SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

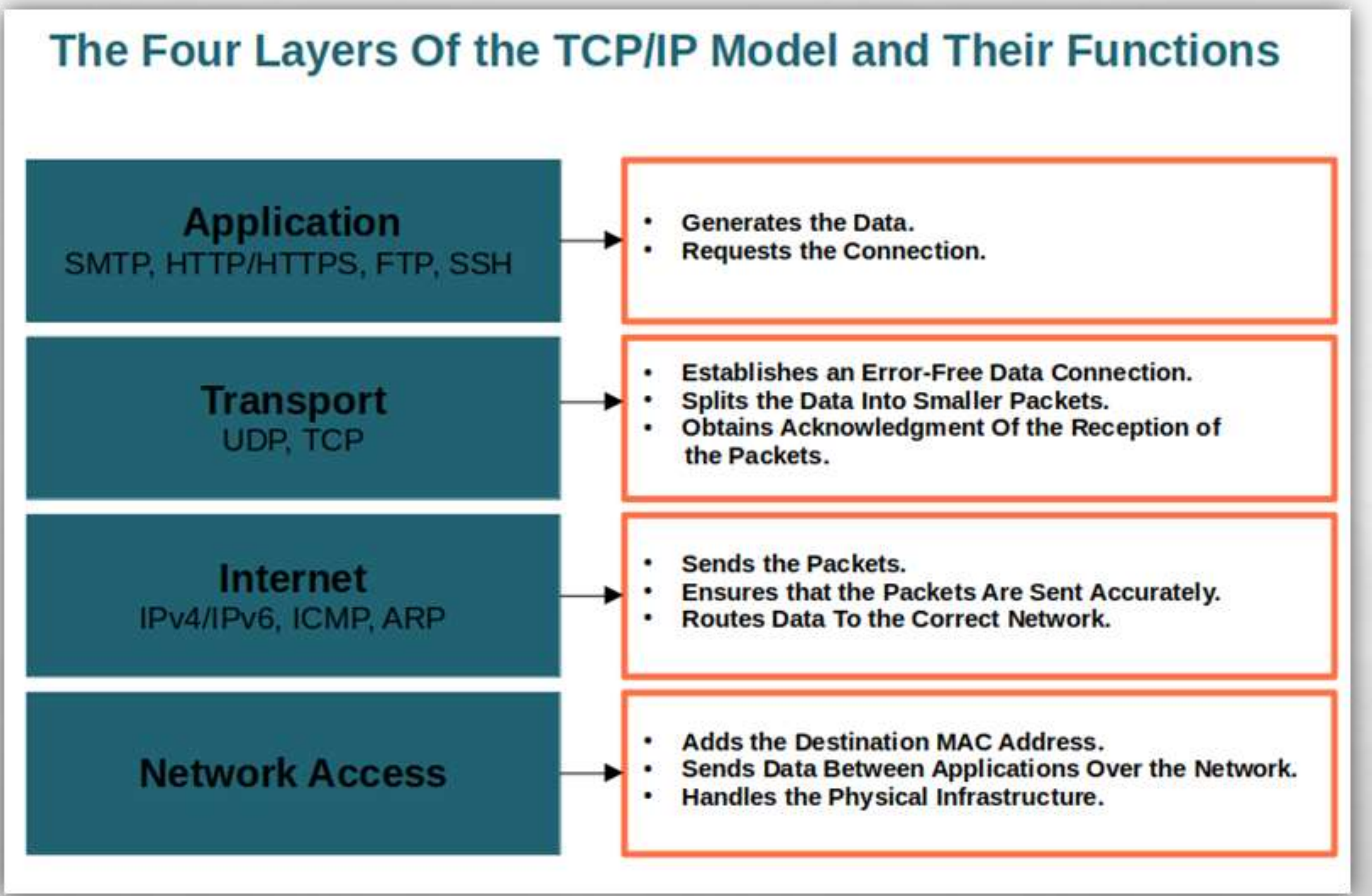
# DEPARTMENT OF MCA

## I YEAR II SEM

# 23CAT606– Java Programming

UNIT III –NETWORKING AND I/O PACKAGES

Topic 17: TCP/IP and datagram



## The Four Layers Of the TCP/IP Model and Their Functions

**Application**
SMTP, HTTP/HTTPS, FTP, SSH

- Generates the Data.
- Requests the Connection.

**Transport**
UDP, TCP

- Establishes an Error-Free Data Connection.
- Splits the Data Into Smaller Packets.
- Obtains Acknowledgment Of the Reception of the Packets.

**Internet**
IPv4/IPv6, ICMP, ARP

- Sends the Packets.
- Ensures that the Packets Are Sent Accurately.
- Routes Data To the Correct Network.

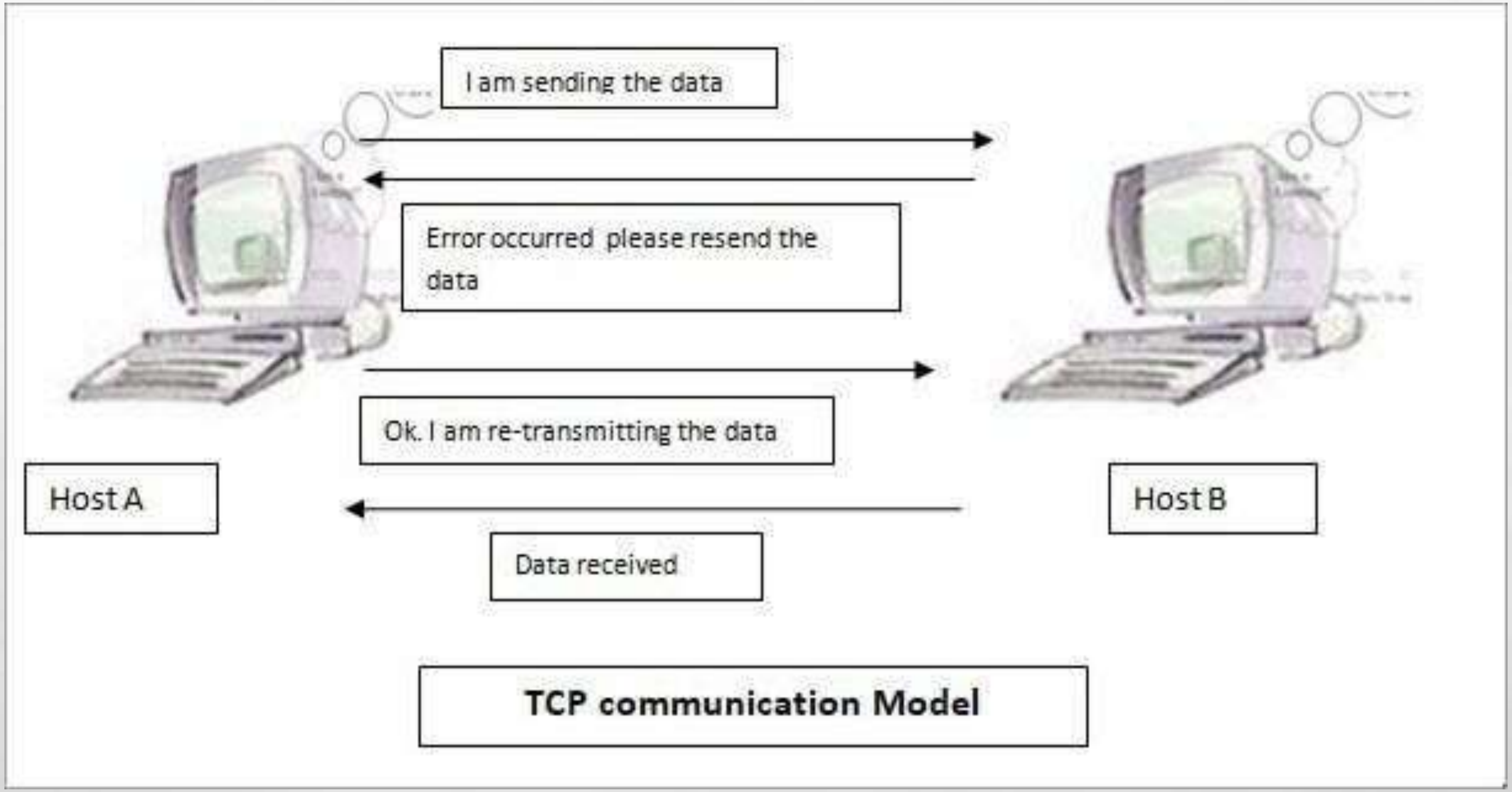
**Network Access**

- Adds the Destination MAC Address.
- Sends Data Between Applications Over the Network.
- Handles the Physical Infrastructure.

# INTRODUCTION: TCP/IP

1. The TCP/IP protocol is a set of protocols of four layers. Overall, these four layers take the responsibility of the communication process and end to end delivery of data, voice, packets over the internet on inter and intra network.
2. The Transmission Control Protocol (TCP) works on the third layer of this protocol model which is the transport layer.
3. TCP is a connection-oriented protocol suite that ensures the delivery of data packet to the next node or destination node by employing a sequence number in each datagram and acknowledgment sessions with each of the communication sessions.
4. This system also ensures secure transmission on each layer for the data packets and thereby provisions the retransmission of data packets unless it reaches a timeout situation or it receives the proper acknowledgment message from the receiver.

# TCP/IP

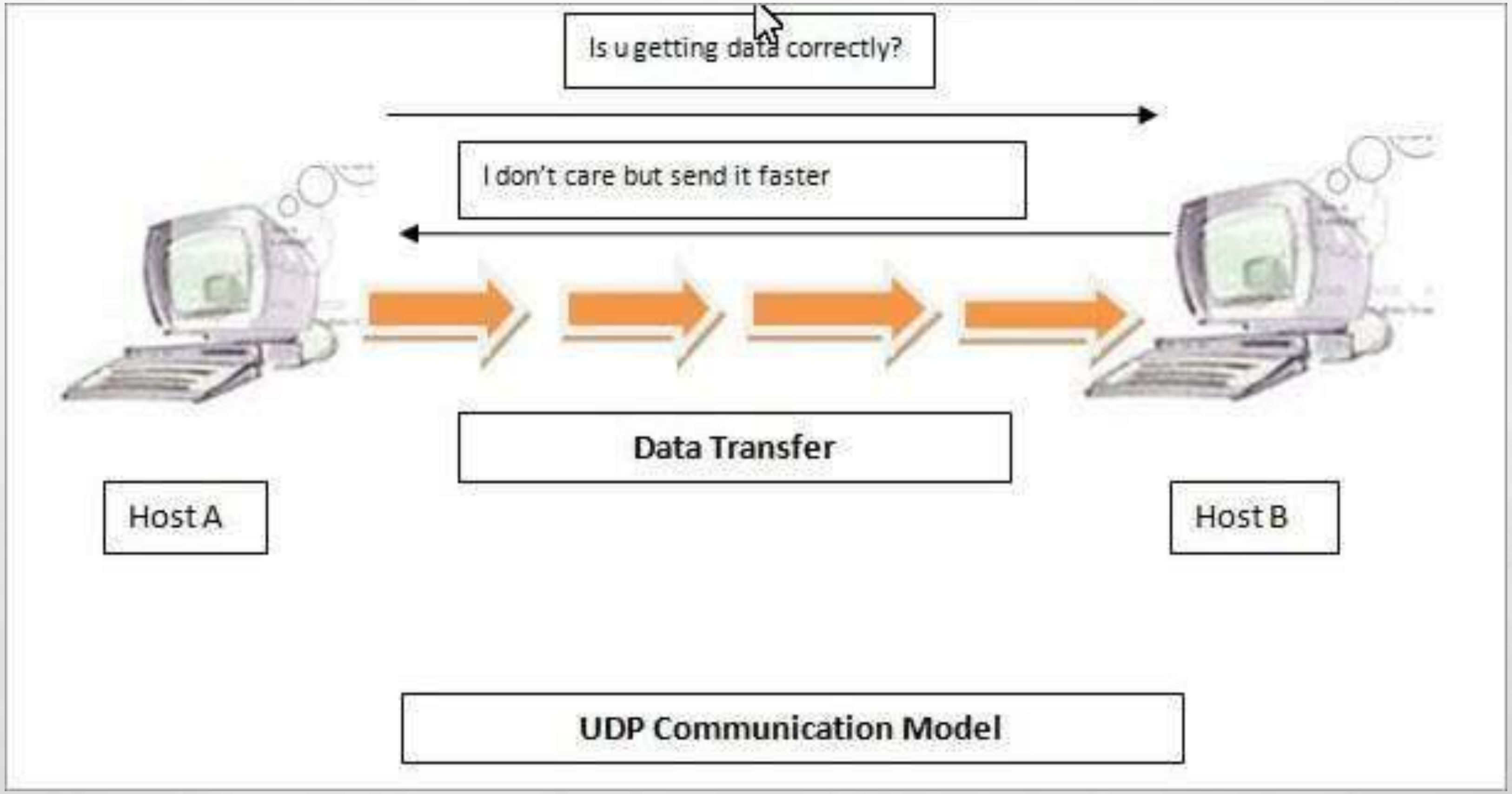

I am sending the data
Error occurred please resend the data
Ok. I am re-transmitting the data
Data received
Host A
Host B
TCP communication Model

# UDP

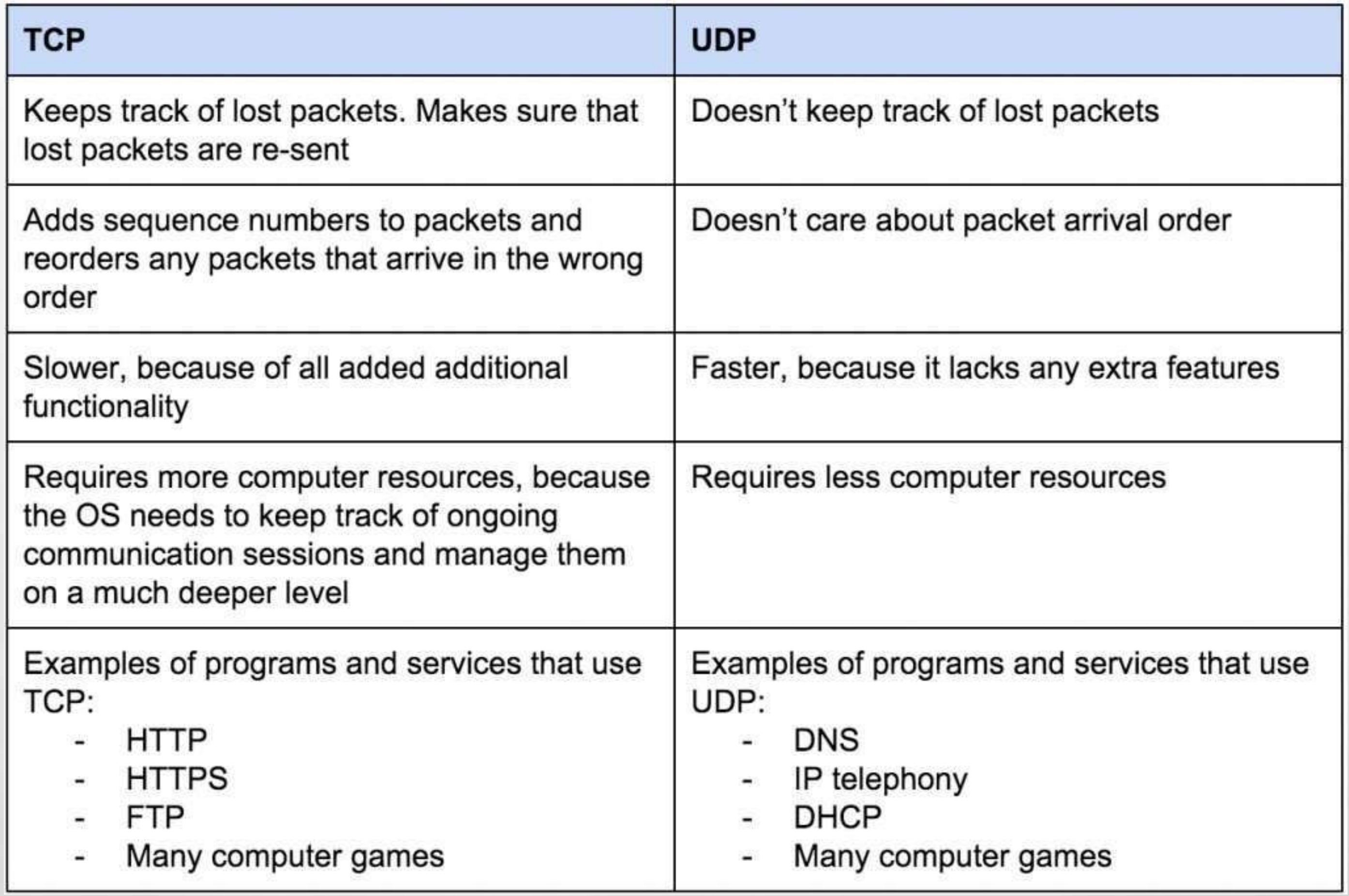
1. User Datagram Protocol (UDP) works on the transport layer which is the third layer of the TCP/IP protocol suite. In contrast to the TCP protocol, it is a connectionless protocol as it does not establish a connection before sending the data over the network for communication.

2. Thus it is best suited for the applications where there is no need for acknowledgments of the data packets required in the communication process such as watching videos online and playing games online.

# UDP

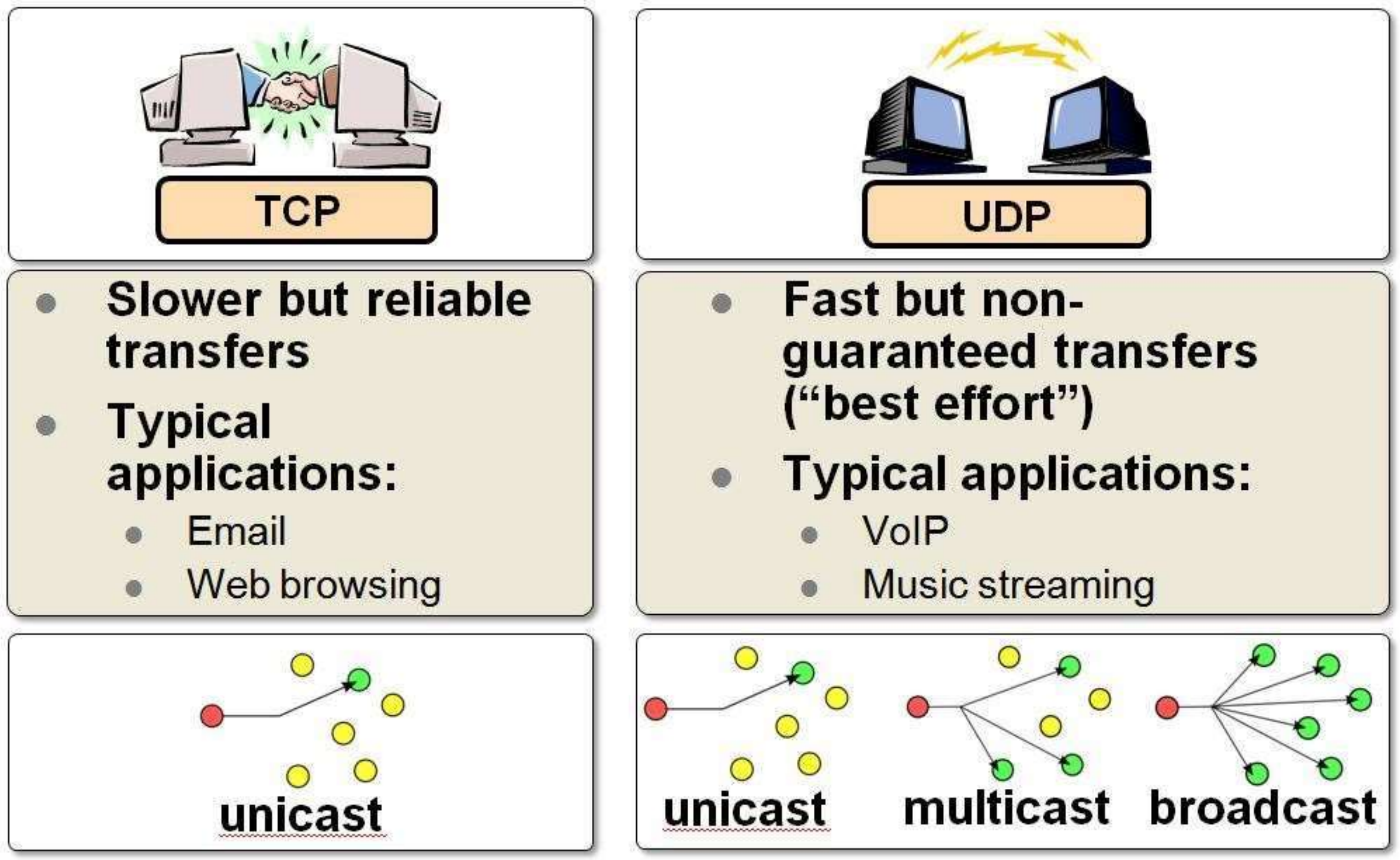

Is u getting data correctly?
I don't care but send it faster
Data Transfer
Host A
Host B
UDP Communication Model

# COMPARISON CHART TCP VS UDP



| TCP | UDP |
|---|---|
| Keeps track of lost packets. Makes sure that lost packets are re-sent | Doesn't keep track of lost packets |
| Adds sequence numbers to packets and reorders any packets that arrive in the wrong order | Doesn't care about packet arrival order |
| Slower, because of all added additional functionality | Faster, because it lacks any extra features |
| Requires more computer resources, because the OS needs to keep track of ongoing communication sessions and manage them on a much deeper level | Requires less computer resources |
| Examples of programs and services that use TCP:<br>- HTTP<br>- HTTPS<br>- FTP<br>- Many computer games | Examples of programs and services that use UDP:<br>- DNS<br>- IP telephony<br>- DHCP<br>- Many computer games |

# COMPARISON CHART TCP VS UDP



TCP

- Slower but reliable transfers
- Typical applications:
  - Email
  - Web browsing

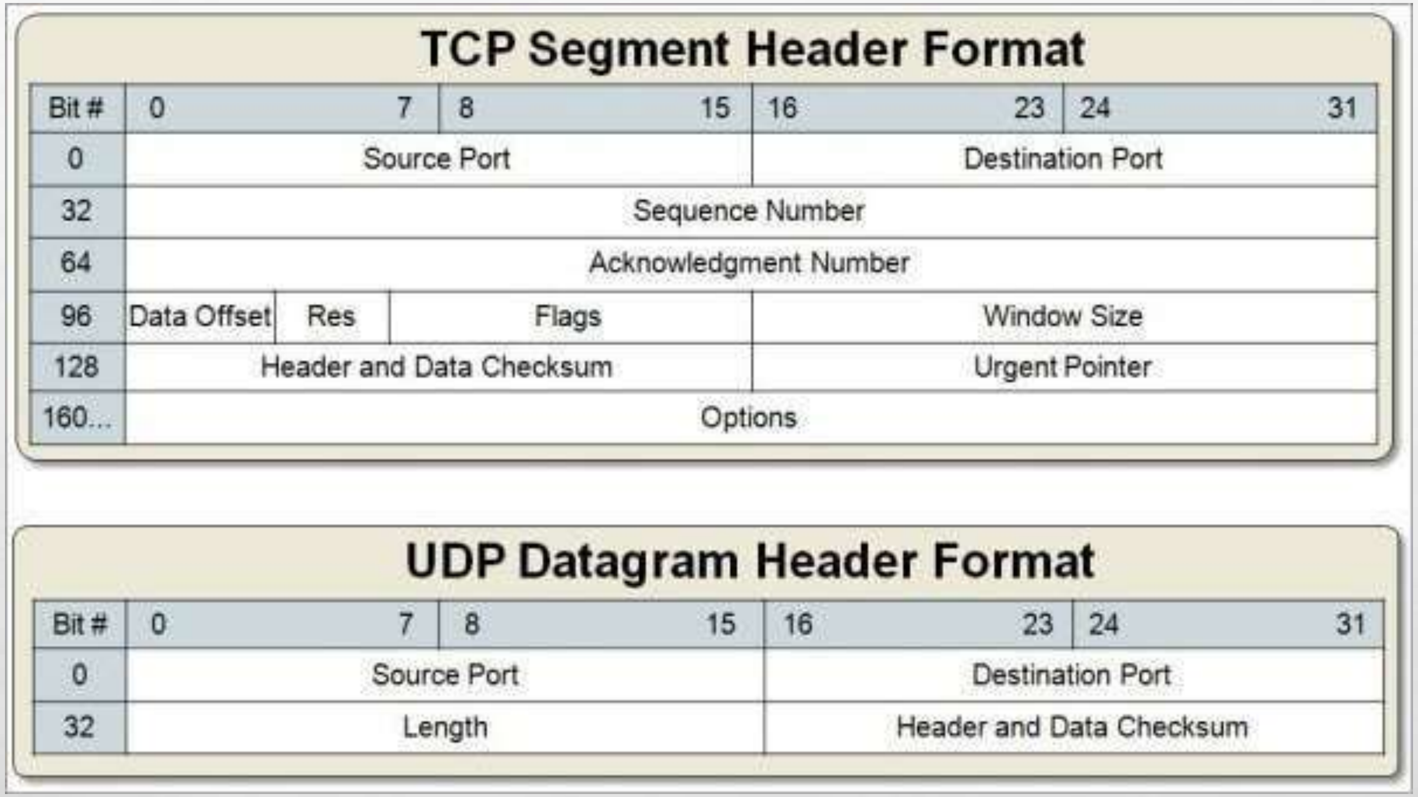
unicast

UDP

- Fast but non-guaranteed transfers (“best effort”)
- Typical applications:
  - VoIP
  - Music streaming

unicast multicast broadcast

# TCP UDP PACKET FORMAT



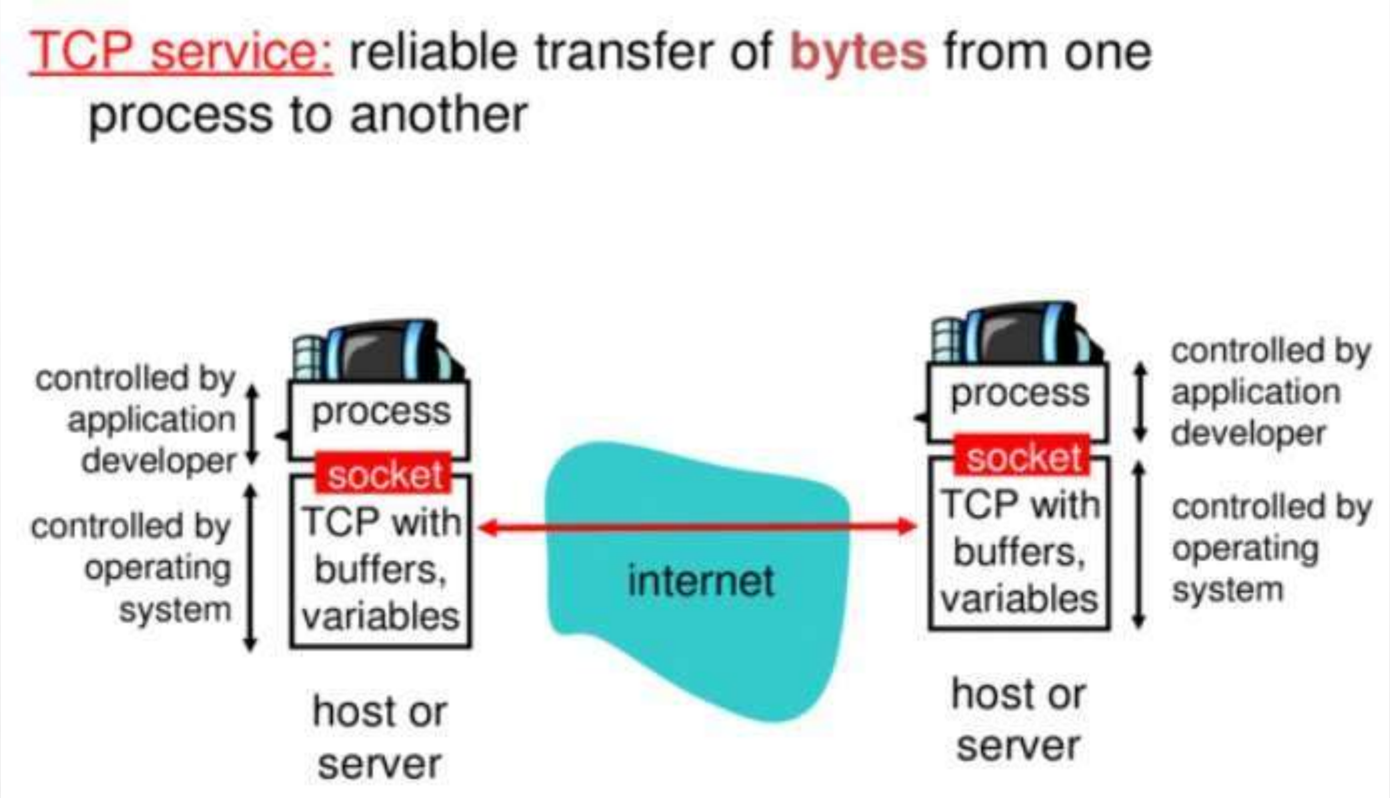
## TCP Segment Header Format

<table>
<tr><th>Bit #</th><th colspan="2">0 7</th><th>8 15</th><th>16 23</th><th>24 31</th></tr>
<tr><td>0</td><td colspan="3">Source Port</td><td colspan="2">Destination Port</td></tr>
<tr><td>32</td><td colspan="5">Sequence Number</td></tr>
<tr><td>64</td><td colspan="5">Acknowledgment Number</td></tr>
<tr><td>96</td><td>Data Offset</td><td>Res</td><td>Flags</td><td colspan="2">Window Size</td></tr>
<tr><td>128</td><td colspan="3">Header and Data Checksum</td><td colspan="2">Urgent Pointer</td></tr>
<tr><td>160...</td><td colspan="5">Options</td></tr>
</table>

## UDP Datagram Header Format

<table>
<tr><th>Bit #</th><th>0 7</th><th>8 15</th><th>16 23</th><th>24 31</th></tr>
<tr><td>0</td><td colspan="2">Source Port</td><td colspan="2">Destination Port</td></tr>
<tr><td>32</td><td colspan="2">Length</td><td colspan="2">Header and Data Checksum</td></tr>
</table>

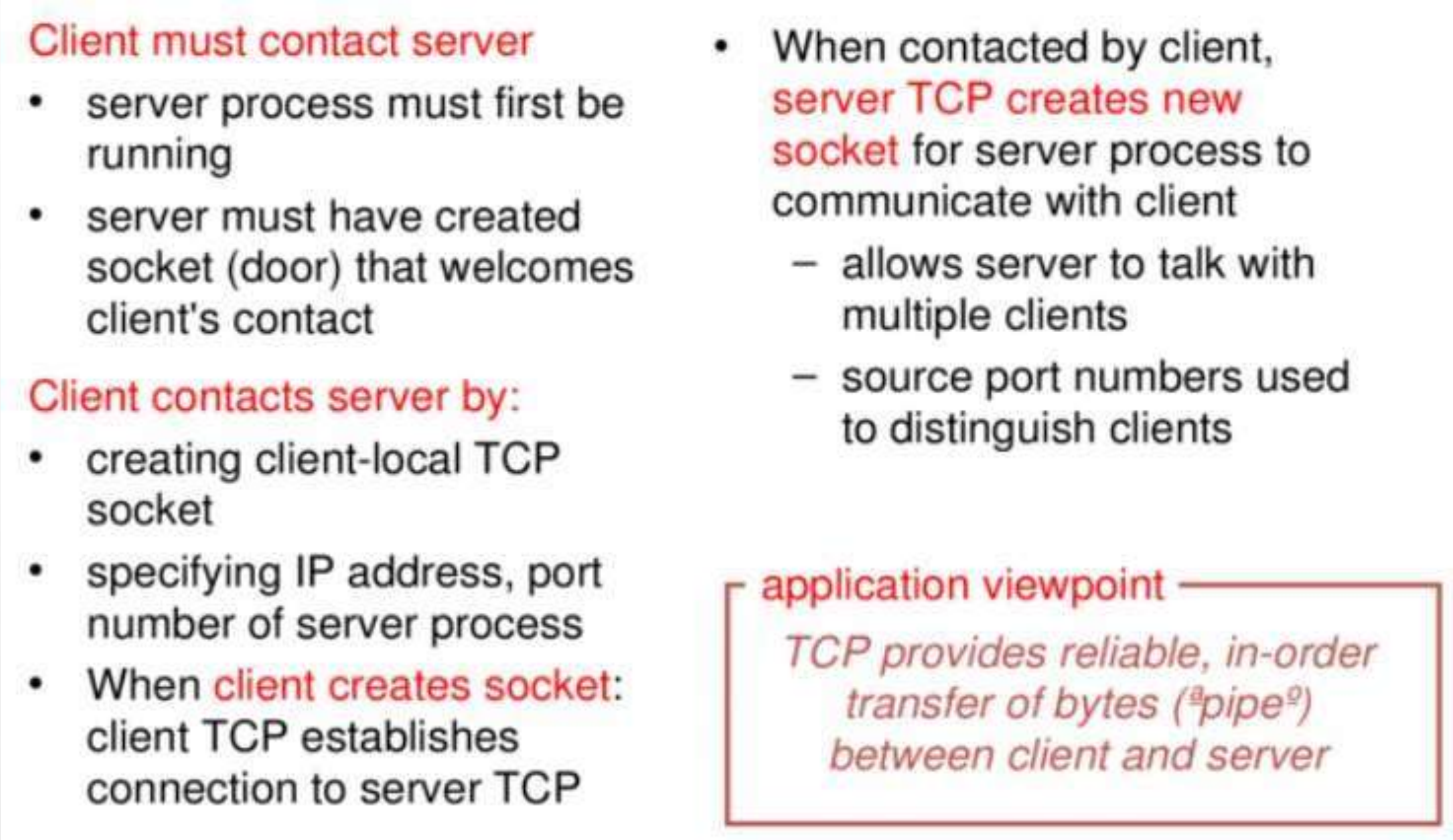
# SOCKET PROGRAMMING USING TCP



TCP service: reliable transfer of bytes from one process to another

controlled by application developer
controlled by operating system
process
socket
TCP with buffers, variables
host or server
internet
process
socket
TCP with buffers, variables
controlled by application developer
controlled by operating system
host or server

# SOCKET PROGRAMMING USING 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:

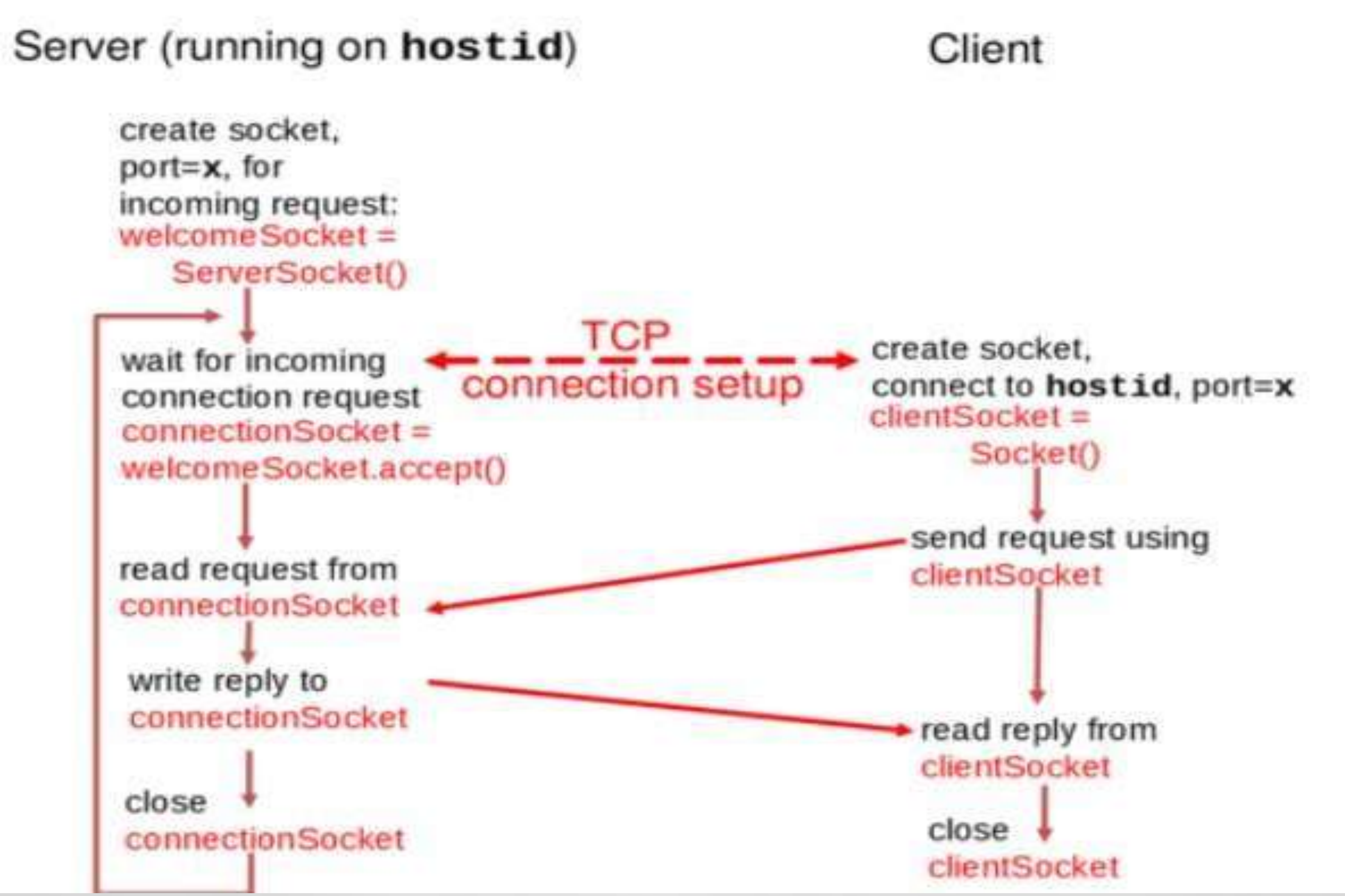
- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients

application viewpoint

*TCP provides reliable, in-order transfer of bytes (ªpipeº) between client and server*

# 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

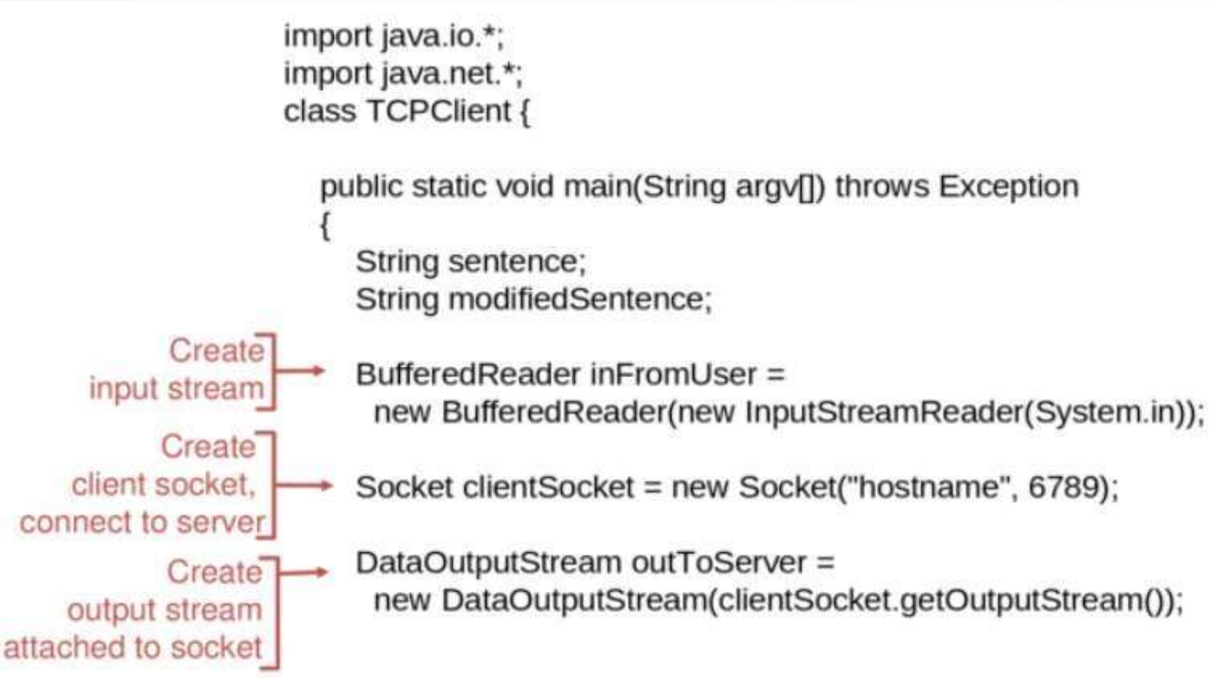
write reply to
connectionSocket

read reply from
clientSocket

close
connectionSocket

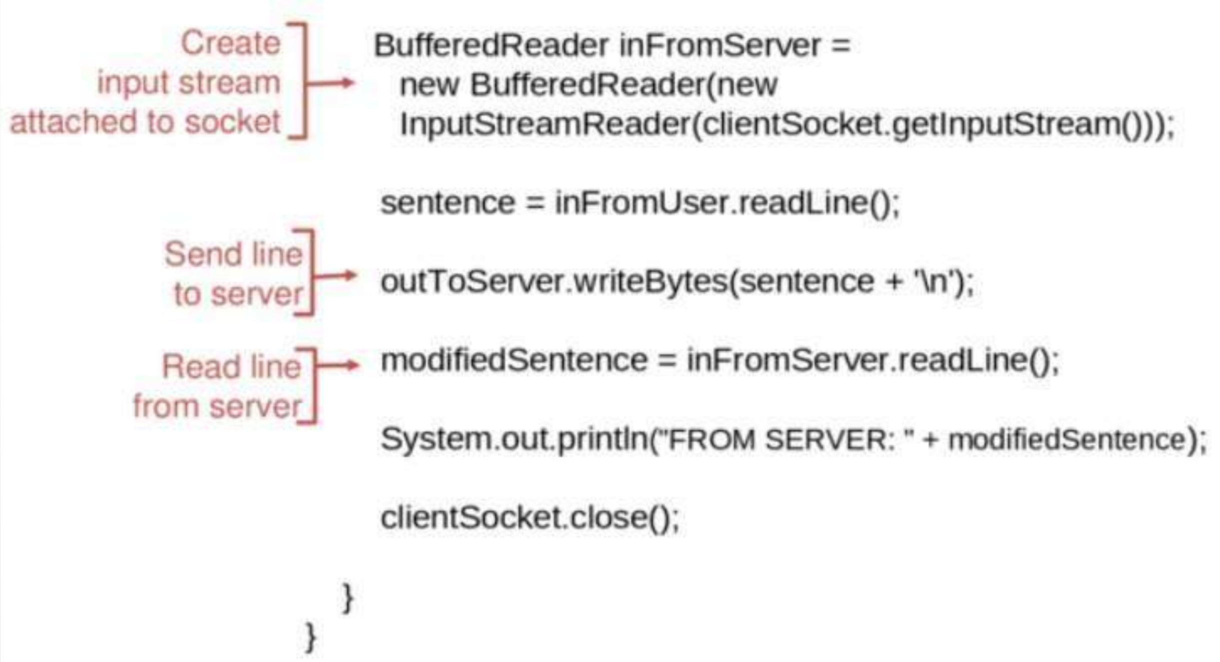
close
clientSocket

# JAVA CLIENT TCP



```
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());
```

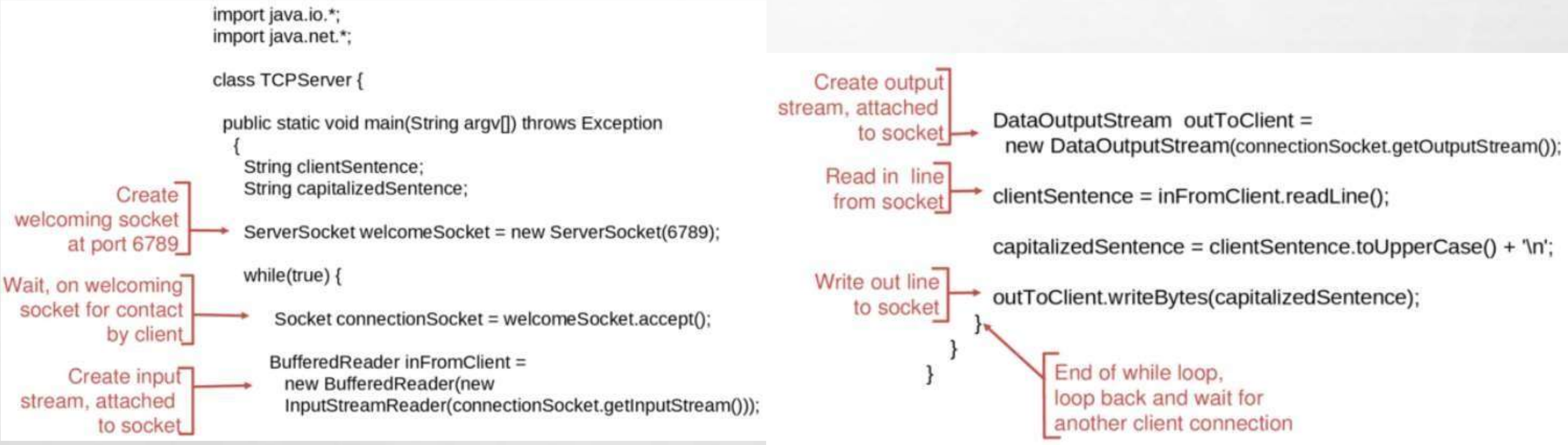
Create input stream
Create client socket, connect to server
Create output stream attached to socket



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

sentence = inFromUser.readLine();

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

modifiedSentence = inFromServer.readLine();

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

clientSocket.close();

    }
}
```

Create input stream attached to socket
Send line to server
Read line from server

# JAVA SERVER TCP



```
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()));

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

         clientSentence = inFromClient.readLine();

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

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

Create welcoming socket at port 6789
Wait, on welcoming socket for contact by client
Create input stream, attached to socket
Create output stream, attached to socket
Read in line from socket
Write out line to socket
End of while loop, loop back and wait for another client connection

# Reference

1. Herbert Schildt " The Complete Reference Java 2, 8th edition , Tata McGraw Hill, 2011
2. Ralph Bravaco, Shai Simonson, "Java Programming: From the Ground up Tata McGraw Hill, 2012
3. https://cupdf.com/document/udp-and-tcp-sockets-in-java.html