



Socket Programming

Peerapon S.

CPE 314: Computer Networks (2/61)

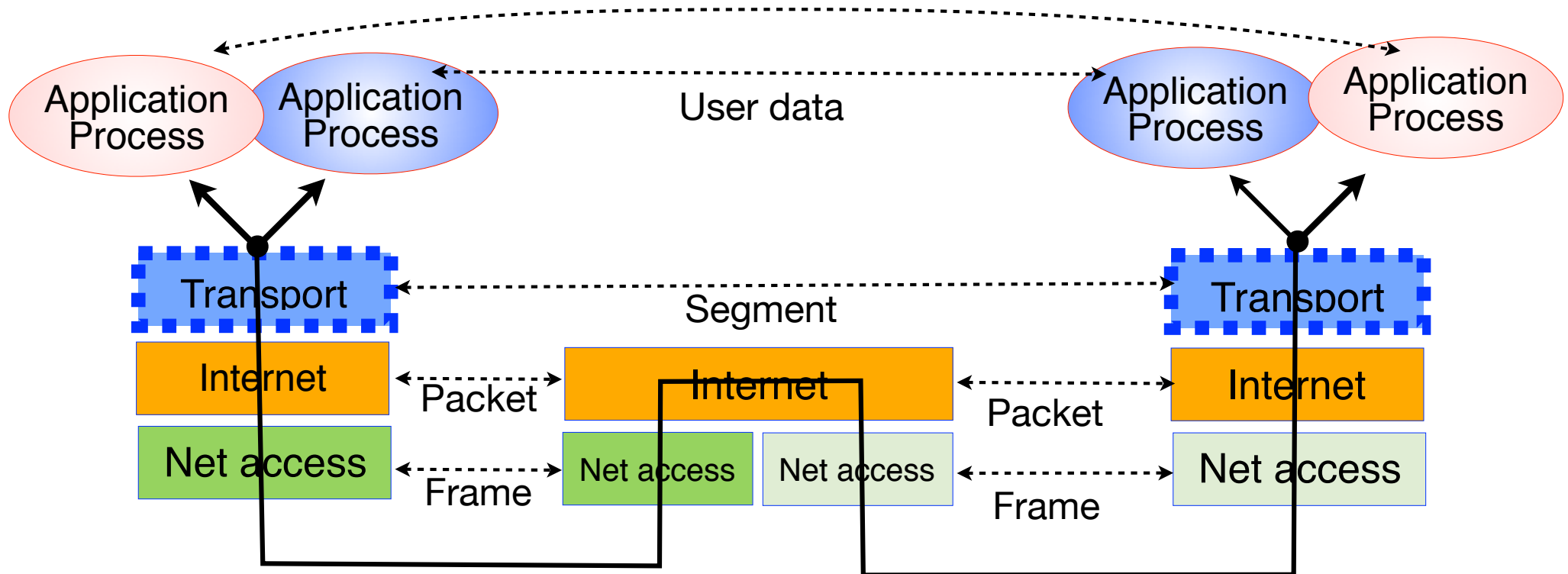
Topics

- ☐ Applications and transport layer services
- ☐ UDP socket
- ☐ TCP socket
- ☐ Concurrent communication

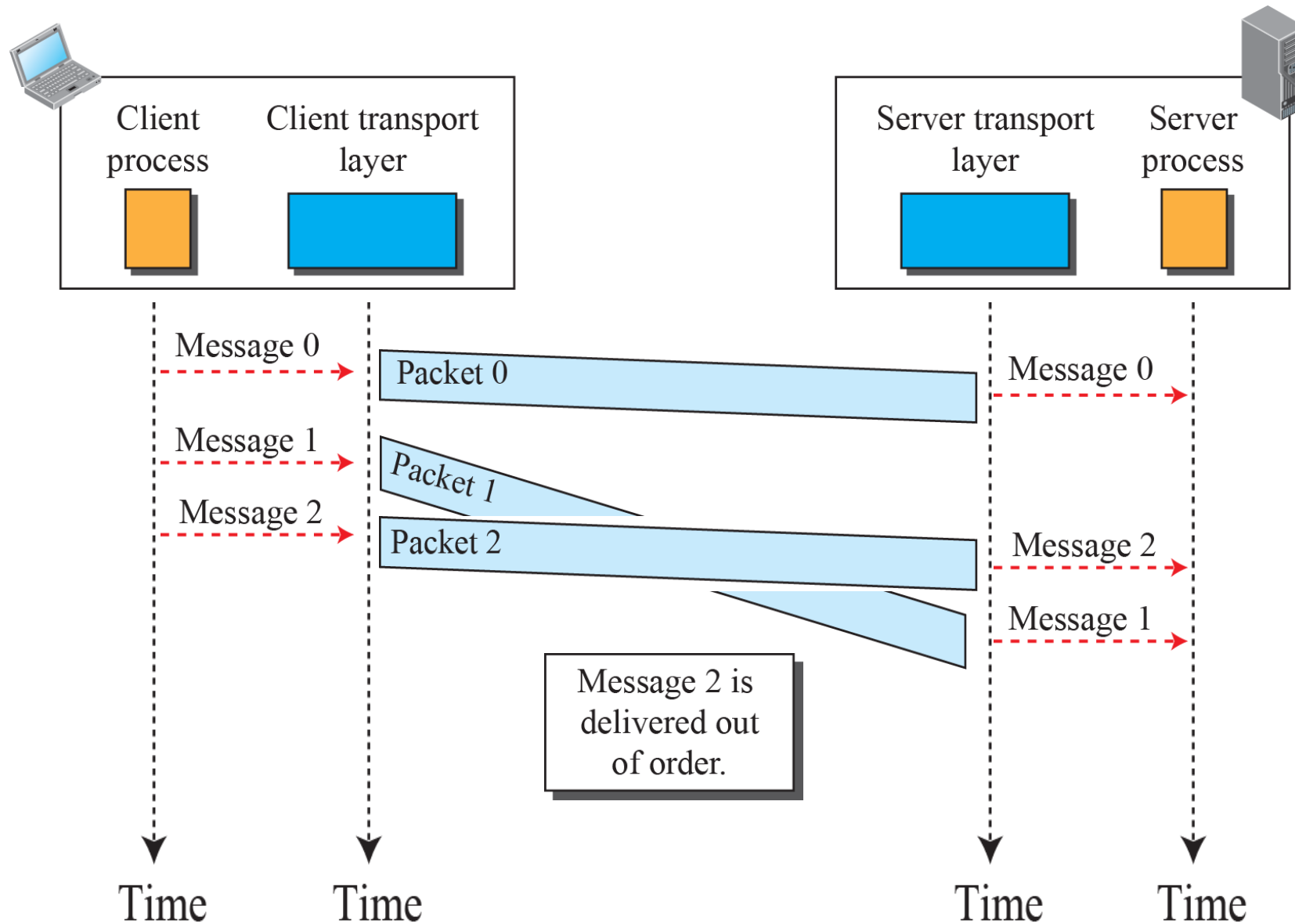
- ☐ Readings
 - Forouzan text: Ch. 2.1, 2.2, 2.5
 - Kurose text: Ch.2.7

Transport Layer (TL) Services

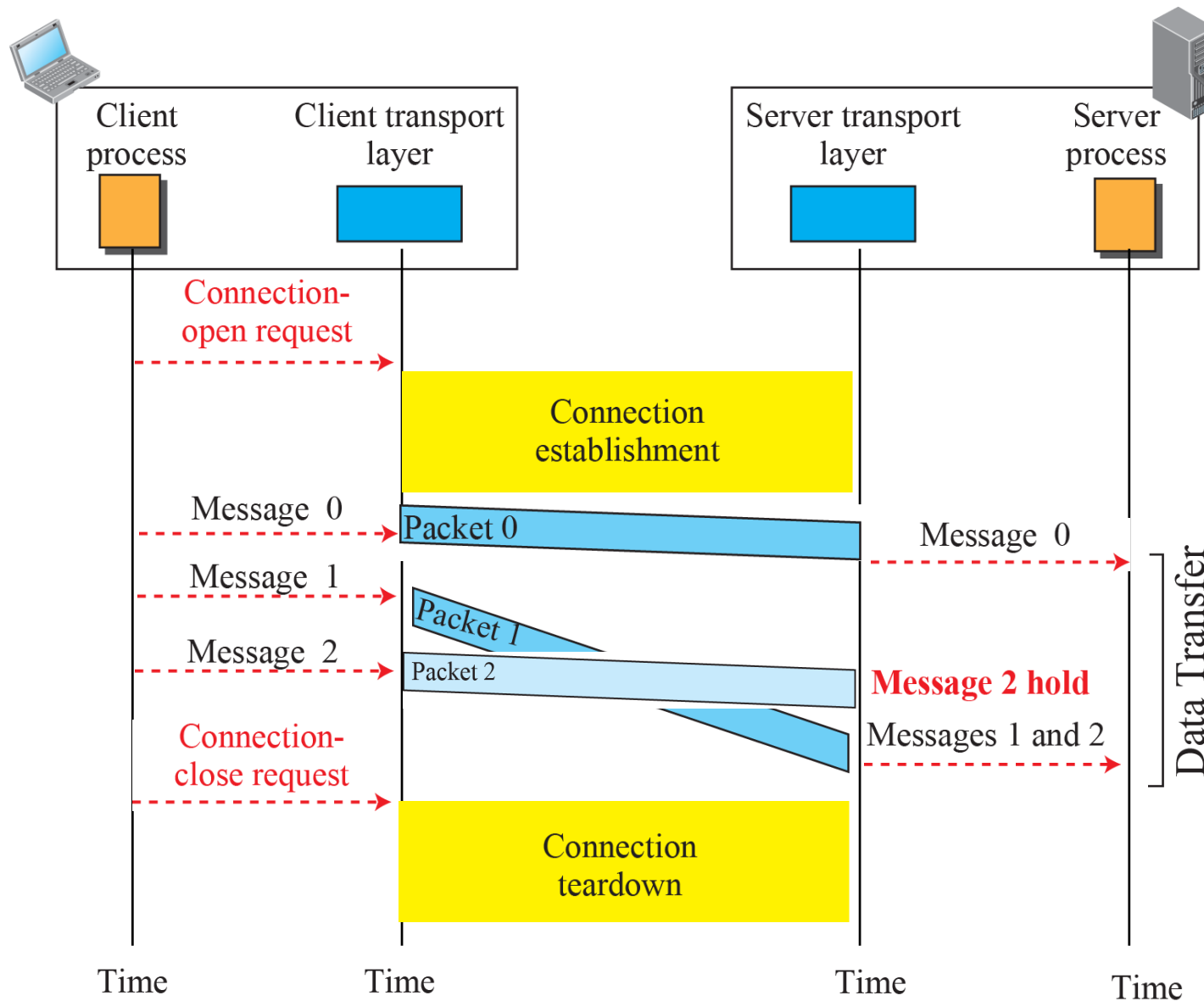
- Process-to-process message delivery service



Connectionless Transport Service



Connection-oriented Transport Service



Two Main Internet Transport Layer Protocols

■ UDP: User Datagram Protocol

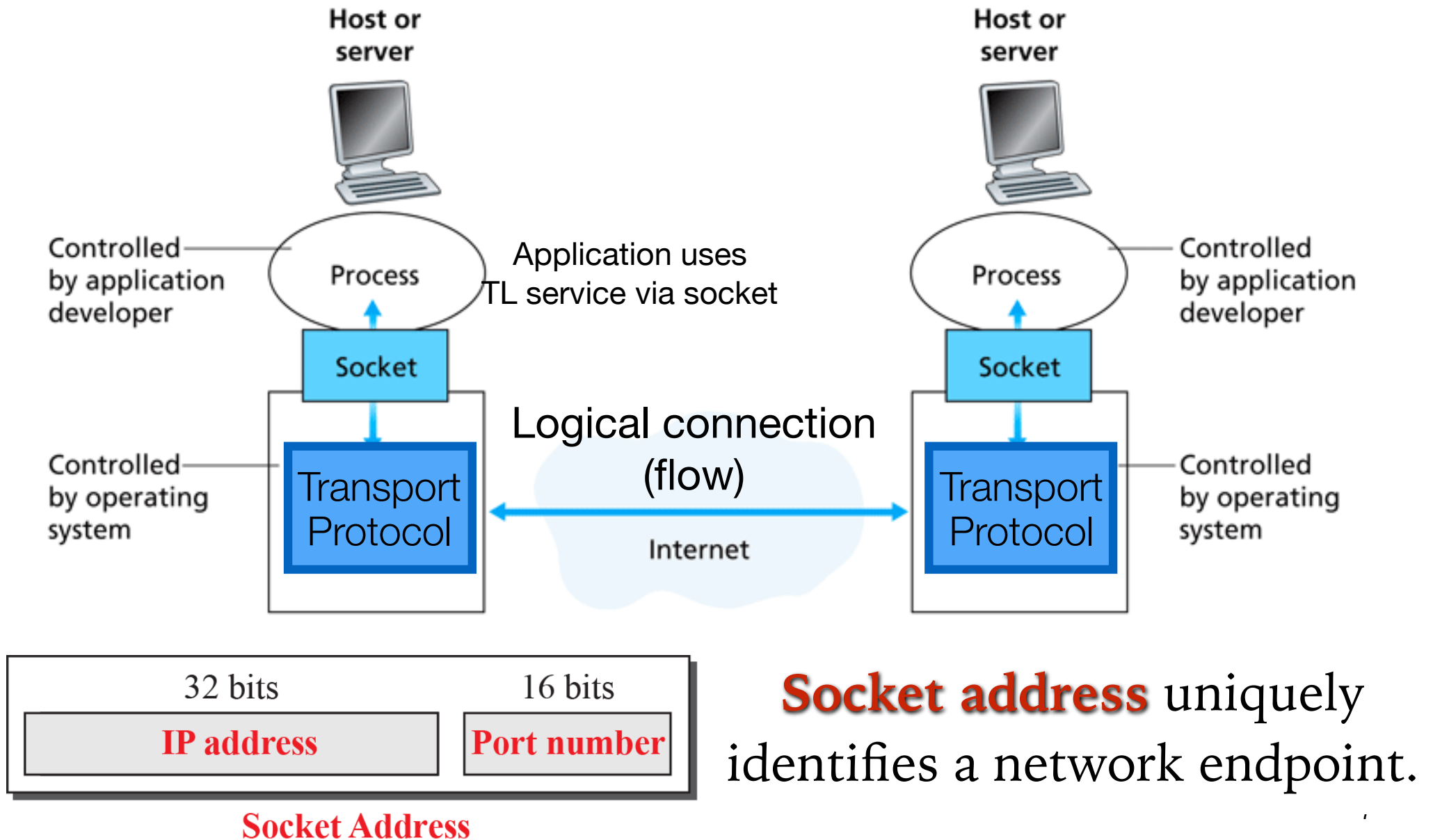
- Connection-less, Unreliable service
- Independent messages delivery

■ TCP: Transport Control Protocol

- Connection-oriented, Reliable service
- Byte-stream delivery

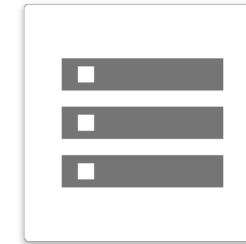
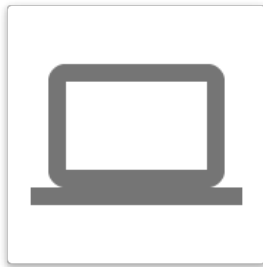
	Loss-free	Order	No duplicate	Bound delay	Throughput
UDP					
TCP	●	●	●		

Socket Abstraction and Socket Address



web client application
(10.35.21.10, 56780)

Web server application
(www.kmutt.ac.th, 80)



To view active connections

Windows: netstat -o -n

MacOS: lsof -i | grep -E "(LISTEN|ESTABLISHED)"

```
$ netstat -a
Active Internet connections (including servers)
Proto Recv-Q Send-Q Local Address      Foreign Address    (state)
tcp4      0      0 10.201.192.137.54990 17.252.156.64.5223 ESTABLISHED
tcp4      0      0 10.201.192.137.54989 mx-ll-110.164.11.http ESTABLISHED
tcp4      0      0 10.201.192.137.54987 sin10s06-in-f69..https ESTABLISHED
tcp4      0      0 10.201.192.137.54986 sin10s06-in-f69..https ESTABLISHED
tcp4      0      0 10.201.192.137.54985 sin10s06-in-f69..https ESTABLISHED
tcp4      0      0 10.201.192.137.54984 sin10s06-in-f69..https ESTABLISHED
tcp4      0      0 10.201.192.137.54976 server-52-84-229.https ESTABLISHED
tcp4     31      0 10.201.192.137.54975 162.125.81.4.https CLOSE_WAIT
tcp4      0      0 10.201.192.137.54974 server-52-84-229.https ESTABLISHED
tcp6      0      0 *.54943            *.*                LISTEN
tcp4      0      0 *.54943            *.*                LISTEN
tcp4      0      0 10.201.192.137.54928 sin10s06-in-f77..https CLOSE_WAIT
tcp4      0      0 10.201.192.137.54919 17.188.166.13.5223 ESTABLISHED
tcp4      0      0 10.201.192.137.54916 ec2-52-45-60-77..https CLOSE_WAIT
tcp4      0      0 10.201.192.137.54905 sin11s02-in-f10..https CLOSE_WAIT
tcp4      0      0 10.201.192.137.54885 162.125.34.129.https ESTABLISHED
tcp4     31      0 192.168.1.54.54702 d.v.dropbox.com.https CLOSE_WAIT
tcp6      0      0 fe80::b146:1efd::black fe80::c125:7371::50015 ESTABLISHED
tcp6      0      0 fe80::b146:1efd::1024 fe80::c125:7371::1024 ESTABLISHED
tcp4      0      0 192.168.1.54.53120 sb-in-f125.1e100.jabbe ESTABLISHED
tcp4      0      0 192.168.1.54.53117 sb-in-f125.1e100.jabbe ESTABLISHED
```


Port Range for Server and Client

- Only some port numbers can be used by your custom app.

0 - 1023	1024 - 49,151	49,152 - 65,535
----------	---------------	-----------------

Well-known
service ports

Registered ports

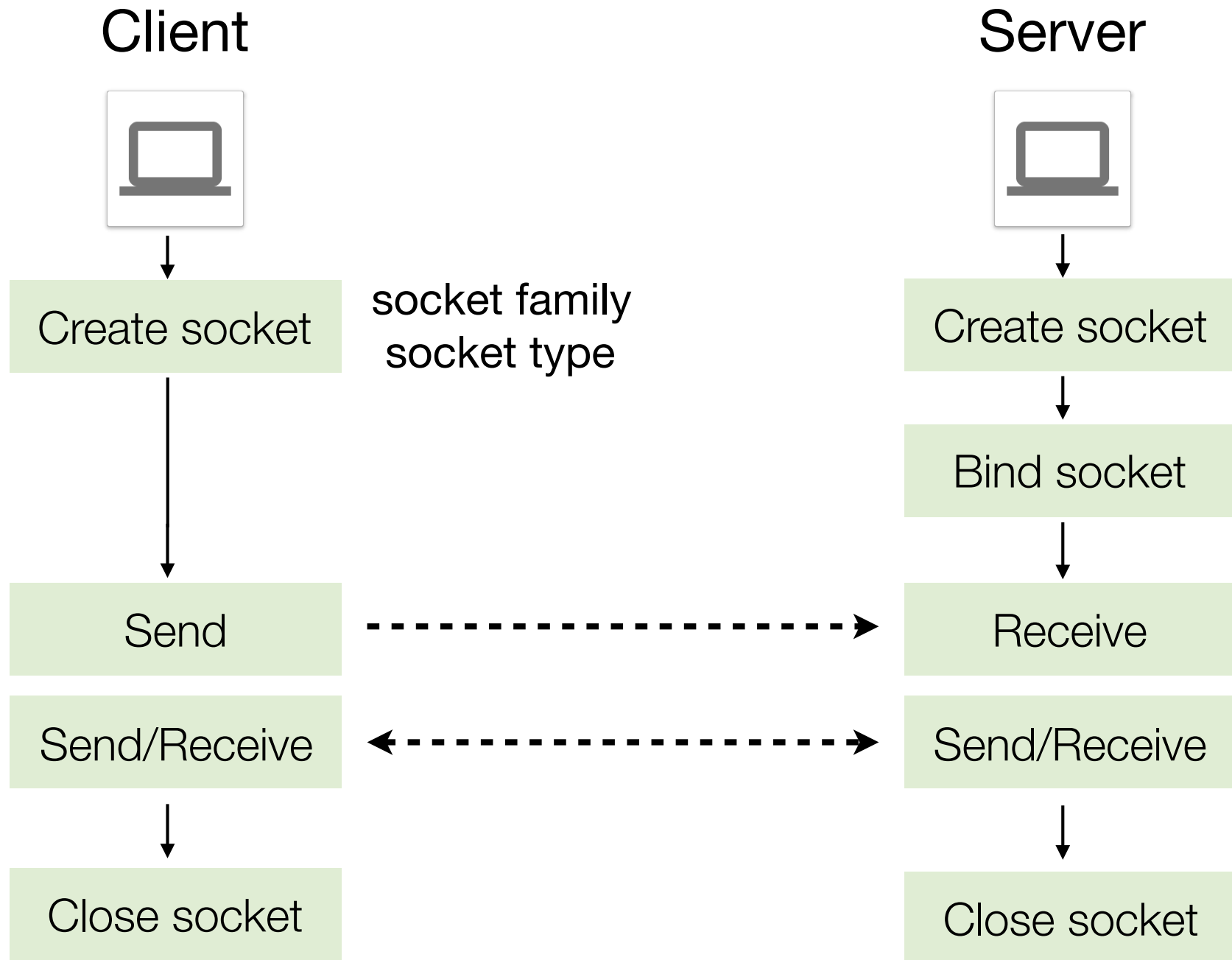
Ephemeral ports
(dynamic, private, temporary)

	Local port	Remote port
Client side	Random from Ephemeral range	Service or Registered port
Server side	Service or Registered port	Derived from client segment

Network Programming in Python

- ❑ Easier than C/C++, Java (IMO)
- ❑ Lots of standard libraries
- ❑ Two basic socket types
 - UDP socket
 - TCP socket

Workflow for UDP socket communication



UDP Server

```
from socket import *
import sys

MAX_BUF = 2048      # Size of buffer to store received bytes
SERV_PORT = 50000   # Server port number

addr = ('127.0.0.1', SERV_PORT)          # Socket address
s = socket(AF_INET, SOCK_DGRAM)          # Create UDP socket
s.bind(addr)                             # Bind socket to address
print ('UDP server started ...')
```

```

while(1):
    print ('Client> ', end = '')
    txtin,addr = s.recvfrom(MAX_BUF)  # txtin = receive text
                                       # addr = client socket address
    print ('%s' %(txtin).decode('utf-8')) # Convert byte to string
    if txtin == b'quit':                # Break if user types 'quit'.
        print('Terminate server ...')
        break
    else:
        txtout = txtin.upper()          # Change text to upper case
        s.sendto(txtout, addr)          # Send it back to the client

```

Some key Python syntax rules:

- Indentation for code blocks
- ':' at the end of control statement.
- No variable declaration needed.

UDP Client

```
from socket import *  
import sys
```

```
MAX_BUF = 2048
```

```
SERV_PORT = 50000
```

Change to something else
if not running in the same computer

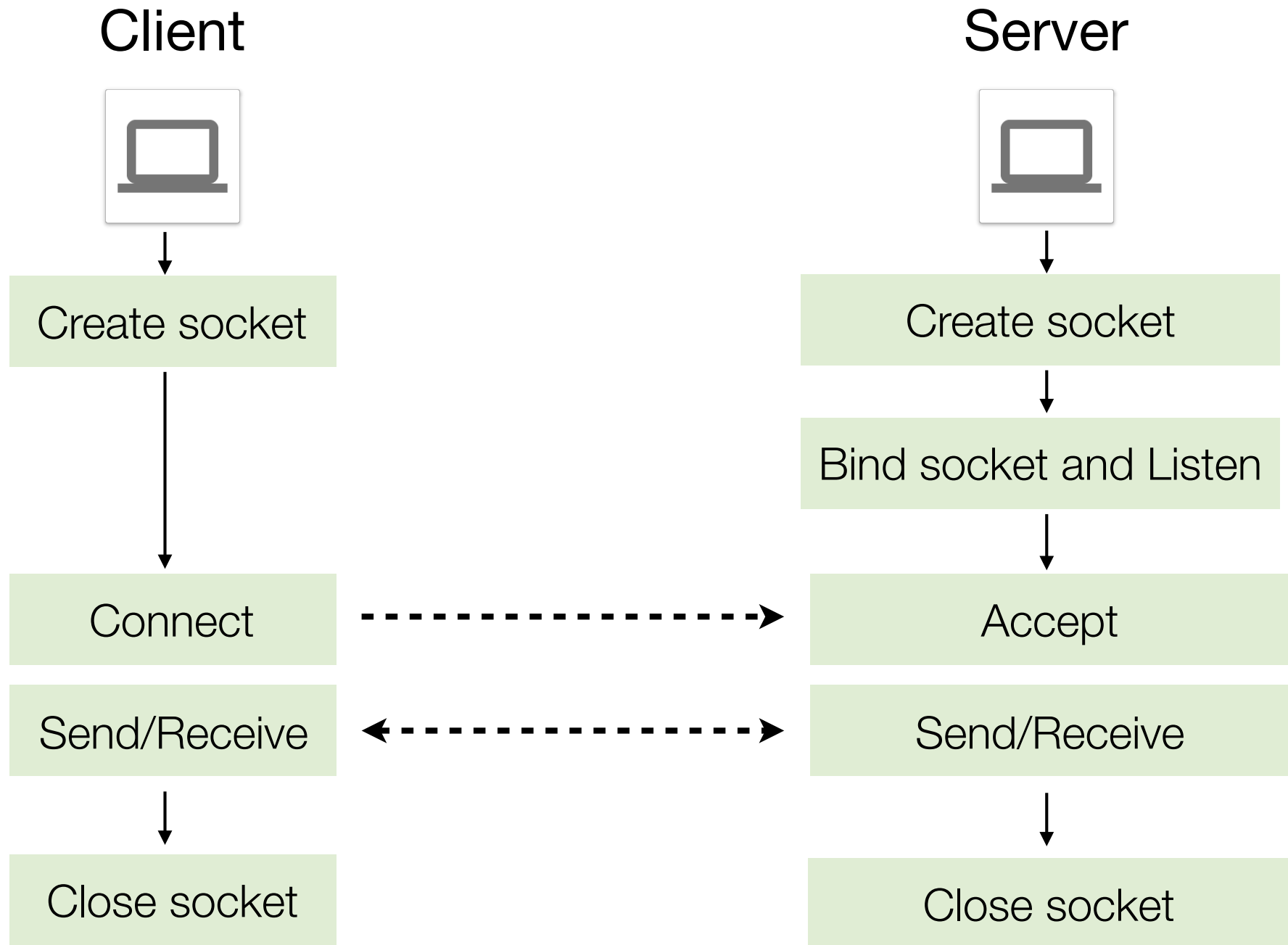


```
addr = ('127.0.0.1', SERV_PORT) # Server socket address
```

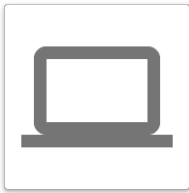
```
s = socket(AF_INET, SOCK_DGRAM) # Create UDP socket
```

```
username = input('Enter your name: ') # text for prompt
while(1):
    print('%s> ' %(username), end='') # Print the prompt
    sys.stdout.flush()
    txtout = sys.stdin.readline().strip() # Take input from user keyboard
    s.sendto(txtout.encode('utf-8'), addr) # Convert to string to byte and send
    if txtout == 'quit':                  # Exit if user types quit
        break
    modifiedMsg, srvAddr = s.recvfrom(2048) # Wait for modified text from server
    print (modifiedMsg.decode('utf-8'))     # Print the modified text.
```

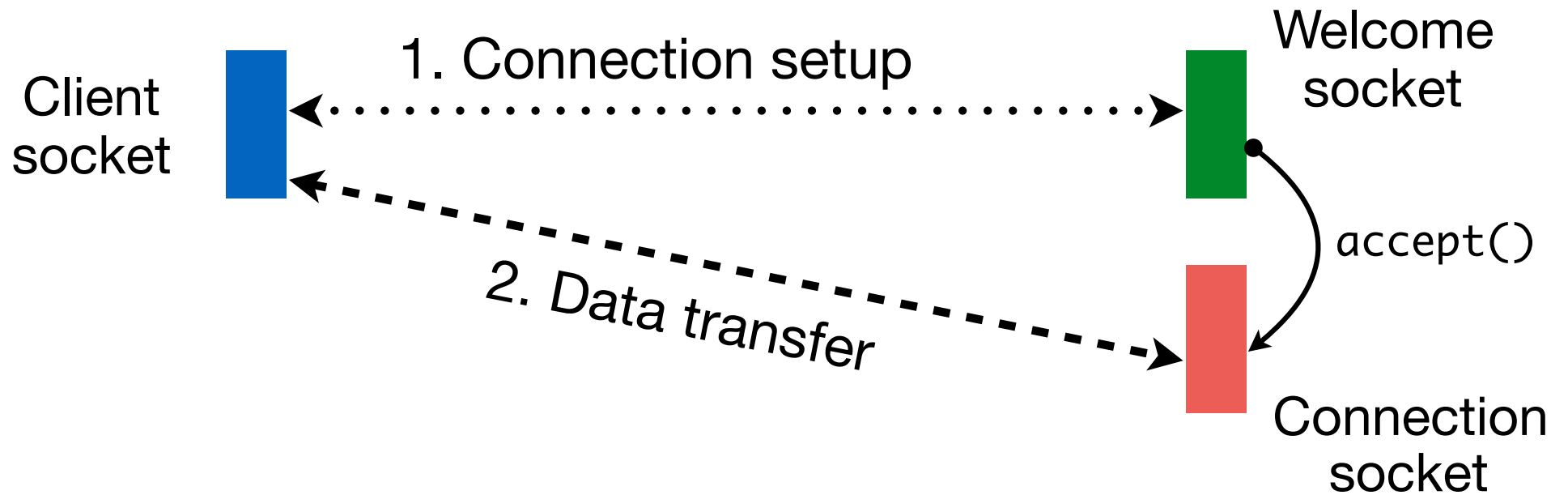
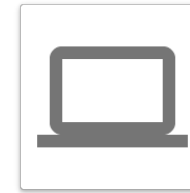
Workflow for TCP socket communication



Client



Server



TCP Server

```
from socket import *  
import sys  
  
SERV_PORT = 50000  
  
addr = ('127.0.0.1', SERV_PORT)  
s = socket(AF_INET, SOCK_STREAM)  
s.bind(addr)  
s.listen(1)  
print ('TCP server started ...')
```

```
while True:
```

```
sckt, addr = s.accept()
```

```
print ('New client connected ..')
```

```
while True:
```

```
    txtin = sckt.recv(1024)
```

```
    print ('Client> %s' %(txtin).decode('utf-8'))
```

```
    if txtin == b'quit':
```

```
        print('Client disconnected ..')
```

```
        print('Waiting for a new client ...')
```

```
        break
```

```
    else:
```

```
        txtout = txtin.upper()
```

```
        sckt.send(txtout)
```

```
sckt.close()
```

TCP Client

```
from socket import *  
import sys
```

```
MAX_BUF = 2048
```

```
SERV_PORT = 50000
```

```
addr = ('127.0.0.1', SERV_PORT)
```

```
s = socket(AF_INET, SOCK_STREAM)
```

```
s.connect(addr)
```

```
username = input('Enter your name: ')
while True:
    print ('%s> ' %(username), end='')
    sys.stdout.flush()
    txtout = sys.stdin.readline().strip()
    s.send(txtout.encode('utf-8'))
    if txtout == 'quit':
        break
    modifiedMsg = s.recv(2048)
    print (modifiedMsg.decode('utf-8'))
```

Use 'netstat - a' to verify the TCP connection

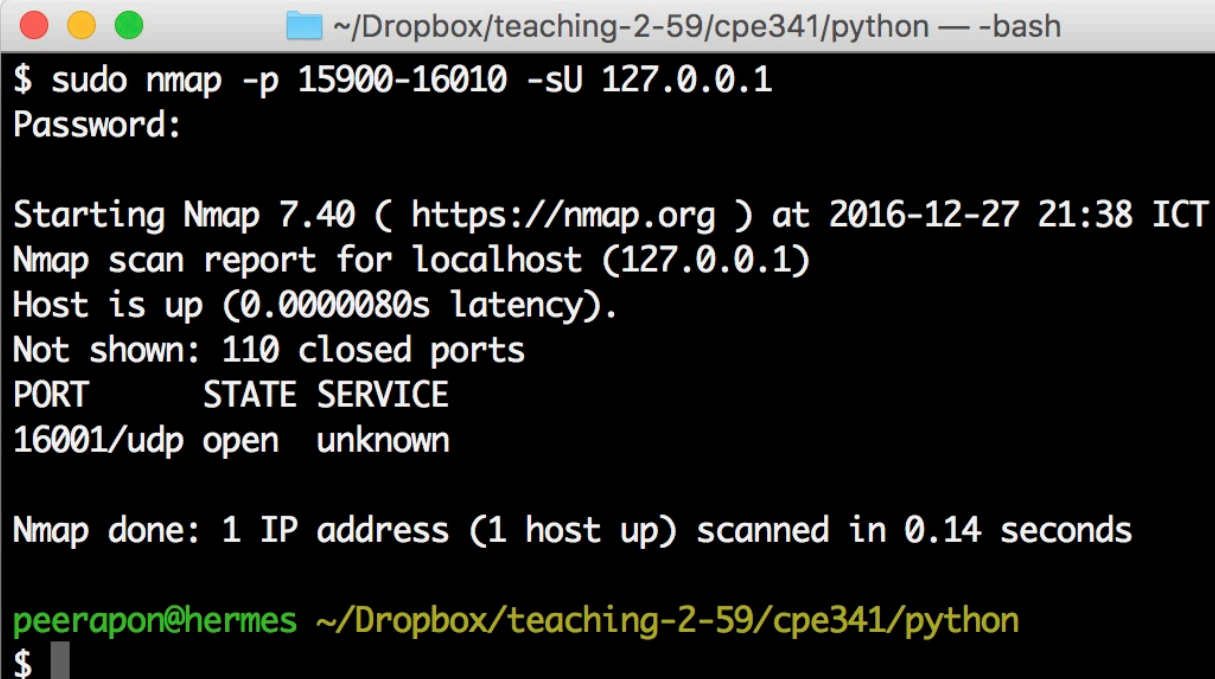
Note that TCP server does not quit when the client terminates.

UDP and TCP Socket Comparison

UDP	TCP
<code>socket(AF_INET, SOCK_DGRAM)</code>	<code>socket(AF_INET, SOCK_STREAM)</code>
No listen and accept at Server	Server listens and accepts
Client sends right away.	Client must connect first
Only one socket at Server	Another socket created after accepting a new connection
<code>sendto()</code> specifies both message and socket address	<code>send()</code> only needs a message argument
<code>recvfrom()</code> also returns socket address	<code>recv()</code> returns only message

- ❑ Scan open ports at a given IP address by using **nmap** utility (<https://nmap.org/download.html>)

```
# Scan port range 3000 to 4000 in local host for TCP server
nmap -p 3000-4000 127.0.0.1
# Scan port range 3000 to 4000 in local host for UDP server
nmap -p 3000-4000 -sU 127.0.0.1
# Same as above but does not ping before scanning
nmap -P0 -p 3000-4000 -sU 127.0.0.1
```

A terminal window with a dark background and light text. The title bar shows a folder icon and the path ~/Dropbox/teaching-2-59/cpe341/python. The terminal displays the command 'sudo nmap -p 15900-16010 -sU 127.0.0.1', the password prompt, and the output of the scan. The output indicates that the host is up and that port 16001 is open for UDP. The terminal ends with the prompt 'peerapon@hermes' and a cursor.

```
~/Dropbox/teaching-2-59/cpe341/python — -bash
$ sudo nmap -p 15900-16010 -sU 127.0.0.1
Password:

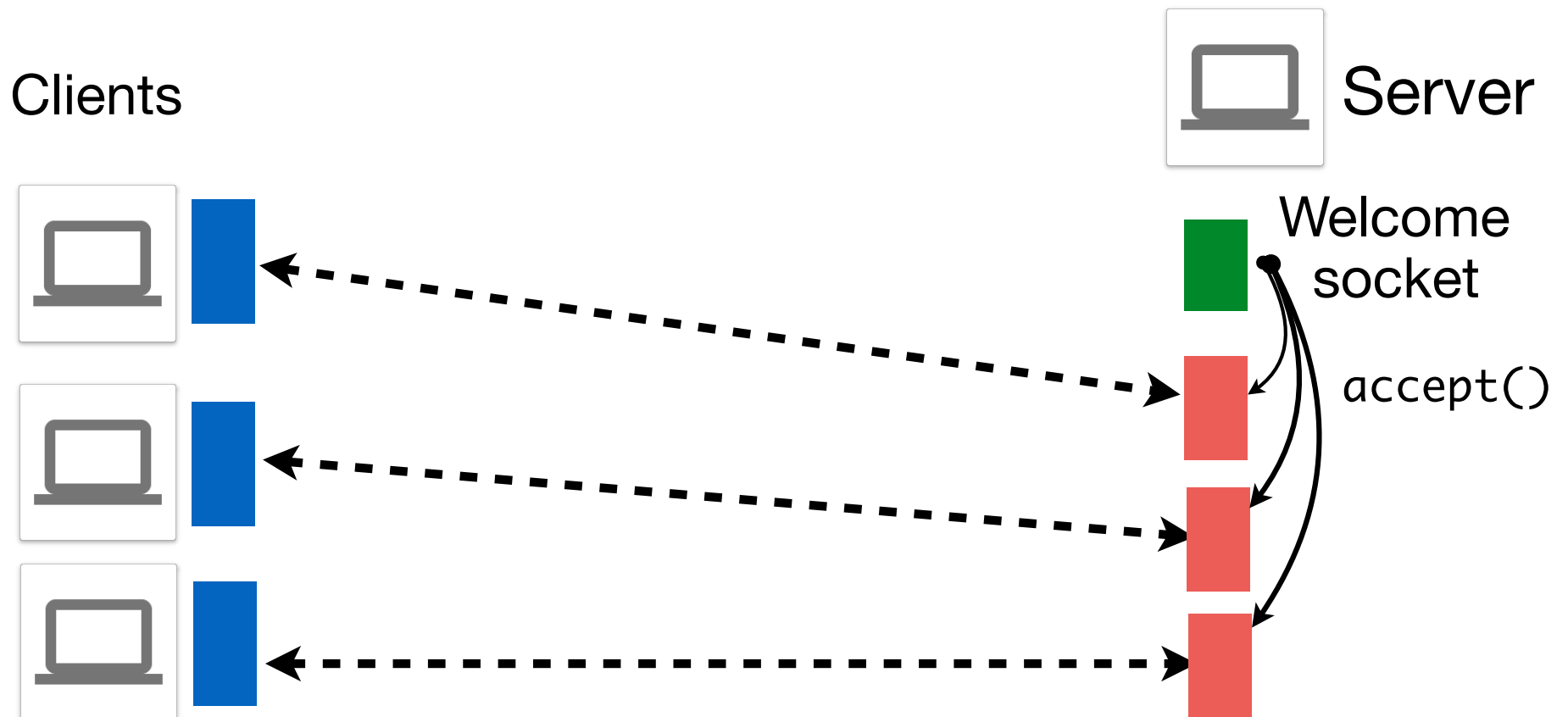
Starting Nmap 7.40 ( https://nmap.org ) at 2016-12-27 21:38 ICT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000080s latency).
Not shown: 110 closed ports
PORT      STATE SERVICE
16001/udp  open  unknown

Nmap done: 1 IP address (1 host up) scanned in 0.14 seconds

peerapon@hermes ~/Dropbox/teaching-2-59/cpe341/python
$
```

Concurrent Connections

- ❑ Server almost always serves multiple clients simultaneously.



Threaded Server

```
from socket import *
from threading import Thread
import os,sys

SERV_PORT = 50000

def handle_client(s):
    while True:
        txtin = s.recv(1024)
        print ('Client> %s' %(txtin).decode('utf-8'))
        if txtin == b'quit':
            print('Client disconnected ...')
            break
        else:
            txtout = txtin.upper()
            s.send(txtout)
    s.close()
    return
```

```
while True:
```

```
    sckt, addr = s.accept()
```

```
    ip, port = str(addr[0]), str(addr[1])
```

```
    print ('New client connected from ..' + ip + ':' + port)
```

```
try:
```

```
    Thread(target=handle_client, args=(sckt,)).start()
```

```
except:
```

```
    print("Cannot start thread..")
```

```
    import traceback
```

```
    traceback.print_exc()
```

Forked Server

```
import os
from socket import *

s = socket(AF_INET, SOCK_STREAM)
s.bind(("", 5000)); s.listen(5)

while True:
    sckt, addr = s.accept()
    if os.fork() == 0: # child process ..
        ...
        sckt.close(); os._exit(0);
    else: # Parent process
        sckt.close()
s.close()
```

Other Methods

- ☐ Asynchronous server
- ☐ SocketServer module (See <https://docs.python.org/2/library/socketserver.html>)