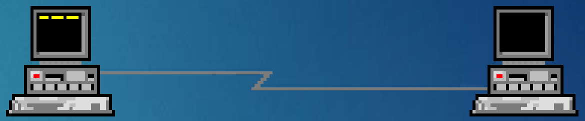
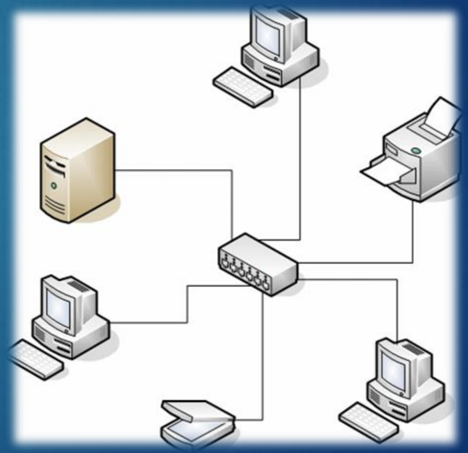
Introduction to Socket Programming

Computer Networks

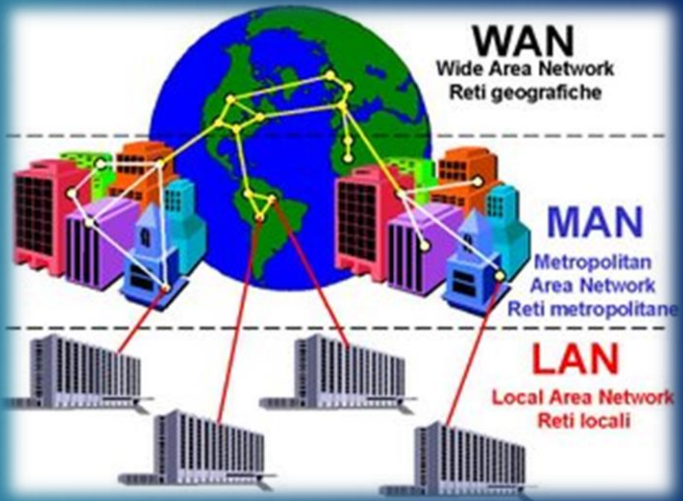


- What is a computer network?
  - Interconnected 2 or more computers or hardware devices
- What is the purpose of Computer networks?
  - Resource sharing
  - Information sharing
  - Communication

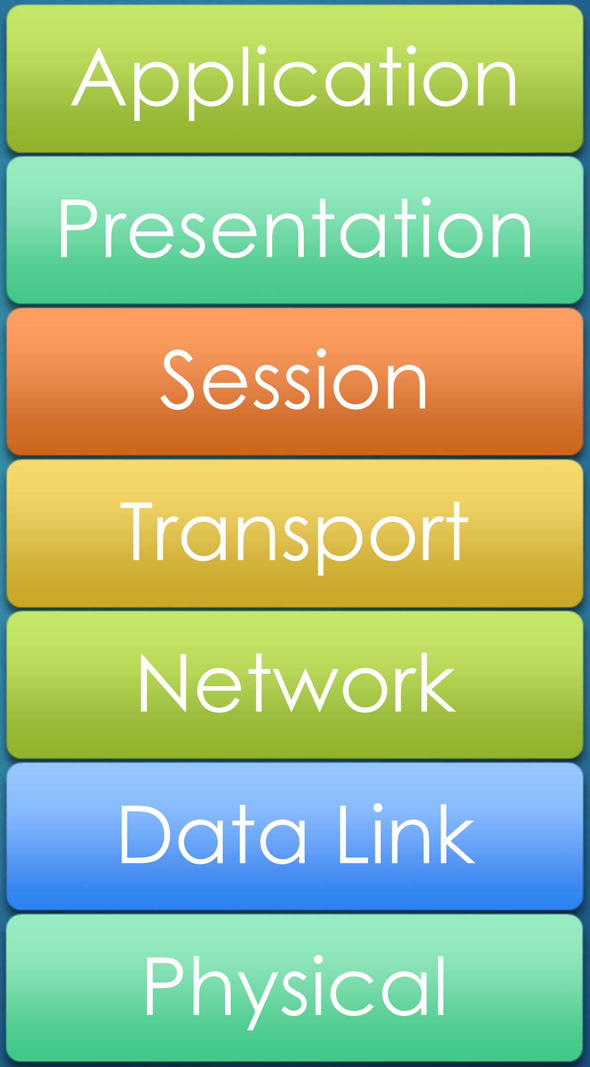


Computer Networks

- PAN (Personal Area Network)
  - 1 m
- LAN (Local Area Network)
  - 10 m to 1000 m (1 km)
- MAN (Metropolitan Area Network)
  - 10 km
- WAN (Wide Area Network)
  - 100 km to 1000 km



OSI Model



Application

Presentation

Session

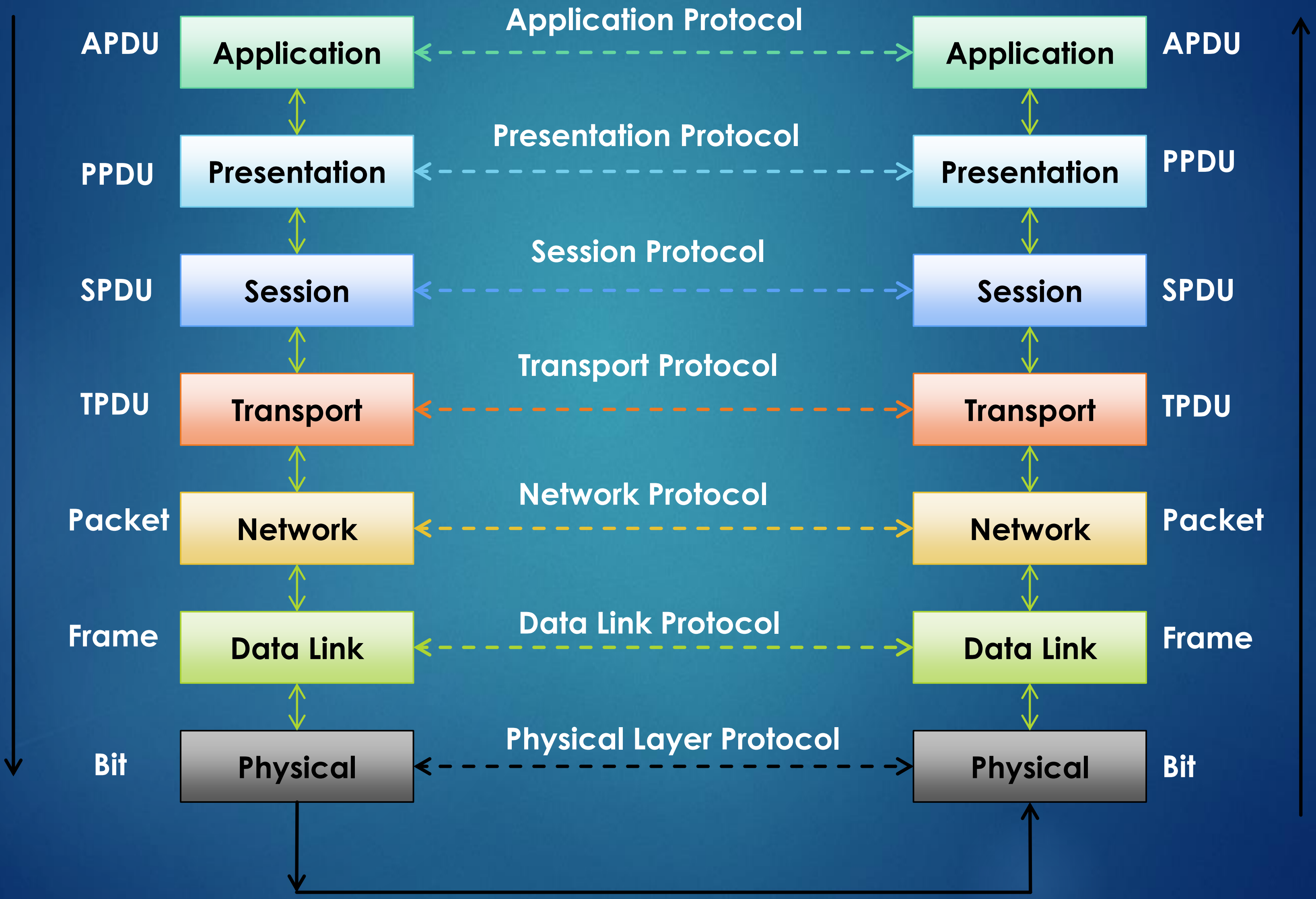
Transport

Network

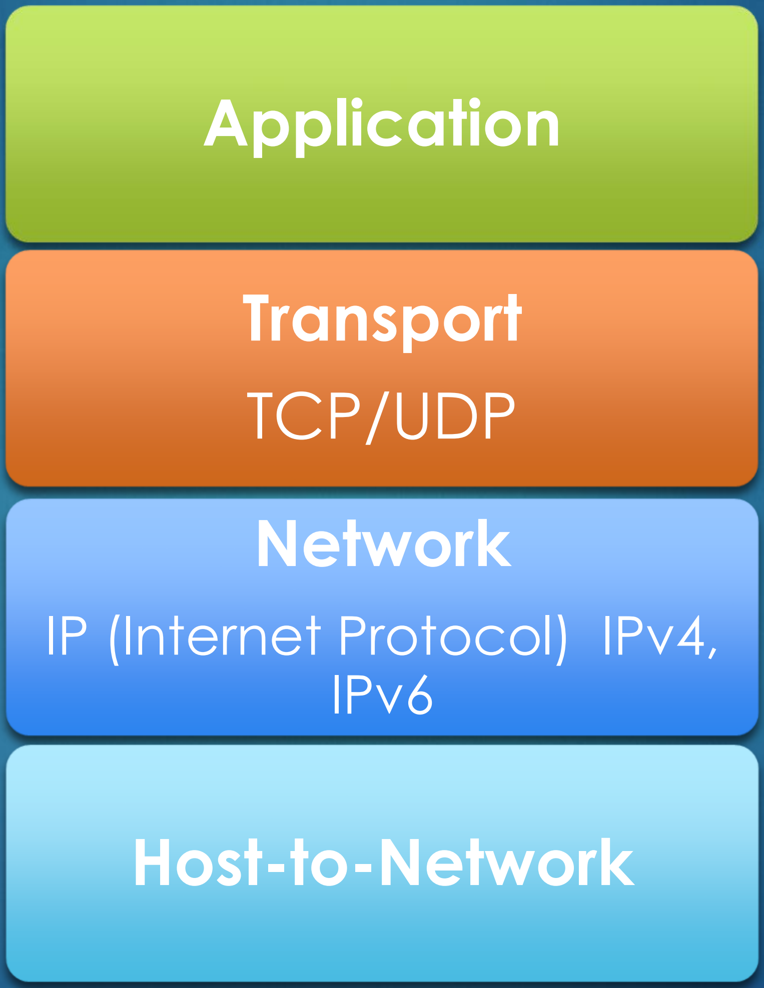
Data Link

Physical

Communication



Internet Protocol Suit



Application

Transport

TCP/UDP

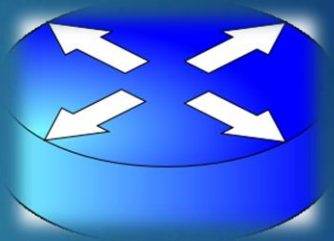
Network

IP (Internet Protocol) IPv4, IPv6

Host-to-Network

Network Devices

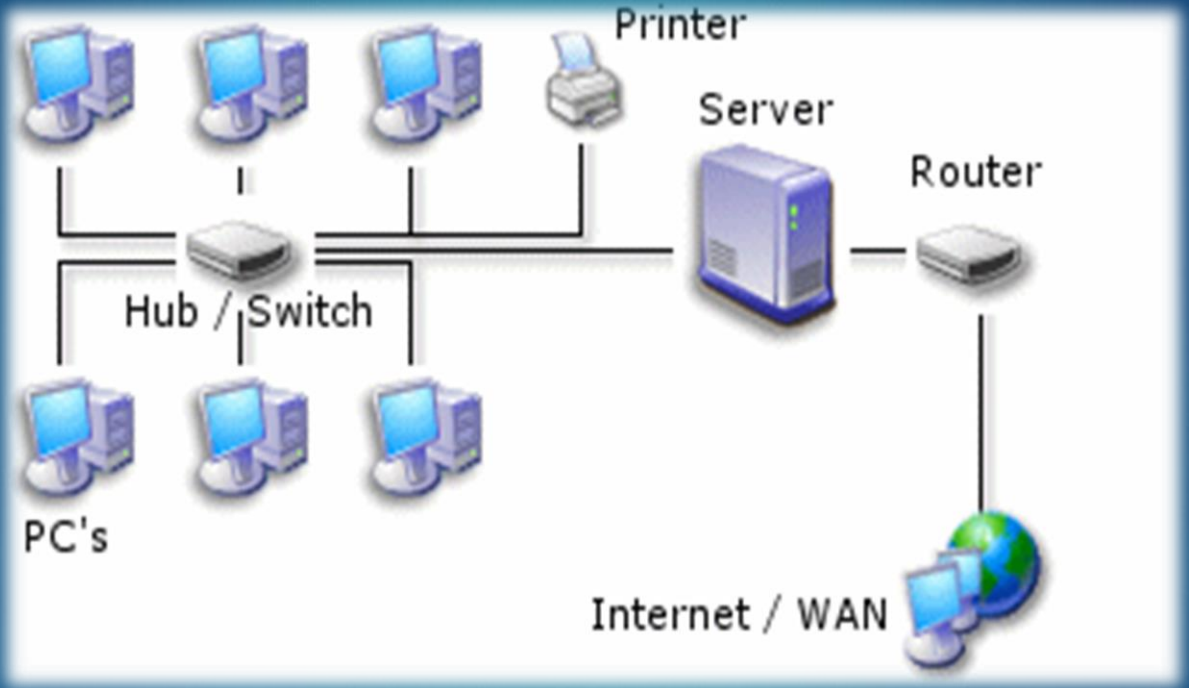
- Routers (Layer 3) Network



- Switch (Layer 2) Data Link

- HUB (Layer 1 device) Physical

Network Devices Use



Addressing

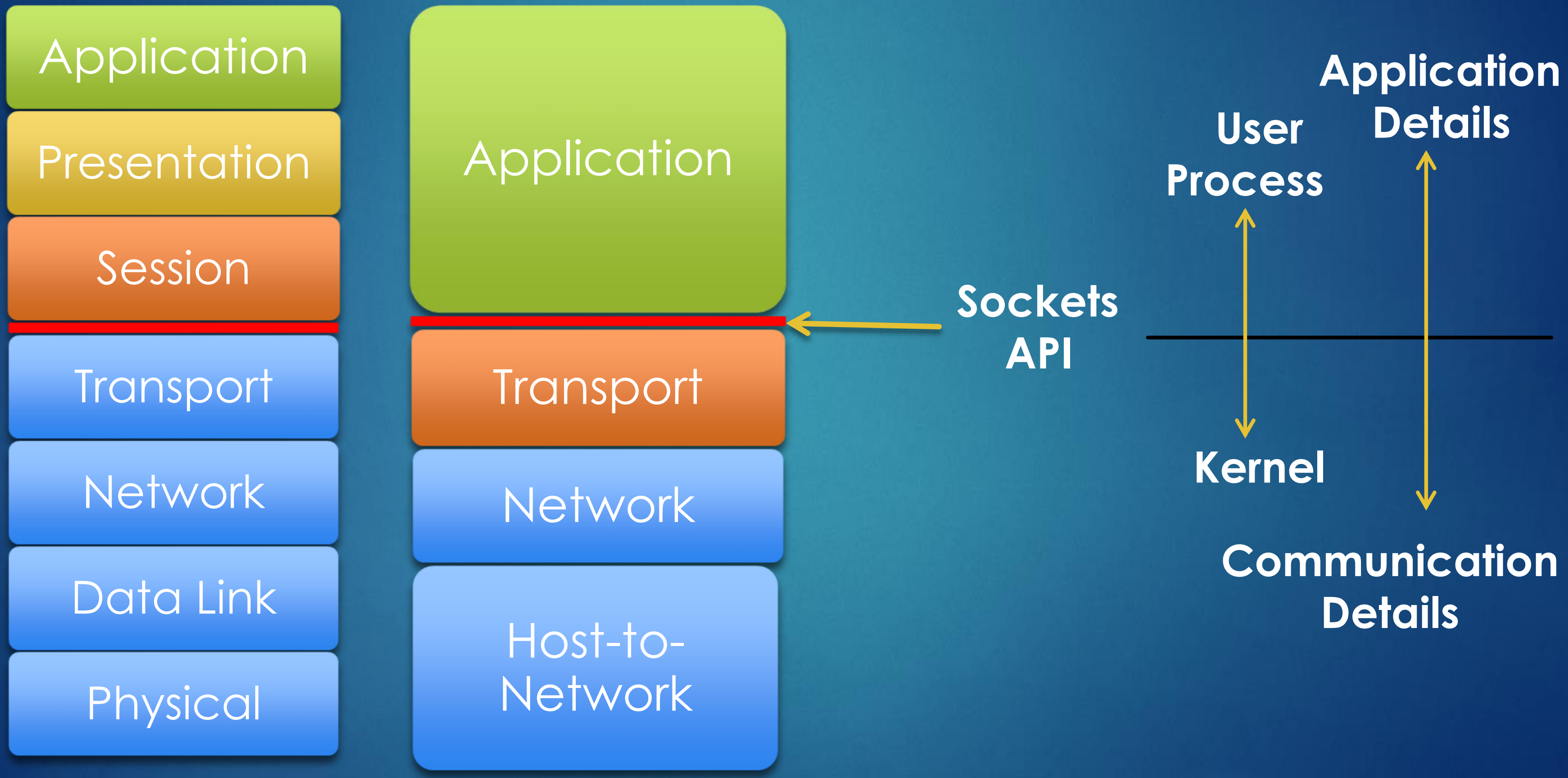
- Mac Address
- IP Address
- Port Address

Socket Programming

- Why Socket Programming?
  - To build any Network Application
  - Web browsers (Internet Explorer , Firefox)
  - Web Apps (Chat, Mail, File Transfer Apps)

What is the Socket?

Socket (An application programming interface(API) for inter process communication)



What is the Socket?

- Socket(Communication End Point)
- Working with Sockets is similar to working with files

File I/O	Socket I/O
Open File	Open Socket
	Name the Socket
	Associate with another Socket
Read and write	Send and Receive between Sockets
Close the File	Close the Socket

- Socket has always an address (**IP and Port**)
- Functionality (Communication)

One application process can communicate with another application process (local or remote) using a socket.

TCP & UDP

- Difference between UDP and TCP
- Where to use what?
- Applications of UDP
- Applications of TCP

Socket Types

- 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

Functions used in Socket Programming

Socket()	Endpoint for communication
Bind()	Assign a unique telephone number
Listen()	Wait for a caller
Connect()	Dial a number
Accept()	Receive a call
Send(), Recv()	Talk
Close()	Hang up

Socket() … Get the file descriptor

```
int sd=Socket(int domain,int type,int protocol);
```

Domain: AF_INET, PF_INET

Type: SOCK_STREAM, SOCK_DGRAM

Protocol: Set to “0” for appropriate protocol selection, IPPROTO_TCP, IPPROTO_UDP

Return: Socket descriptor on success and -1 on error

Example:

```
int U_s=socket(AF_INET, SOCK_STREAM, 0);
int T_s=socket(AF_INET, SOCK_DGRAM, 0);
```

bind()… what port am I on?

Associate a socket id with an address to which other process can connect

int status=bind(int sd, struct sockaddr* addrptr, int size);

Status: 0 on success and on error -1

sd: socket file decriptor created return by socket()

addrptr: pointer to Struct sockaddr type parameter, contains current socket IP and port

size: size of *addrptr*.

connect()... Request for connection

int status = connect(int sd, struct sockaddr *serv_addr, int addrlen);

status: error -1

sd: socket file descriptor

serv_addr: is a pointer to struct sockaddr that contains destination IP address and port

addrlen: size of serv_addr

listen()

Waits for incoming connections

```
int status = listen(int sd,int backlog);
```

sd: socket on which the server is listening

backlog: maximum no of connections pending in a queue

status: return -1 on error

accept()

Blocking System Call Waits for an incoming request, and when received creates a socket for it.

int sid = accept(int sd, struct sockaddr *cli_addr, int *addrlen);

sid: socket file descriptor for communication

sd: socket file descriptor used for listening

addr: pointer to struct sockaddr containing client address IP and Port

addrlen: sizeof struct sockaddr

send()

int sb = send(int sd, const char *msg, int len, int flags);

Sb: return No of bytes send or -1 on error

sd: socket file descriptor

msg: is a pointer to data buffer

len: no of bytes we want to send

flag: set it to 0 default

recv()

int rb = recv(int sd, char *buf, int len, int flags);

rb: No of bytes received or -1 on error **0** if connection is closed at other side

sd: socket file descriptor

buf: is a pointer to data buffer

len: receive up to len bytes in buffer pointer

flag: set it to 0 defalut

close() , Shutdown()

Close connection on given socket and frees the socket descriptor

```
int close(fd);
```

Acts as a partial close, disables sending (how=1) or receiving (how=0). Returns -1 on failure.

```
int shutdown(int sd, int how);
```

Sockaddr, Sockaddr_in

struct sockaddr: Generic Holds socket address information for many types of sockets

```
struct sockaddr {
    unsigned short sa_family;       //address family AF_xxx
    unsigned short sa_data[14];    //14 bytes of protocol addr
}
```

struct sockaddr_in: IPV4 specific

```
struct sockaddr_in {
    short int sin_family;                // set to AF_INET
    unsigned short int sin_port;       // Port number
    struct in_addr sin_addr;           // Internet address
    unsigned char sin_zero[8];         //set to all zeros
}
```

Byte Ordering

- **Network Byte Order: Big Indian** (High-order byte of the number is stored in memory at the lowest address)
- **Host Byte Order: Little Indian** (Low-order byte of the number is stored in memory at the lowest address) or **Big Indian**

htons()	Host to Network Short
htonl()	Host to Network Long
ntohs()	Network to Host Short
ntohl()	Network to Host Long

An dotted decimal IP4 string address to a network byte ordered 32 bit

inet_addr()

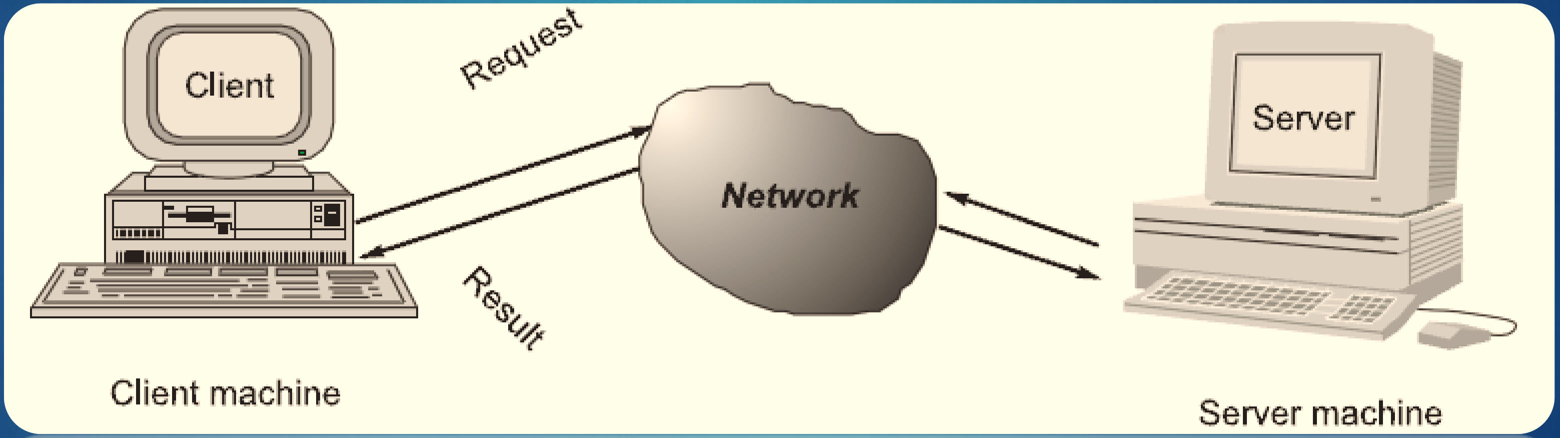
inet_aton()

To convert a 32 bit network byte ordered to a IP4 dotted decimal string

inet_ntoa()

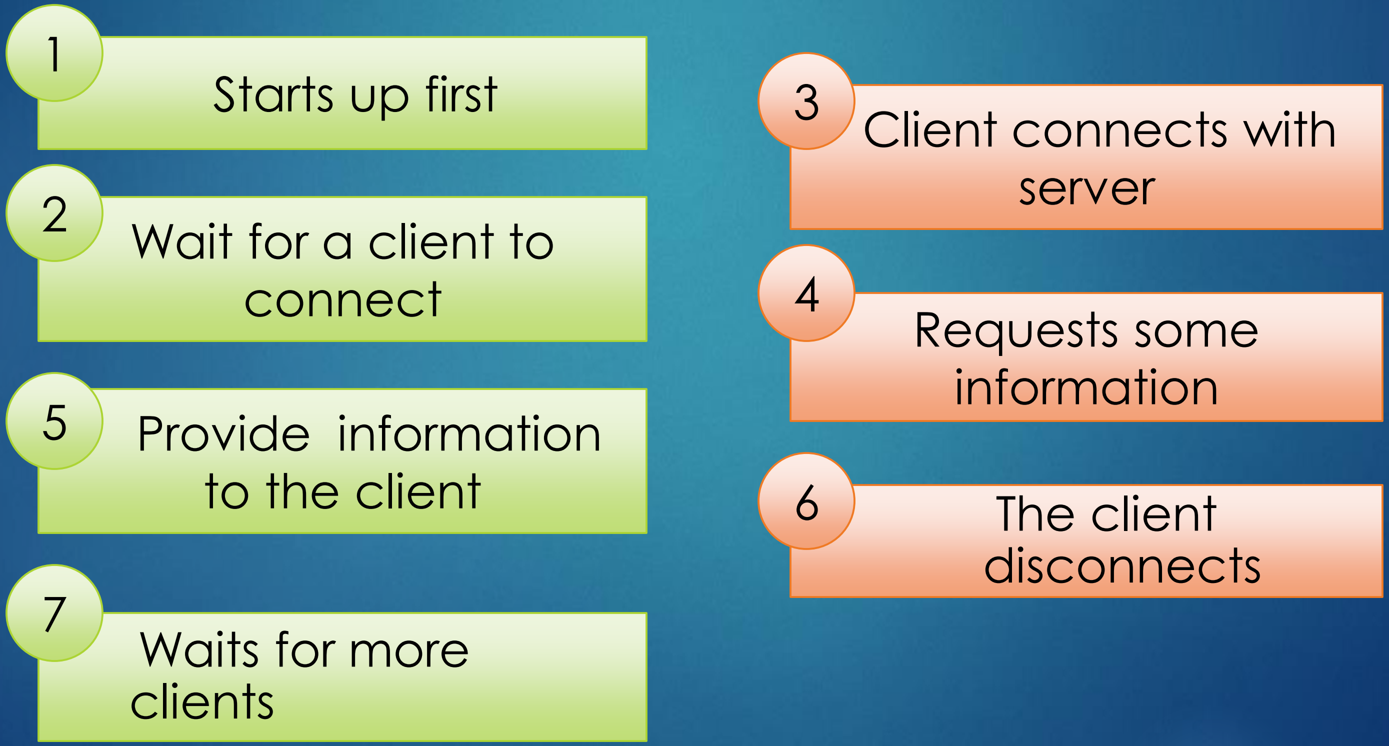
The Client – Server model

- Server – Provider of Services
- Client – Seeker of Services



The Client – Server model

- In the socket programming world almost all communication is based on the Client-Server model.



Server Process

1 Starts up first

2 Wait for a client to connect

5 Provide information to the client

7 Waits for more clients

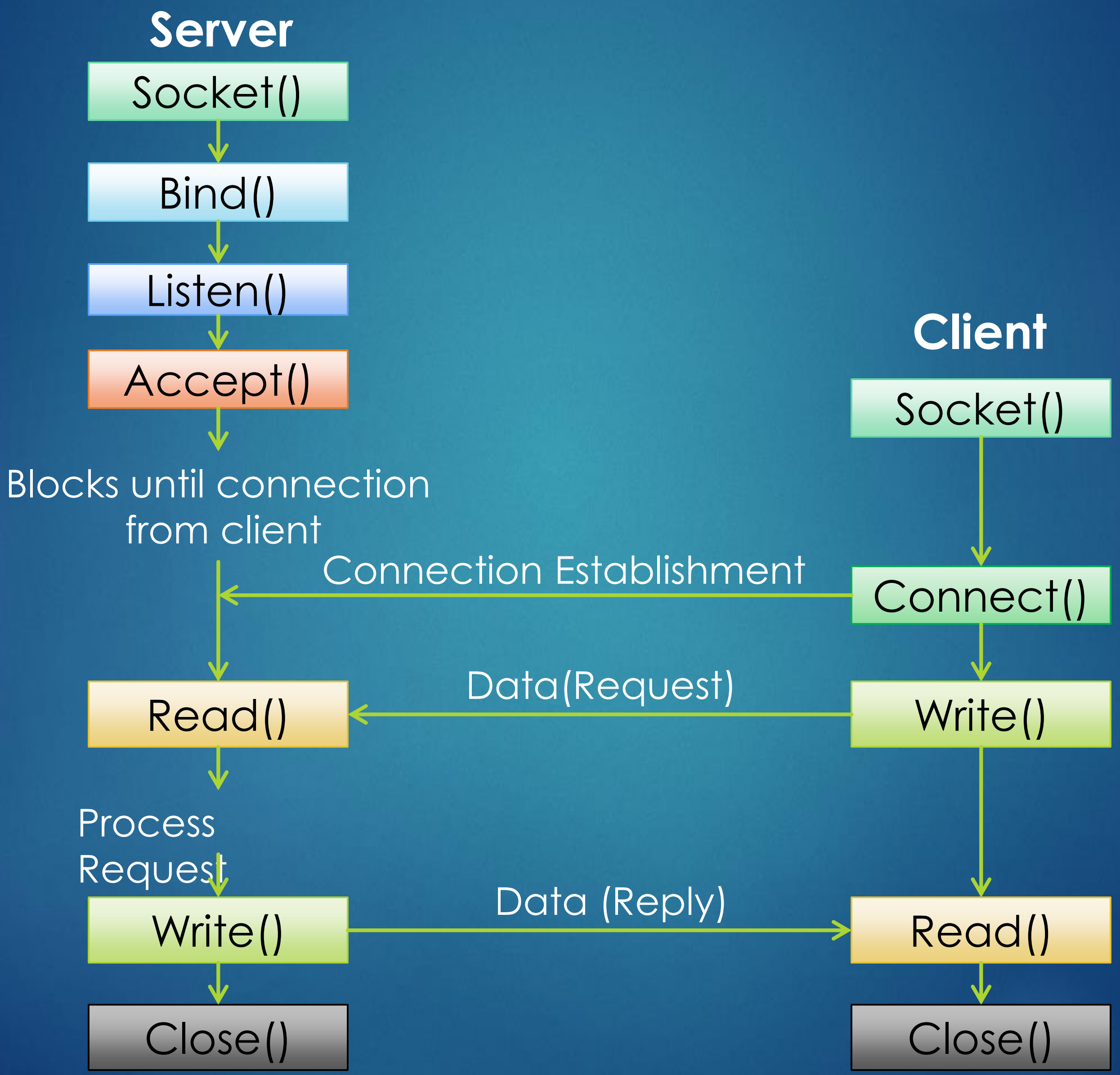
Client Process

3 Client connects with server

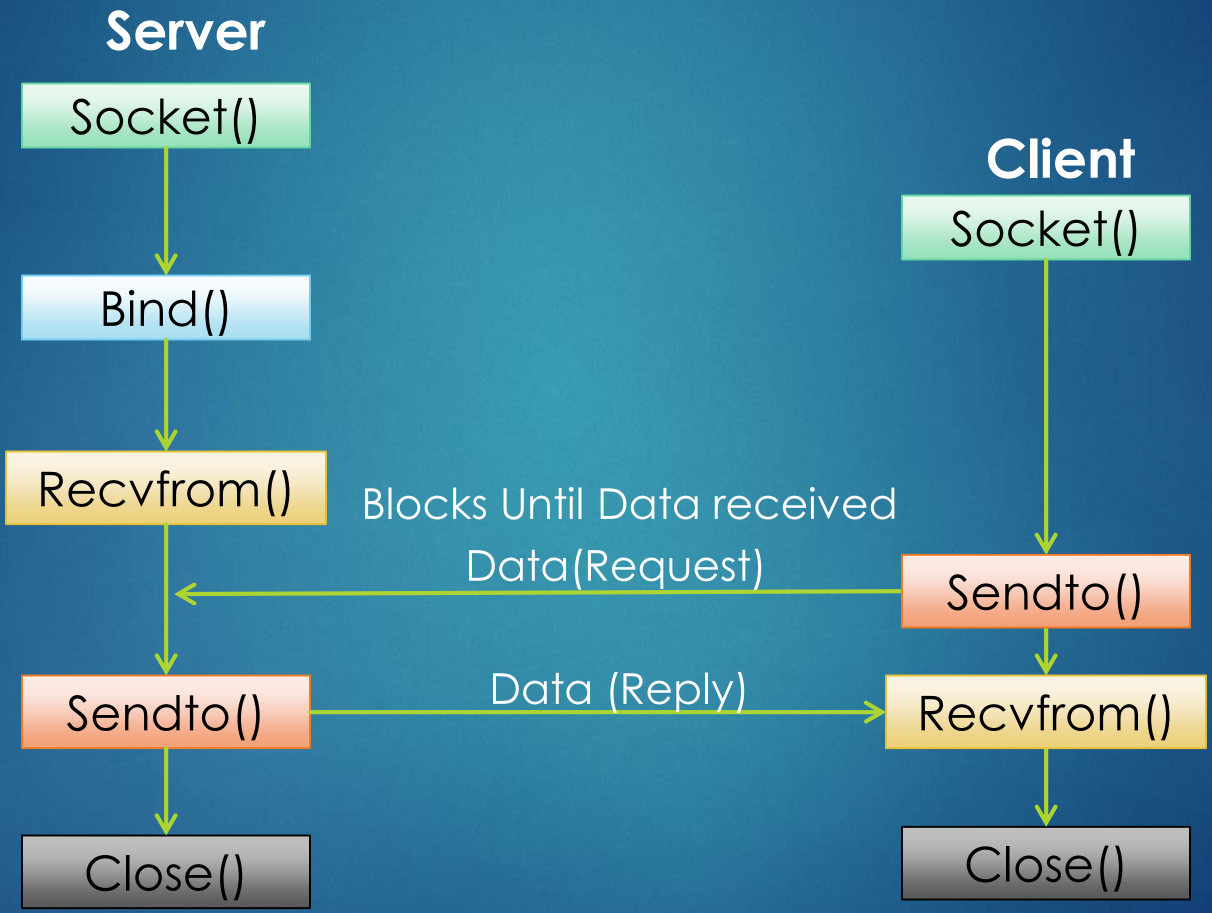
4 Requests some information

6 The client disconnects

TCP Server – Client Interaction



UDP Server – Client Interaction



Some Commands

- ipconfig (for IP inquiry) Windows
- Ifconfig (for IP inquiry) Linux
- ipconfig /all to check Mac Address of System