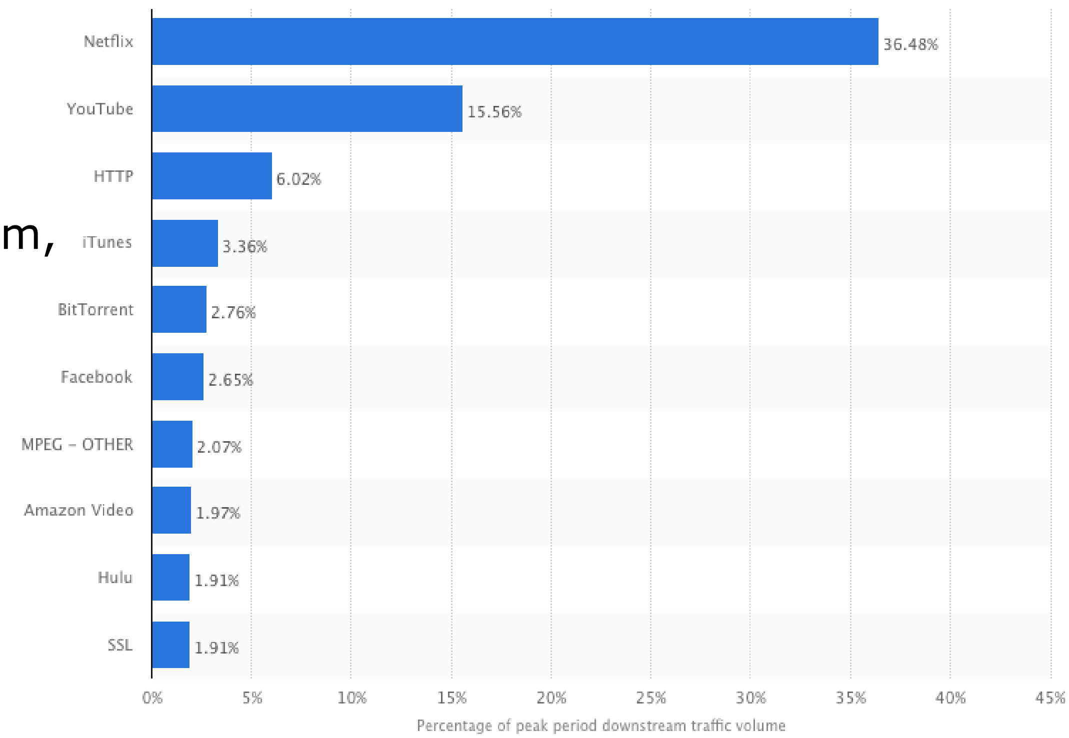
Application Level

- Client-Server and Peer-to-Peer Paradigms
- HTTP: Web Surfing
- FTP: remote connectivity
- SMTP: emailing
- DNS: symbolic addressing
- P2P: file sharing

Some Applications running in the INTERNET

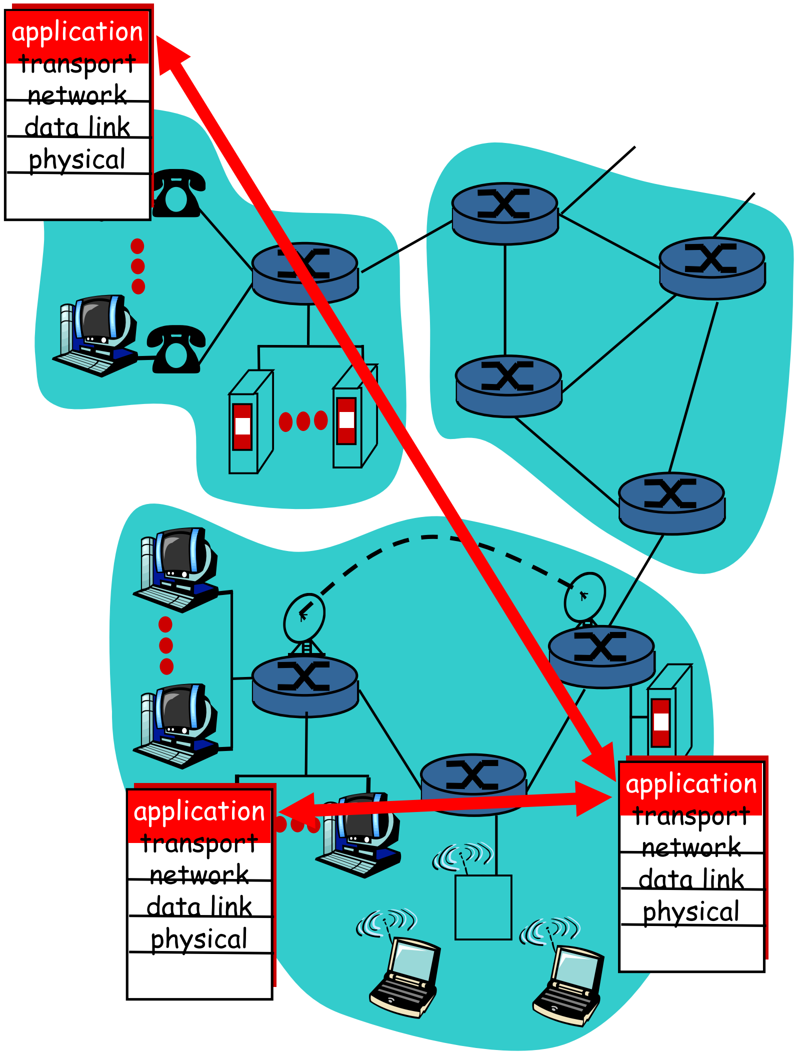
- **World Wide Web**
  - HTTP
- **Posta elettronica**:
  - SMTP, Gmail
- **Social networking**:
  - Facebook, Twitter, Instagram, Snapchat, ecc.. (social networking)
- **P2P file sharing**: BitTorrent, eMule, ecc..
- **Video streaming**:
  - NetFlix, YouTube, Hulu
- **Telefonia**:
  - Skype, Hangout, ecc..
- **Network games**
- **Video conference**
- **Massive parallel computing**
- **Instant messaging**
- **Remote login:**
  - TELNET
- …



Designing network applications

Write programs that

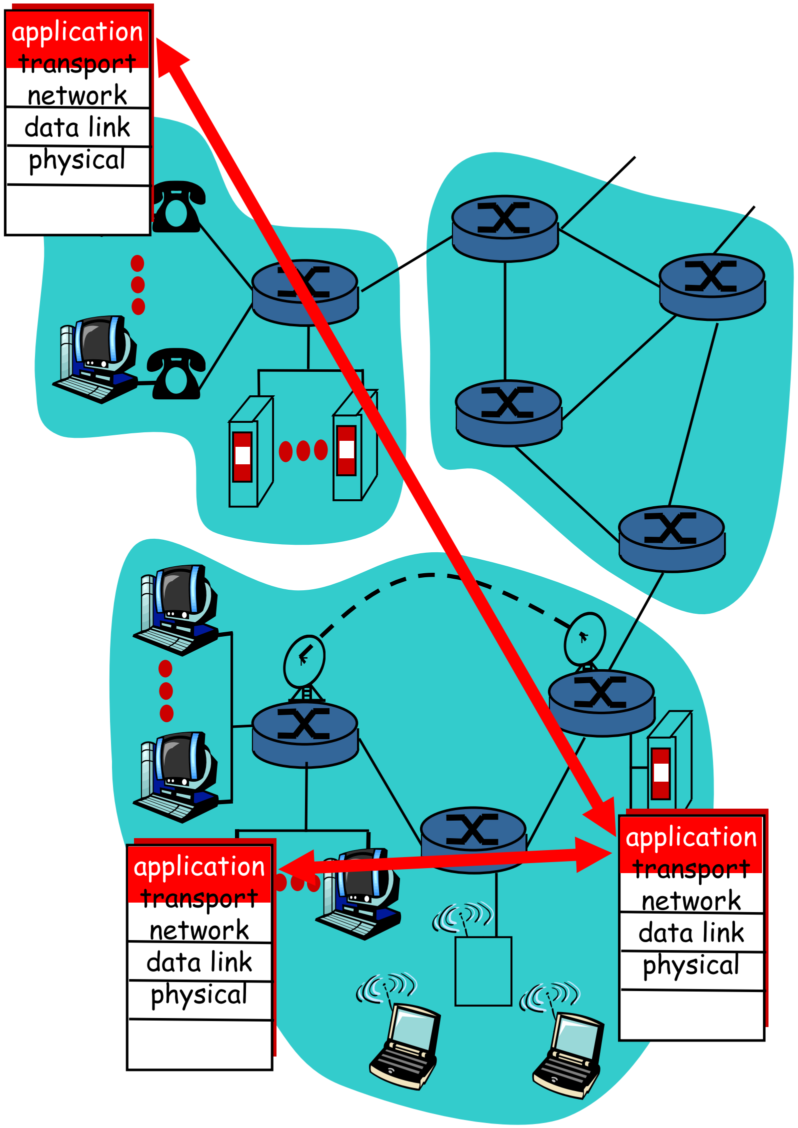
- run on different end systems and
- communicate over a network.
- e.g., Web: Web server software communicates with browser software



Designing network applications

Little software written for devices in network core

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



Communications among Processes

Process: program running within a host.

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

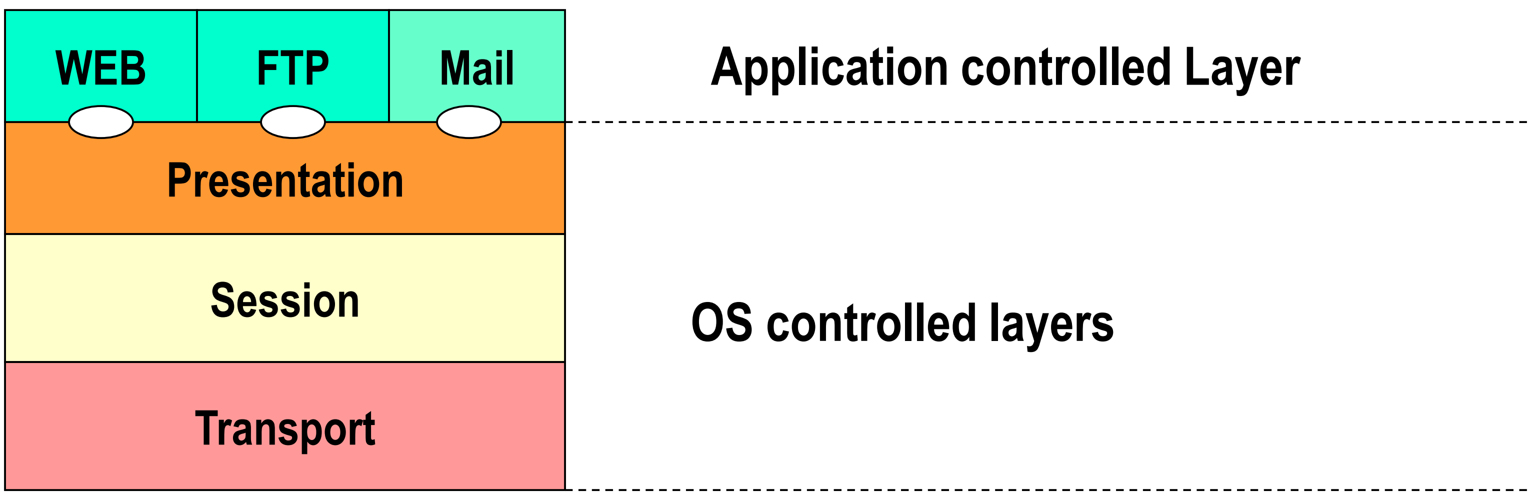
Processes and Protocols

- *Processes* running on remote hosts may exchange messages and services through the network
- The *application protocols* define the rules and the formats of the communication between remote processes

Application	Protocols
Web (web server, browser, HTML)	HTTP
E-mail (mail server, mail client, MIME)	SMTP

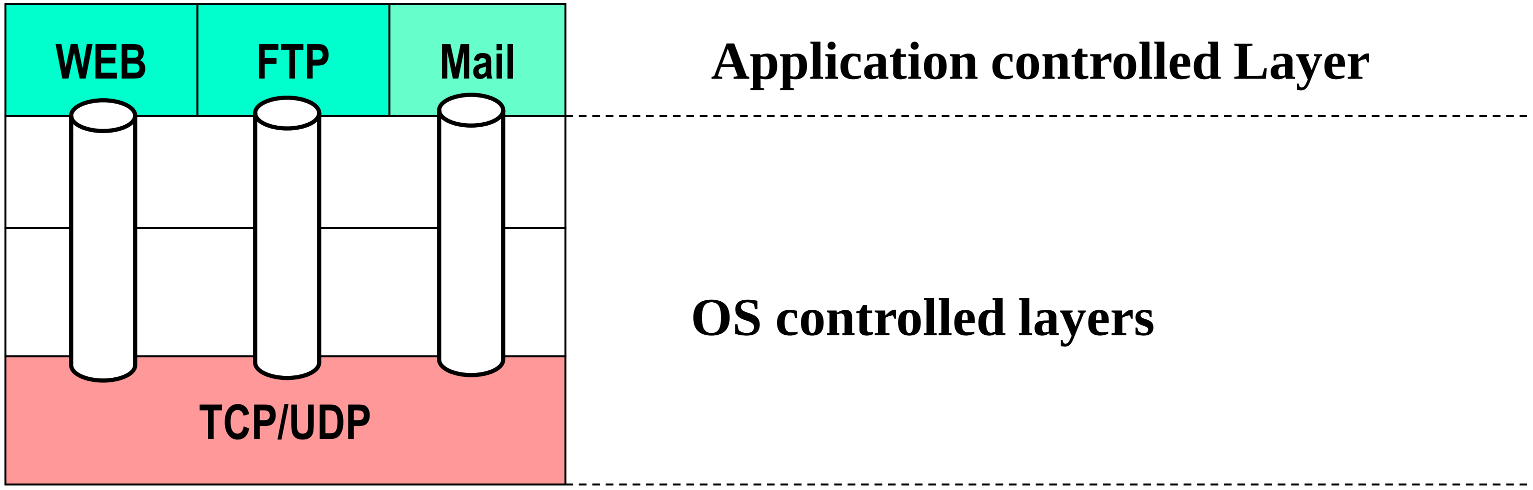
Lower layers interaction

- Application protocols use the services provided by lower layers through the SAPs (*Service Access Point*)
- Each application process is associated to a SAP
- OSI Stack:



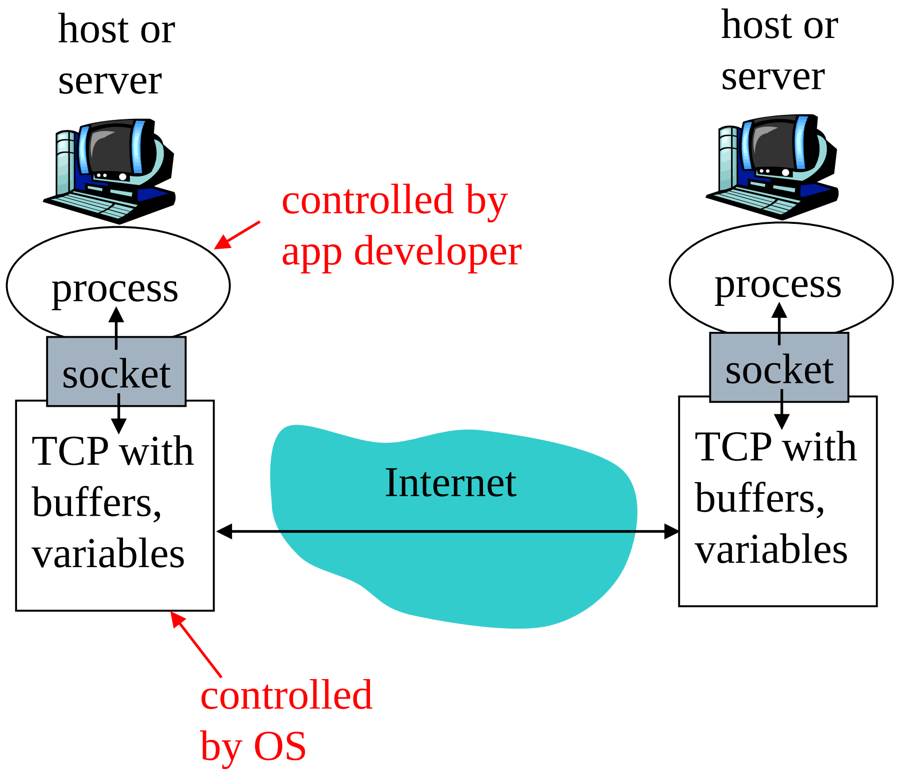
Interaction with Lower layers

- Application protocols directly communicate with the transport layer



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



- Sockets equivalent to SAPs between application and transport layers

Addressing Processes

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

Addressing Processes

- *Answer: NO, many processes can be running on the same host*
- *identifier includes both IP address and port number associated with process on host*
- *Example port numbers (HTTP server: 80, Mail server: 25)*
- *to send HTTP message to www.unibg.it:*
  - *IP address: 193.204.253.1*
  - *Port number: 80*
- *The transport layer multiplexes several flows coming from the application layer*

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., KaZaA

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"

Bandwidth
- some apps (e.g., multimedia) require minimum amount of bandwidth to be "effective"
- other apps ("elastic apps") make use of whatever bandwidth they get

Transport service requirements of common apps

Application	Data loss	Bandwidth	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 bandwidth guarantees

UDP service:

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

Q: why bother? Why is there a UDP?

Applications vs Transport Protocols

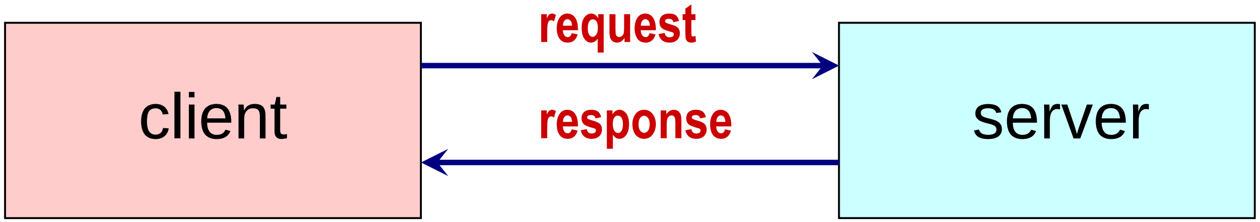
Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Vonage,Dialpad)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

Client-Server Architecture

- The main target of the communication between remote processes is the provision of services
- Two functionalities can be accomplished by a process:
  - Request for services
  - Provide services
- If a given process accomplishes just one of the above functionalities, the communication is a *client-server* one

Client-Server Architecture

- *Client* processes make requests and interpret responses
- S*erver* processes interpret requests and provide the responses
- If the same host needs to issue requests and provide responses two processes are needed

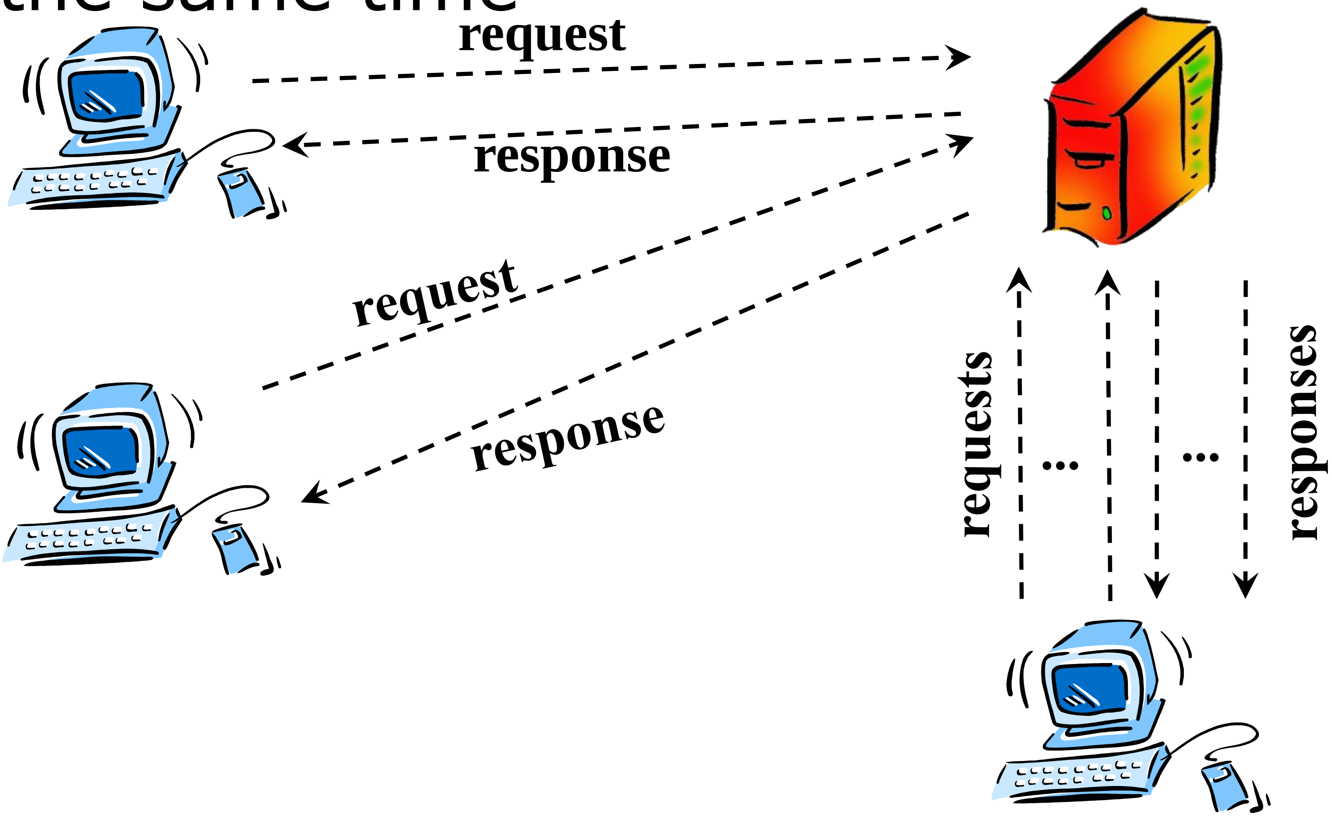


Client/Server Paradigm

- Differences between *program* and *process*
  - Program: *software*
  - Process: instance of the program being executed
- A server process is continuously executed on the host (daemon) and is activated through a *passive open*
- A client process is activated when needed only (by the user or by some other process) through an *active open*
- After the *passive open* the server is able to handle requests from clients
- *The active open* requires the indication of the IP address and the port of the server

Client/Server Paradigm

- Multiple *clients* can issues requests to *a single server*
- Clients may also issue multiple requests at the same time



Client/Server Paradigm

- A *client* may implement both serial and parallel operation modes
  - Example: multiple requests can be issued for all the files contained in a web page
- Even a *server* may implement both serial and parallel operation modes
- Usually, the applications using UDP are handled serially

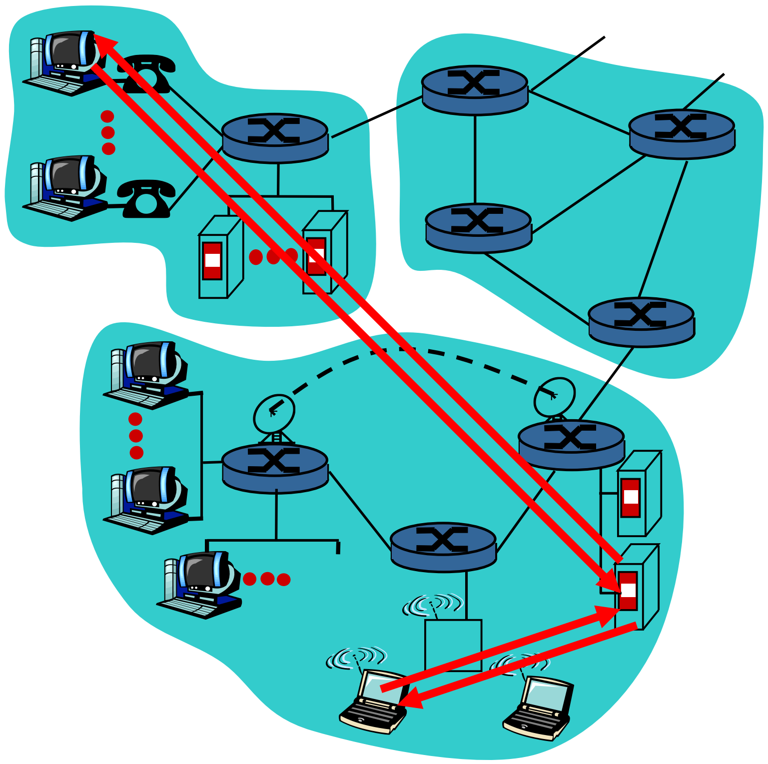
Client/Server Paradigm

- Usually, the servers using TCP implement parallel operation mode
- A TCP connection towards all the clients is opened for all the time needed to exchange requests/responses
- The procedure handling each client is handled via *multi-threading,* using fork operations

Application Protocols: Possible Architectures

- Client-server
  - Terminals act as clients OR as servers
  - Client hosts and server hosts may have different features
- Peer-to-peer (P2P)
  - All the terminals can implement the client process AND the server one
- Hybrid

Client-server architecture



server:

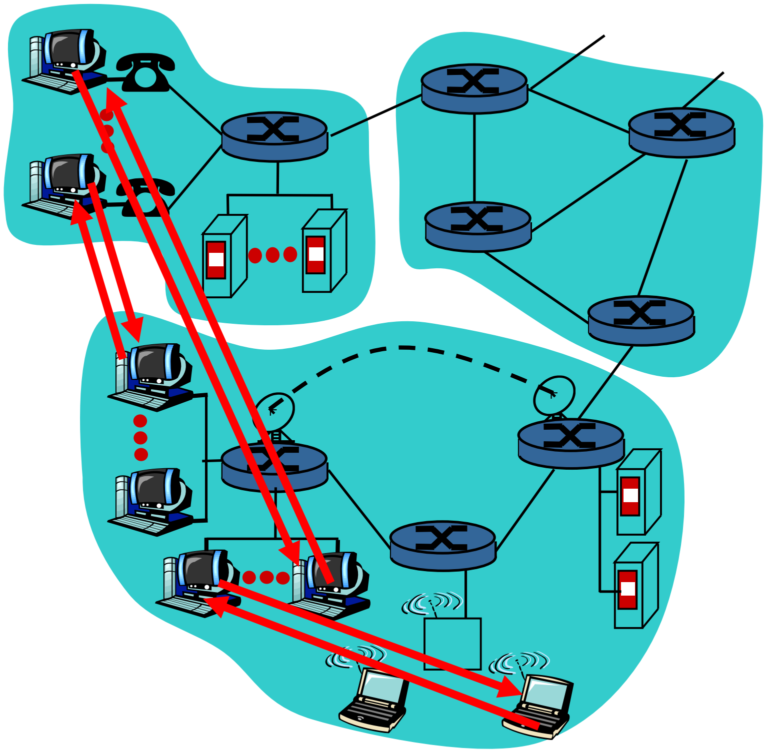
- always-on host
- permanent IP address
- server farms for scaling

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
- example: Gnutella



Highly scalable but
difficult to manage

Hybrid of client-server and P2P

Skype
- Internet telephony app
- Finding address of remote party: centralized server(s)
- Client-client connection is direct (not through server)

Instant messaging
- Chatting between two users is P2P
- Presence detection/location centralized:
  - User registers its IP address with central server when it comes online
  - User contacts central server to find IP addresses of buddies

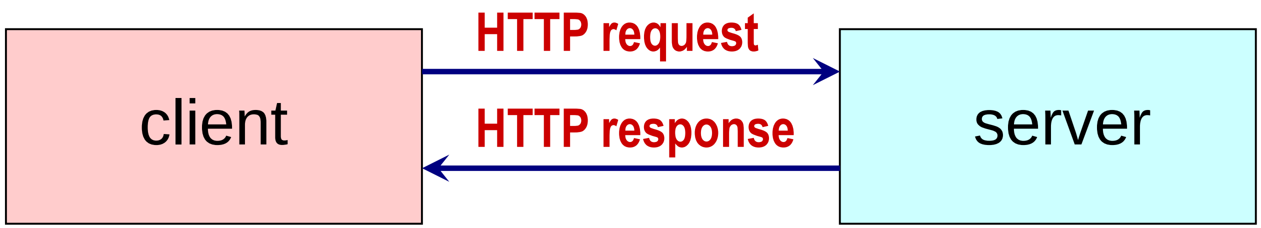
Web Browsing

Hyper Text Transfer Protocol (HTTP)

HyperText Transfer Protocol (HTTP)

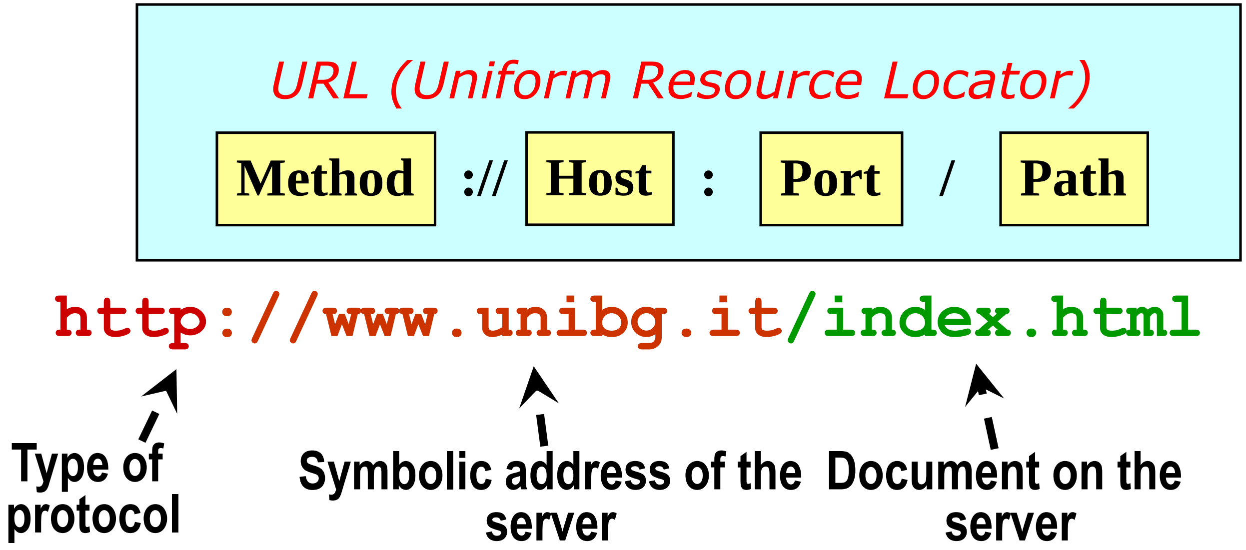
- "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.
- "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997
- "Hypertext Transfer Protocol Version 2 -- HTTP/2", RFC 7540, May 2015
- "Hypertext Transfer Protocol Version 3 -- HTTP/3", RFC 7540, Jan 2020

- *client-server* architecture
- *clients* request objects (*files*) identified through a URL
- *servers* send back the files to the clients
- Stateless operation (no memory on previous requests is mantained)



Message transfer

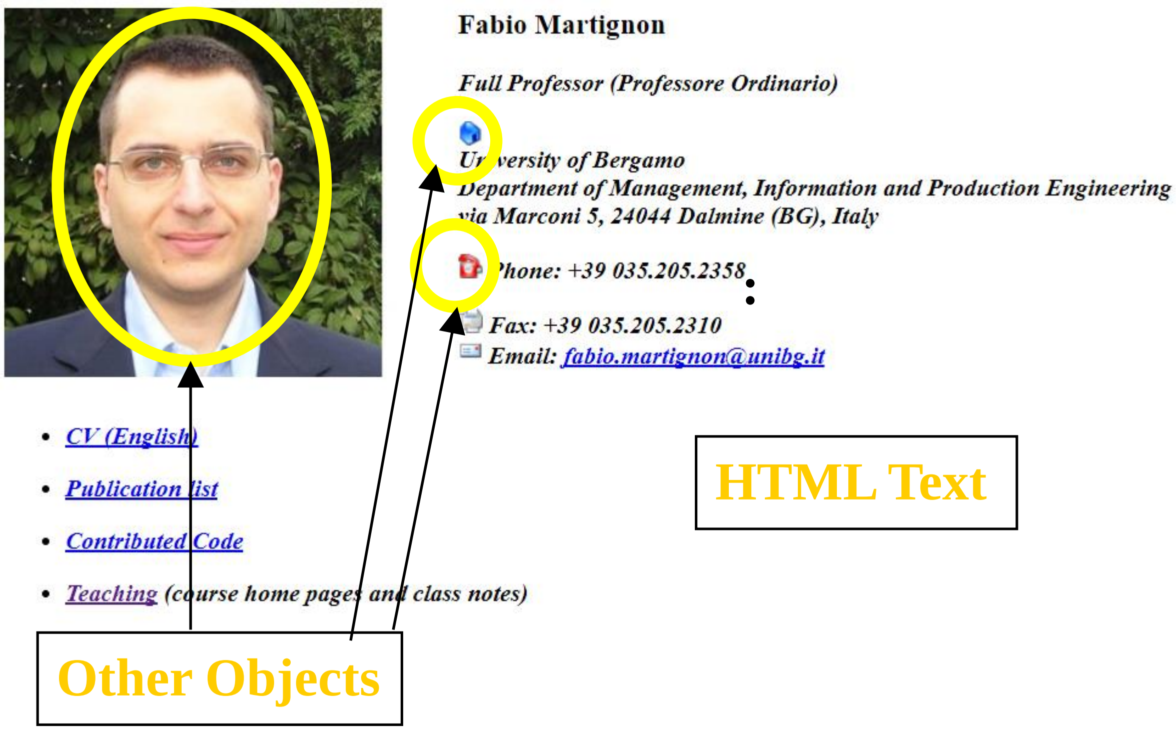
- HTTP relies upon TCP for message transfer
- Usually a web page is composed of a main document (HTML) and multiple linked **objects**
- Object can include JPEG images, JAVA applets, audio and video files, links to other web pages ...
- Requests use the



TCP assigns port number 80 to HTTP servers

Message transfer

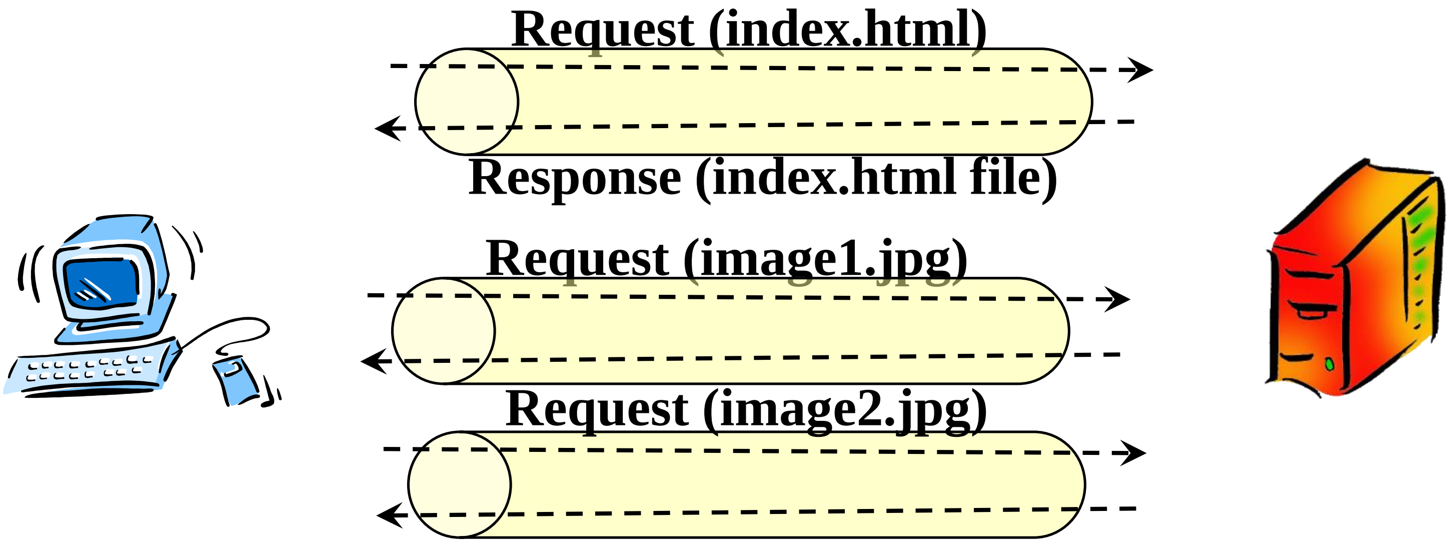
- Suppose a client requests a composite web page (1 main HTML document + 10 figures)



- **Two operation modes can be adopted**
- *Non-persistent connection* **(default mode of HTTP 1.0)**
- *Persistent connection* **(default mode of HTTP 1.1 and HTTP/2)**

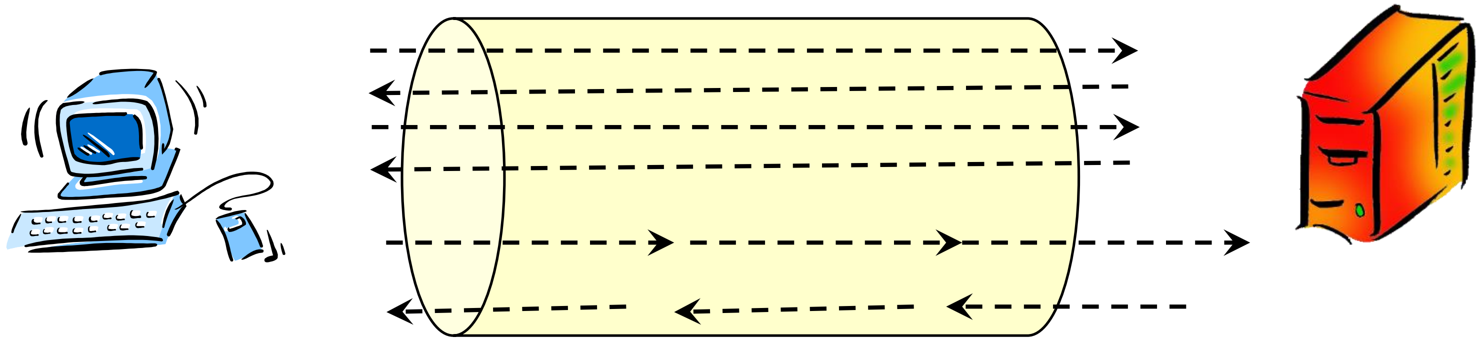
Non persistent

- One TCP connection for each request-response cycle. The server closes the TCP connection once it has sent the requested object
- The same procedure is adopted for all the docs within the required web page
- Multiple TCP connections can be opened in parallel
- The maximum number of connections can be set in the browser configuration options



Persistent connection

- The server does not close the connection after the response
- The same connection can be used to transfer other objects within the same page or even other web pages
- The server closes the connection on a timeout basis
- Two Flavors:
  - *without pipelining:* the client issues a new request only upon reception of the previous response
  - *with pipelining:* multiple requests can be issued at the same time *(default mode HTTP v1.1)*



Example – Nonpersistent connection

The user inserts in the browser the URL:

`www.polimi.it/home/index.html`

(HTML contains text and reference to 10 JPEG images)

1a. The HTTP client establishes a TCP connection with the HTTP server www.polimi.it on port 80

1b. the HTTP server in execution on www.polimi.it is waiting on port 80, it accepts the connection and notifies the client

2 the HTTP client sends an HTTP request (containing the URL) through the TCP connection. The request indicates the client wants the object /home/index.html

3 the HTTP server receives the HTTP request and sends back an HTTP respose containing the HTML file

time

Example – Nonpersistent connection

4. The HTTP server closes the TCP connection

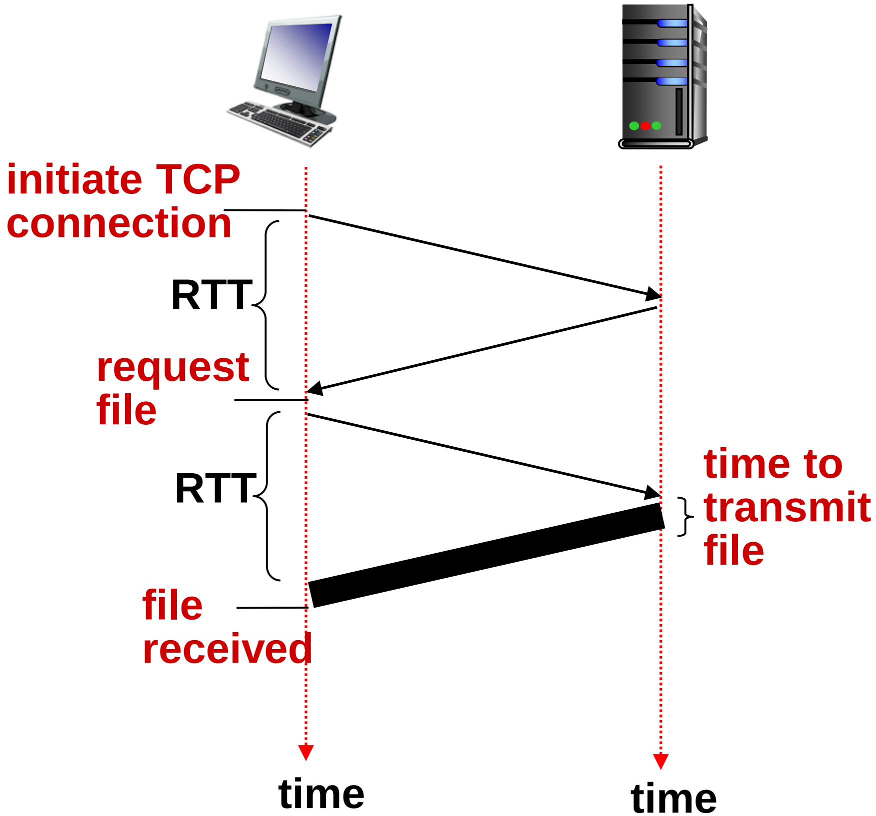
5. The HTTP client receives the response message conaining the HTML page and visualizes it. Analyzing the HTML, it discovers there are 10 other JPEG objects to download.

time

Steps from 1 to 5 are repeated for each one of the 10 JPEG images indicated on the HTML file

Estimation of the time needed for the whole transfer

- *Round trip Time (RTT)* = time to transfer a message from client to server and back
- Response time for HTTP:
  - one RTT to establish the TCP connection
  - one RTT to send the very first byte of the HTTP request and receive the first byte of the HTTP respons
  - Time to transmit the whole bytes of the ojbect (HTML file, images, etc..)

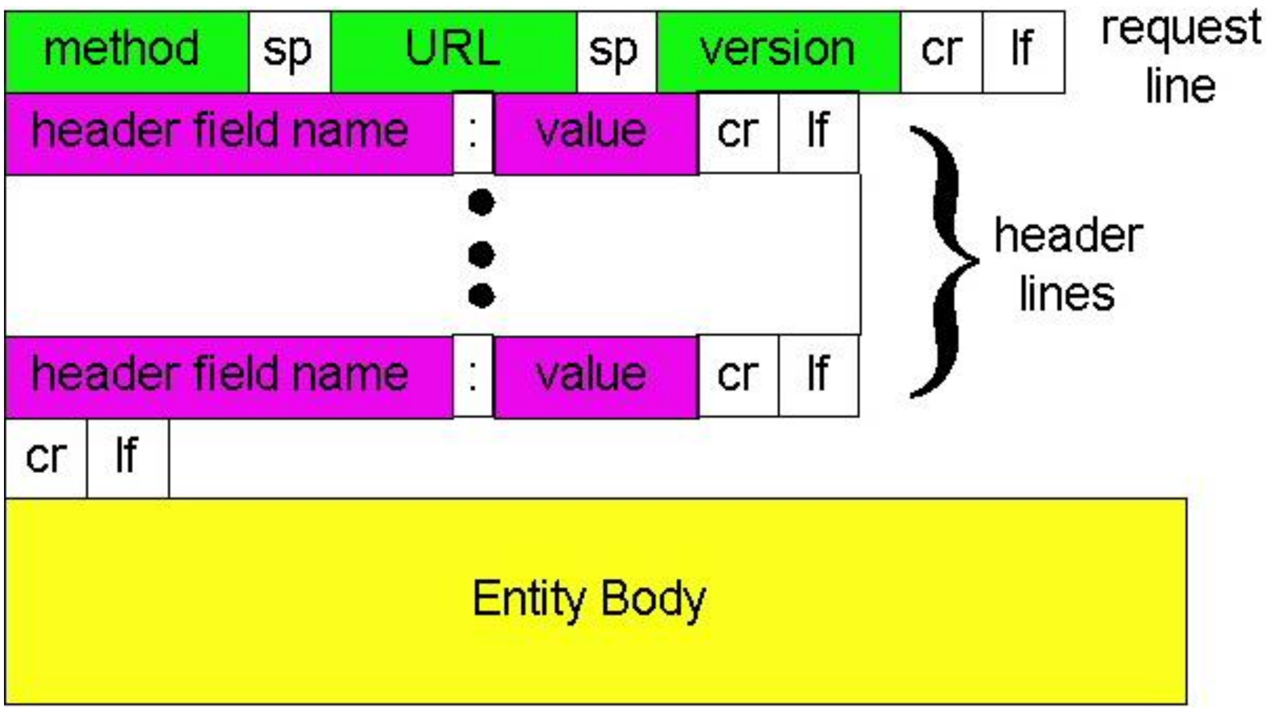


- Supposing the web page contains 11 objects (one HTML file and 10 JPEG images), the download time for the whole page is:

$$T_{nonpers} = \sum_{i=0}^{10} (2RTT + T_i)$$

$$T_{pers} = RTT + \sum_{i=0}^{10} (RTT + T_i)$$

Requests



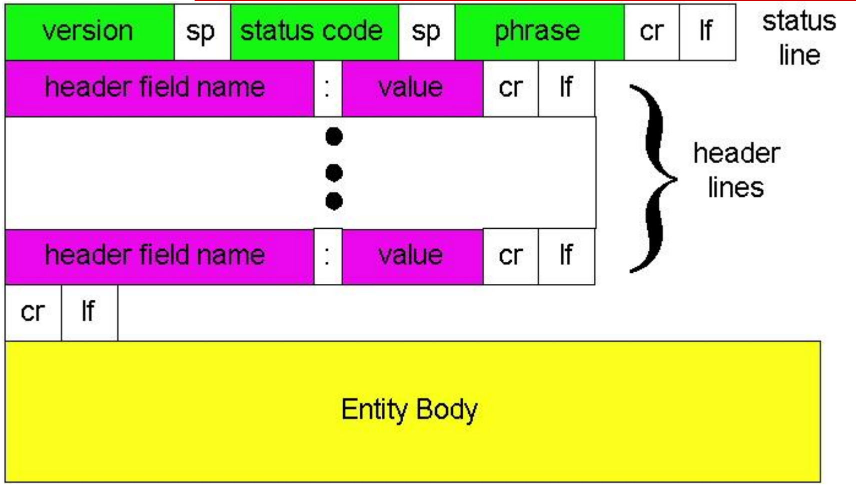

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n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

Some *Methods*

GET	To get a doc from the server. The doc is specified by the URL. The server answers with the required doc in the body of the response message
HEAD	To get info on a specified doc. The server answers with the requested information
POST	To post some input to the server regarding a given object identified by the URL
PUT	To store a doc on the server. The doc is carried by the request message. The URL specifies the position for the doc to be stored.

- **Other Methods:**
  - **PATCH, COPY, MOVE, DELETE, LINK, UNLINK, OPTIONS.**

Responses




```
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 ...
```

Messages in the **status line** are identified with a code[1]:

- **1xx**: informational
- **2xx**: success
- **3xx**: redirection (request is correct, it has been redirected to another server)
- **4xx**: client error (bad request)
- **5xx**: server errore (problem in the server)

Messages are accompanied by a text *"human readable"*

[1]*Full list in RFC 2616*
http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Messages

- 1xx Informational
- 2xx Success
- 3xx Redirection
- 4xx Client error
- 5xx Server error

100 Continue:	Go On
200 OK:	Request OK, the required info is in the field of this message
302 Moved Permanently:	The required object has been moved (perm)
304 Moved Temporalily:	The required object has been moved (temp)
400 Bad Request:	Generic error
401 Unauthorized:	Access failed due to userID or password error
404 Not Found:	File not found
500 Internal server error	Server failure
501 Not implemented	Required functionality not supported
503 Service unavailable	Unavailable service

Headers

Header name	:	Header value

- *headers* are used to exchange further service information
- A message can carry multiple headers
- Examples

`Cache-control`	Cache info
`Accept`	Supported formats
`Accept-language`	Supported languages
`Authorization`	Client permits
`If-modified-since`	send doc. only if modified
`User-agent`	user agent type

Message Exchange

HTTP is textual (ASCII) (human readable)

- Example: request

```
GET /ntw/index.html HTTP/1.1
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language:it
```

- Example: response

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html
data data data data data ...
```

Conditional get

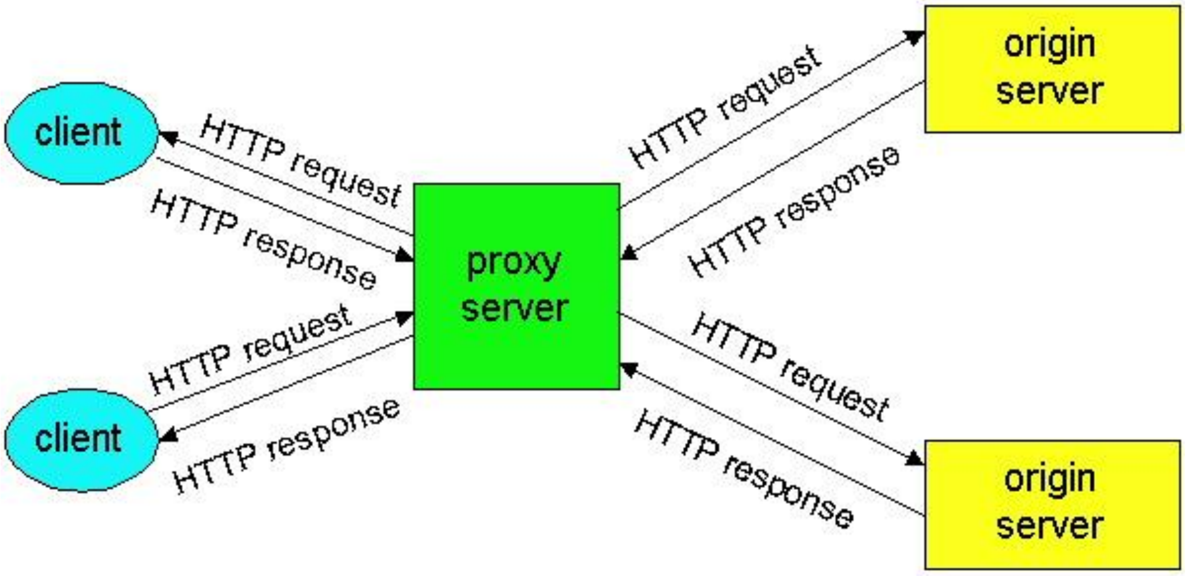
```
Client:
GET /fruit/kiwi.gif HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
If-modified-since: Mon, 22 Jun 1998 09:23:24

Server:
HTTP/1.0 304 Not Modified
Date: Wed, 19 Aug 1998 15:39:29
Server: Apache/1.3.0 (Unix)
```

(empty entity body)

- Also method HEAD can be used

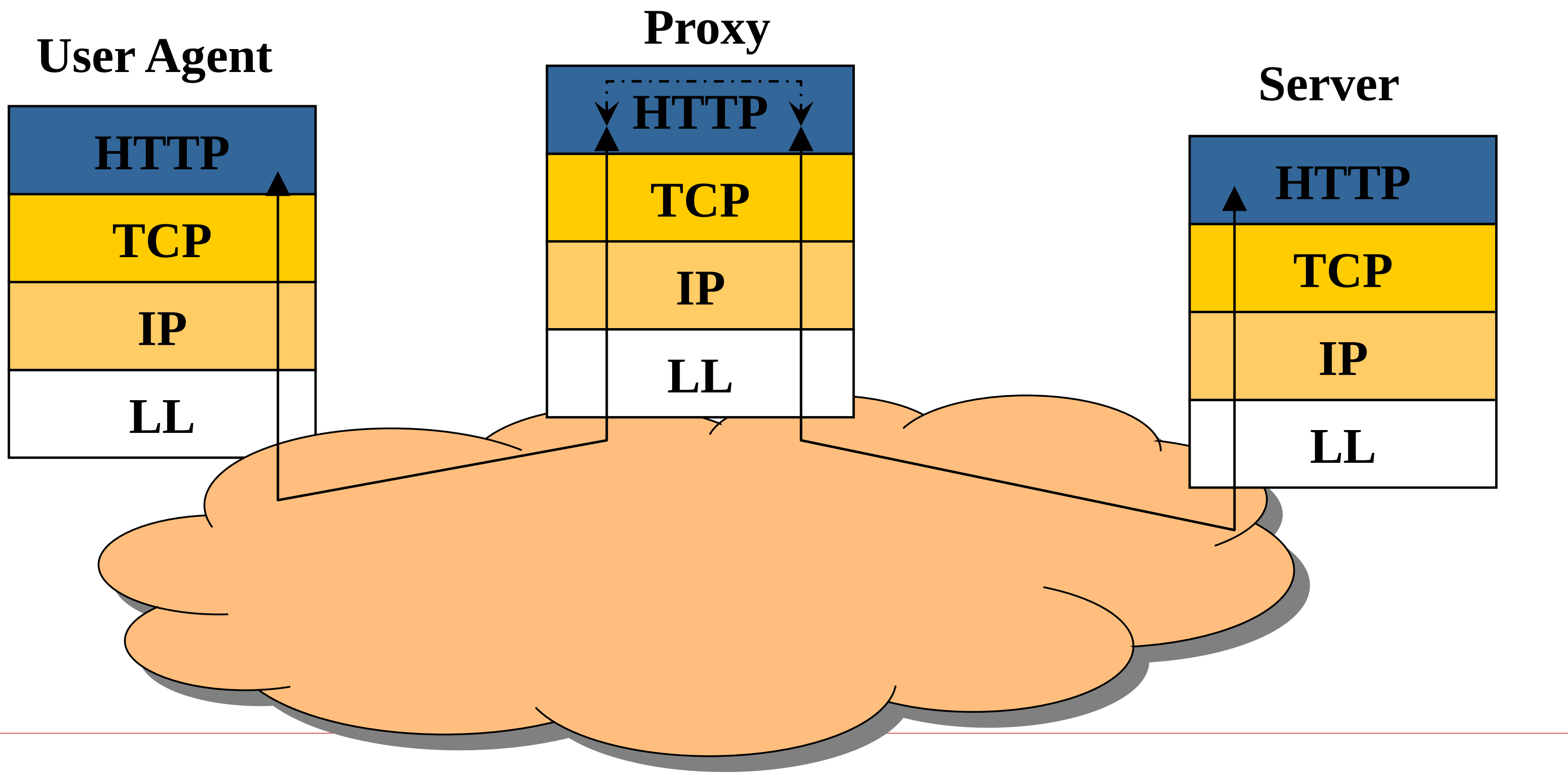
Network caching and proxy

- Main duty of a proxy is to provide a distributed cache memory
- If a doc is stored in a proxy near the client the download time can be reduced



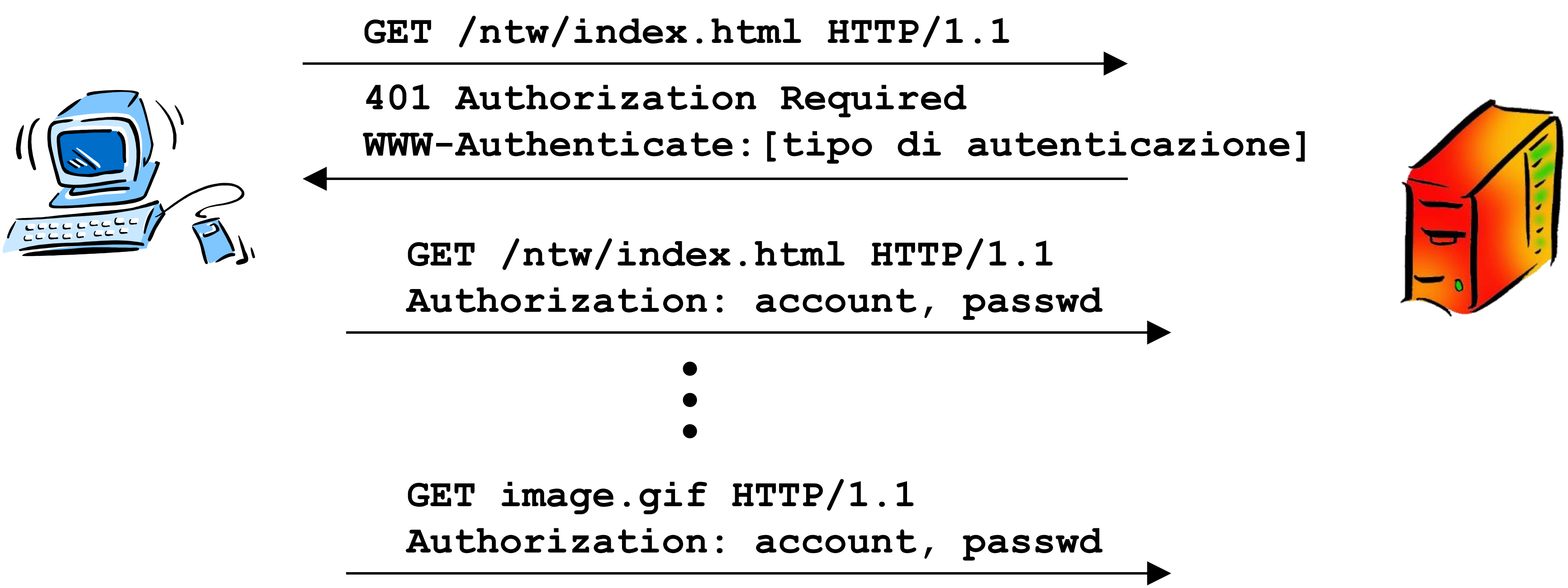
Proxy

- A *proxy* is an *application gateway*, that is, it implements up to the application layer
- It must act both as a client and as a server
- The final server speaks with the client on the proxy (hiding of users)



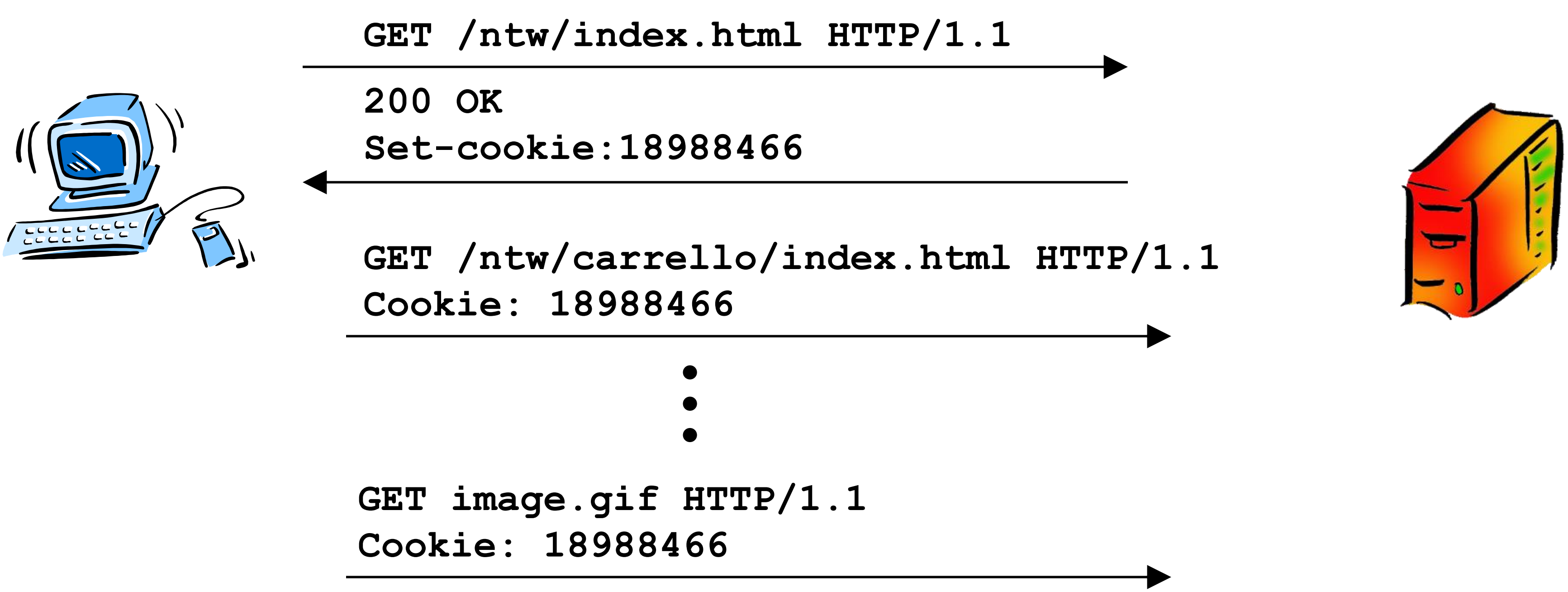
Authentication

- HTTP is *stateless*
- Consecutive requests from the same user cannot be recognized
- Very simple authentication procedure based on userID and password to be inserted in the requests

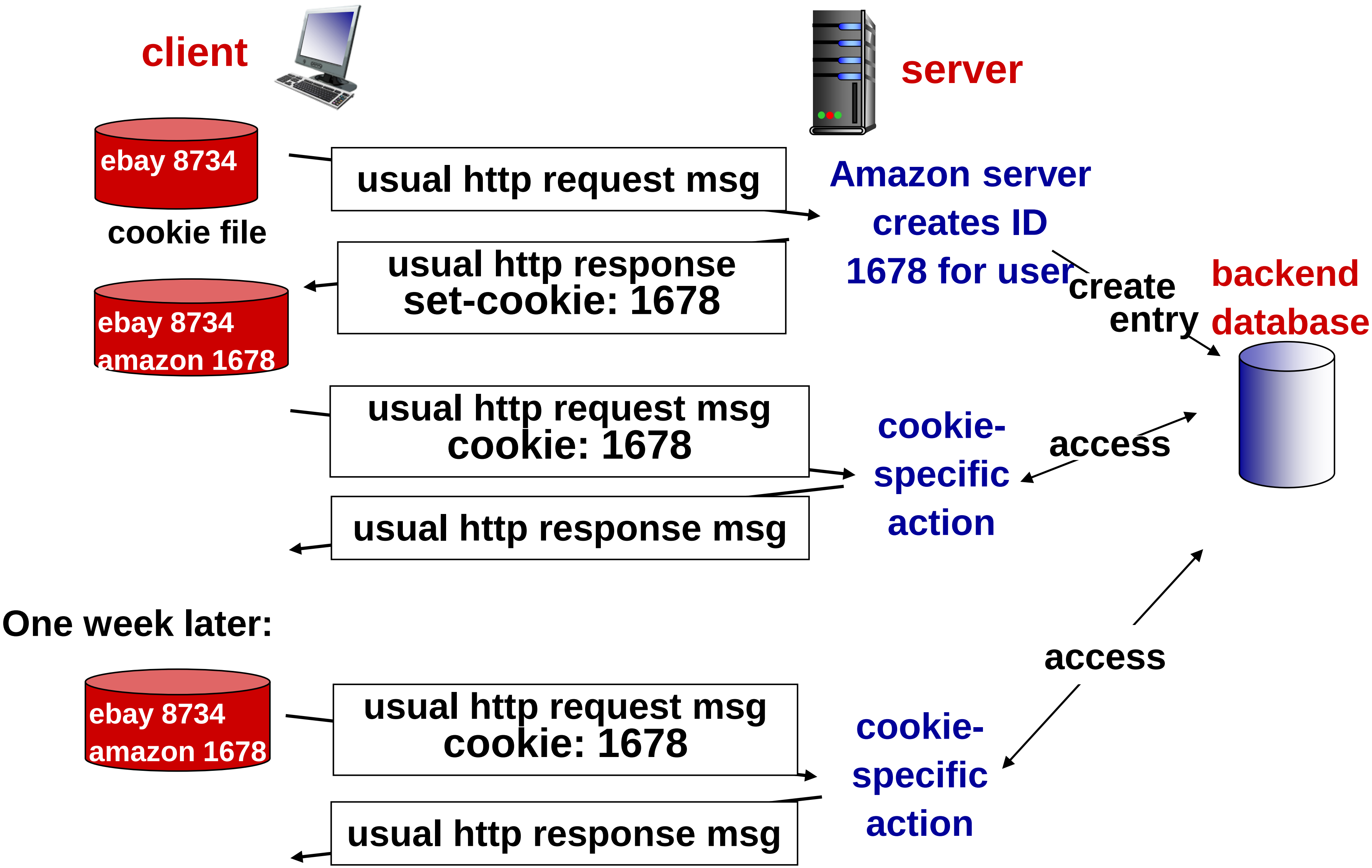


Cookies

- The server can assign to each client a cookie number which identifies the client in future transactions
- The cookie number is stored by the client and used in following requests towards the same server
- Used in e-commerce

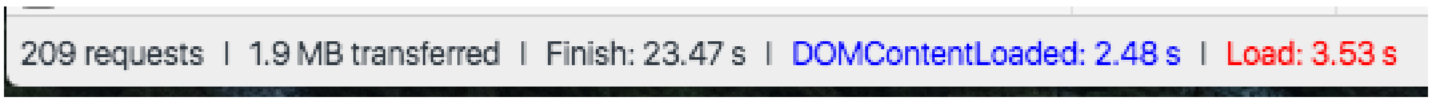


Example: utilization of cookies



HTTP/2 vs HTTP/1.1: differences

- Goal
  - Reduce latency (or *loading time*) of webpages
  - Solve some of the problems of HTTP/1.1



- The site [*www.gazzetta.it*](http://www.gazzetta.it) *includes 209 objects*
  - HTTP/1.0 uses one connection per object -> 209 TCP connections are required
  - HTTP/1.1 uses persistent TCP connections, but they are *«serial»* -> if an object is «slow», it blocks all others (*Head of Line Problem*)

HTTP/2 features

- HTTP/2 is in binary format: it transfers *frames*
- Multiplexing: one TCP connection for multiple *streams*
- Header compression
- Service of *server push*
- Flow control implemented at the application level
- It uses TLS (available also a version without it)

How much do you save? Demo https://http2.akamai.com/demo

Streams, Messages, and Frames

- HTTP/2 introduces a new binary framing mechanism that changes how the data is exchanged between the *client* and *server*. Here is the HTTP/2 terminology:

- *Stream*
  - A bidirectional flow of bytes within an established connection, which may carry one or more messages.

- *Message*
  - A complete sequence of frames that map to a logical request or response message.

- *Frame*
  - The smallest unit of communication in HTTP/2, each containing a frame header, which at a minimum identifies the stream to which the frame belongs.

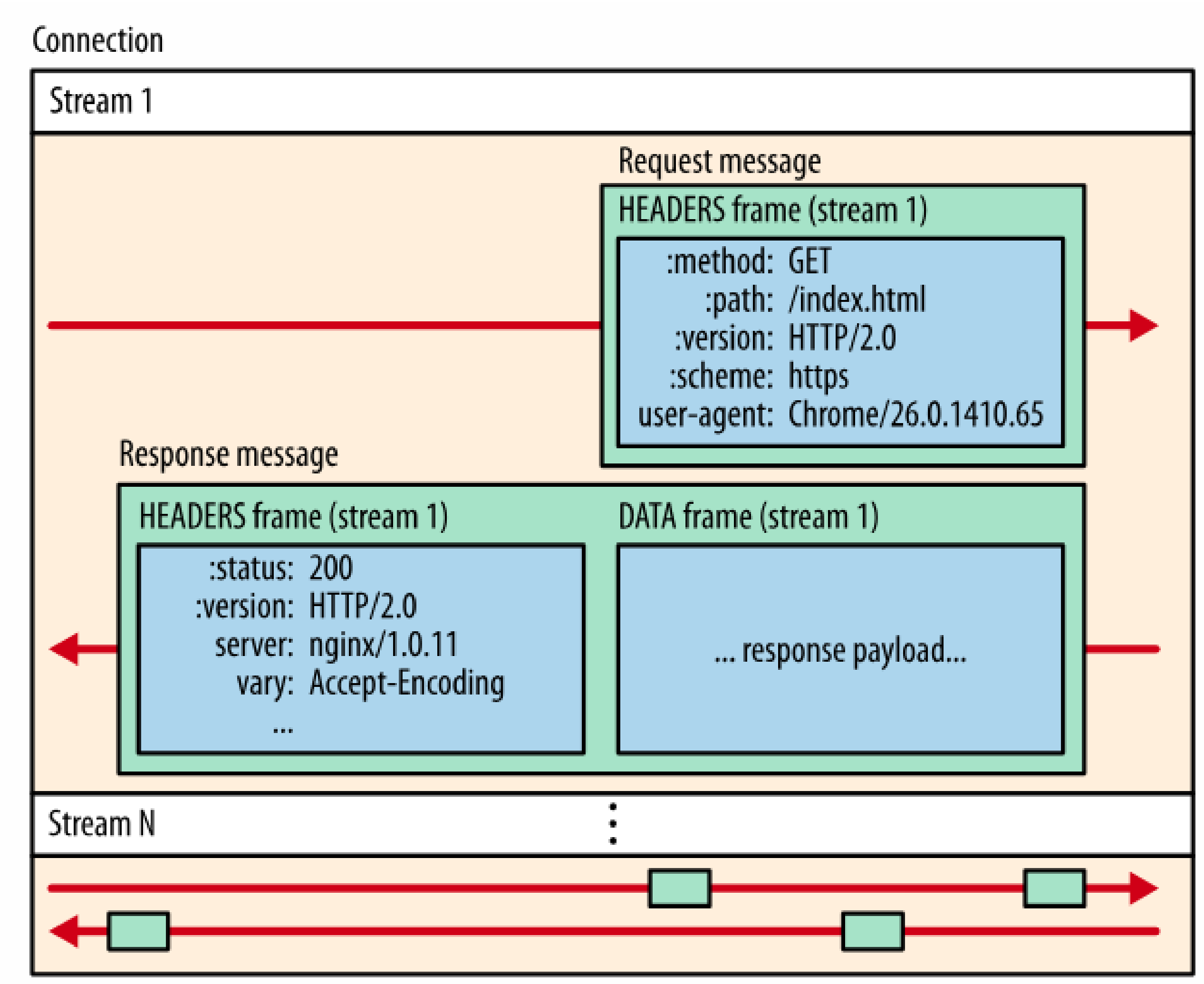
Source: *High Performance Browser Networking* (O'Reilly, Ilya Grigorik)
https://hpbn.co/

Streams, Messages, and Frames

- All communication is performed over a single TCP connection that can carry any number of bidirectional streams.
- Each stream has a unique identifier and optional priority information that is used to carry bidirectional messages.
- Each message is a logical HTTP message, such as a request, or response, which consists of one or more frames.
- The frame is the smallest unit of communication that carries a specific type of data—e.g., HTTP headers, message payload, and so on. Frames from different streams may be *interleaved* and then *reassembled* via the embedded stream identifier in the header of each frame.

Source: *High Performance Browser Networking* (O'Reilly, Ilya Grigorik)
https://hpbn.co/

Streams, Messages, and Frames



Source: ***High Performance Browser Networking*** **(O'Reilly, Ilya Grigorik)**
https://hpbn.co/

HTTP/2 Frames

```
+-----------------------------------------------+
|                 Length (24)                   |
+---------------+---------------+---------------+
|   Type (8)    |   Flags (8)   |
+-+-------------+---------------+-------------------------------+
|R|                 Stream Identifier (31)                      |
+=+=============================================================+
|                   Frame Payload (0...)                      ...
+---------------------------------------------------------------+
```

- Type:
  - DATA: carries data of a *stream*
  - HEADERS: used to open a *stream*
  - PRIORITY: specifies priorities of a *stream*
  - RST_STREAM: to terminate a *stream*
  - SETTINGS: carries configuration parameters
  - PUSH PROMISE: manages the PUSH service
  - PING,GOAWAY, WINDOW_UPDATE, CONTINUATION:

HTTP/2: header compression

▾ Request Headers   view parsed

```
GET / HTTP/1.1
Host: www.gazzetta.it
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.95 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: it-IT,it;q=0.8,en-US;q=0.6,en;q=0.4
Cookie: __gads=ID=7412cf258e0b4ea9:T=1426953830:S=ALNI_MZyituEjznRnrtBlttWZ1qNe4MWNw; is_returning=1; __ric=5295%3ASat%20May%2030%202015%2010%3A14%3A18%20GMT+0200%20%28CEST%29%7C; dnaddnt=0; mlUse
rID=5hejoxJCBTnQ; __qca=P0-538237289-1453997156314; _cb_ls=1; cpmt_xa=5295,5498; _sg_b_n=1461756783562; widgetGazzettathirdColumn_METADATA=%7B%22layoutType%22%3A%22single%22%2C%22currentGroup%22%3
A2%2C%22totalDeals%22%3A3%7D; widgetGazzetta_USER_DATA=%7B%22userId%22%3A%22ba10be1b1a6347dc92cd2e2f4003d5f0%22%7D; gvsC=New; incognitoMode=false; GED_PLAYLIST_ACTIVITY=W3sidSI6IjZialUiLCJ0c2wiOjE
0ODMyMDg3NTIsIm52IjoxLCJ1cHQiOjE0ODMyMDc3NTgsImx0IjoxNDgzMjA4NzUyfV0.; channel=Natural Search|https://www.google.it/|Google - Italy|No Keyword; userid=7D345911-D6EA-A6E2-153A-70B57DFBC241; s_sq=%5
B%5BB%5D%5D; utag_main=v_id:014f8a939b03000c34f86cdefc3f06079001707100838$_sn:579$_ss:1$_st:1486131591176$_pn:1%3Bexp-session$ses_id:1486129791176%3Bexp-session; OAS_SC1=1486129791328; __vrf=14861
29791350gUfvWL6rYGybFmH3ZqBoEsyvp58VqcPd; TSstop=NA|1486129791436; testcookie=true; s_fid=7C687C1454C159A8-0E27B74F8356438A; s_fbsr=1; s_nr=1486129793025-Repeat; gpv_sect=homepage; SC_LNK_GZ=%5B%5
BB%5D%5D; gpv_page=GAZ%2F; s_cc=true; _ga=GA1.2.761966839.1426317491; _sg_b_p=%2F; _ceg.s=oksx35; _ceg.u=oksx35; _gat=1; ch_CBT=tracked; s_ppvl=GAZ%2F%2C5%2C5%2C302%2C1121%2C302%2C2560%2C1440%2C1%
2CP; _cb=zieHgC_uIsSBIZQXv; _chartbeat2=.1426317498062.1486129797169.0000000000000101.C3a4tLCKWncwC1n_lKPB8pYDbZ1m6; _cb_svref=null; dtPC=-; gazzettaNotifications=%7B%22creationDate%22%3A142631749
8004%2C%22skipModal%22%3Afalse%2C%22articles%22%3A%5B%5D%7D; dtLatC=-9876; dtCookie=|_default|0|Gazzetta|1; _sg_b_v=633%3B3739777%3B1486129793; s_ppv=GAZ%2F%2C5%2C5%2C336%2C1399%2C332%2C2560%2C144
0%2C1%2CP; _chartbeat4=t=BAxcExDH0c7TBt6mwQCVC2xWB9i4a_&E=0&EE=0&x=0&c=0.81&y=10184&w=332
```

- The *header* of HTTP requests can have non-negligeable size since it can contain: several *cookies,* several *header line* for authentication, specific of the transaction, etc.
- The *header* of <u>consecutive</u> HTTP (towards the same server) contains redundant information

HTTP/2: HPACK header conmpression

[1]RFC7541, https://tools.ietf.org/html/rfc7541

- *Huffman coding*: gives binary strings to most common symbols
  - ex: a-101, c-0, e-111, p-110, t-100, the word «*accept*» (6 byte if codified in ASCII) *is sent as 101 0 0 111 110 100 (only 2 bytes)*

- *Indexing*: it consists in giving an index to the most common *header lines* and then send only the such index in the messages

- *Differential coding*: the *header* of consecutive requests carries only the difference with respect to the *header* of previous requests

HTTP/2: HPACK header compression

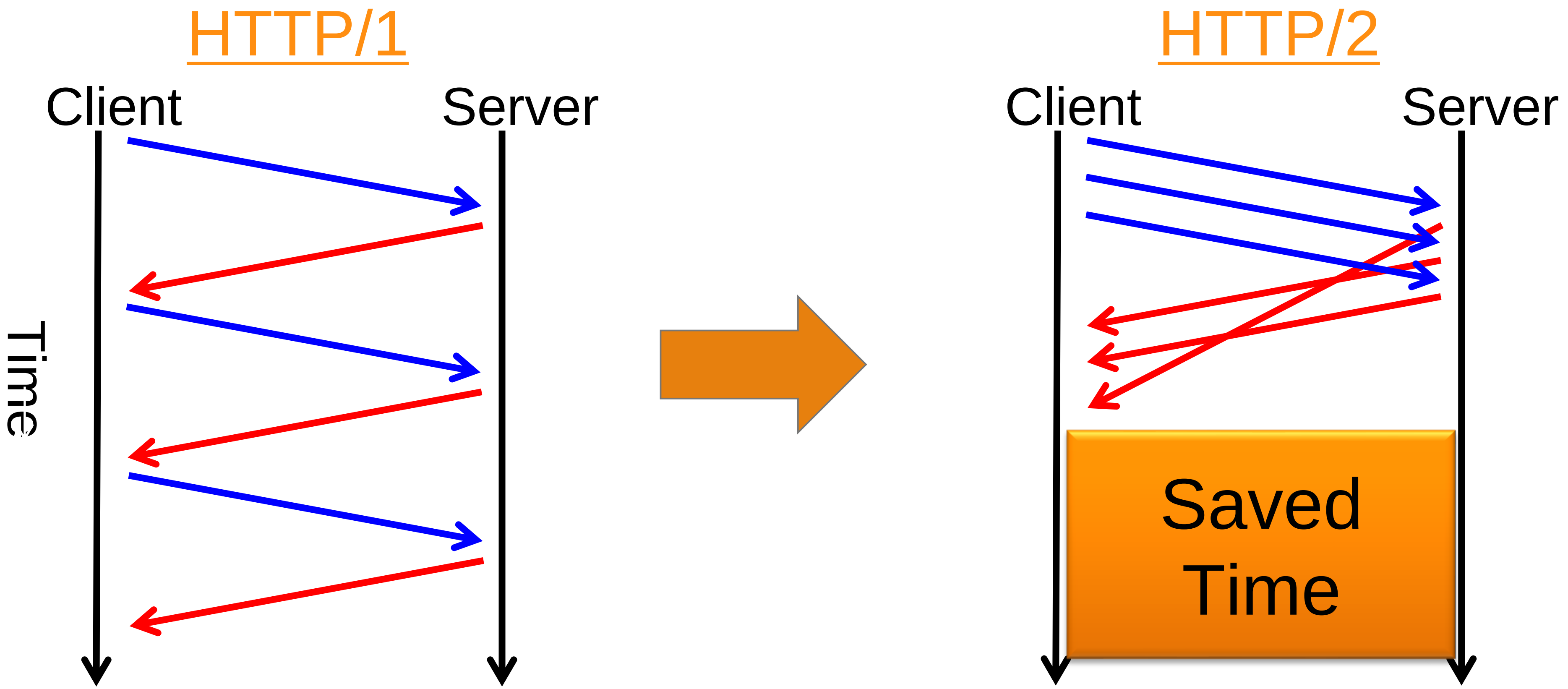
HTTP/2

	Generated Message	Sent Message
First Request	:method: GET :scheme: https :host: www.keycdn.com :path: /index.html referer: https://www.keycdn.com/ accept-encoding: gzip	2 7 38 www.keycdn.com ➔ ID 62 5 51 https://www.keycdn.com/ 16
Second Request	:method: GET :scheme: https :host: www.keycdn.com :path: /logo.svg referer: https://www.keycdn.com/index.html accept-encoding: gzip	2 7 62 4 logo.svg 51 https://www.keycdn.com/index.html 16

HTTP/1.1

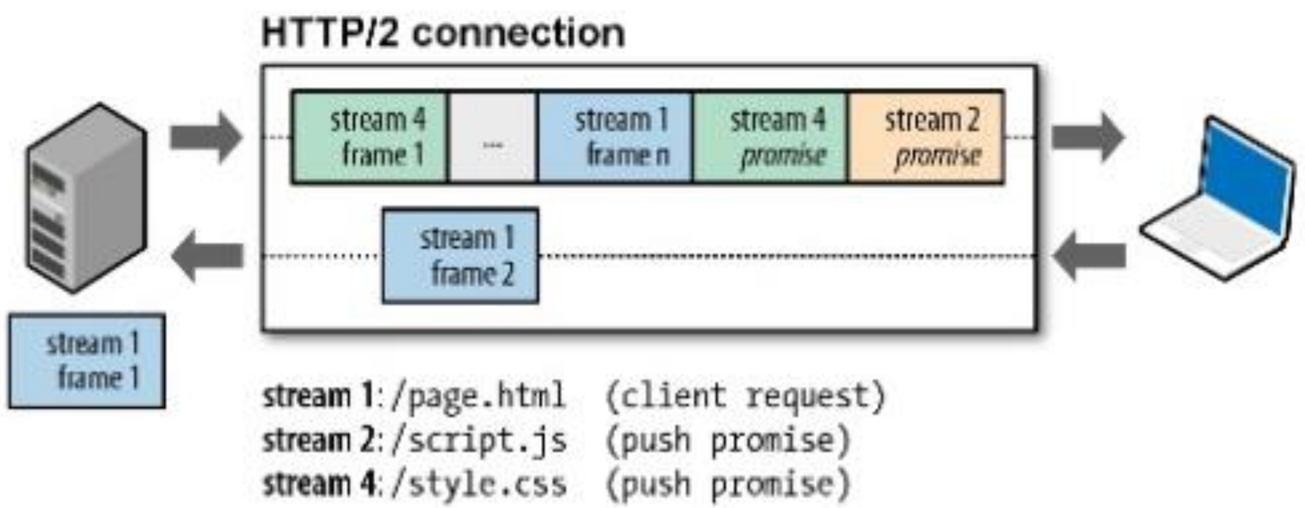
	Generated Message	Sent Message
First Request	GET /index.html HTTP/1.1 Host: www.keycdn.com Referer: https://www.keycdn.com/ Accept-Encoding: gzip	GET /index.html HTTP/1.1 Host: www.keycdn.com Referer: https://www.keycdn.com/ Accept-Encoding: gzip
Second Request	GET /logo.svg HTTP/1.1 Host: www.keycdn.com Referer: https://www.keycdn.com/index.html Accept-Encoding: gzip	GET /logo.svg HTTP/1.1 Host: www.keycdn.com Referer: https://www.keycdn.com/index.html Accept-Encoding: gzip

Multiplexing (1)

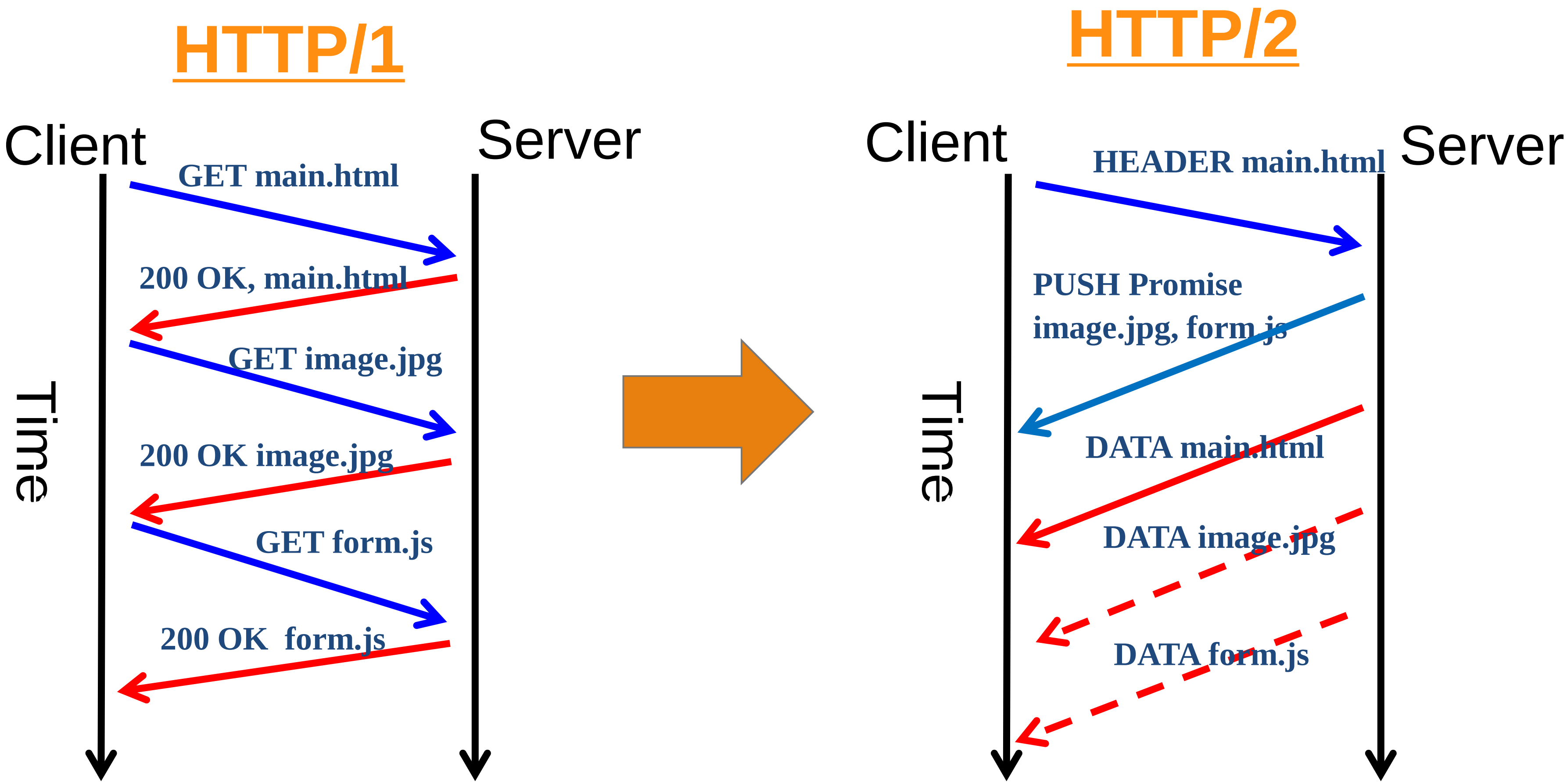


Multiplexing (2)

- The frame exchange between the *client* and the *server* is organized in *streams*
- A *stream* is a logic sequences of *frames*
- Every *stream* has a priority (set by the *browser*)



Server Push



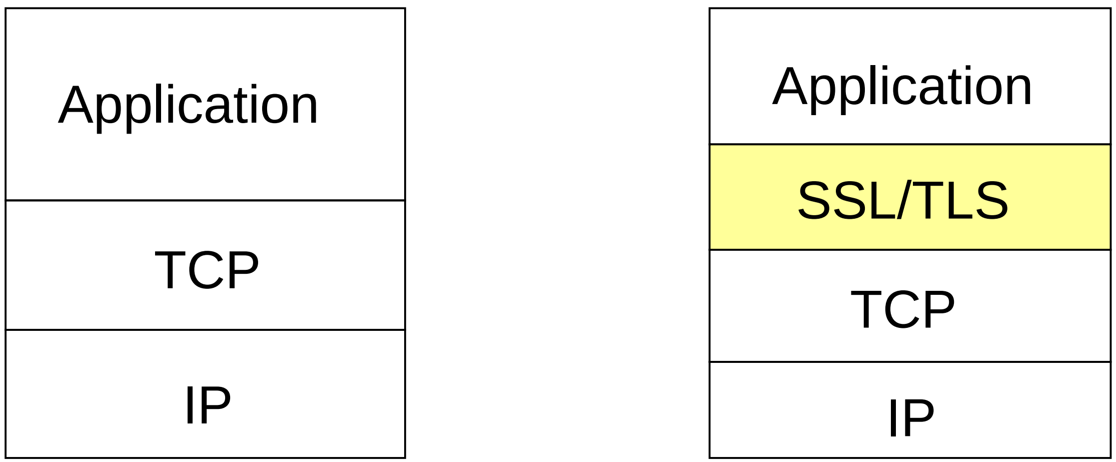
- The *server* can send useful information to the client before the client explicitly asks for it
- This functionality is asked by the *client*

Securing HTTP: HTTPs

- What could happen if transactions made with Amazon would be carried by HTTP?
  - A malevolent player could capture HTTP messages that contain, among other, credit card information (no *confidentiality* of data)
  - Or, it could forge/modify HTTP messages related to the transaction, making the user buy different items, more items than what specified etc… (no *integrity* of data)
  - Or, it could act as Amazon itself and steal information/money from the user (no *authentication* between client and server)

Solutions

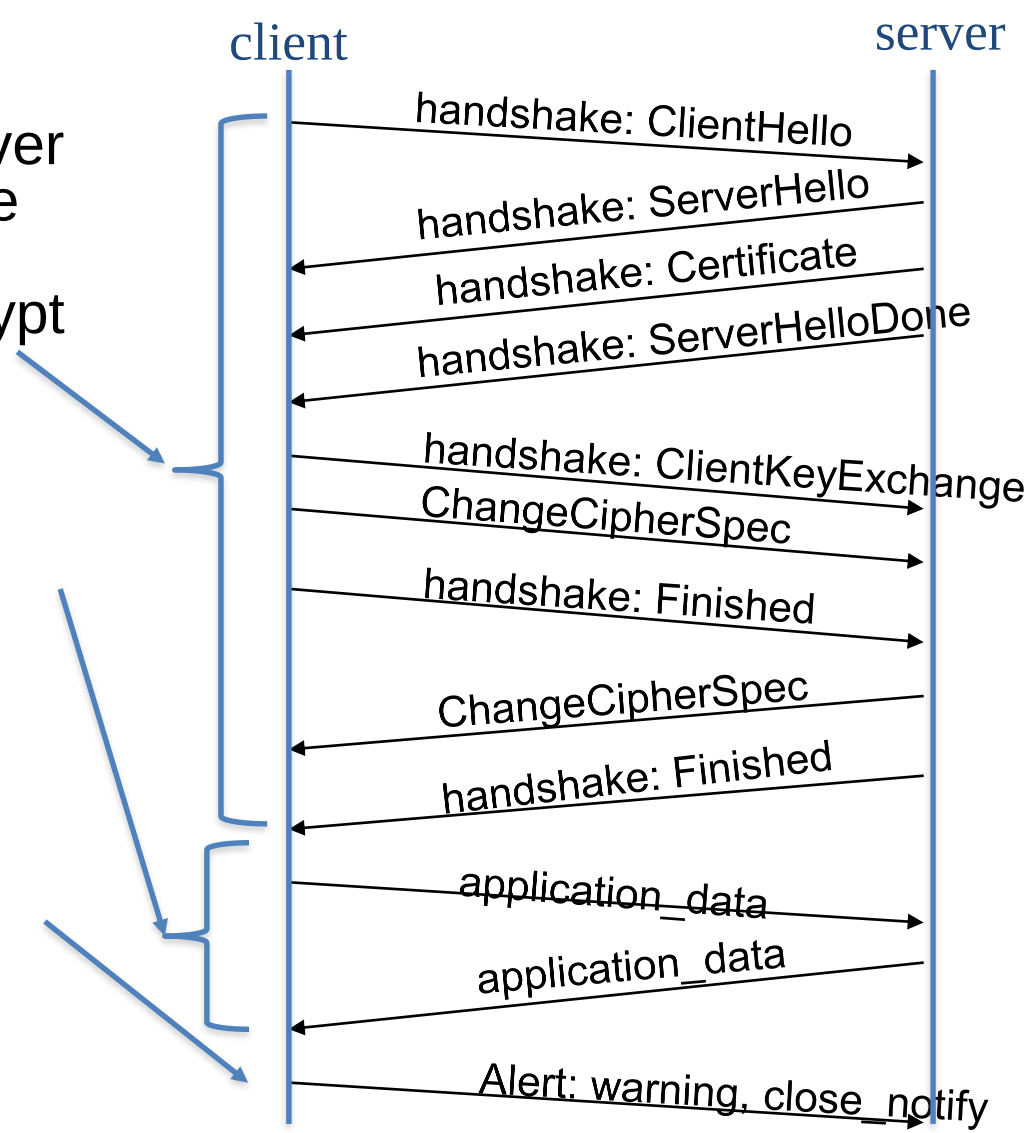
- *Secure Socket Layer* (SSL) and *Transport Layer Security* (TLS) add *confidentiality*, *integrity* and *authentication* to TCP connections



No security *Secure*

SSL/TLS connections

- *Handshake*:
  - Phase in which the server (and client) authenticate and agree on which technique used to encrypt data
- *Tdata transfer*
  - Data are divided in *records* (PDU), each of which is encrypted with the algorithm chosen in the 1st phase
- *Connection closing*
  - A special message is used to close the connection in a secure way



Handshake Phase

- Exchange of *certificate* between server and client (and viceversa) which certifies the identity of the server (client)
  - The certificate is generated by a *Certification Authority* (CA) and contains:
    - the *public key* of the certified entity
    - Additional information (IP address, name, etc)
    - Digital signature of the CA
- Generation and exchange of *symmetric keys* to encrypt the transferred data
- Such exchange of symmetric keys happens on a connection which is, in turn, encrypted with asymmetric keys

HTTP/3 (IETF Draft)

- HTTP over QUIC (a transport protocol)
  - QUIC already incorporates stream multiplexing and per-stream flow control, in a similar way to that provided by HTTP/2
  - QUIC also incorporates TLS 1.3 at the transport layer, offering comparable security to running TLS over TCP, with the improved connection setup latency of TCP Fast Open [RFC7413]
- HTTP/3 provides a transport for HTTP semantics using the QUIC transport protocol and an internal framing layer similar to HTTP/2.
  - Once a client knows that an HTTP/3 server exists at a certain endpoint, it opens a QUIC connection. QUIC provides protocol negotiation, stream-based multiplexing, and flow control.

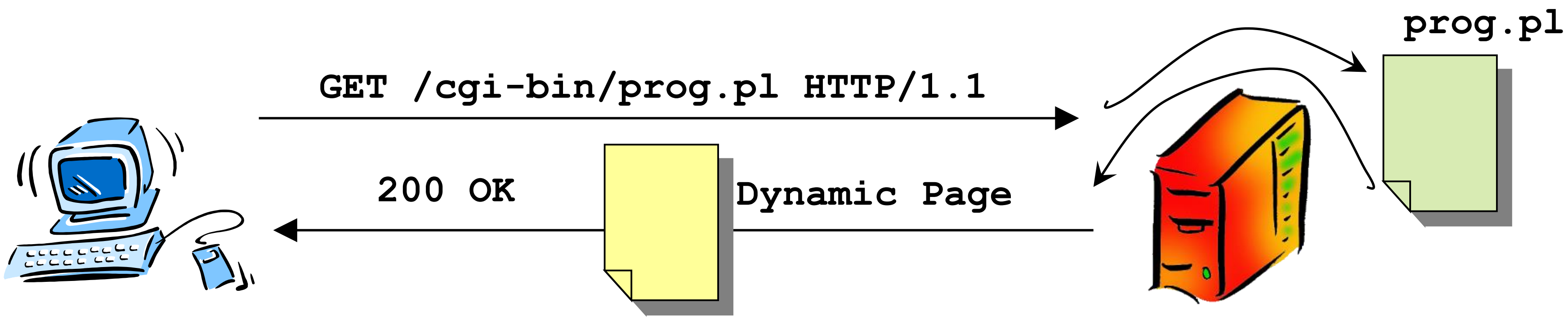
https://datatracker.ietf.org/doc/draft-ietf-quic-http/

HTML (HyperText Markup Language)

- HTTP handles the object transfer and does not account for the object format
- The visualization of the object is done through interpreter programs (browsers)
- Formatted text pages are transferred in ASCII files and are interpreted according to formatting instructions written in HTML
- HTML pages may contain references to other objects which need to be interpreted by the browser as
  - Part of the document to visualize
  - Links to other pages
- If a HTML page is stored on the server and is sent upon request, this is a *static page*

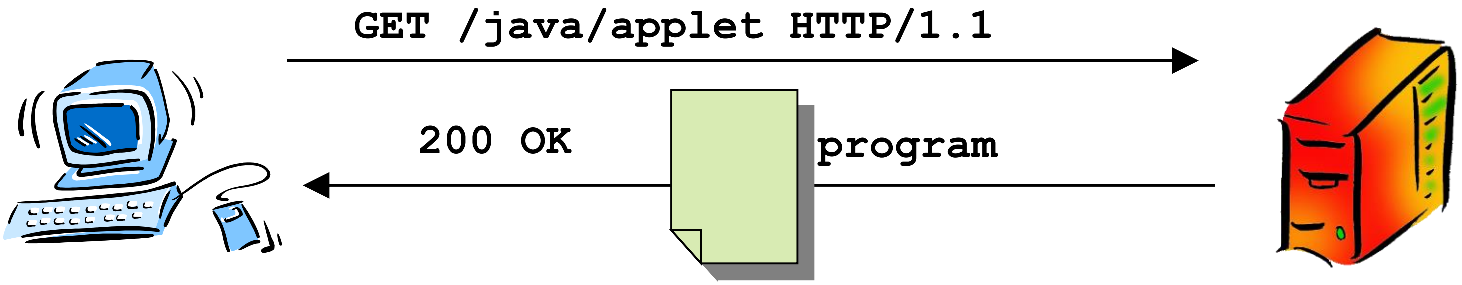
Dynamic WEB Pages

- If a page is created on the fly upon reception of a request, this is a dynamic page
- The server examines the request, executes a program associated to the request and generates the HTML page to be sent back



Active Web Pages

- A web page may contain a program to be executed by the client
- The program is downloaded and executed locally by the client
- This can be used to set up interactive pages, moving graphs, etc.

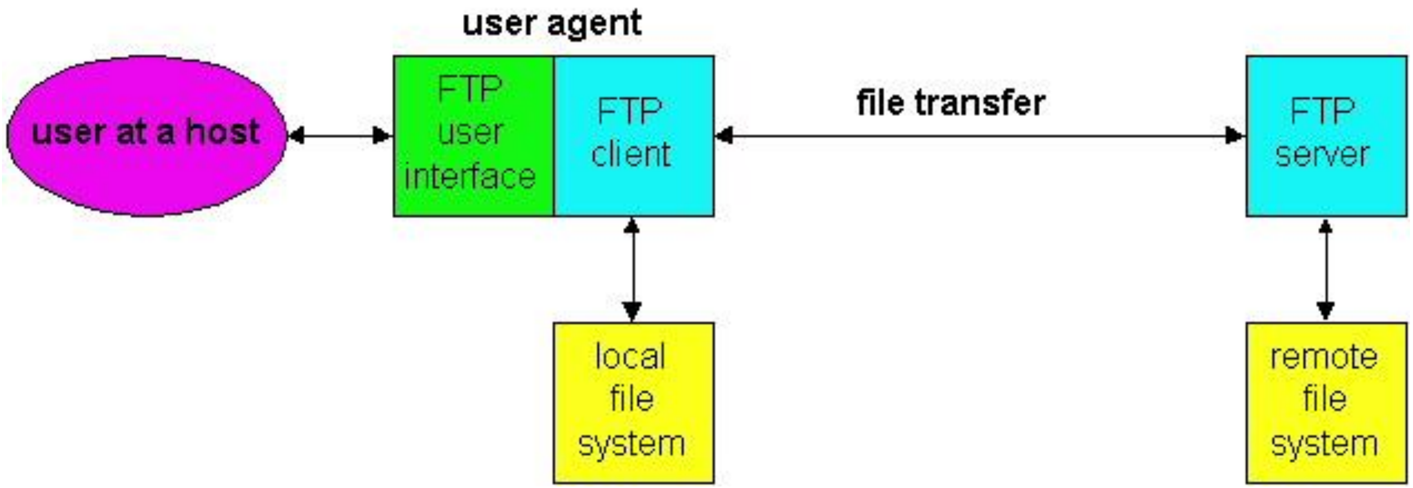


File Transfer Protocol (FTP)

File Transfer Protocol (FTP)

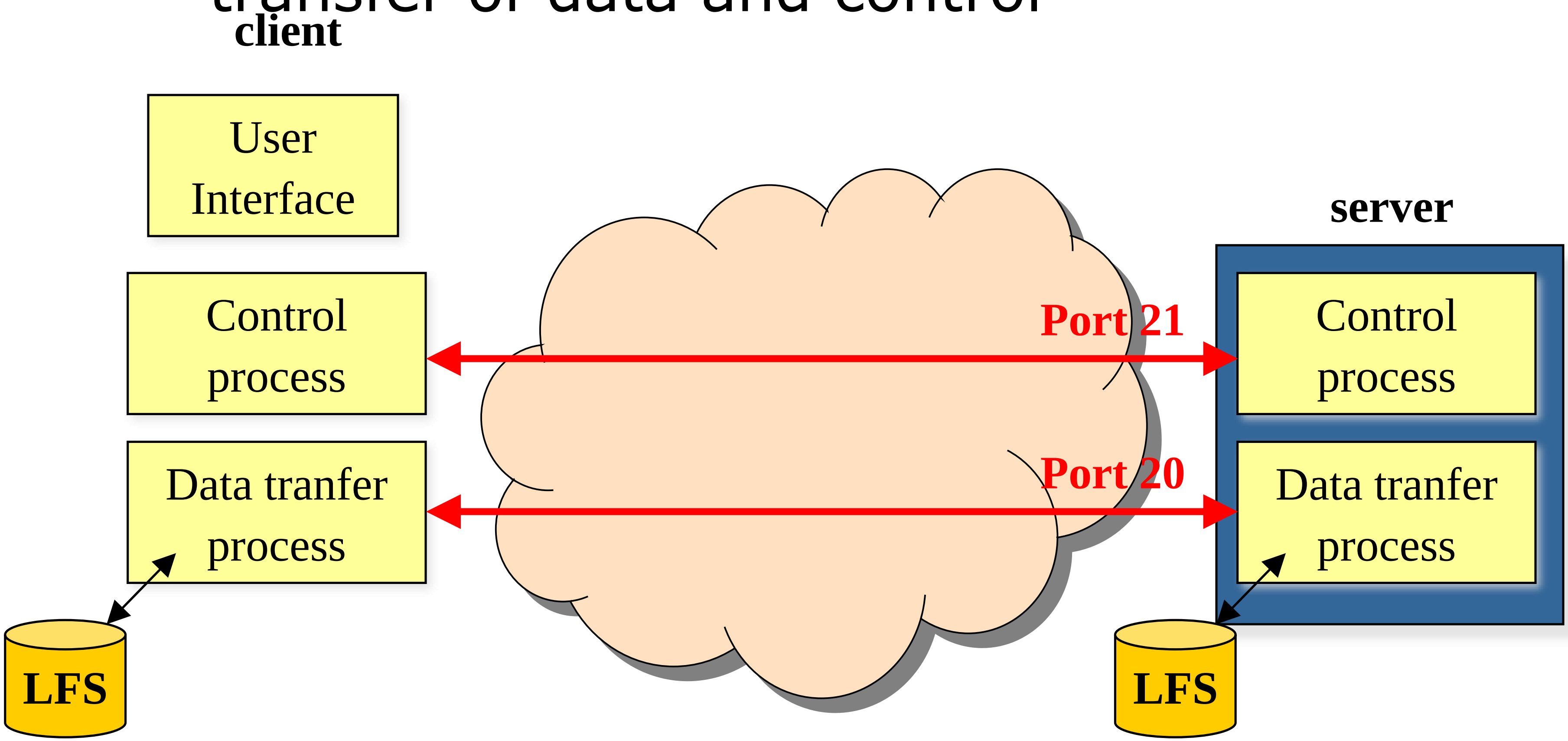
• "File Transfer Protocol", RFC 959, October 1985.

- Used to transfer files between two remote hosts
- The application operates directly on the file system (both at server and at client side)

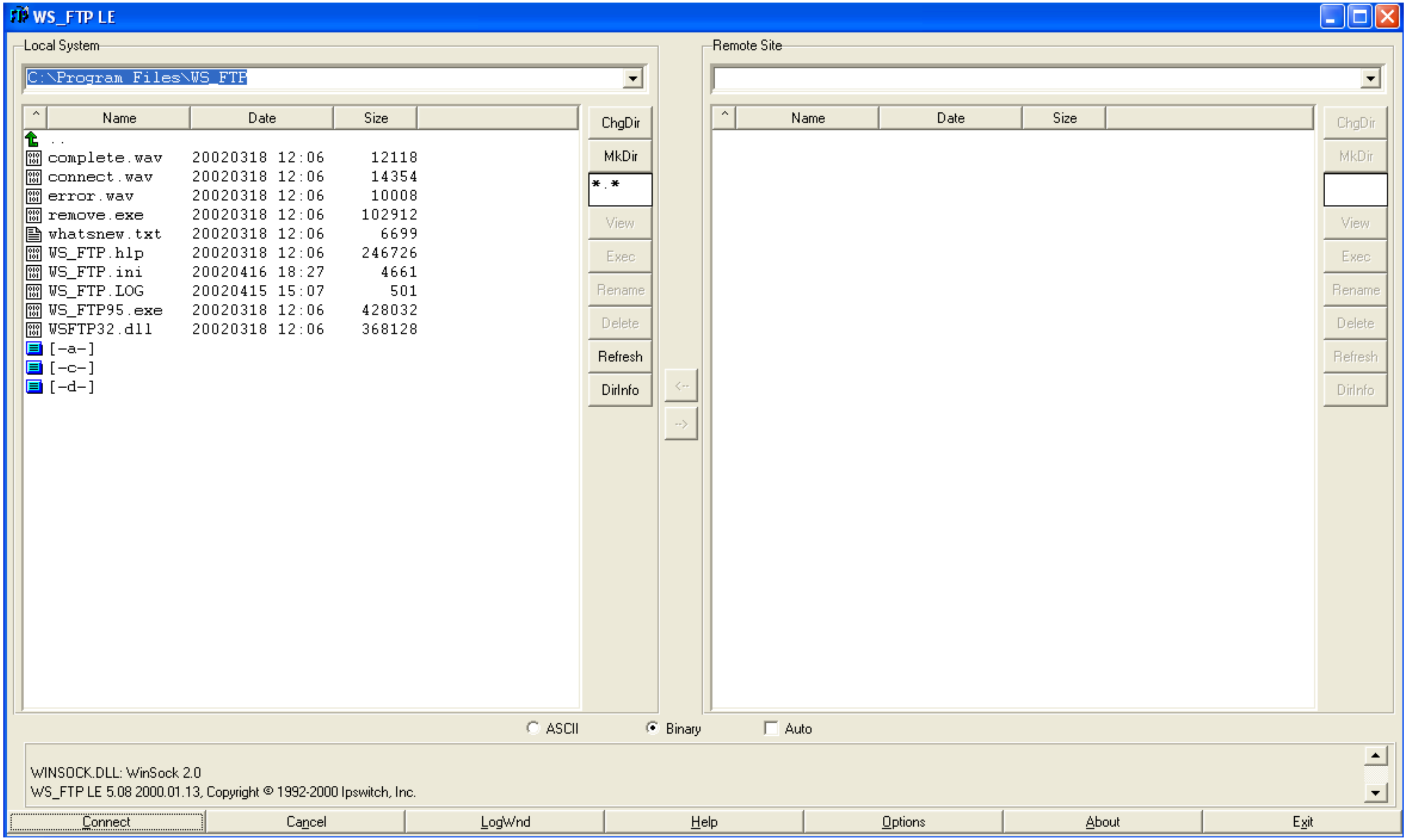


File Transfer Protocol (FTP)

- Uses TCP for the transfer
- Two TCP connections are used for the transfer of data and control

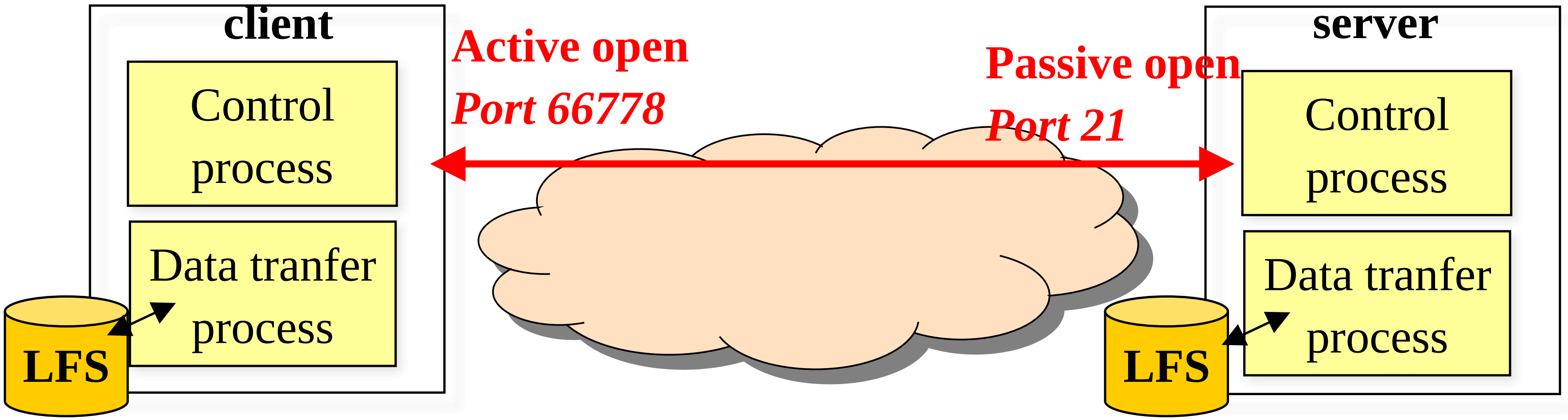


FTP: user interface



FTP: control connection

- It is opened in the usual way
  - The server issues a *passive open* with port number 21 and waits fir requests
  - The *client issues an active open* with a dynamic port number every time it needs to transfer files
- The control connection is *persistent*, and remains open for all the data transfer time



FTP: Data connections

- Data connections are *non-persistent,*
  - one connection for each file to transfer,
  - connection closed upon completion of file transfer
- To open a data connection:

  1st Way:
  - The *client* issues a passive open with a dynamic port number
  - The client notifies the port number to the server on the control connection through the PORT command
  - The server issues an *active open* towards the specified port of the client using 20 as local port number

  2nd Way:
  - The *client sends the* PASV command to the server
  - The *server* chooses a dynamic port number, issues a passive open and communicate the chosen port number to the client
  - The *client issues an active open* using the port number received from the server

FTP: Data connections

- The data transfer can be accomplished in different ways and using different formats:
- *File types*:
  - ASCII
  - *Binary*:
- *Transmission modes*:
  - *Stream mode*: the file is sent down to the TCP as a stream of unstructured bytes
  - *Block mode*: the file is structured in blocks with a header each and sent down to the TCP

FTP: commands

- Commands are transferred in ASCII

Access Commands

```
USER username
PASS password
QUIT log out
```

File Management

```
CWD  change directory
DELE delete file
LIST list files
RETR retrieve file
STOR store file
```

Transfer Management

```
TYPE file type
MODE transfer mode
```

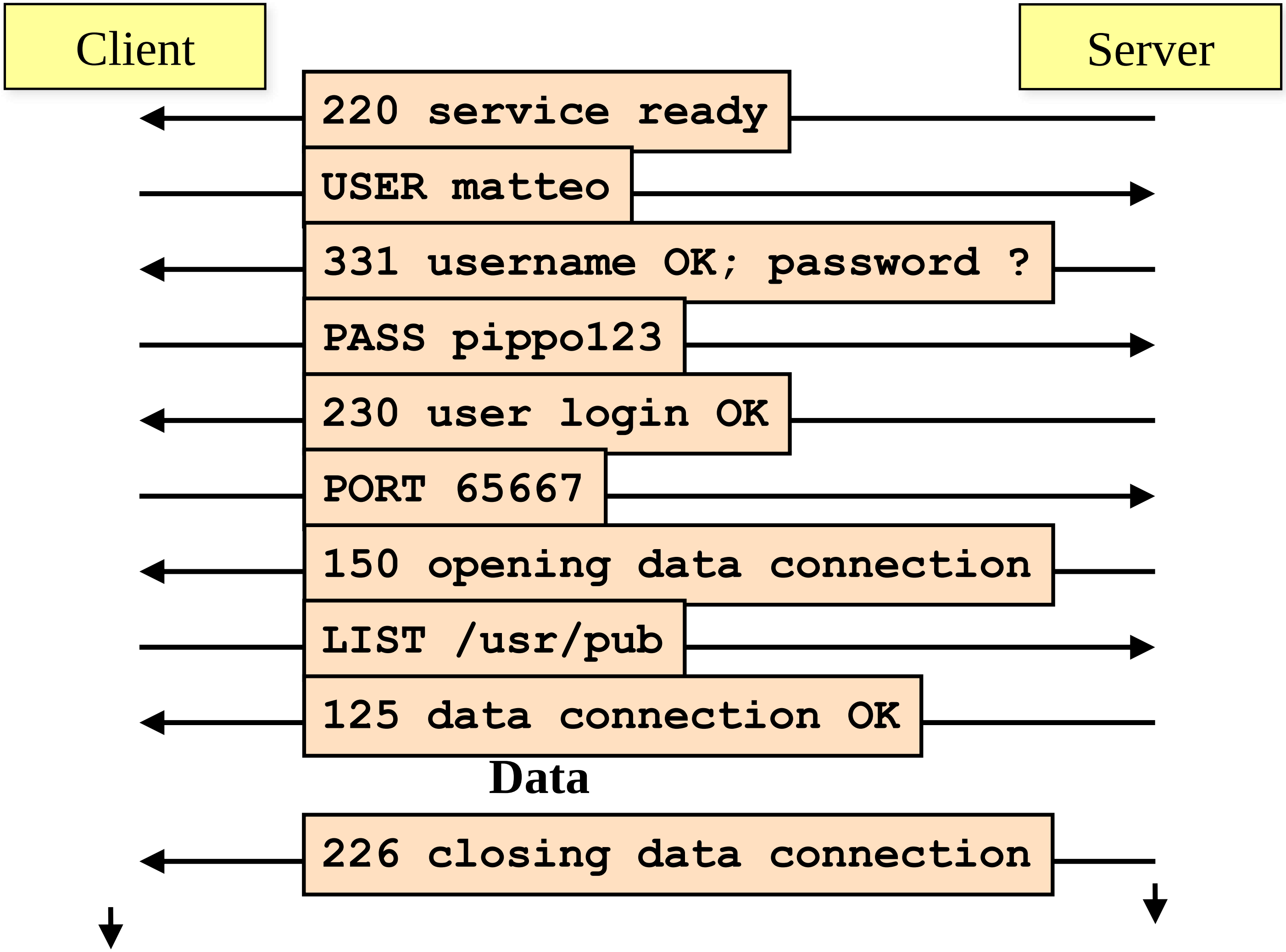
Port Management

```
PORT client port
PASV server choose port
```

FTP: Responses

```
125 Data connection already open; transfer starting
200 Command OK
225 Data connection open
226 Closing data connection
227 Entering passive mode; srv. sends Ip_add.,port
230 User login OK
331 Username OK, password required
425 Can't open data connection
426 Connection closed; tranfer aborted
452 Error writing file
500 Syntax error; unrecognized command
501 Syntax error in parameters or arguments
502 Command not implemented
```

FTP: Transfer Example

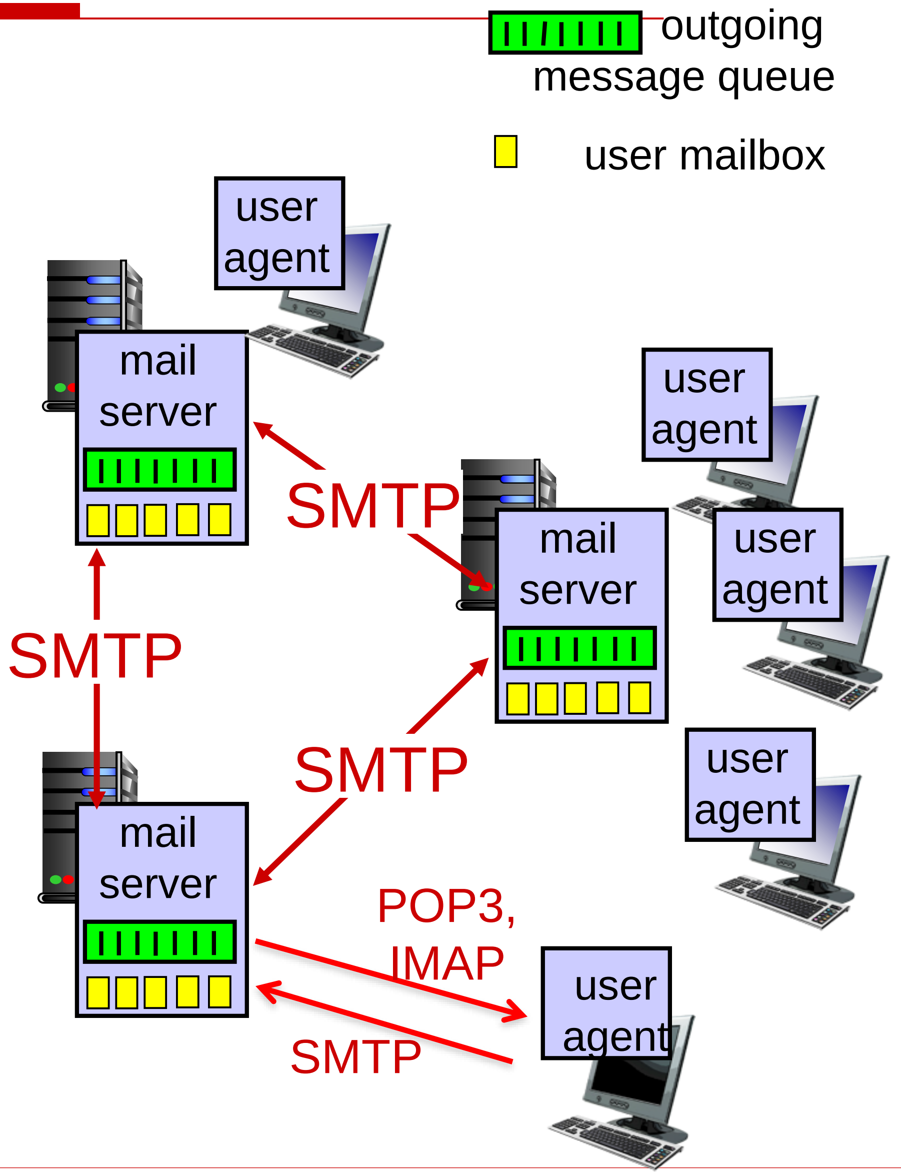


E-mail Service

Simple Mail Transfer Protocol (SMTP)

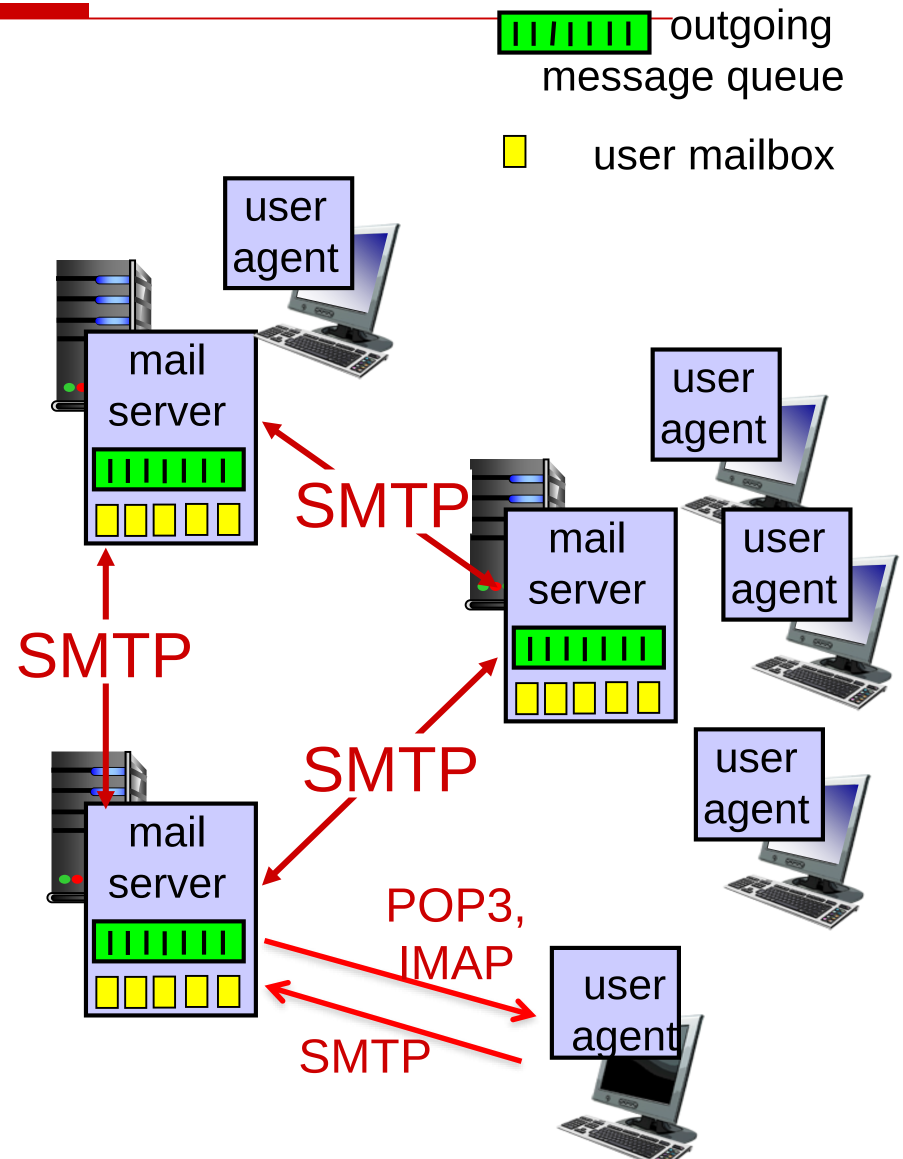
The e-mail service

- **Client** aka **User Agent** (OutLook, Thunderbird, etc.)
- **Mail Server**
- **Simple Mail Transfer Protocol SMTP**: to transfer email from client to the mail server of destination (recipient)
- **Access protocols** to mail servers: to "download" email from own mail server (POP3, IMAP)



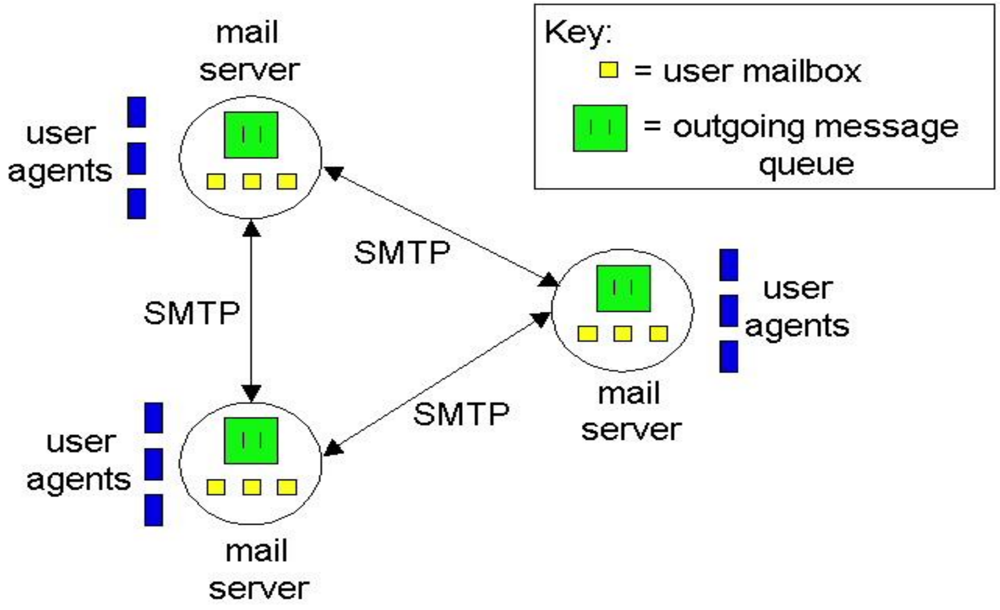
The e-mail service

- Mail servers contain for each controlled client:
  - An incoming email queue (**mailbox**)
  - An **outgoing mail queue**
- Mail servers
  - Receive all mails outgoing from client user «controlled» by them
  - Receive from other mail servers all mails destinaed to controlled clients
- Mail servers “speak”
  - **SMTP** with other mail servers and with clients in uplink
  - **POP3/IMAP** with clients in downlink



E-mail

- Service to send textual messages in an asynchronous way
- It is implemented through a network of mail servers using the SMTP (*Simple Mail Transfer Protocol*)



SMTP

J.B. Postel, "Simple Mail Transfer Protocol," RFC 821, August 1982.

- Textual protocol
- Also the body of the messages needs to be ASCII
  - Binaries must be converted to ASCII
- Once a *server* receives a message from a user *agent*
  - Stores the message in a queue
  - Opens a TCP connection (port 25) with the destination server
  - Sends the message

Client/Server Message exchange

Handshake

```
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
```

Message Format

D.H. Crocker, "Standard for the Format of ARPA Internet Text Messages," RFC 822, August 1982.

- The message format is specified (command DATA)
- Some headers are added to the message

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Request of information
<black line>
<Body>
.
```

Multipurpose Internet Mail Extensions (MIME)

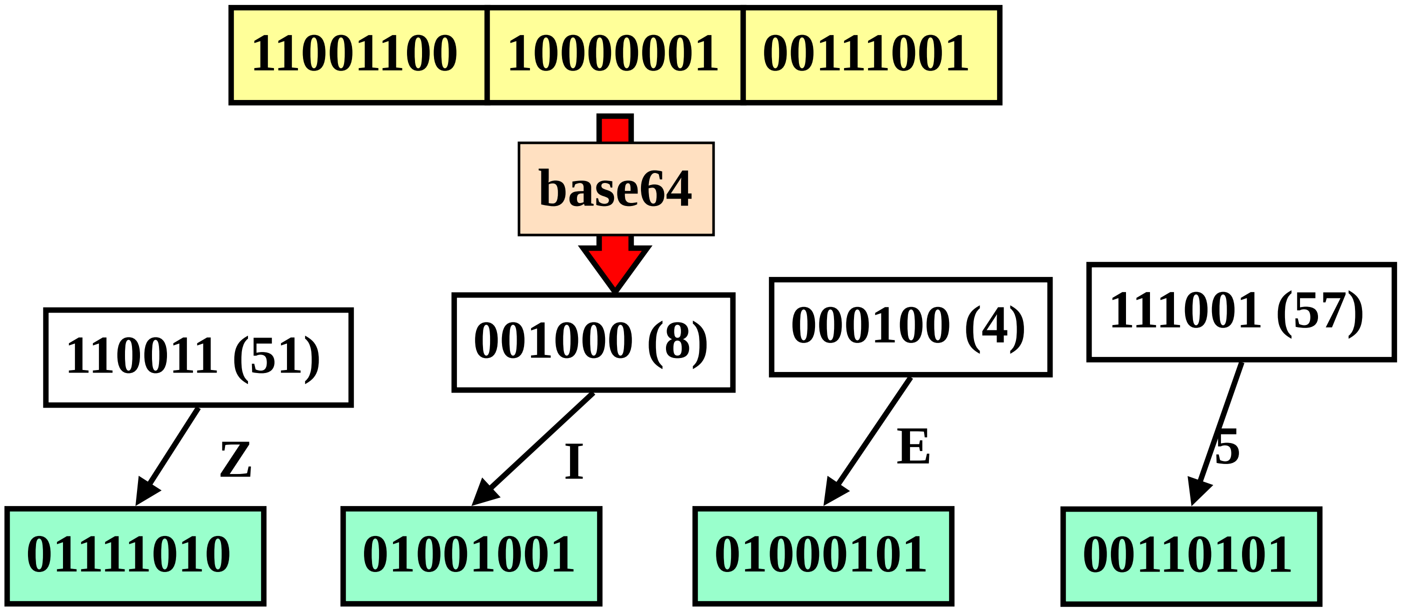
- "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies," RFC 2045, Nov. 1996.
- "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types," RFC 2046, Nov. 1996.

- MIME is used to allow the transfer of non-ASCII messages

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
base64 encoded data .....
.......................
.....base64 encoded data
.
```

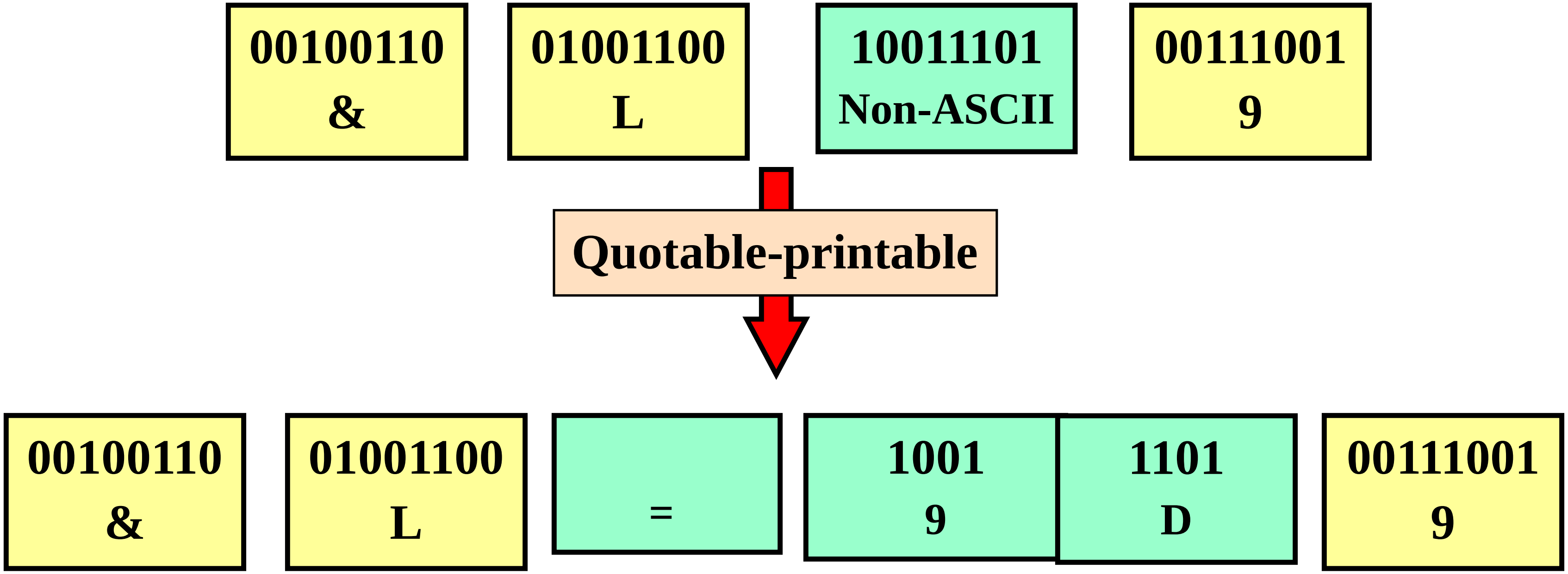
Multipurpose Internet Mail Extensions (MIME)

- Coding techniques:
  - Base64:
    - The flow of bits is divided into chunks of 24 bits each
    - Each chunk is divided into 4 groups of 6 bits each
    - Each chunk is interpreted as a character according to a conversion table



Multipurpose Internet Mail Extensions (MIME)

- Quoted-printable
  - The flow of bits is divided into chunks of 8 bits each
  - If a sequence corresponds to a ASCII character is sent straight away
  - Otherwise is sent as three characters: "=" followed by the hexadecimal representation of the byte

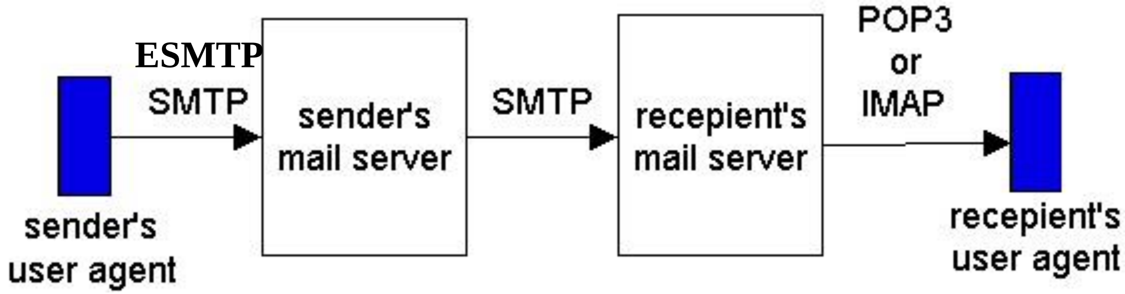


Multipurpose Internet Mail Extensions (MIME)

- MIME allows the transfer of multiple objects within the same message:

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe with commentary
MIME-Version: 1.0
Content-Type: multipart/mixed; Boundary=StartOfNextPart
--StartOfNextPart
Dear Bob,
Please find a picture of an absolutely scrumptious crepe.

--StartOfNextPart
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
base64 encoded data .....

--StartOfNextPart
Let me know if you would like the recipe.
.
```

MailBox Access Protocols



- POP3 (*Post Office Protocol version 3*)
- IMAP (*Internet Mail Access Protocol*)
- HTTP
- Security Issue: the protocols can run over TLS/SSL

POP3

Authorization Phase

- Client Commands:
  - **user**: username
  - **pass**: password
- Server Responses:
  - **+OK**
  - **-ERR**

Transaction Phase, client:

- **list**: list mess. number
- **retr**: get message
- **dele**: delete message
- **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
```

Commands

Login:

USER <username>
PASS <password>

Server responses:

-ERR
+OK

Common Operations:

- **STAT**
  info on the mbox status
- **LIST**
  list # of messages
- **RETR *n***
  read message *n*
- **DELE *n***
  delete message *n*
- **RSET**
  cancel delete operations
- **QUIT**
  exits
- **CAPA**
  show server capabilities

Case History

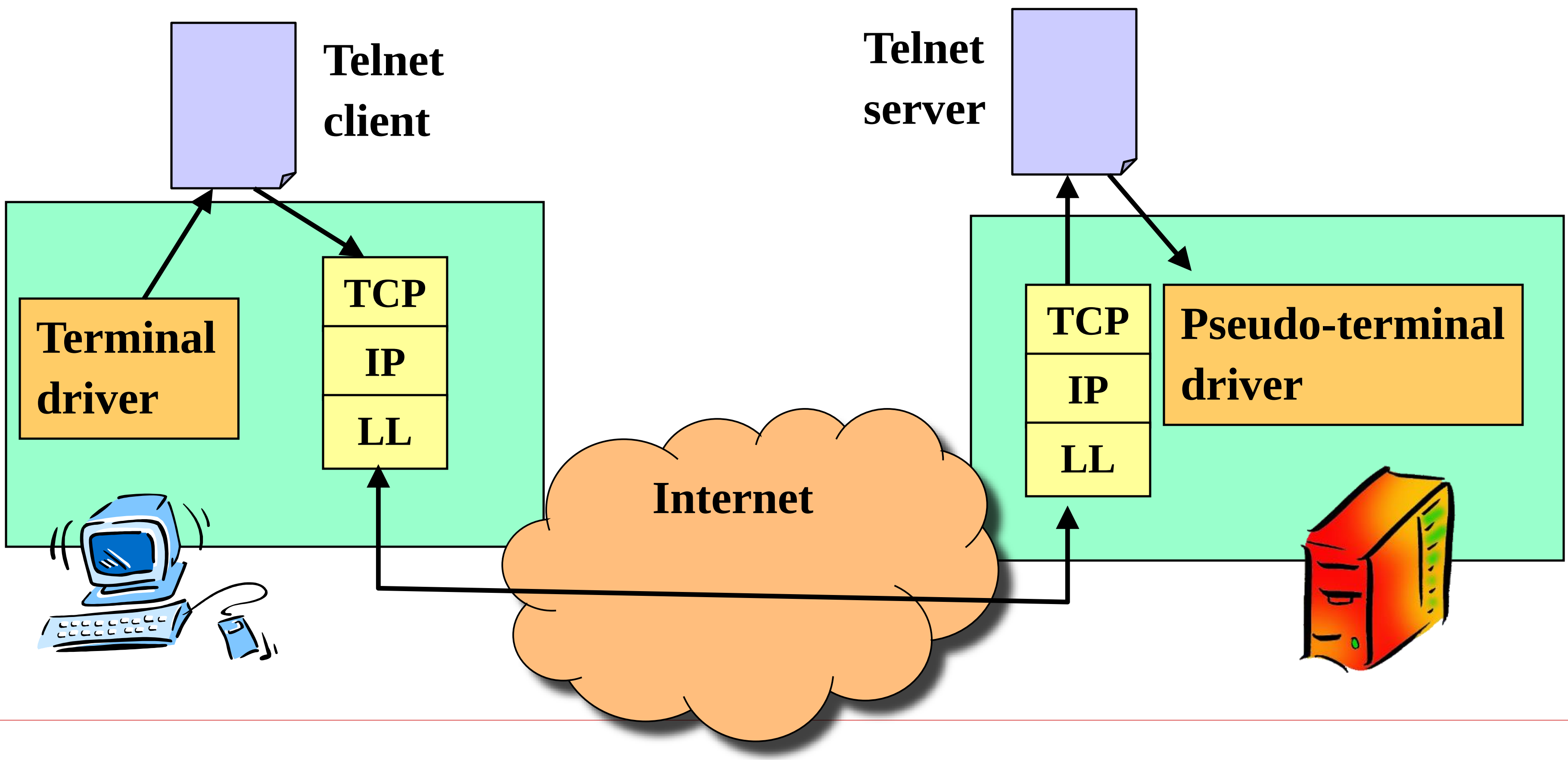
- December 1995, S. Bhatia and J. Smith propose the first web based e-mail service (*Hotmail)*
- In 1 month, 100K users
- In 1 year, 12M users
- December 1997, Hotmail is acquired by Microsoft for $400M
- Example of "*first mover advantage*" and "*viral marketing*"

Remote Terminal

TELNET

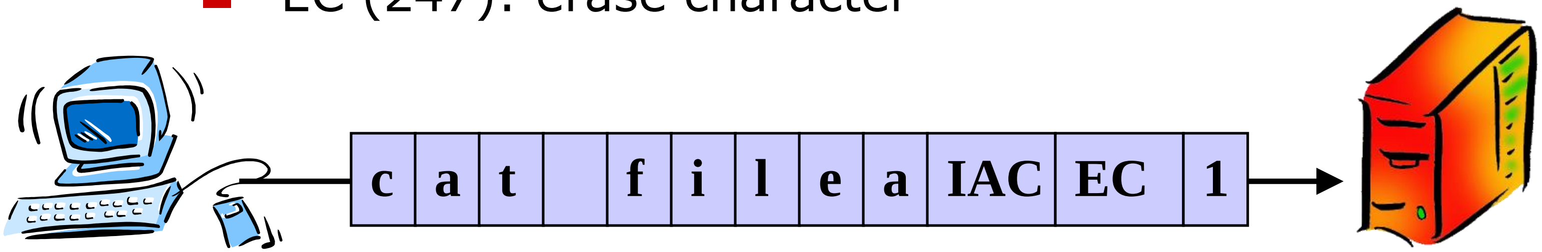
TELNET (TErminaL NETwork)

- Remote terminal application
- The commands are transferred through a TCP connection

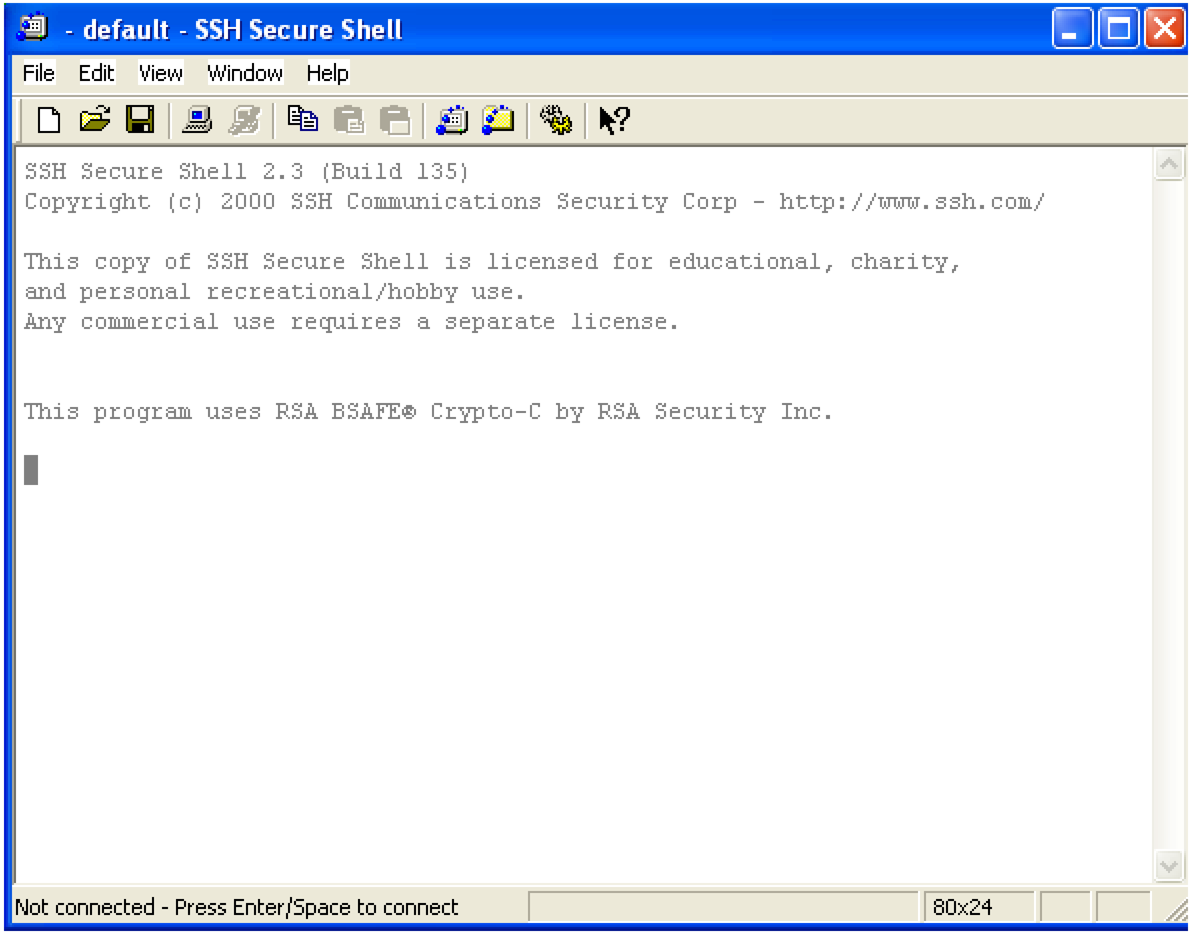


TELNET (TErminaL NETwork)

- TELNET transfers characters
  - Data characters:
    - ASCII with the first byte "0"
    - ASCII characters with the first byte "1" (preceded by a special control byte)
  - Control characters:
    - 8 bit commands (first bit "1")
    - examples
      - IAC (255): next one is a control character
      - EC (247): erase character



TELNET (TErminaL NETwork)



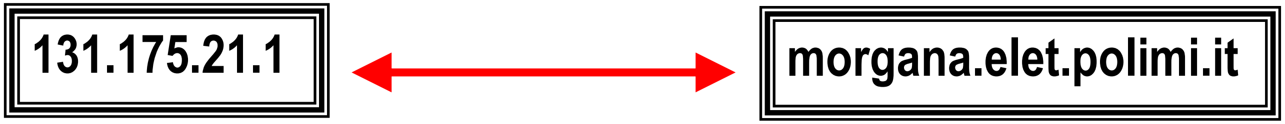
Domain Name System (DNS)

Domain Name System (DNS)

- IP addresses are not suited to be used by applications

Is it better www.google.com or 74.125.206.99?

- Symbolic addresses are more convenient
  - Hierarchical (street, city, state)
  - Independent from layer 3
- Binding is needed

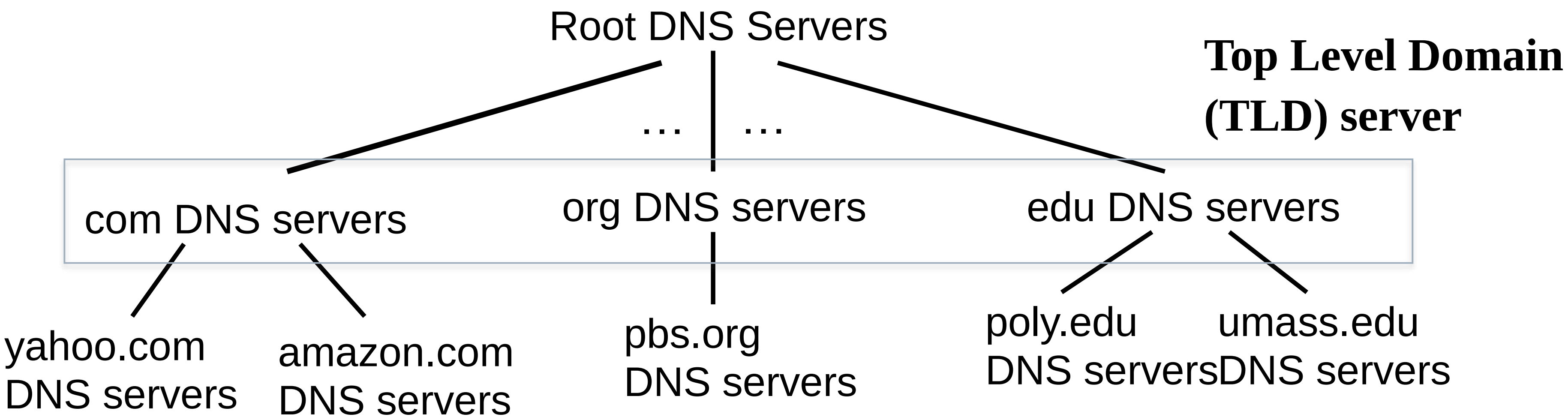


Domain Name System (DNS)

"Domain Names - Concepts and Facilities," RFC 1034, Nov. 1987.
"Domain Names - Implementation and Specification," RFC 1035, Nov. 1987.

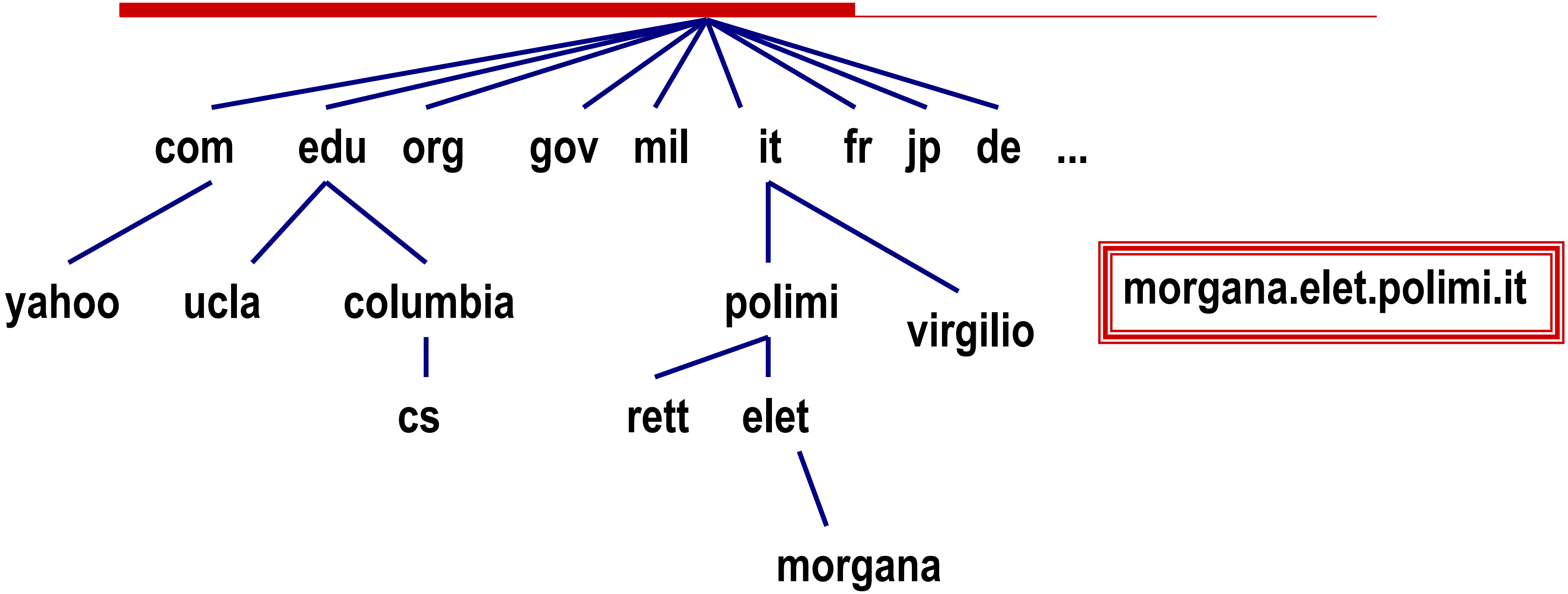
- IP networks provide a symbolic addressing service
- Supported by a distributed database service which handles the binding: DNS (*Domain Name System*)
- DNS is an application layer protocol which uses UDP/IP to transfer its messages
- DNS is currently used also for
  - Host aliasing
  - Mail server aliasing
  - Load distribution

Distributed, hierarchical database



- Each level in the hierarchy has a different «depth» of information
- Example: a user wants the IP address of www.google.com
  - *Root name servers know how to «find» name servers that manage .com domains*
  - *.com servers know hw to find the name server that manages the google.com domain*
  - *google.com name servers know how to resolve the symbolic name www.google.com*

Symbolic Addressing

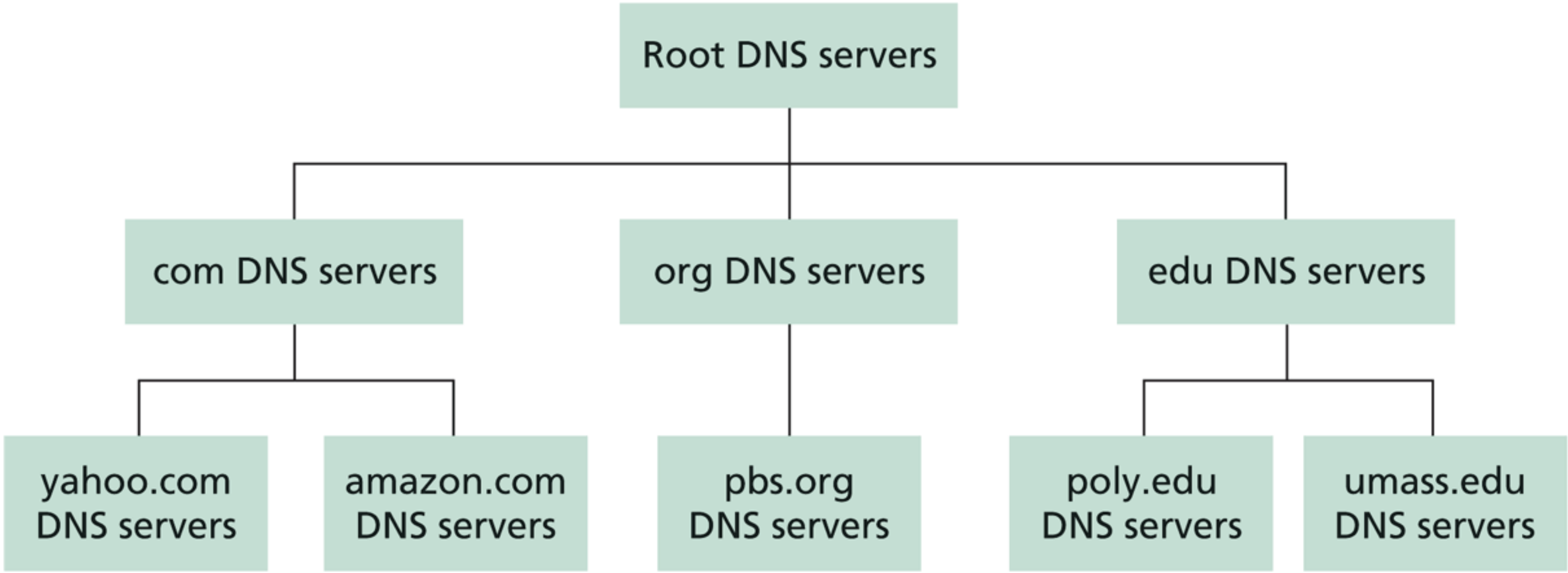


- Hierarchical Addressing
- Each branch is controlled by a known authority
- To get a symbolic address you must go through these authorities

Types of Name Servers

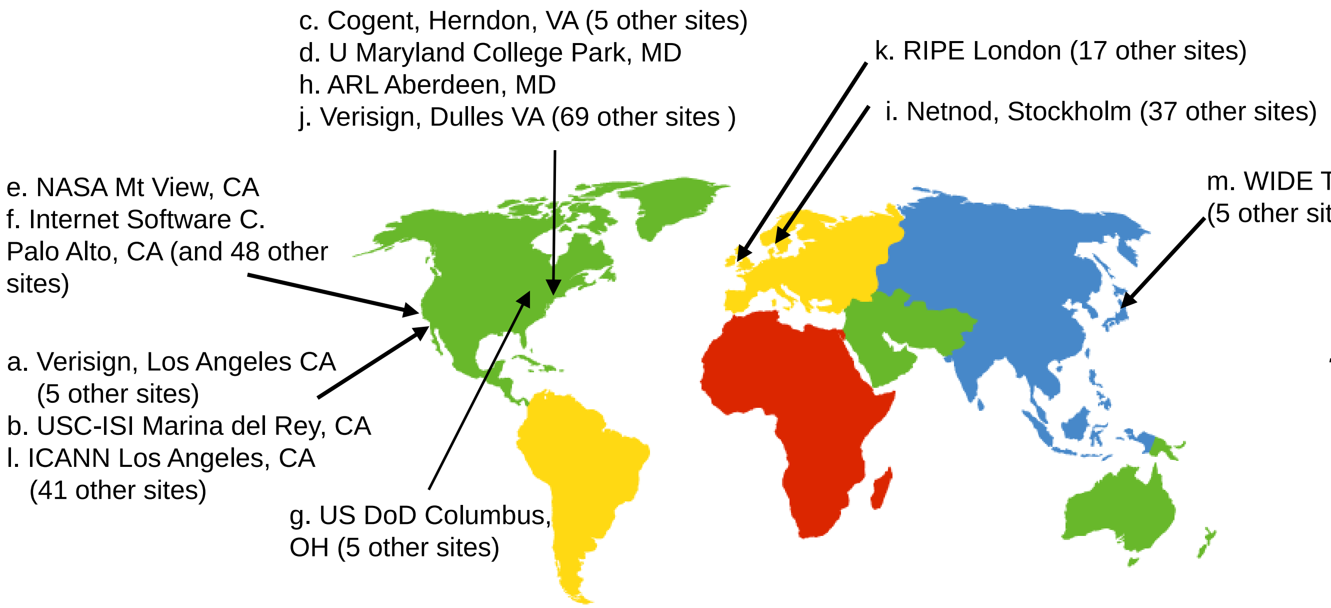
- *Local Name Servers*
  - Directly connected to tyhe hosts
  - Each ISP (residential, university, industry, etc.) has a LNS
  - Talks with the Root NS
- *Root Name Servers*
  - Stores info on the addressing of big groups of hosts and domains
  - Stores info on the *authoritative* NS for a given domain
  - Talks with the Authoritative NS
- *Authoritative Name Servers*
  - NS responsible for a specific hostname

Hierarchical DNS



Source: Computer Networking, J. Kurose

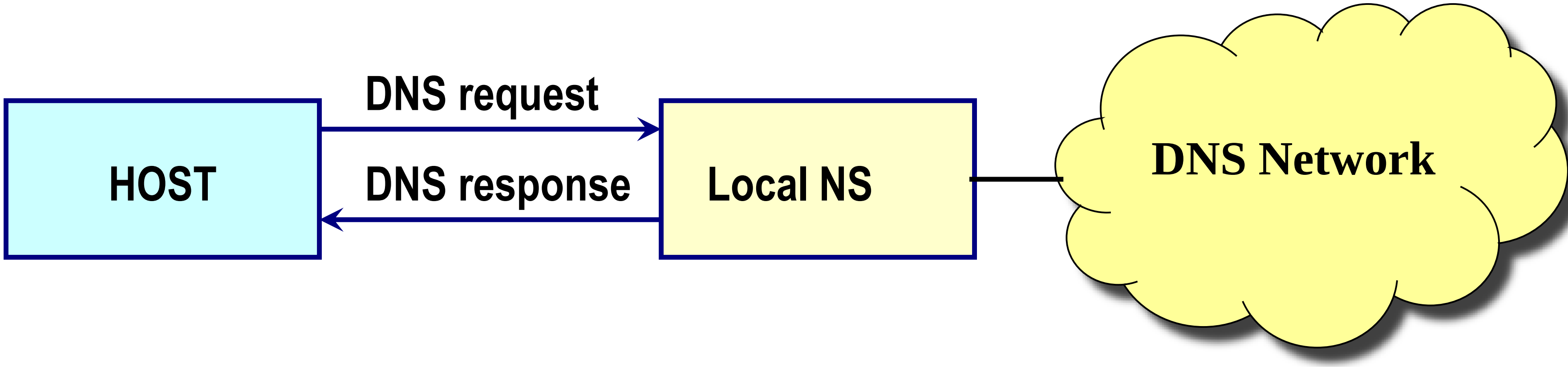
Root NS



13 root name "servers" worldwide

How To resolve a binding

- Every host knows the LNS address
- Each request for resolving a binding is sent to the local NS using UDP
- The LNS gets the info and answers



Stored Info

Name, Value, Type, TTL

- Type
  - A: *Name* is a host name and *Value* is the IP address

    (morgana.elet.polimi.it, 131.175.21.1, A, TTL)
  - NS: *Name* is a *domain* and *Value* is the symbolic name of a *server* which knows how to resolve the name

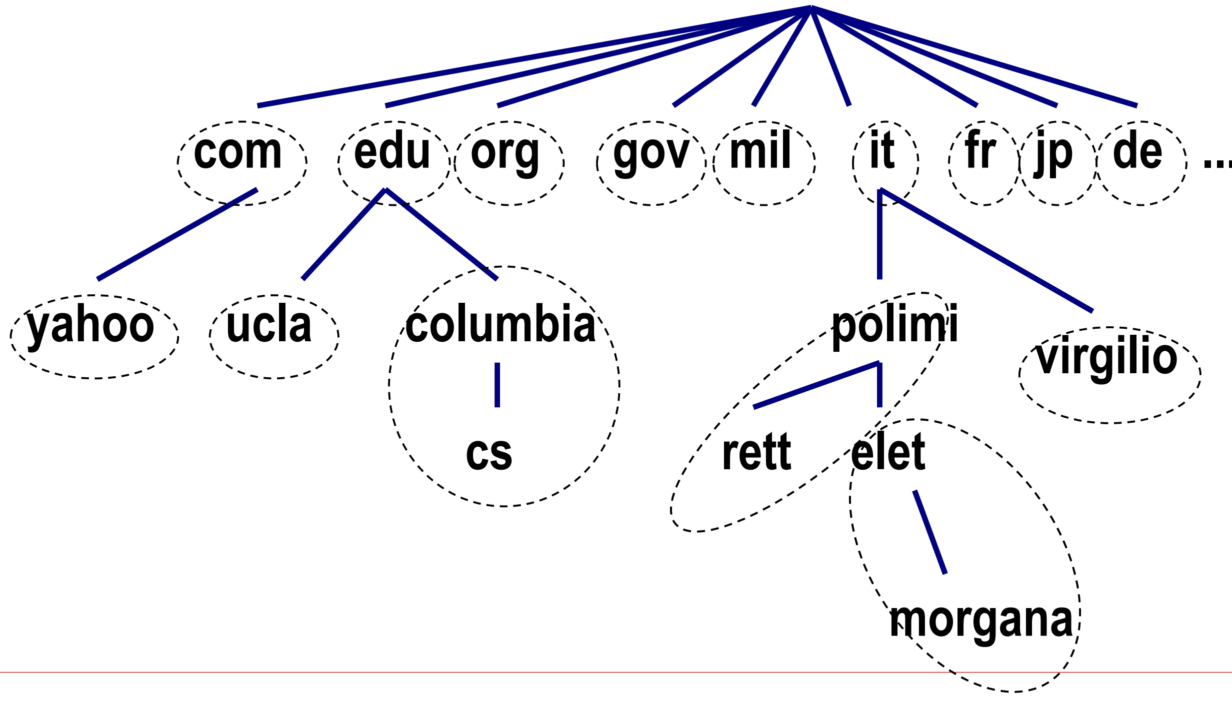
    (elet.polimi.it, morgana.elet.polimi.it, NS, TTL)
  - CNAME: *Name* is an alias and *Value* is the real name

    (www.polimi.it, zephyro.rett.polimi.it, CNAME, TTL)
  - MX: *Name* is a *mail domain or* a mail alias and Value is the name of the *mail server*

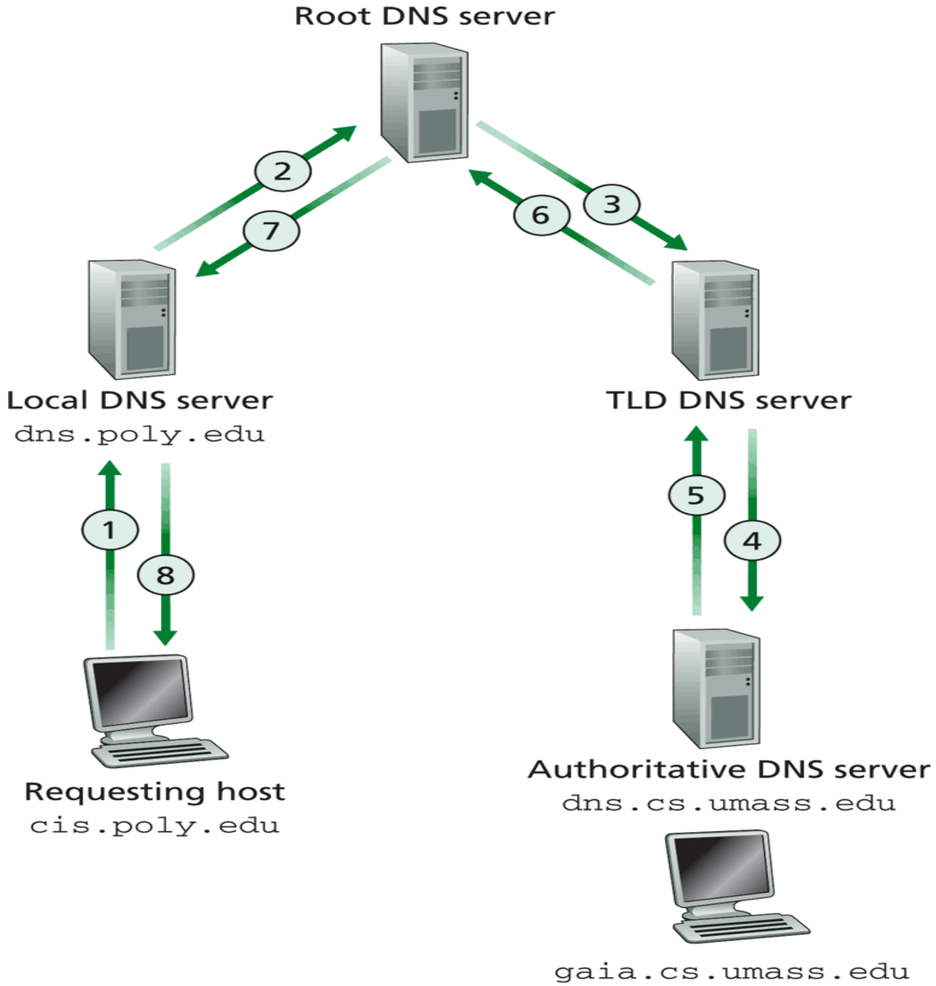
    (elet.polimi.it, mailserver.elet.polimi.it, MX,TTL)

Database Organization

- ARPANET was using a central database
- Internet uses a distributed database structure
- Branches are divided into zones and each zone is associated a DNS
- The server of a zone is *authoritative* for that zone



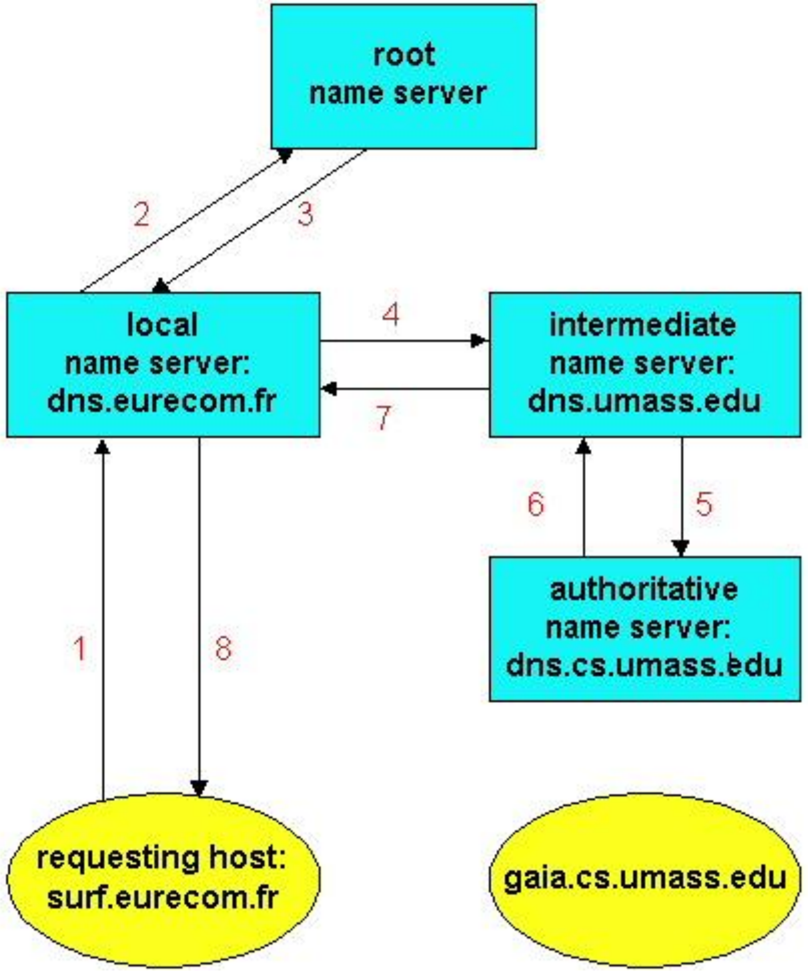
How to get Info



Source: Computer Networking, J. Kurose

- Recursive Way:
  - Requests travel along the hierarchy
  - Responses travel the opposite direction

How to get Info



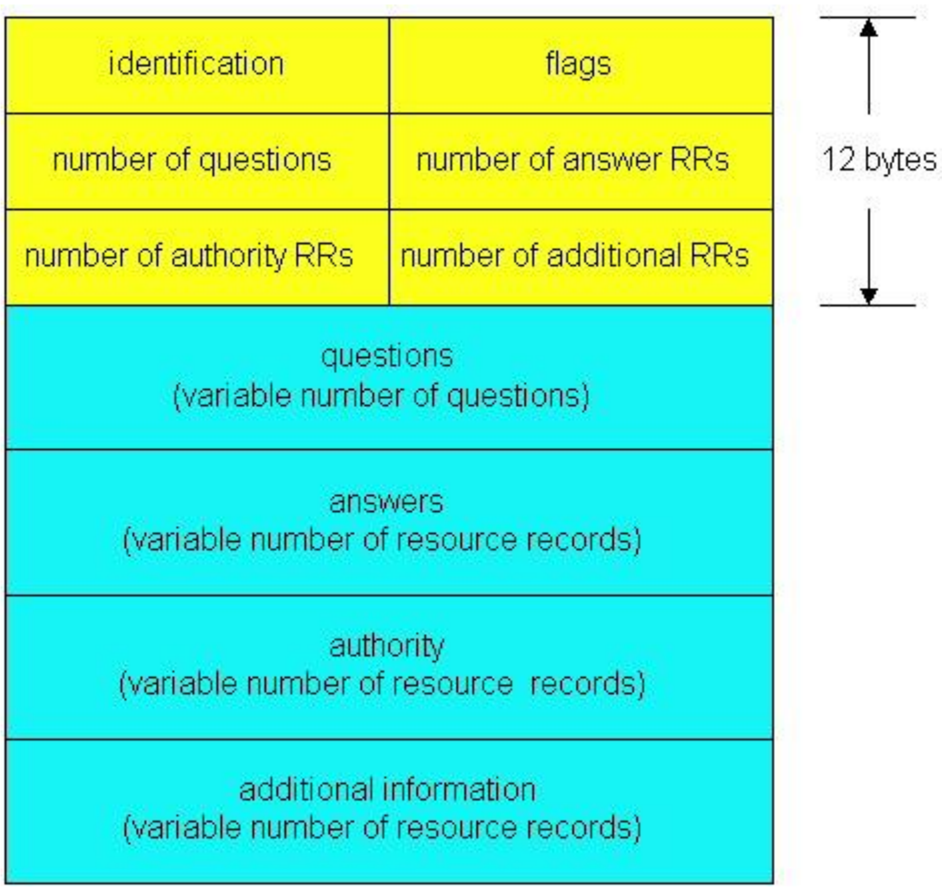
- Iterative Way:
- A server can notify the name of another server where to get the info from

Caching

- A server can cache a info temporarily
- If a request is issued regarding cached info the server can answer even if it is not authoritative for that specific info
- TTL is set by the authoritative server to advertise the "freshness" of a piece of info
- The *non-authoritative* server uses the TTL to set a validity timer for the cached info

DNS Messages

Binary Format (not ASCII)



- *identification*: identifies the couple request/response
- *flag:* request/response, authoritative/non auth., iterative/recursive
- *number of*: field sin the following header sections
- *questions*: name to resolve (usually A or MX)
- *answers*: complete resource records
- *authority*: contains other record provided by other servers
- *additional info*

How to add a new domain to the DNS

- The new startup *I-Like-Networking* vuole wants to register the domain *I-Like-Networking.com* (let us suppose this domain is free)
- *I-Like-Networking* register this domain in one of the *DNS Registrars*
  - *I-Like-Networking* must give to the *DNS registrar* the symbolic name and the corresponding IP addresses of the authoritative name servers
  - The DNS registrar inserts two RR nel TLD server .com

    ```
    I-Like-Networking, dsn1.I-Like-Networking.com, NS
    dns1.I-Like-Networking.com, 212.212.212.1, A
    ```
  - The *DNS registrar* eventually writes a record of type MX for *I-Like-Networking.com*

Simple examples with *nslookup*

- You can use the command `nslookup that permits to send DNS requests to a given` server
- You can look how it works:

  ```
  man nslookup
  ```

- Try to solve a symbolic name:

  ```
  nslookup www.unibg.it
  ```

- Let's find the authoritative name servers for a given domain

  ```
  nslookup -type=NS unibg.it
  ```

Try to find an authoritative response for the symbolic name *www.google.com*

Simple examples with *dig*

- The command *dig* (similar to `nslookup`) gives more details on the DNS messages exchanged
- Try a simple query

dig www.polimi.it

Simple examples with *dig*

```
wMacBook-Pro-di-Matteo:~ teo1$ dig www.polimi.it

; <<>> DiG 9.8.3-P1 <<>> www.polimi.it
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10838
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 2

;; QUESTION SECTION:
;www.polimi.it.                 IN      A

;; ANSWER SECTION:
www.polimi.it.          134     IN      A       131.175.187.72

;; AUTHORITY SECTION:
polimi.it.              1152    IN      NS      ns.polimi.it.
polimi.it.              1152    IN      NS      dns.cineca.it.
polimi.it.              1152    IN      NS      ns2.polimi.it.

;; ADDITIONAL SECTION:
ns2.polimi.it.          2488    IN      A       131.175.12.2
ns.polimi.it.           1546    IN      A       131.175.12.1

;; Query time: 3 msec
;; SERVER: 10.248.17.11#53(10.248.17.11)
;; WHEN: Wed Jan 20 10:30:56 2016
;; MSG SIZE  rcvd: 139

MacBook-Pro-di-Matteo:~ teo1$
```

Header of the DNS message

Description of the request

Response

Authoritative server for the requested domain

Additional information

Information on the performance of the request

Experimentation with *dig*

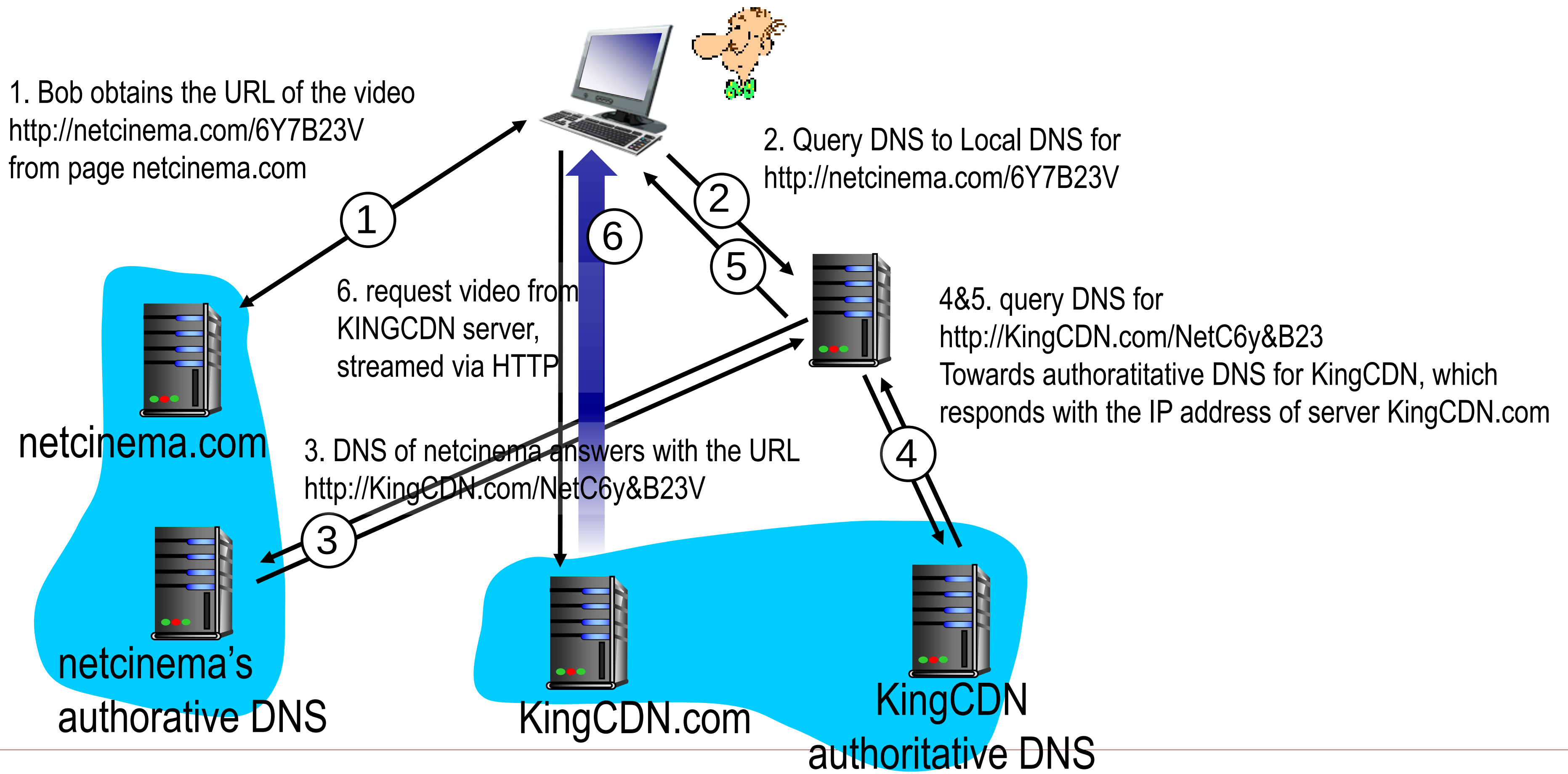
- If you want only the NS records
  ```
  dig -t NS polimi.it +noall +answer
  ```
- If you want only the list of MX records
  ```
  dig -t MX polimi.it +noall +answer
  ```
- If you the list of all records available
  ```
  dig -t  ANY polimi.it +noall +answer
  ```
- dig permits also to analyze the sequence of DNS requests for each query
  ```
  dig -t A polimi.it +noall +answer
  +trace
  ```

Content Delivery Networks

- *Problem:*
  - How to efficiently distribute several contents (video) at the same time to several users (very) far from each other
- *Solution:*
  - Build a network of geographically distributed servers that host copies of the requested content (similarly to a very big distributed cache)
  - *This network of servers (Content Delivery Network, CDN) can be built and owned by the content provider (Google, Netflix, Facebook) or by third parties (Akamai, Limelight, KCDN)*

CDN: Example of access to contents

The firm NetCinema relies on a CDN managed by KingCDN
Bob (client) requests a video http://netcinema.com/6Y7B23V
The video is found in the CDN at http://KingCDN.com/NetC6y&B23V



Choice of the best server

- ***Closest***: choose the closest serve (geographically speaking) to the client

- ***Shortest path***: choose the server with the lowest number of hops towards the client

- ***Let the user decide***: give to the user a list of possible servers, and the user chooses the best (*Netflix*)

Peer-to-Peer Architectures

File sharing, architectures, search

P2P file sharing

Example

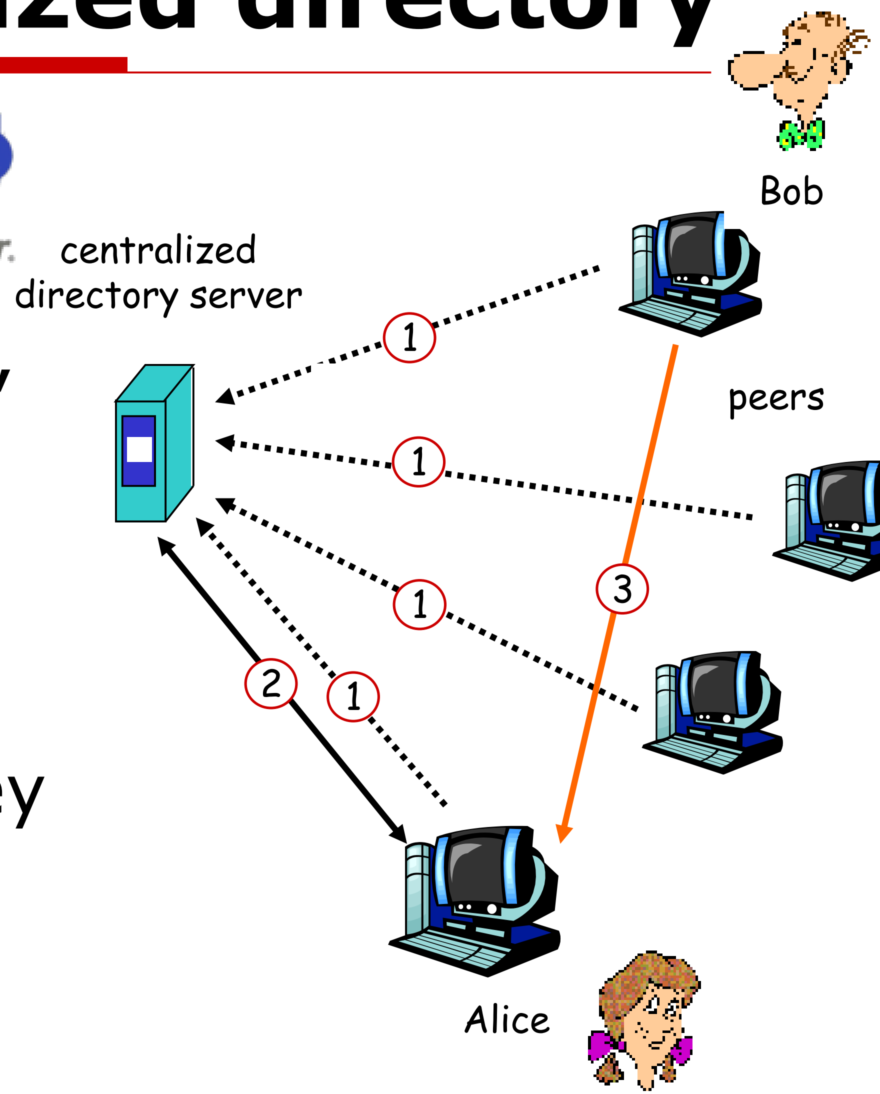
- Alice runs a P2P client application on her notebook computer
- Intermittently connects to Internet; gets new IP address for each connection
- Asks for “Hey Jude”
- Application displays other peers that have copy of Hey Jude.
- Alice chooses one of the peers, Bob.
- File is copied from Bob’s PC to Alice’s notebook: HTTP
- While Alice downloads, other users uploading from Alice.
- Alice’s peer is both a Web client and a transient Web server.

All peers are servers = highly scalable!

P2P: centralized directory

original “Napster” design

1) when peer connects, it informs central server:
   - IP address
   - Shared files
2) Alice queries for “Hey Jude”
3) Alice requests file from Bob



P2P: problems with centralized directory

- Single point of failure:if the server fails, the system is blocked
- Performance bottleneck: the server is the bottleneck
- Copyright infringement: the server can the liable

file transfer is decentralized, but locating content is highly centralized

P2P completely distributed: Gnutella

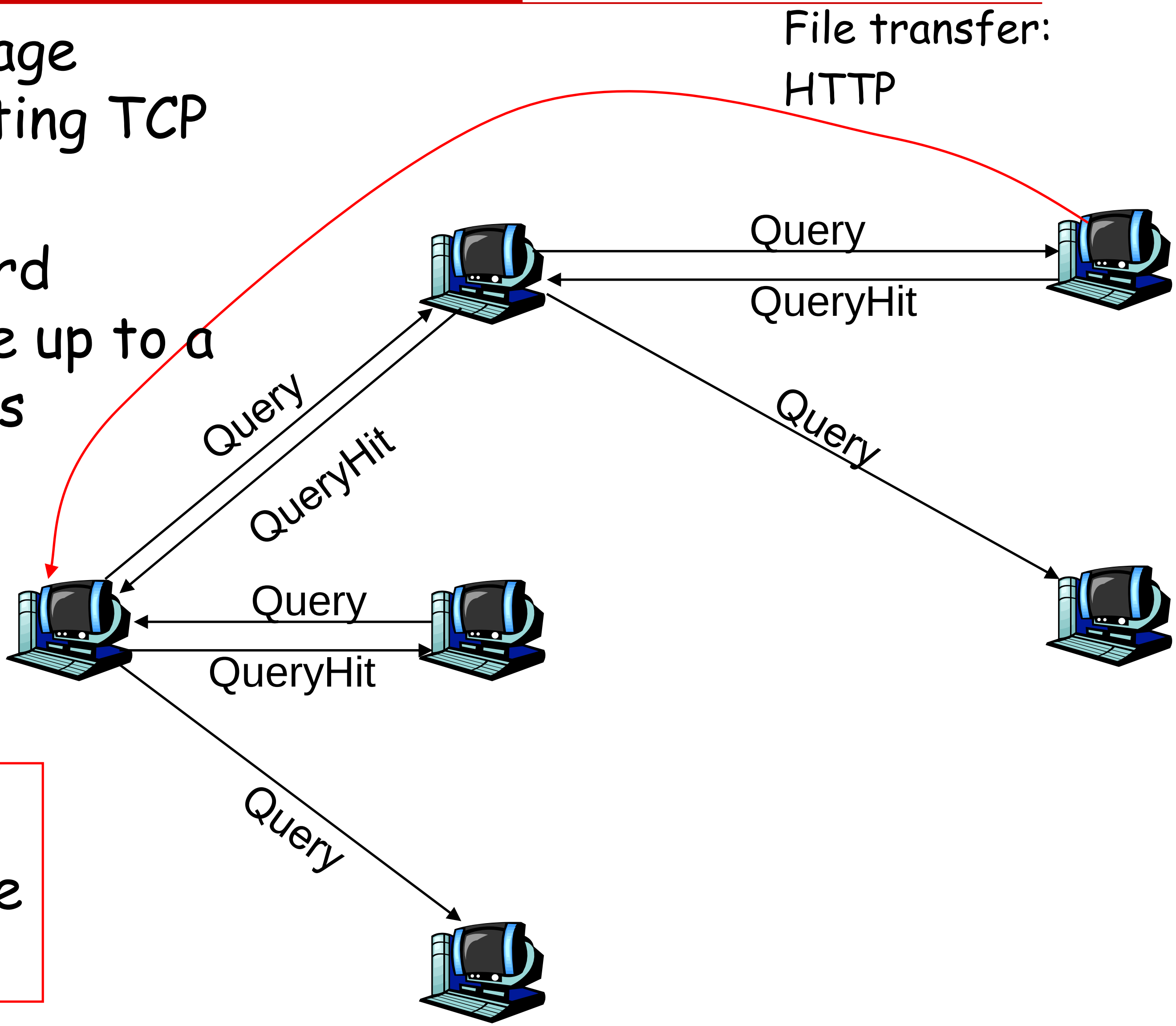
- fully distributed
  - no central server
- public domain protocol
- many Gnutella clients worldwide based on this same protocol

overlay network: graph

- edge between peer X and Y if there’s a TCP connection
- The search of neighbors is distributed in nature
- all active peers and edges are overlay net
- Edge is not a physical link
- Given peer will typically be connected with < 10 overlay neighbors

Gnutella: protocol

- Query message sent over existing TCP connections
- peers forward Query message up to a given # of hops
- QueryHit sent over the reverse path



Scalability: limited scope flooding

Gnutella: Peer joining

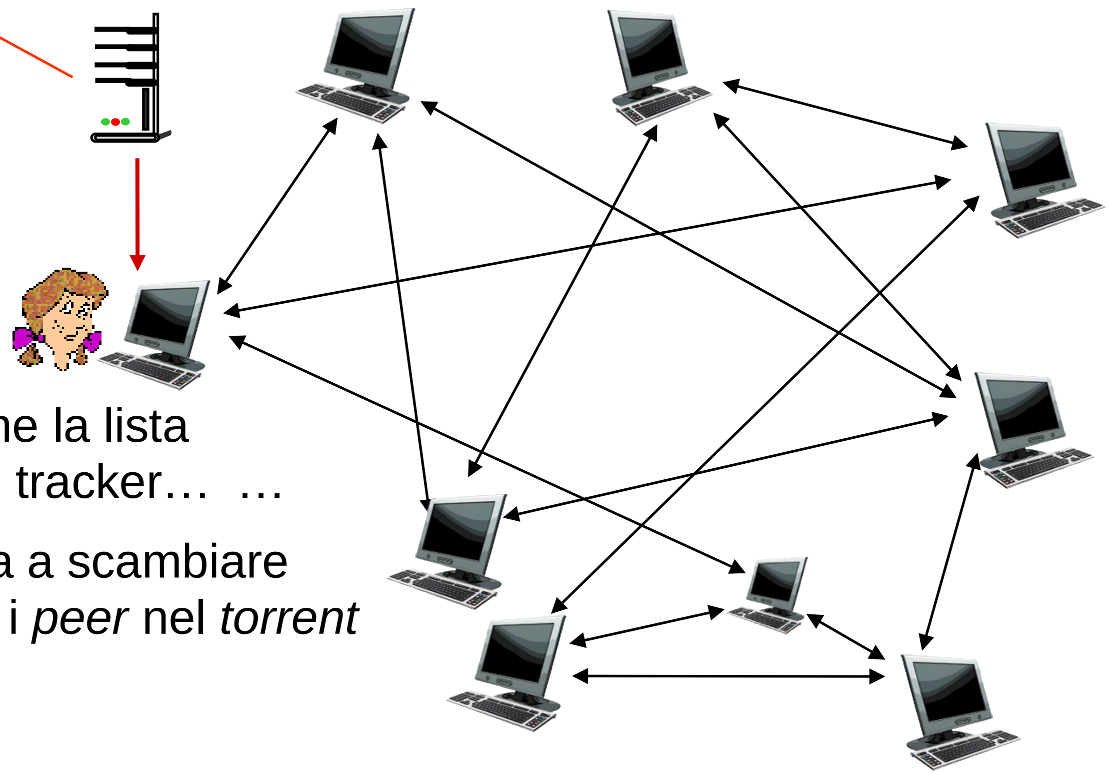
1. Joining peer X must find some other peer in Gnutella network: to use list of candidate peers
2. X sequentially attempts to make TCP with peers on list until connection setup with Y
3. X sends Ping message to Y; Y forwards Ping message.
4. All peers receiving Ping message respond with Pong message
5. X receives many Pong messages. It can then setup additional TCP connections

BitTorrent

- Files are divided in *chunks* of 256 kbytes

tracker: tiene traccia dei peer che partecipano ad un torrent

torrent: gruppo di peer che si scambiano chunk di un file

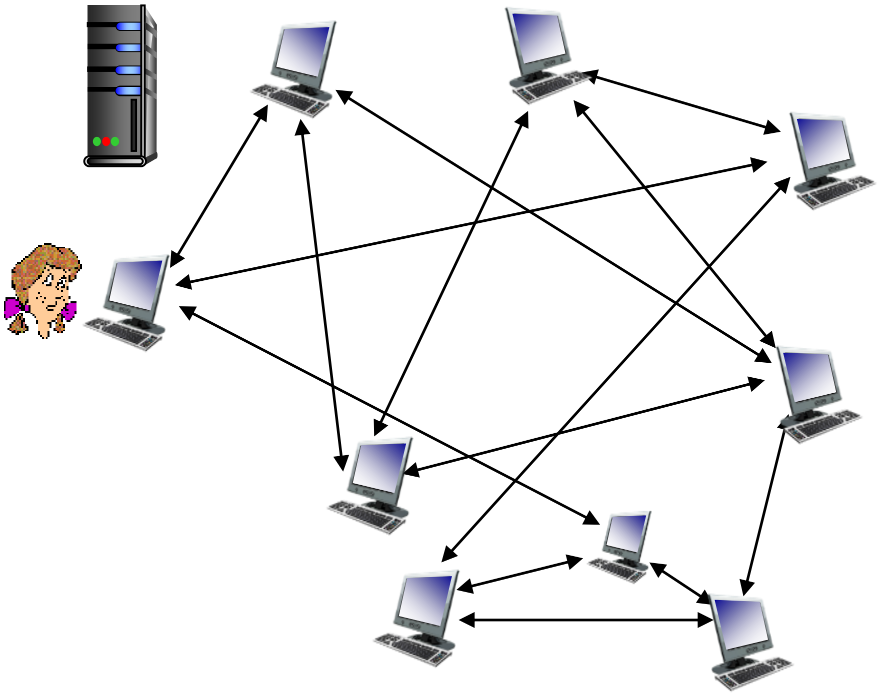


Alice ottiene la lista di peer dal tracker… …

… ed inizia a scambiare *chunk* con i *peer* nel *torrent*

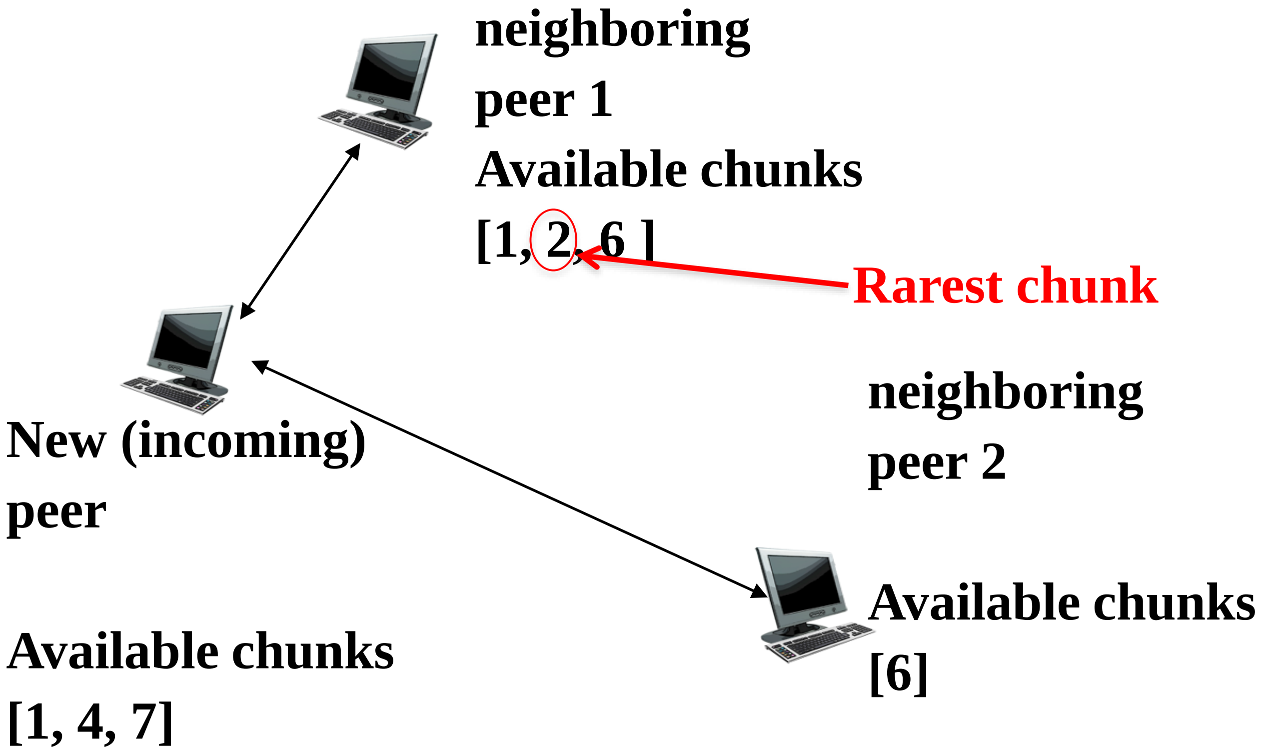
BitTorrent – join the *torrent*

- The *peers* that enter in a *torrent* register on a *tracker* to obtain a list of «active» peers
- The *tracker* sends a list of active peers on a *torrent* (IP addresses)
- The new *peer* establishes TCP connections only with a subset of *peers* in the list (*neighboring peers*)
- N*eighboring peers* send to the new *peer* the list of available chunks
- The new *peer* chooses which *chunk* to download and from which *peer* based on heuristic mechanisms



Chunk request mechanism

- Principle of *Rarest First*
  - The incoming *peer*, among all missing chunks, downloads first the rares chunks in the list of chunks sent by all *neighboring peer*



Sending chunk mechanism

- The new *peer* answers to requests that come from the *x peers* that send chunks at the maximum *rate*
- All the other peers are *choked*
- The best *x peers* are re-determined periodically (10[s])

- Every 30[s] a new peer is chosen randomly to send a chunk to (*optimistic unchoking*)