REDES Y SISTEMAS DISTRIBUÍDOS

LAB 2

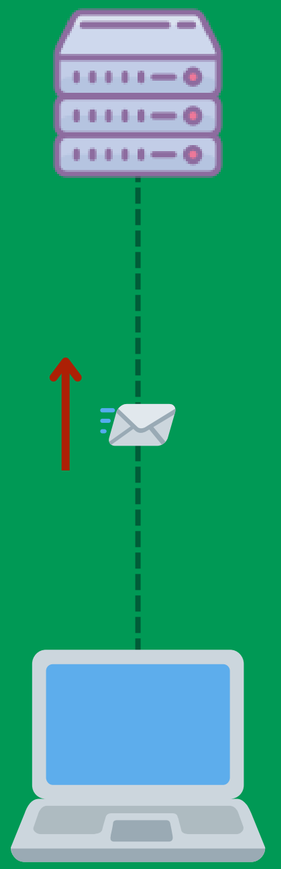
APLICACIÓN SERVIDOR

Alvaro Medina
Ezequiel Pastore
Lorenzo Canovas
Matías Kühn

CONTEXTO & OBJETIVOS

CONTEXTO & OBJETIVOS

1. Comunicación cliente/servidor por medio de la programación de sockets
2. Familiarizarse con un protocolo de aplicación
3. Comprender, diseñar e implementar un programa servidor de archivos en Python.



IMPORTANCIA

PROTOCOLO

DE TRANSFERENCIA DE

DATOS

IMPORTANCIA

PROTOCOLO
DE TRANSFERENCIA DE
DATOS

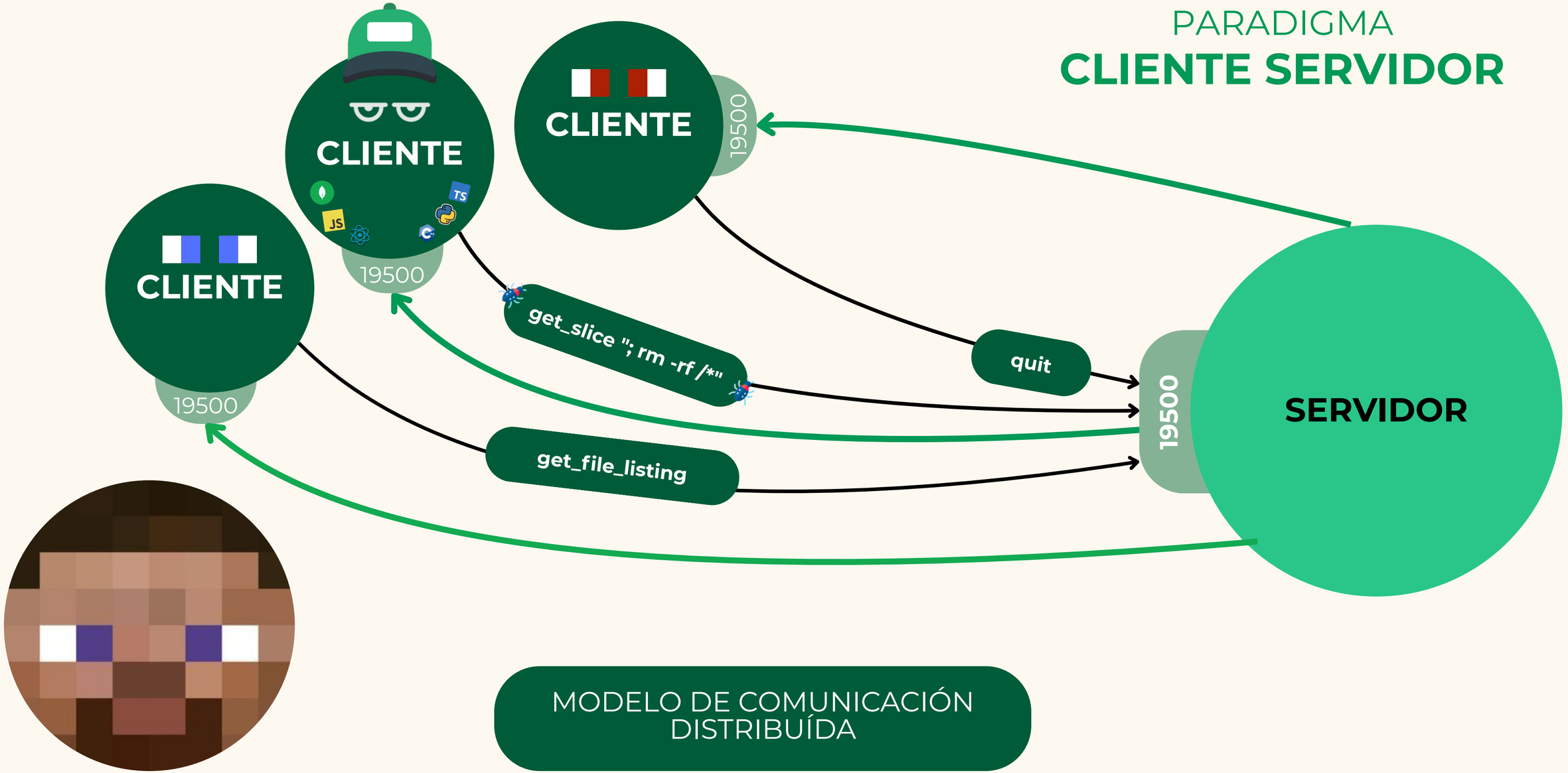
1. Estandarización y Orden en la Comunicación
2. Manejo de Errores y Robustez
3. Seguridad (Diseño Defensivo)
4. Eficiencia en Transferencias

PARADIGMA

CLIENTE SERVIDOR

PARADIGMA

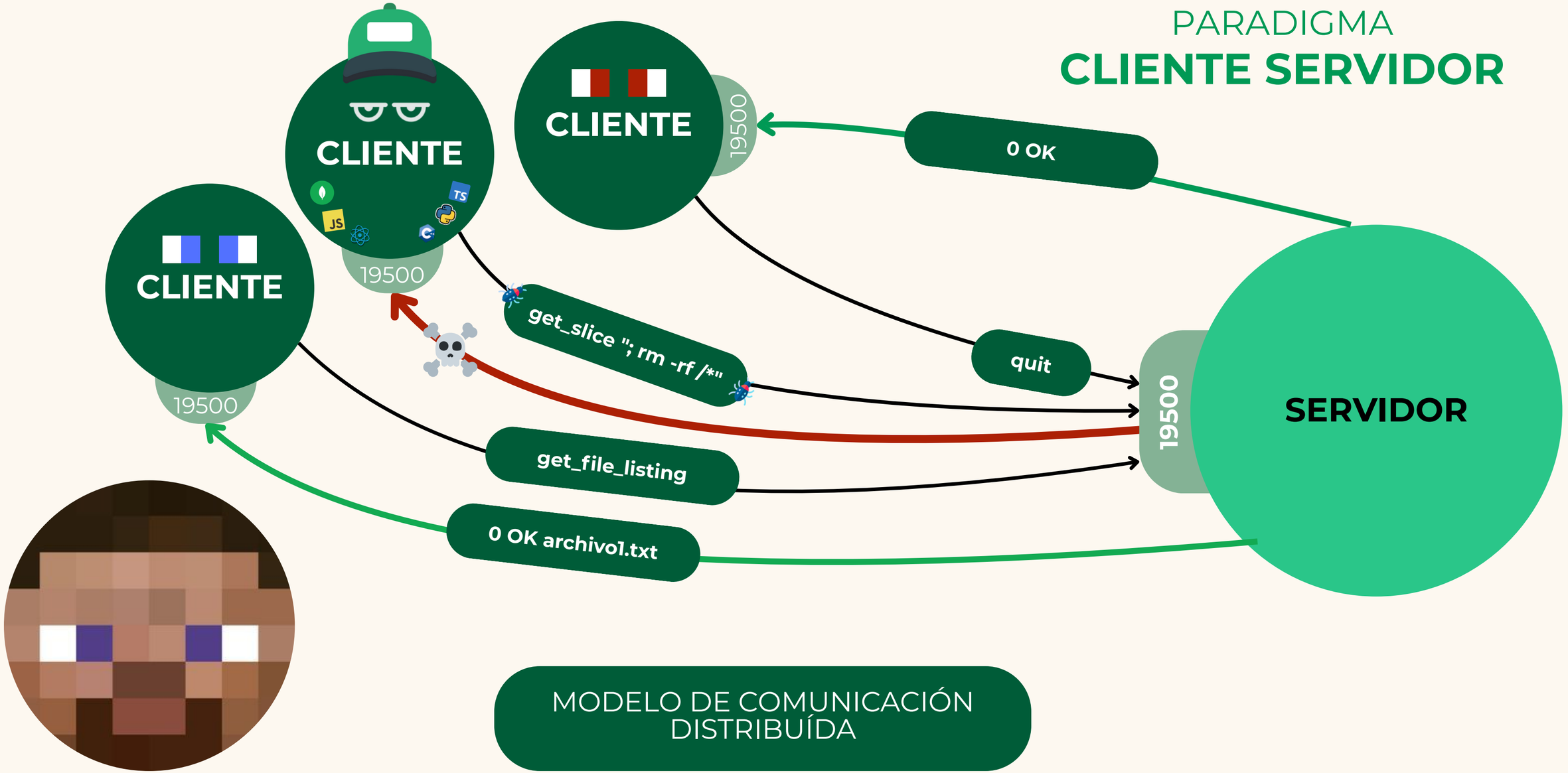
CLIENTE SERVIDOR



MODELO DE COMUNICACIÓN DISTRIBUÍDA

PARADIGMA

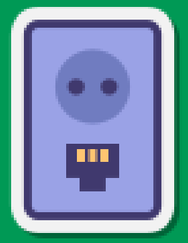
CLIENTE SERVIDOR



MODELO DE COMUNICACIÓN DISTRIBUÍDA

PARADIGMA

CLIENTE SERVIDOR



PROGRAMACIÓN CON

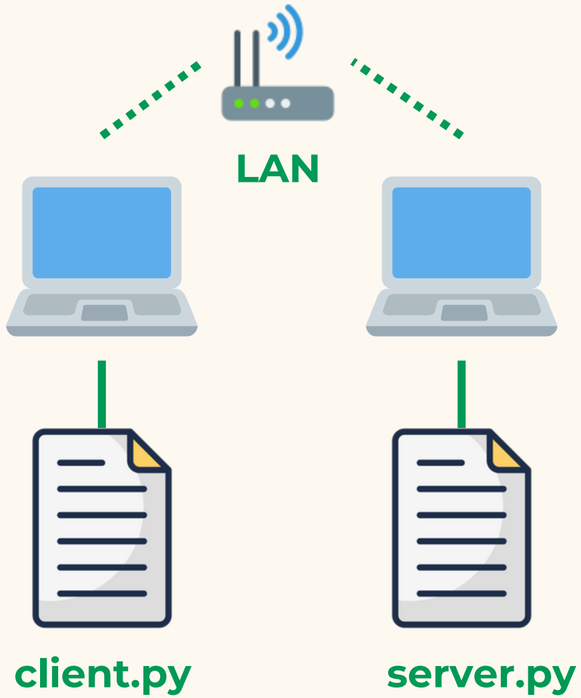
SOCKETS

PROGRAMACIÓN CON

SOCKETS

FORMA 1

FORMA 2



FORMA 1

client.py — server.py

server.py

PROGRAMACIÓN CON

SOCKETS

```python
import socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # TCP/IP
server_socket.bind(("0.0.0.0", 19500))  # Escucha en el puerto 19500
server_socket.listen(5)  # Cola de hasta 5 conexiones pendientes
```

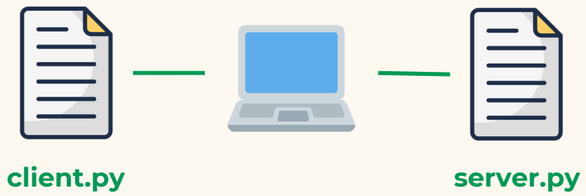
```python
while True:
    client_socket, client_address = server_socket.accept()  # Bloqueante
    print(f"Conexión aceptada desde: {client_address}")
```

```python
data = client_socket.recv(1024).decode()  # Lee hasta 1024 bytes
if data.startswith("get_file_listing"):
    response = "0 OK\r\narchivo1.txt\r\narchivo2.txt\r\n"
    client_socket.send(response.encode())
```

PROGRAMACIÓN CON

SOCKETS

FORMA 1



client.py

1 Conexión con el servidor

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(("127.0.0.1", 19500))  # IP y puerto del servidor
```

2 Enviamos información

```
client_socket.send("get_file_listing\r\n".encode())
```

3 Recibir respuesta del servidor

```
response = client_socket.recv(1024).decode()
print(response)  # Ej: "0 OK\r\narchivo1.txt\r\narchivo2.txt\r\n"
```

4 Cerrar la conexión con el Iservidor

```
client_socket.close()
```

LORENZO

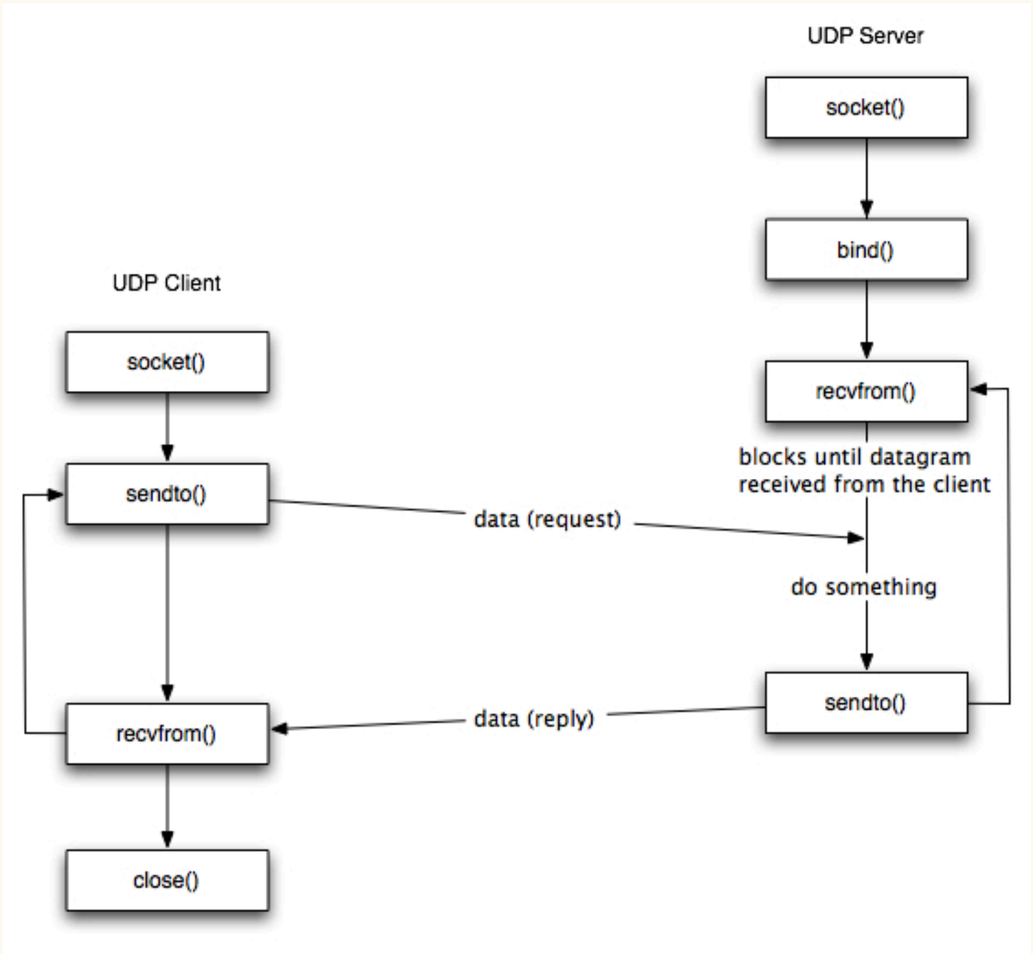
TCP
VS
UDP

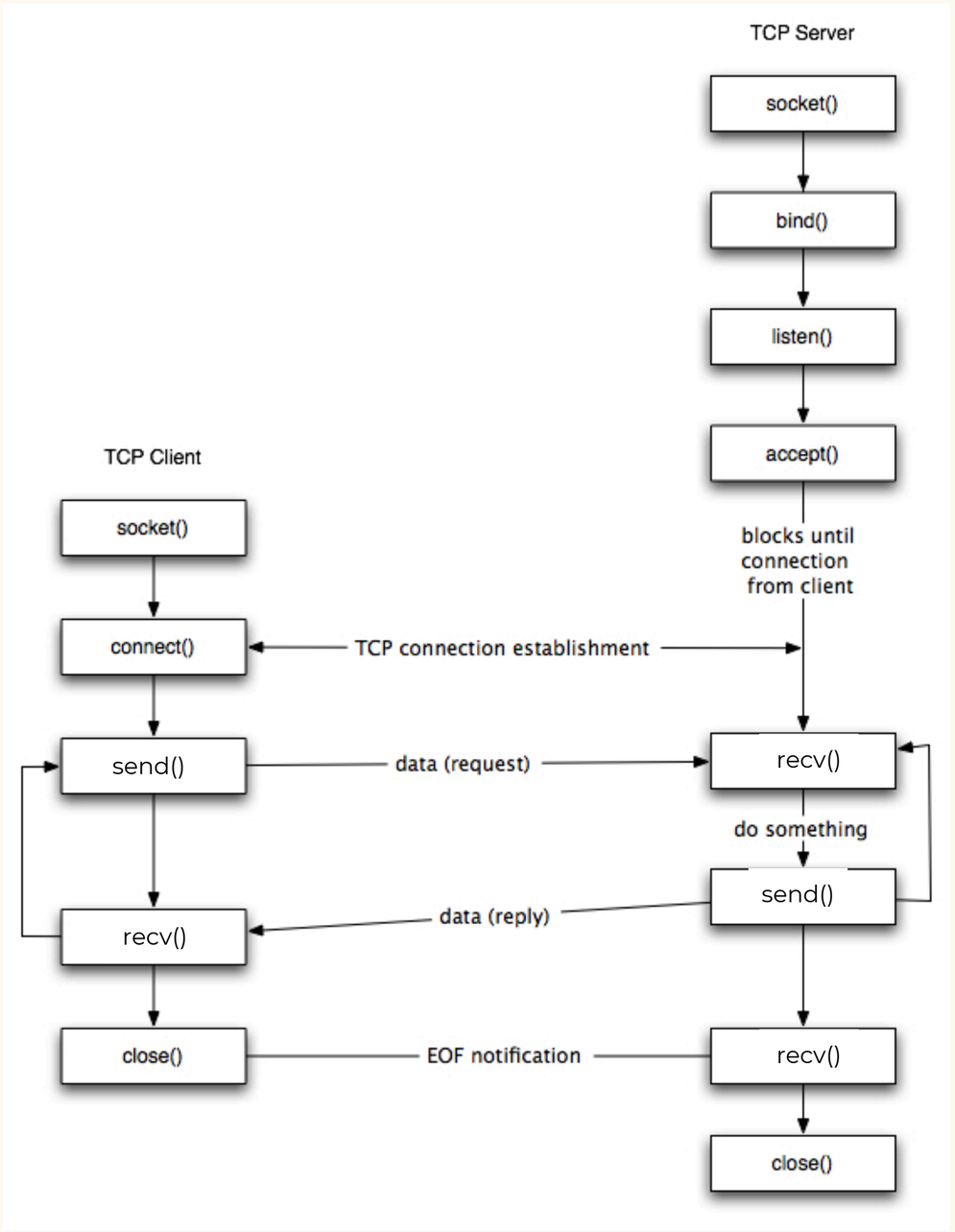
DESDE LA PERSPECTIVA DEL SOCKET

TCP
VS
UDP

Sockets en python

Modulo estándar de sockets

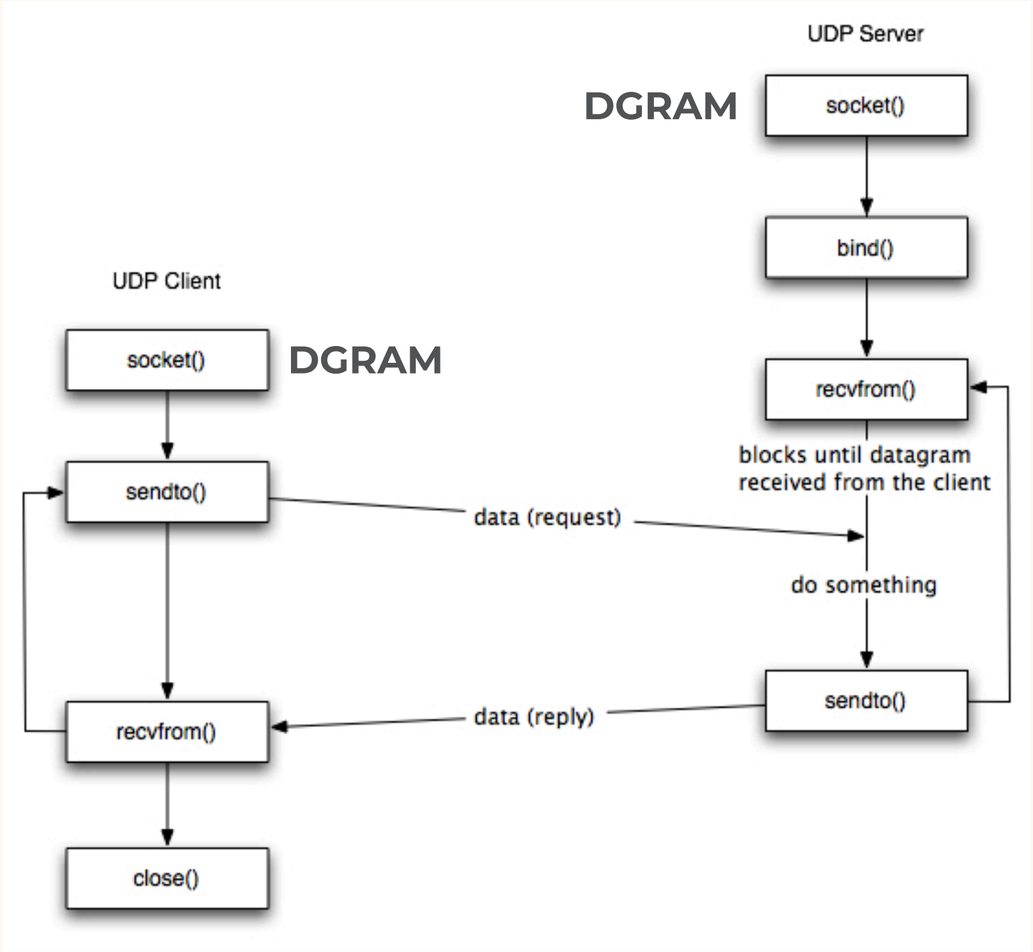


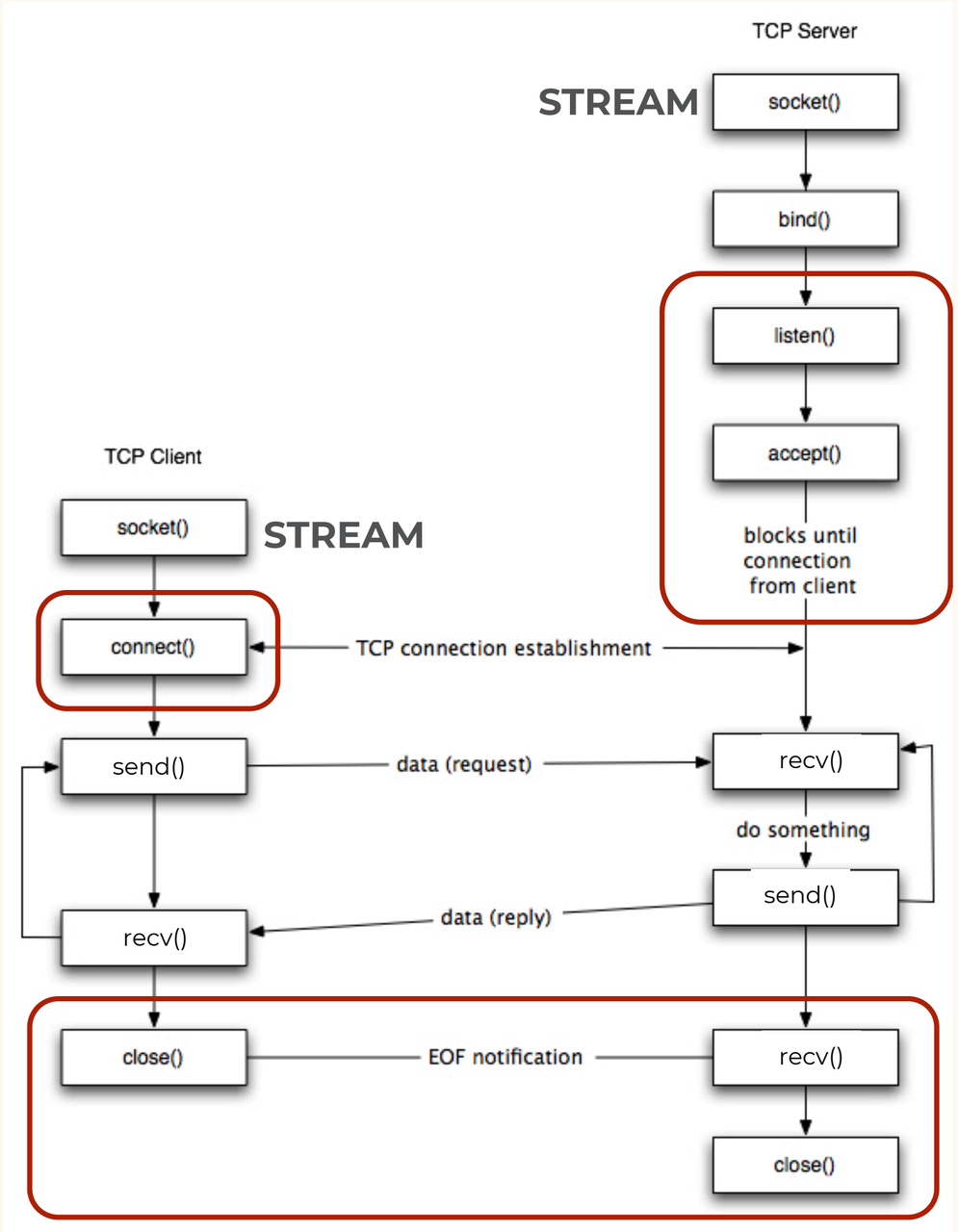


TCP
VS
UDP

Sockets en python

Modulo estándar de sockets





File
Transfer
Protocol

¿PARA QUÉ SIRVE?

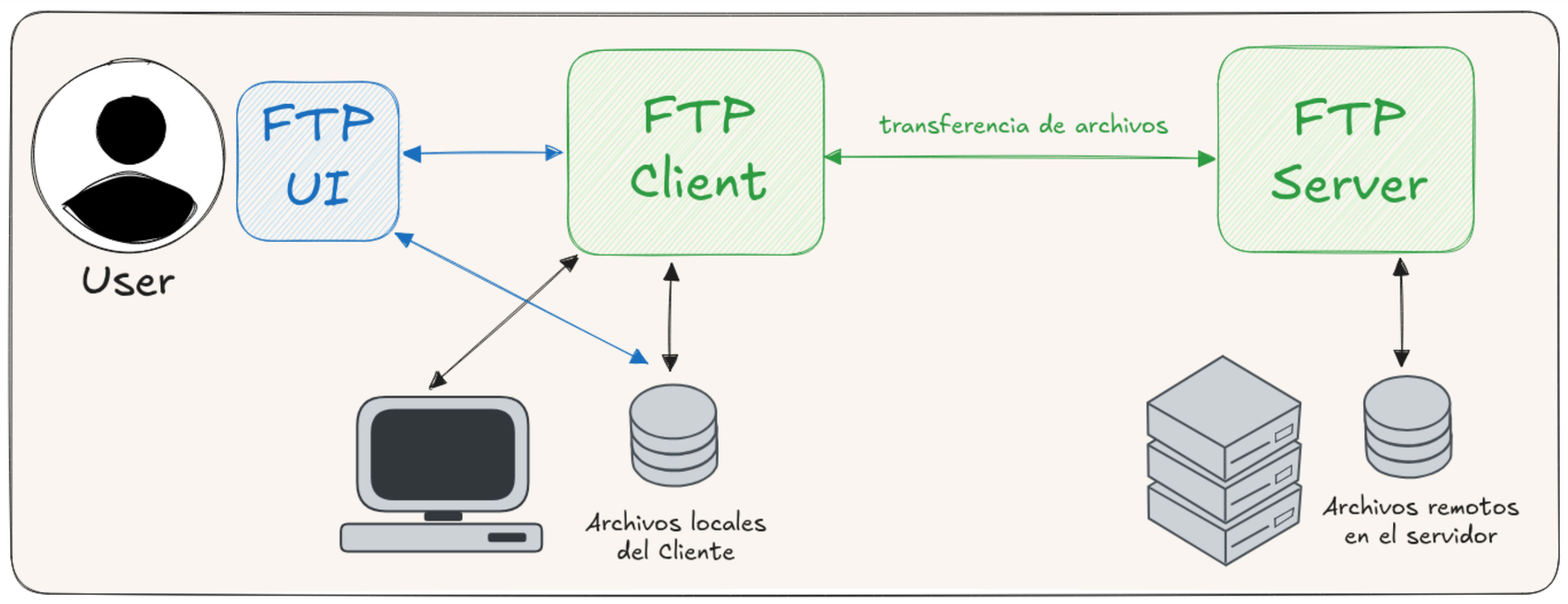
PROTOCOLO

DE TRANSFERENCIA DE

DATOS

Protocolo de Transferencia de archivos

Utiliza la arquitectura Cliente-Servidor



El intercambio de info se realiza en texto plano.

Alternativas:

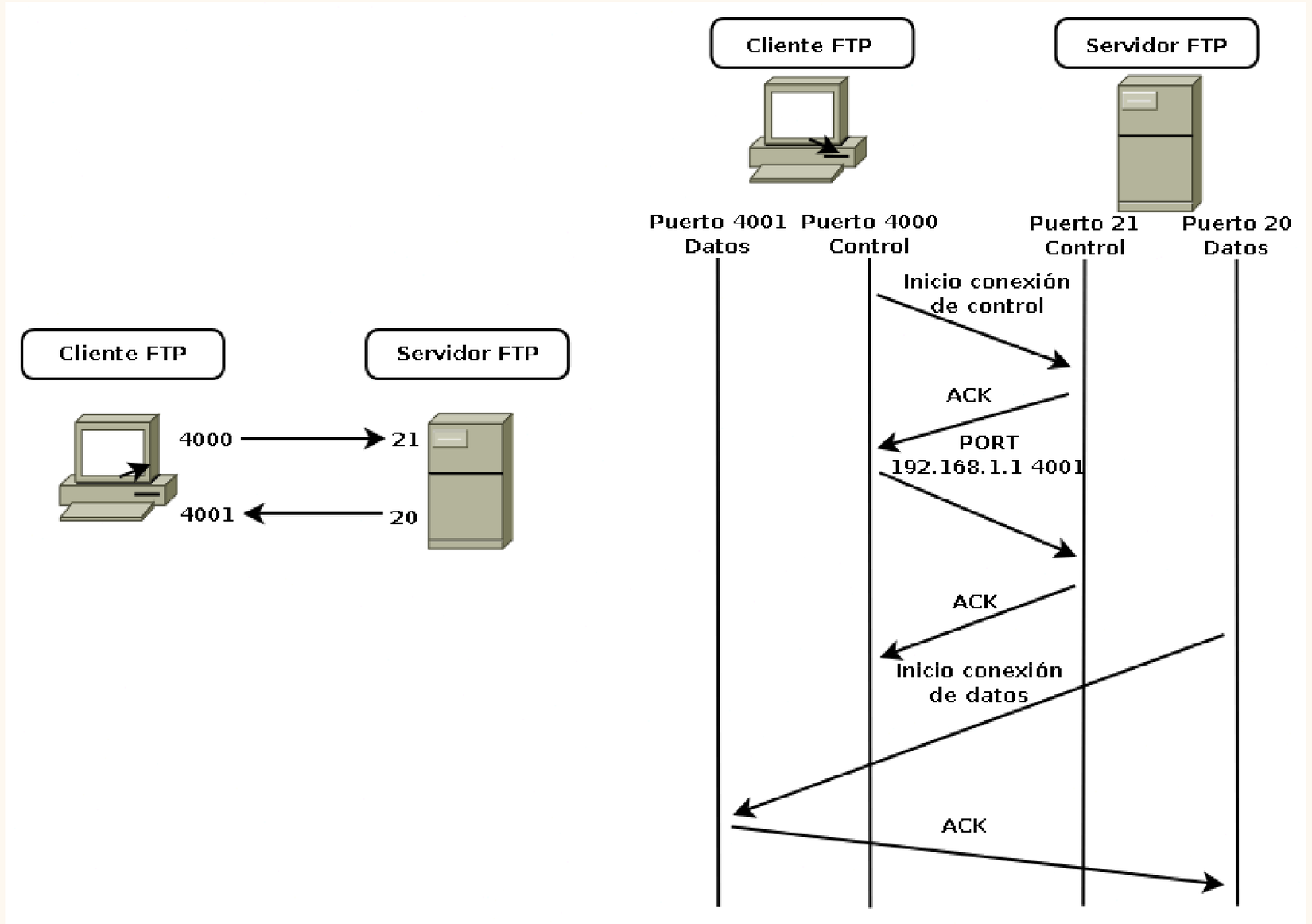
- **SFTP**
- **FTPS**
- **HTTPS**

Mensajes:

- **Comandos**
- **Respuesta**
- **Datos**

La idea general del protocolo HFTP hecho en el lab02 de redes es similar a la de FTP

Funcionamiento de FTP



🤔 ¿POR QUE 64?

🤔 ¿QUE ES?

BASE64

🤔 ¿PARA QUE NOS SIRVE?

⚒ USO EN EL 🧪
LABORATORIO

QUÉ PASA SI...

¿CUAL ES EL SECRETO?

QUIT

GET FILE LISTING

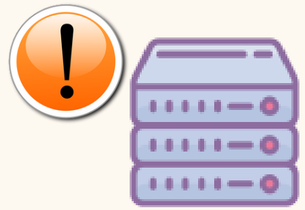
PRINCIPALES FUNCIONES DEL SERVIDOR

GET METADATA

GET SLICE

ERRORES Y DIFICULTADES

Codificación-Decodificación en Base64



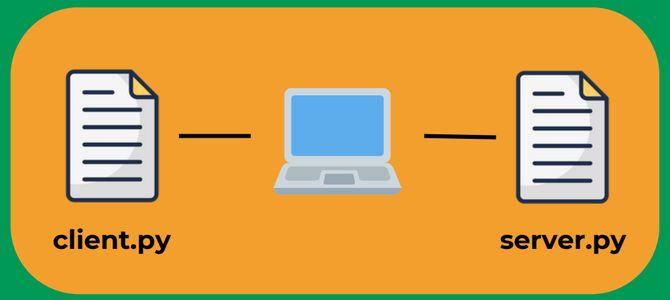
Tamaño de buffer

Protección ante ataques DoS

Concurrencia

Conectarse al server desde otra red/dispositivo

Cierre de las conexiones

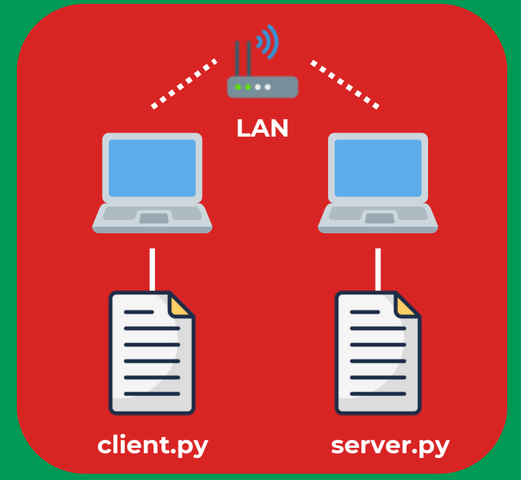


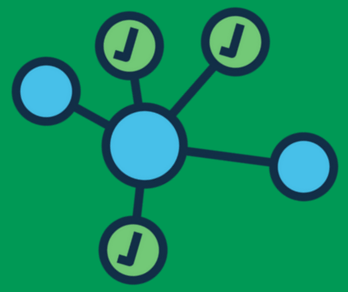
CORRER
EL SERVIDOR

LOCALHOST

VS

0.0.0.0





RELACIÓN
LAB1

Conceptos Fundamentales
de
Protocolos de comunicación

Operaciones CRUD

Paradigma Cliente-Servidor

Manejo de Errores

Testing Exhaustivo

CONCLUSIONES

MUCHAS

GRACIAS