Lecture 3 : Introduction to Socket Programming

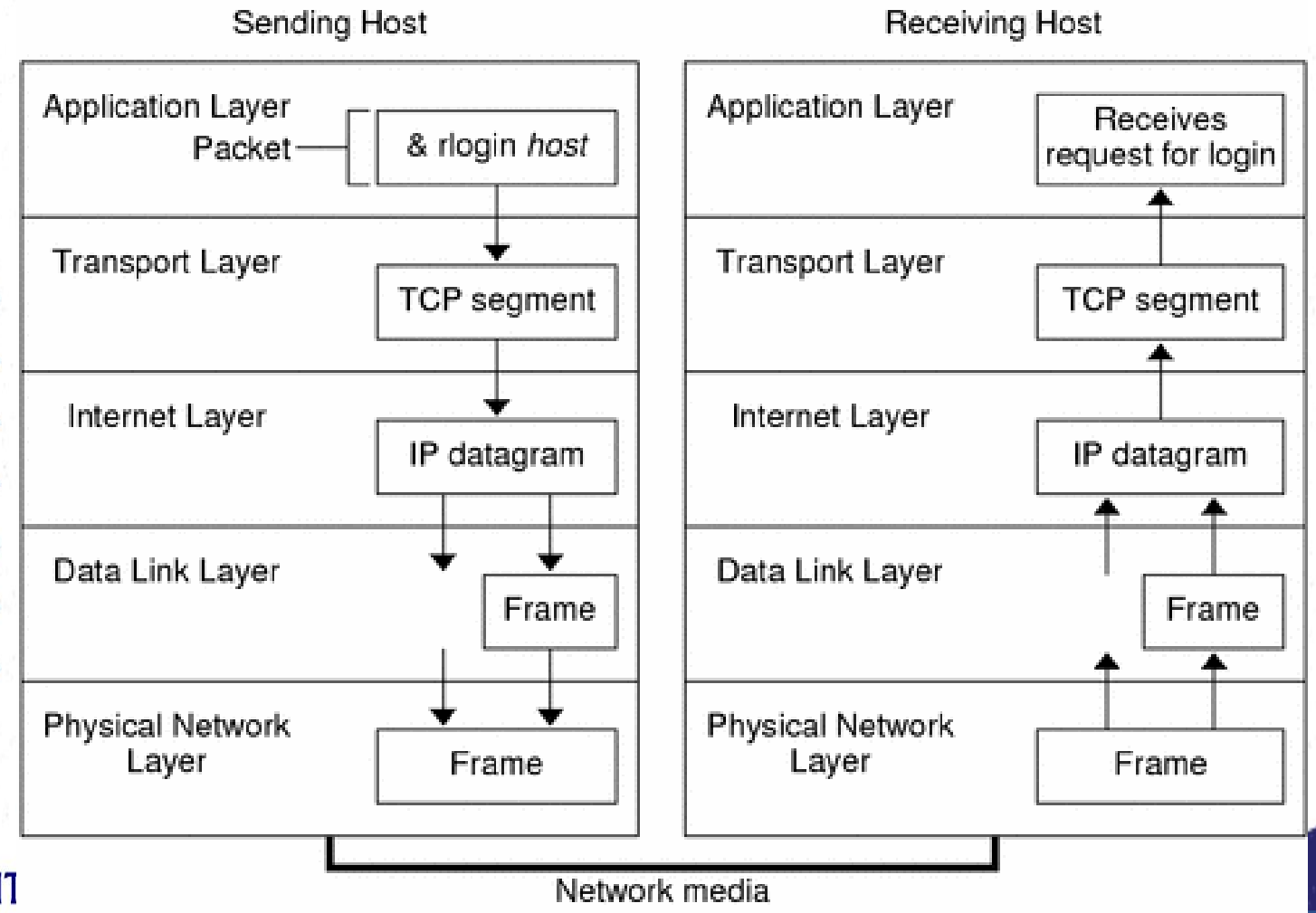
What is a socket?

- Port vs. Socket

- An interface between application and network
  - The application creates a socket
  - The socket *type* dictates the style of communication
    - reliable vs. best effort
    - connection-oriented vs. connectionless

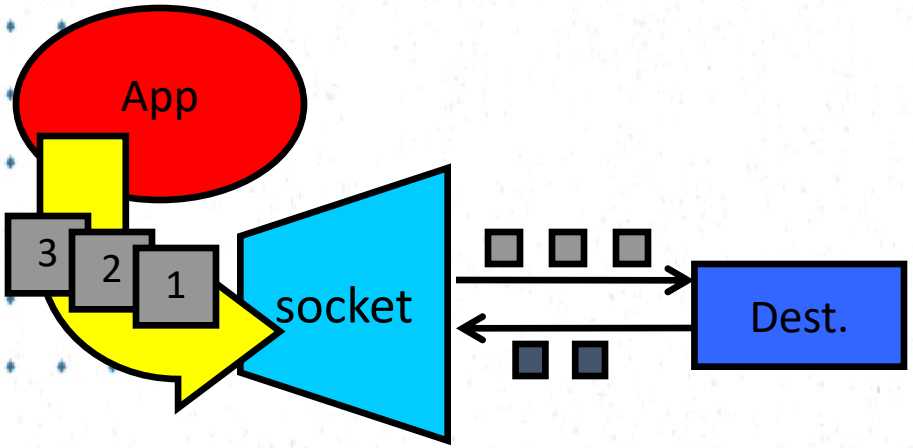
- Once configured the application can
  - pass data to the socket for network transmission
  - receive data from the socket (transmitted through the network by some other host)

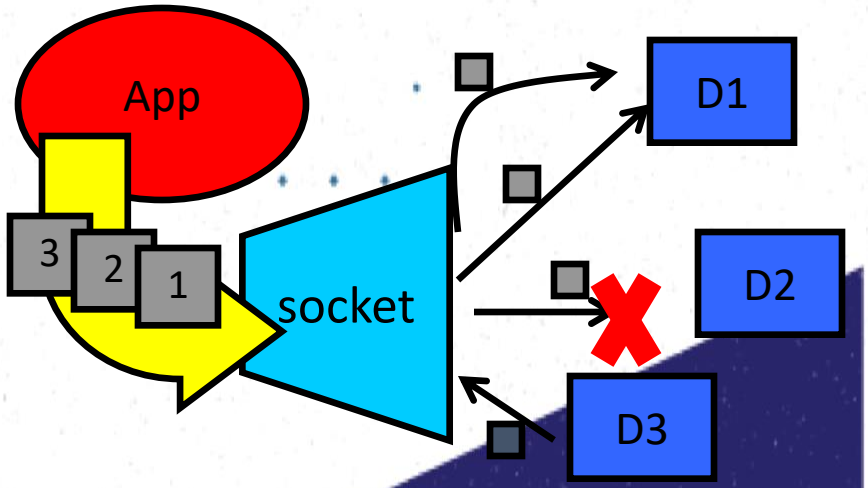
TCP/IP Stack


Sending Host
Receiving Host
Application Layer
Packet
& rlogin *host*
Application Layer
Receives request for login
Transport Layer
TCP segment
Transport Layer
TCP segment
Internet Layer
IP datagram
Internet Layer
IP datagram
Data Link Layer
Frame
Data Link Layer
Frame
Physical Network Layer
Frame
Physical Network Layer
Frame
Network media

Two essential types of sockets

- TCP Socket
  - reliable delivery
  - in-order guaranteed
  - connection-oriented
  - bidirectional
- UDP Socket
  - unreliable delivery
  - no order guarantees
  - no notion of “connection” – app indicates dest. for each packet
  - can send or receive



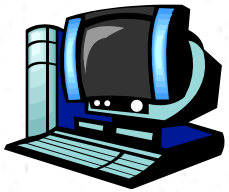


Applications

- TCP (Transmission control protocol)
  - Point to point chat applications, File transfer (FTP), Email (SMTP)
  - Used when there's a requirement for guaranteed delivery

- UDP (User datagram protocol)
  - Streaming, Multicast/Broadcast
  - Useful when the speed of more important than the assurance of delivery

A Socket-eye view of the Internet



medellin.cs.columbia.edu

(128.59.21.14)

newworld.cs.umass.edu

(128.119.245.93)

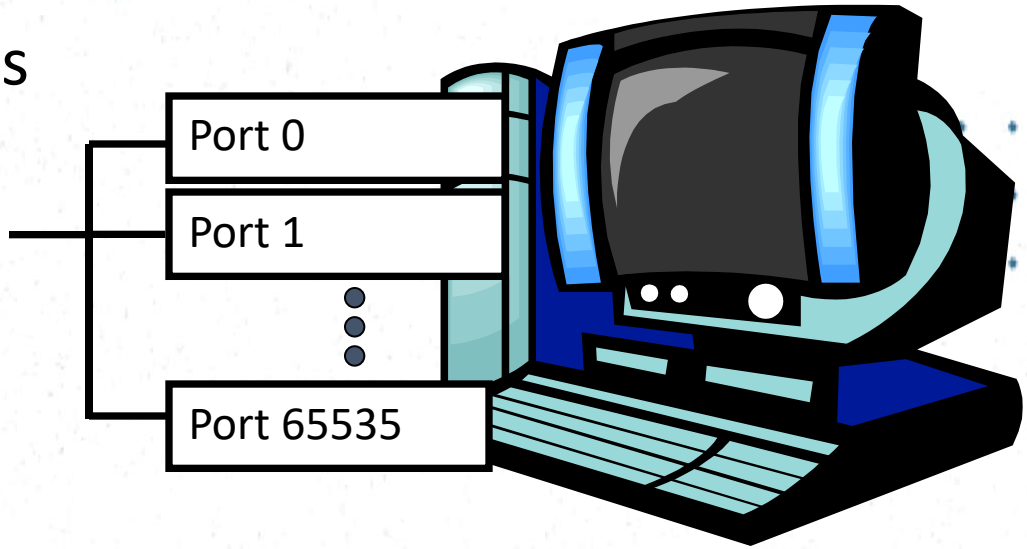
cluster.cs.columbia.edu

(128.59.21.14, 128.59.16.7, 128.59.16.5, 128.59.16.4)

- Each host machine has an IP address
- When a packet arrives at a host

Ports

- Each host has 65,536 ports
- Some ports are *reserved for specific apps*
  - 20,21: FTP
  - 23: Telnet
  - 80: HTTP
  - see RFC 1700 (about 2000 ports are reserved)

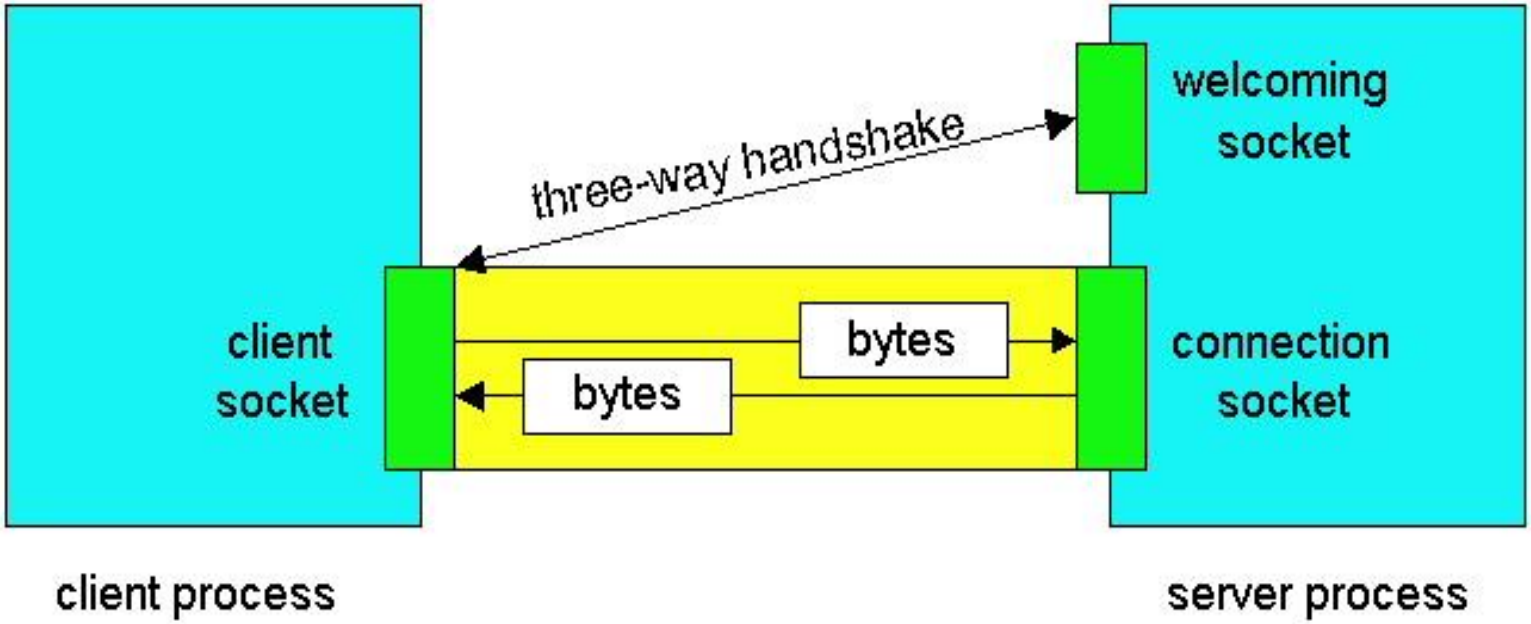


A socket provides an interface to send data to/from the network through a port

Addresses, Ports and Sockets

- In TCP, only one application (process) can listen to a port
- In UDP Multiple applications (processes) may listen to incoming messages on a single port
- Like apartments and mailboxes
  - You are the application
  - Your apartment building address is the address
  - Your mailbox is the port
  - The post-office is the network
  - Each family (process) of the apartment complex (computer) communicates with some same mailbox (port)

TCP Sockets

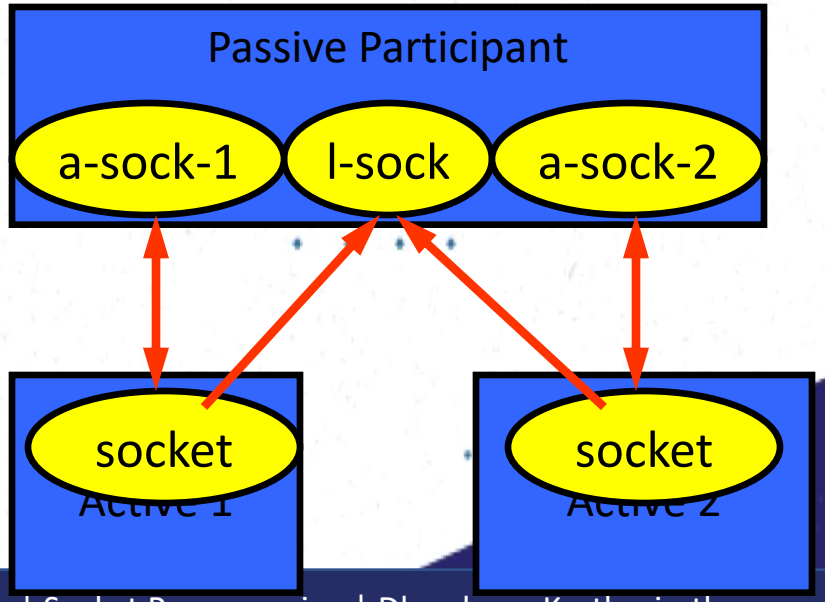


Client socket, welcoming socket (passive) and connection socket (active)

Connection setup

- Passive participant
  - step 1: listen (for incoming requests)
  - step 3: accept (a request)
  - step 4: data transfer
- The accepted connection is on a new socket
- The old socket continues to listen for other active participants

- Active participant
  - step 2: request & establish connection
  - step 4: data transfer



Dealing with blocking

- Calls to sockets can be blocking (no other client may be able to connect to the server)
- Can be resolved using multi-threaded programming
- Start a new thread for every incoming connection

Java Sockets Programming

- The package java.net provides support for sockets programming (and more).
- Typically you import everything defined in this package with:

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

Classes

InetAddress

Socket

ServerSocket

DatagramSocket

DatagramPacket

InetAddress class

- Static methods you can use to create new InetAddress objects.
  - getByName(String host)
  - getAllByName(String host)
  - getLocalHost()

```
InetAddress x = InetAddress.getByName(
                                "cse.unr.edu");
```

❖Throws **`UnknownHostException`**

```
try {

  InetAddress a = InetAddress.getByName(hostname);

  System.out.println(hostname + ":" +
      a.getHostAddress());

} catch (UnknownHostException e) {

  System.out.println("No address found for " +
        hostname);

}
```

Socket class

- Corresponds to active TCP sockets only!
  - client sockets
  - socket returned by accept();
- Passive sockets are supported by a different class:
  - ServerSocket
- UDP sockets are supported by
  - DatagramSocket

JAVA TCP Sockets

- java.net.Socket
  - Implements client sockets (also called just "sockets").
  - An endpoint for communication between two machines.
  - Uses input/output streams to pass messages
    -

- java.net.ServerSocket
  - Implements server sockets.
  - Waits for requests to come in over the network.
  - Accepts the client connection requests
  - Performs some operation based on each request

Socket Constructors

- Constructor creates a TCP connection to a named TCP server.
  - There are a number of constructors:

```
Socket(InetAddress server, int port);

Socket(InetAddress server, int port,
       InetAddress local, int localport);

Socket(String hostname, int port);
```

Socket Methods

```
void close();
InetAddress getInetAddress();
InetAddress getLocalAddress();
InputStream getInputStream();
OutputStream getOutputStream();
```

- Lots more (setting/getting socket options, partial close, etc.)

Socket I/O

- Socket I/O is based on the Java I/O support
  - in the package **`java.io`**
- InputStream and OutputStream are abstract classes
  - common operations defined for all kinds of InputStreams, OutputStreams…

InputStream Basics

```
// reads some number of bytes and
// puts in buffer array b
int read(byte[] b);

// reads up to len bytes
int read(byte[] b, int off, int len);
```

Both methods can throw **`IOException`**.

Both return –1 on EOF.

OutputStream Basics

```
// writes b.length bytes
void write(byte[] b);

// writes len bytes starting
// at offset off
void write(byte[] b, int off, int len);
```

Both methods can throw **`IOException`**.

ServerSocket Class (TCP Passive Socket)

- Constructors:

```
ServerSocket(int port);

ServerSocket(int port, int backlog);

ServerSocket(int port, int backlog,
             InetAddress bindAddr);
```

ServerSocket Methods

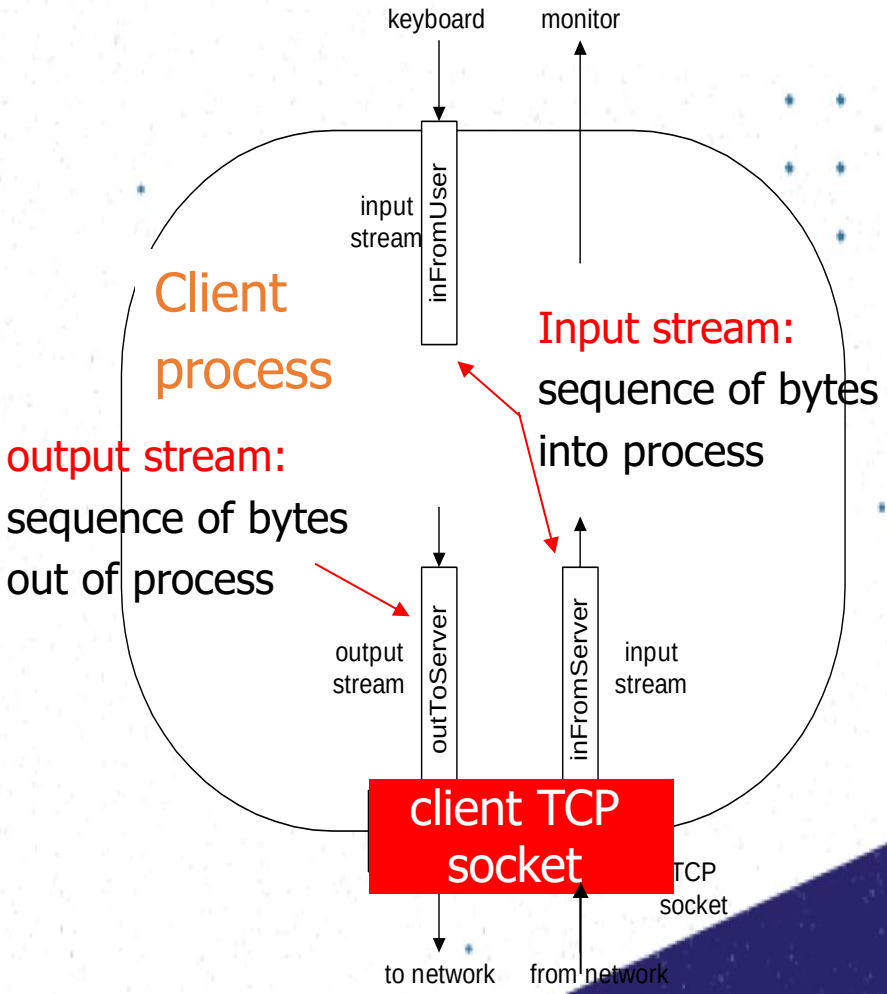
```
Socket accept();

void close();

InetAddress getInetAddress();

int getLocalPort();
```

throw `IOException, SecurityException`

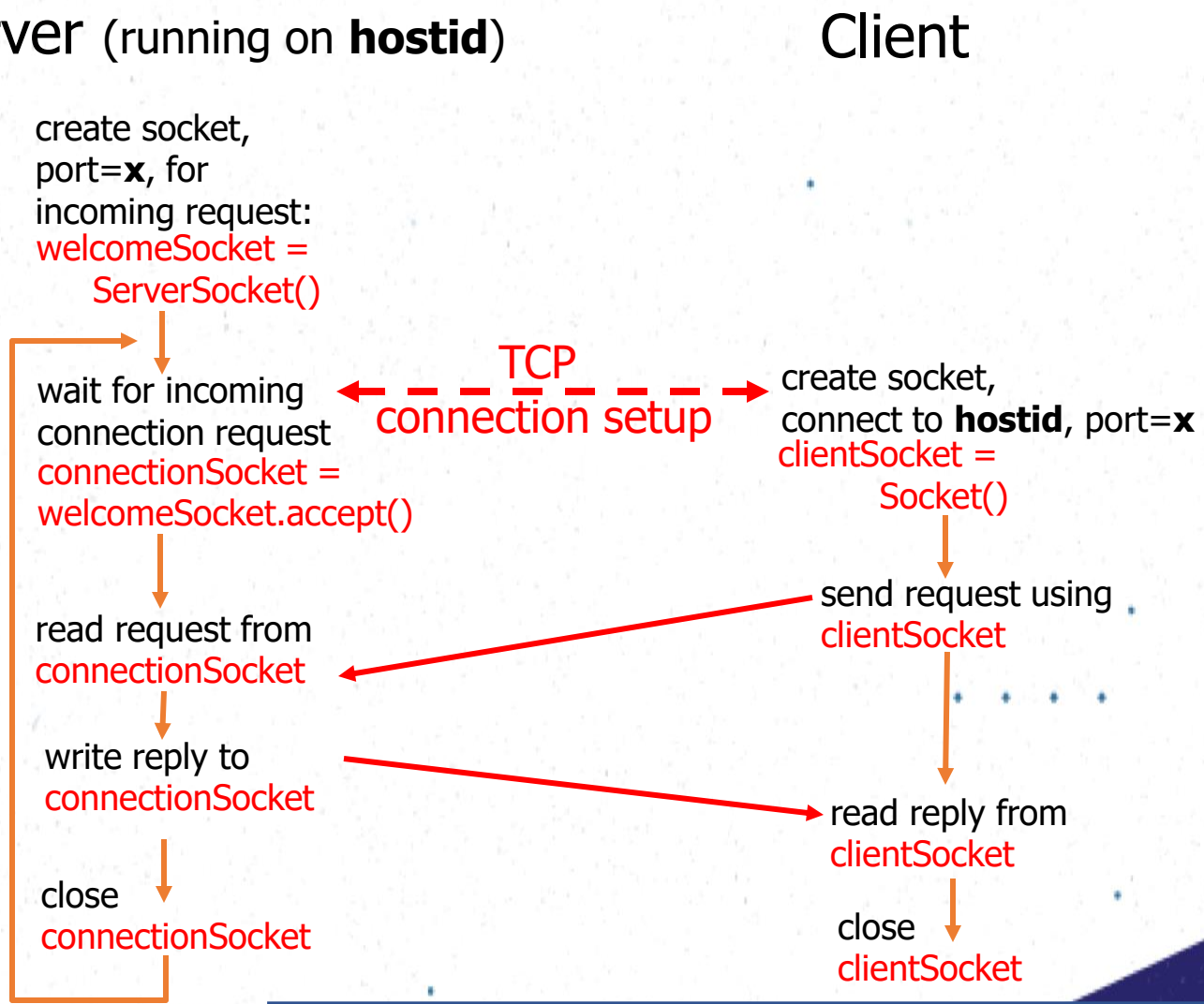
Socket programming with TCP

Example client-server app:

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



Client/server socket interaction: TCP



TCPClient.java

```
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());
```

TCPClient.java

```
        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();

  }
}
```

TCPServer.java

```
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()));
```

TCPServer.java

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

    clientSentence = inFromClient.readLine();

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

    outToClient.writeBytes(capitalizedSentence);

        }
    }
}
```

UDP Sockets

- DatagramSocket class

- DatagramPacket class needed to specify the payload
  - incoming or outgoing

Socket Programming with UDP

- UDP
  - Connectionless and unreliable service.
  - There isn't an initial handshaking phase.
  - Doesn't have a pipe.
  - Transmitted data may be received out of order, or lost

- Socket Programming with UDP
  - No need for a welcoming socket.
  - No streams are attached to the sockets.
  - the sending hosts creates "packets" by attaching the IP destination address and port number to each batch of bytes.
  - The receiving process must unravel to received packet to obtain the packet's information bytes.

JAVA UDP Sockets

- In Package java.net
  - java.net.DatagramSocket
    - A socket for sending and receiving datagram packets.
    - Constructor and Methods
      - DatagramSocket(int port): Constructs a datagram socket and binds it to the specified port on the local host machine.
      - void receive(DatagramPacket p)
      - void send(DatagramPacket p)
      - void close()

DatagramSocket Constructors

```
DatagramSocket();

DatagramSocket(int port);

DatagramSocket(int port, InetAddress a);
```

All can throw SocketException or SecurityException

Datagram Methods

```
void connect(InetAddress, int port);

void close();

void receive(DatagramPacket p);

void send(DatagramPacket p);
```

Lots more!

Datagram Packet

- Contain the payload
  - a byte array
- Can also be used to specify the destination address
  - when not using connected mode UDP

DatagramPacket Constructors

For receiving:

```
DatagramPacket( byte[] buf, int len);
```

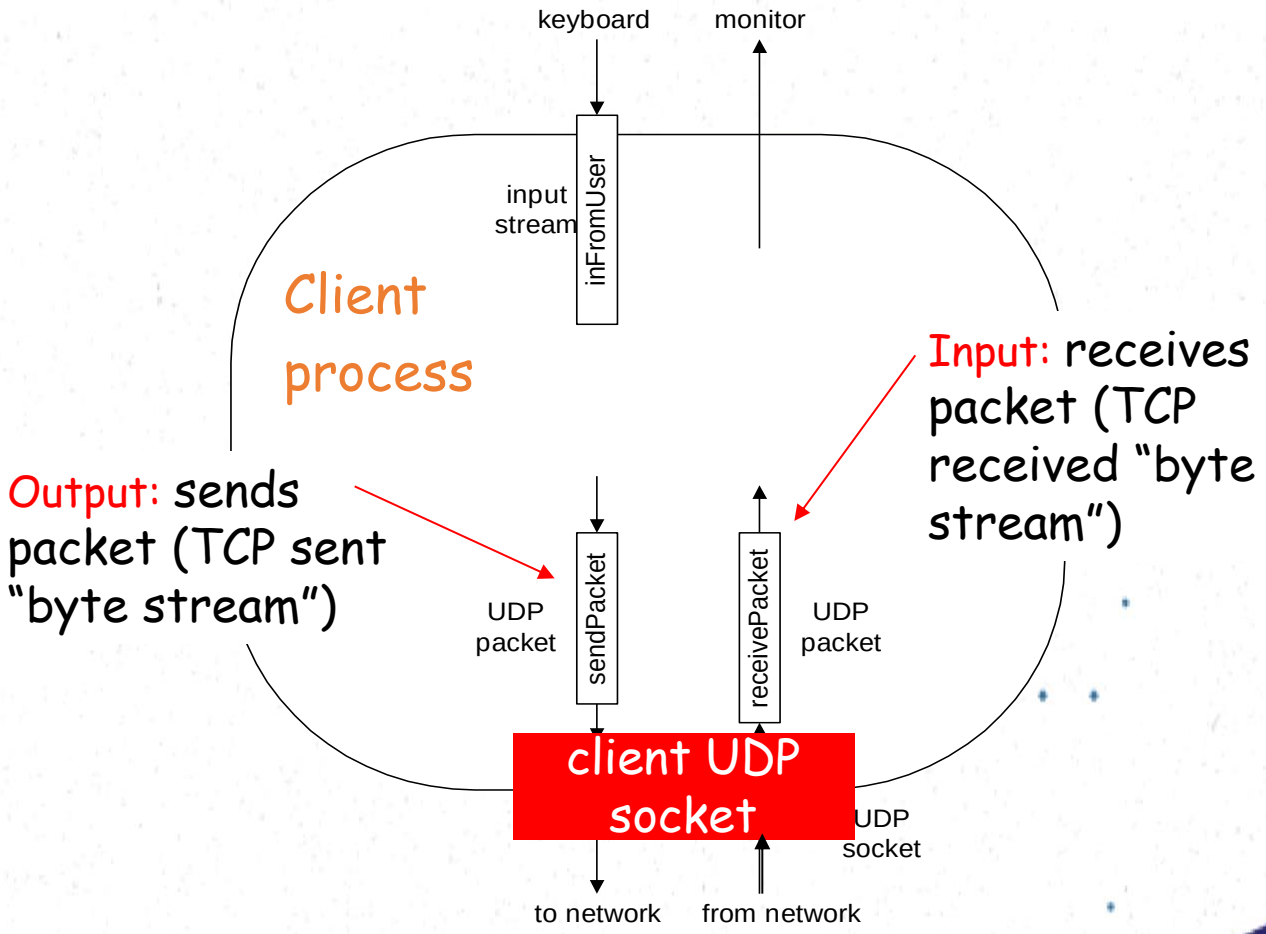
For sending:

```
DatagramPacket( byte[] buf, int len
                InetAddress a, int port);
```

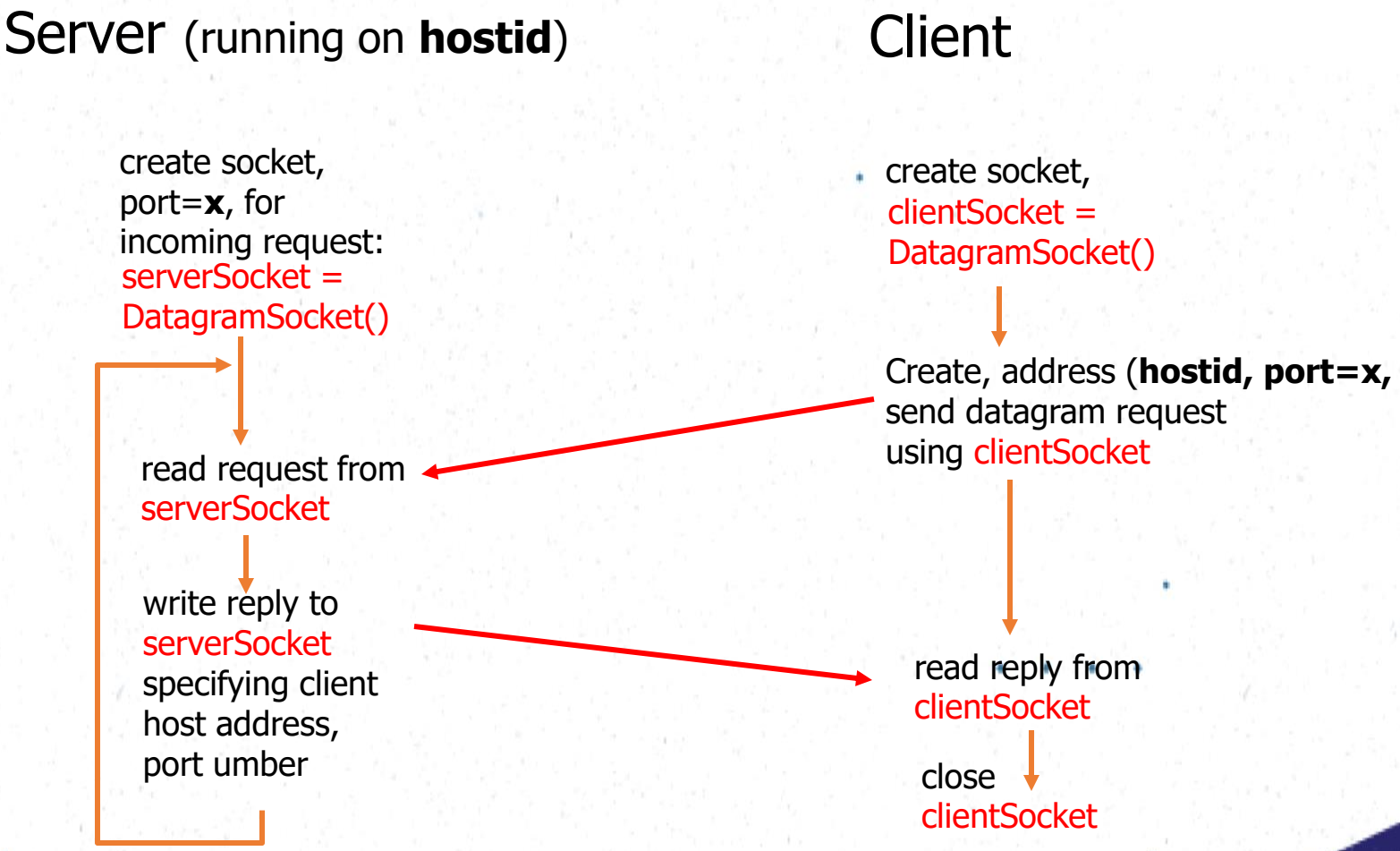
DatagramPacket methods

```
byte[] getData();
void setData(byte[] buf);

void setAddress(InetAddress a);
void setPort(int port);

InetAddress getAddress();
int getPort();
```

Example: Java client (UDP)



Client/server socket interaction: UDP



UDPClient.java

```
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();
```

UDPClient.java

```
    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();

    }
}
```

UDPServer.java

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

UDPServer.java

```
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);
    }
  }
}
```

Summary

- Socket programming is the most fundamental form of Client-Server distributed computing available for app. developers
- Can be used to develop client-server distributed applications (e.g. Messaging applications)
- However, most real-world distributed systems use more high level distributed computing technologies (E.g. Web services, EJBs)
- Yet the underlying communication mechanism of these high level Dist. Computing frameworks is socket communication