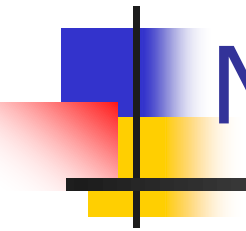
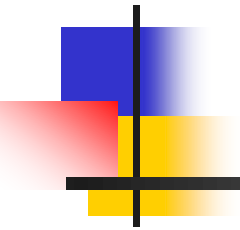


Introduction to Network Programming

The slide has been modified from
Asst. Prof. Chaiporn Jaikaeo, Ph.D.
chaiporn.j@ku.ac.th
http://www.cpe.ku.ac.th/~cpj
Computer Engineering Department
Kasetsart University, Bangkok, Thailand

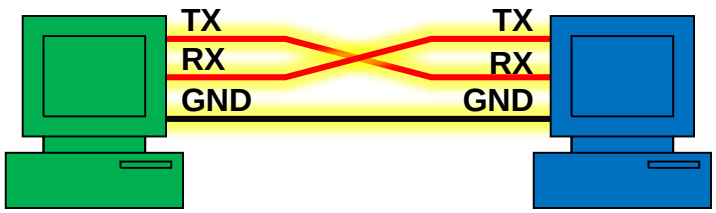


Outline

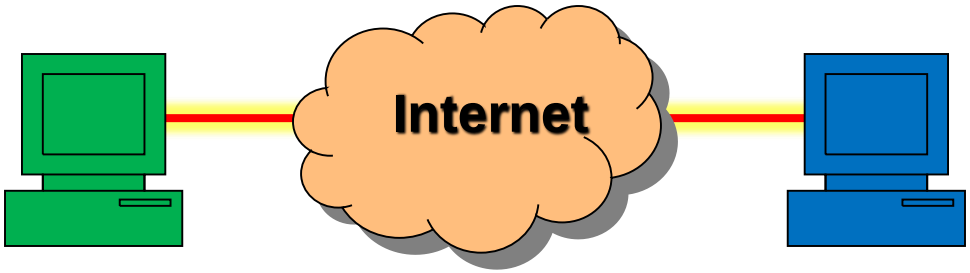
- End-to-end communication
  - Direct serial communication
  - Internet communication
- Simple client-server applications

End-to-End Communication

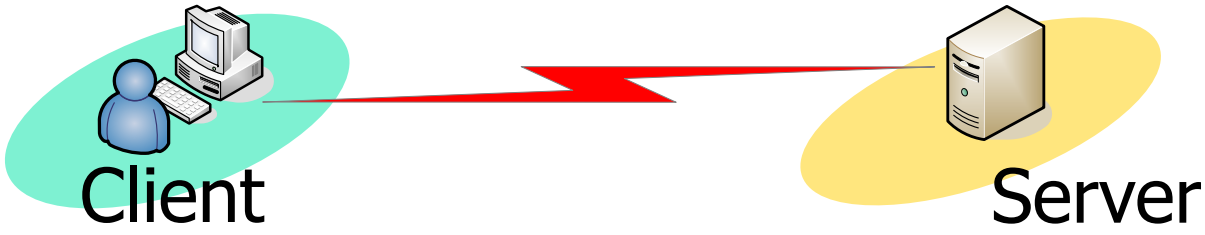
- Direct communication
  - Communicating devices are directly connected



- Internet communication
  - Communication is performed over the Internet

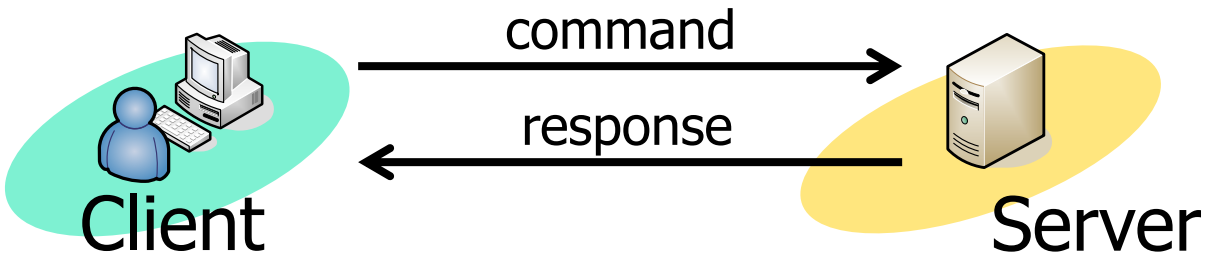


Client-Server Paradigm



- Client
  - Initiates communication
  - Issues command
- Server
  - Awaits and respond to commands
- Most common model for Internet applications

Our Simple Protocol

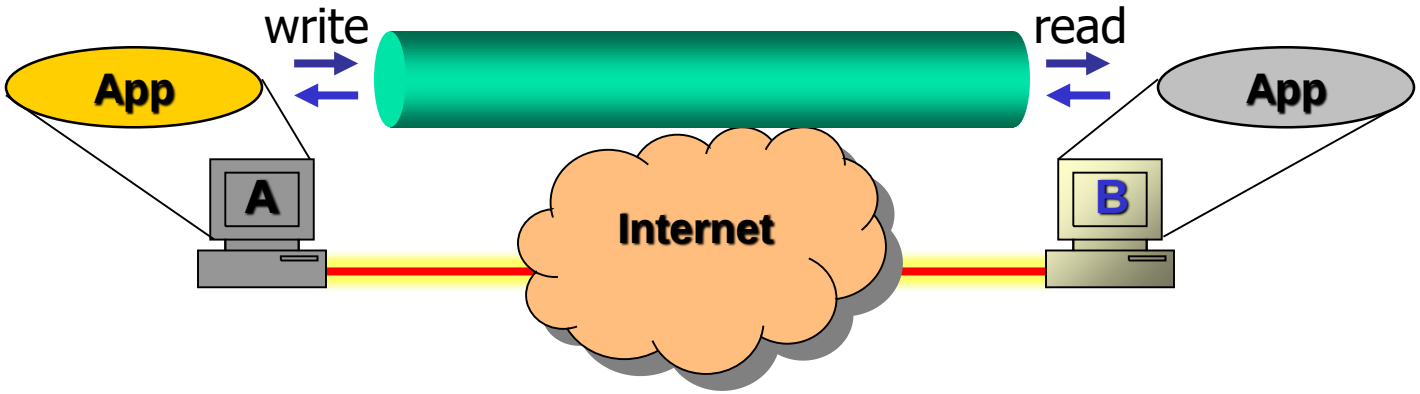


Request Issued by Client	Response by Server
`id\r\n`	Your student ID, followed by `\r\n`
`name\r\n`	Your name, followed by `\r\n`

Server responds with `"ERROR"` when receiving an unknown command.

Internet Comm. - *App's Viewpoint*

- Two network applications should interact as if they were directly connected

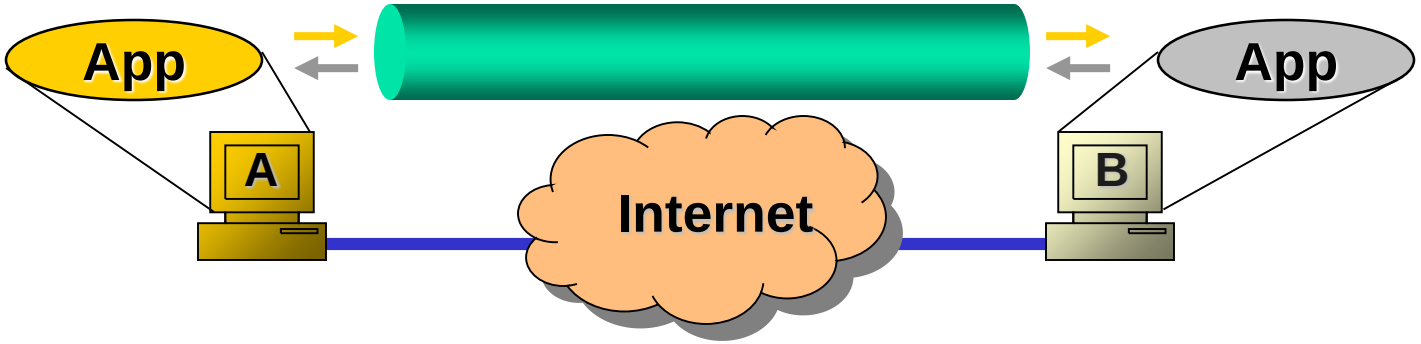


Internet Comm. – Socket API

- API for developing applications that perform inter-process communication
  - most commonly for communications across a computer network
- Example functions
  - **listen** – used by server to wait for contact from client
  - **connect** – used by client to contact server
  - **send** – used by either client or server to send data
  - **recv** – used by either client or server to receive data
  - **close** – close the connection

Services Provided by Socket API

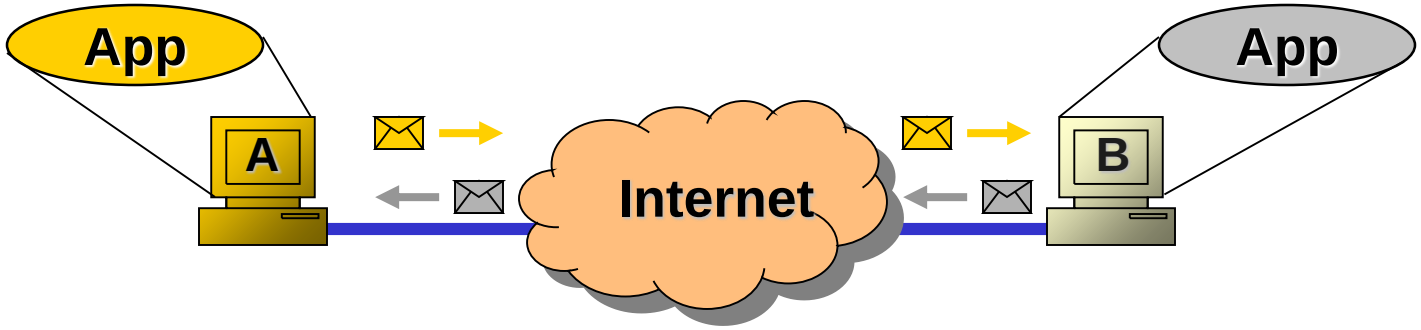
- **Connection-oriented, stream-like service**
  - Provides virtual stream-oriented pipe
  - Data transfer is reliable
    - No loss, in-order arrival



- Both machines use Transmission Control Protocol (TCP) to transfer data

Services Provided by Socket API

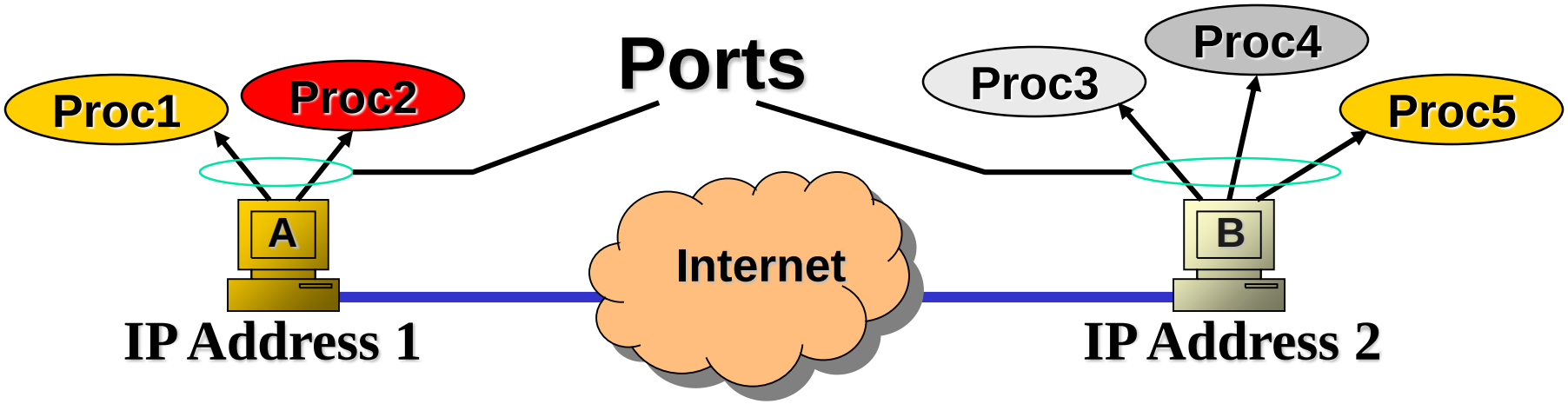
- **Connectionless, datagram service**
  - User must prepare packet of data before sending
  - Data transfer is NOT reliable
    - Loss possible, out-of-order arrival possible

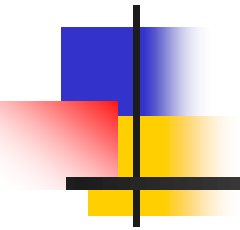


- Both machines use User-Datagram Protocol (UDP) to transfer data

Port Addressing

- IP addresses are used to identify hosts (i.e., machines) on the Internet
- Port numbers are used to distinguish different processes running on the same host

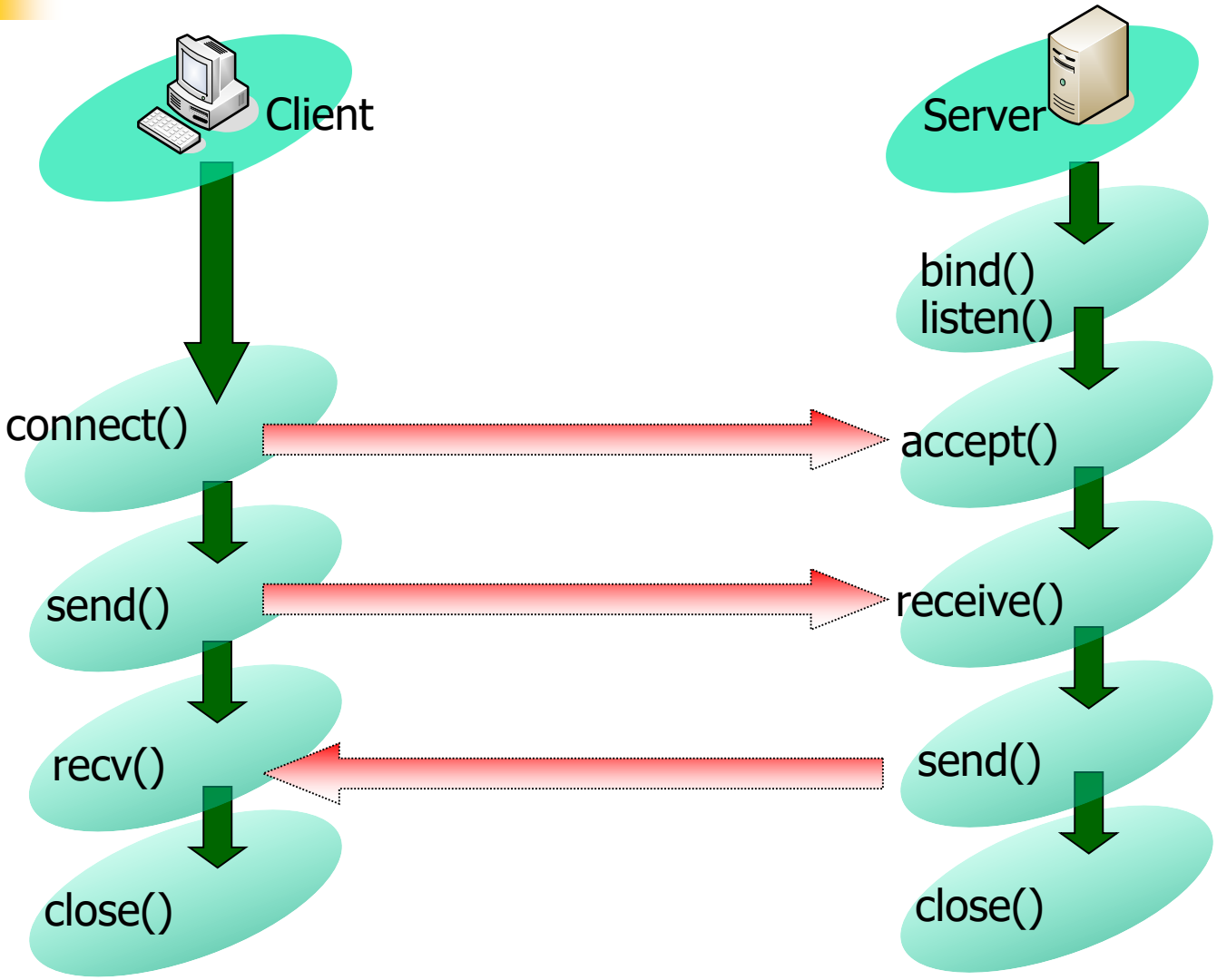




Hands-on Activity: Internet Comm.

- Implement Activity over the Internet using stream-oriented service

TCP Socket: Flow in Python



Server: Creating Socket

- Create a socket object and bind it to port 12345 on any available network interface

```
from socket import *

listen_sock = socket()
listen_sock.bind(('0.0.0.0',12345))
listen_sock.listen(1)   # max. of 1 client can wait
sock,info = listen_sock.accept()
```

Listen on any interface

Port 12345

Client: Create Socket

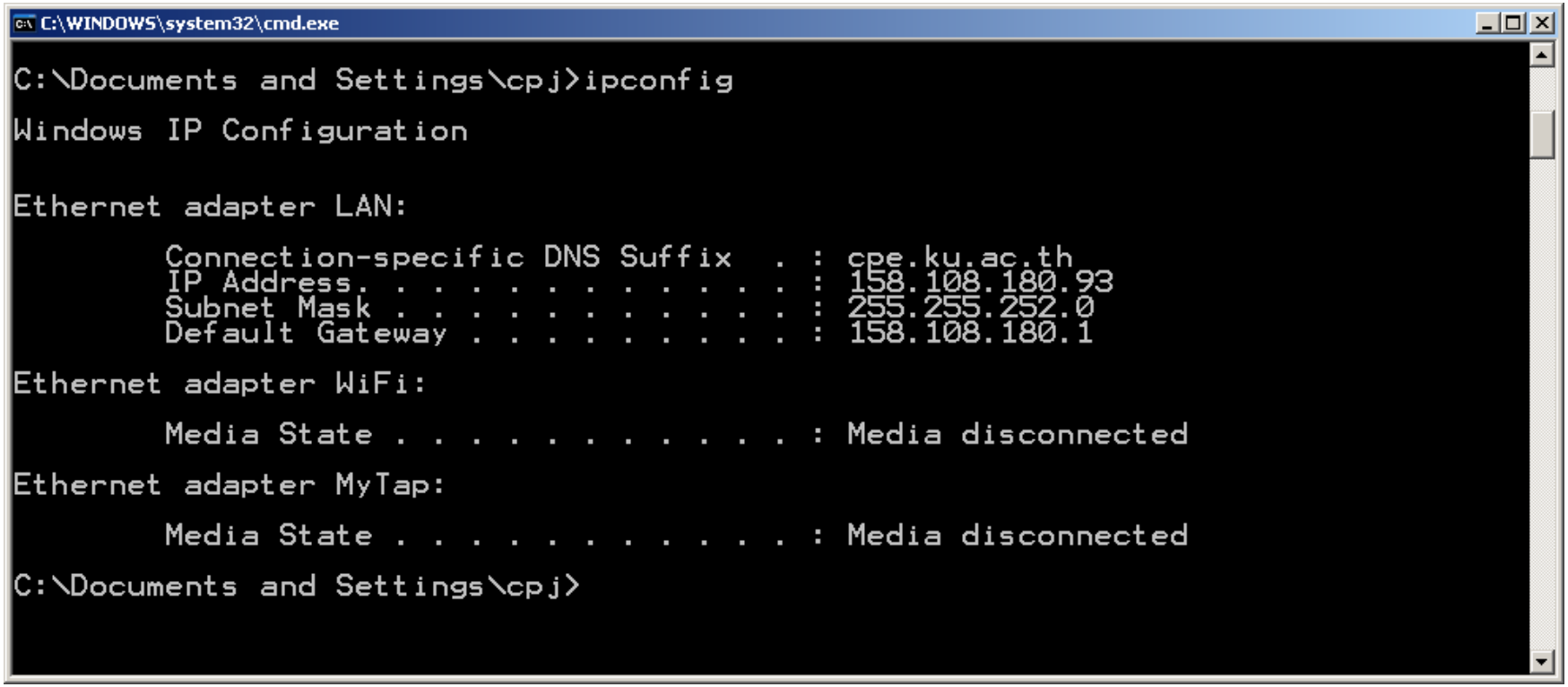
- On another machine, create a client socket and connect to the server's IP and port

```
from socket import *

sock = socket()
server = ("<server's IP>", 12345)
sock.connect(server)
```

Finding Out Your IP Address

- Windows – Run **`ipconfig`** from a command prompt
- MacOS/Linux/Unix – Run **`ifconfig`** from a terminal



Server: Check Client's IP and Port

- The method **`getpeername()`** tells us about client's IP address and port number
  - The method returns a tuple (ip-addr, port)

```
addr,port = sock.getpeername()
print "Client is connected from",addr
```

Transferring Data

- Use **`send()`** and **`recv()`** methods to send and receive data, respectively
- E.g.,
  - At the server, enter the command

```
msg = sock.recv(1024)
```

specify the max number of bytes to be received

The server will block until it receives data

  - At the client, enter the command

```
sock.send('hello')
```

Closing Connection

- Server:

```
sock.close()
listen_sock.close()
```

- Client:

```
sock.close()
```