Módulo 02
La Capa de Aplicaciones (Pt. 6)

Redes de Computadoras
Depto. de Cs. e Ing. de la Comp.
Universidad Nacional del Sur

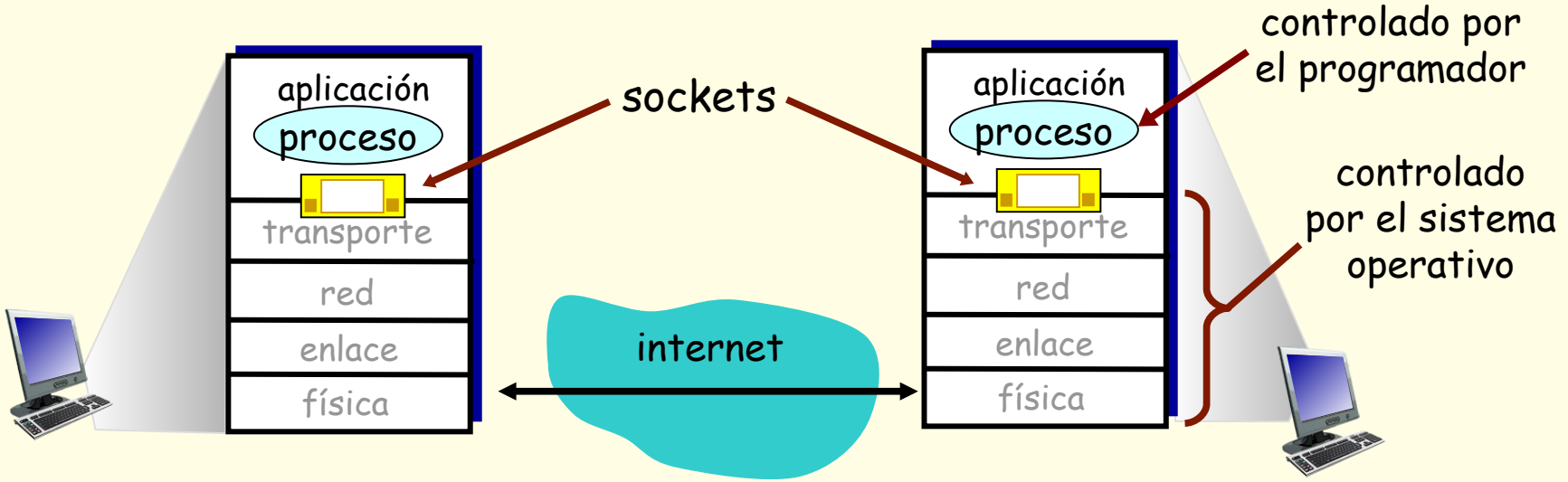
Copyright

Contenidos

- Servicios que requiere la capa de aplicaciones
- Protocolos de la capa de aplicaciones
  - **HTTP**
  - **DNS**
  - **SMTP, POP e IMAP**
- Arquitectura de las aplicaciones **P2P**
- Programación basada en sockets

Programación con sockets

- Un socket es una interfaz local, creada por una cierta aplicación y controlada por el sistema operativo, la cual permite enviar y recibir datos desde/hacia otro proceso



Sockets

- La **API** sockets fue introducida en la versión 4.1 del **UNIX BSD**
  - Los sockets son creados, utilizados y destruidos por las aplicaciones
  - Sigue al estilo arquitectónico cliente-servidor
- Pone a disposición del programador dos servicios de transporte:
  - Transporte no confiable de datagramas
  - Transporte confiable de un flujo de bytes

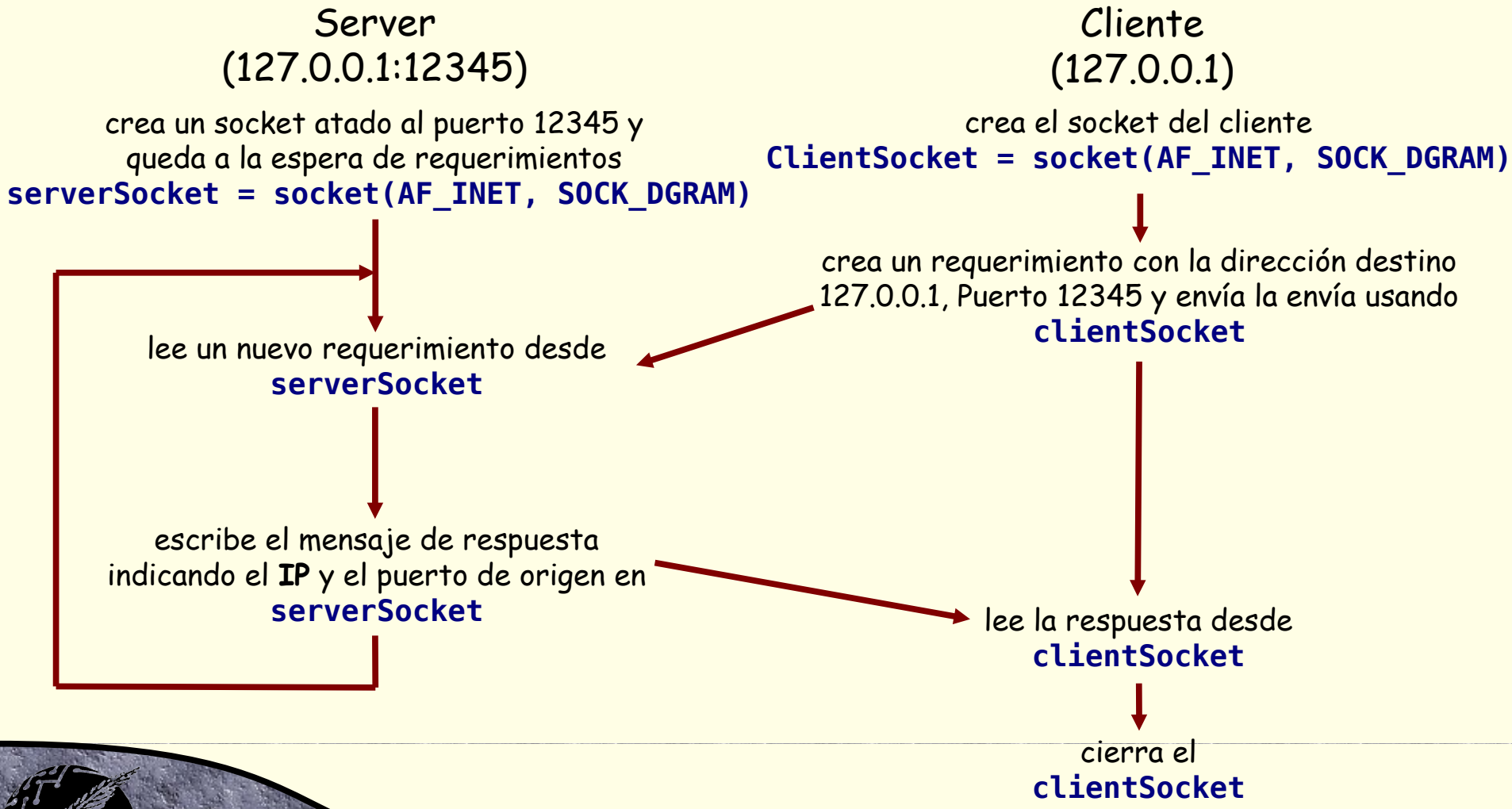
Ejemplo de aplicación

- Se desea implementar una aplicación cliente servidor con las siguientes características:
  - El cliente lee lo ingresado por el usuario en el teclado y lo envía al servidor a través de un socket
  - El servidor lee una línea de su socket y procede a mayusculizarla
  - Luego, el servidor la envía de vuelta al cliente
  - El cliente recibe la respuesta del servidor y finalmente la muestra por pantalla

Programación c/sockets UDP

- El transporte **UDP** no hace uso de una conexión permanente entre cliente y servidor.
  - No requiere inicialización de la conexión
  - Al mandar cada datagrama se debe indicar explícitamente el **IP** y puerto destino
  - El receptor de un datagrama debe tomar nota del **IP** y puerto de origen con el objeto de poder responder al mismo
- Debemos recordar que **UDP** no asegura la transmisión ni el orden de los datagramas

Bosquejo de interacción



Cliente UDP en Python

incluimos la librería Socket de python

crea un socket para datagramas

leemos del teclado

le agregamos nombre y puerto y lo enviamos por el socket

recibimos la respuesta generadas por el servidor

muestra la respuesta por pantalla y cierra el socket

```
from socket import *

serverName = '127.0.0.1'
serverPort = 12345

clientSocket = socket(AF_INET, SOCK_DGRAM)

message = input('Ingrese un string: ')

clientSocket.sendto(bytes(message,'utf-8'),
                    (serverName, serverPort))

modifiedMessage, serverAddress = \
                      clientSocket.recvfrom(2048)

print ('Recibido del servidor:',
       modifiedMessage.decode('utf-8'))

clientSocket.close()
```

Servidor UDP en Python

crea un socket para datagramas

lo ata a la dirección 127.0.0.1, puerto 12345

ciclo infinito

toma del socket el mensaje recibido y registra el IP y puerto de origen

devuelve al cliente la cadena mayusculizada

```
from socket import *

serverSocket = socket(AF_INET, SOCK_DGRAM)

serverSocket.bind(('127.0.0.1', 12345))

print ('El servidor está listo...')

while 1:

    message, clientAddress = \
                    serverSocket.recvfrom(2048)

    modifiedMessage = message.upper()

    serverSocket.sendto(modifiedMessage,
                        clientAddress)
```

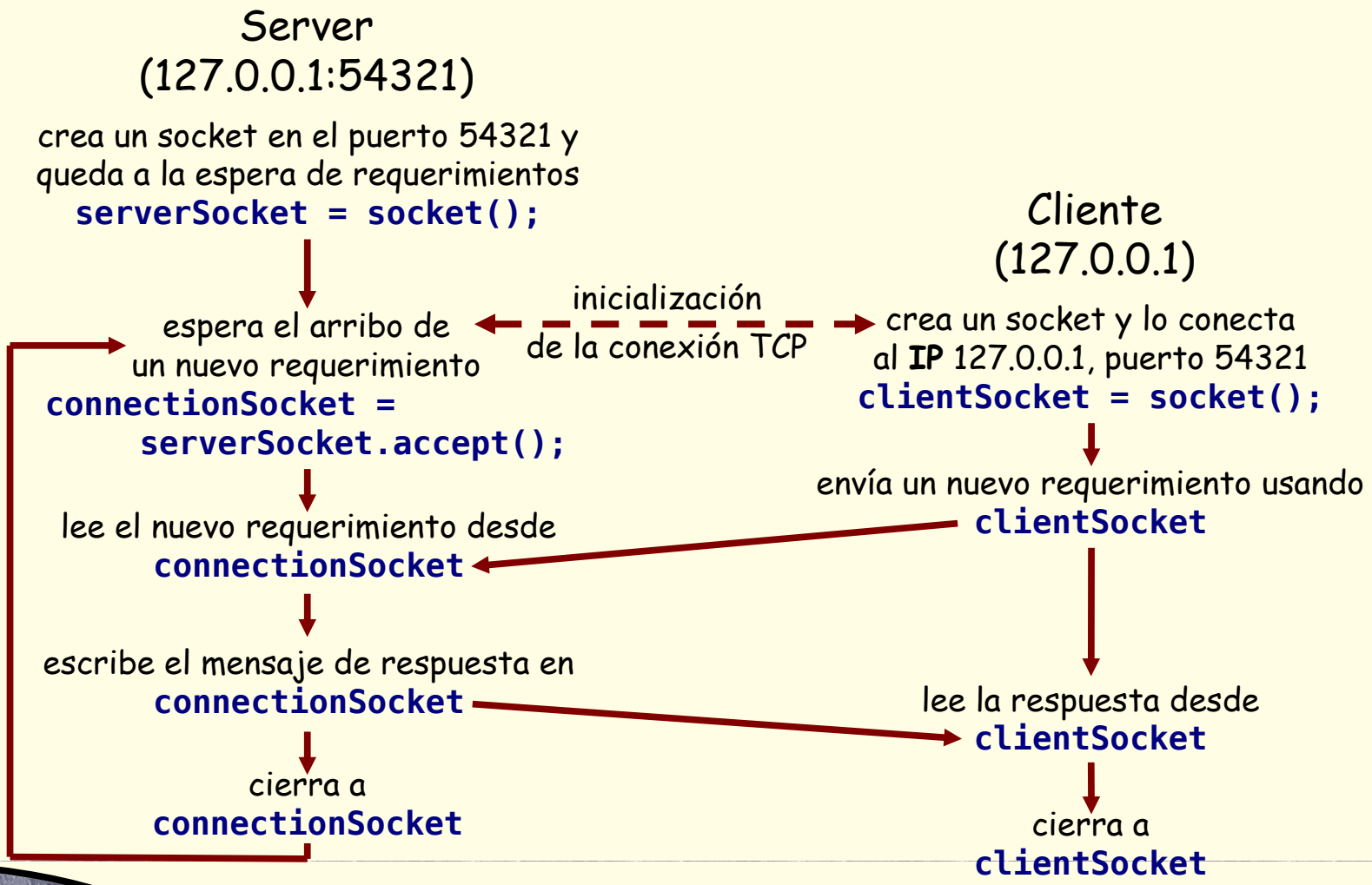
Programación con sockets TCP

- Pasos involucrados en la interacción estándar entre cliente y servidor:
  - El proceso servidor debe estar corriendo al momento de iniciar la comunicación
  - El servidor tiene que haber creado con anterioridad un socket especial que acepte los requerimientos por parte de los clientes
  - El cliente crea un nuevo socket estándar **TCP** especificando la dirección **IP** y el puerto deseado
  - Al crear este socket se establece la conexión **TCP**

Programación con sockets TCP

- Continúa:
  - En el momento de recibir una nueva conexión, el servidor crea un segundo socket a través del cual se comunicará con ese cliente en particular
  - De esta manera, un servidor puede atender múltiples requerimientos de distintos clientes en simultáneo
  - Los clientes se distinguen a través del número de puerto de origen que indican en sus requerimientos

Bosquejo de interacción



Cliente TCP en Python

crea un socket TCP

lo conecta al servidor usando el puerto 54321

no hace falta indicar IP o puerto destino

```
from socket import *

serverName = '127.0.0.1'
serverPort = 54321

clientSocket = socket(AF_INET, SOCK_STREAM)

clientSocket.connect((serverName,serverPort))

sentence = input('Ingrese un string: ')

clientSocket.send(bytes(sentence, 'utf-8'))

modifiedMessage = clientSocket.recv(1024)

print ('Recibido del servidor: ',
       modifiedMessage.decode('utf-8'))

clientSocket.close()
```

Servidor TCP en Python

crea un socket de bienvenida

el servidor empieza a recibir requerimientos

ciclo infinito

el servidor ante el arribo de un requerimiento, crea un nuevo socket

se leen bytes del socket (pero ni IP ni puerto de origen, ¡no hace falta!)

cierra el socket del cliente (no así el de bienvenida)

```
from socket import *

serverSocket = socket(AF_INET,SOCK_STREAM)

serverSocket.bind(('127.0.0.1',54321))

serverSocket.listen(1)

print ('El servidor está listo...')

while 1:

   connectionSocket, addr = serverSocket.accept()

   sentence = connectionSocket.recv(1024)
   capitalizedSentence = sentence.upper()
   connectionSocket.send(capitalizedSentence)
   connectionSocket.close()
```

¿Preguntas?