Application Layer

Goals:

- Conceptual aspects of network application protocols
  - Client server paradigm
  - Service models
- Learn about protocols by examining popular application-level protocols
  - HTTP
  - DNS
  - SMTP, POP3, IMAP
  - FTP
  - Gnutella und KaZaa
  - IRC

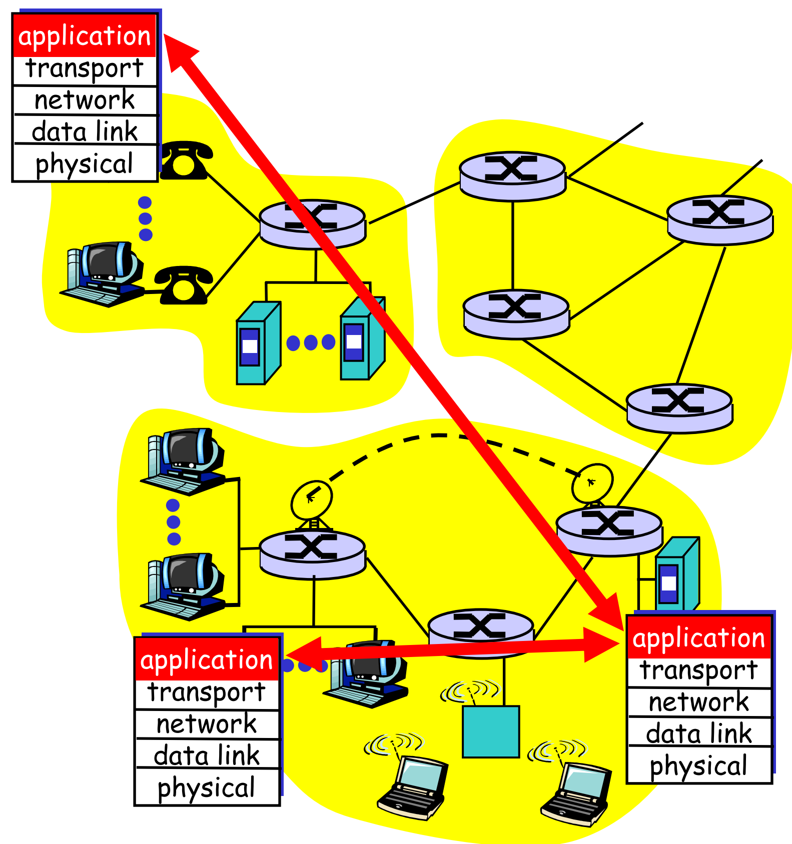
Applications and application-layer protocols

Application: communicating, distributed processes

- Running in network hosts in "user space"
- Exchange messages to implement app
- E.g., email, file transfer, the Web

Application-layer protocols

- One "piece" of an app
- Define messages exchanged by apps and actions taken
- User services provided by lower layer protocols



Client-server paradigm

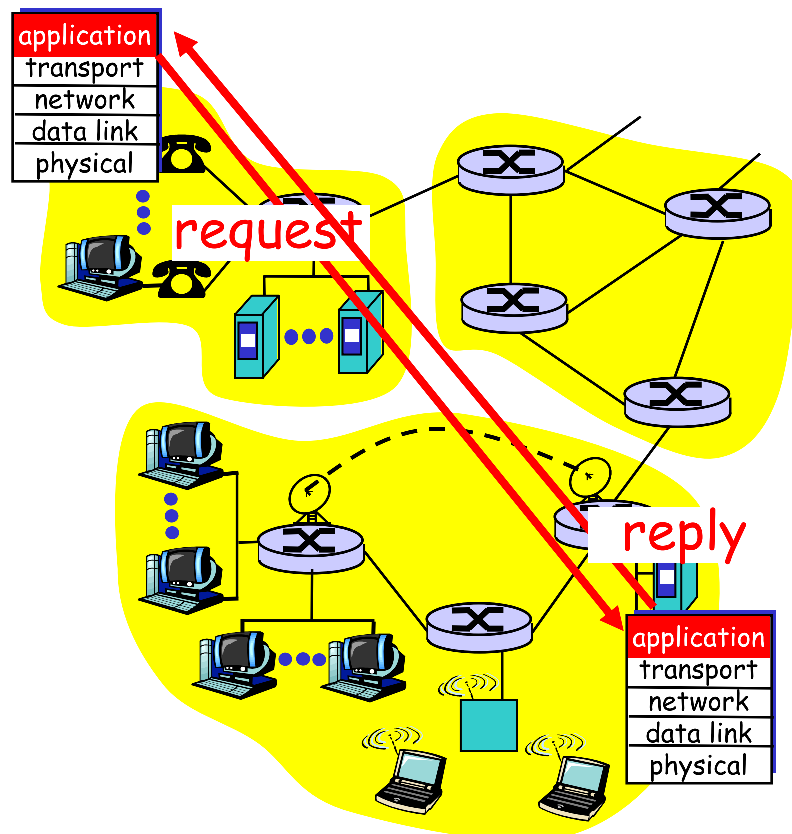
Typical network app has two pieces: *client* and *server*

Client:

- Initiates contact with server ("speaks first")
- Typically requests service from server,
- E.g., request WWW page, send email

Server:

- Provides requested service to client
- E.g., sends requested WWW page, receives/stores received email



Services provided by Internet transport protocols

TCP service:

- *Connection-oriented:* setup required between client, server
- *Reliable transport* between sending and receiving process
- *Flow control:* sender won't overwhelm receiver
- *Congestion control:* throttle sender when network overloaded
- *Does not providing:* timing, minimum bandwidth guarantees

UDP service:

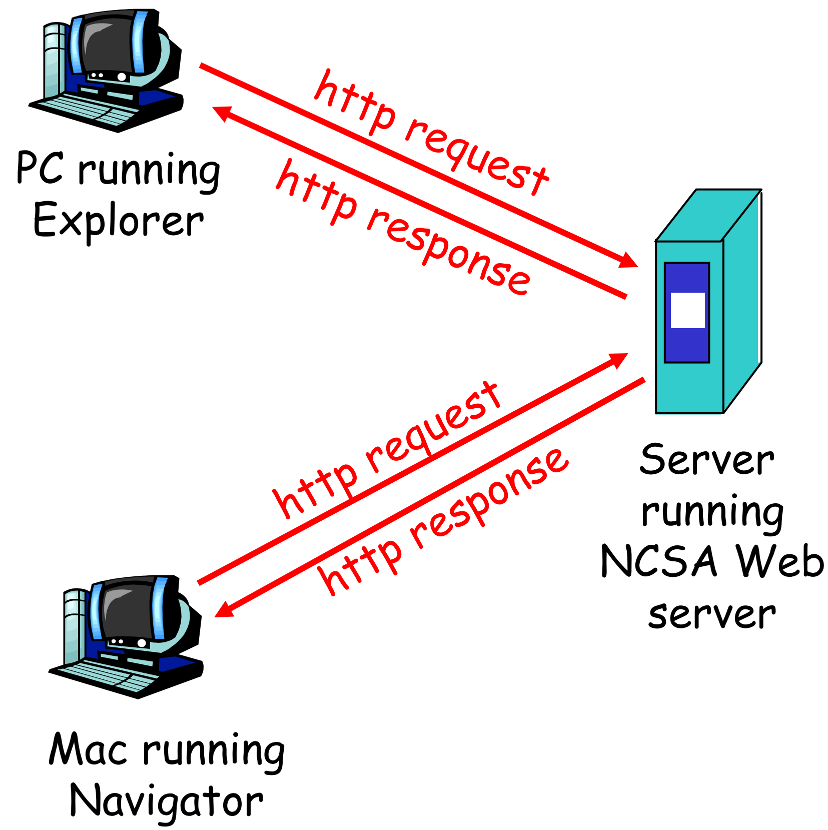
- Unreliable data transfer between sending and receiving process
- Does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

Q: Why bother? Why is there a UDP?

WWW: the HTTP protocol

HTTP: hypertext transfer protocol

- WWW's application layer protocol
- Client/server model
  - *Client:* browser that requests, receives, "displays" WWW objects
  - *Server:* WWW server sends objects in response to requests



HTTP - timeline

- Mar 1990 CERN labs document proposing Web
- Jan 1992 HTTP/0.9 specification
- Dec 1992 Proposal to add MIME to HTTP
- Feb 1993 UDI (Universal Document Identifier) Network
- Mar 1993 HTTP/1.0 first draft
- Jun 1993 HTML (1.0 Specification)
- Oct 1993 URL specification
- Nov 1993 HTTP/1.0 second draft
- Mar 1994 URI in WWW
- May 1996 HTTP/1.0 Informational, RFC 1945
- Jan 1997 HTTP/1.1 Proposed Standard, RFC 2068
- Jun 1999 HTTP/1.1 Draft Standard, RFC 2616
- 2001 HTTP/1.1 Formal Standard

The HTTP protocol: more

HTTP: TCP transport service:

- 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 WWW server (http server)
- TCP connection closed

HTTP is "stateless"

- Server maintains no information about past client requests

aside

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

http message format: request

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

request line (GET, POST, HEAD commands)

```
GET /somedir/page.html HTTP/1.1
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr

(extra carriage return, line feed)
```

header lines

Carriage return, line feed indicates end of message

http message format: reply

status line
(protocol
status code
status phrase)

header
lines

data, e.g.,
requested
html file

```
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 ...
```

http reply status codes

In first line in server → client response message.

A few sample codes:

200 OK

- request succeeded, requested object later in this message

301 Moved Permanently

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

400 Bad Request

- request message not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

HTTP request methods

- Properties:
  - Safe: examines the state of a resource
  - Idempotent: side effects of one request == those of multiple requests
- Methods
  - GET (safe, idempotent)
  - HEAD
  - POST (not safe, not idempotent)
  - PUT (not safe, idempotent)
  - Delete

The HTTP protocol: even more

- Non-persistent connection:
  One object in each TCP connection
  - Some browsers create multiple TCP connections *simultaneously* – one per object
- Persistent connection:
  Multiple objects transferred within one TCP connection
- Pipelined persistent connections:
  Multiple requests issued without waiting for response

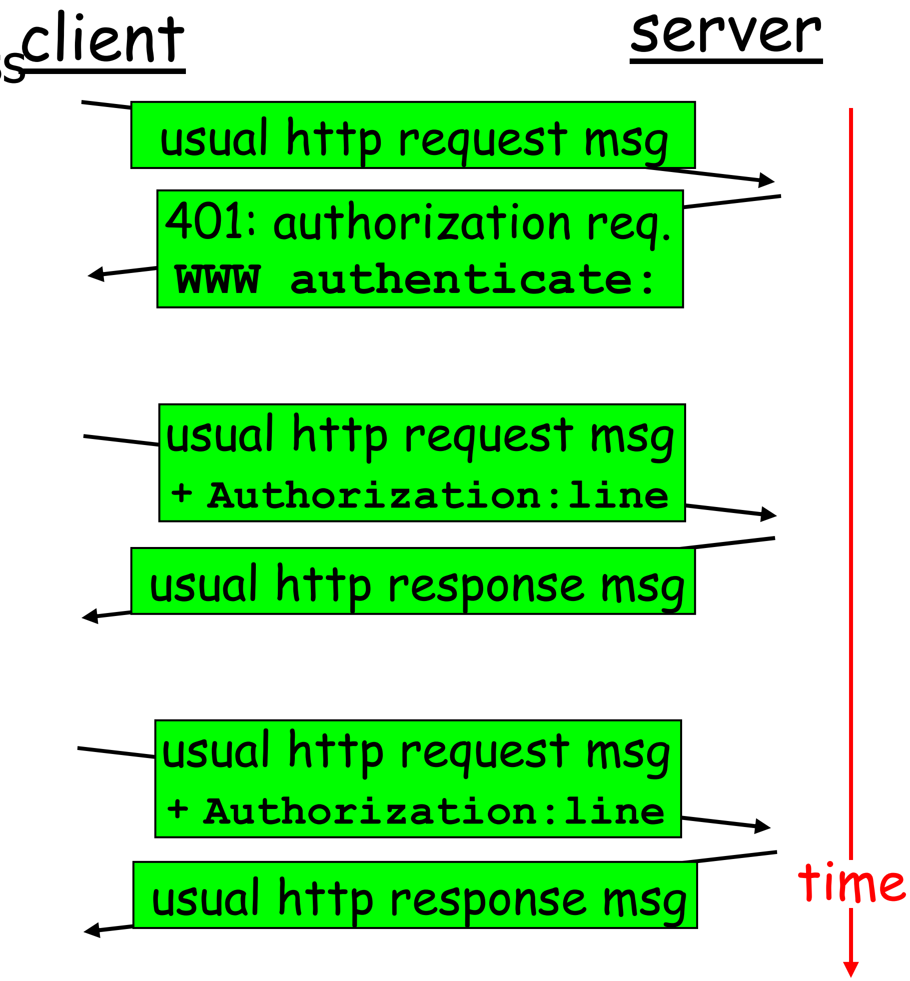
User-server interaction: authentication

Authentication goal: control access to server documents

- Stateless: client must present authorization in each request
- Authorization: typically name, password
  - `authorization:` header line in request
  - If no authorization, server refuses access, sends

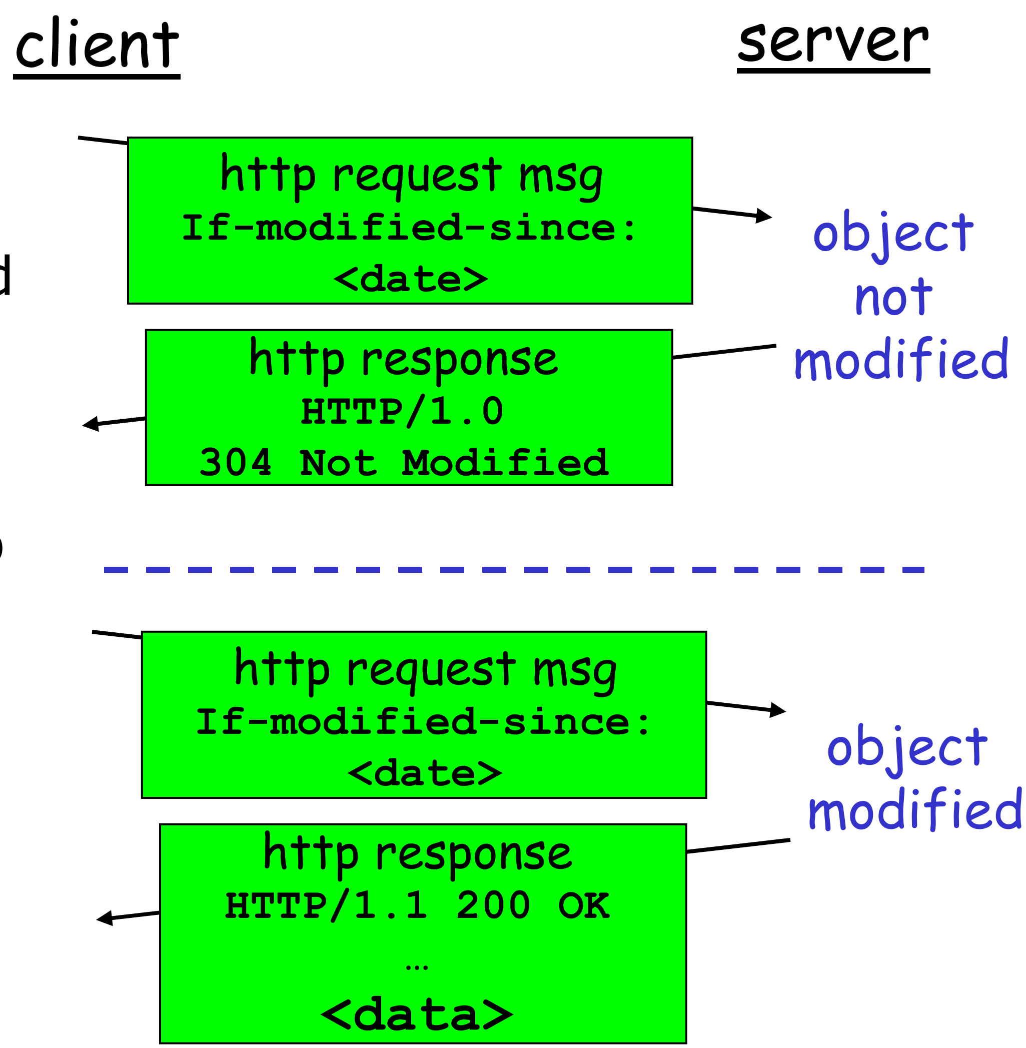
    `WWW authenticate:`

    header line in response



User-server interaction: conditional GET

- Goal: don't send object if client has up-to-date stored (cached) version
- Client: specify date of cached copy in http request
  If-modified-since: <date>
- Server: response contains no object if cached copy up-to-date:
  HTTP/1.0 304 Not Modified



User-server state: cookies

Many major Web sites use cookies

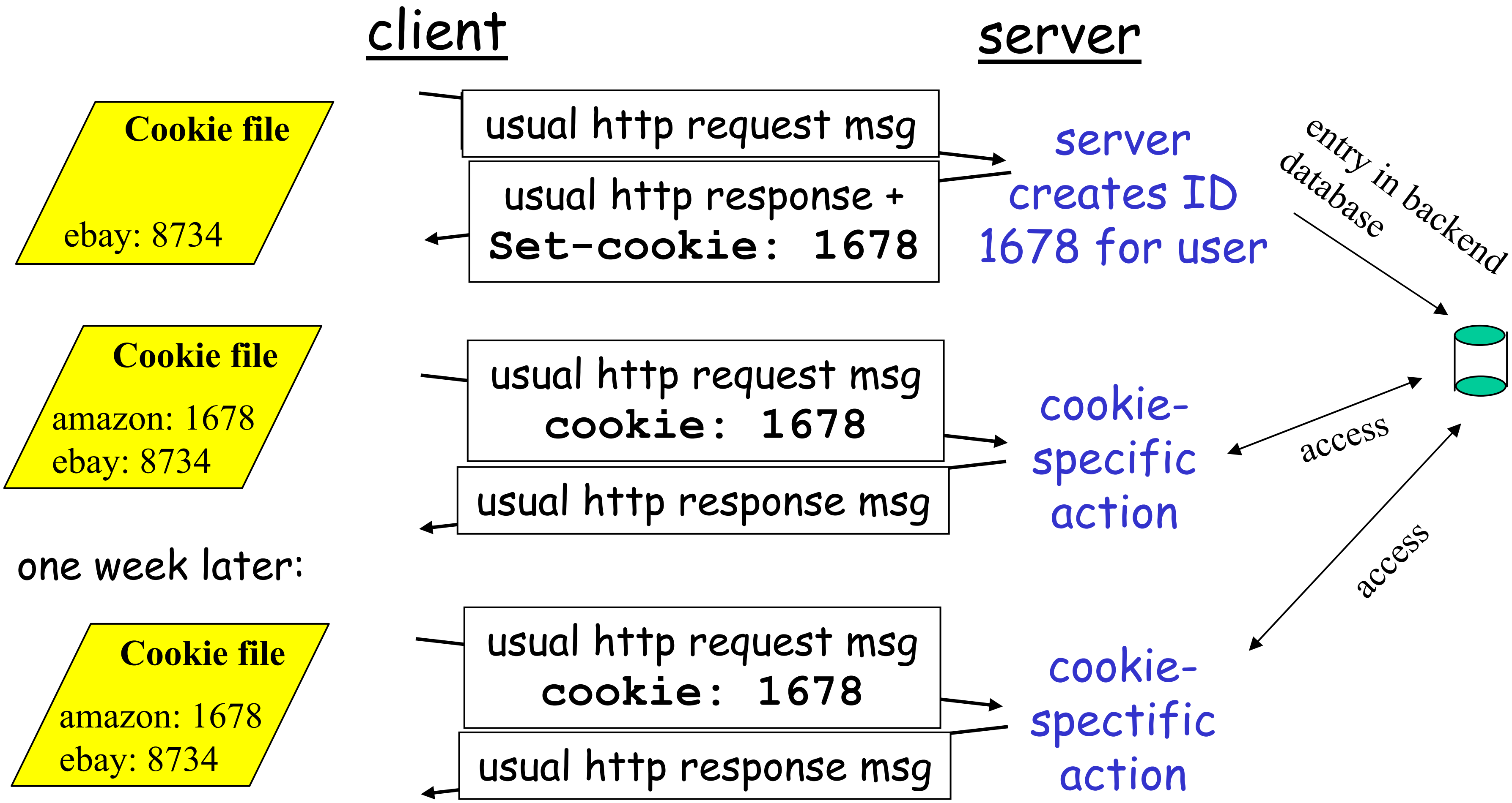
Four components:

1) cookie header line of HTTP *response* message
2) cookie header line in HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) back-end database at Web site

Example:

- Susan access Internet always from same PC
- She visits a specific e-commerce site for first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID

Cookies: keeping "state" (cont.)



Cookies (continued)

What cookies can bring:

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

aside

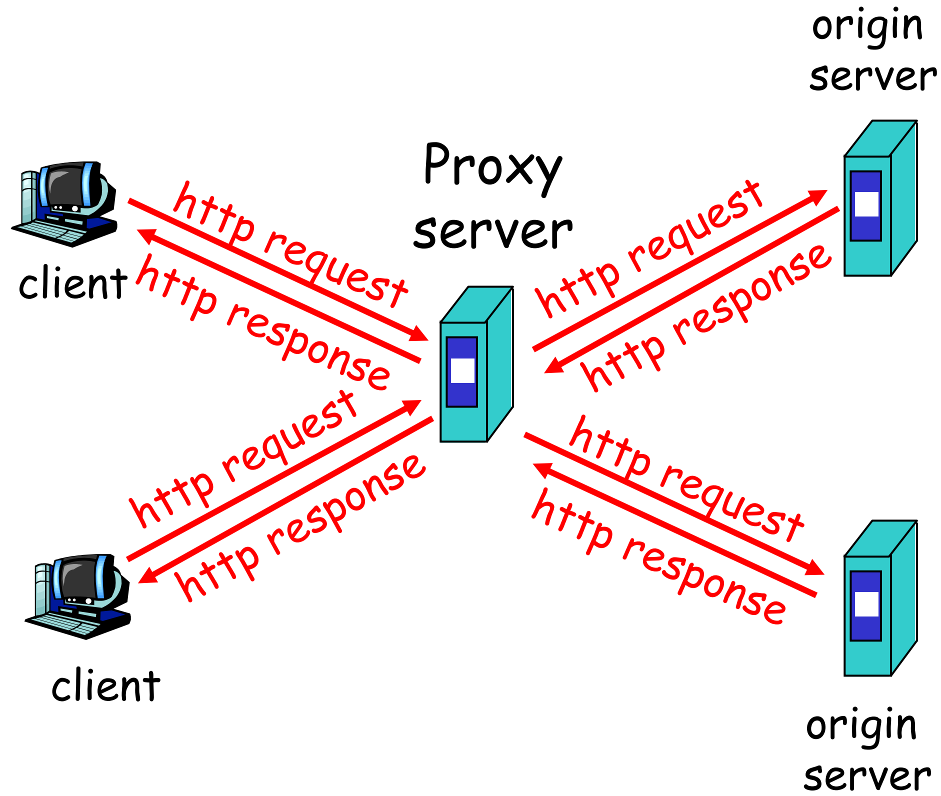
Cookies and privacy:

- 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

Web Caches (proxy server)

Goal: satisfy client request without involving origin server

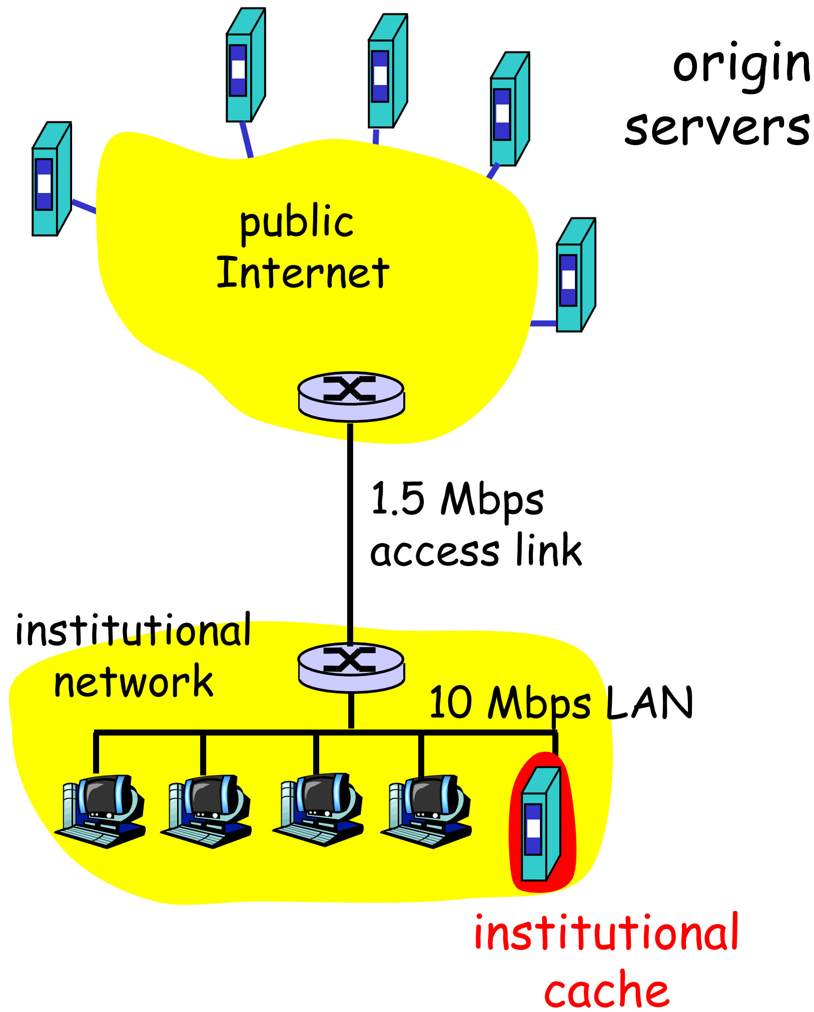
- User sets browser: WWW accesses via web cache
- Client sends all http requests to web cache
  - If object at web cache, web cache immediately returns object in http response
  - Else requests object from origin server, then returns http response to client



Why WWW Caching?

Assume: cache is "close" to client (e.g., in same network)

- smaller response time: cache "closer" to client
- decrease traffic to distant servers
  - link out of institutional/local ISP network often bottleneck



Problems with HTTP/1.0

- Lack of control: cache duration, cache location, selection among cached variants, ...
- Ambiguity of rules for proxies and caches
- Download of full resource instead of necessary part
- Poor use of TCP: short Web responses
- No guarantee for full receipt for dynamically generated responses
- Depletion of IP addresses
- Inability to tailor request, responses according to client, server preference
- Poor level of security
- ...

HTTP/1.1 concepts

- Hop-by-hop mechanism
  - Headers valid only for a single transport-level connection: Transfer-Encoding, Connection
  - Cannot be stored by caches or forwarded by proxies
- Transfer coding
  - Split: message vs. entity (including headers)
  - Content coding is applied to whole entity
  - Transfer coding applies to entity-body
    - Property of message not original entity
    - TE, Transfer-Encoding
- Virtual hosting
- Semantic transparency for caching
- Support for variants of a resource

New headers: Request

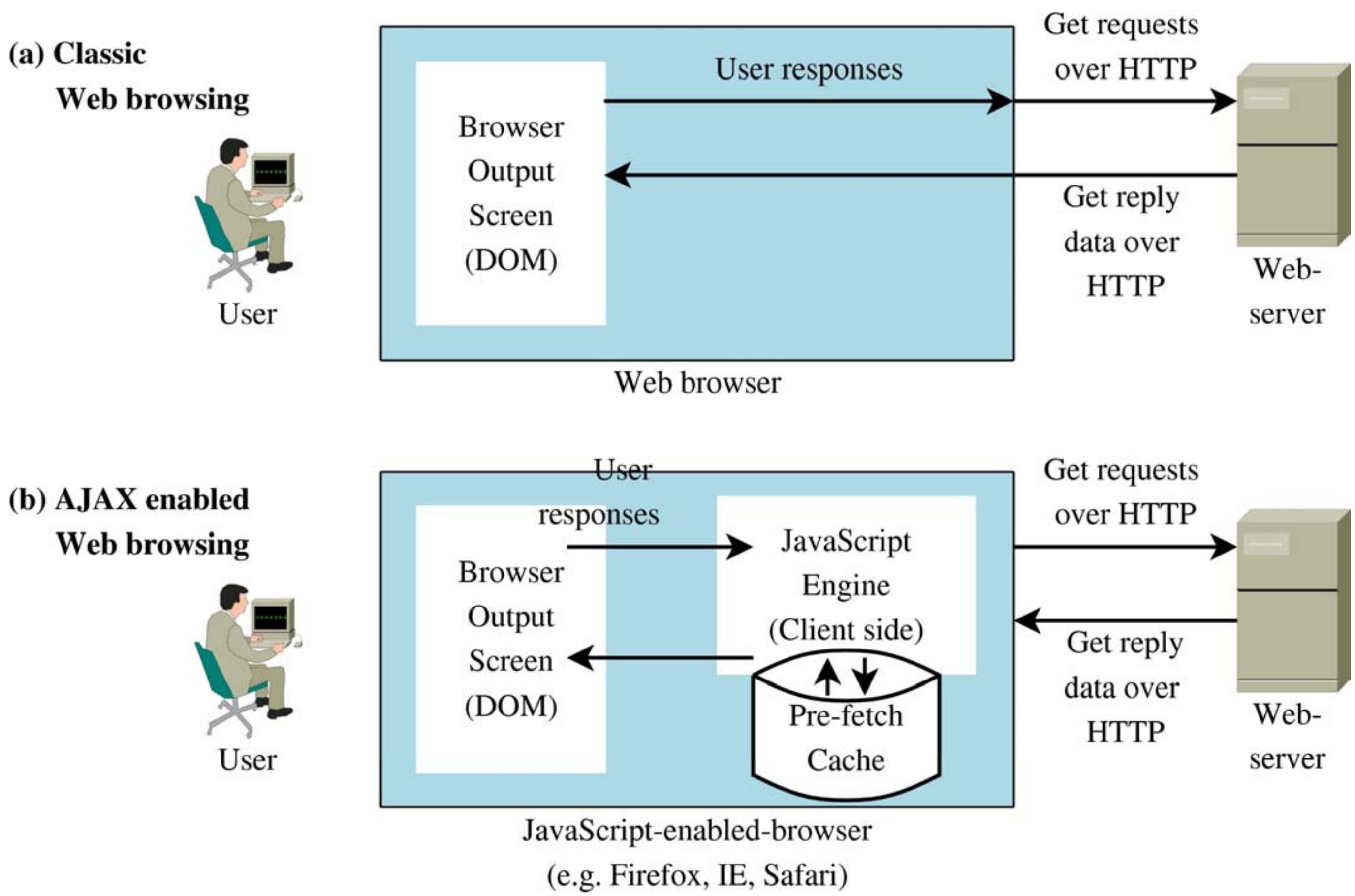
- Response preference
  - New: Accept (charset, encoding, language), TE
- Information
  - Old: Authorization, From, Referer, User-Agent
  - New: Proxy-Authorization
- Conditional request
  - Old: If-Modified-Since
  - New: If-Match, If-None-Match, If-Unmodified-Since, If-Range
- Constraint on server
  - New: Expect, Host, Max-Forwards, Range

New headers: Response

- Redirection:
  - Old: Location
- Information
  - Old: Server
  - New: Retry-After, Accept-Ranges
- Security related
  - Old: WWW-Authenticate
  - New: Proxy-Authenticate
- Caching related
  - New: Etag, Age, Vary

Web 2.0: e.g., AJAX enabled apps

- E.g.: Google Maps: a canonical AJAX application

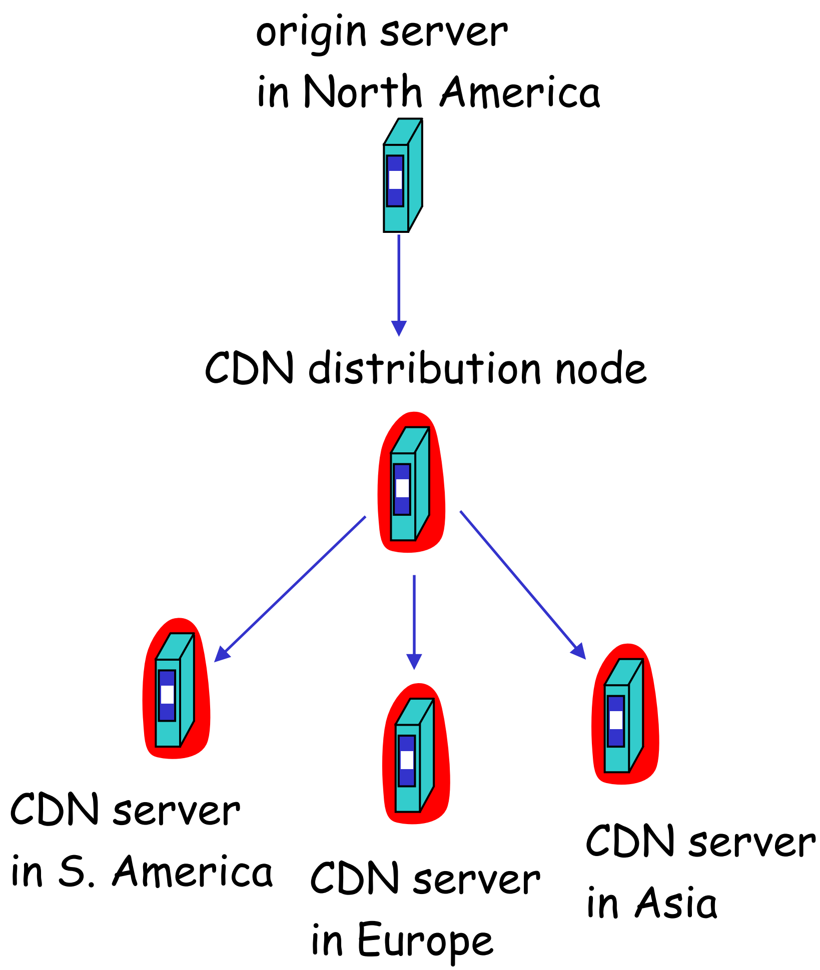


Content distribution networks (CDNs)

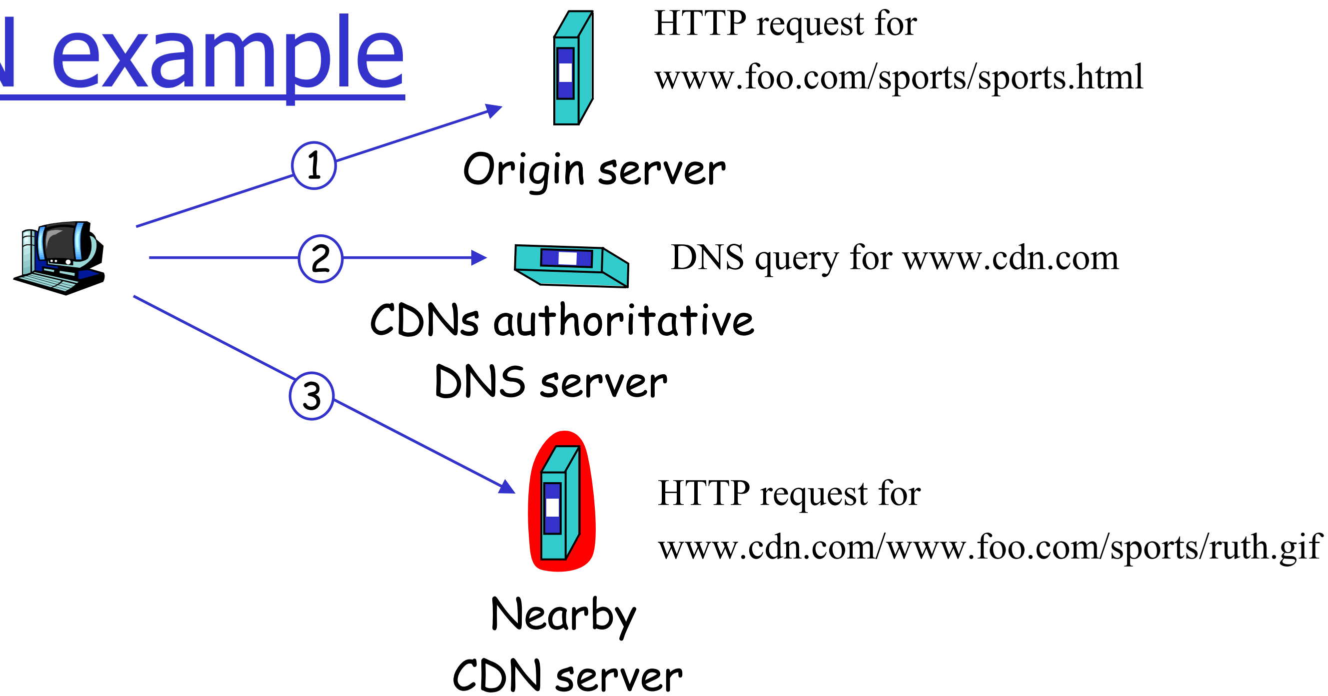
- The content providers are the CDN customers.

Content replication

- CDN company installs hundreds of CDN servers throughout Internet
    - in lower-tier ISPs, close to users
- CDN replicates its customers' content in CDN servers. When provider updates content, CDN updates servers



CDN example



origin server

- www.foo.com
- distributes HTML
- Replaces:

  http://www.foo.com/sports.ruth.gif

  with

  http://www.cdn.com/www.foo.com/sports/ruth.gif

CDN company

- cdn.com
- distributes gif files
- uses its authoritative DNS server to route redirect requests

More about CDNs

routing requests

- CDN creates a "map", indicating distances from leaf ISPs and CDN nodes
- when query arrives at authoritative DNS server:
  - server determines ISP from which query originates
  - uses "map" to determine best CDN server

not just Web pages

- streaming stored audio/video
- streaming real-time audio/video
  - CDN nodes create application-layer overlay network

DNS: Domain Name System

People: many identifiers:

- SSN, name, Passport #

Internet hosts, routers:

- IP address (32 bit) – used for addressing datagrams
- "name", e.g., gaia.cs.umass.edu – used by humans

Q: Map between IP addresses and name?

- Secure Domain Name System (DNS) Dynamic Update: RFC 3007

DNS: Domain Name System

Domain Name System:

- *Distributed database:* implemented in hierarchy of many *name servers*
- *Application-layer protocol:* host, routers, name servers communicate to *resolve* names (address/name translation)
  - Core Internet function implemented as application-layer protocol
  - Complexity at network's "edge"

DNS name servers

Why not centralize DNS?

- Single point of failure
- Traffic volume
- Distant centralized database
- Maintenance

Does not *scale!*

DNS name servers (2)

No server has all name-to-IP address mappings

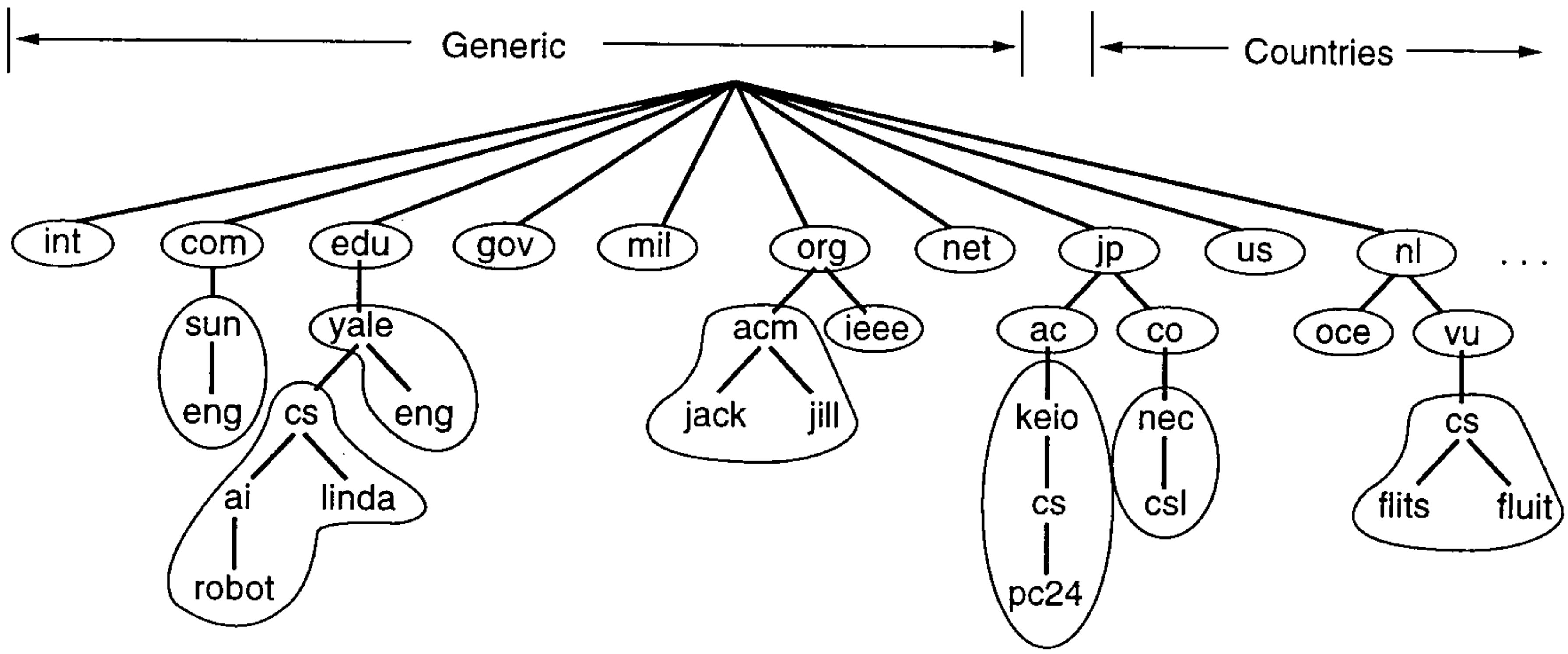
Local name servers:

- Each ISP, company has *local (default) name server*
- Host DNS query first goes to local name server

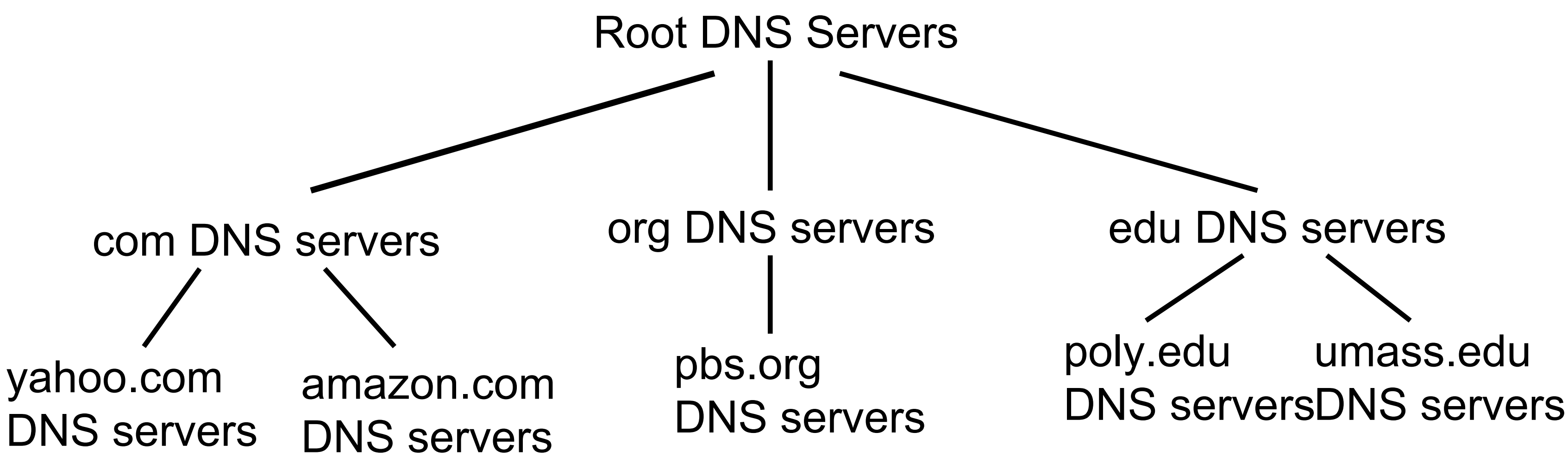
Authoritative name server:

- For a host: stores that host's IP address, name
- Can perform name/address translation for that host's name

DNS: hierarchical naming tree



Distributed, hierarchical database

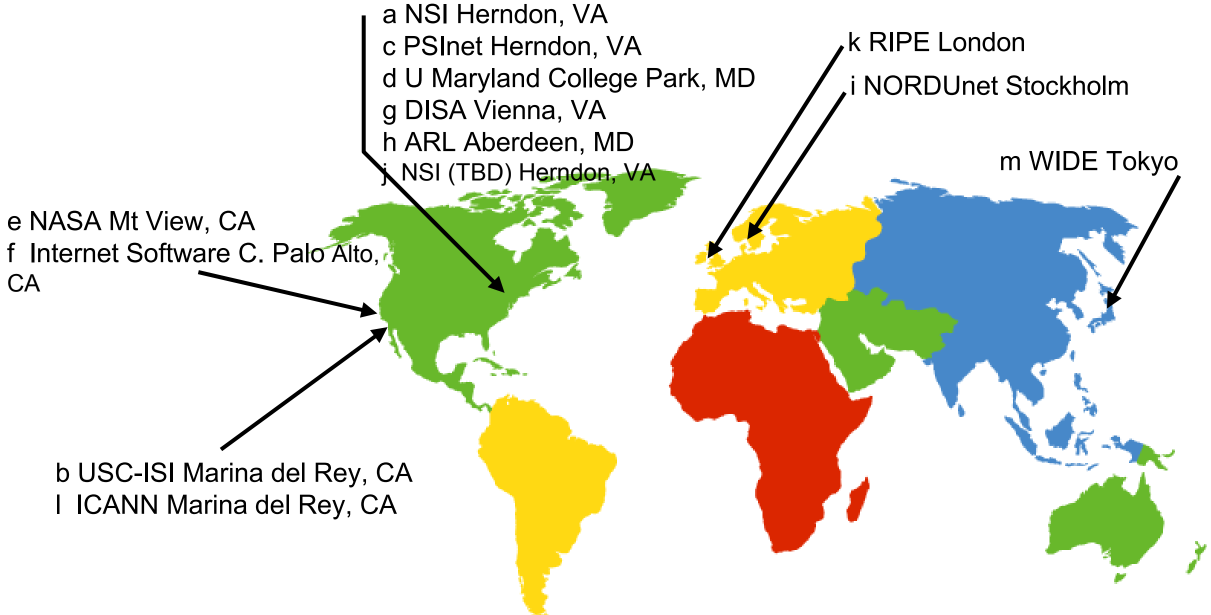


Client wants IP for www.amazon.com; 1st approx:

- Client queries a 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
  - Some use anycast



13 root name servers worldwide

TLD and Authoritative 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
- Authoritative DNS servers: organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web and mail).
  - Can be maintained by organization or service provider

Local Name Server

- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one.
    - Also called "default name server"
- When a host makes a DNS query, query is sent to its local DNS server
    - Acts as a proxy, forwards query into hierarchy.

DNS records

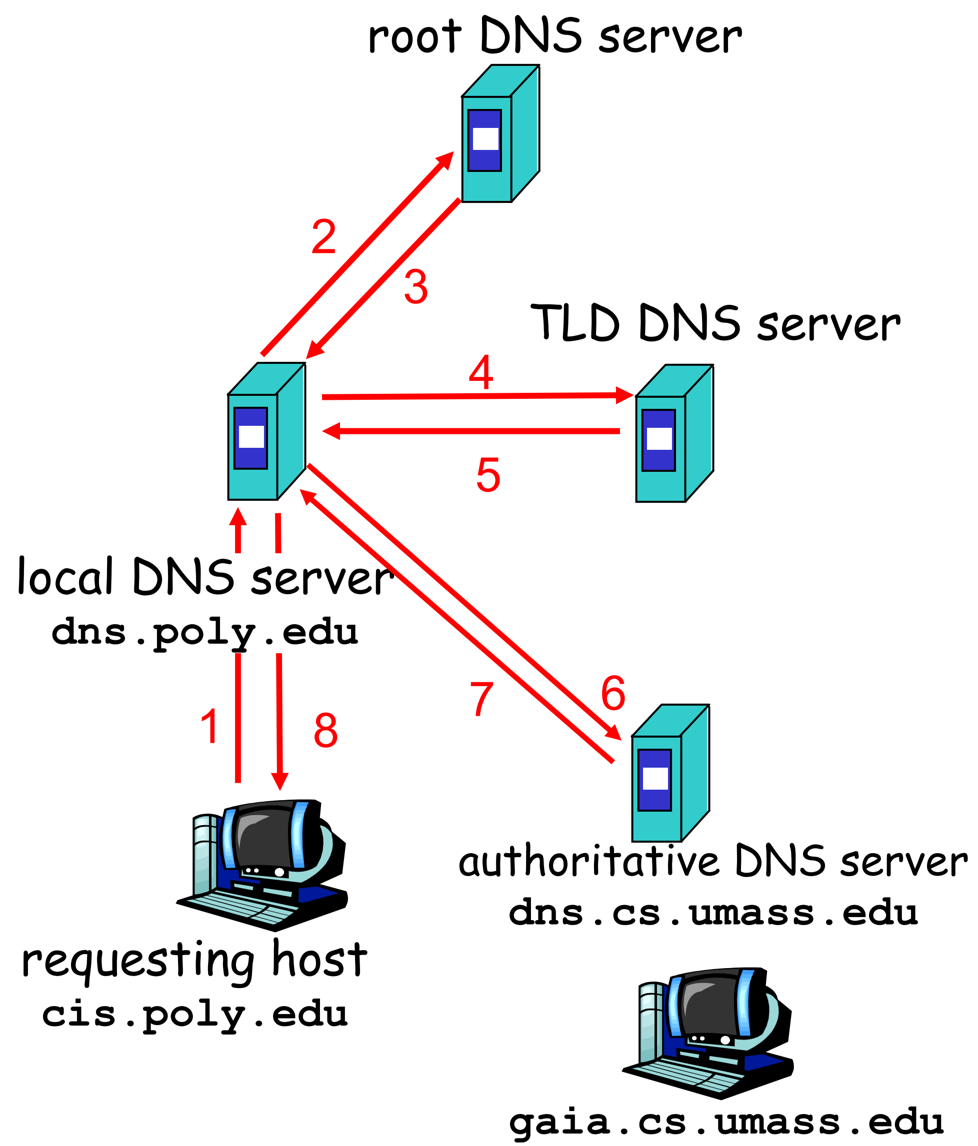
DNS: distributed db storing resource records (RR)

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 IP address of authoritative name server for this domain
- Type=CNAME
  - for alias
- Type=MX
  - for mail

Example

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



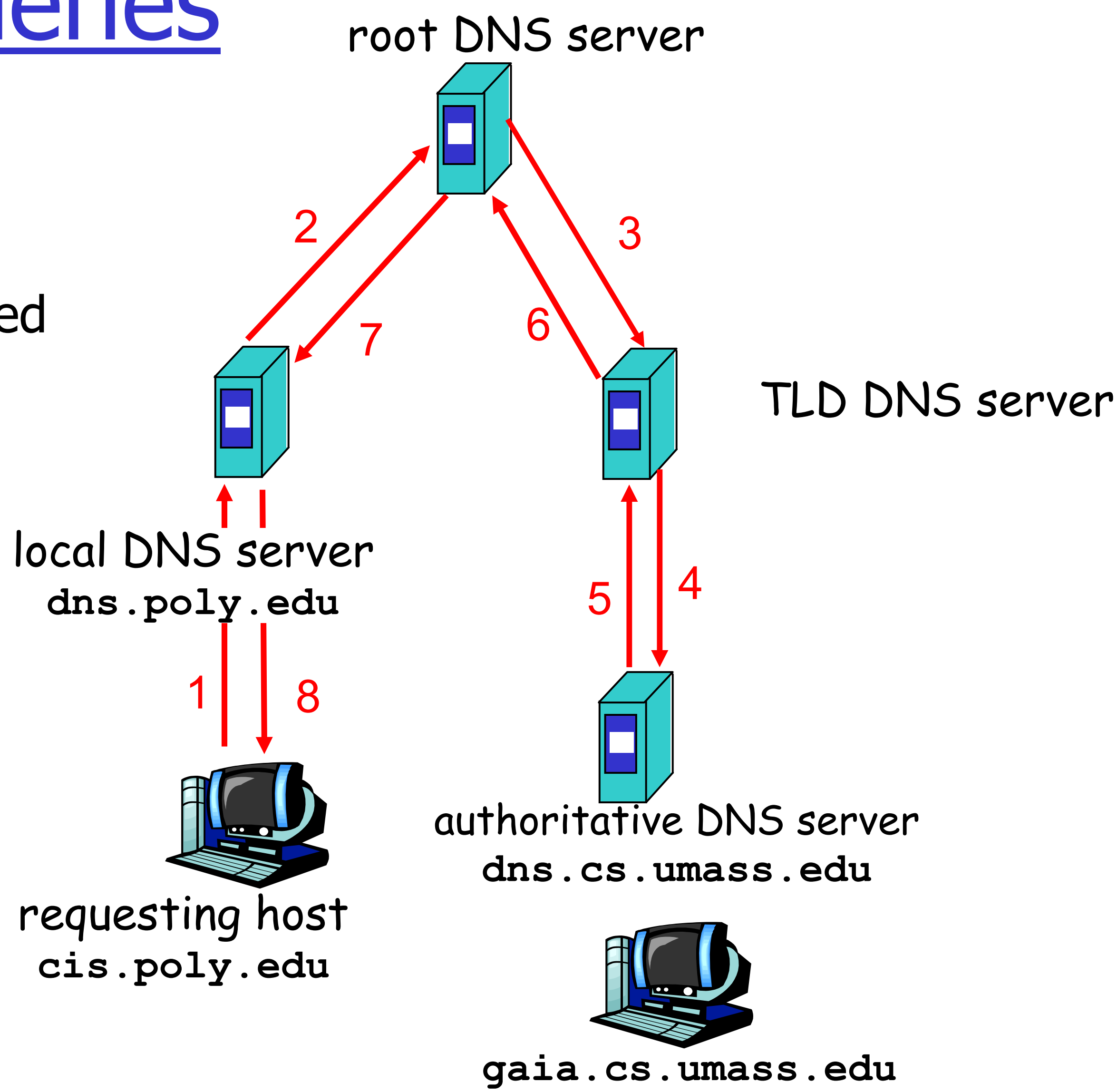
Recursive queries

recursive query:

- puts burden of name resolution on contacted name server
- heavy load?

iterated query:

- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



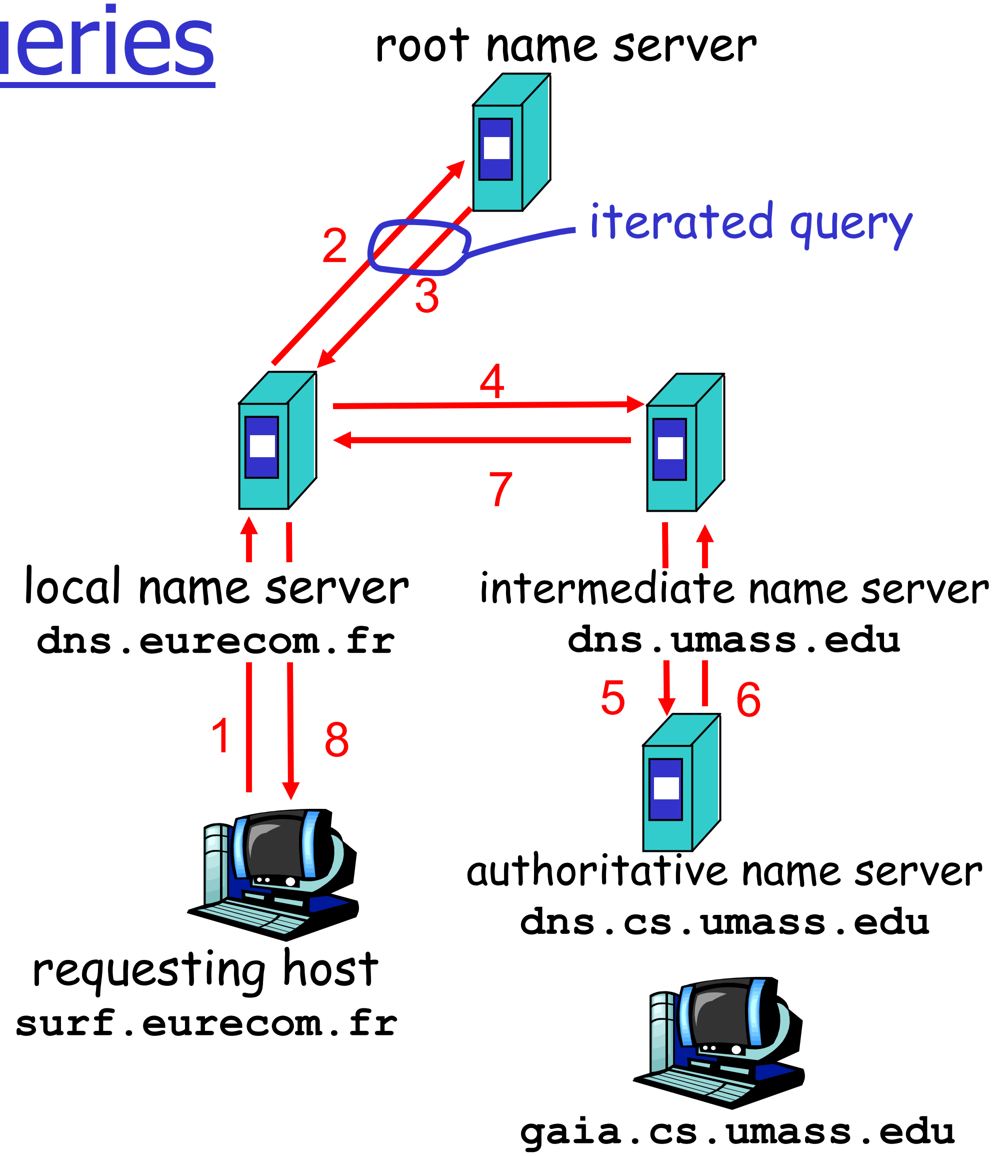
DNS: iterated queries

Recursive query:

- Puts burden of name resolution on contacted name server
- Heavy load?

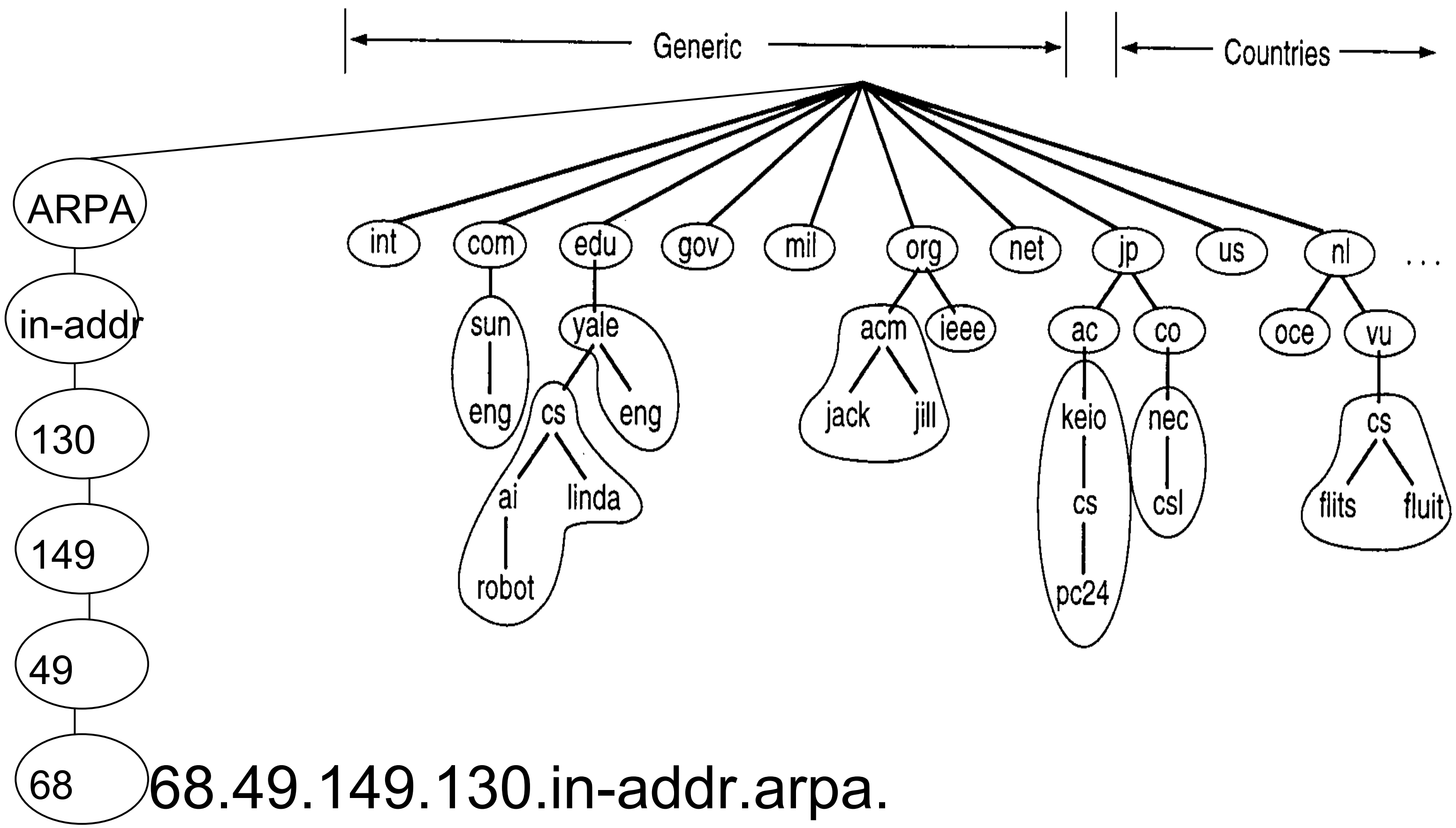
Iterated query:

- Contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



Mapping IP address to names

- Special domain: ARPA



DNS: caching and updating records

- Once (any) name server learns mapping, it *caches* mapping
  - Cache entries timeout (disappear) after some time
- Update/notify mechanisms under design by IETF
  - RFC 3007 (Feb. 2004)
  - http://www.ietf.org/html.charters/dnsind-charter.html

Inserting records into DNS

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

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

- Put in authoritative server Type A record for www.networkuptopia.com and Type MX record for networkutopia.com
- How do people get the IP address of your Web site?