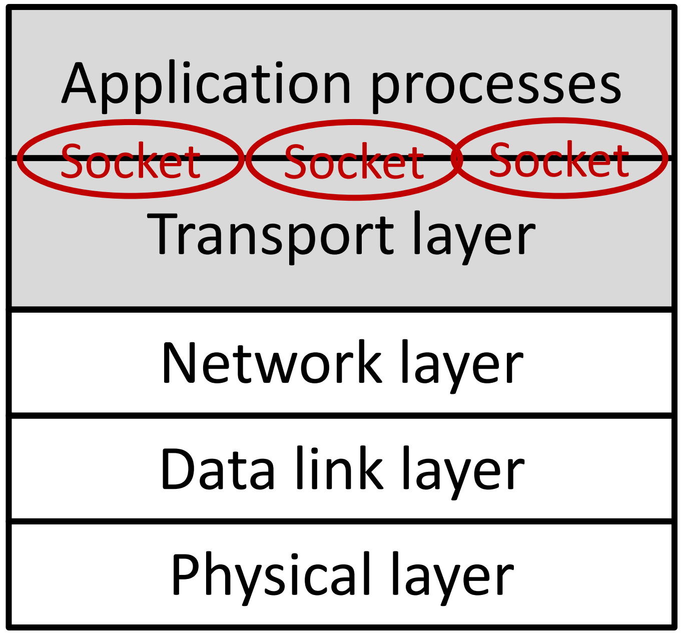
Lab 2: Introduction to Socket Programming

Socket

- Socket: A host-local, application-created, OS-controlled interface (a "door") between application process and end-to-end transport
  - Door, through which data passes from the network to a process and through which data passes from the process to the network
  - There can be many processes running on a host, using different sockets for transmission.
  - Each socket must have a unique identifier, which depends on whether the socket is a UDP or a TCP socket.

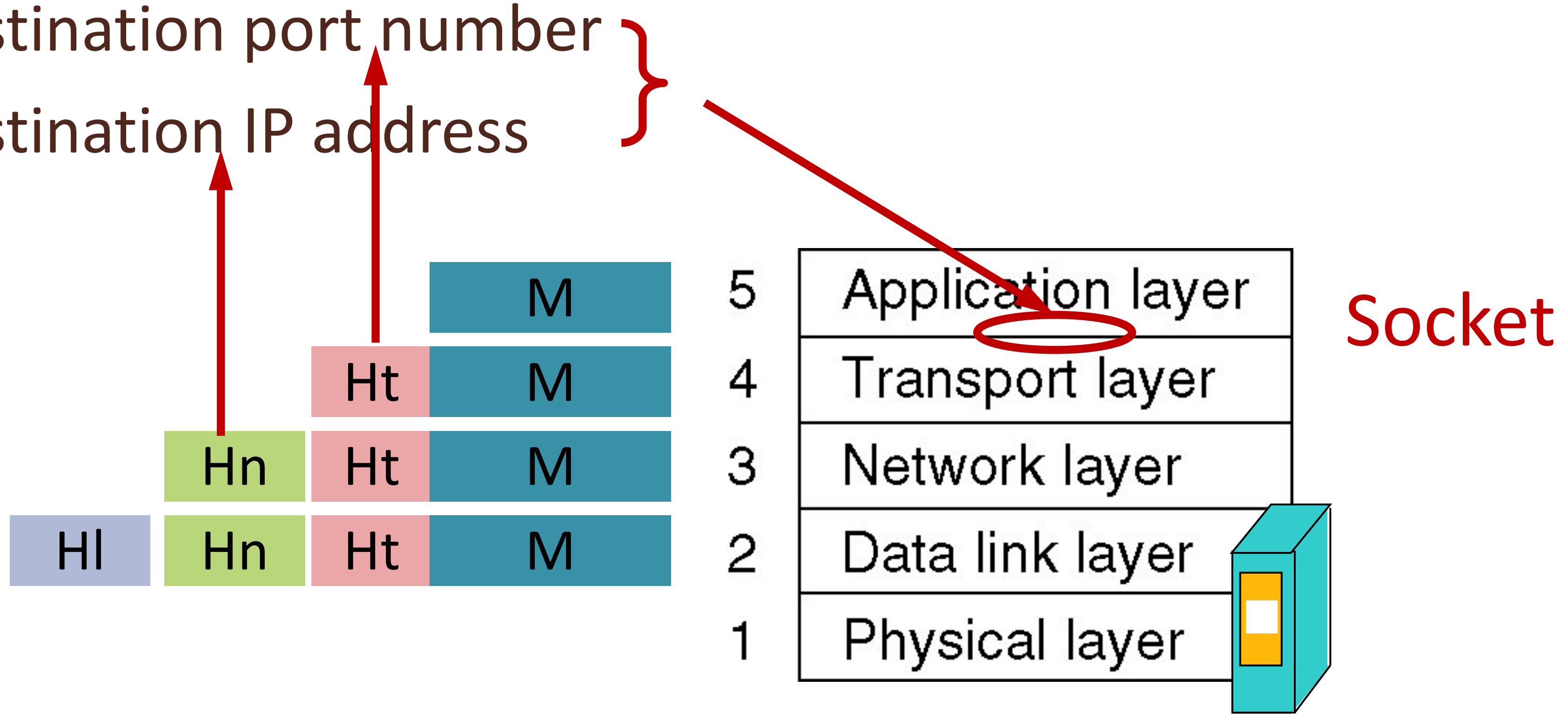


Transport-Layer Protocols

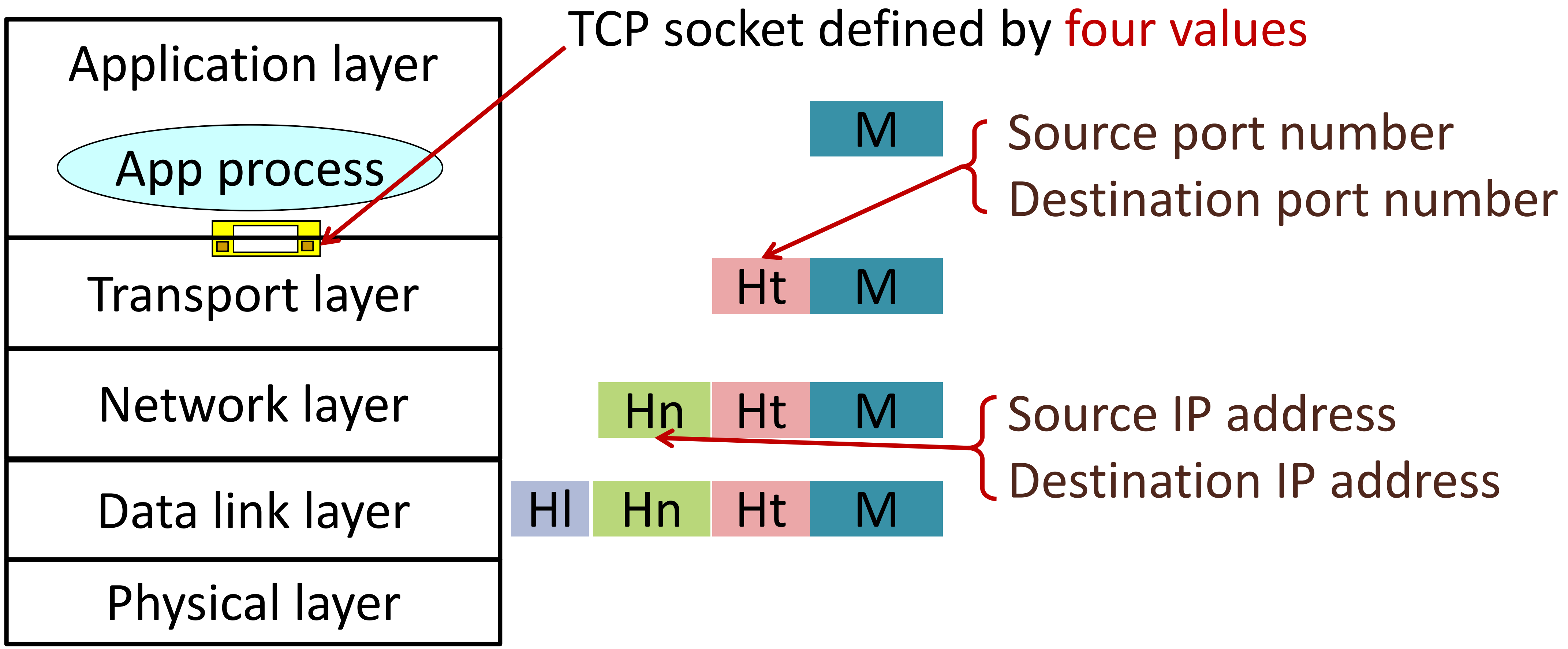
- Two types of transport protocols
  - Connectionless: User datagram protocol (UDP)
  - Connection-oriented: Transport control protocol (TCP)

UDP Socket

- In UDP, a socket is fully identified by a two-tuple:
  - A destination port number
  - A destination IP address

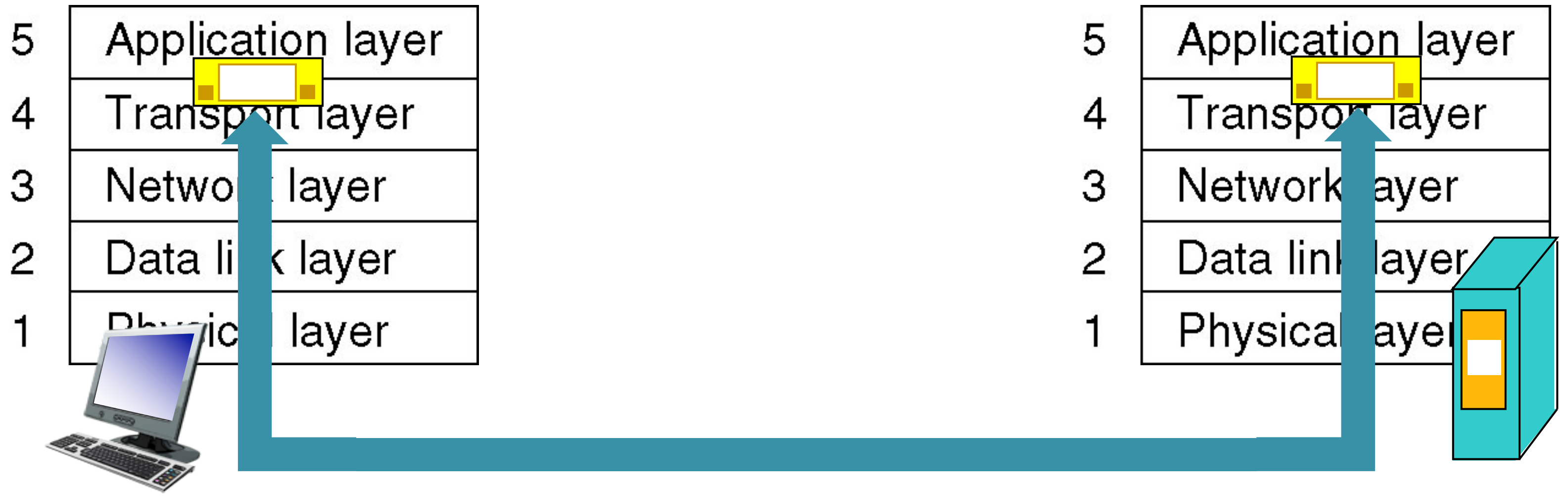


TCP Socket


TCP socket defined by four values
Application layer
App process
Transport layer
Network layer
Data link layer
Physical layer
M
Ht
Hn
Hl
Source port number
Destination port number
Source IP address
Destination IP address

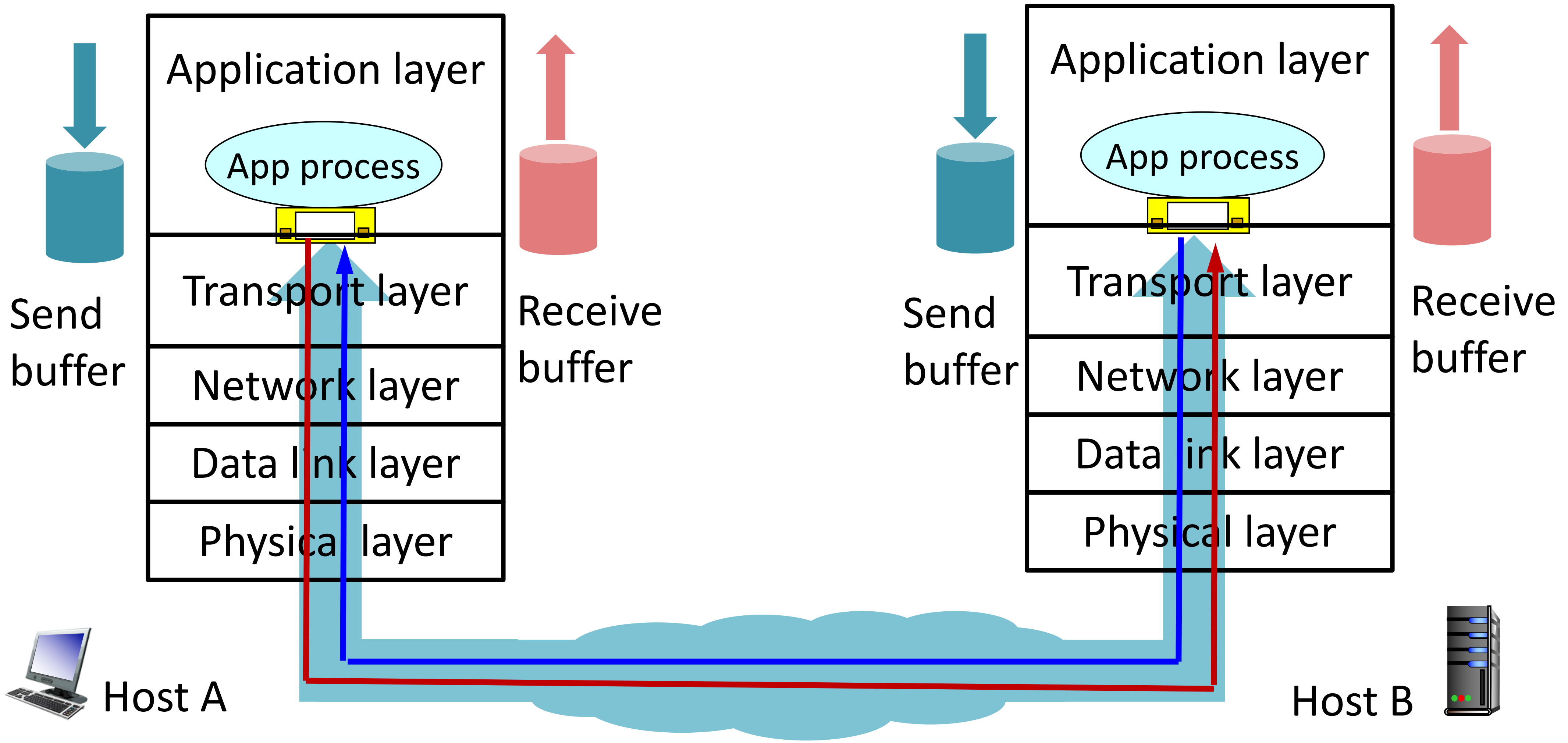
TCP Connection

- Connection is identified by two sockets on both ends: <socket1, socket2>
  - Unicast (point-to-point), no support for multicast or broadcast
  - Bidirectional (full-duplex)



TCP connection like a transmission pipe

Send and Receiver Buffers of TCP



Socket Programming Example

- Socket programming: Create network application programs using sockets
- Example: A client/server echo application
  - Client reads a line of characters (data) from the keyboard and sends the data to the server.
  - Server receives the data and converts characters to uppercase.
  - Server sends the modified characters to the client.
  - Client receives the modified characters and displays them on the screen.

Socket Programming with UDP

- Client

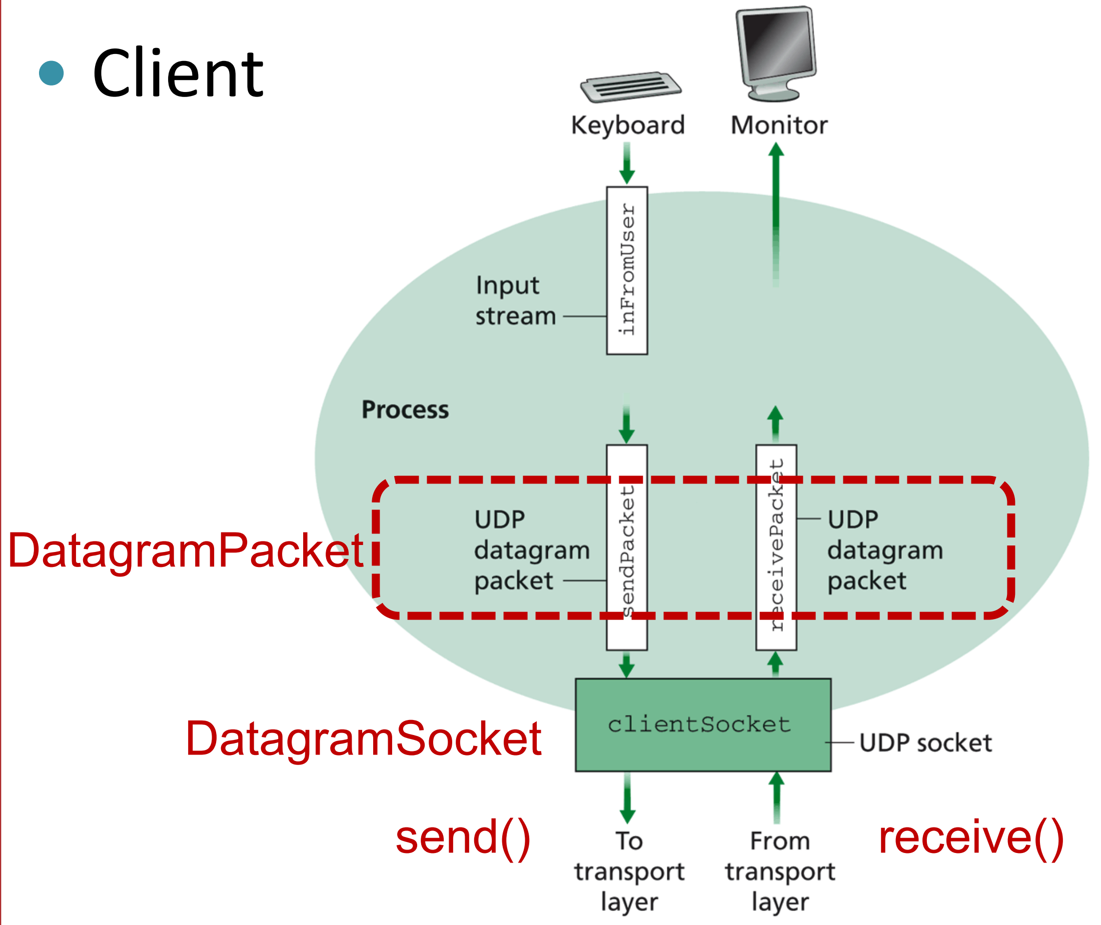


Figure 2.31 ♦ `UDPClient` has one stream; the socket accepts packets from the process and delivers packets to the process.

- Server

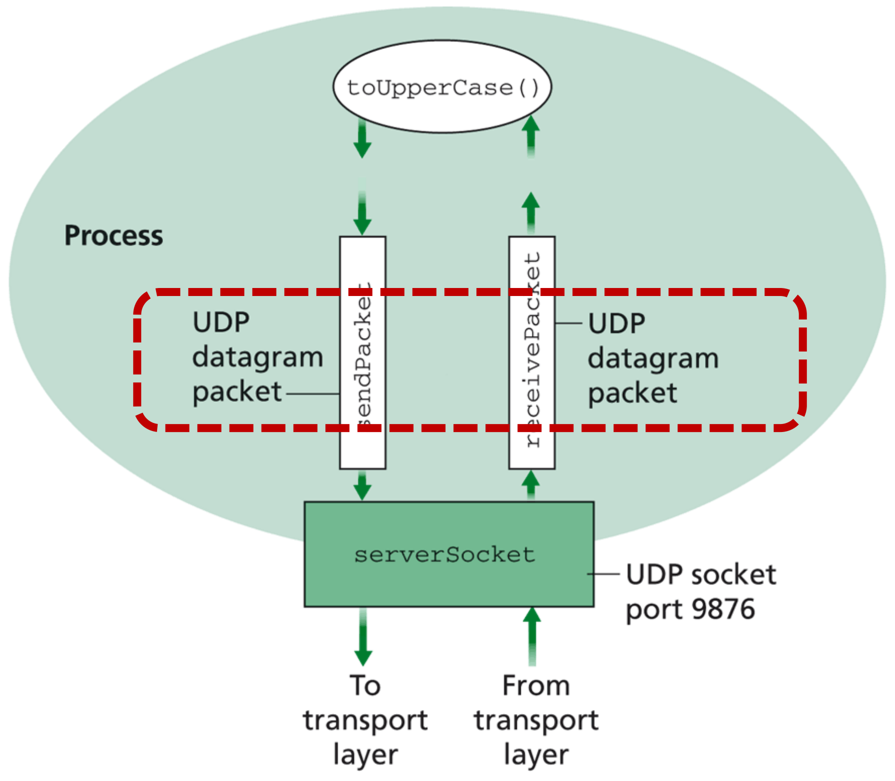


Figure 2.32 ♦ `UDPServer` has no streams; the socket accepts packets from the process and delivers packets to the process.

Example: UDP Client (1)

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

This package defines classes related to sockets

```
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 from user

create client socket

translate hostname to IP address using DNS

server name, e.g., id415m12.cs.unb.ca

Example: UDP Client (2)

create datagram with data-to-send, length, dst IP addr, dst port

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

server port #

send datagram to server

```
clientSocket.send(sendPacket);

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

read datagram from server

```
clientSocket.receive(receivePacket);
```

blocking method

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

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

close socket
(clean up behind yourself!)

```
clientSocket.close();
}
}
```

Example: UDP Server (1)

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

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

      DatagramSocket serverSocket = new DatagramSocket(9876);

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

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

create datagram socket at port 9876 → `DatagramSocket serverSocket = new DatagramSocket(9876);`

create space for received datagram → `DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);`

receive datagram → `serverSocket.receive(receivePacket);`

Example: UDP Server (2)

get IP addr
port #, of sender

```
InetAddress IPAddress = receivePacket.getAddress();
int port = receivePacket.getPort();

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

String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();
```

create datagram
to send to client

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

write out datagram
to socket

```
serverSocket.send(sendPacket);

      }
    }
  }
```

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

Socket Programming with TCP

- Client/Server

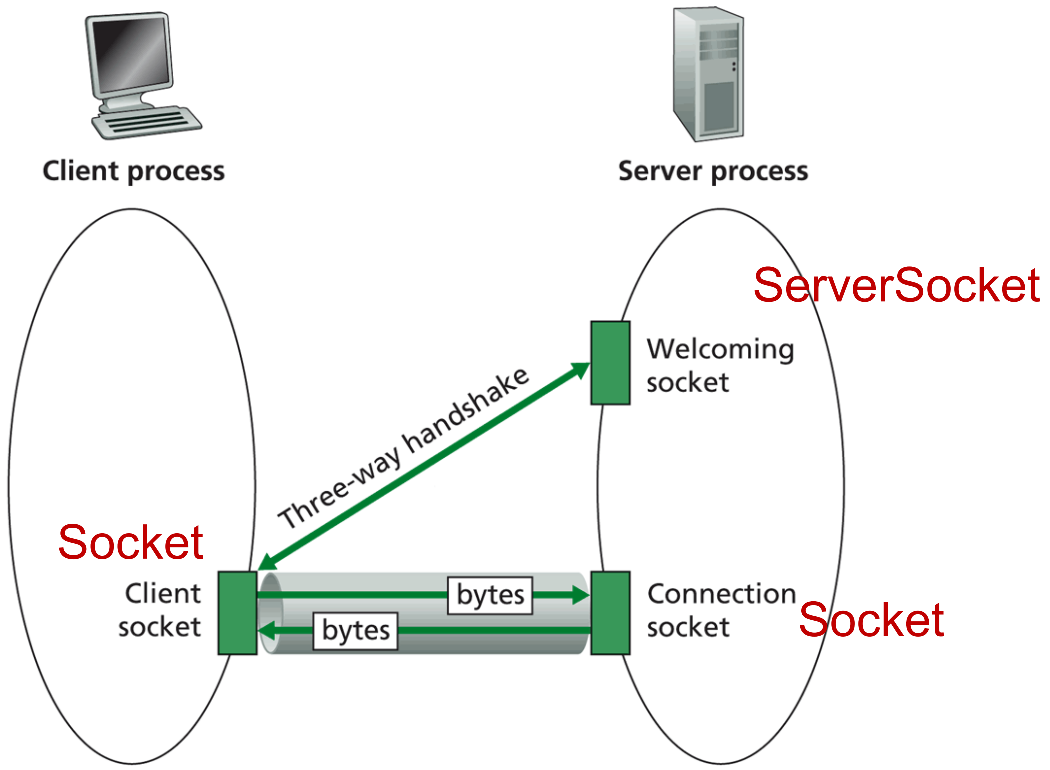


Figure 2.27 ♦ Client socket, welcoming socket, and connection socket

- Client

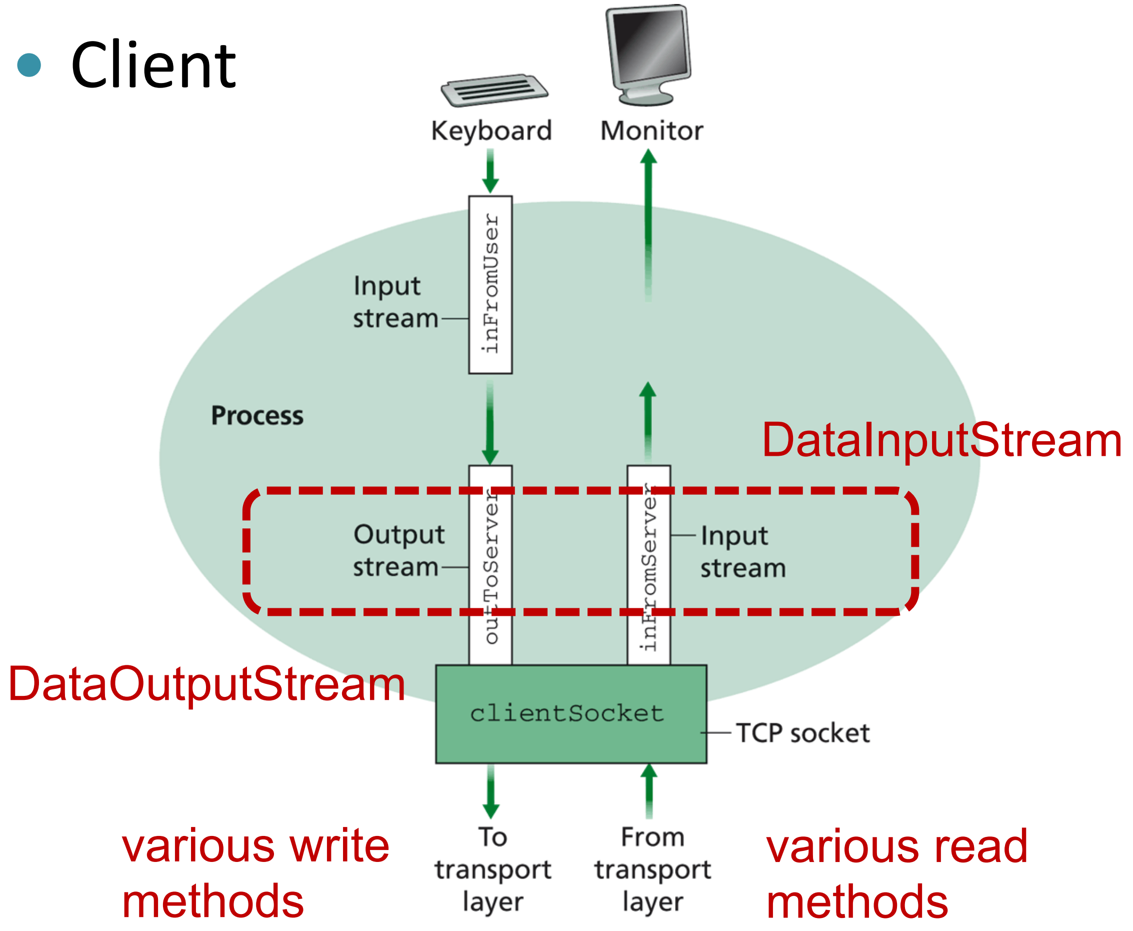


Figure 2.29 ♦ TCPClient has three streams through which characters flow.

Socket Programming with TCP

Client and server processes:

- Server process must be running first, have created socket (door) that welcomes client's contact
- To contact server, client create client-local TCP socket by specifying IP address, port number of server process
- When contacted by client, server TCP creates new socket for server process to communicate with client

Example: TCP Client (1)

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

create clientSocket object of type Socket, connect to server → Socket clientSocket = new Socket("hostname", 6789);

server name, e.g., id415m12.cs.unb.ca → "hostname"

server port # → 6789

create output stream attached to socket → DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());

Example: TCP Client (2)

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);
```

close socket
(clean up behind yourself!) →

```
clientSocket.close();

    }
}
```

Example: TCP Server (1)

```
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 accept() for client contact create new socket on return →

create input stream, attached to socket →

Example: TCP Server (2)

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