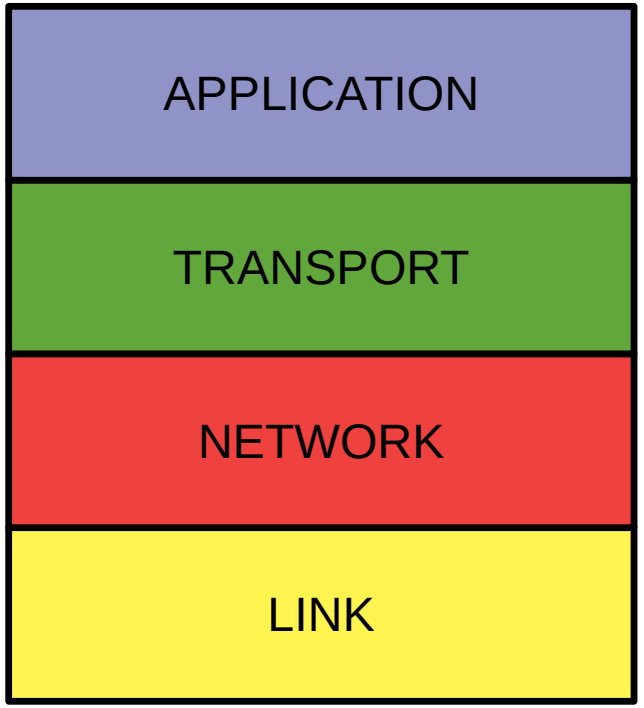
CS 212
Computer Networks Lab
Sockets

Spring 2021

IIT Goa

Slide Credits: Some of the slides and figures are adapted from:

- Dr. Sylvia Ratnasamy's lecture slides for CS168 https://inst.eecs.berkeley.edu/~cs168/fa14/
- Companion slides for: Computer Networks: a Top-down Approach (https://gaia.cs.umass.edu/kurose_ross/index.html)


APPLICATION
TRANSPORT
NETWORK
LINK

The Applications Layer

Consists of: Applications running on hosts communicating over the Internet.

Q: What information is needed for an application to communicate with another application running on a remote end-host?

Q: What is the *Interface* between the Application layer and the Transport layer? How can applications use the services of the Transport layer

- For Reliable, connection-oriented data transfer? (TCP)
- For Best-effort, connectionless data transfer? (UDP)

Recall the difference between

- **A Program**
- **A Process**
- **An Application**

Recall the difference between

- **A Program**
- **A Process**
- **An Application**

So which of these are the end-points of communication?

Sockets

- Sockets serve as an Interface between Applications and the Transport layer

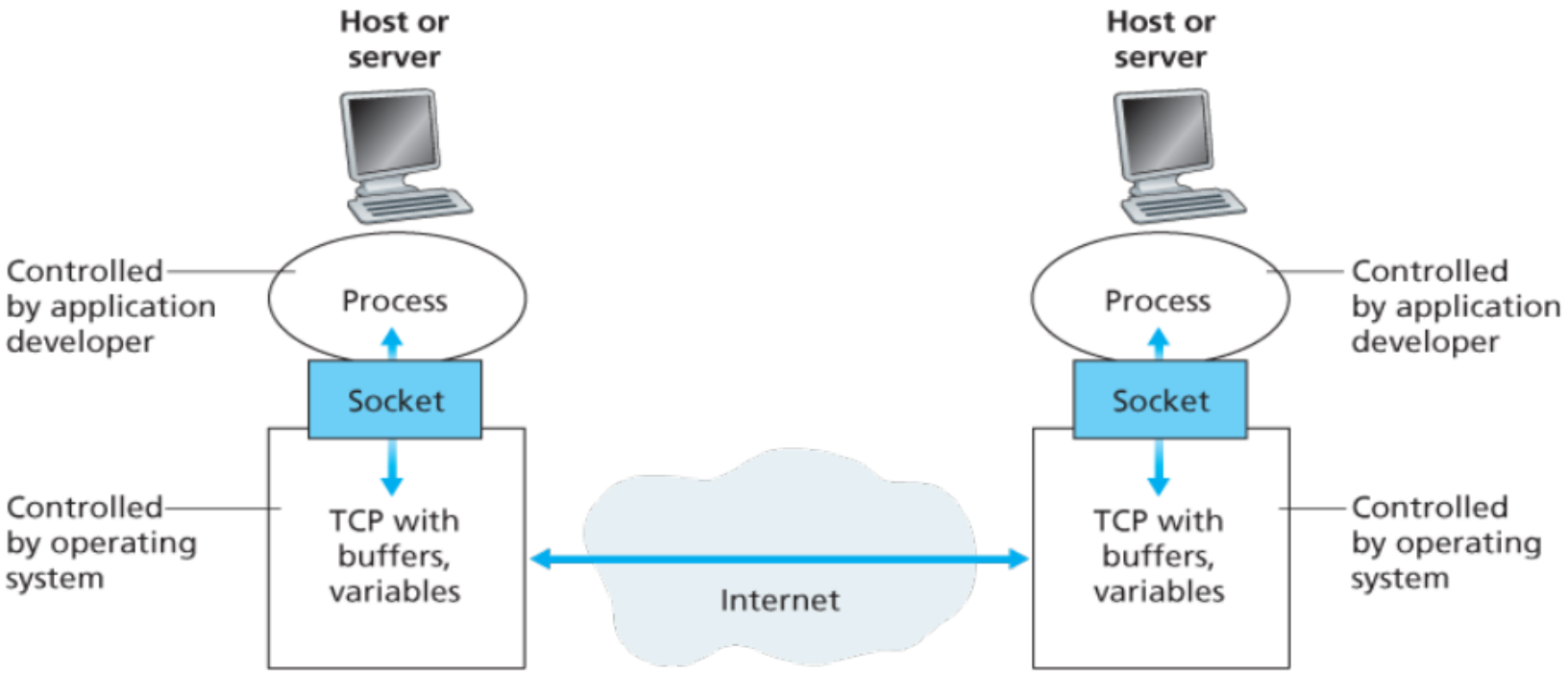
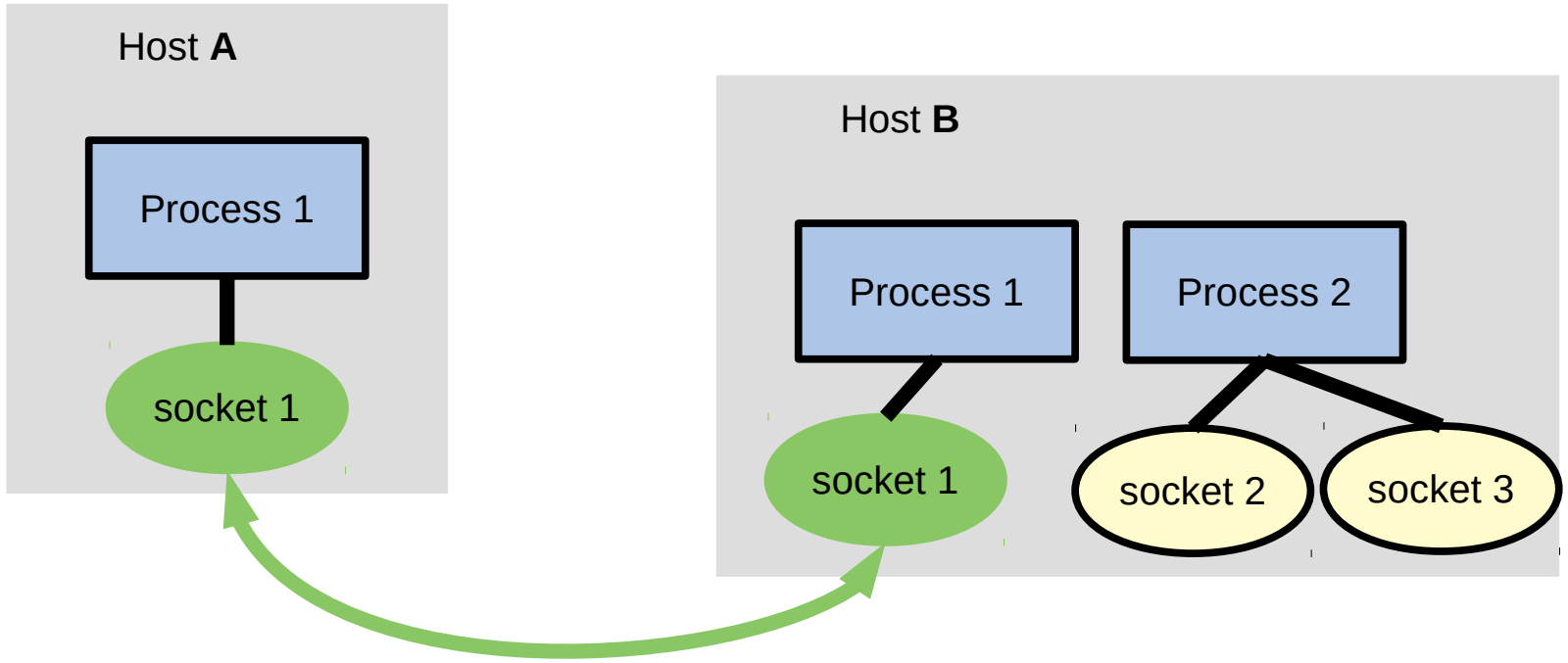


Figure 2.3 Application processes, sockets, and underlying transport protocol

Sockets

- **Sockets are end-points of a communication stream between two processes**



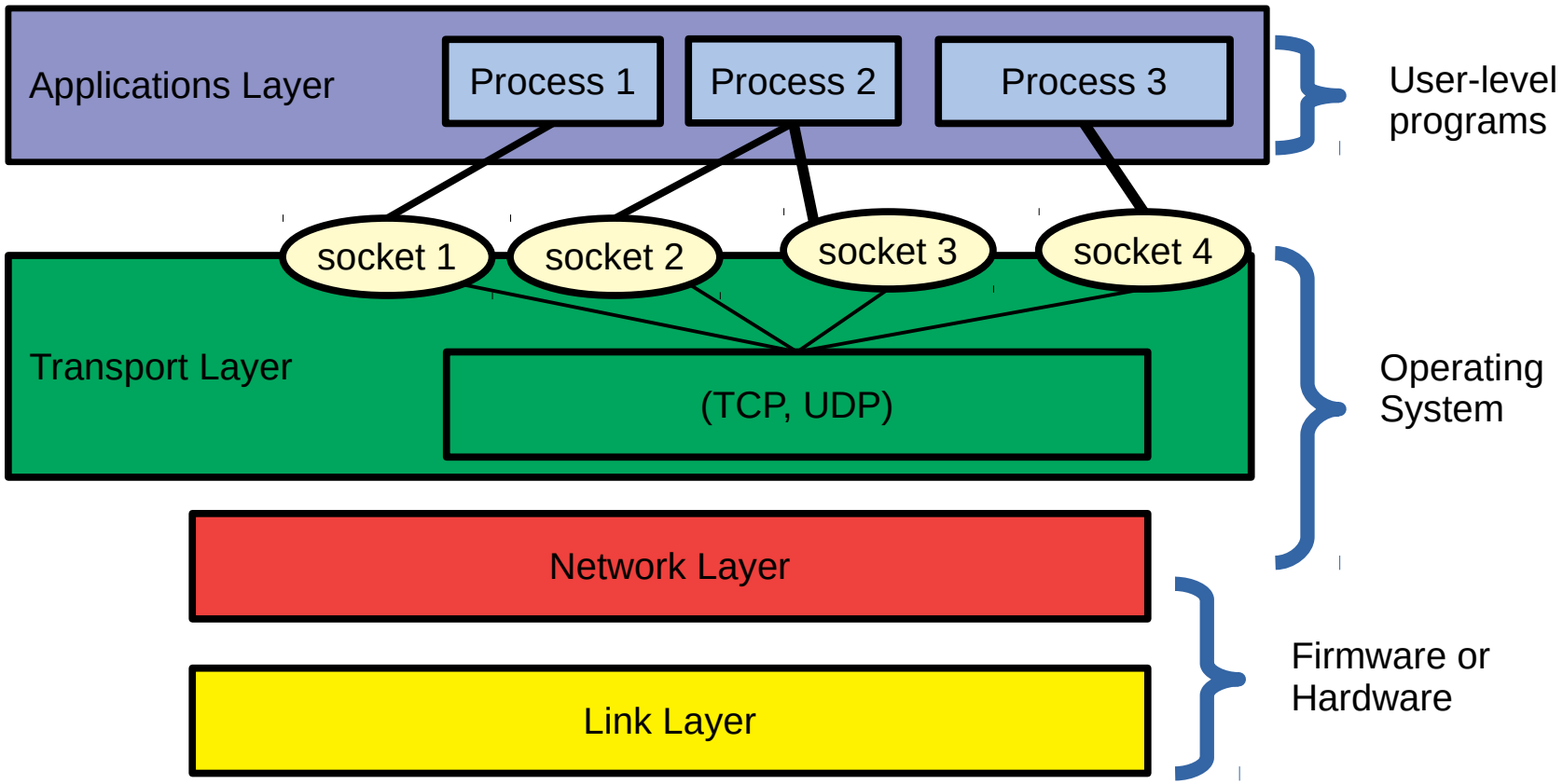
communication stream
(bi-directional Byte stream)

Sockets

- **But what IS a socket?**
  - An object in memory: “state” variables (memory) + some functions such as read(), write()
  - Implemented in the Operating System

Sockets

- Sockets serve as an Interface between Applications and the Transport layer



Addressing

- **To uniquely identify a receiving socket, the sender needs to specify within packet headers..**
  - **Dest IP address** (uniquely identifies a host)
  - **Dest Port number** (uniquely identifies a socket within the destination host)

Port Numbers

- **Port number:** a 16-bit identifier for a Socket
- 0-1023 are “well-known” port numbers, reserved for well-known server-side applications. Examples:
  - 80: HTTP
  - 53: DNS
  - 443: HTTPS
- Higher-numbered ports are available for general use by applications (typically clients) and are known as ephemeral ports.

See: https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

PORT SCANNNG

Port scanning tools (such as `nmap`) can be used for inferring what applications are running on a remote host
Used by admins as well as attackers.

https://en.wikipedia.org/wiki/Port_scanner

SS and NETSTAT

Explore these linux commands for viewing sockets and open ports on your computer.

- **A Program**
- **A Process**
- **An Application**

So which of these are the end-points of communication for applications?

The end-points of communication are **Sockets.**

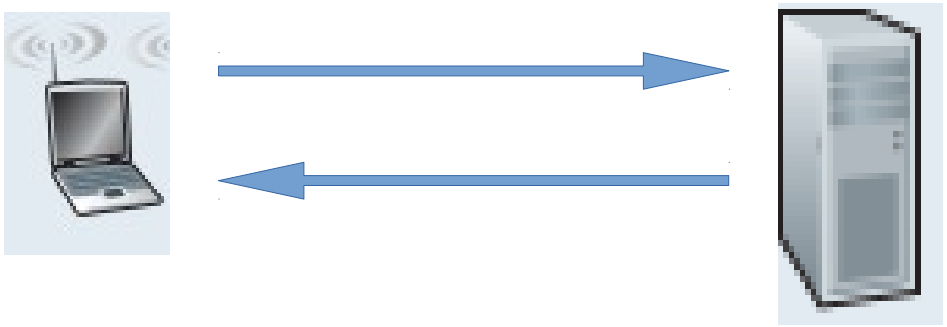
Q: So **how** can I create applications using Sockets?

Q: So does every open Socket on your computer have a unique 16-bit identifier (port number)?

Q: When a packet arrives at your computer, how is it re-directed to the right socket?

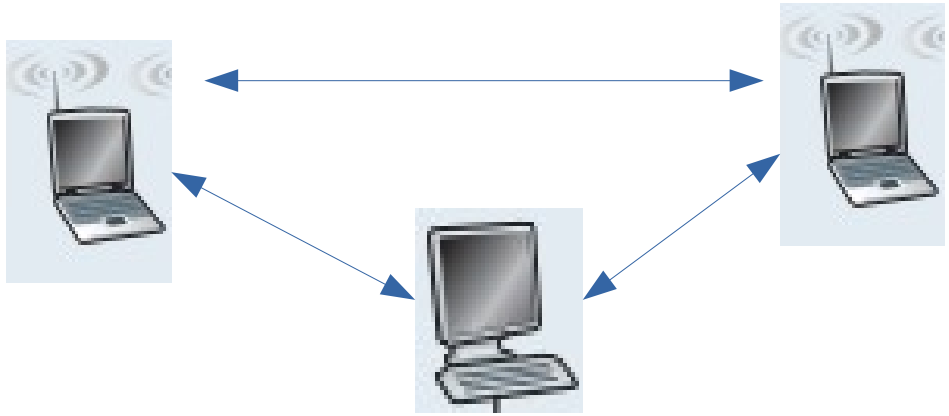
- Does it depend on the type of packet (TCP/UDP)?

Client-Server Architecture



- **Server:** always ON, listening for requests.
  - Has a fixed, well-known IP address, fixed port number.
- **Client:** initiates communication with the Server, makes requests
  - Need not have a well-known IP address
- Example: Web-Browser (Client) and Web-Server

Peer-to-Peer (P2P) Architecture



- **Minimal (or no) reliance on “always-ON” servers**
- Direct communication between pairs of hosts (that may have intermittent connectivity)
- Examples: BitTorrent, Skype

Client and Server Processes

In the context of communicating processes ...

- **Client:** The process that initiates the communication.
  - Client needs to know the address of the server to contact it
- **Server:** The process that waits to be contacted to begin the communication session

Service Models

- **Applications Layer**

  Concerned with what data to send, what to do with received data.

- **Transport Layer:** Process-to-process message delivery service for applications, which can be reliable and in-order (**TCP**) or best-effort (**UDP**)

  TCP: "Give me a message (as a sequence of Bytes) and a <dest IP, dest port> address, and I will deliver them reliably and in-order, to the correct process (socket) at the destination host."

  UDP: "Give me a message and a <dest IP, dest port> address, and I will make a best-effort delivery of that message as a single packet to the correct process (socket) at the destination.

- **Network Layer:** Best-effort host-to-host delivery of a single datagram

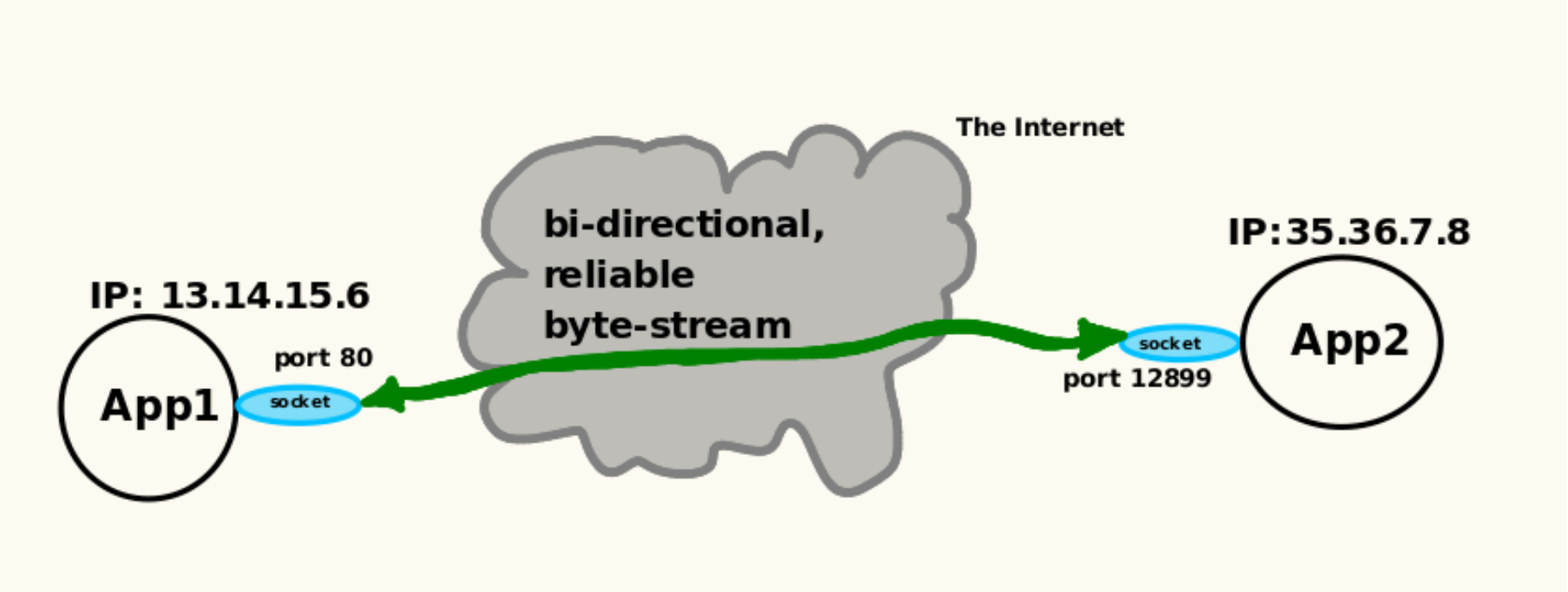
  NW layer: "Give me a payload and a destination IP address. I will create a single datagram containing the payload and make the best effort to deliver it to the destination, but can offer no guarantee of delivery (the packet can get corrupted or lost)."

The Transport Layer

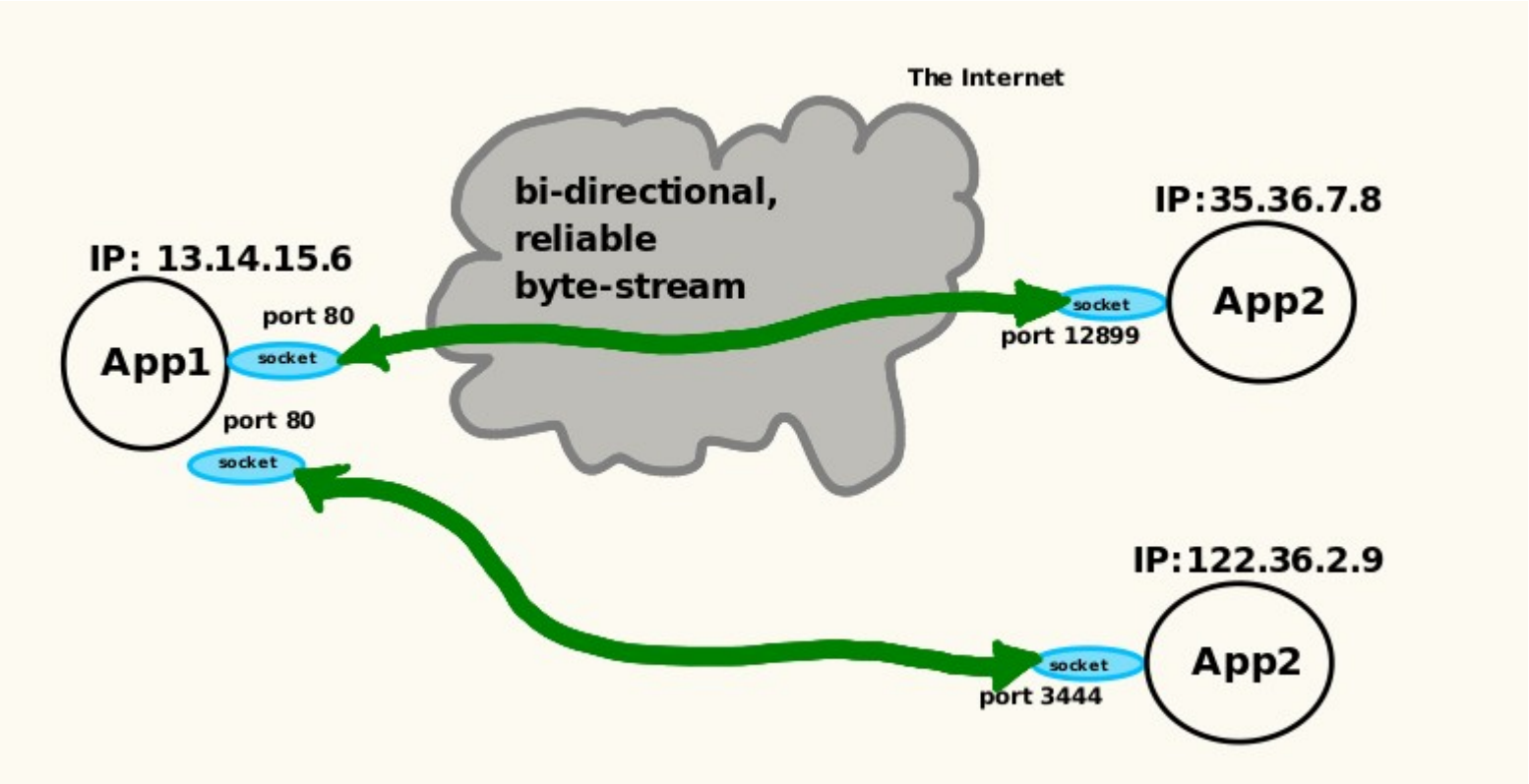
- **TCP is connection oriented**

- **UDP is connection-less**

Applications communicating using a reliable delivery service (TCP)



Applications communicating using a reliable delivery service (TCP)



Applications communicating using a reliable delivery service (TCP)

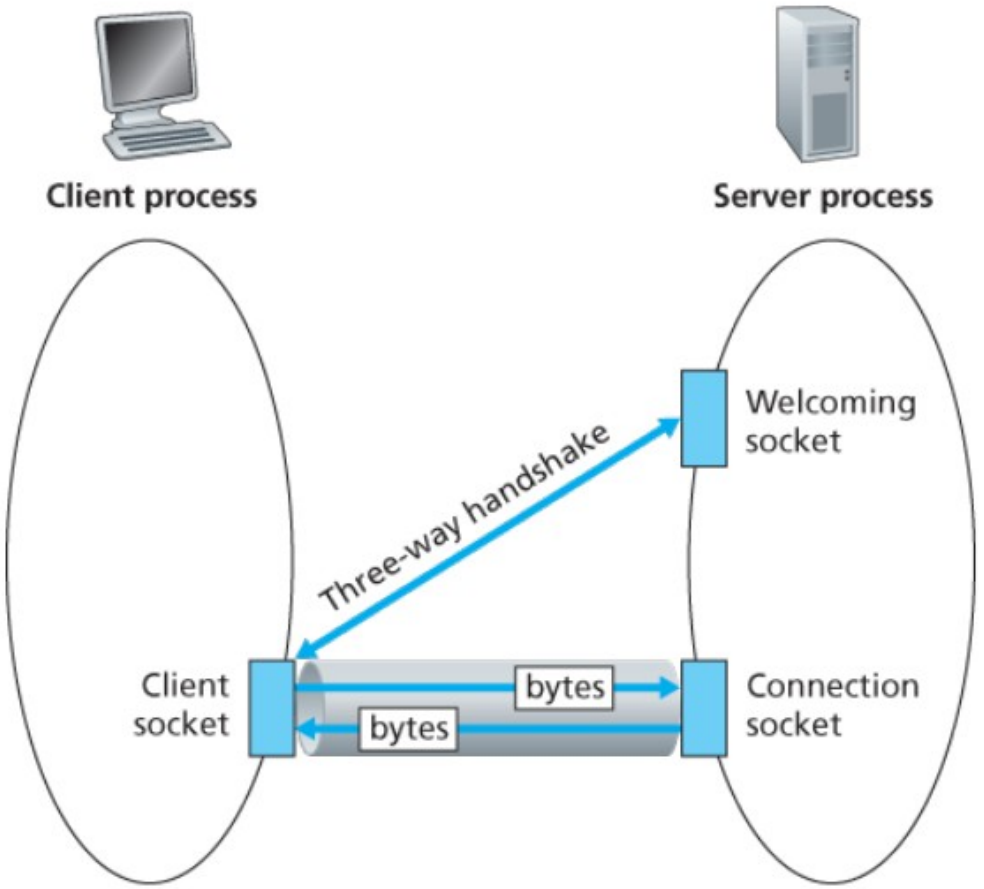
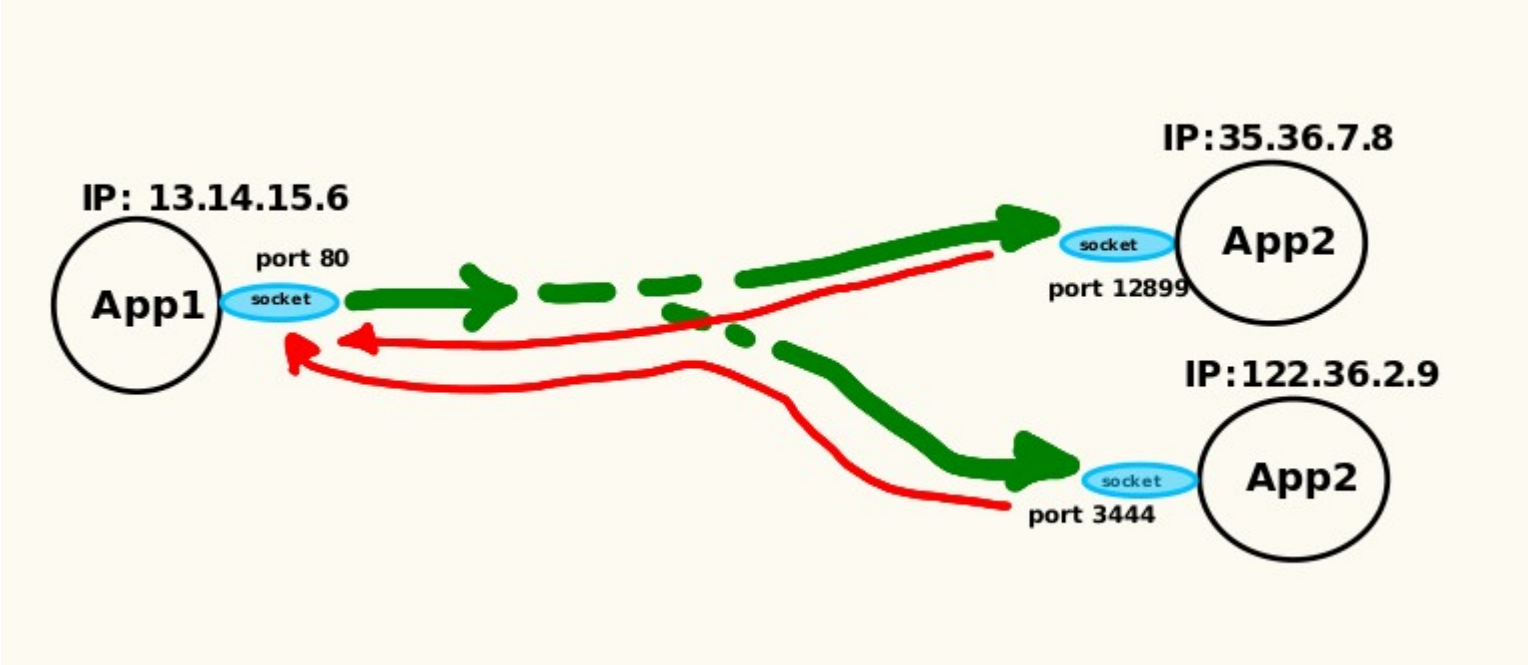


Figure 2.28 *The TCPServer* process has two sockets

Applications communicating using UDP



Q: So HOW can I create applications using Sockets?

Types of Sockets

- **SOCK_STREAM** (Uses **TCP**)
  - Reliable, in-order delivery
  - **Connection-oriented**
  - Congestion-controlled
- **SOCK_DGRAM** (Uses **UDP**)
  - Best-effort (unreliable, not guaranteed to be in-order)
  - **Connection-less**
  - No congestion control

A destination socket on the receiving host is uniquely identified by:

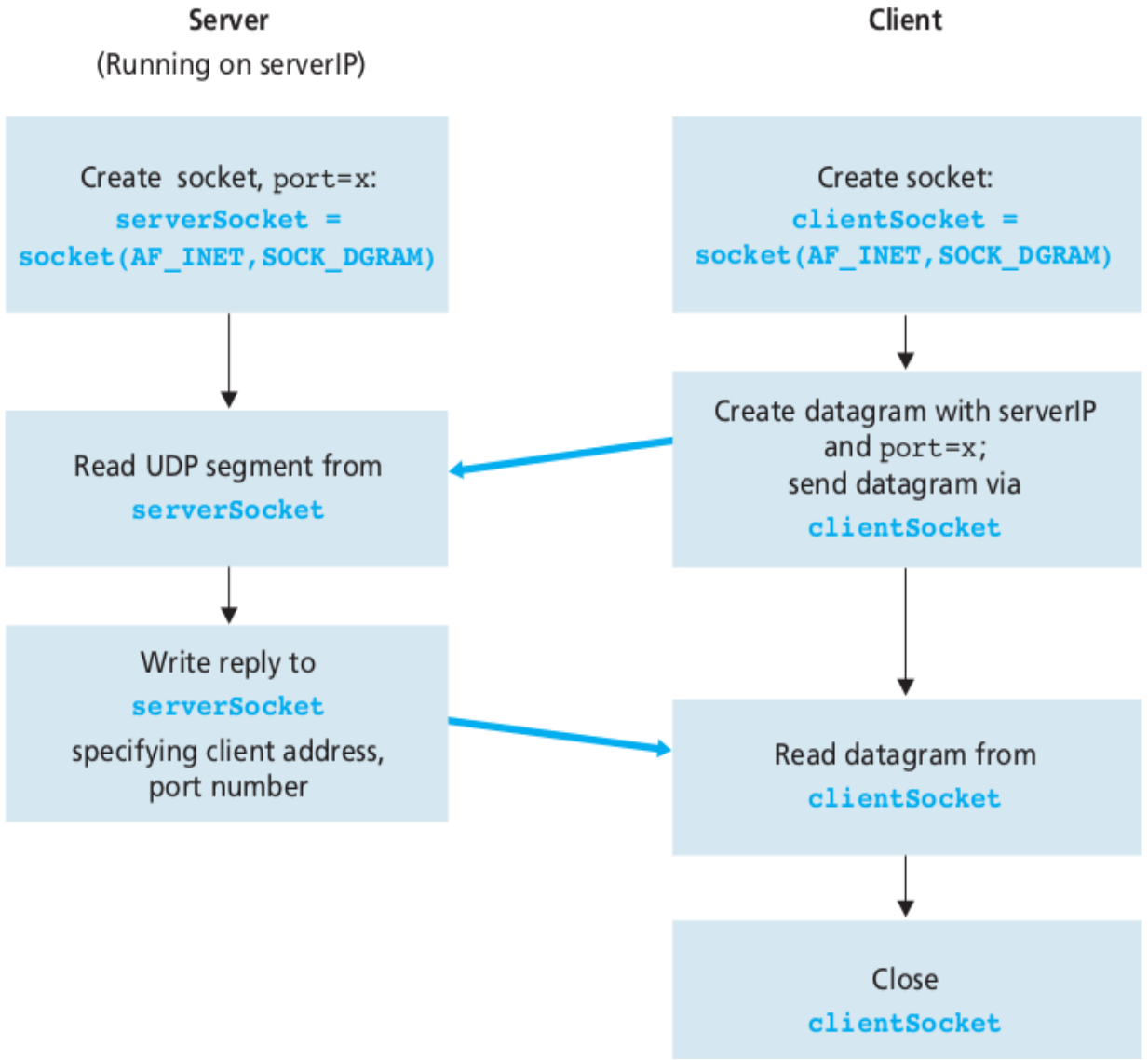
<dest Port, Src IP, Src Port>

A destination socket on the receiving host is uniquely identified by:

<dest Port>

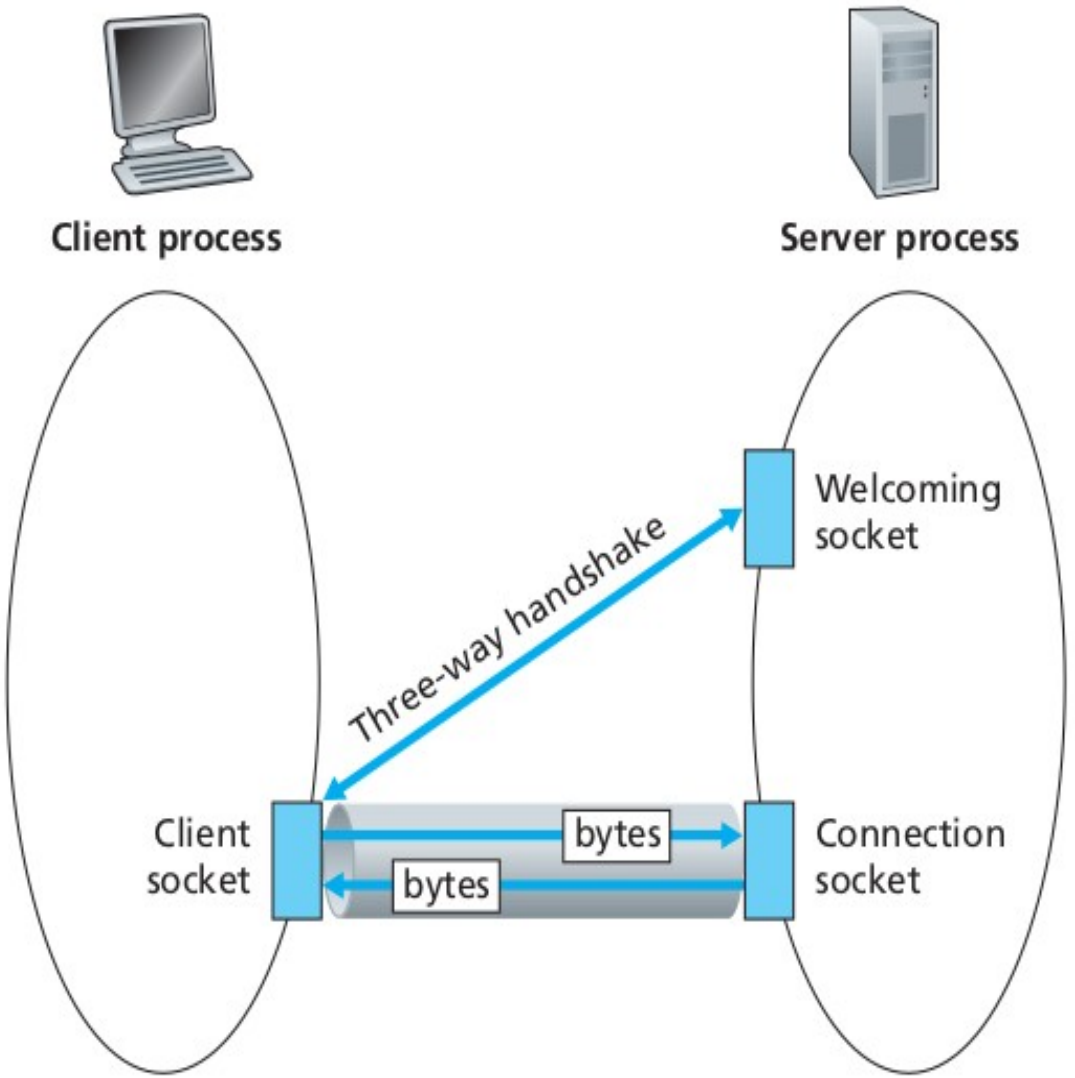
UDP Sockets

UDP Sockets

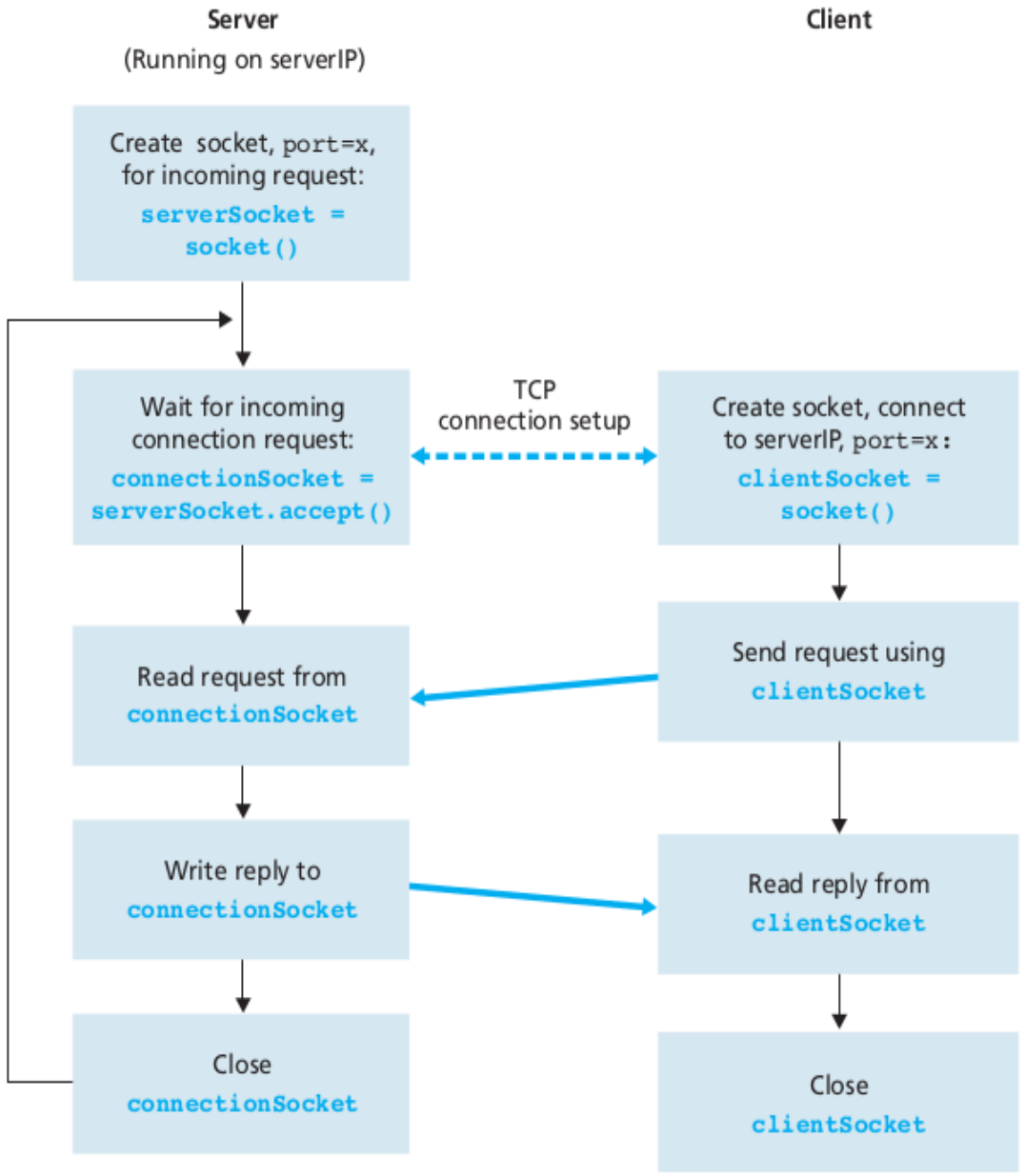

Server
(Running on serverIP)
Client
Create socket, port=x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)
Create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)
Create datagram with serverIP
and port=x;
send datagram via
clientSocket
Read UDP segment from
serverSocket
Write reply to
serverSocket
specifying client address,
port number
Read datagram from
clientSocket
Close
clientSocket

TCP Sockets

TCP Sockets


Client process
Server process
Welcoming socket
Three-way handshake
Client socket
bytes
bytes
Connection socket

TCP Sockets

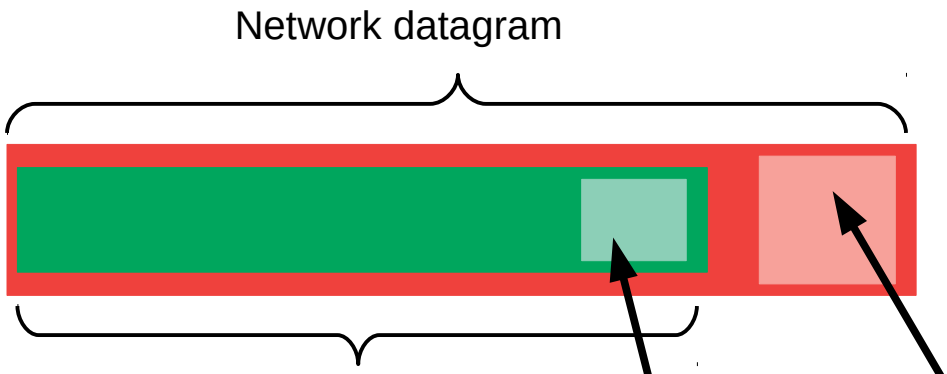

Server
(Running on serverIP)
Client
Create socket, port=x, for incoming request:
serverSocket = socket()
Wait for incoming connection request:
connectionSocket = serverSocket.accept()
TCP connection setup
Create socket, connect to serverIP, port=x:
clientSocket = socket()
Read request from connectionSocket
Send request using clientSocket
Write reply to connectionSocket
Read reply from clientSocket
Close connectionSocket
Close clientSocket

Q: So does every open Socket on your computer have a unique 16-bit identifier (port number)?

Q: When a packet arrives at your computer, how is it re-directed to the right socket?

- Does it depend on the type of packet (TCP/UDP)?

Multiplexing and Demultiplexing



Network-layer header
- src IP
- dest IP
- protocol that the transport layer segment is meant for (TCP/UDP etc)
- ...

Transport-layer header
- src port number
- dest port number

IPv4 Datagram Format

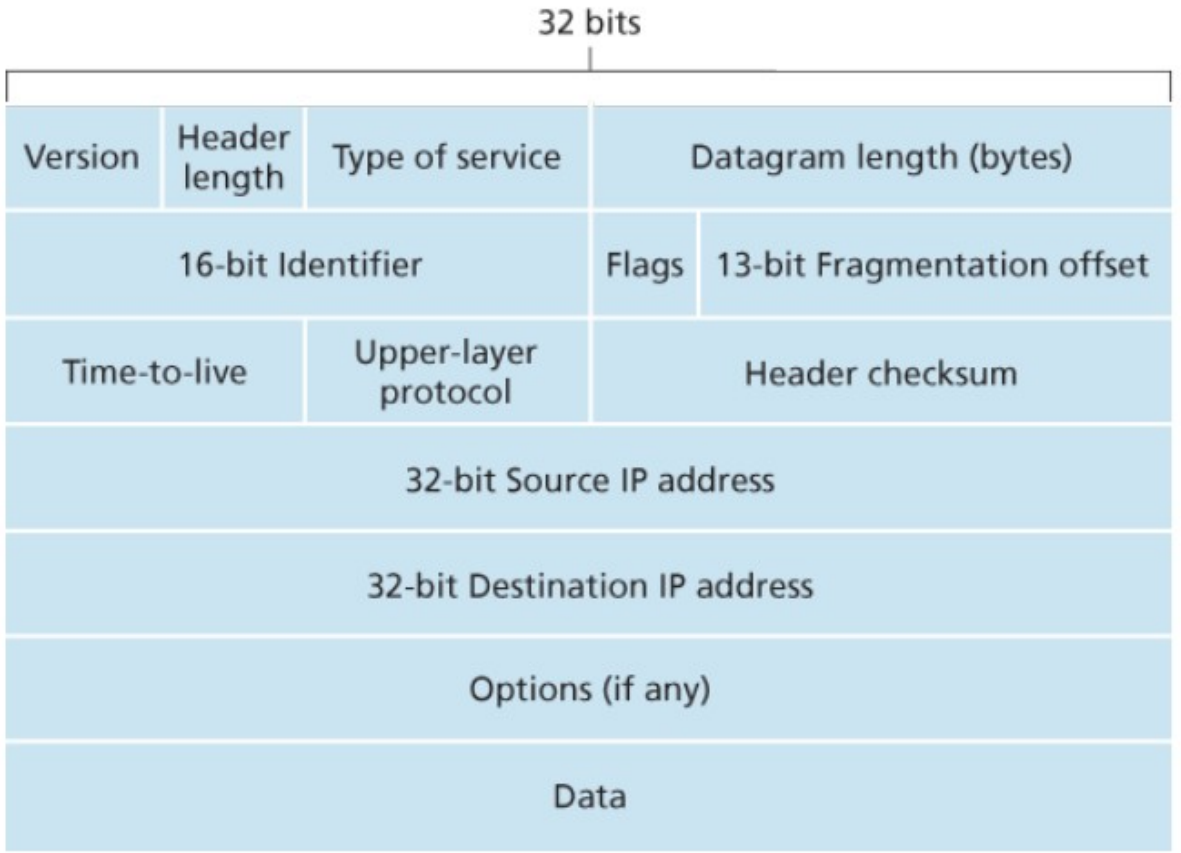
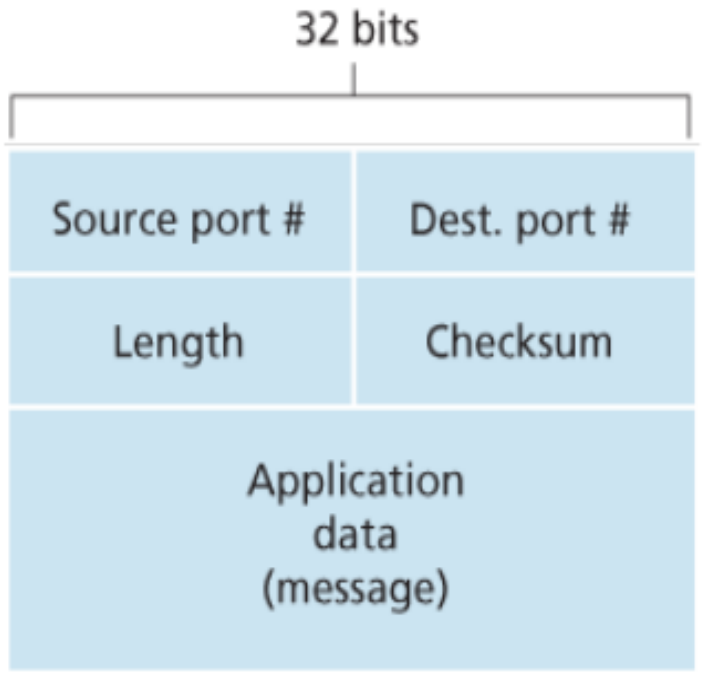


Figure 4.16 IPv4 datagram format

UDP segment format (header is 8 Bytes long)



TCP Segment Format

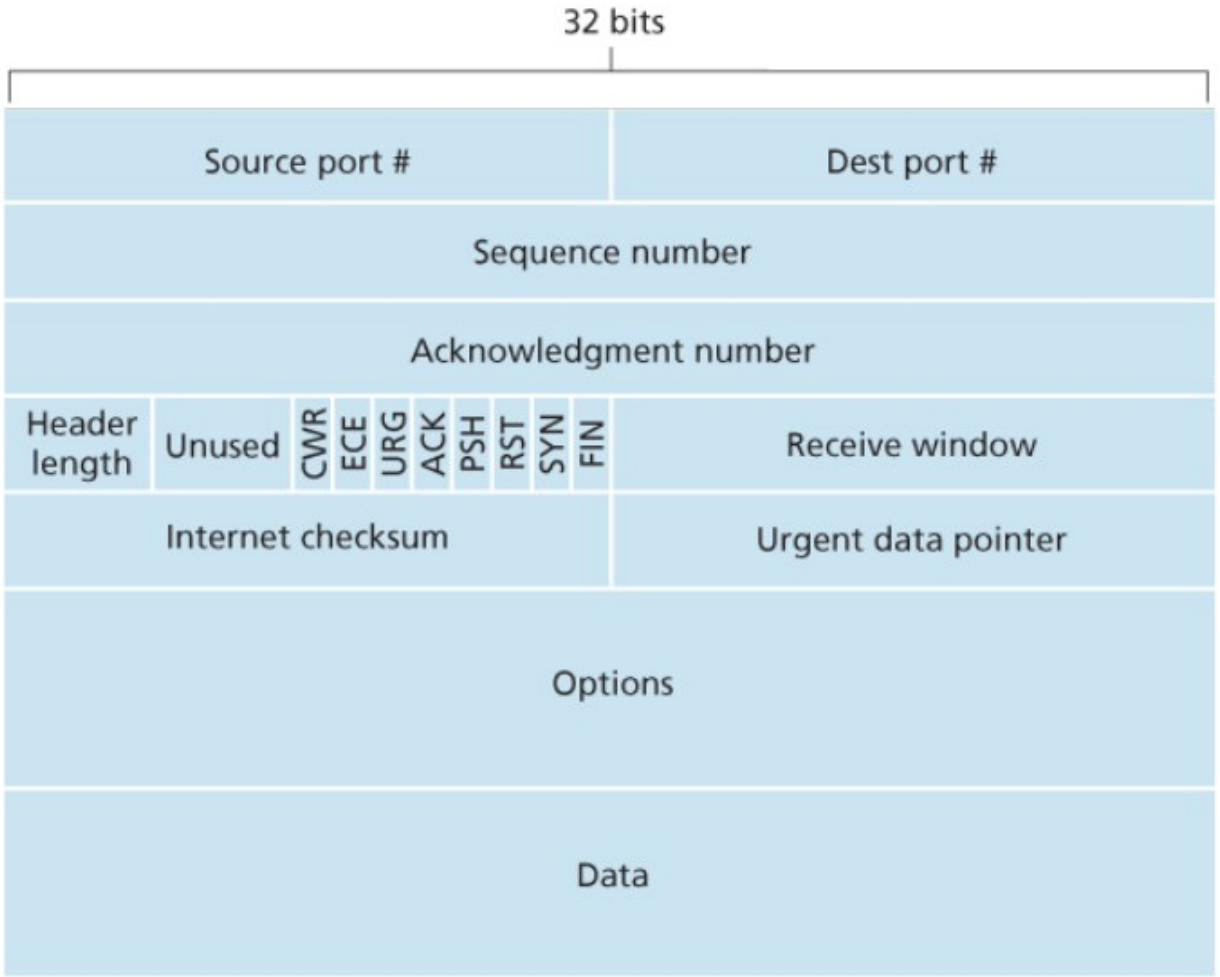
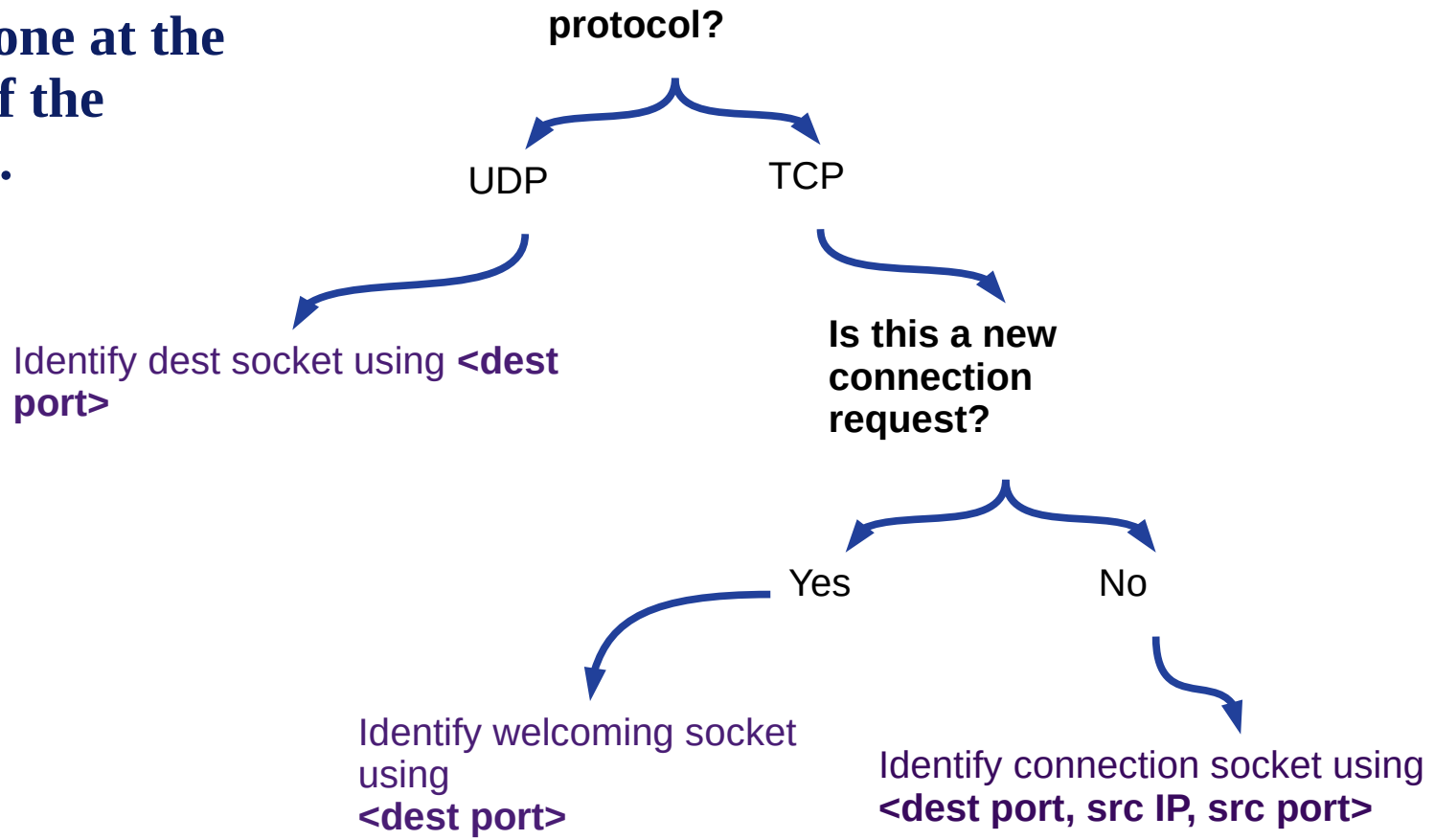


Figure 3.29 TCP segment structure

Multiplexing and Demultiplexing

Demultiplexing done at the Transport layer of the destination host ...



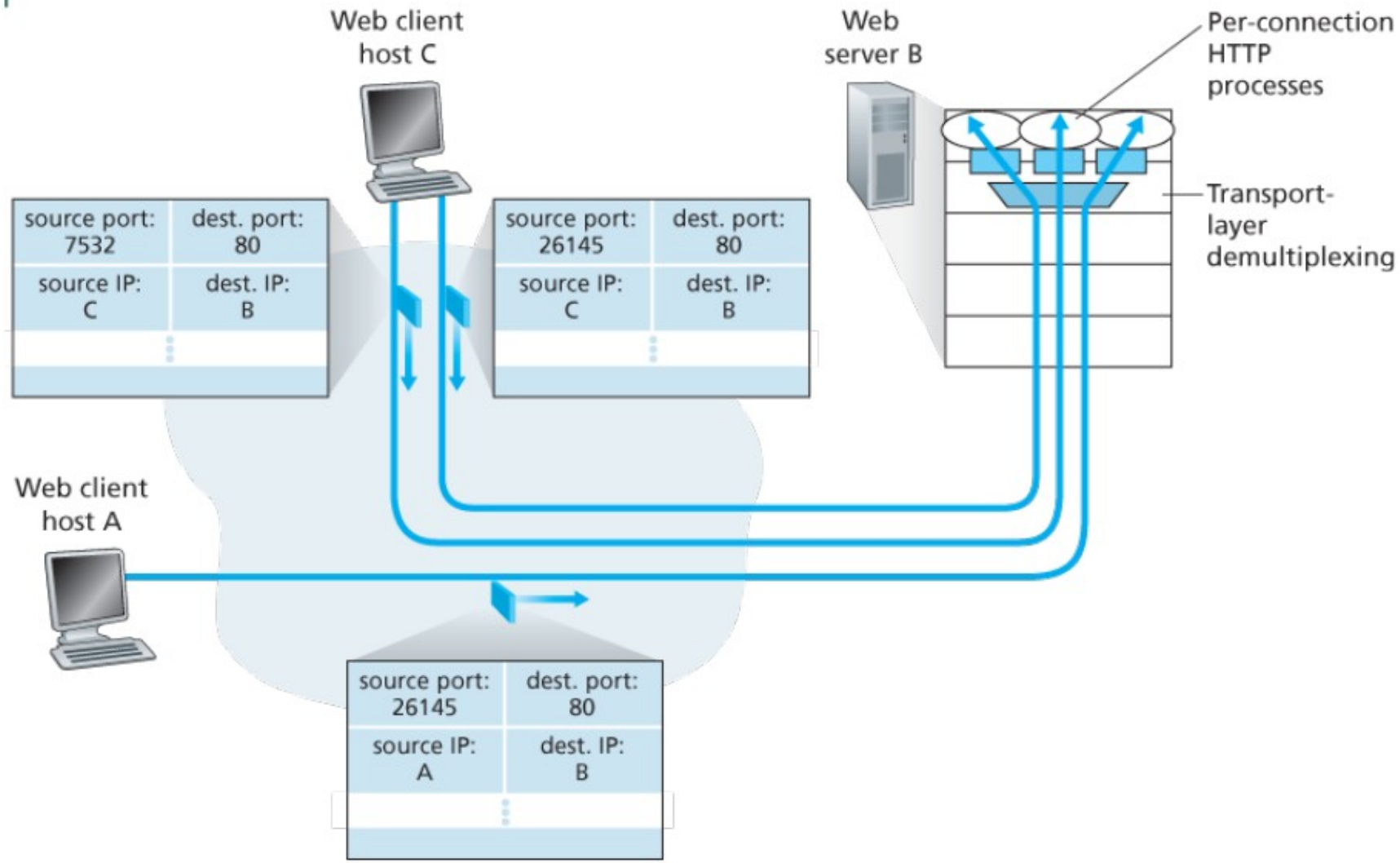


Figure 3.5 Two clients, using the same destination port number (80) to communicate with the same Web server application

Reference and Reading Assignment

- Kurose and Ross 6th ed
  - Section 2.1: Principles of Networked Applications
  - Section 2.7: Socket Programming
- Tutorial on Socket Programming using Python:

  https://realpython.com/python-sockets/

- **Using the Sockets interface and the services provided by the layers below, how can we design/re-create some popular Internet Applications?**

  - A Chat Room
  - Remote Login
  - Email
  - Peer-to-Peer File Sharing
  - **The Web**
  - Multiplayer online games

    ....

Questions coming up later in this course..

- How should the applications interpret the Byte stream?
  - Application level protocols: Stateful vs Stateless
- What is the Web? What does it consist of?
- DNS: How are names/urls translated to IP addresses?