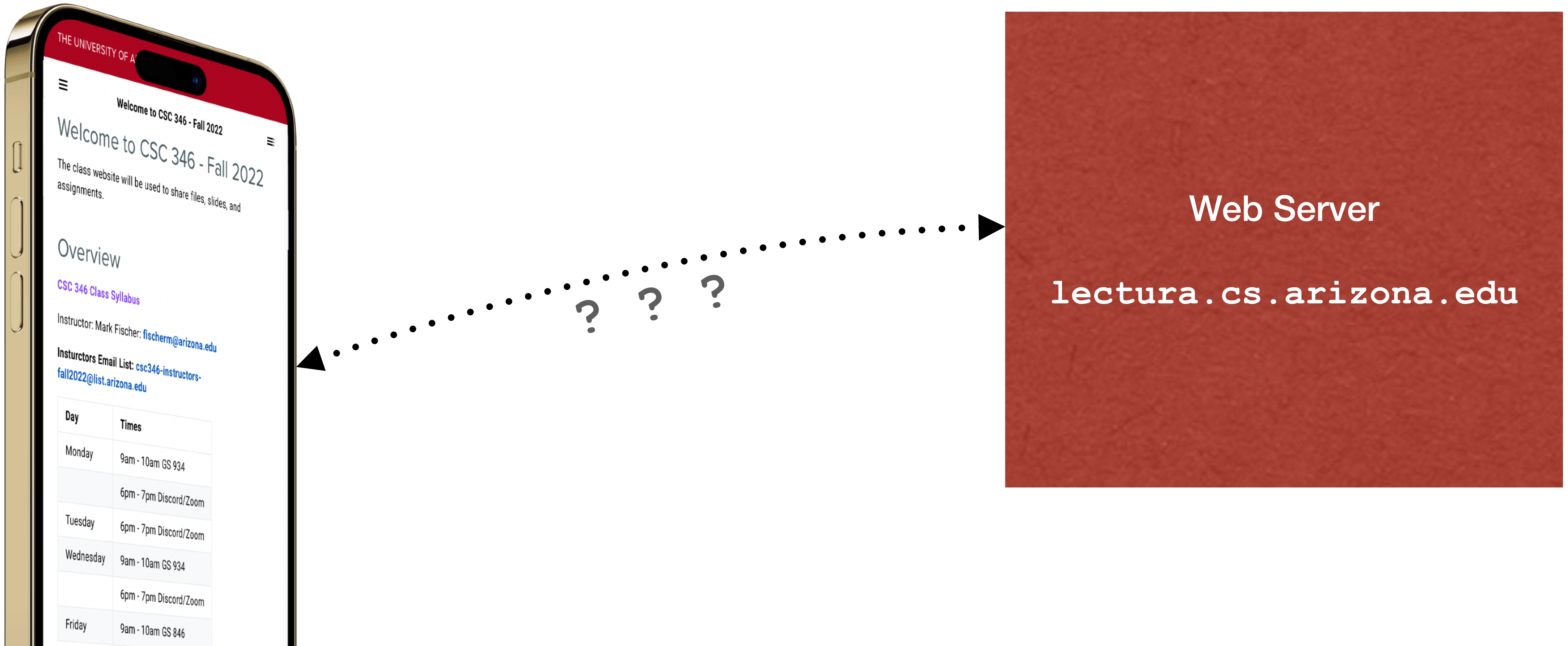
CSC 346 - Cloud Computing

04 - Web Servers, Ports & Sockets

Networking

Sockets

- How do things communicate over the internet? (the simple version)
- This is not a networking class 🤪



Welcome to CSC 346 - Fall 2022

Welcome to CSC 346 - Fall 2022

The class website will be used to share files, slides, and assignments.

Overview

CSC 346 Class Syllabus

Instructor: Mark Fischer: fischerm@arizona.edu

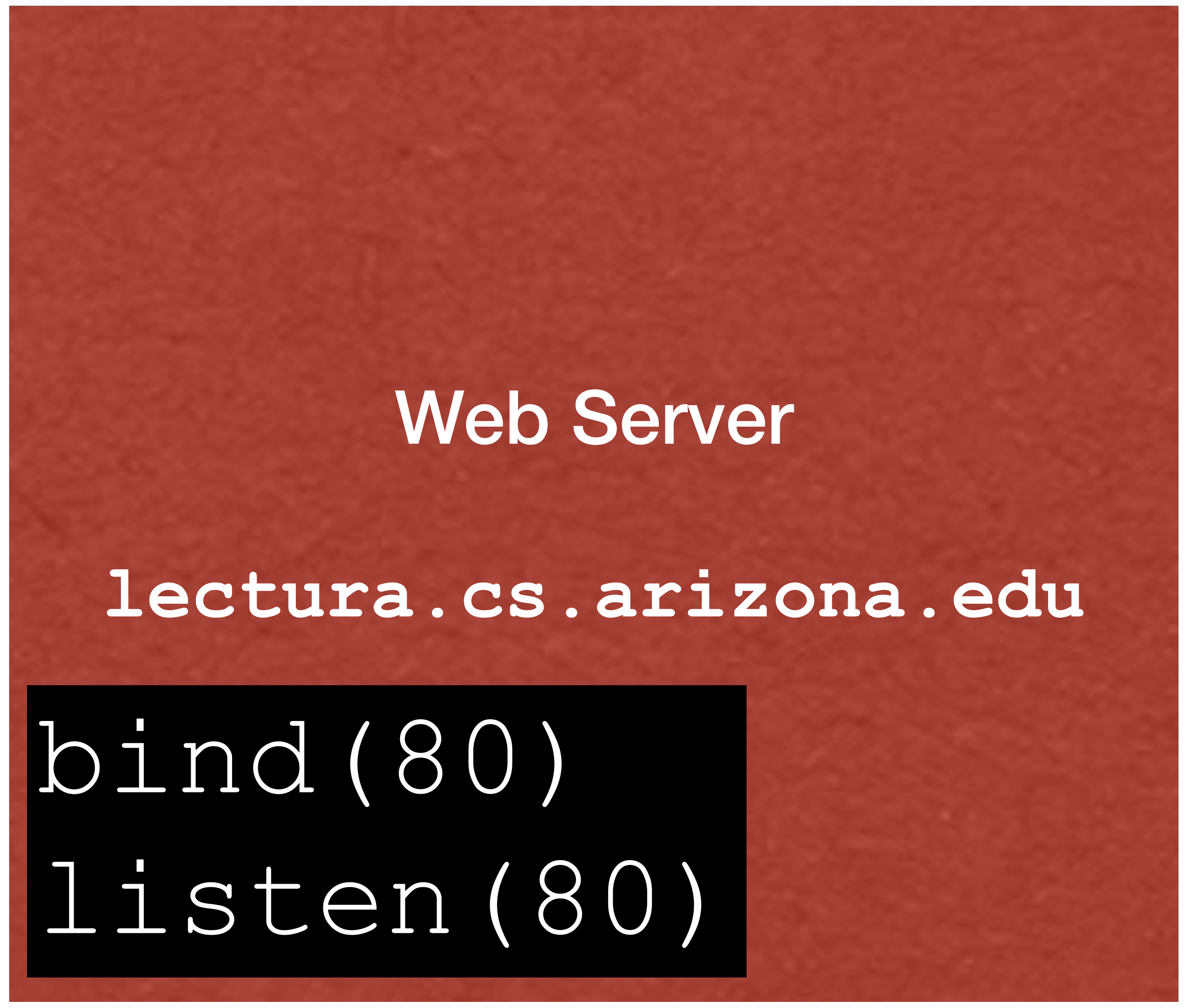
Insturctors Email List: csc346-instructors-fall2022@list.arizona.edu

Day	Times
Monday	9am - 10am GS 934
	6pm - 7pm Discord/Zoom
Tuesday	6pm - 7pm Discord/Zoom
Wednesday	9am - 10am GS 934
	6pm - 7pm Discord/Zoom
Friday	9am - 10am GS 846

Networking

Sockets

- Some computing resource must ***bind*** to a specific ***port*** on its host, and then ***listen*** for incoming connections
- Listens on a specific ***port***
- For a HTTP, this software is our web server
- Since a bind must always precede a listen, we will typically omit the bind in our descriptions
- Most socket libraries will take care of this for you

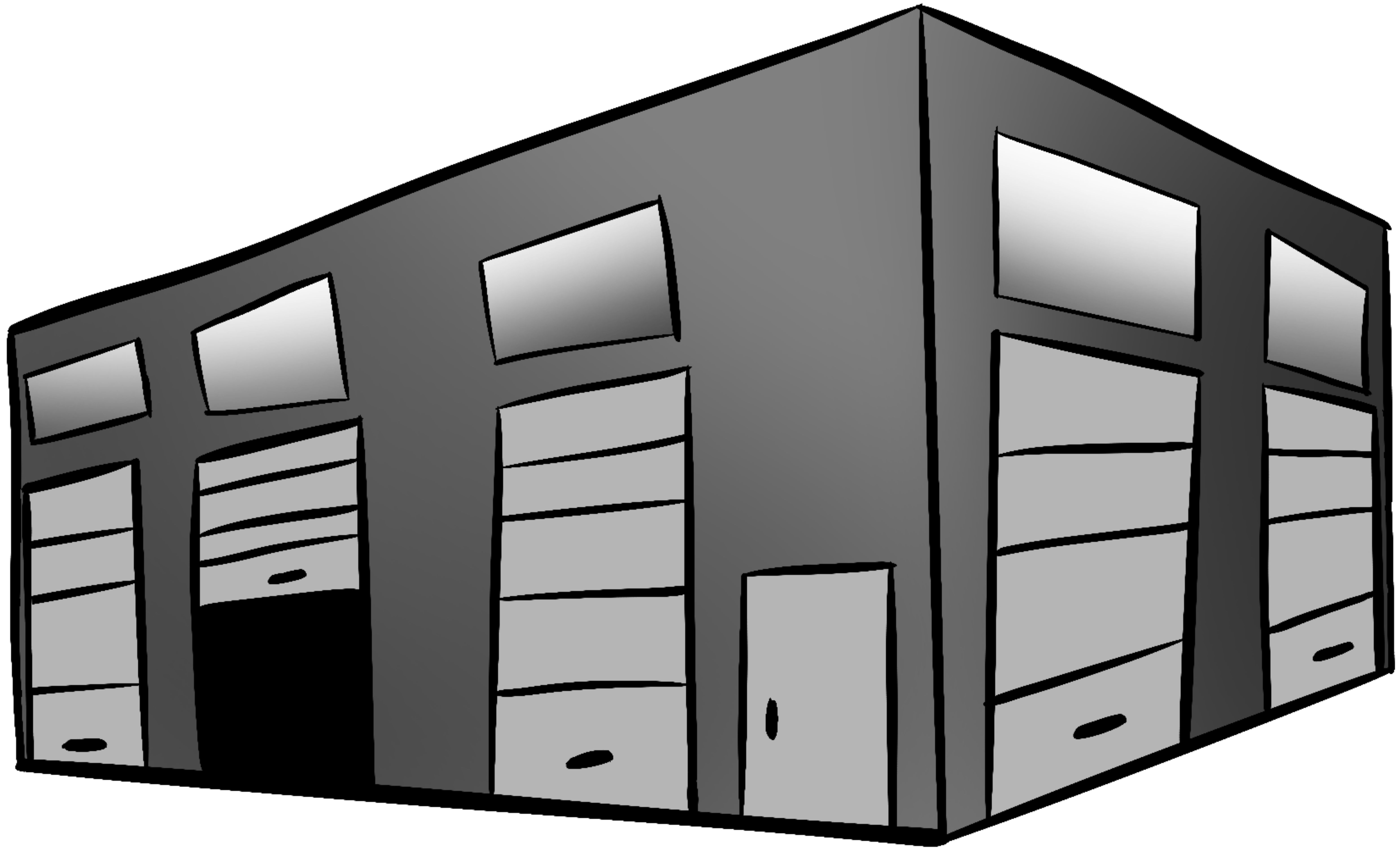


```
bind(80)
listen(80)
```

Networking Ports

What's a Port?

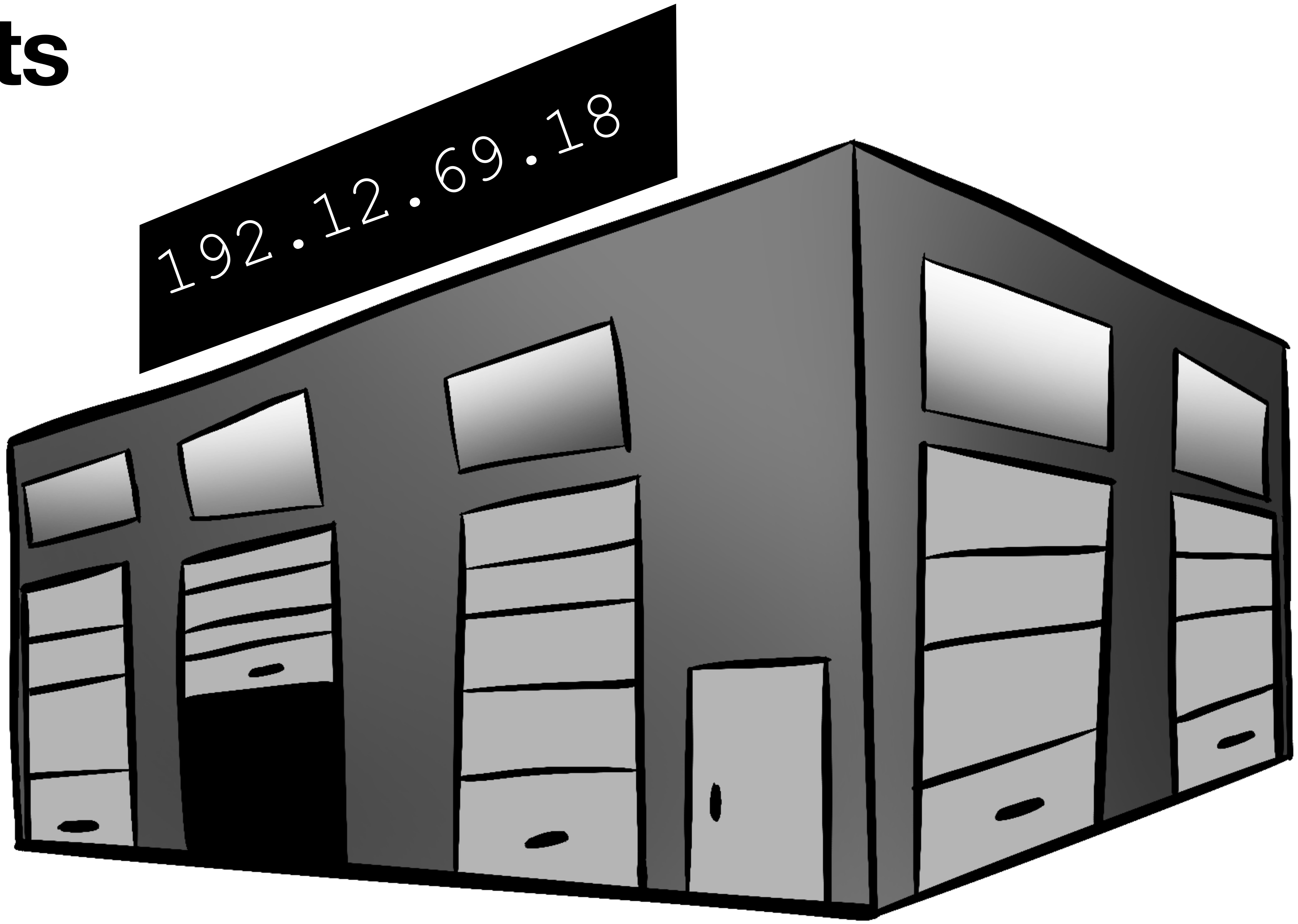
- It's basically a door
  - Italian: *Porta*
  - French: *Porte*
  - Spanish: *Puerta*
- I like to think of a port as a door to a building.



Networking Ports

What's a Port?

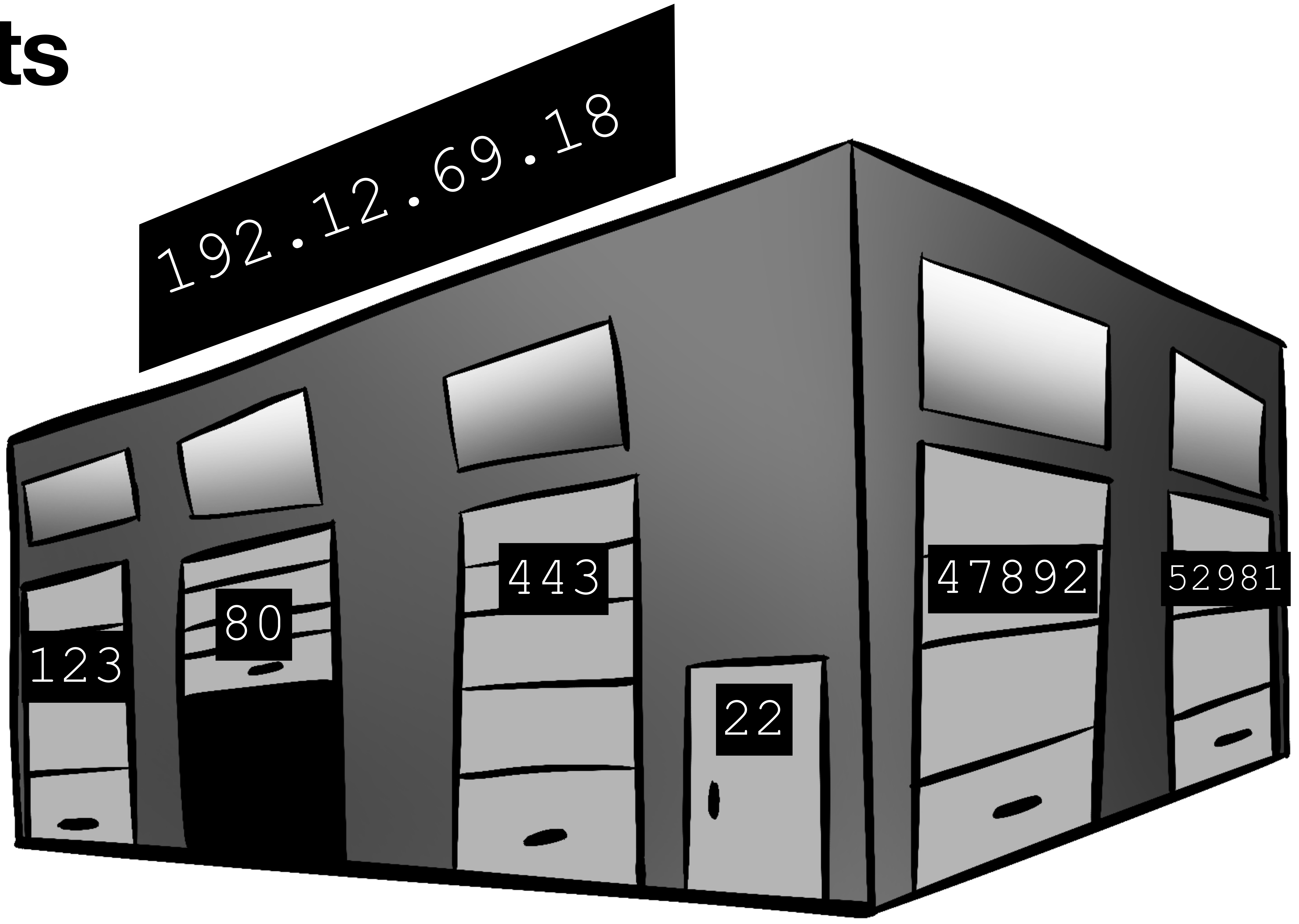
- If we have some device on the internet with an IP address assigned to it, we can think of that as a building.
- A port then can be thought of as a door to the building.
- Doors can let stuff in or out.



Networking Ports

What's a Port?

- Each port has a number
  - 16 bit unsigned integers
  - 0 - 65535
- Internet Assigned Numbers Authority (IANA) has designated different port ranges for different thing, but there's nothing stopping you from using them for whatever



Networking Ports

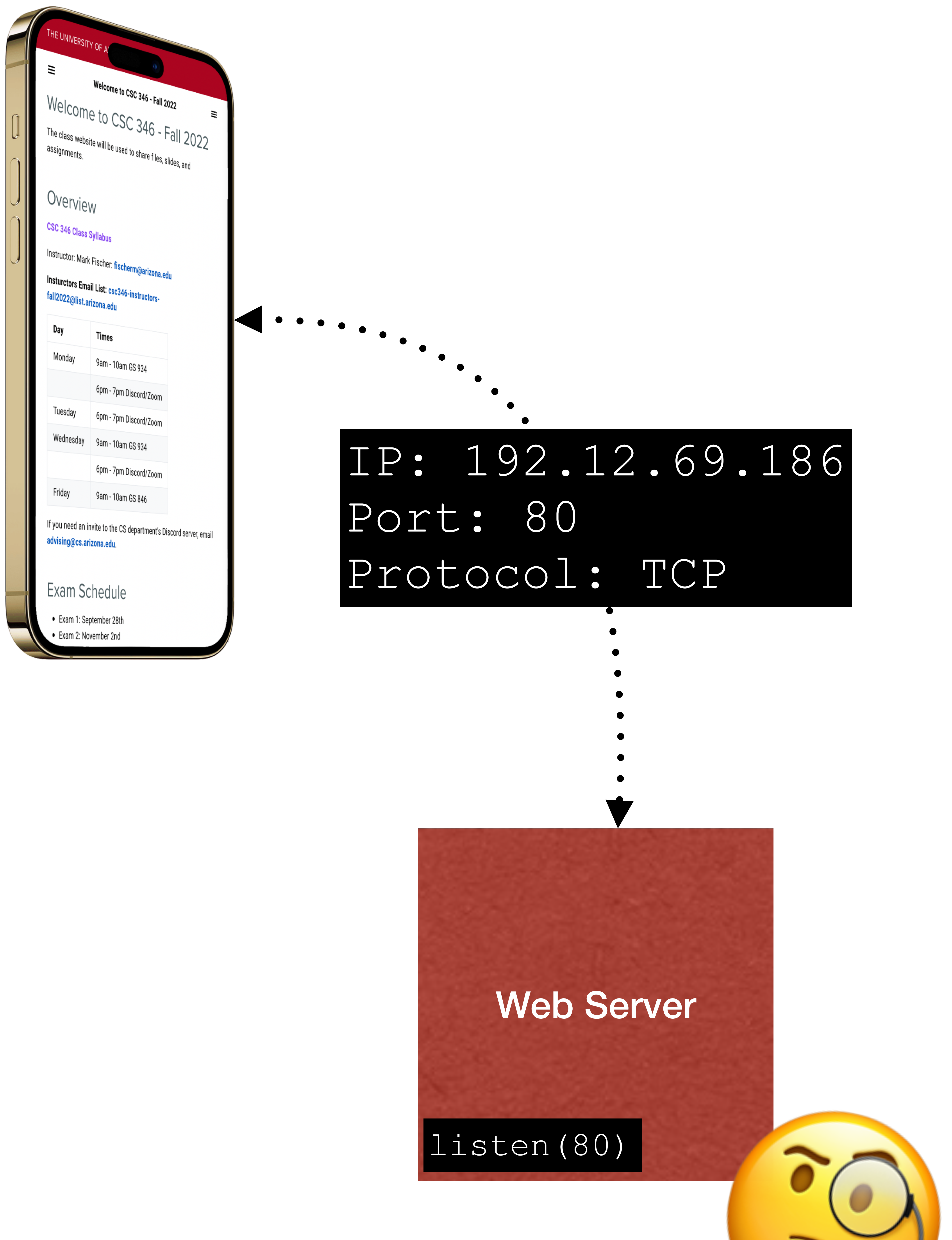
Common Ports

Port Number	Application
`22`	ssh - Secure Shell
`23`	Telnet (unsecure)
`25`	SMTP - Simple Mail Transport Protocol (unsecure)
`80`	HTTP - HyperText Transport Protocol (unsecure)
`123`	NTP - Network Time Protocol
`443`	HTTPS - HTTP Secure
`587`	SMTP Secure
`3306`	MySQL
`25565`	Minecraft

Networking

Sockets

- A client then opens a socket to the server
- A socket data stream that sits on top of the network layer provided by the operating system.
- A socket is described by an ***IP address***, a ***port***, and a ***transport protocol***
- For our class, we'll use TCP for our protocol
    - Transmission Control Protocol


THE UNIVERSITY OF A

Welcome to CSC 346 - Fall 2022

Welcome to CSC 346 - Fall 2022

The class website will be used to share files, slides, and assignments.

Overview

CSC 346 Class Syllabus

Instructor: Mark Fischer: fischerm@arizona.edu

Instructors Email List: csc346-instructors-fall2022@list.arizona.edu

Day	Times
Monday	9am - 10am GS 934
	6pm - 7pm Discord/Zoom
Tuesday	6pm - 7pm Discord/Zoom
Wednesday	9am - 10am GS 934
	6pm - 7pm Discord/Zoom
Friday	9am - 10am GS 846

If you need an invite to the CS department's Discord server, email advising@cs.arizona.edu.

Exam Schedule

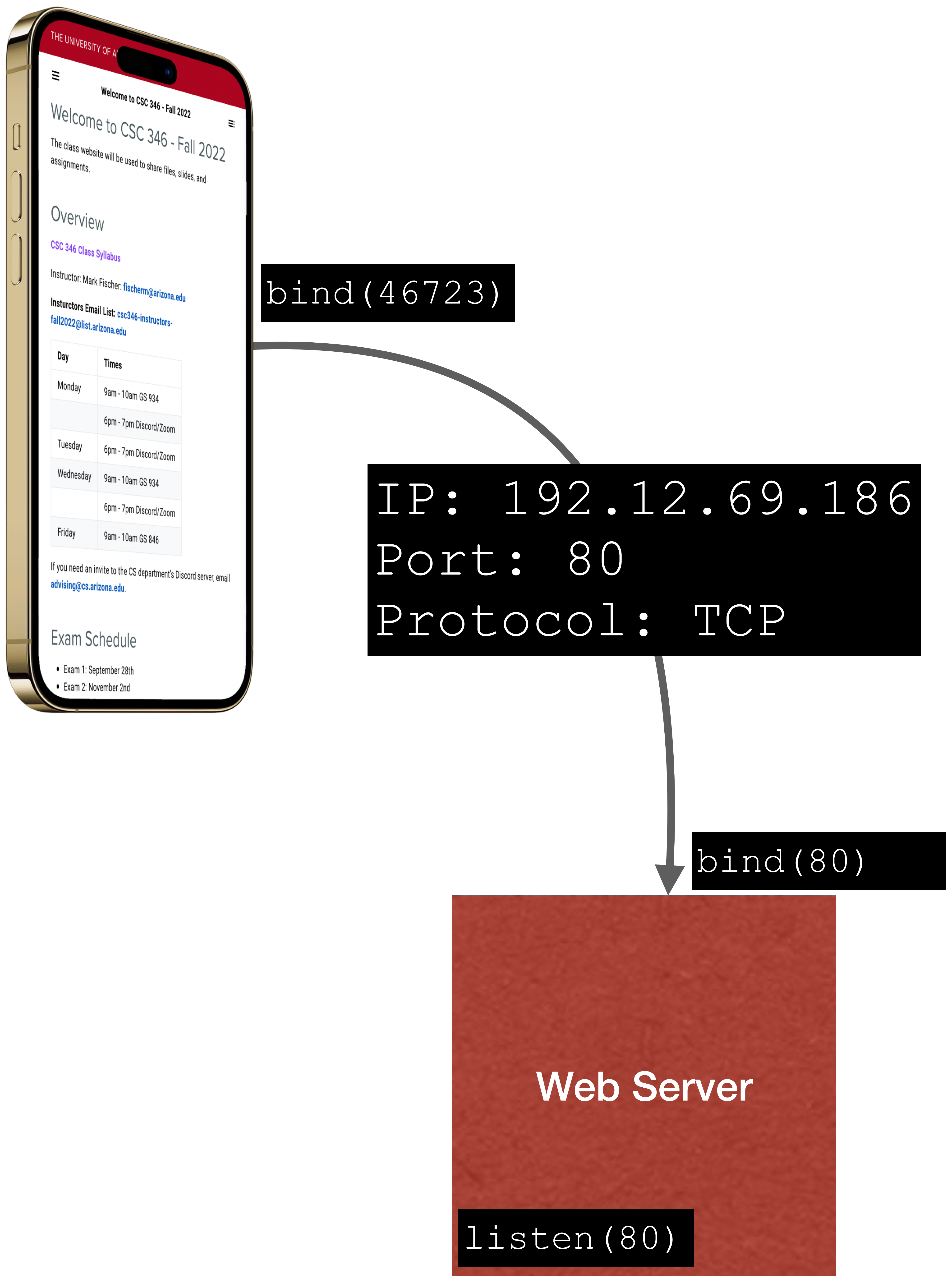
- Exam 1: September 28th
- Exam 2: November 2nd

```
IP: 192.12.69.186
Port: 80
Protocol: TCP
```

Networking

Sockets

- Both sides must ***bind*** to a port
- The server binds to the well known port 80, since the clients need to know this
- The client typically uses a random high number available port
- As part of the socket connection, the client tells the server what port it is using



THE UNIVERSITY OF A

Welcome to CSC 346 - Fall 2022

Welcome to CSC 346 - Fall 2022

The class website will be used to share files, slides, and assignments.

Overview

CSC 346 Class Syllabus

Instructor: Mark Fischer: fischerm@arizona.edu

Insturctors Email List: csc346-instructors-fall2022@list.arizona.edu

Day	Times
Monday	9am - 10am GS 934
	6pm - 7pm Discord/Zoom
Tuesday	6pm - 7pm Discord/Zoom
Wednesday	9am - 10am GS 934
	6pm - 7pm Discord/Zoom
Friday	9am - 10am GS 846

If you need an invite to the CS department's Discord server, email advising@cs.arizona.edu.

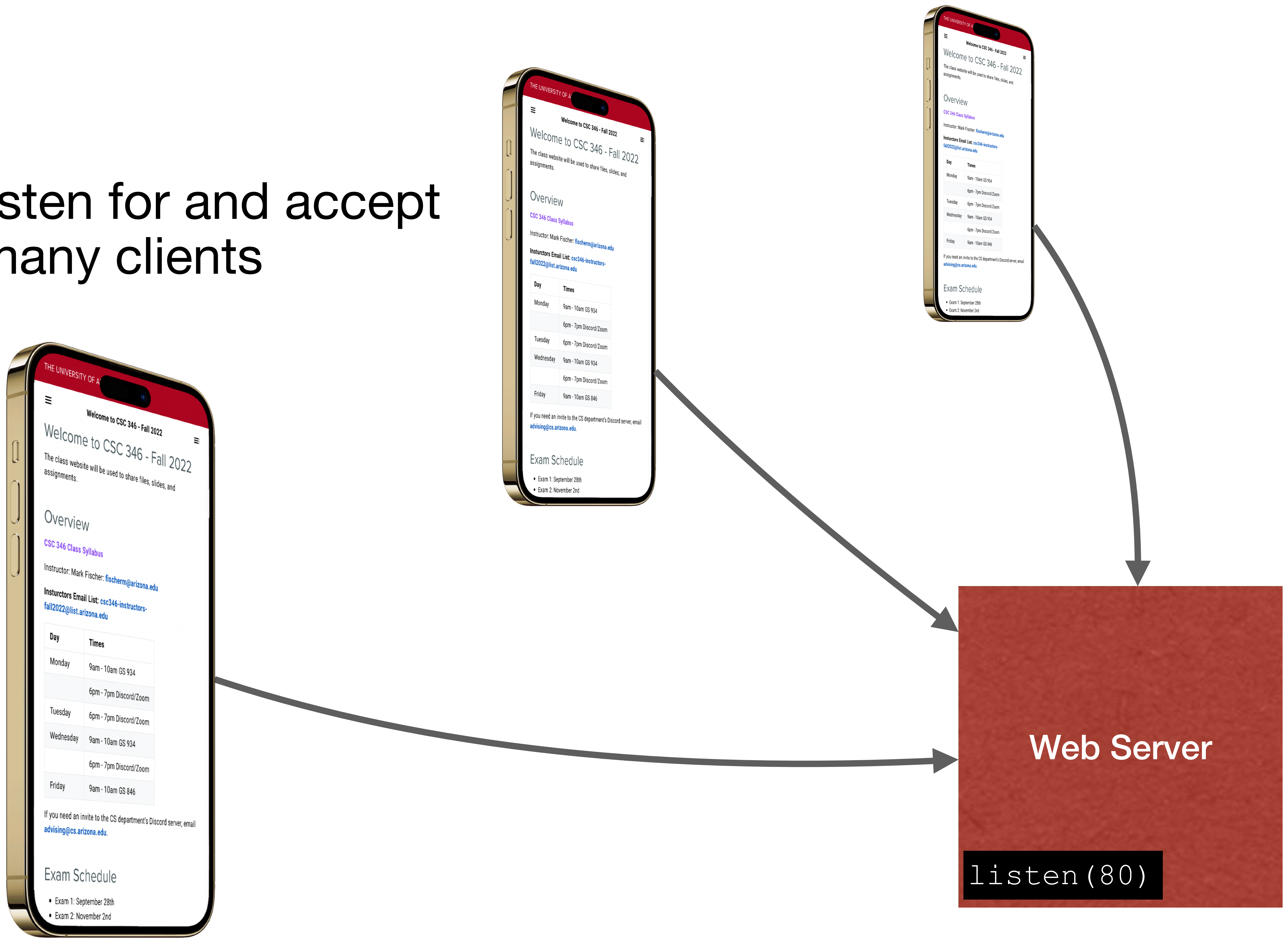
Exam Schedule

- Exam 1: September 28th
- Exam 2: November 2nd

Networking

Sockets

- A web server can listen for and accept connections from many clients

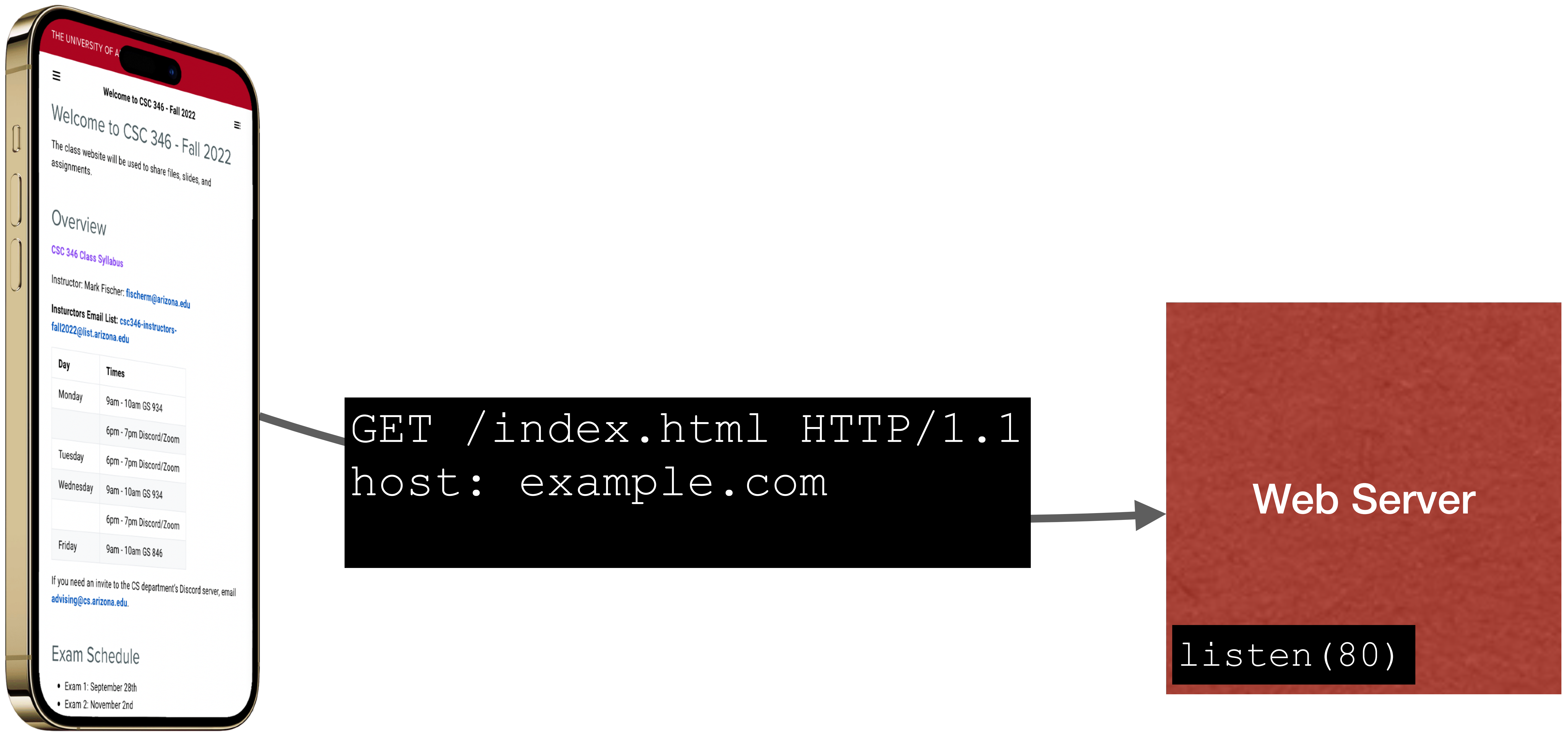


Networking

Sockets

- Once a socket is connected, the client and server can exchange data according to whatever protocol the server supports.

- For web servers,
  this is HTTP



THE UNIVERSITY OF A

Welcome to CSC 346 - Fall 2022

Welcome to CSC 346 - Fall 2022

The class website will be used to share files, slides, and assignments.

Overview

CSC 346 Class Syllabus

Instructor: Mark Fischer: fischerm@arizona.edu

Instructors Email List: csc346-instructors-fall2022@list.arizona.edu

Day	Times
Monday	9am - 10am GS 934
	6pm - 7pm Discord/Zoom
Tuesday	6pm - 7pm Discord/Zoom
Wednesday	9am - 10am GS 934
	6pm - 7pm Discord/Zoom
Friday	9am - 10am GS 846

If you need an invite to the CS department's Discord server, email advising@cs.arizona.edu.

Exam Schedule

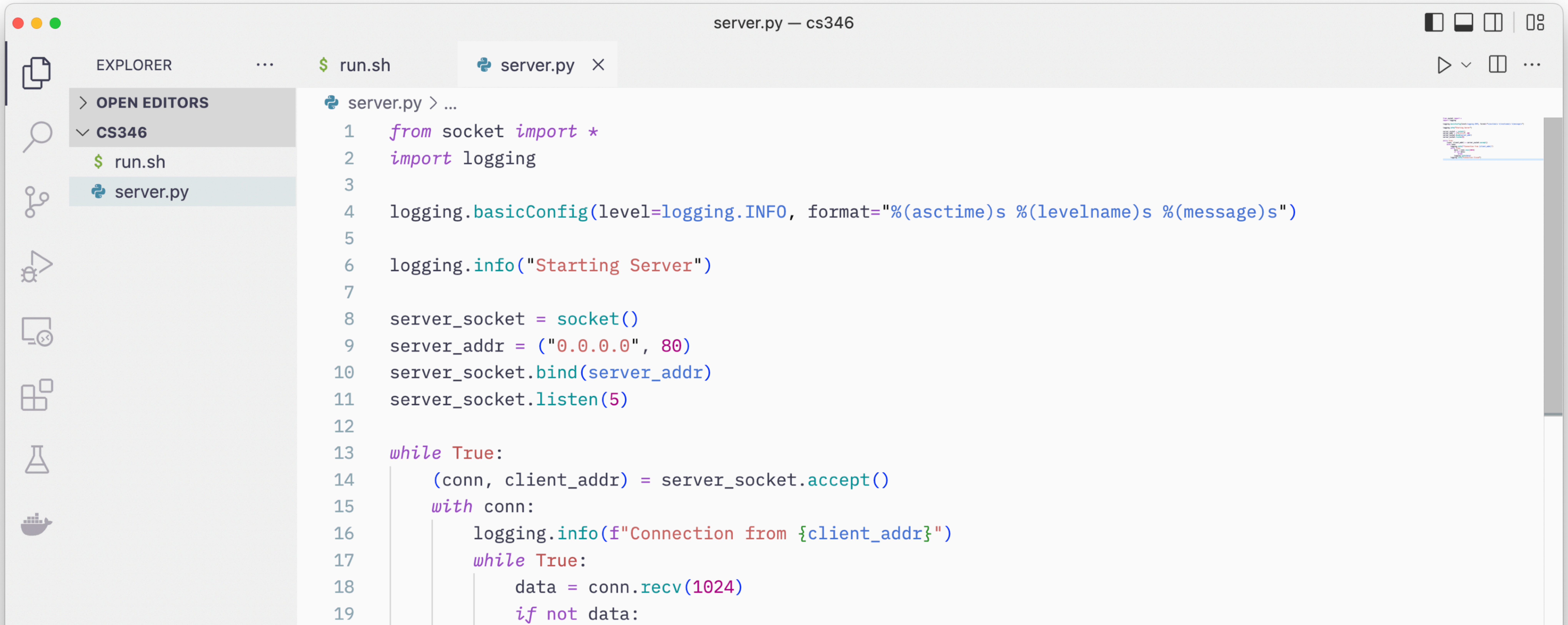
- Exam 1: September 28th
- Exam 2: November 2nd

```
GET /index.html HTTP/1.1
host: example.com
```

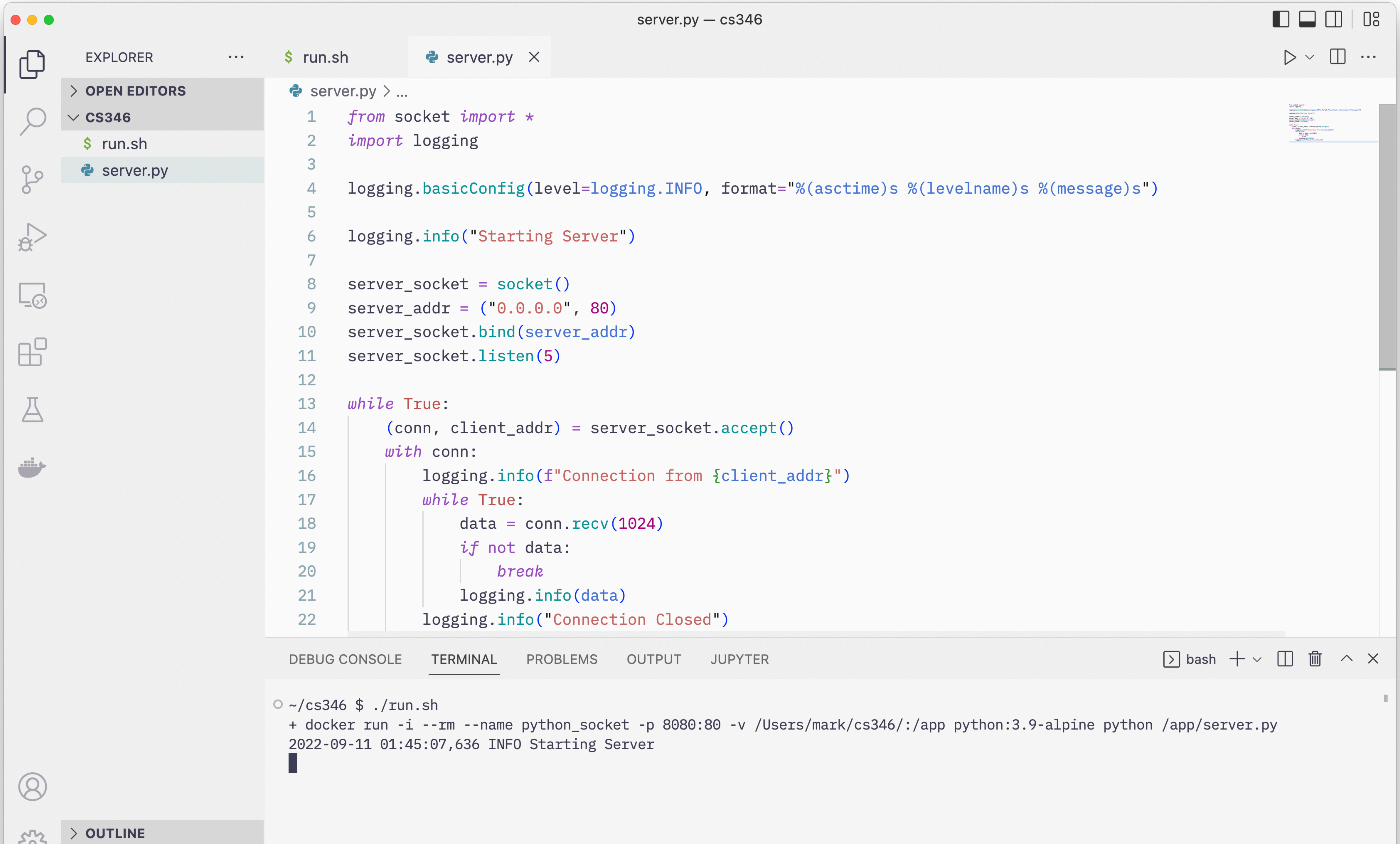
```
listen(80)
```

Echo Server

The world’s worst web server

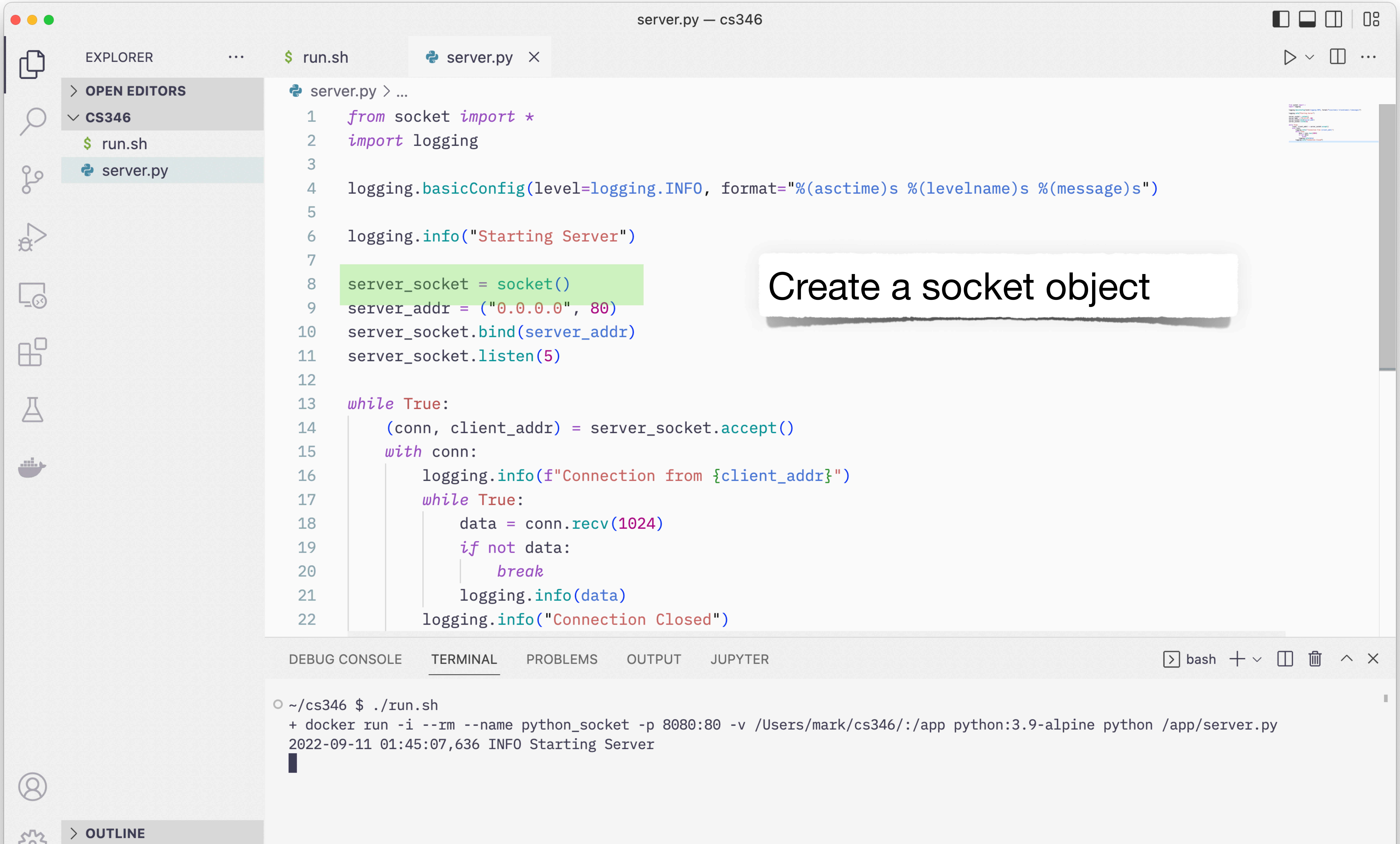


```python
from socket import *
import logging

logging.basicConfig(level=logging.INFO, format="%(asctime)s %(levelname)s %(message)s")

logging.info("Starting Server")

server_socket = socket()
server_addr = ("0.0.0.0", 80)
server_socket.bind(server_addr)
server_socket.listen(5)

while True:
    (conn, client_addr) = server_socket.accept()
    with conn:
        logging.info(f"Connection from {client_addr}")
        while True:
            data = conn.recv(1024)
            if not data:
```



```python
from socket import *
import logging

logging.basicConfig(level=logging.INFO, format="%(asctime)s %(levelname)s %(message)s")

logging.info("Starting Server")

server_socket = socket()
server_addr = ("0.0.0.0", 80)
server_socket.bind(server_addr)
server_socket.listen(5)

while True:
    (conn, client_addr) = server_socket.accept()
    with conn:
        logging.info(f"Connection from {client_addr}")
        while True:
            data = conn.recv(1024)
            if not data:
                break
            logging.info(data)
        logging.info("Connection Closed")
```

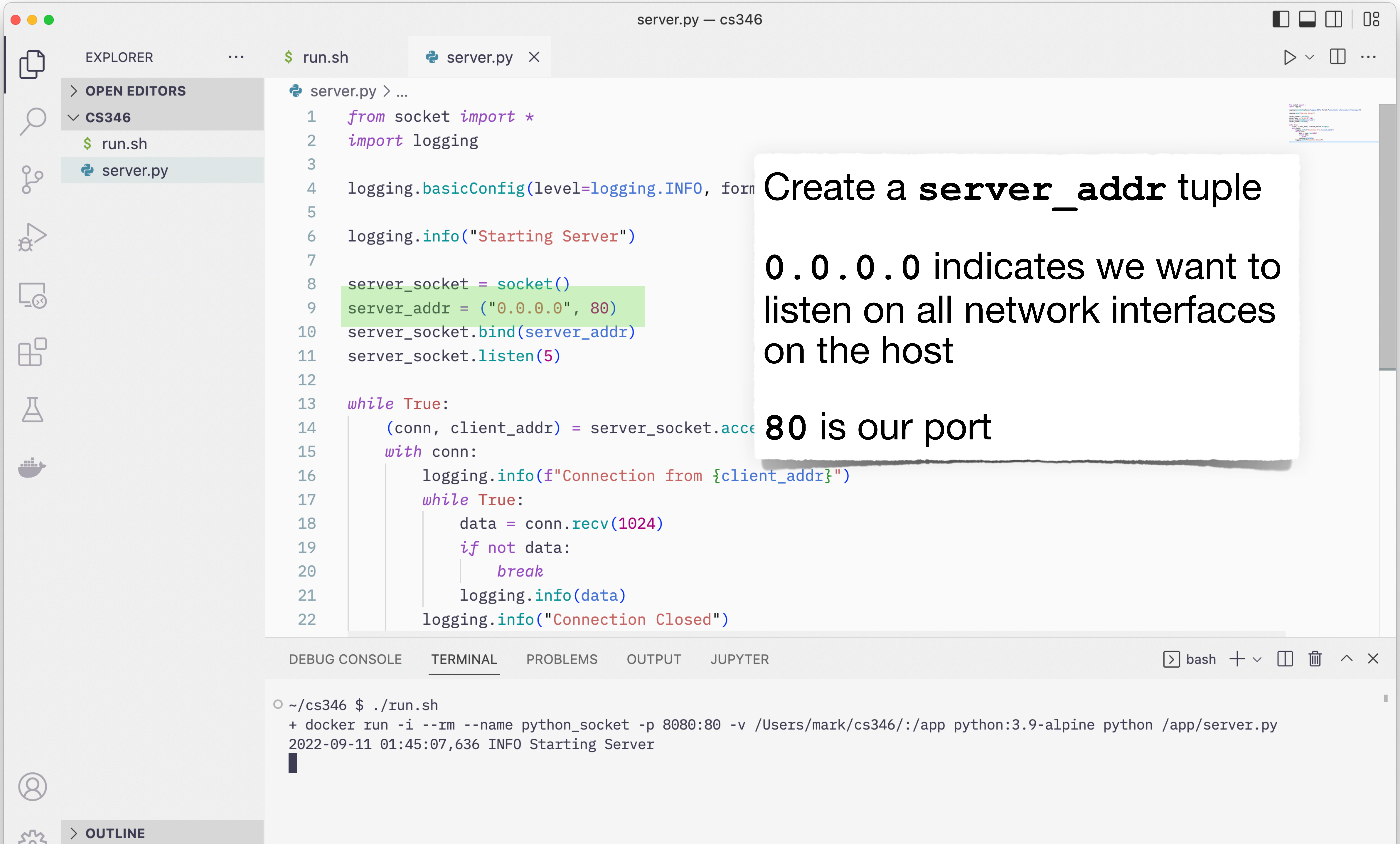
```
~/cs346 $ ./run.sh
+ docker run -i --rm --name python_socket -p 8080:80 -v /Users/mark/cs346/:/app python:3.9-alpine python /app/server.py
2022-09-11 01:45:07,636 INFO Starting Server
```



```python
from socket import *
import logging

logging.basicConfig(level=logging.INFO, format="%(asctime)s %(levelname)s %(message)s")

logging.info("Starting Server")

server_socket = socket()
server_addr = ("0.0.0.0", 80)
server_socket.bind(server_addr)
server_socket.listen(5)

while True:
    (conn, client_addr) = server_socket.accept()
    with conn:
        logging.info(f"Connection from {client_addr}")
        while True:
            data = conn.recv(1024)
            if not data:
                break
            logging.info(data)
        logging.info("Connection Closed")
```

Create a socket object

```
~/cs346 $ ./run.sh
+ docker run -i --rm --name python_socket -p 8080:80 -v /Users/mark/cs346/:/app python:3.9-alpine python /app/server.py
2022-09-11 01:45:07,636 INFO Starting Server
```



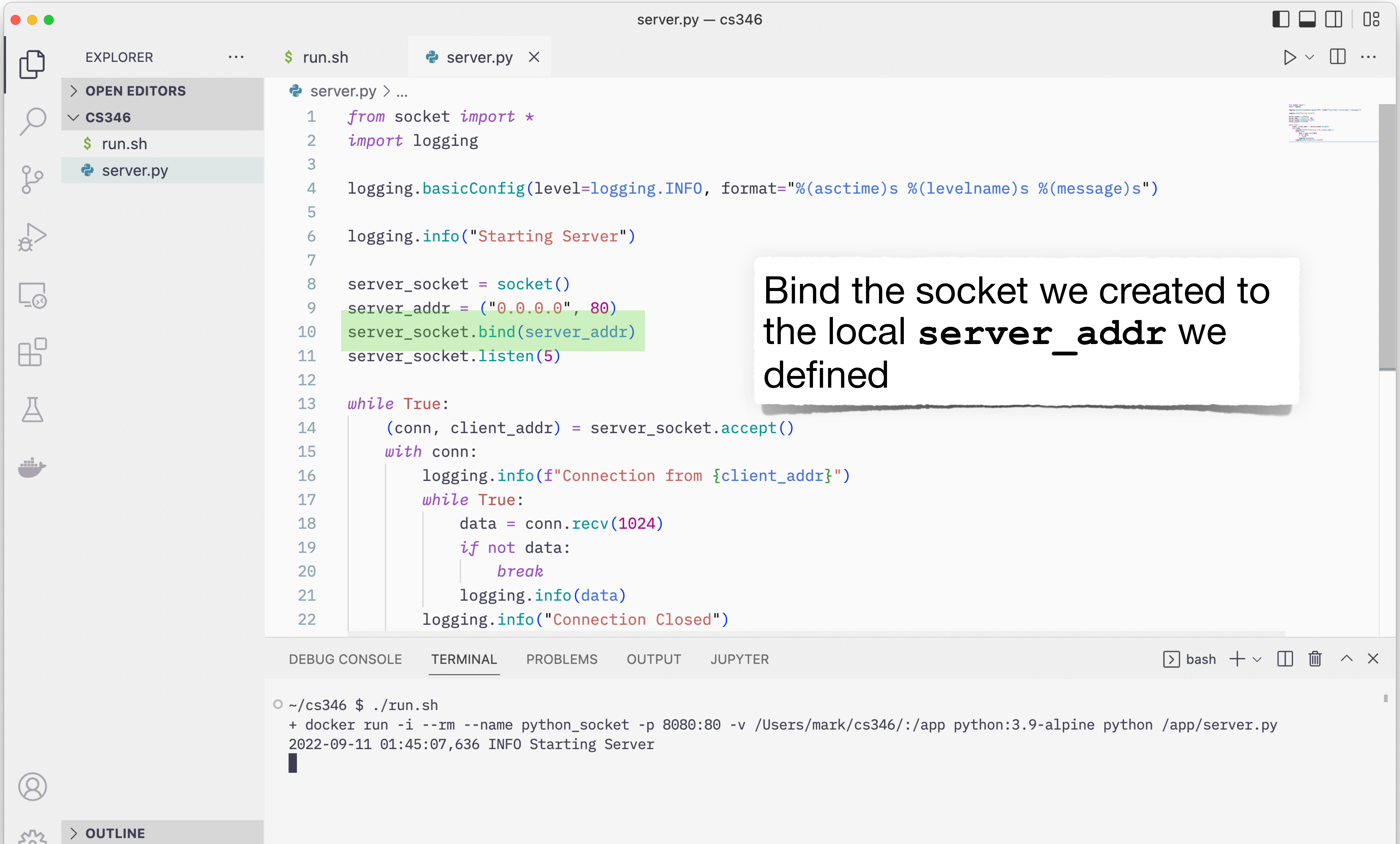
```python
from socket import *
import logging

logging.basicConfig(level=logging.INFO, form

logging.info("Starting Server")

server_socket = socket()
server_addr = ("0.0.0.0", 80)
server_socket.bind(server_addr)
server_socket.listen(5)

while True:
    (conn, client_addr) = server_socket.acce
    with conn:
        logging.info(f"Connection from {client_addr}")
        while True:
            data = conn.recv(1024)
            if not data:
                break
            logging.info(data)
        logging.info("Connection Closed")
```

```
~/cs346 $ ./run.sh
+ docker run -i --rm --name python_socket -p 8080:80 -v /Users/mark/cs346/:/app python:3.9-alpine python /app/server.py
2022-09-11 01:45:07,636 INFO Starting Server
```

Create a **`server_addr`** tuple

`0.0.0.0` indicates we want to listen on all network interfaces on the host

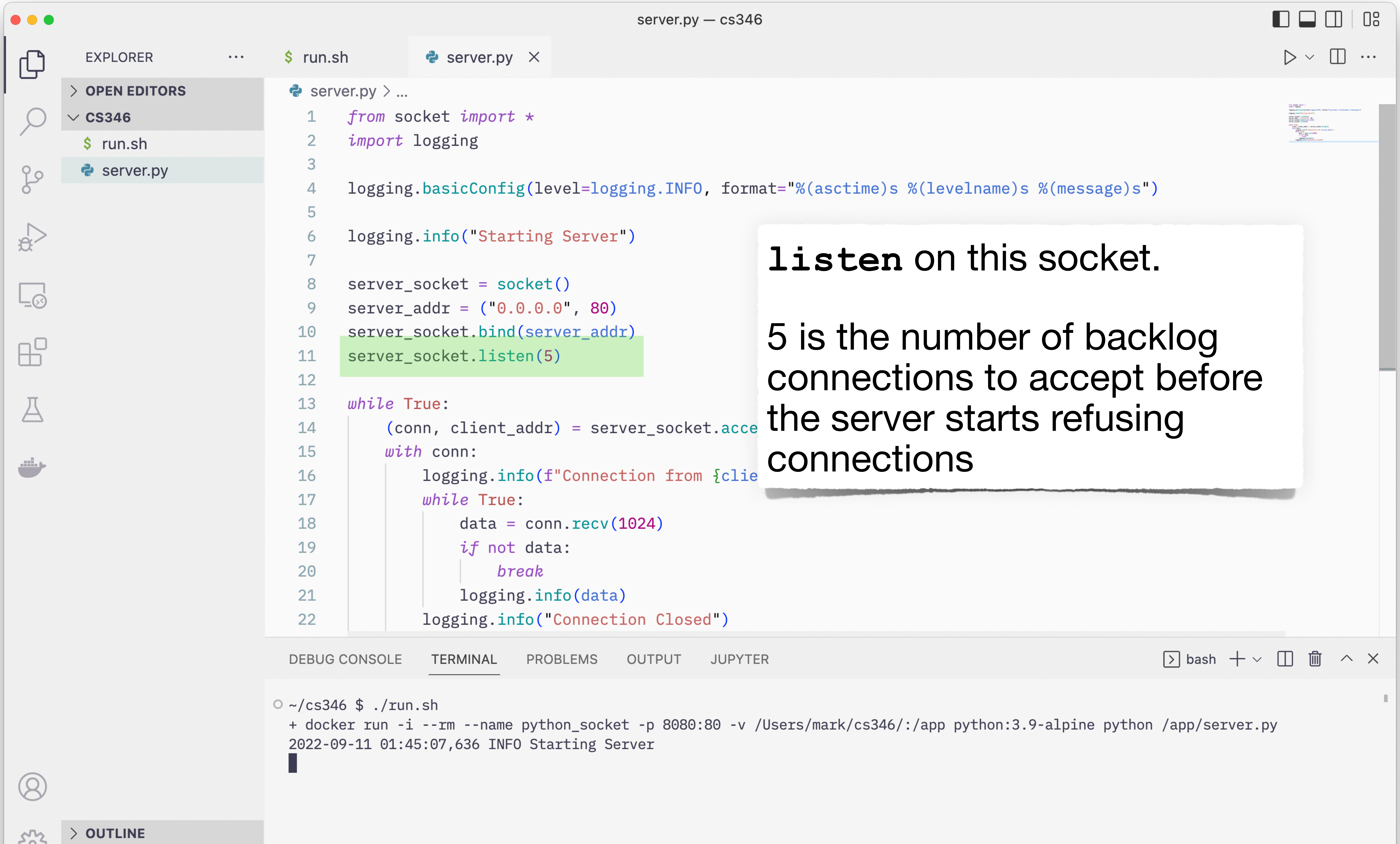
`80` is our port



```python
from socket import *
import logging

logging.basicConfig(level=logging.INFO, format="%(asctime)s %(levelname)s %(message)s")

logging.info("Starting Server")

server_socket = socket()
server_addr = ("0.0.0.0", 80)
server_socket.bind(server_addr)
server_socket.listen(5)

while True:
    (conn, client_addr) = server_socket.accept()
    with conn:
        logging.info(f"Connection from {client_addr}")
        while True:
            data = conn.recv(1024)
            if not data:
                break
            logging.info(data)
        logging.info("Connection Closed")
```

Bind the socket we created to the local **`server_addr`** we defined

```
~/cs346 $ ./run.sh
+ docker run -i --rm --name python_socket -p 8080:80 -v /Users/mark/cs346/:/app python:3.9-alpine python /app/server.py
2022-09-11 01:45:07,636 INFO Starting Server
```

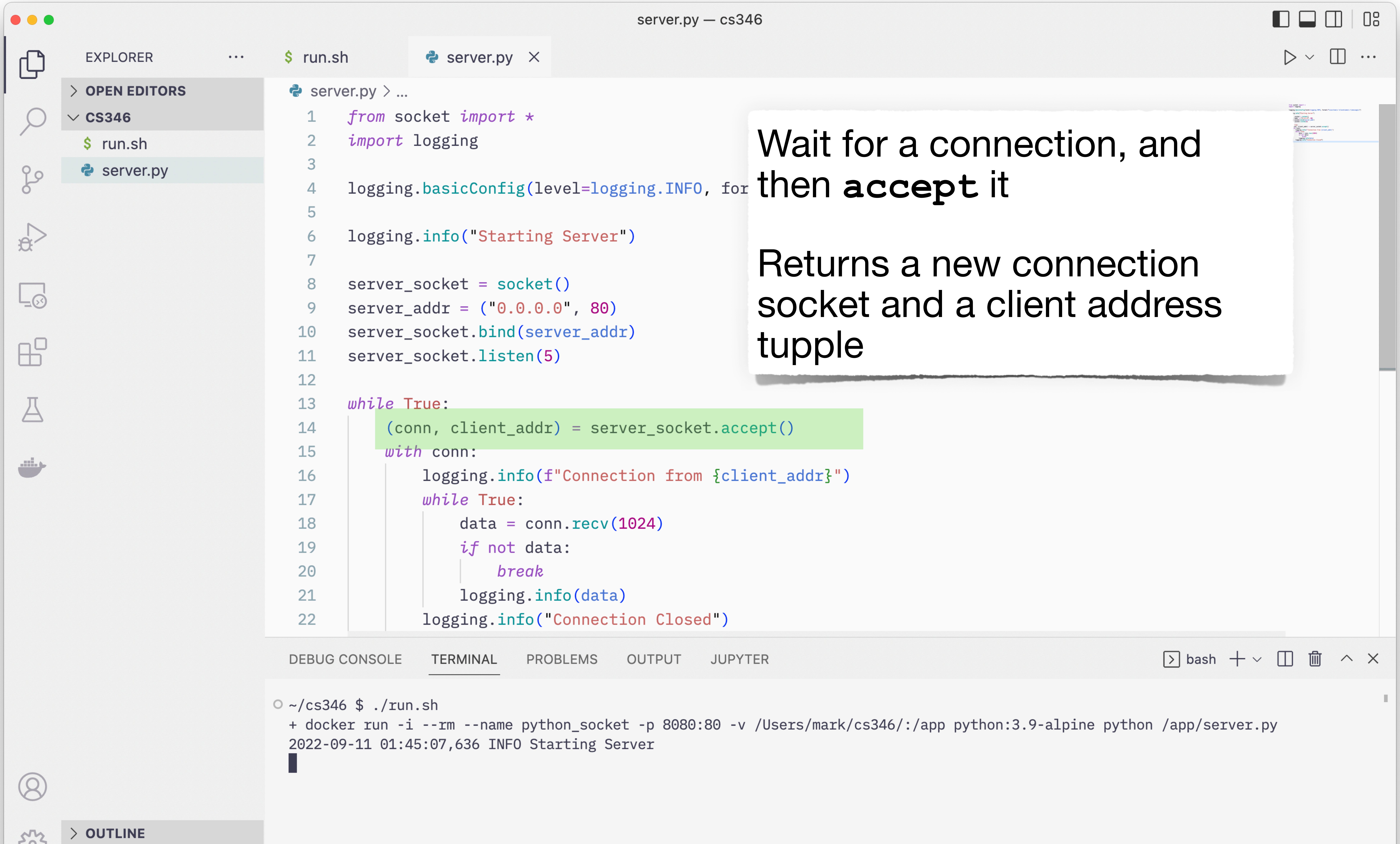


```python
from socket import *
import logging

logging.basicConfig(level=logging.INFO, format="%(asctime)s %(levelname)s %(message)s")

logging.info("Starting Server")

server_socket = socket()
server_addr = ("0.0.0.0", 80)
server_socket.bind(server_addr)
server_socket.listen(5)

while True:
    (conn, client_addr) = server_socket.acce
    with conn:
        logging.info(f"Connection from {clie
        while True:
            data = conn.recv(1024)
            if not data:
                break
            logging.info(data)
        logging.info("Connection Closed")
```

`listen` on this socket.

5 is the number of backlog connections to accept before the server starts refusing connections

```
~/cs346 $ ./run.sh
+ docker run -i --rm --name python_socket -p 8080:80 -v /Users/mark/cs346/:/app python:3.9-alpine python /app/server.py
2022-09-11 01:45:07,636 INFO Starting Server
```

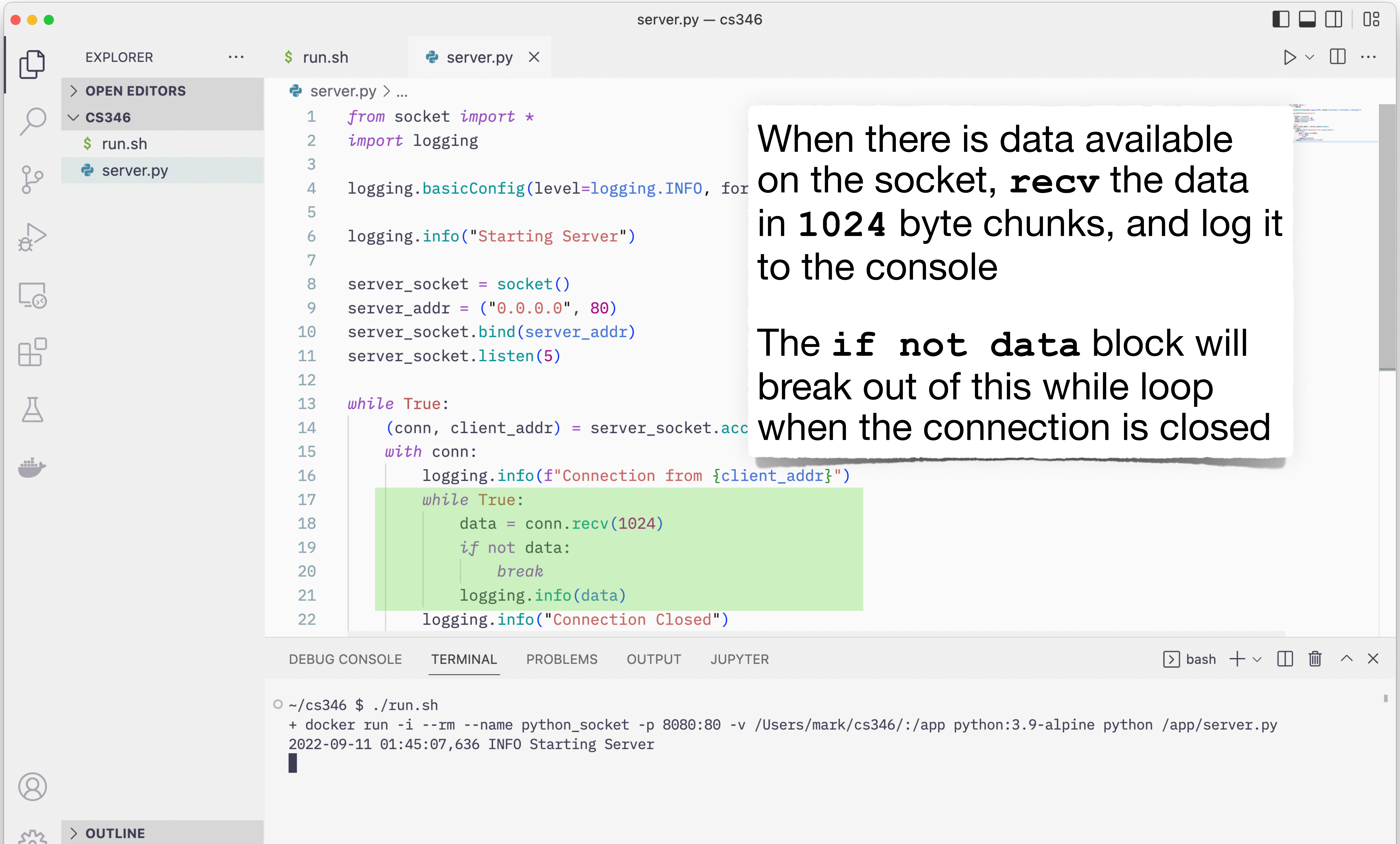


```python
from socket import *
import logging

logging.basicConfig(level=logging.INFO, for

logging.info("Starting Server")

server_socket = socket()
server_addr = ("0.0.0.0", 80)
server_socket.bind(server_addr)
server_socket.listen(5)

while True:
    (conn, client_addr) = server_socket.accept()
    with conn:
        logging.info(f"Connection from {client_addr}")
        while True:
            data = conn.recv(1024)
            if not data:
                break
            logging.info(data)
        logging.info("Connection Closed")
```

Wait for a connection, and then **`accept`** it

Returns a new connection socket and a client address tupple

```
~/cs346 $ ./run.sh
+ docker run -i --rm --name python_socket -p 8080:80 -v /Users/mark/cs346/:/app python:3.9-alpine python /app/server.py
2022-09-11 01:45:07,636 INFO Starting Server
```

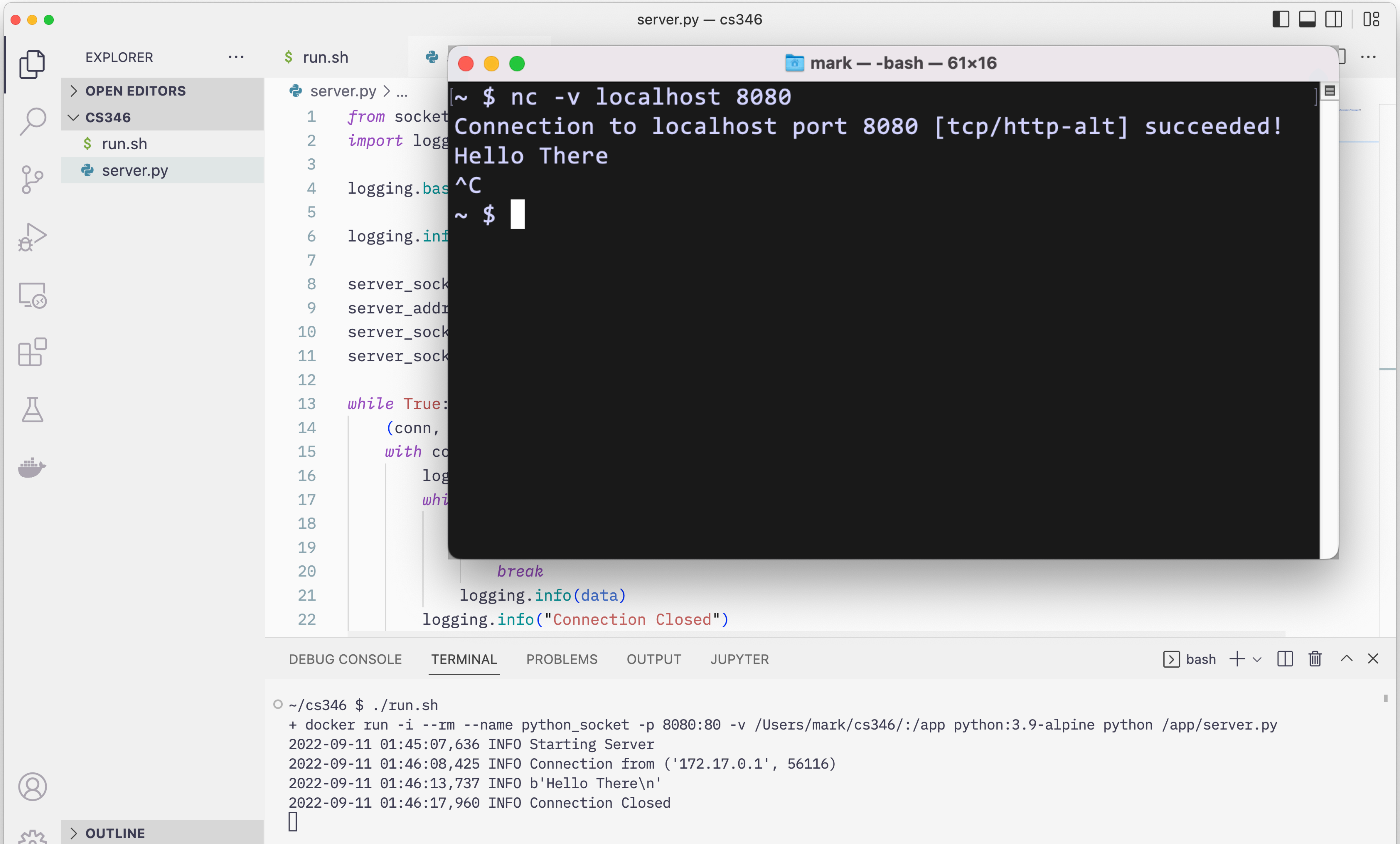


```python
from socket import *
import logging

logging.basicConfig(level=logging.INFO, for

logging.info("Starting Server")

server_socket = socket()
server_addr = ("0.0.0.0", 80)
server_socket.bind(server_addr)
server_socket.listen(5)

while True:
    (conn, client_addr) = server_socket.acc
    with conn:
        logging.info(f"Connection from {client_addr}")
        while True:
            data = conn.recv(1024)
            if not data:
                break
            logging.info(data)
        logging.info("Connection Closed")
```

When there is data available on the socket, **`recv`** the data in **`1024`** byte chunks, and log it to the console

The **`if not data`** block will break out of this while loop when the connection is closed

```
~/cs346 $ ./run.sh
+ docker run -i --rm --name python_socket -p 8080:80 -v /Users/mark/cs346/:/app python:3.9-alpine python /app/server.py
2022-09-11 01:45:07,636 INFO Starting Server
```

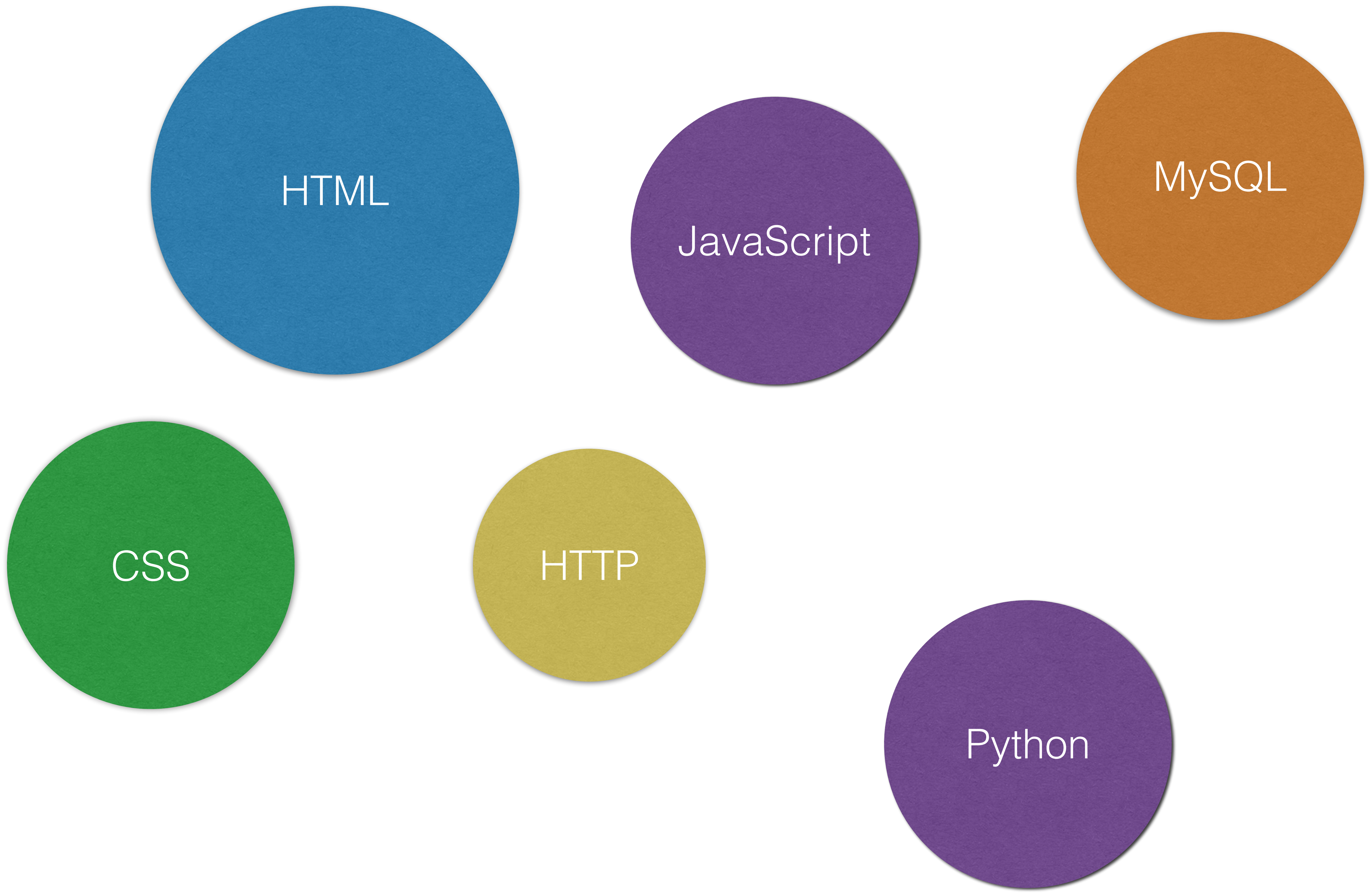


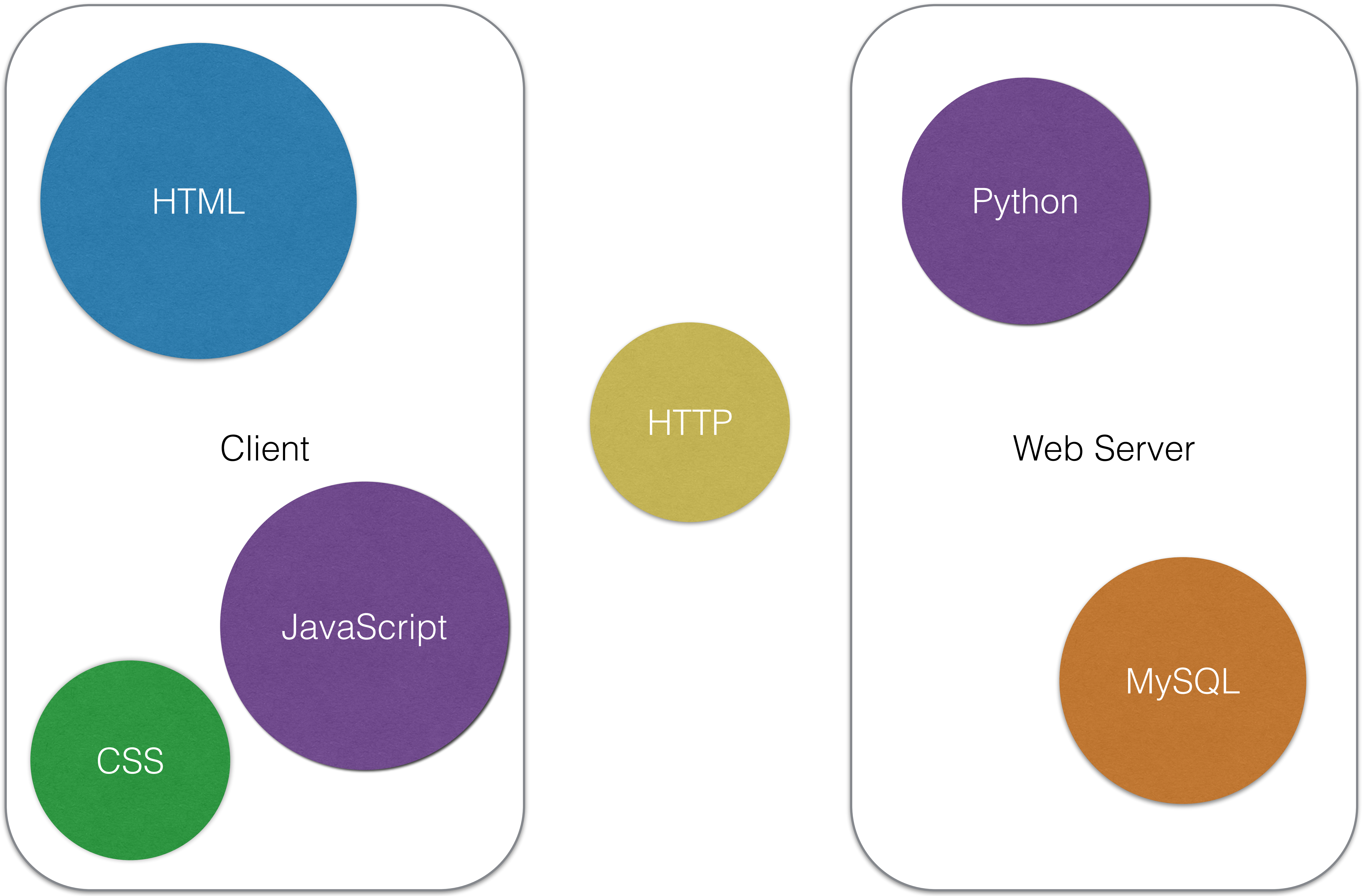
```
from socket
import logg

logging.bas

logging.inf

server_sock
server_addr
server_sock
server_sock

while True:
    (conn,
    with co
        log
        whi


                break
            logging.info(data)
        logging.info("Connection Closed")
```

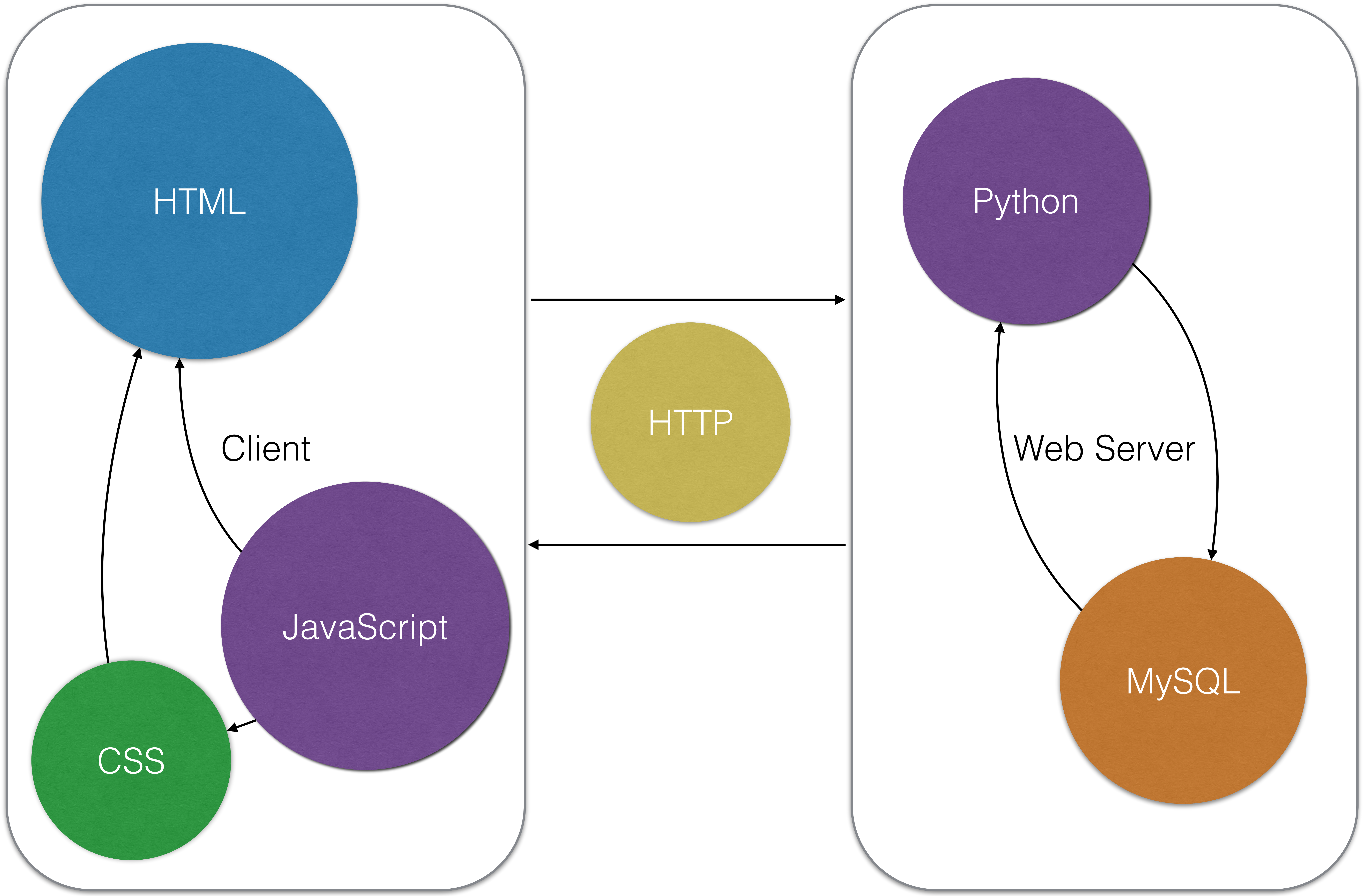
```
~ $ nc -v localhost 8080
Connection to localhost port 8080 [tcp/http-alt] succeeded!
Hello There
^C
~ $
```

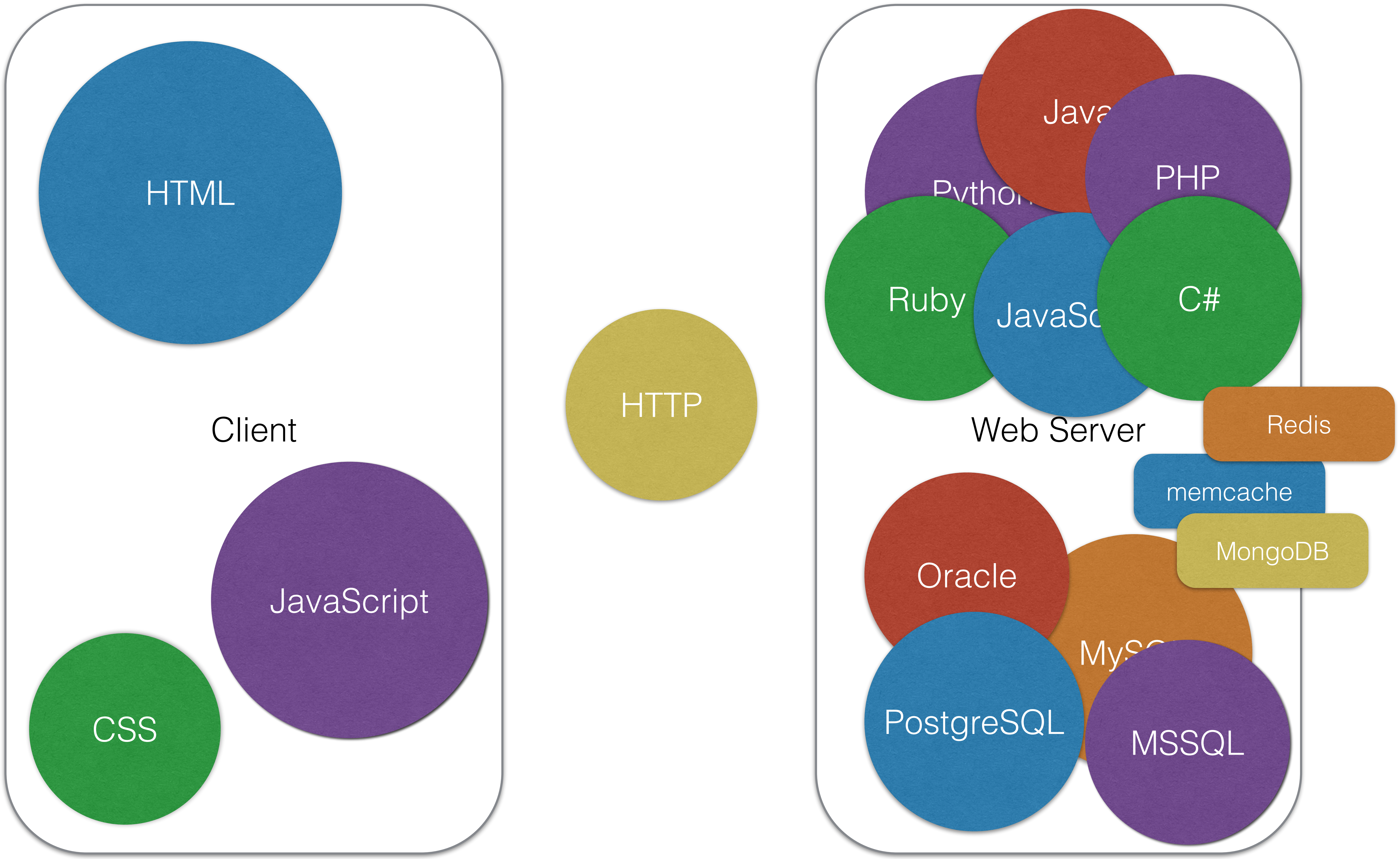
```
~/cs346 $ ./run.sh
+ docker run -i --rm --name python_socket -p 8080:80 -v /Users/mark/cs346/:/app python:3.9-alpine python /app/server.py
2022-09-11 01:45:07,636 INFO Starting Server
2022-09-11 01:46:08,425 INFO Connection from ('172.17.0.1', 56116)
2022-09-11 01:46:13,737 INFO b'Hello There\n'
2022-09-11 01:46:17,960 INFO Connection Closed
```

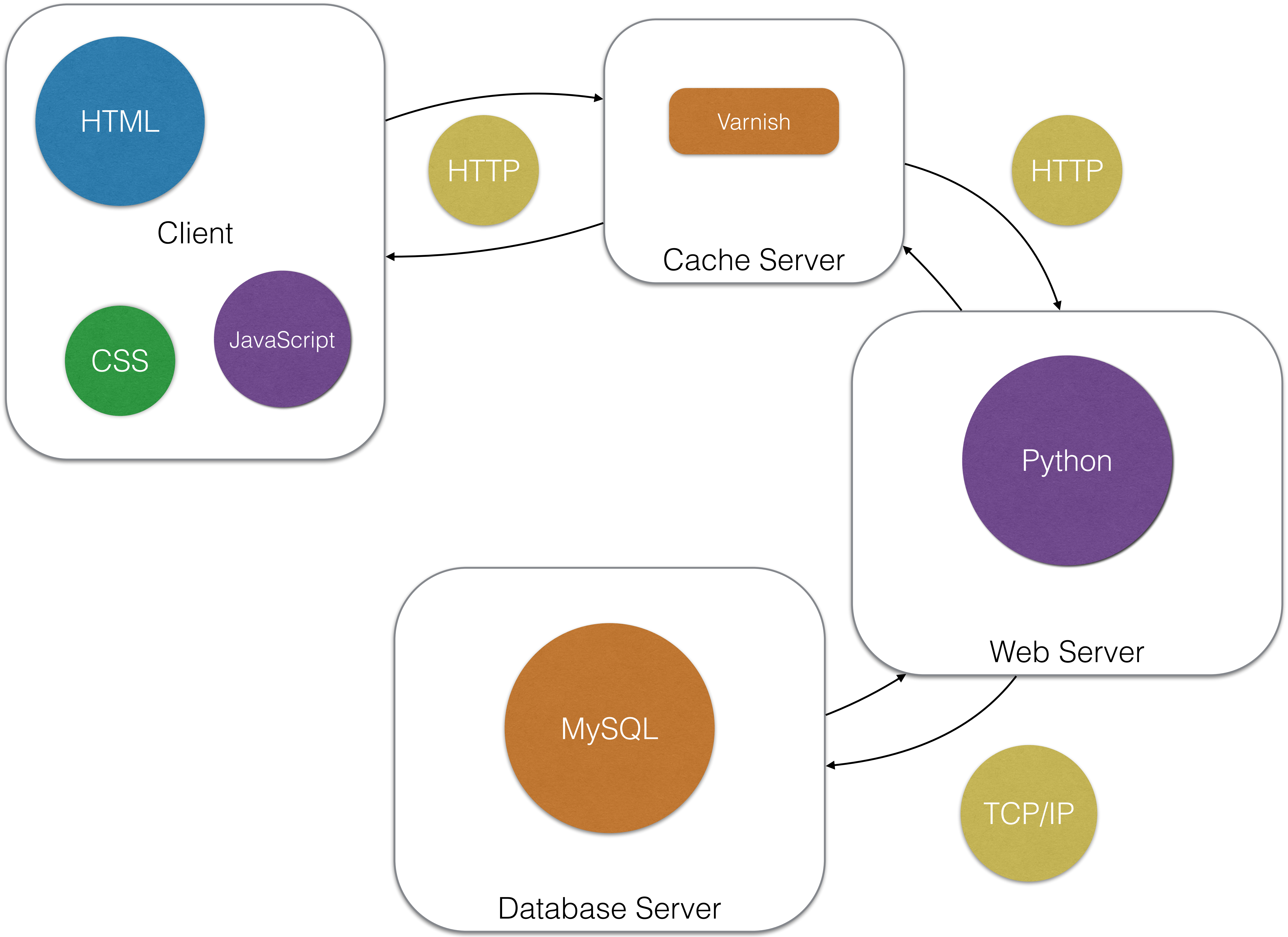
The Big Picture

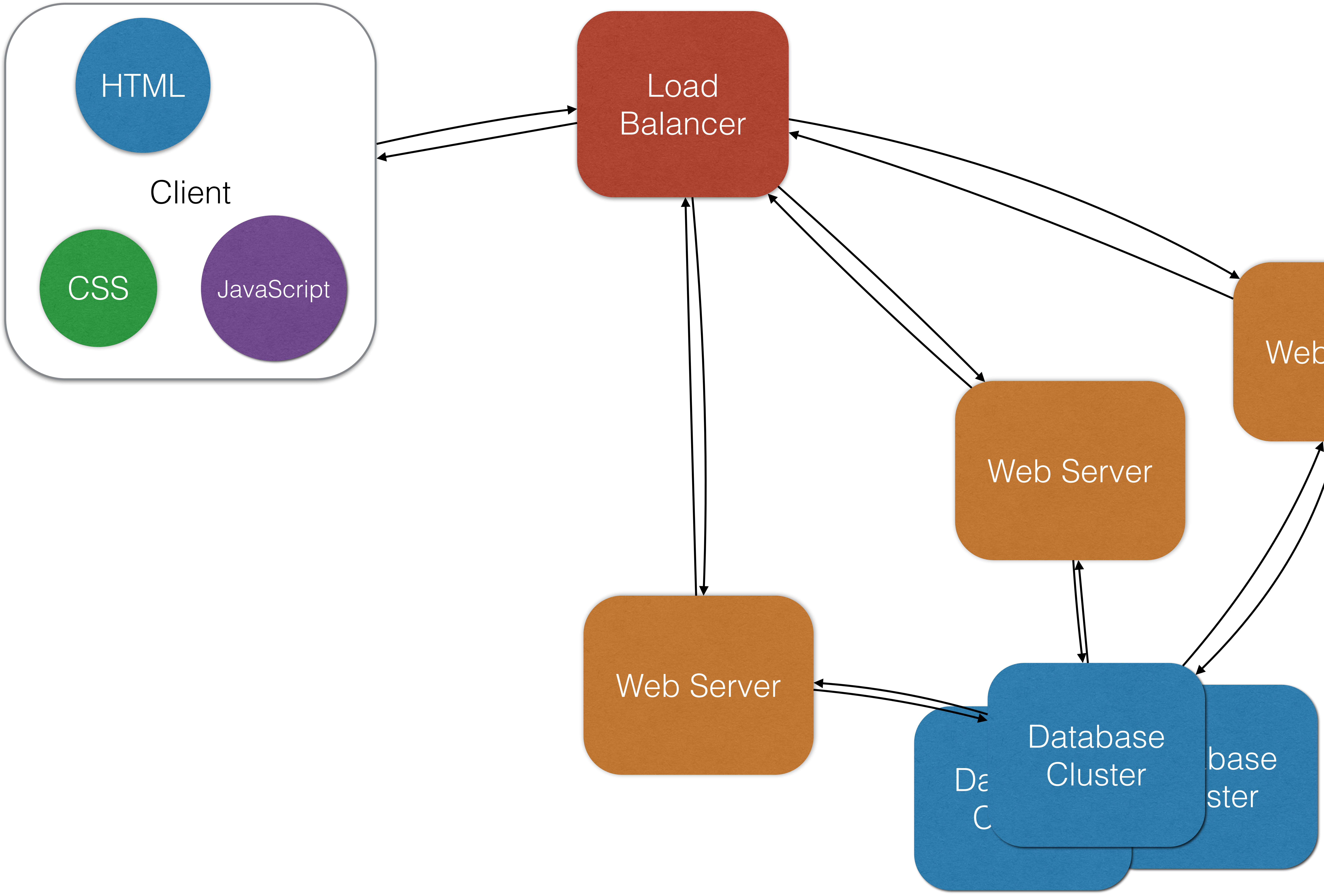


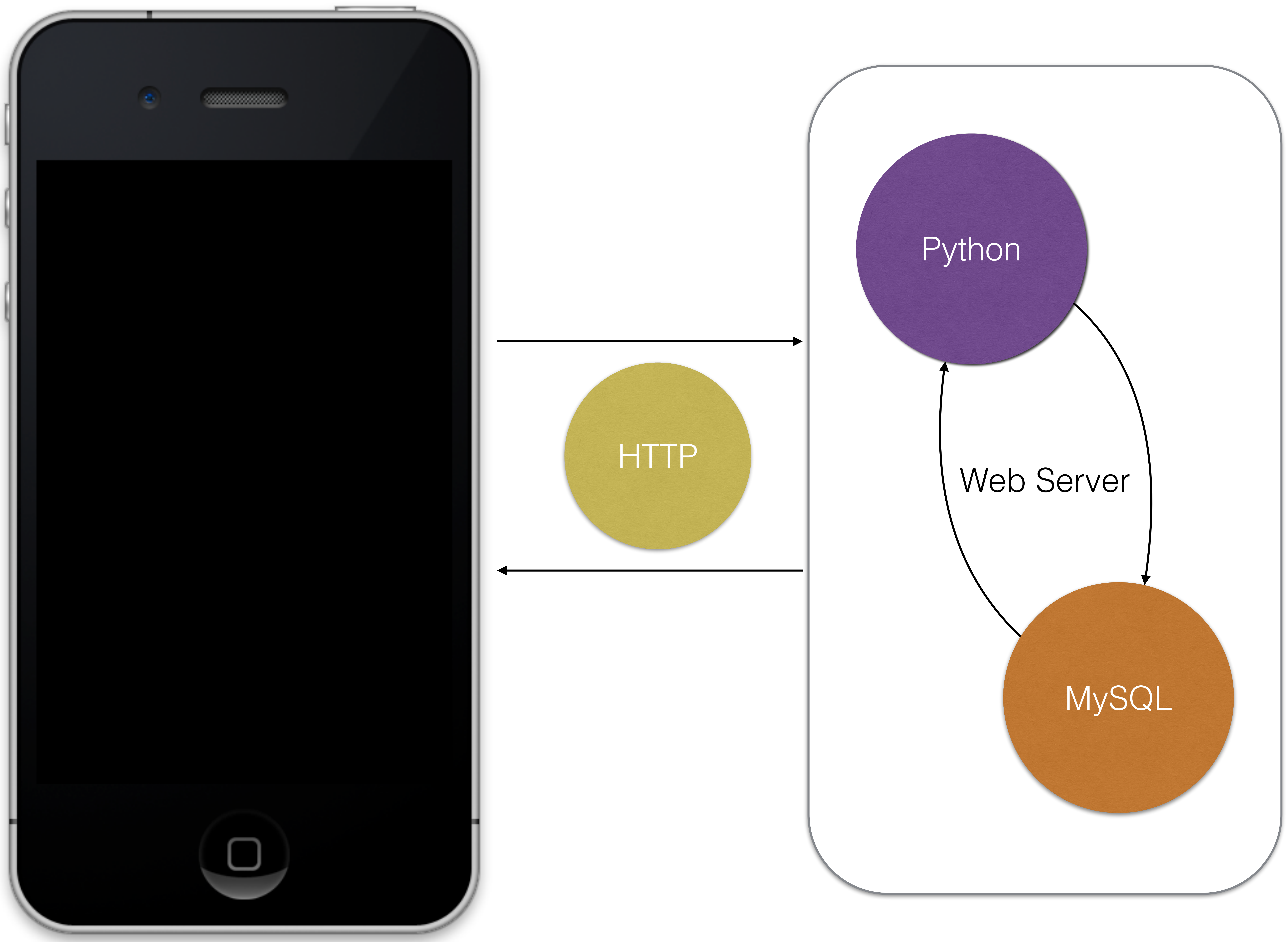

HTML
Client
JavaScript
CSS
HTTP
Python
Web Server
MySQL

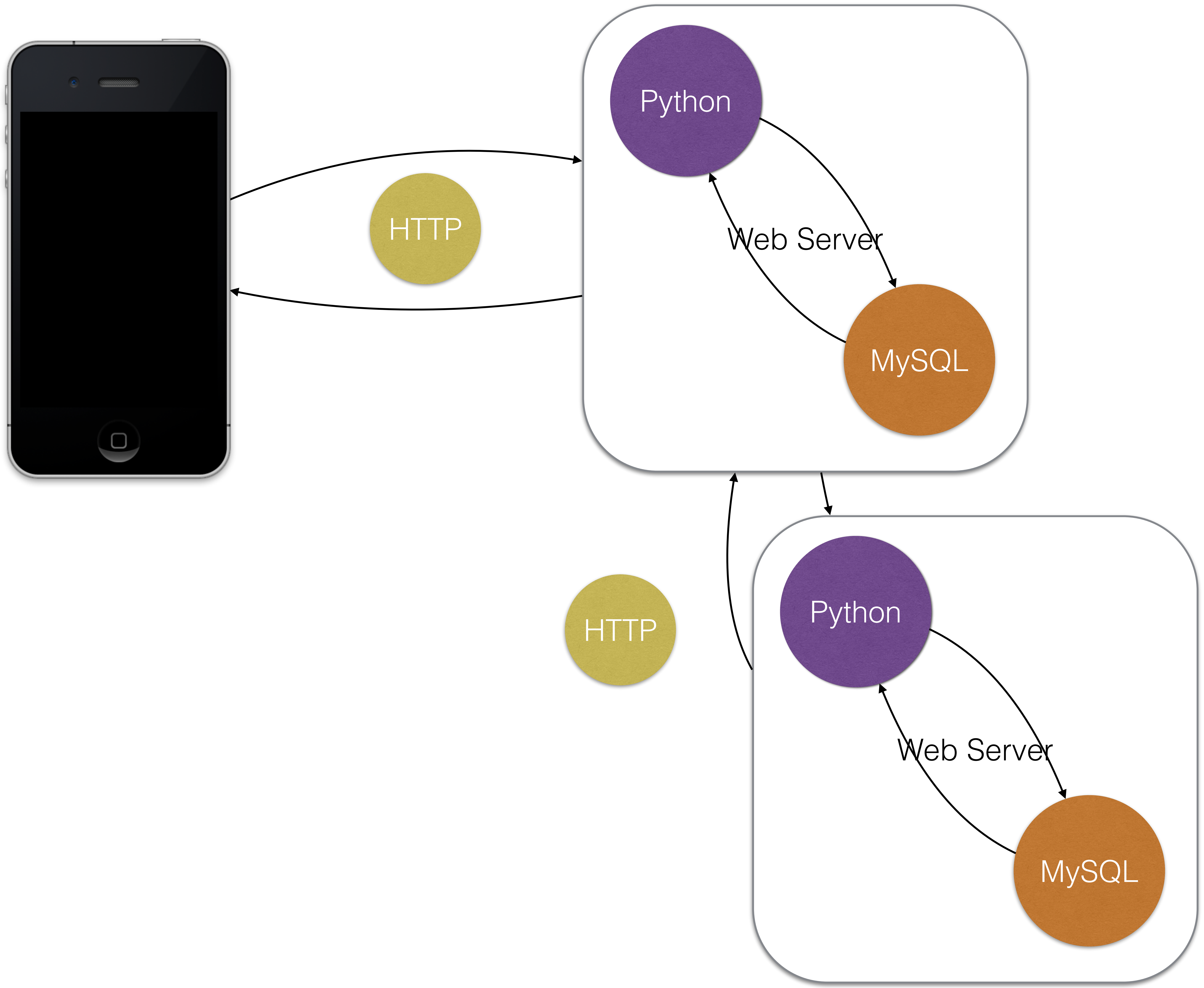

HTML
Client
JavaScript
CSS
HTTP
Python
Web Server
MySQL

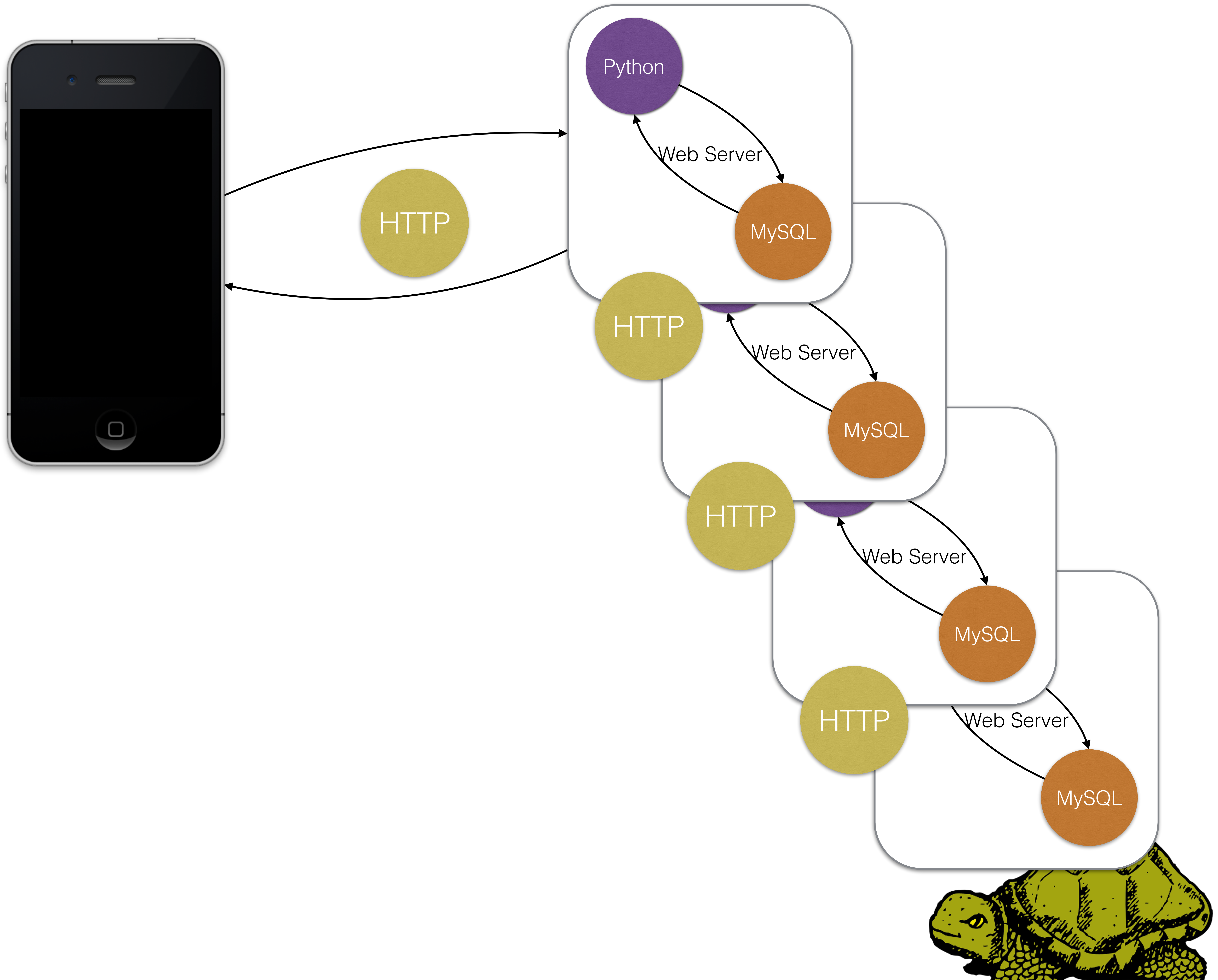

HTML
Client
JavaScript
CSS
HTTP
Java
Python
PHP
Ruby
JavaSc
C#
Web Server
Redis
memcache
MongoDB
Oracle
MyS
PostgreSQL
MSSQL


HTML
Client
CSS
JavaScript
HTTP
Varnish
Cache Server
HTTP
Python
Web Server
MySQL
Database Server
TCP/IP


HTML
Client
CSS
JavaScript
Load Balancer
Web Server
Web Server
Web Server
Database Cluster
Da C
base ster

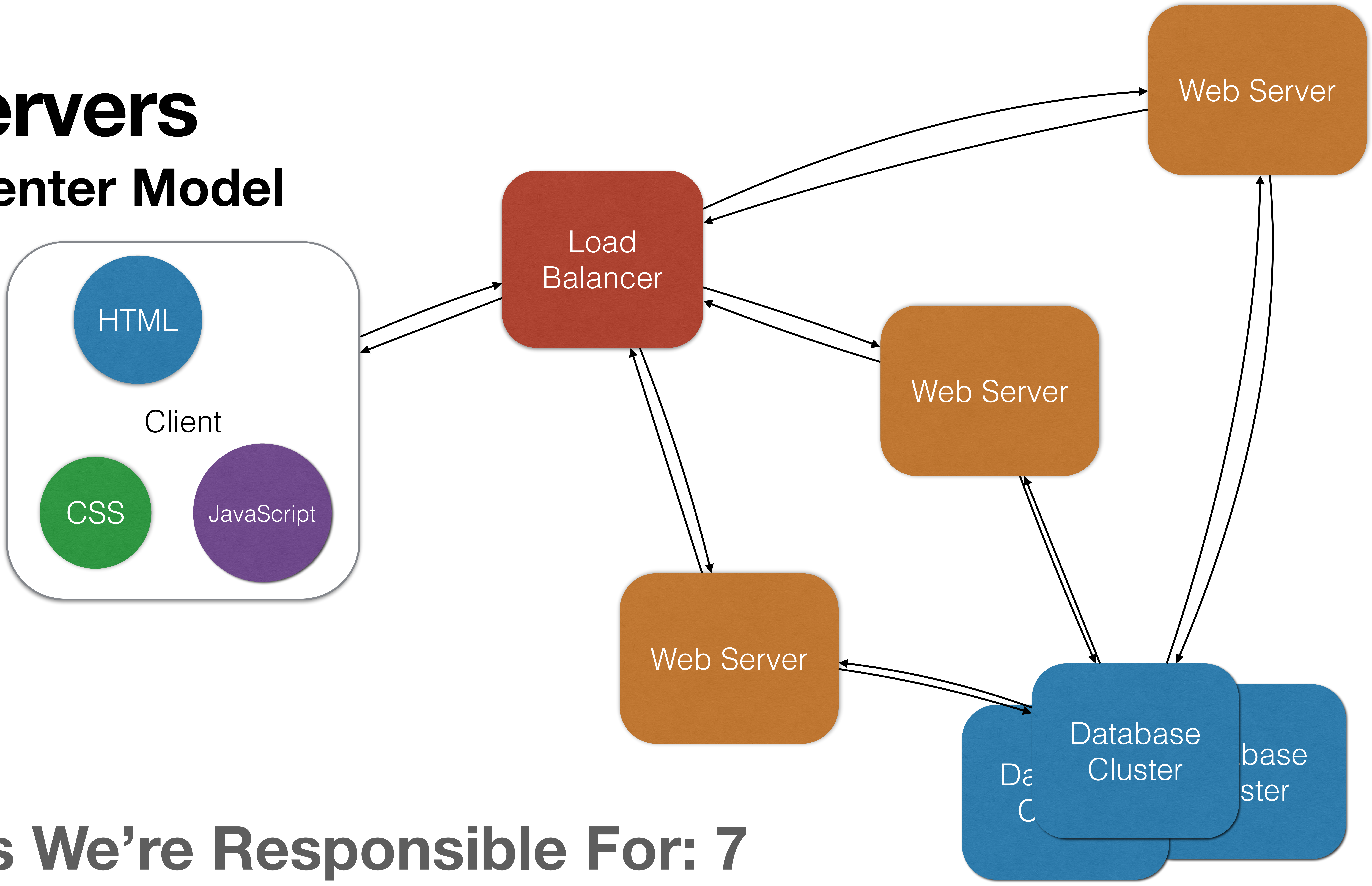

Python
HTTP
Web Server
MySQL


HTTP
Python
Web Server
MySQL
HTTP
Python
Web Server
MySQL


HTTP
Python
Web Server
MySQL
HTTP
Web Server
MySQL
HTTP
Web Server
MySQL
HTTP
Web Server
MySQL

Web Servers

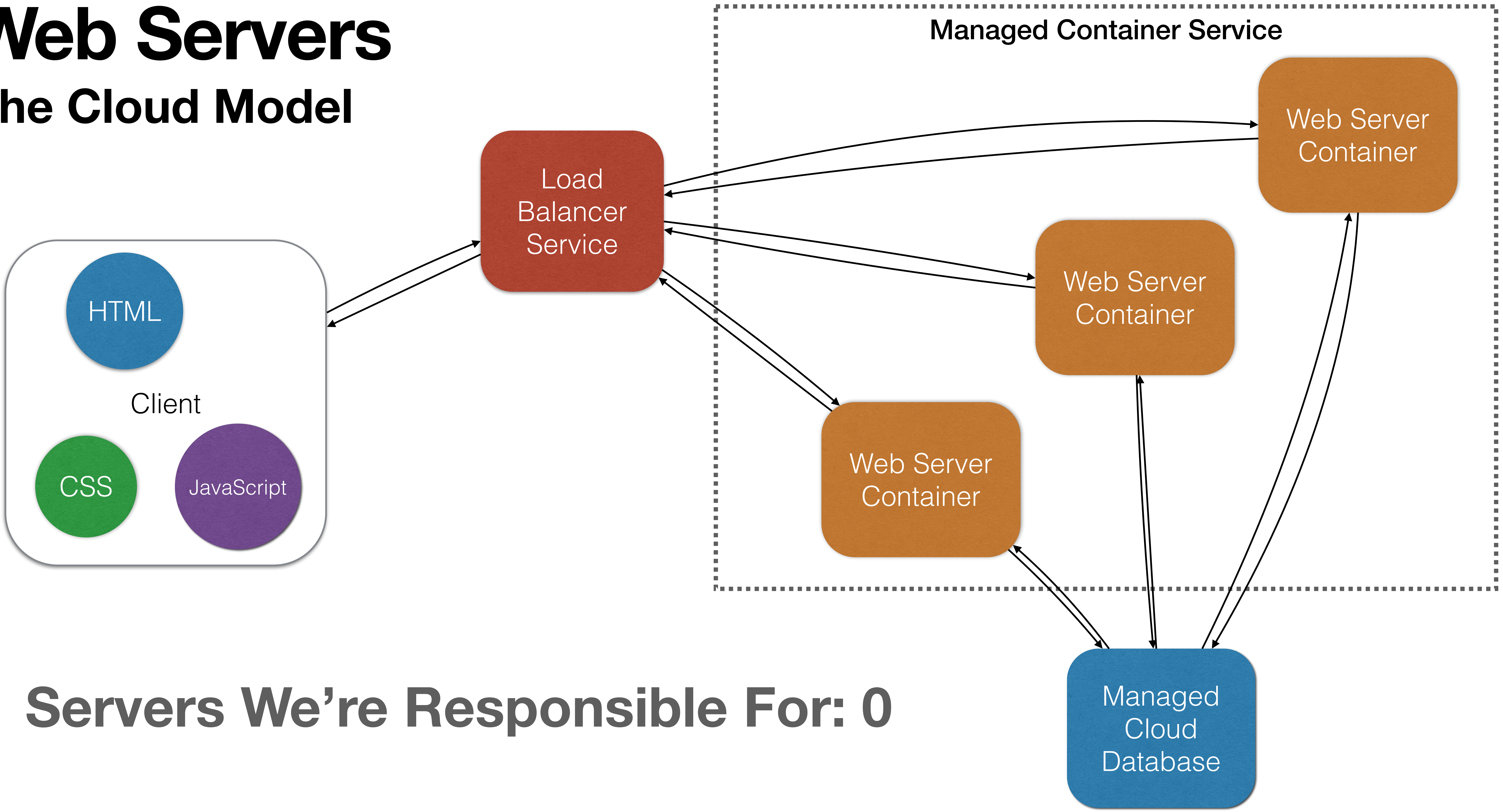
The Datacenter Model



Servers We're Responsible For: 7

Web Servers

The Cloud Model



Servers We're Responsible For: 0

Web Servers

Many Different Types

- Apache 2 - httpd
- nginx (pronounced “Engine X”)
- IIS
- Tomcat
- Jetty
- Gunicorn

Web Servers

Many Different Types

- Apache 2 - httpd
- nginx (pronounced “Engine X”)
- IIS
- Tomcat
- Jetty
- Gunicorn

General Purpose
HTTP Servers

Web Servers

Many Different Types

- Apache 2 - httpd
- nginx (pronounced “Engine X”)
- IIS
- Tomcat
- Jetty
- Gunicorn

Language Specific
HTTP Servers

Web Servers

Revisiting Containers

- We’ve already used containers to run a web server in Homework 2

```
docker run -it --rm -p 8080:80 hw02:latest
```

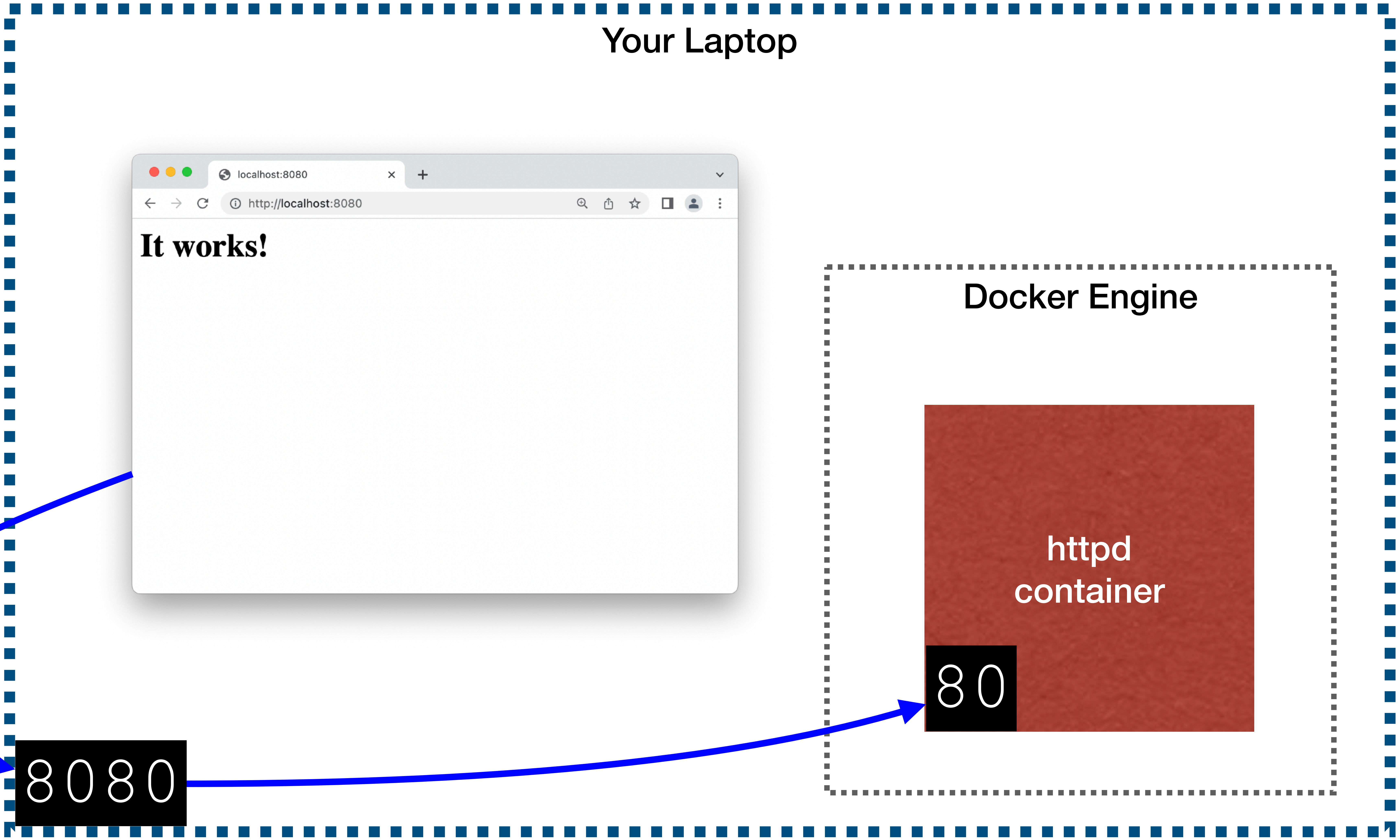
- Let’s look closer at what those port mappings mean

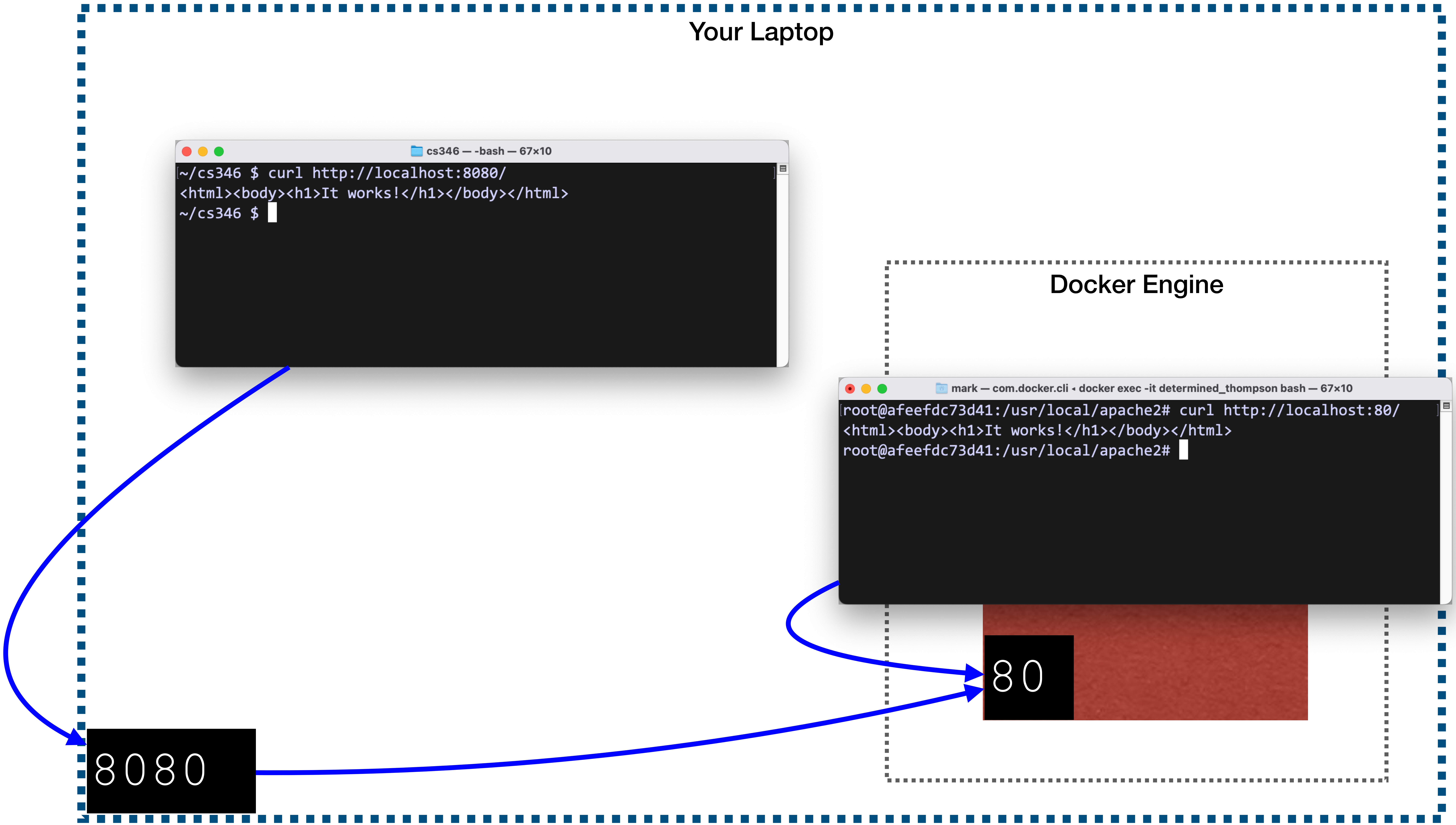
Web Servers

Revisiting Containers

```
-p 8080:80
```

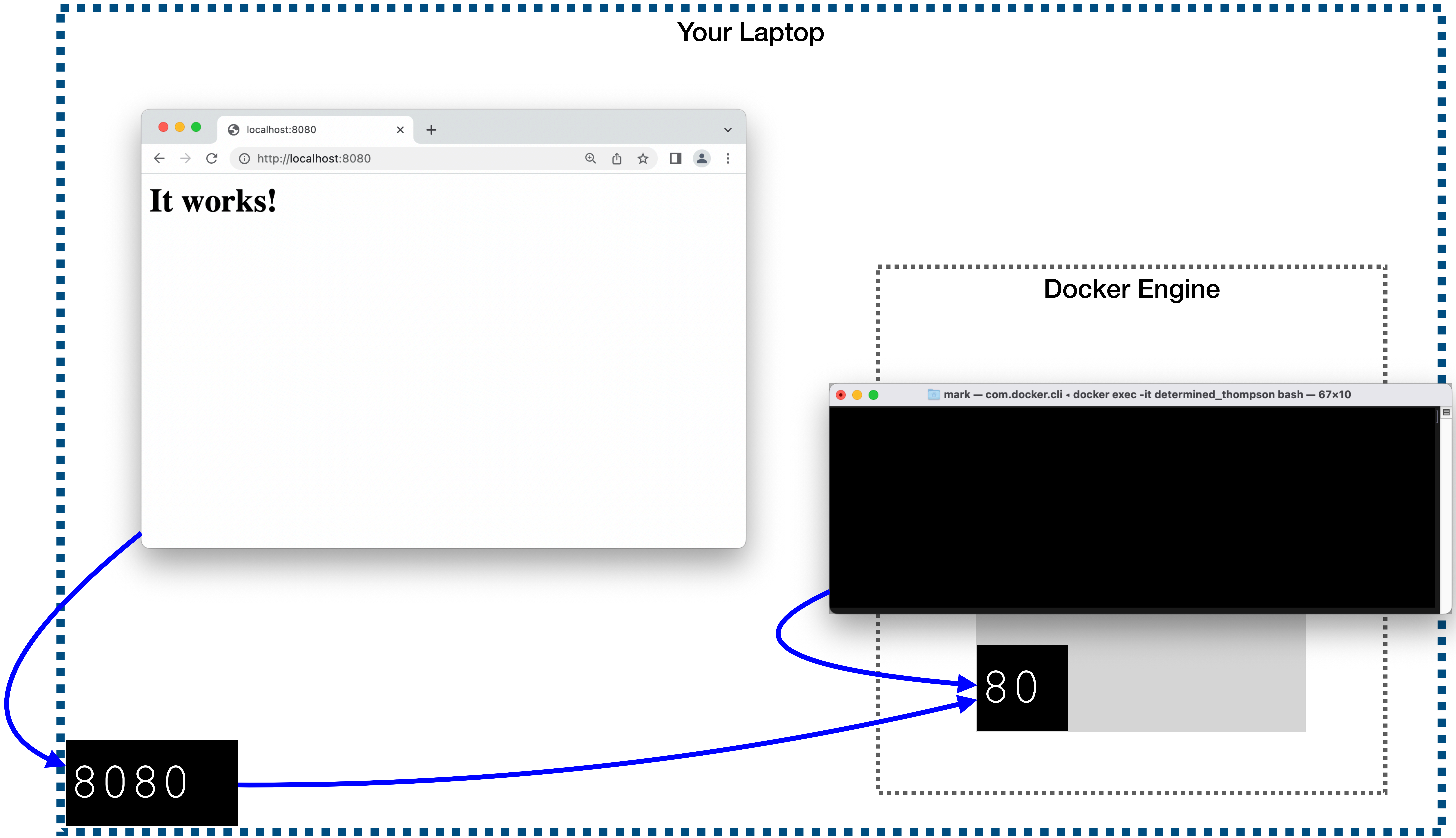
- Maps port **`8080`** on your host to the container's port **`80`**.



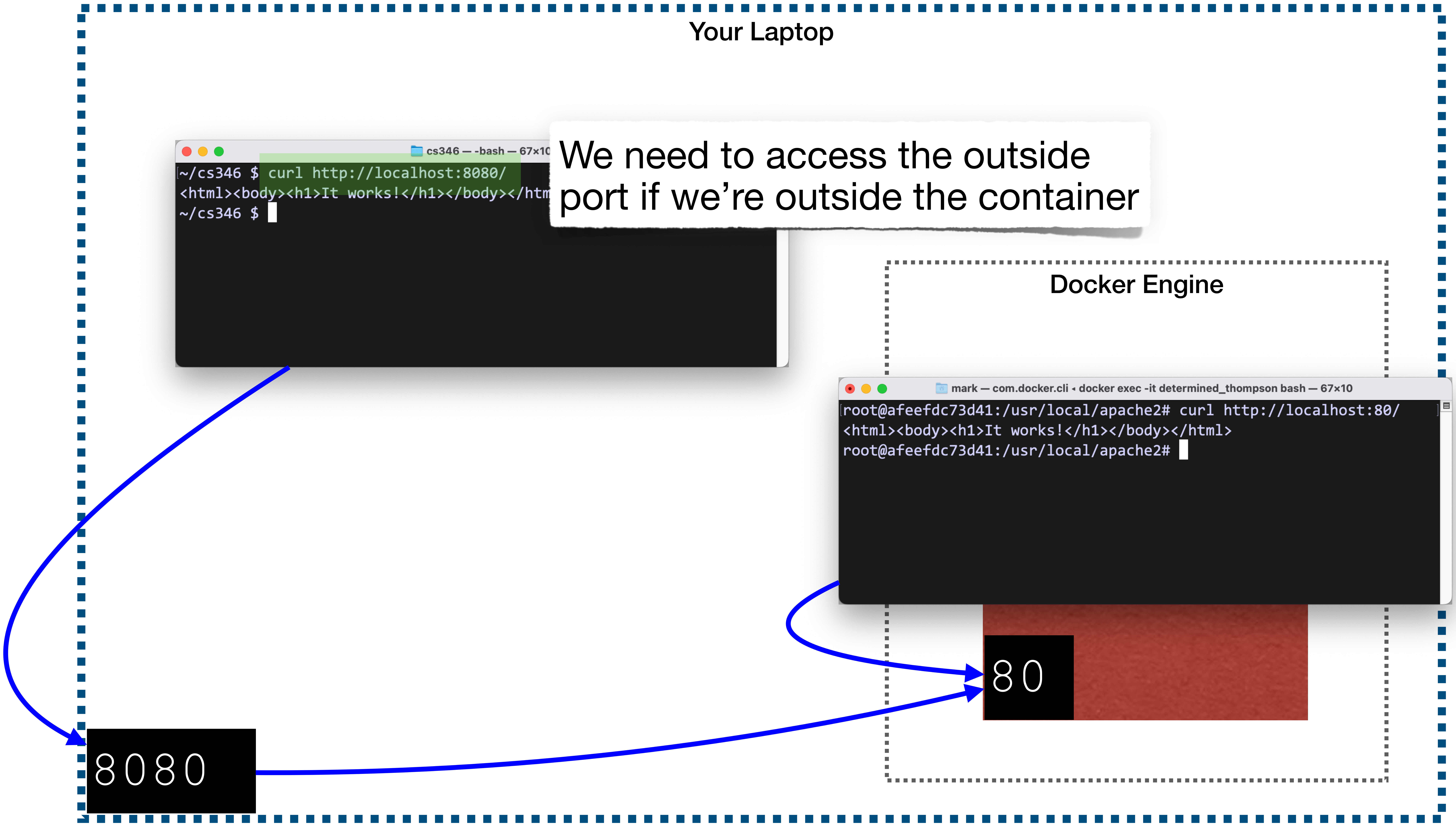

Your Laptop
Docker Engine
cs346 — -bash — 67×10
mark — com.docker.cli ◂ docker exec -it determined_thompson bash — 67×10
8080
80

```
~/cs346 $ curl http://localhost:8080/
<html><body><h1>It works!</h1></body></html>
~/cs346 $
```

```
root@afeefdc73d41:/usr/local/apache2# curl http://localhost:80/
<html><body><h1>It works!</h1></body></html>
root@afeefdc73d41:/usr/local/apache2#
```

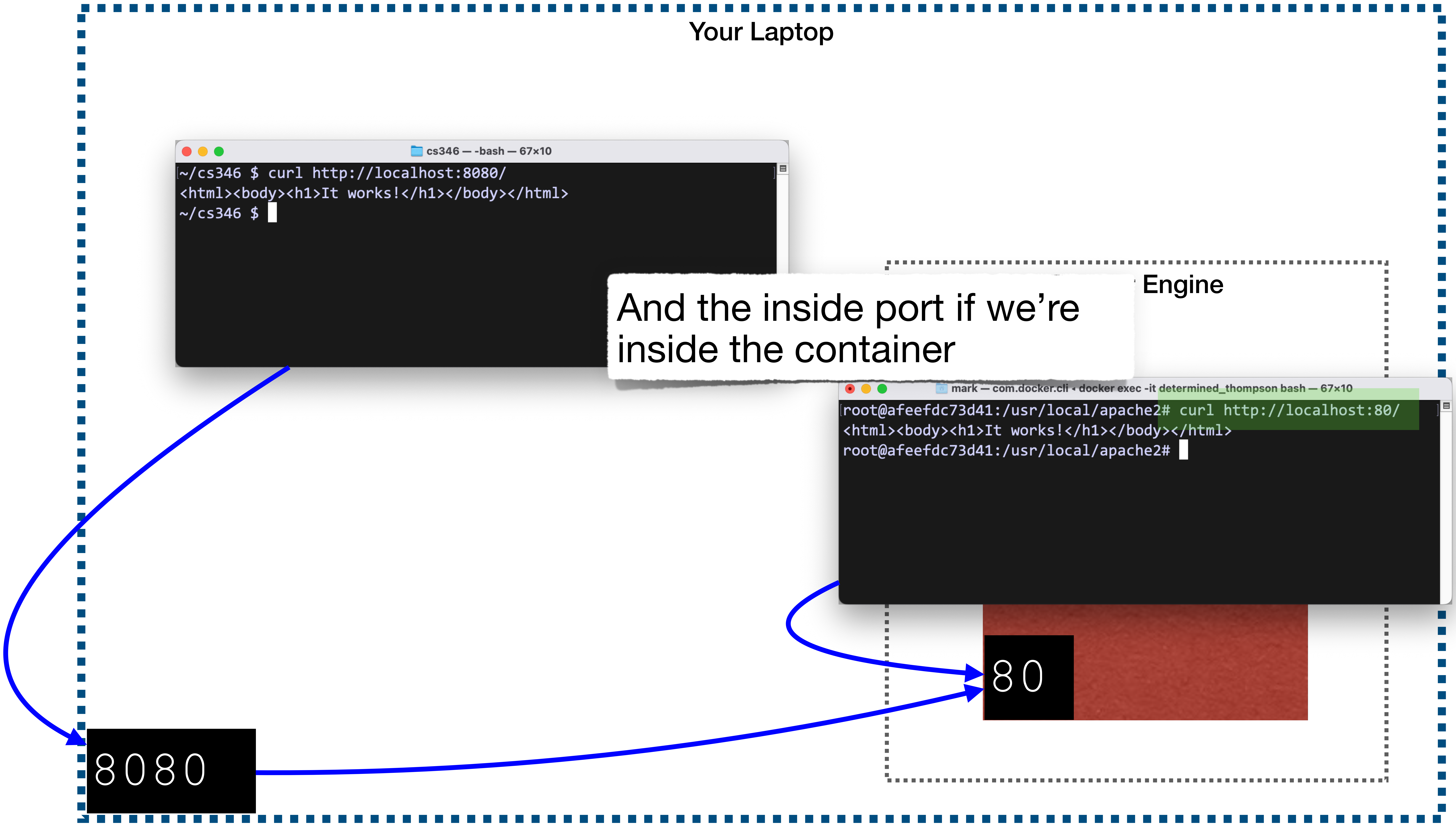

Your Laptop
localhost:8080
http://localhost:8080
It works!
Docker Engine
mark — com.docker.cli ◂ docker exec -it determined_thompson bash — 67×10
80
8080

We need to access the outside port if we're outside the container


Your Laptop
Docker Engine
cs346 — -bash — 67×10
mark — com.docker.cli ‹ docker exec -it determined_thompson bash — 67×10
8080
80

```
~/cs346 $ curl http://localhost:8080/
<html><body><h1>It works!</h1></body></htm
~/cs346 $
```

```
root@afeefdc73d41:/usr/local/apache2# curl http://localhost:80/
<html><body><h1>It works!</h1></body></html>
root@afeefdc73d41:/usr/local/apache2#
```


Your Laptop
cs346 — -bash — 67×10
Engine
mark — com.docker.cli ‹ docker exec -it determined_thompson bash — 67×10
8080
80

```
~/cs346 $ curl http://localhost:8080/
<html><body><h1>It works!</h1></body></html>
~/cs346 $
```

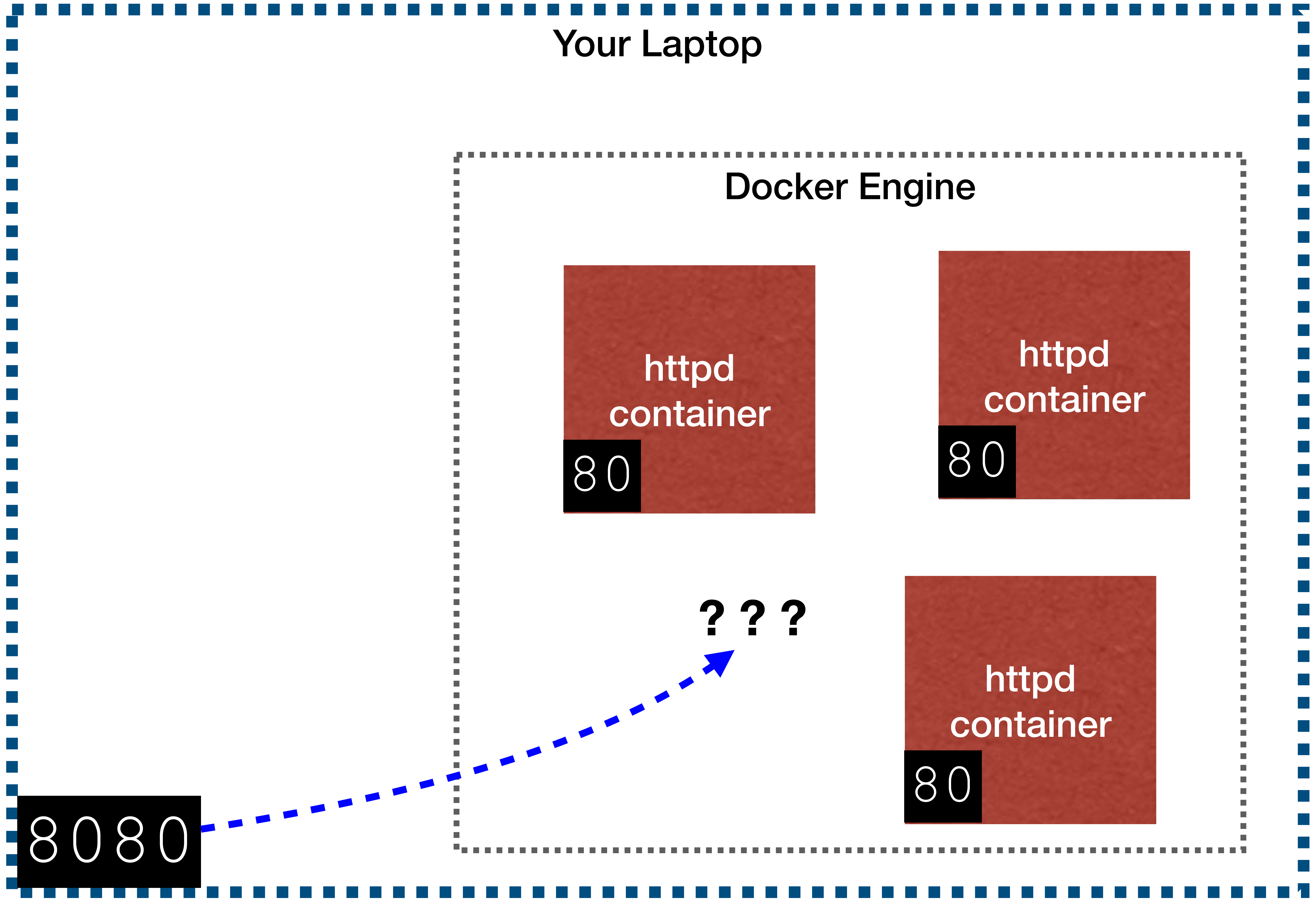
And the inside port if we're inside the container

```
root@afeefdc73d41:/usr/local/apache2# curl http://localhost:80/
<html><body><h1>It works!</h1></body></html>
root@afeefdc73d41:/usr/local/apache2#
```

Web Servers

Revisiting Containers

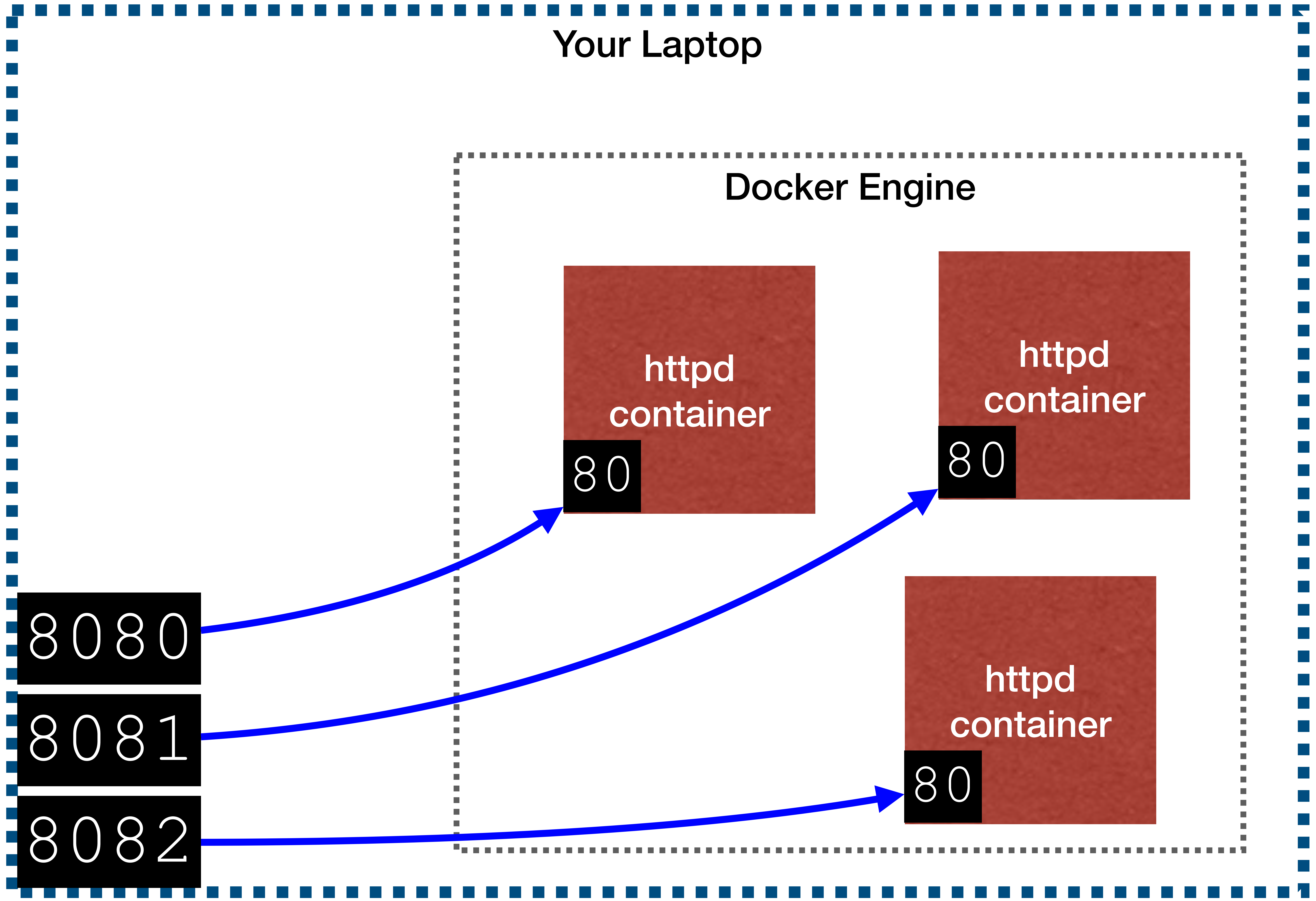
- We can run multiple containers, all with the same internal port.
- We can't map the same port on the host to multiple containers!



Web Servers

Revisiting Containers

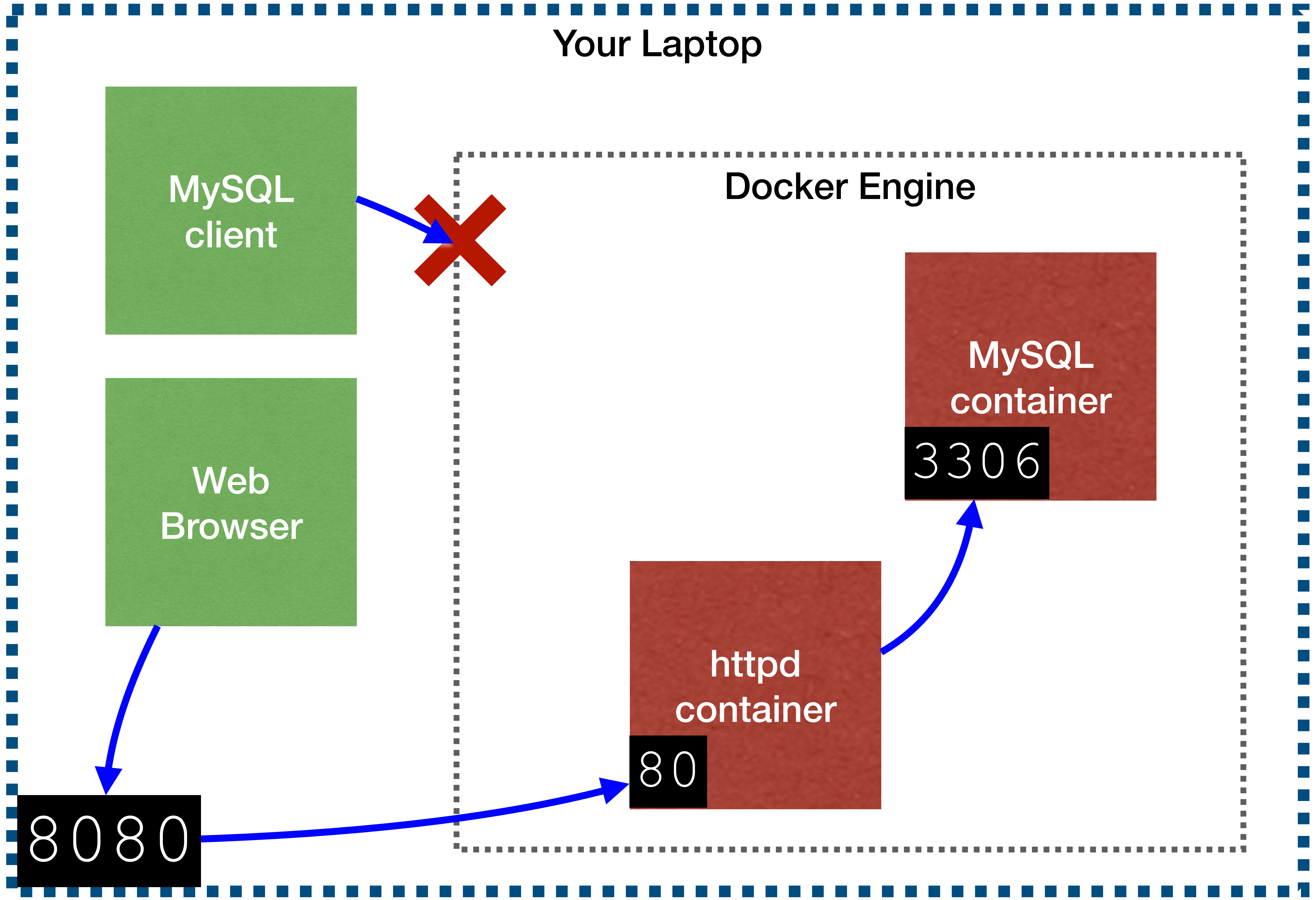
- We need separate ports on the host for each container we want to forward traffic to



Web Servers

Revisiting Containers

- Not all containers need their ports mapped to the host
- Containers can also talk to each other directly, without having to leave the internal docker network



Up Next: Javascript!