The echo-server

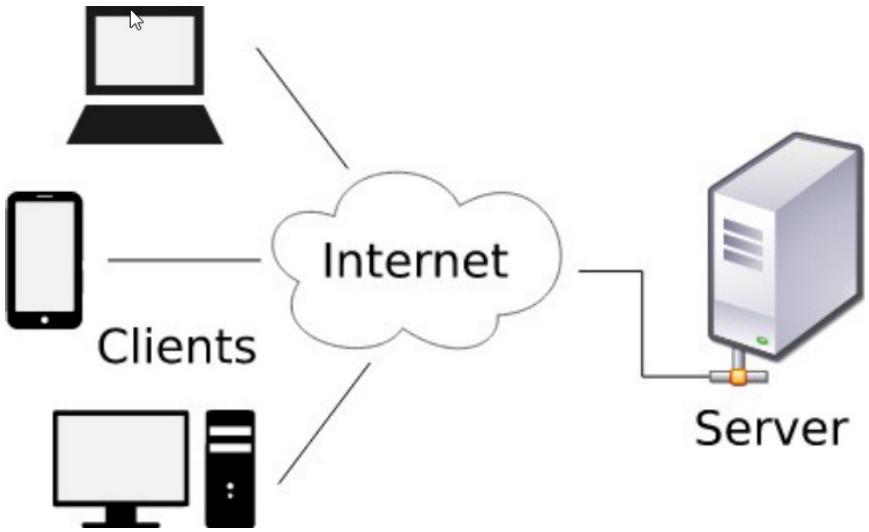
A guide in socket programming in Python

CSEE 4119 Shuyue Yu

Summary

- Socket programming in Python: echo-server
  - UDP and TCP, respectively
  - Example: The client reads a line from keyboard and sends to the server; the server receives message from client, makes it uppercase and sends it back to the client.
- Netcat

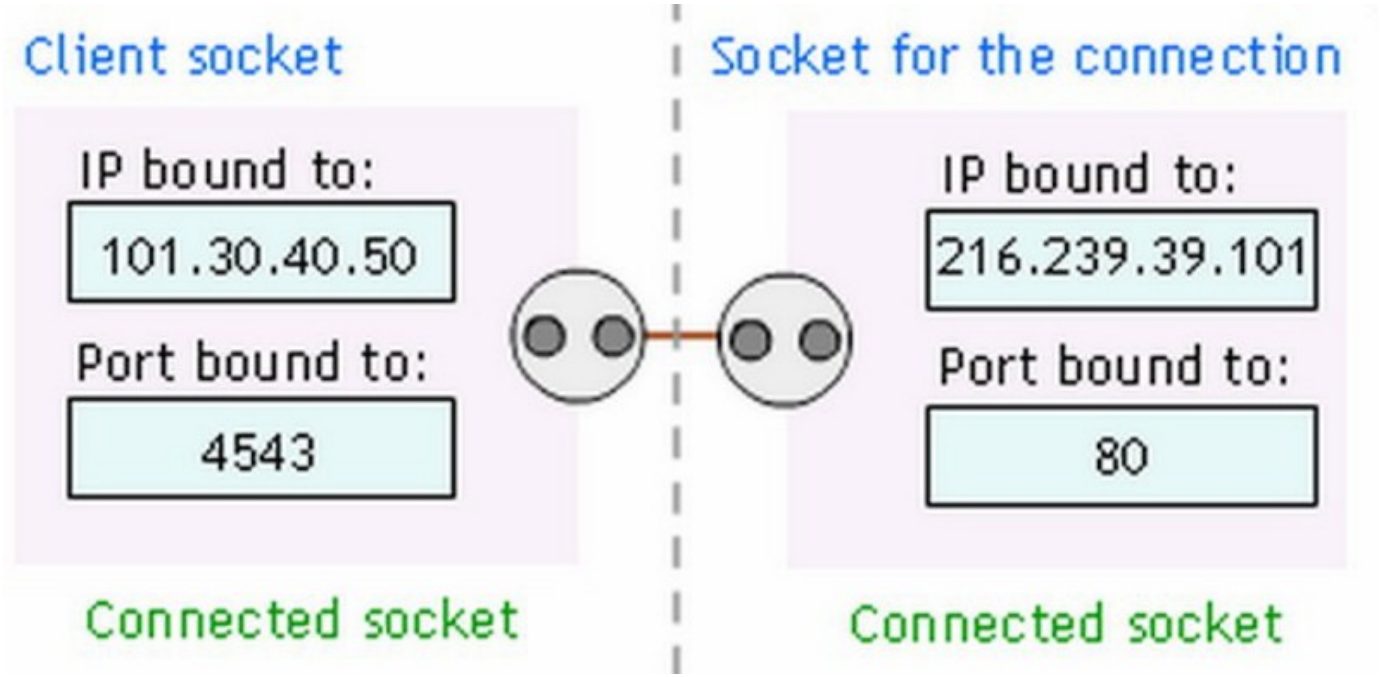
The client/server model



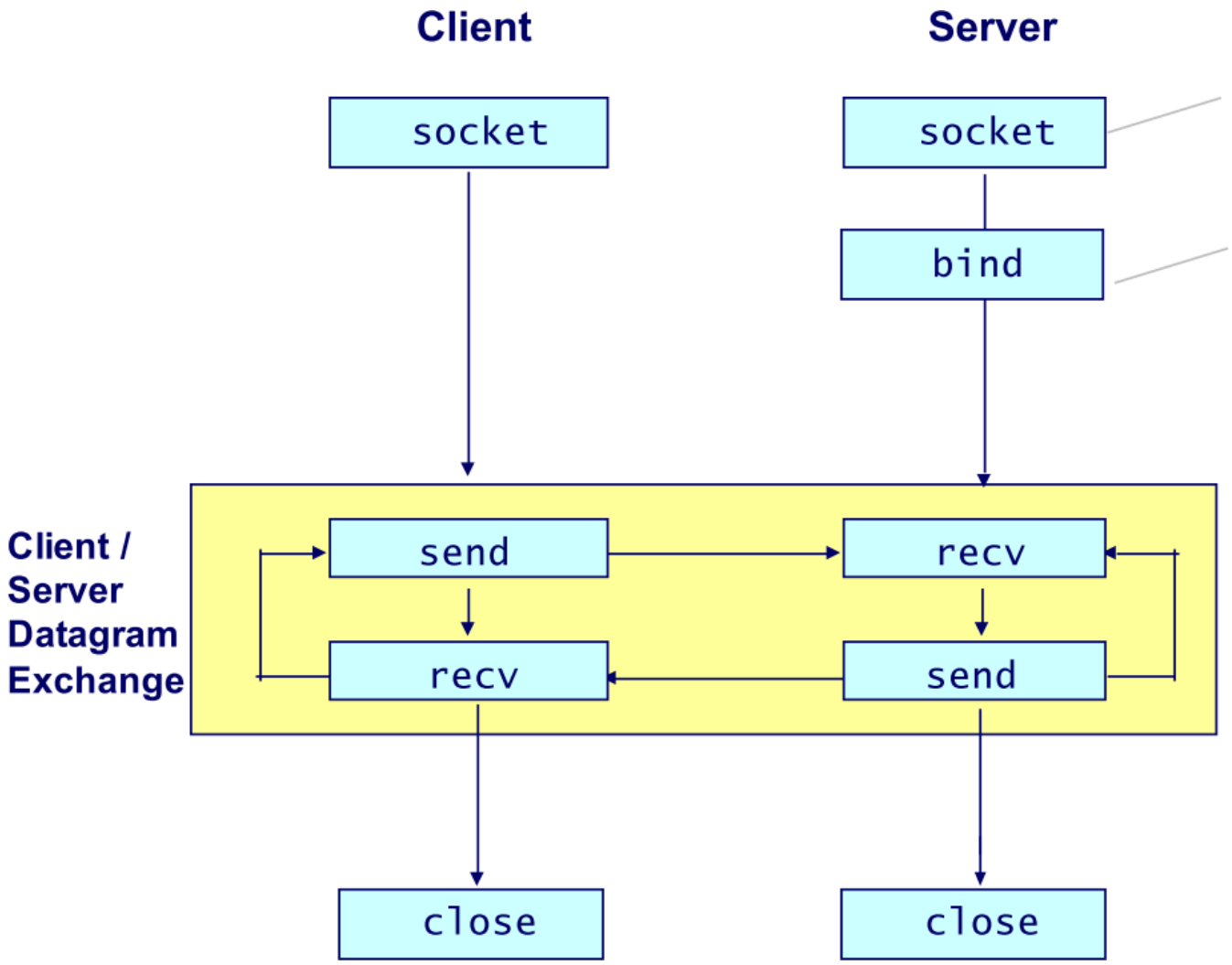
- Server waits for incoming requests over the network from clients
- e.g. Web browser/server

Socket

- A tool for inter-process communication for processes separated by a network, such as clients and servers.
- Analogy: process = house, socket = door.
- Each socket has an address: IP number and port.



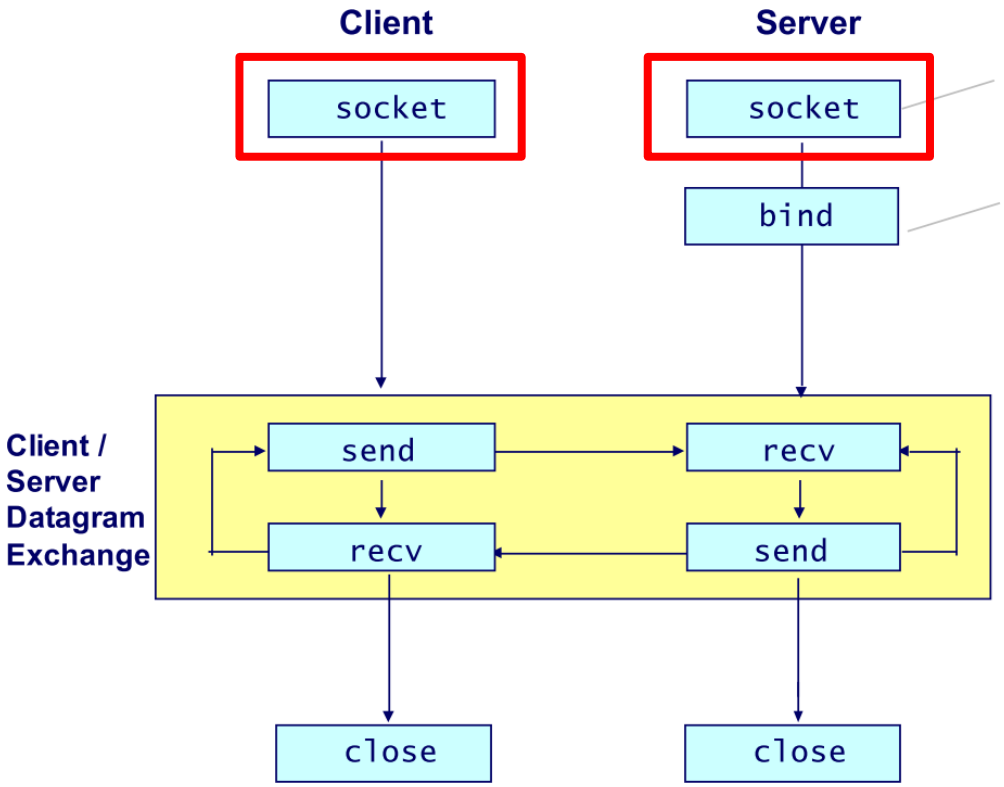
Client / Server Flow for UDP



https://roxanageambasu.github.io/ds-class/assets/lectures/lectu[5]re4.pdf

Create a UDP socket

- Called by both client and server
- Creates a socket used to send and receive UDP datagrams



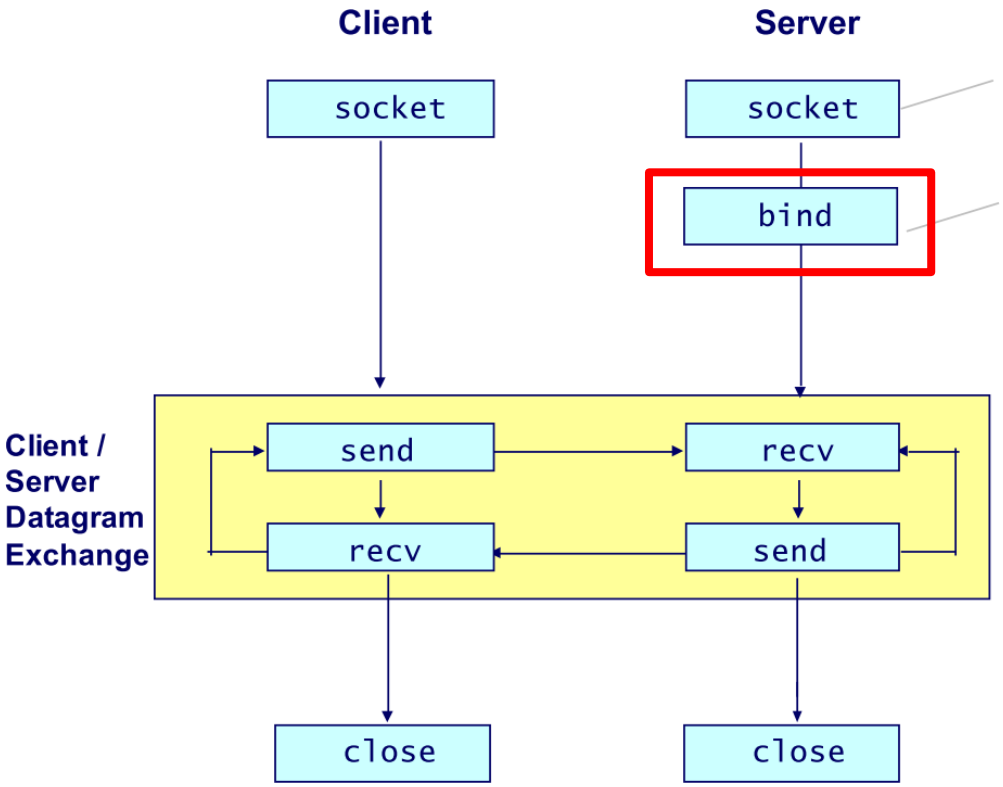
Create a UDP Socket: Code

```
from socket import *
serverSocket = socket(AF_INET, SOCK_DGRAM)
```

- AF_INET: underlying network is using IPv4
- SOCK_DGRAM: UDP socket

Server: Bind a UDP Socket

- Binds the listening socket to a specific address that should be known to the client



Bind a UDP Socket: Code

- "OS, redirect all the packets that come to 1.0.0.1 with destination port 8080 to my socket"

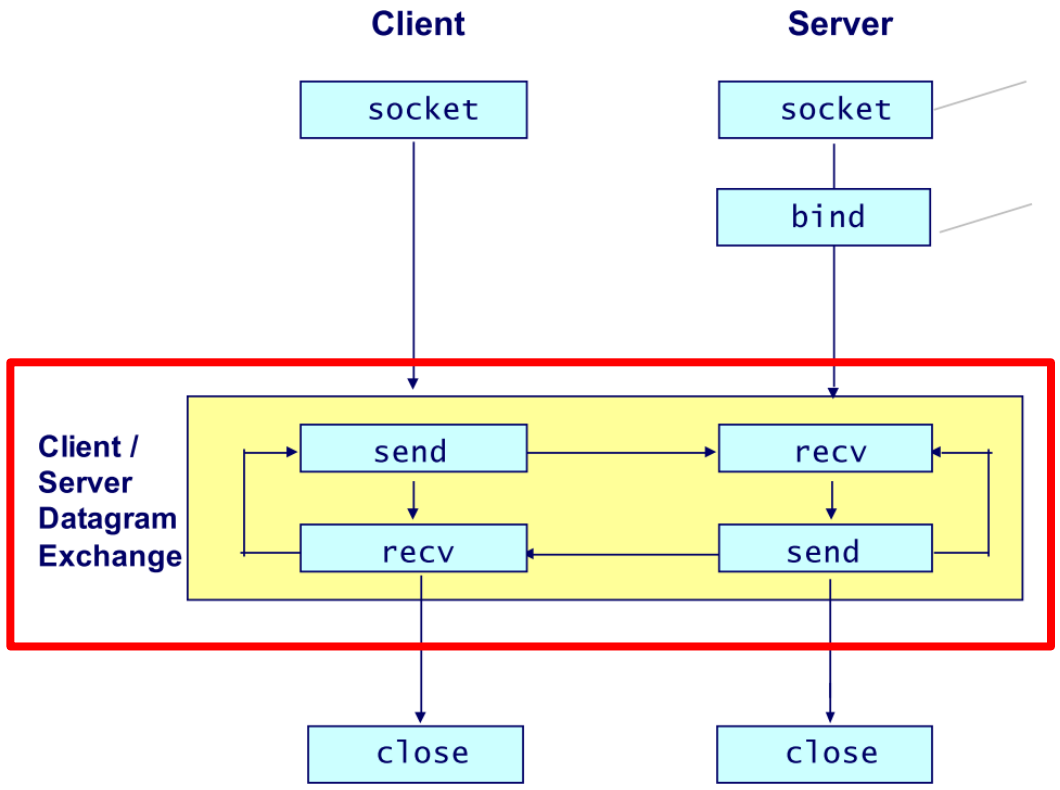
```
serverSocket.bind( (‘1.0.0.1’, 8080 ) )

# reachable by any address the machine has
# serverSocket.bind( ( ‘ ’, 8080) )

# OS will pick one port
# serverSocket.bind( (‘1.0.0.1’, 0) )
```

Send and Receive UDP Datagrams

- Called by both server and client
- Wraps and delivers a datagram from the server socket to the client socket (and vice-versa)



Send and Receive UDP Datagrams: Code

```
# happens in client
clientSocket.sendto(message, (serverName, serverPort))
serverMessage, serverAddress = clientSocket.recvfrom(2048)
```

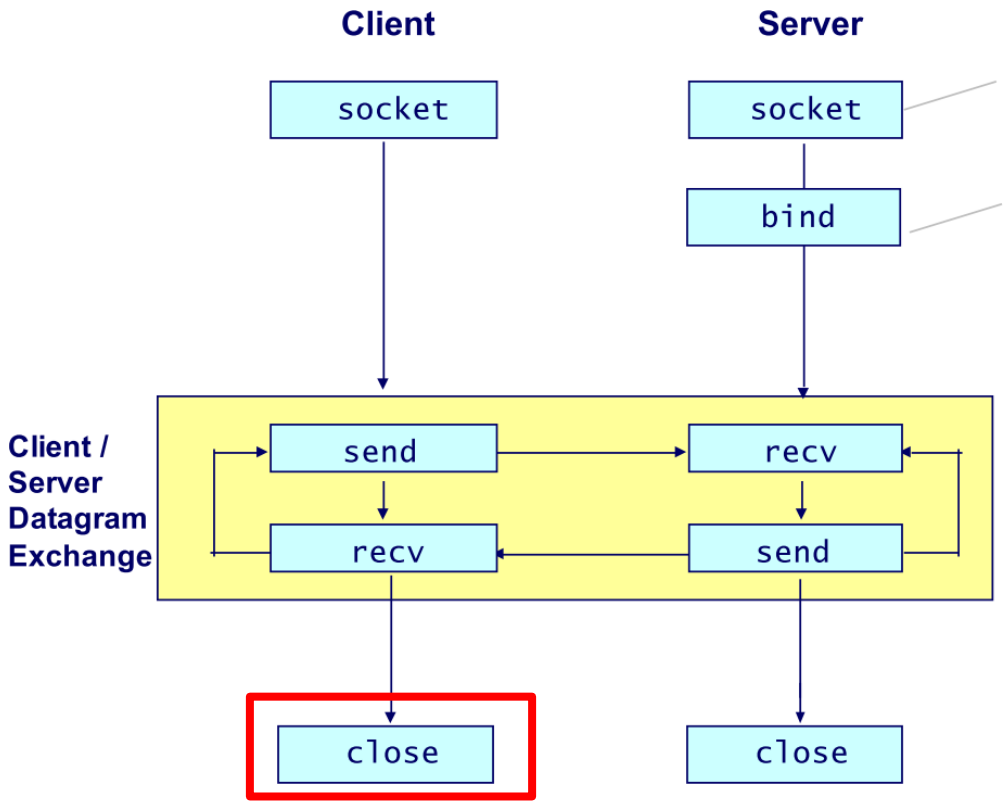
- sendto and recvfrom take bytes type input instead of string
  - e.g. b'hello'; use encode() and decode() to convert from and back to string
- 2048 is the buffer size
  - message larger than 2048 -> receive 2048
- recvfrom will block until receives a message

Send and Receive UDP Datagrams: Code

```
# happens in server
message, clientAddress = serverSocket.recvfrom(2048)
# modify message
serverSocket.sendto(modifiedMessage, clientAddress)
```

Close the Socket

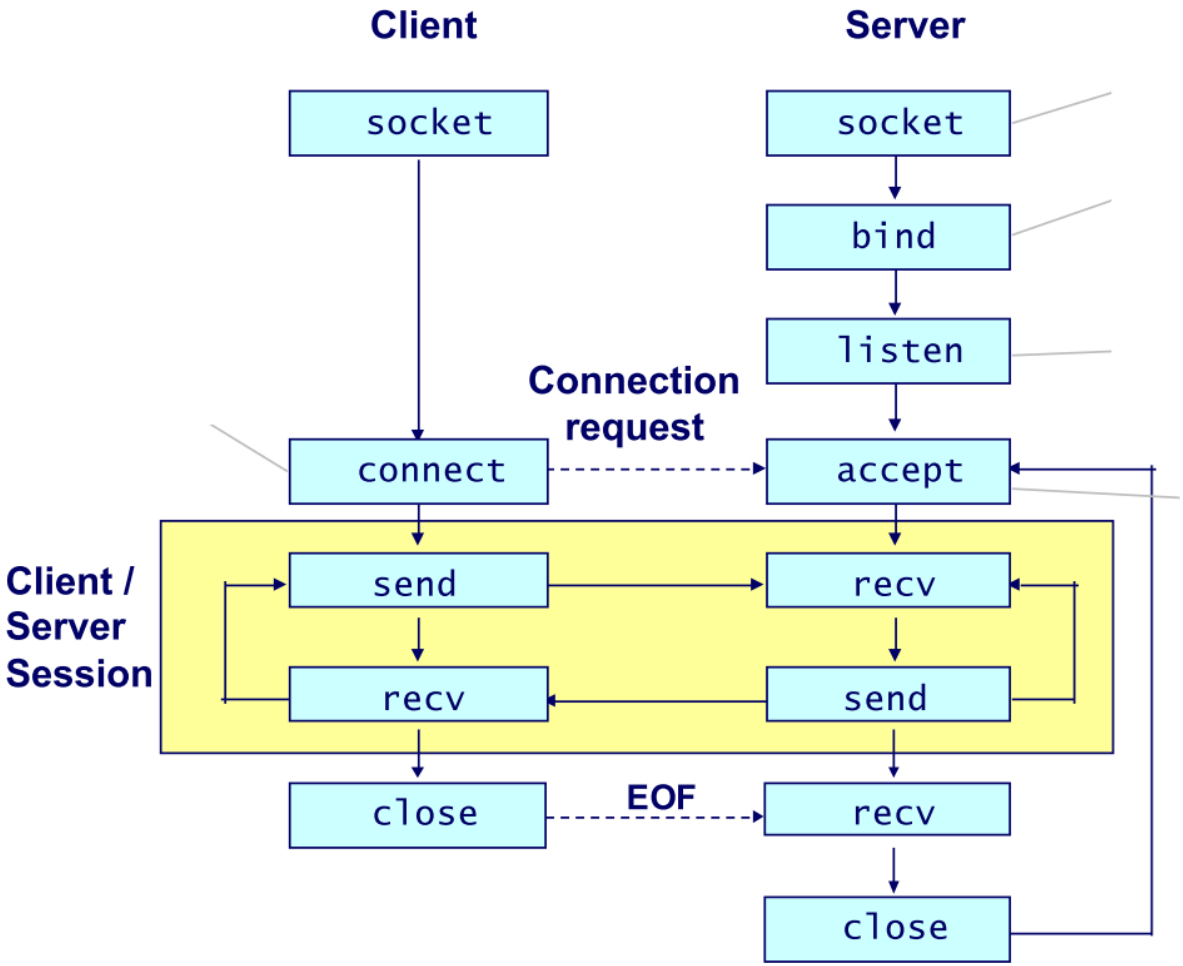
- clientSocket.close( )



Put Them Together!

- UDPServer.py and UDPClient.py
- Demo

Client/Server Flow for TCP



https://roxanageambasu.github.io/ds-class/assets/lectures/lecture4.pdf

Create a TCP Socket: Code

```
from socket import *
serverSocket = socket(AF_INET, SOCK_STREAM)
```

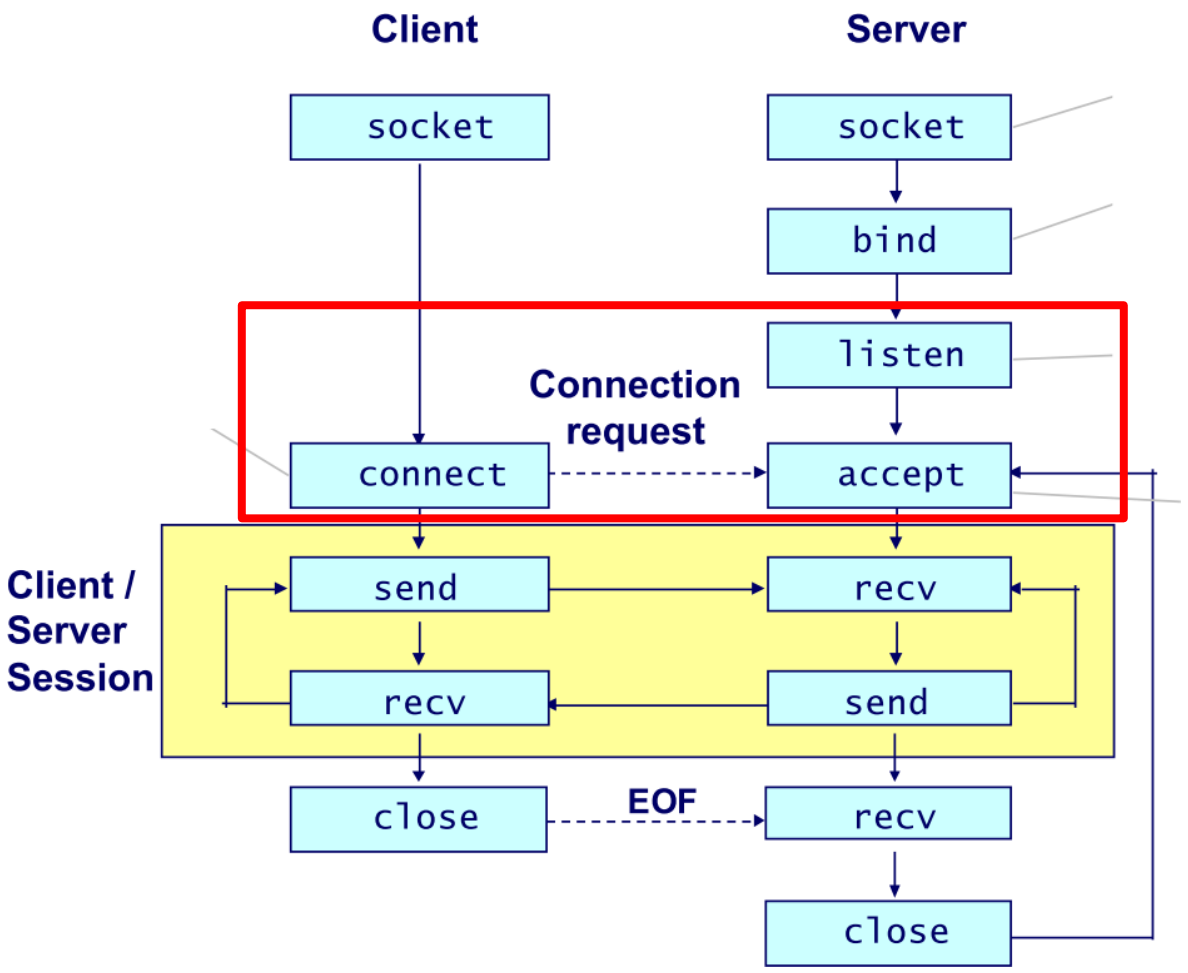
- We use SOCK_STREAM for TCP, instead of SOCK_DGRAM (UDP)

Server: Bind a TCP Socket

serverSocket.bind((‘1.0.0.1’, 8080))

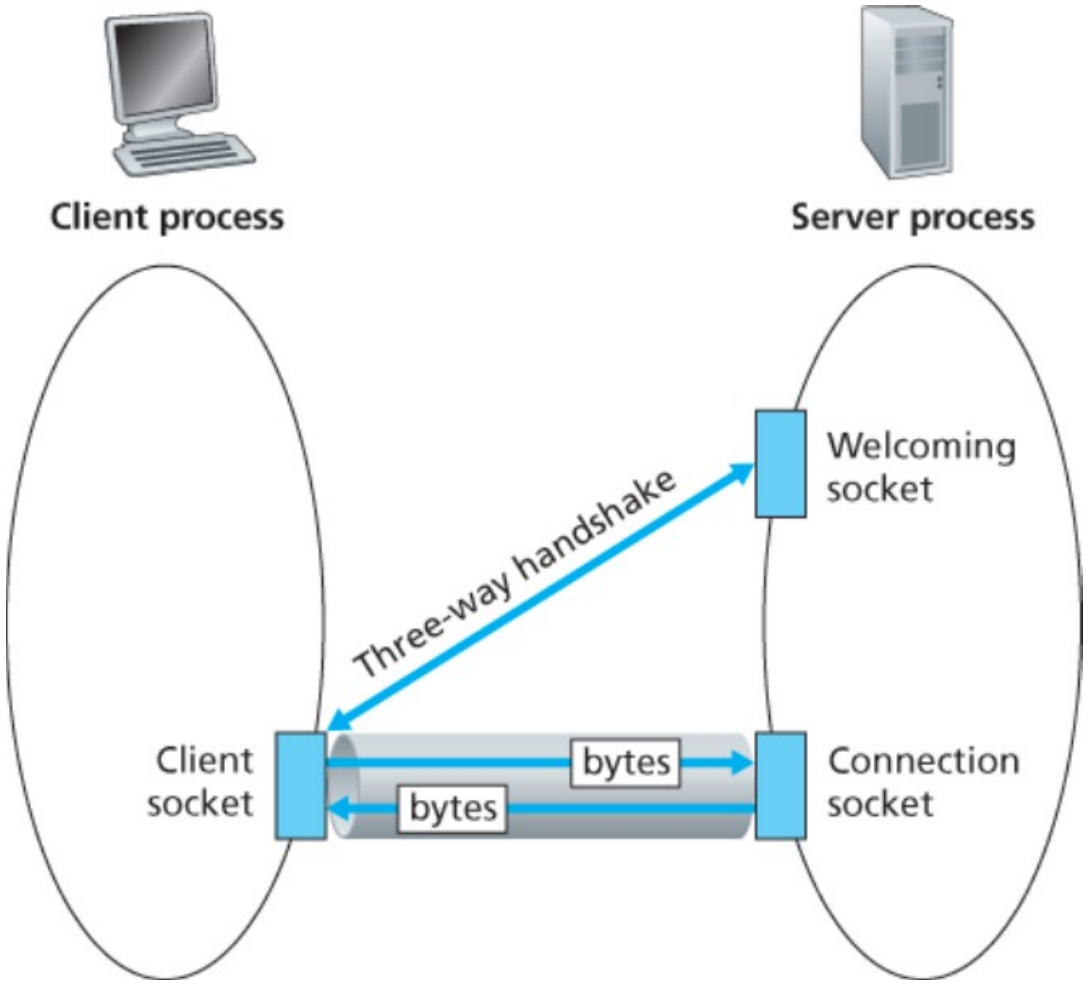
- Same as UDP

Listen, Connect and Accept



- TCP is a connection-oriented protocol
- Analogy: knocking on the welcoming door

How TCP works



- Two sockets! After contacting the welcoming socket, there is a newly created socket dedicated to the client.

image source: computer networking: a top-down approach seventh edition

Listen (server)

serverSocket.listen(1)

- Sets up listening socket to accept connections
- Parameter: maximum number of queued connections

https://docs.python.org/3/library/socket.html

Connect (client)

clientSocket.connect((serverName, serverPort))

- Three-way handshake happens here

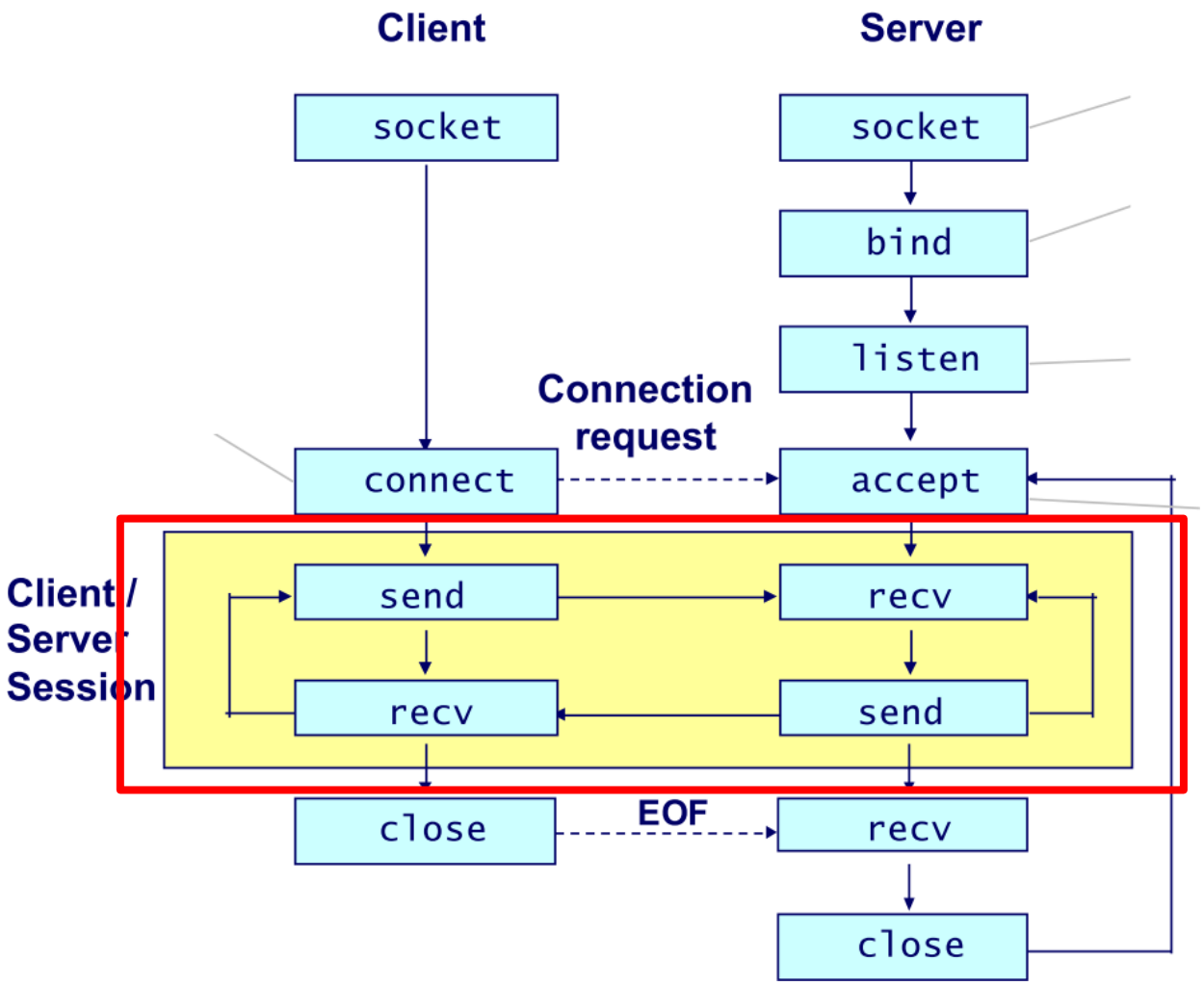
Accept (server)

connectionSocket, addr = serverSocket.accept()

- By default blocks until a connection request arrives
- Creates and returns a new socket for each new client
- Then resumes waiting for another connection

- connectionSocket: a new socket dedicated to that client
- addr: address of the client

Send and Receive Message



Send and Receive Message: Code

- client

```
clientSocket.send(message)
response = clientSocket.recv(2048)
```

- server

```
message = connectionSocket.recv(2048)
// modify message
connectionSocket.send(message)
```

- recv( ) is stream-based, not message-based
  - TCP: a continuous flow of bytes
  - UDP: explicitly create a packet and attach destination to it
- buffer size: You may want to use while loop to receive a very long message. Think carefully about the termination condition.

Close the socket

- In our echo-server example, client and server close sockets after sending message

connectionSocket.close( )

clientSocket.close( )

- May need to think more about when to close in project

Put Them Together!

- TCPServer.py and TCPClient.py
- Demo

Netcat

- Networking utility that reads and writes data across network connections
- Useful commands
  - Netcat server
    - nc –l <listening port>
  - Netcat client
    - nc <server IP> <server port>

http://netcat.sourceforge.net/

Thanks for listening

Codes will be available on Ed