Socket Programming: Creating Network Applications

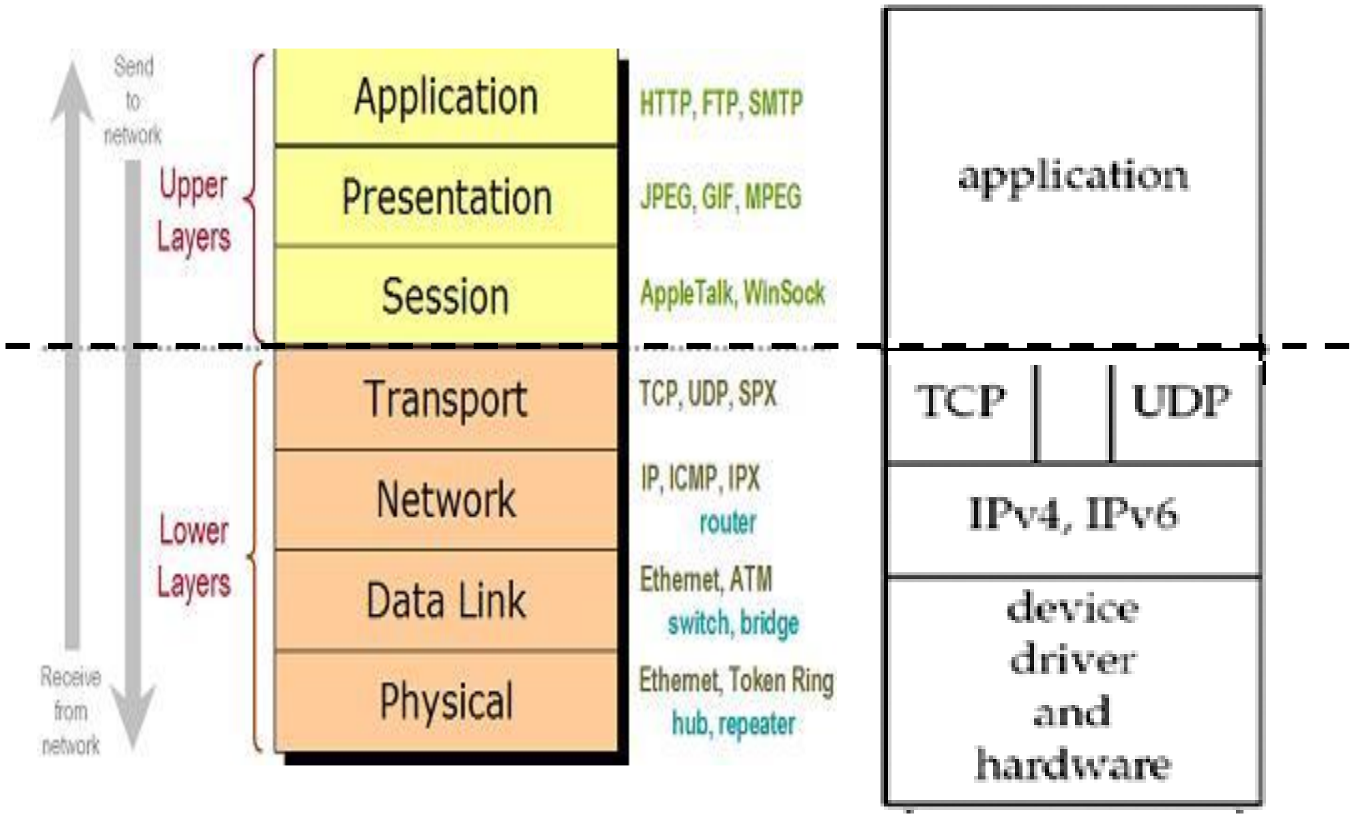
- Socket Programming with UDP
- Socket Programming with TCP

QUICK REVIEW

- What is a SOCKET?
  - To the kernel, a socket is an endpoint of communication.
  - To an application, a socket is a file descriptor that lets the application read/write from/to the network.
- Clients and servers communicate with each by reading from and writing to socket descriptors.
- Kinds of Sockets:
  - Datagram Sockets (or UDP Sockets).
  - Steam Sockets (or TCP Sockets)
  - Row Sockets (or Raw IP sockets)

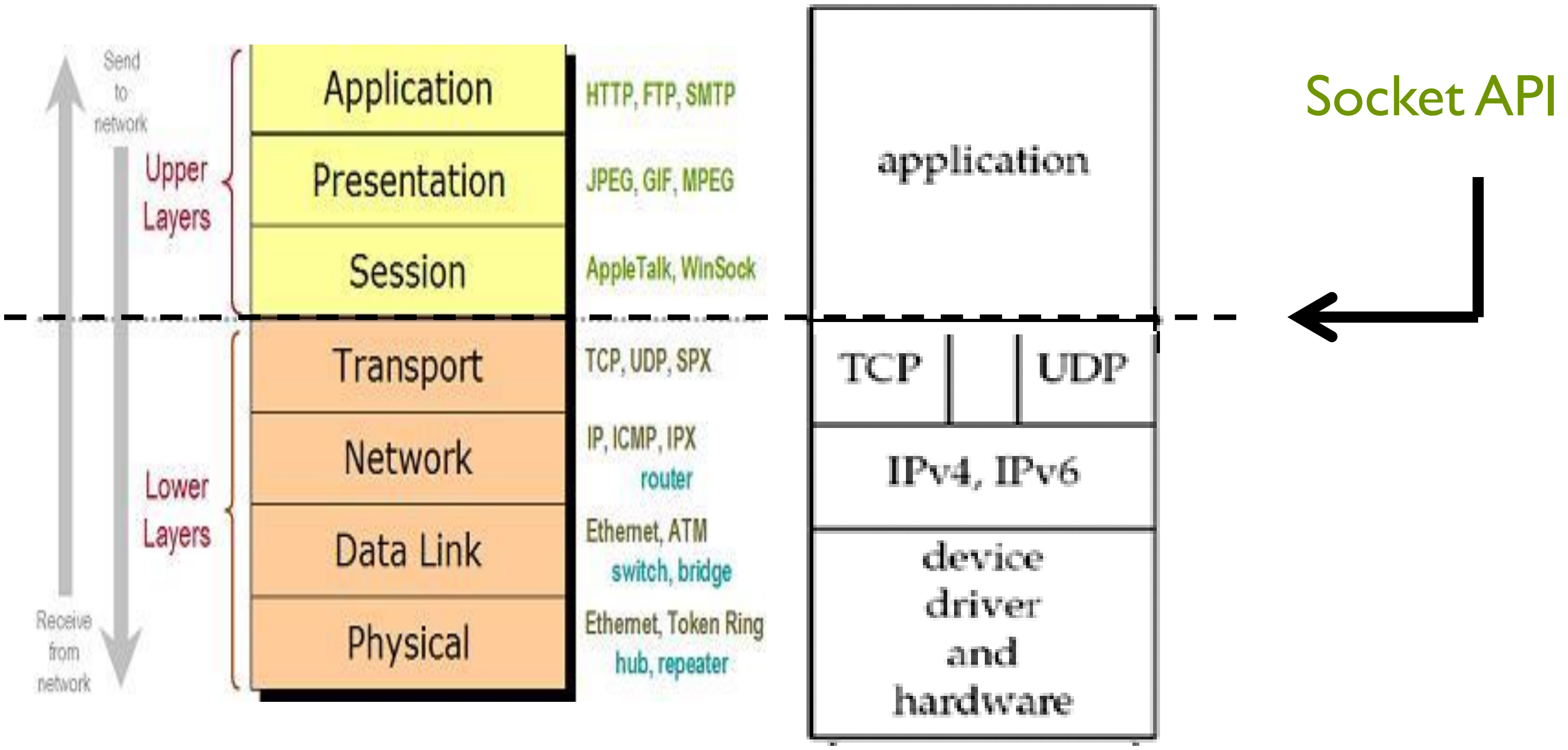
The Socket Interface :

Where is the socket programming interface in relation to the protocol stack?



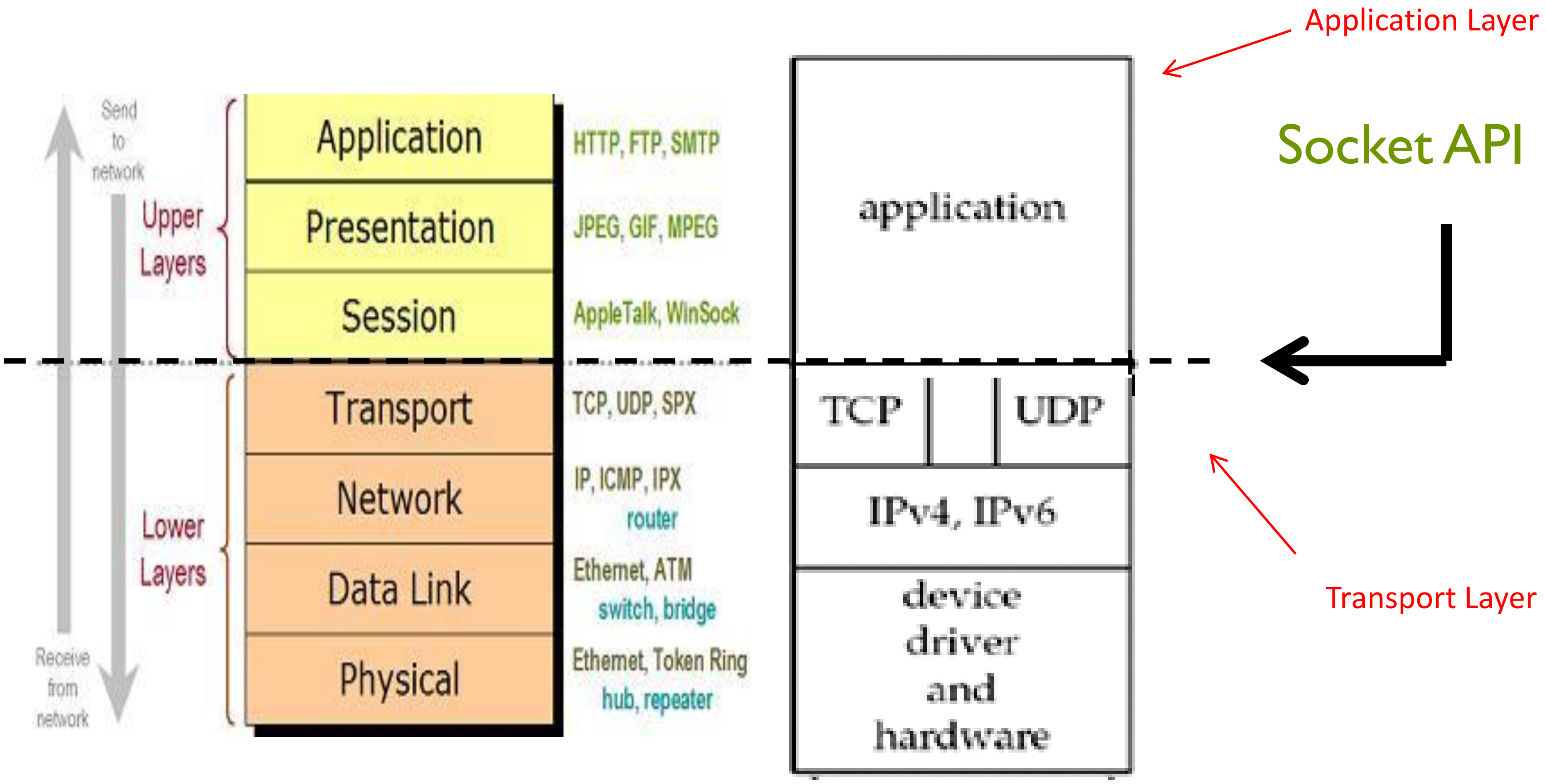
The Socket Interface :

Where is the socket programming interface in relation to the protocol stack?

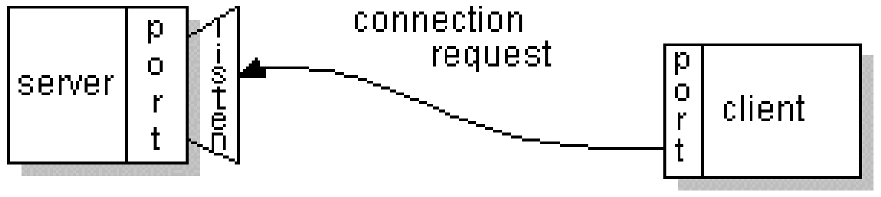


The Socket Interface :

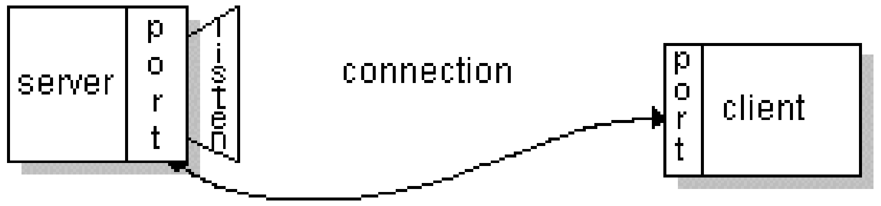
Where is the socket programming interface in relation to the protocol stack?



- On the server-side: A server has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.
- On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening.



- If everything goes well, the server accepts the connection. The server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client.



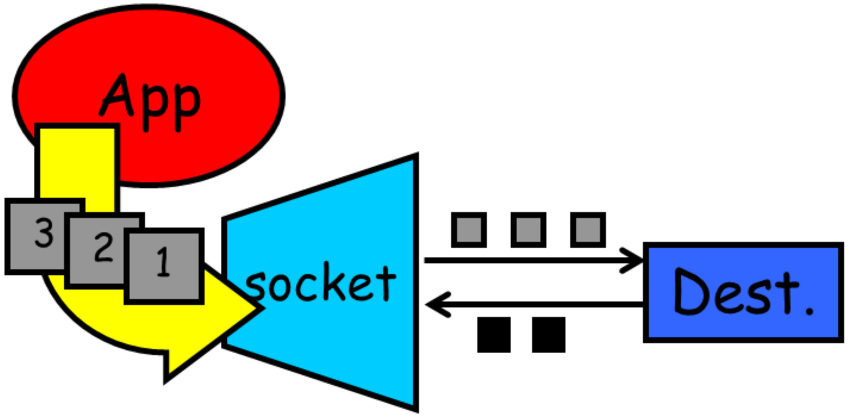
- On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.
- The client and server can now communicate by writing to or reading from their sockets.

TCP versus UDP as a Transport Layer Protocol:

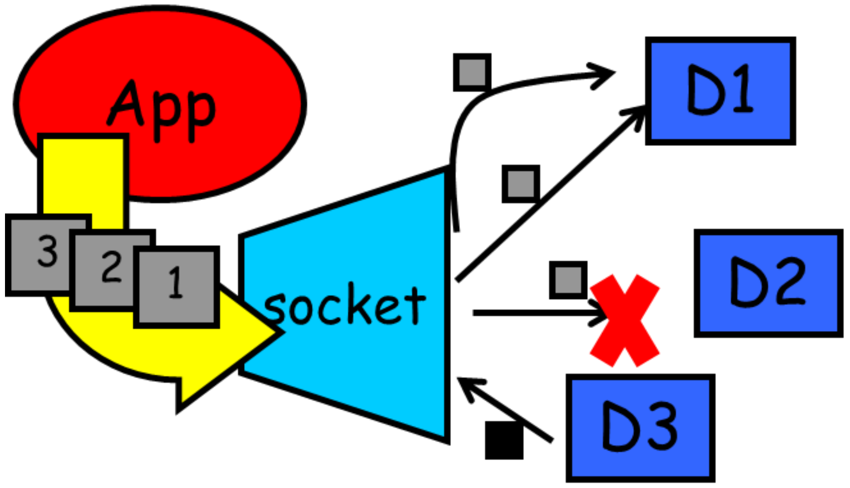
TCP	UDP
Reliable, guaranteed	Unreliable. Instead, prompt delivery of packets.
Connection-Oriented	Connectionless
Used in applications that require safety guarantee. (e.g. file applications.)	Used in media applications. (e.g. video or voice transmissions.)
Flow control, sequencing of packets, error-control.	No flow or sequence control, handled manually.
Uses byte stream as unit of transfer. (stream sockets)	Uses datagrams as unit of transfer. (datagram sockets)
Allows two-way data exchange, once the connection is established. (full-duplex)	Allows data to be transferred in one direction at once. (half-duplex)

TCP Vs UDP

TCP

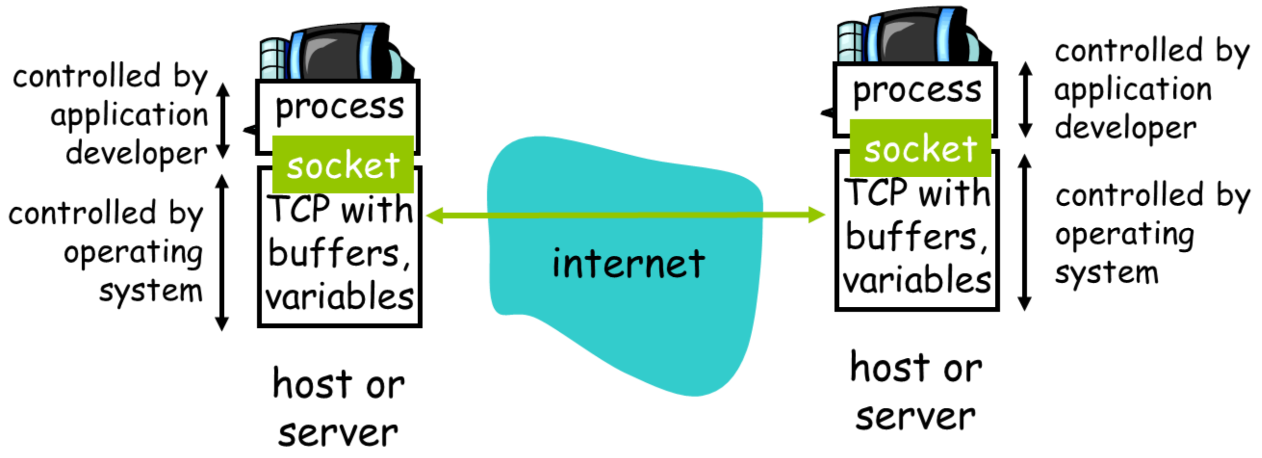


UDP



Socket Programming

- Why Programming Sockets?
  - Creating network applications needs sockets to communicate with client/server.
- Basics you should know about.
  - Two types of network applications.
  - TCP or UDP?
  - Programming languages?
- The application developer has control of everything on the application-layer side of the socket; however, it has little control of the transport-layer side.

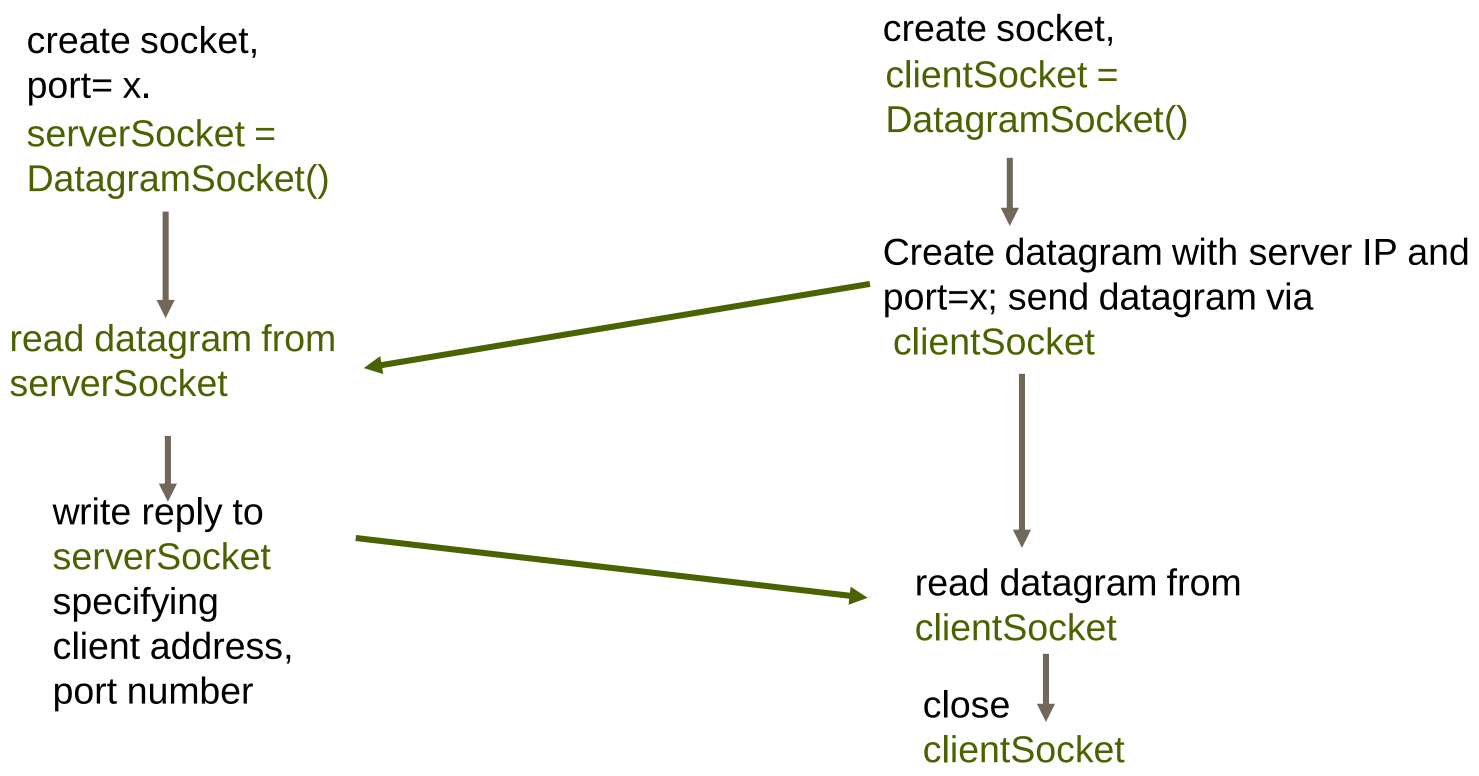

controlled by application developer
controlled by operating system
process
socket
TCP with buffers, variables
host or server
internet
process
socket
TCP with buffers, variables
host or server
controlled by application developer
controlled by operating system

Socket Programming with UDP

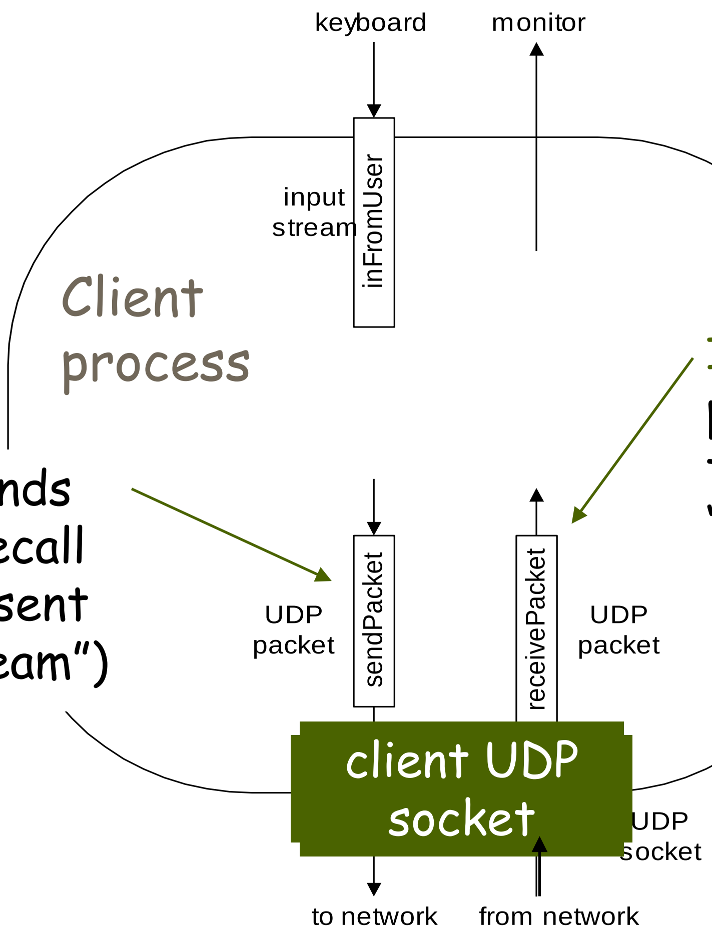
- UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server.
  - A **datagram** is a basic transfer unit associated with a packet-switched network. The delivery, arrival time, and order of arrival need not be guaranteed by the network.
- No handshaking.
- Sender explicitly attaches IP address and port of destination to each packet.
- Server must extract IP address, port of sender from received packet.

Client/server socket interaction: UDP

Server (running on `hostid`) **Client**



Example: Java client (UDP)



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

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

Example: Java client (UDP)

Create input stream

Create client socket

Translate hostname to IP address using DNS

```
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();
```

Example: Java client (UDP), cont.

Create datagram with data-to-send, length, IP adder, 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: Java server (UDP)

```
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: Java server (UDP), 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

Socket-programming using TCP

- reliable transfer of **bytes** from one process to another.

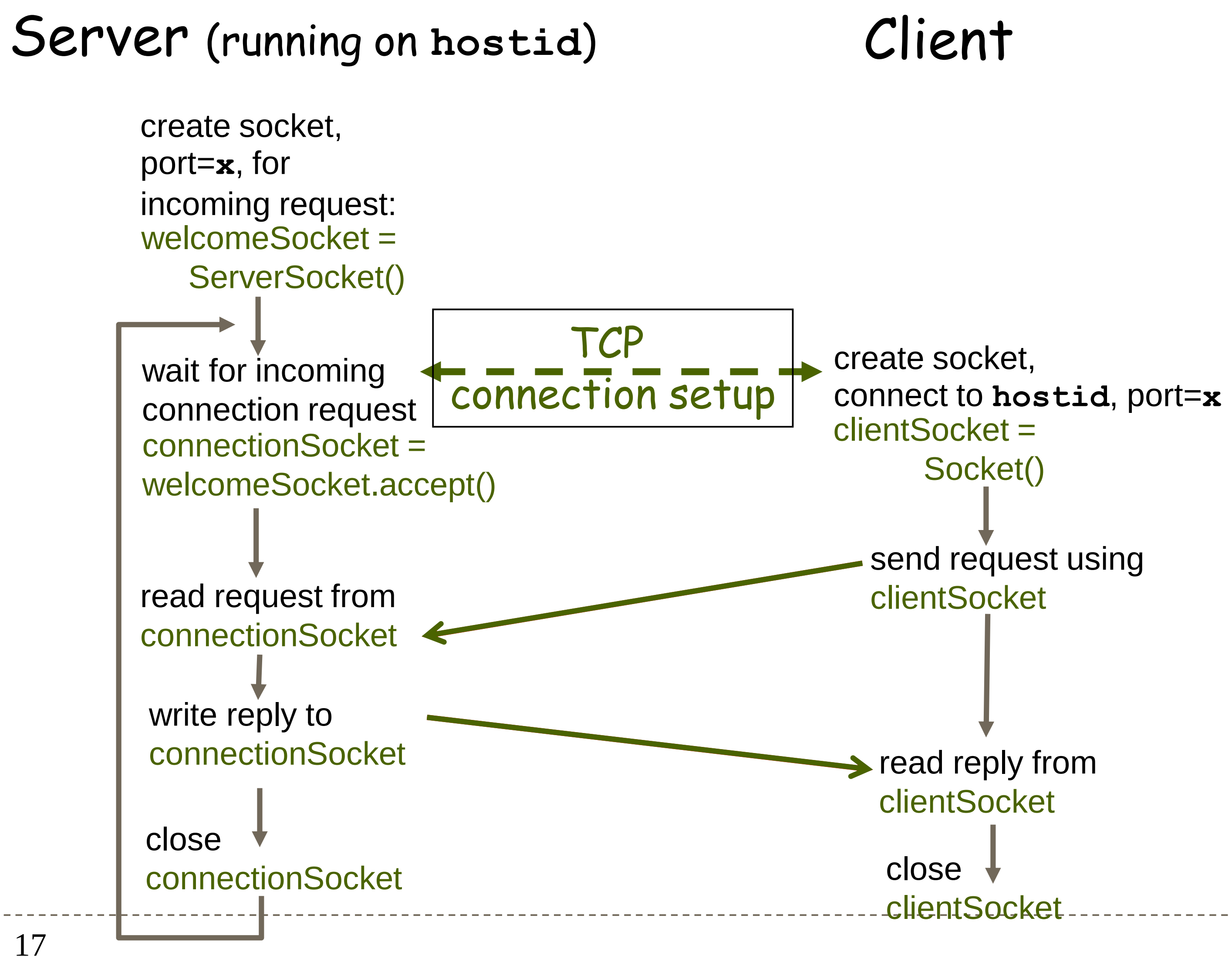
Client must contact server

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

Client contacts server by:

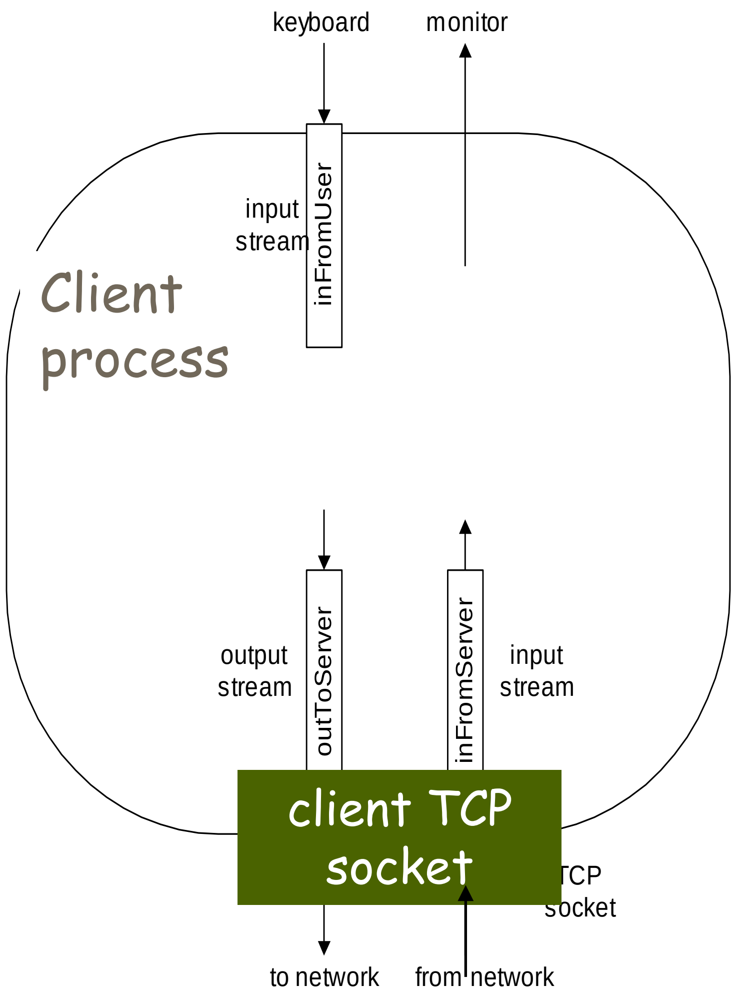
- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP
- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients

Client/server socket interaction: TCP


Server (running on hostid)
Client
create socket, port=x, for incoming request:
welcomeSocket = ServerSocket()
TCP connection setup
wait for incoming connection request
connectionSocket = welcomeSocket.accept()
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

Stream jargon

- A stream is a sequence of characters that flow into or out of a process.
- An input stream is attached to some input source for the process, e.g., keyboard or socket.
- An output stream is attached to an output source, e.g., monitor or socket.



Socket programming with TCP

Example client-server app:

1) client reads line from standard input (**`inFromUser`** stream) , sends to server via socket (**`outToServer`** stream)
2) server reads line from socket
3) server converts line to uppercase, sends back to client
4) client reads, prints modified line from socket (**`inFromServer`** stream)

Example: 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

Example: Java client (TCP), 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: 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()));
```

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

Example: Java server (TCP), 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