College of Information Studies

University of Maryland Hornbake Library Building College Park, MD 20742-4345

# Networked System Architectures

Session 28

INST 346

Technologies, Infrastructure and Architecture

# Goals for Today

- Internet Architectures

- Building an Internet app

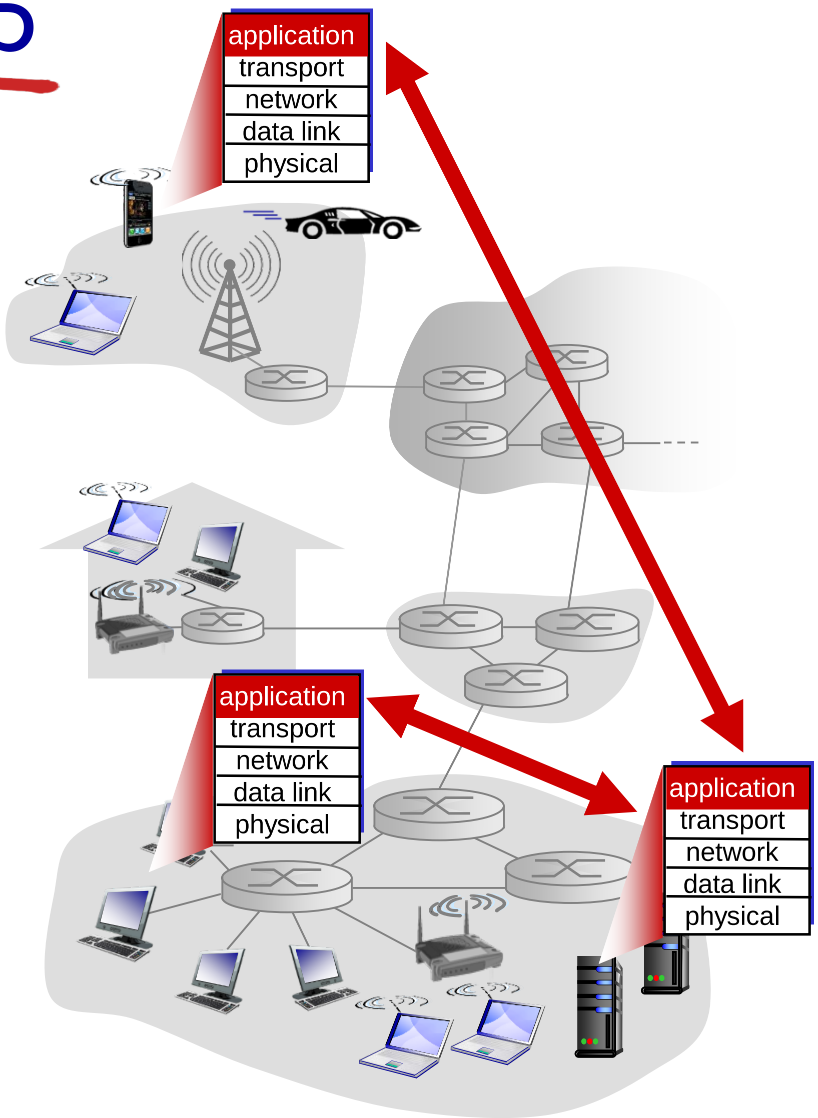
# Creating a network app

write programs that:

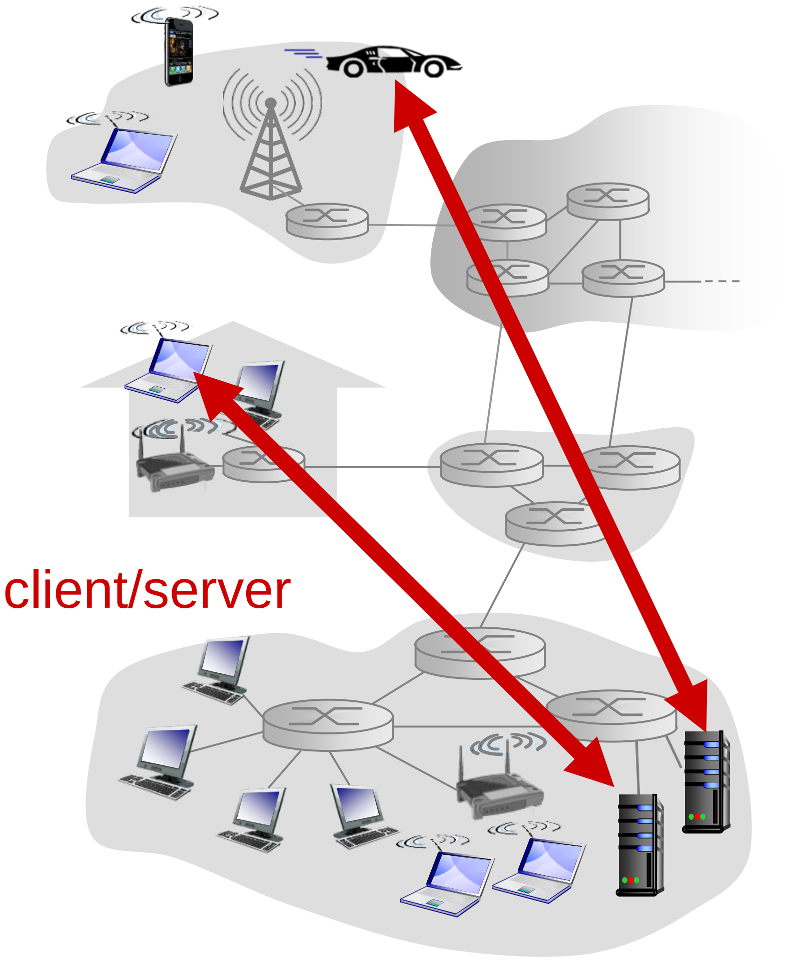
- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software

no need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation



# Client-server architecture (e.g., Web)



server:

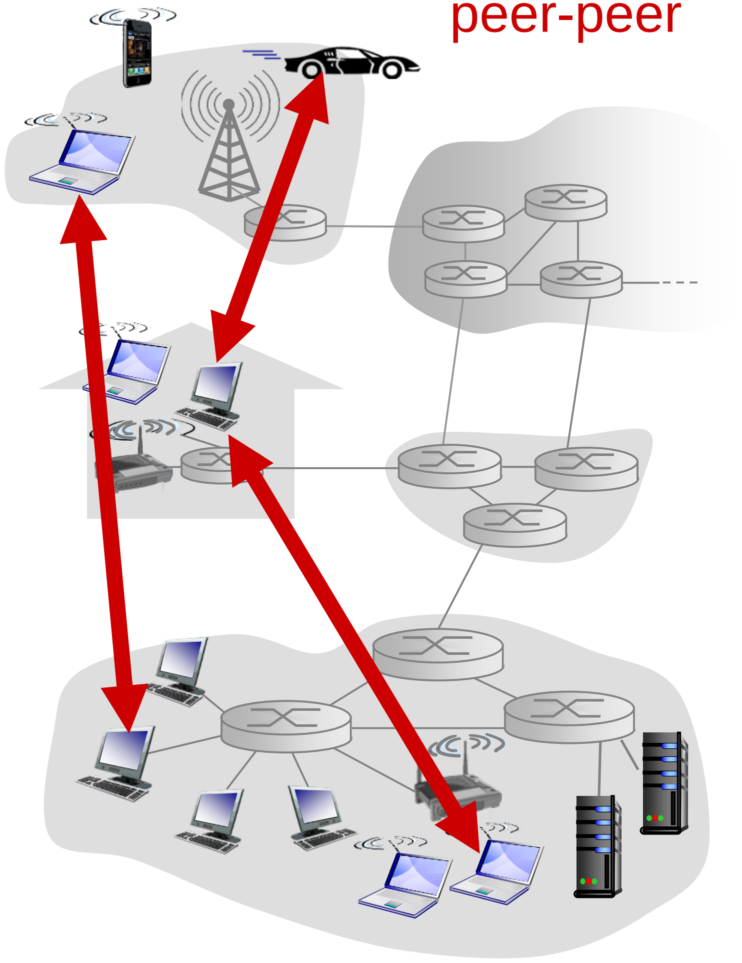
- always-on host
- permanent IP address
- data centers for scaling

clients:

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

# P2P architecture (e.g., Skype)

- *no* central server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management



# App-layer protocol must define:

- types of messages
  - e.g., request, response
- message syntax
  - what fields in messages
  - how fields are delineated
- message semantics
  - meaning of information in fields
- rules for when and how processes send & respond to messages

"open" protocols:

- e.g., HTTP, SMTP
- defined in "Requests for Comment" (RFC's)
- designed for interoperability

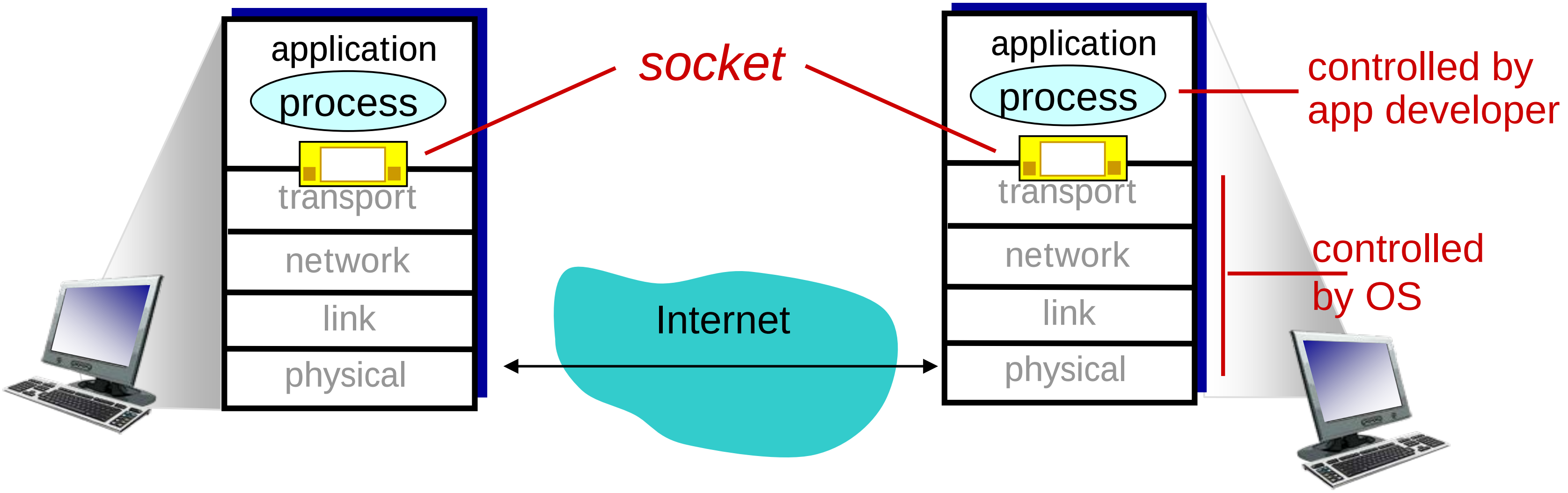
proprietary protocols:

- e.g., Skype

# Socket programming

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

*socket:* outbox/inbox between application process and end-end-transport protocol



# Socket programming

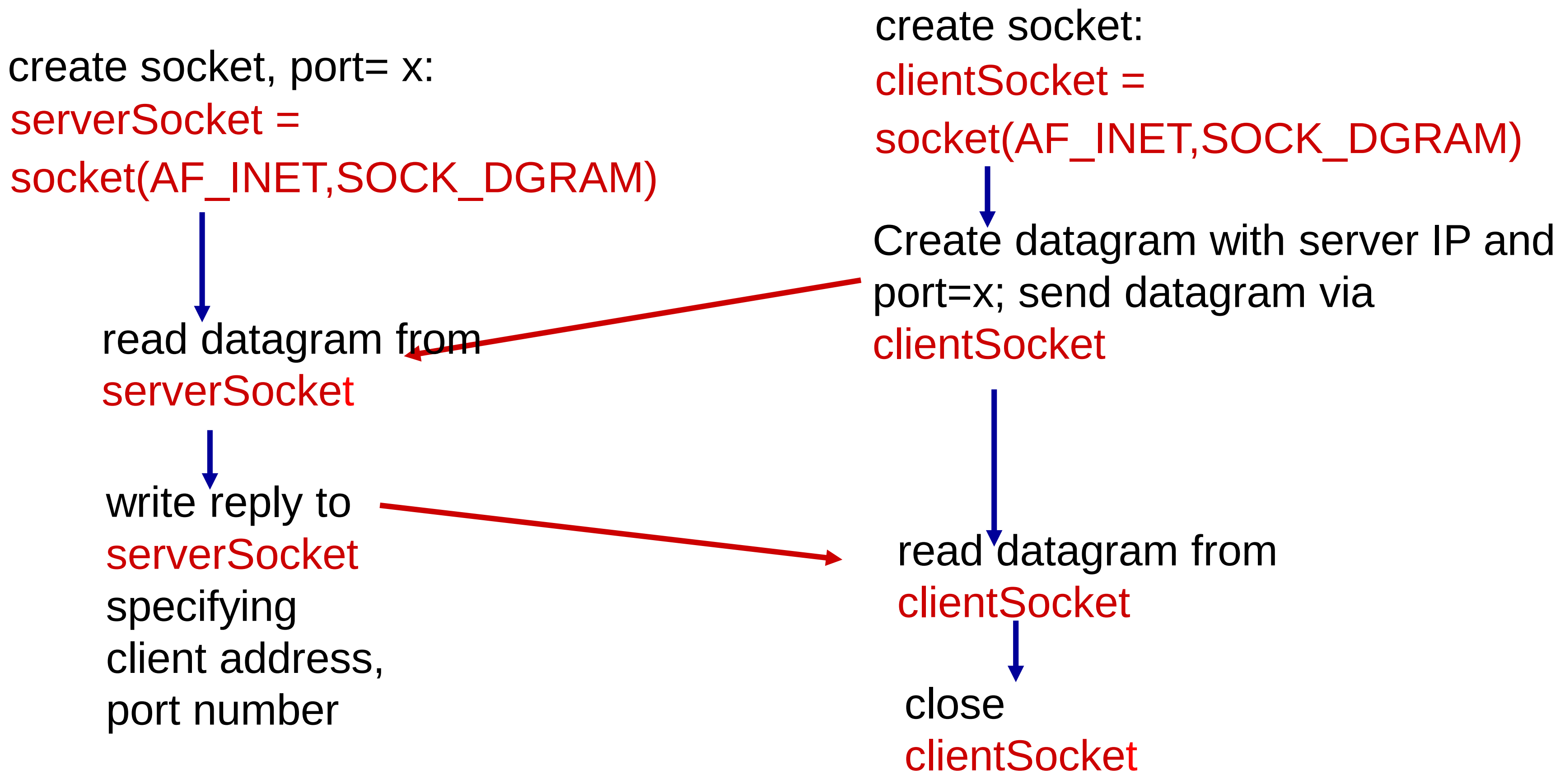
*Two socket types for two transport services:*

- *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

# 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 = ‘localhost’
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*

create UDP socket

bind socket to local port number 12000

loop forever

Read from UDP socket into message, getting client’s address (client IP and port)

send upper case string back to this client

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print (“The server is ready to receive”)
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(),
                                    clientAddress)
```

# Running Python

- Install the latest Python 3 from:
  - https://www.python.org/downloads/
- Download the programs
  - Materials used in class link from schedule
- Open two shell windows
  - On a PC, type "cmd" in the search box
  - On a Mac, open a terminal
- In one shell, type:
  - python udpserver.py
- In the other, type:
  - python udpclient.py

# Socket programming *with TCP*

client must contact server

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

client contacts server by:

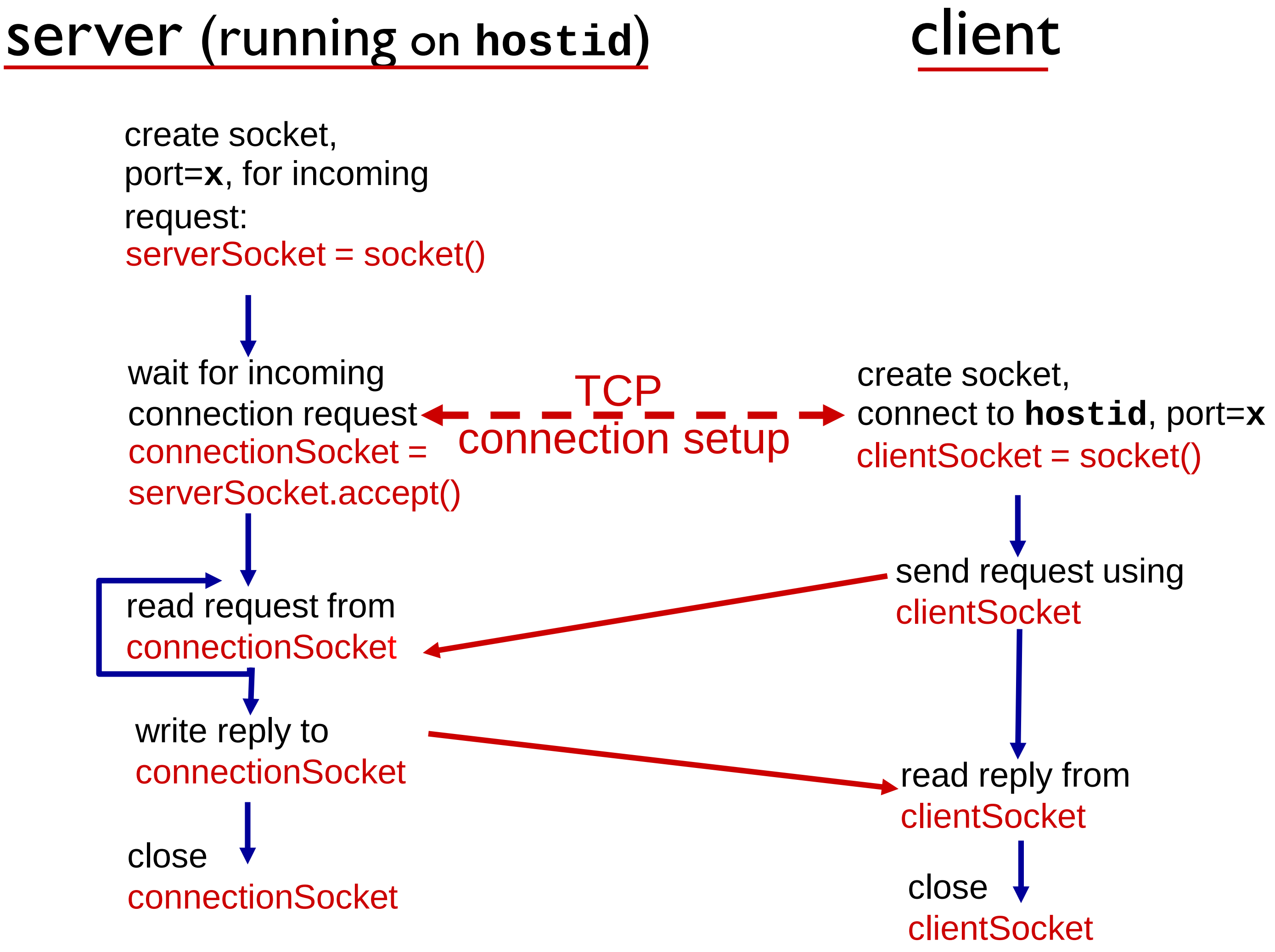
- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- 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**

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

wait for incoming
connection request
connectionSocket =
serverSocket.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

# Example app: TCP client

*Python TCPClient*

```
from socket import *
serverName = ’localhost’
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = input(‘Input lowercase sentence:’)
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print (‘From Server:’, modifiedSentence.decode())
clientSocket.close()
```

create TCP socket for server, remote port 12000

No need to attach server name, port

# 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()
```