



THE UNIVERSITY OF WINNIPEG

# ACS-3911-050 Computer Network

## Chapter 2 Application Layer

# ACS-3911-050 – Slides Used In The Course

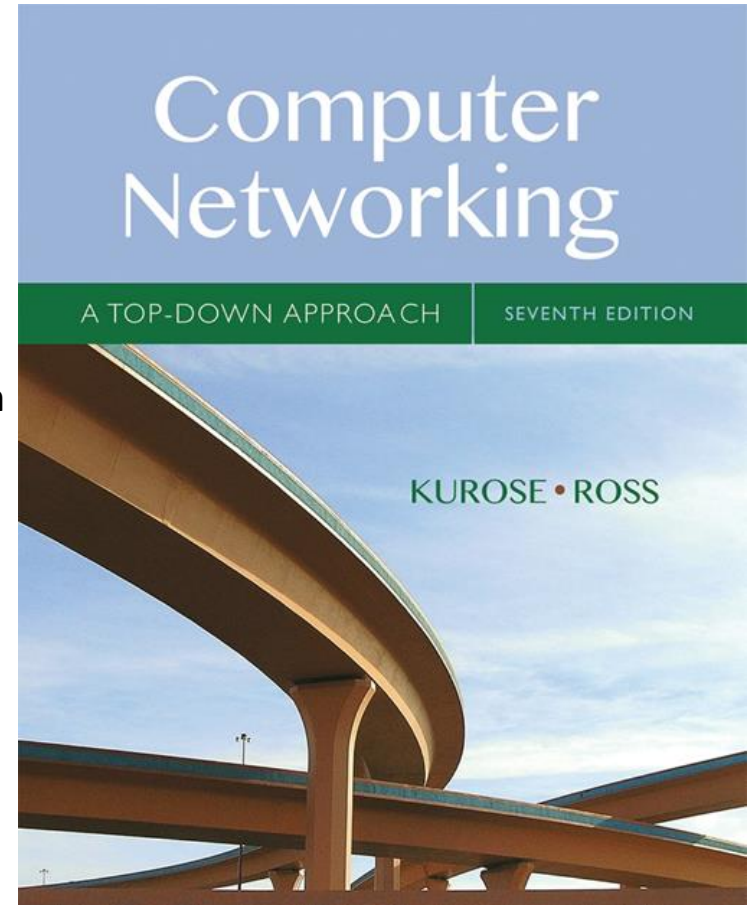
## A note on the use of these PowerPoint slides:

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

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

Thanks and enjoy! JFK/KWR

All material copyright 1996-2016  
J.F Kurose and K.W. Ross, All Rights Reserved



# Conditional GET



- **Goal:** don't send object if cache has up-to-date cached version

client



server



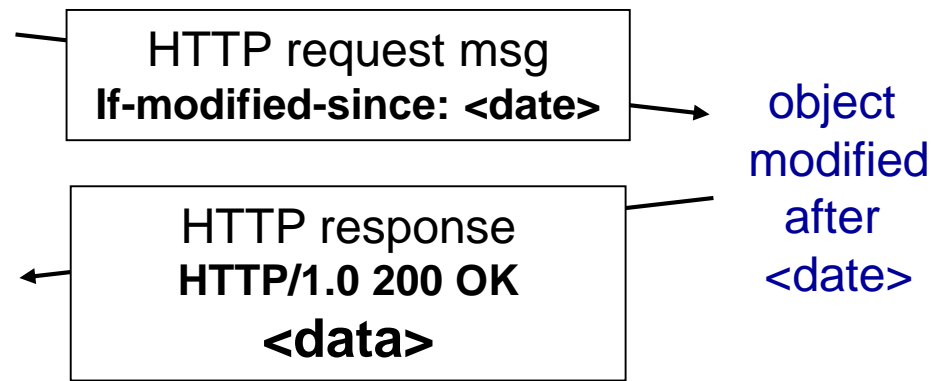
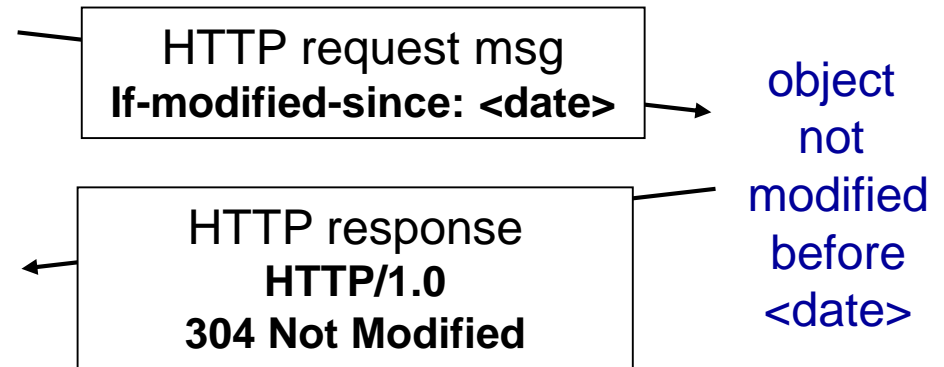
- no object transmission delay
- lower link utilization

- *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**



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

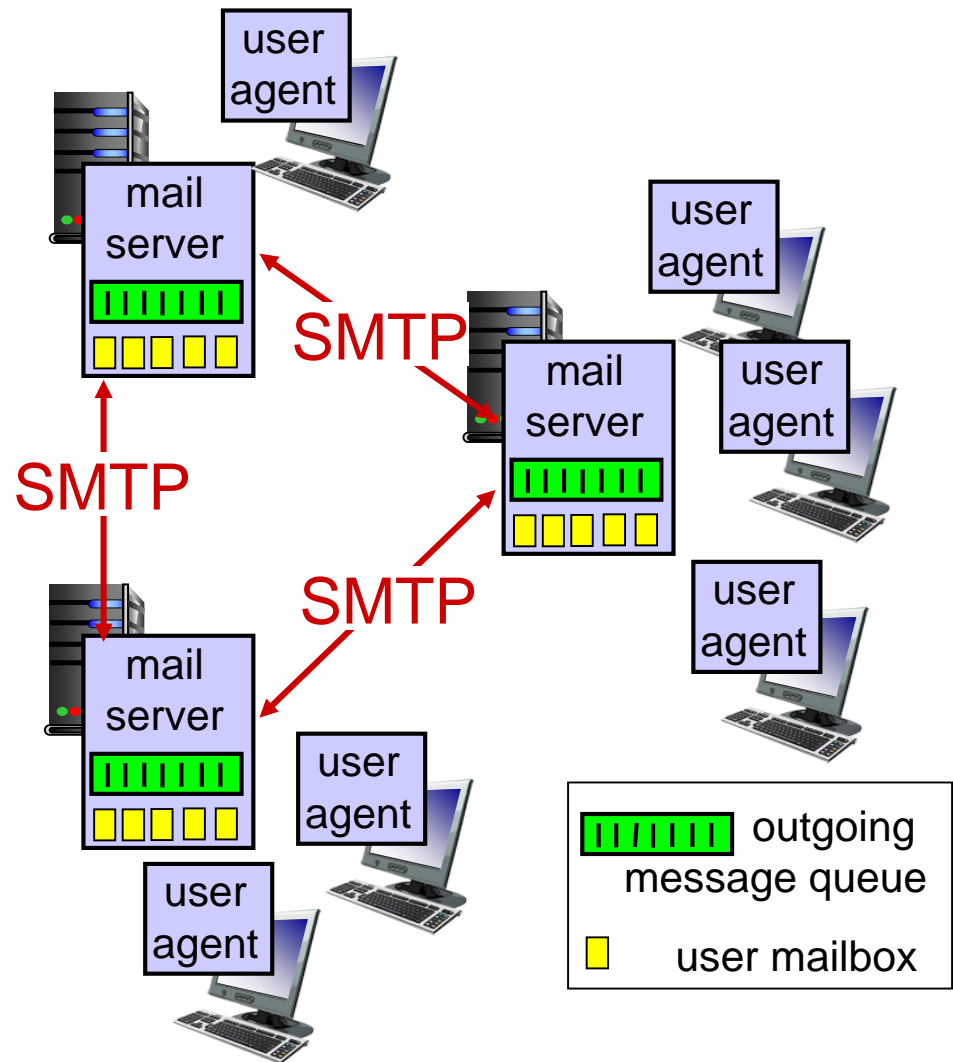
# Electronic Mail

## Three major components:

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

## User Agent

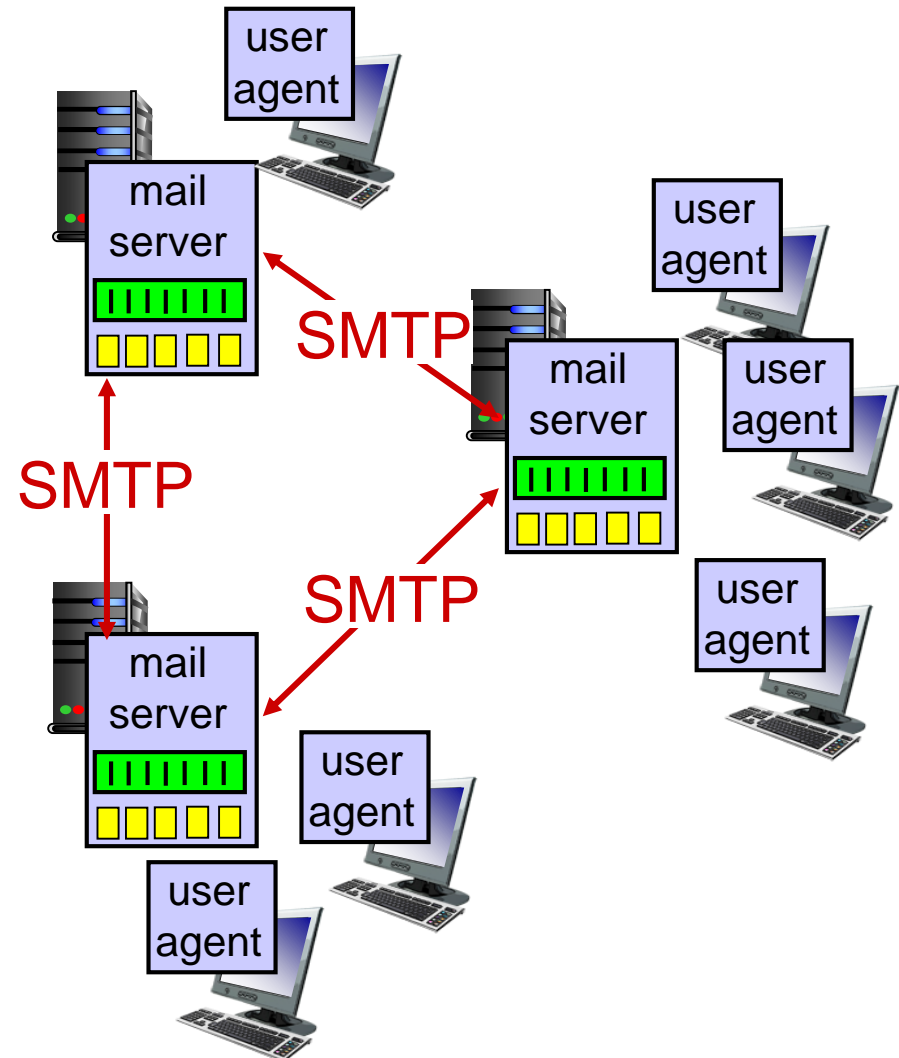
- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



# Electronic Mail: Mail Server

## 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



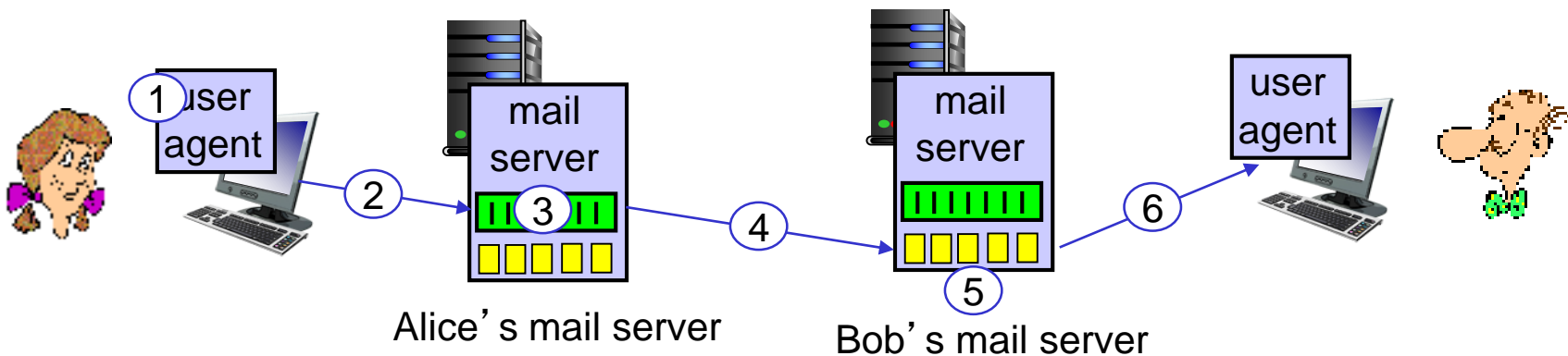


- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction (like HTTP, FTP)
  - **commands:** ASCII text
 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



# 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 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 message
- SMTP: multiple objects sent in multipart message

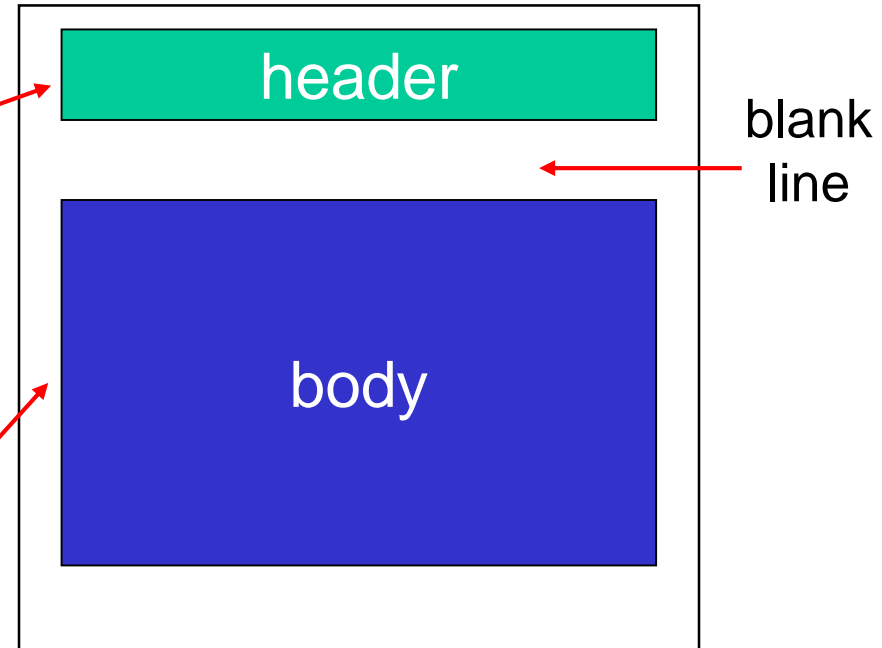
# Mail Message Format

SMTP: protocol for exchanging email messages

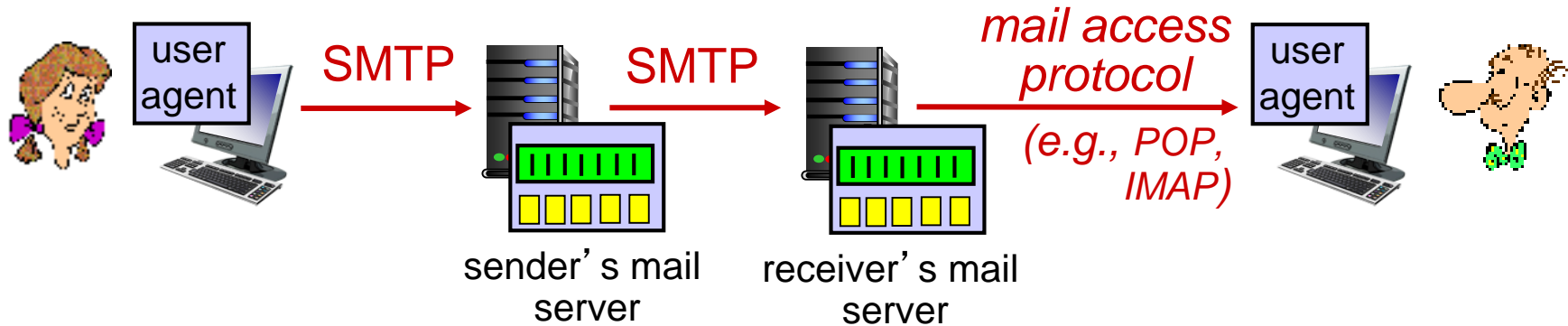
RFC 822: standard for text message format:

- ❖ header lines, e.g.,
  - To:
  - From:
  - Subject:

*different* from SMTP MAIL FROM, RCPT TO: commands!
- ❖ Body: the “message”
  - ASCII characters only



# Mail Access Protocol



- **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**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

## *transaction phase, client:*

- ❖ **list**: list message numbers
- ❖ **retr**: retrieve message by number
- ❖ **dele**: delete
- ❖ **quit**

```
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



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

# Domain Name System (DNS)

*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*

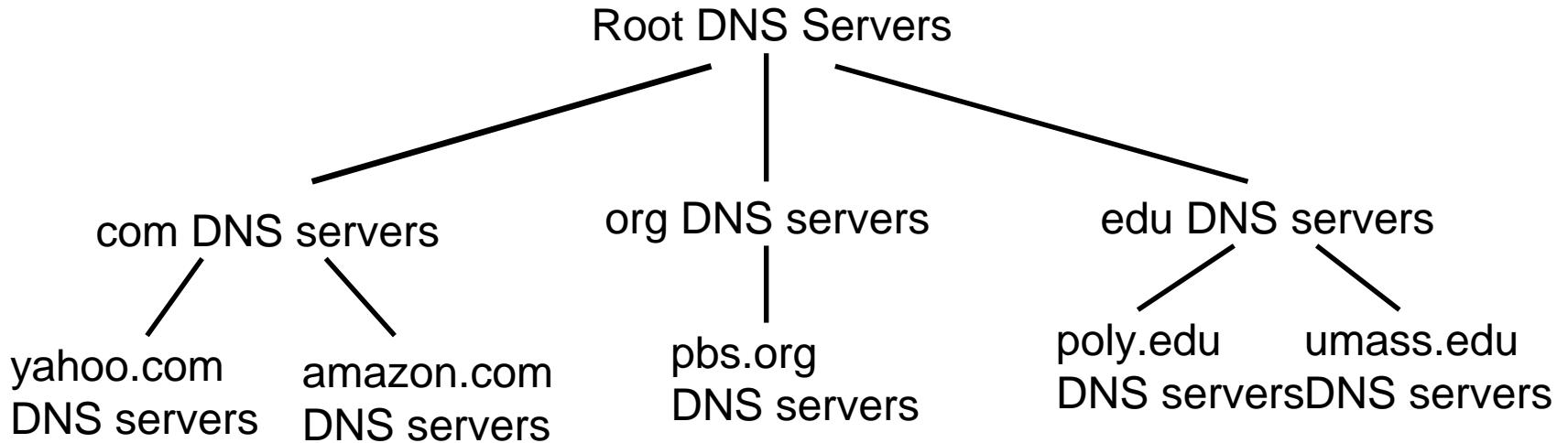
- ❖ hostname to IP address translation
- host aliasing
  - canonical, alias names
- ❖ mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

## *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

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)

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

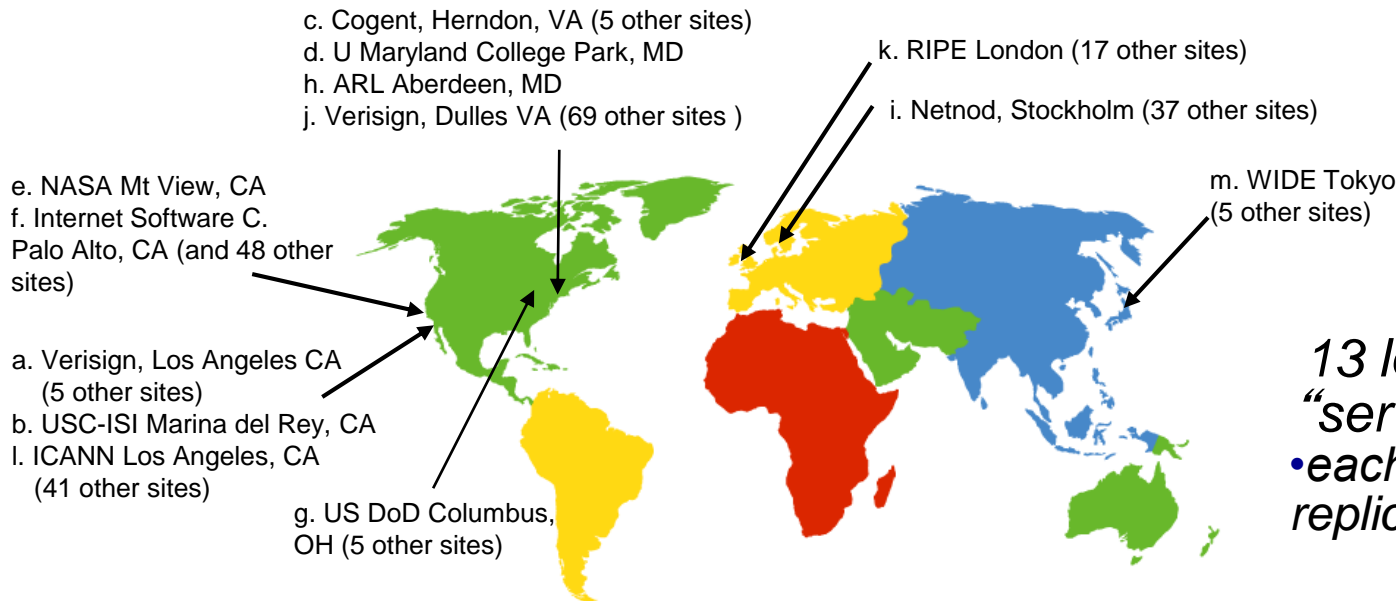


m. WIDE Tokyo  
(5 other sites)

*13 root name  
“servers”  
worldwide*

# 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 logical root name  
“servers” worldwide*  
• *each “server”  
replicated many times*

## *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

---

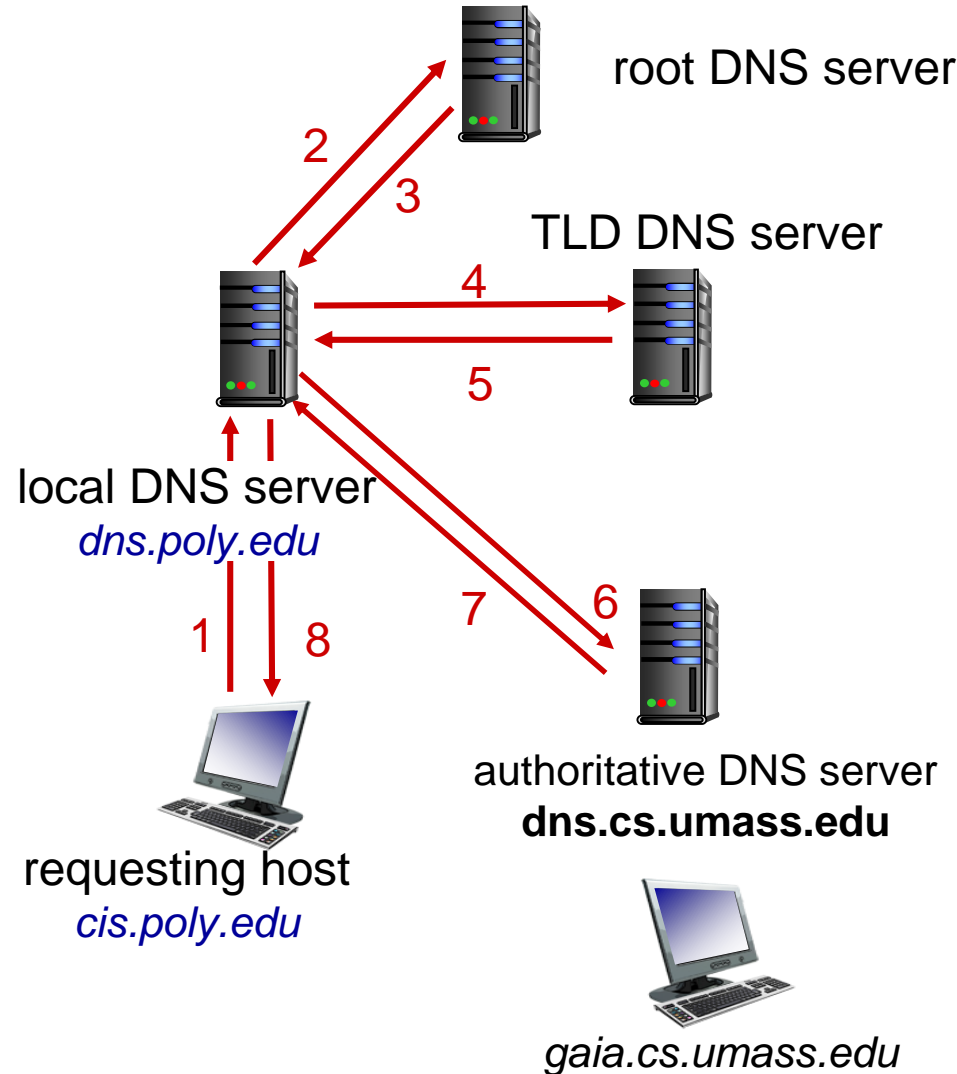
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called “default name server”
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts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

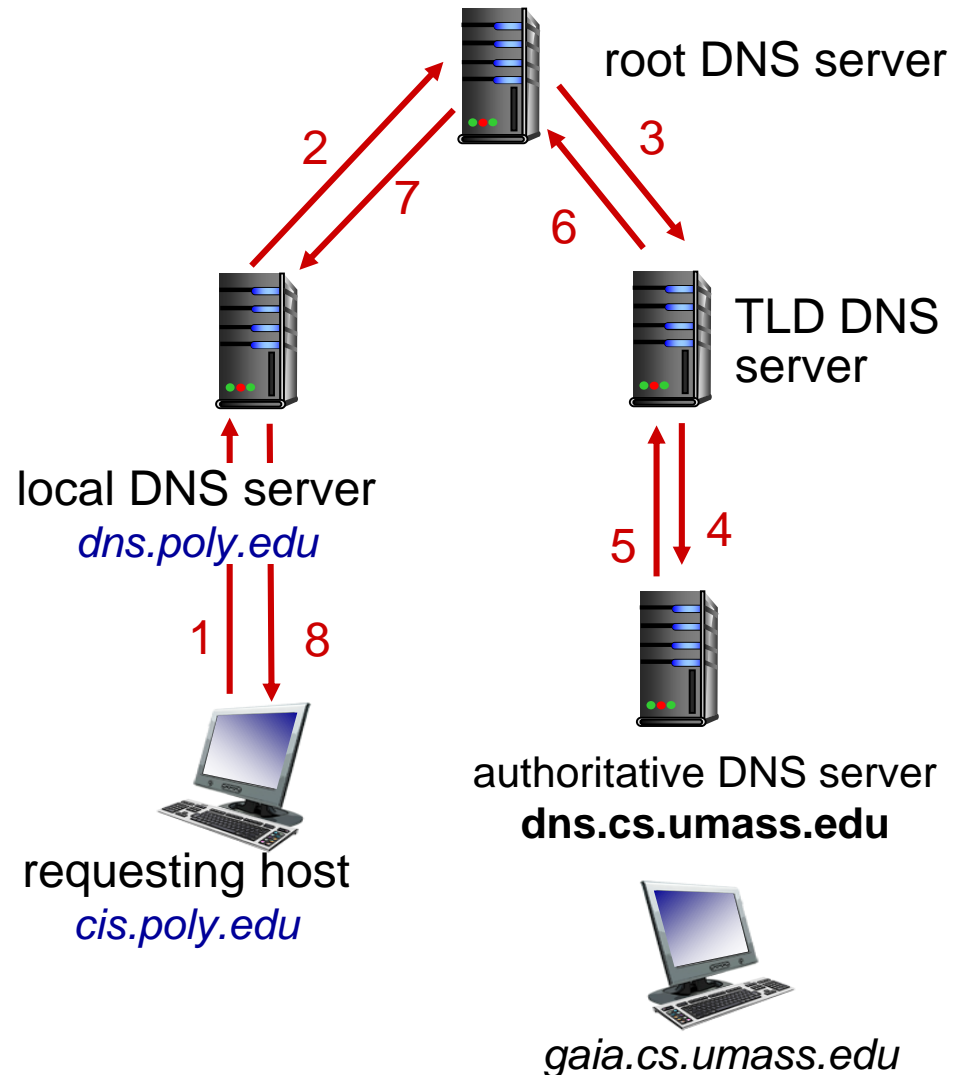
- contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”



# DNS Name Resolution Example

## *recursive query:*

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



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

**DNS:** 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

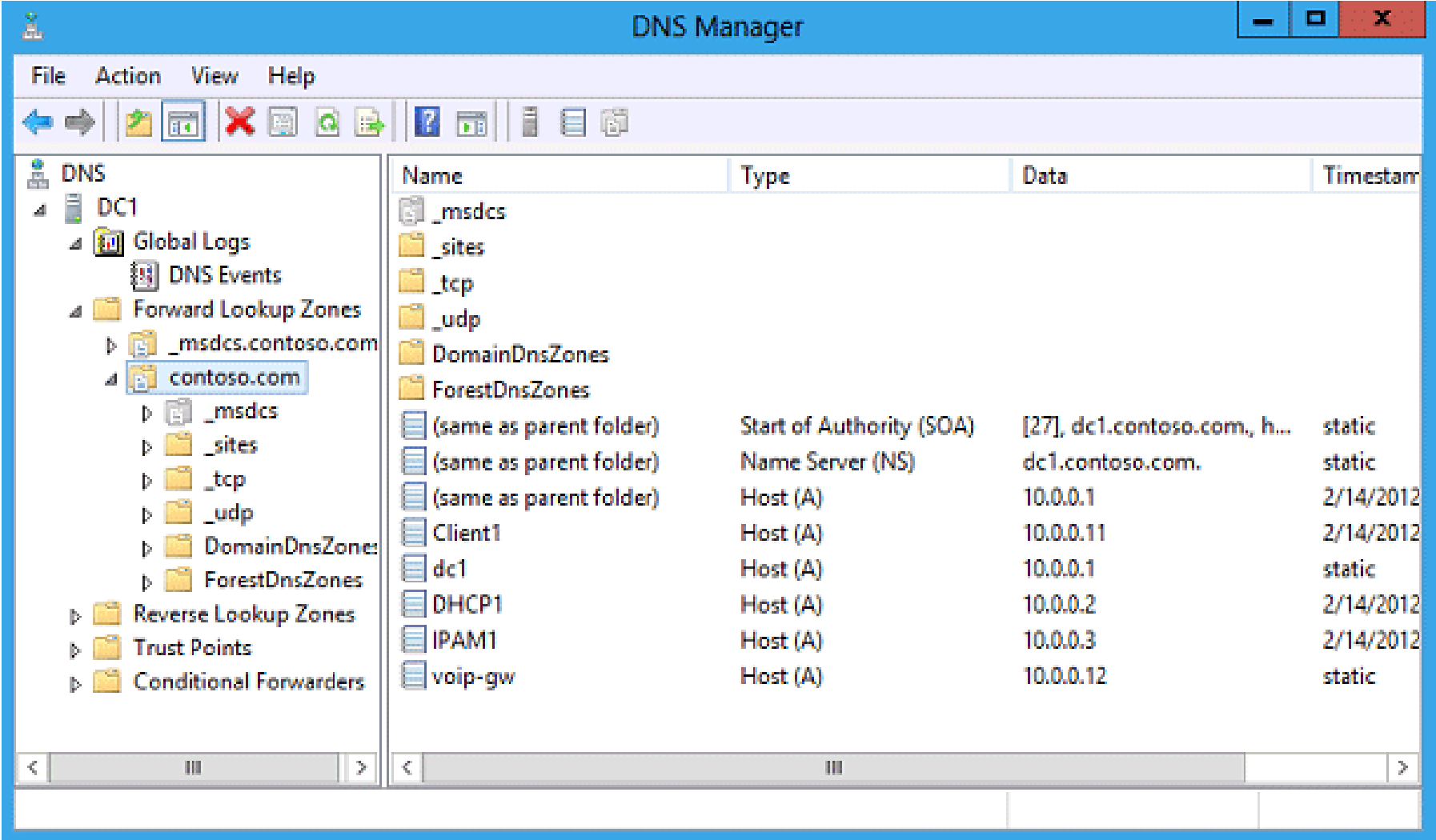
## type=CNAME

- **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 Manager Screenshot



The screenshot shows the DNS Manager console window. The left pane displays a tree view of the DNS hierarchy, with the 'contoso.com' zone selected under 'Forward Lookup Zones'. The right pane shows a table of DNS records for this zone.

Name	Type	Data	Timestamp
_msdcs			
_sites			
_tcp			
_udp			
DomainDnsZones			
ForestDnsZones			
(same as parent folder)	Start of Authority (SOA)	[27], dc1.contoso.com., h...	static
(same as parent folder)	Name Server (NS)	dc1.contoso.com.	static
(same as parent folder)	Host (A)	10.0.0.1	2/14/2012
Client1	Host (A)	10.0.0.11	2/14/2012
dc1	Host (A)	10.0.0.1	static
DHCP1	Host (A)	10.0.0.2	2/14/2012
IPAM1	Host (A)	10.0.0.3	2/14/2012
voip-gw	Host (A)	10.0.0.12	static

# DNS Protocol and Messages

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

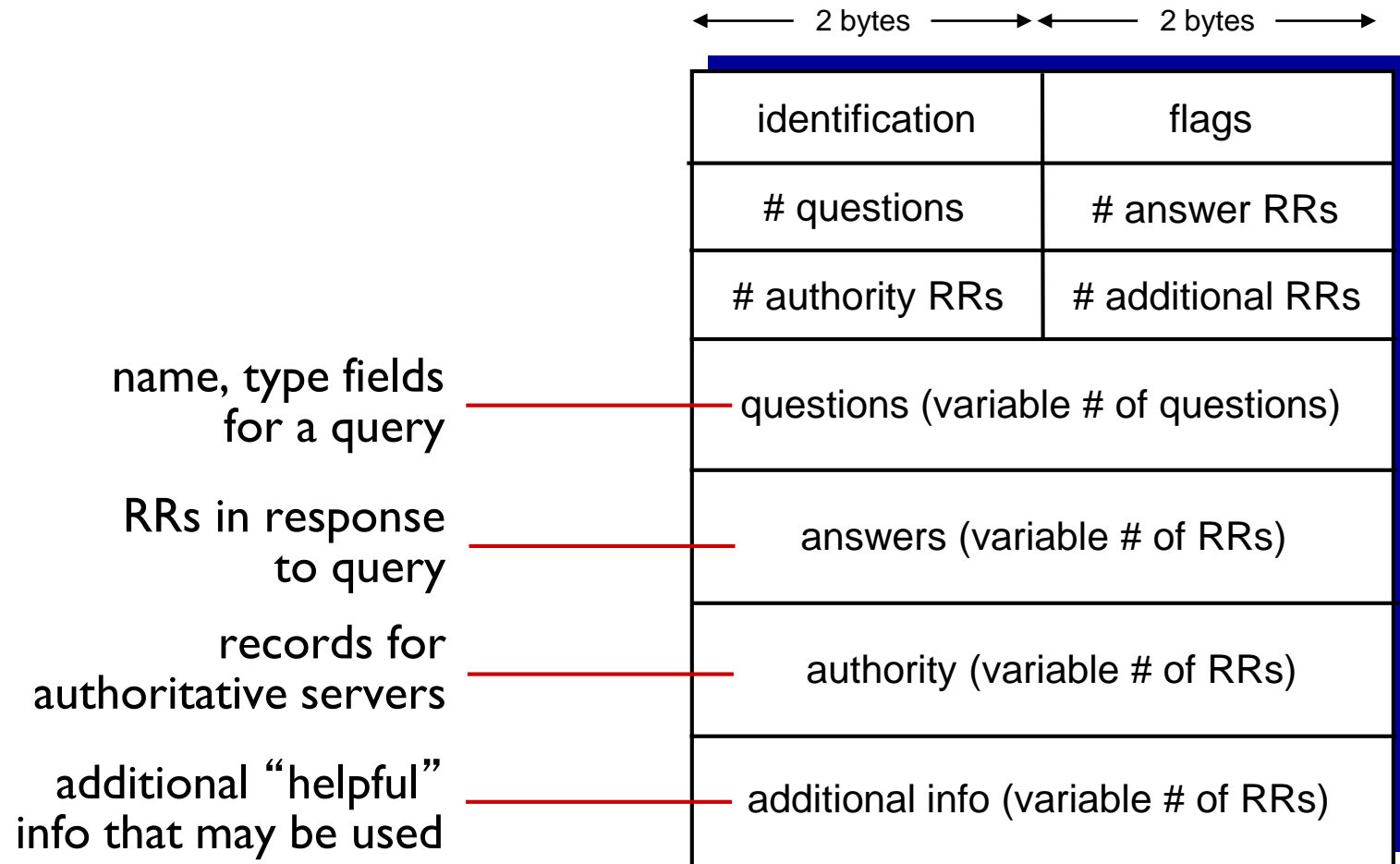
## message header

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

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 Protocol and Messages (cont.)



# 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

## DDoS attacks

- Bombard root servers with traffic
  - Not successful to date
  - Traffic Filtering
  - Local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombard TLD servers
  - Potentially more dangerous

## Redirect attacks

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

## Exploit DNS for DDoS

- Send queries with spoofed source address: target IP
- Requires amplification

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

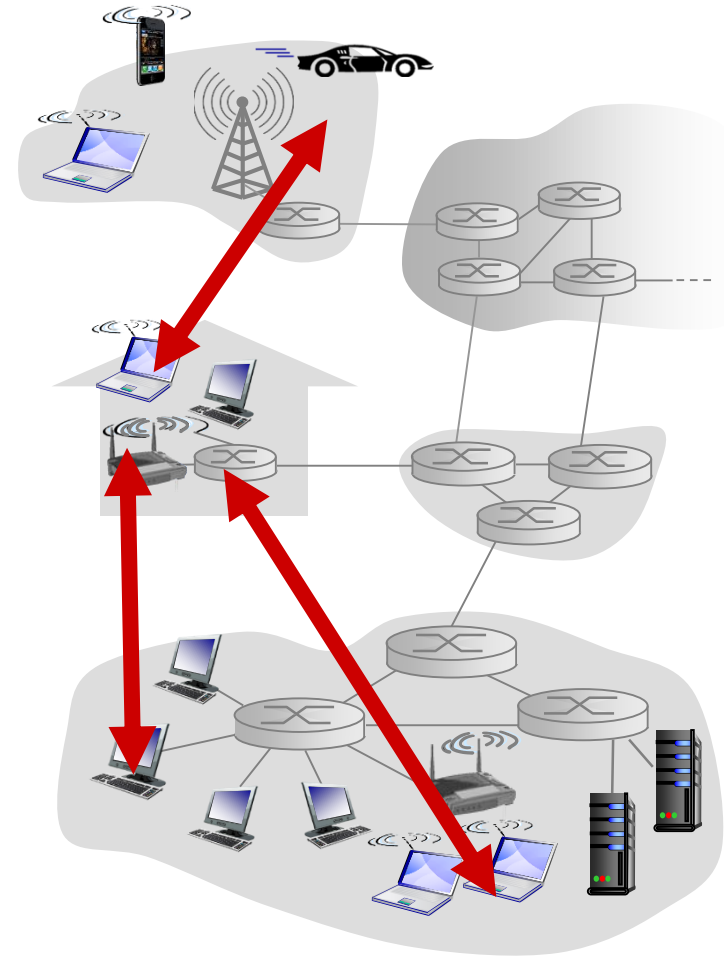
2.7 socket programming with UDP and TCP

# Pure P2P Architecture

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

## *examples:*

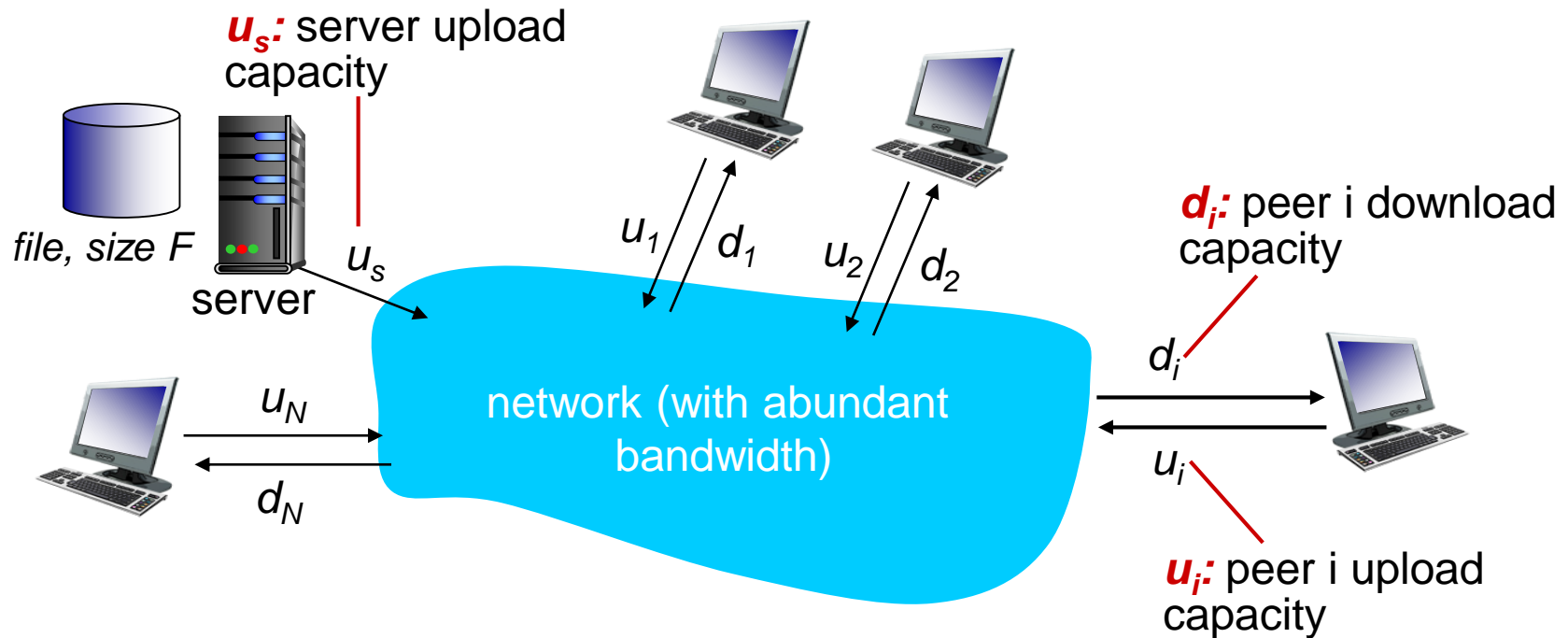
- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)



# File Distribution: Client/Server vs P2P

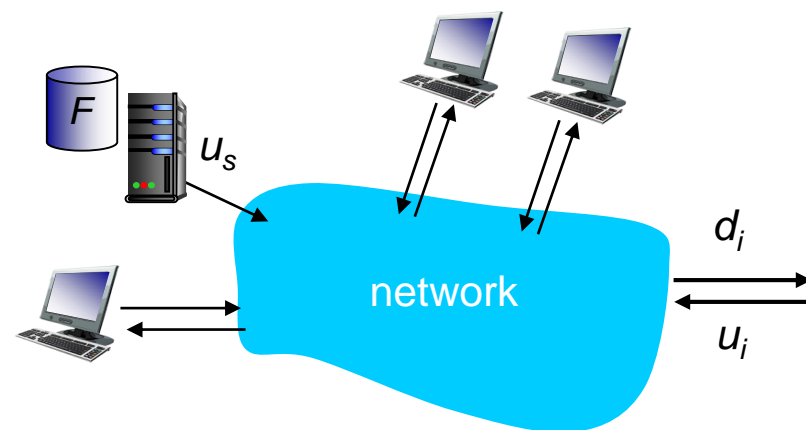
**Question:** how much time to distribute file (size  $F$ ) from one server to  $N$  peers?

- 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*

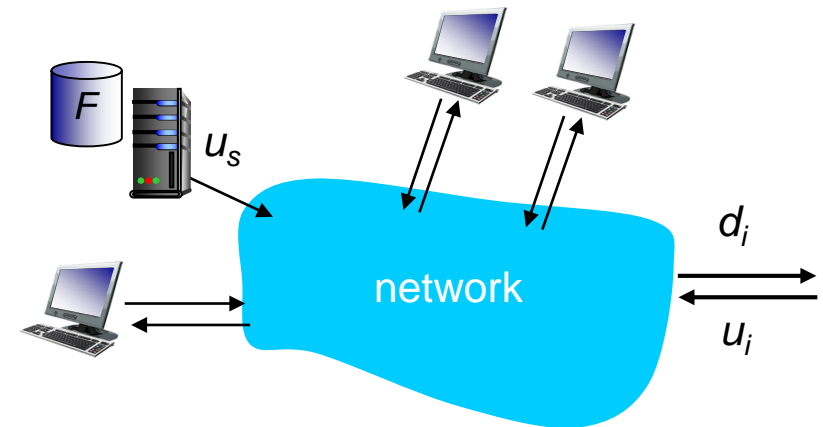
$$D_{c-s} \geq \max\{NF/u_s, F/d_{\min}\}$$

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



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

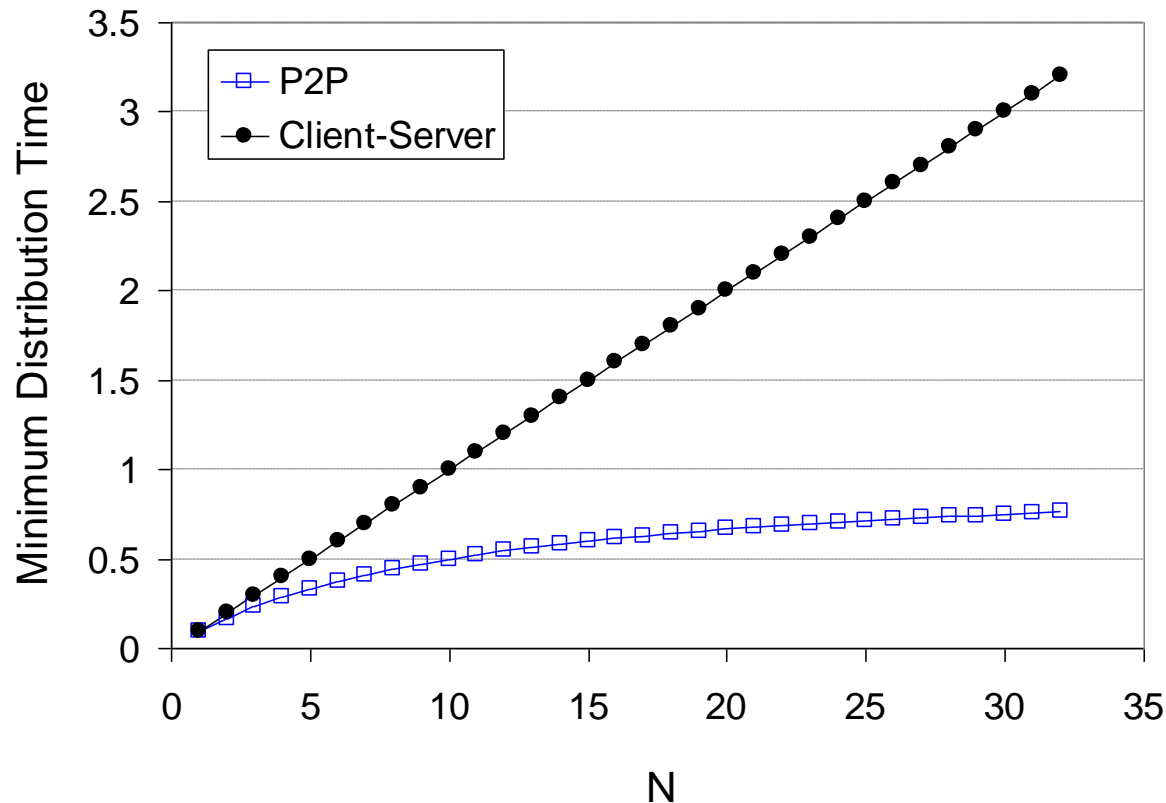
$$D_{P2P} \geq \max\{F/u_s, F/d_{\min}, NF/(u_s + \sum u_i)\}$$

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,  $u_s = 10u$ ,  $d_{min} \geq u_s$

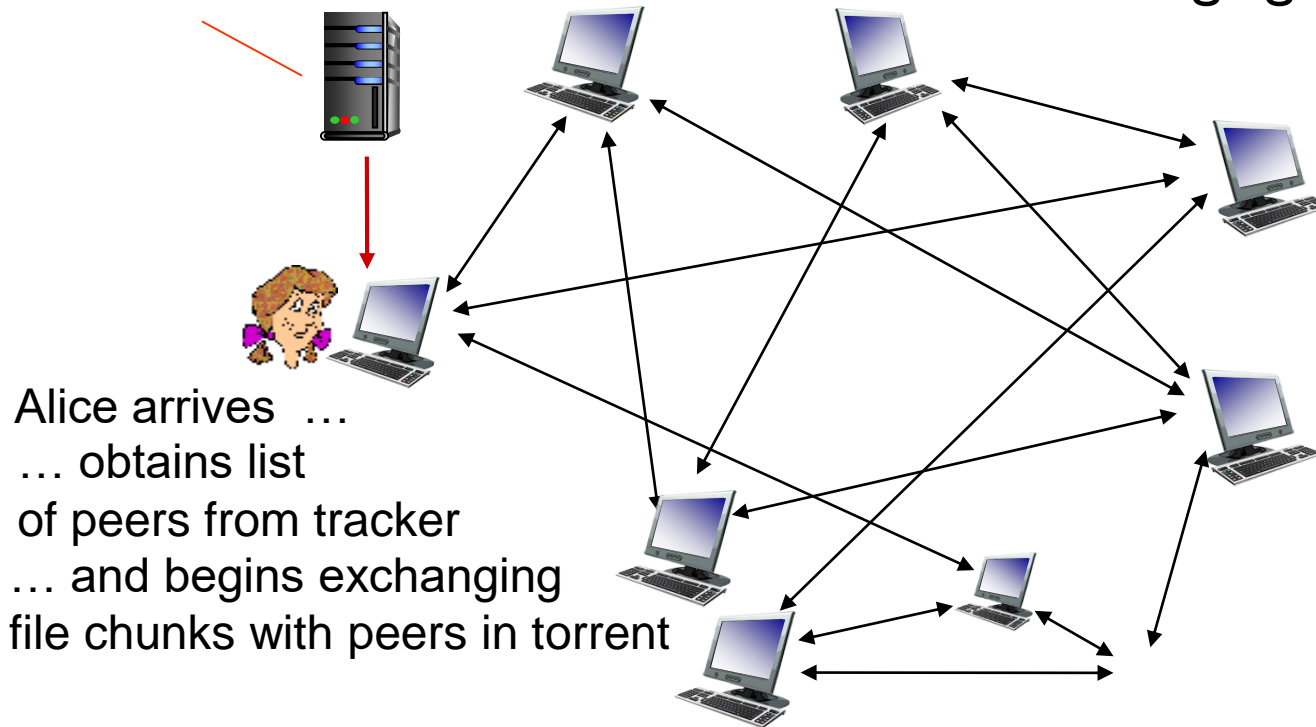


# P2P file distribution: BitTorrent

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

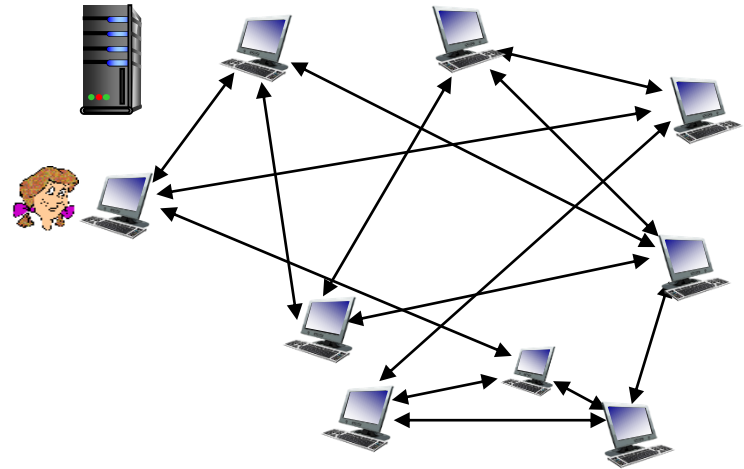
**tracker:** tracks peers participating in torrent

**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



## *requesting chunks:*

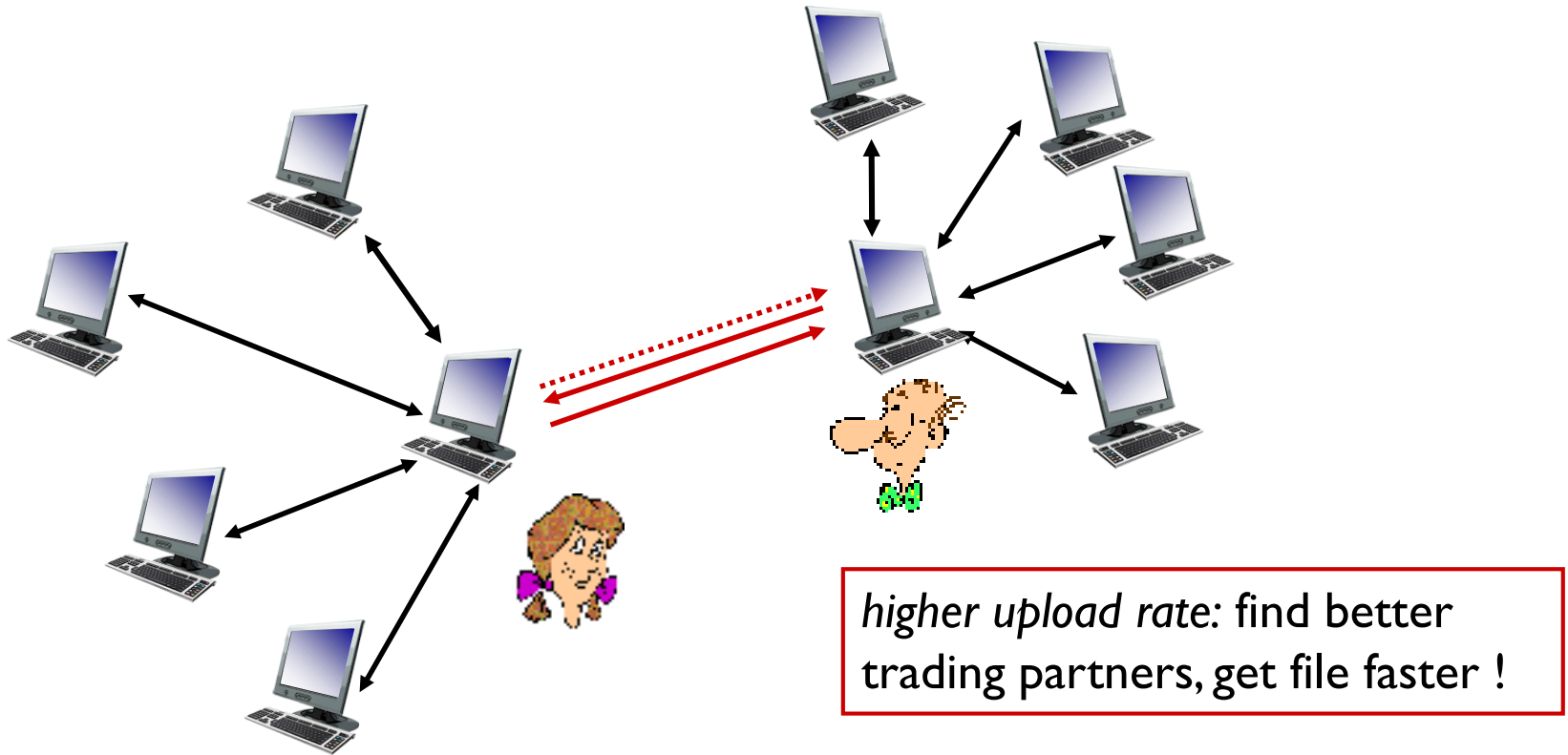
- ❖ at any given time, different peers have different subsets of file chunks
- ❖ periodically, Alice asks each peer for list of chunks that they have
- ❖ Alice requests missing chunks from peers, rarest first

## *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





**QUESTIONS**

**now**