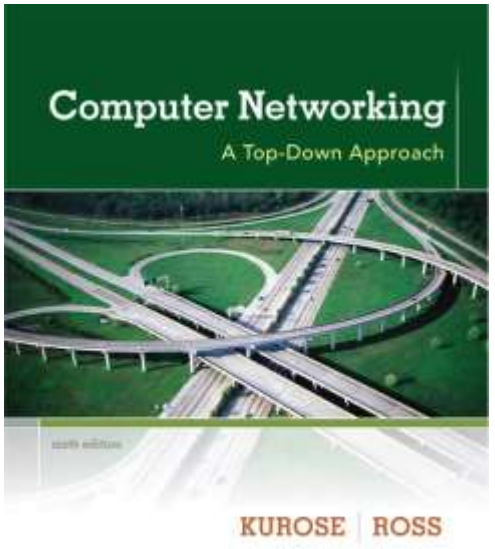
Application Layer

Computer Communications and Networks
ITCS 6166/8166

Dr. Dewan Tanvir Ahmed
Department of Computer Science
University of North Carolina at Charlotte

Chapter 2
Application Layer



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

- *If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)*
- *If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.*

Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

Chapter 2: outline

2.1 Principles of network applications

2.2 Web and HTTP

2.3 FTP

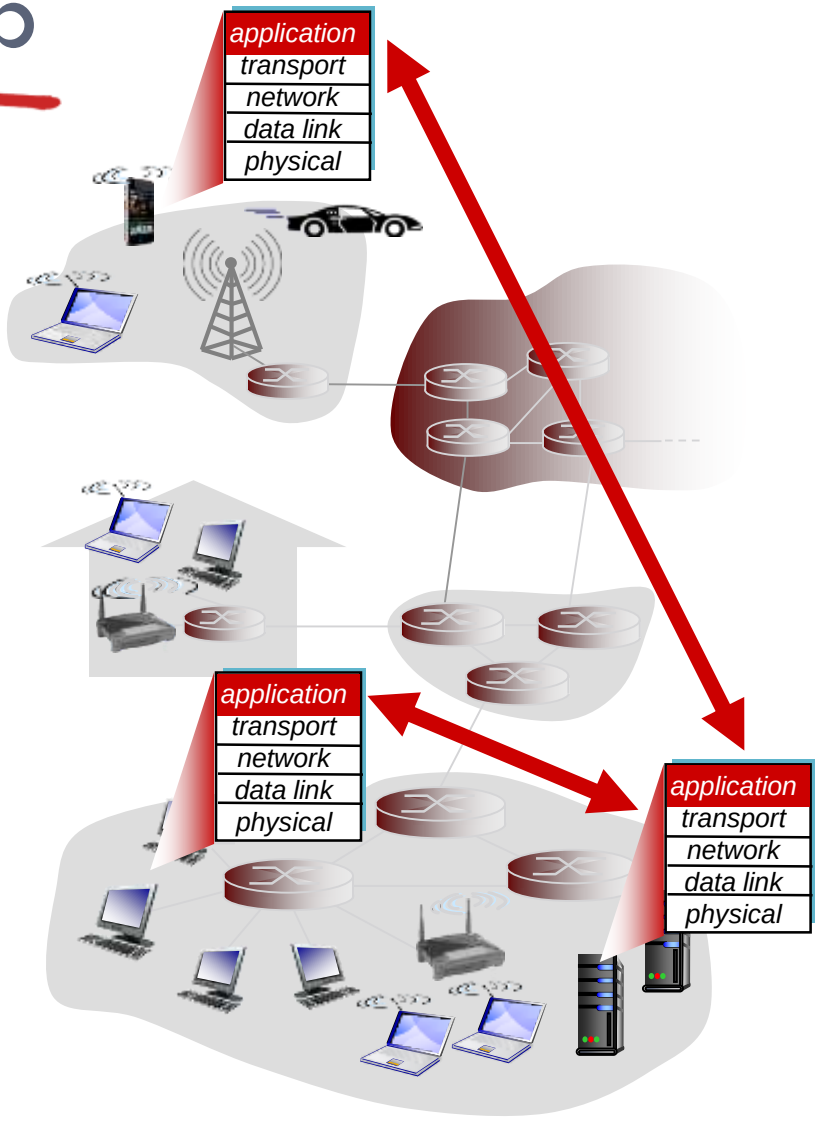
2.4 Electronic mail

- SMTP, POP3, IMAP

2.5 DNS

Creating a network app

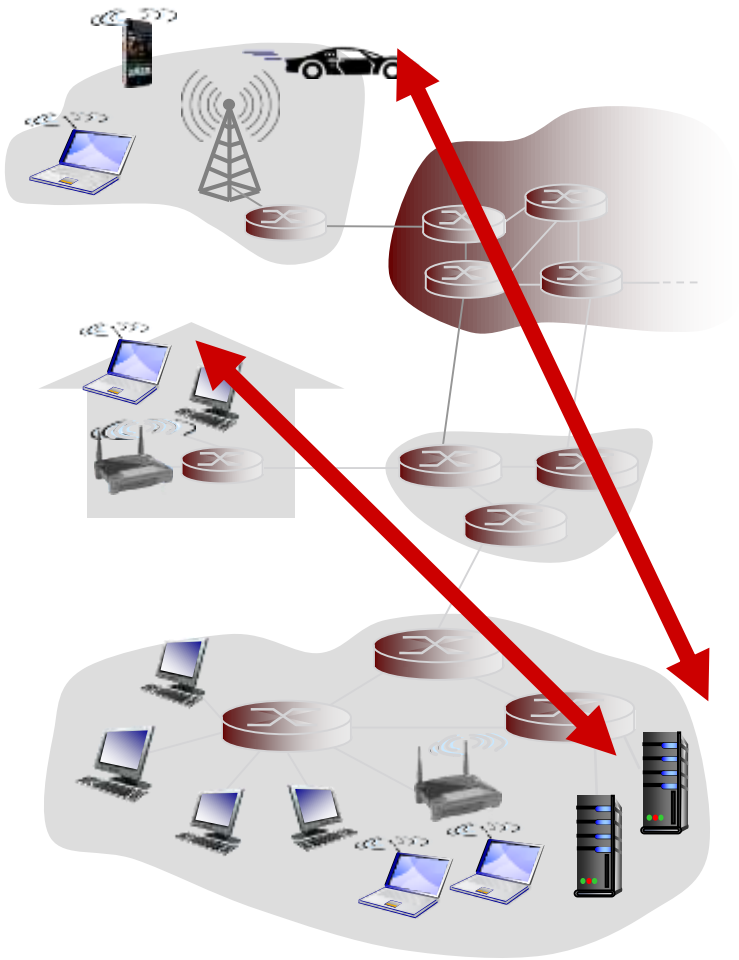
- write programs that:
  - run on (different) end systems
  - communicate over network
    - e.g., web server software communicates with browser software
- no need to write software for network-core devices
  - network-core devices do not run user applications



Application architectures

- Possible structure of applications:
  - client-server
  - peer-to-peer (P2P)

Client-server architecture



client/server

- Server:
  - always-on host
  - permanent IP address
  - data centers for scaling
- Clients:
  - communicate with server
  - may be intermittently connected
  - may have dynamic IP addresses
  - do not communicate directly with each other

Processes communicating

process: program running within a host

- within same host, two processes communicate using inter-process communication (defined by OS)
- processes in different hosts communicate by exchanging messages

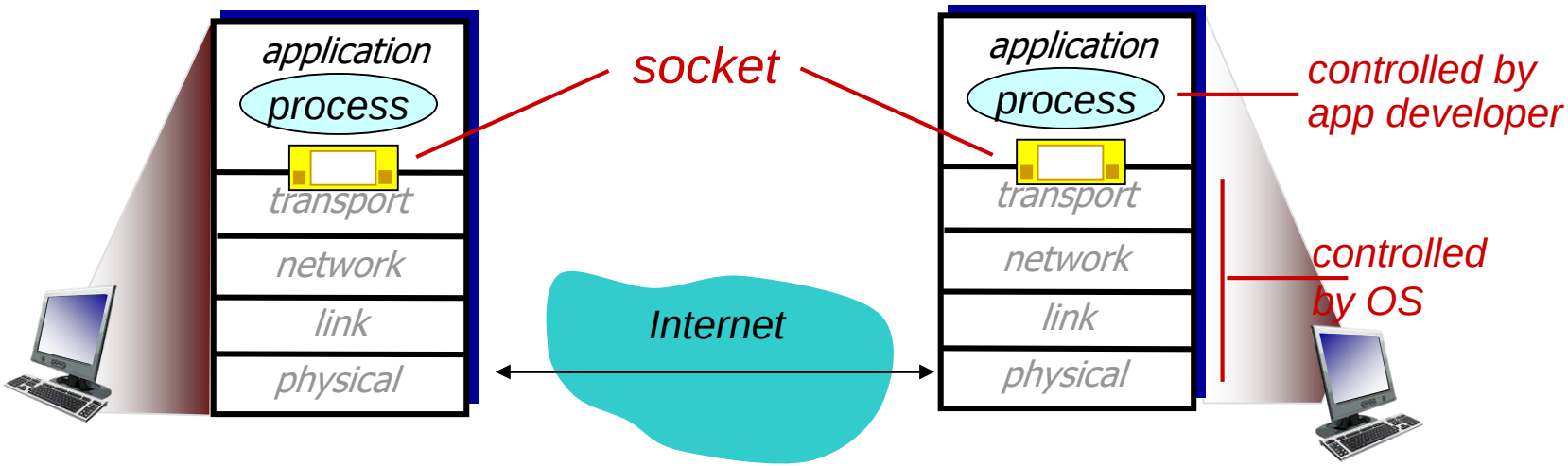
clients, servers

client process: process that initiates communication

server process: process that waits to be contacted

Sockets

- process sends/receives messages to/from its socket



Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- Q: does IP address of host on which process runs suffice for identifying the process?
  - A: *no, many processes can be running on same host*

- *identifier* includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80

What transport service does an app need?

data integrity

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- other apps ("elastic apps") make use of whatever throughput they get

security

- encryption, data integrity, …

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	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
text messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP service:

- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantee, security
- *connection-oriented:* setup required between client and server processes

UDP service:

- *unreliable data transfer* between sending and receiving process
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup,

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

App-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

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 Electronic mail

- SMTP, POP3, IMAP

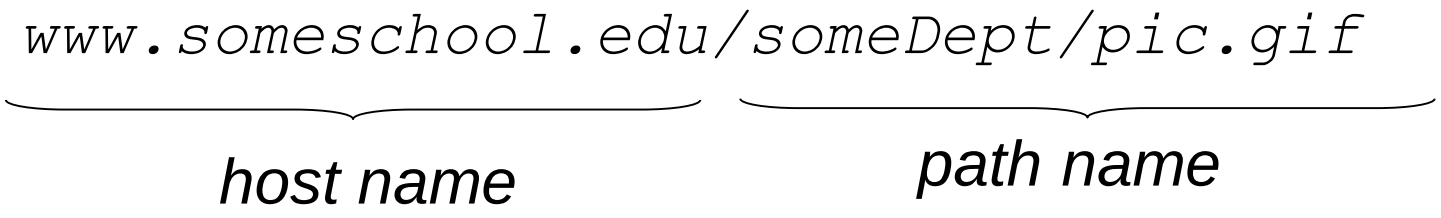
2.5 DNS

Web and HTTP

First, a review…

- web page consists of base HTML-file which includes several referenced objects
  - object can be JPEG image, Java applet, audio file,…

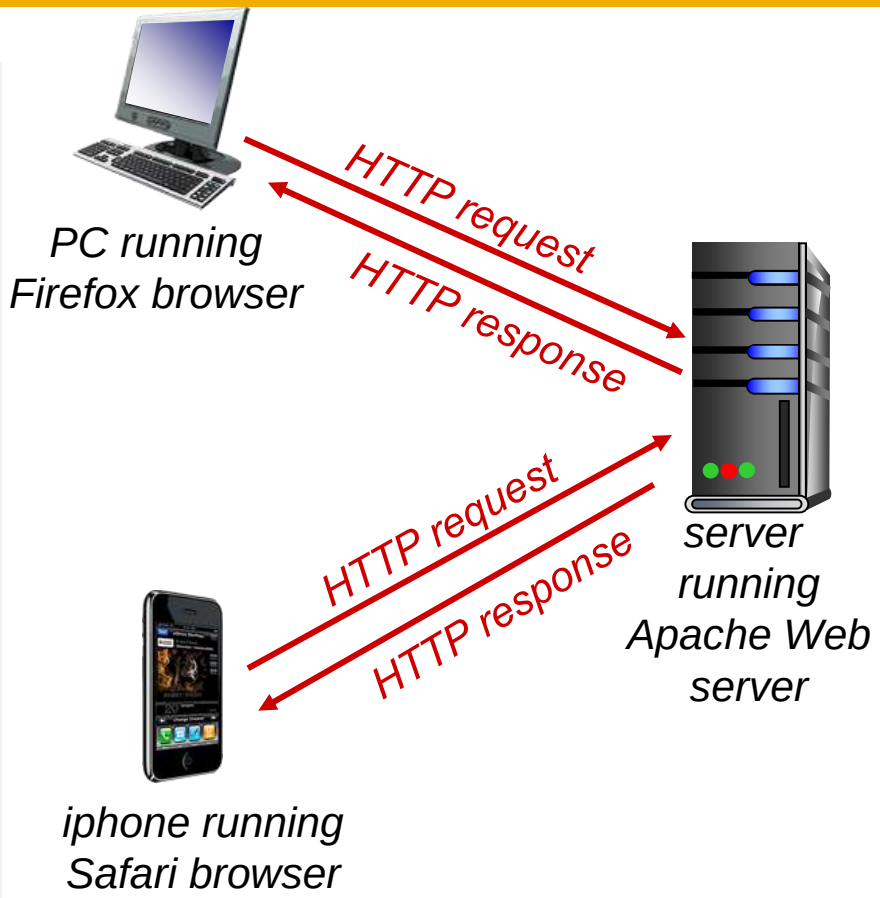
- each object is addressable by a URL, e.g.,



HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) 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 exchanged between browser and Web server
- TCP connection closed

HTTP is "stateless"

- server maintains no information about past client requests

aside

protocols that maintain "state" are complex!

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

HTTP connections

non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

persistent HTTP

- multiple objects can be sent over single TCP connection between client and server
- HTTP uses persistent connections in its default mode

Non-persistent HTTP

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 on port 80

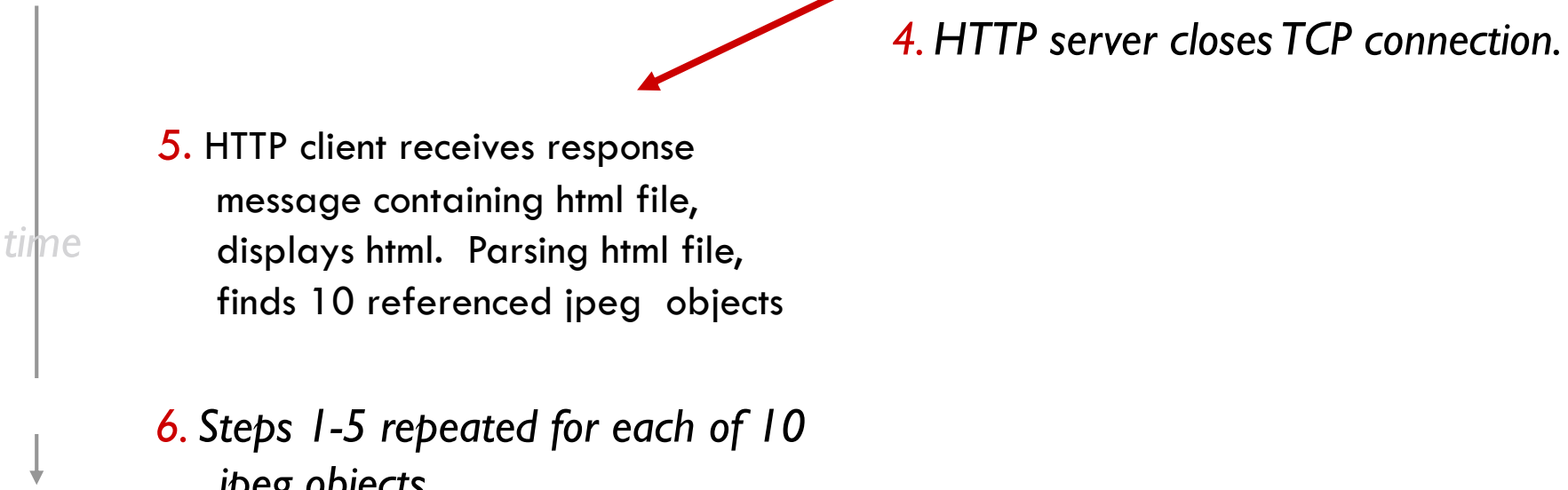
1b. *HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client*

2. *HTTP client sends HTTP request message (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index*

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

time

Non-persistent 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*

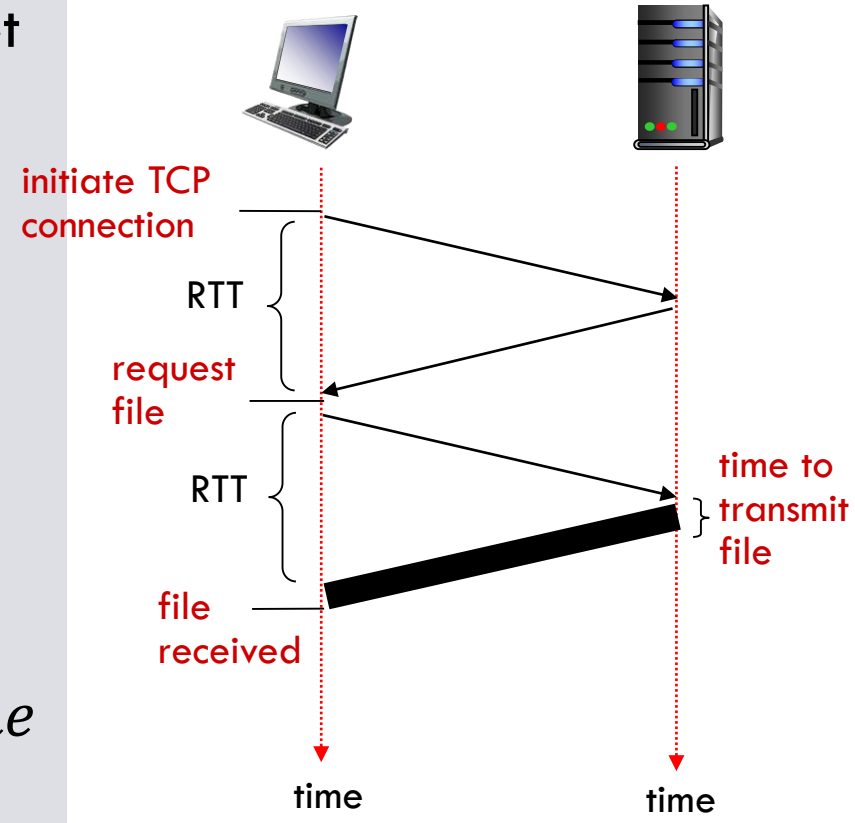
Non-persistent HTTP: response time

RTT (definition): time for a small packet to travel from client to server and back

HTTP response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response
- non-persistent HTTP response time

$= 2RTT + file\ transmission\ time$



Persistent HTTP

non-persistent HTTP issues:

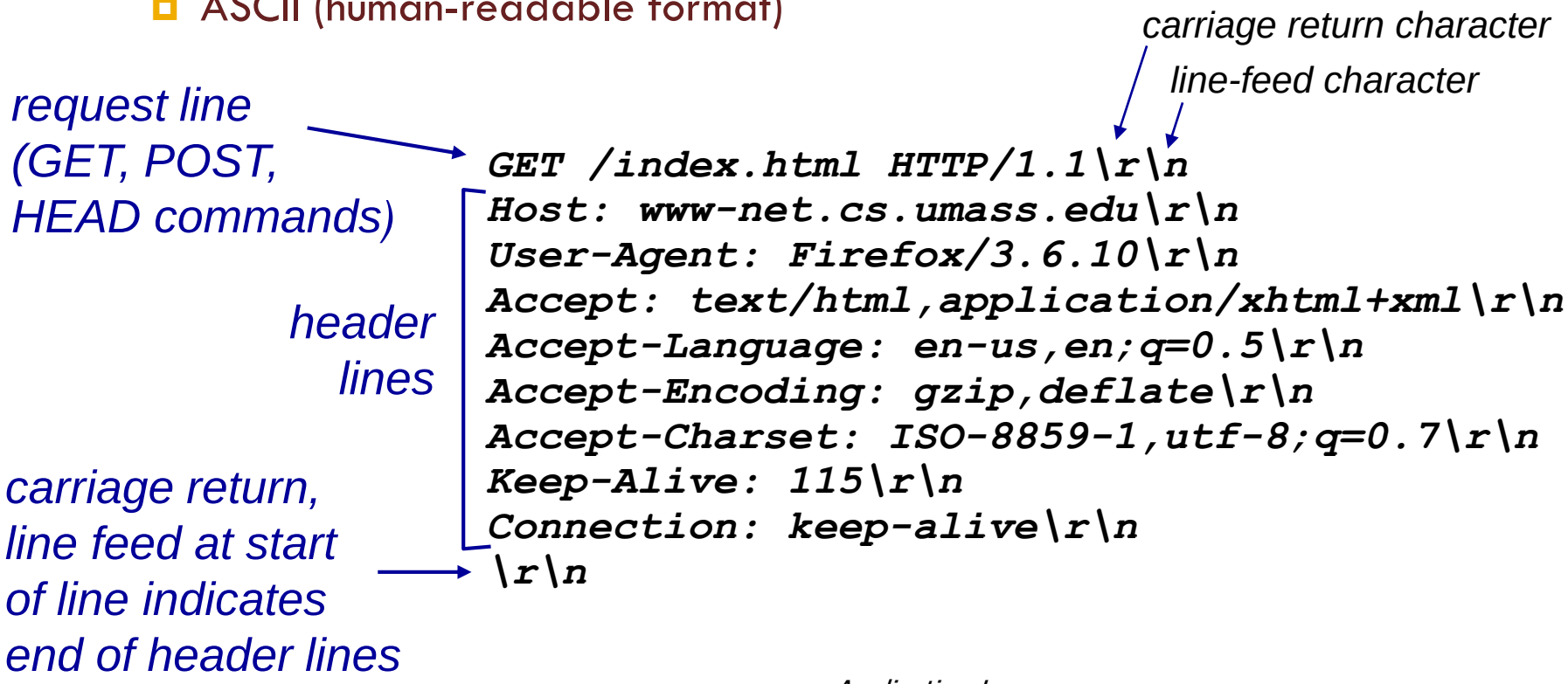
- requires 2 RTTs per object
- OS overhead
  - For each connection, TCP buffers must be allocated and TCP variables must be kept in both the client and server.
- browsers often open parallel TCP connections to fetch referenced objects

persistent HTTP:

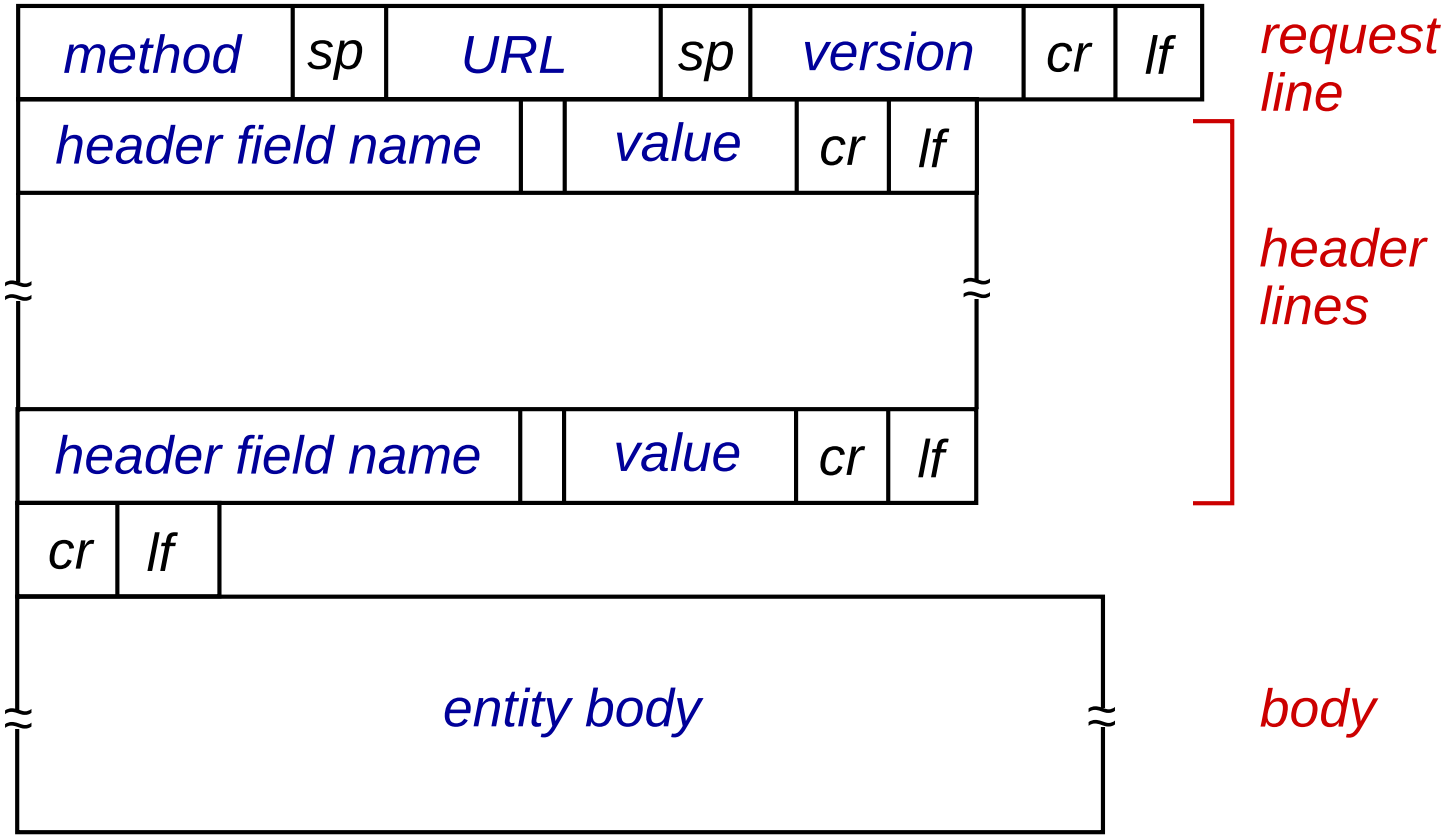
- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection (pipelining)
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects
- Typically, the HTTP server closes a connection when it isn't used for a certain.

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
- input is uploaded to server in entity body
- The entity body is empty with the GET method, but is used with the POST method.

URL method:

- uses GET method
- input is uploaded in URL field of request line:

www.somesite.com/animalsearch?monkeys&banana

Method types

HTTP/1.0:

- GET
- POST
- HEAD
  - Similar to GET method
  - asks server to leave requested object out of response
  - May be used for debugging

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - used by applications that need to upload objects to Web servers.
- DELETE
  - deletes file specified in the URL field

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 2010 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
data data data data data ...
```

HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

200 OK

- request succeeded, requested object later in this msg

301 Moved Permanently

- requested object moved, new location specified later in this msg (Location:)

400 Bad Request

- requested msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

User-server state: cookies

many Web sites use cookies

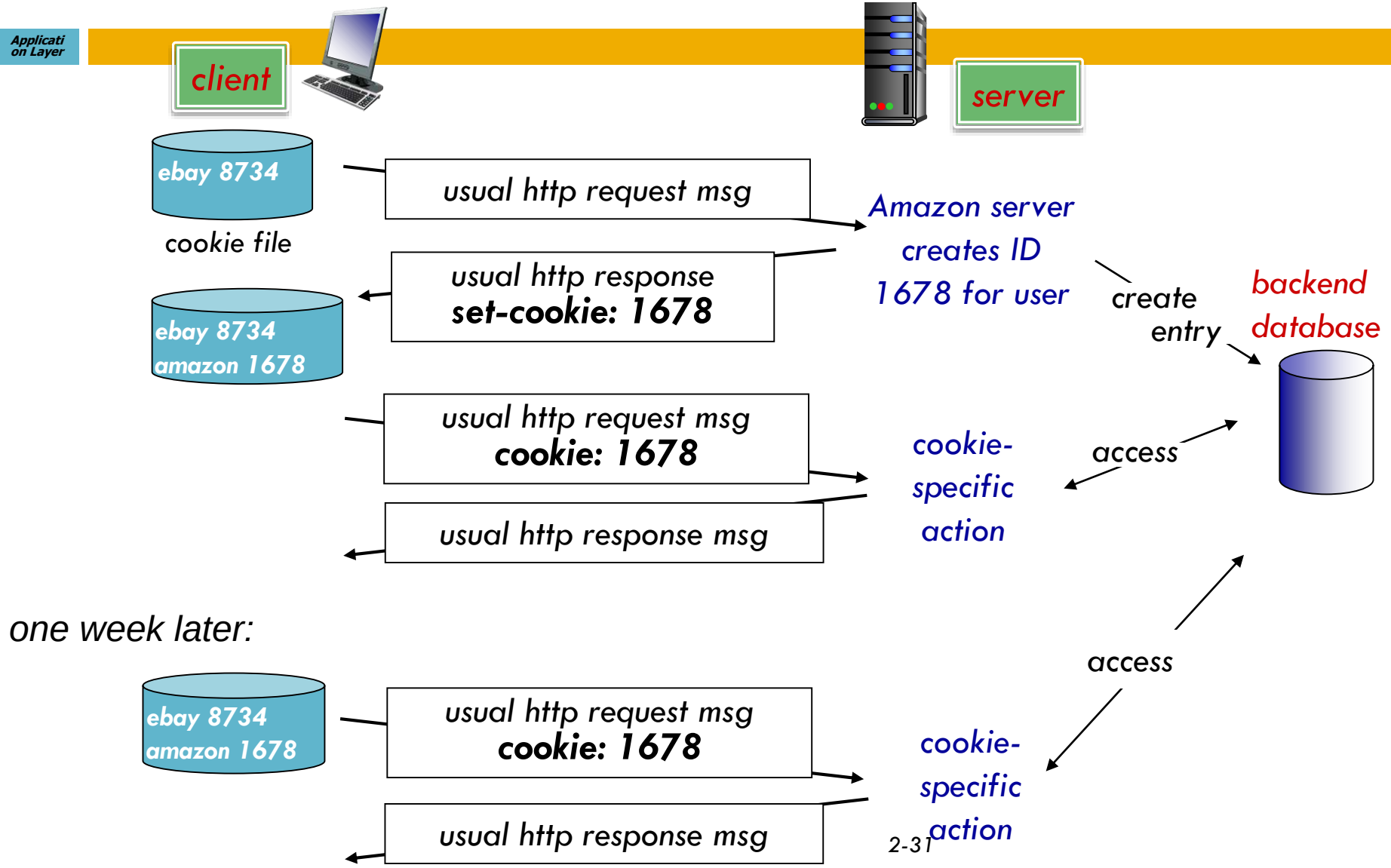
four components:

1) cookie header line of HTTP *response* message
2) cookie header line in next HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) back-end database at Web site

example:

- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies: keeping "state" (cont.)



Cookies (continued)

what cookies can be used for:

- authorization
- shopping carts
- recommendations

aside

cookies and privacy:

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

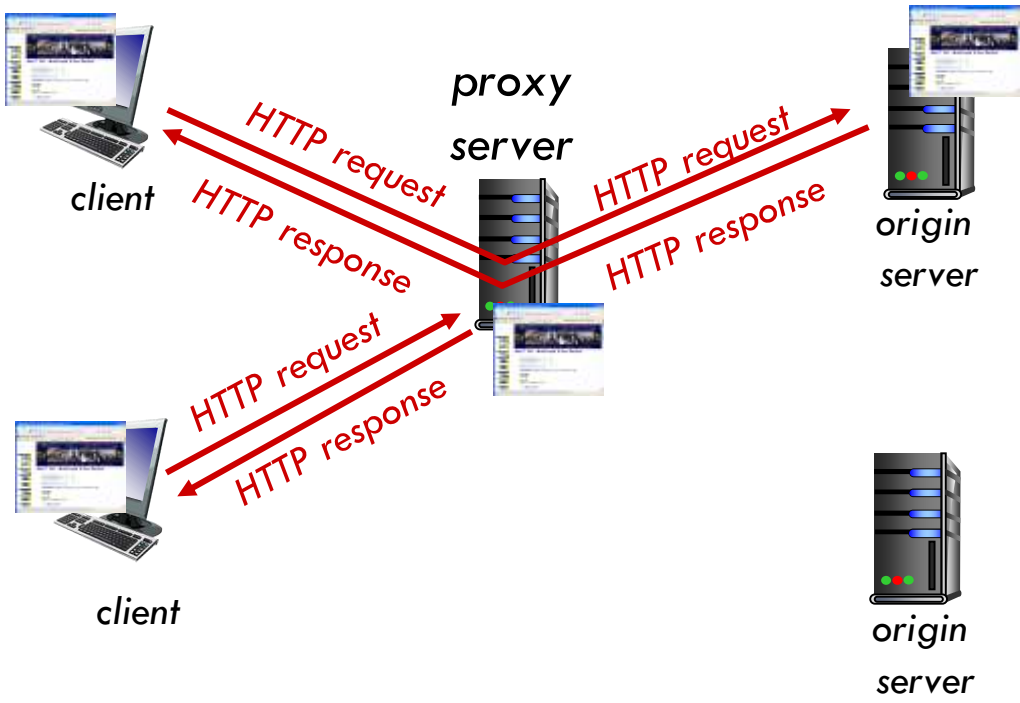
how to keep "state":

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

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

why Web caching?

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

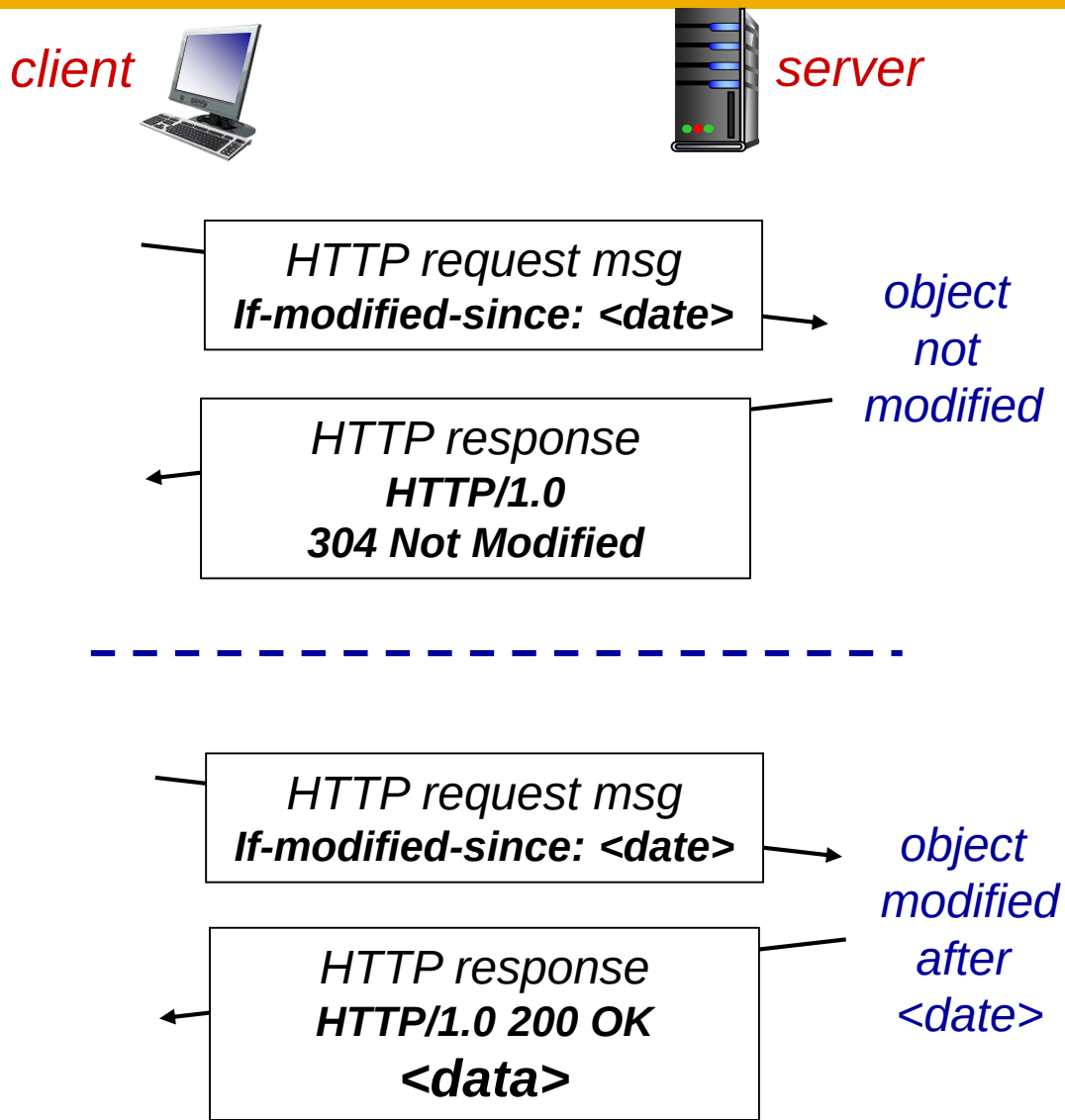
- typically cache is installed by ISP (university, company, residential ISP)

Conditional GET

- *Goal:* don't send object if cache has up-to-date cached version
  - no object transmission delay
  - lower link utilization
- *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`



Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

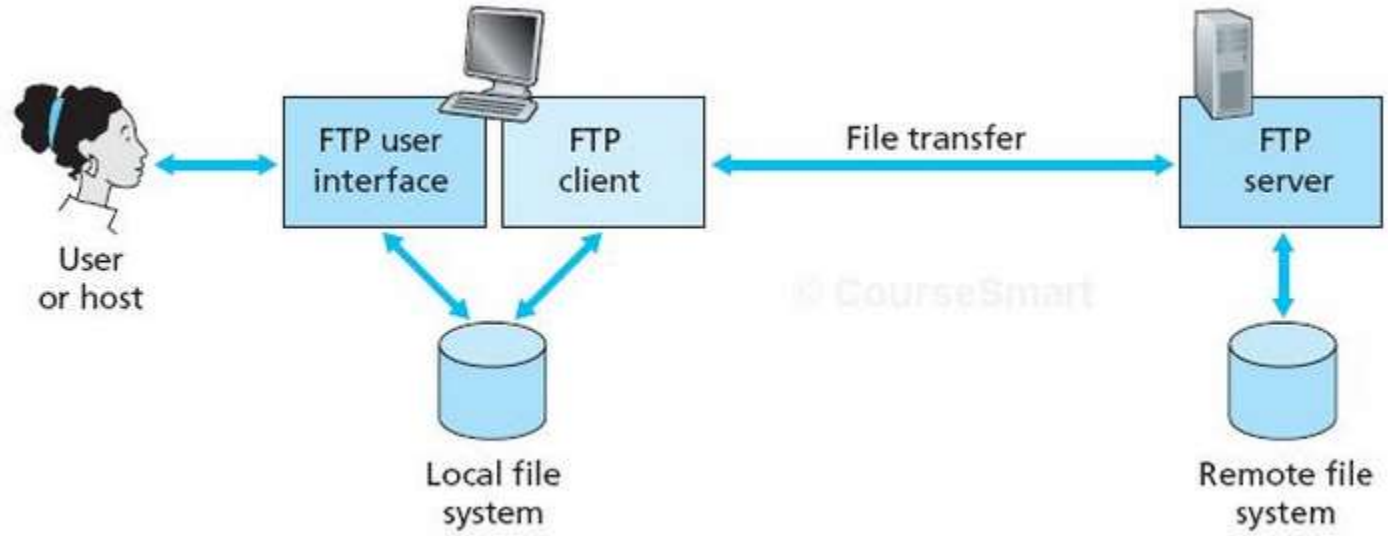
2.3 FTP

2.4 Electronic mail

- SMTP, POP3, IMAP

2.5 DNS

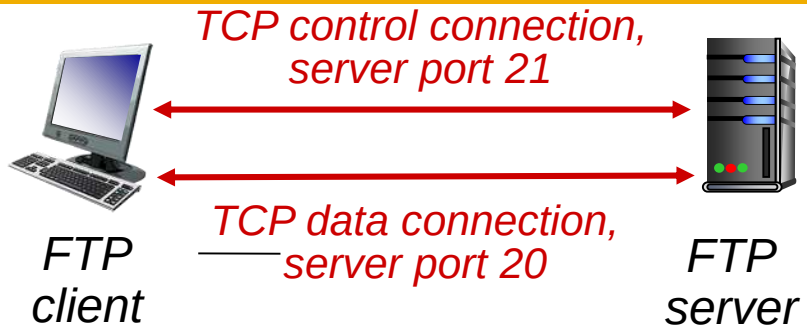
FTP: the file transfer protocol



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

FTP: separate control, data connections

- FTP client contacts FTP server at port 21, using TCP
- Over control connection
  - client authorized
  - client browses remote directory, sends commands
- when server receives file transfer command, *server* opens 2nd TCP data connection (for file) *to* client
- after transferring one file, server closes data connection



- FTP is said to send its control information out-of- band
- FTP server maintains "state": current directory, earlier authentication
- data connections are non-persistent

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**

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 Electronic mail

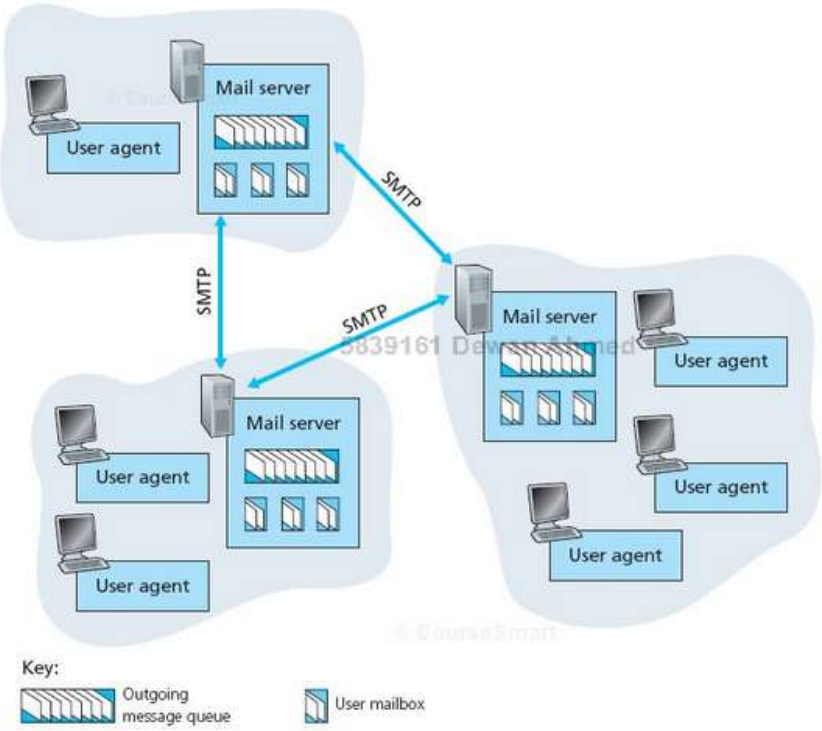
- SMTP, POP3, IMAP

2.5 DNS

Electronic mail

Three major components:

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



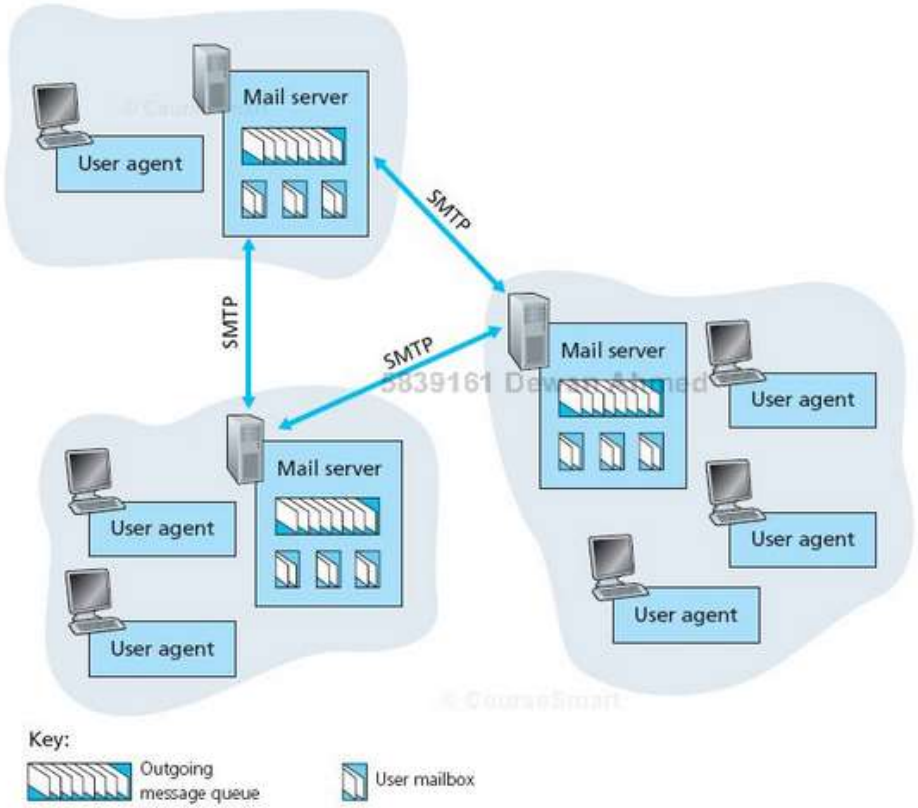
User Agent

- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- Outgoing and incoming messages stored on server

Electronic mail: mail servers

mail servers:

- *mailbox* contains incoming message for user
- *message queue* of outgoing (to be sent) mail messages
- *SMTP protocol* between mail servers to send email messages

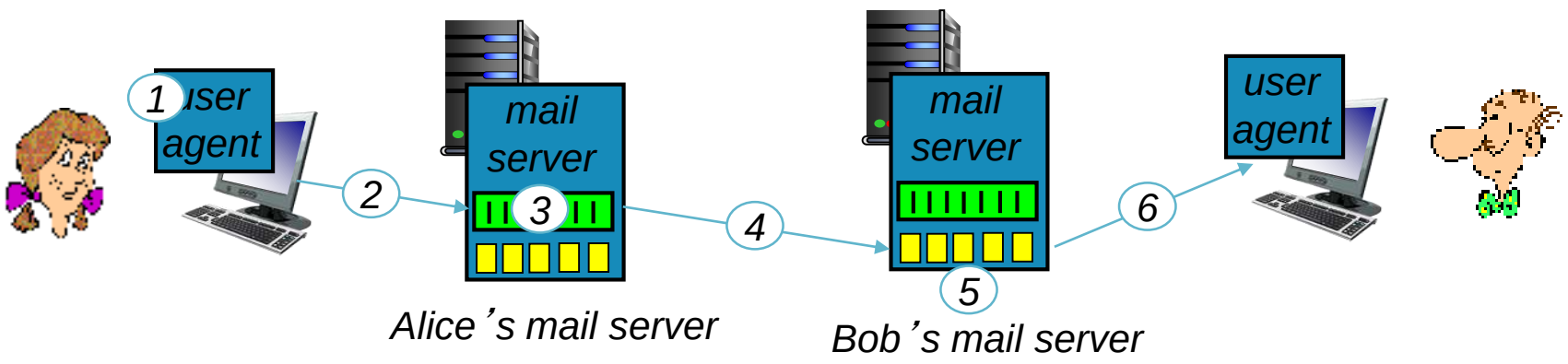


Electronic Mail: SMTP [RFC 2821]

- uses TCP and port 25
- Direct transfer: sending server to receiving server
- Three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction (like HTTP, FTP)
  - 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 "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



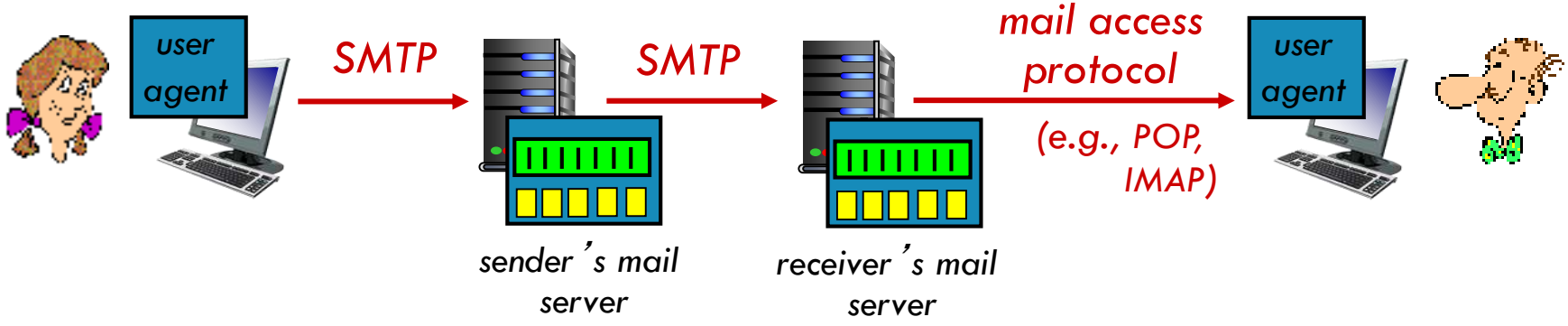
SMTP: final words

- 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

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
  - HTTP: gmail, 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 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
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP3 (more) and IMAP

more about POP3

- previous example uses POP3 "download and delete" mode
  - cannot re-read e-mail if you change client
- POP3 "download-and-keep": copies of messages on different clients

IMAP

- keeps all messages in one place: at server
- allows user to organize messages in folders
- keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name
- Good for low-bandwidth connection
  - permit a user agent to obtain components of messages.

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 Electronic mail

- SMTP, POP3, IMAP

2.5 DNS

DNS: domain name system

Internet hosts and 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 and name servers communicate to *resolve* names (address/name translation)

DNS: services, structure

All DNS query and reply messages are sent within UDP datagrams to port 53.

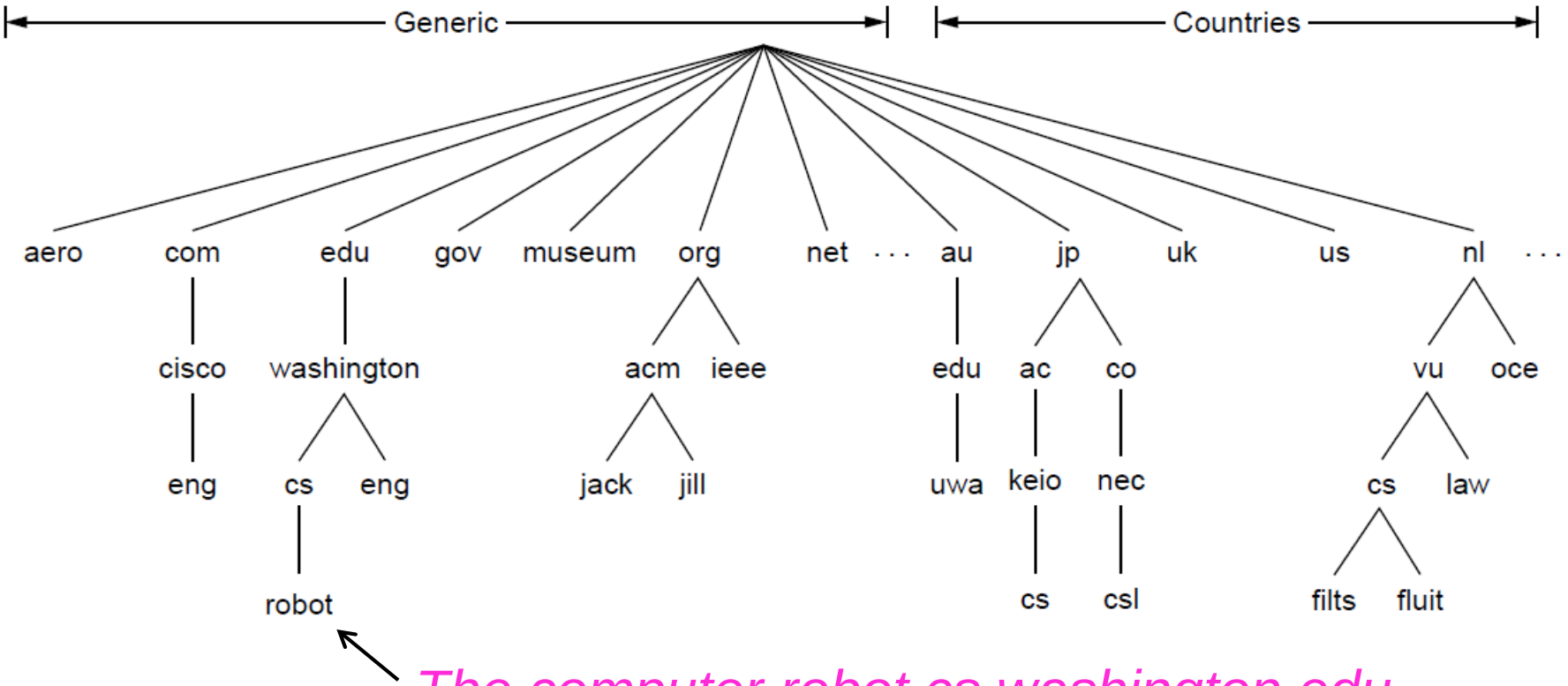
DNS services

- 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

The DNS Name Space (1)

DNS namespace is hierarchical from the root down

- Different parts delegated to different organizations

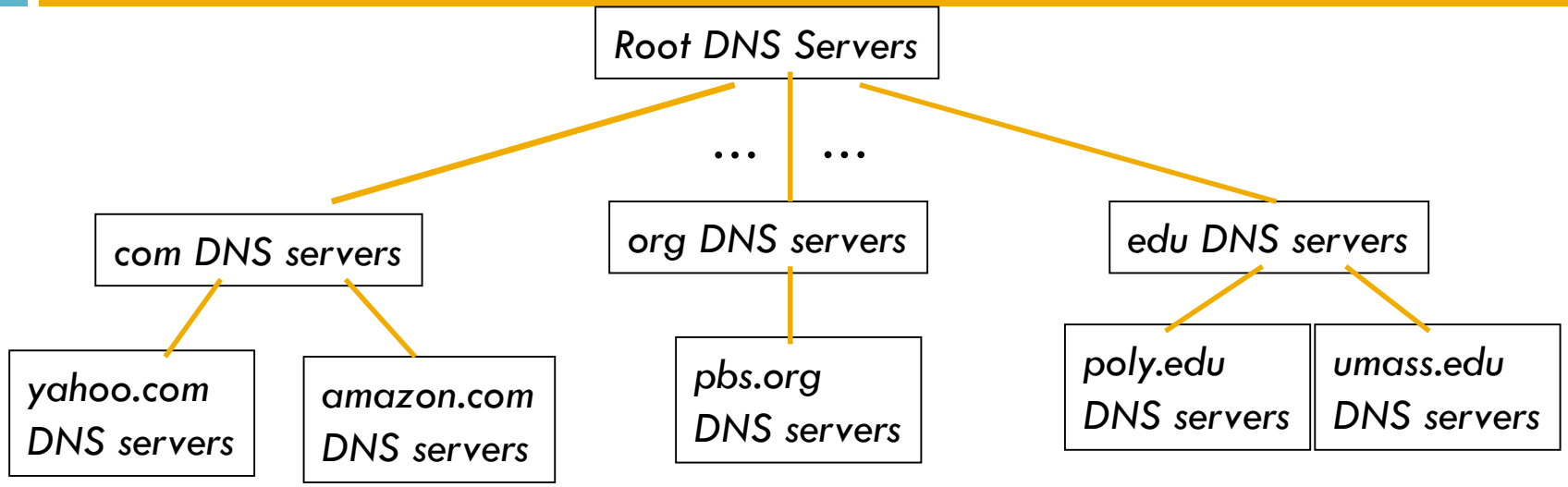


The DNS Name Space (2)

Generic top-level domains are controlled by ICANN (Internet Corporation for Assigned Names and Numbers) who appoints registrars to run them

Domain	Intended use	Start date	Restricted?
com	Commercial	1985	No
edu	Educational institutions	1985	Yes
gov	Government	1985	Yes
int	International organizations	1988	Yes
mil	Military	1985	Yes
net	Network providers	1985	No
org	Non-profit organizations	1985	No
aero	Air transport	2001	Yes
biz	Businesses	2001	No
coop	Cooperatives	2001	Yes
info	Informational	2002	No
museum	Museums	2002	Yes
name	People	2002	No
pro	Professionals	2002	Yes
cat	Catalan	2005	Yes
jobs	Employment	2005	Yes
mobi	Mobile devices	2005	Yes
tel	Contact details	2005	Yes
travel	Travel industry	2005	Yes
xxx	Sex industry	2010	No

DNS: a distributed, hierarchical database

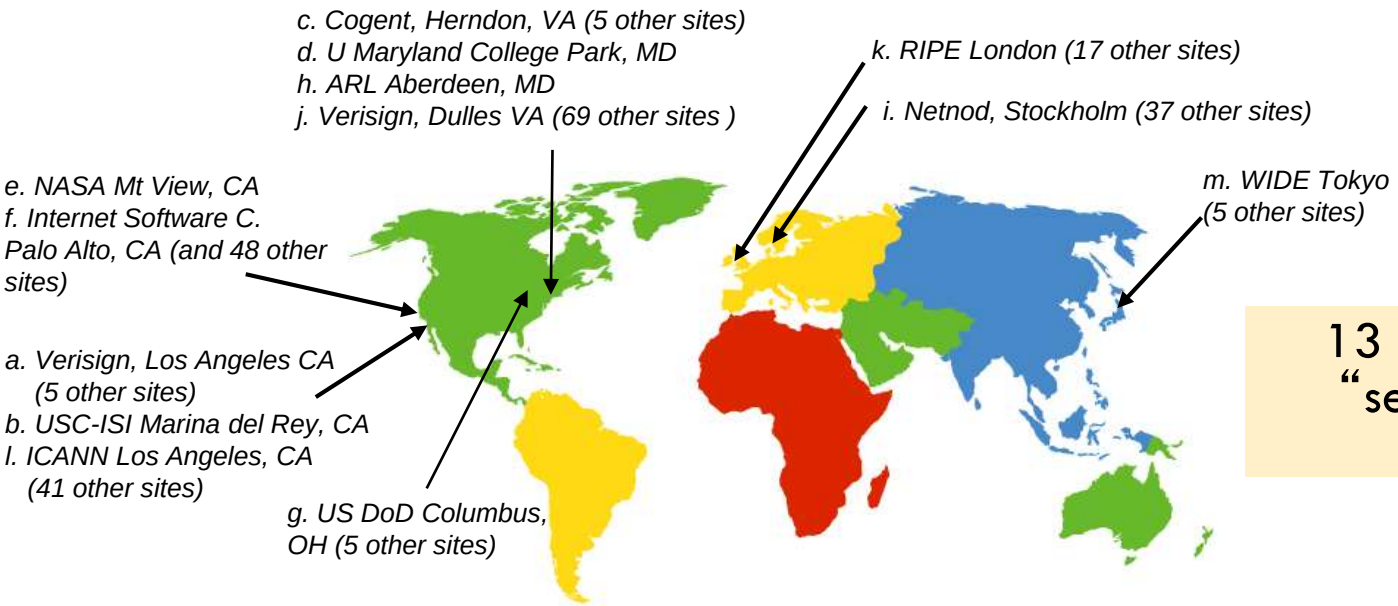


client wants IP for www.amazon.com; 1st approx:

- 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

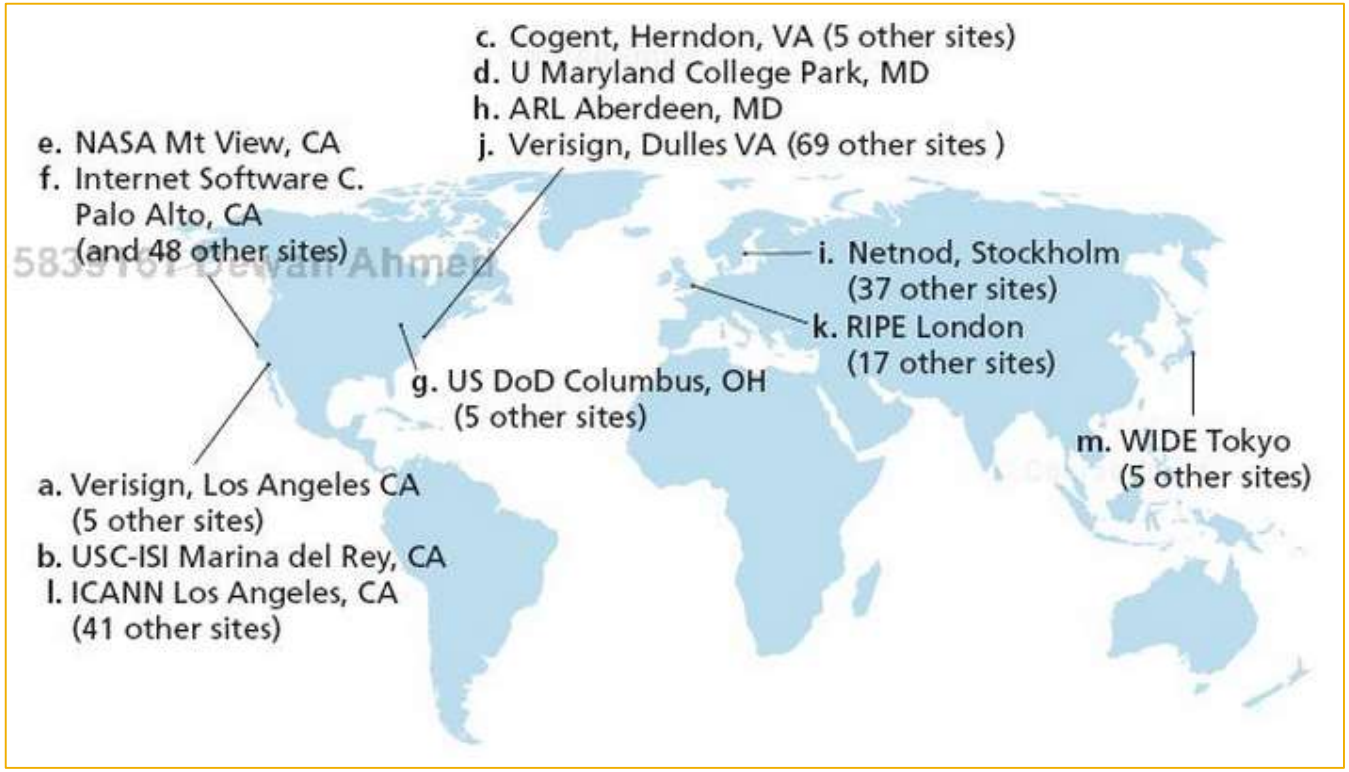
- contacted by local name server that can not resolve name



13 root name "servers" worldwide

DNS: root name servers

- Each " root server" is actually a network of replicated servers, for both security and reliability purposes.
- All together, there are 247 root servers as of fall 2011. **[504 as of 17 January 2016]**



TLD and 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

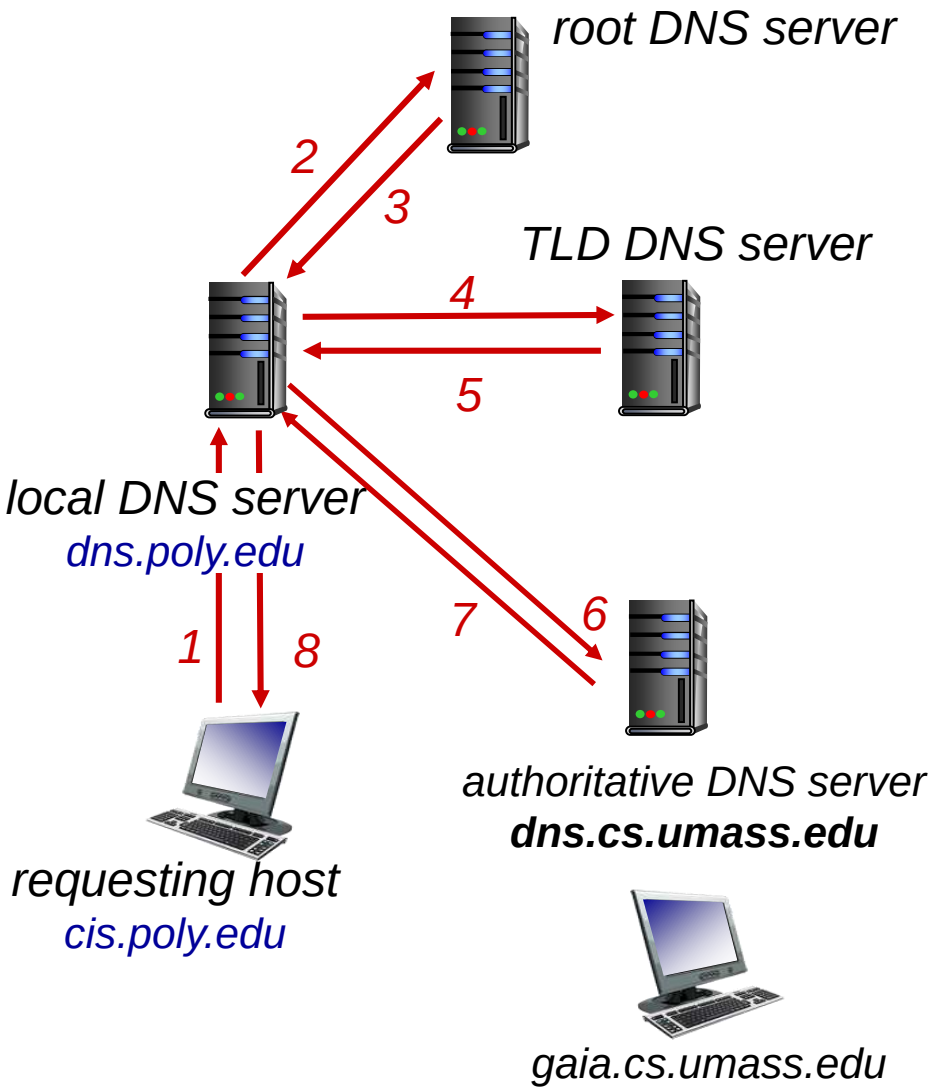
Local DNS name server

- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one *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

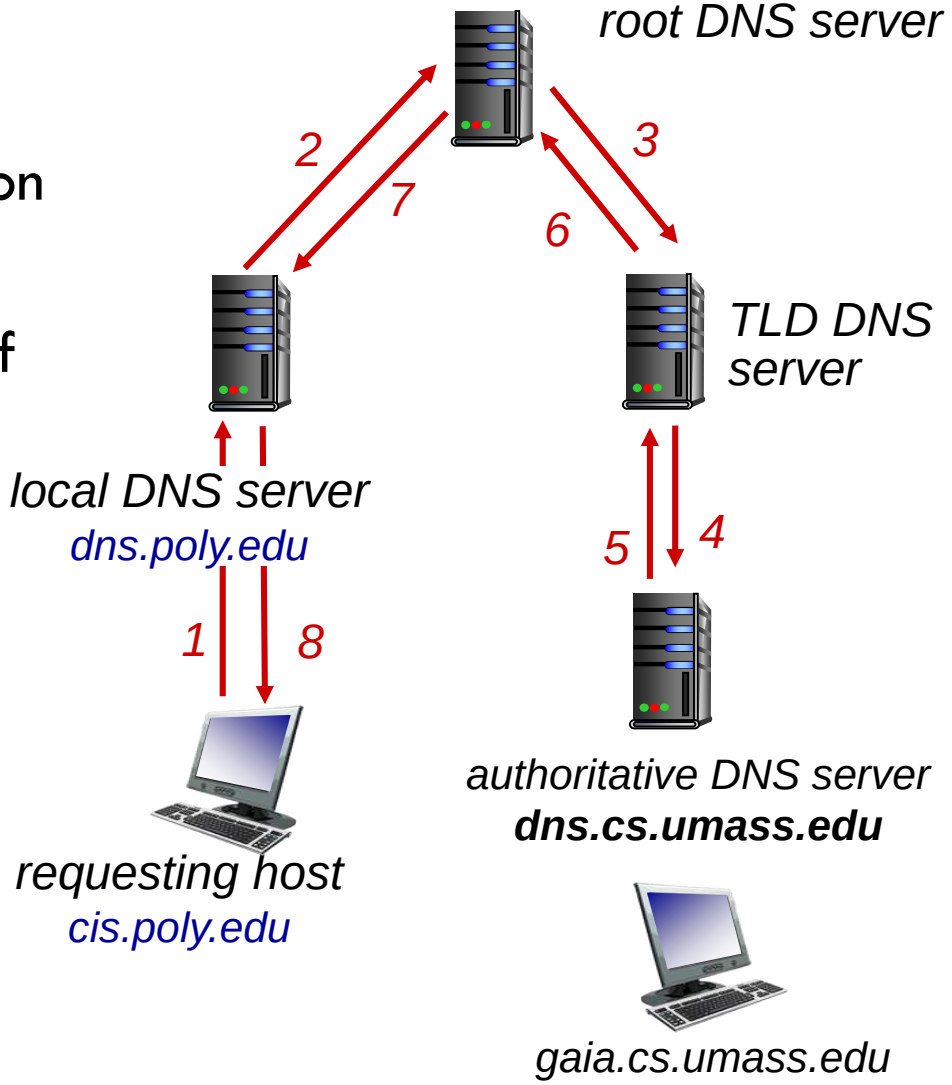
- host at cis.poly.edu wants IP address for gaia.cs.umass.edu

- **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: caching, updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- 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

DNS records

DNS: distributed db storing resource records (RR)

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

type=A

- ***`name`*** *is hostname*
- ***`value`*** *is IP address*

(relay1.bar.foo.com, 145.37.93.126, A)

type=NS

- **`name`** is domain (e.g., foo.com)
- **`value`** is hostname of authoritative name server for this domain

(foo.com, dns.foo.com, NS)

DNS records

DNS: distributed db storing resource records (RR)

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

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*

(foo.com, relay1.bar.foo.com, CNAME)

type=MX

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

(foo.com, mail.bar.foo.com, MX)

Domain Resource Records (1)

- The key resource records in the namespace are IP addresses (A/AAAA) and name servers (NS), but there are others too (e.g., MX)

Type	Meaning	Value
SOA	Start of authority	Parameters for this zone
A	IPv4 address of a host	32-Bit integer
AAAA	IPv6 address of a host	128-Bit integer
MX	Mail exchange	Priority, domain willing to accept email
NS	Name server	Name of a server for this domain
CNAME	Canonical name	Domain name
PTR	Pointer	Alias for an IP address
SPF	Sender policy framework	Text encoding of mail sending policy
SRV	Service	Host that provides it
TXT	Text	Descriptive ASCII text

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 (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)
```

Thank you!