Clase 4
Named FIFOS - Networking

Autor: Esp. Ing. Ernesto Gigliotti - Lic. Danilo Zecchin

Named FIFOs

- Un pipe con nombre
- Se utiliza un solo file descriptor
- Cualquier proceso puede acceder para R/W
- Aparece como un archivo en el filesystem
- Soluciona limitación de pipe: Cualquier proceso tiene acceso

MKNOD

- Comando / Función
- Crea un archivo de un tipo especial
- Se utilizaba para crear los devices de caracter o bloque en /dev

Name	Description
S_IFIFO	FIFO-special
S_IFCHR	Character-special (non-portable)
S_IFDIR	Directory (non-portable)
S_IFBLK	Block-special (non-portable)
S_IFREG	Regular (non-portable)

Creando FIFO

- Utilizamos la función **mknod()**

```
int mknod(const char *pathname, mode_t mode, dev_t dev);
```

- Primer argumento: Nombre del archivo
- Segundo argumento: tipo + permisos
- Tercer argumento: Device number. No se utiliza con FIFOs

```
mknod("myfifo", S_IFIFO | 0644 , 0);
```

Writer

```c
#define FIFO_NAME "myfifo"
int main(void)
{
   char s[BUFFER_SIZE];
   int32_t bytesWrote, fd;

   mknod(FIFO_NAME, S_IFIFO | 0666, 0);

   printf("waiting for readers...\n");
   fd = open(FIFO_NAME, O_WRONLY);
   printf("got a reader--type some stuff\n");

   while (1) {
      fgets(s,BUFFER_SIZE,stdin);
      if ((bytesWrote = write(fd, s, strlen(s))) == -1)
         perror("write");
      else
         printf("writer: wrote %d bytes\n", bytesWrote);
   }
```

Probamos Writer con cat

- Ejecutamos el writer.
- Se creará el archivo myfifo

```
prw-rw-r-- 1 ernesto ernesto     0 mar  4 16:04 myfifo
```

- Leemos el archivo con el comando “cat”

```
> cat myfifo
```

Reader

```
#define FIFO_NAME "myfifo"
int main(void)
{
   char s[BUFFER_SIZE];
   int32_t bytesRead, fd;
   mknod(FIFO_NAME, S_IFIFO | 0666, 0);
   printf("waiting for writers...\n");
   fd = open(FIFO_NAME, O_RDONLY);
   printf("got a writer\n");

   do {
      if ((bytesRead = read(fd, s, BUFFER_SIZE)) == -1)
         perror("read");
      else {
         s[bytesRead] = '\0';
         printf("Read %d bytes: \"%s\"\n", bytesRead, s);
      }
   }
   while (bytesRead > 0);
```

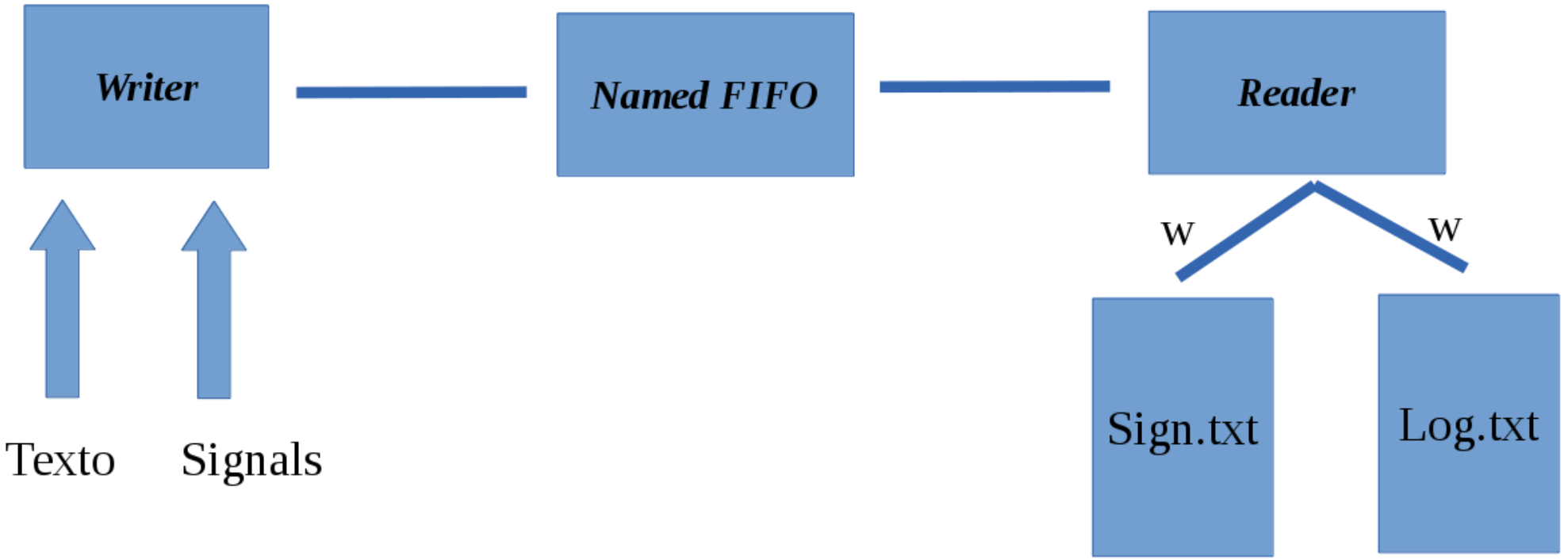
Finalización

- Si finalizamos el writer:
  - En el reader, la función read devolverá 0 (EOF)
  - La variable num va a ser cero y se sale del while y termina el programa.

- Si finalizamos el reader:
  - En el writer, al ejecutar write(), se recibirá una signal SIGPIPE y el programa terminará.

Trabajo Práctico

- Entrega: hasta clase 6
- Arquitectura



Networking

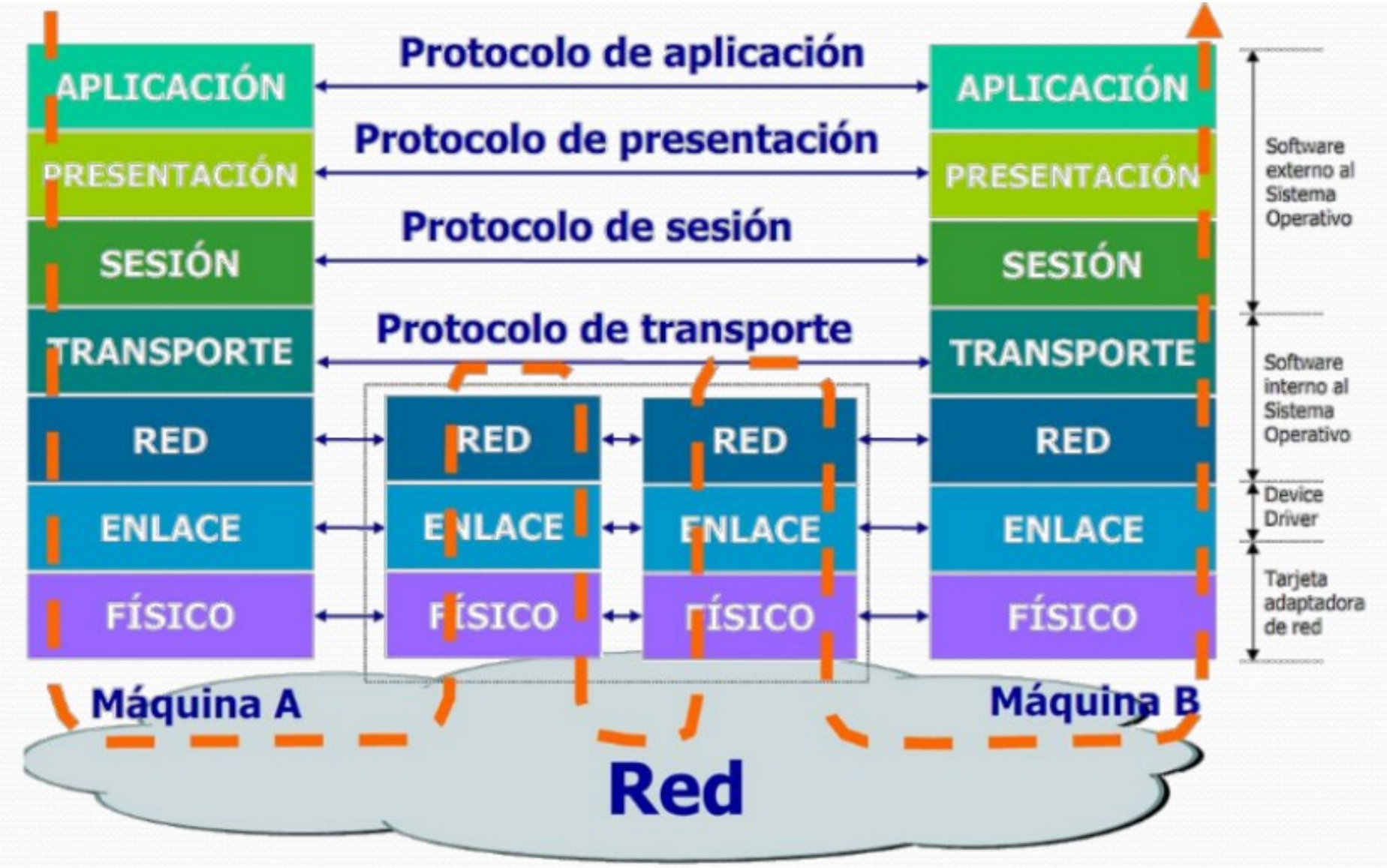
Redes de datos

- Vinculan recursos y datos
- Sistemas distribuidos
- Abstracción del hardware
- Oculta red física: equipos y medios de transmisión
- Capa de software del sistema operativo

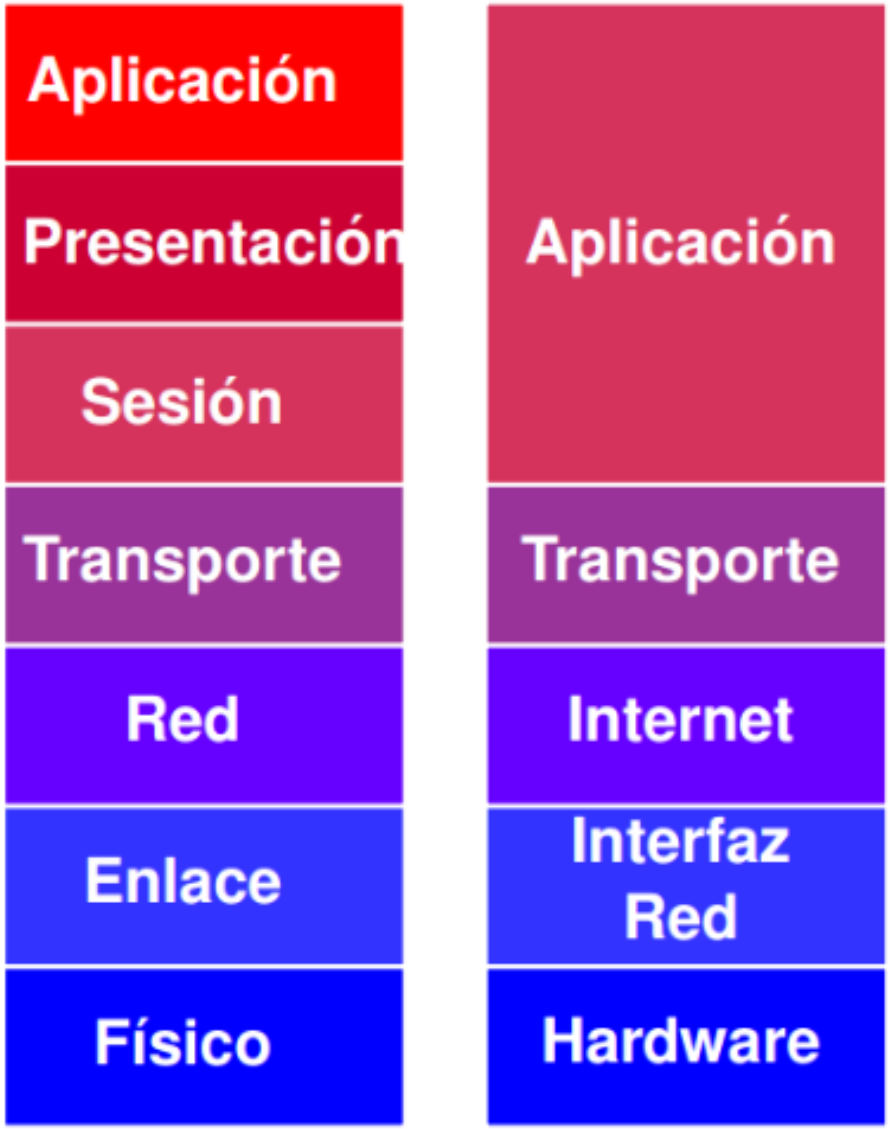
Modelo de capas

Aplicación	Interfaz con el usuario del mas alto nivel. Maquinas virtuales, APIs, Acceso a servicios.
Presentación	Acondicionamiento de la información hacia o desde la aplicación. Encriptado/Desencriptado, Compresión/Descompresión, etc
Sesión	Establecimiento de sesiones entre nodos para transmisión bidireccional. Ej: HTTP, HTTPS, SSH, FTP, SFTP, NFS
Transporte	Transmisión confiable de segmentos entre puntos de una red, multiplexado y acuse de recibo. Ej: TCP, UDP, MBF
Red	Transmisión de paquetes entre nodos en una internet, direccionamiento único, control y enrutamiento. Ej: IPv4, IPv6, IPSEC, AppleTalk
Enlace	Garantiza la transmisión de frames entre dos nodos conectados por el mismo medio físico, sin errores.Ej: IEEE 802.2, L2TP, LLDP, MAC, PPP
Físico	Es el Nivel en el que las señales se acondicionan para viajar por el medio físico de transmisión (el cual no está incluido).

Modelo de capas

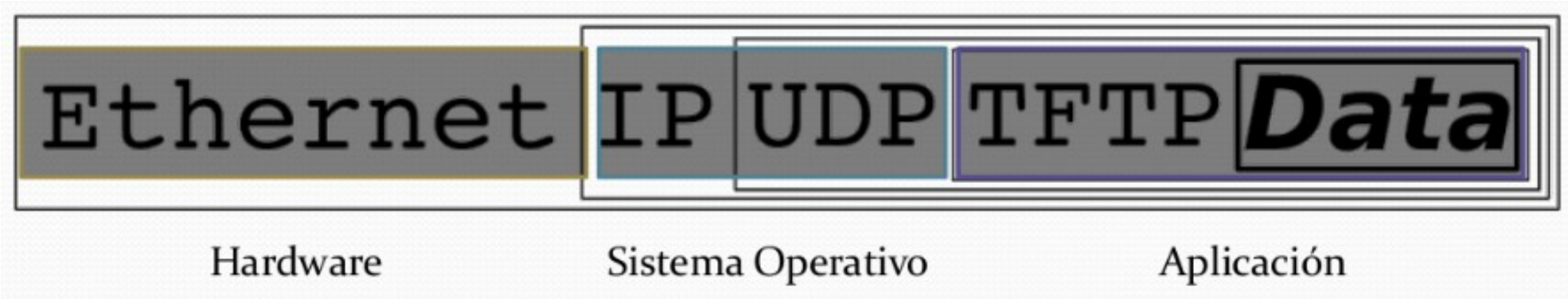


Modelo de capas

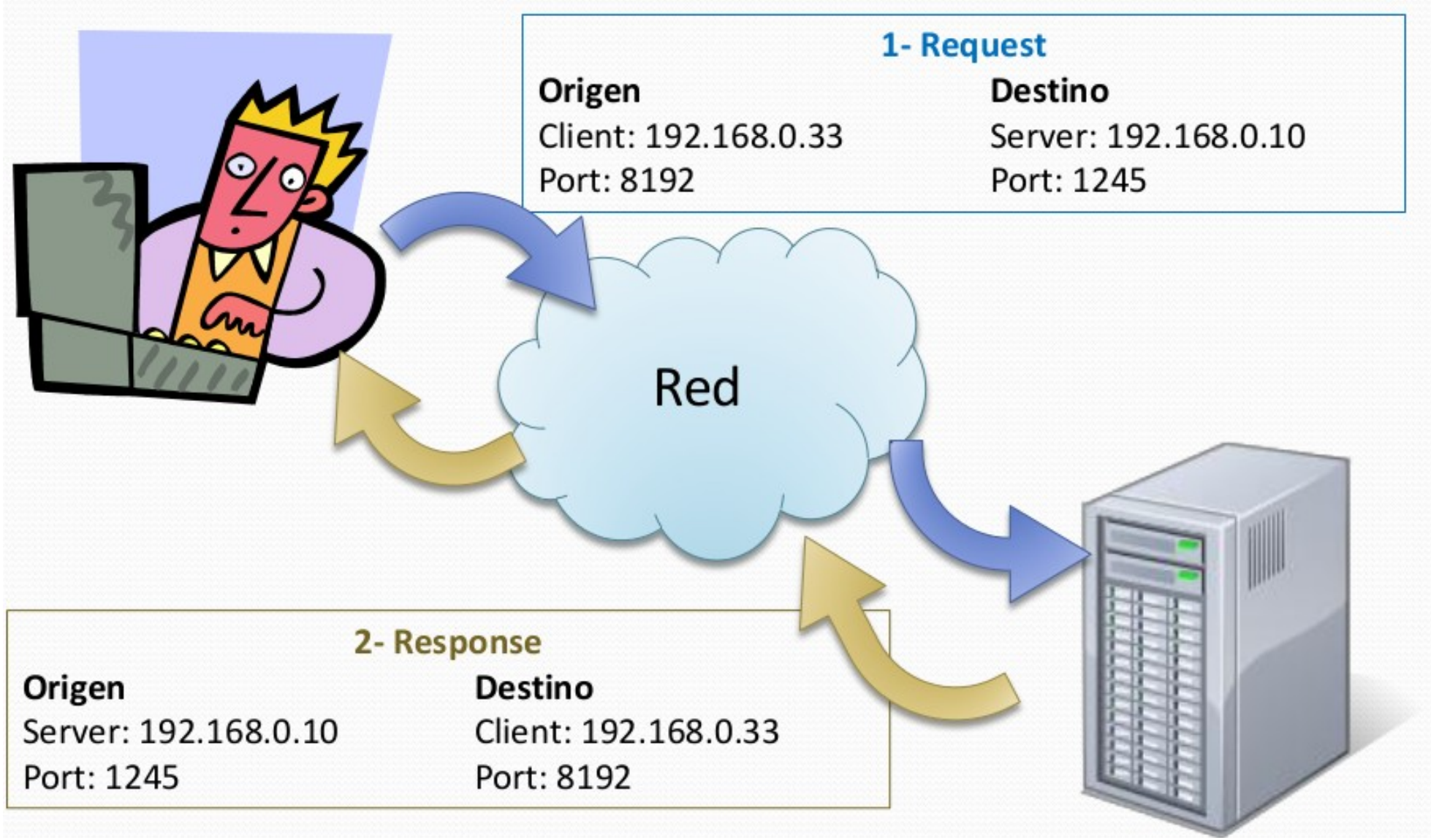


Modelo de capas

- Cada capa del modelo maneja un protocolo
- Cada capa incluye datos de control llamados Header



Modelo cliente - servidor



UDP

- Sobre capa IP
- Comunicación simple
- Se necesita IP/Puerto
- Poco overhead
- No orientada a conexión
  - Los paquetes pueden no llegar
  - Los paquetes pueden llegar desordenados
  - No hay acuse de recibo

TCP

- Sobre capa IP
- Se necesita IP/Puerto
- Comunicación compleja
- Mucho overhead
- Orientada a conexión
  - Retransmisión de paquetes
  - Los paquetes llegan ordenados
  - Hay acuse de recibo

Sockets

- Abstracción que representa una conexión
- Interfaz de programación
- Permite comunicar procesos distribuidos
- Modelo cliente-servidor

Sockets

- **Internet sockets**
  - Manejan direcciones de red
  - Comunicación de sistemas distribuidos
  - Comunicación local
  - Multiplataforma

- **Unix sockets**
  - Comunicación local
  - Solo sistemas unix-like

Cliente UDP

```
import socket

message = "Mensaje desde cliente"

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

sock.sendto(message.encode(), ("localhost",4096))

sock.close()
```

Server UDP

```
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

sock.bind(("localhost",4096))

data, address = sock.recvfrom(1024)

print(data.decode())

sock.close()
```

Bibliografía

- Brian “Beej Jorgensen” Hall. (2015). Beej's Guide to Unix IPC.
- Pablo Ridolfi (2010). Presentación Redes de datos.
- Alejandro Furfaro (2016). Presentación Internetworking.
- http://man7.org/linux/man-pages/man2/mknod.2.html
- http://man7.org/linux/man-pages/man2/socket.2.html