

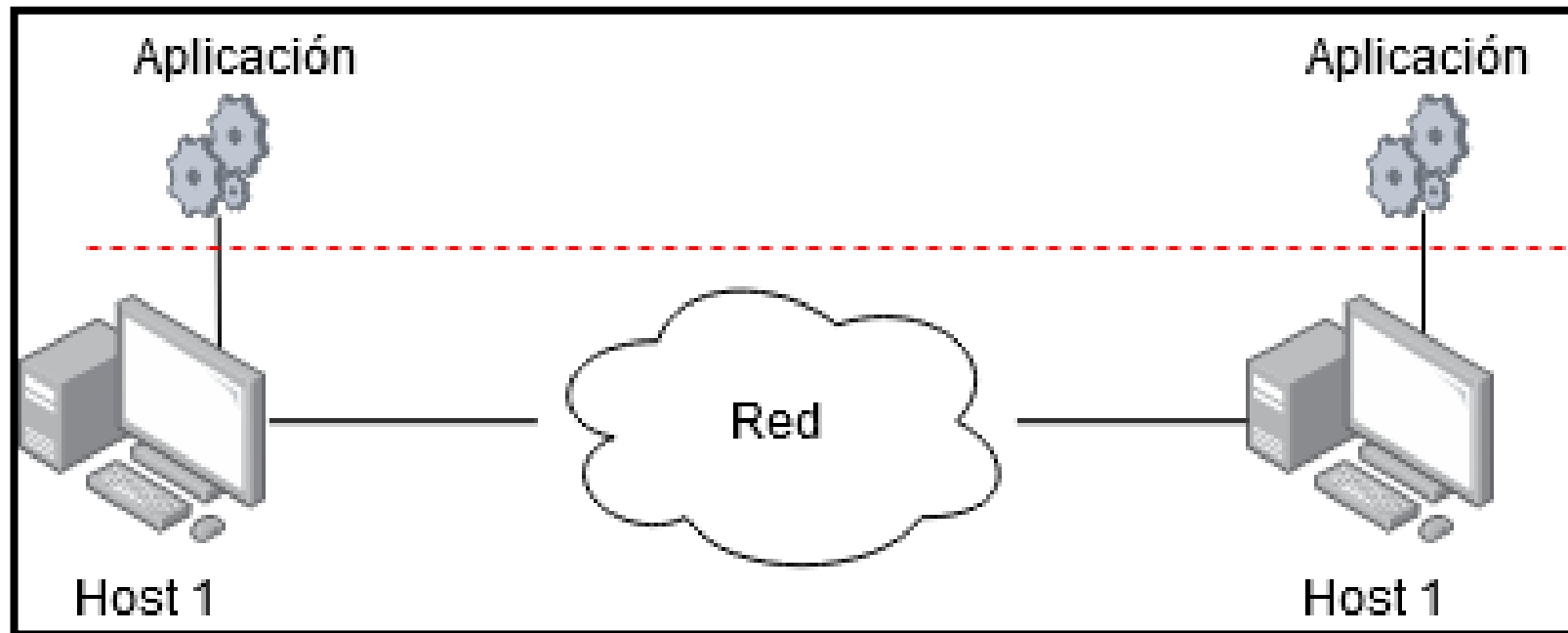
Sockets

Adaptación

Juan Felipe Muñoz Fernández

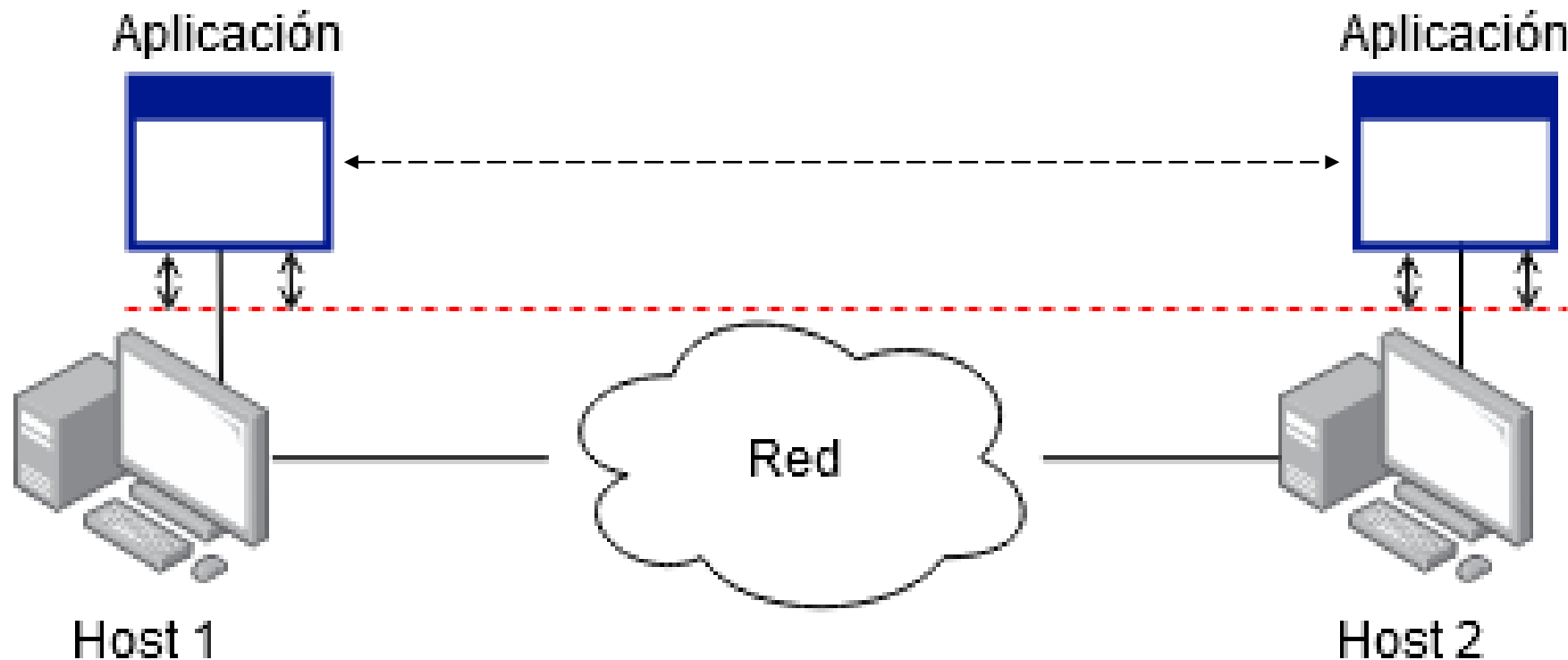
Interfaz Red – Aplicación

- Interfaz que define como las aplicaciones usan la red
- Recordemos...

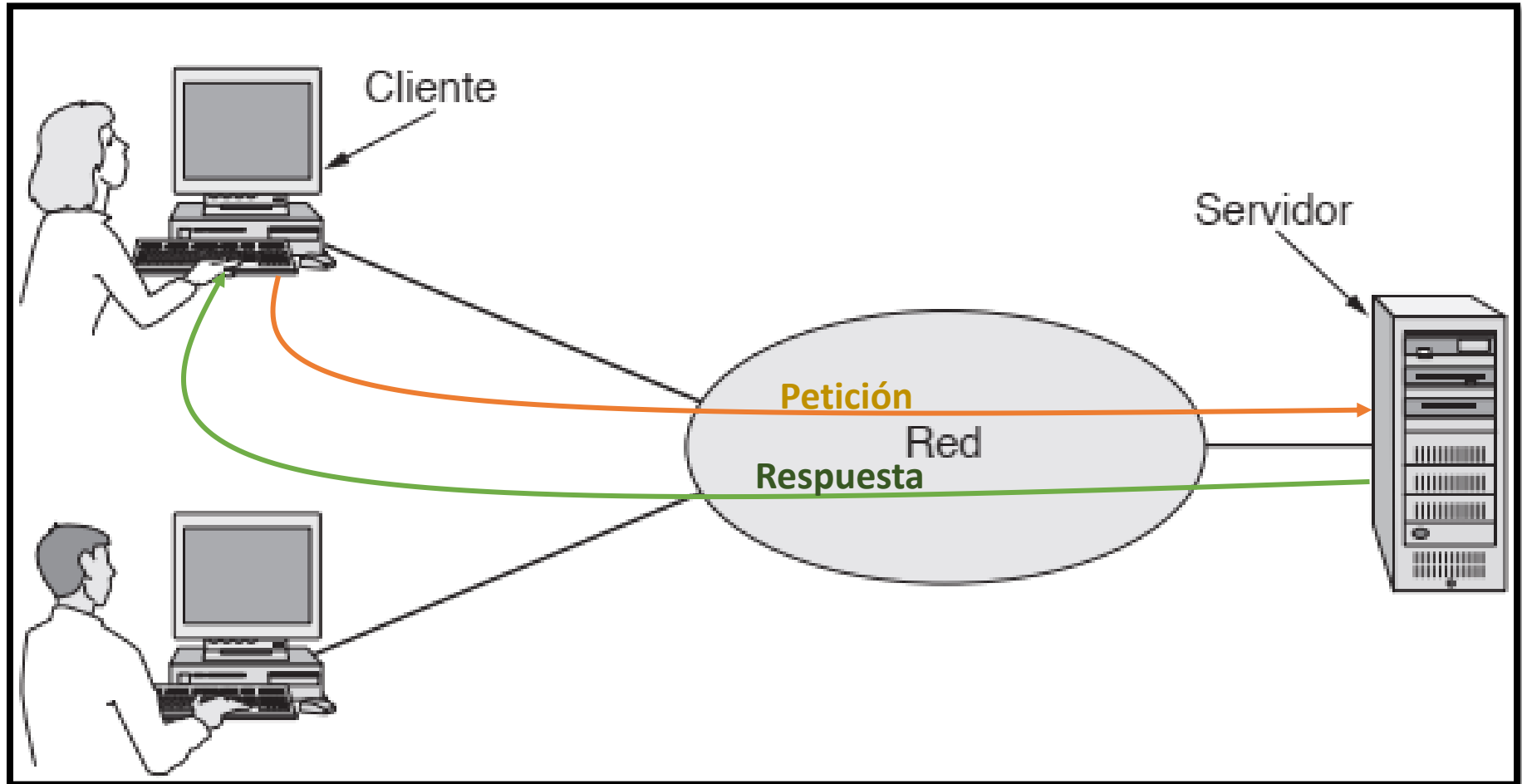


Interfaz Red – Aplicación

- Oculta los detalles de la red a las aplicaciones



Modelo cliente servidor



Aplicación cliente – servidor

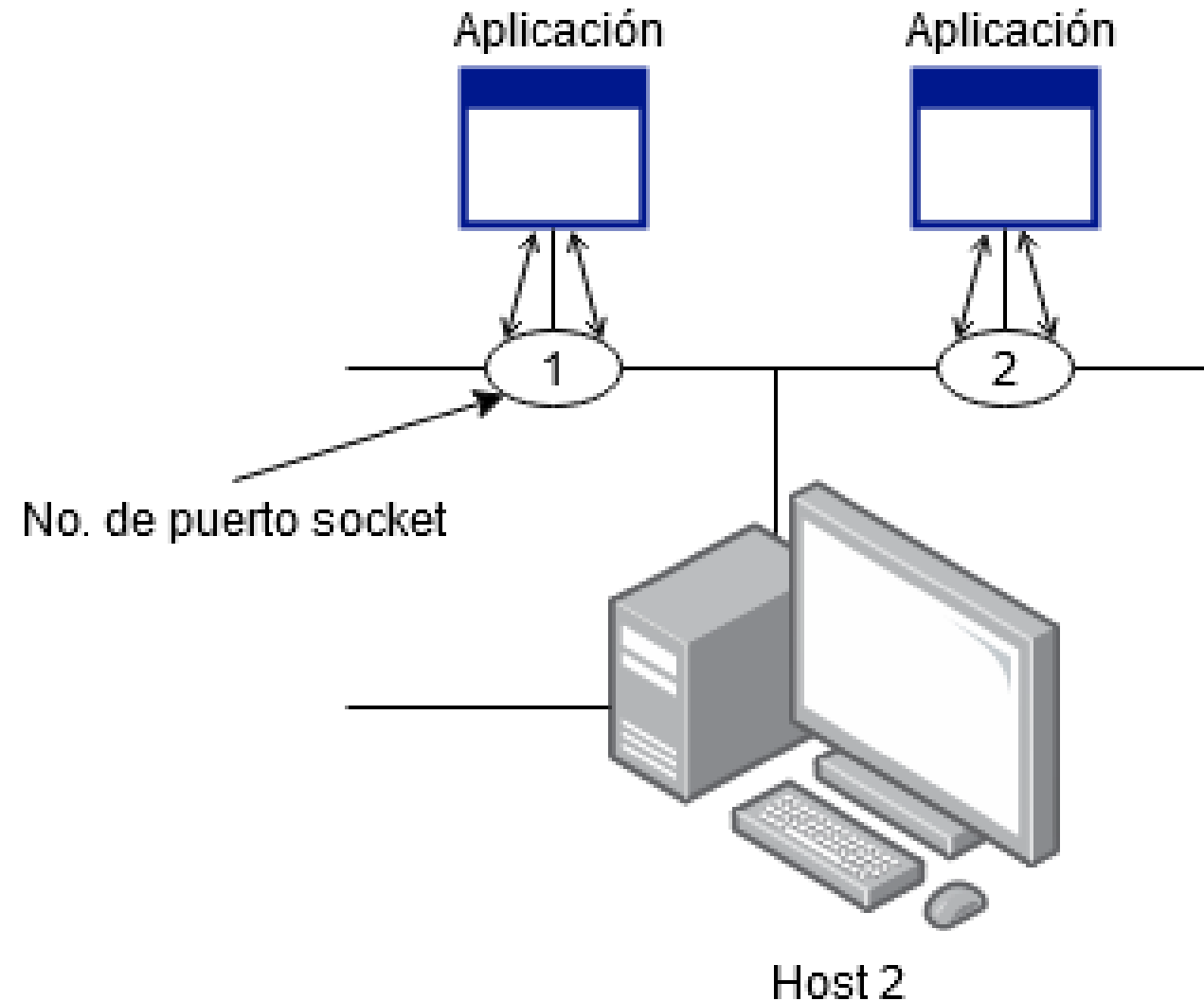
- Es el patrón básico de comunicaciones para muchas aplicaciones
 - Transferencia de archivos. P. Ej.: FTP
 - Obtener una página Web desde un servidor
 - Enviar una página a Web de acuerdo con una solicitud
 - Echo: enviar un mensaje y obtener el mismo mensaje de respuesta

API Sockets

- **Abstracción para que las app's usen la red**
 - API disponible en la mayoría de SO's desde Unix Berkeley 1983
 - Winsock → Windows
 - Sockets → Unix-Like
 - Portabilidad a través de SO's.
- **Dos servicios de conexión**
 - Basado en flujos: conexión confiable, flujos de bytes
 - Basado en datagramas: no confiable, envío de mensajes independientes

API Sockets

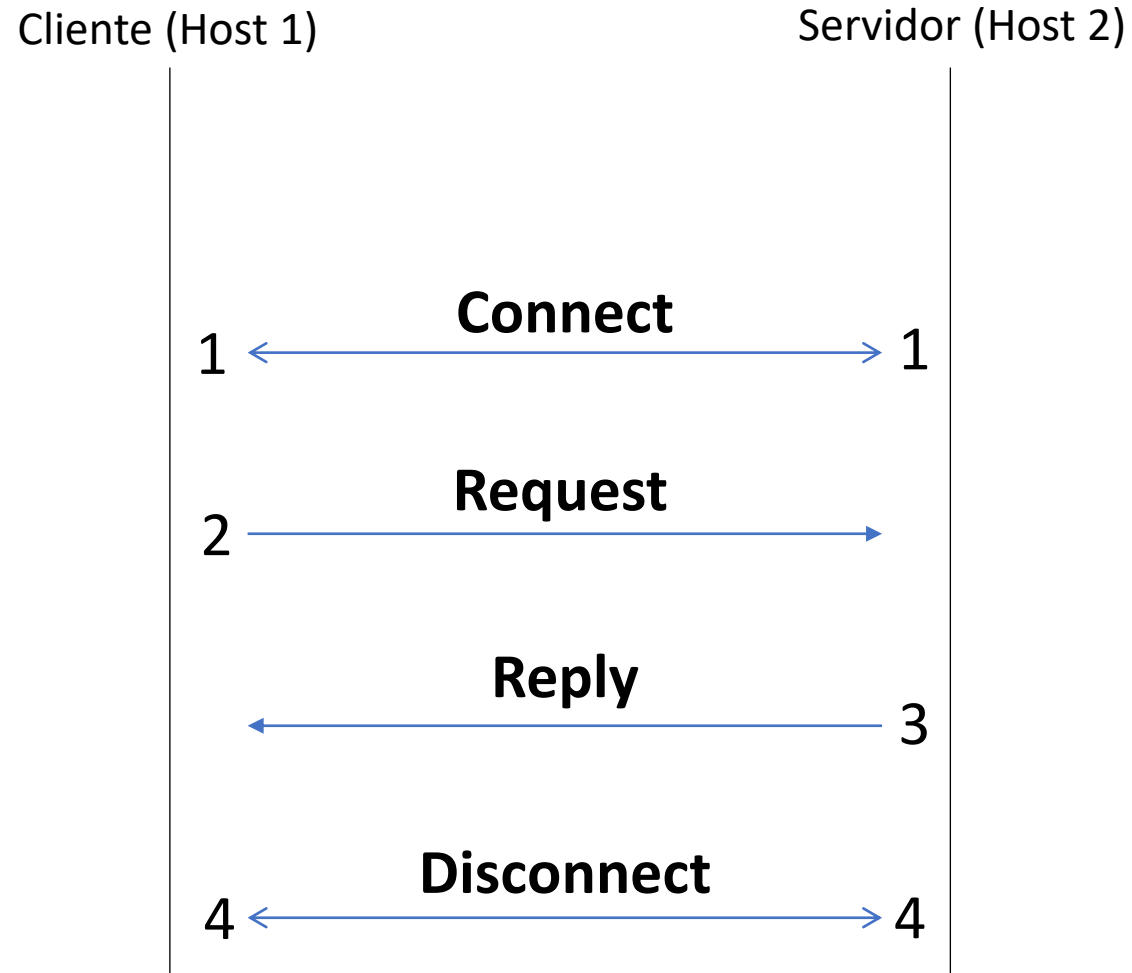
- Socket permite a las aplicaciones *conectarse* a la red local en diferentes puertos
- Socket punto de “enganche” de la aplicación a la red.
- Números de puertos bien conocidos para las aplicaciones.



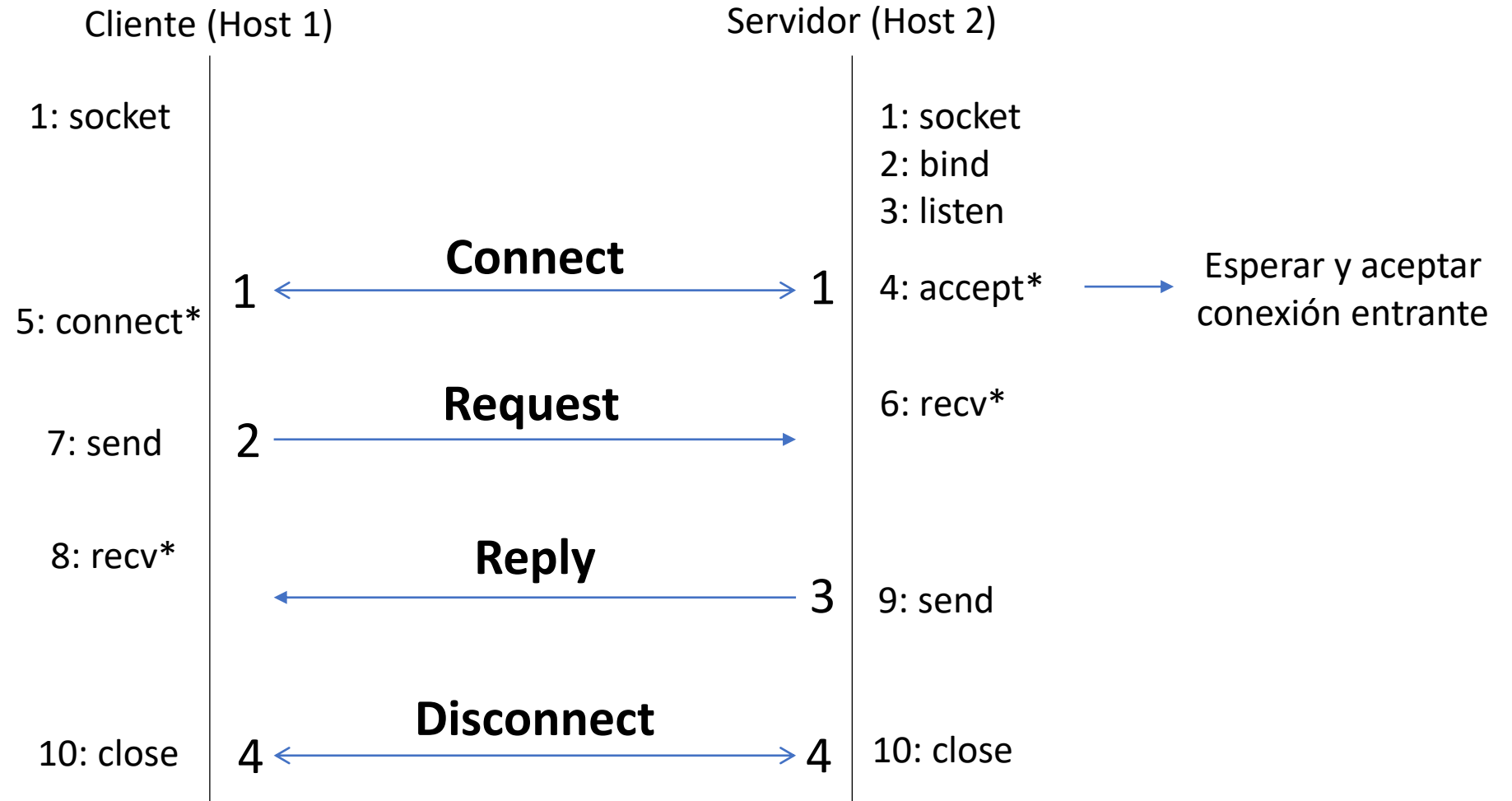
API Sockets – Primitivas de servicio

PRIMITIVA	SIGNIFICADO
SOCKET	Crea un <i>endpoint</i> para la comunicación
BIND	Asocia una dirección local con la estructura (C, C++) socket
LISTEN	Anuncia la disponibilidad para atender solicitudes
ACCEPT	Establece una conexión entrante (pasiva)
CONNECT	Intento por establecer una conexión (activa)
SEND	Envía datos sobre la conexión
RECEIVE	Recibe datos desde la conexión
CLOSE	Cierra la conexión

Uso del API Sockets



Uso del API Sockets



Patrón de cliente

```
socket()          /* crear el socket */
getaddrinfo()     /* Nombre del servidor y puerto */
connect()         /* Conectar al servidor [bloqueo] */
...
send()            /* Enviar solicitud */
recv()            /* Esperar respuesta [bloqueo] */
...
close()           /* Cerrar conexión (eventualmente) */
```

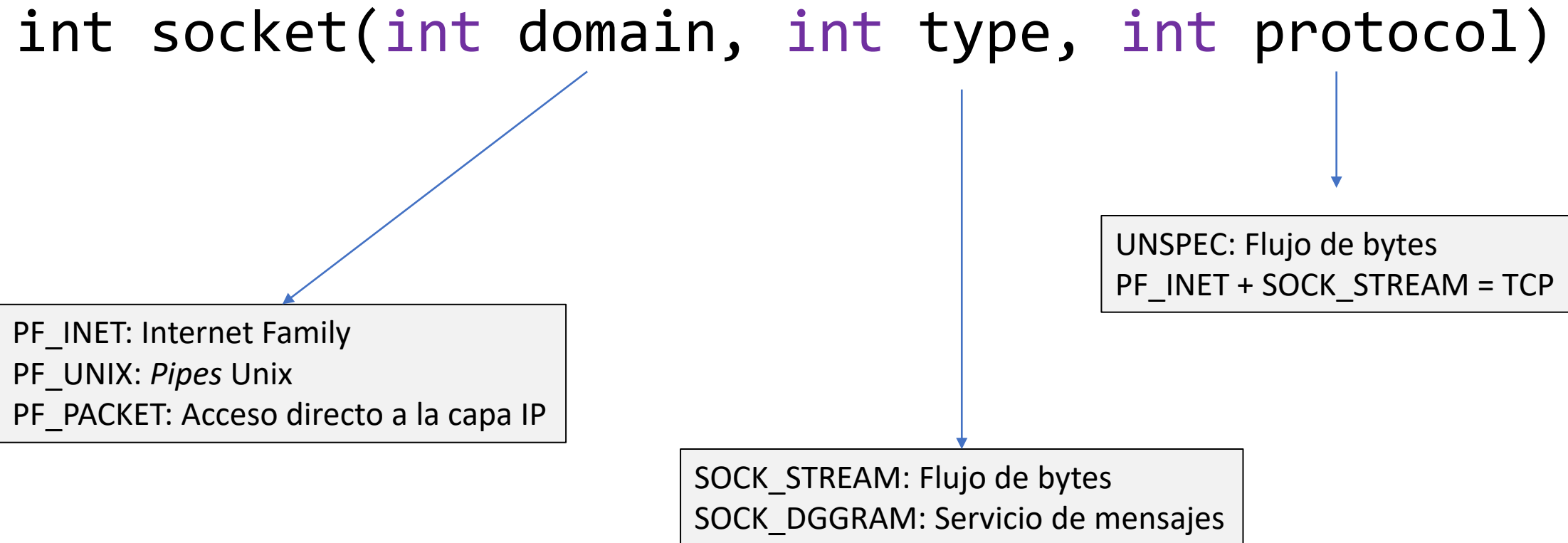
Patrón de servidor

```
socket()          /* crear el socket */
getaddrinfo()     /* Puertos a la escucha en el host */
bind()            /* Asociar puertos a socket */
listen()          /* Prepararse para aceptar conexiones */
accept()          /* Esperar por conexión [bloqueo] */
...
recv()            /* Esperar por solicitud */
...
send()            /* Enviar respuesta */
close()           /* Cerrar conexión (eventualmente) */
```

Loop

API Sockets

```
int socket(int domain, int type, int protocol)
```



PF_INET: Internet Family
PF_UNIX: *Pipes* Unix
PF_PACKET: Acceso directo a la capa IP

SOCK_STREAM: Flujo de bytes
SOCK_DGRAM: Servicio de mensajes

UNSPEC: Flujo de bytes
PF_INET + SOCK_STREAM = TCP

API Sockets: Llamadas desde el servidor

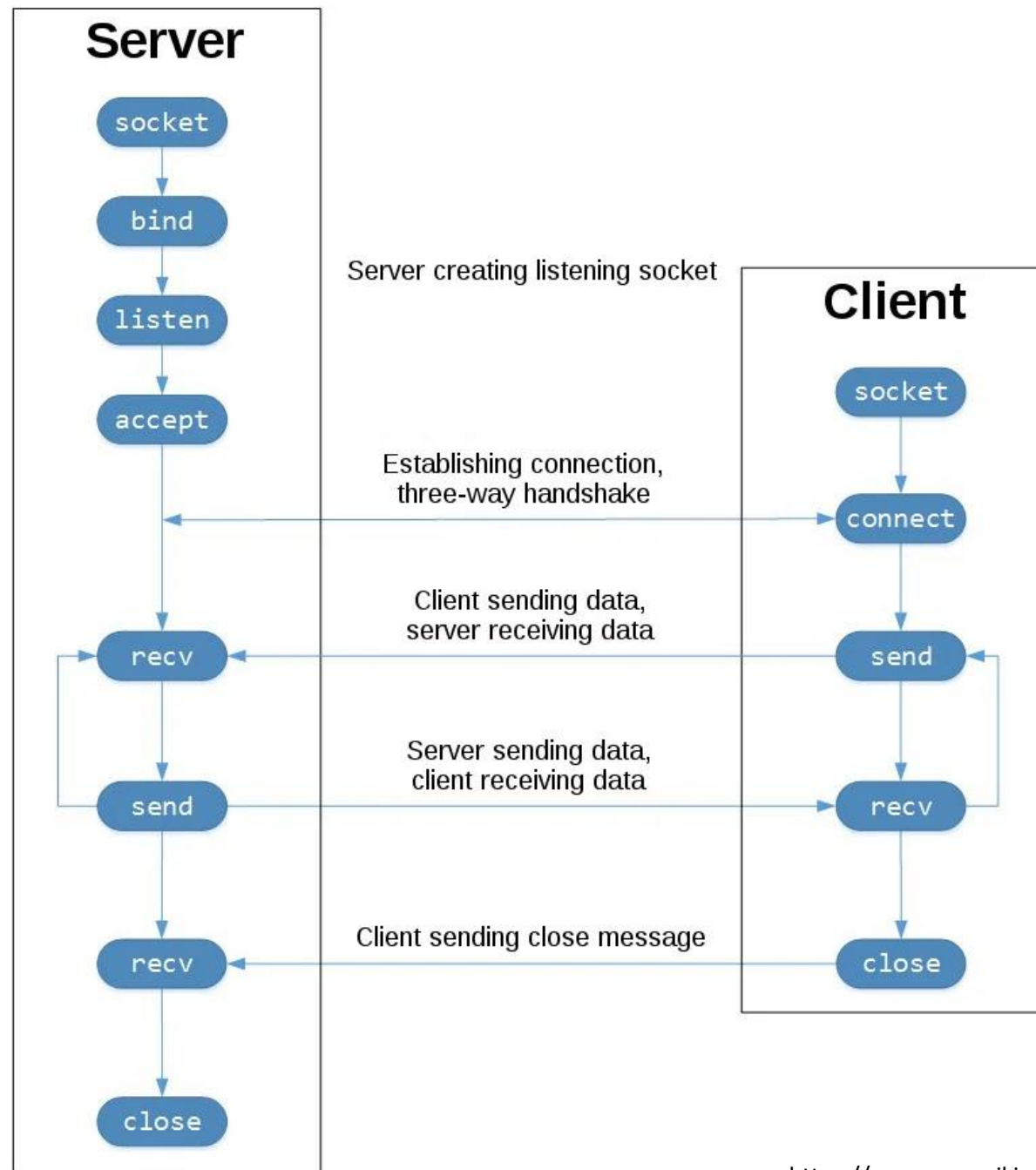
- `int bind(int socket, struct sockaddr *address, int addr len)`
 - Enlaza el socket a una dirección y puerto a través de la estructura sockaddr
- `int listen(int socket, int backlog)`
 - Control de las conexiones que están pendientes por atender
- `int accept(int socket, struct sockaddr *address, int *addr len)`
 - Abre la conexión de manera pasiva
 - Operación bloqueante
 - No retorna hasta que el otro extremo ha establecido la conexión

API Sockets: Llamadas desde el cliente

- `int connect(int socket, struct sockaddr *address, int addr len)`
 - La operación se bloquea si es TCP
 - TCP: Se establece conexión y luego se envían los datos
 - `address`: contiene la dirección remota del otro extremo
 - Al cliente no le importa que número de puerto está usando él. El SO usualmente se lo asigna
 - Se asigna número de puerto tanto para la aplicación cliente como para la aplicación destino

API Sockets

- `int send(int socket, char *message, int msg len, int flags)`
- `int recv(int socket, char *buffer, int buf len, int flags)`
 - Enviar y recibir datos a través del socket
 - Requieren dimensionamiento de búffer cuando se reciben
 - Flags para el control de la operación



Servidor en Python

```
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('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data:
                break
            conn.sendall(data)
```

Cliente en Python

```
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('Received', repr(data))
```

Importante

- Todo lo que se desee transmitir en el ejemplo anterior debe convertirse a bytes.

```
# Convertir un string a bytes
datos_enviar = 'Hola mundo!'
bytes_enviar = str.encode(datos_enviar)
```

- Para convertir los bytes a string

```
# Convertir los datos recibidos a string
datos_recv = data.decode()
```

- La variable `addr` es una tupla de datos tipo `string`

```
addr[0] # Dirección IP
addr[1] # No. de puerto
```

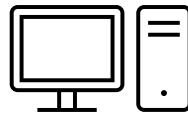
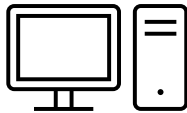
Demo con Python

- Diagrama de topología

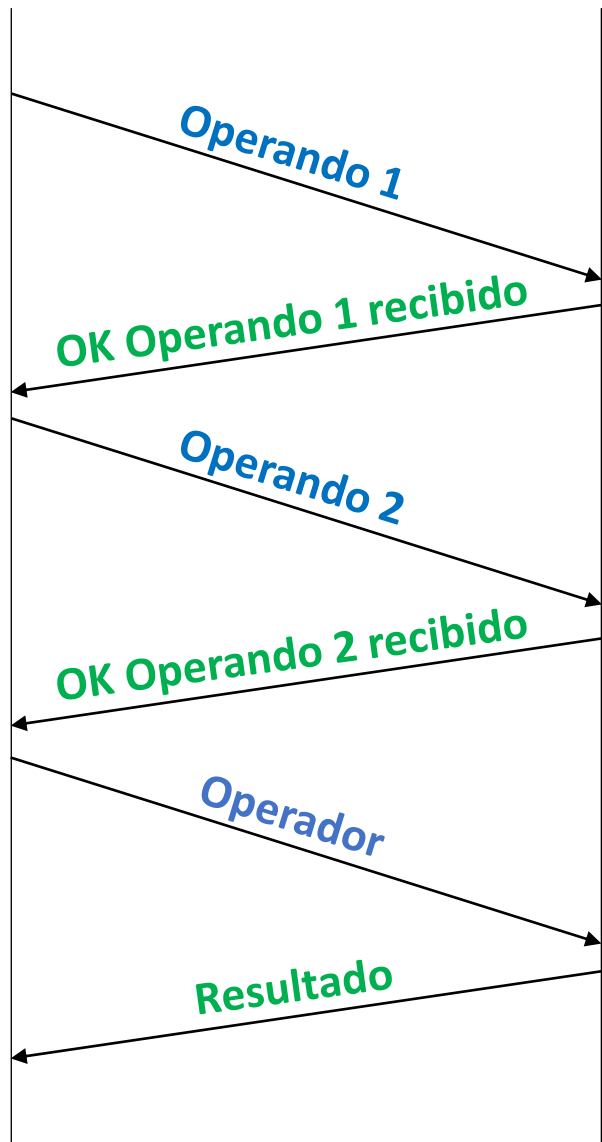
Ejercicio

- Diseñe un protocolo para una calculadora simple en un modelo de aplicación **cliente – servidor** e implemente la aplicación
 - El servidor debe quedar a la escucha en el puerto **TCP 51216**
 - Por cada mensaje que envía el cliente, se obtiene una respuesta del servidor
 - El cliente especifica la operación **en posfijo**: primero operandos y luego operador. **Máximo dos operandos por transacción con el servidor.**
 - El cliente envía **un operando** a la vez.
 - Por cada operando recibido, se envía una respuesta al cliente: OK o Fallo.
- **¿Qué hacer frente al error?**
- El cliente envía el operador una vez haya enviado correctamente los operandos y obtiene el **resultado desde el servidor**.

Cliente



Servidor



Lecturas recomendadas

- Socket Programming in C/C++:
<https://www.geeksforgeeks.org/socket-programming-cc/>
- Socket Programming in Python:
<https://www.geeksforgeeks.org/socket-programming-python/>