Chapter 2: Application Layer

Course on Computer Communication and Networks, CTH/GU

The slides are adaptation of the slides made available by the authors of the course's main textbook:
Computer Networking: A Top Down Approach,
5th edition.
Jim Kurose, Keith Ross
Addison-Wesley, 2009.

Chapter 2: Application Layer

Chapter goals:

- conceptual + implementation aspects of network application protocols
  - client server, p2p paradigms (*we will study the latter seperately*)
  - service models
- learn about protocols by examining popular application-level protocols (more will come later, when studying real-time traffic aspects)
- specific protocols:
  - http, (ftp), smtp, pop, dns,
  - p2p file sharing, multimedia apps: we cover that later, after having an overview of the layers
- programming network applications
  - socket programming

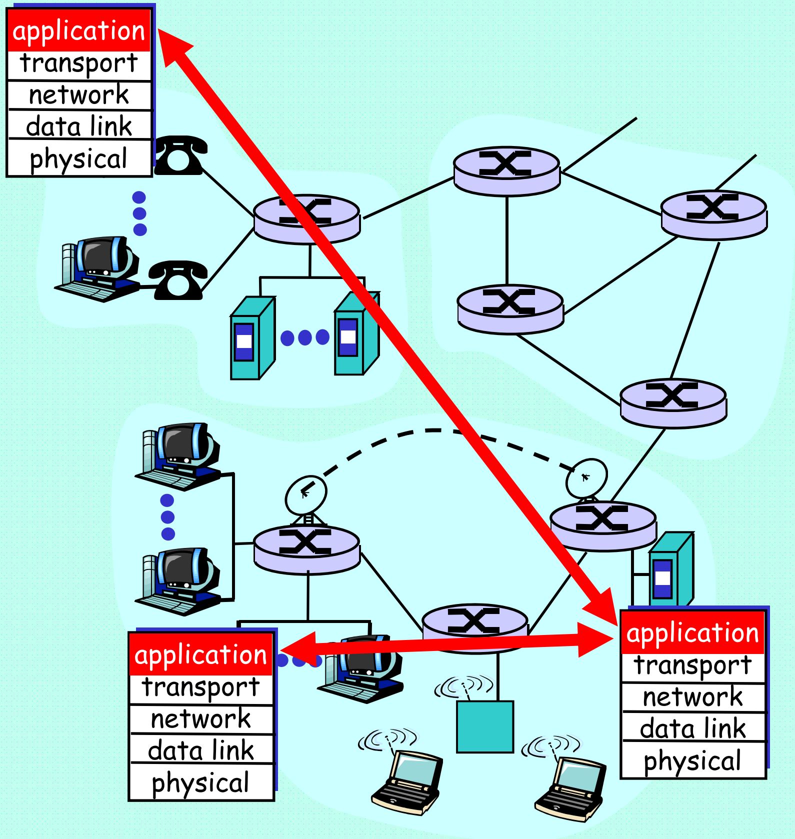
Applications and application-layer protocols

Application: communicating, distributed processes

- running in network hosts in "user space"
- exchange messages
- e.g., email, file transfer, the Web

Application-layer protocols

- one "piece" of an application - others are e.g. user agents.
  - Web:browser
  - E-mail: mail reader
  - streaming audio/video: media player
- define messages exchanged by apps and actions taken
- use services provided by lower layer protocols



Client-server paradigm

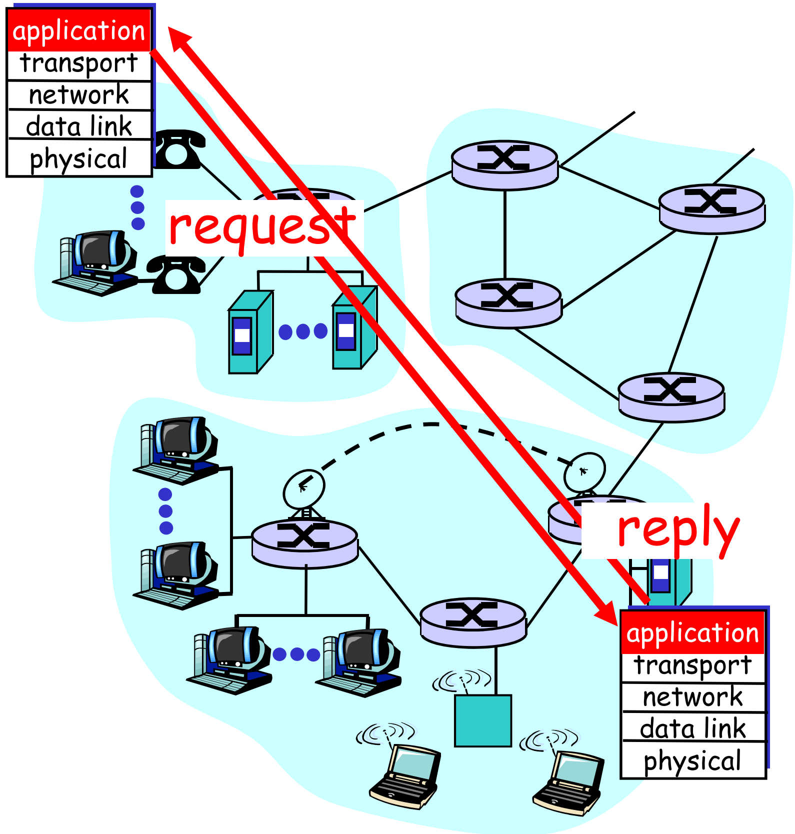
Typical network app has two pieces: *client* and *server*

Client:

- initiates contact with server ("speaks first")
- typically requests service from server,
- for Web, client is implemented in browser; for e-mail, in mail reader

Server:

- provides requested service to client
- e.g., Web server sends requested Web page, mail server delivers e-mail



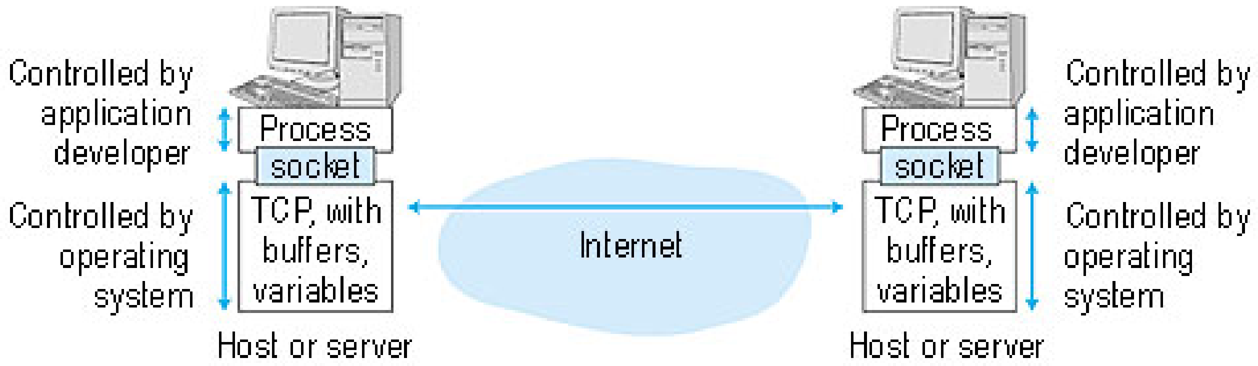
Auxiliary terms ++

socket: Internet application programming interface

- 2 processes communicate by sending data into socket, reading data out of socket (like sending out, receiving in via doors)

Q: how does a process "identify" the other process with which it wants to communicate?

- IP address of host running other process
- "port number" - allows receiving host to determine to which local process the message should be delivered



… more: cf programming project guidelines

Properties of transport service of interest to the app

Reliability-related

- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, telnet) require 100% reliable data transfer
- Connection-oriented vs connectionless services

Bandwidth, Timing

- some apps (e.g., multimedia) require minimum amount of bandwidth
- some apps (e.g., Internet telephony, interactive games) require low delay and/or low jitter
- other apps (elastic apps, e.g. file transfer) make use of whatever bandwidth, timing they get

Transport service requirements of common apps

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	No-loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kb-1Mb video:10Kb-5Mb	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few Kbps up	yes, 100's msec
financial apps	no loss	elastic	yes and no

What we need

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
- *does not provide:* timing, minimum bandwidth guarantees
- *(extra service: for the health of the NW, not for each user: congestion control:* throttle sender when network overloaded)

UDP service:

- *connectionless*
- *unreliable transport* between sending and receiving process
- *does not provide*: flow control, congestion control, timing, or bandwidth guarantee

Q: why bother? Why is there a UDP?

What we have

Internet apps: their 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 + "tricks"
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP, TCP also possible + "tricks"
nslookup and many others	» DNS [RFC 882, 883,1034,1035]	UDP

What we do

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 and path name:

www.someSchool.edu/someDept/pic.gif

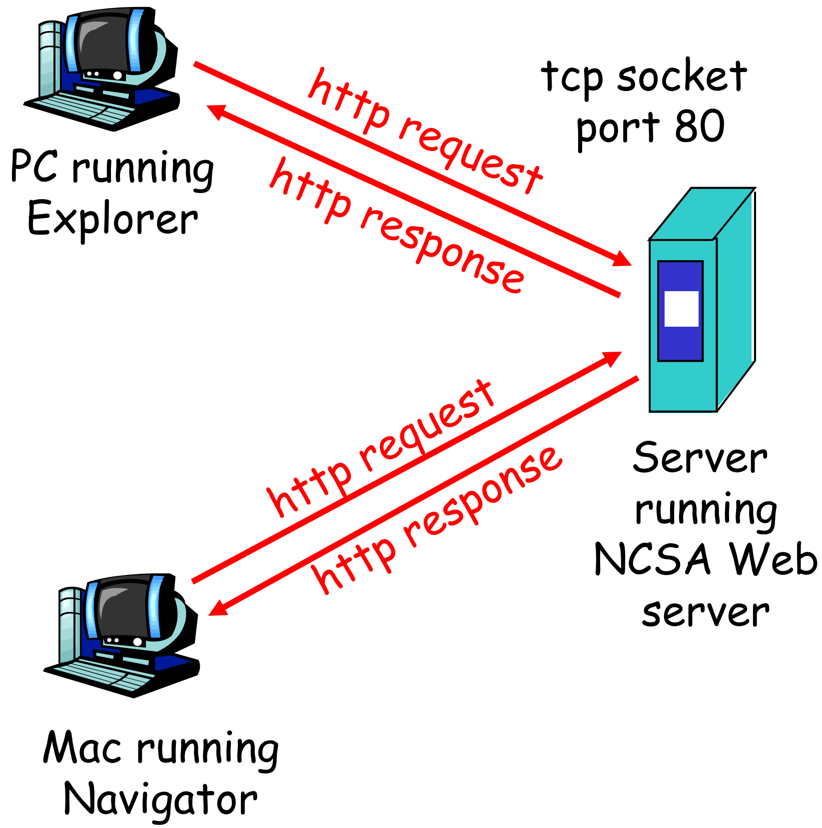
- User agent for Web is called a browser:
  - MS Internet Explorer
  - Netscape Communicator
- Server for Web is called Web server:
  - Apache (public domain)
  - MS Internet Information Server
  - Netscape Enterprise Server

The Web: the http protocol

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

server accepts TCP connection

- 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

Protocols that maintain "state" are complex!

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

- http1.0: RFC 1945
- http1.1: RFC 2068

http example

Suppose user enters URL
www.someSchool.edu/someDepartment/home.index

(contains text, references to 10 jpeg images)

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

1b. http server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying 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 (someDepartment/home.index), sends message into socket

time

http example (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

http message format: request

ASCII (human-readable format;
try telnet to www server, port 80)

request line (GET, POST, HEAD (PUT, DELETE in v 1.1.) commands)

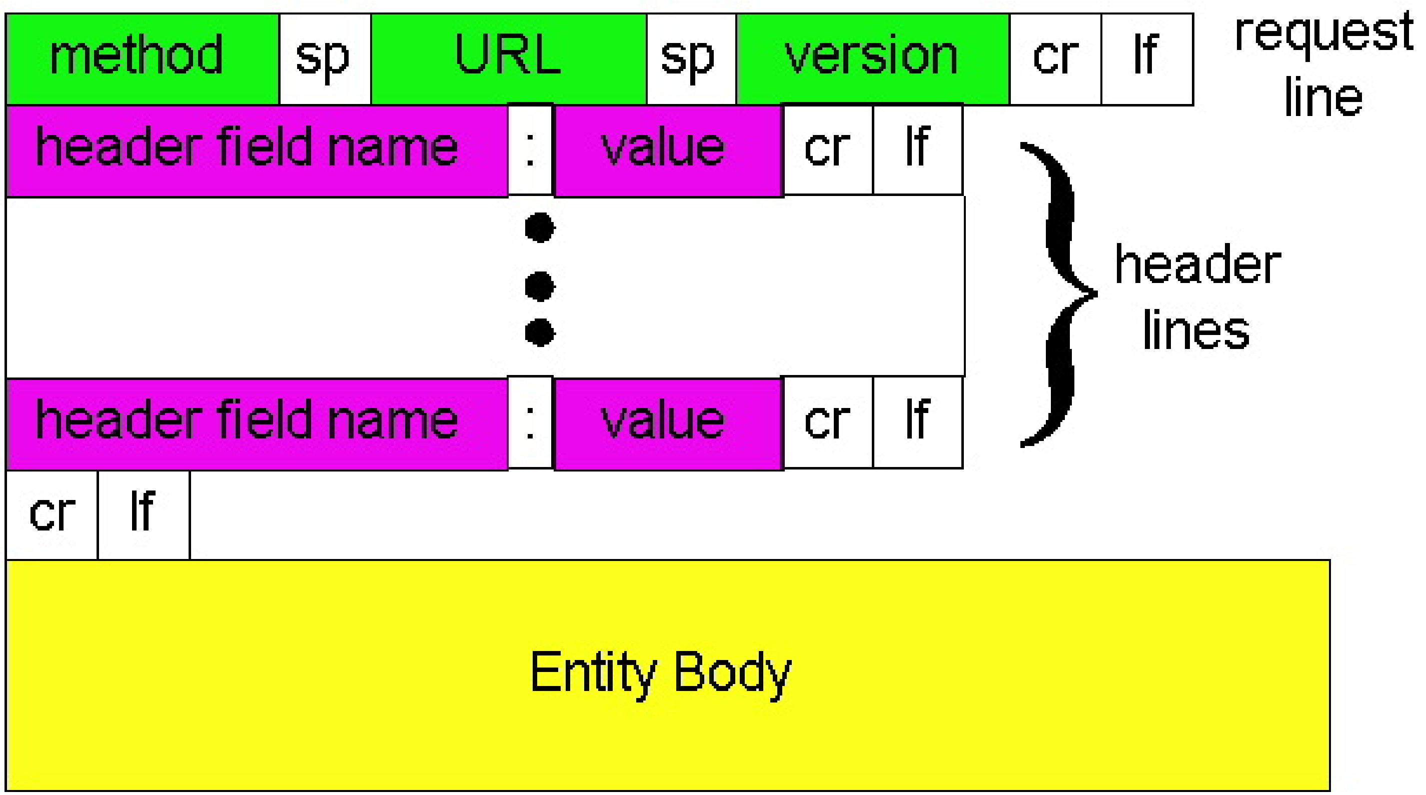
header lines

```
GET /somedir/page.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr
```

(extra carriage return, line feed)

Carriage return, line feed indicates end of message

http request message: general format



http message format: respone

status line (protocol status code status phrase)

```
HTTP/1.0 200 OK
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

Trying out http (client side) for yourself

1. Telnet to your favorite Web 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!

Non-persistent and persistent connections

Non-persistent

- HTTP/1.0
- server parses request, responds, and closes TCP connection
- new TCP connection for each object => extra overhead per object

But most 1.0 browsers use parallel TCP connections.

Persistent

- 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;
- Less overhead per object
- Objects are fetched sequentially

But can also pipeline requests (resembles non-persistent optimised behaviour)

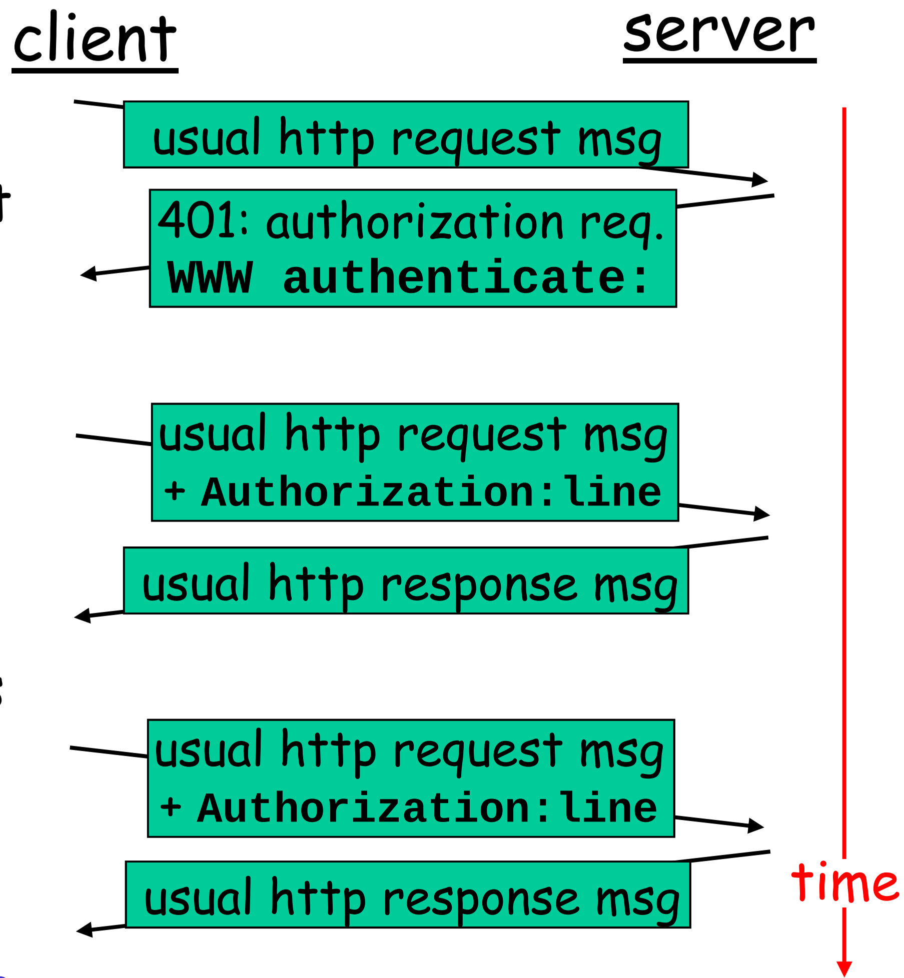
User-server interaction: authentication

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

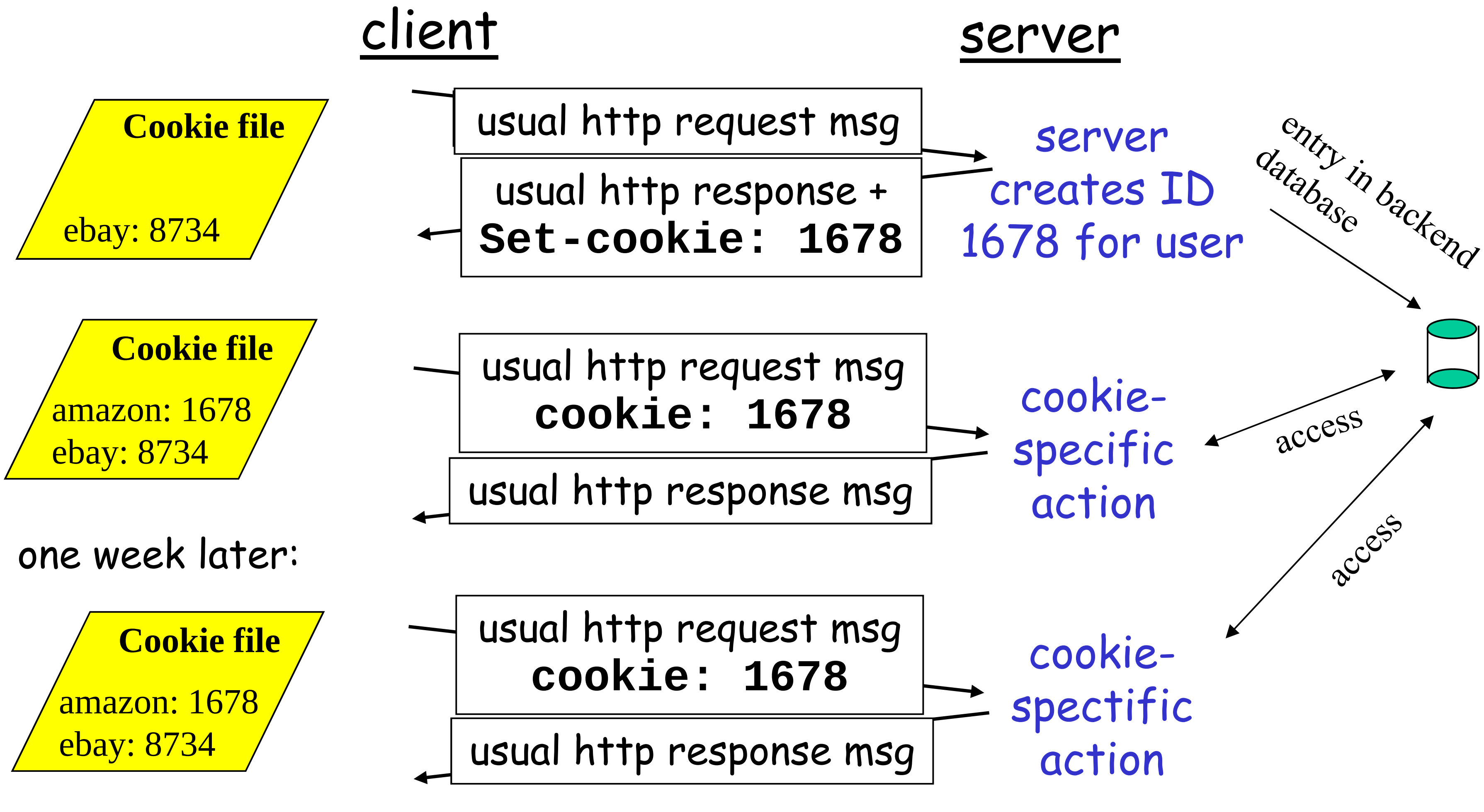
    WWW authenticate:

    header line in response



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

Cookies: keeping "state"



Cookies (continued)

What cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state

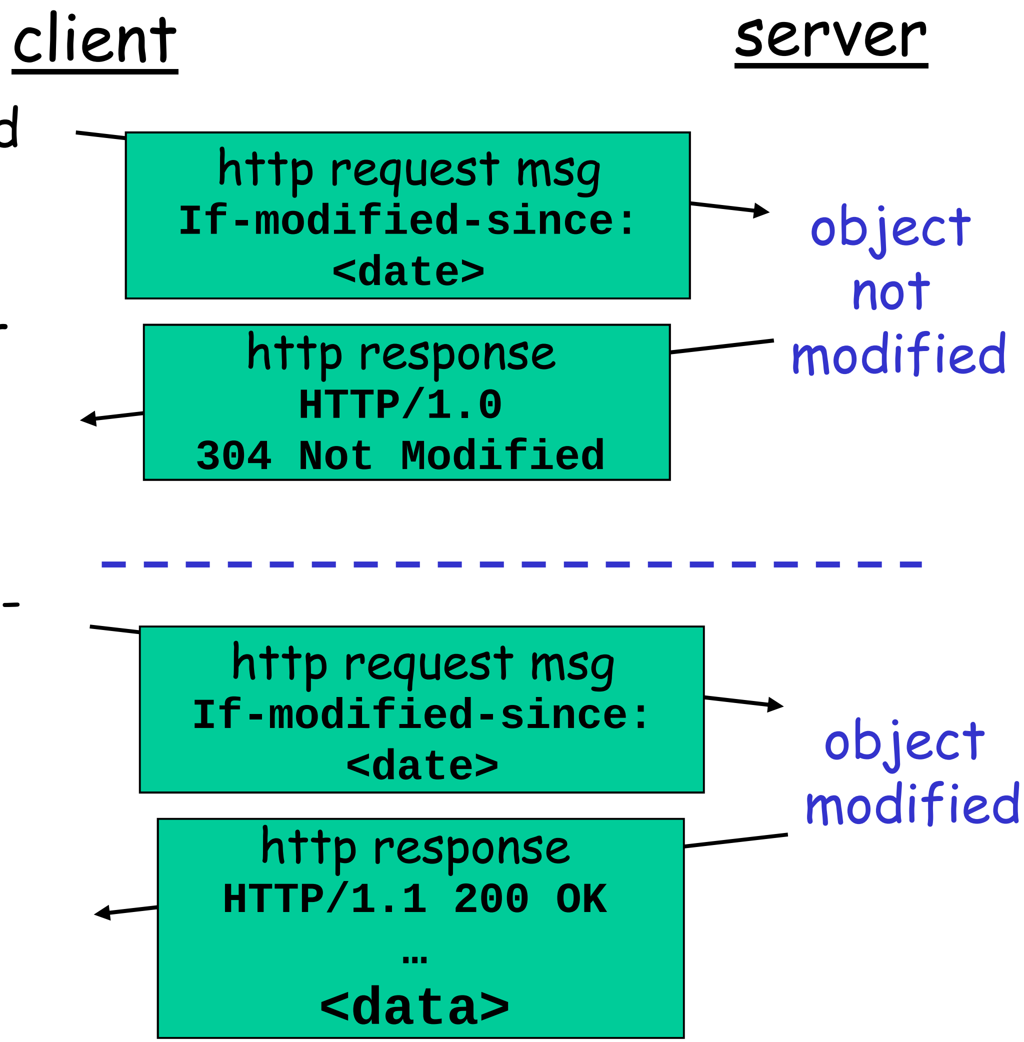
aside

Cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites
- search engines use cookies to learn yet more
- advertising companies obtain info across sites

Conditional GET: client-side caching

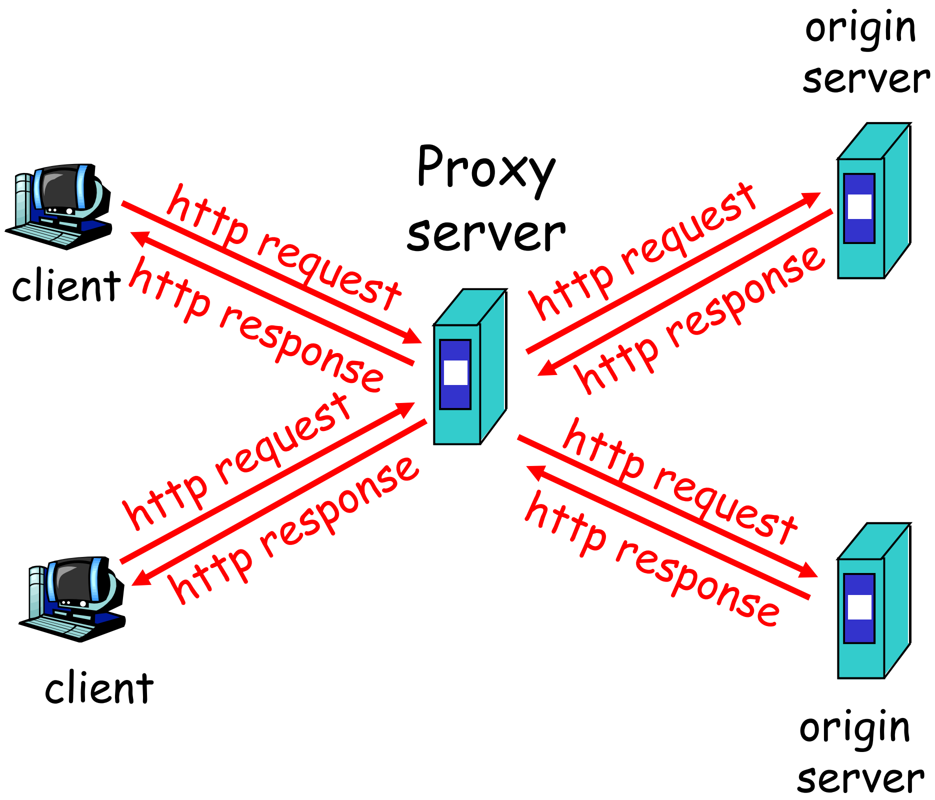
- Goal: don't send object if client has up-to-date stored (cached) version
- client: specify date of cached copy in http request
  If-modified-since: <date>
- server: response contains no object if cached copy up-to-date:
  HTTP/1.0 304 Not Modified



Web Caches (proxy server)

Goal: satisfy client request without involving origin server

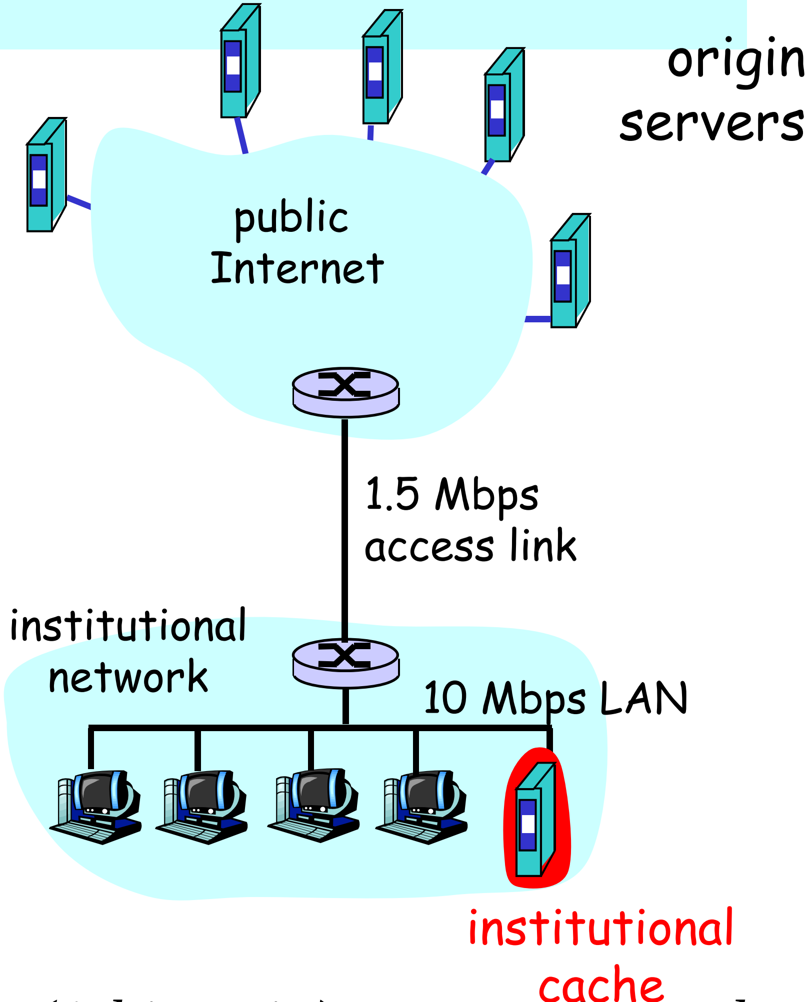
- user configures browser: Web accesses via web cache
- client sends all http requests to web cache
  - if object at web cache, web cache immediately returns object (http response)
  - else requests object from origin server (or from next cache), then returns http response to client
- Hierarchical, cooperative caching, ICP: Internet Caching Protocol



Why Web Caching?

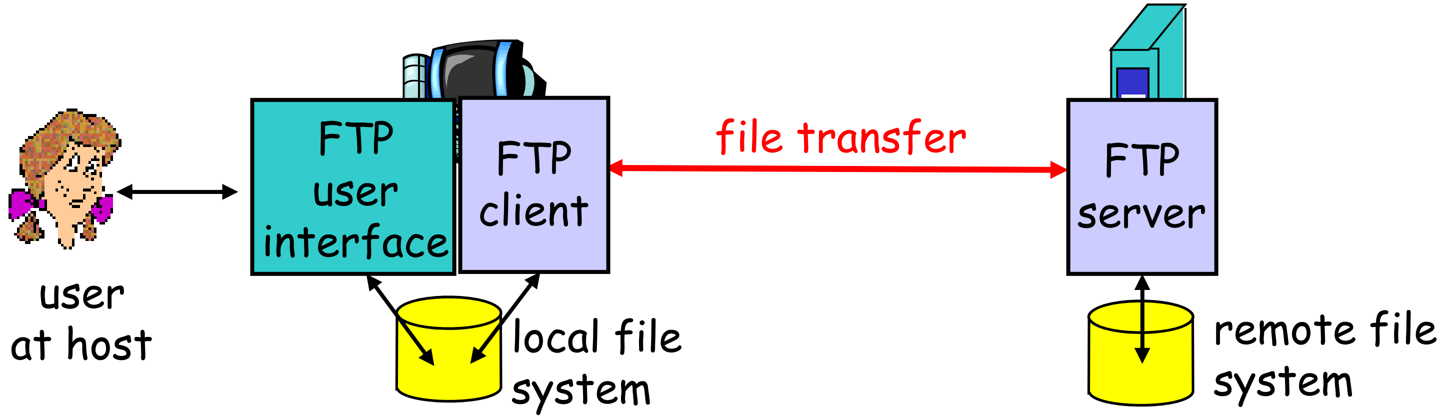
Assume: cache is "close" to client (e.g., in same network)

- smaller response time: cache "closer" to client
- decrease traffic to distant servers
  - link out of institutional/local ISP network often bottleneck
- Important for large data applications (e.g. video,...)
- Performance effect:



*E(delay)=hitRatio*LocalAccDelay + (1-hitRatio)*RemoteAccDelay*

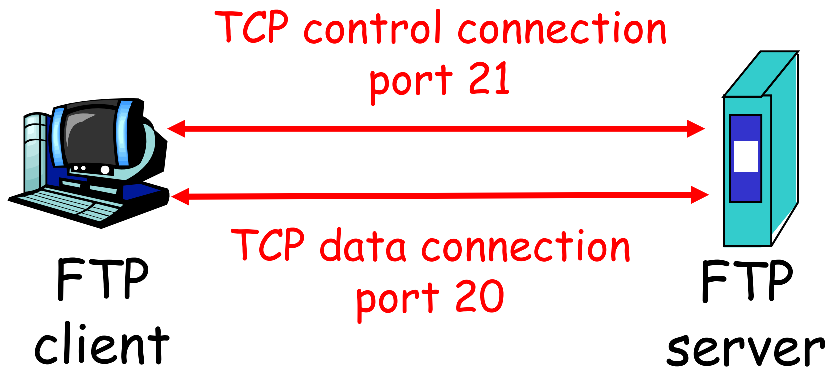
ftp: the file transfer protocol



- 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

ftp: separate control, data connections

- ftp client contacts ftp server at port 21, specifying TCP as transport protocol
- two parallel TCP connections opened:
  - control: exchange commands, responses between client, server.
    "out of band control"
  - data: file data to/from server
- ftp server maintains "state": current directory, earlier authentication



ftp commands, responses

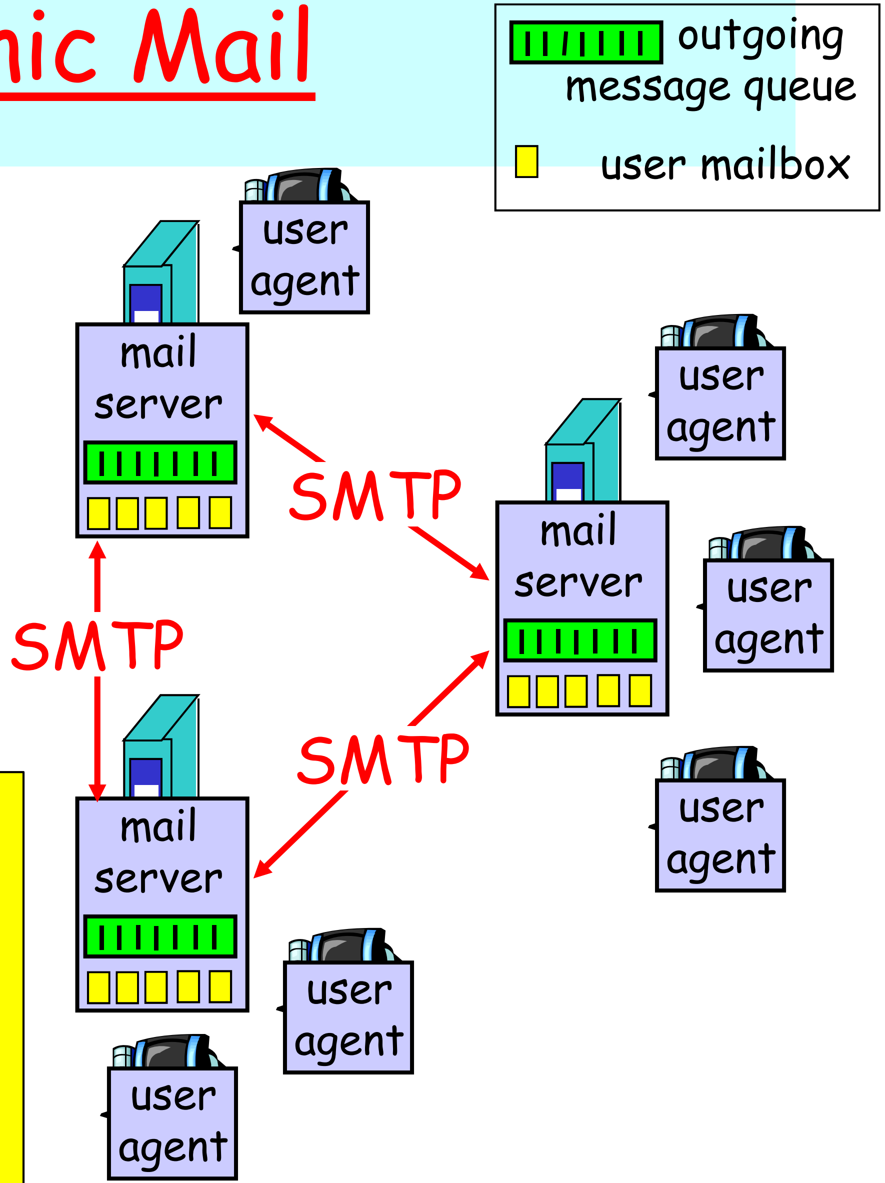
Sample commands:

- 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**

Electronic Mail



User Agent

- a.k.a. "mail reader: composing, editing, reading mail messages -e.g., Outlook, Mozzila messenger

Mail Servers

- Mailbox: incoming messages (yet to be read) 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 821, 2821]

- uses TCP to reliably transfer email msg 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

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
```

try smtp interaction for yourself:

- **`telnet servername 25`**
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

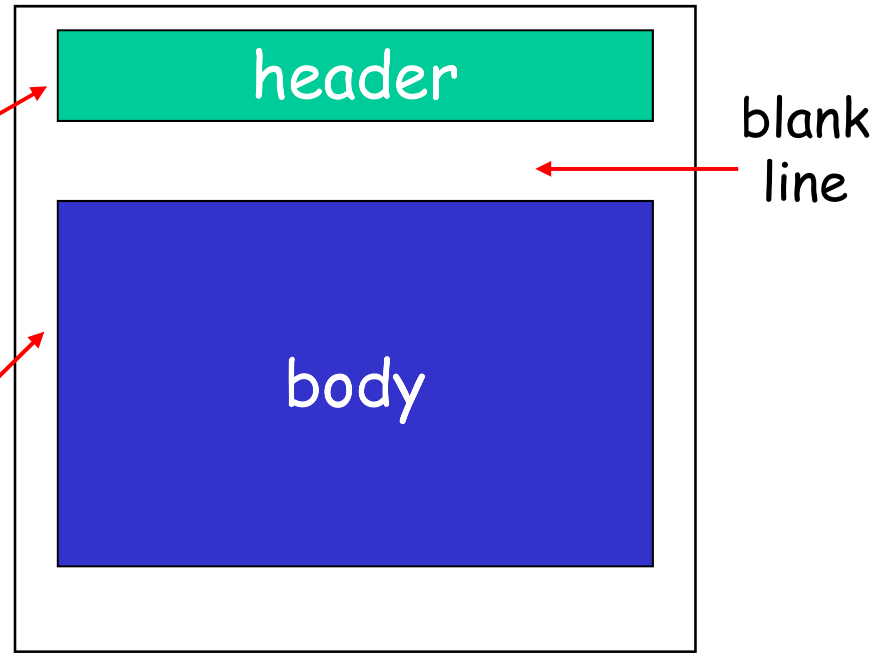
Mail message format

smtp: protocol for exchanging email msgs

RFC 822: standard for text message format:

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

  different from smtp commands!
- body
  - the "message", ASCII characters only



Message format: multimedia extensions

- MIME: multimedia mail extension, RFC 2045, 2056
- additional lines in msg header declare MIME content type

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
```

MIME version

method used to encode data

multimedia data type, subtype, parameter declaration

encoded data

(base 64: encode everything in A-Z, a-z, 0-9, +, /; good for binary

quoted-printable: 8-bit chars = "= [hd hd]" (hd= hexadecimal digit); good for ascii extensions

MIME types

Content-Type: type/subtype; parameters

Text

- example subtypes: **plain**, **html**

Image

- example subtypes: **jpeg**, **gif**

Audio

- exampe subtypes: **basic** (8-bit mu-law encoded), **32kadpcm (32 kbps coding)**

Video

- example subtypes: **mpeg**, **quicktime**

Application

- other data that must be processed by reader before "viewable"
- example subtypes: **msword**, **octet-stream**

Multipart Type

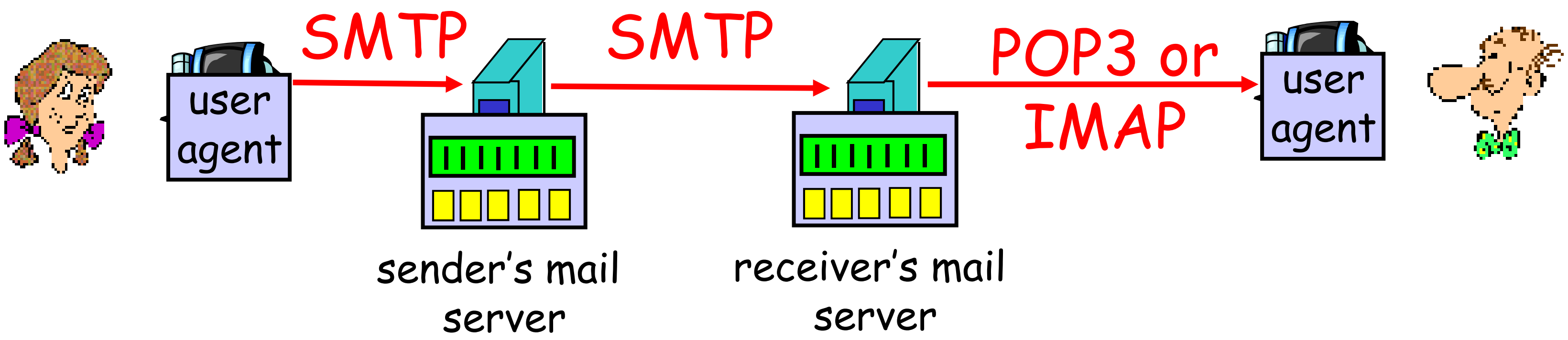
```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789

--98766789
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain

Dear Bob,
Please find a picture of a crepe.
--98766789
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
--98766789--
```

Mail access protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) and download
    - cannot re-read e-mail if he changes client
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - Manipulation, organization (folders) of stored msgs (folders, etc) on one place: the IMAP server
    - keeps user state across sessions:
  - HTTP: Hotmail , Yahoo! Mail, 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 alice
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
C: retr 2
S: <message 2 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

DNS: Domain Name System

People: many identifiers:

- SSN, name, Passport #

Internet hosts, routers: IP address (32 bit) - used for addressing datagrams (129.16.237.85)

- "name", e.g., (www.cs.chalmers.se)- used by humans
- name (alphanumeric addresses) hard to process @ router

Q: map between IP addresses and name ?

DNS: 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"
- More services by DNS:
  - alias host names, i.e. mnemonic → canonical (more complex) name
  - load distribution: different canonical names, depending on who is asking
- The Internet Corporation for Assigned Names and Numbers (http://www.icann.org/) and Domain Name Supporting Organization main coordinators

DNS name servers

Why not centralize DNS?

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

doesn't *scale!*

local name servers:

- each ISP, company has one
- host DNS query first goes to local name server; acts as proxy/cache

root name servers: contacts authoritative name server if name mapping not known (~ dozen root name servers worldwide)

Top-level domain (TLD) servers: responsible for (e.g. knowing the authoritative name servers) com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.

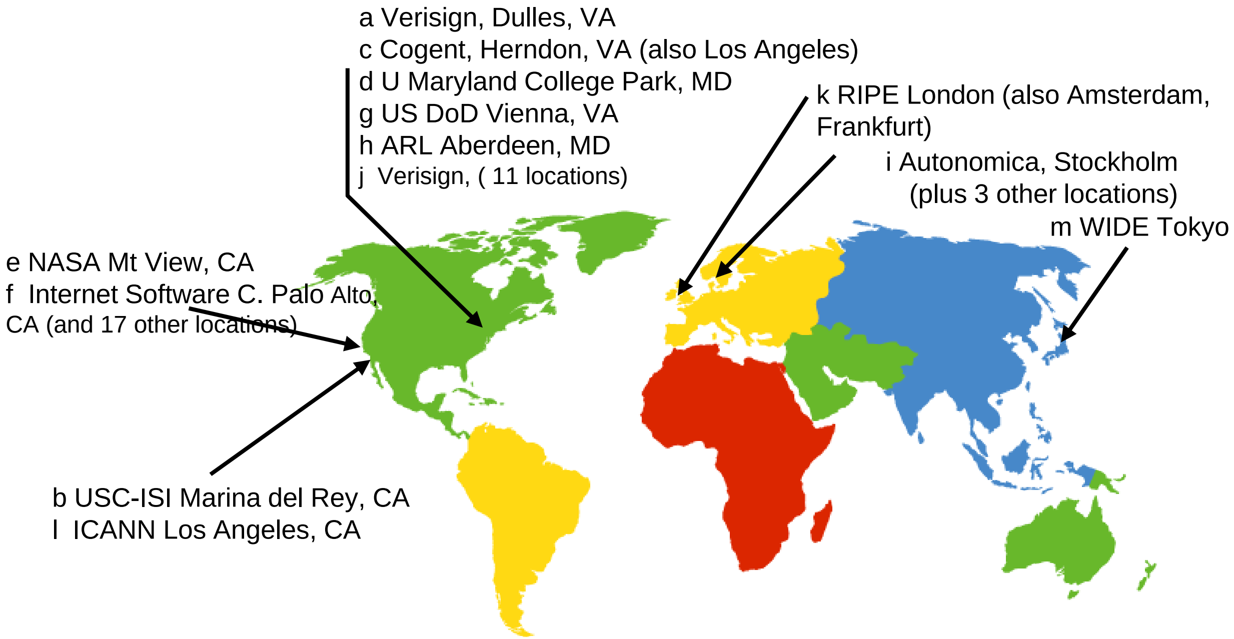
authoritative name server:

- for a host: stores that host's IP address, name

http://www.youtube.com/watch?v=2ZUxoi7YNgs&feature=related

DNS: Root name servers

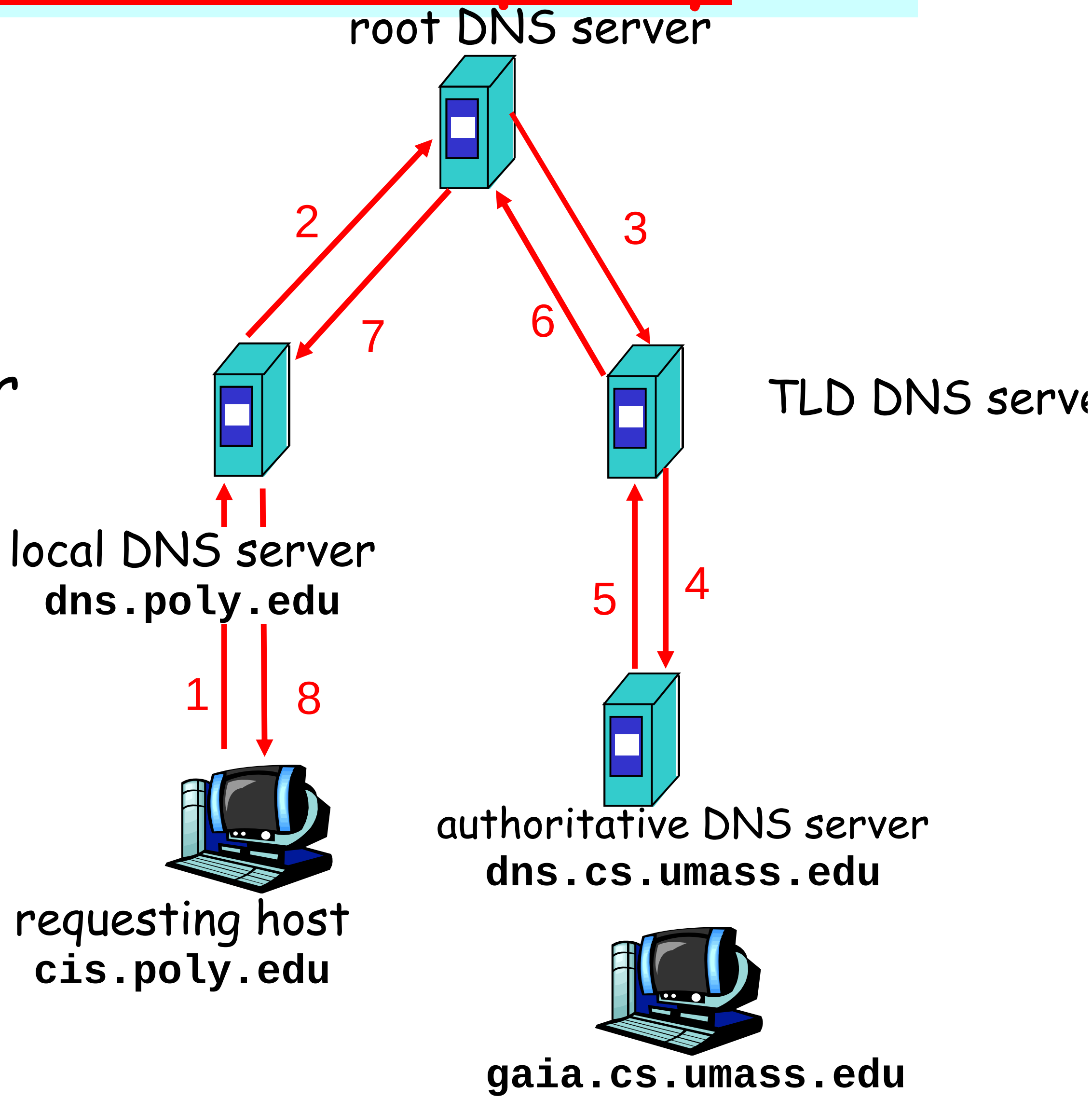
- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server



13 root name servers worldwide

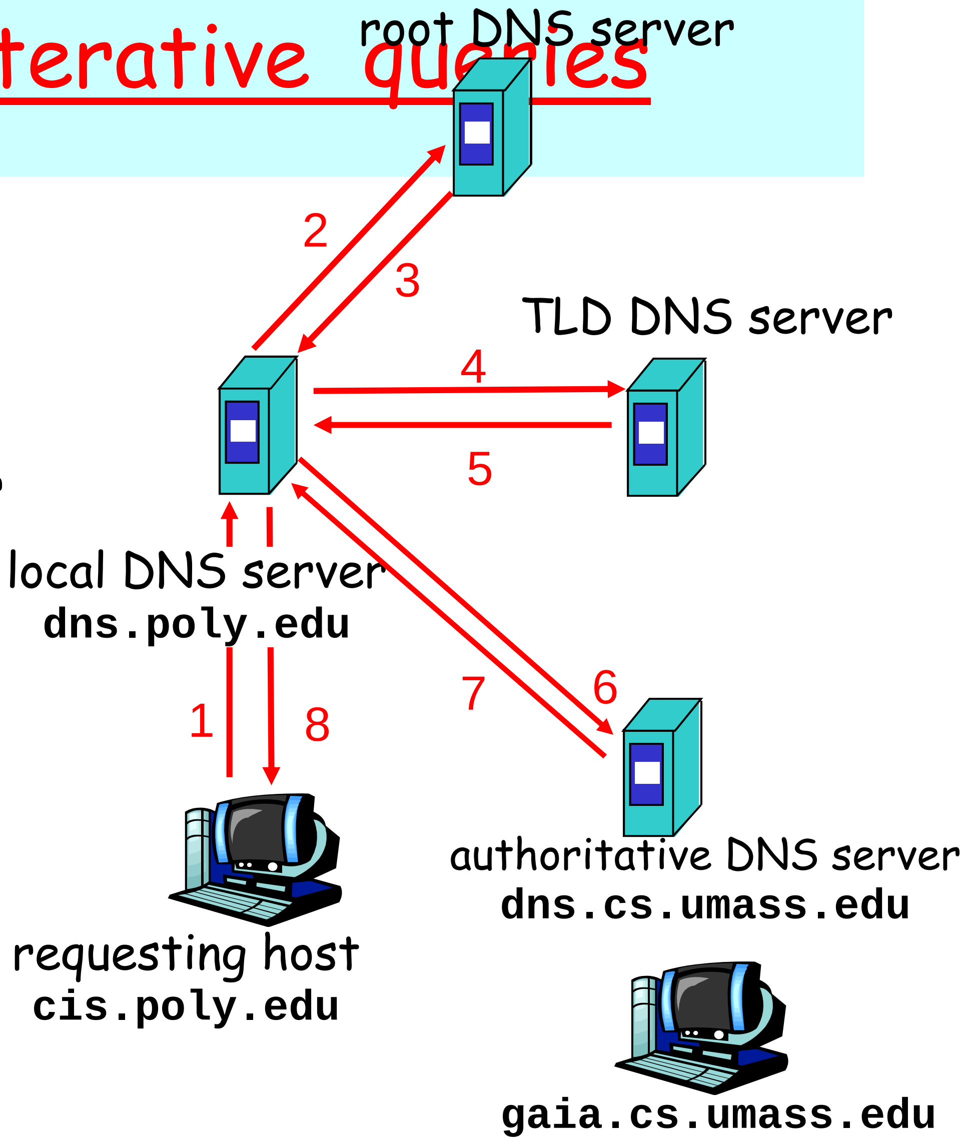
Example: recursive query

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



Recursive vs iterative 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 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
  - **name** is an alias name
  - **value** is canonical name
- Type=MX
  - **value** is hostname of mailserver associated with **name**

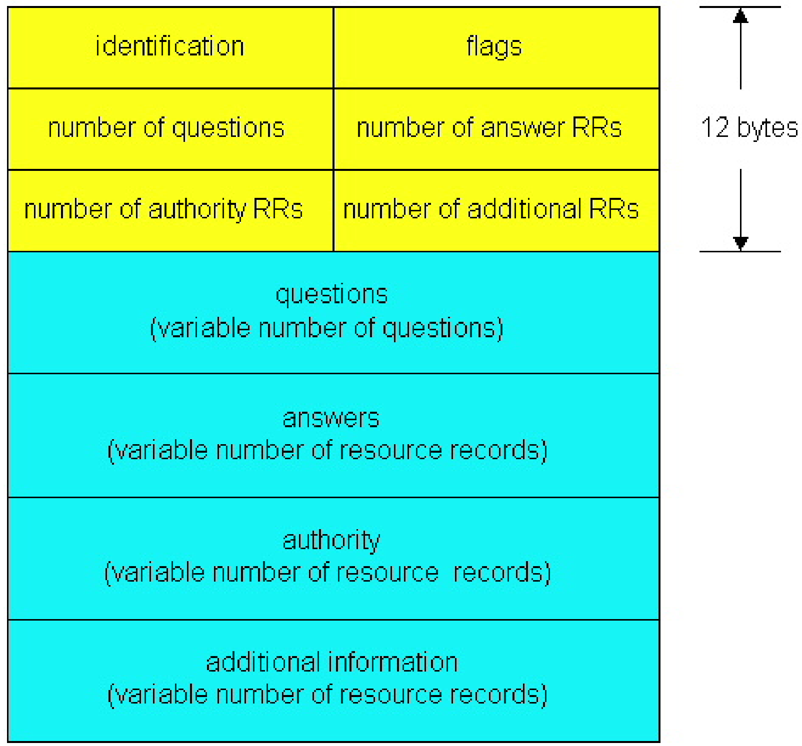
ttl = time to live

DNS protocol, messages

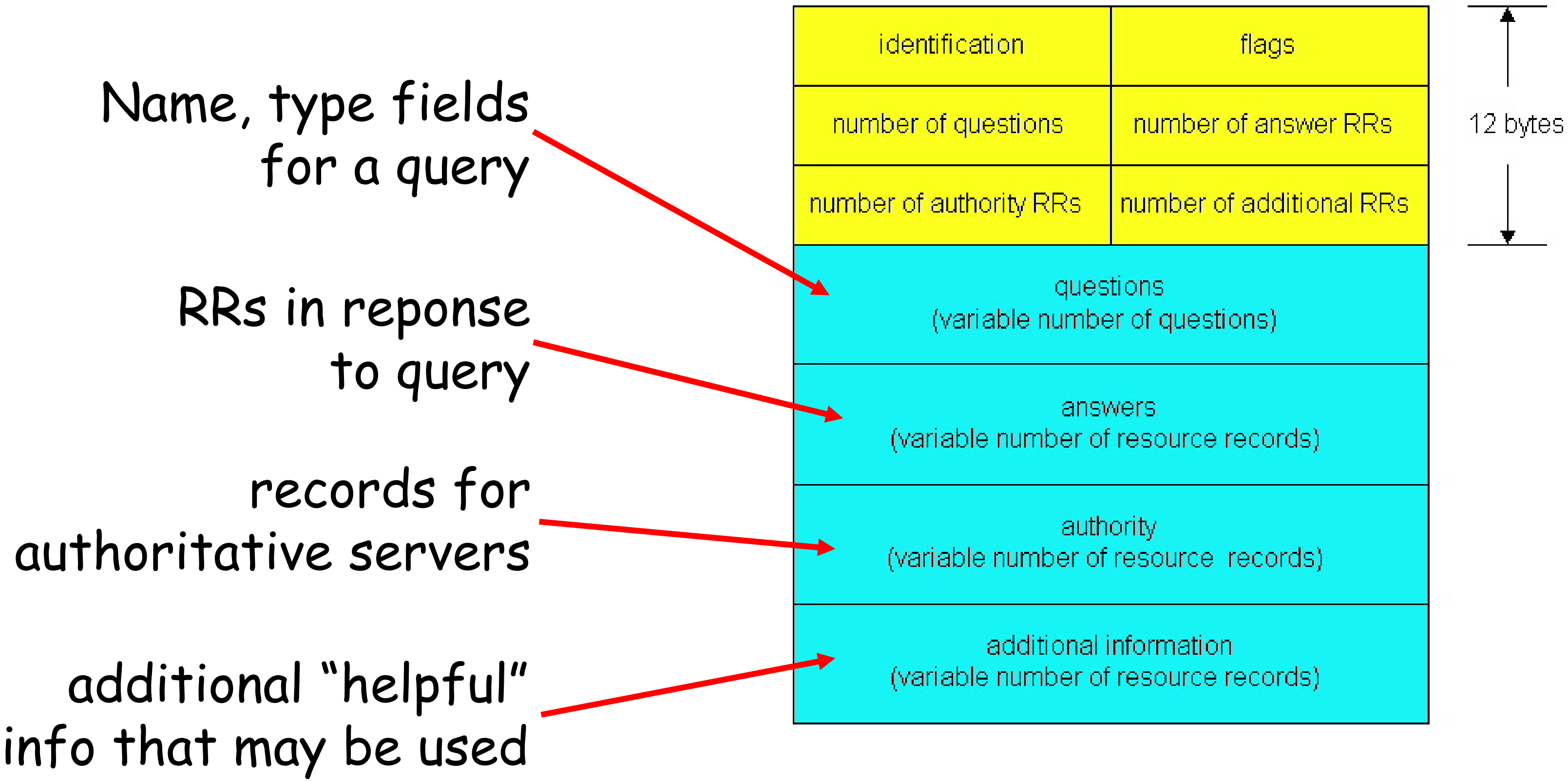
DNS protocol : *query* and *reply* messages, both with same *message format*

msg header

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



DNS protocol, messages



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 e.g. www.networkuptopia.com and Type MX record for e.g. mail.networkutopia.com

DNS: caching and updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time
- update/notify mechanisms (and more, incl. security) cf.
  - RFC 2136, 3007 (ddns)
  - http://www.ietf.org/html.charters/dnsext-charter.html
- http://www.youtube.com/watch?v=Xau_jPGeJ24

To come later on (after all "layers")

- Peer-to-peer (p2p) applications