# Internet applications

# Recall Internet architecture

Intelligence at *end systems*

- e.g., web server software communicates with browser software

No need to write software for network-core devices when deploying new applications

- Network-core devices do not run user applications or tamper with packet payloads
- applications on end systems allows for rapid app development, propagation

# Application protocols

- Language spoken between a client application (i.e. web browser) and a server application (i.e. a web server)
- Describes how clients and servers communicate with each other
  - Types of messages (e.g., request & response)
  - Syntax of messages
  - Semantics of the fields
  - Rules for processing

# Application layer protocols

Types of application protocols

- Public-domain protocols
  - defined in RFCs from IETF
  - allows for interoperability
  - e.g., HTTP, SMTP
- Proprietary protocols
  - e.g., KaZaA, Skype

# Understanding application requirements

**Data loss**

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

**Timing**

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

**Bandwidth**

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

# Transport service requirements of common apps

| Application | Data loss | Bandwidth | Time Sensitive |
|---|---|---|---|
| file transfer | no loss | elastic | no |
| e-mail | no loss | elastic | no |
| Web documents | no loss | elastic | no |
| real-time audio/video | loss-tolerant | audio: 5kbps-1Mbps<br>video:10kbps-5Mbps | yes, 100's msec |
| streaming audio/video | loss-tolerant | same as above | yes, few secs |
| interactive games | loss-tolerant | few kbps up | yes, 100's msec |
| instant messaging | no loss | elastic | yes and no |

# Internet transport protocols services

## TCP service:

- *connection-oriented:* setup required between client and server processes
- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum bandwidth guarantees

## UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

# Internet apps: application, transport protocols

| Application | Application layer protocol | Underlying transport protocol |
|---|---|---|
| e-mail | SMTP [RFC 2821] | TCP |
| remote terminal access | Telnet [RFC 854] | TCP |
| Web | HTTP [RFC 2616] | TCP |
| file transfer | FTP [RFC 959] | TCP |
| streaming multimedia | proprietary (e.g. RealNetworks) | TCP or UDP |
| Internet telephony | proprietary (e.g., Vonage,Dialpad) | typically UDP |

# Web/HTTP

# Web and HTTP

First some jargon

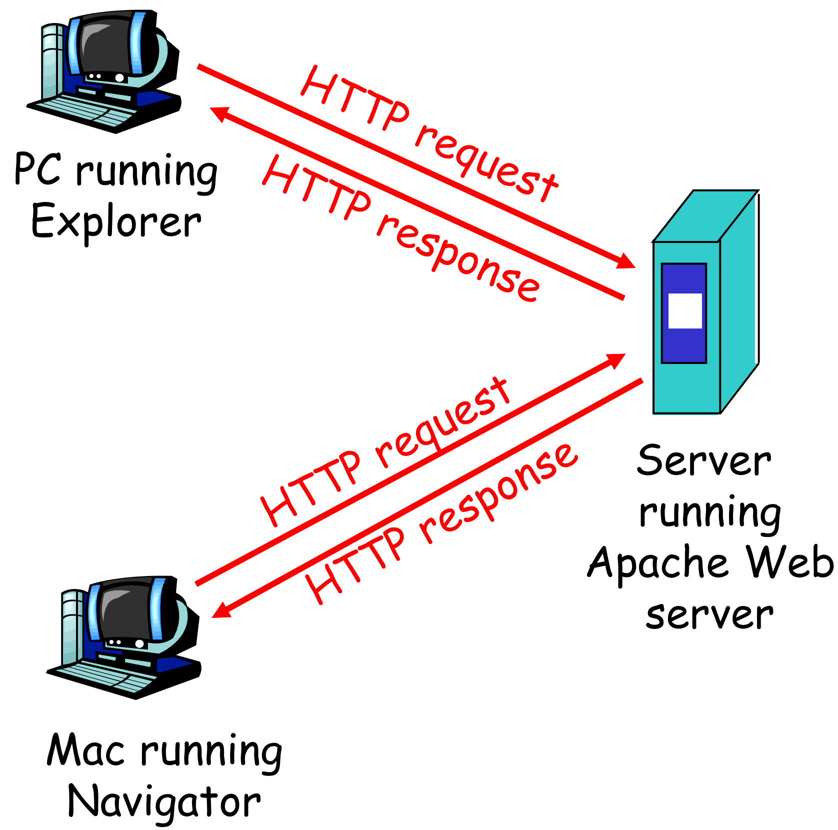
- Web page consists of objects
- Object can be HTML file, JPEG image, Java applet, audio file,…
- Each object is addressable by a URL
- Web page consists of base HTML-file which includes several referenced objects
- Example URL:

```
www.someschool.edu/someDept/pic.gif
```

host name — path name

# HTTP overview

- HTTP: hypertext transfer protocol
- Web's application layer protocol
- client/server model
- HTTP 1.0: RFC 1945
  - http://www.rfc-editor.org/rfc/rfc1945.txt
- HTTP 1.1: RFC 2068
  - http://www.rfc-editor.org/rfc/rfc2068.txt



# HTTP overview (continued)

Uses TCP:

- client initiates bi-directional TCP connection (via 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)
  - Messages encoded in text
- TCP connection closed

# HTTP request message

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)

request line (GET, POST, HEAD commands)

header lines

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

(extra carriage return, line feed)

Carriage return, line feed indicates end of message

`http://www.someschool.edu/somedir/page.html`

# HTTP response message

status line (protocol status code status phrase)

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

header lines

data, e.g., requested HTML file

# Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

Opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. Anything typed in sent to port 80 at cis.poly.edu

2. Type in a GET HTTP request:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. Look at response message sent by HTTP server!

# HTTP in action

- Other examples
  - http://www.thefengs.com/wuchang/work/courses/cs347u/http.txt
  - http://www.thefengs.com/wuchang/work/courses/cs347u/http_post.txt

# User-server state: cookies

HTTP initially "stateless"

- Didn't remember users or prior requests

Many major Web sites need state

- Yahoo mail
- Amazon shopping cart

HTTP state management (cookies): RFC 2109

- http://www.rfc-editor.org/rfc/rfc2109.txt

# User-server state: cookies

Four components:

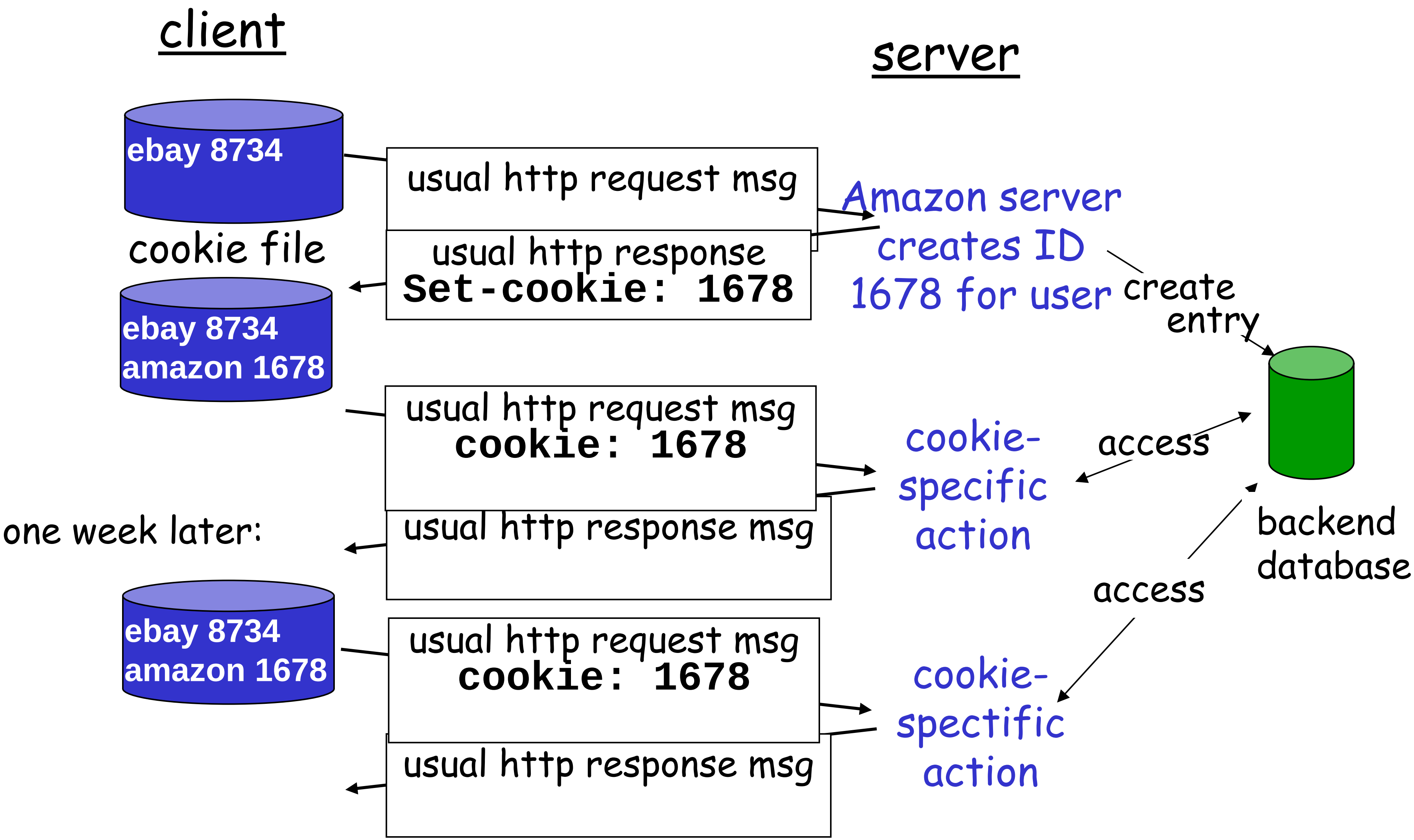
1) cookie header line of HTTP *response* message
`Set-cookie:`

2) cookie header line in HTTP *request* message
`Cookie:`

3) cookie file kept on user's host, managed by user's browser

4) back-end database at Web site

# Cookies: keeping "state" (cont.)



# Cookies (continued)

What cookies can bring:

- authorization
- shopping carts
- Site preferences
- recommendations
- user session state (Web e-mail)

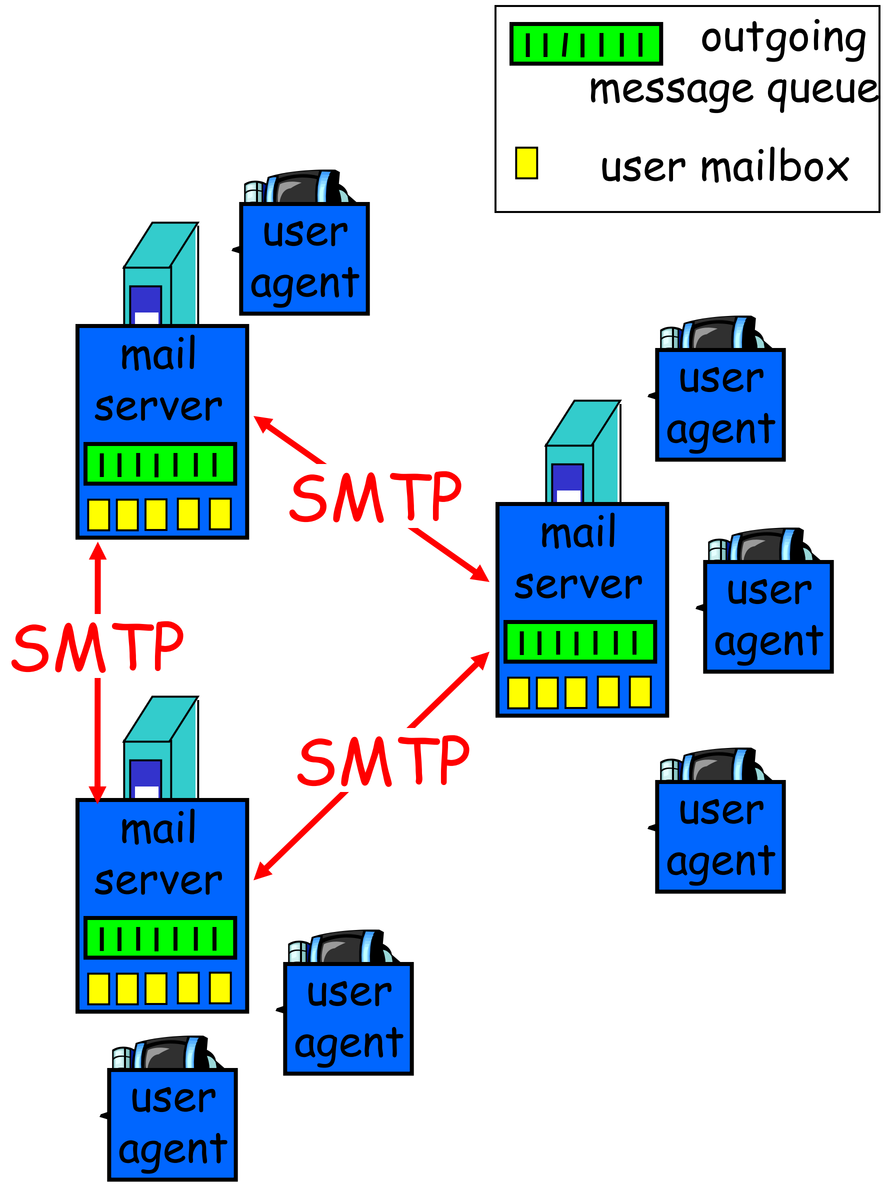
aside

Cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites
- search engines use redirection & cookies to learn yet more
- advertising companies obtain info across sites

# E-mail

# 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., Eudora, Outlook, elm, Mozilla Thunderbird
- outgoing, incoming messages stored on server

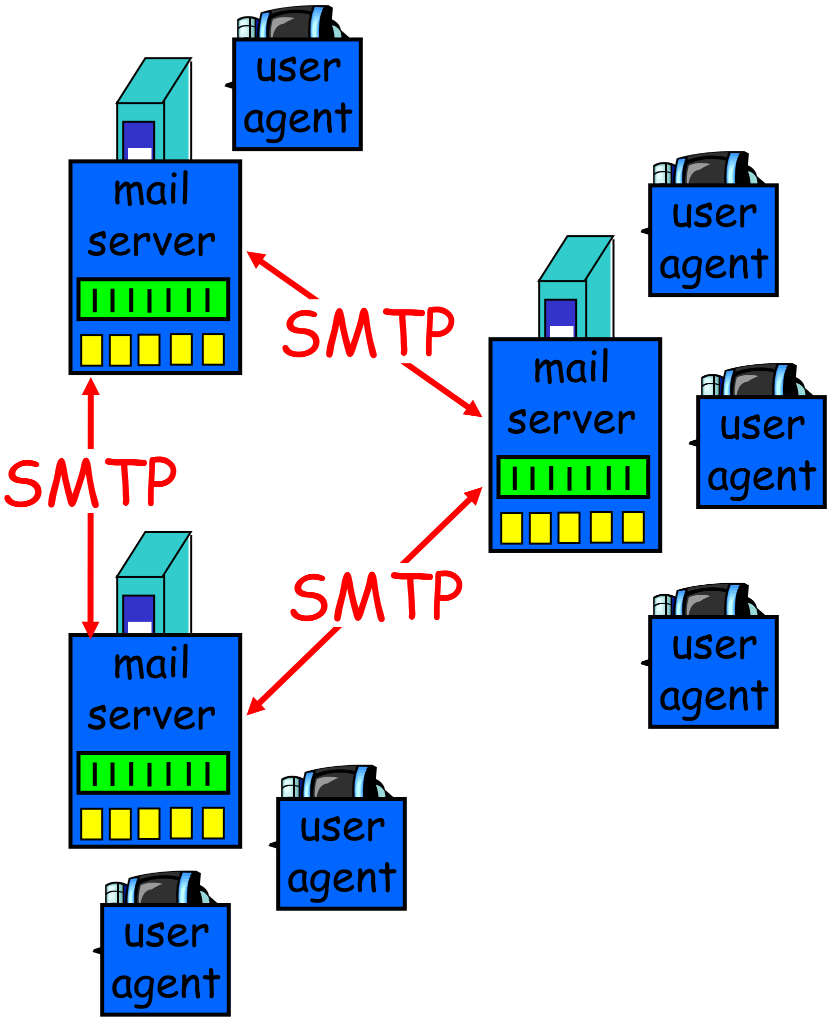
# Electronic Mail: mail servers

## Mail Servers

- mailbox contains incoming messages for user
- message queue of outgoing (to be sent) mail messages
- e.g. sendmail, postfix, Exchange

## SMTP protocol

- Between mail servers to send email messages
- Mail servers are both clients and servers



# Electronic Mail: SMTP [RFC 821]

- uses TCP to reliably transfer email message from client to server, port 25
  - User agent to sending server (sometimes)
  - Sending server to receiving server (always)
- command/response interaction
  - commands:
  - response: status code and phrase

# Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

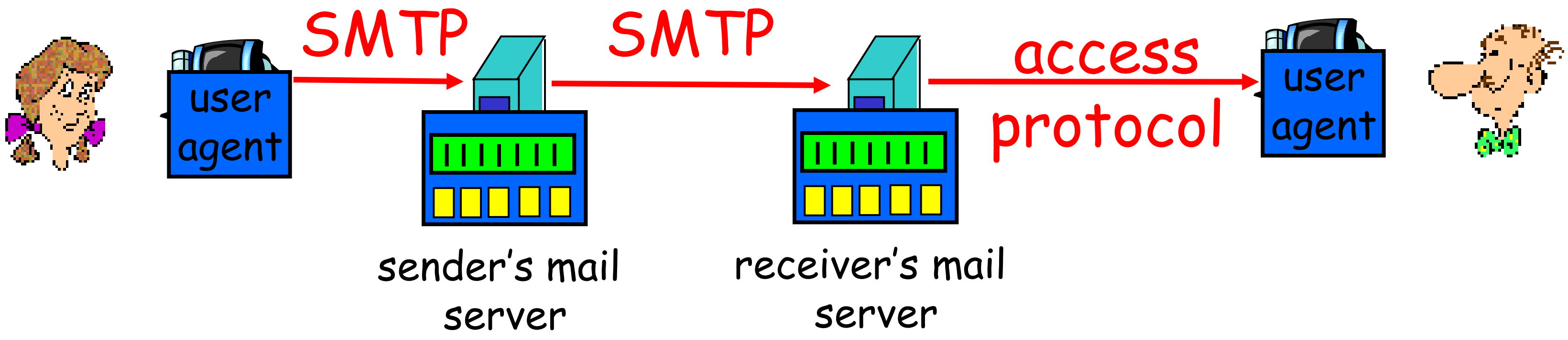
Q: Client fills in the “From:” field.  Is this a problem?

# SMTP

Comparison with HTTP:

- HTTP: pull
- SMTP: push
  - Some argue this should have been a pull as well due to spam
- Both have ASCII command/response interaction, status codes

# Mail access protocols




- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - Direct (telnet or ssh followed by "mail")
  - POP: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) and download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - more features (more complex)
    - manipulation of stored msgs on server
  - HTTP: Hotmail , Yahoo! Mail, Horde/IMP, etc.

# POP3 protocol

**authorization phase**

- client commands:
  - **user**: declare username
  - **pass**: password
- server responses
  - **+OK**
  - **-ERR**

**transaction phase**

- **list**: list message numbers
- **retr**: retrieve message by number
- **dele**: delete
- **quit**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

# DNS

# Domain Name System (DNS)

- Internet hosts, routers like to use fixed-length addresses (numbers)
  - IP address (32 bit) - used for addressing datagrams
- Humans like to use variable-length names
  - www.cs.pdx.edu
  - keywords
- DNS, keywords, naming protocols
  - Map names to numbers (IP addresses)

# Original Name to Address Mapping

- Flat namespace
  - /etc/hosts.txt
  - SRI kept main copy
  - Downloaded regularly
- Problems
  - Count of hosts was increasing
    - From machine per domain to machine per user
    - Many more downloads of hosts.txt
    - Many more updates of hosts.txt

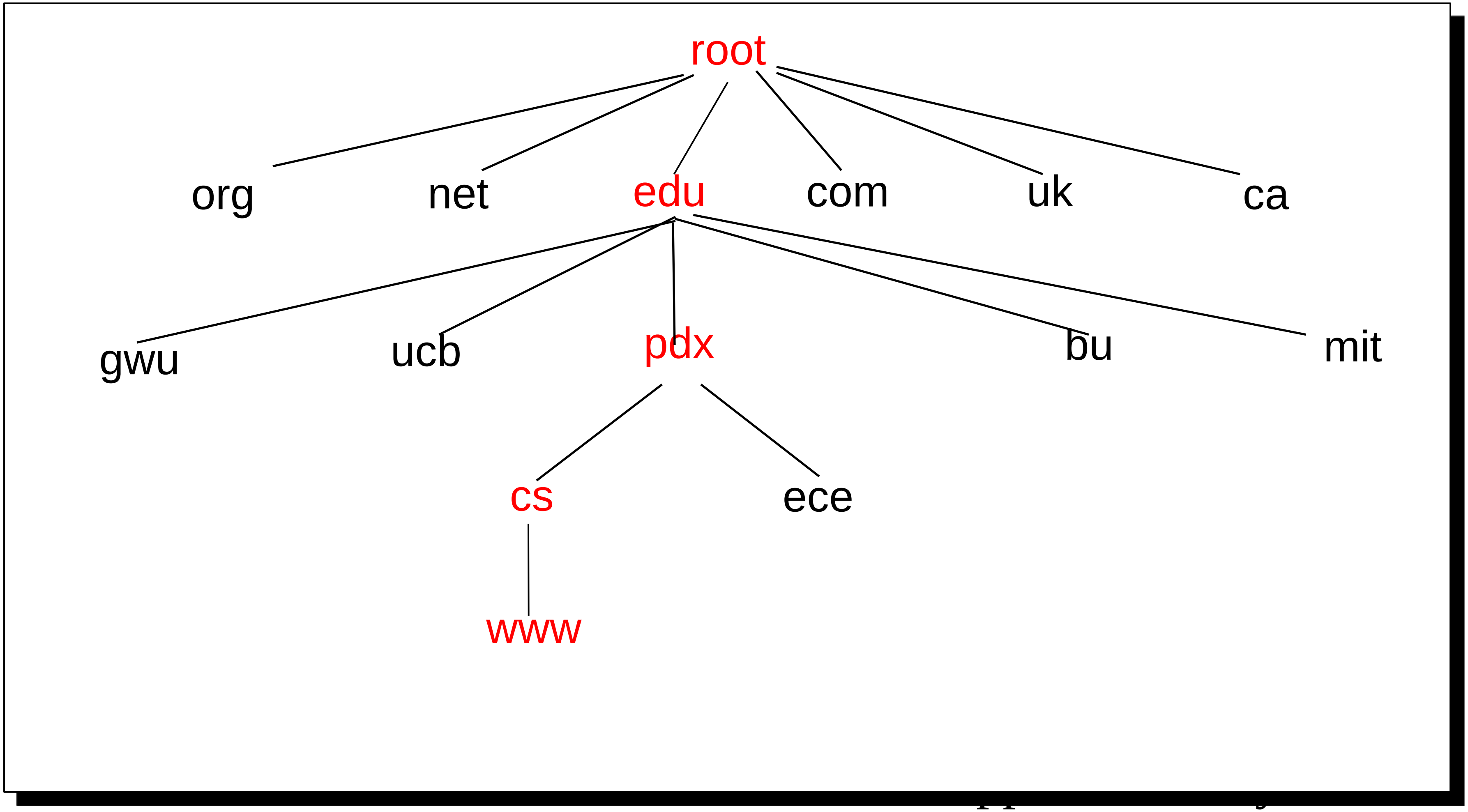
# DNS: Domain Name System (1984)

- *Distributed database* implemented as a hierarchy of many *name servers*
  - Goals
    - Scalability
    - Decentralized maintenance
    - Fault-tolerance
    - Global scope
      - Names mean the same thing everywhere
  - Why not centralize DNS?
    - Not scalable, hard to maintain, single point of failure
  - http://www.rfc-editor.org/rfc/rfc1034.txt
  - http://www.rfc-editor.org/rfc/rfc1035.txt

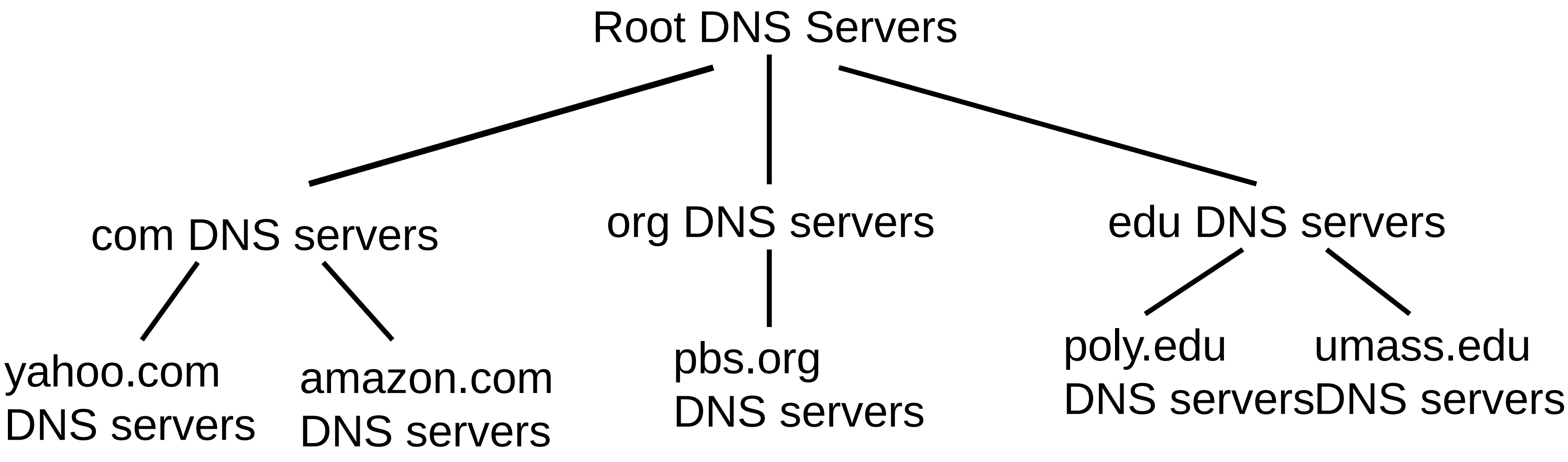
# DNS: Domain Name System (1984)

- *Application-layer* protocol used by hosts and name servers
  - communicate to *resolve* names (address/name translation)
  - core Internet function, implemented as application-layer protocol
    - complexity at network's "edge"
    - compare to phone network
      - naming (none supported)
      - addressing (complex mechanism within network)

# DNS hierarchical canonical name space



# Namespace maps closely to name servers



# What is stored at these servers?

DNS: distributed db storing resource records (RR)

RR format: `(name, value, type, ttl)`

- Type=A
  - **name** is hostname
  - **value** is IP address
- Type=NS
  - **name** is domain (e.g. foo.com)
  - **value** is hostname of authoritative name server for this domain
- Type=MX
  - **value** is name of mailserver associated with **name**

# Main parts of DNS

- Client resolver
- Local DNS servers
- Root servers
- TLD servers
- Authoritative servers

# Client resolver

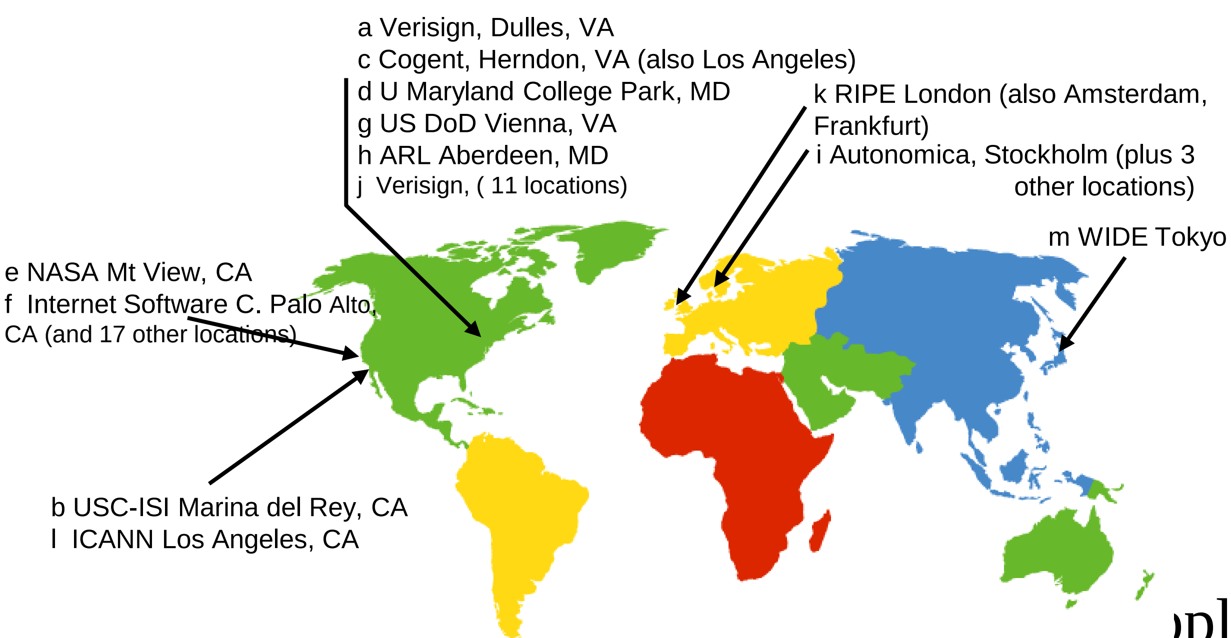
- Code on client to query DNS hierarchy
  - gethostbyname()

# Local Name Server

- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one.
  - Also called "default name server"
  - Specified in /etc/resolv.conf or given by DHCP
- Host's DNS queries sent to local DNS server
  - Acts as a proxy, forwards query into hierarchy.
  - Typically answer queries about local zone directly
  - Do a lookup of distant host names for local hosts
- Each local DNS server points to root servers
  - Hard-coded IP addresses in all name server distributions
  - Currently {a-m}.root-servers.net

# Root name servers

- Contacted by local name server that can not resolve name
- Typically returns information on next level of hierarchy (TLD server) for local name server to query
  - 13 root name servers worldwide for fault-tolerance
    - http://www.root-servers.org



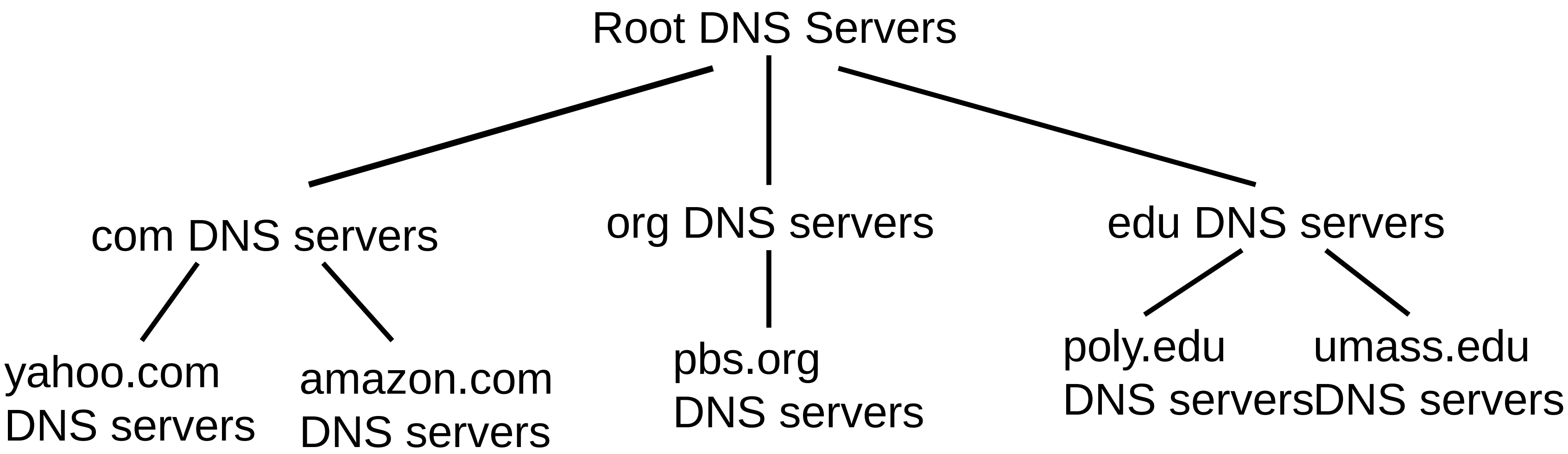
# TLD Servers

- Top-level domain (TLD) servers: responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
  - Network Solutions maintains servers for com TLD
  - Educause for edu TLD
  - Pass back information on next level of hierarchy (e.g. authoritative servers)

# Authoritative Servers

- Provides authoritative hostname to IP mappings
  - Typically, one per organization
  - Hand mappings out for organization's servers (Web & mail).
- Store parts of the database
  - Each part of a name is assigned to an authoritative server
  - Server responds to all queries for name it is the authority
  - Can be maintained by organization or service provider
  - Example
    - Authority for .edu is a root server
    - Authority for pdx.edu is the ".edu" TLD server
    - Authority for www.pdx.edu is dns0.pdx.edu (131.252.120.128)

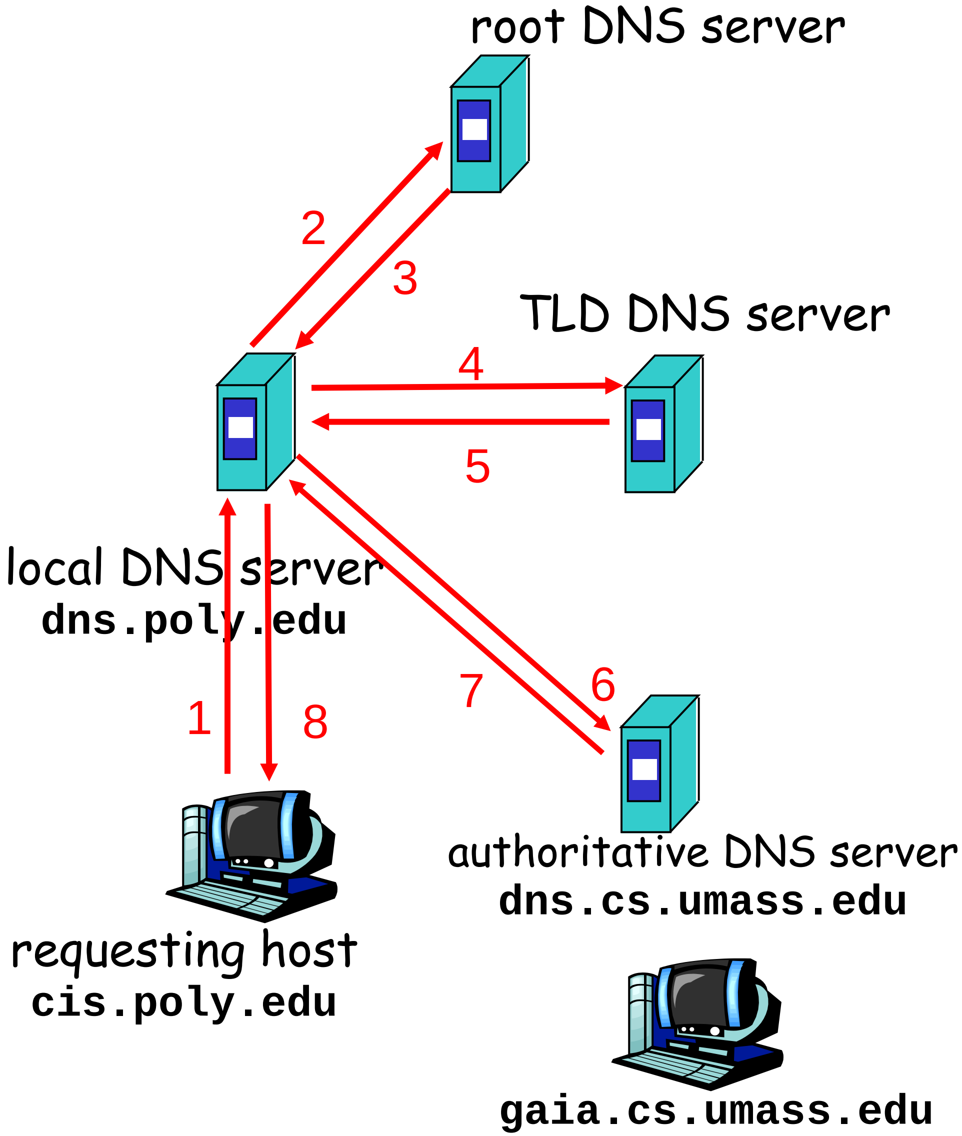
# DNS in action




Client wants IP for www.amazon.com

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

# DNS query example




- Host at cis.poly.edu wants IP address for gaia.cs.umass.edu

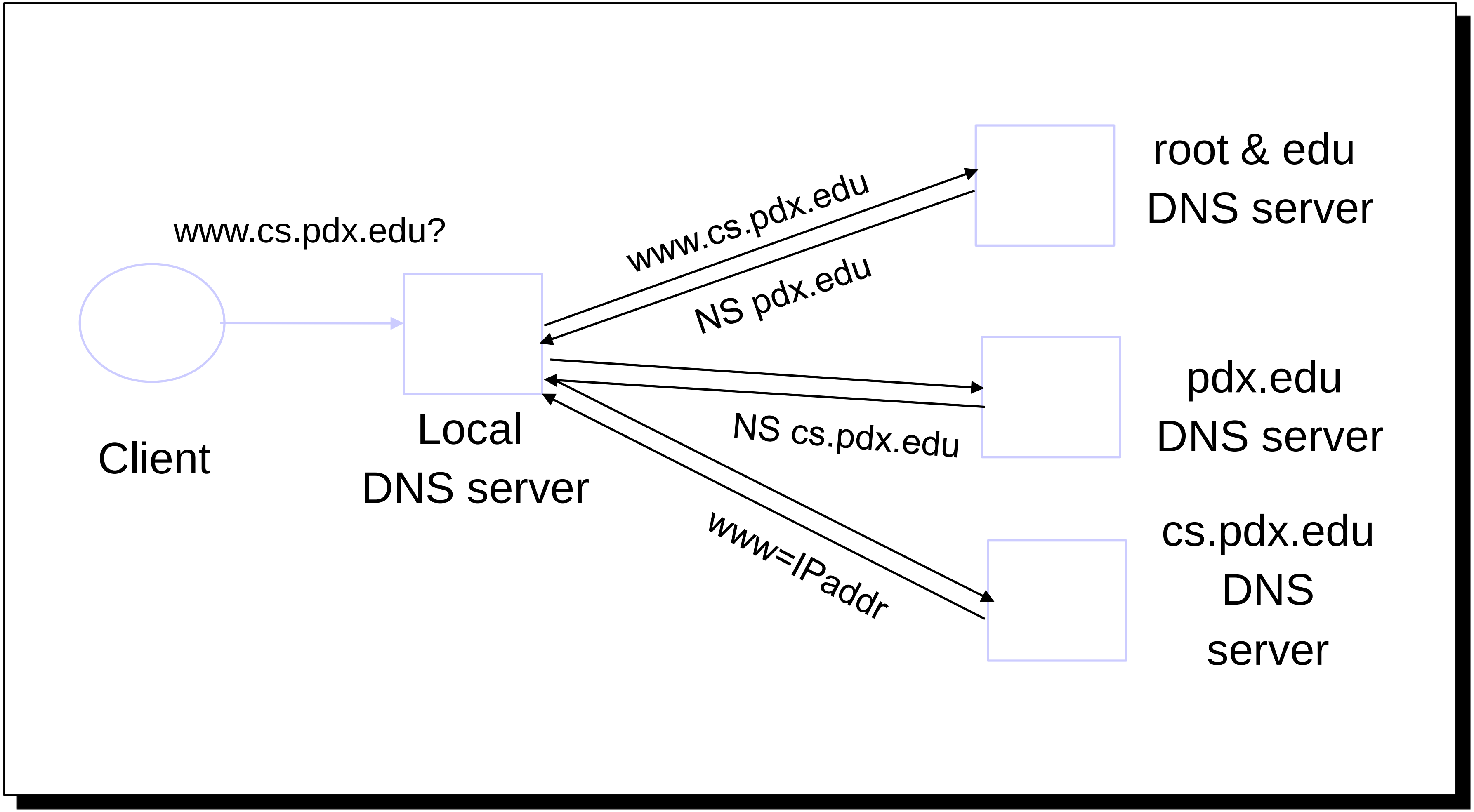
# Typical Resolution

- Client does recursive request to local name server
- Local name server does iterative requests for name
- Steps for resolving A record of pdx.edu
  - Application calls gethostbyname()
  - Resolver contacts local name server ($S_1$)
  - $S_1$ queries root server ($S_2$) for (.edu)
  - $S_2$ returns NS record for .edu TLD ($S_3$)
  - $S_1$ queries $S_3$ for pdx.edu
  - $S_3$ returns NS record for pdx.edu ($S_4$)
  - $S_1$ queries $S_4$ for pdx.edu
  - $S_4$ returns A record for pdx.edu
    - Can return multiple addresses -> what does this mean?

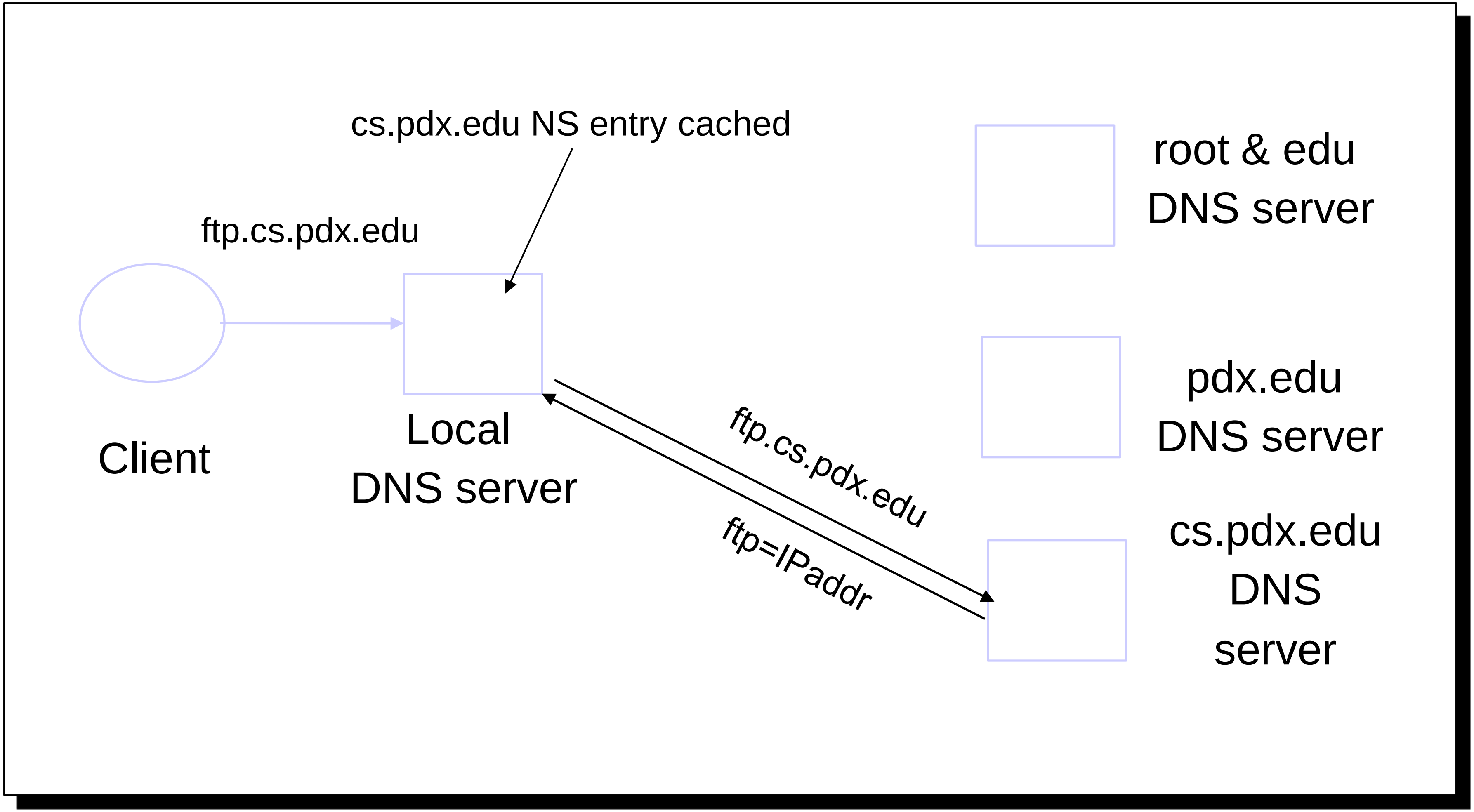
# DNS: caching and updating records

- DNS responses *cached* throughout hierarchy
  - Other queries may reuse some parts of lookup
    - NS records for domains reused often (xxx.yahoo.com)
  - Entries timeout after some time (soft state)
    - TTL field controlled by authority
    - Affects DNS-based load balancing
  - TLD servers often cached in local name servers
    - Thus, root name servers not often visited
  - Negative responses also cached
    - Don't repeat past mistakes (misspellings)
- update/notify mechanisms
  - RFC 2136
  - http://www.ietf.org/

# DNS Lookup Caching Example



# Subsequent Lookup Example



# DNS tools

- dig and nslookup
  - Can query specific DNS servers
  - Can query different resource record types

```
cat /etc/resolv.conf                            # local DNS server
dig +norecurse www.thefengs.com.                # do an iterative query to local DNS server
dig                                             # List root servers
dig @192.5.5.241 +norecurse www.thefengs.com.   # do an iterative query to IP addr of F root
dig @192.41.162.30 +norecurse www.thefengs.com. # do an iterative query to IP addr of L TLD
dig @216.21.236.249 +norecurse www.thefengs.com.  # do an iterative query to IP addr of NS at register.com
dig +norecurse www.thefengs.com.                # do an iterative query again to local DNS server
                                                #    NOTHING was cached at local DNS server!
dig +recurse www.thefengs.com.                  # now do a recursive query through local DNS server
dig +norecurse www.thefengs.com.                # now you get a cached result

                                                # Negative results also cached
dig +norecurse www.jjkkllmmnnoopp.com.          # returns pointer to root name servers
dig +recurse www.jjkkllmmnnoopp.com.            # returns status: NXDOMAIN
dig +norecurse www.jjkkllmmnnoopp.com.          # returns status: NXDOMAIN
```

# Creating your own site

- Example: just created startup "Network Utopia"
- Register name networkuptopia.com at a registrar (e.g., Network Solutions)
  - Give registrar names and IP addresses of your authoritative name server
  - Registrar inserts two RRs into the com TLD server:

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

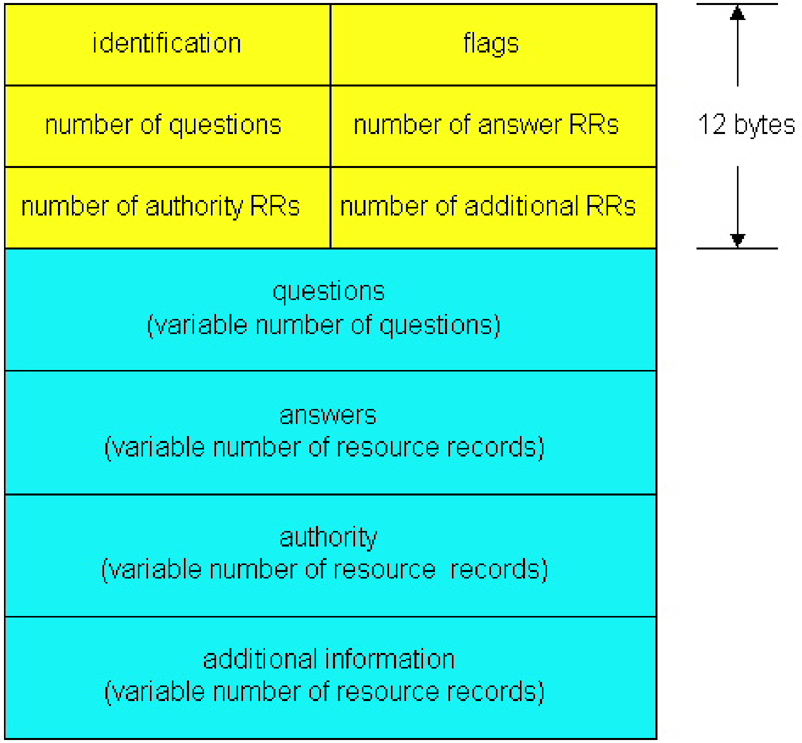
- Set up authoritative server (212.212.212.1)
  - Install DNS server (BIND)
  - Enter A record for www.networkuptopia.com
  - Enter MX record for networkutopia.com

# DNS protocol, messages

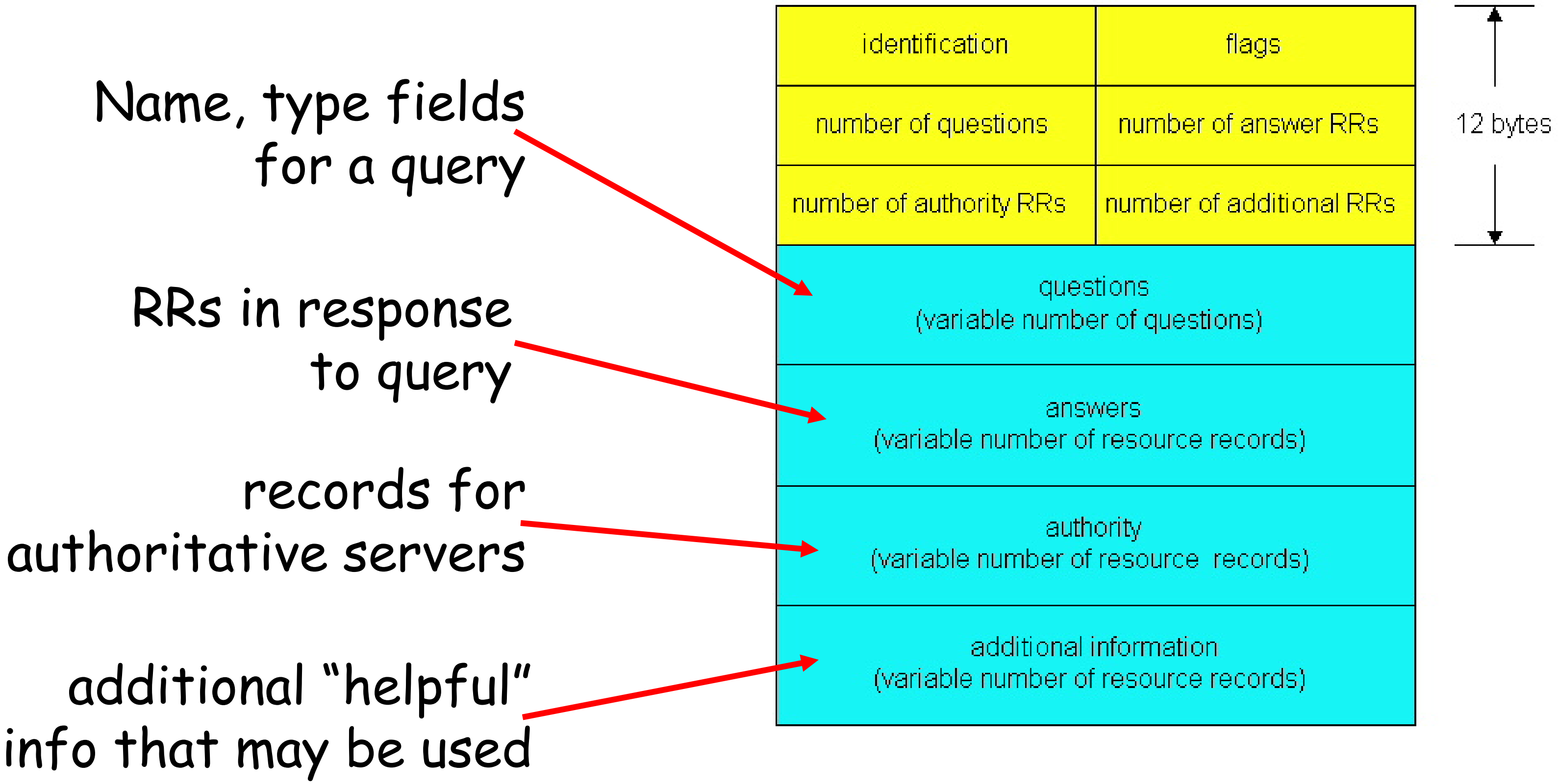
DNS protocol : *query* and *reply* messages, both with same *message format*

msg header

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



# DNS protocol, messages



# DNS issues

- UDP used for queries
  - Need reliability -> Why not TCP?
  - No rate control
- Centralized caching per site not required
- Vulnerability of 13 static root servers
  - Attacks on root servers have occurred
  - Jon Postel and his mobility "experiment"
- Spoofing identity
  - Adversary on the same network returning a bogus answer