Computer Network and Distributed Systems

Socket Programming

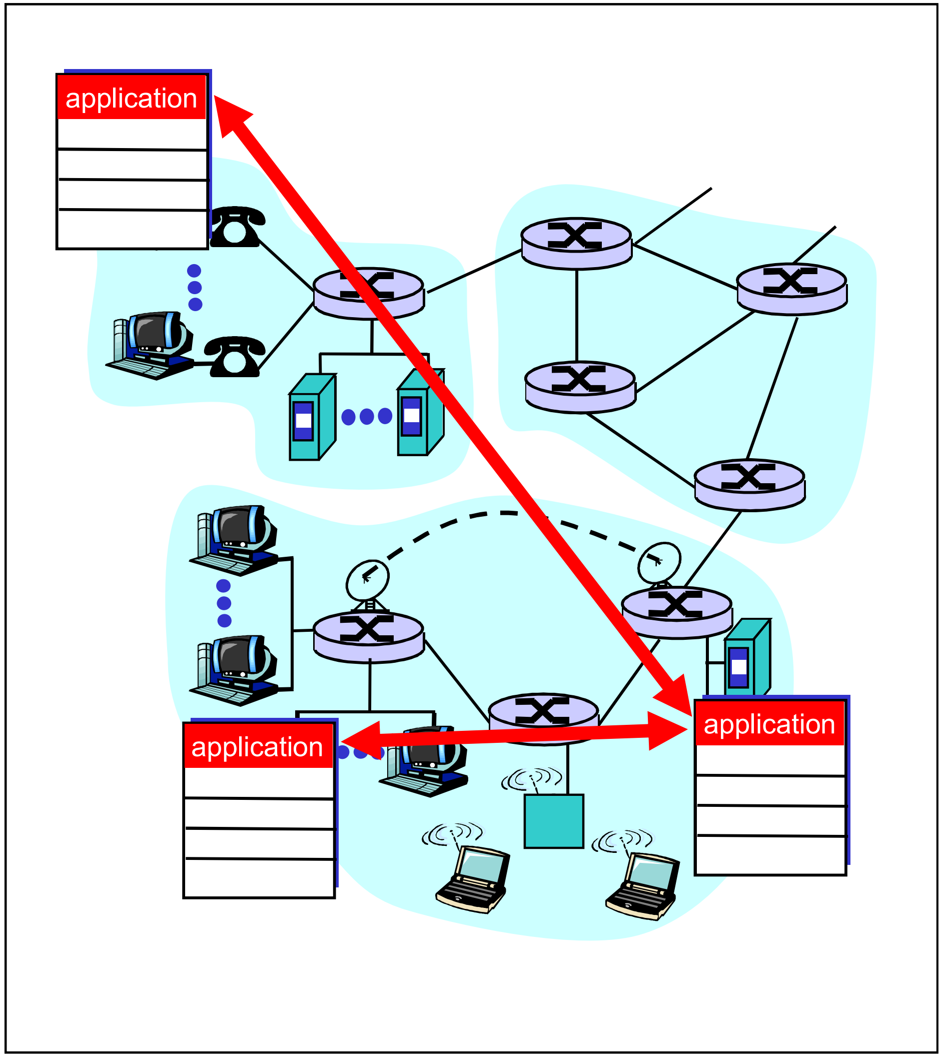
Applications and Application-Layer Protocols

Application: communicating, distributed processes

- Running in network hosts in "user space"
- Exchange messages to implement app
- e.g., email, file transfer, the Web

Application-layer protocols

- One "piece" of an app
- Define messages exchanged by apps and actions taken
- User services provided by lower layer protocols



Client-Server Paradigm

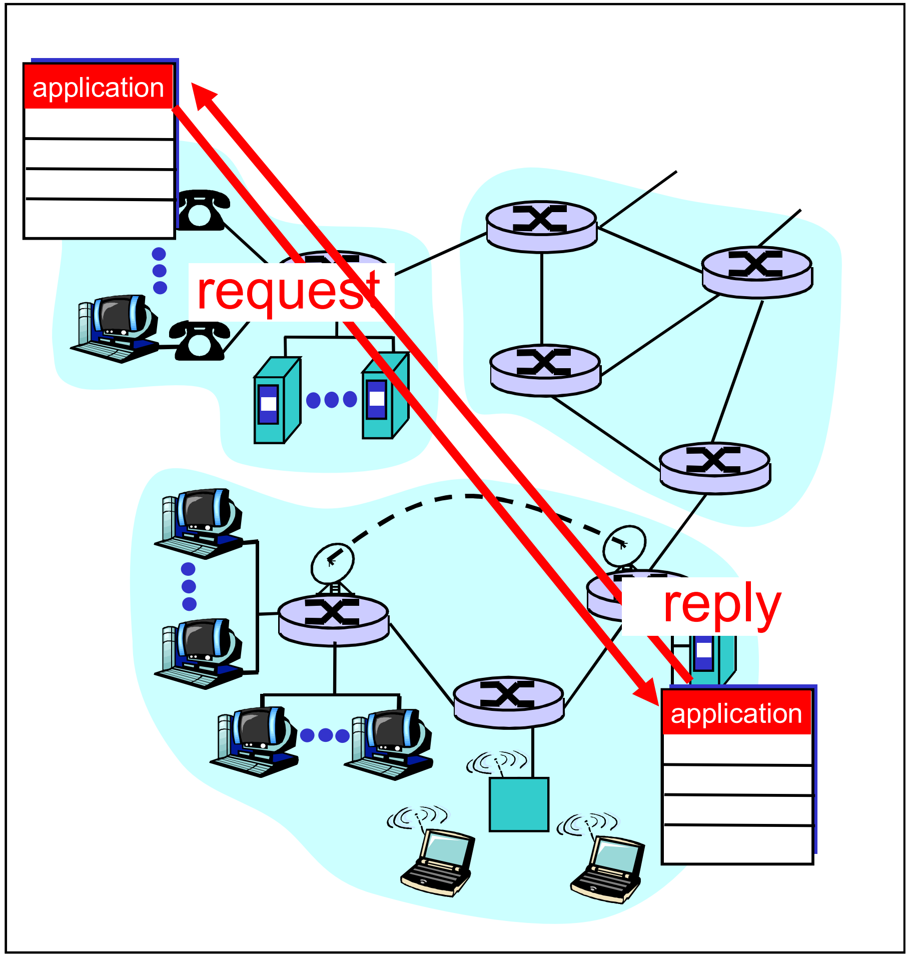
Typical network app has two pieces: *client* and *server*

Client:

- Initiates contact with server ("speaks first")
- Typically requests service from server,
- For Web, client is implemented in browser; for e-mail, in mail reader

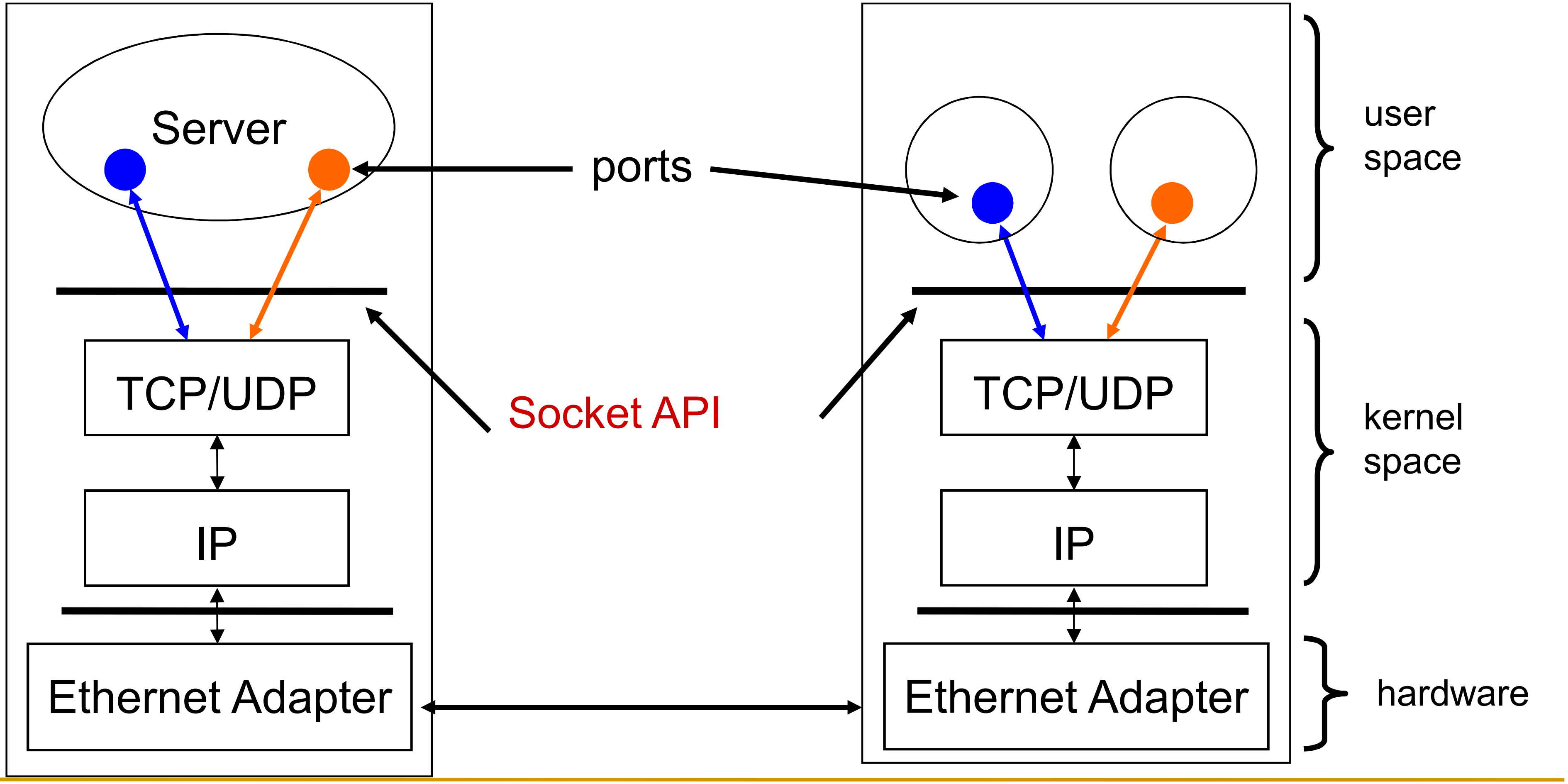
Server:

- Provides requested service to client
- e.g., Web server sends requested Web page, mail server delivers e-mail



Server and Client

Server and Client exchange messages over the network through a common Socket API



Transmission Control Protocol (TCP): An Analogy

TCP

Reliable – guarantee delivery

Byte stream – in-order delivery

Connection-oriented – single socket per connection

Setup connection followed by data transfer

Telephone Call

- *Guaranteed delivery*
- *In-order delivery*
- *Connection-oriented*
- *Setup connection followed by conversation*

Example TCP applications

Web, Email, Telnet

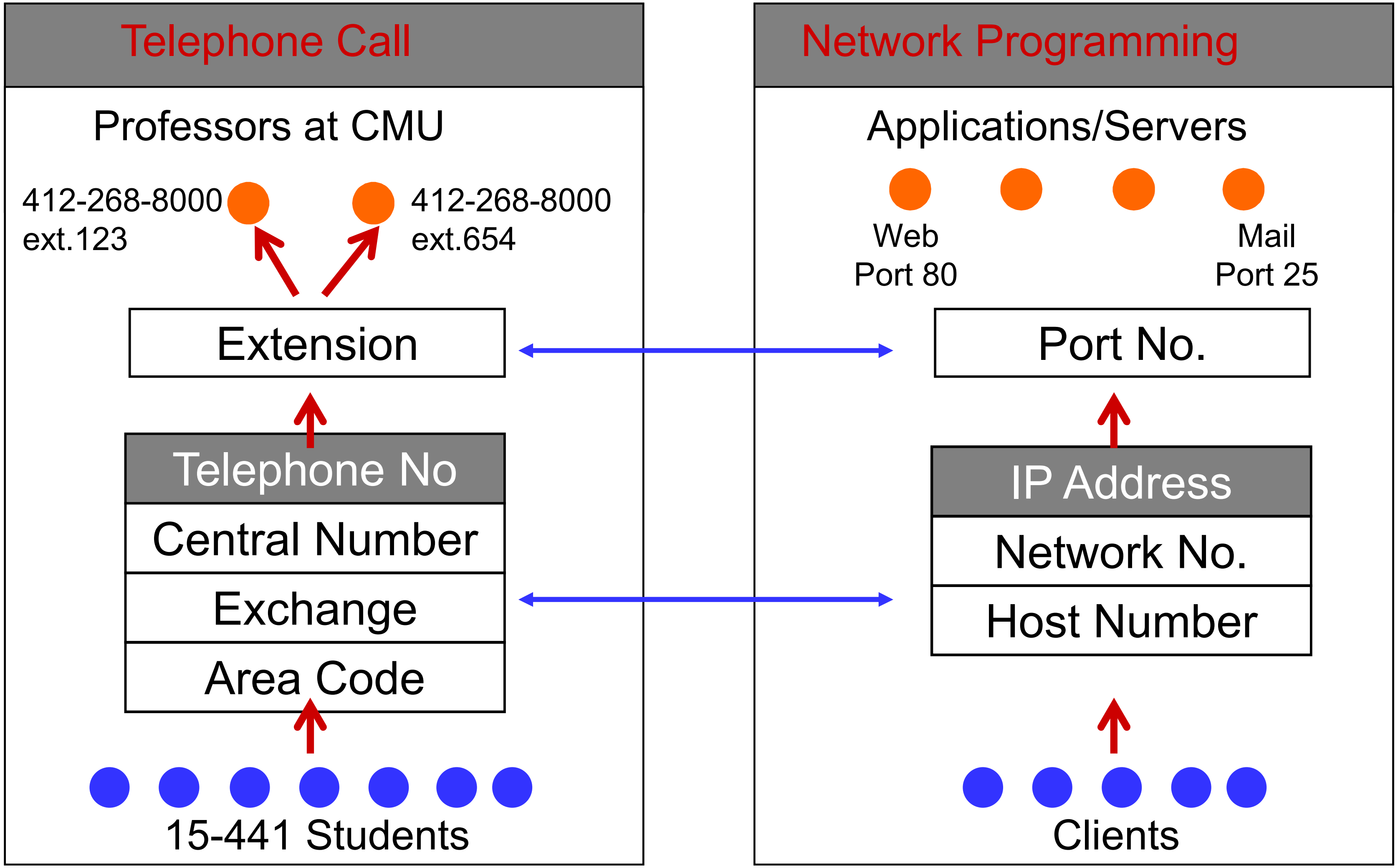
User Datagram Protocol(UDP): An Analogy

UDP	Postal Mail
Single socket to receive messages	*Single mailbox to receive letters*
No guarantee of delivery	*Unreliable ☺*
Not necessarily in-order delivery	*Not necessarily in-order delivery*
Datagram – independent packets	*Letters sent independently*
Must address each packet	*Must address each reply*

Example UDP applications

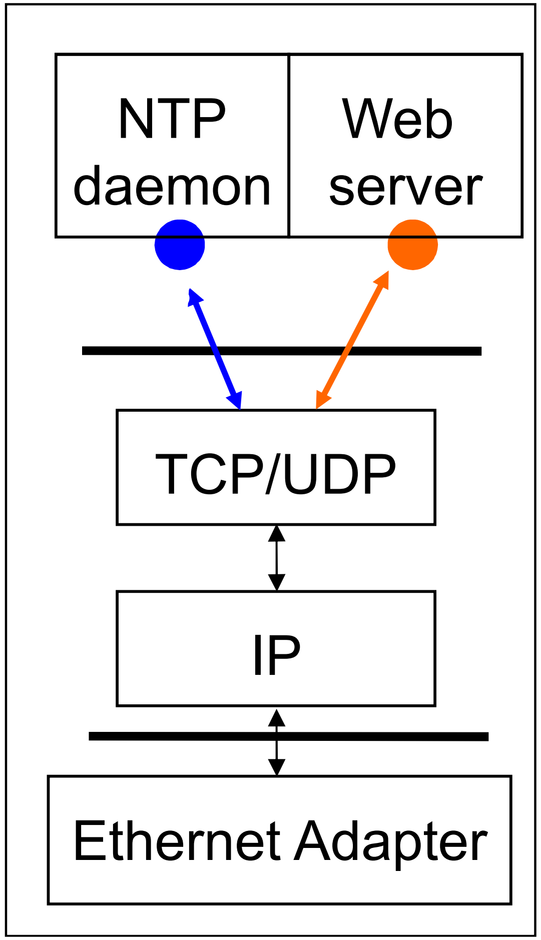
Multimedia, voice over IP

Network Addressing Analogy



Concept of Port Numbers

- Port numbers are used to identify "entities" on a host
- Port numbers can be
  - Well-known (port 0-1023)
  - Dynamic or private (port 1024-65535)
- Servers/daemons usually use well-known ports
  - Any client can identify the server/service
  - HTTP = 80, FTP = 21, Telnet = 23, ...
  - /etc/service defines well-known ports
- Clients usually use dynamic ports
  - Assigned by the kernel at run time



Socket System Calls

socket()—Get the File Descriptor!

- ❑ AF_INET: associates a socket with the Internet protocol family
- ❑ SOCK_STREAM: selects the TCP protocol
- ❑ SOCK_DGRAM: selects the UDP protocol

bind()—What port (*and address*) am I on?

connect()—Hey, you!

listen()—Will somebody please call me?

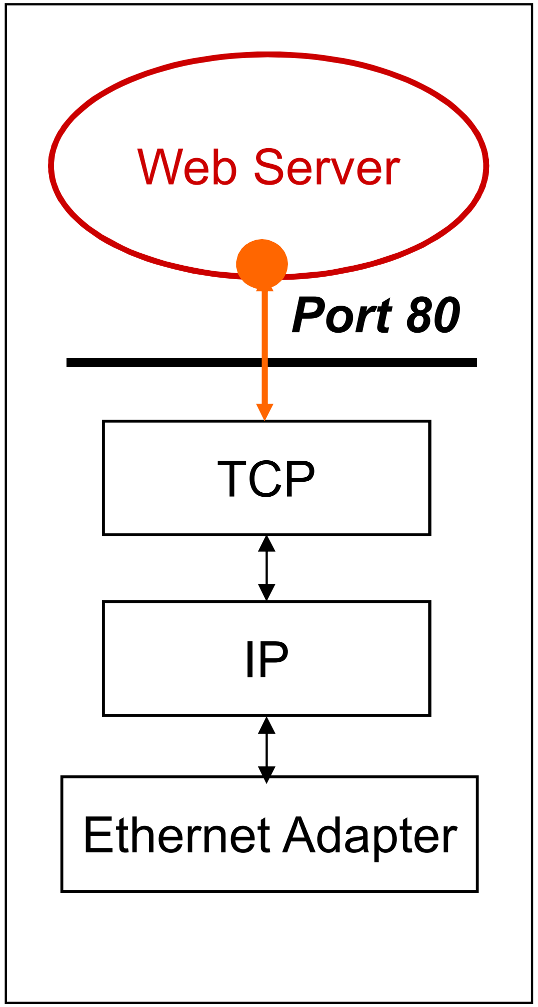
accept()—Thank you for calling port 3490.

send() and recv()—Talk to me, buddy!

sendto() and recvfrom()—Talk to me, DGRAM-style

close() and shutdown()—Get outta my face!

TCP Server/client example



Web Server

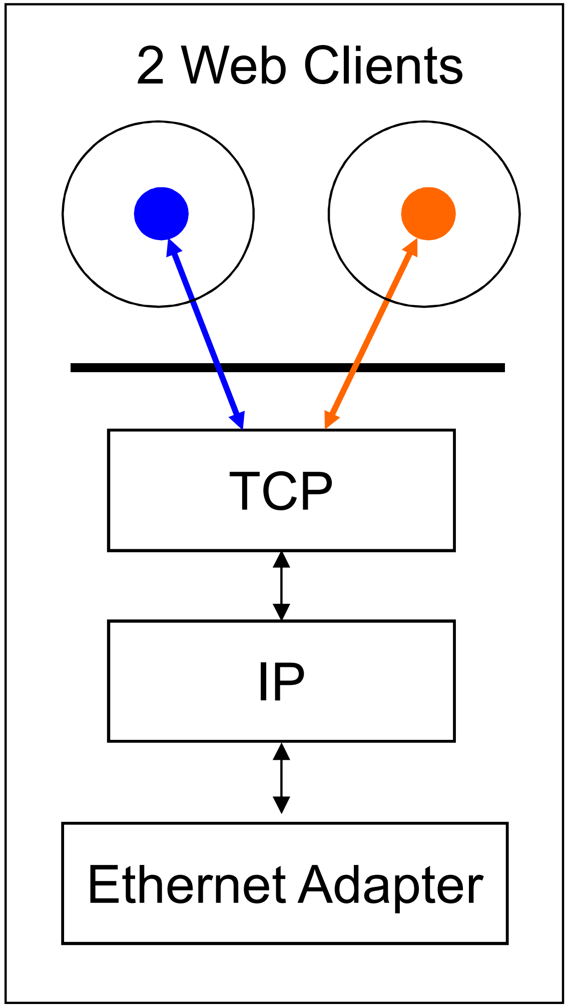
Port 80

TCP

IP

Ethernet Adapter

What does a *web server* need to do so that a *web client* can connect to it?



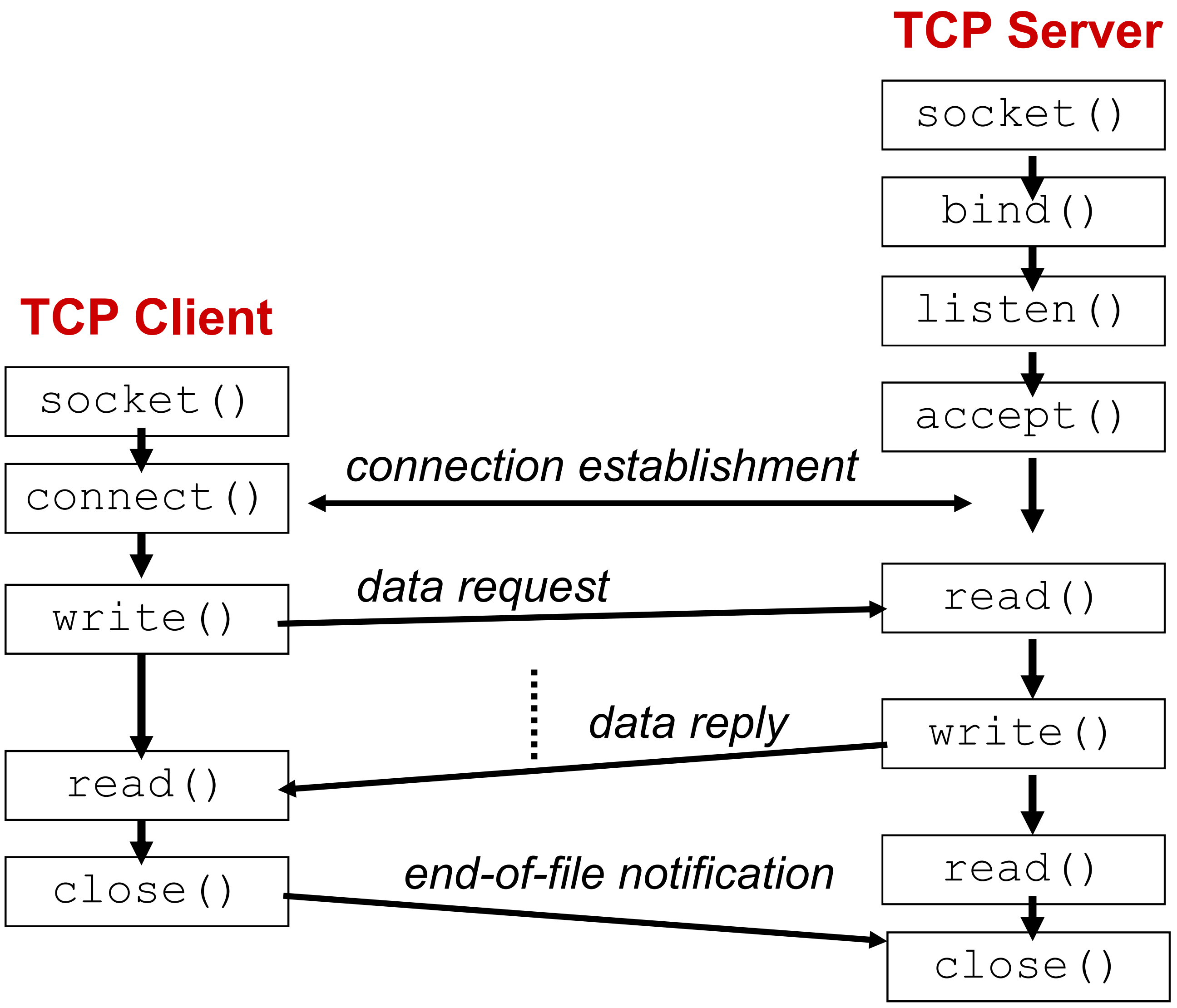
2 Web Clients

TCP

IP

Ethernet Adapter

TCP Client-Server Interaction



TCP Server

socket()

bind()

listen()

accept()

TCP Client

socket()

connect()

connection establishment

write()

data request

read()

data reply

write()

read()

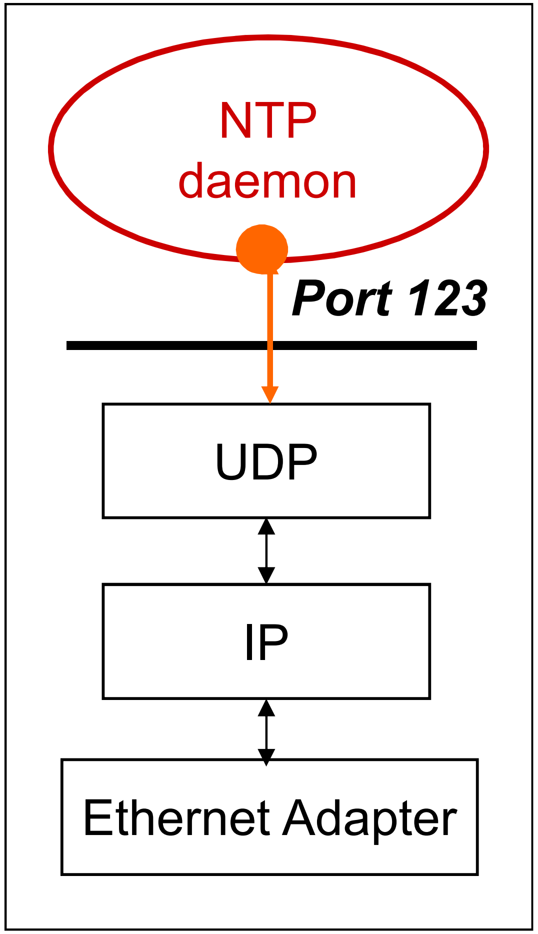
read()

close()

end-of-file notification

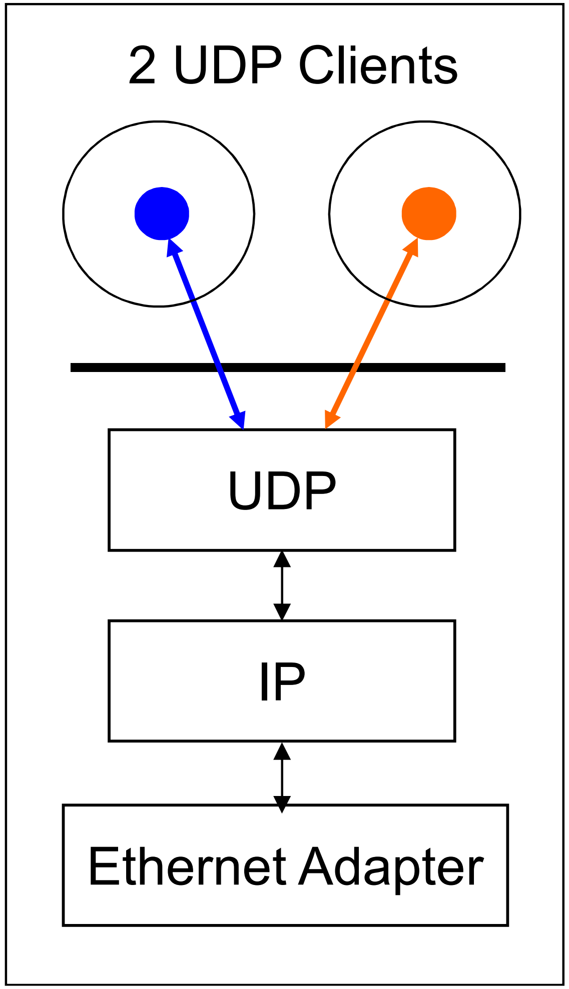
close()

UDP Server/client Example

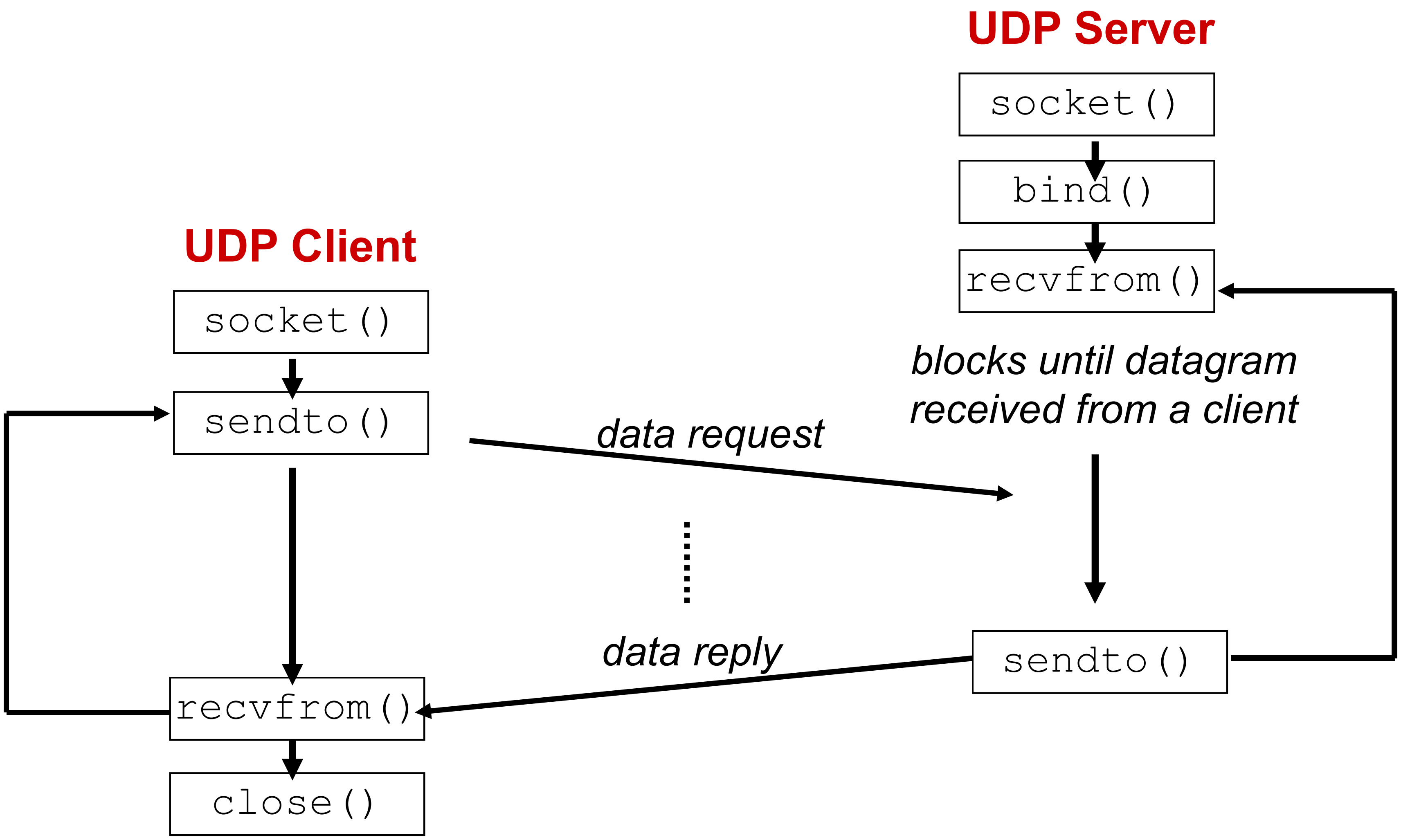


For example: NTP daemon

What does a *UDP server* need to do so that a *UDP client* can connect to it?



UDP Client-Server Interaction



Byte Ordering

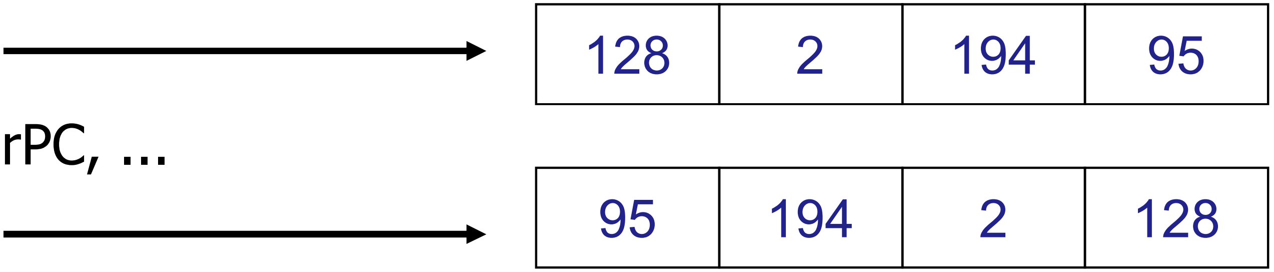
```
union {
    u_int32_t addr;   /* 4 bytes address */
    char c[4];
} un;
/* 128.2.194.95 */
un.addr = 0x8002c25f;
/* c[0] = ? */
```

Big Endian

  Sun Solaris, PowerPC, ...

Little Endian

  i386, alpha, ...



128	2	194	95
95	194	2	128

Network byte order = Big Endian

Byte Ordering Functions

Converts between **host byte order** and **network byte order**

‘h’ = host byte order

‘n’ = network byte order

‘l’ = long (4 bytes), converts IP addresses

‘s’ = short (2 bytes), converts port numbers

```
#include <netinet/in.h>

unsigned long int htonl(unsigned long int hostlong);
unsigned short int htons(unsigned short int
hostshort);
unsigned long int ntohl(unsigned long int netlong);
unsigned short int ntohs(unsigned short int
netshort);
```