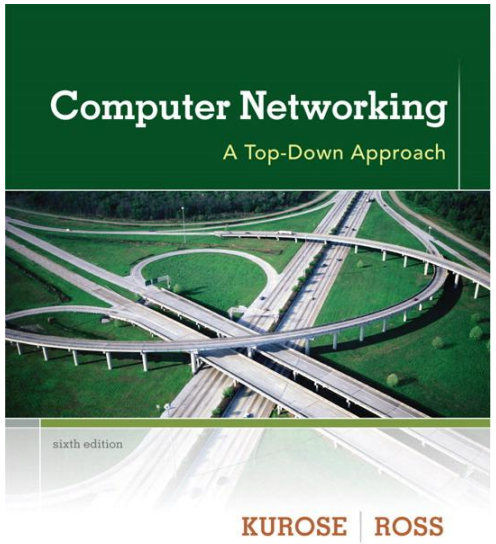
Chapter 2
Application Layer



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

- SMTP, POP3, IMAP

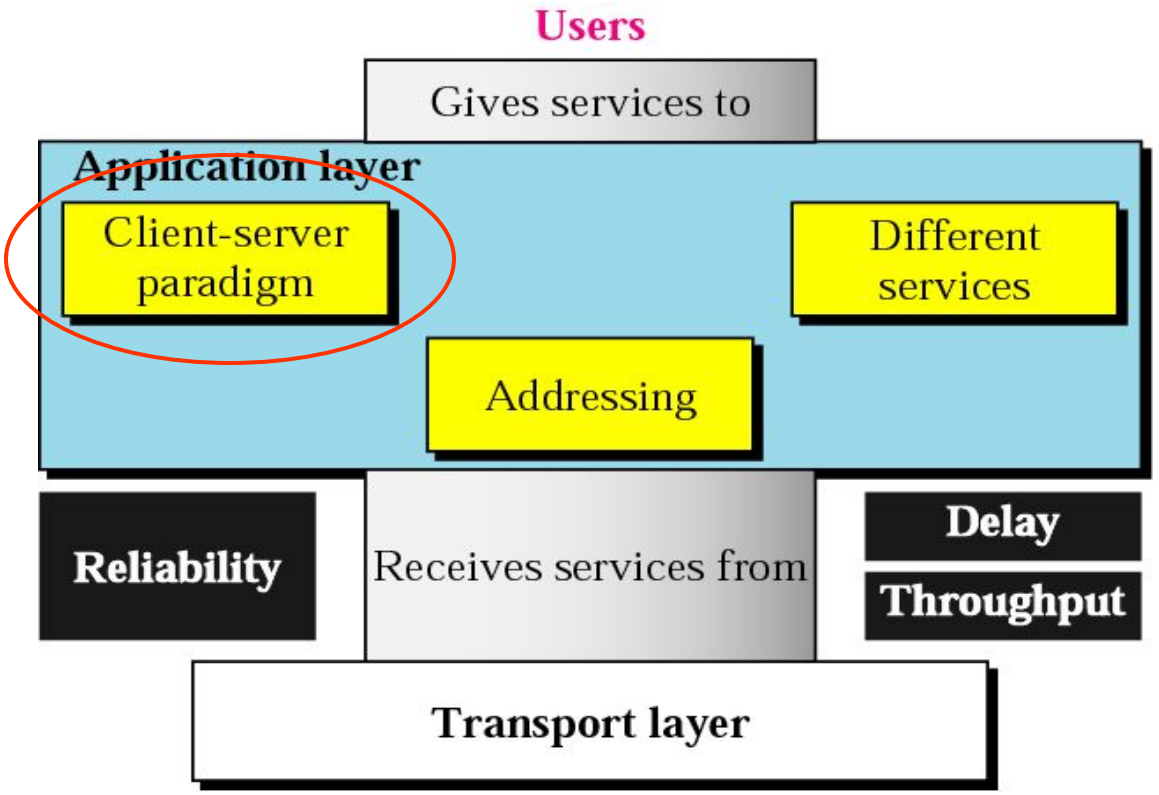
2.5 DNS

2.6 P2P applications

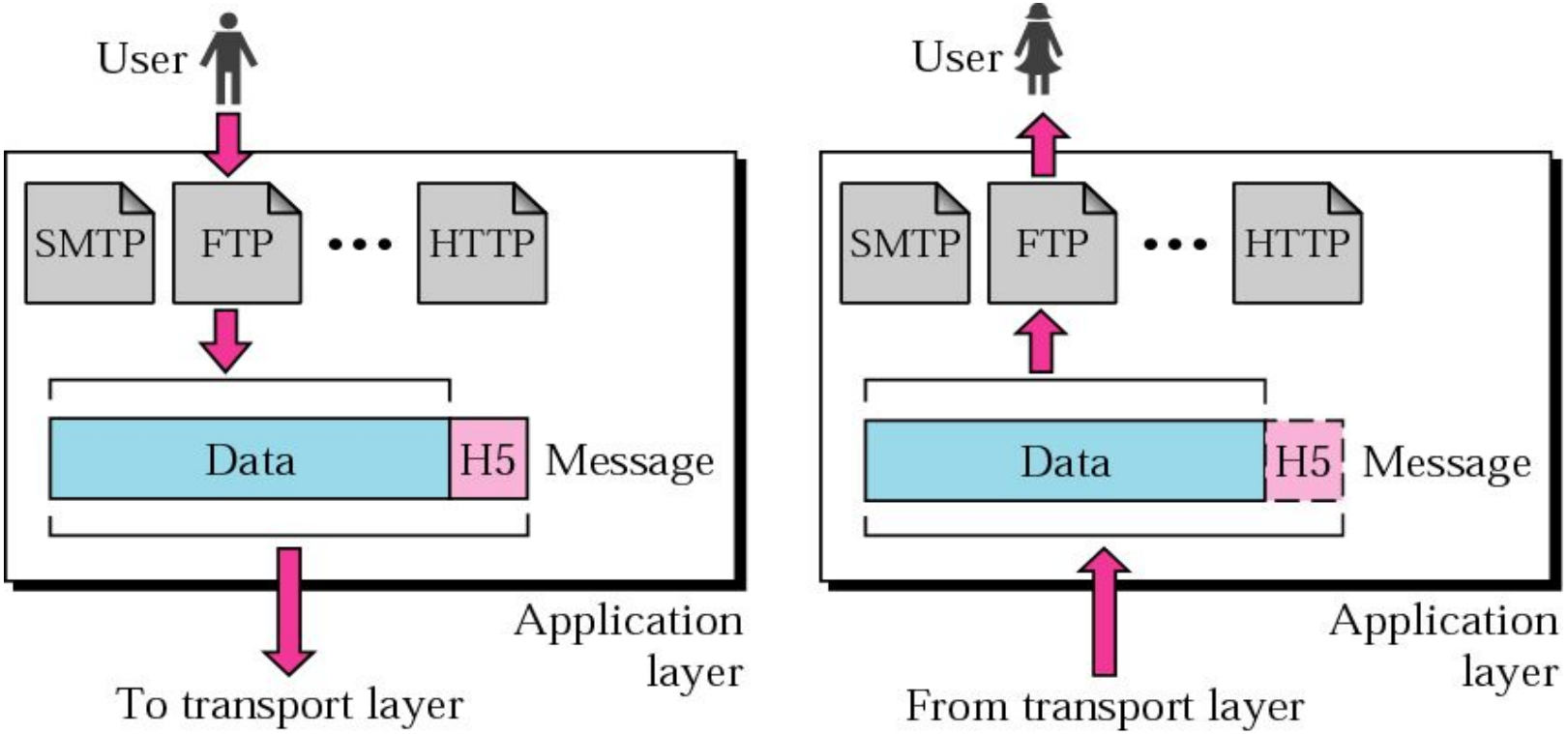
2.7 socket programming with UDP and TCP

Application layer

- This layer allows people to use the Internet
- Other 4 layers are just made so people can use application programs



Application layer

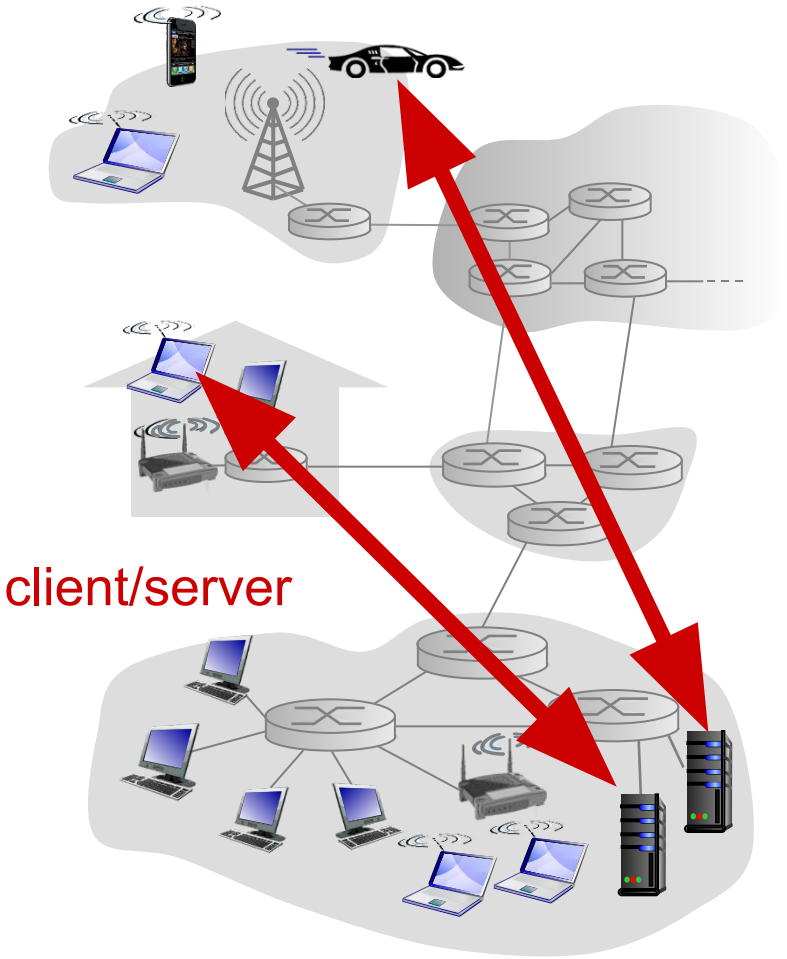


Application architectures

possible structure of applications:

- client-server
- peer-to-peer (P2P)

Client-server architecture



server:

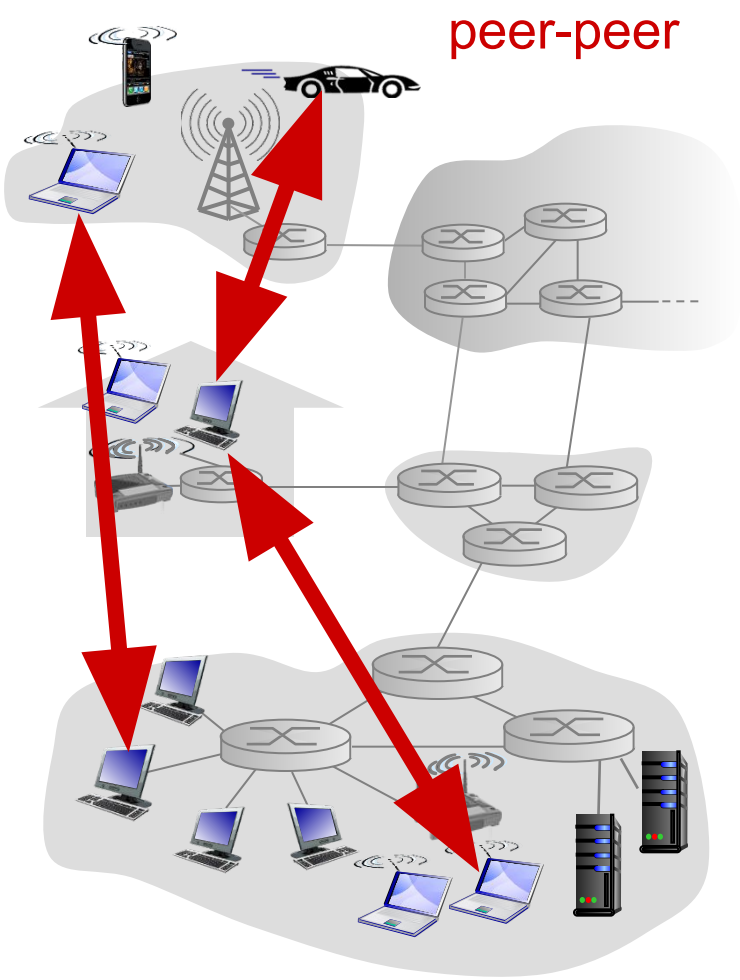
- always-on host
- permanent IP address
- data centers for scaling

clients:

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

P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management



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

clients, servers

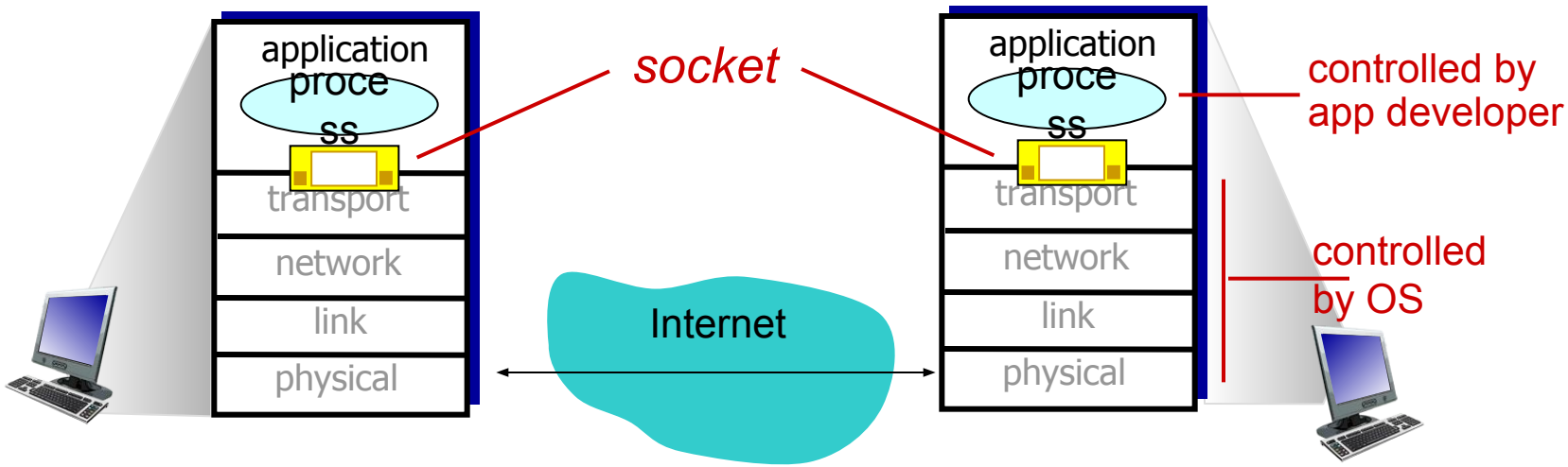
client process: process that initiates communication

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 to deliver message to socket at receiving 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…

Network applications vs. Application-layer protocols

- Network application: communicating, distributed processes
  - Processes running in different host communicate by an application-layer protocol
    - SMTP, HTTP..

- Application-layer protocols
  - one piece of an application
  - define messages exchanged by applications and & actions taken
  - implementing services by using the service provided by lower layers

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

open protocols:

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

proprietary protocols:

- e.g., Skype

What transport service does an app need?

data integrity

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

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: 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
text messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP service:

- *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 guarantee, security
- *connection-oriented:* setup required between client and server processes

UDP service:

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

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)	TCP or UDP

Chapter 2: outline

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

2.6 P2P applications

2.7 socket programming with UDP and TCP

WWW

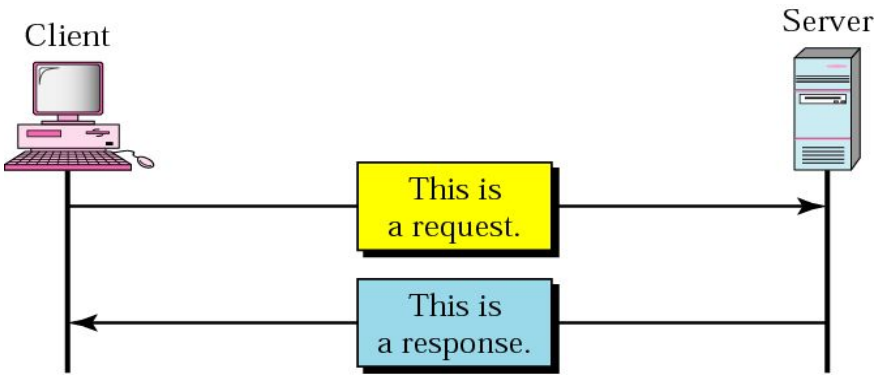
- World Wide Web
- Introduced in 1991
- originated from the CERN High-Energy Physics laboratories in Geneva, Switzerland.

The Web: Some Jargon

- Web page
  - consists of objects (HTML file, JPEG image, GIF image…)
  - addressed by URL
- Most Web pages consist of
  - base HTML page
  - several referenced objects
- URL
  - A standard way of specifying the location of an object, typically a web page, on the Internet
- User agent for Web is called a browser
  - MS Internet Explorer
- Server for Web is called a Web server

HyperText Transfer Protocol

- Web's application layer protocol
  - Used to access data on the World Wide Web
  - Rapid jump from one document to another
- Client-server model
  - client: browser that requests, receives, "displays" web objects
  - server: Web server sends objects in response to request
- uses one TCP connection on the well-known port 80



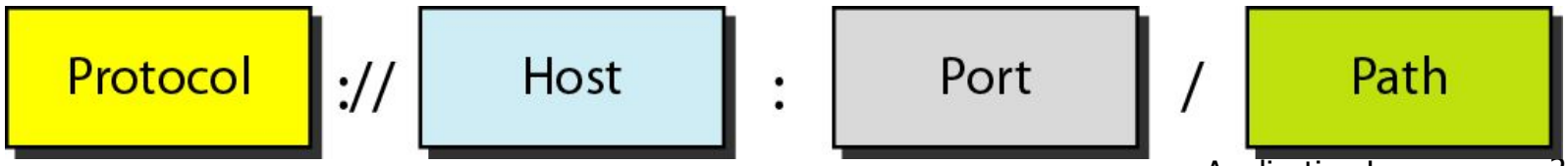
Web and HTTP

First, a review…

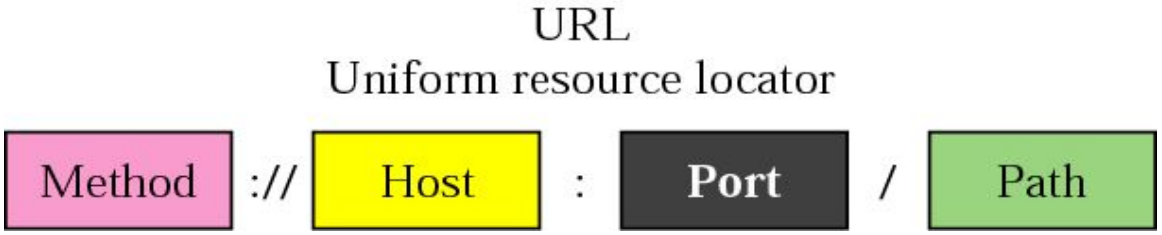
- *web page* consists of *objects*
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects*
- each object is addressable by a *URL,* e.g.,

`www.someschool.edu/someDept/pic.gif`

host name | path name



URL - continued



- method
  - protocol used to retrieve the document (FTP, HTTP, …)
- host
  - a computer where the info is located
  - the name of the computer can be an alias (not necessary www)
- port
  - optional port # of the server
- path
  - the path name of the file where the info is located

Figure 27.1 *Architecture of WWW*

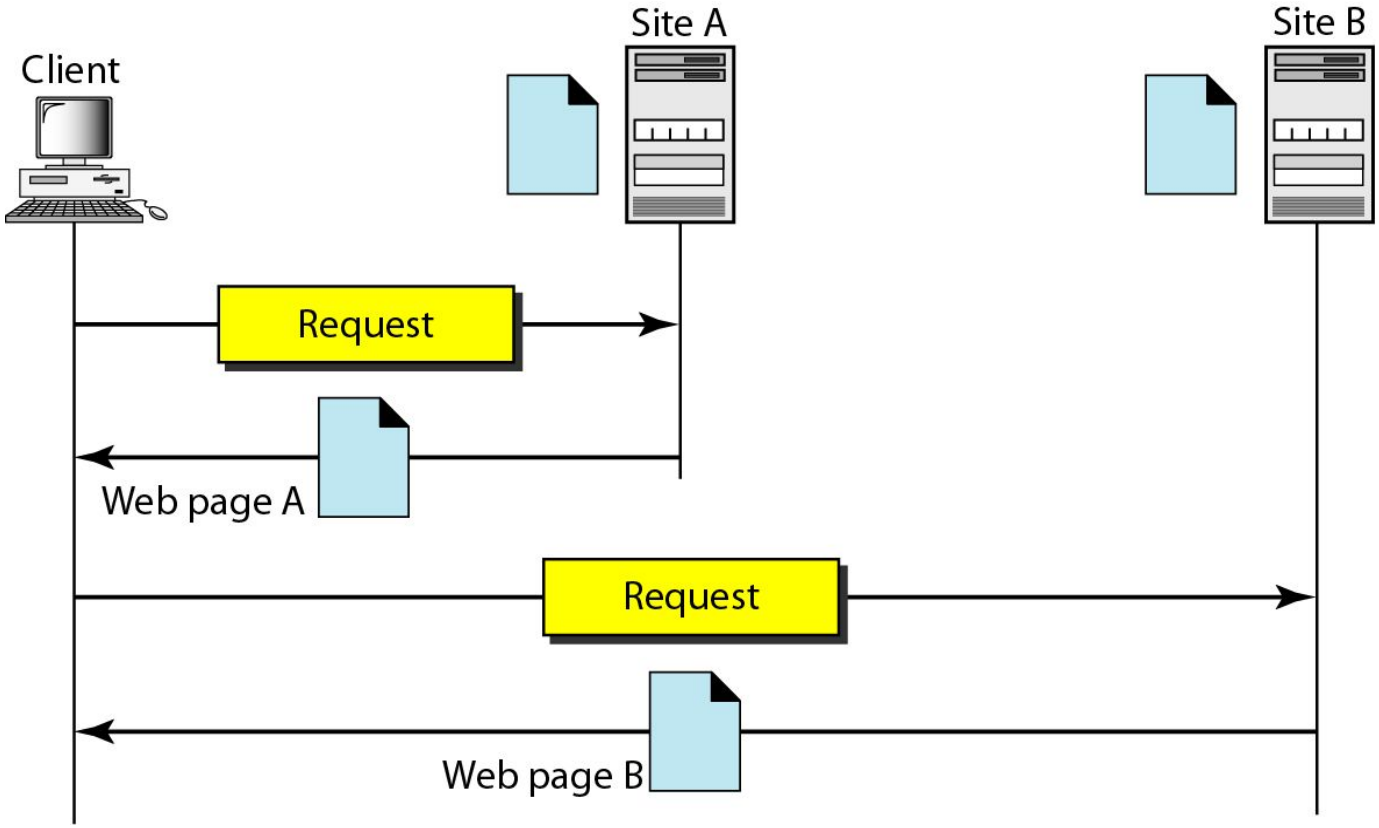
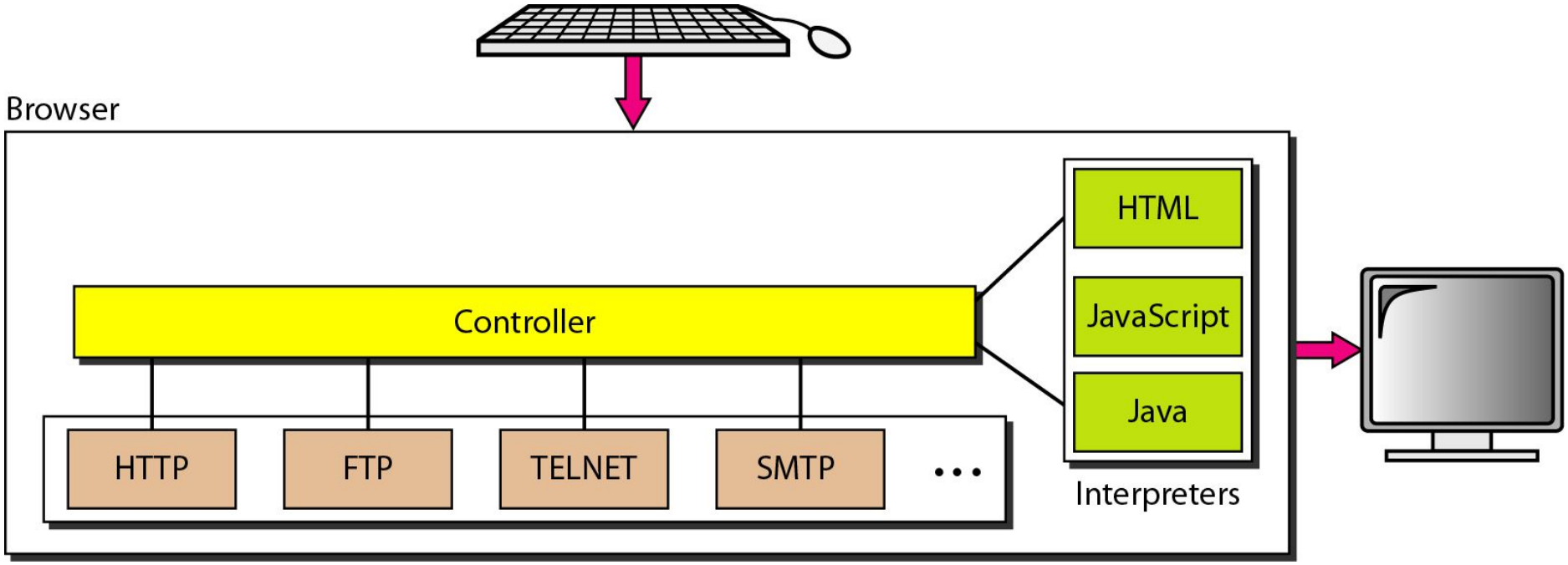


Figure 27.2 *Browser*



27-2 WEB DOCUMENTS

The documents in the WWW can be grouped into three broad categories: static, dynamic, and active. The category is based on the time at which the contents of the document are determined.

Topics discussed in this section:

Static Documents

Dynamic Documents

Active Documents

Figure 27.4 *Static document*

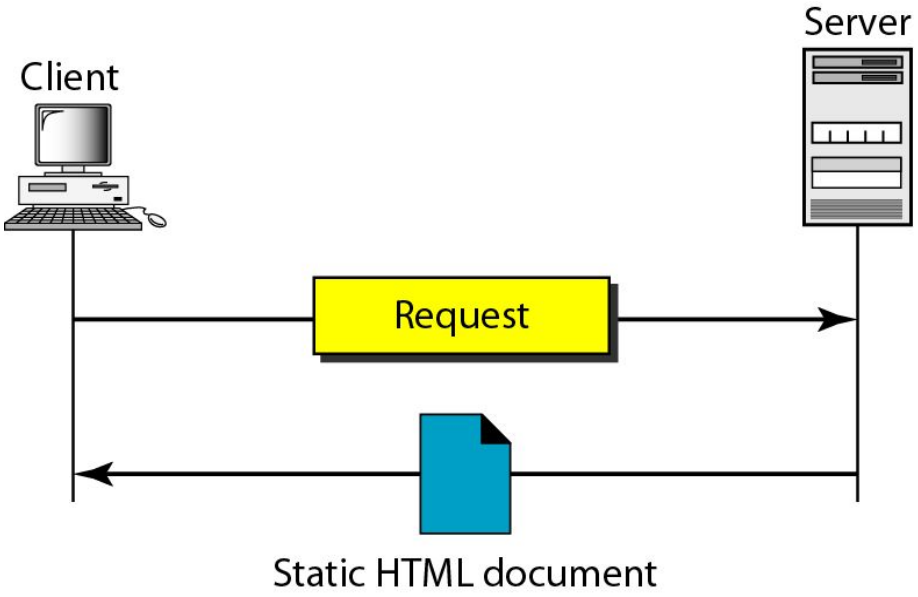


Figure 27.8 *Dynamic document using CGI*

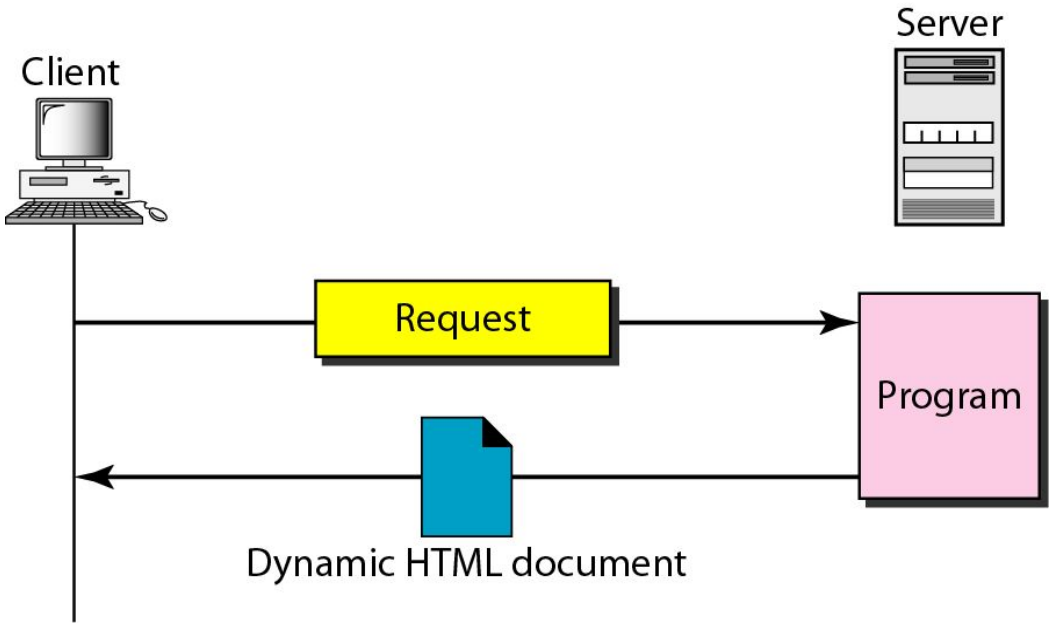


Figure 27.9 *Dynamic document using server-site script*

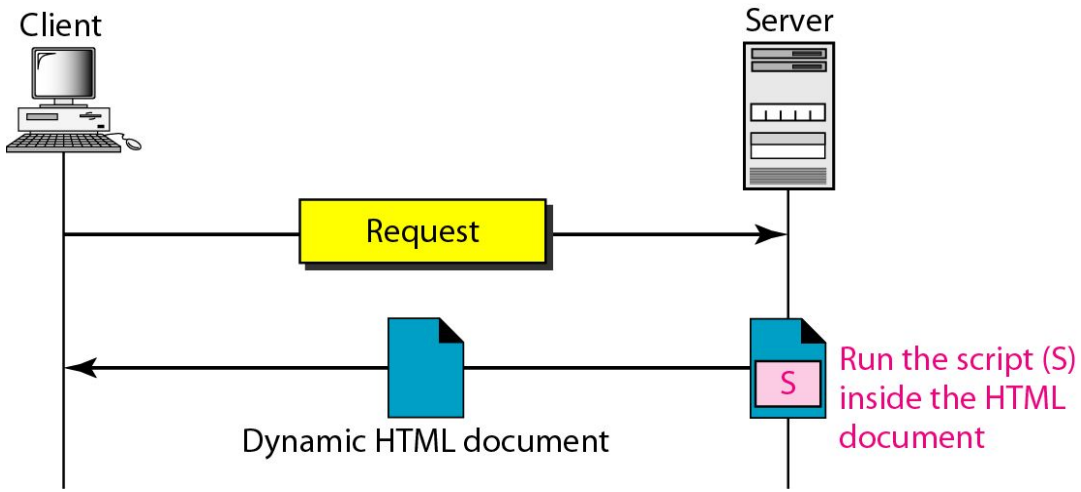


Figure 27.10 *Active document using Java applet*

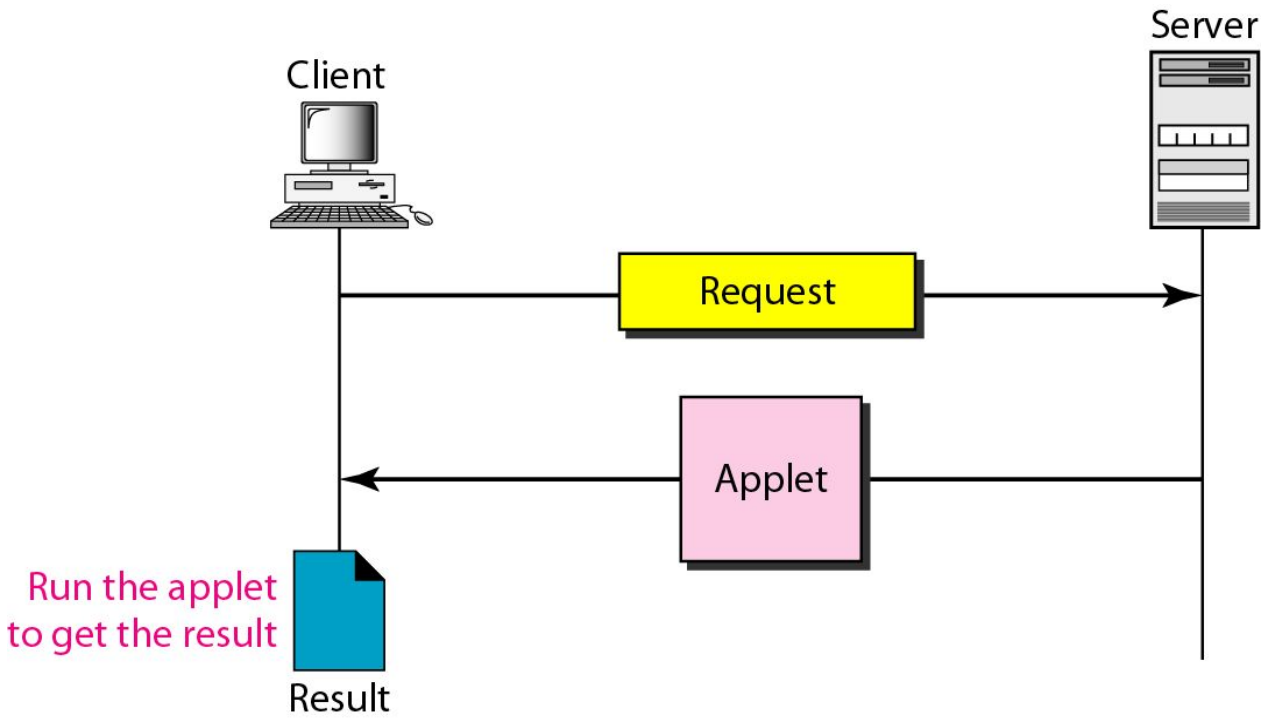
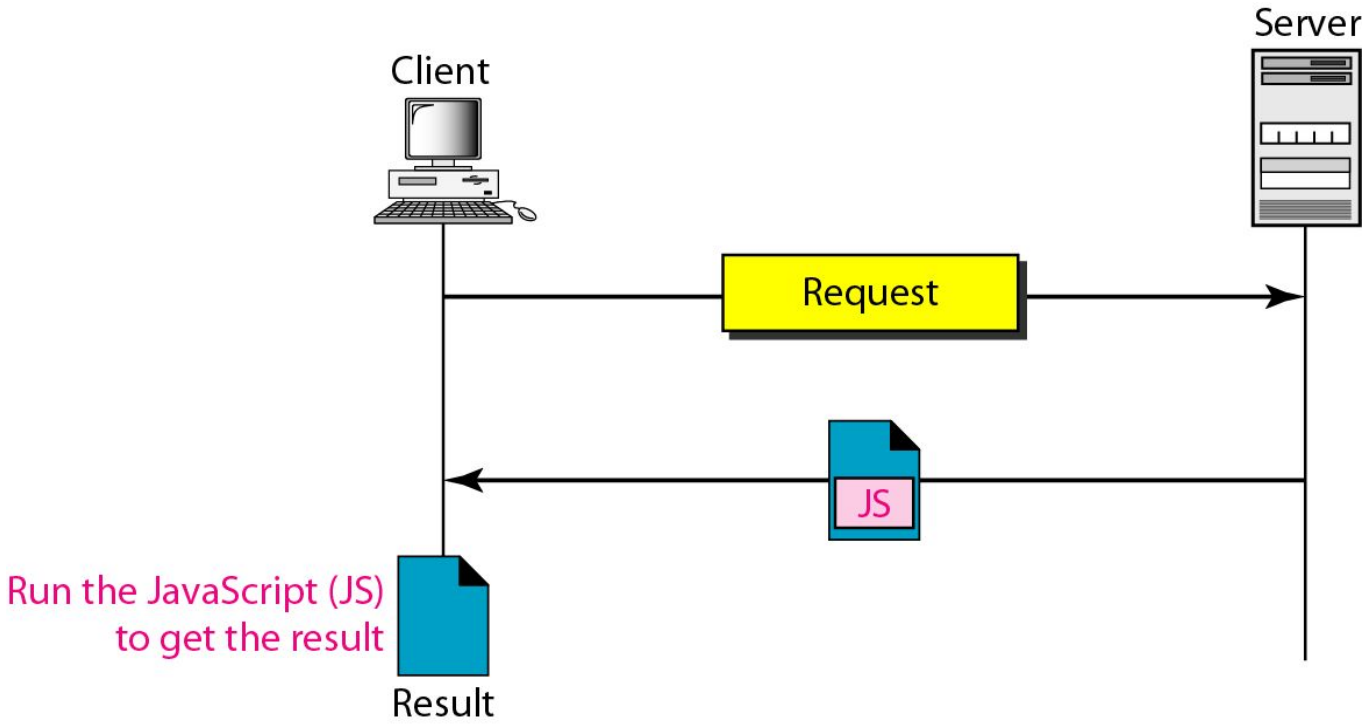


Figure 27.11 *Active document using client-site script*



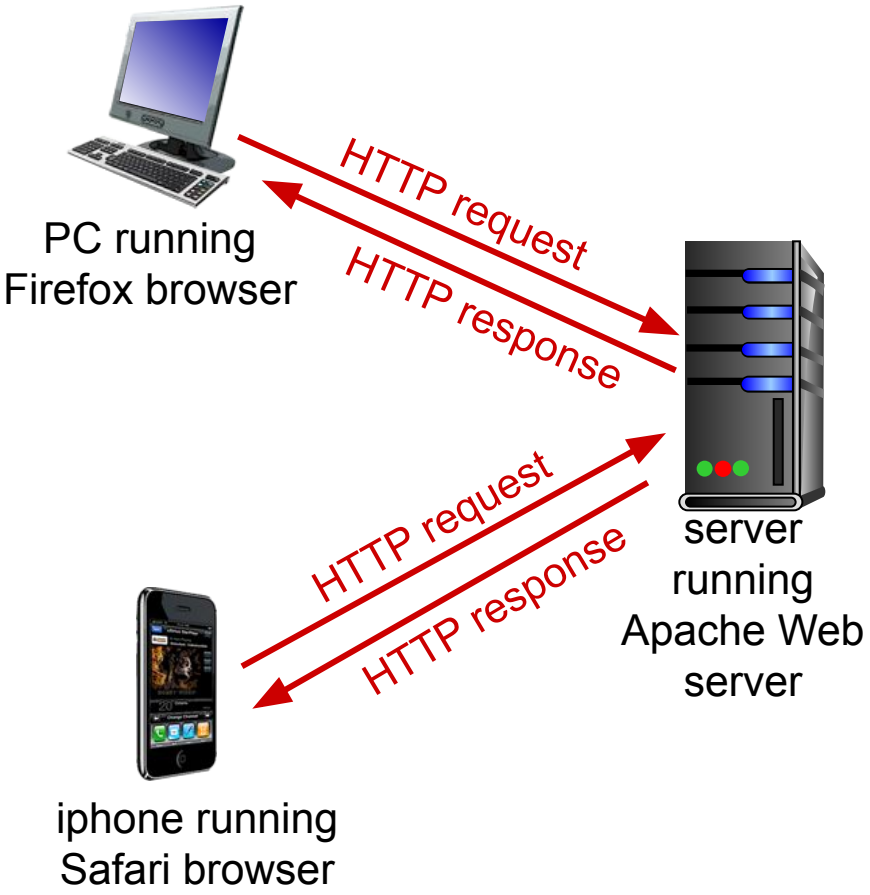
Note

Active documents are sometimes referred to as client-site dynamic documents.

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) 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

asid

protocols that maintain "state" are complex! *e*

- 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
  - connection then closed
- downloading multiple objects required multiple connections

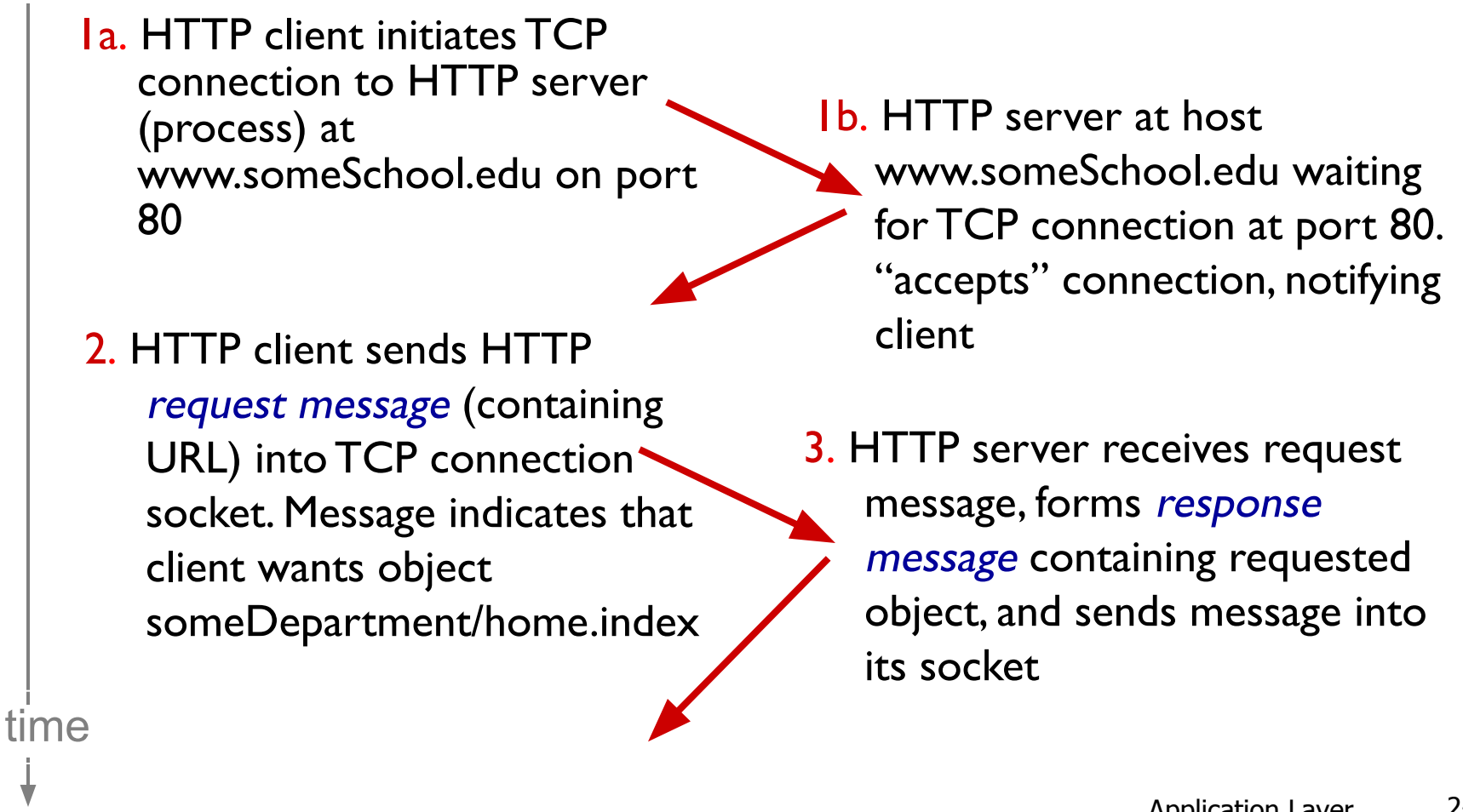
persistent HTTP

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

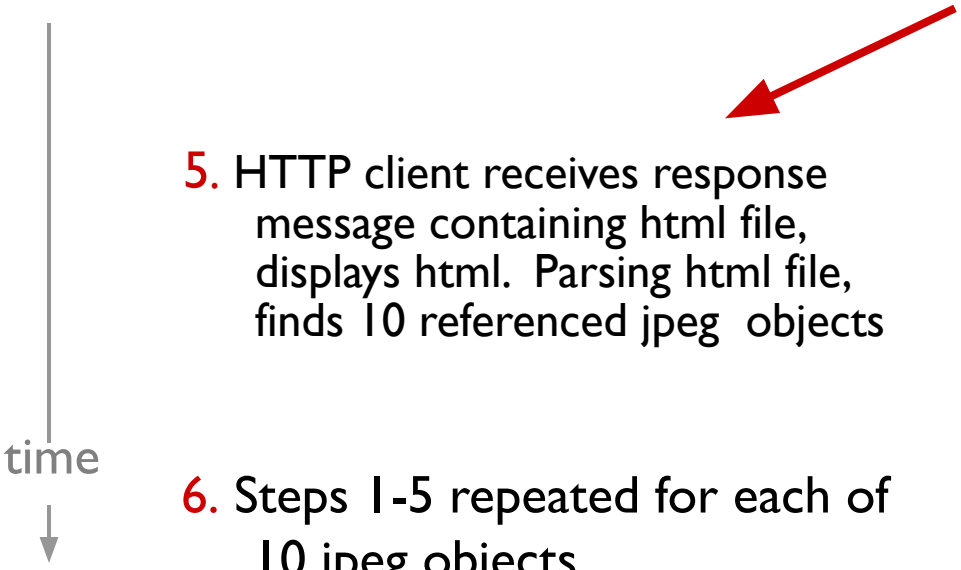
Non-persistent HTTP

suppose user enters URL:
`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)



Non-persistent HTTP (cont.)



4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

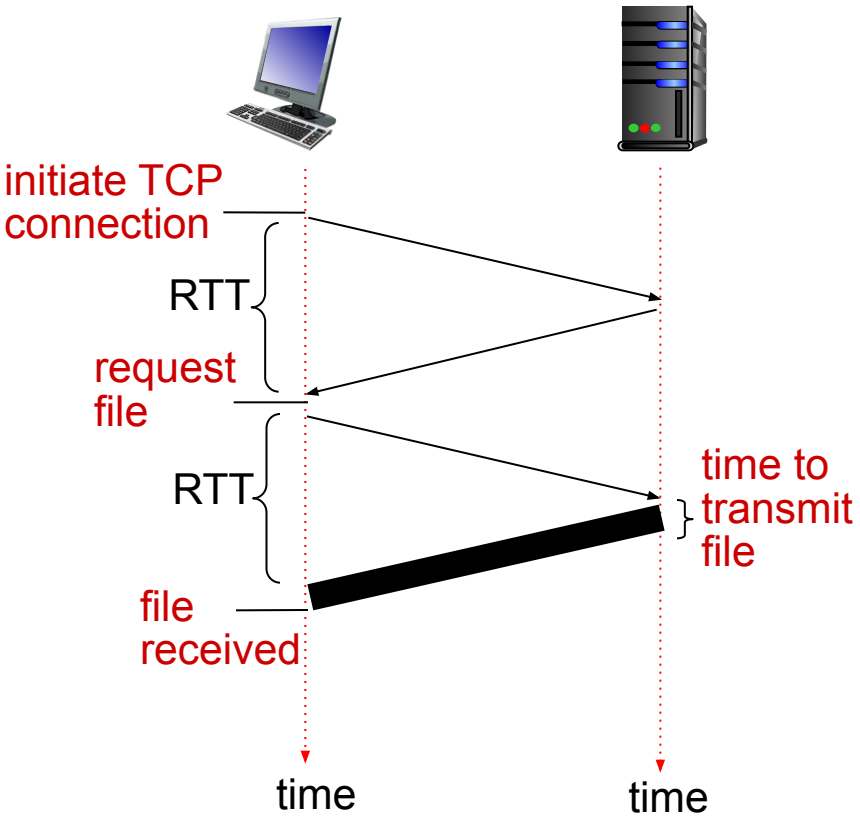
Non-persistent HTTP: response time

RTT (definition): time for a small packet to travel from client to server and back

HTTP 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
- non-persistent HTTP response time =

  2RTT+ file transmission time



Non-persistent and persistent connections

Non-persistent

- HTTP/1.0
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

But most 1.0 browsers use parallel TCP connections.

Persistent

- default for HTTP/1.1
- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- Client sends requests for all referenced objects as soon as it receives base HTML.
- Fewer RTTs and less slow start.

Pipelining

- Issue multiple requests at a time
  - Don't have to wait for previous response
  - More efficient use of link
- Use carefully
  - POST requests should not be pipelined (changes server state)
  - GET/HEAD requests are usually okay

Persistent HTTP

Non-persistent HTTP issues:

- Connection setup for each request
- But browsers often open parallel connections

Persistent HTTP:

- Server leaves connection open after sending response
- Subsequent HTTP messages between same client/server are sent over connection

Persistent without pipelining:

- Client issues new request only when previous response has been received
- One RTT for each object

Persistent with pipelining:

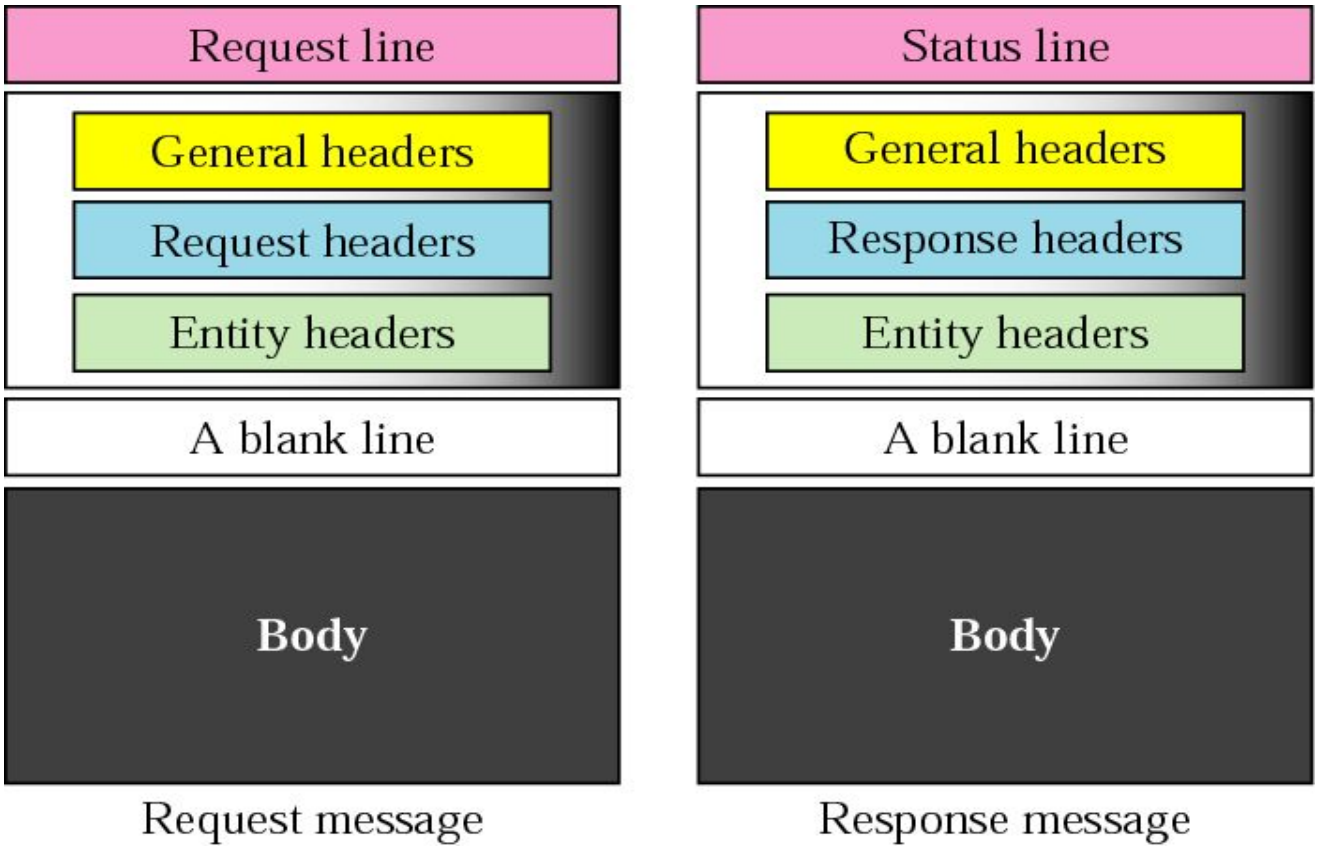
- Default in HTTP/1.1 spec
- Client sends multiple requests
- As little as one RTT for all the referenced objects
- Server must handle responses in same order as requests

"Persistent without pipelining" most common

- When does pipelining work best?
  - Small objects, equal time to serve each object
  - Small because pipelining simply removes additional 1 RTT delay to request new content

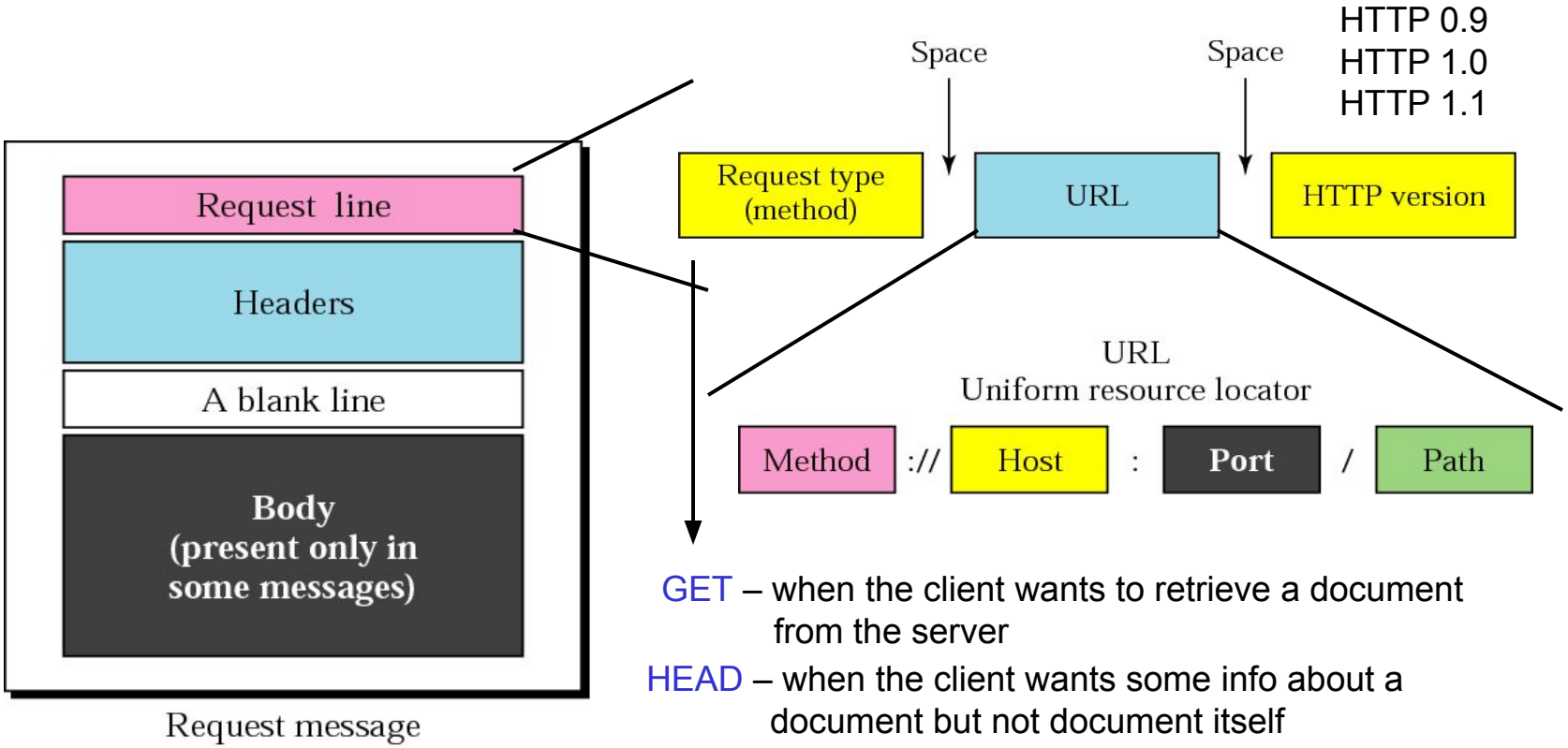
HTTP – message format

- two types of messages: request & *response*
  - ASCII (human-readable format)



HTTP protocol – message format

- HTTP request message



GET – when the client wants to retrieve a document from the server

HEAD – when the client wants some info about a document but not document itself

COPY – copies the file to another location

Method types

HTTP/1.0:

- GET
- POST
- HEAD
  - asks server to leave requested object out of response
  - i.e. request info about a doc

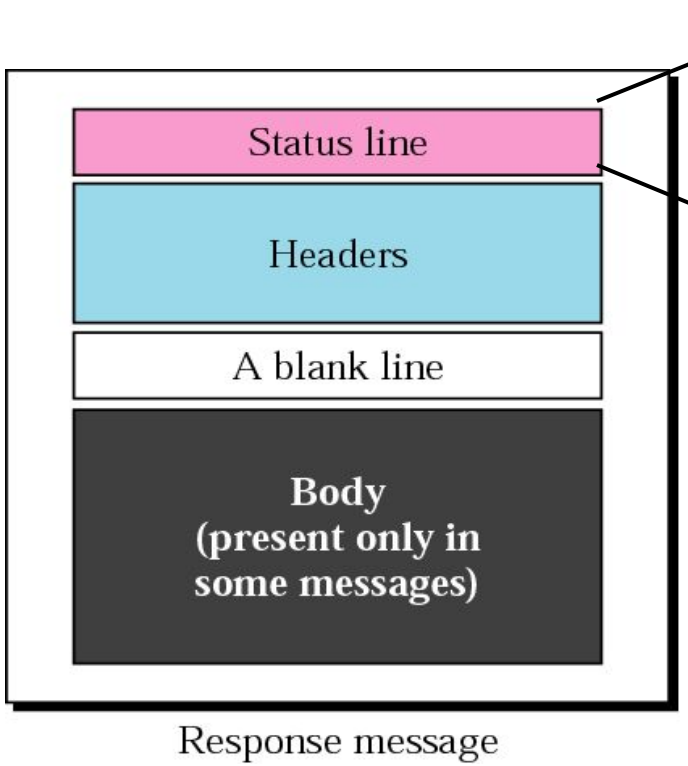
HTTP/1.1:

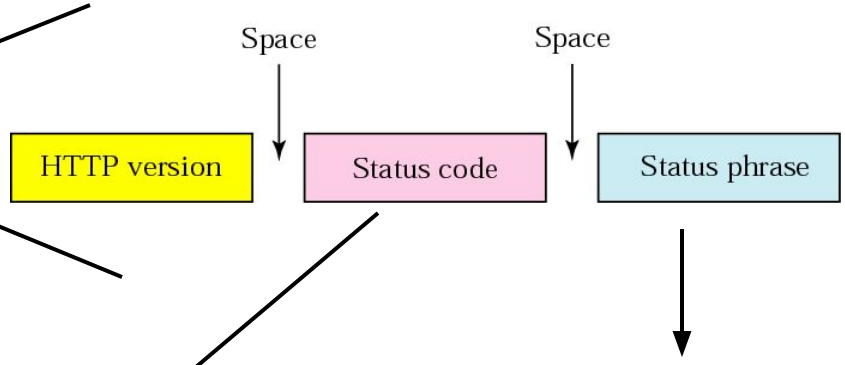
- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

HTTP – message format

- HTTP response message

http://www.w3.org/Protocols/HTTP/HTRESP.html





explains the status code in text form

200 OK – request succeeded

301 Moved Permanently – object moved

400 Bad Request – not understood by server

404 Not Found – req. document not found

HTTP response status codes

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

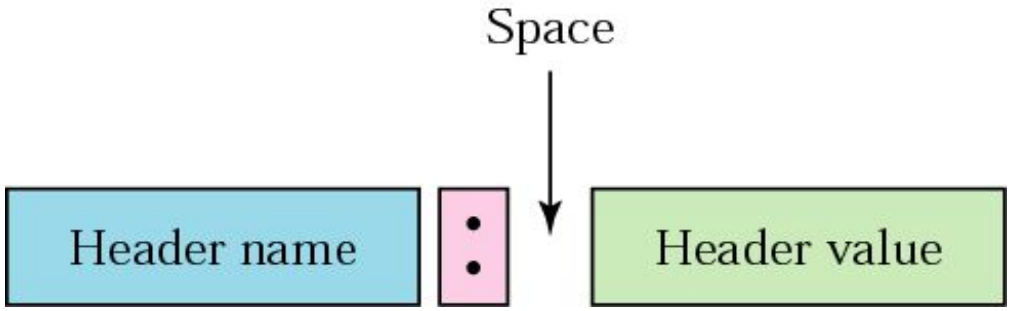
Code	*Phrase*	*Description*
	Informational	
100	Continue	The initial part of the request has been received, and the client may continue with its request.
101	Switching	The server is complying with a client request to switch protocols defined in the upgrade header.
	Success	
200	OK	The request is successful.
201	Created	A new URL is created.
202	Accepted	The request is accepted, but it is not immediately acted upon.
204	No content	There is no content in the body.

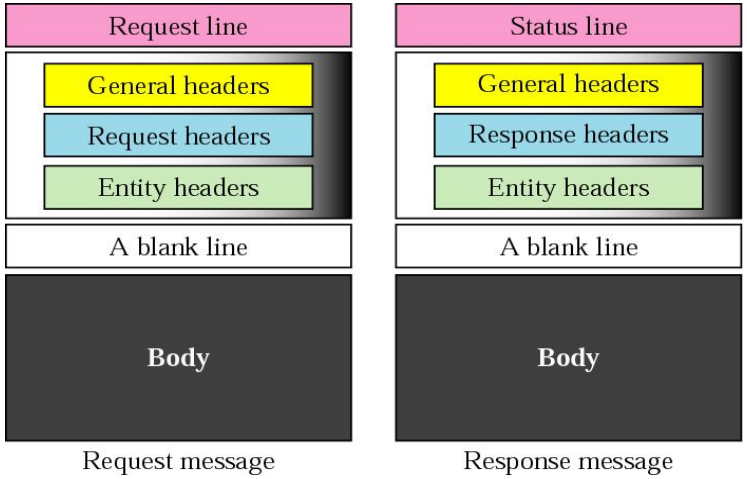
HTTP response status codes

Code	*Phrase*	*Description*
Redirection		
301	Moved permanently	The requested URL is no longer used by the server.
302	Moved temporarily	The requested URL has moved temporarily.
304	Not modified	The document has not been modified.
Client Error		
400	Bad request	There is a syntax error in the request.
401	Unauthorized	The request lacks proper authorization.
403	Forbidden	Service is denied.
404	Not found	The document is not found.
405	Method not allowed	The method is not supported in this URL.
406	Not acceptable	The format requested is not acceptable.
Server Error		
500	Internal server error	There is an error, such as a crash, at the server site.
501	Not implemented	The action requested cannot be performed.
503	Service unavailable	The service is temporarily unavailable, but may be requested in the future.

HTTP – message format

- **Headers**
  - exchange additional information between the client & the server
  - example
    - doc in a special format
    - extra info about document





HTTP – message format- headers

Table 27.3 *General headers*

Header	*Description*
Cache-control	Specifies information about caching
Connection	Shows whether the connection should be closed or not
Date	Shows the current date
MIME-version	Shows the MIME version used
Upgrade	Specifies the preferred communication protocol

HTTP – message format

Header	*Description*
Accept	Shows the medium format the client can accept
Accept-charset	Shows the character set the client can handle
Accept-encoding	Shows the encoding scheme the client can handle
Accept-language	Shows the language the client can accept
Authorization	Shows what permissions the client has
From	Shows the e-mail address of the user
Host	Shows the host and port number of the server
If-modified-since	Sends the document if newer than specified date
If-match	Sends the document only if it matches given tag
If-non-match	Sends the document only if it does not match given tag
If-range	Sends only the portion of the document that is missing
If-unmodified-since	Sends the document if not changed since specified date
Referrer	Specifies the URL of the linked document
User-agent	Identifies the client program

Table 27.4 *Request headers*

HTTP – message format

Table 27.5 *Response headers*

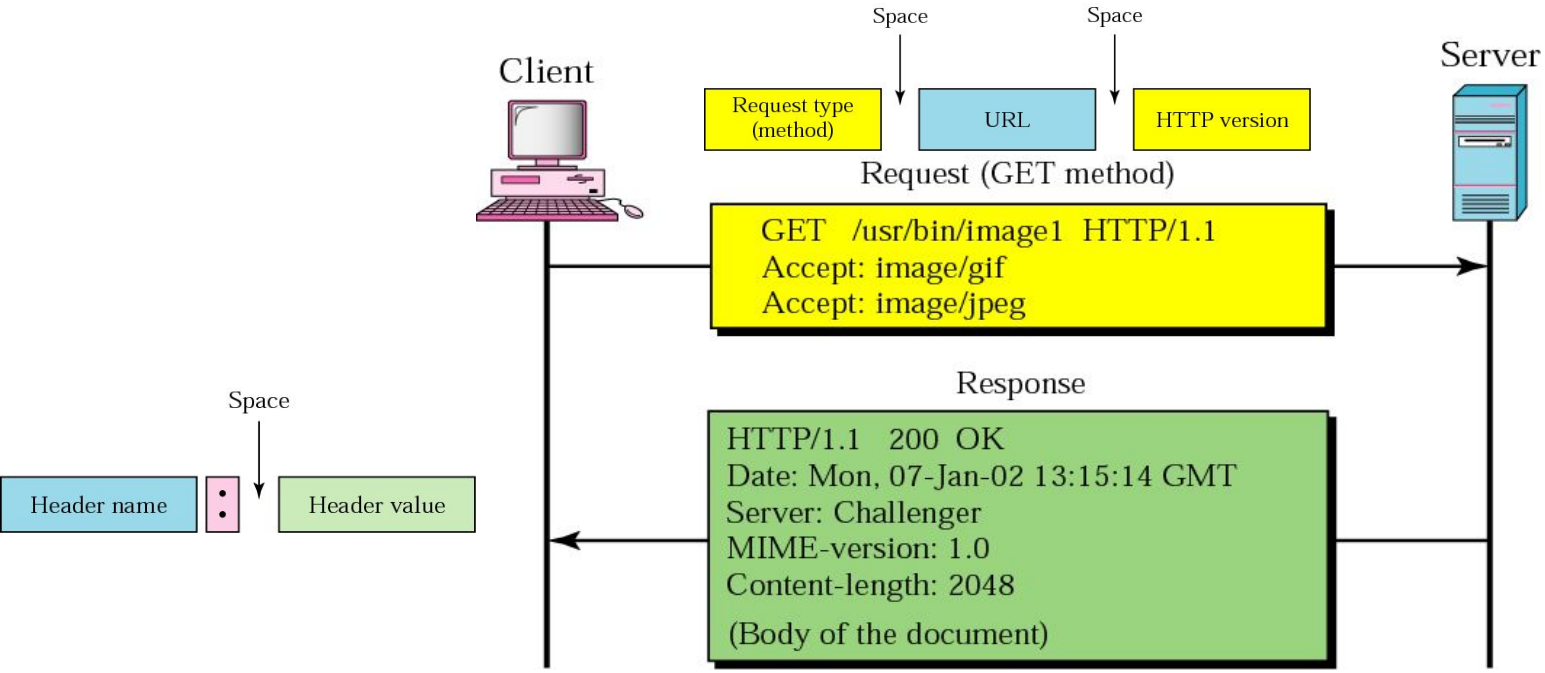
Header	*Description*
Accept-range	Shows if server accepts the range requested by client
Age	Shows the age of the document
Public	Shows the supported list of methods
Retry-after	Specifies the date after which the server is available
Server	Shows the server name and version number

HTTP – message format

Table 27.6 *Entity headers*

Header	*Description*
Allow	Lists valid methods that can be used with a URL
Content-encoding	Specifies the encoding scheme
Content-language	Specifies the language
Content-length	Shows the length of the document
Content-range	Specifies the range of the document
Content-type	Specifies the medium type
Etag	Gives an entity tag
Expires	Gives the date and time when contents may change
Last-modified	Gives the date and time of the last change
Location	Specifies the location of the created or moved document

HTTP messages – an example



This example retrieves a document.

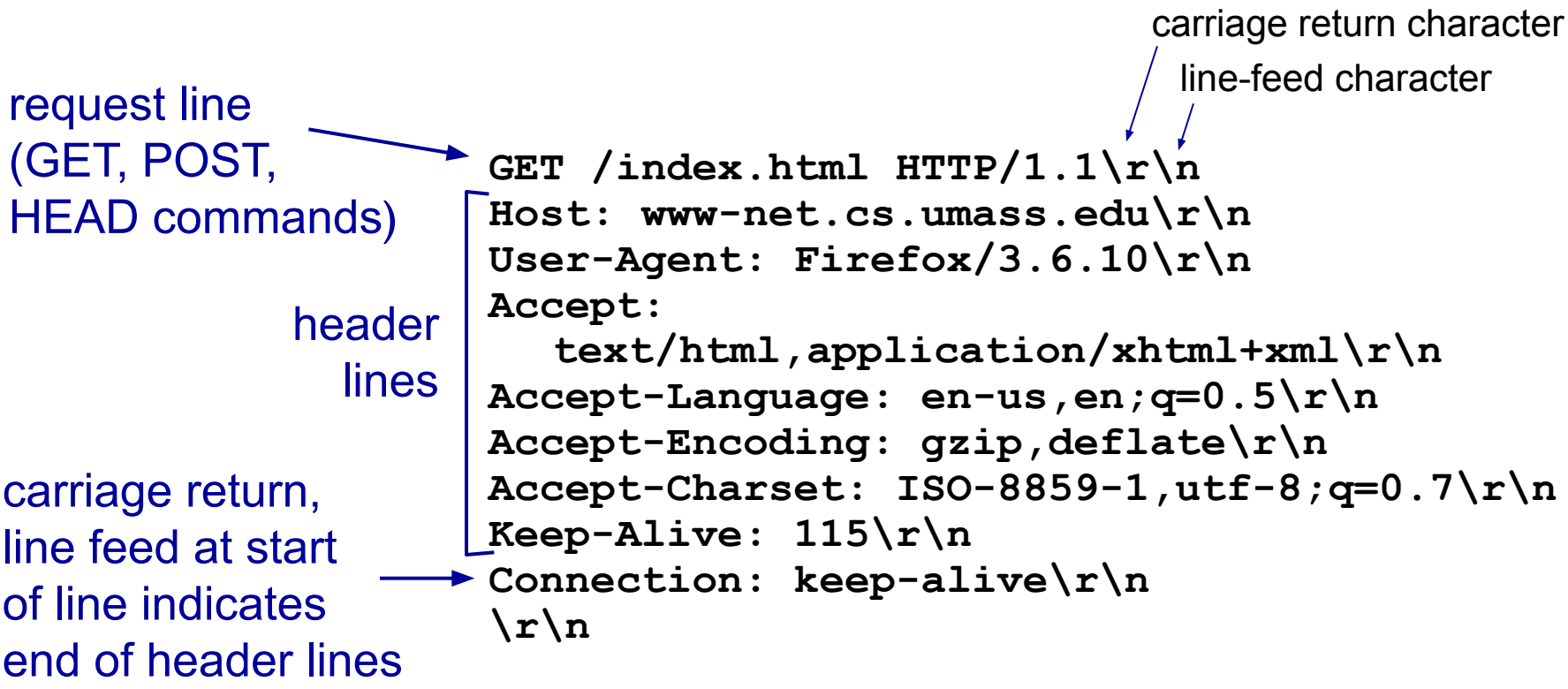
We use the GET method to retrieve an image with the path/usr/bin/image1.

The request line shows the method (GET), the URL, and the HTTP version (1.1).

The header has two lines that show that the client can accept images in GIF and JPEG format.

HTTP request message- example

- HTTP request message:



HTTP response message

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
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 ...
```

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/animalsearch?monkeys&banana
```

User-server state: cookies

many Web sites use cookies

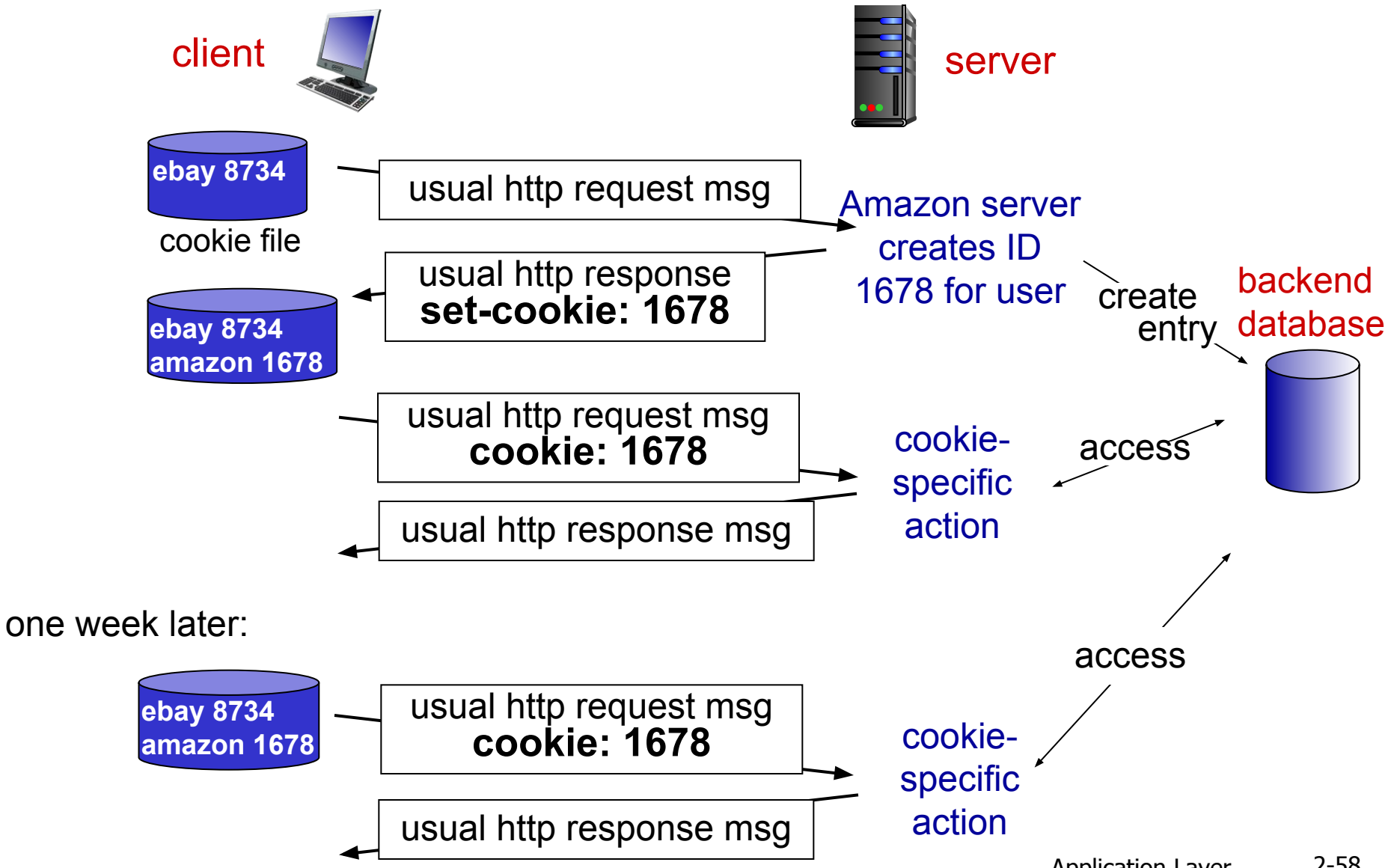
four components:

1) cookie header line of HTTP *response* message
2) cookie header line in next 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

Cookies: keeping "state" (cont.)



Cookies (continued)

what cookies can be used for:

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

aside

cookies and privacy.

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

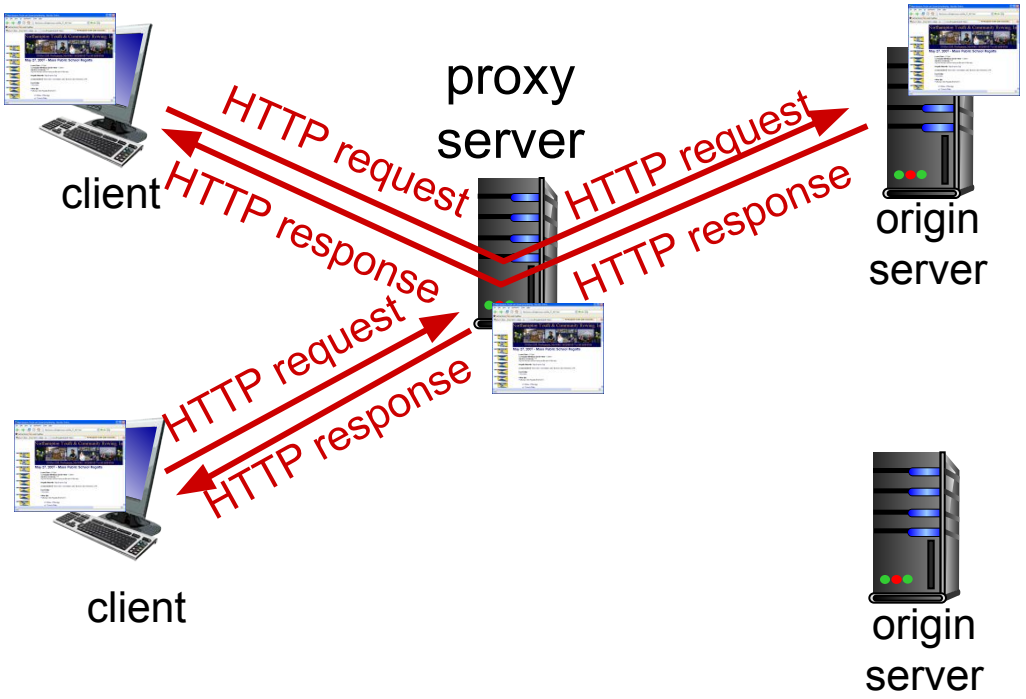
how to keep "state":

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

Web caches (proxy server)

- HTTP supports Proxy servers
- Proxy server
  - a computer that keeps copies of responses to recent requests
- Goal: satisfy a client's request without involving the original server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object 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
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)

why Web caching?

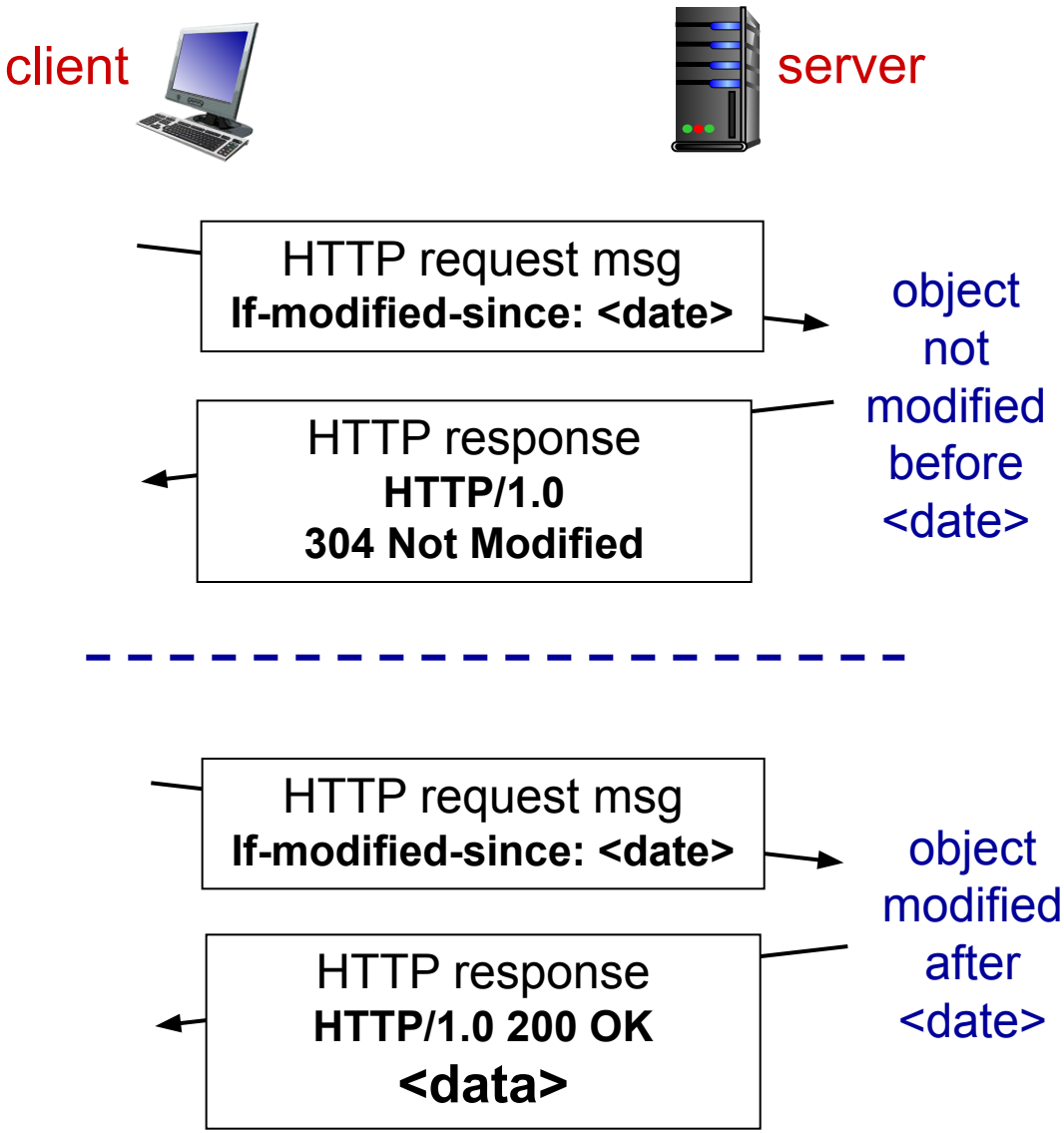
- reduce response time for client request
- reduce traffic on an institution's access link
- Internet dense with caches: enables "poor" content providers to effectively deliver content (so too does P2P file sharing)

Conditional GET

- *Goal:* don't send object if cache has up-to-date cached version
  - no object transmission delay
  - lower link utilization
- *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: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

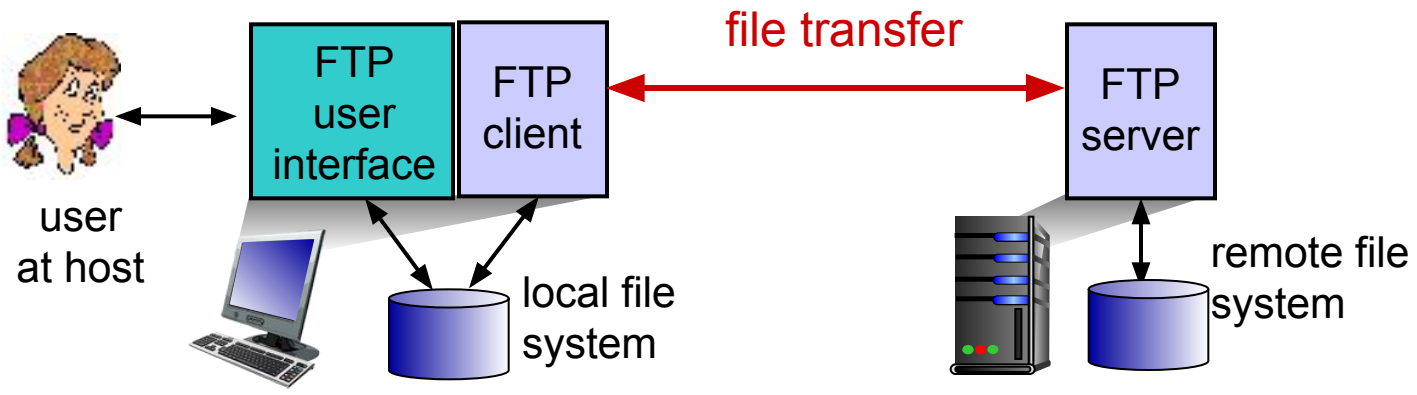
2.4 electronic mail
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

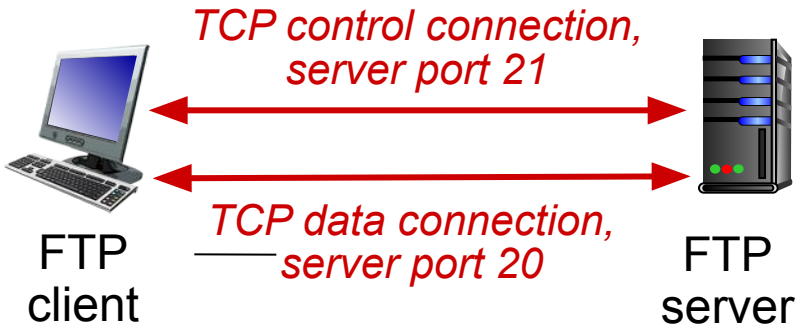
FTP: the file transfer protocol



- 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, using TCP
- client authorized over control connection
- client browses remote directory, sends commands over control connection
- when server receives file transfer command, *server* opens *2nd* TCP data 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`**

Chapter 2: outline

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

2.6 P2P applications

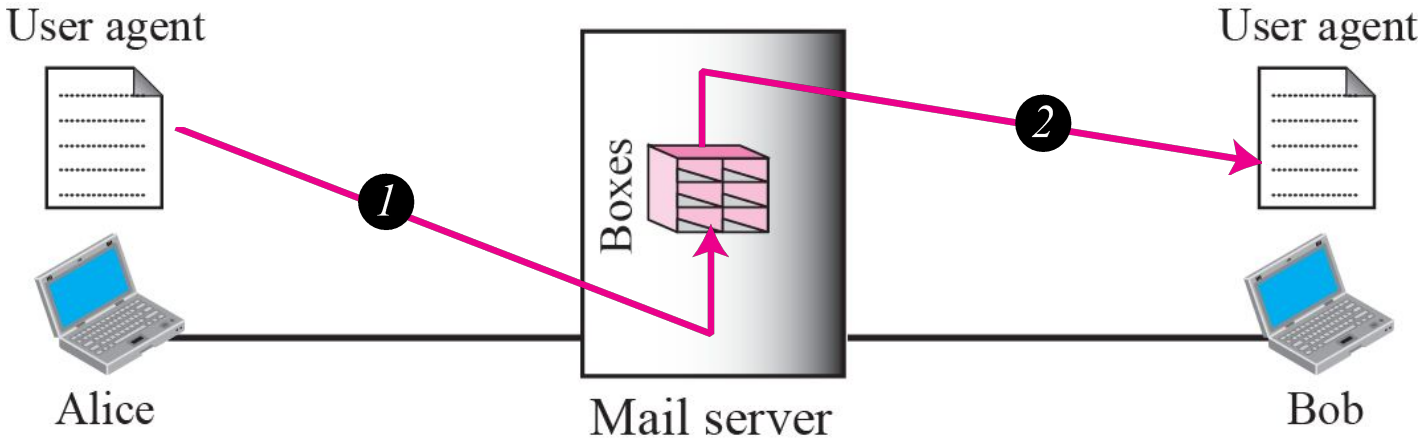
2.7 socket programming with UDP and TCP

Electronic Mail

- One of the most popular Internet services is electronic mail (e-mail).
- The designers of the Internet probably never imagined the popularity of this application program.
- Its architecture consists of several components that we discuss in this chapter.

Email - Architecture

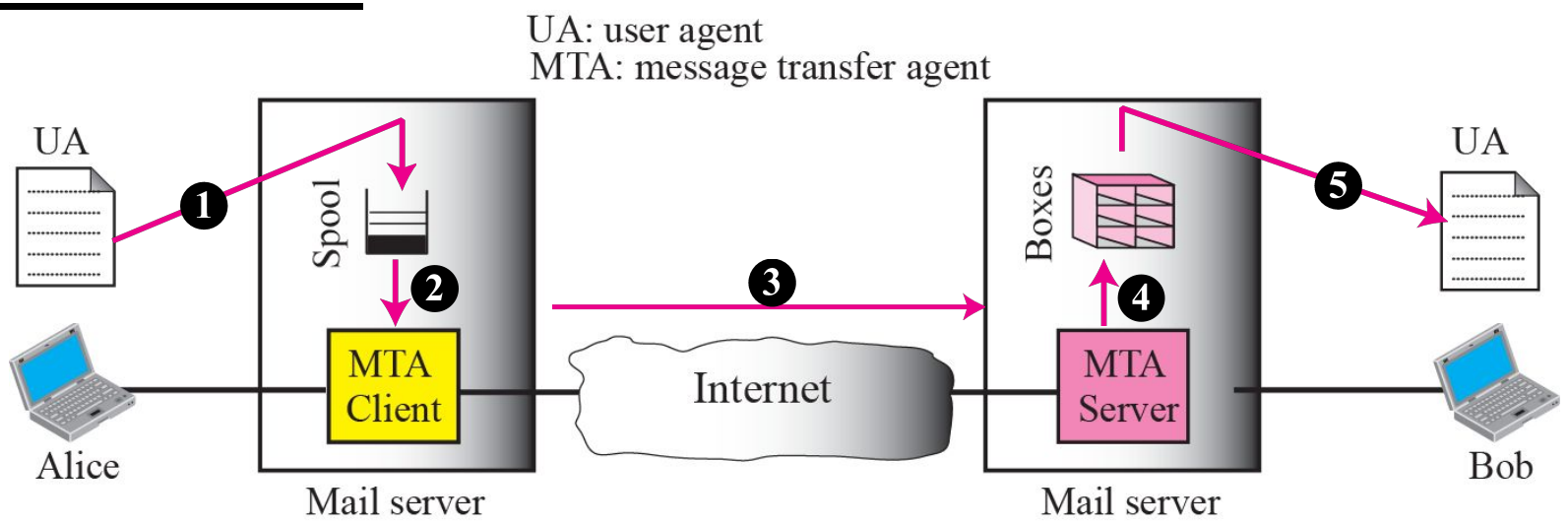
- First scenario



- When the sender and the receiver of an e-mail are on the same mail server,
- we need only two user agents.

Email - Architecture

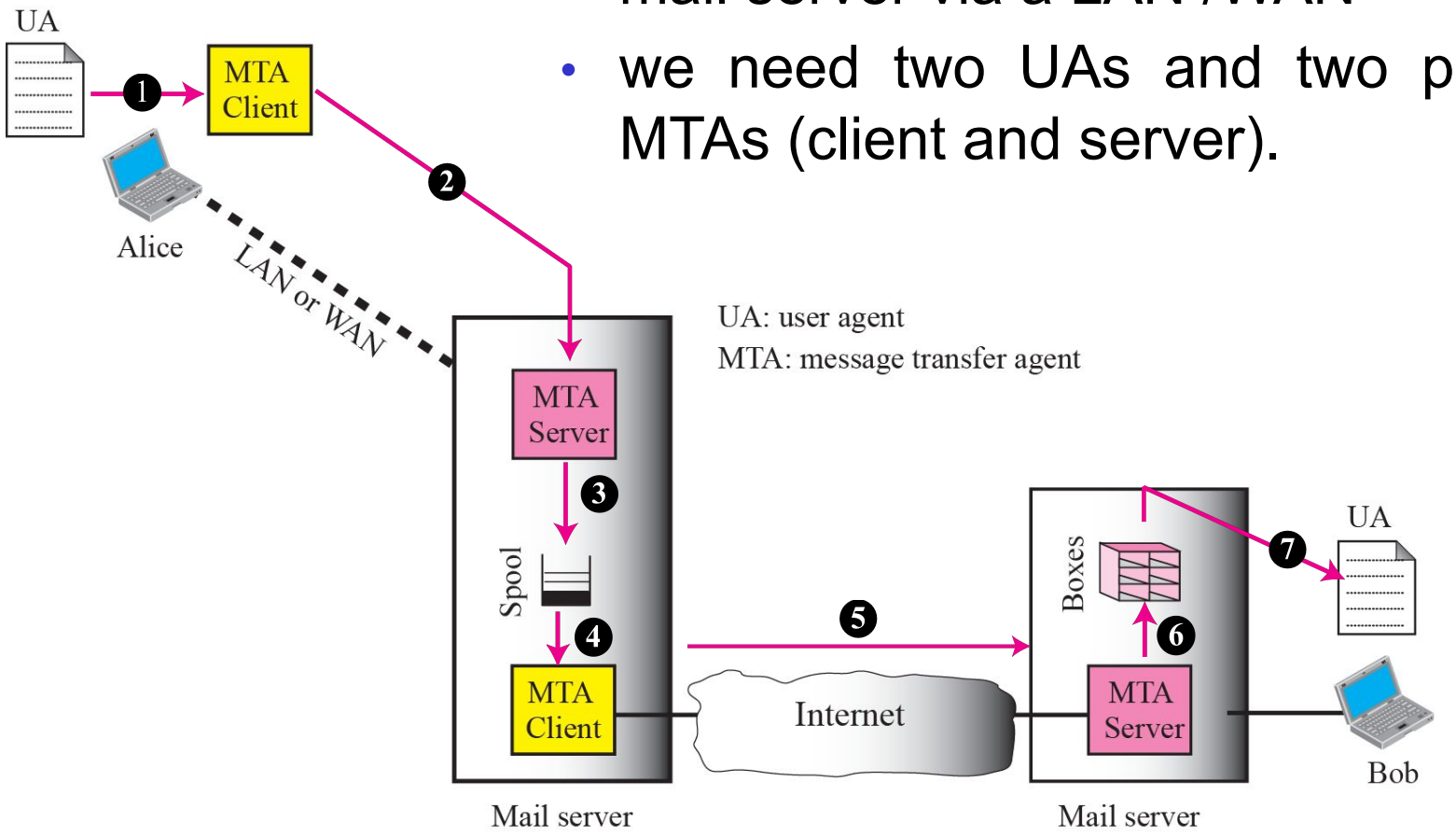
- 2nd scenario



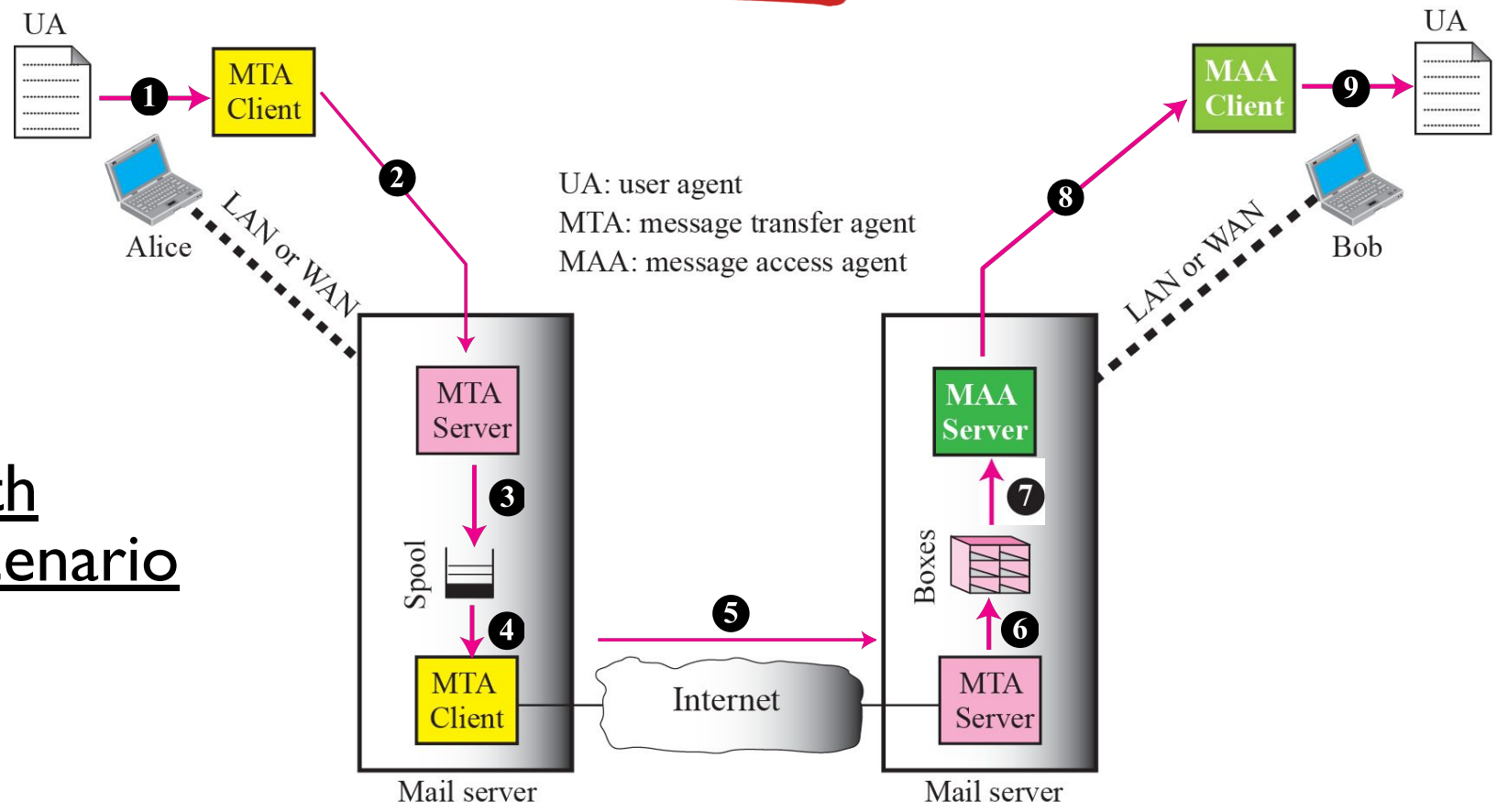
- When the sender and the receiver of an e-mail are on different mail servers,
- we need two UAs and a pair of MTAs (client and server).

Email - Architecture

- 3rd scenario
  - When the sender is connected to the mail server via a LAN /WAN
  - we need two UAs and two pairs of MTAs (client and server).



Email – Architecture

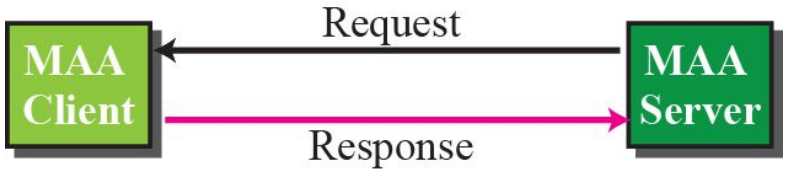


- **4th scenario**

- both sender and receiver are connected to the mail server via a LAN/ WAN
- need two UAs, two pairs of MTAs (client and server), and a pair of MAAs (client and server).
- This is the most common situation today.

Figure 23.5 *Push versus pull*

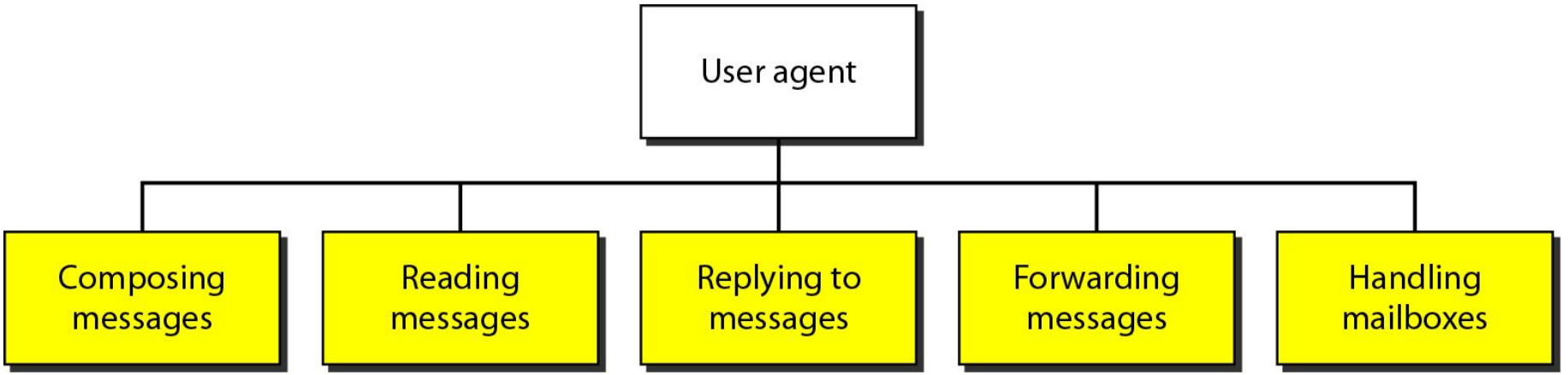
a. Client pushes messages



b. Client pulls messages

User Agent

- It provides service to the user to make the process of sending and receiving a message easier.
- e.g., Outlook, Thunderbird, iPhone mail client



Message Transfer Agent

- The actual mail transfer is done through message transfer agents (MTAs).
- To send mail, a system must have the client MTA, and to receive mail, a system must have a server MTA.
- The formal protocol that defines the MTA client and server in the Internet is called Simple Mail Transfer Protocol (SMTP).

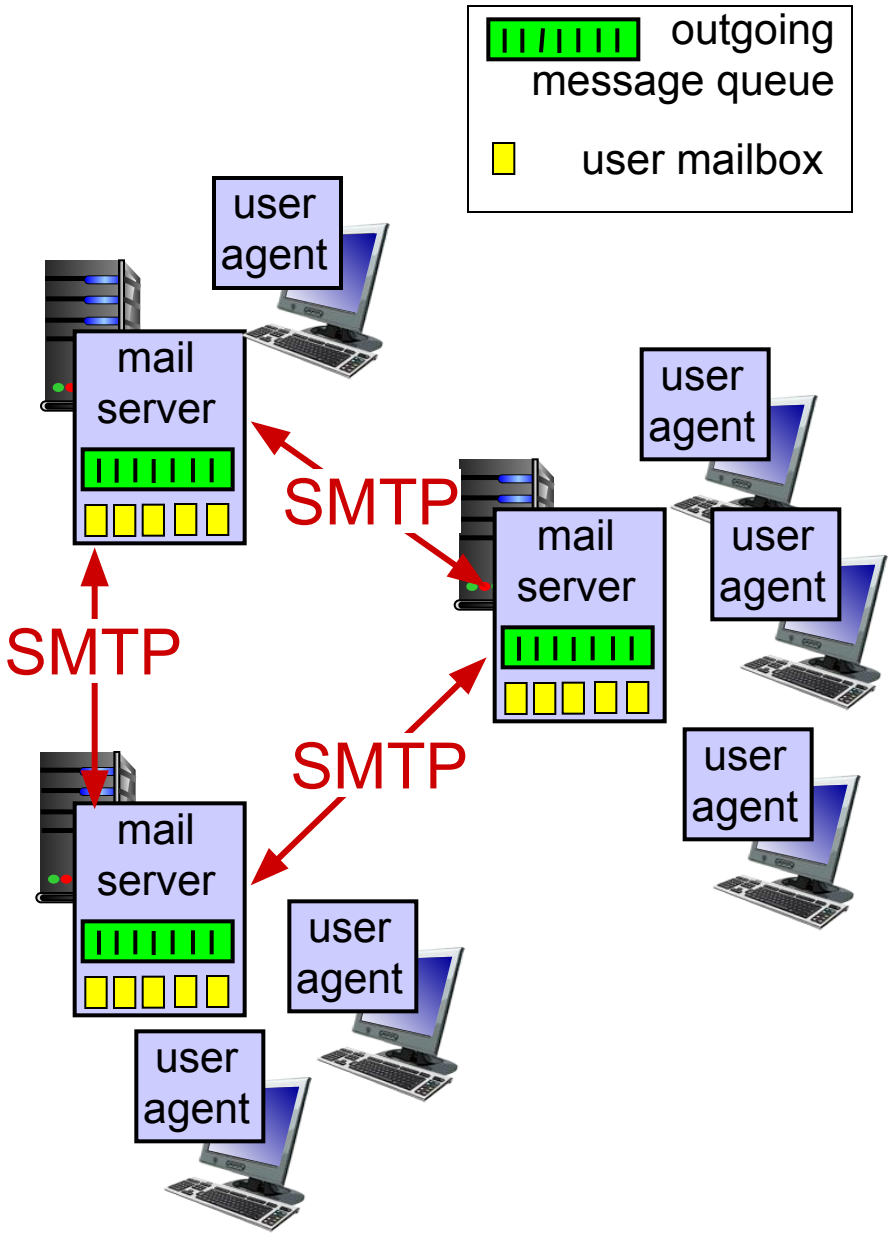
Electronic mail

Three major components:

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

User Agent

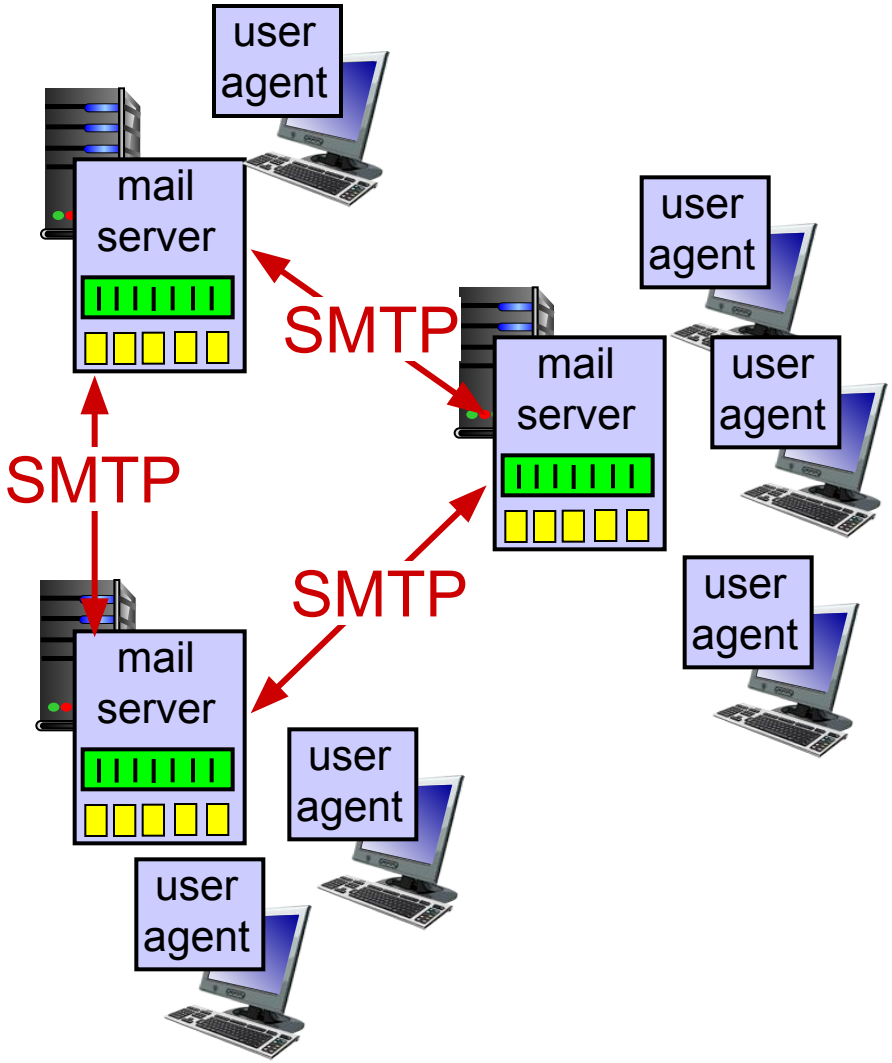
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



Electronic mail: mail servers

mail servers:

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

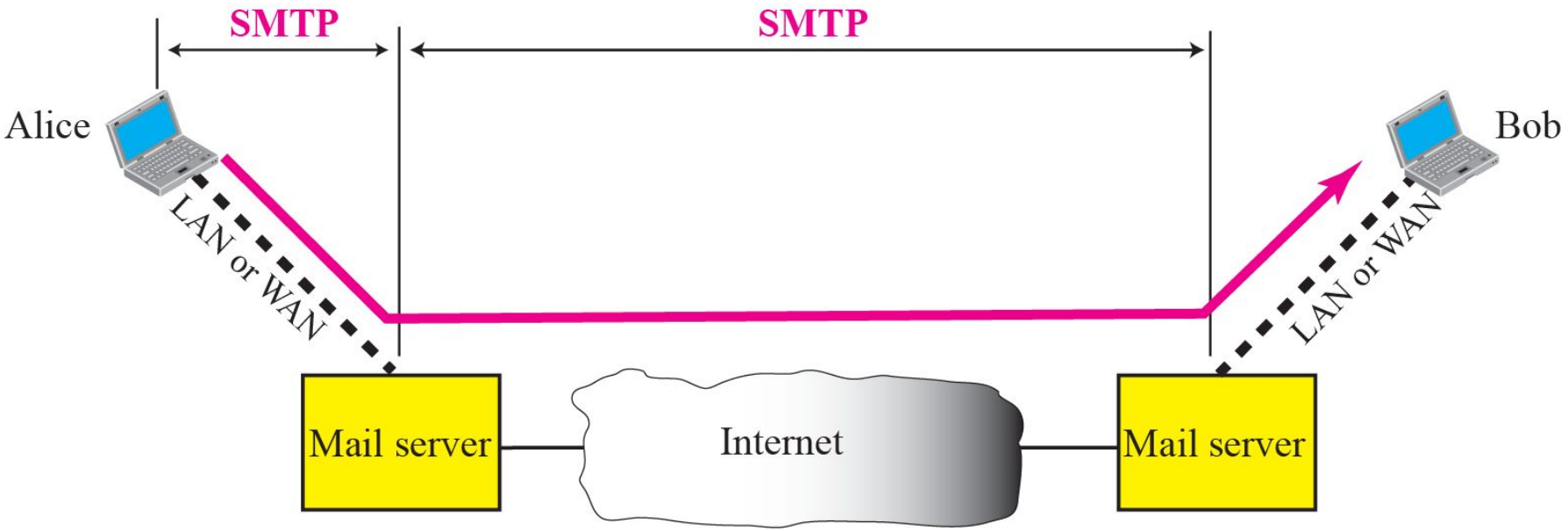


Electronic Mail: SMTP [RFC 2821]

- 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 (like HTTP, FTP)
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCI

Electronic Mail: SMTP [RFC 2821]

- Range of SMTP



Electronic Mail: SMTP [RFC 2821]

- Commands

Keyword	*Argument(s)*	*Keyword*	*Argument(s)*
HELO	Sender's host name	NOOP	
MAIL FROM	Sender of the message	TURN	
RCPT TO	Intended recipient	EXPN	Mailing list
DATA	Body of the mail	HELP	Command name
QUIT		SEND FROM	Intended recipient
RSET		SMOL FROM	Intended recipient
VRFY	Name of recipient	SMAL FROM	Intended recipient

Electronic Mail: SMTP [RFC 2821]

- Responses

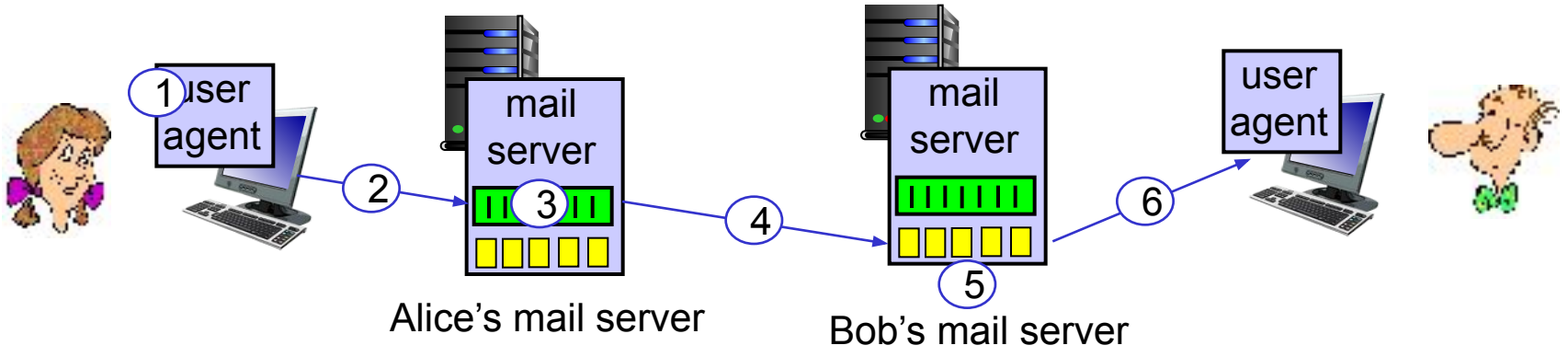
Code	*Description*
	Positive Completion Reply
211	System status or help reply
214	Help message
220	Service ready
221	Service closing transmission channel
250	Request command completed
251	User not local; the message will be forwarded
	Positive Intermediate Reply
354	Start mail input

Electronic Mail: SMTP [RFC 2821]

Transient Negative Completion Reply	
421	Service not available
450	Mailbox not available
451	Command aborted: local error
452	Command aborted; insufficient storage
Permanent Negative Completion Reply	
500	Syntax error; unrecognized command
501	Syntax error in parameters or arguments
502	Command not implemented
503	Bad sequence of commands
504	Command temporarily not implemented
550	Command is not executed; mailbox unavailable
551	User not local
552	Requested action aborted; exceeded storage location
553	Requested action not taken; mailbox name not allowed
554	Transaction failed

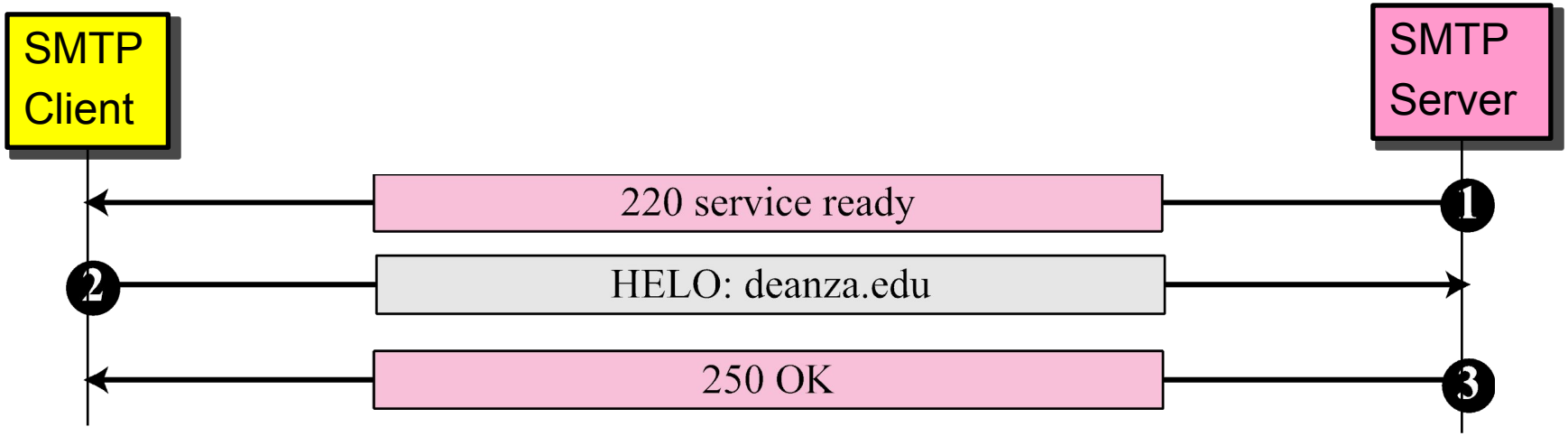
Scenario: Alice sends message to Bob

1) Alice uses UA to compose message "to" `bob@someschool.edu`
2) Alice's UA sends message to her mail server; message placed in message queue
3) client side of SMTP opens TCP connection with Bob's mail server
4) SMTP client sends Alice's message over the TCP connection
5) Bob's mail server places the message in Bob's mailbox
6) Bob invokes his user agent to read message

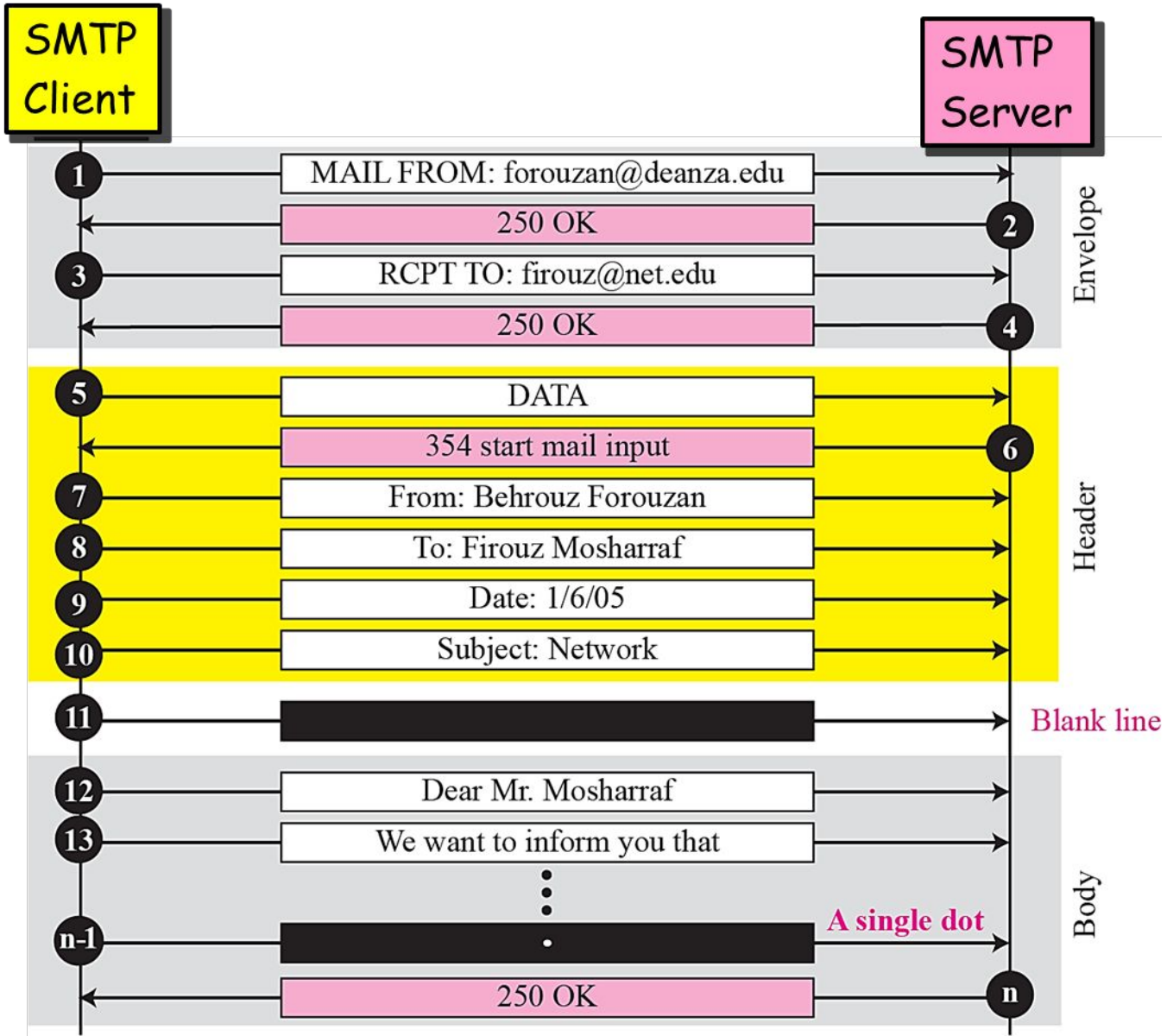


Sample SMTP interaction

- Connection Establishment or handshaking

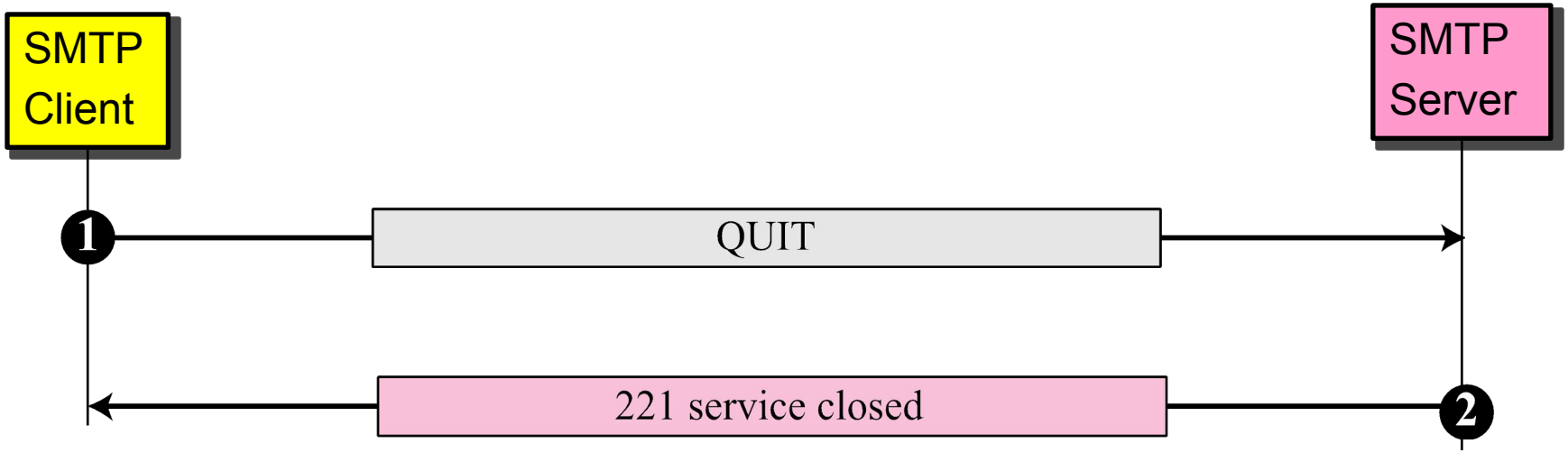


Sample SMTP interaction- Message Transfer



Sample SMTP interaction

- Connection termination



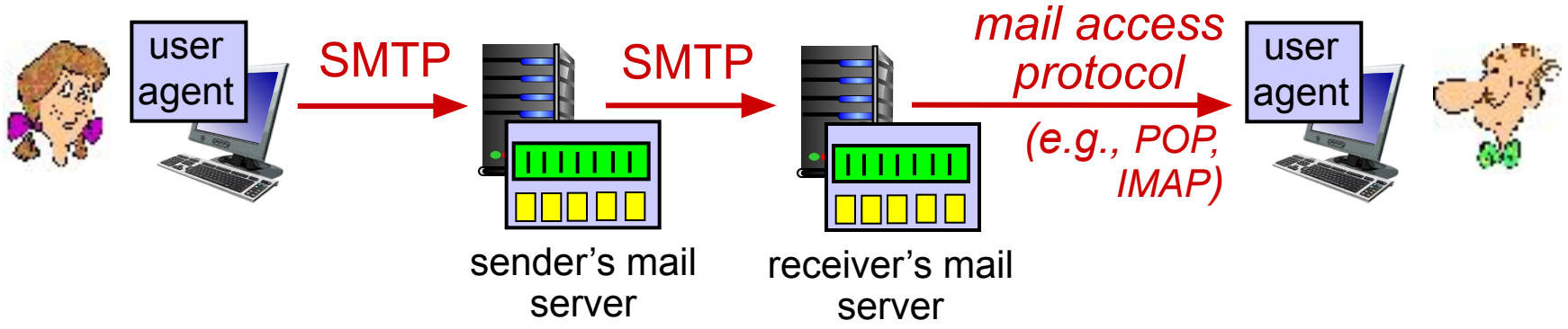
SMTP: final words

- 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:

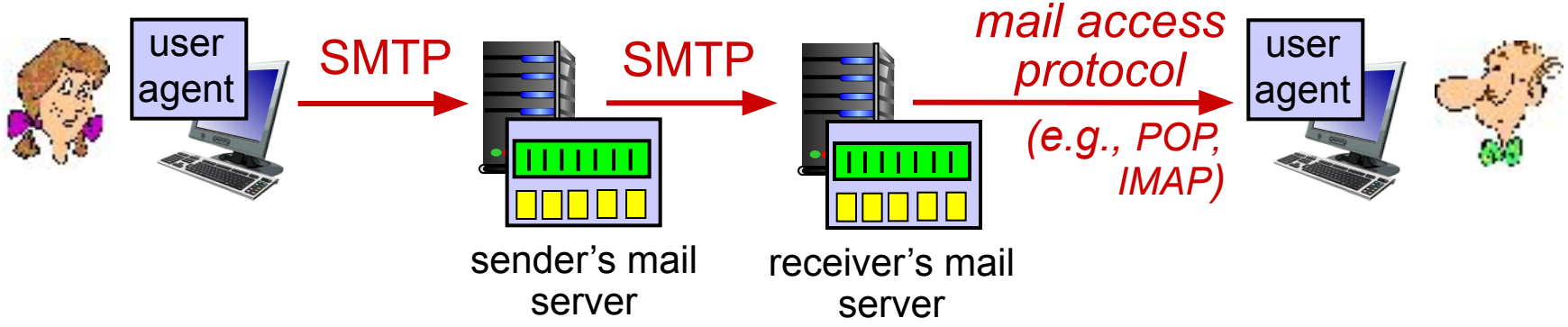
- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

Mail Access Agents



- SMTP: is a push protocol; it pushes the message from the client to the server.
- On the other hand, the third stage needs a pull protocol;
- the client must pull messages from the server.
- The third stage uses a message access agent.

Mail access protocols



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

POP3 protocol

authorization phase

- client commands:
  - **`user`**: declare username
  - **`pass`**: password
- server responses
  - **`+OK`**
  - **`-ERR`**

transaction phase, client:

- **`list`**: list message numbers
- **`retr`**: retrieve message by number
- **`dele`**: delete
- **`quit`**

```
S: +OK POP3 server ready
C: user 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 2 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

Limitations of POP

- Does not handle multiple mailboxes easily
  - Designed to put user's incoming e-mail in one folder
- Not designed to keep messages on the server
  - Instead, designed to download messages to the client
- Poor handling of multiple-client access to mailbox
  - Increasingly important as users have home PC, work PC, laptop, cyber café computer, friend's machine, etc.
- High network bandwidth overhead
  - Transfers all of the e-mail messages, often well before they are read (and they might not be read at all!)

IMAP – Internet Message Access Protocol

- What happens if a user opens his mailbox in more than one place?
- Many people have a single e-mail account at work or school and want to access it from work, from their home PC, from their laptop when on business trips, etc.
- While POP3 allows this, since it normally downloads all stored messages at each contact, the result is that the user's e-mail quickly gets spread over multiple machines, more or less at random, some of them not even the user's.

Internet Mail Access Protocol (IMAP)

- Developed after POP and attempts to fix POP deficiencies
  - allows keeping all mail on the server
  - allows mail categorization via folder metaphor
  - mail is easily flagged (answered, draft, deleted, seen, recent); this isn't the same on all servers
  - provides for multiple connections to the server

IMAP - process

- make connection
- send user credentials (userid and password)
  - repeat until done
    - send a command
    - read response
  - disconnect

IMAP - Commands

- login
- list
- status
- examine
- select
- create, delete, rename
- fetch
- store
- close
- expunge
- copy
- idle
- lsub, subscribe, unsubscribe
- logout
- capability, getquotaroot, getacl

POP3 (more) and IMAP

more about POP3

- previous example uses POP3 "download and delete" mode
  - Bob cannot re-read e-mail if he changes client
- POP3 "download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

IMAP

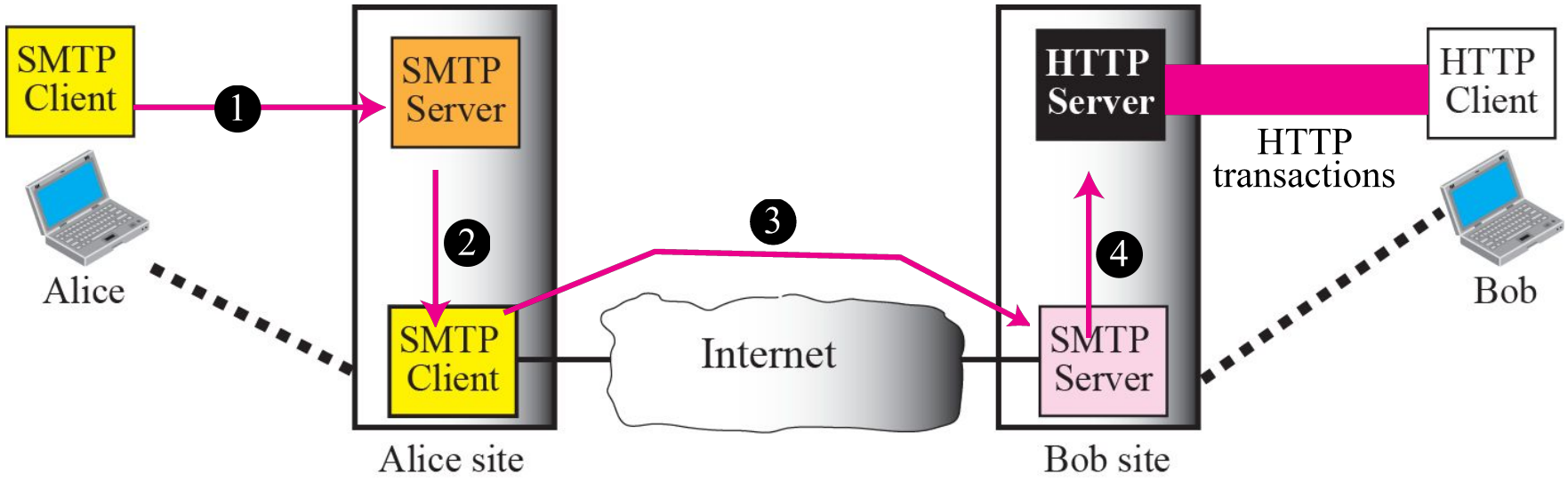
- keeps all messages in one place: at server
- allows user to organize messages in folders
- keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name

A comparison of POP3 and IMAP

Feature	POP3	IMAP
Where is emails stored?	User's PC	Server
Where is emails read?	Off-line	On-line
Connect time required	Little	Much
Use of server resources	Minimal	Extensive
Multiple mailboxes	No	Yes
User control over downloading	Little	Great
Partial message downloads	No	Yes
Are disk quotas a problem?	No	Could be
Simple to implement	Yes	No
Keeps user state	No	Yes

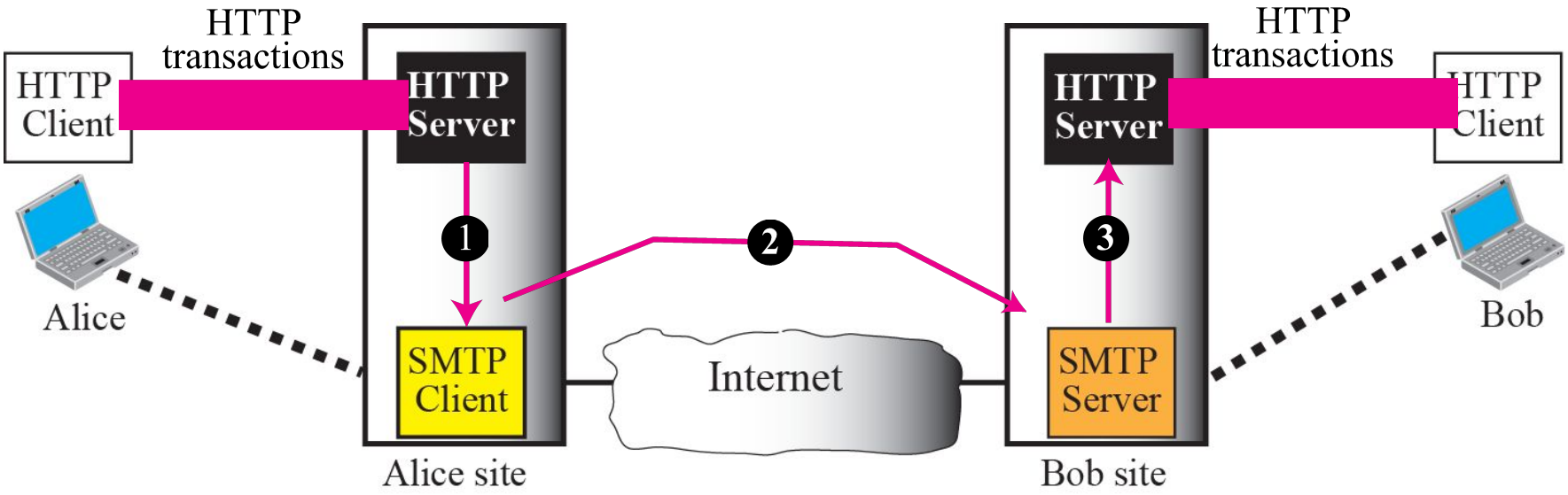
Web-Based E-Mail

case 1



Web-Based E-Mail

case 2



Web-Based E-Mail

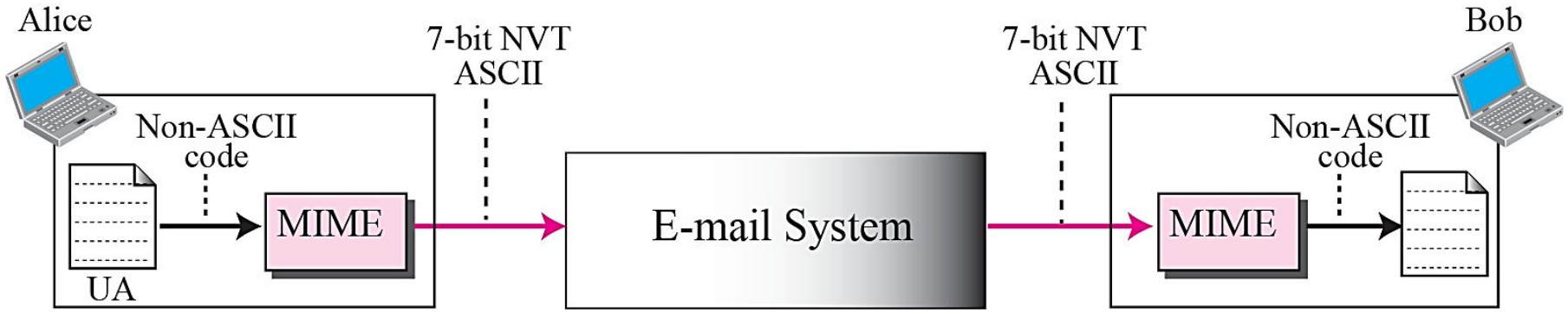
- User agent is an ordinary Web browser
  - User communicates with server via HTTP
  - E.g., Gmail, Yahoo mail, and Hotmail
- Reading e-mail
  - Web pages display the contents of folders
  - … and allow users to download and view messages
  - "GET" request to retrieve the various Web pages
- Sending e-mail
  - User types the text into a form and submits to the server
  - "POST" request to upload data to the server
  - Server uses SMTP to deliver message to other servers
- Easy to send anonymous e-mail (e.g., spam)

Limitation: Sending Non-Text Data

- E-mail body is 7-bit U.S. ASCII
  - What about non-English text?
  - What about binary files (e.g., images and executables)?
- Solution: MIME

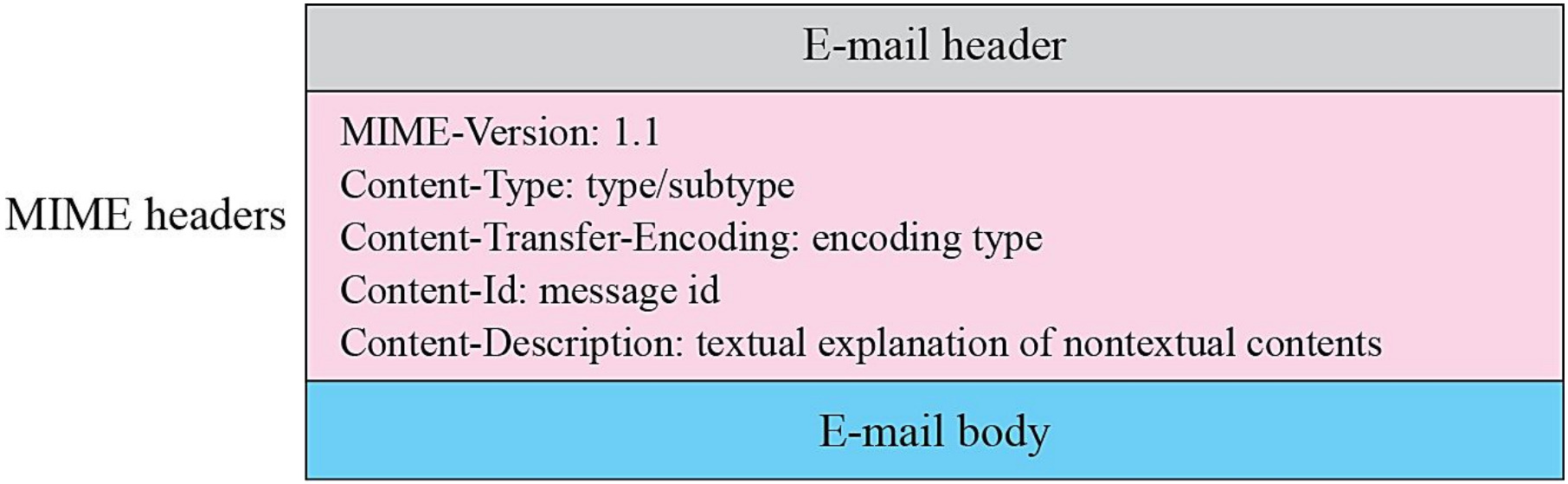
MIME – What is it?

- **Multipurpose Internet Mail Extensions**
- MIME permits the inclusion of virtually any type of file or document in an email message.
- Specifically, MIME messages can contain
  - text, images, audio, video
  - application-specific data.
    - spreadsheets
    - word processing documets



MIME

- Additional headers to describe the message body



Chapter 2: outline

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

2.6 P2P applications

2.7 socket programming with UDP and TCP

DNS: domain name system

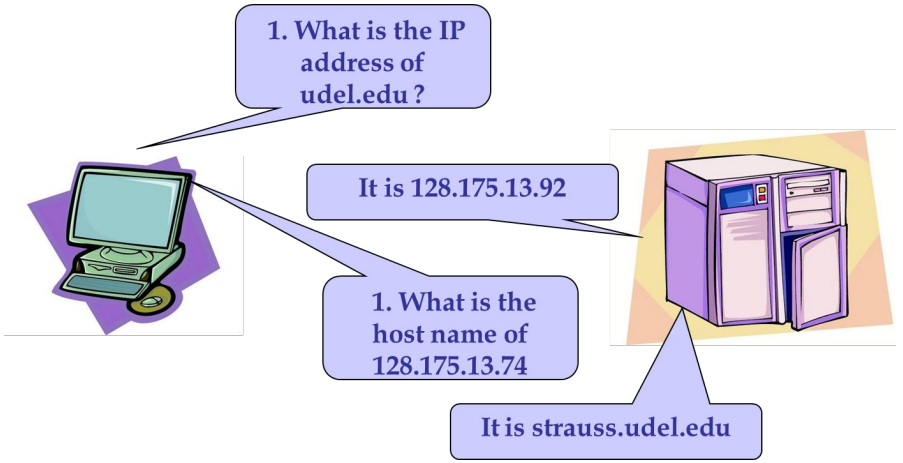
people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- "name", e.g., www.yahoo.com - used by humans

Q: how to map between IP address and name, and vice versa ?



DNS: domain name system

- *distributed database* implemented in hierarchy of many *name servers*
  - Data is maintained locally, but retrievable globally
    - No single computer has all DNS data

- *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)

DNS: services, structure

DNS services

- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

why not centralize DNS?

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

A: doesn't scale!

DNS Components

There are 3 components:

- Name Space:

  Specifications for a structured name space and data associated with the names

- Resolvers:

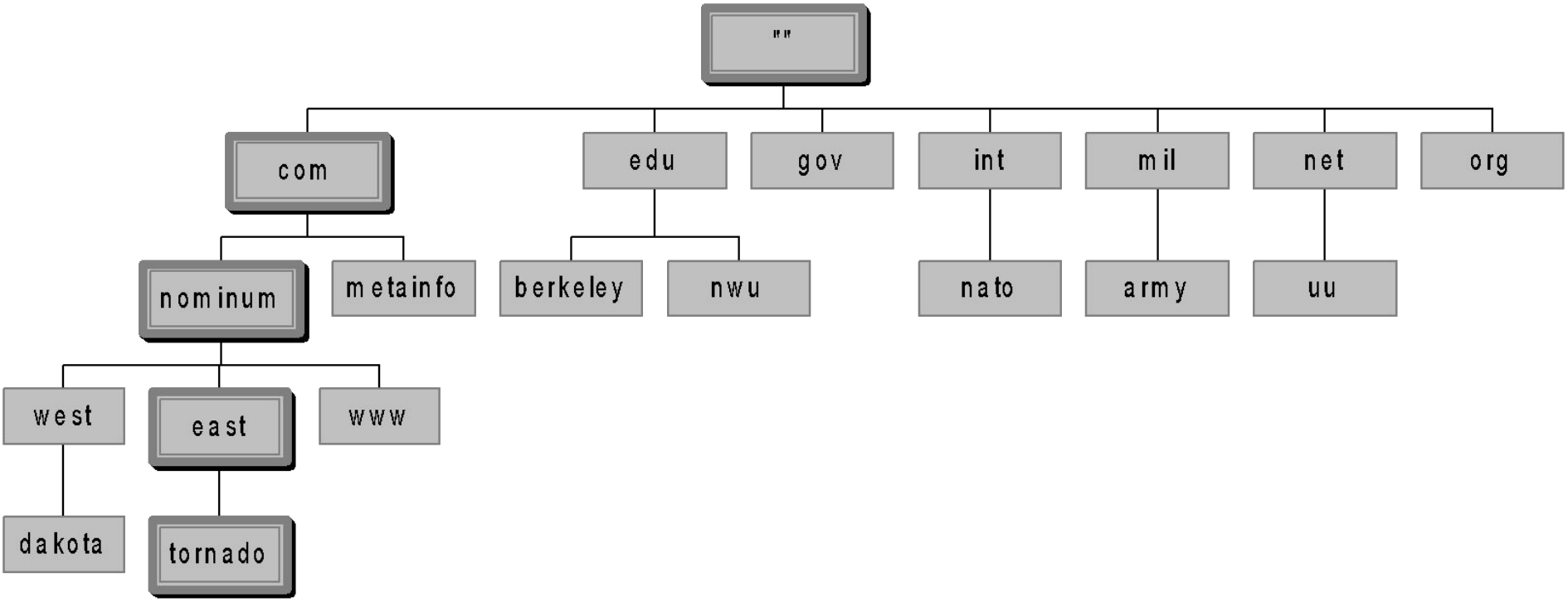
  Client programs that extract information from Name Servers.

- Name Servers:

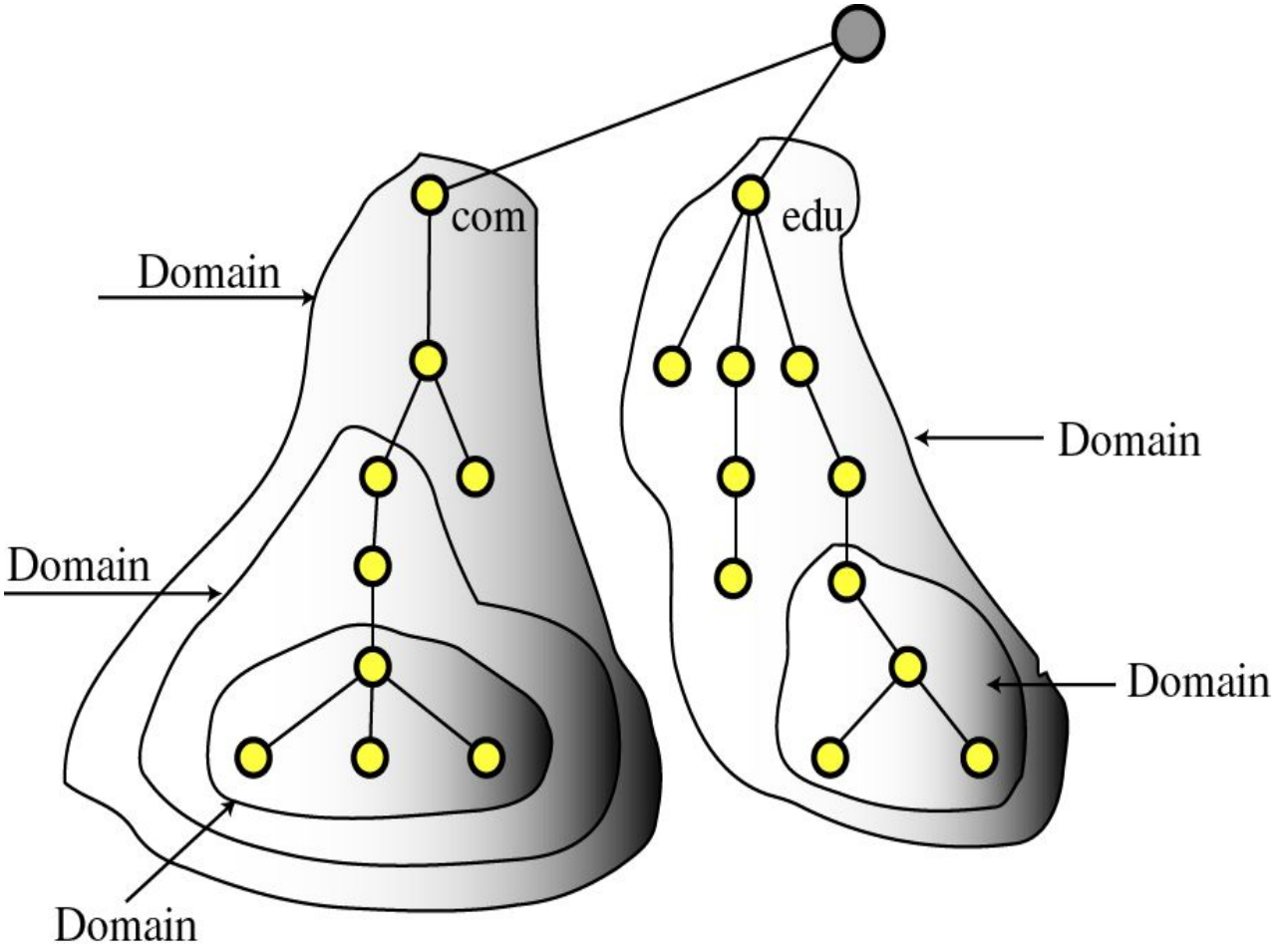
  Server programs which hold information about the structure and the names.

Domain Names

- A *domain name* is the sequence of labels from a node to the root, separated by dots (“.”s), read left to right
  - The name space has a maximum depth of 127 levels
  - Domain names are limited to 255 characters in length
- A node’s domain name identifies its position in the name space

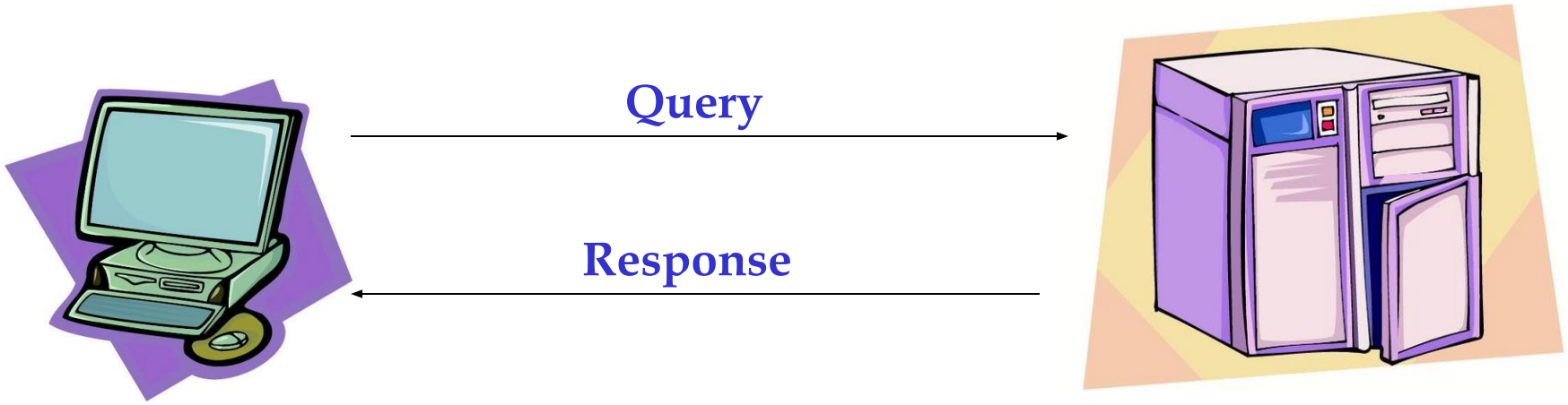


Name Space

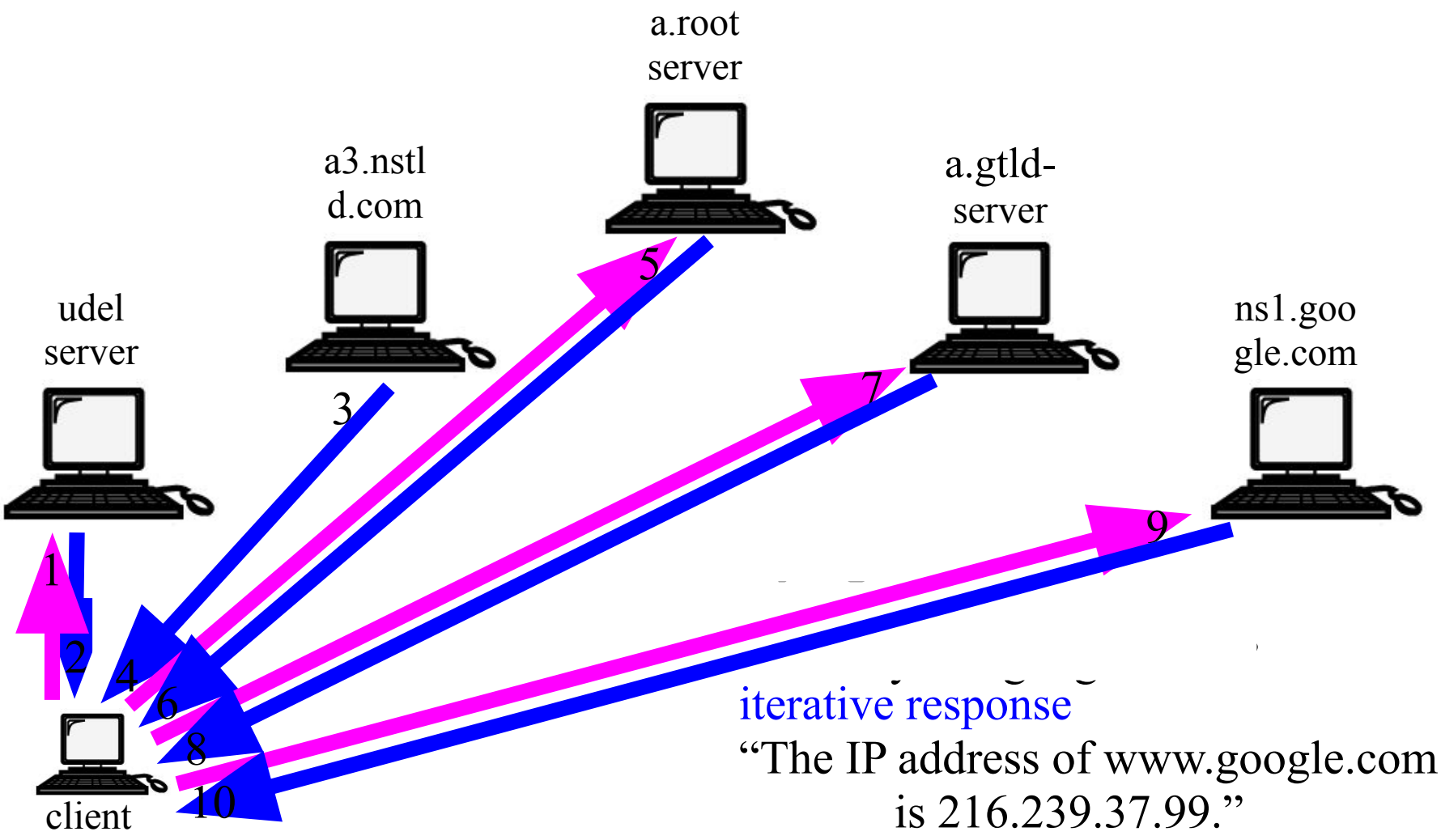


Resolvers

A Resolver maps a name to an address and vice versa.



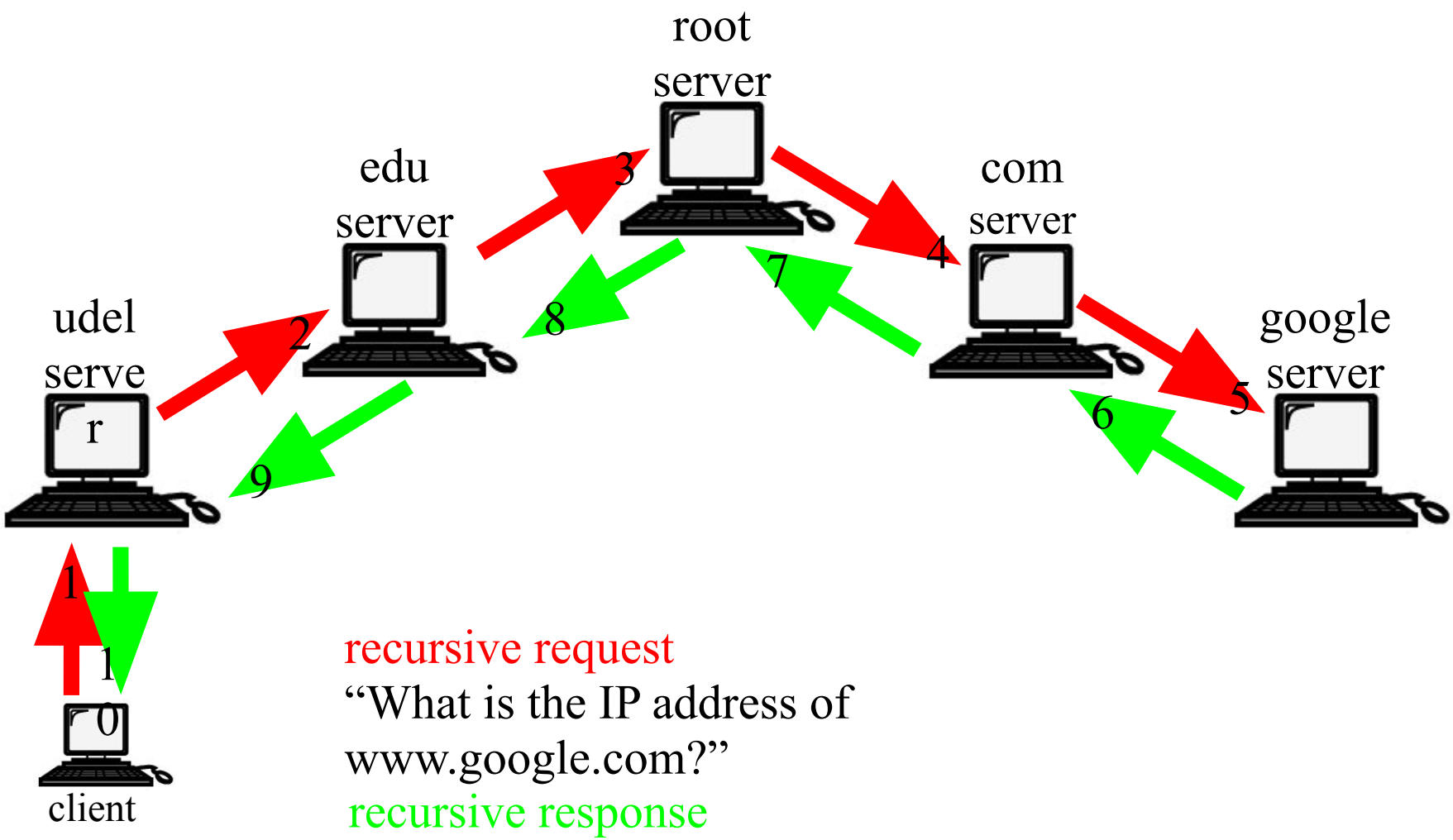
Iterative Resolution



iterative request
"What is the IP address of www.google.com?"

iterative response
"The IP address of www.google.com is 216.239.37.99."

Recursive Resolution

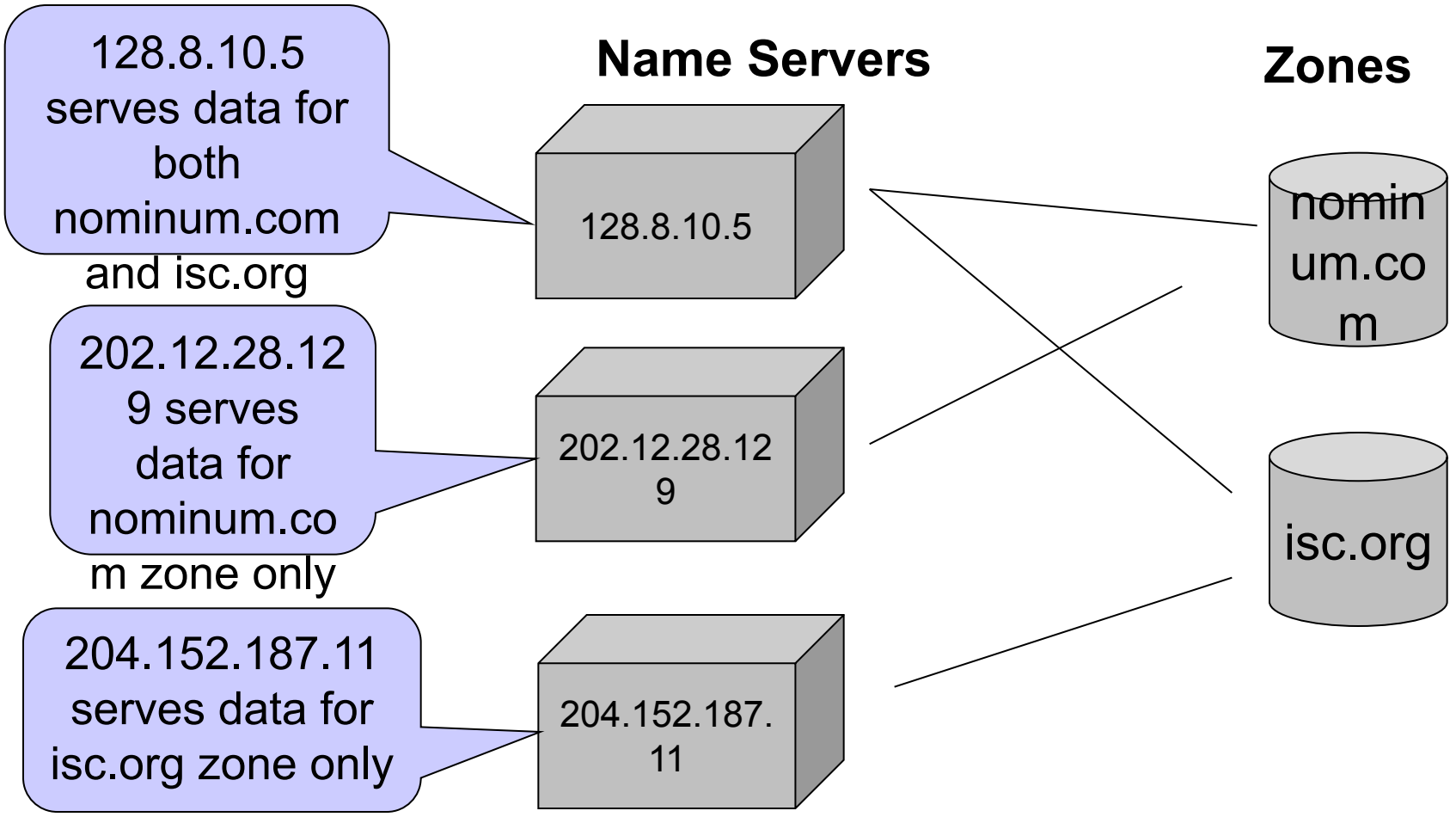


recursive request
"What is the IP address of www.google.com?"
recursive response
"The IP address of www.google.com is 216.239.37.99."

Name Servers

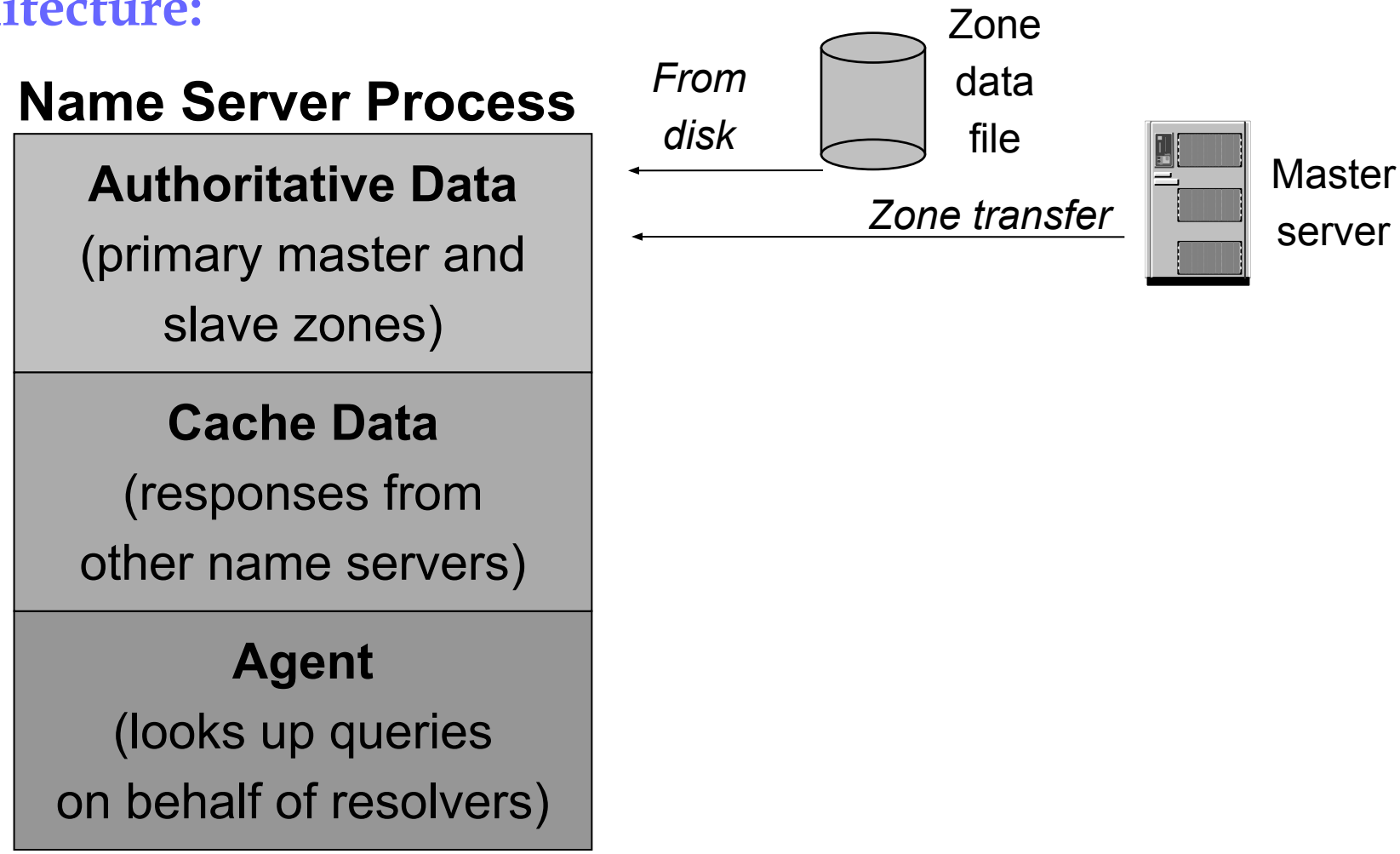
- Name servers store information about the name space in units called "zones"
  - The name servers that load a complete zone are said to "have authority for" or "be authoritative for" the zone
- Usually, more than one name server are authoritative for the same zone
  - This ensures redundancy and spreads the load
- Also, a single name server may be authoritative for many zones

Name Servers and Zones



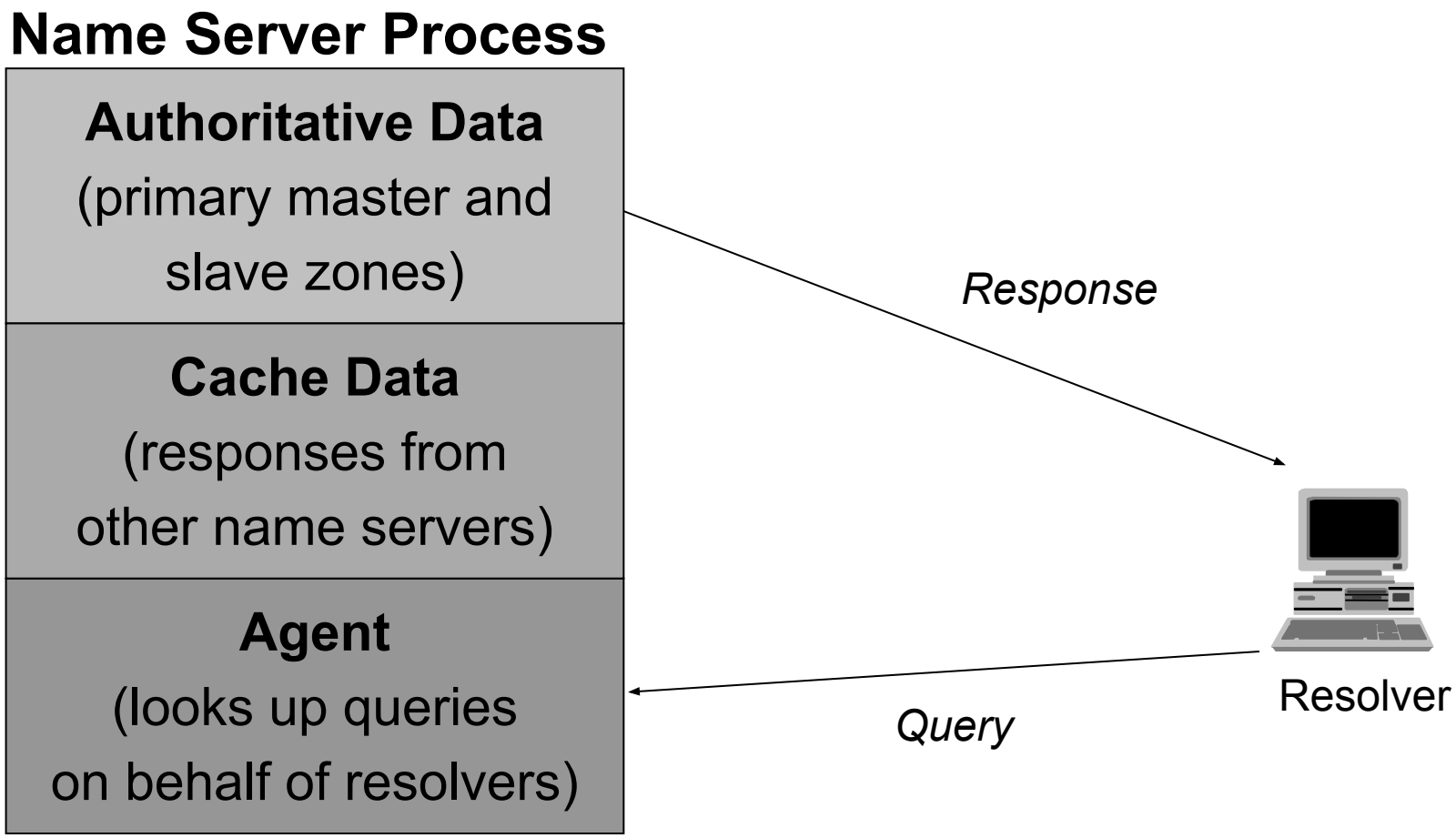
Name Server

Architecture:



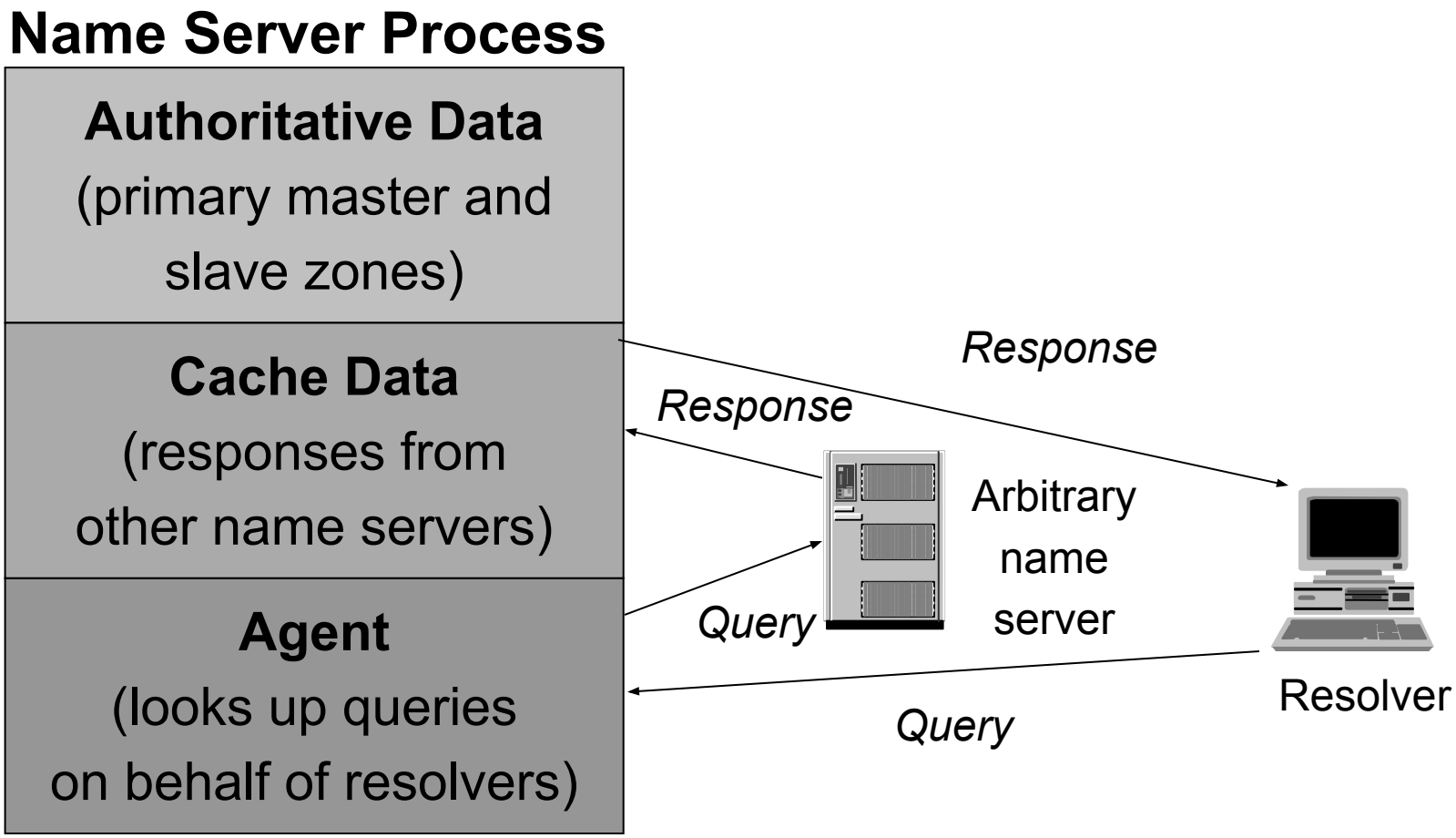
Name Server (cont'd)

Authoritative Data:



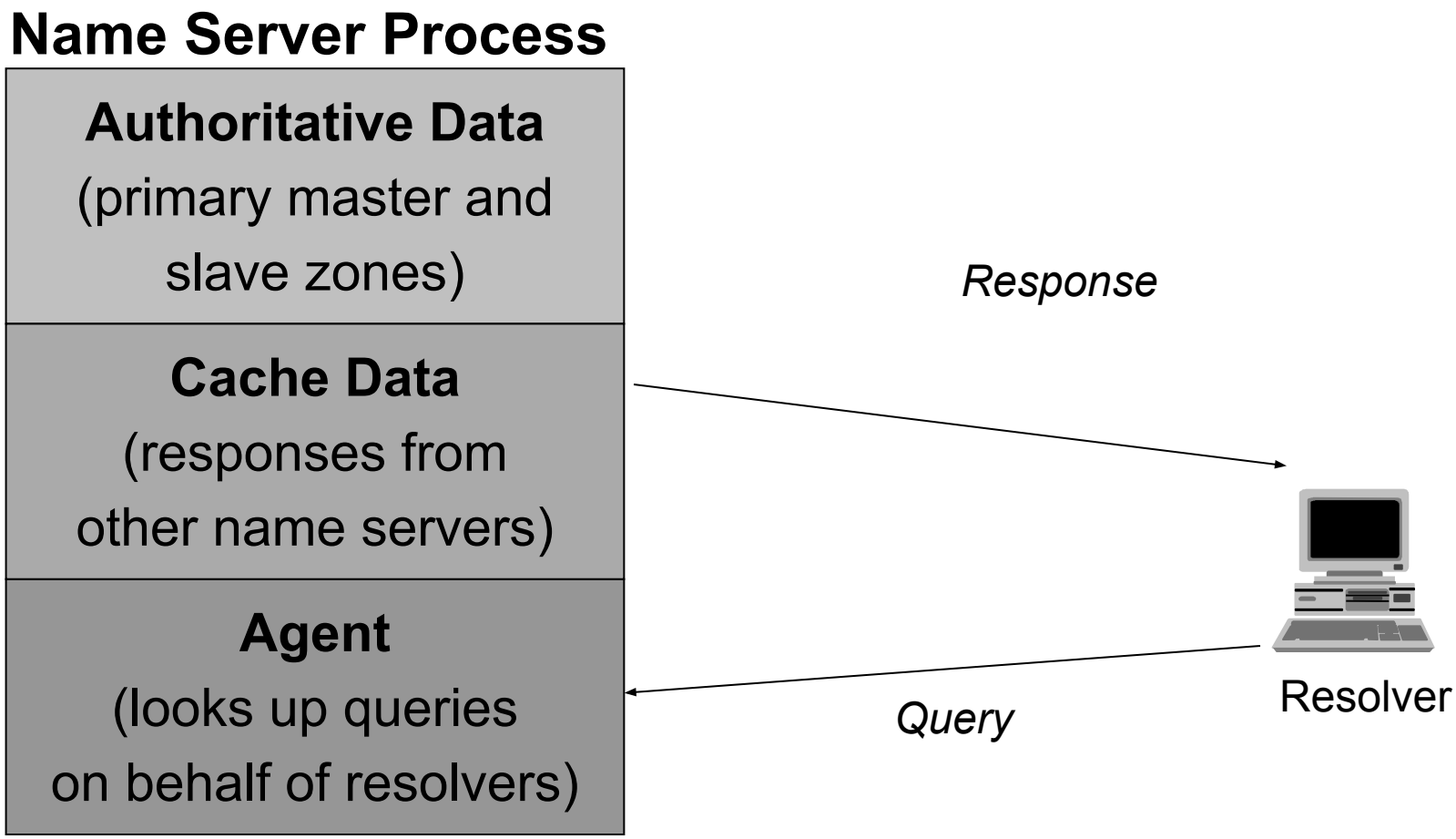
Name Server (cont'd)

Using Other Name Servers:



Name Server (cont'd)

Cached Data :

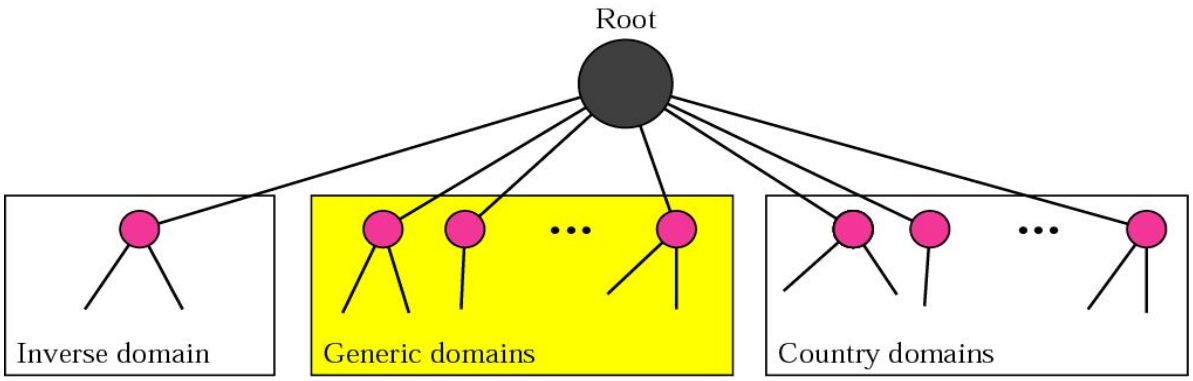


DNS Structure and Hierarchy

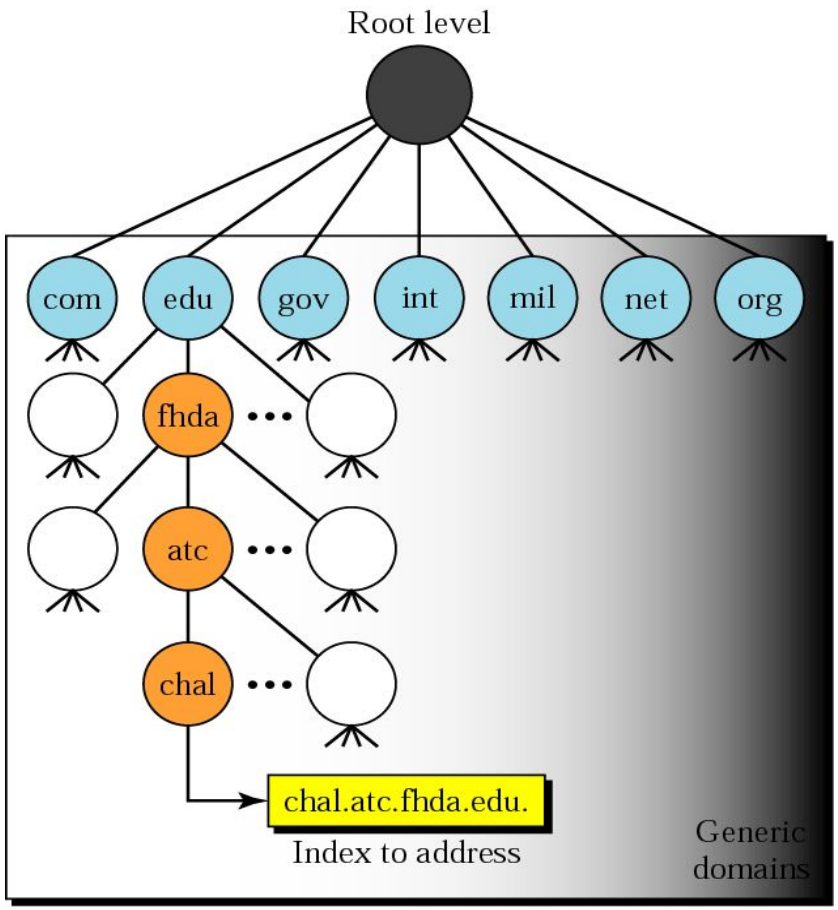
- The DNS imposes no constraints on how the DNS hierarchy is implemented except:
  - A single root
  - The label restrictions
  - So, can we create a host with a name *a.wonderful.world*?
- If a site is not connected to the Internet, it can use any domain hierarchy it chooses
  - Can make up whatever TLDs (top level domains) you want
- Connecting to the Internet implies use of the existing DNS hierarchy

DNS in the Internet

- Generic domains
  - registered host according to their generic behavior
- Inverse domain
  - used to map an address to a name
- Country domains
  - the same format as in generic domain just 2 character format
    - us; nl; be; fr;

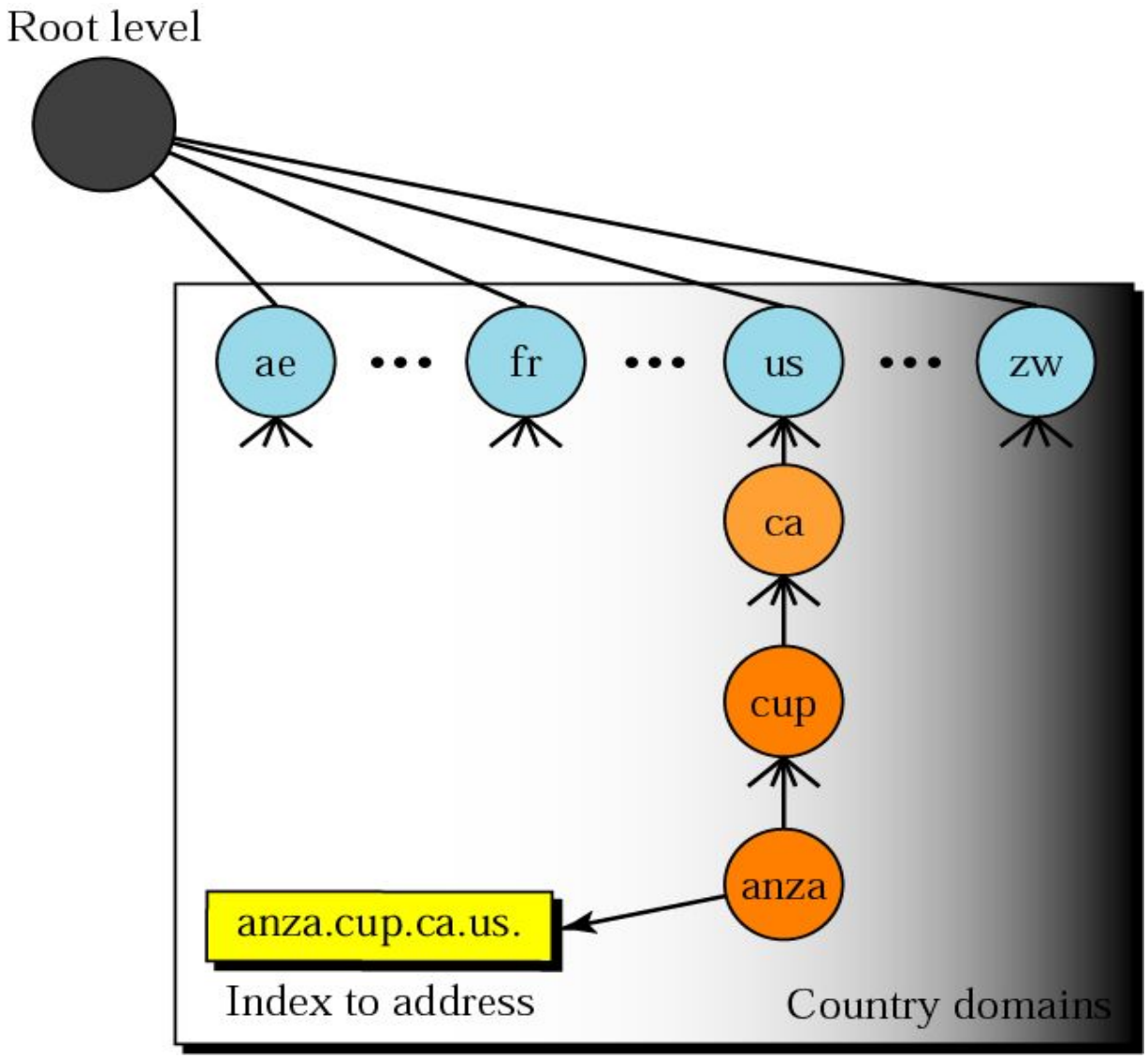


Generic domain



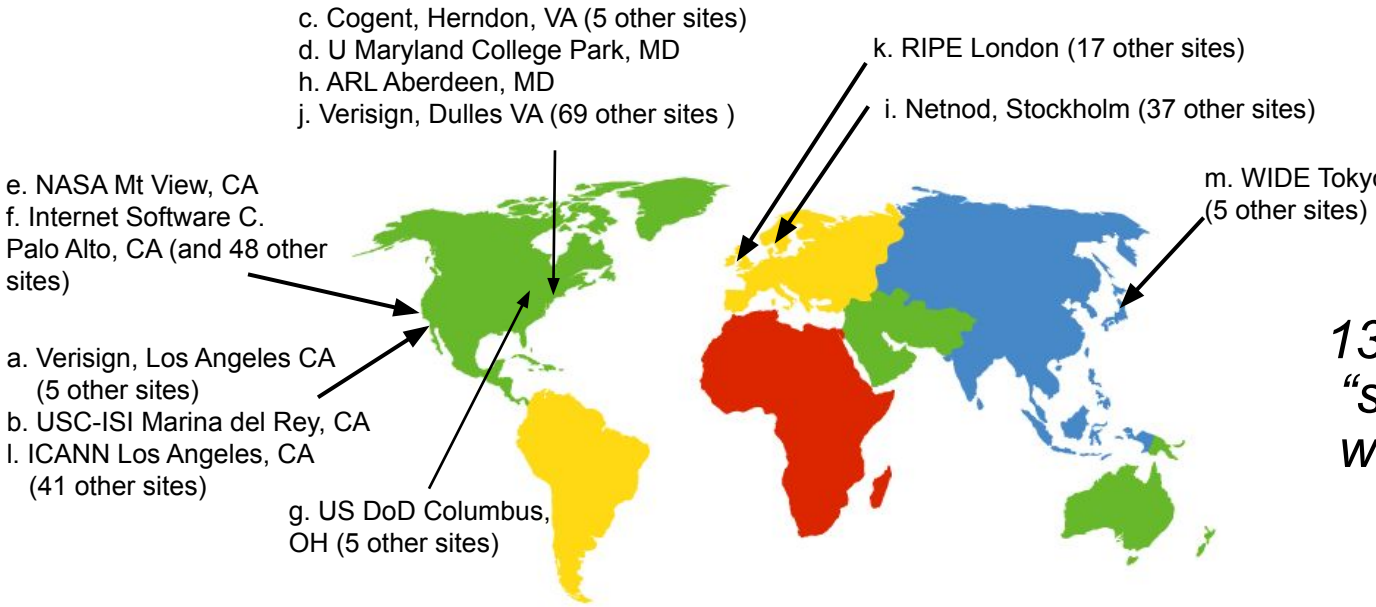
Label	Description
com	Commercial organizations
edu	Educational institutions
gov	Government institutions
int	International organizations
mil	Military groups
net	Network support centers
org	Nonprofit organizations
aero	Airlines and aerospace companies
biz	Businesses or firms (similar to com)
coop	Cooperative business organizations
info	Information service providers
museum	Museums and other nonprofit organizations
name	Personal names (individuals)
pro	Professional individual organizations

Country domains



DNS: root name servers

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



13 root name "servers" worldwide

TLD, authoritative servers

top-level domain (TLD) servers:

- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
- Network Solutions maintains servers for .com TLD
- Educause for .edu TLD

authoritative DNS servers:

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

Local DNS name server

- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

DNS: caching, updating records

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

DNS records

DNS: distributed 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`**

Attacking DNS

DDoS attacks

- Bombard root servers with traffic
  - Not successful to date
  - Traffic Filtering
  - Local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombard TLD servers
  - Potentially more dangerous

Redirect attacks

- Man-in-middle
  - Intercept queries
- DNS poisoning
  - Send bogus relies to DNS server, which caches

Exploit DNS for DDoS

- Send queries with spoofed source address: target IP
- Requires amplification

Chapter 2: outline

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

2.6 P2P applications

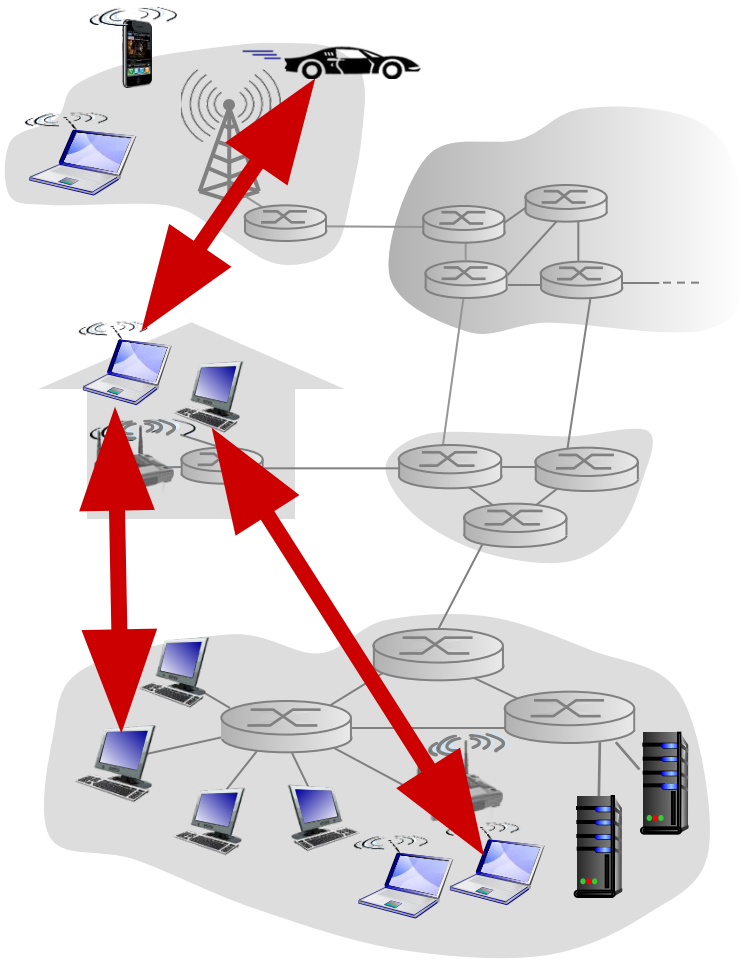
2.7 socket programming with UDP and TCP

Pure P2P architecture

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

examples:

- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)

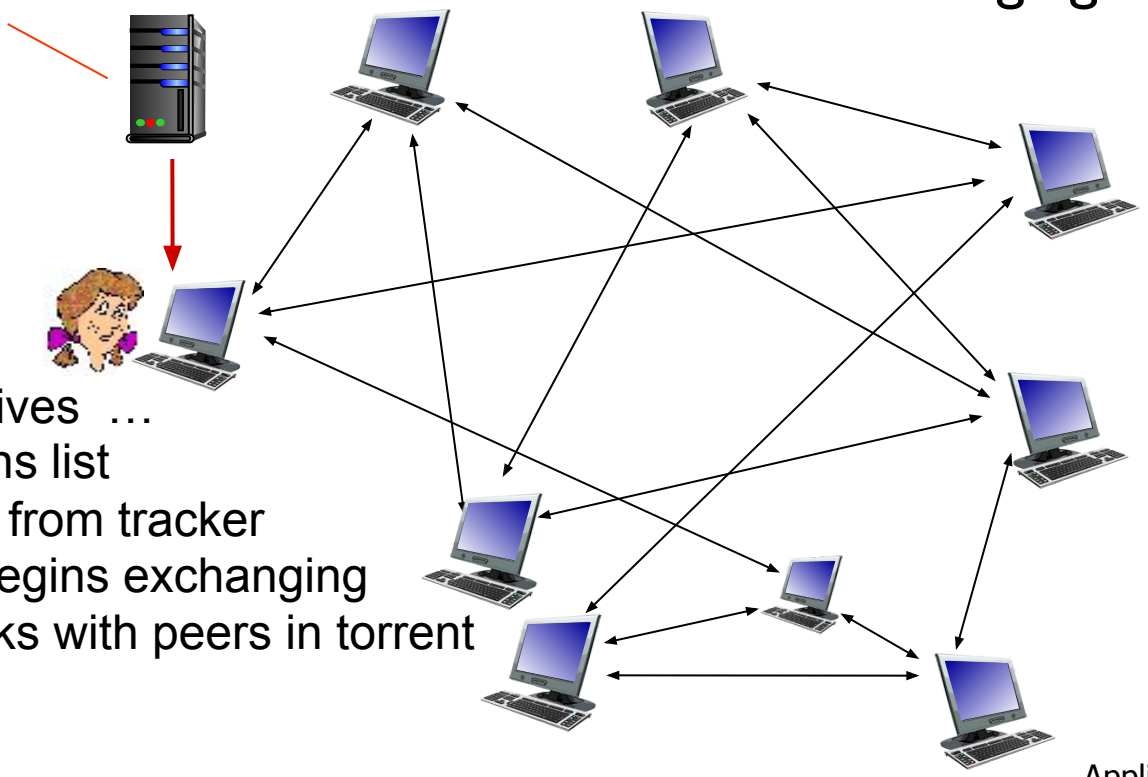


P2P file distribution: BitTorrent

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks

tracker: tracks peers participating in torrent

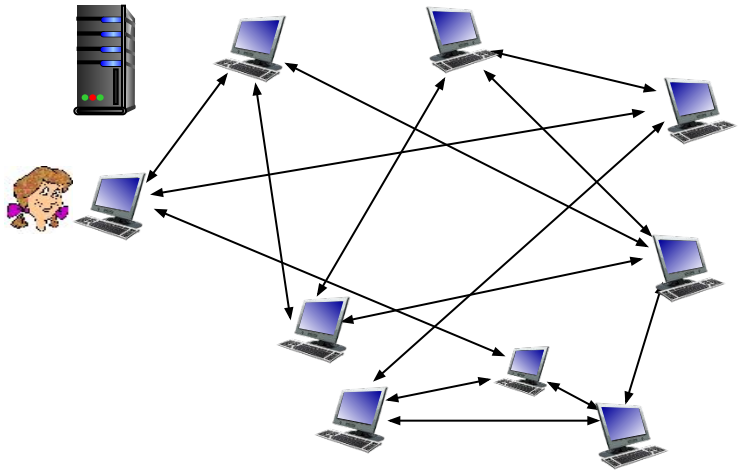
torrent: group of peers exchanging chunks of a file



Alice arrives …
… obtains list
of peers from tracker
… and begins exchanging
file chunks with peers in torrent

P2P file distribution: BitTorrent

- peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")



- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn:* peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

BitTorrent: requesting, sending file chunks

requesting chunks:

- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first

sending chunks: tit-for-tat

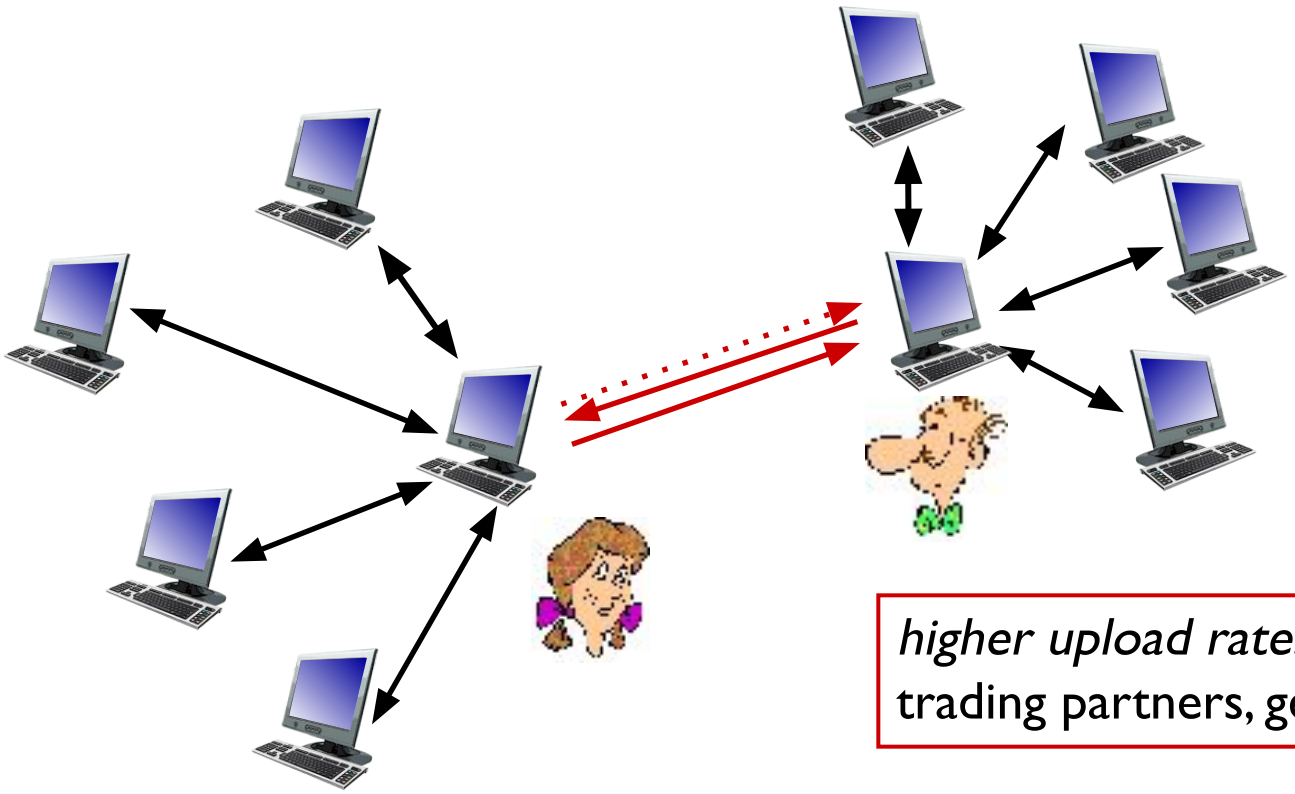
- Alice sends chunks to those four peers currently sending her chunks *at highest rate*
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "optimistically unchoke" this peer
  - newly chosen peer may join top 4

BitTorrent: tit-for-tat

(1) Alice "optimistically unchokes" Bob

(2) Alice becomes one of Bob's top-four providers; Bob reciprocates

(3) Bob becomes one of Alice's top-four providers



higher upload rate: find better trading partners, get file faster !

Distributed Hash Table (DHT)

- DHT: a *distributed P2P database*
- database has (key, value) pairs; examples:
  - key: ss number; value: human name
  - key: movie title; value: IP address
- Distribute the (key, value) pairs over the (millions of peers)
- a peer queries DHT with key
  - DHT returns values that match the key
- peers can also insert (key, value) pairs

Chapter 2: outline

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

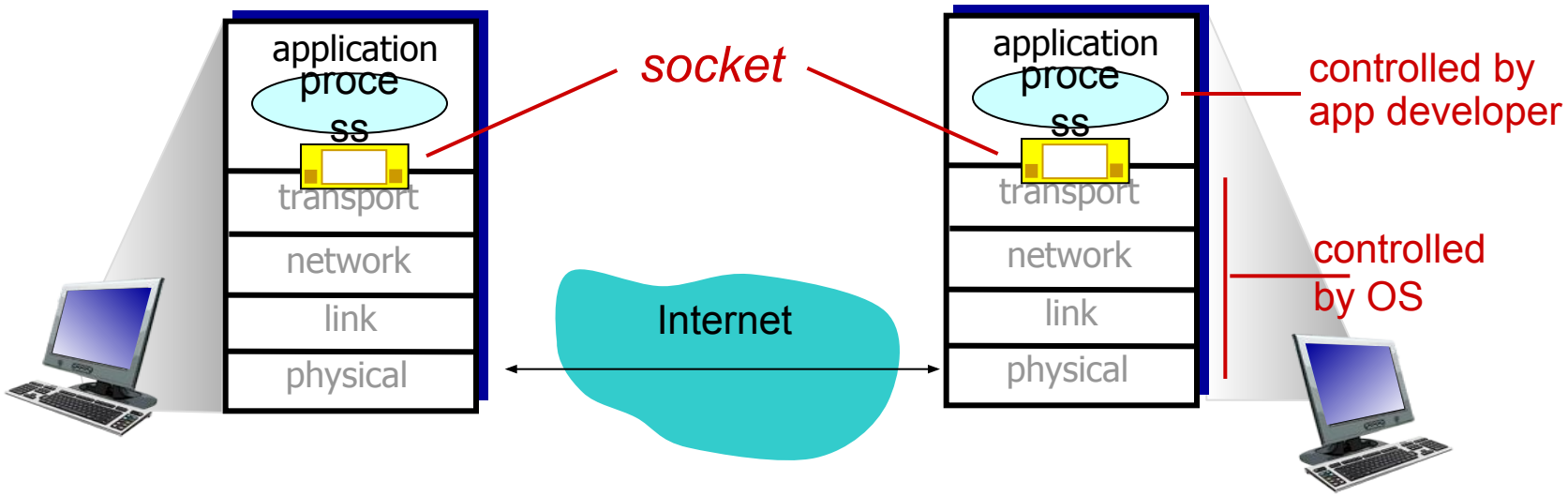
2.6 P2P applications

2.7 socket programming with UDP and TCP

Socket programming

goal: learn how to build client/server applications that communicate using sockets

socket: door between application process and end-end-transport protocol



Socket programming *with UDP*

UDP: no "connection" between client & server

- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- rcvr extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

- UDP provides *unreliable* transfer of groups of bytes ("datagrams") between client and server

Socket programming *with TCP*

client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

client contacts server by:

- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint:

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

Chapter 2: summary

our study of network apps now complete!

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP
- specific protocols:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent, DHT
- socket programming: TCP, UDP sockets

Chapter 2: summary

most importantly: learned about protocols!

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - headers: fields giving info about data
  - data: info being communicated

important themes:

- control vs. data msgs
  - in-band, out-of-band
- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable msg transfer
- "complexity at network edge"