Distributed System Design

David E. Culler

CS162 – Operating Systems and Systems Programming

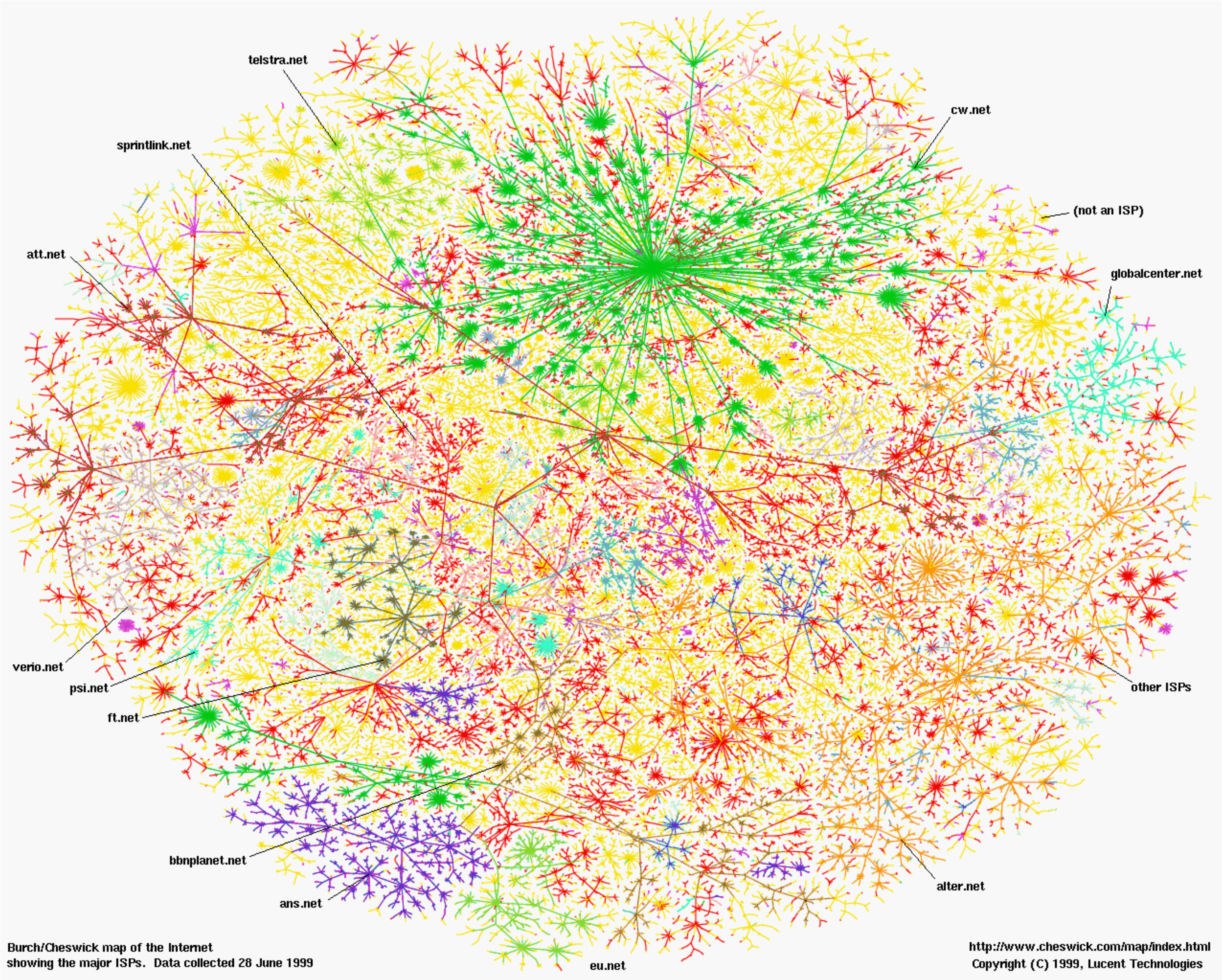
http://cs162.eecs.berkeley.edu/

Lecture 31

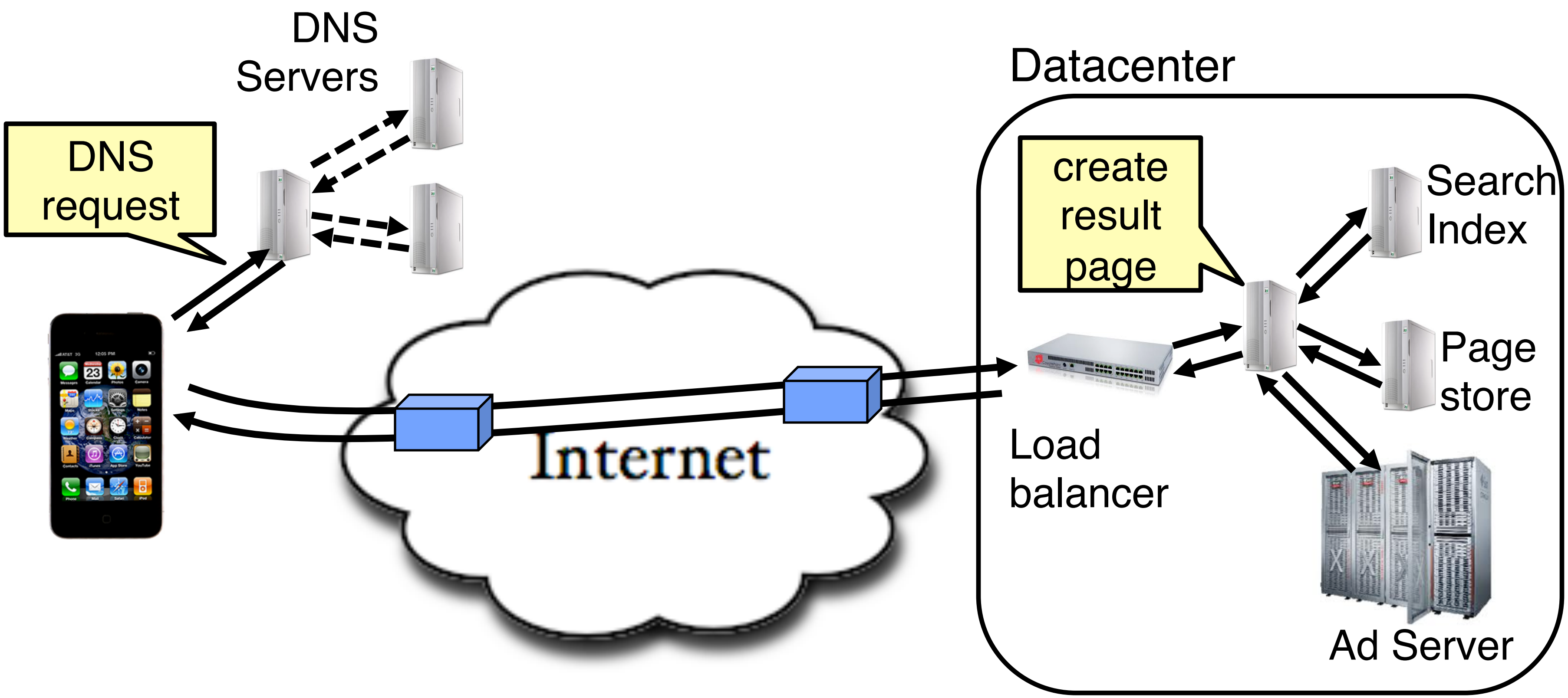
Nov 10, 2014

Read: end-2-end
HW 5: Due 11/12
Mid 2: 11/14
Proj 3: due 12/8

Greatest Artifact of Human Civilization …

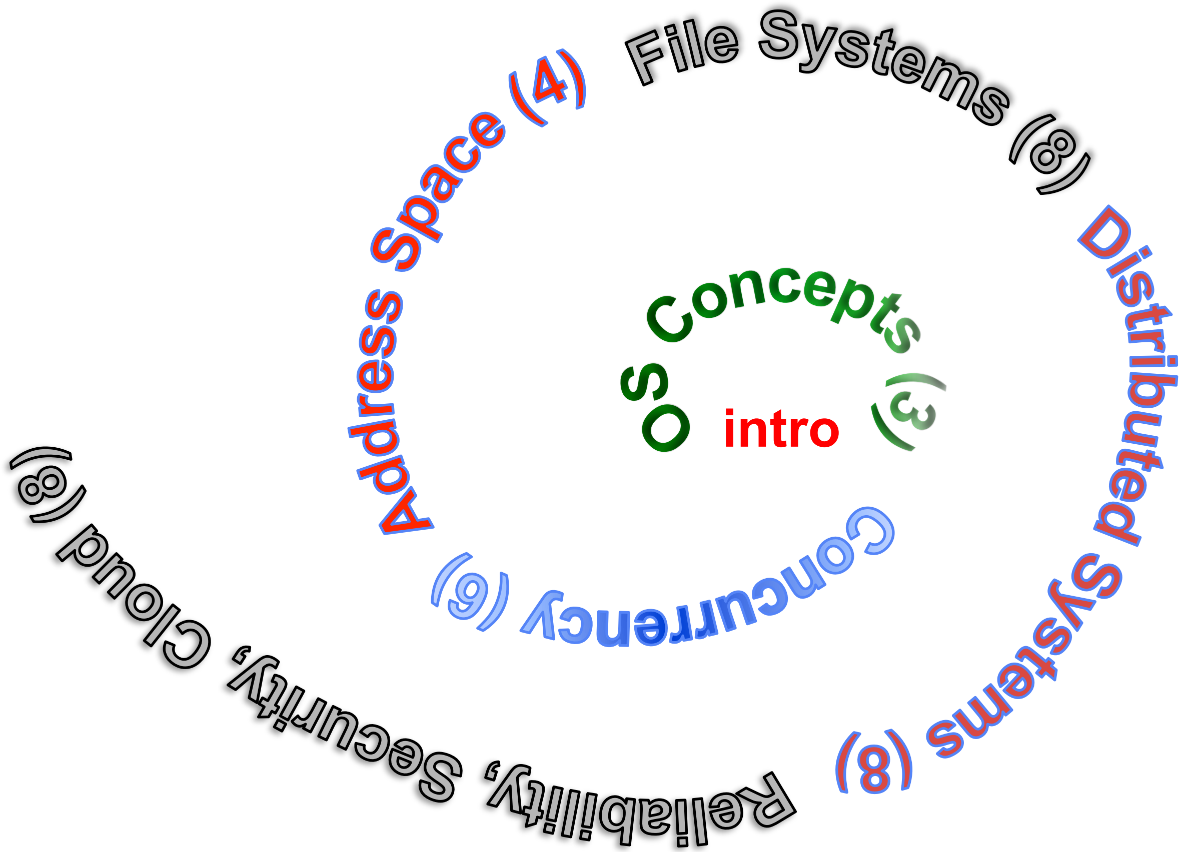


Example: What's in a Search Query?

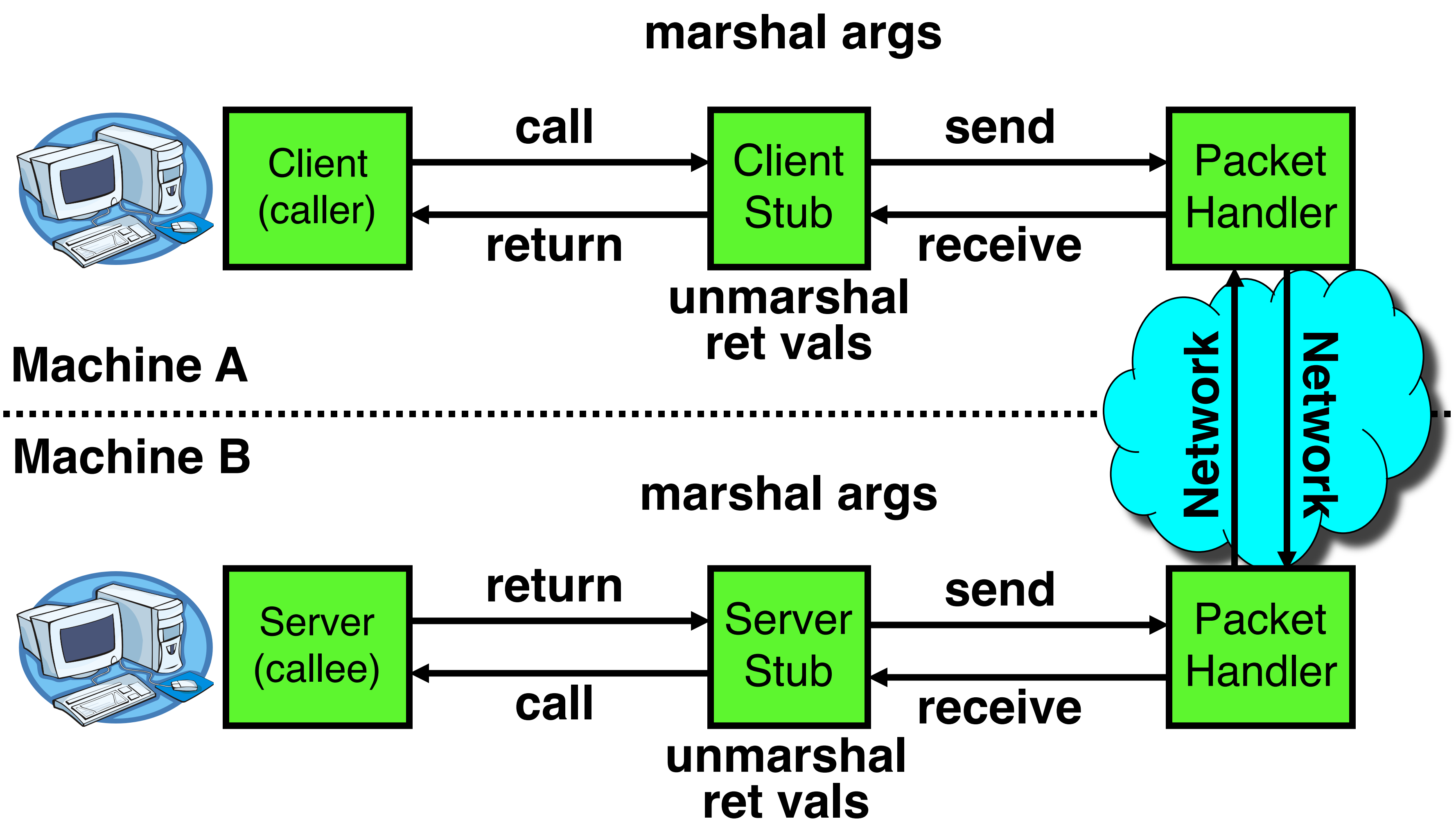


- **Complex interaction of multiple components in multiple administrative domains**
  - **Systems, services, protocols, …**

Course Structure: Spiral

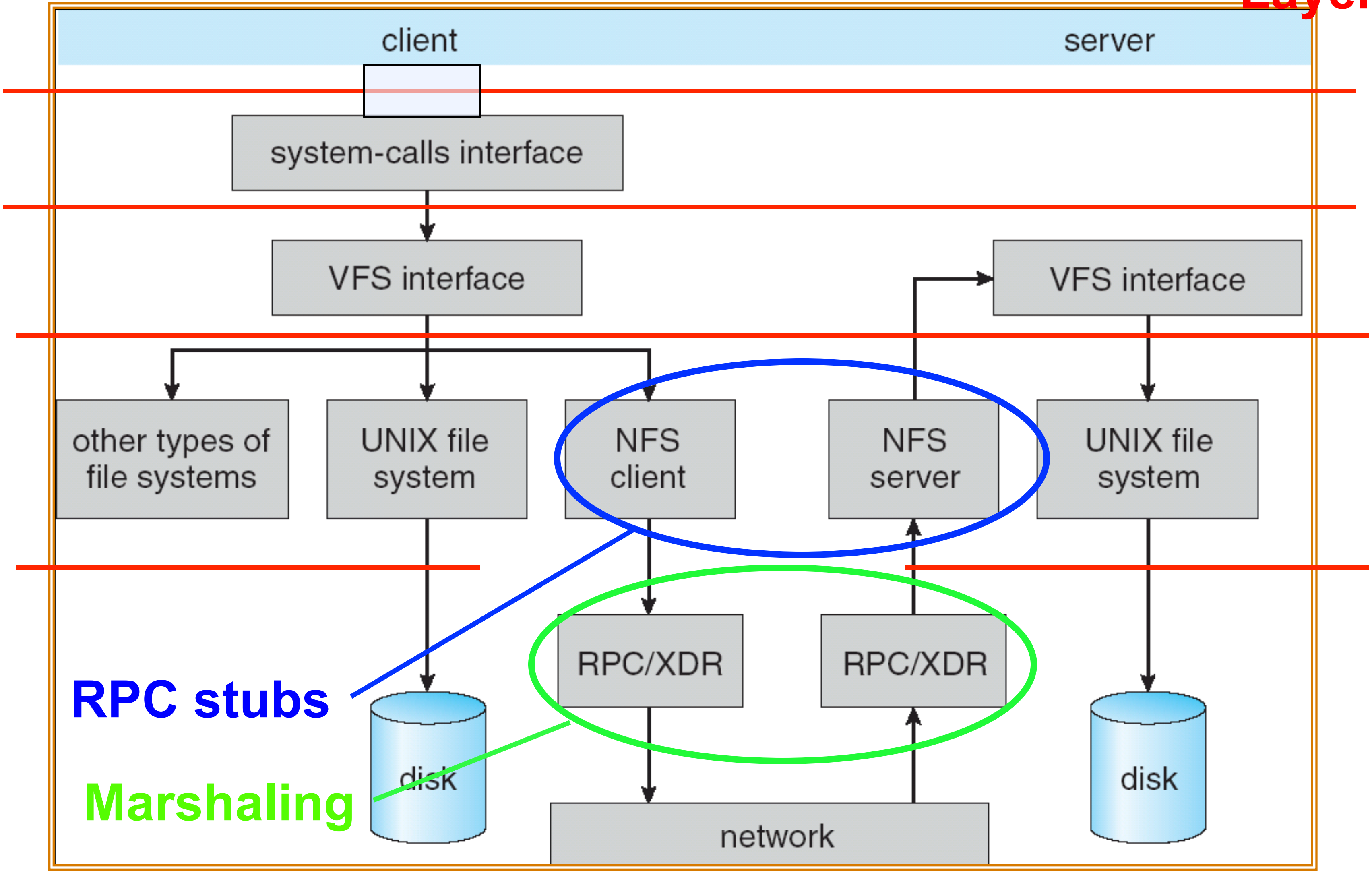


Review: Remote Procedure Call



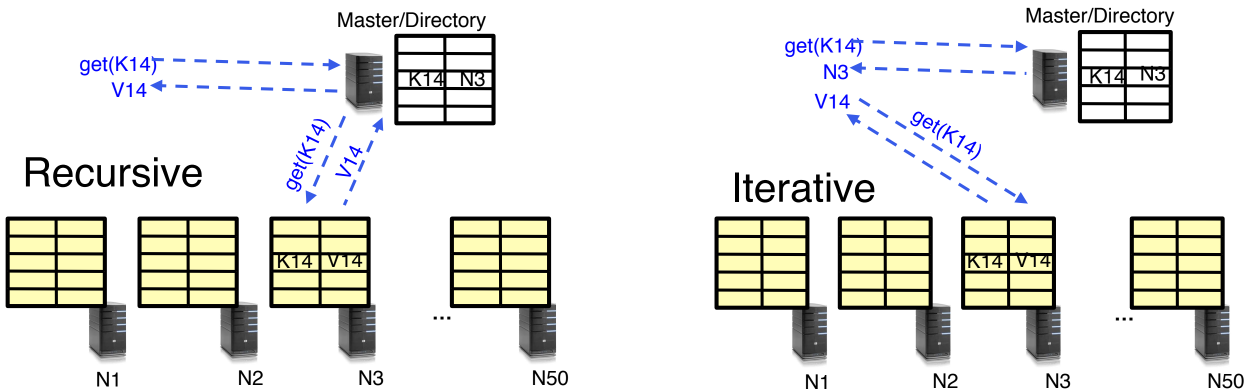
Review: Schematic View of NFS Architecture

Layering



Protocol Trade-offs

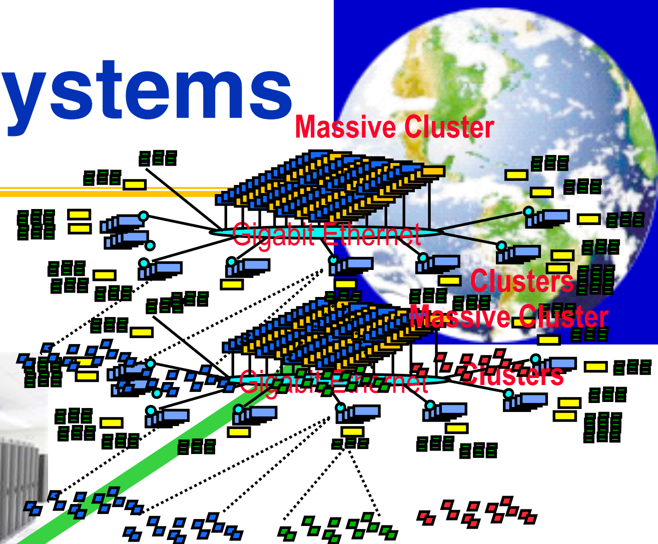
Discussion: Iterative vs. Recursive Query



- Recursive Query:
  - Advantages:
    - » Faster, as typically master/directory closer to nodes
    - » Easier to maintain consistency, as master/directory can serialize puts()/gets()
  - Disadvantages: scalability bottleneck, as all "Values" go through master/directory
- Iterative Query
  - Advantages: more scalable
  - Disadvantages: slower, harder to enforce data consistency

11/3/2014 Ion Stoica CS162 ©UCB Fall 2014 13

Societal Scale Information Systems

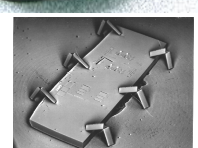


- **The world is a large distributed system**
  - **Microprocessors in everything**
  - **Vast infrastructure behind them**

Internet Connectivity

Scalable, Reliable, Secure Services

Databases
Information Collection
Remote Storage
Online Games
Commerce
…



MEMS for Sensor Nets

What Is A Protocol?

- A protocol is an agreement on how to communicate
- Includes
  - Syntax: how a communication is specified & structured
    - » Format, order messages are sent and received
  - Semantics: what a communication means
    - » Actions taken when transmitting, receiving, or when a timer expires
- Described formally by a state machine
  - Often represented as a message transaction diagram

Examples of Protocols in Human Interactions

- Telephone
  1. (Pick up / open up the phone)
  2. Listen for a dial tone / see that you have service
  3. Dial
  4. Should hear ringing …
  5. Callee: "Hello?"
  6. Caller: "Hi, it's John…."
     Or: "Hi, it's me" (← what's *that* about?)
  7. Caller: "Hey, do you think … blah blah blah …" pause
  8. Callee: "Yeah, blah blah blah …" pause
  9. Caller: Bye
  10. Callee: Bye
  11. Hang up

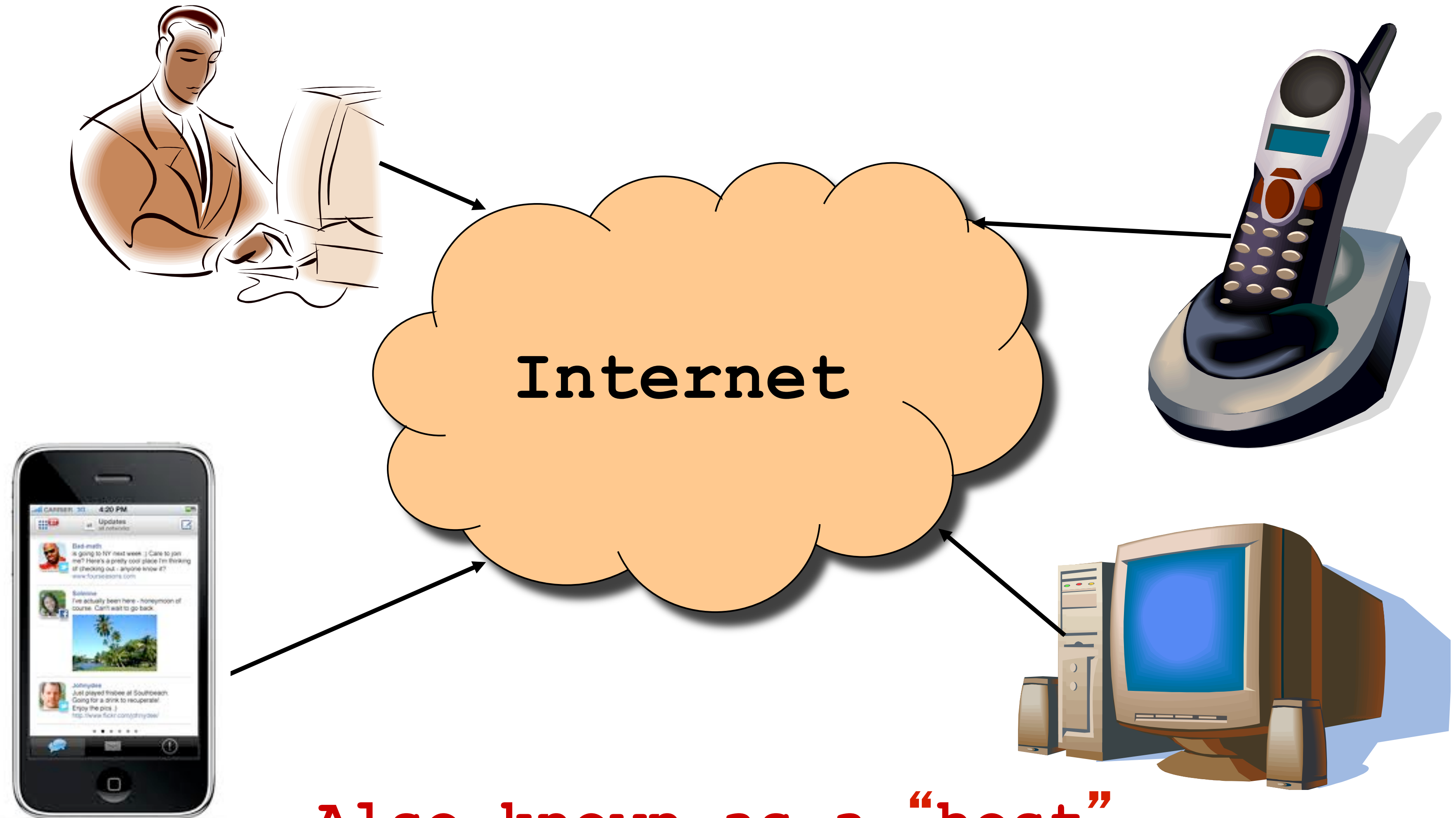
Protocols in Human Interactions

Asking a question

1. Raise your hand
2. Wait to be called on

3. Or: wait for speaker to pause and vocalize

End System: Computer on the ‘Net



Also known as a “host”…

What's in a name?

Namespaces for communication

- Hostname
  - www.eecs.berkeley.edu
- IP address
  - 128.32.244.172 (ipv6?)
- Port Number
  - 0-1023 are "well known" or "system" ports
    - Superuser privileges to bind to one
  - 1024 – 49151 are "registered" ports (registry)
    - Assigned by IANA for specific services
  - 49152–65535 ($2^{15}+2^{14}$ to $2^{16}-1$) are "dynamic" or "private"
    - Automatically allocated as "ephemeral Ports"

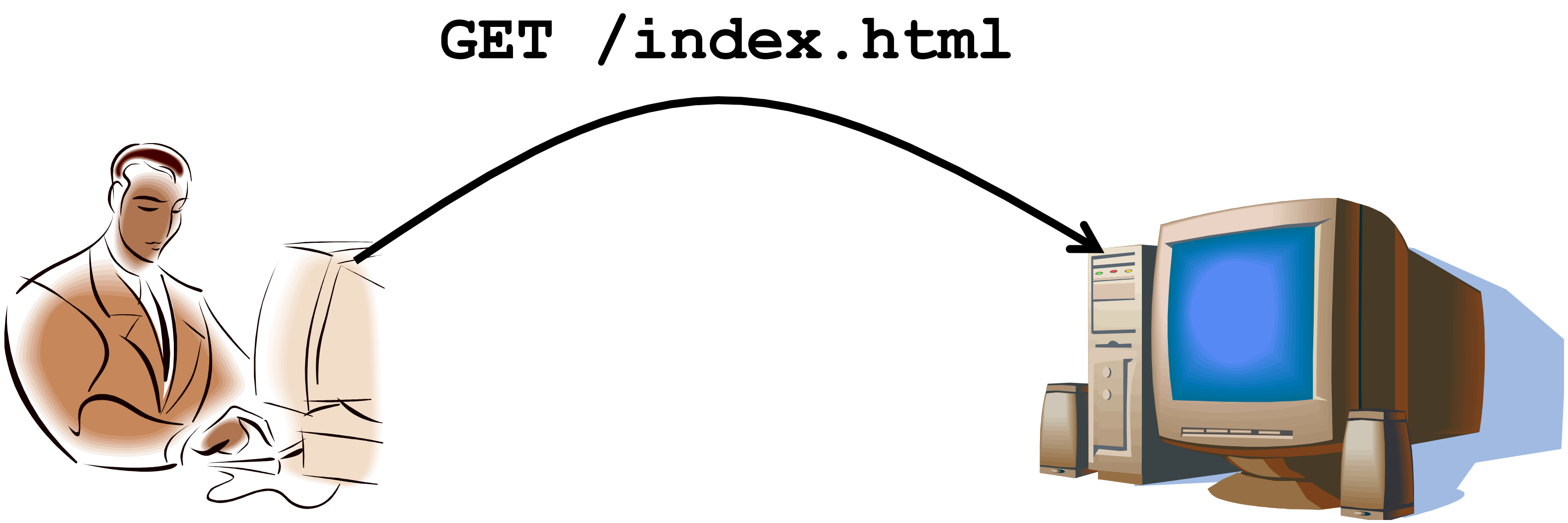
Recall:

Client: getting the server address

```
struct hostent *buildServerAddr(struct sockaddr_in *serv_addr,
                                char *hostname, int portno) {
  struct hostent *server;
  /* Get host entry associated with a hostname or IP address */
  server = gethostbyname(hostname);
  if (server == NULL) {
    fprintf(stderr,"ERROR, no such host\n");
    exit(1);
  }

  /* Construct an address for remote server */
  memset((char *) serv_addr, 0, sizeof(struct sockaddr_in));
  serv_addr->sin_family = AF_INET;
  bcopy((char *)server->h_addr,
        (char *)&(serv_addr->sin_addr.s_addr), server->h_length);
  serv_addr->sin_port = htons(portno);

return server;
}
```

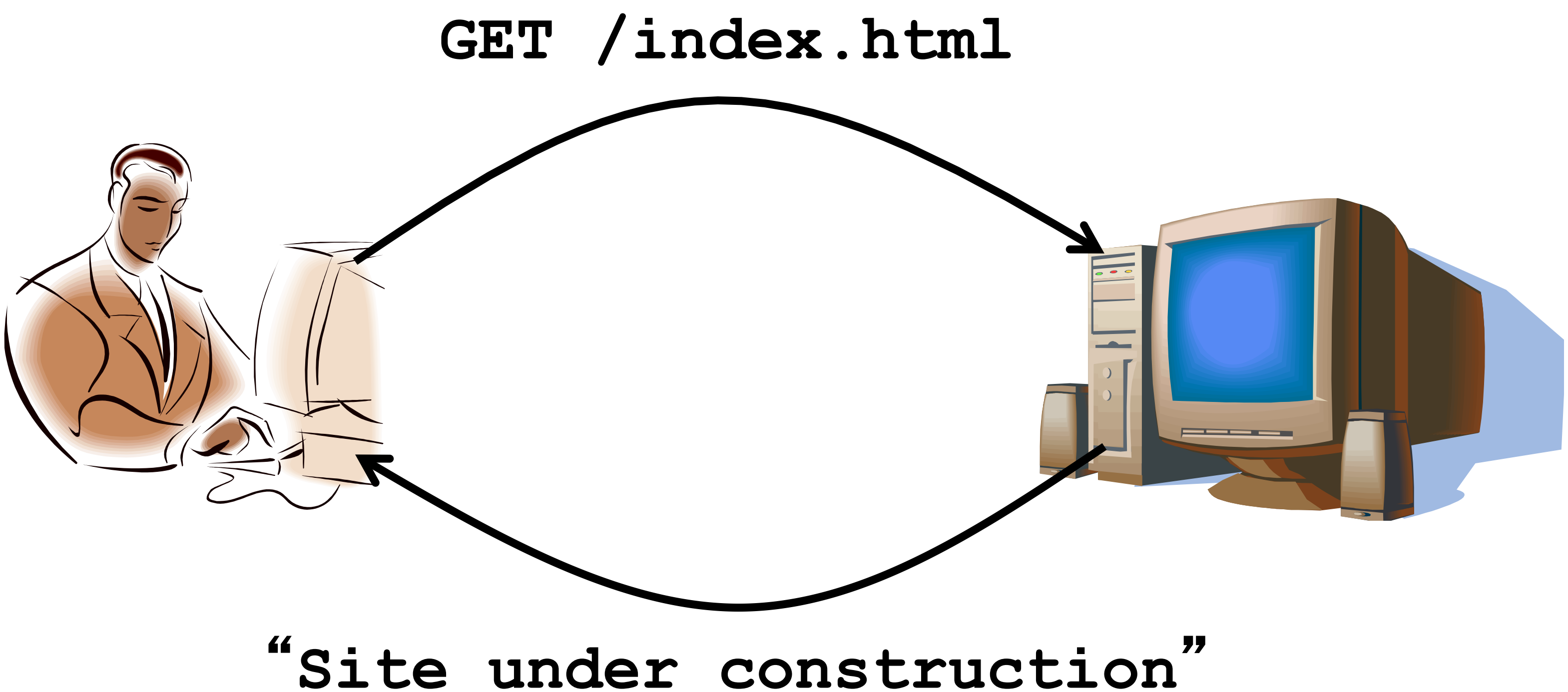
Clients and Servers

- Client program
  - Running on end host
  - Requests service
  - E.g., Web browser



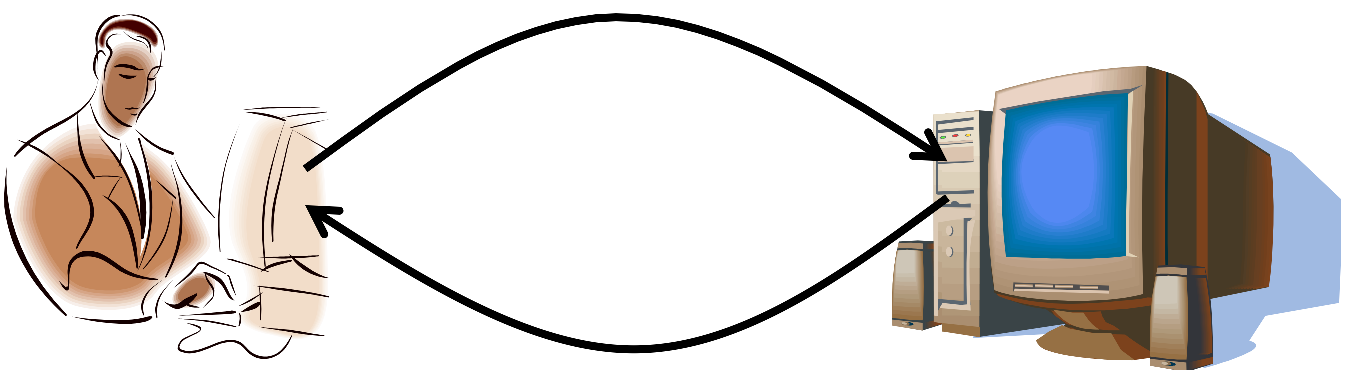
Clients and Servers

- Client program
  - Running on end host
  - Requests service
  - E.g., Web browser
- Server program
  - Running on end host
  - Provides service
  - E.g., Web server



Client-Server Communication

- Client "sometimes on"
  - Initiates a request to the server when interested
  - E.g., Web browser on your laptop or cell phone
  - Doesn't communicate directly with other clients
  - Needs to know the server's address
- Server is "always on"
  - Services requests from many client hosts
  - E.g., Web server for the *www.cnn.com* Web site
  - Doesn't initiate contact with the clients
  - Needs a fixed, well-known address



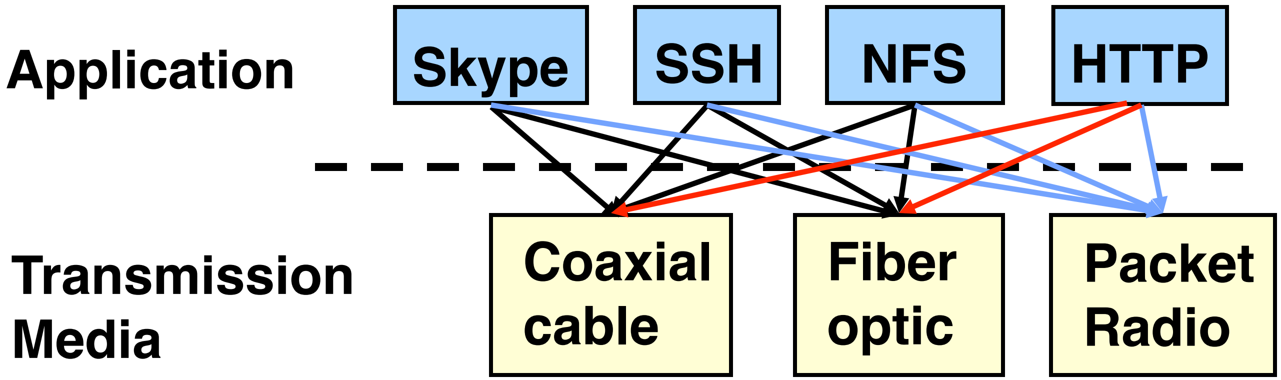
Peer-to-Peer Communication

- No always-on server at the center of it all
  - Hosts can come and go, and change addresses
  - Hosts may have a different address each time
- Example: peer-to-peer file sharing (e.g., BitTorrent)
  - Any host can request files, send files, query to find where a file is located, respond to queries, and forward queries
  - Scalability by harnessing millions of peers
  - Each peer acting as both a client and server

The Problem

- **Many different applications**
  - **email, web, P2P, etc.**
- **Many different network styles and technologies**
  - **Wireless vs. wired vs. optical, etc.**
- **How do we organize this mess?**

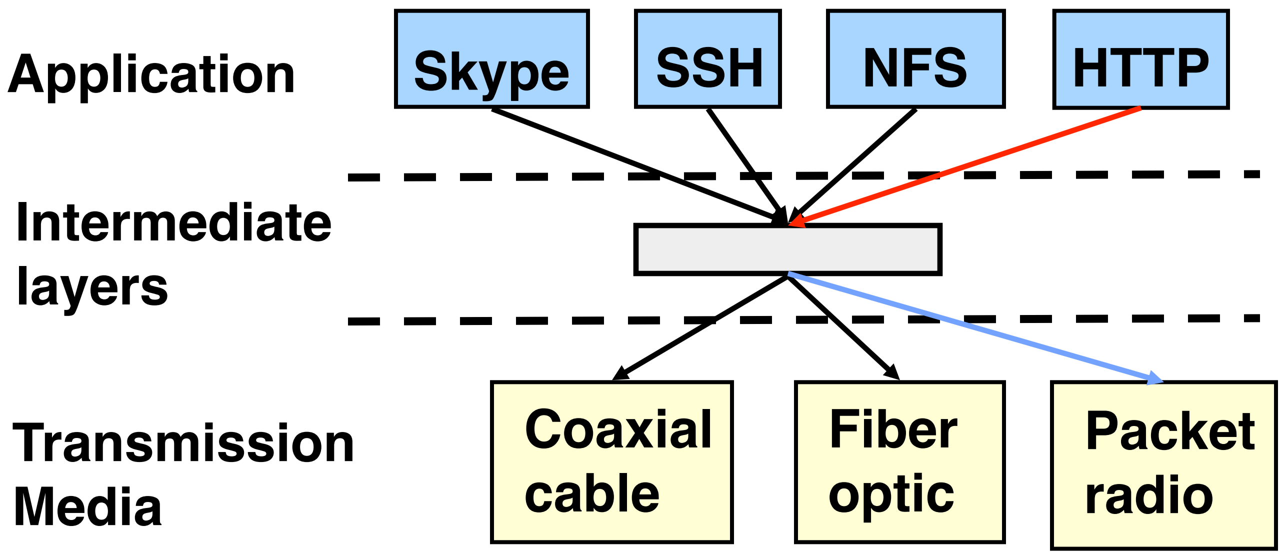
The Problem (cont'd)



- Re-implement every application for every technology?
- No! But how does the Internet design avoid this?

Solution: Intermediate Layers

- Introduce intermediate layers that provide set of abstractions for various network functionality & technologies
  - A new app/media implemented only once
  - Variation on "add another level of indirection"



Software System Modularity

Partition system into modules & abstractions:

- Well-defined interfaces give flexibility
  - *Hides* implementation - thus, it can be freely changed
  - Extend functionality of system by adding new modules
- E.g., libraries encapsulating set of functionality
- E.g., programming language + compiler abstracts away not only how the particular CPU works …
  - … but also the basic computational model
- Well-defined interfaces hide information
  - Present high-level abstractions
  - But can impair performance

Network System Modularity

Like software modularity, but:

- **Implementation distributed across many machines (routers and hosts)**

- **Must decide:**
  - **How to break system into modules:**
    - » **Layering**
  - **What functionality does each module implement:**
    - » **End-to-End Principle: don't put it in the network if you can do it in the endpoints.**

- **We will address these choices more in next lecture**

Layering: A Modular Approach

- **Partition the system**
  - Each layer solely relies on services from layer below
  - Each layer solely exports services to layer above
- **Interface between layers defines interaction**
  - Hides implementation details
  - Layers can change without disturbing other layers

Protocol Standardization

- **Ensure communicating hosts speak the same protocol**
  - **Standardization to enable multiple implementations**
  - **Or, the same folks have to write all the software**
- **Standardization: Internet Engineering Task Force**
  - **Based on working groups that focus on specific issues**
  - **Produces "Request For Comments" (RFCs)**
    - » **Promoted to standards via rough consensus and running code**
  - **IETF Web site is http://www.ietf.org/**
  - **RFCs archived at http://www.rfc-editor.org/**
- **De facto standards: same folks writing the code**
  - **P2P file sharing, Skype, <your protocol here>…**

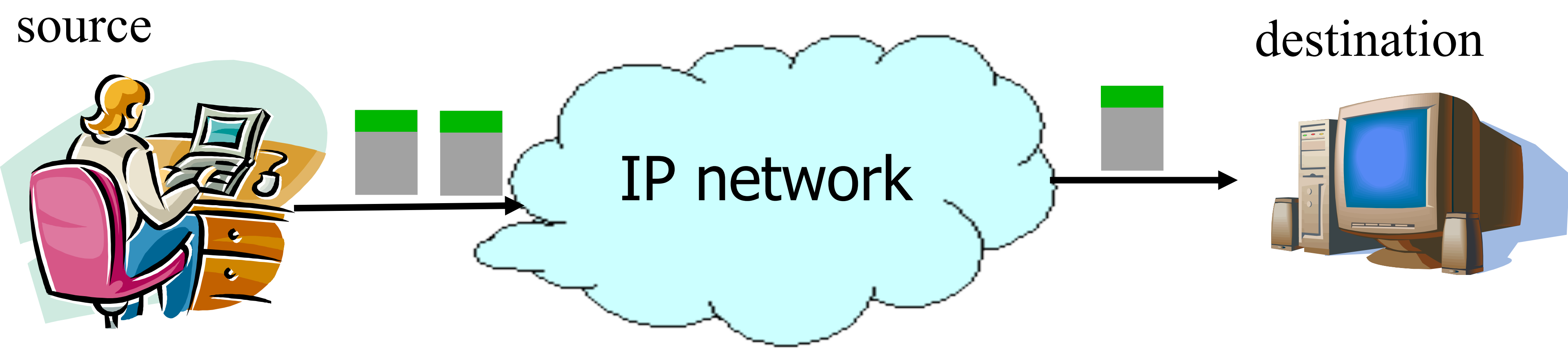
Administration Break

- Midterm 2: Friday 11/14 6-7:30 @ 1 Pimentel
  - Bring one 2-sides 8.5 x 11
  - Email cs162@eecs for conflicts
- Study guide answers releases
- Review session in Section this week
- Focused on Lectures 12-27
  - But assumes earlier material

- Project 3: Key-Value Store in Java !!!
- Less readings ahead – lecture even more important

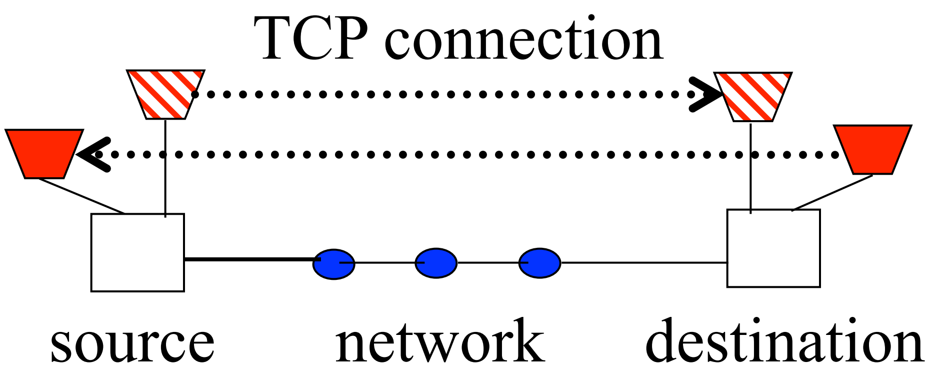
Example: The Internet Protocol (IP): "Best-Effort" Packet Delivery

- **Datagram packet switching**
  - Send data in packets
  - Header with source & destination address
- **Service it provides:**
  - Packet arrives quickly (if it does)
  - Packets may be lost
  - Packets may be corrupted
  - Packets may be delivered out of order



Example: Transmission Control Protocol (TCP)

- **Communication service**
  - Ordered, reliable byte stream
  - Simultaneous transmission in both directions
- **Key mechanisms at end hosts**
  - Retransmit lost and corrupted packets
  - Discard duplicate packets and put packets in order
  - Flow control to avoid overloading the receiver buffer
  - Congestion control to adapt sending rate to network load



Recall: Socket Protocol

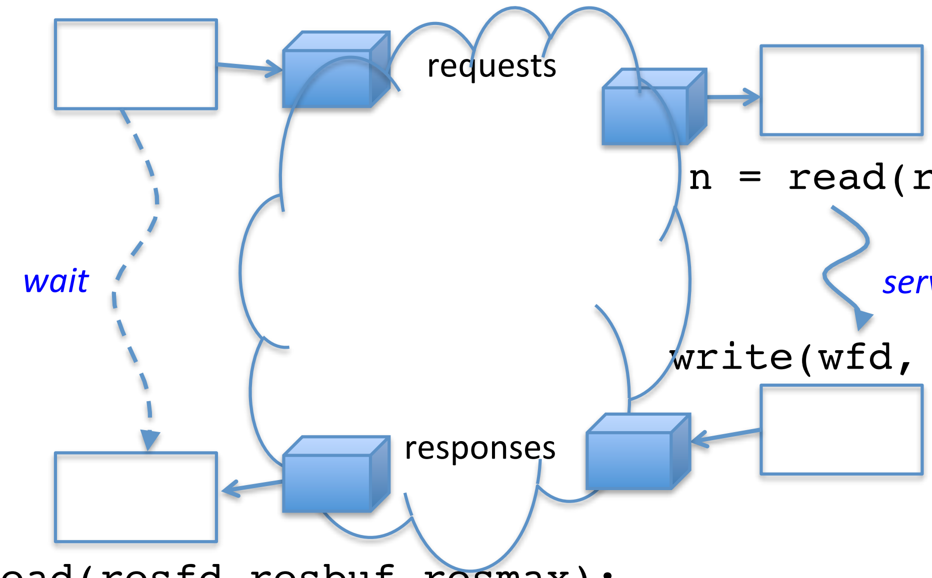
Recall: Sockets

Request Response Protocol

Client (issues requests)

Server (performs operations)

`write(rqfd, rqbuf, buflen);`



`n = read(rfd,rbuf,rmax);`

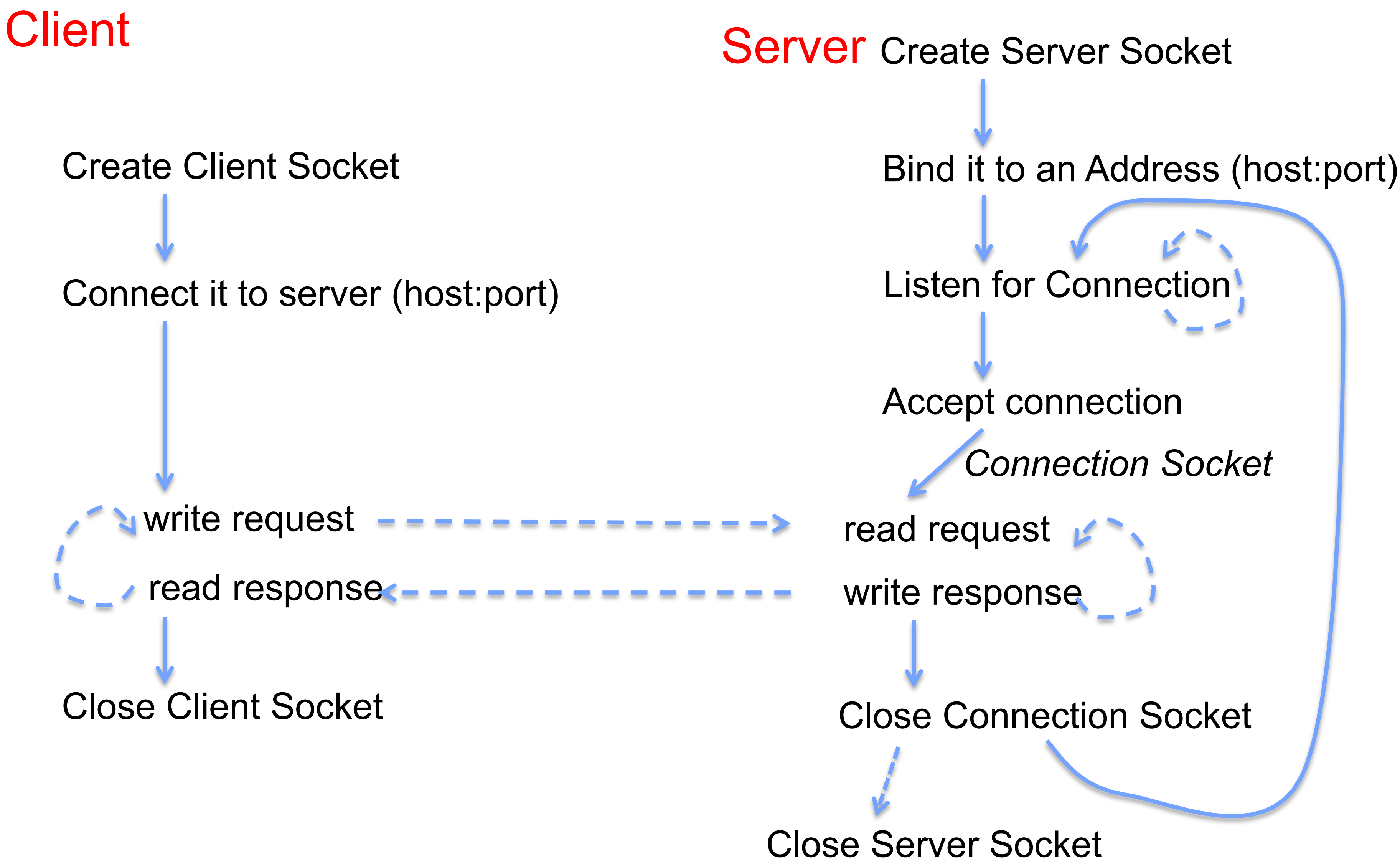
`write(wfd, respbuf, len);`

`n = read(resfd,resbuf,resmax);`

Recall: Socket creation and connection

- File systems provide a collection of permanent objects in structured name space
  - Processes open, read/write/close them
  - Files exist independent of the processes
- Sockets provide a means for processes to communicate (transfer data) to other processes.
- Creation and connection is more complex
- Form 2-way pipes between processes
  - Possibly worlds away

Recall: Sockets in concept



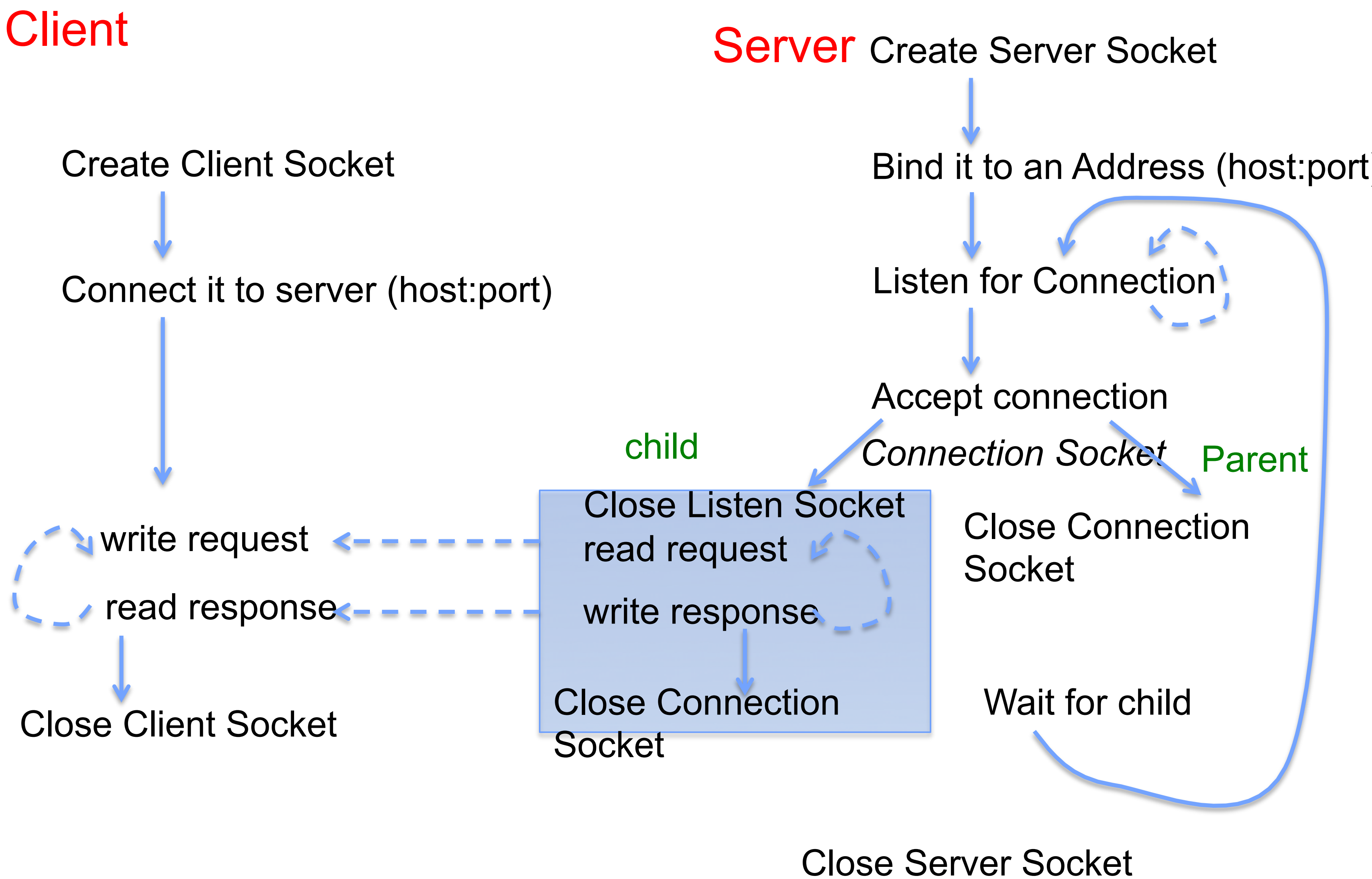
Client Protocol

```
char *hostname;
int sockfd, portno;
struct sockaddr_in serv_addr;
struct hostent *server;

server = buildServerAddr(&serv_addr, hostname, portno);

/* Create a TCP socket */
sockfd = socket(AF_INET, SOCK_STREAM, 0)

/* Connect to server on port */
connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)
printf("Connected to %s:%d\n",server->h_name, portno);

/* Carry out Client-Server protocol */
client(sockfd);

/* Clean up on termination */
close(sockfd);
```

Server Protocol (v1)

```
/* Create Socket to receive requests*/
lstnsockfd = socket(AF_INET, SOCK_STREAM, 0);

/* Bind socket to port */
bind(lstnsockfd, (struct sockaddr *)&serv_addr,sizeof(serv_addr));
while (1) {
/* Listen for incoming connections */
   listen(lstnsockfd, MAXQUEUE);

/* Accept incoming connection, obtaining a new socket for it */
   consockfd = accept(lstnsockfd, (struct sockaddr *) &cli_addr,
                      &clilen);

   server(consockfd);

   close(consockfd);
  }
close(lstnsockfd);
```

Sockets in concept: fork

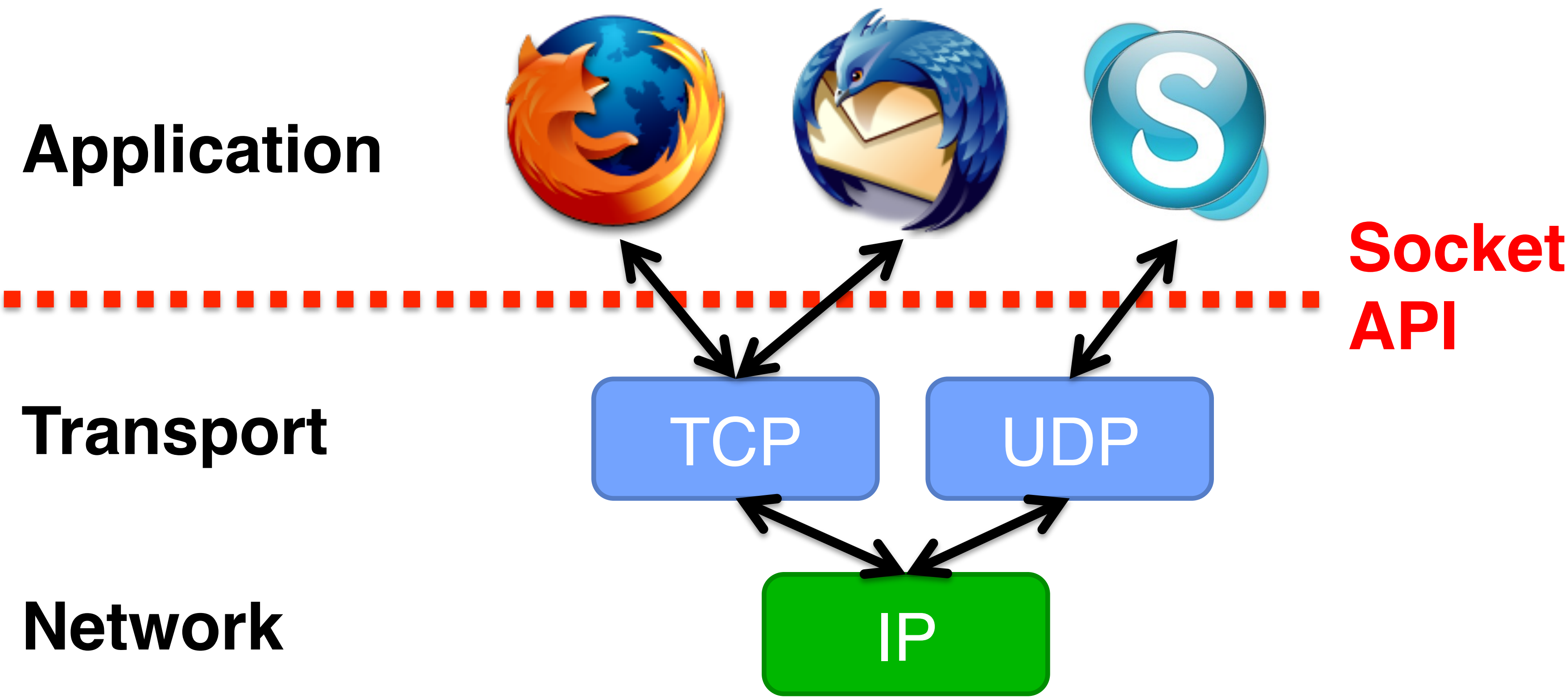


Server Protocol (v2)

```
while (1) {
    listen(lstnsockfd, MAXQUEUE);
    consockfd = accept(lstnsockfd, (struct sockaddr *) &cli_addr,
                                                &clilen);
    cpid = fork();                  /* new process for connection */
    if (cpid > 0) {                 /* parent process */
      close(consockfd);
      tcpid = wait(&cstatus);
    } else if (cpid == 0) {          /* child process */
      close(lstnsockfd);            /* let go of listen socket */

      server(consockfd);

      close(consockfd);
      exit(EXIT_SUCCESS);             /* exit child normally */
    }
  }
close(lstnsockfd);
```

Socket API

- **Base level Network programming interface**



BSD Socket API

- **Created at UC Berkeley (1980s)**
- **Most popular network API**
- **Ported to various OSes, various languages**
  - **Windows Winsock, BSD, OS X, Linux, Solaris, …**
  - **Socket modules in Java, Python, Perl, …**
- **Similar to Unix file I/O API**
  - **In the form of *file descriptor* (sort of handle).**
  - **Can share same `read()`/`write()`/`close()` system calls**

TCP: Transport Control Protocol

- **Reliable, in-order, and at most once delivery**
- **Stream oriented: messages can be of arbitrary length**
- **Provides multiplexing/demultiplexing to IP**
- **Provides congestion and flow control**
- **Application examples: file transfer, chat**

TCP Service

1) Open connection: 3-way handshaking

2) Reliable byte stream transfer from
   (IPa, TCP_Port1) to (IPb, TCP_Port2)
   - Indication if connection fails: Reset

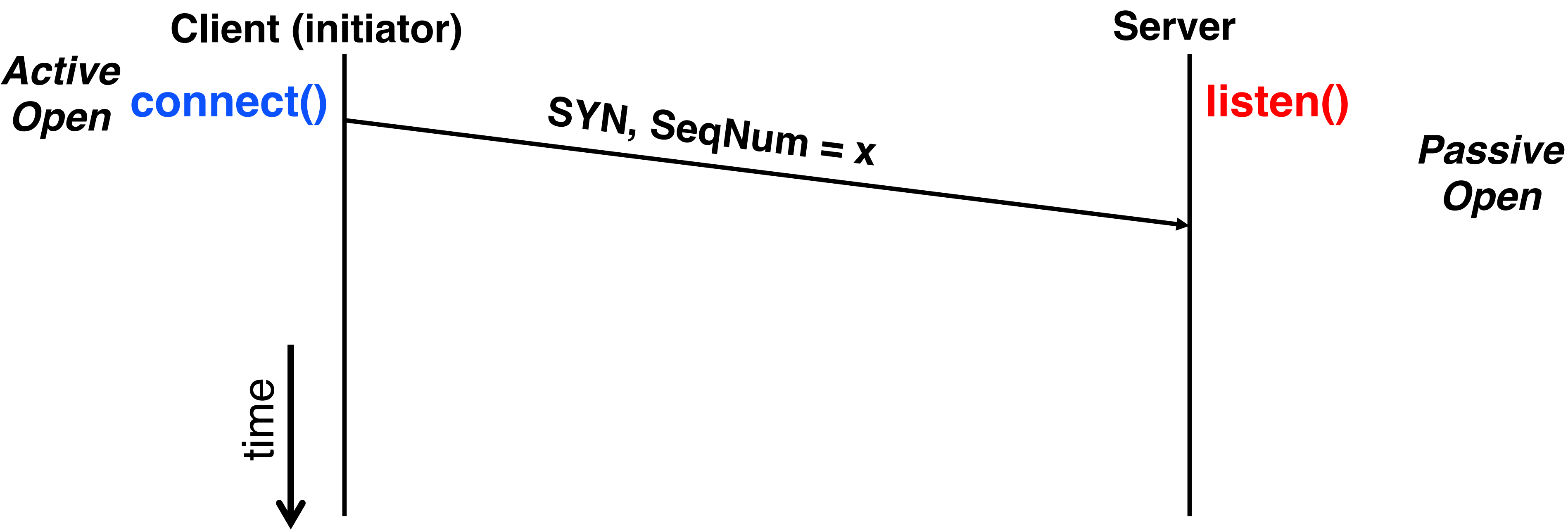
3) Close (tear-down) connection

Open Connection: 3-Way Handshaking

- Goal: agree on a set of parameters, i.e., the start sequence number for each side
  - Starting sequence number: sequence of first byte in stream
  - Starting sequence numbers are random

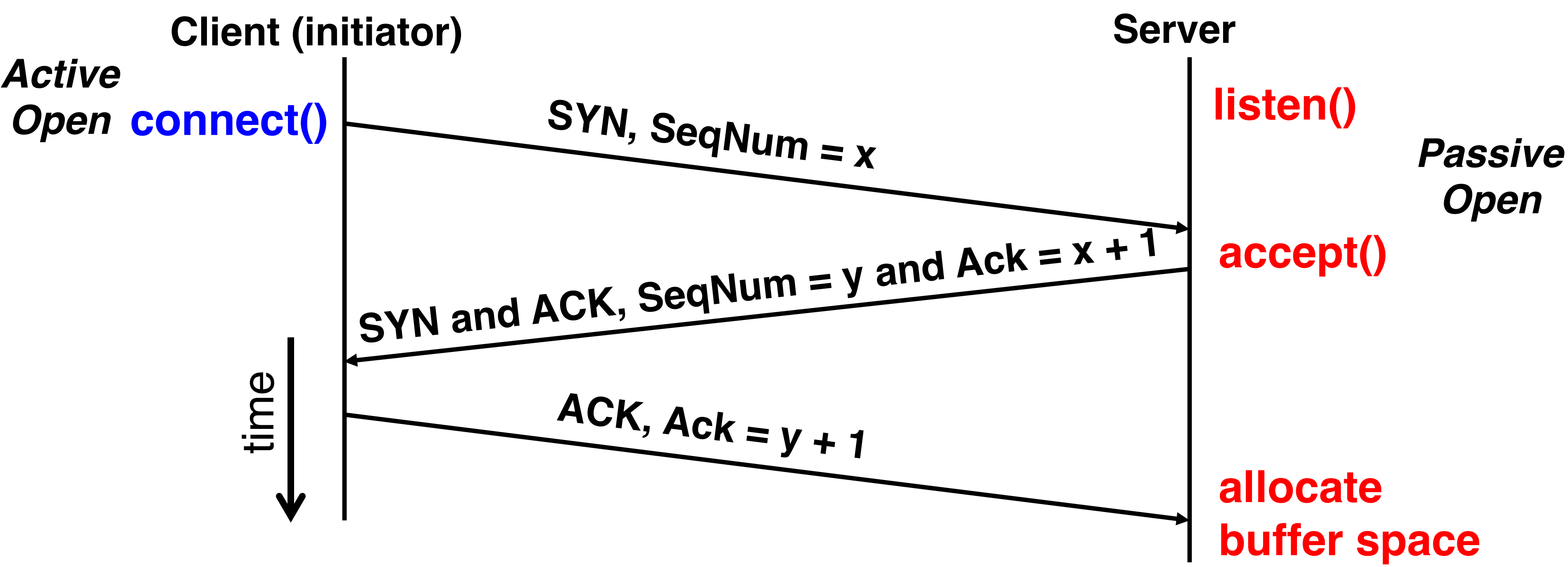
Open Connection: 3-Way Handshaking

- **Server waits for new connection calling listen()**
- **Sender call connect() passing socket which contains server's IP address and port number**
  - **OS sends a special packet (SYN) containing a proposal for first sequence number, x**



Open Connection: 3-Way Handshaking

- **If it has enough resources, server calls accept() to accept connection, and sends back a SYN ACK packet containing**
  - Client's sequence number incremented by one, (x + 1)
    - » Why is this needed?
  - A sequence number proposal, y, for first byte server will send



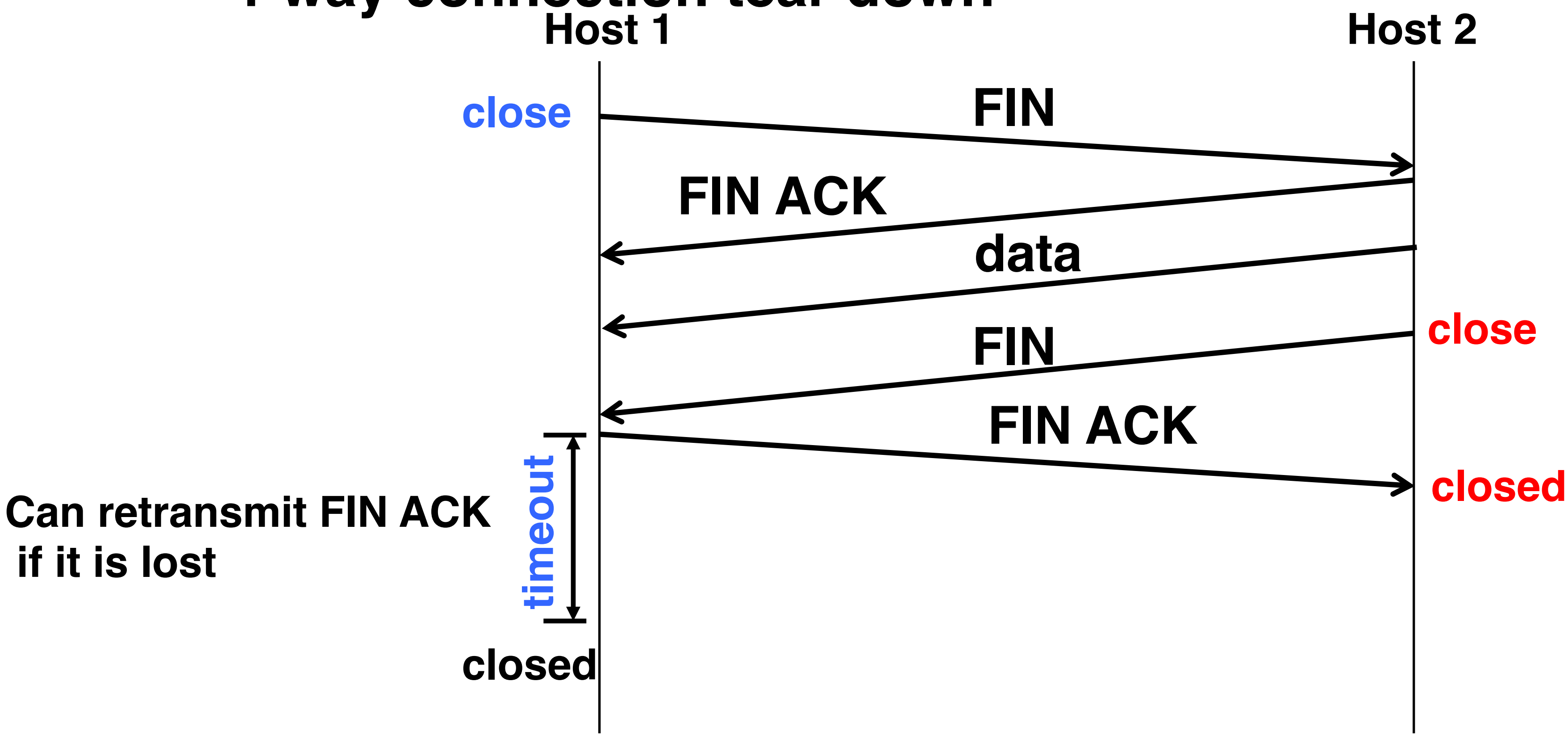
3-Way Handshaking (cont'd)

- **Three-way handshake adds 1 RTT delay**

- **Why?**
  - **Congestion control: SYN (40 byte) acts as cheap probe**
  - **Protects against delayed packets from other connection (would confuse receiver)**

Close Connection

- Goal: both sides agree to close the connection
- 4-way connection tear down



Quiz 15.2: Protocols

- Q1: True _ False _ Protocols specify the syntax and semantics of communication
- Q2: True _ False _ Protocols specify the implementation
- Q3: True _ False _ Layering helps to improve application performance
- Q4: True _ False _ "Best Effort" packet delivery ensures that packets are delivered in order
- Q5: True _ False _ In p2p systems a node is both a client and a server
- Q6: True _ False _ TCP ensures that each packet is delivered within a predefined amount of time

Quiz 15.2: Protocols

- Q1: True X False _ Protocols specify the syntax and semantics of communication
- Q2: True _ False X Protocols specify the implementation
- Q3: True _ False X Layering helps to improve application performance
- Q4: True _ False X "Best Effort" packet delivery ensures that packets are delivered in order
- Q5: True X False _ In p2p systems a node is both a client and a server
- Q6: True _ False X TCP ensures that each packet is delivered within a predefined amount of time

Summary

- Important roles of
  – Protocols, standardization
  – Clients, servers, peer-to-peer
- A layered architecture is a powerful means for organizing complex networks
  – But, layering has its drawbacks too
- Next lecture
  – Layering
  – End-to-End arguments (please read the paper before lecture!)