Network Programming

Java Sockets Programming (Low-Level Networking)

Sockets

- A socket is a construct that represents one *end-point* of a *two-way* communication channel between two programs running on the network.

Sockets (cont)

- Using sockets, the OS provides processes a file-like access to the channel.
  - i.e., sockets are allocated a file descriptor, and processes can access (read/write) the socket by specifying that descriptor

Sockets (cont)

- A specific socket is identified by the machine's IP and a port within that machine.
- A socket stores the IP and port number of the other end-point computer of the channel.

Sockets (cont)

- Sockets can be used with both the TCP and the UDP transport layer protocols.
- When writing to a socket, the written bytes are sent to the other computer and port (e.g., over TCP/IP or UDP).
  - That is, remote IP and port are attached to the packets.

Sockets (cont)

- When OS receives packets on the network, it uses their destination port to decide which socket should get the received bytes.

Sockets (cont)

- Sockets can:
  1. Connect to a remote machine
  2. Send data
  3. Receive data
  4. Close a connection
  5. Bind to a port
  6. Listen for incoming connection
  7. Accept connections from remote machines on a bound port

Java Sockets Programming
(Low-Level Networking)

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

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

Java Sockets Programming
(Low-Level Networking)

- 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

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

The Java Socket Class

- The Socket class supports the
  1. Connect to a remote machine `[socket = new Socket(…)]`
  2. Send data `[socket.write()]`
  3. Receive data `[socket.read()]`
  4. Close a connection `[socket.close()]`
  - Normally a socket is encapsulated in a `InputStream` class or a `Reader` class.

The Java ServerSocket Class

- The ServerSocket class additionally supports the
  5. Bind to a port
     `[server_socket.bind()]`
  6. Listen for incoming connection
     `[server_socket.listen()]`
  7. Accept connections from remote machines on a bound port
     `[server_socket.accept()]`

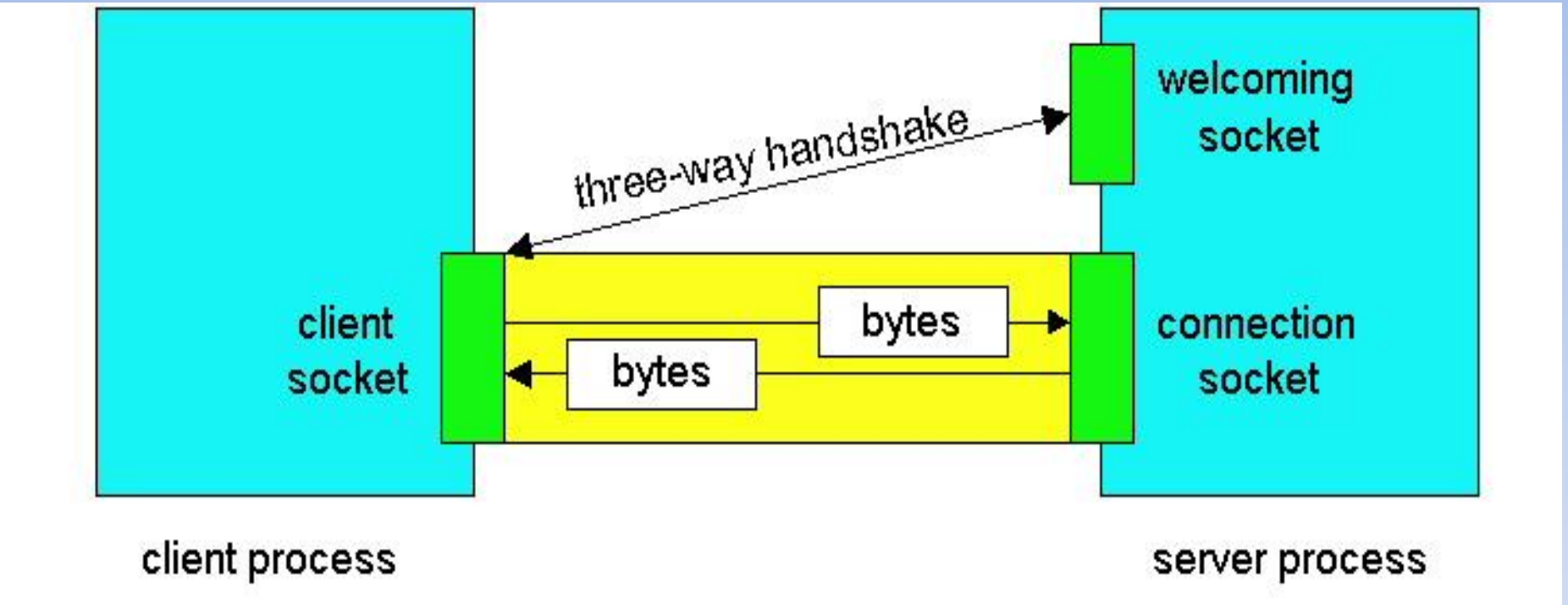
JAVA TCP Sockets

- 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 TCP Sockets

- 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)
    - 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(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…

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

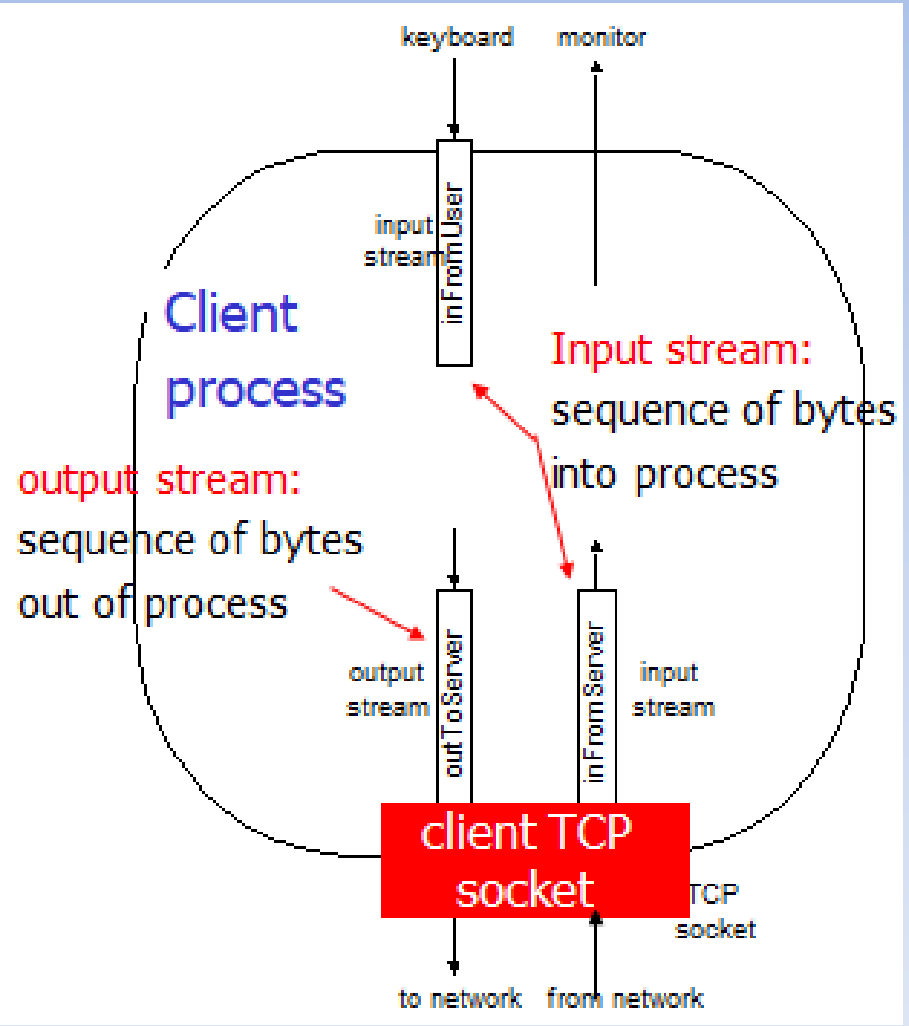
- Example client-server app:

✓ client reads line from standard input (**inFromUser** stream) , sends to server via socket (**outToServer** stream)

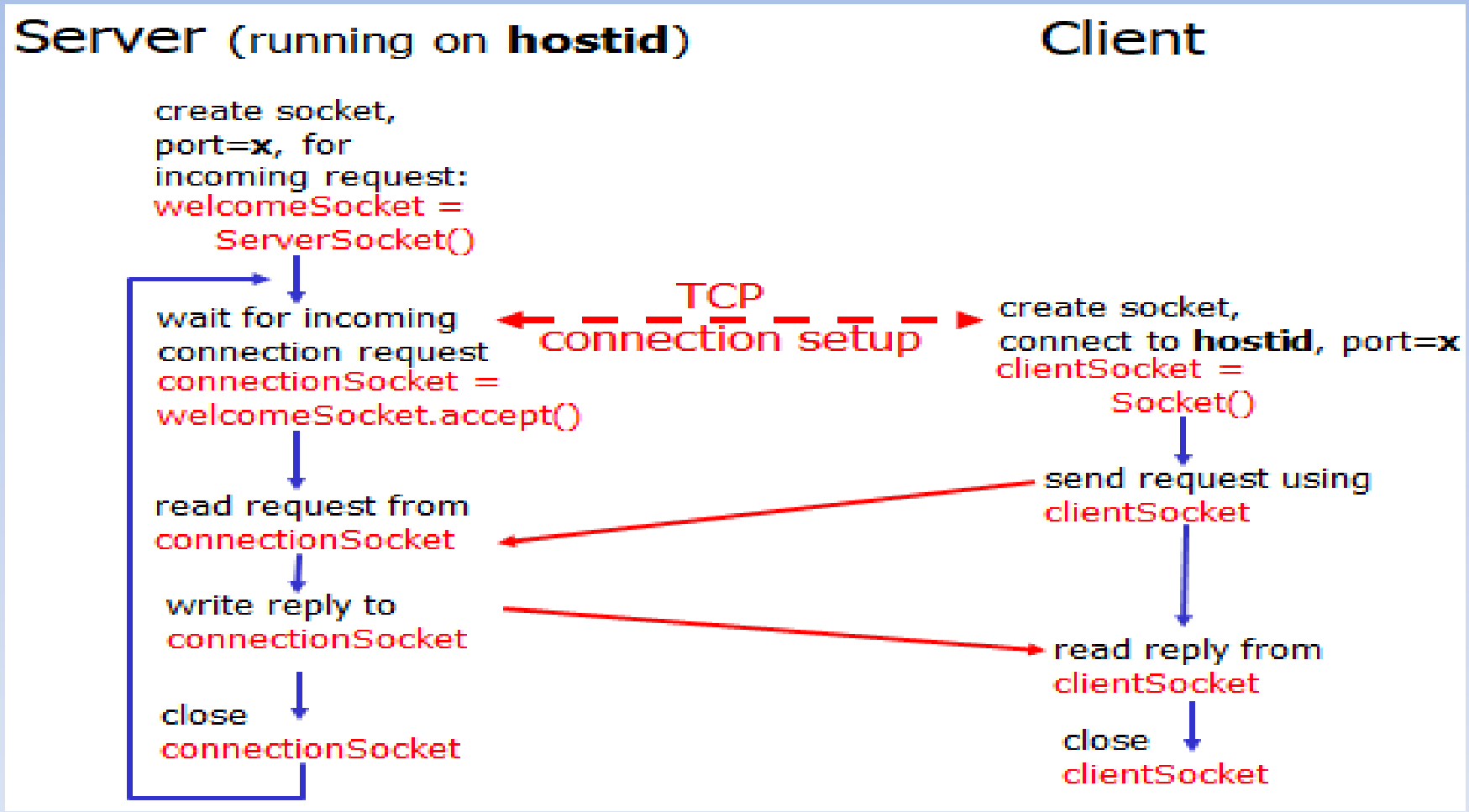
✓ server reads line from socket

✓ 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 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();

    }
}
```

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 cont.

```
            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 (cont.)

- 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 a, 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

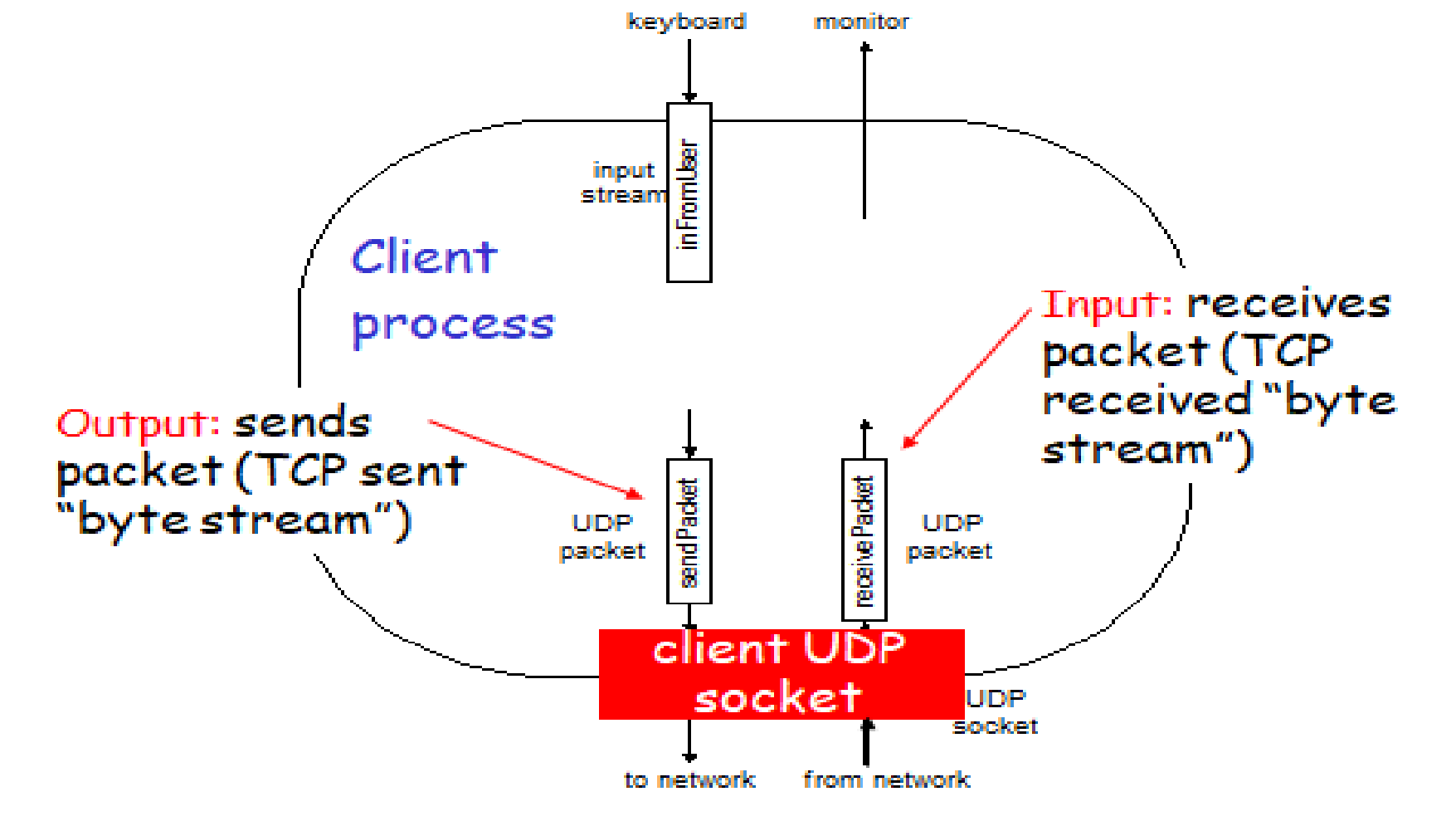
- ✓**byte[] getData();** (Returns the data buffer).
- ✓**void setData(byte[] buf);** **(Set the data buffer for this packet).**
- ✓**void setAddress(InetAddress a);**

  (Sets the IP address of the machine to which this datagram is being sent.

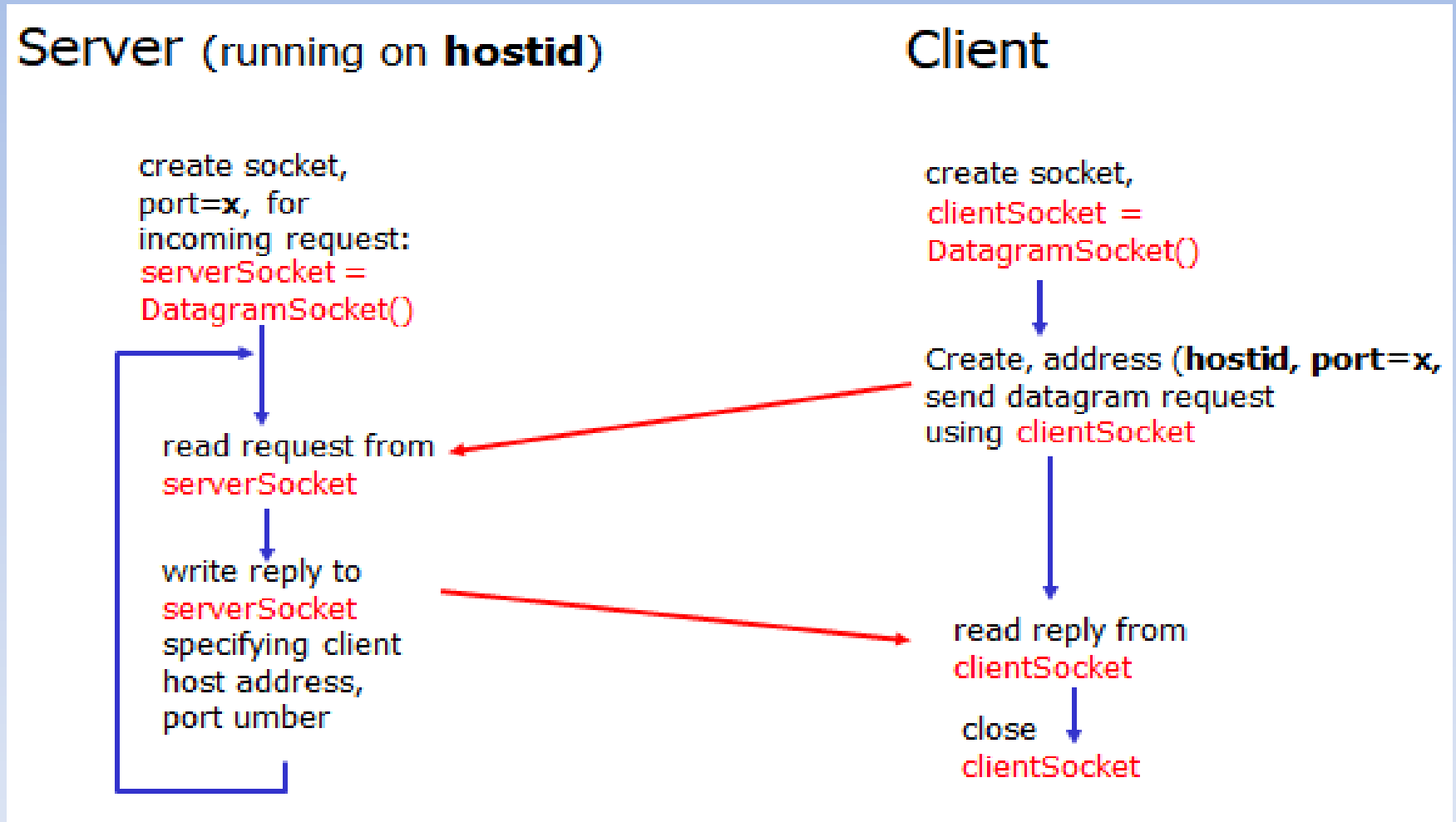
DatagramPacket methods

- ✓ **`void setPort(int port);`** Sets the port number on the remote host to which this datagram is being sent.
- ✓ **`InetAddress getAddress();`** Returns the IP address of the machine to which this datagram is being sent or from which the datagram was received.
- ✓ **`int getPort();`** Returns the port number on the remote host to which this datagram is being sent or from which the datagram was received.

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 cont.

```
 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 cont.

```
        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);

      }
   }
}
```

Socket functional calls

- socket (): Create a socket
- bind(): bind a socket to a local IP address and port #

- listen(): passively waiting for connections
- connect(): initiating connection to another socket
- accept(): accept a new connection

- Write(): write data to a socket
- Read(): read data from a socket
- sendto(): send a datagram to another UDP socket
- recvfrom(): read a datagram from a UDP socket

- close(): close a socket (tear down the connection)