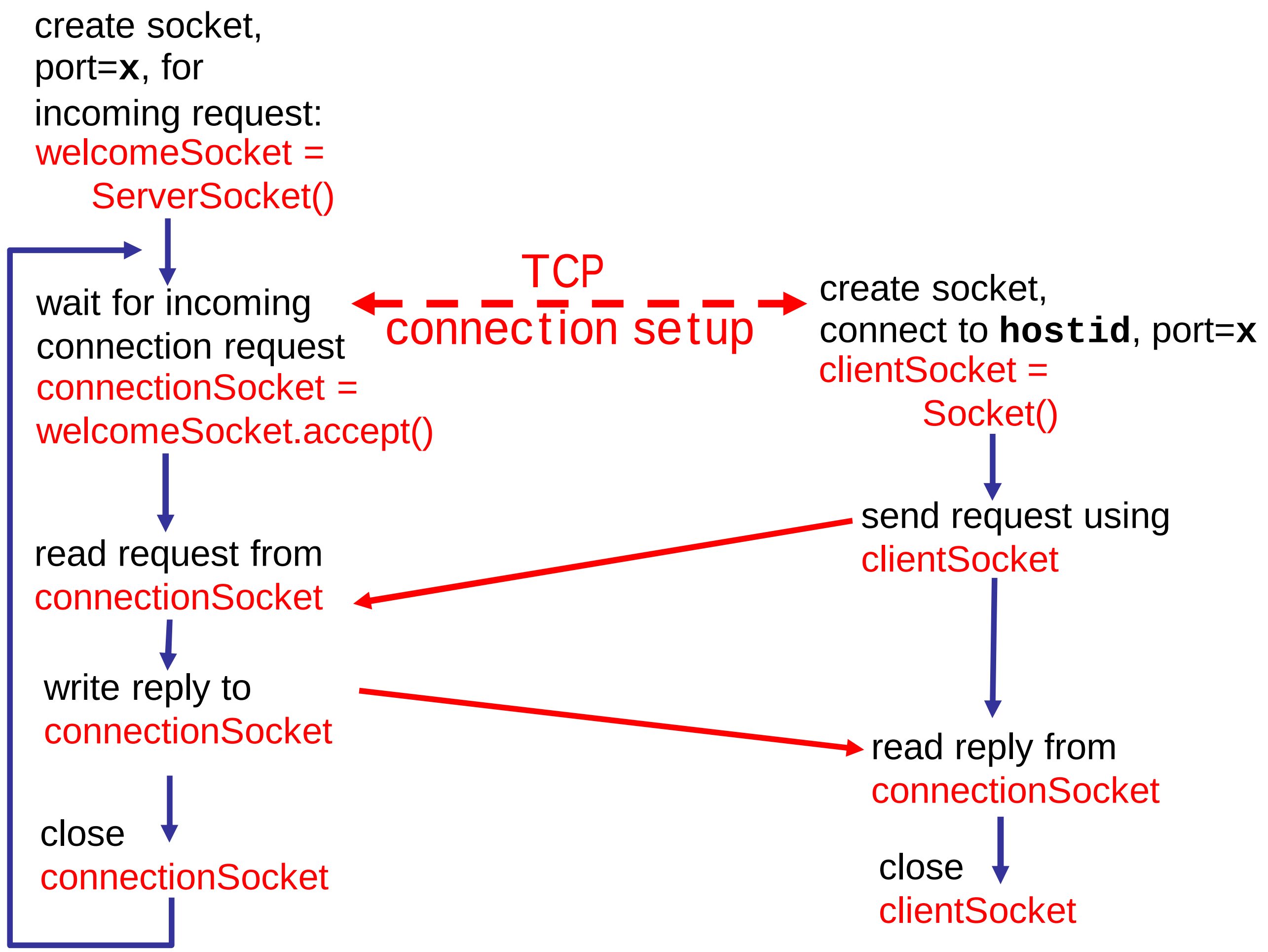
Socket programing

socket

- A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program
- The java.net package provides two classes--Socket and ServerSocket--that implement the client side of the connection and the server side of the connection, respectively.

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 connectionSocket
close connectionSocket
close clientSocket

- Really only 3 additional classes are needed
- **java.net.InetAddress**
- **java.net.Socket**
- **java.net.ServerSocket**

How to Open a Socket?
-Client-

Socket MyClient;

MyClient = new Socket("Machine name", PortNumber);

How to Open a Socket? -Client-

- Machine name is the machine you are trying to open a connection to(ex: ip address or workstation name), and PortNumber is the port (a number) on which the server you are trying to connect to is running.
- When selecting a port number, you should note that port numbers between 0 and 1,023 are reserved for privileged users (that is, super user or root).

How to Open a Socket? -Client-

- With exception handling, the code look like following:

```
Socket MyClient;
try {
    MyClient = new Socket("Machine name", PortNumber);
}
catch (IOException e) {
    System.out.println(e);
}
```

How to Open a Socket? -Server-

```
ServerSocket MyService;
try {
    MyService = new ServerSocket(PortNumber);
}
catch (IOException e) {
    System.out.println(e);
}
```

How to Open a Socket? -Server-

- When implementing a server you also need to create a socket object from the ServerSocket in order to listen for and accept connections from clients.

```
 Socket clientSocket = null;
 try {
     serviceSocket = MyService.accept();
 }
catch (IOException e) {
     System.out.println(e);
}
```

How Do I Create an Input Stream? -Client-

- On the client side, you can use the DataInputStream class to create an input stream to receive response from the server:

```
DataInputStream input;
try {
     input = new    DataInputStream(MyClient.getInputStream());
}
catch (IOException e) {
    System.out.println(e);
}
```

How Do I Create an Input Stream? -Client-

- The class DataInputStream allows you to read lines of text and Java primitive data types in a portable way.
- It has methods such as read, readChar, readInt, readDouble, and readLine,.
- Use whichever function you think suits your needs depending on the type of data that you receive from the server.

How Do I Create an Input Stream? -Server-

- On the server side, you can use DataInputStream to receive input from the client

```
DataInputStream input;
try {
     input = new DataInputStream(serviceSocket.getInputStream());
 }
catch (IOException e) {
    System.out.println(e);
 }
```

How do I Create an Output Stream? -Client-

- On the client side, you can create an output stream to send information to the server socket using the class PrintStream or DataOutputStream of java.io:

How do I Create an Output Stream? -Client-

```
PrintStream output;
try {
    output = new PrintStream(MyClient.getOutputStream());
}
catch (IOException e) {
    System.out.println(e);
}
```

How do I Create an Output Stream? -Client-

- The class PrintStream has methods for displaying textual representation of Java primitive data types.
- you may use the DataOutputStream

How do I Create an Output Stream? -Client-

```
 DataOutputStream output;
try {
      output = new DataOutputStream(MyClient.getOutputStream());
}
catch (IOException e) {
     System.out.println(e);
}
```

How do I Create an Output Stream? -Client-

- The class DataOutputStream allows you to write Java primitive data types; many of its methods write a single Java primitive type to the output stream.
- The method writeBytes is a useful one.

How do I Create an Output Stream? -Server-

- On the server side, you can use the class PrintStream to send information to the client.

```
PrintStream output;
try {
      output = new
 PrintStream(serviceSocket.getOutputStream());
 }
catch (IOException e) {
     System.out.println(e);
 }
```

How do I Create an Output Stream? -Server-

- You can use the class DataOutputStream as mentioned

```
DataOutputStream output;
try {
    output = new
DataOutputStream(serviceSocket.getOutputStre
am());
}
catch (IOException e) {
    System.out.println(e);
}
```

How Do I Close Sockets? -Client-

- You should always close the output and input stream before you close the socket.

```
try {
      output.close();
      input.close();
      MyClient.close();
}
catch (IOException e) {
      System.out.println(e);
}
```

How Do I Close Sockets? -Server-

```
try {
      output.close();
      input.close();
      serviceSocket.close();
      MyService.close();
 }
catch (IOException e) {
      System.out.println(e);
}
```

Examples
-Client-

- When programming a client, you must follow these four steps:

  1. Open a socket.
  2. Open an input and output stream to the Socket.
  3. Read from and write to the socket according to the server's protocol.
  4. Clean up.

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

Create client socket, connect to server → `Socket clientSocket = new Socket("hostname", 6789);`

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

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;
```

Create welcoming socket at port 6789

```
      ServerSocket welcomeSocket = new ServerSocket(6789);

      while(true) {
```

Wait, on welcoming socket for contact by client

```
          Socket connectionSocket = welcomeSocket.accept();
```

Create input stream, attached to socket

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

Example: Java server (TCP), cont

Create output stream, attached to socket

Read in line from socket

Write out line to socket

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

            clientSentence = inFromClient.readLine();

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

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

End of while loop, loop back and wait for another client connection

Socket programming with UDP

UDP: no “connection” between client and server

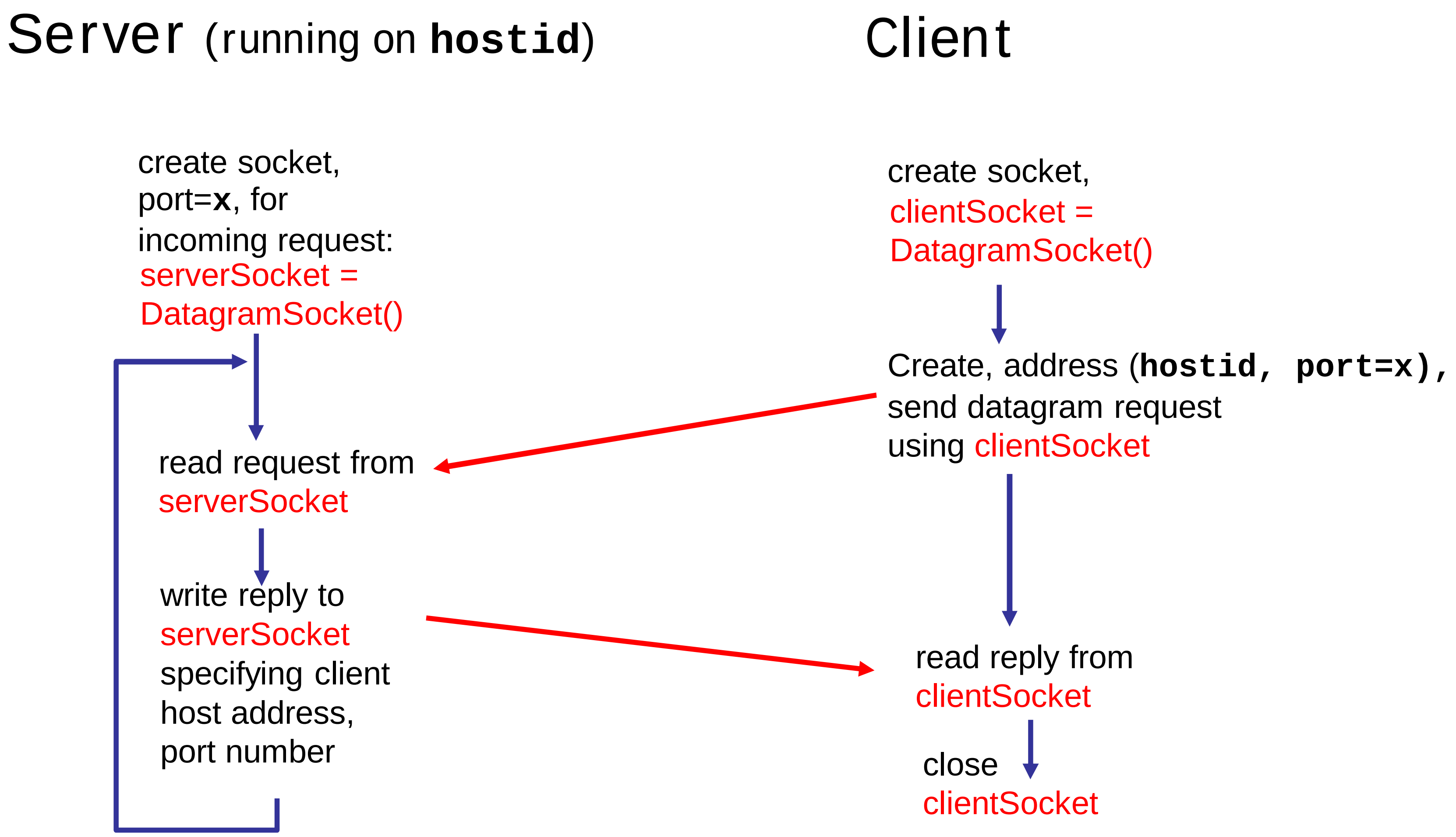
- no handshaking
- sender explicitly attaches IP address and port of destination
- server must extract IP address, port of sender from received datagram

UDP: transmitted data may be received out of order, or lost

application viewpoint

UDP provides unreliable transfer of groups of bytes (“datagrams”) between client and server

Client/server socket interaction: UDP


Server (running on **hostid**)
Client
create socket, port=**x**, for incoming request: serverSocket = DatagramSocket()
read request from serverSocket
write reply to serverSocket specifying client host address, port number
create socket, clientSocket = DatagramSocket()
Create, address (**hostid, port=x**), send datagram request using clientSocket
read reply from clientSocket
close clientSocket

Example: Java client (UDP)

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

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

Create input stream

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

Create client socket

```
      DatagramSocket clientSocket = new DatagramSocket();
```

Translate hostname to IP address using DNS

```
      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 addr, port

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

Send datagram to server

```
    clientSocket.send(sendPacket);

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

Read datagram from server

```
    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 client connection