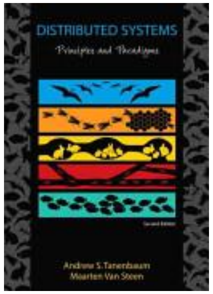
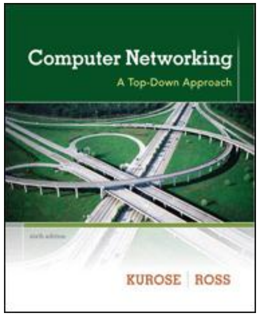
Chapter 0: COMPUTER NETWORKING
Part 2

Communications in Distributed Systems

Client-server paradigm and Socket Programming



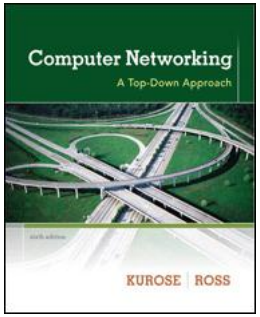


Thanks to the authors of the textbook [**TS**] and [**KR**] for providing the base slides. I made several changes/additions.
These slides may incorporate materials kindly provided by Prof. Dakai Zhu.
So I would like to thank him, too.

Turgay Korkmaz
korkmaz@cs.utsa.edu

Chapter 0: Computer Networking

- Layered Protocols
- Grand tour of computer networking, the Internet
- Client-server paradigm,
- Socket Programming



Objectives

- To understand how processes communicate (the heart of distributed systems)
- To understand computer networks and their layers (part 1)
- To understand client-server paradigm and low-level message passing using **sockets**

Request (R) protocol

Request-Reply (RR) protocol

Request-Reply-Acknowledgement (RRA) protocol

CLIENT-SERVER COMMUNICATION MODELS

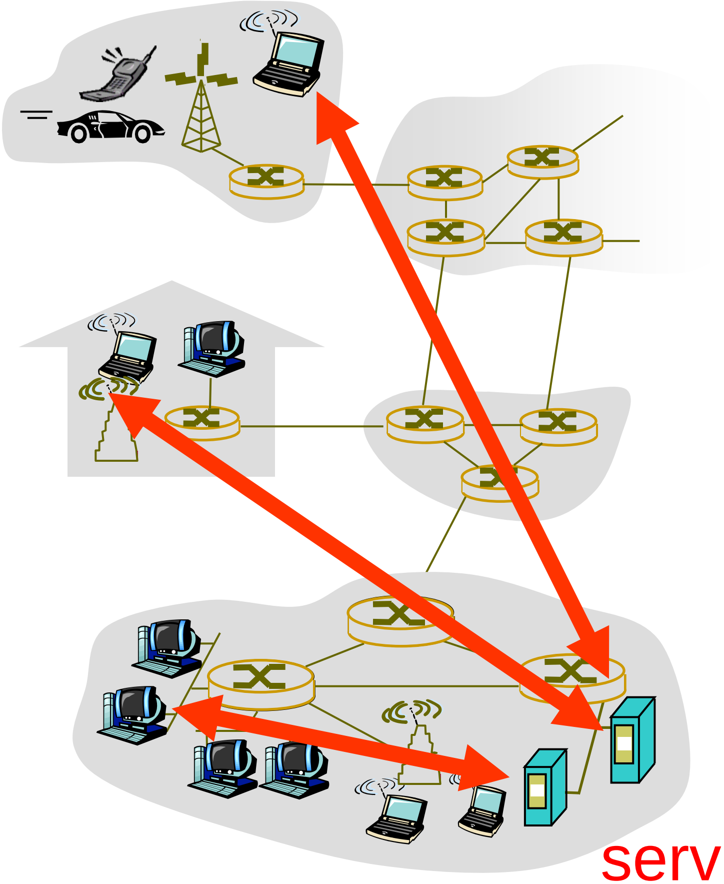
Client-server architecture

clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Clients and servers communicate through

- Socket, RPC, RMI



servers:

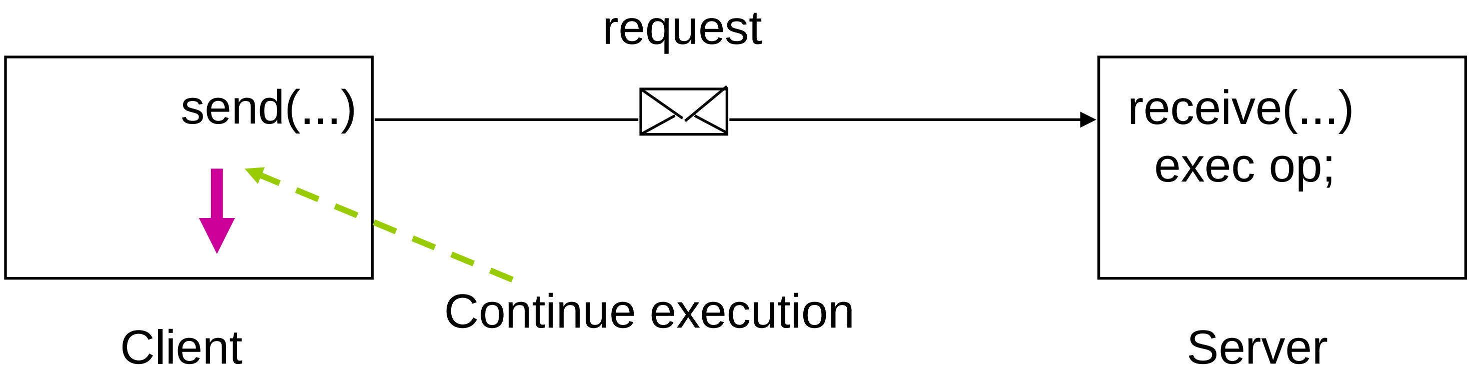
- always-on host
- permanent IP address
- server farms for scaling

From Computer Networking by Kurose and Ross.

Request Protocol (R)

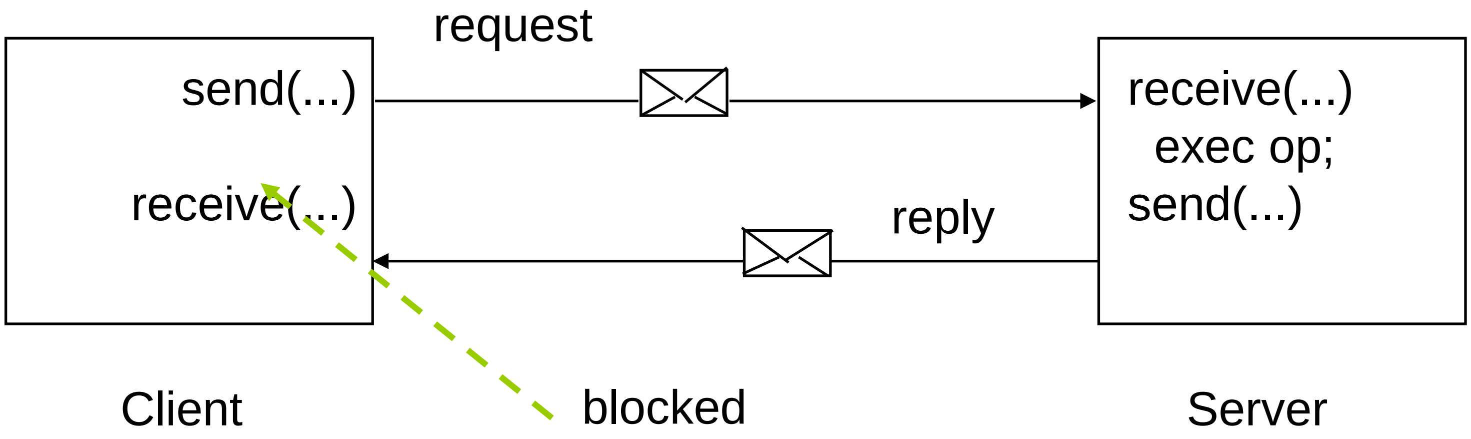
- If service
  - does not have output parameters and
  - does not have a return type

client may not want to wait for server to finish.



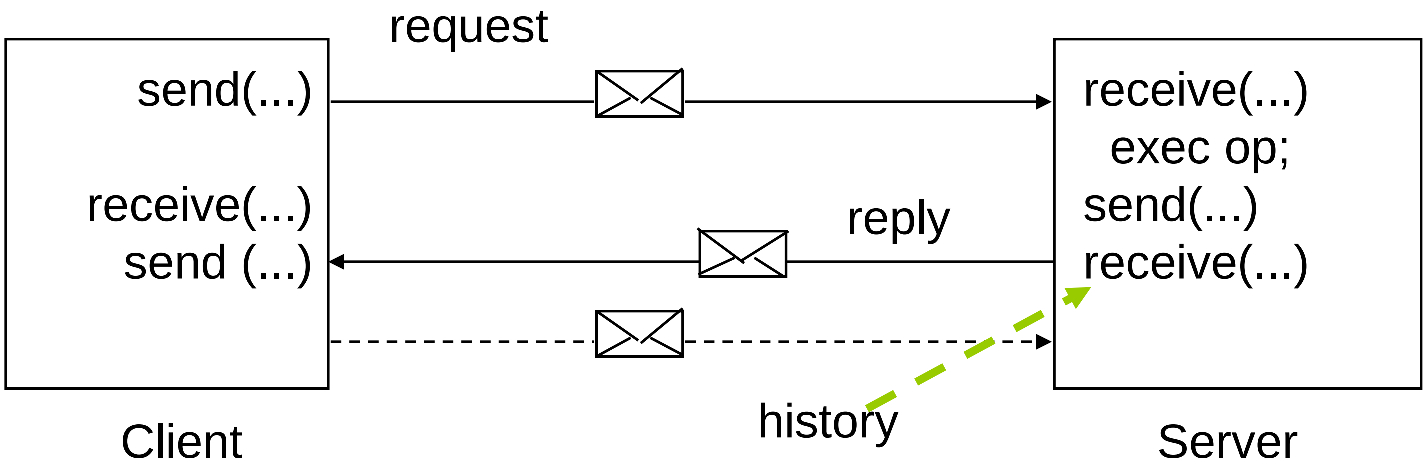
Request-Reply protocol (RR)

- To be applied if client expects result from server
  - Client requests service execution from server through request message, and
  - Delivery of service result in reply message
- ***Most client-server interactions are built on RR protocol***



Request-Reply-Acknowledge Protocol (RRA)

- In addition to RR protocol, client sends acknowledgement after it received reply
- Acknowledgement sent asynchronously

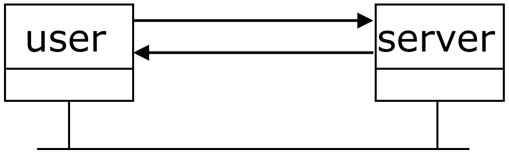


Issues in Client-Server Communication

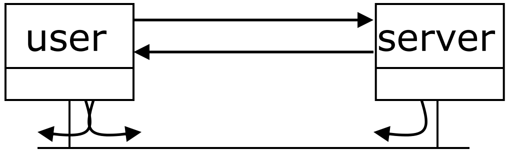
- Addressing
- Blocking versus non-blocking
- Buffered versus unbuffered
- Reliable versus unreliable
- Server architecture:
  - concurrent versus sequential
- Scalability

Addressing Issues

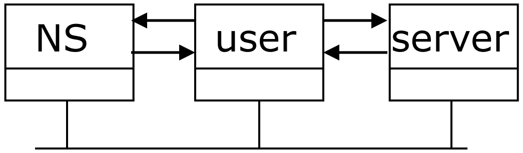
- *Question:* how is the server located?
- Hard-wired address
  - Machine address and process address are known a priori



- Broadcast-based
  - Server chooses address from a sparse address space
  - Client broadcasts request
  - Can cache response for future



- Locate address via name server

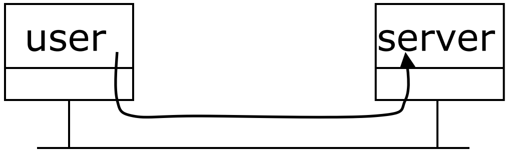


Blocking versus Non-blocking

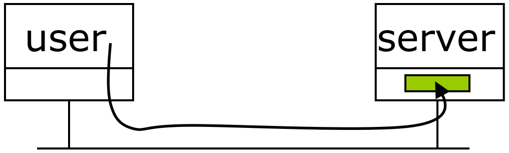
- Blocking communication (synchronous)
  - Sender blocks until message is actually sent
  - Receiver blocks until message is actually received
- Non-blocking communication (asynchronous)
  - Sender returns immediately
  - Receiver does not block either
- Examples:

Buffering Issues

- Unbuffered communication
  - Server must call receive before client can call send

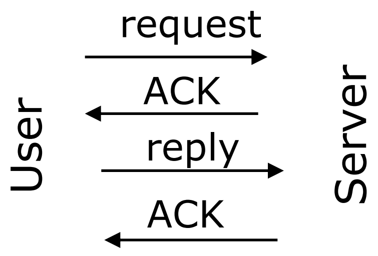


- Buffered communication
  - Client send to a mailbox
  - Server receives from a mailbox

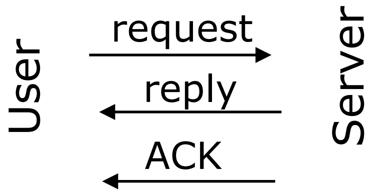


Reliability

- Unreliable channel
  - Need acknowledgements (ACKs)
  - Applications handle ACKs
  - ACKs for both request and reply



- Reliable channel
  - Reply acts as ACK for request
  - Explicit ACK for response



- Reliable communication on unreliable channels
  - Transport protocol handles lost messages

Server Architecture

- Sequential
  - Serve one request at a time
  - Can service multiple requests by employing events and asynchronous communication
- Concurrent
  - Server spawns a process or thread to service each request
  - Can also use a pre-spawned pool of threads/processes (apache)
- Thus servers could be
  - Pure-sequential, event-based, thread-based, process-based
- Discussion: which architecture is most efficient?

Scalability

- *Question:* How can you scale the server capacity?
- Buy bigger machine!
- Replicate
- Distribute data and/or algorithms
- Ship code instead of data
- Cache

Putting it all together: Email

- User uses mail client to compose a message
- Mail client connects to mail server
- Mail server looks up address to destination mail server
- Mail server sets up a connection and passes the mail to destination mail server
- Destination stores mail in input buffer (user mailbox)
- Recipient checks mail at a later time

How do application and middleware layers use the services provided by transport layer?

SOCKETS

From Computer Networking by Kurose and Ross.

Socket programming

Goal: learn how to build client/server application that communicate using sockets

Socket API

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

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

From Computer Networking by Kurose and Ross.

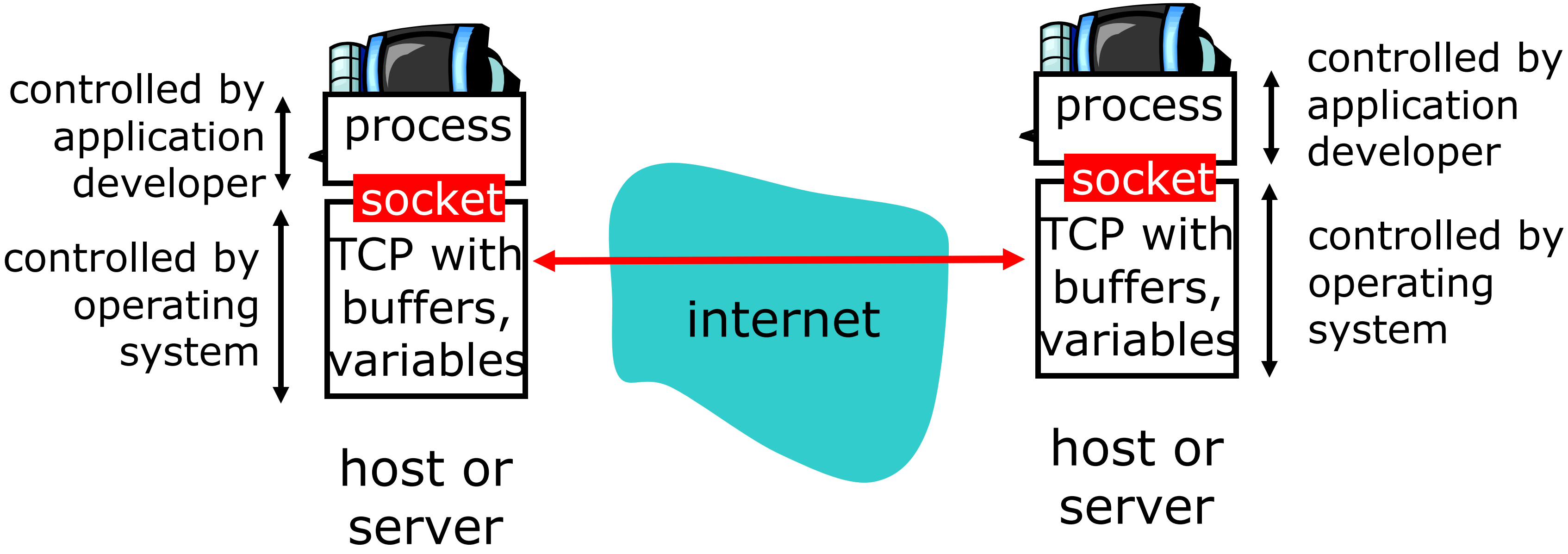
SOCKET PROGRAMMING
C

From Computer Networking by Kurose and Ross.

Socket-programming using TCP

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

TCP service: reliable transfer of **bytes** from one process to another


controlled by application developer
controlled by operating system
process
socket
TCP with buffers, variables
host or server
internet
process
socket
TCP with buffers, variables
host or server
controlled by application developer
controlled by operating system

From Computer Networking by Kurose and Ross.

Socket programming *with TCP*

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:

- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

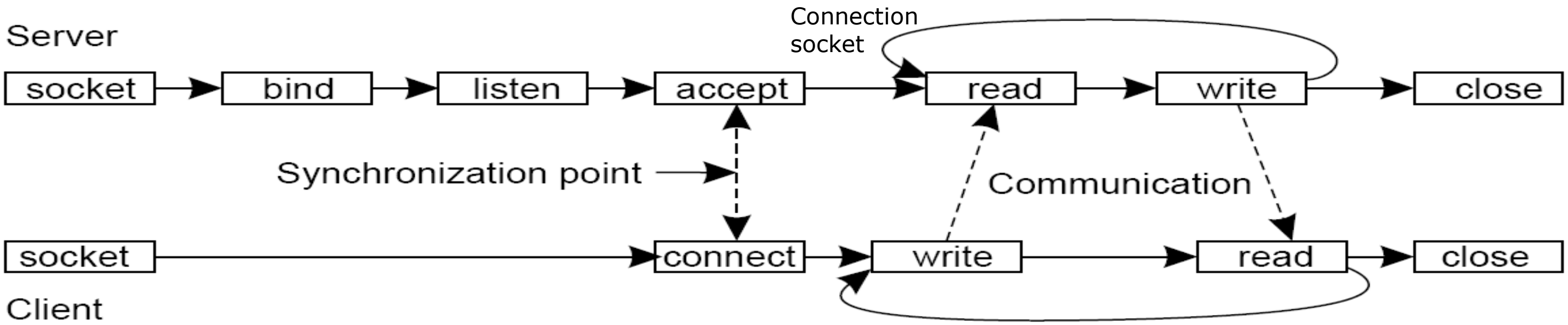
- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint

TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

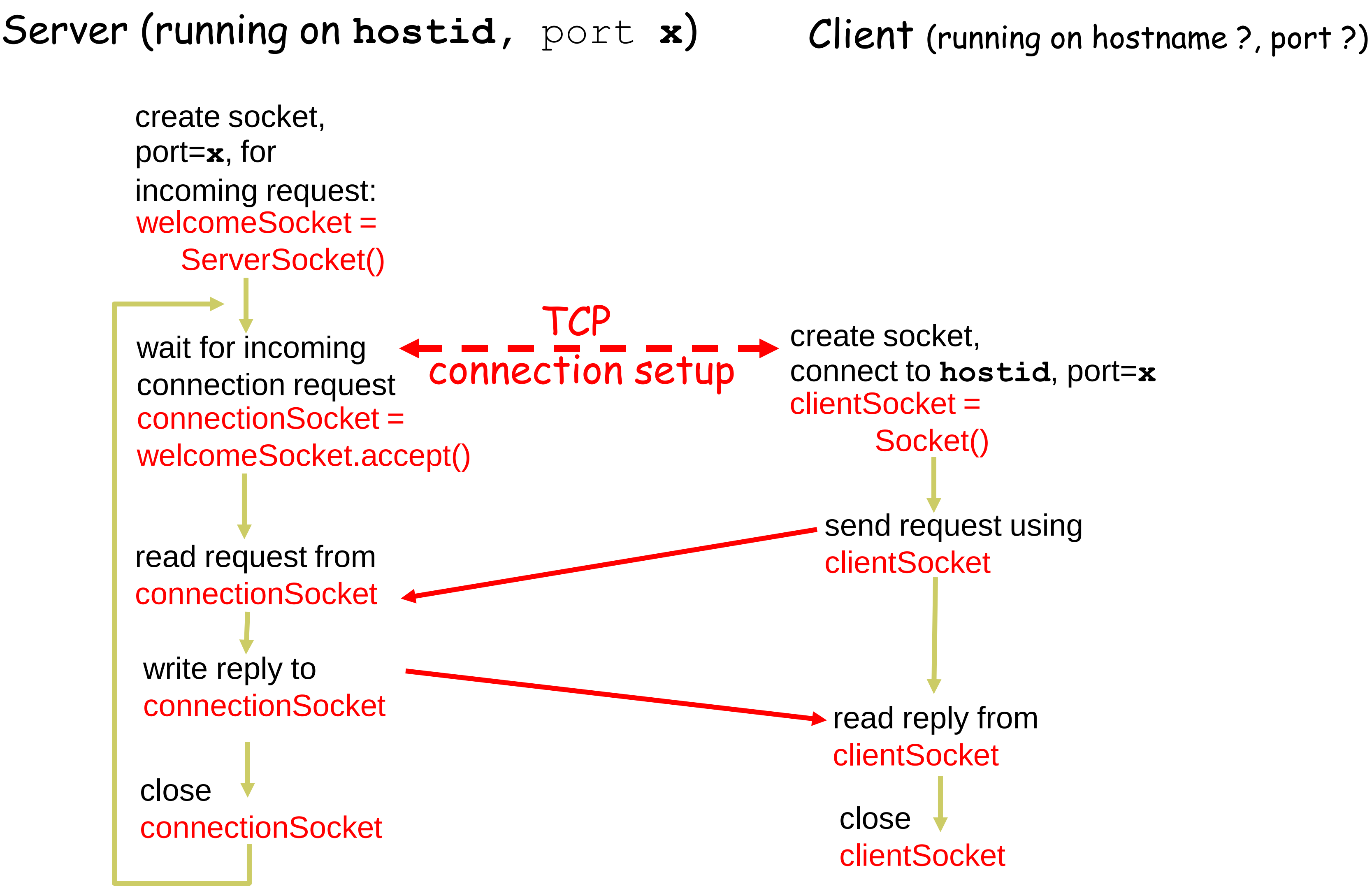
From Computer Networking by Kurose and Ross.

TCP Socket Primitives



Primitive	**Function**
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Recv	Receive some data over the connection
Close	Release the connection

Client/server socket interaction: TCP



Server (running on **hostid**, port **x**)

Client (running on hostname ?, port ?)

create socket,
port=**x**, for
incoming request:
welcomeSocket =
ServerSocket()

wait for incoming
connection request
connectionSocket =
welcomeSocket.accept()

create socket,
connect to **hostid**, port=**x**
clientSocket =
Socket()

send request using
clientSocket

read request from
connectionSocket

write reply to
connectionSocket

read reply from
clientSocket

close
connectionSocket

close
clientSocket

Example: C server (TCP)

```
/* server.c  */
void main(int argc, char *argv[])
{
struct sockaddr_in sad; /* structure to hold an IP address */
struct sockaddr_in cad;
int welcomeSocket, connectionSocket; /* socket descriptor */
struct hostent *ptrh; /* pointer to a host table entry */

char clientSentence[128];
char capitalizedSentence[128];

port = atoi(argv[1]);

welcomeSocket = socket(PF_INET, SOCK_STREAM, 0);
            memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
            sad.sin_family = AF_INET; /* set family to Internet */
            sad.sin_addr.s_addr = INADDR_ANY; /* set the local IP address */
            sad.sin_port = htons((u_short)port);/* set the port number */
bind(welcomeSocket, (struct sockaddr *)&sad, sizeof(sad));
```

Create welcoming socket at port
&
Bind a local address

Example: C server (TCP), cont

```
/* Specify the maximum number of clients that can be queued */
listen(welcomeSocket, 10)

while(1) {

    connectionSocket=accept(welcomeSocket, (struct sockaddr *)&cad, &alen);

    n=read(connectionSocket, clientSentence, sizeof(clientSentence));

    /* capitalize Sentence and store the result in capitalizedSentence*/


    n=write(connectionSocket, capitalizedSentence, strlen(capitalizedSentence)+1);

    close(connectionSocket);
    }
}
```

Wait, on welcoming socket
for contact by a client

Write out the result to socket

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

Example: C client (TCP)

```
/* client.c  */
void main(int argc, char *argv[])
{
struct sockaddr_in sad; /* structure to hold an IP address */
int clientSocket; /* socket descriptor */
struct hostent *ptrh; /* pointer to a host table entry */

char Sentence[128];
char modifiedSentence[128];

host = argv[1]; port = atoi(argv[2]);

clientSocket = socket(PF_INET, SOCK_STREAM, 0);
        memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
        sad.sin_family = AF_INET; /* set family to Internet */
        sad.sin_port = htons((u_short)port);
        ptrh = gethostbyname(host); /* Convert host name to IP address */
        memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);
connect(clientSocket, (struct sockaddr *)&sad, sizeof(sad));
```

Create client socket,
connect to server

Example: C client (TCP), cont

Get input stream from user

```
gets(Sentence);
```

Send line to server

```
n=write(clientSocket, Sentence, strlen(Sentence)+1);
```

Read line from server

```
n=read(clientSocket, modifiedSentence, sizeof(modifiedSentence));

printf("FROM SERVER: %s\n”,modifiedSentence);
```

Close connection

```
close(clientSocket);

}
```

Socket programming *with UDP*

UDP: no "connection" between client and server

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

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

application viewpoint

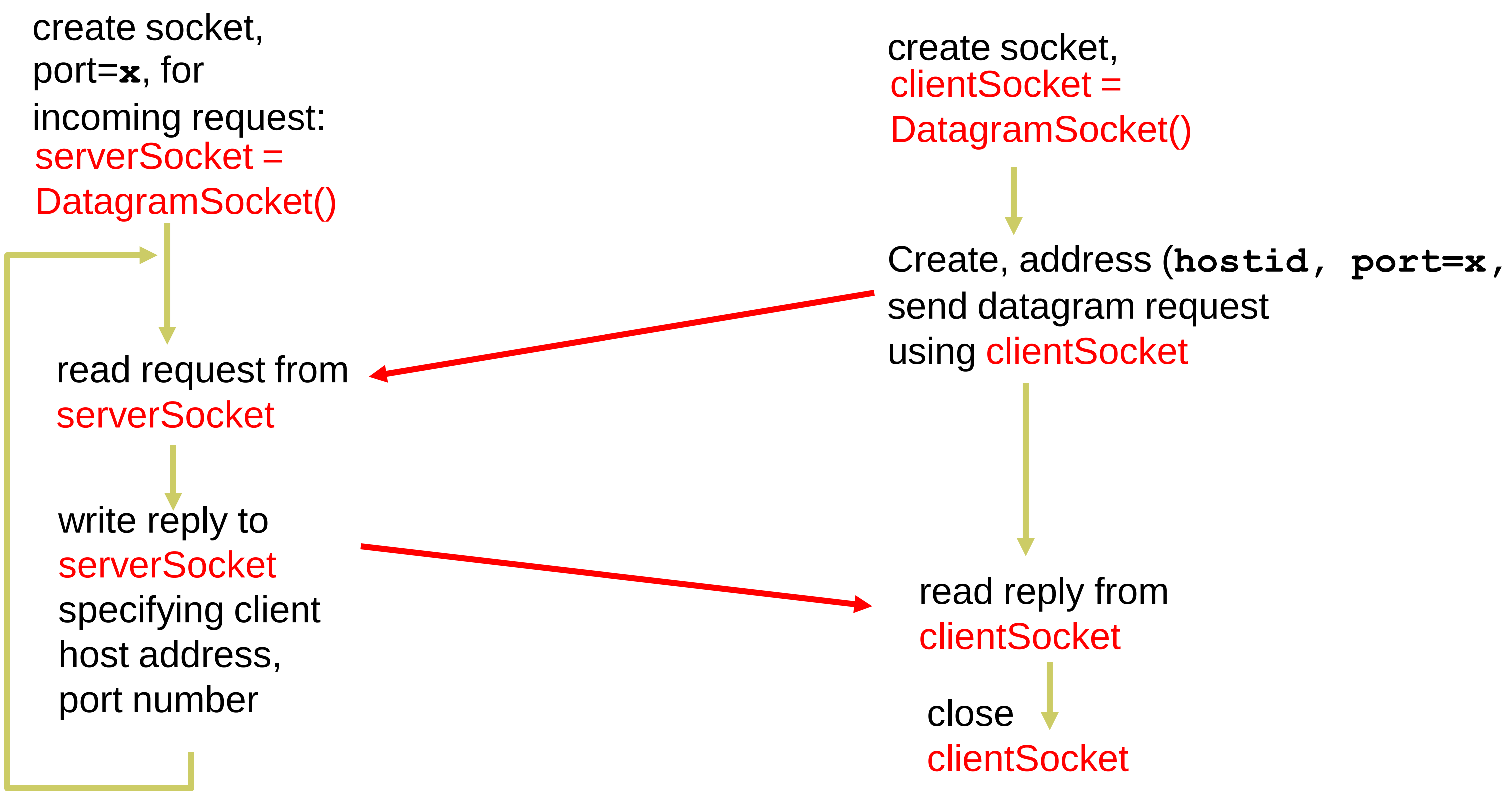
UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server

From Computer Networking by Kurose and Ross.

Client/server socket interaction: UDP

Server (running on **hostid**, port **x**)

Client (running on hostname ?, port ?)



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: C server (UDP)

```
/* server.c  */
void main(int argc, char *argv[])
{
struct sockaddr_in sad; /* structure to hold an IP address */
struct sockaddr_in cad;
int serverSocket; /* socket descriptor */
struct hostent *ptrh; /* pointer to a host table entry */

char clientSentence[128];
char capitalizedSentence[128];

port = atoi(argv[1]);

serverSocket = socket(PF_INET, SOCK_DGRAM, 0);
        memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
        sad.sin_family = AF_INET; /* set family to Internet */
        sad.sin_addr.s_addr = INADDR_ANY; /* set the local IP address */
        sad.sin_port = htons((u_short)port);/* set the port number */
bind(serverSocket, (struct sockaddr *)&sad, sizeof(sad));
```

Create welcoming socket at port
&
Bind a local address

Example: C server (UDP), cont

```
while(1) {

    n=recvfrom(serverSocket, clientSentence, sizeof(clientSentence), 0
                    (struct sockaddr *) &cad, &addr_len );

    /* capitalize Sentence and store the result in capitalizedSentence*/

    n=sendto(connectionSocket, capitalizedSentence, strlen(capitalizedSentence)+1,0
                (struct sockaddr *) &cad, &addr_len);

    close(connectionSocket);
    }
}
```

Receive messages from clients

Write out the result to socket

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

Example: C client (UDP)

```
/* client.c  */
void main(int argc, char *argv[])
{
struct sockaddr_in sad; /* structure to hold an IP address */
int clientSocket; /* socket descriptor */
struct hostent *ptrh; /* pointer to a host table entry */

char Sentence[128];
char modifiedSentence[128];

host = argv[1]; port = atoi(argv[2]);

clientSocket = socket(PF_INET, SOCK_DGRAM, 0);

/* determine the server's address */
        memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
        sad.sin_family = AF_INET; /* set family to Internet */
        sad.sin_port = htons((u_short)port);
        ptrh = gethostbyname(host); /* Convert host name to IP address */
        memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);
```

Create client socket,
NO connection to server

Example: C client (UDP), cont.

Get input stream from user

```
gets(Sentence);
```

Send line to server

```
addr_len =sizeof(struct sockaddr);
n=sendto(clientSocket, Sentence, strlen(Sentence)+1,
                  (struct sockaddr *) &sad, addr_len);
```

Read line from server

```
n=recvfrom(clientSocket, modifiedSentence, sizeof(modifiedSentence),
                  (struct sockaddr *) &sad, &addr_len);
```

```
printf("FROM SERVER: %s\n”,modifiedSentence);
```

Close connection

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

Other related functions

- getpeername( )
- gethostbyname( )
- gethostbyaddr( )

- getsockopt( )
- setsockopt ( )
- signal(SIGINT,sigf);

```
if ( (pid=fork()) == 0) {
   /* CHILD PROC */
   close(welcomeSocket);
   /* give service */
   exit(0);
}
/* PARENT PROC */
close(connectionSocket);
```

Waiting something from both socket and stdin

```
FD_ZERO(&rset);
FD_SET(welcomeSocket, &rset);
FD_SET(fileno(stdin), &rset);
maxfd =max(welcomeSocket,fileno(stdin)) + 1;
select(maxfd, &rset, NULL,  NULL, NULL);
if (FD_ISSET(fileno(stdin), &rset)){
   /* read something from stdin */
}
```

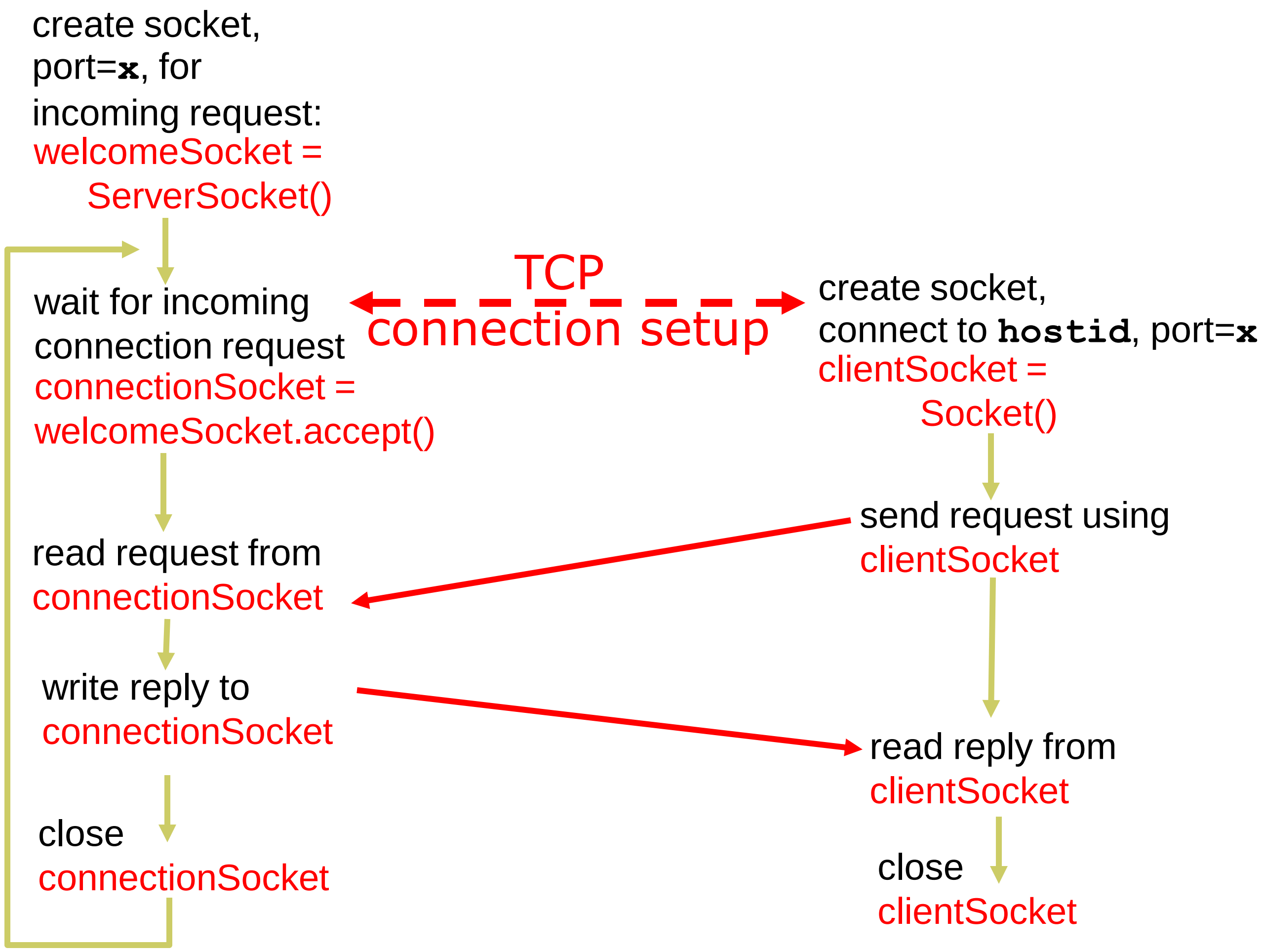
SOCKET PROGRAMMING JAVA

From Computer Networking by Kurose and Ross.

Client/server socket interaction: TCP

Server (running on **hostid, port x**)

Client (running on hostname ?, port ?)



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
clientSocket

close
connectionSocket

close
clientSocket

From Computer Networking by Kurose and Ross.

Socket programming with TCP

Example client-server app:

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

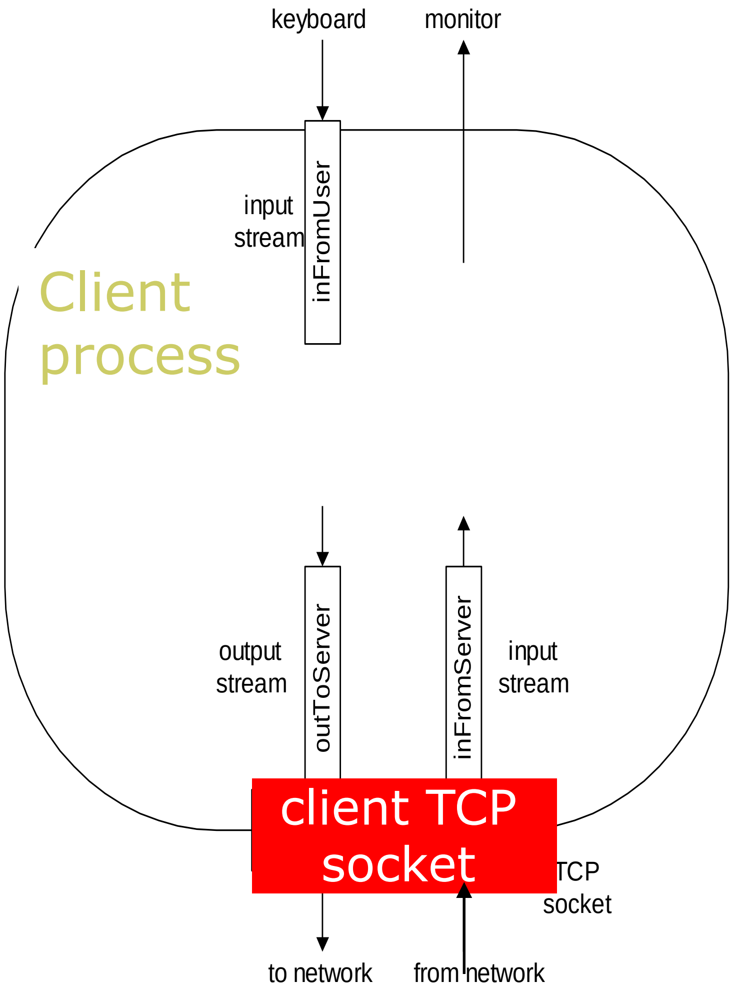
2) server reads line from socket

3) server converts line to uppercase, sends back to client

4) client reads, prints modified line from socket (**`inFromServer`** stream)

Stream jargon

- A stream is a sequence of characters that flow into or out of a process.
- An input stream is attached to some input source for the process, e.g., keyboard or socket.
- An output stream is attached to an output source, e.g., monitor or socket.



From Computer Networking by Kurose and Ross.

Example: Java server (TCP)

```
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

From Computer Networking by Kurose and Ross.

Example: Java server (TCP), cont

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

From Computer Networking by Kurose and Ross.

Example: Java client (TCP)

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

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

Create
input stream

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

Create
client socket,
connect to server

```
        Socket clientSocket = new Socket(“localhost", 6789);
```

Create
output stream
attached to socket

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

From Computer Networking by Kurose and Ross.

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

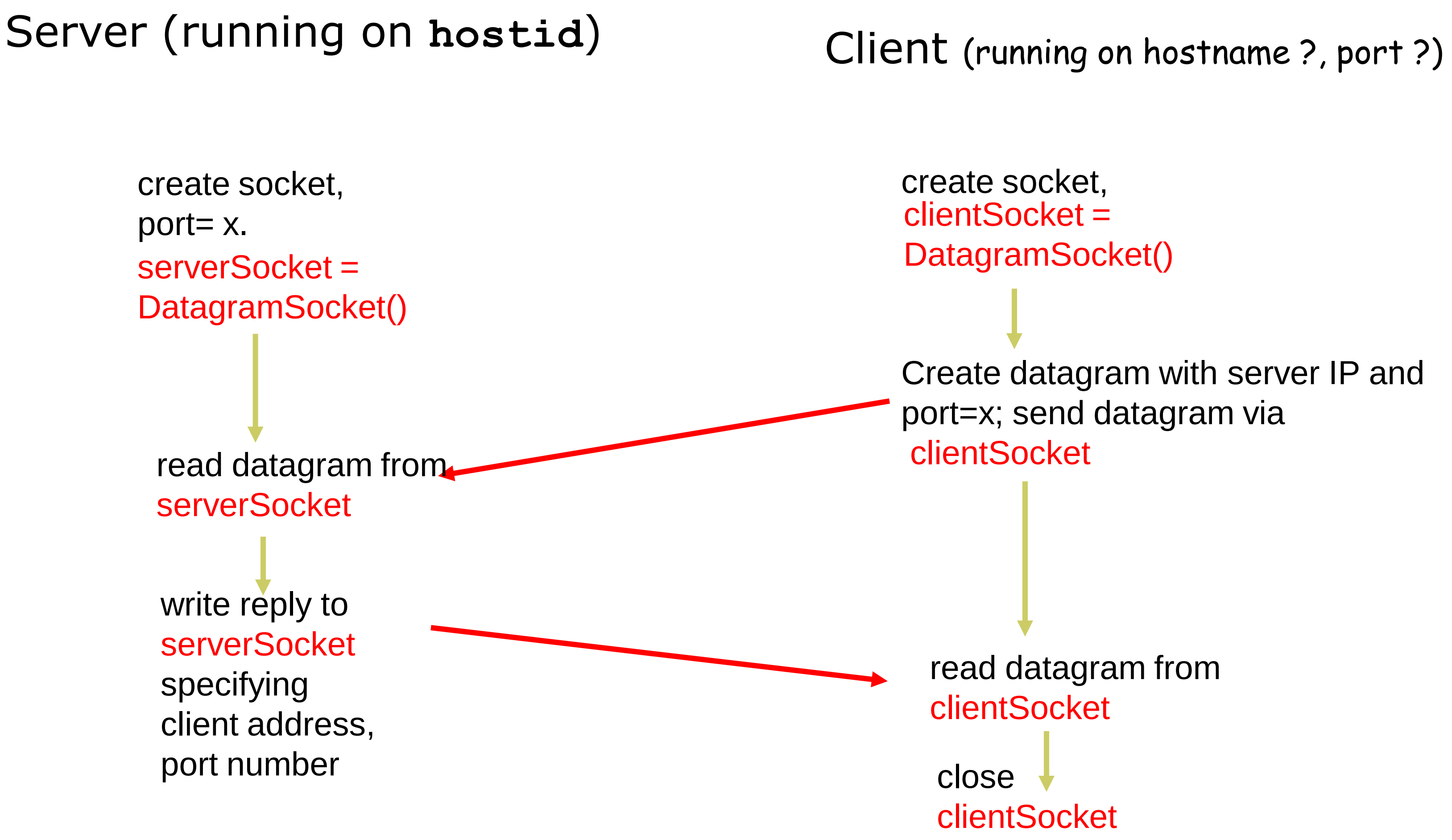
```
        outToServer.writeBytes(sentence + '\n');
```

Read line from server

```
        modifiedSentence = inFromServer.readLine();

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

        clientSocket.close();

    }
}
```

From Computer Networking by Kurose and Ross.

Client/server socket interaction: UDP



Server (running on **hostid**)

create socket,
port= x.
serverSocket =
DatagramSocket()

read datagram from
serverSocket

write reply to
serverSocket
specifying
client address,
port number

Client (running on hostname ?, port ?)

create socket,
clientSocket =
DatagramSocket()

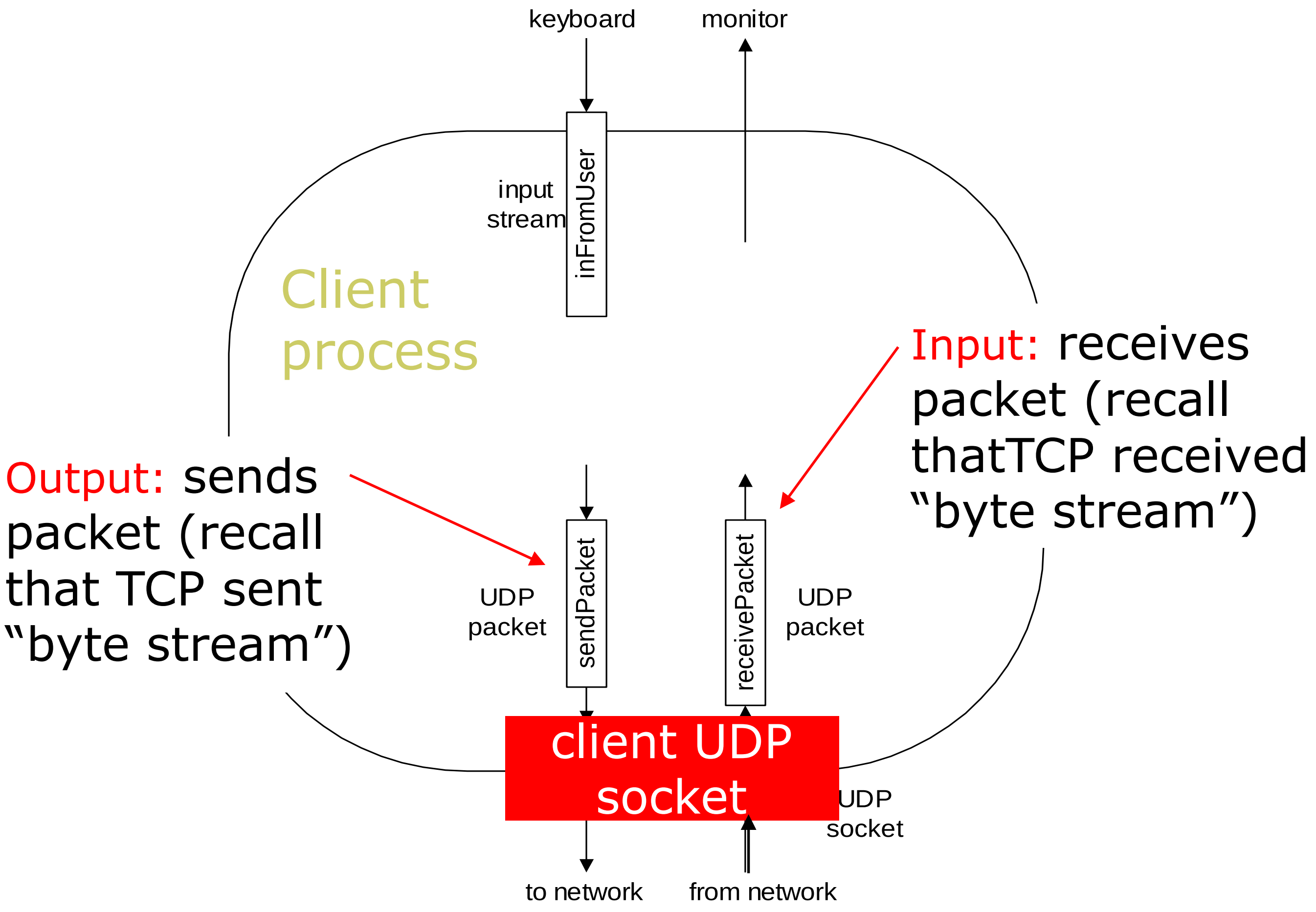
Create datagram with server IP and
port=x; send datagram via
clientSocket

read datagram from
clientSocket

close
clientSocket

From Computer Networking by Kurose and Ross.

Example: Java client (UDP)



Output: sends packet (recall that TCP sent "byte stream")

Input: receives packet (recall thatTCP received "byte stream")

From Computer Networking by Kurose and Ross.

Example: Java server (UDP)

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

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

Create datagram socket at port 9876

```
      DatagramSocket serverSocket = new DatagramSocket(9876);

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

      while(true)
        {
```

Create space for received datagram

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

Receive datagram

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

From Computer Networking by Kurose and Ross.

Example: Java server (UDP), cont

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

Get IP addr port #, of sender

```
     InetAddress IPAddress = receivePacket.getAddress();

     int port = receivePacket.getPort();

           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

From Computer Networking by Kurose and Ross.

Example: Java client (UDP)

```
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(“localhost");

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

      String sentence = inFromUser.readLine();
      sendData = sentence.getBytes();
```

Create input stream

Create client socket

Translate hostname to IP address using DNS

From Computer Networking by Kurose and Ross.

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();
    }
}
```

From Computer Networking by Kurose and Ross.

Multi threaded

```
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()));

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

           clientSentence = inFromClient.readLine();

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

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