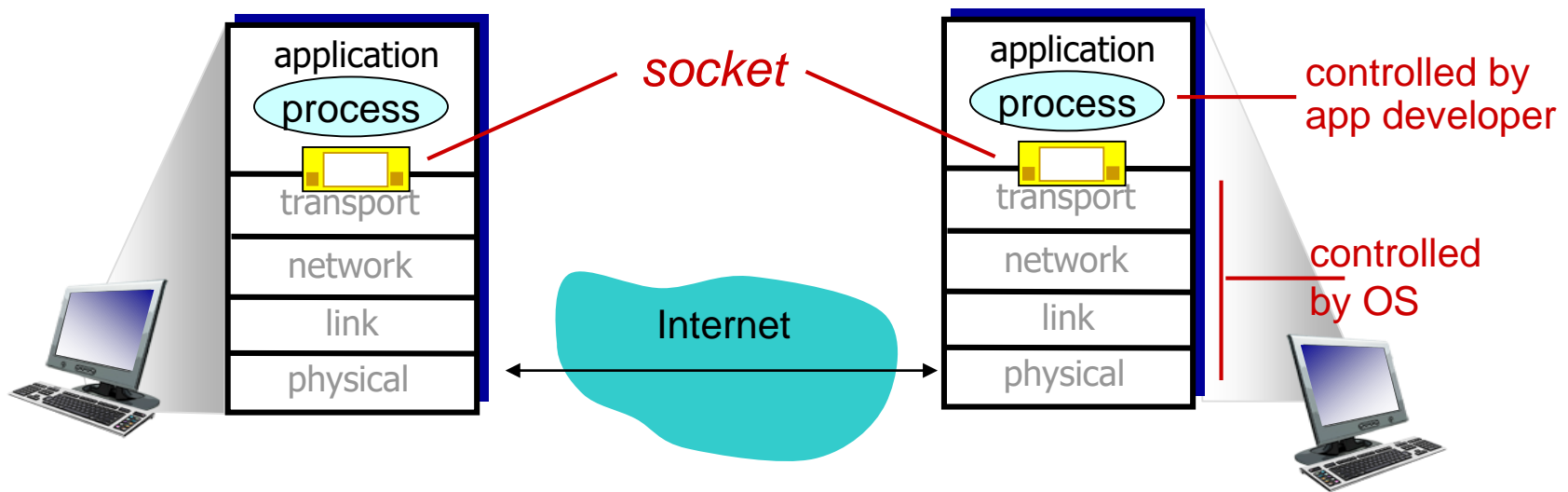


# Socket programming

**goal:** learn how to build client/server applications that communicate using sockets

**socket:** door between application process and end-end-transport protocol



# Socket programming

*Two socket types for two transport services:*

- **UDP:** unreliable datagram
- **TCP:** reliable, byte stream-oriented

*Application Example:*

1. client reads a line of characters (data) from its keyboard and sends data to server
2. server receives the data and converts characters to uppercase
3. server sends modified data to client
4. client receives modified data and displays line on its screen

# Socket programming *with* UDP

UDP: no “connection” between client & server

- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

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

Application viewpoint:

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

# Client/server socket interaction: UDP

## server (running on *serverIP*)

create socket, port= x:  
`serverSocket =  
socket(AF_INET,SOCK_DGRAM)`

↓  
read datagram from  
`serverSocket`

↓  
write reply to  
`serverSocket`  
specifying  
client address,  
port number

## client

create socket:  
`clientSocket =  
socket(AF_INET,SOCK_DGRAM)`

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

↓  
read datagram from  
`clientSocket`

↓  
close  
`clientSocket`

# Example app: UDP client

## *Python UDPClient*

include Python's socket library

```
from socket import *  
serverName = 'hostname'  
serverPort = 12000
```

create UDP socket for server

```
clientSocket = socket(AF_INET,  
                      SOCK_DGRAM)
```

get user keyboard input

```
message = input('Input lowercase sentence:')
```

Attach server name, port to message; send into socket

```
clientSocket.sendto(message.encode(),  
                    (serverName, serverPort))
```

read reply characters from socket into string

```
modifiedMessage, serverAddress =  
clientSocket.recvfrom(2048)
```

print out received string and close socket

```
print modifiedMessage.decode()  
clientSocket.close()
```

# Example app: UDP server

## *Python UDPServer*

```
from socket import *
serverPort = 12000

create UDP socket → serverSocket = socket(AF_INET, SOCK_DGRAM)
bind socket to local port number 12000 → serverSocket.bind(("", serverPort))
print ("The server is ready to receive")

loop forever → while True:
    Read from UDP socket into message, getting client's address (client IP and port) → message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    send upper case string back to this client → serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

# 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 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 that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

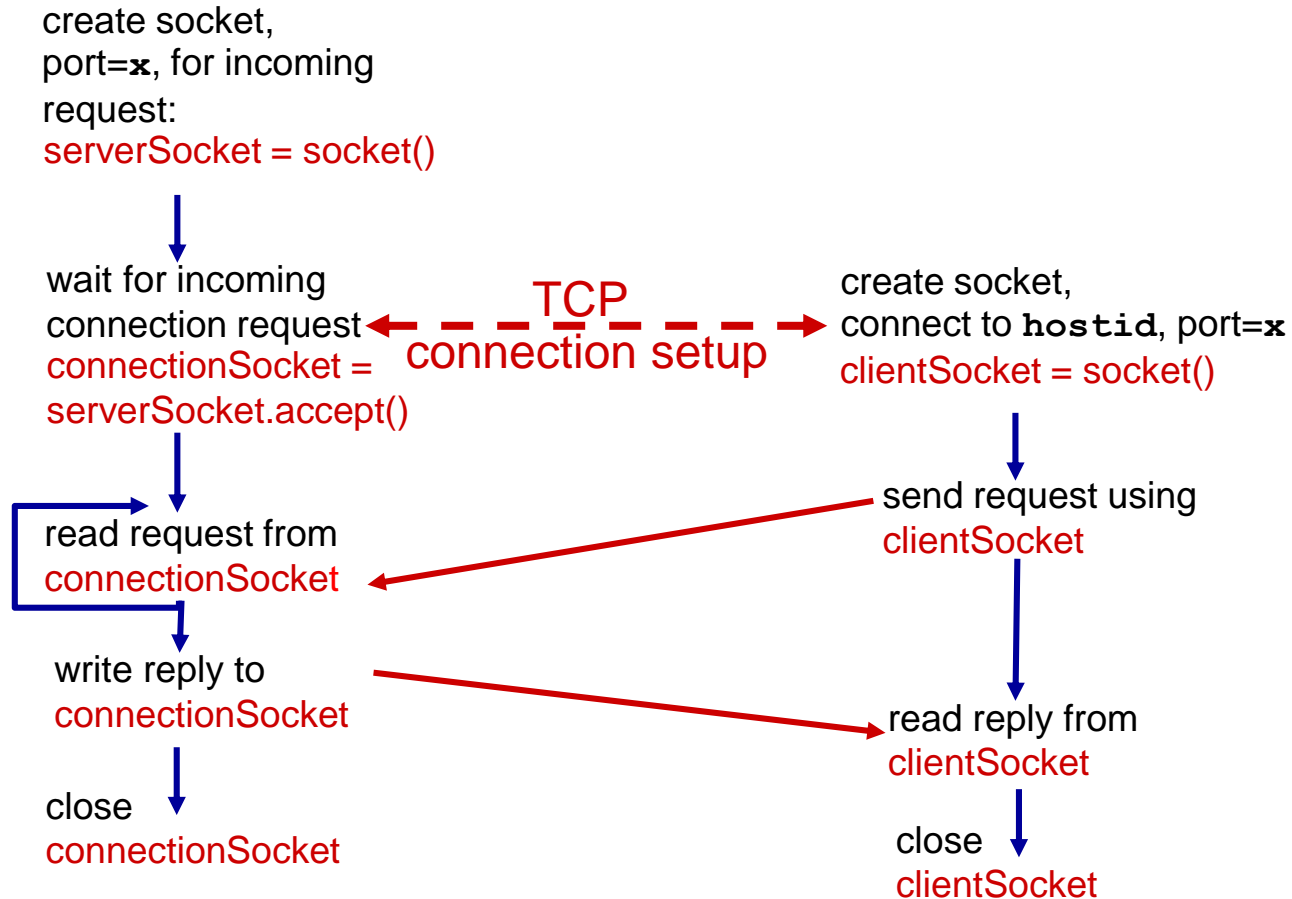
## application viewpoint:

TCP provides reliable, in-order byte-stream transfer (“pipe”) between client and server

# Client/server socket interaction: TCP

server (running on `hostid`)

client





# Example app: TCP client

## *Python TCPClient*

```
from socket import *
```

```
serverName = 'servername'
```

```
serverPort = 12000
```

create TCP socket for  
server, remote port 12000

```
clientSocket = socket(AF_INET, SOCK_STREAM)
```

```
clientSocket.connect((serverName, serverPort))
```

```
sentence = input('Input lowercase sentence:')
```

No need to attach server  
name, port

```
clientSocket.send(sentence.encode())
```

```
modifiedSentence = clientSocket.recv(1024)
```

```
print ('From Server:', modifiedSentence.decode())
```

```
clientSocket.close()
```

# Example app: TCP server

## *Python TCPServer*

create TCP welcoming  
socket



server begins listening for  
incoming TCP requests



loop forever



server waits on accept()  
for incoming requests, new  
socket created on return



read bytes from socket (but  
not address as in UDP)



close connection to this  
client (but *not* welcoming  
socket)



```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while True:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.
                           encode())
    connectionSocket.close()
```