

Chapter 2

Application Layer

Chapter 2: Application layer

2.1 Principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 Electronic Mail

- SMTP, POP3, IMAP

2.5 DNS

Chapter 2: Application Layer

Our goals:

- ❖ conceptual, implementation aspects of network application protocols
 - transport-layer service models
 - client-server paradigm
 - peer-to-peer paradigm
- ❖ learn about protocols by examining popular application-level protocols
 - HTTP
 - FTP
 - SMTP / POP3 / IMAP
 - DNS
- ❖ programming network applications
 - socket API

Some network apps

- ❖ e-mail
- ❖ web
- ❖ instant messaging
- ❖ remote login
- ❖ P2P file sharing
- ❖ multi-user network games
- ❖ streaming stored video (YouTube)
- ❖ voice over IP
- ❖ real-time video conferencing
- ❖ cloud computing
- ❖ ...

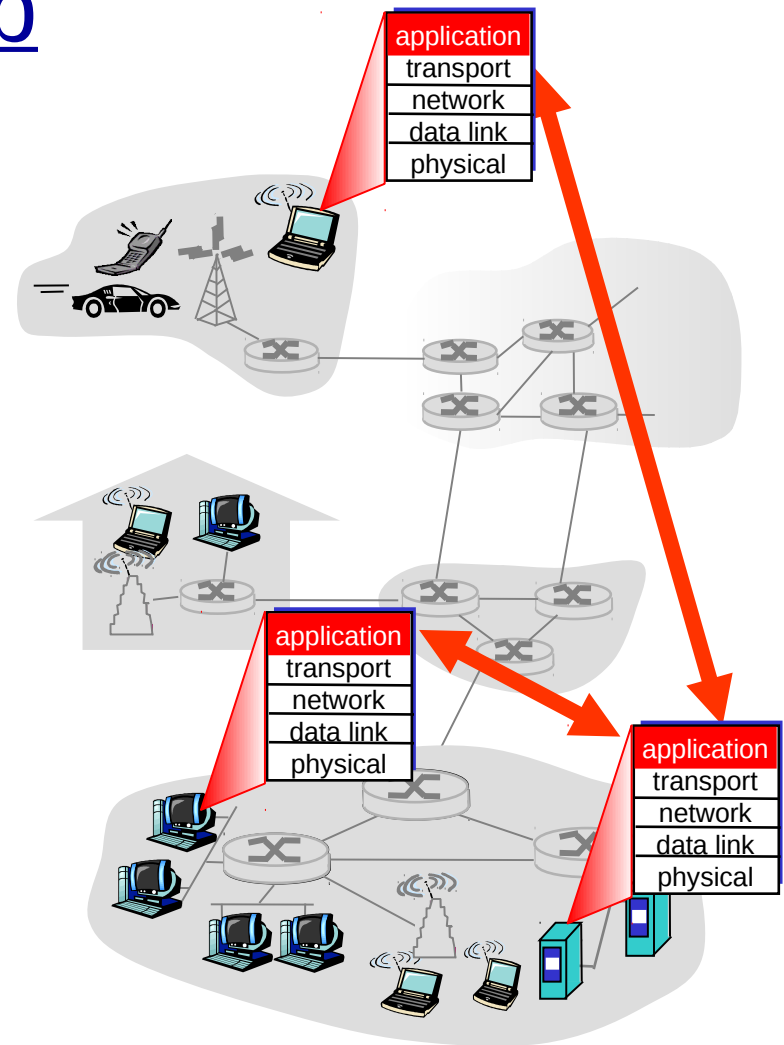
Creating a network app

write programs that

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

No need to write software for network-core devices

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



Chapter 2: Application layer

2.1 Principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 Electronic Mail

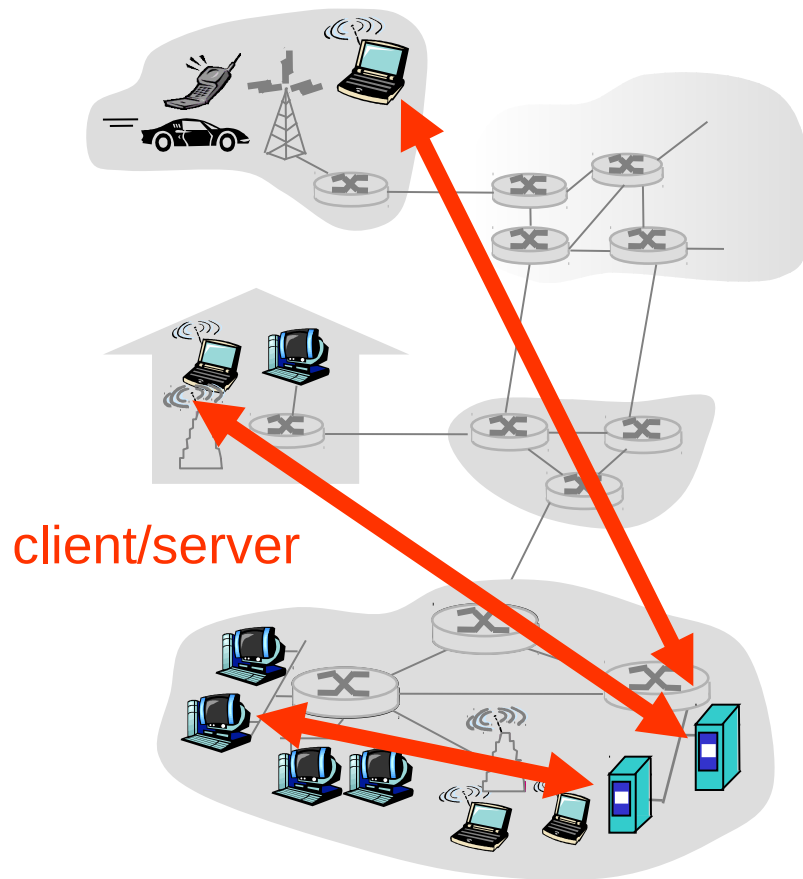
SMTP, POP3, IMAP

2.5 DNS

Application architectures

- ❖ client-server
- ❖ peer-to-peer (P2P)
- ❖ hybrid of client-server and P2P

Client-server architecture



server:

- always-on host
- permanent IP address
- server farms for scaling

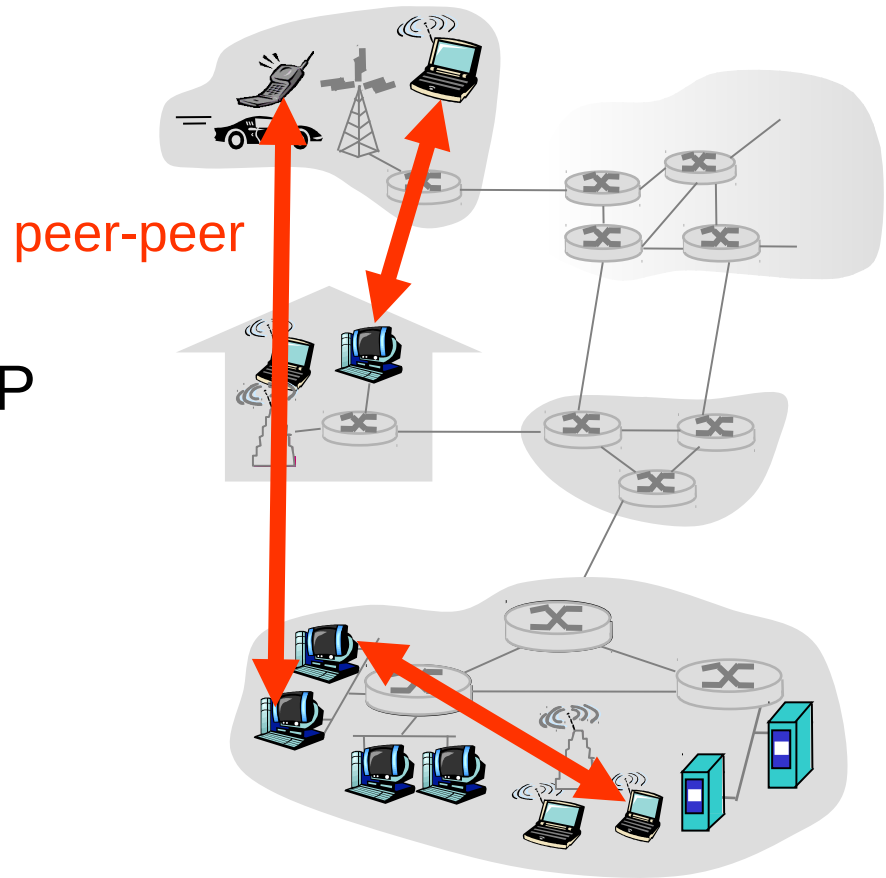
clients:

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

Pure P2P architecture

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

highly scalable but difficult to manage



Hybrid of client-server and P2P

Skype

- voice-over-IP P2P application
- centralized server: finding address of remote party:
- client-client connection: direct (not through server)

Instant messaging

- chatting between two users is P2P
- centralized service: client presence detection/location
 - user registers its IP address with central server when it comes online
 - user contacts central server to find IP addresses of buddies

Processes communicating

process: program running within a host.

- ❖ within same host, two processes communicate using **inter-process communication** (defined by OS).
- ❖ processes in different hosts communicate by exchanging **messages**

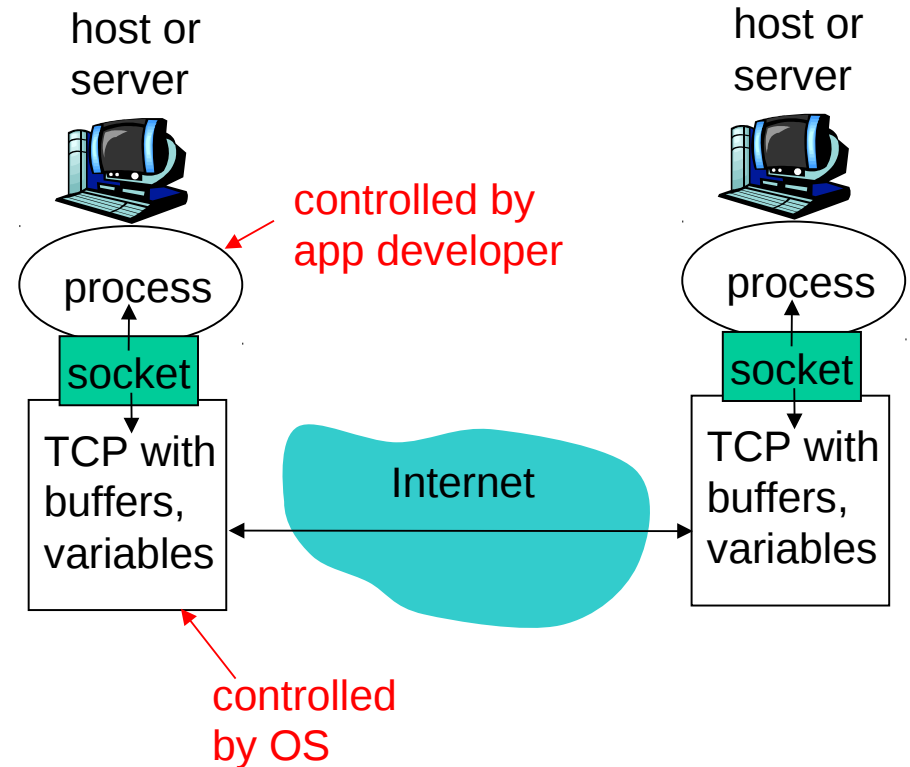
client process: process that initiates communication

server process: process that waits to be contacted

- ❖ aside: applications with P2P architectures have client processes & server processes

Sockets

- ❖ process sends/receives messages to/from its **socket**
- ❖ socket analogous to door
 - sending process shoves message out door
 - sending process relies on transport infrastructure on other side of door which brings message to socket at receiving process
- ❖ API: (1) choice of transport protocol; (2) ability to fix a few parameters (lots more on this later)



Addressing processes

- ❖ to receive messages, process must have *identifier*
- ❖ host device has unique 32-bit IP address
- ❖ Q: does IP address of host on which process runs suffice for identifying the process?

Addressing processes

- ❖ to receive messages, process must have *identifier*
- ❖ host device has unique 32-bit IP address
- ❖ Q: does IP address of host on which process runs suffice for identifying the process?
 - A: No, *many* processes can be running on same host
- ❖ *identifier* includes both **IP address** and **port numbers** associated with process on host.
- ❖ example port numbers:
 - HTTP server: 80
 - Mail server: 25
- ❖ to send HTTP message to gaia.cs.umass.edu web server:
 - **IP address:** 128.119.245.12
 - **Port number:** 80
- ❖ more shortly...

App-layer protocol defines

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

public-domain protocols:

- ❖ defined in RFCs
- ❖ allows for interoperability
- ❖ e.g., HTTP, SMTP

proprietary protocols:

- ❖ e.g., Skype

What transport service does an app need?

Data loss

- ❖ some apps (e.g., audio) can tolerate some loss
- ❖ other apps (e.g., file transfer, telnet) require 100% reliable data transfer

Timing

- ❖ some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”

Throughput

- ❖ some apps (e.g., multimedia) require minimum amount of throughput to be “effective”
- ❖ other apps (“**elastic apps**”) make use of whatever throughput they get

Security

- ❖ encryption, data integrity, ...

Transport service requirements of common apps

Application	Data loss	Throughput	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP service:

- ❖ *connection-oriented*: setup required between client and server processes
- ❖ *reliable transport* between sending and receiving process
- ❖ *flow control*: sender won't overwhelm receiver
- ❖ *congestion control*: throttle sender when network overloaded
- ❖ *does not provide*: timing, minimum throughput guarantees, security

UDP service:

- ❖ unreliable data transfer between sending and receiving process
- ❖ does not provide: connection setup, reliability, flow control, congestion control, timing, throughput guarantee, or security

Q: why bother? Why is there a UDP?

Internet apps: application, transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

Chapter 2: Application layer

2.1 Principles of network applications

- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 Electronic Mail

- SMTP, POP3, IMAP

2.5 DNS

Web and HTTP

First, a review...

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

www.someschool.edu/someDept/pic.gif

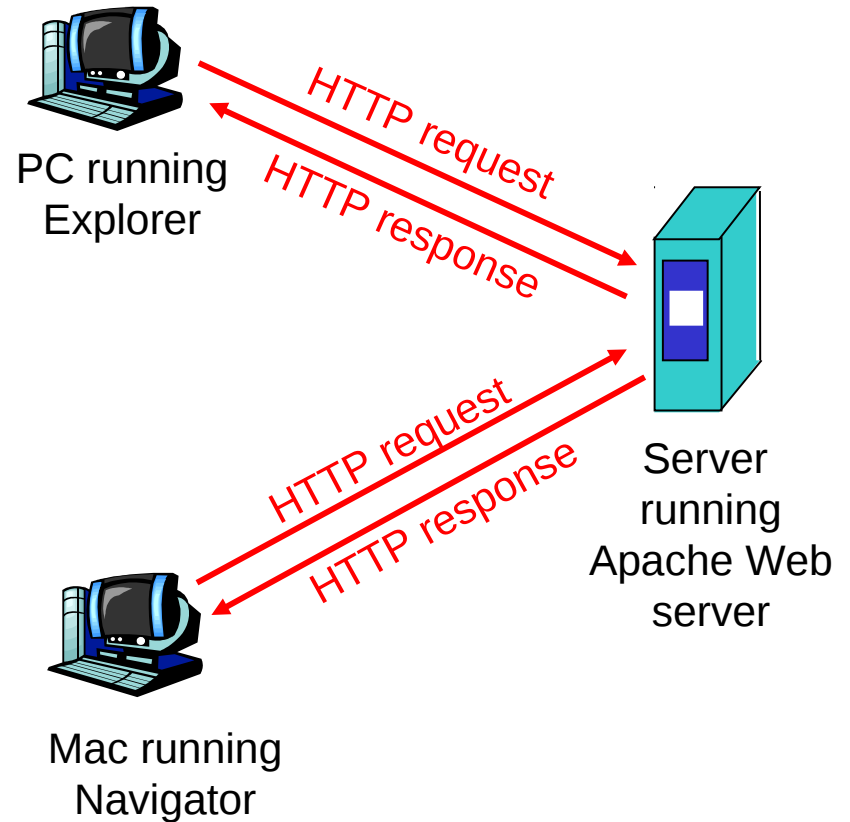
host name

path name

HTTP overview

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



HTTP overview (continued)

Uses TCP:

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

HTTP is “stateless”

- ❖ server maintains no information about past client requests

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

HTTP connections

non-persistent HTTP

- ❖ at most one object sent over TCP connection.

persistent HTTP

- ❖ multiple objects can be sent over single TCP connection between client, server.

Nonpersistent HTTP

suppose user enters **URL:**

www.someSchool.edu/someDepartment/home.index

(contains text,
references to 10
jpeg images)

1a. HTTP **client** initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

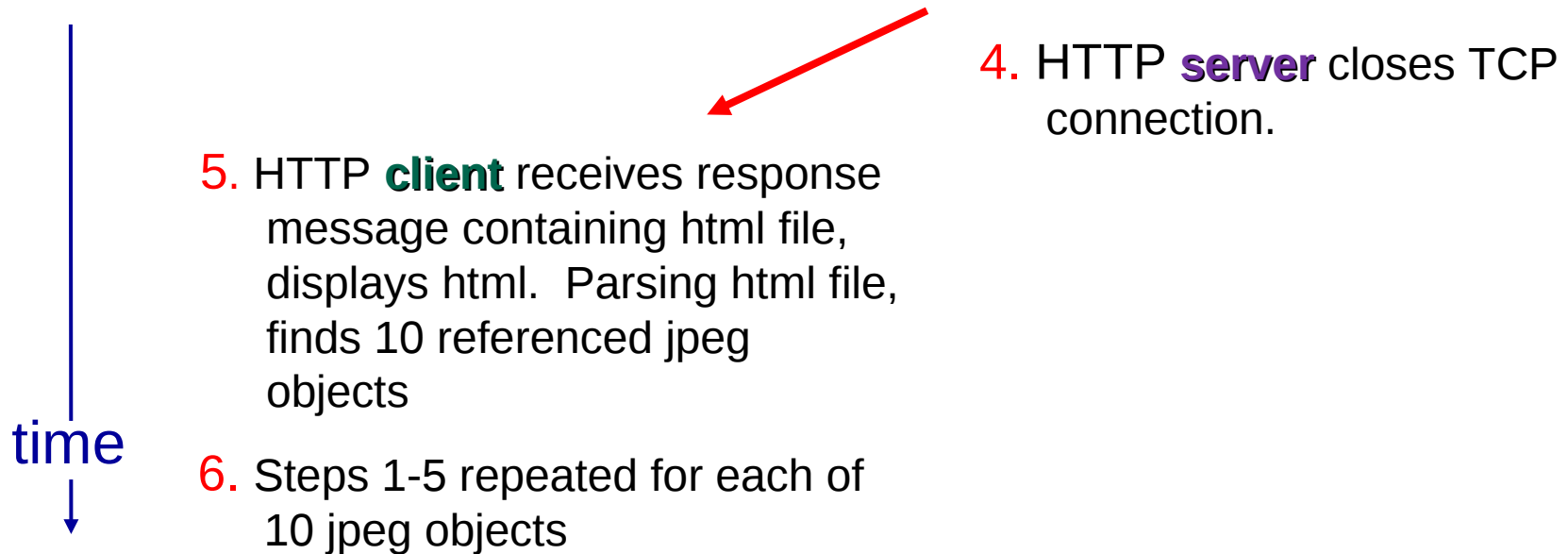
1b. HTTP **server** at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP **client** sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object **someDepartment/home.index**

3. HTTP **server** receives request message, forms *response message* containing requested object, and sends message into its socket

time
↓

Nonpersistent HTTP (cont.)



Non-Persistent HTTP: Response time

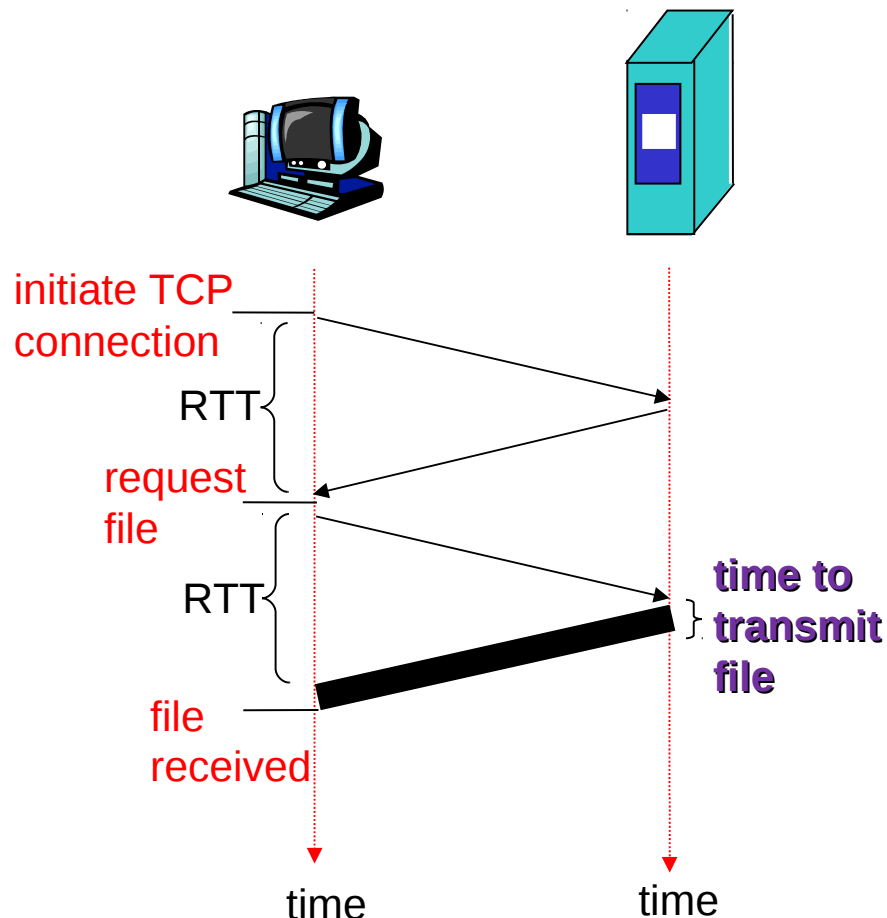
definition of **Round Trip**

Time: time for a small packet to travel from client to server and back.

response time:

- ❖ one RTT to initiate TCP connection
- ❖ one RTT for HTTP request and first few bytes of HTTP response to return
- ❖ file transmission time

total = 2RTT + transmit time



Persistent HTTP

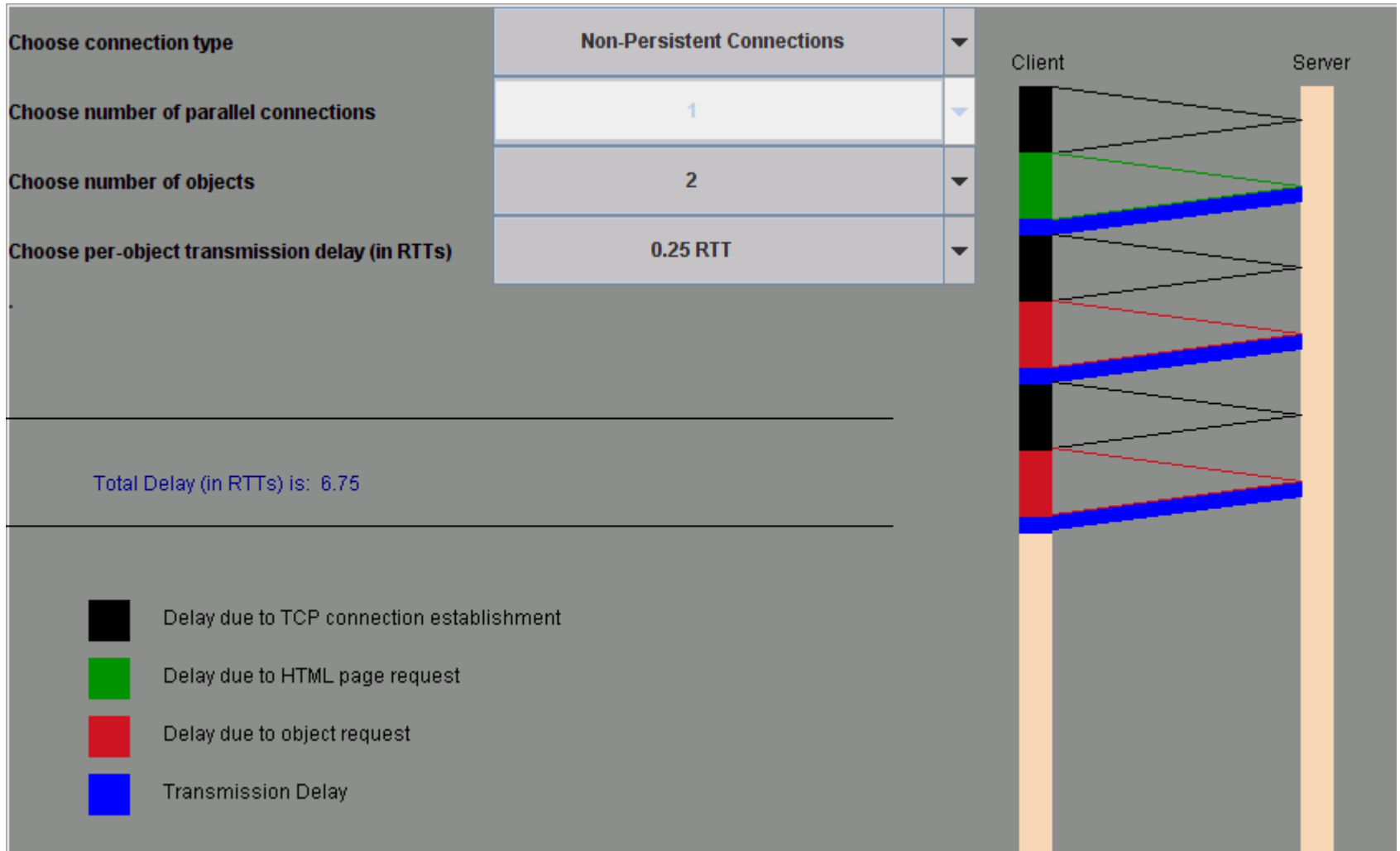
non-persistent HTTP issues:

- ❖ requires 2 RTTs per object
- ❖ OS overhead for *each* TCP connection
- ❖ browsers often open parallel TCP connections to fetch referenced objects

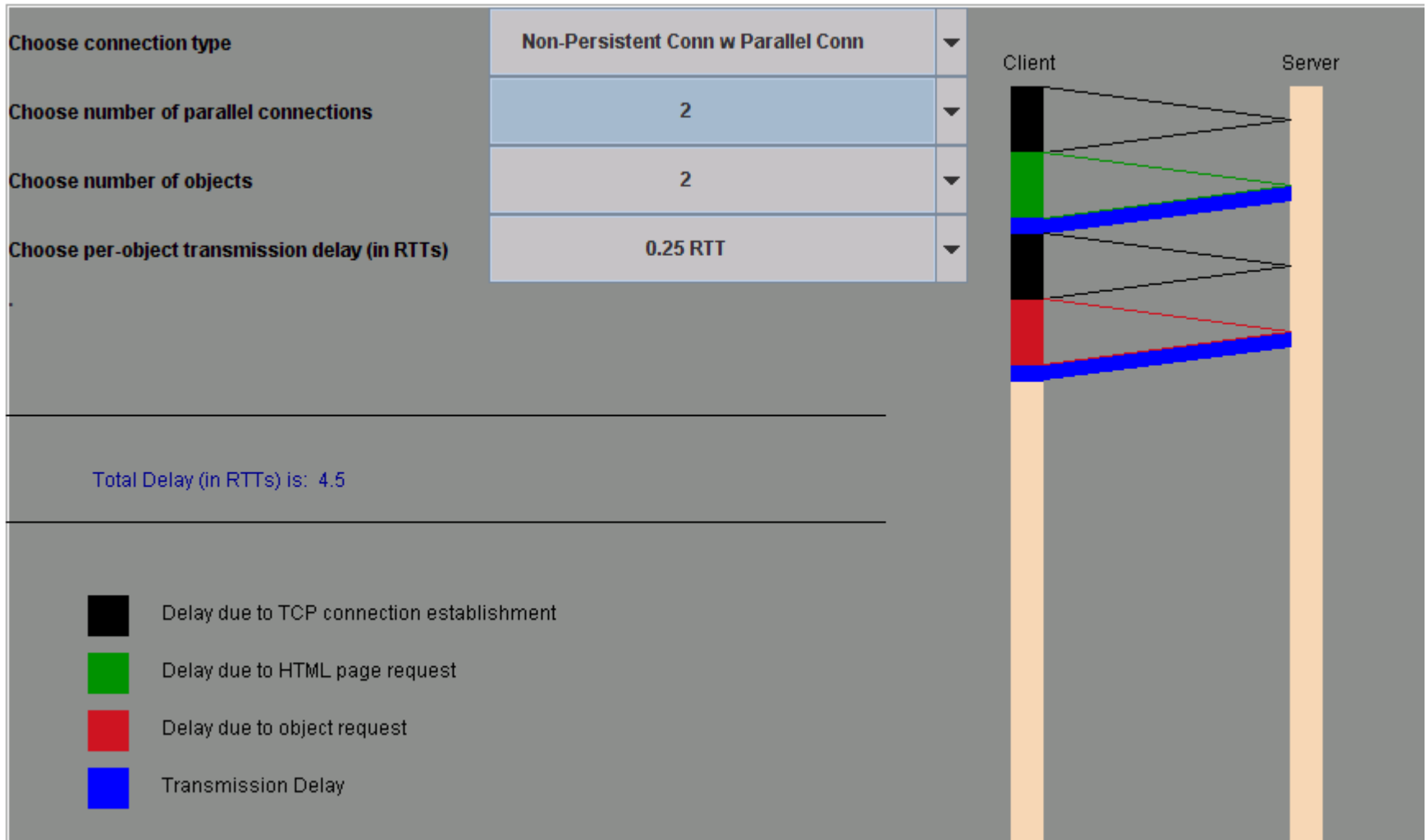
persistent HTTP

- ❖ server leaves connection open after sending response
- ❖ subsequent HTTP messages between same client/server sent over open connection
- ❖ client sends requests as soon as it encounters a referenced object
- ❖ as little as one RTT for all the referenced objects

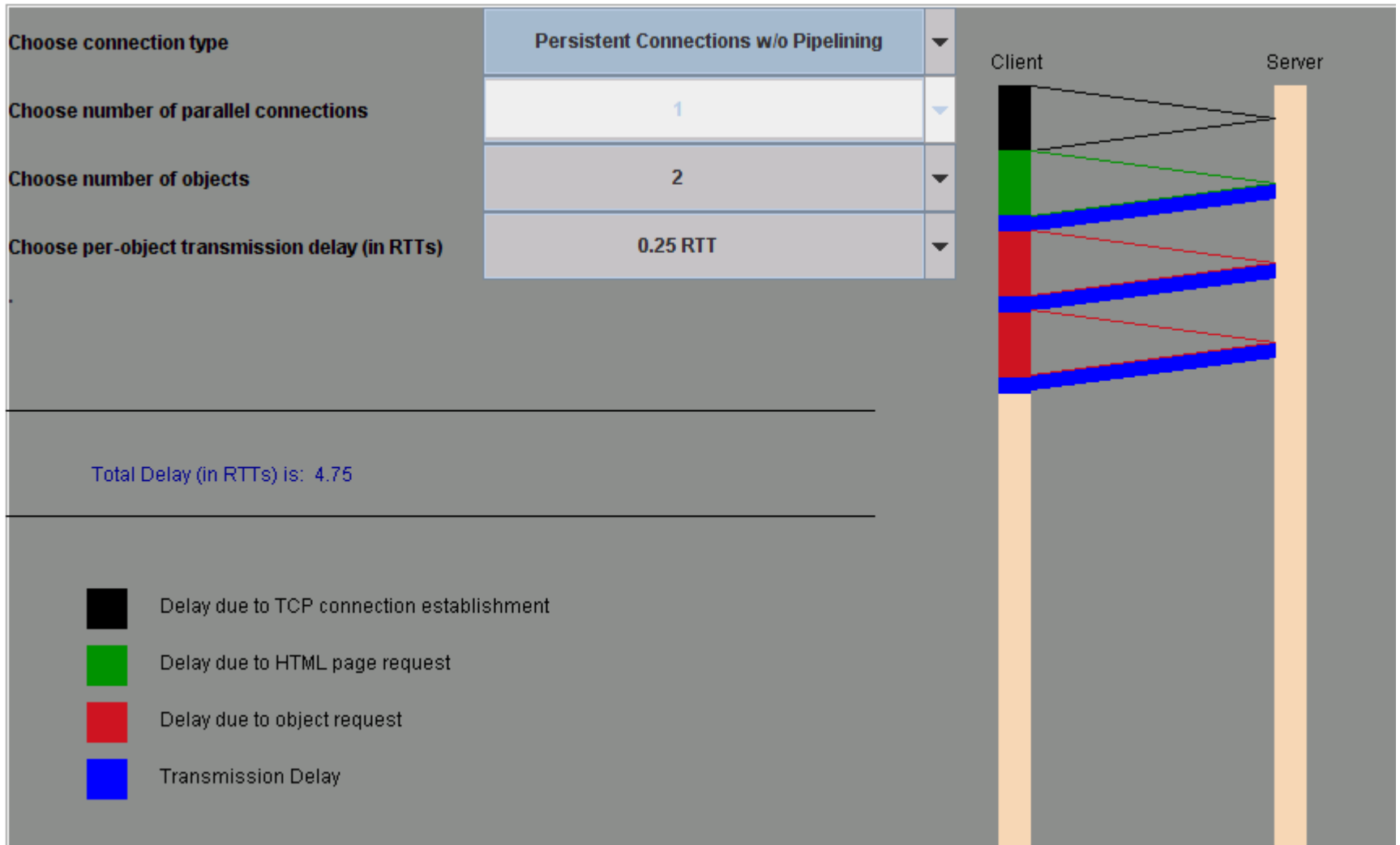
HTTP: example



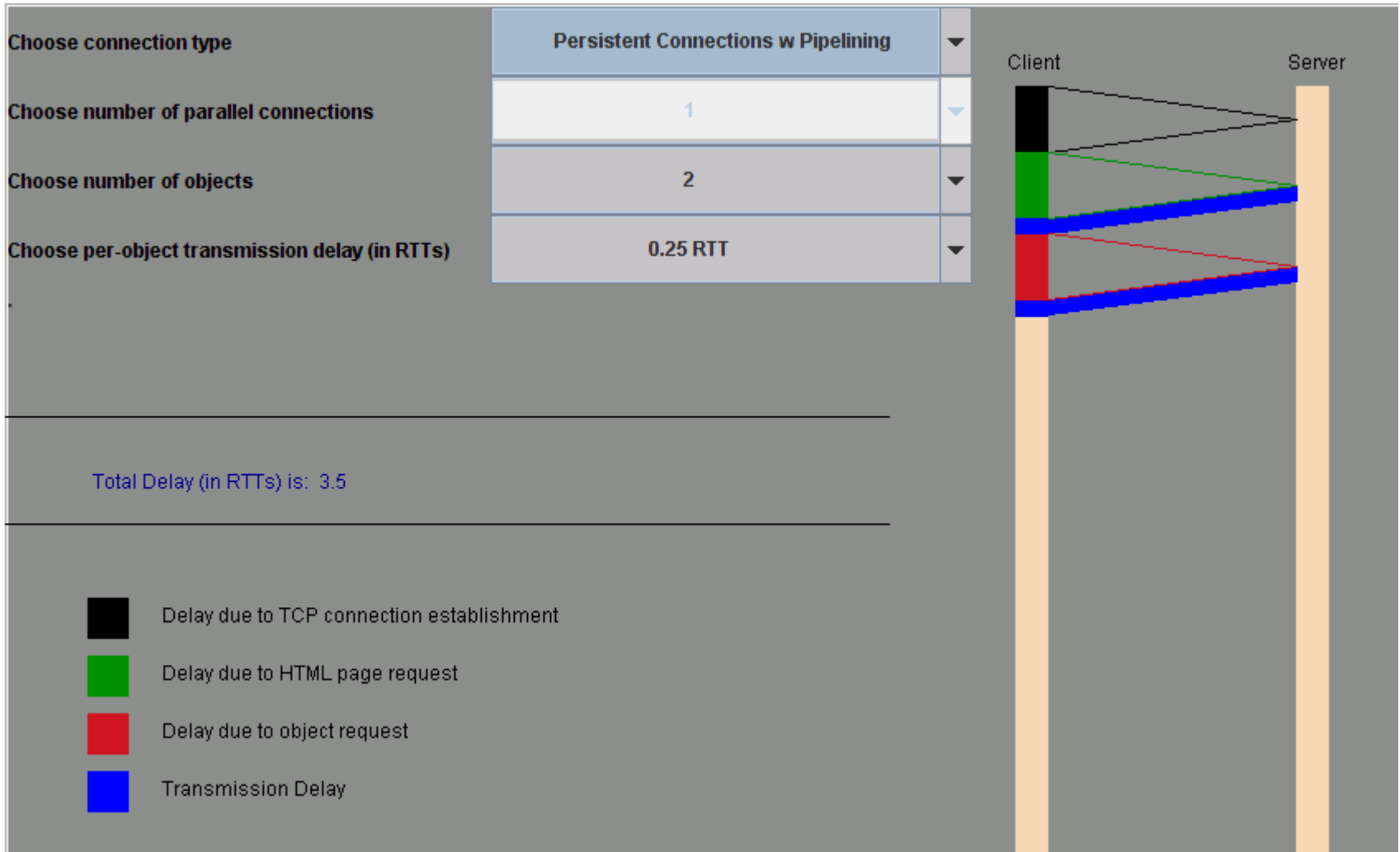
HTTP: example



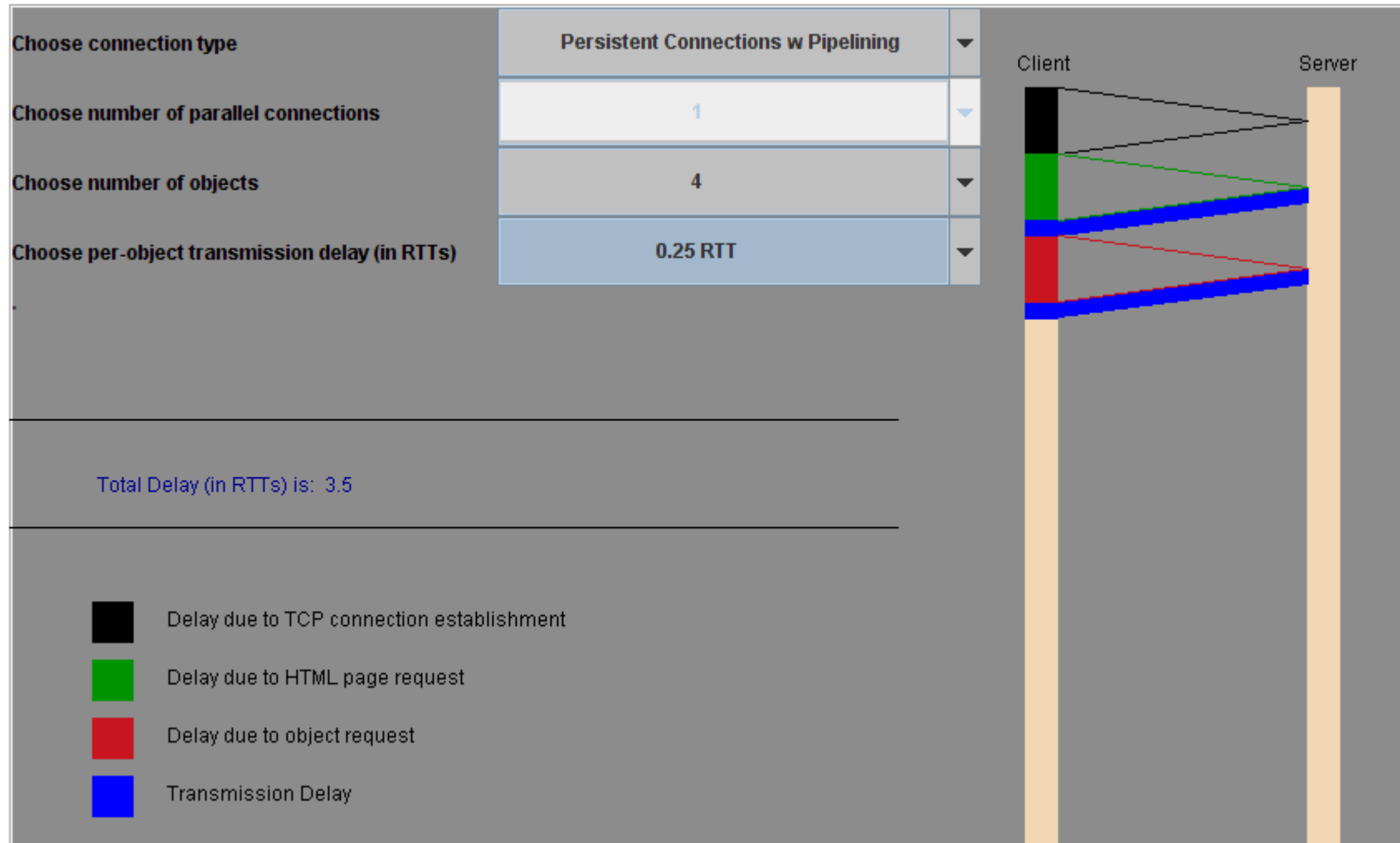
HTTP: example



HTTP: example



HTTP: example (4 objects)



HTTP request message

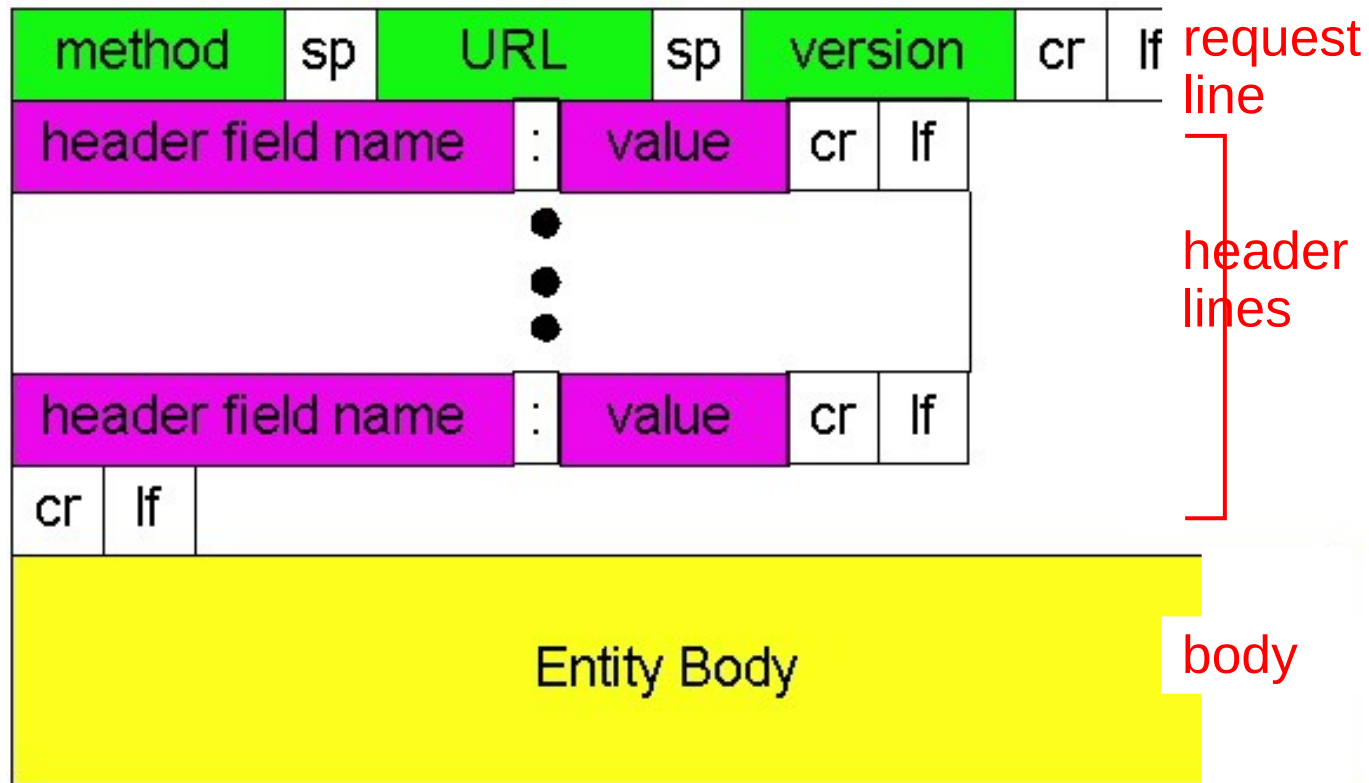
- ❖ two types of HTTP messages: *request, response*
- ❖ **HTTP request message:**
 - ASCII (human-readable format)

The diagram illustrates the structure of an HTTP request message. It shows a sequence of lines: a request line, followed by header lines, and ending with a blank line. Annotations with arrows point to specific parts of the message:

- request line (GET, POST, HEAD commands)**: Points to the first line of the message.
- header lines**: A bracket on the left side groups the lines from 'Host:' to 'Connection:'.
- carriage return, line feed at start of line indicates end of header lines**: Points to the blank line at the end of the header section.
- carriage return character**: Points to the '\r' character at the end of the first line.
- line-feed character**: Points to the '\n' character at the end of the first line.

```
GET /index.html HTTP/1.1\r\nHost: www-net.cs.umass.edu\r\nUser-Agent: Firefox/3.6.10\r\nAccept: text/html,application/xhtml+xml\r\nAccept-Language: en-us,en;q=0.5\r\nAccept-Encoding: gzip,deflate\r\nAccept-Charset: ISO-8859-1,utf-8;q=0.7\r\nKeep-Alive: 115\r\nConnection: keep-alive\r\n\r\n
```

HTTP request message: general format



Uploading form input

POST method:

- web page often includes form input
- ❖ input is uploaded to server in entity **body**

URL method:

- ❖ uses GET method
- ❖ input is uploaded in URL field of request line:

`www.somesite.com/page1exec?var1=value1&var2=value2`

Uploading form input

The server

sees:

GET method:

GET /path/page1exec?var1=value1&var2=value2

HTTP/1.0

User-Agent: HTTPTool/1.0

Content-Type:

application/x-www-form-urlencoded

POST method:

POST /path/page1exec HTTP/1.0

User-Agent: HTTPTool/1.0

Content-Type:

application/x-www-form-urlencoded

Content-Length: 23

var1=value1&var2=value2

Uploading form input

How do you get the server to receive var1 and var2?

Short answer: depends on the server and application

Long answer: 159339 Internet Programming

E.g., CGI "scripts":

pageexec is an executable in /var/www/cgi-bin
using Apache in Linux,

GET method:

```
char * data;  
data = getenv("QUERY_STRING");//here it is your data!!!
```

POST method:

```
getenv("CONTENT_LENGTH");  
....  
sscanf(input,"var1=%ld&var2=%ld",&var1,&var2);
```

Method types

HTTP/1.0

- ❖ GET
- ❖ POST
- ❖ HEAD
 - asks server to leave requested object out of response

HTTP/1.1


- ❖ GET, POST, HEAD
- ❖ PUT
 - uploads file in entity body to path specified in URL field
- ❖ DELETE
 - deletes file specified in the URL field

HTTP response message

status line
(protocol
status code
status phrase)

header
lines

data, e.g.,
requested
HTML file



The diagram shows an HTTP response message structure. A red arrow points from the 'status line' label to the first line of the message. A red bracket on the left side groups the subsequent lines as 'header lines'. Another red arrow points from the 'data' label to the final line of the message.

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
data data data data data ...
```


HTTP response status codes

- ❖ status code appears in 1st line in server->client response message.
- ❖ some sample codes:

200 OK

- request succeeded, requested object later in this msg

301 Moved Permanently

- requested object moved, new location specified later in this msg (Location:)

400 Bad Request

- request msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

telnet www.massey.ac.nz 80

opens TCP connection to port 80
(default HTTP server port)
anything typed in sent
to port 80 at www.massey.ac.nz

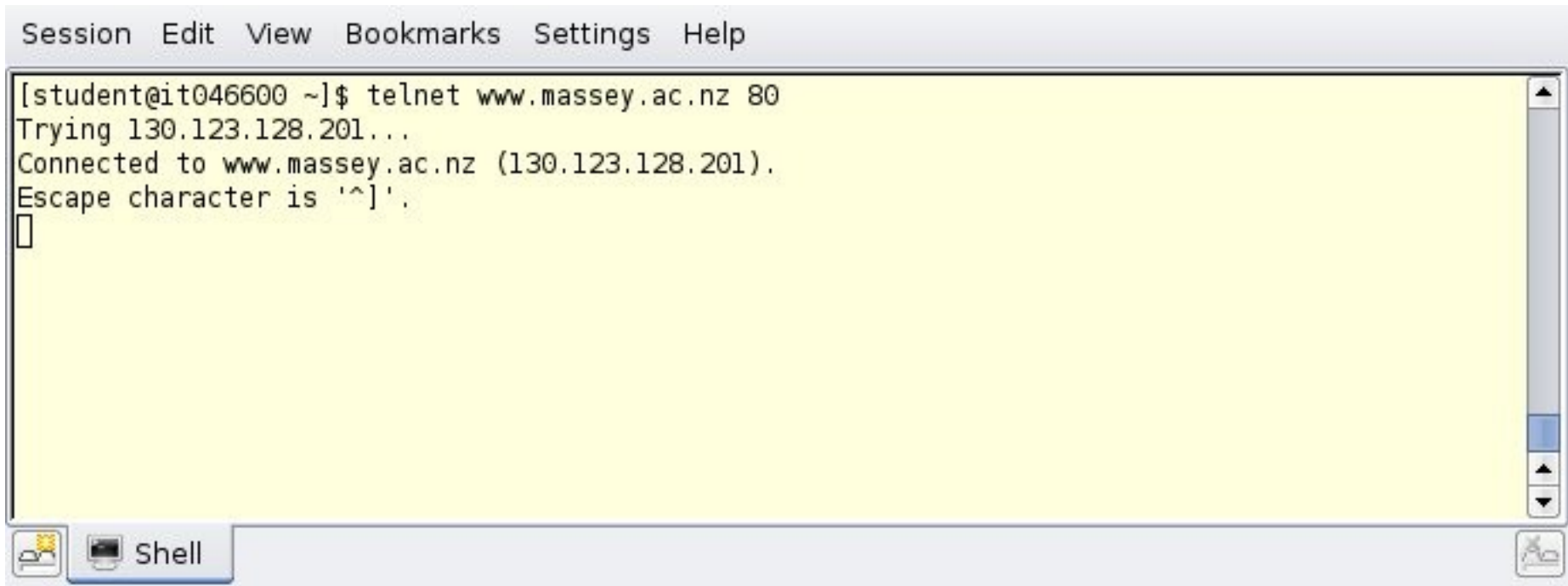
2. type in a GET HTTP request:

GET /

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!

Using **telnet** to test HTTP



A screenshot of a terminal window with a menu bar (Session, Edit, View, Bookmarks, Settings, Help) and a yellow background. The terminal shows the following text: [student@it046600 ~]\$ telnet www.massey.ac.nz 80, Trying 130.123.128.201..., Connected to www.massey.ac.nz (130.123.128.201), Escape character is '^]', and a cursor on the next line. At the bottom, there is a 'Shell' tab and a status bar.

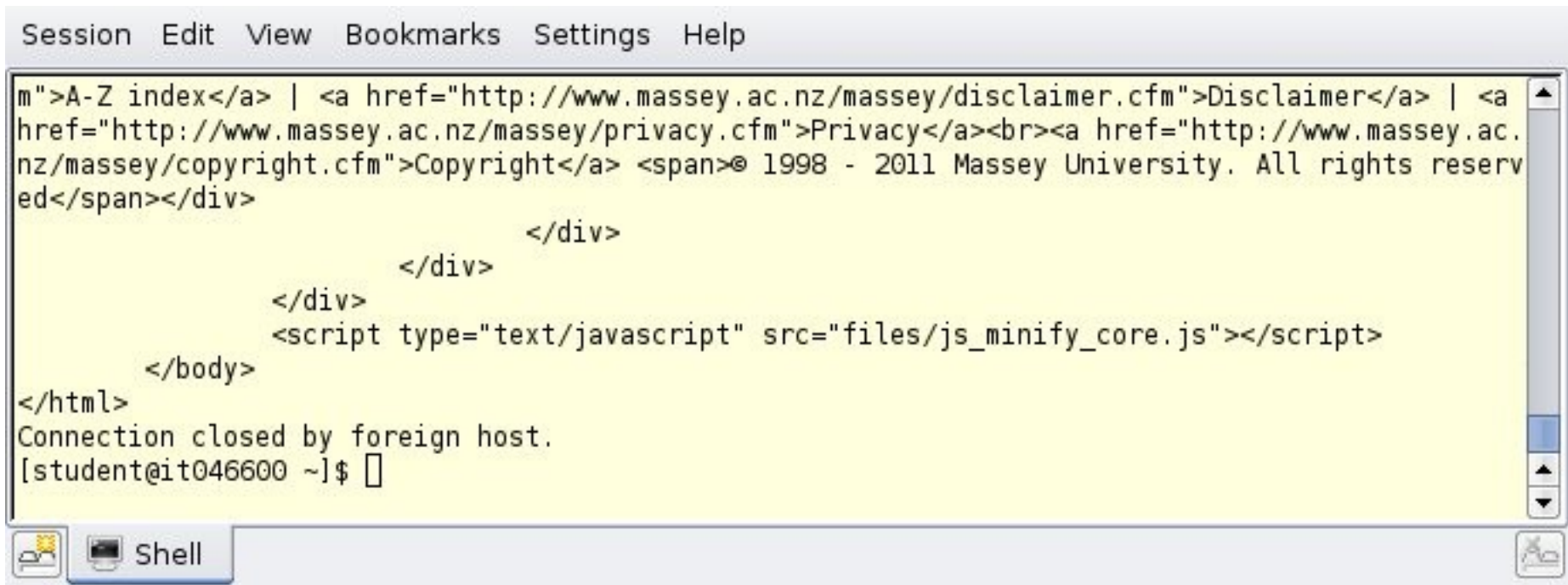
```
[student@it046600 ~]$ telnet www.massey.ac.nz 80
Trying 130.123.128.201...
Connected to www.massey.ac.nz (130.123.128.201).
Escape character is '^]'.
█
```

Type: GET /

Press the **<Enter>** key twice afterwards.

Using **telnet** to test HTTP

Here's the response from the server, it's the html file:



The screenshot shows a telnet session window with a menu bar (Session, Edit, View, Bookmarks, Settings, Help) and a text area containing an HTTP response. The response is an HTML document from Massey University. The text in the terminal is as follows:

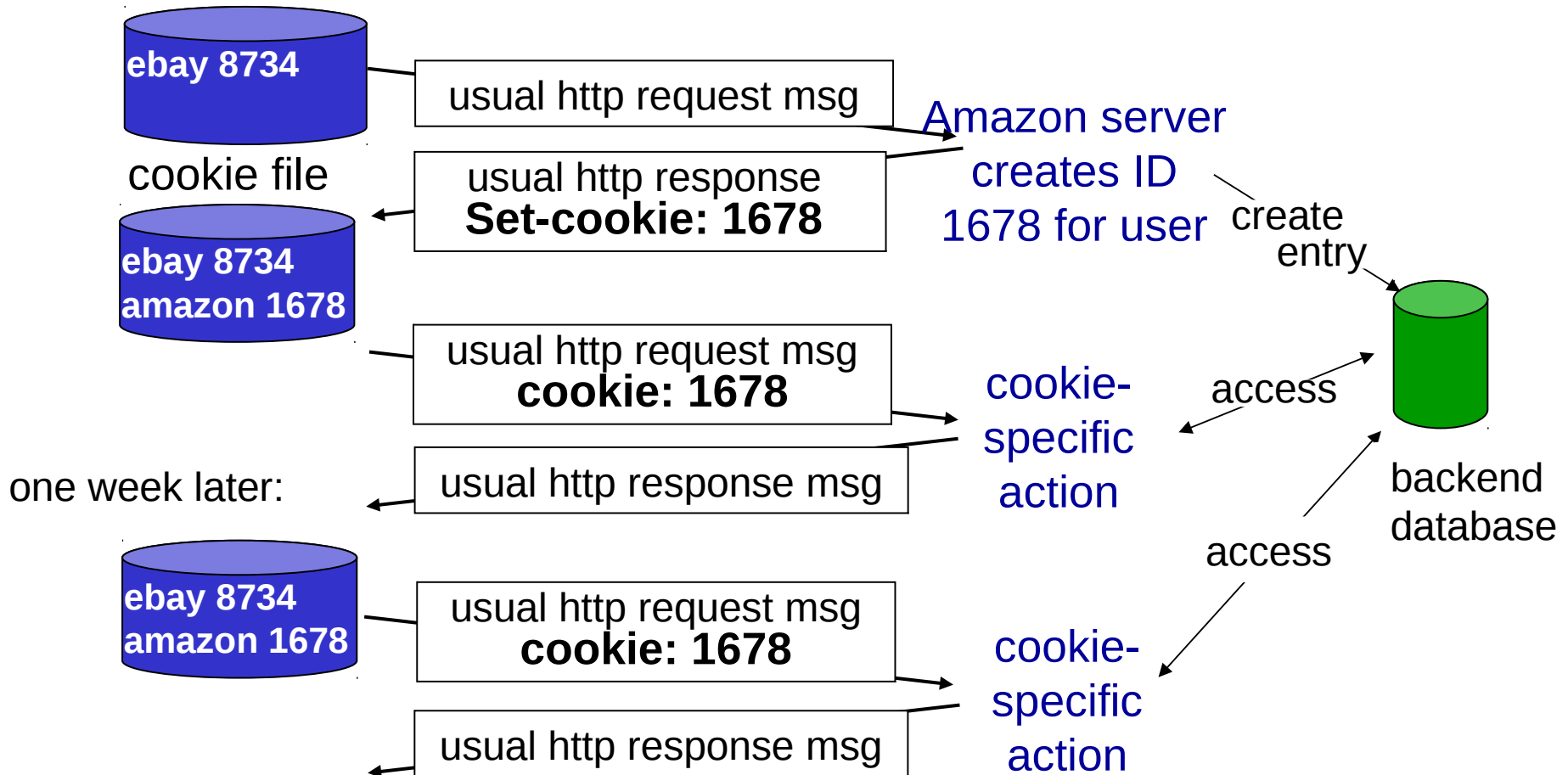
```
Session Edit View Bookmarks Settings Help
m">A-Z index</a> | <a href="http://www.massey.ac.nz/massey/disclaimer.cfm">Disclaimer</a> | <a
href="http://www.massey.ac.nz/massey/privacy.cfm">Privacy</a><br><a href="http://www.massey.ac.
nz/massey/copyright.cfm">Copyright</a> <span>© 1998 - 2011 Massey University. All rights reserv
ed</span></div>
                                </div>
                        </div>
                </div>
        <script type="text/javascript" src="files/js_minify_core.js"></script>
</body>
</html>
Connection closed by foreign host.
[student@it046600 ~]$
```

At the bottom of the window, there is a 'Shell' button and a small icon of a computer monitor.

Cookies: keeping “state” (cont.)

client

server



Cookies (continued)

what cookies can bring:

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

how to keep “state”:

- ❖ protocol endpoints: maintain state at sender/receiver over multiple transactions
- ❖ cookies: http messages carry state

aside

cookies and privacy:

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

User-server state: cookies

many Web sites use cookies

four components:

- 1) cookie header line of HTTP *response* message
- 2) cookie header line in HTTP *request* message
- 3) cookie file kept on user's host, managed by user's browser
- 4) back-end database at Web site

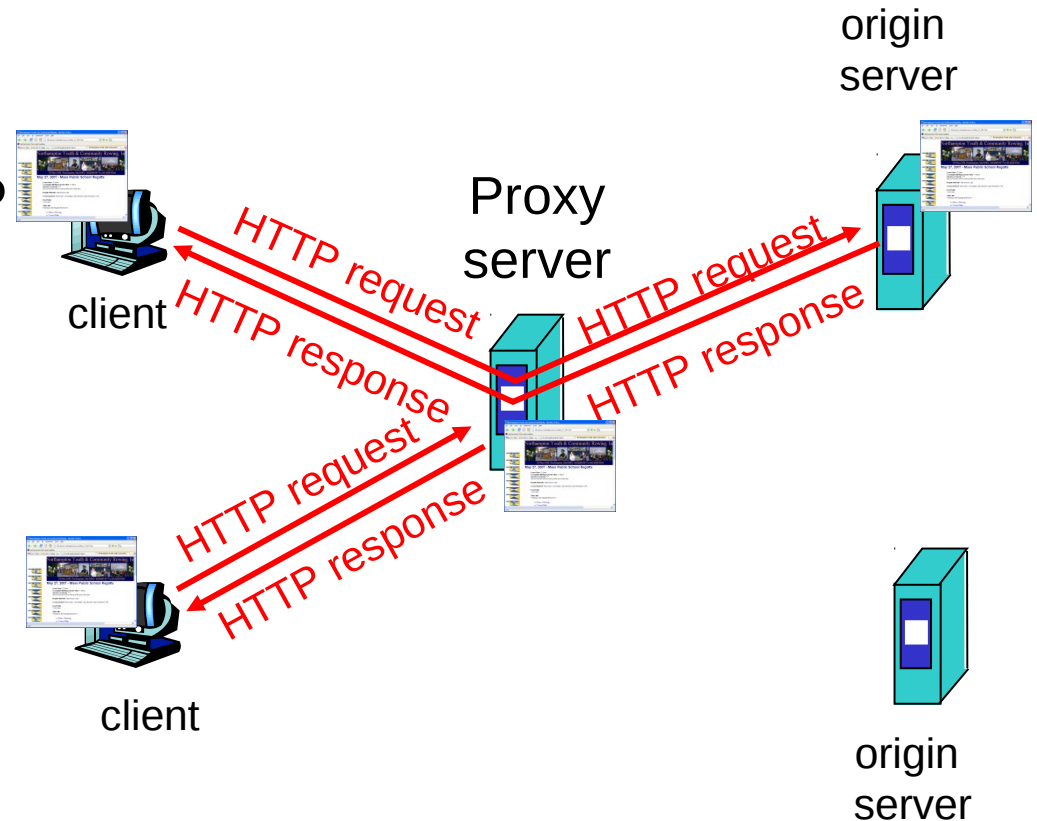
example:

- ❖ Susan always access Internet from PC
- ❖ visits specific e-commerce site for first time
- ❖ when initial HTTP requests arrives at site, site creates:
 - unique ID
 - entry in backend database for ID

Web caches (proxy server)

Goal: satisfy client request without involving origin server

- ❖ user sets browser: Web accesses via cache
- ❖ browser sends all HTTP requests to cache
 - If the object is in cache: cache returns object
 - else cache requests object from origin server, then returns object to client



More about Web caching

- ❖ cache acts as both client and server
- ❖ 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 (but so does P2P file sharing)

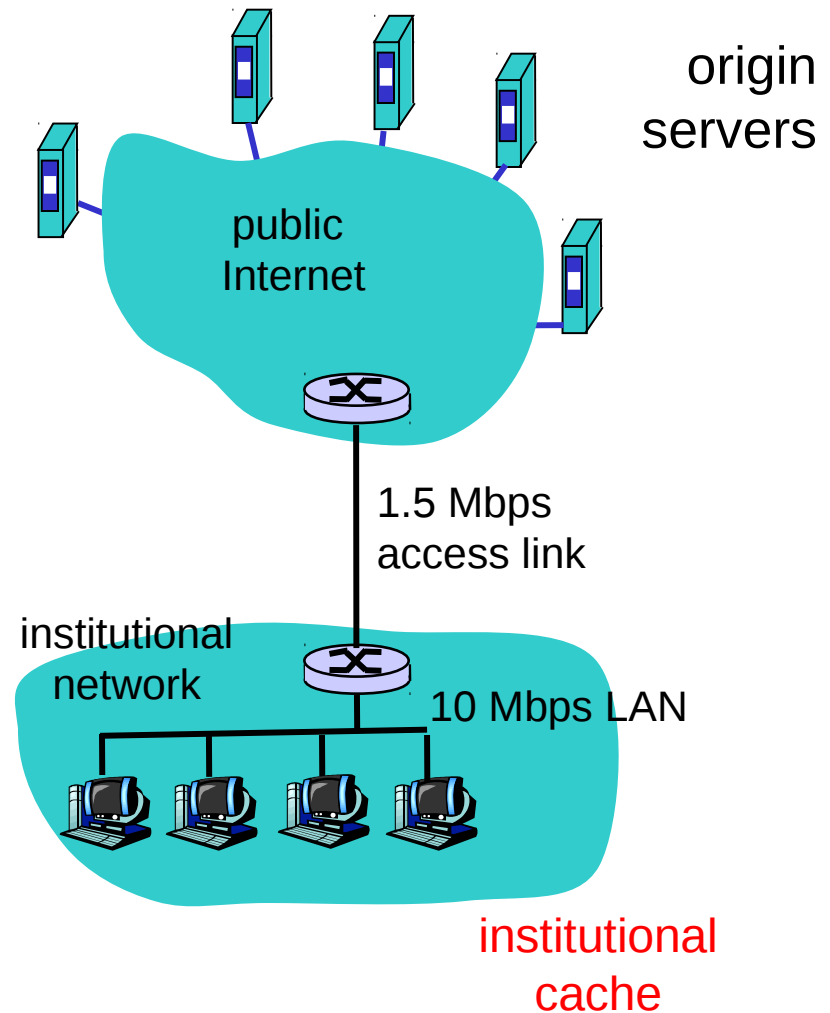
Caching example

assumptions

- ❖ average object size = 100,000 bits
- ❖ avg. request rate from institution's browsers to origin servers = 15/sec
- ❖ delay from institutional router to any origin server and back to router = 2 sec

consequences

- ❖ utilization on LAN = 15%
- ❖ utilization on access link = 100%
- ❖ total delay = Internet delay + access delay + LAN delay
= 2 sec + minutes + milliseconds



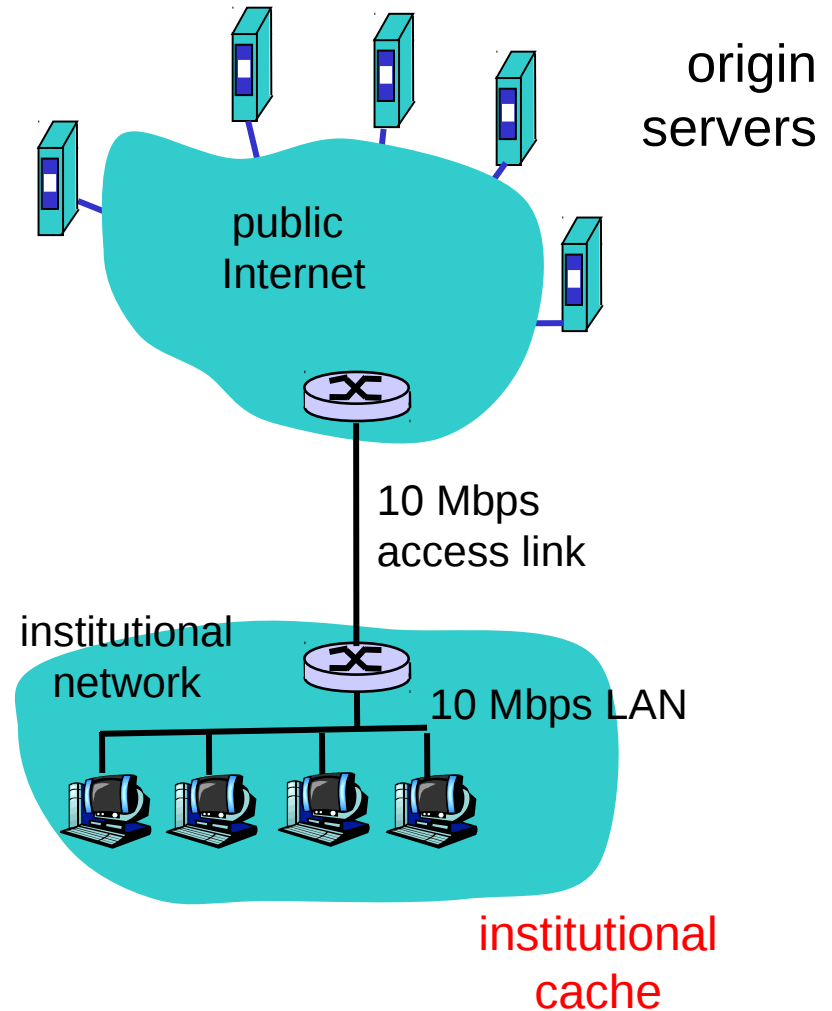
Caching example (cont)

possible solution

- ❖ increase bandwidth of access link to, say, 10 Mbps

consequence

- ❖ utilization on LAN = 15%
- ❖ utilization on access link = 15%
- ❖ Total delay = Internet delay + access delay + LAN delay
= 2 sec + msecs + msecs
- ❖ often a costly upgrade



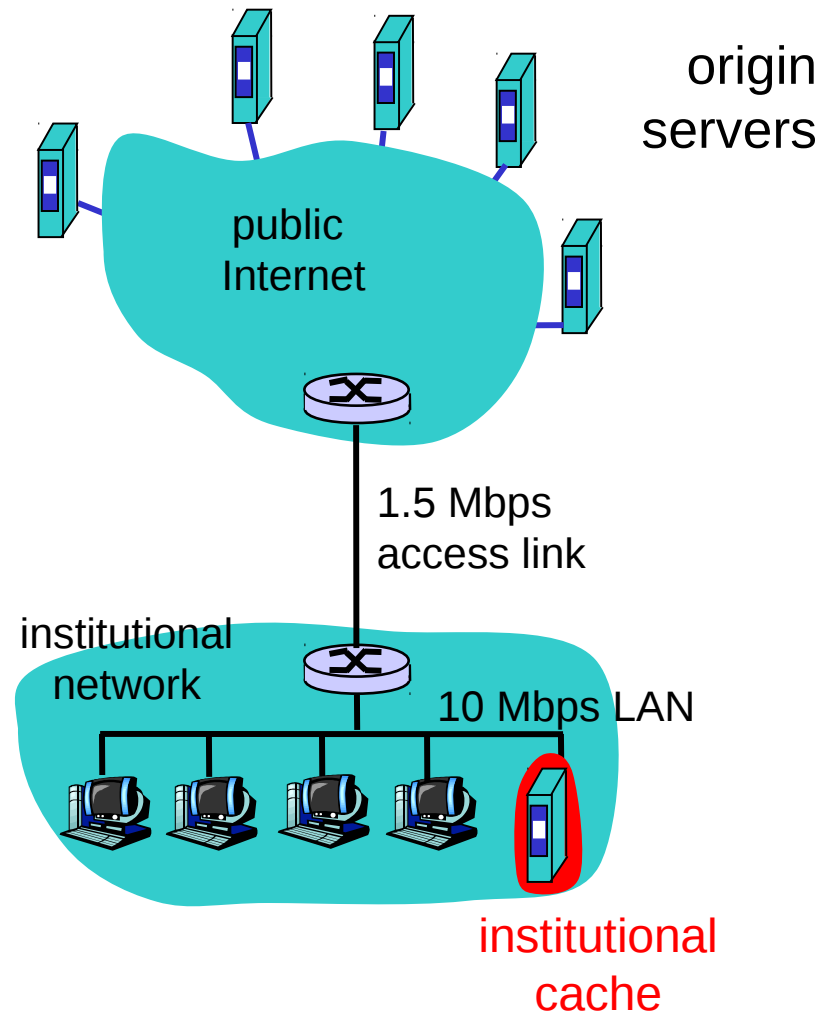
Caching example (cont)

possible solution:

install cache

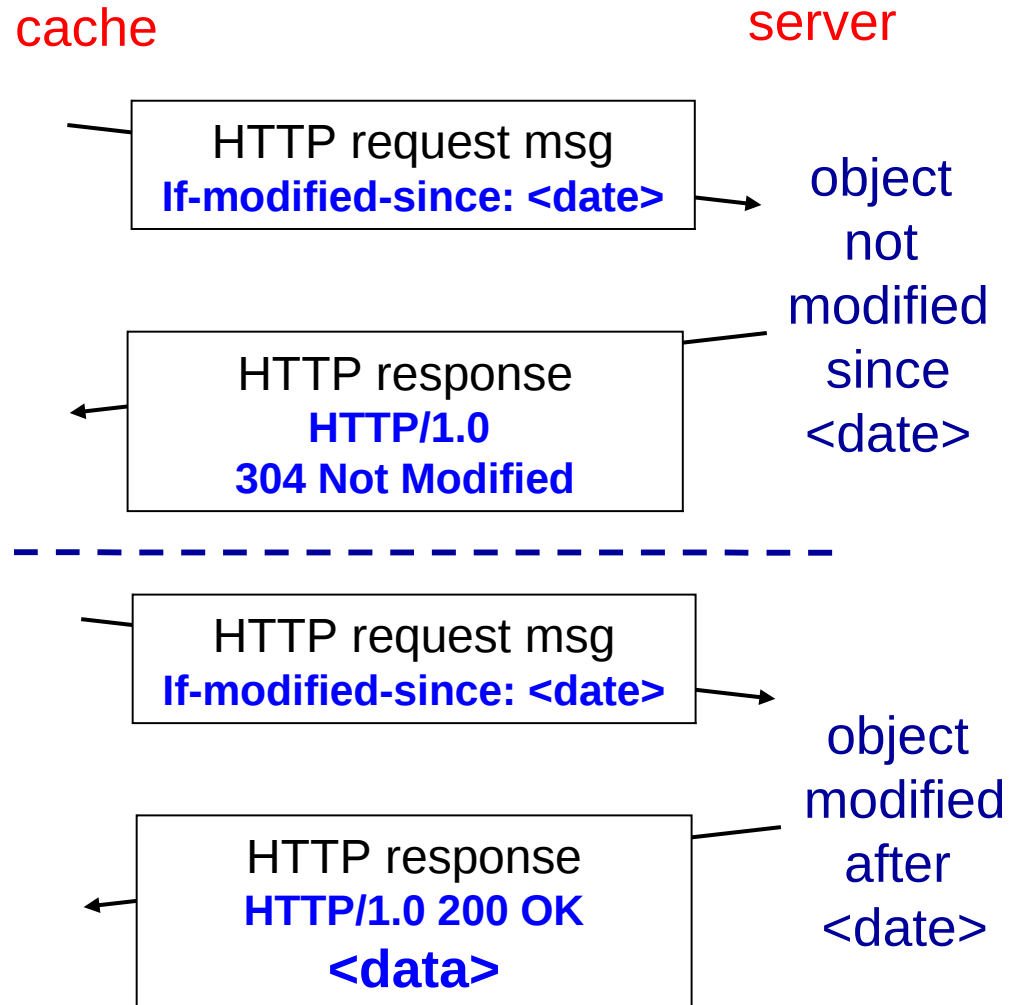
consequence

- ❖ suppose hit rate is 0.4
 - 40% requests will be satisfied almost immediately
 - 60% requests satisfied by origin server
- ❖ utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- ❖ total avg delay = Internet delay + access delay + LAN delay =
 $.6 \cdot (2.01) \text{ secs} + .4 \cdot \text{milliseconds}$
- ❖ $< 1.4 \text{ secs}$



Conditional GET

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



Chapter 2: Application layer

2.1 Principles of network applications

2.2 Web and HTTP

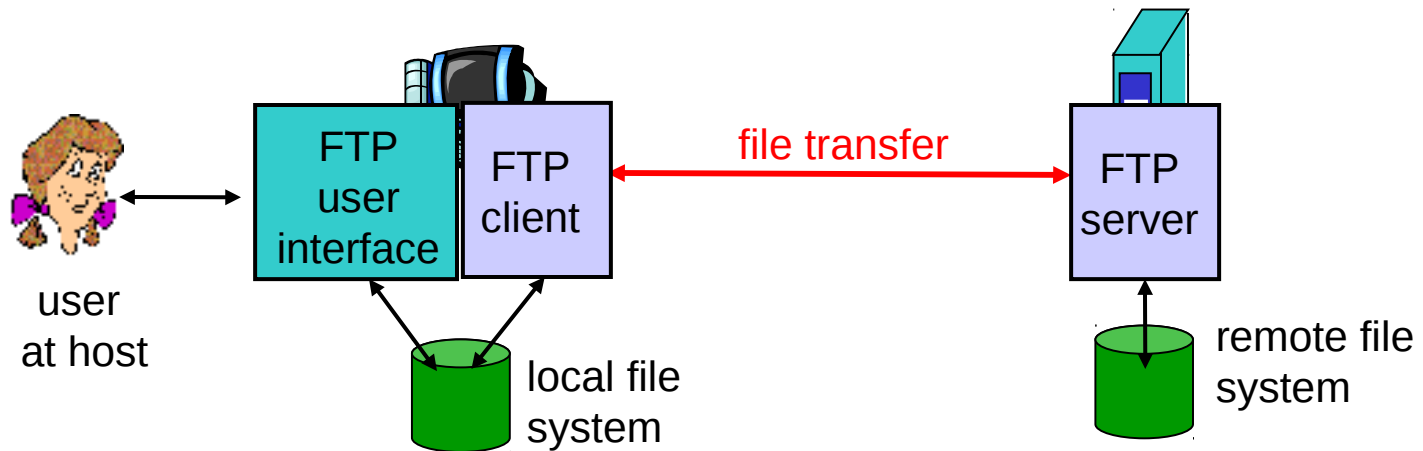
2.3 FTP

2.4 Electronic mail

- SMTP, POP3, IMAP

2.5 DNS

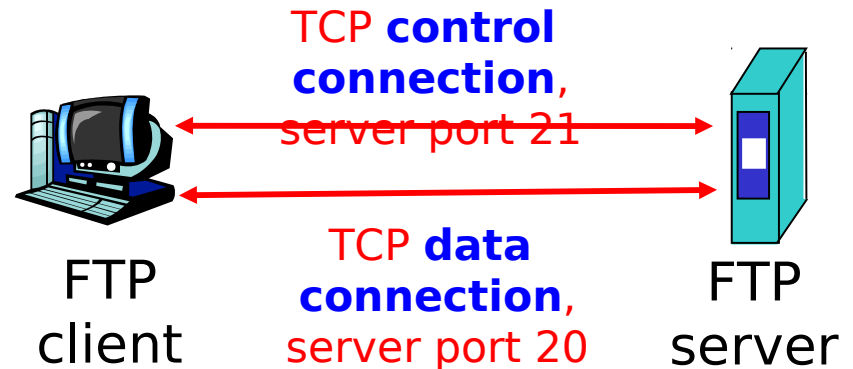
FTP: the file transfer protocol



- ❖ 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, TCP is transport protocol
- ❖ client authorized over control connection
- ❖ client browses remote directory by sending commands over control connection.
- ❖ when server receives file transfer command, server opens 2nd TCP connection (for file) to client
- ❖ after transferring one file, server closes data connection.



- ❖ server opens another 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

FTP commands different than application commands!!!

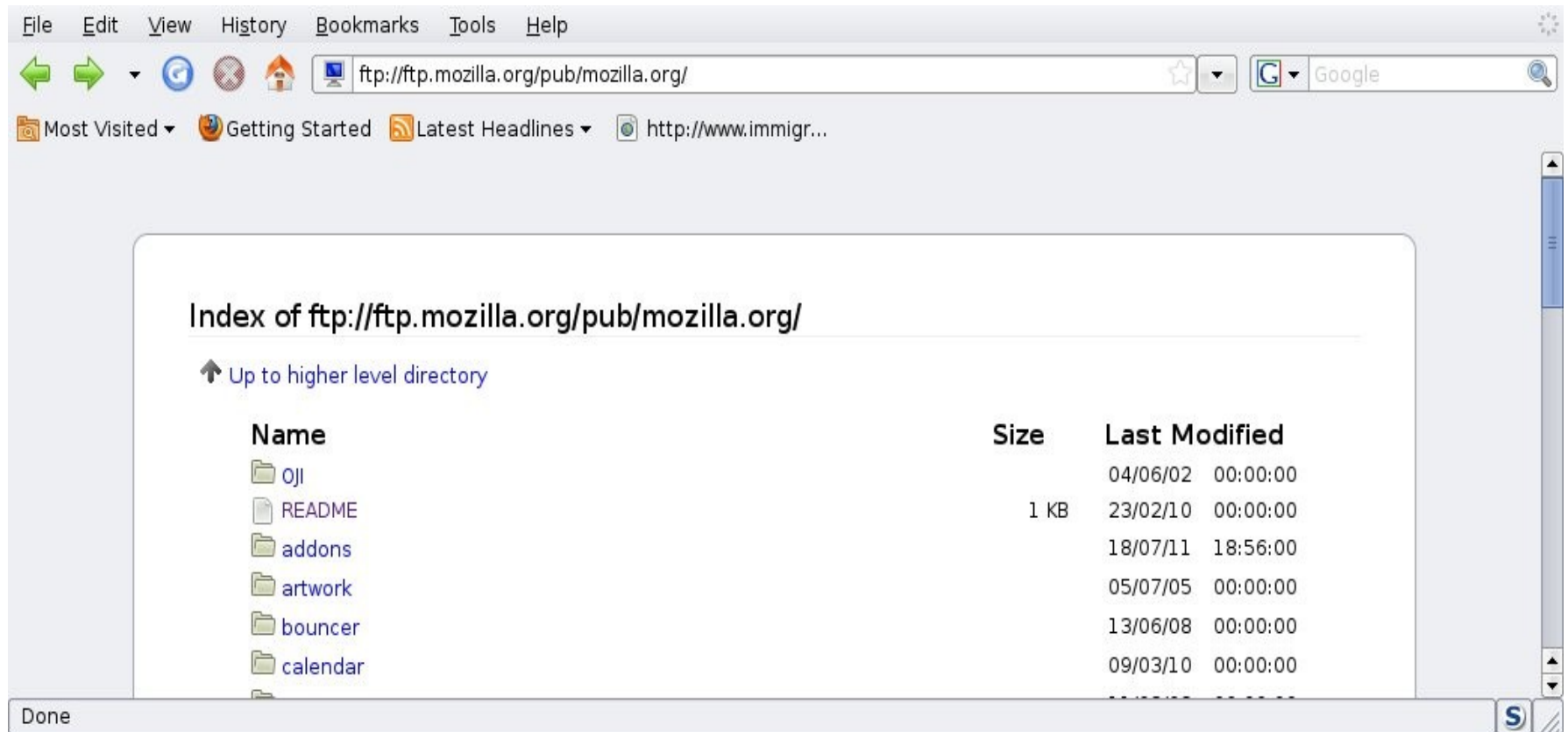
Source of confusion

- ❖ FTP protocols
- ❖ FTP applications...
- ❖ The applications are a front-end for users to issue an FTP command
- ❖ e.g., command line FTP may issue a certain protocol command
- ❖ FTP via browser may have buttons to issue the same protocol command

FTP clients

- ❖ FTP clients come in different packages
- ❖ Windows X Unix X Macs?
- ❖ Command line X browsers
- ❖ Servers understand some, not all
- ❖ Some client are really bad in terms of following the standards.
- ❖ Try using command-line ftp clients, it is useful

FTP clients: firefox



FTP clients: command line

Ftp client commands

- ❖ help
- ❖ ls (list files)
- ❖ cd (change directories)
- ❖ get (download files)
- ❖ put (upload files)
- ❖ open (connect)
- ❖ quit (disconnect)

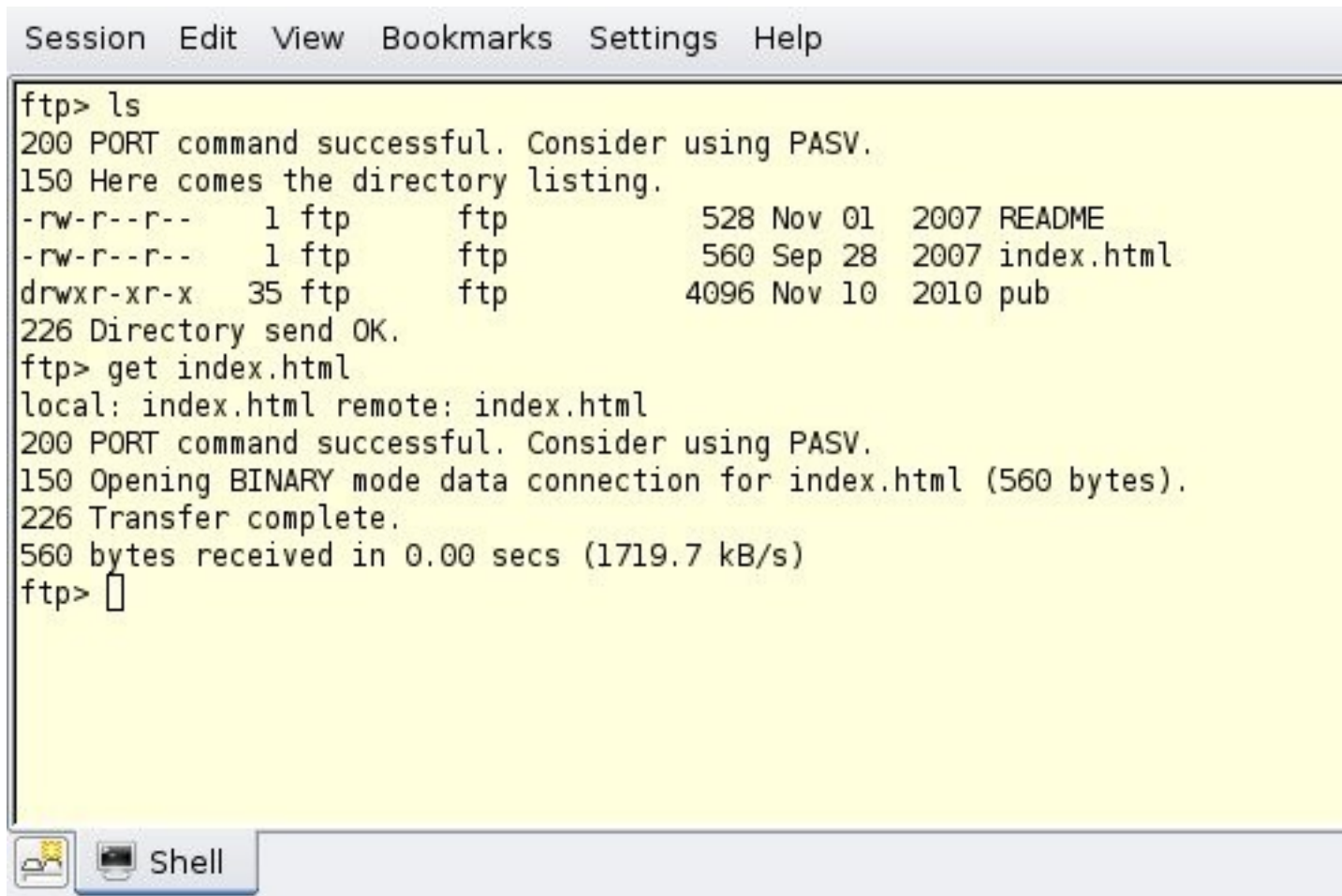
FTP clients: command line connection

```
ftp> open ftp.mozilla.org
Connected to ftp.generic-zlb.sj.mozilla.com.
220-
220- ftp.mozilla.org / archive.mozilla.org - files are in /pub/mozilla.org
220-
220- Notice: This server is the only place to obtain nightly builds and needs to
220- remain available to developers and testers. High bandwidth servers that
220- contain the public release files are available at ftp://releases.mozilla.org/
220- If you need to link to a public release, please link to the release server,
220- not here. Thanks!
220-
220- Attempts to download high traffic release files from this server will get a
220- "550 Permission denied." response.
220
Name (ftp.mozilla.org:albarcza): anonymous
331 Please specify the password.
Password:
230-
230- ftp.mozilla.org / archive.mozilla.org - files are in /pub/mozilla.org
230-
230- Notice: This server is the only place to obtain nightly builds and needs to
230- remain available to developers and testers. High bandwidth servers that
230- contain the public release files are available at ftp://releases.mozilla.org/
230- If you need to link to a public release, please link to the release server,
230- not here. Thanks!
230-
230- Attempts to download high traffic release files from this server will get a
230- "550 Permission denied." response.
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> █
```



Shell

FTP clients: command line download



The screenshot shows a window titled "Session Edit View Bookmarks Settings Help". The main area displays the output of an FTP session. The user enters the command "ls", and the server responds with a directory listing. The user then enters "get index.html", and the server responds with the file details and transfer status. The window has a taskbar at the bottom with a "Shell" button.

```
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-r--r--    1 ftp      ftp          528 Nov 01  2007 README
-rw-r--r--    1 ftp      ftp          560 Sep 28  2007 index.html
drwxr-xr-x   35 ftp      ftp        4096 Nov 10  2010 pub
226 Directory send OK.
ftp> get index.html
local: index.html remote: index.html
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for index.html (560 bytes).
226 Transfer complete.
560 bytes received in 0.00 secs (1719.7 kB/s)
ftp> 
```

Chapter 2: Application layer

2.1 Principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 Electronic Mail

- SMTP, POP3, IMAP

2.5 DNS

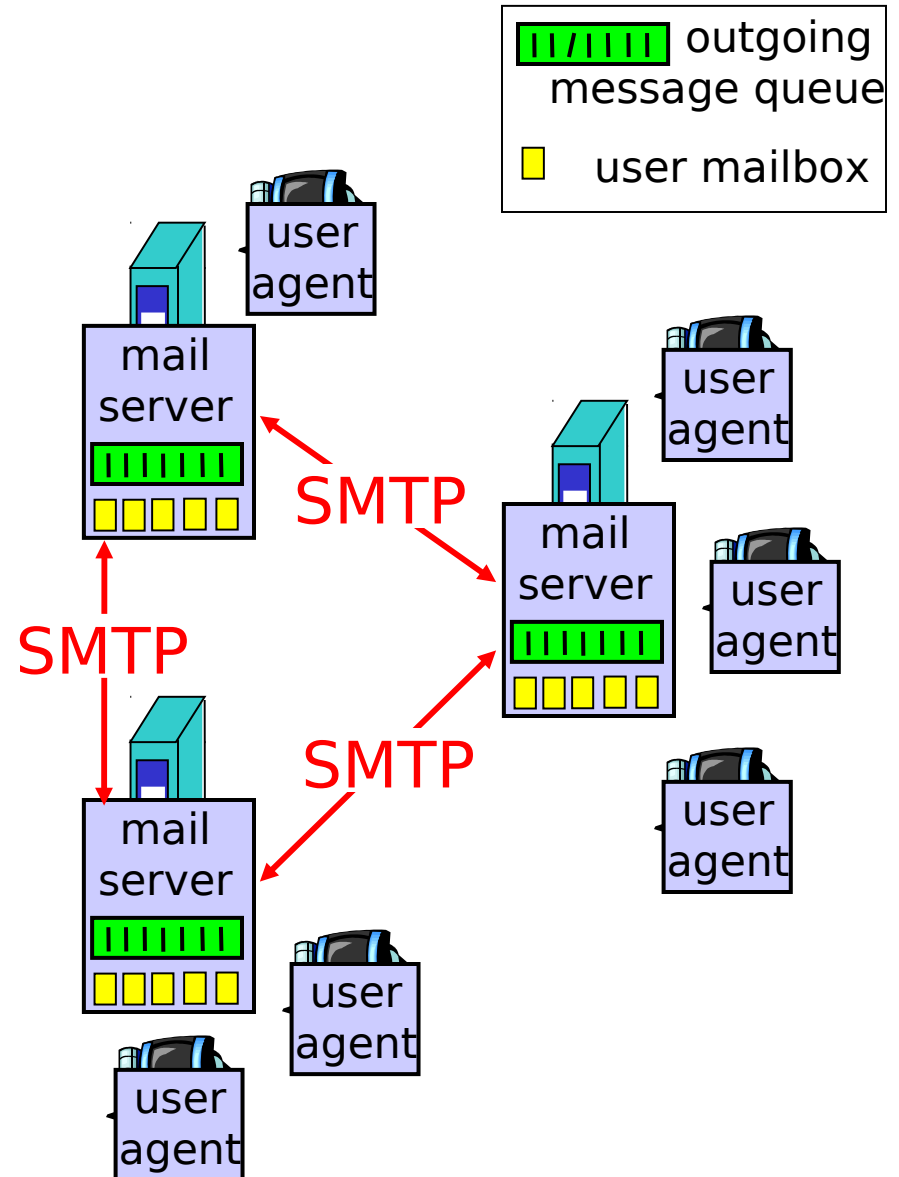
Electronic Mail

Three major components:

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

User Agent

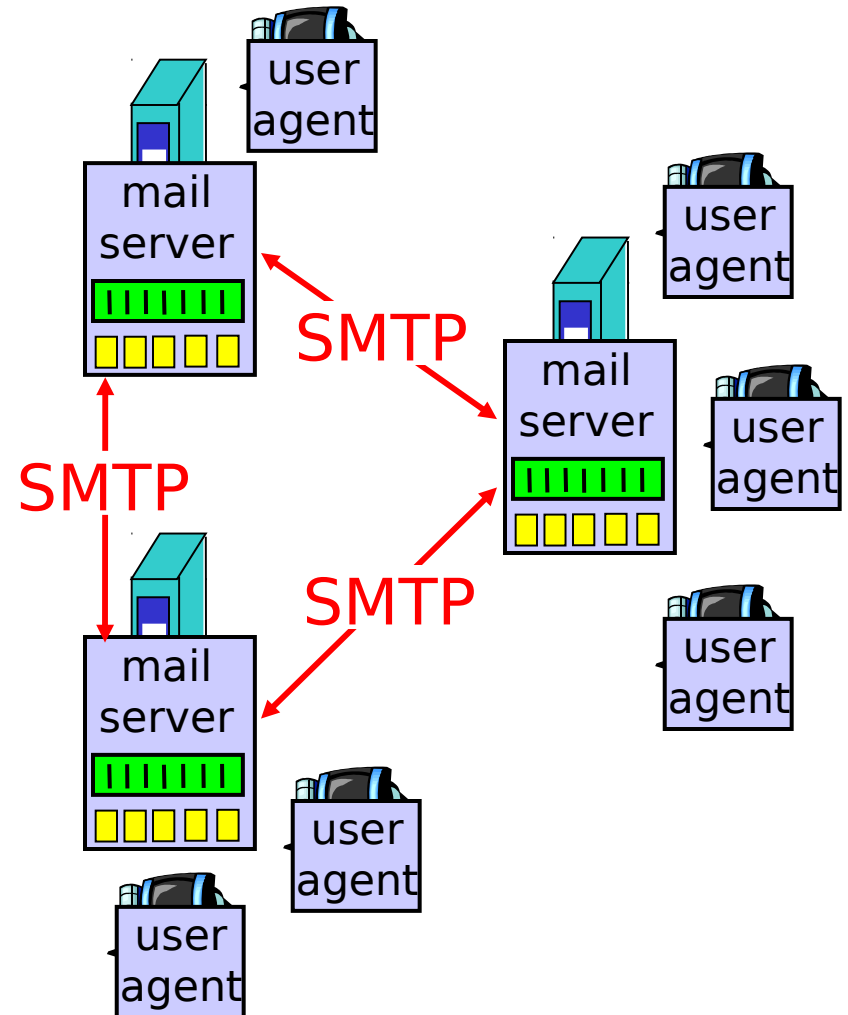
- ❖ a.k.a. “mail reader”
- ❖ composing, editing, reading mail messages
- ❖ e.g., Outlook, elm, Mozilla Thunderbird, iPhone mail client
- ❖ outgoing, incoming messages stored on server



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

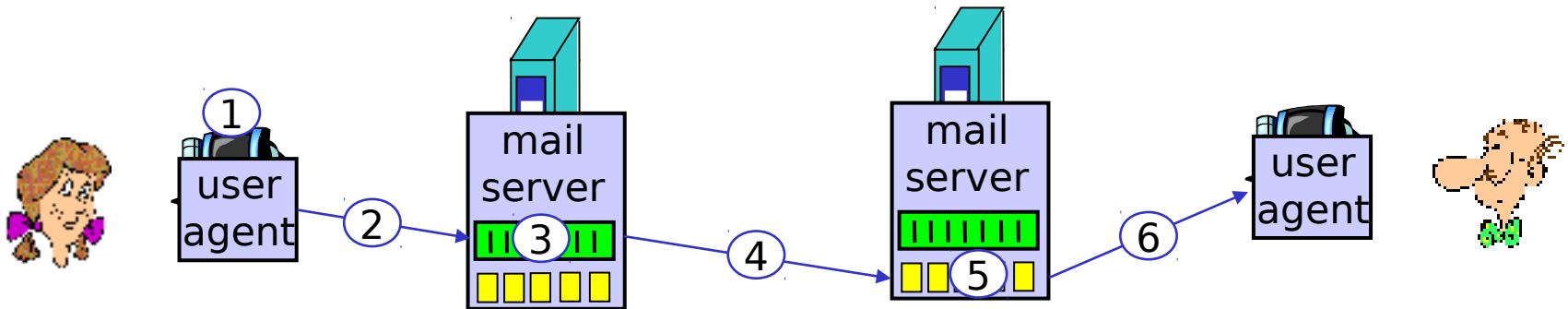


Electronic Mail: SMTP [RFC 2821]

- ❖ 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
 - **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 and "to" bob@somechool1.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:

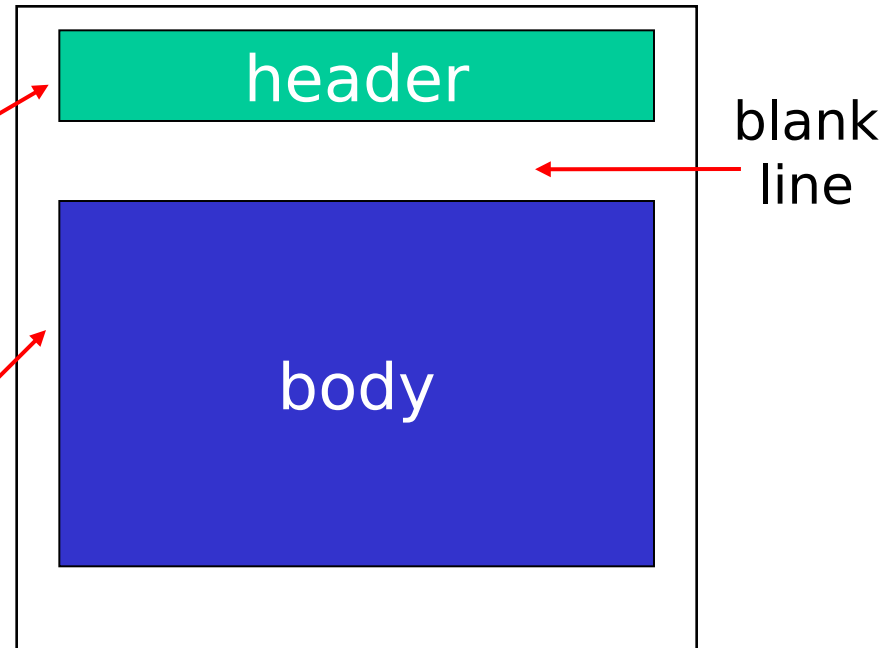
❖ header lines, e.g.,

- To:
- From:
- Subject:

different from SMTP commands!

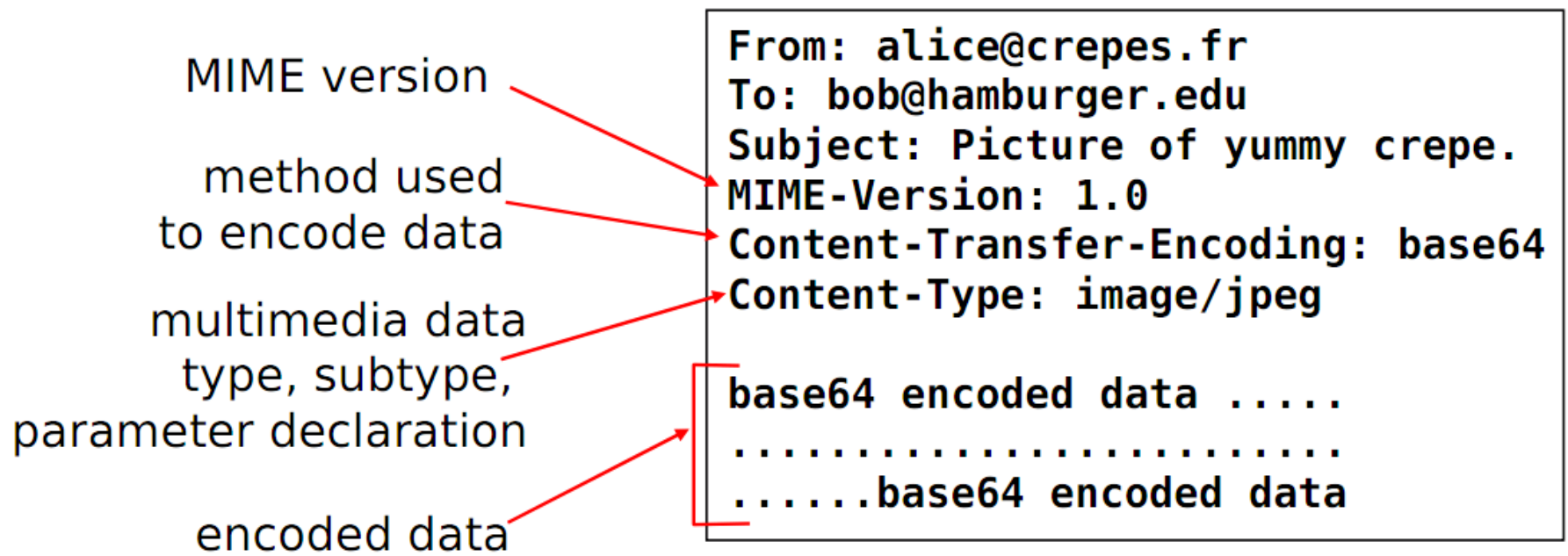
❖ body

- the “message”, ASCII characters only

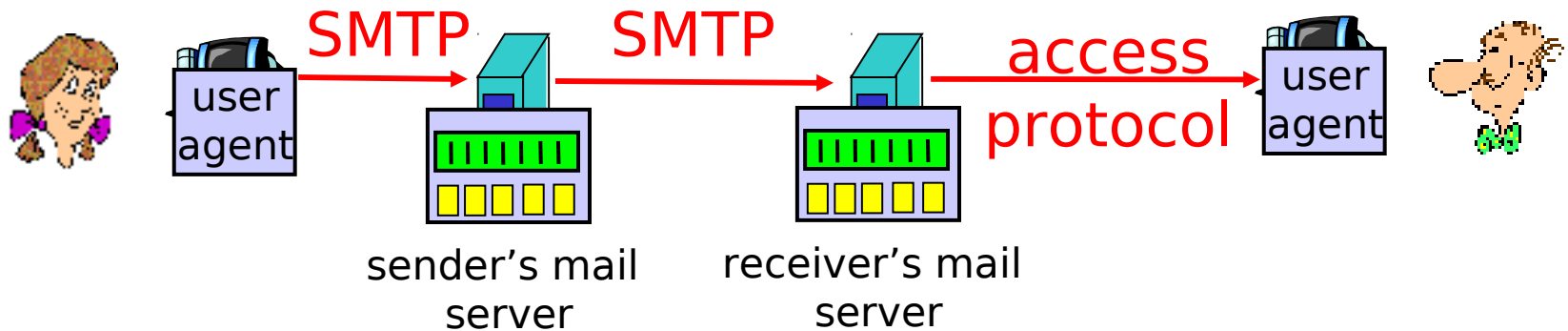


Message Format: Multimedia extensions

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



Mail access protocols



- ❖ SMTP: delivery/storage to receiver's server
- ❖ mail access protocol: retrieval from server
 - **POP**: Post Office Protocol [RFC 1939]: port **110** (POP3)
 - authorization (agent <-->server) and download
 - **IMAP**: Internet Mail Access Protocol [RFC 1730]: port **143**
 - more features (more complex)
 - manipulation of stored msgs on server
 - **HTTP**: Gmail, Hotmail, Yahoo! Mail, etc.

POP3 protocol (port 110)

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 “download and delete” mode.
- ❖ Bob cannot re-read e-mail if he changes client
- ❖ “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: Application layer

- ❖ 2.1 Principles of network applications
- ❖ 2.2 Web and HTTP
- ❖ 2.3 FTP
- ❖ 2.4 Electronic Mail
 - SMTP, POP3, IMAP
- ❖ 2.5 DNS

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: map between IP address and name, and vice versa ?

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: Domain Name System

Is there a way to find the host name, given the IP address?

<http://dnstools.com/>

Example:

130.123.128.201

resolves to **proxy.massey.ac.nz**

Port number: 53

DNS

DNS services

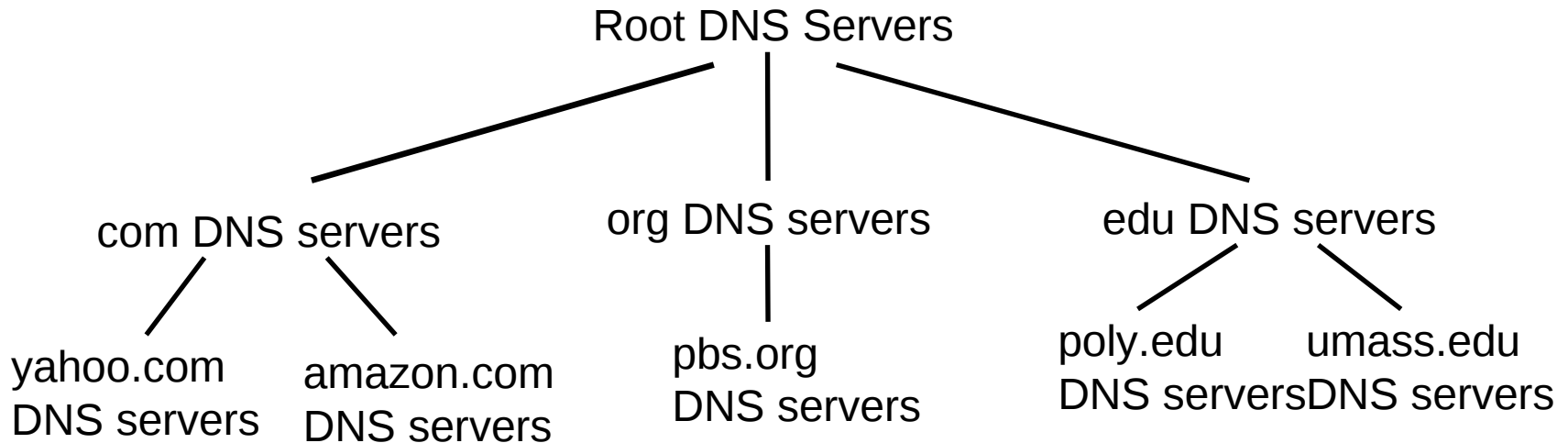
- ❖ hostname to IP address translation
- ❖ host aliasing
 - Canonical, alias names
- ❖ mail server aliasing
- ❖ load distribution
 - replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?

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

doesn't *scale*!

Distributed, Hierarchical Database

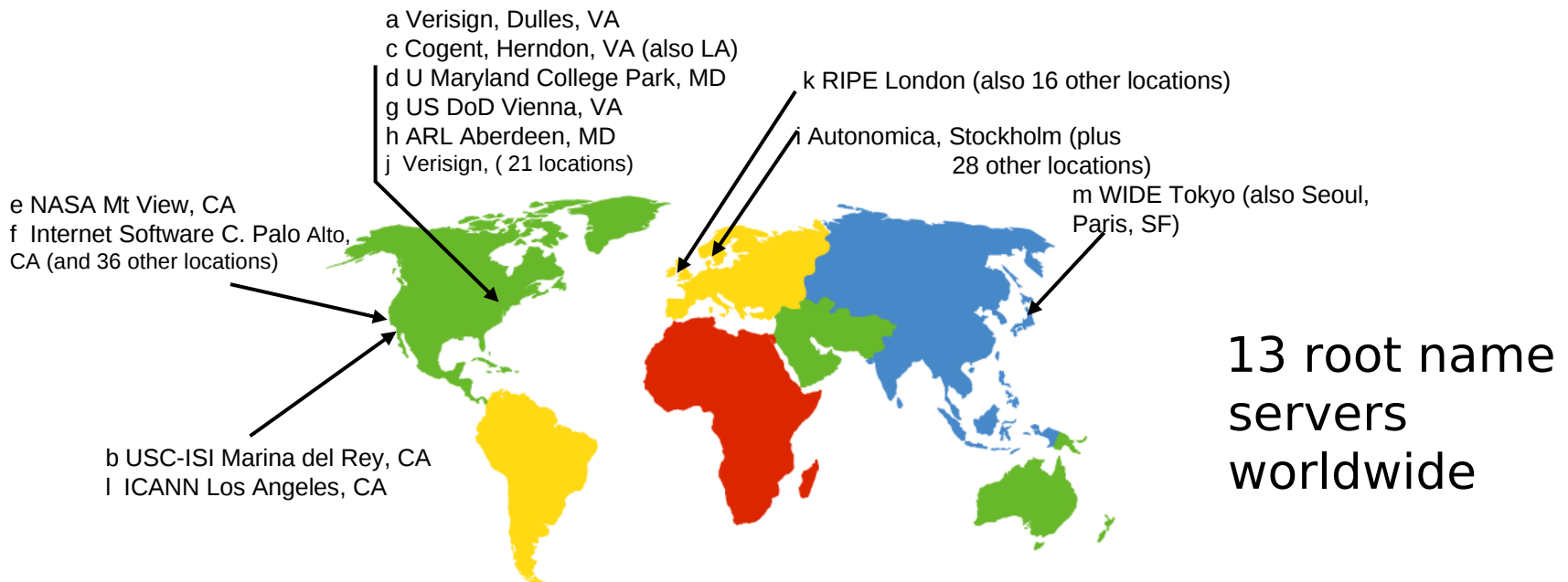


client wants IP for www.amazon.com; 1st approx:

- ❖ client queries a 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



TLD and 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 DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web, mail).
- can be maintained by organization or service provider

Local Name Server

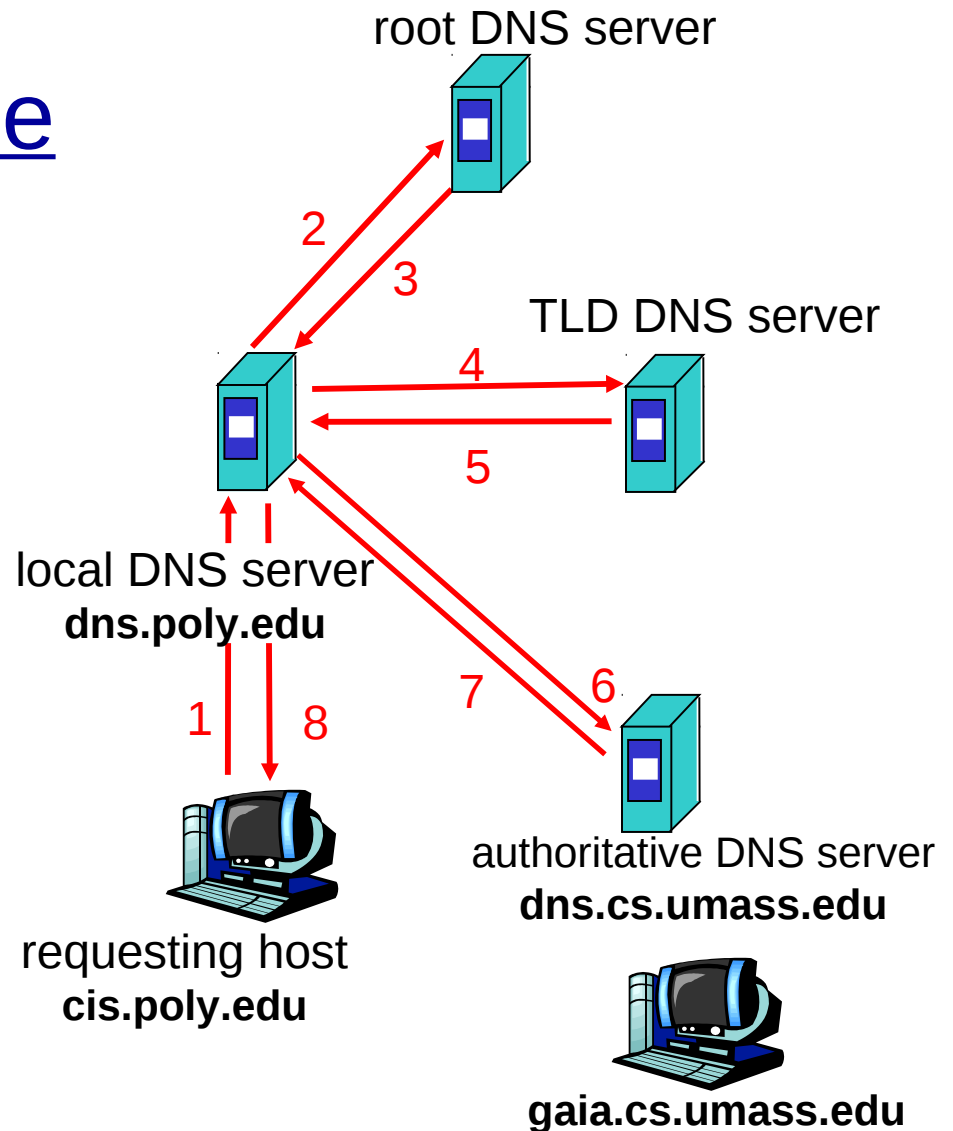
- ❖ does not strictly belong to hierarchy
- ❖ each ISP (residential ISP, company, university) has one
 - also called “default name server”
- ❖ when host makes DNS query, query is sent to its local DNS server
 - acts as proxy, forwards query into hierarchy

DNS name resolution example

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

iterated query:

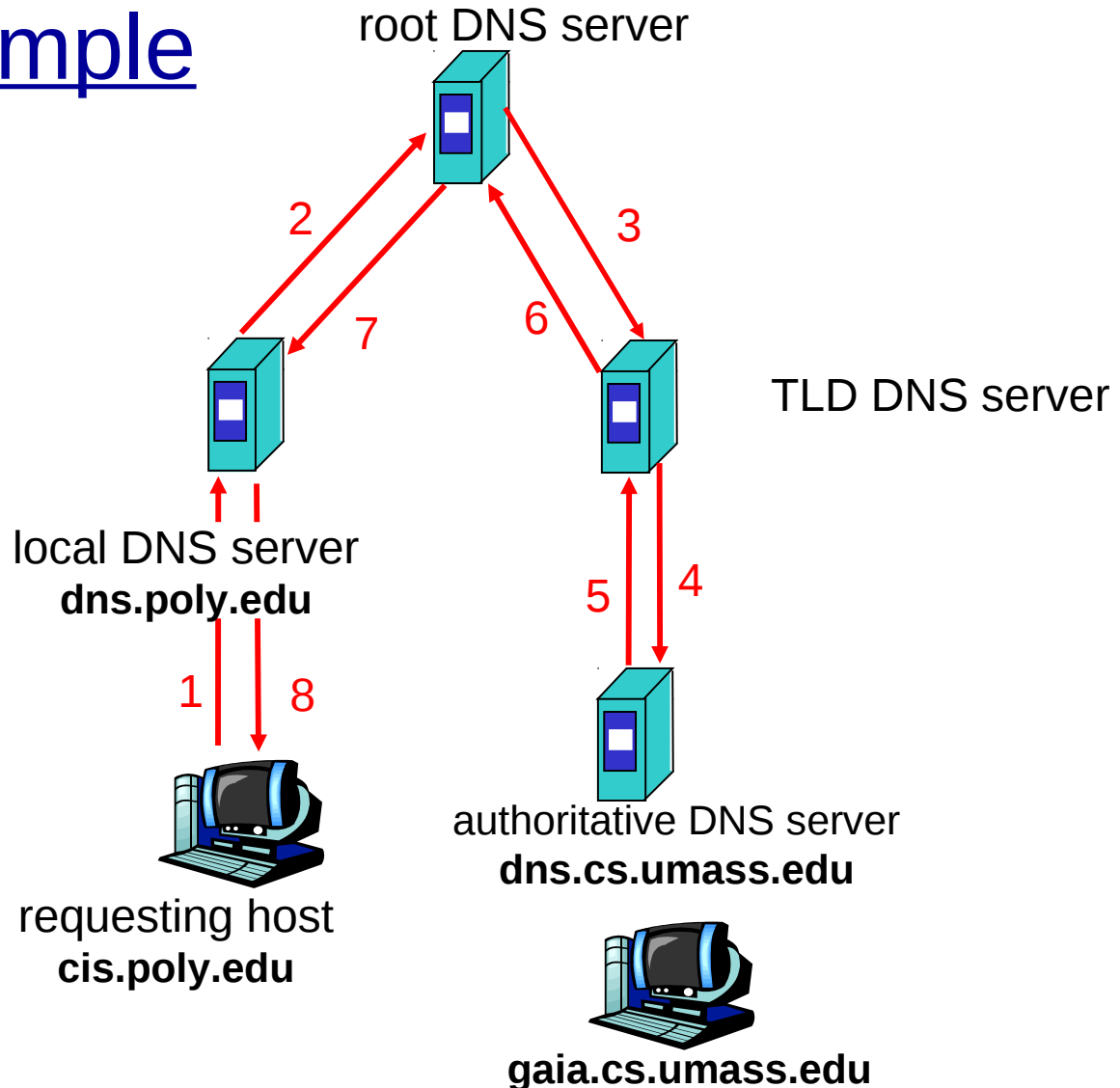
- ❖ contacted server replies with name of server to contact
- ❖ “I don’t know this name, but ask this server”



DNS name resolution example

recursive query:

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



DNS: caching and updating records

- ❖ once (any) name server learns mapping, it *caches* mapping
 - cache entries timeout (disappear) after some time
 - TLD servers typically cached in local name servers
 - Thus root name servers not often visited
- ❖ update/notify mechanisms proposed IETF standard
 - RFC 2136

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

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

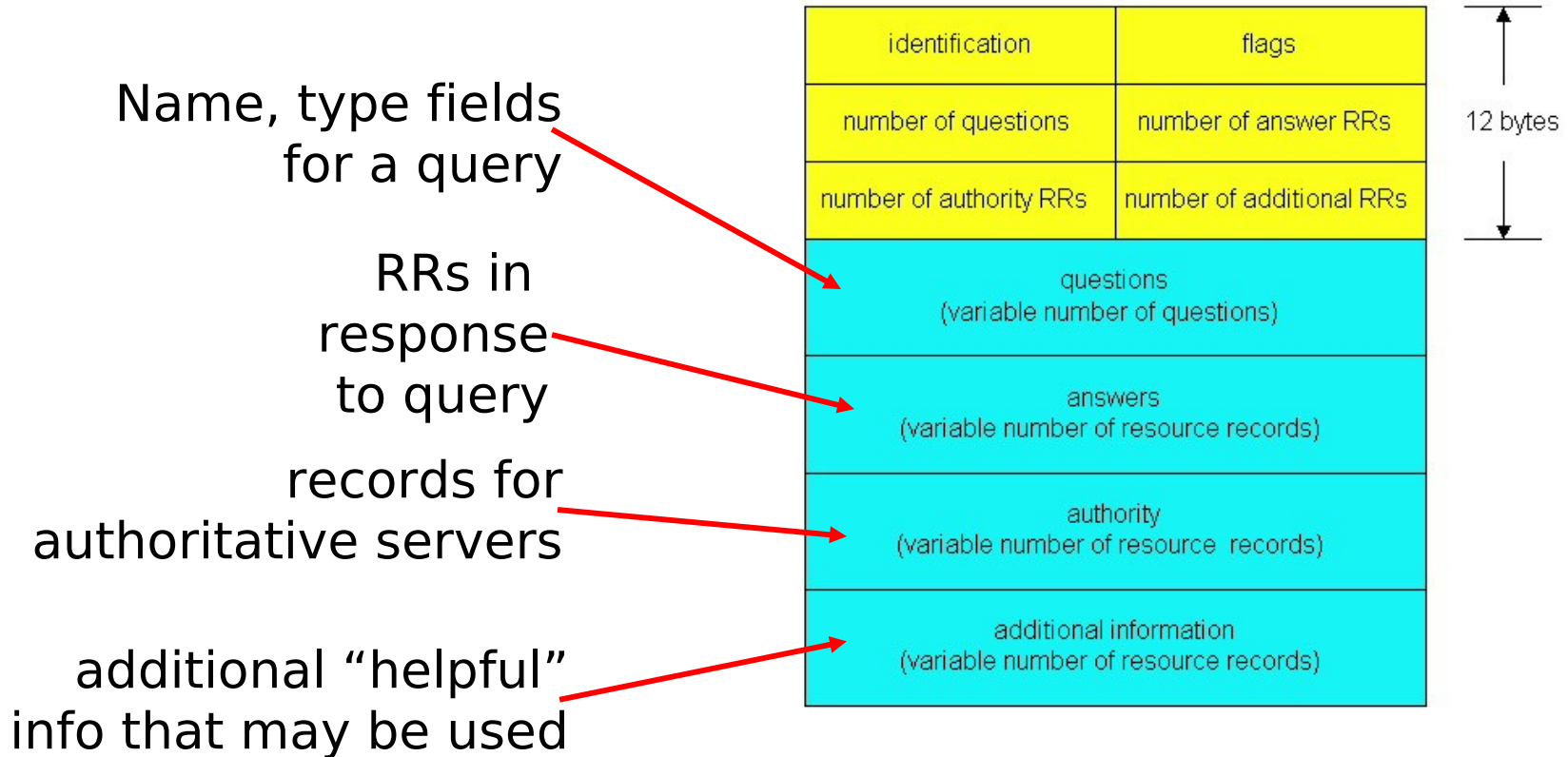
msg header

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

identification	flags
number of questions	number of answer RRs
number of authority RRs	number of additional RRs
questions (variable number of questions)	
answers (variable number of resource records)	
authority (variable number of resource records)	
additional information (variable number of resource records)	

↑
12 bytes
↓

DNS protocol, messages



Some Well-known TCP port numbers

20 FTP data

21 FTP control

22 SSH (secure 'telnet')

23 Telnet

25 SMTP

53 DNS

80 HTTP

110 POP3

143 IMAP

161 SNMP

443 HTTPS etc etc...

How many ports?

Approx. 64K

Or

From 0 to $2^{16}-1$

Or

16 bits of information

**Can I use port numbers
randomly?**

**Short answer: yes, but be
very careful...**