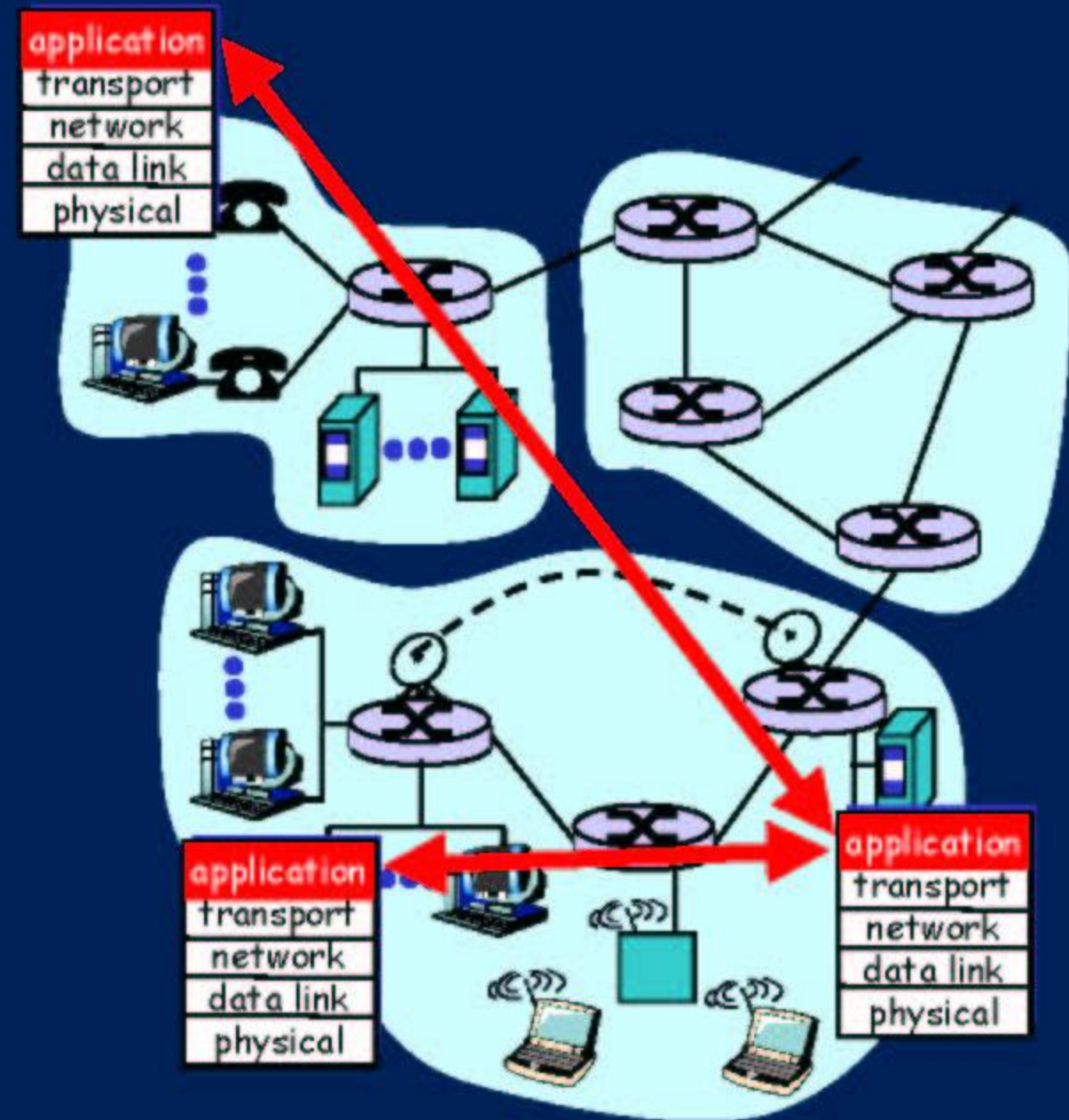


# Applications and app-layer protocols

Application: communicating, distributed processes

- running in network hosts in “user space”
- exchange messages to implement app
- e.g., email, file transfer



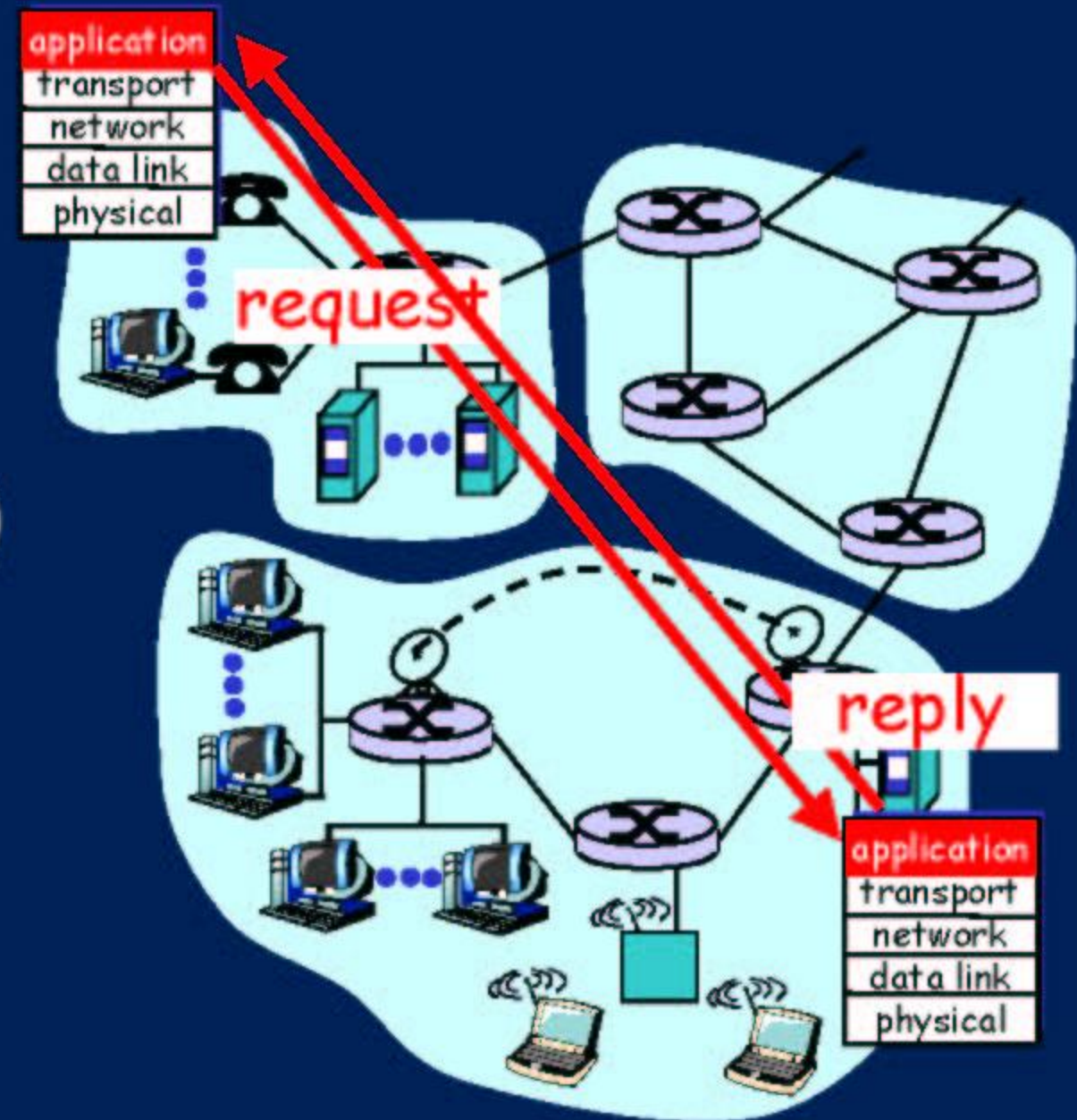


# Client-server paradigm

Typical network app has two pieces: *client* and *server*

## Client:

- initiates contact with server (“speaks first”)
- typically **requests** service from server,
- for Web, client is implemented in browser; for e-mail, in mail reader



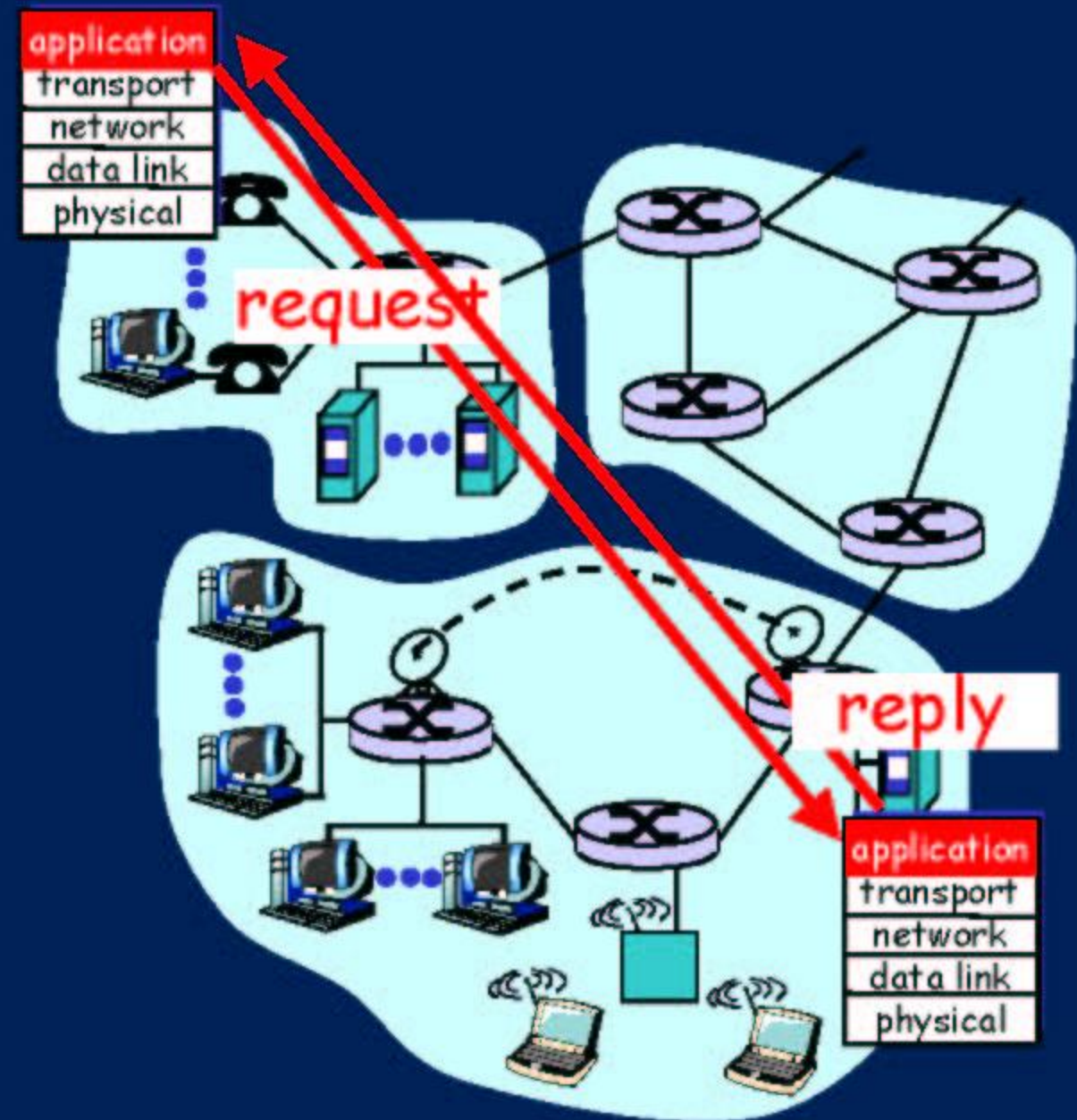


# Client-server paradigm

Typical network app has two pieces: *client* and *server*

## Server:

- provides requested service to client, via **replies**
- e.g., Web server sends requested Web page, mail server delivers e-mail





# Services provided by Internet transport protocols

## TCP service:

- ***connection-oriented***: setup required between client, server
- ***reliable transport*** between sending and receiving process
- ***flow control***
- ***congestion control***

## UDP service:

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

# Internet apps: their protocols and transport protocols

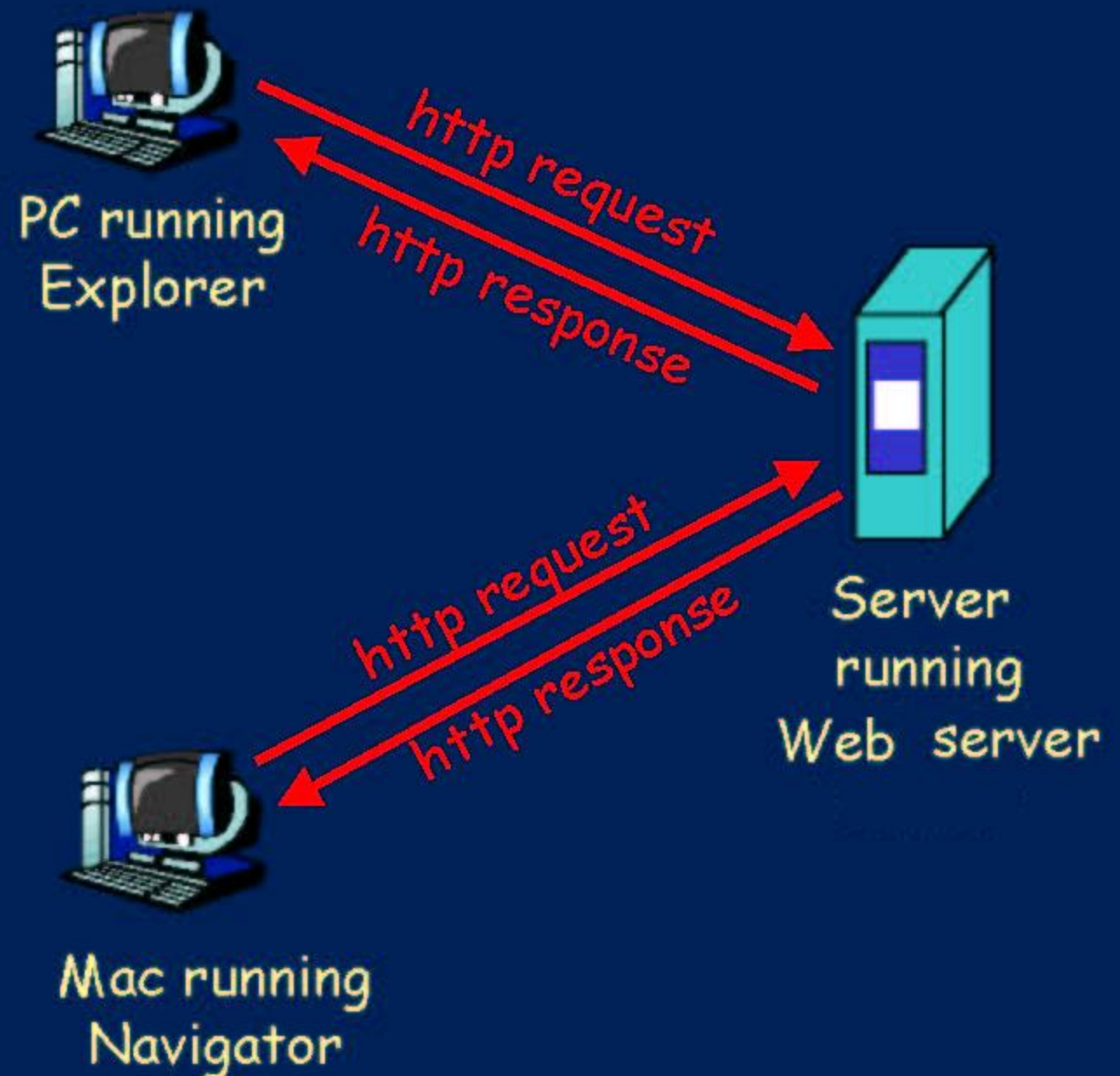
Application	Application layer protocol	Underlying transport protocol
e-mail	smtp [RFC 821]	TCP
remote terminal access	telnet [RFC 854]	TCP
Web	http [RFC 2068]	TCP
file transfer	ftp [RFC 959]	TCP
streaming multimedia	proprietary	TCP or UDP



# The Web: the *http* protocol

## http: hypertext transfer protocol

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





# The http protocol: more

## http: TCP transport service:

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



# http example

Suppose user enters URL `www.School.edu/Department/home.index`

(contains text, references to 10 jpeg images)

**1a.** http client *initiates* TCP connection to http server (process) at `www.School.edu`. Port 80 is default for http server.

**1b.** http server at host `www.School.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


**3.** http server receives request message, forms *response message* containing requested object (`Department/home.index`), sends message into socket

time





# http example (cont.)

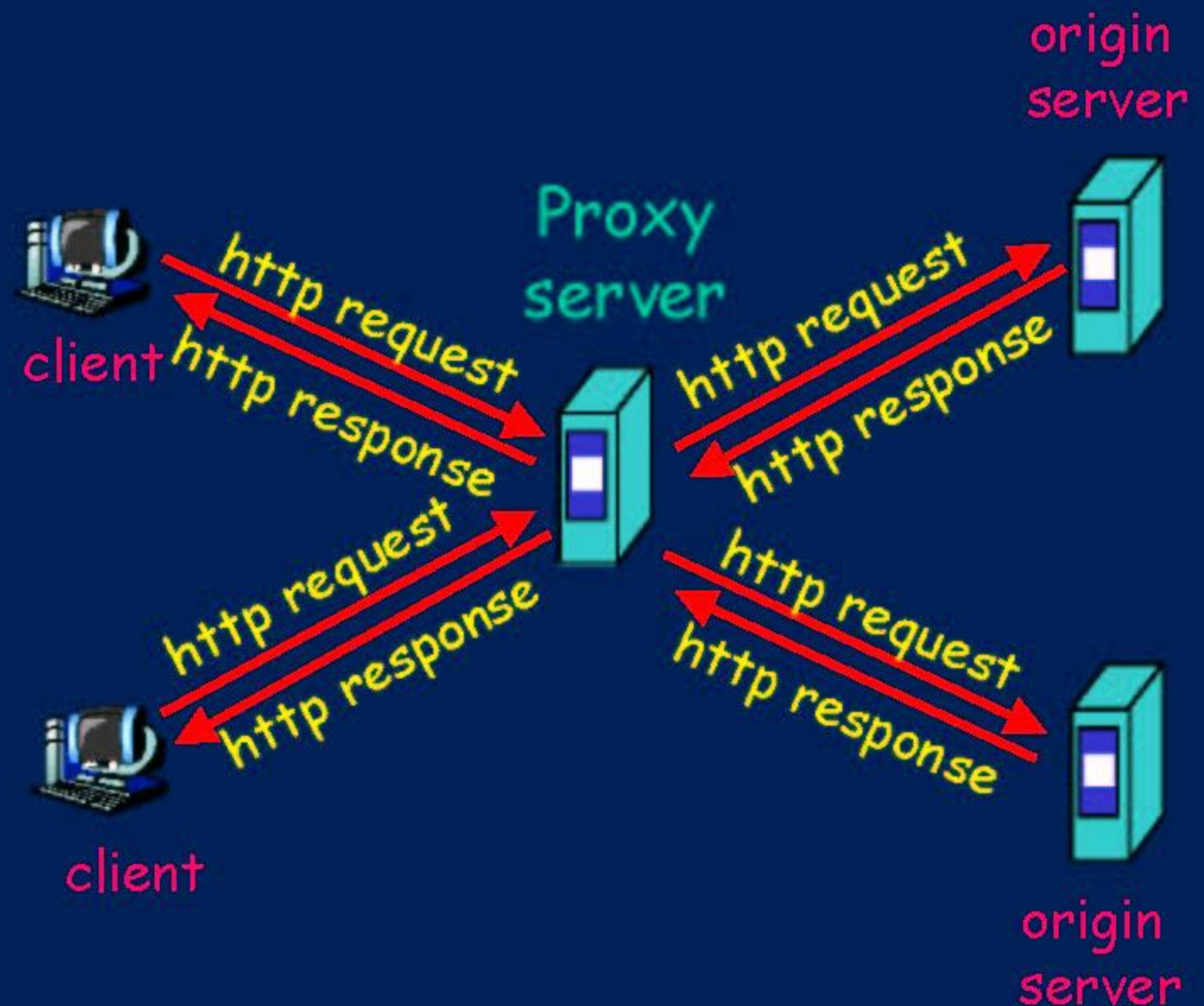
- 
- 5.** http client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects
- 4.** http server closes TCP connection.
- 6.** Steps 1-5 repeated for each of 10 jpeg objects
- time**



# Web Caches (proxy server)

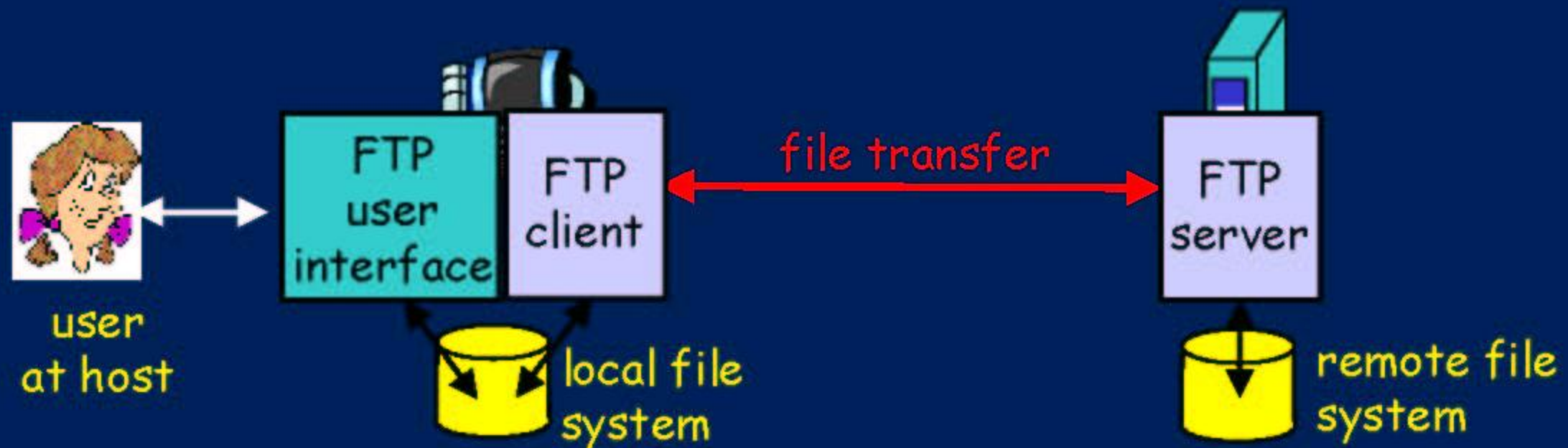
**Goal:** satisfy client request without involving origin server

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





# 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



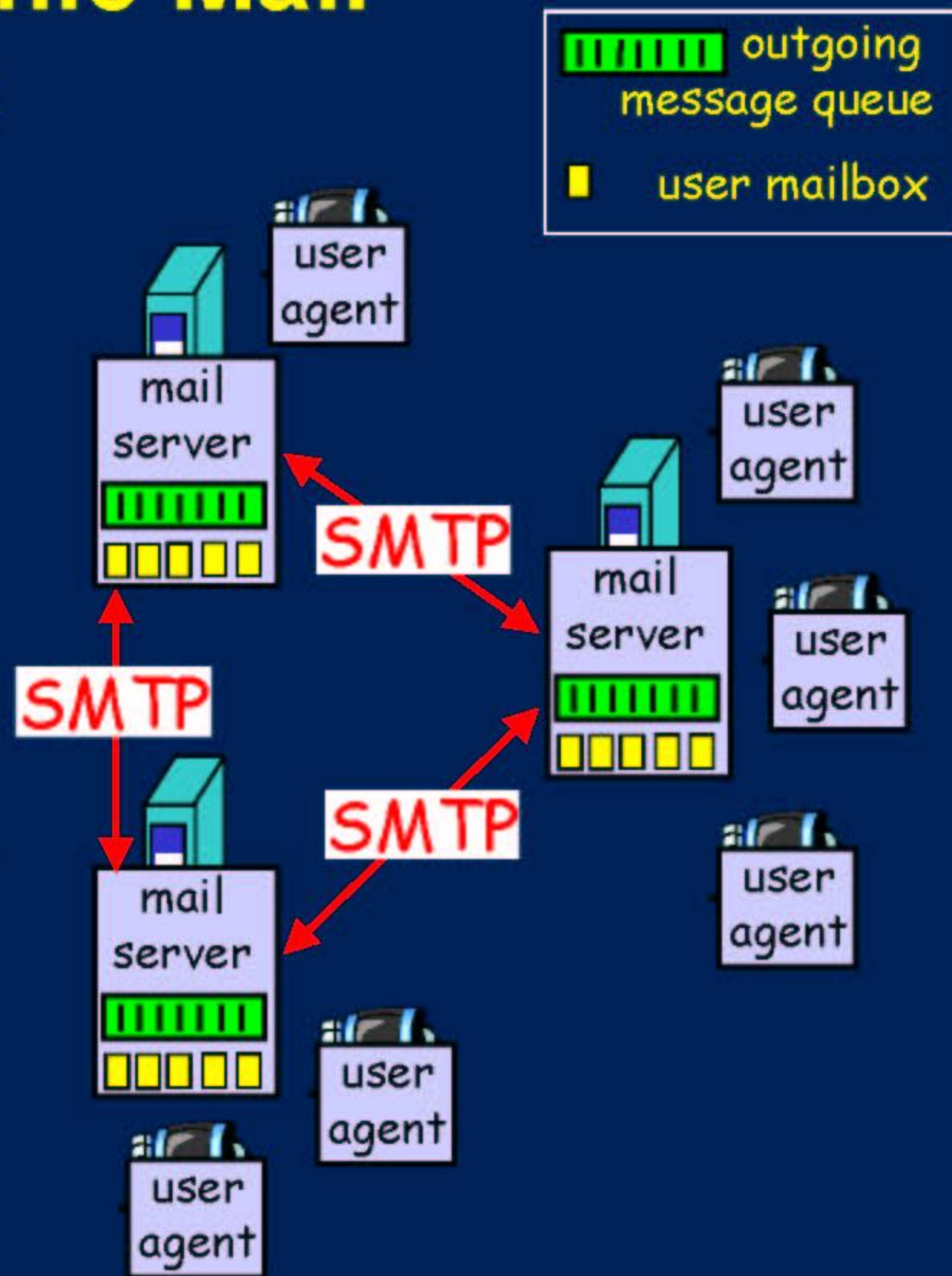
# Electronic Mail

## Three major components:

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

## User Agent

- mail reader
- composing, editing, reading mail messages
- e.g., Eudora, Outlook, elm, Netscape Messenger
- outgoing, incoming messages stored on server



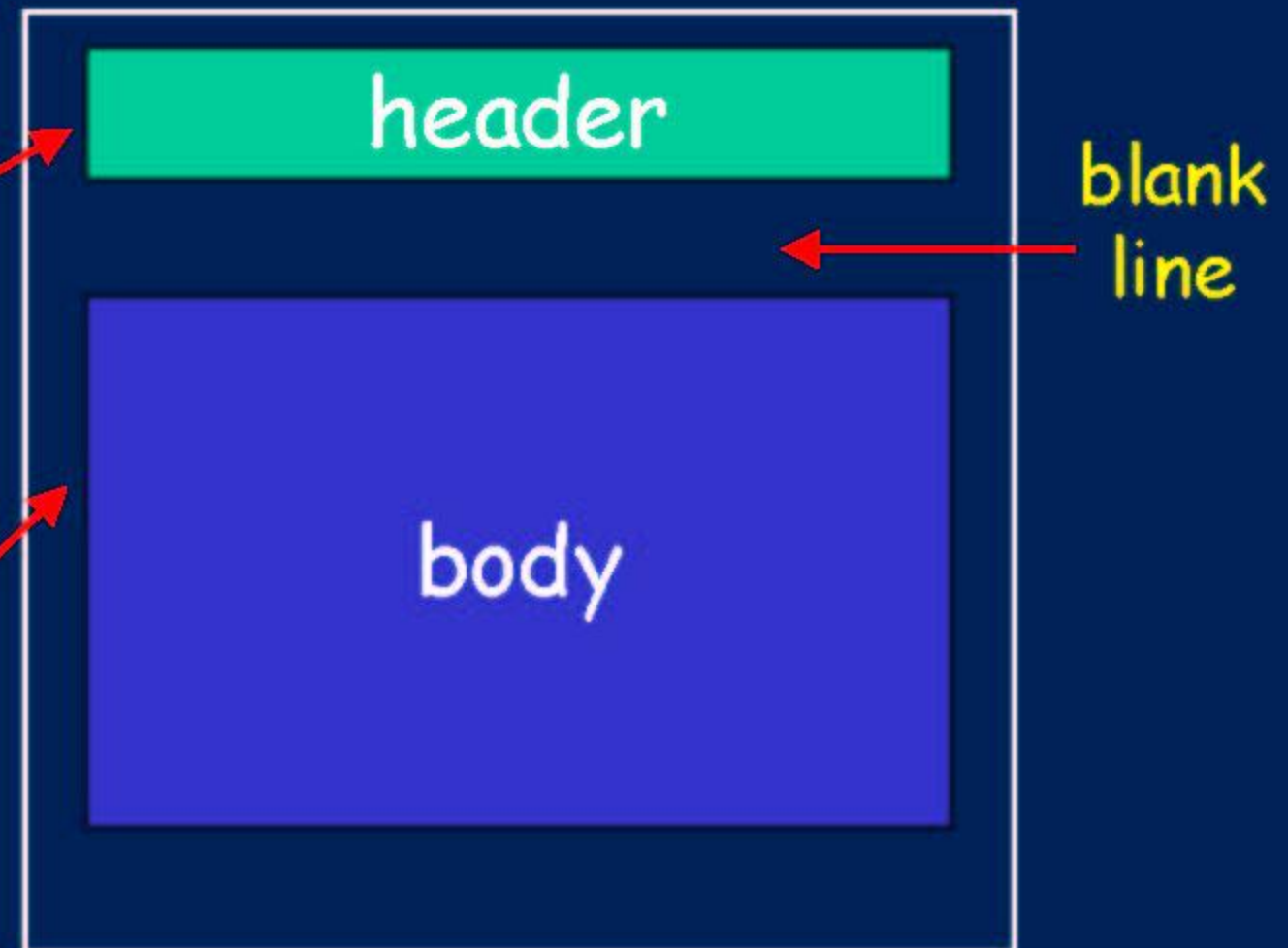


# Mail message format

smtp: protocol for exchanging email msgs

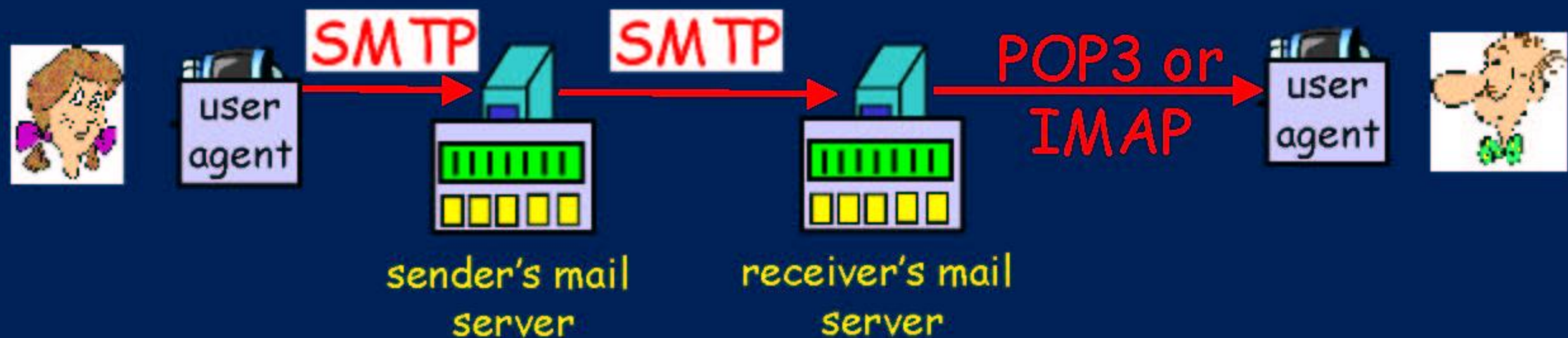
RFC 822: standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:*different from smtp commands!*
- body
  - the “message”, ASCII characters only





# Mail access protocols



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



# DNS: Domain Name System

## 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)

## Internet hosts, routers:

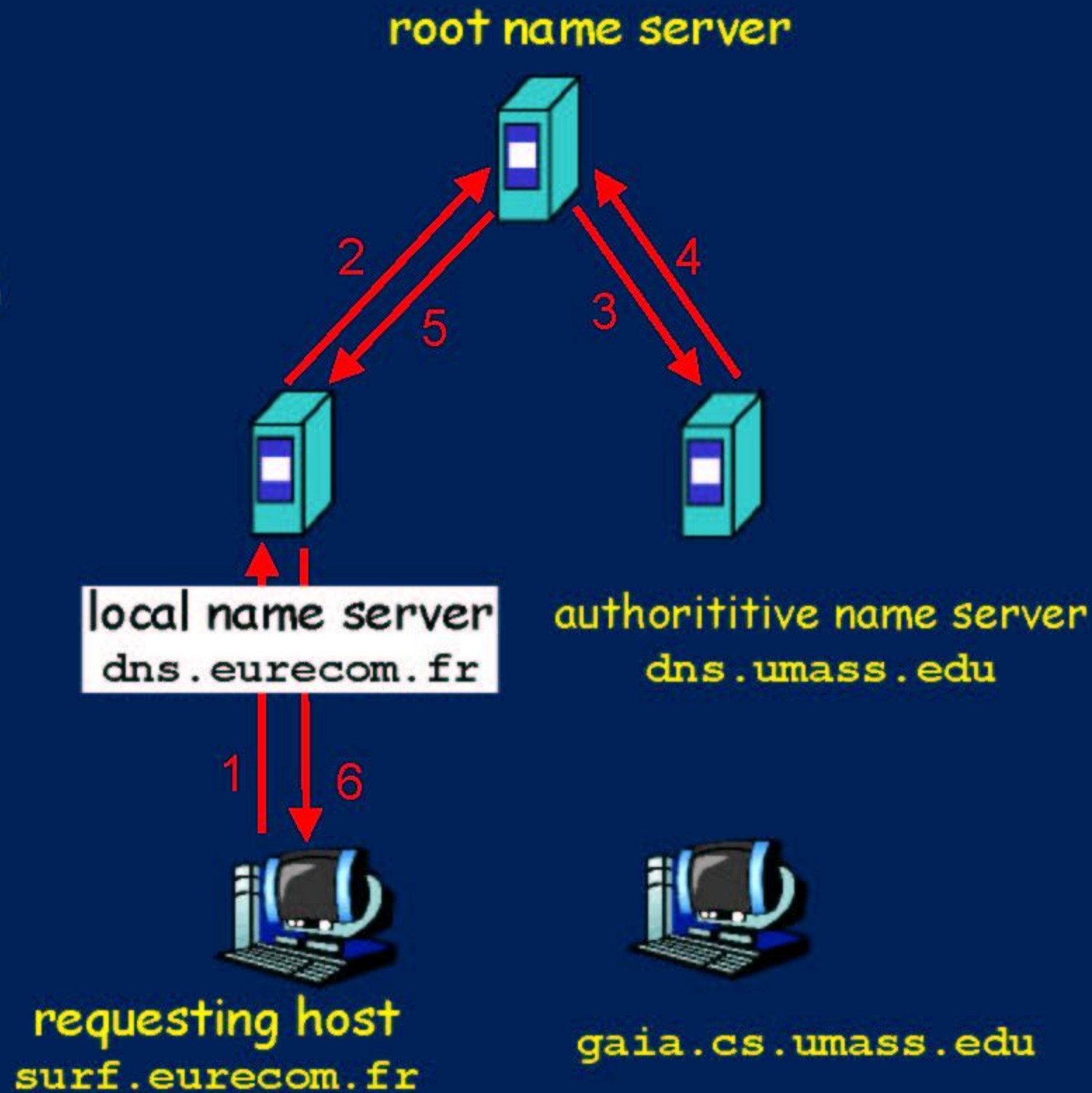
- IP address (32 bit) - used for addressing datagrams
- “name”, e.g., gaia.cs.umass.edu - used by humans



# Simple DNS example

host `surf.eurecom.fr`  
wants IP address of  
`gaia.cs.umass.edu`

1. **Contacts its local DNS server,**  
`dns.eurecom.fr`
2. `dns.eurecom.fr`  
**contacts root name server,** if necessary
3. **root name server**  
**contacts authoritative name server,**  
`dns.umass.edu`, if necessary





# DNS example

## Root name server:

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

