

Course on Computer Communication and Networks

Lecture 3

Chapter 2: Application-layer, basic applications

EDA344/DIT 423, CTH/GU

Based on the book Computer Networking: A Top Down Approach, Jim Kurose, Keith Ross, Addison-Wesley.

Chapter 2: Application Layer

Chapter goals:

- conceptual + implementation aspects of basic application protocols
- specific protocols:
 - http, smtp, pop, dns,

(p2p & streaming, CDN: later in the course)

Applications and application-layer protocols

Application: communicating, distributed processes

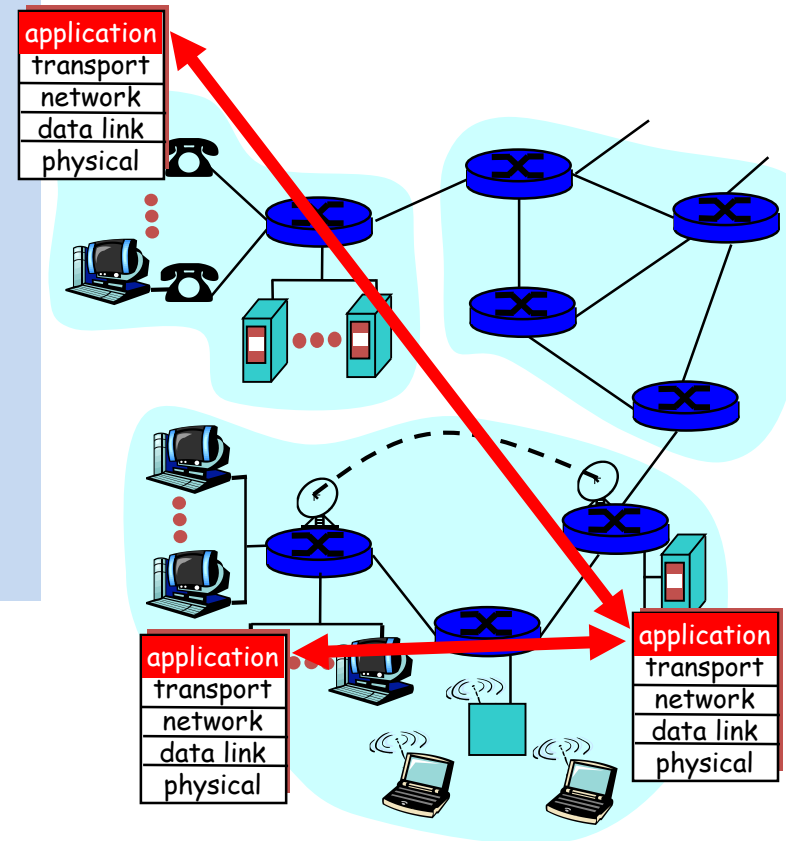
- running in network hosts in “user space”
- e.g., email, file transfer, the Web

Application-layer protocols

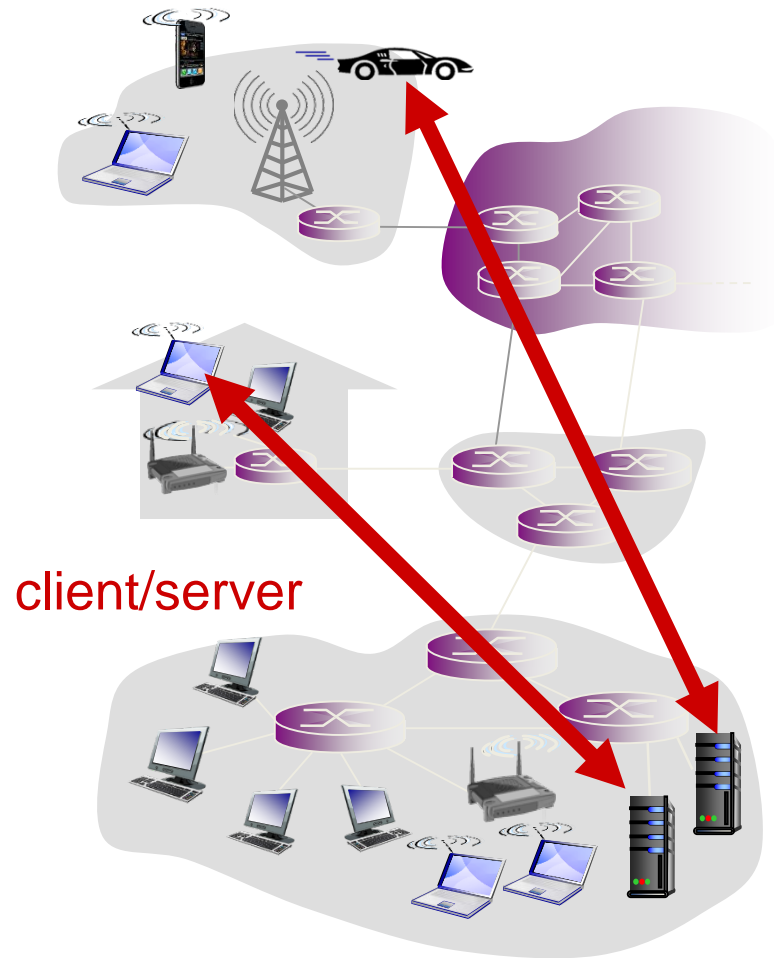
- Are only one “piece” of an application -others are e.g. **user agents**.
 - Web: browser
 - E-mail: mail reader
 - streaming audio/video: media player

App-layer protocols:

- **define** messages exchanged and actions taken
- **use services** provided by lower layer protocols



Client-server architecture



server:

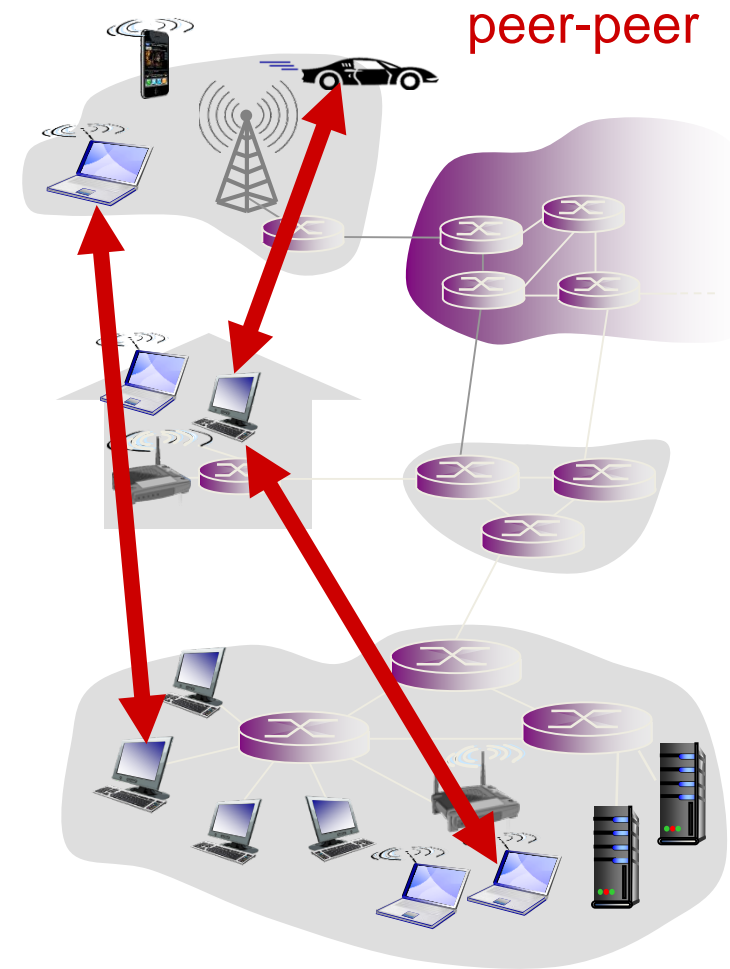
- always-on
- permanent host address
- clusters of servers for scaling

clients:

- communicate with server
- may be intermittently connected
- may have dynamic host addresses

Peer2Peer architecture

- *no* always-on server
- peers request service from other peers, provide service in return
- peers are intermittently connected and may change addresses
 - complex management



Roadmap



- Addressing and Applications needs from transport layer
- Http
 - General description and functionality
 - Authentication, cookies and related aspects
 - Caching and proxies
- SMTP (POP, IMAP)
- DNS

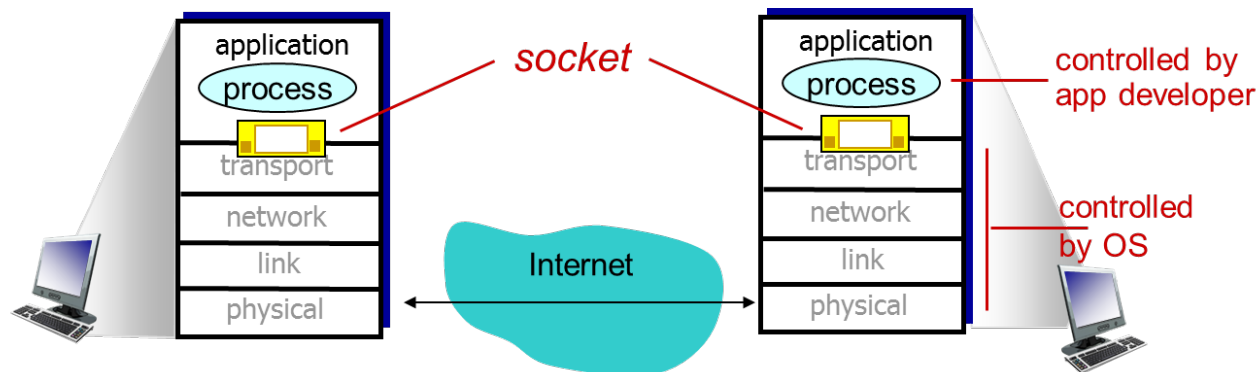
Addressing, sockets

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** (unique) of host running other process
- **“port number”** - allows receiving host to determine to which local process the message should be delivered



Roadmap



- Addressing and Applications needs from transport layer
- Http
 - General description and functionality
 - Authentication, cookies and related aspects
 - Caching and proxies
- SMTP (POP, IMAP)
- DNS

Transport service requirements: common apps

application	data loss	throughput	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: 5kbps-1Mbps video: 10kbps-5Mbps	y, 100' s msec
interactive games	loss-tolerant	few kbps up	y, 100' s msec
text messaging	no loss	elastic	yes and no

Services to upper layer by Internet transport protocols

UDP service:

- *connectionless*
- *Unreliable, “best-effort” transport* between sending and receiving process
- *does not provide*: timing, or bandwidth guarantees



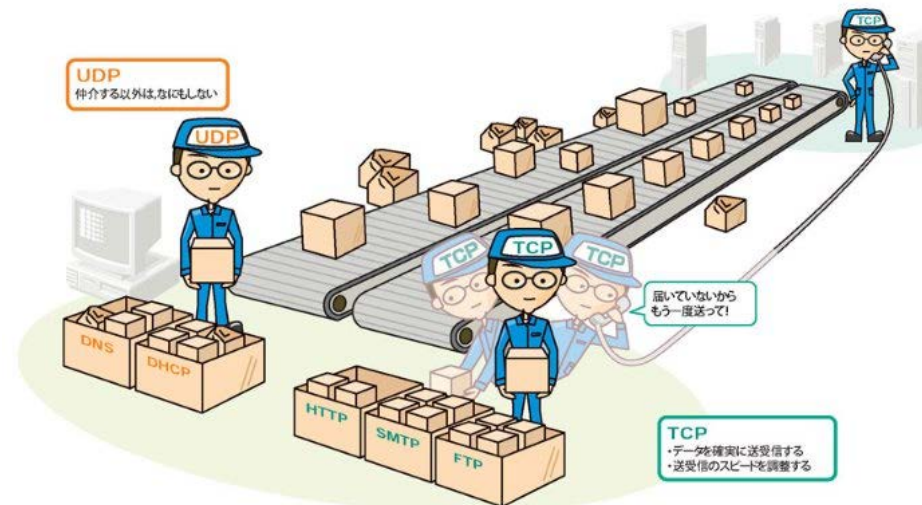
UDP

TCP service:

- *connection-oriented reliable transport* between sending and receiving process
 - correct, in-order delivery of data
 - setup required between client, server
- *does not provide*: timing, bandwidth guarantees



TCP



Internet apps: application, transport protocols

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

Roadmap



- Addressing and Applications needs from transport layer
- Http
 - General description and functionality
 - Authentication, cookies and related aspects
 - Caching and proxies
- SMTP (POP, IMAP)
- DNS

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*, e.g.,

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

host name

path name

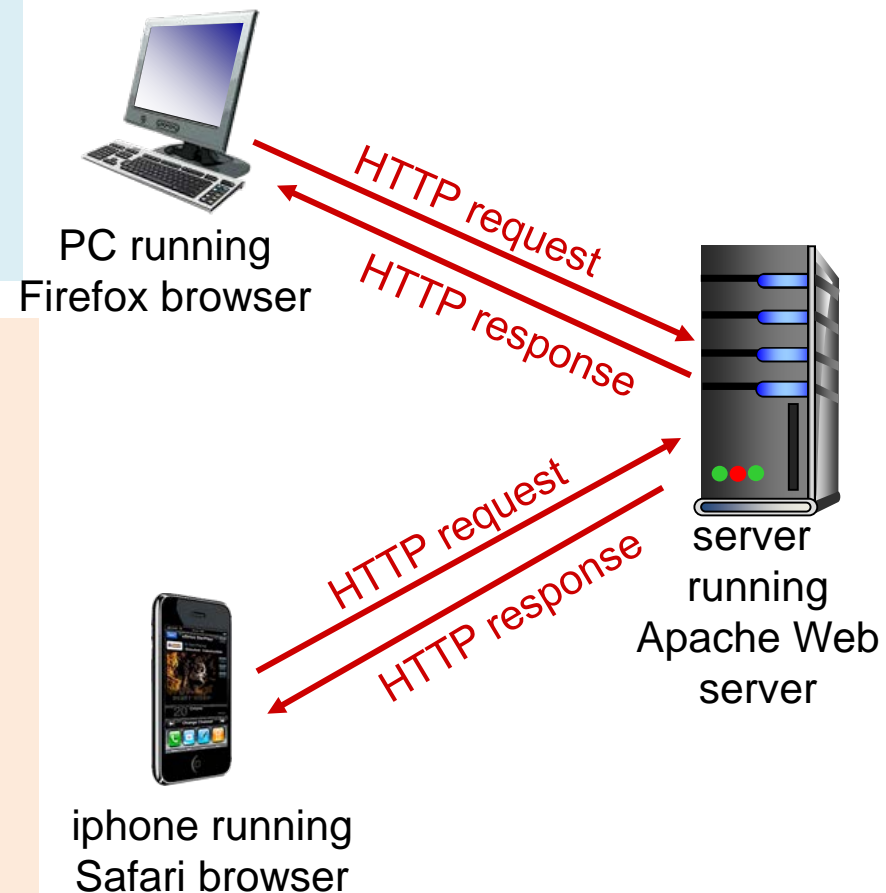
HTTP: **hypertext transfer protocol** overview

Web' s application layer protocol

- *http client*: web browser; requests, receives, displays Web objects
- *http server*: Web server sends objects

uses TCP:

1. client initiates TCP connection to server, **port 80**
2. server accepts TCP connection
3. HTTP messages (application-layer protocol messages) exchanged
4. TCP connection closed



http example

user enters URL

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

(contains text,
references to 10 jpeg images)

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

2. **http server** at host `www.someSchool.edu` waiting for TCP connection at port 80. “accepts” connection, notifying client

3a. client sends http *request message* (containing URL) into TCP connection socket

3b. server receives request, forms *response message* with requested object (`someDepartment/home.index`), sends message into socket

3c. client receives response msg with file, displays html. Parsing html file, finds 10 referenced jpeg objects

4. server closes TCP connection.

Steps 1-5 repeated for each of 10 jpeg objects

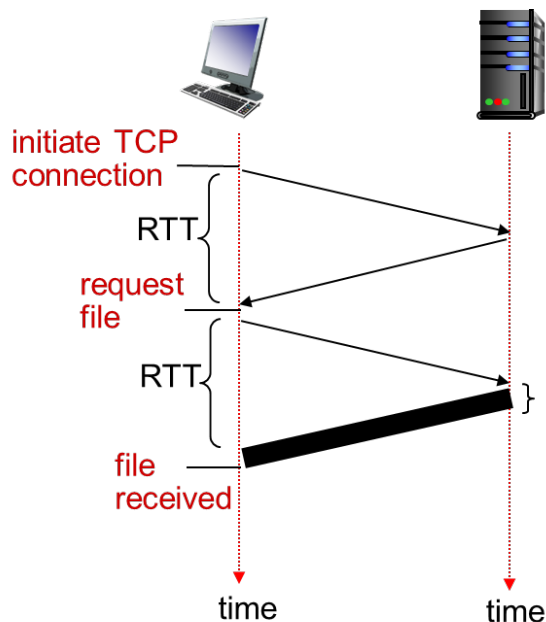
time



Non-persistent and persistent http

Non-persistent (http1.0)

- server parses request, responds, closes TCP connection
- *non-persistent HTTP response time = $2RTT + \text{file transmission time}$*
- **new TCP connection for each object** => extra overhead per object



Persistent

- **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 (http 1.1)**
 - update http/2: fetches in priority ordering

With both, browsers can open parallel sessions

HTTP request message

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

The diagram illustrates the structure of an HTTP request message. It shows a sequence of lines: a request line, followed by several header lines, and a final blank line. Annotations with arrows point to specific parts of the message:

- request line (GET, POST, HEAD commands):** Points to the first line: `GET /index.html HTTP/1.1\r\n`
- header lines:** A bracket groups the following lines: `Host: www-net.cs.umass.edu\r\n`, `User-Agent: Firefox/3.6.10\r\n`, `Accept: text/html,application/xhtml+xml\r\n`, `Accept-Language: en-us,en;q=0.5\r\n`, `Accept-Encoding: gzip,deflate\r\n`, and `Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n`
- carriage return, line feed at start of line indicates end of header lines:** Points to the blank line `\r\n` that separates the headers from the body.
- carriage return character** and **line-feed character:** Arrows point to the `\r` and `\n` characters at the end of the first line.

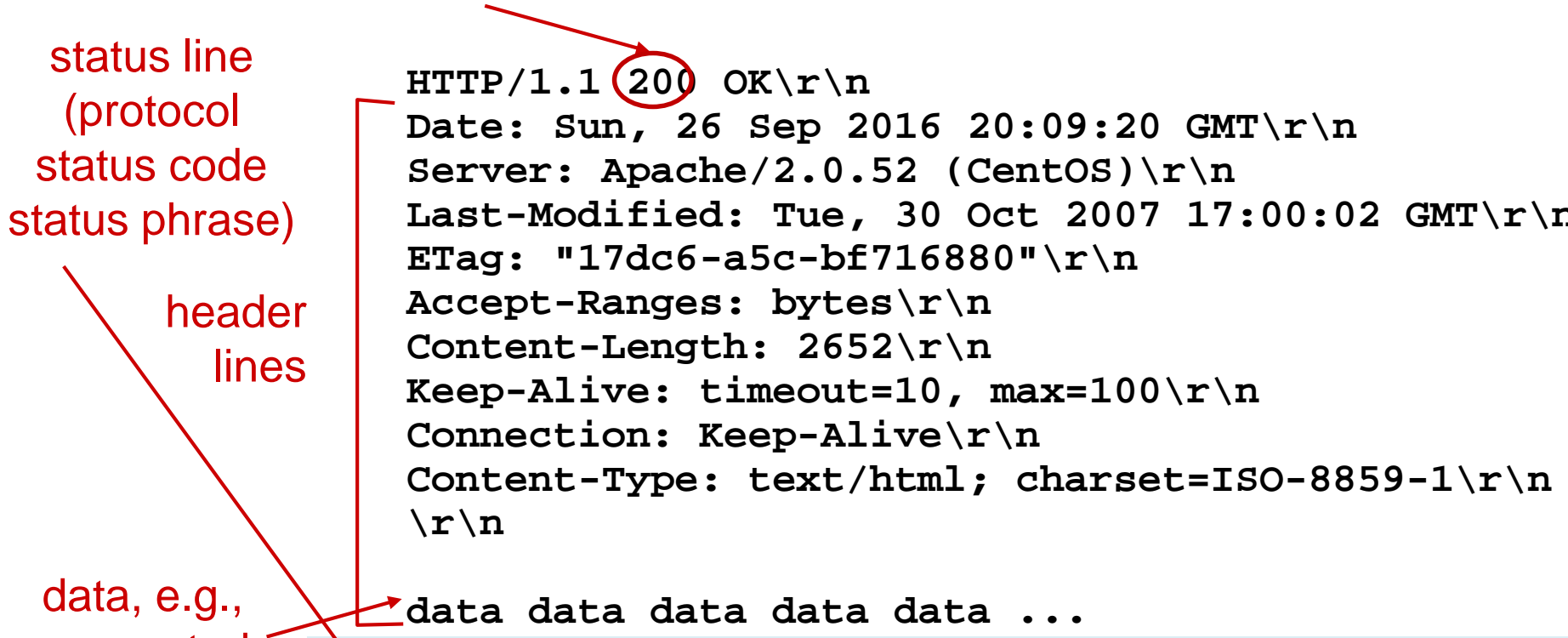
```
GET /index.html HTTP/1.1\r\nHost: www-net.cs.umass.edu\r\nUser-Agent: Firefox/3.6.10\r\nAccept: text/html,application/xhtml+xml\r\nAccept-Language: en-us,en;q=0.5\r\nAccept-Encoding: gzip,deflate\r\nAccept-Charset: ISO-8859-1,utf-8;q=0.7\r\nKeep-Alive: 115\r\nConnection: keep-alive\r\n\r\n
```

HTTP response message

status line
(protocol
status code
status phrase)

header
lines

data, e.g.,
requested
HTML file



```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2016 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
```

The diagram shows an HTTP response message. A red arrow points from the 'status line' label to the first line 'HTTP/1.1 200 OK\r\n', where the status code '200' is circled. Another red arrow points from the 'header lines' label to the subsequent lines of headers. A third red arrow points from the 'data, e.g., requested HTML file' label to the 'data data data data data ...' line.

data data data data data ...

200 OK: request succeeded, requested object in this msg

301 Moved Permanently: requested object moved, new location specified later in this message (Location:)

400 Bad Request: request message not understood

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 a Web server:

```
telnet cis.poly.edu 80
```

opens TCP connection to port 80
(default HTTP server port) at cis.poly.edu.
anything typed in sent
to port 80 at cis.poly.edu

2. type in a GET HTTP request:

```
GET /~ross/ HTTP/1.1  
Host: cis.poly.edu
```

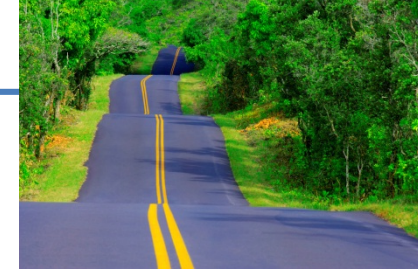
by typing this in (hit carriage
return twice), you send
this minimal (but complete)
GET request to HTTP server

3. look at response message sent by HTTP server!

Topic of the programming assignment

1 (multithreaded server,
ed! You will learn useful stuff

Roadmap



- Addressing and Applications needs from transport layer
- Http
 - General description and functionality
 - Authentication, cookies and related aspects
 - Caching and proxies
- SMTP (POP, IMAP)
- DNS

HTTP is “stateless”

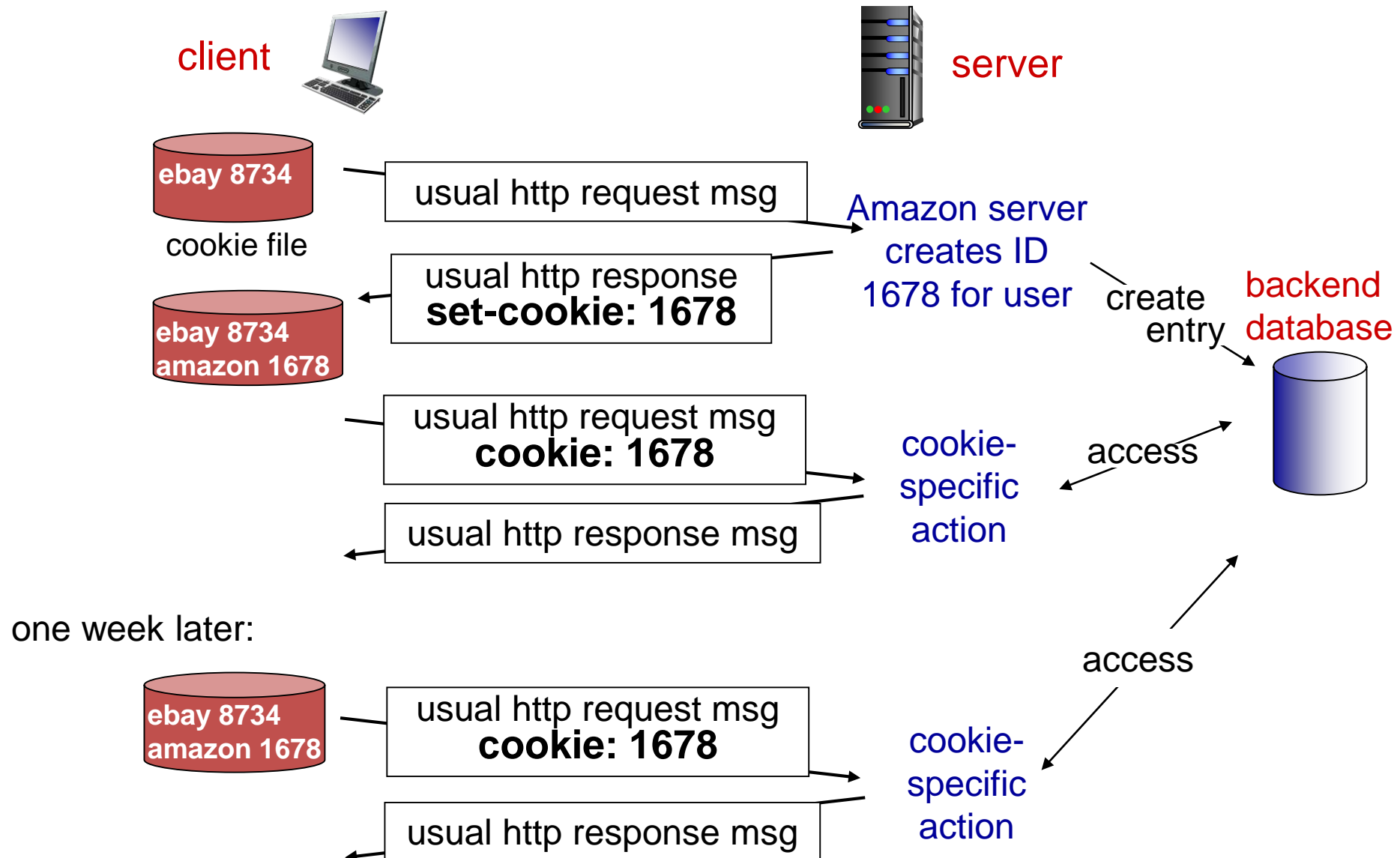
HTTP server maintains no information about past client requests

protocols that maintain “state” are *aside* complex!

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

Q: how do web applications keep state though?

Cookies: keeping “state”



Cookies (continued)

cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state

Cookies and privacy:

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

aside

Roadmap

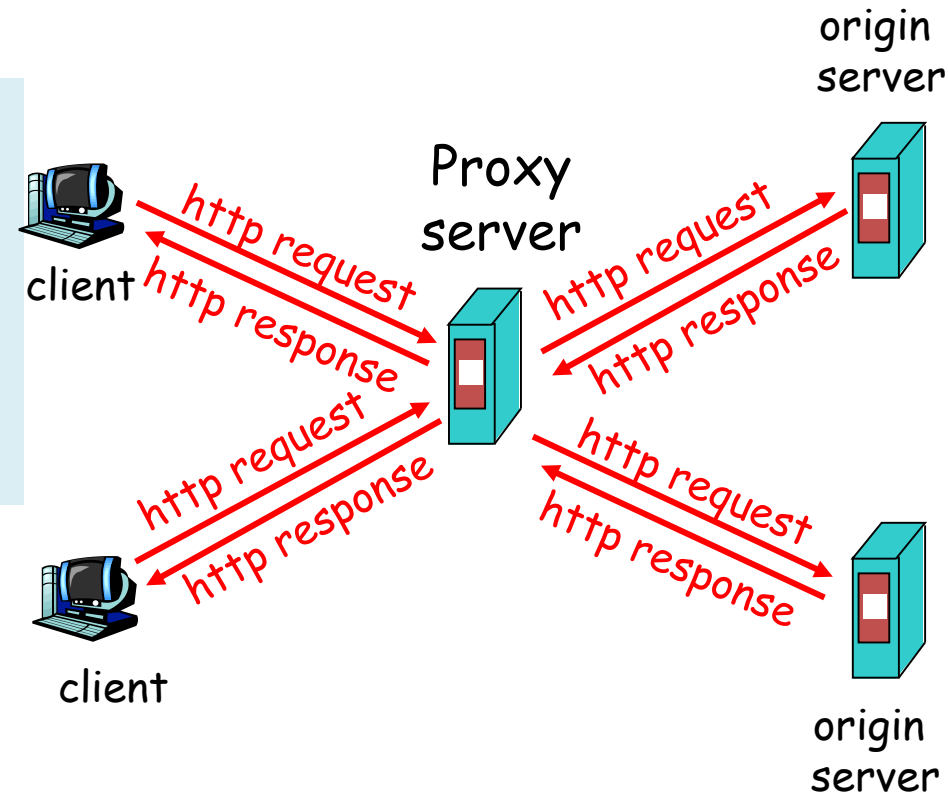


- Addressing and Applications needs from transport layer
- Http
 - General description and functionality
 - Authentication, cookies and related aspects
 - Caching and proxies
- SMTP (POP, IMAP)
- DNS

Web Caches (proxy server)

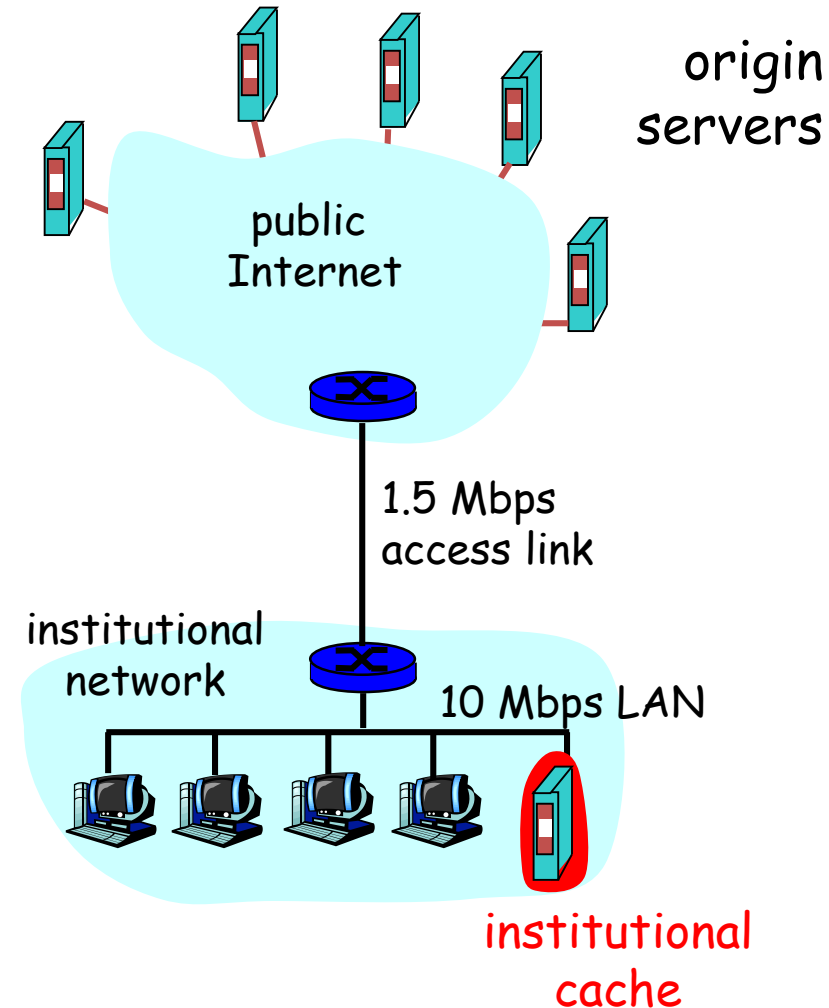
Goal: satisfy client request without involving origin server

- user configures browser: Web accesses via web cache
- client sends all http requests to web cache; the cache(proxy) server acts as usual caches do
- Hierarchical, cooperative caching, ICP: Internet Caching Protocol (RFC2187)



Why Web Caching?

- Assume:** cache is close to client (e.g., in same network)
- smaller response time
 - decrease traffic to distant servers
 - link out of institutional/local ISP network can be bottleneck
 - Important for large data applications (e.g. video,...)



Performance effect:

$$E(delay) = hitRatio * LocalAccessDelay + (1 - hitRatio) * RemoteAccessDelay$$

Roadmap



- Addressing and Applications needs from transport layer
- Http
 - General description and functionality
 - Authentication, cookies and related aspects
 - Caching and proxies
- SMTP (POP, IMAP)
- DNS

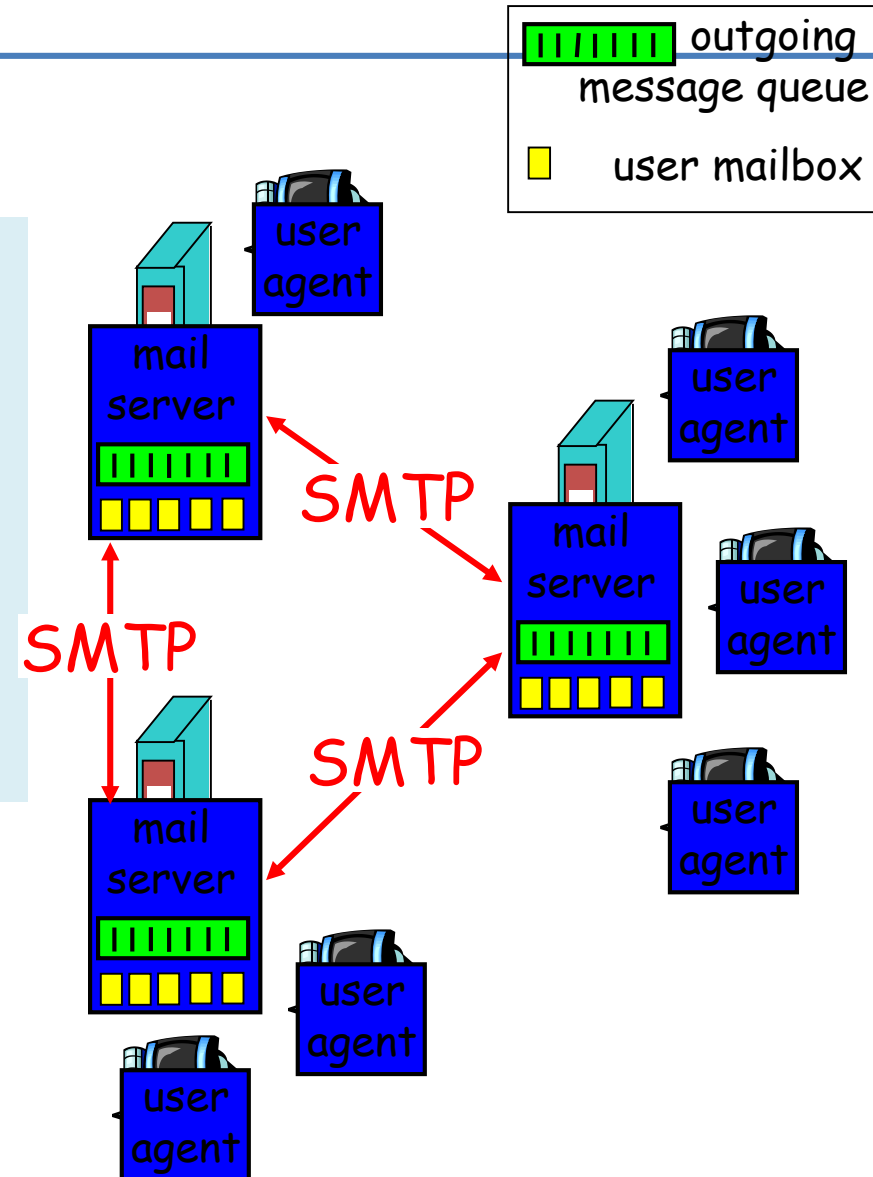
Electronic Mail

User Agent

- a.k.a. “mail reader”: composing, editing, reading mail messages -e.g., Outlook, gmail

Mail Servers

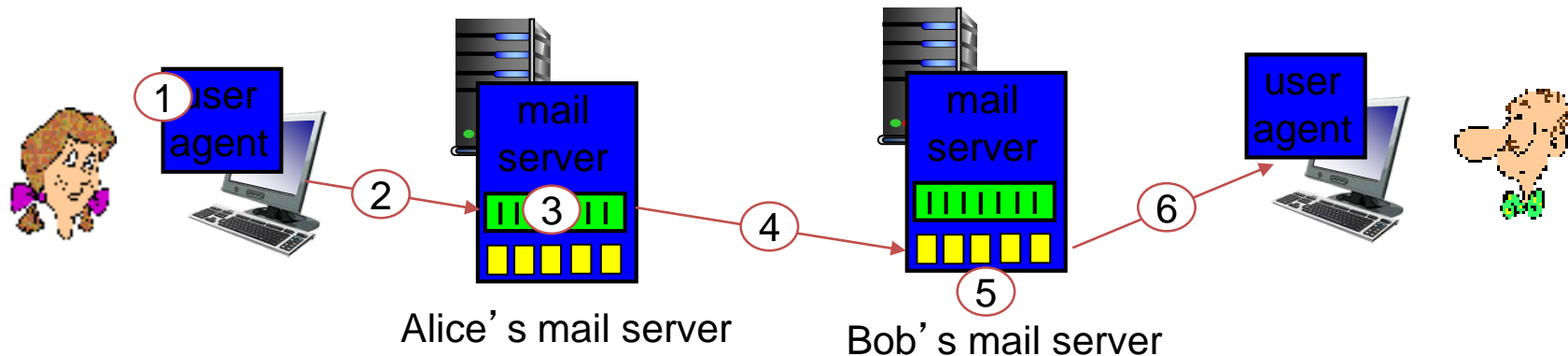
- Simple Mail Transfer Protocol (SMTP) between mail servers to send email messages
 - “client”: sending mail server
 - server: receiving mail server



Scenario: Alice sends message to Bob

- 1) Alice, UA: message “to” bob@someschool.edu
- 2) Alice, UA: sends message to her mail server's queue
- 3) Alice, mail server: TCP connection with Bob's mail server (acting as a client of SMTP)

- 4) Alice's mail server sends Alice's message over the TCP connection
- 5) Bob's mail server places the message in Bob's mailbox
- 6) Bob invokes his UA 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 (RFC 2821) uses TCP, port 25
- three phases
 - handshaking (greeting)
 - transfer of messages
 - closure

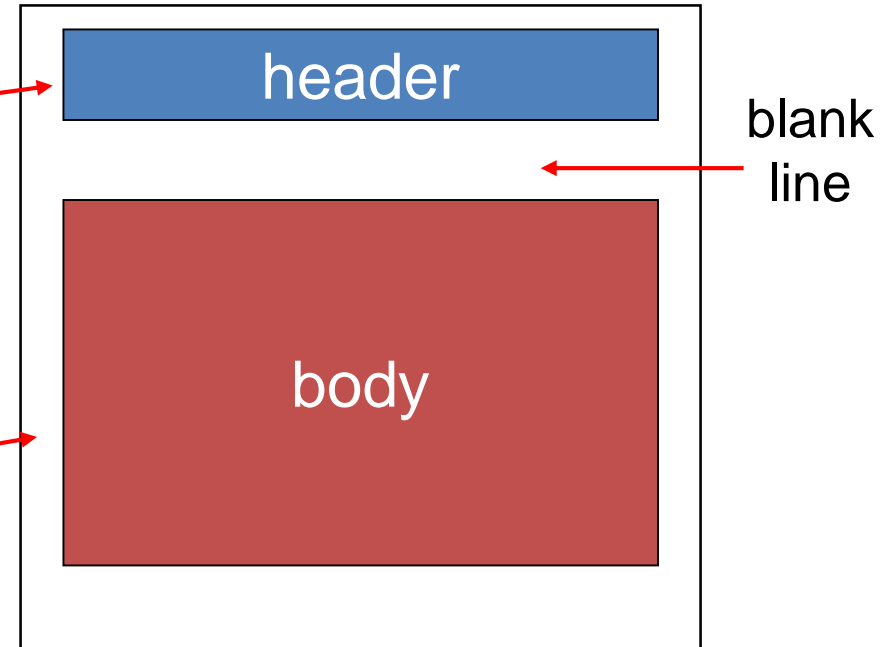
You can try it out through
telnet servername 25

- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

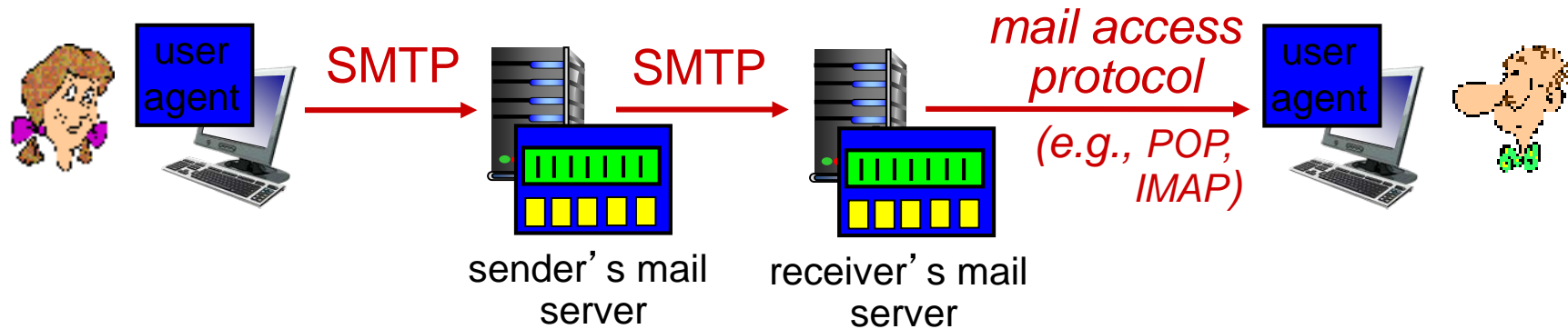
SMTP & Mail message format

RFC 822: standard for text message format:

- header lines, e.g.,
 - To:, From:, Subject:*different from* SMTP MAIL FROM, RCPT TO: commands
- Body: the “message”
 - ASCII 7-bit characters only



Mail access protocols



- **SMTP:** delivery/storage to receiver's server
- mail access protocol: retrieval from server
 - **POP:** Post Office Protocol [RFC 1939]: authorization, download
 - **IMAP:** Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored msgs on server

Roadmap



- Addressing and Applications needs from transport layer
- Http
 - General description and functionality
 - Authentication, cookies and related aspects
 - Caching and proxies
- SMTP (POP, IMAP)
- DNS

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

Q: map between IP addresses and name ?

Hostname to IP address translation

- **Example:** `www.chalmers.se` `129.16.71.10`
- File with mapping may be edited on the system
 - eg Unix: `/etc/hosts` - Windows: `c:\windows\system32\drivers\etc\hosts`
 - Example of an entry manually entered in the file: `"129.16.20.245 fibula.ce.chalmers.se fibula"`

Does not scale for all possible hosts, hard to change

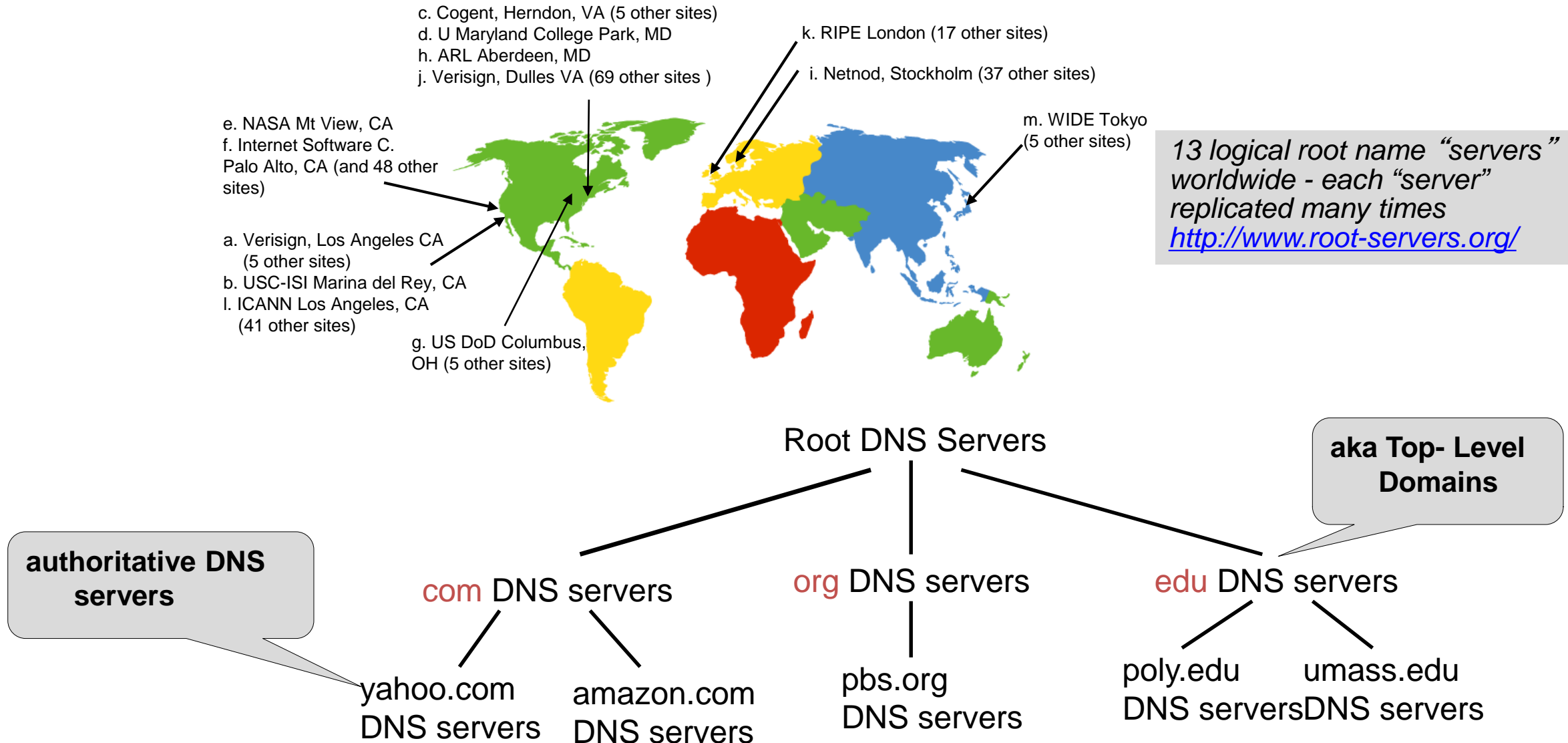
- All hosts need one copy of the file
- Impossible on the Internet

Alternative: DNS, a large **distributed** database
DNS – **D**omain **N**ame **S**ystem

why not centralize DNS?

- single point of failure
- traffic volume
- maintenance

DNS: a distributed, hierarchical database



DNS name resolution example

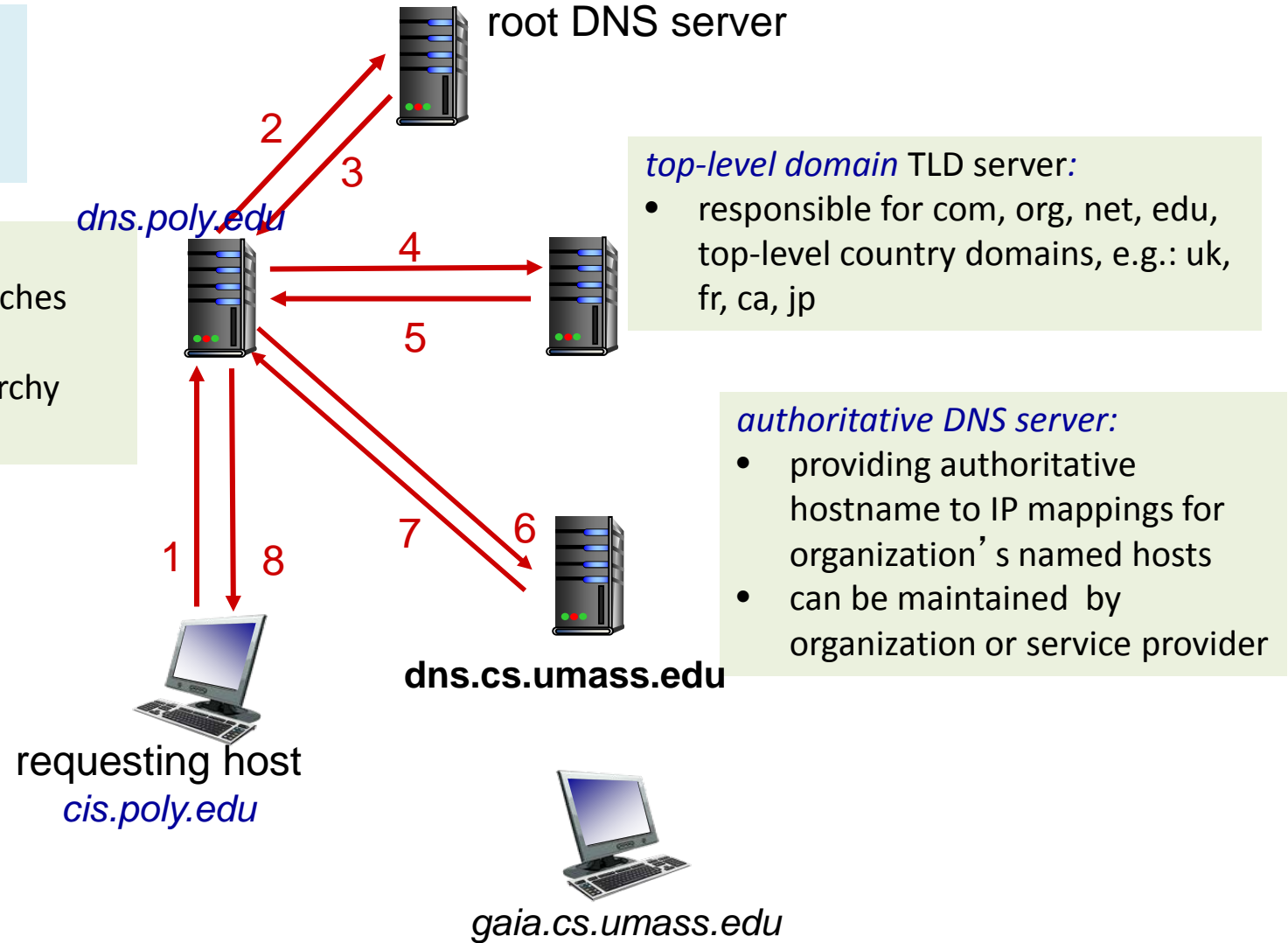
- host at cis.poly.edu wants IP address for gaia.cs.umass.edu

Local name server

- acts as proxy for clients, caches entries for TTL
- Sends queries to DNS hierarchy
- each ISP has one

iterated query:

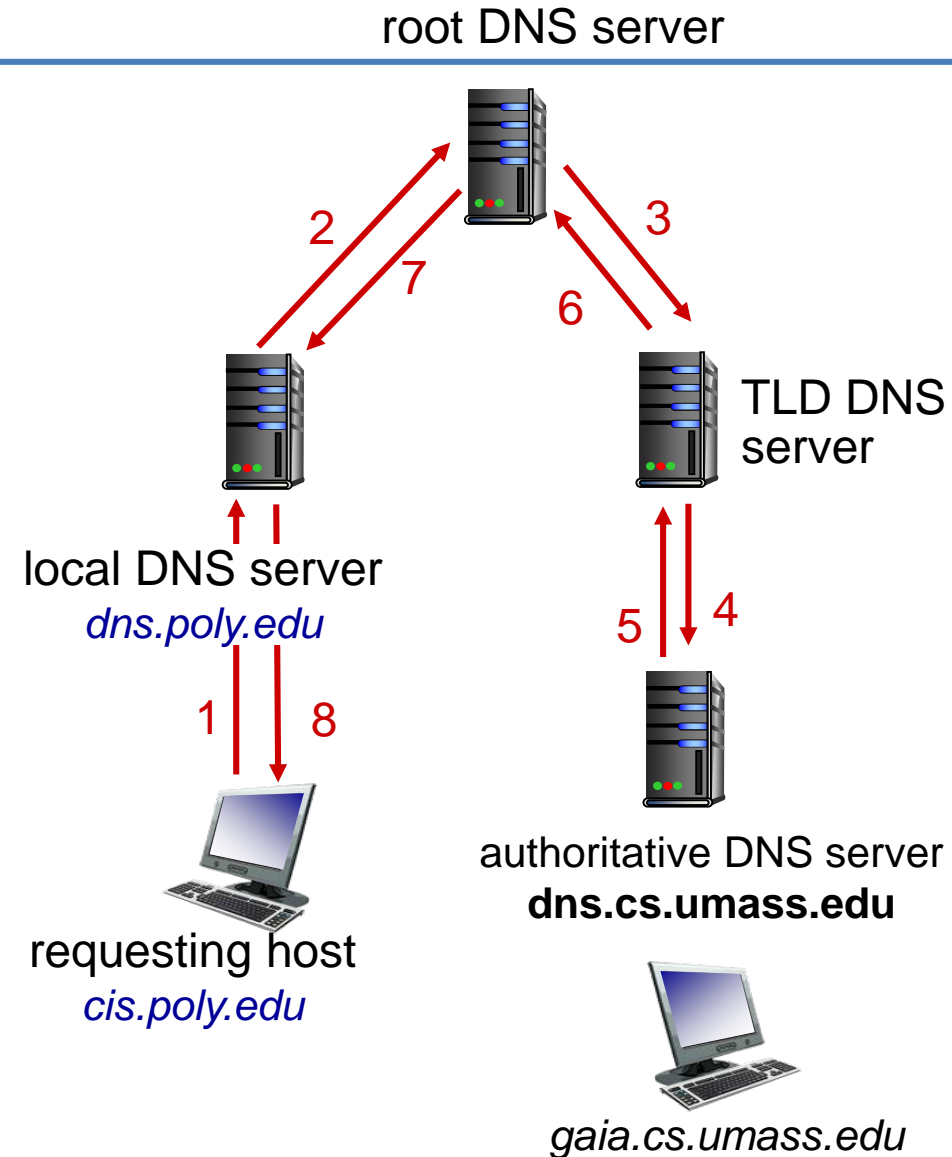
- contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”



DNS name resolution, another example

recursive query:

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



DNS records

DNS: distributed db storing resource records (RR)

DNS services

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

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

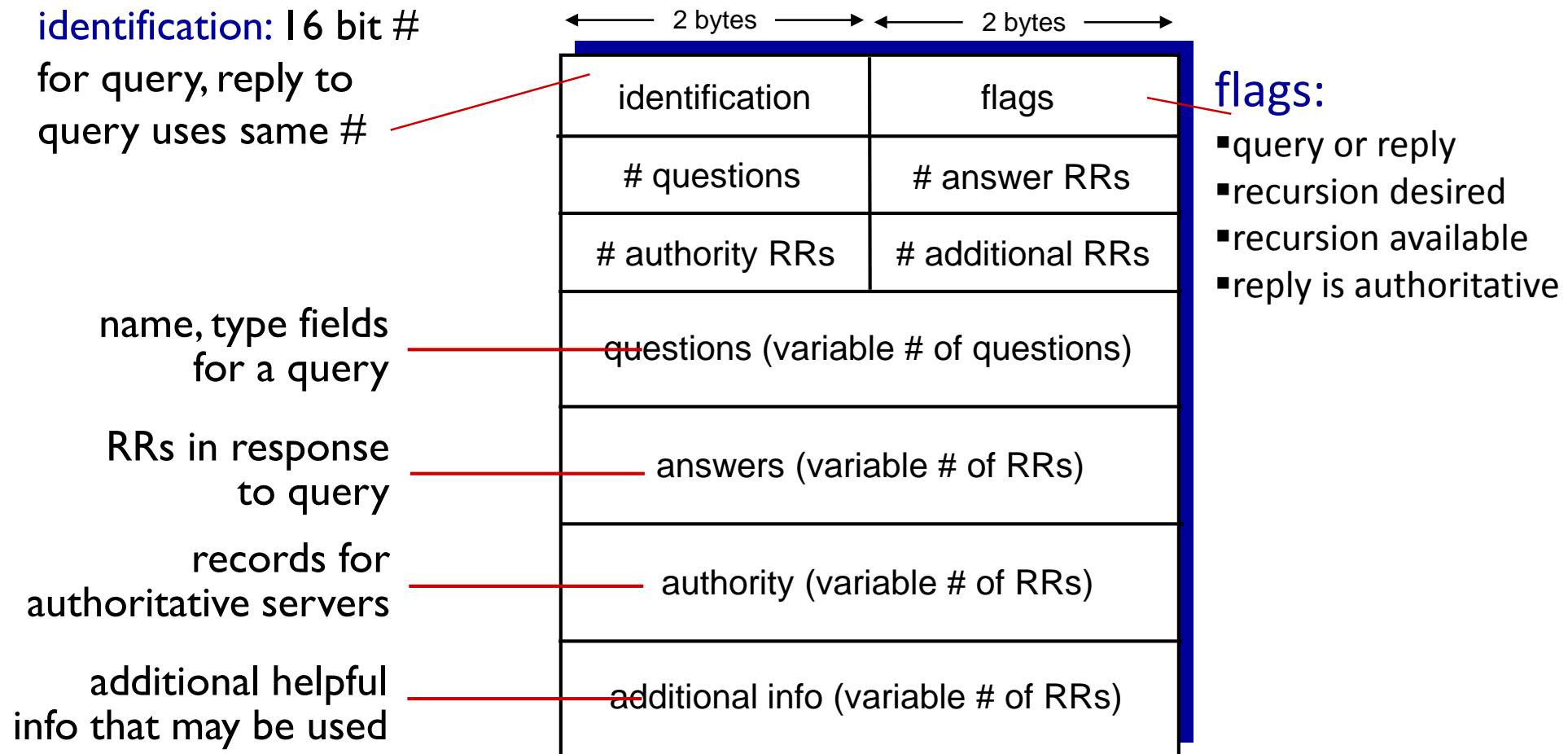
- **name** is alias name for some “canonical” (the real) name
- Eg `www.ibm.com` is really `servereast.backup2.ibm.com`
- **value** is canonical name

type=MX

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

DNS protocol, messages

- *query* and *reply* messages (use UDP), both with same *message format*

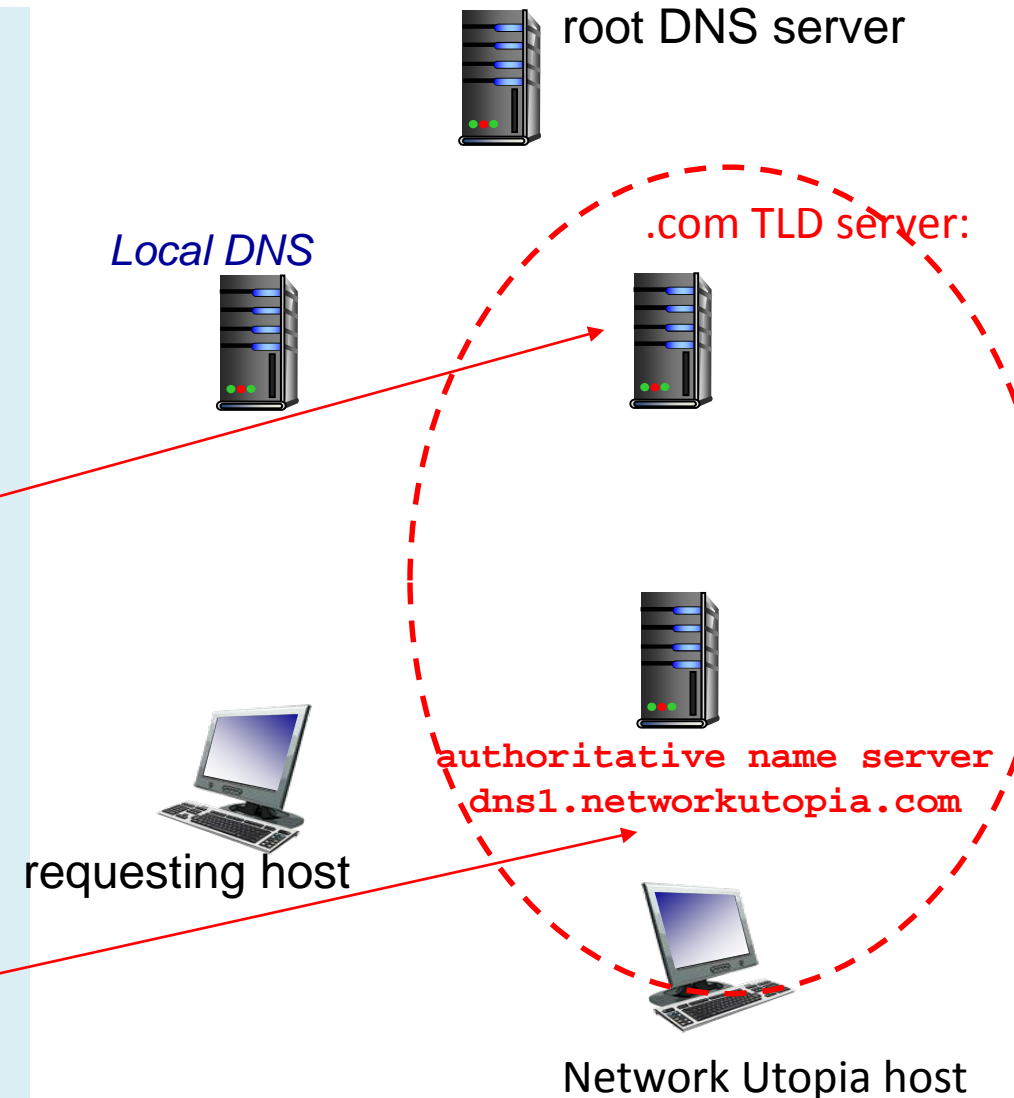


DNS: side note on caching / updating records

- once (any) name server learns mapping, it *caches* mapping
 - cache entries timeout after some time (TTL)
- cached entries may be *out-of-date* (best effort name-to-address translation!)
 - if name host changes IP address, may not be known Internet-wide until all TTLs expire
- update/notify mechanisms proposed IETF standard
 - RFC 2136

Inserting records into DNS

- example: new startup “Network Utopia”
- register name networkutopia.com at **DNS registrar** (e.g., Network Solutions)
 - provide names, IP addresses of **authoritative name server**
 - registrar inserts two RRs into .com TLD server:
(networkutopia.com,
dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1,
A)
- Adding a **new host/service to domain**:
 - Add to authoritative name server
 - type A record for www.networkutopia.com
 - type MX record for networkutopia.com (mail)



DNS and security risks

DDoS attacks

- Bombard root servers
 - Mitigation (it actually works 😊): local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombard TLD servers
 - Potentially more dangerous

Redirect attacks

- Man-in-middle
 - Intercept queries
- DNS poisoning
 - Send bogus replies to DNS server, which it caches

Exploit DNS for DDoS

- Send queries with spoofed source address

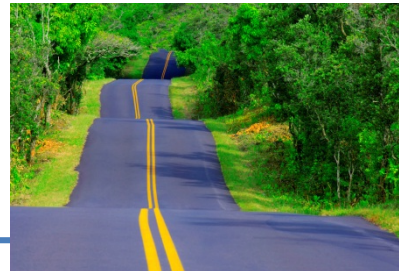
Summary

- Addressing and Applications needs from Transport layer
- Application architectures
 - client-server
 - *(p2p: will study later in the course, after the layers-centered study)*
- specific protocols:
 - Http (connection to programming assignment)
 - Caching etc
 - SMTP (POP, IMAP)
 - DNS

Link to representative questions for week 1
Please reply by Saturday 20/1, 7:00
<https://goo.gl/forms/7FpM6SJWIRILERee2>

Coming soon, after a pass of the 4 top layers

- P2P applications
- video streaming, content distribution networks and more



Resources

Reading list main textbook:

- Study: 6/e: 2.2, 2.4-2.5, 2.7-2.8, 7/e: 2.2-2.4
- Quick reading: 6/e: 2.1, 2.3, 2.6, 7/e: 2.1, 2.5

Review questions from the book, useful for summary study

- Chapter 2: 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 16, 19, 20

Extra slides/notes

Example review question

Properties of transport service of interest to the app.

Data loss

- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, telnet) require 100% reliable data transfer
- In-order vs arbitrary-order delivery

Bandwidth, Timing, Security

- ❑ some apps (e.g., multimedia, interactive games) require minimum amount of **bandwidth**, and/or low **delay** and/or low **jitter**
- ❑ other apps (elastic apps, e.g. file transfer) are ok with any bandwidth, timing they get

Some apps also require **confidentiality**, **integrity** (more in network security)

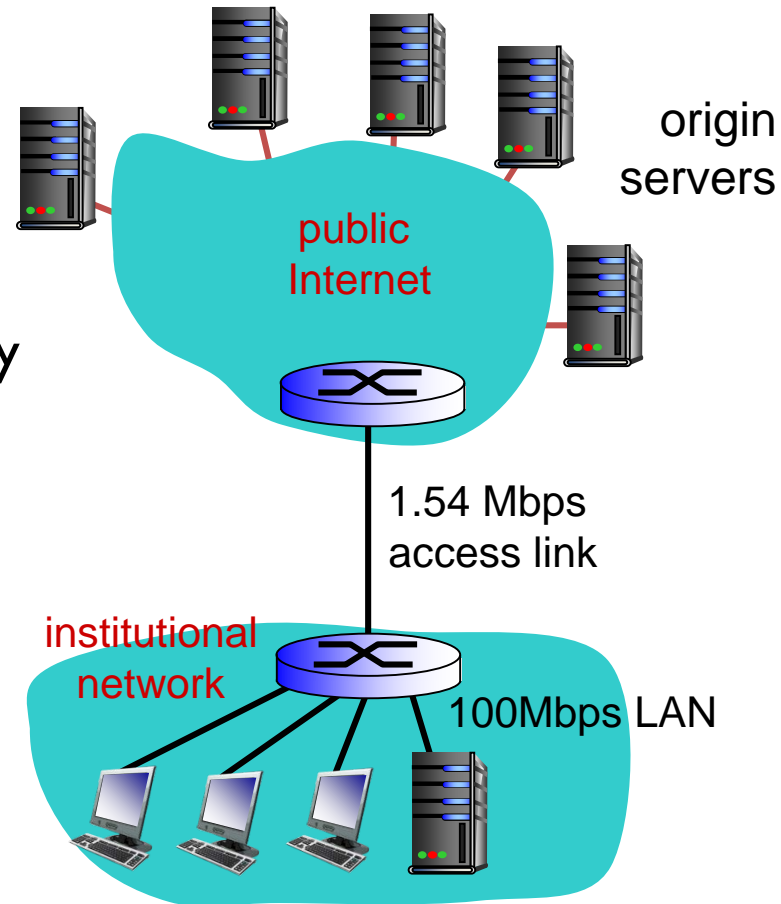
Caching example, as an exercise:

assumptions:

- ❖ avg object size: 100K bits
- ❖ avg request rate from browsers to origin servers: 15/sec
 - ❖ i.e. avg data rate to browsers: 1.50 Mbps
- ❖ RTT from institutional router to any origin server: 2 sec
- ❖ access link rate: 1.54 Mbps

consequences:

- ❖ LAN utilization: 1.5%
- ❖ access link utilization = 99% *problem!*
- ❖ total delay = Internet delay + access delay + LAN delay
= 2 sec + minutes + quite_small



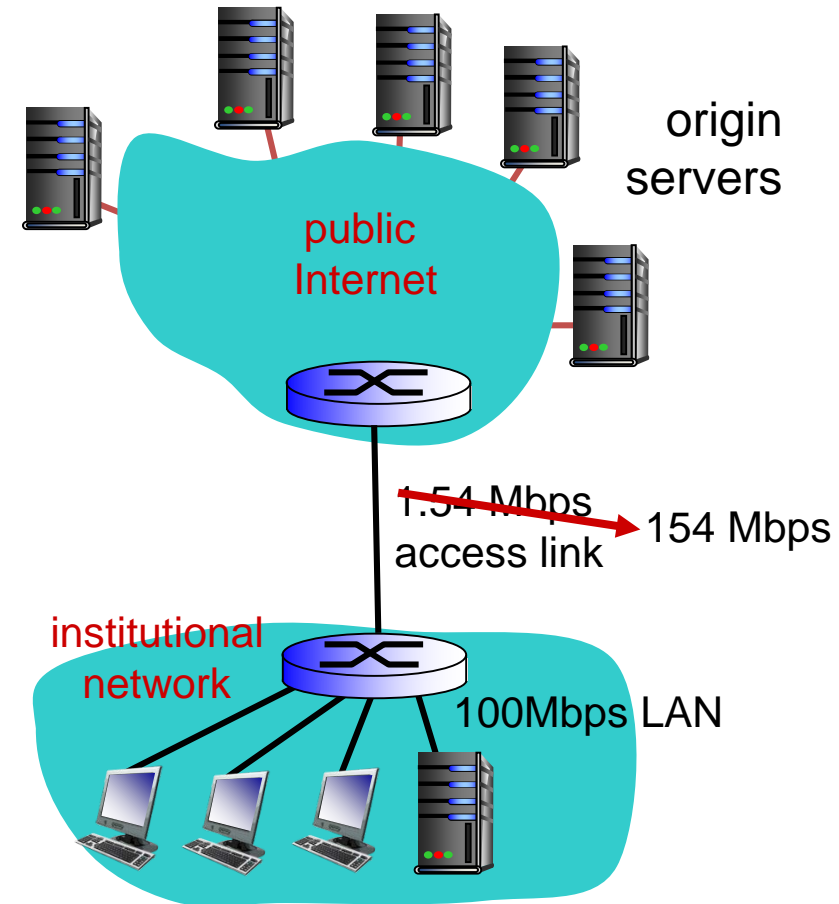
Caching example: faster access link

assumptions:

- ❖ avg object size: 100K bits
- ❖ avg request rate from browsers to origin servers: 15/sec
 - ❖ i.e. avg data rate to browsers: 1.50 Mbps
- ❖ RTT from institutional router to any origin server: 2 sec
- ❖ access link rate: ~~1.54 Mbps~~

consequences:

- ❖ LAN utilization: 1.5%
- ❖ access link utilization = ~~99%~~ → 9.9%
- ❖ total delay = Internet delay +
access delay + LAN delay
= 2 sec + minutes + usecs
→ msecs



Cost: increased access link speed (not cheap!)

Caching example: install local cache

assumptions:

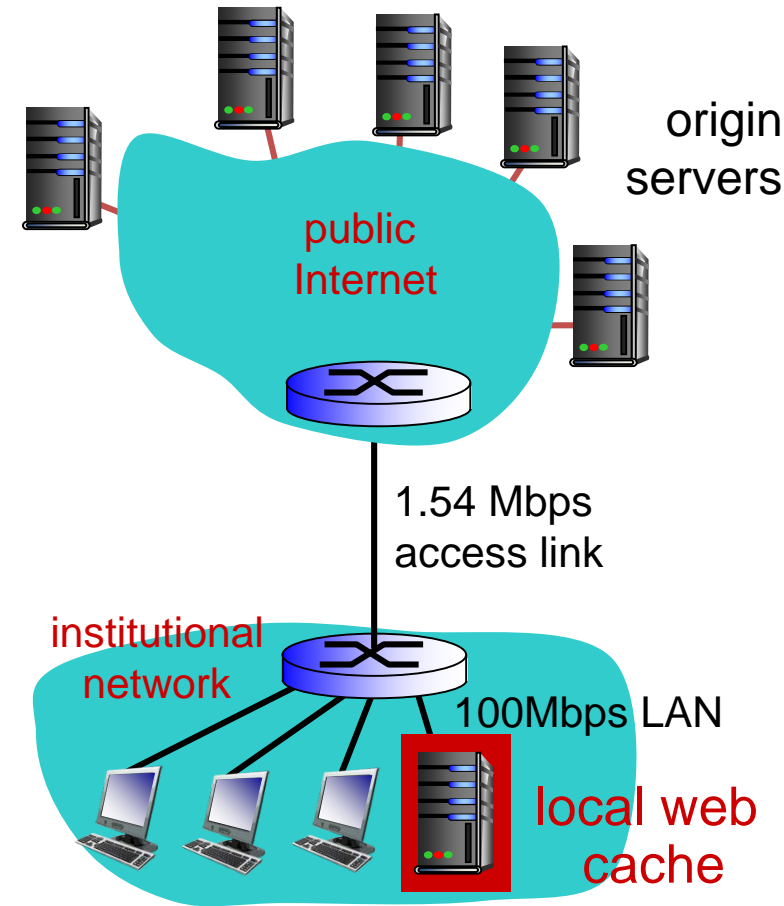
- ❖ avg object size: 100K bits
- ❖ avg request rate from browsers to origin servers: 15/sec
 - ❖ i.e. avg data rate to browsers: 1.50 Mbps
- ❖ RTT from institutional router to any origin server: 2 sec
- ❖ access link rate: 1.54 Mbps

consequences:

- ❖ LAN utilization: 1.5%
- ❖ access link utilization ?
- ❖ total delay ?

How to compute link utilization, delay?

Cost: web cache (cheap!)



Caching example: install local cache

Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4
 - 40% requests satisfied at cache, 60% requests satisfied at origin
- ❖ access link utilization:
 - 60% of requests use access link
- ❖ data rate to browsers over access link = $0.6 * 1.50 \text{ Mbps} = .9 \text{ Mbps}$
 - utilization = $0.9 / 1.54 = .58$
- ❖ total delay
 - = $0.6 * (\text{delay from origin servers}) + 0.4 * (\text{delay when satisfied at cache})$
 - = $0.6 (2.01) + 0.4 (\sim \text{msecs})$
 - = $\sim 1.2 \text{ secs}$
 - less than with 154 Mbps link (and cheaper too!)

