

# Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks

2.7 socket programming with UDP and TCP

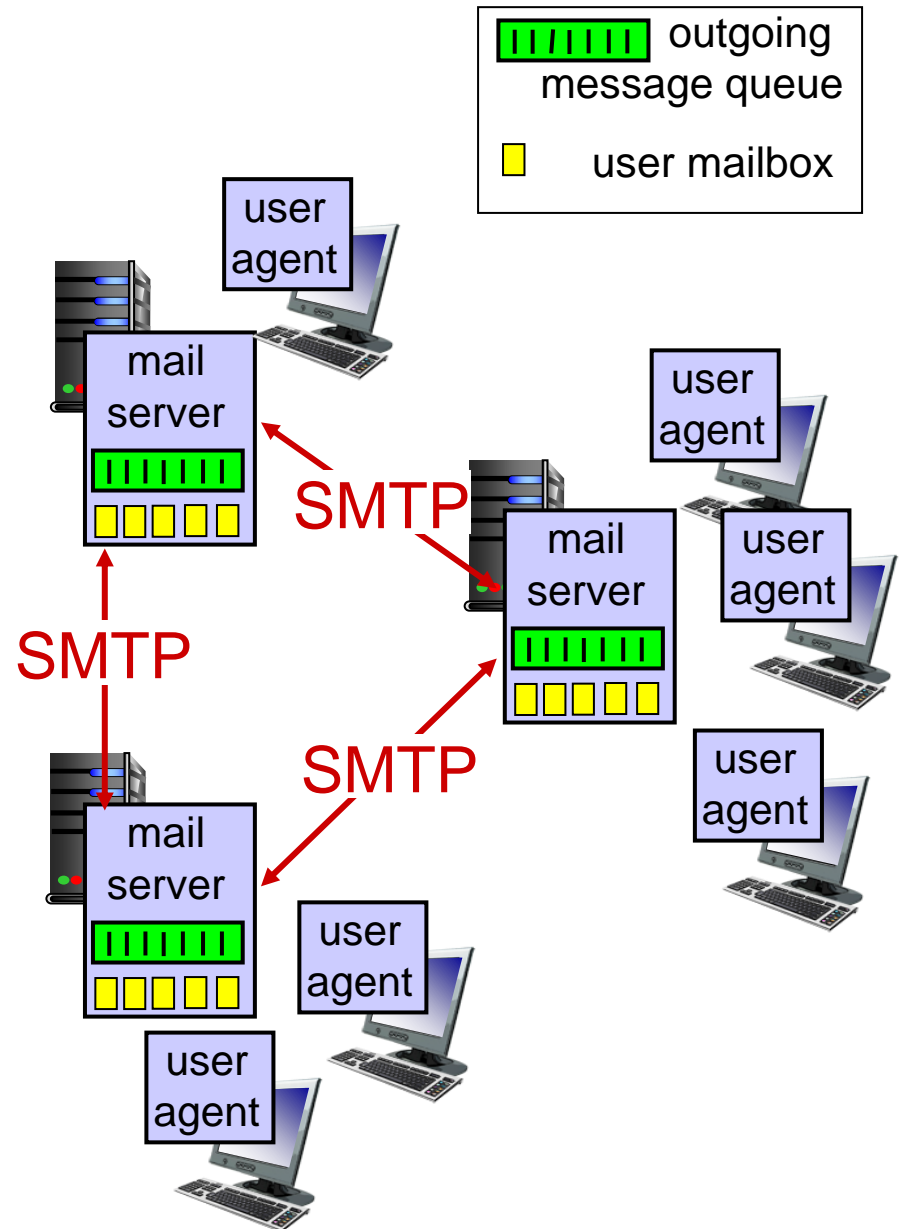
# Electronic mail

## *Three major components:*

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

## *User Agent*

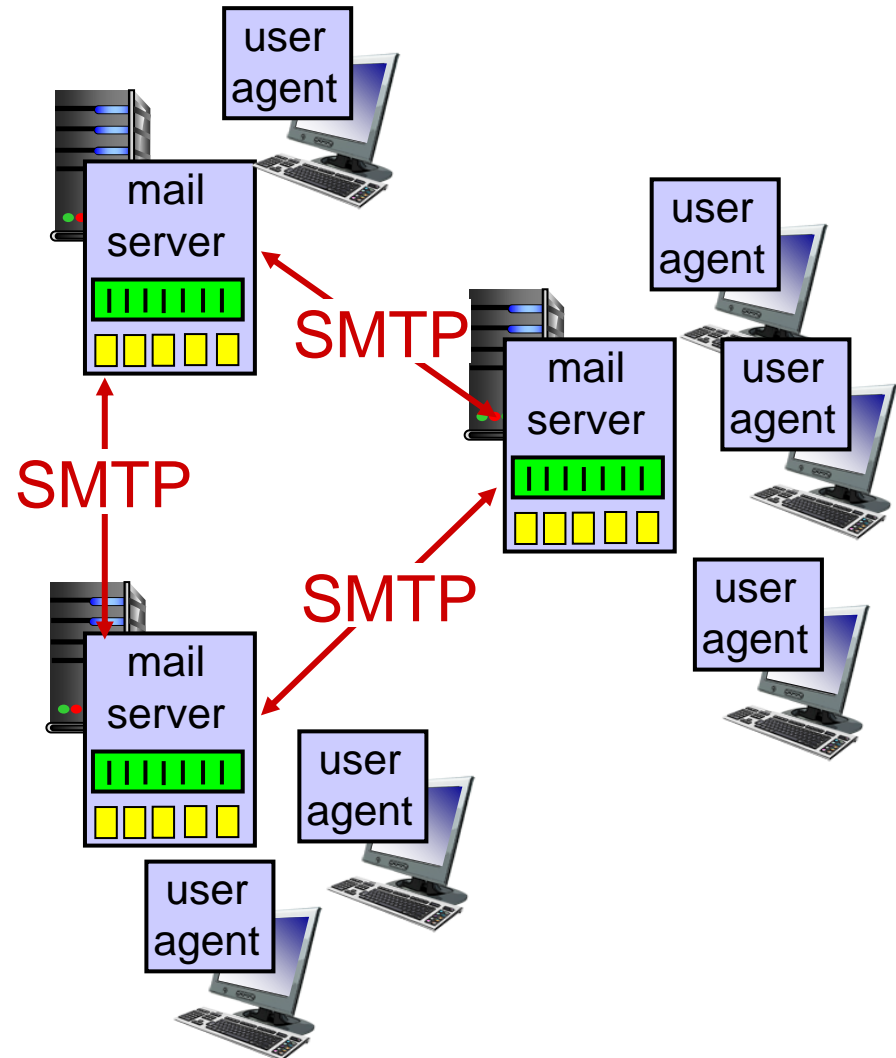
- ❖ a.k.a. “mail reader”
- ❖ composing, editing, reading mail messages
- ❖ e.g., Outlook, Thunderbird, iPhone mail client
- ❖ outgoing, incoming messages stored on server



# 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

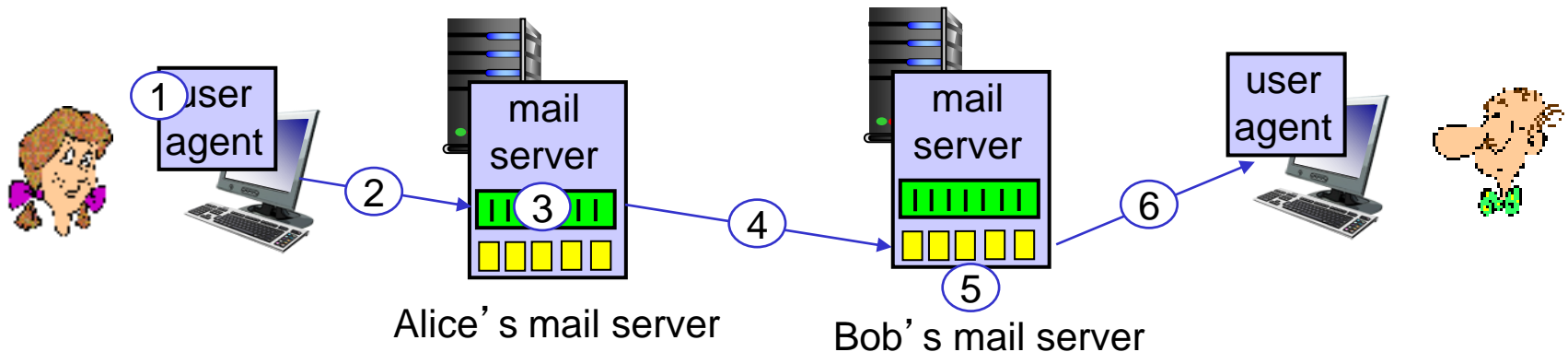


# Electronic Mail: SMTP [RFC 2821]

- ❖ 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
  - Base64 to convert binary to text

# 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



# Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

# Try SMTP interaction for yourself:

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

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

# 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
- ❖ both have ASCII command/response interaction, status codes
- ❖ HTTP: each object encapsulated in its own response msg
- ❖ SMTP: multiple objects sent in multipart msg



# Mail message format

SMTP: protocol for exchanging email msgs

RFC 822: standard for text message format:

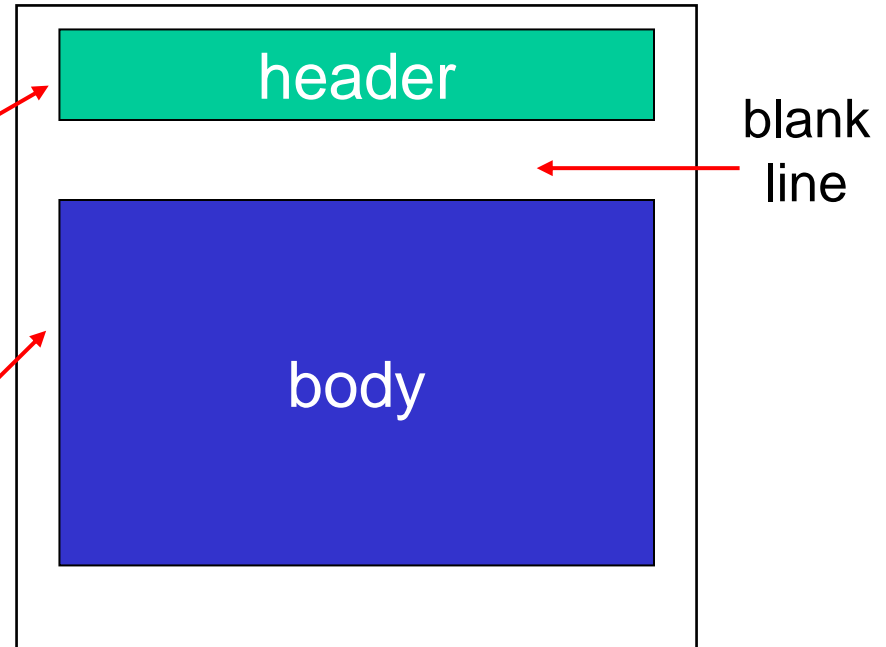
❖ header lines, e.g.,

- To:
- From:
- Subject:

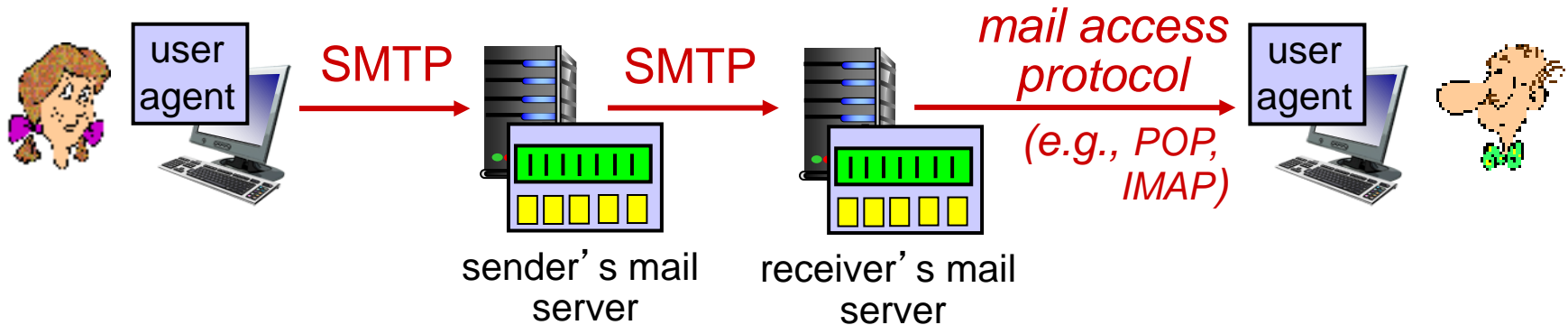
*different* from SMTP MAIL  
FROM, RCPT TO:  
commands!

❖ 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 [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
  - 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
- ❖ keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name

# Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks

2.7 socket programming with UDP and TCP

# DNS: domain name system

*people*: many identifiers:

- SSN, name, passport #

*Internet hosts, 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, name servers communicate to *resolve* names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network's “edge”

# DNS: services, structure

## *DNS services*

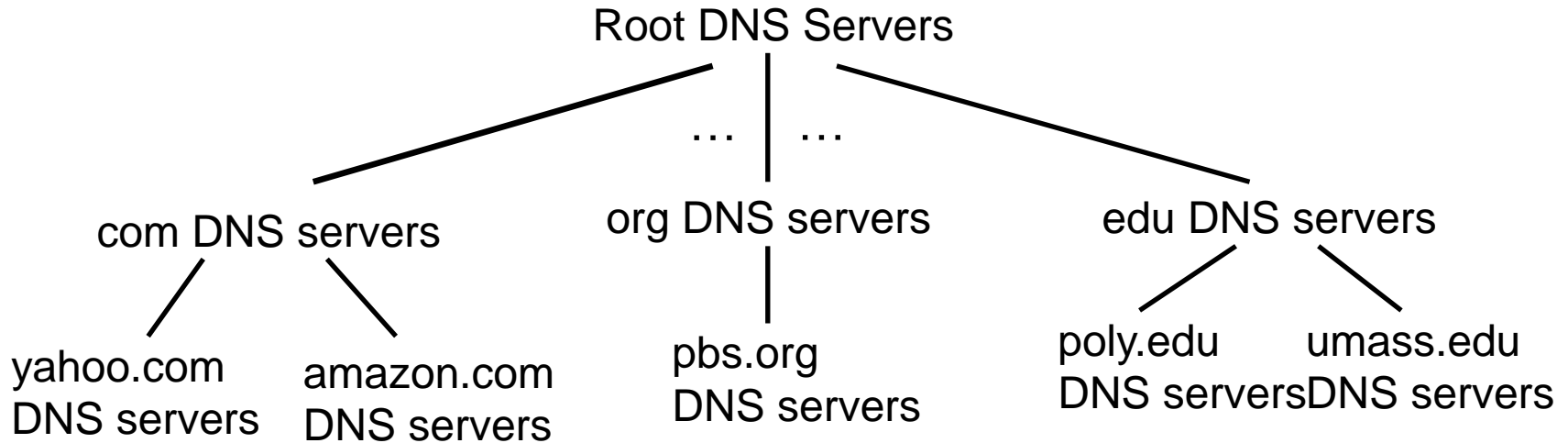
- ❖ hostname to IP address translation
- ❖ host aliasing
  - canonical, alias names
- ❖ mail server aliasing
- ❖ load distribution
  - replicated Web servers: many IP addresses correspond to one name

## *why not centralize DNS?*

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

*A: doesn't scale!*

# DNS: a distributed, hierarchical database



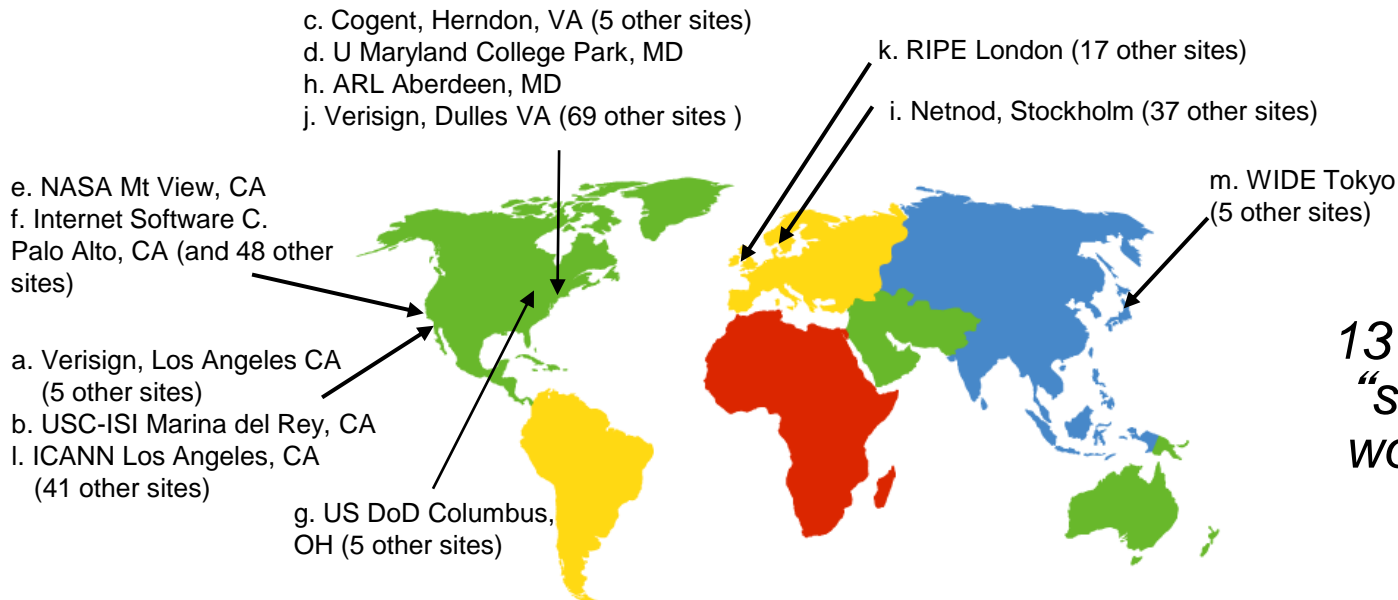
*client wants IP for www.amazon.com; 1<sup>st</sup> approx:*

- ❖ 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
- ❖ root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server



*13 root name  
“servers”  
worldwide*

# 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
- 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

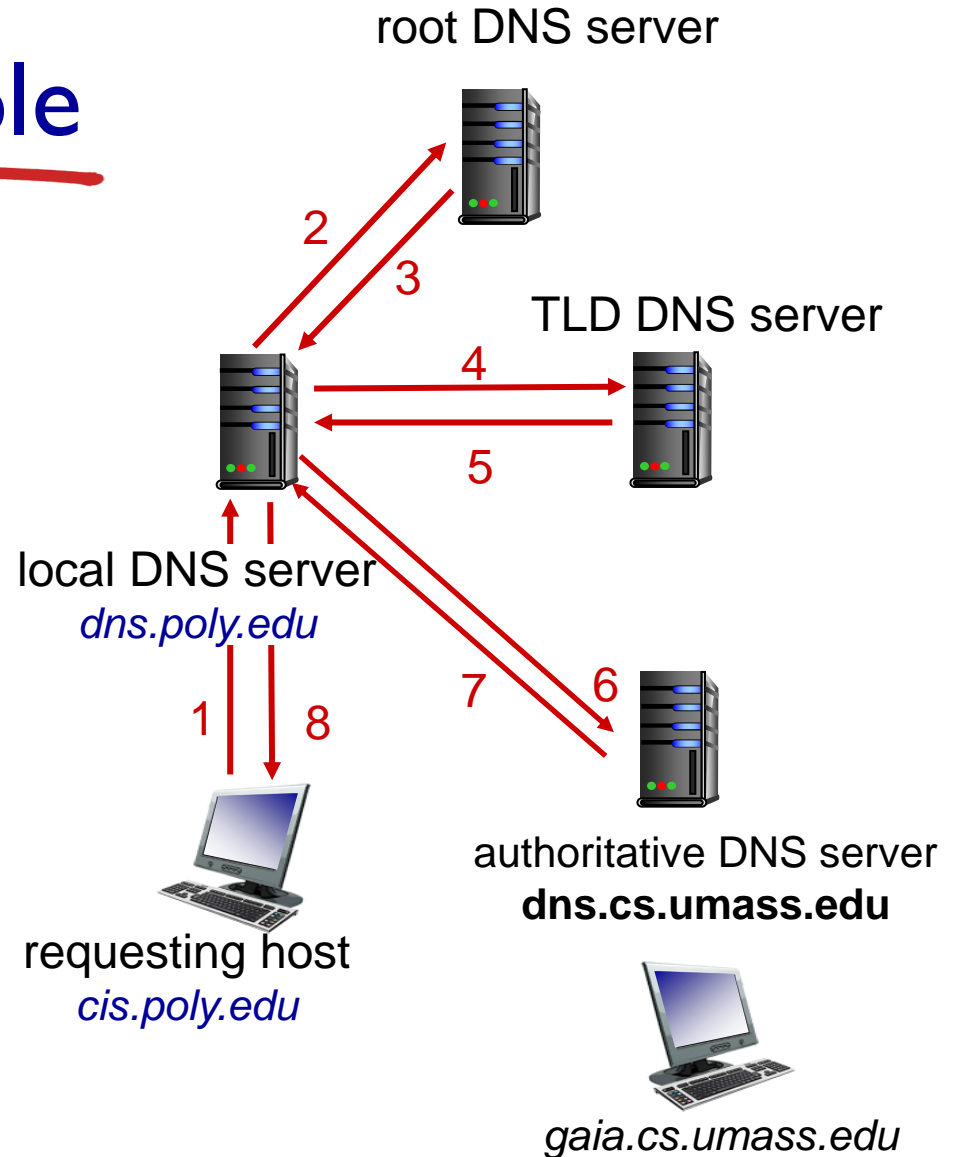
- ❖ does not strictly belong to hierarchy
- ❖ each ISP (residential ISP, company, university) has one
  - also called “default name server”
- ❖ when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
 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

# DNS name resolution example

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

## *iterated query:*

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



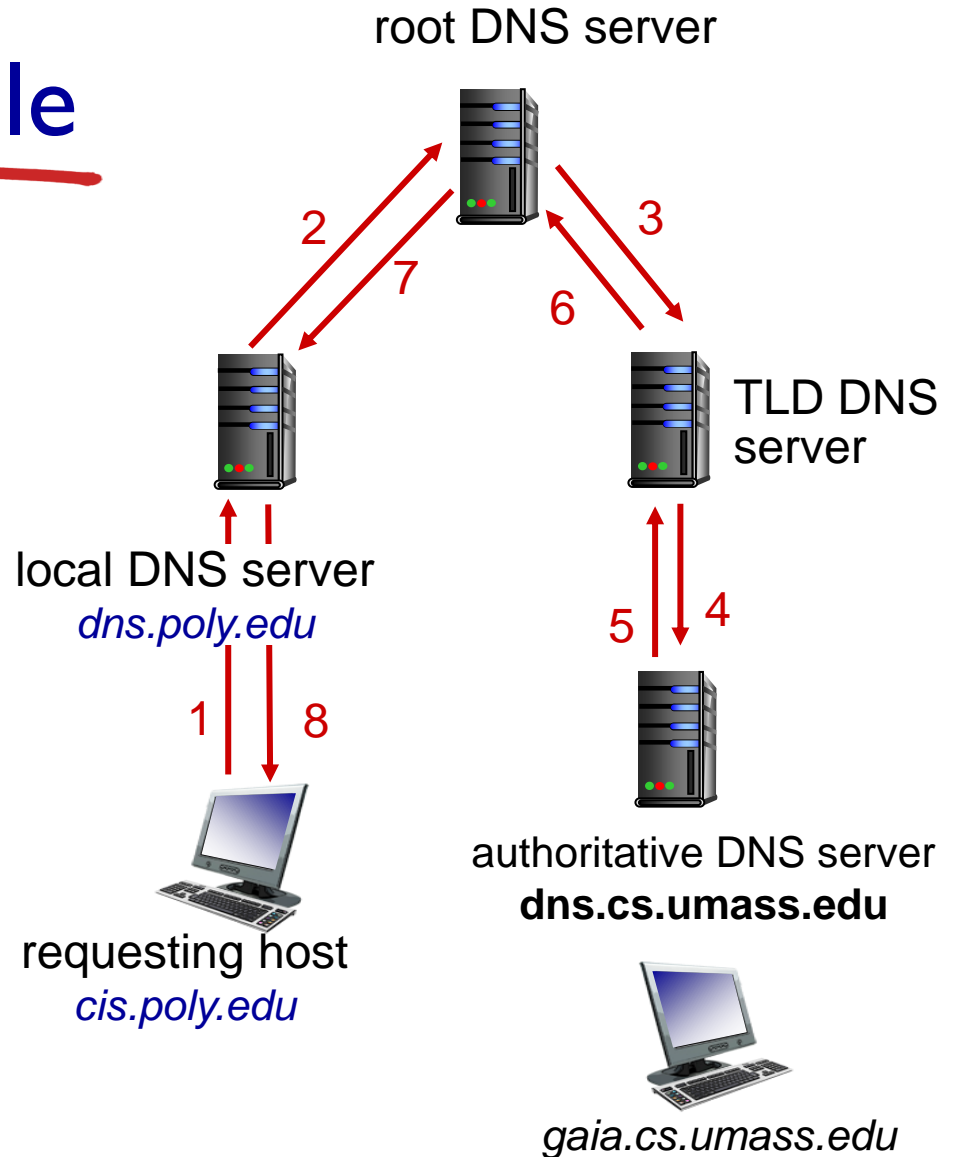
# DNS: caching, updating records

- ❖ 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

# DNS name resolution example

## *recursive query:*

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



# DNS records

**DNS:** distributed db storing resource records (RR)

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

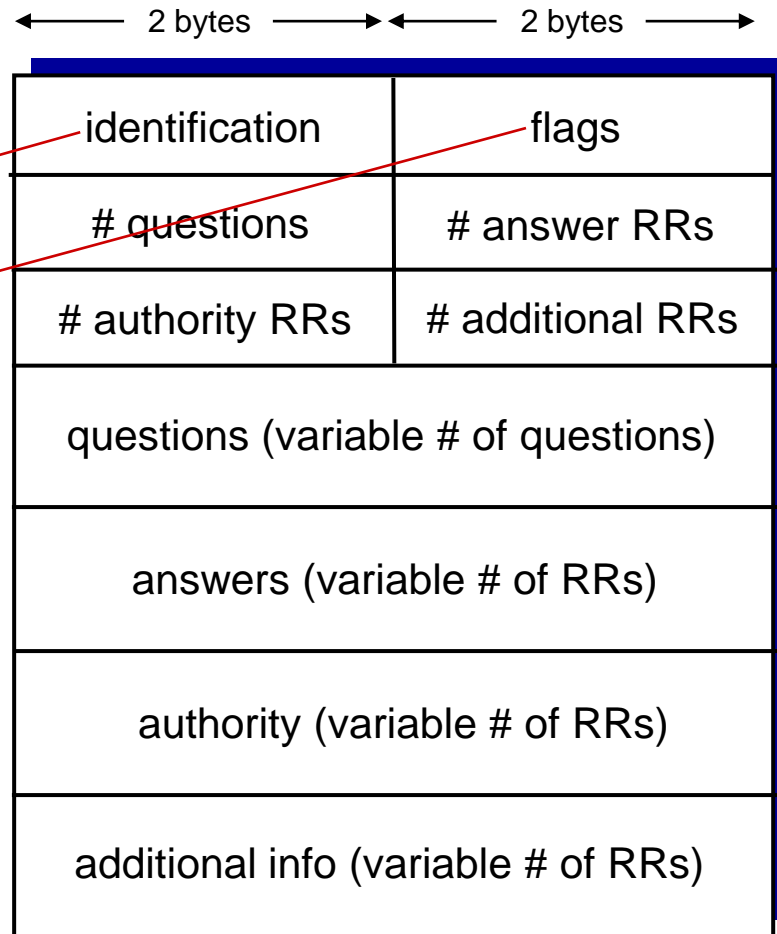
- **value** is name of mailserver associated with **name**

# DNS protocol, messages

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

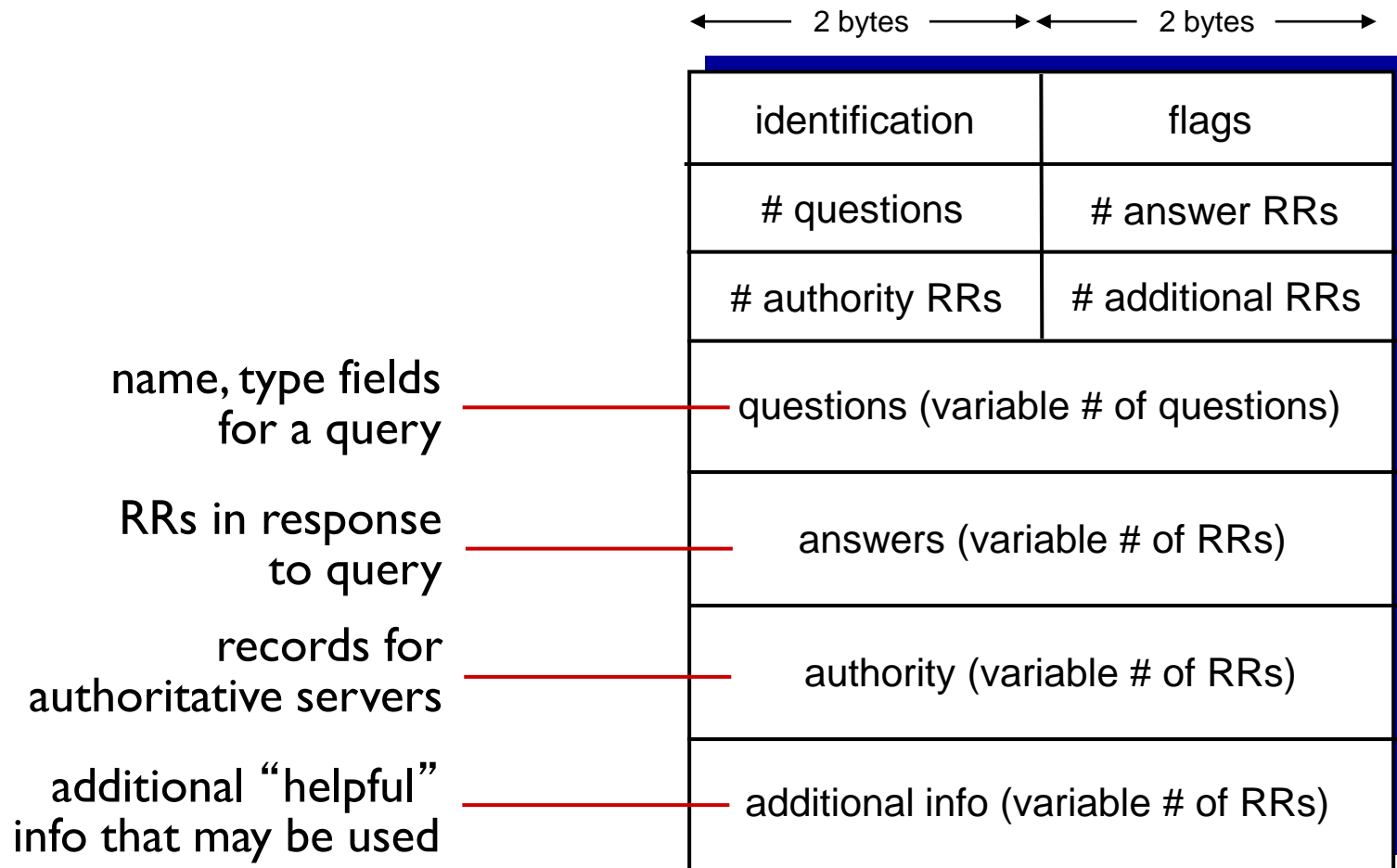
msg header

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





# DNS protocol, messages



# 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)
- ❖ create authoritative server type A record for www.networkutopia.com; type MX record for networkutopia.com at *authoritative DNS server*
- ❖ How do people get IP address of your Web site?

# Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks

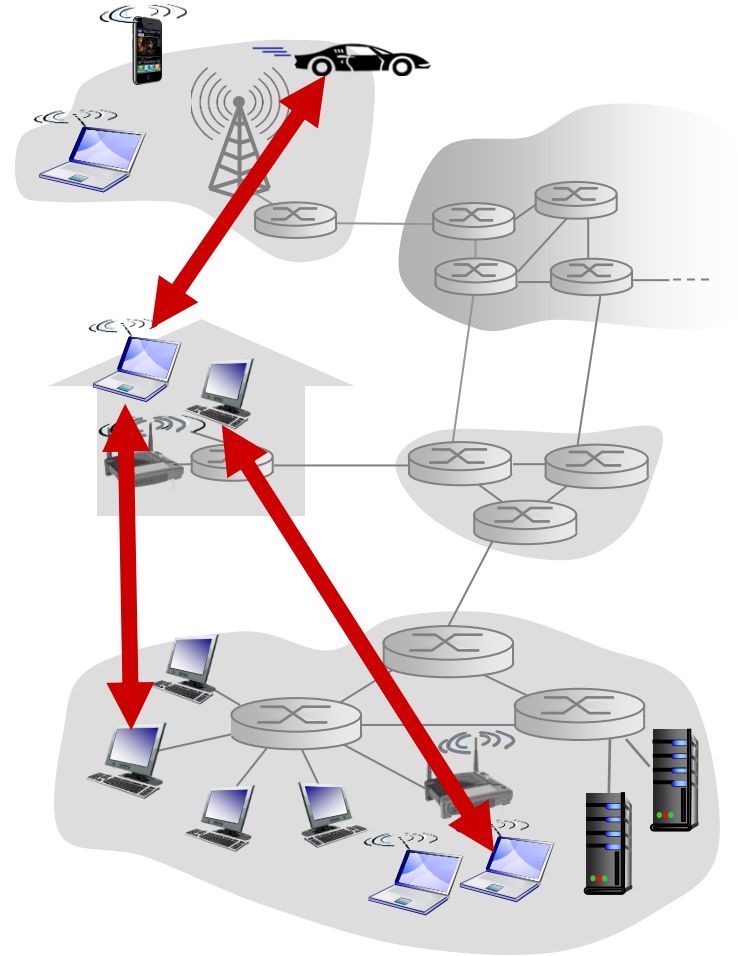
2.7 socket programming with UDP and TCP

# Pure P2P architecture

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

## *examples:*

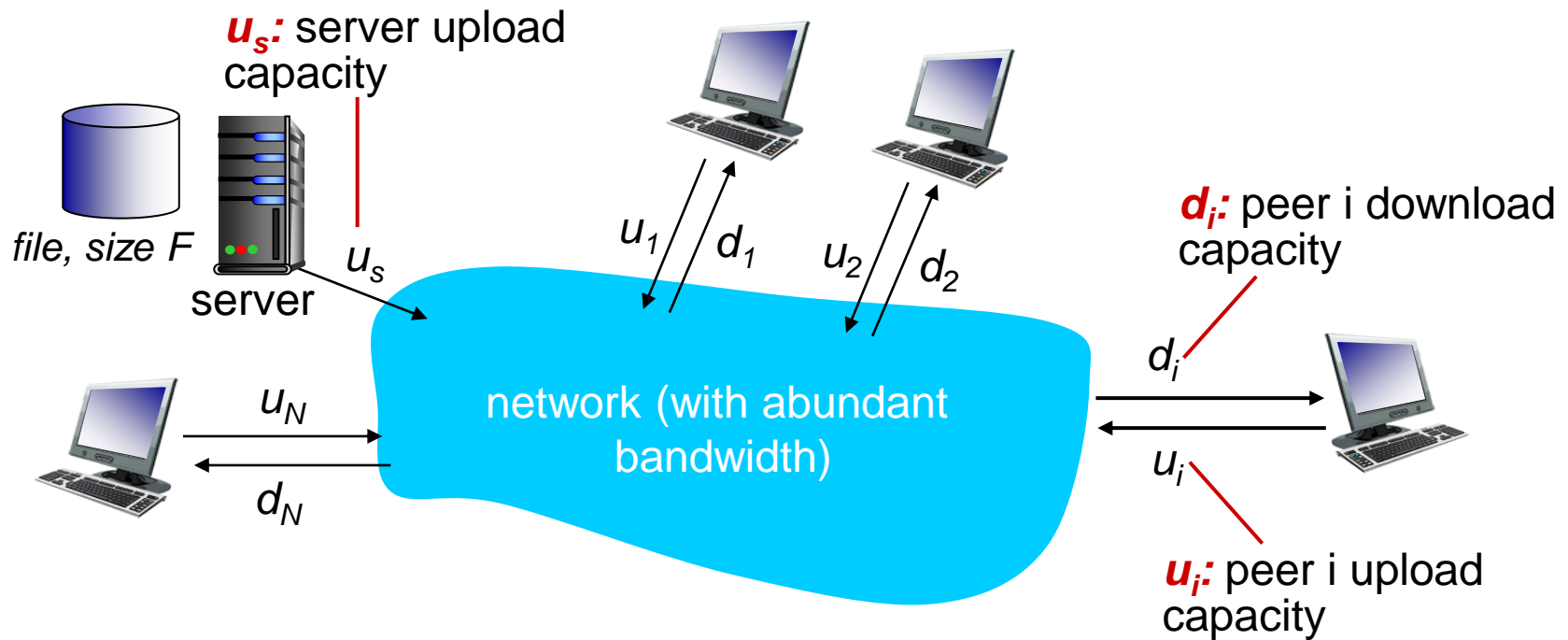
- file distribution (BitTorrent)
- Streaming (PPTV)
- VoIP (Skype)



# 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



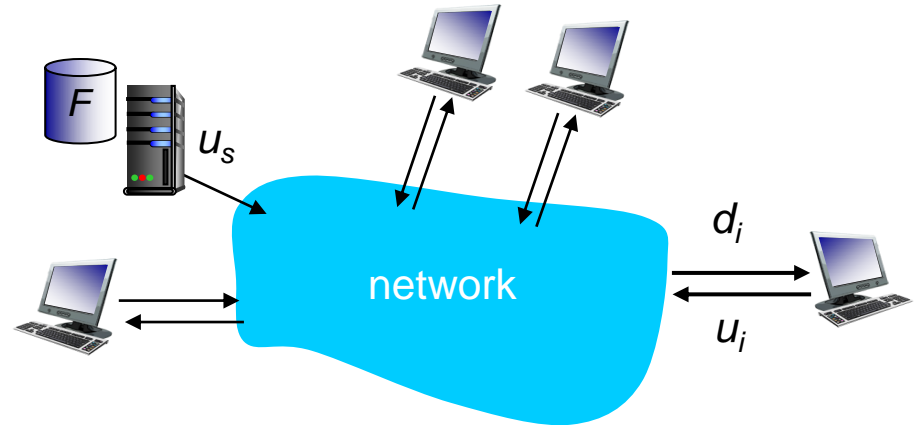
# File distribution time: client-server

- ❖ **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:  $F/d_{\min}$



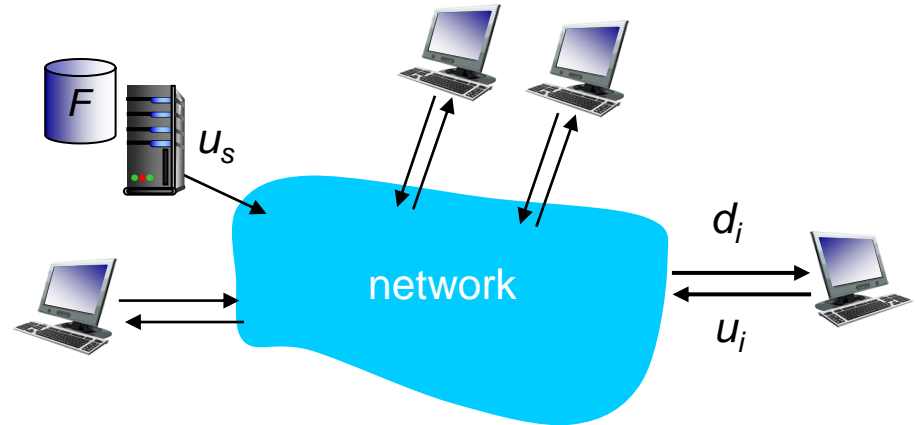
*time to distribute  $F$   
to  $N$  clients using  
client-server approach*

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

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 (limiting max download rate) is  $u_s + \sum u_i$



*time to distribute  $F$   
to  $N$  clients using  
P2P approach*

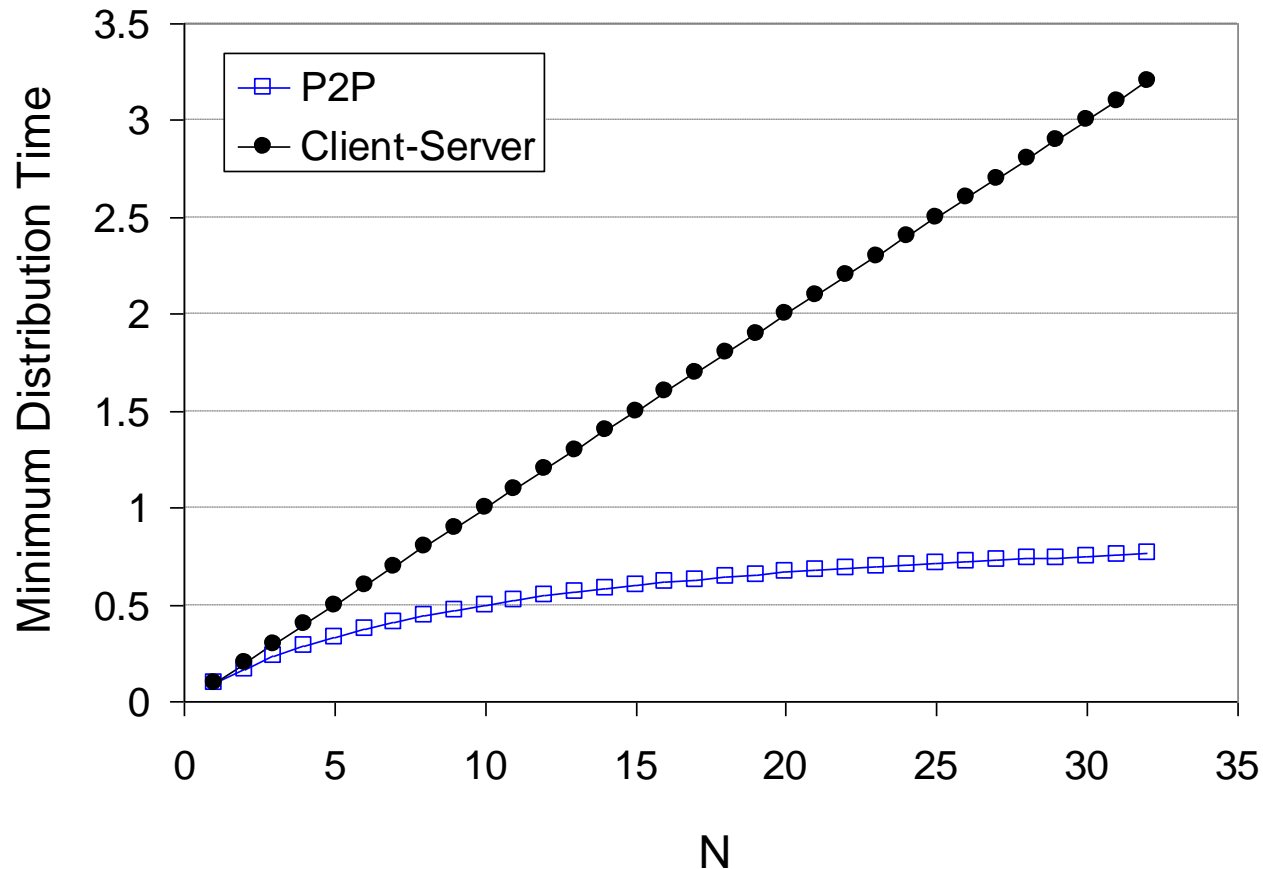
$$D_{P2P} \geq \max\{F/u_s, F/d_{\min}, NF/(u_s + \sum u_i)\}$$

increases linearly in  $N$  ...

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

# Client-server vs. P2P: example

client upload rate =  $u$ ,  $F/u = 1$  hour,  $u_s = 10u$ ,  $d_{min} \geq u_s$





# Example

Consider distributing a file of  $F = 1$  Gbytes to 10 peers. The server has an upload rate of  $u_s = 80$  Mbps, and each peer has a download rate of  $d_i = 20$  Mbps and an upload rate of  $u_i = 8$  Mbps.

a. What is the minimum distribution time for server-client distribution?

$$\begin{aligned} D_{c-s} &= \max\{NF/u_s, F/d_{\min}\} \\ &= \max\{10 \cdot 10^9 \cdot 8 / (80 \cdot 10^6), 10^9 \cdot 8 / (20 \cdot 10^6)\} \\ &= \max\{1000, 400\} \\ &= 1000 \end{aligned}$$

b. What is the minimum distribution time for P2P distribution?

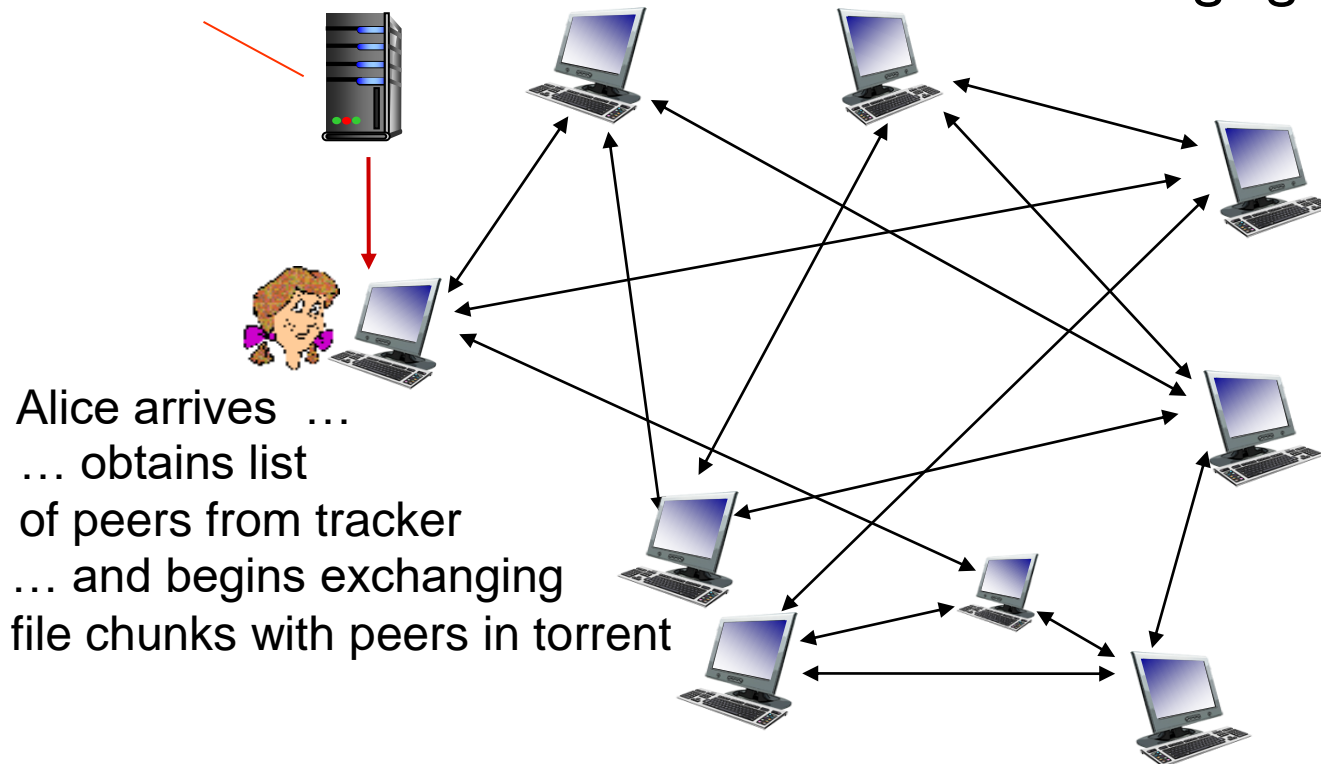
$$\begin{aligned} D_{P2P} &= \max\{F/u_s, F/d_{\min}, NF/(u_s + \sum u_i)\} \\ &= \max\{10^9 \cdot 8 / (80 \cdot 10^6), 10^9 \cdot 8 / (20 \cdot 10^6), \\ &\quad 10 \cdot 10^9 \cdot 8 / (80 \cdot 10^6 + 10 \cdot 8 \cdot 10^6)\} \\ &= \max\{100, 400, 500\} \\ &= 500 \end{aligned}$$

# P2P file distribution: BitTorrent

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

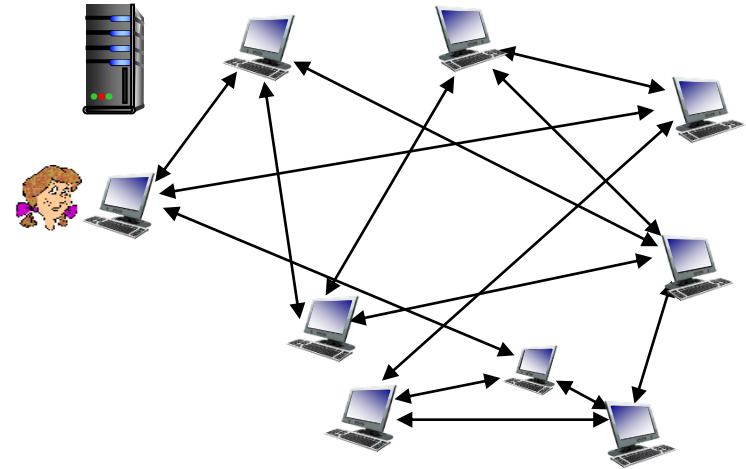
**tracker:** tracks peers participating in torrent

**torrent:** group of peers exchanging chunks of a file



# 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
- ❖ peer may change peers with whom it exchanges chunks
- ❖ **churn**: 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:*

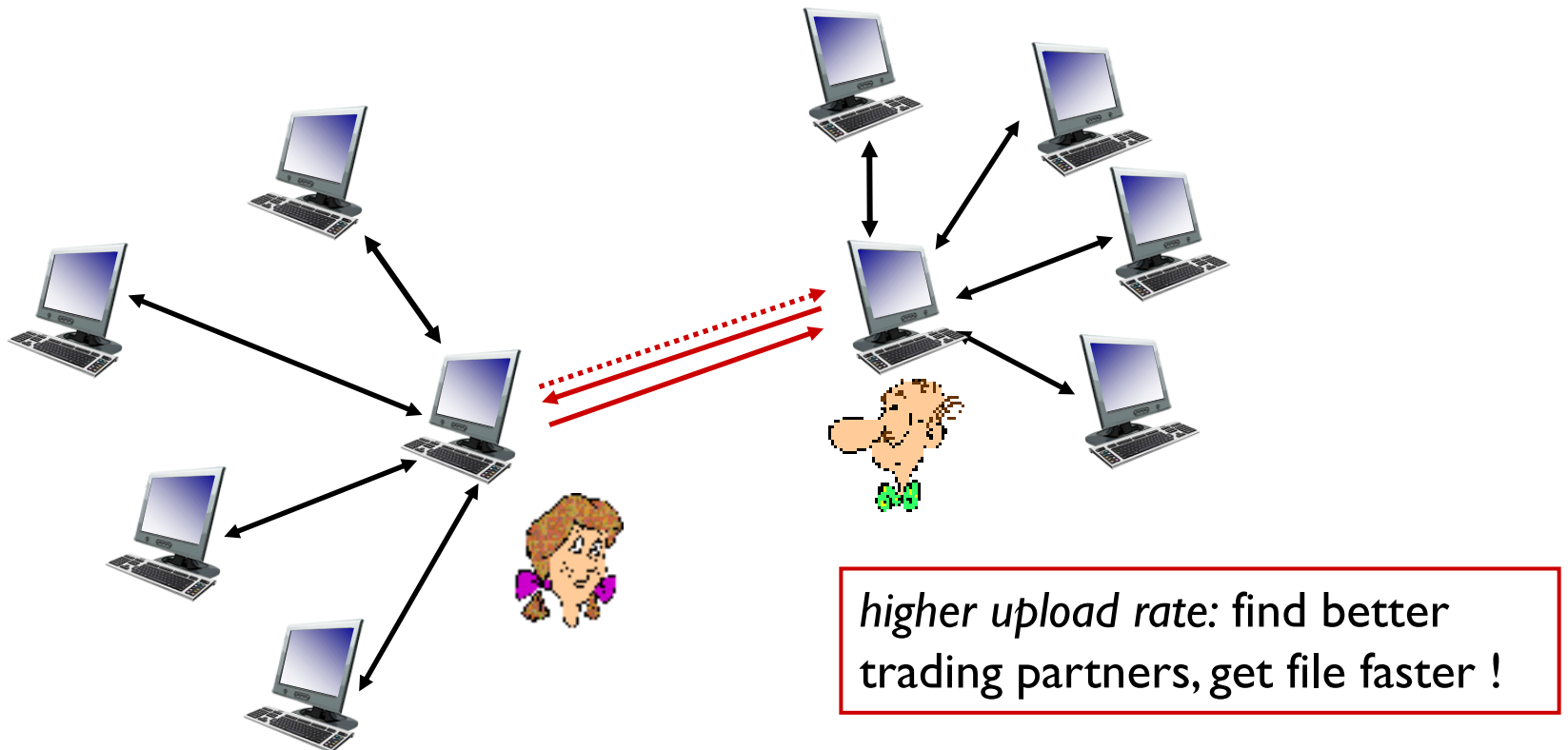
- ❖ at any given time, different peers have different subsets of file chunks
- ❖ periodically, Alice asks each peer for list of chunks that they have
- ❖ Alice requests missing chunks from peers, rarest first

## *sending chunks: tit-for-tat*

- ❖ Alice sends chunks to those four peers currently sending her chunks *at highest rate*
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every 10 secs
- ❖ every 30 secs: randomly select another peer, starts sending chunks
  - “optimistically unchoke” this peer
  - newly chosen peer may join top 4

# BitTorrent: tit-for-tat

- (1) Alice “optimistically unchokes” Bob
- (2) Alice becomes one of Bob’s top-four providers; Bob reciprocates
- (3) Bob becomes one of Alice’s top-four providers



# Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

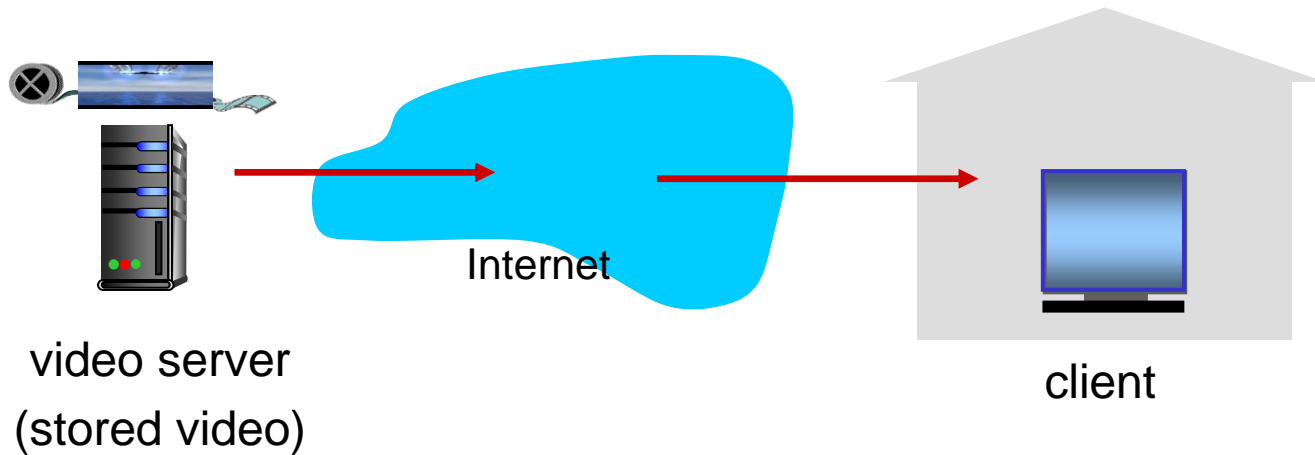
# Video Streaming and CDNs: context

- video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
  - ~1B YouTube users, ~75M Netflix users
- challenge: scale - how to reach ~1B users?
  - single mega-video server won't work (why?)
- challenge: heterogeneity
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- **solution:** distributed, application-level infrastructure



# Streaming stored video:

simple scenario:





# Streaming multimedia: DASH

- *DASH*: *D*ynamic, *A*daptive *S*treaming over *H*TTP
- *server*:
  - divides video file into multiple chunks
  - each chunk stored, encoded at different rates
  - *manifest file*: provides URLs for different chunks
- *client*:
  - periodically measures server-to-client bandwidth
  - consulting manifest, requests one chunk at a time
    - chooses maximum coding rate sustainable given current bandwidth
    - can choose different coding rates at different points in time (depending on available bandwidth at time)

# Streaming multimedia: DASH

---

- *DASH: Dynamic, Adaptive Streaming over HTTP*
- “intelligence” at client: client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is “close” to client or has high available bandwidth)

# Content distribution networks

---

- *challenge*: how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?
- *option 1*: single, large “mega-server”
  - single point of failure
  - point of network congestion
  - long path to distant clients
  - multiple copies of video sent over outgoing link

....quite simply: this solution *doesn't scale*

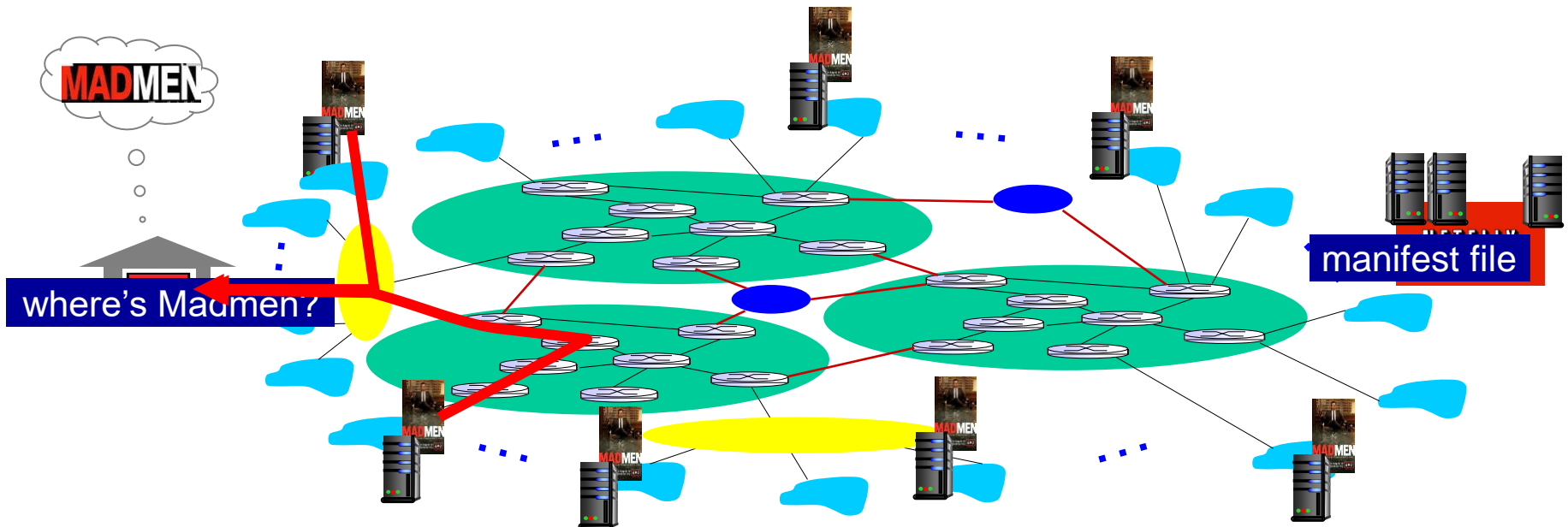
# Content distribution networks

---

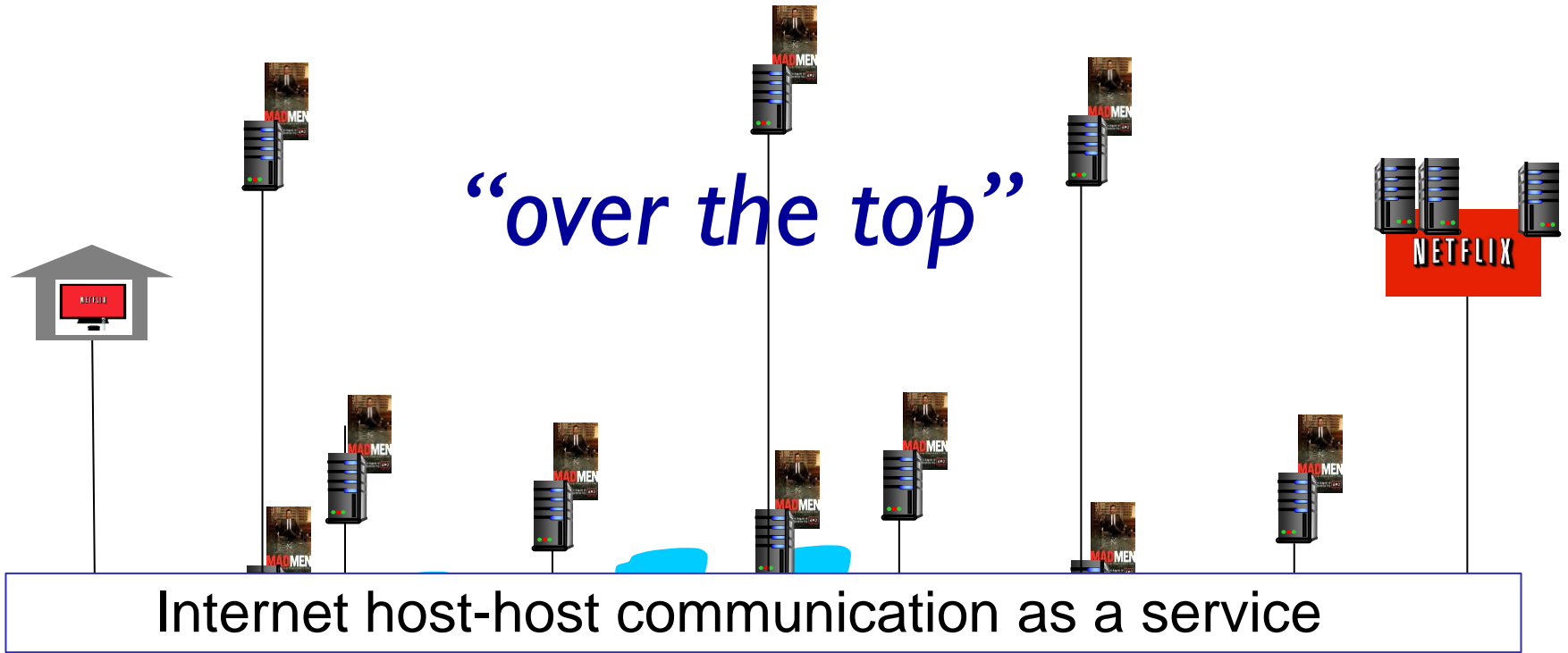
- *challenge*: how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?
- *option 2*: store/serve multiple copies of videos at multiple geographically distributed sites (*CDN*)
  - *enter deep*: push CDN servers deep into many access networks
    - close to users
    - used by Akamai, 1700 locations
  - *bring home*: smaller number (10's) of larger clusters in POPs near (but not within) access networks
    - used by Limelight

# Content Distribution Networks (CDNs)

- CDN: stores copies of content at CDN nodes
  - e.g. Netflix stores copies of MadMen
- subscriber requests content from CDN
  - directed to nearby copy, retrieves content
  - may choose different copy if network path congested



# Content Distribution Networks (CDNs)



**OTT challenges:** coping with a congested Internet

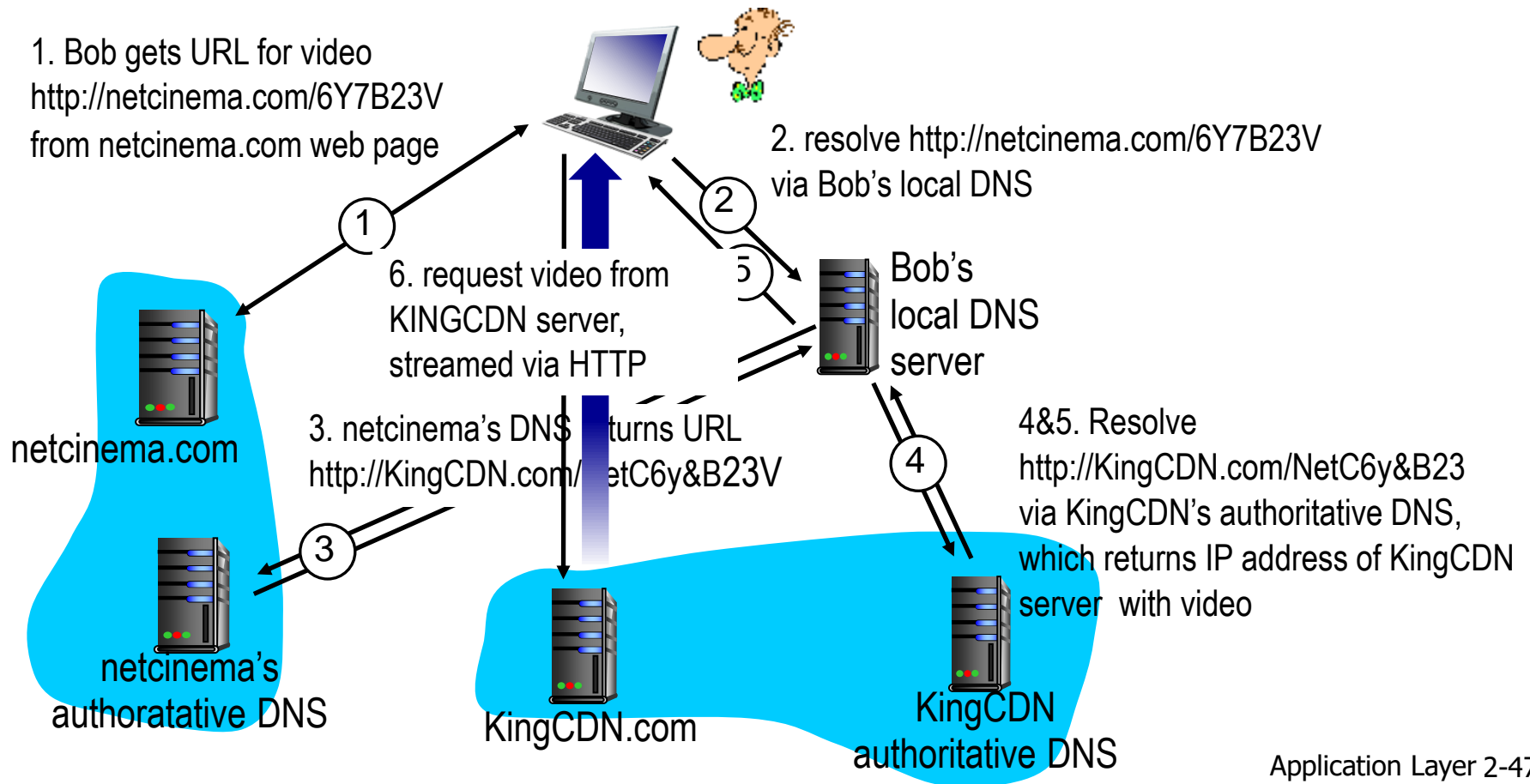
- from which CDN node to retrieve content?
- viewer behavior in presence of congestion?
- what content to place in which CDN node?

*more .. in chapter 7*

# CDN content access: a closer look

Bob (client) requests video `http://netcinema.com/6Y7B23V`

- video stored in CDN at `http://KingCDN.com/NetC6y&B23V`



# Case study: Netflix

