# Τεχνολογίες Διαδικτύου 2025-26 (DIT 315)

Δρ. Ειρήνη Λιώτου

eliotou@hua.gr

4/11/2025

# 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

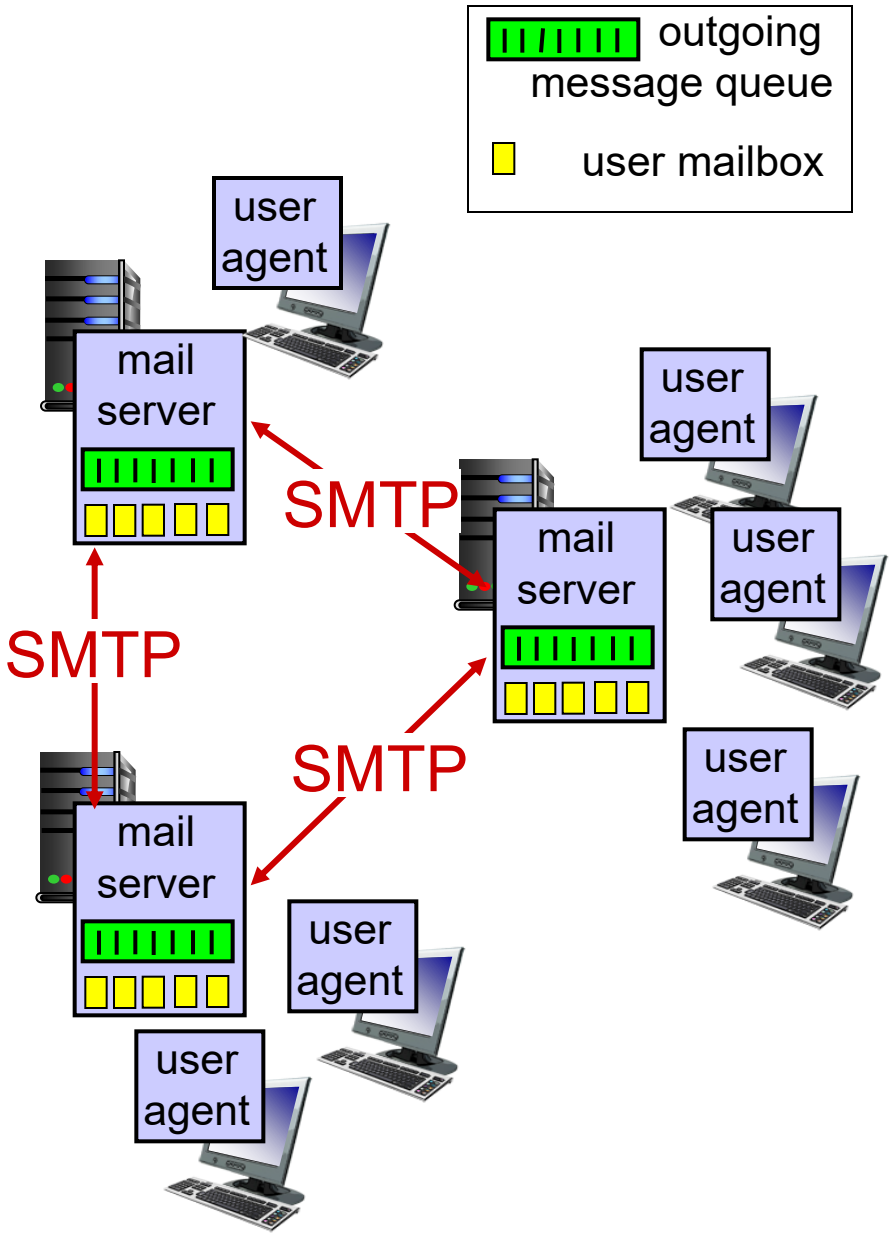
# Electronic mail

*Three major components:*

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

*User Agent*

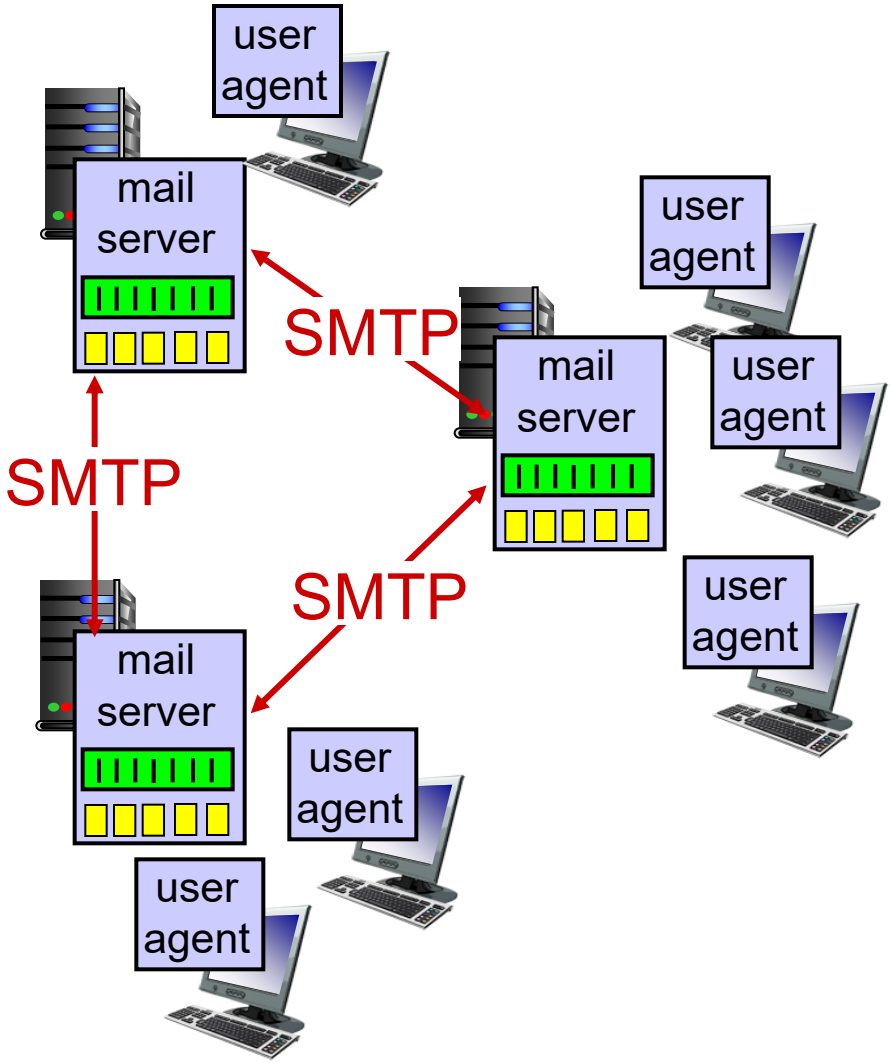
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



# Electronic mail: mail servers

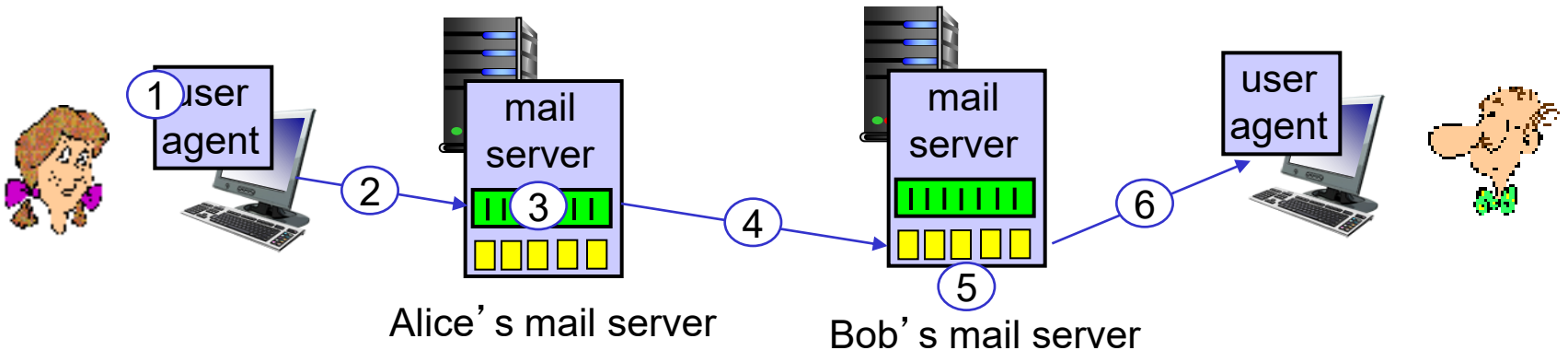
mail servers:

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



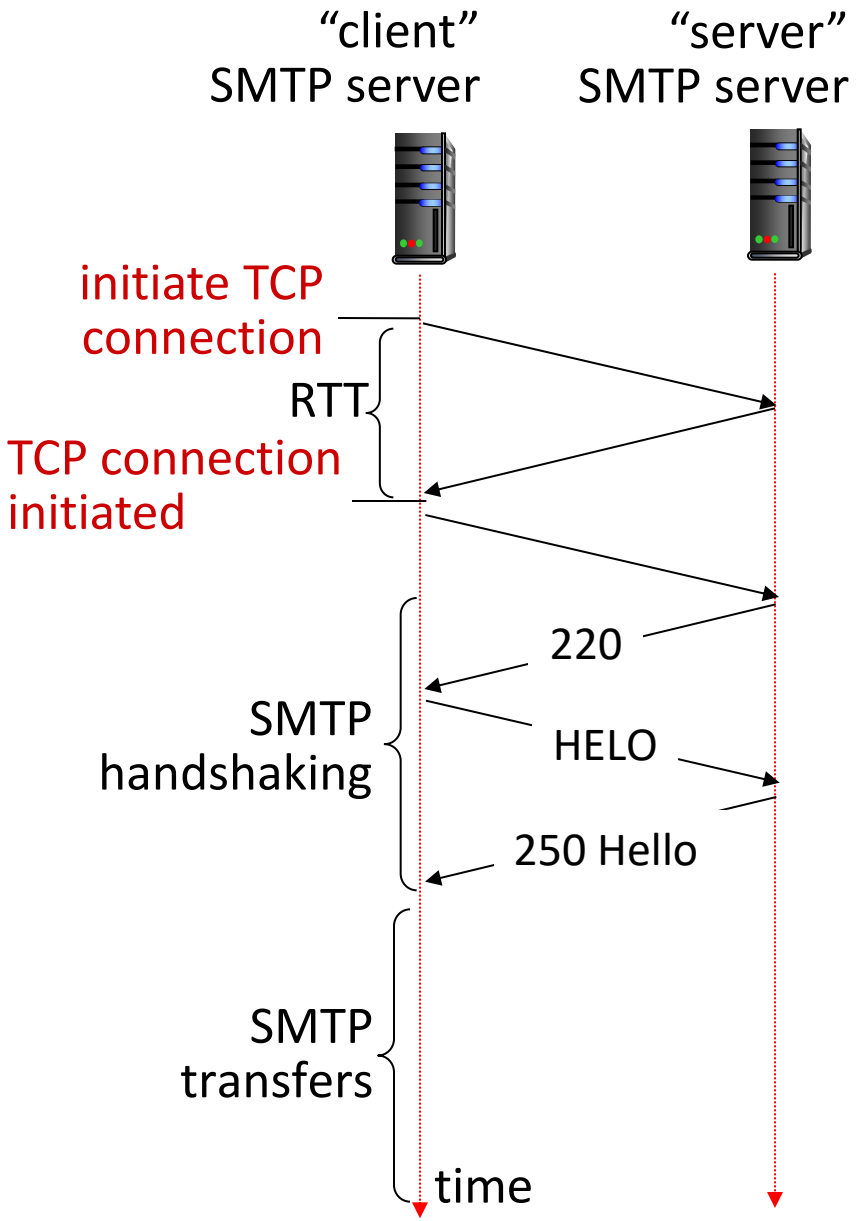
# Scenario: Alice sends message to Bob

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 RFC (5321)

- uses TCP to reliably transfer email message from client (mail server initiating connection) to server, port 25
  - direct transfer: sending server (acting like client) to receiving server
- three phases of transfer
  - SMTP handshaking (greeting)
  - SMTP transfer of messages
  - SMTP closure
- command/response interaction (like HTTP)
  - commands: ASCII text
  - response: status code and phrase



# 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
```

# SMTP: final words

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP client 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

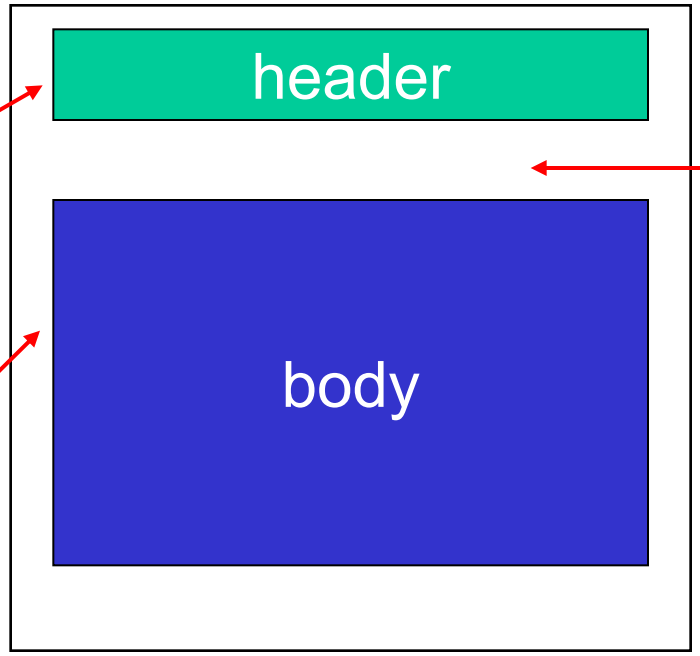
# Mail message format

SMTP: protocol for exchanging email messages

RFC 822: standard for text message format:

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

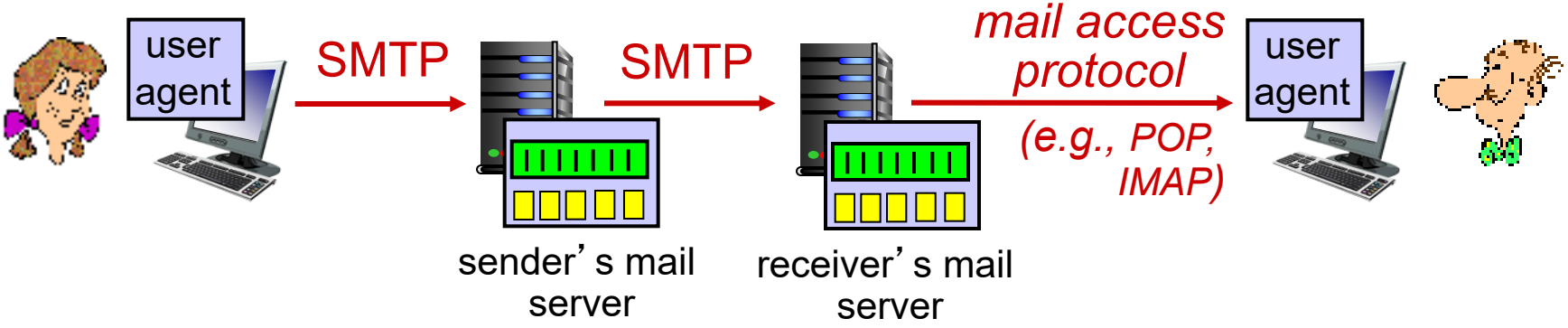
  *different from* SMTP MAIL FROM, RCPT TO: commands!
- Body: the "message"
  - ASCII characters only




```
From: someone@example.com
To: someone_else@example.com
Subject: An RFC 822 formatted message

This is the plain text body of the message. Note the blank line
between the header information and the body of the message.
```

# Mail access protocols




- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
  - POP3: Post Office Protocol [RFC 1939]: authorization, download
  - IMAP: Internet Mail Access Protocol [RFC 3501]: messages stored on server, IMAP provides retrieval, deletion, folders of stored messages on server
  - HTTP: Gmail, Hotmail, Yahoo!Mail, etc. provides web-based interface on top of STMP (to send), IMAP (or POP) to retrieve e-mail messages

# GMAIL

| Phase | From | To | Protocol |
| --- | --- | --- | --- |
| Compose & send | Sender's browser | Gmail web server | **HTTPS** |
| Server-to-server | Gmail's SMTP server | Recipient's mail server | **SMTP** |
| Read (if Gmail user) | Recipient's browser | Gmail web server | **HTTPS** |

# 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: domain name system

- Request the URL *www.someschool.edu/index.html*

1. The same 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
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

***The DNS protocol runs over UDP and uses port 53.***

*why not centralize DNS?*

- single point of failure
- traffic volume
- distant centralized database
- Maintenance - updates

*A: doesn't scale!*

- Comcast DNS servers alone: 600B DNS queries/day
- Akamai DNS servers alone: 2.2T DNS queries/day

# Thinking about the DNS

humongous distributed database:

- ~ billion records, each simple

handles many *trillions* of queries/day:

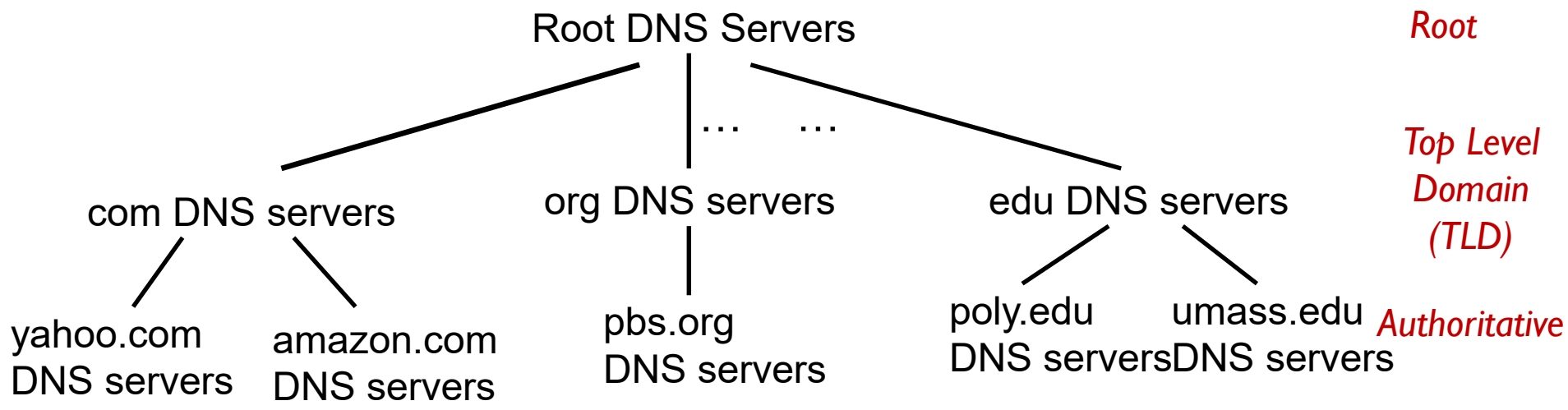
- *many* more reads than writes
- *performance matters:* almost every Internet transaction interacts with DNS - msecs count!

organizationally, physically decentralized:

- millions of different organizations responsible for their records

"bulletproof": reliability, security

# DNS: a distributed, hierarchical database

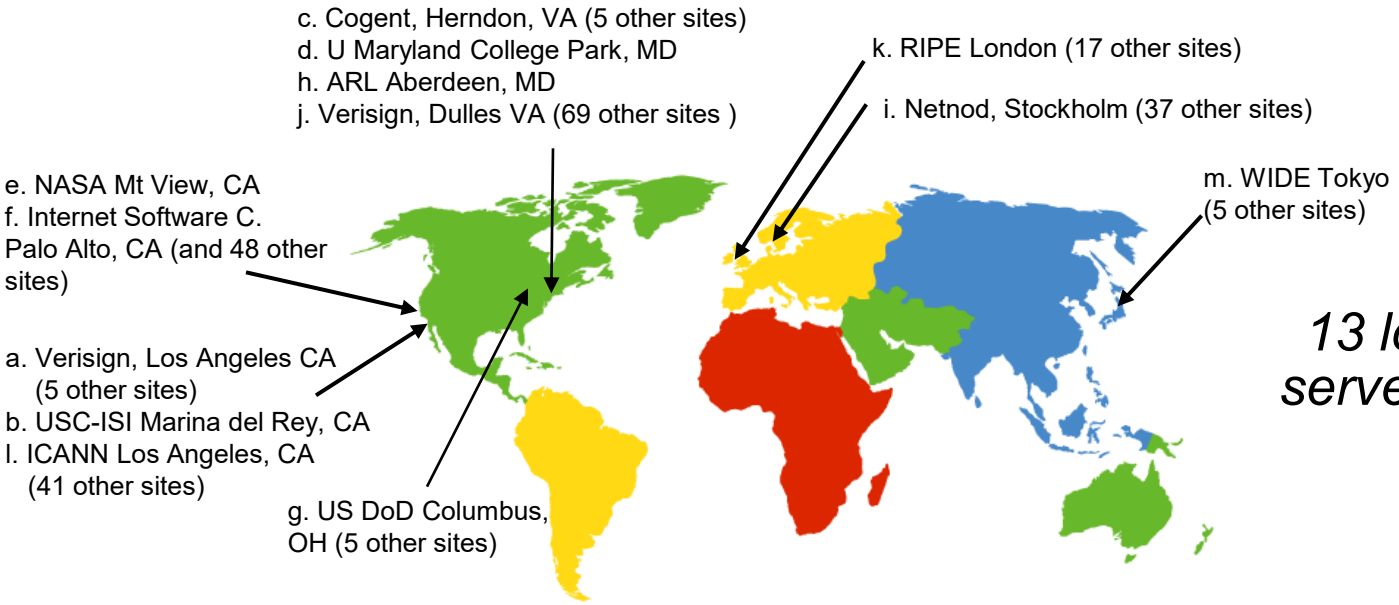



*client wants IP for www.amazon.com; 1st approximation:*

- 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

- official, contact-of-last-resort by name servers that cannot resolve name
- *incredibly important* Internet function
  - Internet couldn't function without it!




*13 logical root name servers' organizations 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

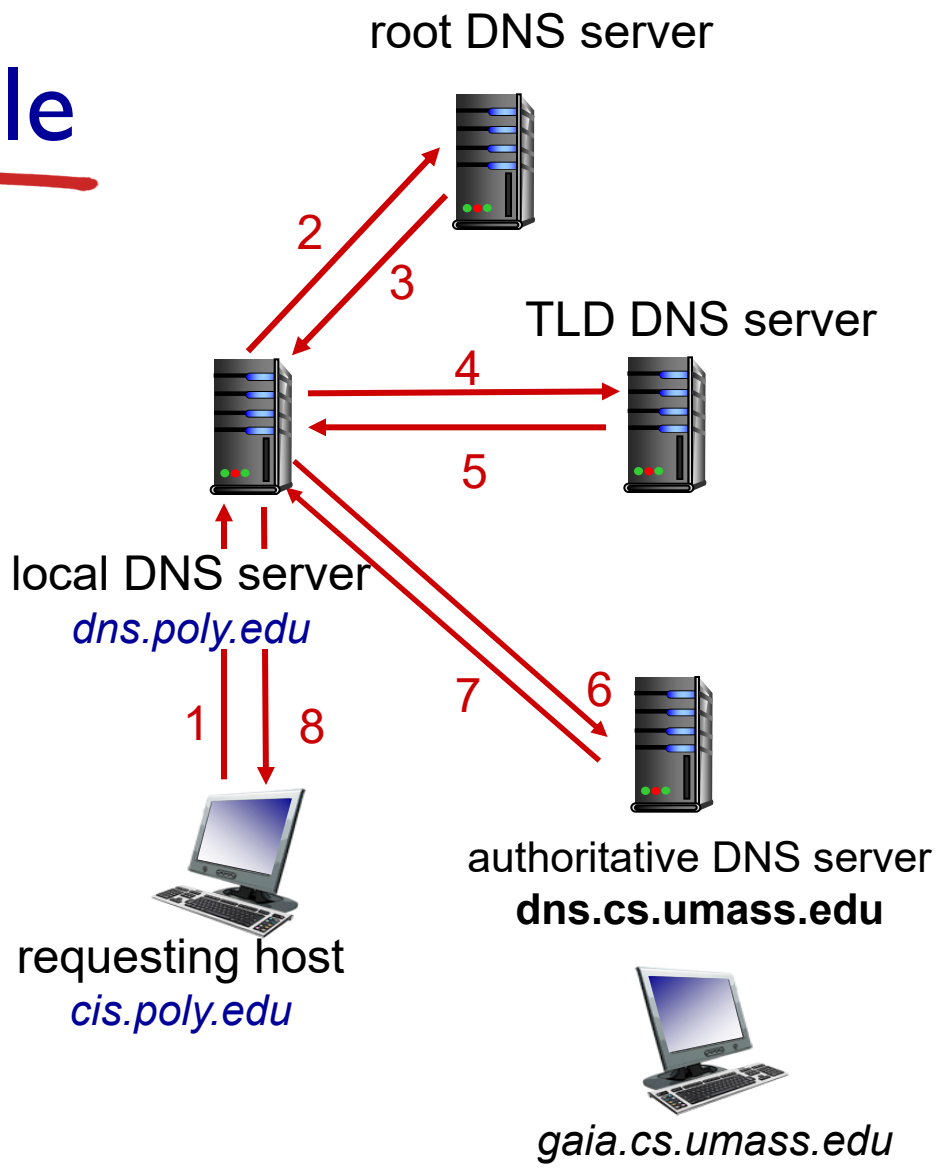
- does not strictly belong to previous hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy
- each ISP has local DNS name server; to find yours:
    - MacOS: % scutil --dns
    - Windows: >ipconfig /all

# DNS name resolution example

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

*Iterated query:*

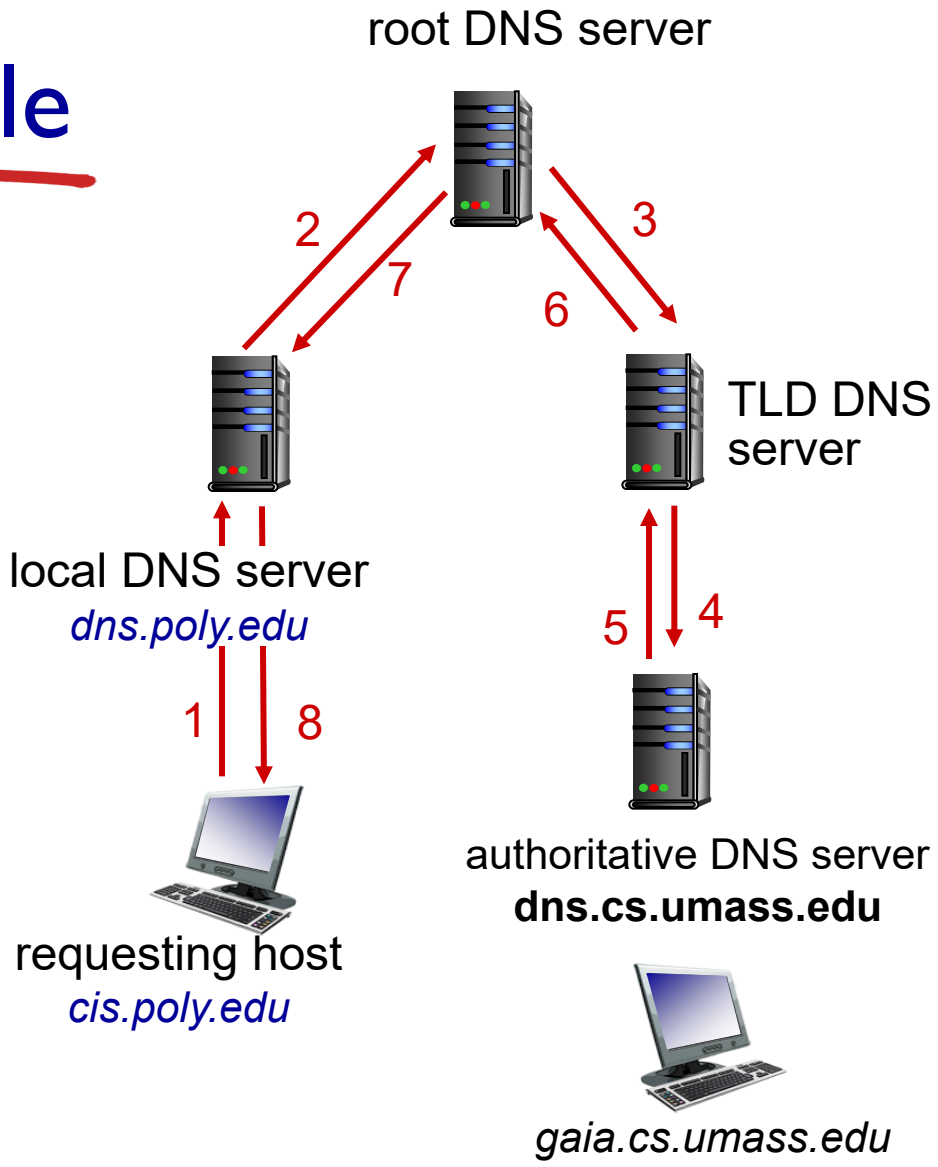
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



# DNS name resolution example

*Recursive query:*

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



# DNS: caching, updating records

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

# DNS records

*DNS:* distributed database 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

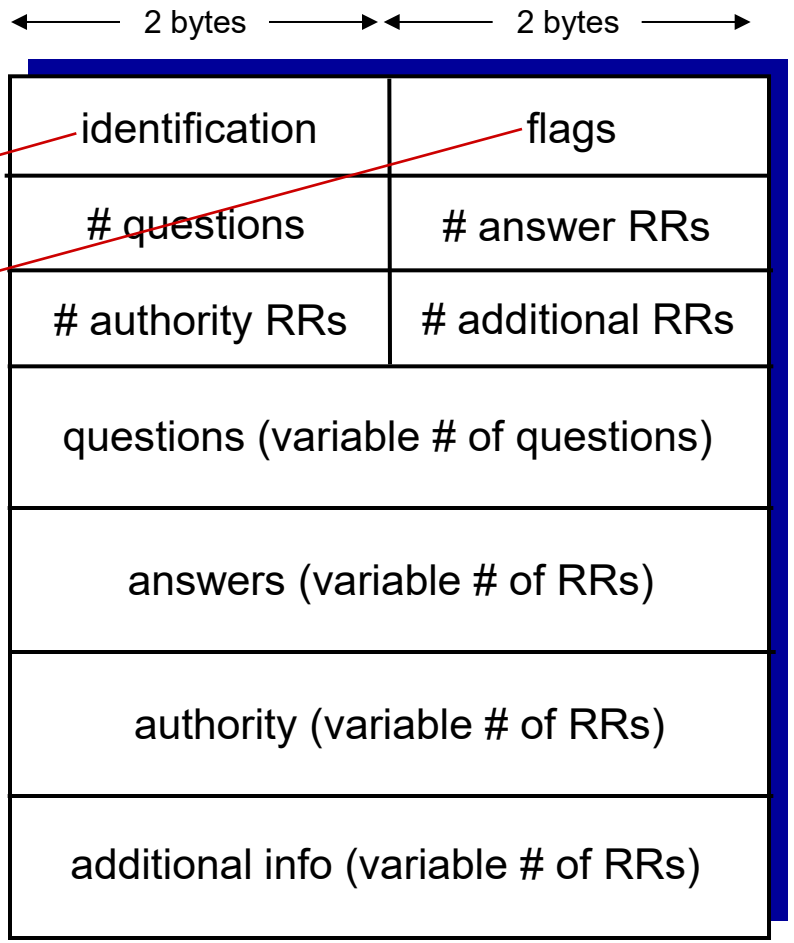
- **`value`** is name of mailserver associated with **`name`**

# DNS protocol, messages

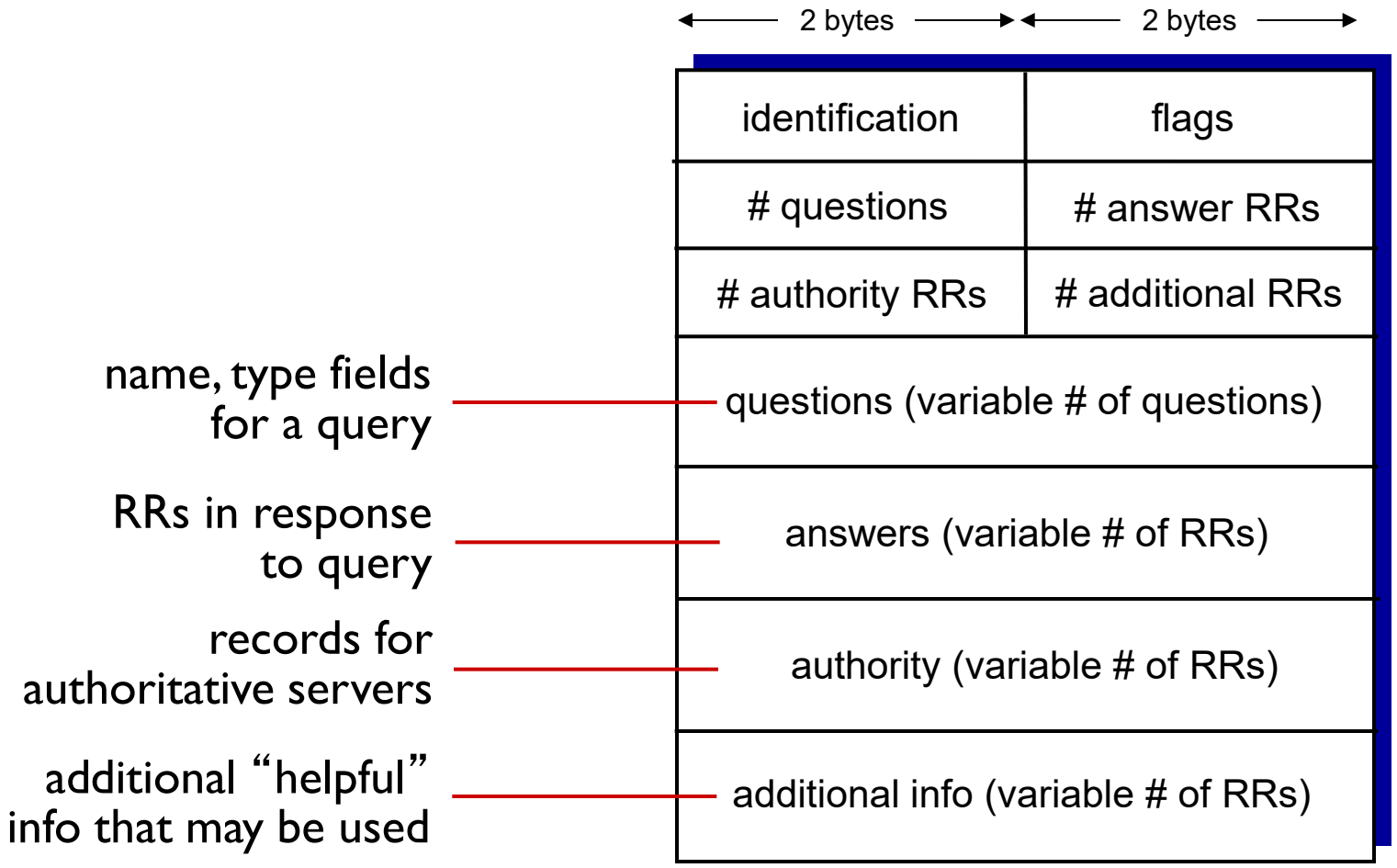
- *query* and *reply* messages, both with same *message format*

message 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) at TLD
  - 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

# Attacking DNS

## 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