Module - 1

Application Layer

1.1 principles of network applications

1.2 Web and HTTP

1.3 FTP

1.4 electronic mail
  - SMTP, POP3, IMAP

1.5 DNS

1.6 P2P applications

1.7 socket programming with UDP and TCP

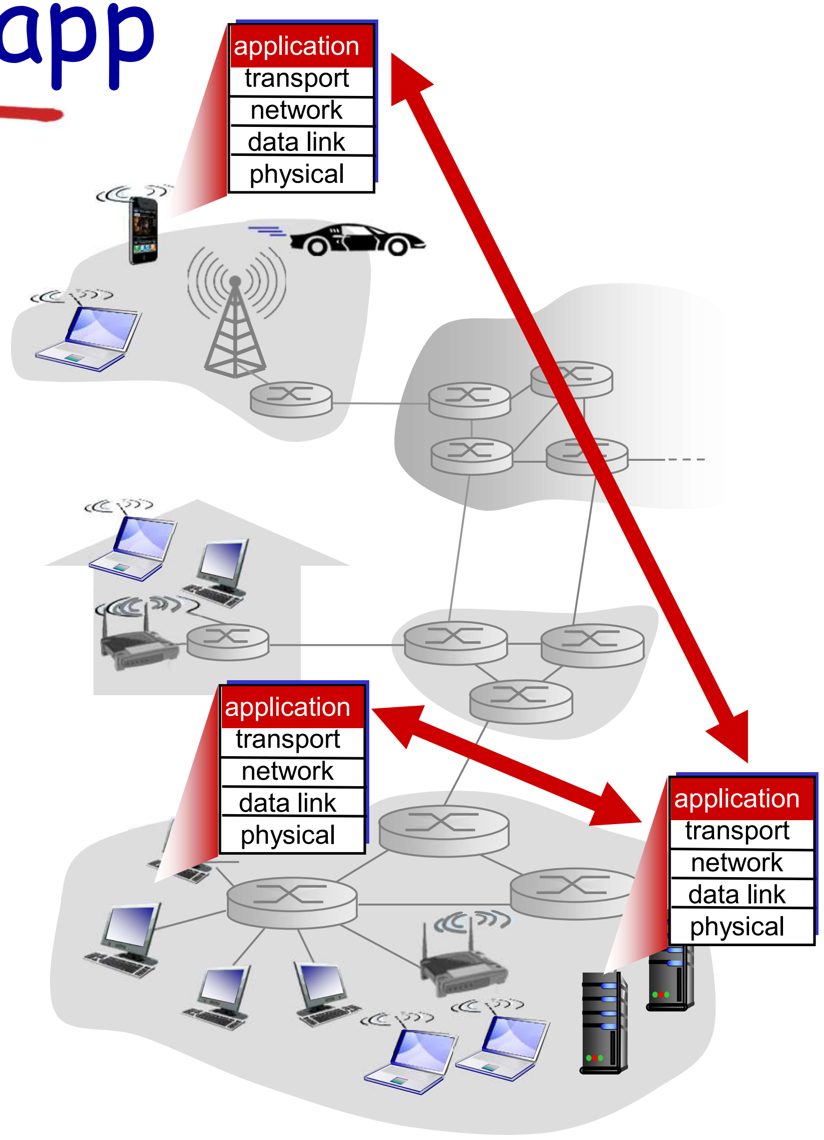
Creating a network app

write programs that:

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

no need to write software for network-core devices

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

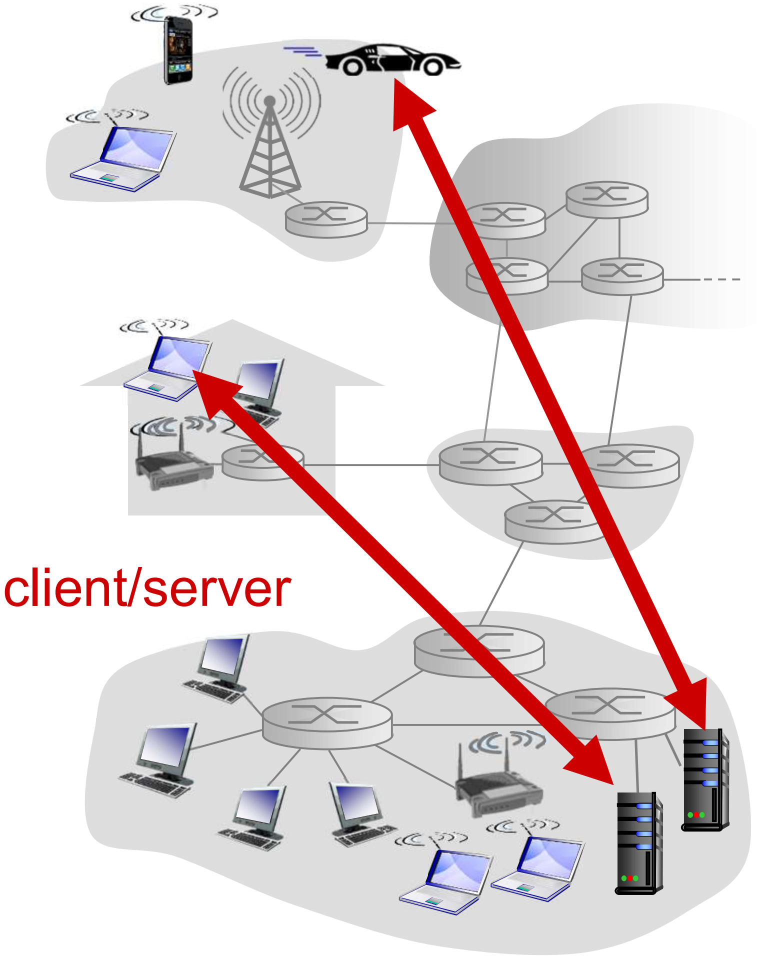


Application architectures

possible structure of applications:

- client-server architecture
- peer-to-peer (P2P) architecture
- BitTorrent is a protocol that enables
  - Fast downloading of large files using minimum internet b/w
  - Maximize transfer speed by gathering pieces of the file you want and downloading these pieces simultaneously from people who already have them

Client-server architecture



server:

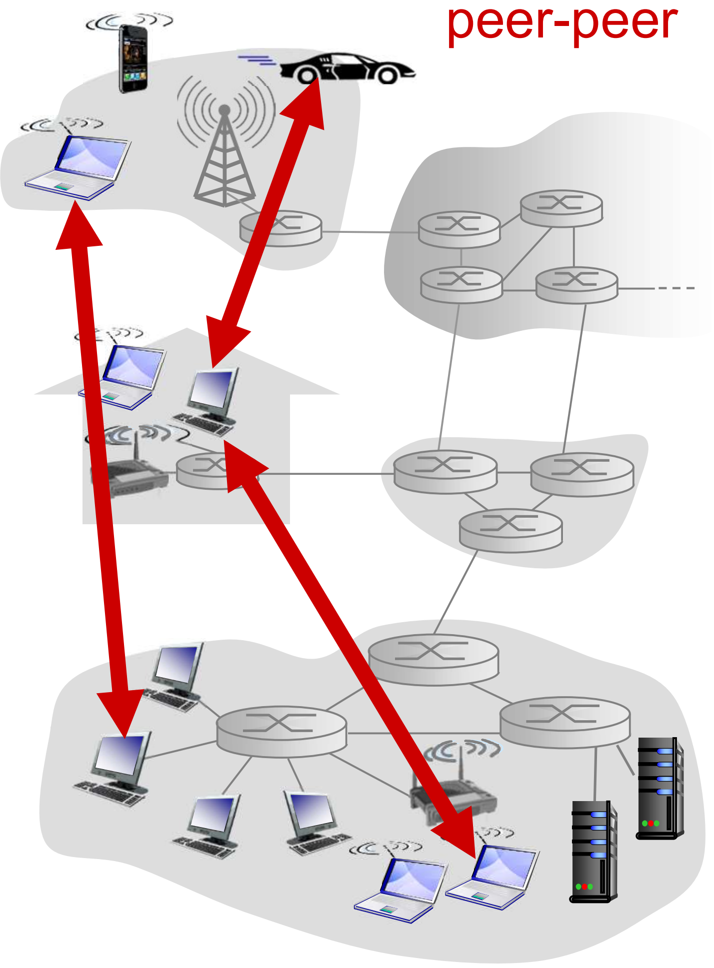
- always-on host
- permanent IP address
- data centers for scaling

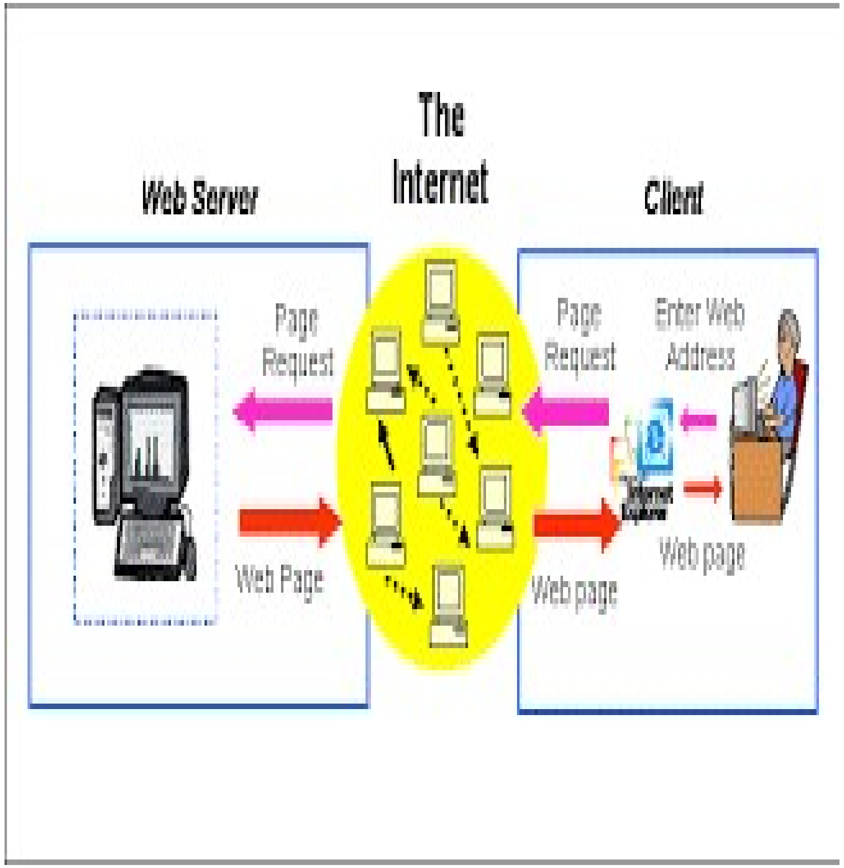
clients:

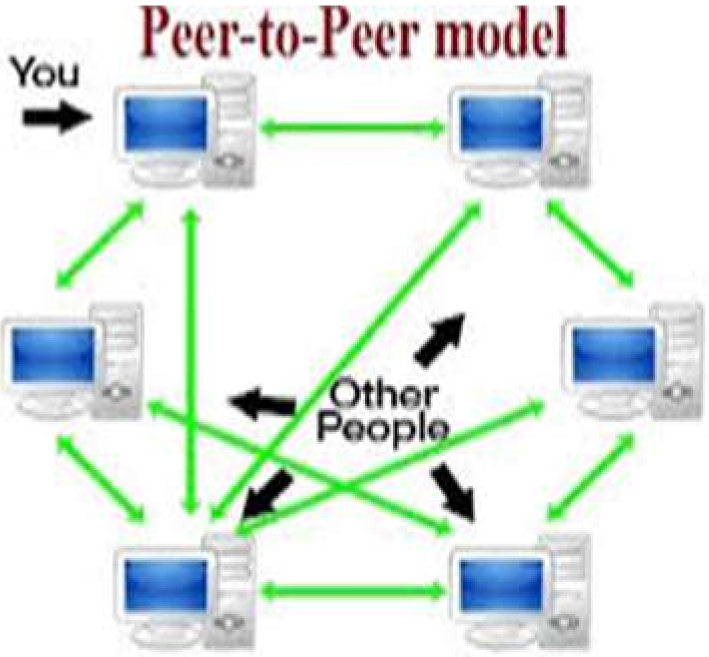
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management




Web Server
The Internet
Client
Page Request
Web Page
Page Request
Enter Web Address
Web page
Web page


Peer-to-Peer model
You
Other People

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

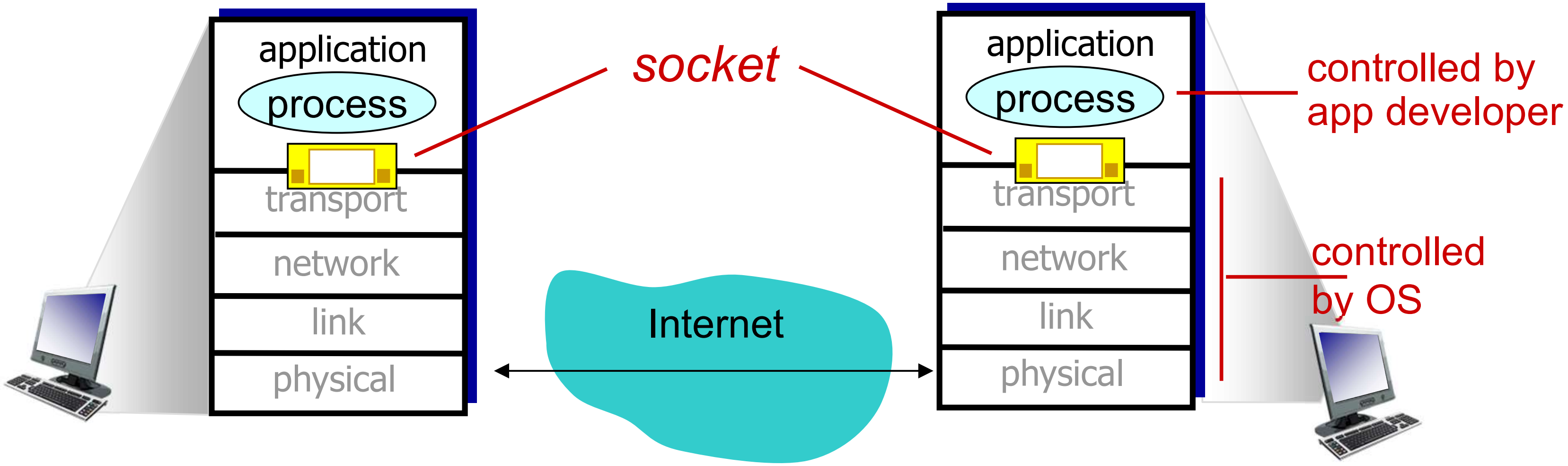
clients, servers

client process: process that initiates communication

server process: process that waits to be contacted

Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address

- *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

Transport services available to application .

Reliable Data Transfer

- 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, …

Some network apps

- e-mail
- web
- text messaging
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)
- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
- search
- …
- …

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	
interactive games	loss-tolerant	few kbps up	yes, few secs
text messaging	no loss	elastic	yes, 100’s
			msec
			yes and no

Transport services provided by the Internet

TCP service:

- *reliable transport*
- *flow control*
- *congestion control*
- *connection-oriented*

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

Application-Layer Protocols

- defines:
- types of messages exchanged
- syntax of the various message types
- semantics of the fields
- Rules

App-layer protocol defines

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

Web and HTTP

- What is web and **HTTP?**
- Where HTTP is implemented?

- Terminology
    - *A* **Web page**

- Faster, smoother browsing
- Works with your phone's camera and contacts
- No periodic updates - just 1 easy download

It's free and always will be.

By clicking Sign Up, you agree to our Terms and that you have read and understand our Data Use Policy, including our Cookie Use.

Web and HTTP

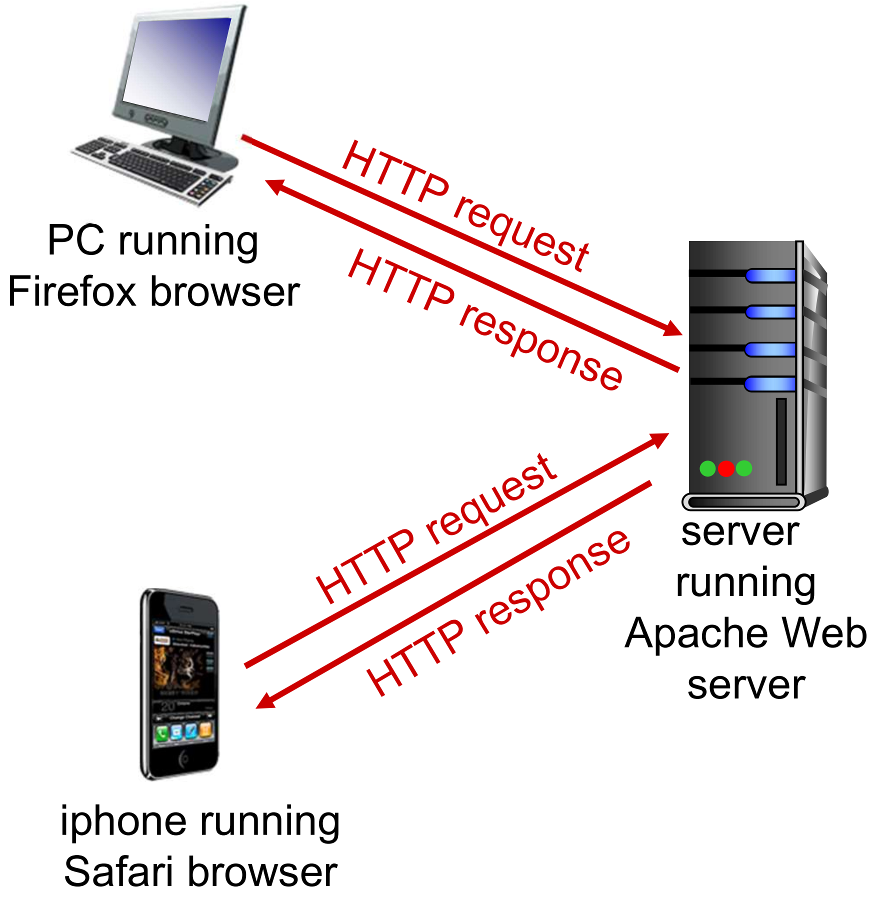
- *web page* consists of *objects*
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects*
- each object is addressable by a *URL,* e.g.,

$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

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 (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

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, server

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

time

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

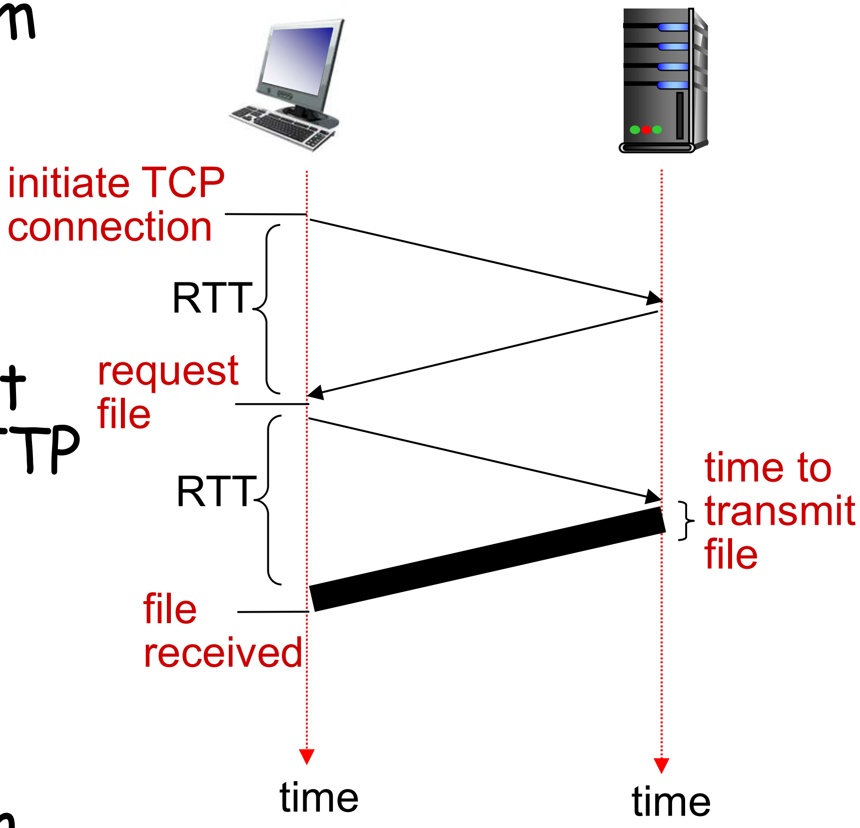
Non-persistent HTTP: response time

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 to return
- file transmission time
- non-persistent HTTP response time =

  2RTT+ file transmission time



Persistent HTTP

non-persistent 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 Message Format

- 2 types of message formats
  - Request Message
  - Response Message

HTTP request message

ASCII (human-readable format)
Example:

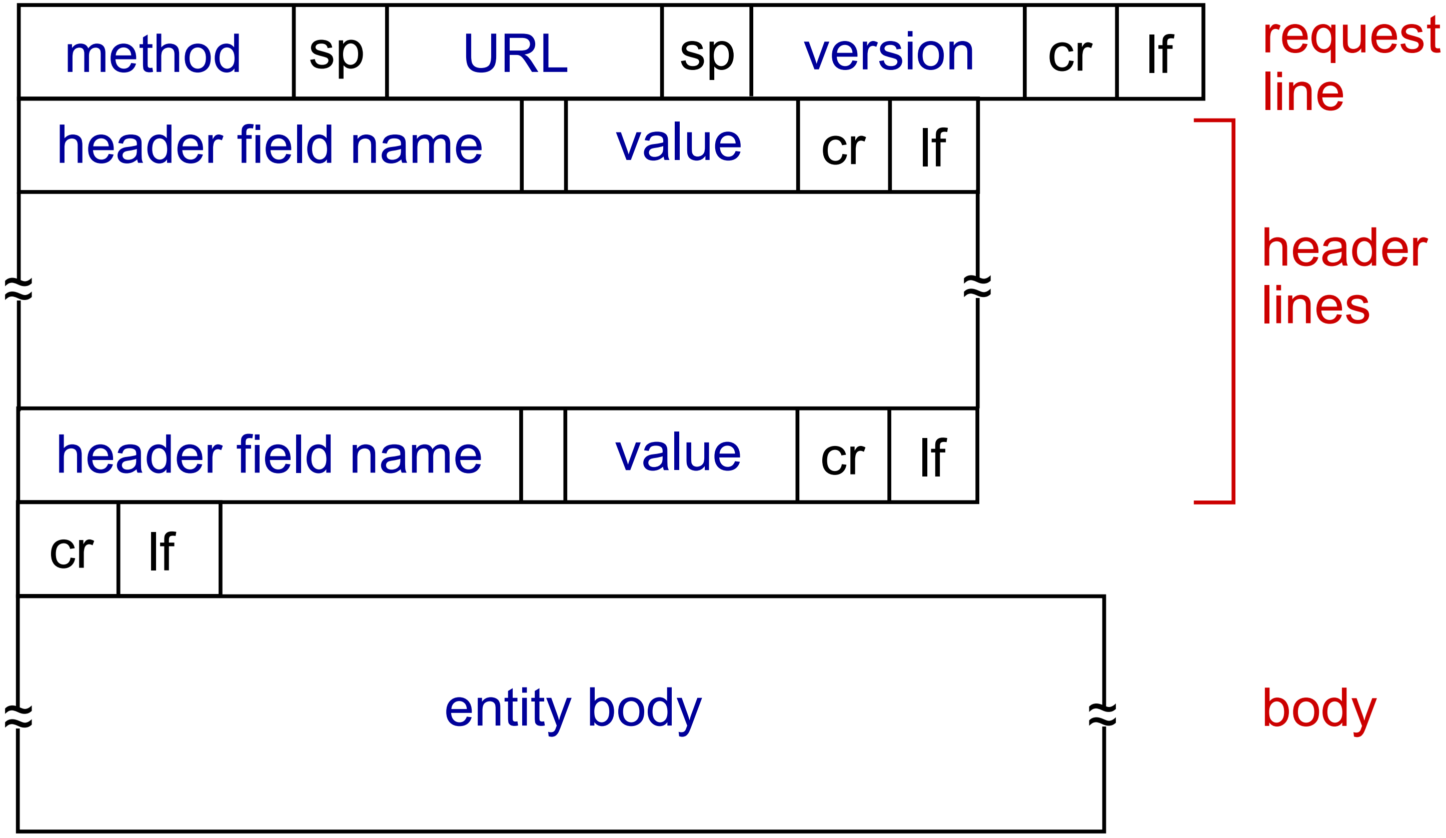
1. GET /somedir/page.html HTTP/1.1
2. Host: www.someschool.edu
3. Connection: close
4. User-agent: Mozilla/5.0
5. Accept-language: fr

Line 1:- request line

3 fields :- method field, URL field & HTTP version

Line 2 to 5 :- header lines

HTTP request message: general format



method sp URL sp version cr lf

request line

header field name value cr lf

header field name value cr lf

header lines

cr lf

entity body

body

	GET The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
2	**HEAD** Same as GET, but transfers the status line and header section only.
3	**POST**A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
4	**PUT** Replaces all current representations of the target resource with the uploaded content.
5	**DELETE** Removes all current representations of the target resource given by a URI

Method types

HTTP/1.0:

- GET
- POST
- HEAD
    - asks server to leave requested object out of response
    - Requests that only header fields(no body) be returned in the response.

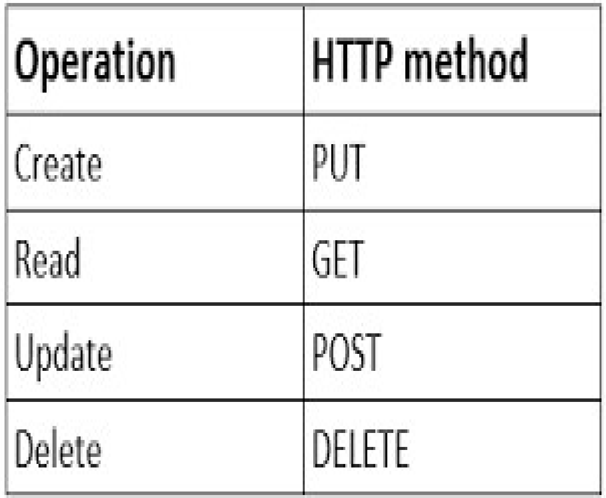
HTTP/1.1:

- 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



Operation	HTTP method
Create	PUT
Read	GET
Update	POST
Delete	DELETE

HTTP response message

Example:

1. **HTTP/1.1 200 OK**
2. **Connection: close**
3. **Date: Tue, 09 Aug 2011 15:44:04 GMT**
4. **Server: Apache/2.2.3 (CentOS)**
5. **Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT**
6. **Content-Length: 6821**
7. **Content-Type: text/html**

(data data data data data ...)

3 sectoins

Line 1:status line → **3 -***protocol version field ,status code & corresponding status message*

Line2 to 7: header lines

Then the entity body

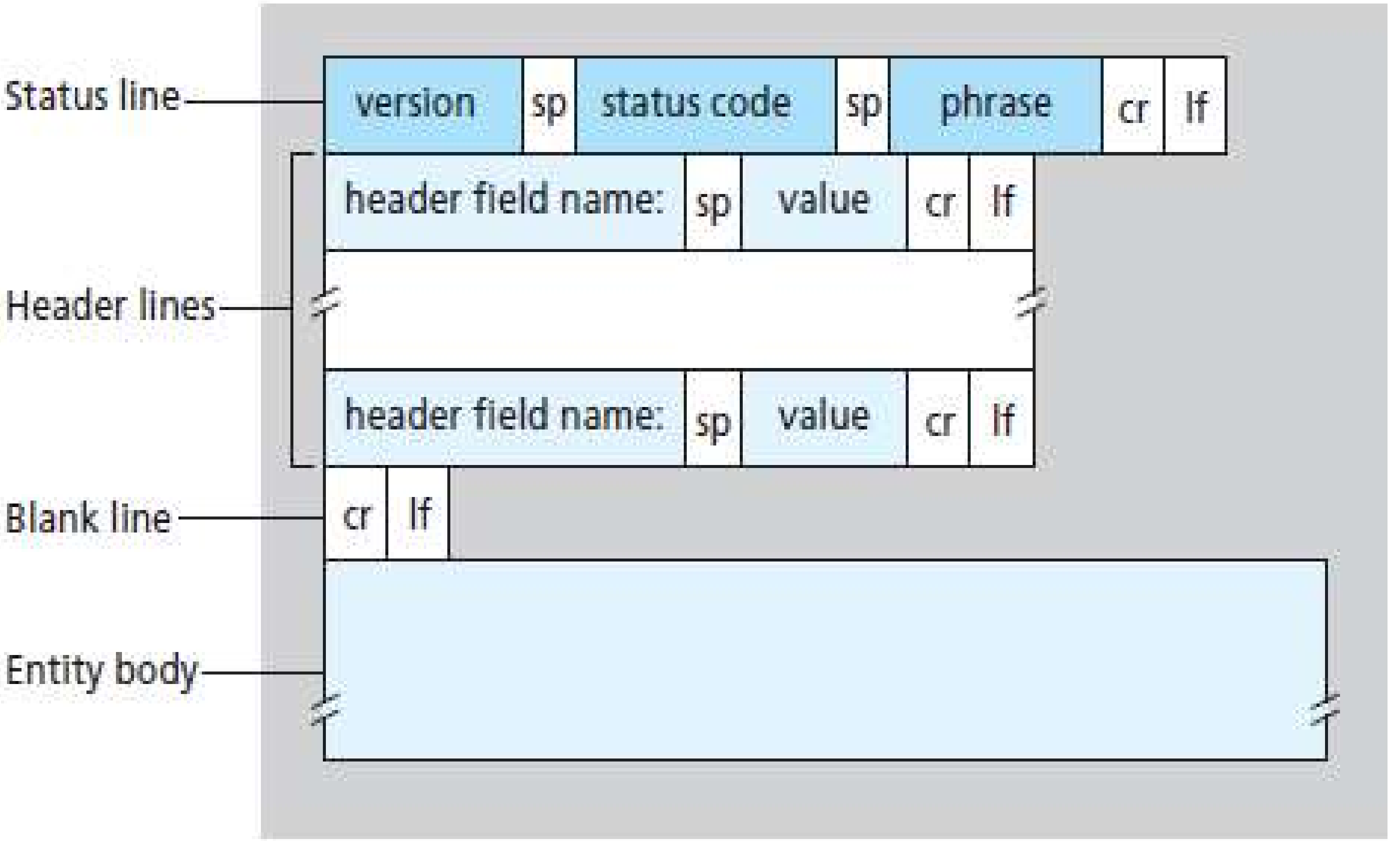


Figure 2.9 ♦ General format of an HTTP response message

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 message (Location:)

  `400 Bad Request`
    - request msg not understood by server

  `404 Not Found`
    - requested document not found on this server

  `505 HTTP Version Not Supported`

User-server Interaction: cookies

many Web sites use cookies

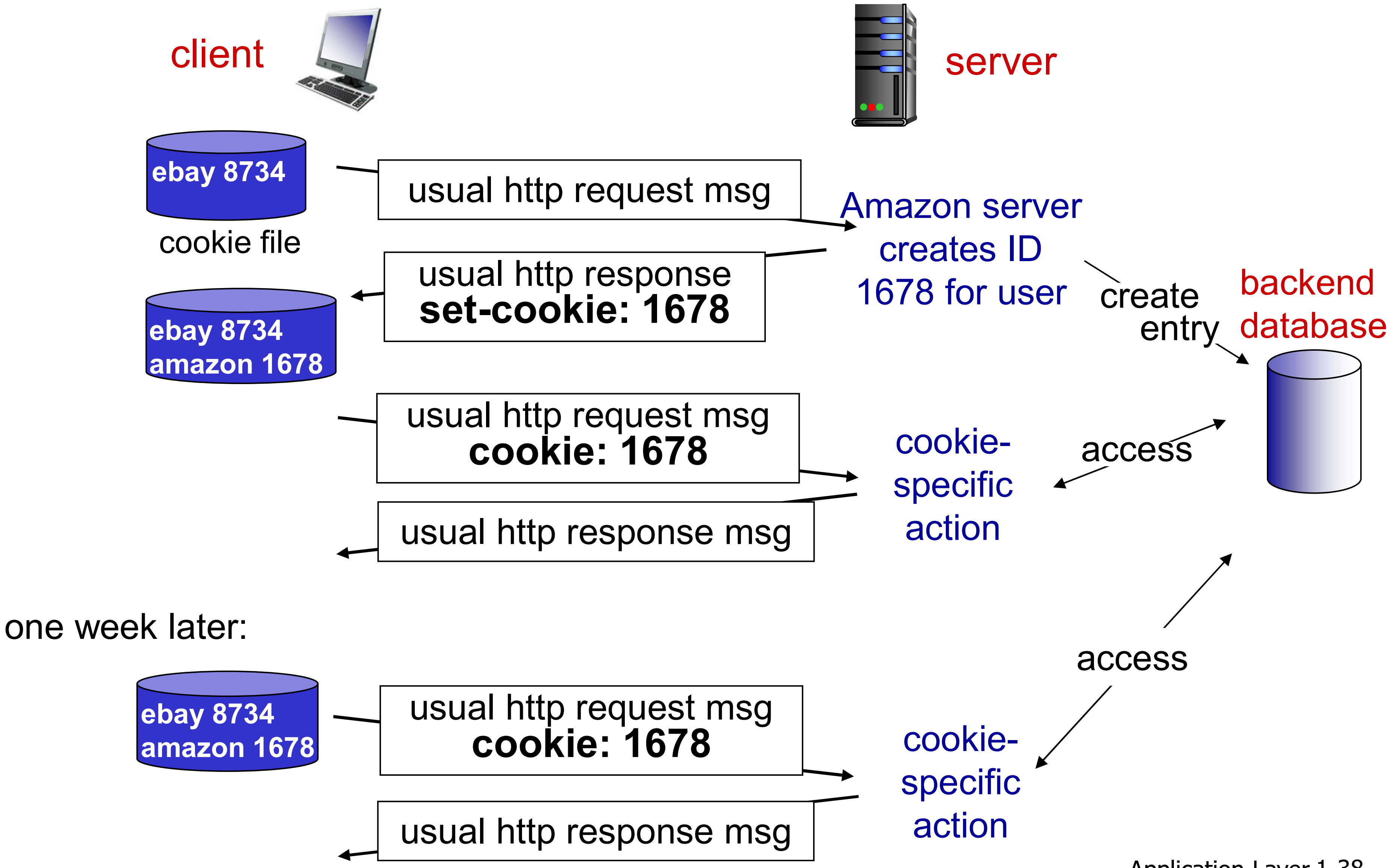
four components:

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:

- Susan always access Internet 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.)



client

server

ebay 8734

cookie file

usual http request msg

Amazon server creates ID 1678 for user

create entry

backend database

usual http response
set-cookie: 1678

ebay 8734
amazon 1678

usual http request msg
cookie: 1678

cookie-specific action

access

usual http response msg

one week later:

ebay 8734
amazon 1678

usual http request msg
cookie: 1678

access

cookie-specific action

usual http response msg

Cookies (continued)

what cookies can be used for:

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

aside

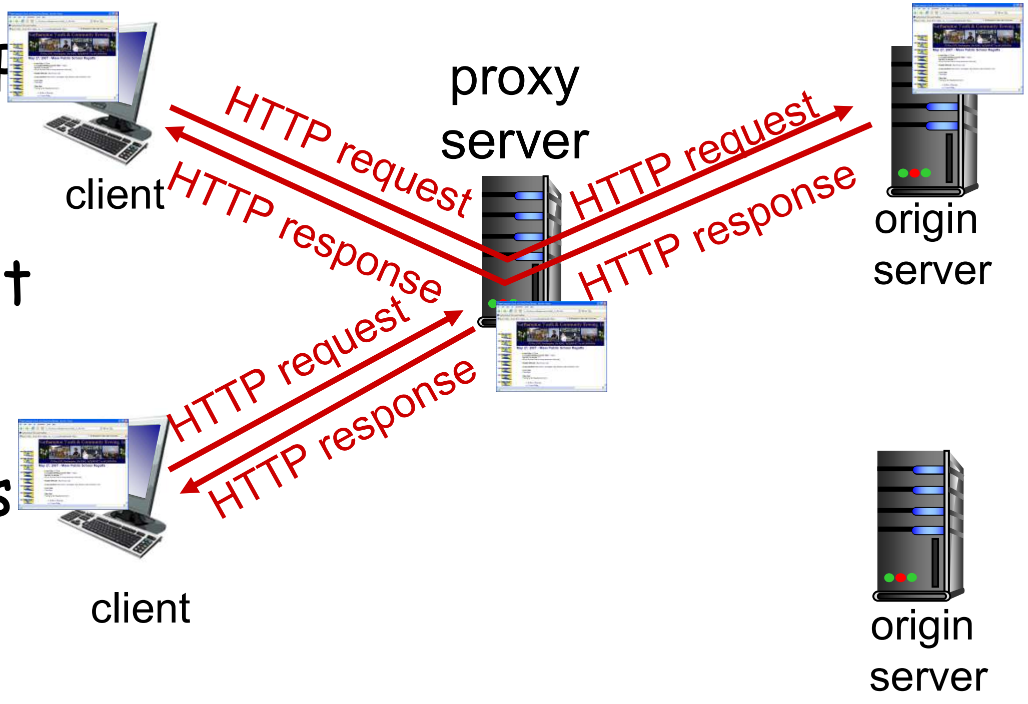
cookies and privacy:

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

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
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)

why Web caching?

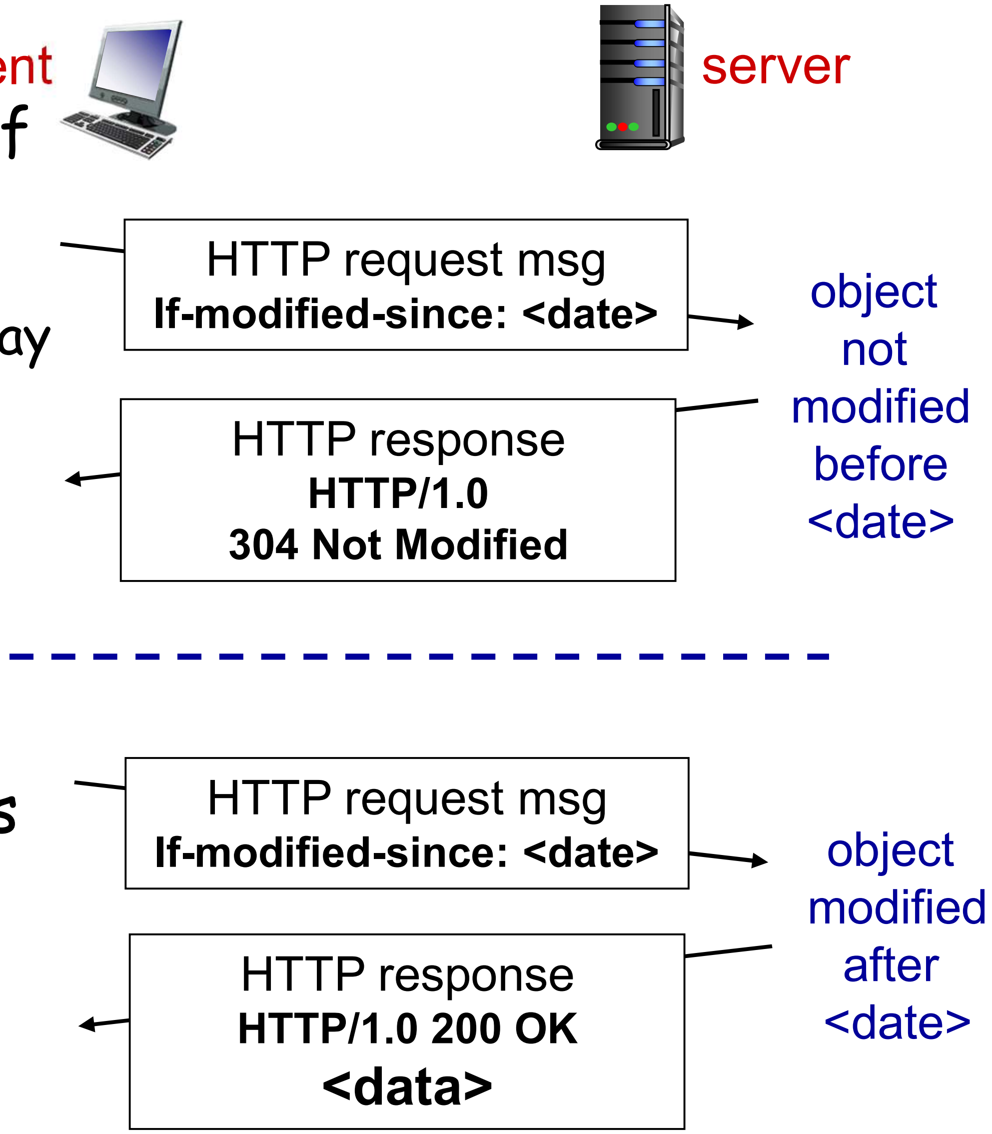
- reduce response time for client request
- reduce traffic on an institution's access link

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`



client

server

HTTP request msg
If-modified-since: <date>

HTTP response
HTTP/1.0
304 Not Modified

object not modified before <date>

HTTP request msg
If-modified-since: <date>

HTTP response
HTTP/1.0 200 OK
<data>

object modified after <date>

Conditional GET

- caching can reduce user-perceived response times,
  - new problem—
    - The object housed in the Web server may have been modified since the copy was cached at the client.

- HTTP has a mechanism that allows a cache to verify that its objects are up to date. This mechanism is called the **conditional GET**.

- An HTTP request message is a so-called conditional GET message if
- (1) the request message uses the **GET method**
- (2) the request message includes an **If-Modified-Since:** header line.

Example:

On the behalf of a requesting browser, a proxy cache sends a request message to a Web server:

- GET /fruit/kiwi.gif HTTP/1.1
- Host: www.exotiquecuisine.com

Web server sends a response message with the requested object to the cache:

- HTTP/1.1 200 OK
- Date: Sat, 8 Oct 2011 15:39:29
- Server: Apache/1.3.0 (Unix)
- Last-Modified: Wed, 7 Sep 2011 09:23:24
- Content-Type: image/gif
- (data data data data data ...)

The cache forwards the object to the requesting browser but also caches the object locally.

Cont..

1 week later:

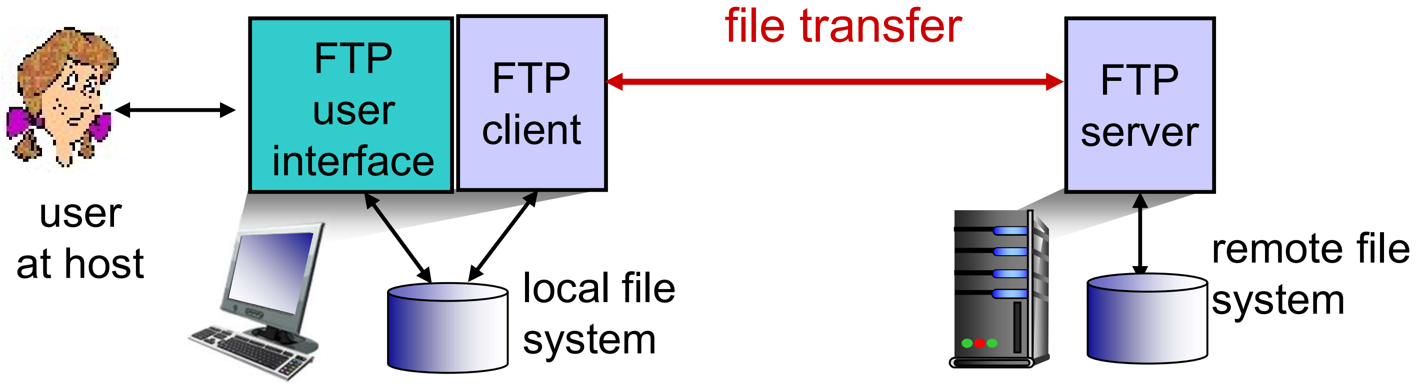
The cache performs an up-to-date check by issuing a conditional GET. Specifically, the cache sends:

- GET /fruit/kiwi.gif HTTP/1.1
- Host: www.exotiquecuisine.com
- If-modified-since: Wed, 7 Sep 2011 09:23:24

Web server sends a response message to the cache:

- HTTP/1.1 304 Not Modified
- Date: Sat, 15 Oct 2011 15:39:29
- Server: Apache/1.3.0 (Unix)
- (empty entity body)

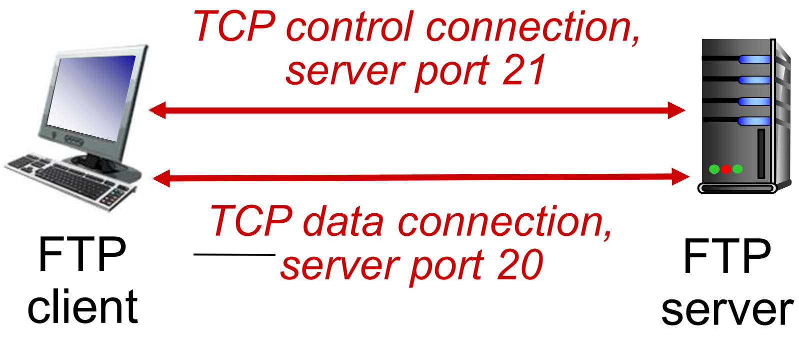
1.3 FTP: the file transfer protocol



- 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
- client authorized over control connection
- client browses remote directory, sends commands over control connection
- after transferring one file, server closes data connection



- server opens another TCP data connection to transfer another file
- 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`**

Difference between HTTP & FTP

1. FTP uses two parallel TCP connections to transfer a file, a control connection and a data connection.	1. HTTP sends request and response header lines into the same TCP connection that carries the transferred file itself.
2. FTP is said to send its control information out-of-band.	2. HTTP is said to send its control information in-band.
3. Port number: 20 and 21	3. Port number: 80
4. FTP server maintains state about the user.	4. HTTP stateless protocol

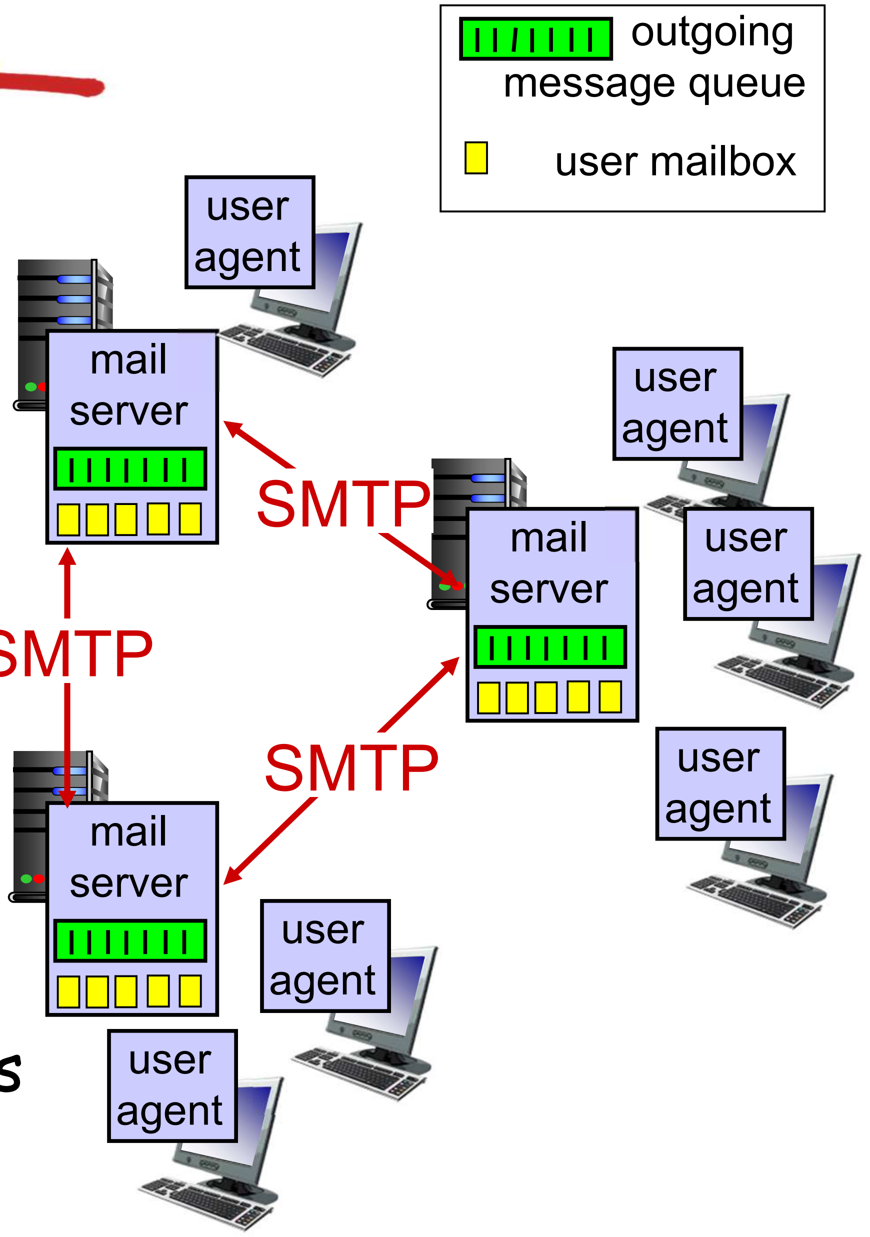
1.4 Electronic mail

Three major components:

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

User Agent

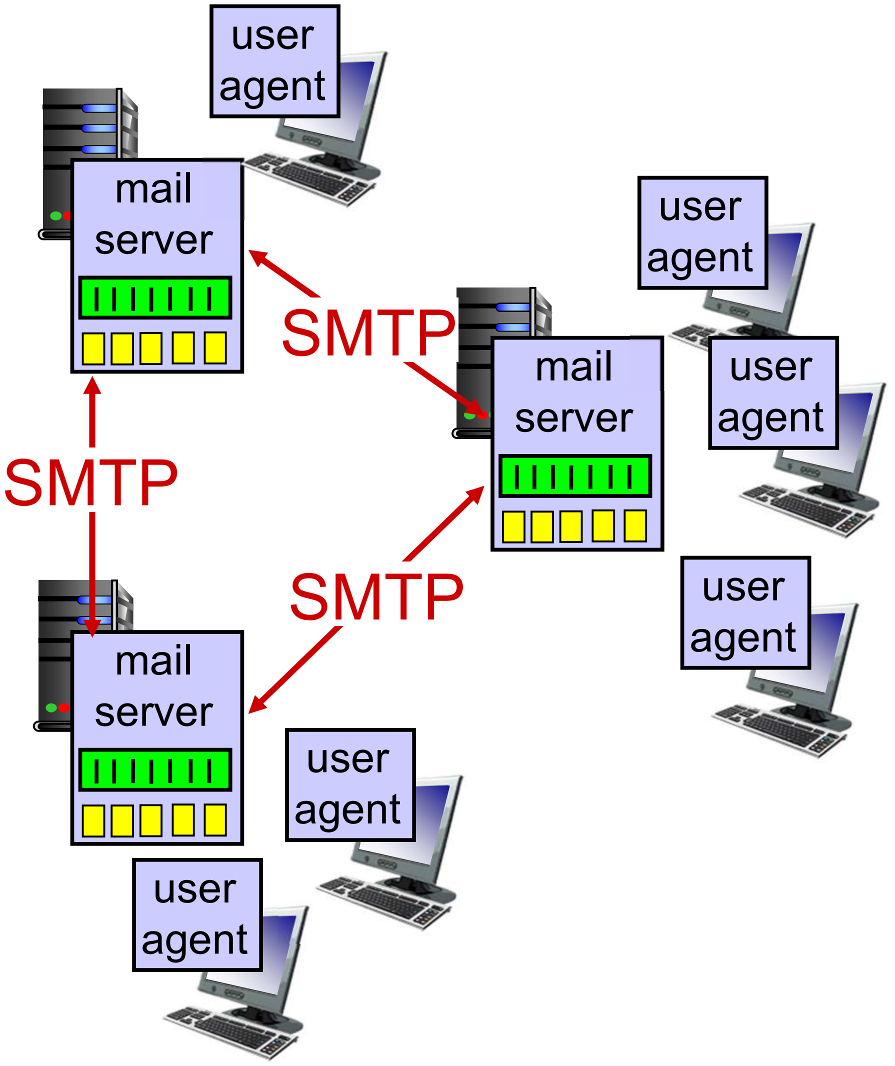
- "mail reader"
- composing, editing, reading mail messages
- outgoing, incoming messages stored on server



Electronic mail: mail servers

mail servers:

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



Electronic Mail: SMTP

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

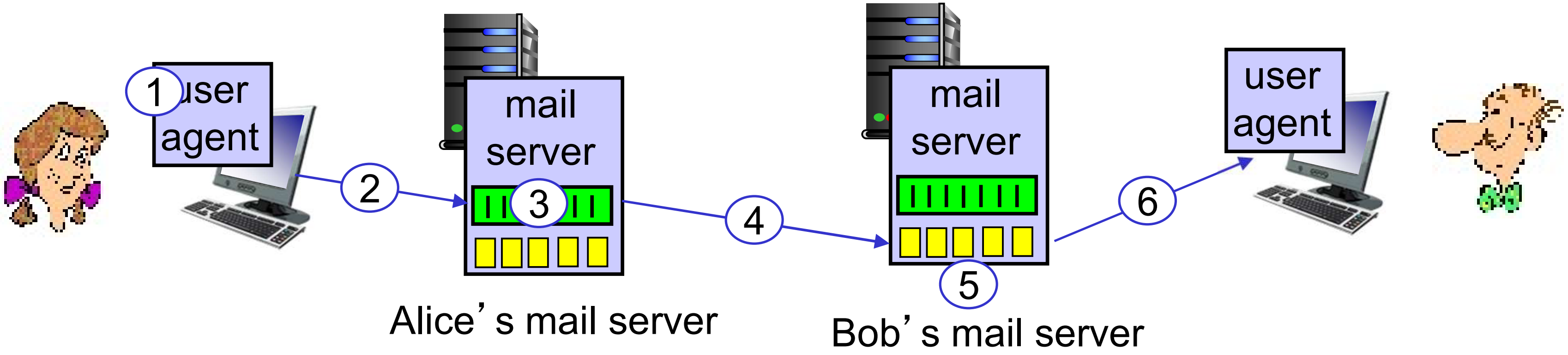
Here be secret messages!!1

Binary

```
01001000 01100101 01110010 01100101 00100000 01100010 01100101
00100000 01110011 01100101 01100011 01110010 01100101 01110100
00100000 01101101 01100101 01110011 01110011 01100001 01100111
01100101 01110011 00100001 00100001 00110001
```

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

- SMTP uses persistent connections
- **message transfer agent**

comparison with HTTP:

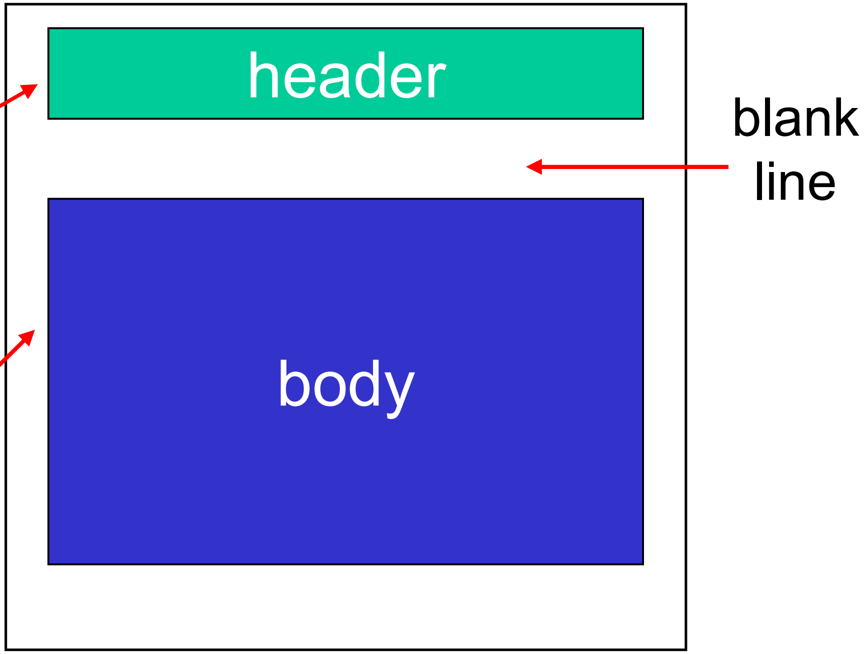
1. HTTP: pull protocol	1. SMTP: push protocol
2. No restrictions	2. SMTP requires message (header & body) to be in 7-bit ASCII
3. each object encapsulated in its own response message.	3. Internet mail places all of the message's objects into one message.
4. Port no 80	4. Port no 25

Mail message format

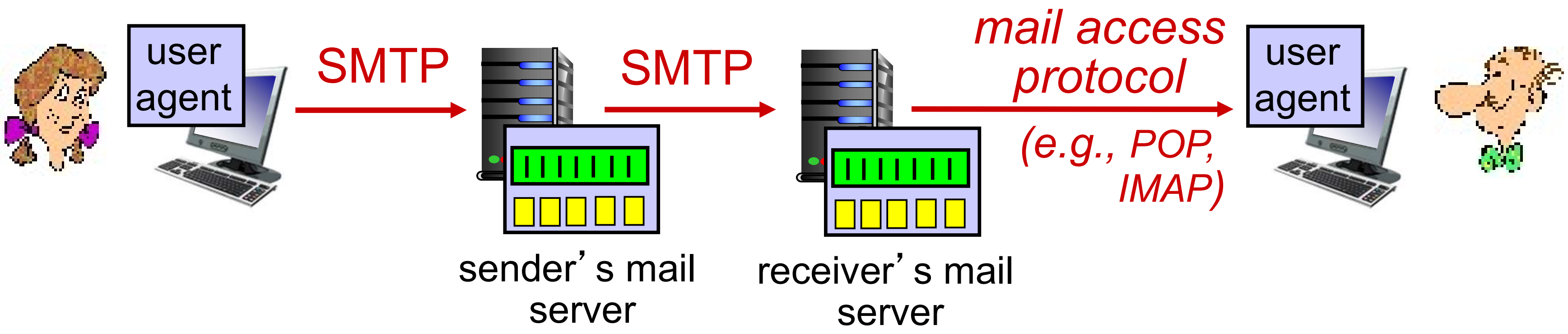
SMTP: protocol for exchanging email msgs

standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:
- Body: the "message"
  - ASCII characters only



Mail access protocols



- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
  - POP: Post Office Protocol: authorization, download
  - IMAP: Internet Mail Access Protocol: more features, including manipulation of stored msgs on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

POP3 protocol

- POP3 is an extremely simple mail access protocol.
- which is short and quite readable.
- functionality is rather limited.
- POP3 begins when the user agent (the client) opens a TCP connection to the mail server (the server) **on port 110.**

POP3 progresses through three phases:

1. authorization,
2. transaction
3. update.

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
```

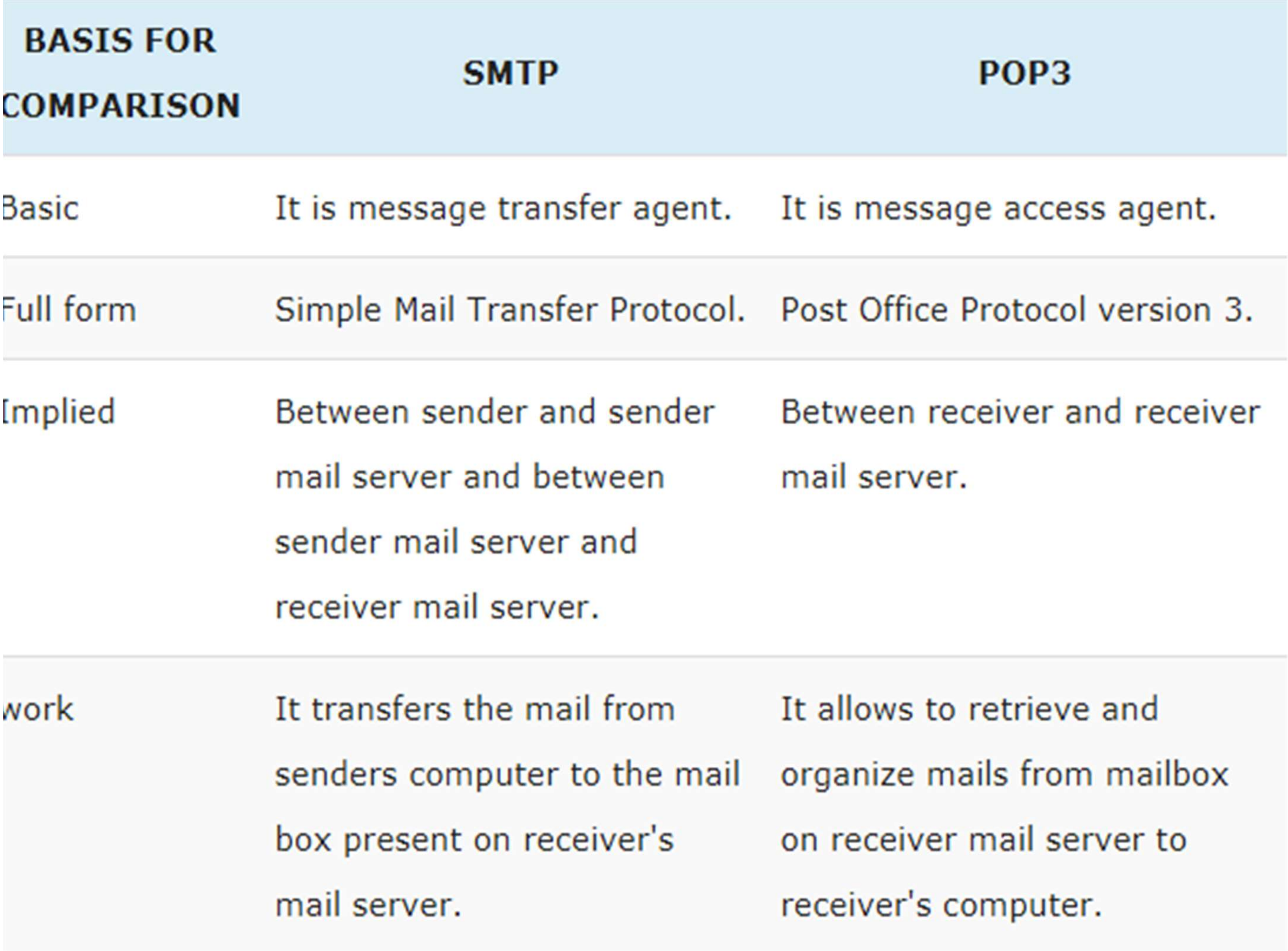
POP3 (more) and IMAP

more about POP3

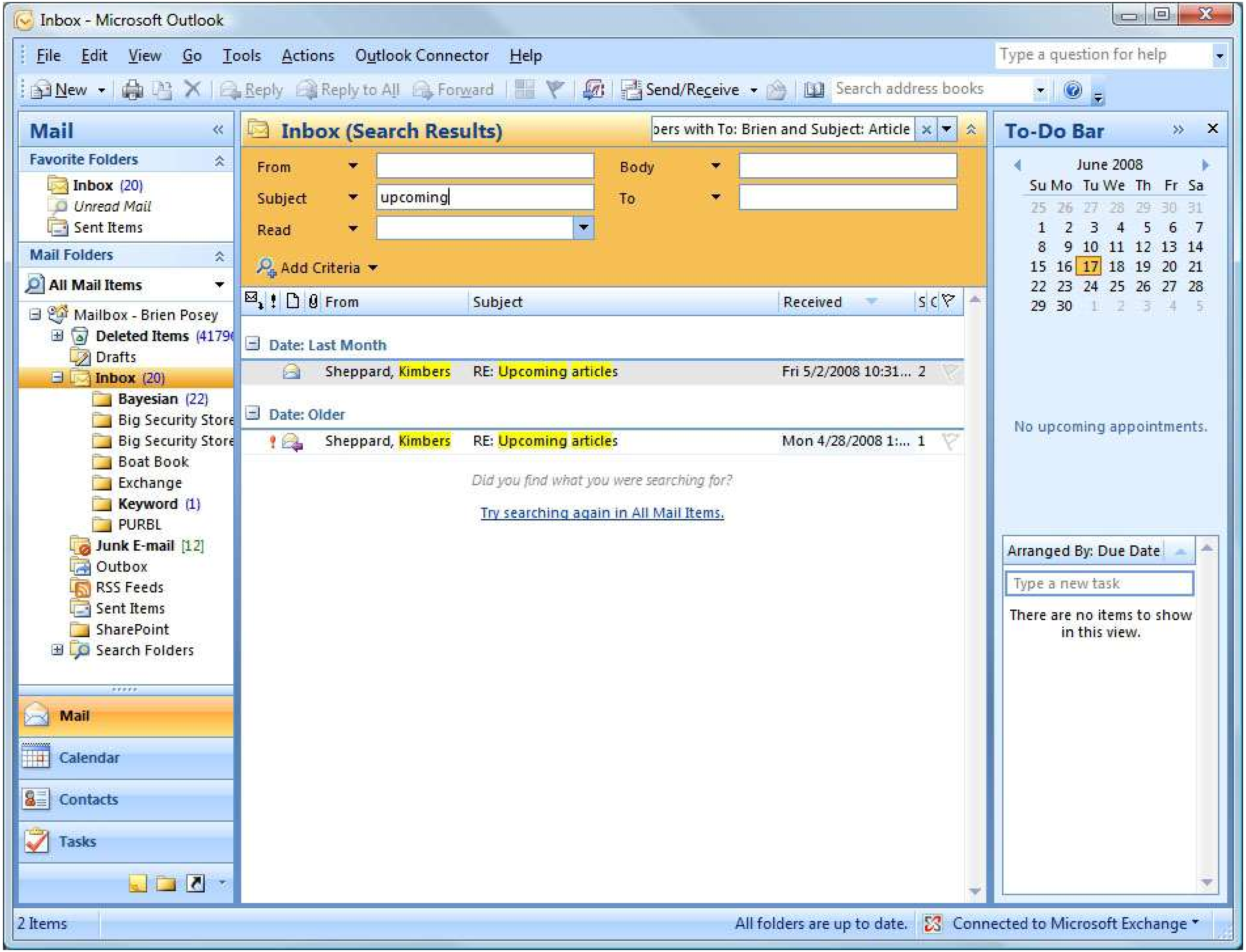
- previous example uses POP3 “download and delete” mode
    - Bob cannot re-read e-mail if he changes client
- POP3 “download-and-keep”: copies of messages on different clients
- POP3 is stateless across sessions

IMAP

- keeps all messages in one place: at server
- allows user to organize messages in folders
- Port no 143
- keeps user state across sessions:
    - names of folders and mappings between message IDs and folder name



BASIS FOR COMPARISON	SMTP	POP3
Basic	It is message transfer agent.	It is message access agent.
Full form	Simple Mail Transfer Protocol.	Post Office Protocol version 3.
Implied	Between sender and sender mail server and between sender mail server and receiver mail server.	Between receiver and receiver mail server.
Work	It transfers the mail from senders computer to the mail box present on receiver's mail server.	It allows to retrieve and organize mails from mailbox on receiver mail server to receiver's computer.



ɔers with To: Brien and Subject: Article

From:
Subject: upcoming
Read:
Body:
To:

From	Subject	Received	S
Date: Last Month			
Sheppard, Kimbers	RE: Upcoming articles	Fri 5/2/2008 10:31...	2
Date: Older			
Sheppard, Kimbers	RE: Upcoming articles	Mon 4/28/2008 1:...	1

Did you find what you were searching for?

Try searching again in All Mail Items.

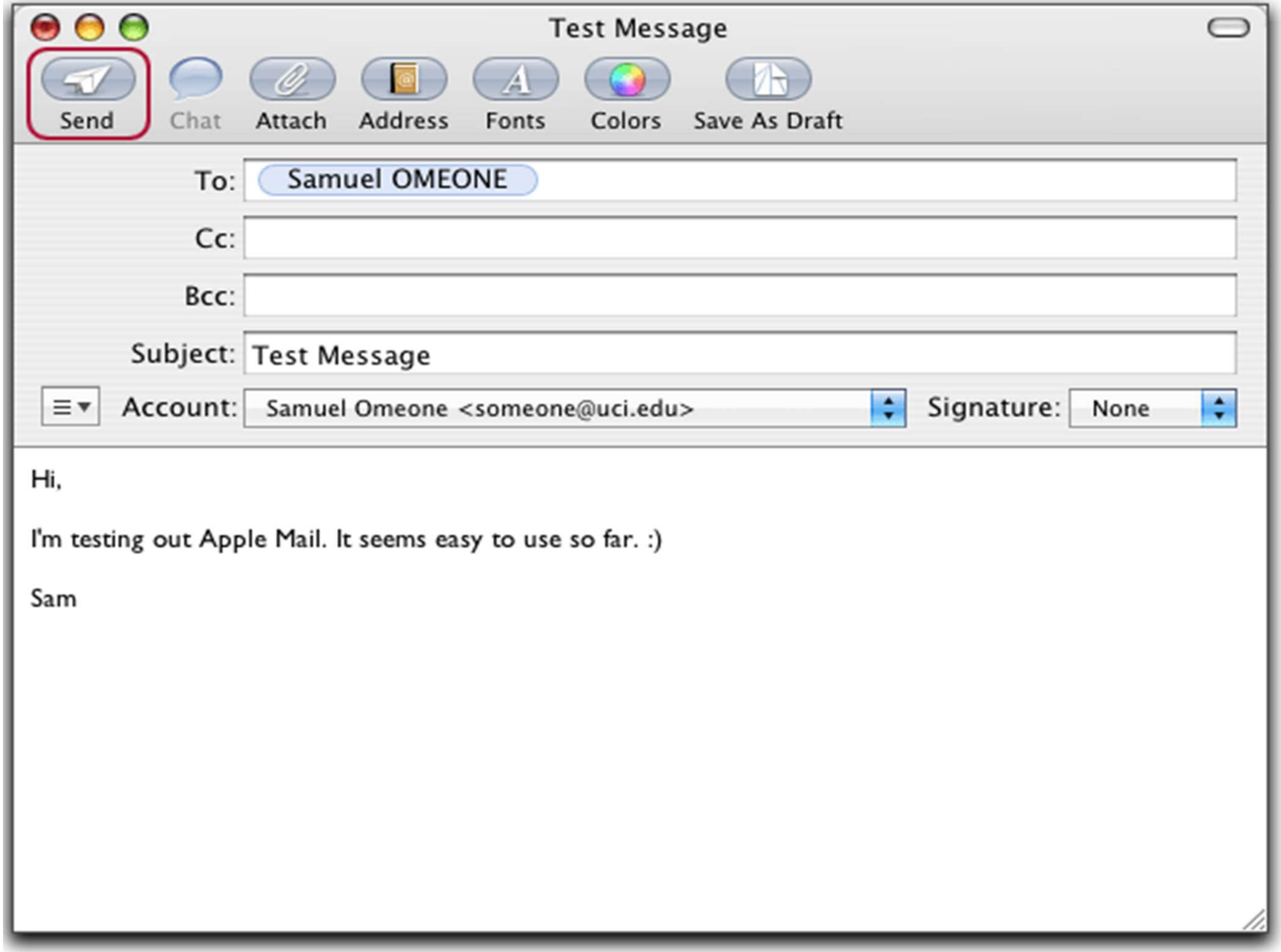
No upcoming appointments.

There are no items to show in this view.

2 Items

All folders are up to date.

Connected to Microsoft Exchange



To: Samuel OMEONE

Cc:

Bcc:

Subject: Test Message

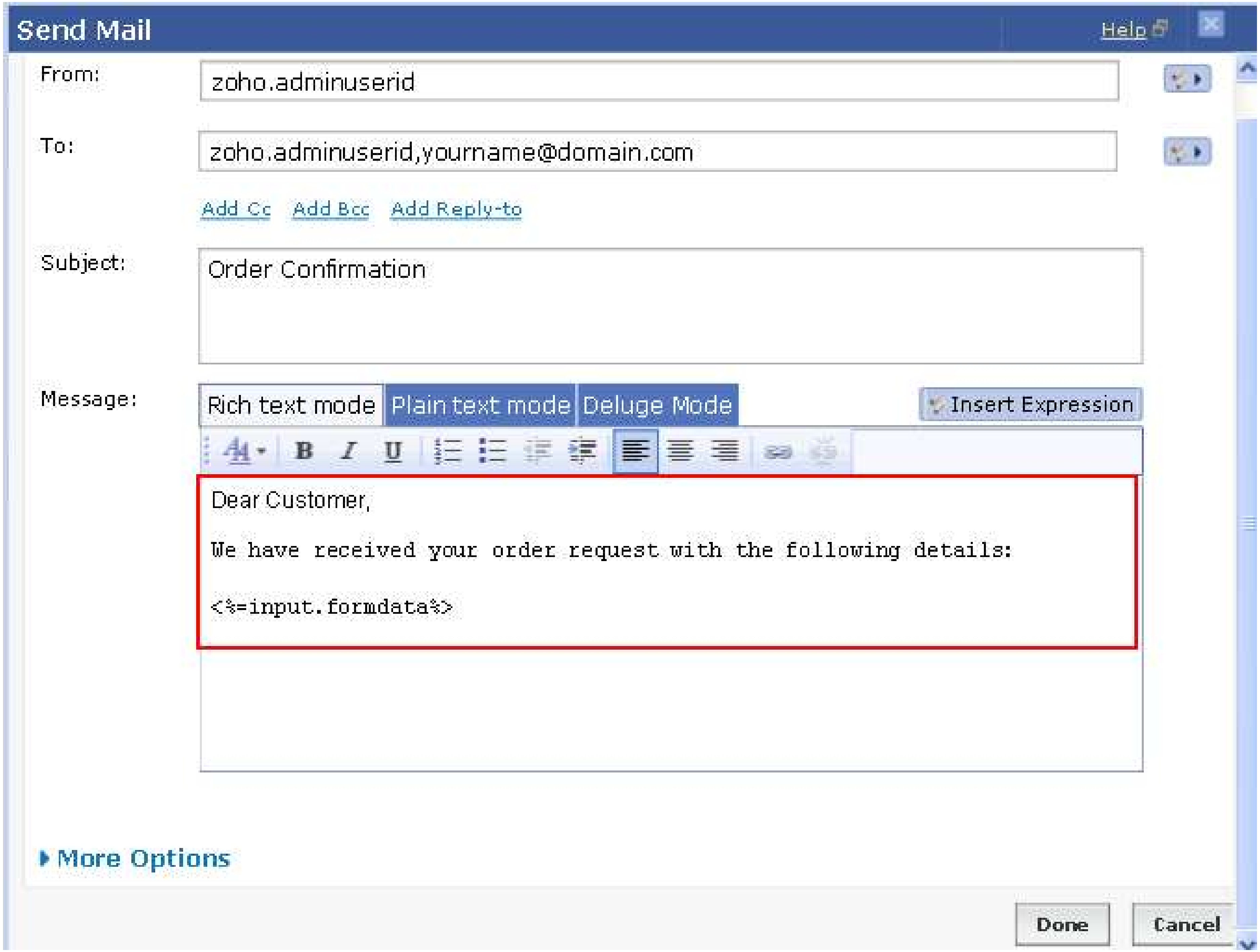
Account: Samuel Omeone <someone@uci.edu>

Signature: None

Hi,

I'm testing out Apple Mail. It seems easy to use so far. :)

Sam



From: zoho.adminuserid

To: zoho.adminuserid,yourname@domain.com

Subject: Order Confirmation

Message:

Dear Customer,

We have received your order request with the following details:

```
<%=input.formdata%>
```

1 .5 DNS: domain name system

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

  - Identified by the host name and IP address (121.7.106.83)

- 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 ?

- The DNS is
  - (1) a distributed database implemented in a hierarchy of **DNS servers**, and
  - (2) an application-layer protocol that allows hosts to query the distributed database.
- The DNS protocol runs over UDP
- DNS is commonly employed by other application-layer protocols
  - —including HTTP, SMTP, and FTP
  - to translate user-supplied hostnames to IP addresses
  - port 53.

1. The user machine runs the client side of the DNS application.

2. The browser extracts the hostname, ***www.someschool.edu***, from the URL and passes the hostname to the client side of the DNS application.

3. The DNS client sends a query containing the hostname to a DNS server.

4. The DNS client eventually receives a reply, which includes the IP address for the hostname.

5. Once the browser receives the IP address from DNS, it can initiate a TCP connection to the HTTP server process located at port 80 at that IP address.

DNS: services, structure

DNS services

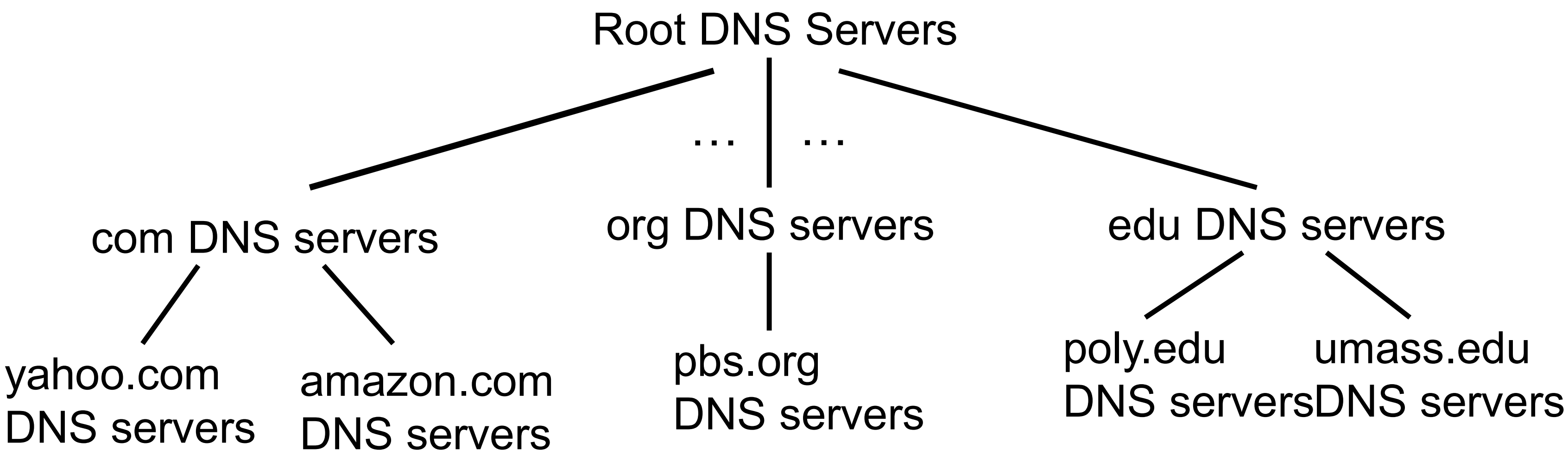
- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing:
  - highly desirable that e-mail addresses be mnemonic
- load distribution
  - replicated Web servers

why not centralize DNS?

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

A: *doesn't scale!*

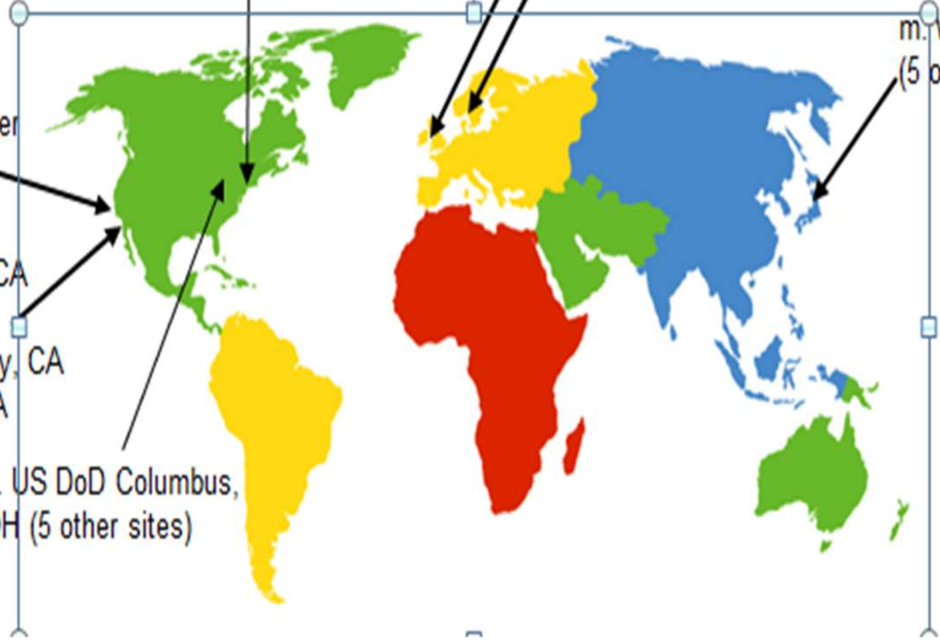
DNS: a distributed, hierarchical database



- **Classes of DNS server**
  - Root DNS server
  - Top-level domain DNS server
  - Authoritative DNS server

DNS: root name servers

- In the Internet there are 13 root DNS servers (labeled A through M), most of which are located in North America.



c. Cogent, Herndon, VA (5 other sites)
d. U Maryland College Park, MD
h. ARL Aberdeen, MD
j. Verisign, Dulles VA (69 other sites)

k. RIPE London (17 other sites)

i. Netnod, Stockholm (37 other sites)

e. NASA Mt View, CA
f. Internet Software C. Palo Alto, CA (and 48 other sites)

m. WIDE Tokyo (5 other sites)

a. Verisign, Los Angeles CA (5 other sites)
b. USC-ISI Marina del Rey, CA
l. ICANN Los Angeles, CA (41 other sites)

g. US DoD Columbus, OH (5 other sites)

13 root name "servers" worldwide

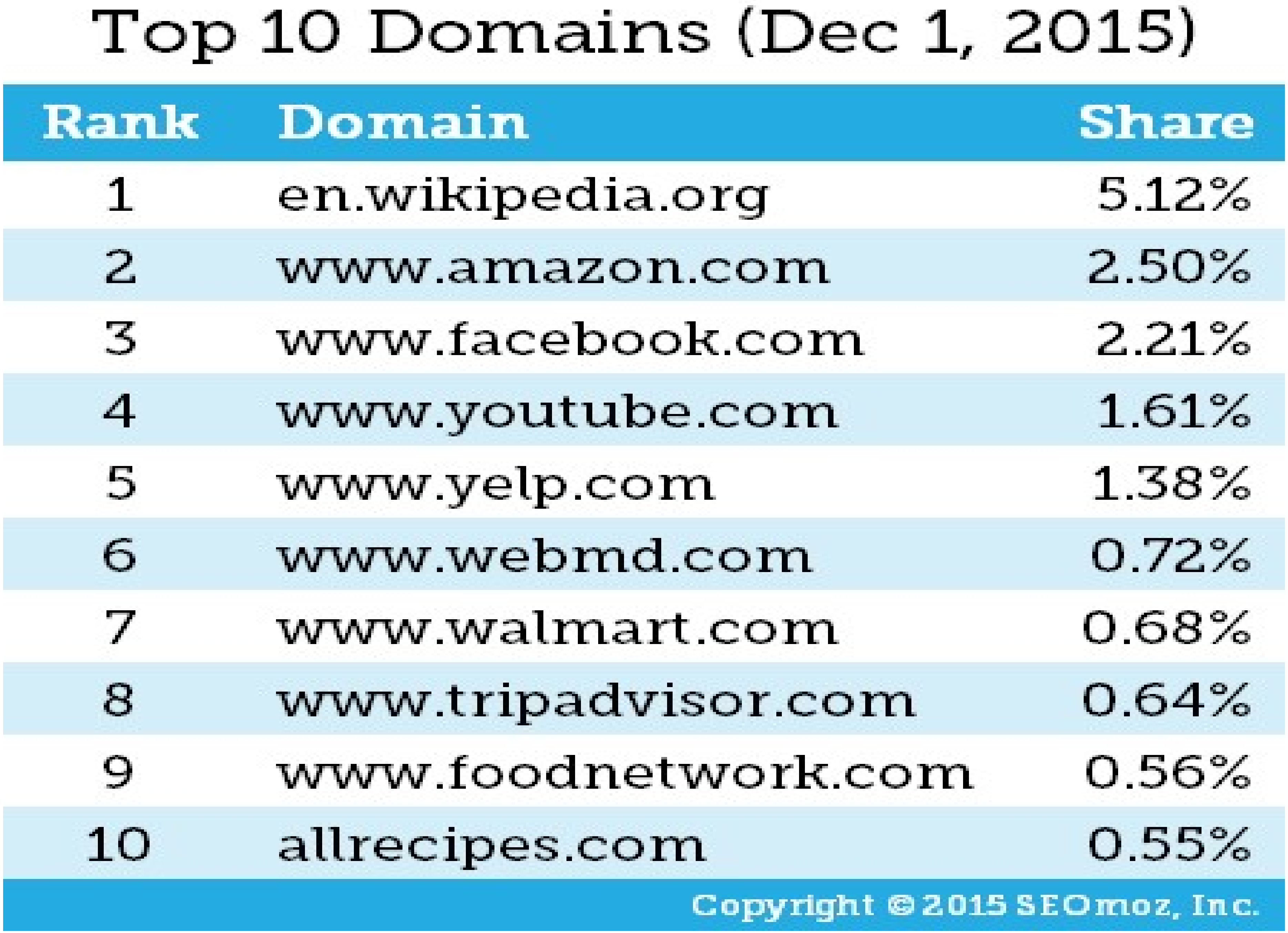
TLD, authoritative servers

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
- Edu cause 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



Top 10 Domains (Dec 1, 2015)

Rank	Domain	Share
1	en.wikipedia.org	5.12%
2	www.amazon.com	2.50%
3	www.facebook.com	2.21%
4	www.youtube.com	1.61%
5	www.yelp.com	1.38%
6	www.webmd.com	0.72%
7	www.walmart.com	0.68%
8	www.tripadvisor.com	0.64%
9	www.foodnetwork.com	0.56%
10	allrecipes.com	0.55%

ZONE	DEFINITION	FOR USE BY
.com	Commercial	Businesses
.edu	Education	Universities
.gov	Government	U.S. federal government agencies
.int	International	Organizations established by international treaties
.mil	Military	U.S. military
.net	Network	Network providers, administrator computers, network node computers
.org	Organization	Non-profit and miscellaneous organizations

Local DNS server

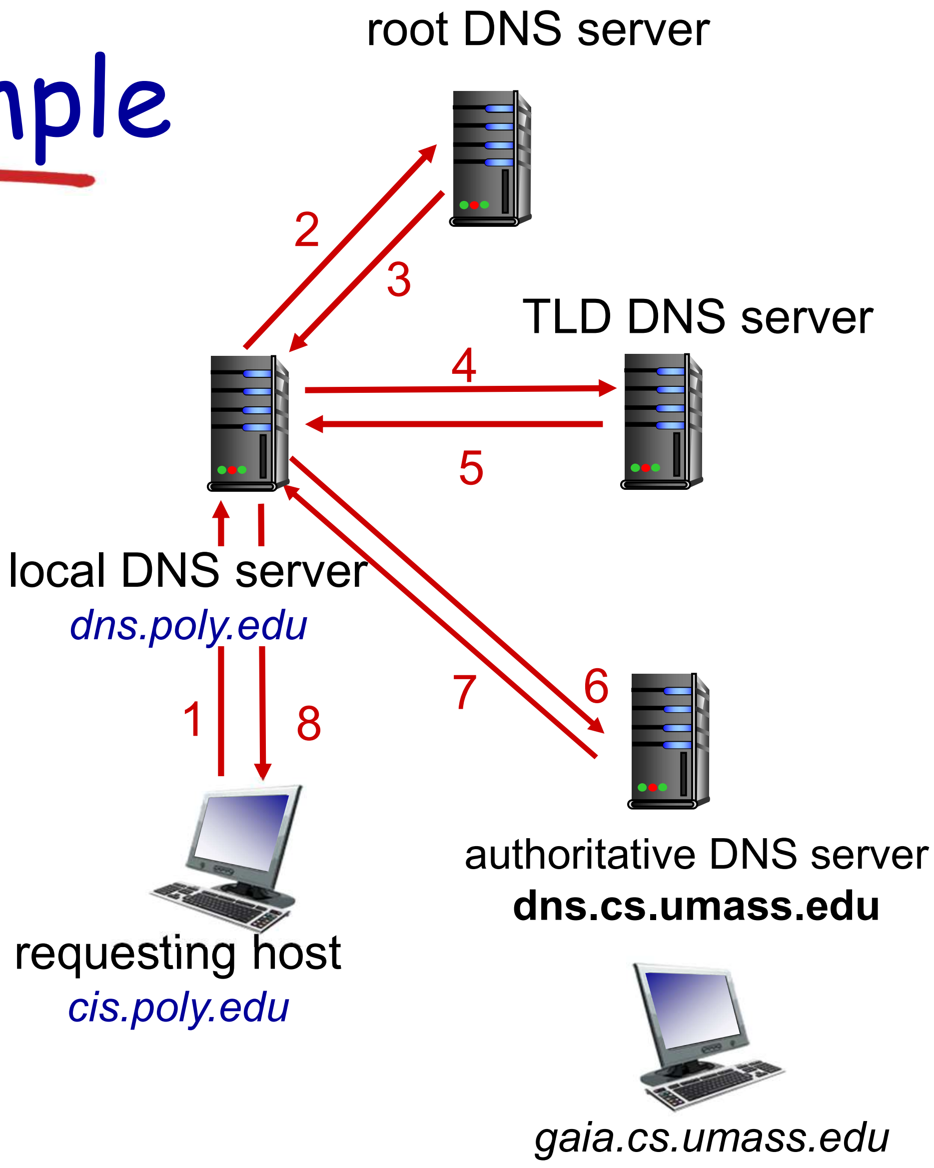
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

DNS name resolution example

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

Iterative query:

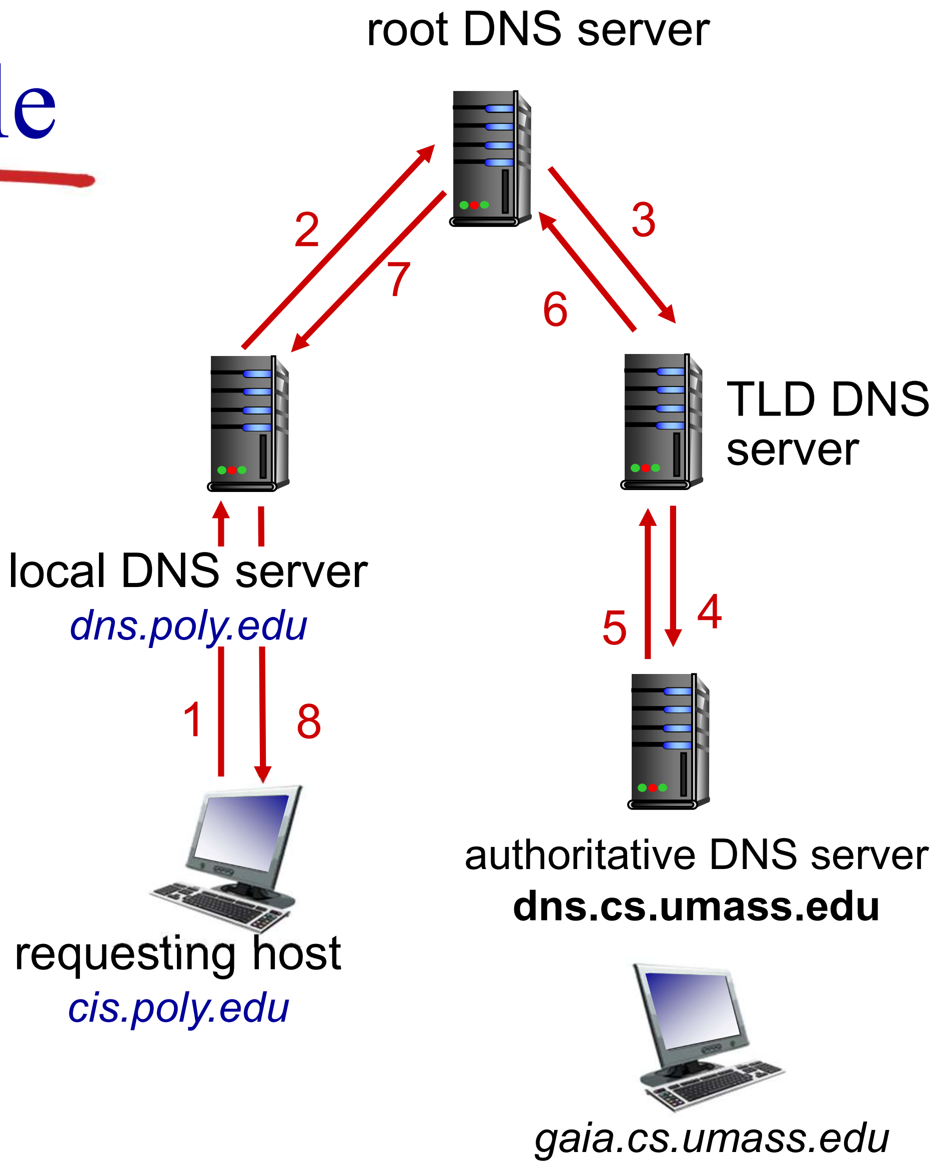
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example

recursive query:

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



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

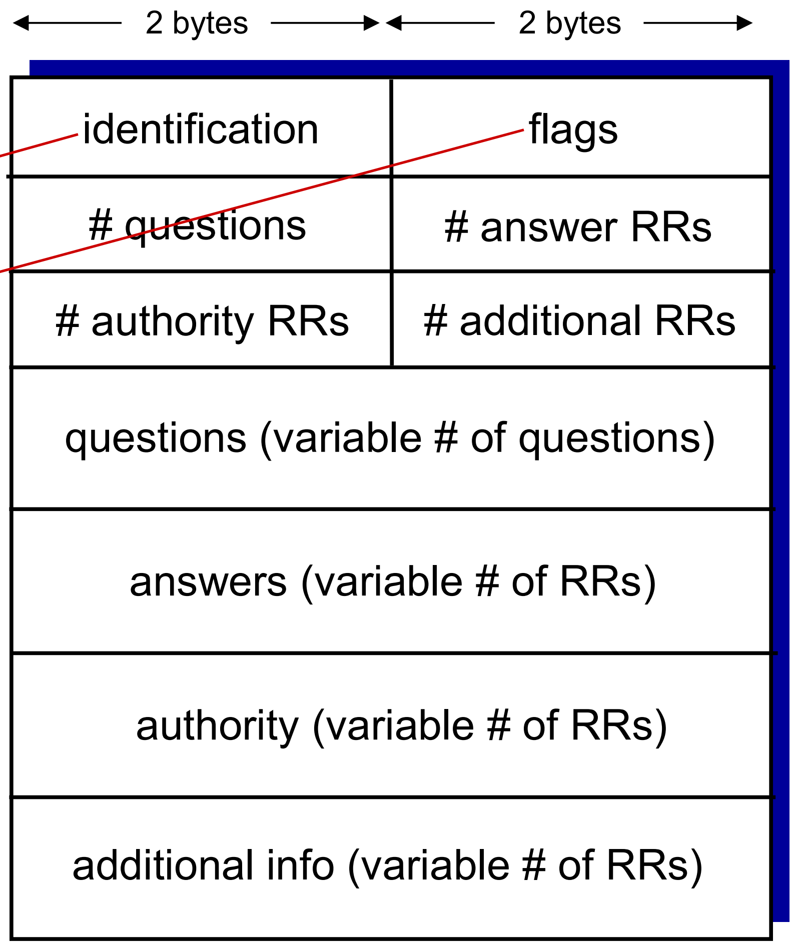
- **name:** is alias name for some "canonical" (the real) name
- **value** is canonical name

DNS protocol, messages

- *query* and *reply* messages, both with same *message format*

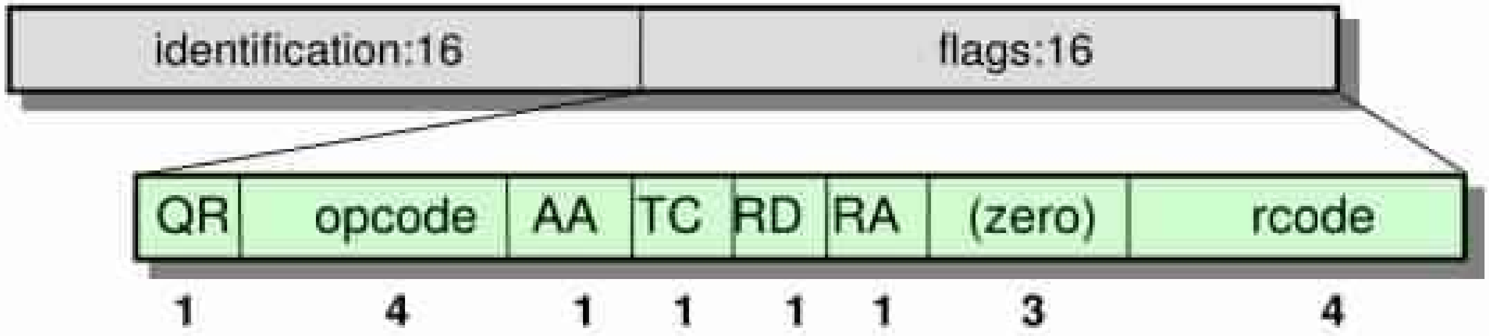
msg header

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



2 bytes	2 bytes
identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

DNS message format: detail

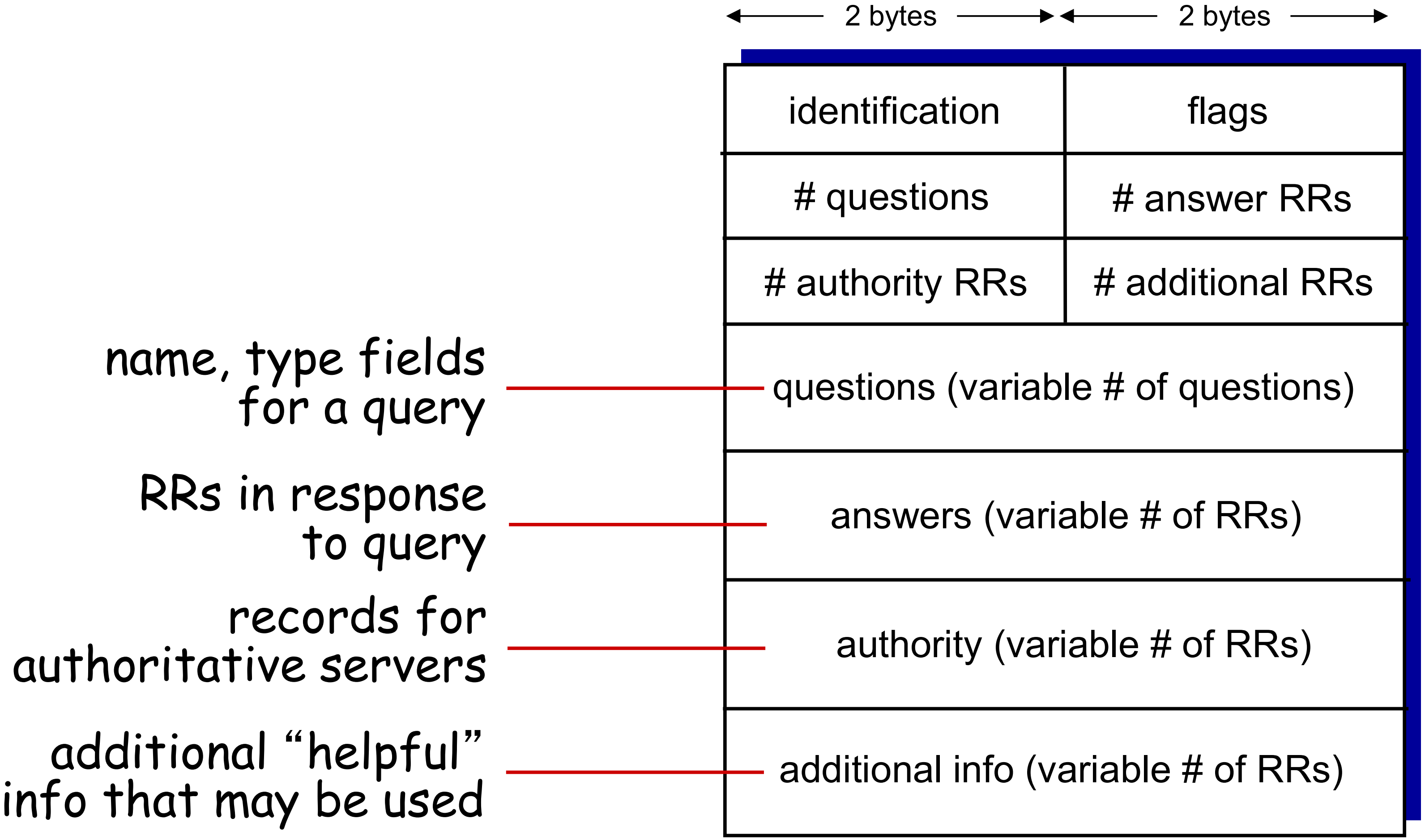


identification:16	flags:16

QR	opcode	AA	TC	RD	RA	(zero)	rcode
1	4	1	1	1	1	3	4

QR	**0= query, 1= response**
opcode	**0= standard query, 1=inverse query, 2=server status request**
AA	**0= authoritatived answer,1 = non authoritatived answer**
TC	**1= truncated. using UDP, reply was>512 bytes, return only 512 bytes**
RD	**1= recursive desired, 0= iterative**
RA	**1= recursion available (server support recursion)**
rcode	**return code : 0=no error, 3=name error**

DNS protocol, messages



Inserting Records into the DNS Database

- new start up company called Network Utopia
- register the domain name networkutopia.com
- need to provide the registrar with the names and IP addresses of your primary and secondary authoritative DNS servers

- Suppose the names and IP addresses are
- dns1.networkutopia.com, dns2.networkutopia.com,
- 212.212.212.1, and 212.212.212.2

- the registrar would insert the following two resource records into the DNS system:
- (networkutopia.com,dns1.networkutopia.com, NS)
- (dns1.networkutopia.com, 212.212.212.1, A)

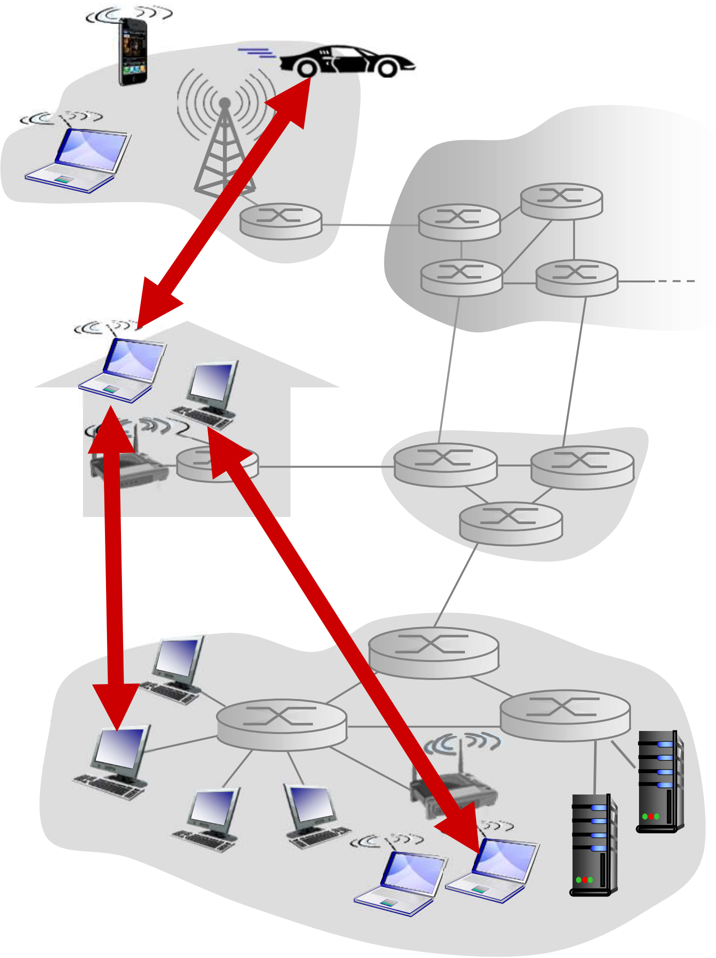
1.6 P2P applications

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

- *Applications:*
  - BitTorrent
  - DHT

examples:

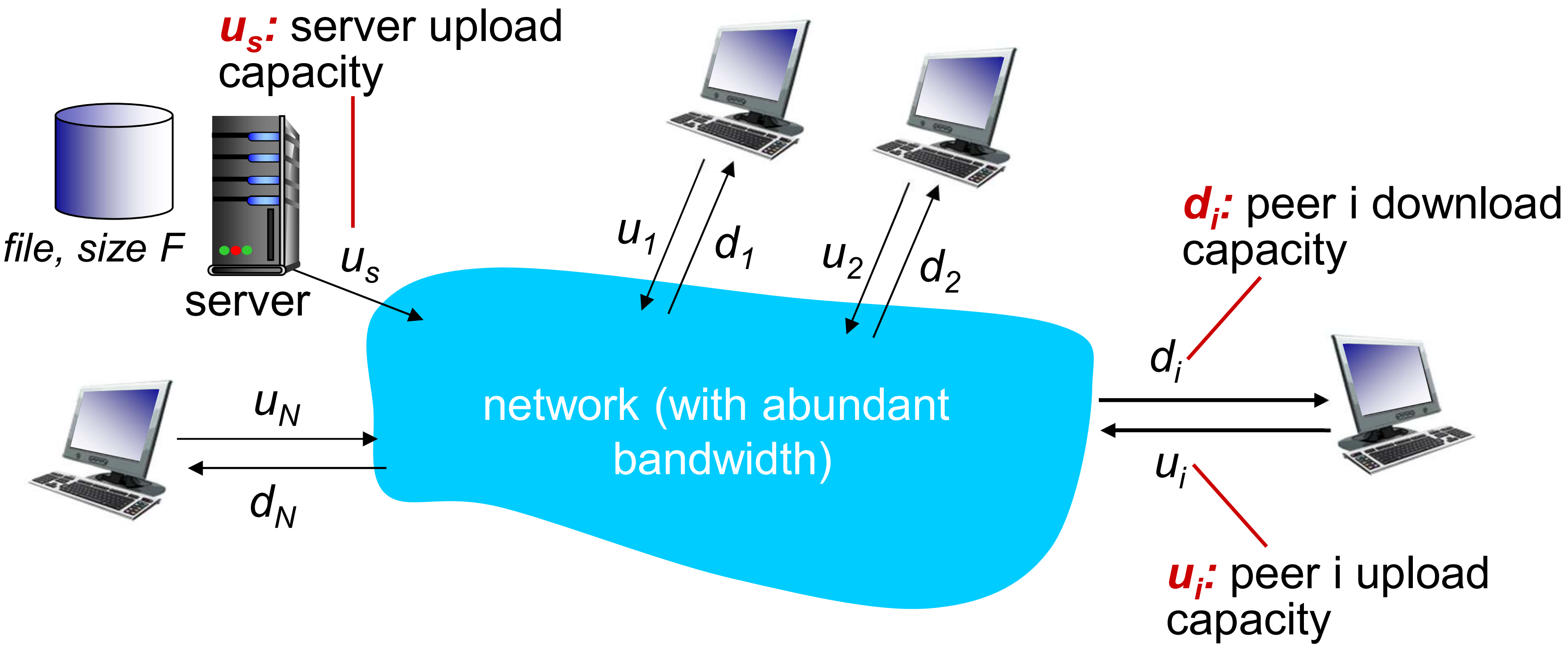
  - file distribution (BitTorrent)
  - Streaming (KanKan)
  - VoIP (Skype)



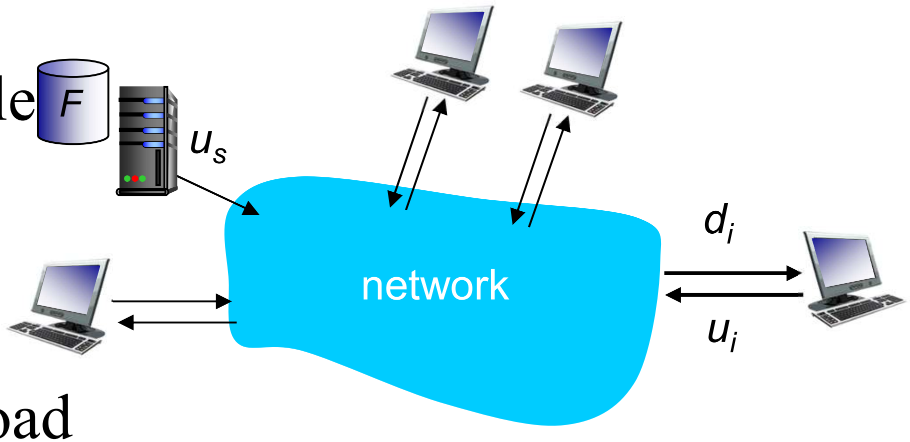
File distribution: client-server vs P2P

Question: how much time to distribute file (size *F*) from one server to *N* *peers*?

- peer upload/download capacity is limited resource



File distribution time: client-server



- *server transmission:* must sequentially send (upload) *N* file copies:
  - time to send one copy: F/u_s
  - time to send N copies: NF/u_s
- *client:* each client must download file copy
  - d_{min} = min client download rate
  - min client download time: F/d_{min}

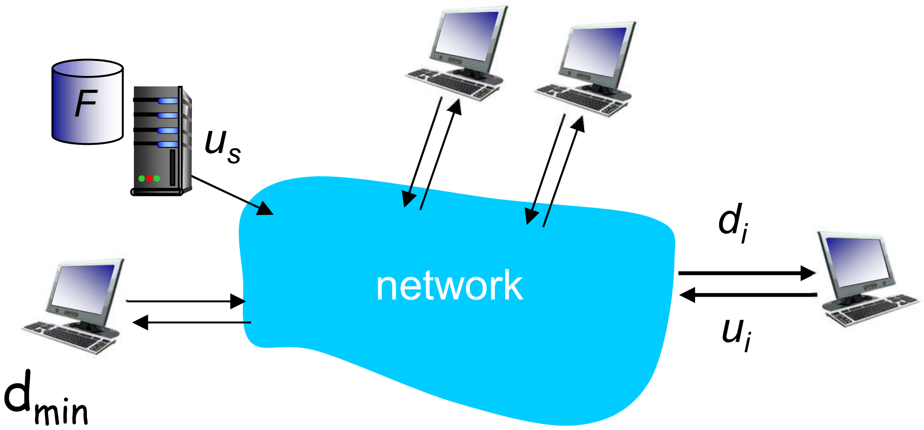
time to distribute F to N clients using client-server approach

$$D_{c-s} \geq max\{NF/u_{s,},F/d_{min}\}$$

increases linearly in N

File distribution time: P2P

- *server transmission:* must upload at least one copy
  - time to send one copy: F/u_s
- *client:* each client must download file copy
  - min client download time: F/d_{min}



- *clients:* as aggregate must download *NF* bits
  - max upload rate (limting max download rate) is $u_s + \Sigma u_i$

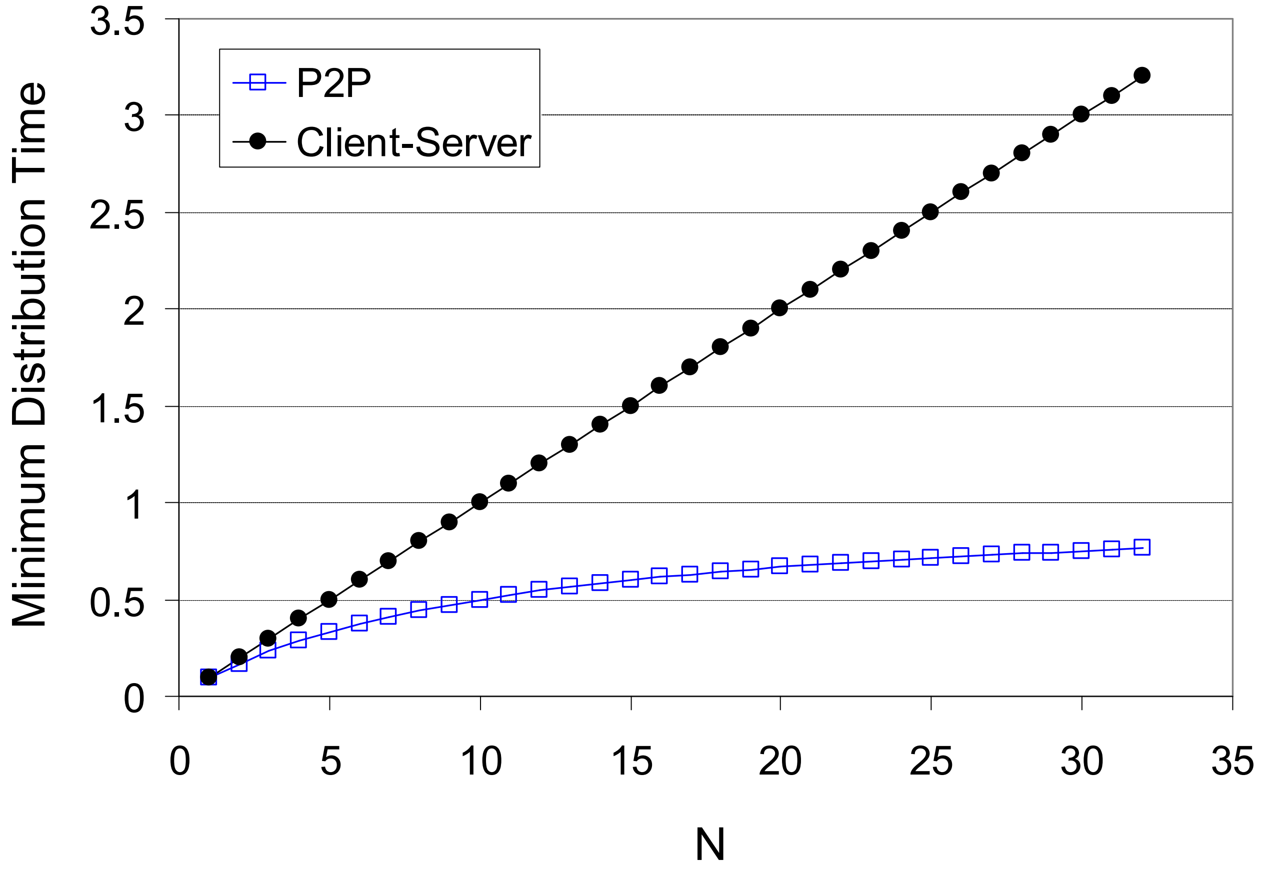
time to distribute F to N clients using P2P approach

$$D_{P2P} \geq max\{F/u_{s,},F/d_{min,},NF/(u_s + \Sigma u_i)\}$$

increases linearly in *N* …

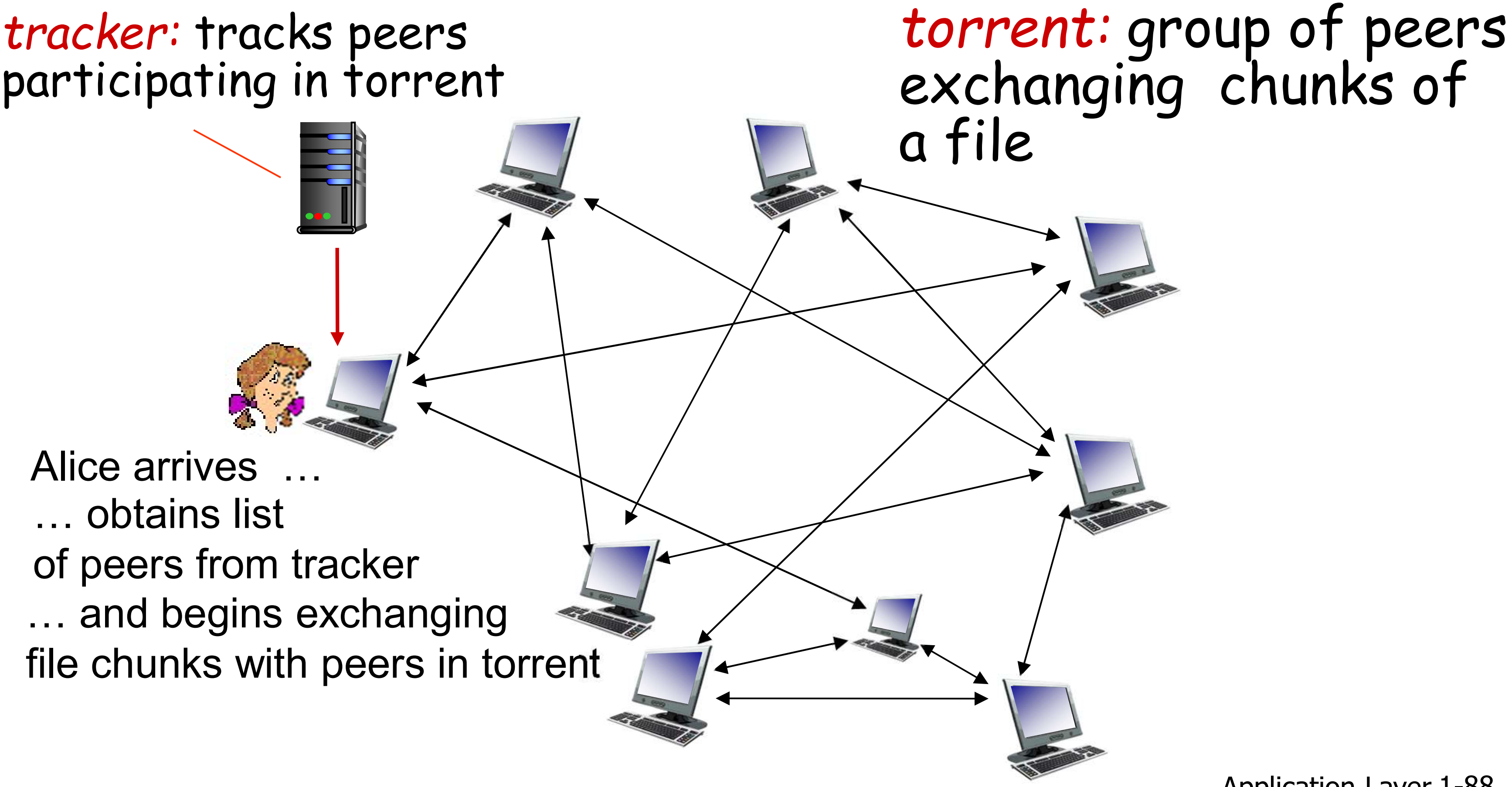
… but so does this, as each peer brings service capacity

Client-server vs. P2P: example



P2P file distribution: BitTorrent

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks

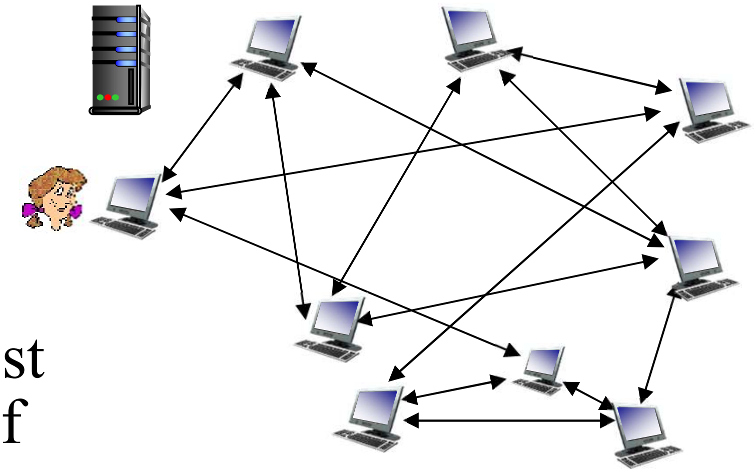


tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file

Alice arrives …
… obtains list
of peers from tracker
… and begins exchanging
file chunks with peers in torrent

P2P file distribution: BitTorrent



- peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers (“neighbors”)

- while downloading, peer uploads chunks to other peers
- peers may come and go
- once peer has entire file, it may (selfishly) **leave** or (altruistically) **remain** in torrent

BitTorrent: requesting, sending file chunks

requesting chunks:

- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, **rarest first**

sending chunks:

- Alice sends chunks to those four peers currently sending her chunks *at highest rate*

DHT: Simple Database

Simple database with(key, value) pairs:

- key: human name; value: social security #

Key	Value
John Washington	132-54-3570
Diana Louise Jones	761-55-3791
Xiaoming Liu	385-41-0902
Rakesh Gopal	441-89-1956
Linda Cohen	217-66-5609
…….	………
Lisa Kobayashi	177-23-0199

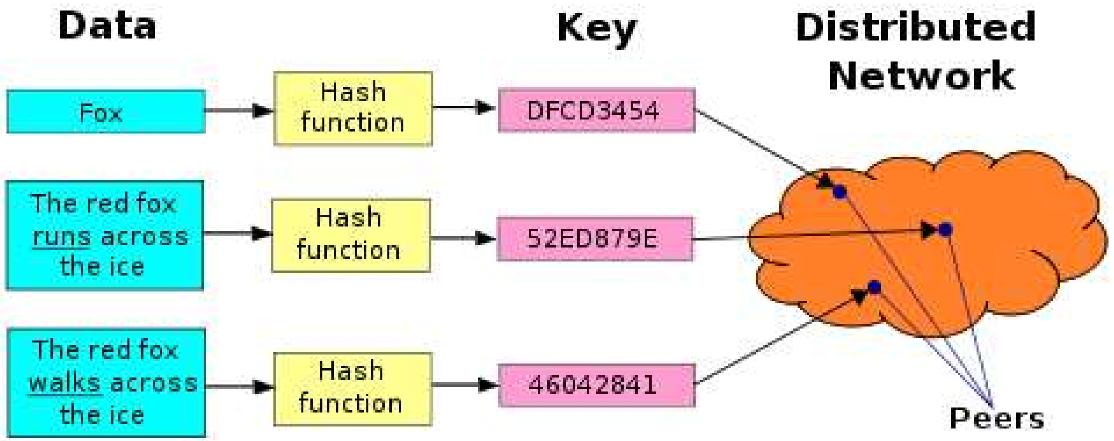
- key: movie title; value: IP address

Hash Table

- More convenient to store and search on numerical representation of key
- key = hash(original key)

Original Key	Key	Value
John Washington	8962458	132-54-3570
Diana Louise Jones	7800356	761-55-3791
Xiaoming Liu	1567109	385-41-0902
Rakesh Gopal	2360012	441-89-1956
Linda Cohen	5430938	217-66-5609
…….		………
Lisa Kobayashi	9290124	177-23-0199

Distributed Hash Table (DHT)



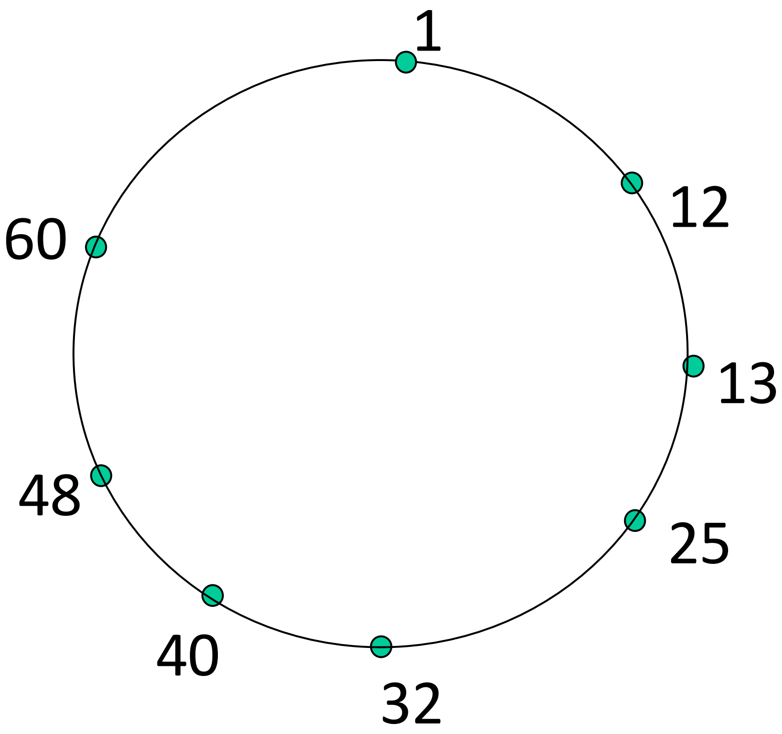
- Distribute (key, value) pairs over millions of peers
  - pairs are evenly distributed over peers
- Any peer can query database with a key
  - database returns value for the key
  - To resolve query, small number of messages exchanged among peers
- Each peer only knows about a small number of other peers
- Robust to peers coming and going

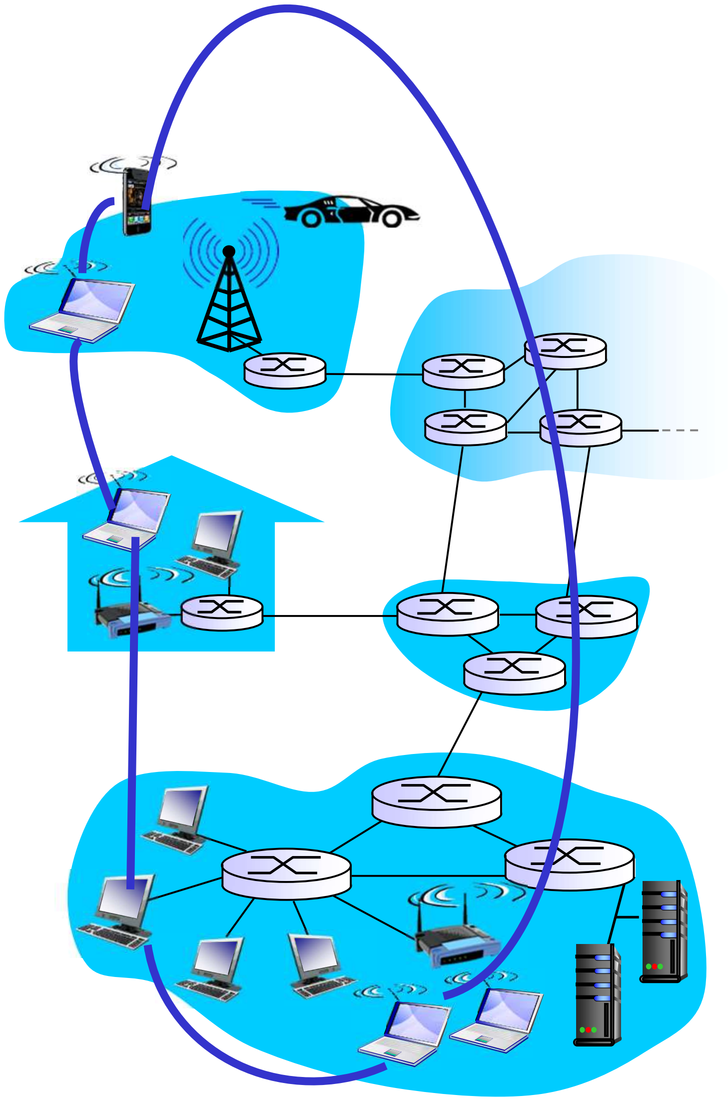
Assign key-value pairs to peers

- rule: assign key-value pair to the peer that has the *closest* ID.
- convention: closest is the *immediate successor* of the key.
- e.g., ID space {0,1,2,3,…,63}
- suppose 8 peers: 1,12,13,25,32,40,48,60
  - If key = 51, then assigned to peer 60
  - If key = 60, then assigned to peer 60
  - If key = 61, then assigned to peer 1

Circular DHT

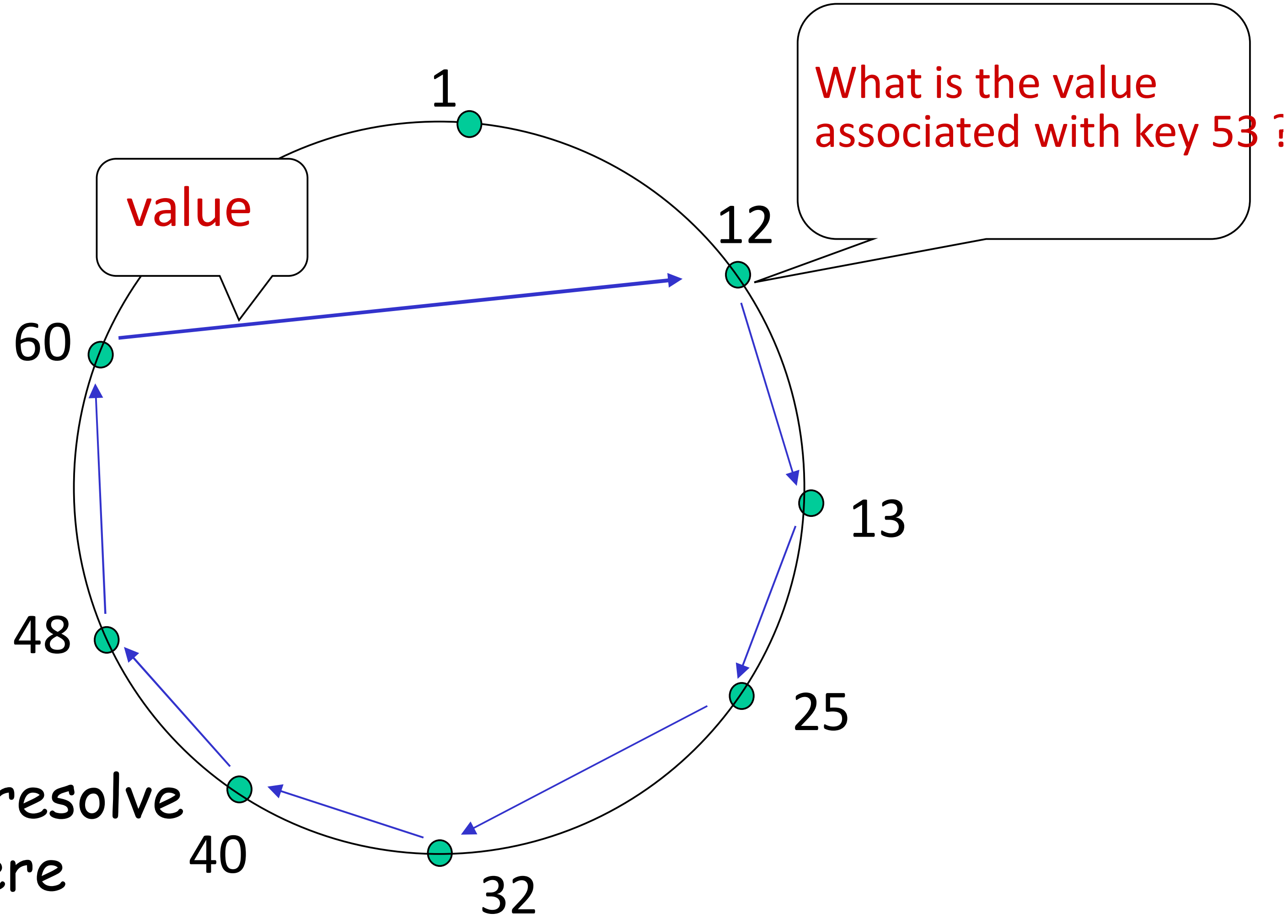
- each peer *only* aware of immediate successor and predecessor.





"overlay network"

Resolving a query

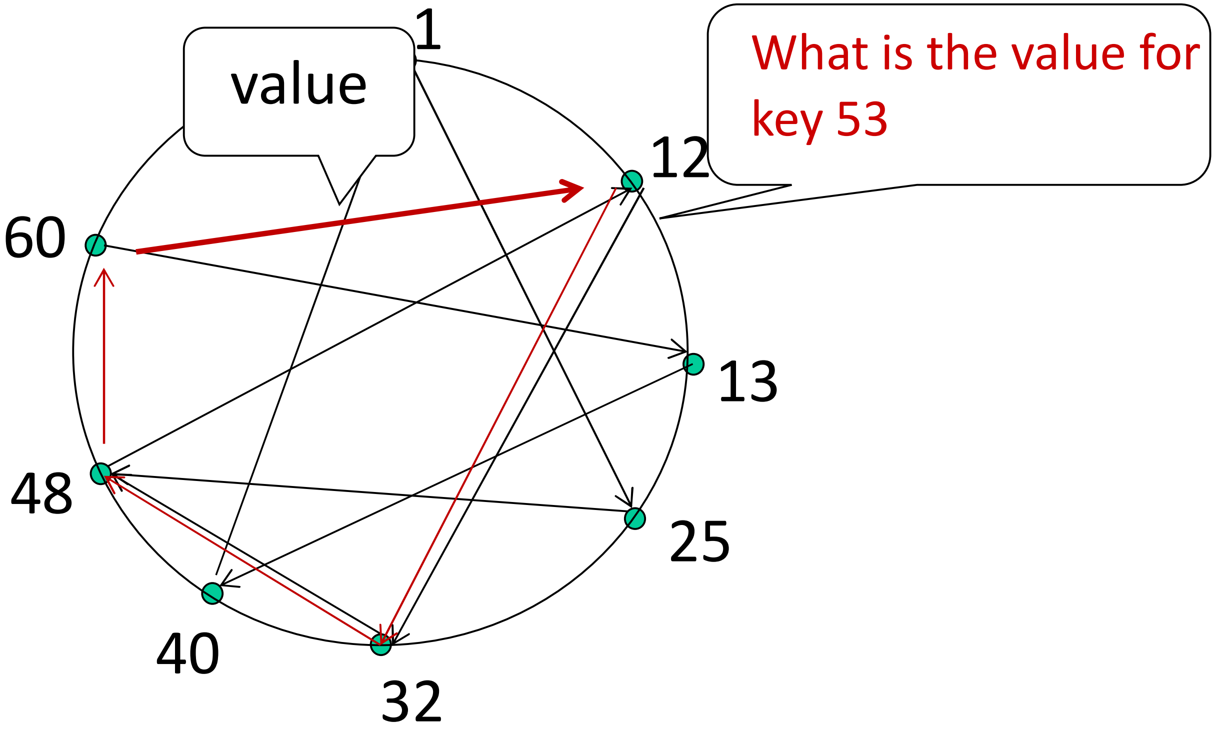


What is the value associated with key 53 ?

value

O(*N*) messages on avgerage to resolve query, when there are *N* peers

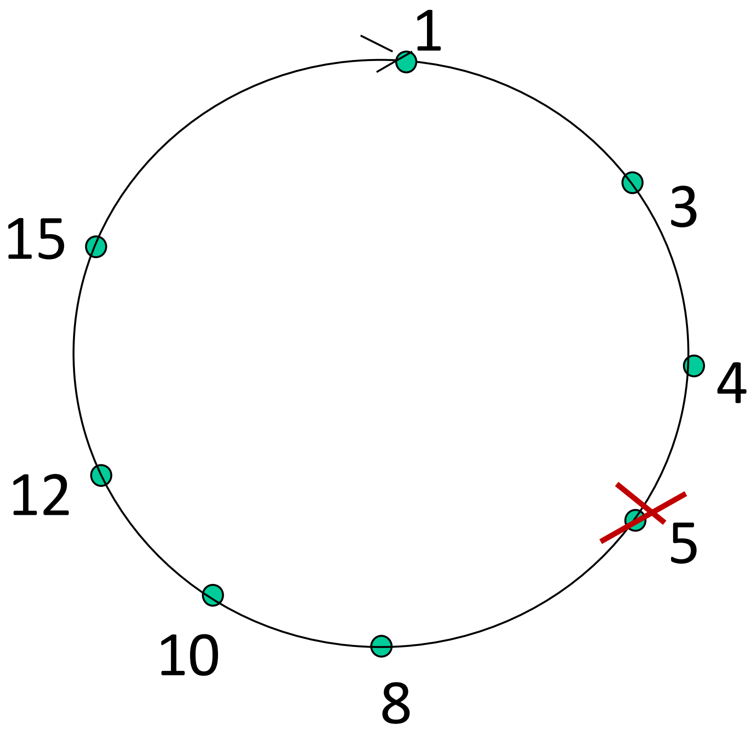
Circular DHT with shortcuts



- each peer keeps track of IP addresses of predecessor, successor, short cuts.

- reduced from 6 to 3 messages.

Peer churn

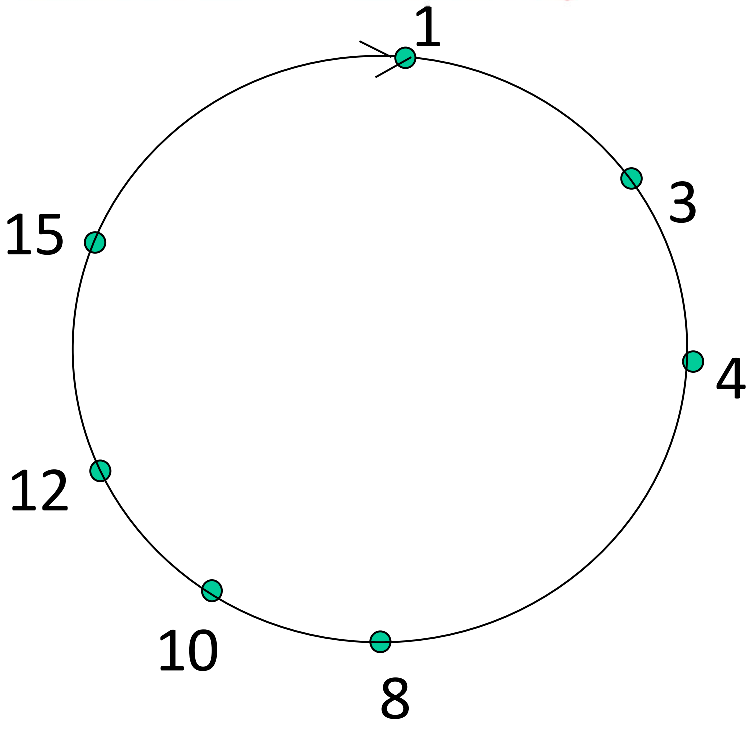


handling peer churn:

- peers may come and go (churn)
- each peer knows address of its two successors
- each peer periodically pings its two successors to check aliveness
- if immediate successor leaves, choose next successor as new immediate successor

example: peer 5 abruptly leaves

Peer churn



handling peer churn:

- peers may come and go (churn)
- each peer knows address of its two successors
- each peer periodically pings its two successors to check aliveness
- if immediate successor leaves, choose next successor as new immediate successor

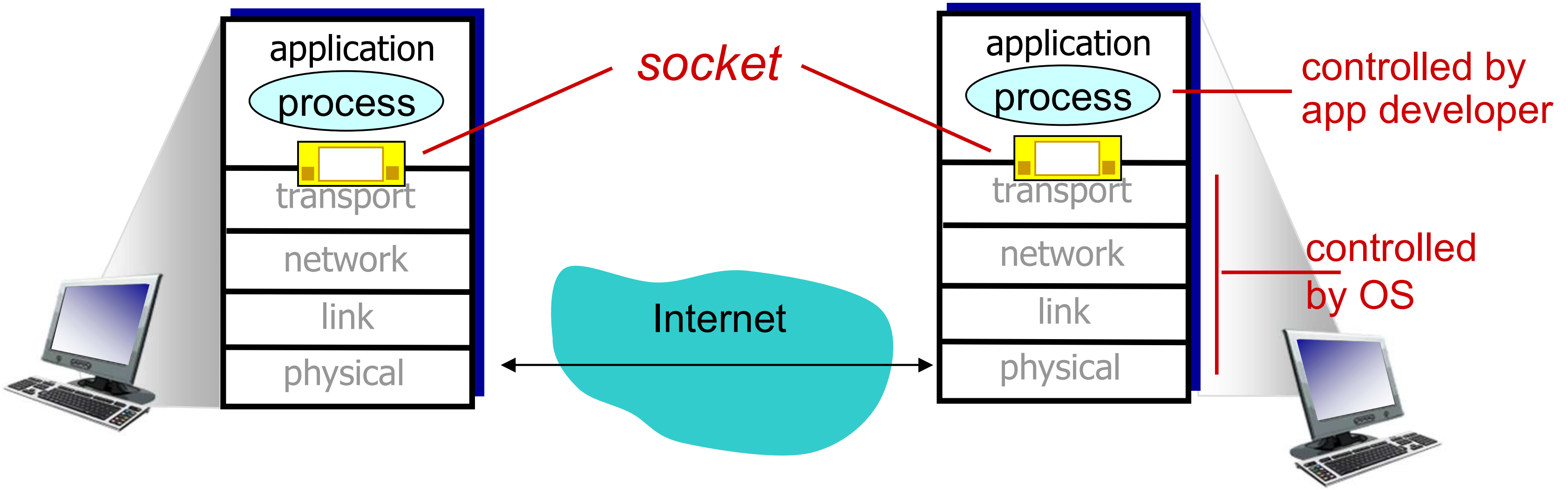
example: peer 5 abruptly leaves

- peer 4 detects peer 5's departure; makes 8 its immediate successor
- 4 asks 8 who its immediate successor is; makes 8's immediate successor its second successor.

Socket programming

goal: learn how to build client/server applications that communicate using sockets

socket: door between application process and end-end-transport protocol



Socket programming

Two socket types for two transport services:

- *UDP:* unreliable datagram
- *TCP:* reliable, byte stream-oriented

Application Example:

1. Client reads a line of characters (data) from its keyboard and sends the data to the server.
2. The server receives the data and converts characters to uppercase.
3. The server sends the modified data to the client.
4. The client receives the modified data and displays the line on its screen.

Socket programming *with UDP*

UDP: no "connection" between client & server

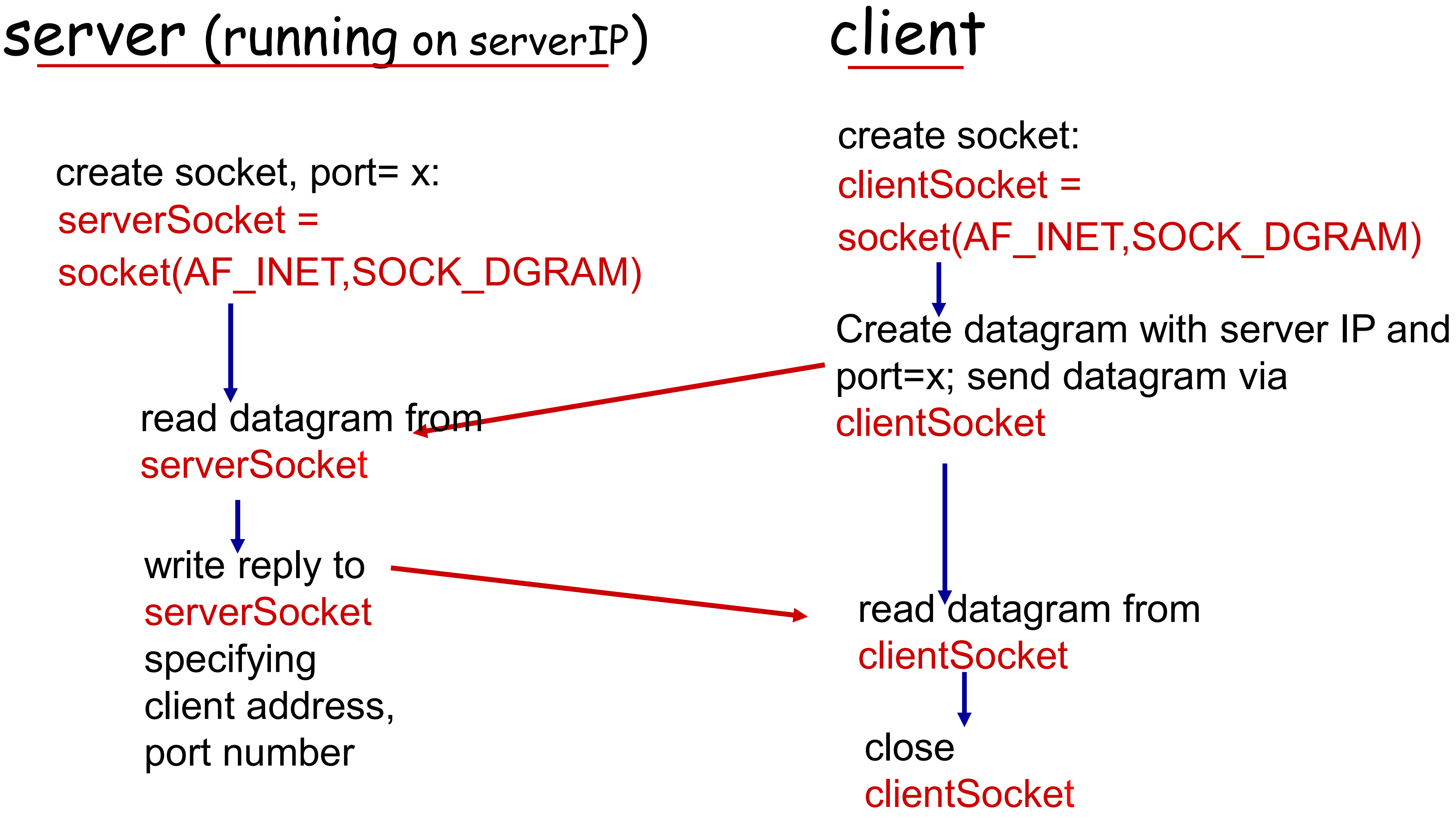
- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- rcvr extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

- UDP provides *unreliable* transfer of groups of bytes ("datagrams") between client and server

Client/server socket interaction: UDP



server (running on serverIP)

create socket, port= x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)

read datagram from
serverSocket

write reply to
serverSocket
specifying
client address,
port number

client

create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)

Create datagram with server IP and
port=x; send datagram via
clientSocket

read datagram from
clientSocket

close
clientSocket

Example app: UDP client

Python UDPClient

include Python’s socket library

```
from socket import *
serverName = ‘hostname’
serverPort = 12000
```

create UDP socket for server

```
clientSocket = socket(socket.AF_INET,
                              socket.SOCK_DGRAM)
```

get user keyboard input

```
message = raw_input(’Input lowercase sentence:’)
```

Attach server name, port to message; send into socket

```
clientSocket.sendto(message,(serverName, serverPort))
```

read reply characters from socket into string

```
modifiedMessage, serverAddress =
                              clientSocket.recvfrom(2048)
```

print out received string and close socket

```
print modifiedMessage
clientSocket.close()
```

Example app: UDP server

Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind((", serverPort))
print “The server is ready to receive”
while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

create UDP socket

bind socket to local port number 12000

loop forever

Read from UDP socket into message, getting client’s address (client IP and port)

send upper case string back to this client

Socket programming *with TCP*

client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

client contacts server by:

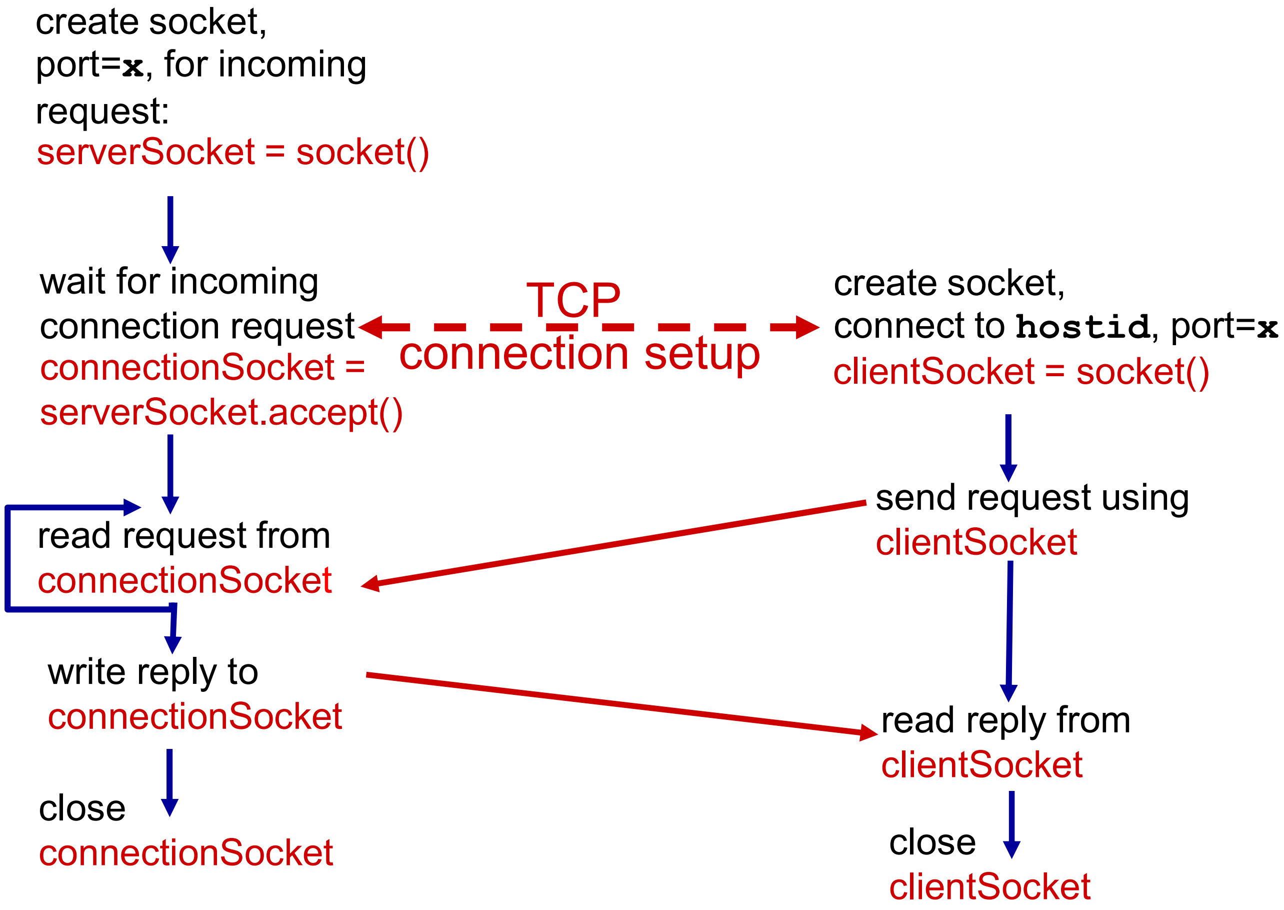
- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients

application viewpoint:

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

Client/server socket interaction: TCP



server (running on **hostid**)

create socket,
port=**x**, for incoming
request:
serverSocket = socket()

wait for incoming
connection request
connectionSocket =
serverSocket.accept()

read request from
connectionSocket

write reply to
connectionSocket

close
connectionSocket

client

create socket,
connect to **hostid**, port=**x**
clientSocket = socket()

send request using
clientSocket

read reply from
clientSocket

close
clientSocket

Example app: TCP client

Python TCPClient

```
from socket import *
serverName = ’servername’
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input(‘Input lowercase sentence:’)
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print ‘From Server:’, modifiedSentence
clientSocket.close()
```

create TCP socket for server, remote port 12000

No need to attach server name, port

Example app: TCP server

Python TCPServer

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

server waits on accept() for incoming requests, new socket created on return

read bytes from socket (but not address as in UDP)

close connection to this client (but *not* welcoming socket)

```
 from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind((‘’,serverPort))
serverSocket.listen(1)
print ‘The server is ready to receive’
while 1:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```