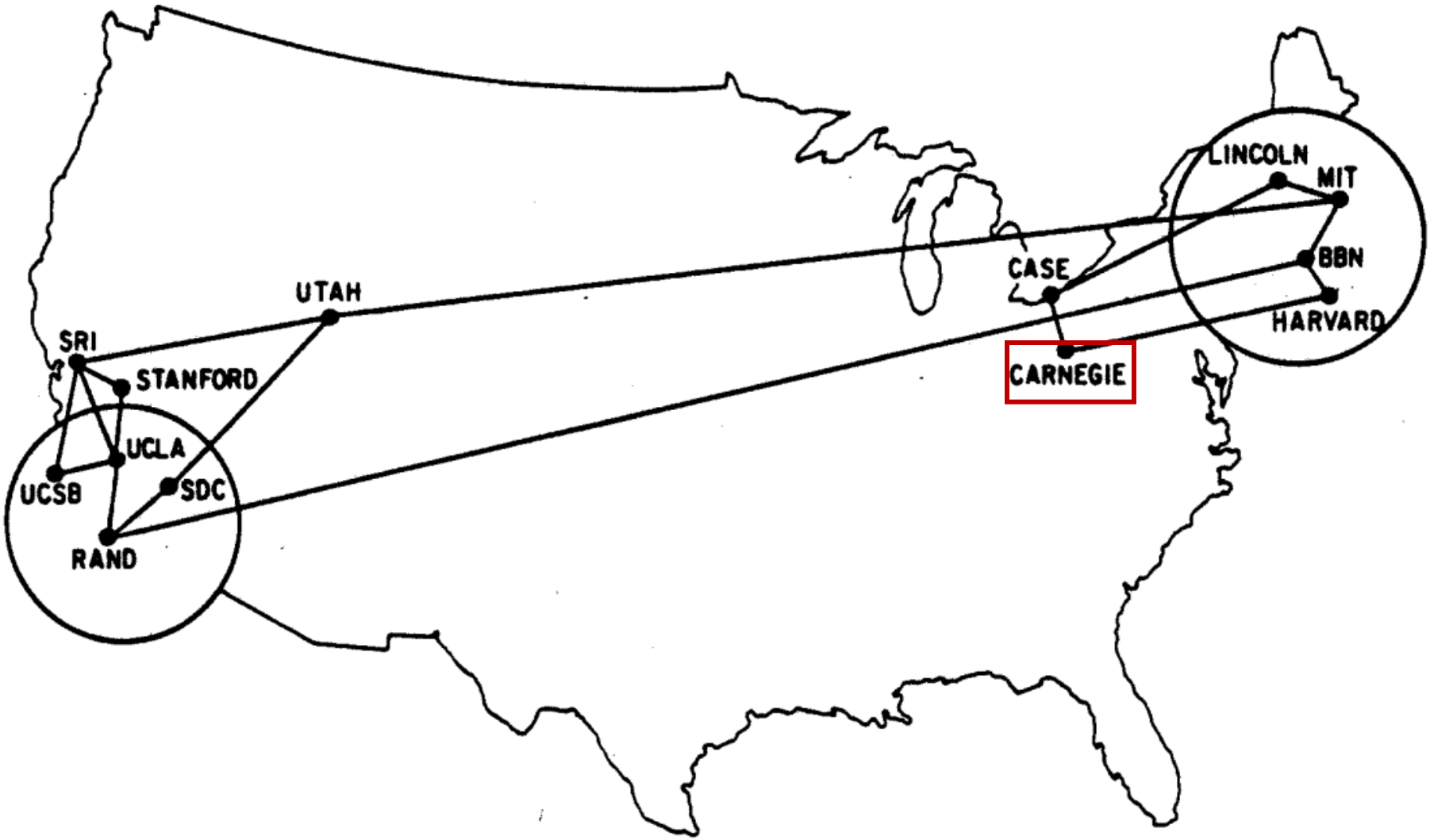

WELCOME
TO
15-213
14-513
18-213
18-613
15-513

Network Programming

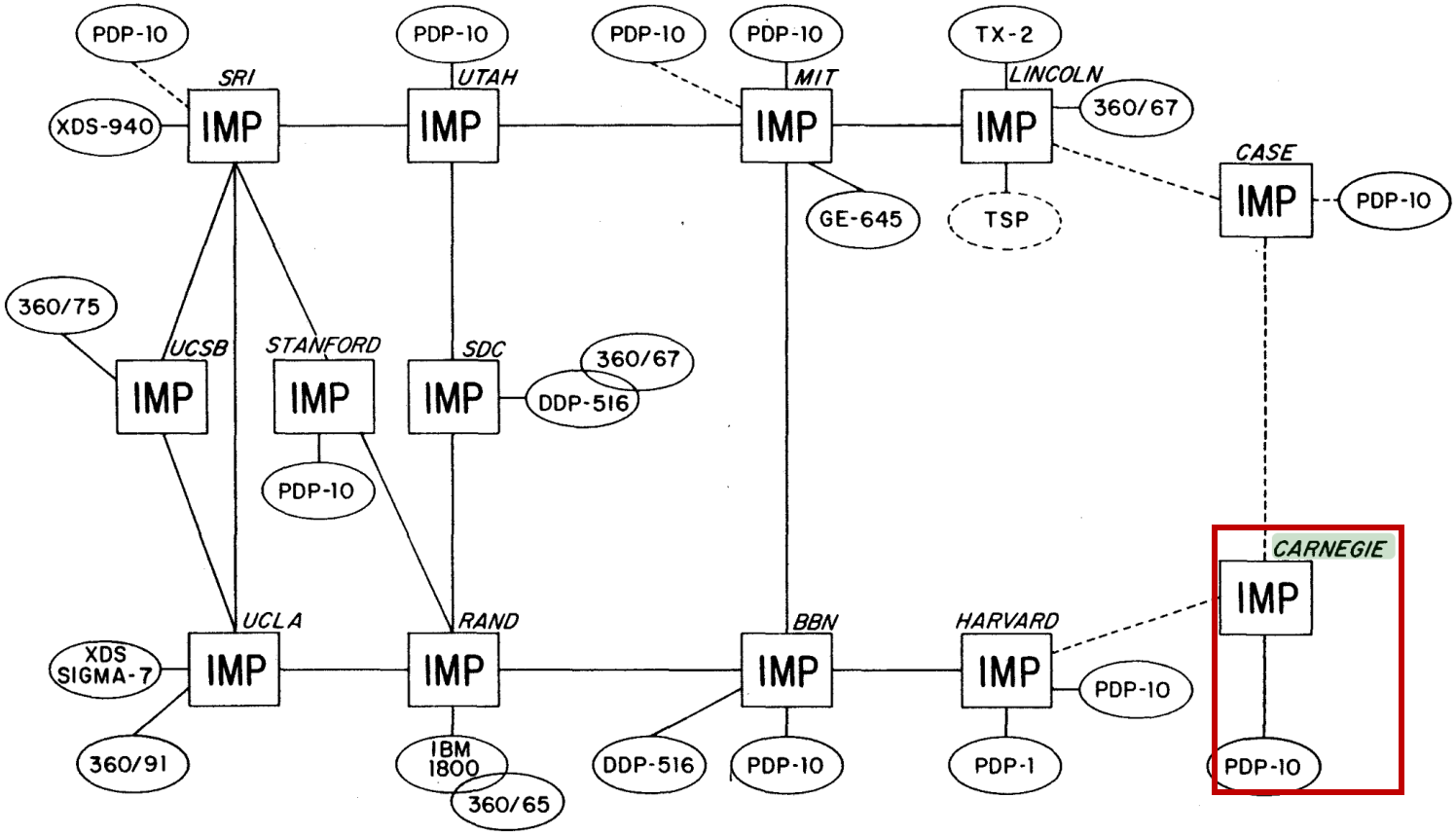
15-213/15-513/14-513: Introduction to Computer Systems
19st Lecture, November 10, 2026



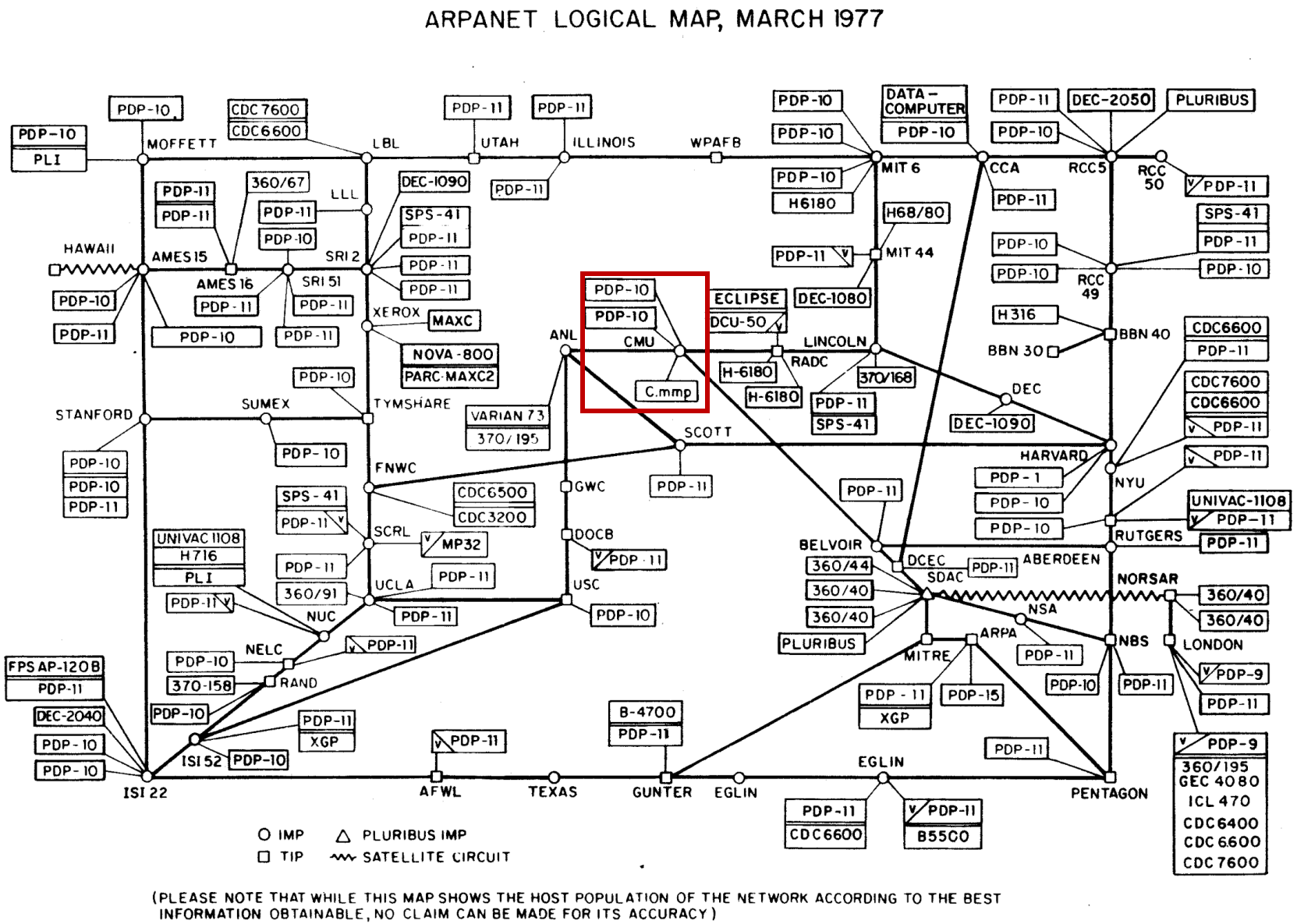
The ARPANET in December 1969



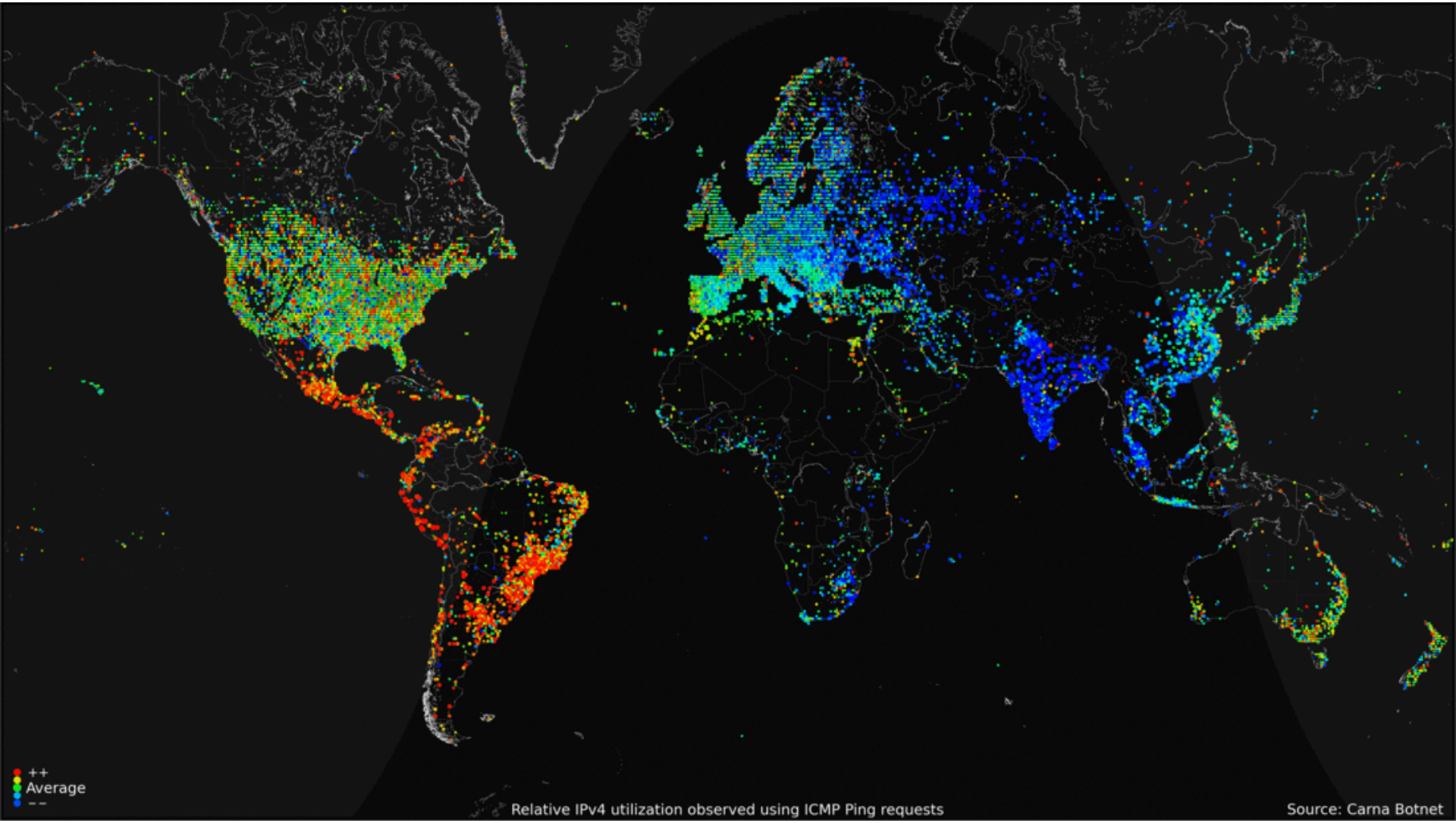
The ARPANET in December 1970



ARPA NET, DECEMBER 1970


ARPANET LOGICAL MAP, MARCH 1977
PDP-10
PLI
MOFFETT
CDC 7600
CDC6600
LBL
PDP-11
UTAH
PDP-11
ILLINOIS
WPAFB
PDP-10
PDP-10
PDP-10
H6180
DATA-COMPUTER
PDP-10
MIT 6
PDP-11
DEC-2050
PLURIBUS
PDP-10
CCA
RCC5
RCC 50
PDP-11
PDP-11
PDP-11
360/67
LLL
DEC-1090
PDP-11
SPS-41
PDP-11
PDP-11
PDP-10
H68/80
MIT 44
PDP-11
SPS-41
PDP-11
HAWAII
AMES15
SRI 2
PDP-10
PDP-10
RCC 49
PDP-10
AMES 16
SRI 51
PDP-11
PDP-11
PDP-10
ECLIPSE
DCU-50
DEC-1080
H316
BBN 40
CDC6600
PDP-11
PDP-10
PDP-11
XEROX
MAXC
NOVA-800
PARC-MAXC2
ANL
CMU
C.mmp
LINCOLN
RADC
H-6180
H-6180
370/168
BBN 30
CDC7600
CDC6600
PDP-11
PDP-10
TYMSHARE
VARIAN 73
370/195
PDP-11
SPS-41
DEC
DEC-1090
STANFORD
SUMEX
PDP-10
SCOTT
HARVARD
PDP-1
PDP-10
PDP-10
PDP-10
PDP-10
PDP-11
FNWC
CDC6500
CDC3200
GWC
PDP-11
PDP-11
NYU
UNIVAC-1108
PDP-11
SPS-41
PDP-11
SCRL
MP32
DOCB
PDP-11
RUTGERS
PDP-11
UNIVAC 1108
H716
PLI
PDP-11
360/91
PDP-11
UCLA
PDP-11
USC
PDP-10
BELVOIR
360/44
360/40
360/40
DCEC
SDAC
PDP-11
ABERDEEN
NORSAR
360/40
360/40
NUC
NSA
PLURIBUS
ARPA
MITRE
PDP-11
NBS
LONDON
PDP-9
PDP-11
NELC
PDP-11
PDP-10
370-158
RAND
PDP-10
PDP-11
PDP-10
PDP-11
XGP
PDP-15
PDP-11
PDP-9
360/195
GEC 4080
ICL 470
CDC6400
CDC 6600
CDC 7600
FPSAP-120B
PDP-11
DEC-2040
PDP-10
PDP-10
ISI52
PDP-10
ISI 22
PDP-11
AFWL
TEXAS
B-4700
PDP-11
GUNTER
EGLIN
EGLIN
PDP-11
CDC6600
PDP-11
B5500
PENTAGON
O IMP
△ PLURIBUS IMP
□ TIP
SATELLITE CIRCUIT
(PLEASE NOTE THAT WHILE THIS MAP SHOWS THE HOST POPULATION OF THE NETWORK ACCORDING TO THE BEST INFORMATION OBTAINABLE, NO CLAIM CAN BE MADE FOR ITS ACCURACY)

A Map of 460 Billion Device Connections to the Internet collected by the Carna Botnet



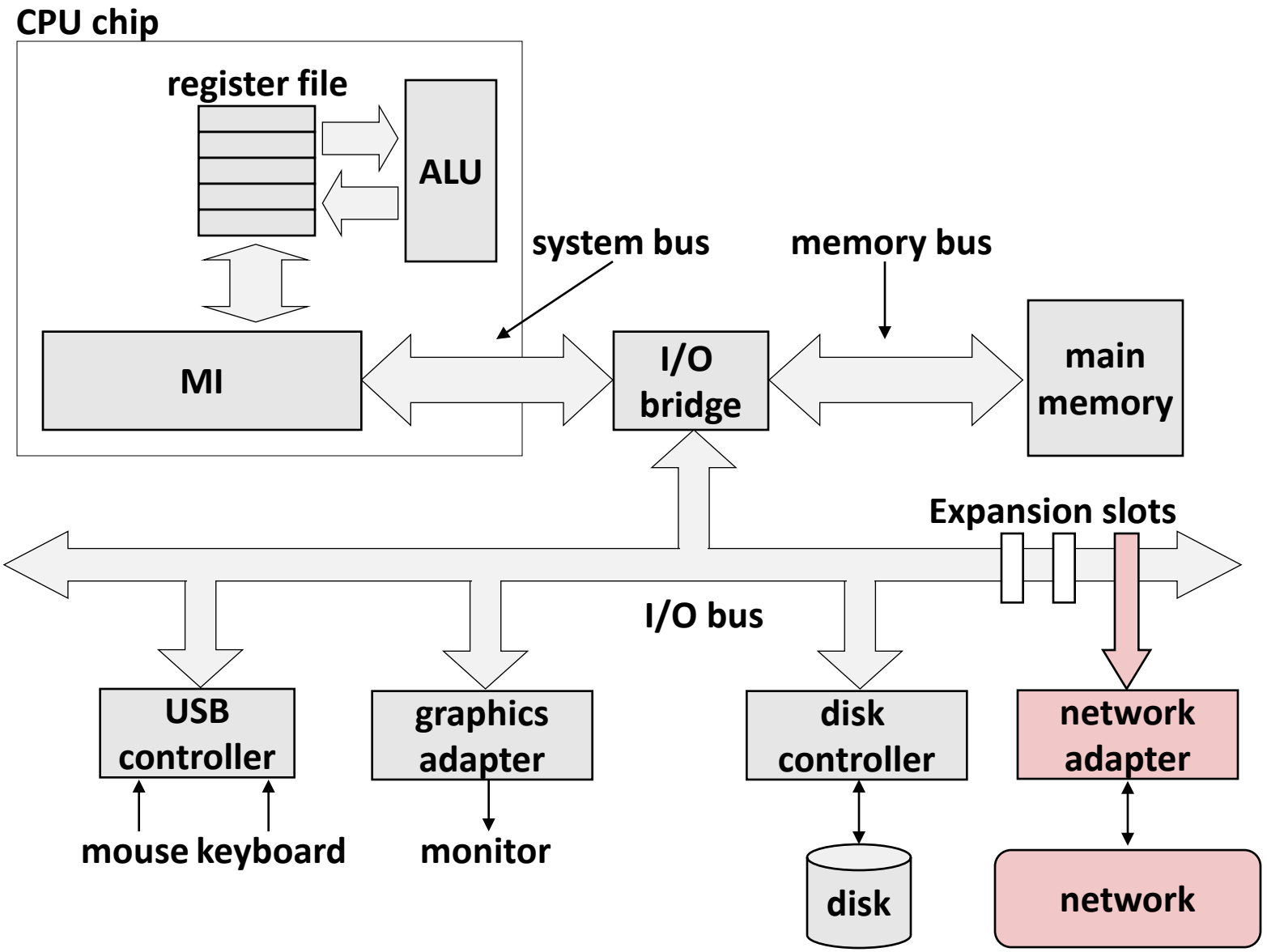
Outline

- Network types and structures
- Locating a host
- Setting up a connection
- HTTP Example

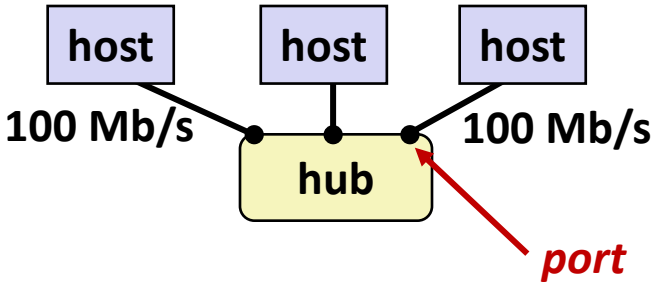
Computer Networks

- **A *network* is a hierarchical system of boxes and wires organized by geographical proximity**
  - SAN (System Area Network) spans a rack or a room
  - LAN (Local Area Network) spans a building or campus
    - Ethernet is most prominent example
  - WAN (Wide Area Network) spans country or world
    - Typically high-speed point-to-point (mostly optical) links
  - Also: SAN (Storage area network), MAN (Metropolitan), etc., etc.
- **An *internetwork* (*internet*) is an interconnected set of networks**
  - The Global IP Internet (uppercase "I") is the most famous example of an internet (lowercase "i")
- **Let's see how an internet is built from the ground up**

Hardware Organization of a Network Host

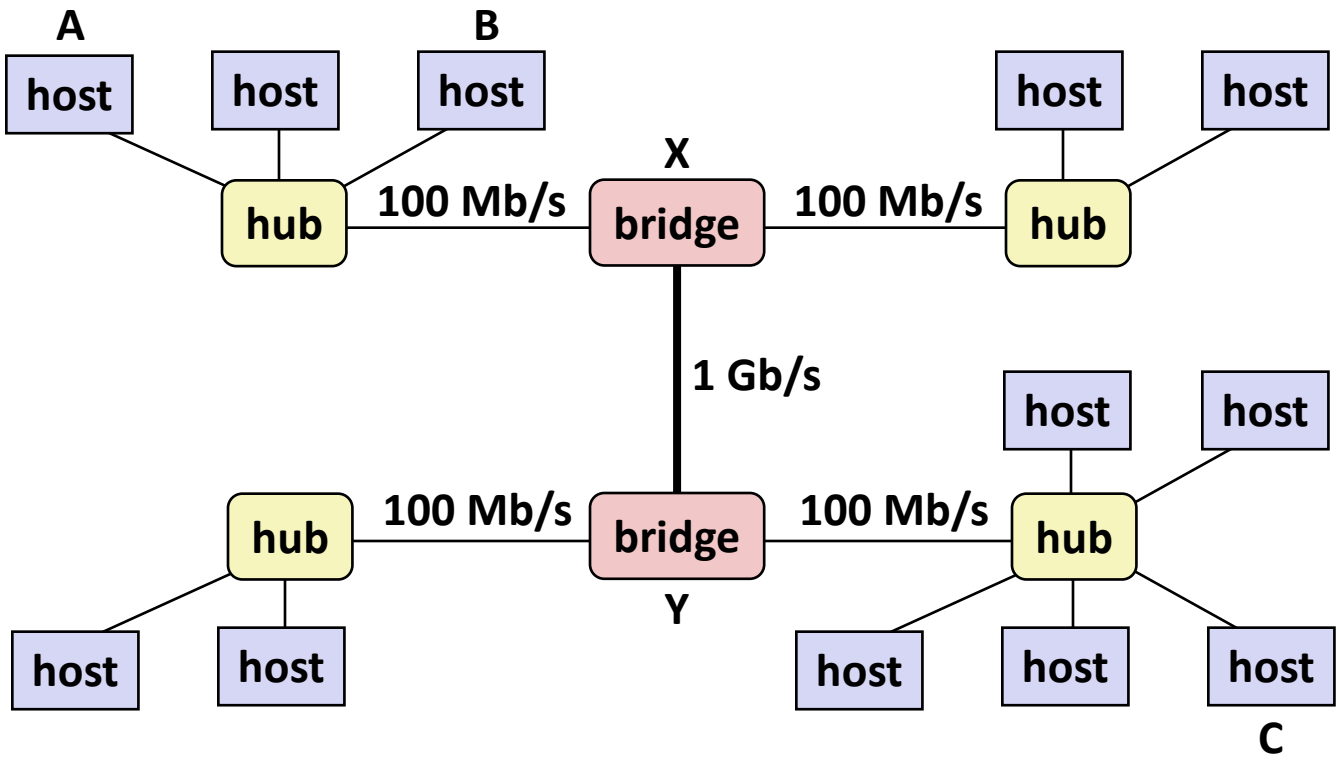


Example Lowest Level: Ethernet



- Ethernet segment consists of a collection of *hosts* connected by wires (twisted pairs) to a *hub*
- Spans room or floor in a building
- Operation
  - Each Ethernet adapter has a unique 48-bit address (MAC address)
    - E.g., 00:16:ea:e3:54:e6

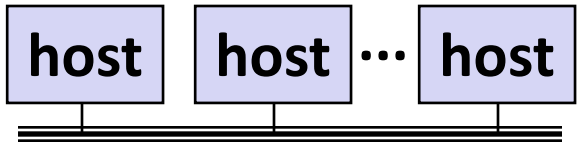
Next Level: Bridged Ethernet Segment



- Spans building or campus
- Bridges cleverly learn which hosts are reachable from which ports and then selectively copy frames from port to port

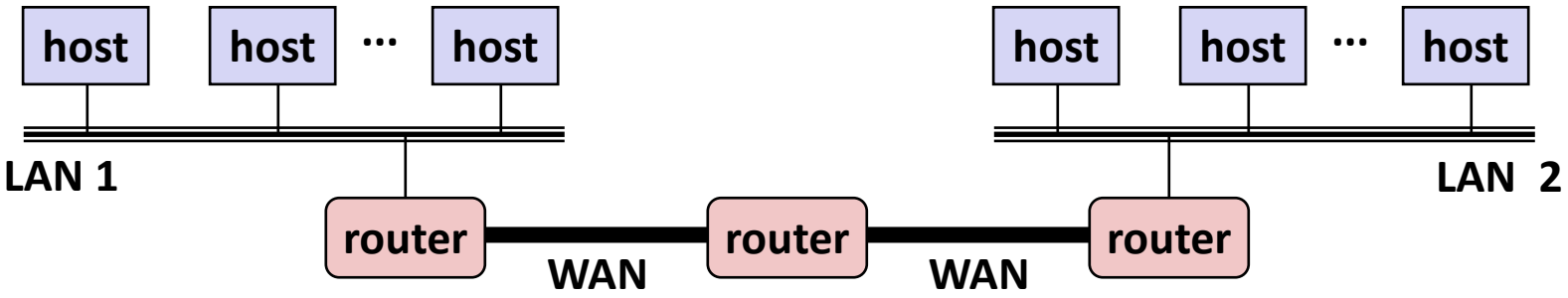
Conceptual View of LANs

- For simplicity, hubs, bridges, and wires are often shown as a collection of hosts attached to a single wire:



Next Level: internets

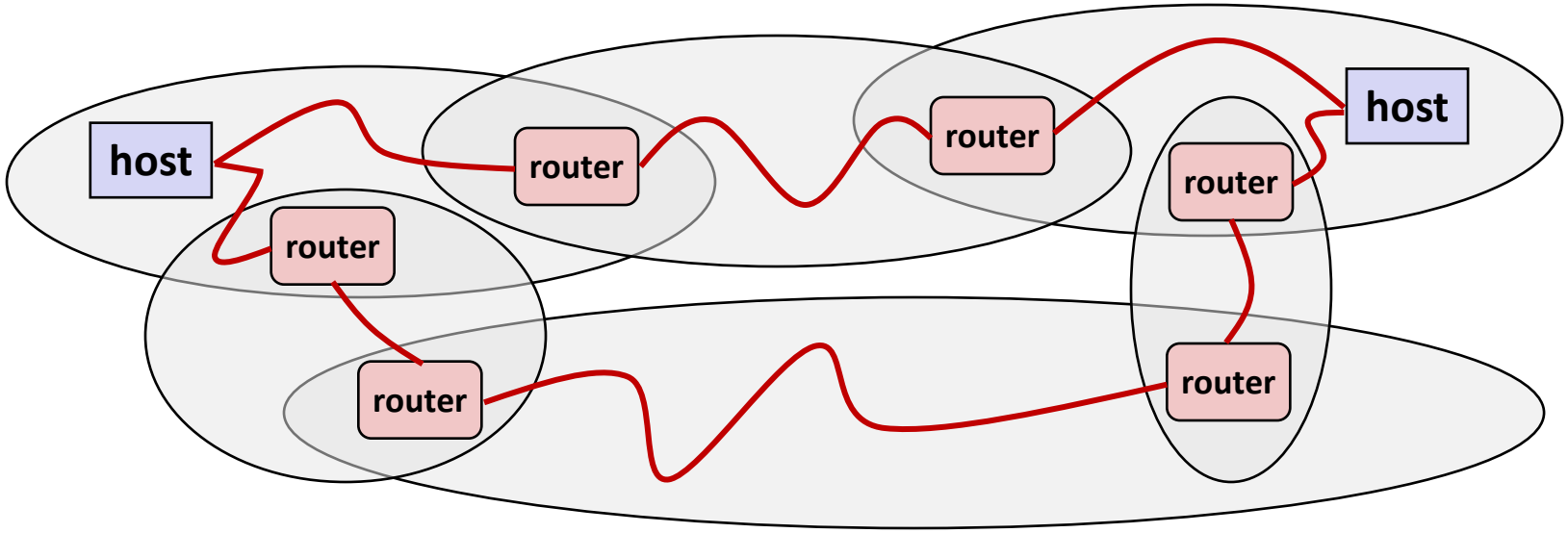
- Multiple incompatible LANs can be physically connected by specialized computers called *routers*
- The connected networks are called an *internet* (lower case)



LAN 1 and LAN 2 might be completely different, totally incompatible

(e.g., Ethernet, Fibre Channel, 802.11, T1-links, DSL, …)*

Logical Structure of an internet



- **Ad hoc interconnection of networks**
  - No particular topology
  - Vastly different router & link capacities
- **Send packets from source to destination by hopping through networks**
  - Router forms bridge from one network to another
  - Different packets may take different routes

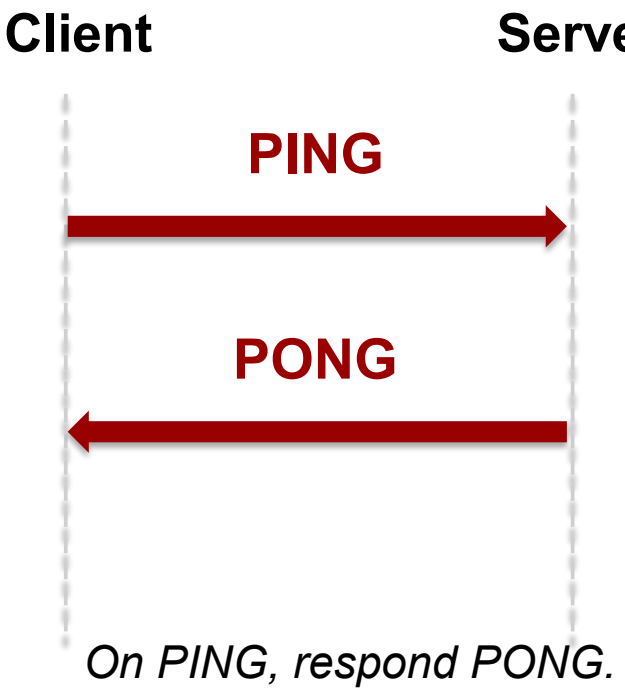
The Notion of an internet Protocol

- **How is it possible to send bits across incompatible LANs and WANs?**

- **Solution: *protocol* software running on each host and router**
  - Protocol is a set of rules that governs how hosts and routers should cooperate when they transfer data from network to network.
  - Smooths out the differences between the different networks

What Is a Protocol?

An agreement about how independent parties communicate.



- **Message format**
  Four ASCII bytes? A newline terminator?
- **Meaning**
  What exactly does PONG acknowledge?
- **Ordering**
  Can another PING be sent before PONG?
- **Failures**
  What if PONG never arrives?

Delivering bytes is not the same as communicating.

What Makes Something a Protocol?

- **Syntax**
  What messages look like.
- **Semantics**
  What the messages mean.
- **Sequencing**
  When each party sends and responds.
- **Error handling**
  What to do when something goes wrong.

What does IP deliberately leave out?

IP is BEST EFFORT

It does NOT provide:

- Guaranteed delivery
- In-order delivery
- Retransmission
- Connection setup

TCP adds reliability above IP when needed.

Protocols at Different Layers

Application	**HTTP · SMTP · DNS · SSH**	Web requests, mail, naming, remote access
Transport	**TCP · UDP**	Process-to-process communication
Internet	**IPv4 · IPv6 · ICMP (control)**	Addressed datagrams, forwarding, error reporting
Link	**Ethernet · Wi-Fi · LTE/5G**	Frame delivery over the next local hop

A common IP layer can connect many applications and many networks.

Why Open, Minimal Protocols Matter

- **Open specifications**
  Anyone can read and implement the standards.
- **Interoperability**
  Independent vendors and systems work together.
- **Abstraction**
  Applications need not know the underlying links.
- **Independent evolution**
  Networks and software can change separately.

The power of doing less

IP only promises to try to forward each packet.

That simple contract works over almost any network.

TCP can add reliability above IP.

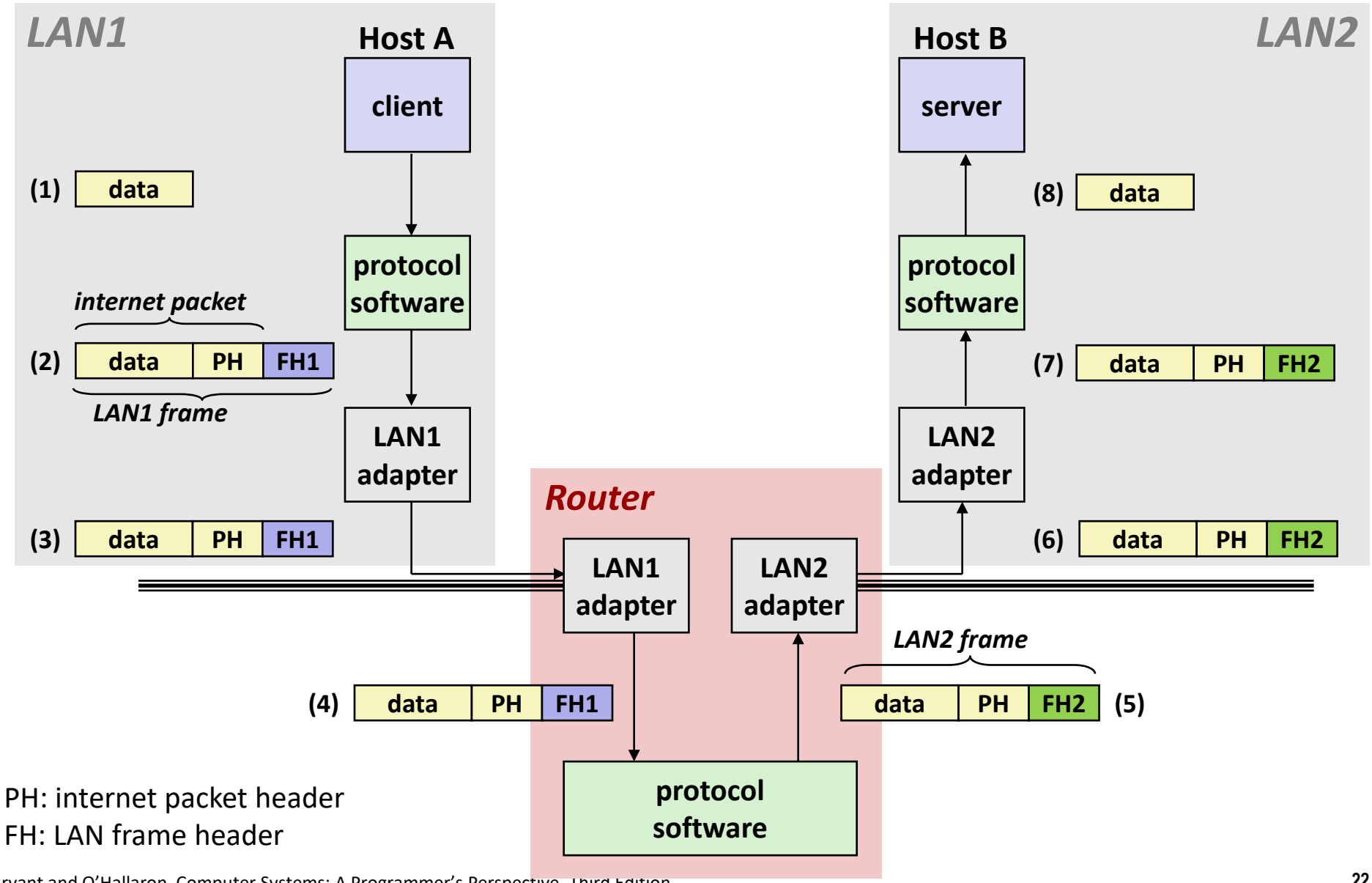
A small, open contract is easier to implement everywhere.

What Does an internet Protocol Do?

- **Provides a *naming scheme***
  - An internet protocol defines a uniform format for ***host addresses***
  - Each host (and router) is assigned at least one of these internet addresses that uniquely identifies it

- **Provides a *delivery mechanism***
  - An internet protocol defines a standard transfer unit (***packet***)
  - Packet consists of ***header*** and ***payload***
    - Header: contains info such as packet size, source and destination addresses
    - Payload: contains data bits sent from source host

Transferring internet Data Via Encapsulation



PH: internet packet header
FH: LAN frame header

Other Issues

- **We are glossing over a number of important questions:**
  - What if different networks have different maximum frame sizes? (segmentation)
  - How do routers know where to forward frames?
  - How are routers informed when the network topology changes?
  - What if packets get lost?

- **These (and other) questions are addressed by the area of systems known as *computer networking***

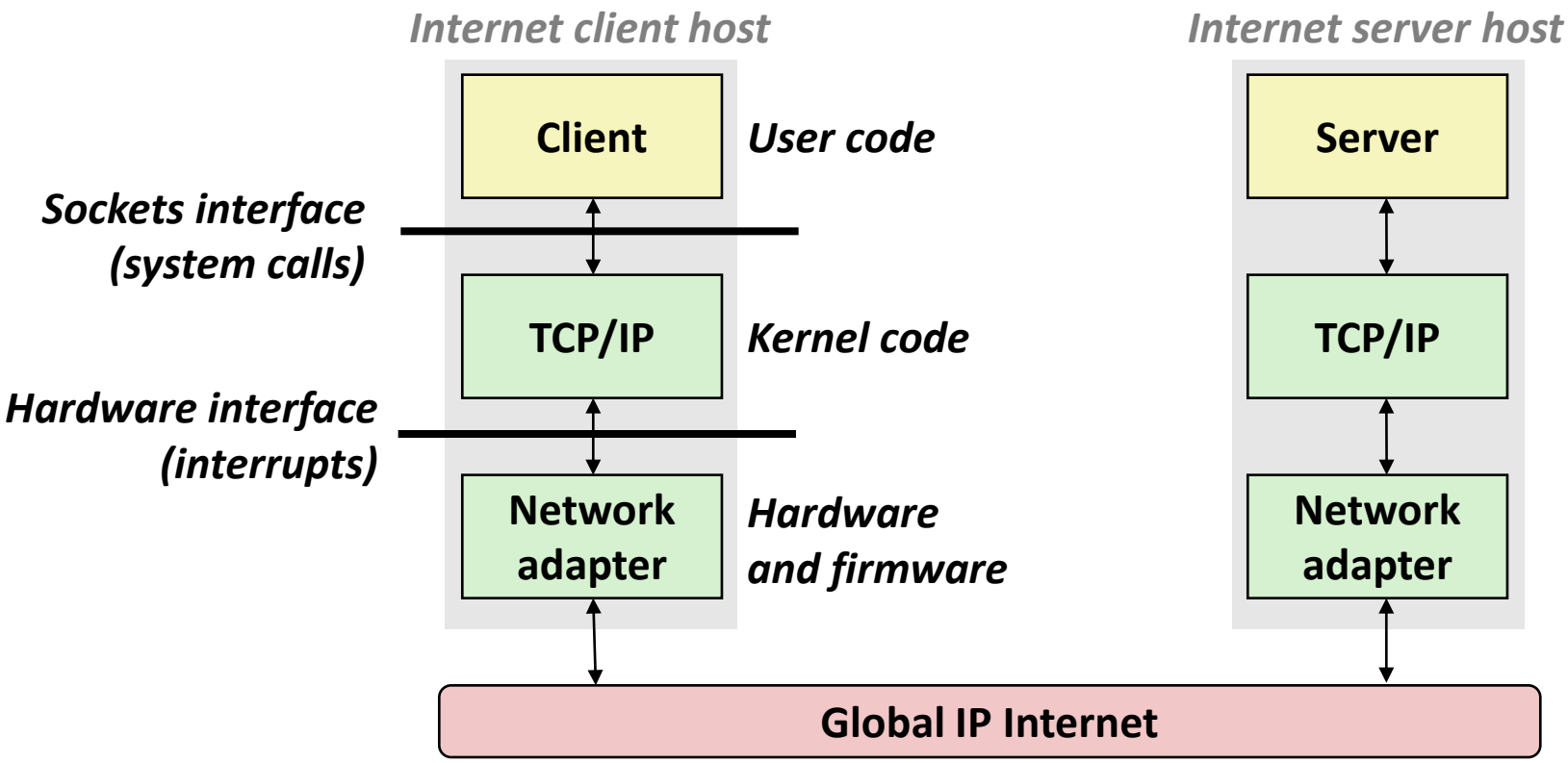
Global IP Internet (upper case)

- **Most famous example of an internet**

- **Based on the TCP/IP protocol family**
  - IP (Internet Protocol)
    - Provides *basic naming scheme* and unreliable *delivery capability* of packets (datagrams) from *host-to-host*
  - UDP (User Datagram Protocol)
    - Uses IP to provide *unreliable* datagram delivery from *process-to-process*
  - TCP (Transmission Control Protocol)
    - Uses IP to provide *reliable* byte streams from *process-to-process* over *connections*

- **Accessed via a mix of Unix file I/O and functions from the *sockets interface***

Hardware and Software Organization of an Internet Application



A Programmer's View of the Internet

1. Hosts are mapped to a set of 32-bit* *IP addresses*

- 128.2.203.179
- 127.0.0.1 (always *localhost*)

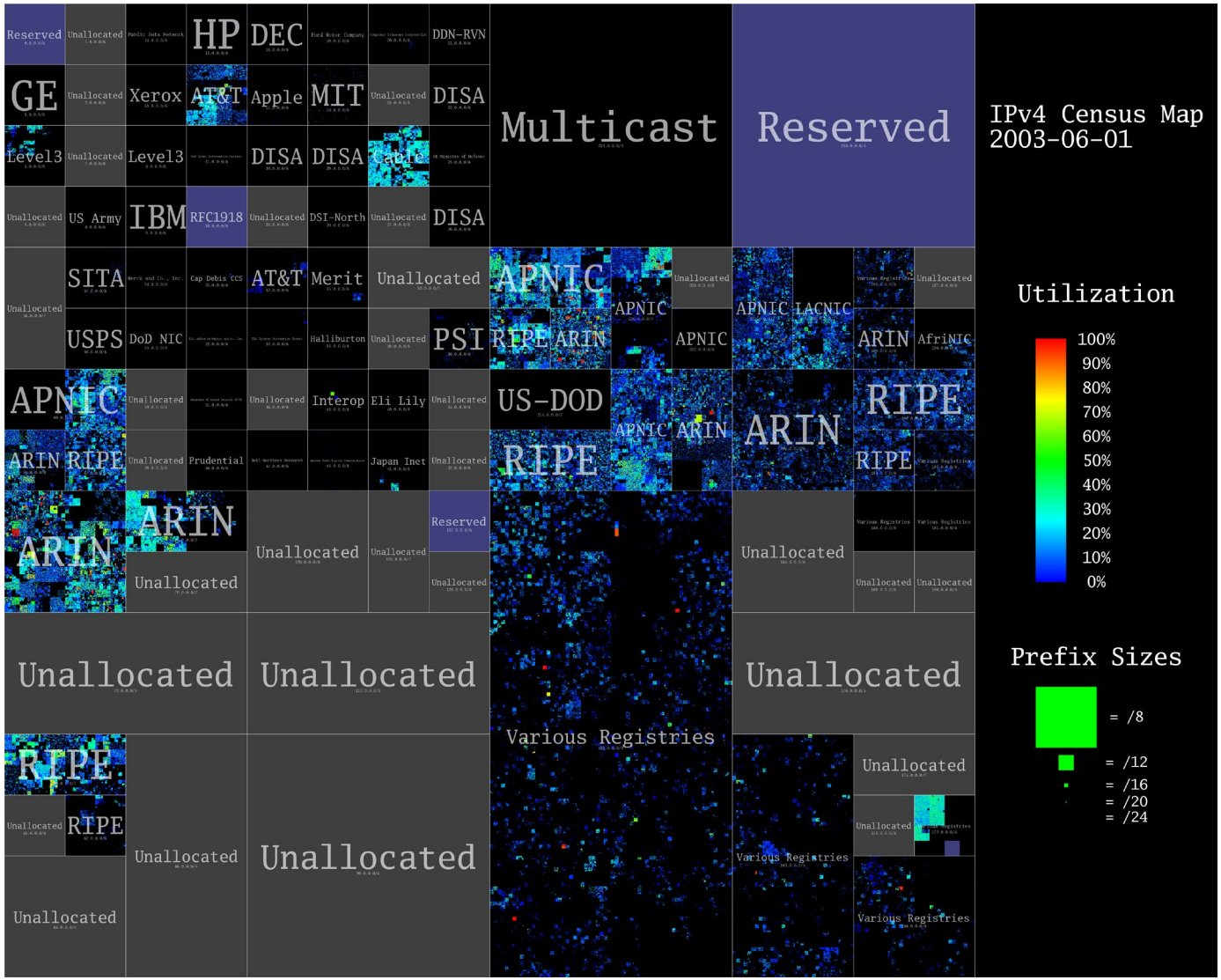
2. As a convenience for humans, the Domain Name System maps a set of identifiers called Internet *domain names* to IP addresses:

- www.cs.cmu.edu "resolves to" 128.2.42.95

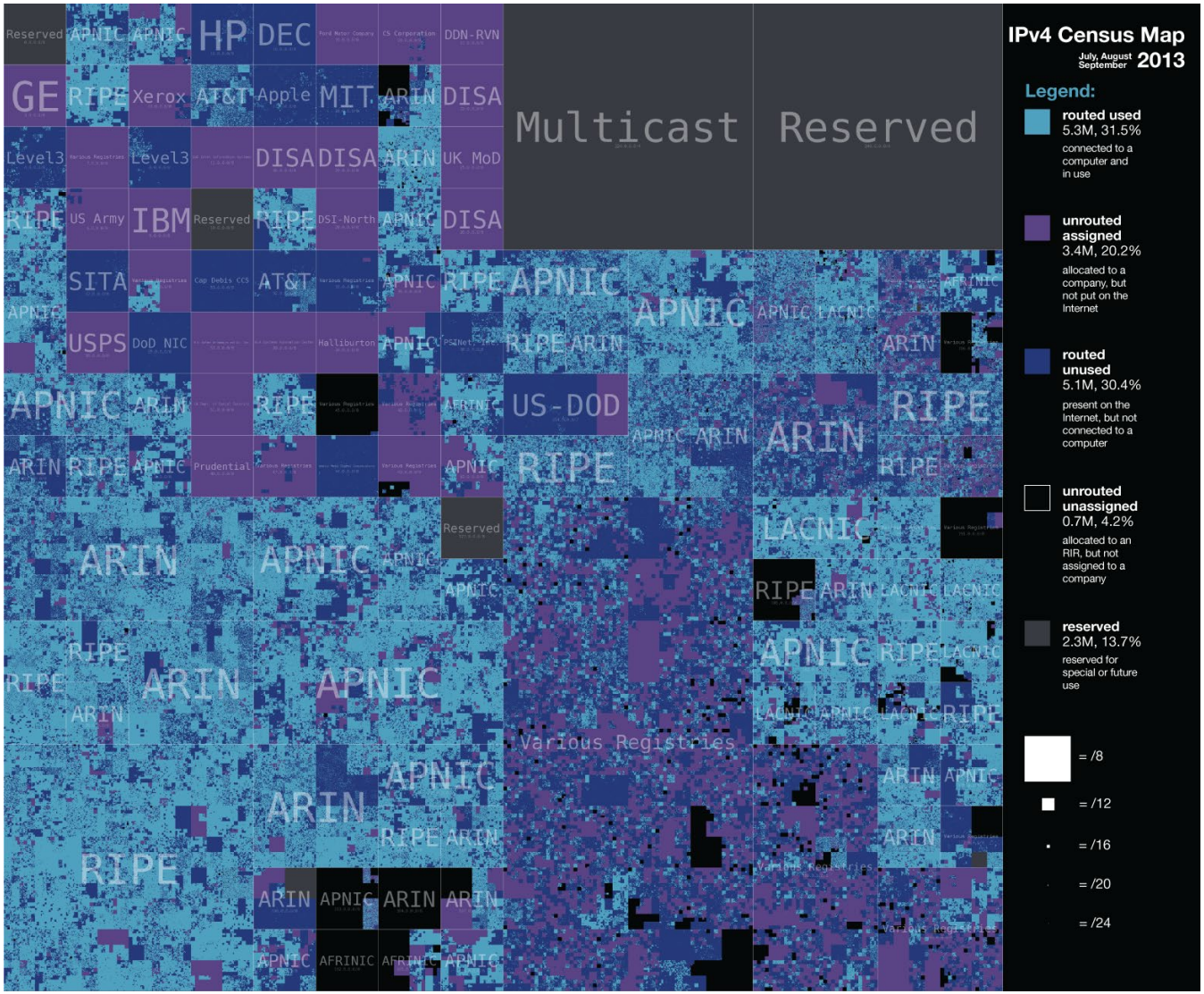
3. A process on one Internet host can communicate with a process on another Internet host over a *connection*

***: IPv4**

IPv4 map circa 2003



IPv4 map circa 2013



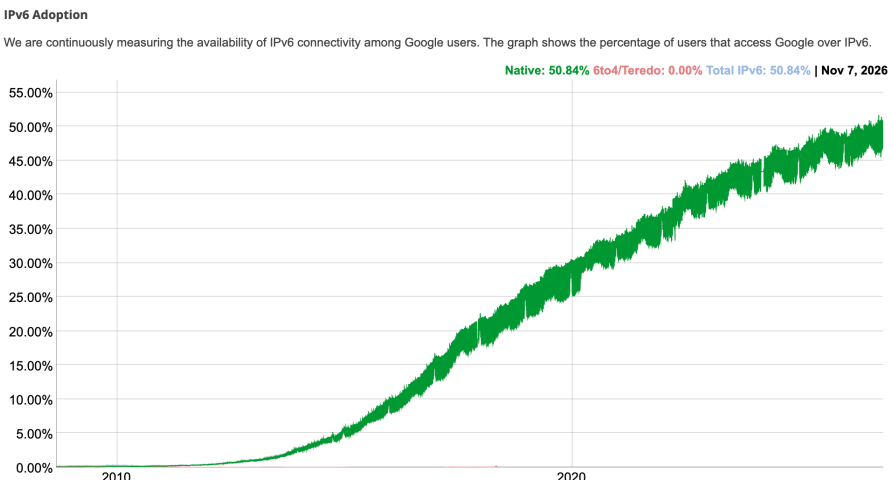
Aside: IPv4 and IPv6

- ***IPv4*** **(Internet Protocol version 4) specified 1981**
  - 32-bit host addresses (192.0.2.43)
  - Known to not be enough for everyone since ~1990
- **IPv6 (Internet Protocol version 6) specified 1996**
  - 128-bit addresses (2001:0db8:0:0:0:0:cafe:la7e)
  - Intended to replace IPv4
  - Very slow adoption due to need to replace routers (CMU's network doesn't support IPv6 at all!)

- **Application programmers mostly don't have to care**
  - Sockets API makes it easy to write code that seamlessly uses either, as necessary

IPv6 traffic to Google

https://www.google.com/intl/en/ipv6/statistics.html



(1) IP Addresses

- **32-bit IP addresses are stored in an *IP address struct***
  - IP addresses are always stored in memory in *network byte order* (big-endian byte order)
  - True in general for any integer transferred in a packet header from one machine to another.
    - E.g., the port number used to identify an Internet connection.

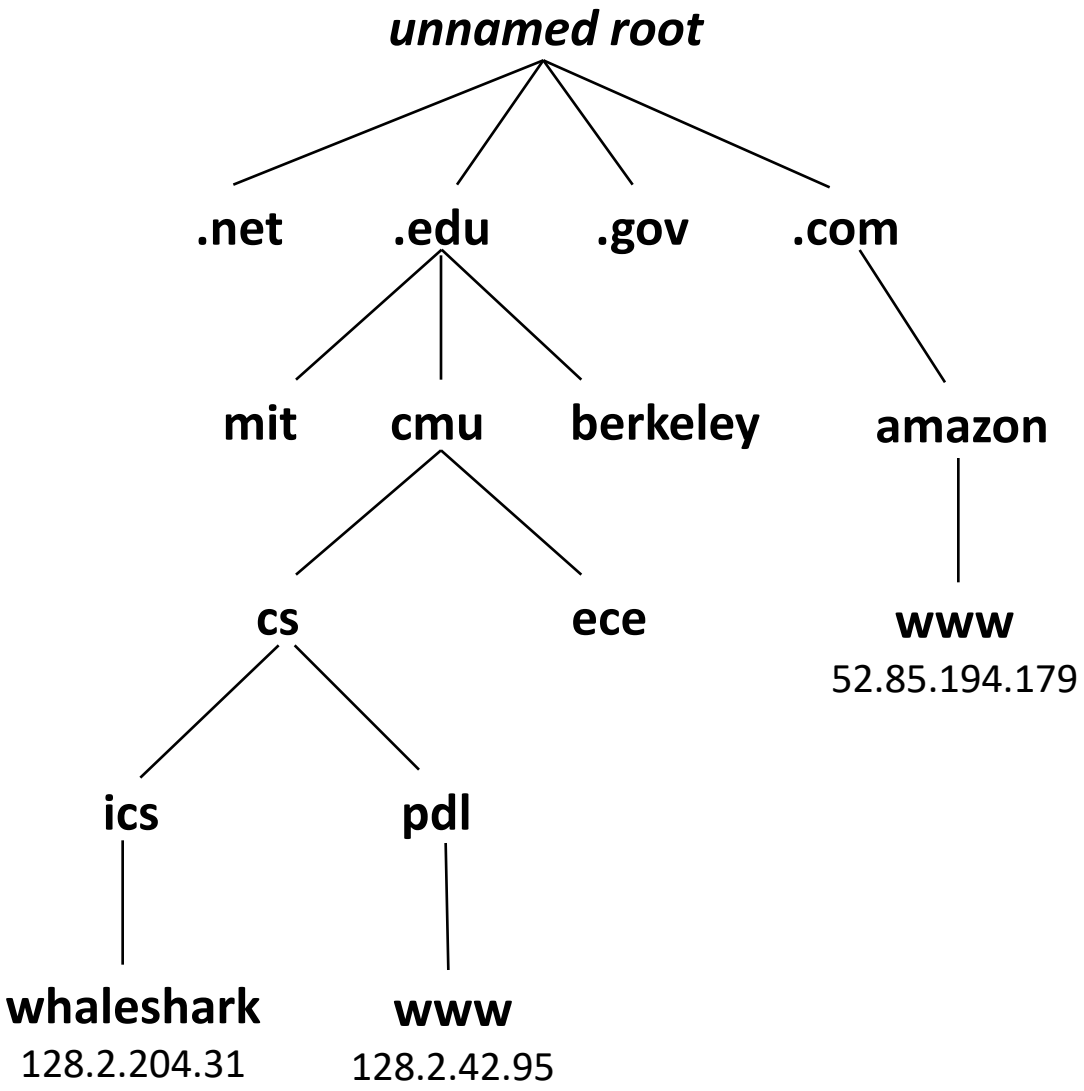
```
/* Internet address structure */
struct in_addr {
    uint32_t  s_addr; /* network byte order (big-endian) */
};
```

Dotted Decimal Notation

- **By convention, each byte in a 32-bit IP address is represented by its decimal value and separated by a period**
  - IP address: `0x8002C2F2 = 128.2.194.242`

- **Use `getaddrinfo` and `getnameinfo` functions (described later) to convert between IP addresses and dotted decimal format.**

(2) Internet Domain Names



First-level domain names

Second-level domain names

Third-level domain names

Domain Naming System (DNS)

- **The Internet maintains a mapping between IP addresses and domain names in a worldwide distributed database called *DNS***

- **Conceptually, programmers can view the DNS database as a collection of millions of *host entries.***
  - Each host entry defines the mapping between a set of domain names and IP addresses.
  - In a mathematical sense, a host entry is an equivalence class of domain names and IP addresses.

Properties of DNS Mappings

- **Can explore properties of DNS mappings using `nslookup`**
  - (Output edited for brevity)

- **Each host has a locally defined domain name `localhost` which always maps to the *loopback address* `127.0.0.1`**

```
linux> nslookup localhost
Address: 127.0.0.1
```

- **Use `hostname` to determine real domain name of local host:**

```
linux> hostname
whaleshark.ics.cs.cmu.edu
```

Properties of DNS Mappings (cont)

- Simple case: one-to-one mapping between domain name and IP address:

```
linux> nslookup whaleshark.ics.cs.cmu.edu
Address: 128.2.204.31
```

- Multiple domain names mapped to the same IP address:

```
linux> nslookup cs.cmu.edu
Address: 128.2.42.95
linux> nslookup scs.cmu.edu
Address: 128.2.42.95
```

- And backwards:

```
linux> nslookup 128.2.42.95
128.2.42.95.in-addr.arpa   name = SCS-WEB-LB.ANDREW.CMU.EDU.
```

Properties of DNS Mappings (cont)

- **Multiple domain names mapped to multiple IP addresses:**

```
linux> nslookup cnn.com
Address: 151.101.67.5
Address: 151.101.131.5
Address: 151.101.195.5
Address: 151.101.3.5

linux> nslookup cnn.com
Address: 151.101.3.5
Address: 151.101.67.5
Address: 151.101.131.5
Address: 151.101.195.5
```

- **Some valid domain names don't map to any IP address:**

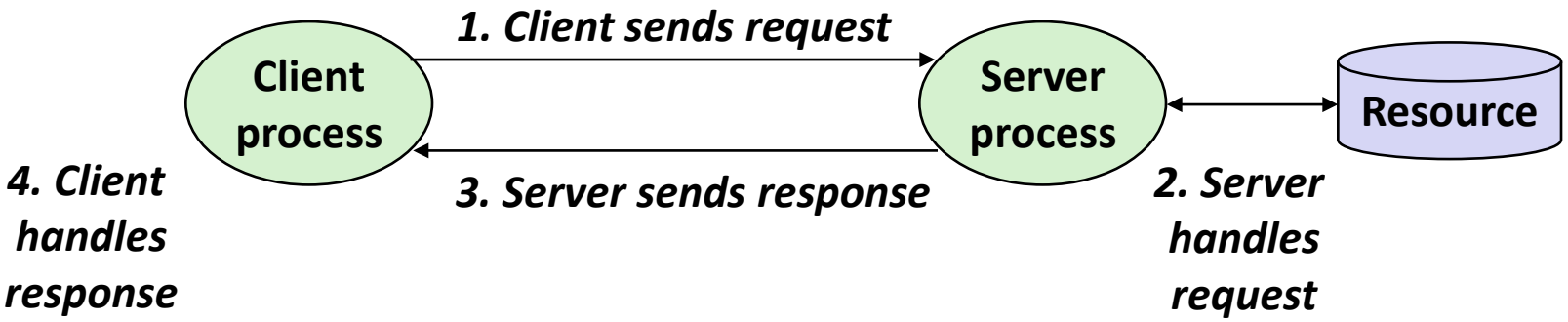
```
linux> nslookup ics.cs.cmu.edu
(No Address given)
```

Outline

- Network types and structures
- Locating a host
- **Setting up a connection**
- HTTP Example

A Client-Server Transaction

- **Most network applications are based on the client-server model:**
  - A ***server*** process and one or more ***client*** processes
  - Server manages some ***resource***
  - Server provides ***service*** by manipulating resource for clients
  - Server activated by request from client (vending machine analogy)



Note: clients and servers are processes running on hosts (can be the same or different hosts)

(3) Internet Connections

- **Clients and servers most often communicate by sending streams of bytes over TCP *connections*. Each connection is:**
  - *Point-to-point*: connects a pair of processes.
  - *Full-duplex*: data can flow in both directions at the same time,
  - *Reliable*: stream of bytes sent by the source is eventually received by the destination in the same order it was sent.

- ***A socket* is an endpoint of a connection**
  - *Socket address* is an `IPaddress:port` pair

- **A *port* is a 16-bit integer that identifies a process:**
  - ***Ephemeral port*: Assigned automatically by client kernel when client makes a connection request.**
  - ***Well-known port:* Associated with some *service* provided by a server (e.g., port 80 is associated with Web servers)**

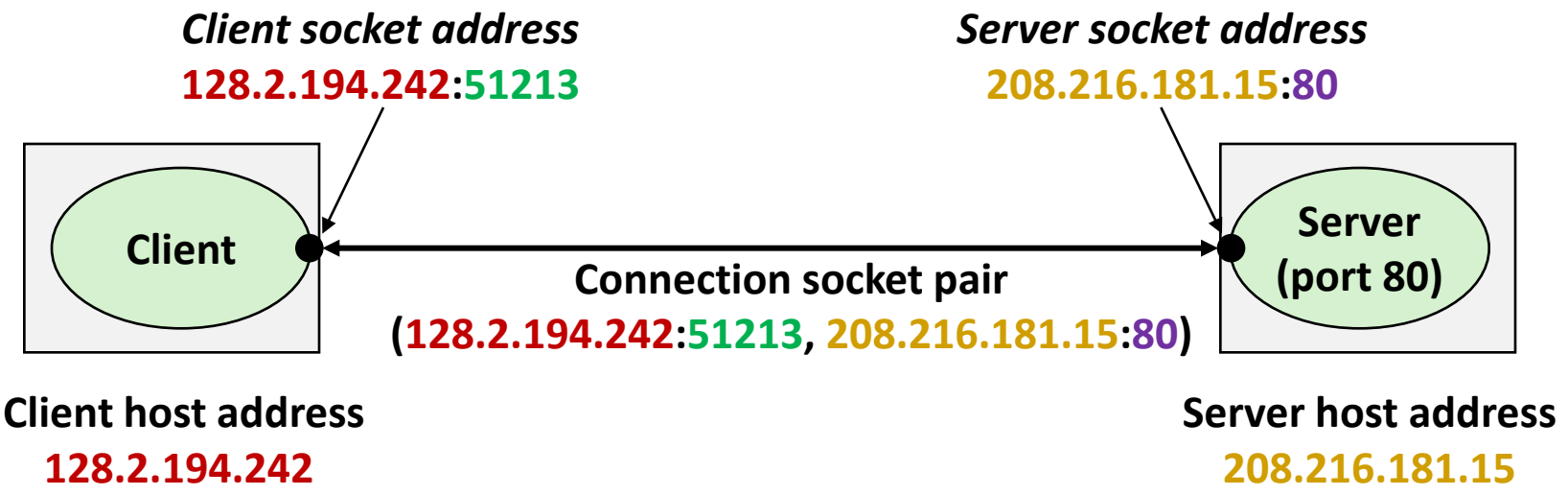
Well-known Service Names and Ports

- **Popular services have permanently assigned *well-known ports* *and* corresponding *well-known service names*:**
  - echo servers: echo 7
  - ftp servers: ftp 21
  - ssh servers: ssh 22
  - email servers: smtp 25
  - Unencrypted Web servers: http 80
  - SSL/TLS encrypted Web: https 443

- **Mappings between well-known ports and service names is contained in the file `/etc/services` on each Linux machine.**

Anatomy of a Connection

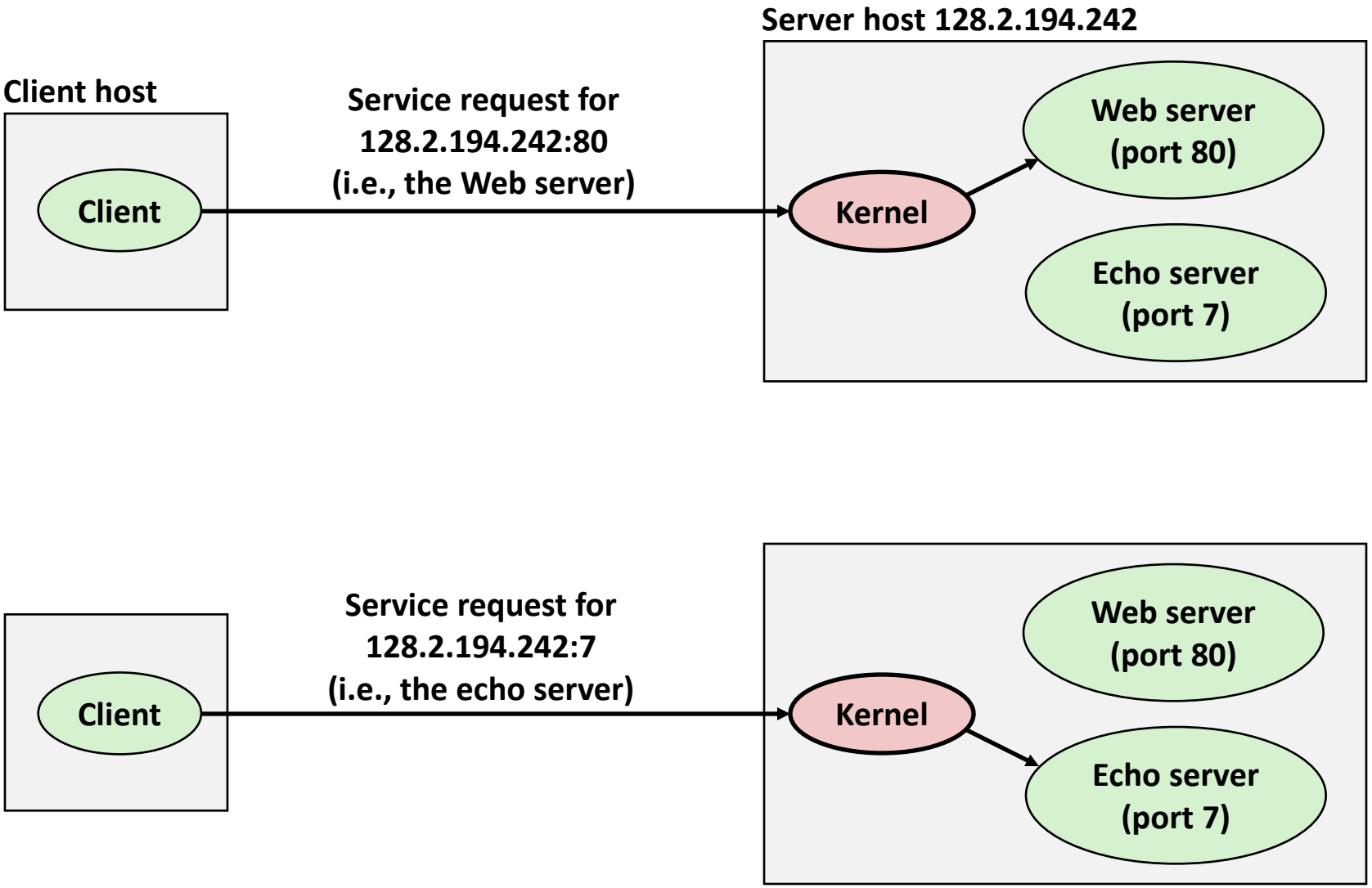
- A connection is uniquely identified by the socket addresses of its endpoints (*socket pair*)
  - `(cliaddr:cliport, servaddr:servport)`



51213 is an ephemeral port allocated by the kernel

80 is a well-known port associated with Web servers

Using Ports to Identify Services



Quiz

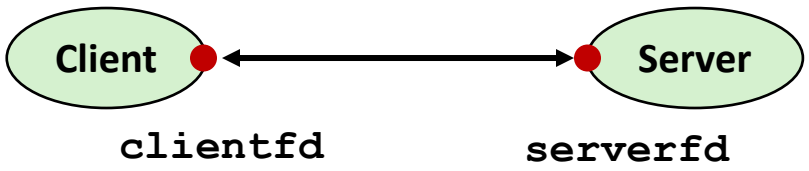
canvas.cmu.edu/courses/55964/quizzes/169264

Sockets Interface

- **Set of system-level functions used in conjunction with Unix I/O to build network applications.**
- **Created in the early 80's as part of the original Berkeley distribution of Unix that contained an early version of the Internet protocols.**
- **Available on all modern systems**
  - Unix variants, Windows, OS X, IOS, Android, ARM

Sockets

- **What is a socket?**
  - To the kernel, a socket is an endpoint of communication
  - To an application, a socket is a file descriptor that lets the application read/write from/to the network
  - Using the FD abstraction lets you reuse code & interfaces
- **Clients and servers communicate with each other by reading from and writing to socket descriptors**



- **The main distinction between regular file I/O and socket I/O is how the application "opens" the socket descriptors**

Socket Programming Example

- **Echo server and client**
- **Server**
  - Accepts connection request
  - Repeats back lines as they are typed
- **Client**
  - Requests connection to server
  - Repeatedly:
    - Read line from terminal
    - Send to server
    - Read reply from server
    - Print line to terminal

Echo Server/Client Session Example

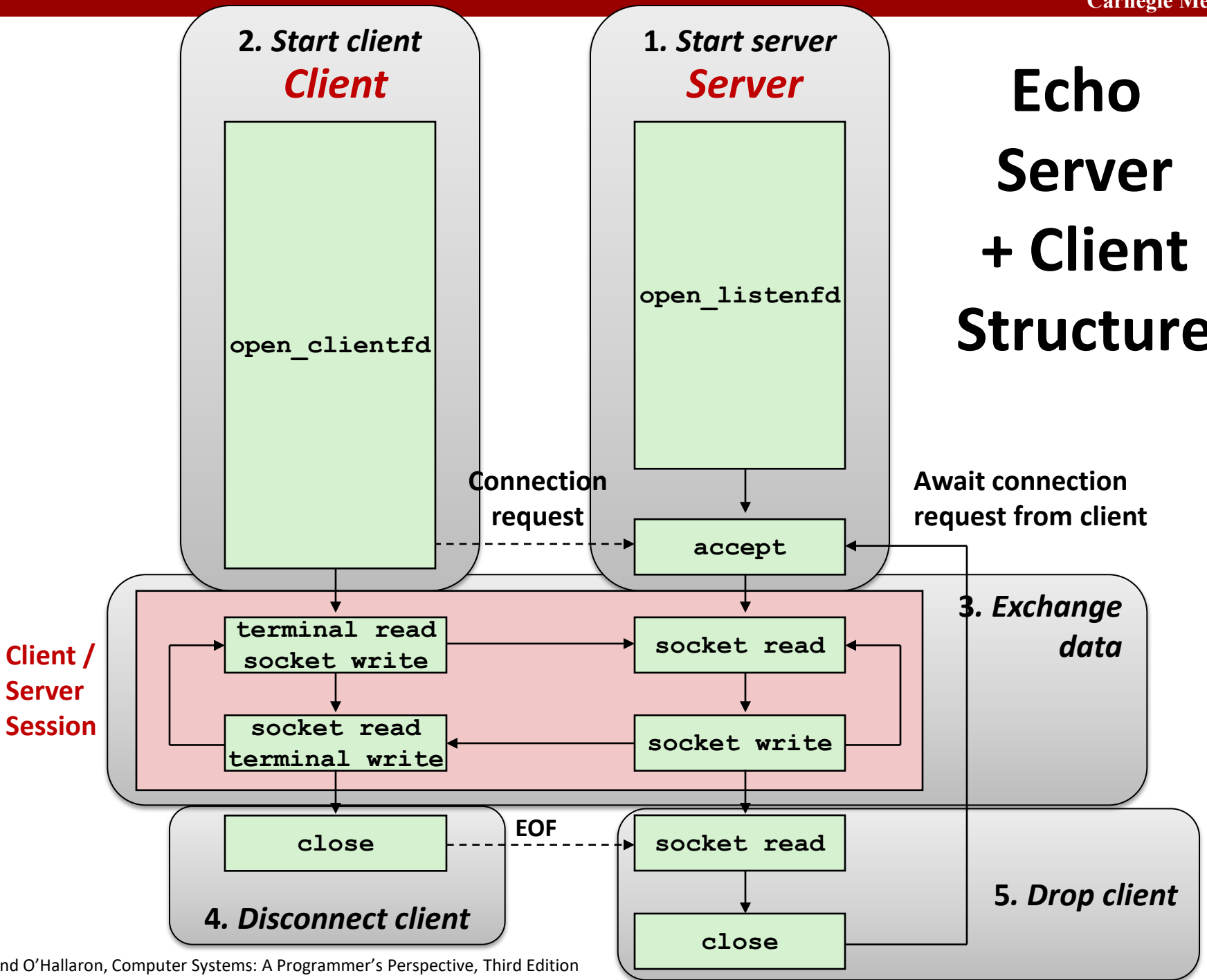
Client

```
bambooshark: ./echoclient whaleshark.ics.cs.cmu.edu 6616        (A)
This line is being echoed                                       (B)
This line is being echoed
This one is, too                                                (C)
This one is, too
^D
bambooshark: ./echoclient whaleshark.ics.cs.cmu.edu 6616        (D)
This one is a new connection                                    (E)
This one is a new connection
^D
```

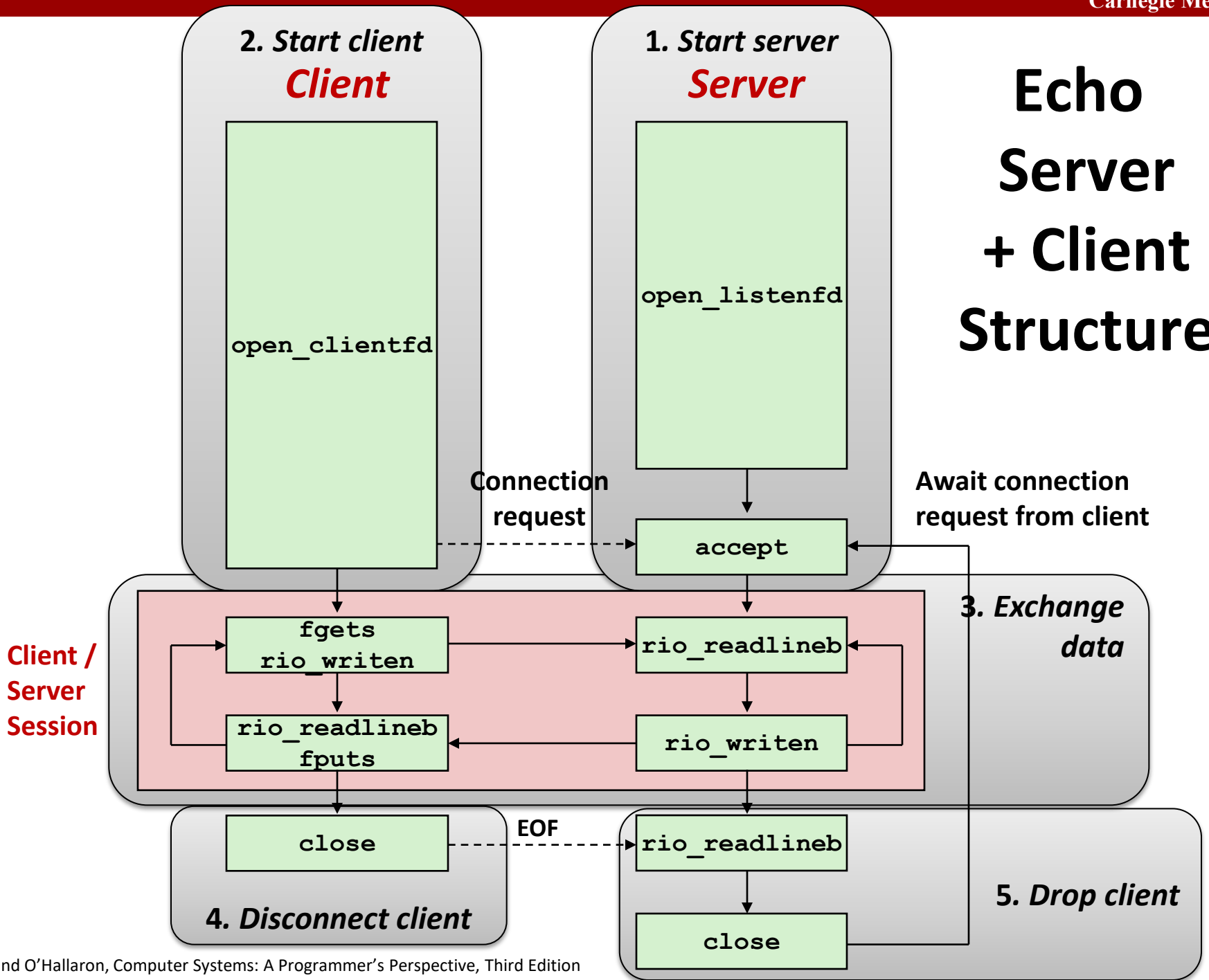
Server

```
whaleshark: ./echoserveri 6616
Connected to (BAMBOOSHARK.ICS.CS.CMU.EDU, 33707)        (A)
server received 26 bytes                                (B)
server received 17 bytes                                (C)
Connected to (BAMBOOSHARK.ICS.CS.CMU.EDU, 33708)        (D)
server received 29 bytes                                (E)
```

Echo Server + Client Structure



Echo Server + Client Structure



Recall: Unbuffered RIO Input/Output

- Same interface as Unix `read` and `write`
- Especially useful for transferring data on network sockets

```
#include "csapp.h"

ssize_t rio_readn(int fd, void *usrbuf, size_t n);
ssize_t rio_writen(int fd, void *usrbuf, size_t n);
```

Return: num. bytes transferred if OK, 0 on EOF (`rio_readn` only), -1 on error

- **`rio_readn`** returns short count only if it encounters EOF
  - Only use it when you know how many bytes to read
- **`rio_writen`** never returns a short count
- Calls to **`rio_readn`** and **`rio_writen`** can be interleaved arbitrarily on the same descriptor

Recall: Buffered RIO Input Functions

- **Efficiently read text lines and binary data from a file partially cached in an internal memory buffer**

```
#include "csapp.h"

void rio_readinitb(rio_t *rp, int fd);

ssize_t rio_readlineb(rio_t *rp, void *usrbuf, size_t maxlen);
ssize_t rio_readnb(rio_t *rp, void *usrbuf, size_t n);
```

Return: num. bytes read if OK, 0 on EOF, -1 on error

- **`rio_readlineb`** reads a ***text line*** of up to **`maxlen`** bytes from file **`fd`** and stores the line in **`usrbuf`**
  - Especially useful for reading text lines from network sockets
- Stopping conditions
  - **`maxlen`** bytes read
  - EOF encountered
  - Newline (‘**`\n`**’) encountered

Echo Client: Main Routine

```
#include "csapp.h"

int main(int argc, char **argv)
{
    int clientfd;
    char *host, *port, buf[MAXLINE];
    rio_t rio;

    host = argv[1];
    port = argv[2];

    clientfd = Open_clientfd(host, port);
    Rio_readinitb(&rio, clientfd);

    while (Fgets(buf, MAXLINE, stdin) != NULL) {
        Rio_writen(clientfd, buf, strlen(buf));
        Rio_readlineb(&rio, buf, MAXLINE);
        Fputs(buf, stdout);
    }
    Close(clientfd);
    exit(0);
}
```

echoclient.c

Echo Server + Client Structure

