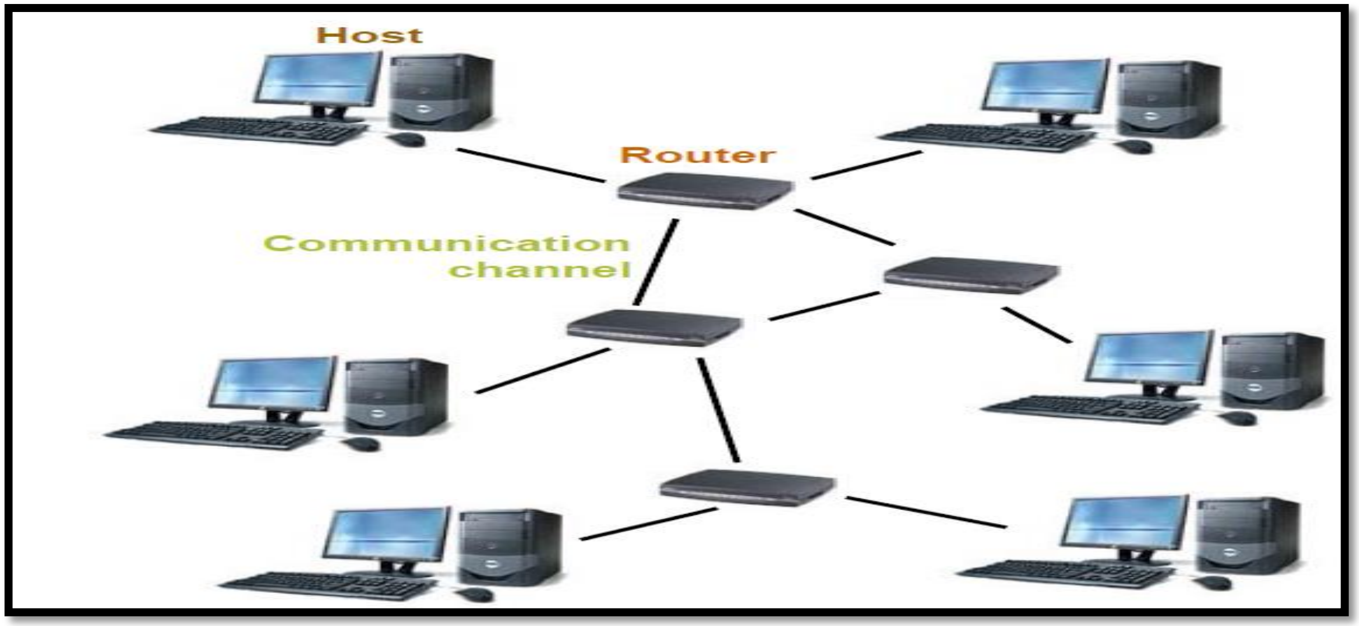
Socket Programming

Background

❖Computer Networks:

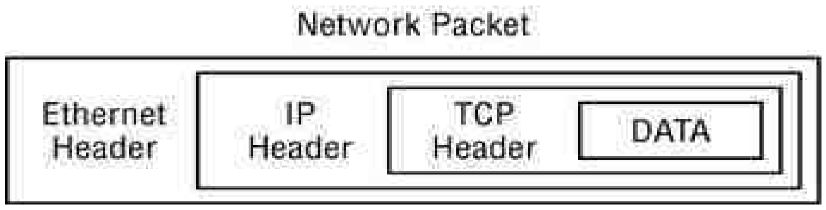
- Consists of **Machines** Interconnected by **communication channels**

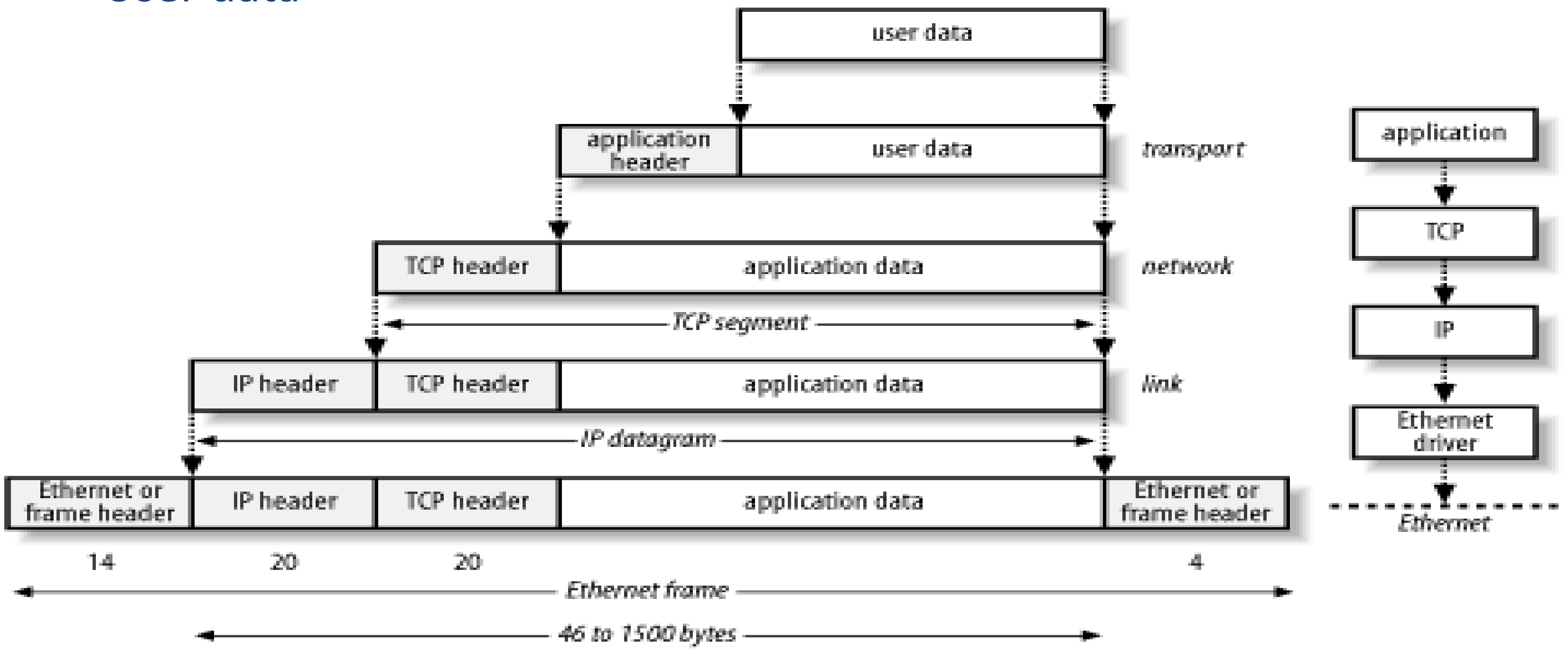


- **Machines** are Hosts and Routers
  - **Hosts** run applications
  - **Routers** forward *information* among communication channels
- **Communication channels** is a means of conveying sequences of bytes from one host to another (Ethernet, dial-up, satellite, etc.)

❖Packets:

- Sequences of **bytes** that are constructed and interpreted by programs
- **A packet contains**
  - Control information:
    - Used by routers to figure out how to forward every packet.
    - e.g. packet destination
  - User data



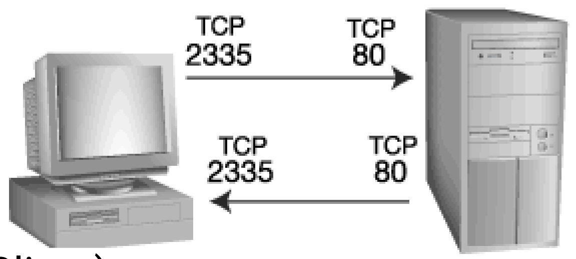


❖Protocol:

- An agreement about the *packets exchanged* by communicating programs and *what they mean*.
- A protocol tells
  - how packets are structured
    - where the distention information is located in the packet
    - how big it is
- Protocols are designed to solve specific problems
  - TCP/IP is such collection of solutions (protocol suite or family):
    - IP, TCP, UDP, DNS, ARP, HTTP, and many more
- How can we access the services provided by TCP/IP suite?
  - **Sockets API.**

❖Addresses:

- Before one program can communicate with another program, it has to tell the network where to find the other program
- In TCP/IP, it takes two piece of information:
  - Internet Address, used by IP (e.g. Company's main phone number)
  - Port Number, interpreted by TCP & UDP (extension number of an individual in the company)



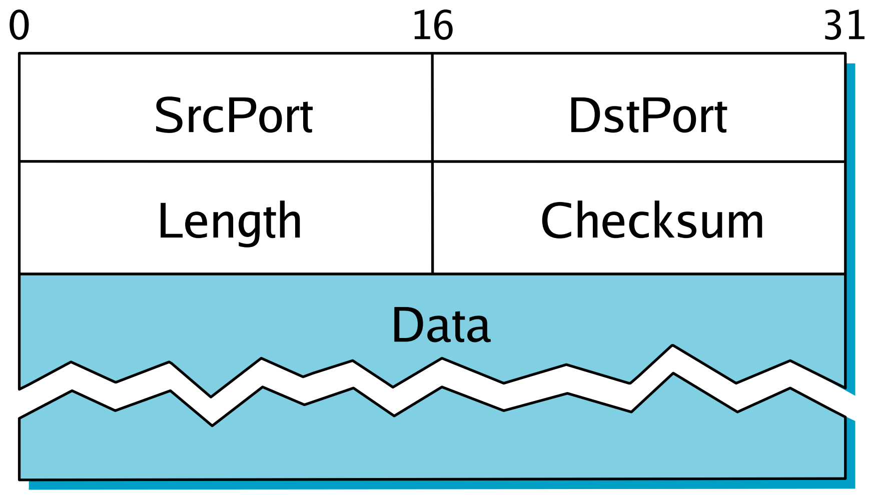
My PC (Client)
192.168.100.10

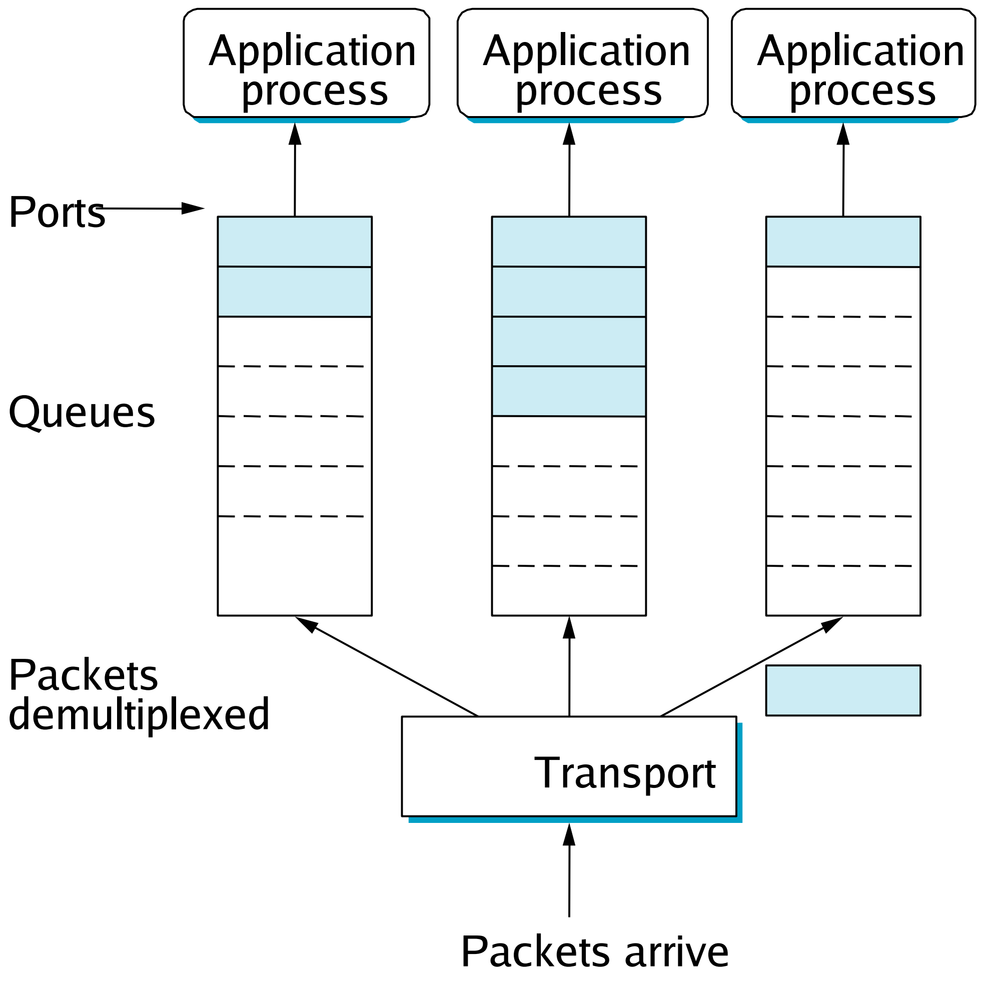
www.myserver.com (Server)
70.122.92.100

FIGURE 1: Sample TCP session

Demultiplexing

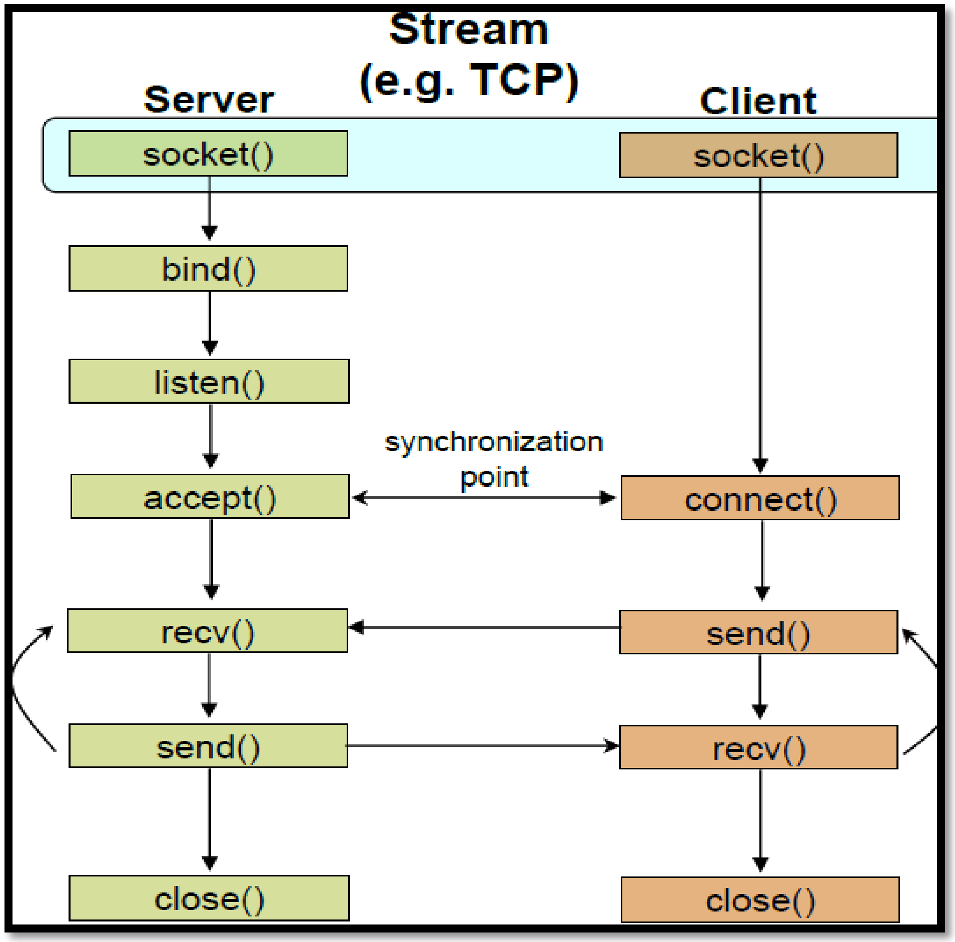
- Convert host-to-host packet delivery service into a process-to-process communication channel





❖Client and server

- **Server**: *passively* waits for and responds to clients
- **Client**: initiates the communication
  - must know the address and the port of the server



- Socket(): endpoint for communication
- Bind(): assign a unique number
- Listen(): wait for a caller
- Connect(): dial a number
  Accept(): receive a call
- Send() and Receive(): Talk
- Close(): Hang up

❑ **Server**

1. Create a TCP socket using socket()
2. Assign a port number to the socket with bind()
3. Tell the system to allow connections to be made to that port using listen()
4. Repeatedly do the following:
   - Call accept() to get a new socket for each client connection
   - communicate with the client using send() and recv()
   - Close the client connection using close()

❑ **Client**

1. Create a TCP socket using socket()
2. Establish a connection to server using connect()
3. communicate using send() and recv()
4. Close connection using close()

❖ Why socket programming?

- To build network applications.
    - Firefox, google chrome, etc.
    - Apache Http server

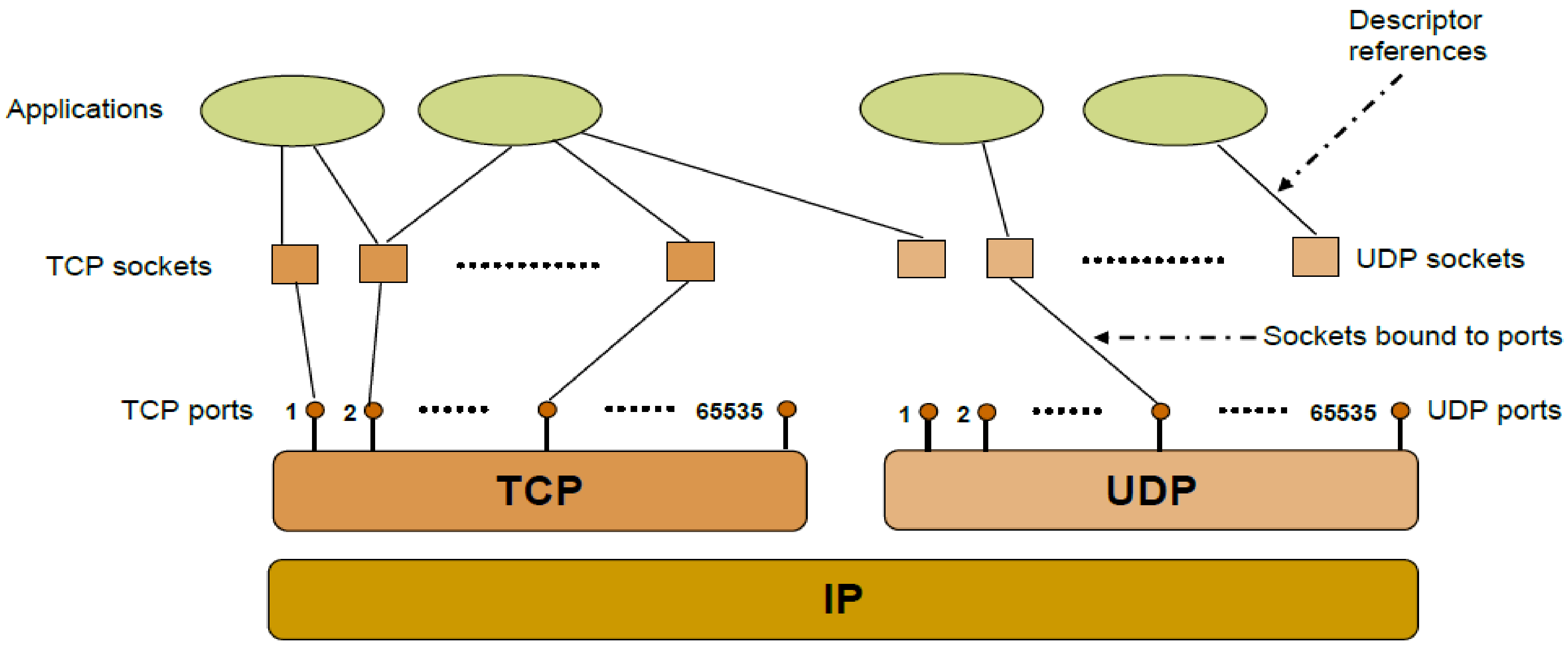
❖ What is a socket?

- It is an abstraction through which an application may send and receive data
- File is an analogy: read (receive) and write (send)

❖ Types of sockets

- Stream sockets (TCP): reliable byte-stream service
- Datagram sockets (UDP): best effort datagram service

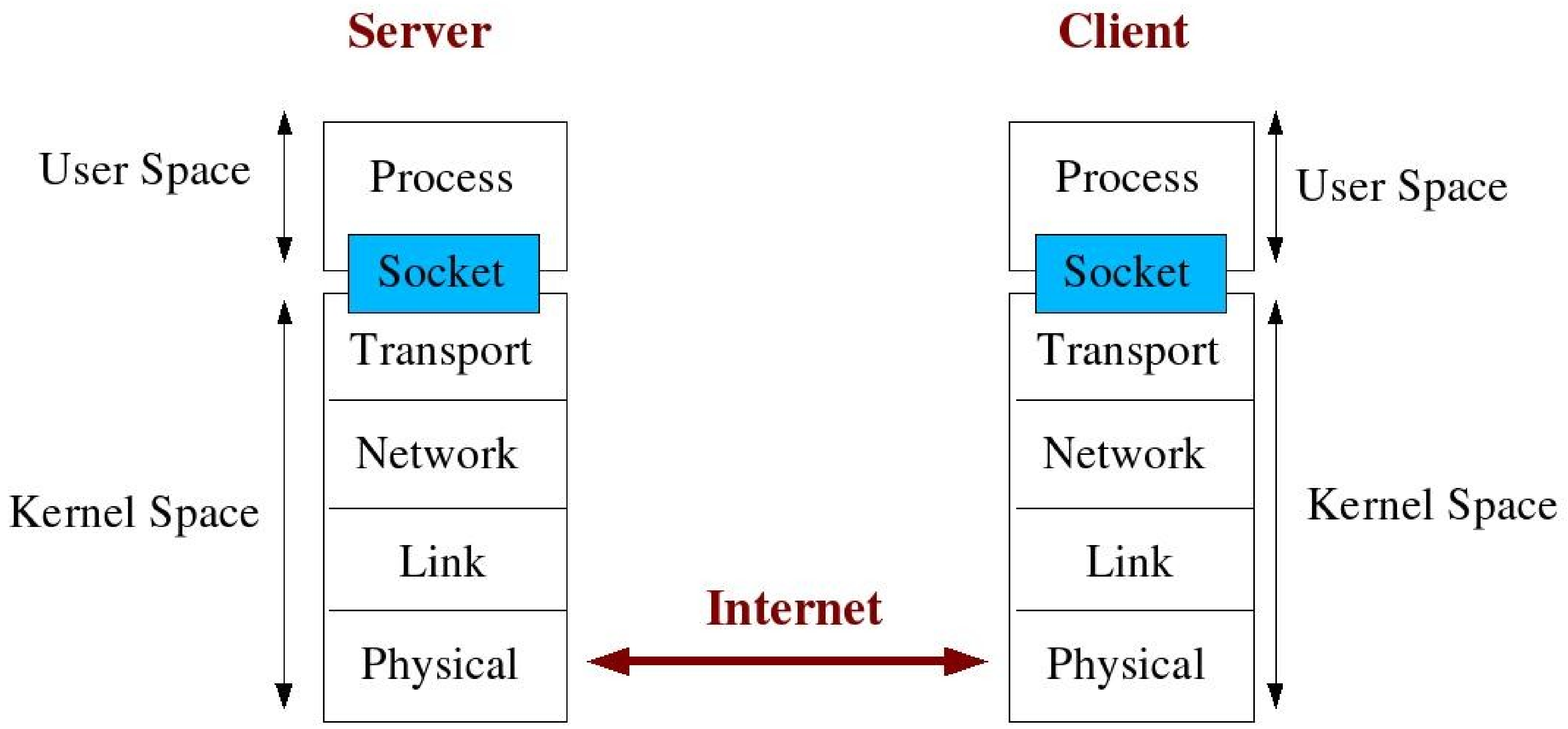
- **What is a socket API?**
  - An interface between application and network
  - **Applications** access the services provided by **TCP** and **UDP** through the **sockets API**



What is a socket?

- Socket: An interface between an application process and transport layer
  - The application process can send/receive messages to/from another application process (local or remote)via a socket
- In Unix jargon, a socket is a file descriptor – an integer associated with an open file
- Types of Sockets: **Internet Sockets**, unix sockets, X.25 sockets etc
  - Internet sockets characterized by IP Address (4 bytes), port number (2 bytes)

Socket Description



Types of Internet Sockets

- Stream Sockets (SOCK_STREAM)
  - Connection oriented
  - Rely on TCP to provide reliable two-way connected communication
- Datagram Sockets (SOCK_DGRAM)
  - Rely on UDP
  - Connection is unreliable