Redes de Computadores

Application Layer

Manuel Ricardo, Rui Prior

Universidade do Porto

Based on slides provided by J.F Kurose and K.W. Ross for their book “Computer Networking: a Top-Down Approach”

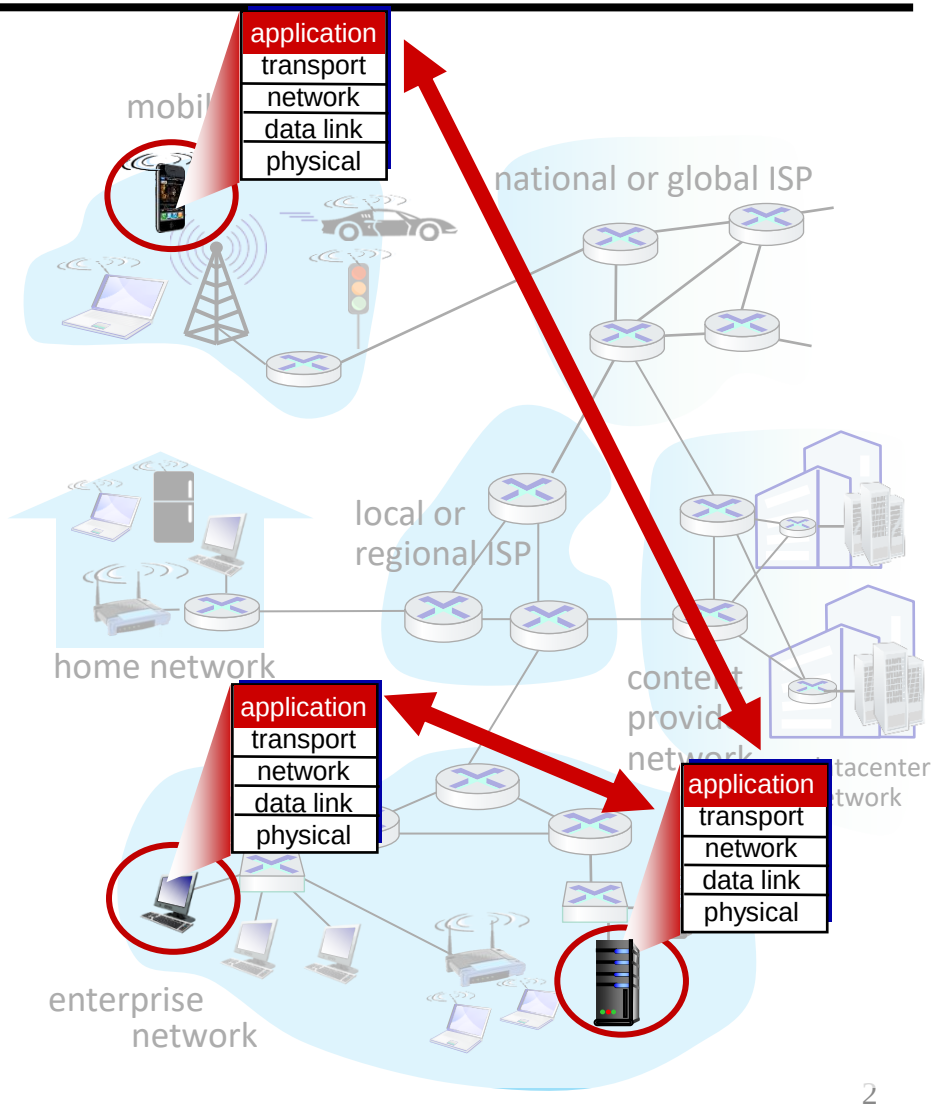
Creating a network app

write programs that

- run on (different) end systems
- communicate over network
- e.g., web server software communicates with browser software

no need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allow for rapid app development, propagation



An application-layer protocol defines:

- types of messages exchanged,
  - » e.g., request, response
- message syntax
  - » what fields in messages & how fields are delineated
- message semantics
  - » meaning of information in fields
- rules for when and how processes send & respond to messages

open application protocols:

- defined in RFCs, everyone has access to protocol definition
- allows for interoperability
- e.g., HTTP, SMTP

proprietary application protocols:

- e.g., Skype, Zoom

A note on security

- TCP and UDP offer no security
  - » e.g., passwords are sent in cleartext, visible on the Internet
- Transport Layer Security (TLS) is an additional "layer" providing
  - » Encryption (privacy)
    - – Eavesdroppers on the Internet see only "random" bits
  - » Message integrity
    - – Message has not been forged or tampered with
  - » Endpoint authentication
    - – Through the use of certificates
- TLS implemented as a library, included in the application layer
  - » Talks to TCP

HyperText Transfer Protocol (HTTP)

Web and HTTP

First some jargon

- 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:

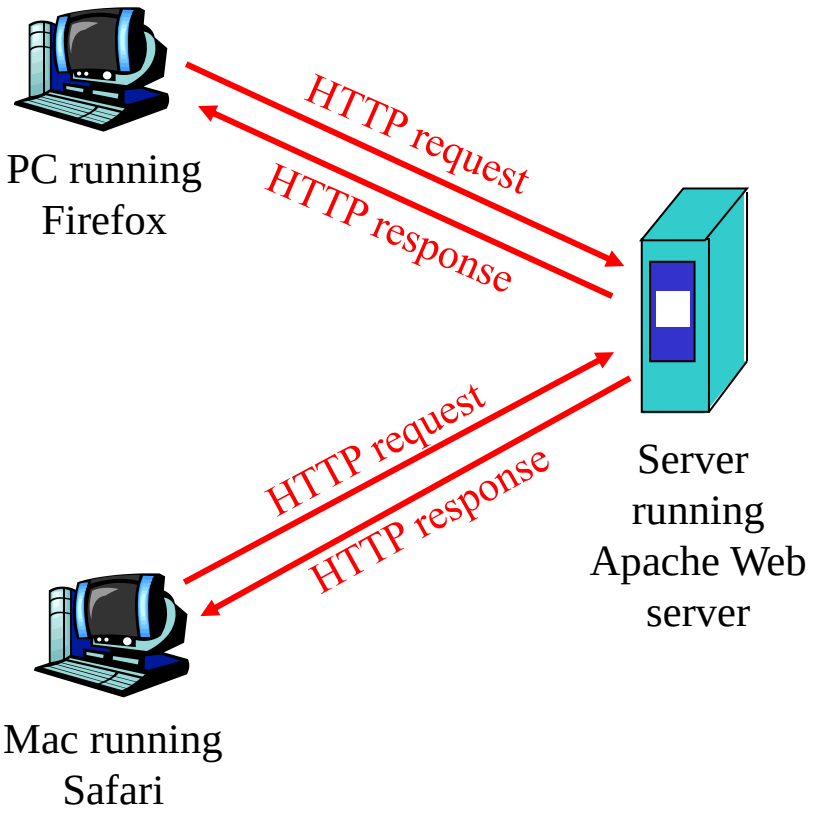
`http://www.someschool.edu/someDept/pic.gif`

protocol | host name | path name

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 overview (continued)

Uses TCP:

- 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 Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- server maintains no information about past client requests

Nonpersistent HTTP

- At most one object is sent over a TCP connection

Persistent HTTP

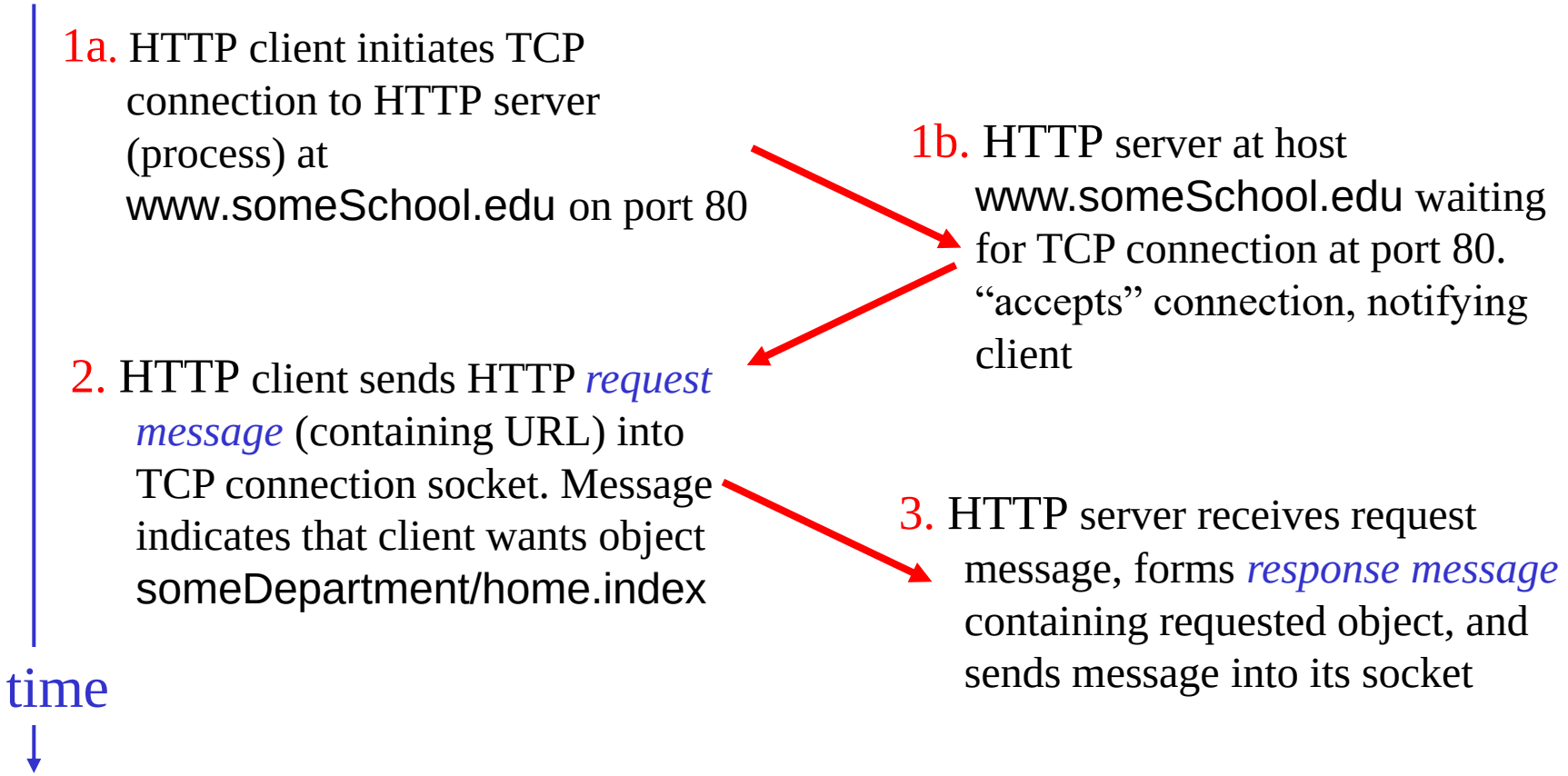
- Multiple objects can be sent over single TCP connection between client and server

Nonpersistent HTTP

Suppose user enters URL

`www.someSchool.edu/someDepartment/home.index`

(contains text and references to 10 jpeg images)



Nonpersistent HTTP (cont.)

4. HTTP server closes TCP connection.

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

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

time

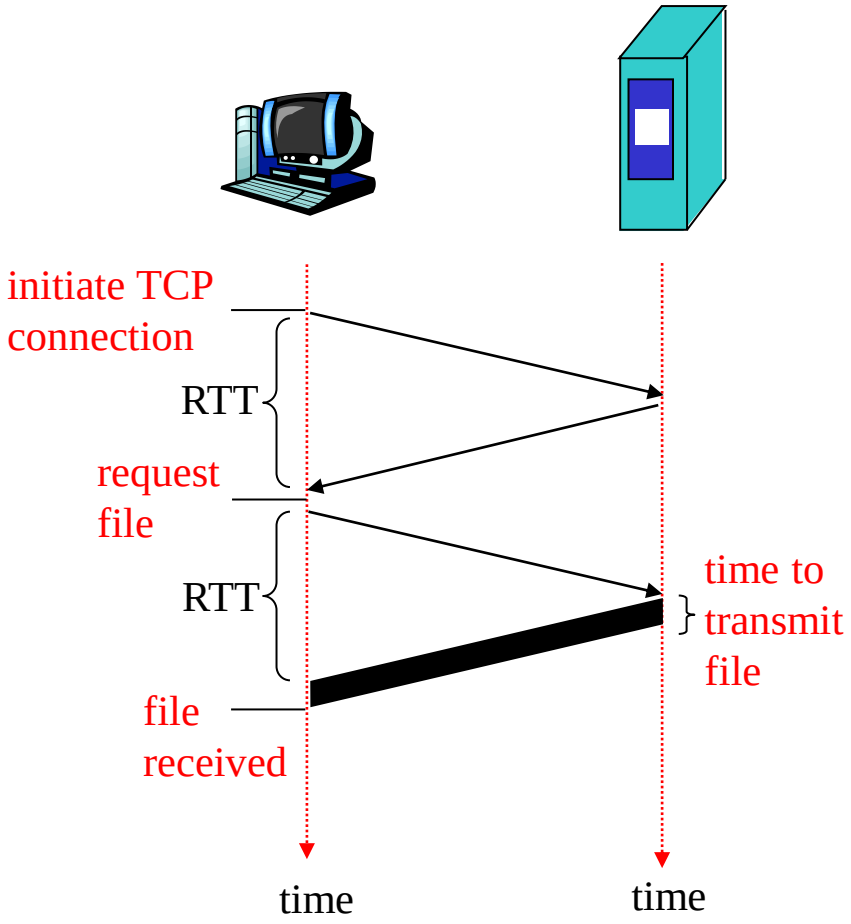
Non-Persistent HTTP: Response time

Definition of RTT: time for 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 time



Persistent HTTP

Nonpersistent HTTP 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

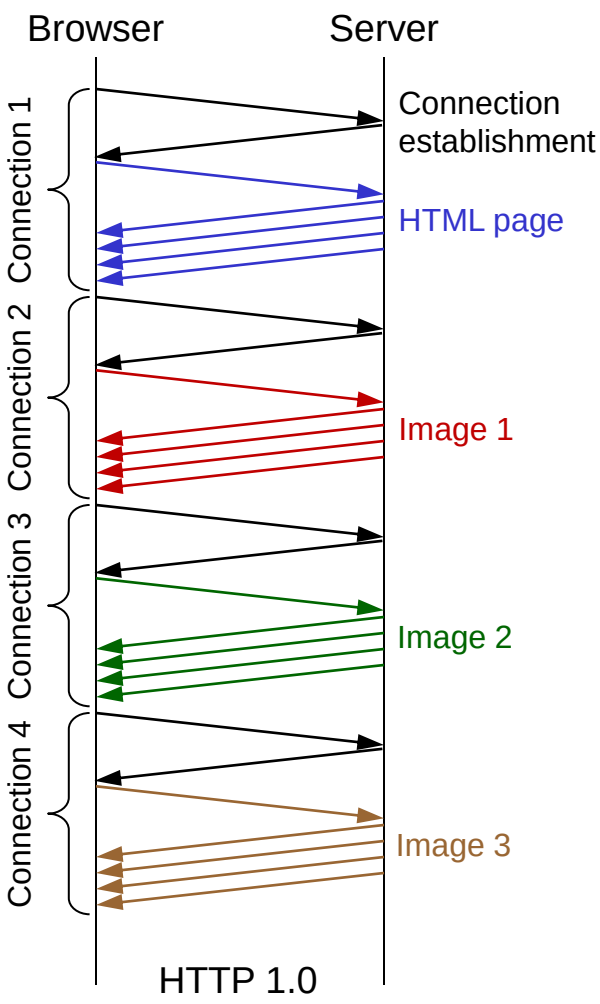
Persistent **without** pipelining:

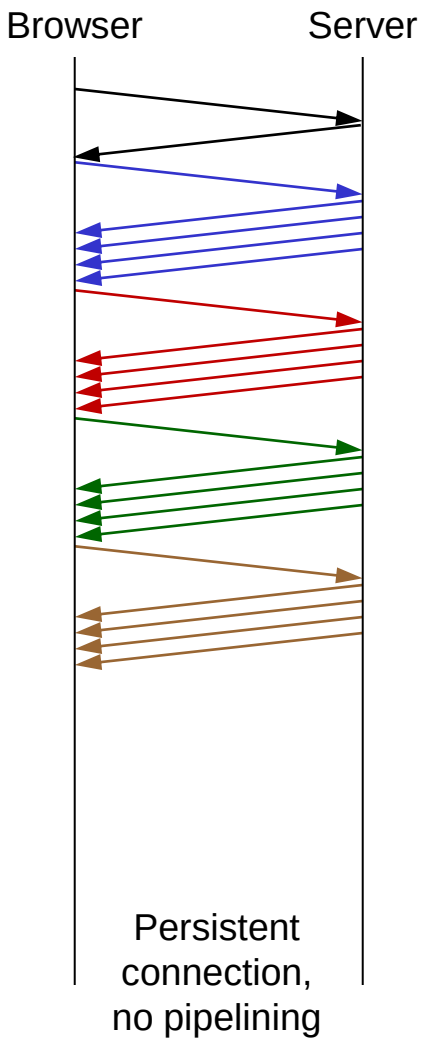
- client makes new request only after receiving the previous reply
- one RTT per referenced object

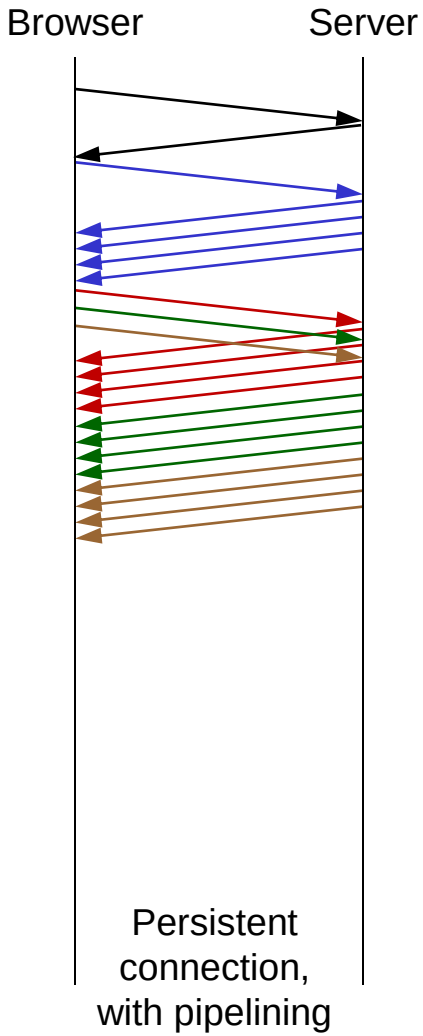
Persistent **with** pipelining:

- default in HTTP/1.1
- client requests new object as soon as it finds its reference
- one RTT for all referenced objects

Persistent connections and pipelining

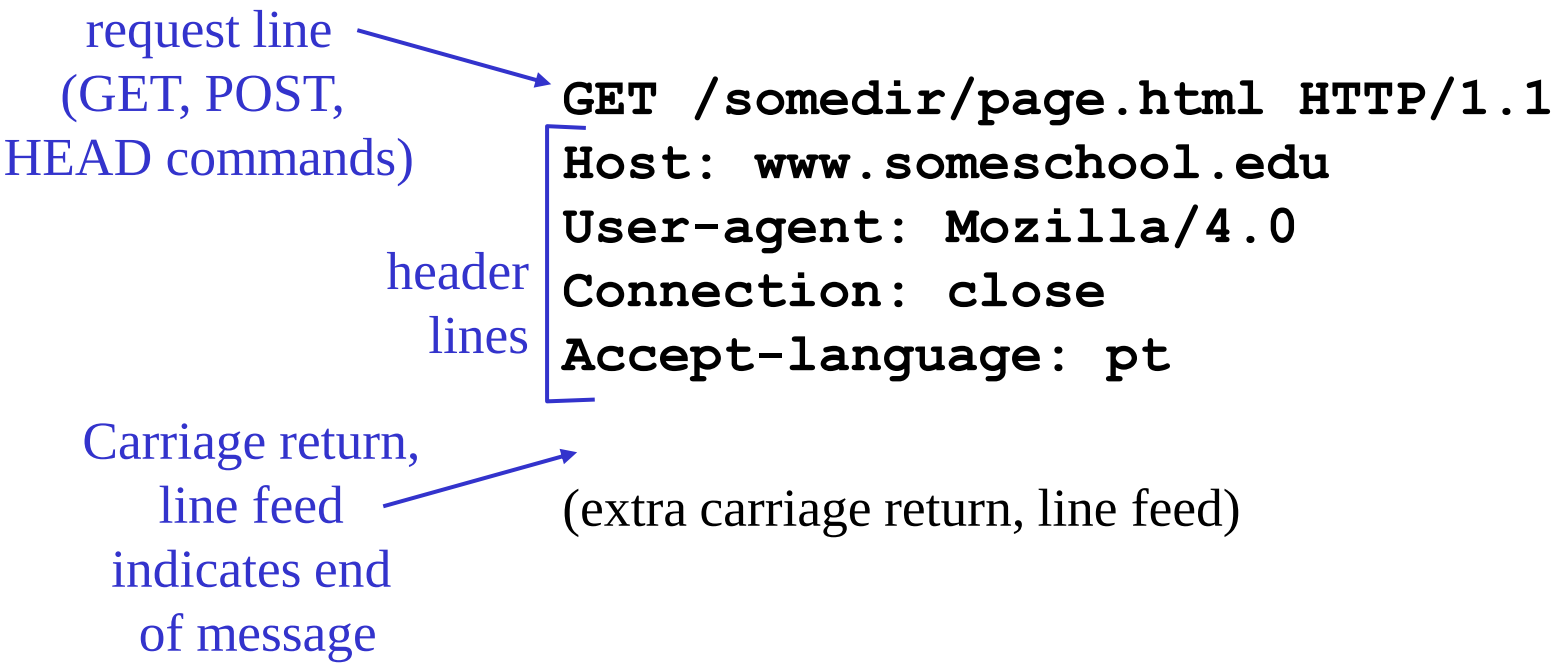




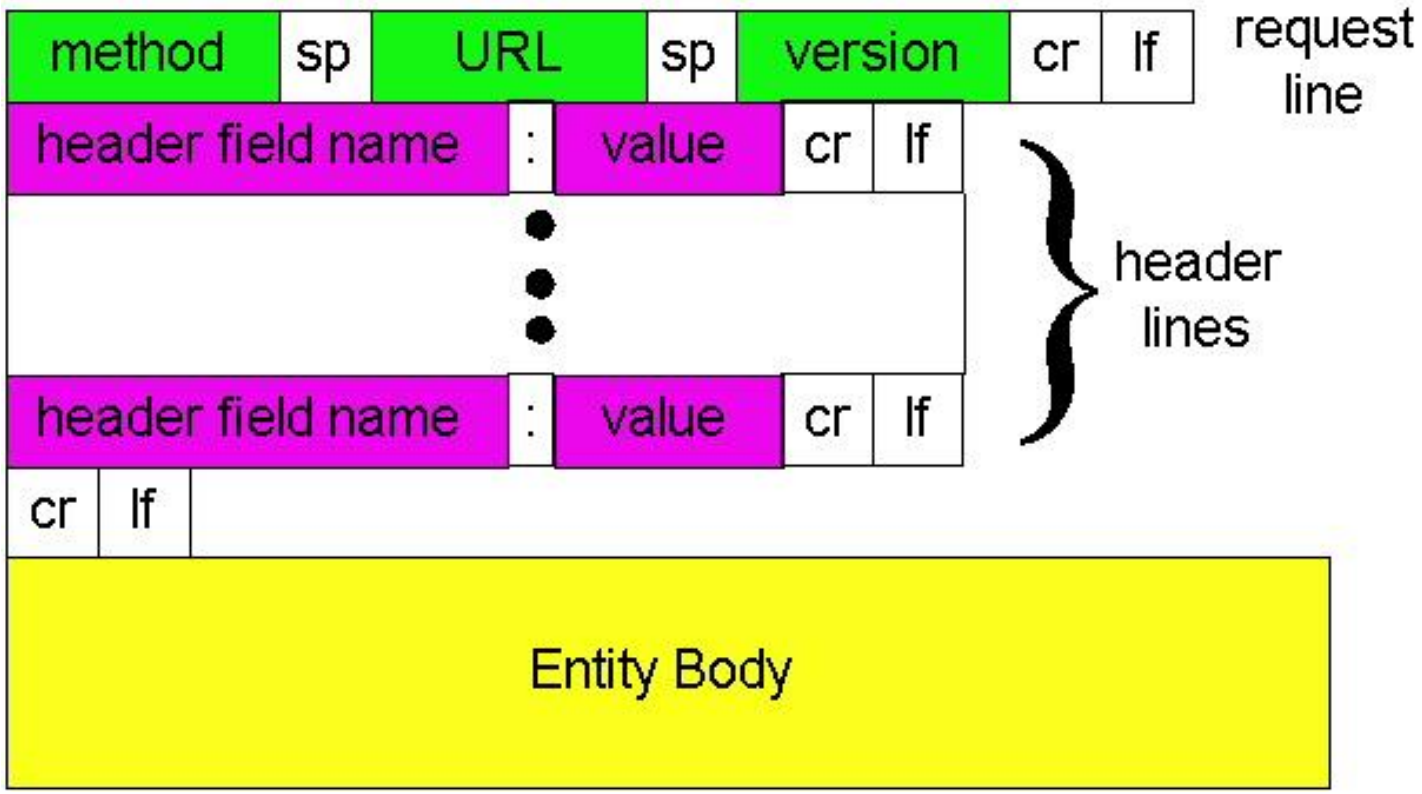


HTTP request message

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



HTTP request message: general format



Uploading form input

POST method:

- Web page often includes form input
- Using POST method, input is uploaded to server in entity body

GET method:

- Input is uploaded in URL field of request line:

```
http://www.animals.com/search?animal=monkeys&likes=banana
```

Some methods

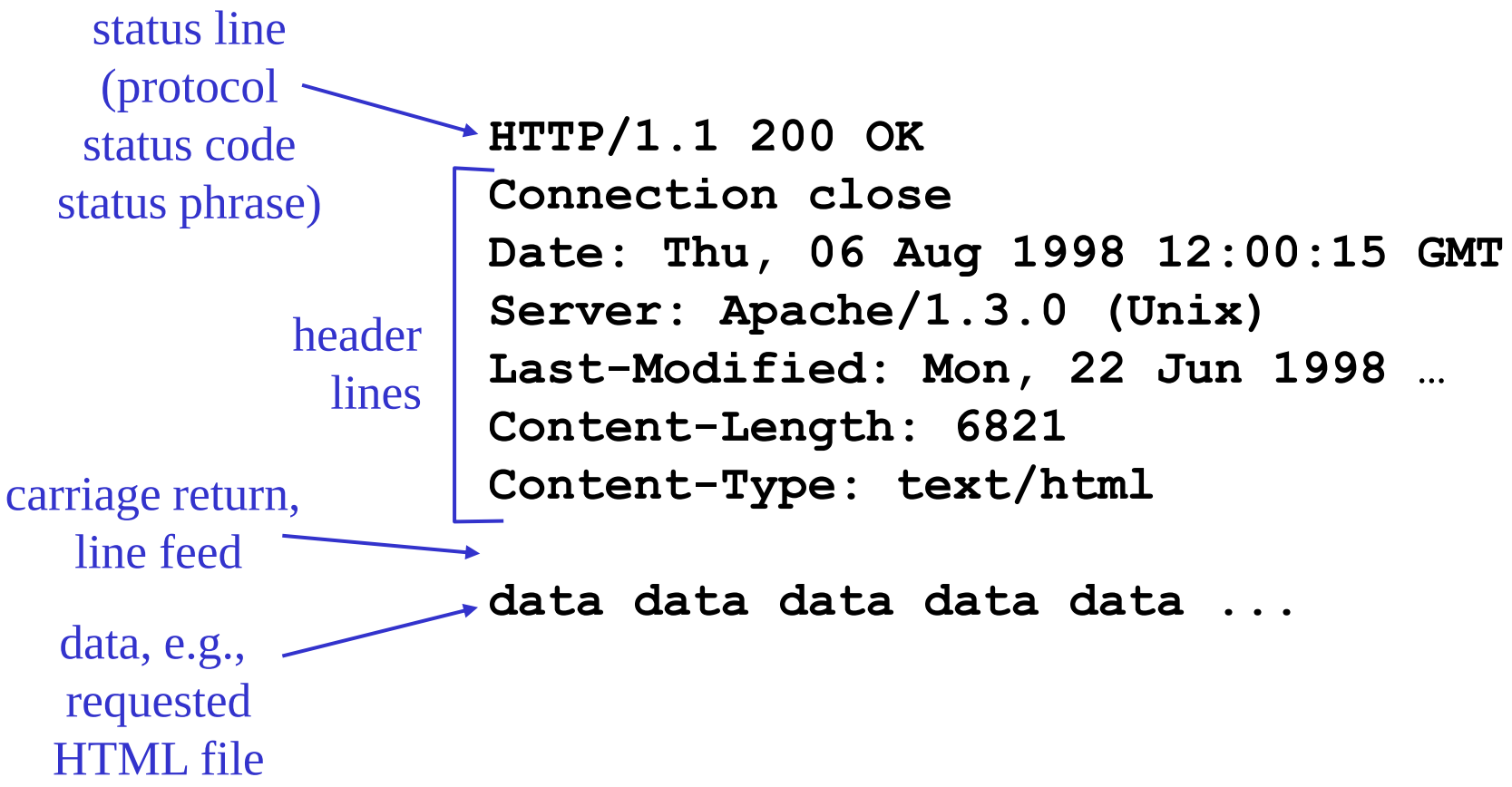
HTTP/1.0

- GET
- POST

HTTP/1.1

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

HTTP response message



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`

User-server state: cookies

- The HTTP protocol itself is stateless
- State information can be conveyed using cookies

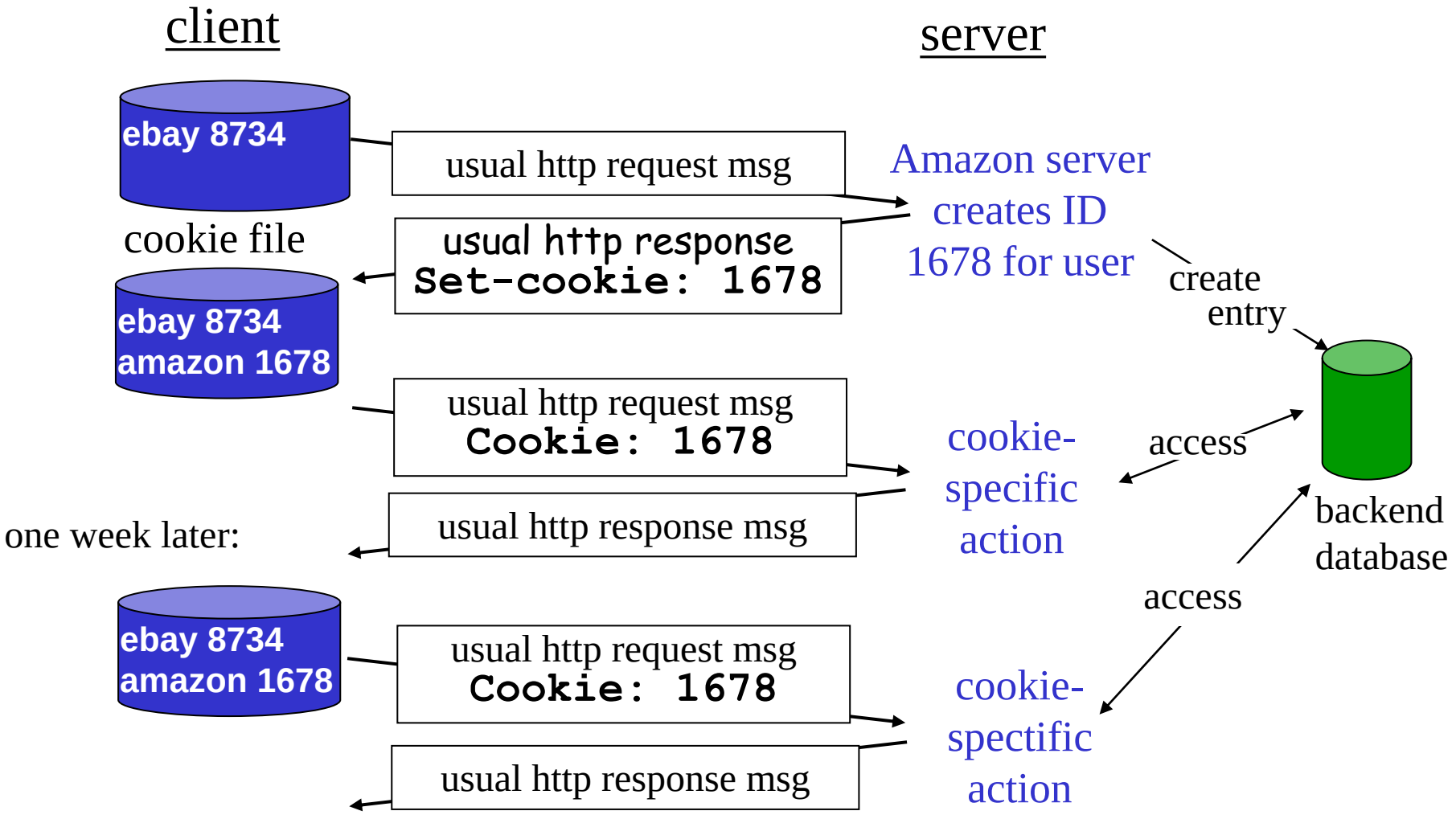
Four components:

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 accesses the Internet from her PC
- visits specific e-commerce site for the first time
- when initial HTTP request arrives at site, site creates:
  - » unique ID
  - » entry in backend database for ID

Cookies: keeping "state"



Cookies (continued)

What cookies can bring:

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

How to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

aside

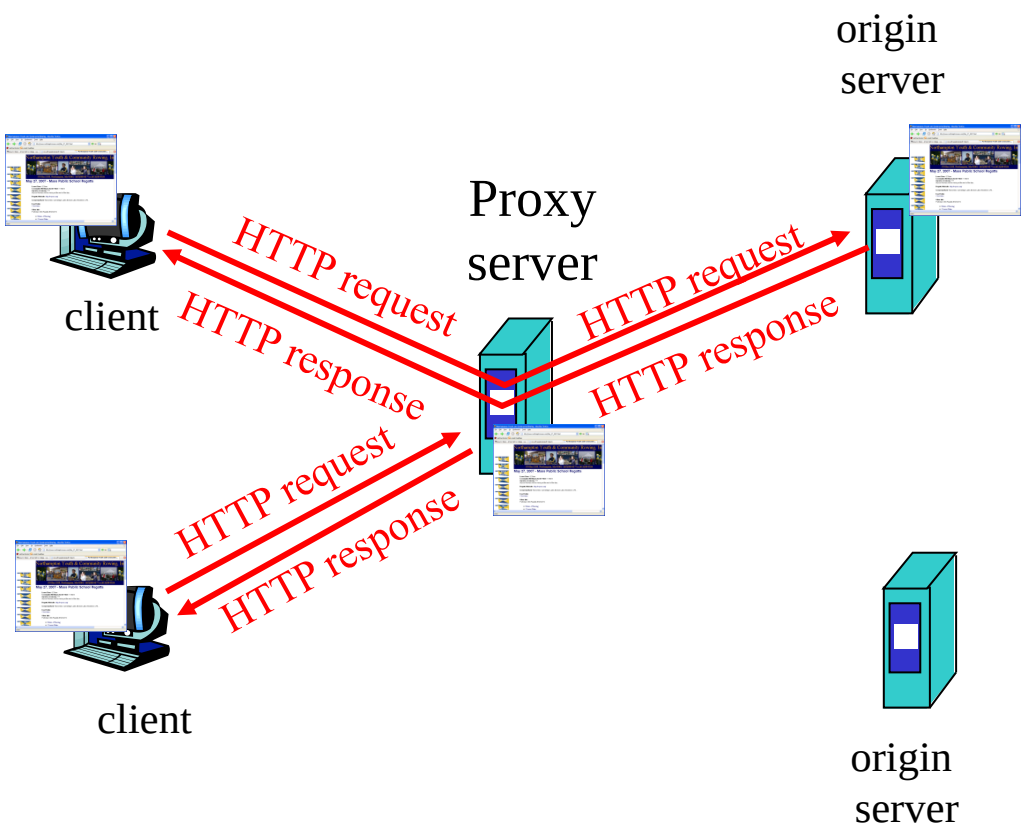
Cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

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



More about Web caching

- cache acts as both client and server
- typically, cache is installed by ISP (university, company, residential ISP)

Why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link

Final notes on HTTP

Newer versions of the protocol:

- HTTP/2, HTTP/3
- Use the same messages but faster / more efficient transport

New uses of Web and HTTP:

- The Internet of Things
- REST APIs
  - The core IETF working group
  - RFC 6690

Electronic Mail (e-mail):
SMTP, POP3, IMAP

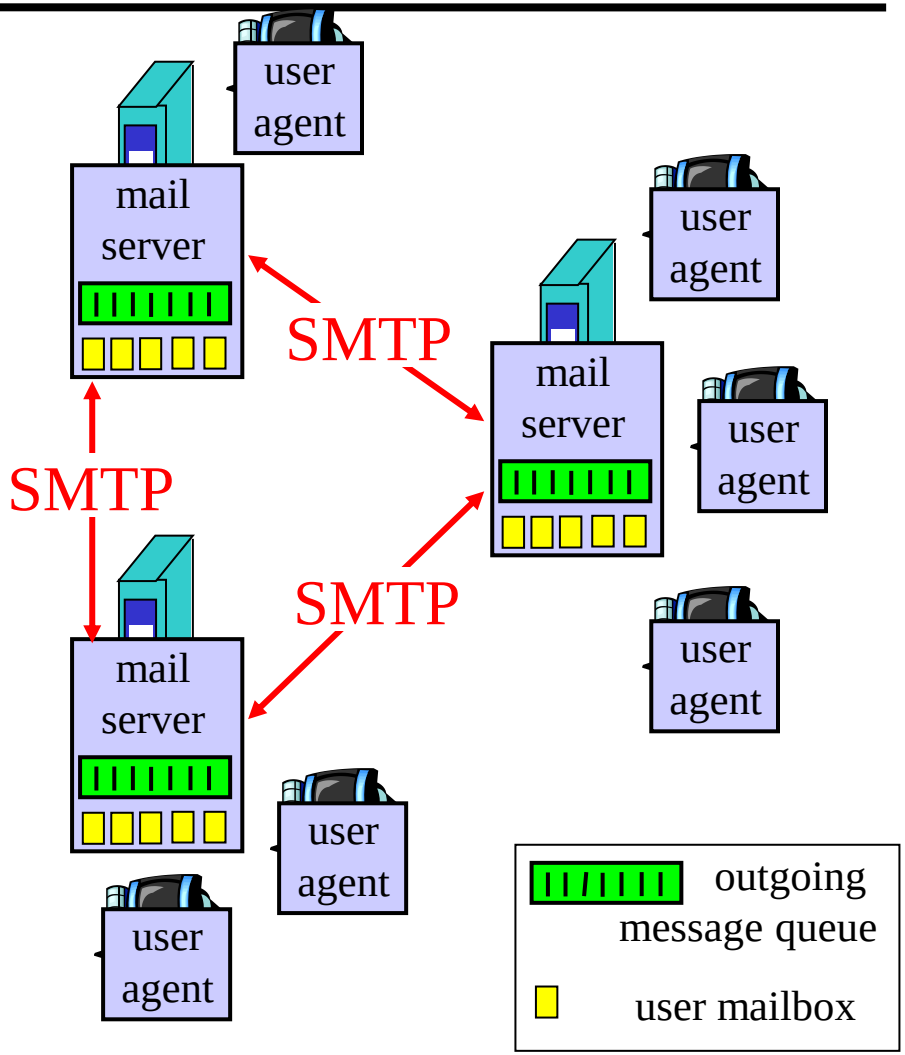
Electronic Mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

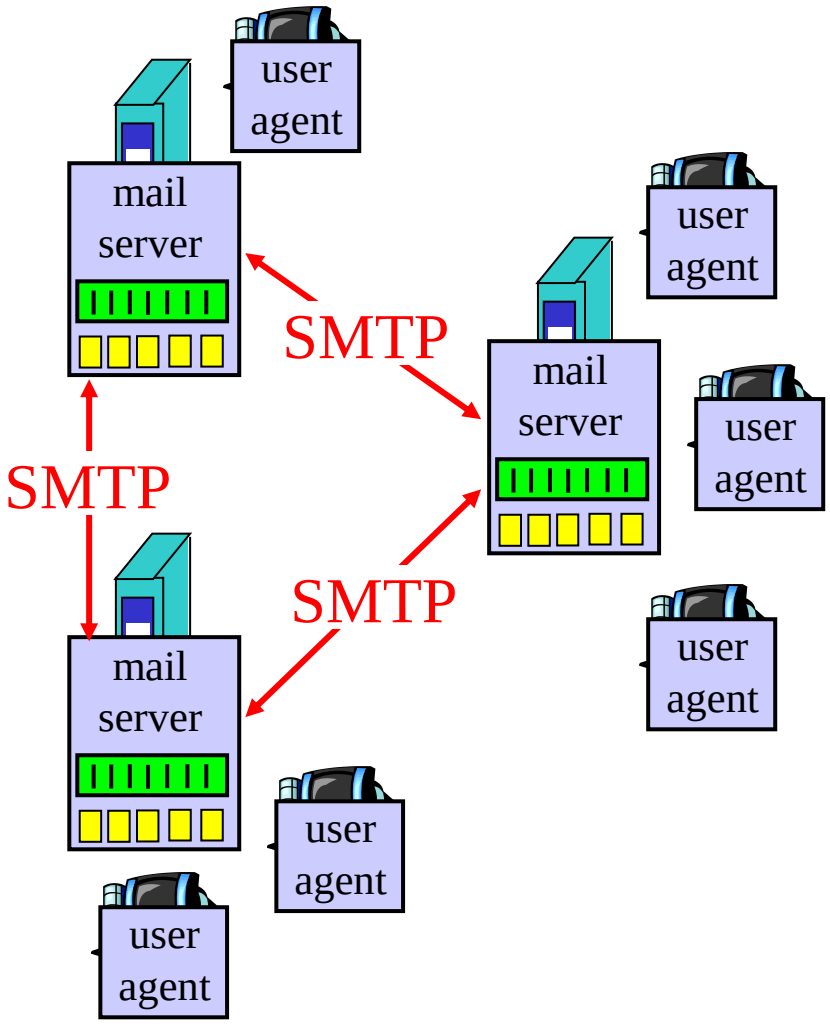
- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Eudora, Outlook, elm, Mozilla Thunderbird
- outgoing, incoming messages stored on server



Electronic Mail: mail servers

Mail Servers

- mailbox contains incoming messages for user
- message queue of outgoing (to be sent) mail messages
- SMTP protocol between mail servers to send email messages
  - » client: sending mail server
  - » "server": receiving mail server

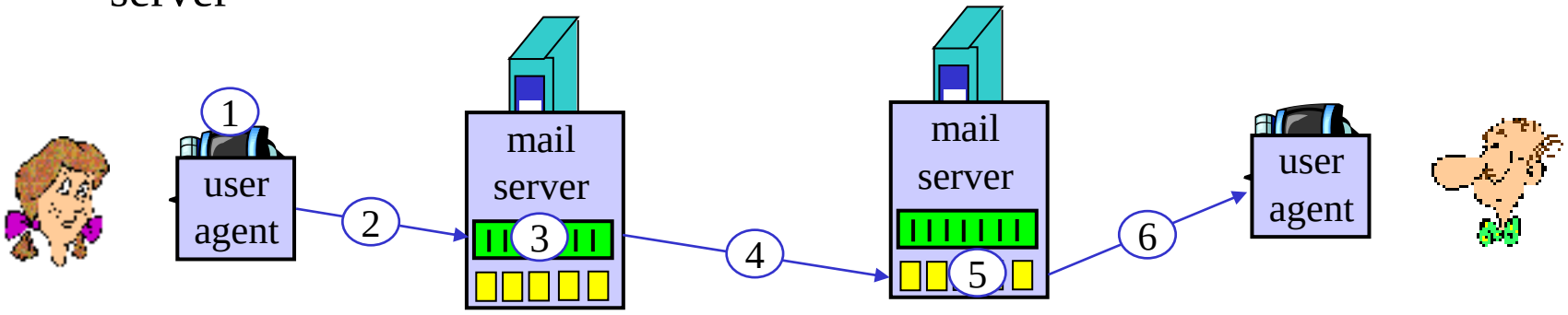


Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - » handshaking (greeting)
  - » transfer of messages
  - » closure
- command/response interaction
  - » commands: ASCII text
  - » response: status code and phrase
- messages must be in 7-bit ASCII

Scenario: Alice sends message to Bob

1) Alice uses UA to compose message and "to" `bob@someschool.edu`
2) Alice's UA sends message to her mail server; message placed in message queue
3) Client side of SMTP opens TCP connection with Bob's mail server
4) SMTP client sends Alice's message over the TCP connection
5) Bob's mail server places the message in Bob's mailbox
6) Bob invokes his user agent to read message



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
```

SMTP: final words

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses `CRLF.CRLF` to determine end of message

Comparison with HTTP:

- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

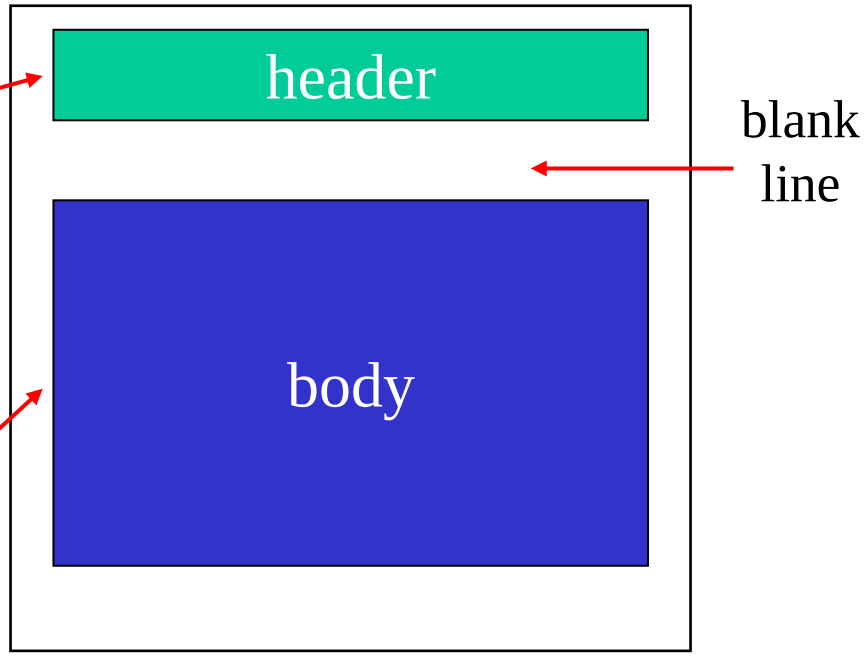
Mail message format

SMTP: protocol for exchanging email messages

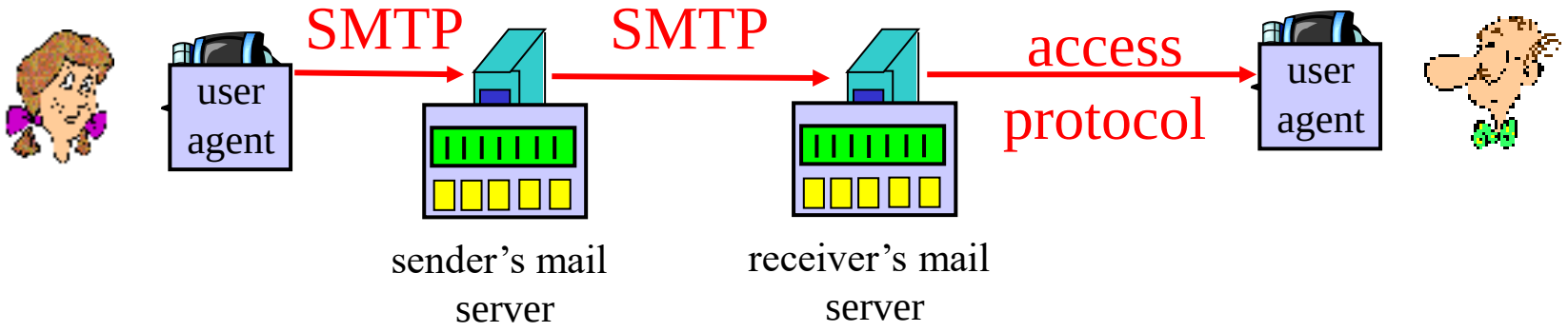
RFC 822: standard for text message format:

- header lines, e.g.,
  - » To:
  - » From:
  - » Subject:

  different from SMTP commands!
- body
  - » the “message”, ASCII characters only



Mail access protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - » POP3: Post Office Protocol [RFC 1939]
    - – authorization (agent ↔ server) and download
  - » IMAP: Internet Mail Access Protocol [RFC 1730]
    - – more features (more complex)
    - – manipulation of stored messages on server
  - » HTTP: Gmail, Outlook, ProtonMail, etc.

POP3 protocol

authorization phase

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

transaction phase, client:

- **`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
S: +OK Message 1 deleted.
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
S: +OK Message 2 deleted.
C: quit
S: +OK POP3 server signing off
```

POP3 (more) and IMAP

More about POP3

- Previous example uses "download and delete" mode
- Bob cannot re-read e-mail if he changes client
- "Download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

IMAP

- Keep all messages in one place: the server
- Allows user to organize messages in folders
- IMAP keeps user state across sessions:
  - » names of folders and mappings between message IDs and folder name

Domain Name System (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., ww.yahoo.com - 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

DNS services

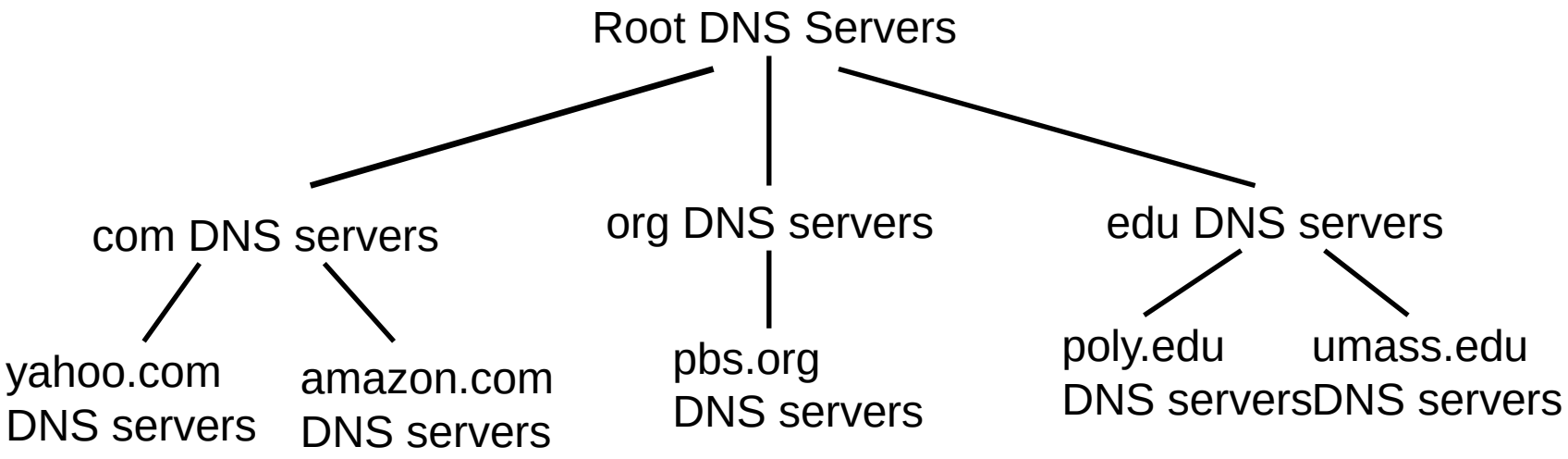
- hostname to IP address translation
- host aliasing
  - » Canonical, alias names
- mail server aliasing
- load distribution
  - » replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?

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

➔ would not *scale!*

Distributed, Hierarchical Database

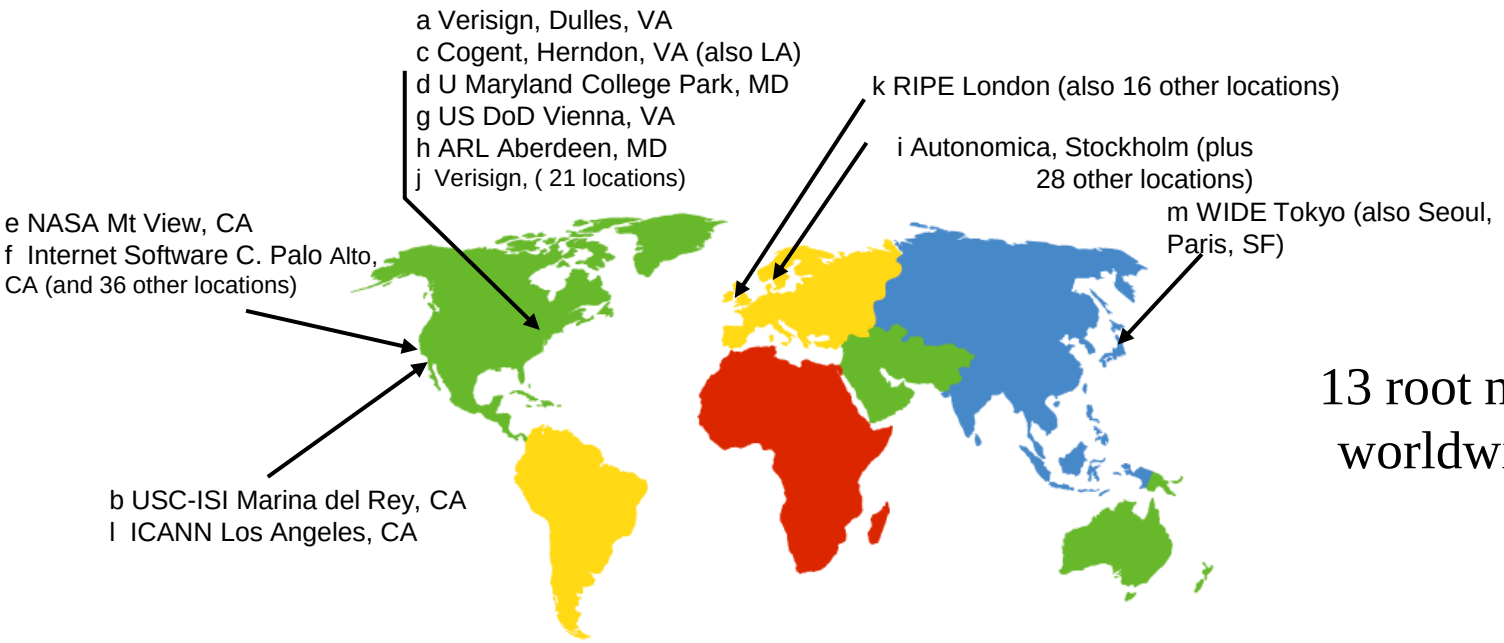


Client wants IP for www.amazon.com; 1st approx:

- 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
- return references to top-level domain servers



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.

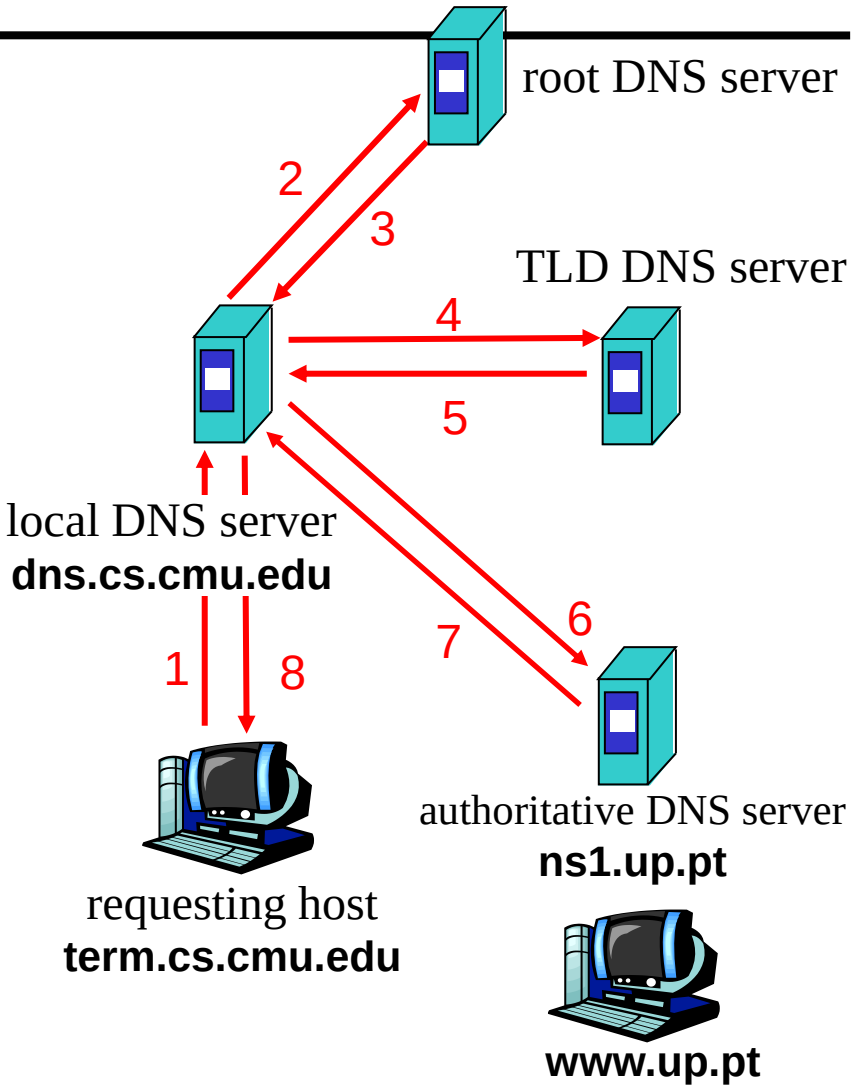
- Authoritative DNS servers:
  - » organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web, mail).
  - » can be maintained by organization or service provider

Local Name Server

- does not strictly belong to the hierarchy
- each ISP (residential ISP, company, university) has one
  - » also called “default name server”
- hosts make DNS queries to the local DNS server
  - » acts as proxy, querying the hierarchy to obtain the response
  - » caches results for efficiency

DNS name resolution example

- Host at cs.cmu.edu wants IP address for www.up.pt
- Makes recursive query to the local DNS server
  - » Response must be requested information or error
- The local DNS server resolves the name iteratively
  - » Makes non-recursive queries down the hierarchy
  - » Responses can be references to other DNS servers: "*I don't know, but ask this server…*"
- The local DNS server sends the final result to the client
  - » Caches final and intermediate results



DNS records

DNS: distributed database storing resource records (RR)

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

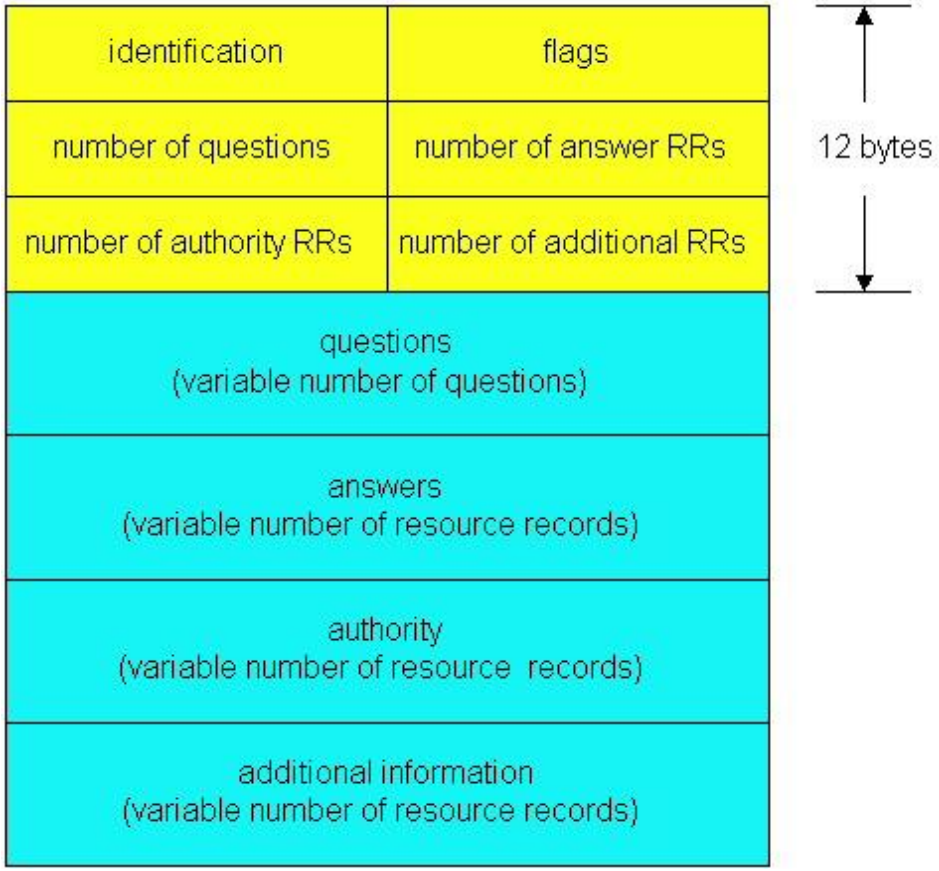
- Type=A
  - » **`name`** is hostname
  - » **`value`** is IP address
- Type=NS
  - » **`name`** is domain (e.g. foo.com)
  - » **`value`** is hostname of authoritative name server for this domain
- Type=CNAME
  - » **`name`** is alias name for some "canonical" (the real) name `www.ibm.com` is really `servereast.backup2.ibm.com`
  - » **`value`** is canonical name
- Type=MX
  - » **`value`** is name of mailserver associated with **`name`**

DNS protocol, messages

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

DNS message header:

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



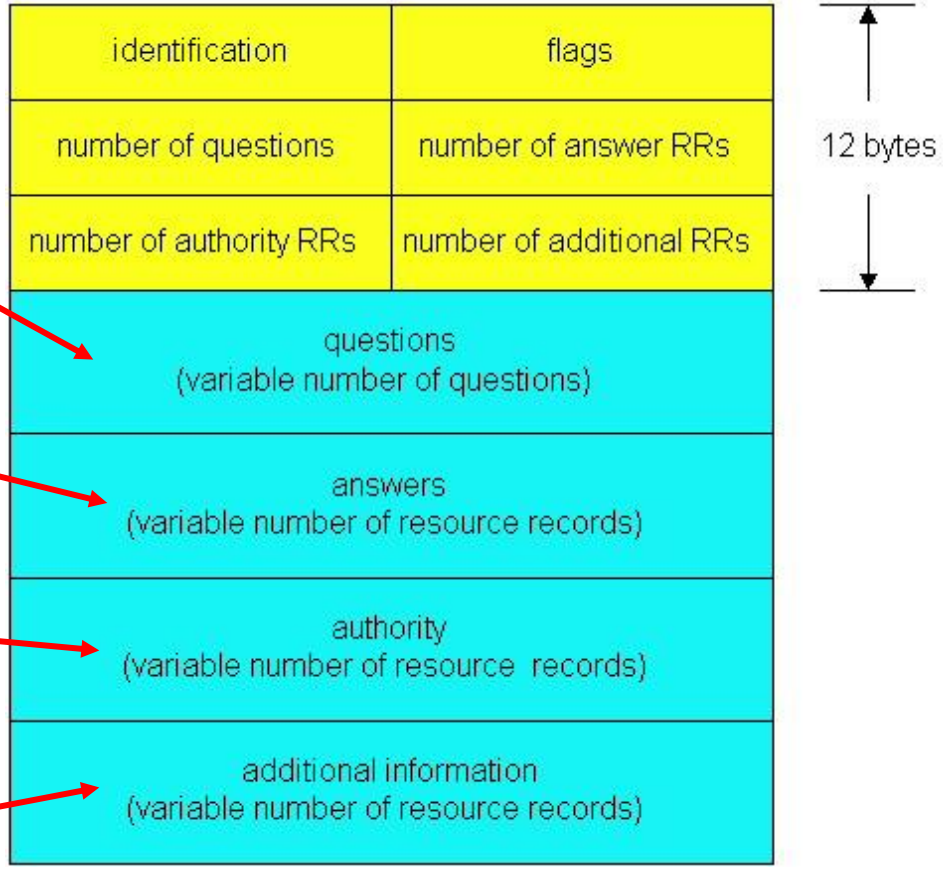
DNS protocol, messages

Name, type fields for a query

RRs in response to query

records for authoritative servers

additional "helpful" info that may be used



Example of DNS response

```
[myself@localhost ~]$ dig mit.edu

; <<>> DiG 9.5.0-P1 <<>> mit.edu
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 28372
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;mit.edu.                       IN        A

;; ANSWER SECTION:
mit.edu.            60         IN        A         18.7.22.69

;; AUTHORITY SECTION:
mit.edu.            11809      IN        NS        STRAWB.mit.edu.
mit.edu.            11809      IN        NS        BITSY.mit.edu.
mit.edu.            11809      IN        NS        W20NS.mit.edu.

;; ADDITIONAL SECTION:
BITSY.mit.edu.                 363        IN        A         18.72.0.3
W20NS.mit.edu.                 3667       IN        A         18.70.0.160
STRAWB.mit.edu.                3667       IN        A         18.71.0.151
```

Inserting records into DNS

- example: new startup "Network Utopia"
- register name networkuptopia.com at *DNS registrar* (e.g., Network Solutions)
  - » provide name(s) and IP address(es) of authoritative name server(s)
  - » registrar inserts two RRs into `com` TLD server:

```
networkutopia.com NS dns1.networkutopia.com
dns1.networkutopia.com A 212.212.212.1
```

- in the authoritative server, create records for names in the domain
  - » A record for www.networkuptopia.com
  - » MX record for networkutopia.com
  - » …

Q: How do people get IP address of your Web site?