Cookies Overview and HTTP Proxies

What is a Cookie?

- Small piece of data generated by a web server, stored on the client's hard drive.
- Serves as an add-on to the HTTP specification (remember, HTTP by itself is stateless.)
- Still somewhat controversial, as it enables web sites to track web users and their habits…

Why use Cookies?

- Tracking unique visitors
- Creating personalized web sites
- Shopping Carts
- Tracking users across your site:
  - e.g. do users that visit your sports news page also visit your sports store?

Example Cookie Use

- Website wants to track the number of unique visitors who access its site.
- If the website checks the HTTP Server logs, it can determine the number of “hits”, but cannot determine the number of unique visitors.
- That’s because HTTP is stateless. It retains no memory regarding individual users.
- Cookies provide a mechanism to solve this problem.

Tracking Unique Visitors

- Step 1: Person A requests the website.
- Step 2: Web Server generates a new unique ID.
- Step 3: Server returns home page plus a cookie set to the unique ID.
- Step 4: Each time Person A returns to the website, the browser automatically sends the cookie along with the GET request.

Cookie Notes

- Created in 1994 for Netscape 1.1
- Cookies cannot be larger than 4K
- No domain (e.g. netscape.com, microsoft.com) can have more than 20 cookies.
- Cookies stay on your machine until:
  - they automatically expire
  - they are explicitly deleted
- Cookies work the same on all browsers.

Cookie Standards

- Version 0 (Netscape):
  - The original cookie specification
  - Implemented by all browsers and servers
  - We will focus on this Version
- Version 1
  - A proposed standard of the Internet Engineering Task Force (IETF)
  - Not very widely used (hence, we will stick to Version 0.)

Cookie Anatomy

- Version 0 specifies six cookie parts:
  - Name
  - Value
  - Domain
  - Path
  - Expires
  - Secure

Cookie Parts: Name/Value

- Name
  - Name of your cookie (Required)
  - Cannot contain white spaces, semicolons or commas.
- Value
  - Value of your cookie (Required)
  - Cannot contain white spaces, semicolons or commas.

Cookie Parts: Domain

- Only pages from the domain which created a cookie are allowed to read the cookie.
- For example, amazon.com cannot read yahoo.com's cookies (imagine the security flaws if this were otherwise!)
- By default, the domain is set to the full domain of the web server that served the web page.
  - For example, myserver.mydomain.com would automatically set the domain to .myserver.mydomain.com

Cookie Parts: Domain

- Note that domains are always prepended with a dot.
  - This is a security precaution: all domains must have at least two periods.
- You can however, set a higher level domain
  - For example, myserver.mydomain.com can set the domain to .mydomain.com. This way hisserver.mydomain.com and herserver.mydomain.com can all access the same cookies.
- No matter what, you cannot set a domain other than your own.

Cookie Parts: Path

- Restricts cookie usage within the site.
- By default, the path is set to the path of the page that created the cookie.
- Example: user requests page from mymall.com/storea. By default, cookie will only be returned to pages for or under /storea.
- If you specify the path to / the cookie will be returned to all pages (a common practice.)

Cookie Parts: Expires

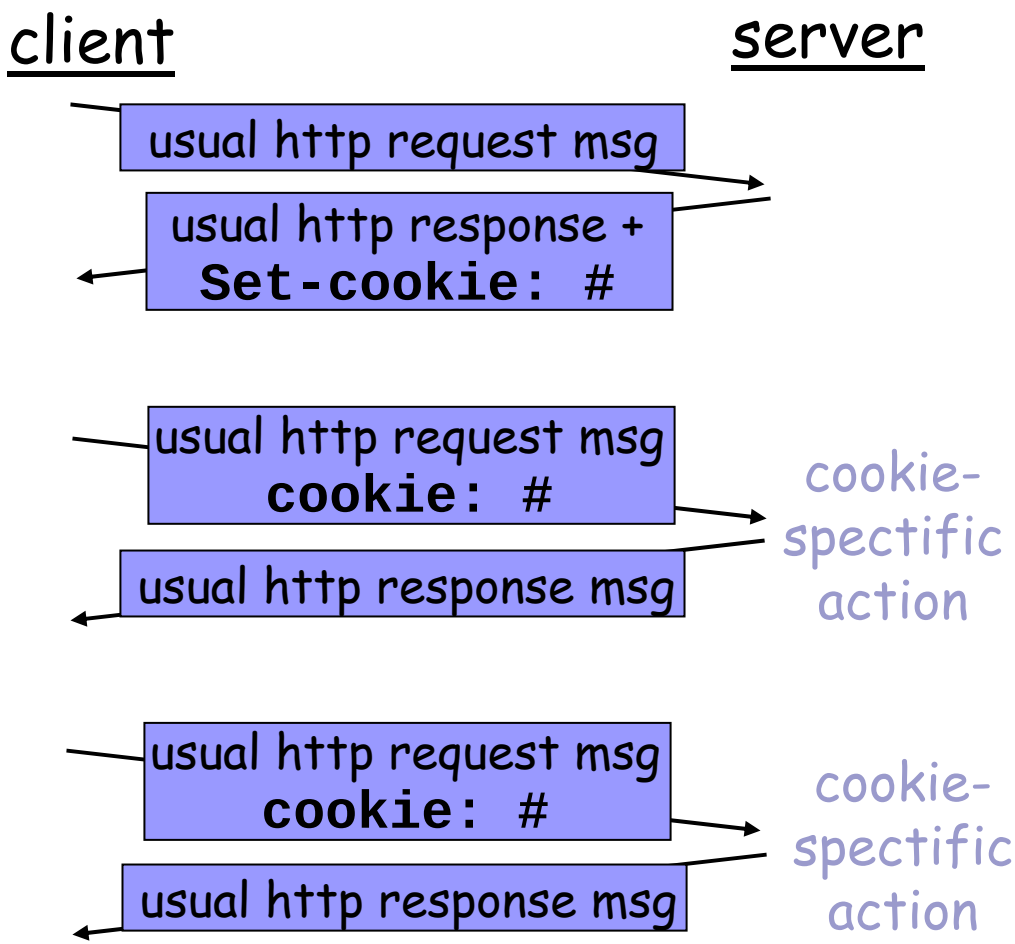
- Specifies when the cookie will expire.
- Specified in Greenwich Mean Time (GMT):
  - Wdy DD-Mon-YYYY HH:MM:SS GMT
- If you leave this value blank, browser will delete the cookie when the user exits the browser.
  - This is known as a *session cookies*, as opposed to a *persistent cookie*.

Cookie Parts: Secure

- The secure flag is designed to encrypt cookies while in transit.
- A secure cookie will only be sent over a secure connection (such as SSL.)
- In other words, if a cookie is set to secure, and you only connect via a non-secure connection, the cookie will not be sent.

User-server interaction: cookies

- server sends "cookie" to client in response msg
  `Set-cookie: 1678453`
- client stores & presents cookie in later requests
  `cookie: 1678453`
- server matches presented-cookie with server-stored info
  - authentication
  - remembering user preferences, previous choices



Cookie example

telnet www.google.com 80

Trying 216.239.33.99...
Connected to www.google.com.
Escape character is '^]'.

GET /index.html HTTP/1.0

HTTP/1.0 200 OK
Date: Wed, 10 Sep 2003 08:58:55 GMT
Set-Cookie:
PREF=ID=43bd8b0f34818b58:TM=1063184203:LM=1063184203:S
=DDqPgTb56Za88O2y; expires=Sun, 17-Jan-2038 19:14:07 GMT;
path=/; domain=.google.com

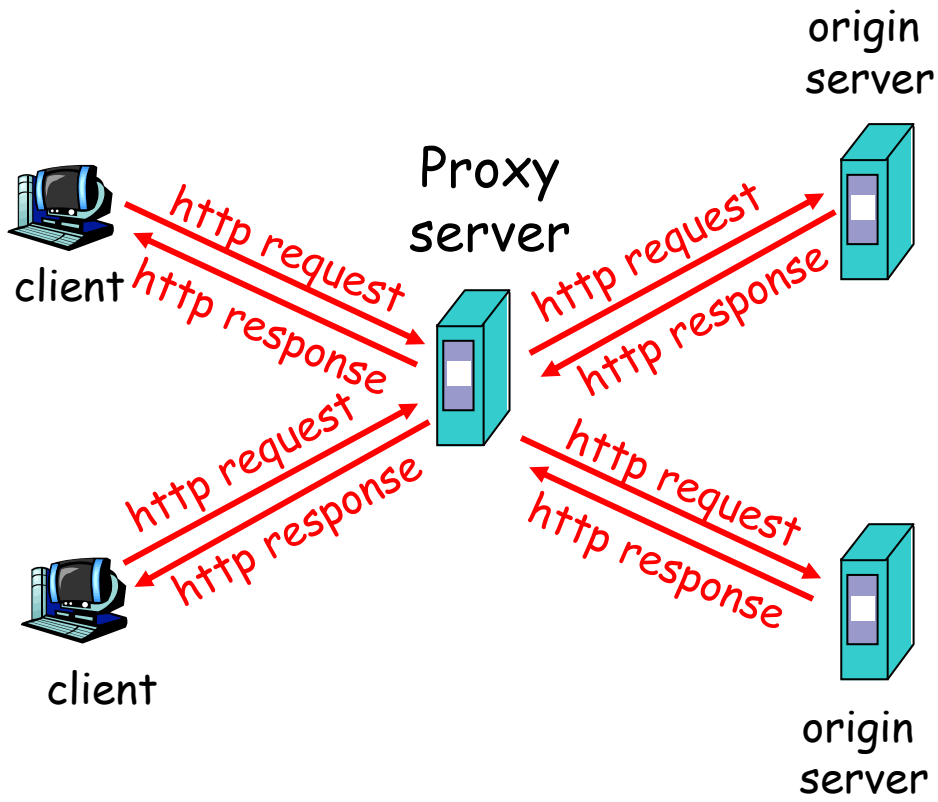
.

.

Web Caches (proxy server)

Goal: satisfy client request without involving origin server

- user sets browser: Web accesses via web cache
- client sends all http requests to web cache
  - if object at web cache, web cache immediately returns object in http response
  - else requests object from origin server, then returns http response to client



More about Web caching

- Cache acts as both client and server
- Cache can do up-to-date check using
  `If-modified-since`
  HTTP header
  - Issue: should cache take risk and deliver cached object without checking?
  - Heuristics are used.
- 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.
- Internet dense with caches enables "poor" content providers to effectively deliver content

Note: Meta tags and http-equiv

- HTTP servers use the property name specified by the http-equiv attribute to create an [RFC822]-style header in the HTTP response.

- The following sample META declaration:

<META http-equiv="Expires" content="Tue, 20 Aug 1996 14:25:27 GMT">

will result in the HTTP header:

Expires: Tue, 20 Aug 1996 14:25:27 GMT

This can be used by caches to determine when to fetch a fresh copy of the associated document.

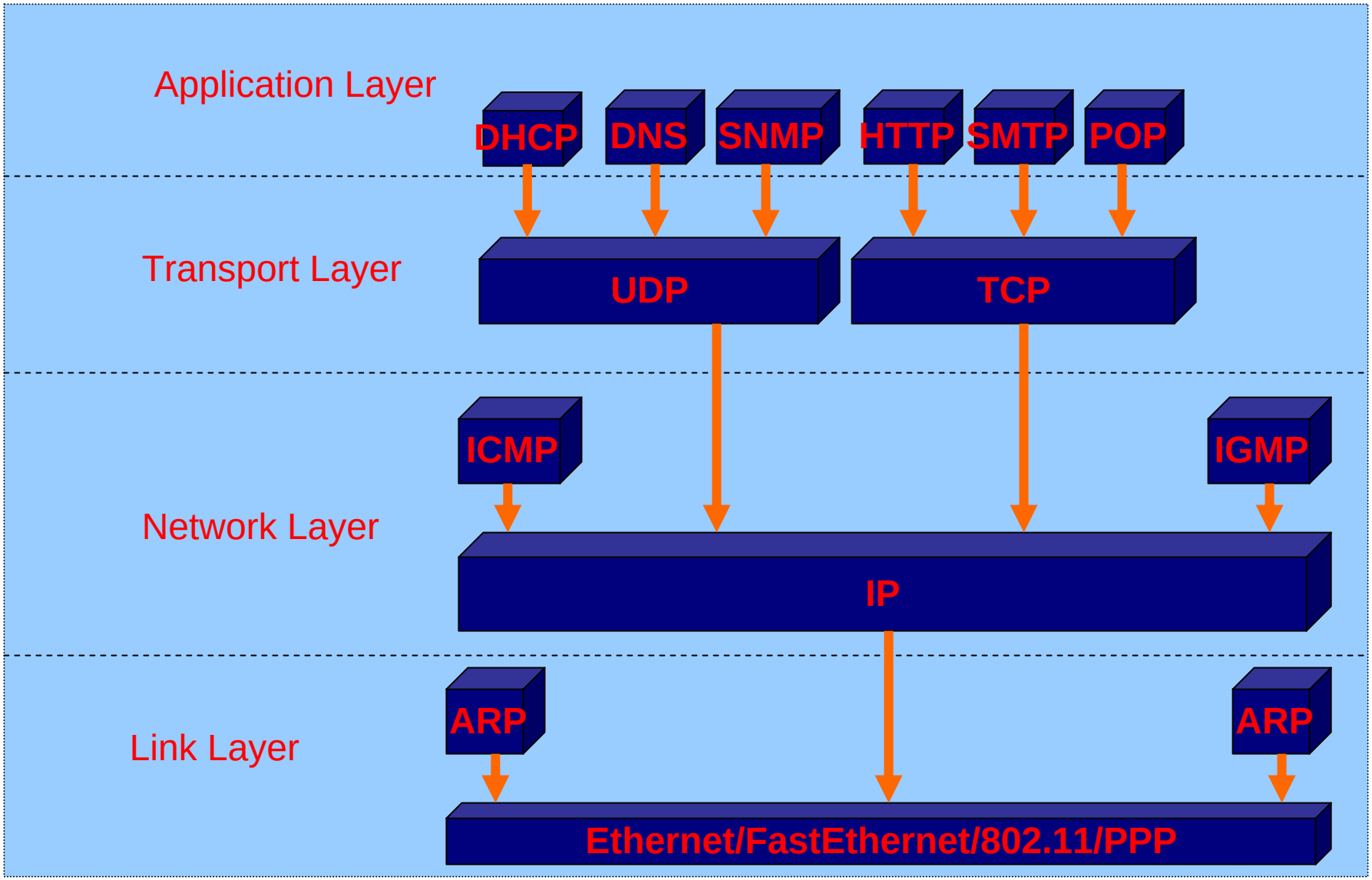
References

- V22.0480-001, Sana` Odeh , Computer Science Department, New York University.
- Representation and Management of Data on the Internet (67633), Yehoshua Sagiv, The Hebrew University - Institute of Computer Science.
- Java Network Programming and Distributed Computing, Reilly & Reilly.
- *Computer Networking: A Top-Down Approach Featuring the Internet,* Kurose & Rose, Pearson Addison Wesley.

Domain Name System (DNS)

RFC 1034
RFC 1035
http://www.ietf.org

TCP/IP Protocol Suite



DNS: Domain Name System

People: many identifiers:

- SSN, name, Passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- “name”, e.g., gaia.cs.umass.edu - used by humans

Q: map between IP addresses and name ?

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol* host, routers, name servers to communicate to *resolve* names (address/name translation)
  - note: core Internet function implemented as application-layer protocol
  - complexity at network’s “edge”

DNS name servers

Why not centralize DNS?

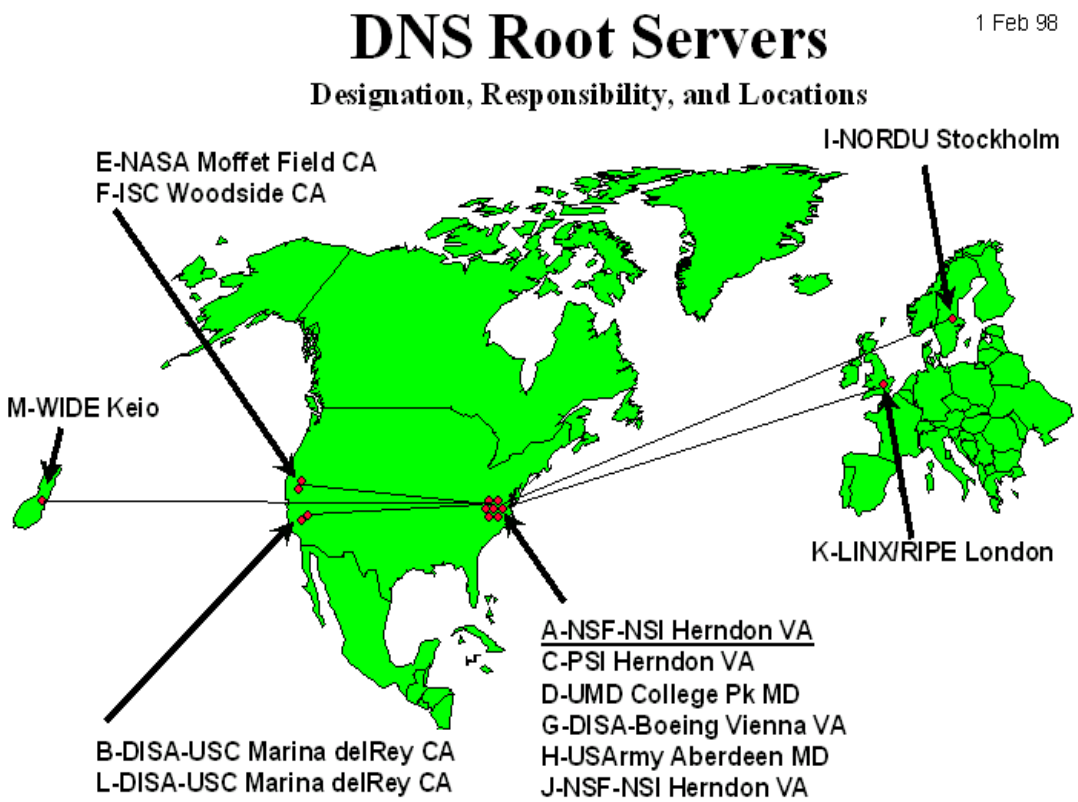
- single point of failure
- traffic volume
- distant centralized database
- maintenance

- doesn't *scale!*

- no server has all name-to-IP address mappings
- local name servers:
  - each ISP, company has *local (default) name server*
  - host DNS query first goes to local name server
- authoritative name server:
  - for a host: stores that host's IP address, name
  - can perform name/address translation for that host's name

DNS: Root name servers

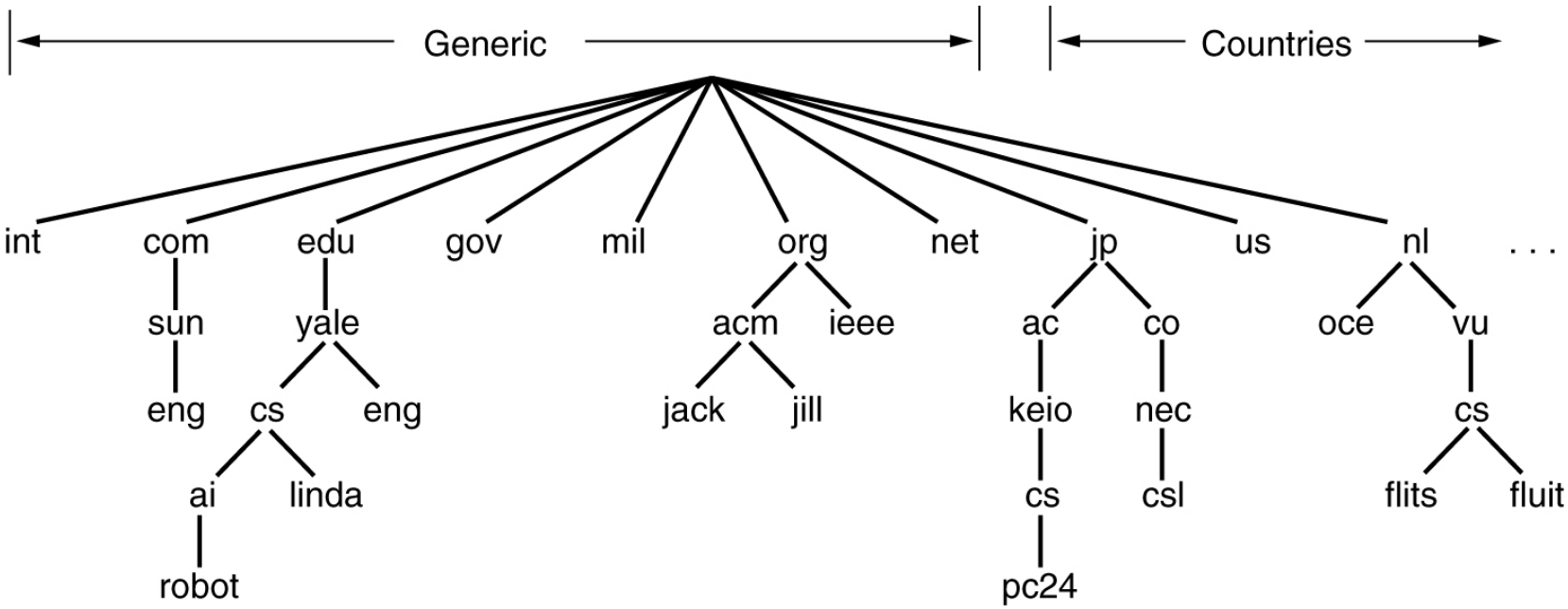
- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server
- ~ 13 root name servers worldwide



Further information about the root name servers can be found at: http://netmon.grnet.gr/stathost/rootns/

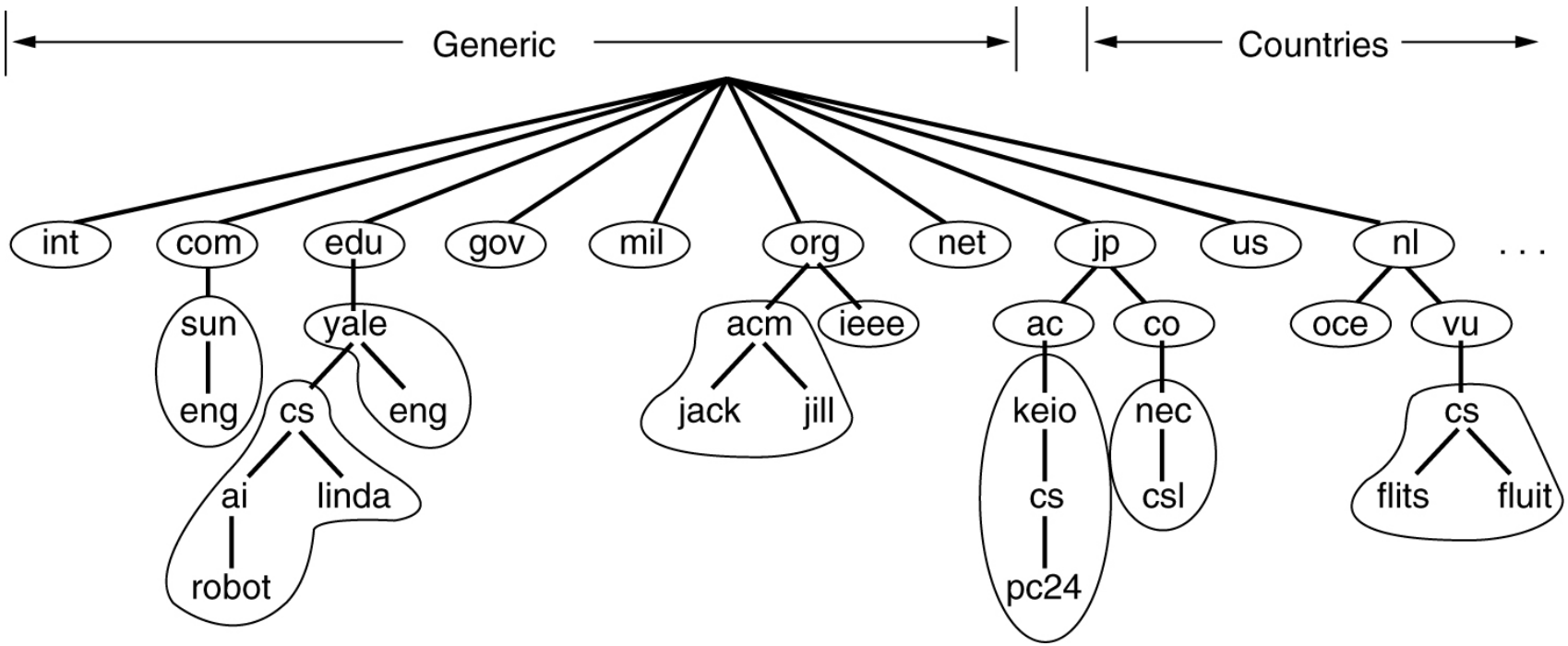
The DNS Name Space

A portion of the Internet domain name space showing some top Level Domains (TLDs).



Name Servers

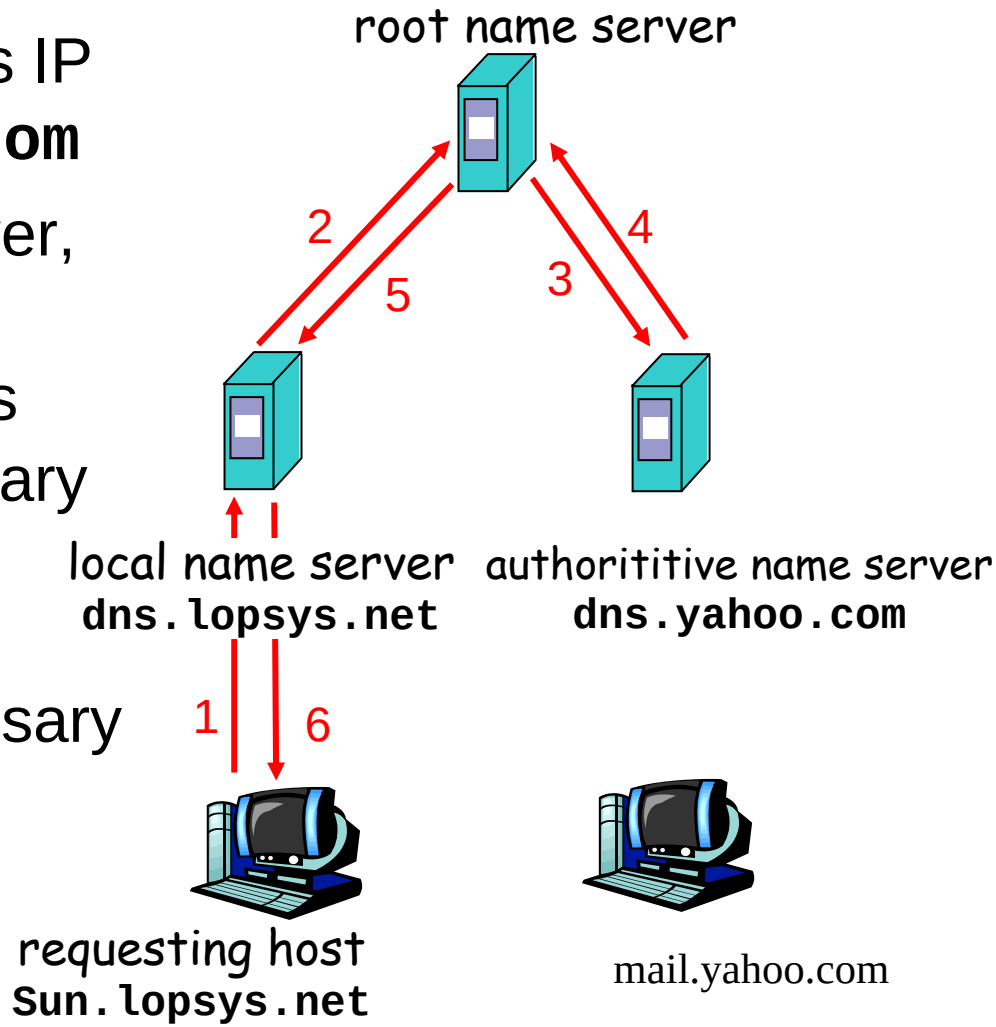
Part of the DNS name space showing the division into zones.



Simple DNS example

host **`sun.lopsys.net`** wants IP address of **`mail.yahoo.com`**

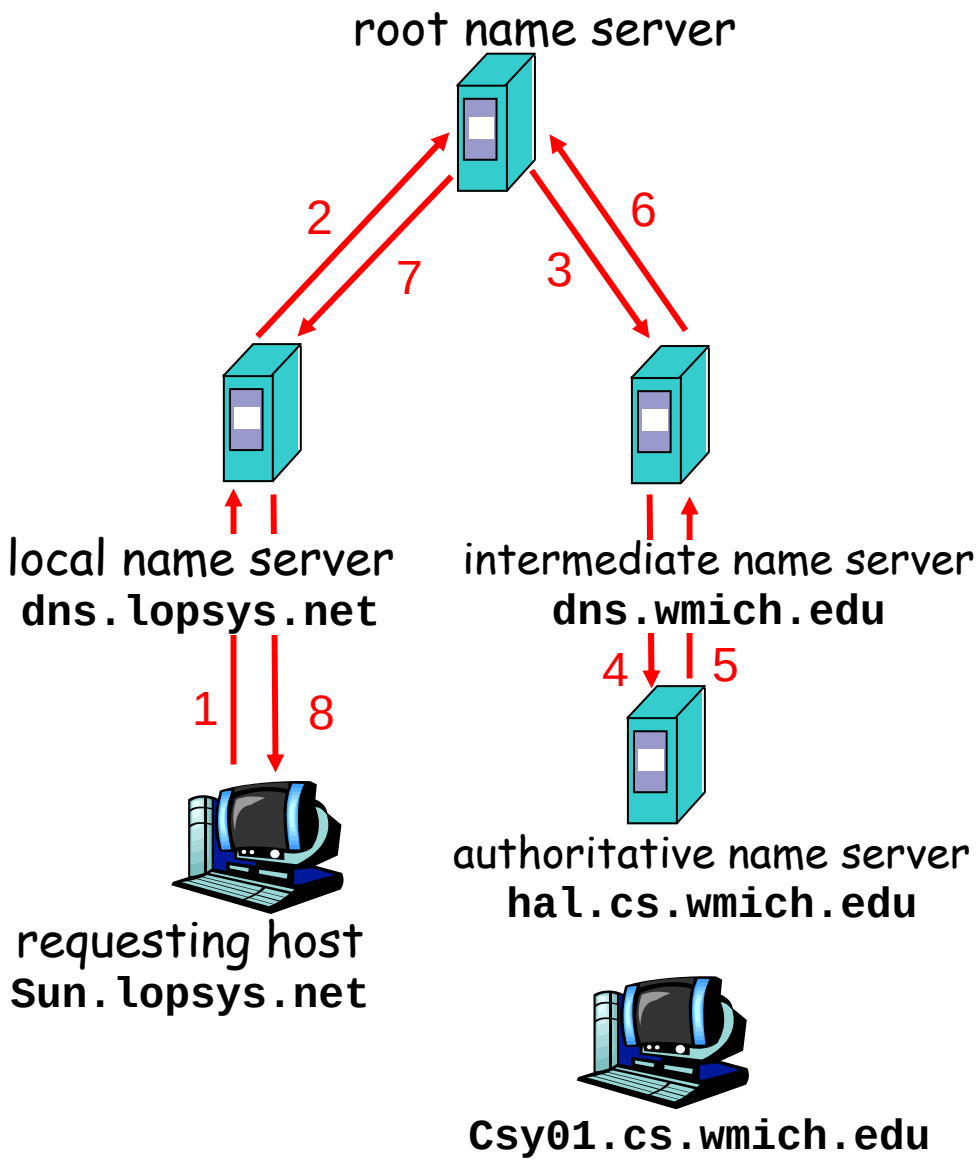
1. Contacts its local DNS server, **`dns.lopsys.net`**
2. **`dns.lopsys.net`** contacts root name server, if necessary
3. root name server contacts authoritative name server, **`dns.yahoo.com,`** if necessary



DNS example

Root name server:

- may not know authoratiative name server
- may know *intermediate name server:* who to contact to find authoritative name server



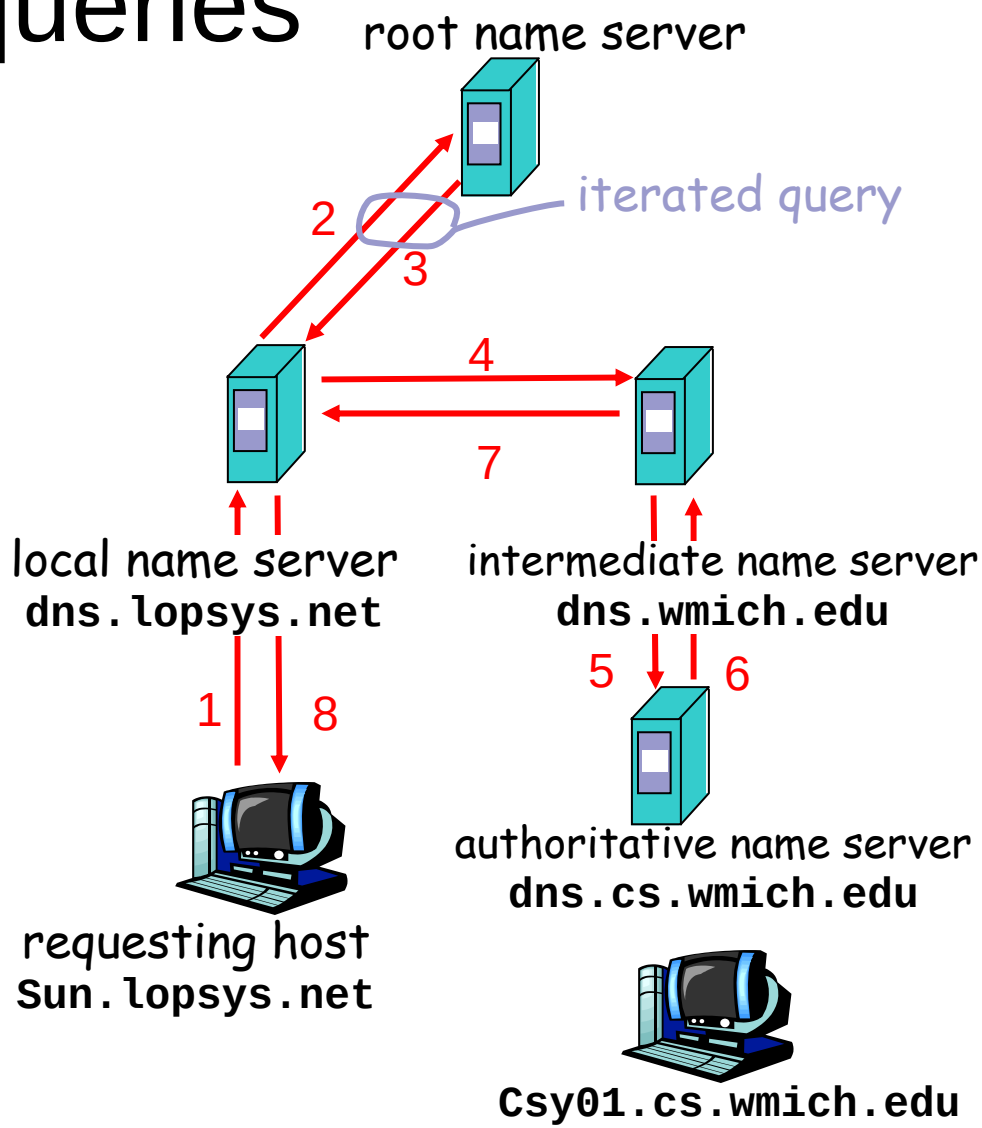
DNS: Iterated queries

recursive query:

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

iterated query:

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



DNS: caching and updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL usually 24 hours)

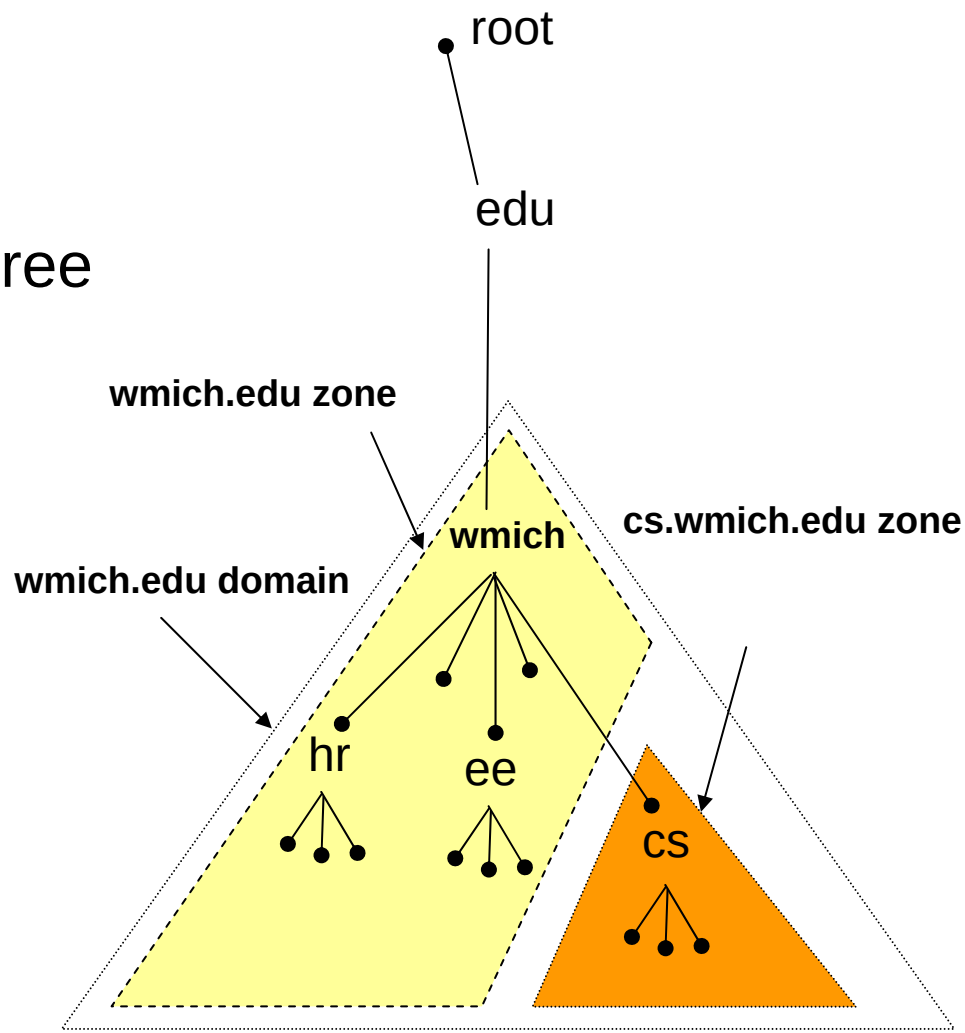
- update/notify mechanisms under design by IETF
  - RFC 2136
  - http://www.ietf.org/html.charters/dnsind-charter.html

Domains, Zones, Authority, Delegation

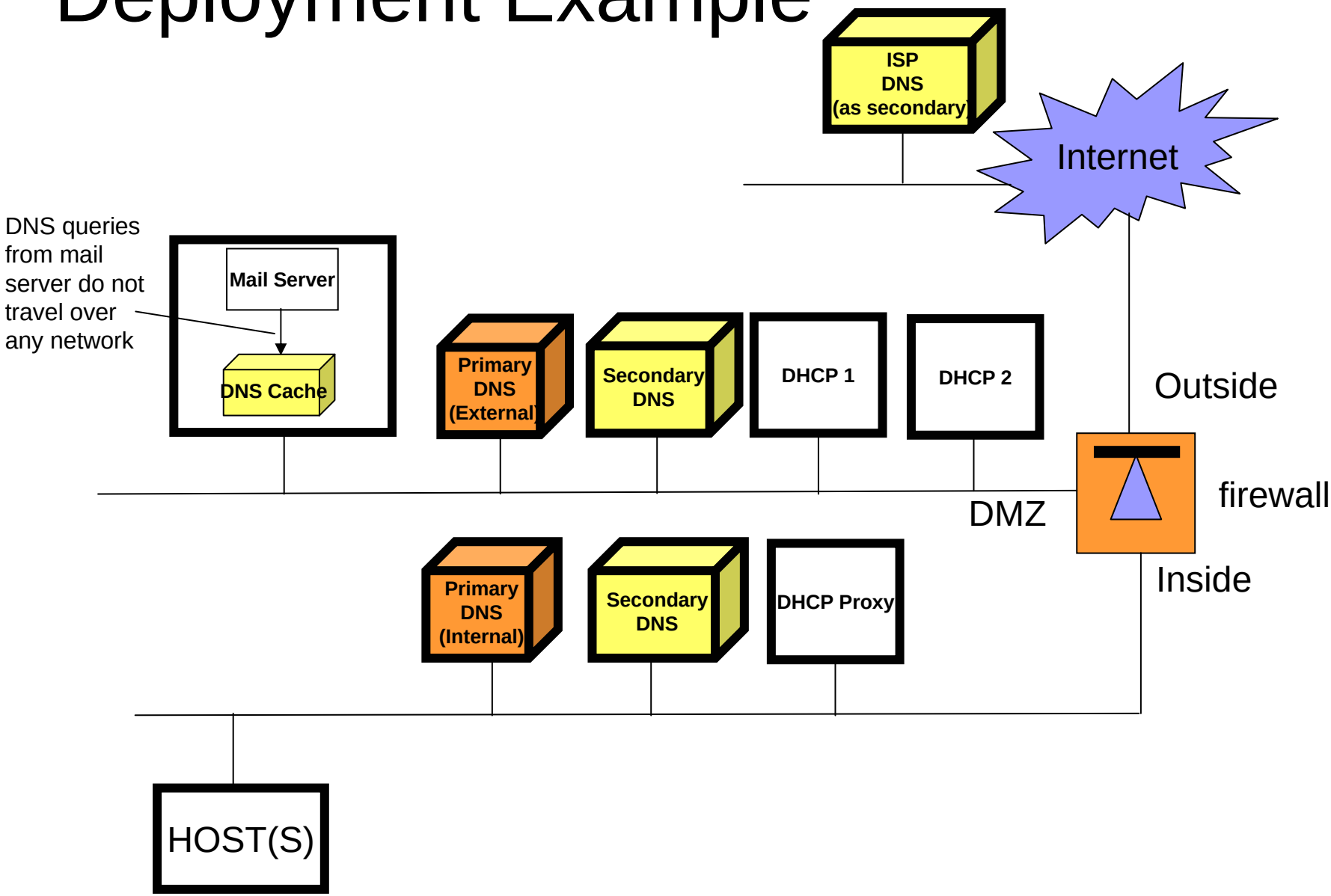
•Domain: is a node in the DNS tree, which includes all the nodes (domains) underneath it.

•Zone: is a portion of the DNS tree that a particular DNS server is ***authoritative*** for.

•A DNS Server may ***delegate*** authority of its subdomains to other organizations or departments.



Deployment Example

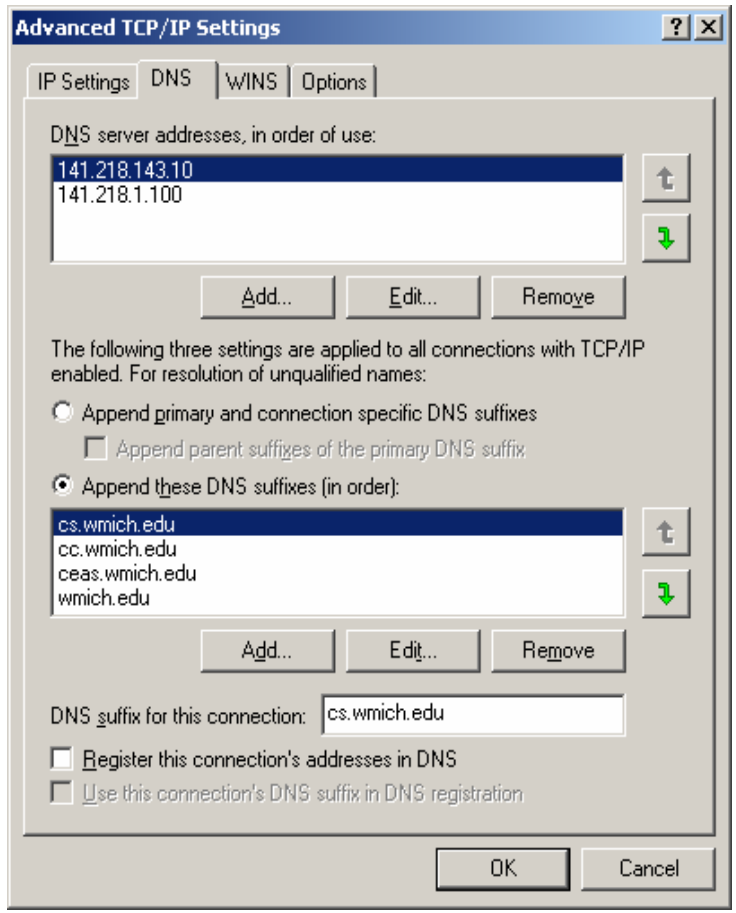


DNS Clients (resolver configuration)

- A DNS client is called a *resolver*.
- A call to **`getByName()`** is handled by a resolver (typically part of the client).

UNIX: /etc/resolv.conf

```
nameserver 141.218.143.12
nameserver 141.218.40.10
nameserver 141.218.1.100
domain cs.wmich.edu
```



DNS Servers

- The name of the DNS server in UNIX is ***named***
- The configuration file for ***named*** can be found usually in /etc/named.conf
- The zone files are usually kept in /var/named with all the the zone resource records (e.g., A, PTR, MX, NS, CNAME).
- BIND (Berkeley Internet Name Domain) is an common implementation of DNS server, source code and binaries are freely available http://www.isc.org

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 IP address of authoritative name server for this domain
- Type=CNAME
  - **name** is an alias name for some "cannonical" (the real) name
  - **value** is cannonical name
- Type=MX
  - **value** is hostname of mailserver associated with **name**

Resource Records

The principal DNS resource records types.

Type	Meaning	Value
SOA	Start of Authority	Parameters for this zone
A	IP address of a host	32-Bit integer
MX	Mail exchange	Priority, domain willing to accept e-mail
NS	Name Server	Name of a server for this domain
CNAME	Canonical name	Domain name
PTR	Pointer	Alias for an IP address
HINFO	Host description	CPU and OS in ASCII
TXT	Text	Uninterpreted ASCII text

Resource Records (2)

```
; Authoritative data for cs.vu.nl
cs.vu.nl.          86400   IN   SOA      star boss (952771,7200,7200,2419200,86400)
cs.vu.nl.          86400   IN   TXT      "Divisie Wiskunde en Informatica."
cs.vu.nl.          86400   IN   TXT      "Vrije Universiteit Amsterdam."
cs.vu.nl.          86400   IN   MX       1 zephyr.cs.vu.nl.
cs.vu.nl.          86400   IN   MX       2 top.cs.vu.nl.

flits.cs.vu.nl.    86400   IN   HINFO    Sun Unix
flits.cs.vu.nl.    86400   IN   A        130.37.16.112
flits.cs.vu.nl.    86400   IN   A        192.31.231.165
flits.cs.vu.nl.    86400   IN   MX       1 flits.cs.vu.nl.
flits.cs.vu.nl.    86400   IN   MX       2 zephyr.cs.vu.nl.
flits.cs.vu.nl.    86400   IN   MX       3 top.cs.vu.nl.
www.cs.vu.nl.      86400   IN   CNAME    star.cs.vu.nl
ftp.cs.vu.nl.      86400   IN   CNAME    zephyr.cs.vu.nl

rowboat                    IN   A        130.37.56.201
                           IN   MX       1 rowboat
                           IN   MX       2 zephyr
                           IN   HINFO    Sun Unix

little-sister              IN   A        130.37.62.23
                           IN   HINFO    Mac MacOS

laserjet                   IN   A        192.31.231.216
                           IN   HINFO    "HP Laserjet IIISi" Proprietary
```

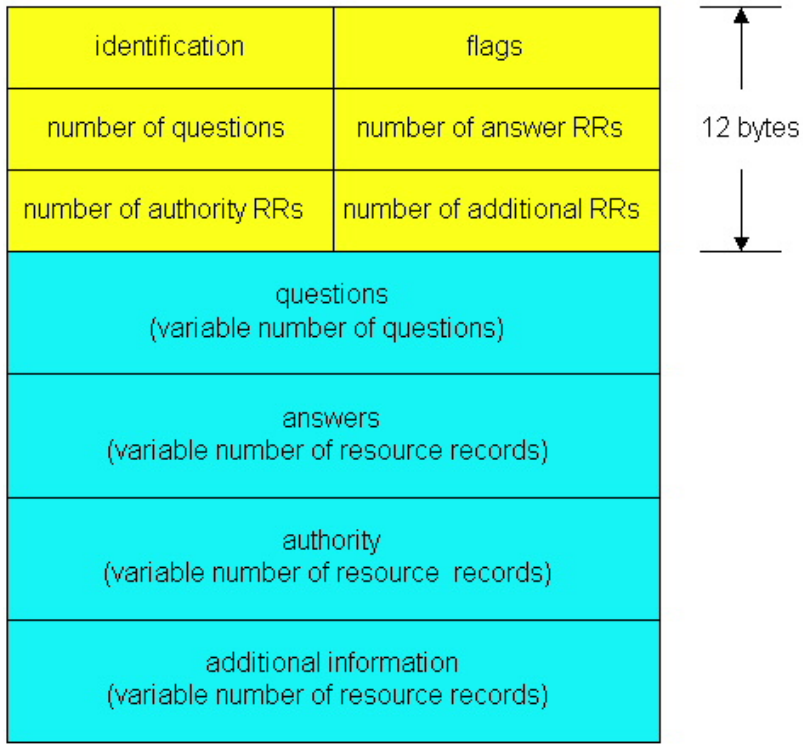
A portion of a possible DNS database for *cs.vu.nl.*

DNS protocol, messages

DNS protocol : *query* and *reply* messages, both with same *message format*

msg header

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



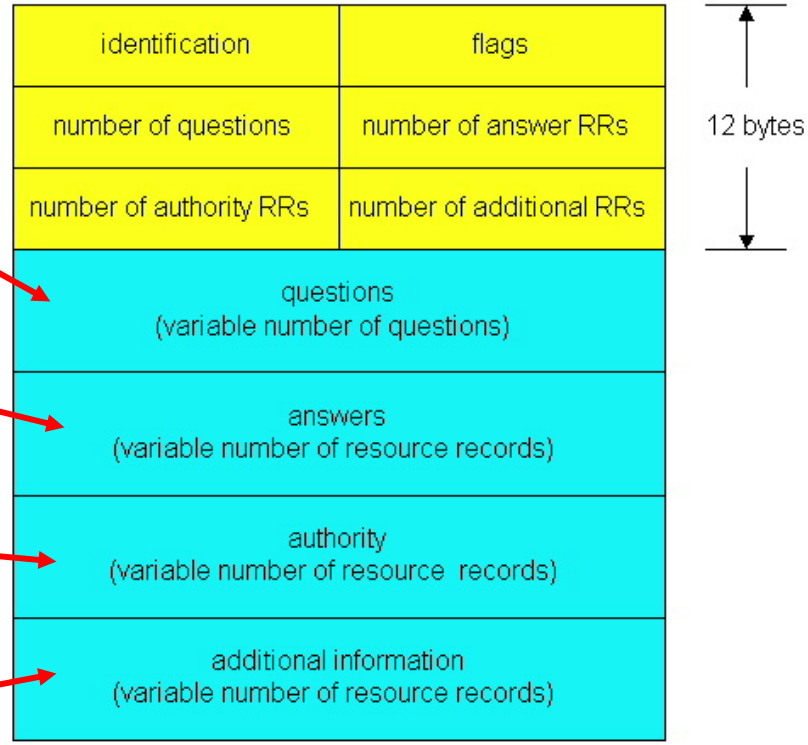
DNS protocol, messages

Name, type fields for a query

RRs in response to query

records for authoritative servers

additional "helpful" info that may be used



nslookup

$ nslookup -d csy01.cs.wmich.edu

```
------------
Got answer:
    HEADER:
        opcode = QUERY, id = 6, rcode = NOERROR
        header flags:  response, auth. answer, want
recursion, recursion avail.
        questions = 1,  answers = 1,  authority records = 4,
additional = 4

    QUESTIONS:
        csy01.cs.wmich.edu, type = A, class = IN
    ANSWERS:
    ->  csy01.cs.wmich.edu
        internet address = 141.218.143.215
        ttl = 14400 (4 hours)
    AUTHORITY RECORDS:
    ->  cs.wmich.edu
        nameserver = gumby.cc.wmich.edu
        ttl = 14400 (4 hours)
    ->  cs.wmich.edu
        nameserver = hal.cs.wmich.edu
        ttl = 14400 (4 hours)
    ADDITIONAL RECORDS:
    ->  gumby.cc.wmich.edu
        internet address = 141.218.20.114
        ttl = 3120 (52 mins)
    ->  hal.cs.wmich.edu
        internet address = 141.218.143.10
        ttl = 14400 (4 hours)
------------
Name:    csy01.cs.wmich.edu
Address:  141.218.143.215
```

$ nslookup -querytype=MX cnn.com

```
Server:  hal.cs.wmich.edu
Address:  141.218.143.10

Non-authoritative answer:
cnn.com MX preference = 10, mail exchanger = atlmail1.turner.com
cnn.com MX preference = 10, mail exchanger = atlmail4.turner.com
cnn.com MX preference = 20, mail exchanger = atlmail2.turner.com
cnn.com MX preference = 30, mail exchanger = nymail1.turner.com
cnn.com MX preference = 5, mail exchanger = atlmail3.turner.com

com     nameserver = a.gtld-servers.net
com     nameserver = g.gtld-servers.net
com     nameserver = h.gtld-servers.net
com     nameserver = c.gtld-servers.net
com     nameserver = i.gtld-servers.net
com     nameserver = b.gtld-servers.net
com     nameserver = d.gtld-servers.net
com     nameserver = l.gtld-servers.net
com     nameserver = f.gtld-servers.net
com     nameserver = j.gtld-servers.net
com     nameserver = k.gtld-servers.net
com     nameserver = e.gtld-servers.net
com     nameserver = m.gtld-servers.net
atlmail1.turner.com     internet address = 64.236.240.146
atlmail4.turner.com     internet address = 64.236.221.5
atlmail2.turner.com     internet address = 64.236.240.147
nymail1.turner.com      internet address = 64.236.170.7
nymail1.turner.com      internet address = 64.236.170.8
atlmail3.turner.com     internet address = 64.236.240.169
g.gtld-servers.net      internet address = 192.42.93.30
h.gtld-servers.net      internet address = 192.54.112.30
```

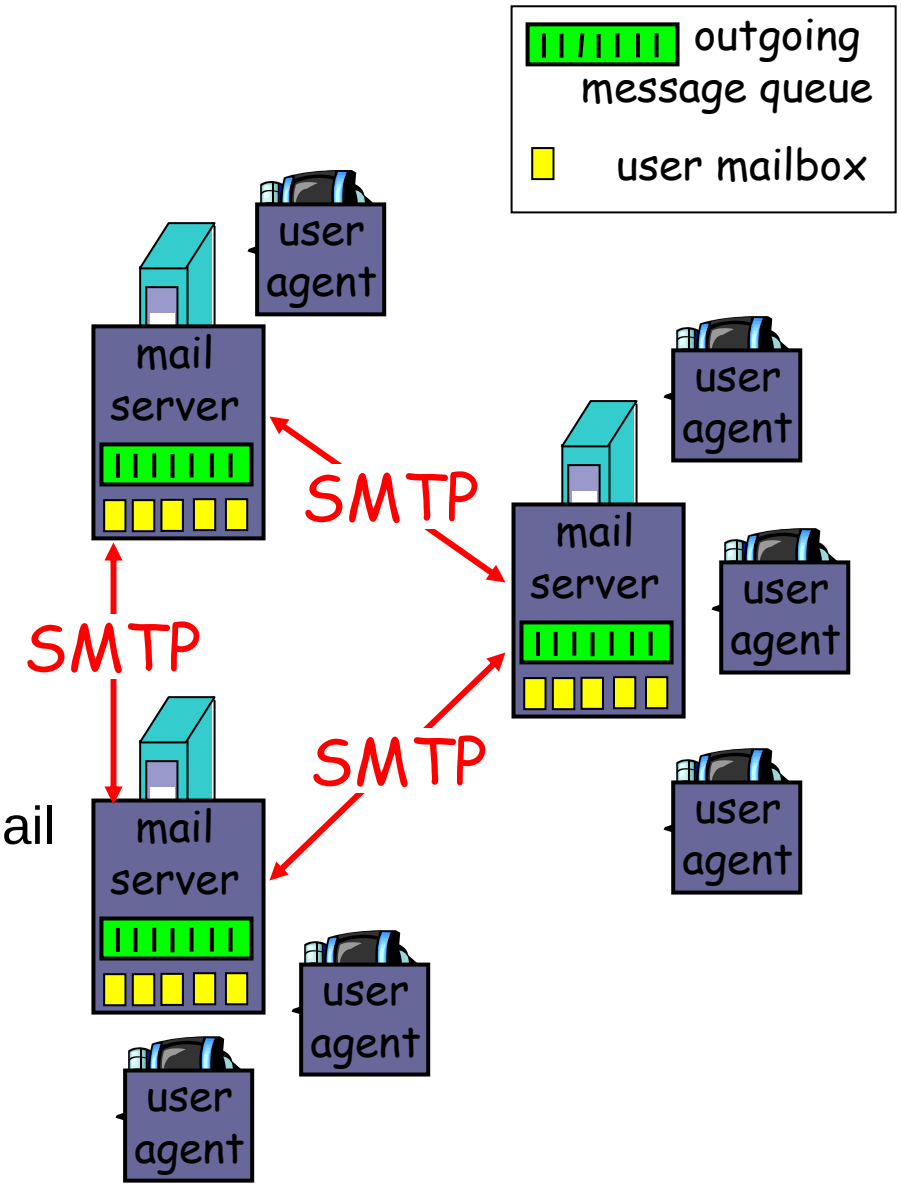
Electronic Mail

Electronic Mail

Three major components:

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

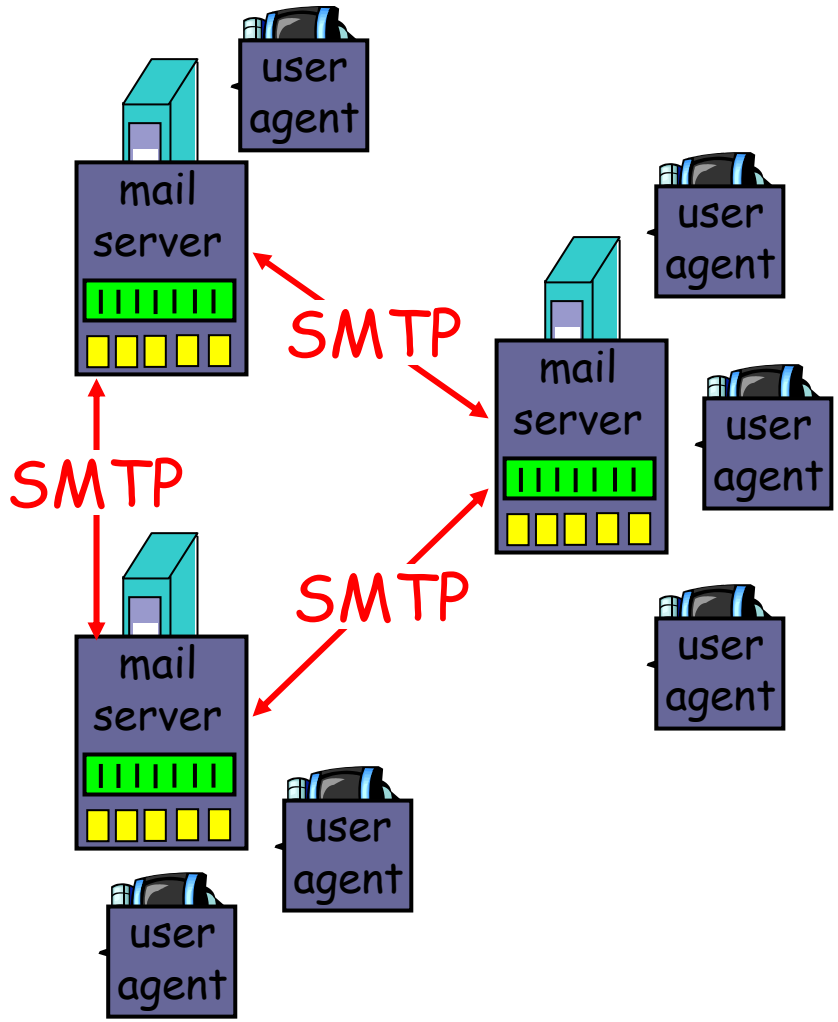
- User Agent
- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Eudora, Outlook, elm, Netscape Messenger
- outgoing, incoming messages stored on server



Electronic Mail: mail servers

Mail Servers

- mailbox contains incoming messages (yet to be read) 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 821]

- uses TCP to reliably transfer email msg 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
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCII

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 Example:

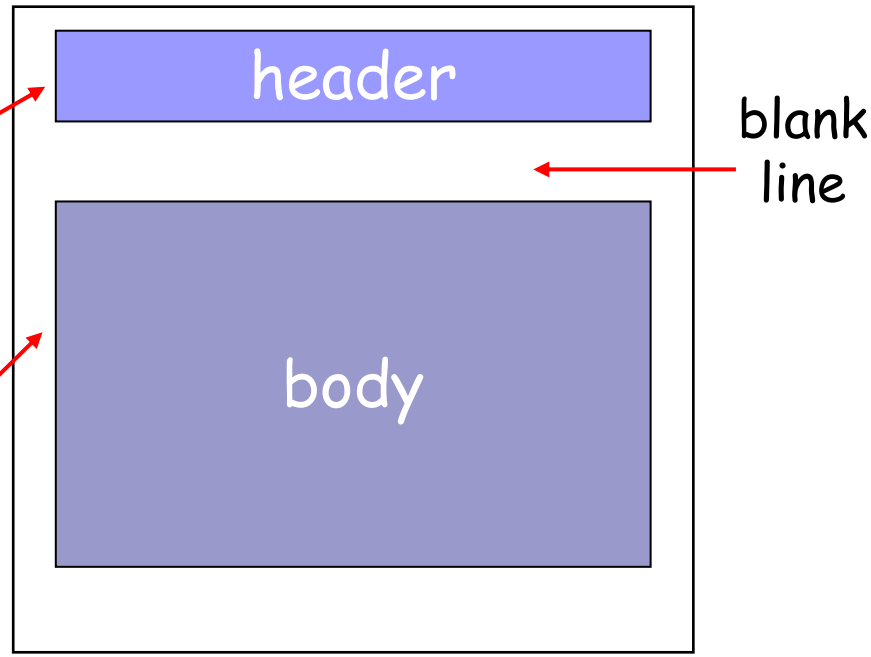
```
[pasward@sandrock pasward]$ telnet www.ynpcsc.gov.cn 25
Trying 202.98.190.193...
Connected to www.ynpcsc.gov.cn.
Escape character is '^]'.
220 www.ynpcsc.gov.cn ESMTP Server (Microsoft Exchange
Internet Mail Service 5.5.2650.21) ready
MAIL FROM: <jqRandom@hotmail.com>
250 OK - mail from <jqRandom@hotmail.com>
RCPT TO: <pasward@big.uwaterloo.ca>
250 OK - Recipient <pasward@big.uwaterloo.ca>
DATA
354 Send data.  End with CRLF.CRLF
yet another open relay
.
250 OK
Quit
221 closing connection
Connection closed by foreign host
```

And this was e-mailed to me:

```
pasward@big pasward]$ cat /var/spool/mail/pasward
From jqRandom@hotmail.com  Wed Mar  6 12:57:00 2002
Return-Path: <jqRandom@hotmail.com>
Received: from www.ynpcsc.gov.cn ([202.98.190.193])
  by big.uwaterloo.ca (8.11.0/8.11.0) with ESMTP id g26HuvY23119
  for <pasward@big.uwaterloo.ca>;
Wed, 6 Mar 2002 12:56:58 -0500
Date: Wed, 6 Mar 2002 12:56:58 -0500
From: jqRandom@hotmail.com
Message-Id: <200203061756.g26HuvY23119@big.uwaterloo.ca>
Received: from sandrock.uwaterloo.ca ([129.97.105.32])
  by www.ynpcsc.gov.cn with SMTP (Microsoft Exchange Internet
    Mail Service Version 5.5.2650.21)id GH8G9YG8;
  Thu, 7 Mar 2002 01:58:55 +0800
yet another open relay
```

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 commands*!
- body
- the “message”, ASCII characters only



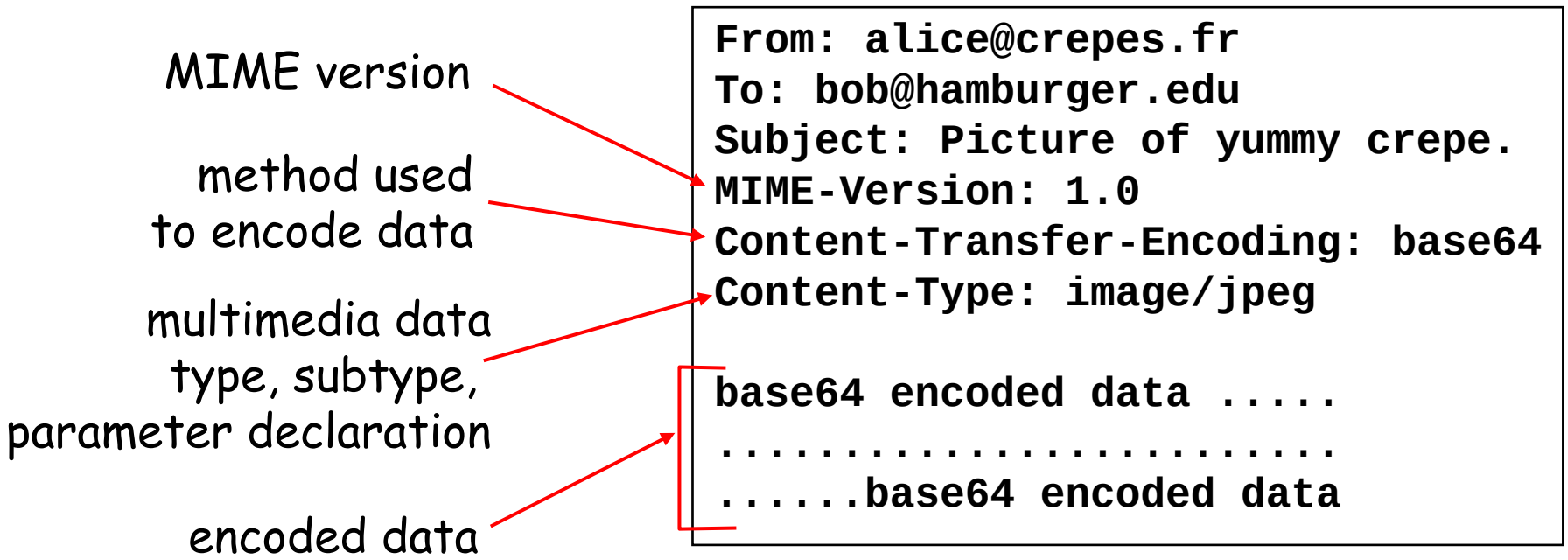
MIME – Multipurpose Internet Mail Extensions

Problems with international languages:

- Languages with accents (French, German).
- Languages in non-Latin alphabets (Hebrew, Russian).
- Languages without alphabets (Chinese, Japanese).
- Messages not containing text at all (audio or images).

Message format: multimedia extensions

- MIME: multimedia mail extension, RFC 2045, 2056
- Additional lines in msg header declare MIME content type



MIME types

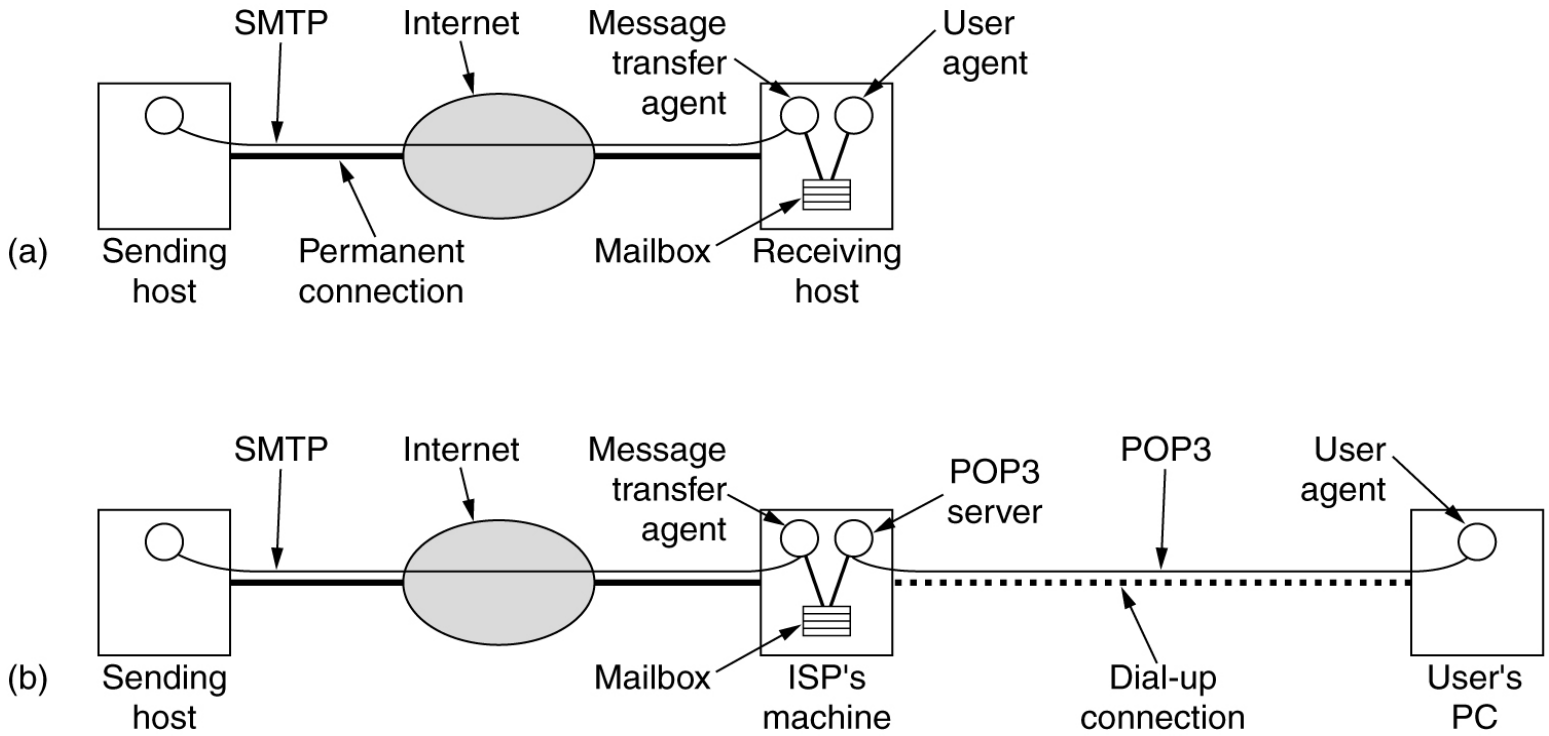
`Content-Type: type/subtype; parameters`

- Text
  - example subtypes: **`plain`**, **`html`**
- Image
  - example subtypes: **`jpeg`**, **`gif`**
- Audio
  - exampe subtypes: **`basic`** (8-bit mu-law encoded), **`32kadpcm`** **(32 kbps coding)**
- Video
  - example subtypes: **`mpeg`**, **`quicktime`**
- Application
  - other data that must be processed by reader before “viewable”
  - example subtypes: **`msword`**, **`octet-stream`**

Multipart Type

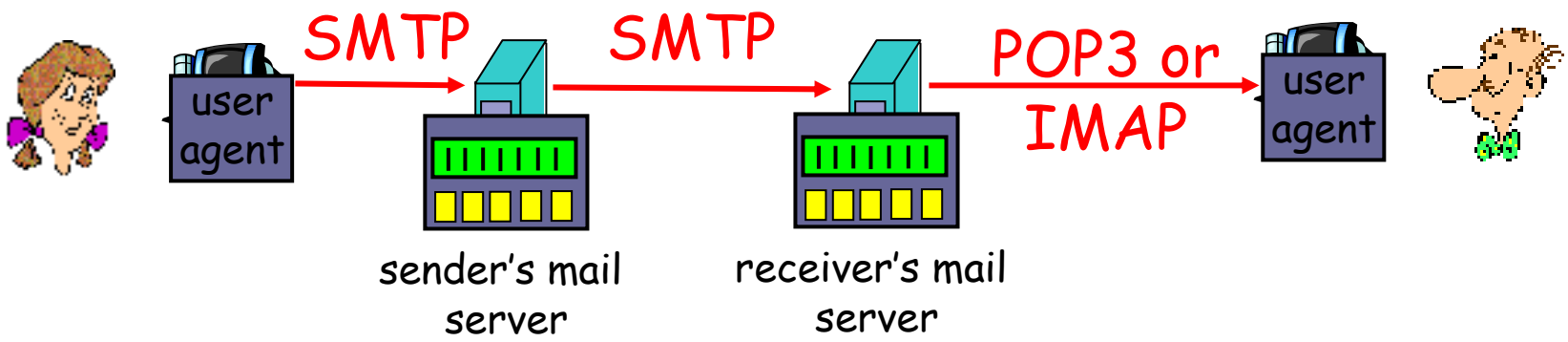
```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789

--98766789
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain

Dear Bob,
Please find a picture of a crepe.
--98766789
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
--98766789--
```

Final Delivery



(a) Sending and reading mail when the receiver has a permanent Internet connection and the user agent runs on the same machine as the message transfer agent. (b) Reading e-mail when the receiver has a dial-up connection to an ISP.

Mail access protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) and download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - more features (more complex)
    - manipulation of stored msgs on server
  - HTTP: 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 alice
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

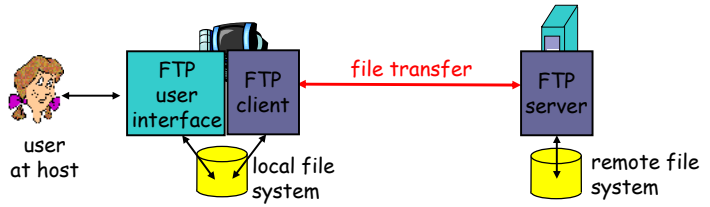
```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

A comparison of POP3 and IMAP.

Feature	POP3	IMAP
Where is protocol defined?	RFC 1939	RFC 2060
Which TCP port is used?	110	143
Where is e-mail stored?	User's PC	Server
Where is e-mail read?	Off-line	On-line
Connect time required?	Little	Much
Use of server resources?	Minimal	Extensive
Multiple mailboxes?	No	Yes
Who backs up mailboxes?	User	ISP
Good for mobile users?	No	Yes
User control over downloading?	Little	Great
Partial message downloads?	No	Yes
Are disk quotas a problem?	No	Could be in time
Simple to implement?	Yes	No
Widespread support?	Yes	Growing

FTP

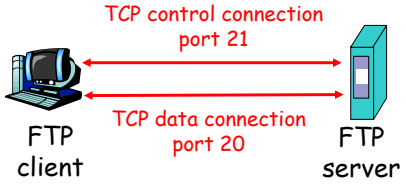
FTP: the file transfer protocol



- transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ftp: RFC 959
- ftp server: port 21

FTP: separate control, data connections

- FTP client contacts FTP server at port 21, specifying TCP as transport protocol
- Client obtains authorization over control connection
- Client browses remote directory by sending commands over control connection.
- When server receives a command for a file transfer, the server opens a TCP data connection to client
- After transferring one file, server closes connection.



- Server opens a second TCP data connection to transfer another file.
- Control connection: "out of band"
- 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

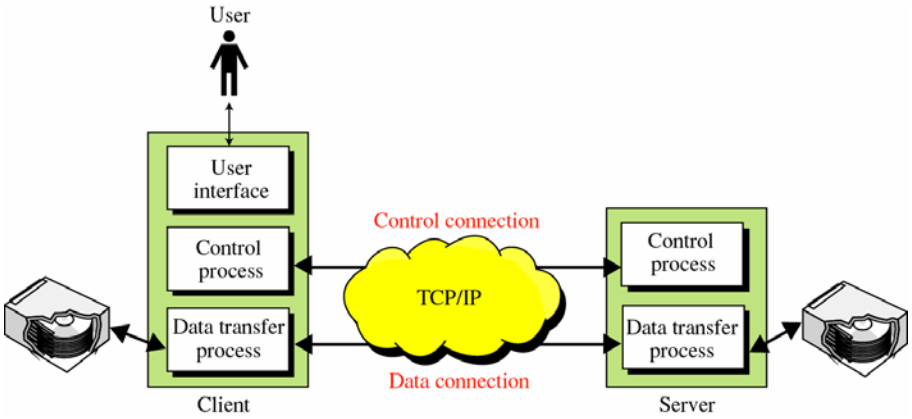
Sample return codes

- 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`**

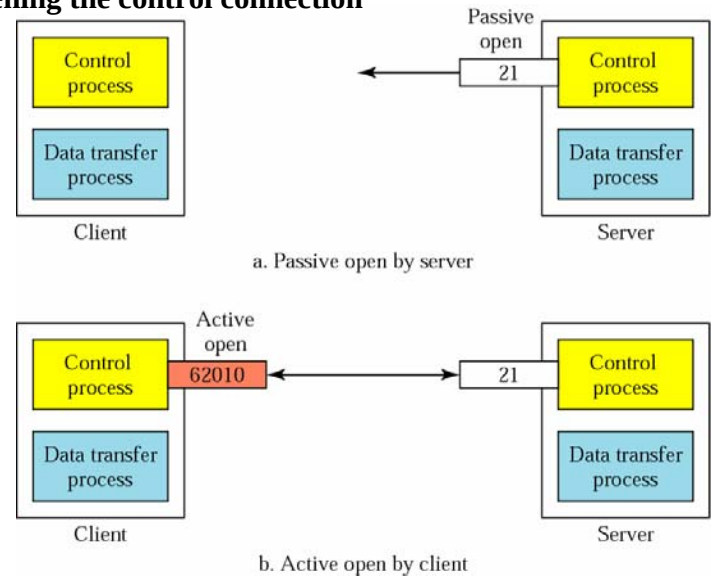
Note

FTP uses the services of TCP. It needs two TCP connections. The well-known port 21 is used for the control connection and the well-known port 20 for the data connection.

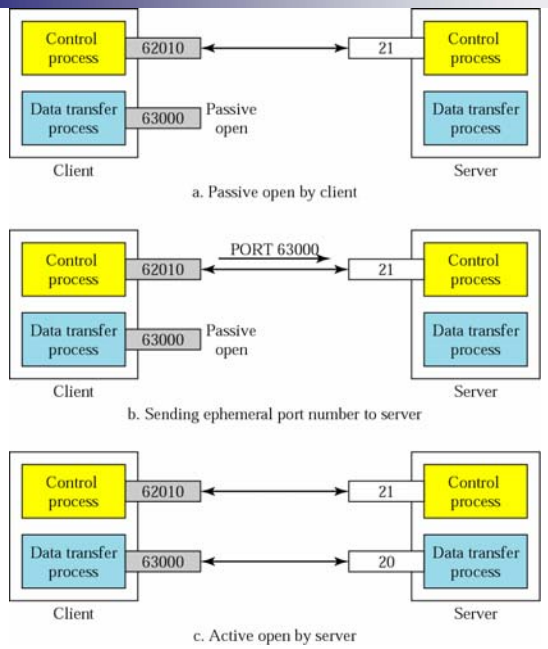
FTP



Opening the control connection



Creating the data connection



Command processing

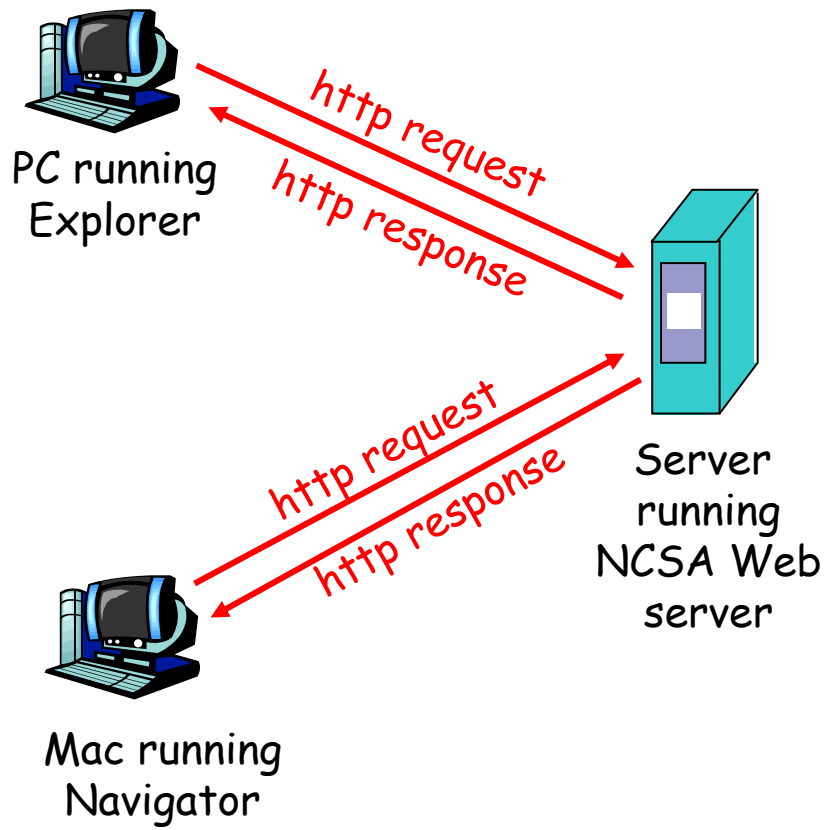
File transfer

HTTP – HyperText Transfer Protocol

The Web: the http protocol

http: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests
- http1.0: RFC 1945
- http1.1: RFC 2068



Universal Resource Locator

protocol://host:port/path#anchor?parameters

http://www.google.com/search?hl=en&g=blabla			

- There are other types of URL's
  - mailto:<account@site>
  - news:<newsgroup-name>

The http protocol: more

http: TCP transport service:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- http messages (application-layer protocol messages) exchanged between browser (http client) and Web server (http server)
- TCP connection closed

http is "stateless"

- server maintains no information about past client requests

aside

Protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

Persistent vs. Non-Persistent Connection

- A page that we see on the browser can include more than one resource
- The resources are sent from the server to the client one after the other
- Sending the resources to the browser can be by using a persistent connection or by using a non-persistent connection

Non-Persistent Connection

1. Browser opens TCP connection to port 80 of server (handshake)
2. Browser sends http request message
3. Server receives request, locates object, sends response
4. Server closes TCP connection
5. Browser receives response, parses object
6. Browser repeats steps 1-5 for each embedded object

Persistent Connection

1. Browser opens TCP connection to port 80 of server (handshake)
2. Browser sends http request message
3. Server receives request, locates object, sends response
4. Browser receives response, parses object
5. Browser repeats steps 2-4 for each embedded object
6. TCP connection closes on demand or timeout

Advantages of Persistent Connection

- **CPU time** saved in routers and hosts
- HTTP requests and responses can be **pipelined** on a connection
- **Network congestion** is reduced
- **Latency** on subsequent requests is reduced

What are the disadvantages of persistent connection?

Pipelines

- 2 types of persistent connections
  - without pipelining
    - the client issues a new request only after the previous response has arrived
  - with pipelining
    - client sends the request as soon as it encounters a reference
    - multiple requests/responses

http message format: request

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

request line (GET, POST, HEAD commands)

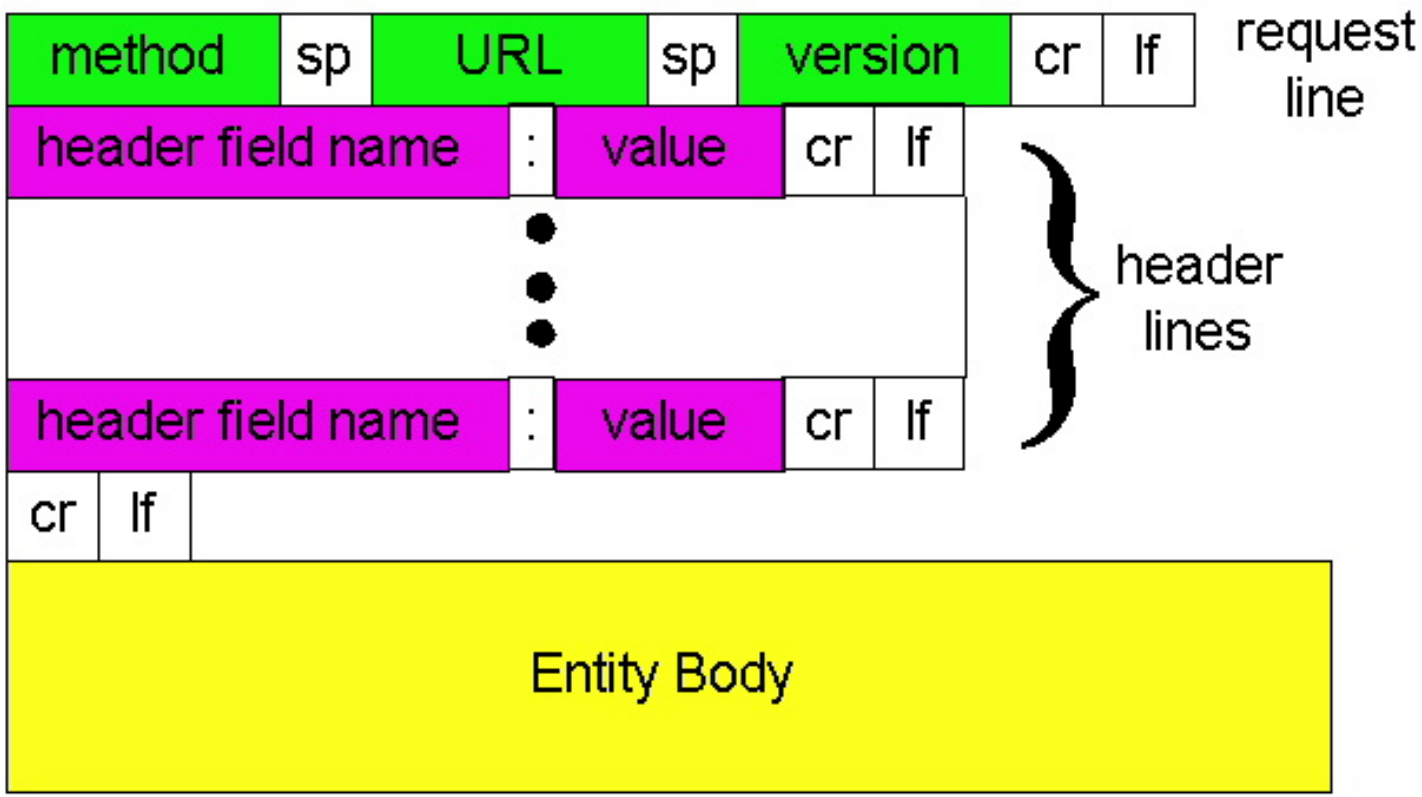
header lines

```
GET /somedir/page.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr
```

(extra carriage return, line feed)

Carriage return, line feed indicates end of message

http request message: general format



http request message: more info

- http/1.0 has only three request *methods*
  - *GET:*
  - *POST:* for forms. Uses *Entity Body* to transfer form info
  - *HEAD:* Like *GET* but response does not actually return any info. This is used for debugging/test purposes

- http/1.1 has two additional request *methods*
  - *PUT:* Allows uploading object to web server
  - *DELETE:* Allows deleting object from web server

Trying out http (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

Opens TCP connection to port 80 (default http server port) at cis.poly.edu. Anything typed in sent to port 80 at cis.poly.edu.

2. Type in a GET http request:

```
GET /~ross/index.html HTTP/1.0
```

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to http server

3. Look at response message sent by http server!

Try telnet www.cs.wmich.edu 80

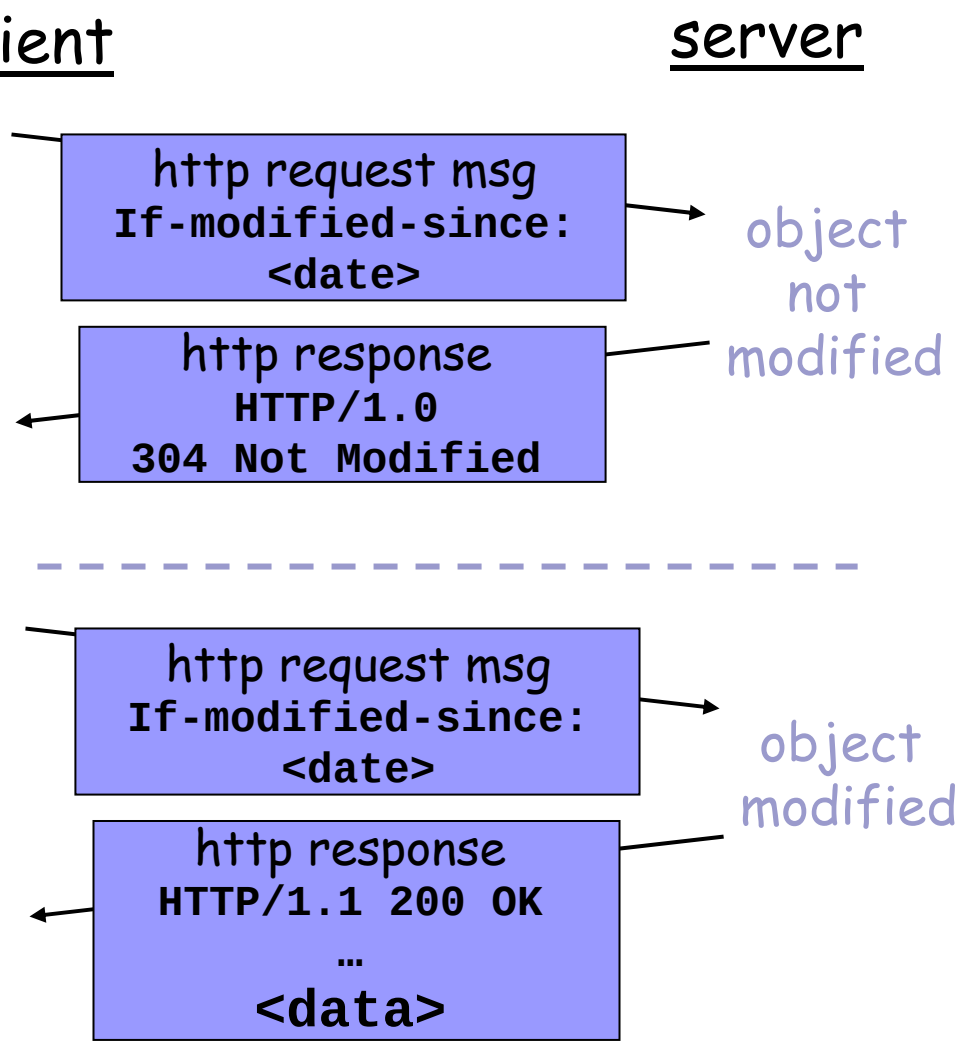
User-server interaction: conditional GET

- Goal: don't send object if client has up-to-date stored (cached) version
- client: specify date of cached copy in http request

```
If-modified-since:
   <date>
```

- server: response contains no object if cached copy up-to-date:

```
HTTP/1.0 304 Not
   Modified
```



Post Example

- Here's a typical form submission, using POST:

```
POST /path/register.cgi HTTP/1.0
From: frog@cs.huji.ac.il
User-Agent: HTTPTool/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 35

home=Ross+109&favorite+flavor=flies
```

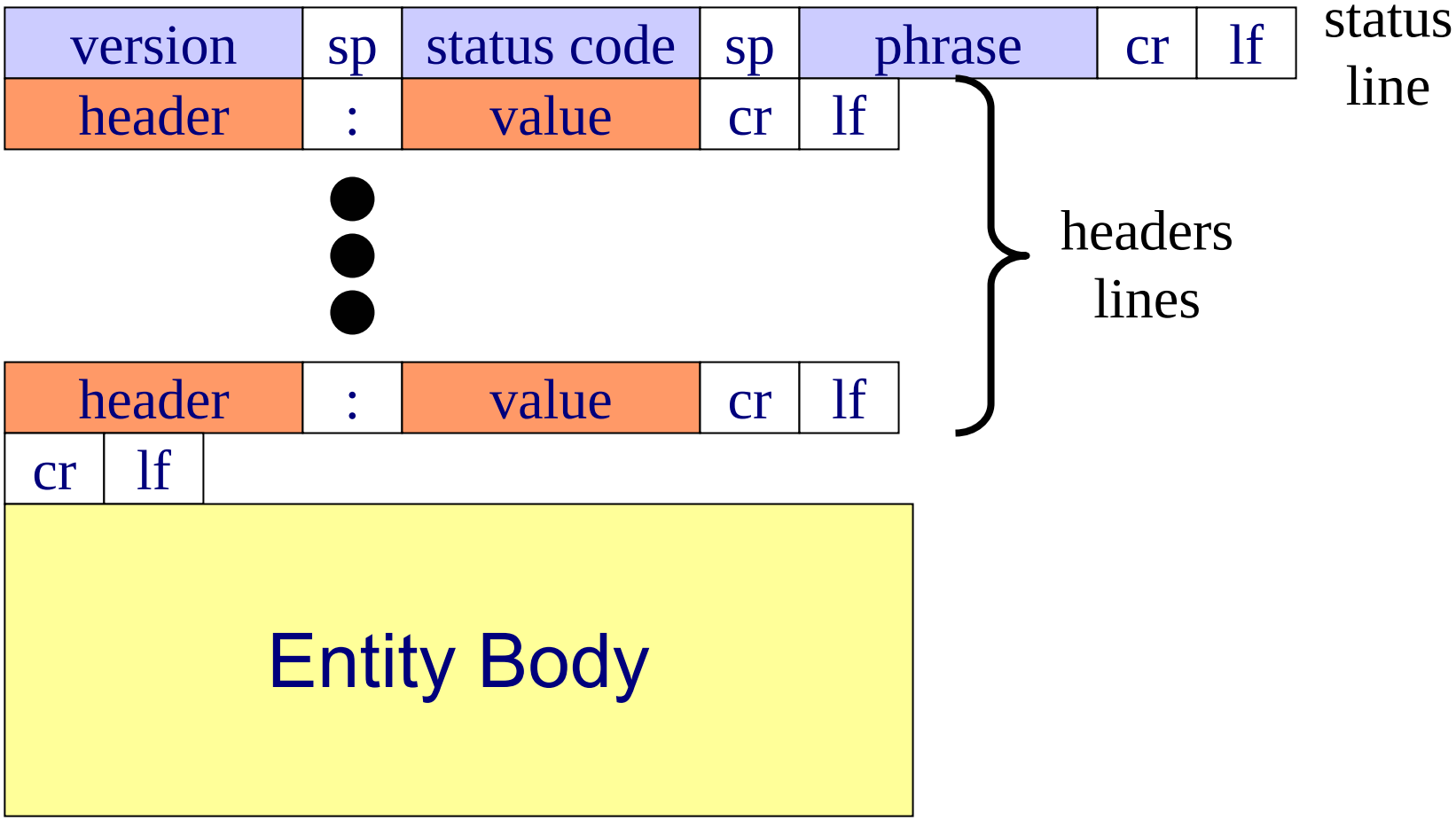
Virtual Hosts

- With HTTP 1.1, one server at one IP address can be multi-homed:
  - “www.cs.wmich.edu” and “www.ece.wmich.edu” can live on the same server
  - These are called **virtual hosts**
  - Without this mechanism, we have to use 2 different IP addresses
- It is like several people sharing one phone
- An HTTP request must specify the host name (and possibly port) for which the request is intended (this is done using the Host header).

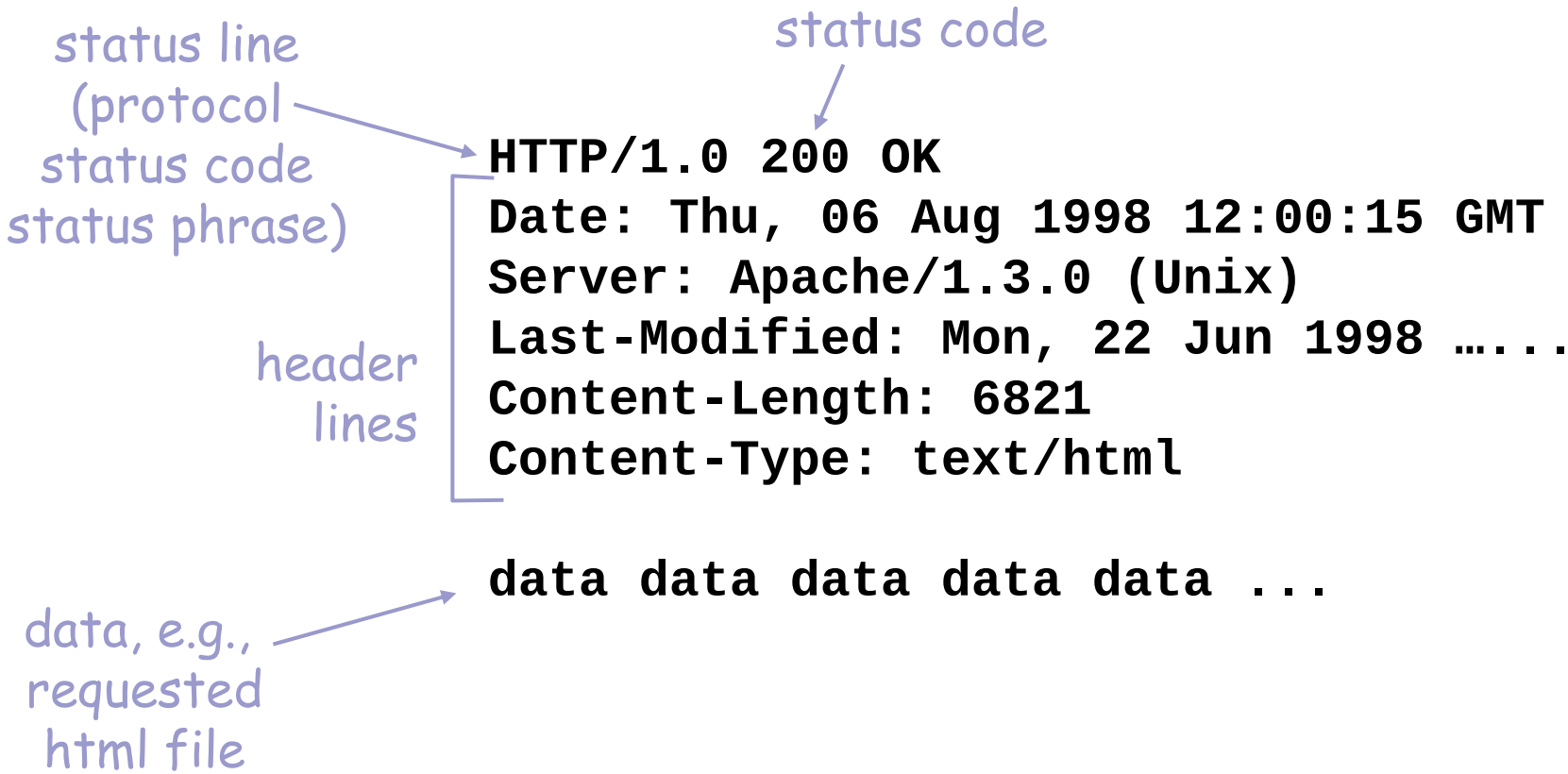
Virtual Hosting (cont.)

- Virtual hosting
  - reduces hardware expenditures
  - extends the ability to support additional servers
  - makes load balancing and capacity planning much easier
- Without it
  - each host name requires a unique IP address, and we are quickly running out of IP addresses with the explosion of new domains

Format of Response



http message format: respone



Status Code

- The status code is a three-digit integer, and the first digit identifies the general category of response:
  - 1xx indicates an informational message
  - 2xx indicates success of some kind
  - 3xx redirects the client to another URL
  - 4xx indicates an error on the client's part
    - Yes, the system blames it on the client if a resource is not found (i.e., 404)
  - 5xx indicates an error on the server's part

Redirection Process

- Client asks for /foo, which is really a directory
- Server guesses that client meant /foo/ and so it replies with
  - 302 Moved
  - Location: /foo/
- Most browsers retry the new location automatically

Advantages of Redirection

- Simple Uses: Fix clients naming errors
- Complex Uses: Server can send client dynamically to a different page depending on
  - Who they are
  - What server is managing their session, etc.
- Note the changing URL in the browser

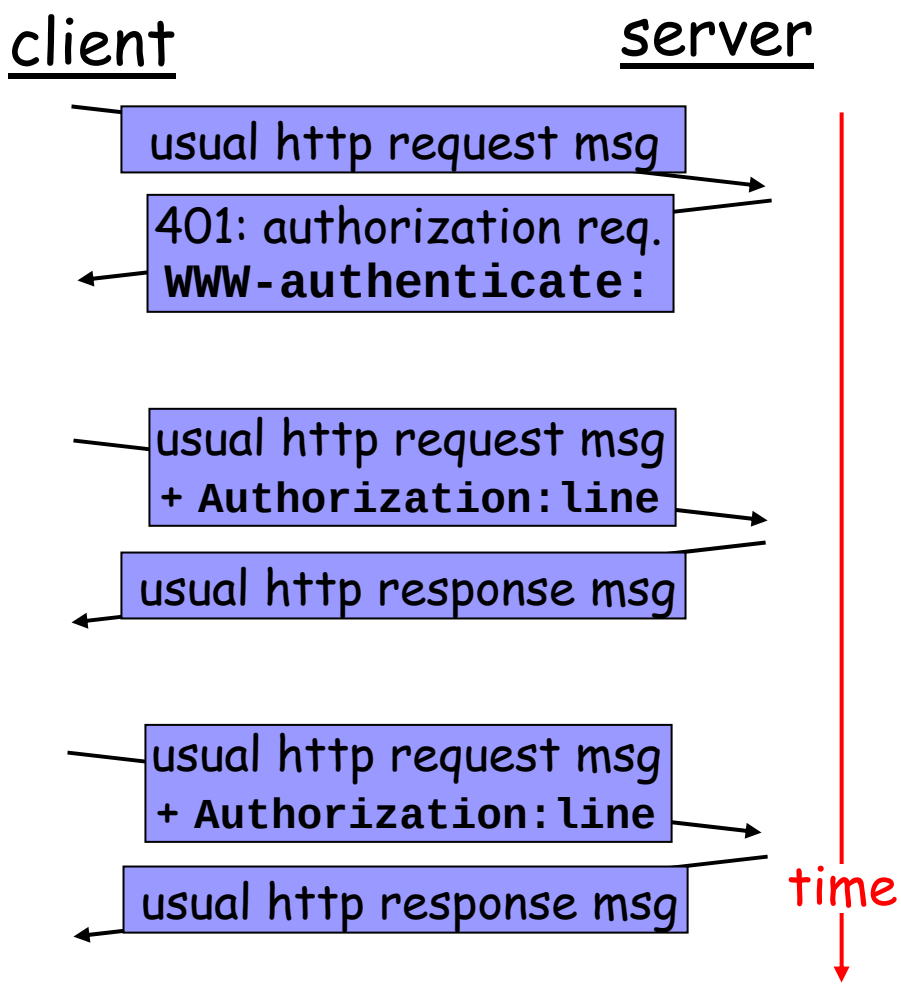
User-server interaction: authentication

Authentication goal: control access to server documents

- stateless: client must present authorization in each request
- authorization: typically name, password
  - **`authorization:`** header line in request
  - if no authorization presented, server refuses access, sends

    `WWW authenticate:`

    header line in response



Browser caches name & password so that user does not have to repeatedly enter it.

References

- V22.0480-001, Sana` Odeh , Computer Science Department, New York University.
- Representation and Management of Data on the Internet (67633), Yehoshua Sagiv, The Hebrew University - Institute of Computer Science.
- Java Network Programming and Distributed Computing, Reilly & Reilly.
- *Computer Networking: A Top-Down Approach Featuring the Internet,* Kurose & Rose, Pearson Addison Wesley.