CN (IT-3001)

Application Layer

Prof. Amit Jha

School of Electronics Engineering (SOEE)

KIIT Deemed to be University

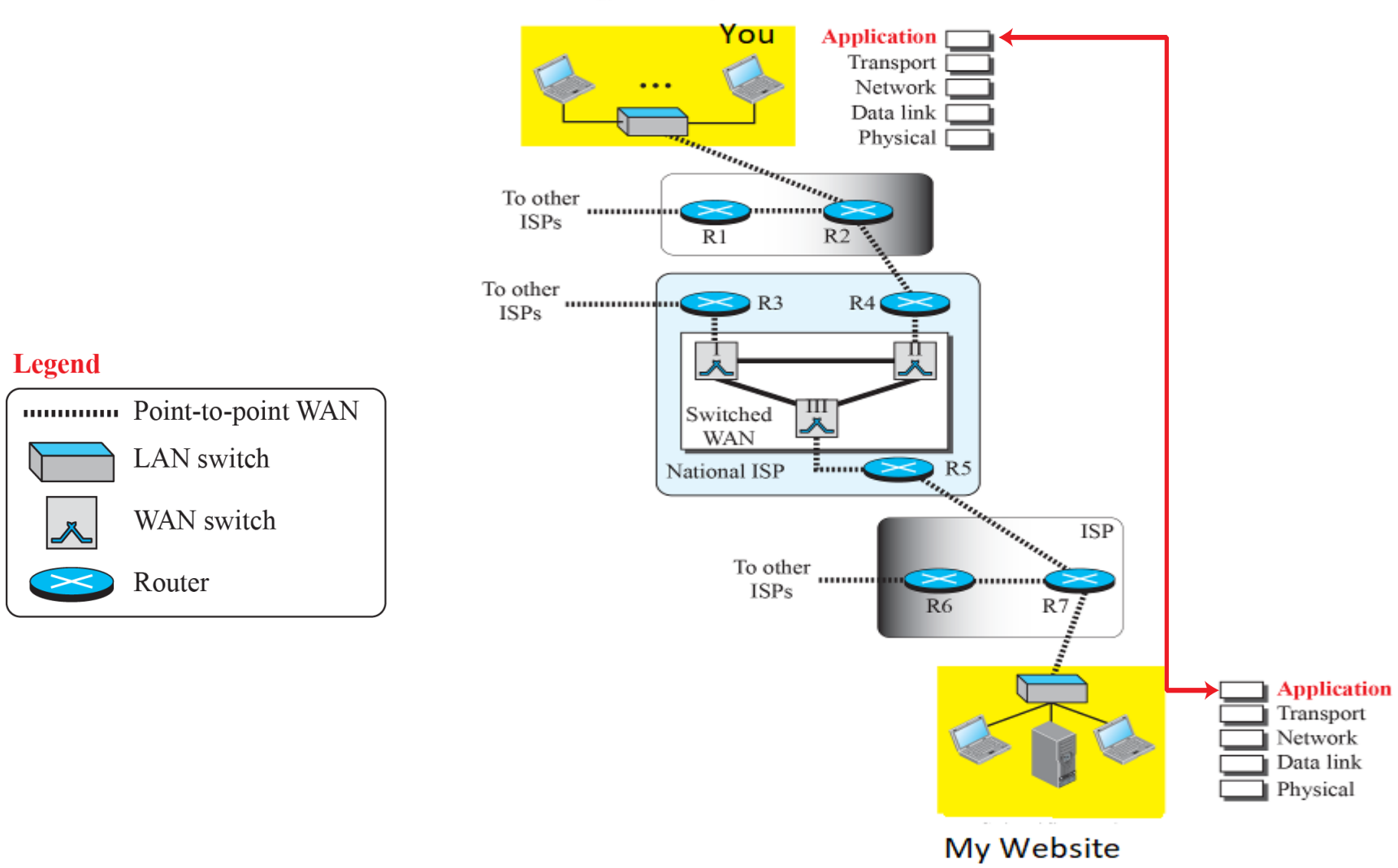
Content

- Responsibility of Application Layer
- Logical Connections at the Application layer
- Different Application Layer Paradigms
    1. Client-Server Paradigm
    2. Peer-to-Peer Paradigm
    3. Hybrid
- Application Programming Interface (API)
- Socket address and its significance
- Transport Layer Keypoints

Responsibility of Application Layer

- Application layer provides **services** to users.
- Communication is provided using a **logical connection**, which means that the two application layers assume that there is an imaginary connection through which they communicate the data.
- The communication at application layer is logical and **not physical**.
- However, actual communication takes place through several devices and several physical channels as shown in the next slide.
- Protocols at this layer do not provide services to any other protocols in the suite; they only receive services form the protocols in the transport layer. This Means
  - Protocols can be **removed** from this layer easily.
  - Also, a new protocols can be **added** to this layer easily as long as the new protocol can utilize the service provided by any of the transport layer protocols→ TCP, UDP and STCP.

Logical Connection at the Application Layer



Application Layer Paradigm

- In the Internet, **two application programs should interact** with each other in order to communicate the data:
  - One running on a computer somewhere in the world;
  - The other running on another computer somewhere else in the world.

- Now, the question arises:
  - Should **both** application programs to be able to **request** services?
  - Should **both** application programs to be able to **receive** services?
  - Should **one** application programs to be able to request services whereas other should provide service or vice-versa?

Application Layer Paradigm

- In the Internet, **two application programs should interact** with each other in order to communicate the data:
  - One running on a computer somewhere in the world;
  - The other running on another computer somewhere else in the world.

Now, the question arises:

- Should **both** application programs to be able to **request** services?
- Should **both** application programs to be able to **receive** services?
- Should **one** application programs to be able to request services whereas other should provide service or vice-versa?

- To answer this question, following three application layer paradigm have been developed:
  1. Traditional Paradigm: Client-Server
  2. New Paradigm: Peer-to-Peer
  3. Hybrid

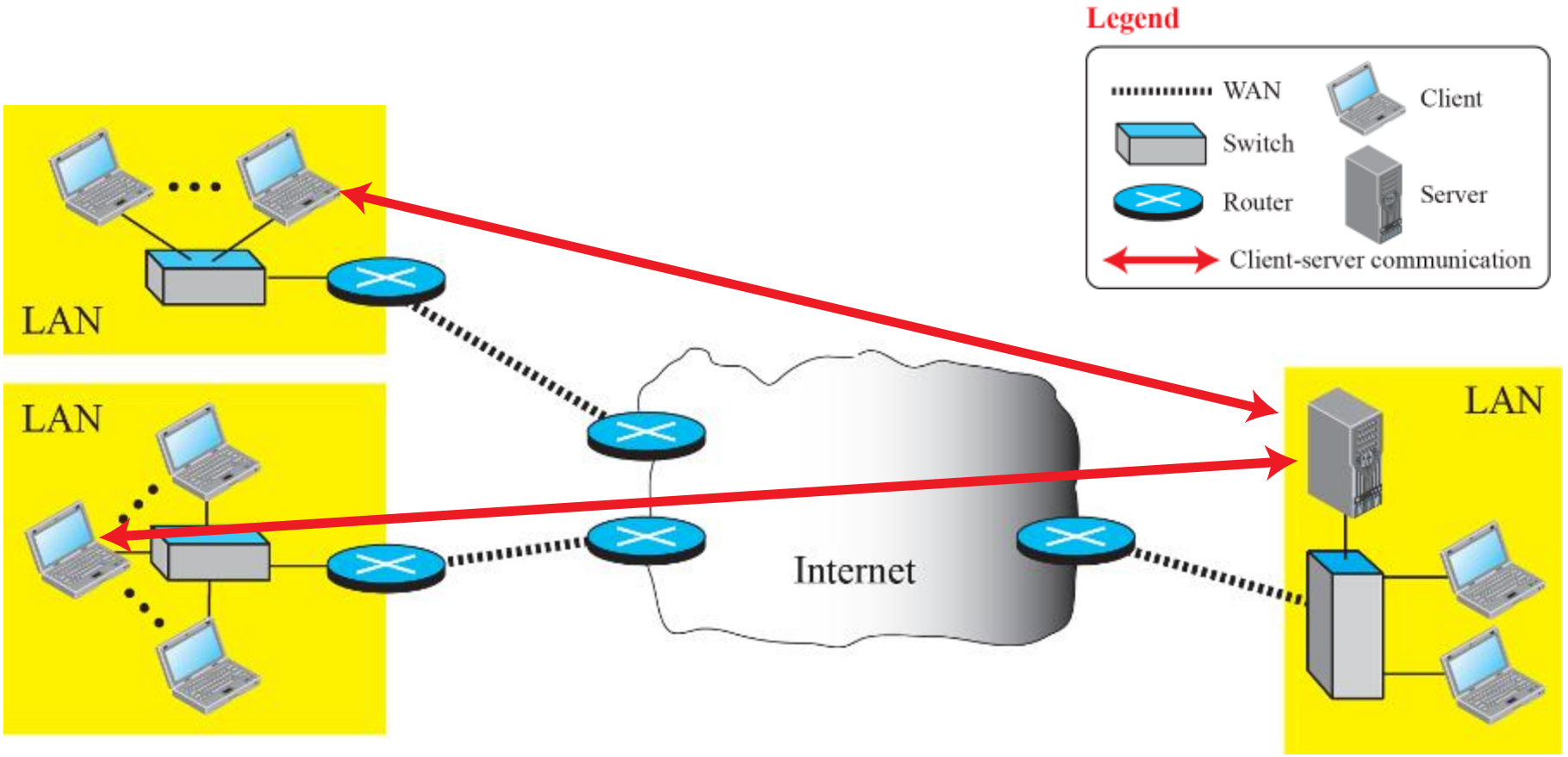
Client-Server Paradigm

- Most popular until few years ago.
- In this we have two systems:

1. **Client:**
   - It request the server for the connection and communication.
   - It does not run all the time, runs only when it needs to receive service.
2. **Server**
   - It runs continuously, waiting for another application program, called client.
   - It runs all the time.

- **Examples:** World Wide Web (www), Hyper Text Transfer Protocol (HTTP), File Transfer Protocol (FTP), Secure Shell (SSH), e-mail, etc.

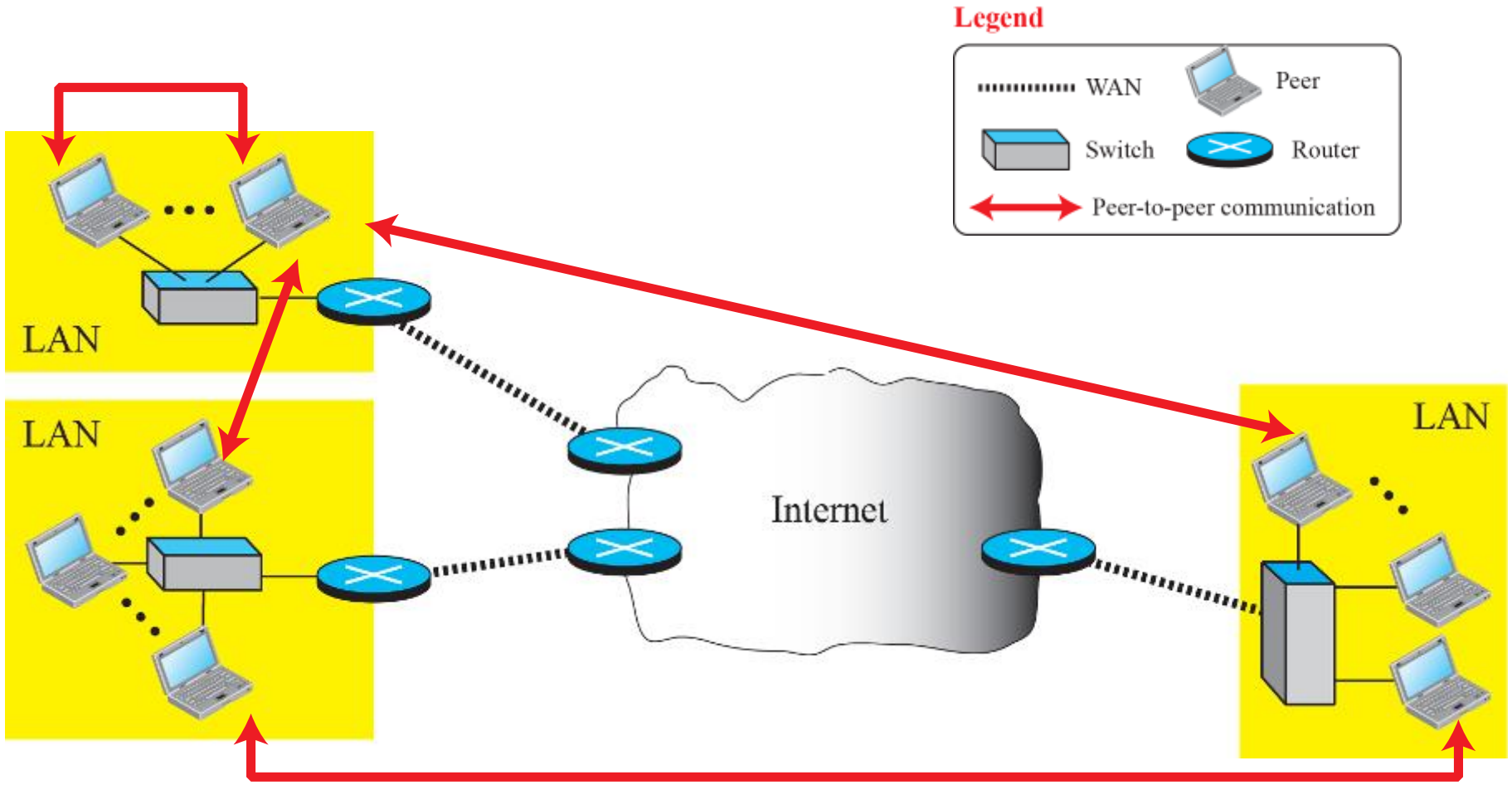
Client-Server Paradigm: Illustration



Peer-to-Peer Paradigm (P2P)

- Most popular now a days.
- **In this,**
- there is no need for a server process running all the time and waiting a client process to connect.
- The responsibility is shared between peers.
- A computer connected to the Internet can either
  - Provide service at one time and receive at another time **or**
  - Can provide and receive the service at the same time.
- **Examples:** Internet telephony, BitTorrent, Skype, IPTV, etc.

Peer-to-Peer Paradigm (P2P): Illustration



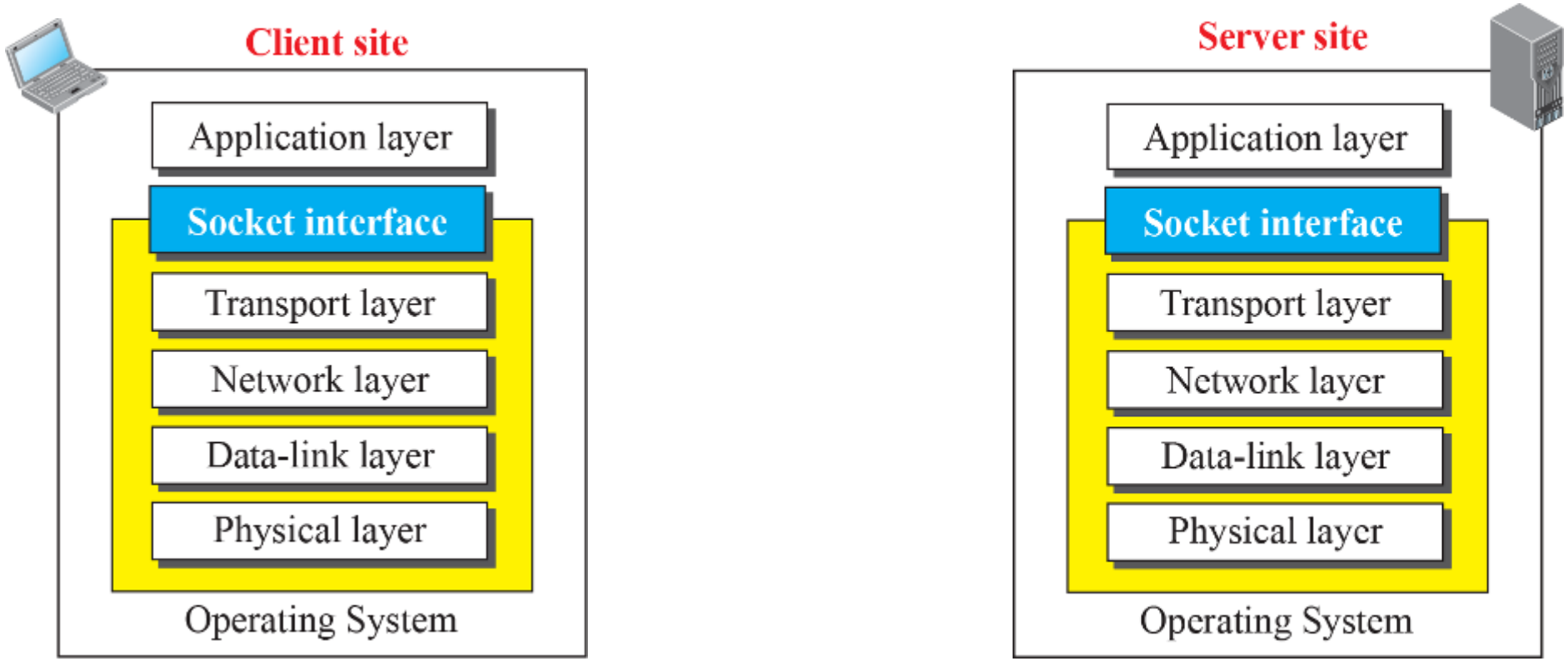
- How can a client process communicate with a server process ?

- How can a client process communicate with a server process ?
- Using **Application Programming Interface** (API).
  - It is used to represent a set of instructions to tell the lowest four layers of TCP/IP suite to open the connection, send and receive the data from the other end, and close the connection.

- **Interface:** An interface in programming is a set of instructions between two entities.
- In this case, one of the entities is the
  - **process at the application** layer and other is the
  - **operating system** that encapsulates the first four layers of the TCP/IP suite.
- In other words, a computer manufacture needs to build the first four layers of the suite in the operating system and include an API.
- In this way, the processes running at the application layer are able to communicate with the operating system when sending or receiving message through the Internet.
- **Few examples:** socket interface, Transport Layer Interface (TLI) and STREAM.

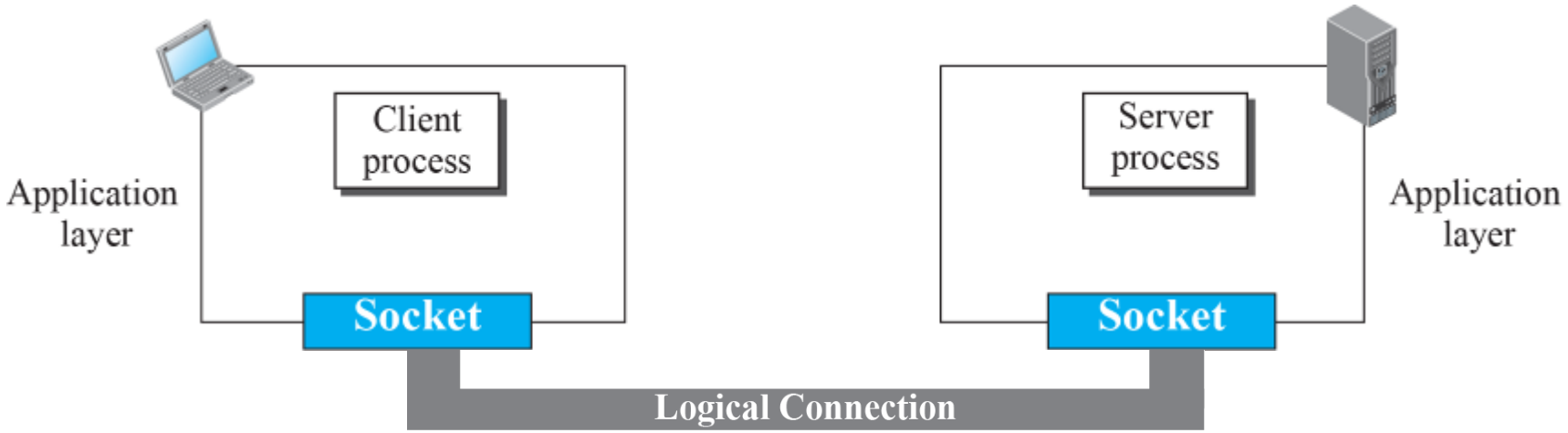
Sockets

- It is a ***data structure*** created and used by the application program.
- Position of the socket interface is shown below:



Sockets

- From the application layer point-of-view, the communication between client process and server process takes place through the sockets, created at two ends as shown below.

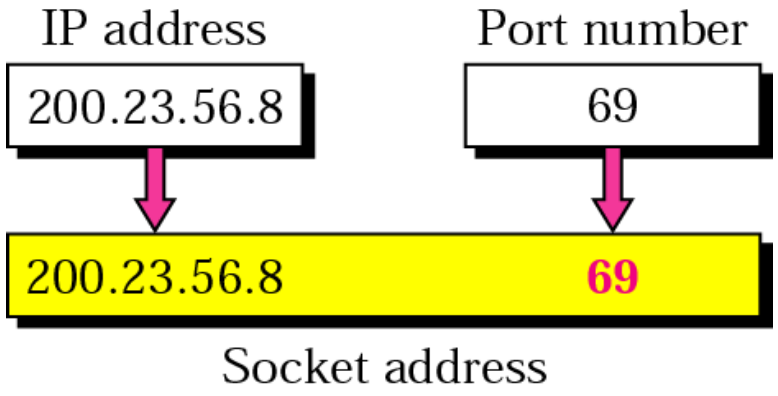


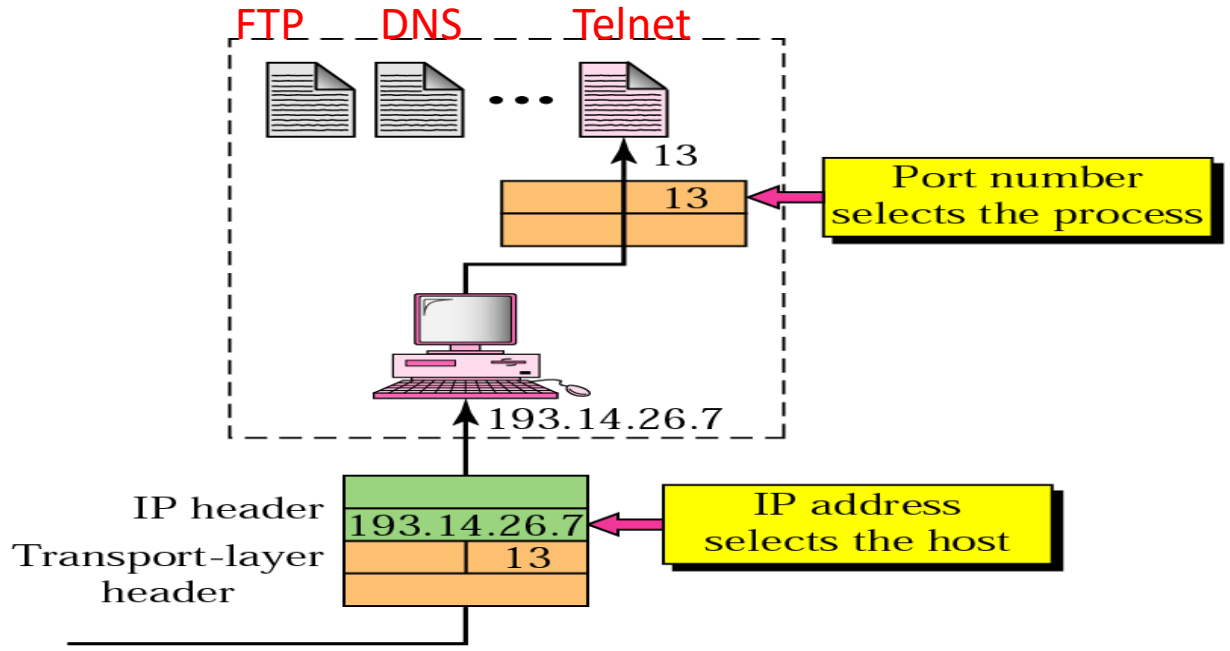
Sockets Address

- In two-way communication, we need a pair of addresses:
  - Local (sender) address
  - Remote (receiver) address
- Note: local address in one direction is remote address in other direction and vice-versa.
- Socket address is combination of **32-bit IP address** (to uniquely identify the device) and **16-bit port number** (to identify the process) as shown below.

Sockets Address

- In two-way communication, we need a pair of addresses:
  - Local (sender) address
  - Remote (receiver) address
- Note: local address in one direction is remote address in other direction and vice-versa.
- Socket address is combination of **32-bit IP address** (to uniquely identify the device) and **16-bit port number** (to identify the process) as shown below.





Sockets Address: Key Points

- ***Port number:*** It is 16-bit integers between 0 to 65,535.
- **Client port number:** Client chooses its port number randomly from 0 to 65,535 using the transport layer software running on the client host. This is called ***ephemeral***(**temporary**) port number.
- **Server port number:** Server process is also defined by a port number. But, its not randomly chosen. If it is random, then client will not know the port number in order to access that server. These are called ***well-known port number***.

Transport Layer Protocols: Only Key Points

- Three protocols are defined at the transport layer:
  1. TCP
     - Connection-oriented
     - Reliable→ supports flow and error control mechanism
     - Supports full-duplex
     - It is a byte-stream oriented protocol
  2. UDP
     - Connection less
     - unreliable→ no flow and error control mechanism
  3. STCP
     - Provides service which is a combination of TCP and UDP.
     - Connection-oriented
     - Reliable
     - But it is not byte-stream oriented protocol. It is message oriented protocol like UDP.