Application Layer Protocols: Examples

Hui Chen
Department of Computer & Information Science
CUNY Brooklyn College

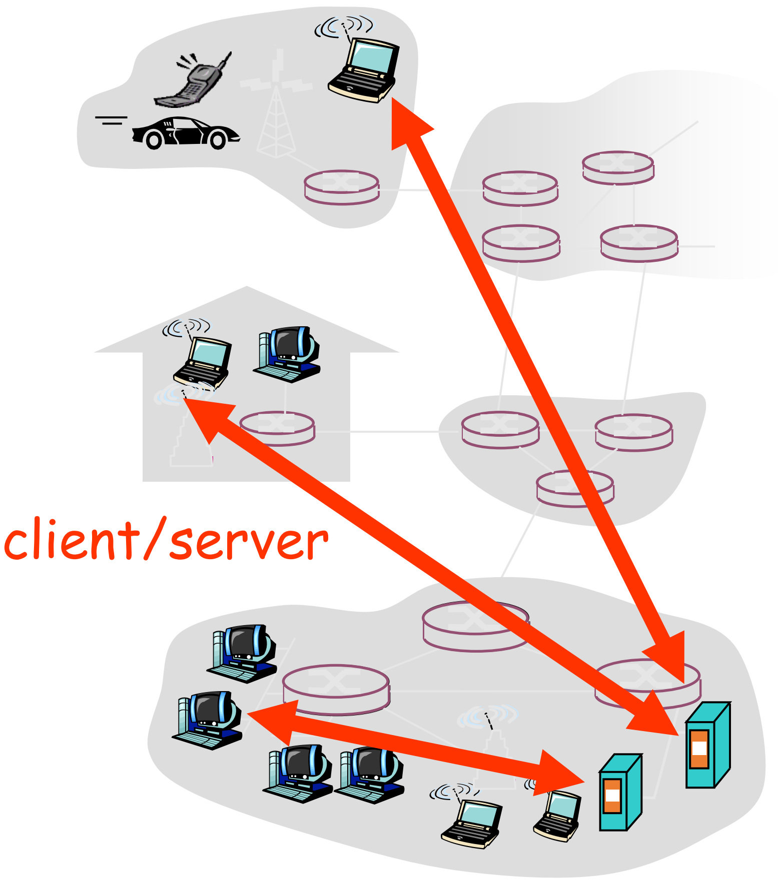
Outline

- Network application architecture
  - Peer-to-peer
  - Client-server
  - Hybrid
- Naming services
  - DNS
- E-mail
  - SMTP
- The World Wide Web
  - HTTP
- Web Services
  - REST

Application architectures

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P

Client-Server Architecture



server:

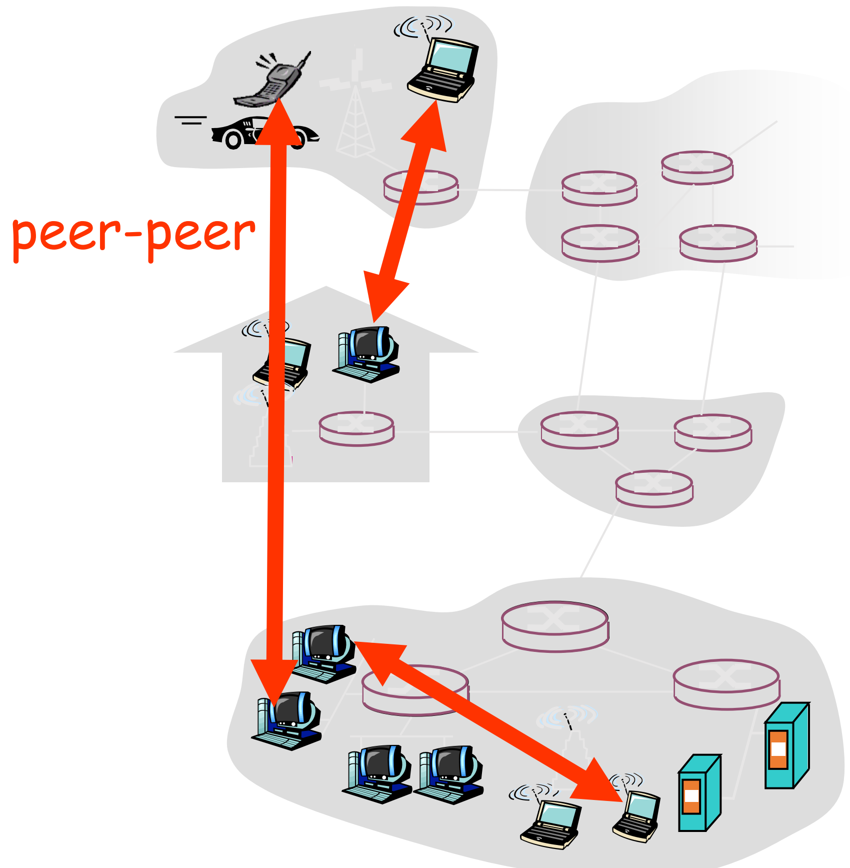
- always-on host
- permanent address
- server farms for scaling

clients:

- communicate with server
- may be intermittently connected
- often have dynamic addresses
- do not communicate directly with each other

Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change addresses



Highly scalable but difficult to manage

Hybrid of Client-Server and P2P

- centralized server
  - finding address of remote party vs. user registers its address with central server when it comes online
- user contacts central server to find addresses of the others to communicate with
  - client-client connection: direct (not through server)

Questions?

- Network application architecture
  - Peer-to-peer
  - Client-server
  - Hybrid

Naming

- Overview
- Domain Naming System
- Distributed File Systems

Overview

- Why do names do?
  - Identify objects
  - Help locate objects
  - Define membership in a group
  - Specify a role
  - Convey knowledge of a secret
- Name space
  - Defines set of possible names
  - Consists of a set of name to value bindings
    - Example:
      - Value Name
      - 123 ABC
- Resolution mechanism
  - Returns the corresponding value when invoked with a name

Properties

- Names vs. addresses
- Location transparent vs. location-dependent
- Flat vs. hierarchical
- Global vs. local
- Absolute vs. relative
- By architecture vs. by convention
- Unique vs. ambiguous

TCP/IP Name Services and Resolution

- Local names
- Files
  - Examples
    - Unix/Linux: /etc/hosts
    - Windows: C:\WINDOWS\system32\drivers\etc\hosts
- Sun Network Information Service (NIS) and NIS+
- Network Security Services (NSS) Database
  - Example: NSS library for the Berkeley DB
- Light Weight Directory Service (LDAP)
  - Example: OpenLDAP
- Domain Name Service (DNS)

Put things together: real life example: Windows

- Unix/Linux
  - Local names
    - /etc/hostname
  - Naming services
    - System Databases and Name Service Switch configuration file: /etc/nsswitch.conf
    - man nsswitch.conf
- Related configuration files
  - /etc/hosts
  - /etc/resolv.conf

Put things together: real life example: Windows

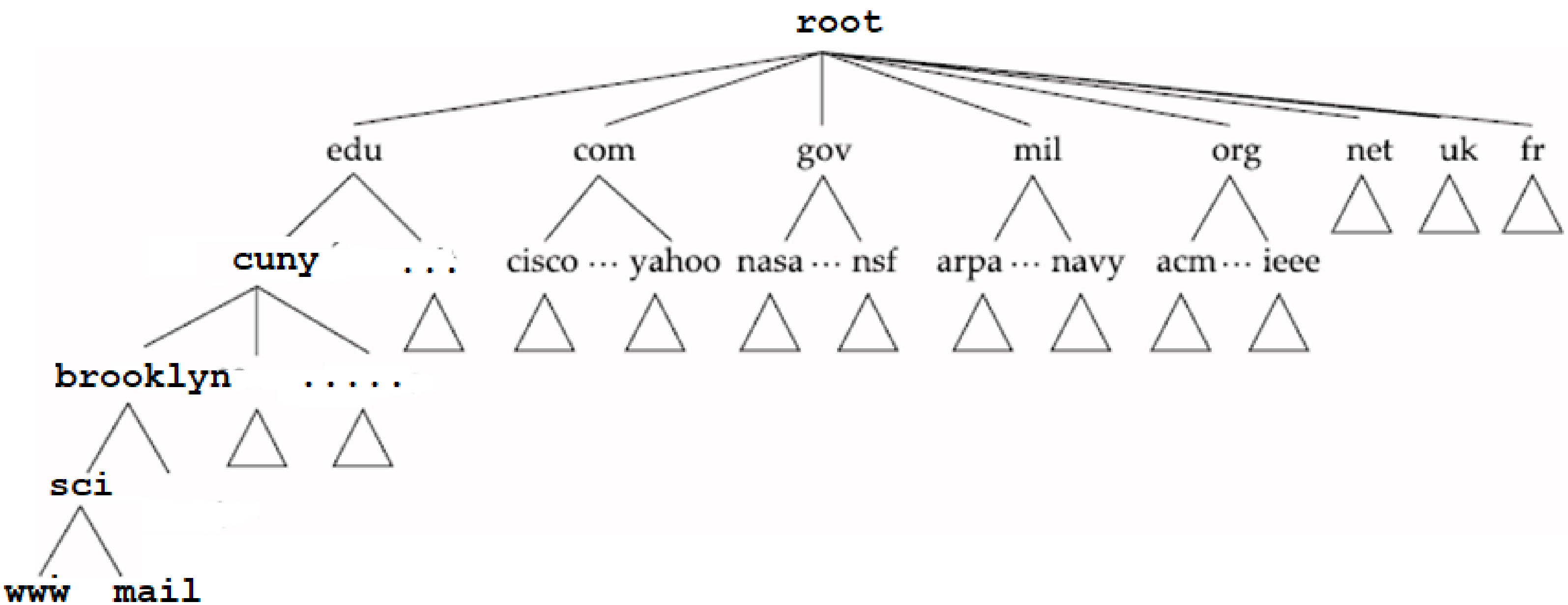
- The configuration of naming service is in Windows Registry – a hierarchical database for storing the settings of Windows and applications
  - The database is stored in several files, e.g.,
    - C:\Windows\system32\Config\SYSTEM
  - Windows provides a tool to work with Windows Registry
    - regedit
- Local names
  - \HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\ComputerName\ActiveComputerName
- Naming services
  - \HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\TCPIP\ServiceProvider

Questions?

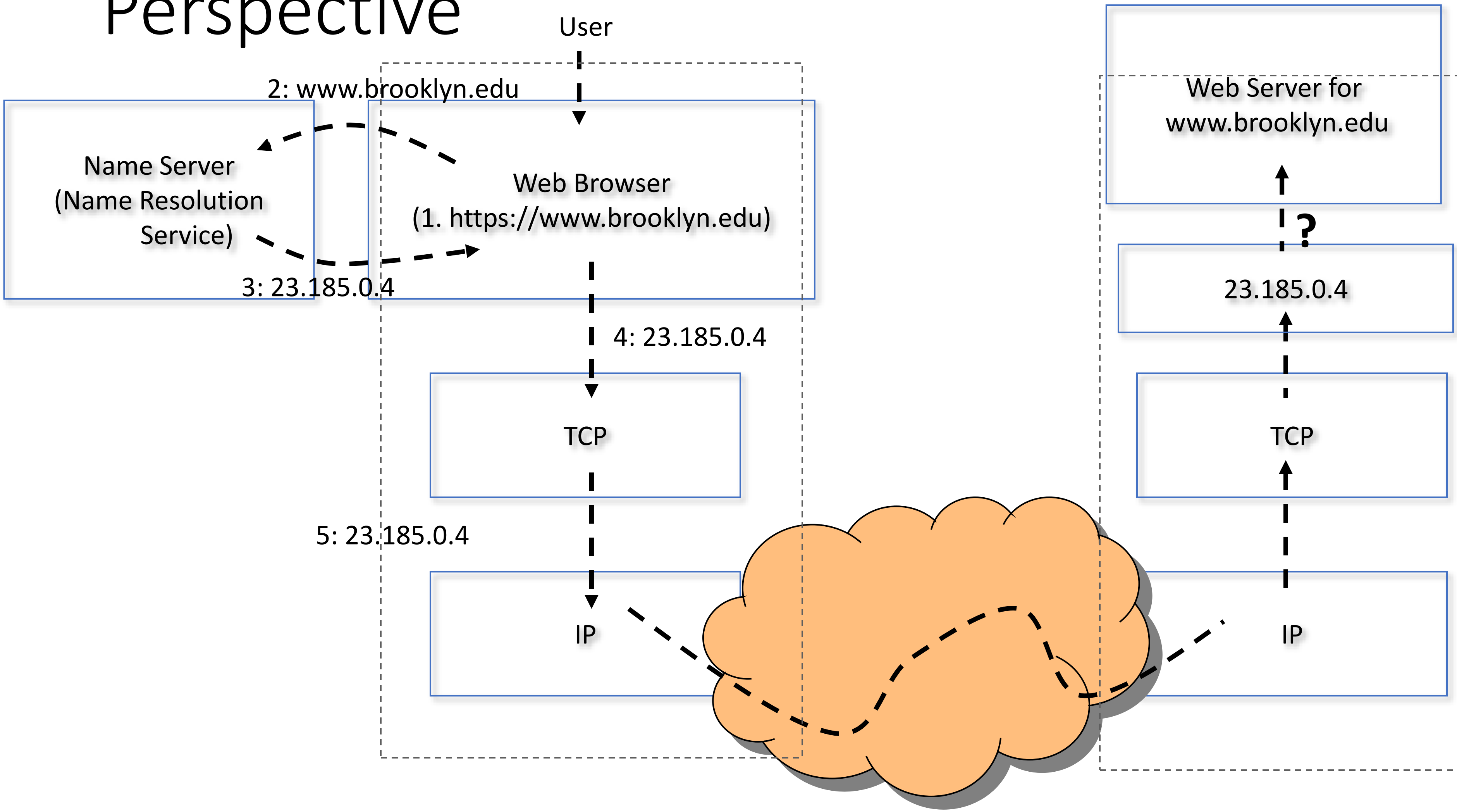
- Overview of name services

Domain Name System

- Provide a directory for names, services, and other resources on the Internet
  - Example:
    - hostname to IP address mapping
    - Email service lookup
- Organized as a hierarchy



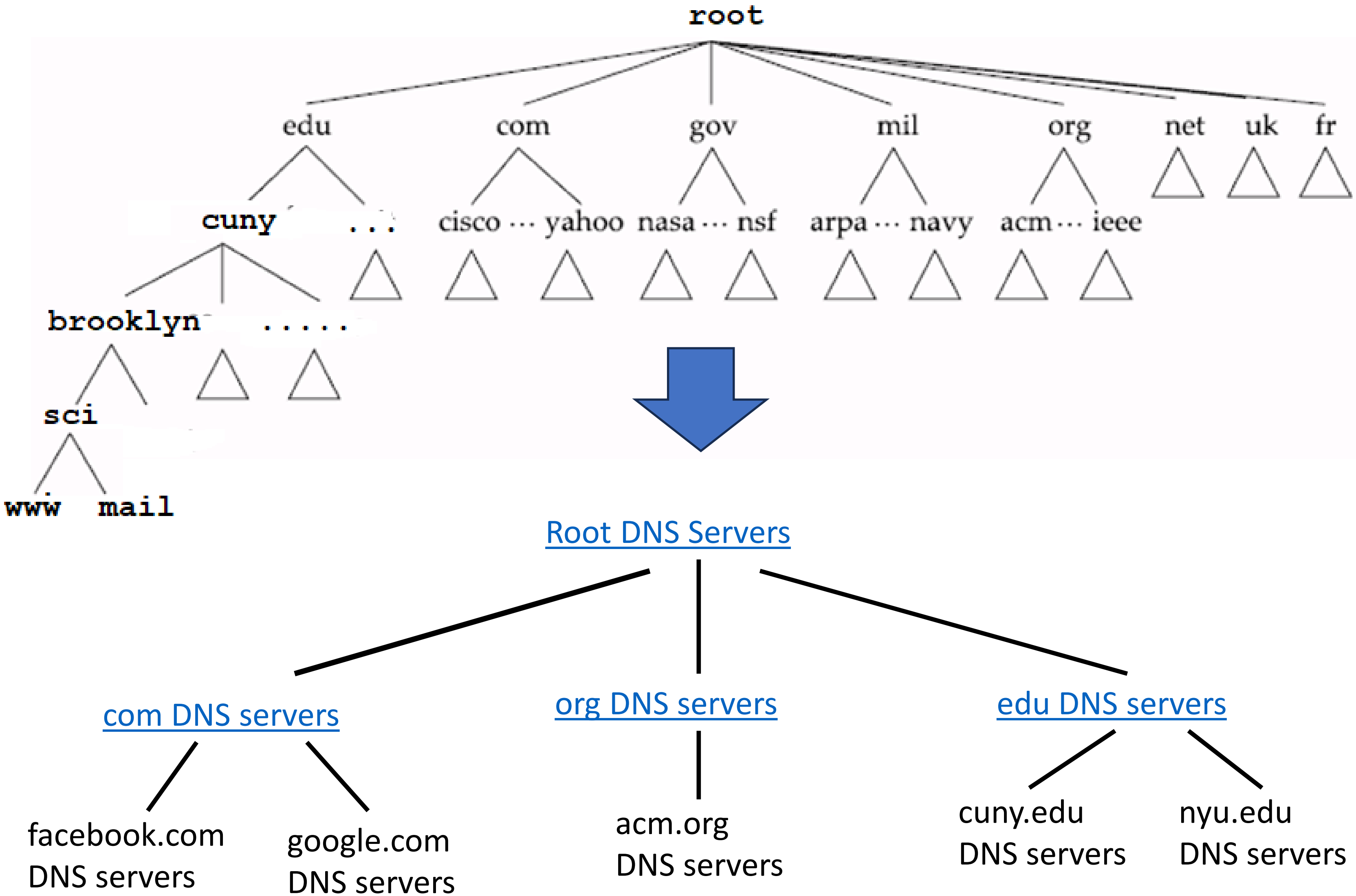
Name Resolution: Application Perspective



Name Resolution: Name System Perspective

- DNS consists of distributed and hierachical database servers
- Name resolution may involve multiple rounds of message exchanges

Distributed and Hierarchical Database



TLD and Authoritative Servers

- Root servers: https://www.iana.org/domains/root/servers
- Top-level domain (TLD) servers
  - Responsible for com, org, net, edu, … and all top-level country domains uk, fr, ca, jp.
    - Examples
      - VeriSign maintains servers for com TLD
      - Educause for edu TLD
    - See https://www.iana.org/domains/root/db
- 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
  - Example: cuny.edu (acme.ucc.cuny.edu)

Local Name Server

- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one.
  - also called “default name server”
- When host makes DNS query, query is sent to its local DNS server
  - acts as proxy, forwards query into hierarchy
- Example: see /etc/resolv.conf on Linux systems

Local Name Servers

- A few well-known public local name servers for testing
- Google's public DNS servers
  - 8.8.8.8, 8.8.4.4
- Level 3's Public DNS servers
  - 209.244.0.3, 209.244.0.4, 4.2.2.1, 4.2.2.2, 4.2.2.3, 4.2.2.4
- OpenDNS's DNS servers
  - 208.67.222.222, 208.67.220.220

Name Resolution

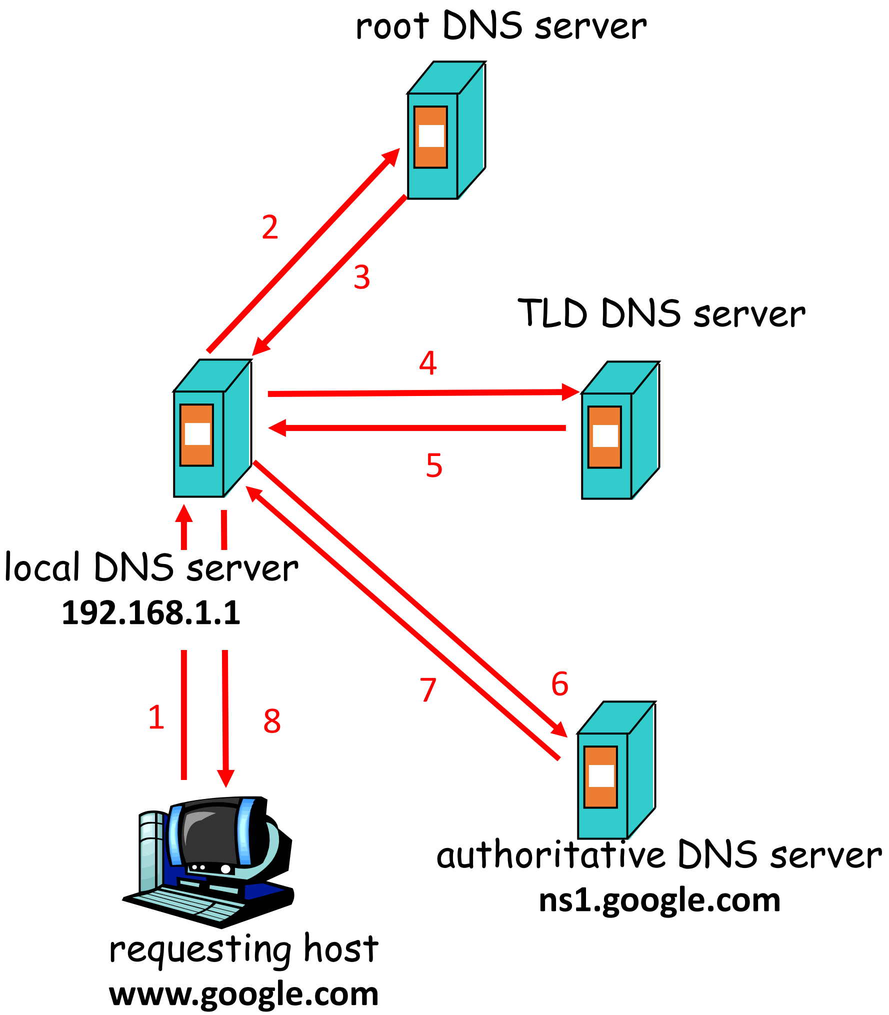
- Strategies
  - Forward
  - Iterative
  - Recursive
- Local server
  - Benefits
    - know the root at only one place (not each host)
    - Can have site-wide cache (site: the network that the local server serves)

DNS Name Resolution: Iterated Query

- Host wants IP address for www.google.com

<u>iterated query:</u>

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



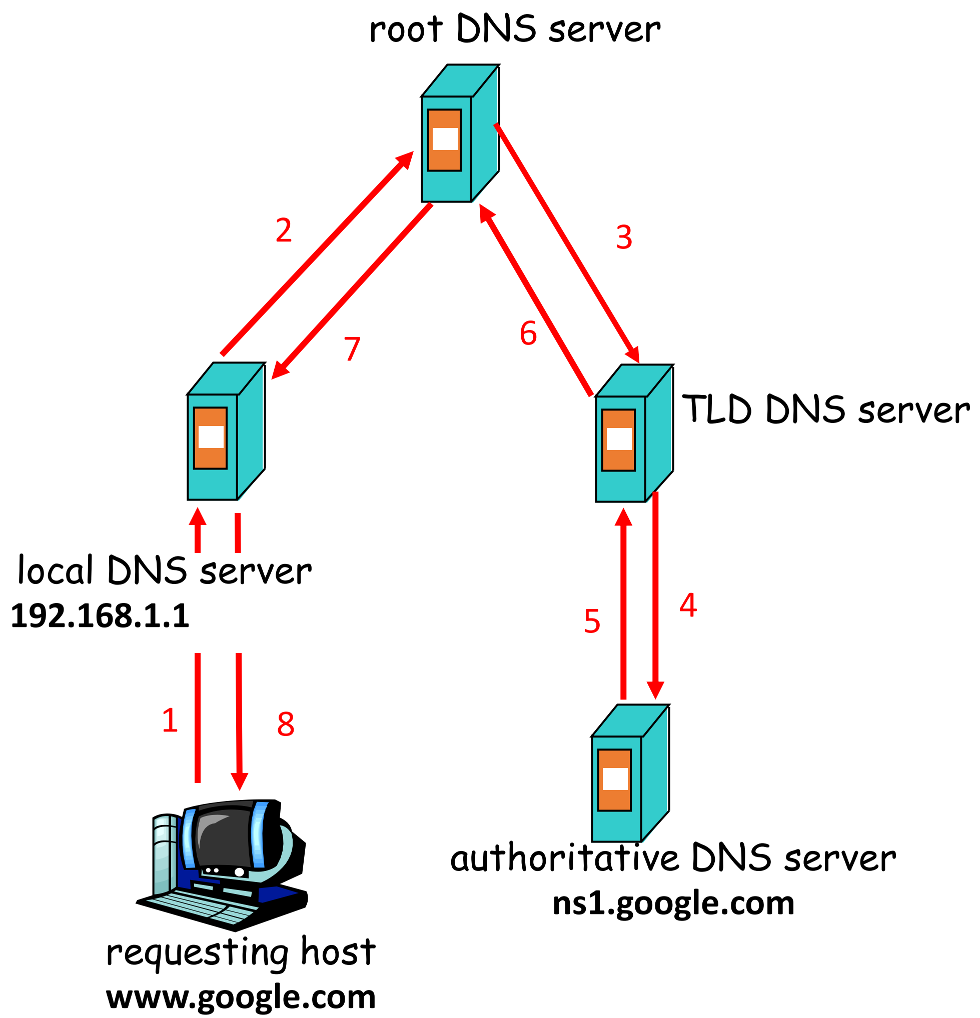
DNS Name Resolution: Recursive Query

Local DNS servers are generally configured to use recursive query

Host wants IP address for www.google.com

<u>recursive query:</u>

- puts burden of name resolution on contacted name server
- heavy load?



DNS: Caching and Updating Records

- Once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time
  - TLD servers typically cached in local name servers
    - Thus root name servers not often visited
- Dynamic update/notify mechanisms
  - RFC 2136
  - https://tools.ietf.org/html/rfc2136
- Implementation of DNS caching is application/system specific

Example: Windows DNS Cache at Hosts

```
C:\>ipconfig /displaydns | more


Windows IP Configuration


    117.200.245.146.in-addr.arpa
    ----------------------------------------
    Record Name . . . . . : 117.200.245.146.in-addr.arpa.
    Record Type . . . . . : 12
    Time To Live  . . . . : 0
    Data Length . . . . . : 8
    Section . . . . . . . : Answer
    PTR Record  . . . . . : cassini.brooklyncollege.local

    Record Name . . . . . : 117.200.245.146.in-addr.arpa.
    Record Type . . . . . : 12
    Time To Live  . . . . : 0
    Data Length . . . . . : 8
    Section . . . . . . . : Answer
    PTR Record  . . . . . : cassini.brooklyn.cuny.edu

    Record Name . . . . . : 117.200.245.146.in-addr.arpa.
    Record Type . . . . . : 12
    Time To Live  . . . . : 0
    Data Length . . . . . : 8
    Section . . . . . . . : Answer
    ......
```

Example: Chrome/Firefox DNS Cache

- Firefox Web Browser
  - Enter in the address bar
  - about:networking#dns
- Chrome Web Browser
  - Enter in the address bar
  - chrome://net-internals/#dns

Questions?

- DNS
  - Name hierarchy
  - Server hierarchy

DNS Records

DNS: distributed db storing resource records (RR)

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

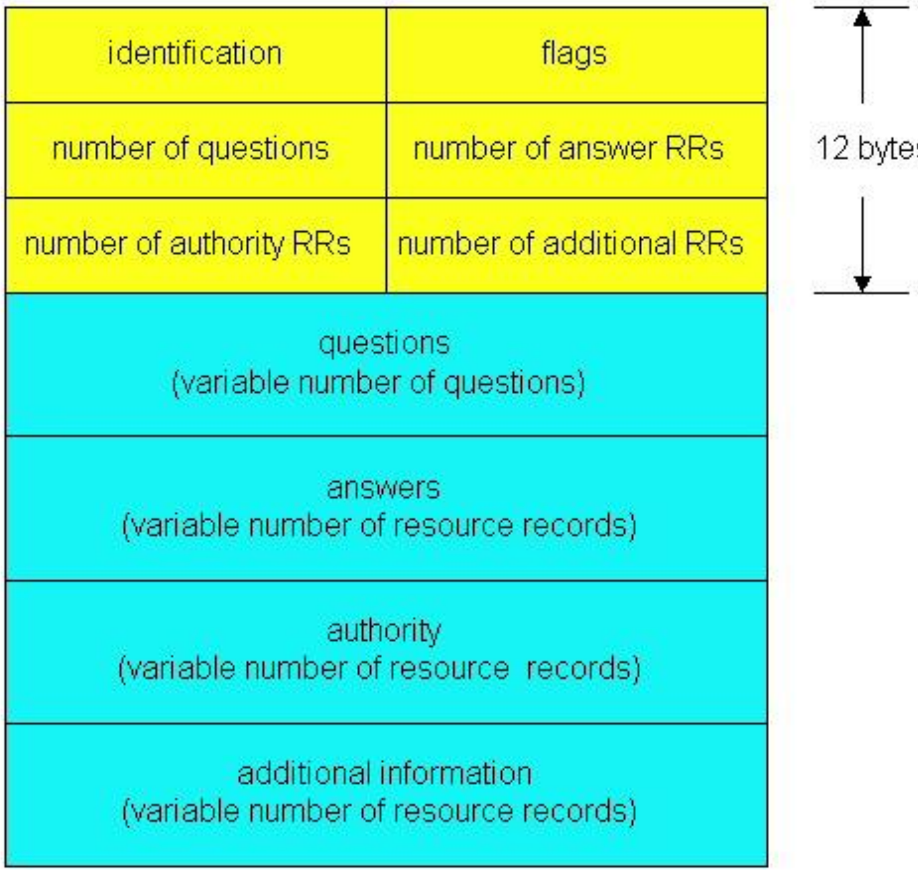
- Type=A
  - **name** is hostname
  - **value** is IP address
- Type=NS
  - **name** is domain (e.g. foo.com)
  - **value** is hostname of authoritative name server for this domain
- Type=CNAME
  - **name** is alias name for some "canonical" (the real) name
    `www.ibm.com` is really
    `servereast.backup2.ibm.com`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

msg header

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



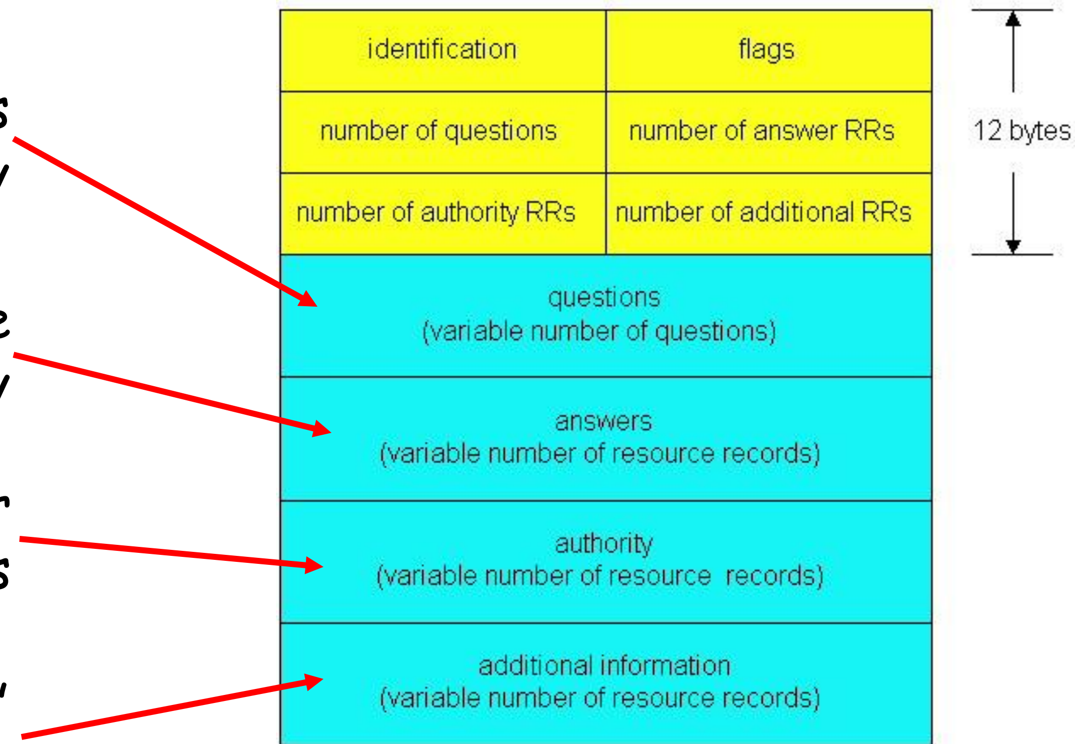
DNS Protocol Messages

Name, type fields for a query

RRs in response to query

records for authoritative servers

additional "helpful" info that may be used



Inserting Records into DNS

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

```
(networkutopia.com, dns1.networkutopia.com,
 NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

- create authoritative server Type A record for www.networkuptopia.com; Type MX record for networkutopia.com

Exercise 1

- Q: How do people get IP address of www.networkutopia.com from a host on Brooklyn College campus given the following?

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

Questions?

- DNS record
- DNS server configuration

Electronic Mail

- Message Format
- Message Transfer
- Mail Reader

Electronic Mail

- Email is one of the oldest network applications
- It is important
    - to distinguish the user interface (i.e., your mail reader) from the underlying message transfer protocols (such as SMTP or IMAP), and
    - to distinguish between this transfer protocol and a companion protocol (RFC 822 and MIME) that defines the format of the messages being exchanged

Electronic Mail: Message Format – Brief Description

- RFC 822: header + body
  - ASCII text
  - MIME → all sorts of data
- Header
  - <CRLF> terminated lines
    - To: ….
    - From: …
- Body
  - MIME
    - Header lines
      - MIME-Version
      - Content-Description: such as Subject: line
      - Definitions content types: Can be multipart
      - Encoding method: Example: base64
- Header and Body is separated by a blank line

An Example of MIME Email Message

```
MIME-Version: 1.0
Content-Type: multipart/mixed;
boundary="-------417CA6E2DE4ABCAFBC5"
From: Alice Smith <Alice@cisco.com>
To: Bob@cs.Princeton.edu
Subject: promised material
Date: Mon, 07 Sep 1998 19:45:19 -0400

---------417CA6E2DE4ABCAFBC5
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Bob,

Here's the jpeg image and draft report I promised.

--Alice

---------417CA6E2DE4ABCAFBC5
Content-Type: image/jpeg
Content-Transfer-Encoding: base64
... unreadable encoding of a jpeg figure

---------417CA6E2DE4ABCAFBC5
Content-Type: application/postscript; name="draft.ps"
Content-Transfer-Encoding: 7bit
... readable encoding of a PostScript document
```

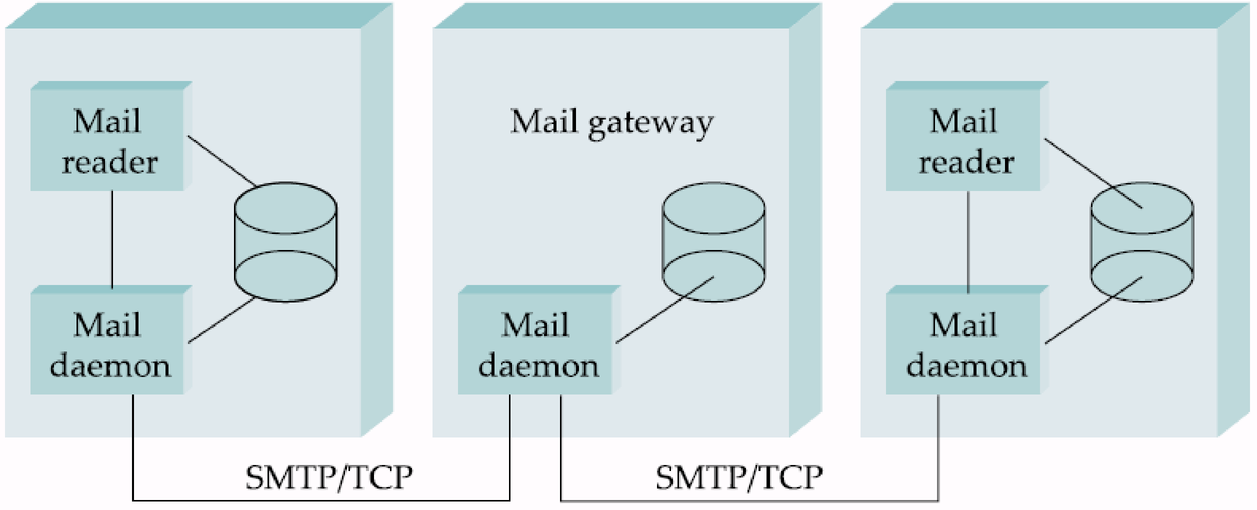
MIME Header

MIME Header

Body

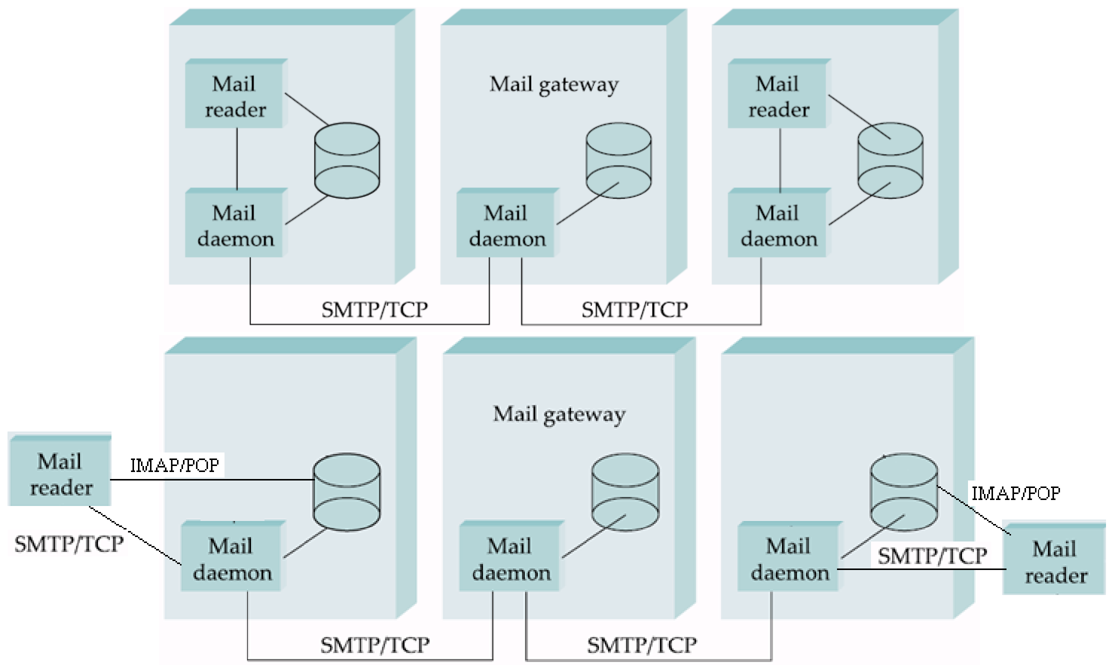
Electronic Mail: Message Transfer

- Message transfer agent (MTA): the mail daemon that uses the Simple Mail Transfer Protocol (SMTP) running over TCP to transmit the message to a daemon running on another machine
- MTA at the receiving end puts incoming messages into the user's mailbox
- Note:
  - SMTP has many different implementations
  - There may be many MTAs in between



Electronic Mail: Mail Reader

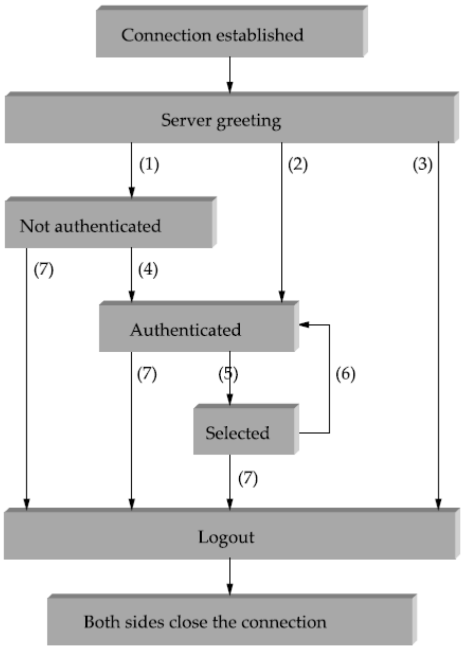
- Users use mail readers to actually retrieve messages from mailbox: read, rely, and save a copy
  - Local reader: reside on the machine where the mailbox is.
  - Remote reader: access mailbox on a remote machine using other protocol
    - Examples: the Post Office Protocol (POP) and the Internet Message Access Protocol (IMAP)



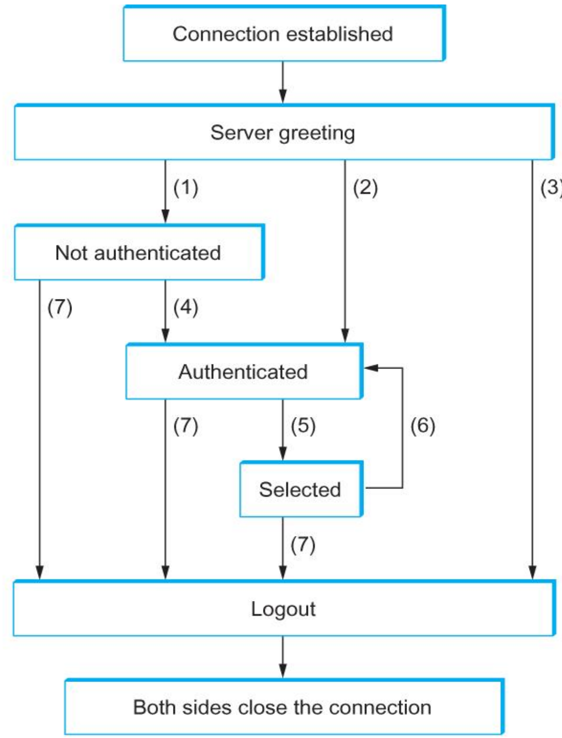
Electronic Mail: IMAP

- IMAP is similar to SMTP in many ways.
- Client/server protocol running over TCP
  - client (running on the user's desktop machine) issues commands in the form of <CRLF>-terminated ASCII text lines
  - mail server (running on the machine that maintains the user's mailbox) responds in-kind.
  - Begins with the client authenticating him or herself, and identifying the mailbox he or she wants to access.

Electronic Mail: IMAP



(1) connection without preauthentication (OK greeting)
(2) preauthenticated connection (PREAUTH greeting)
(3) rejected connection (BYE greeting)
(4) successful LOGIN or AUTHENTICATE command
(5) successful SELECT or EXAMINE command
(6) CLOSE command, or failed SELECT or EXAMINE command
(7) LOGOUT command, server shutdown, or connection closed



(1) Connection without preauthentication (OK greeting)
(2) Preauthenticated connection (PREAUTH greeting)
(3) Rejected connection (BYE greeting)
(4) Successful LOGIN or AUTHENTICATE command
(5) Successful SELECT or EXAMINE command
(6) CLOSE command, or failed SELECT or EXAMINE command
(7) LOGOUT command, server shutdown, or connection closed

IMAP State Transition Diagram

Exercise 2

- From networking perspective, what happens when someone sends an email to somebody@somecompany.com? Describe messages sent.

Web and HTTP

<u>First some jargon</u>

- 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:

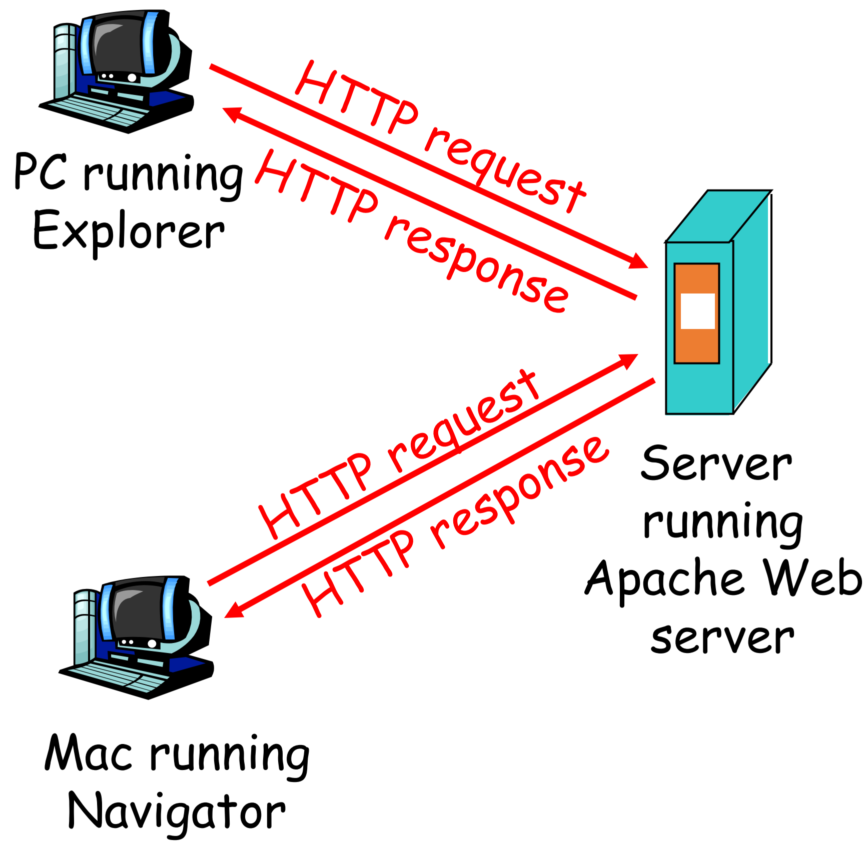
```
www.someschool.edu/someDept/pic.gif
```

host name (www.someschool.edu) | path name (/someDept/pic.gif)

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

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

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, references to 10 jpeg images)

time

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

Nonpersistent HTTP (Continued)

4. HTTP server closes TCP connection.

time

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

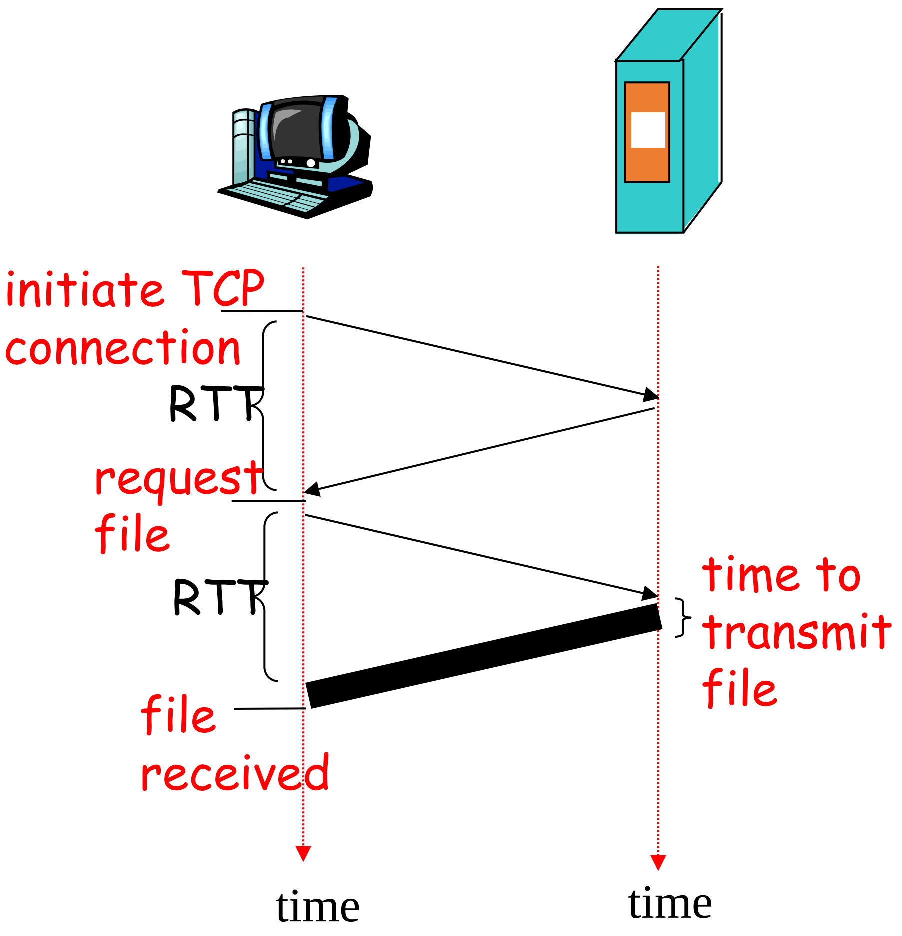
Non-Persistent HTTP: Response time

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
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

HTTP request message

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

request line (GET, POST, HEAD commands)

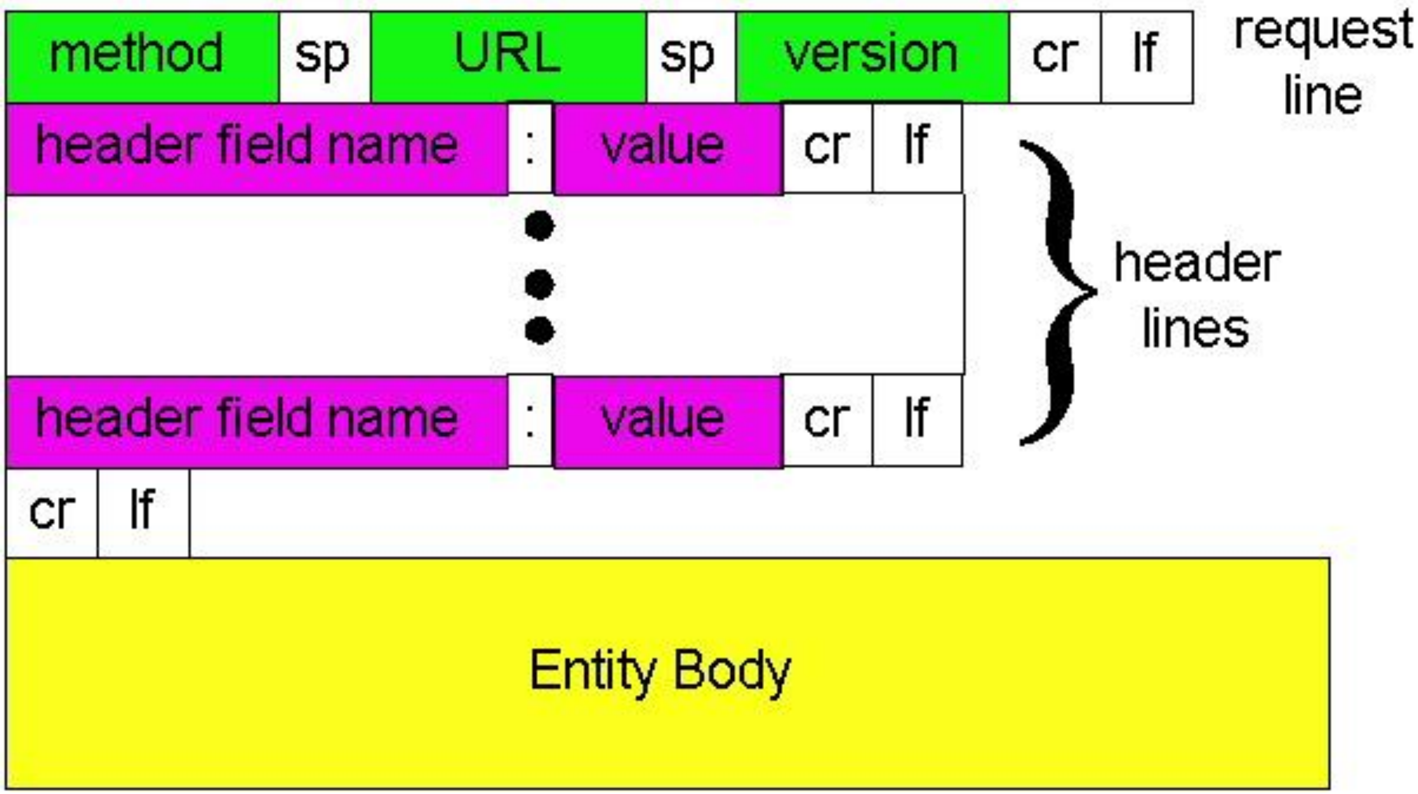
header lines

Carriage return, line feed indicates end of message

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

(extra carriage return, line feed)

HTTP request message: general format



Method types

HTTP/1.0

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

HTTP/1.1

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

Uploading form input

Post method:

- Web page often includes form input
- Input is uploaded to server in entity body

URL method:

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

```
www.somesite.com/animalsearch?monkeys&banana
```

HTTP response message

status line (protocol status code status phrase)

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

header lines

data, e.g., requested HTML file

HTTP response status codes

In first line in server->client response message.
A few sample codes:

200 OK

- request succeeded, requested object later in this message

301 Moved Permanently

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

400 Bad Request

- request message not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet turing.mathcs.vsu.edu 80
```

Opens TCP connection to port 80
(default HTTP server port) at turing.mathcs.vsu.e
Anything typed in sent
to port 80 at turing.mathcs.vsu.edu

2. Type in a GET HTTP request:

```
GET /~hchen/hello.html HTTP/1.1
Host: turing.mathcs.vsu.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!

User-server state: cookies

Many major Web sites use cookies

Four components:

1) cookie header line of HTTP *response* message

2) cookie header line in HTTP *request* message

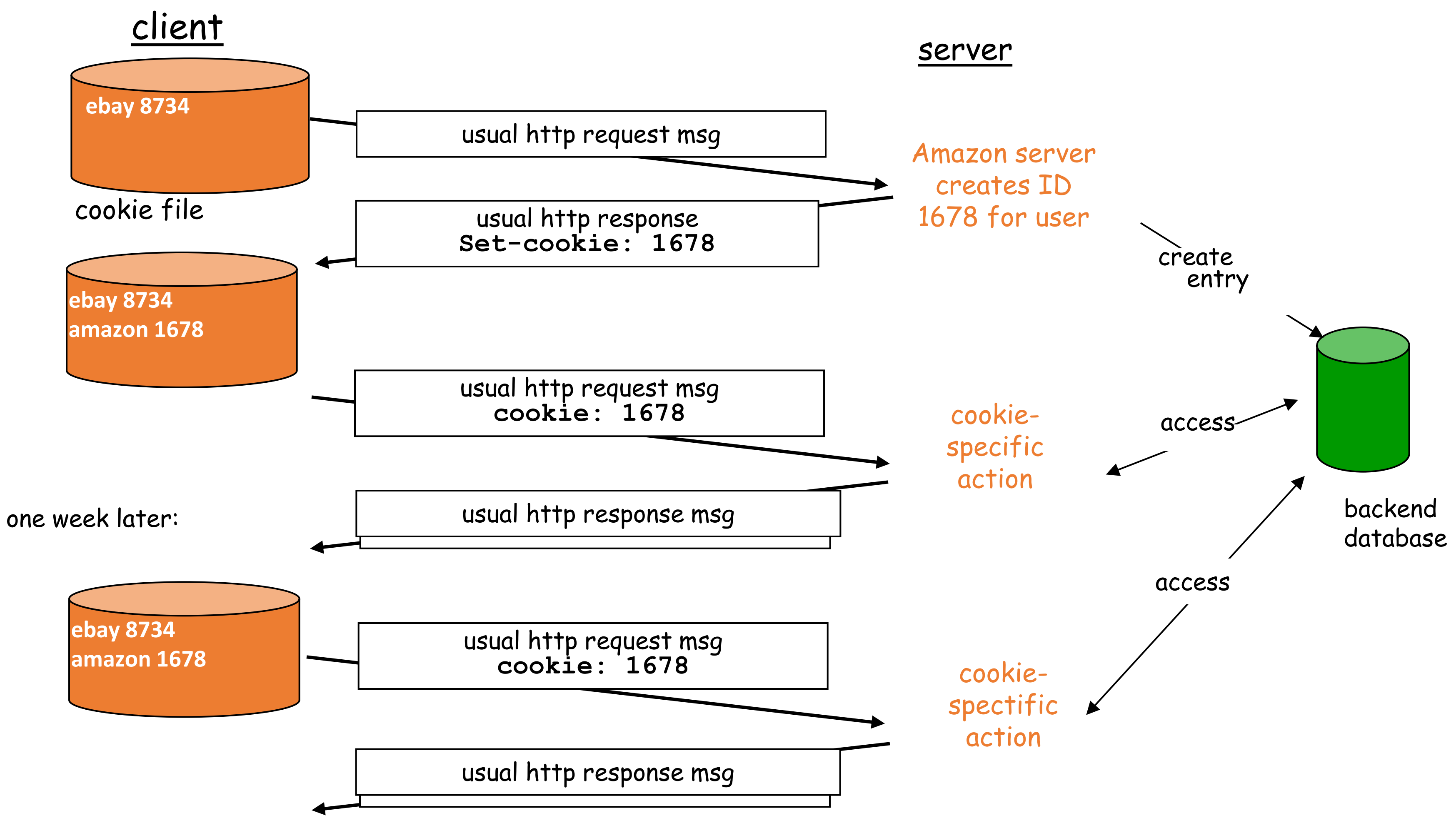
3) cookie file kept on user's host, managed by user's browser

4) back-end database at Web site

Example:

- Susan always access Internet always from PC
- 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 bring:

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

aside

Cookies and privacy:

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

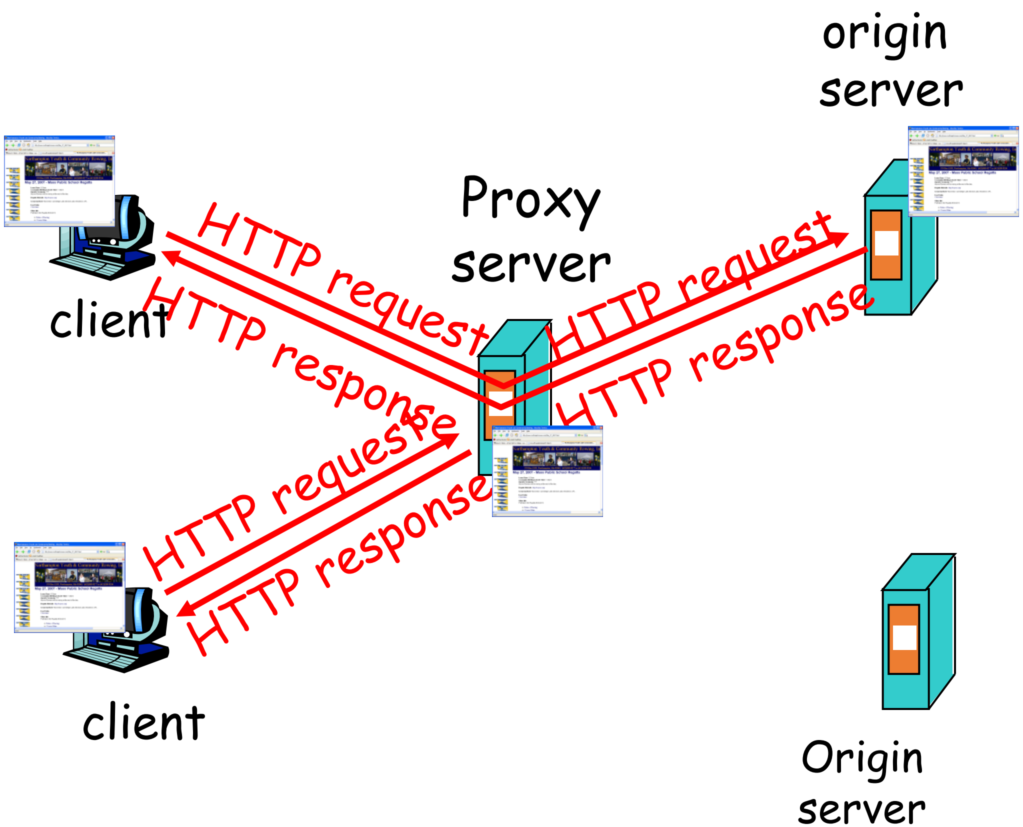
How to keep "state":

- 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



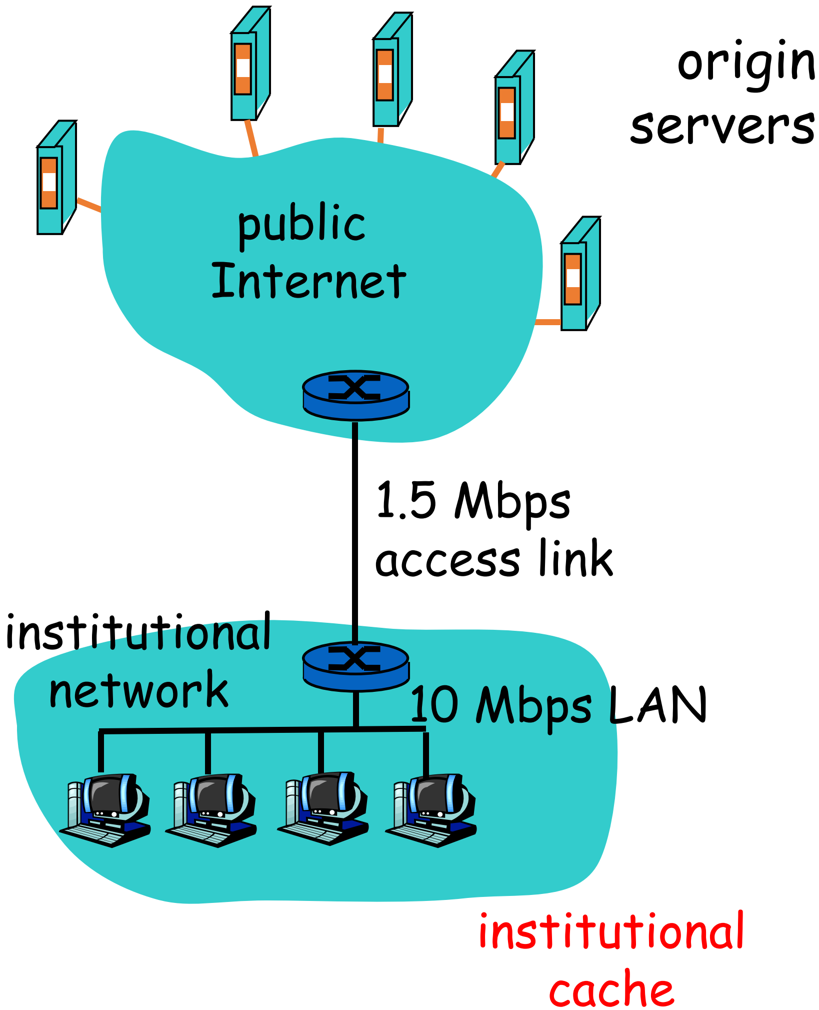
More about Web caching

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

<u>Why Web caching?</u>

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

Caching example



Assumptions

- average object size = 100,000 bits
- avg. request rate from institution’s browsers to origin servers = 15/sec
- delay from institutional router to any origin server and back to router = 2 sec

Consequences

- utilization on LAN = 15%
- utilization on access link = 100%
- total delay = Internet delay + access delay + LAN delay

= 2 sec + minutes + milliseconds

Caching example (cont)

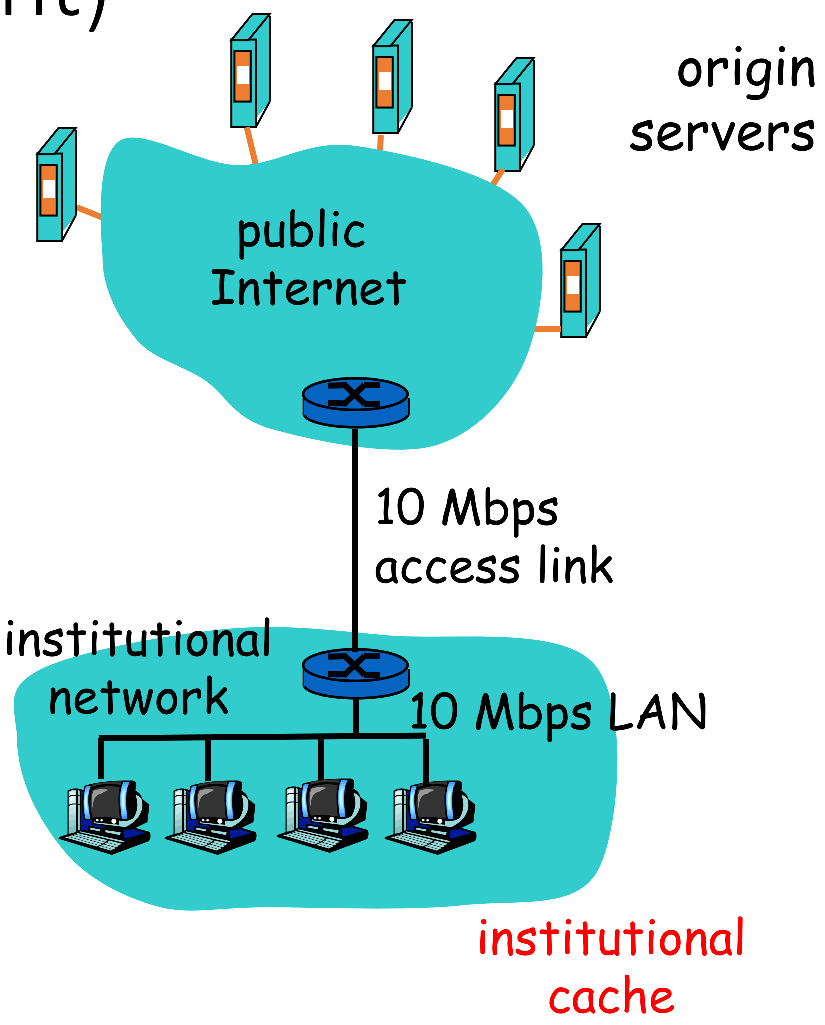
<u>possible solution</u>

- increase bandwidth of access link to, say, 10 Mbps

<u>consequence</u>

- utilization on LAN = 15%
- utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay

  = 2 sec + msecs + msecs
- often a costly upgrade



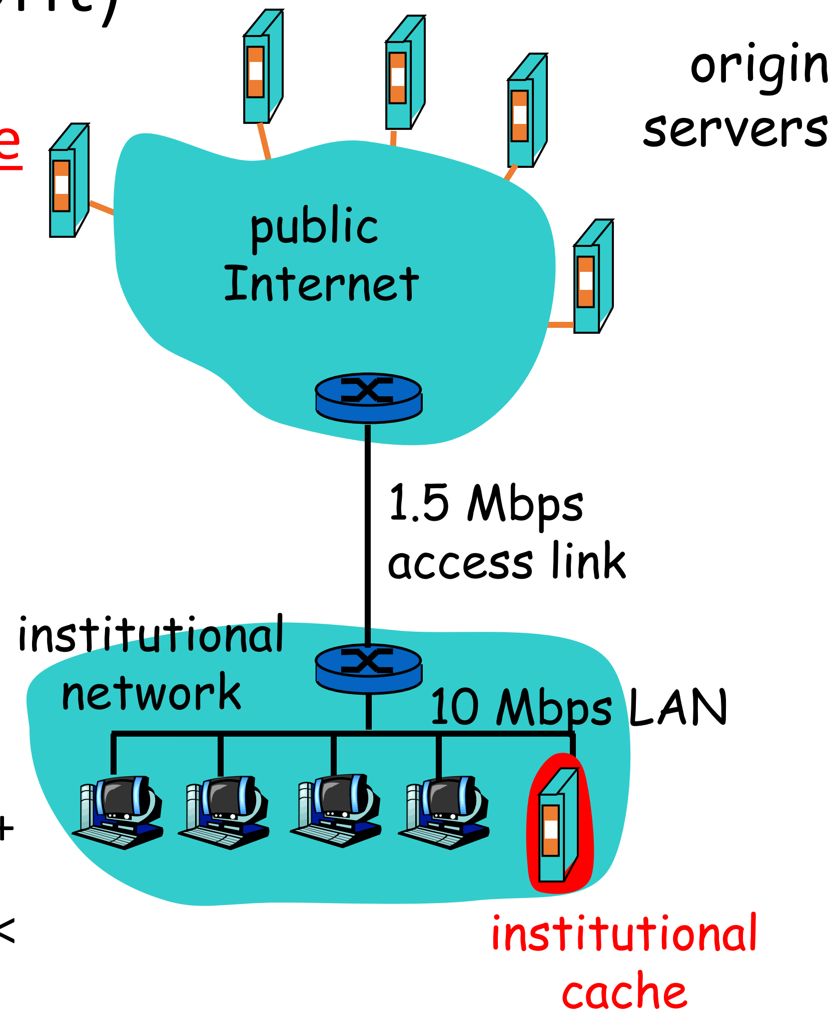
Caching example (cont)

possible solution: install cache

- suppose hit rate is 0.4

consequence

- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total avg delay = Internet delay + access delay + LAN delay = .6*(2.01) secs + .4*milliseconds < 1.4 secs

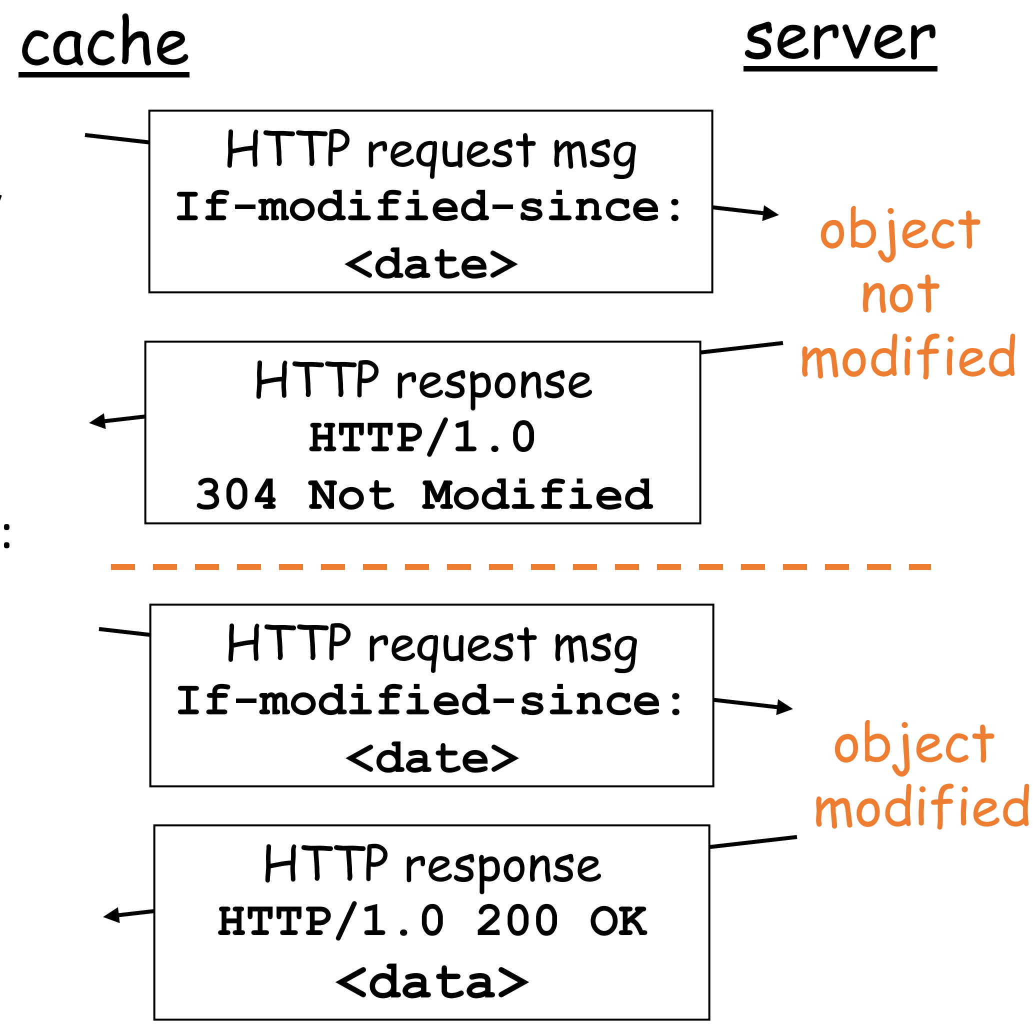


Conditional GET

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

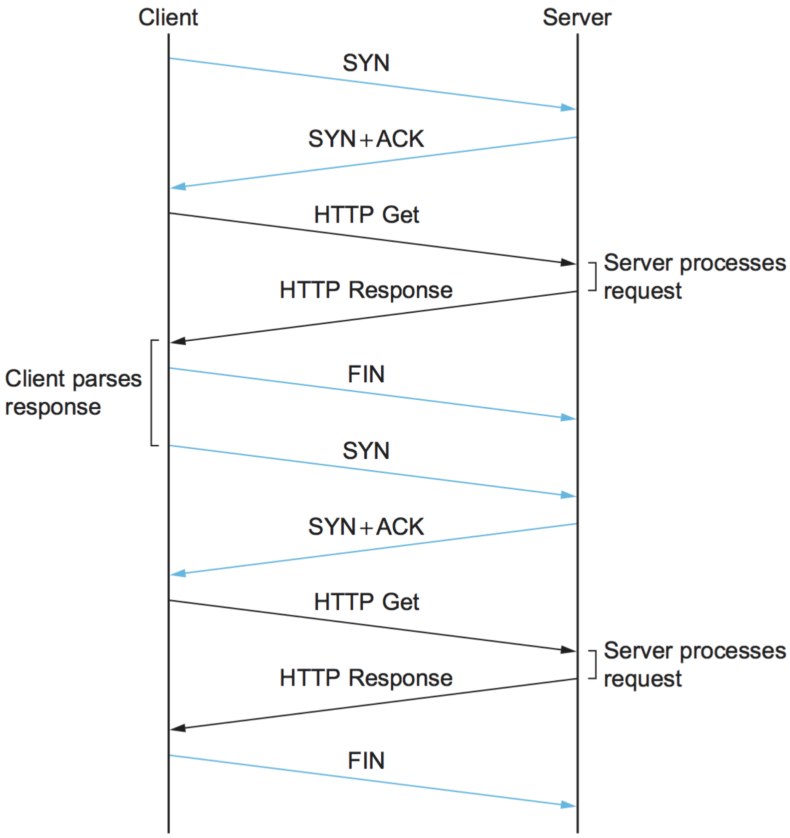
  `If-modified-since: <date>`
- server: response contains no object if cached copy is up-to-date:

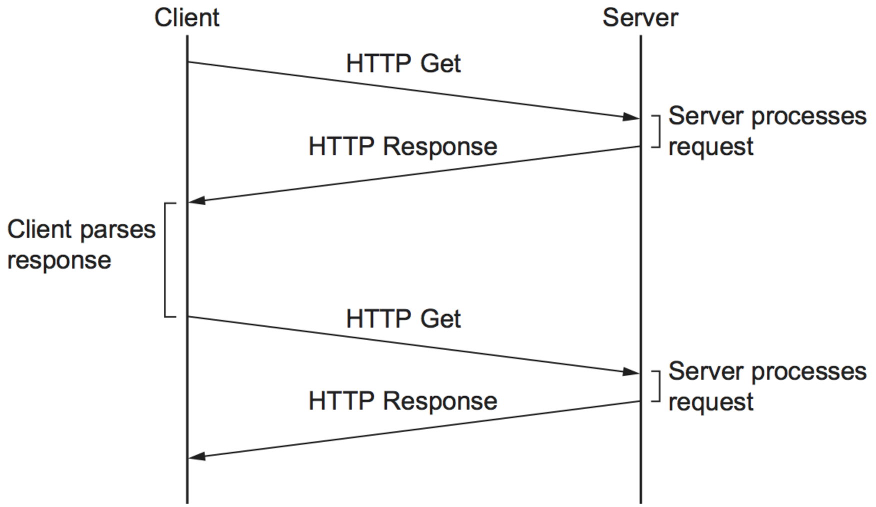
  `HTTP/1.0 304 Not Modified`



Evolution of HTTP (1.0 → 1.1)

- HTTP/1.0 → HTTP/1.1





Evolution of HTTP (→2→3+QUIC)

- HTTP/2
  - Server *push* to minify the data that the server sends back to the client
  - Multiplex several requests on a single TCP connection
    - defines a *channel* abstraction (called *streams*)
    - permits multiple concurrent streams to be active at a given time (each labeled with a unique *stream id*), and
    - limits each stream to one active request/reply exchange at a time.

Evolution of HTTP (→3+QUIC)

- HTTP/3+QUIC
  - Motivation
    - Mismatch: TCP provides a byte-stream abstraction vs. HTTP is a request/response protocol.
    - Application trend: Web applications + Secure channel (TLS/SSL, https)
  - Improvement
    - supports stream multiplexing at the transport layer.
      - a single packet loss only impacts the delivery of the stream that suffered the loss, but does not stall the TCP connection (other streams multiplexed on the connection)
    - reduces the steps required to secure an HTTP connection

Questions

- HTTP
- Evolution of HTTP

Web Services

- *Interactions between a human and a web server* vs. *direct application-to-application communication*
    - What happens when you buy something on Amazon?
- Need to enable speedy development of network applications
    - So, simplify and automate the task of application protocol design and implementation
    - Solution: Web services
        - *SOAP* and *REST*

REpresentational State Transfer (REST)

- Individual Web Services = World Wide Web resources
  - identified by URIs and accessed via HTTP
    - Examples?
- Essentially, the REST architecture is just the Web architecture
  - Strengths: stability and a demonstrated scalability (in the network-size sense)
  - Weakness: HTTP is not well suited to the usual procedural or operation-oriented style of invoking a remote service.
  - Argument: can the weakness be offset by the design of RESTful API? (using more data-oriented and document-passing style design: return XML or JSON objects?)

REST and HTTP

- REST uses the small set of available HTTP methods, such as GET and POST
  - URL path and Header
    - method and other header fields
  - Payload
    - the complexity is shifted from the protocol to the payload
- Design RESTful API
  - URL path and Header
  - define the document structure (i.e., the state representation)
    - Typically, in JSON or XML
- Use RESTful API
  - URL path and Header
  - understand the document structure

Summary

- Network application architecture
  - Peer-to-peer
  - Client-server
  - Hybrid
- Infrastructure Application:
  - name services and DNS
- Traditional applications
  - The World Wide Web
    - HTTP
  - E-mail
    - SMTP
  - Web Service