# Python Lab. 2.

# Socket Programming

## Lab Overview
## Client/Server Programming with UDP & TCP

### Goal

❖ Build simple **client–server applications** using **Python sockets**

- ➢ Learn how **UDP** and **TCP** communication works

**UDP** – *User Datagram Protocol*

➡️ A fast, connectionless protocol that sends data without guaranteeing delivery.

**TCP** – *Transmission Control Protocol*

➡️ A reliable, connection-oriented protocol that ensures all data arrives correctly and in order.

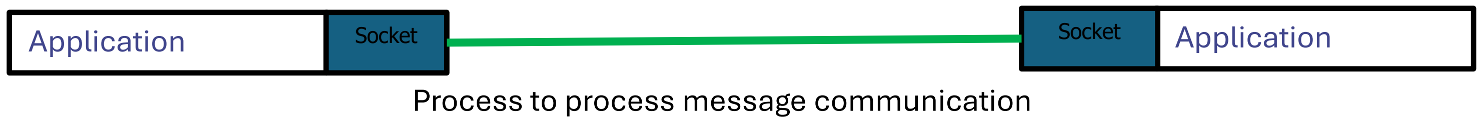
### What We'll Do

Create a **"ToUpper"** application for **UDP** and **TCP**

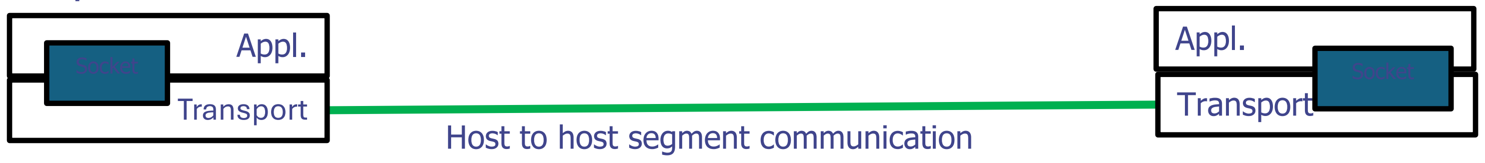
The app will:

- **Client**: reads a line of text from the keyboard
- **Client → Server**: sends the text
- **Server**: converts it to **UPPERCASE**
- **Server → Client**: sends it back
- **Client**: displays the result

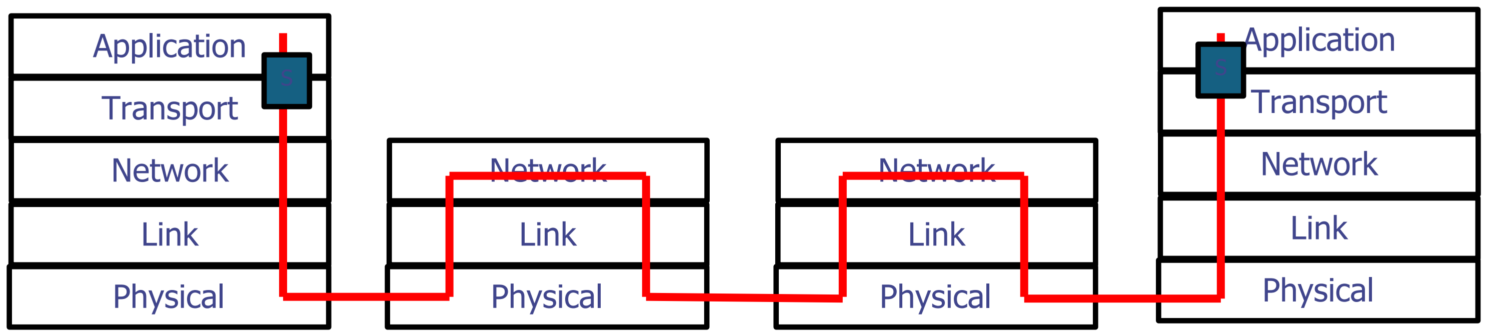
## Application view



## Transport view



## Real view

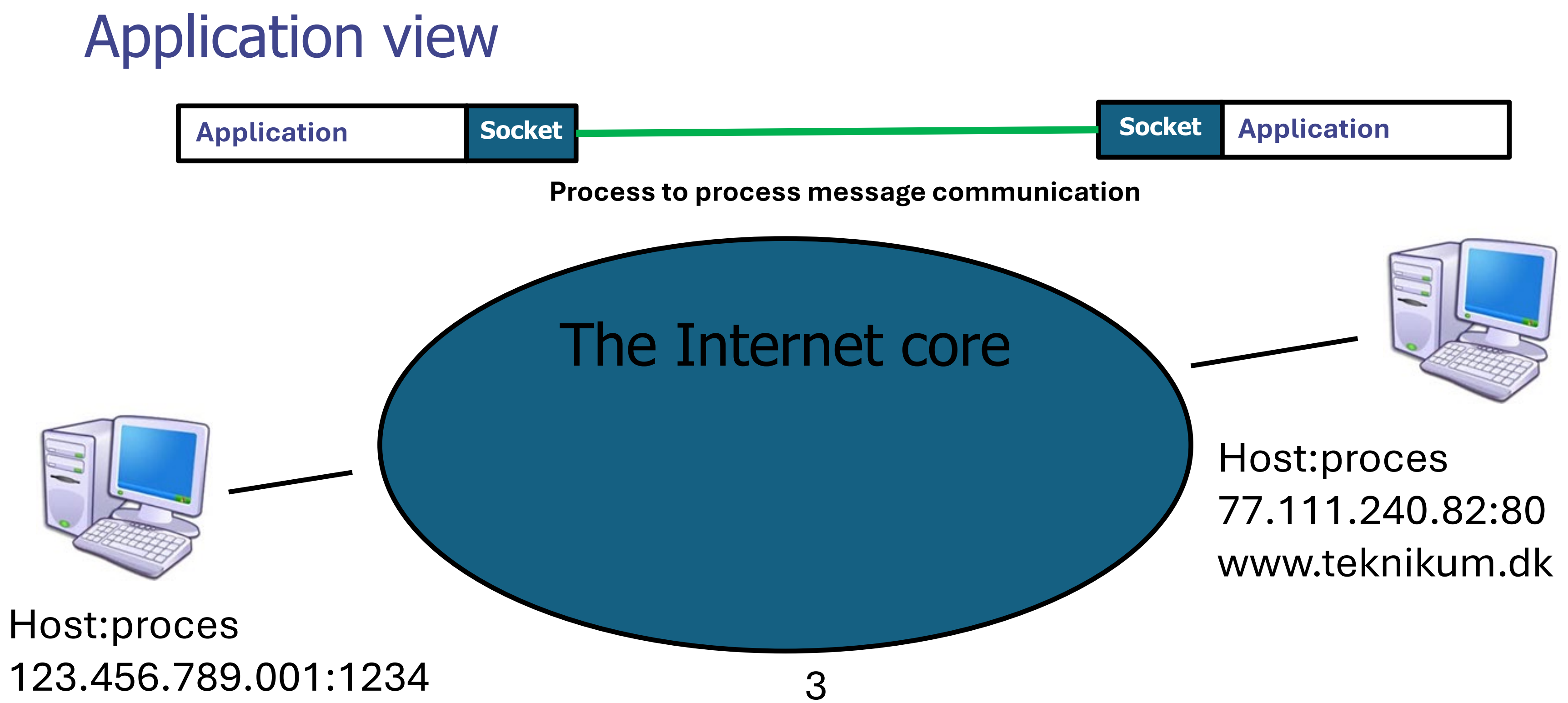


## 🌐 Process-to-Process Communication (Application Layer)

➢ **At the application layer, we want process-to-process communication. (socket-to-socket)**

- A **process** = a program that is running on a computer (like our web browser).
- Each process uses a **socket** to send and receive data.
- So when we say **"process-to-process communication"**, it really means:

  "Data is sent from one program's socket on one computer to another program's socket on another computer."



## Processes communicating

*process:* program running within a host

- within same host, two processes communicate using inter-process communication (defined by OS)
- processes in different hosts communicate by exchanging messages

**clients, servers**

*client process:* process that initiates communication

*server process:* process that waits to be contacted

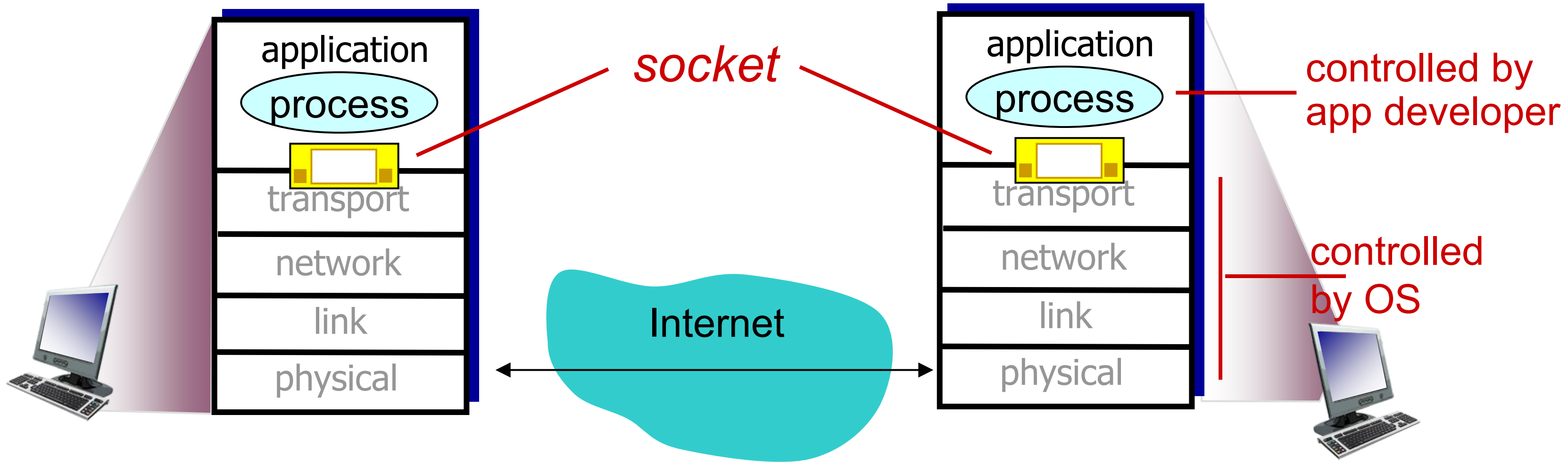
## Addressing processes

- to receive messages, process must have *identifier*
- host device has unique IP address
- *identifier* includes both IP address and port numbers associated with process on host.

- example port numbers:
  - HTTP server: 80
  - mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80

# Sockets

- process sends/receives messages to/from its socket
- 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



AF_INET6
F_INET6

## The socket

IP-address.

Port number.

Address Family:
- AF_INIT
- AF_INET6
- + app. 30 more rarely used for other kind of networks.

Socket types:
- SOCK_DGRAM
- SOCK_STREAM
- SOCK_RAW
- + a couple more rarely used.

```
Sock = socket(address family, type)
Sock.bind( IP, Port))
```

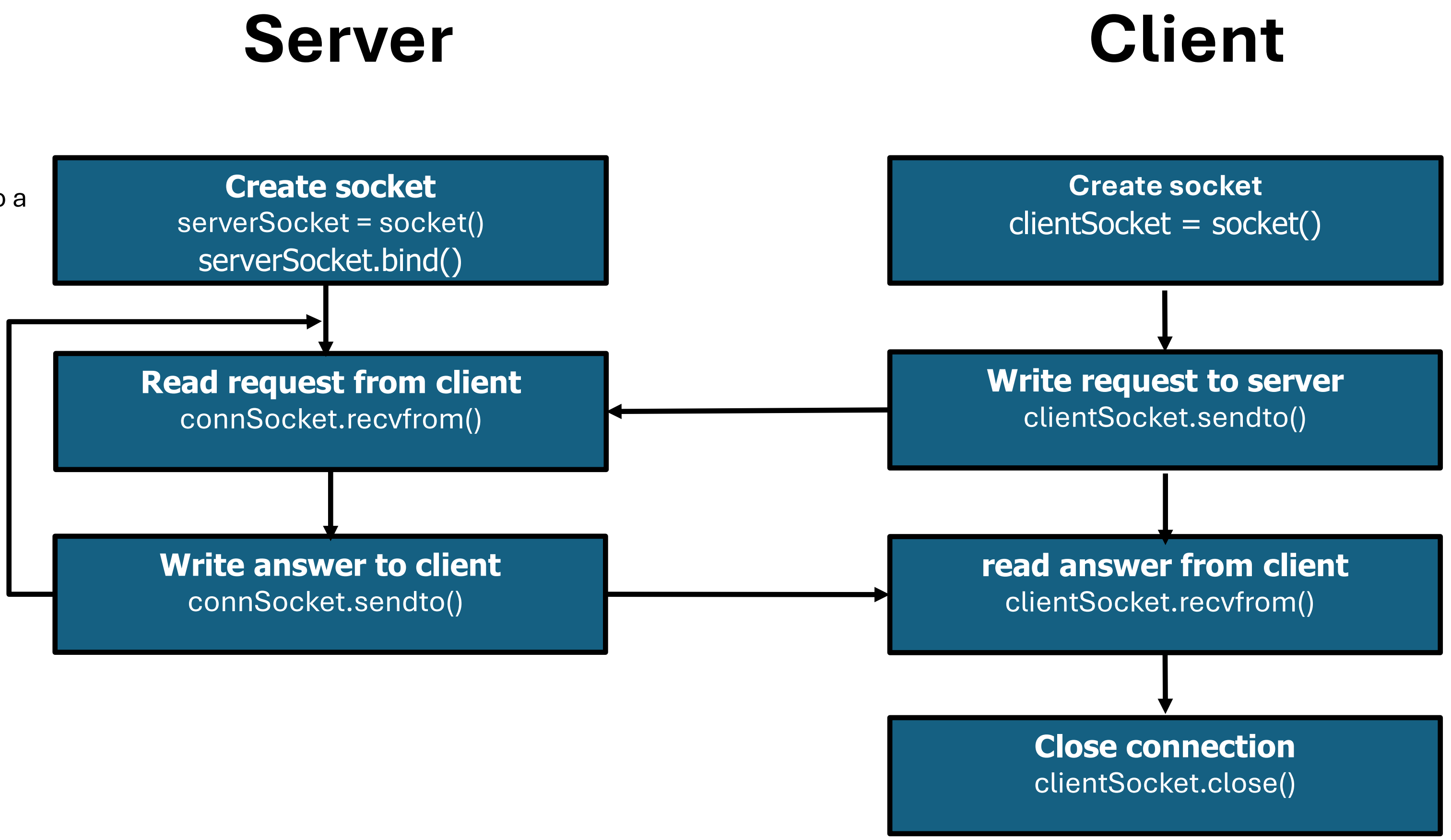
## Client Server Program using sockets

- Socket: an abstraction for processes at the application layer to connect through the network.
  - Needs to know

| | |
|---|---|
| UDP or TCP | # select transport layer protocol |
| Source Address | # for the IP-header |
| Source Port number | # for the TCP/UDP header |
| Destination Address | # for the IP-header |
| Destination Port number | # for the TCP/UDP header |

# The client-server application using UDP

**Bind:** attaching the socket to a specific IP address and port number.



# The client-server application using TCP

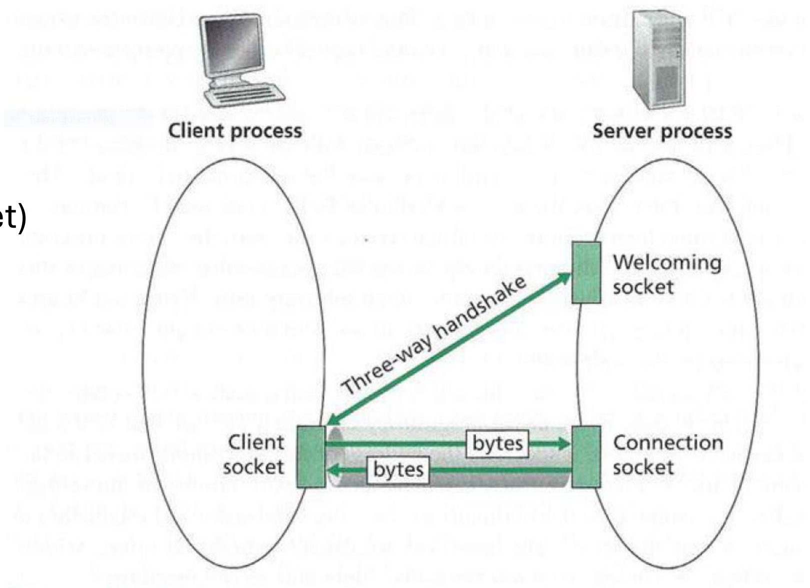
Client:

ClientSocket

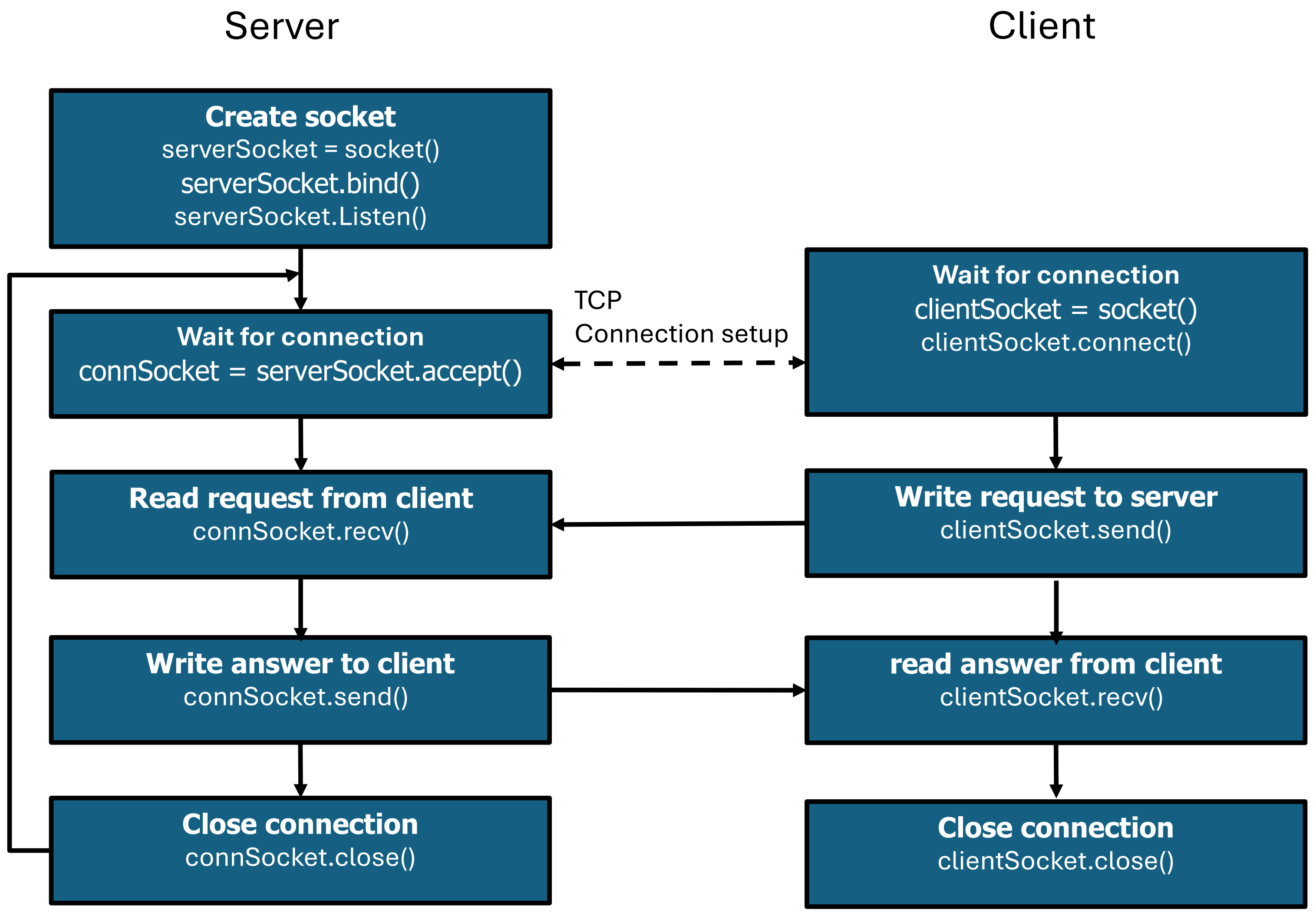
Server:

Serversocket (Welcomming socket)

ConnectionSocket



# The client-server application using TCP


Server
Client
Create socket
serverSocket = socket()
serverSocket.bind()
serverSocket.Listen()
Wait for connection
connSocket = serverSocket.accept()
TCP
Connection setup
Wait for connection
clientSocket = socket()
clientSocket.connect()
Read request from client
connSocket.recv()
Write request to server
clientSocket.send()
Write answer to client
connSocket.send()
read answer from client
clientSocket.recv()
Close connection
connSocket.close()
Close connection
clientSocket.close()