Lecture 19: Network Programming

CS213 – Intro to Computer Systems

Branden Ghena – Fall 2023

Some slides borrowed from:
Bryant and O'Hallaron (CMU) and Eleftherios Kosmas (UCSD)

Reminders

- SETI Lab is due today!
  - Slip days / late policy can still be used if you don't finish tonight
  - Remember to submit to Gradescope too

- Midterm Exam 2 details
  - 80-minute exam, same style as last time
  - Covers lectures 8-19 (Assembly Procedures to Networks)
    - Particular emphasis:
      - Assembly Procedures, Pointers, Arrays, and Structs
      - Caches and Cache Performance
      - Virtual Memory
  - **Two 8.5" x 11" sheets of paper** with notes on front and back
  - Practice exams posted on Canvas

Today's Goals

- How do computers communicate?
- What choices does the Internet make about how to communicate?
- What system calls can programs use to communicate with other computers?
- See CS340 for more details!

Outline

- **Computer Networks**
  - **Topology**
  - Inter-network communication
- The Internet
  - Protocol choices
- Sockets
  - System calls for communicating with other computers
- Web Servers

Computer networks

- A network is a connection of two or more computers
  - Definition includes hardware and software components

- Networks might serve a variety of regions
  - Local Area Network (LAN)
    - Connects computers in some small area
    - Home, building, campus

  - Wide Area Network (WAN)
    - Connects computers over distant areas
      - City, country, world

- Could be wired or wireless

How do we connect computers?

- First question of a network: how are the computers actually connected?

- There are various choices here with different implications

How to connect: point-to-point networks

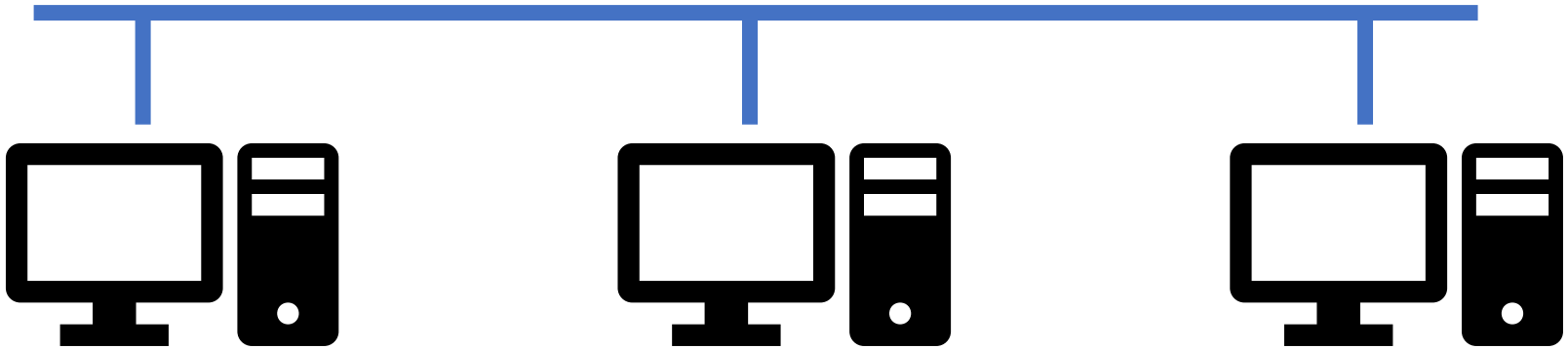
- How do we connect computers in a network?
  - This is a question of "network topology"
  - Simple option: just connect them directly

- Problem: what if I find a third computer?

How to connect: bus networks

- Connect everything to one wire in parallel
  - Actually a "multidrop bus"
  - Scales pretty well to many computers

- Problem: which computer gets to transmit when?
  - Simultaneous transmissions conflict



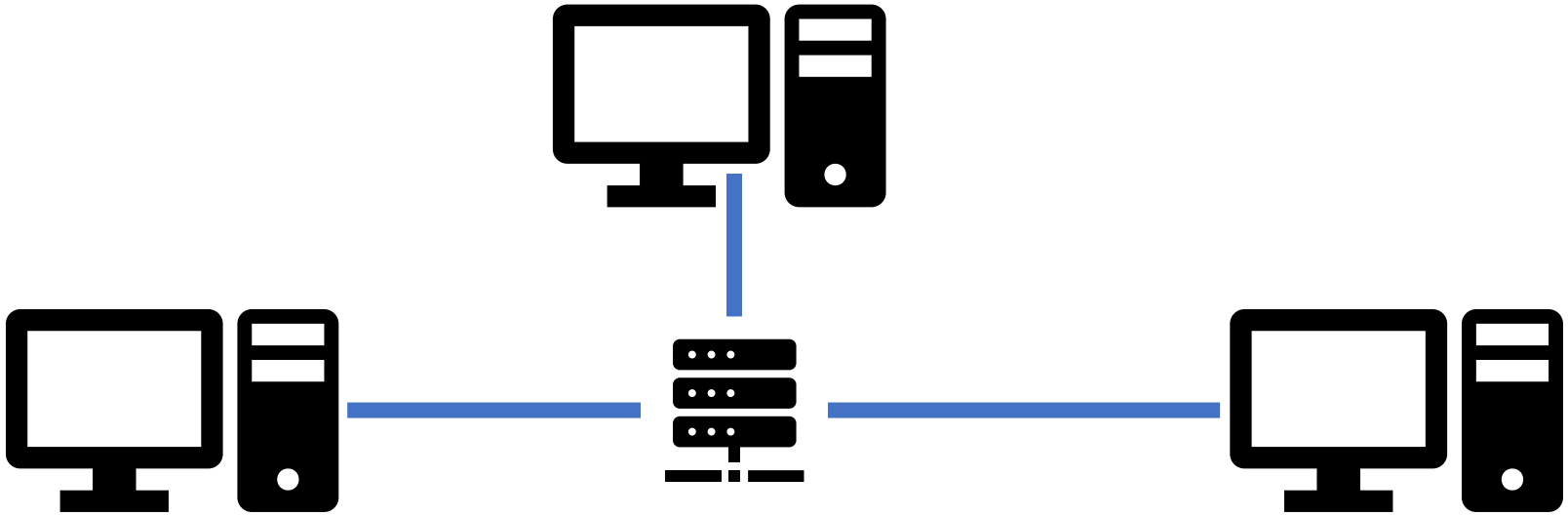
How to connect: ring networks

- Connect everything with point-to-point connections
  - Connect the last computer back to the first computer

- Problem: what if a computer stops sending?

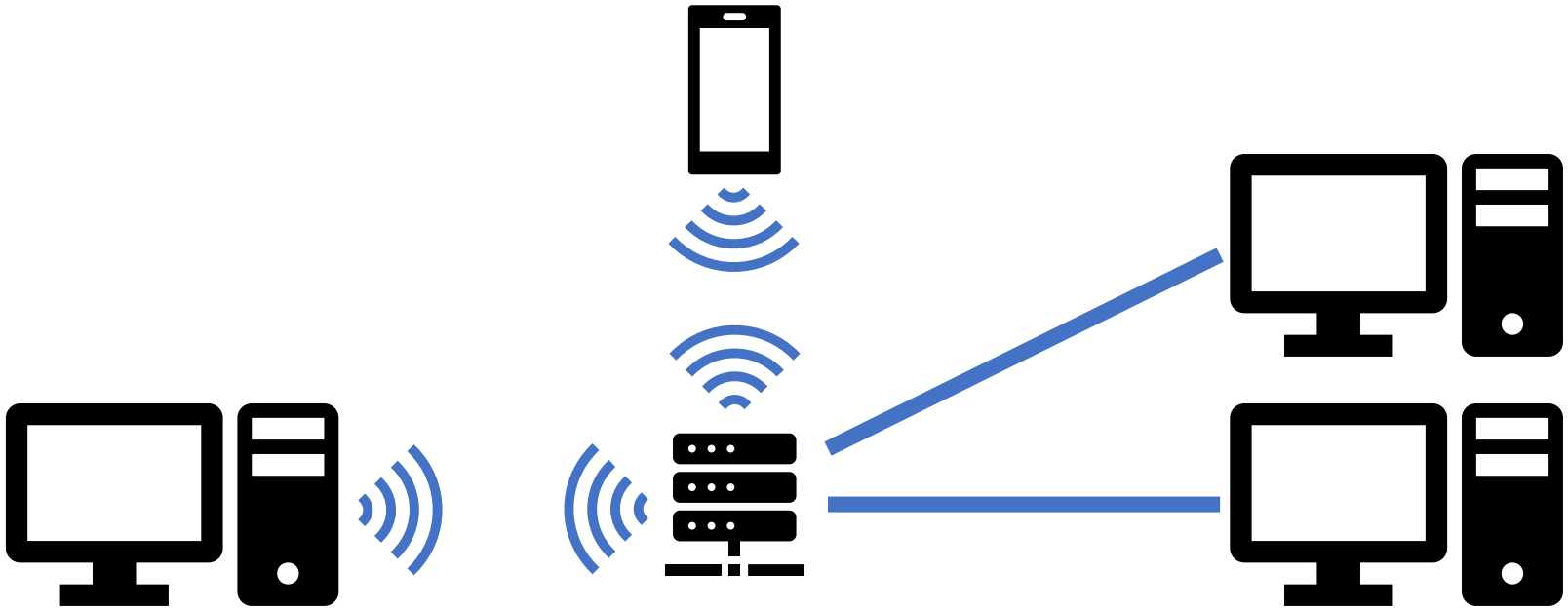
How to connect: star networks

- Connect to a hub with point-to-point connections
  - Hub connects all computers
  - Hub is a simple computer with one job: transfer communications between computers
    - Hopefully more reliable than any of the computers



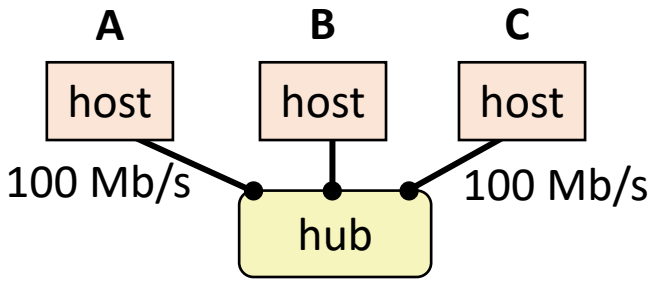
Common home networks

- Example of a star topology network
  - Your home router is the hub connecting multiple computers
  - Could be wired (Ethernet) or wireless (WiFi)

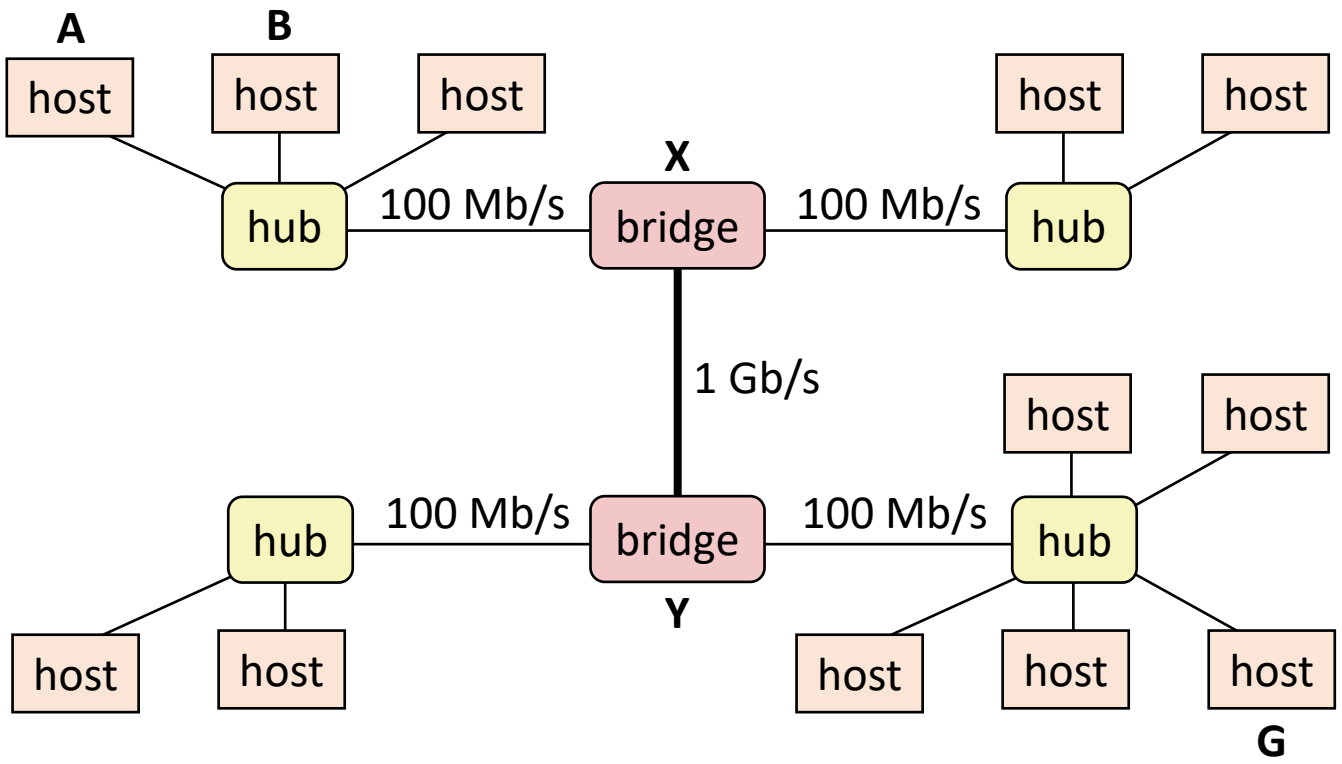


Building a network: ethernet segment

- Smallest division of a network
- **Hosts** are connected to **hub** via wires
- Operation
  - Each Ethernet adapter has a unique 48-bit address (MAC Address)
    - e.g. 00:16:ea:e3:54:e6
  - Hosts send bits to any other host in chunks called **frames**
  - Hub simply copies bits from one input to *all* outputs
    - Every host sees every transmitted bit



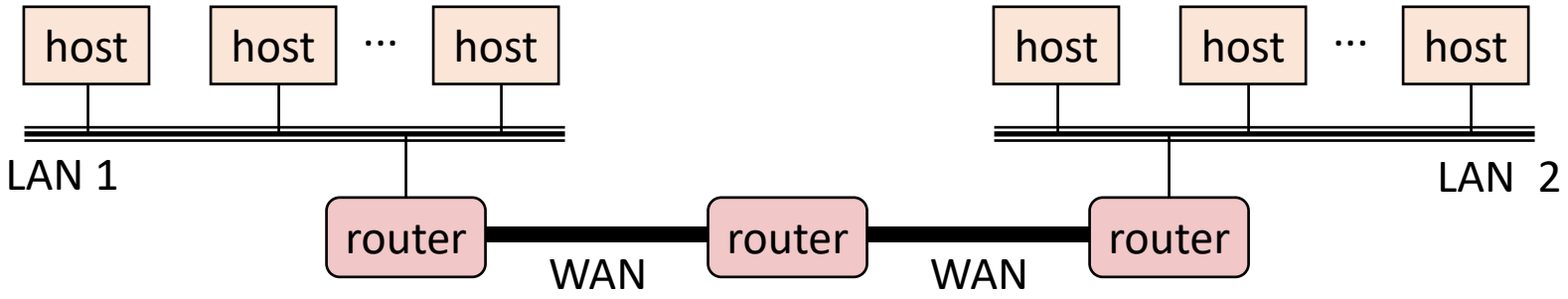
Building a network: bridged ethernet segment



- Bridges connect multiple hubs together
  - Learn which hosts are on which hub and *route* packets accordingly
  - Modern household routers are actually bridges
  - Creates a Local Area Network (LAN)

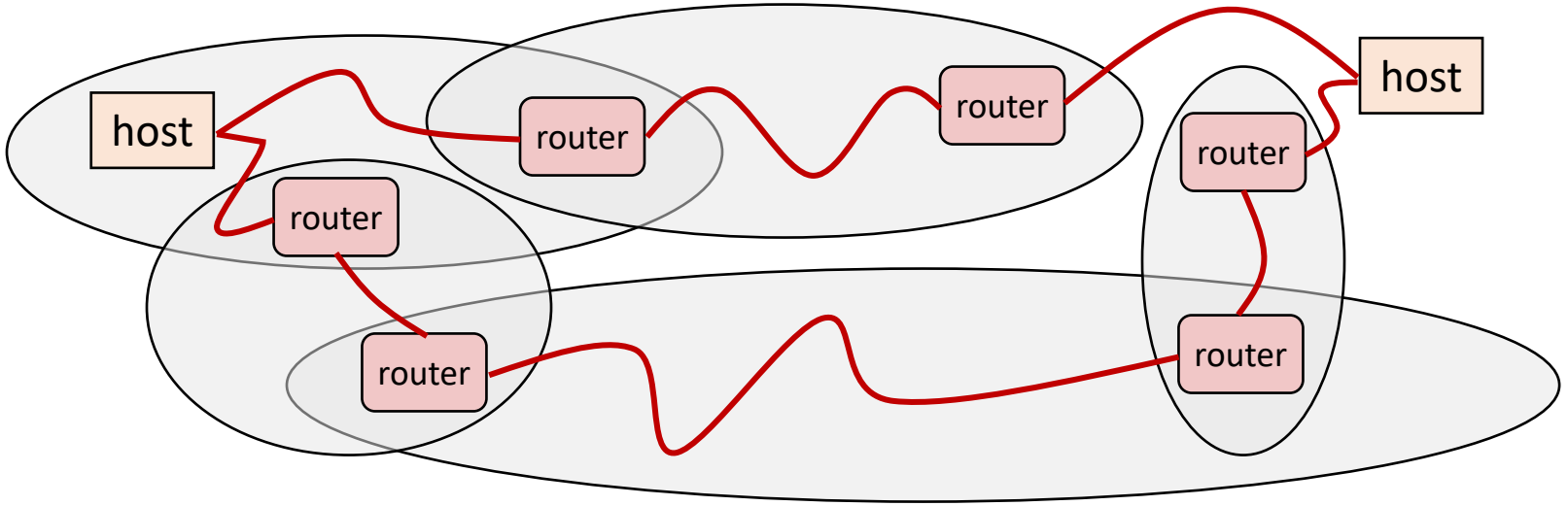
Building a network: internets

- Multiple local area networks (LANs) can be physically connected by specialized computers called **routers**
  - The connected networks are called an *internet* (short for internetworking)
  - Networks might be running totally separate protocols
    - Ethernet, WiFi, …



- Note: still star topology (not bus), just abstracting details away

Structure of an internet



- Ad hoc interconnection of networks
  - No particular topology
  - Possibly vastly different link capabilities
- Send packets from source to destination by hopping through networks

Outline

- **Computer Networks**
  - Topology
  - **Inter-network communication**
- The Internet
  - Protocol choices
- Sockets
  - System calls for communicating with other computers
- Web Servers

Managing inter-network communication with a protocol

- Now we have the ability to form and connect networks
  - But how do we transmit data between them?
  - Especially given that they differ in capabilities and design

- 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

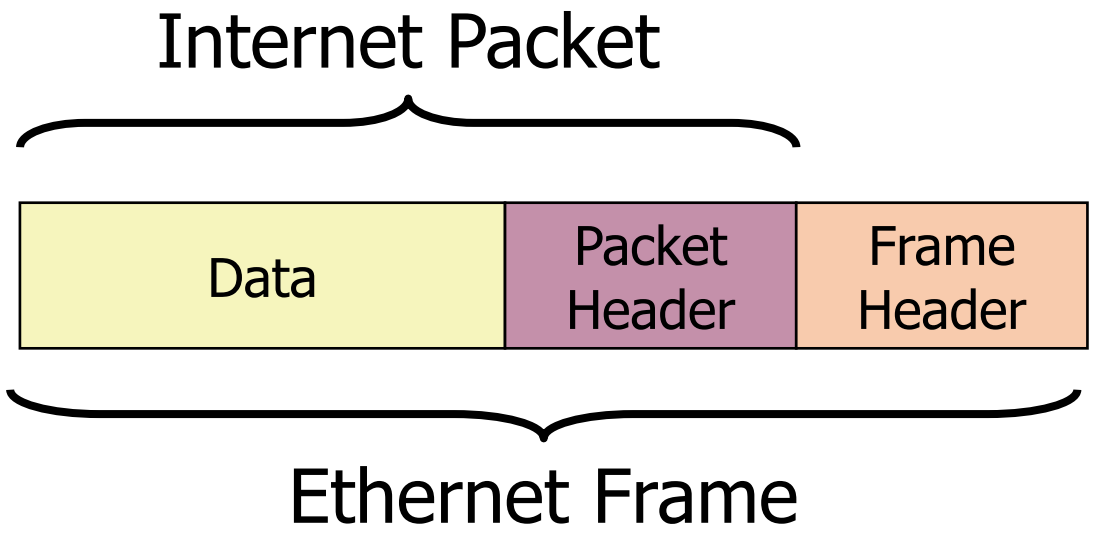
The role of an internet protocol

- Provides a **naming scheme**
  - We cannot rely on any particular lower-layer address
  - Defines a uniform format for *host addresses*
  - Each host (and router) is assigned at least one of these internet addresses that uniquely identifies it

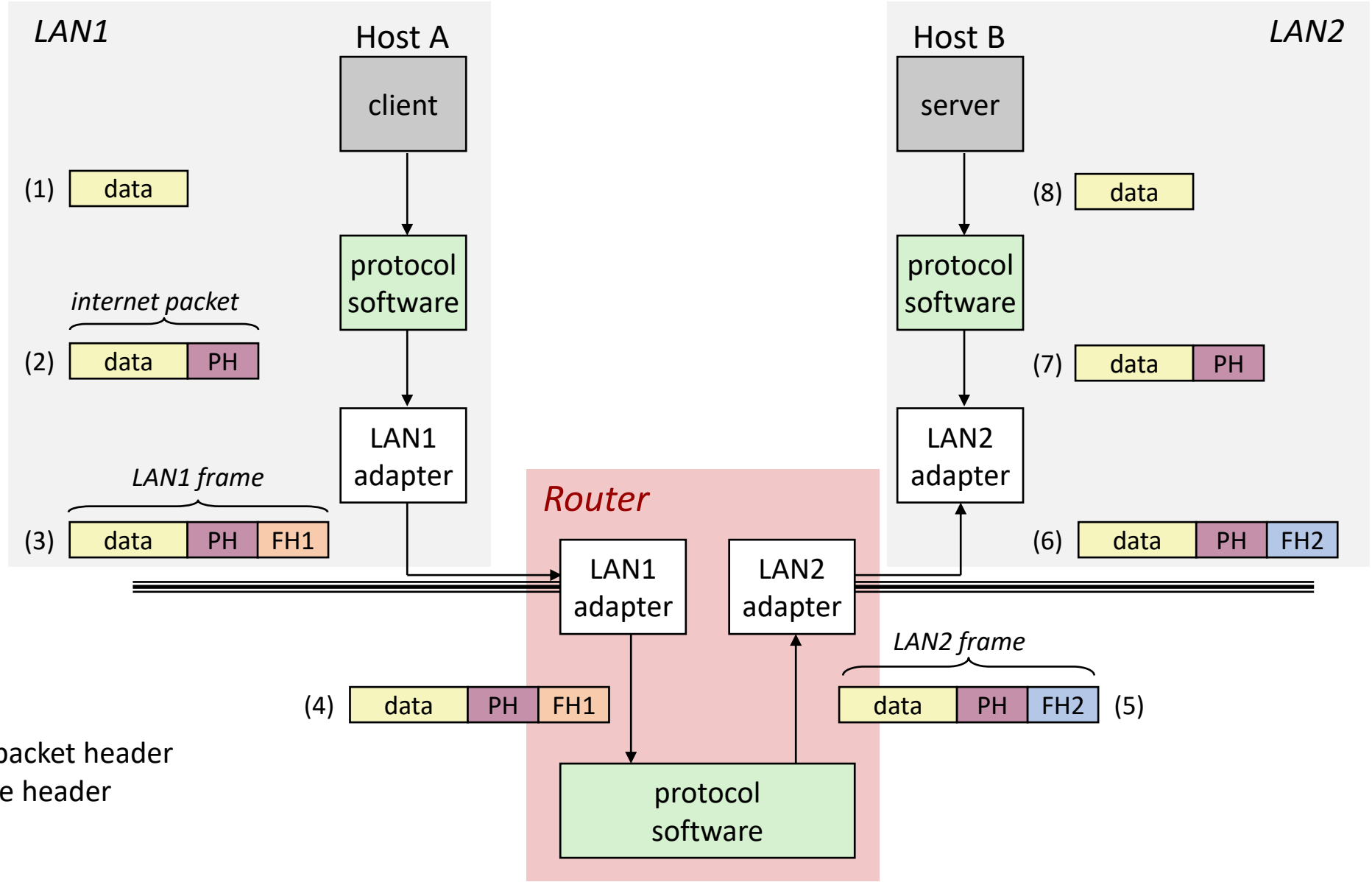
- Provides a **delivery mechanism**
  - Defines a standard transfer unit: packet
  - A packet consists of a header and a payload
    - Header: Metadata such as packet size, source and destination addresses
    - Payload: data bits sent from the source host

Protocols are "layered"

- Headers for each layer of communication wrap data
  - Data is wrapped with header for the network to make a packet
    - i.e., bytes are added to the start/end of it
  - Packet is wrapped with header for the link to make a frame



Transmitting data between networks



PH: Internet packet header
FH: LAN frame header

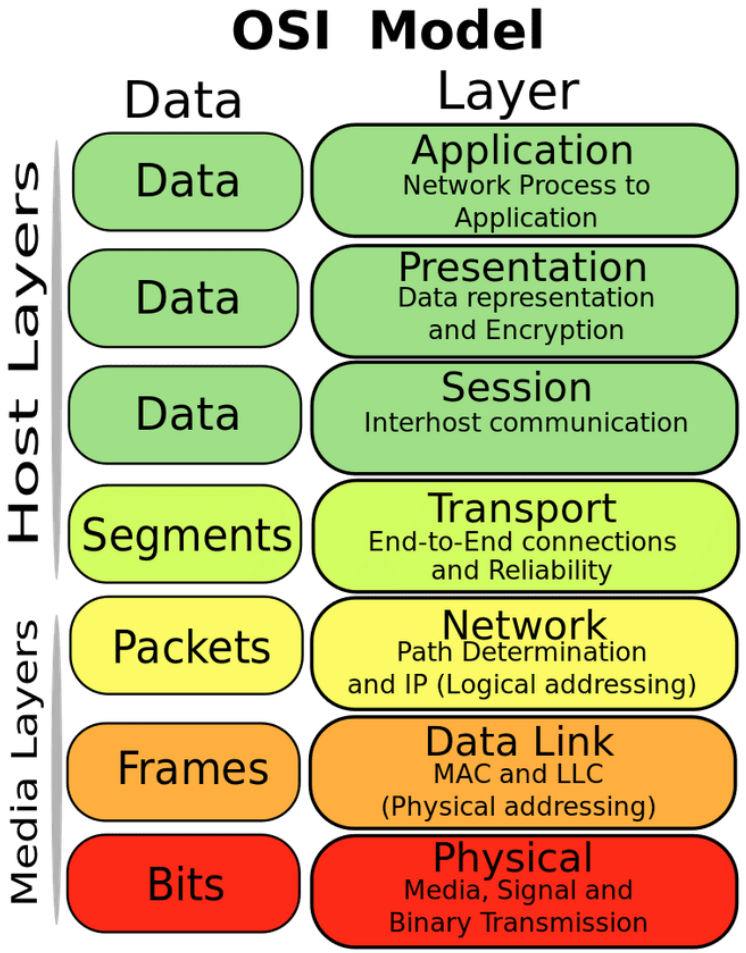
Other inter-network issues

- 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?

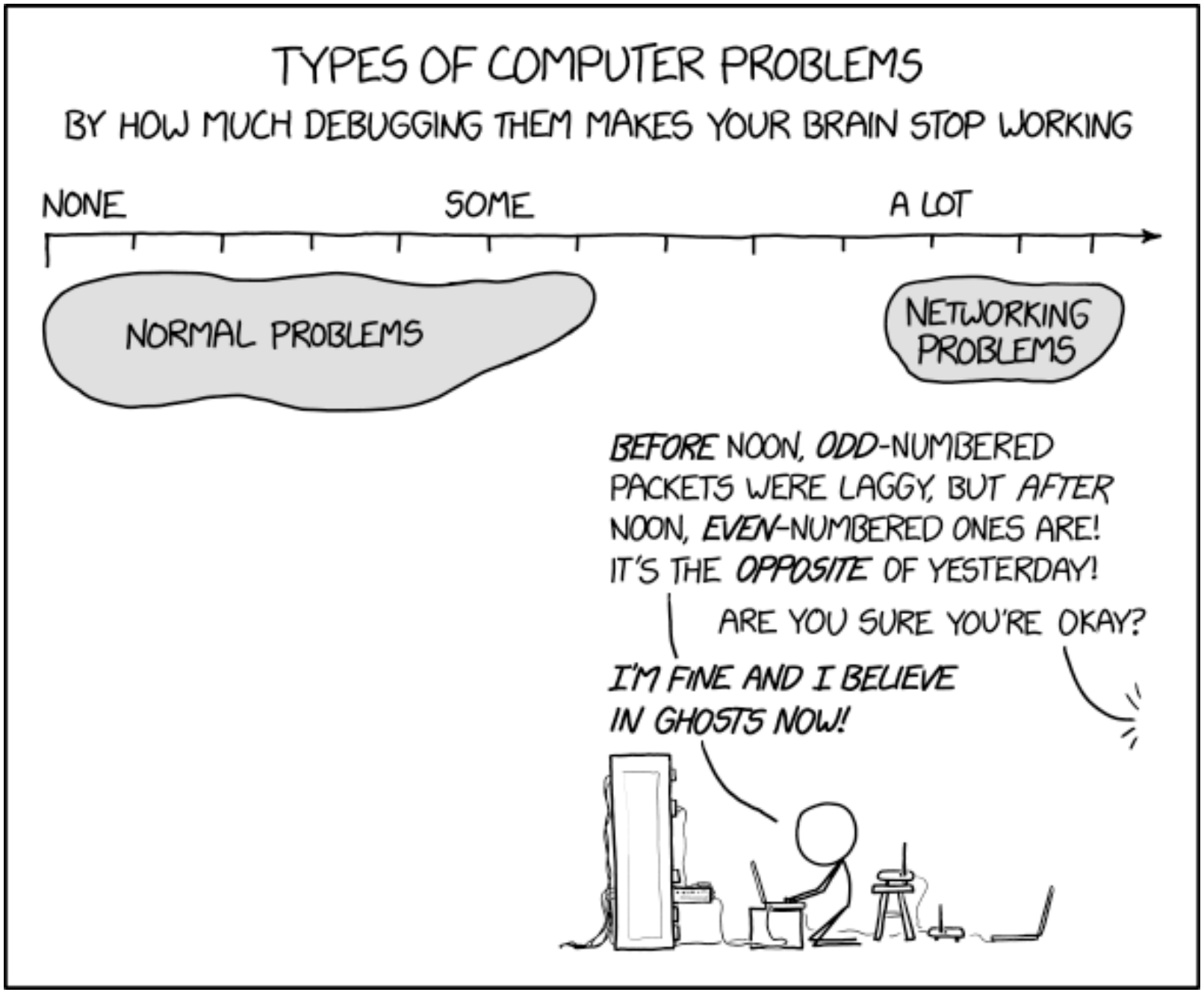
- These (and other) questions are addressed by the area of systems known as **computer networking**
  - CS340 – Intro to Computer Networks

OSI model of communication layers

- Transport
  - How to form connections between computers
  - TCP and UDP
- Network
  - How to send packets between networks
  - IP
- Data Link
  - How to send frames of data
  - Ethernet, WiFi
- Physical
  - How to send individual bits
  - Ethernet, WiFi



Break + xkcd

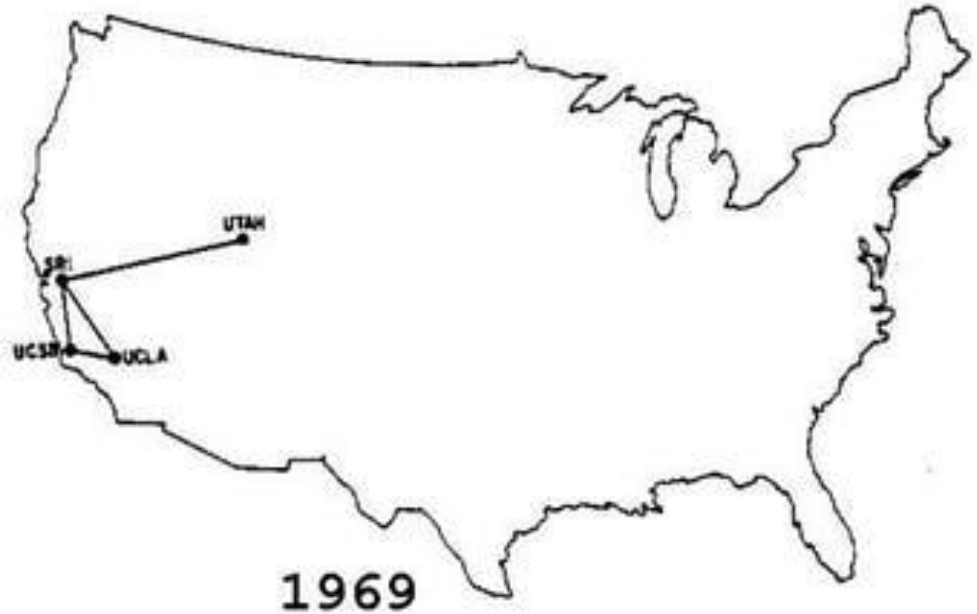


https://xkcd.com/2259/

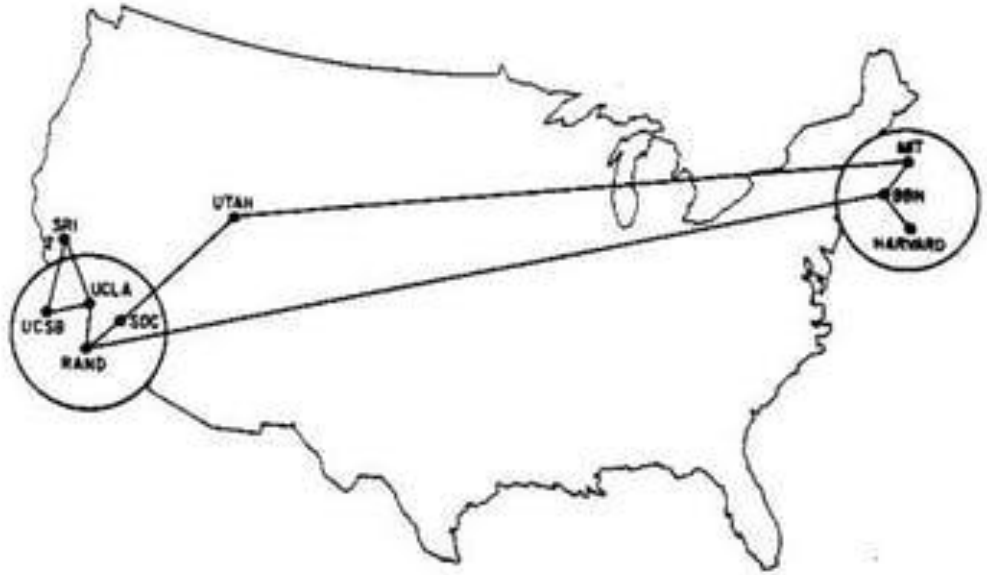
Outline

- Computer Networks
  - Topology
  - Inter-network communication
- **The Internet**
  - **Protocol choices**
- Sockets
  - System calls for communicating with other computers
- Web Servers

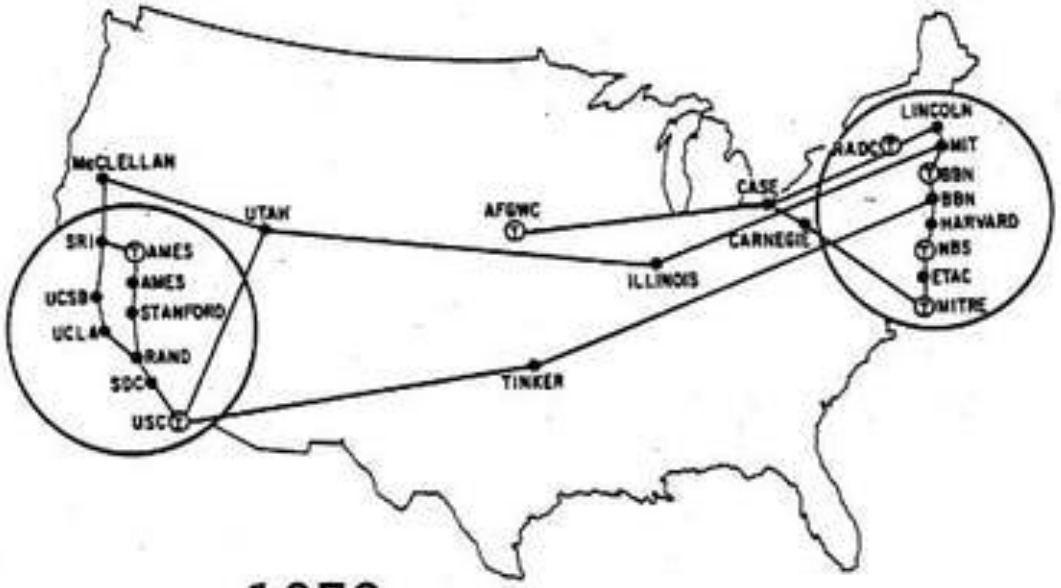
The ARPANET in December 1969



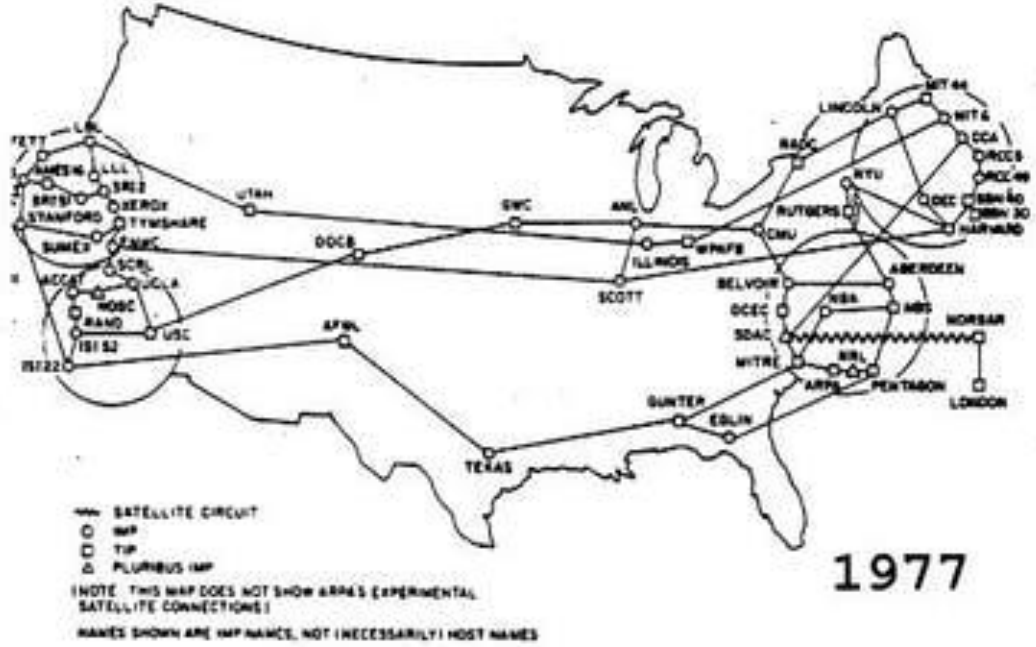
1969



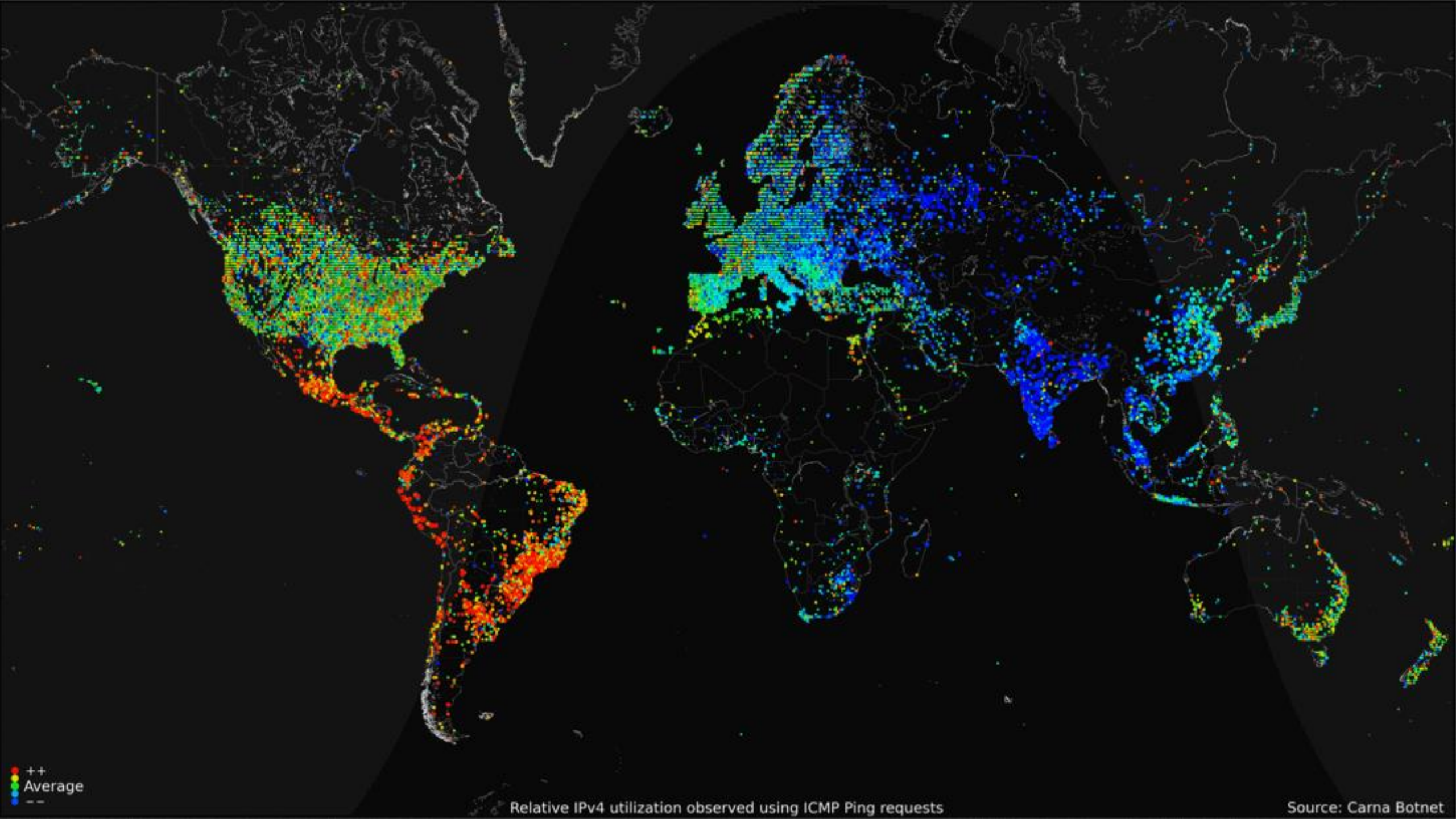
1970

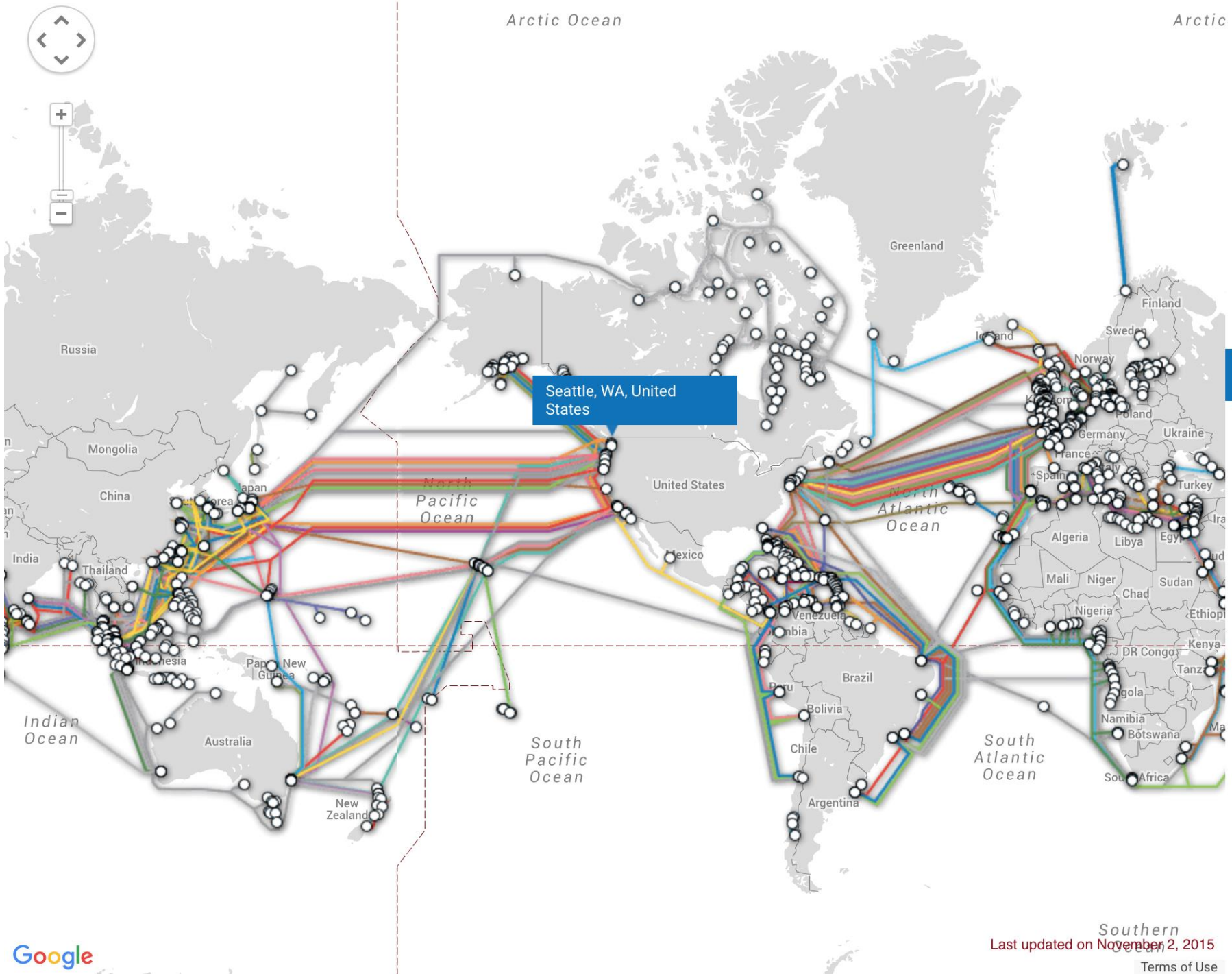


1972



1977


++
Average
--
Relative IPv4 utilization observed using ICMP Ping requests
Source: Carna Botnet



TeleGeography

Submarine Cable Map

The Submarine Cable Map is a free resource from TeleGeography. Data contained in this map is drawn from the Global Bandwidth Research Service and is updated on a regular basis.

To learn more about TeleGeography or this map please click here.

HUAWEI MARINE NETWORKS

Sponsored in part by Huawei Marine

Feedback

Search

Submarine Cable List

Seattle, WA, United States

Email link

Cables

Arctic Fibre

Last updated on November 2, 2015

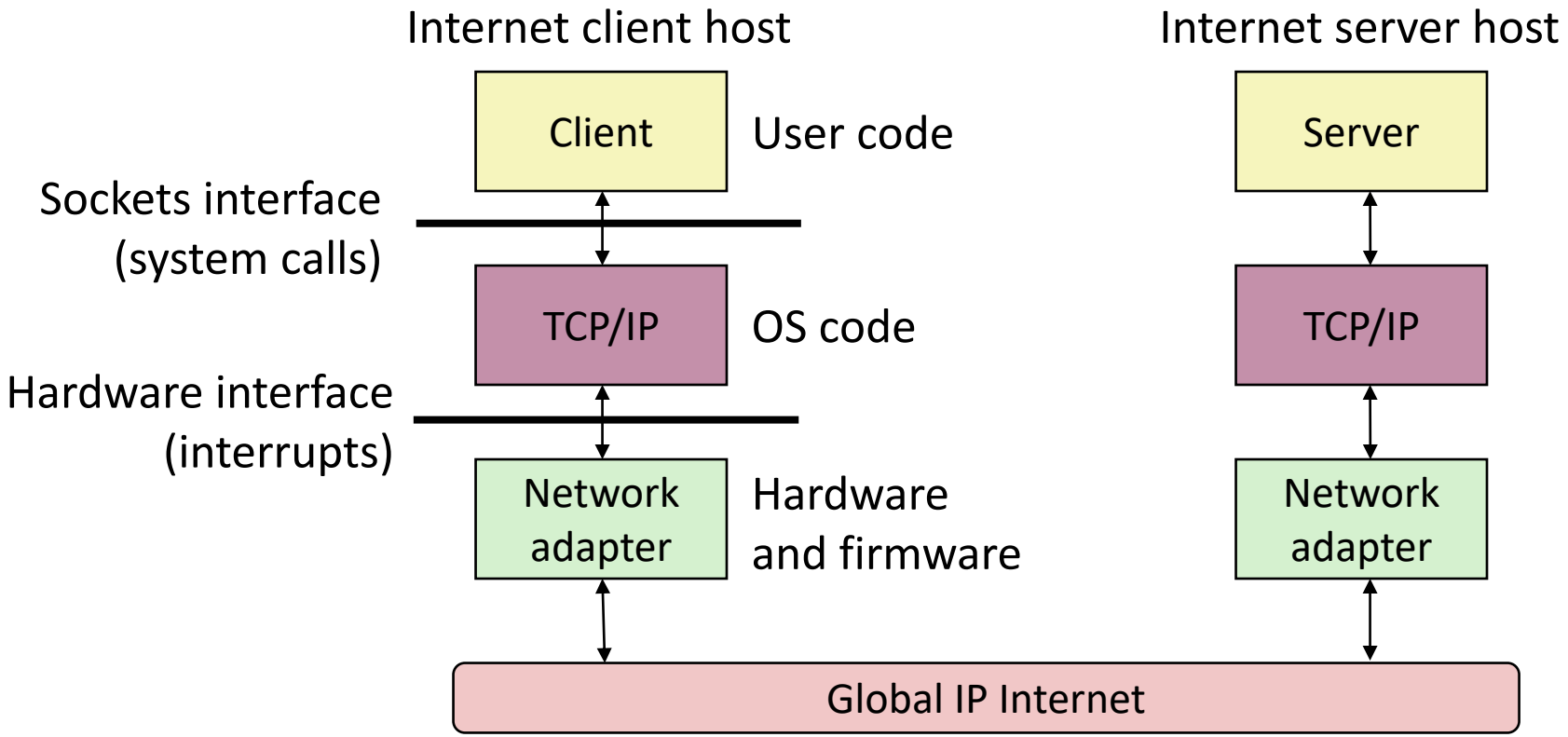
The global Internet

- Most famous example of an internet (uppercase to distinguish)

- Based on the TCP/IP protocol family
  - **IP** (Internet Protocol)
    - Provides a *naming scheme* and unreliable *delivery of packets* from **host-to-host**
  - **UDP** (Unreliable Datagram Protocol)
    - Uses IP to provide *unreliable data delivery* from **process-to-process**
  - **TCP** (Transmission Control Protocol)
    - Uses IP to provide *reliable data delivery* from **process-to-process**

- Accessed via a mix of Unix file I/O and 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**
   - 129.105.7.30

2. The set of IP addresses is mapped to a set of identifiers called Internet **domain names**
   - 129.105.7.30 is mapped to moore.wot.eecs.northwestern.edu

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

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)
    - Remember: most computers use little-endian😭
  - 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) */
};
```

- By convention, each byte in a 32-bit IP address is represented by its decimal value and separated by a period
  - IP address: `0x8169071E` = `129.105.7.30`

How many hosts can there be?

- IP addresses are 32-bits

How many hosts can there be?

- IP addresses are 32-bits
  - 2^32 = 4.3 billion hosts

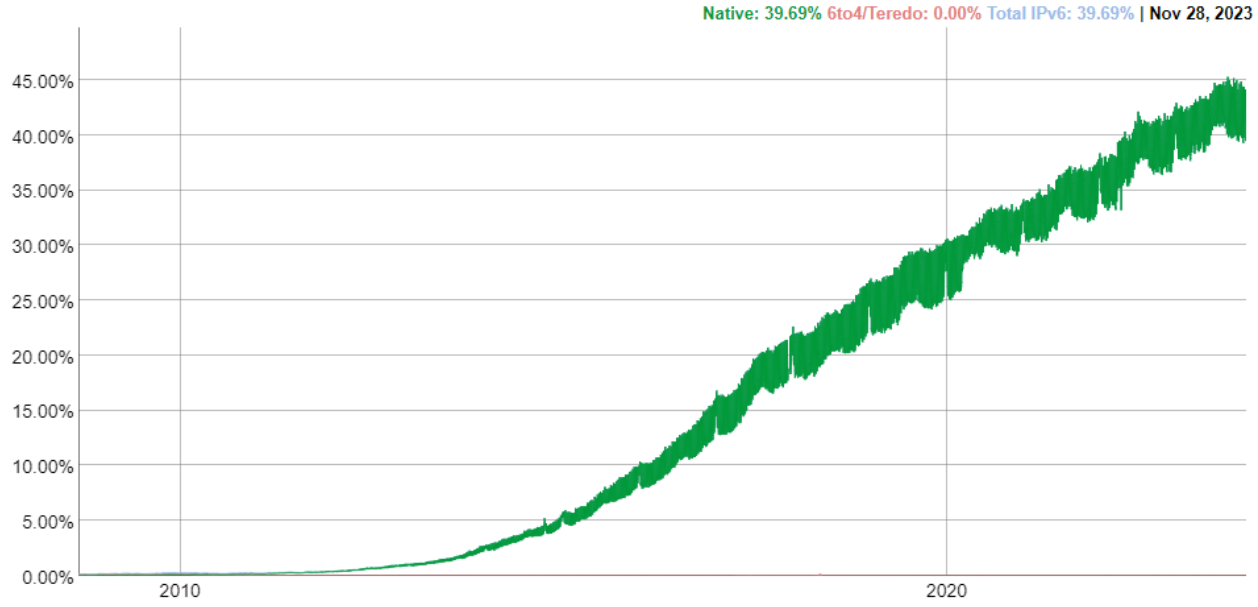
- **However,** some addresses are reserved for special purposes
  - Private networks (10.x.x.x, 192.168.x.x, …)
  - Within a computer (127.x.x.x)
  - Various testing or reserved purposes

- 588 million are reserved
  - So 3.7 billion **publicly accessible** hosts

- Typically, a home router has **one** publicly accessible IP address
  - All devices within the home are on a private network
  - Phones on cellular data are also within a private network

Aside: IPv4 and IPv6

- The original Internet Protocol, with its 32-bit addresses, is known as *Internet Protocol Version 4* (IPv4)
- 1996: Internet Engineering Task Force (IETF) introduced *Internet Protocol Version 6* (IPv6) with 128-bit addresses
  - Intended as the successor to IPv4
- Majority of Internet traffic still carried by IPv4 (we'll focus on it today)



IPv6 traffic at Google

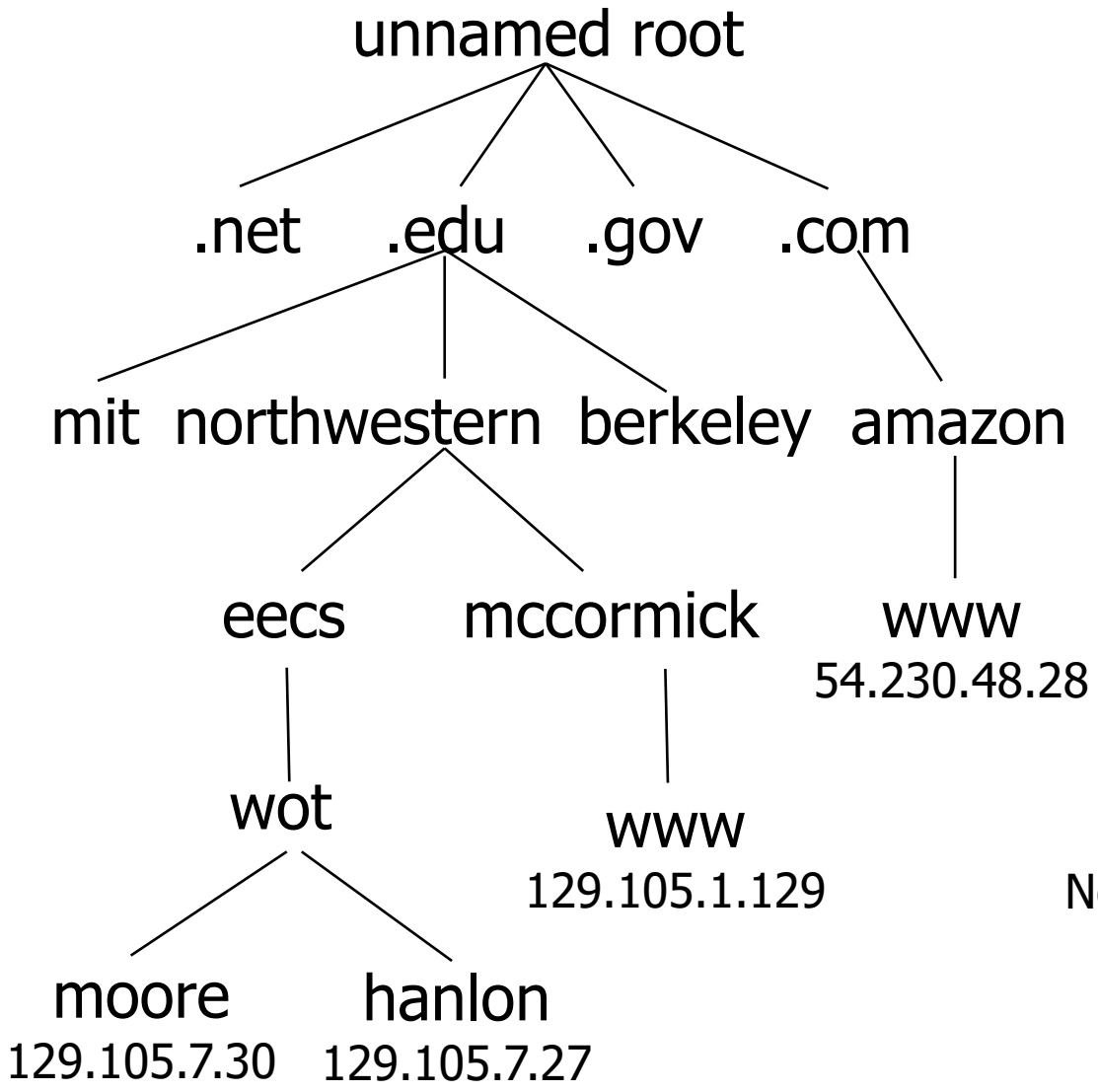
A programmer's view of the internet

1. Hosts are mapped to a set of 32-bit **IP addresses**
   - 129.105.7.30

2. The set of IP addresses is mapped to a set of identifiers called Internet **domain names**
   - 129.105.7.30 is mapped to moore.wot.eecs.northwestern.edu

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

2. Internet domain names



Top-level domain names

Second-level domain names

Third-level domain names and onwards…

Note: Northwestern owns 129.105.x.x

Some top-level domain names (TLDs)

.space	.store	.stream	.studio
.study	.style	.supplies	.supply
.support	.surf	.surgery	.sydney
.systems	.taipei	.tattoo	.tax
.taxi	.team	.tech	.technology
.tennis	.theater	.theatre	.tienda
.tips	.tires	.tirol	.today
.tokyo	.tools	.top	.tours
.town	.toys	.trade	.trading
.training	.tube	.university	.uno
.vacations	.vegas	.ventures	.versicherung
.vet	.viajes	.video	.villas
.vin	.vip	.vision	.vlaanderen
.vodka	.vote	.voting	.voto
.voyage	.wales	.wang	.watch
.webcam	.website	.wed	.wedding
.whoswho	.wien	.wiki	.win
.wine	.work	.works	.world
.wtf	.在线	.移动	.онлайн
.сайт	.קום	.opr	.中文网
.संगठन	.机构	.みんな	.游戏
.企业	.xyz	.yoga	.yokohama
.zone			

Domain Naming System (DNS)

- The Internet maintains a mapping between IP addresses and domain names in a huge 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

- A special name: **localhost**
  - Refers back to the computer being used (IP address 127.0.0.1)

A programmer's view of the internet

1. Hosts are mapped to a set of 32-bit **IP addresses**
   - 129.105.7.30

2. The set of IP addresses is mapped to a set of identifiers called Internet **domain names**
   - 129.105.7.30 is mapped to moore.wot.eecs.northwestern.edu

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

3. Internet connections

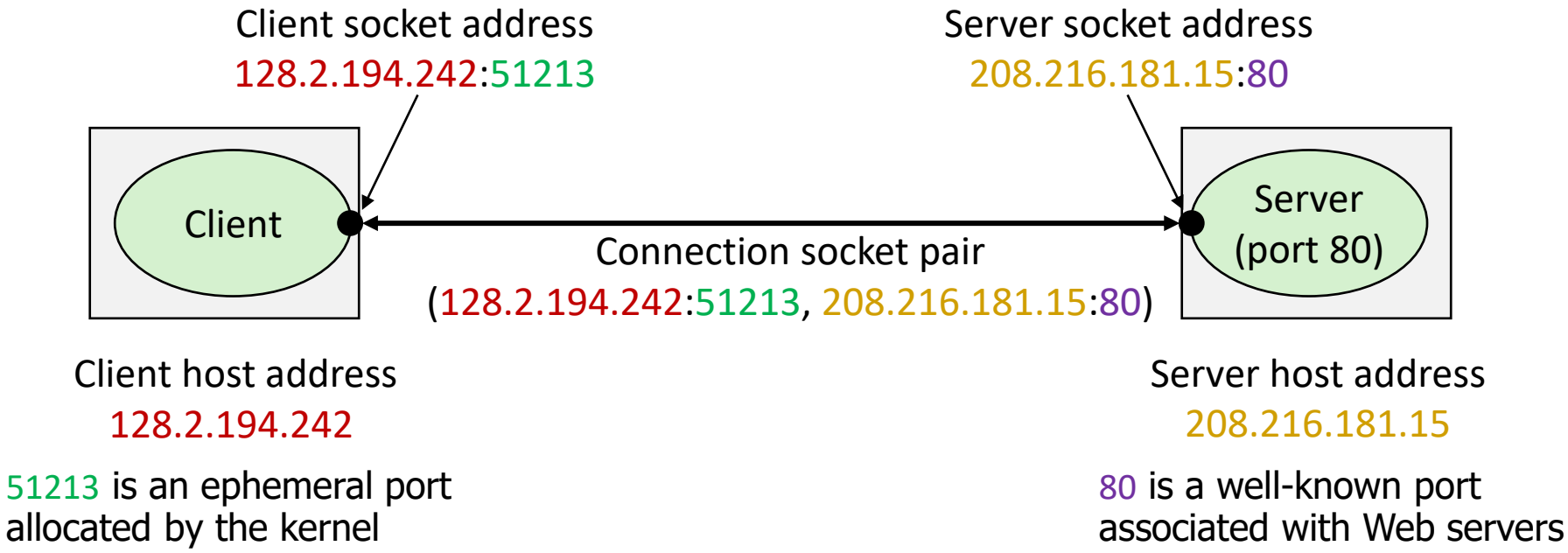
- Clients and servers communicate by sending streams of bytes over TCP **connections**. (There are other types, like UDP, but we'll focus on TCP)
  Each connection is:
  - Point-to-point: connects a pair of processes.
  - Full-duplex: data can flow in both directions at the same time,
  - Reliable: stream 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

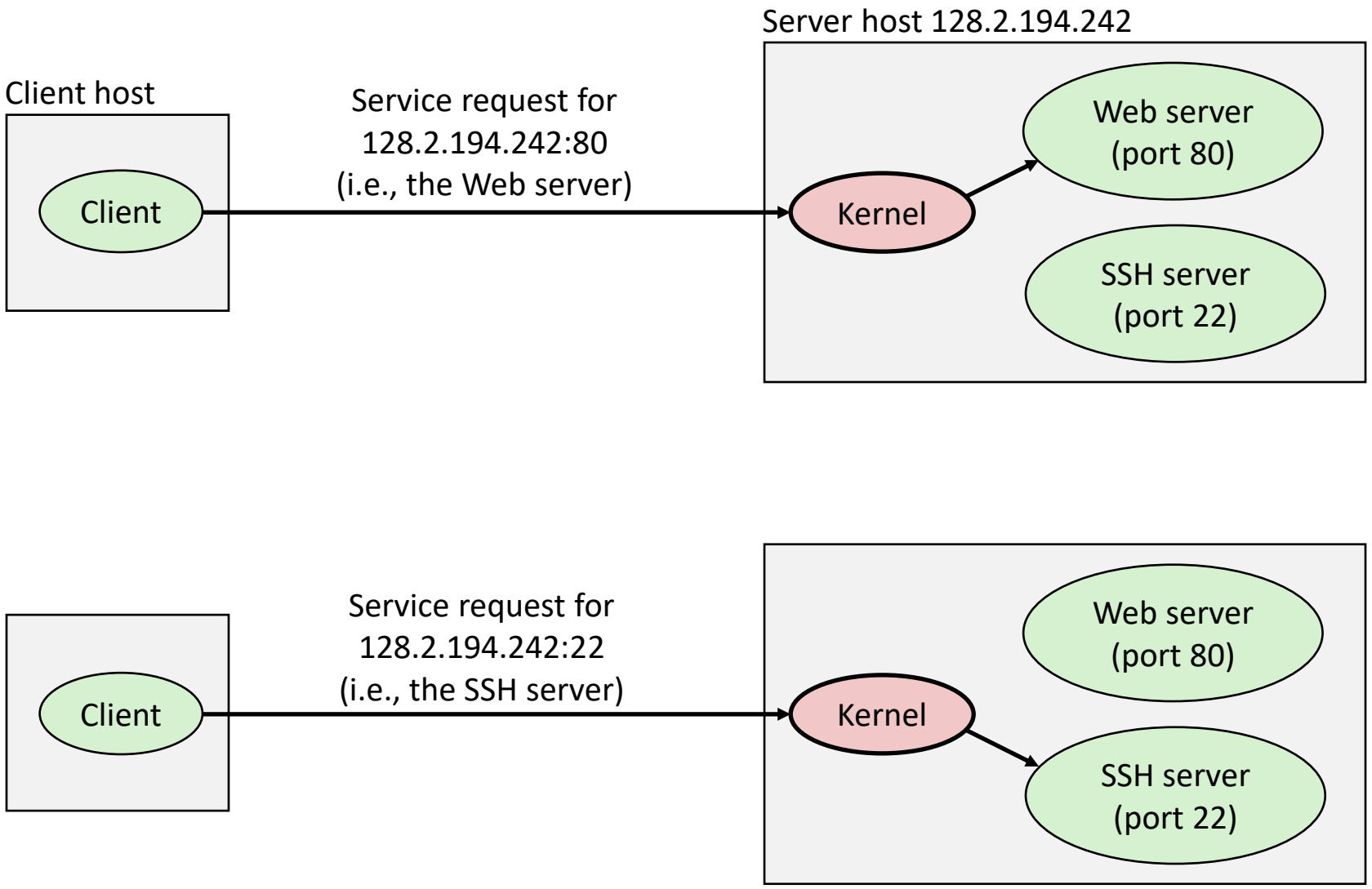
- A **port** is a 16-bit integer that identifies a process:
  - Ephemeral port
    - Assigned automatically by client kernel when client makes a connection request
    - Usually 1000 and up
  - Well-known port
    - Associated with some service provided by a server

Anatomy of a connection

- A connection is uniquely identified by socket addresses of endpoints
  - Socket address is an IP_address:Port pair
  - Connection: (client_address:client_port, server_address:server_port)



Ports are used to identify services to the kernel



Well-known service ports and names

- 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
  - Web servers: http 80

- Mappings between well-known ports and services are listed in `/etc/services` on Linux

Break + Thinking

- What are the steps for viewing a website?

1. You enter a domain name for the website

2. ???

Break + Thinking

- What are the steps for viewing a website?

1. You enter a domain name for the website
2. Computer looks up domain name to get IP Address
3. Computer sends request to IP_address:80
4. Computer gets back data, which it renders into a website

Outline

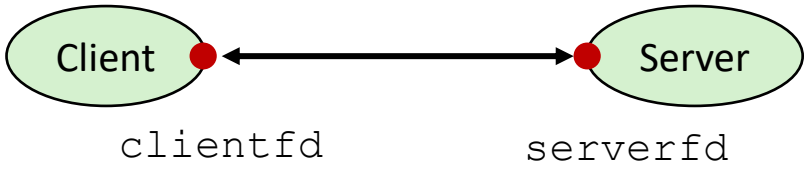
- Computer Networks
  - Topology
  - Inter-network communication
- The Internet
  - Protocol choices
- **Sockets**
  - **System calls for communicating with other computers**
- Web Servers

What is a socket?

- To an application, a socket is a file descriptor that lets the application read/write from/to the network
  - Connects a process on one host to a process on another

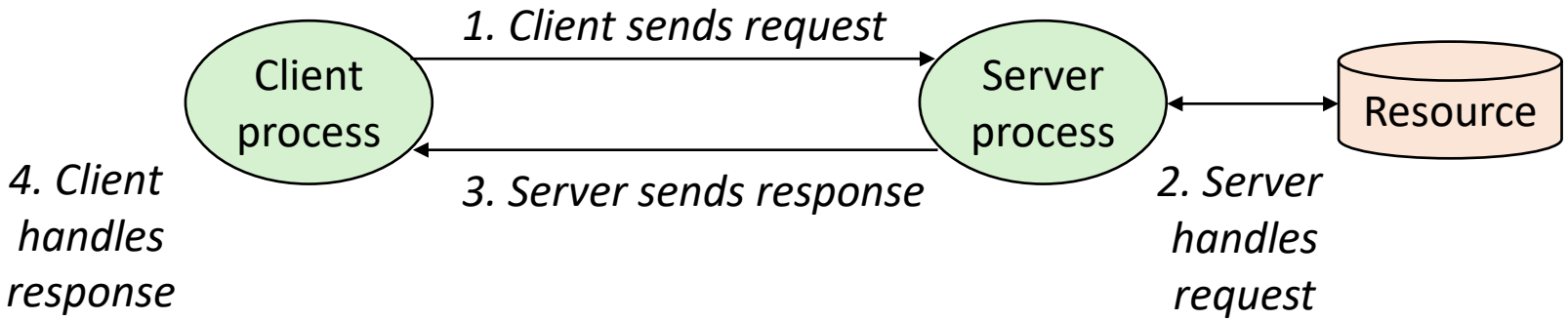
- Clients and servers communicate with each other by reading from and writing to socket descriptors

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



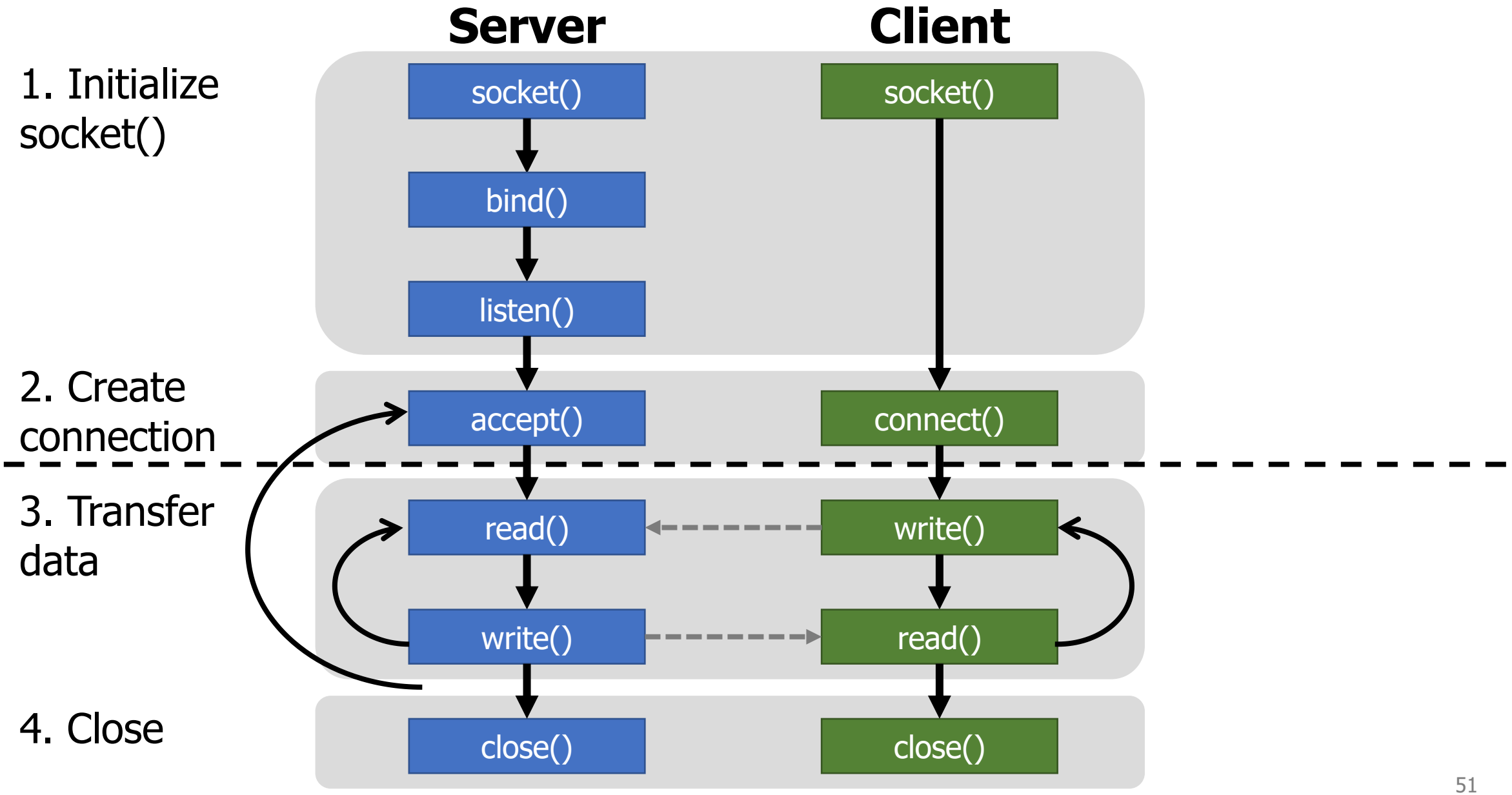
Client-Server transaction design pattern

- 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

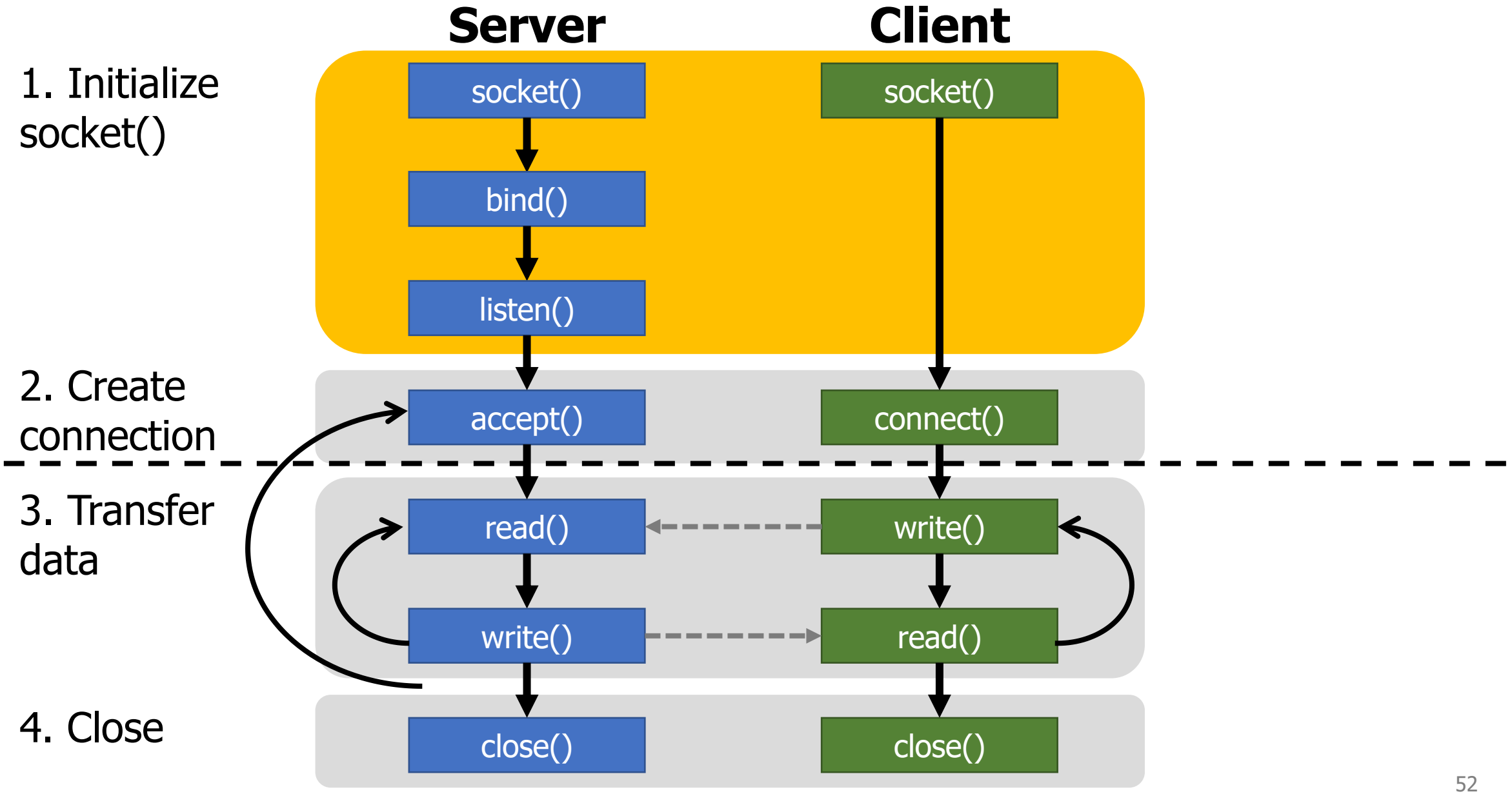


Note: clients and servers are processes running on hosts (can be the same or different hosts)

Socket communication flow



Socket communication flow



Creating a socket

- `int socket(int domain, int type, int protocol);`

- Domain: broad communication category
  - AF_INET for internet communication (Also: AF_INET6, AF_BLUETOOTH, etc.)

- Type: communication type within that category
  - SOCK_STREAM for TCP streams
  - SOCK_DGRAM for UDP datagrams

- Protocol: protocol ID within that type
  - Basically always 0

- Returns: a file descriptor for the socket

Is creating a socket enough?

- If we are a client
  - We will be connecting to a server
  - So no other steps are necessary
    - OS will just pick some arbitrary port to send on

- If we are a server
  - Some client will be connecting to us
  - We need to specify the IP_address:Port pair for the server
    - Pick any available port (or a well-known one)
    - Pick an IP address, if the computer has multiple
  - We also need to tell the OS we will be accepting messages

Picking an address and port for the socket

- **`int bind(int` *`sockfd`*`, const struct sockaddr *`*`addr`*`, socklen_t` *`addrlen`*`);`**

- Addr: a struct containing port and IP address

```
struct sockaddr_in  {
  uint16_t        sin_family;  /* Protocol family (always AF_INET) */
  uint16_t        sin_port;    /* Port num in network byte order */
  struct in_addr  sin_addr;    /* IP addr in network byte order */
  unsigned char   sin_zero[8]; /* Pad to sizeof(struct sockaddr) */
};
```

  - in_addr: just holds a uint32_t
  - INADDR_ANY selects all IP addresses

- Arguments and structs are intentionally generic to handle other protocols

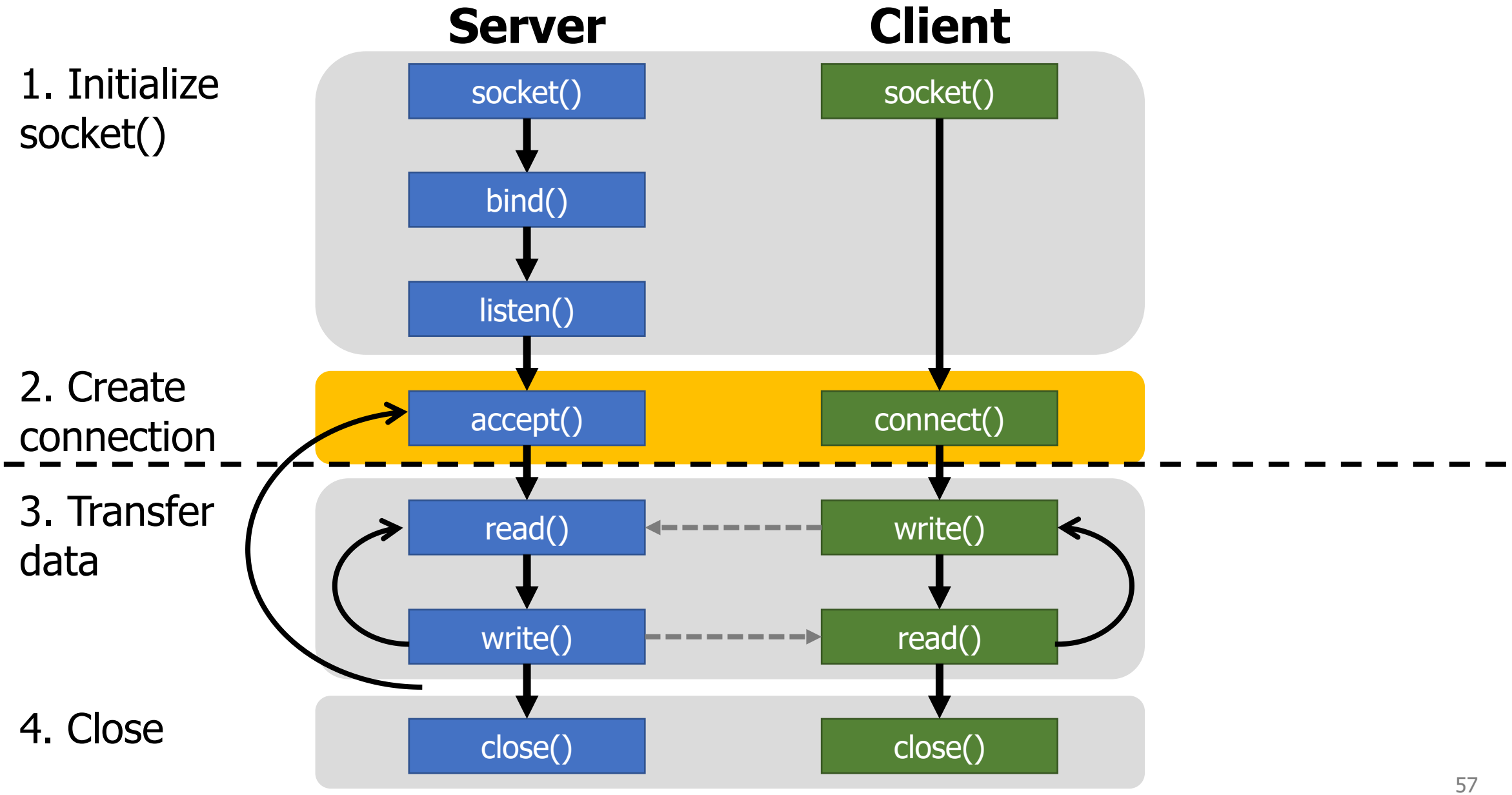
Inform the kernel that this process will receive packets

- `int listen(int sockfd, int backlog);`

- Backlog: maximum queue length for pending connections before they should be refused

- Note: this doesn't actually accept connections
  - Returns immediately

Socket communication flow



Starting a connection as a server

- `int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);`

- Blocks until a connection occurs
- Addr: Pointer to address structure, to be filled with client metadata

- Returns: a new file descriptor for this connection
  - New descriptor is used to perform data transfer
  - Original socket file descriptor remains functional
  - **Why?**

Starting a connection as a server

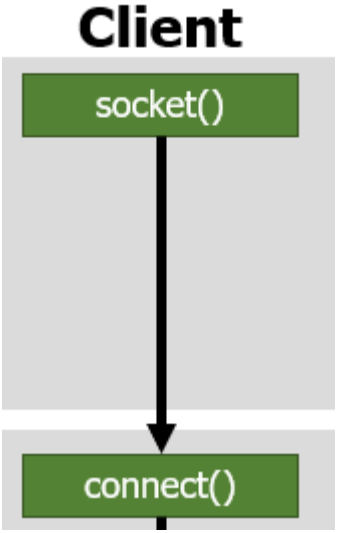
- `int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);`

- Blocks until a connection occurs
- Addr: Pointer to address structure, to be filled with client metadata

- Returns: a new file descriptor for this connection
  - New descriptor is used to perform data transfer
  - Original socket file descriptor remains functional
  - **Why?** To allow for additional connections to be made
    - Might accept multiple connections simultaneously
    - Separate threads might handle the created connections!

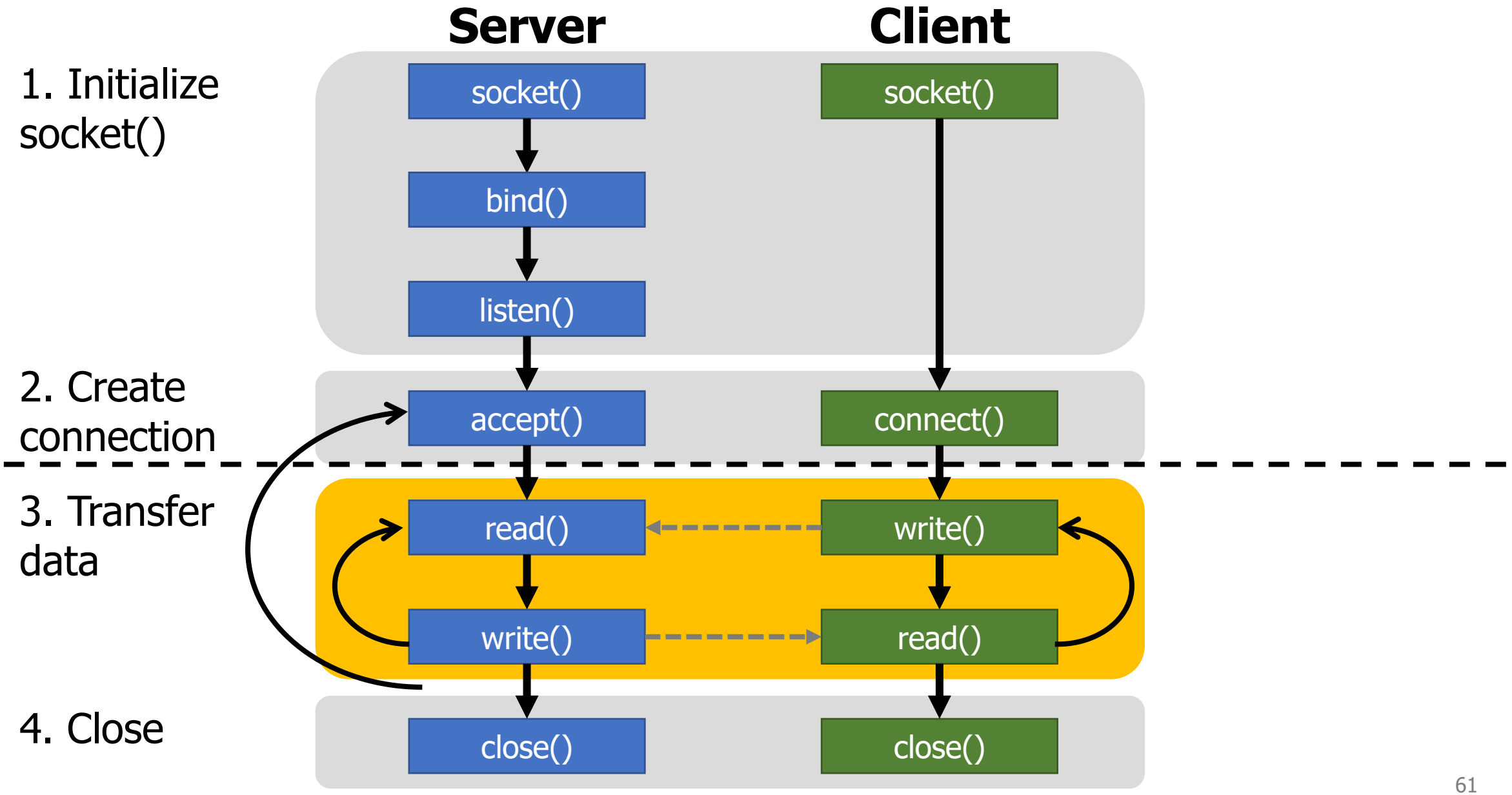
Starting a connection as a client

- `int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);`



- Blocks until a connection is made
- Addr: address of the server to connect to
  - Includes IP address and Port
  - Remember: all the client has done is created a socket(). This call specifies the location to connect the socket to.
- Only returns an error code. No need to reuse socket for client.

Socket communication flow



Socket-specific send/receive calls

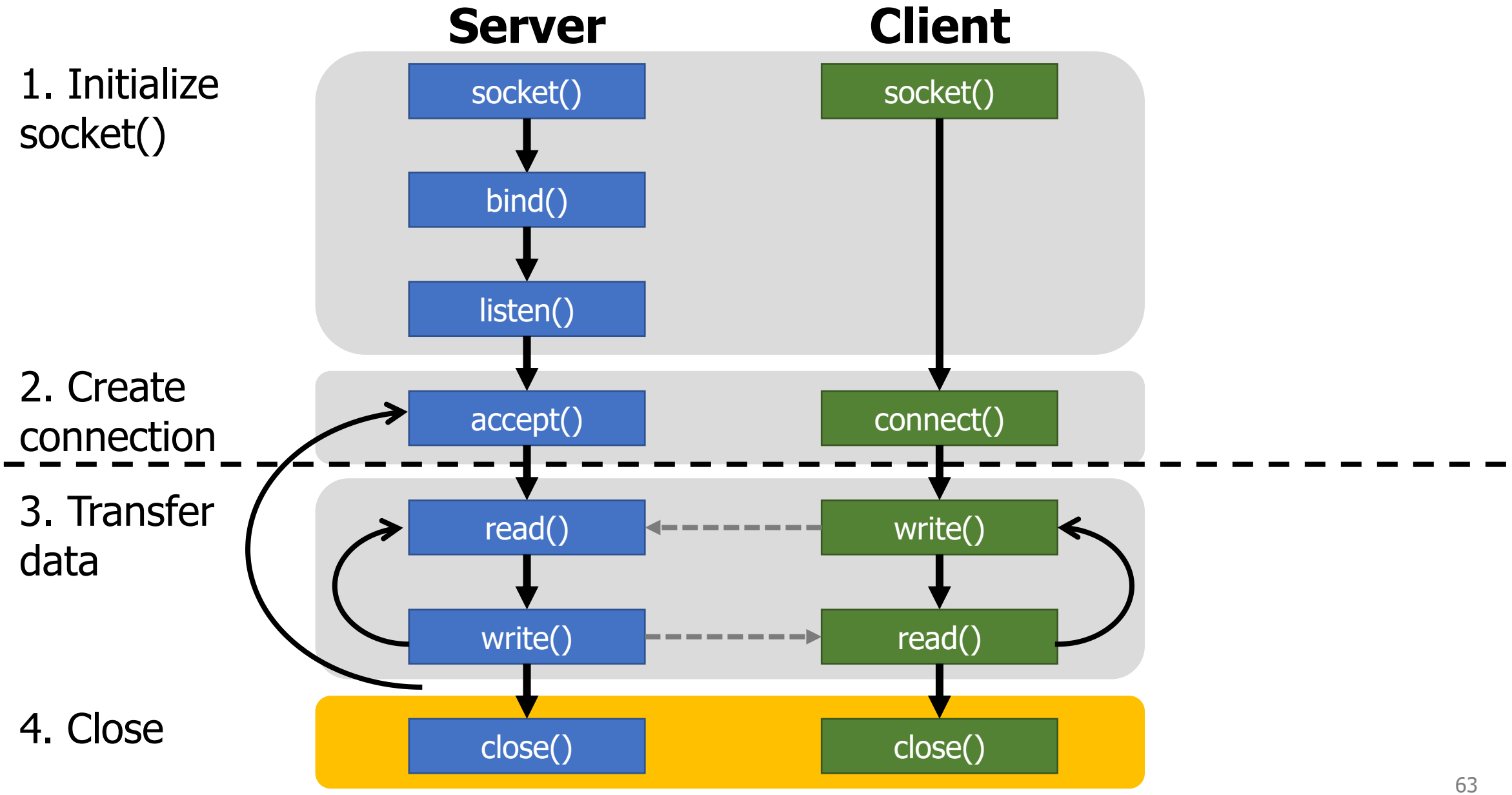
- The same as read/write for files except with additional flags
  - Blocks until data is completed

- `ssize_t recv(int sockfd, void *buf, size_t len, int flags);`
  - Receive data into an array (equivalent to read)

- `ssize_t send(int sockfd, const void *buf, size_t len, int flags);`
  - Send data from an array (equivalent to write)

- Flags: various networking-specific configurations
  - Wait for all requested data, or Never wait for data
  - Request confirmation of send, Hint that there will be more data after this
  - Etc.

Socket communication flow



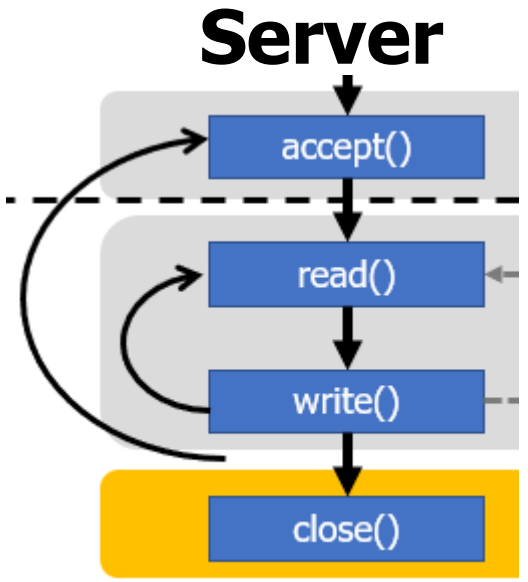
Closing the connection

- `int close(int fd);`

- Exactly the same close() system call from file I/O

- If other computer is waiting on a read/recv, gets end-of-file
  - Reads in data with a length of zero
  - Write/send gets an error

- Server might close connection-specific socket
  - And then call accept() again to handle new clients
  - Using the original, bind+listen socket



Example client and server code

- Adapted from: https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/
  - Warning: this code is written with terrible style
  - But is a good socket example

- Sidebar: should you write your own server in C?
  - That's just asking for trouble (see buffer overflow attacks)
  - But if you're a major web company and then you might need to for performance reasons

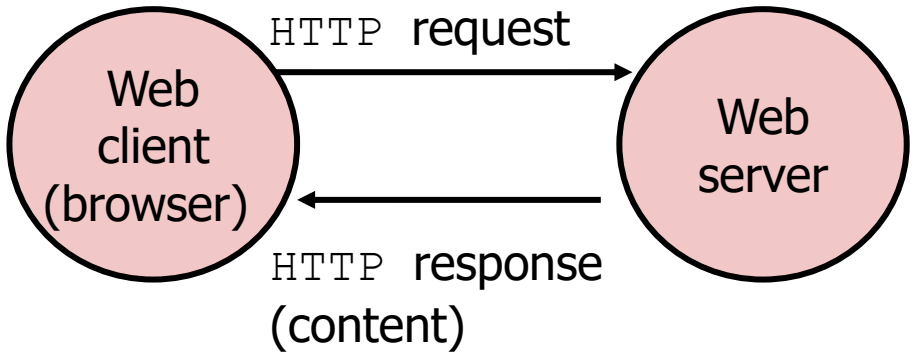
  - Generally: use some existing server library instead

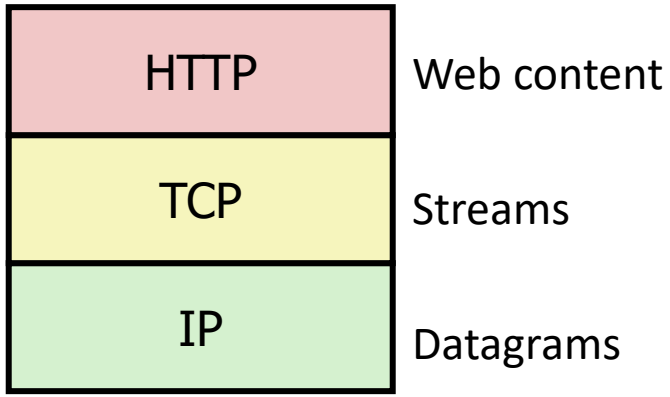
Outline

- Computer Networks
  - Topology
  - Inter-network communication
- The Internet
  - Protocol choices
- Sockets
  - System calls for communicating with other computers
- **Web Servers**

Web server basics

- Clients and servers communicate using HyperText Transfer Protocol (HTTP)
  1. Client and server establish TCP connection
  2. Client requests content
  3. Server responses with requested content
  4. Client and server close connection (eventually)





URLs and how they are used

- Unique name for a file: URL (Universal Resource Locator)
- Example URL: `http://www.northwestern.edu:80/index.html`
- Clients use prefix (`http://www.northwestern.edu:80`) to infer:
  - What kind (protocol) of server to contact (HTTP)
  - Where the server is (`www.northwestern.edu`)
  - What port it is listening on (80)
- Servers use suffix (`/index.html`) to:
  - Determine if request is for static or dynamic content.
    - No hard and fast rules for this
    - Although there are many conventions
  - Find file on file system
    - Initial "`/`" in suffix denotes home directory for requested content.
    - Minimal suffix is "`/`", which server expands to configured default filename (usually, `index.html`)

HTTP requests

- HTTP request is a request line, followed by zero or more request headers

- Request line: **`<method> <uri> <version>`**
  - **`<method>`** is one of **`GET`**, **`POST`**, **`PUT`**, **`DELETE`**, etc.
  - **`<uri>`** is typically URL for proxies, URL suffix for servers
    - A URL is a type of URI (Uniform Resource Identifier)
    - See http://www.ietf.org/rfc/rfc2396.txt
  - **`<version>`** is HTTP version of request (**`HTTP/1.0`** or **`HTTP/1.1`**)

- Request headers: **`<header name>: <header data>`**
  - Provide additional information to the server

HTTP responses

- HTTP response is a response line followed by zero or more response headers, possibly followed by content, with blank line ("`\r\n`") separating headers from content.

- Response line:

  `<version> <status code> <status msg>`
  - **<version>** is HTTP version of the response
  - **<status code>** is numeric status
  - **<status msg>** is corresponding English text
    - 200 OK Request was handled without error
    - 301 Moved Provide alternate URL
    - 404 Not found Server couldn't find the file
- Response headers: **`<header name>: <header data>`**
  - Provide additional information about response
  - **`Content-Type:`** MIME type of content in response body
  - **`Content-Length:`** Length of content in response body

Web content

- Web servers return *content* to clients
  - *content:* a sequence of bytes with an associated MIME type (Multipurpose Internet Mail Extensions) which describe how to interpret the data

- Example MIME types
  - `text/html` HTML document
  - `text/plain` Unformatted text
  - `image/gif` Binary image encoded in GIF format
  - `image/png` Binary image encoded in PNG format
  - `image/jpeg` Binary image encoded in JPEG format

List of MIME types: http://www.iana.org/assignments/media-types/media-types.xhtml

Static and dynamic content

- The content returned in HTTP responses can be either **static** or **dynamic**
  - Static content: content stored in files and retrieved in response to an HTTP request
    - Examples: HTML files, images, audio clips
    - Request identifies which content file
  - Dynamic content: content produced on-the-fly in response to an HTTP request
    - Example: content produced by a program executed by the server on behalf of the client
    - Request identifies name of content desired from executable code
- Bottom line: Web content is associated with a filename that is managed by the server

CS340 – Intro to Computer Networking

- Computer-to-Computer Communication
  - OSI Model
  - IPv4, IPv6, and Routing
  - TCP, UDP
  - Application Protocols: HTTP, FTP
  - Sockets and Web Servers

- A systems class that feels more *different* from CS213 rather than more similar
- Usually taught twice a year

Outline

- Computer Networks
  - Topology
  - Inter-network communication
- The Internet
  - Protocol choices
- Sockets
  - System calls for communicating with other computers
- Web Servers