Socket Programming

WHAT IS A SOCKET

- ❑ One end-point of a two-way communication link between two programs running on the network
- ❑ Connection-oriented sockets

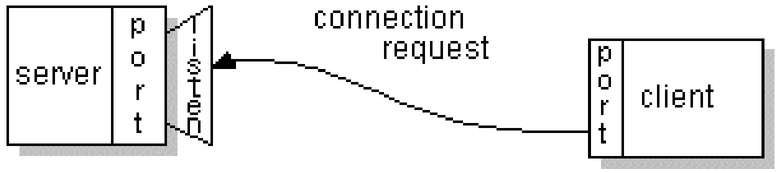
Source IP address	Source port number	Dest IP address	Dest port number

- ❑ Connectionless sockets

Dest IP address	Dest port number

WHAT IS A SOCKET

1. Server runs on a specific computer and has a socket that is bound to a specific port number
   - Server just waits, listening to the socket for a client to make a connection request



2. Client knows the hostname of the machine on which the server is running and the port number on which the server is listening
   - Client also needs to identify itself to the server so it binds to a local port number that it will use during this connection

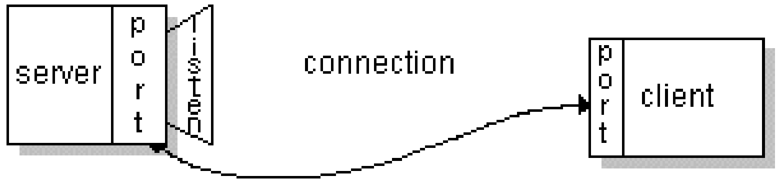


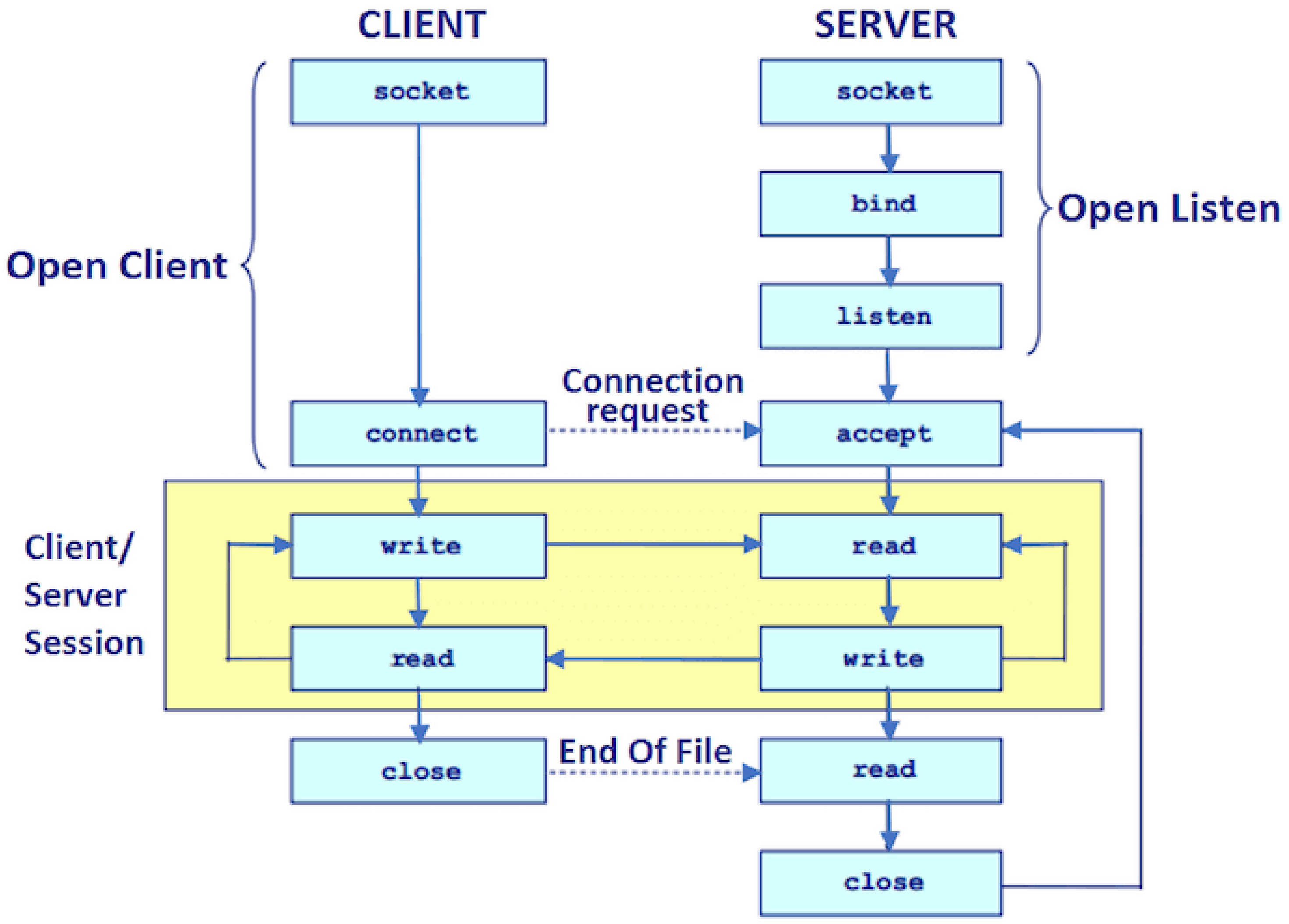
Image source: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

WHAT IS A SOCKET

3. Server accepts the connection
   - Upon acceptance, the server gets a new socket bound to the same local port and has its remote endpoint set to the address and port of the client
   - A socket is successfully created on the client-side and the client can use the socket to communicate with the server
4. The client and server can now communicate by writing to or reading from their sockets

SOCKET PROGRAMMING

- Socket classes are used to represent the connection between a client program and a server program
- The **java.net** package provides two classes
  - Socket – client-side connection
  - ServerSocket – server-side connection



SOCKET API

Image source: https://www.javatpoint.com/socket-programming

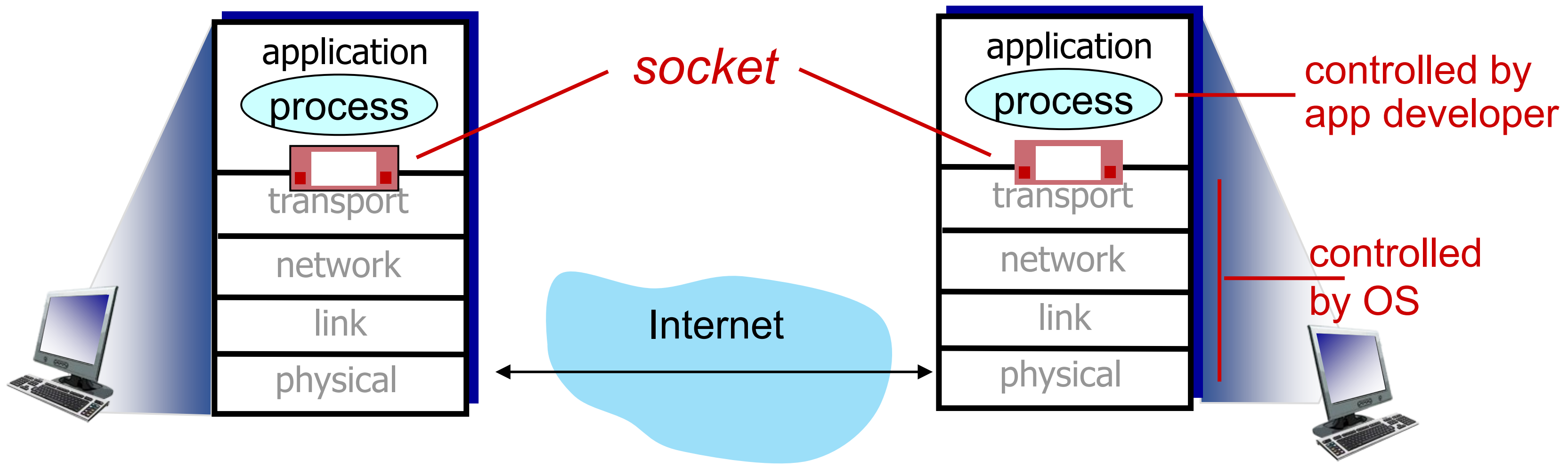
OUTLINE

- ❑ Sockets and Socket Programming
- ❑ Writing the Server-side Application
- ❑ Writing the Client-side Application

SOCKET PROGRAMMING

goal: learn how to build client/server applications that communicate using sockets

socket: door between application process and end-end-transport protocol



SOCKET PROGRAMMING

Two socket types for two transport services:

- *UDP:* unreliable datagram
- *TCP:* reliable, byte stream-oriented

Application Example:

1. client reads a line of characters (data) from its keyboard and sends data to server
2. server receives the data and converts characters to uppercase
3. server sends modified data to client
4. client receives modified data and displays line on its screen

SOCKETS AND SOCKET PROGRAMMING

PYTHON PROGRAMMING LANGUAGE

SOCKET PROGRAMMING WITH UDP

UDP: no "connection" between client and server:

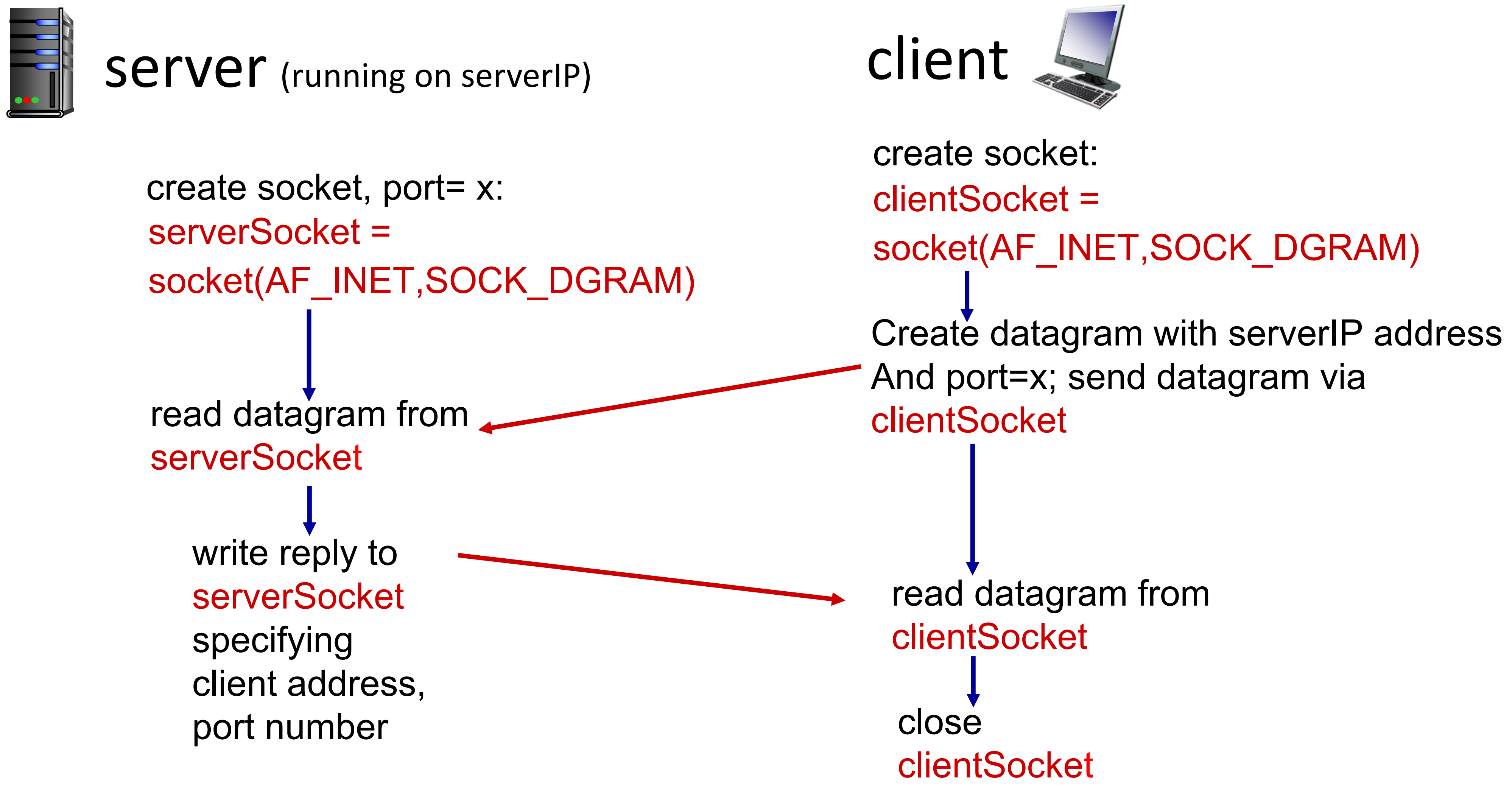
- ❑ no handshaking before sending data
- ❑ sender explicitly attaches IP destination address and port # to each packet
- ❑ receiver extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

- UDP provides *unreliable* transfer of groups of bytes ("datagrams") between client and server processes

CLIENT/SERVER SOCKET INTERACTION: UDP



server (running on serverIP)

create socket, port= x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)

read datagram from
serverSocket

write reply to
serverSocket
specifying
client address,
port number

client

create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)

Create datagram with serverIP address
And port=x; send datagram via
clientSocket

read datagram from
clientSocket

close
clientSocket

EXAMPLE APP: UDP CLIENT

Python UDPClient

include Python's socket library →	`from socket import *`
	`serverName = 'hostname'`
	`serverPort = 12000`
create UDP socket →	`clientSocket = socket(AF_INET, SOCK_DGRAM)`
get user keyboard input →	`message = input('Input lowercase sentence:')`
attach server name, port to message; send into socket →	`clientSocket.sendto(message.encode(), (serverName, serverPort))`
read reply data (bytes) from socket →	`modifiedMessage, serverAddress = clientSocket.recvfrom(2048)`
print out received string and close socket →	`print(modifiedMessage.decode())`
	`clientSocket.close()`

Note: this code update (2023) to Python 3

EXAMPLE APP: UDP SERVER

Python UDPServer

```
                                                 from socket import *
                                                 serverPort = 12000
                     create UDP socket  ⟶  serverSocket = socket(AF_INET, SOCK_DGRAM)
      bind socket to local port number 12000  ⟶  serverSocket.bind(('', serverPort))
                                                 print('The server is ready to receive')
                          loop forever  ⟶  while True:
Read from UDP socket into message, getting  ⟶      message, clientAddress = serverSocket.recvfrom(2048)
client's address (client IP and port)
                                                     modifiedMessage = message.decode().upper()
  send upper case string back to this client  ⟶      serverSocket.sendto(modifiedMessage.encode(),
                                                                                 clientAddress)
```

Note: this code update (2023) to Python 3

SOCKET PROGRAMMING WITH TCP

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:

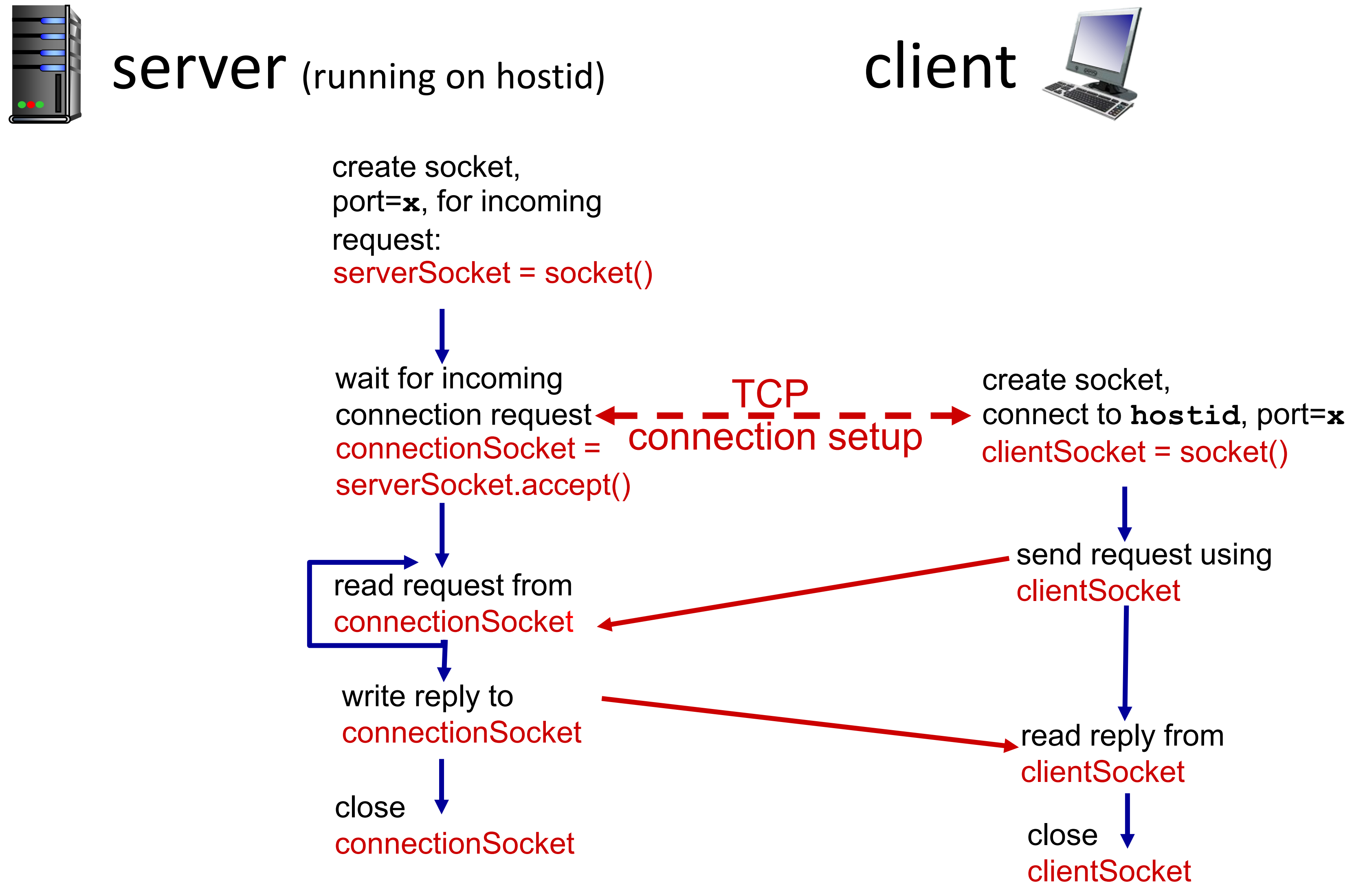
- creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - client source port # and IP address used to distinguish clients (more in Chap 3)

Application viewpoint

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server processes

CLIENT/SERVER SOCKET INTERACTION: TCP



server (running on hostid)

client

create socket,
port=**x**, for incoming
request:
serverSocket = socket()

wait for incoming
connection request
connectionSocket =
serverSocket.accept()

TCP
connection setup

create socket,
connect to **hostid**, port=**x**
clientSocket = socket()

send request using
clientSocket

read request from
connectionSocket

write reply to
connectionSocket

read reply from
clientSocket

close
connectionSocket

close
clientSocket

EXAMPLE APP: TCP CLIENT

Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

create TCP socket for server, remote port 12000 →

No need to attach server name, port →

Note: this code update (2023) to Python 3

EXAMPLE APP: TCP SERVER

Python TCPServer

create TCP welcoming socket →

server begins listening for incoming TCP requests →

loop forever →

server waits on accept() for incoming requests, new socket created on return →

read bytes from socket (but not address as in UDP) →

close connection to this client (but *not* welcoming socket) →

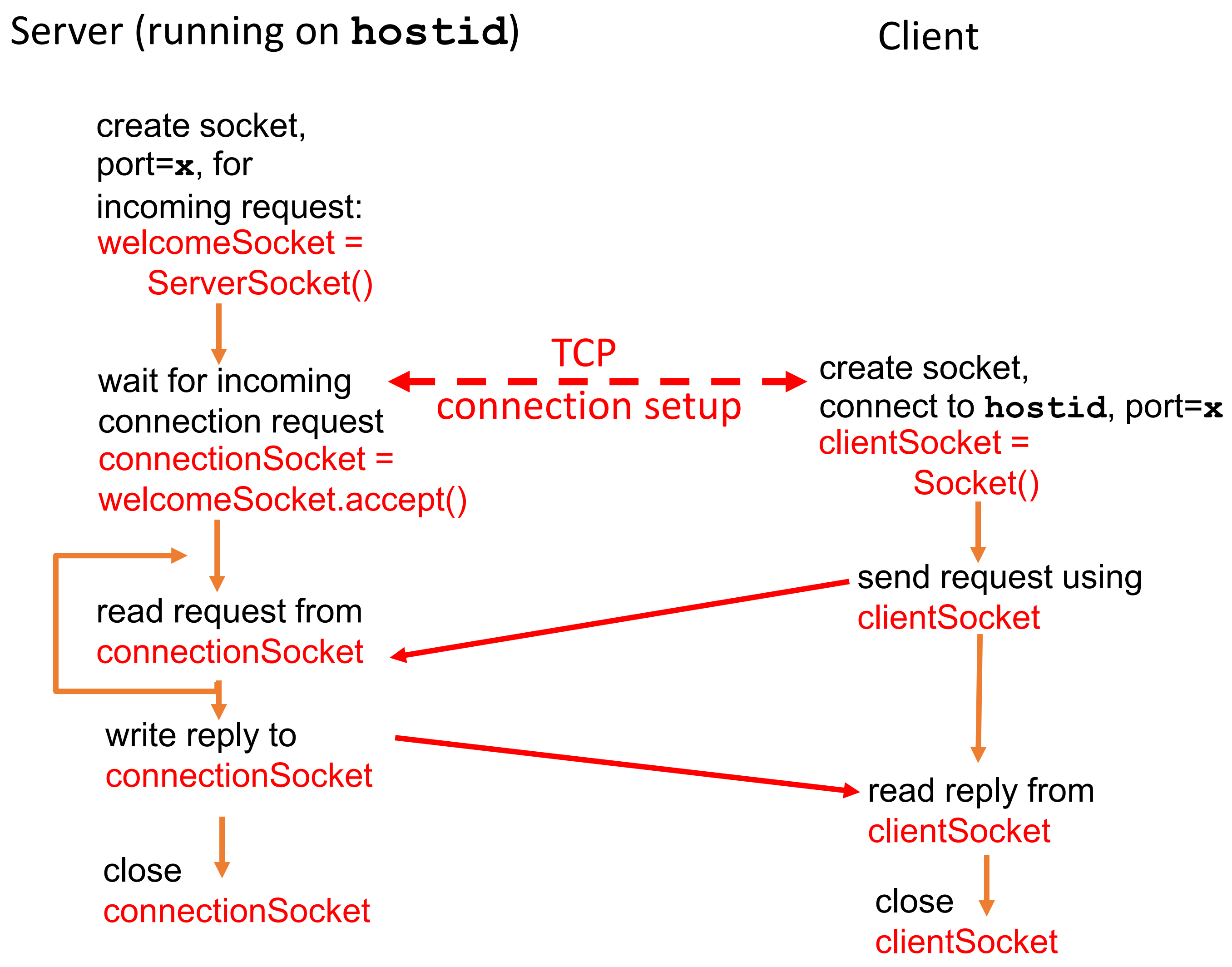
```
 from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print('The server is ready to receive')
while True:
     connectionSocket, addr = serverSocket.accept()

     sentence = connectionSocket.recv(1024).decode()
     capitalizedSentence = sentence.upper()
     connectionSocket.send(capitalizedSentence.
                                         encode())
     connectionSocket.close()
```

Note: this code update (2023) to Python 3

SOCKETS AND SOCKET PROGRAMMING

JAVA PROGRAMMING LANGUAGE

CLIENT/SERVER SOCKET INTERACTION: TCP



Server (running on **hostid**)
Client
create socket, port=**x**, for incoming request:
welcomeSocket = ServerSocket()
wait for incoming connection request
connectionSocket = welcomeSocket.accept()
TCP connection setup
create socket, connect to **hostid**, port=**x**
clientSocket = Socket()
send request using clientSocket
read request from connectionSocket
write reply to connectionSocket
read reply from clientSocket
close connectionSocket
close clientSocket

EXAMPLE APP: TCP CLIENT

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream

Create client socket, connect to server

Create output stream attached to socket

EXAMPLE APP: TCP CLIENT CONT.

Create input stream attached to socket

```
BufferedReader inFromServer =
  new BufferedReader(new
  InputStreamReader(clientSocket.getInputStream()));
```

```
sentence = inFromUser.readLine();
```

Send line to server

```
outToServer.writeBytes(sentence + '\n');
```

Read line from server

```
modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

    }
}
```

EXAMPLE APP: TCP SERVER

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

        Socket connectionSocket = welcomeSocket.accept();

       BufferedReader inFromClient =
          new BufferedReader(new
          InputStreamReader(connectionSocket.getInputStream()));
```

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

EXAMPLE APP: TCP SERVER CONT.

Create output stream, attached to socket

```
DataOutputStream  outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());
```

Read in line from socket

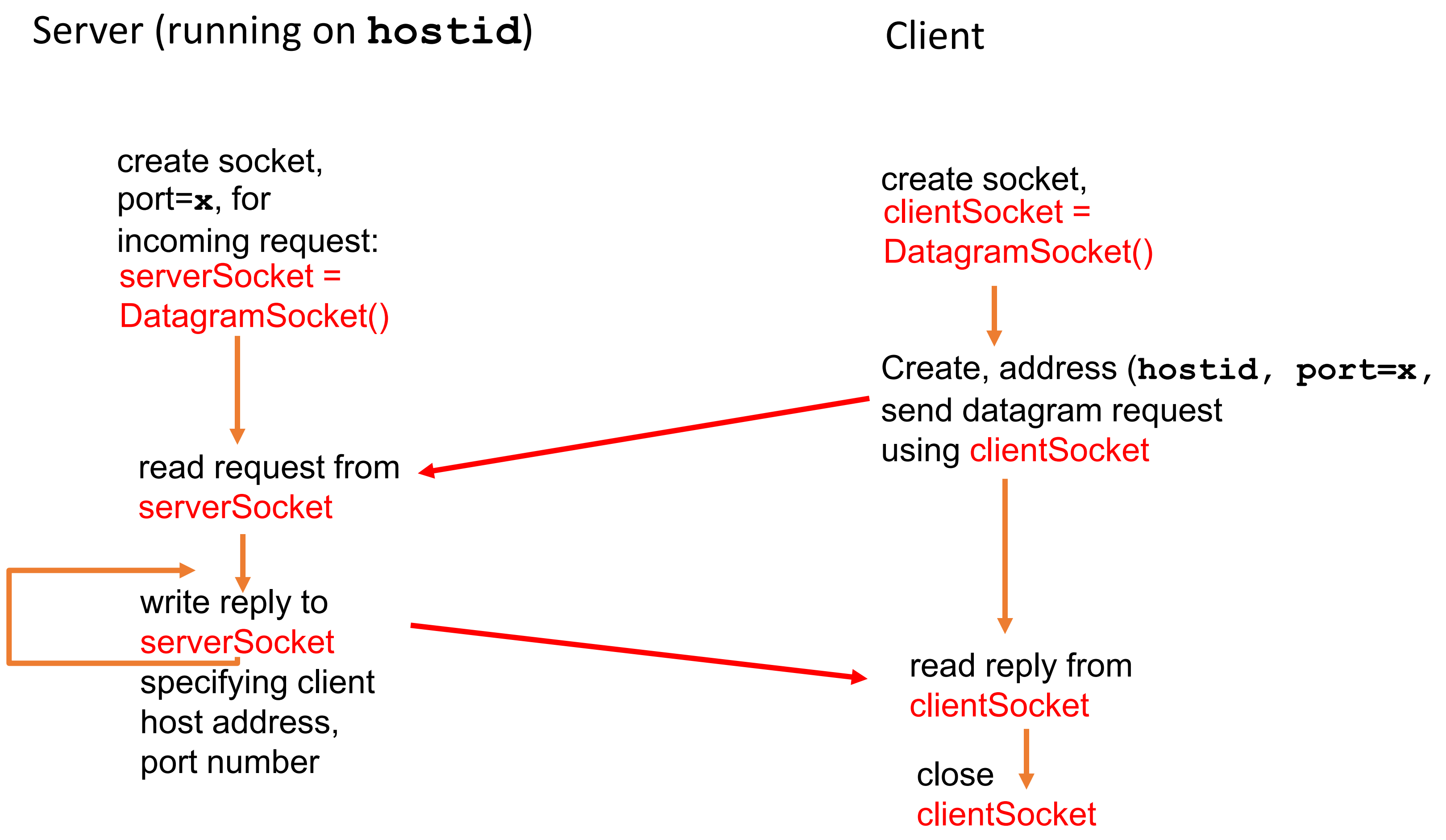
```
clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

Write out line to socket

```
outToClient.writeBytes(capitalizedSentence);
}
   }
}
```

End of while loop, loop back and wait for another client connection

CLIENT/SERVER SOCKET INTERACTION: UDP



Server (running on **hostid**)

create socket,
port=**x**, for
incoming request:
serverSocket =
DatagramSocket()

read request from
serverSocket

write reply to
serverSocket
specifying client
host address,
port number

Client

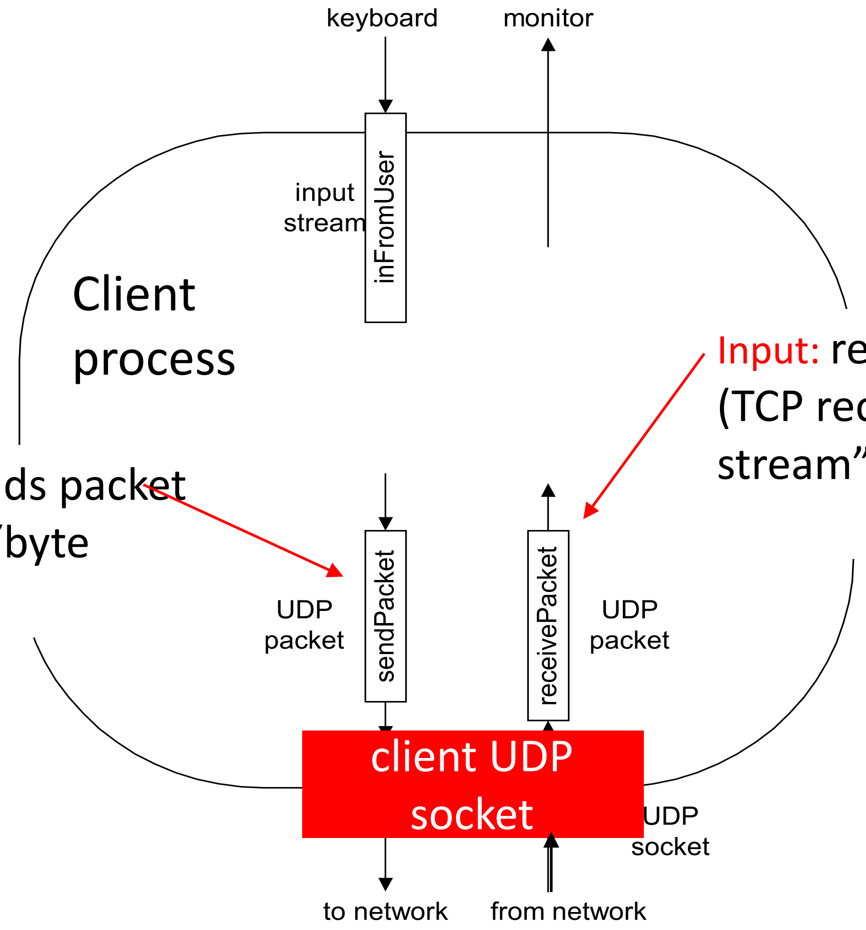
create socket,
clientSocket =
DatagramSocket()

Create, address (**hostid, port=x,**
send datagram request
using clientSocket

read reply from
clientSocket

close
clientSocket

EXAMPLE APP: UDP CLIENT



Output: sends packet (TCP sent "byte stream")

Input: receives packet (TCP received "byte stream")

EXAMPLE APP: UDP CLIENT

```
import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String args[]) throws Exception
    {

      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      DatagramSocket clientSocket = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName("hostname");

      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];

      String sentence = inFromUser.readLine();
      sendData = sentence.getBytes();
```

Create input stream

Create client socket

Translate hostname to IP address using DNS

EXAMPLE APP: UDP CLIENT CONT.

Create datagram with data-to-send, length, IP addr, port

Send datagram to server

Read datagram from server

```
DatagramPacket sendPacket =
   new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

clientSocket.send(sendPacket);

DatagramPacket receivePacket =
   new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

String modifiedSentence =
    new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
}
}
```

EXAMPLE APP: UDP SERVER

```
import java.io.*;
import java.net.*;

class UDPServer {
  public static void main(String args[]) throws Exception
    {

      DatagramSocket serverSocket = new DatagramSocket(9876);

      byte[] receiveData = new byte[1024];
      byte[] sendData  = new byte[1024];

      while(true)
        {

          DatagramPacket receivePacket =
             new DatagramPacket(receiveData, receiveData.length);
          serverSocket.receive(receivePacket);
```

Create datagram socket at port 9876

Create space for received datagram

Receive datagram

EXAMPLE APP: UDP SERVER CONT.

```
String sentence = new String(receivePacket.getData());

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

        String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =
   new DatagramPacket(sendData, sendData.length, IPAddress,
                  port);

serverSocket.send(sendPacket);
}
}
}
```

Get IP addr port #, of sender

Create datagram to send to client

Write out datagram to socket

End of while loop, loop back and wait for another datagram

REFERENCES

- Kurose, James F. (2013). Computer networking : a top-down approach. Pearson. TK5105.875.I57 K88 2013b