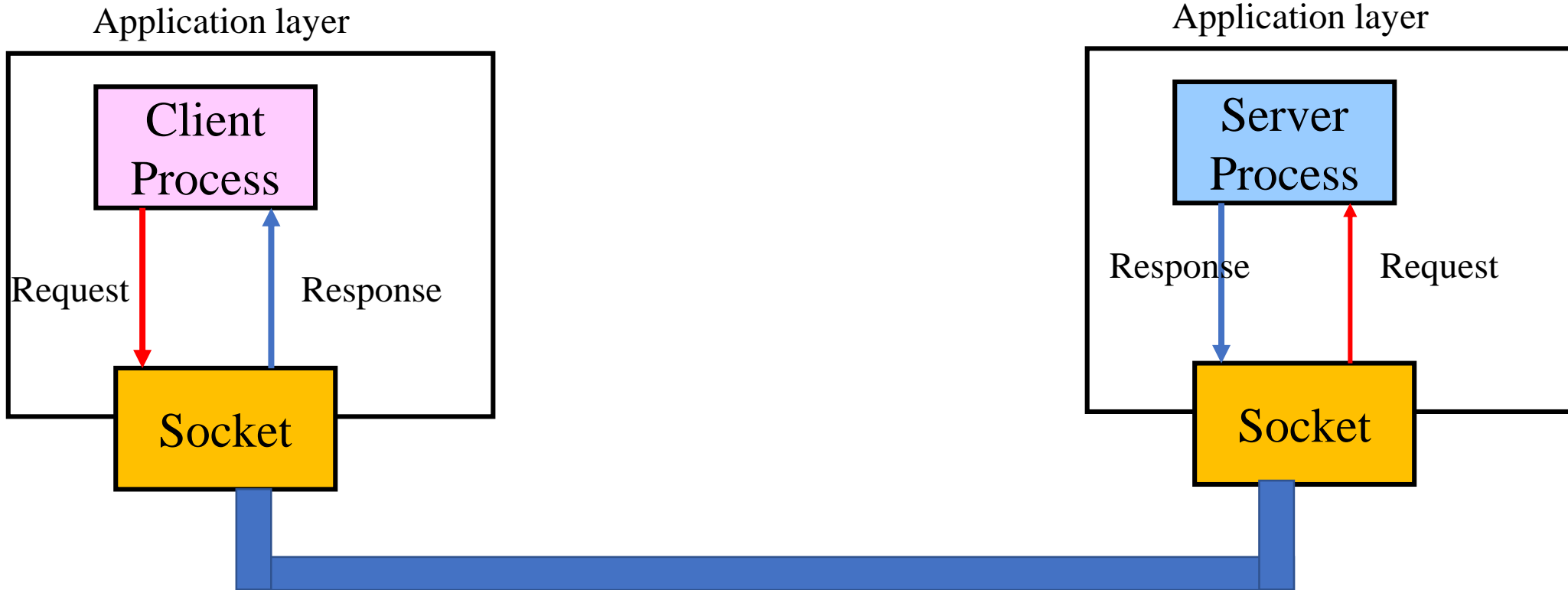


Socket Programming

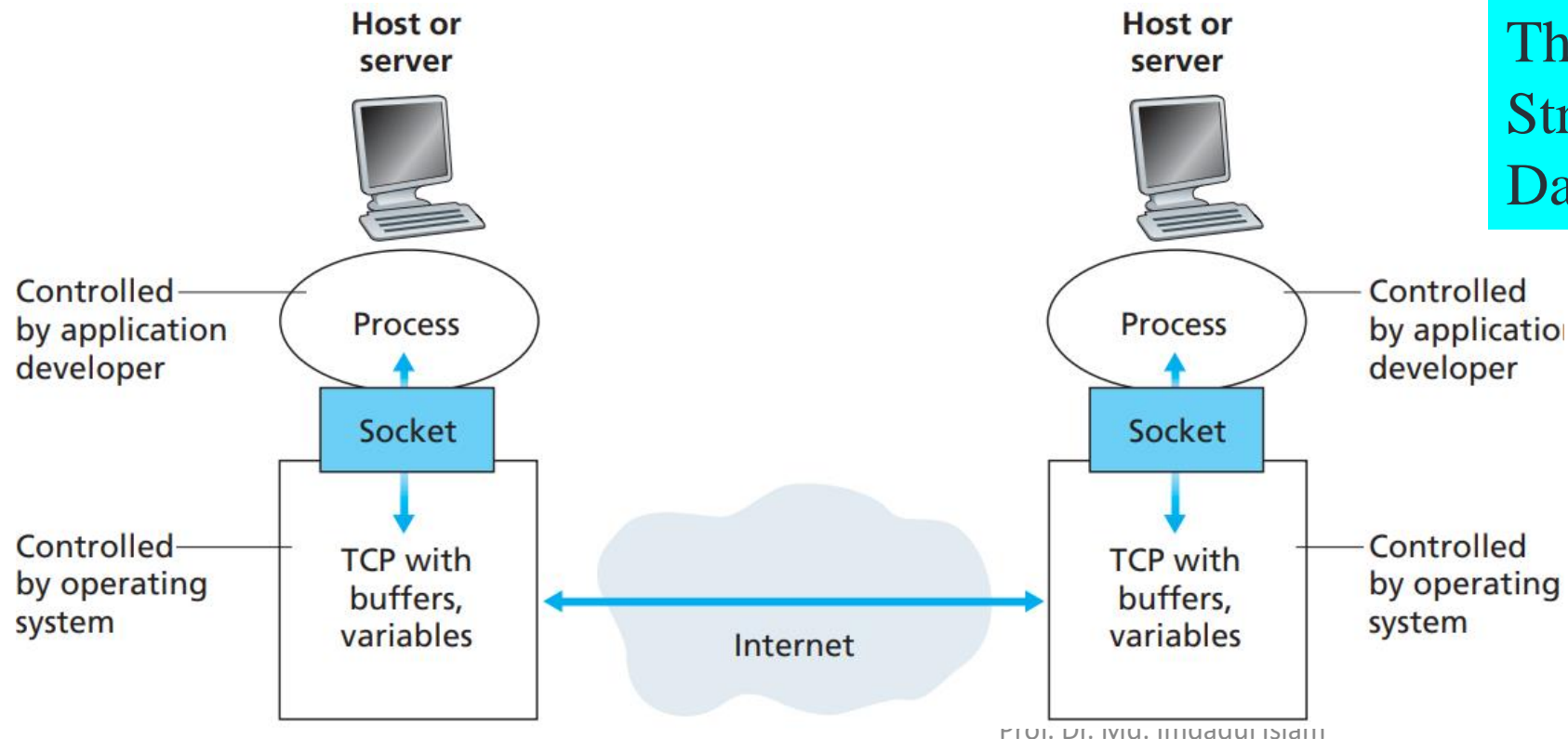
- ✓ A process sends messages into, and receives messages from, the network through a **software interface / object / point entity** called a socket.

- ✓ Application layer deals with communication between client process and server process, which is done between two **sockets** created at two ends.



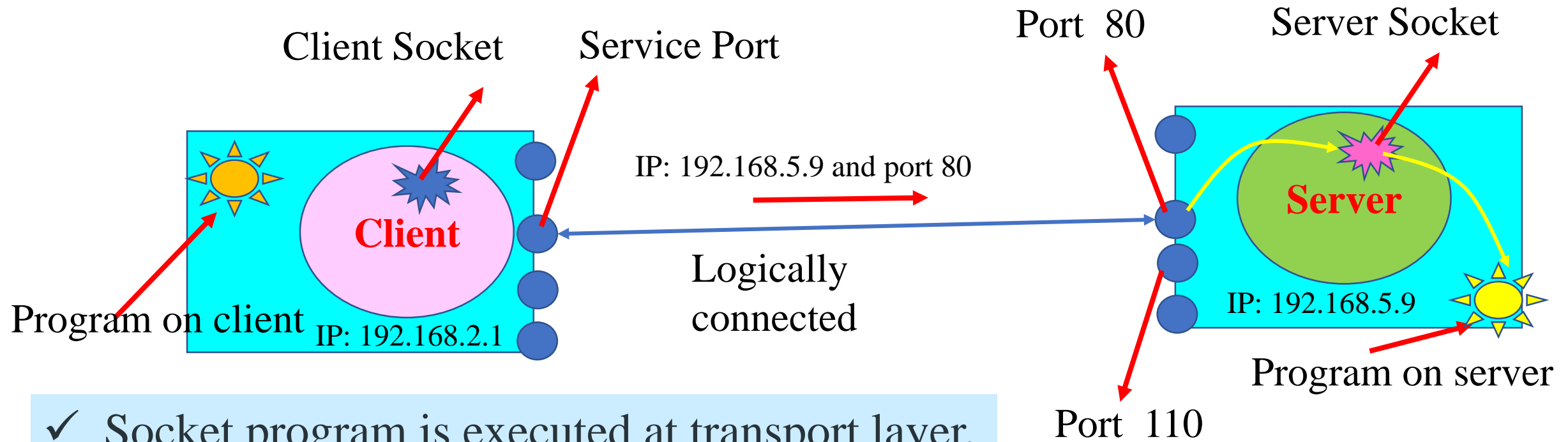
- ✓ The client thinks that the socket is the entity that receives the request and gives the response.
- ✓ The server thinks that the socket is the entity that has a request and needs the response.

- ✓ **During transmission**, a process sends messages to socket then socket sends them to transport layer and the responsibility is taken by the OS. Again during reception, messages are received by application from the socket. A process is analogous to a house and its socket is analogous to its door.
- ✓ A socket is the interface between the application layer and the transport layer within a host. It is also referred to as the Application Programming Interface (API).

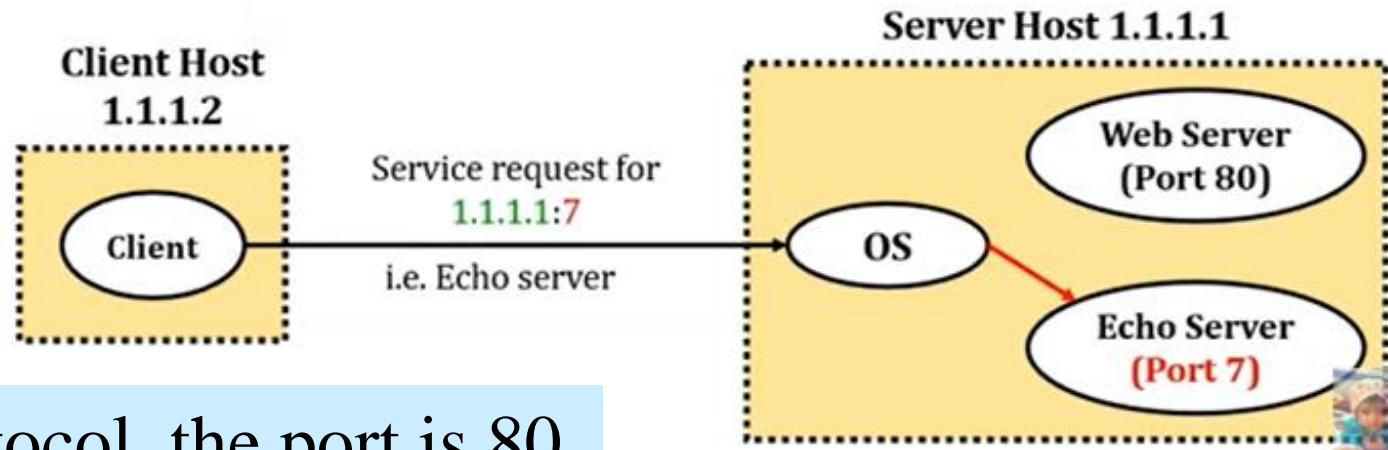
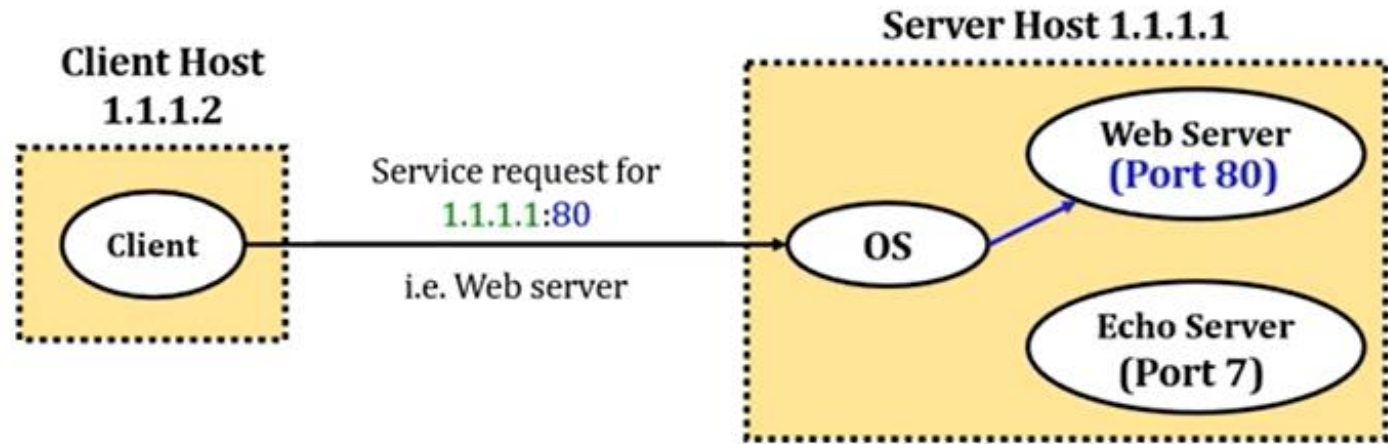


There are two types of sockets:
Stream Socket (TCP)
Datagram Socket (UDP)

- ✓ **Socket address** is the combination of IP address and service port number (16 bits number) i.e. a socket binds IP and port together. For example IP: 192.168.5.9 and port 80 gives the socket address.
- ✓ Message is received by the **transport layer** identifies the application port number (for example, http port 80), then the message is sent to the socket whose response is finally sent to the application program/process.



- ✓ Socket program is executed at transport layer.



Web service under *http* protocol, the port is 80.
POP of e-mail, the port is 110.
SMTP of e-mail the port is 25.

The main functions of socket programming:

BIND - attach a socket to a specific local IP address and port number.

CONNECT- attempt to establish a connection

LISTEN- indicate that the socket can accept new connections.

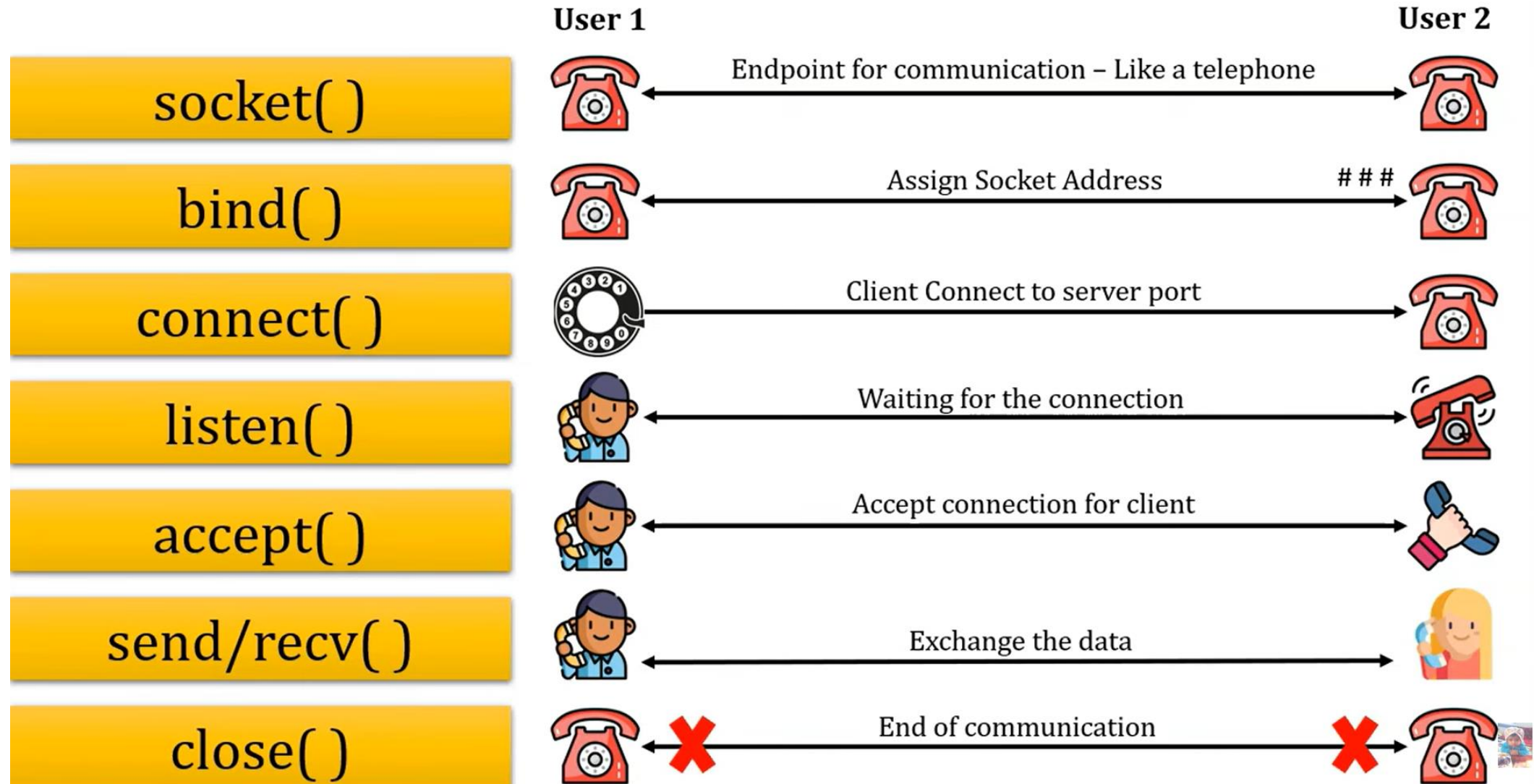
ACCEPT-begin a network connection following an incoming connection request.

SEND- transmit data over the connection.

RECEIVE- receive data over the connection.

CLOSE- release the connection.

- ✓ Socket is considered as the endpoint (like an object at the communication device i.e. telephone set) of a 2-way communication.



Create a socket or node named s, where s is also called socket object

An example script to connect to Google using socket

programming in Python

import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

print ('Socket successfully created')

default port for socket

port=80

host_ip = socket.gethostbyname('www.google.com')

connecting to the server

s.connect((host_ip, port))

print ('IP of www.google.com',host_ip)

socket.SOCK_STREAM
represents TCP

.AF_INET represents IPv4

www.juniv.edu


```
C:\Users\HP>tracert www.google.com

Tracing route to www.google.com [142.250.192.228]
over a maximum of 30 hops:

 1  <1 ms    <1 ms    <1 ms    172.16.48.1
 2  <1 ms    <1 ms    <1 ms    172.16.251.33
 3  <1 ms    <1 ms    <1 ms    172.16.251.9
 4  <1 ms    <1 ms    <1 ms    163.47.36.9
 5   3 ms     3 ms     3 ms     100.100.4.22
 6   3 ms     3 ms     3 ms     10.100.240.57
 7   *        *        *        Request timed out.
 8   3 ms     3 ms     3 ms     100.100.0.57
 9  12 ms    10 ms    11 ms    123.49.8.146
10  40 ms    40 ms    40 ms    123.49.13.0
11  40 ms    40 ms    40 ms    108.170.253.122
12  59 ms    54 ms    54 ms    72.14.239.10
13  55 ms    55 ms    55 ms    74.125.243.97
14  55 ms    55 ms    54 ms    142.251.54.65
15  54 ms    54 ms    54 ms    del11s13-in-f4.1e100.net [142.250.192.228]

Trace complete.

C:\Users\HP>
```

We got the same IP from *tracert* and *socket* programming.

Access google using
<http://142.250.192.228>

Socket successfully created
IP of www.google.com **142.250.192.228**

```

C:\Users\HP>tracert www.juniv.edu

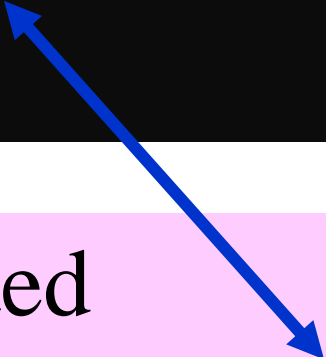
Tracing route to juniv.edu [72.249.68.156]
over a maximum of 30 hops:

  0  <1 ms    <1 ms    <1 ms    172.16.48.1
  1  <1 ms    <1 ms    <1 ms    172.16.251.33
  2  <1 ms    <1 ms    <1 ms    172.16.251.9
  3  <1 ms    <1 ms    <1 ms    163.47.36.9
  4  <1 ms    <1 ms    <1 ms    100.100.4.22
  5  3 ms     3 ms     5 ms     10.100.240.57
  6  3 ms     3 ms     3 ms     10.100.240.57
  7  *         *         *         Request timed out.
  8  3 ms     3 ms     3 ms     100.100.0.57
  9  63 ms    62 ms    62 ms    203.208.191.141
 10  62 ms    61 ms    60 ms    203.208.182.249
 11  243 ms   236 ms   236 ms    203.208.158.46
 12  238 ms   238 ms   238 ms    203.208.172.234
 13  243 ms   242 ms   242 ms    64.125.35.189
 14  278 ms   278 ms   277 ms    ae6.cs2.sjc2.us.eth.zayo.com [64.125.25.100]
 15  *        278 ms   *        ae2.cs2.lax112.us.eth.zayo.com [64.125.28.197]
 16  *        267 ms   *        ae12.cs2.dfw2.us.zip.zayo.com [64.125.26.182]
 17  280 ms   280 ms   280 ms    ae24.er2.dfw2.us.zip.zayo.com [64.125.27.105]
 18  283 ms   283 ms   283 ms    64.124.196.226.t00876-01.above.net [64.124.196.226]
 19  282 ms   278 ms   291 ms    ae3.jvalue1.dal.tierpoint.net [206.123.64.17]
 20  267 ms   267 ms   267 ms    ae0.jvalue3.dal.tierpoint.net [207.210.229.6]
 21  278 ms   278 ms   278 ms    207.210.228.50
 22  292 ms   278 ms   278 ms    vps.juniv.edu [72.249.68.156]

Trace complete.

C:\Users\HP>

```



Socket successfully created
IP of www.juniv.edu 72.249.68.156

Client Server Communications

Server side

Import the python **socket** module, this is a built-in module

IP address 127.0.0.1

```
import socket
LOCALHOST = "127.0.0.1"
PORT = 8080
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((LOCALHOST, PORT))
server.listen(1)
print("Server started")
print("Waiting for client request..")
```

We pick a random high TCP port to listen on

BIND - attach a socket to a specific local IP address and port number. For a local server, we might choose to bind to the IP address 127.0.0.1 with the TCP port number 8080.

It indicates that server has the capability of listening.

```
clientConnection, clientAddress = server.accept()
print("Connected client :" , clientAddress)
msg = ""
while True:
    in_data = clientConnection.recv(1024)
    msg = in_data.decode()
    if msg=='bye':
        break
    print("From Client :" , msg)
    out_data = input()
    clientConnection.send(bytes(out_data,'UTF-8'))
print("Client disconnected....")
clientConnection.close()
```

Incoming connections will be accepted and provide a new socket to communicate with a given client (clientConnection) and the address of the client (clientAddress) is stored at the server

After a connection is established, the server prints out the client address and then waits for data.

Byte code (UNICODE) to original text converter for received data

Received data from client in byte format where buffer size of 1024 bytes at a time.

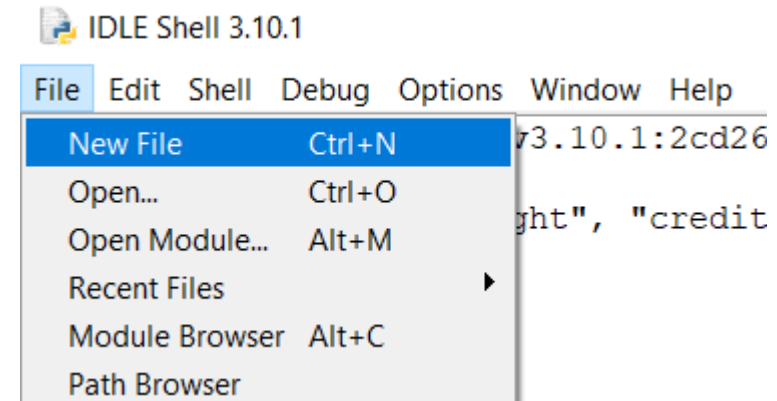
Original Text is converted to byte (UNICODE) during transmission

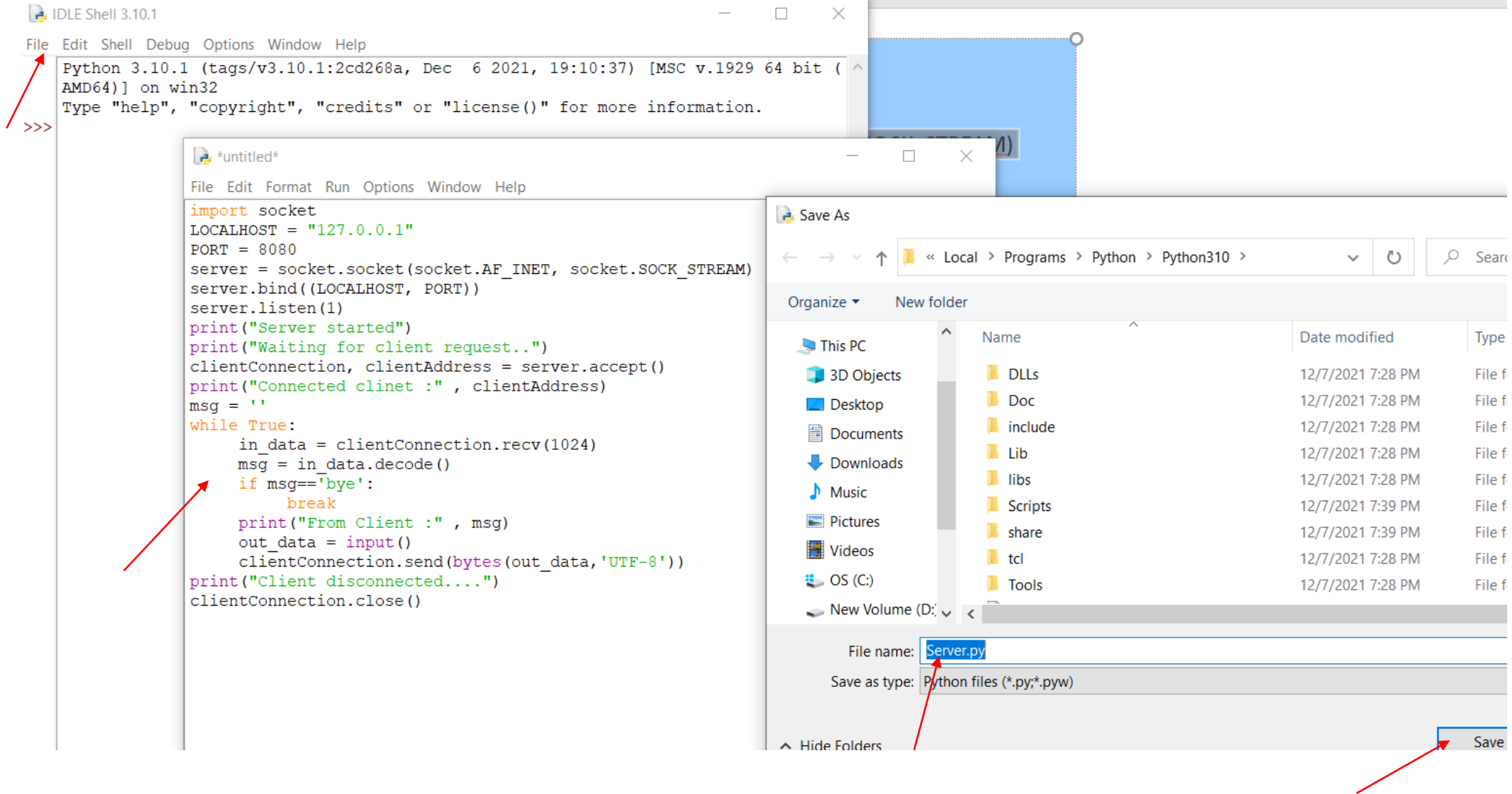
```

import socket
LOCALHOST = "127.0.0.1"
PORT = 8080
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((LOCALHOST, PORT))
server.listen(1)
print("Server started")
print("Waiting for client request..")
clientConnection, clientAddress = server.accept()
print("Connected client : " , clientAddress)
msg = ""
while True:
    in_data = clientConnection.recv(1024)
    msg = in_data.decode()
    if msg=='bye':
        break
    print("From Client : " , msg)
    out_data = input()
    clientConnection.send(bytes(out_data,'UTF-8'))
print("Client disconnected....")
clientConnection.close()

```

Copy it on new file of idle shell
and save it as Server.py





Client side

```
import socket
SERVER = "127.0.0.1"
PORT = 8080
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((SERVER, PORT))
client.sendall(bytes("This is from Client",'UTF-8'))
while True:
    in_data = client.recv(1024)
    print("From Server :",in_data.decode())
    out_data = input()
    client.sendall(bytes(out_data,'UTF-8'))
    if out_data=='bye':
        break
client.close()
```

1.The client also starts by calling SOCKET to create a new socket.

2.Next, the client will attempt to CONNECT to the server. The client doesn't need to be explicitly told to BIND to a specific address - it will pick an appropriate IP and port (usually a random high port).

3.After the server has accepted the connection, the client can SEND and RECEIVE data according to the protocols being used. With HTTP, the client will send GET and POST requests and receive HTTP responses back from the server.

4.Finally, when both client and server issue the CLOSE primitive, the connection will be torn down.

Open a new shell and copy it on new file of idle shell and save it as Cleint.py. Make conversation with the server and client.

Client

```
=== RESTART:  
C:\Users\CSEJU\AppData\Local\Programs\Python\Python310\client.py ==  
From Server : Hellow  
I am client (2)  
From Server : Nice to hear  
me too (4)  
From Server : bye  
Bye (5)
```

Repeat the job using
command prompt of C:

Server

```
=== RESTART:  
C:\Users\CSEJU\AppData\Local\Programs\Python\Python310\Server.py ==  
Server started  
Waiting for client request..  
Connected client : ('127.0.0.1', 57676)  
From Client : This is from Client  
Hellow (1)  
From Client : I am client  
Nice to hear (3)  
From Client : me too  
Bye (5)  
Client disconnected....
```