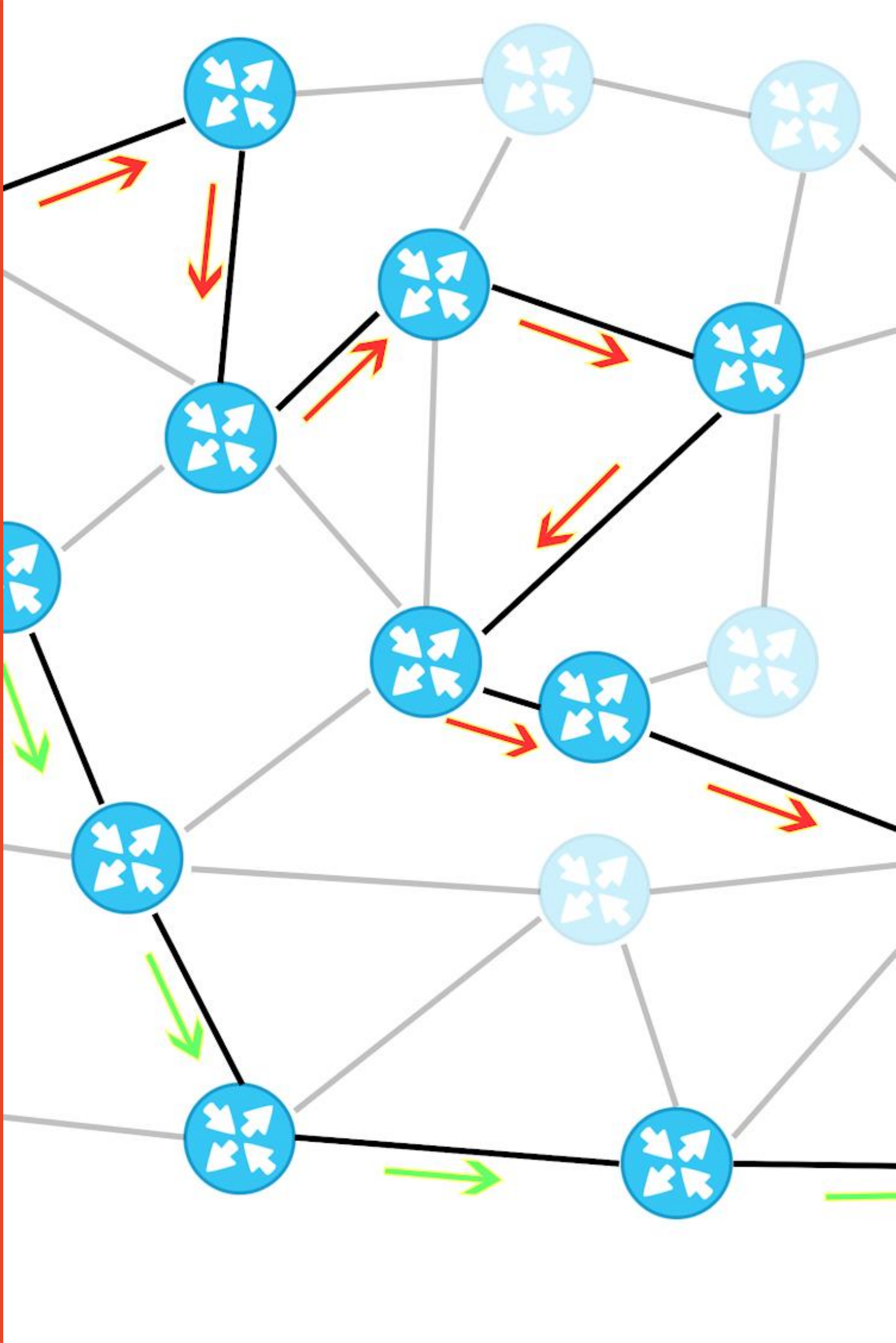
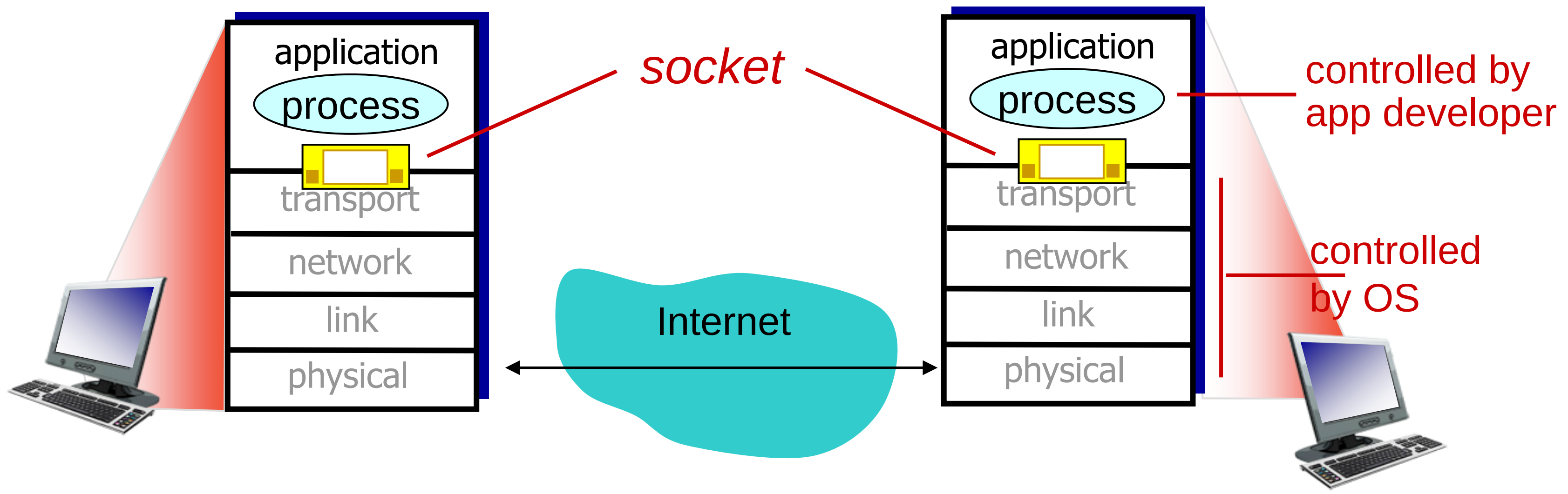
Distributed Systems

Lab 3: Client/Server Communication

THE UNIVERSITY OF SYDNEY



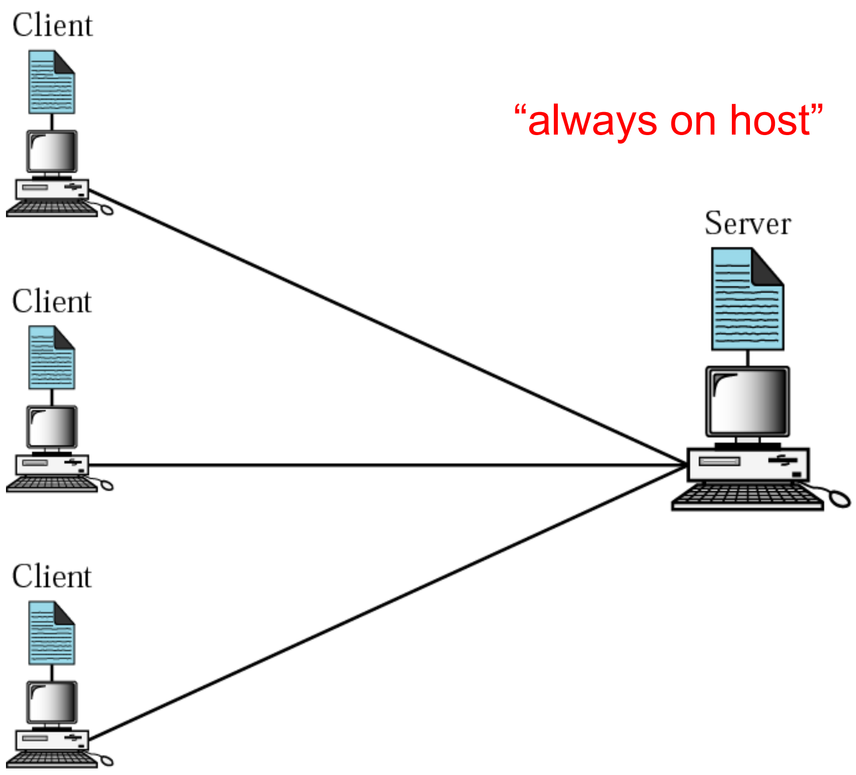
Socket



- A communication channel through which two programs communicate over a network
- An end-point for an Internet network connection
  - what the application layer "plugs into"
  - *determined by two things: Host Address (IP address) & Port Number*
- Most common type of socket applications: *client-server applications*

Client/Server Communication

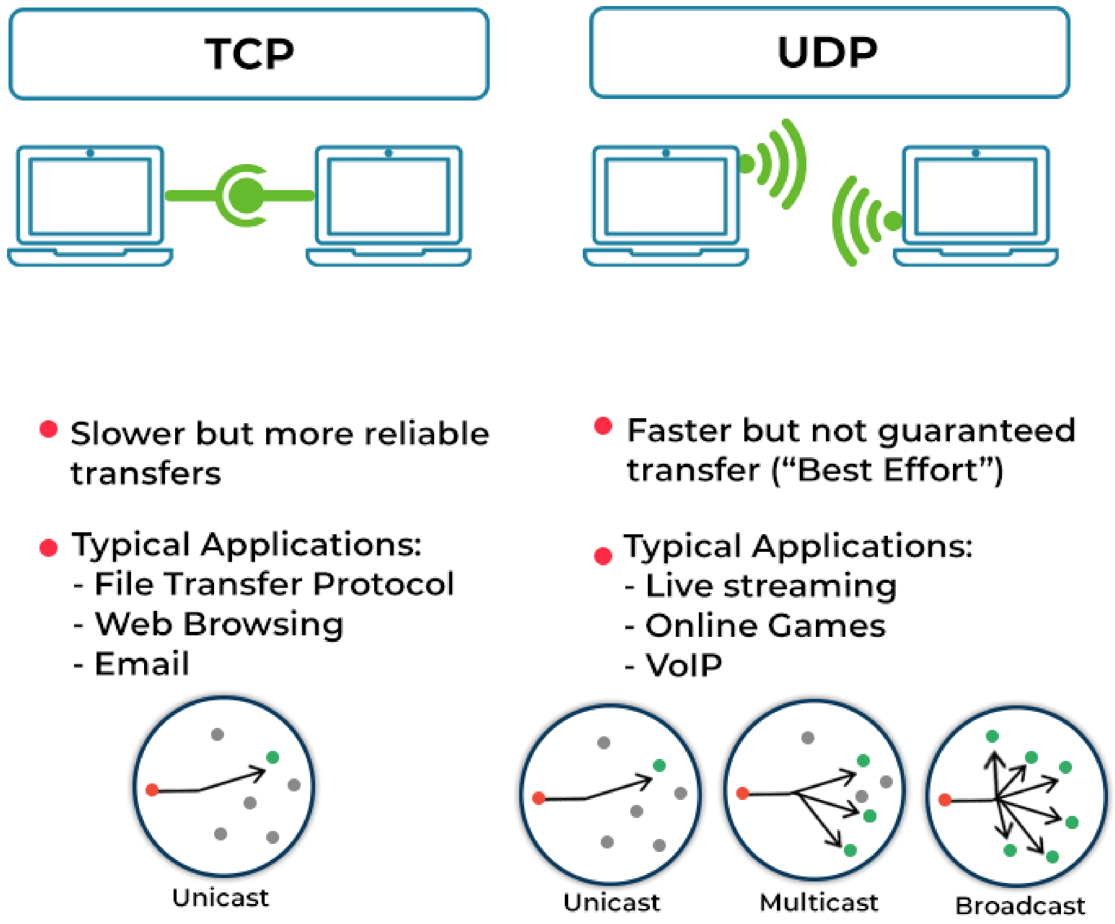
- ***Servers***
  - Provide a certain type of service, e.g. *emails, files, etc.*
  - Run all the time
  - Listen to a well-known port and passively open connection.
- ***Clients***
  - Run when needed, then terminate
  - Actively Open TCP or UDP connection with Server's socket.



Q: What is the main difference between TCP and UDP?

➢ *TCP is a connection-based protocol, UDP is connectionless.*

Client/Server Communication



- Slower but more reliable transfers
- Typical Applications:
  - File Transfer Protocol
  - Web Browsing
  - Email

- Faster but not guaranteed transfer ("Best Effort")
- Typical Applications:
  - Live streaming
  - Online Games
  - VoIP

Q: What might be problems of UDP?

➢ *Lost packets (Not resent) or Out-of-order packets.*

Socket programming for Client/Server Applications

❑ Client

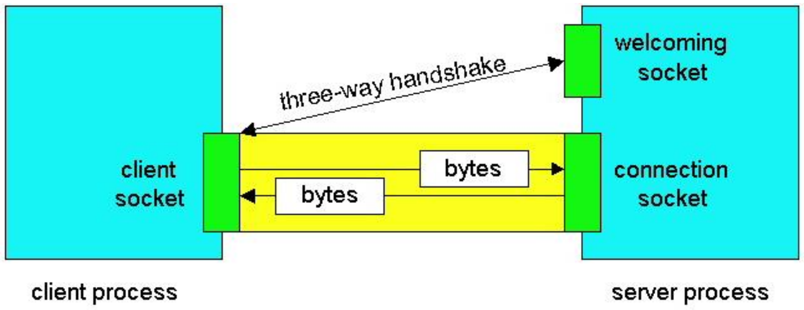
1. Create client-local socket
2. Configure IP address, port number of Server's process
3. Establish connection to server
4. Wait for acknowledgement from server
5. Send message to server
6. Receive message from server

❑ Server

1. Create a listening socket
2. Bind the local port and connection address
3. Listen for client connection
4. Accept connection from client
5. Send Acknowledgment
6. Receive message from client
7. Send message to client

➢ **How it works:**

- Server must first be running and listening on a welcoming socket
- When contacted by clients, Server creates new socket to communicate with client
  - ➢ allows to talk with multiple clients



Q: How to distinguish between different connection sockets?

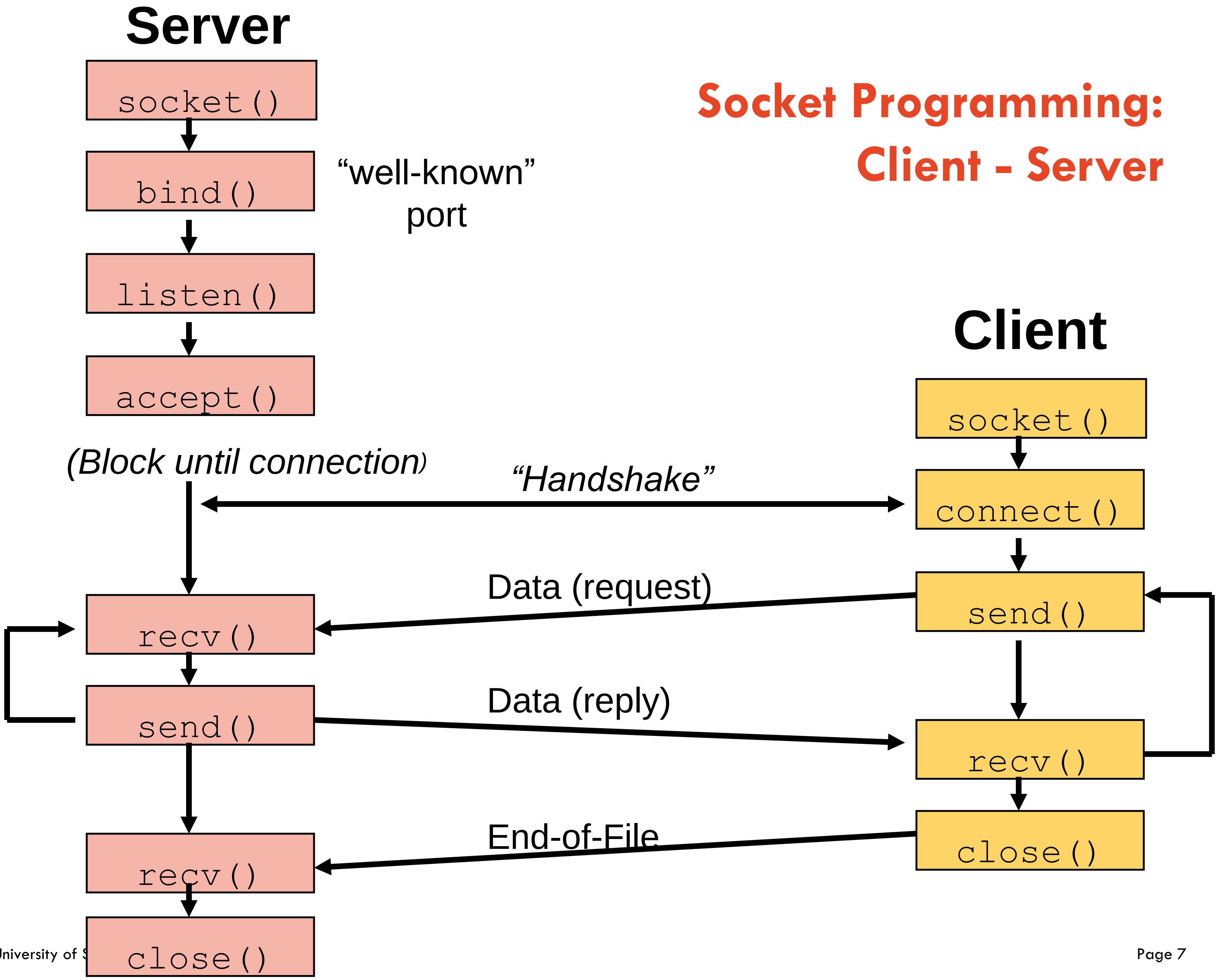
➢ *IP address & a random source port number upon connection initiation.*

Socket Programming: APIs

The primary socket API functions and methods:

socket()	• creates a socket object
.bind()	• associate the socket with a specific network interface and port number
.listen()	• listens for connections from clients.
.accept()	• accept, or complete, the connection.
.connect()	• establish a connection to the server and initiate the handshake
.send()	• send data
.recv()	• receive data
.close()	• close socket connection

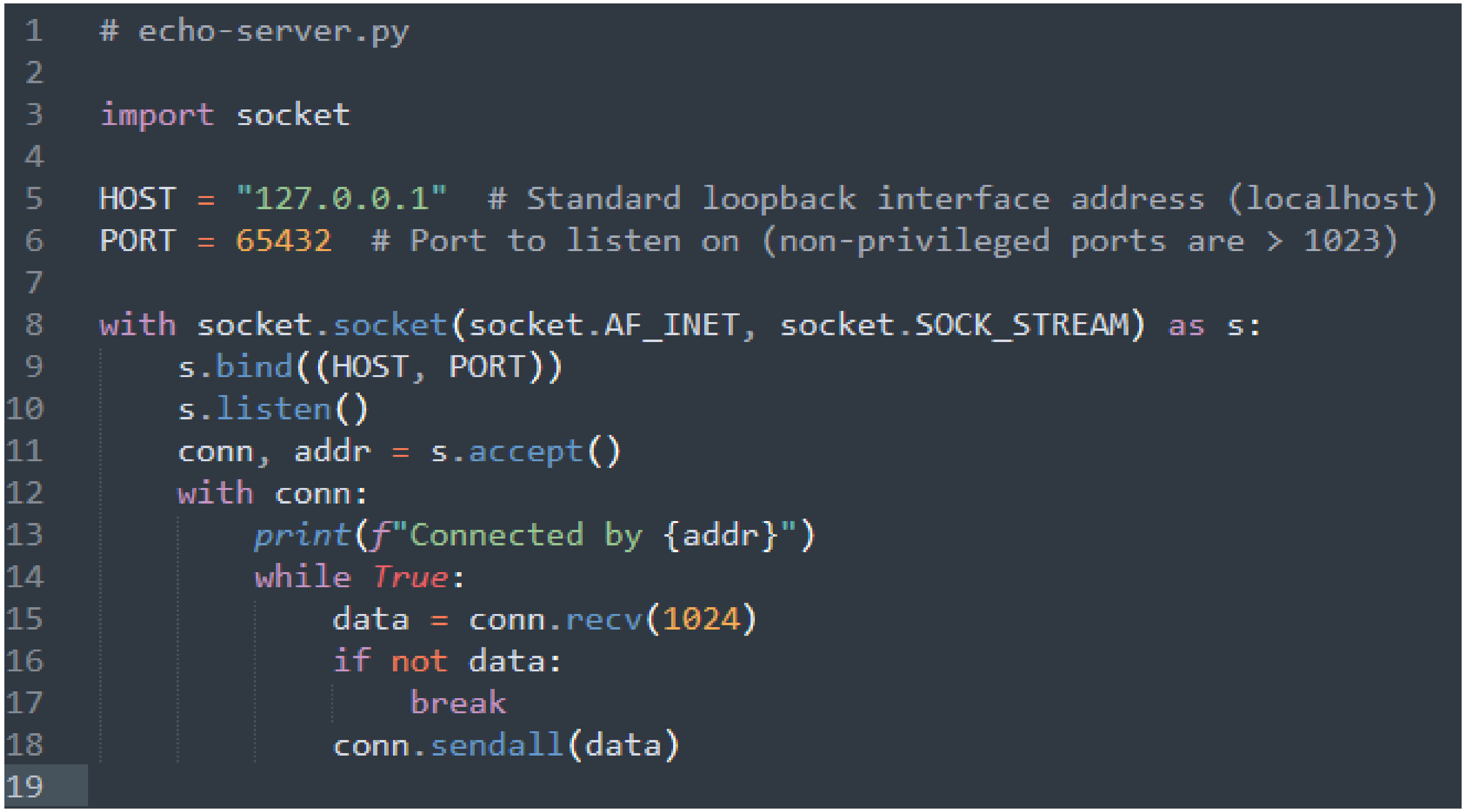
Socket Programming: Client - Server



Socket Programming: Ports

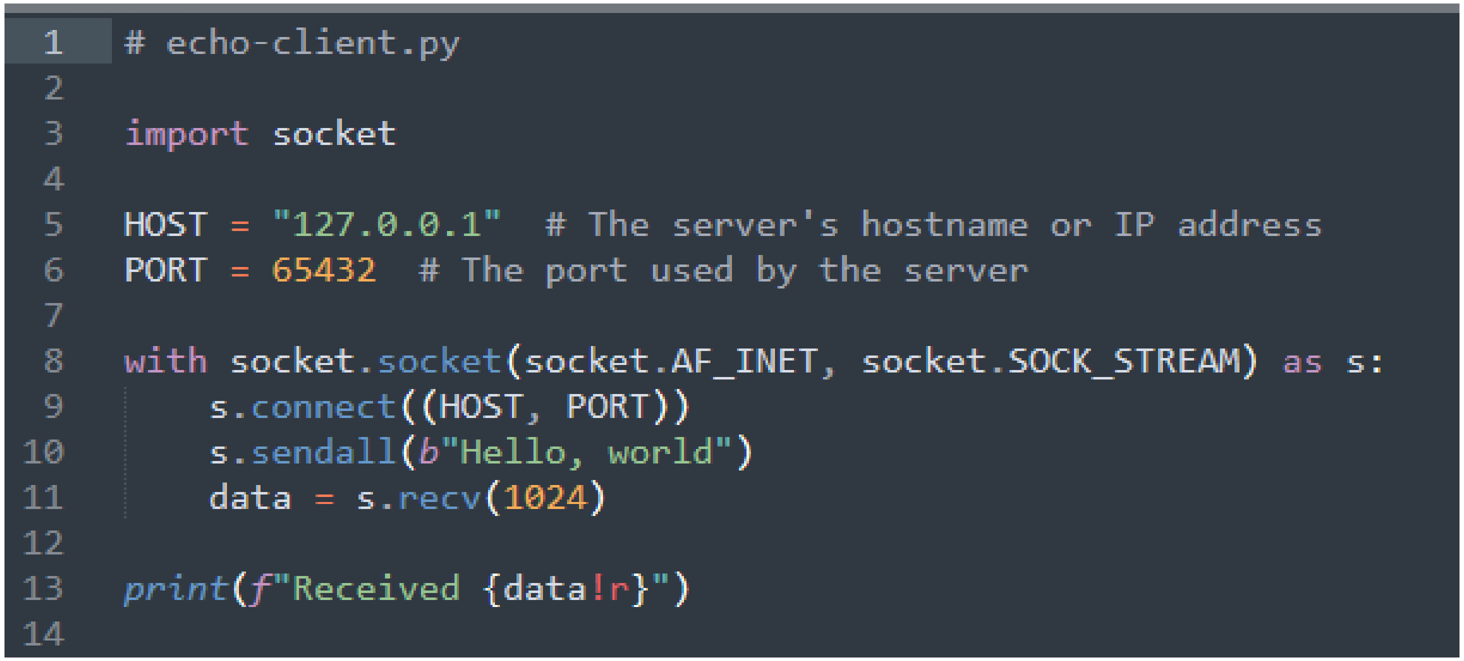
- Numbers (typical, since vary by OS):
  - `0-1023` “reserved”, must be root
  - `1024 - 5000` “ephemeral”
  - Above `5000` for general use
    - (`50,000` is specified max)
- Well-known, reserved services (see `/etc/services` in Unix):
  - `ftp    21/tcp`
  - `telnet 23/tcp`
  - `finger 79/tcp`
  - `snmp   161/udp`
  - `smtp   25`

Socket Programming: Simple Server



```python
# echo-server.py

import socket

HOST = "127.0.0.1"  # Standard loopback interface address (localhost)
PORT = 65432  # Port to listen on (non-privileged ports are > 1023)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print(f"Connected by {addr}")
        while True:
            data = conn.recv(1024)
            if not data:
                break
            conn.sendall(data)

```

Socket Programming: Simple Client



```python
# echo-client.py

import socket

HOST = "127.0.0.1"  # The server's hostname or IP address
PORT = 65432  # The port used by the server

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b"Hello, world")
    data = s.recv(1024)

print(f"Received {data!r}")
```

Q: How this client/server program works?

- ➢ *The server will simply echo whatever it receives back to the client.*

Q: If multiple clients connect to the server, how can it handle?

- ➢ *Multithreading*

Exercise 1

“What time is it?” Server

<u>The server, written in a class DateServer, file DateServer.py</u>

- using Socket in order to accept an incoming connection
- should always be listening to some incoming connections on a non-reserved port,
  - e.g., 6015.
- wait the connection from client and sends information in response to some client request
- Run the server in the background before running the client

Exercise 2

“What time is it?” Client

<u>The client, written in a class DateClient, file DateClient.py</u>

- ❑ Initiating the connection by creating a socket targeting the local machine,
  - ❑ identified by the IP address “127.0.0.1” and the service port is chosen.
- ❑ Sending a message to the server to ask about the current date and time
- ❑ Reading the response from the server to print it out.

Exercise 3

Logging Server

Design a new LoggingDateServer, written in a class LoggingDateServer, file LoggingDateServer.py

- in addition to answering the date to the client (as in Ex1,2), log each received request from a client
- Each time the server receives a request from a client,
  - stores a line in a new file whose name contains a monotonically increasing number, writing for example files log0.txt, log1.txt, log2.txt