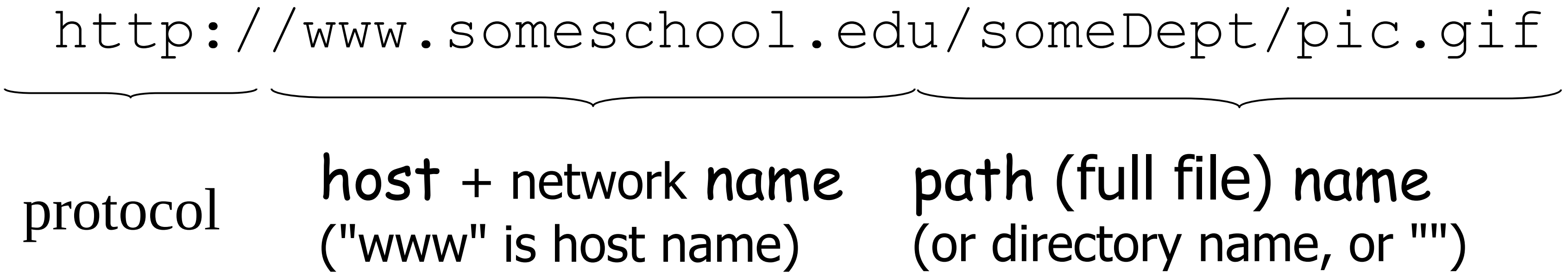
APPLICATION LAYER HTTP - FTP - TELNET

Web and HTTP

- 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
- Example URL:



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

protocol

host + network name ("www" is host name)

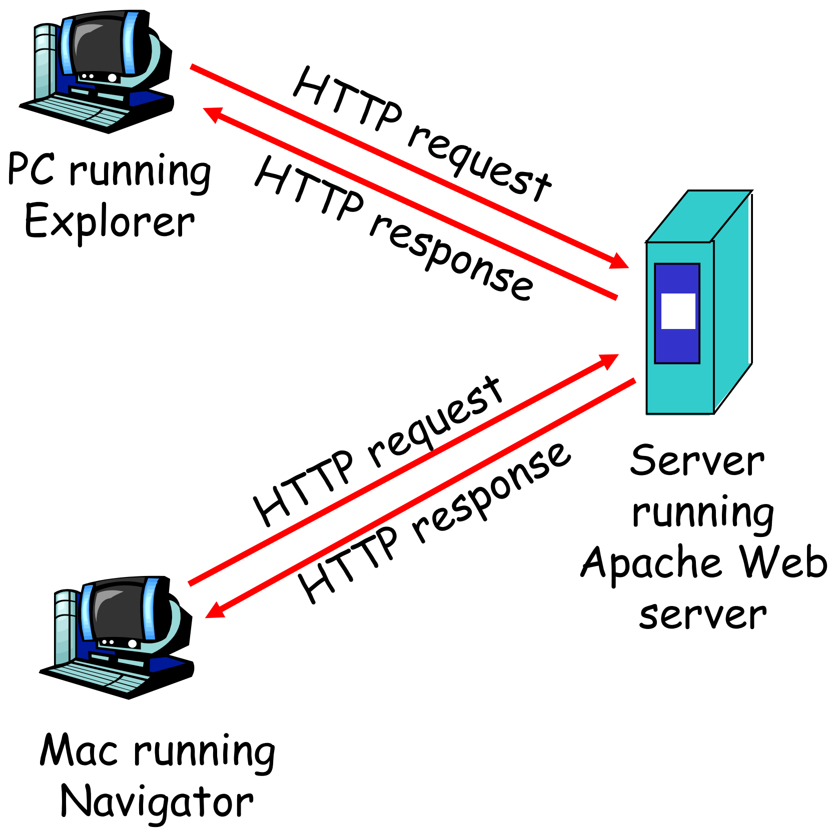
path (full file) name (or directory name, or "")

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- 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

- **RFC: specification**



HTTP Overview

Uses TCP:

- client initiates TCP connection (creates socket) to server, port 80 (default)
- 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

HTTP connection

Nonpersistent HTTP

- At most one object is sent over a TCP connection.
- HTTP/1.0 uses nonpersistent HTTP

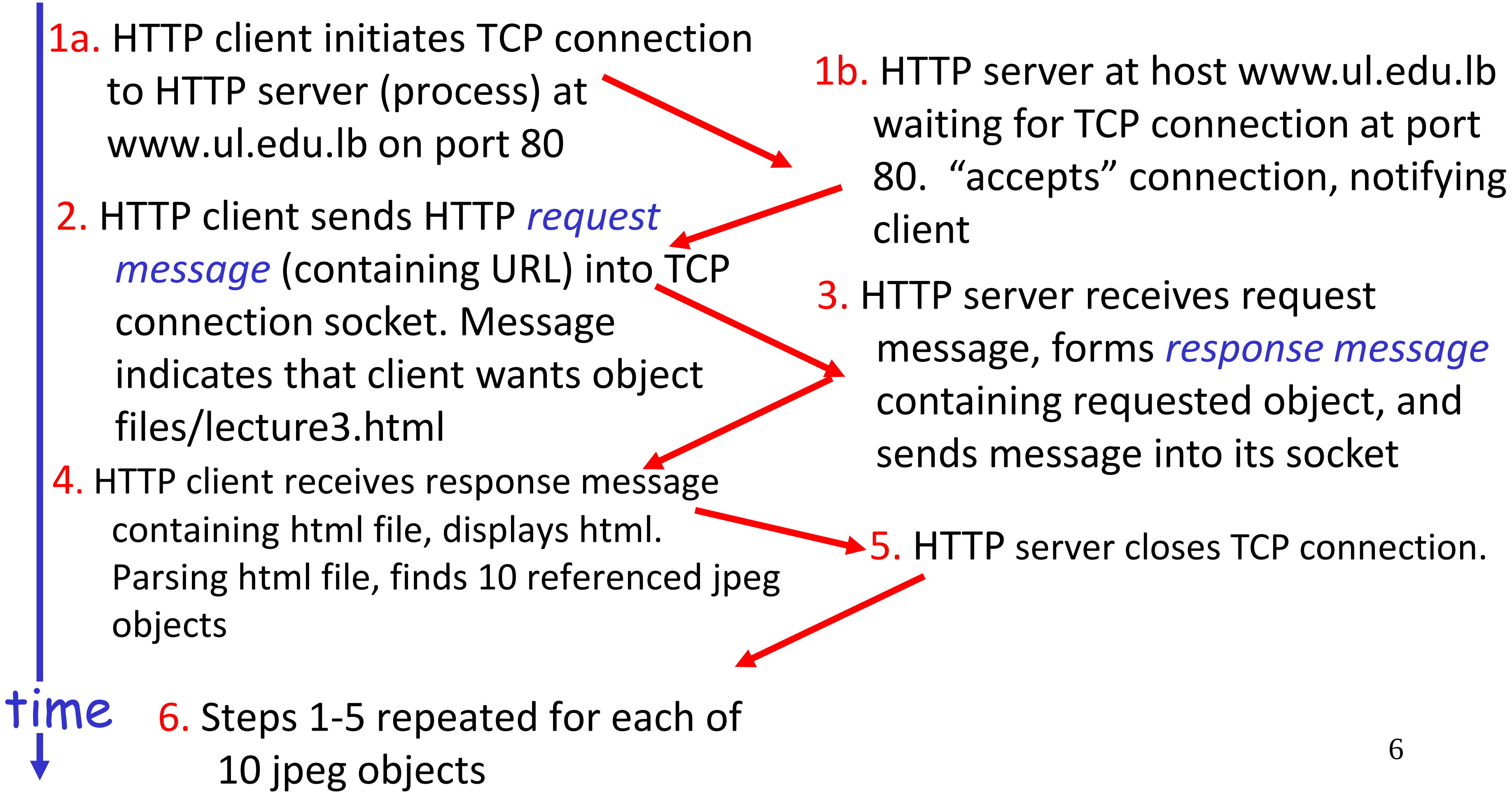
Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server.
- HTTP/1.1 uses persistent connections in default mode

Nonpersistent HTTP

- When you enter the following URL to your navigator:

www.ul.edu.lb/files/lecture3.html



1a. HTTP client initiates TCP connection to HTTP server (process) at www.ul.edu.lb on port 80

1b. HTTP server at host www.ul.edu.lb waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object files/lecture3.html

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

4. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

5. HTTP server closes TCP connection.

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

Non-persistent HTTP: Response time

Definition of RTT: time to send a small packet to travel from client to server and back.

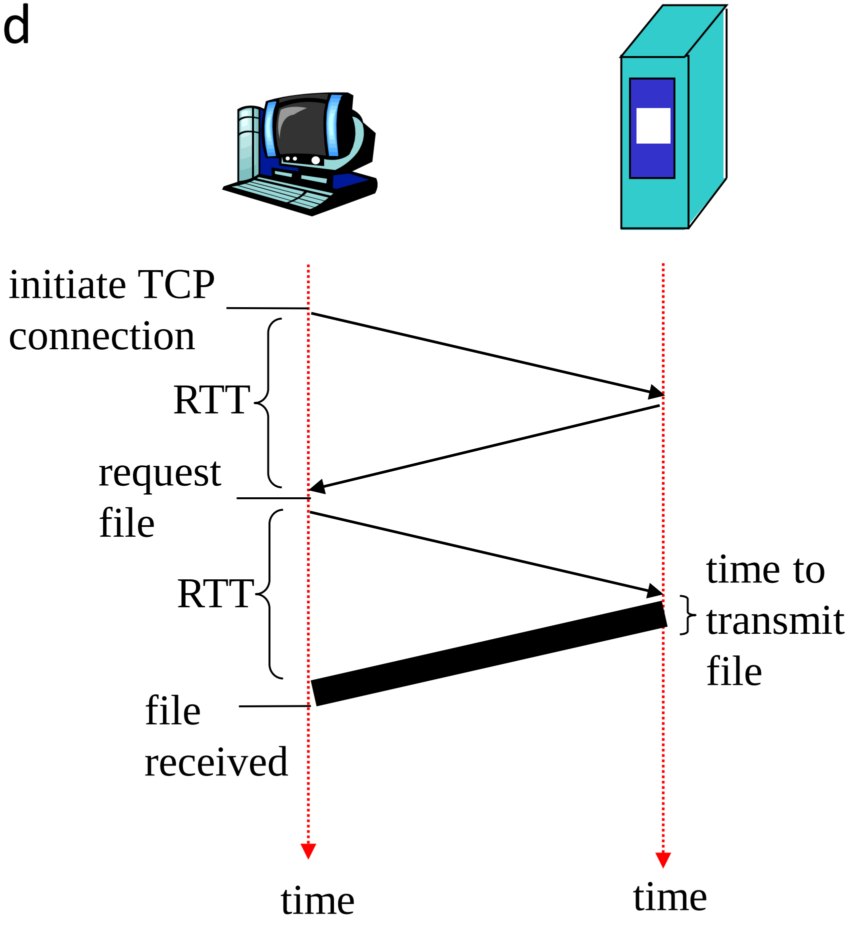
Response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time

total = 2RTT+transmit times

Issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects



Persistent HTTP

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection

Persistent ***without*** pipelining:

- client issues new request only when previous response has been received ➔ one RTT for each referenced object

Persistent ***with*** pipelining:

- default in HTTP/1.1
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

HTTP: method type

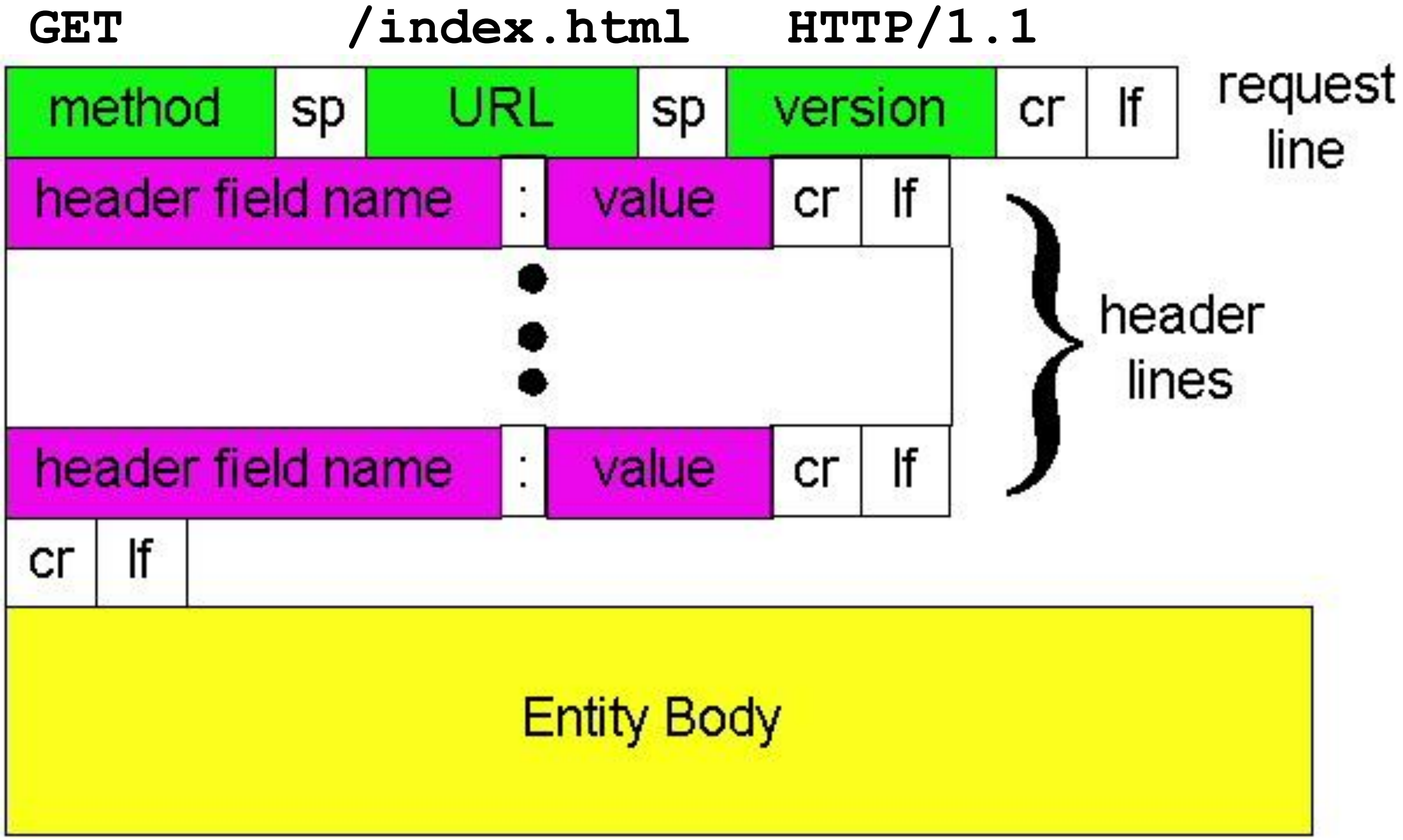
HTTP/1.0

- GET
- POST
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

HTTP request message: general form



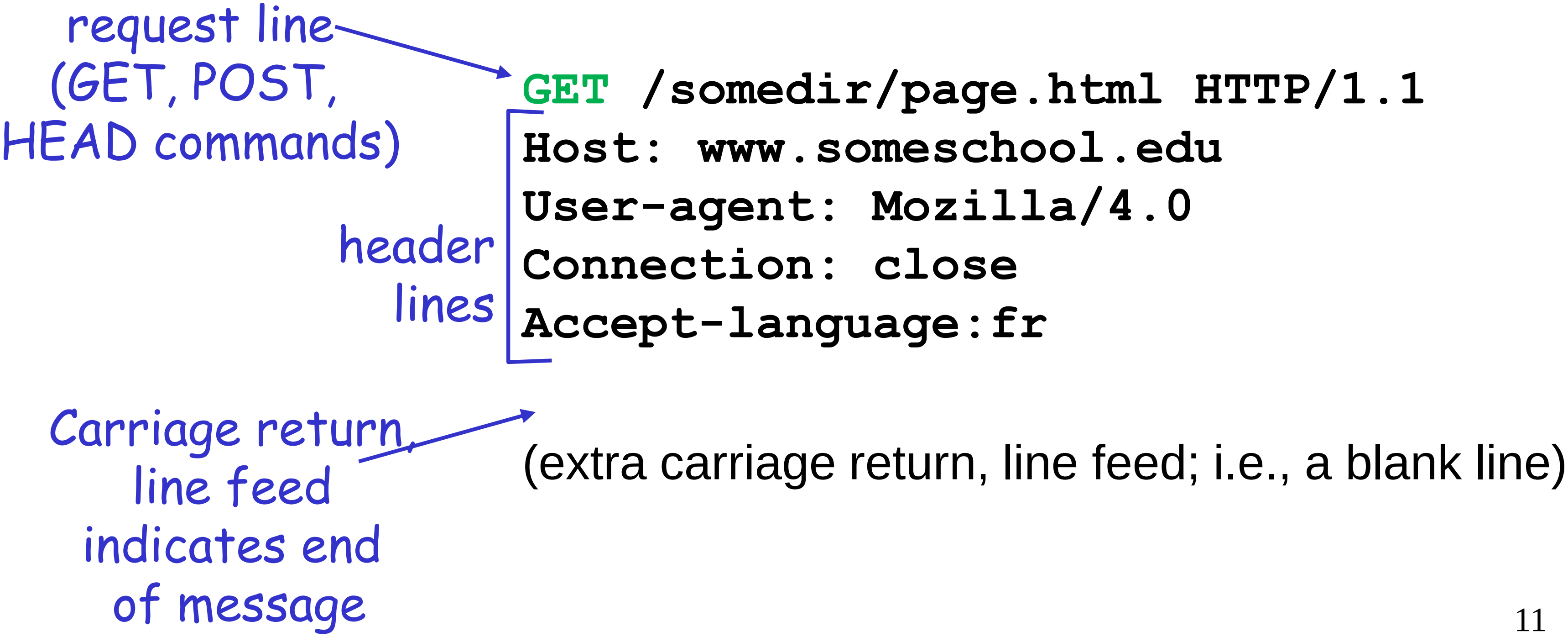
sp: separator can be a space

cr: carriage return

lf: line feed

Example: 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

Carriage return, line feed indicates end of message

```
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; i.e., a blank line)

Uploading form input

Post method:

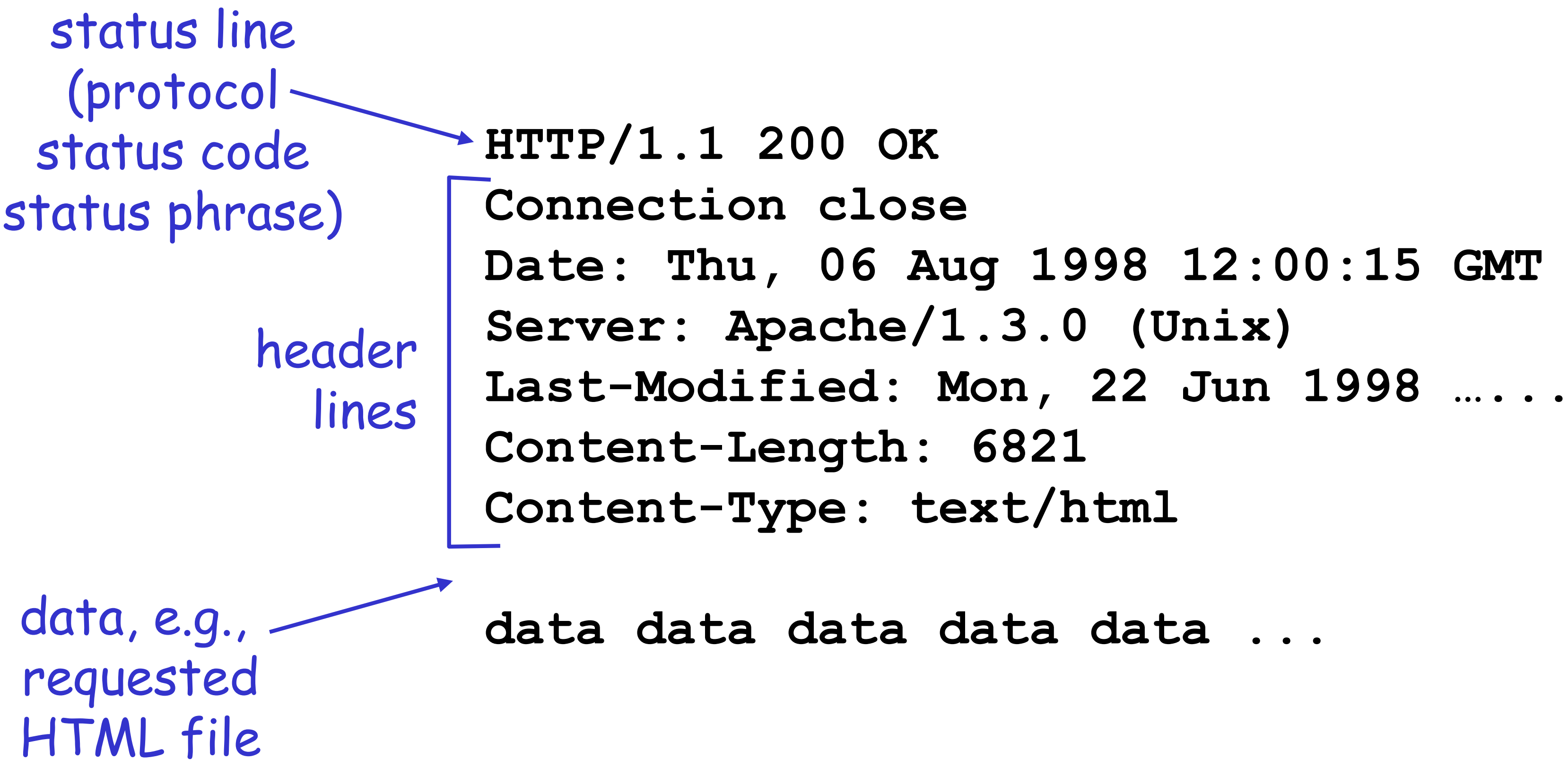
- Web page often includes form input
- Input is uploaded to server in entity body (when you click "Enter")

URL method:

- Uses GET method
- Input is uploaded
- in URL field of request line

www.somesite.com/animalsearch?monkeys&banana=3
? - start of request parameters
& - start of next parameter (name=value)

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

HTTP response 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

LAB: telnet

Telnet is a tool to connect to a remote computer (telnet /?)

telnet can be used to communicate with a web server.

Type telnet www.ul.edu.lb 80
(80 is used to specify a port number. HTTP uses port 80. By default Telnet uses port 23)

Try the GET command

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

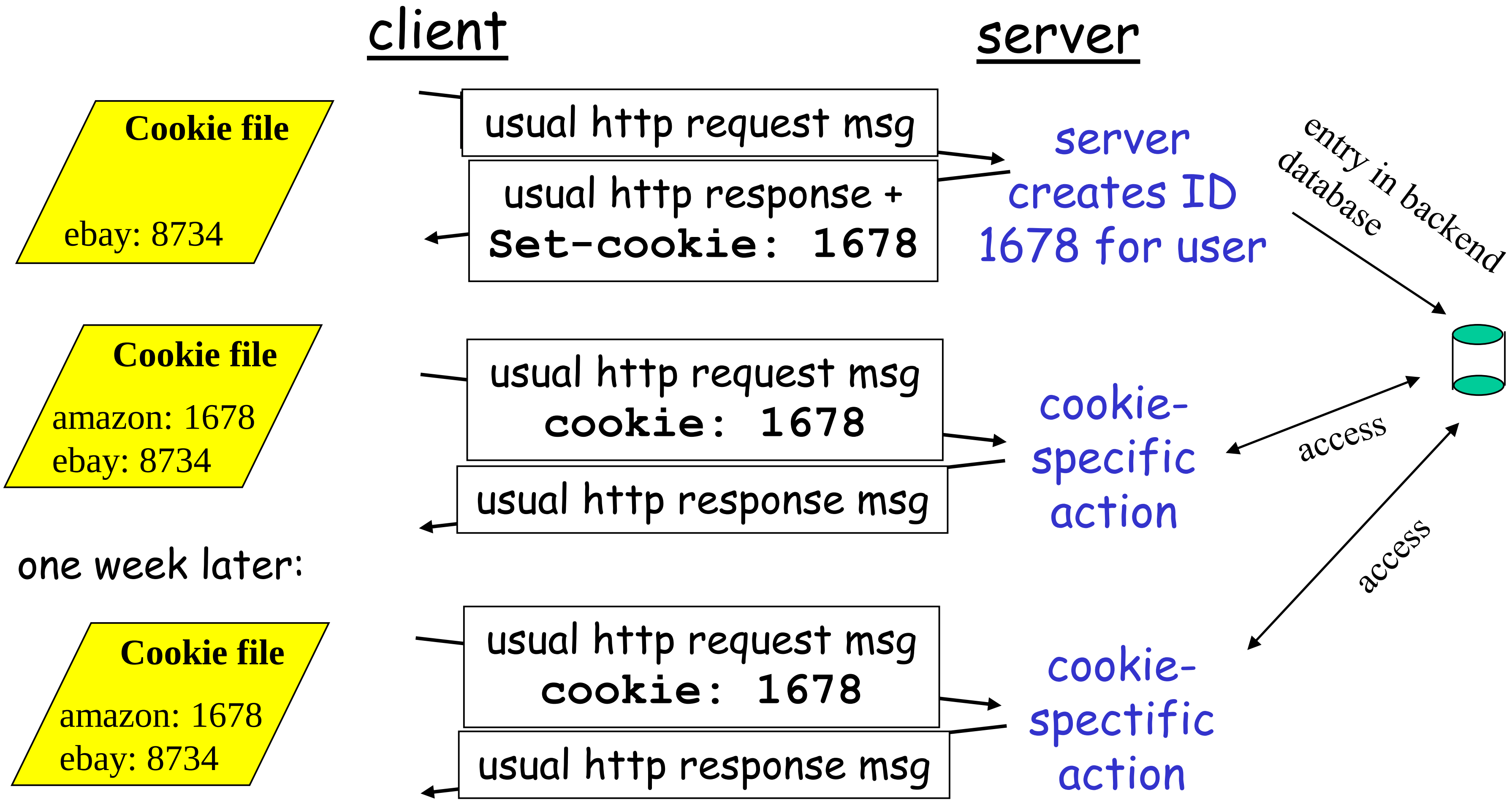
3) cookie file kept on user’s host, managed by user’s browser

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

Example: cookies



client

server

Cookie file

ebay: 8734

usual http request msg

usual http response +

`Set-cookie: 1678`

server creates ID 1678 for user

entry in backend database

Cookie file

amazon: 1678

ebay: 8734

usual http request msg

`cookie: 1678`

usual http response msg

cookie-specific action

access

one week later:

Cookie file

amazon: 1678

ebay: 8734

usual http request msg

`cookie: 1678`

usual http response msg

cookie-spectific action

access

LAB: cookies

Cookies are installed on your computer.
Under windows 7, cookies are in directory:
C:|Users|BH|AppData|Local|Microsoft|Windows|Temporary Internet Files

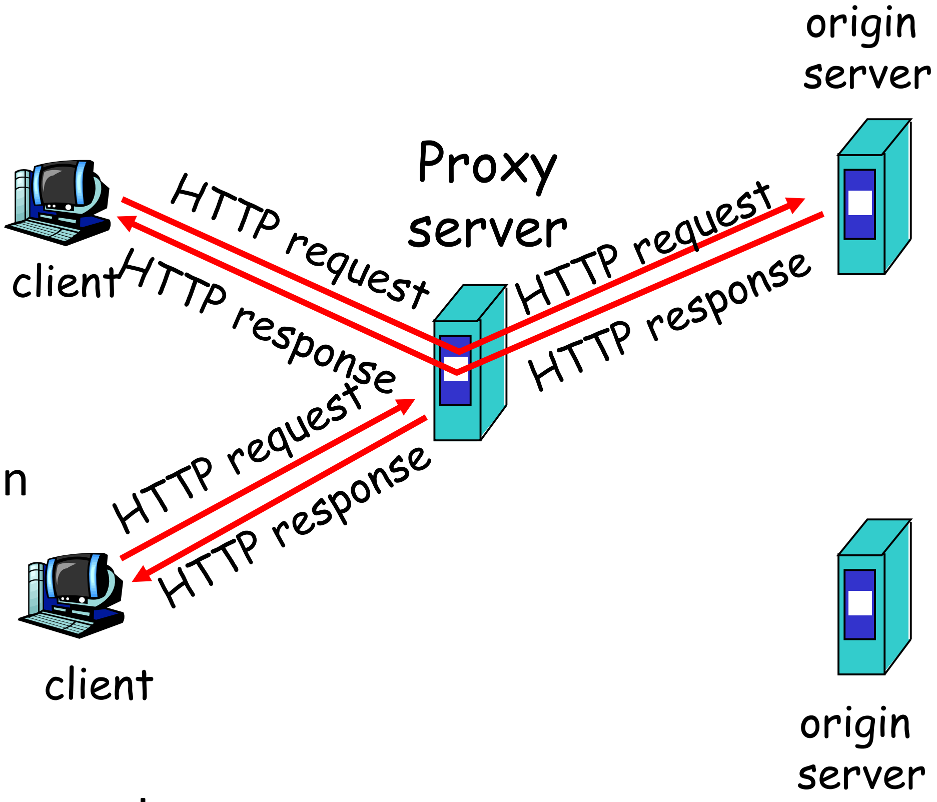
Connect to a web site (www.yahoo.com)

- Open and read the cookies.
- Modify them.
- Reconnect to the site.

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



Web caching:

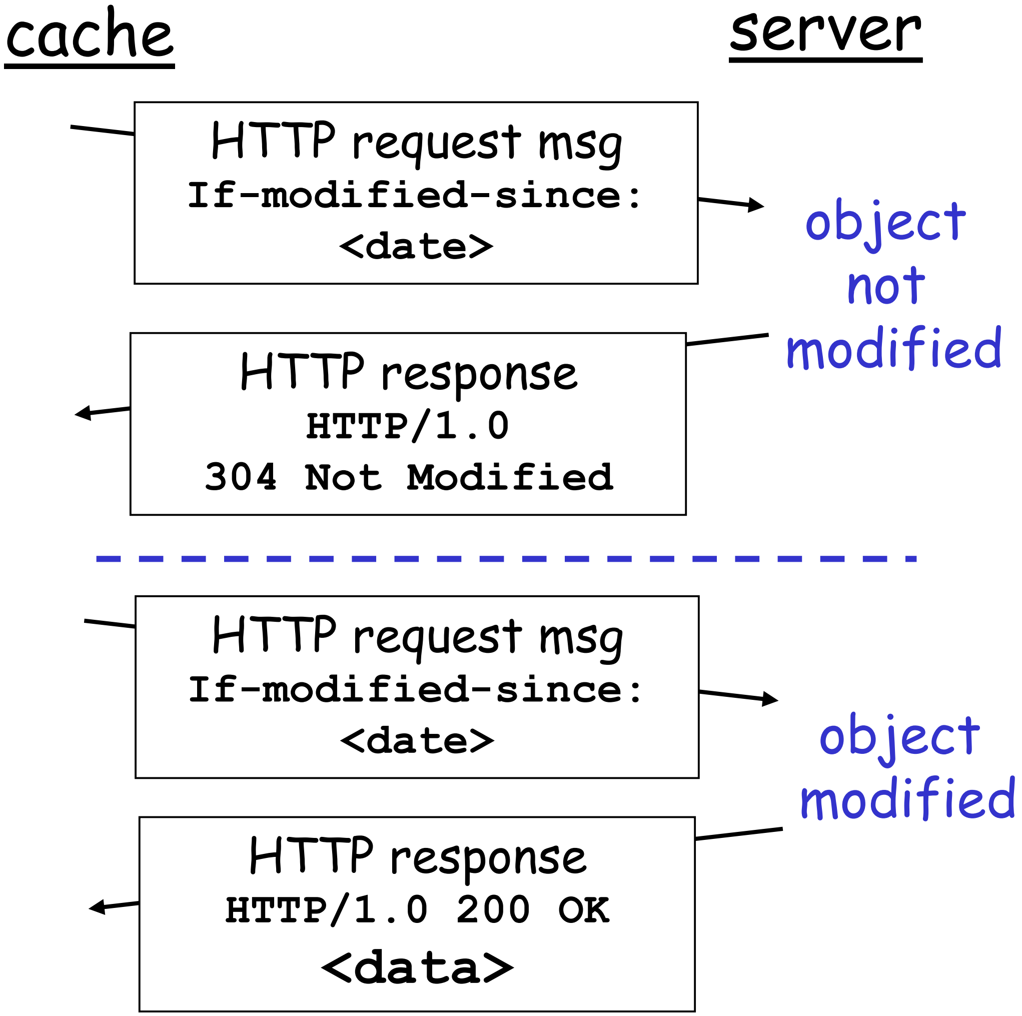
- Reduce response time for client request.
- Reduce traffic on an institution’s access link.
- Internet dense with caches: enables “poor” content providers to effectively deliver content (but so does P2P file sharing)

Conditional get (used with caching)

- Goal: don’t send object if cache has up-to-date cached version
- cache: specify date of cached copy in HTTP request

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

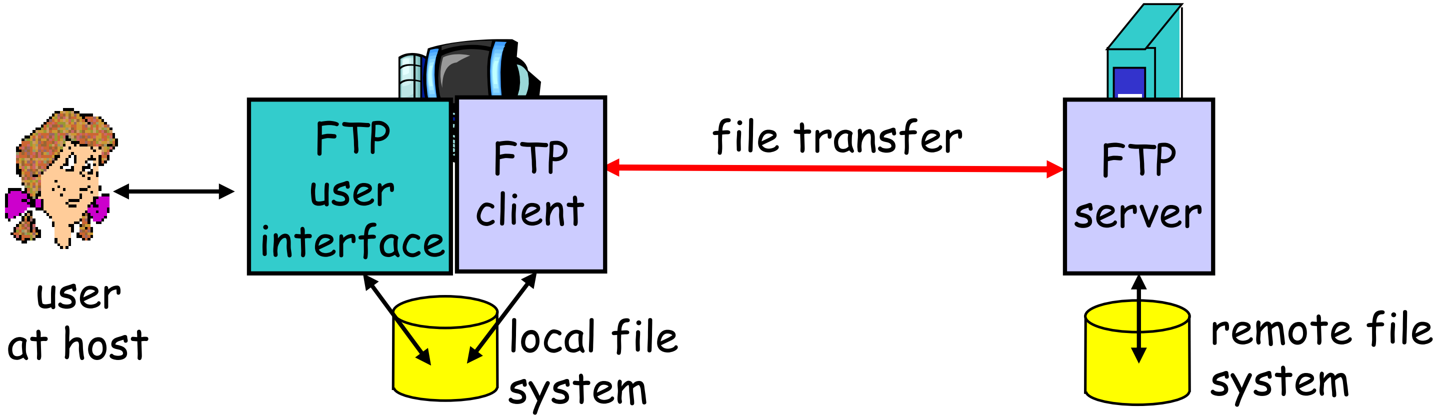
  `HTTP/1.0 304 Not Modified`



FTP

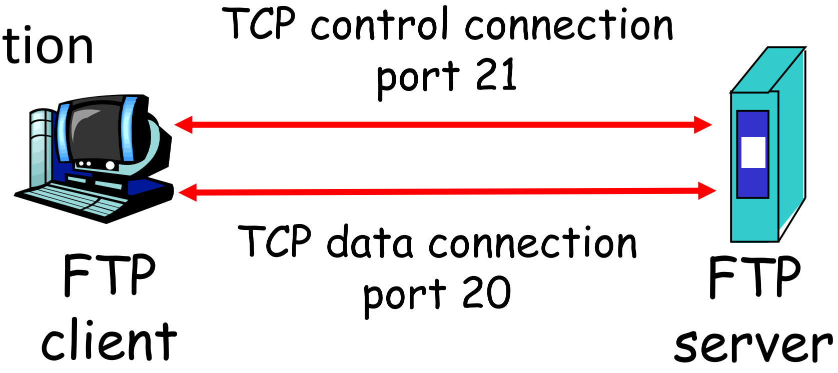
FTP: the file transfer protocol

- transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ftp: RFC 959
- ftp server: **port 21** (control)



FTP: two connections, control & data

- FTP client contacts FTP server at port 21, specifying TCP as transport protocol
- Client obtains authorization over control connection
- Client browses remote directory by sending commands over control connection.
- When server receives file transfer command, server opens *2nd* TCP connection (for file) to client
- After transferring one file, server closes data connection.
- Server opens another TCP data connection to transfer another file.
- Control connection: "out of band"
- FTP server maintains "state": current directory, earlier authentication



FTP: commands - responses

Sample commands:

- sent as ASCII text over control channel
- **USER *username***
- **PASS *password***
- **LIST** return list of file in current directory
- **RETR filename** retrieves (gets) file
- **STOR filename** stores (puts) file onto remote host

Sample return codes

status code and phrase (as in HTTP)

- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can’t open data connection**
- **452 Error writing file**

TELNET vs. telnet

- TELNET is a *protocol* that provides "a general, bi-directional, eight-bit byte oriented communications facility".
- **telnet** is a *program* that supports the TELNET protocol over TCP.
- Many application protocols are built upon the TELNET protocol.

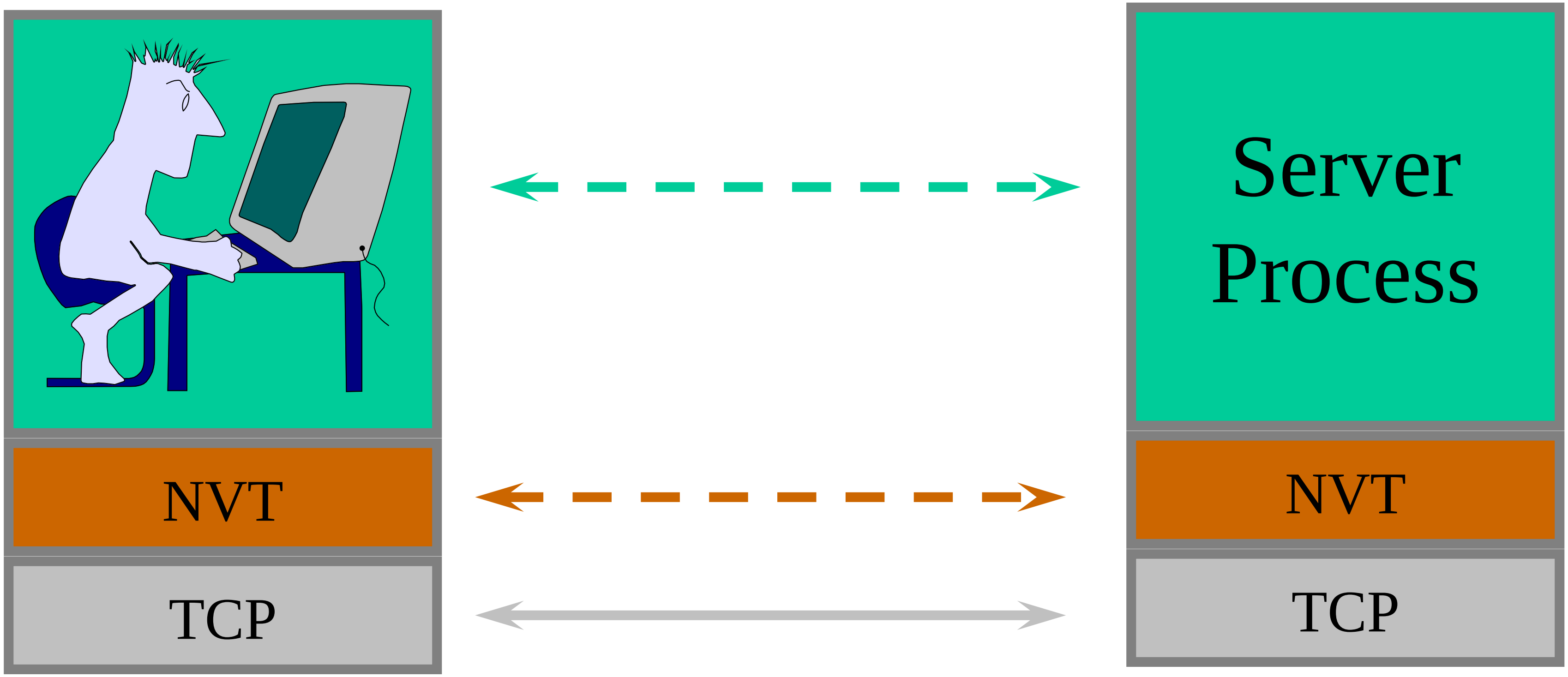
The TELNET Protocol

- Reference: RFC 854

- TCP connection
- data and control over the same connection.

- Network Virtual Terminal
  - intermediate representation of a generic terminal.
  - provides a standard language for communication of terminal control functions.

Network Virtual Terminal



Server Process

NVT

TCP

NVT

TCP

Playing with TELNET

- You can use the **telnet** program to play with the TELNET protocol.
- **telnet** is a *generic* TCP client.
  - Sends whatever you type to the TCP socket.
  - Prints whatever comes back through the TCP socket
  - Useful for testing TCP servers (ASCII based protocols).
- Many Unix systems have these servers running (by default):

echo	port 7	**discard**	port 9
daytime	port 13	**chargen**	port 19

DNS

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 - 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

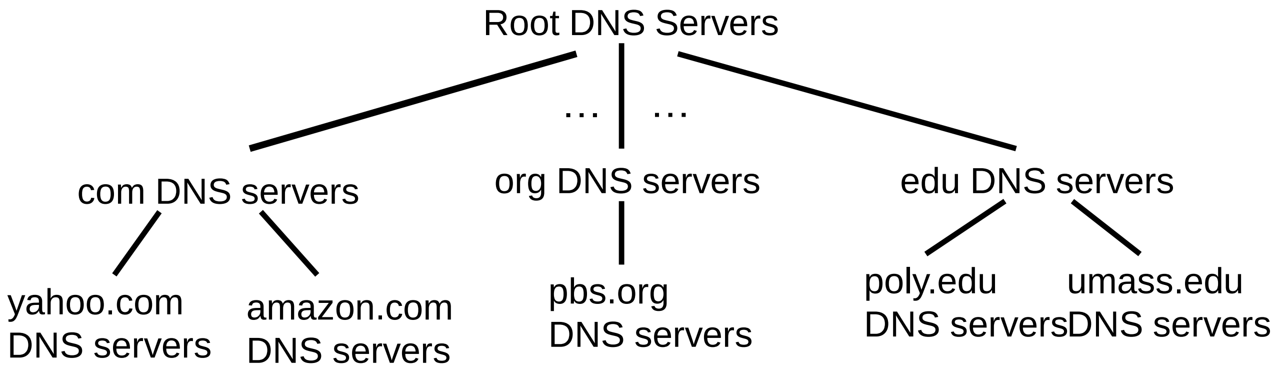
- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

why not centralize DNS?

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

A: doesn‘t scale!

DNS: a distributed, hierarchical database

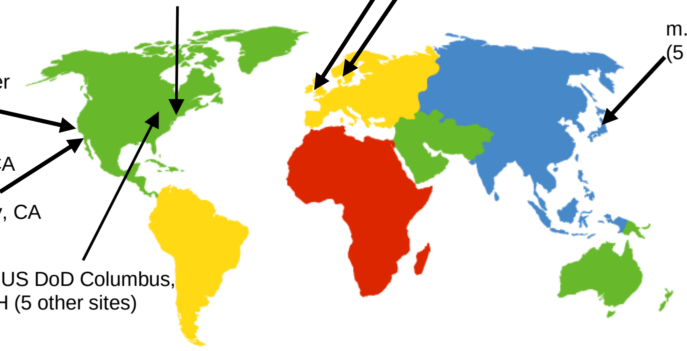


client wants IP for www.amazon.com; 1st approximation:

- 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



c. Cogent, Herndon, VA (5 other sites)
d. U Maryland College Park, MD
h. ARL Aberdeen, MD
j. Verisign, Dulles VA (69 other sites)

k. RIPE London (17 other sites)

i. Netnod, Stockholm (37 other sites)

m. WIDE Tokyo
(5 other sites)

e. NASA Mt View, CA
f. Internet Software C.
Palo Alto, CA (and 48 other
sites)

a. Verisign, Los Angeles CA
(5 other sites)
b. USC-ISI Marina del Rey, CA
l. ICANN Los Angeles, CA
(41 other sites)

g. US DoD Columbus,
OH (5 other sites)

13 logical root name “servers ” worldwide

- *each “server” replicated many times*

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

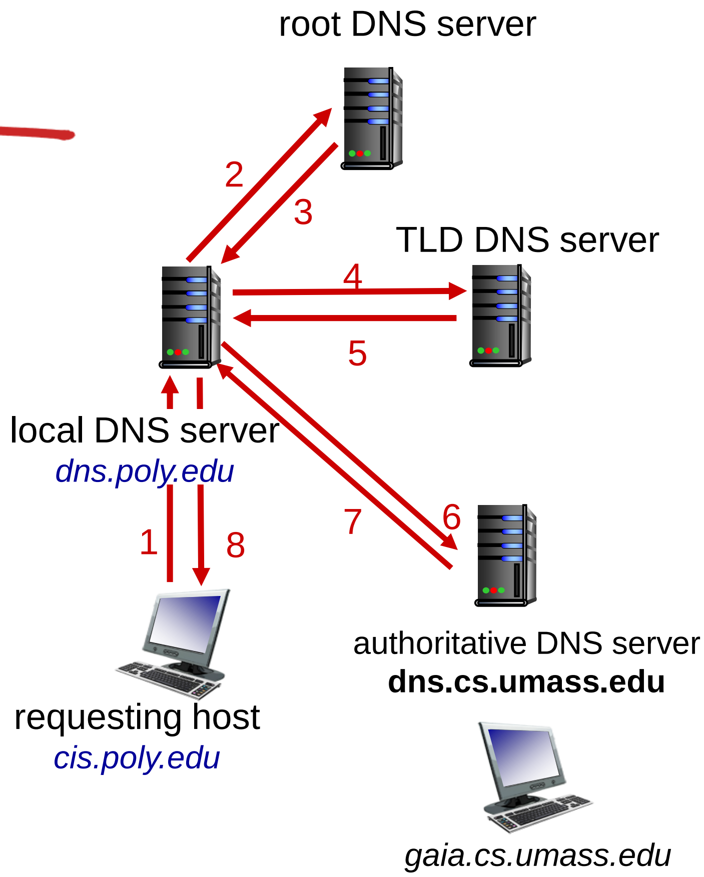
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts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

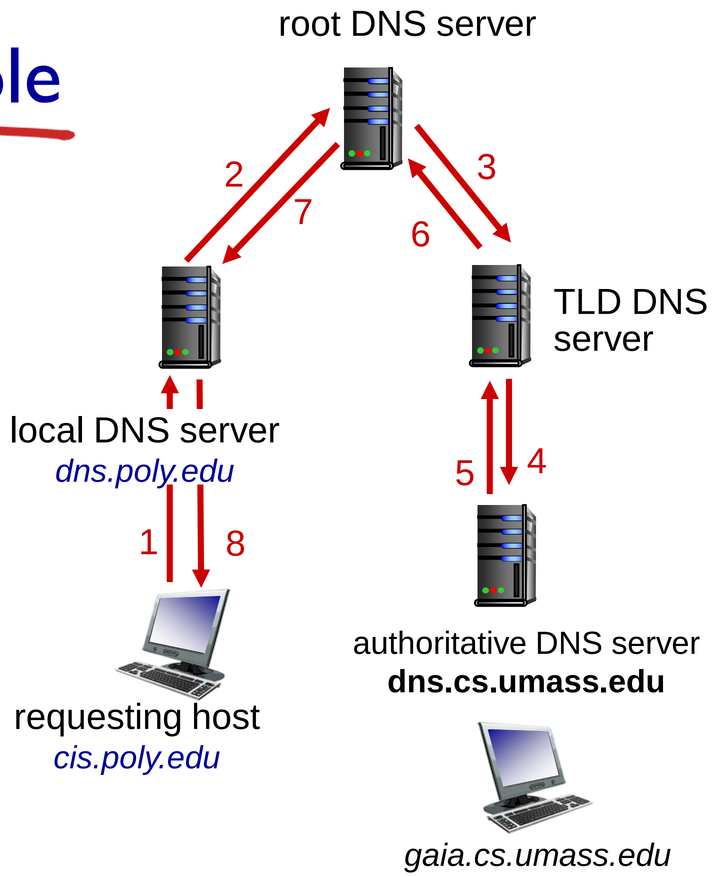
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example

recursive query:

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



DNS records

DNS: distributed database 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=CNAME

- **`name`** is alias name for some "canonical" (the real) name
- **`www.ibm.com`** is really

  `servereast.backup2.ibm.com`
- **`value`** is canonical name

type=MX

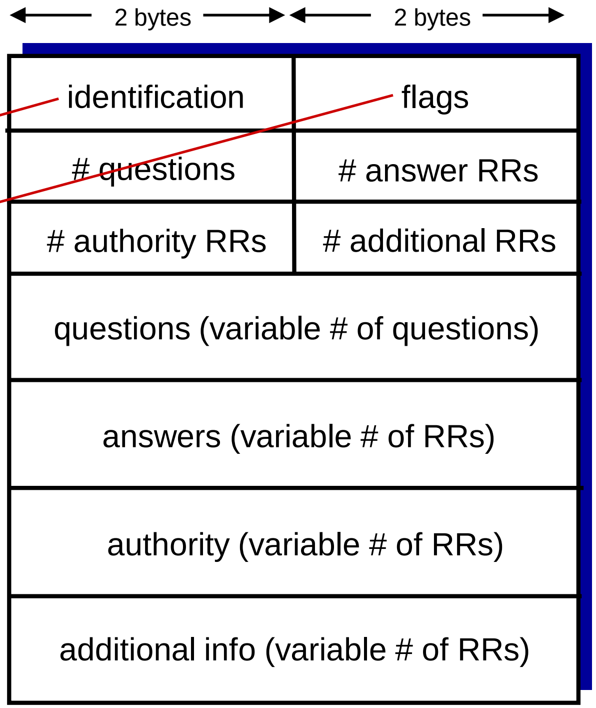
- **`value`** is name of mailserver associated with **`name`**

DNS protocol, messages

- *query* and *reply* messages, both with same *message format*

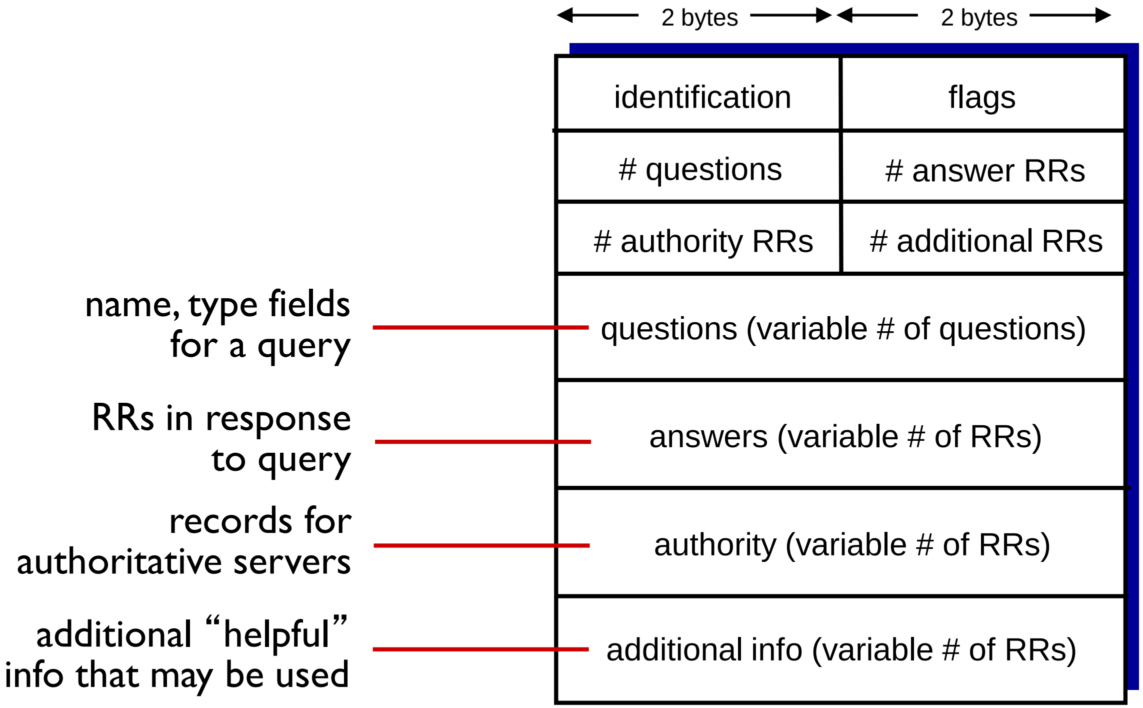
message header

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



2 bytes	2 bytes
identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

DNS protocol, messages



2 bytes	2 bytes
identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

name, type fields for a query

RRs in response to query

records for authoritative servers

additional "helpful" info that may be used

Inserting records into DNS

- example: new startup "Network Utopia"
- register name networkuptopia.com at *DNS registrar* (e.g., Network Solutions)
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts two RRs into .com TLD server:

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

    `(dns1.networkutopia.com, 212.212.212.1, A)`
- create authoritative server type A record for www.networkuptopia.com; type MX record for networkutopia.com