COMP3234B
Computer and Communication Networks

2nd semester 2023-2024

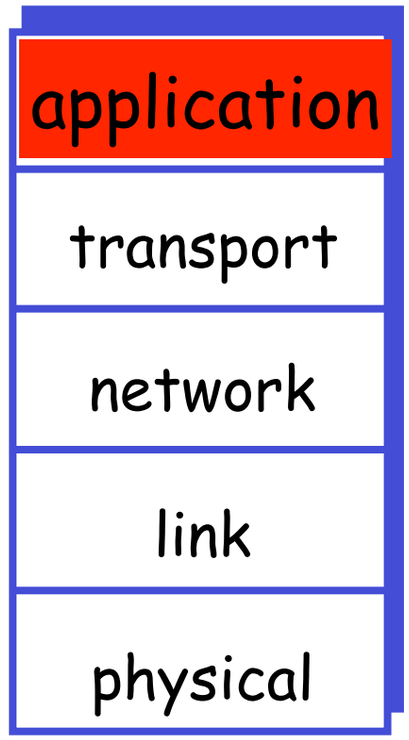
Application Layer (Part I)

Prof. C Wu

Department of Computer Science
The University of Hong Kong

Course roadmap

- Introduction to computer network
- Application layer
  - Key concepts and implementation aspects of network applications (ILO1)
  - Socket programming (ILO4)
  - Protocols for example applications (ILO2)
    Web, Email, DNS
- Transport layer
- Network layer
- Link layer



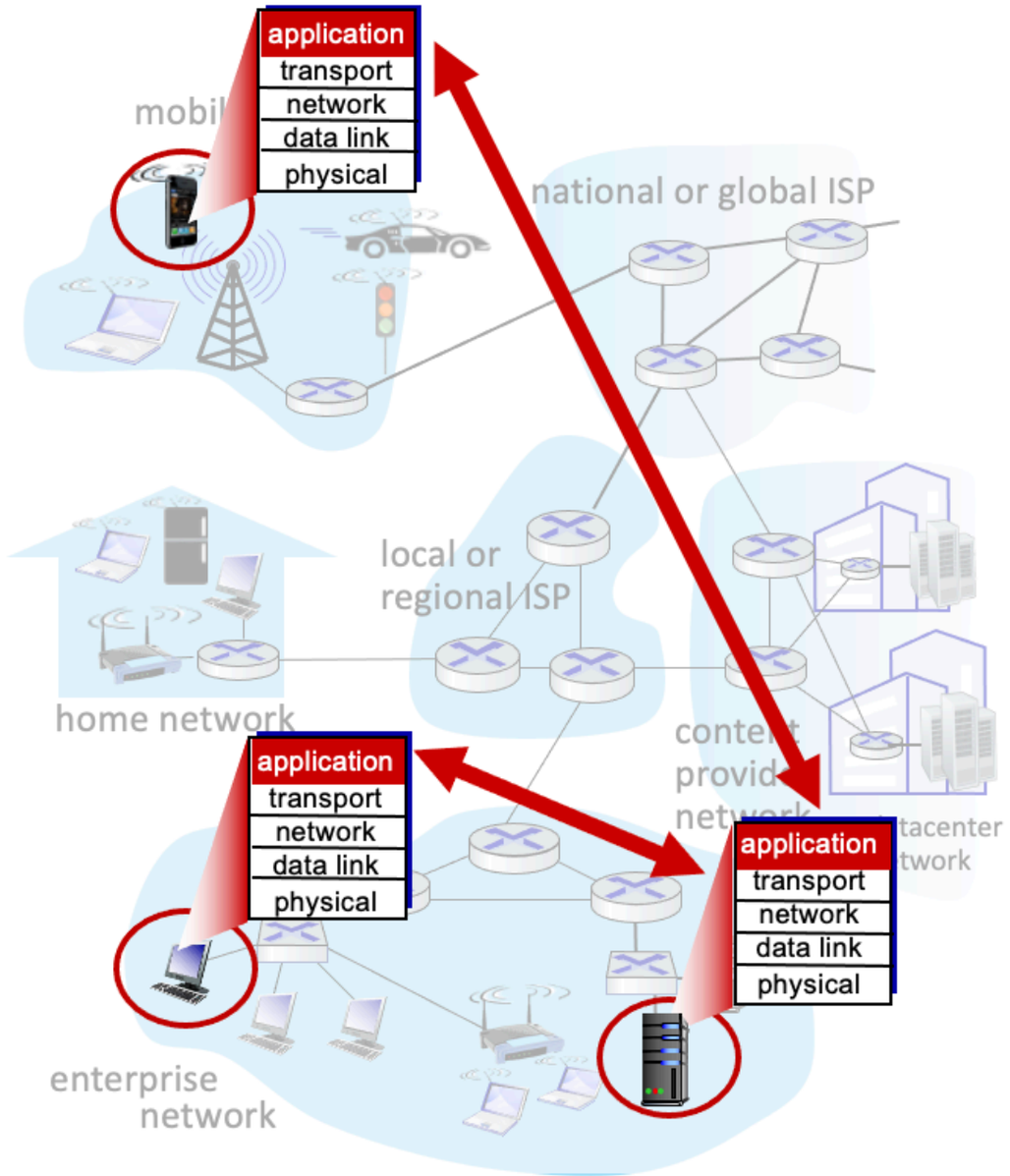
What is a network application?

- A program that
  - runs on (different) end systems
  - communicates over the network

    e.g., web browser software communicates with web server software

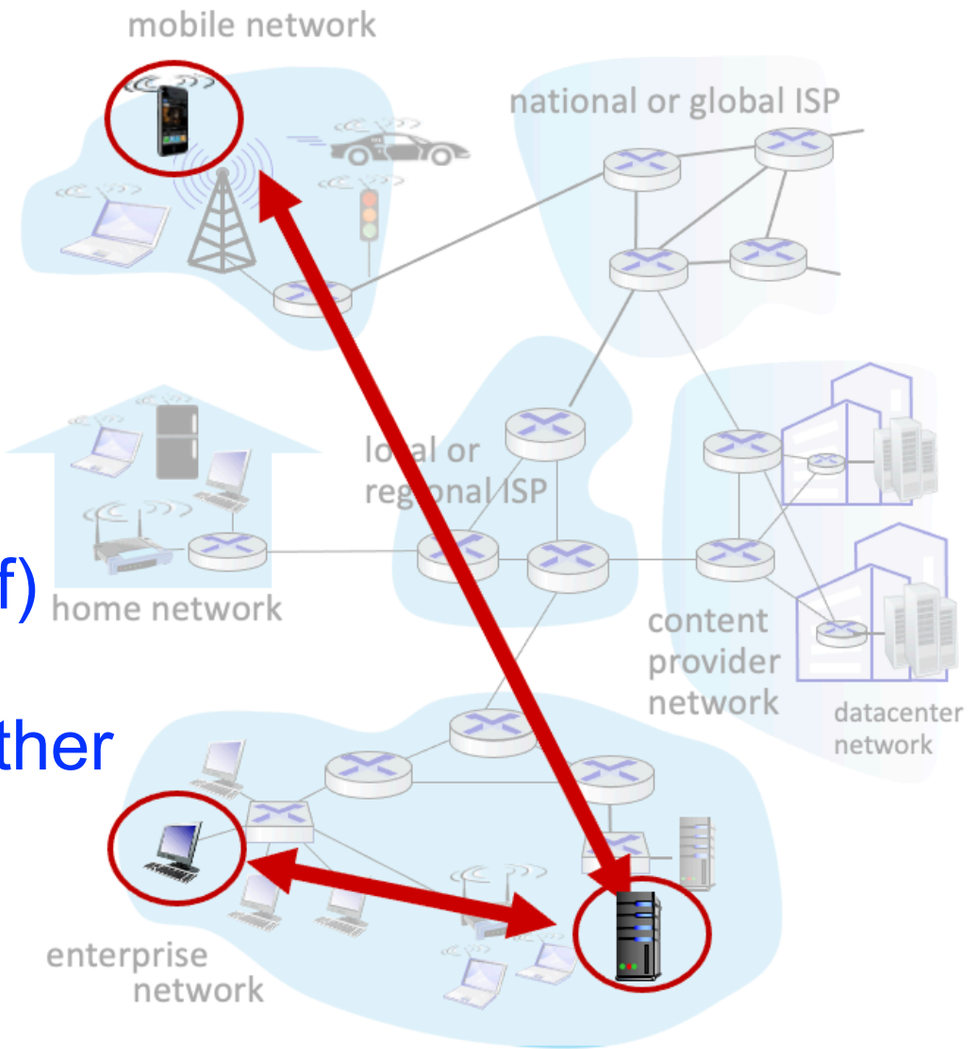
- Example network applications
  - email, web, instant messaging, remote login, network games, VoIP, video streaming, ...

- Application architecture
  - client-server
  - peer-to-peer (P2P)
  - hybrid of client-server and P2P



Client-server applications

- Clients request/receive services/data from servers
- Server
  - always-on host
  - permanent IP addresses
  - server cluster/farm, data center for scaling
- Client
  - may be intermittently connected (on and off)
  - may have dynamic IP addresses
  - does not communicate directly with each other
- Process
  - The program running on a host

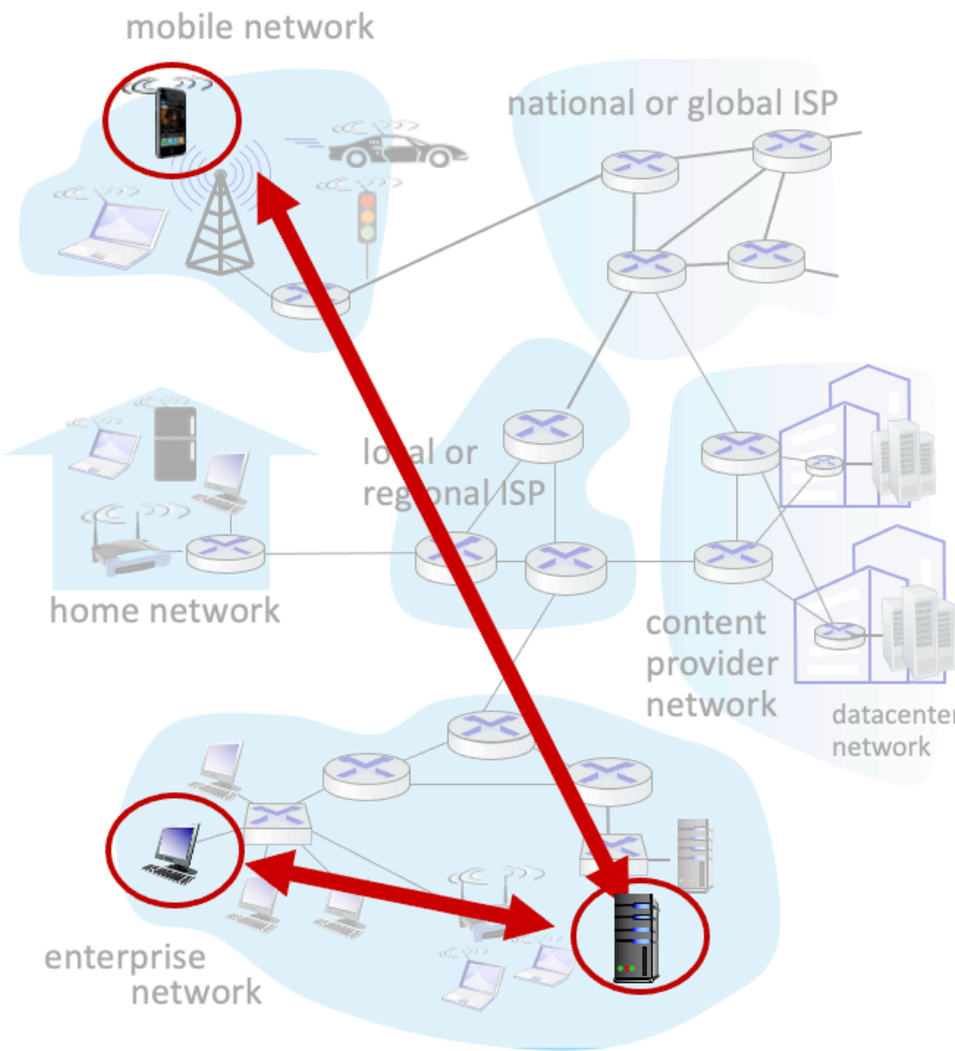


Process

- Client process
  - process that initiates communication
- Server process
  - process that waits to be connected

- Processes in different hosts communicate by exchanging *messages*
- Process sends/receives messages to/from its *socket*

Process: program running within a host



Socket

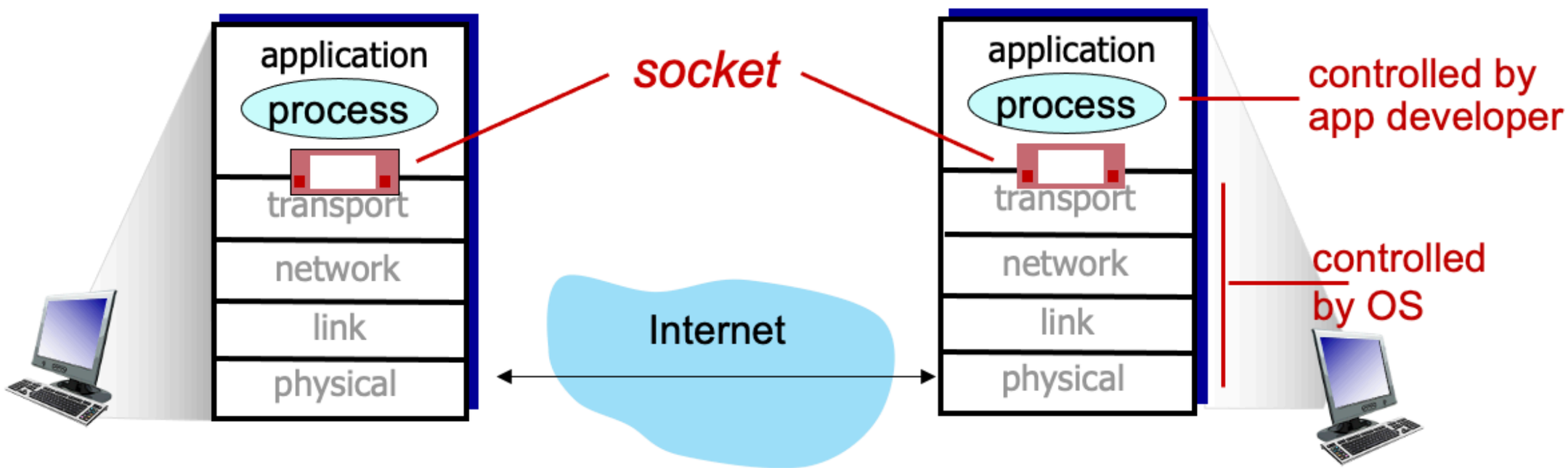
- What is a socket?
  - A software interface between application layer and transport layer

    via which application process pushes messages to transport service, or receives messages from the transport service.

  - socket analogous to door

    sending process shoves message out door

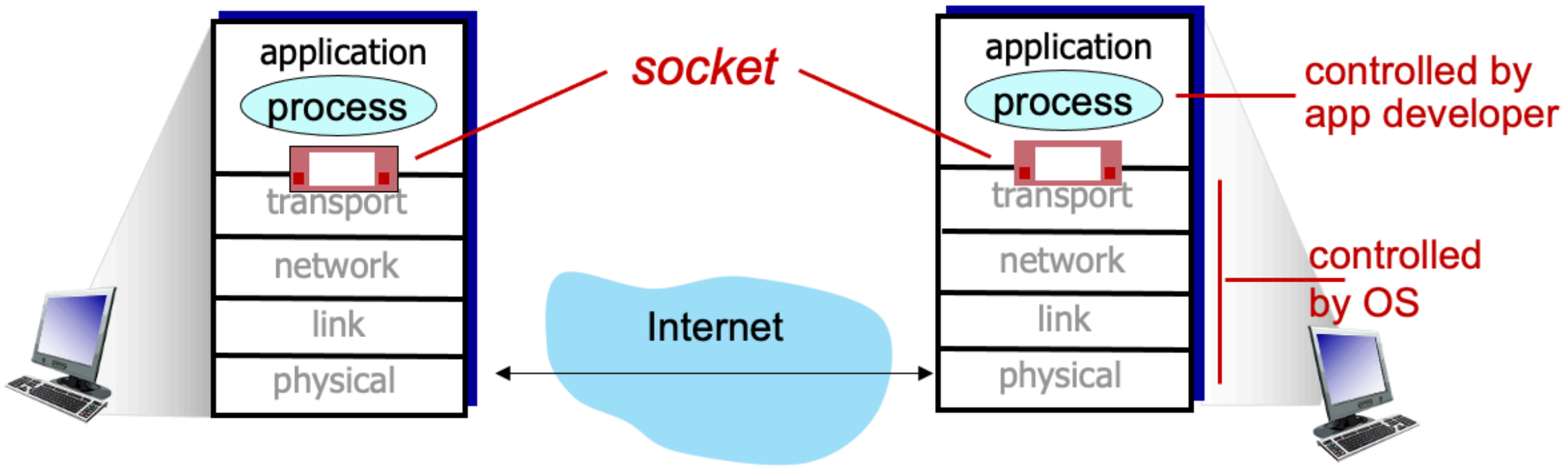
    sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



Socket (cont'd)

- Socket API
  - Application Programming Interface (API) between application layer and transport layer
  - has control on the transport layer

    choices of transport protocol and a few parameters (e.g., max segment size)



Transport services available to applications

	TCP	UDP
Data Loss	Reliable data transfer between sending and receiving processes. Guaranteed no loss.	Best effort Service. No guarantee of losses.
Delay	Connection setup/teardown required	No connection setup/ teardown delay involved
Example applications	Web, Email, file transfer/ download, Instant Messaging	real-time or on-demand audio/video streaming, games, VoIP

Different applications have different requirements for performance => choice of different transport protocols !

Addressing a process

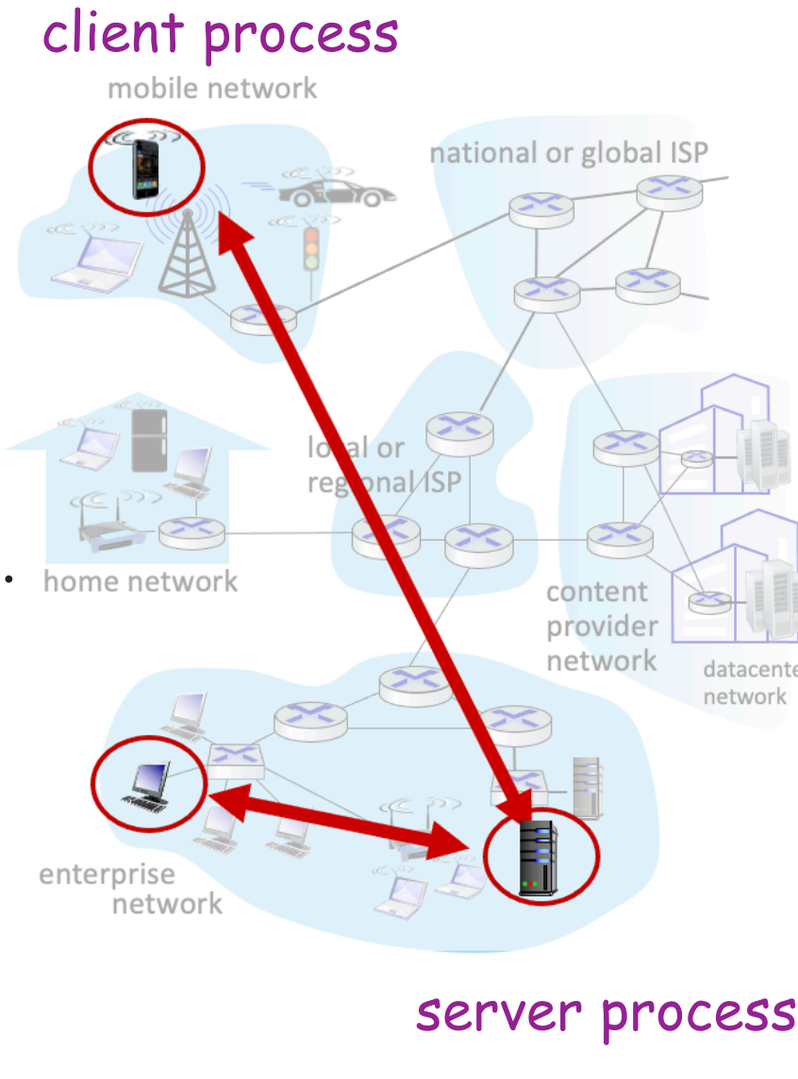
- To receive messages, process must have identifier
- IP address
  - a 32-bit quantity that uniquely identifies the host
  - Does IP address suffice to identify a process?

    No. Many processes may be running on the same host.
- Identifier of a process must include
  - IP address of the host
  - port number associated with the process

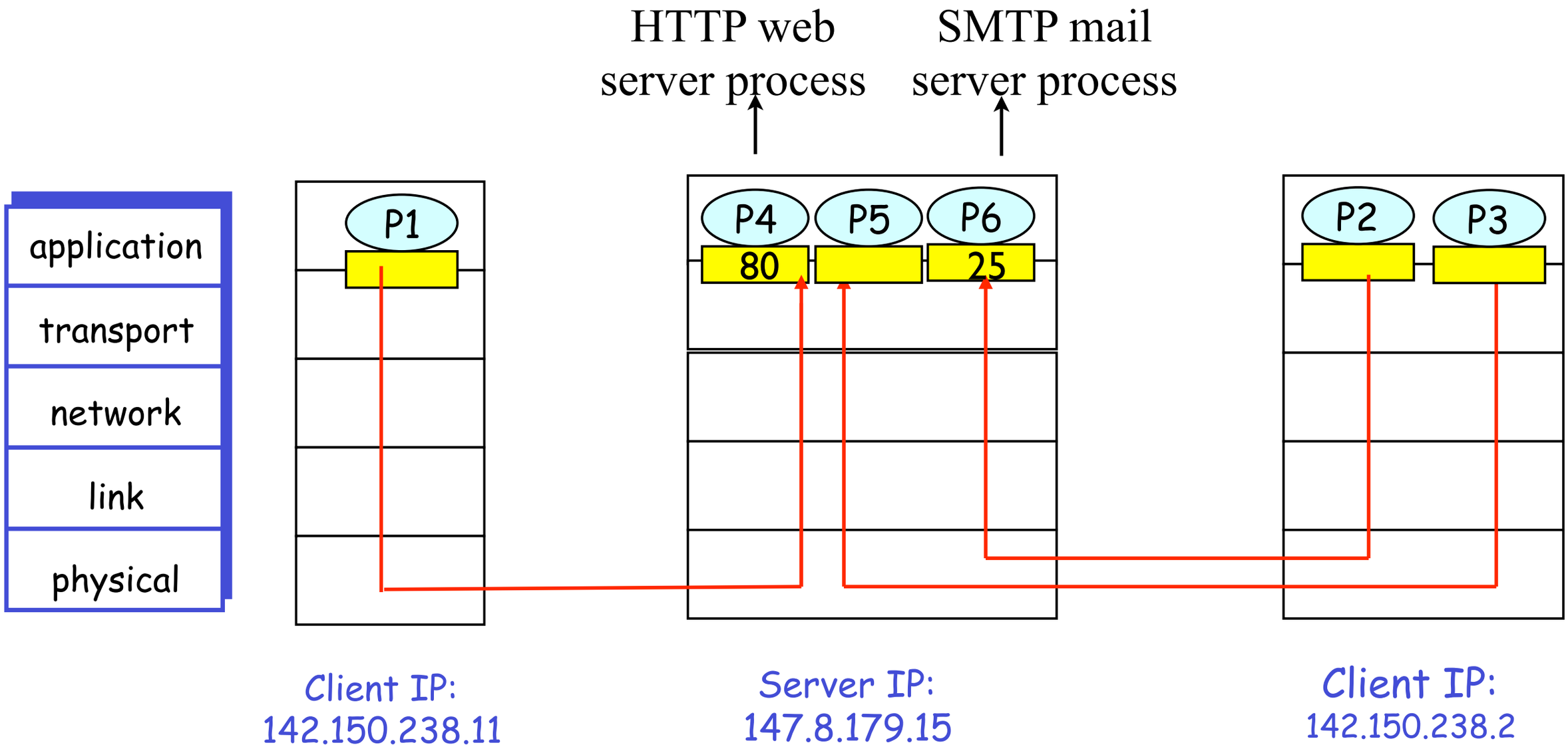
    A number to distinguish processes on a host

    0 — 65535

    e.g., HTTP server: 80, SMTP mail server: 25



Addressing a process (cont'd)



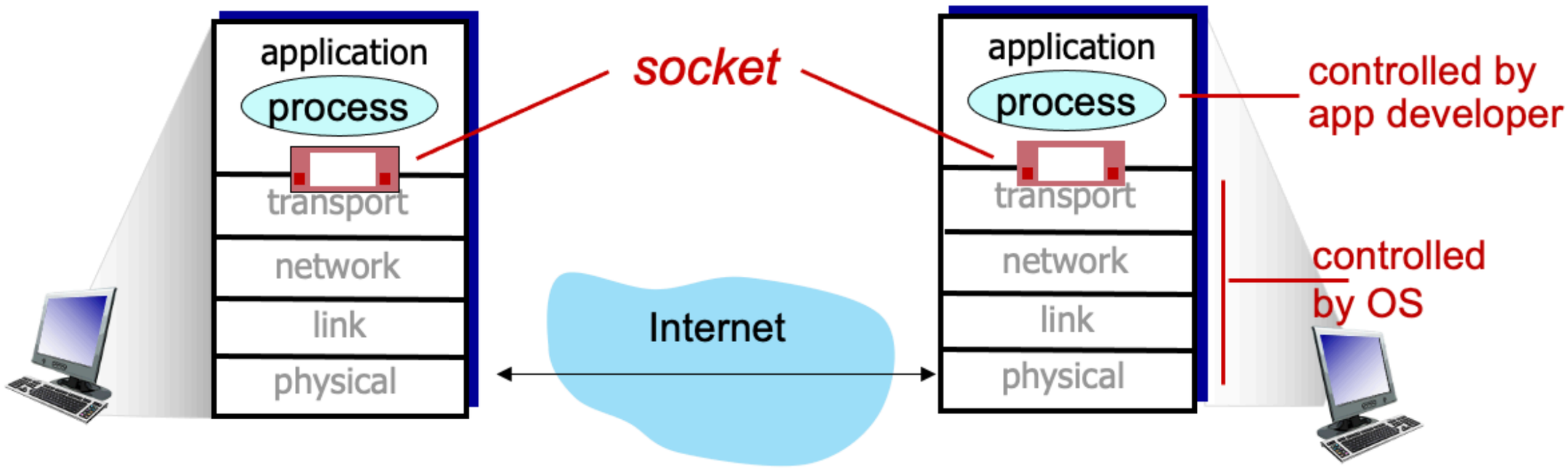
Socket programming

- Socket programming
  - Program network applications that communicate using socket API

socket

- A software interface via which

  application process pushes/receives messages to/from transport service
- Application Programming Interface (API) between application layer and transport layer



Socket API

- Application creates, uses, releases socket(s) using socket APIs
- Choice of TCP or UDP as transport protocol

  TCP: connection-oriented, reliable transfer of bytes from one process to another

  UDP: connectionless, unreliable transfer of packets from one process to another

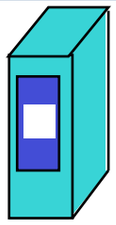
Was originally designed for BSD Unix

Now is industry standard

— available on almost all operating systems

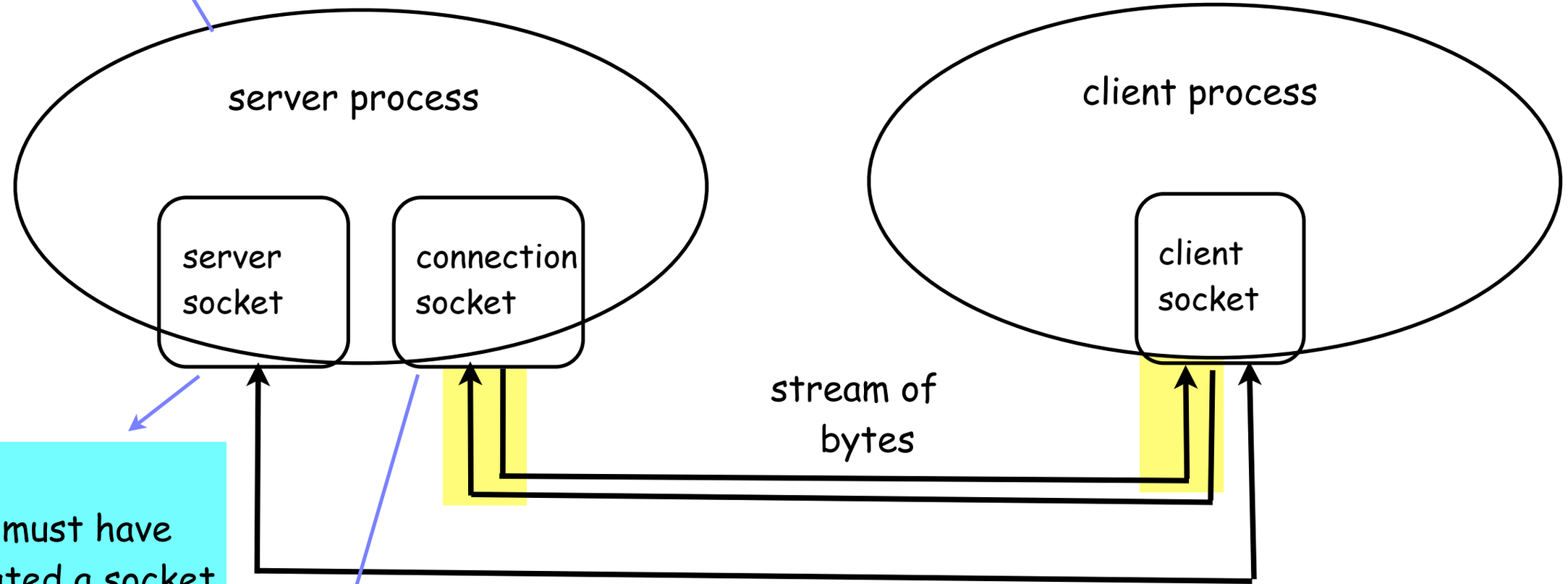
— available in different “forms” under different Languages

Socket programming using TCP: an illustration



client process creates client local TCP socket, specifying IP address, port number of server process

server process must first be running



server must have created a socket that welcomes client's contact

the socket that handles the particular connection

Python Socket modules

- Socket modules
  - Socket Module: low-level networking interface (close to the BSD API)
  - SocketServer Module: provides classes that simplify the development of server programs

- To use sockets in your Python programs, you must include the Socket Module in your programs
  - "import socket"

    It does not import the names of the functions defined in socket module directly into the current module's symbol table

    access functions/names in the module using the module name, e.g, socket.socket(socket.AF_INET, socket.SOCK_STREAM)

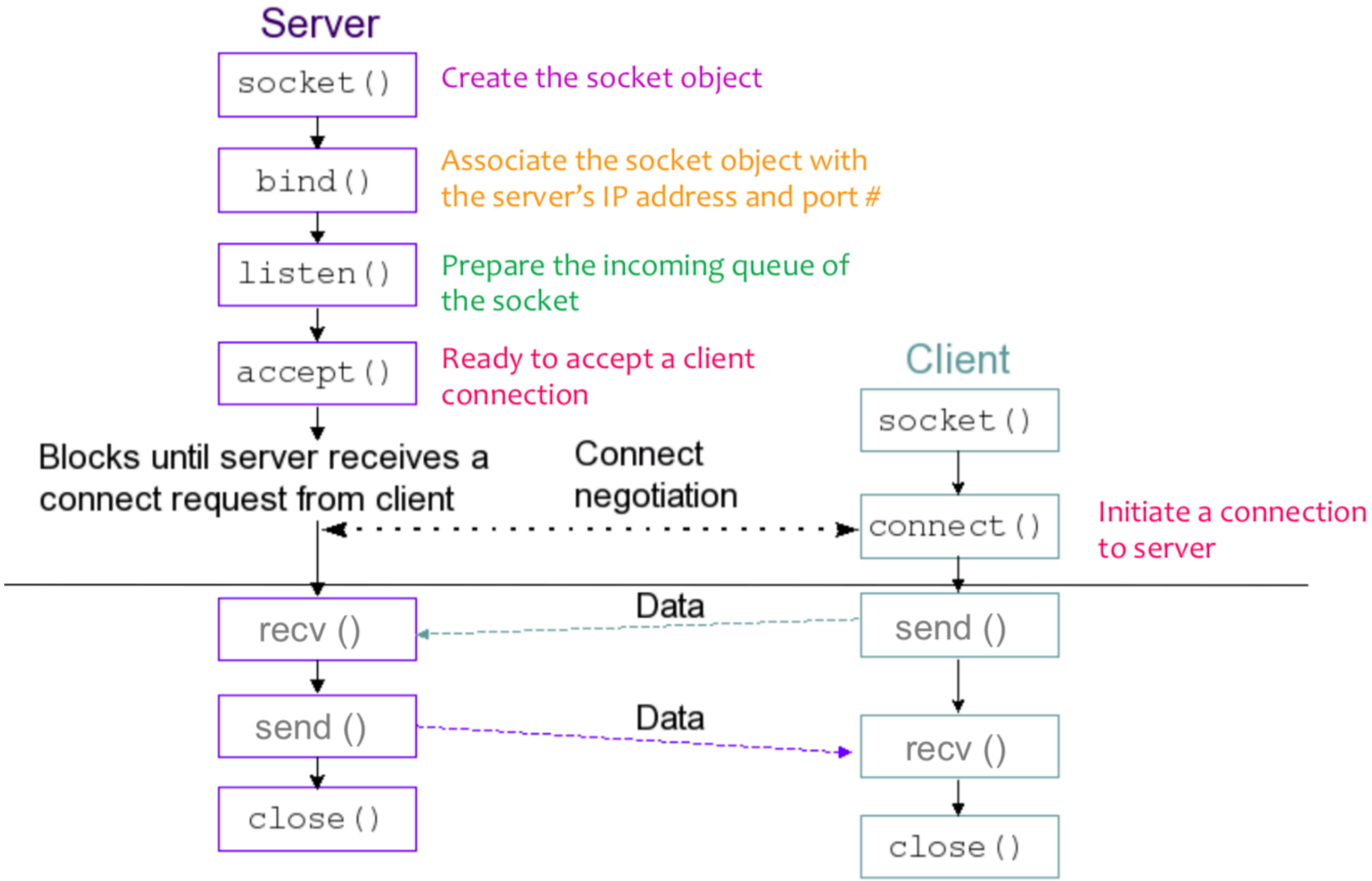
  - "from socket import *"

    It imports names from a module directly into the current module's symbol table

    access functions/names directly, e.g., socket(AF_INET, SOCK_STREAM)

    (read more at https://docs.python.org/3/tutorial/modules.html)

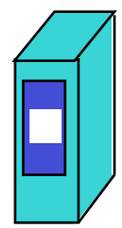
Python Socket APIs for connection-oriented mode (TCP)



Detailed APIs: refer to Lab 1 handout

Example client-server application using TCP

1) client reads a line from keyboard, sends to server via socket
2) server reads the line from socket
3) server converts the line to uppercase and sends it back to client
4) client reads modified line from socket and prints it on monitor screen



keyboard monitor

inputstream outputstream

inputstream outputstream

server process

welcoming socket

connection socket

outputstream inputstream

client process

client socket

bytes

Three-way handshake

Example client-server application: TCP client

```
from socket import *

clientSocket = socket(AF_INET, SOCK_STREAM)

serverName = "localhost"

serverPort = 12000


clientSocket.connect( (serverName, serverPort) )

# Get input for sending
sentence = input("Input a lowercase sentence:")

clientSocket.send(sentence.encode())

modifiedSentence = clientSocket.recv(1024)

print("From Server:", modifiedSentence.decode())

clientSocket.close()
```

Create a socket, specifying the address family to use in the socket and socket type

AF_INET: an address family (default) which represents address in a pair, (host, port), where host is a hostname (e.g., 'cs.hku.hk' or an IPv4 address (e.g, '147.8.179.10'), and port is the port number

SOCK_STREAM: the socket type (default) for sending/receiving TCP byte stream

Connect to a remote socket at the specified address (format according to the address family specified in above socket() call)

Send data to the socket (Python String encode() return utf-8 encoded version of the string); return the number of bytes sent

Receive data from the socket (parameter is bufsize - maximum amount of data to be received); return a bytes object representing the data received

Close the socket

Example client-server application: TCP server (sequential)

```
from socket import *

serverPort = 12000

serverSocket = socket(AF_INET, SOCK_STREAM)

serverSocket.bind( ("", serverPort) )

serverSocket.listen(5)

print("The server is ready to receive")

while True:

    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024)

    capitalizedSentence = sentence.decode().upper()

    connectionSocket.send(capitalizedSentence.encode())

    connectionSocket.close()

serverSocket.close()
```

Create a socket, specifying the address family to use in the socket and socket type

Bind the socket to the specified address (format according to the address family specified in above socket() call)

Begin listening to incoming connection requests; parameter is backlog (optional), specifying the number of unaccepted connections that the system will allow before refusing new connections

Accept a connection; return a pair, (conn, addr), where conn is a new socket object usable to send and receive data on the connection, and addr is the address bound to the socket on the other end of the connection

Receive data from the socket

Send data to the socket

Close connection to this client

The server only accepts and handles one client connection at a time: sequential (iterative) server

Example client-server application: TCP server (concurrent)

Concurrent server: handle multiple client connections at the same time

```
import socket
import threading

def thd_func(client):
    connectionSocket, addr = client
    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.decode().upper()
    connectionSocket.send(capitalizedSentence.encode())
    connectionSocket.close()

serverPort = 12000

serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

serverSocket.bind( ("", serverPort) )

serverSocket.listen(5)

print("The server is ready to receive")

while True:

    client = serverSocket.accept()
    newthd = threading.Thread(target=thd_func, args=(client,))
    newthd.start()

serverSocket.close()
```

Python function definition (https://docs.python.org/3/reference/compound_stmts.html#def):

def funcname ([parameter_list]):

we need to use "socket.xx" since we import socket module by "import socket"

create a new thread:

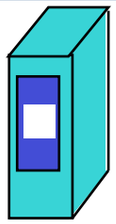
target: the callable object to be invoked by running the thread

args: argument tuple for the target invocation

start the thread's activity

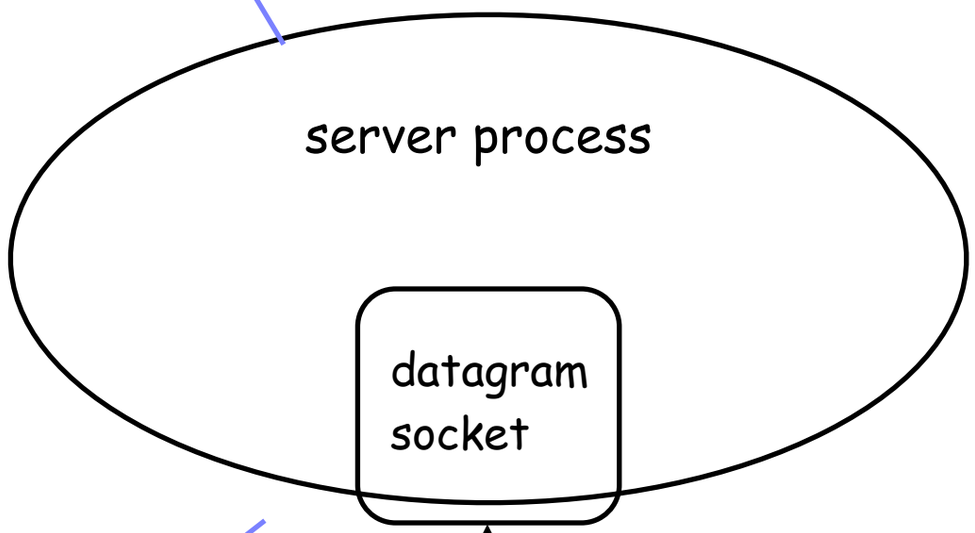
More about Python threading at https://docs.python.org/3/library/threading.html

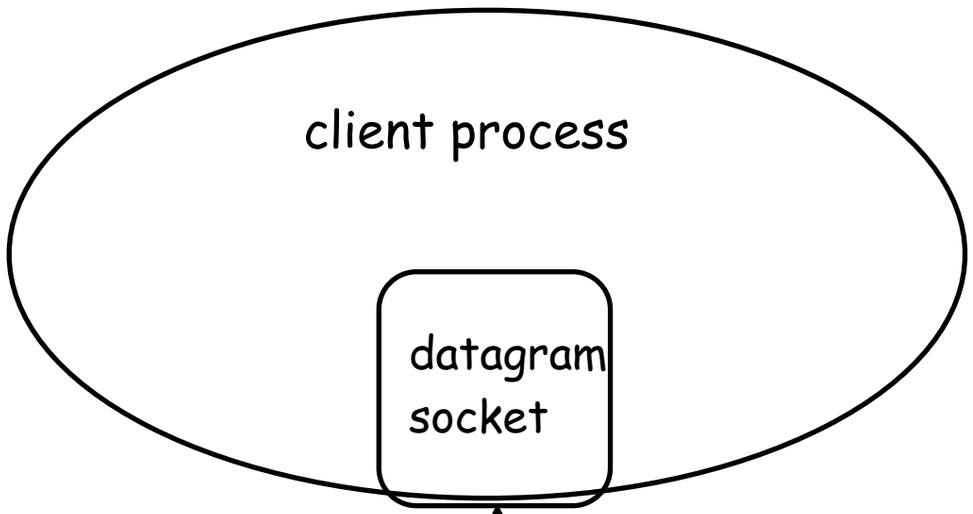
Socket programming using UDP: an illustration



client process explicitly
attaches IP address and port
of server to each packet

server process must first be running





server creates
socket to receive
datagram packets
from client

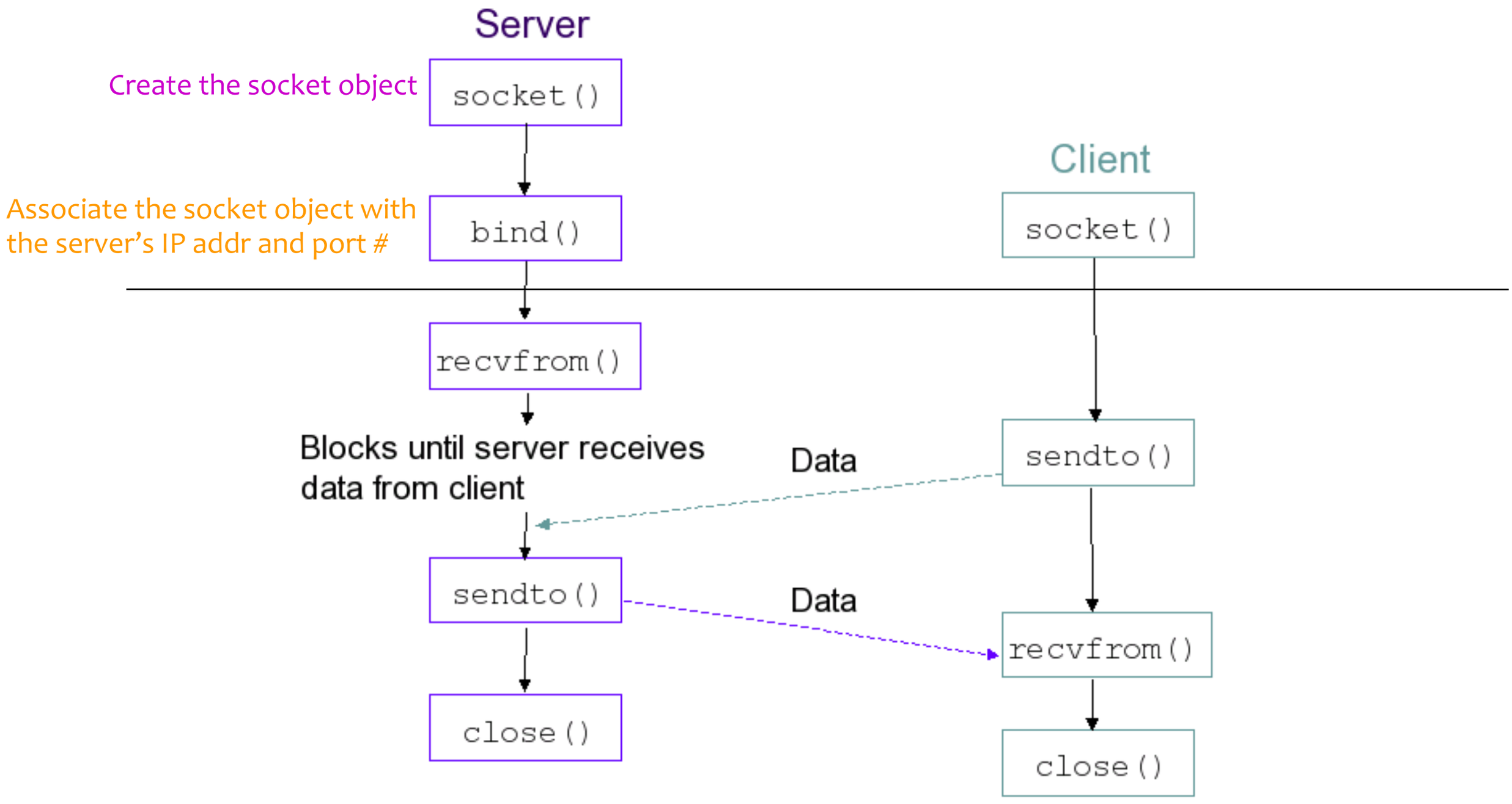
no streams are attached to the sockets!

packets (server IP, server port)

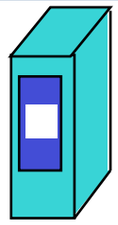
packets (client IP, client port)

UDP: no handshaking, no need of separate sockets to handle different clients

Python Socket APIs for connectionless mode (UDP)

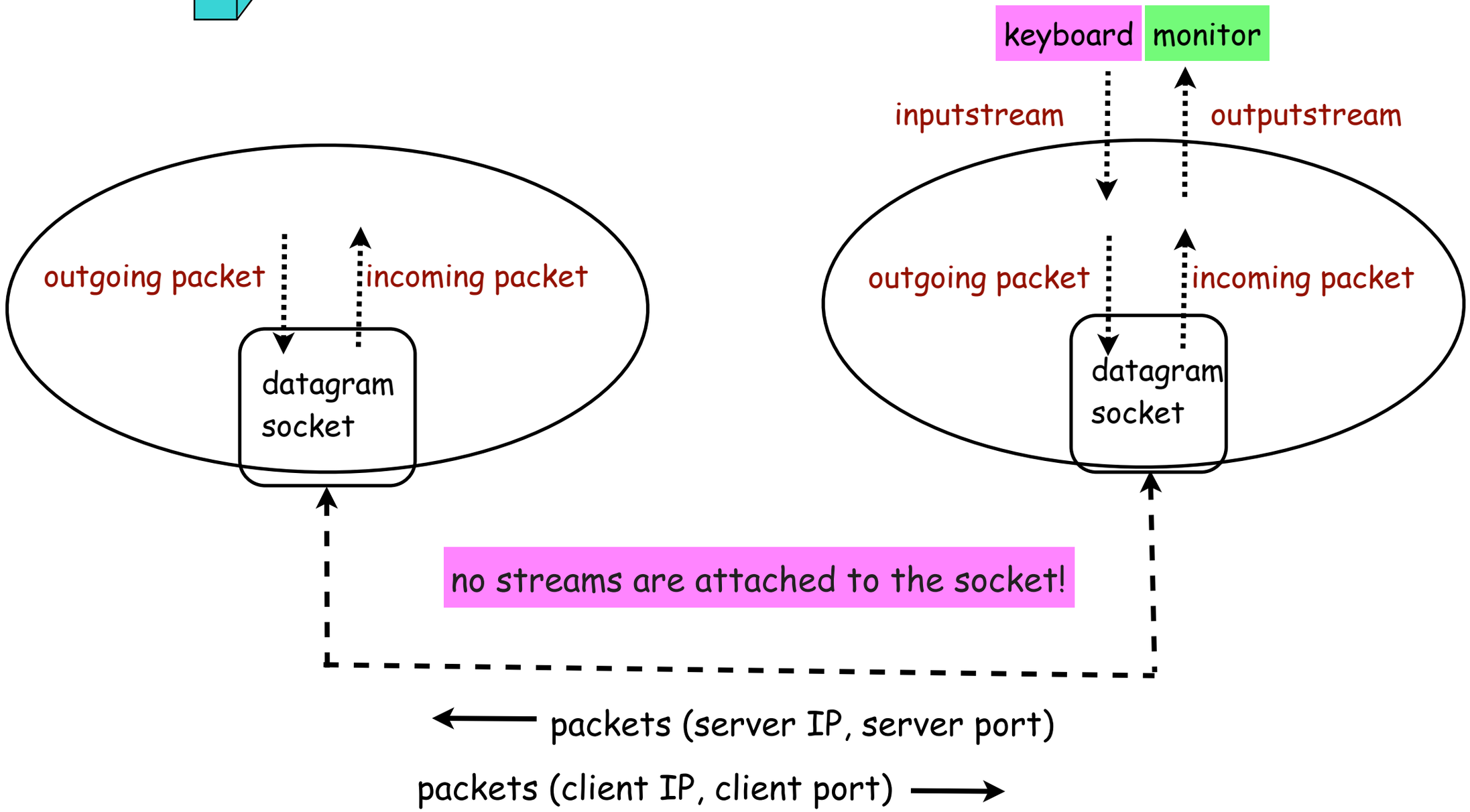


Example client-server application using UDP



server process

client process



Example client-server application: UDP client

```
from socket import *

clientSocket = socket(AF_INET, SOCK_DGRAM)

serverName = "localhost"

serverPort = 12000


# Get input for sending
sentence = input("Input a lowercase sentence:")

clientSocket.sendto(sentence.encode(), (serverName, serverPort))

modifiedSentence, serverAddress = clientSocket.recvfrom(1024)

print("From Server:", modifiedSentence.decode())

clientSocket.close()
```

Create a socket, specifying the address family to use in the socket and socket type

SOCK_DGRAM: the socket type for sending/receiving datagrams

Send data to the socket; parameters are bytes to send and address of the destination socket); return the number of bytes sent

Receive data from the socket (parameter is bufsize); return a pair, (bytes, address), where bytes is an object representing the data received and address is the address of the socket sending the data object

Close the socket

Example client-server application: UDP server

```
from socket import *

serverPort = 12000

serverSocket = socket(AF_INET, SOCK_DGRAM)

serverSocket.bind( ("", serverPort) )

print("The server is ready to receive")

while True:

    sentence, clientAddress = serverSocket.recvfrom(1024)

    capitalizedSentence = sentence.decode().upper()

    serverSocket.sendto(capitalizedSentence.encode(), clientAddress)

serverSocket.close()
```

Create a socket, specifying the address family to use in the socket and socket type

SOCK_DGRAM: the socket type for sending/receiving datagrams

Bind the socket to the specified address (format according to the address family specified in above socket() call)

Receive data from the socket (parameter is bufsize); return a pair, (bytes, address), where bytes is an object representing the data received and address is the address of the socket sending the data object

Send data to the socket; parameters are bytes to send and address of the destination socket); return the number of bytes sent

- Required reading:
  - Chapter 2.1, 2.7, Computer Networking: A Top Down Approach (8th Edition)

- Python socket programming references:
  - Python Socket Programming HowTo: https://docs.python.org/3.11/howto/sockets.html
  - Socket – Low-level networking interface: https://docs.python.org/3/library/socket.html
  - TutorialsPoints – Python 3 Network Programming: https://www.tutorialspoint.com/python3/python_networking.htm

- Python tutorials:
  - W3Schools – Python Tutorial: https://www.w3schools.com/python/
  - TutorialsPoints – Python Tutorial: https://www.tutorialspoint.com/python3/index.htm

- Acknowledgement:
  - Some materials are extracted from the slides created by Prof. Jim F. Kurose and Prof. Keith W. Ross for the textbook