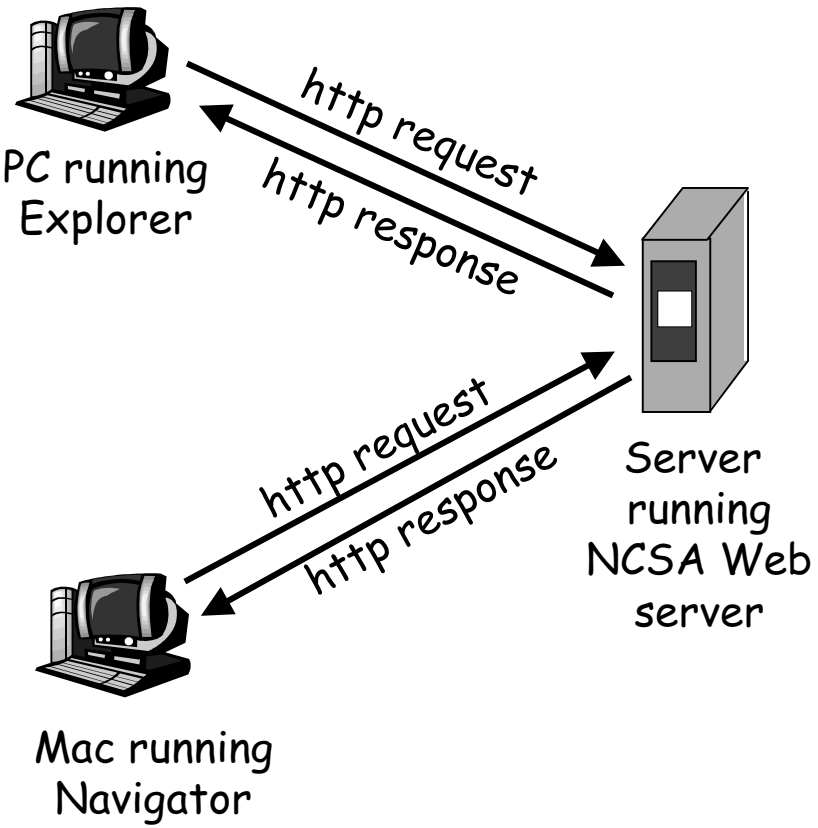
# Internet apps: their protocols and transport protocols

| Application | Application layer protocol | Underlying transport protocol |
|---|---|---|
| e-mail | smtp [RFC 821] | TCP |
| remote terminal access | telnet [RFC 854] | TCP |
| Web | http [RFC 2068] | TCP |
| file transfer | ftp [RFC 959] | TCP |
| streaming multimedia | proprietary (e.g. RealNetworks) | TCP or UDP |
| remote file server | NSF | TCP or UDP |
| Internet telephony | proprietary (e.g., Vocaltec) | typically 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
- http1.0: RFC 1945
- http1.1: RFC 2068



# 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!

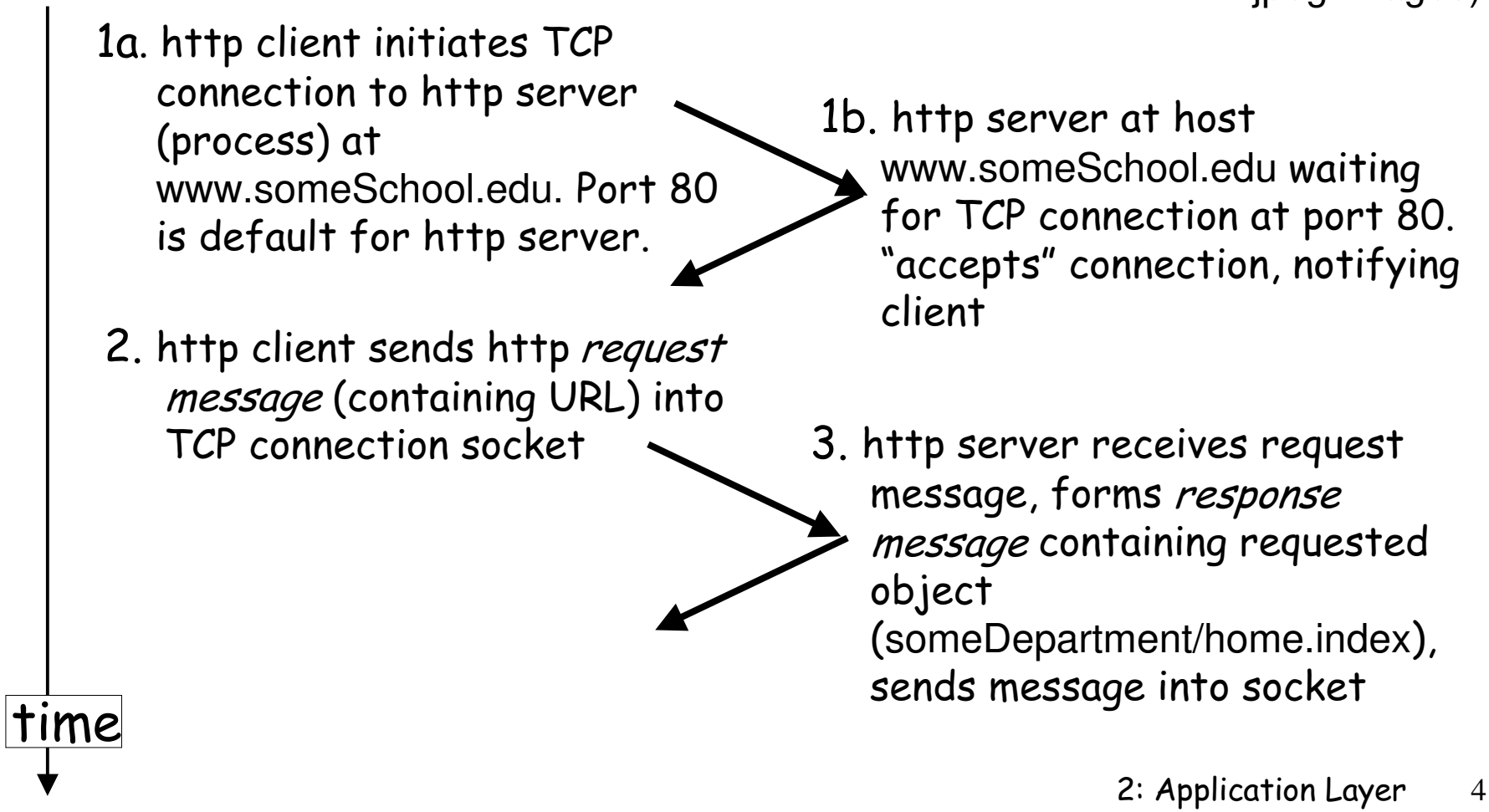
- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

# http example

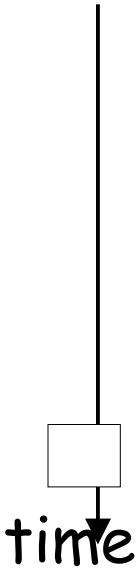
Suppose user enters URL

www.someSchool.edu/someDepartment/home.index

(contains text, references to 10 jpeg images)



# http example (cont.)




4. http server closes TCP connection.

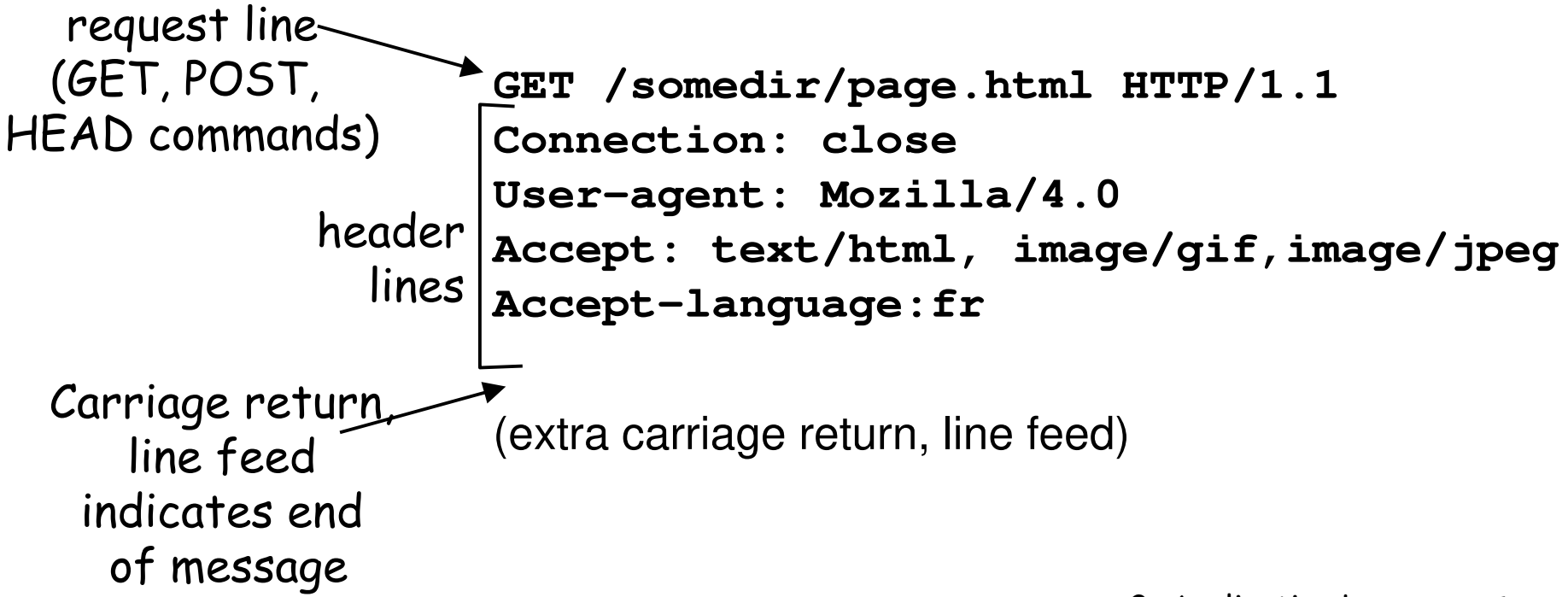
5. http client receives response message containing html file, displays html. Parsing html file, findis10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

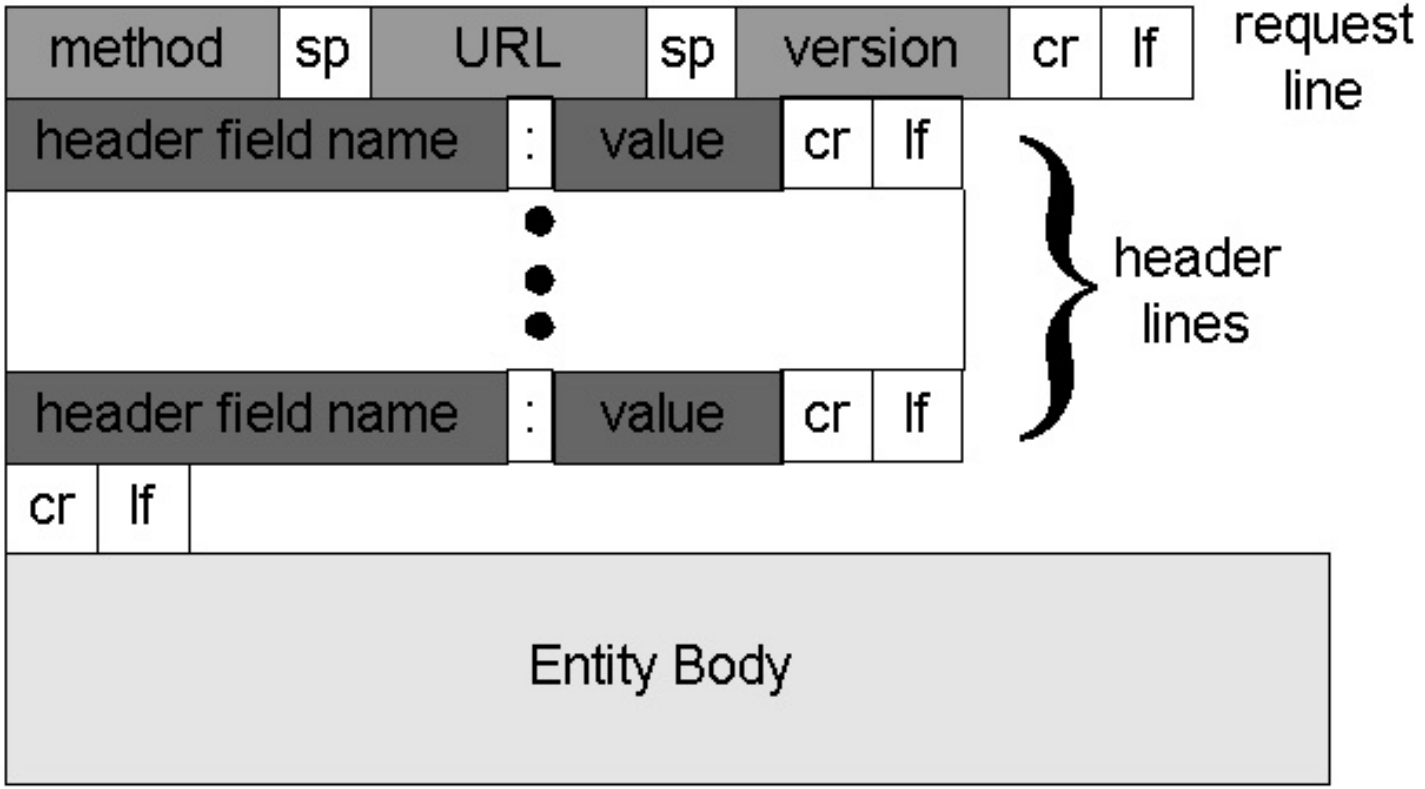
- non-persistent connection: one object in each TCP connection
  - some browsers create multiple TCP connections *simultaneously* - one per object
- persistent connection: multiple objects transferred within one TCP connection

# http message format: request

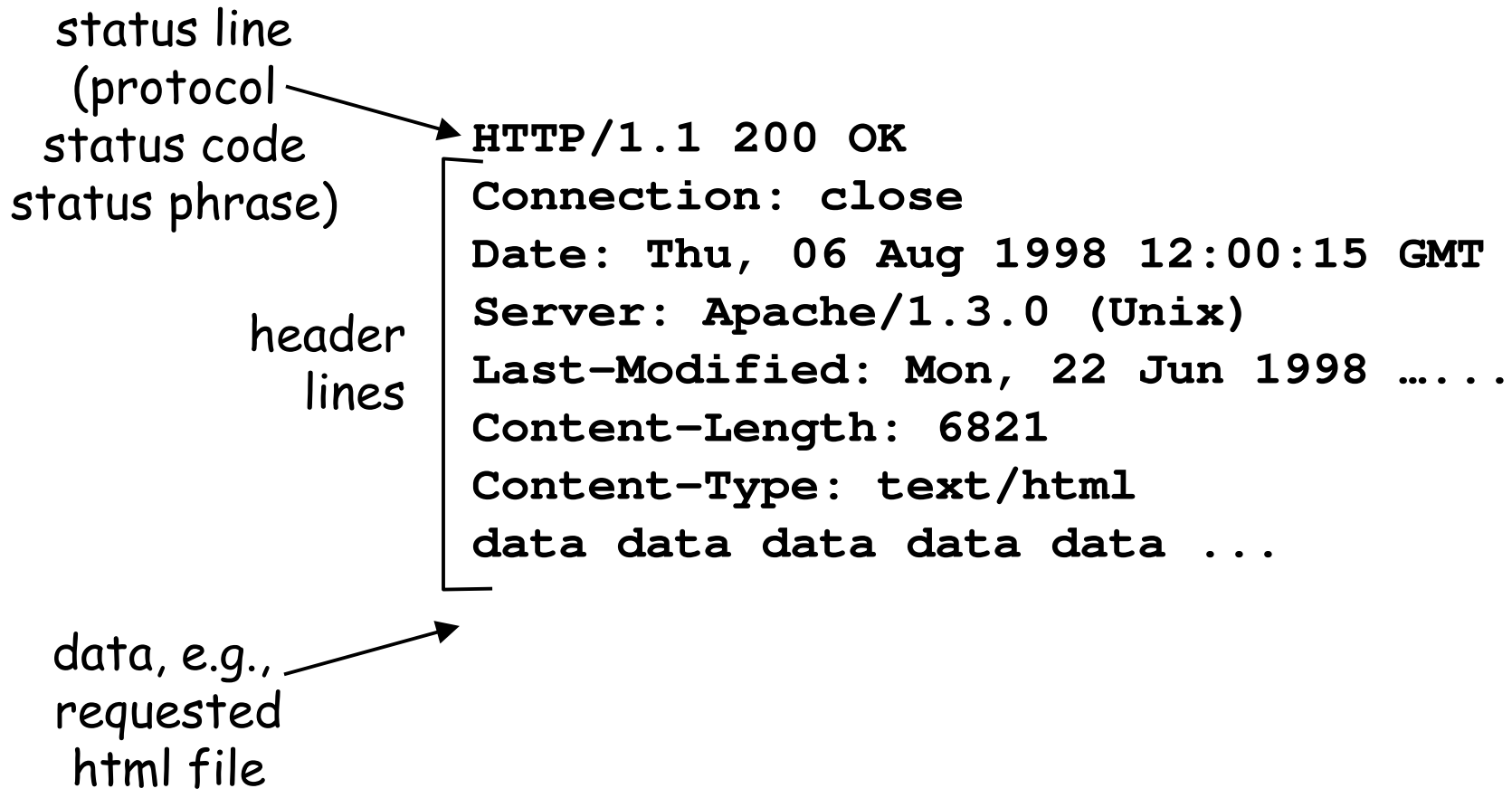
- two types of http messages: *request*, *response*
- http request message:
  - ASCII (human-readable format)



# http request message: general format



# http message format: reply



# 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**

# Trying out http (client side) for yourself

1. Telnet to your favorite WWW server:

```
telnet www.eurecom.fr 80
```

Opens TCP connection to port 80 (default http server port) at www.eurecom.fr. Anything typed in sent to port 80 at www.eurecom.fr

2. Type in a GET http request:

```
GET /~ross/index.html HTTP/1.0
```

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!

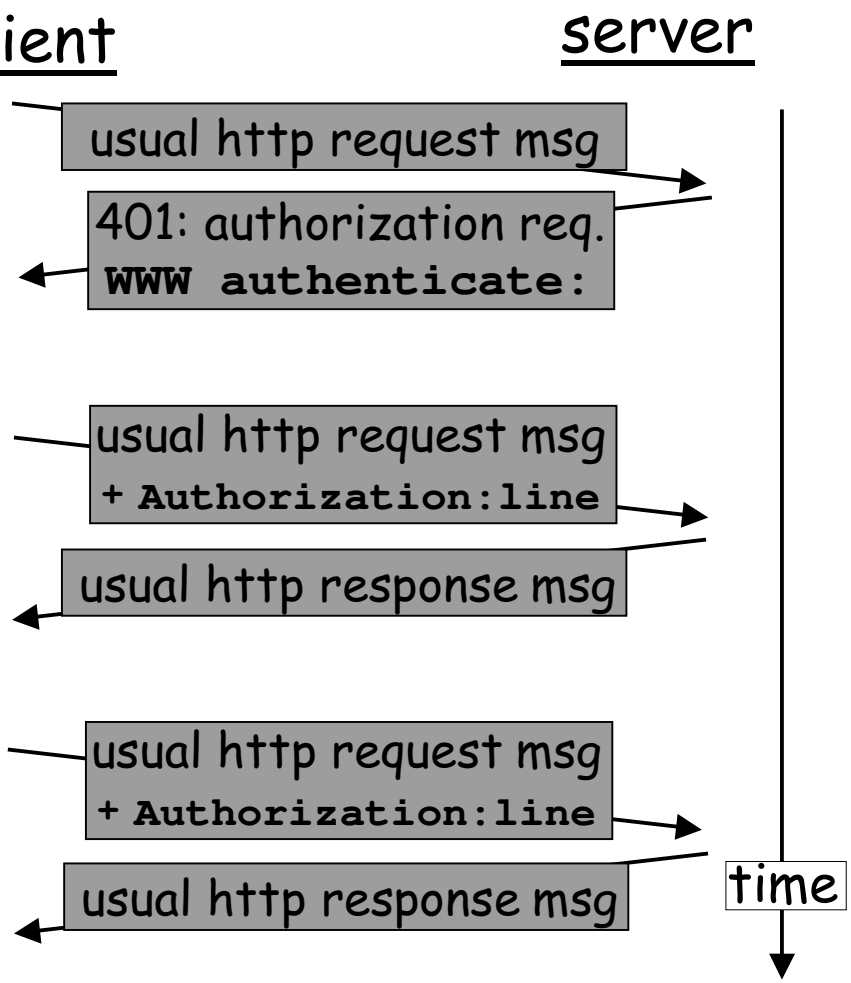
# User-server interaction: authentication

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 presented, server refuses access, sends

    **`WWW authenticate:`**

    header line in response

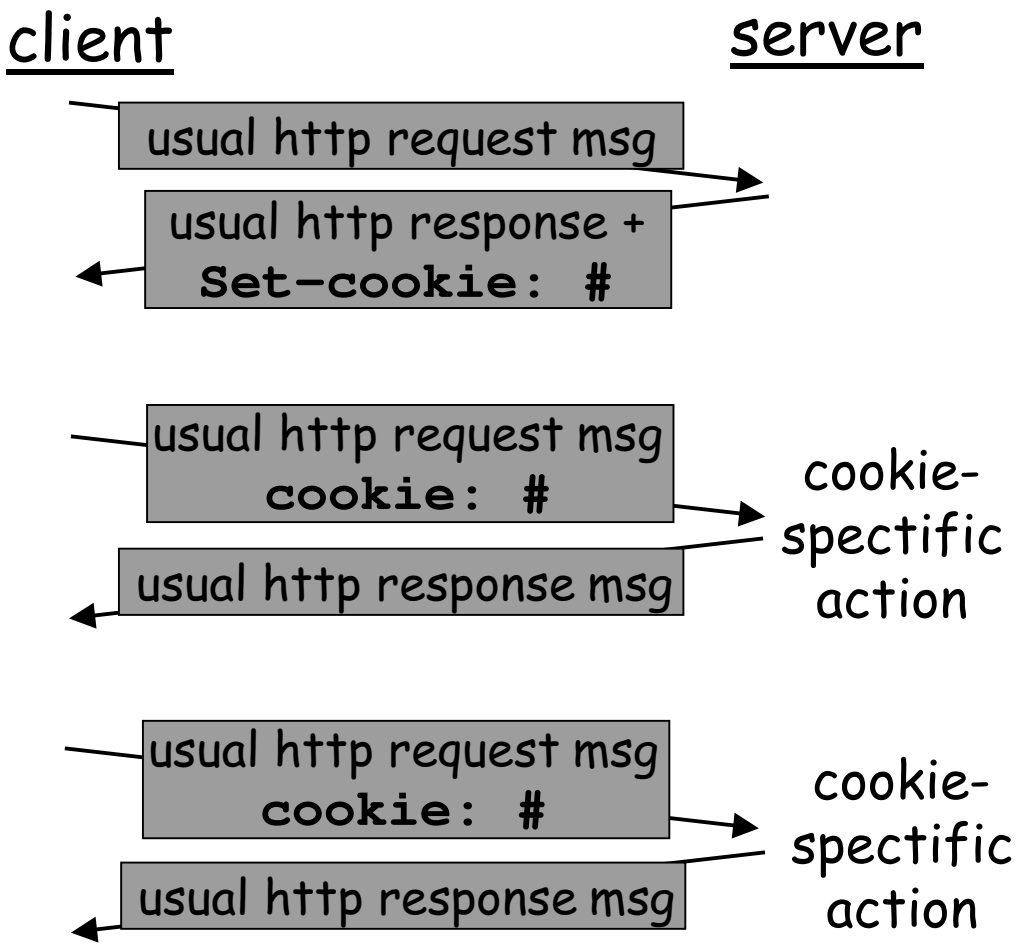


# User-server interaction: cookies

- server sends "cookie" to client in response

  **`Set-cookie: #`**
- client present cookie in later requests

  **`cookie: #`**
- server matches presented-cookie with server-stored cookies
  - authentication
  - remembering user preferences, previous choices



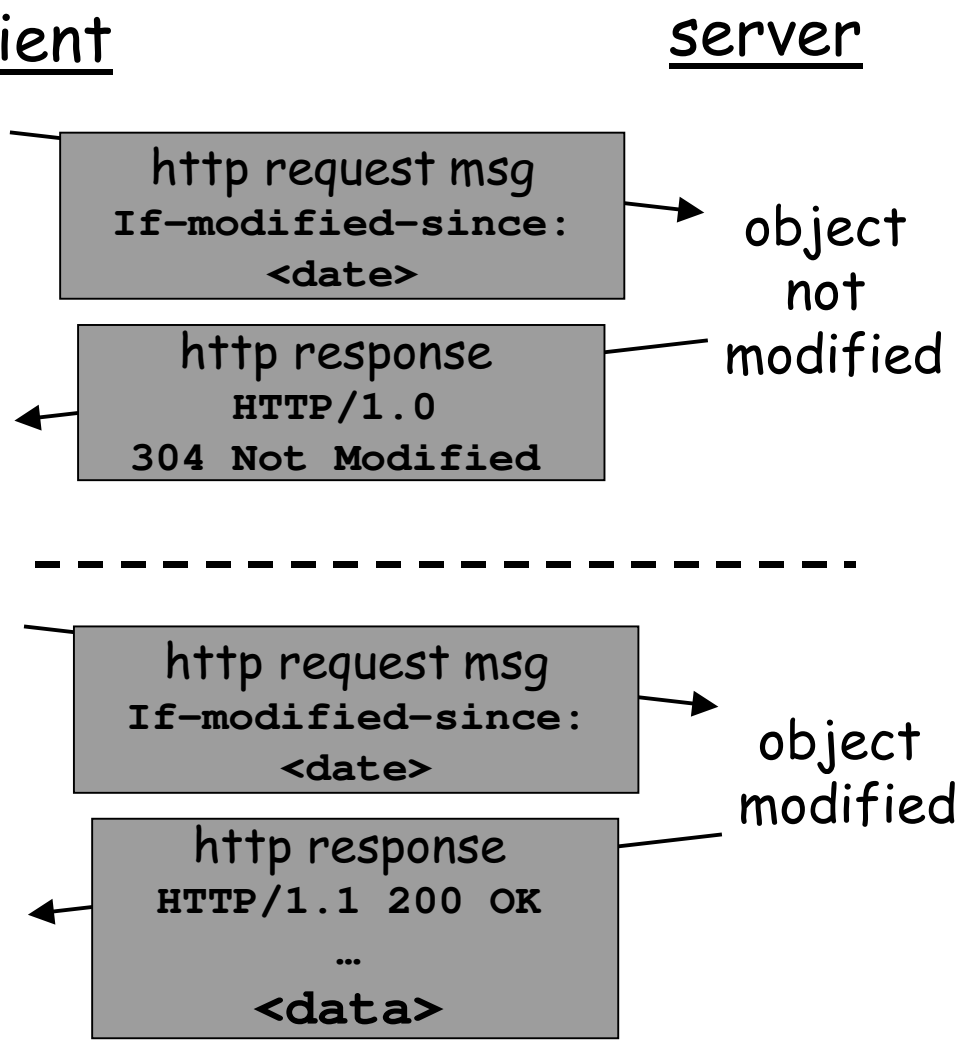
# User-server interaction: conditional GET

- 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:

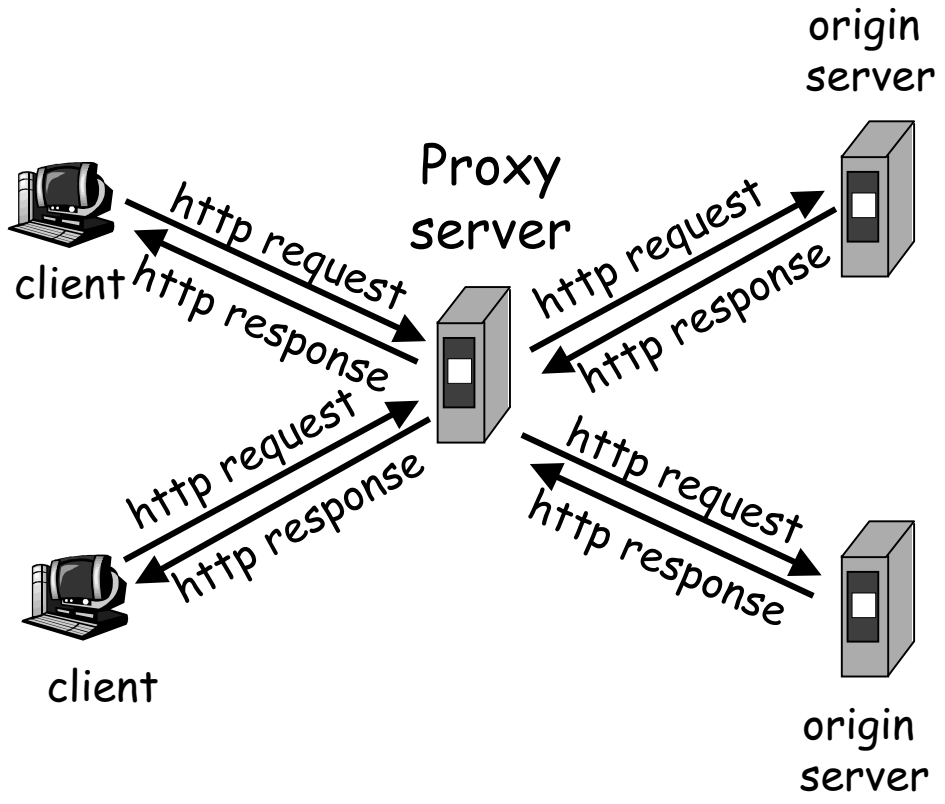
```
HTTP/1.0 304 Not
   Modified
```



# Web Caches (proxy server)

Goal: satisfy client request without involving origin server

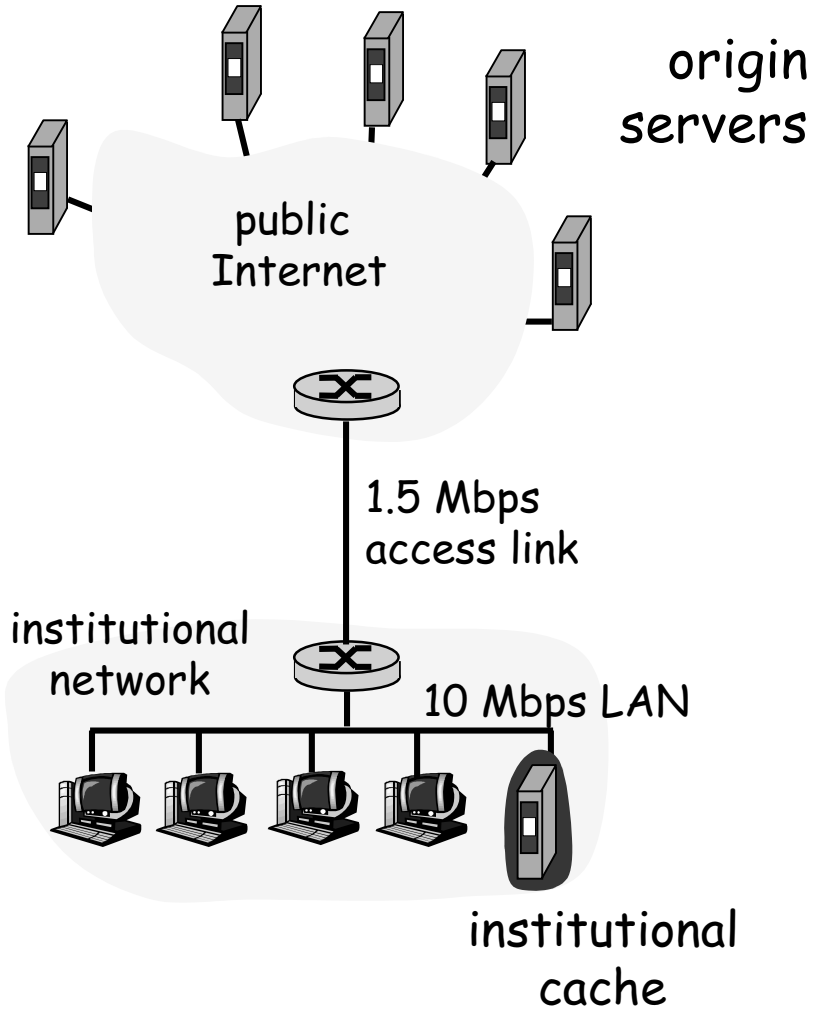
- user sets browser: WWW accesses via web cache
- client sends all http requests to web cache
  - if object at web cache, web cache immediately returns object in http response
  - else requests object from origin server, then returns http response to client



# 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



# 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 ?

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol* host, routers, name servers to communicate to *resolve* names (address/name translation)
  - note: 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

doesn't *scale!*

- no server has all name-to-IP address mappings

local name servers:

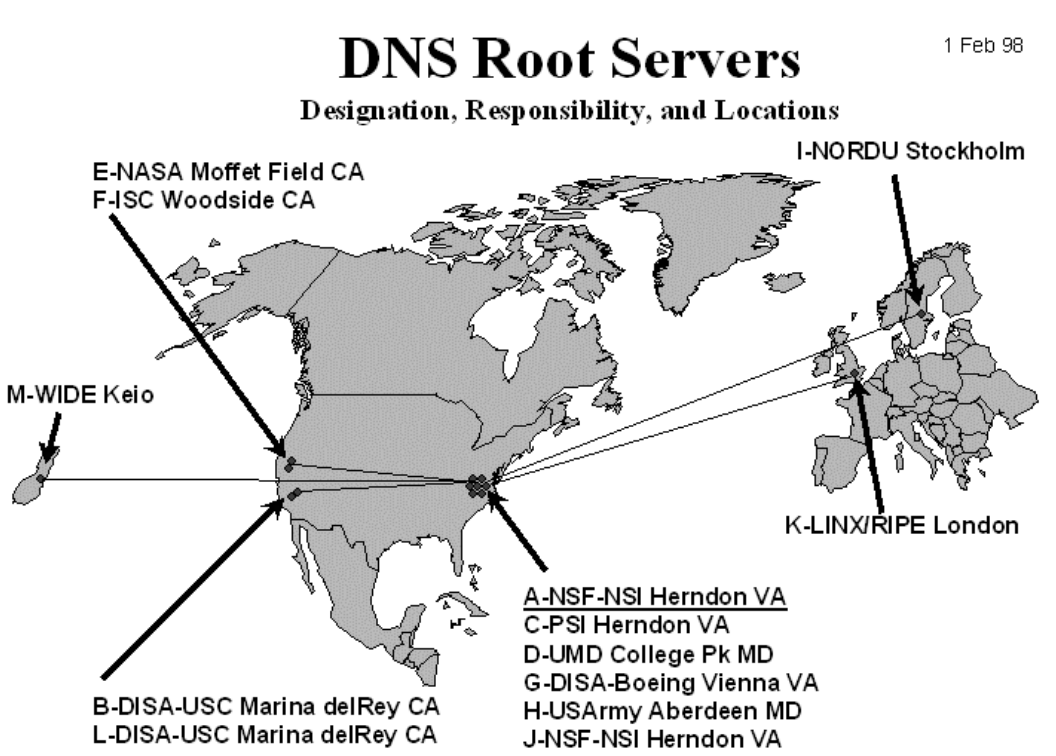
- each ISP, company has *local (default) name server*
- host DNS query first goes to local name server

authoritative name server:

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

# DNS: Root name servers

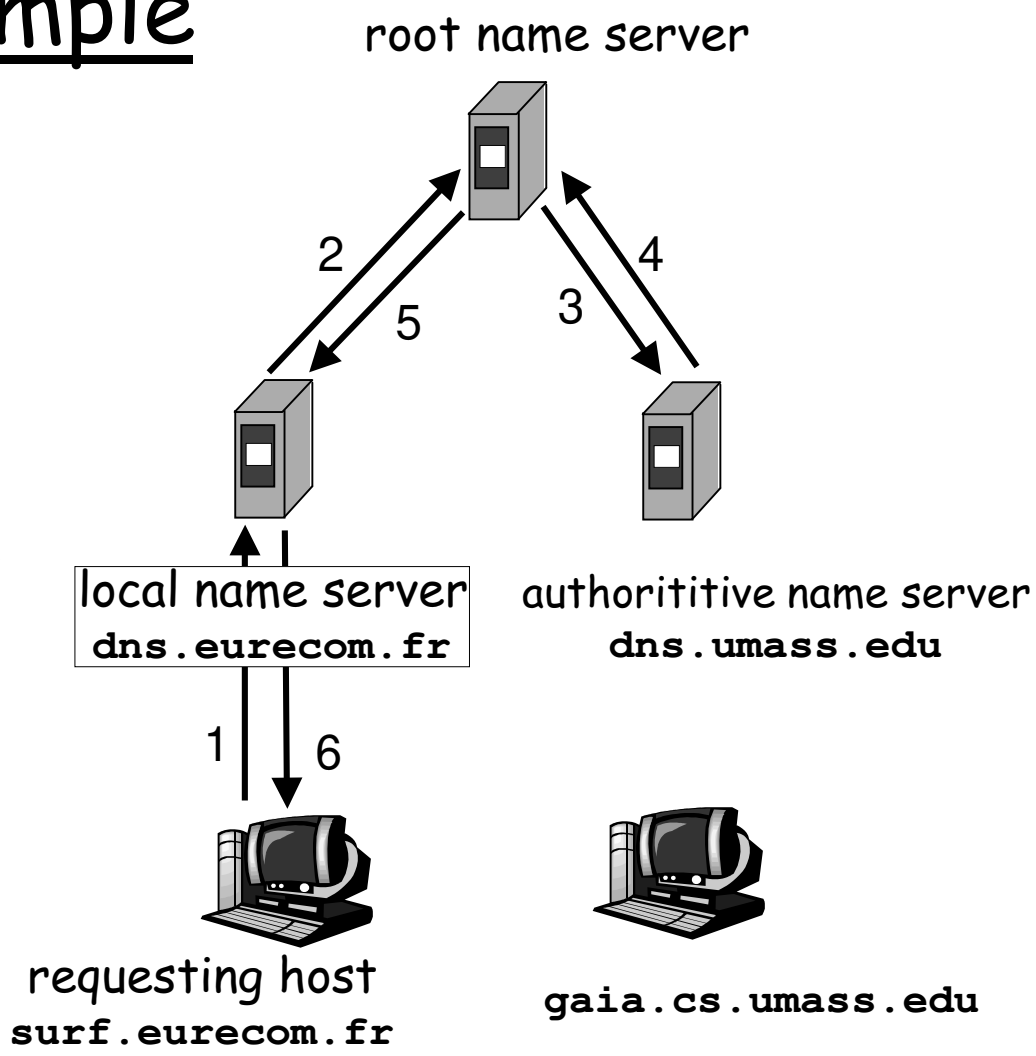
- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server
- ~ dozen root name servers worldwide



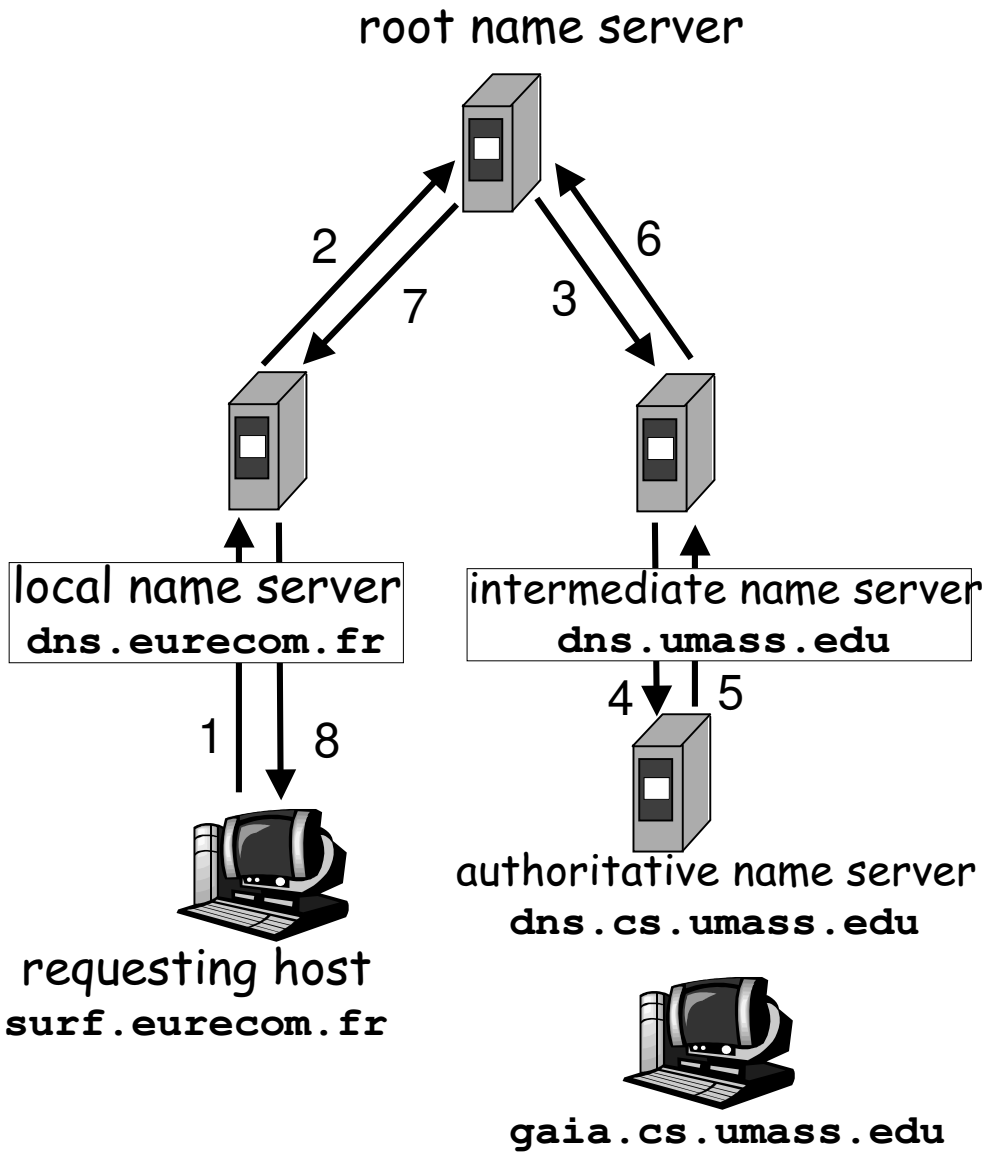
# Simple DNS example

host **surf.eurecom.fr** wants IP address of **gaia.cs.umass.edu**

1. Contacts its local DNS server, **dns.eurecom.fr**
2. **dns.eurecom.fr** contacts root name server, if necessary
3. root name server contacts authoritative name server, **dns.umass.edu**, if necessary



# DNS example

Root name server:

- may not know authoratiative name server
- may know *intermediate name server:* who to contact to find authoritative name server