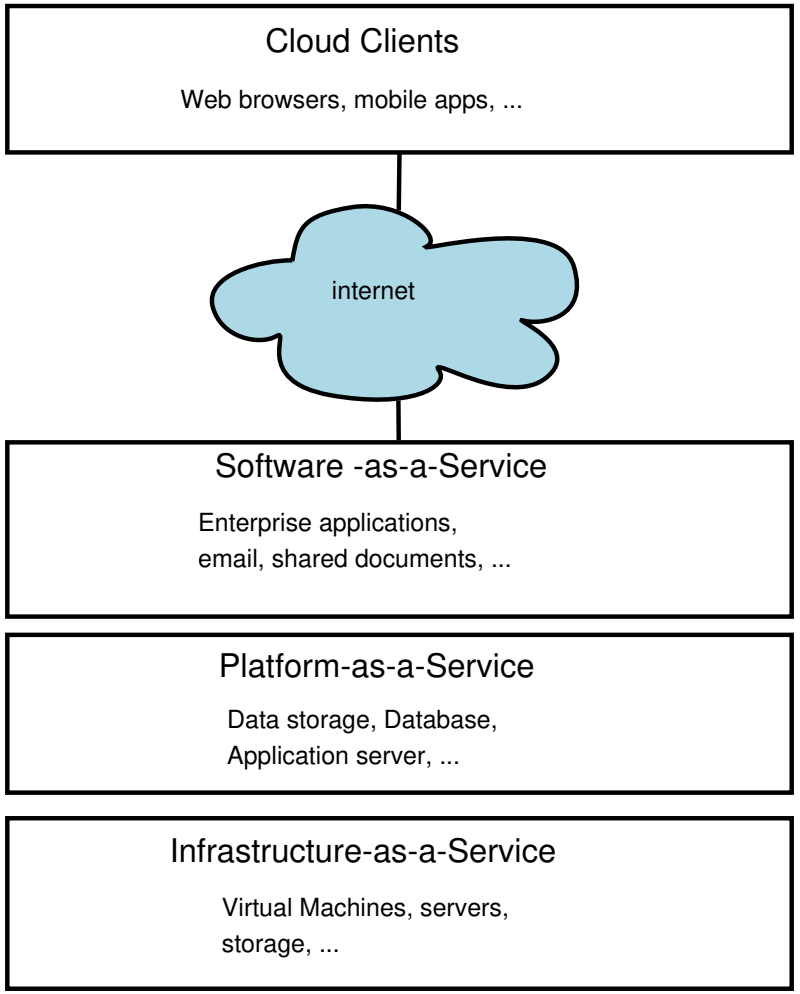
Data Integration and Distributed Databases

- Data integration between multiple distributed databases
- Benefits:
  - 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.

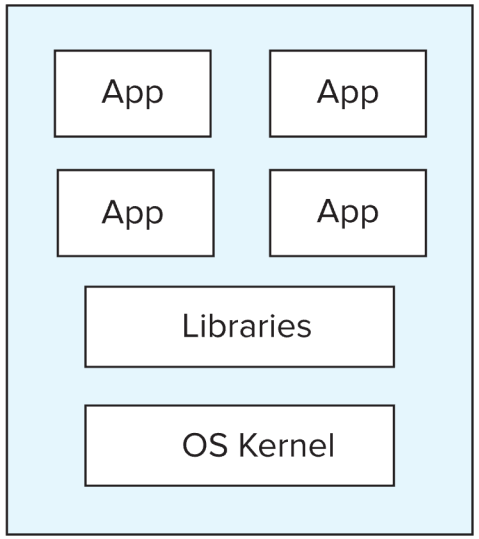
Cloud Based Services

- Cloud computing widely adopted today
  - On-demand provisioning and **elasticity**
    - ability to scale up at short notice and to release of unused resources for use by others
- Infrastructure as a service
  - Virtual machines/real machines
- Platform as a service
  - Storage, databases, application server
- Software as a service
  - Enterprise applications, emails, shared documents, etc,
- Potential drawbacks
  - Security
  - Network bandwidth

Cloud Service Models

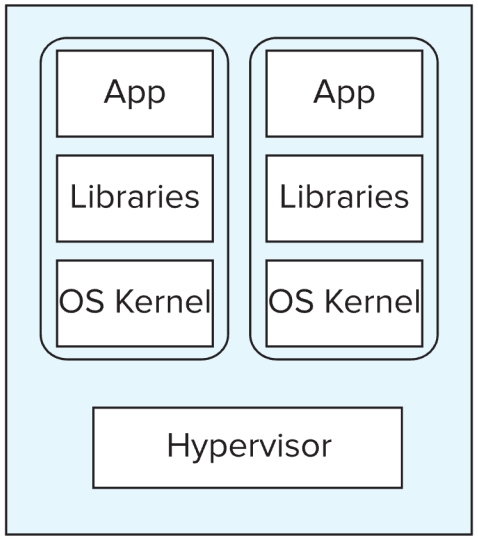


Application Deployment Alternatives



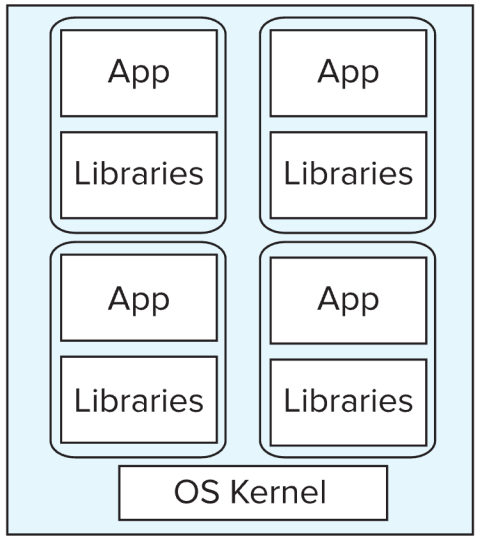
a) Multiple applications on a single machine

Individual Machines



b) Each application running on its own VM, with multiple VMs running in a machine

Virtual Machines
(e.g. VMWare, KVM, ..)



c) Each application running in its own container, with multiple containers running in a machine

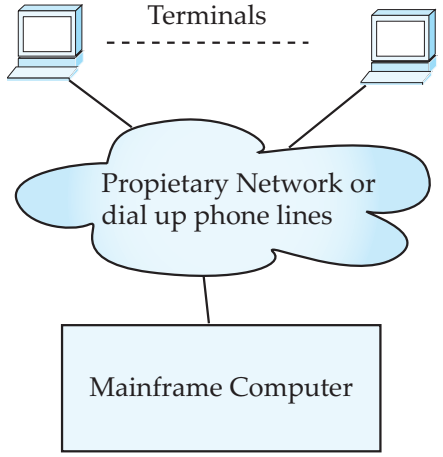
Containers
(e.g. Docker)

- Services
- Microservice Architecture
  - Application uses a variety of services
  - Service can add or remove instances as required
- Kubernetes supports containers, and microservices

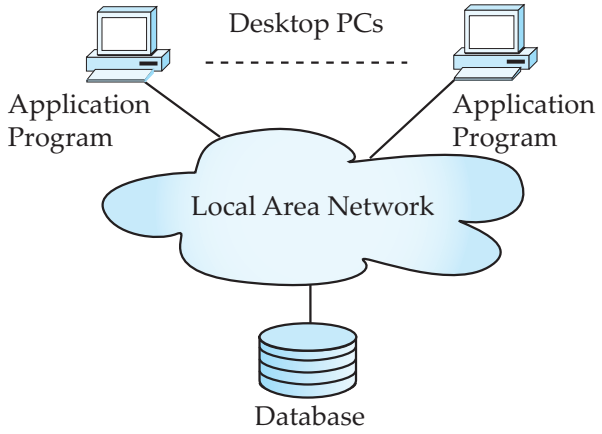
Application Development

Application Architecture Evolution

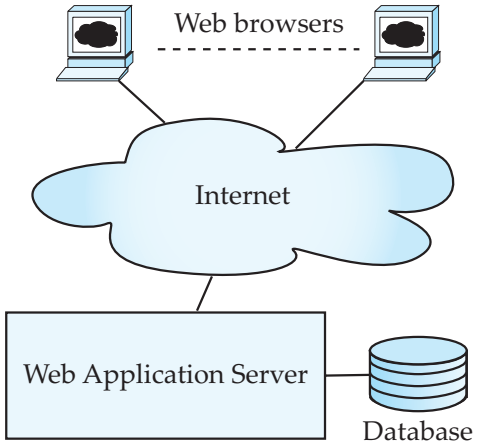
- Three distinct era's of application architecture
  - Mainframe (1960's and 70's)
  - Personal computer era (1980's)
  - Web era (mid 1990's onwards)
  - Web and Smartphone era (2010 onwards)



(a) Mainframe Era



(b) Personal Computer Era



(c) Web era

Web Interface

Web browsers have become the de-facto standard user interface to databases

- Enable large numbers of users to access databases from anywhere
- Avoid the need for downloading/installing specialized code, while providing a good graphical user interface
  - Javascript, Flash and other scripting languages run in browser, but are downloaded transparently
- Examples: banks, airline and rental car reservations, university course registration and grading, an so on.

Sample HTML Source Text

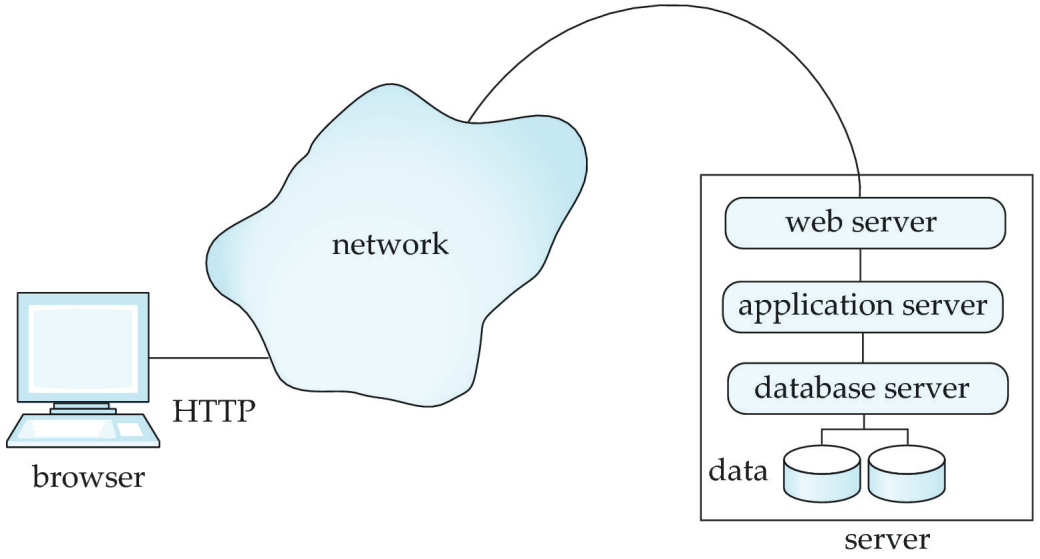
```
<html>
<body>
 <table border>
   <tr> <th>ID</th> <th>Name</th> <th>Department</th> </tr>
   <tr> <td>00128</td> <td>Zhang</td> <td>Comp. Sci.</td> </tr>
   ….
 </table>
  <form action="PersonQuery" method=get>
   Search for:
     <select name="persontype">
        <option value="student" selected>Student </option>
        <option value="instructor"> Instructor </option>
     </select> <br>
   Name: <input type=text size=20 name="name">
   <input type=submit value="submit">
 </form>
</body> </html>
```

Display of Sample HTML Source

ID	Name	Department
00128	Zhang	Comp. Sci.
12345	Shankar	Comp. Sci.
19991	Brandt	History

Search for: Student
Name:
submit

Three-Layer Web Architecture



Application Architectures

Application Architectures

- Application layers
  - Presentation or user interface
    - ***model-view-controller (MVC)*** architecture
      - **model**: business logic
      - **view**: presentation of data, depends on display device
      - **controller**: receives events, executes actions, and returns a view to the user
  - ***business-logic*** layer
    - provides high level view of data and actions on data
      - often using an object data model
    - hides details of data storage schema
  - ***data access*** layer
    - interfaces between business logic layer and the underlying database
    - provides mapping from object model of business layer to relational model of database

Business Logic Layer

- Provides abstractions of entities
  - E.g., students, instructors, courses, etc
- Enforces ***business rules*** for carrying out actions
  - E.g., student can enroll in a class only if she has completed prerequsites, and has paid her tuition fees
- Supports ***workflows*** which define how a task involving multiple participants is to be carried out
  - E.g., how to process application by a student applying to a university
  - Sequence of steps to carry out task
  - Error handling
    - E.g. what to do if recommendation letters not received on time
  - Workflows discussed in Section 26.2

Object-Relational Mapping

- Allows application code to be written on top of object-oriented data model, while storing data in a traditional relational database
  - Alternative: implement object-oriented or object-relational database to store object model
    - Has not been commercially successful
- Schema designer has to provide a mapping between object data and relational schema
  - E.g., Java class *Student* mapped to relation *student,* with corresponding mapping of attributes
  - An object can map to multiple tuples in multiple relations
- Application opens a session, which connects to the database
- Objects can be created and saved to the database using session.save(object)
  - Mapping used to create appropriate tuples in the database
- Query can be run to retrieve objects satisfying specified predicates

Web Services

- Allow data on Web to be accessed using remote procedure call mechanism
- Two approaches are widely used
  - **Representation State Transfer (REST)**: allows use of standard HTTP request to a URL to execute a request and return data
    - Returned data is encoded either in XML, or in **JavaScript Object Notation (JSON)**
  - **Big Web Services**:
    - Uses XML representation for sending request data, as well as for returning results
    - Standard protocol layer built on top of HTTP
- **Disconnected Operations**
  - Tools for applications to use the Web when connected, but operate locally when disconnected from the Web
  - Make use of HTML5 local storage

Rapid Application Development

- A lot of effort is required to develop Web application interfaces
  - More so, to support rich interaction functionality associated with Web 2.0 applications
- Several approaches to speed up application development
  - Function library to generate user-interface elements
  - Drag-and-drop features in an IDE to create user-interface elements
  - Automatically generate code for user interface from a declarative specification
- Above features have been in used as part of ***rapid application development (RAD)*** tools even before advent of Web

Application Security

Application-Level Authorization

- Current SQL standard does not allow fine-grained authorization such as "students can see their own grades, but not other's grades"
  - Problem 1: Database has no idea who are application users
  - Problem 2: SQL authorization is at the level of tables, or columns of tables, but not to specific rows of a table
- One workaround: use views such as

  create view *studentTakes* **as**
  select *
  from *takes*
  where *takes.ID = syscontext.user_id*()
  - where syscontext.user_id() provides end user identity
    - End user identity must be provided to the database by the application
  - Having multiple such views is cumbersome

Application-Level Authorization (Cont.)

- Currently, authorization is done entirely in application
- Entire application code has access to entire database
  - Large surface area, making protection harder
- Alternative: **fine-grained (row-level) authorization** schemes
  - Extensions to SQL authorization proposed but not currently implemented
  - Oracle Virtual Private Database (VPD) allows predicates to be added transparently to all SQL queries, to enforce fine-grained authorization
    - E.g., add *ID= sys_context.user_id()* to all queries on student relation if user is a student

Audit Trails

- Applications must log actions to an audit trail, to detect who carried out an update, or accessed some sensitive data
- Audit trails used after-the-fact to
  - Detect security breaches
  - Repair damage caused by security breach
  - Trace who carried out the breach
- Audit trails needed at
  - Database level, and at
  - Application level