Intro. to Computer Science :

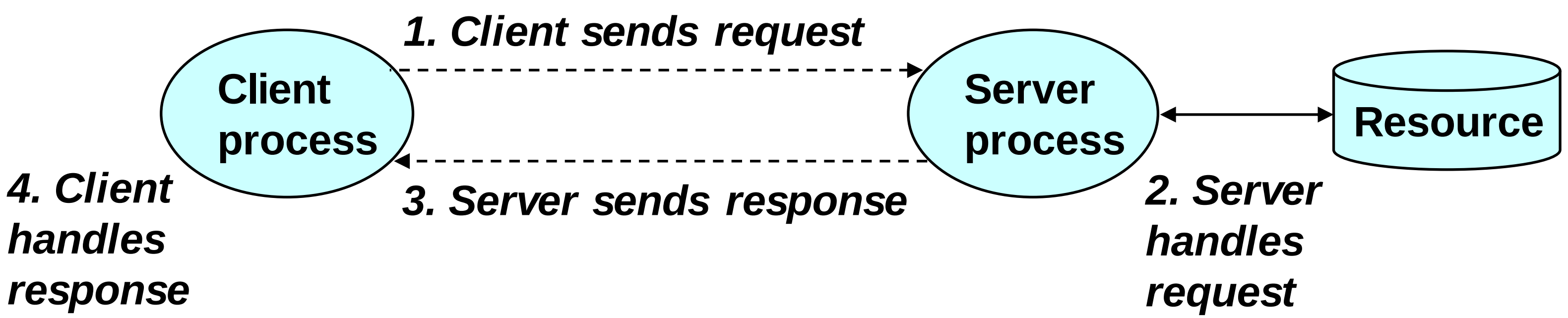
Network System

May 2020

Honguk Woo

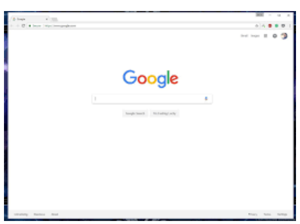
Client-Server Model

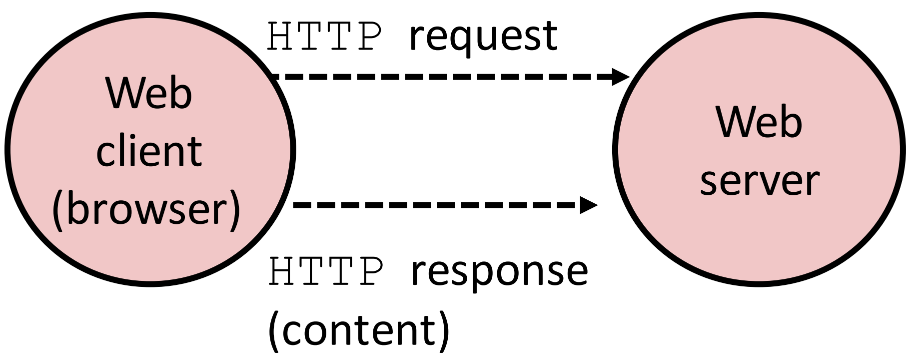
- Every network application is based on the client-server model:
  - A *server* process and one or more *client* processes
  - Server manages some *resource*
  - Server provides *service* by manipulating resource for clients
  - Clients and servers are processes running on hosts (can be the same or different hosts).



e.g., Web : Browser and Server

- Network Client and Server Model in Web Architecture
  - Web Browser : Google Chrome, MS explorer, Mozilla Firefox
  - Web Server : Web hosts for web services
    - Web server software : Apache HTTP server, MS IS, NGINX ….
  - Clients and servers communicate using the HyperText Transfer Protocol (HTTP)
    - Client and server establish TCP connection
    - Client requests content
    - Server responds with requested content
    - Client and server close connection (eventually)





content: a sequence of bytes with an associated MIME (Multipurpose Internet Mail Extensions) type

<u>Example MIME types</u>

`text/html` HTML document
`text/plain` Unformatted text
`image/gif` Binary image encoded in GIF format
`image/png` Binar image encoded in PNG format
`image/jpeg` Binary image encoded in JPEG format

e.g., Web : 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 file containing executable code
- *Web content is associated with a file that is managed by the server*

e.g., Web : Resources

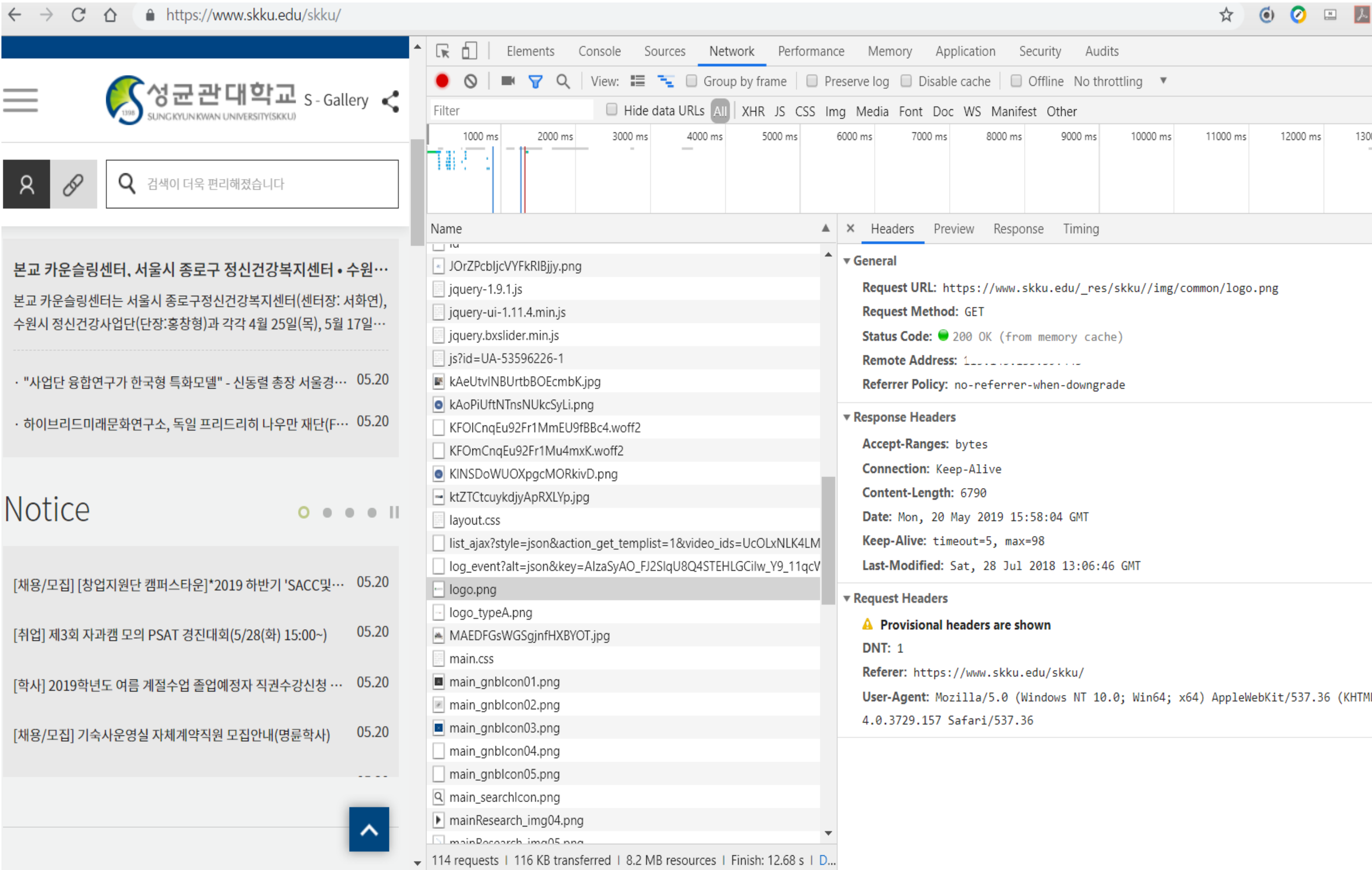
- Unique name for a *file*: URL (Universal Resource Locator)
- Example URL: http://www.host.edu:80/index.html
    - Protocol: The application-level protocol used by the client and server, e.g., HTTP
    - Hostname: The DNS domain name of the server
    - Port: The TCP port number that the server is listening for incoming requests from the clients
    - Path-and-file-name: The name and location of the requested resource, under the server document base directory

- Use *prefix* (http://www.host.edu:80) to infer:
    - What kind (protocol) of server to contact (HTTP)
    - Where the server is (www.host.edu)
    - What port it is listening on (80)

- Use *suffix* (/index.html) to:
    - 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)

e.g., Web : Resources – skku site

Chrome -> dev tool -> network



e.g., Web : Resources – skku site

- An HTTP client sends an HTTP request to a server in the form of a request message which includes following format
  - Request-line : **GET** https://www.skku.edu/skku/index.do HTTP/1..1
  - Zero or more header fields
  - An empty line indicating the end of the header fields
  - Optionally a message-body

  - *Try : How to view HTTP headers in Google Chrome?*
    ① In Chrome, visit a URL, right click, select Inspect to open the developer tools
    ② Select Network tab
    ③ Reload the page, select any HTTP request on the left panel, and the HTTP headers will be displayed on the right panel

    - *https://www.mkyong.com/computer-tips/how-to-view-http-headers-in-google-chrome/*

e.g., Web : sample HTTP request, response

- HTTP Request

GET /path/file.html HTTP/1.1
Host: www.host1.com:80
User-Agent: Mozilla/3.0
[blank line here]

- HTTP Response

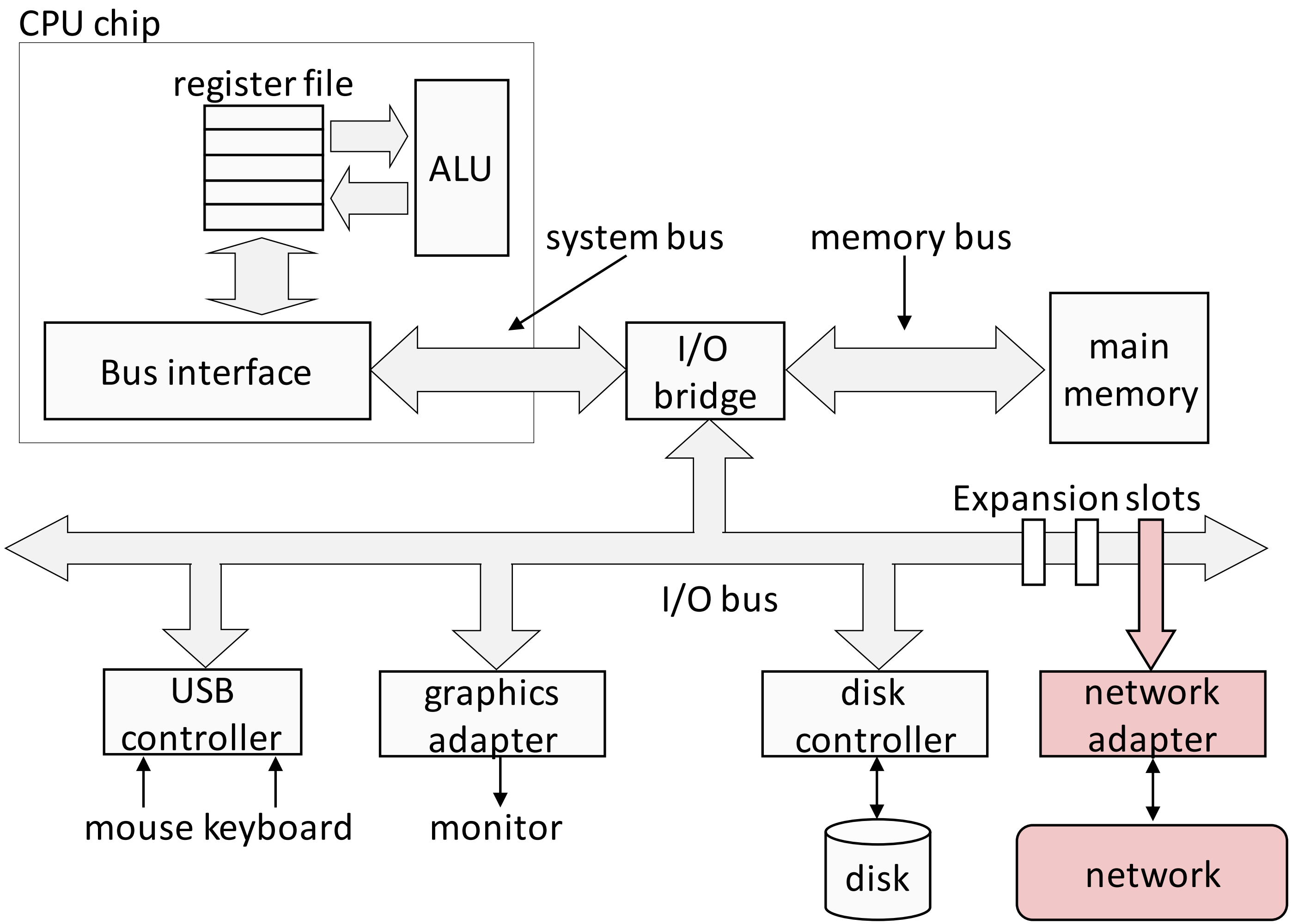
HTTP/1.1 200 OK
Date: Fri, 31 Dec 201x 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html>
<body>
<h1>Happy New Millennium!</h1>
(more file contents) . . . </body>
</html>

History of Web

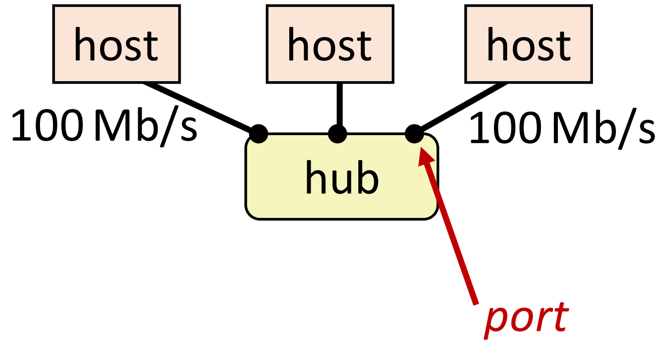
- https://home.cern/science/computing/birth-web/short-history-web
- https://www.lopezferrando.com/a-brief-history-of-the-web/

Hardware Organization of a Network Host



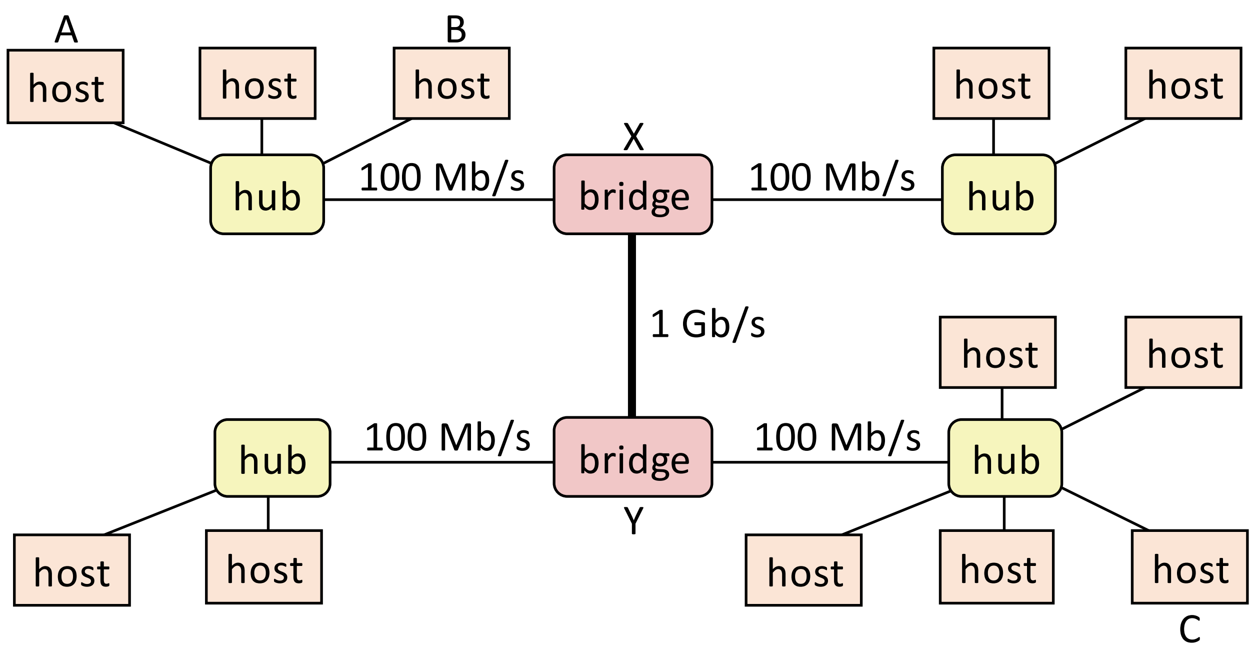
LAN (Local Area Network) - Ethernet Segment

- **Ethernet segment** consists of a collection of *hosts* connected by wires 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
  - Hosts send bits to any other host in chunks called ***frames***
  - **Hub copies each bit from each port to every other port**
    - Every host sees every bit



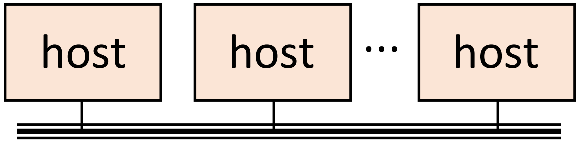
LAN - Bridged Ethernet Segment

- Spans building or campus
- *Bridges* cleverly learn which hosts are reachable from which ports and then selectively copy frames from **port to port**
  - Bridge looks at the destination before forwarding unlike a hub



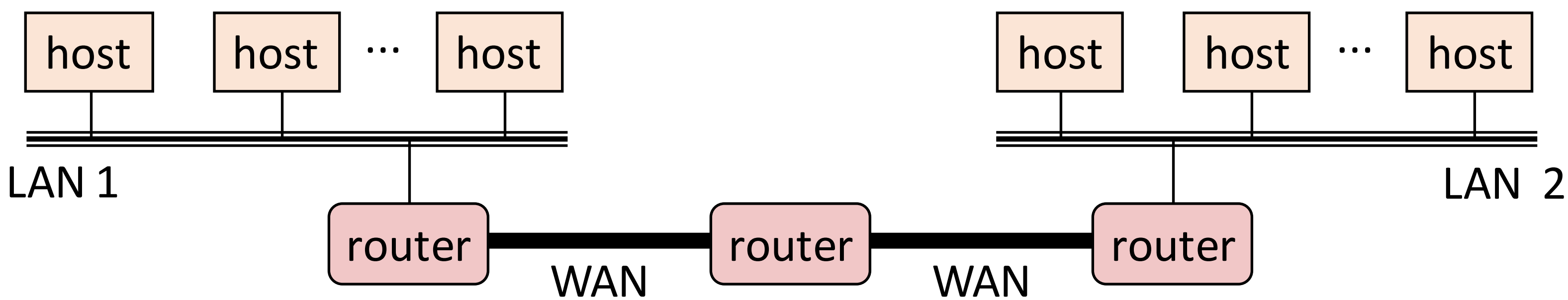
LAN – Conceptual View

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



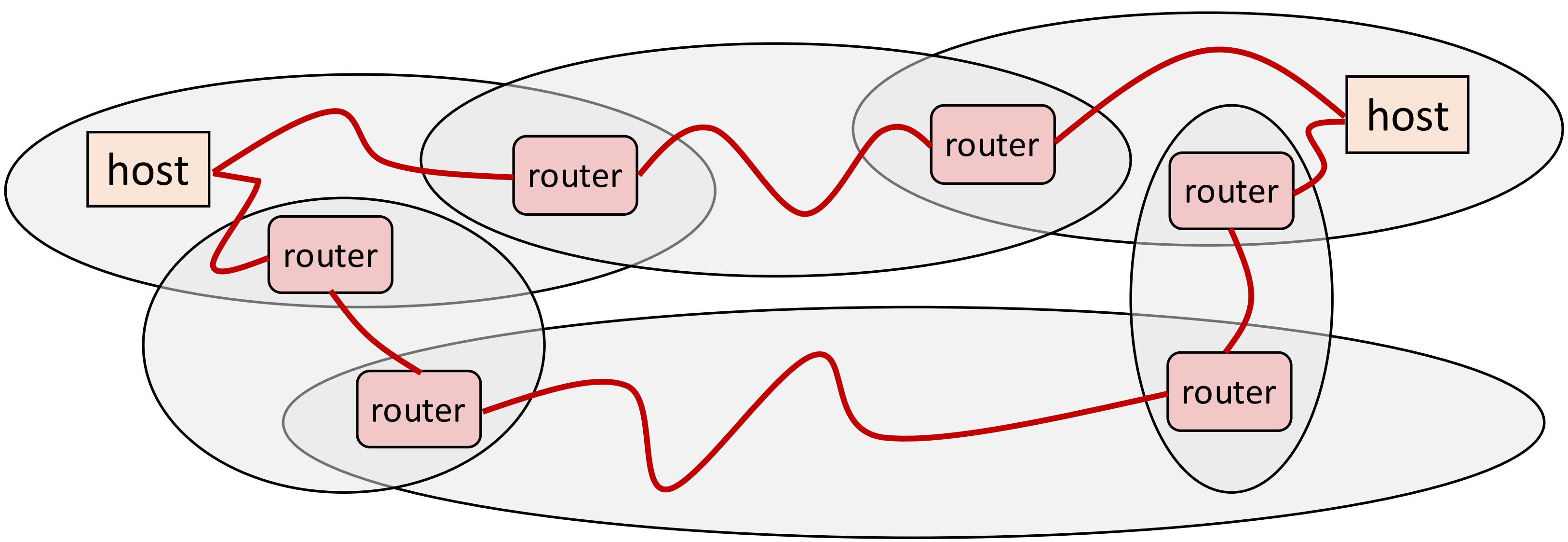
Next Level: **internets**

- Multiple incompatible LANs can be physically connected by specialized computers called *routers*
- The connected networks are called an *internet* (lower case)
  - *Internetwork (internet)*
  - The Global IP Internet (uppercase "I") is the most famous example of an internet (lowercase "i")
- *What's AP (access point) at your home ?*



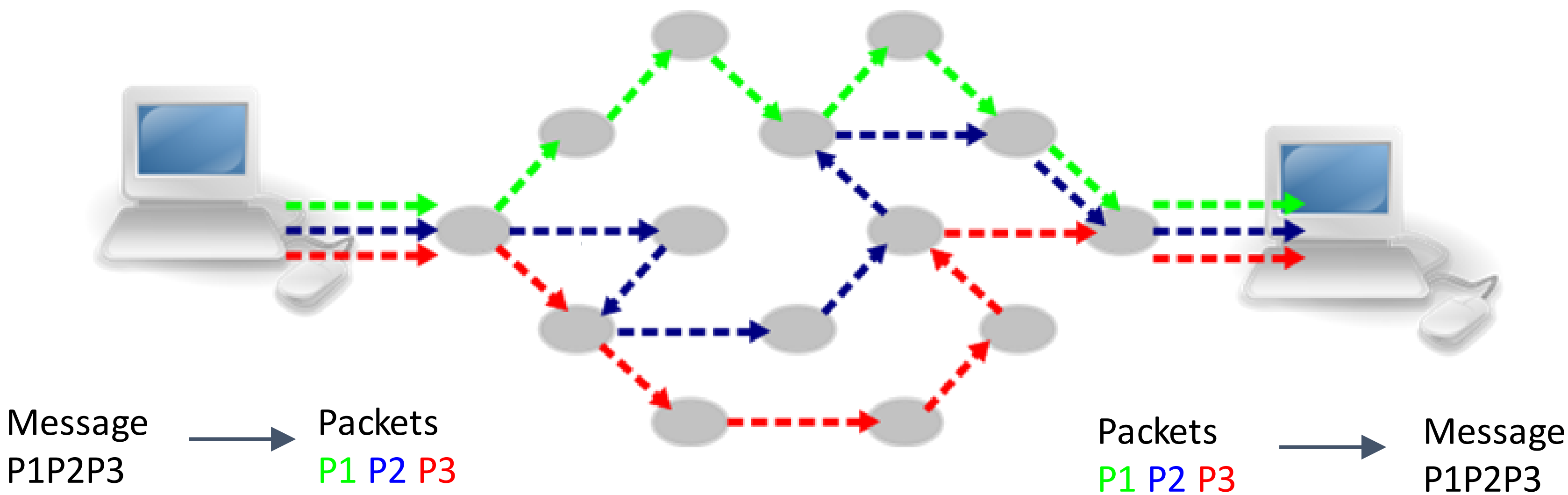
Logical Structure of an internet

- 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



Logical Structure of an internet : why ?

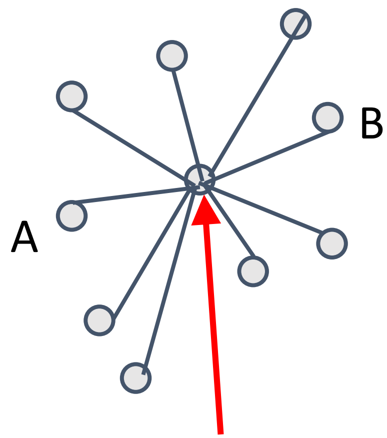
- Packet switching
  - Messages are broken into fixed-sized packets, Packets are routed independently, and the original message is re-assembled at the destination



Logical Structure of an internet : why ?

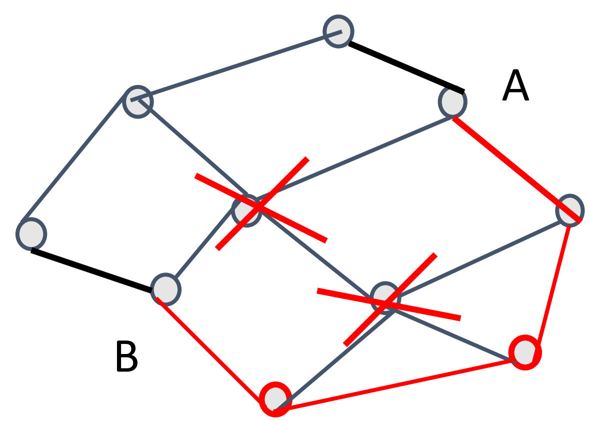
- A **decentralized** network with multiple paths between points A and B
  - Decentralized networks with redundant paths provide **robustness** in network design
  - Dividing the message into small packets that are routed independently
  - Each router along the path forwards packets to another router along the path

Centralized Network



Vulnerability

Decentralized Network



Robustness

The Notion of an internet Protocol

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

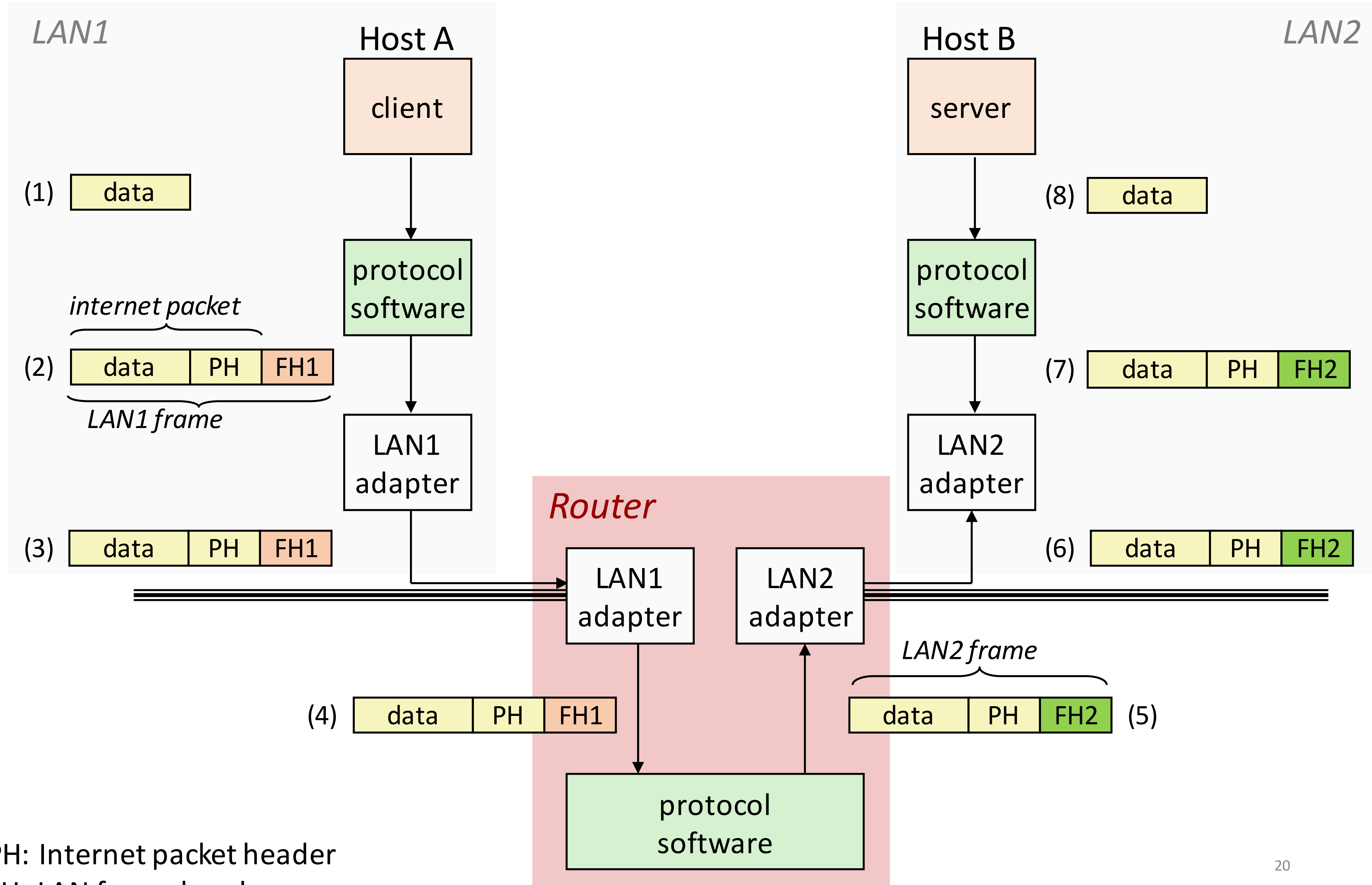
- Solution: *protocol* software running on each host and router
  - Protocol is a set of rules that governs how hosts and routers should cooperate when they transfer data from network to network.
  - Smooths out the differences between the different networks

Role of Internet Protocol

- 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 information such as packet size, source and destination addresses.
    - Payload: contains data bits sent from source host.

Transferring internet Data Via Encapsulation



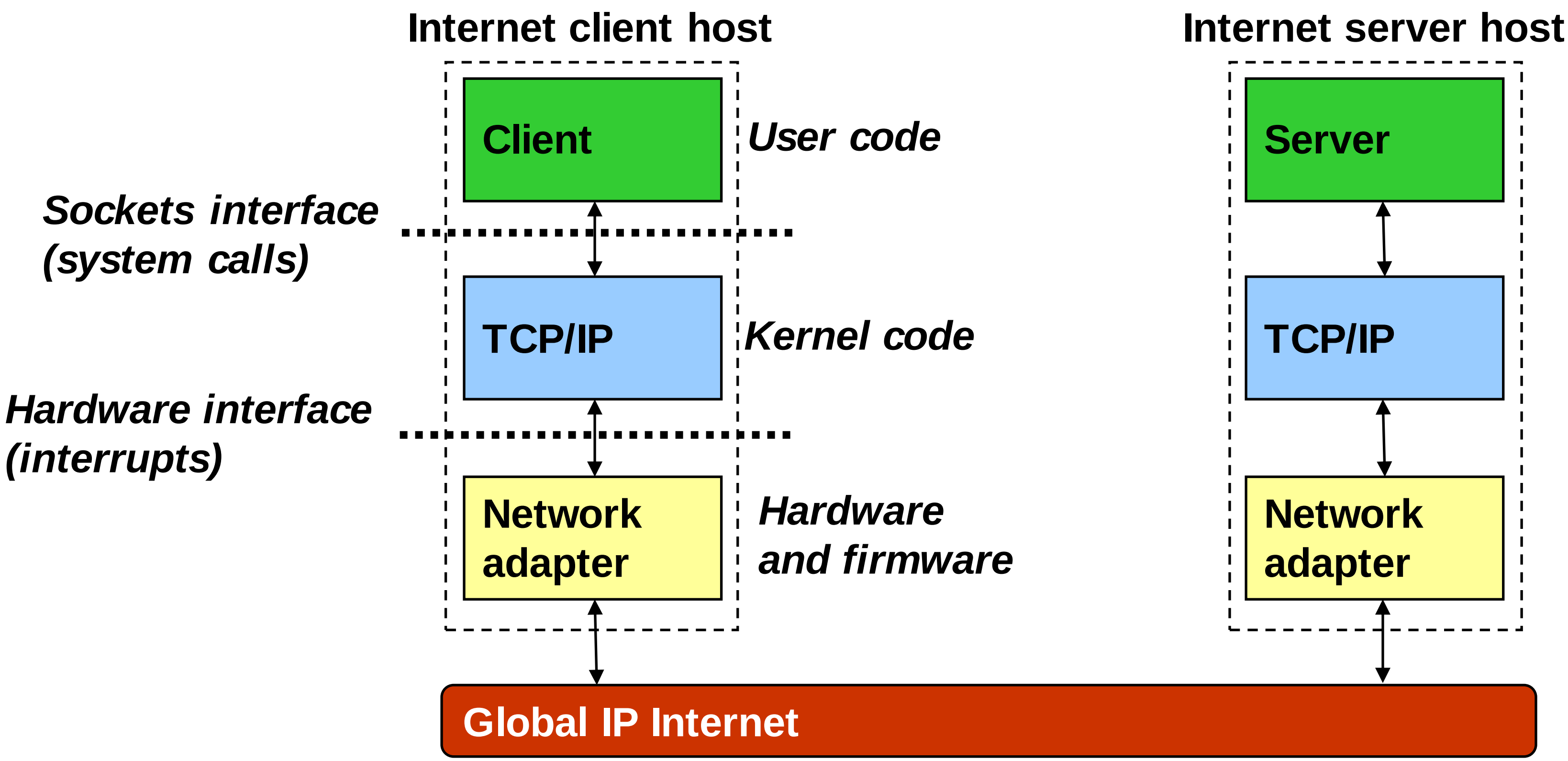
PH: Internet packet header
FH: LAN frame header

The Internet

- 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 (Unreliable 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*

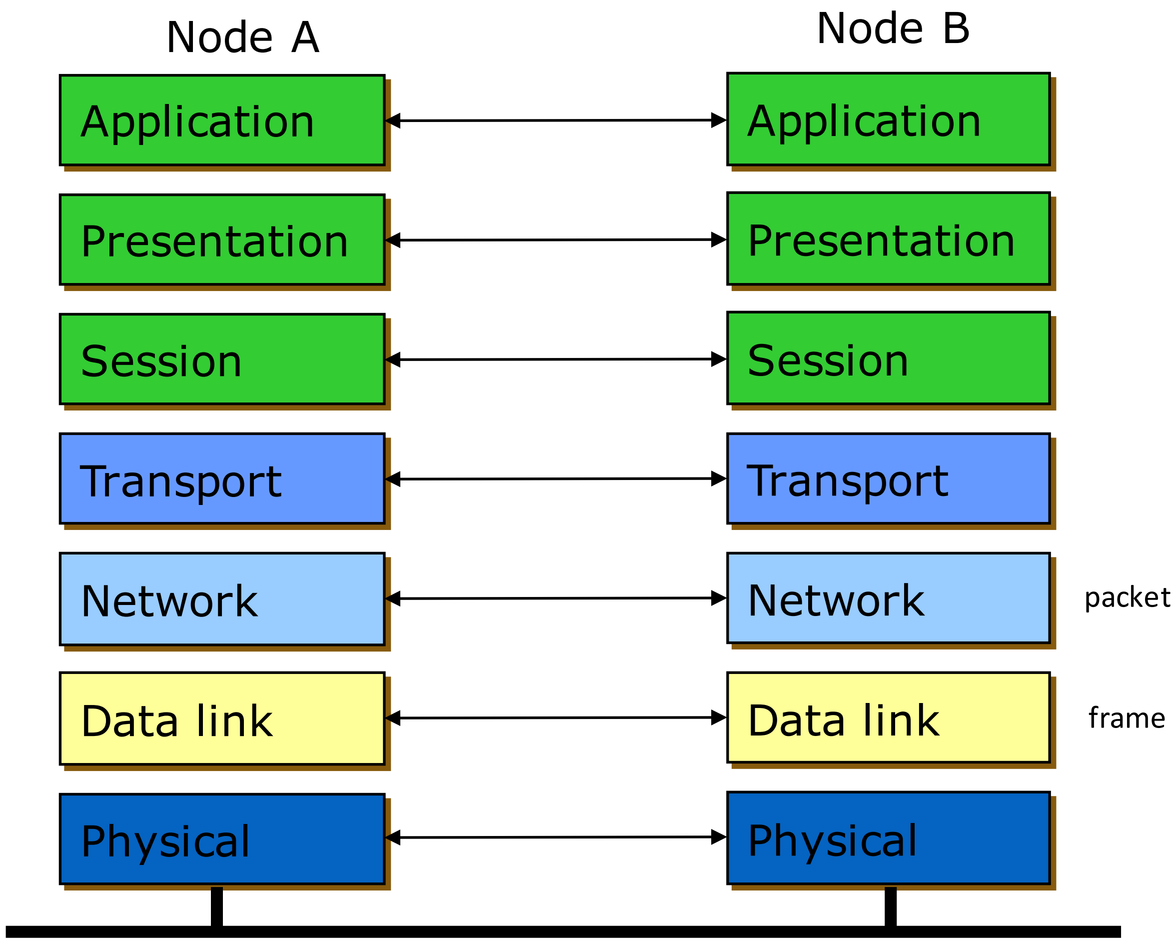
Organization of an Internet Application

- Hardware + Software



OSI reference model

- OSI (open system interconnection model)



A Programmer’s View of the Internet

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

- 115.145.129.40

(2) The set of IP addresses is mapped to a set of identifiers called Internet *domain names*

- 115.145.129.40 is mapped to www.skku.edu

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

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

  - As of 2015, vast majority of Internet traffic still carried by IPv4

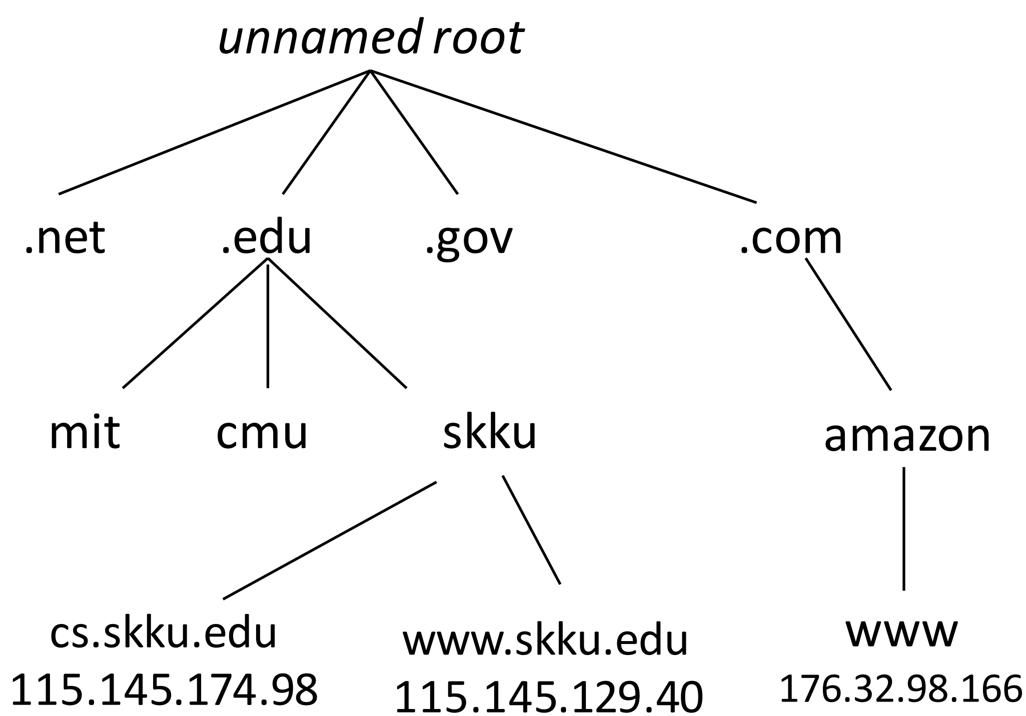
(1) IP Addresses

- 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.

- 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`**

(2) Internet Domain Names

- More human-friendly domain names



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.
  - In a mathematical sense, a host entry is an equivalence class of domain names and IP addresses.

(3) Internet Connections

- Clients and servers communicate by sending streams of bytes over *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)

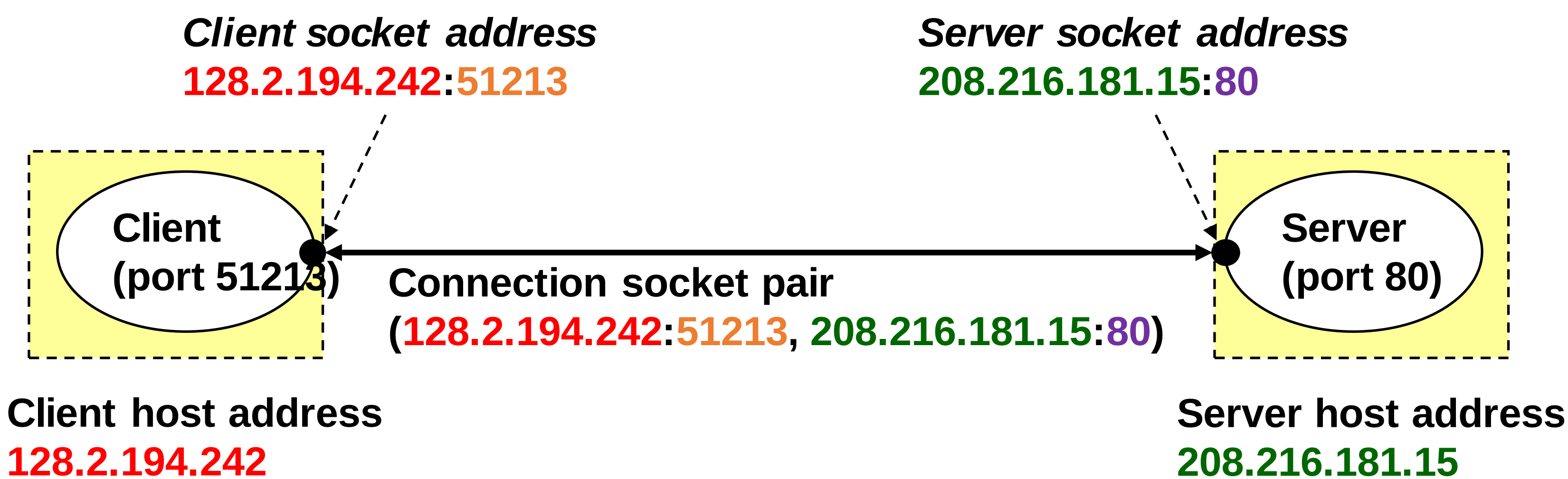
Well-known Ports and Service Names

- Popular services have permanently assigned *well-known ports and* corresponding *well-known service names*:
  - echo server: 7/echo
  - ssh servers: 22/ssh
  - email server: 25/smtp
  - Web servers: 80/http

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

Internet Connections

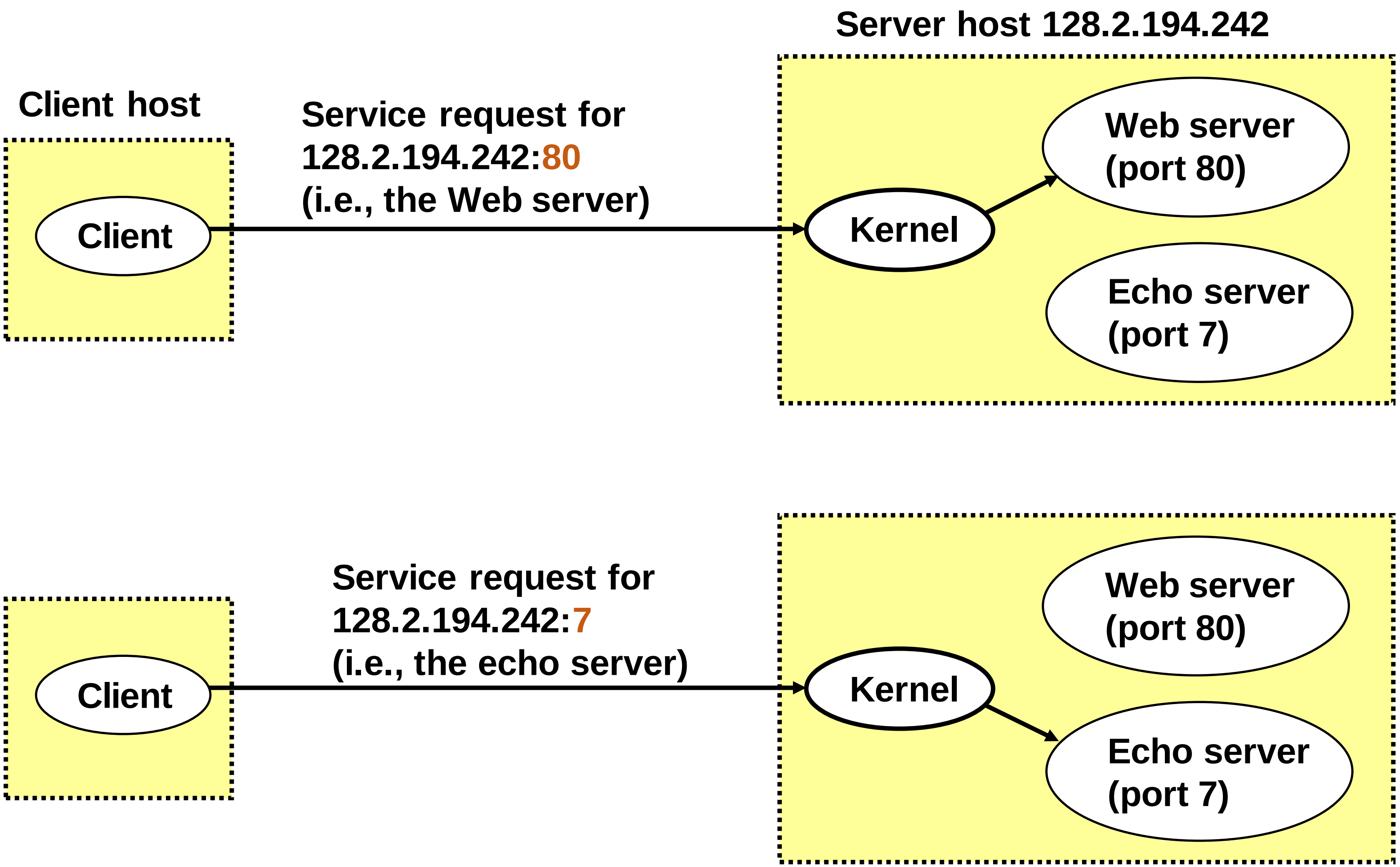
- A connection is uniquely identified by the socket addresses of its endpoints (*socket pair*)
  - **`(cliaddr:cliport, servaddr:servport)`**



Note: 51213 is an ephemeral port allocated by the kernel

Note: 80 is a well-known port associated with Web servers

Using Ports

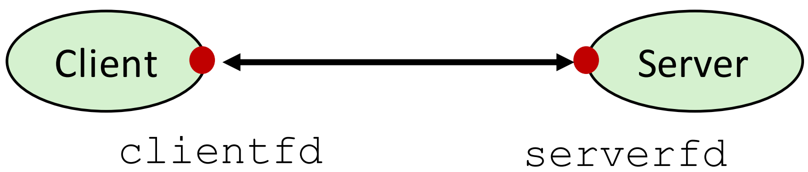

Server host 128.2.194.242
Client host
Service request for
128.2.194.242:80
(i.e., the Web server)
Client
Kernel
Web server
(port 80)
Echo server
(port 7)
Service request for
128.2.194.242:7
(i.e., the echo server)
Client
Kernel
Web server
(port 80)
Echo server
(port 7)

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

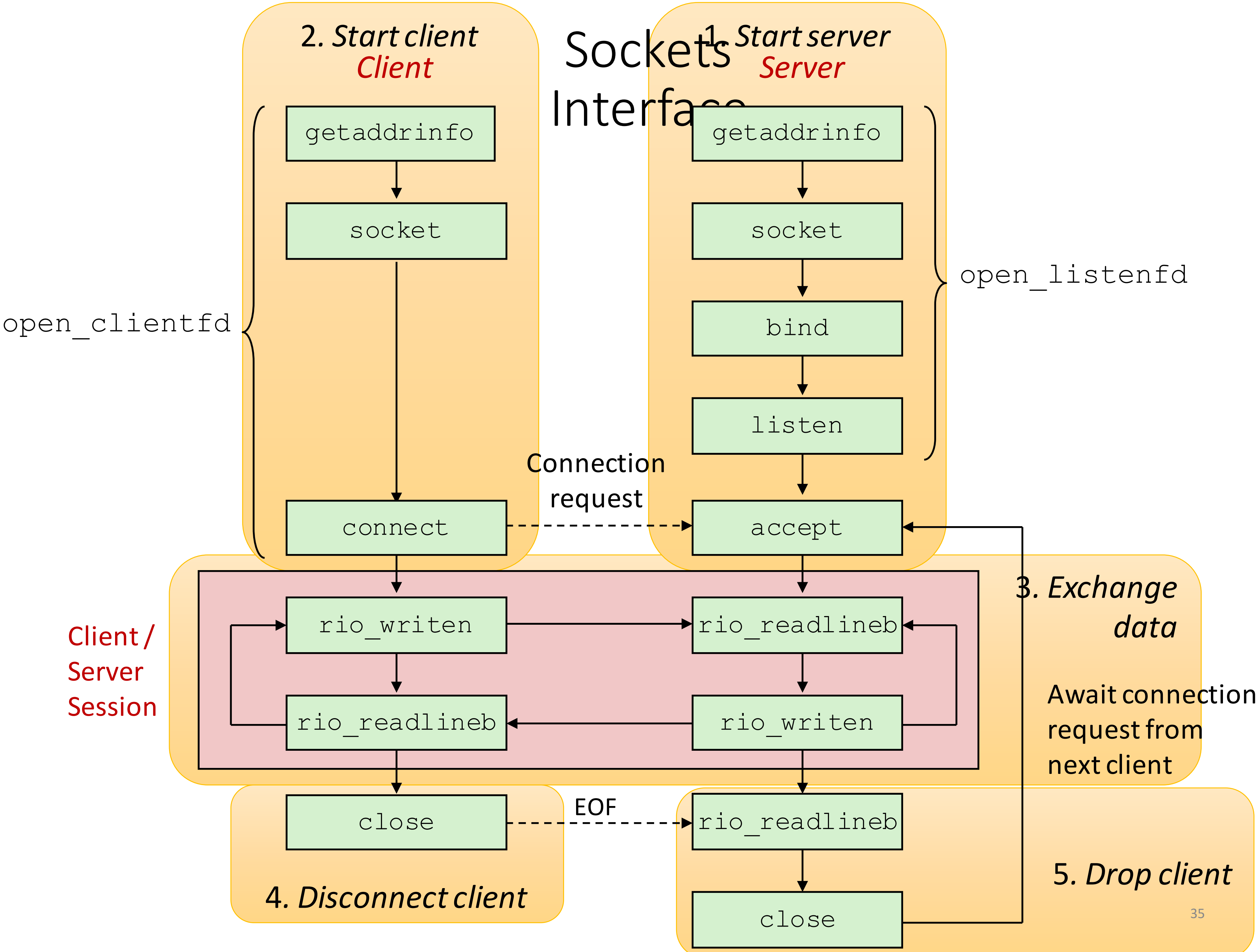
Sockets

- What is a socket?
  - To the kernel, a socket is an **endpoint** of communication (connection)
  - To an application, a socket is **a file descriptor (an open file w/ a corresponding descriptor)** that lets the application read/write from/to the network
    - ***Remember:*** All Unix I/O devices, including networks, are modeled as files
- 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

Sockets Interface



Discussion : Explain how system calls are used to implement this echo server
Echo Server (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#define MAXLINE 80

int main (int argc, char *argv[]) {
   int n, listenfd, connfd, caddrlen;
   struct hostent *h;
   struct sockaddr_in saddr, caddr;
   char buf[MAXLINE];
   int port = atoi(argv[1]);

   if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
      printf("socket() failed.\n");
      exit(1);
   }

   bzero((char *)&saddr, sizeof(saddr));
   saddr.sin_family = AF_INET;
   saddr.sin_addr.s_addr = htonl(INADDR_ANY);
   saddr.sin_port = htons(port);
```

Echo Server (2)

```
if (bind(listenfd, (struct sockaddr *)&saddr,
         sizeof(saddr)) < 0) {
   printf("bind() failed.\n");
   exit(2);
}

if (listen(listenfd, 5) < 0) {
   printf("listen() failed.\n");
   exit(3);
}

while (1) {
   caddrlen = sizeof(caddr);
   if ((connfd = accept(listenfd, (struct sockaddr *)&caddr,
                         &caddrlen)) < 0) {
      printf ("accept() failed.\n");
      continue;
   }
```

Echo Server (3)

```
        // echo
        while ((n = read(connfd, buf, MAXLINE)) > 0) {
           printf ("got %d bytes from client.\n", n);
           write(connfd, buf, n);
        }

        printf("connection terminated.\n");
        close(connfd);
    }
}
```