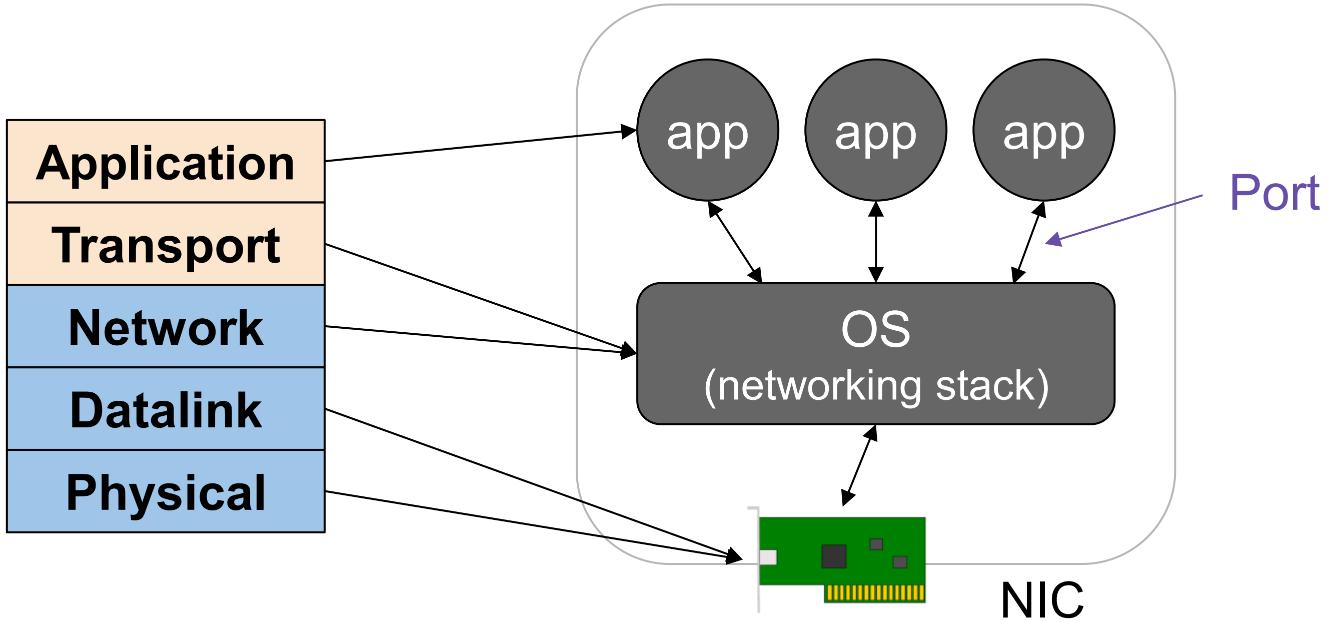
Lab1 Background: Socket Programming with UDP and TCP

Sockets

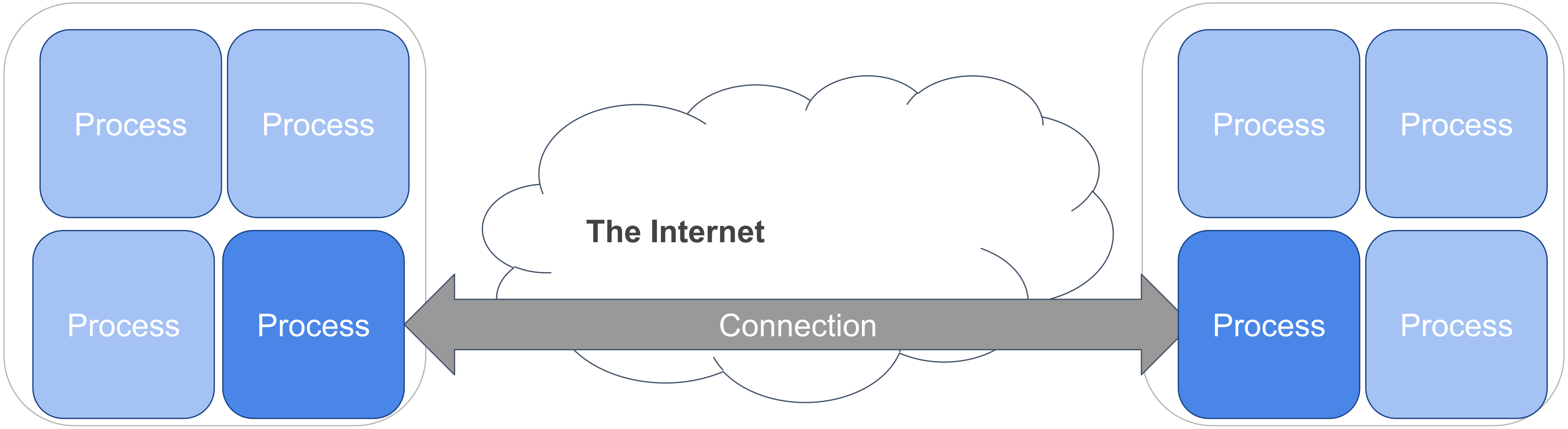
- Endpoint for sending or receiving data across a network
- OS abstraction for **connections**
- Allow L7 applications to operate on data streams (not packets)
  - Connect, listen, accept, send, receive
- Open a socket between:
  - Source IP address : *port*
  - Destination IP address : *port*



(Network Interface Card)

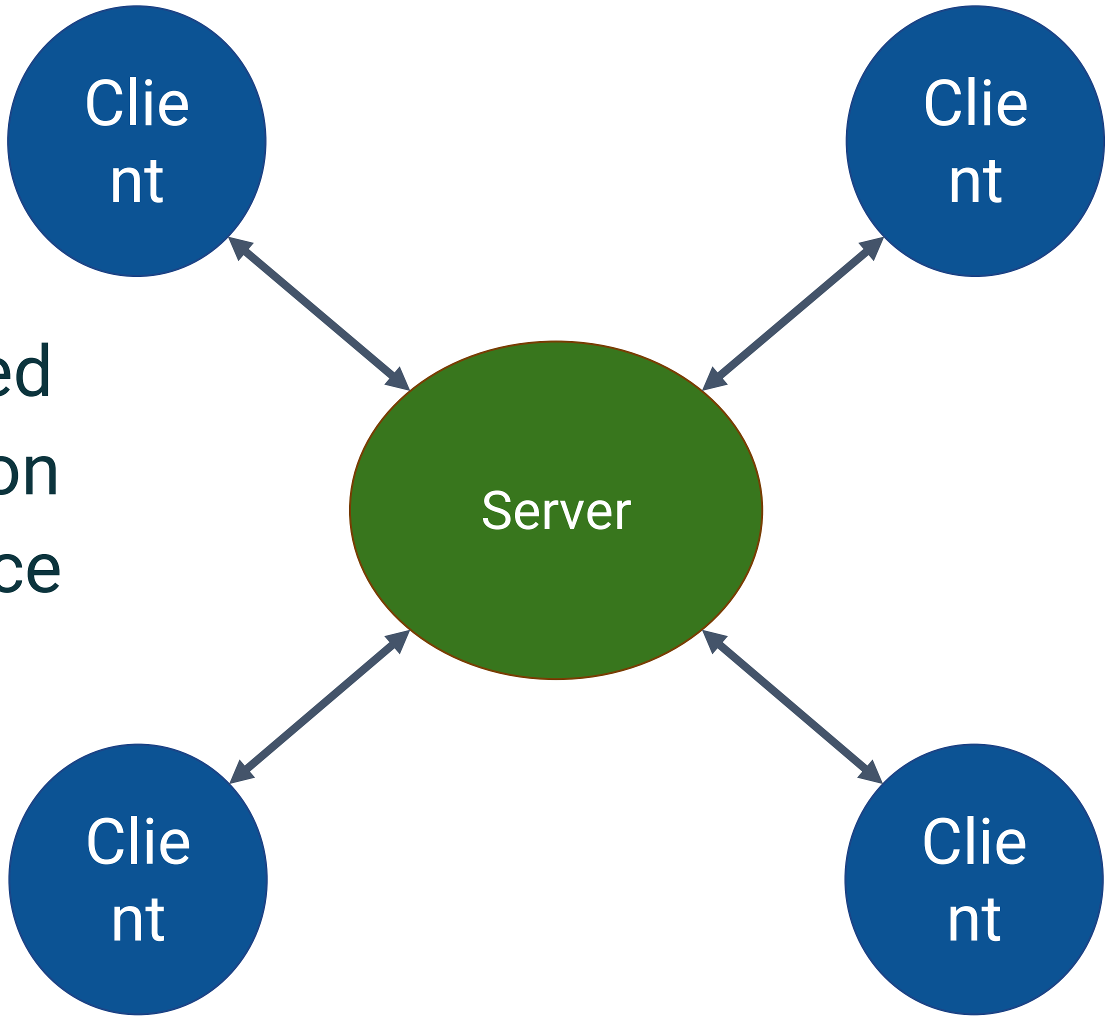
Connection (the basic abstraction)

- Pipes data between two processes (on different hosts)
- Data flows both ways
- Data is sent as a stream of bits
- Reconstruction of bits only at the endpoints



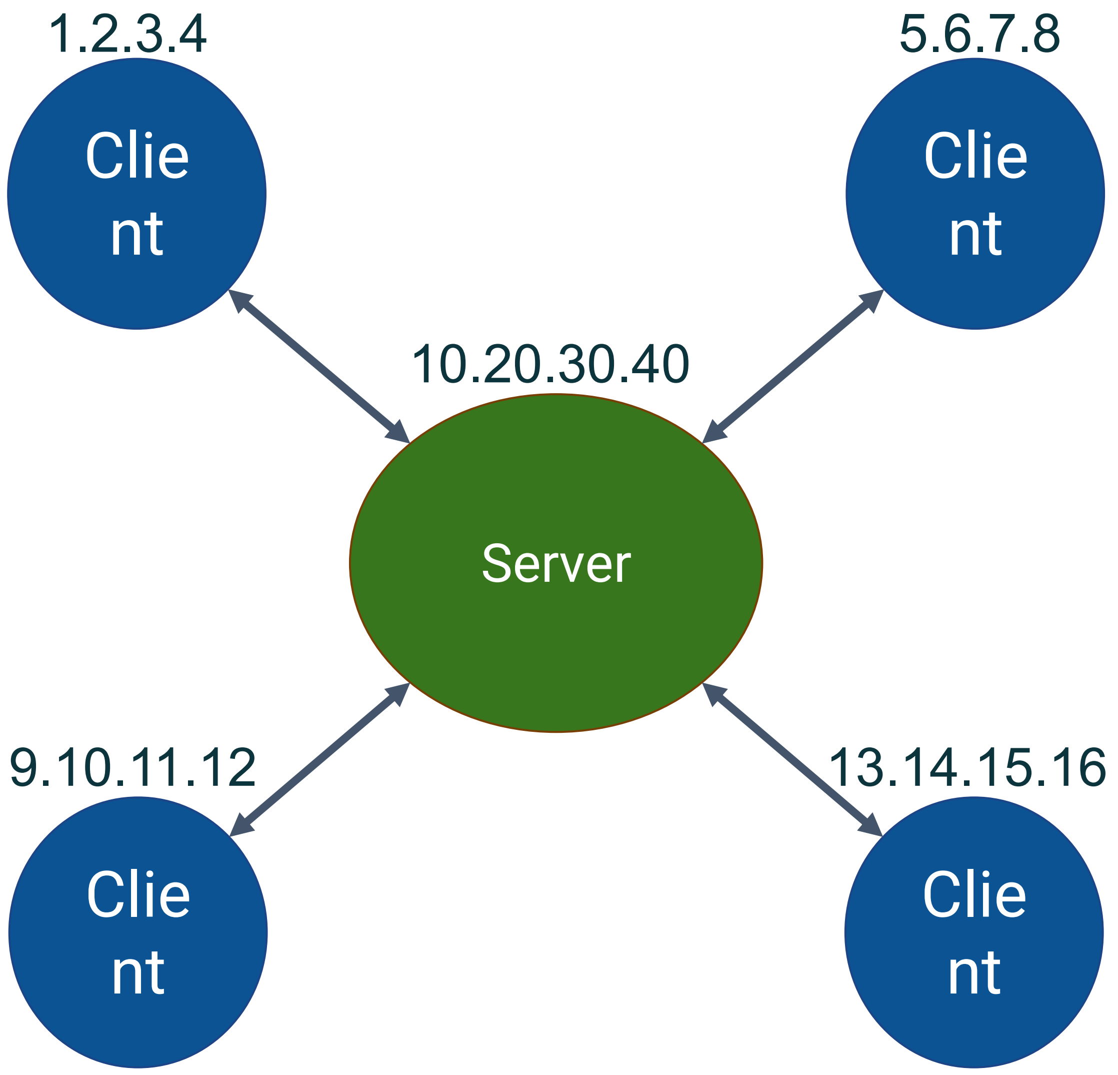
Connections

- Two types of sockets
  - **Server** and **Client**
- Servers *listen* for clients to connect to them
  - Wait until a connection is attempted
    - Accept and dispatch connection
  - Usually serving many clients at once
- Clients *initiate* new connections to servers
- Example
  - Server: www.hofstra.edu
  - Client: Your internet browser



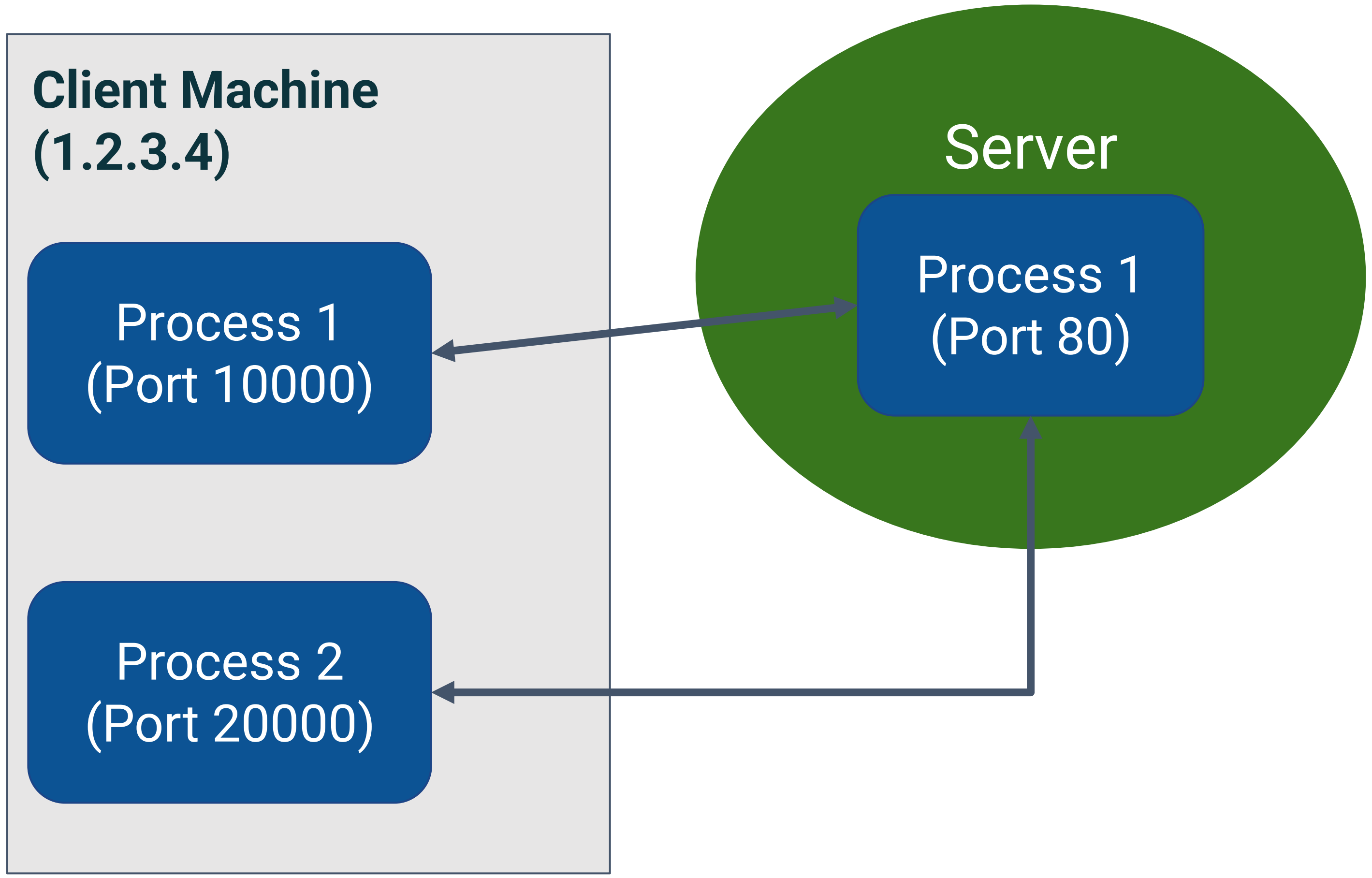
Connections

- Hosts have addresses
  - Unique identifier (just like a street address)
- Clients (different users) find servers with their addresses
  - Servers send data back with the client address
- IP addresses are not enough
  - Also need ports



Ports

- Sockets are identified by unique IP:port pairs
- A port is a number associated with a socket when it is created
  - i.e. sending to address "1.2.3.4:10000" would send data to the socket owned by Process 1
- Each server listens on a well-known port
  - Which one depends on application
  - HTTP: 80
  - SSH: 22
- Client also has a port
  - Port number can be any (large) number

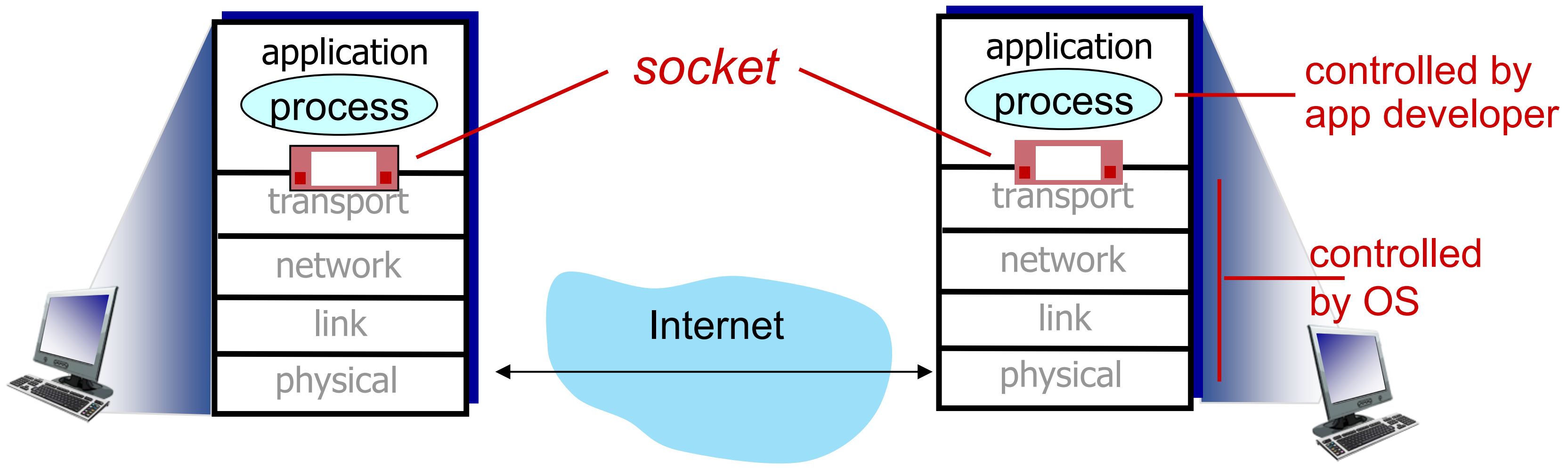


IMPORTANT

Socket programming

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

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



IMPORTANT

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

IMPORTANT

Socket programming with UDP

UDP: no "connection" between client and 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

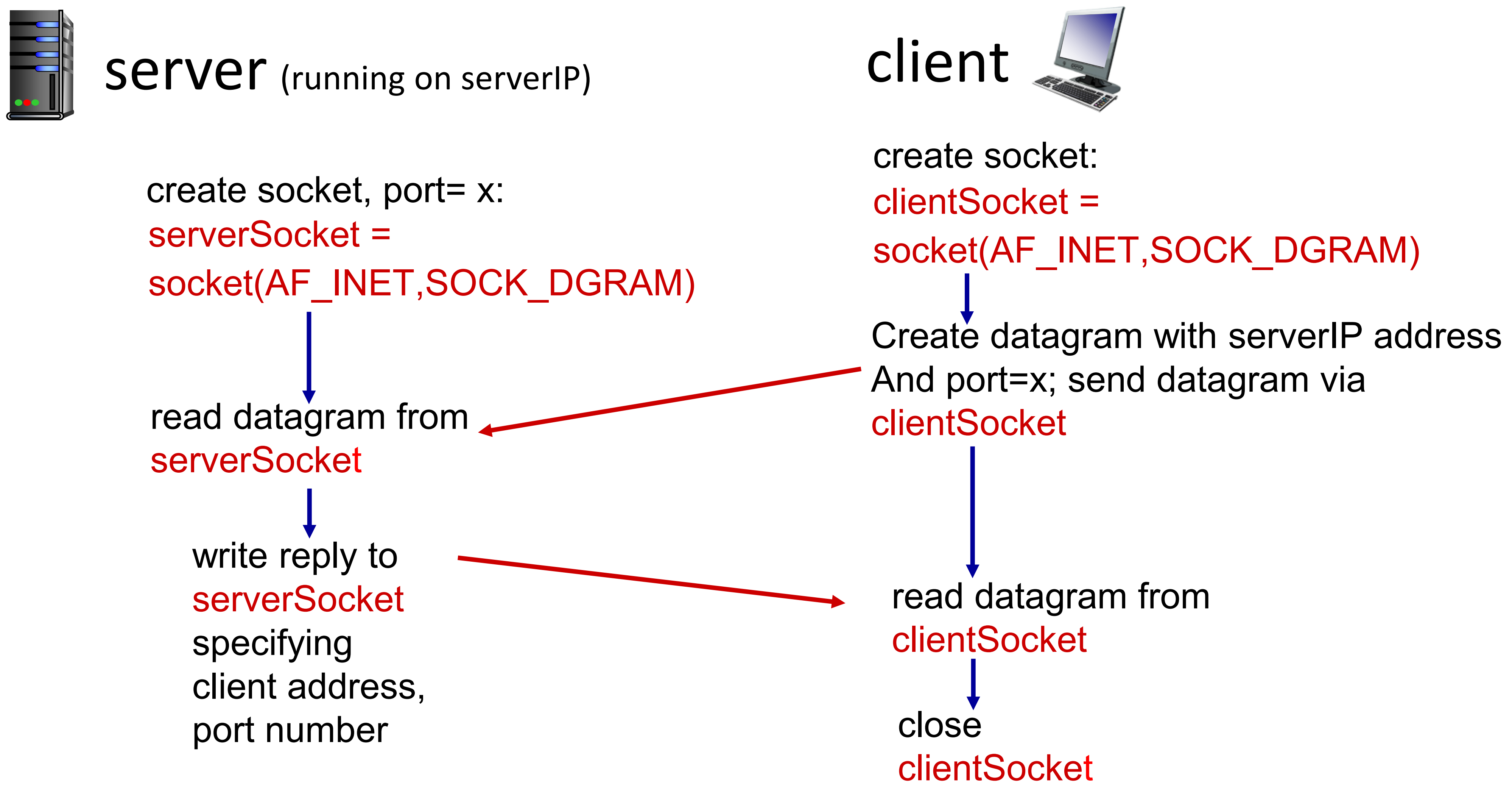
UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

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

IMPORTANT

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 serverIP address
And port=x; send datagram via
clientSocket

read datagram from
clientSocket

close
clientSocket

IMPORTANT

Example app: UDP client

Python UDPClient

include Python’s socket library →
```
from socket import *
serverName = 'hostname'
serverPort = 12000
```
create UDP socket →
```
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 data (bytes) from socket →
```
modifiedMessage, serverAddress =
                      clientSocket.recvfrom(2048)
```
print out received string and close socket →
```
print(modifiedMessage.decode())
clientSocket.close()
```

Note: this code update (2023) to Python 3

IMPORTANT

Example app: UDP server

Python UDPServer

```
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)
```

create UDP socket → serverSocket = socket(AF_INET, SOCK_DGRAM)

bind socket to local port number 12000 → serverSocket.bind(('', serverPort))

loop forever → while True:

Read from UDP socket into message, getting client's address (client IP and port) → message, clientAddress = serverSocket.recvfrom(2048)

send upper case string back to this client → serverSocket.sendto(modifiedMessage.encode(), clientAddress)

Note: this code update (2023) to Python 3

IMPORTANT

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

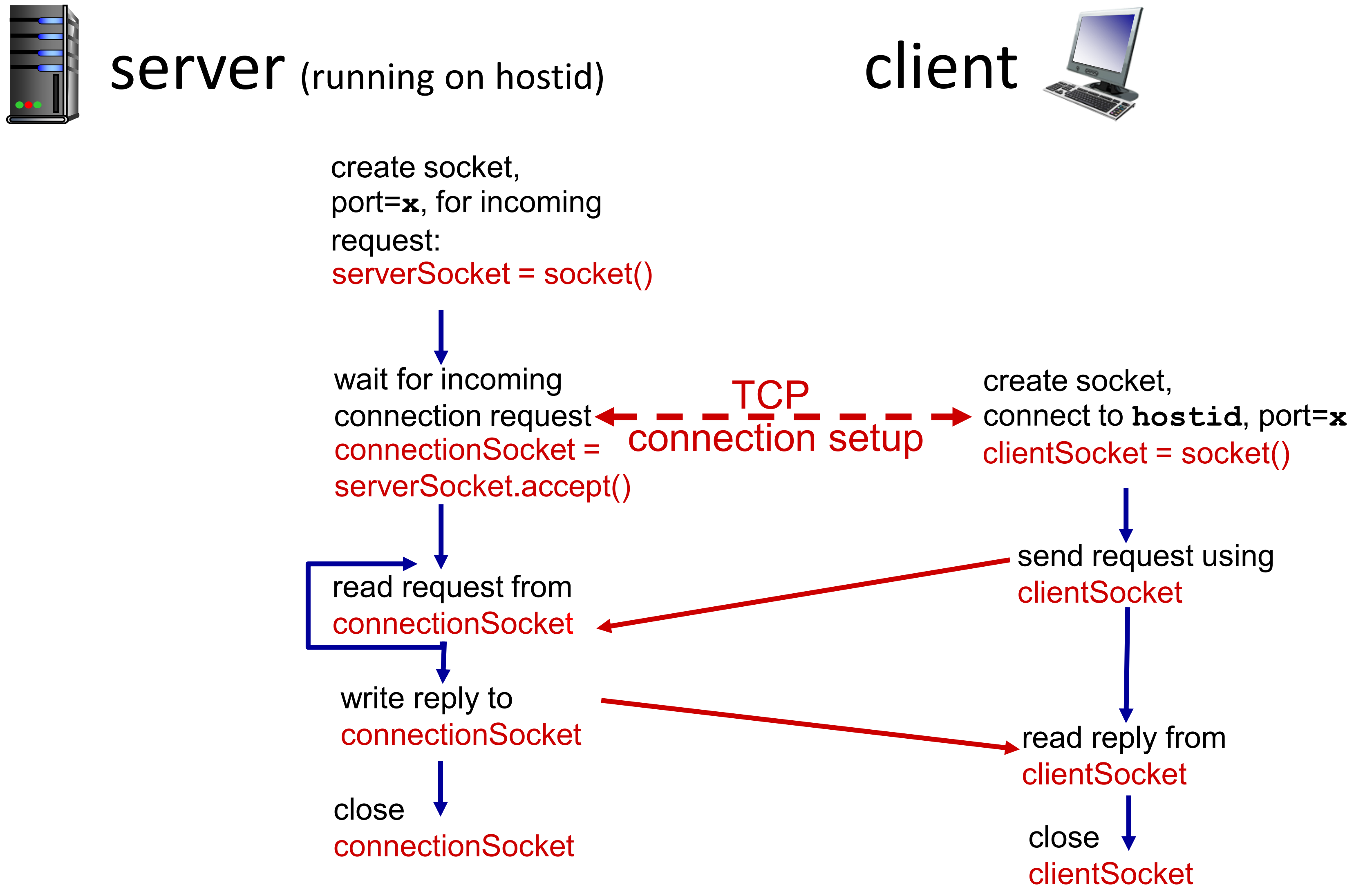
- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - client source port # and IP address used to distinguish clients (more in Chap 3)

Application viewpoint

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

IMPORTANT

Client/server socket interaction: TCP



IMPORTANT

Example app: TCP client

Python TCPClient

create TCP socket for server, remote port 12000

No need to attach server name, port

```
from socket import *
serverName = 'servername'
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()
```

Note: this code update (2023) to Python 3

IMPORTANT

Example app: TCP server

Python TCPServer

```
 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()
```

- create TCP welcoming socket → `serverSocket = socket(AF_INET,SOCK_STREAM)`
- server begins listening for incoming TCP requests → `serverSocket.listen(1)`
- loop forever → `while True:`
- server waits on accept() for incoming requests, new socket created on return → `connectionSocket, addr = serverSocket.accept()`
- read bytes from socket (but not address as in UDP) → `sentence = connectionSocket.recv(1024).decode()`
- close connection to this client (but *not* welcoming socket) → `connectionSocket.close()`

Note: this code update (2023) to Python 3