Note:

There are many different types of clients and servers that can be part of a network, and the distinctions between them have become a bit more complex over time. Generally speaking, there are four types of computers that are commonly used as servers:

1. A **mainframe** is a very large general-purpose computer (usually costing millions of dollars) that is capable of performing an immense number of simultaneous functions, supporting an enormous number of simultaneous users, and storing huge amounts of data.
2. A **personal computer** is the type of computer we use. Personal computers used as servers can be small.
3. A **cluster** is a group of computers linked together so that they act as one computer. Requests arrive at the cluster (e.g., Web requests) and are distributed among the computers so that no one computer is overloaded. Each computer is separate, so that if one fails, the cluster simply bypasses it. Clusters are more complex than single servers because work must be quickly coordinated and shared among the individual computers. Clusters are very scalable because one can always add one more computer to the cluster.
4. A **virtual server** is one computer that acts as **several servers**. Using special software, several operating systems are installed on the same physical computer so that one physical computer appears as several different servers to the network. These virtual servers can perform the same or separate functions (e.g., a printer server, Web server, file server). This improves **efficiency** (when each server is not fully used, there is no need to buy separate physical computers) and may improve **effectiveness** (if one virtual server crashes, it does not crash the other servers on the same computer).

There are five commonly used types of clients:

1. A **personal computer** is the most common type of client today. This includes desktop and laptop computers, as well as Tablet PCs that enable the user to write with a pen-like stylus instead of typing on a keyboard.
2. A **terminal** is a device with a monitor and keyboard but no central processing unit (CPU). **Dumb terminals,** so named because they do not participate in the processing of the data they display, have the bare minimum required to operate as input and output devices (a TV screen and a keyboard). In most cases when a character is typed on a dumb terminal, it transmits the character through the circuit to the server for processing. Every keystroke is processed by the server, even simple activities such as the up arrow.
3. A **network computer** is designed primarily to communicate using Internet-based standards (e.g., HTTP, Java) but has no hard disk. It has only limited functionality.
4. A **transaction terminal** is designed to support specific business transactions, such as the automated teller machines (ATM) used by banks. Other examples of transaction terminals are point-of-sale terminals in a supermarket.
5. **A handheld computer, Personal Digital Assistant (PDA), or mobile phone** can also be used as a network client.

Host-Based Architectures

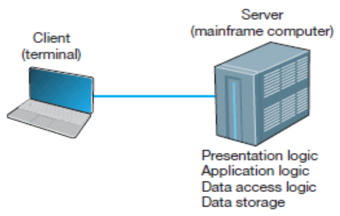

FIGURE 2.1 Host-based architecture

The very first data communications networks developed in the 1960s were host-based, with the server (usually a large mainframe computer) performing all four functions. The clients (usually terminals) enabled users to send and receive messages to and from the host computer. The clients merely captured keystrokes, sent them to the server for processing, and accepted instructions from the server on what to display. This very simple architecture often works very well. Application software is developed and stored on the one server along with all data. **Terminal** is an example of host-based application.

There are fundamental problems with host-based networks. First, the server must process all messages. As the demands for more and more network applications grow, many servers become overloaded and unable to quickly process all the users' demands. Prioritizing users' access becomes difficult. Response time becomes slower, and network managers are required to spend increasingly more money to upgrade the server (expensive).

Client-Based Architectures

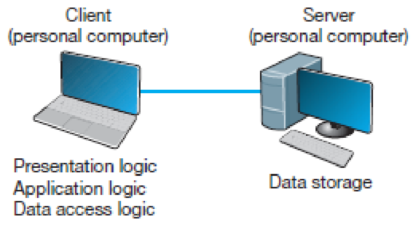

FIGURE 2.2 Client-based architecture

Client-based architectures, the clients are personal computers on a LAN, and the server is usually another personal computer on the same network. The application software on the client computers is responsible for the presentation logic, the application logic, and the data access logic; the server simply stores the data.

This simple architecture often works very well. If you've ever used a word processor and stored your document file on a server (or written a program in Visual Basic or C that runs on your computer but stores data on a server), it is that you've used a client-based architecture.

The fundamental problem in client-based networks is that all data on the server must travel to the client for processing. For example, suppose the user wishes to display a list of all employees with company life insurance. All the data in the database (or all the indices) must travel from the server where the database is stored over the network circuit to the client, which then examines each record to see if it matches the data requested by the user. This can overload the network circuits because far more data is transmitted from the server to the client than the client actually needs.

Client-Server Architectures

Most applications written today make use of client-server architecture. Client-server architectures attempt to balance the processing between the client and the server by having both do some of the logics. In these networks, the client is responsible for the presentation logic, whereas the server is responsible for the data access logic and data storage. The application logic may reside on the client, reside on the server, or be split between both.

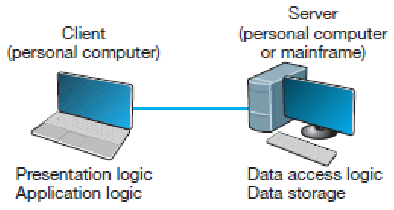

FIGURE 2.3 Two-tier client-server architecture

Fig above shows the simplest case, with the presentation logic and application logic on the client and the data access logic and data storage on the server. In this case, the client software accepts user requests and performs the application logic that produces database requests that are transmitted to the server. The server software accepts the database requests, performs the data access logic, and transmits the results to the client.

The client software accepts the results and presents them to the user. For e.g. when we use a Web browser to get pages from a Web server, we make use of client-server architecture. Also if you've ever written a program that uses SQL to talk to a database on a server, you've used client-server architecture.

For example, if the user requests a list of all employees with company life insurance, the client would accept the request, format it so that it could be understood by the server, and transmit it to the server. On receiving the request, the server searches the database for all requested records and then transmits only the matching records to the client, which would then present them to the user.

Types of Client- Server Architecture

1. **2-Tier Architecture**: A two-tier architecture is a software architecture in which a presentation layer or interface runs on a client, and a data layer or data structure gets stored on a server. Here the presentation logic and business logic is present on server and data (data access logic and data storage) is on a server. The Railway Reservation System, Contact Management System that one can create with the MS-Access etc. are examples of 2 –tier Architecture. Simply a desktop application that requires user to log into an online account is 2-tier architecture.
   Fig is shown above.
2. **3-Tier Architecture:** three-tier architecture uses three sets of computers, as shown in Figure In this case, the software on the client computer is responsible for presentation logic, an application server is responsible for the application logic, and a separate database server is responsible for the data access logic and data storage. I.e. it has a presentation tier, applications tier and a data tier.
   E.g. As a simple example of 3-tier architecture, suppose you are looking to find movie times in your area using a web application. First, the presentation layer displays a web page with some fields for you to enter, like the date you want to view the movie and your zip code. This information is then passed to the application layer, which formats a query and passes it to the database layer. The database system runs the query and returns the results (a list of movies available within your geographic area) to the application layer,

which formats it into a web page. The page is then sent back to the browser, where the presentation layer displays it on a laptop or other device.

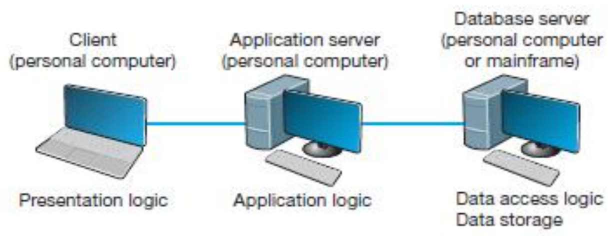

FIGURE 2.4 Three-tier client-server architecture

3. **N-Tier Architecture:** N-tier architecture is also called multi-tier architecture because the software is engineered to have the processing, data management, and presentation functions physically and logically separated. That means that these different functions are hosted on several machines or clusters, ensuring that services are provided without resources being shared and, as such, these services are delivered at top capacity. **N-tier** architecture uses more than three sets of computers. In this case, the client is responsible for presentation logic, a database server is responsible for the data access logic and data storage, and the application logic is spread across two or more different sets of servers.

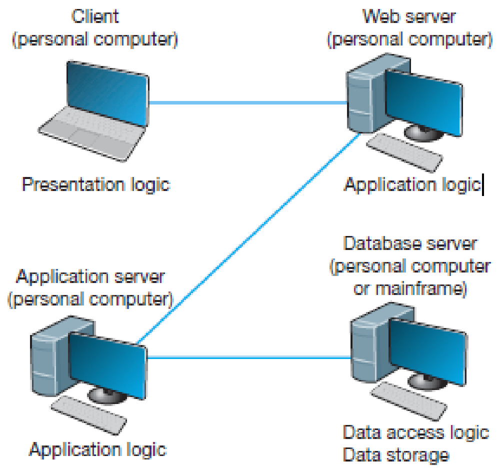

FIGURE 2.5 The *n*-tier client-server architecture

Thin Clients versus Thick Clients Another way of classifying client-server architectures is by examining how much of the application logic is placed on the client computer.

A **thin-client** approach places little or no application logic on the client (e.g., Figure 2.5 n-tier client server architecture). A **thin client** is lightweight computer that relies on the resources of the host computer. It works by connecting with a remote server, where applications and data are stored. They don't have storage capability to store user's data and data is stored at a remote server to which thin client is connected to.

A **thick-client** (also called *fat-client*) approach places all or almost all of the application logic on the client (e.g., Figure 2.3 two tier architecture). It relies lightly upon the server and the majority of data processing is performed by thick clients.

Thin clients are much easier to manage. If an application changes, only the server with the application logic needs to be updated. With a thick client, the software on all of the clients would need to be updated. Also thin clients are more secured than thick clients.
Thin-client architectures are the future,

Peer-to-Peer Architectures: With a P2P architecture, all computers act as both a client and a server. All computers perform all four functions: presentation logic, application logic, data access logic, and data storage. With a P2P file sharing application, a user uses the presentation, application, and data access logic installed on his or her computer to access the data stored on another computer in the network. With a P2P application sharing network (e.g., grid computing such as seti.org), other users in the network can use others' computers to access application logic as well. The advantage of P2P networks is that the data can be installed anywhere on the network. They spread the storage throughout the network, even globally, so they can be very resilient to the failure of any one computer. The challenge is finding the data. Security is a major concern in most P2P networks, so P2P architectures are not commonly used in organizations, except for specialized computing needs (e.g., grid computing **Grid computing** is a group of computers physically connected (over a network or with Internet) to perform a dedicated tasks together, such as analyzing e-commerce data and solve a complex problem).

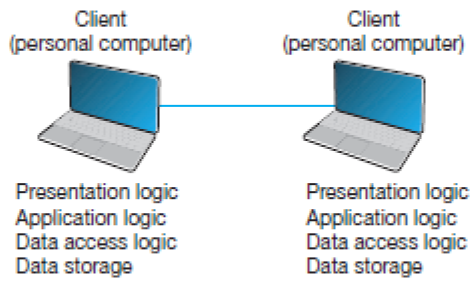

FIGURE 2.7 Peer-to-peer architecture

Choosing Architectures:
Each of the above architectures has certain costs and benefits, so how to choose the "right" architecture? In many cases, the architecture is simply a given; the organization has a certain architecture, and one simply has to use it. In other cases, the organization is acquiring new equipment and writing new software and has the opportunity to develop a new architecture, at least in some part of the organization.
Almost all new applications today are client-server applications. Client-server architectures provide the best **scalability**, the ability to increase (or decrease) the capacity of the servers to meet changing needs. For example, we can easily add or remove application servers or database servers depending on whether we need more or less capacity for application software or database software and storage.
Client-server architectures are also the most **reliable**. We can use multiple servers to perform the same tasks, so that if one server fails, the remaining servers continue to operate and users don't notice problems.
Finally, client-server architectures are usually the **cheapest** because many tools exist to develop them. And lots of client-server software exists for specific parts of applications so we can more quickly buy parts of the application we need. For example, no one writes
Shopping Carts anymore; it's cheaper to buy a Shopping Carts software application and put it on an application server than it is to write on own.

Client-server architectures also enable **cloud computing**. With thin-client client-server architectures, all software resides on a server, whether it is a Web server, application server, or database server. Normally, these servers would be owned and operated by the company itself, but it is not necessary—a server can be anywhere on the Internet and still provide its services to the clients and servers that need them**. Gmail** is a good example of **cloud computing**. Gmail is available to individuals for free, but Google has made the same service available to companies for a fee. Companies pay $50 per user per year for private email service using their company name (e.g. pratiksha@nccs.edu.np). Many companies have determined that it is cheaper to pay for Gmail than to buy and operate their own email servers.
Client server architecture also enables **server virtualization**, Server virtualization means to install many virtual or logical servers on the same physical computer. ((Server virtualization combines many separate logical servers onto the same physical computer, but keeps each of the servers separate so that if one has problems, it does not affect the other servers running on the same physical computer. This way the company is not wasting money to buy and operate many separate computers that sit idle much of the time.)) Server virtualization is also a major part of **green computing.**

Server virtualization requires special software on a physical computer that will host the different logical servers. This software (VMware is one of the leading packages) creates a separate partition on the physical server for each of the logical servers.