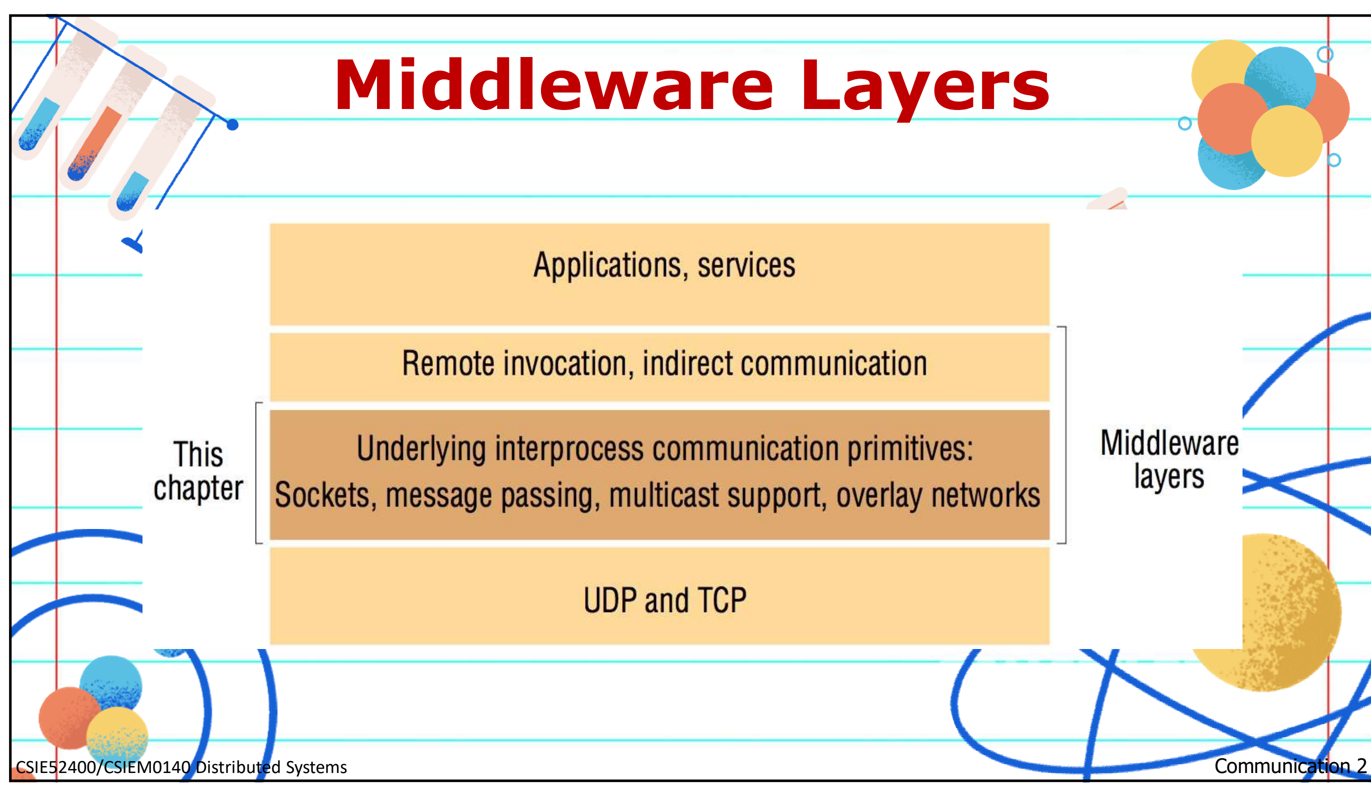
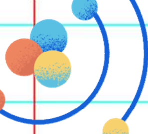
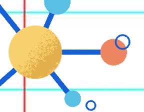
CSIE52400/CSIEM0140
Distributed Systems
Lecture 06
Communication
Shiow-yang Wu (吳秀陽)
Department of Computer Science and Information Engineering
National Dong Hwa University
CSIE52400/CSIEM0140 Distributed Systems
1


Middleware Layers
Applications, services
Remote invocation, indirect communication
Underlying interprocess communication primitives:
Sockets, message passing, multicast support, overlay networks
UDP and TCP
This chapter
Middleware layers
CSIE52400/CSIEM0140 Distributed Systems
Communication 2

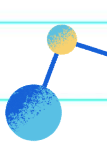


Message Synchronization

- **Asynchronous Communication**
  - Non-blocking send: sender process released after message copied into sender's kernel
- **Synchronous Communication**
  - Blocking send: sender process released after message transmitted to network
  - Reliable blocking send: sender process released after messages received by receiver's kernel
  - Explicit blocking send: sender process released after message received by receiver process
  - Request and Reply: sender process released after message been processed by receiver and response returned to sender



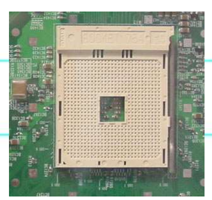
Message Destinations

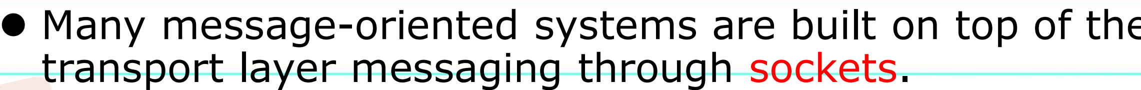


- **IP (address, port)**
  - A port has exactly one receiver but can have many senders.
  - Processes may use multiple ports to receive messages.
- To provide **location transparency**
  - refer to services by name and use a name server or binder for name to server location translation
  - the OS provides location independent identifiers and handles the identifiers to lower level address mapping

Sockets and Ports

- Various sockets… Any similarity?

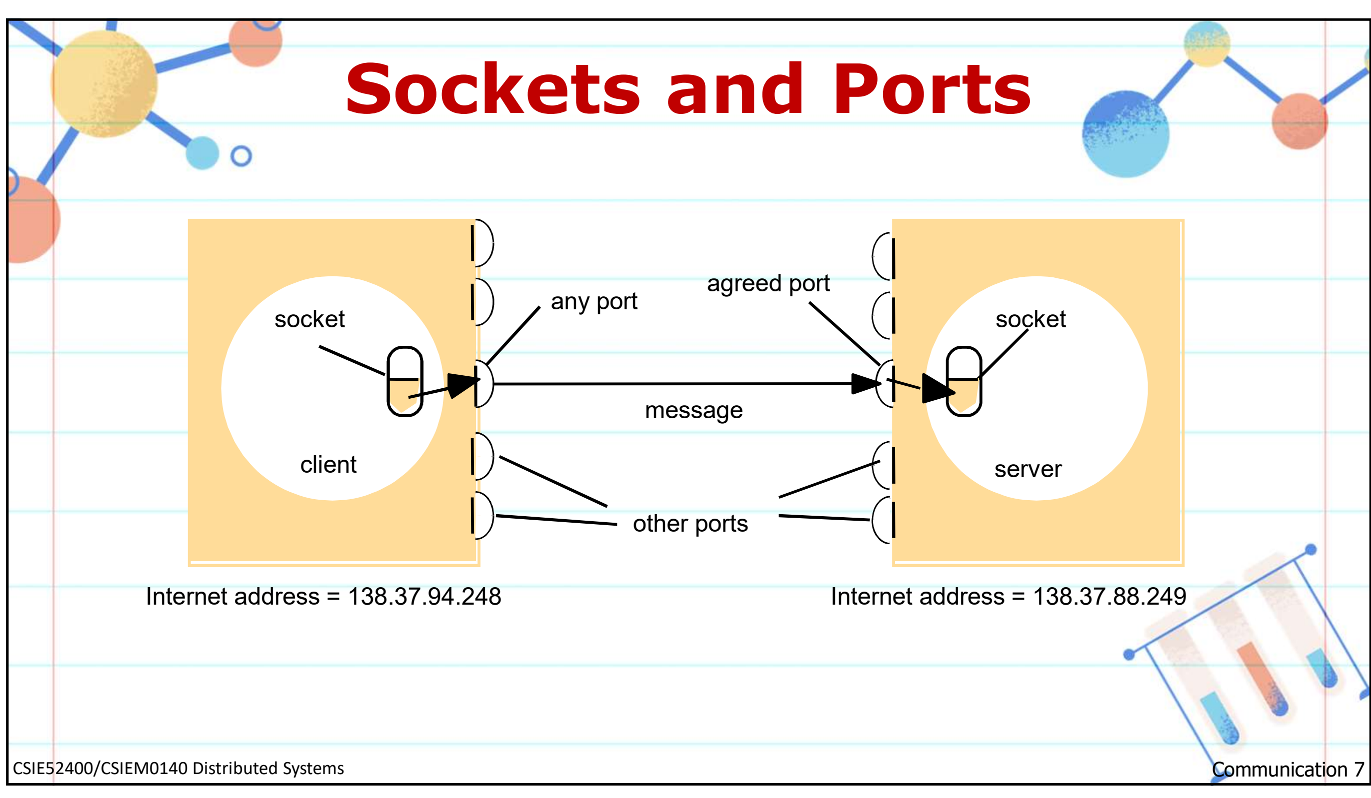


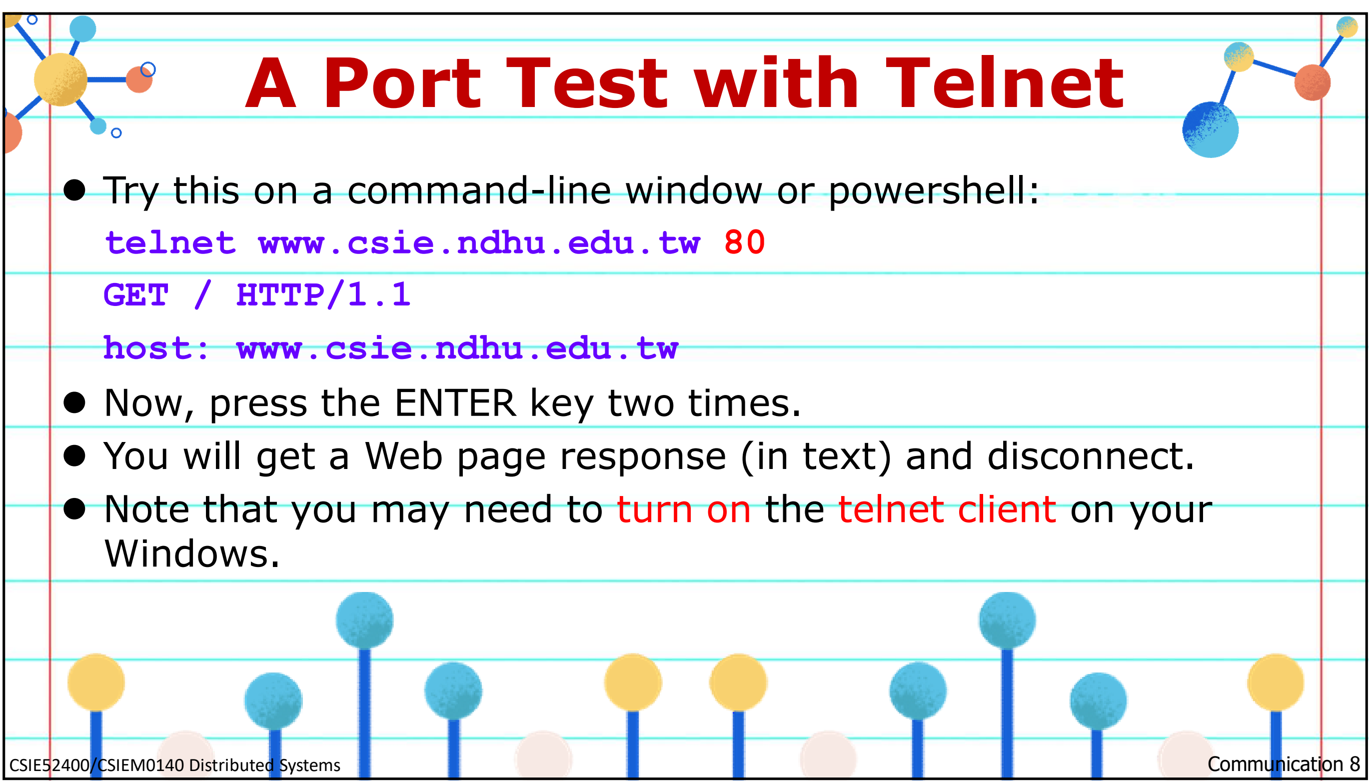


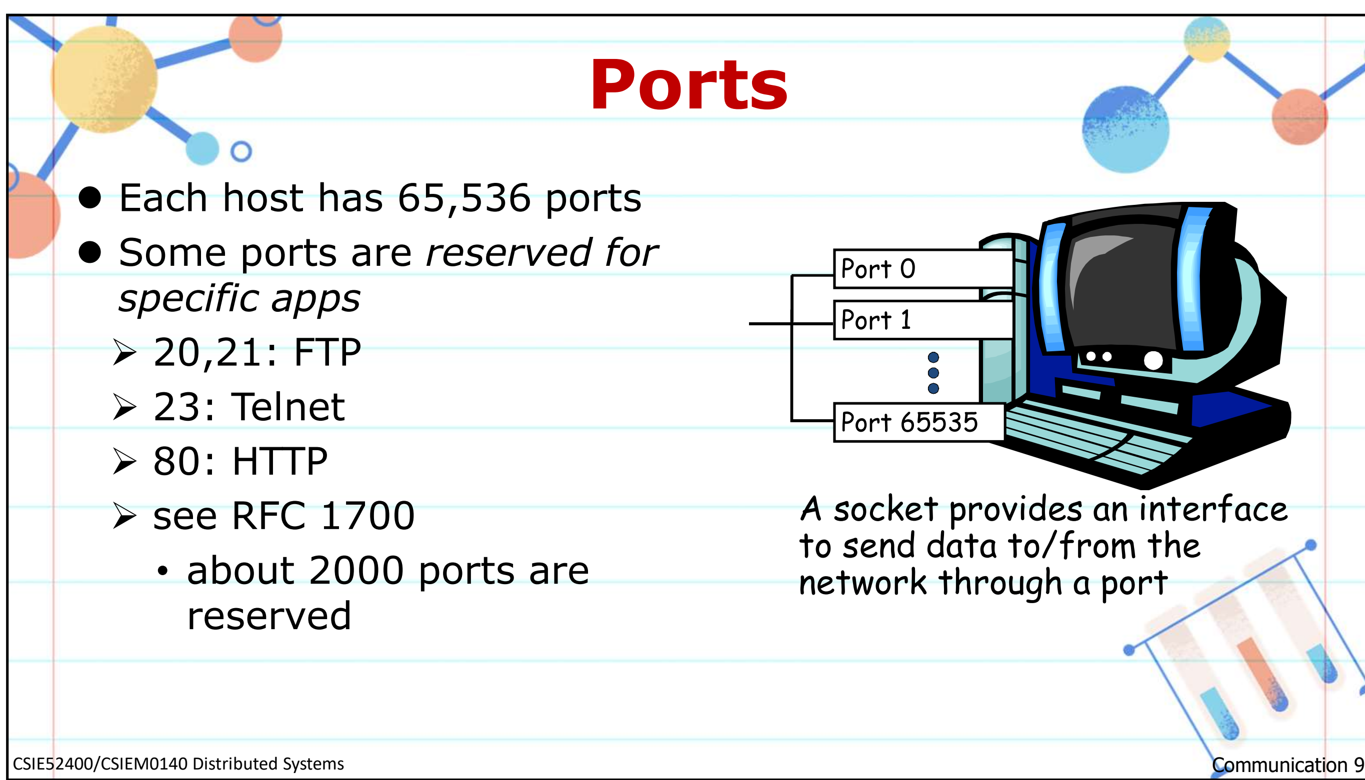
- Many message-oriented systems are built on top of the transport layer messaging through sockets.
- Socket abstraction provides an endpoint for communication between processes.
- Messages are transmitted between a socket in one process and a socket in another.

Sockets and Ports

- A socket is an endpoint of a connection
  - Identified by IP address and Port number
- To receive messages, a socket must be bound to a local port and IP address.
- Messages sent to a (address, port) can be received only by a process whose socket is associated with that (address, port).
- A process can not share ports with others.
- May use the same socket for sending and receiving.
- Each socket is associated with a protocol.


Sockets and Ports
socket
client
any port
agreed port
message
other ports
socket
server
Internet address = 138.37.94.248
Internet address = 138.37.88.249
CSIE52400/CSIEM0140 Distributed Systems
Communication 7

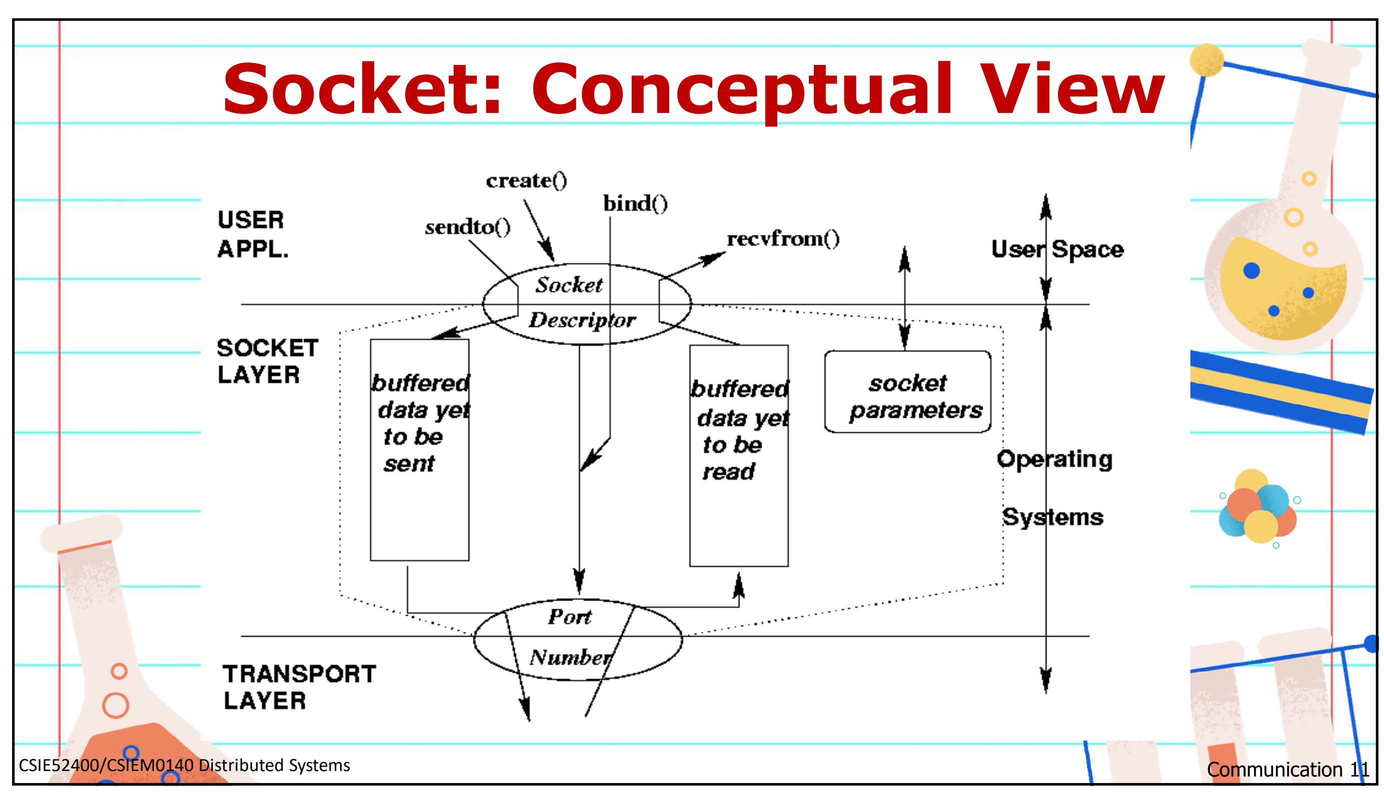

A Port Test with Telnet
Try this on a command-line window or powershell:
telnet www.csie.ndhu.edu.tw 80
GET / HTTP/1.1
host: www.csie.ndhu.edu.tw
Now, press the ENTER key two times.
You will get a Web page response (in text) and disconnect.
Note that you may need to turn on the telnet client on your Windows.
CSIE52400/CSIEM0140 Distributed Systems
Communication 8

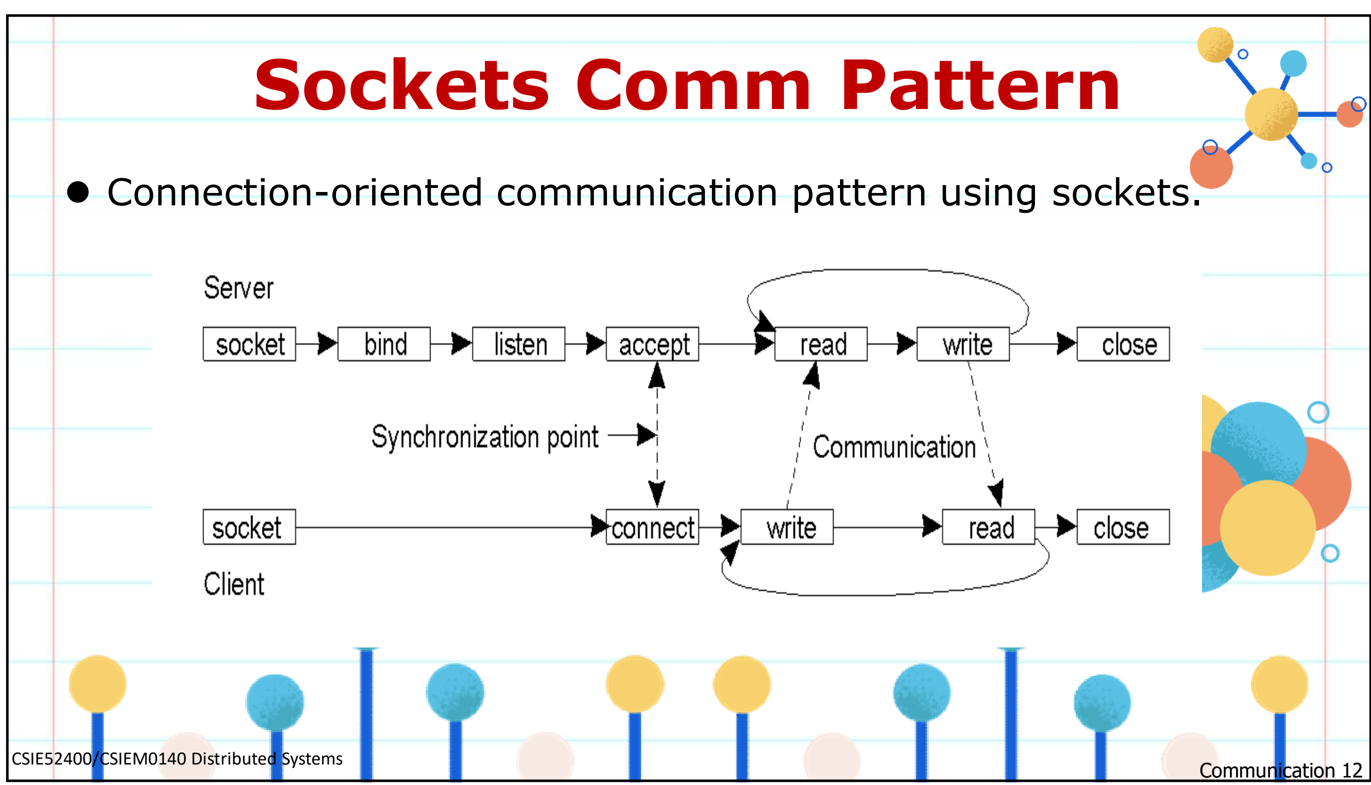


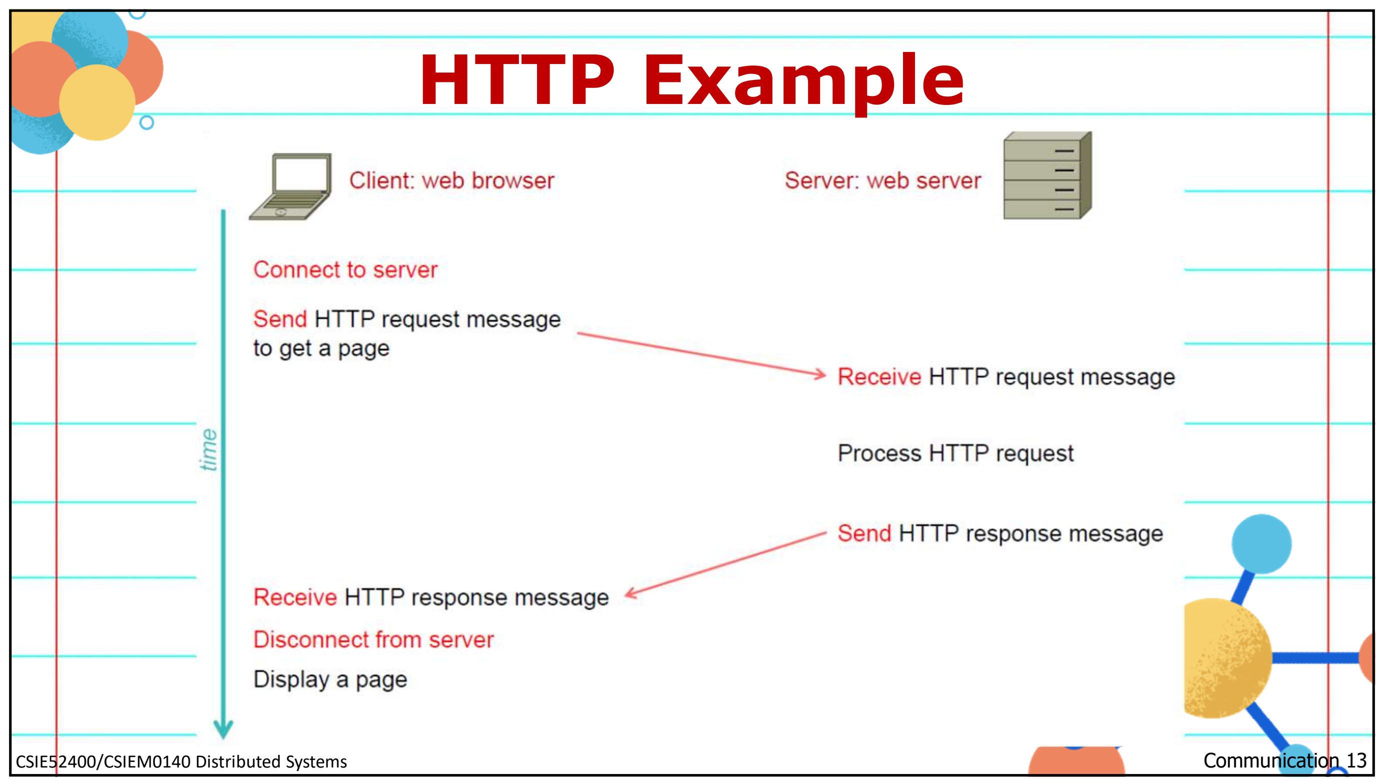
Socket Primitives

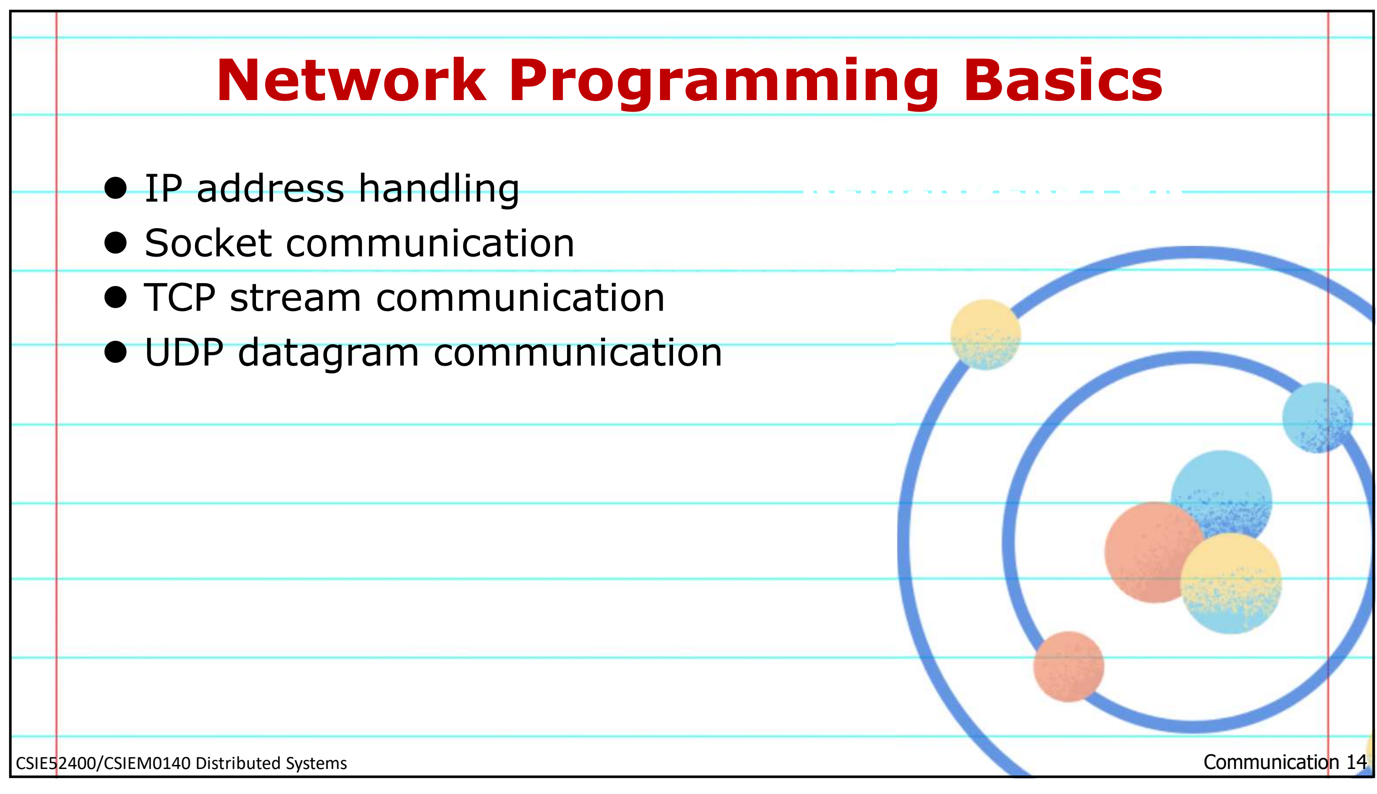
- Socket primitives for TCP/IP.

Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection


Socket: Conceptual View
USER APPL.
SOCKET LAYER
TRANSPORT LAYER
create()
bind()
sendto()
recvfrom()
Socket Descriptor
buffered data yet to be sent
buffered data yet to be read
socket parameters
Port Number
User Space
Operating Systems
CSIE52400/CSIEM0140 Distributed Systems
Communication 11


Sockets Comm Pattern
● Connection-oriented communication pattern using sockets.
Server
socket
bind
listen
accept
read
write
close
Synchronization point
Communication
socket
connect
write
read
close
Client
CSIE52400/CSIEM0140 Distributed Systems
Communication 12


HTTP Example
Client: web browser
Server: web server
time
Connect to server
Send HTTP request message to get a page
Receive HTTP request message
Process HTTP request
Send HTTP response message
Receive HTTP response message
Disconnect from server
Display a page
CSIE52400/CSIEM0140 Distributed Systems
Communication 13


Network Programming Basics
IP address handling
Socket communication
TCP stream communication
UDP datagram communication
CSIE52400/CSIEM0140 Distributed Systems
Communication 14

Python IP and DNS Lookup

- Get IP address from domain name:

```
>>> import socket
>>> addr = socket.gethostbyname('www.ndhu.edu.tw')
>>> print(addr)
134.208.11.217
```

- Get domain name from IP address:

```
>>> name = socket.gethostbyaddr(addr)
>>> print(name)
('134-208-11-217.ndhu.edu.tw', [], ['134.208.11.217'])
```


Python IP Handling

- The ipaddress module is for IP addresses, networks and interfaces handling.
- Handle both IPv4 and IPv6 addresses

```
>>> ipaddress.ip_address('192.168.0.1')
IPv4Address('192.168.0.1')
>>> ipaddress.ip_address('2001:db8::')
IPv6Address('2001:db8::')
```

- Read the doc for other related objects.

Java API for IP Addresses

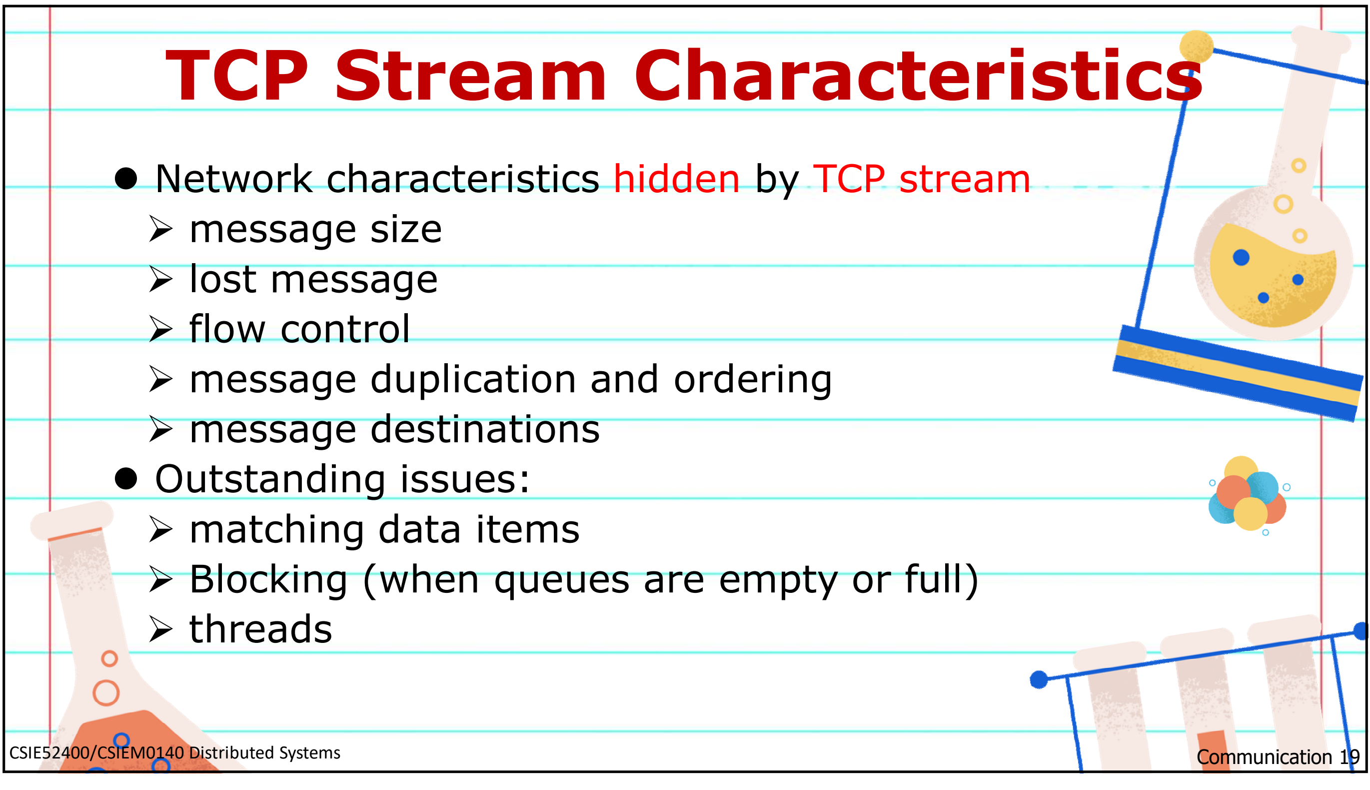
- Java provides the InetAddress class to represent Internet addresses.

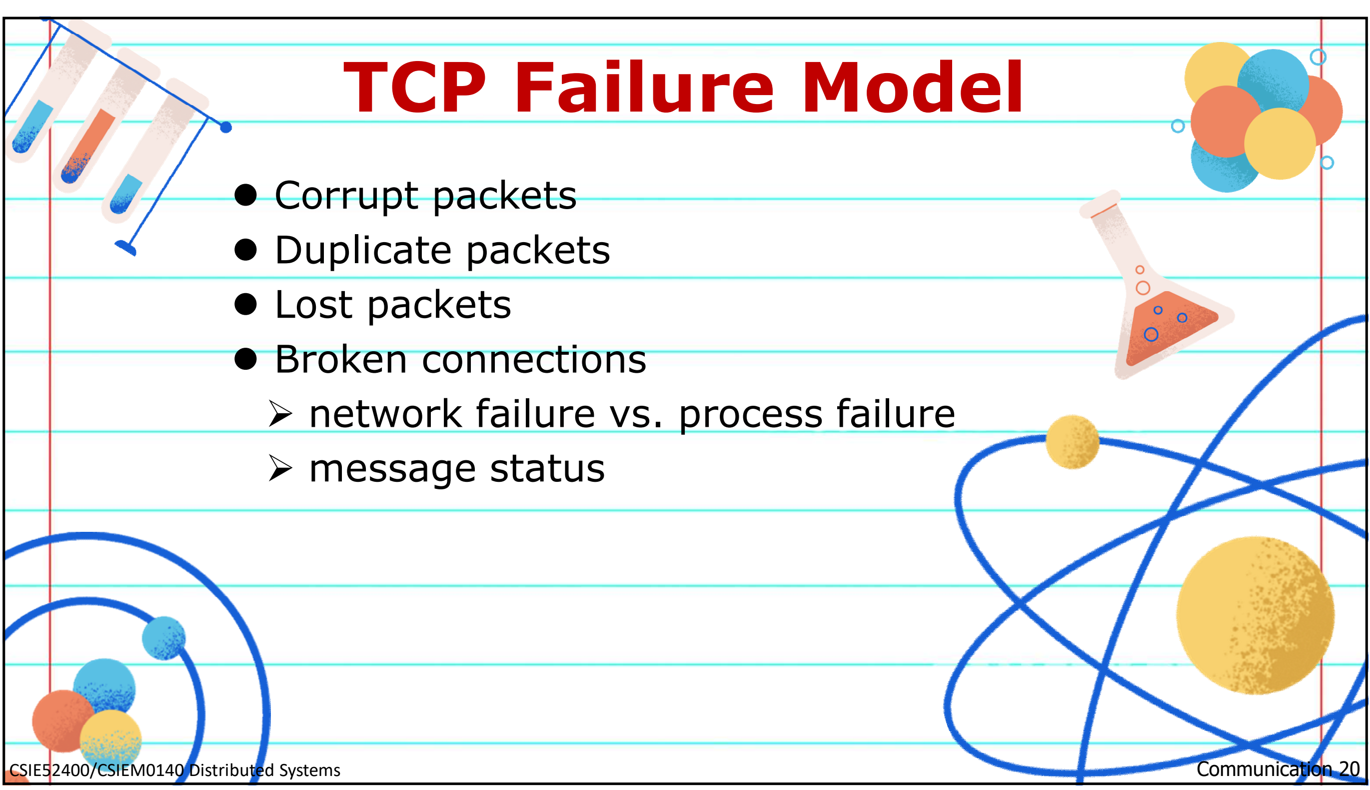
```
InetAddress aHost =
  InetAddress.getByName("www.csie.ndhu.edu.tw");
```

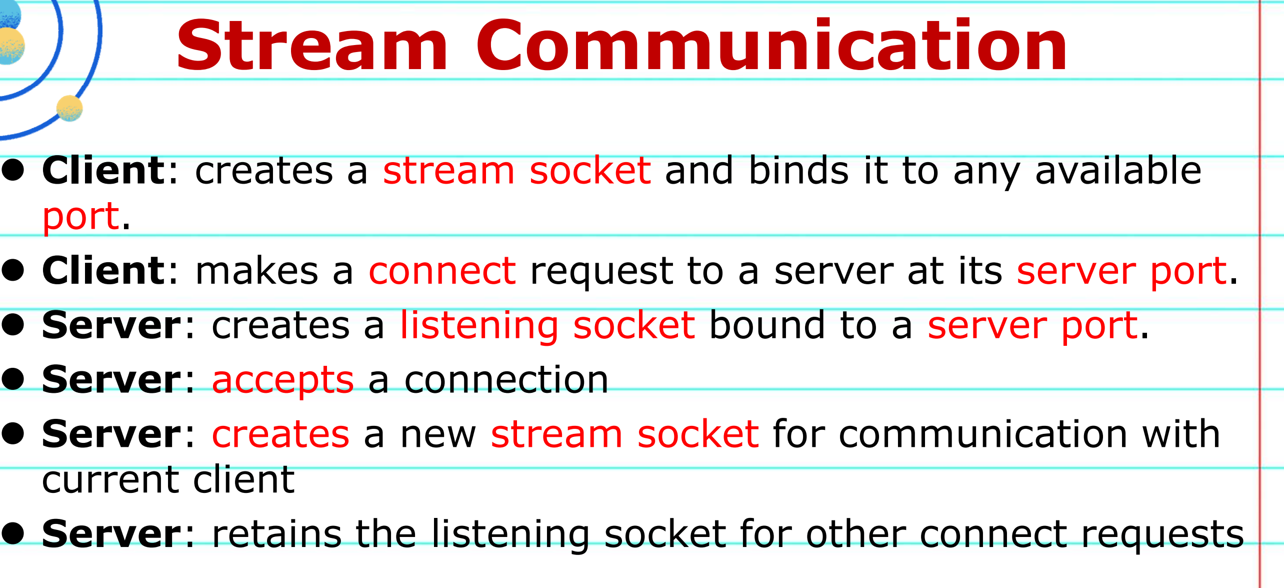
- The method can throw an UnknownHostException.
- The class allows us to access Internet hosts by their DNS names instead of numeric IP address.
- We will discuss Java network programming along the way when Java is the selected language for the semester.

TCP Stream Communication

- Connection-oriented communication
- Need to set up a connection between client and server first.
- After setting up the connection, the two processes could be peers.
- Stream communication procedure:
  - Client sends a connect request to server
  - Server sends a accept request to client
  - Establish a stream between client and server


TCP Stream Characteristics
● Network characteristics hidden by TCP stream
➢ message size
➢ lost message
➢ flow control
➢ message duplication and ordering
➢ message destinations
● Outstanding issues:
➢ matching data items
➢ Blocking (when queues are empty or full)
➢ threads
CSIE52400/CSIEM0140 Distributed Systems
Communication 19


TCP Failure Model
● Corrupt packets
● Duplicate packets
● Lost packets
● Broken connections
➢ network failure vs. process failure
➢ message status
CSIE52400/CSIEM0140 Distributed Systems
Communication 20



Python TCP Client-Server Flow

Server
socket
bind
listen
accept
recv
send
recv
close

Server creating listening socket

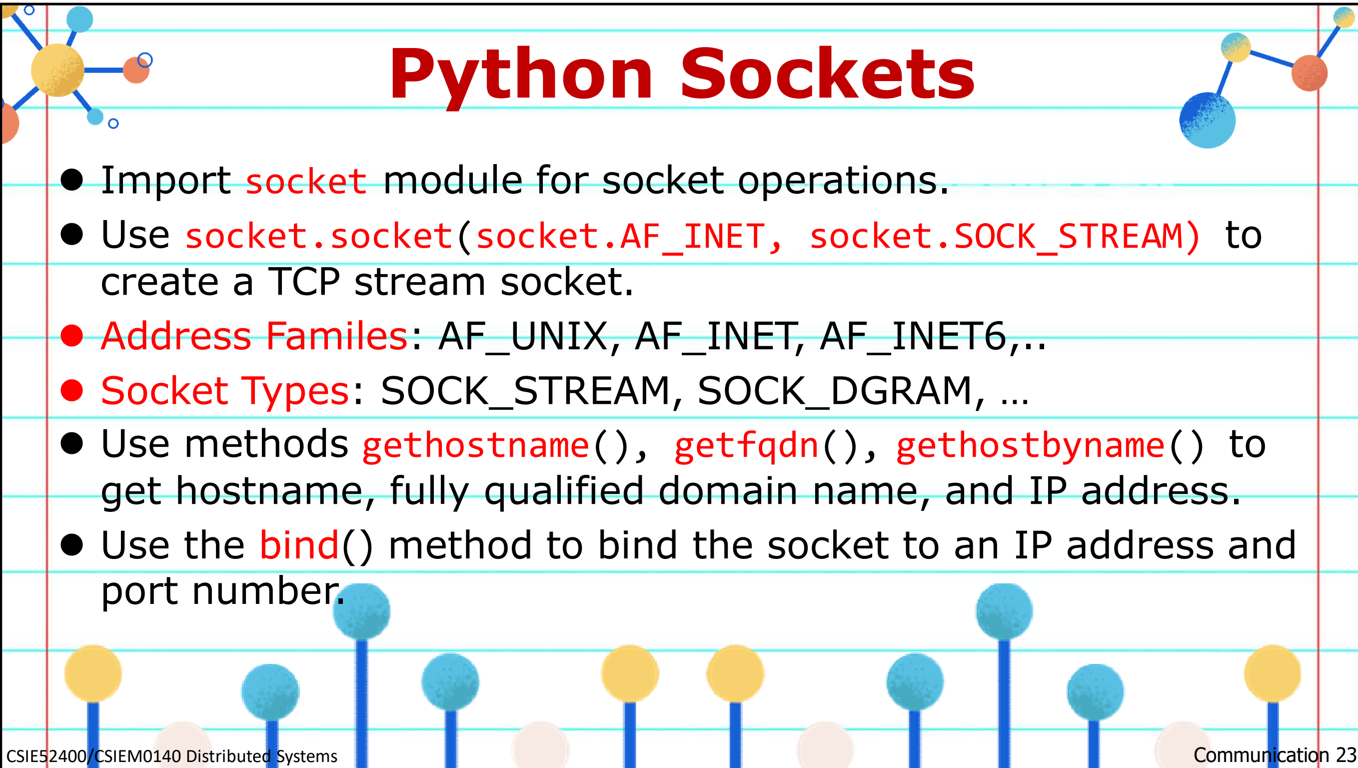
Client
socket
connect
send
recv
close

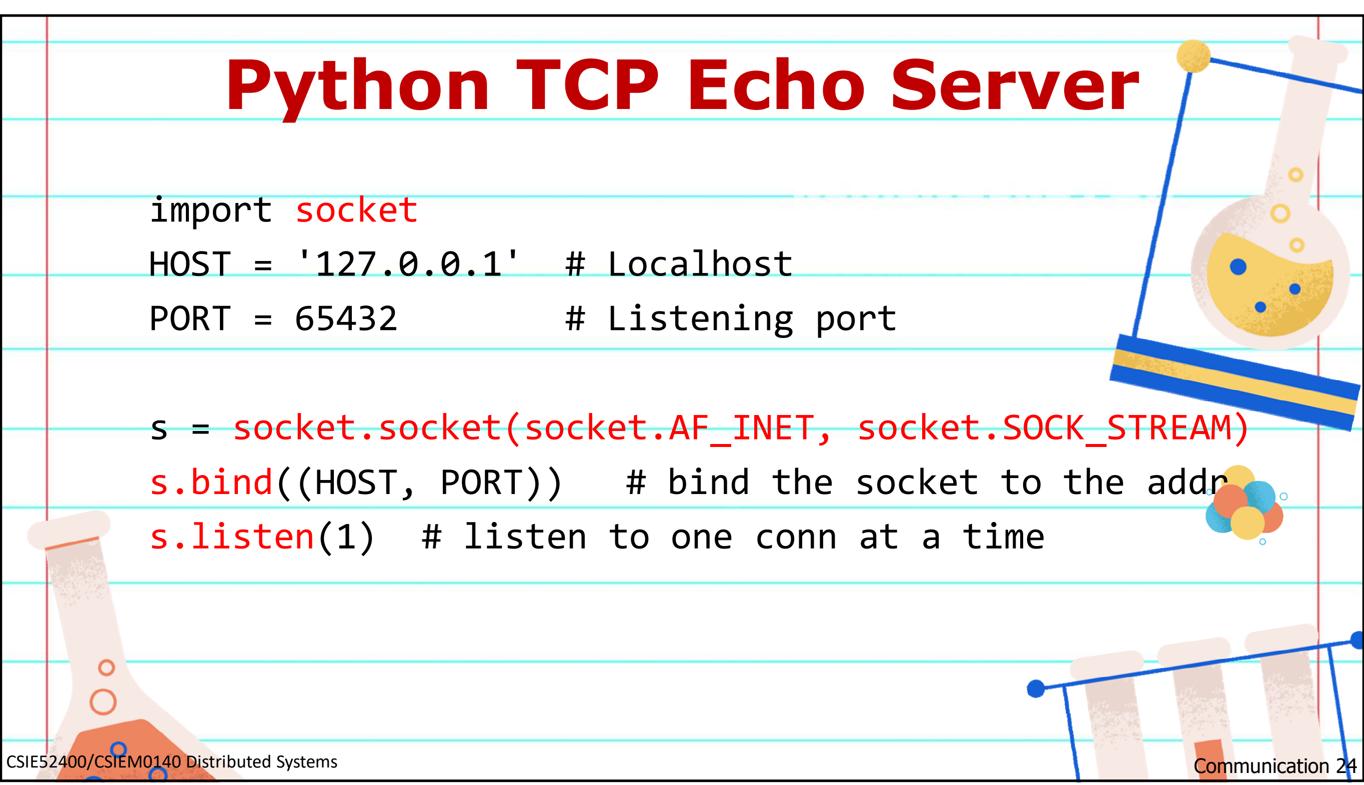
Establishing connection, three-way handshake

Client sending data, server receiving data

Server sending data, client receiving data

Client sending close message


Python Sockets
Import socket module for socket operations.
Use socket.socket(socket.AF_INET, socket.SOCK_STREAM) to create a TCP stream socket.
Address Familes: AF_UNIX, AF_INET, AF_INET6,..
Socket Types: SOCK_STREAM, SOCK_DGRAM, ...
Use methods gethostname(), getfqdn(), gethostbyname() to get hostname, fully qualified domain name, and IP address.
Use the bind() method to bind the socket to an IP address and port number.
CSIE52400/CSIEM0140 Distributed Systems
Communication 23

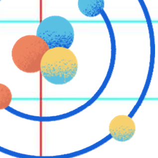

Python TCP Echo Server
import socket
HOST = '127.0.0.1' # Localhost
PORT = 65432   # Listening port
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))   # bind the socket to the addr
s.listen(1) # listen to one conn at a time
CSIE52400/CSIEM0140 Distributed Systems
Communication 24

Python TCP Echo Server

```
while True:
    print('waiting for a connection')
    conn, addr = s.accept()
    try:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data:
                break
            conn.sendall(data)  # echo
    finally:
        # Clean up the connection
        conn.close()
```


Python TCP Client

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

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
data = ["Mon", "Tue", "Wed", "Thu", "Fri"] # Data to send
for d in data:
    s.sendall(d.encode("utf-8"))    # encode before send
    print("Sent: ", d)
    time.sleep(1)
    response = s.recv(1024).decode("utf-8") # decode resp
    print('Received: ', response)
```



Socket Utility Functions

- `htonl(i)`, `htons(i)`
  - 32-bit or 16-bit integer to network format
- `ntohl(i)`, `ntohs(i)`
  - 32-bit or 16-bit integer to host format
- `inet_aton(ipstr)`, `inet_ntoa(packed)`
  - Convert addresses between regular strings and 4-byte packed strings

Timeout Capabilities

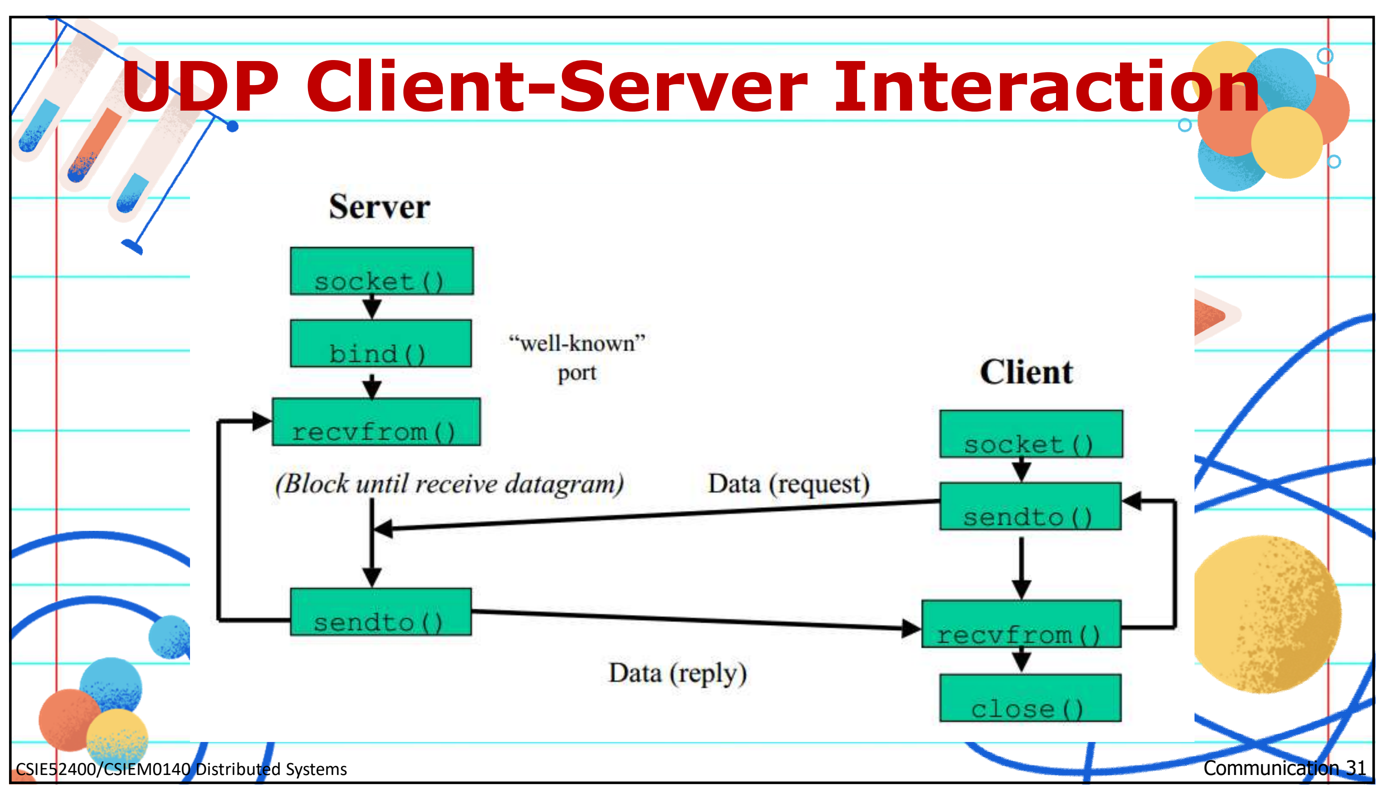
- Can set a default for all sockets
  - `socket.setdefaulttimeout(seconds)`
  - Argument is float # of seconds
  - Or `None` (indicates no timeout)
- Can set a timeout on an existing socket `s`
  - `s.settimeout(seconds)`

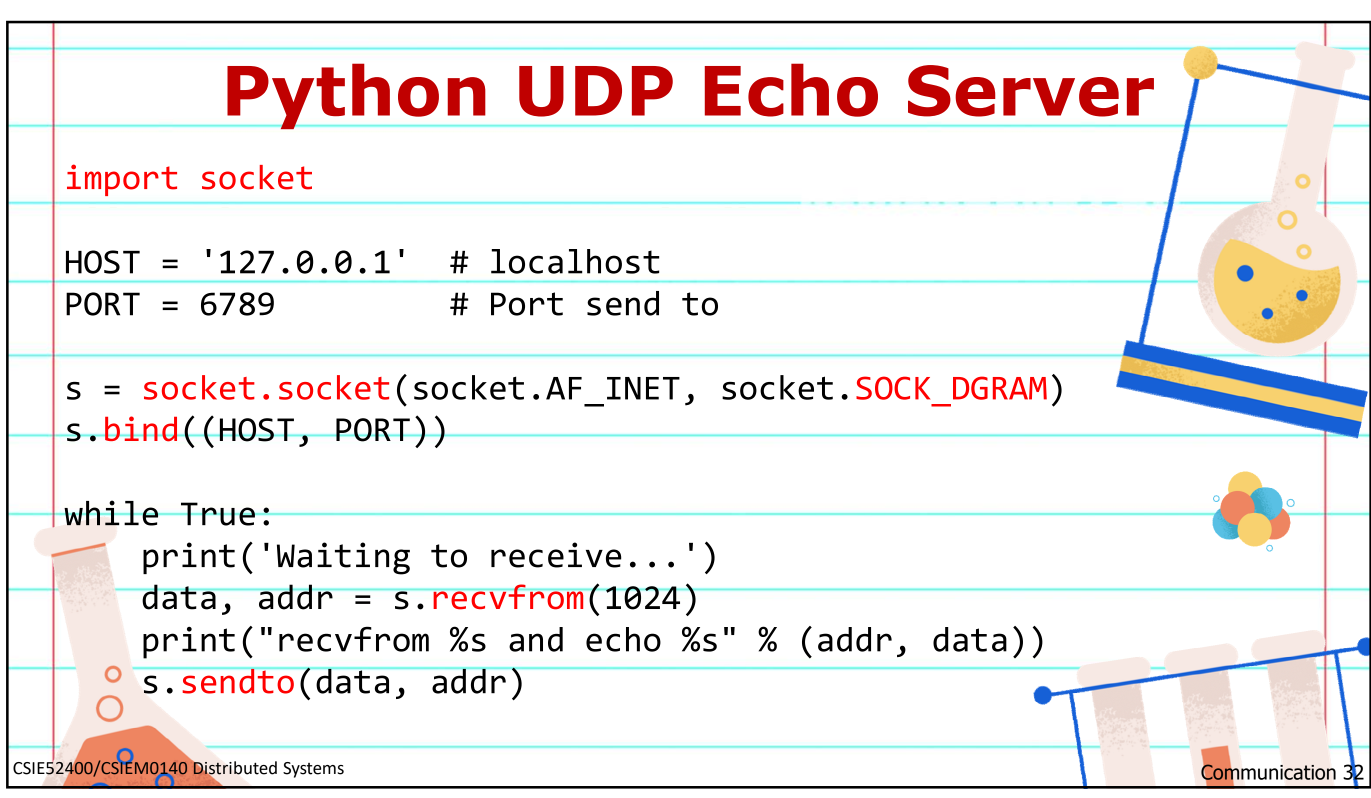
UDP Datagram Comm

- No acknowledgement or retries (best effort)
- **Message buffering and size**
  - must use buffer (array of bytes) to receive
  - if the message is too big, it is truncated
  - IP allows 2^{16} bytes message length
  - 8K is the most commonly used message size
- **Blocking** - non-blocking sends, blocking receives
- **Timeouts** - can be set on sockets
- **Receive from any**
  - a receive gets a message addressed to its socket from any origin
  - the IP address and port can be checked

UDP Failure Model

- **Omission failures**
  - use checksum to detect corrupted message
  - both send-omission and receive-omission are treated as omission failures in the channel
- **Ordering**
  - message can be delivered out of order
- Applications using UDP must provide their own checks for reliable communication.
- This can be achieved by using acknowledgements.


UDP Client-Server Interaction
Server
socket()
bind()
"well-known" port
recvfrom()
(Block until receive datagram)
Data (request)
Client
socket()
sendto()
sendto()
recvfrom()
Data (reply)
close()
CSIE52400/CSIEM0140 Distributed Systems
Communication 31


Python UDP Echo Server
import socket

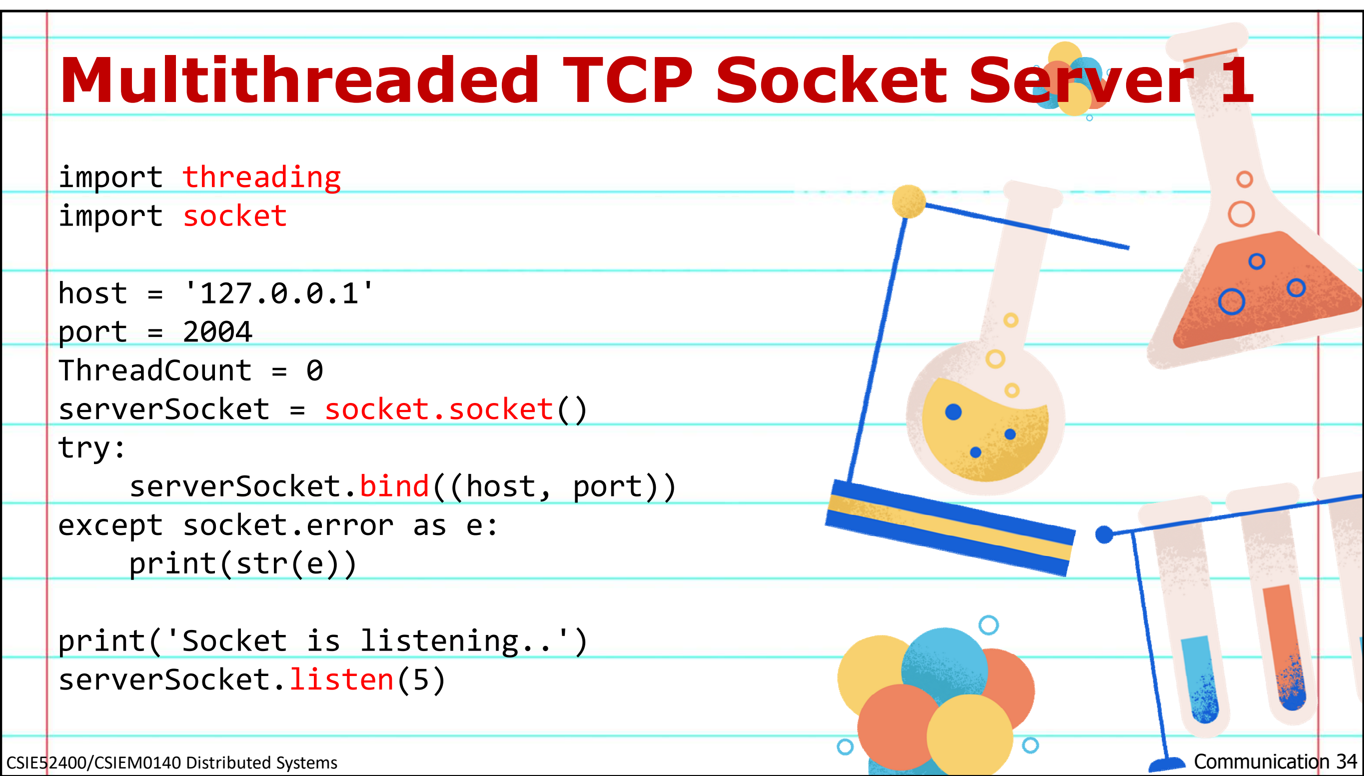
HOST = '127.0.0.1'   # localhost
PORT = 6789   # Port send to

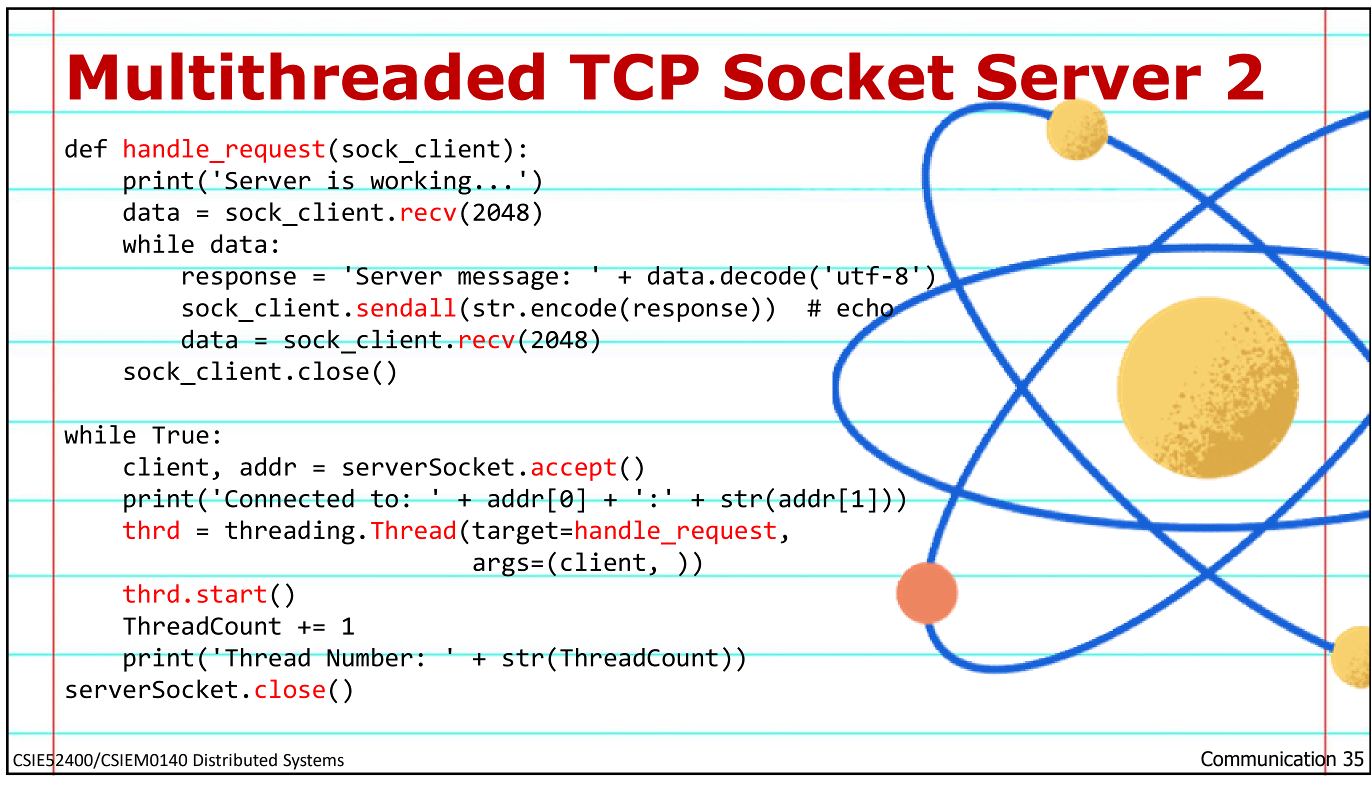
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind((HOST, PORT))

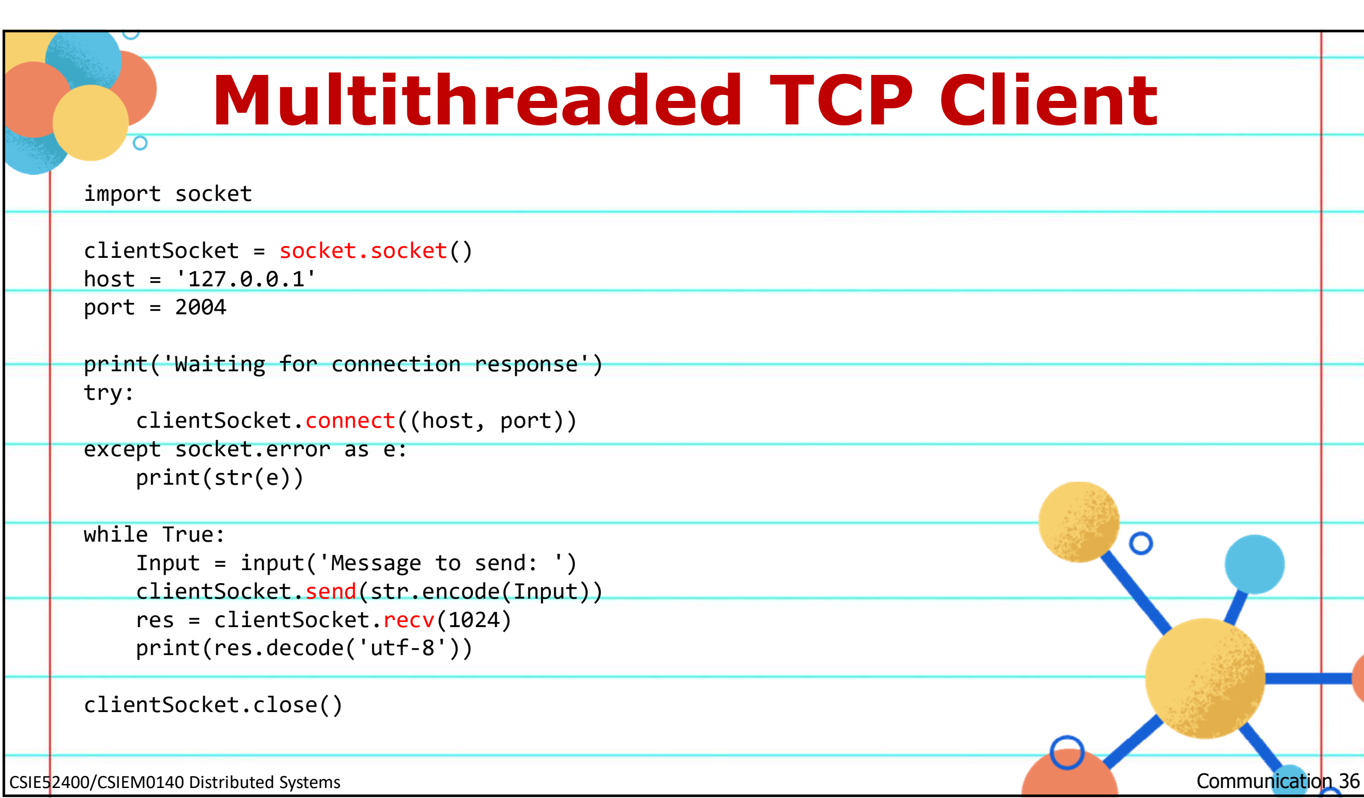
while True:
    print('Waiting to receive...')
    data, addr = s.recvfrom(1024)
    print("recvfrom %s and echo %s" % (addr, data))
    s.sendto(data, addr)
CSIE52400/CSIEM0140 Distributed Systems
Communication 32

Python UDP Client

```
import socket
ADDRESS = "127.0.0.1"
PORT = 6789

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
data = ["Mon", "Tue", "Wed", "Thu", "Fri"] # Data to send
for d in data:
    s.sendto(d.encode(), (ADDRESS, PORT))  # encode first
    print("Send: ", d)
    response, addr = s.recvfrom(1024)
    print("Receive %s from %s" % (response.decode(), addr))
```


Multithreaded TCP Socket Server 2
def handle_request(sock_client):
    print('Server is working...')
    data = sock_client.recv(2048)
    while data:
        response = 'Server message: ' + data.decode('utf-8')
        sock_client.sendall(str.encode(response)) # echo
        data = sock_client.recv(2048)
    sock_client.close()
while True:
    client, addr = serverSocket.accept()
    print('Connected to: ' + addr[0] + ':' + str(addr[1]))
    thrd = threading.Thread(target=handle_request,
                            args=(client,))
    thrd.start()
    ThreadCount += 1
    print('Thread Number: ' + str(ThreadCount))
serverSocket.close()
CSIE52400/CSIEM0140 Distributed Systems
Communication 35

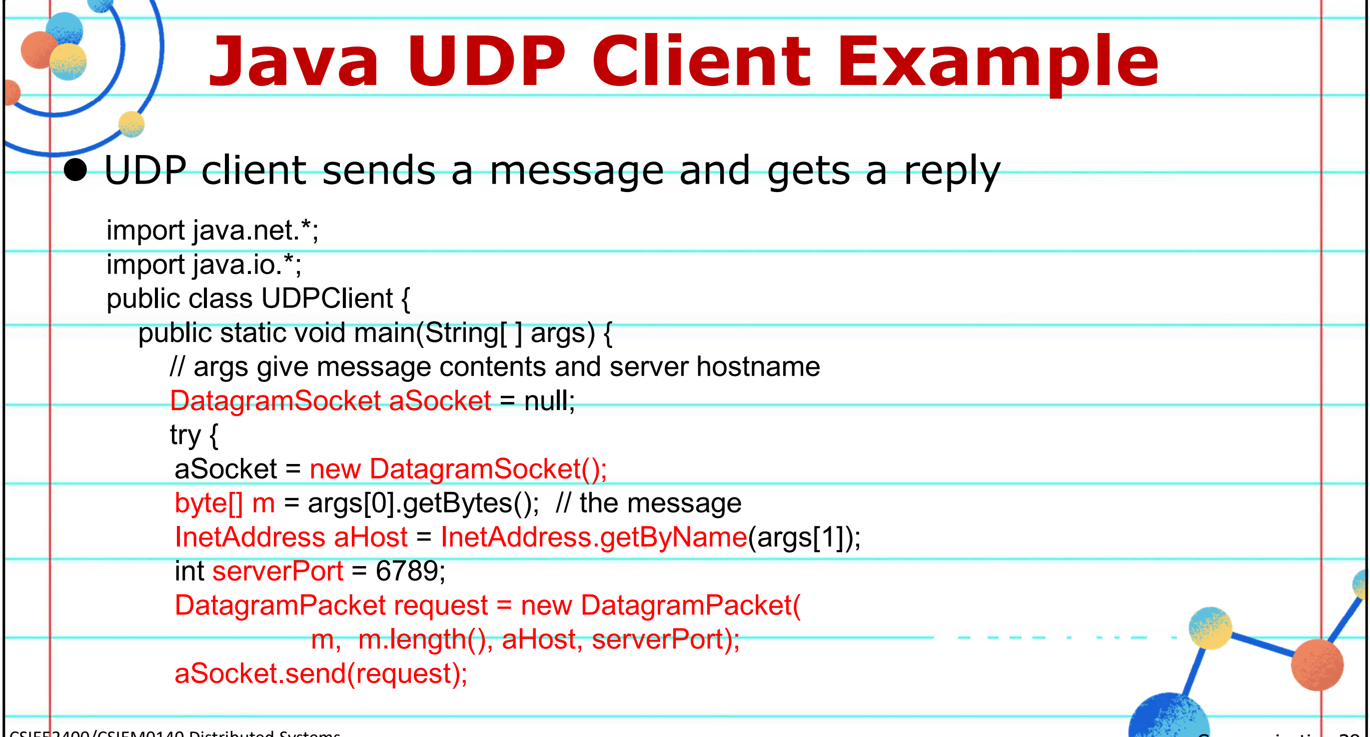

Multithreaded TCP Client
import socket
clientSocket = socket.socket()
host = '127.0.0.1'
port = 2004
print('Waiting for connection response')
try:
    clientSocket.connect((host, port))
except socket.error as e:
    print(str(e))
while True:
    Input = input('Message to send: ')
    clientSocket.send(str.encode(Input))
    res = clientSocket.recv(1024)
    print(res.decode('utf-8'))
clientSocket.close()
CSIE52400/CSIEM0140 Distributed Systems
Communication 36

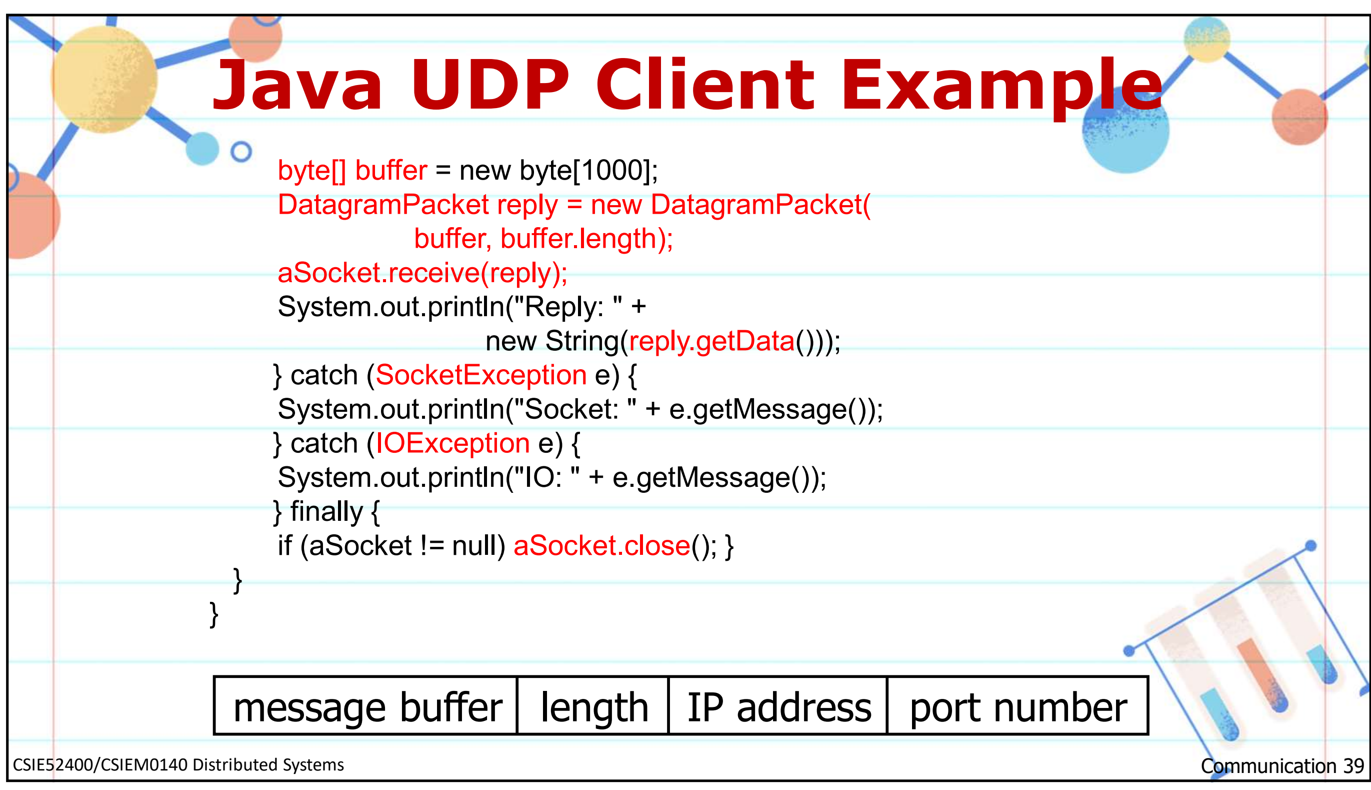
Java API for UDP Datagrams

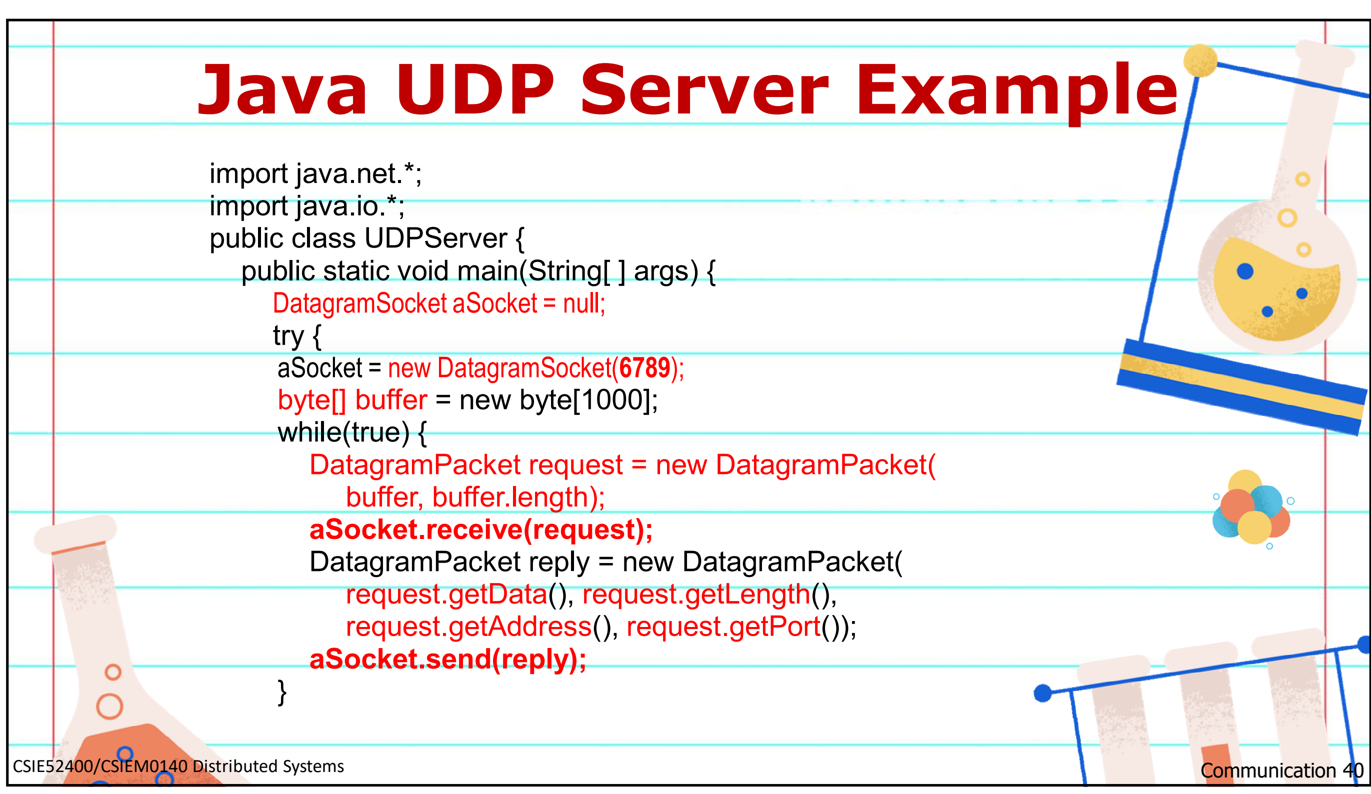
- Java API provides to classes for UDP.
- **DatagramPacket** is for constructing and receiving messages.

message buffer	length	IP address	port number

- **DatagramSocket** supports sockets for sending and receiving UDP datagrams.
  - **send**, **receive**
  - **getData**, **getPort**, **getAddress**
  - **setSoTimeout** // set timeout

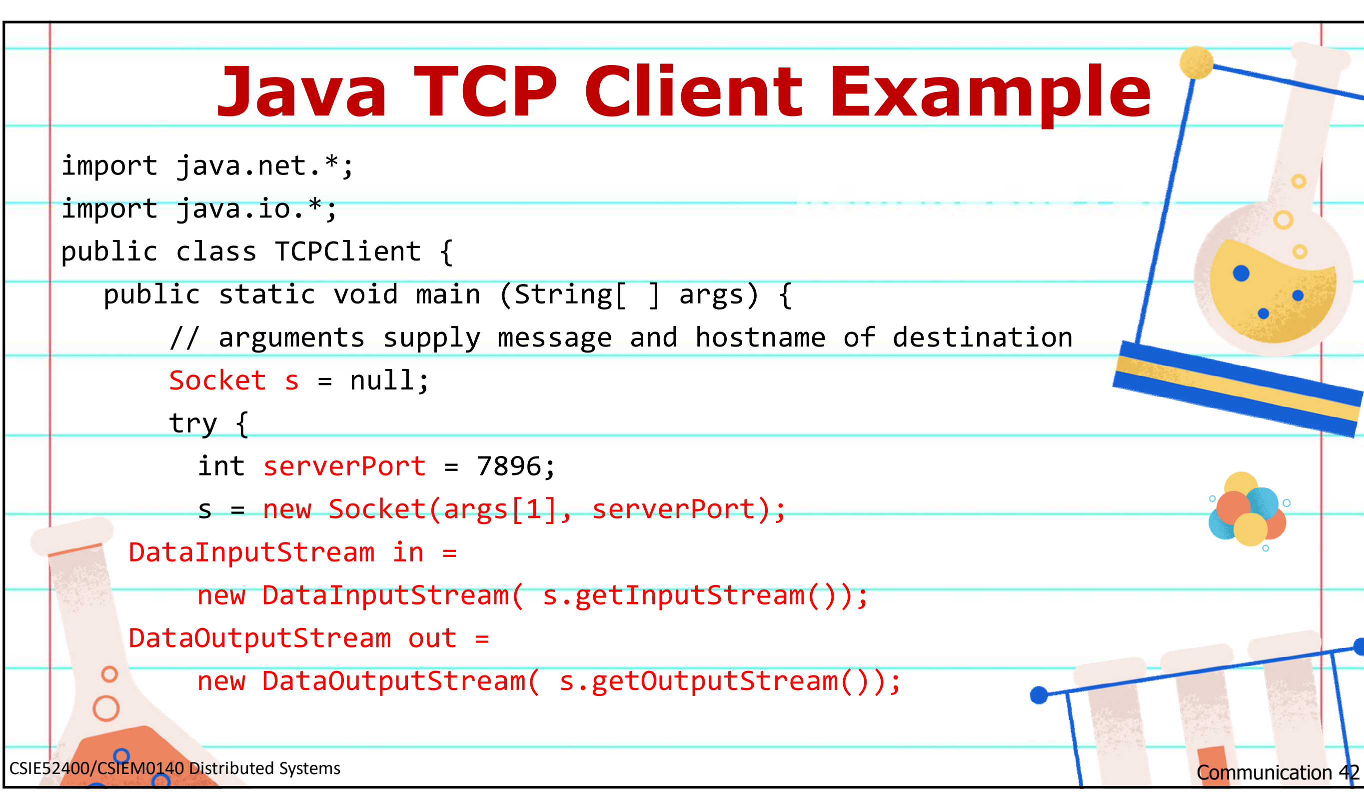


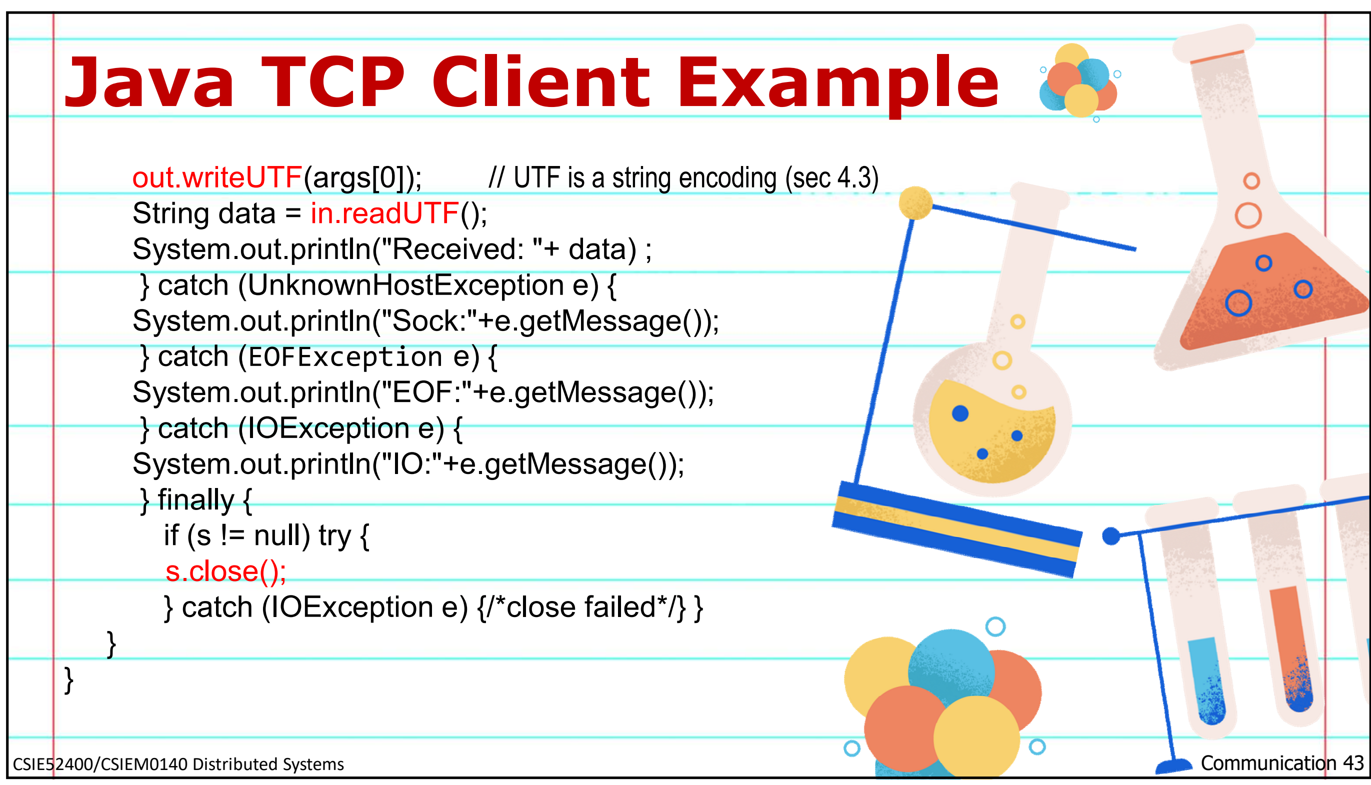

Java UDP Client Example
byte[] buffer = new byte[1000];
DatagramPacket reply = new DatagramPacket(
buffer, buffer.length);
aSocket.receive(reply);
System.out.println("Reply: " +
new String(reply.getData()));
} catch (SocketException e) {
System.out.println("Socket: " + e.getMessage());
} catch (IOException e) {
System.out.println("IO: " + e.getMessage());
} finally {
if (aSocket != null) aSocket.close(); }
}
}
message buffer
length
IP address
port number
CSIE52400/CSIEM0140 Distributed Systems
Communication 39

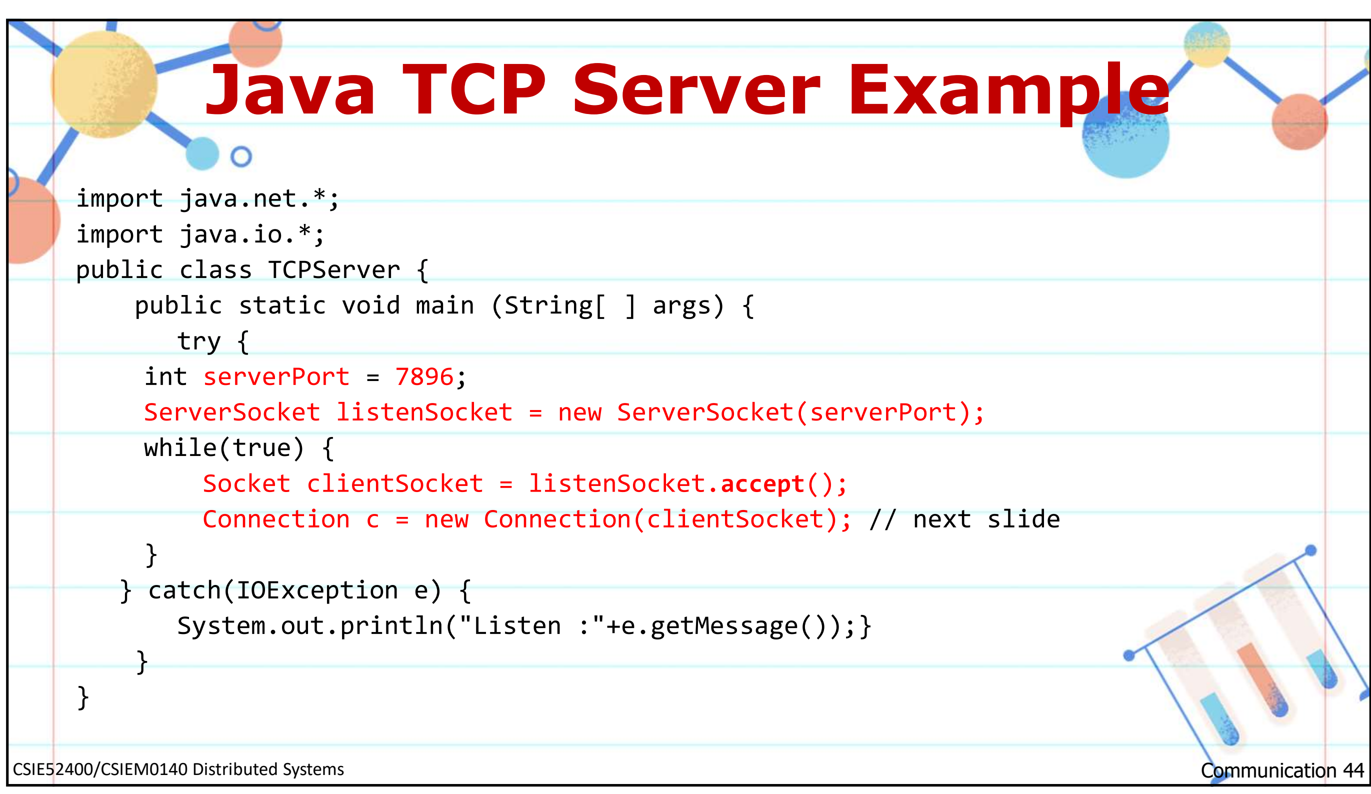

Java UDP Server Example
import java.net.*;
import java.io.*;
public class UDPServer {
public static void main(String[ ] args) {
DatagramSocket aSocket = null;
try {
aSocket = new DatagramSocket(6789);
byte[] buffer = new byte[1000];
while(true) {
DatagramPacket request = new DatagramPacket(
buffer, buffer.length);
aSocket.receive(request);
DatagramPacket reply = new DatagramPacket(
request.getData(), request.getLength(),
request.getAddress(), request.getPort());
aSocket.send(reply);
}
CSIE52400/CSIEM0140 Distributed Systems
Communication 40

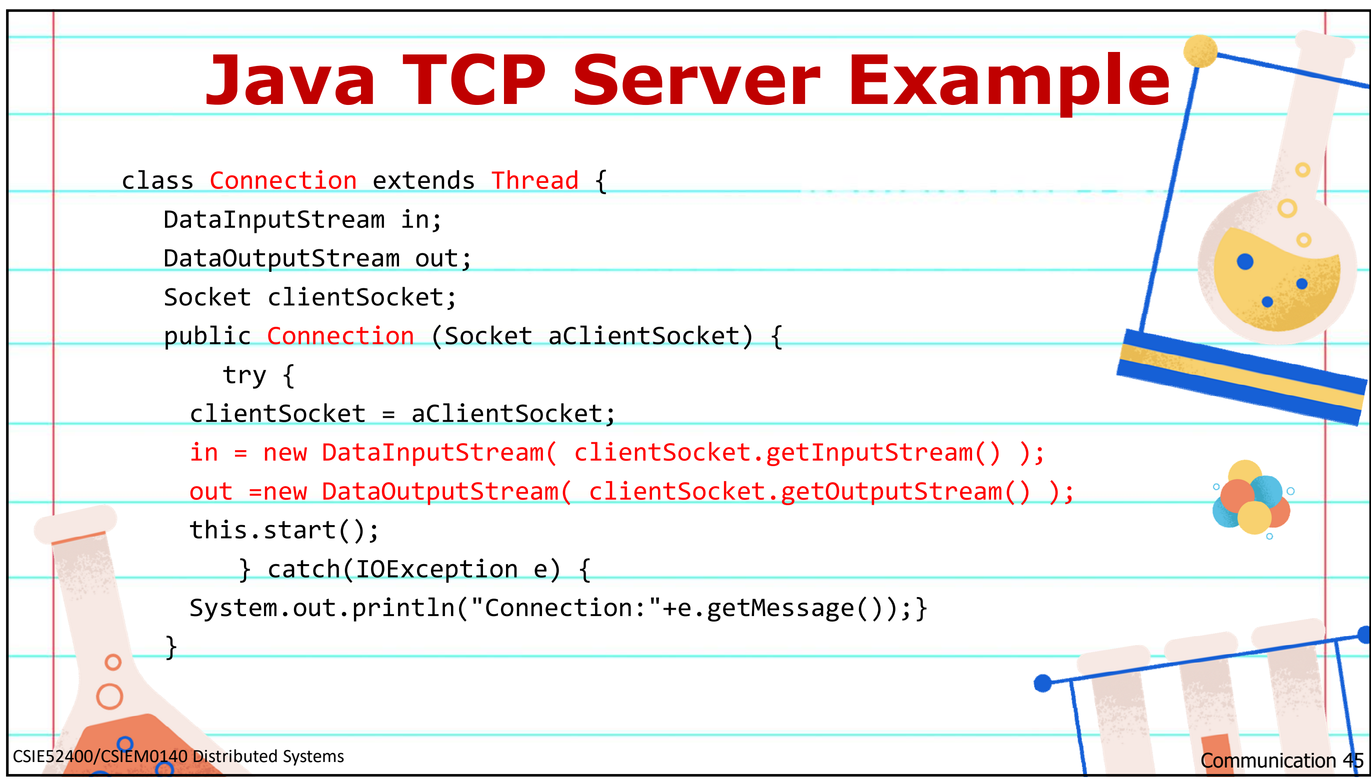
Java UDP Server Example

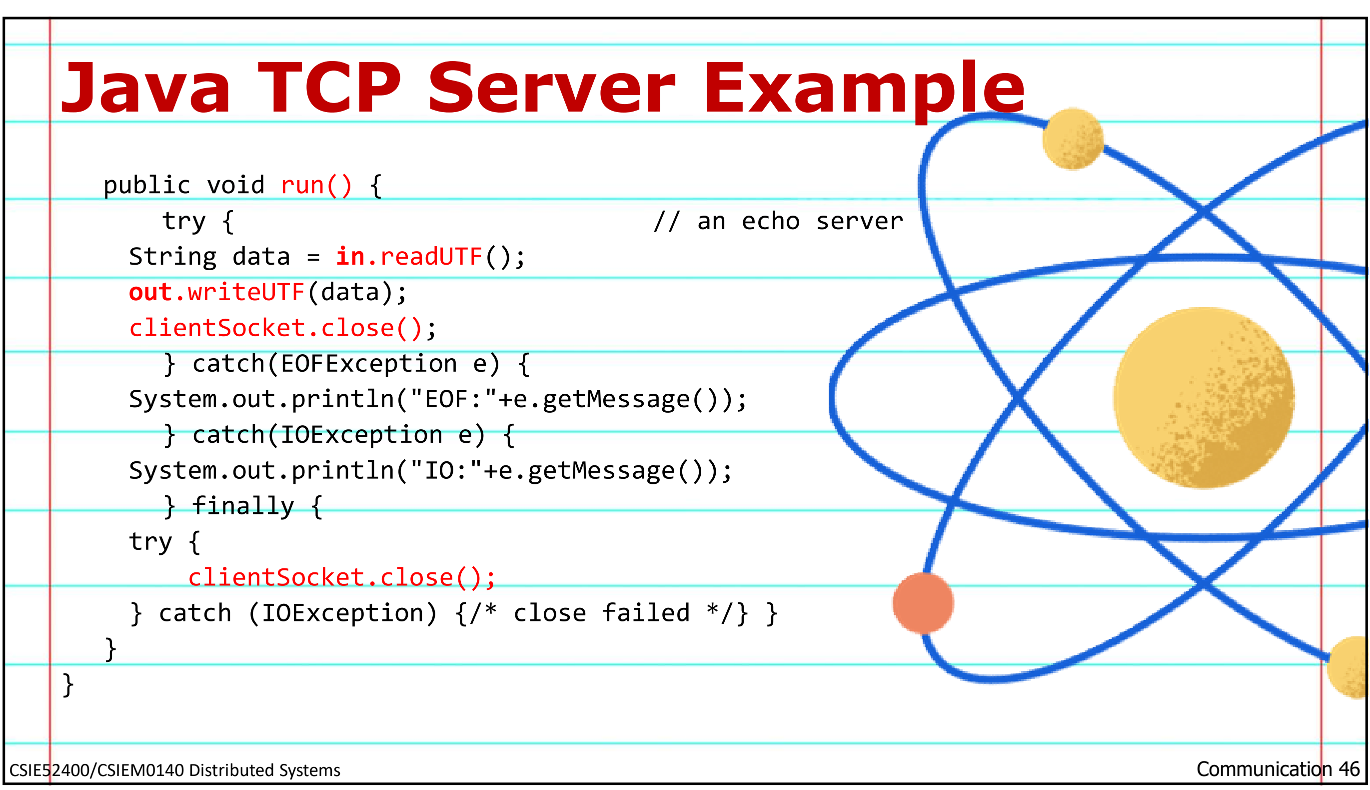
```
            } catch (SocketException e) {
        System.out.println("Socket: " + e.getMessage());
            } catch (IOException e) {
        System.out.println("IO: " + e.getMessage());
            } finally {
                if (aSocket != null) aSocket.close(); }
        }
    }
```


Java TCP Client Example
out.writeUTF(args[0]); // UTF is a string encoding (sec 4.3)
String data = in.readUTF();
System.out.println("Received: "+ data) ;
} catch (UnknownHostException e) {
System.out.println("Sock:"+e.getMessage());
} catch (EOFException e) {
System.out.println("EOF:"+e.getMessage());
} catch (IOException e) {
System.out.println("IO:"+e.getMessage());
} finally {
if (s != null) try {
s.close();
} catch (IOException e) {/*close failed*/} }
}
}
CSIE52400/CSIEM0140 Distributed Systems
Communication 43


Java TCP Server Example
import java.net.*;
import java.io.*;
public class TCPServer {
public static void main (String[ ] args) {
try {
int serverPort = 7896;
ServerSocket listenSocket = new ServerSocket(serverPort);
while(true) {
Socket clientSocket = listenSocket.accept();
Connection c = new Connection(clientSocket); // next slide
}
} catch(IOException e) {
System.out.println("Listen :"+e.getMessage());}
}
}
CSIE52400/CSIEM0140 Distributed Systems
Communication 44


Java TCP Server Example
class Connection extends Thread {
DataInputStream in;
DataOutputStream out;
Socket clientSocket;
public Connection (Socket aClientSocket) {
try {
clientSocket = aClientSocket;
in = new DataInputStream(clientSocket.getInputStream());
out =new DataOutputStream(clientSocket.getOutputStream());
this.start();
} catch(IOException e) {
System.out.println("Connection:"+e.getMessage());}
}
CSIE52400/CSIEM0140 Distributed Systems
Communication 45

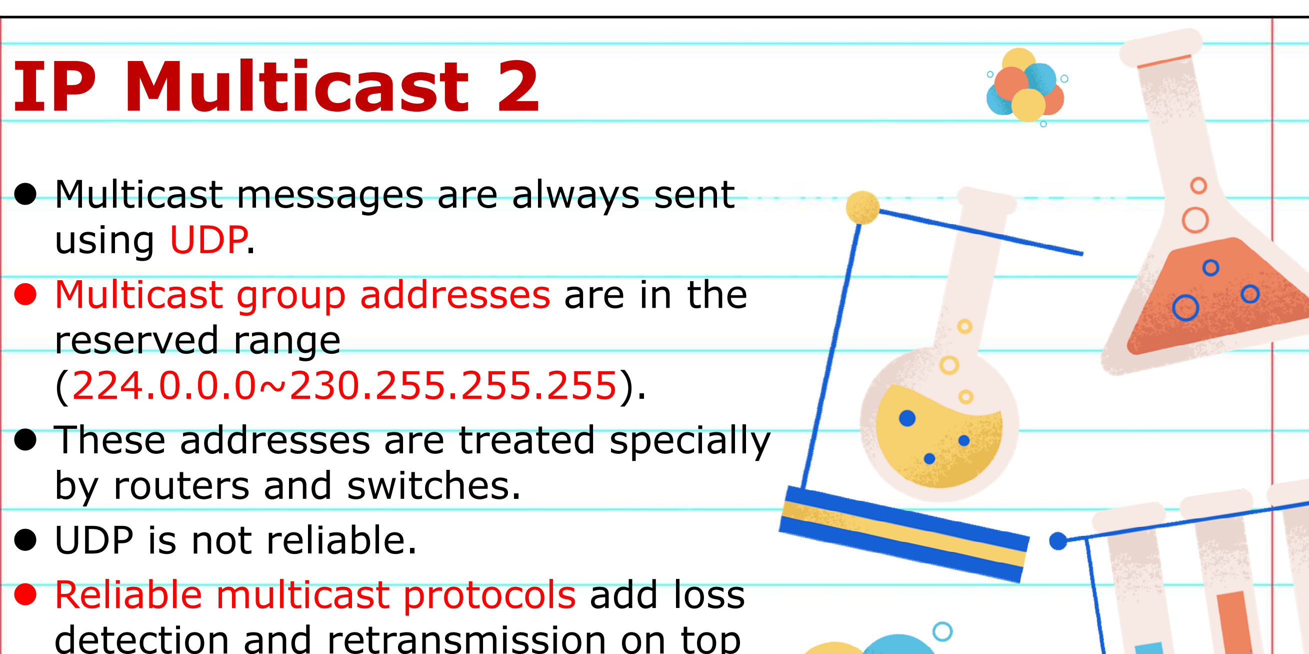

Java TCP Server Example
public void run() {
try {
// an echo server
String data = in.readUTF();
out.writeUTF(data);
clientSocket.close();
} catch(EOFException e) {
System.out.println("EOF:"+e.getMessage());
} catch(IOException e) {
System.out.println("IO:"+e.getMessage());
} finally {
try {
clientSocket.close();
} catch (IOException) {/* close failed */} }
}
}
CSIE52400/CSIEM0140 Distributed Systems
Communication 46

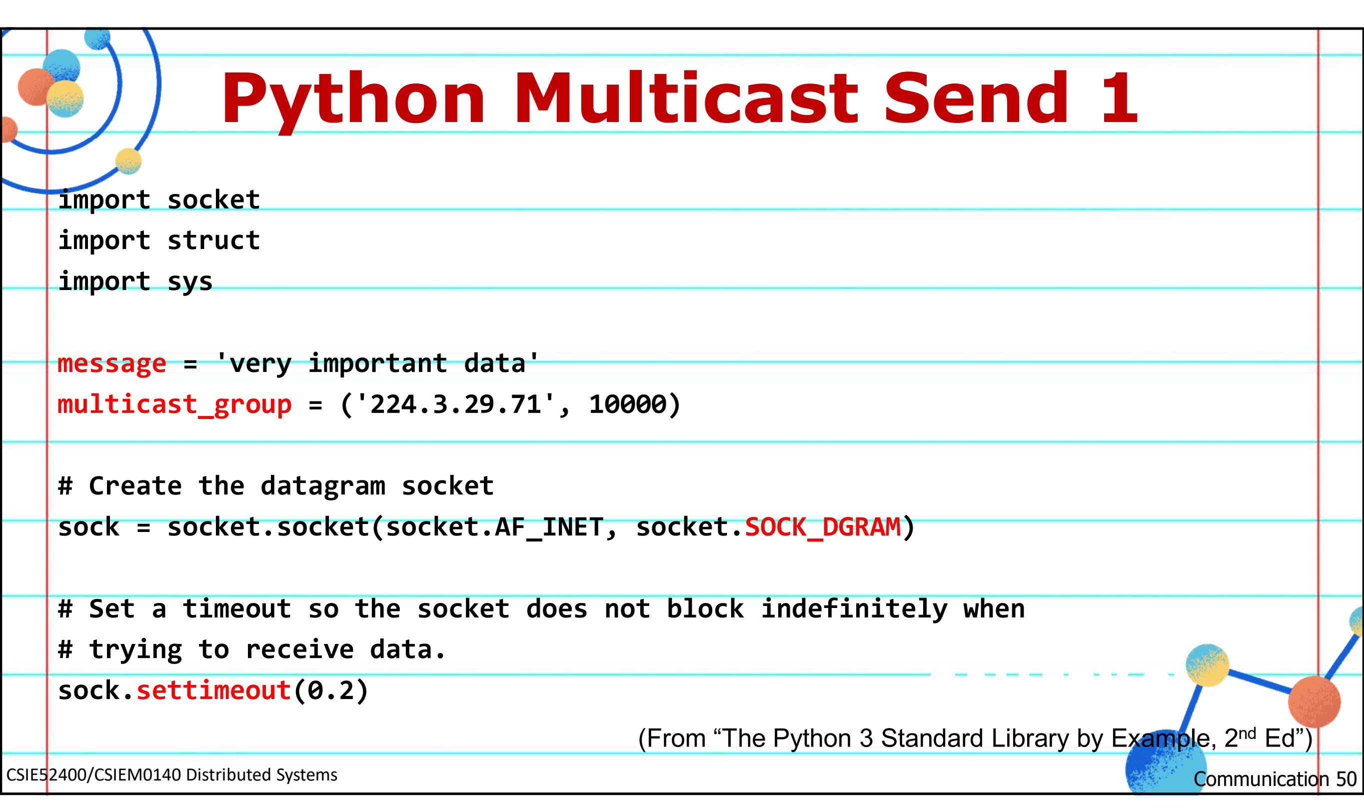
Multicast Communication

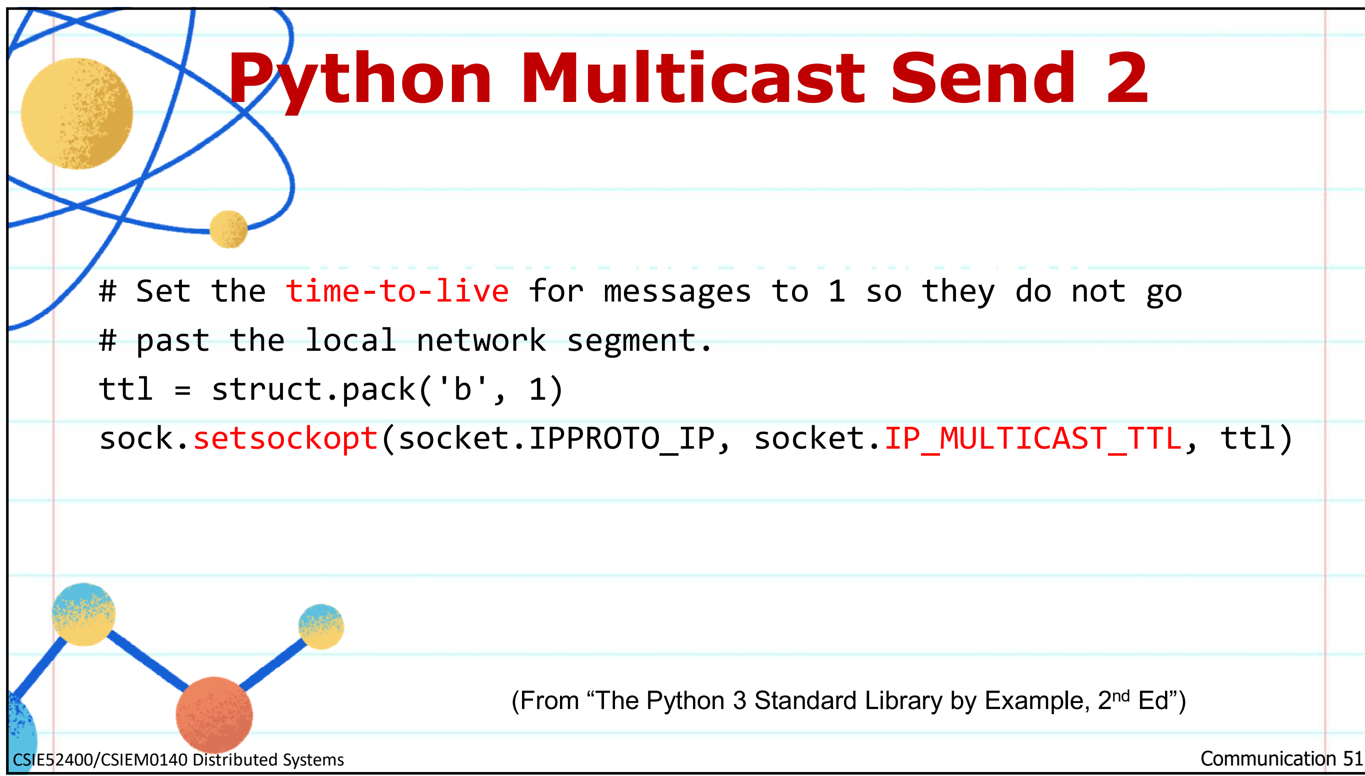
- Multicast operation: sending one message from a process to a group of processes.
- Membership is transparent to the sender.
- Example usage of multicasting
  - Fault tolerance based on replicated services
  - Discovering services in spontaneous networking
  - Better performance through replicated data
  - Propagation of event notifications

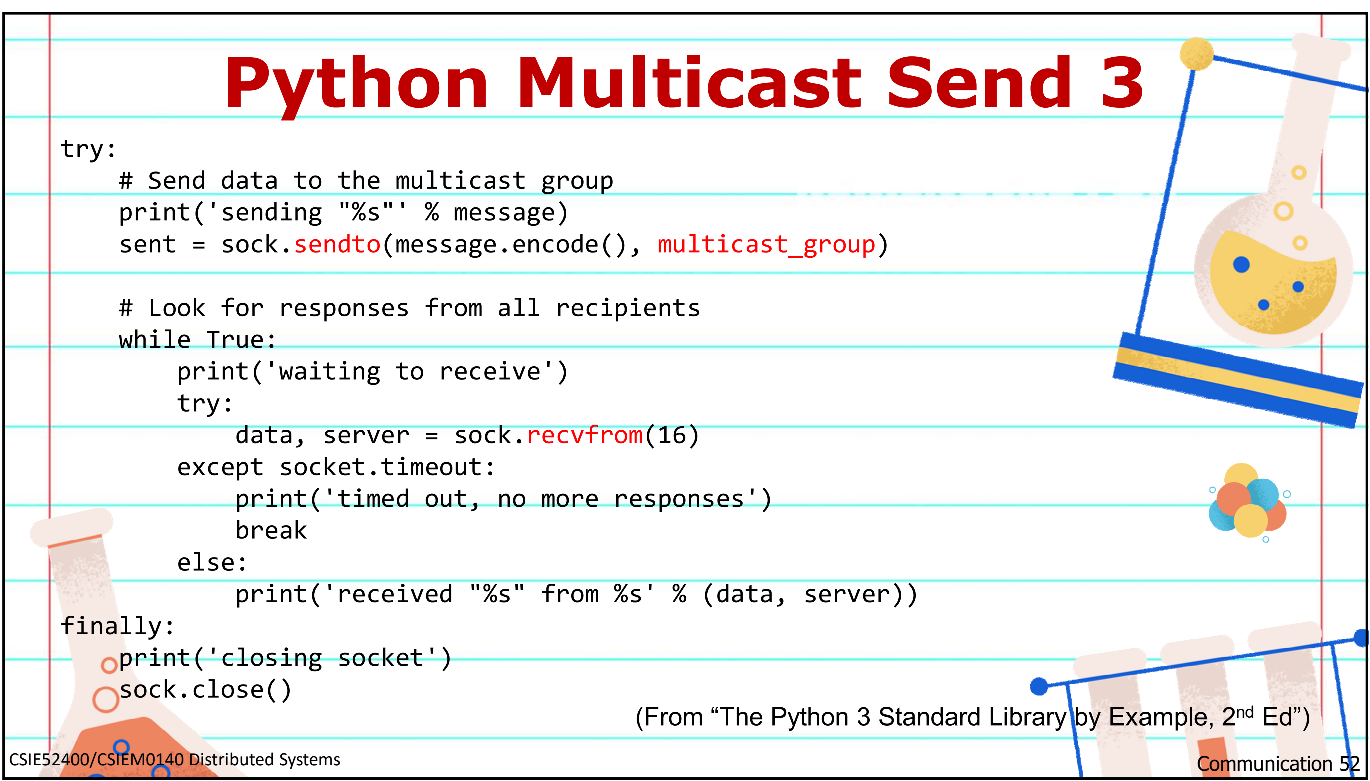
IP Multicast 1

- IP multicast is built on top of IP.
- A single IP packet can be sent to a group of hosts that form a multicast group
- Sender unaware of the receivers and group size
- A multicast group is specified by a Class D address.
- Group membership is dynamic
- Can send to a group w/o being a member
- Multicast addresses may be permanent or temporary
- Permanent groups exist even w/o members
- Multicasting datagrams has the same failure model as UDP datagrams


IP Multicast 2
Multicast messages are always sent using UDP.
Multicast group addresses are in the reserved range (224.0.0.0~230.255.255.255).
These addresses are treated specially by routers and switches.
UDP is not reliable.
Reliable multicast protocols add loss detection and retransmission on top of IP multicast.
CSIE52400/CSIEM0140 Distributed Systems
Communication 49


Python Multicast Send 1
import socket
import struct
import sys
message = 'very important data'
multicast_group = ('224.3.29.71', 10000)
# Create the datagram socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# Set a timeout so the socket does not block indefinitely when
# trying to receive data.
sock.settimeout(0.2)
(From "The Python 3 Standard Library by Example, 2nd Ed")
CSIE52400/CSIEM0140 Distributed Systems
Communication 50


Python Multicast Send 2
# Set the time-to-live for messages to 1 so they do not go
# past the local network segment.
ttl = struct.pack('b', 1)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, ttl)
(From "The Python 3 Standard Library by Example, 2nd Ed")
CSIE52400/CSIEM0140 Distributed Systems
Communication 51

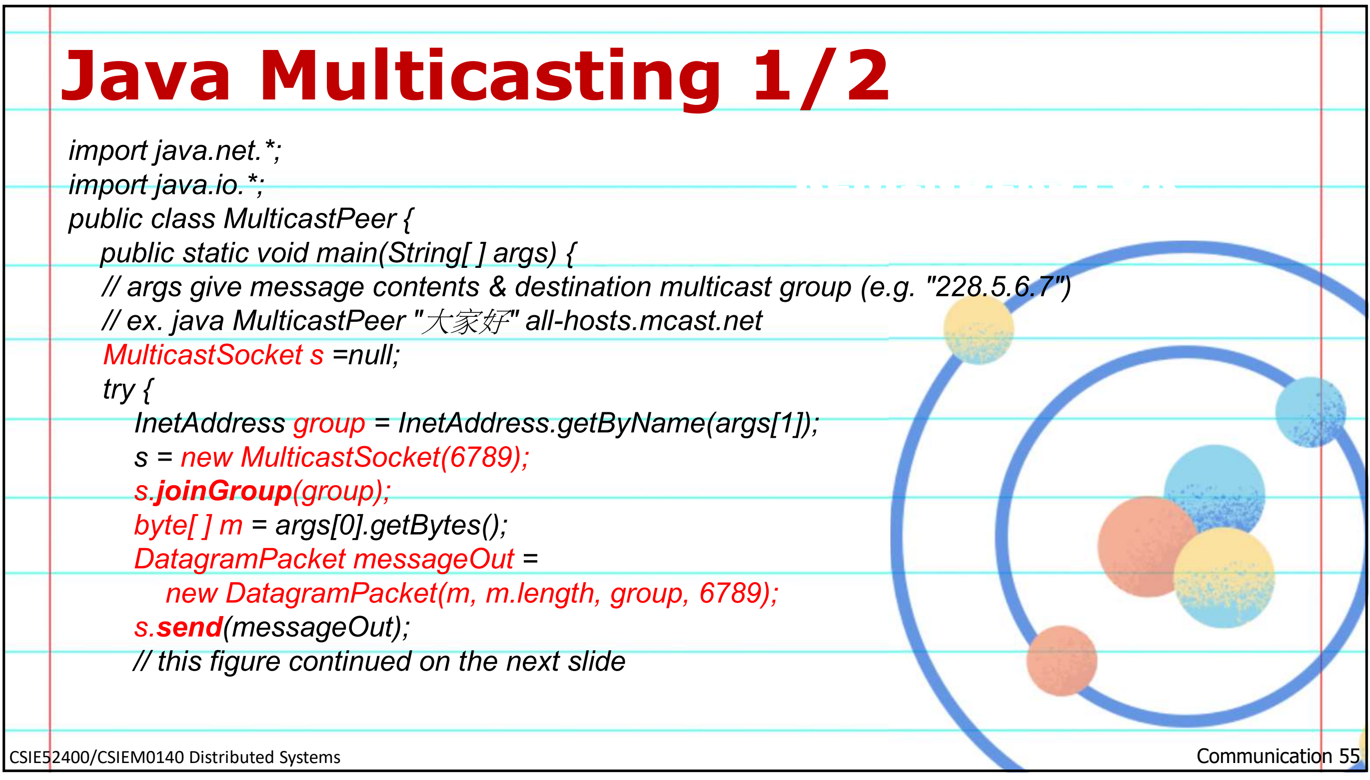

Python Multicast Send 3
try:
# Send data to the multicast group
print('sending "%s"' % message)
sent = sock.sendto(message.encode(), multicast_group)
# Look for responses from all recipients
while True:
print('waiting to receive')
try:
data, server = sock.recvfrom(16)
except socket.timeout:
print('timed out, no more responses')
break
else:
print('received "%s" from %s' % (data, server))
finally:
print('closing socket')
sock.close()
(From "The Python 3 Standard Library by Example, 2nd Ed")
CSIE52400/CSIEM0140 Distributed Systems
Communication 52

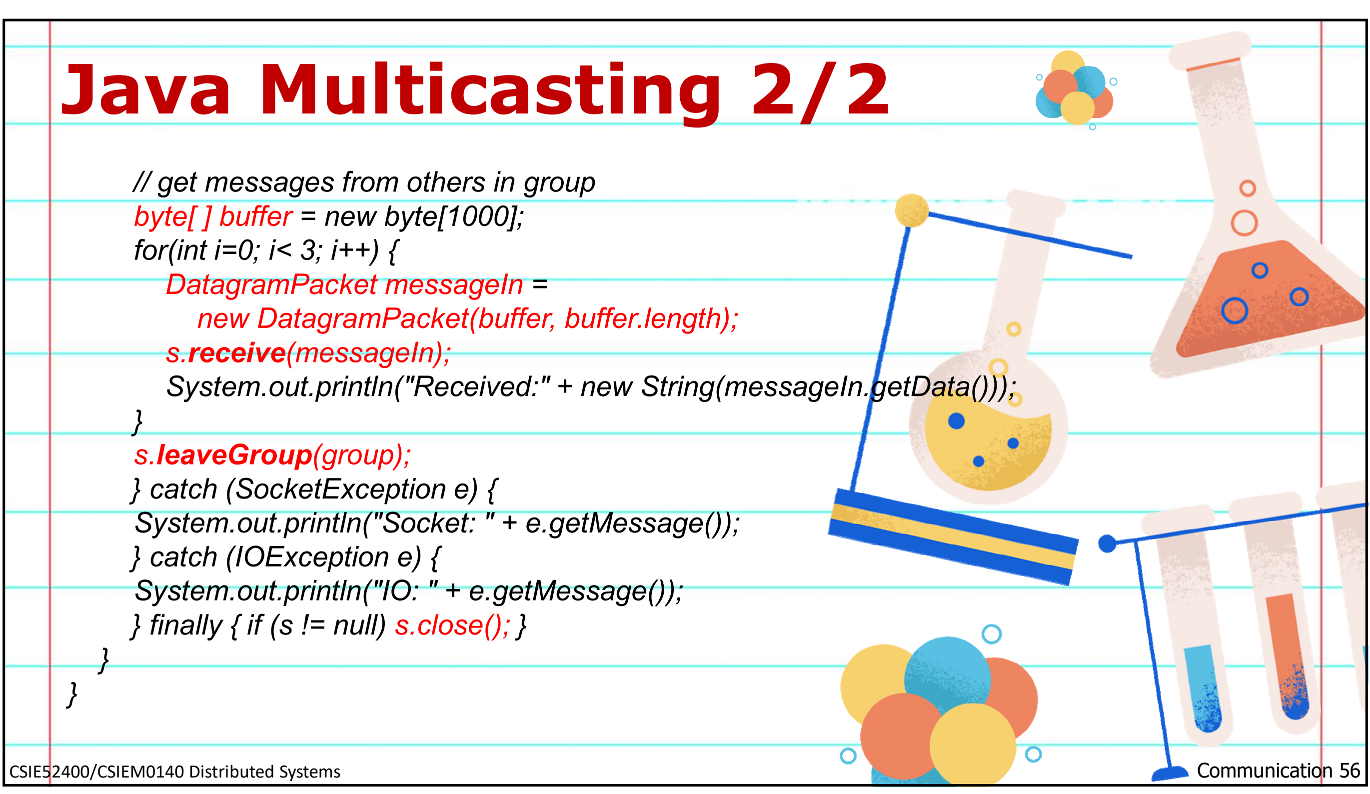
Python Multicast Receive 1

```
import socket
import struct
import sys

multicast_group = '224.3.29.71'
server_address = ('', 10000)

# Create the socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind to the server address
sock.bind(server_address)

# Tell the operating system to add the socket to the multicast group
# on all interfaces.
group = socket.inet_aton(multicast_group)
mreq = struct.pack('4sl', group, socket.INADDR_ANY)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)
```


Python Multicast Receive 2

```
# Receive/respond loop
while True:
    print('\nwaiting to receive message')
    data, address = sock.recvfrom(1024)

    print('received %s bytes from %s' % (len(data),
address))
    print(data)

    print('sending acknowledgement to', address)
    sock.sendto(('ack').encode(), address)
```


Java Multicasting 1/2
import java.net.*;
import java.io.*;
public class MulticastPeer {
public static void main(String[ ] args) {
// args give message contents & destination multicast group (e.g. "228.5.6.7")
// ex. java MulticastPeer "大家好" all-hosts.mcast.net
MulticastSocket s =null;
try {
InetAddress group = InetAddress.getByName(args[1]);
s = new MulticastSocket(6789);
s.joinGroup(group);
byte[ ] m = args[0].getBytes();
DatagramPacket messageOut =
new DatagramPacket(m, m.length, group, 6789);
s.send(messageOut);
// this figure continued on the next slide
CSIE52400/CSIEM0140 Distributed Systems
Communication 55


Java Multicasting 2/2
// get messages from others in group
byte[ ] buffer = new byte[1000];
for(int i=0; i< 3; i++) {
DatagramPacket messageIn =
new DatagramPacket(buffer, buffer.length);
s.receive(messageIn);
System.out.println("Received:" + new String(messageIn.getData()));
}
s.leaveGroup(group);
} catch (SocketException e) {
System.out.println("Socket: " + e.getMessage());
} catch (IOException e) {
System.out.println("IO: " + e.getMessage());
} finally { if (s != null) s.close(); }
}
}
CSIE52400/CSIEM0140 Distributed Systems
Communication 56

Assignment 3: Python TCP, UDP, Client-Server and Multicasting

1. Try ALL the Python TCP/UDP client-server and multithreaded examples. Correct any problem.
2. Test run the Python multicast example. Correct any problem.
3. Write simple TCP/UDP calculate servers that accept simple arithmetic computing requests such as "+ 4 5" (design your own requrest forms) and send back the result. Choose a GUI lib you like (e.g. Tkinter, PySimpleGUI, Kivy, …) and construct simple GUI client(s) to test your servers.
4. Using Python multicast, develop a message board server where publishers can create message boards of different topics. Any new message on a board should be received by all subscribers of that board.

Due date: 3 weeks