SIT32004
ICT Application Development

Network Programming & Telegram

Prof. Changbeom Choi

Socket for Network(Socket) Programming

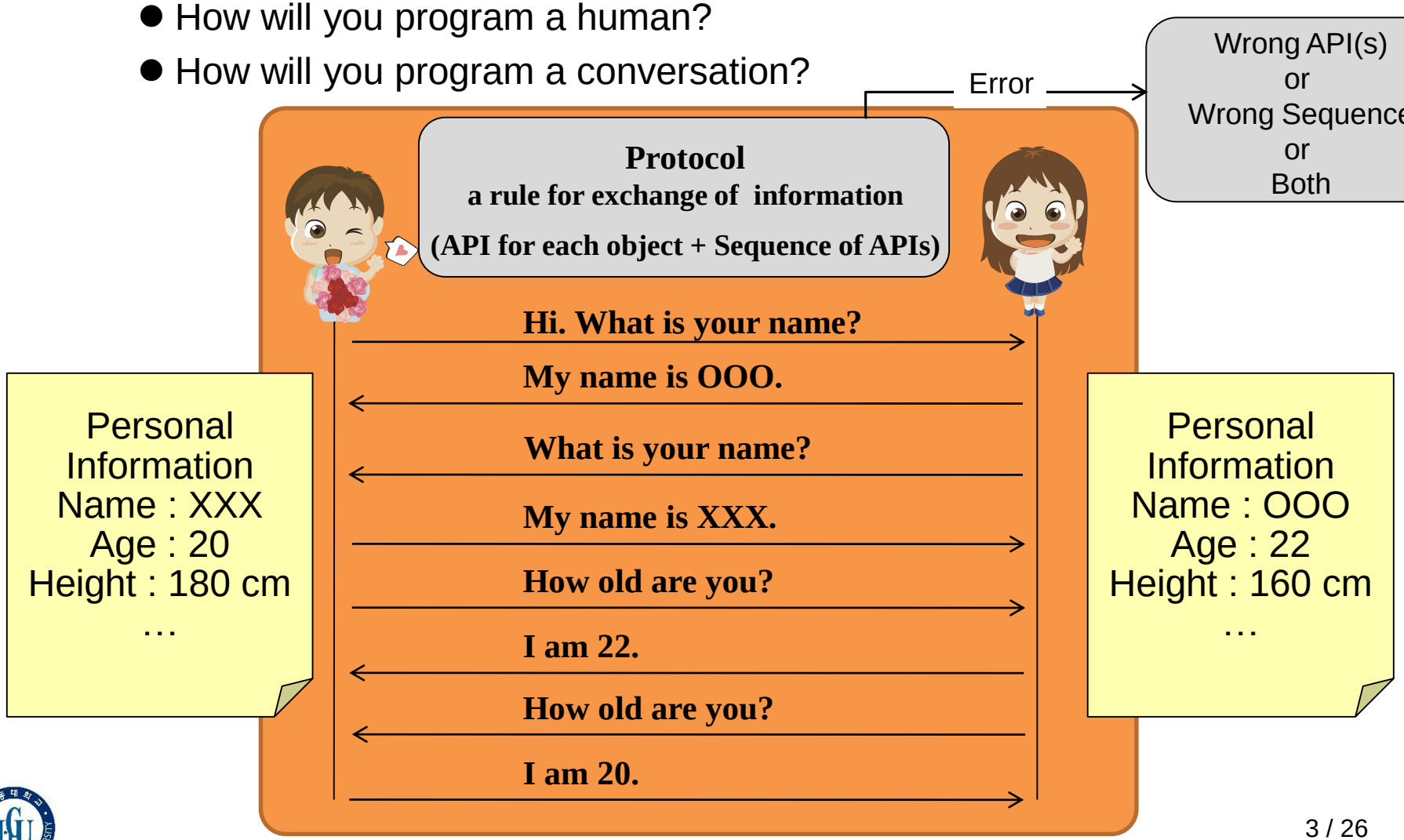
- History
  - –Socket API originated with the 4.2 BSD (Berkeley Unix) system released in 1983

- Sockets : A means to speak to other programs using file descriptors.
  - –A file descriptor is an integer associated with an open file.
  - –An abstract key to access a file, which can be used for a network connection
  - –Kinds of Sockets
    - »DARPA Internet addresses(Internet Sockets)
    - »Unix Sockets
    - »X.25 Sockets, etc.

- Types of Internet Sockets
  - – SOCK_STREAM uses TCP (Transmission Control Protocol) Connection oriented and Reliable
  - – SOCK_DGRAM uses UDP (User Datagram Protocol) Connectionless and Unreliable

OO Programming Example: Greeting (1/2)

- How will you program the greeting between two person?
  - How will you program a human?
  - How will you program a conversation?



OO Programming Example: Greeting (2/2)

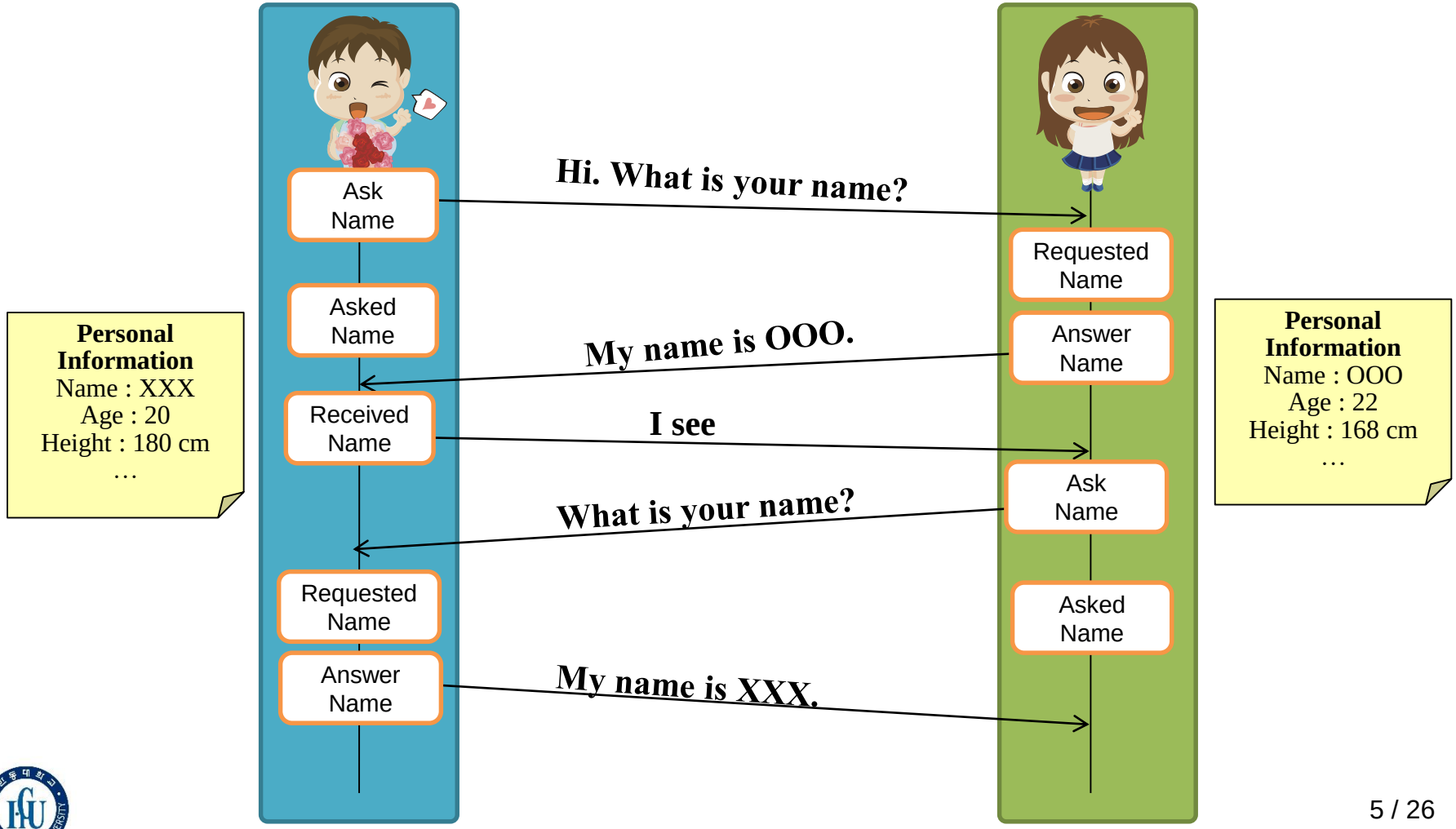
- Simulating greetings
  - Target: Two persons
  - Contents
    - » One person asks name
    - » Other person answer the question
    - » One person asks age
    - » Other person answer the question

```
class Human(object):
    def __init__(self, _name, _age):
        self.name = _name
        self.age = _age

    def ask_name(self):
        print("{}: Hi. What is your name?".format(self.name))

    def answer_name(self):
        print("{}: My name is {}.".format(self.name, self.name))

    def ask_age(self):
        print("{}: How old are you?".format(self.name))

    def answer_age(self):
        print("{}: I am {}.".format(self.name, self.age))
```

```
Hong, Gil Dong: Hi. What is your name?
Kim, Younghee: My name is Kim, Younghee.
Hong, Gil Dong: How old are you?
Kim, Younghee: I am 22.
```

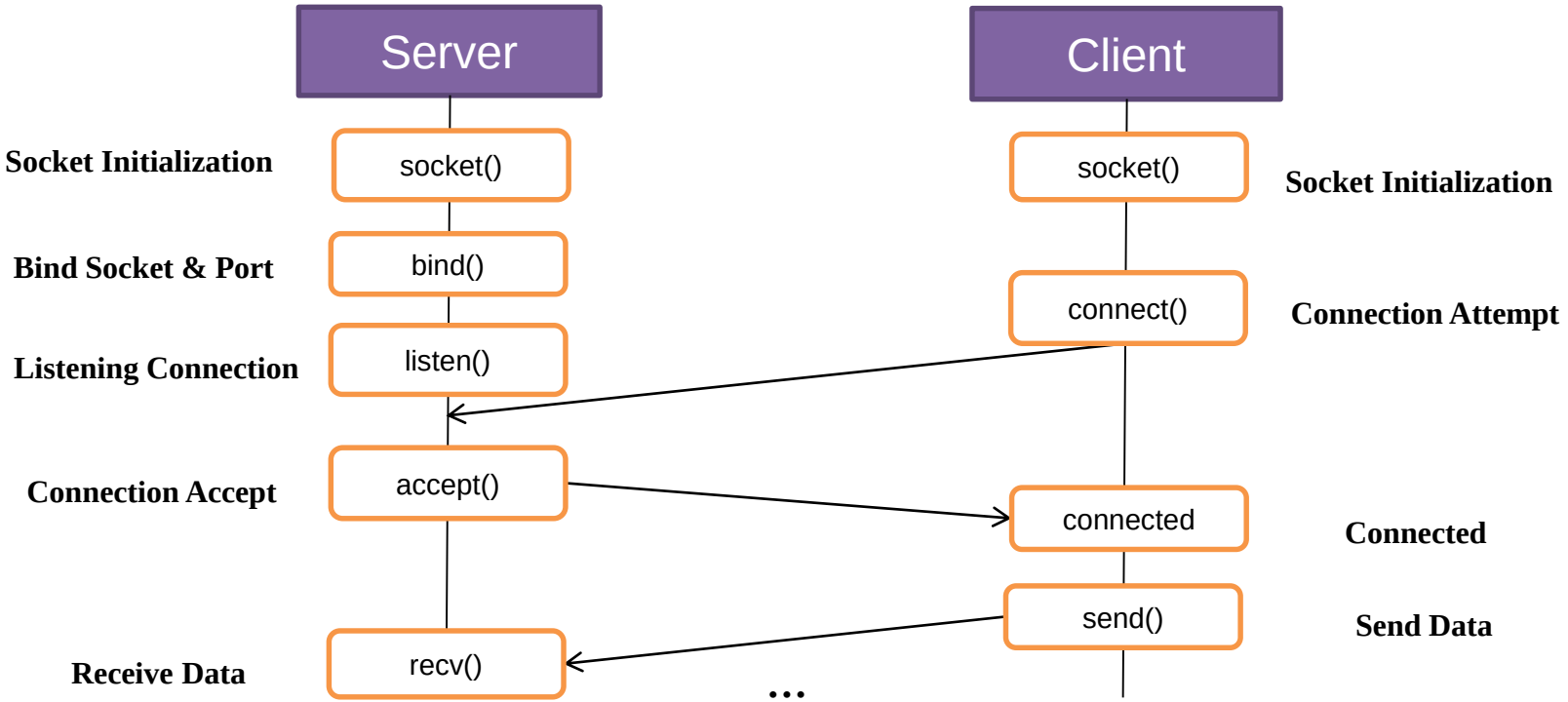
Protocol Example: Greeting Revisited (1/2)

- How will you program the greeting between two person in the Network?
  - Protocol: Define procedure and data type for communication



TCP Sockets

- Characteristics
  - Reliable
    - » During the communication, a sender detects the packet drop, and retransmit the dropped packet
  - In-order data delivery
    - » Data is read by your application in the order it was written by the sender

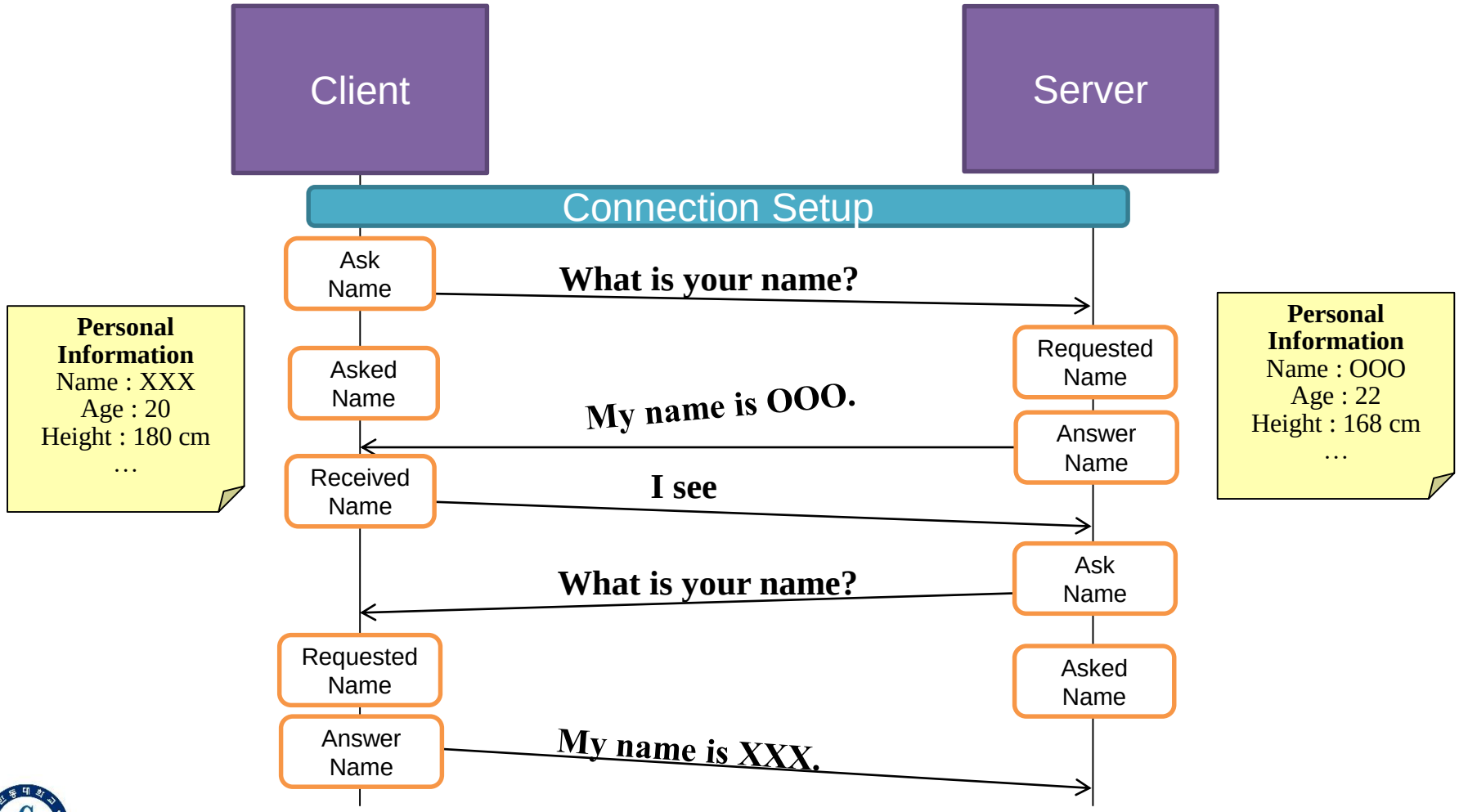


Server and Client[2, 3]

- **Definition**
  - Server: A system that responds to requests across a computer network worldwide to provide, or help to provide, a network or data service
  - Client: A hardware or software that accesses a remote service on another computer
- **APIs**
  - socket()
    - » Create a new socket and return a socket descriptor
  - bind()
    - » What port I am on / what port to attach to
  - connect()
    - » Connect to a remote host
  - listen()
    - » Waiting for someone to connect to my port
  - accept()
    - » Get a file descriptor for a incoming connection
  - send()
    - » Send data over a connection
  - recv()
    - » Receive data over a connection

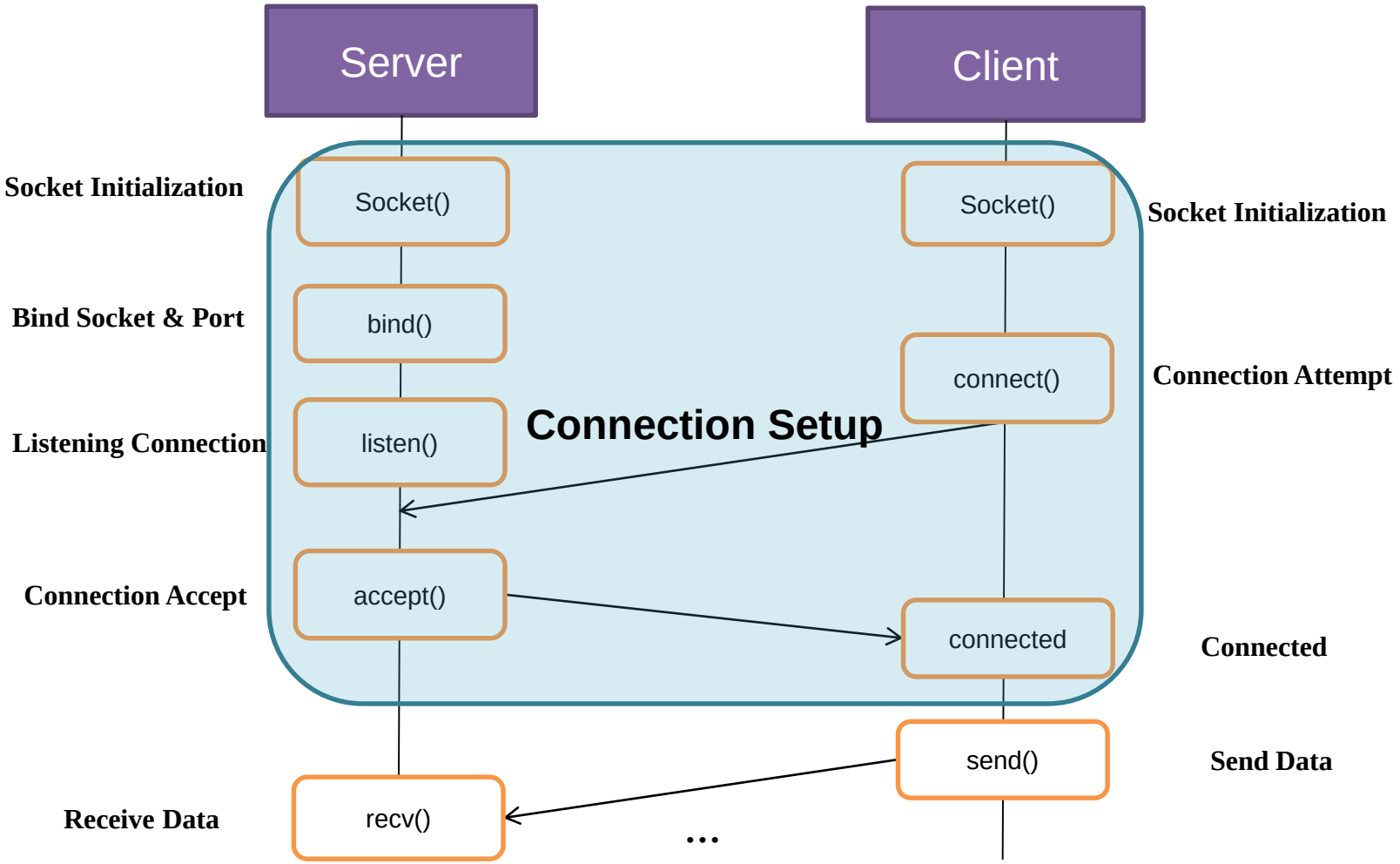
Greeting Example : Revisited

❖ Network Protocol for Greeting



Client-Server Programming

- TCP Client Server Example



Echo Client and Server (1/2)

- Server

```
import socket

HOST = '127.0.0.1'  # Standard loopback interface address (localhost)
PORT = 65432        # Port to listen on (non-privileged ports are > 1023)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data:
                break
            print("Received {} from client:{}".format(data, HOST))
            input("Press enter to proceed")
            conn.sendall(data)
```

```
Connected by ('127.0.0.1', 61538)
Received b'Hello, world' from client:127.0.0.1
Press enter to proceed
```

Echo Client and Server (2/2)

- Client

```
import socket

HOST = '127.0.0.1'  # The server's hostname or IP address
PORT = 65432        # The port used by the server

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    print("I am going to send Hello world!")
    input("Press enter to proceed")
    s.sendall(b'Hello, world')
    data = s.recv(1024)

print('Received', repr(data))
```

```
I am going to send Hello world!
Press enter to proceed
Received b'Hello, world'
```

Handling Multiple Connections

- Considerations for handling multiple connection
  - The role of the server is a router to relay messages
  - The server should know who sends the message and the destination of the message
  - The server should serve multiple request at the same time

- Use selectors[4]
  - Selectors allows high-level and efficient I/O multiplexing, built upon the select module primitives
    - » You may register socket to the selector
      - EVENT_READ: Available for read
      - EVENT_WRITE: Available for write
    - » When registered event occurs the selectors.select function returns the tuple, (key, mask)
      - key.fileobj denotes the File object registered(the socket)
      - Key.data denotes the Optional opaque data associated to this file object: for example, this could be used to store a per-client session ID(the packet data)
      - mask denotes the event that occurred (e.g. EVENT_READ, EVENT_WRITE)

Multiple Server (1/2)

```
import socket
import types
import selectors

sel = selectors.DefaultSelector()

def accept_wrapper(sock):
    conn, addr = sock.accept()  # Should be ready to read
    print('accepted connection from', addr)
    conn.setblocking(False)
    data = types.SimpleNamespace(addr=addr, inb=b'', outb=b'')
    events = selectors.EVENT_READ | selectors.EVENT_WRITE
    sel.register(conn, events, data=data)

def service_connection(key, mask):
    sock = key.fileobj
    data = key.data
    if mask & selectors.EVENT_READ:
        recv_data = sock.recv(1024)  # Should be ready to read
        if recv_data:
            data.outb += recv_data
        else:
            print('closing connection to', data.addr)
            sel.unregister(sock)
            sock.close()
    if mask & selectors.EVENT_WRITE:
        if data.outb:
            print('echoing', repr(data.outb), 'to', data.addr)
            sent = sock.send(data.outb)  # Should be ready to write
            data.outb = data.outb[sent:]
```

Multiple Server (2/2)

```
host = 'localhost'
port = 65432

lsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
lsock.bind((host, port))
lsock.listen()
print('listening on', (host, port))
lsock.setblocking(False)
sel.register(lsock, selectors.EVENT_READ, data=None)

try:
    while True:
        events = sel.select(timeout=None)
        for key, mask in events:
                if key.data is None:
                    accept_wrapper(key.fileobj)
                else:
                    service_connection(key, mask)
except KeyboardInterrupt:
    print("caught keyboard interrupt, exiting")
finally:
    sel.close()
```

Multiple Client (1/3)

```
import socket
import types
import selectors

sel = selectors.DefaultSelector()
messages = [b'Message 1 from client.', b'Message 2 from client.']

def start_connections(host, port, num_conns):
    server_addr = (host, port)
    for i in range(0, num_conns):
        connid = i + 1
        print('starting connection', connid, 'to', server_addr)
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.setblocking(False)
        sock.connect_ex(server_addr)
        events = selectors.EVENT_READ | selectors.EVENT_WRITE
        data = types.SimpleNamespace(connid=connid,
                                     msg_total=sum(len(m) for m in messages),
                                     recv_total=0,
                                     messages=list(messages),
                                     outb=b'')
        sel.register(sock, events, data=data)
```

Multiple Client (2/3)

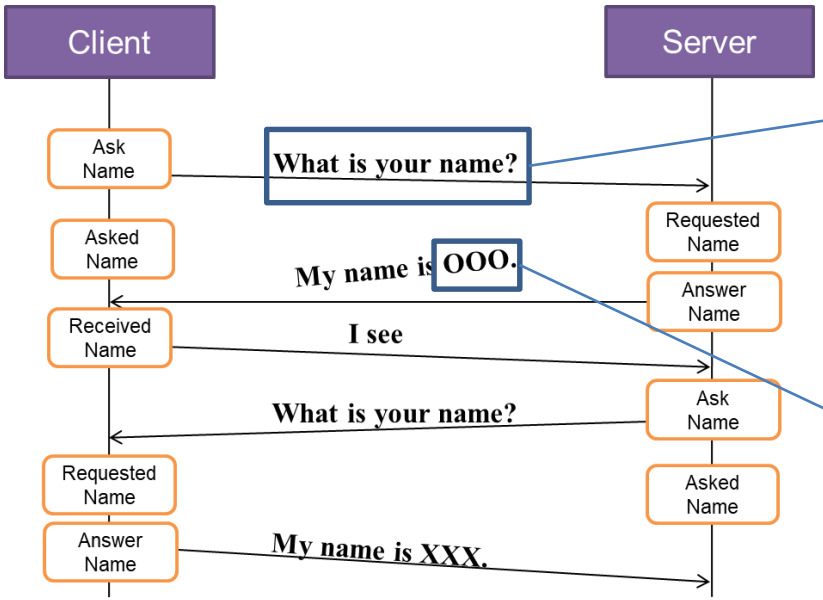
```
def service_connection(key, mask):
    sock = key.fileobj
    data = key.data
    if mask & selectors.EVENT_READ:
        recv_data = sock.recv(1024)  # Should be ready to read
        if recv_data:
            print('received', repr(recv_data), 'from connection', data.connid)
            data.recv_total += len(recv_data)
        if not recv_data or data.recv_total == data.msg_total:
            print('closing connection', data.connid)
            sel.unregister(sock)
            sock.close()
    if mask & selectors.EVENT_WRITE:
        if not data.outb and data.messages:
            data.outb = data.messages.pop(0)
        if data.outb:
            print('sending', repr(data.outb), 'to connection', data.connid)
            sent = sock.send(data.outb)  # Should be ready to write
            data.outb = data.outb[sent:]
```

Multiple Client (3/3)

```
host = 'localhost'
port = 65432
start_connections(host, port, 2)

while True:
    events = sel.select(timeout=1)
    if events != None:
        for key, mask in events:
            service_connection(key, mask)

    if not sel.get_map():
            break

sel.close()
```

Defining Protocol

- Definition of the Protocol [5]
  - a defined set of rules and regulations that determine how data is transmitted in telecommunications and computer networking
  - Core elements of the communication protocol
    - » Data representation
    - » Data handling



Tell Server what to do

What function should be invoked by the server?

Put information to Packet Header

Send data to Client

What data should be sent to the client?

Put information to Packet Payload

Defining Protocol

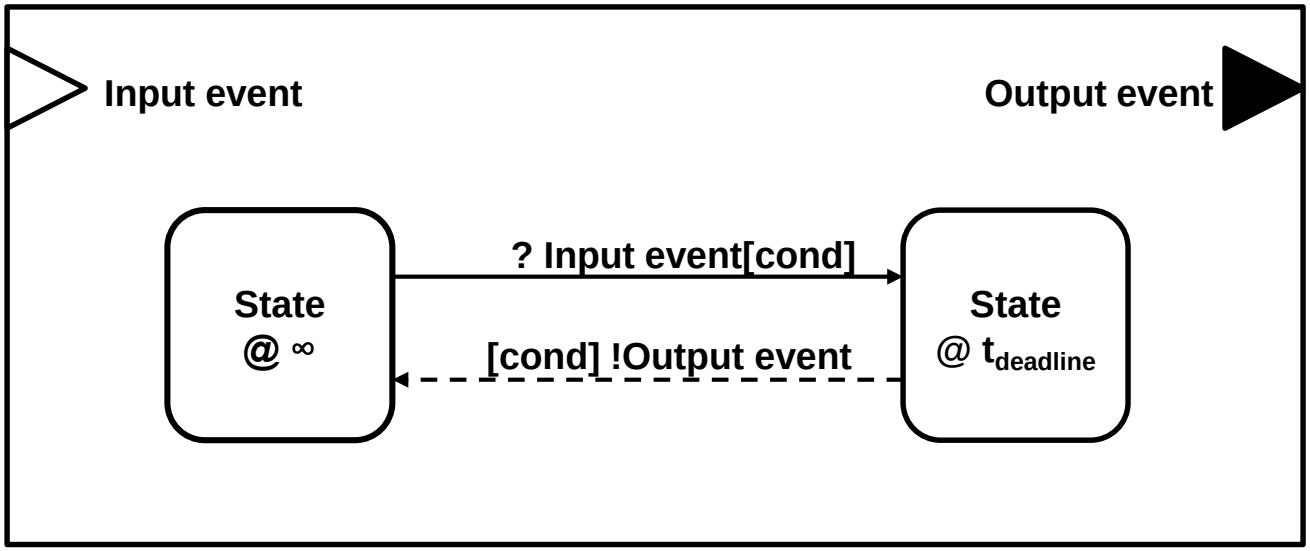
- In the network communication, the data is a stream, so that each computer should interpret data from the data stream

- In general,
  - Packet has fixed-length header
    - » Type: denotes what to do
    - » Body length: denotes the size of packet
  - Payload has variable-length
    - » The size of the payload may be different based on the application

- Handling Protocol may be represented by Discrete Event System Diagram
  - Describe a system based on the external event, internal event

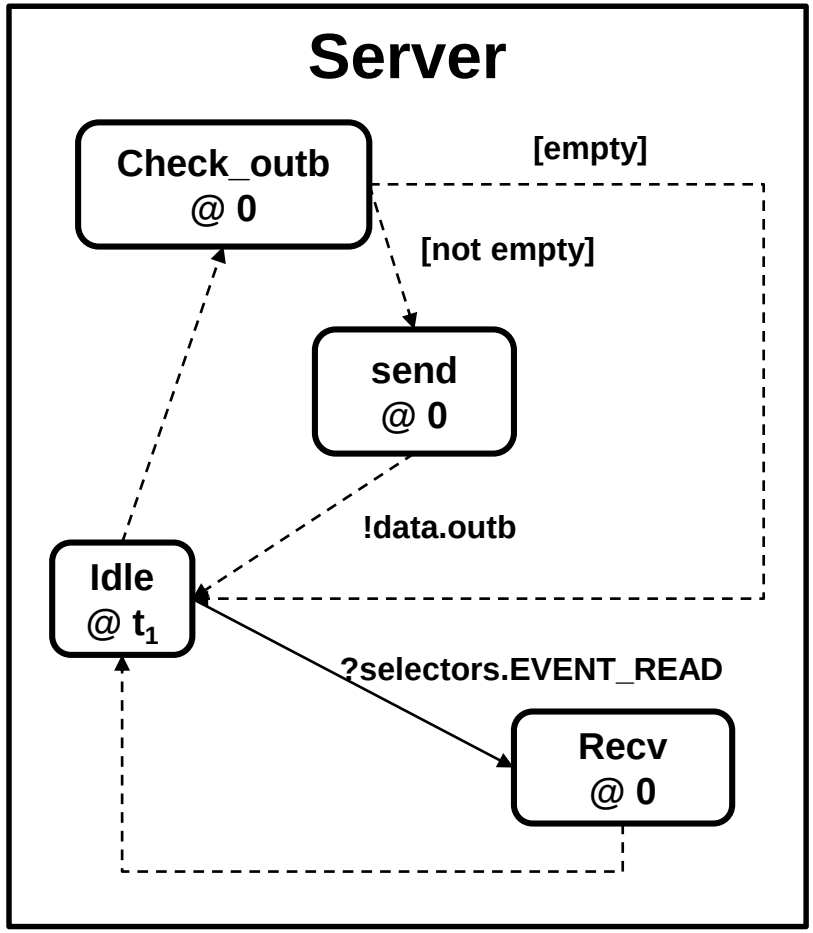
Defining Protocol

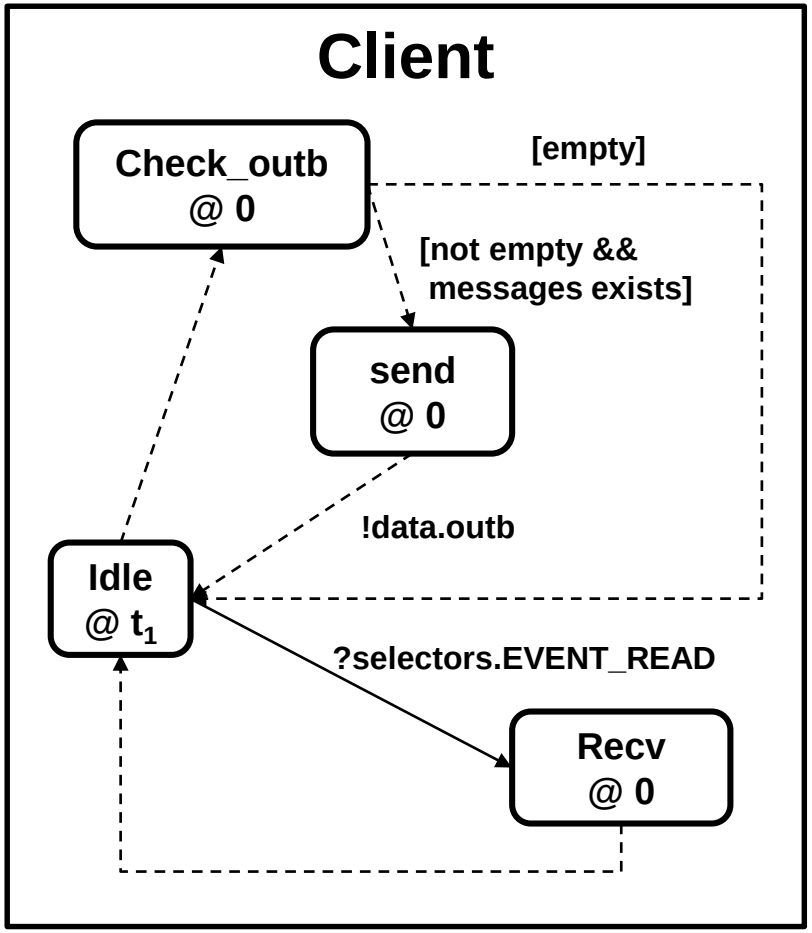
- DEVS Diagram (Atomic Model)
  - “?”: Denotes an input event
  - "!”: Denotes an output event
  - @ : Denotes the deadline of the given state
    - » If the deadline is ∞ then the model must stay for infinite time
    - » If the deadline is 0 then the model should move to the next state
  - **[] : Denotes the condition**



Defining Protocol

- Example





Telegram[6] & Telegram Bot API[7]

- Telegram
  - Telegram is a messaging app with a focus on speed and security, it’s super-fast, simple and free

- Telegram Bot
  - Telegram Bots are special accounts that do not require an additional phone number to set up
  - These accounts serve as an interface for code running somewhere on your server.
  - **How do bots work**
    - » Send messages and commands to bots by opening a chat with them or by adding them to groups.
    - » Send requests directly from the input field by typing the bot's @username and a query
  - **How do I create a bot?**
    - » Use BotFather
    - » BotFather creates an authorization token

Creating a new bot (1/2)

- Use the **/newbot** command to create a new bot
  - The BotFather will ask you for a name and username, then generate an authorization token for your new bot.
    - » **name** of your bot is displayed in contact details and elsewhere.
    - » **Username** is a short name, to be used in mentions and telegram.me links. Usernames are 5-32 characters long and are case insensitive, but may only include Latin characters, numbers, and underscores. Your bot's username **must** end in ‘bot’, e.g. ‘tetris_bot’ or ‘TetrisBot’.
    - » **token** is a string along the lines something like “110201543:AAHdqTcvCH1vGWJxfSeofSAs0K5PALDsaw” that is required to authorize the bot and send requests to the Bot API. Keep your token secure and store it safely, it can be used by anyone to control your bot.
      - use the /token command to generate a new one
  - Your application is a bot, and you may communicate with other people using the authorization token

Creating a new bot (2/2)

CC **Changbeom** 11:38:44 PM
/start

BO **BotFather** 11:38:44 PM
I can help you create and manage Telegram bots. If you're new to the Bot API, please see the manual.

You can control me by sending these commands:

/newbot - create a new bot
/mybots - edit your bots **[beta]**

Edit Bots
/setname - change a bot's name
/setdescription - change bot description
/setabouttext - change bot about info
/setuserpic - change bot profile photo
/setcommands - change the list of commands
/deletebot - delete a bot

Bot Settings
/token - generate authorization token
/revoke - revoke bot access token
/setinline - toggle inline mode
/setinlinegeo - toggle inline location requests
/setinlinefeedback - change inline feedback settings
/setjoingroups - can your bot be added to groups?
/setprivacy - toggle privacy mode in groups

Games
/mygames - edit your games **[beta]**
/newgame - create a new game
/listgames - get a list of your games
/editgame - edit a game

CC **Changbeom** 11:38:55 PM
/newbot

BO **BotFather** 11:38:56 PM
Alright, a new bot. How are we going to call it? Please choose a name for your bot.

CC **Changbeom** 11:39:06 PM
eb1

BO **BotFather** 11:39:07 PM
Good. Now let's choose a username for your bot. It must end in `bot`. Like this, for example: TetrisBot or tetris_bot.

CC **Changbeom** 11:39:17 PM
eb1Bot

BO **BotFather** 11:39:17 PM
Sorry, this username is already taken. Please try something different.

CC **Changbeom** 11:39:25 PM
cbchoiBot

BO **BotFather** 11:39:26 PM
Done! Congratulations on your new bot. You will find it at t.me/cbchoiBot. You can now add a description, about section and profile picture for your bot, see /help for a list of commands. By the way, when you've finished creating your cool bot, ping our Bot Support if you want a better username for it. Just make sure the bot is fully operational before you do this.

Use this token to access the HTTP API:

Developing Telegram Bot using Python

- python-telegram-bot
  - A wrapper for telegram bot API

- Installation
  - pip install python-telegram-bot --upgrade

- Example
  - import telegram
  - bot = telegram.Bot(token='TOKEN') # put authorization token here
  - print(bot.get_me())

Reference

[1] Socket Programming in Python (Guide), https://realpython.com/python-sockets/

[2] Server (computing), https://en.wikipedia.org/wiki/Server_(computing)

[3] Client (computing), https://en.wikipedia.org/wiki/Client

[4] selectors, https://docs.python.org/3/library/selectors.html

[5] Protocol, https://en.wikipedia.org/wiki/Communication_protocol

[6] Telegram, https://telegram.org/

[7] Telegram Bot API, https://core.telegram.org/api