University of
Applied Sciences

Programming 2

Networking with Python

Dr. Dhungana
Dr. Klikovits
SS2024 • Krems

Assuming you know:

Protocol: A protocol is a set of rules and standards that govern how data is transmitted over a network. Examples of protocols include TCP/IP, HTTP, and FTP.

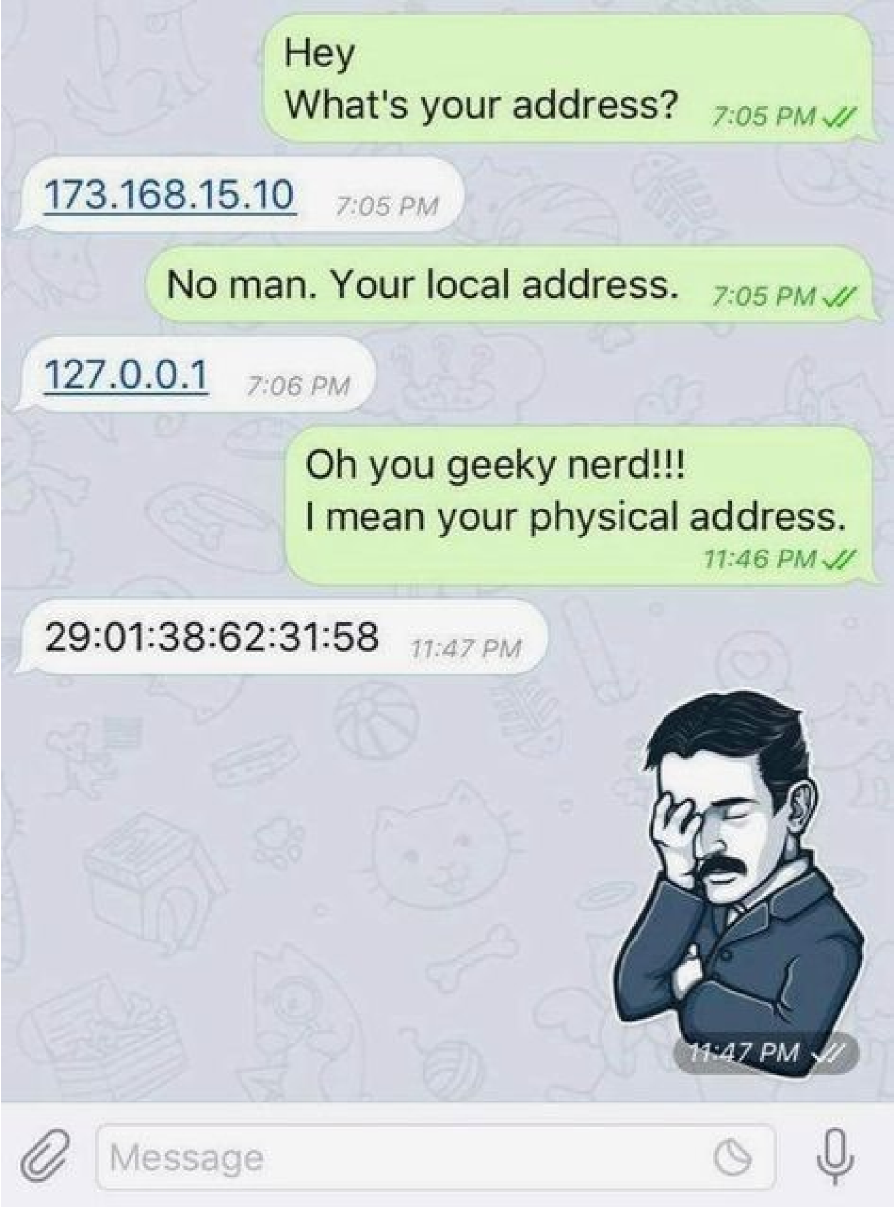
Topology: Network topology refers to the physical and logical arrangement of nodes on a network. The common network topologies include bus, star, ring, mesh, and tree.

LAN: A Local Area Network (LAN) is a network that covers a small area, such as an office or a home. LANs are typically used to connect computers and other devices within a building or a campus.

WAN: A Wide Area Network (WAN) is a network that covers a large geographic area, such as a city, country, or even the entire world. WANs are used to connect LANs together and are typically used for long-distance communication.

IP Address: An IP address is a unique numerical identifier that is assigned to every device on a network. IP addresses are used to identify devices and enable communication between them.

DNS: The Domain Name System (DNS) is a protocol that is used to translate human-readable domain names (such as www.google.com) into IP addresses that computers can understand.



Hey
What's your address? 7:05 PM

173.168.15.10 7:05 PM

No man. Your local address. 7:05 PM

127.0.0.1 7:06 PM

Oh you geeky nerd!!!
I mean your physical address. 11:46 PM

29:01:38:62:31:58 11:47 PM

11:47 PM

Client-Server Communication Paradigm

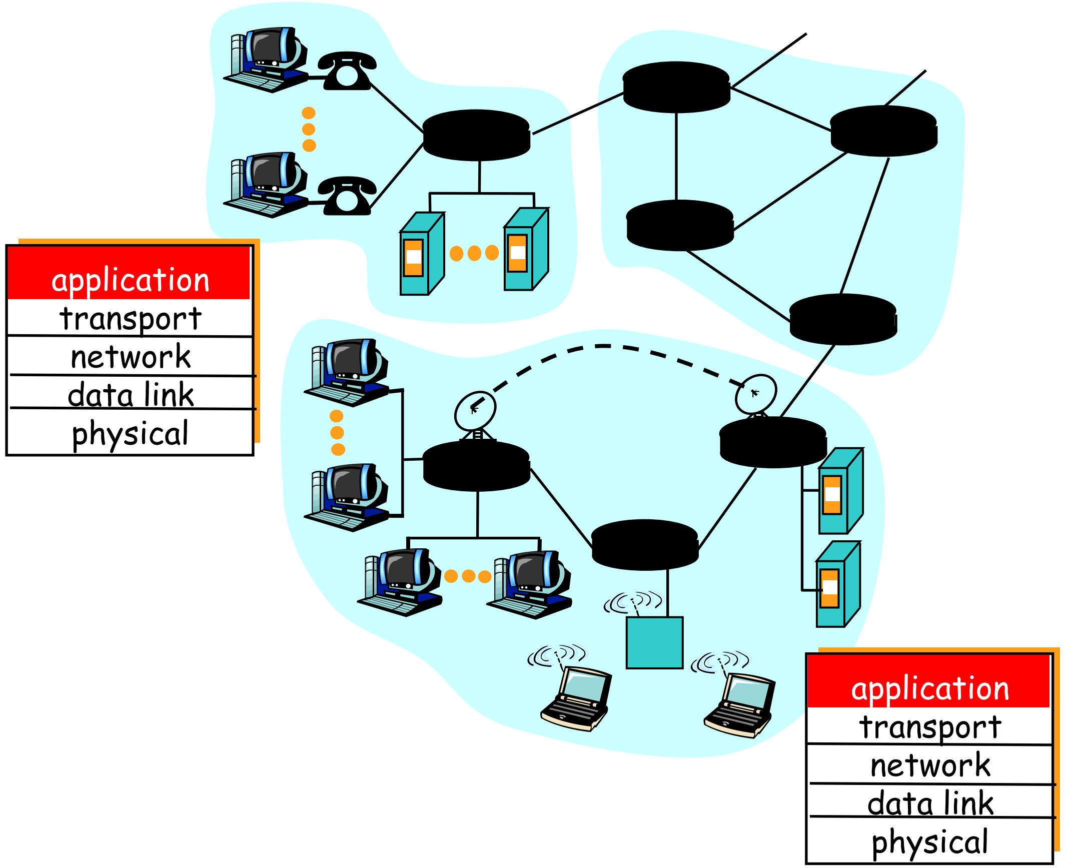
Typical network app has two sides: client and server

Client:

- initiates contact with server ("speaks first")
- typically requests service from server

Server:

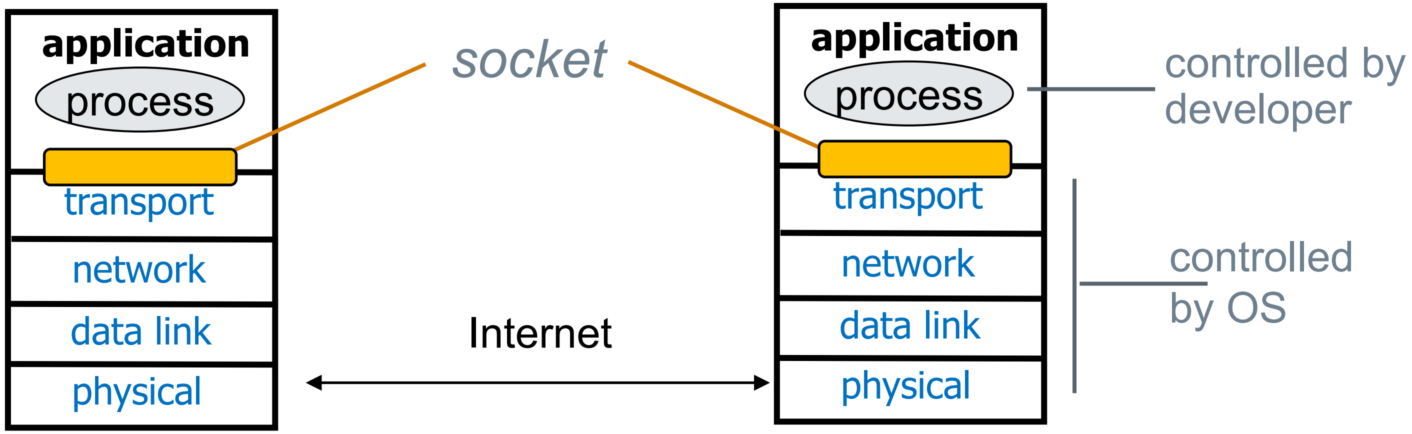
- provides requested service to client



Communication through Sockets

Socket: Interface between the application layer and the transport layer within a host

Analogous to a door: The sending process which is created in an application, shoves the messages out of the “door”



Sockets in Python

To create a socket

```
import socket

s = socket.socket(addr_family, type)
```

```
import socket

s = socket.socket(AF_INET,SOCK_STREAM)
```

Address family:

socket.AF_INET	Internet protocol (IPv4)
socket.AF_INET6	Internet protocol (IPv6)
socket.AF_UNIX	Unix Domain Sockets (UDS)

Socket types

socket.SOCK_STREAM	Connection based stream (TCP)
socket.SOCK_DGRAM	Datagrams (UDP)

General Information through sockets

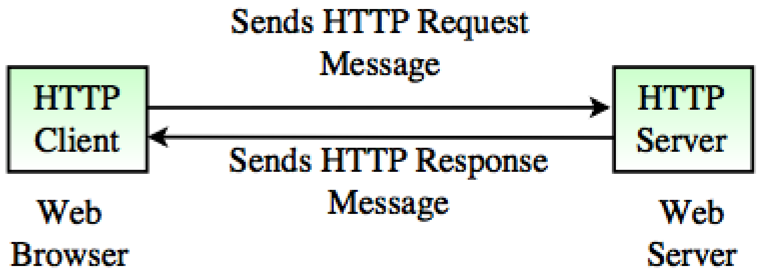
https://docs.python.org/3/library/socket.html

```
socket.gethostname() # get name of the current host

socket.gethostbyname("www.google.at")# get name of the current host

socket.gethostbyaddr("172.217.20.3") # get host based on a given address

socket.getaddrinfo('www.python.org', 'http') # additional info about a host

socket....(…) # see documentation for more
```

Simulating a HTTP client

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("example.com" , 80))
s.sendall(b"GET / HTTP/1.1\r\nHost: example.com\r\nAccept: text/html\r\n\r\n")
print(str(s.recv(4096), 'utf-8'))
s.close()
```

```
import requests
x = requests.get("http://example.com")
print (x.text)
```

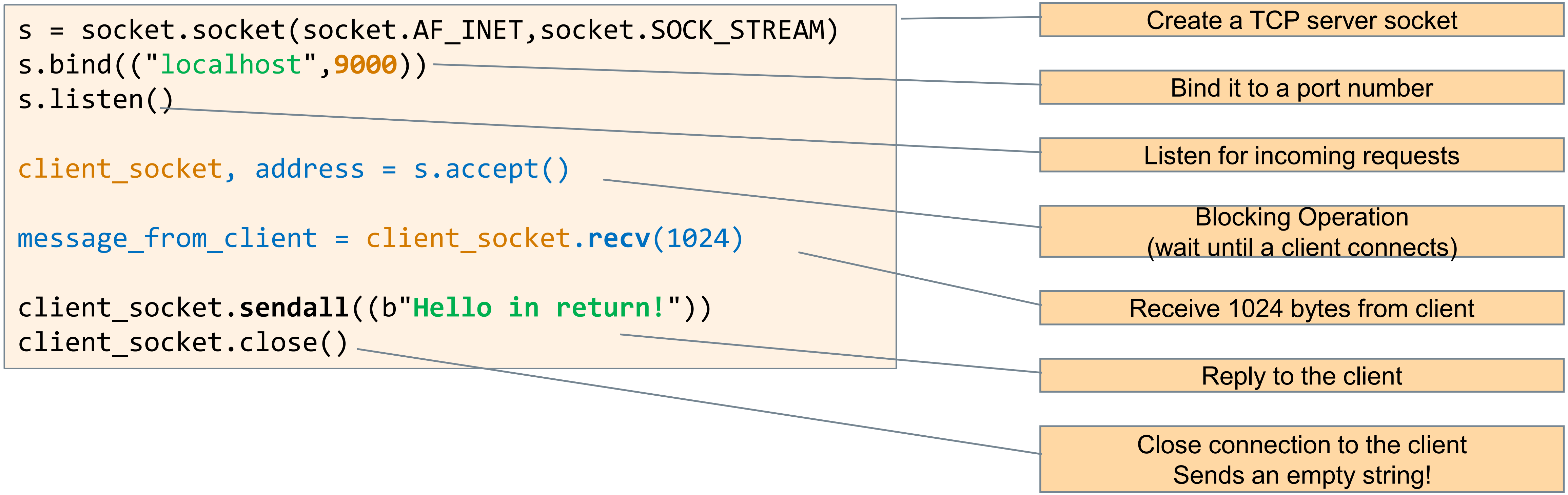


A “Hello-world” TCP client

```
# Connecting to a local server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("localhost", 9000))

msg = b"Hi Server" # can also use .encode('utf-8')

s.send(msg)
message_from_server = s.recv(4096)

s.close()
```

```
s.connect(("localhost", 9000))
ConnectionRefusedError: [WinError
10061] No connection could be made
because the target machine actively
refused it
```

A “Hello-world” TCP server



```
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.bind(("localhost",9000))
s.listen()

client_socket, address = s.accept()

message_from_client = client_socket.recv(1024)

client_socket.sendall((b"Hello in return!"))
client_socket.close()
```

Create a TCP server socket

Bind it to a port number

Listen for incoming requests

Blocking Operation
(wait until a client connects)

Receive 1024 bytes from client

Reply to the client

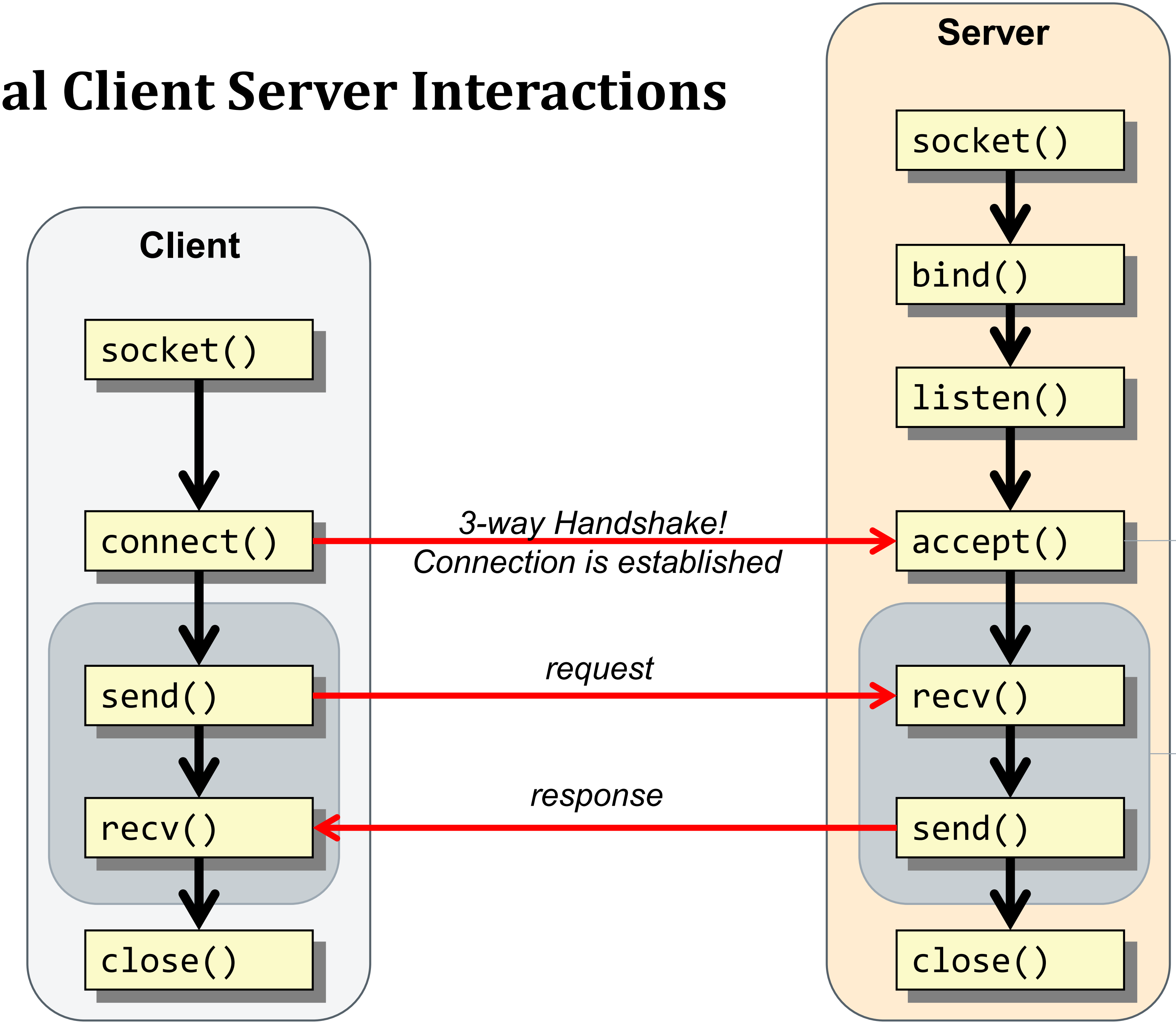
Close connection to the client
Sends an empty string!

Processing Client Requests

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('localhost', 5050))
s.listen(1)
conn, addr = s.accept()
while True:
    data = conn.recv(1024)
    formatRequested = data.decode ("utf-8")
    if not data: # end of data = ''
        break

conn.sendall(datetime.now().strftime(formatRequested).
encode("utf-8"))
conn.close()
```

Use a loop to keep the
connection alive to the client

Typical Client Server Interactions



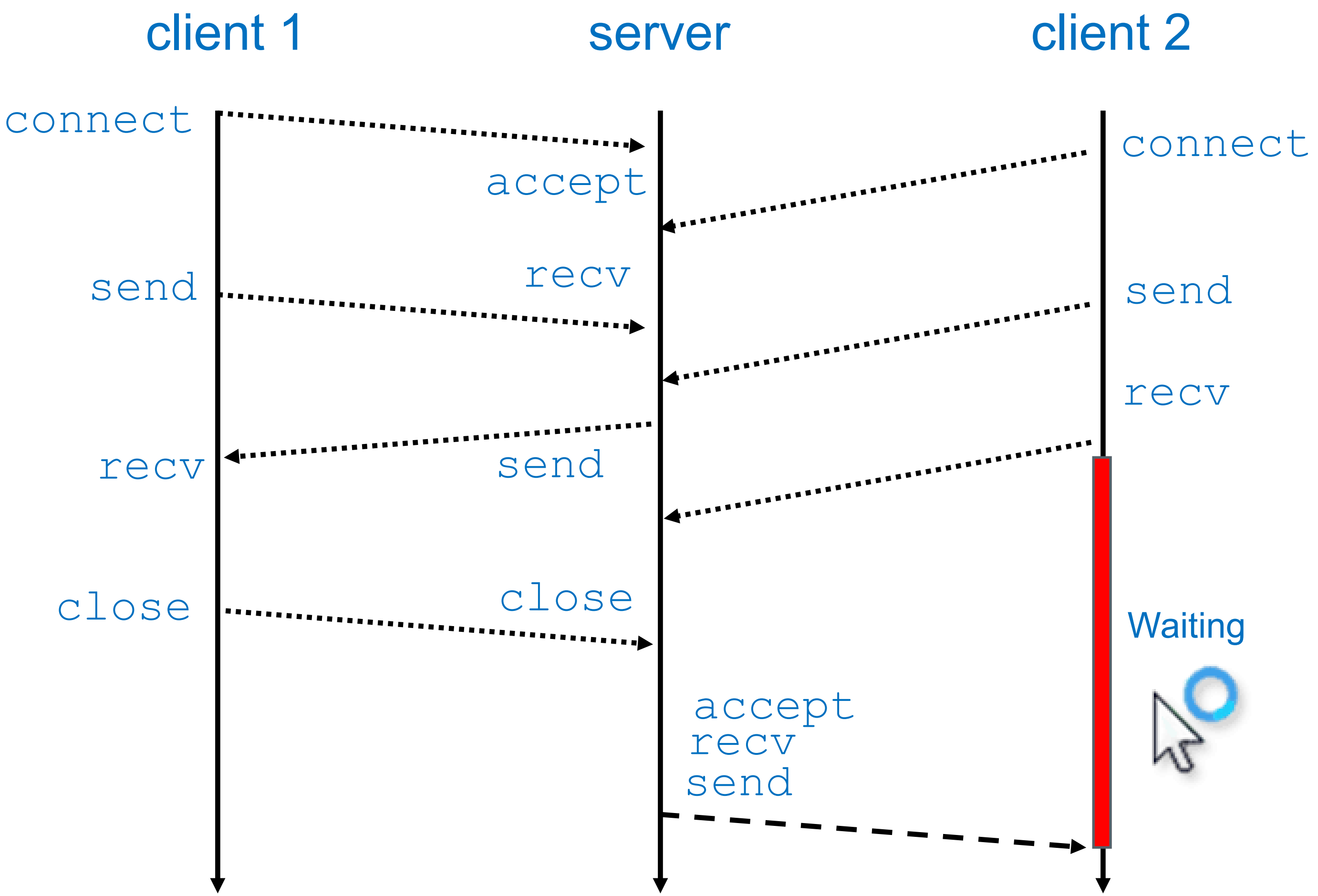
Wait for connections
On server "welcome socket"

recv() and send()
Through client-specific socket

Repeat until the end of the application

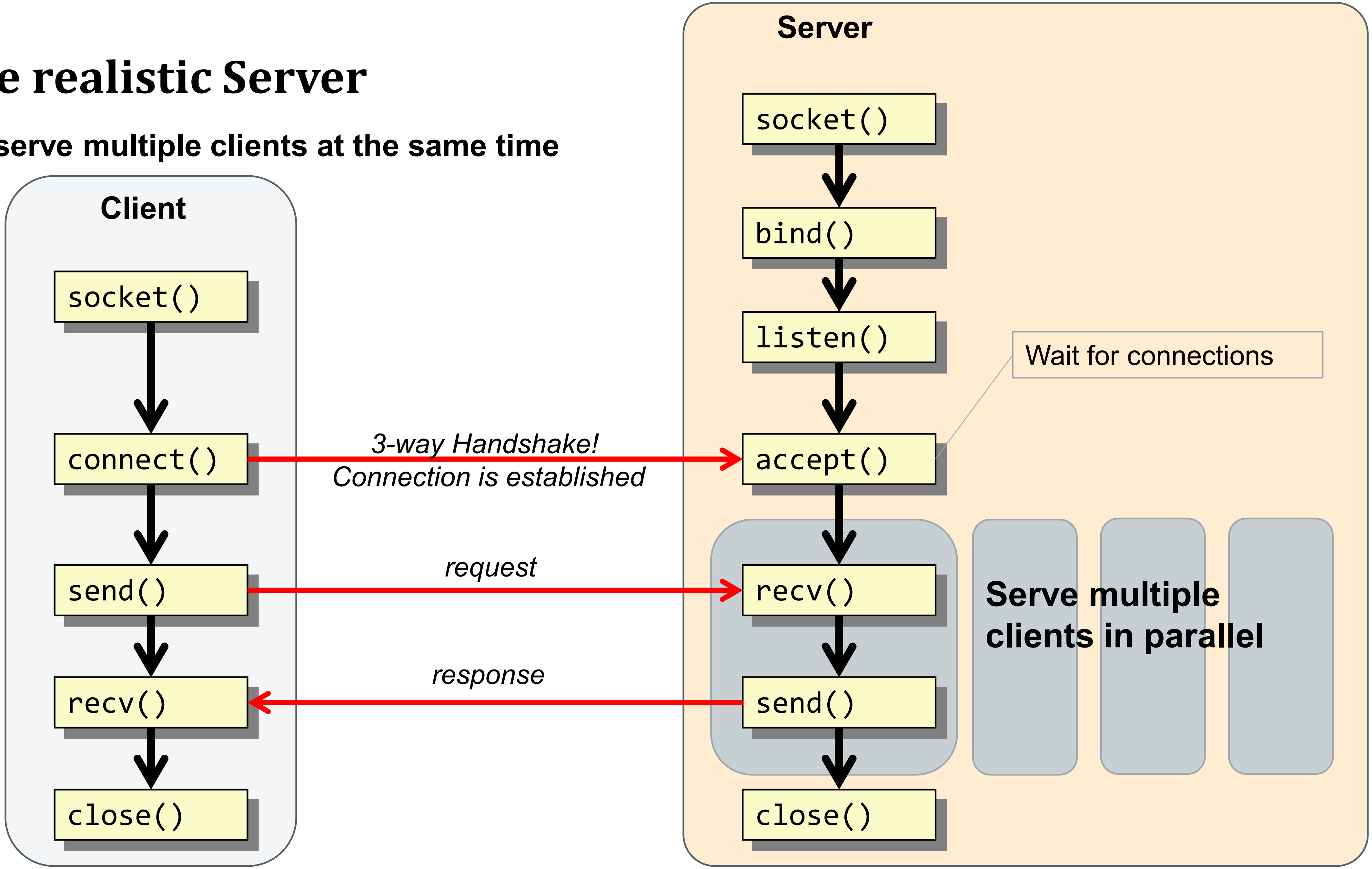
Iterative Server

Iterative servers process one request at a time



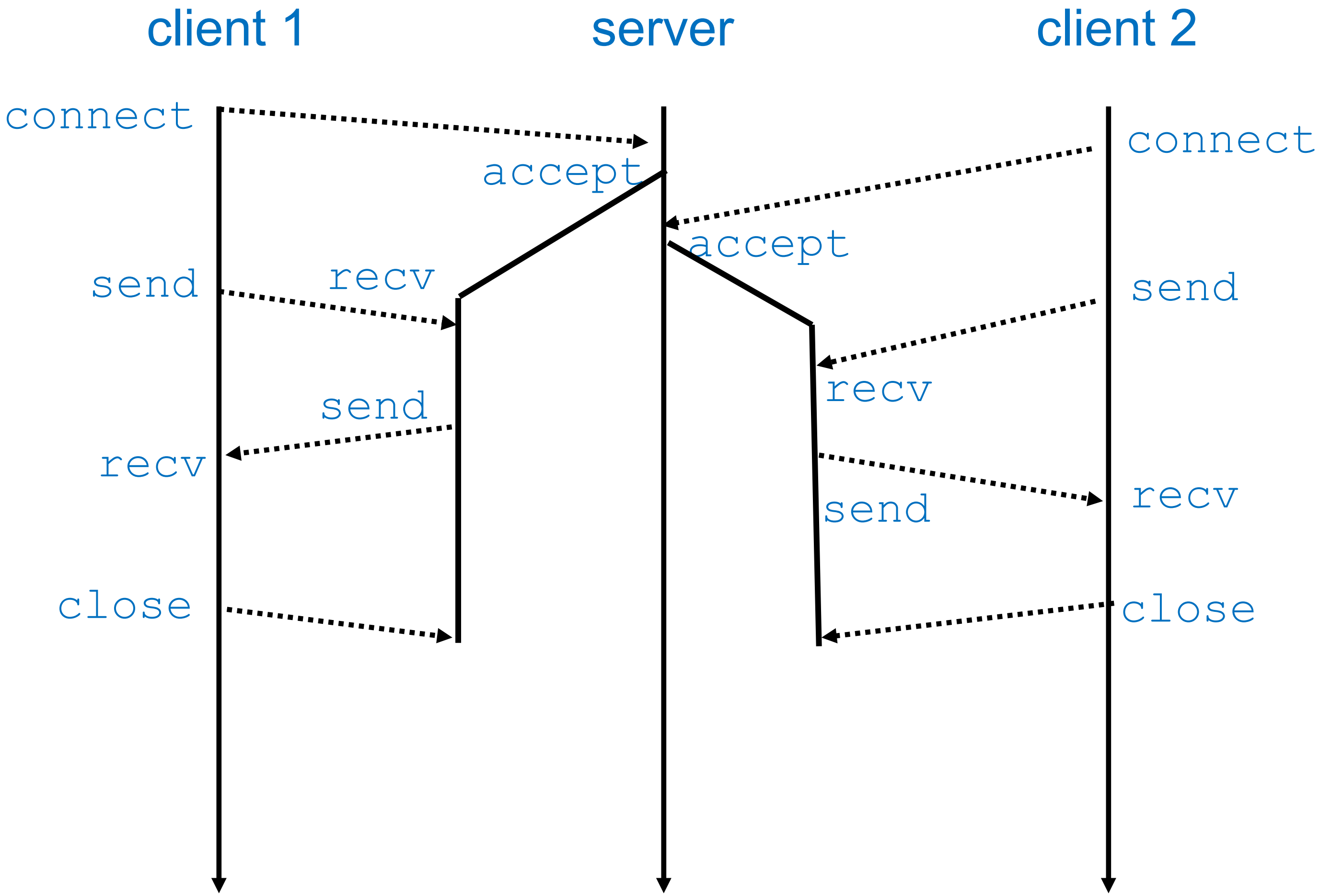
A more realistic Server

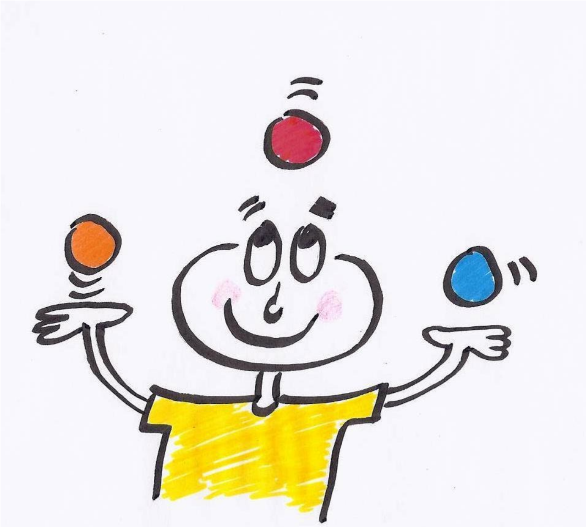
Typically serve multiple clients at the same time



Concurrent Server

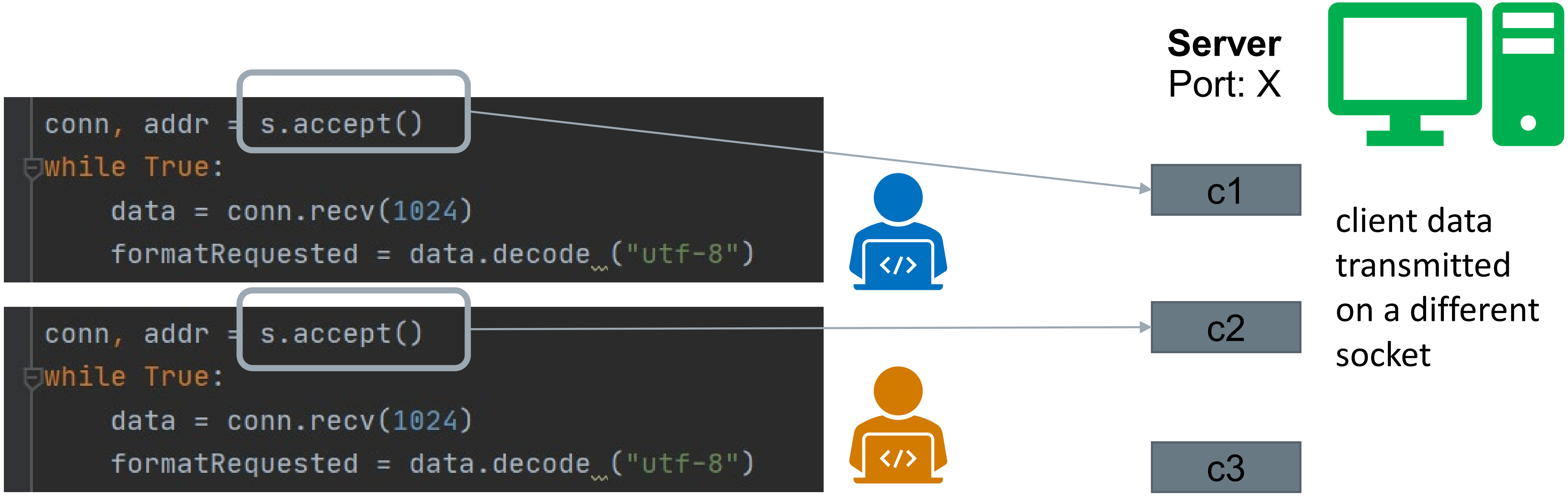
Spawn separate process/thread for each client





Sockets and Concurrency

- Servers usually handle multiple clients
- Each client gets its own socket on server
- New connections make a new socket



```
conn, addr = s.accept()
while True:
    data = conn.recv(1024)
    formatRequested = data.decode ("utf-8")
```

```
conn, addr = s.accept()
while True:
    data = conn.recv(1024)
    formatRequested = data.decode ("utf-8")
```

Server
Port: X

client data transmitted on a different socket

Multi-threading (More details later in the semester)

In computing, a **process** is an instance of a computer program that is being executed.

Any process has 3 basic components:

- An executable program.
- The associated data needed by the program (variables, work space, buffers, etc.)
- The execution context of the program (State of process)

A **Thread**:

- A thread is an entity within a process that can be scheduled for execution.
- **Smallest unit of processing** that can be performed in an OS (Operating System).

Run Functions as Threads

Creates a Thread object, and its **run()** method just calls the given function

```
import time
import threading
import random

def countFunction (name):
    count = 0
    while count <= 10:
        print(name + str(count))
        count += 1
        time.sleep(random.random())
```

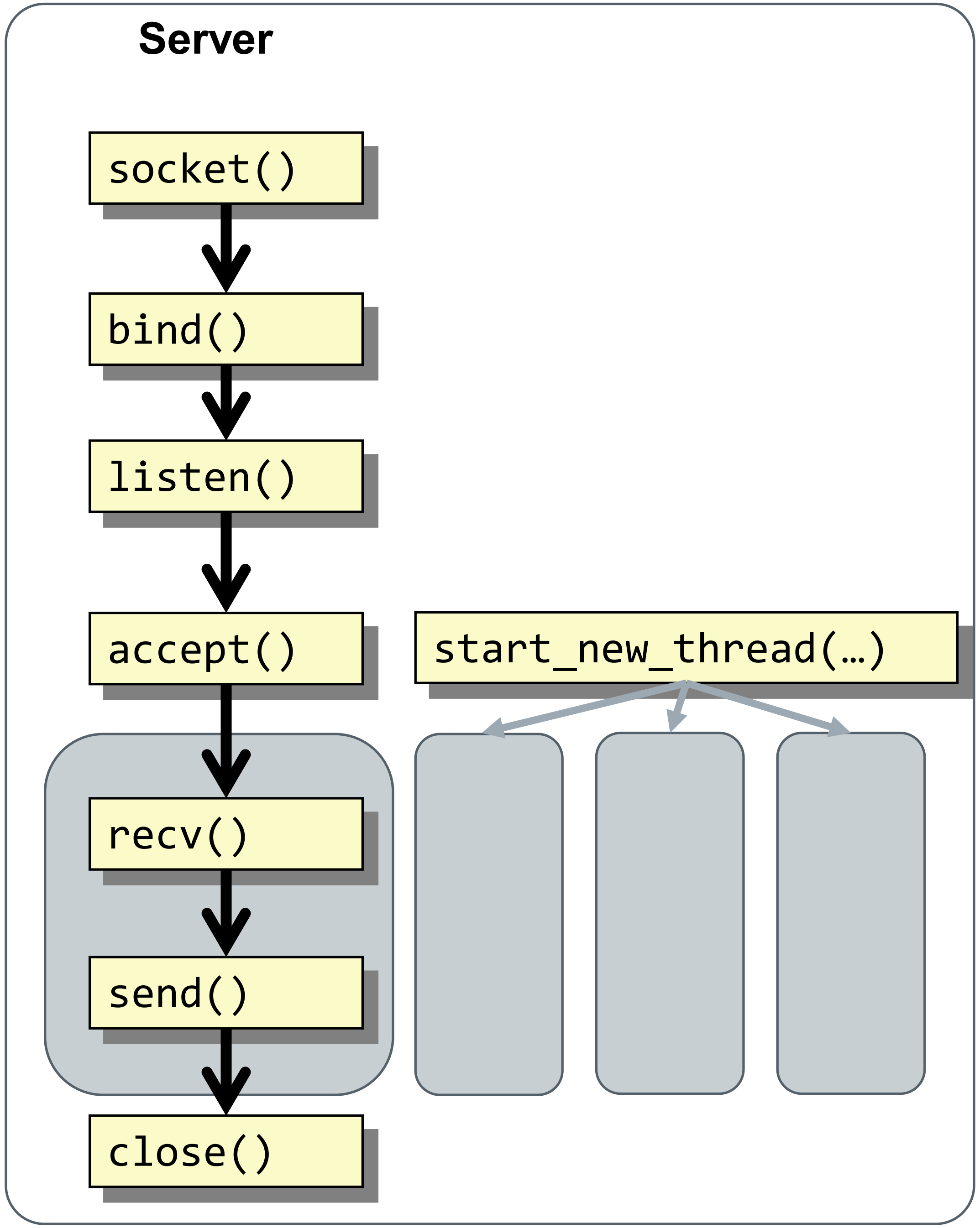
```
t1 = threading.Thread(target=countFunction, args=("A",))
t1.start()
```

Handling Multiple Clients

- Server must always be ready to accept new connections
- Must allow each client to operate independently (each may be performing different tasks on the server)
- Each client is handled by a separate thread

```
while True:
    client_socket, address = s.accept()
    t1 = threading.Thread
        (target=handle_client,
        args=(address, client_socket))
    t1.start()

client_socket.close()
```

```
def handle_client (address, socket):
   ## Your code to handle the client
```



Sample Application: Client Server Quiz

Task 5.1 Sample Application: Client Server Quiz

Create a simple client-server application where:

Clients connect to the server and starts a quiz game (sends her name and the number of questions she wants to play)

The server sends a multiple-choice question to the client (repeat for n questions)

The client answers the questions

The server keeps track of the points achieved by the user

At the end of the game, the server reports the total number of points of the user

----

BONUS Task 1: The server reports the rank of the user in comparison to other users

BONUS Task 2: The server stores the results persistently (e.g., in a file) and the user can continue the game later, by entering the same name

BONUS Task 3: The users are registered and can login by using a user-name and password

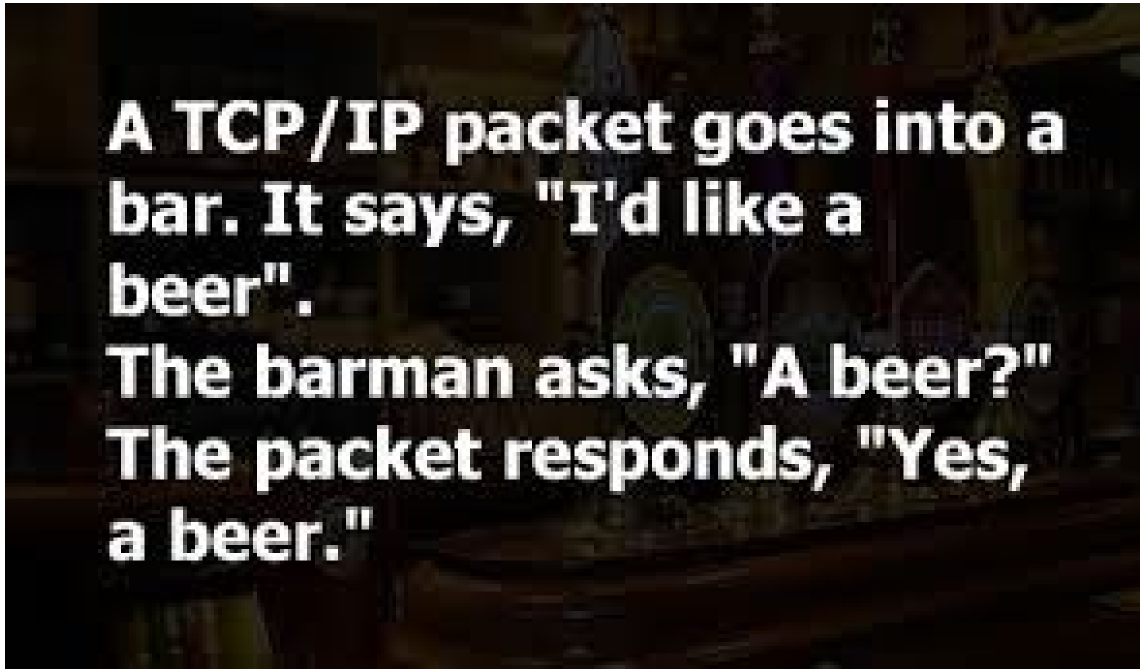
Protocols of Communication

- Transmission Control Protocol (TCP) – used for data transmission
- User Datagram Protocol (UDP) – used by programs to send short datagram messages
- Hypertext Transfer Protocol (HTTP) – application protocol that uses hyperlinks between nodes containing text
- Post Office Protocol (POP) – used by local email clients to retrieve email from a remote server over TCP IP
- File Transfer Protocol (FTP) – protocol to transfer computer files from a server to a client and vice versa
- Internet Message Access Protocol (IMAP) – a communication protocol used by email clients to retrieve messages from a mail server over TCP IP

WHAT DO I DO NOW?
WHEN IS HE GOING TO TAKE MY CARD?
HE SEEMS A BIT UNFRIENDLY?
WHOOA! PERSONAL SPACE INVASION!
I'M NEXT, WHAT SHOULD I DO ?

More on Protocols

- FTP https://tools.ietf.org/html/rfc959
- POP3 https://tools.ietf.org/html/rfc1081
- HTTP https://tools.ietf.org/html/rfc2616
- XMPP https://tools.ietf.org/html/rfc6120
- ...



A TCP/IP packet goes into a bar. It says, "I'd like a beer".
The barman asks, "A beer?"
The packet responds, "Yes, a beer."

Transport Protocols

TCP service:

- connection-oriented: setup required between client, server
- reliable transport between sender and receiver
- flow control: sender won’t overwhelm receiver
- congestion control: throttle sender when network overloaded

UDP service:

- unreliable data transfer between sender and receiver
- does not provide: connection setup, reliability, flow control, congestion control

Socket Programming with UDP

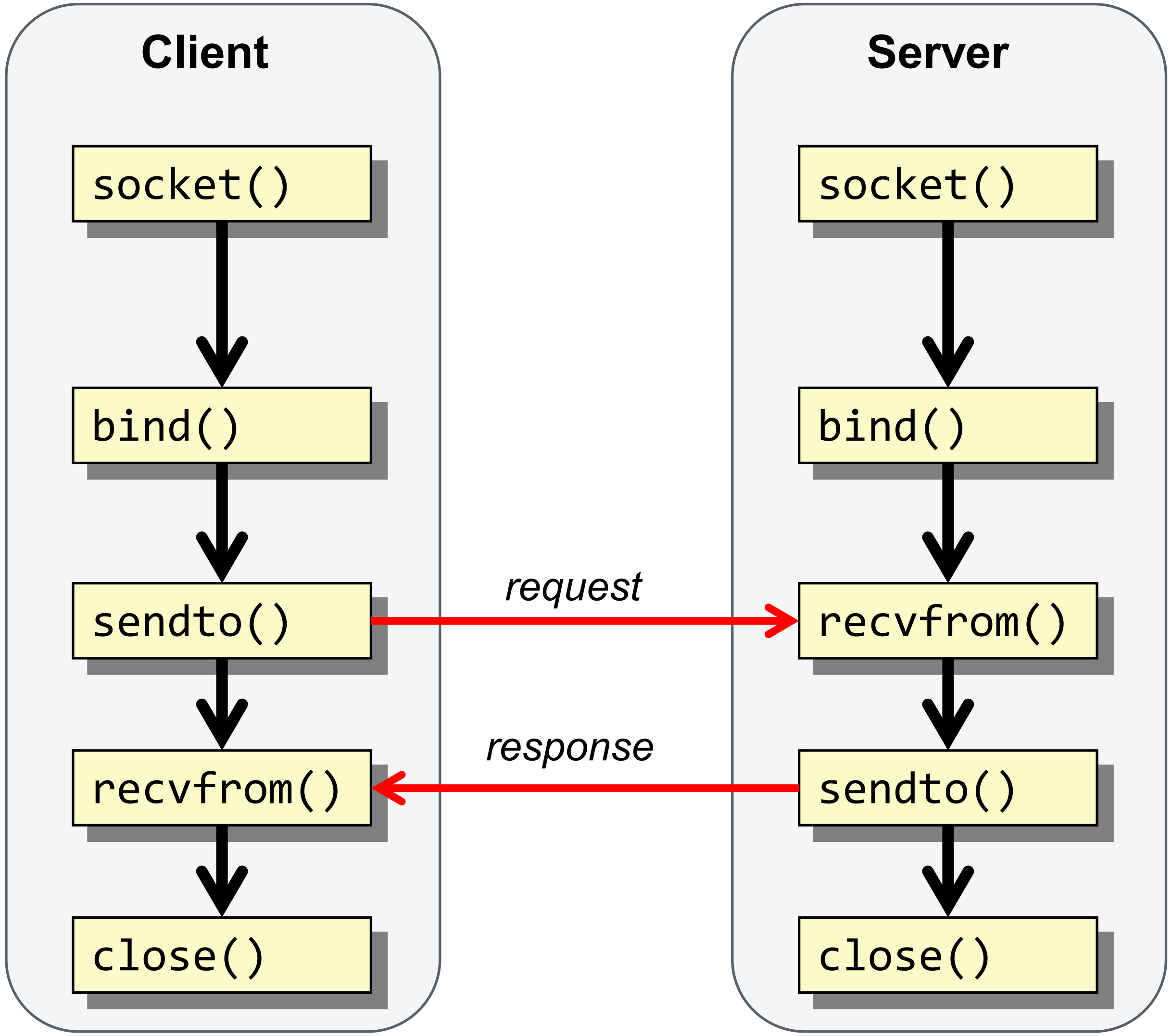
UDP: no "connection" between client & server

- 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

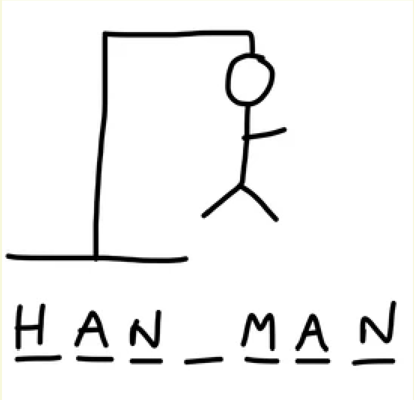


Network Hangman (single client)

Task 5.2 A networked Hangman game

Create a simple client-server application for a hangman game, where:

Clients connect to the server and request for a game
The server responds with meta-data about the word to guess (length)
The client guesses a character
The server responds with the position of the character in the word (if present)
The client guesses another character
….
The server responds with “end” of game signal
..

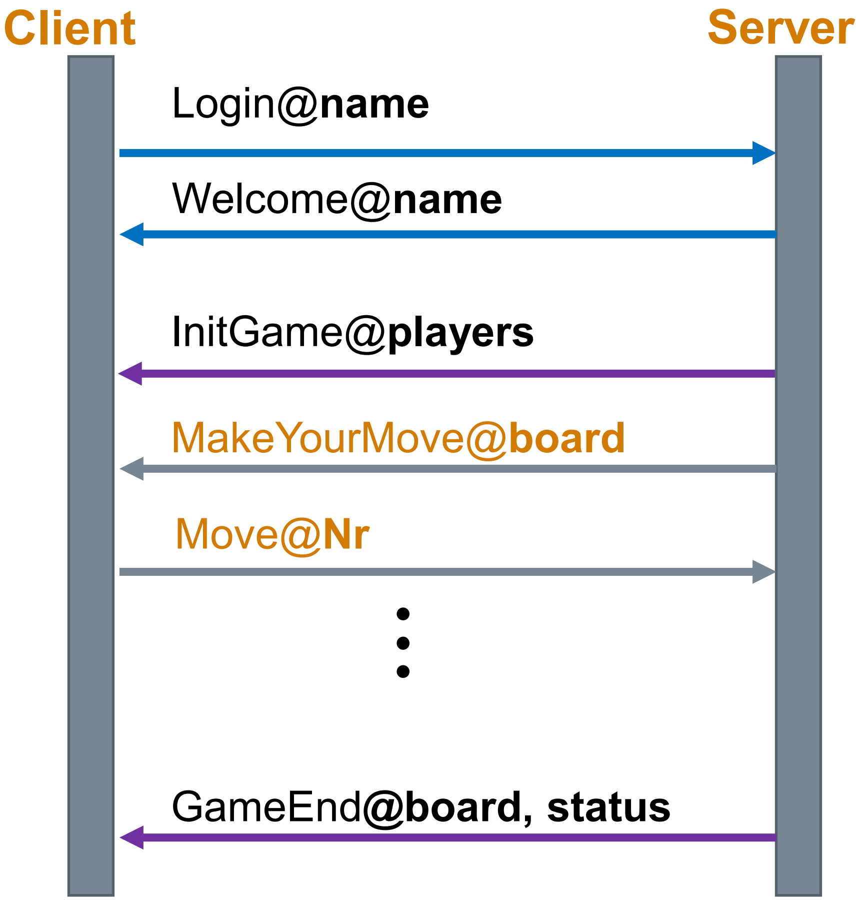


Implement a Tictactoe Protocol (two clients)

Task 5.3 A networked Tic Tac Toe game

Assume we have a server that allows two players to play tic-tac-toe against each other.

- Client connects to server and sends its name
- Server responds with “Welcome Name”
- Server finds two players and initiates a game
- Server sends “init”@ gameid and name of players
- Server requests each player in turn – to make a move
- Server checks the status of the game and determines the winner
- Server sends “GameEnd” message



Tictactoe.py is given, use it on the server side to manage the game logic.