W12 - NETWORKING

OBJECTIVES

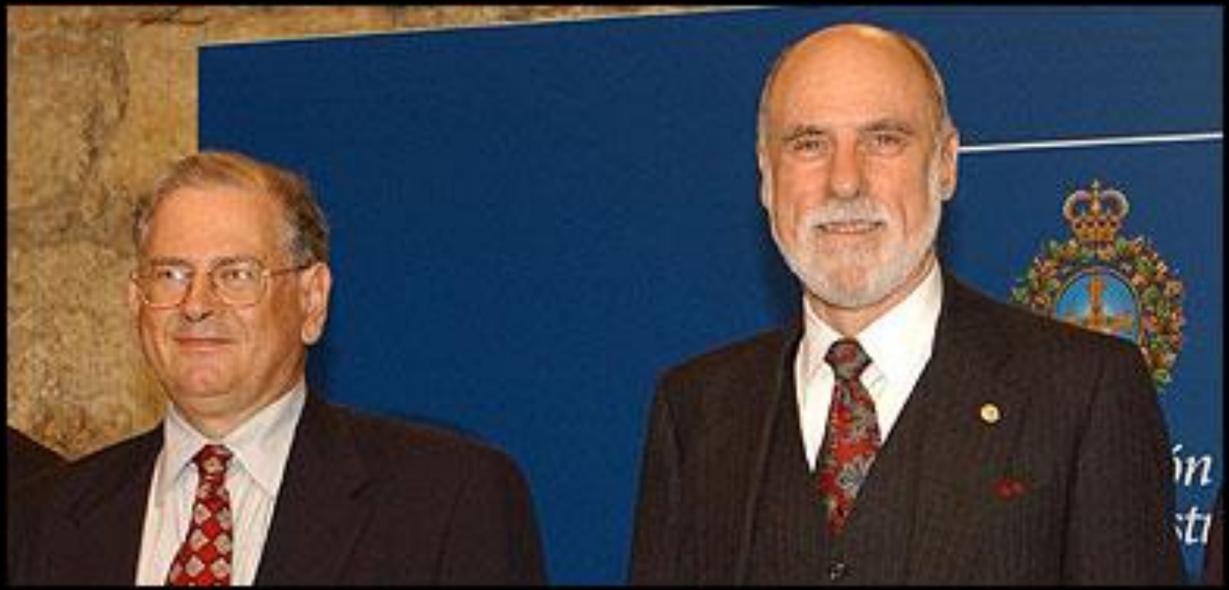
- Get exposure to the basic underpinnings of the Internet

- Learn to use network socket interfaces

WHAT ARE WE GOING TO DISCUSS?

- ❑Basic communication mechanisms
- ❑How did internet come about
- ❑Anatomy of the internet
- ❑Network programming

The 2004 Turing Award

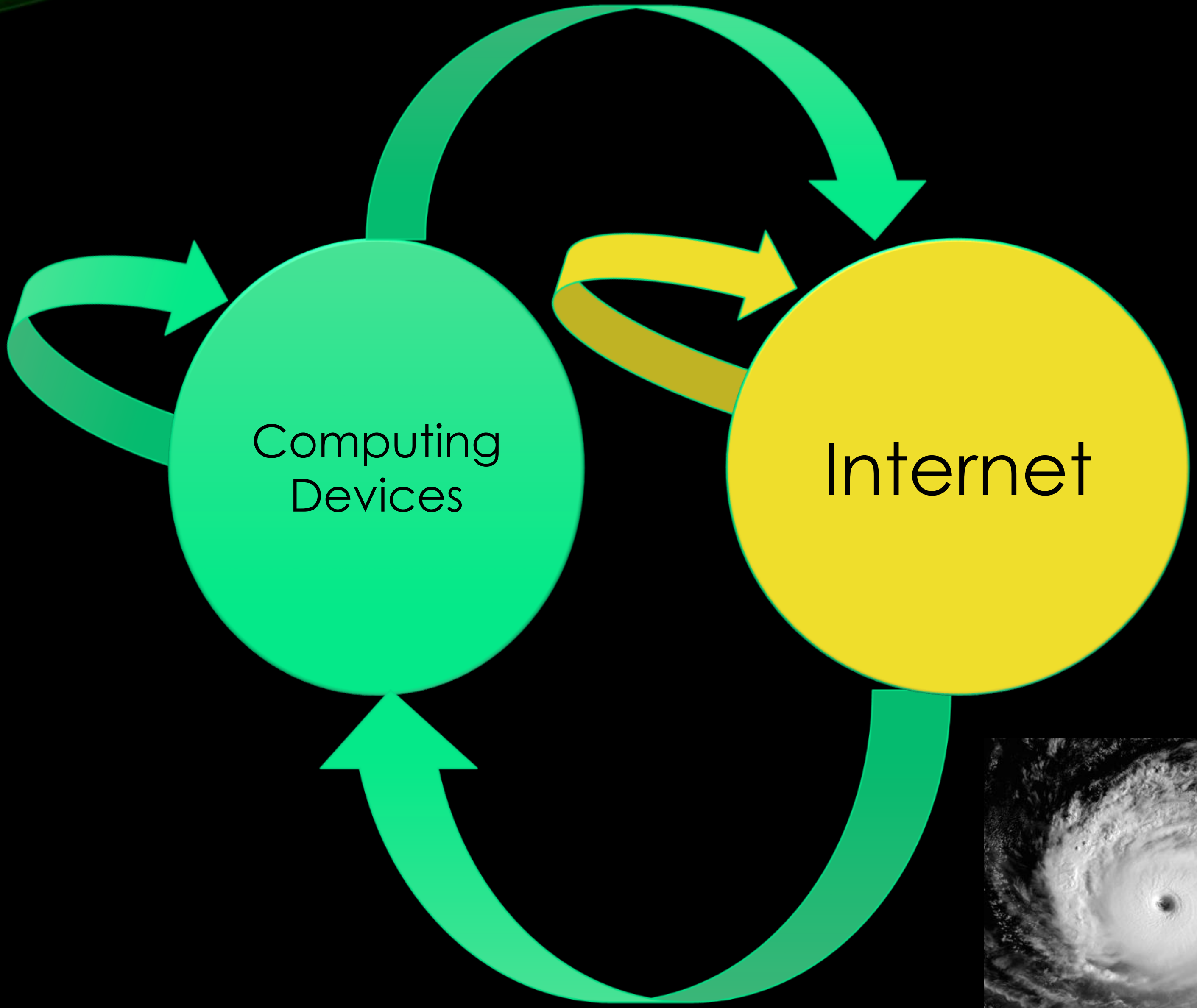


Bob Kahn Vint Cerf

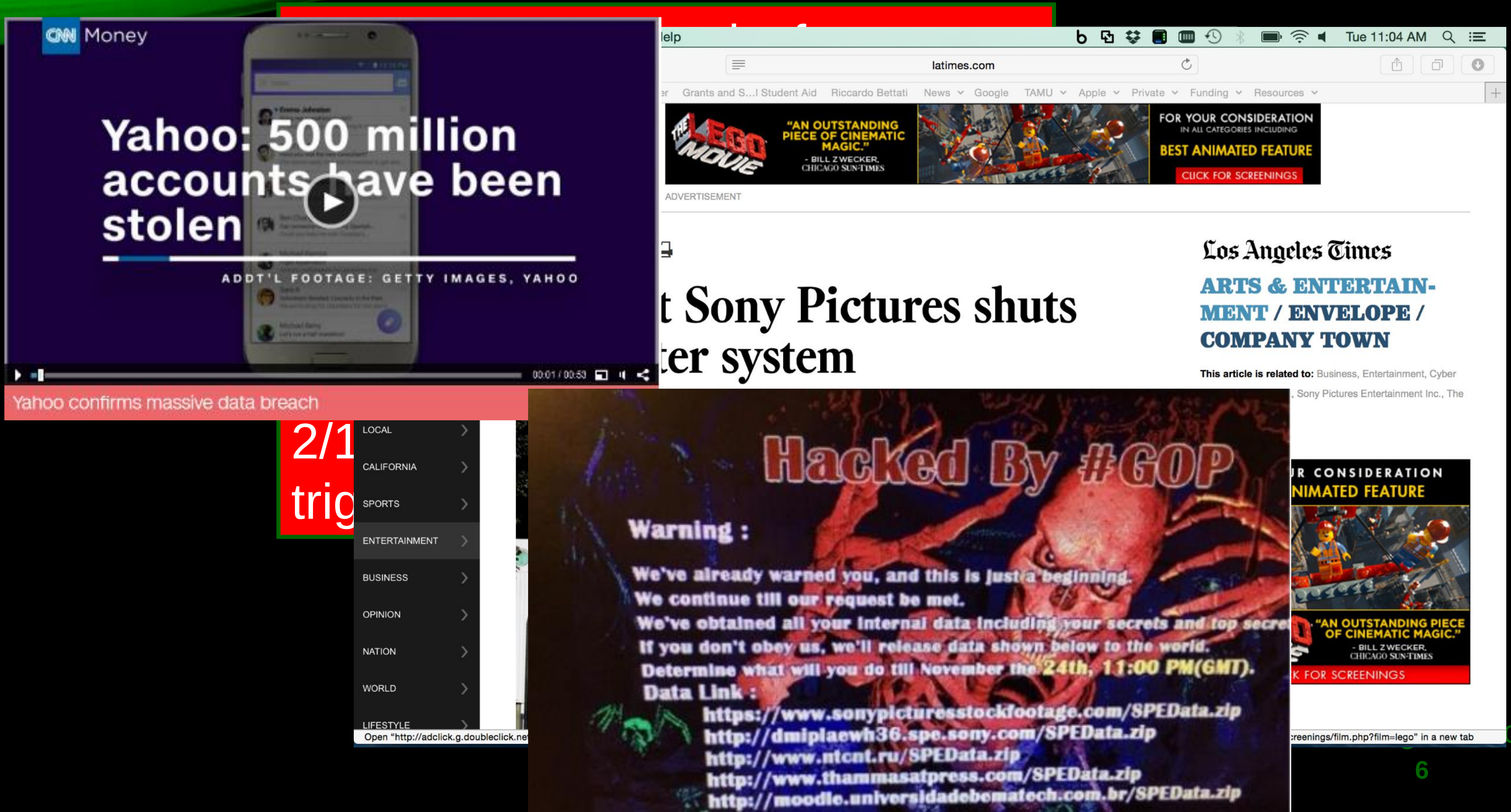
"For pioneering work on internetworking, including the design and implementation of the Internet's basic communications protocols, TCP/IP, and for inspired leadership in networking."

The only Turing Award given to-date to recognize work in computer networking

OUR AMAZING POSITIVE FEEDBACK LOOP



But at the Same Time...



BOTTOM-LINE…..

- Internet has a ubiquitous presence in our lives
- Issues such as Security Lapses present themselves as opportunities for making our ways of communication more robust (e.g. safety, security)
- Let’s now trace back the history from the early days of telephony in the next few slides

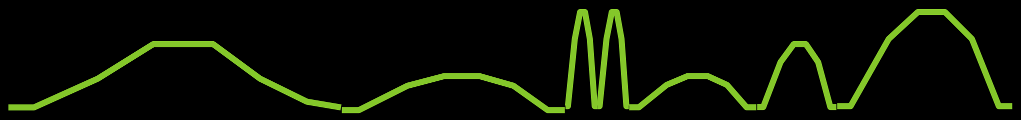
Telephony

Interactive telecommunication between people

Analog voice

- Transmitter/Receiver continuously in contact with electronic circuit
- Electric current varies with acoustic pressure

Analog/Continuous Signal

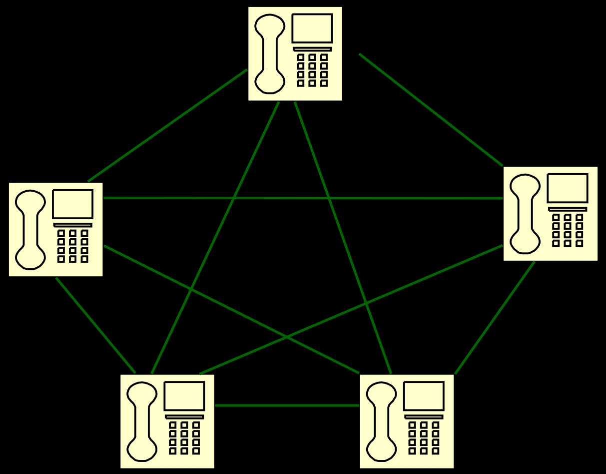


Telephony Milestones

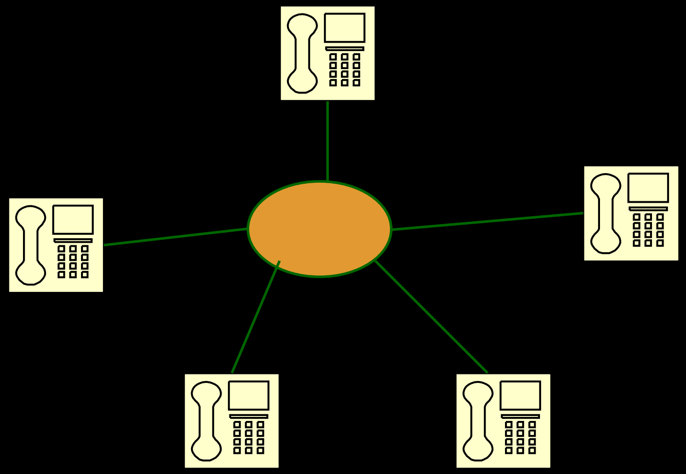
1876: Alexander Bell invented telephone

1878: Public switches installed at New Haven and San Francisco, public switched telephone network is born

- People can talk without being on the same wire!



Without Switch



With Switch

Telephony Milestones

1878: First telephone directory; White House line

1881: Insulated, balanced twisted pair introduced

1885: AT&T formed

1892: First automatic commercial telephone switch

1903: 3 million telephones in U.S.

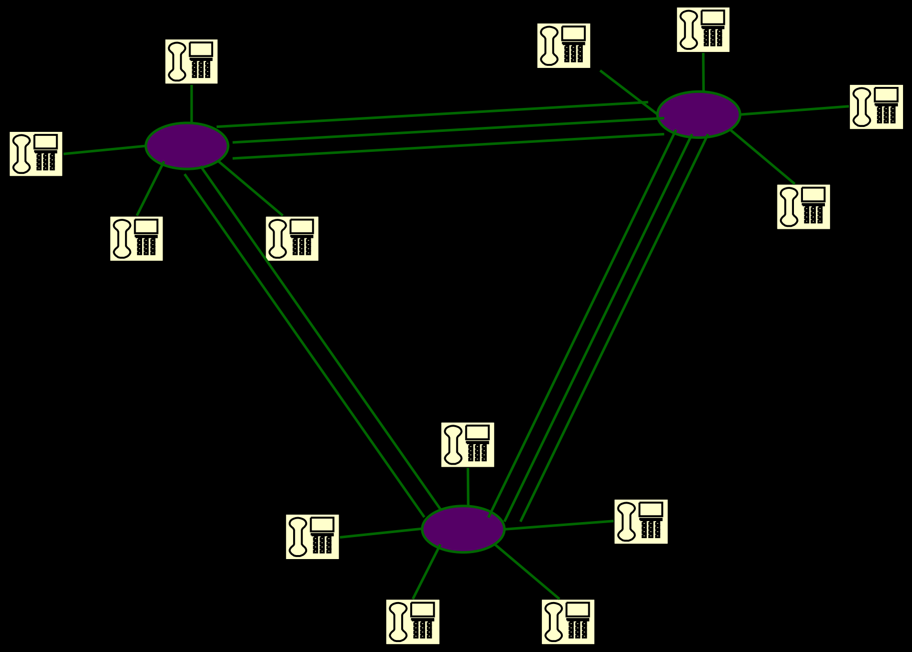
1915: First transcontinental telephone line

1927: First commercial transatlantic commercial service

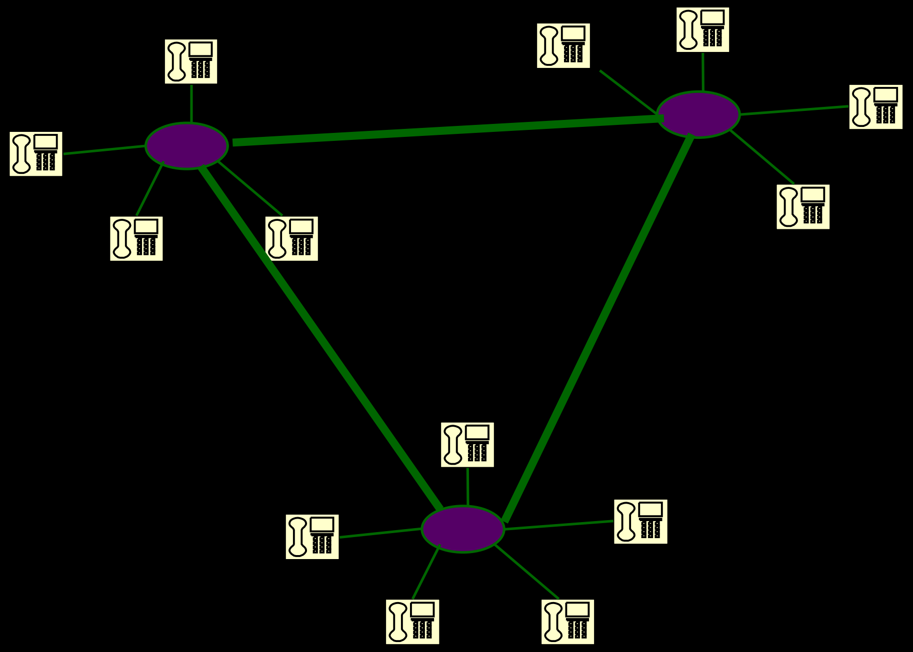
Telephony Milestones

1937: Multiplexing introduced for inter-city calls

- One link carries multiple conversations



Without Multiplexing



With Multiplexing

Data or Computer Networks

Networks designed for computers to computers or devices

- vs. communication between human beings

Digital information

- vs. analog voice

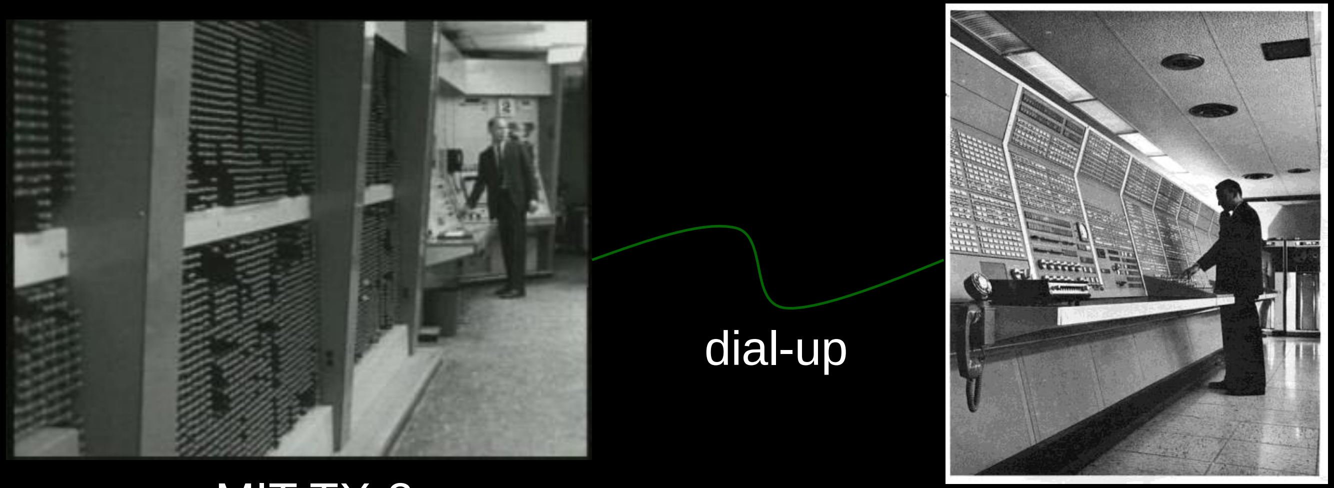
Digital/Discrete Signal

Not a continuous stream of bits, rather, discrete “packets” with lots of silence in between

- Dedicated circuit hugely inefficient

Major Internet Milestones

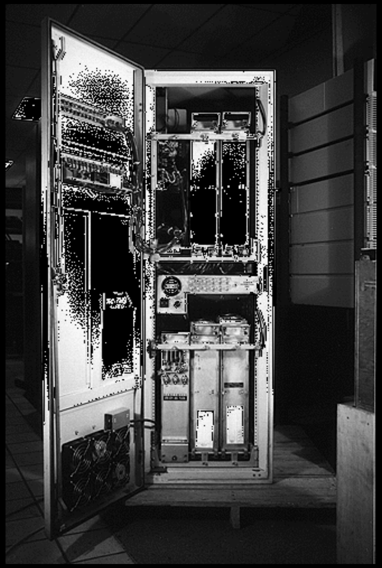
1965 First time two computers talked to each other using packets (Roberts, MIT; Marill, SDC)



MIT TX-2

SDC Q32

Major Internet Milestones



1969 The first ARPANET message transmitted between UCLA (Kleinrock) and SRI (Engelbart)

- We sent an “L”, did you get the “L”? Yep!
- We sent an “O”, did you get the “O”? Yep!
- We sent a “G”, did you get the “G”?

Crash!

Major Internet Milestones

- 1970 First packet radio network ALOHANET (Abramson, U Hawaii)
- 1973 Ethernet invented (Metcalfe, Xerox PARC)
- 1974 “A protocol for Packet Network Interconnection” published by Cerf and Kahn
  - First internetworking protocol TCP
  - This paper was cited for their Turing Award
- 1977 First TCP operation over ARPANET, Packet Radio Net, and SATNET
- 1985 NSF commissions NSFNET backbone
- 1991 NSF opens Internet to commercial use

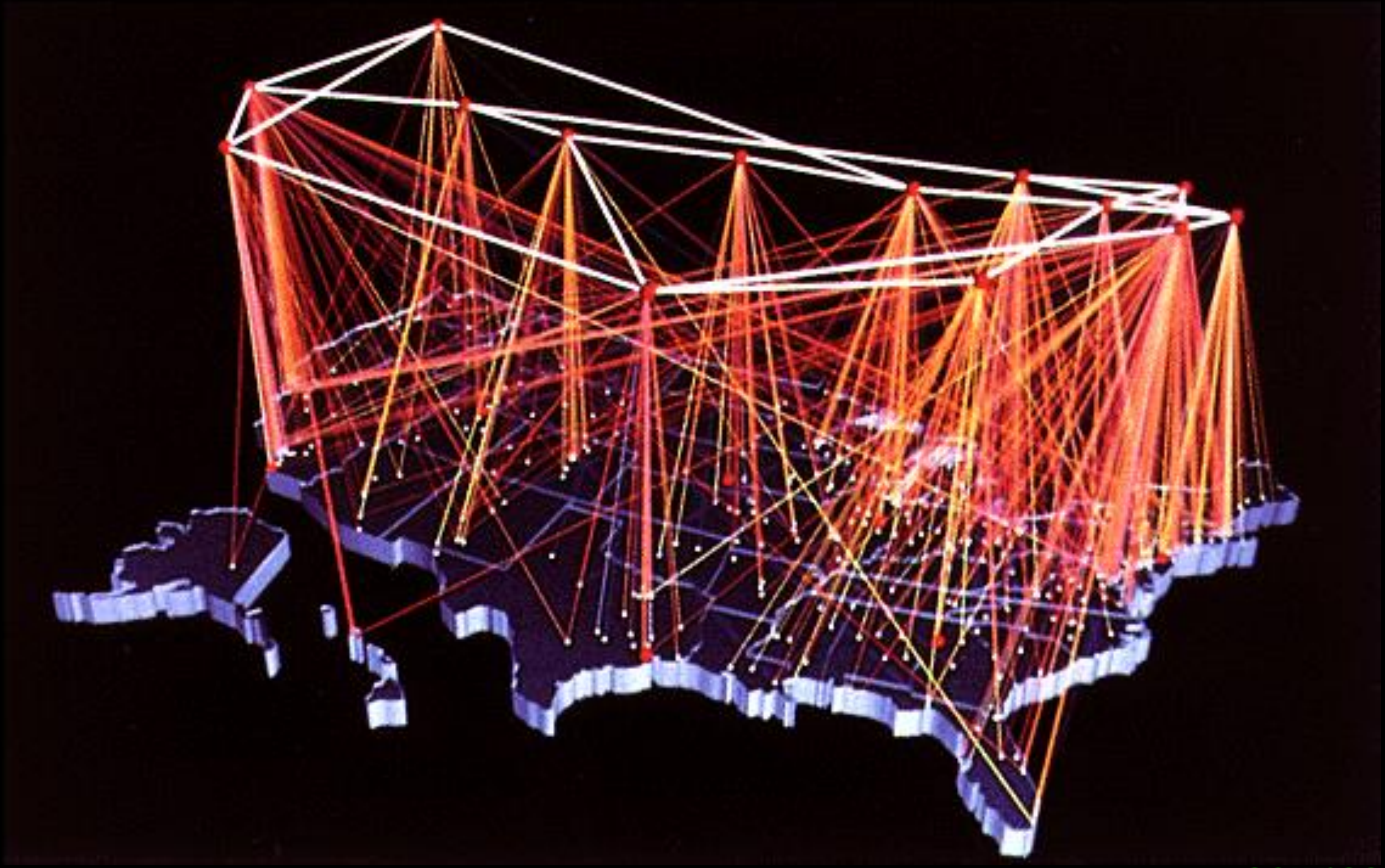
The Internet Circa 1986

In 1986, the Internet consisted of one backbone (NSFNET) that connected 13 sites via 45 Mbps T3 links

- **Merit (Univ of Mich)**
- **NCSA (Illinois)**
- **NYSERNET Cornell Theory Center**
- **Pittsburgh Supercomputing Center**
- **San Diego Supercomputing Center**
- **John von Neumann Center (Princeton)**
- **BARRNet (Palo Alto)**
- **MidNet (Lincoln, NE)**
- **WestNet (Salt Lake City)**
- **NorthwestNet (Seattle)**
- **SESQUINET (Rice)**
- **SURANET (Georgia Tech)**
- **NEARNET (New England)**

Connecting to the Internet involved connecting one of your routers to a router at a backbone site, or to a regional network that was already connected to the backbone

NSFNET Internet Backbone

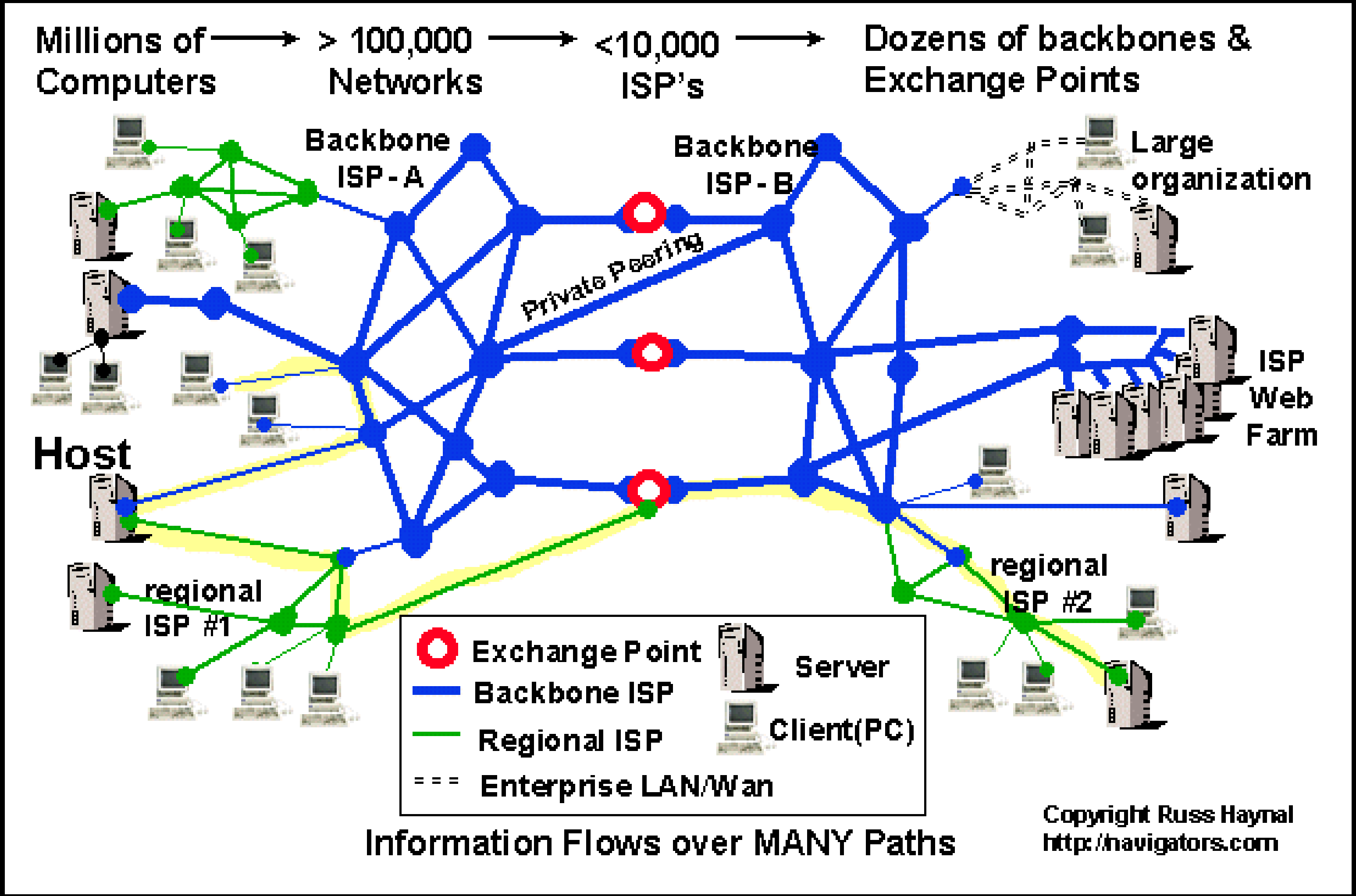


source: www.eef.org

After NSFNET

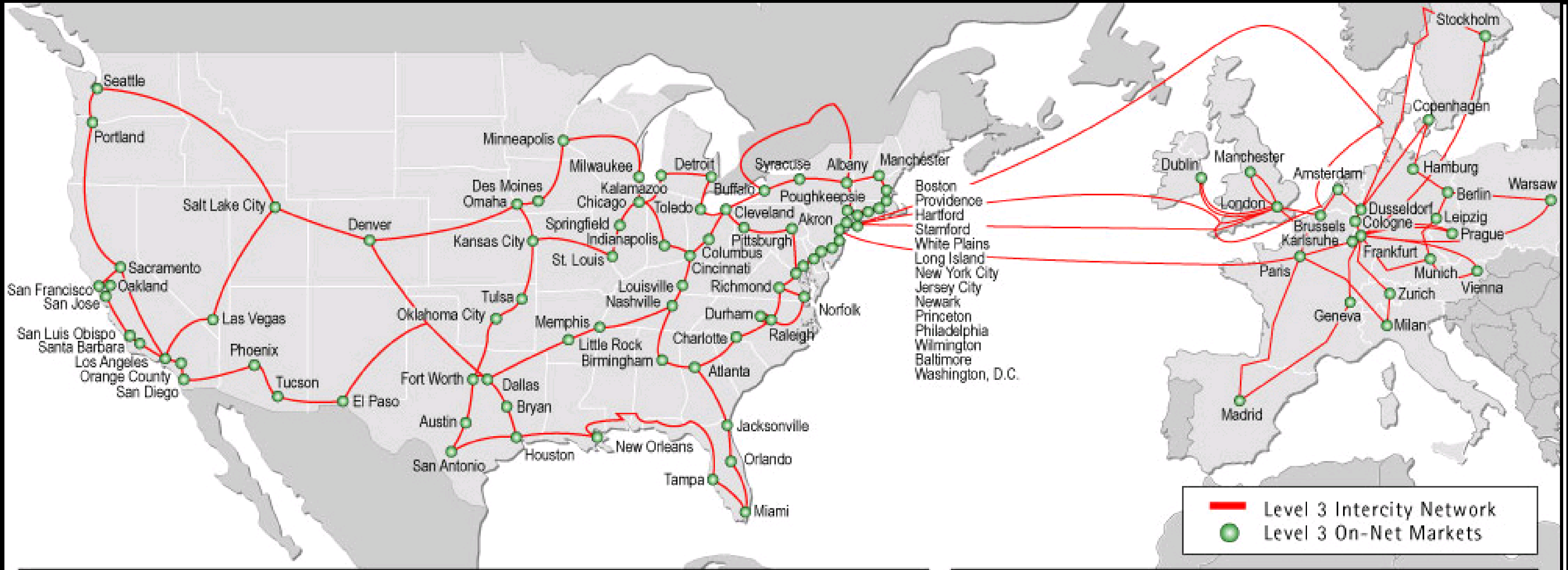
- Early 90s
  - Commercial enterprises began building their own high-speed backbones
  - Backbone would connect to NSFNET, sell access to companies, ISPs, and individuals
- 1995
  - NSFNET decommissioned
  - NSF fostered the creation of network access points (NAPs) to interconnect the commercial backbones

Current Internet Architecture - Conceptual



Level 3 Backbone

Level(3) COMMUNICATIONS



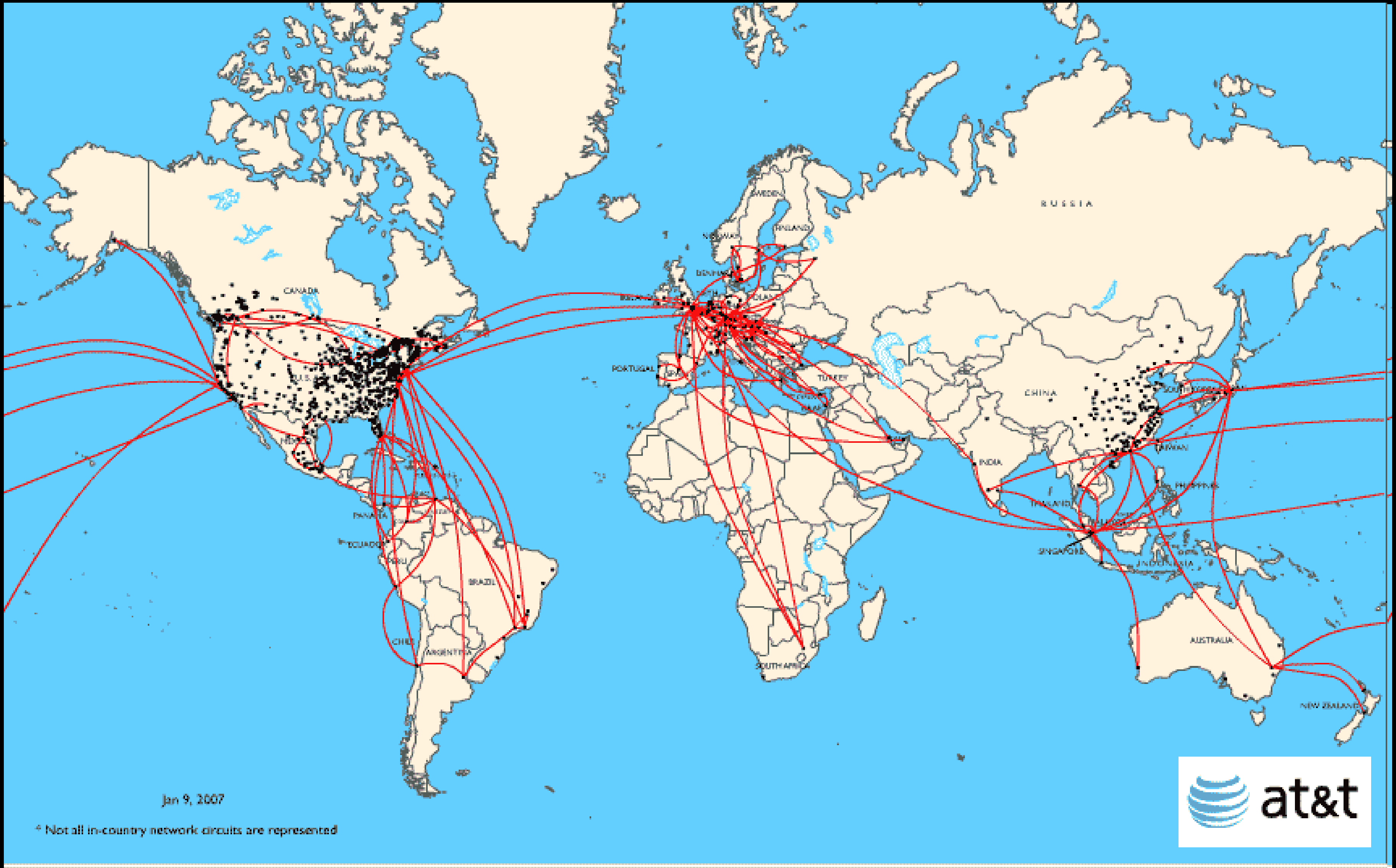
Metro Networks

Amsterdam	Chicago	Frankfurt	Miami	Paris	San Jose
Atlanta	Cincinnati	Hamburg	Munich	Philadelphia	Seattle
Baltimore	Dallas	Houston	Newark	Phoenix	St. Louis
Berlin	Denver	Jersey City	New York City	Portland	Stamford
Boston	Detroit	London	Orange County	San Diego	Tampa
Brussels	Dusseldorf	Los Angeles	Orlando	San Francisco	Washington, D.C.

Network Statistics

- 23,000 intercity route miles
- 2,200 metropolitan route miles
- 947,000 miles of installed metro fiber
- 320 Gbps of transatlantic capacity
- 850 international points of presence
- 100 on-net markets
- 300+ voice markets

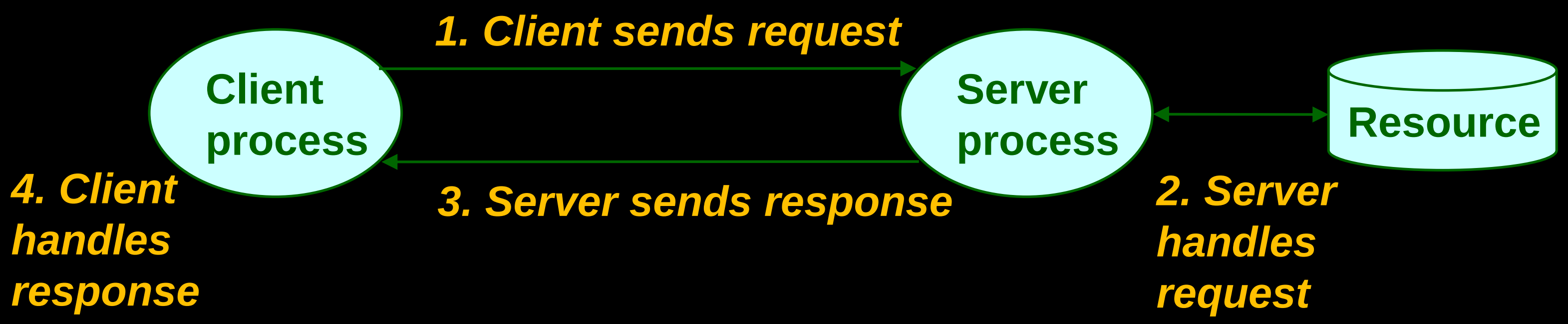
AT&T Backbone



A Client-Server Transaction

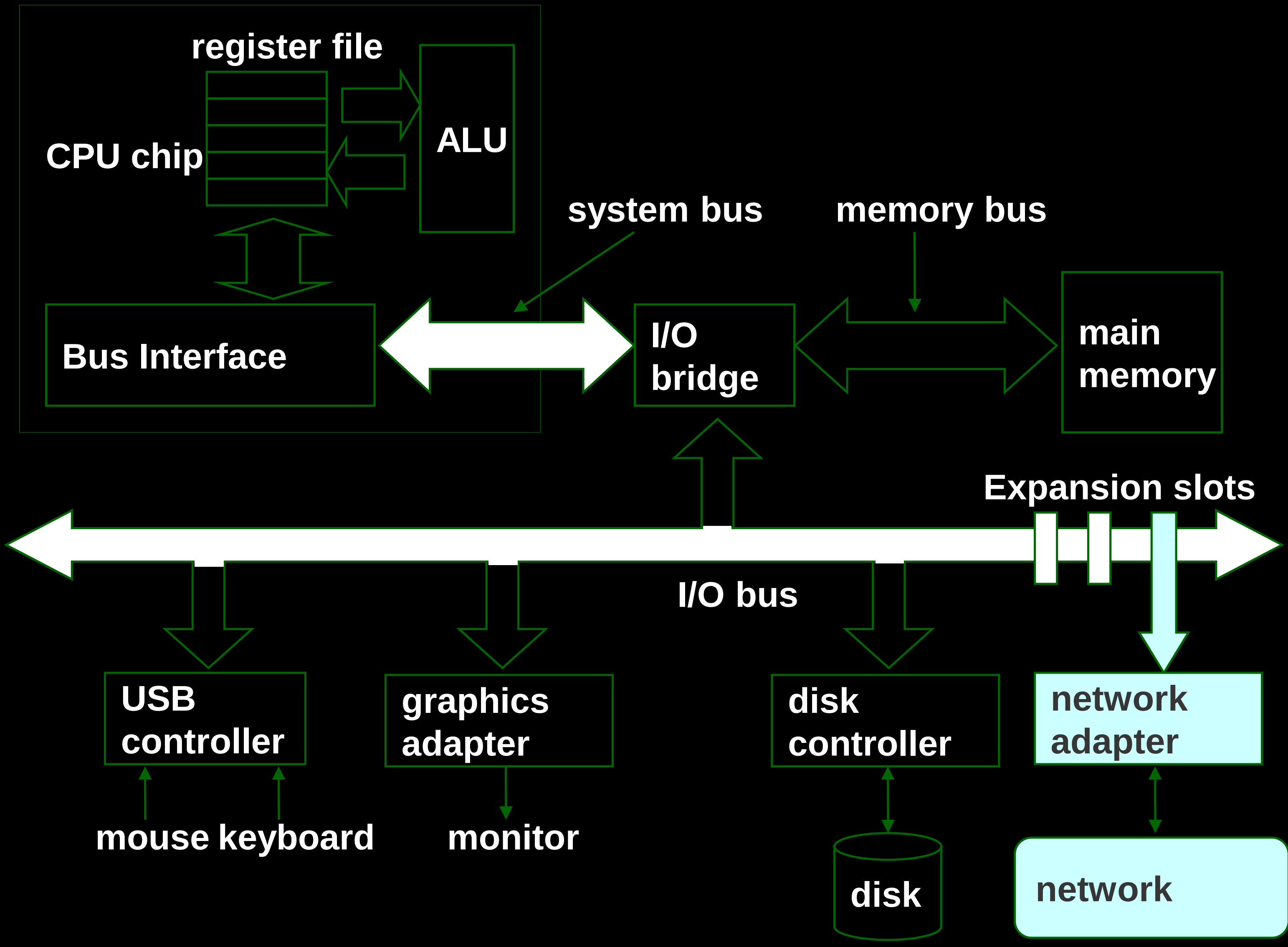
Most network applications are based on the client-server model:

- A server process and one or more client processes
- Server manages some resource
- Server provides service by manipulating resource for clients



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

Network Hardware



Computer Networks

A network is a hierarchical system of boxes and wires organized by geographical proximity

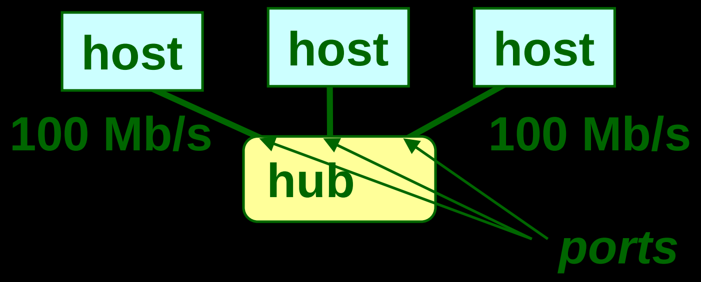
- LAN (local area network) spans a building or campus
  - Ethernet is most prominent example
- WAN (wide-area network) spans very long distance
  - A high-speed point-to-point link
  - Leased line or SONET/SDH circuit, or MPLS/ATM circuit

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")

Lowest Level: Ethernet Segment

Ethernet segment consists of a collection of hosts connected by wires (twisted pairs) to a hub



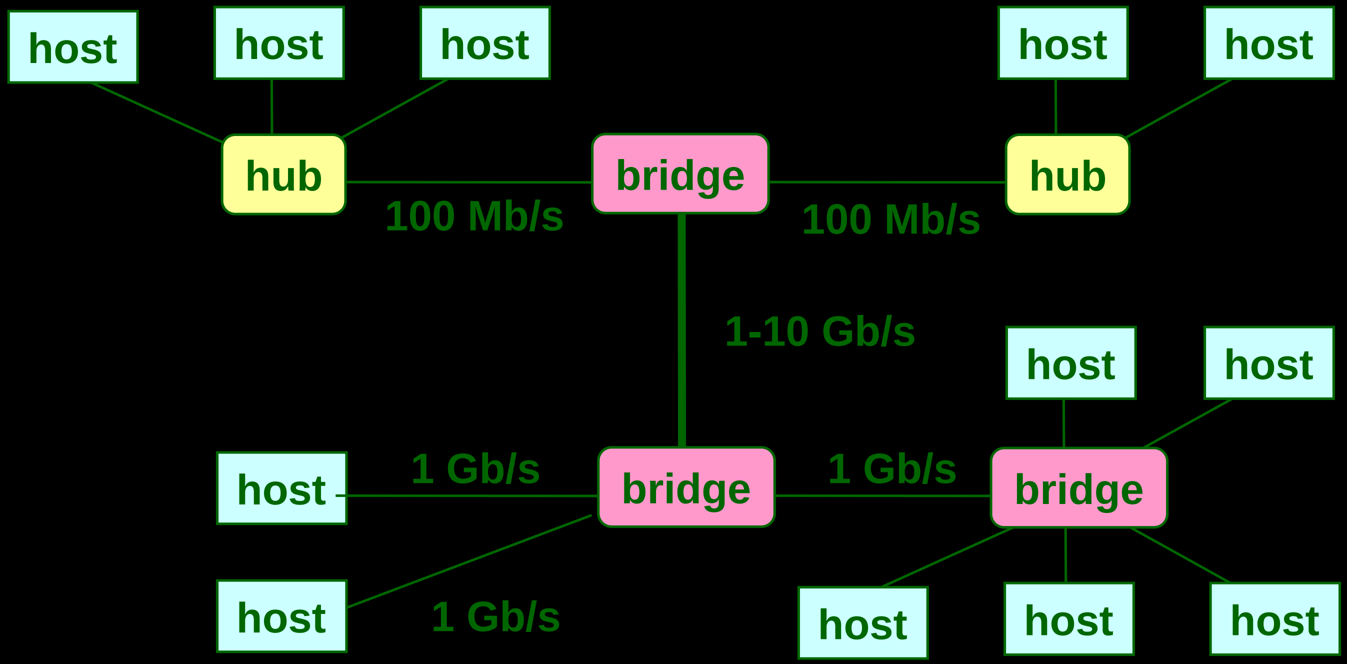
Operation

- Each Ethernet adapter has a unique 48-bit address
- 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
- Note: Hubs are largely obsolete
  - Bridges (switches, routers) became cheap enough to replace them (don't broadcast all traffic)

Next Level: Bridged Ethernet Segment

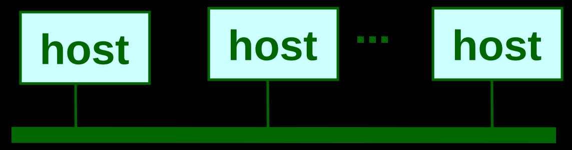
Spans room, building, or campus

Bridges cleverly learn which hosts are reachable from which ports and then selectively copy frames from port to port



Conceptual View of LANs

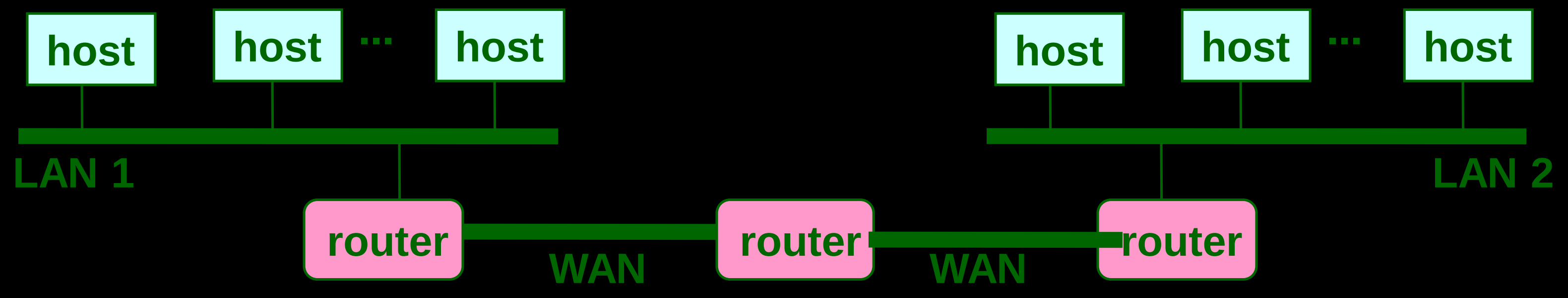
For simplicity, hubs, bridges, and wires are often shown as a collection of hosts attached to a single wire:



Next Level: internets

Multiple incompatible LANs can be physically connected by specialized computers called routers

The connected networks are called an internet



LAN 1 and LAN 2 might be completely different, totally incompatible LANs (e.g., Ethernet and WiFi, 802.11*, T1-links, DSL, …)

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 smoothens out the differences between the different networks

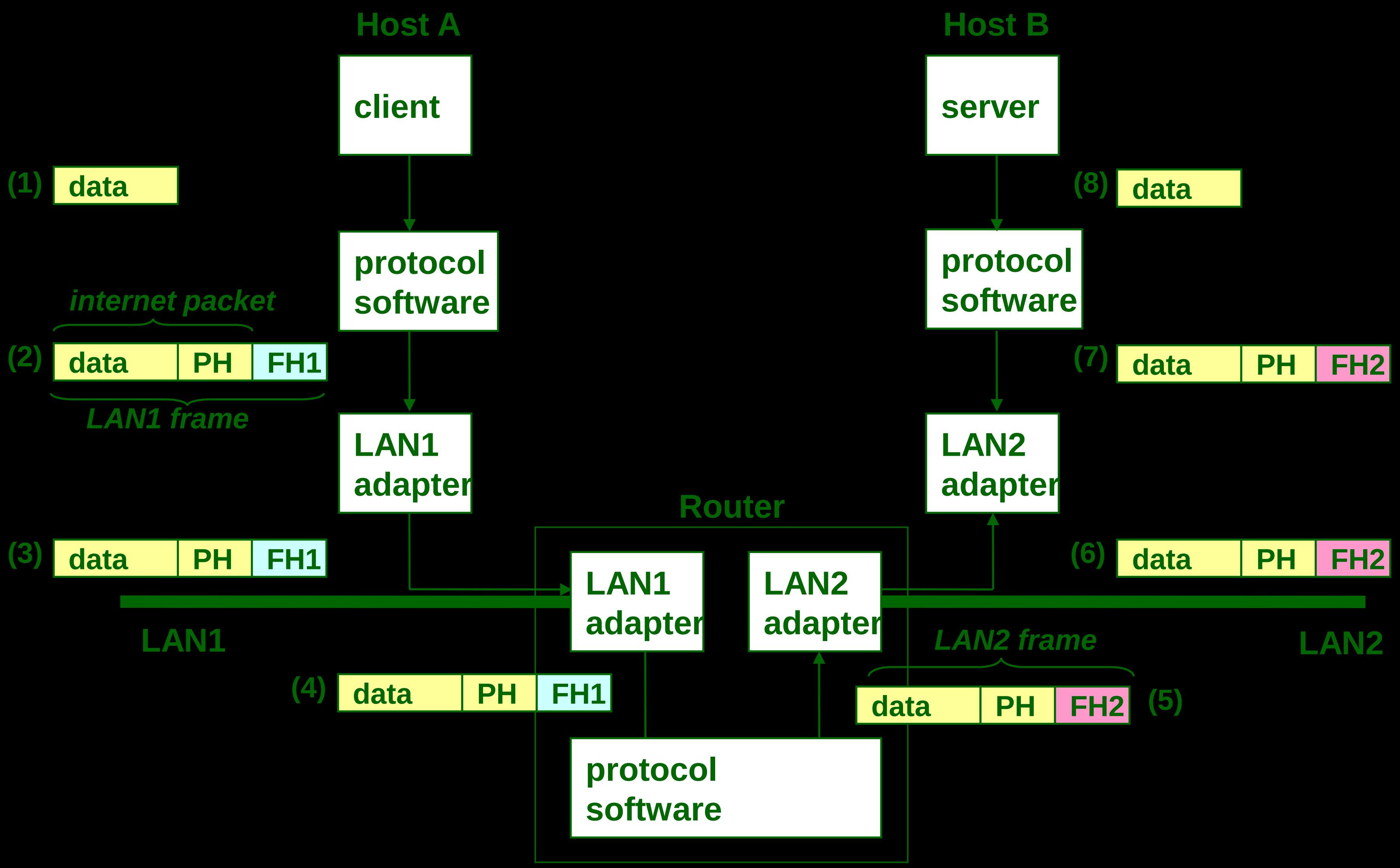
Implements an internet protocol (i.e., set of rules) that governs how hosts and routers should cooperate when they transfer data from network to network

- TCP/IP is the protocol for the global IP Internet

What Does an Internet Protocol Do?

1. 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
2. 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 Data Over an internet



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?

We'll leave the discussion of these question to computer networking classes

→ Dr. Loguinov CSCE-463 class

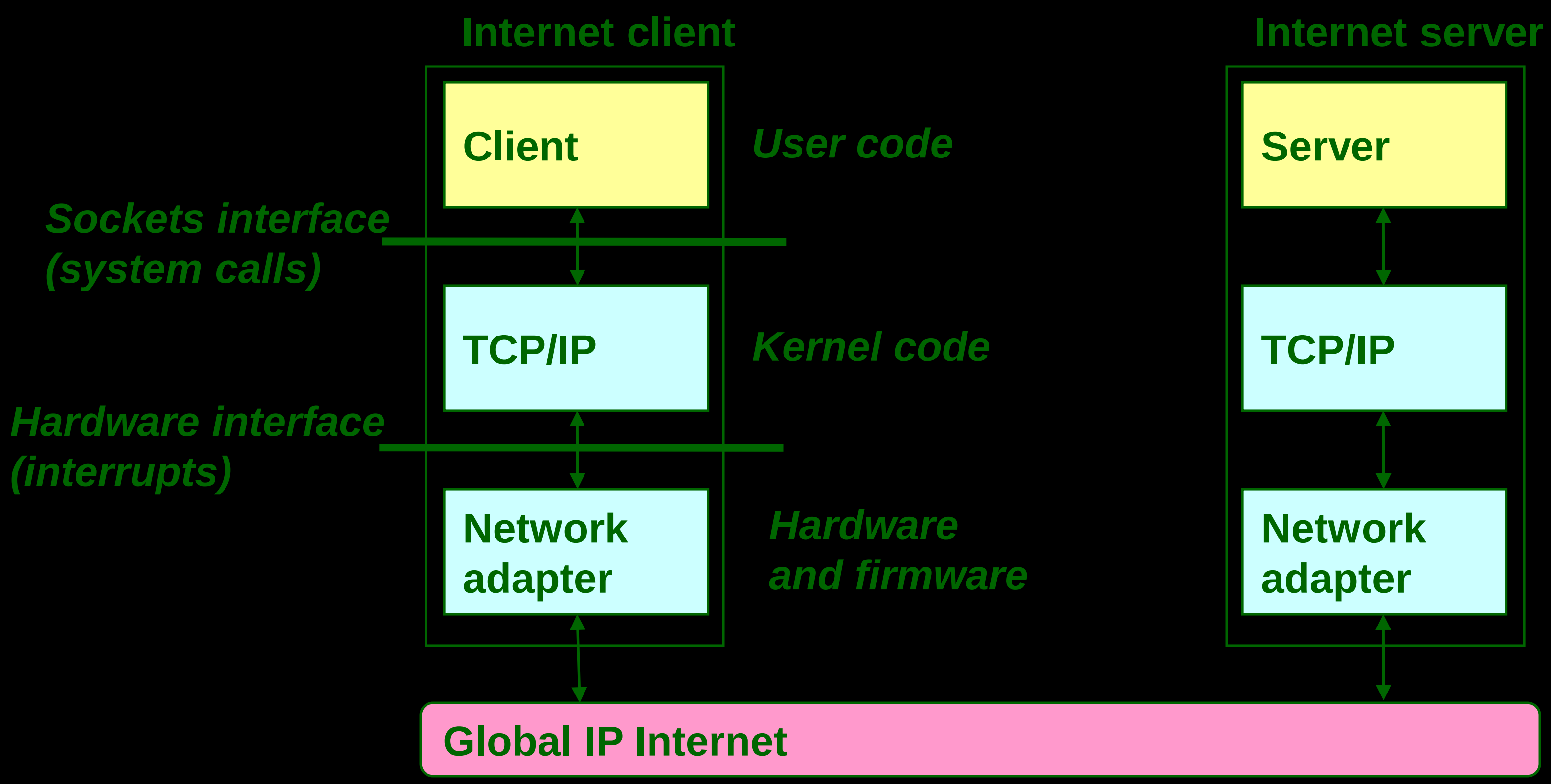
Global IP Internet

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

Organization of an Internet Application



A Programmer's View of the Internet

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

- e.g. 128.194.255.88 (4 * 8 bits)

A set of identifiers called Internet domain names are mapped to the set of IP addresses for convenience (Domain Name Server aka DNS)

- linux2.cs.tamu.edu is mapped to 128.194.138.88
- A process on one Internet host can communicate with a process on another Internet host over a connection

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 {
    unsigned int s_addr; /* network byte order (big-endian) */
};
```

Handy network byte-order conversion functions:

`htonl`: **convert** `long int` **from host to network byte order**
`htons`: **convert** `short int` **from host to network byte order**
`ntohl`: **convert** `long int` **from network to host byte order**
`ntohs`: **convert** `short int` **from network to host byte order**

Dotted Decimal Notation

By convention, each byte in a 32-bit IP address is represented by its decimal value and separated by a period

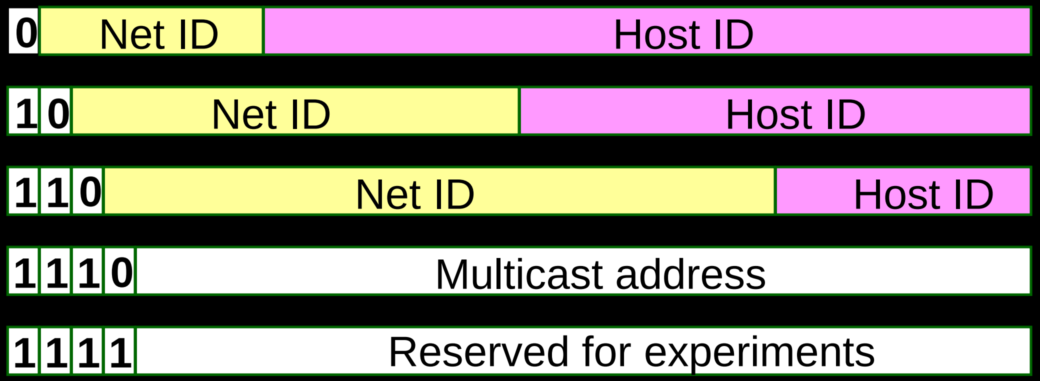
- IP address `0x8002C2F2 = 128.2.194.242`

Functions for converting between binary IP addresses and dotted decimal strings:

- `inet_pton`: converts a dotted decimal string to an IP address in network byte order
- `inet_ntop`: converts an IP address in network by order to its corresponding dotted decimal string
- “`n`” denotes network representation, “`p`” denotes presentation representation

IP Address Structure

IP (V4) Address space divided into classes:



Special Addresses for routers and gateways (all 0/1’s)

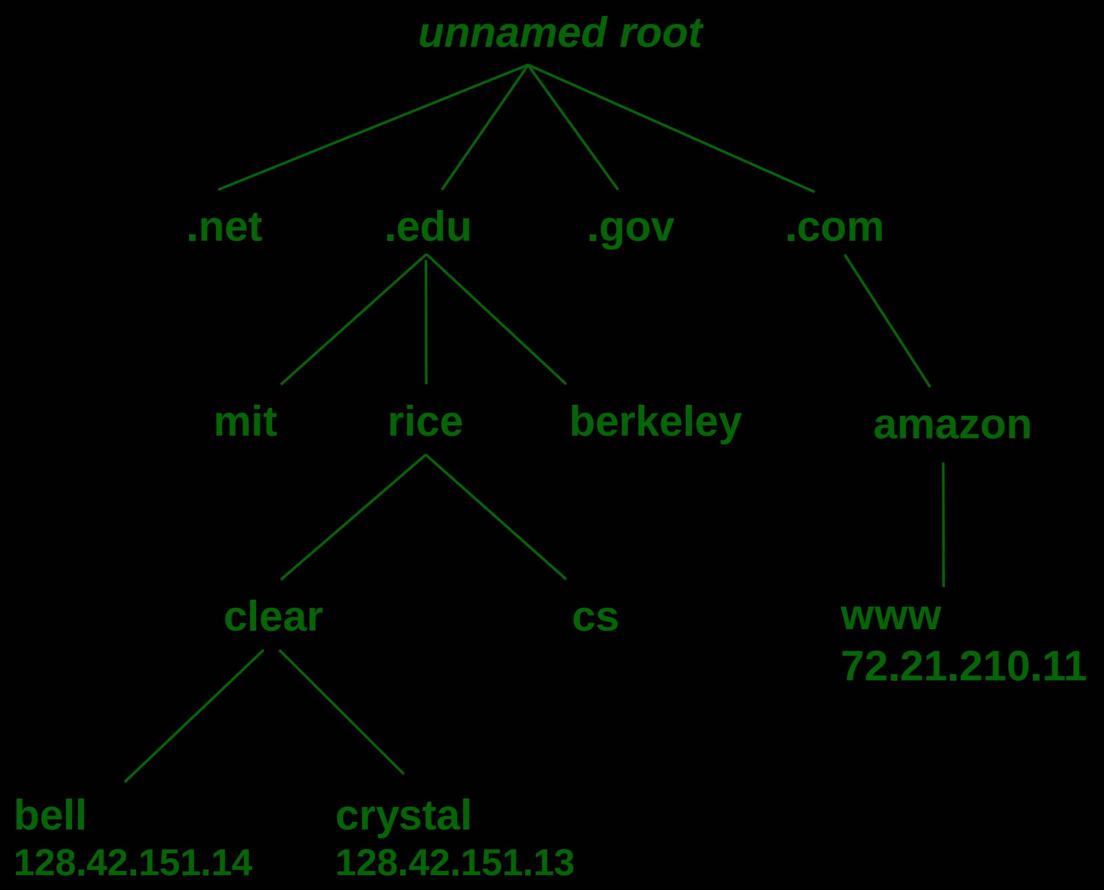
Loop-back address: 127.0.0.1

Unrouted (private) IP addresses:

- 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16

Dynamic IP addresses (DHCP)

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 huge worldwide distributed database called DNS

- Conceptually, programmers can view the DNS database as a collection of millions of addrinfo structures:

```
struct addrinfo {
    int             ai_flags;     /* flags for getaddrinfo */
    int             ai_family;    /* address type (AF_INET or AF_INET6) */
    int             ai_socktype;  /* the socket type */
    int             ai_protocol;  /* the type of protocol */
    size_t          ai_addrlen;   /* length of ai_addr */
    struct sockaddr   *ai_addr;       /* pointer to a sockaddr struct */
    char            *ai_canonname;/* the canonical name */
    struct addrinfo *ai_next; /* pointer to the next addrinfo struct */
};
```

Functions for retrieving host entries from DNS:

- getaddrinfo: query DNS using domain name or IP
- getnameinfo: query DNS using sockaddr struct

Properties of DNS Host Entries

Each host entry is an equivalence class of domain names and IP addresses

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

Different kinds of mappings are possible:

- Simple case: 1 domain name maps to one IP address
- Multiple domain names mapped to the same IP address
- Multiple domain names mapped to multiple IP addresses
- Some valid domain names don’t map to any IP address

Querying DNS

Domain Information Groper (dig) provides a scriptable command line interface to DNS

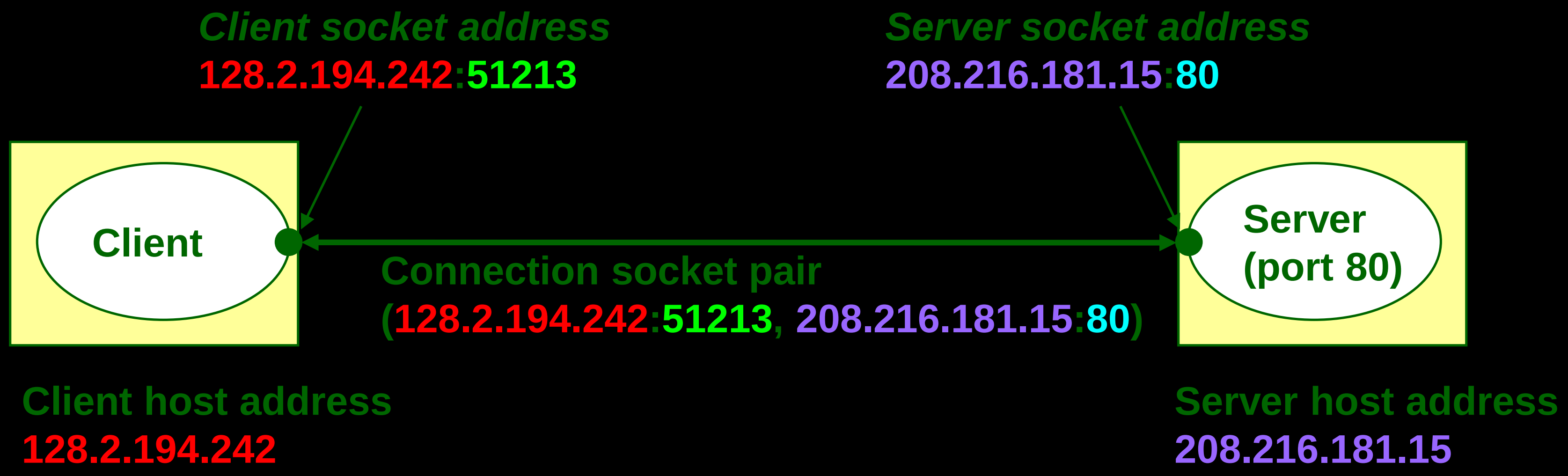
- Lots of web interfaces (google "domain information groper")

```
unix> dig +short linux2.cse.tamu.edu
128.194.138.88
unix> dig +short -x 128.194.138.85
chevron.cs.tamu.edu.
unix> dig +short google.com
74.125.227.174
74.125.227.169
74.125.227.160
```

Internet Connections

- Clients and servers communicate by sending streams of bytes over connections:
  - Point-to-point, full-duplex (2-way communication), and reliable
- A socket is an endpoint of a connection
  - Socket address is an IP address, port pair
- A port is a 16-bit integer that identifies a process:
  - Ephemeral port: Assigned automatically on client 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)
- A connection is uniquely identified by the socket addresses of its endpoints (socket pair)
  - (cliaddr:cliport, servaddr:servport)

Putting it all Together: Anatomy of an Internet Connection



Clients

- Examples of client programs
  - Web browsers, ftp, telnet, ssh
- How does a client find the server?
  - The IP address in the server socket address identifies the host (more precisely, an adapter on the host)
  - The (well-known) port in the server socket address identifies the service, and thus implicitly identifies the server process that performs that service

Servers

- Servers are long-running processes (daemons)
  - Created at boot-time (typically) by the init process (process 1)
  - Run continuously until the machine is turned off
- Each server waits for requests to arrive on a well-known port associated with a particular service
  - Port 23: telnet server
  - Port 25: mail server
  - Port 80: HTTP server
- A machine that runs a server process is also often referred to as a “server”

Server Examples

Web server (port 80)

- Resource: files/compute cycles (CGI programs)
- Service: retrieves files and runs CGI programs on behalf of the client

FTP server (20, 21)

- Resource: files
- Service: stores and retrieve files

Telnet server (23)

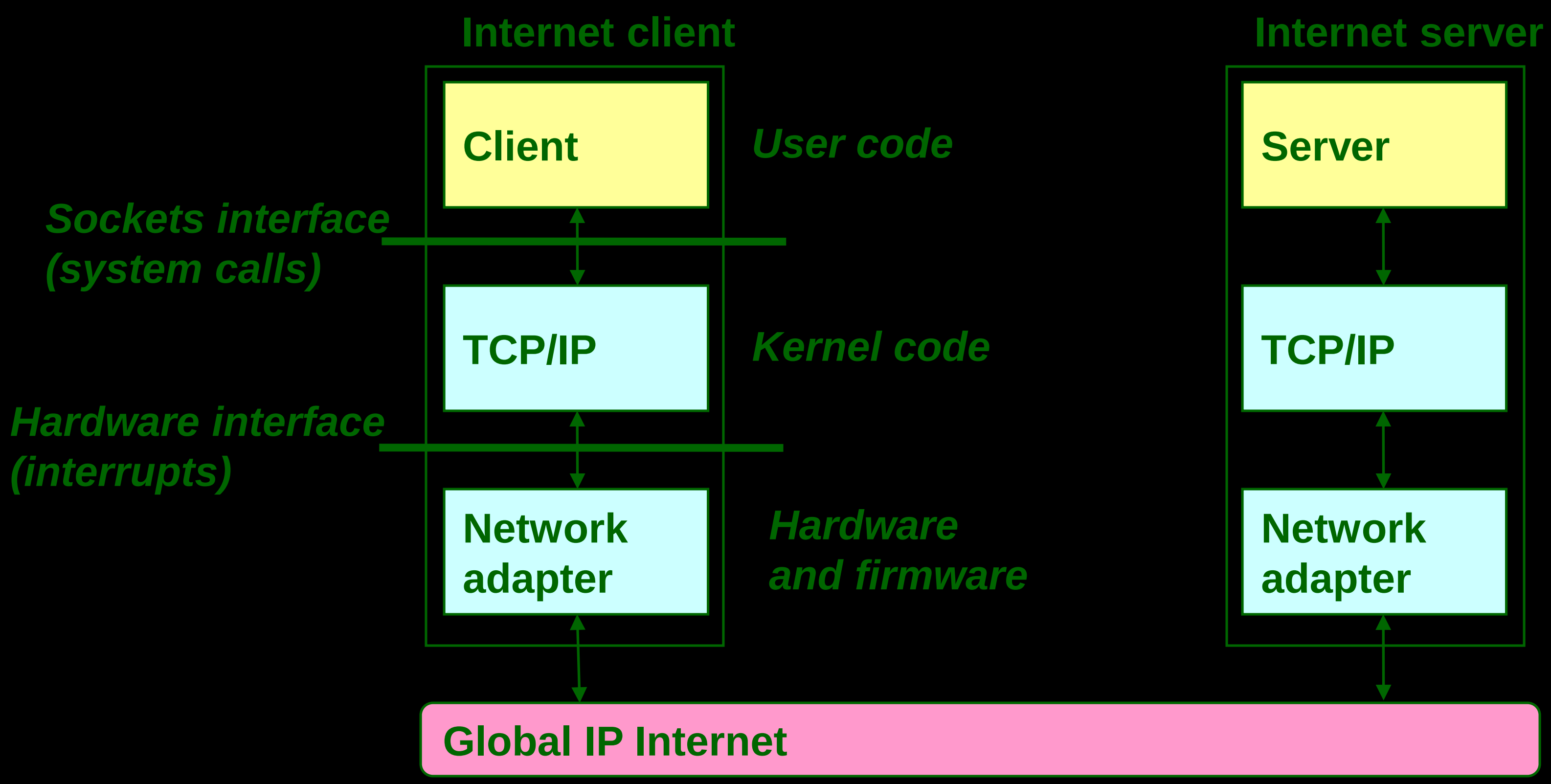
- Resource: terminal
- Service: proxies a terminal on the server machine

Mail server (25)

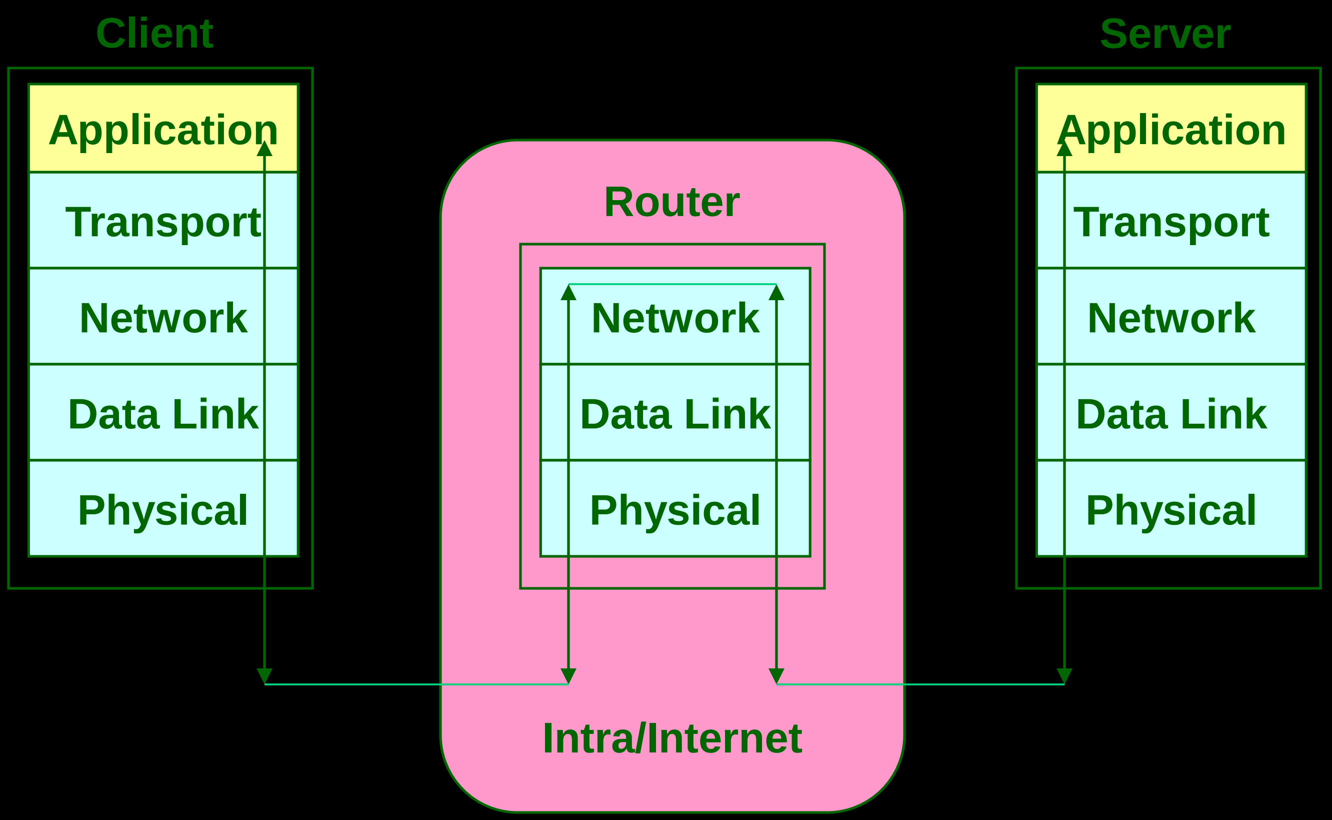
- Resource: email “spool” file
- Service: stores mail messages in spool file

See `/etc/services` for a comprehensive list of the services available on a UNIX machine

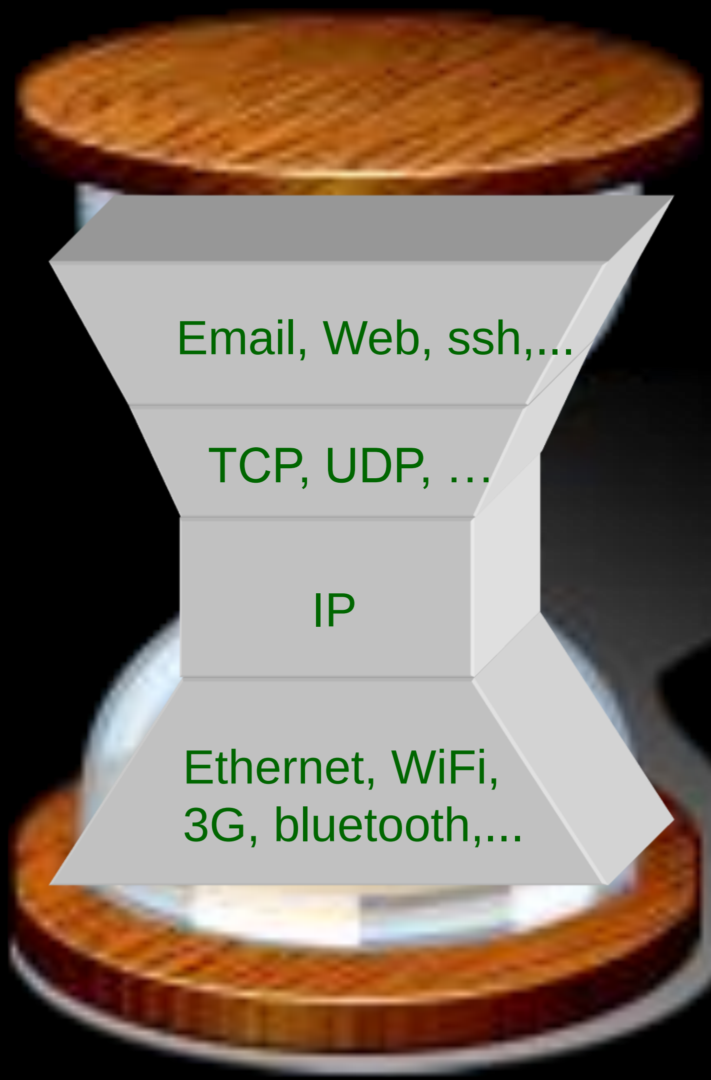
Organization of an Internet Application

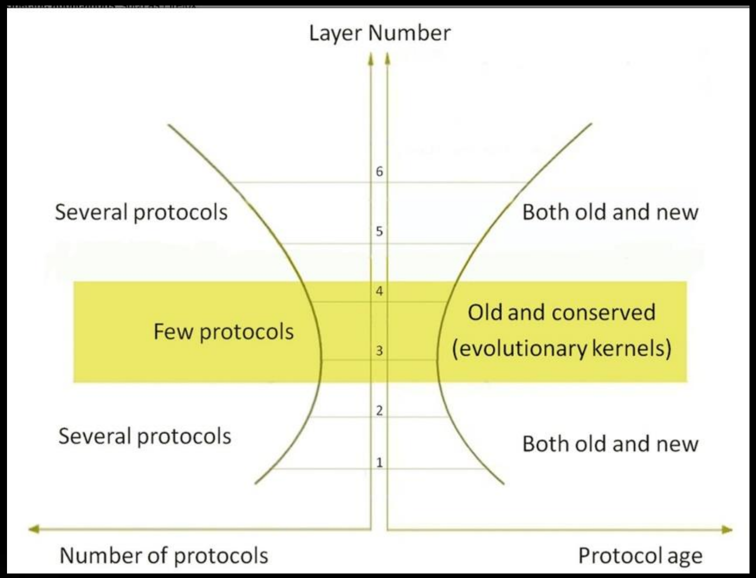


OSI Model (Layers)



Internet Hourglass Architecture





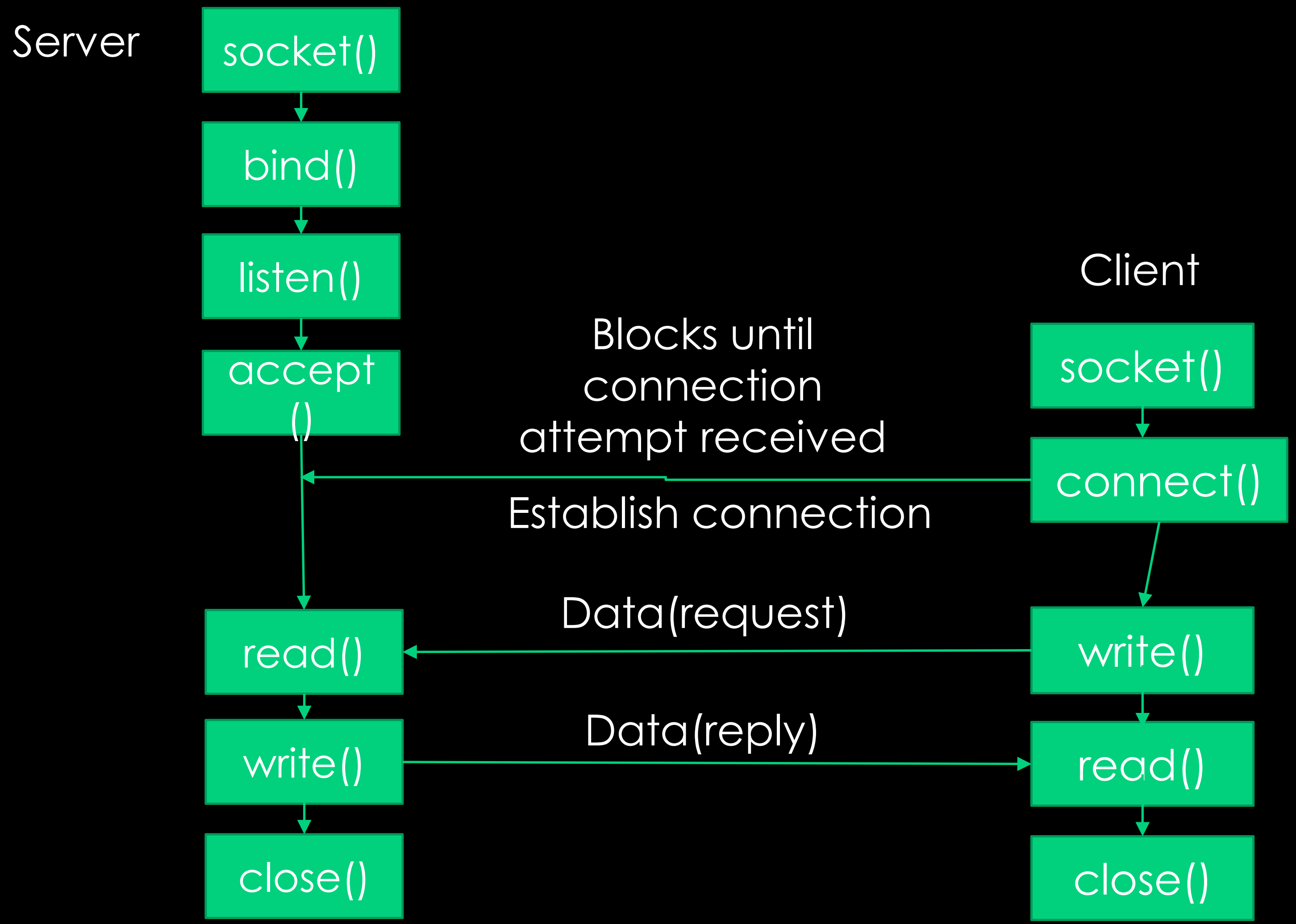
Source: GATECH Internet Hourglass Architecture

From top to bottom, the Internet architecture consists of six layers:

1. **Specific applications**, such as Firefox;
2. **Application protocols**, such as Hypertext Transfer Protocol (HTTP);
3. **Transport protocols**, such as Transmission Control Protocol (TCP);
4. **Network protocols**, such as Internet Protocol (IP);
5. **Data-link protocols**, such as Ethernet; and
6. **Physical layer protocols**, such as DSL.

Layers near the top and bottom contain many items, called protocols.
The central transport layer contains two protocols
and the network layer contains only one, creating an hourglass architecture.

A Server-Client Interaction in TCP – POSIX Functions



A Closer Look into POSIX Functions

- `getaddrinfo()`
- `socket()`
- `bind()`
- `listen()`
- `accept()`
- `connect()`
- `write()`, `send()`, `sendto()`
- `read()`, `recv()`, `recvfrom()`
- `close()`

Data Structures

```
/* structure for looking up IP address */
struct addrinfo{
    int      ai_flags;
    int      ai_family; // AF_INET=IPv4,AF_INET6= IPv6
    int      ai_socktype;        // TCP or UDP
    int      ai_protocol;
    socklen_t  ai_addrlen;      // length of ai_addr
    struct sockaddr* ai_addr; // contains IP+PORT
    char*    ai_canonname;       // canonical name
    struct addrinfo* ai_next; // next pointer of result
link list
};
/* data structure for IP details (+PORT) */
struct sockaddr_in{
    short        sin_family;  // IPv4 or IPv6
    unsigned short sin_port; // port number
    struct in_addr sin_addr; // 32 bit IP
    char sin_zero [8];
};
/* just a wrapper for the numeric IP */
struct in_addr{
    unsigned long s_addr;
};
```

getaddrinfo() – Looking Up IP address from name

- **The first step to locate a server by the client**
- **Converts easy-to-remember DNS names (e.g., linux.cs.tamu.edu) into machine-usable IP address**
- **Queries DNS servers (a collection of mappings)**

```
int getaddrinfo(char* name, char* port, struct addrinfo* hints, struct addrinfo** result);
```

`name` = name of the host

`port` = the port where the service (e.g., http, your data server in MP6) is running

`hints` = provides some initial hint (IPv4/IPv6, TCP/UDP etc.)

result = linked list of looked up addresses

- **Example:**

```
getaddrinfo("www.example.com", "3490", &hints, &res);
```

getaddrinfo() - Detailed

```c
int status;
struct addrinfo hints;
struct addrinfo *servinfo;  // will point to the results

//preparing hints data structure
memset(&hints, 0, sizeof hints); // make sure the struct is empty
hints.ai_family = AF_UNSPEC;     // don't care IPv4 or IPv6
hints.ai_socktype = SOCK_STREAM; // TCP stream sockets

// look up the IP address from the name: "www.example.com"
status = getaddrinfo("www.example.net", "3490", &hints, &servinfo);

for(p = res;p != NULL; p = p->ai_next) {
	void *addr;
	char *ipver;
	// get the pointer to the address itself,
	// different fields in IPv4 and IPv6:
	if (p->ai_family == AF_INET) { // IPv4
		struct sockaddr_in *ipv4 = (struct sockaddr_in *)p->ai_addr;
		addr = &(ipv4->sin_addr);
		ipver = "IPv4";
	} else { // IPv6
		struct sockaddr_in6 *ipv6 = (struct sockaddr_in6 *)p->ai_addr;
		addr = &(ipv6->sin6_addr);
		ipver = "IPv6";
	}
	// convert the IP to a string and print it:
	inet_ntop(p->ai_family, addr, ipstr, sizeof ipstr);
	printf("  %s: %s\n", ipver, ipstr);
}
```

socket() – A Connection End Point

- **Creates a communication end-point for a network connection**

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

  domain = PF_INET (IPv4) / PF_INET6 (IPv6)

  type = SOCK_STREAM (TCP) / SOCK_DGRAM (UDP)

  protocol = 0
- **Example:**

```
    s = socket (PF_INET, SOCK_STREAM, 0)
```

will create a TCP socket

- **The above call returns 0 on success and -1 on failure**

connect() – Client Attempting Server Connection

- **Called by the client to attempt a connection with the server**
- **Blocks until the server accepts it**

```
int connect(int sockfd, struct sockaddr* server,
socketlen_t server_len)
```

```
sockfd = socket
```

```
server = address of the server (returned by getaddrinfo)
```

- **Example:**

```
// lookup
getaddrinfo("www.example.com", "3490", &hints, &res);
// make a socket:
sockfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
// connect to server. Once successful, the socket becomes ready as the
endpoint
connect(sockfd, res->ai_addr, res->ai_addrlen);
```

close() – Close a Session

- Called by both the client and the server
- Signals end of a communication
- Internally, frees resources associated with a connection
  - Important for busy servers, also for MP6 ☺

```
int close(int sock)
```

- Example:

```
close (sock)
```

bind() – Server Attaching to a Port

- **A server process calls this to associate its socket to a given port**
- **Port number is used by the kernel to forward an incoming packet to a certain service's socket**

  `int bind(int sockfd, struct sockaddr* addr, socketlen_t addrlen)`

- **Example:**

```
// lookup
getaddrinfo(NULL, "3490", &hints, &res);

// make a socket:
sockfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);

// bind it to the port we passed in to getaddrinfo():
bind(sockfd, res->ai_addr, res->ai_addrlen);
```

listen() – Setting up Server

- This is a prerequisite before a connection is accepted
- Incoming connections wait in a queue before accepted, listen () sets the size of that queue

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

- Example:

```
listen (sockfd, 20); // 20 is good for most purposes, at least
for your data server in MP6
```

accept() – Server Accepting Client Connection

- **This is called by the server to accept a new client connection**

  `int accept(int sockfd, struct sockaddr* client, socketlen_t client_len)`

`sockfd = socket`

`client = will hold the client address details`

`client_len = address length`

- **Example:**

```
struct sockaddr client;
accept (sockfd, &client, sizeof (client));
```

send()/recv() – Finally Data

- **Called by both client and server to exchange data**
- **Blocks until the server accepts it**

```
int send(int sock, void* msg, size_t len, int flags)
int recv(int sock, void* msg, size_t len, int flags)
Msg = buffer pointer to send/receive data from/to
Len = sender: length of the message,
      receiver: buffer capacity (to avoid overflow)
```

- **Example:**

```
char *send_msg = "a sample message";
int sent_bytes = send (sockfd, send_msg, strlen
(send_msg)+1, 0);

char recv_buffer [1024];
int recv_len = recv (sockfd, recv_buffer, 1024, 0);
```

SOURCES

- RICE COMP-221 Lecture Notes in Networking
  - Acknowledgment: Prof. Cox
- TAMU CSCE-313 Lecture Notes in Networking
  - Acknowledgment: Profs Gu, Bettati, Ahmed
- Russ Haynal - http://navigators.com/
- U-Wisconsin CS-354 Lecture Notes in Networking
  - Acknowledgment: Prof. Arpaci-Dusseau
- Socket Programming 101
  - Acknowledgment: Vivek Ramachandran
- U-Illinois CS241 Lecture Notes in Networking
  - Acknowledgment: Prof. Angrave
- Beej's Guide to Network Programming, Ver. 3.0.15