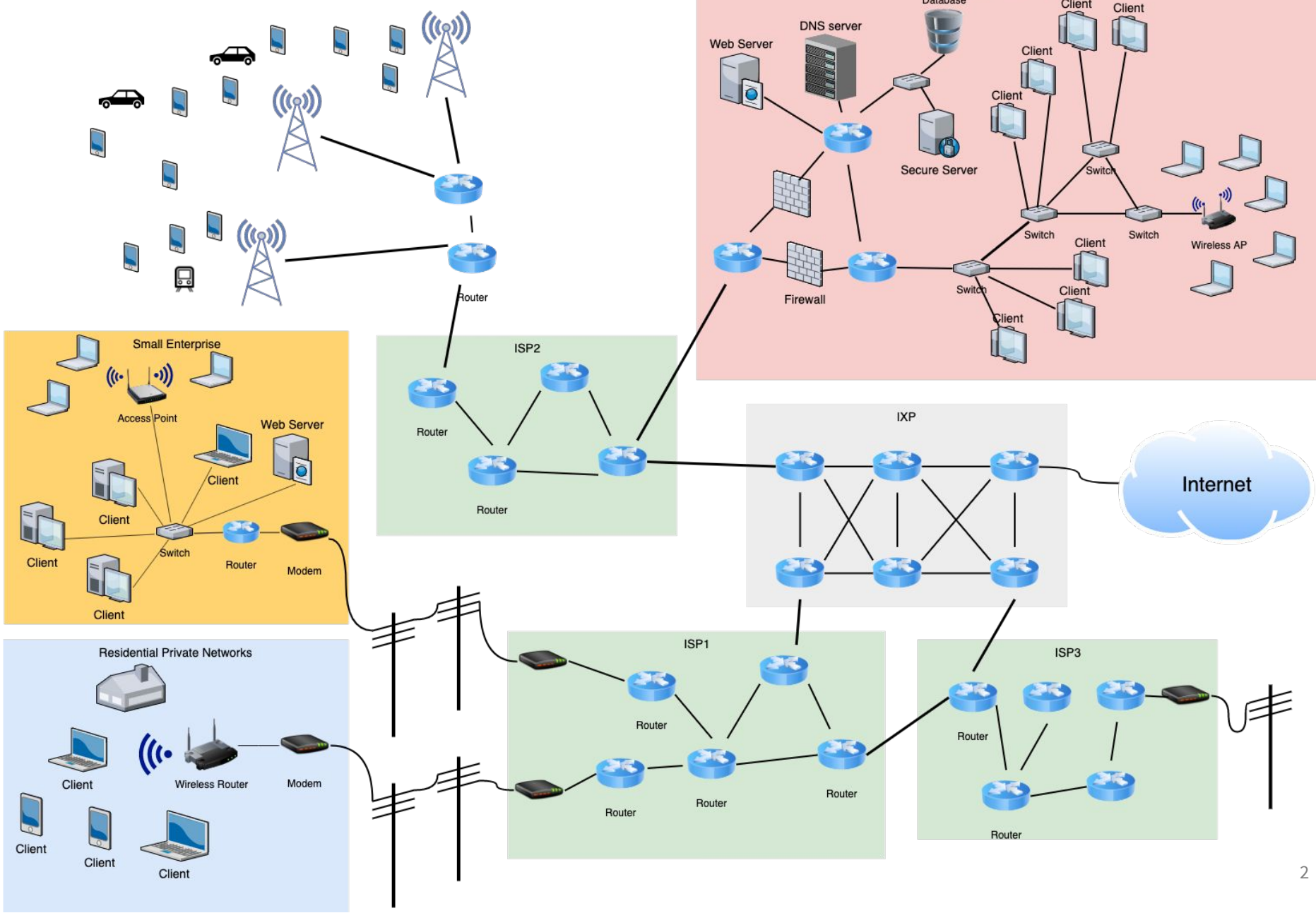
Networking

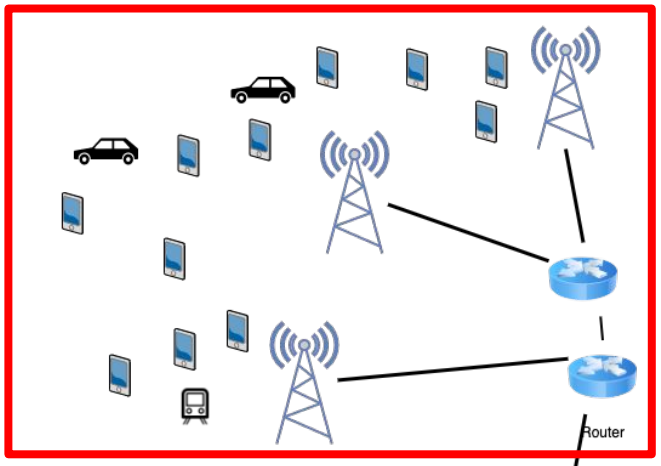
Semana 12 - Jueves 21 de noviembre de 2019

Redes

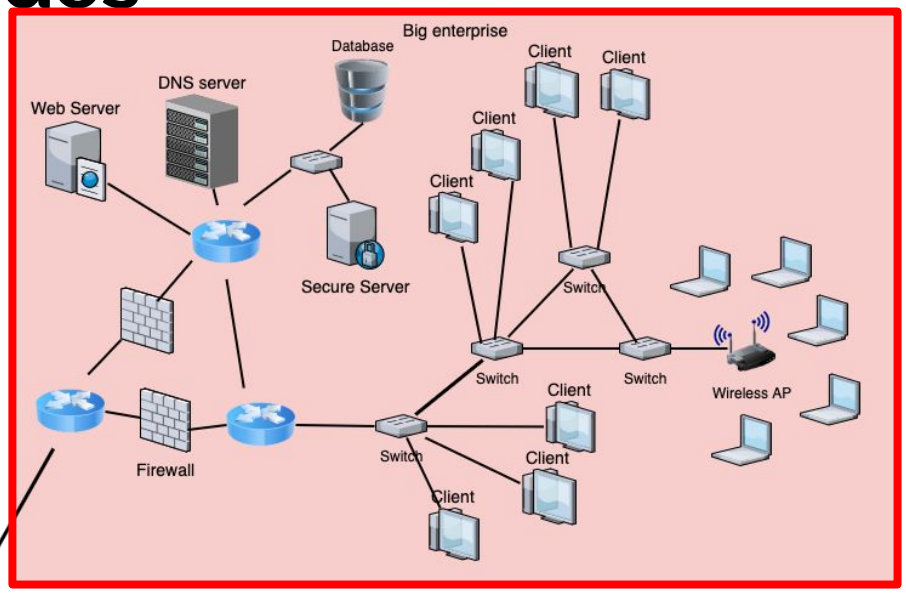

Router
Big enterprise
Database
DNS server
Web Server
Client
Client
Client
Client
Secure Server
Switch
Switch
Switch
Wireless AP
Client
Client
Client
Firewall
Switch
Small Enterprise
Access Point
Web Server
Client
Client
Client
Client
Switch
Router
Modem
ISP2
Router
Router
IXP
Internet
Residential Private Networks
Client
Wireless Router
Modem
Client
Client
Client
ISP1
Router
Router
Router
Router
ISP3
Router
Router

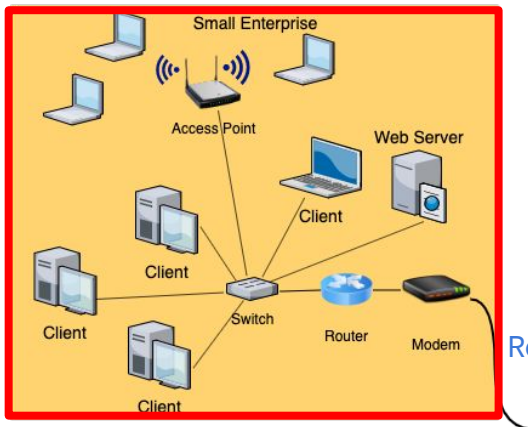
Subredes

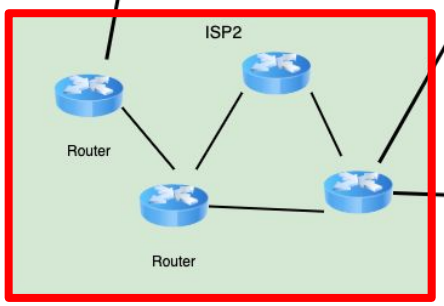
Red: conexión móvil

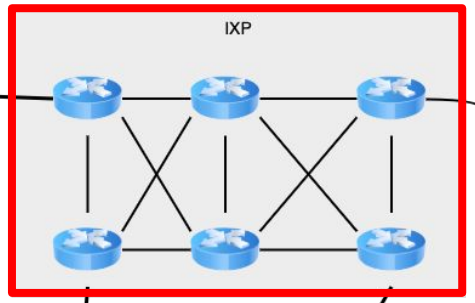


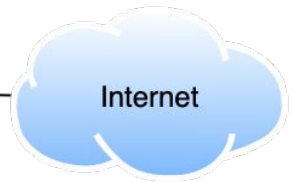
Red: empresa mediana-grande





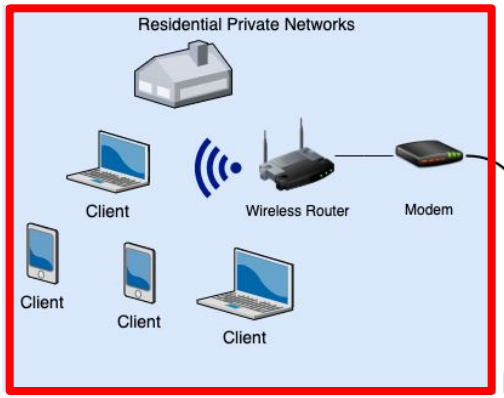


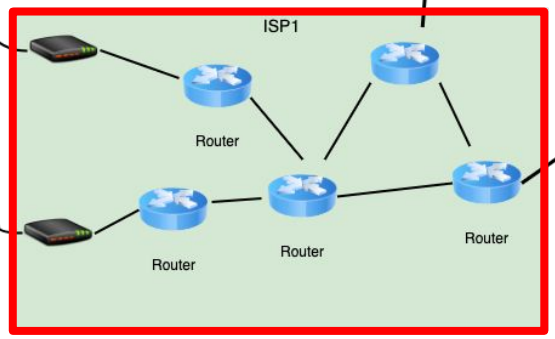


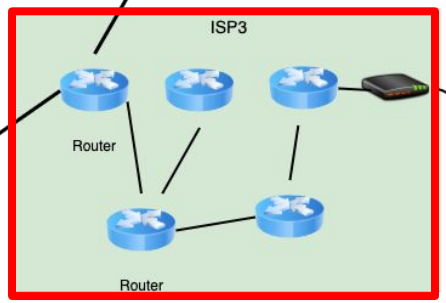


IXP o PIT: Punto de intercambio de Internet

Red: empresa pequeña





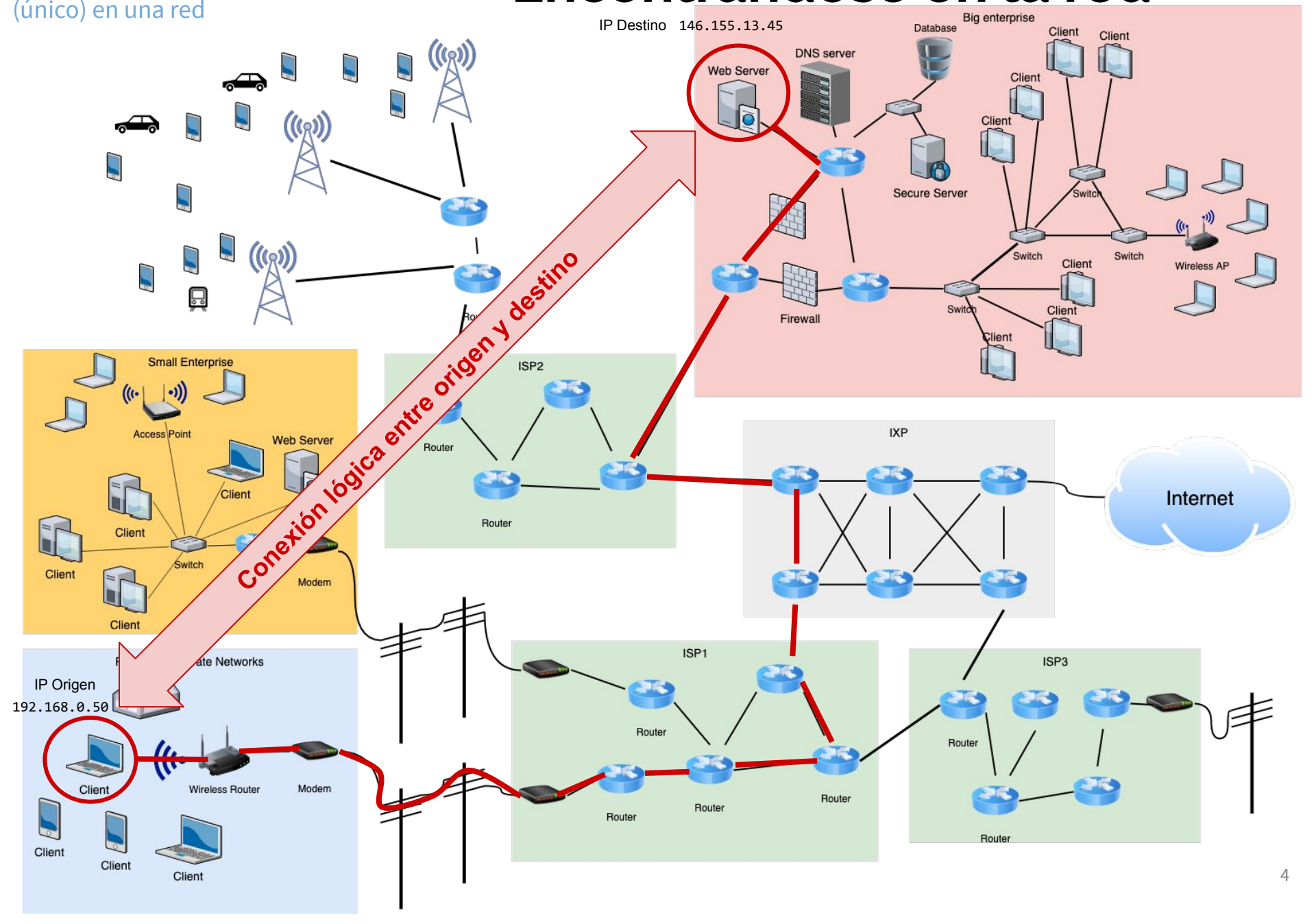


Red domiciliaria

ISP: Internet Service Provider (VTR, Movistar, Claro, Entel, etc)

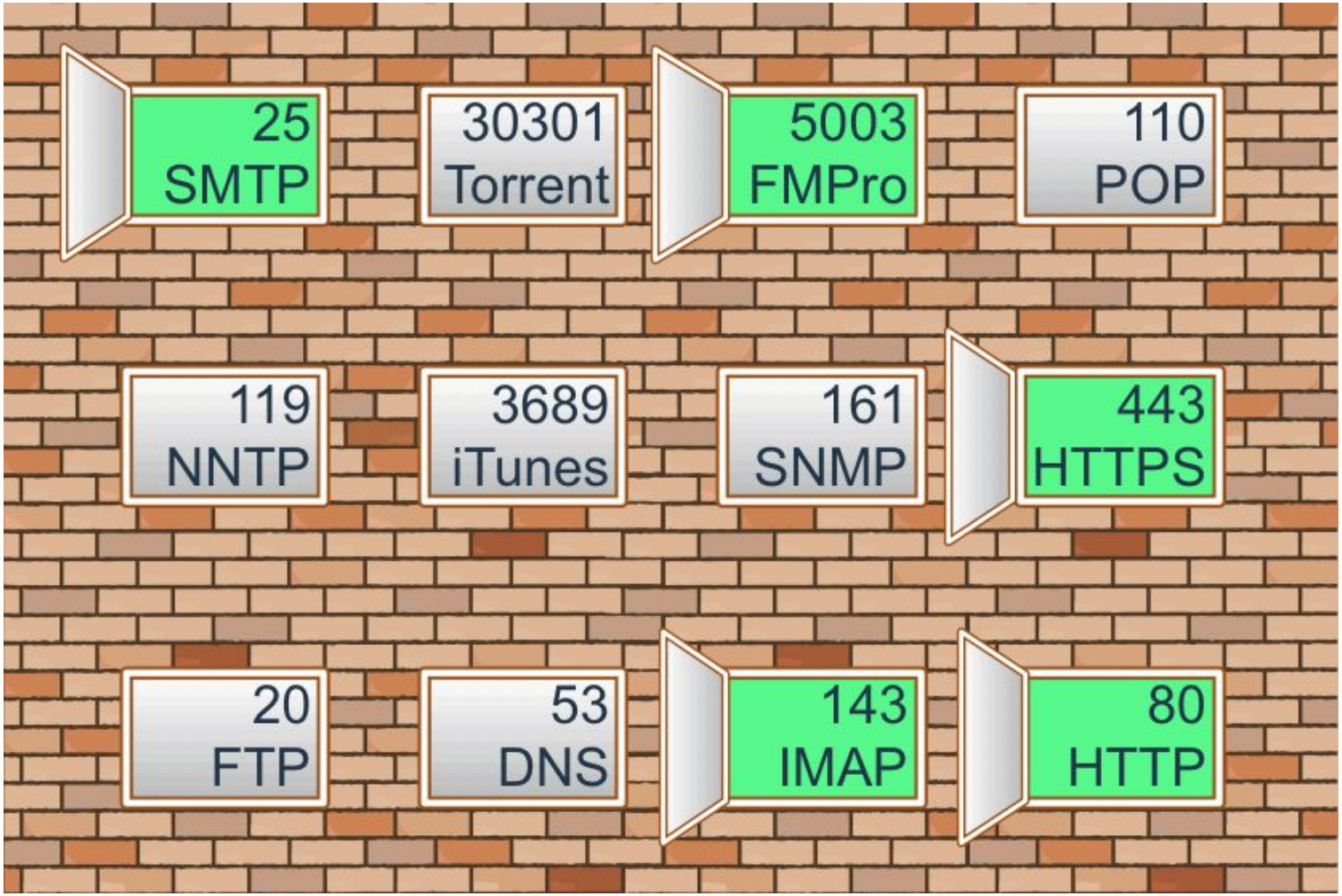
Dirección IP: permite encontrar un miembro (único) en una red

Encontrándose en la red

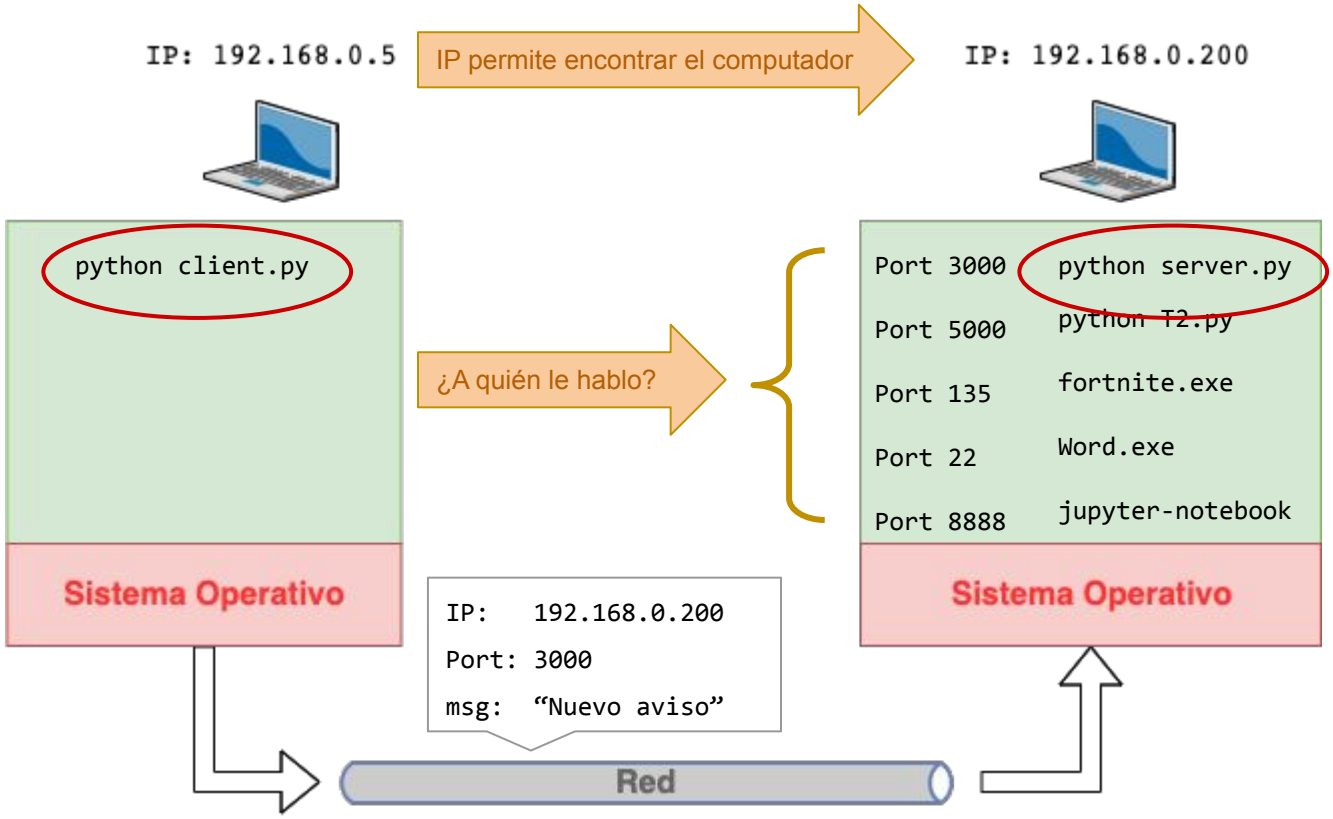


Puertos e IP

146.155.13.45



Conectando dos computadores



7: Capa de Aplicación
¿Qué mensaje envío?

6: Capa de Presentación

5: Capa de Sesión

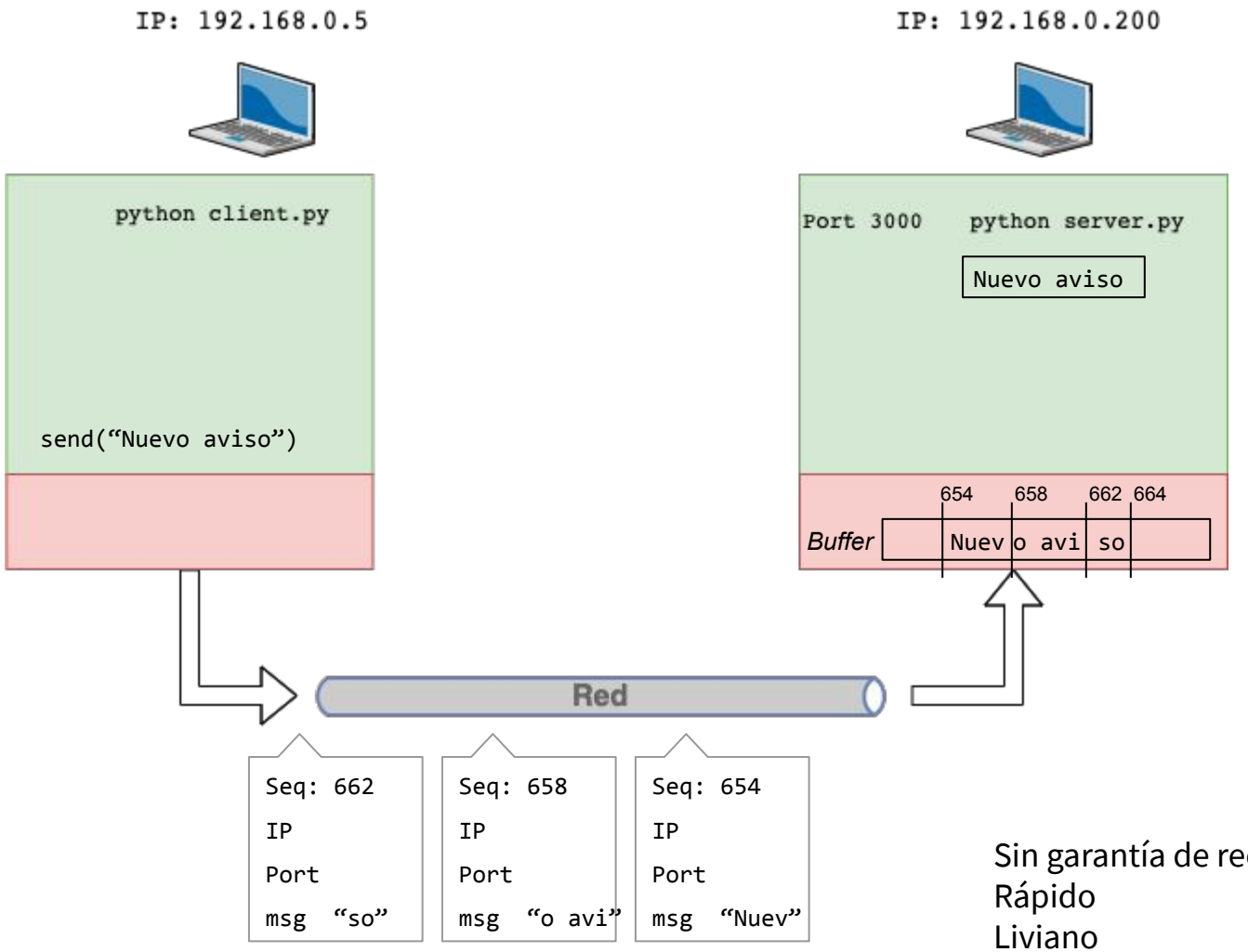
4: Capa de Transporte
¿cómo y a quién envío el mensaje?

3: Capa de Red
¿cómo encuentra el destinatario?

2: Capa de Enlace
¿cómo entrego datos a mi vecino?

1: Capa Física
¿cómo envío un byte?

Transporte: Protocolo UDP



7: Capa de Aplicación
¿Qué mensaje envío?

6: Capa de Presentación

5: Capa de Sesión

4: Capa de Transporte
¿cómo y a quién envío el mensaje?

3: Capa de Red
¿cómo encuentra el destinatario?

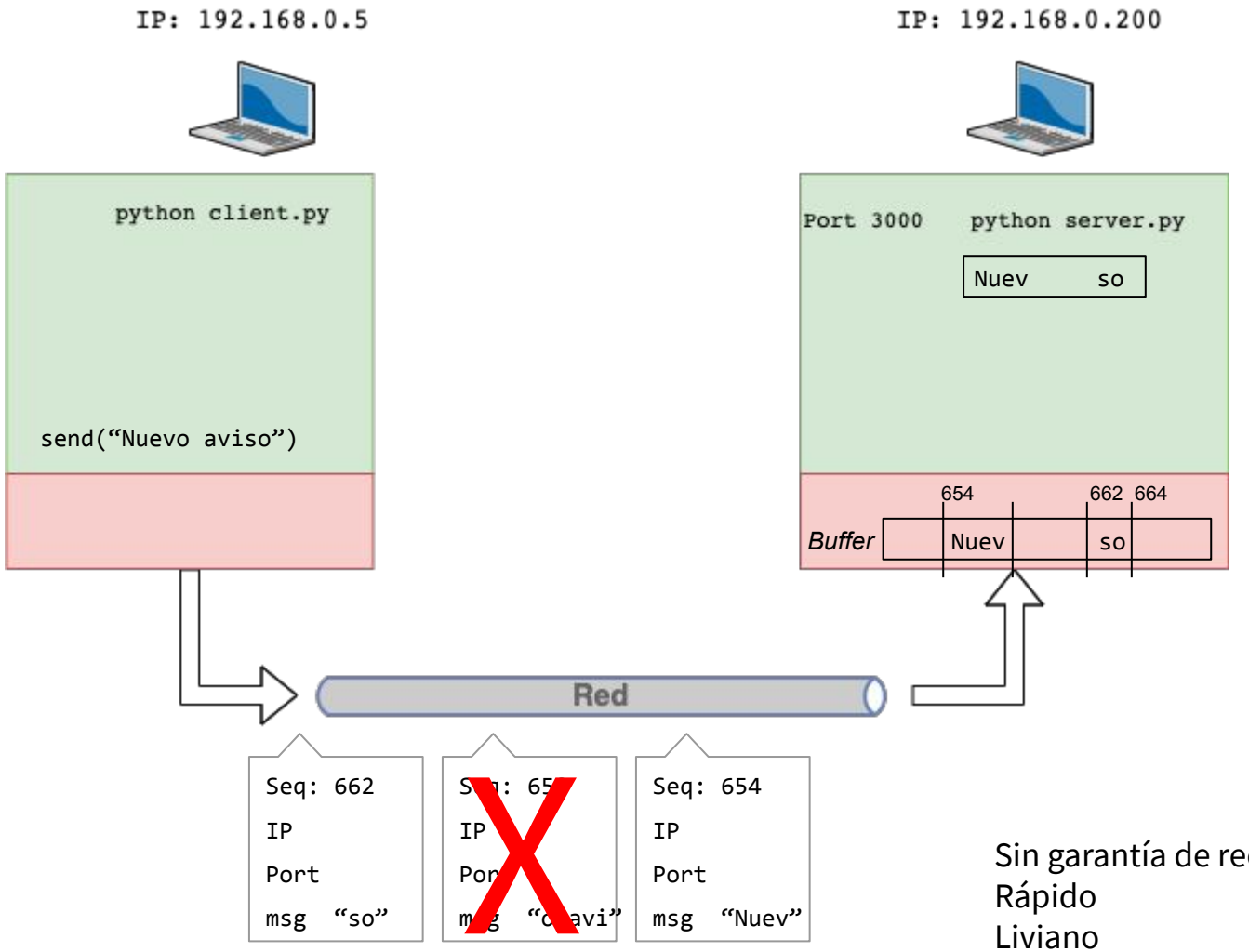
2: Capa de Enlace
¿cómo entrego datos a mi vecino?

1: Capa Física
¿cómo envío un byte?

Sin garantía de recepción
Rápido
Liviano

Transporte: Protocolo UDP

Si un paquete no llega, mensaje puede quedar incompleto



7: Capa de Aplicación
¿Qué mensaje envío?

6: Capa de Presentación

5: Capa de Sesión

4: Capa de Transporte
¿cómo y a quién envío el mensaje?

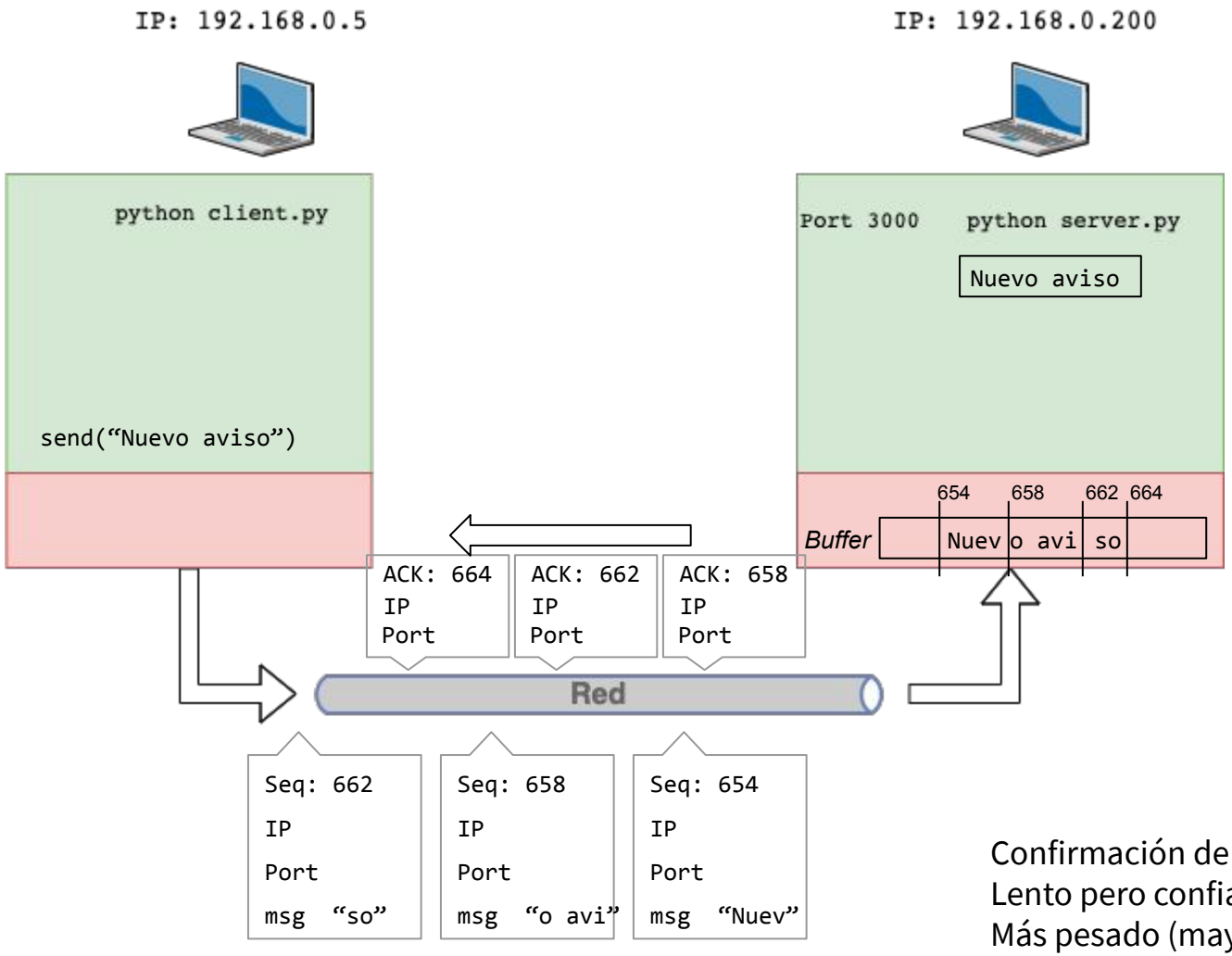
3: Capa de Red
¿cómo encuentra el destinatario?

2: Capa de Enlace
¿cómo entrego datos a mi vecino?

1: Capa Física
¿cómo envío un byte?

Sin garantía de recepción
Rápido
Liviano

Transporte: Protocolo TCP



7: Capa de Aplicación
¿Qué mensaje envío?

6: Capa de Presentación

5: Capa de Sesión

4: Capa de Transporte
¿cómo y a quién envío el mensaje?

3: Capa de Red
¿cómo encuentra el destinatario?

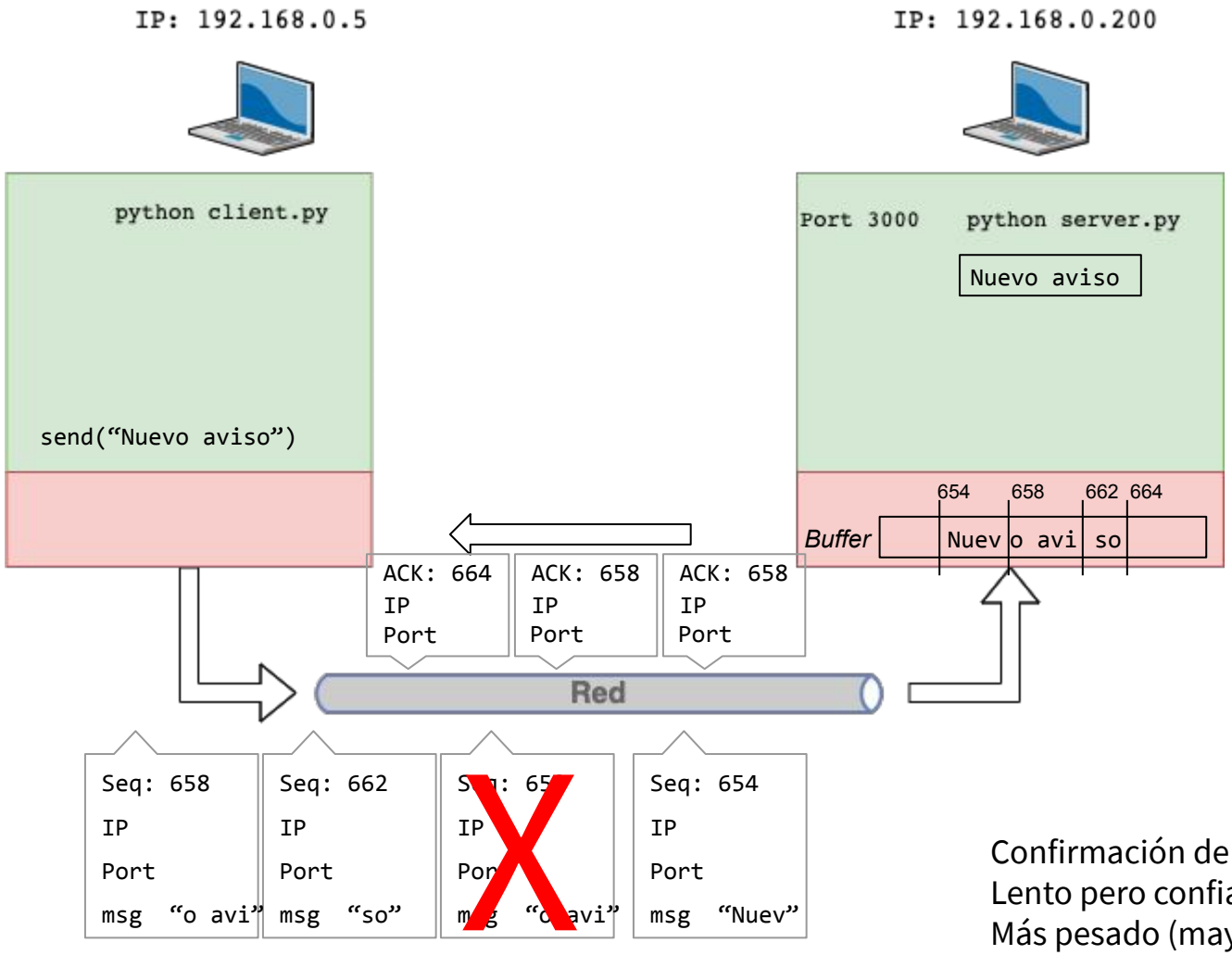
2: Capa de Enlace
¿cómo entrego datos a mi vecino?

1: Capa Física
¿cómo envío un byte?

Confirmación de recepción (ACK)
Lento pero confiable
Más pesado (mayor *overhead*)

Transporte: Protocolo TCP

Si un paquete no llega, se descubre gracias a los ACK, y se reenvía



7: Capa de Aplicación
¿Qué mensaje envío?

6: Capa de Presentación

5: Capa de Sesión

4: Capa de Transporte
¿cómo y a quién envío el mensaje?

3: Capa de Red
¿cómo encuentra el destinatario?

2: Capa de Enlace
¿cómo entrego datos a mi vecino?

1: Capa Física
¿cómo envío un byte?

Confirmación de recepción (ACK)
Lento pero confiable
Más pesado (mayor *overhead*)

Protocolo de aplicación

- ¿Cómo se envían mensajes en mi aplicación?
- ¿Qué deben tener los mensajes?
- ¿Cómo identificar los mensajes?

Ejemplo cliente-servidor

Cliente

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

sock.connect((host_recep, puerto_recep))
print("Conexión establecida.")

with open('enviar.bin', 'rb') as binf:
    datos = binf.read()
    largo = len(datos)
    sock.sendall(largo.to_bytes(4,
byteorder='big'))
    sock.sendall(datos)

print("¡Archivo enviado!")

print("Respuesta:", sock.recv(4096).
decode('utf-8'))

# Cerramos el socket.
sock.close()
```

Servidor

```
sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
sock.bind((host, puerto))
sock.listen()
print('Escuchando...')

sock_cl, _ = sock.accept()
print("Conexión entrante aceptada.")

largo = int.from_bytes(sock_cl.recv(4),
byteorder='big')
datos = bytearray()

while len(datos) < largo:
    leer = min(4096, largo - len(datos))
    recibidos = sock_cl.recv(leer)
    datos.extend(recibidos)

with open('recibido.bin','wb') as binf:
    binf.write(datos)

print("¡Archivo recibido!")
sock_cl.sendall("Gracias".encode('utf-8'))

# Cerramos los sockets.
sock_cl.close()
sock.close()
```

Ejemplo cliente-servidor

Cliente

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

sock.connect((host_recep, puerto_recep))
print("Conexión establecida.")

with open('enviar.bin', 'rb') as binf:
    datos = binf.read()
    largo = len(datos)
    sock.sendall(largo.to_bytes(4,
byteorder='big'))
    sock.sendall(datos)

print("¡Archivo enviado!")

print("Respuesta:", sock.recv(4096).
decode('utf-8'))

# Cerramos el socket.
sock.close()
```

Cliente y servidor crean sockets TCP

Servidor

```
sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
sock.bind((host, puerto))
sock.listen()
print('Escuchando...')

sock_cl, _ = sock.accept()
print("Conexión entrante aceptada.")

largo = int.from_bytes(sock_cl.recv(4),
byteorder='big')
datos = bytearray()

while len(datos) < largo:
    leer = min(4096, largo - len(datos))
    recibidos = sock_cl.recv(leer)
    datos.extend(recibidos)

with open('recibido.bin','wb') as binf:
    binf.write(datos)

print("¡Archivo recibido!")
sock_cl.sendall("Gracias".encode('utf-8'))

# Cerramos los sockets.
sock_cl.close()
sock.close()
```

Ejemplo cliente-servidor

Cliente

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

sock.connect((host_recep, puerto_recep))
print("Conexión establecida.")

with open('enviar.bin', 'rb') as binf:
    datos = binf.read()
    largo = len(datos)
    sock.sendall(largo.to_bytes(4,
byteorder='big'))
    sock.sendall(datos)

print("¡Archivo enviado!")

print("Respuesta:", sock.recv(4096).
decode('utf-8'))

# Cerramos el socket.
sock.close()
```

Servidor espera conexión.
Cliente solicita conexión.

Servidor obtiene socket adicional para el cliente.

Servidor

```
sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
sock.bind((host, puerto))
sock.listen()
print('Escuchando...')

sock_cl, _ = sock.accept()
print("Conexión entrante aceptada.")

largo = int.from_bytes(sock_cl.recv(4),
byteorder='big')
datos = bytearray()

while len(datos) < largo:
    leer = min(4096, largo - len(datos))
    recibidos = sock_cl.recv(leer)
    datos.extend(recibidos)

with open('recibido.bin','wb') as binf:
    binf.write(datos)

print("¡Archivo recibido!")
sock_cl.sendall("Gracias".encode('utf-8'))

# Cerramos los sockets.
sock_cl.close()
sock.close()
```

Ejemplo cliente-servidor

Cliente

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

sock.connect((host_recep, puerto_recep))
print("Conexión establecida.")

with open('enviar.bin', 'rb') as binf:
    datos = binf.read()
    largo = len(datos)
    sock.sendall(largo.to_bytes(4,
byteorder='big'))
    sock.sendall(datos)

print("¡Archivo enviado!")

print("Respuesta:", sock.recv(4096).
decode('utf-8'))

# Cerramos el socket.
sock.close()
```

Cliente envía 4 bytes con el tamaño, y luego el archivo.
Servidor recibe el tamaño, y lo usa para recibir *chunks* de 4096 bytes hasta completar el archivo.

Servidor

```
sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
sock.bind((host, puerto))
sock.listen()
print('Escuchando...')

sock_cl, _ = sock.accept()
print("Conexión entrante aceptada.")

largo = int.from_bytes(sock_cl.recv(4),
byteorder='big')
datos = bytearray()

while len(datos) < largo:
    leer = min(4096, largo - len(datos))
    recibidos = sock_cl.recv(leer)
    datos.extend(recibidos)

with open('recibido.bin','wb') as binf:
    binf.write(datos)

print("¡Archivo recibido!")
sock_cl.sendall("Gracias".encode('utf-8'))

# Cerramos los sockets.
sock_cl.close()
sock.close()
```

Ejemplo cliente-servidor

Cliente

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

sock.connect((host_recep, puerto_recep))
print("Conexión establecida.")

with open('enviar.bin', 'rb') as binf:
    datos = binf.read()
    largo = len(datos)
    sock.sendall(largo.to_bytes(4,
byteorder='big'))
    sock.sendall(datos)

print("¡Archivo enviado!")

print("Respuesta:", sock.recv(4096).
decode('utf-8'))

# Cerramos el socket.
sock.close()
```

Servidor

```
sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
sock.bind((host, puerto))
sock.listen()
print('Escuchando...')

sock_cl, _ = sock.accept()
print("Conexión entrante aceptada.")

largo = int.from_bytes(sock_cl.recv(4),
byteorder='big')
datos = bytearray()

while len(datos) < largo:
    leer = min(4096, largo - len(datos))
    recibidos = sock_cl.recv(leer)
    datos.extend(recibidos)

with open('recibido.bin','wb') as binf:
    binf.write(datos)

print("¡Archivo recibido!")
sock_cl.sendall("Gracias".encode('utf-8'))

# Cerramos los sockets.
sock_cl.close()
sock.close()
```

Servidor guarda archivo, y responde con mensaje para el cliente.

Cliente recibe el mensaje

Ejemplo cliente-servidor

Cliente

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

sock.connect((host_recep, puerto_recep))
print("Conexión establecida.")

with open('enviar.bin', 'rb') as binf:
    datos = binf.read()
    largo = len(datos)
    sock.sendall(largo.to_bytes(4,
byteorder='big'))
    sock.sendall(datos)

print("¡Archivo enviado!")

print("Respuesta:", sock.recv(4096).
decode('utf-8'))

# Cerramos el socket.
sock.close()
```

Cliente y servidor cierran sockets

Servidor

```
sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
sock.bind((host, puerto))
sock.listen()
print('Escuchando...')

sock_cl, _ = sock.accept()
print("Conexión entrante aceptada.")

largo = int.from_bytes(sock_cl.recv(4),
byteorder='big')
datos = bytearray()

while len(datos) < largo:
    leer = min(4096, largo - len(datos))
    recibidos = sock_cl.recv(leer)
    datos.extend(recibidos)

with open('recibido.bin','wb') as binf:
    binf.write(datos)

print("¡Archivo recibido!")
sock_cl.sendall("Gracias".encode('utf-8'))

# Cerramos los sockets.
sock_cl.close()
sock.close()
```

Protocolos de transporte

TCP

"Hi, I'd like to hear a TCP joke."
"Hello, would you like to hear a TCP joke?"
"Yes, I'd like to hear a TCP joke."
"OK, I'll tell you a TCP joke."
"Ok, I will hear a TCP joke."
"Are you ready to hear a TCP joke?"
"Yes, I am ready to hear a TCP joke."
"Ok, I am about to send the TCP joke. It will last 10 seconds, it has two characters, it does not have a setting, it ends with a punchline."
"Ok, I am ready to get your TCP joke that will last 10 seconds, has two characters, does not have an explicit setting, and ends with a punchline."
"I'm sorry, your connection has timed out.
...Hello, would you like to hear a TCP joke?"

UDP