

# Some network apps

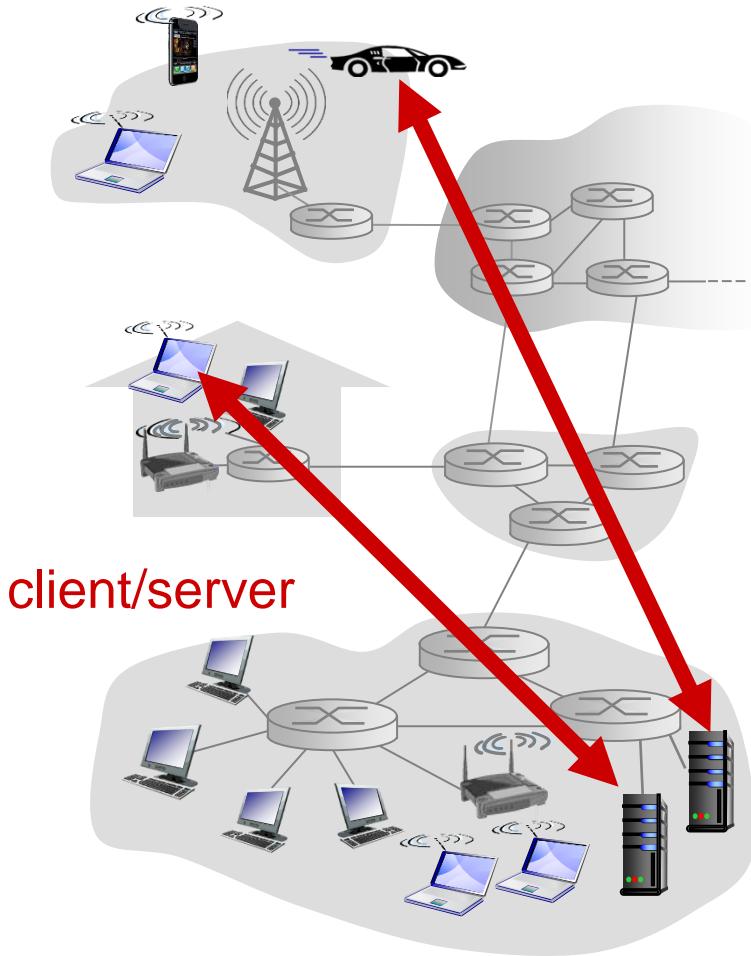
- ❖ e-mail
- ❖ web
- ❖ text messaging
- ❖ remote login
- ❖ P2P file sharing
- ❖ multi-user network games
- ❖ streaming stored video  
(YouTube, Hulu, Netflix)
- ❖ voice over IP (e.g., Skype)
- ❖ real-time video conferencing
- ❖ social networking
- ❖ search
- ❖ ...
- ❖ ...

# Application architectures

possible structure of applications:

- ❖ client-server
- ❖ peer-to-peer (P2P)

# Client-server architecture



## server:

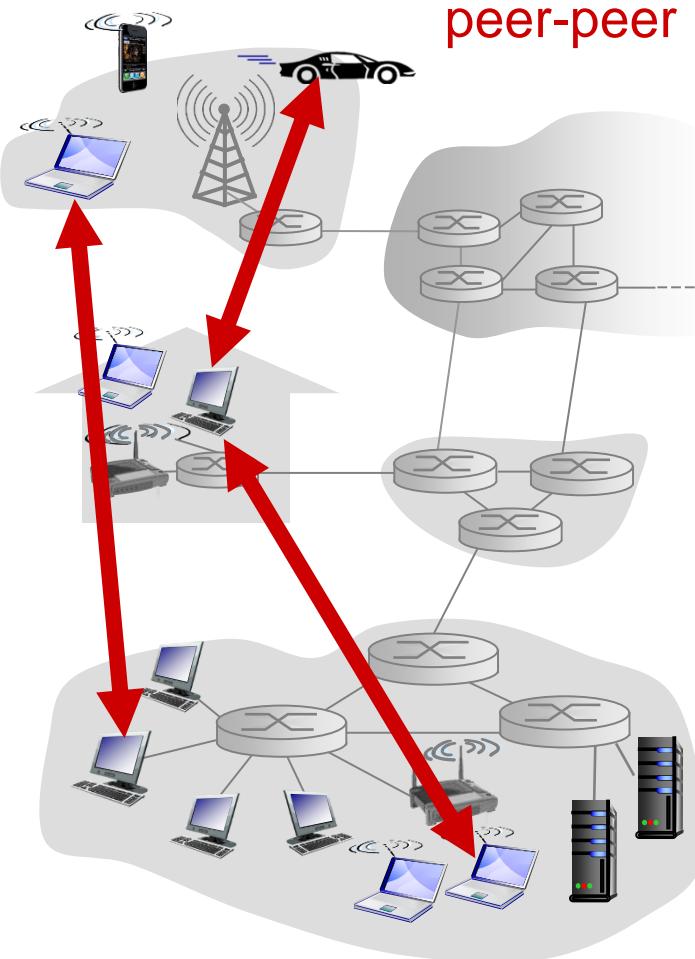
- ❖ always-on host
- ❖ permanent IP address
- ❖ data centers for scaling

## clients:

- ❖ communicate with server
- ❖ may be intermittently connected
- ❖ may have dynamic IP addresses
- ❖ do not communicate directly with each other

# P2P architecture

- ❖ no always-on server
- ❖ arbitrary end systems directly communicate
- ❖ peers are intermittently connected and change IP addresses
  - complex management



# App-layer protocol defines

- ❖ types of messages exchanged,
  - e.g., request, response
- ❖ message syntax:
  - what fields in messages & how fields are outlined
- ❖ message semantics
  - meaning of information in fields
- ❖ rules for when and how processes send & respond to messages

# What transport service does an app need?

## data integrity

- ❖ some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- ❖ other apps (e.g., audio) can tolerate some loss

## timing

- ❖ some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”

## throughput

- ❖ some apps (e.g., multimedia) require minimum amount of throughput to be “effective”
- ❖ other apps (“elastic apps”) make use of whatever throughput they get

## security

- ❖ encryption, data integrity,

...

# Internet apps: application, transport protocols

<b>application</b>	<b>application layer protocol</b>	<b>underlying transport protocol</b>
e-mail	SMTP	TCP
remote terminal access	Telnet	TCP
Web	HTTP	TCP
file transfer	FTP	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

# Web and HTTP

*First, a review...*

- ❖ *web page* consists of *objects*
- ❖ object can be HTML file, JPEG image, Java applet, audio file,...
- ❖ web page consists of *base HTML-file* which includes *several referenced objects*
- ❖ each object is addressable by a *URL*, e.g.,

www.someschool.edu/someDept/pic.gif

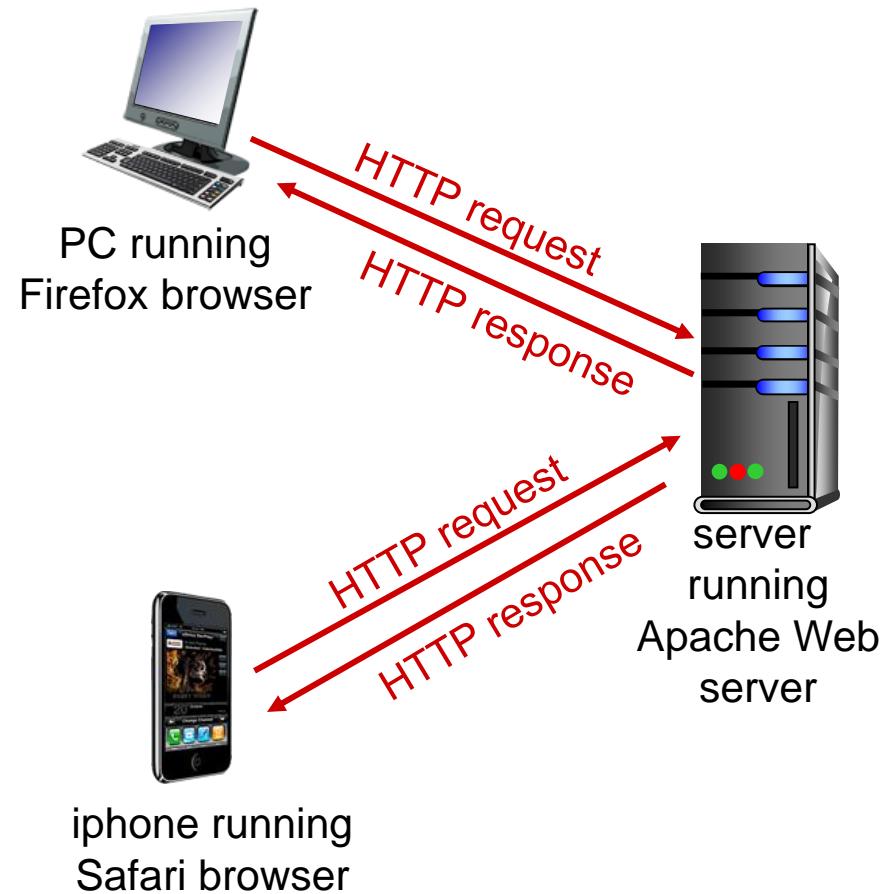
host name

path name

# HTTP overview

## HTTP: hypertext transfer protocol

- ❖ Web's application layer protocol
- ❖ client/server model
  - **client:** browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - **server:** Web server sends (using HTTP protocol) objects in response to requests



# HTTP overview (continued)

**uses TCP:**

- ❖ client initiates TCP connection (creates socket) to server, port 80
- ❖ server accepts TCP connection from client
- ❖ HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- ❖ TCP connection closed

**HTTP is “stateless”**

- ❖ server maintains no information about past client requests

**protocols that maintain “state” are complex!**

- ❖ past history (state) must be maintained
- ❖ if server/client crashes, their views of “state” may be inconsistent, must be reconciled

*aside*

# HTTP response status codes

- ❖ status code appears in 1st line in server-to-client response message.
- ❖ some sample codes:

## **200 OK**

- request succeeded, requested object later in this msg

## **301 Moved Permanently**

- requested object moved, new location specified later in this msg  
(Location:)

## **400 Bad Request**

- request msg not understood by server

## **404 Not Found**

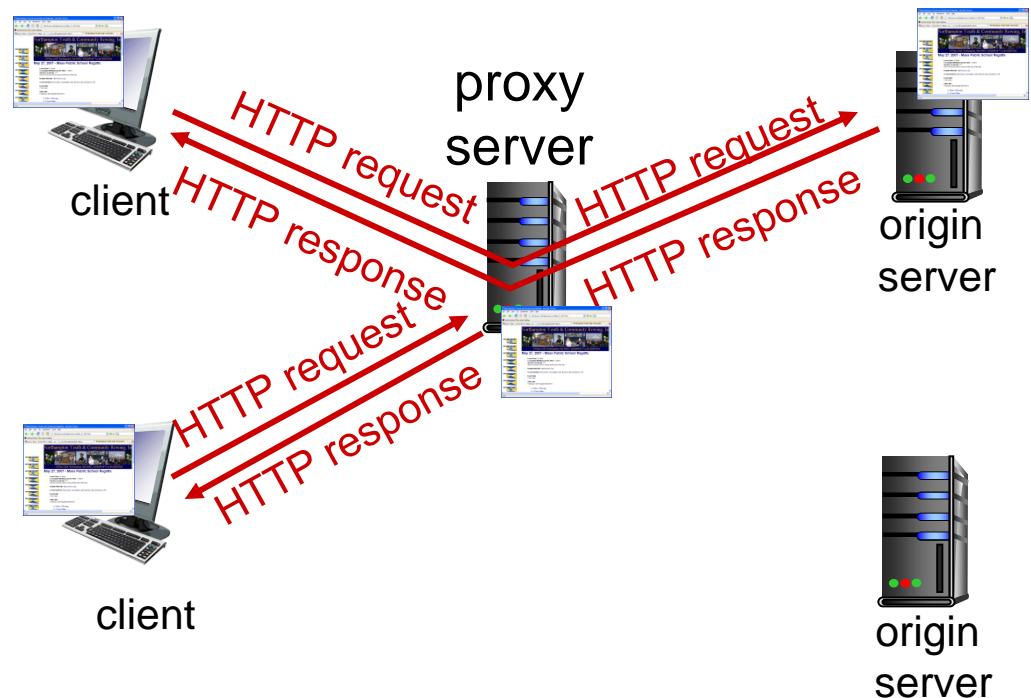
- requested document not found on this server

## **505 HTTP Version Not Supported**

# Web caches (proxy server)

**goal:** satisfy client request without involving origin server

- ❖ user sets browser: Web accesses via cache
- ❖ browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



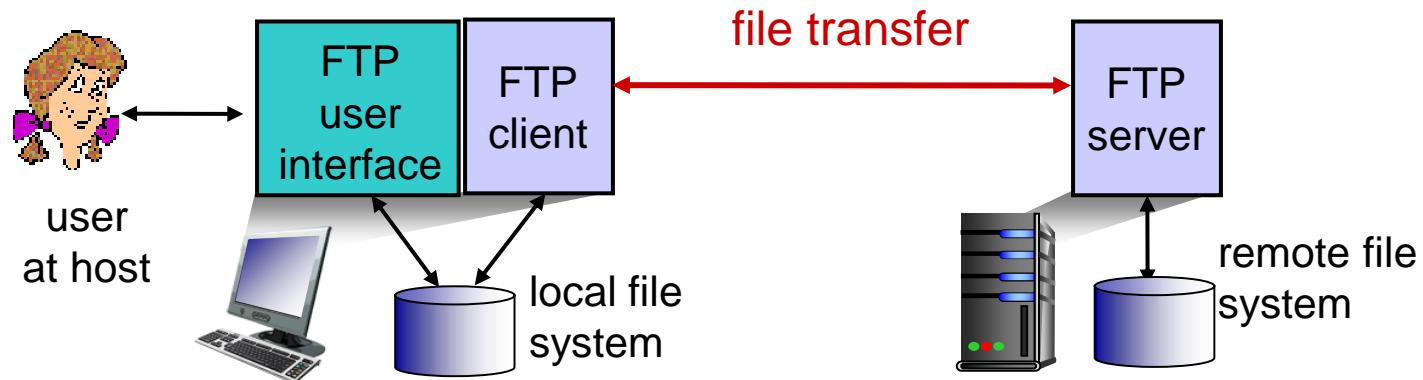
# More about Web caching

- ❖ cache acts as both client and server
  - server for original requesting client
  - client to origin server
- ❖ typically cache is installed by ISP (university, company, residential ISP)

## *why Web caching?*

- ❖ reduce response time for client request
- ❖ reduce traffic on an institution's access link
- ❖ Internet dense with caches: enables “poor” content providers to effectively deliver content (so too does P2P file sharing)

# FTP: the file transfer protocol



- ❖ transfer file to/from remote host
- ❖ client/server model
  - *client*: side that initiates transfer (either to/from remote)
  - *server*: remote host
- ❖ ftp server: port 21

# FTP: separate control, data connections

- ❖ FTP client contacts FTP server at port 21, using TCP
- ❖ client authorized over control connection
- ❖ client browses remote directory, sends commands over control connection
- ❖ when server receives file transfer command, **server** opens 2<sup>nd</sup> TCP data connection (for file) to client
- ❖ after transferring one file, server closes data connection

# Electronic mail

*Three major components:*

- ❖ user agents
- ❖ mail servers
- ❖ simple mail transfer protocol: SMTP

## *User Agent*

- ❖ a.k.a. “mail reader”
- ❖ composing, editing, reading mail messages
- ❖ e.g., Outlook, Thunderbird, iPhone mail client
- ❖ outgoing, incoming messages stored on server

# Electronic mail: mail servers

## mail servers:

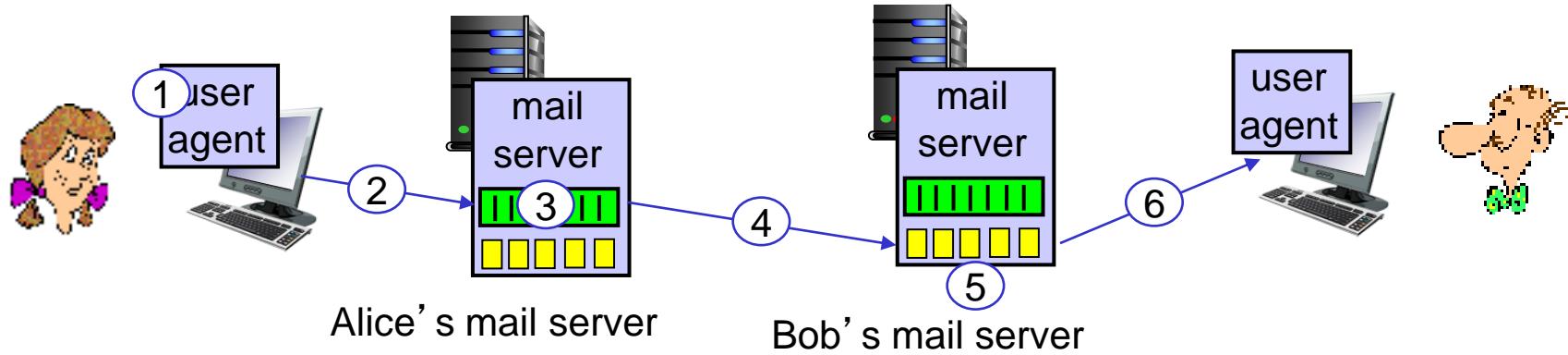
- ❖ *mailbox* contains incoming messages for user
- ❖ *message queue* of outgoing (to be sent) mail messages
- ❖ *SMTP protocol* between mail servers to send email messages
  - client: sending mail server
  - “server”: receiving mail server

# Electronic Mail: SMTP [RFC 2821]

- ❖ uses TCP to reliably transfer email message from client to server, port 25
- ❖ direct transfer: sending server to receiving server
- ❖ three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure

# Scenario: Alice sends message to Bob

- 1) Alice uses UA to compose message “to” bob@someschool.edu
- 2) Alice’s UA sends message to her mail server; message placed in message queue
- 3) client side of SMTP opens TCP connection with Bob’s mail server
- 4) SMTP client sends Alice’s message over the TCP connection
- 5) Bob’s mail server places the message in Bob’s mailbox
- 6) Bob invokes his user agent to read message



# Mail message format

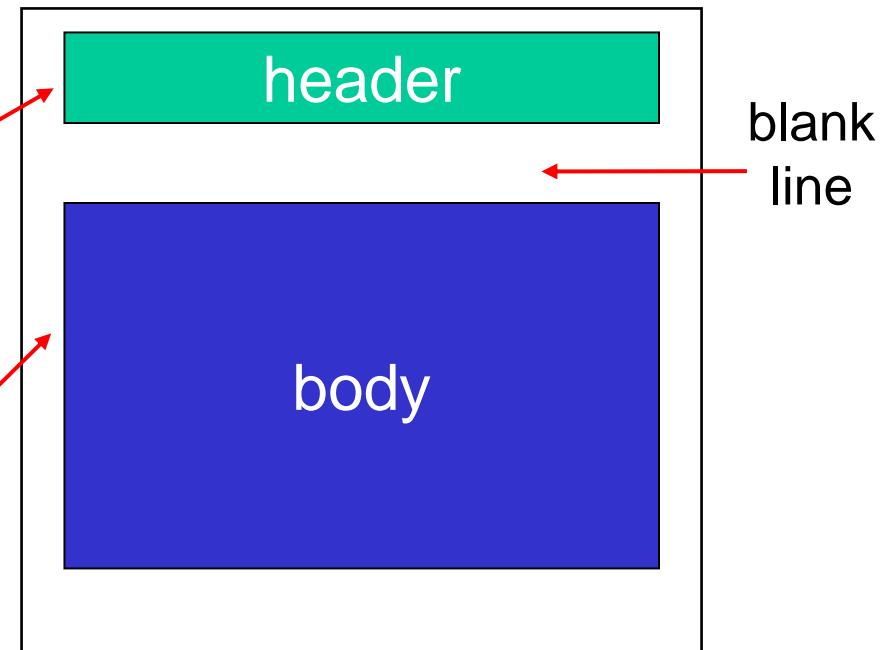
SMTP: protocol for  
exchanging email msgs

RFC 822: standard for text  
message format:

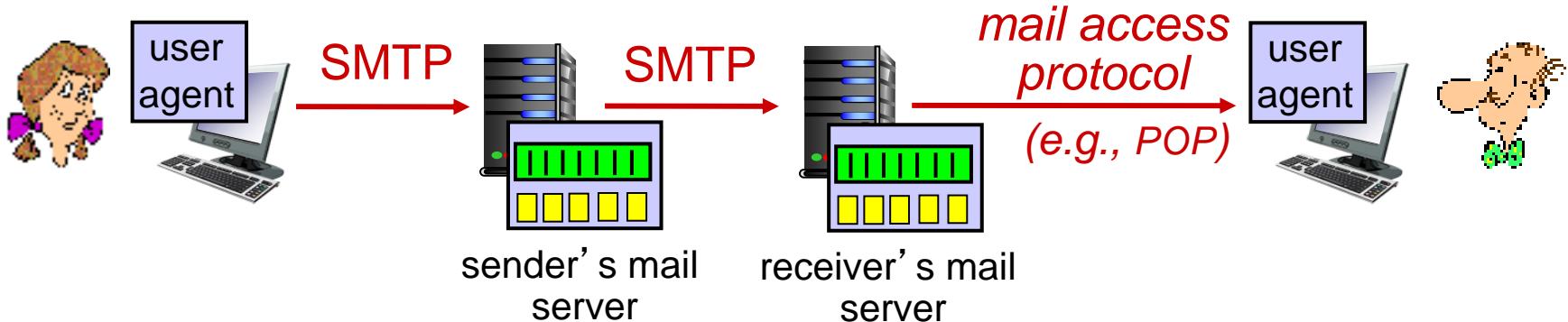
- ❖ header lines, e.g.,
  - To:
  - From:
  - Subject:

*different from SMTP MAIL  
FROM, RCPT TO:  
commands!*

- ❖ Body: the “message”
  - ASCII characters only



# Mail access protocols



- ❖ **SMTP:** delivery/storage to receiver's server
- ❖ mail access protocol: retrieval from server
  - **POP:** Post Office Protocol [RFC 1939]: authorization, download
  - **HTTP:** gmail, Hotmail, Yahoo! Mail, etc.

# DNS: domain name system

*people*: many identifiers:

- SSN, name, passport #

*Internet hosts, routers*:

- IP address (32 bit) - used for addressing datagrams
- “name”, e.g., [www.yahoo.com](http://www.yahoo.com) - used by humans

*Q*: how to map between IP address and name, and vice versa ?

## *Domain Name System:*

- ❖ *distributed database* implemented in hierarchy of many *name servers*
- ❖ *application-layer protocol*: hosts, name servers communicate to *resolve* names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network’s “edge”

# DNS: services, structure

## *DNS services*

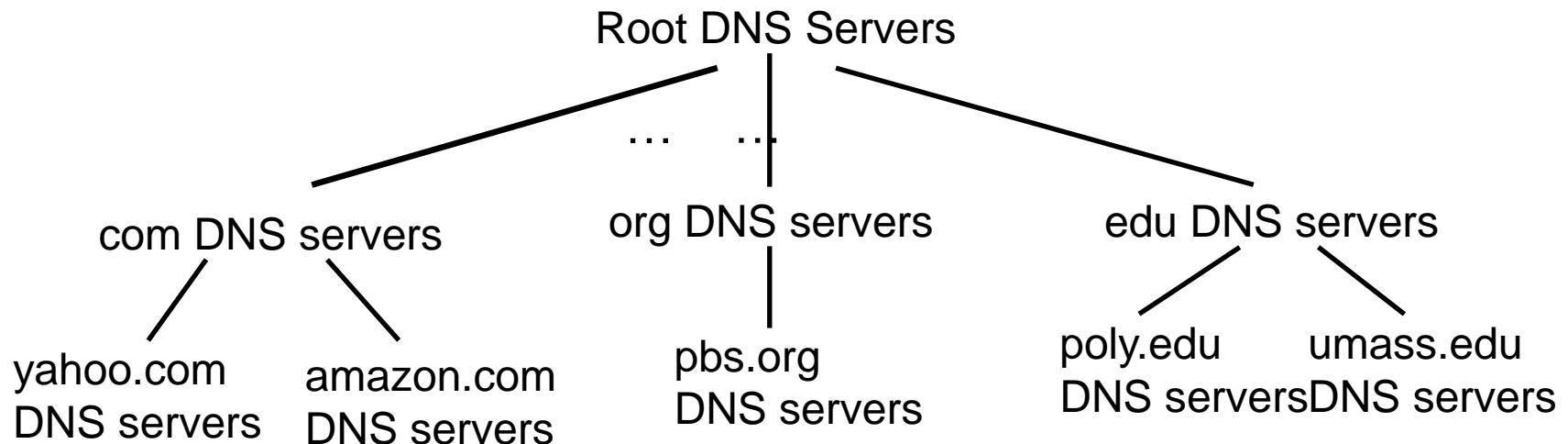
- ❖ hostname to IP address translation
- ❖ host aliasing
  - canonical, alias names
- ❖ mail server aliasing
- ❖ load distribution
  - replicated Web servers:  
many IP addresses  
correspond to one  
name

## *why not centralize DNS?*

- ❖ single point of failure
- ❖ traffic volume
- ❖ distant centralized database
- ❖ maintenance

*A: doesn't scale!*

# DNS: a distributed, hierarchical database

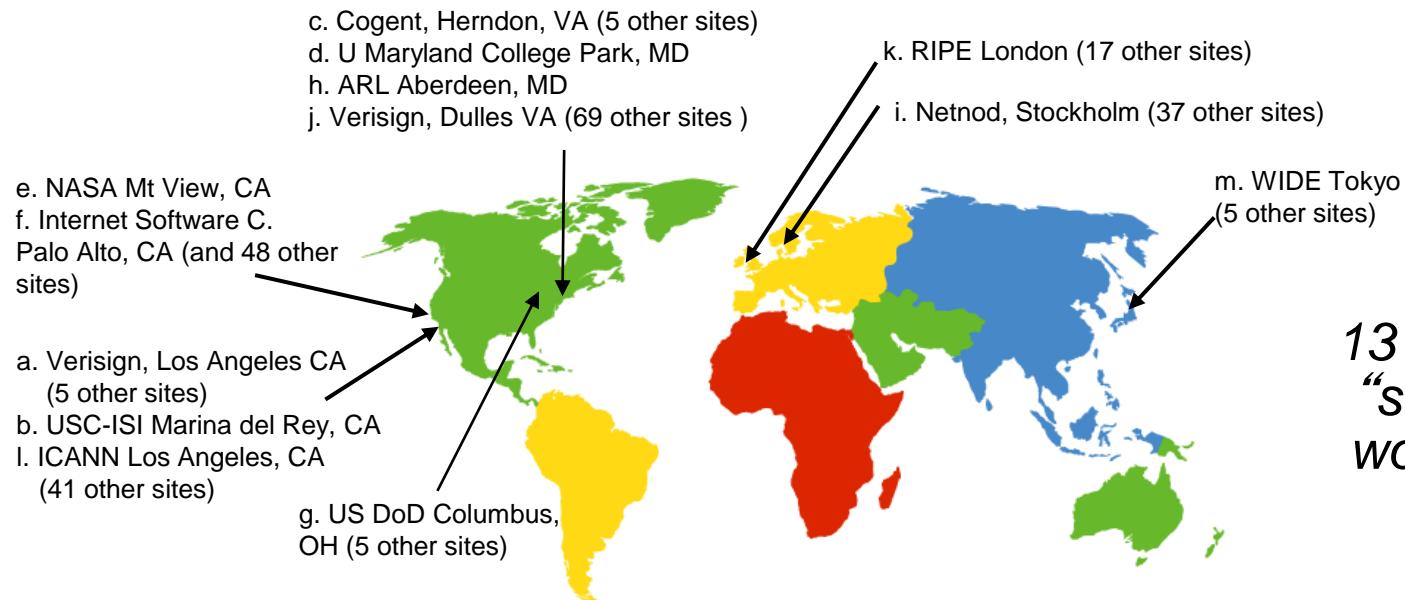


*client wants IP for www.amazon.com; 1<sup>st</sup> approx:*

- ❖ client queries root server to find com DNS server
- ❖ client queries .com DNS server to get amazon.com DNS server
- ❖ client queries amazon.com DNS server to get IP address for www.amazon.com

# DNS: root name servers

- ❖ contacted by local name server that can not resolve name
- ❖ root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server



*13 root name  
“servers”  
worldwide*

# TLD, authoritative servers

## *top-level domain (TLD) servers:*

- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
- Network Solutions maintains servers for .com TLD
- Educause for .edu TLD

## *authoritative DNS servers:*

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

# Local DNS name server

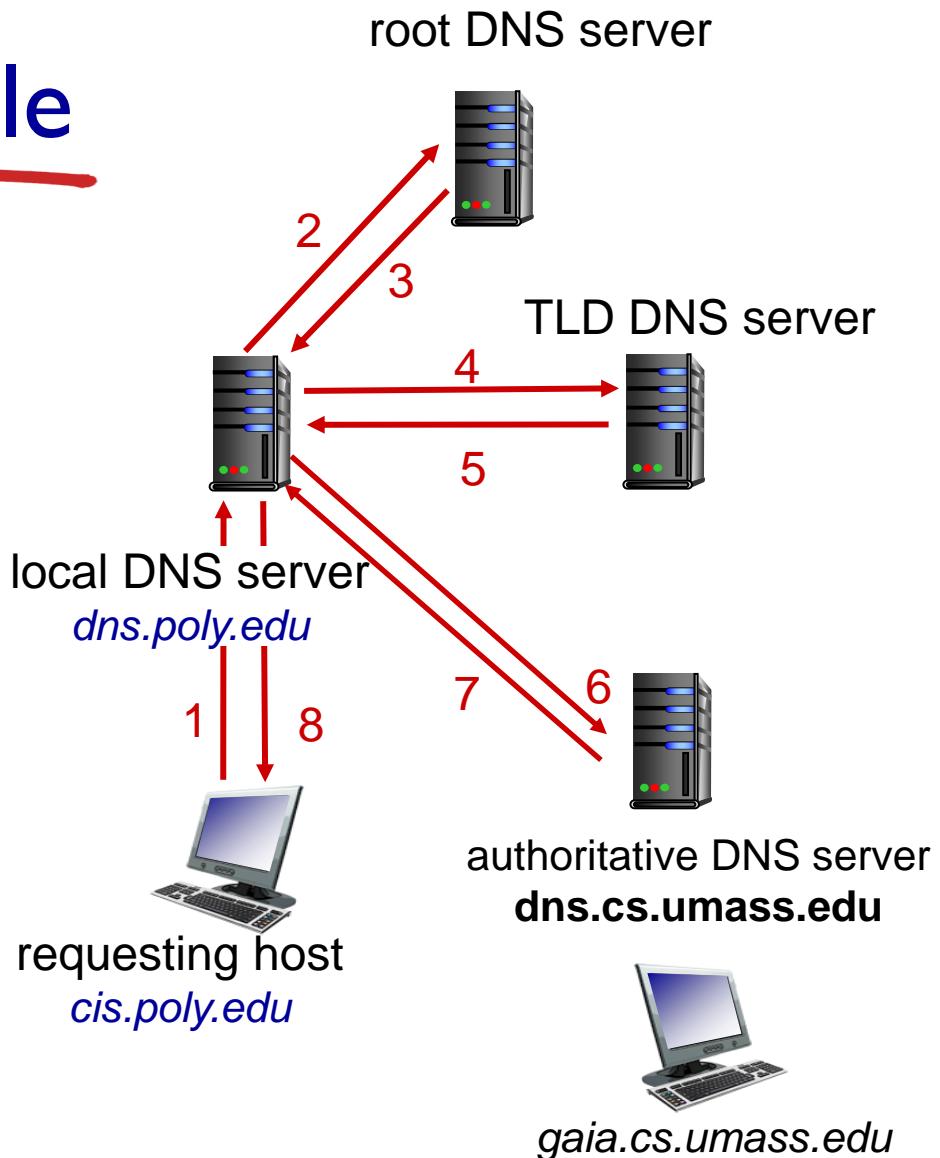
- ❖ does not strictly belong to hierarchy
- ❖ each ISP (residential ISP, company, university) has one
  - also called “default name server”
- ❖ when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

# DNS name resolution example

- ❖ host at `cis.poly.edu` wants IP address for `gaia.cs.umass.edu`

## *iterated query:*

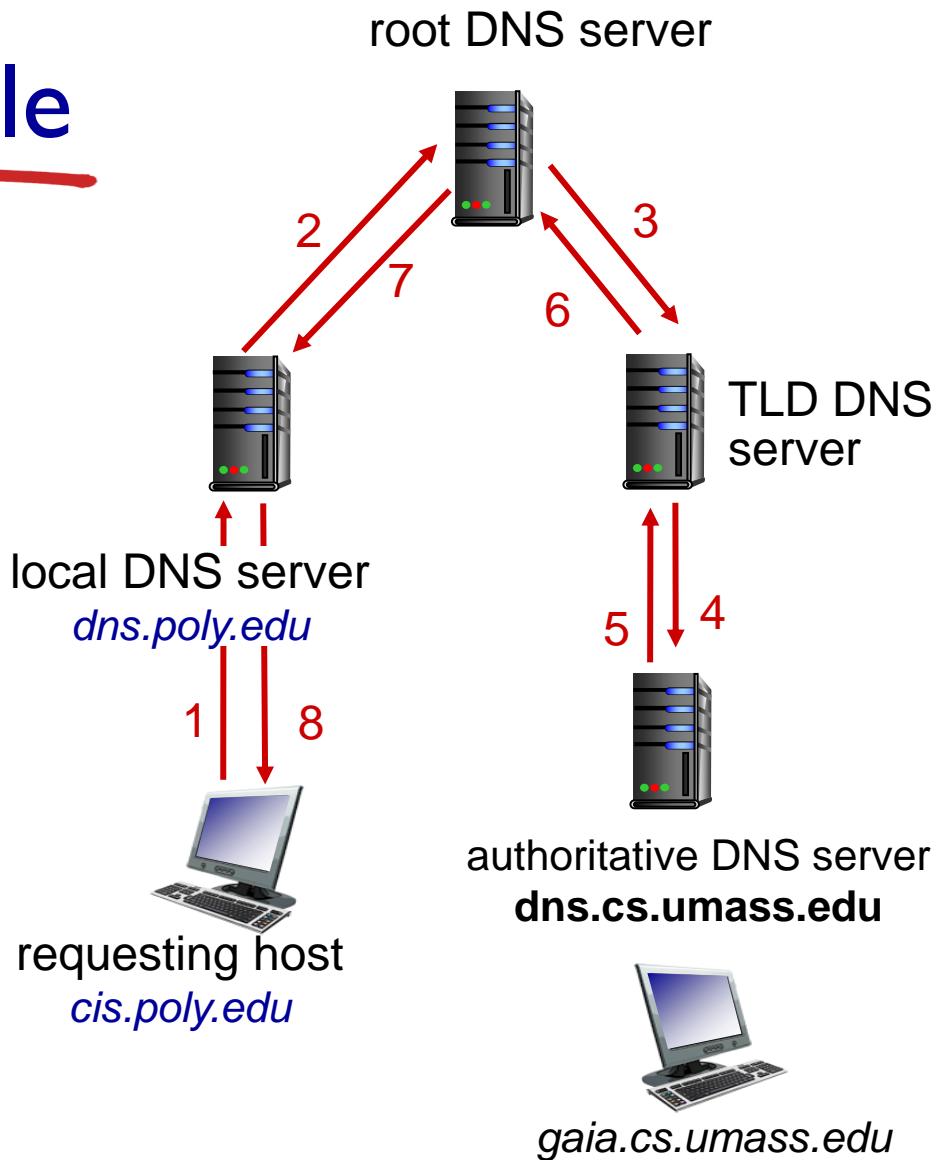
- ❖ contacted server replies with name of server to contact
- ❖ “I don’t know this name, but ask this server”



# DNS name resolution example

*recursive query:*

- ❖ puts burden of name resolution on contacted name server
- ❖ heavy load at upper levels of hierarchy?



# DNS: caching, updating records

- ❖ once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- ❖ cached entries may be *out-of-date* (best effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire

# Attacking DNS

## DDoS attacks

- ❖ Bombard root servers with traffic
  - Not successful to date
  - Traffic Filtering
  - Local DNS servers cache IPs of TLD servers, allowing root server bypass
- ❖ Bombard TLD servers
  - Potentially more dangerous

## Redirect attacks

- ❖ Man-in-middle
  - Intercept queries
- ❖ DNS poisoning
  - Send bogus replies to DNS server, which caches

## Exploit DNS for DDoS

- ❖ Send queries with spoofed source address: target IP
- ❖ Requires amplification