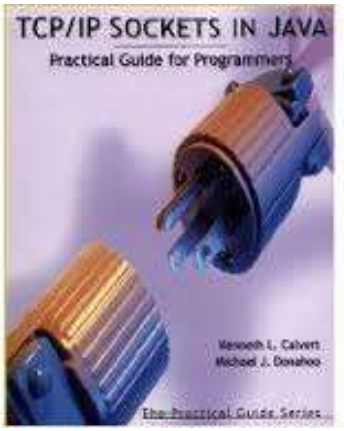
Active Learning

Socket Programming #1



Most slides are from web site of the textbook “Computer Networking: A Top Down Approach,” written by Jim Kurose and Keith Ross and “**TCP/IP Sockets in Java: Practical Guide for Programmers**”, written by Kenneth L. Calvert and Michael J. Donahoo

TRY IT OUT:
TCP Socket programming example

Example simple client-server app:

- **TRY IT OUT !!**
  - **1. New Project**
  - **2. Add the following two Java classes;**
    - **TCPServer1**
    - **TCPClient1**
  - **3. Run !**

Example: Java server (TCP)

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

class TCPServer1 {

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

      ServerSocket welcomeSocket = new ServerSocket(6789);
      System.out.println("Server start..\n ");
      while(true) {

            Socket connectionSocket = welcomeSocket.accept();

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

Example: Java server (TCP), cont

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

        clientSentence = inFromClient.readLine();
        System.out.println("FROM CLIENT: " + clientSentence );

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

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

Example: Java client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient1 {

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

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

        Socket clientSocket = new Socket("127.0.0.1", 6789);

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

Example: Java client (TCP), cont.

```
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();

    }
}
```

Run !

Outline

- Overview
- What is a socket?
- Using sockets
  - Types (Protocols)
  - Associated functions
  - Styles

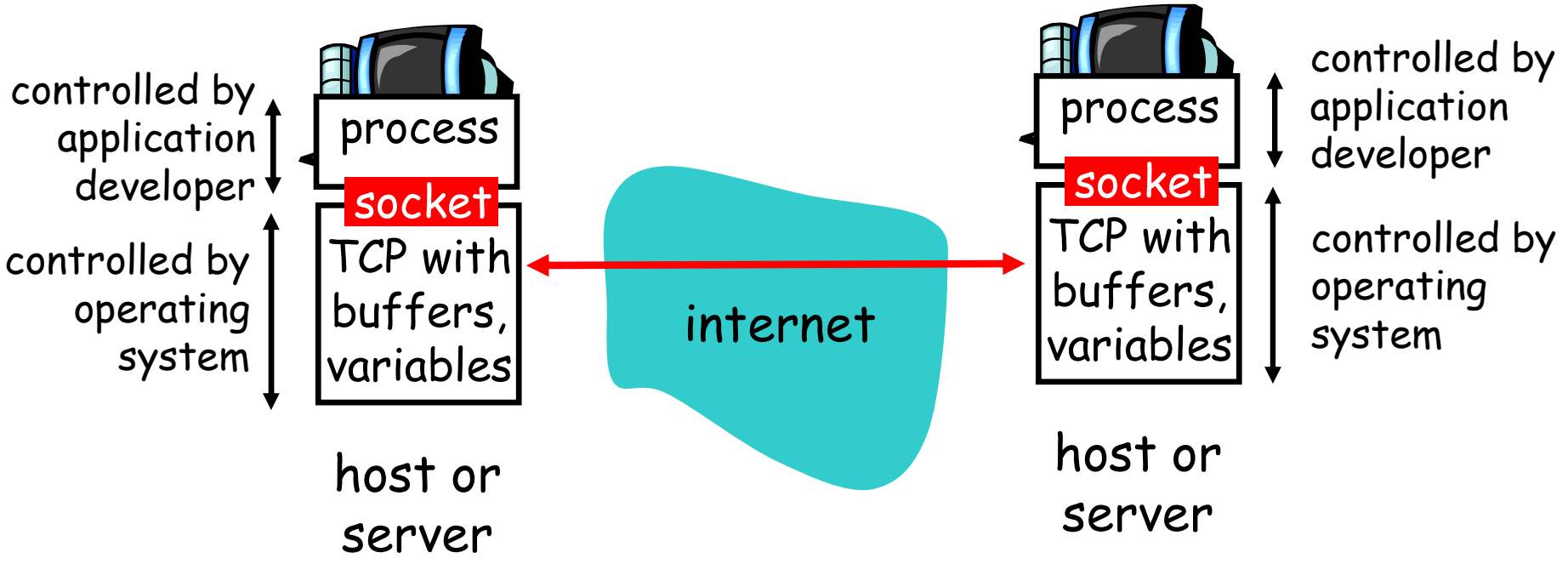
  - We will look at using sockets in JAVA
    - Note: C/C++ sockets are conceptually quite similar

What is a socket?

Socket: a door between application process and end-end-transport protocol (UDP or TCP)

socket

a *host-local*, *application-created*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another application process

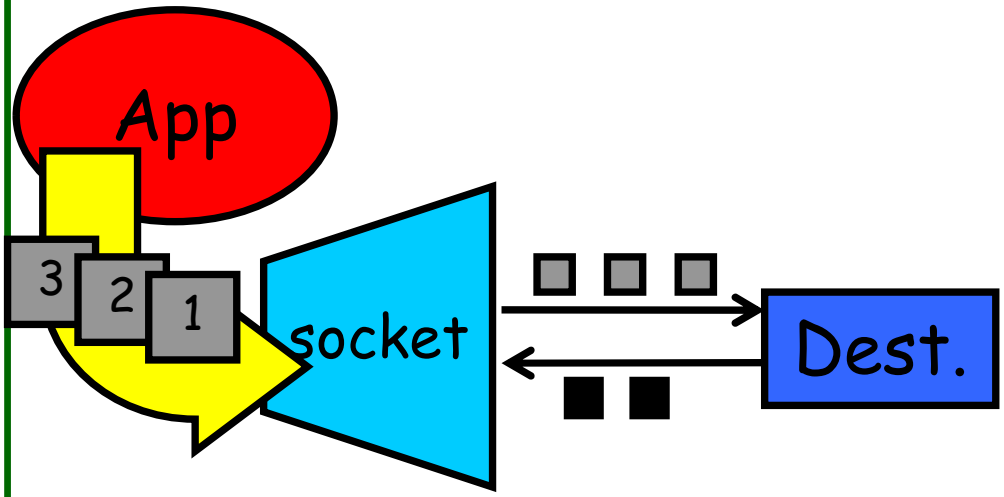


What is a socket? (cont.)

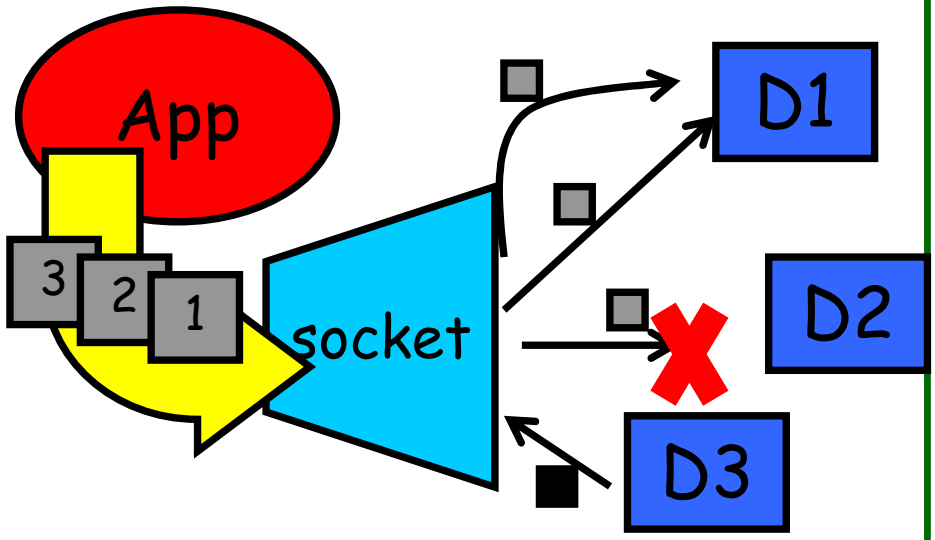
- An **interface** between application and network
  - The application creates a socket
  - Once configured, the application can
    - pass data to the socket for network transmission
    - receive data from the socket (transmitted through the network by some other host)

Two essential types of sockets

- **Connection-oriented**
  - a.k.a. **TCP**
  - reliable delivery
  - in-order guaranteed
  - connection-oriented
  - bidirectional



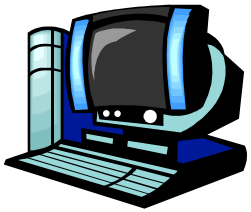
- **Connectionless**
  - a.k.a. **UDP**
  - not guaranteed delivery
  - no order guarantees
  - no notion of “connection” – app indicates dest. for each packet
  - can send or receive



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)

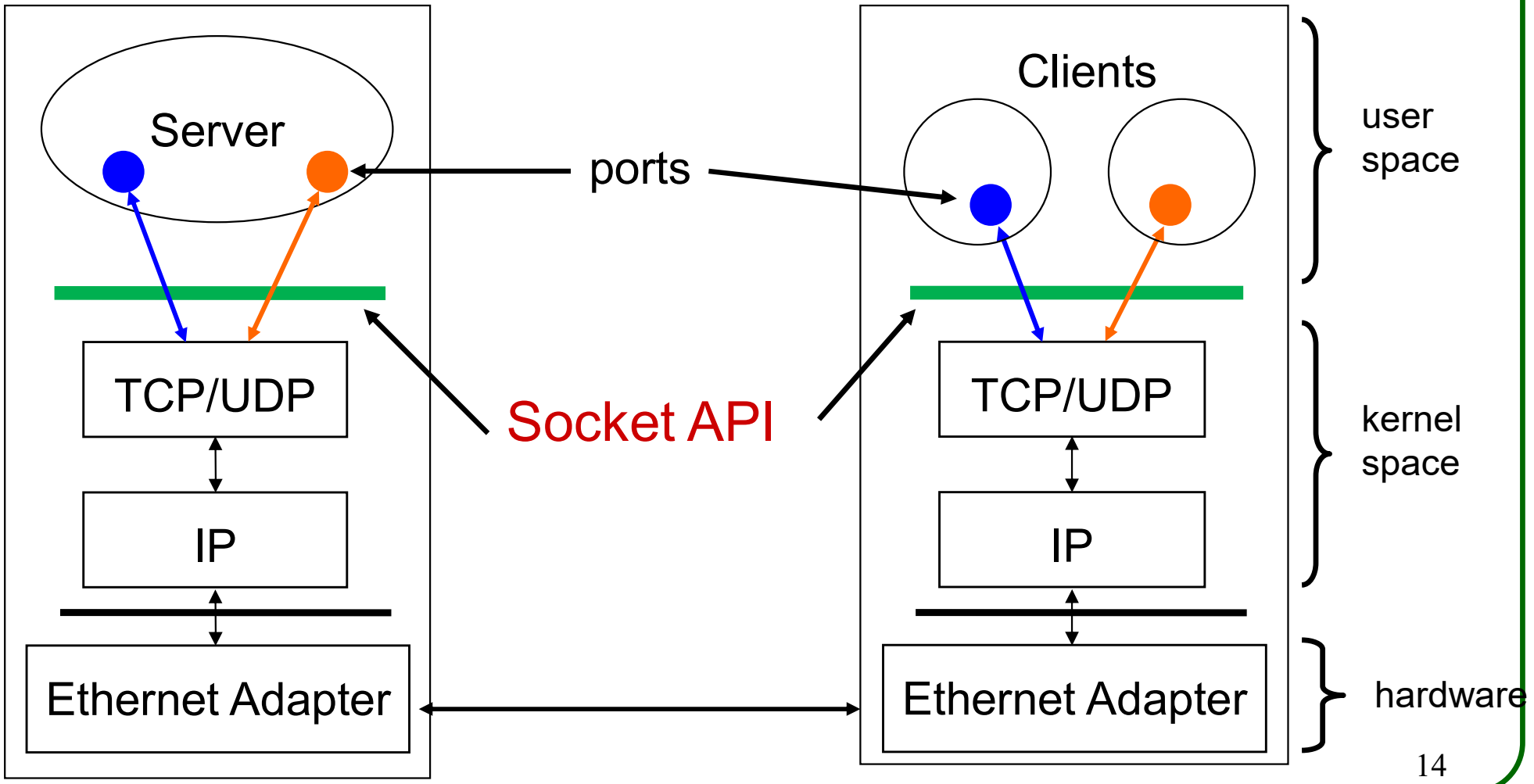
- Each host machine has an IP address
- When a packet arrives at a host, how to associate with a process

Example API functions

- JAVA
  - Socket class, ServerSocket class
  - socket()
  - accept()
  - getInputStream()
  - getOutputStream()
  - getLocalHost()
  - close()
  - …
- C
  - socket()
  - bind()
  - accept()
  - send()
  - recv()
  - select()
  - close()
  - …

Server and Client

- Server and Client exchange messages over the network through a common Socket API



TCP Client/Server Interaction

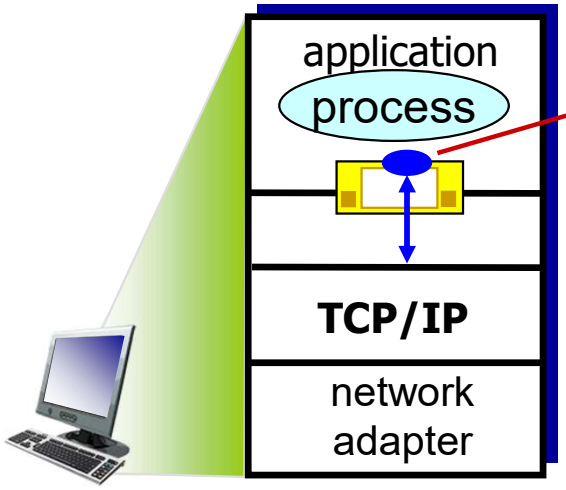
Server starts by getting ready to receive client connections…

Server

1. Create a TCP socket
2. Repeatedly:
   a. Listen (wait) & Accept new connection
   b. Communicate
   c. Close the connection

Client

1. Create a TCP socket & Connect server
2. Communicate
3. Close the connection



Server socket (identified by port#)

controlled by OS

Create welcoming socket at port, e.g. 6789

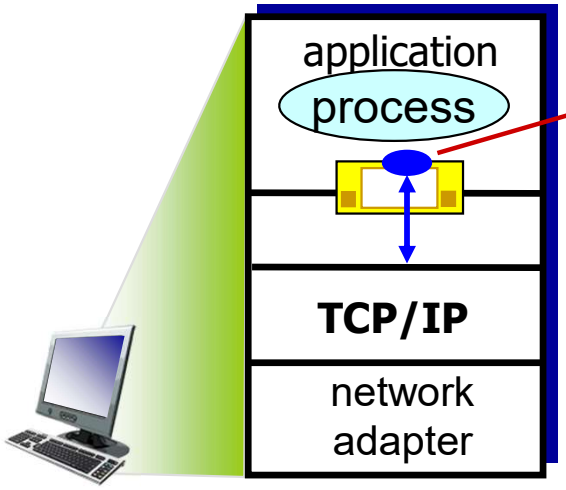
```
ServerSocket servSock = new ServerSocket(servPort);
```

Server

1. **Create a TCP socket**
2. Repeatedly:
   a. Listen (wait) & Accept new connection
   b. Communicate
   c. Close the connection

Client

1. Create a TCP socket & Connect server
2. Communicate
3. Close the connection



Server socket (identified by port#)

Wait, on welcoming socket for contact by client

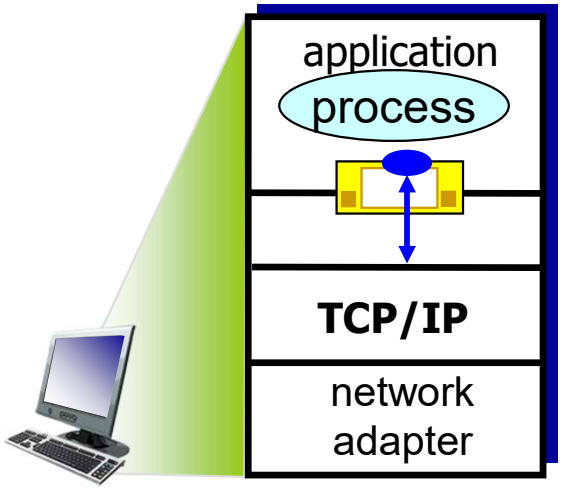
```
for (;;) {
    Socket clntSock =   servSock.accept();
```

Server

1. Create a TCP socket
2. Repeatedly:
   a. **Listen (wait) & Accept new connection**
   b. Communicate
   c. Close the connection

Client

1. Create a TCP socket & Connect server
2. Communicate
3. Close the connection



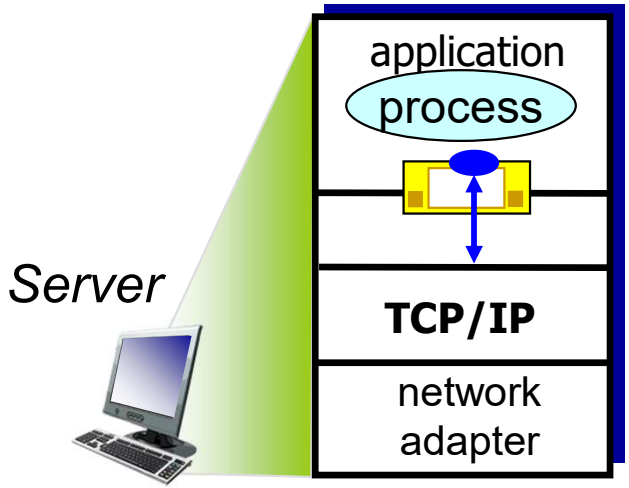
Server is now blocked waiting for connection from a client

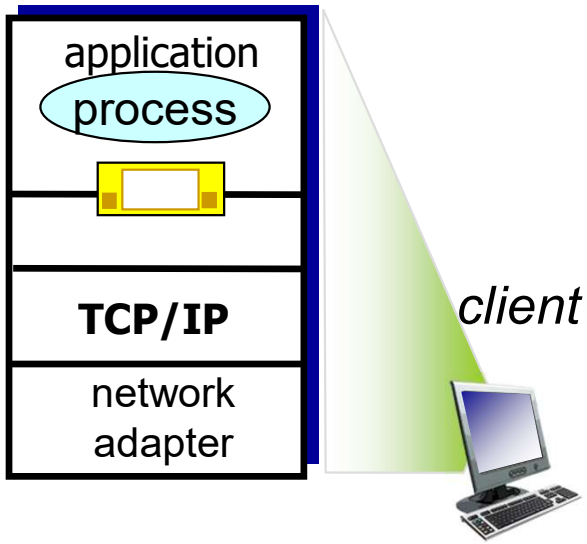
Server

1. Create a TCP socket
2. Repeatedly:
   a. **Listen (wait) & Accept new connection**
   b. Communicate
   c. Close the connection

Client

1. Create a TCP socket & Connect server
2. Communicate
3. Close the connection





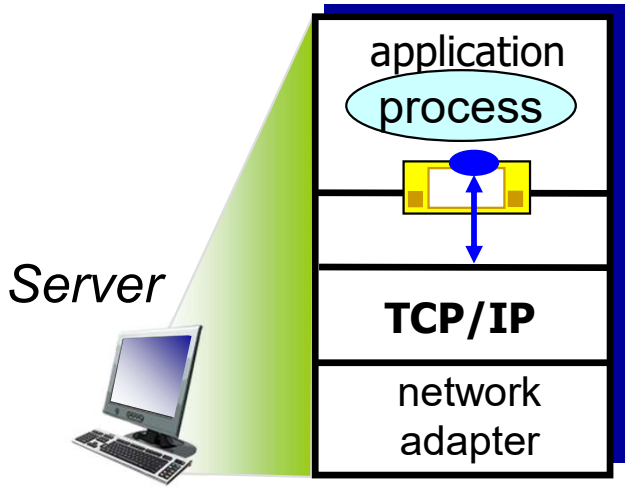
Later, a client decides to talk to the server…

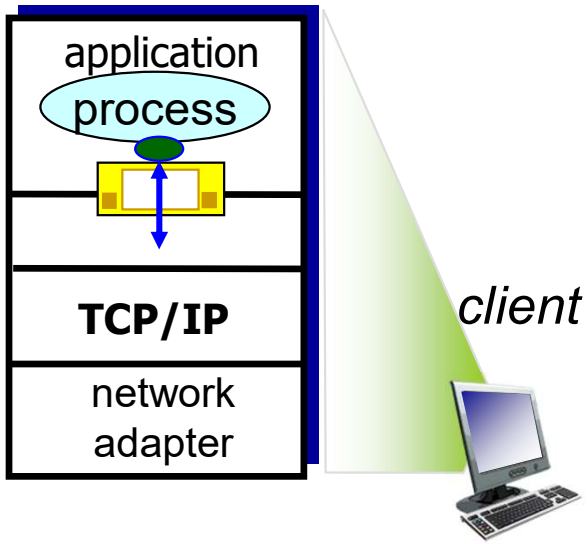
Server

1. Create a TCP socket
2. Repeatedly:
   a. **Listen (wait) & Accept new connection**
   b. Communicate
   c. Close the connection

Client

1. Create a TCP socket & Connect server
2. Communicate
3. Close the connection





Create
client socket,
connect to server

Socket socket = new Socket(serverIP, servPort);

Server

1. Create a TCP socket
2. Repeatedly:
   a. **Listen (wait) & Accept new connection**
   b. Communicate
   c. Close the connection

Client

1. Create a TCP socket & Connect server
2. Communicate
3. Close the connection

Socket clntSock = servSock.accept(); ← Accept! & server TCP creates new socket for the client

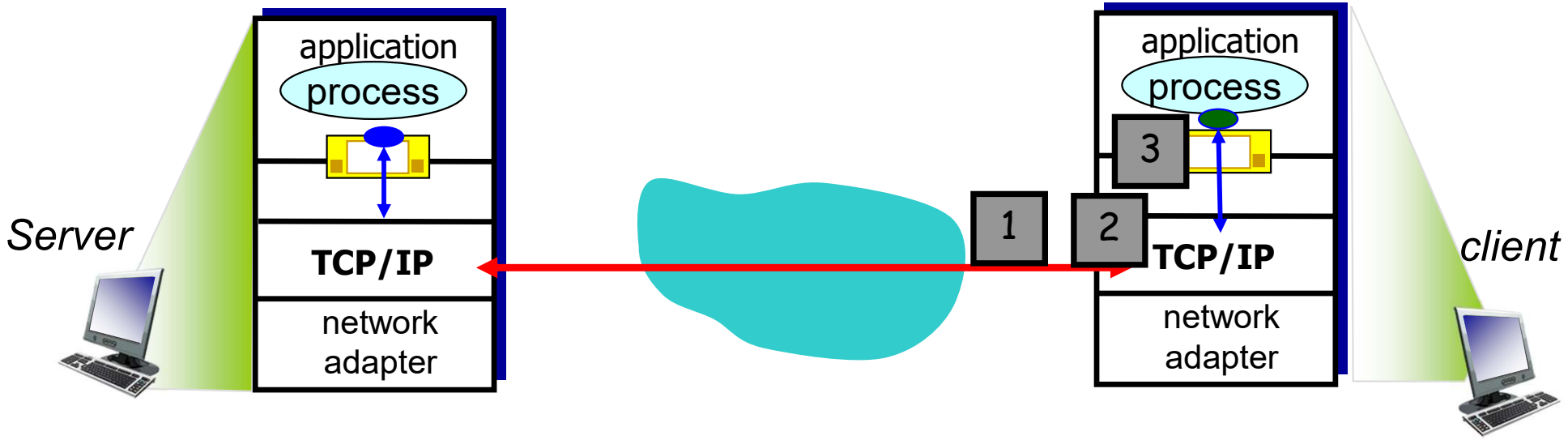
```
InputStream in = clntSock.getInputStream();
recvMsgSize = in.read(byteBuffer);
```

Server

1. Create a TCP socket
2. Repeatedly:
   a. Listen (wait) & Accept new connection
   b. Communicate
   c. Close the connection

Client

1. Create a TCP socket & Connect server
2. Communicate
3. Close the connection




```
InputStream in = clntSock.getInputStream();
recvMsgSize = in.read(byteBuffer);
```

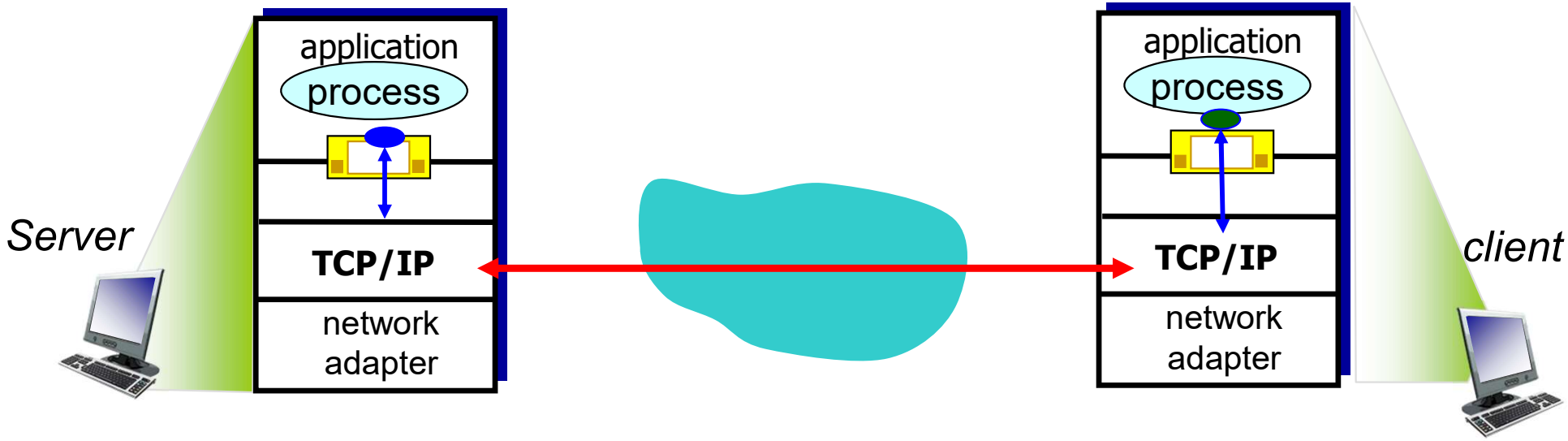
```
OutputStream out = socket.getOutputStream();
out.write(byteBuffer);
```

Server

1. Create a TCP socket
2. Repeatedly:
   a. Listen (wait) & Accept new connection
   b. Communicate
   c. Close the connection

Client

1. Create a TCP socket & Connect server
2. Communicate
3. Close the connection



close(clntSocket)

close(sock);

Server

1. Create a TCP socket
2. Repeatedly:
   a. Listen (wait) & Accept new connection
   b. Communicate
   c. Close the connection

Client

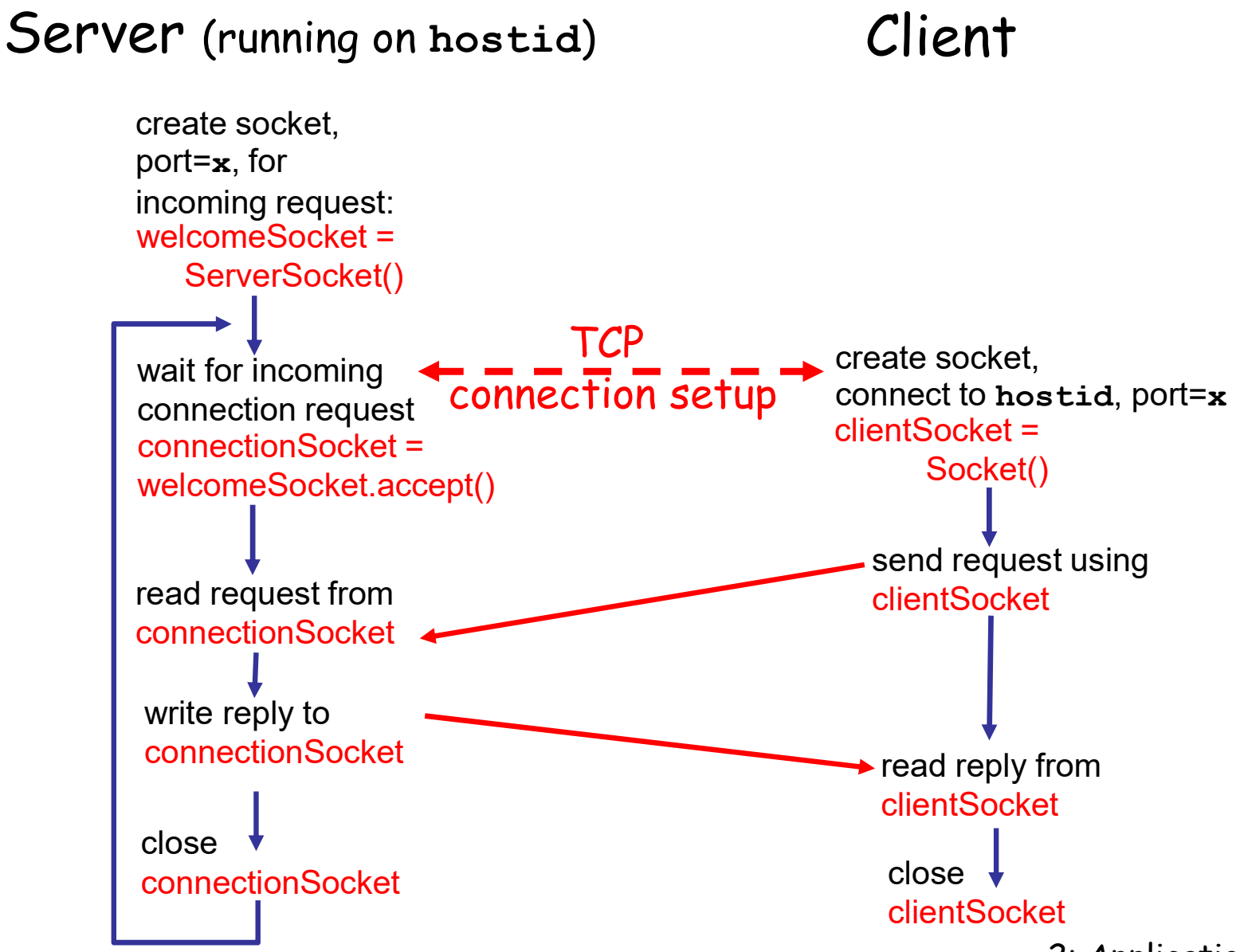
1. Create a TCP socket & Connect server
2. Communicate
3. **Close the connection**

Client/server socket interaction: Summary

- **Server**: program/computer **providing** a service
  - Creates a local socket
  - Binds local socket to a specific port (in Java, this step is included in the creation step)
  - Listens for incoming connections
  - Accepts a connection, assigning a new socket for the connection (in Java, listen & accept are done together)
  - Sends/receives data

- **Client**: program/computer **requesting** a service
  - **Client knows server address and port**
  - Creates a local socket
  - Connects to remote socket
  - Sends/receives data

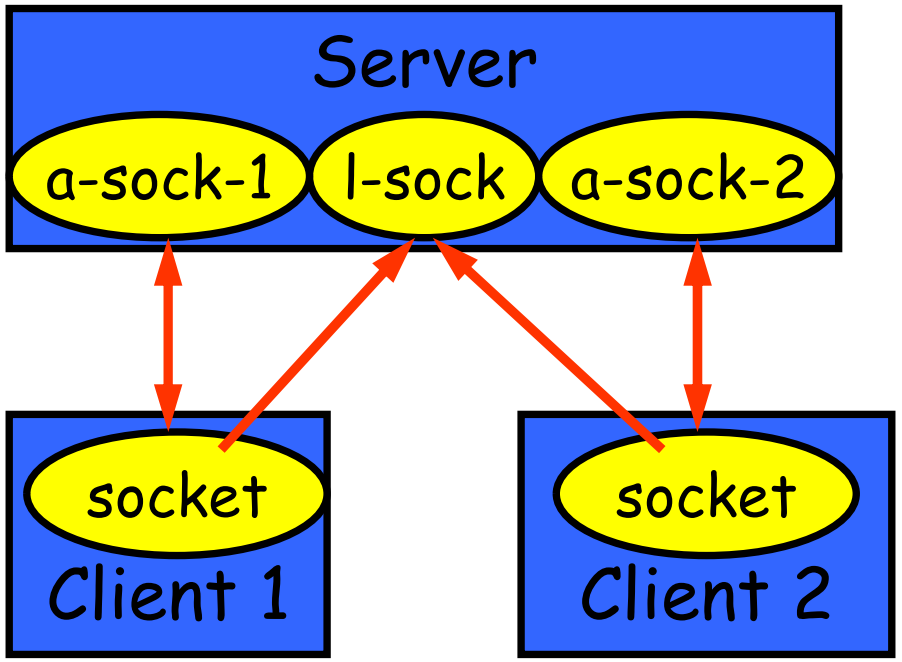
Client/server socket interaction: Summary



Connection setup @ server side

connectionSocket =
welcomeSocket.accept()

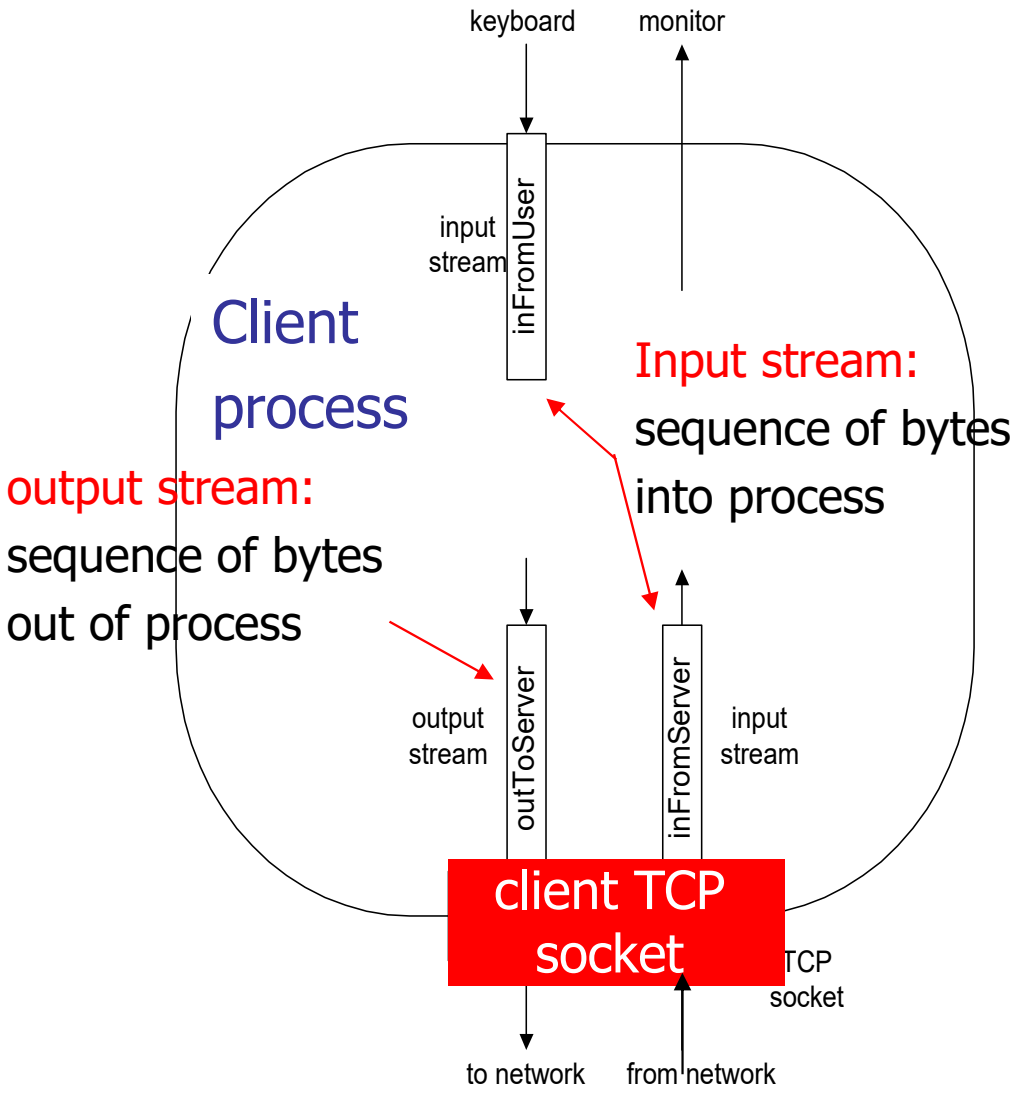
- When contacted by a client, server TCP creates a new socket for server process to communicate with the client
- The accepted connection is on a new socket
- The listen-socket continues to listen for other active participants
- Why?
  - allows server to talk with multiple clients !!



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)



TCPClient.java

USE: 127.0.0.1

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

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

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

        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
        DataOutputStream outToServer =
            new DataOutputStream(clientSocket.getOutputStream());
```

Create client socket, connect to server

Create input stream

Create output stream attached to socket

TCPClient.java

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

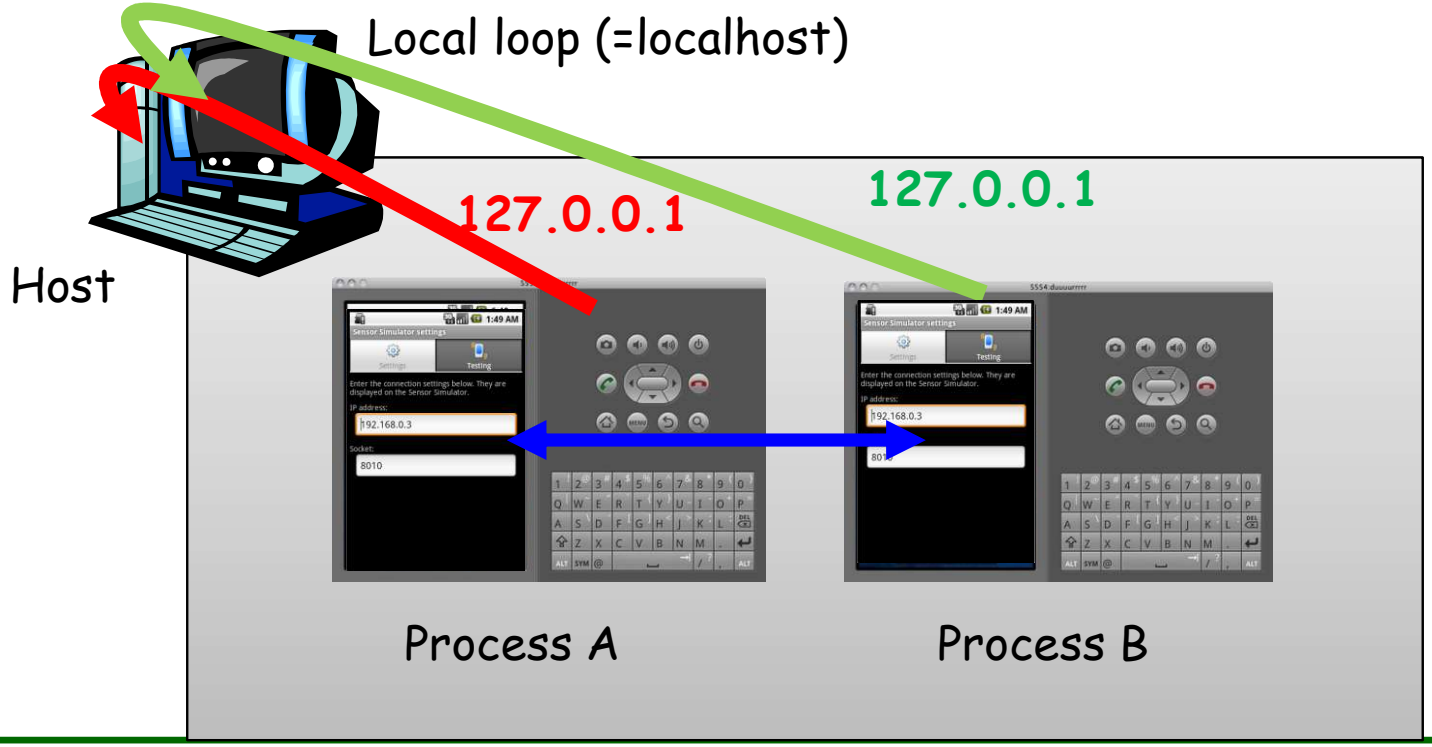
```
modifiedSentence = inFromServer.readLine();

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

clientSocket.close();

    }
}
```

NOTE: 127.0.0.1

IP address: 127.0.0.1 ?

a special purpose address reserved for use on each computer. → localhost or local loop

Used to access a local computer's TCP/IP network resources (or to test server-client apps locally)



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()));
```

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

TCPServer.java

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

Java Socket Programming

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.*;
```

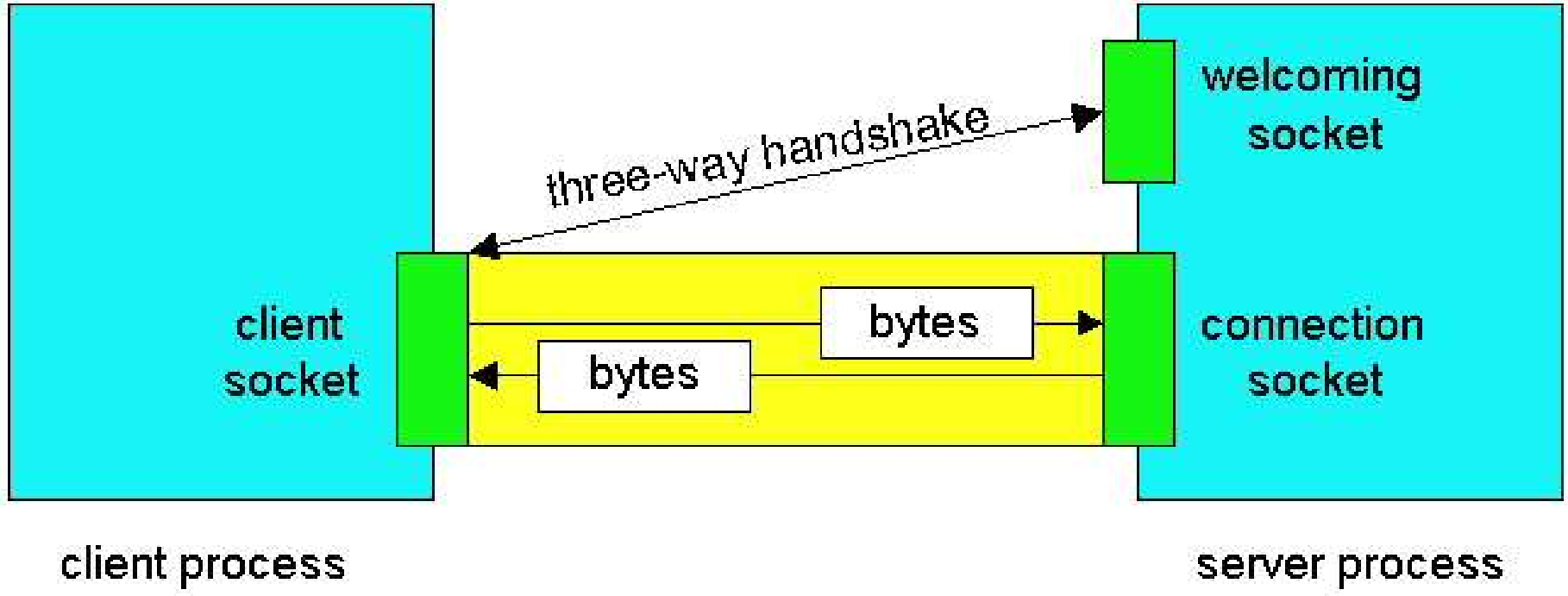
Socket class

- Corresponds to active **TCP sockets** only!
  - **client** sockets
  - socket returned by **accept();**
- Server-side listen (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.
  - Constructor and Methods
    - **Socket(String host, int port):** Creates a stream socket and connects it to the specified port number on the named host.
    - InputStream getInputStream()
    - OutputStream getOutputStream()
    - close()
- java.net.ServerSocket
  - Implements server sockets.
  - Waits for requests to come in over the network.
  - Performs some operation based on the request.
  - Constructor and Methods
    - **ServerSocket(int port) :** Creates a server socket and binds it to the specified local port number
    - Socket **Accept():** Listens for a connection to be made to this socket and accepts it. This method blocks until a connection is made.

Sockets



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

Socket Constructors

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

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

Socket(String hostname, int port);

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

Socket I/O

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

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

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`**.

End.