Network Applications: HTTP and DNS

Outline

- Application examples
  - Email
  - FTP
  - HTTP

The Web: Some Jargon

- Web page:
  - Consists of "objects"
  - Addressed by a URL
- Most Web pages consist of:
  - Base HTML page, and
  - Several referenced objects
- URL has two components: host name, port number and path name:

- User agent for Web is called a browser, e.g.
  - Mozilla Firefox
  - MS Internet Explorer
- Server for Web is called Web server:
  - Apache
  - MS Internet Information Server

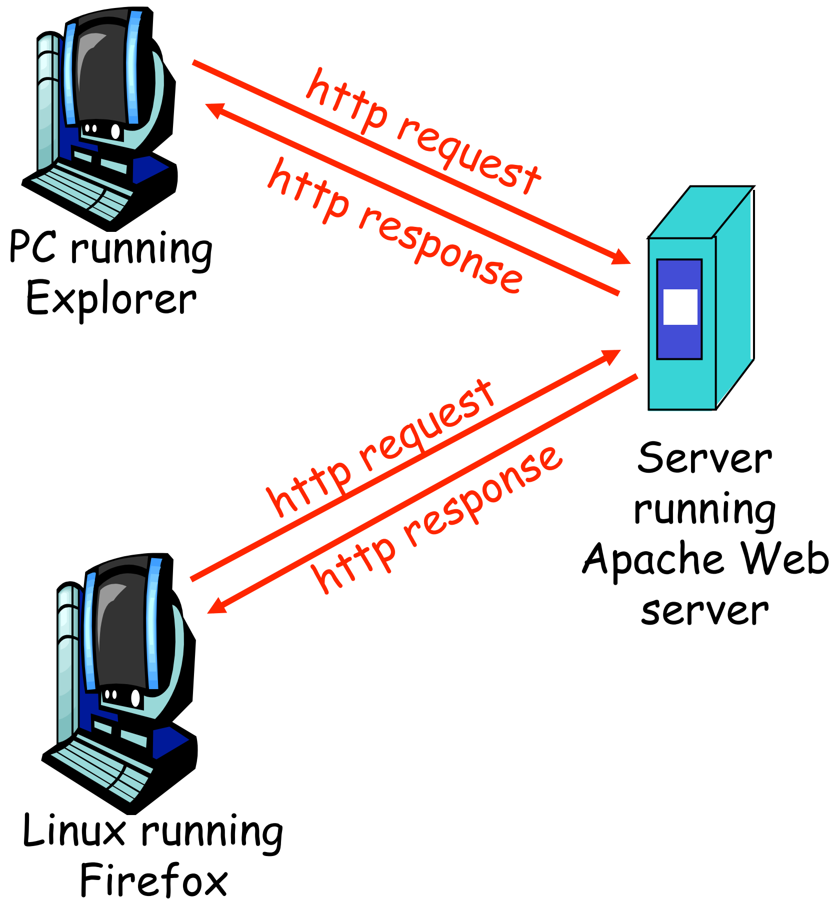
`http://www.sjtu.edu.cn:80/index.html`

The Web: the HTTP Protocol

HTTP: hypertext transfer protocol

- Web's application layer protocol
- HTTP uses TCP as transport service
- Client/server model
  - *Client:* browser that requests, receives, "displays" Web objects
  - *Server:* Web server sends objects in response to requests

- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



HTTP 1.0 Message Flow

- Client initiates TCP connection (creates socket) to server, port 80
- Server waits for requests from clients
- Client sends request for a document
- Web server sends back the document
- TCP connection closed

- Client parses the document to find embedded objects (images)
  - Repeat above for each image

HTTP 1.0 Message Flow (more detail)

Suppose user enters URL
www.sjtu.edu.cn/index.html

0. http server at host www.sjtu.edu.cn waiting for TCP connection at port 80.

1a. http client initiates TCP connection to http server (process) at www.sjtu.edu.cn. Port 80 is default for http server.

1b. server "accepts" connection, ack. client

2. http client sends http *request message* (containing URL) into TCP connection socket

3. http server receives request message, forms *response message* containing requested object (index.html), sends message into socket (the sending speed increases slowly, which is called slow-start)

time

HTTP 1.0 Message Flow (cont.)

4. http server closes TCP connection.

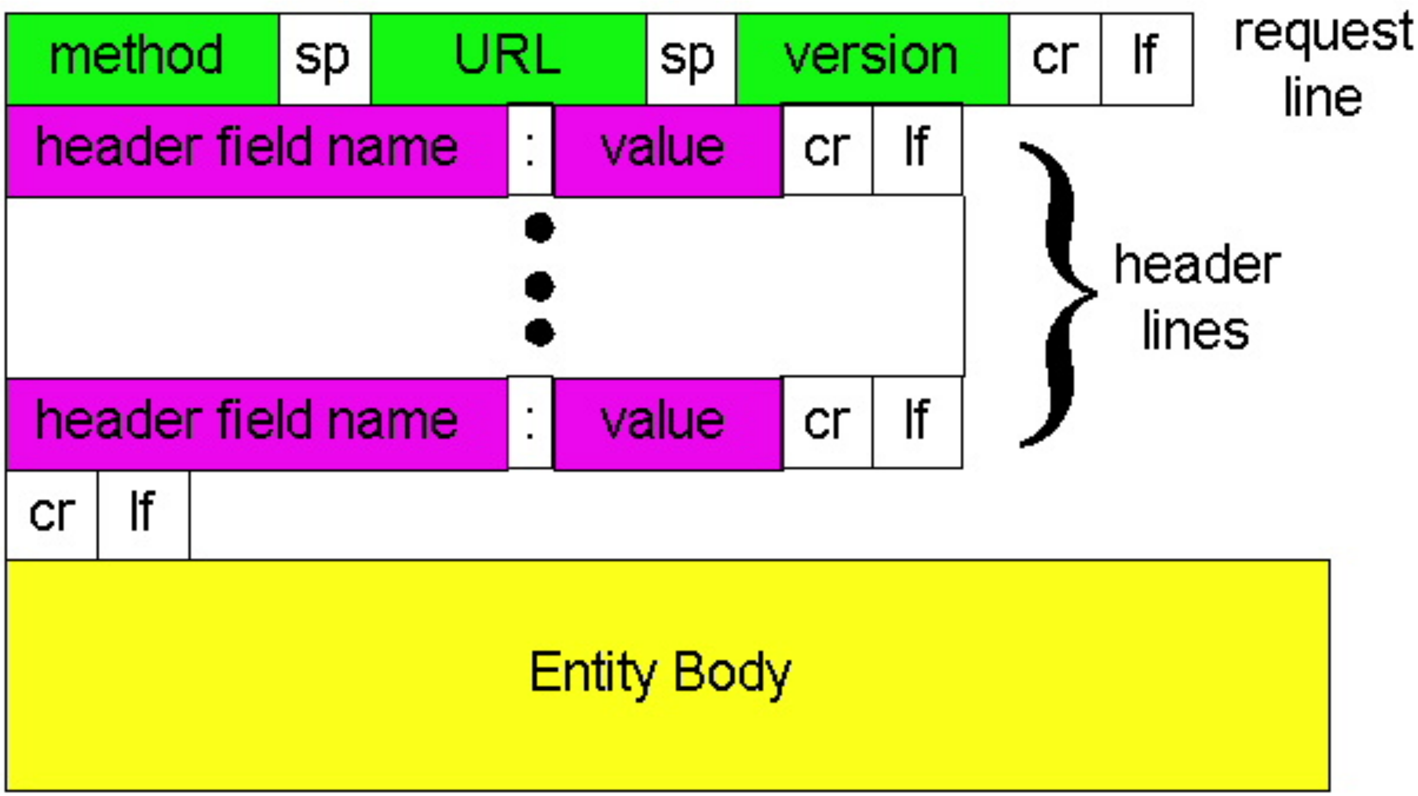
5. http client receives response message containing html file, parses html file, finds embedded image

time

6. Steps 1-5 repeated for each of the embedded images

HTTP Request Message: General Format

- ASCII (human-readable format)



HTTP Request Message Example: GET

request line (GET, POST, HEAD commands)

```
GET /somedir/page.html HTTP/1.0
Host: www.somechool.edu
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: en

```

header lines

Carriage return, line feed indicates end of message

(extra carriage return, line feed)

HTTP Response Message

status line
(protocol
status code
status phrase)

header
lines

data, e.g.,
requested
html file

```
HTTP/1.0 200 OK
Date: Wed, 23 Jan 2008 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 Response Status Codes

In the first line of the 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 Web server:

```
telnet www.sjtu.edu.cn 80
```

Opens TCP connection to port 80 (default http server port) at www.sjtu.edu.cn
Anything typed in sent to port 80 at www.sjtu.edu.cn

2. Type in a GET http request:

```
GET /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 the http server.

HTTP/1.0 Delay

- For each object:
  - TCP handshake --- 1 RTT
  - Client request and server responds --- at least 1 RTT (if object can be contained in one packet)

- Discussion: how to reduce delay?

HTTP Message Flow: Persistent HTTP

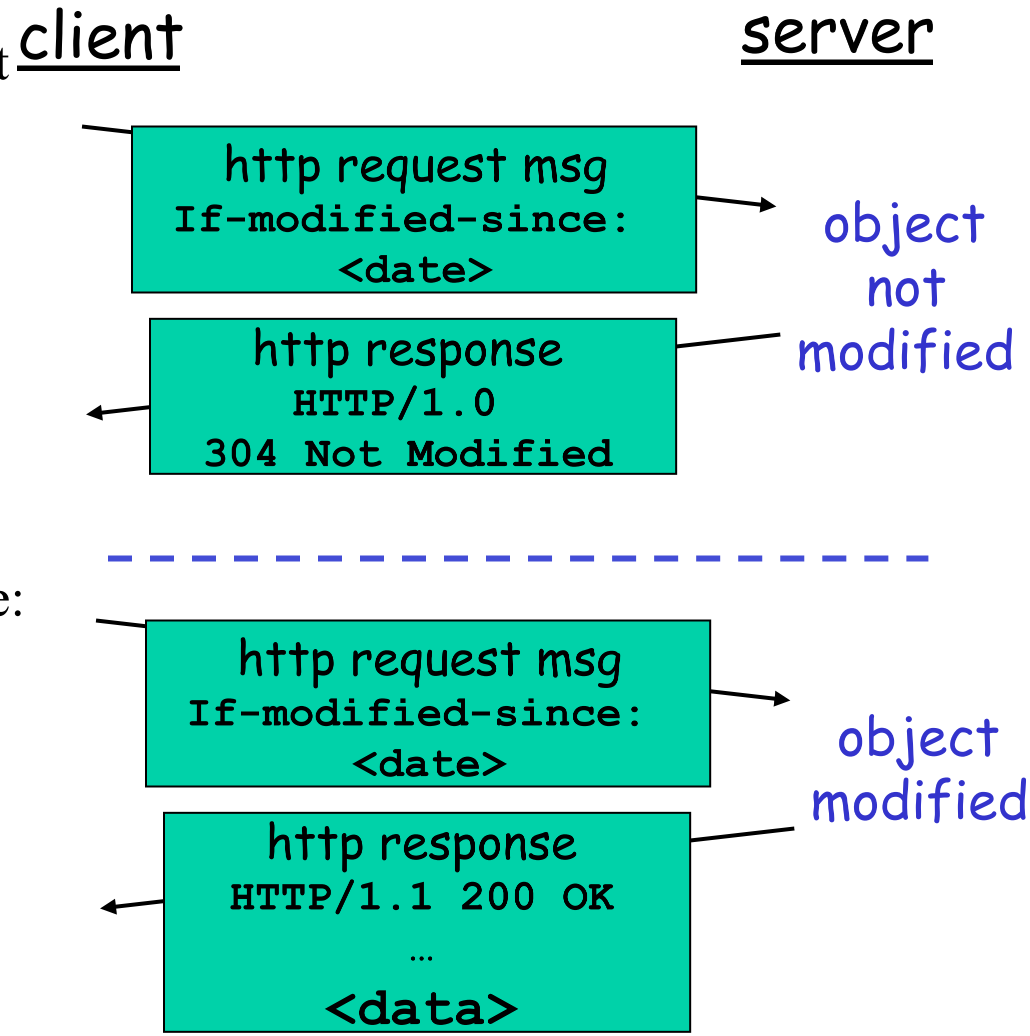
- Default for HTTP/1.1
- On same TCP connection: server parses request, responds, parses new request, …
- Client sends requests for all referenced objects as soon as it receives base HTML
- Fewer RTTs

Browser Cache and 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

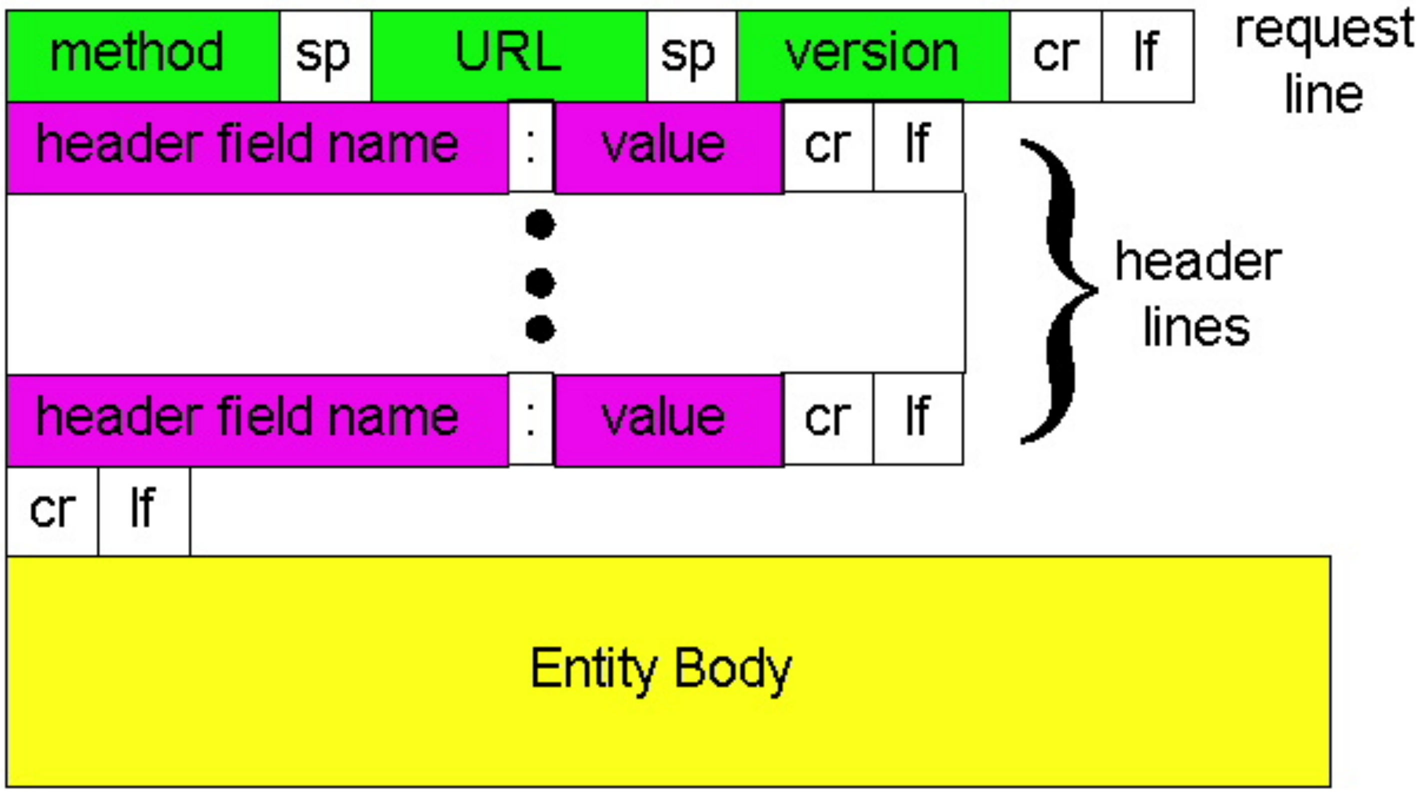
  `If-modified-since: <date>`
- server: response contains no object if cached copy up-to-date:

  `HTTP/1.0 304 Not Modified`



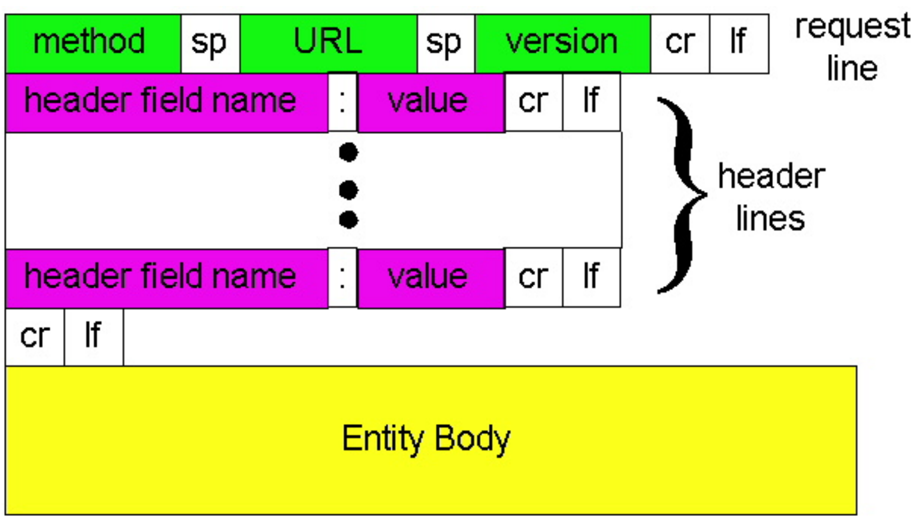
HTTP Message Extension: Form

- If an HTML page contains forms, they are encoded in message body



HTTP Message Flow Extensions: Keeping State

- Why do we need to keep state?
- In FTP, the server keeps the connection open with each client, and thus the state (e.g., current dir/ password). Why does't HTTP use this approach?



User-server Interaction: Cookies

Goal: no explicit application level session

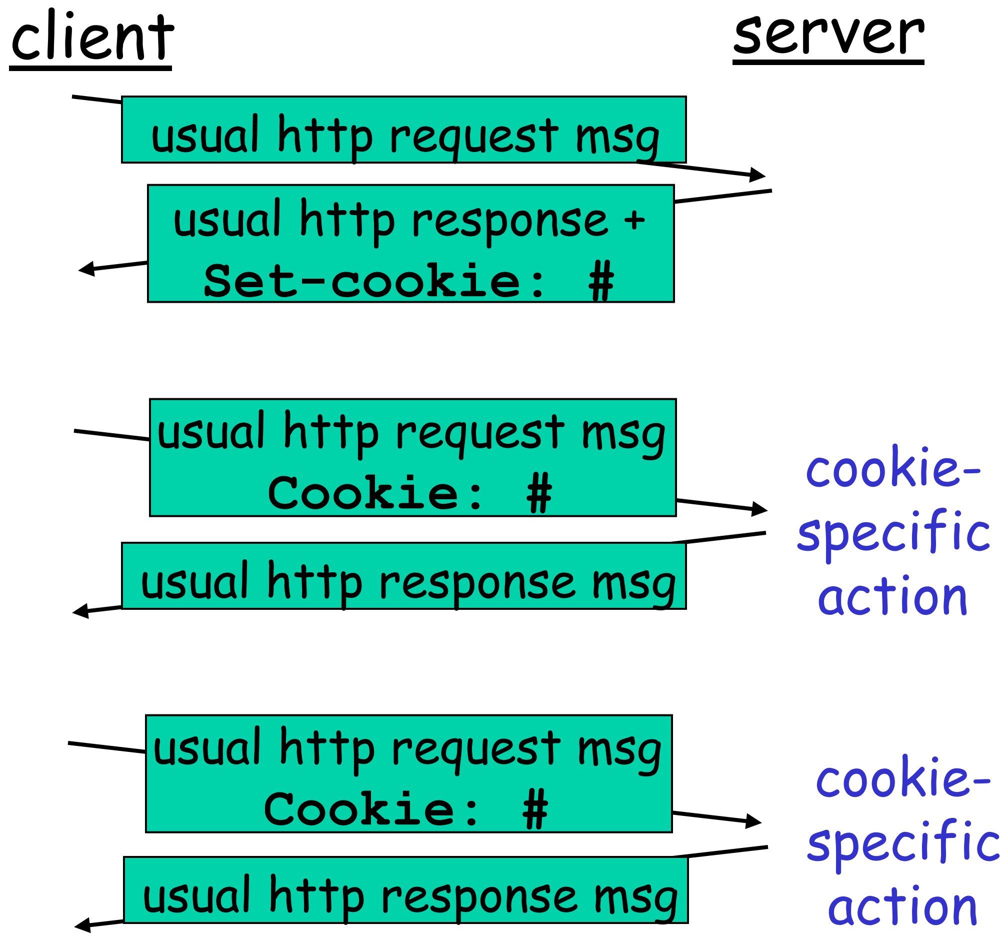
- Server sends "cookie" to client in response msg

  `Set-cookie: 1678453`

- Client presents cookie in later requests

  `Cookie: 1678453`

- Server matches presented-cookie with server-stored info
  - Authentication
  - Remembering user preferences, previous choices



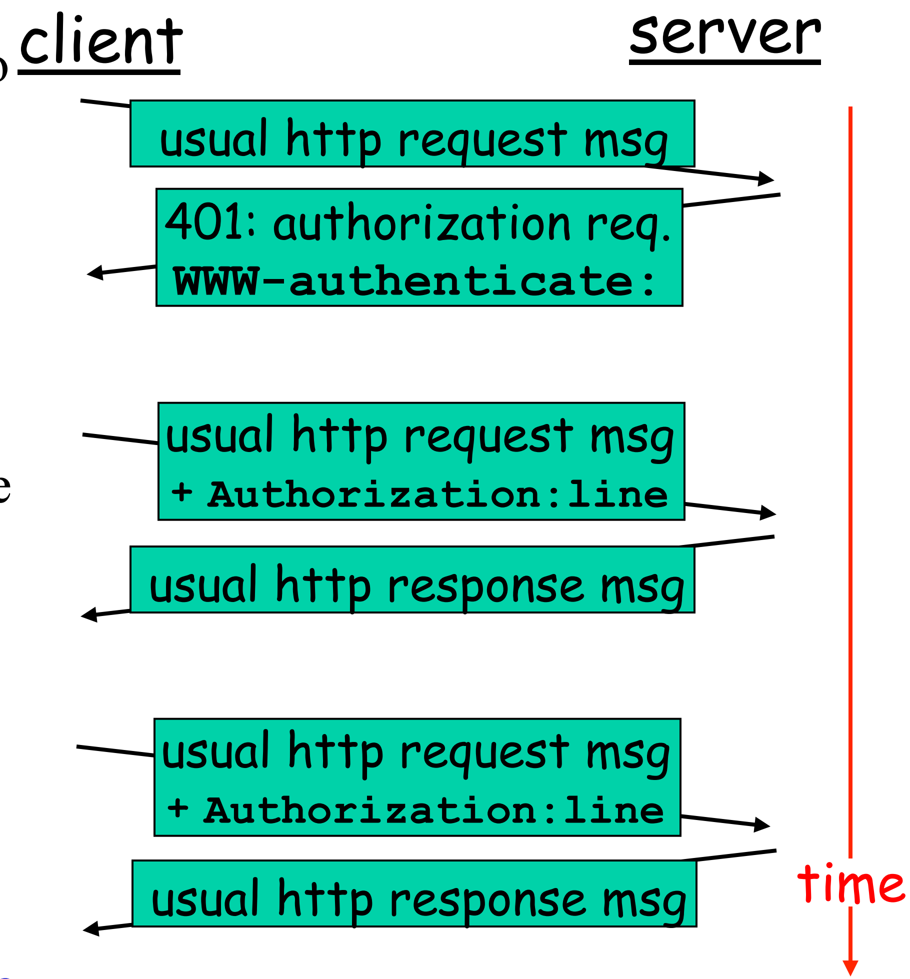
User-Server Interaction: Authentication

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:`

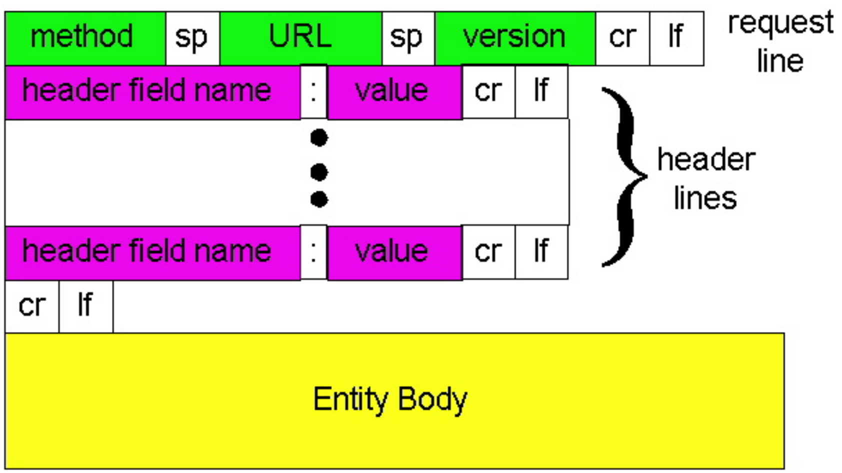
    header line in response



Browser caches name & password so that user does not have to repeatedly enter it.

Summary: HTTP

- How does a client locate a server?
- Is the application extensible, robust, scalable?



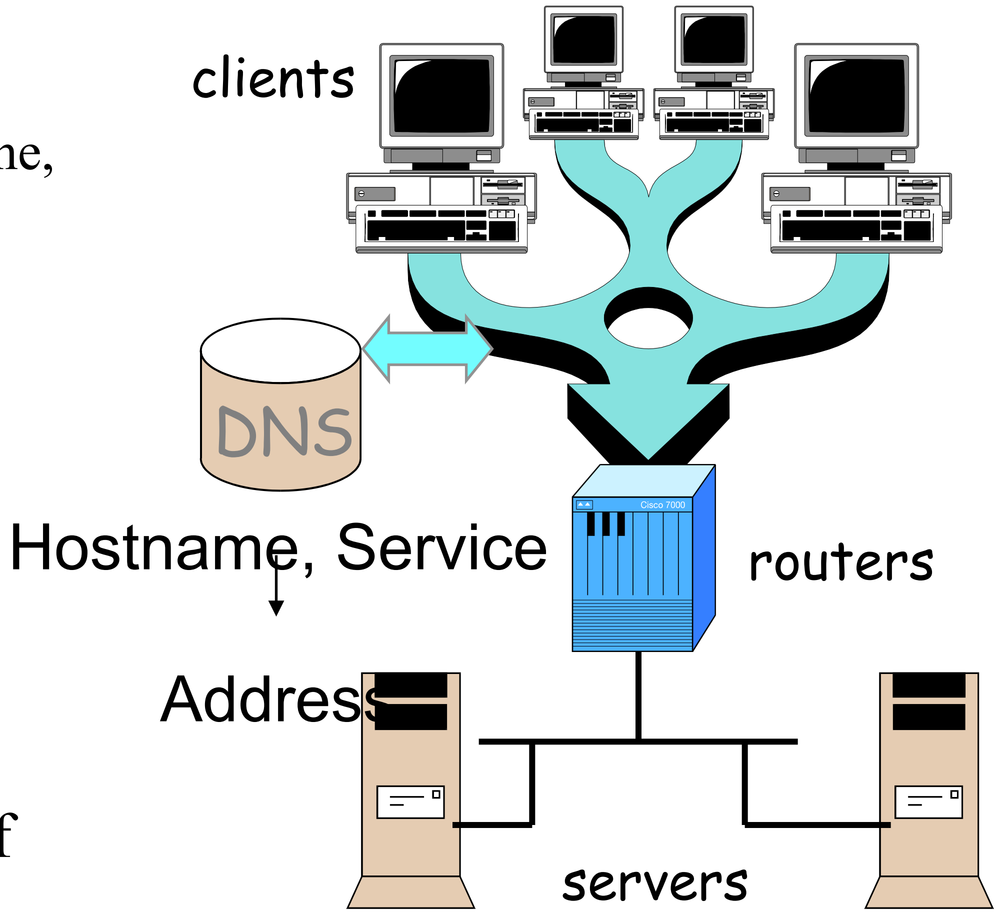
- HTTP message format
  - ASCII (human-readable format) requests, header lines, entity body, and responses line
- HTTP message flow
  - Stateless server
    - Each request is self-contained;
      thus cookie and authentication are needed in each message
  - Reducing latency
    - Persistent HTTP
      - The problem is introduced by layering !
    - Conditional GET reduces server/network workload and latency
    - Cache and proxy reduce traffic and latency

Outline

- Applications example
  - FTP
  - HTTP
  - *DNS*

DNS: Domain Name System

- Function
  - Map between (domain name, service) to value, e.g.,
    - (www.sjtu.edu.cn, Addr) -> 202.120.2.102
    - (cs.yale.edu, Email) -> netra.cs.yale.edu
    - (netra.cs.yale.edu, Addr) -> 128.36.229.21
  - gethostbyname()
- Why use name, instead of IP address directly?

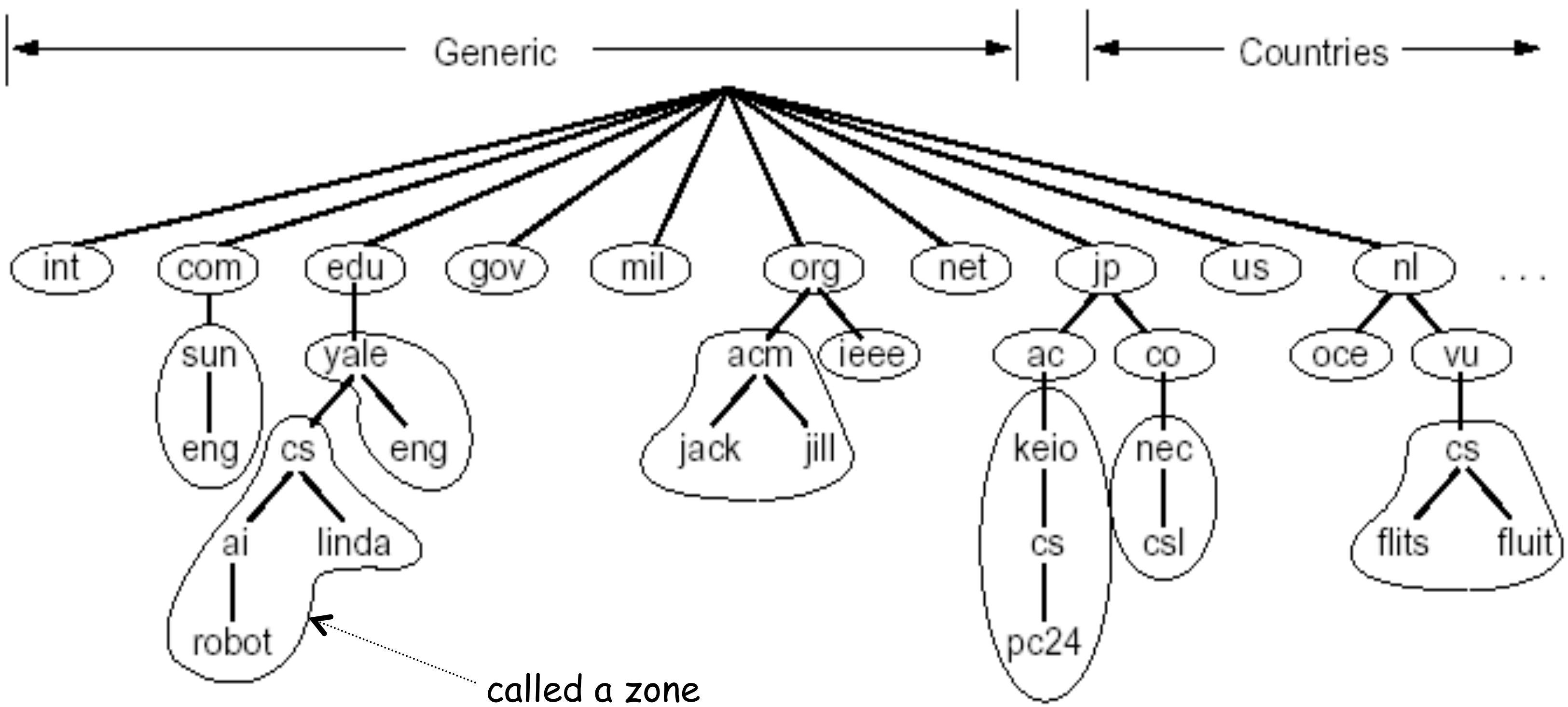


Translate Machine Names to IPs

- /etc/hosts
  - OK for small networks
  - Not scalable, up-to-date
  - Conflicts
- DNS
  - Solve the above problems

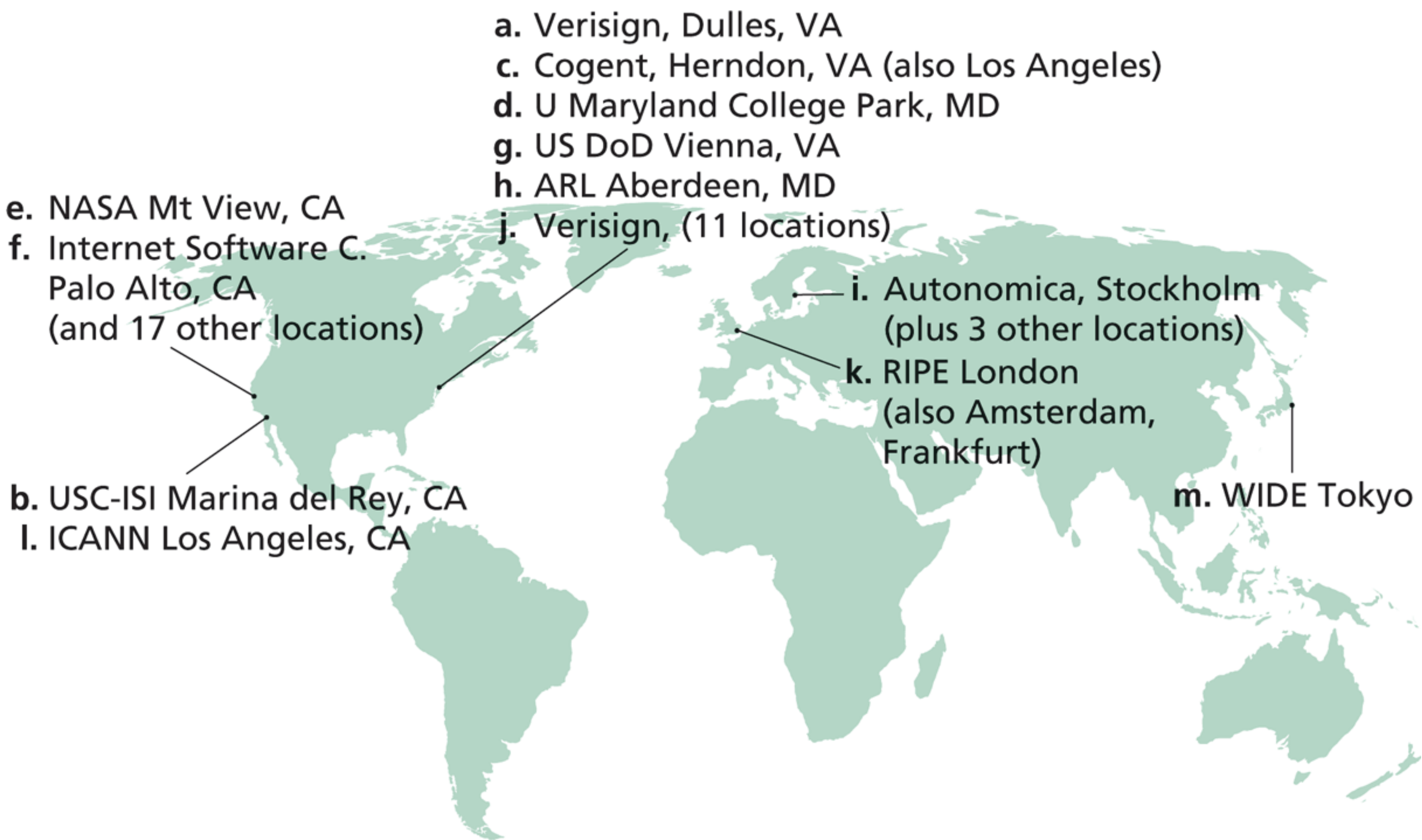
DNS: Domain Name System

- A distributed database managed by authoritative name servers
  - Each zone has its own **authoritative name servers**
  - An authoritative name server of a zone may delegate a subset (i.e. a sub-tree) of its zone to another name server



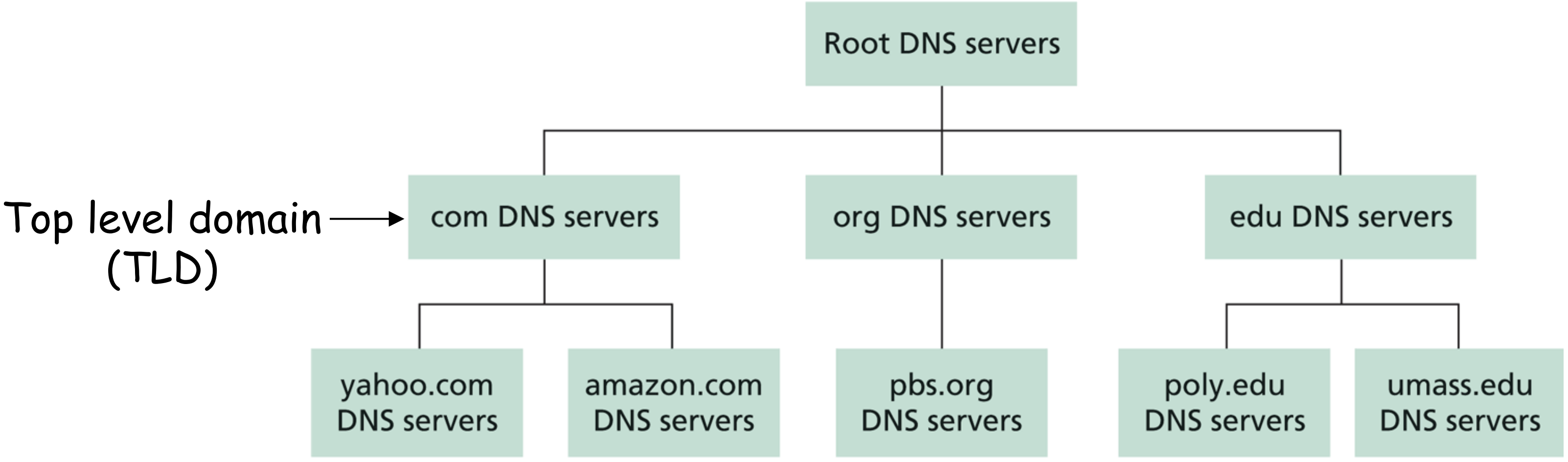
Root Zone and Root Servers

- The root zone is managed by the root name servers
  - 13 root name servers worldwide



Linking the Name Servers

- Each name server knows the addresses of the root servers
- Each name server knows the addresses of its immediate children (i.e., those it delegates)



DNS Message Flow: Two Types of Queries

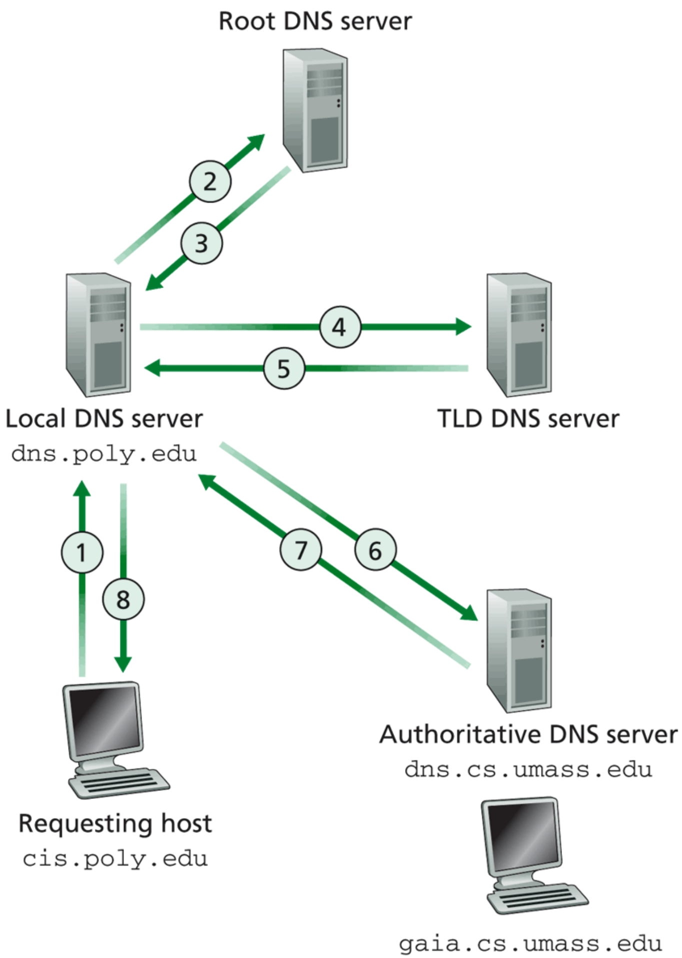
Recursive query:

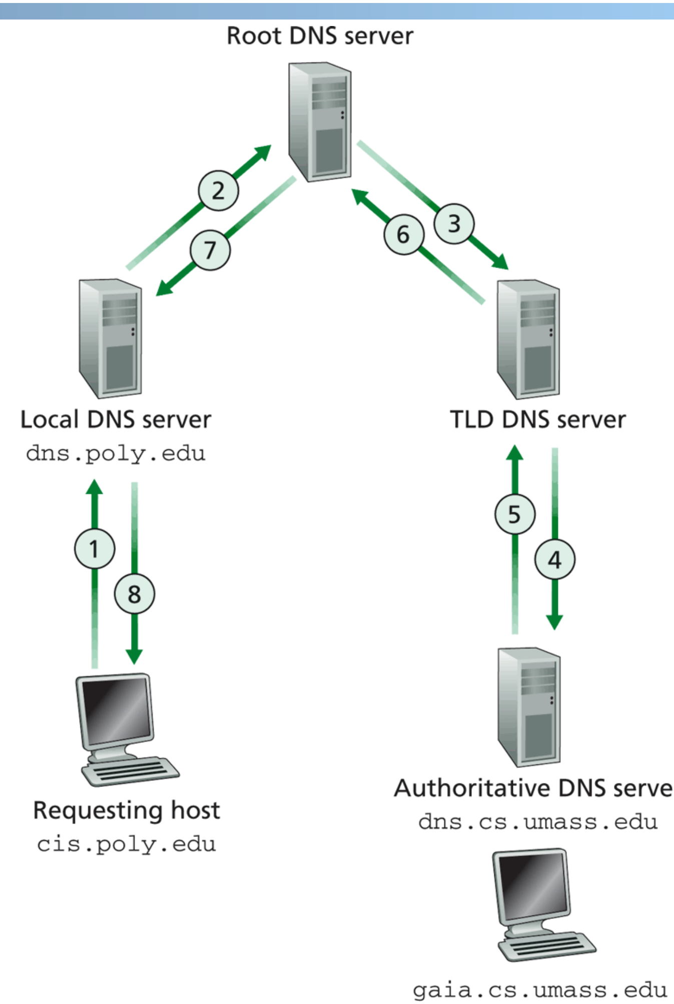
- Puts burden of name resolution on contacted name server
  - The contacted name server resolves the name completely

Iterated query:

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

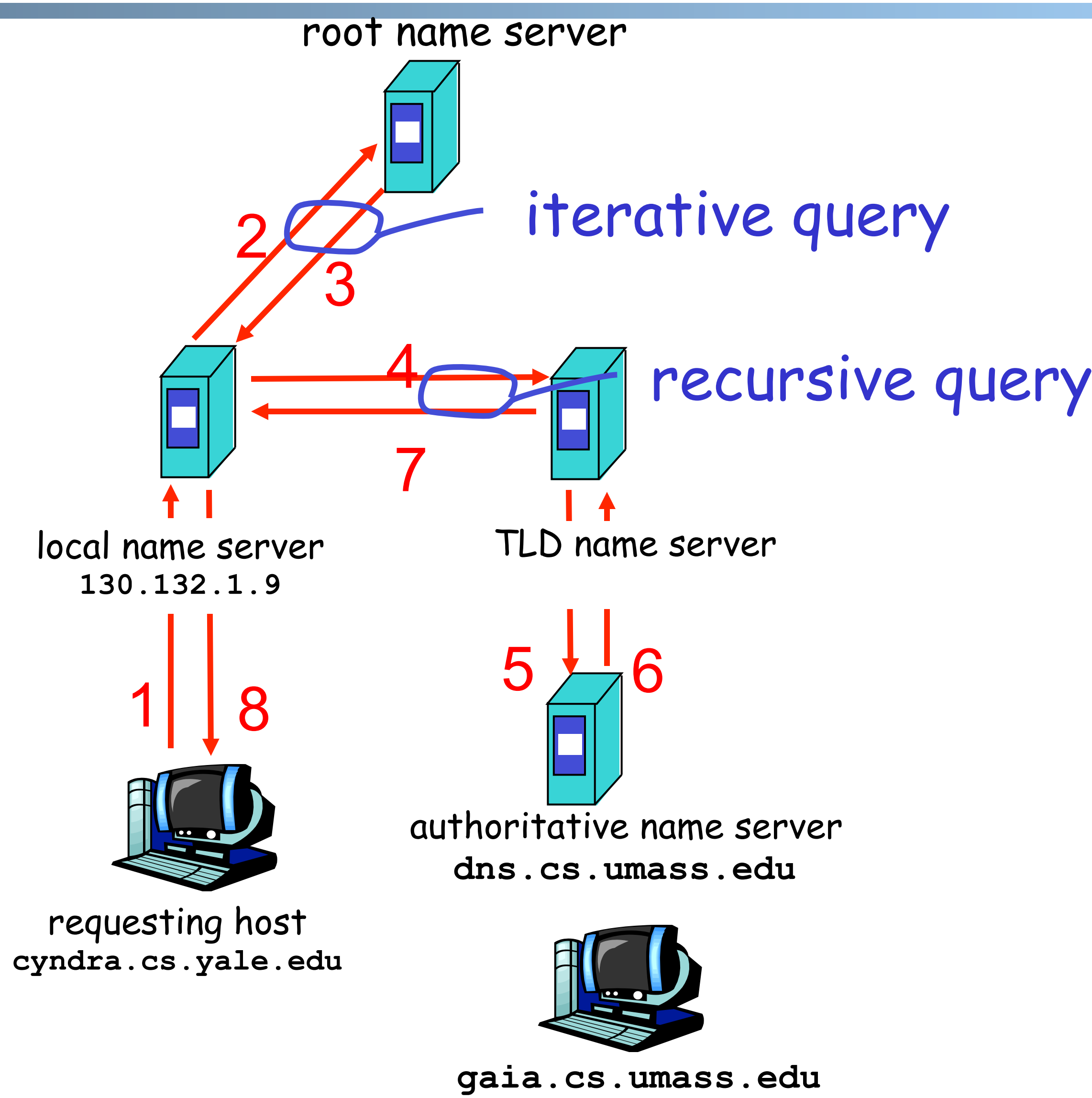
DNS Message Flow: Examples





Local DNS server helps requesting hosts to query the DNS system
Local DNS server is learned from DHCP, or configured, e.g. /etc/resolv.conf

DNS Message Flow: The Hybrid Case



DNS Records

DNS: distributed db storing resource records (RR)

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

- Type=A
  - **name** is hostname
  - **value** is IP address
- Type=NS
  - **name** is domain (e.g. yale.edu)
  - **value** is the name of the authoritative name server for this domain
- Type=CNAME
  - **name** is an alias name for some "canonical" (the real) name
  - **value** is canonical name
- Type=MX
  - **value** is hostname of mail server associated with **name**
- Type=SRV
  - general extension

A Sample DNS Database

; Authoritative data for cs.vu.nl

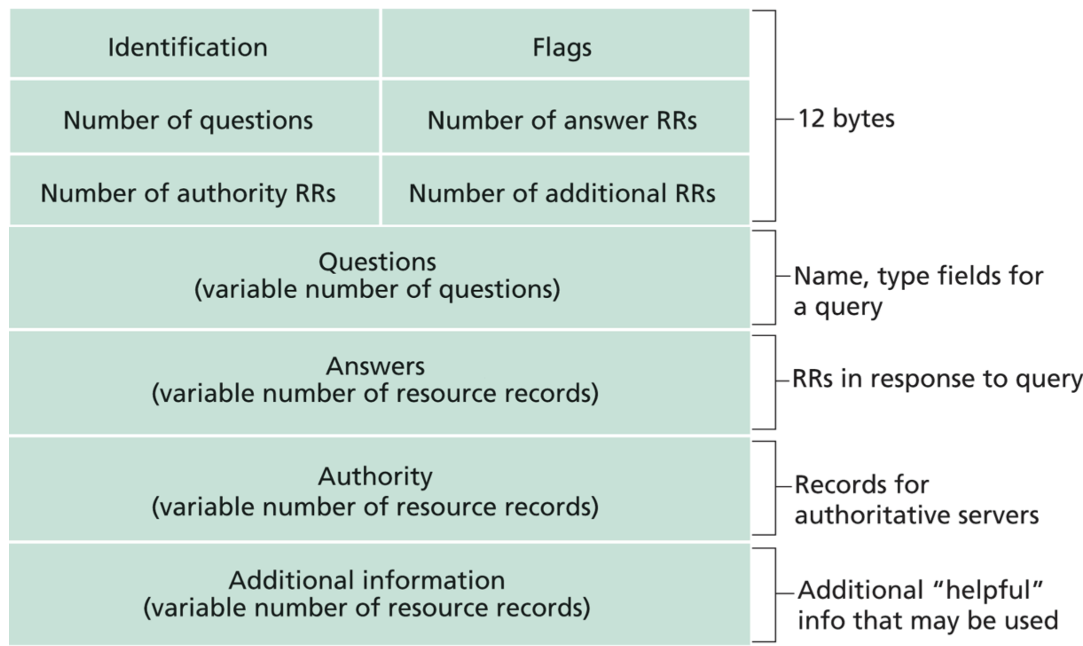
cs.vu.nl.	86400	IN	SOA	star boss (952771,7200,7200,2419200,86400)
cs.vu.nl.	86400	IN	TXT	"Divisie Wiskunde en Informatica."
cs.vu.nl.	86400	IN	TXT	"Vrije Universiteit Amsterdam."
cs.vu.nl.	86400	IN	MX	1 zephyr.cs.vu.nl.
cs.vu.nl.	86400	IN	MX	2 top.cs.vu.nl.
flits.cs.vu.nl.	86400	IN	HINFO	Sun Unix
flits.cs.vu.nl.	86400	IN	A	130.37.16.112
flits.cs.vu.nl.	86400	IN	A	192.31.231.165
flits.cs.vu.nl.	86400	IN	MX	1 flits.cs.vu.nl.
flits.cs.vu.nl.	86400	IN	MX	2 zephyr.cs.vu.nl.
flits.cs.vu.nl.	86400	IN	MX	3 top.cs.vu.nl.
www.cs.vu.nl.	86400	IN	CNAME	star.cs.vu.nl
ftp.cs.vu.nl.	86400	IN	CNAME	zephyr.cs.vu.nl
rowboat		IN	A	130.37.56.201
		IN	MX	1 rowboat
		IN	MX	2 zephyr
		IN	HINFO	Sun Unix
little-sister		IN	A	130.37.62.23
		IN	HINFO	Mac MacOS
laserjet		IN	A	192.31.231.216
		IN	HINFO	"HP Laserjet IIISi" Proprietary

DNS Protocol, Messages

DNS protocol : over UDP/TCP; *query* and *reply* messages, both with the same *message format*

DNS Msg header:

- identification: 16 bit # for query, the reply to a query uses the same #
- flags:
  - Query or reply
  - Recursion desired
  - Recursion available
  - Reply is authoritative



Observing DNS

- How does a client locate a server?
- Is the application extensible, robust, scalable?

- Use the command dig (or nslookup):
  - Force iterated query to see the trace:

    ```
    %dig +trace www.cnn.com
    ```

    - See the manual for more details

- Capture the messages using Ethereal
  - DNS server is at port 53

What DNS did Right?

- Hierarchical delegation avoids central control, improving manageability and scalability

- Redundant servers improve robustness
  - See http://www.internetnews.com/dev-news/article.php/1486981 for DDoS attack on root servers in Oct. 2002 (9 of the 13 root servers were crippled, but only slowed the network)
  - See http://www.cymru.com/DNS/index.html for performance monitoring

- Caching reduces workload and improve robustness

Problems of DNS

- Domain names may not be the best way to name other resources, e.g. files
- Relatively static resource types make it hard to introduce new services or handle mobility
- Although theoretically you can update the values of the records, it is rarely enabled
- Simple query model make it hard to implement advanced query
- Early binding (separation of DNS query from application query) does not work well in mobile, dynamic environments
  - E.g., load balancing, locate the nearest printer