Unit No:2

Application Layer

Marwadi University

Department of Computer Engineering

Computer Networks (3150710)

Dr. Sushil Kumar Singh
Associate Professor

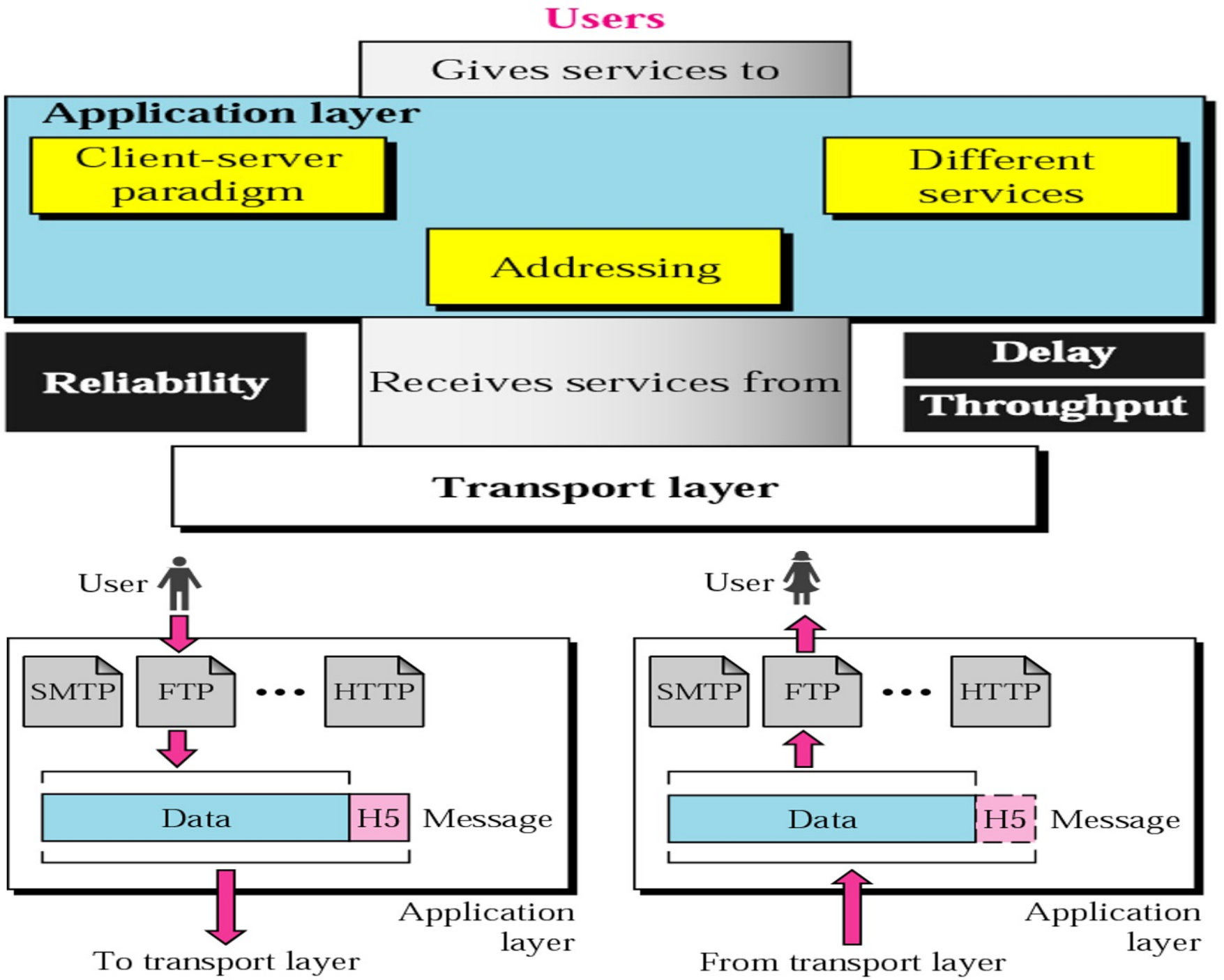
Syllabus

- 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 TCP
- 2.8 Socket programming with UDP

Our goals:

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

Position of Application Layer



Some Network Applications

- e-mail
- web
- instant messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video clips
- voice over IP
- real-time video conferencing
- grid computing
-
-
-

Creating a Network Apps

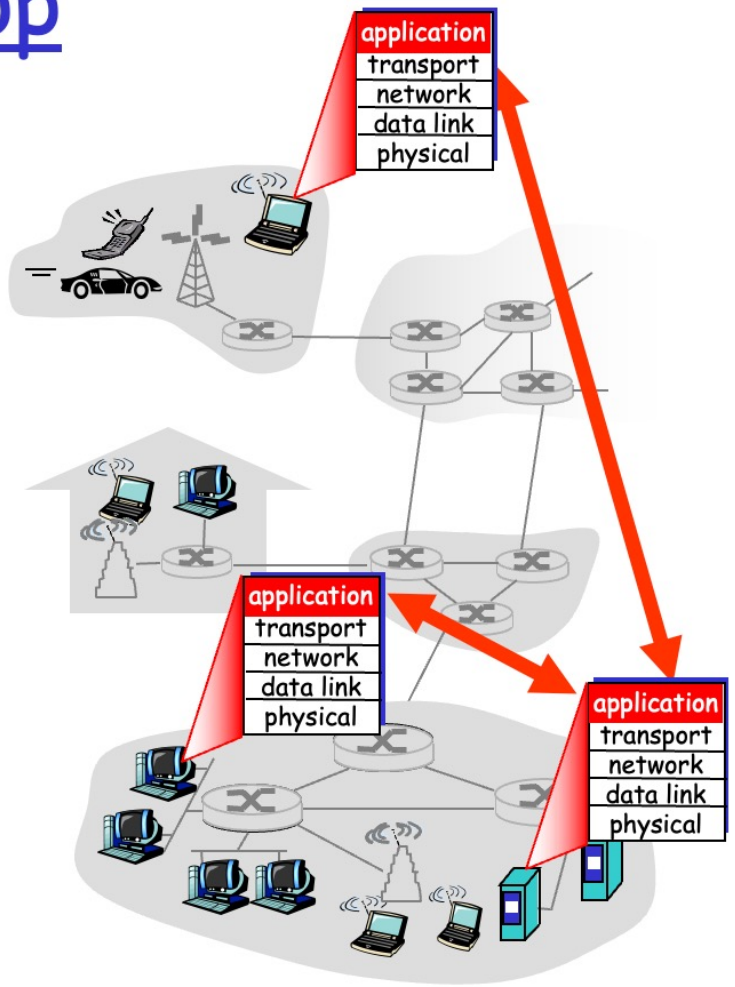
Creating a network app

write programs that

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

No need to write software for network-core devices

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



Application-Layer Paradigm

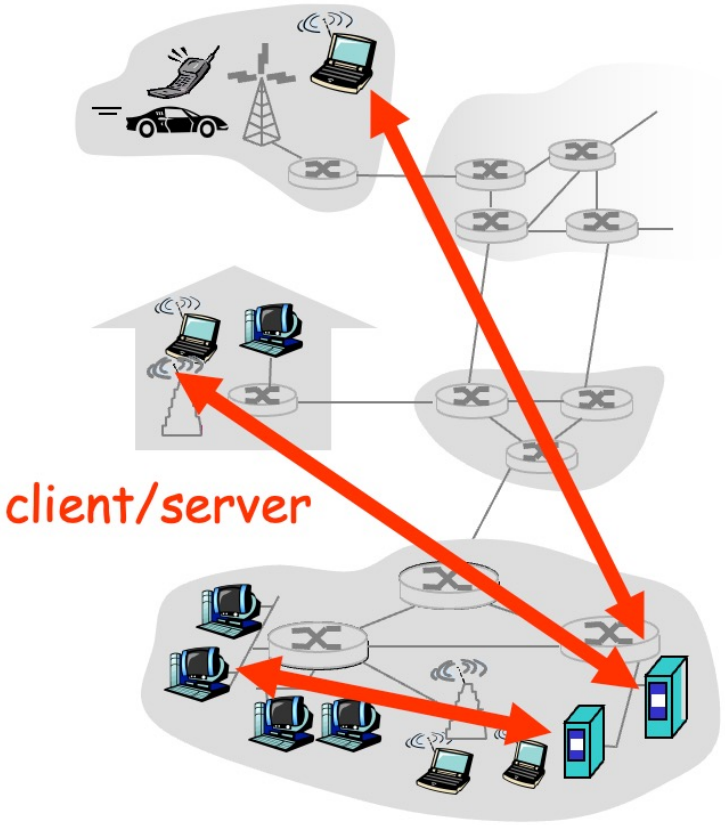
- It should be clear that to use the Internet we need two application programs to interact with each other:
  - one running on a computer somewhere in the world.
  - the other running on another computer somewhere else in the world.
- The two programs need to send messages to each other through the Internet infrastructure.
- However, we have not discussed what the relationship should be between these programs. Should both application programs be able to request services and provide services, or should the application programs just do one or the other?

Application Architecture

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P

Client Server Architecture

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

Figure : Example of a client-server paradigm

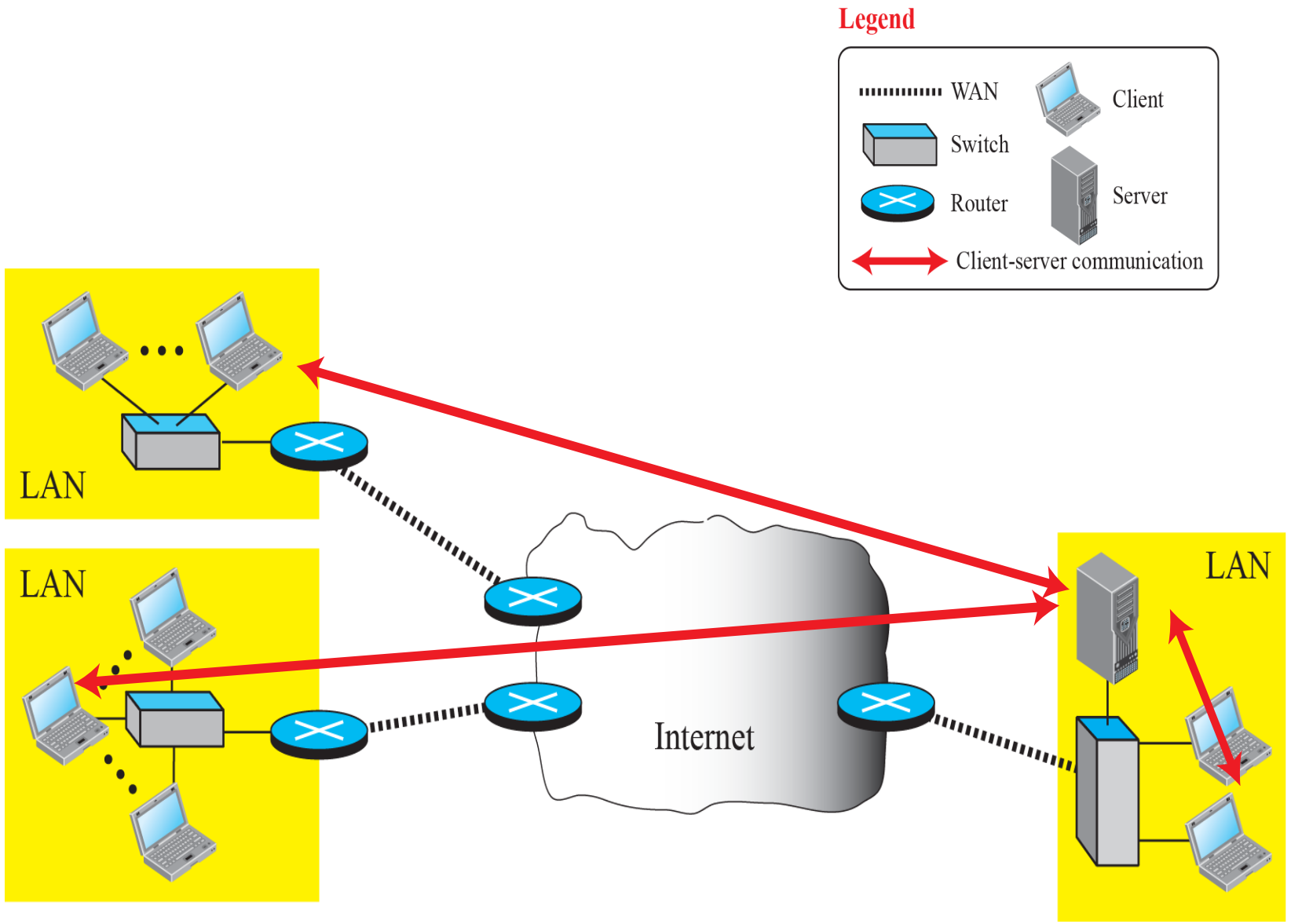


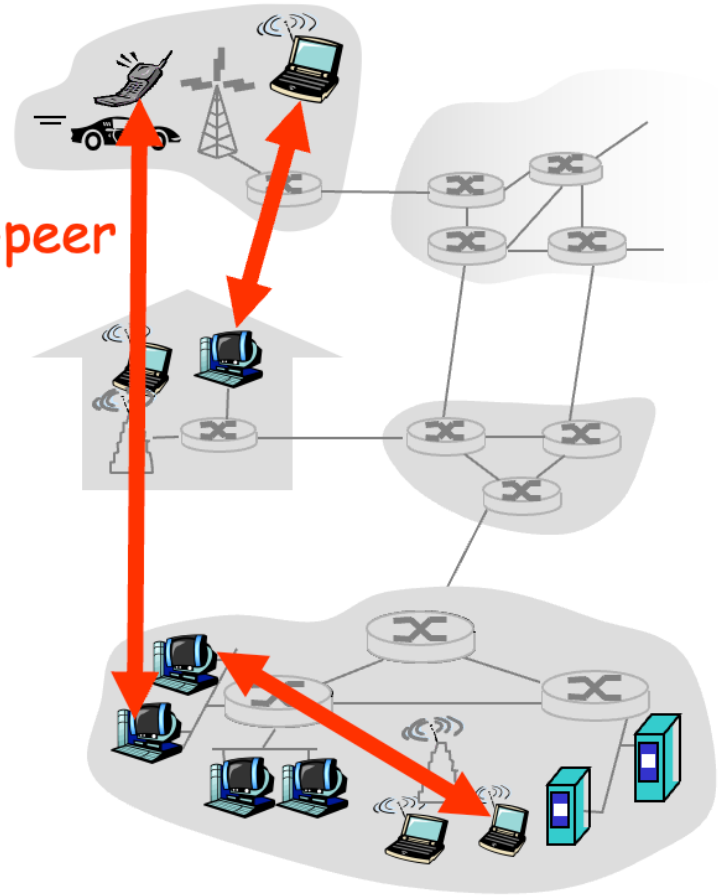
Figure : Example of a client-server paradigm

- In this paradigm, communication at the application layer is between two running application programs called processes: a client and a server.
- A client is a running program that initializes the communication by sending a request;
- a server is another application program that waits for a request from a client.

P2P Architecture

Pure P2P architecture

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



Highly scalable but difficult to manage

Hybrid of Client Server and P2P Architecture

Hybrid of client-server and P2P

Skype

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

Instant messaging

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

Processes Communicating

Processes communicating

Process: program running within a host.

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

Client process: process that initiates communication

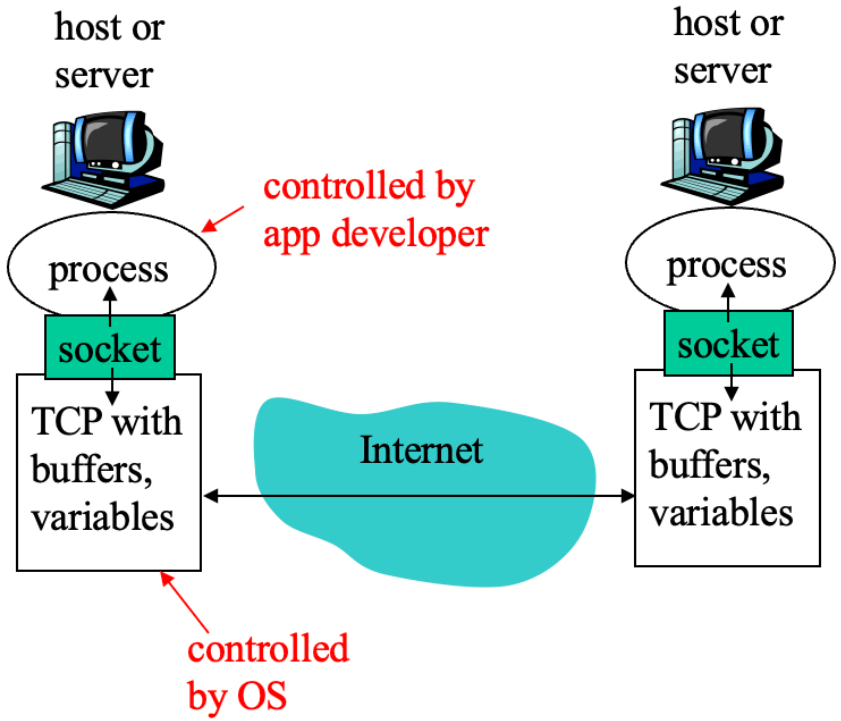
Server process: process that waits to be contacted

- Note: applications with P2P architectures have client processes & server processes

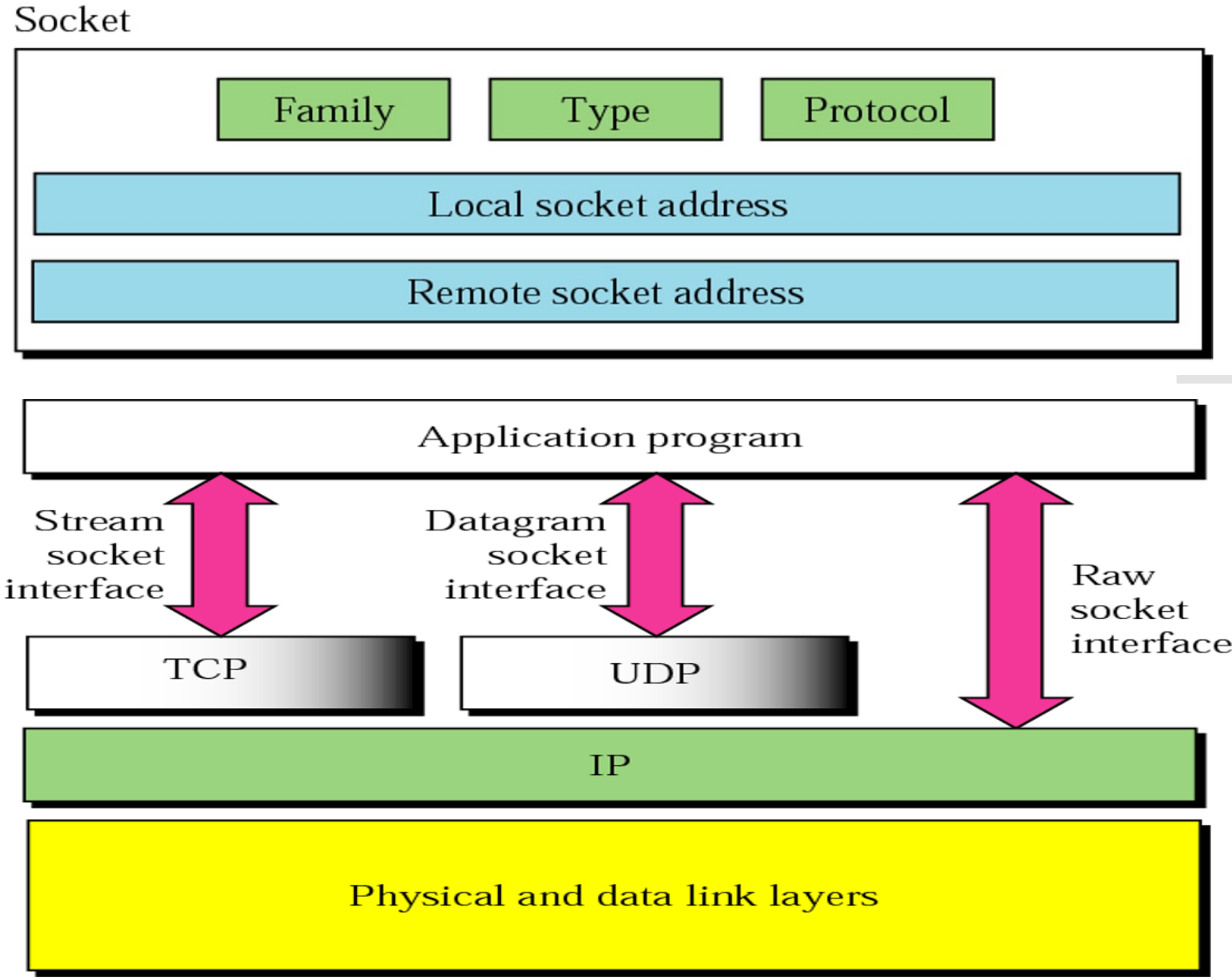
Sockets

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



Socket Structure and Types



Addressing Processes

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

App-Layer Protocol Defines

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

Public-domain protocols:

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

Proprietary protocols:

- e.g., Skype

Transport Service

What transport service does an app need?

Data loss

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

Timing

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

Throughput

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

Security

- Encryption, data integrity, …

Transport Service Requirement

Transport service requirements of common apps

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

Internet Transport Protocols Services

Internet transport protocols services

TCP service:

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

UDP service:

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

Q: why bother? Why is there a UDP?

World Wide Web and HTTP

- In this section, we first introduce the World Wide Web (abbreviated WWW or Web).
- We then discuss the Hyper Text Transfer Protocol (HTTP), the most common client-server application program used in relation to the Web.

World Wide Web and HTTP

First some jargon

- Web page consists of objects
- Object can be HTML file, JPEG image, Java applet, audio file,…
- Web page consists of base HTML-file which includes several referenced objects
- Each object is addressable by a URL
- Example URL:

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

host name | path name

Figure : Example (Retrieving two files and one image)

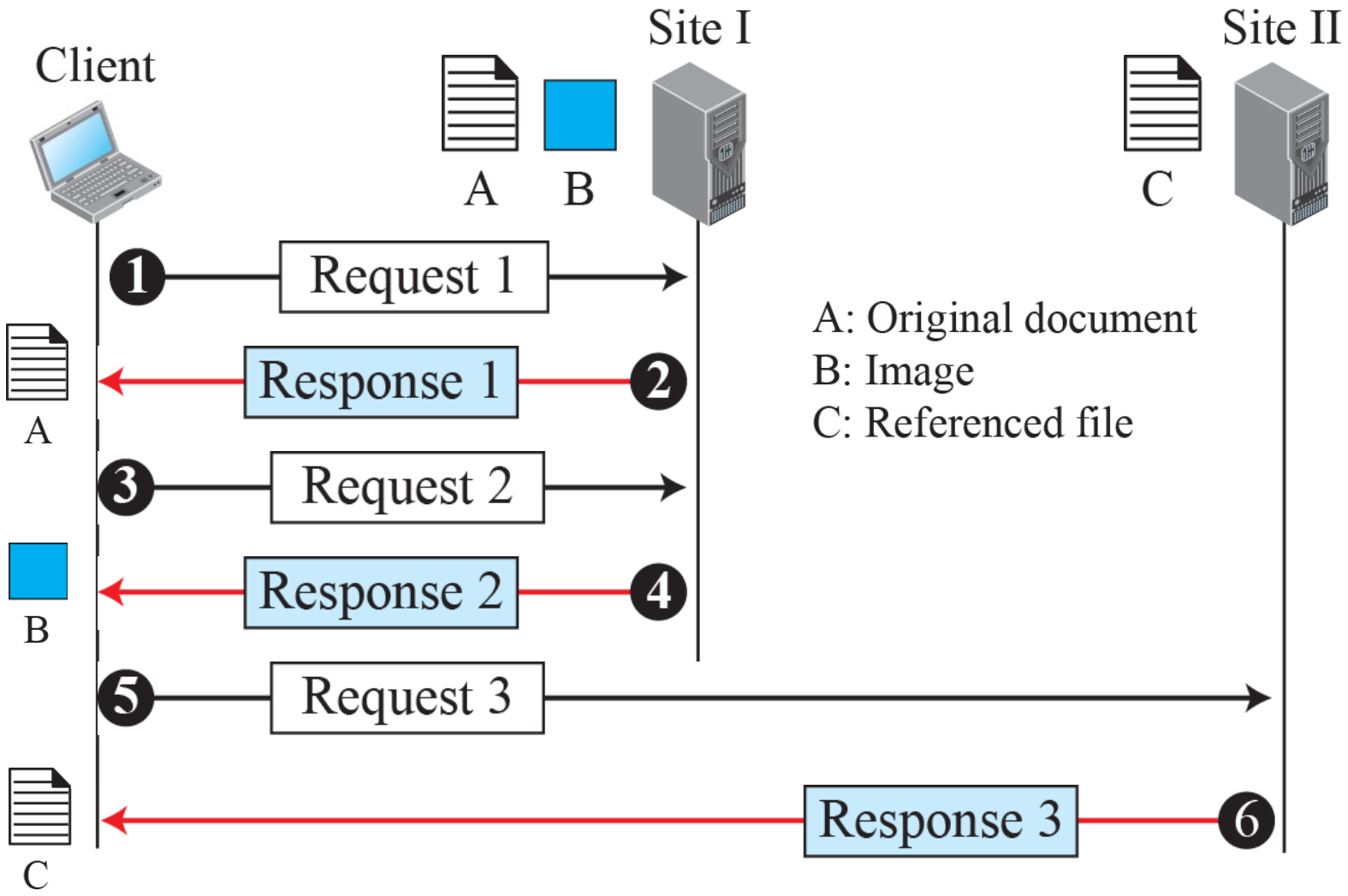
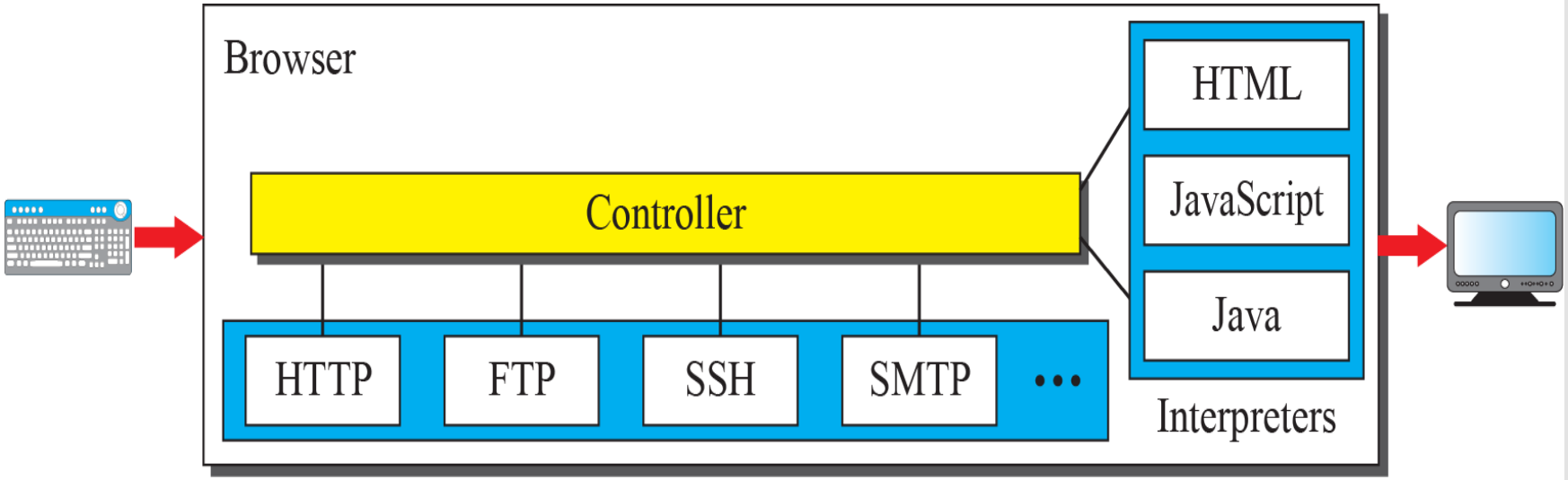


Figure : Browser:

- **Browser:** A web browser or Internet browser is a software application for retrieving, presenting, and traversing information resources on the World Wide Web.



Example

- The URL *http://www.mhhe.com/compsci/forouzan/* defines the web page related to one of the computer in the McGraw-Hill company
- (the three letters www are part of the host name and are added to the commercial host).
- The path is compsci / forouzan /, which defines Forouzan's web page under the directory compsci (computer science).

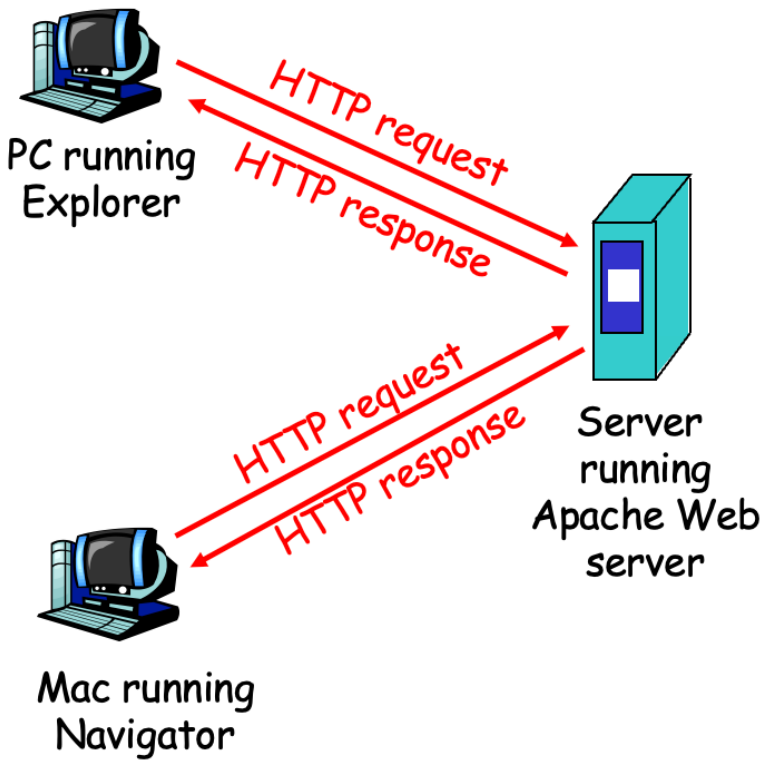
HTTP

(Overview)

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests



HTTP

(Overview)

HTTP overview (continued)

Uses TCP:

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

HTTP is "stateless"

- server maintains no information about past client requests

aside

Protocols that maintain "state" are complex!

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

HTTP

(Connections)

HTTP connections

Nonpersistent HTTP

- At most one object is sent over a TCP connection.
- *downloading multiple objects required multiple connections*

Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server.

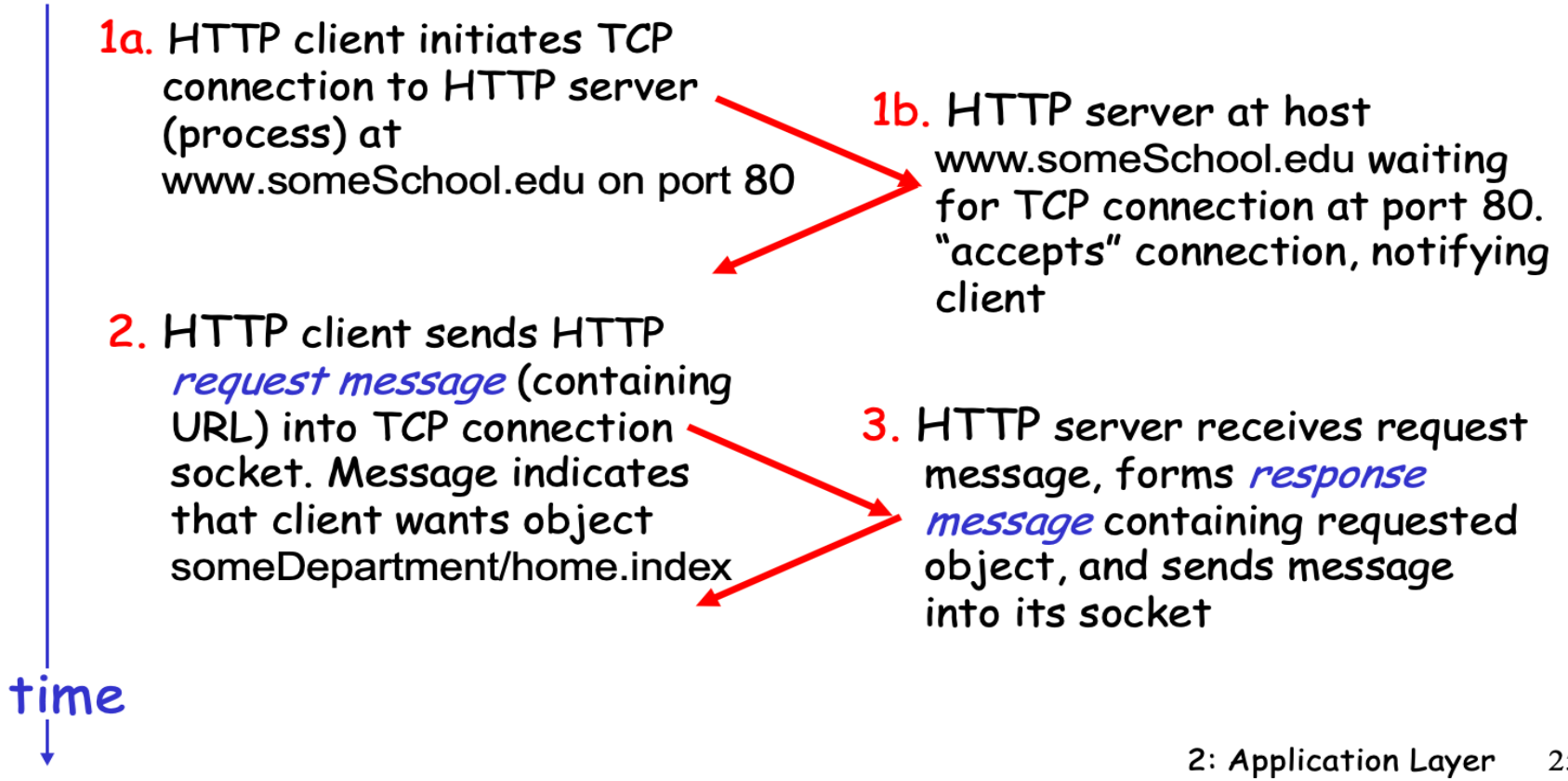
Nonpersistent HTTP

Nonpersistent HTTP

Suppose user enters URL

`www.someSchool.edu/someDepartment/home.index`

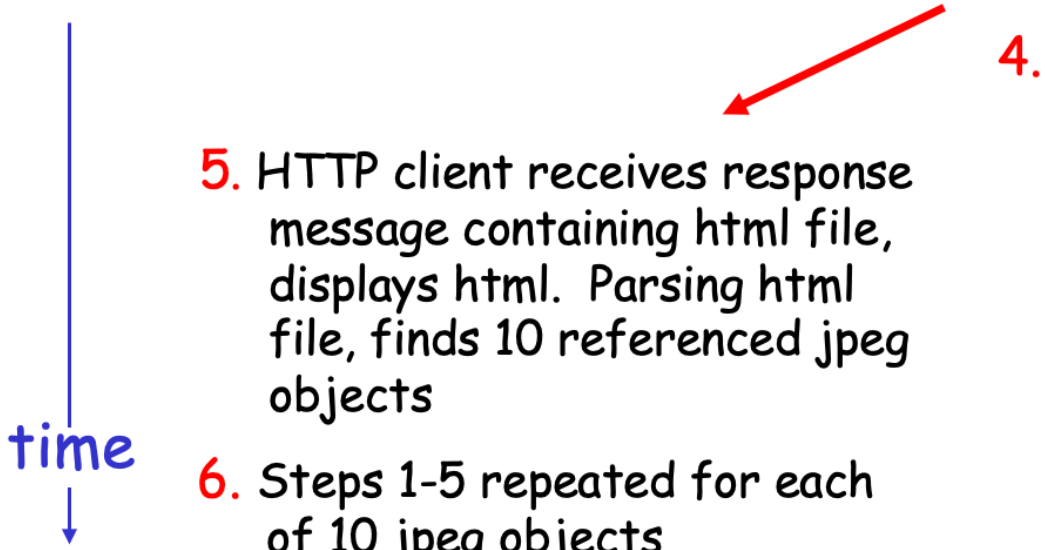
(contains text, references to 10 jpeg images)



Nonpersistent HTTP

(Continue)

Nonpersistent HTTP (cont.)

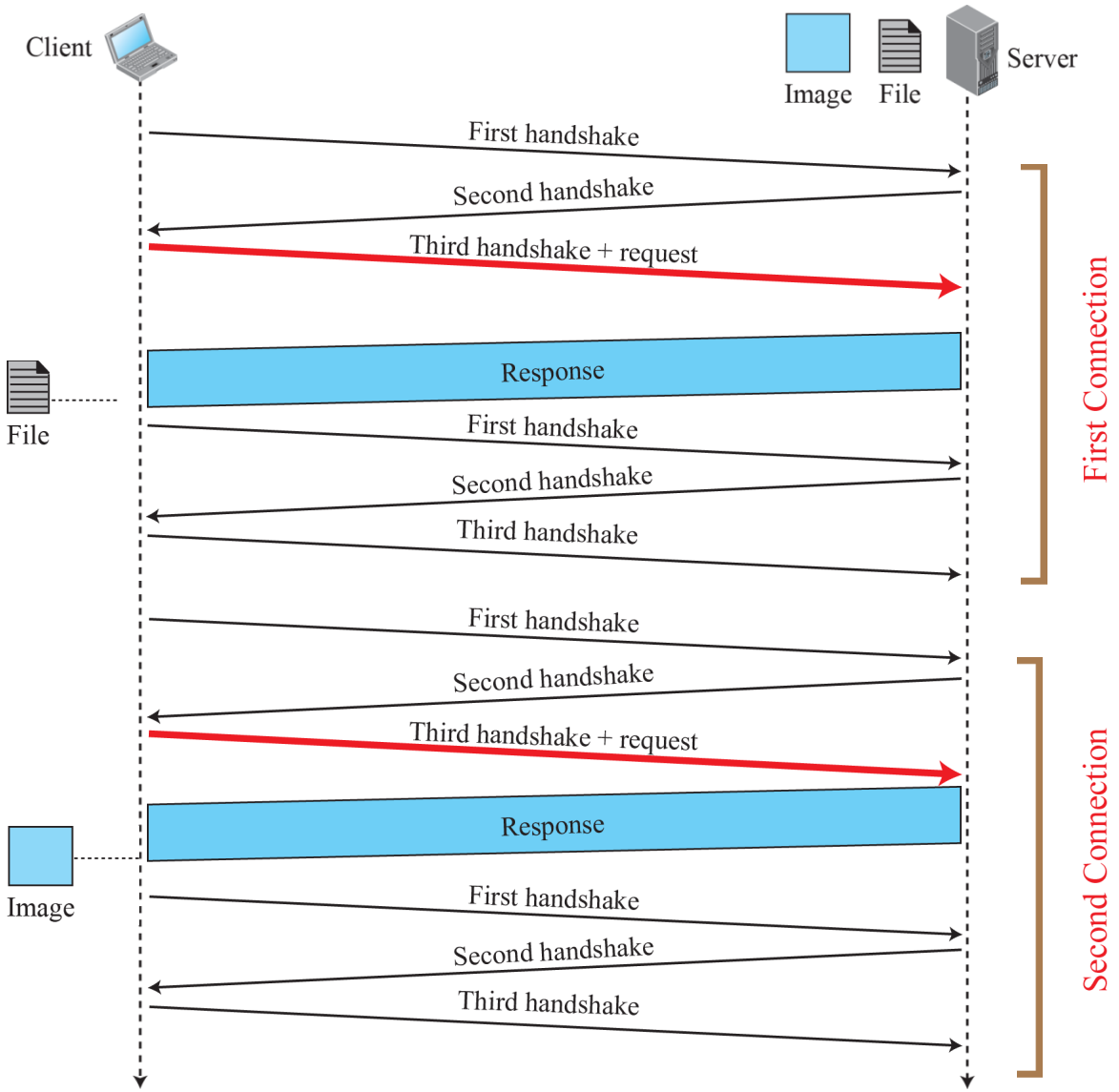


4. HTTP server closes TCP connection.

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

Figure : Non Persistent connection Example



Nonpersistent HTTP

(Response Time)

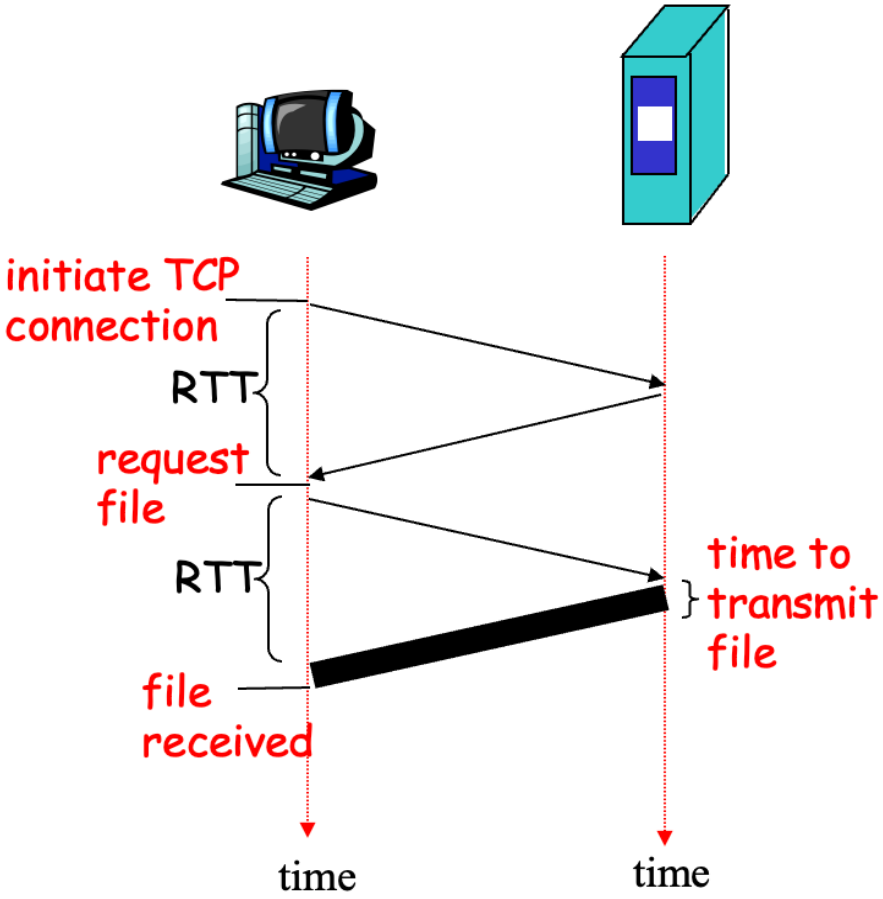
Non-Persistent HTTP: Response time

Definition of RTT: time for a small packet to travel from client to server and back.

Response time:

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

total = 2RTT+transmit time



Persistent HTTP

Persistent HTTP

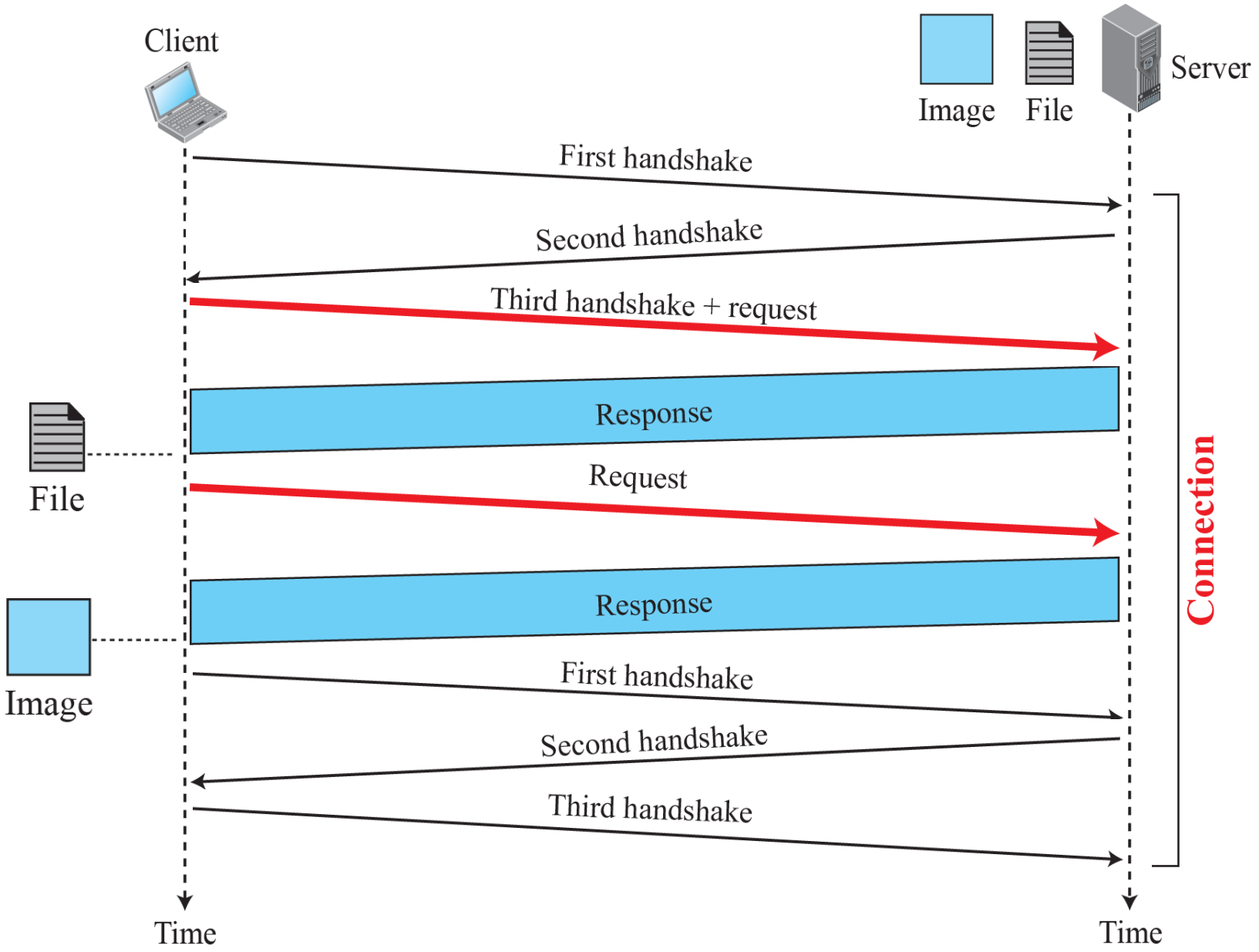
Nonpersistent HTTP issues:

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

Persistent HTTP

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

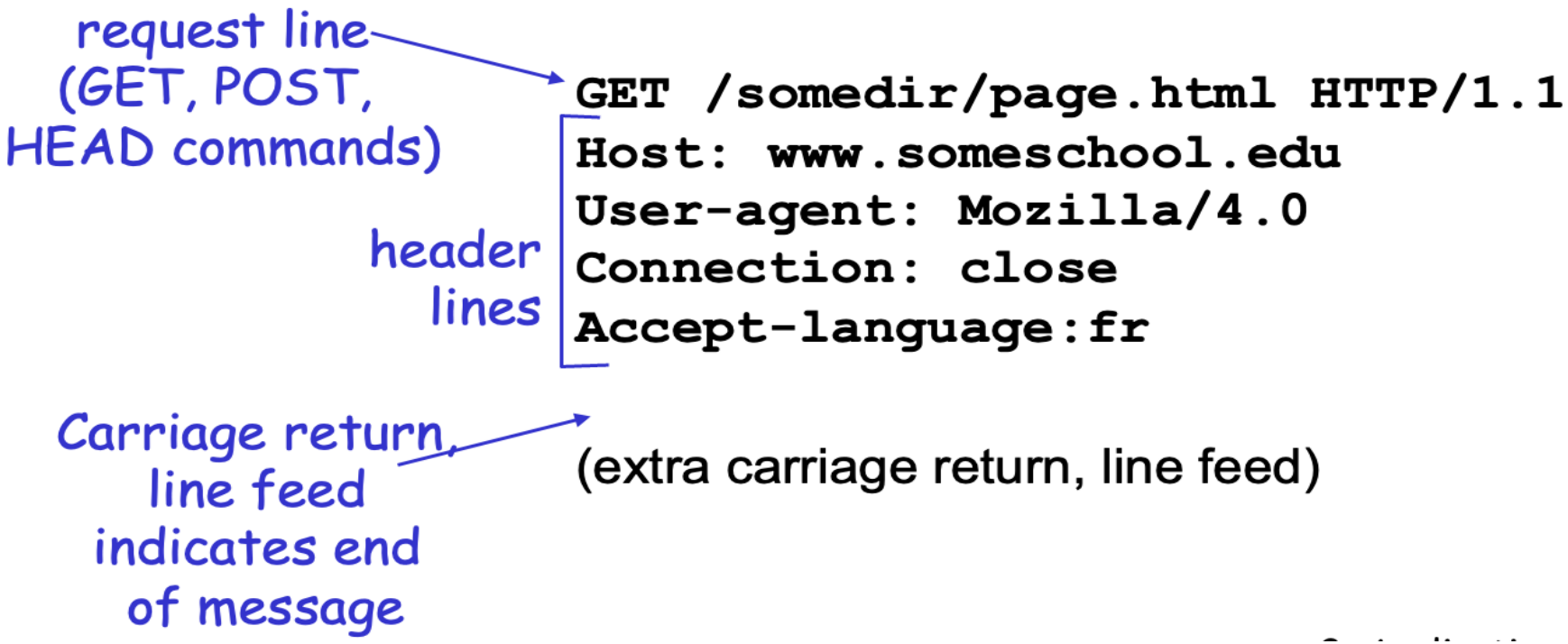
Figure : Persistent Connection Example



HTTP Request Message

HTTP request message

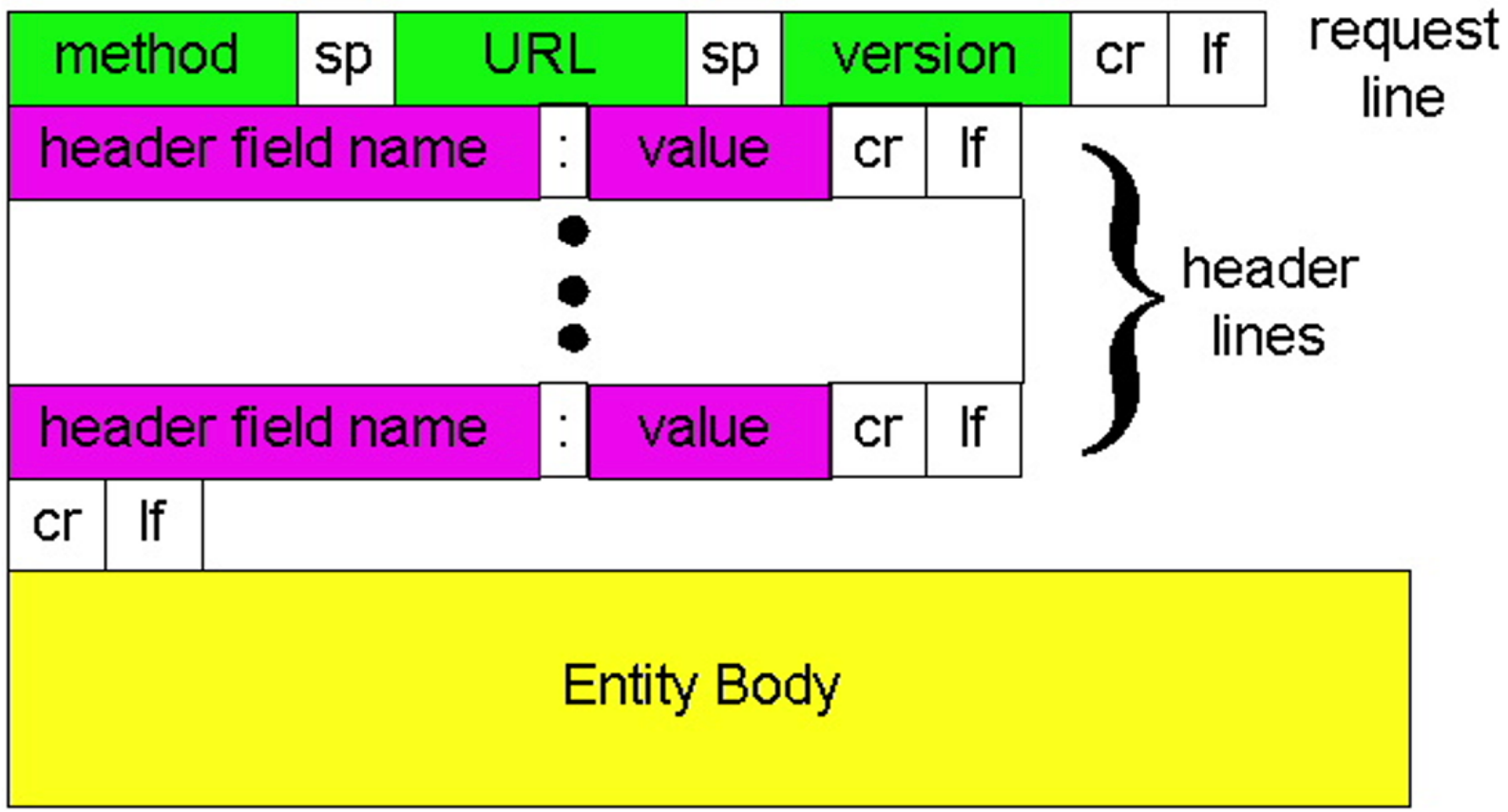
- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)



HTTP Request Message

(General Format)

HTTP request message: general format



Uploading Form Inputs

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
```

Method Types

Method types

HTTP/1.0

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

HTTP/1.1

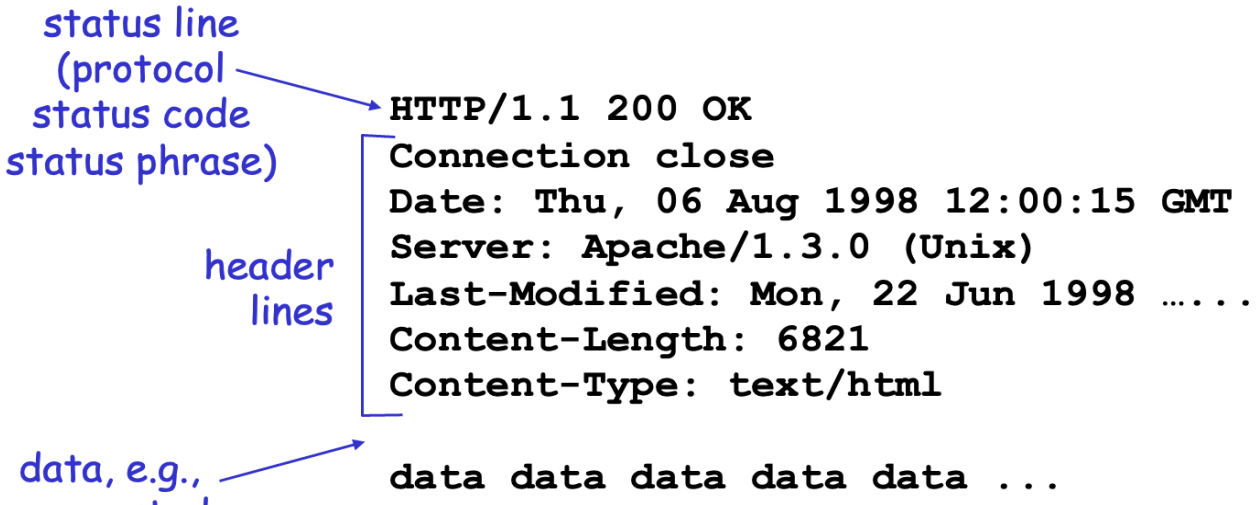
- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field

HTTP/1.1

- DELETE
  - deletes file specified in the URL field
- TRACE
  - Echoes request msg from server
- OPTIONS
  - Returns HTTP methods that the server supports
- CONNECT
  - TCP/IP tunnel for HTTP

HTTP response message

HTTP Response Message



In first line in server->client response message.

A few sample codes:

200 OK

- request succeeded, requested object later in this message

301 Moved Permanently

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

400 Bad Request

- request message not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

User Server State: Cookies

User-server state: cookies

Many major Web sites use cookies

Four components:

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

Example:

- Susan always access Internet always 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

(Continue)

Cookies (continued)

What cookies can bring:

- 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":

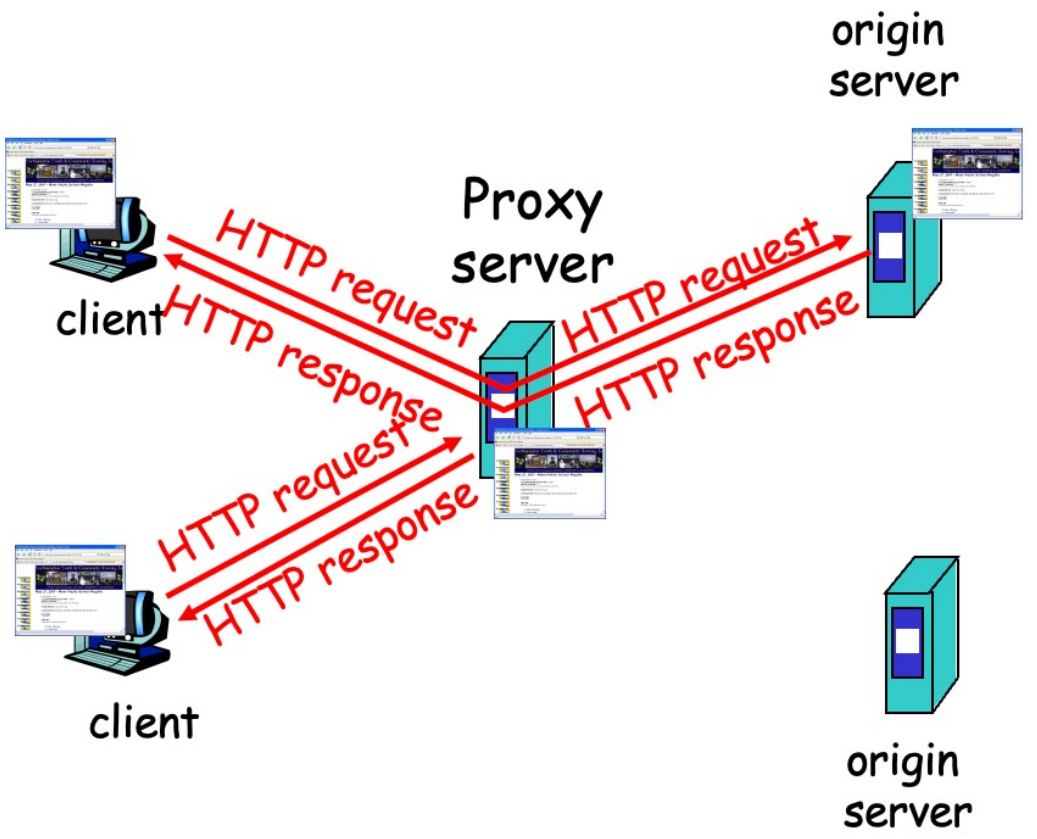
- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

Web Caches (Proxy Server)

Web caches (proxy server)

Goal: satisfy client request without involving origin server

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



Web Caches
(Proxy Server)

(Continue)

More about Web caching

- cache acts as both client and server
- typically cache is installed by ISP (university, company, residential ISP)

Why Web caching?

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

Caching Example

Caching example

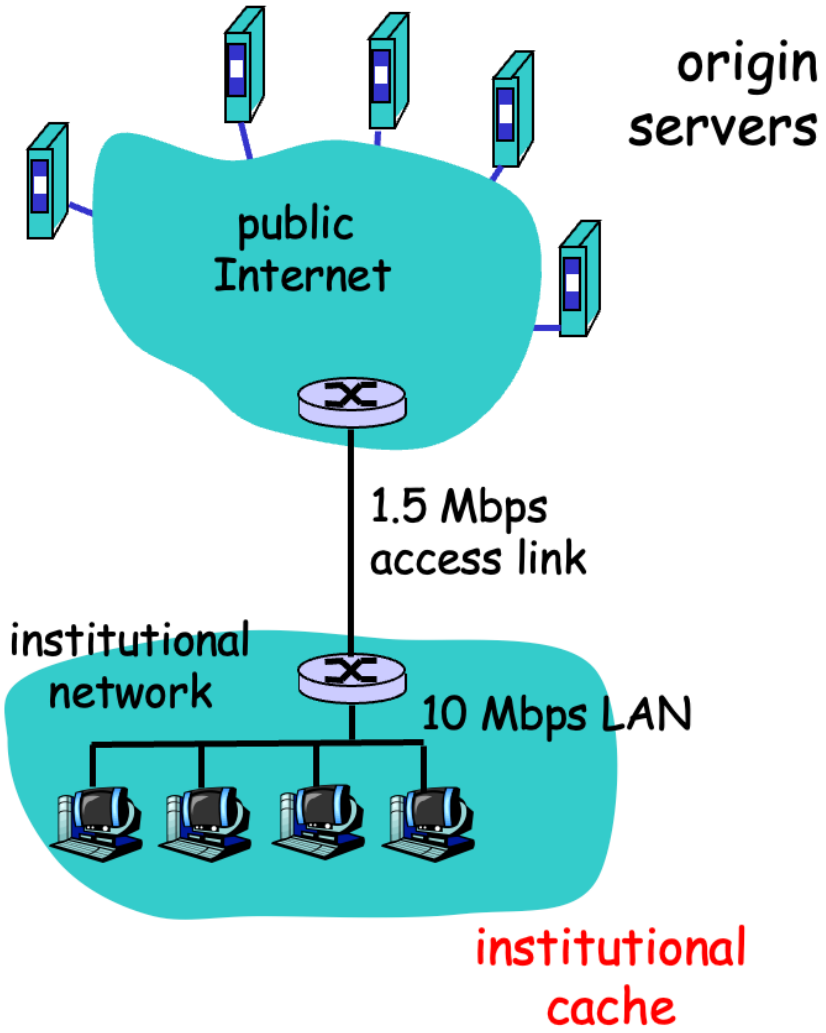
Assumptions

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

Consequences

- utilization on LAN = 15%
- utilization on access link = 100%
- total delay = Internet delay + access delay + LAN delay

  = 2 sec + minutes + milliseconds



Caching Example

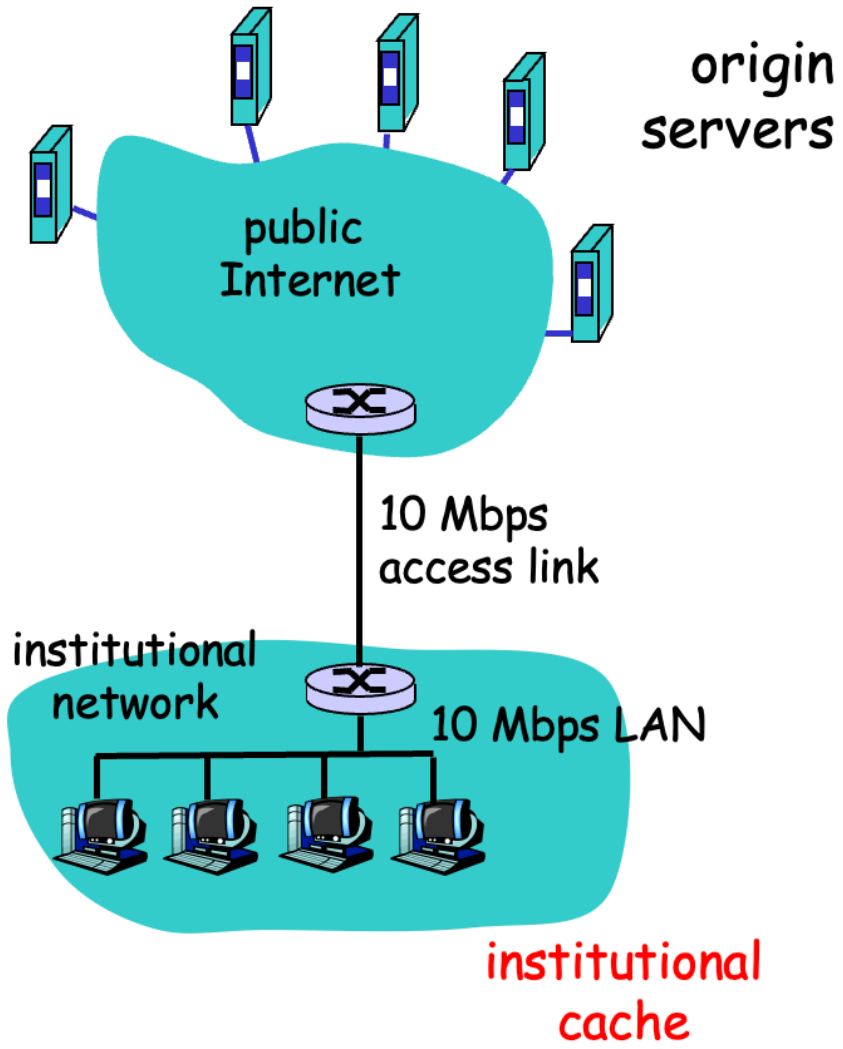
Caching example (cont)

possible solution

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

consequence

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



Caching Example

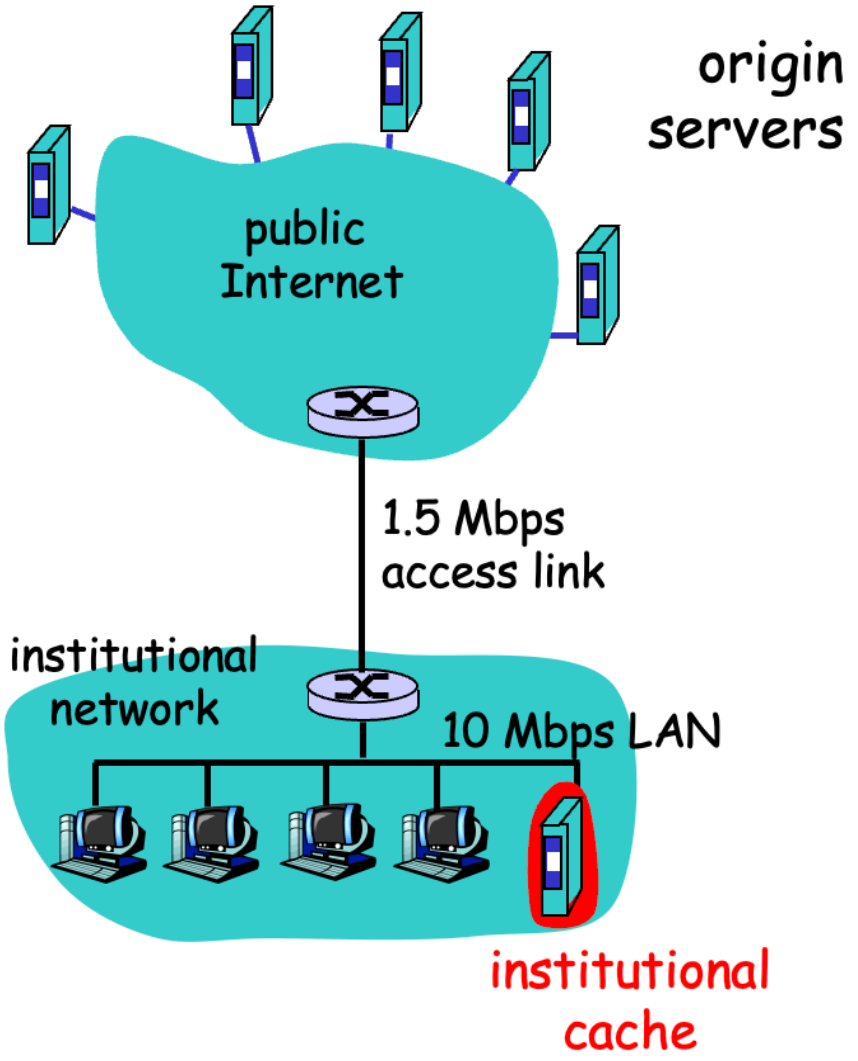
Caching example (cont)

possible solution: install cache

- suppose hit rate is 0.4

consequence

- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total avg delay = Internet delay + access delay + LAN delay = .6*(2.01) secs + .4*milliseconds < 1.4 secs



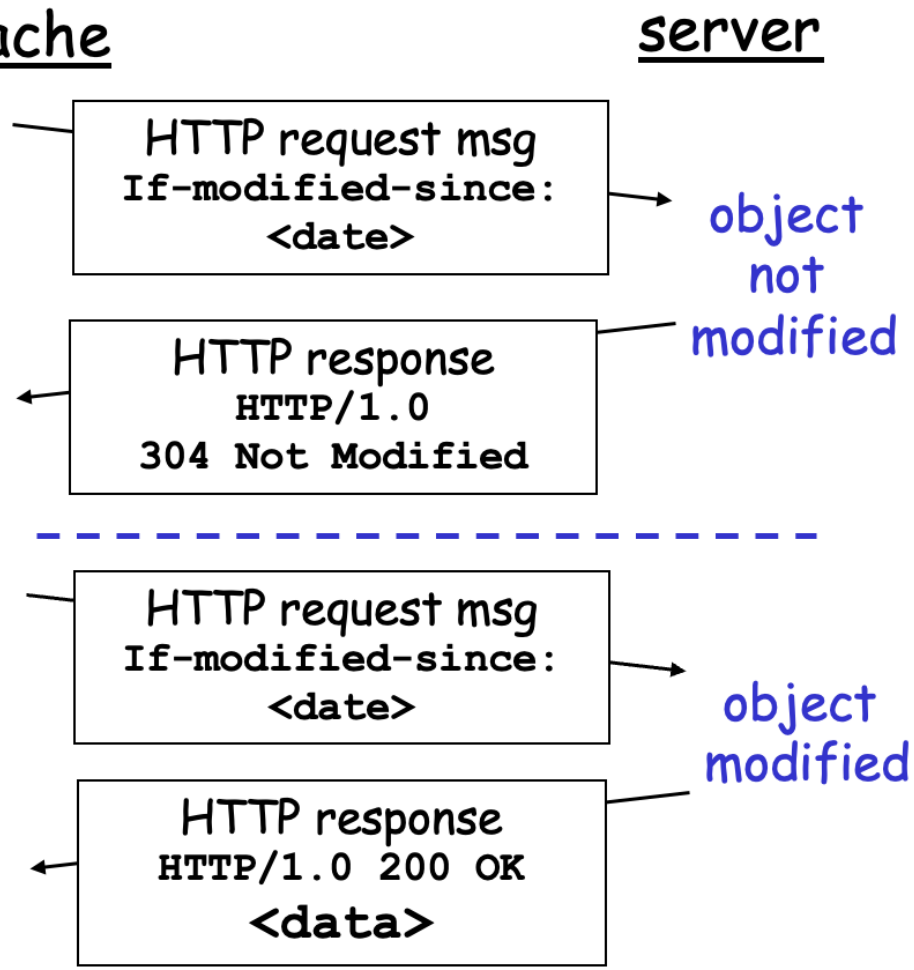
Conditional GET

Conditional GET

- Goal: don't send object if cache has up-to-date cached version
- cache: specify date of cached copy in HTTP request

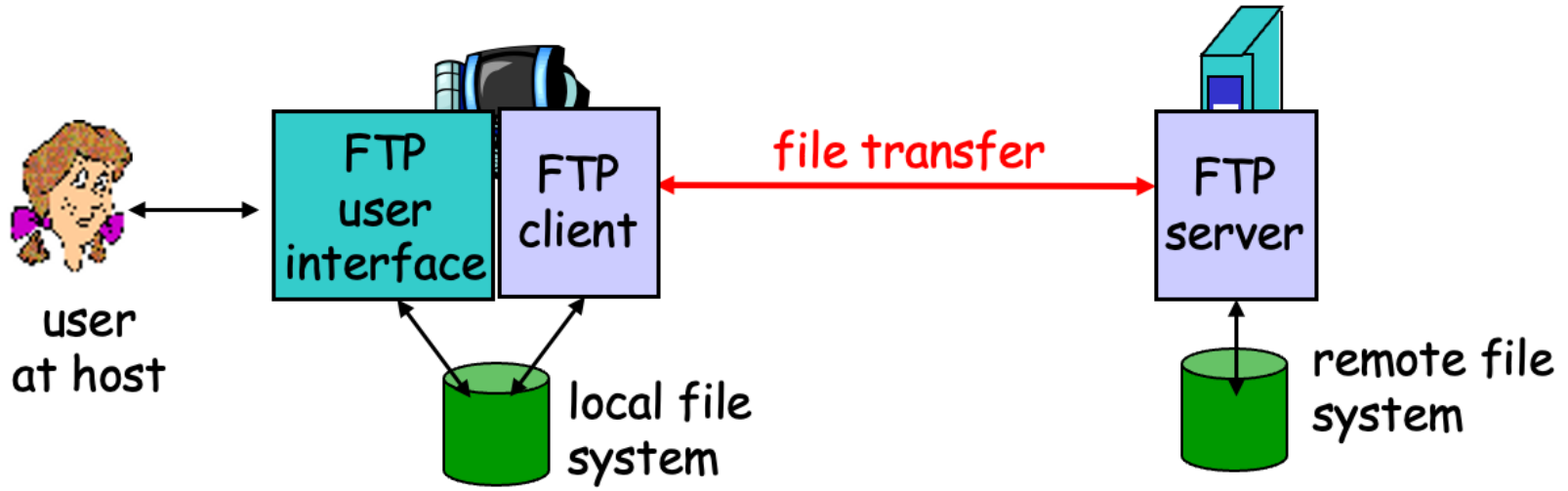
  `If-modified-since: <date>`
- server: response contains no object if cached copy is up-to-date:

  `HTTP/1.0 304 Not Modified`



FTP: the file transfer protocol

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

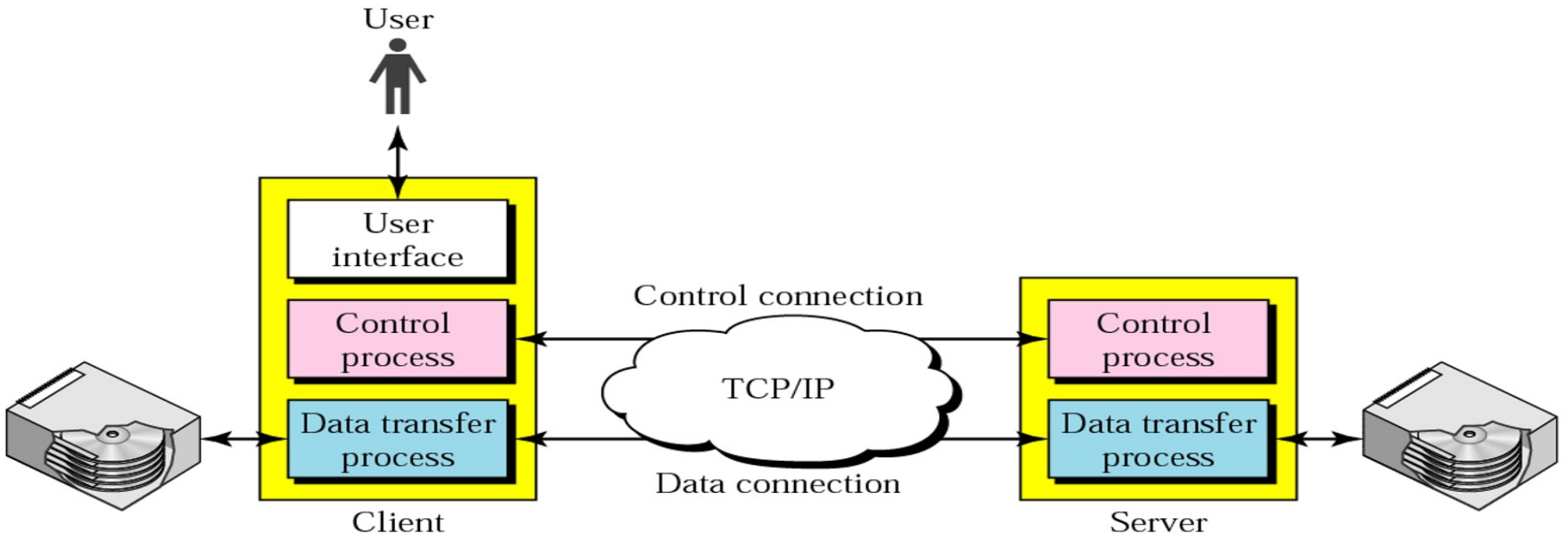
FTP (File Transfer Protocol)

- File Transfer Protocol (FTP) is the standard protocol provided by TCP/IP for copying a file from one host to another.
  - Although transferring files from one system to another seems simple and straightforward, some problems must be dealt with first.
  - For example, two systems may use different file name conventions. Two systems may have different ways to represent data. All of these problems have been solved by FTP in a very simple and elegant approach.

FTP: the file transfer protocol

Note:

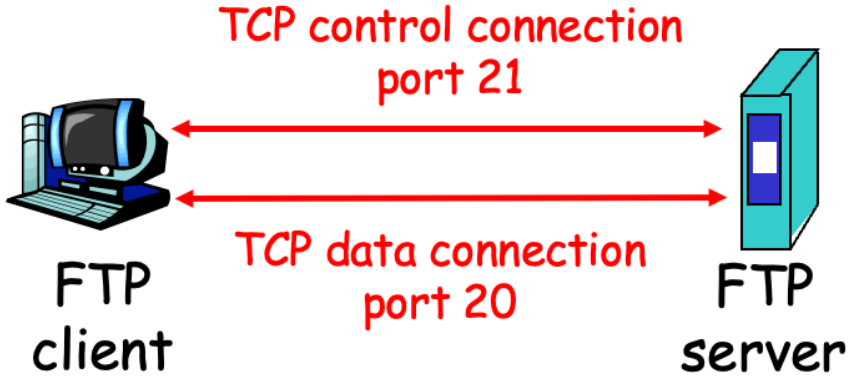
FTP uses the services of TCP. It needs two TCP connections. The well-known port 21 is used for the control connection, and the well-known port 20 is used for the data connection.



FTP: the file transfer protocol

FTP: separate control, data connections

- FTP client contacts FTP server at port 21, TCP is transport protocol
- client authorized over control connection
- client browses remote directory by sending commands over control connection.
- when server receives file transfer command, server opens 2^{nd} TCP connection (for file) to client
- after transferring one file, server closes data connection.



- server opens another TCP data connection to transfer another file.
- control connection: "out of band"
- FTP server maintains "state": current directory, earlier authentication

FTP: the file transfer protocol

(Commands and Responses)

FTP commands, responses

Sample commands:

- sent as ASCII text over control channel
- **USER *username***
- **PASS *password***
- **LIST** return list of file in current directory
- **RETR filename** retrieves (gets) file
- **STOR filename** stores (puts) file onto remote host

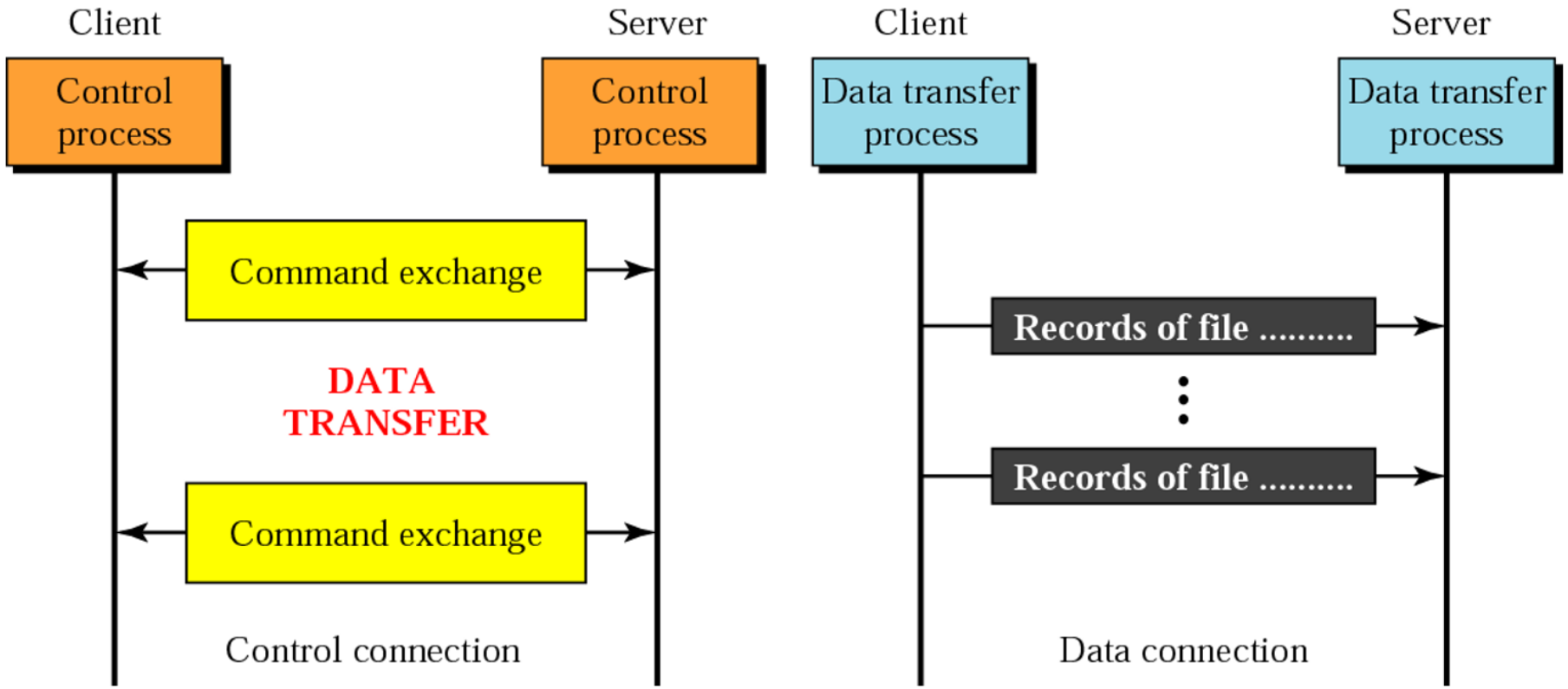
Sample return codes

- status code and phrase (as in HTTP)
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

FTP: the file transfer protocol

(Connections)

- Lifetimes of Two Connections
  - Control Connection
  - Data Connection
- Communication over Data Connection
- File Transfer



Some FTP File Transfer Protocol commands

Command	*Argument(s)*	*Description*
ABOR		Abort the previous command
CDUP		Change to parent directory
CWD	Directory name	Change to another directory
DELE	File name	Delete a file
LIST	Directory name	List subdirectories or files
MKD	Directory name	Create a new directory
PASS	User password	Password
PASV		Server chooses a port
PORT	port identifier	Client chooses a port
PWD		Display name of current directory
QUIT		Log out of the system
RETR	File name(s)	Retrieve files; files are transferred from server to client
RMD	Directory name	Delete a directory
RNFR	File name (old)	Identify a file to be renamed

FTP (File Transfer Protocol) Port Number - 21

Some FTP File Transfer Protocol commands

(continued)

Command	*Argument(s)*	*Description*
RNTO	File name (new)	Rename the file
STOR	File name(s)	Store files; file(s) are transferred from client to server
STRU	**F**, **R**, or **P**	Define data organization (**F**: file, **R**: record, or **P**: page)
TYPE	**A**, **E**, **I**	Default file type (**A**: ASCII, **E**: EBCDIC, **I**: image)
USER	User ID	User information
MODE	**S**, **B**, or **C**	Define transmission mode (**S**: stream, **B**: block, or **C**: compressed

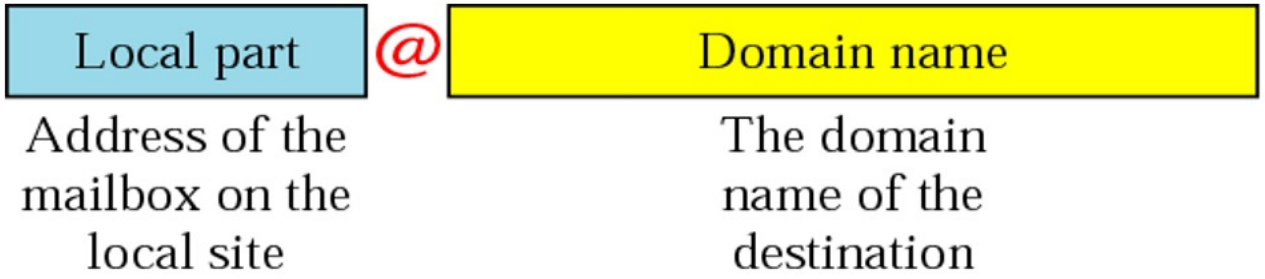
Some responses in FTP

Code	*Description*	*Code*	*Description*
125	Data connection open	**250**	Request file action OK
150	File status OK	**331**	User name OK; password is needed
200	Command OK	**425**	Cannot open data connection
220	Service ready	**450**	File action not taken; file not available
221	Service closing	**452**	Action aborted; insufficient storage
225	Data connection open	**500**	Syntax error; unrecognized command
226	Closing data connection	**501**	Syntax error in parameters or arguments
230	User login OK	**530**	User not logged in

Electronic Mail

(E-mail)

- Electronic mail (or e-mail) allows users to exchange messages.
- The nature of this application, however, is different from other applications discussed so far. In an application such as HTTP or FTP, the server program is running all the time, waiting for a request from a client.
- When the request arrives, the server provides the service. In the case of electronic mail, the situation is different.



Electronic Mail

(E-mail)

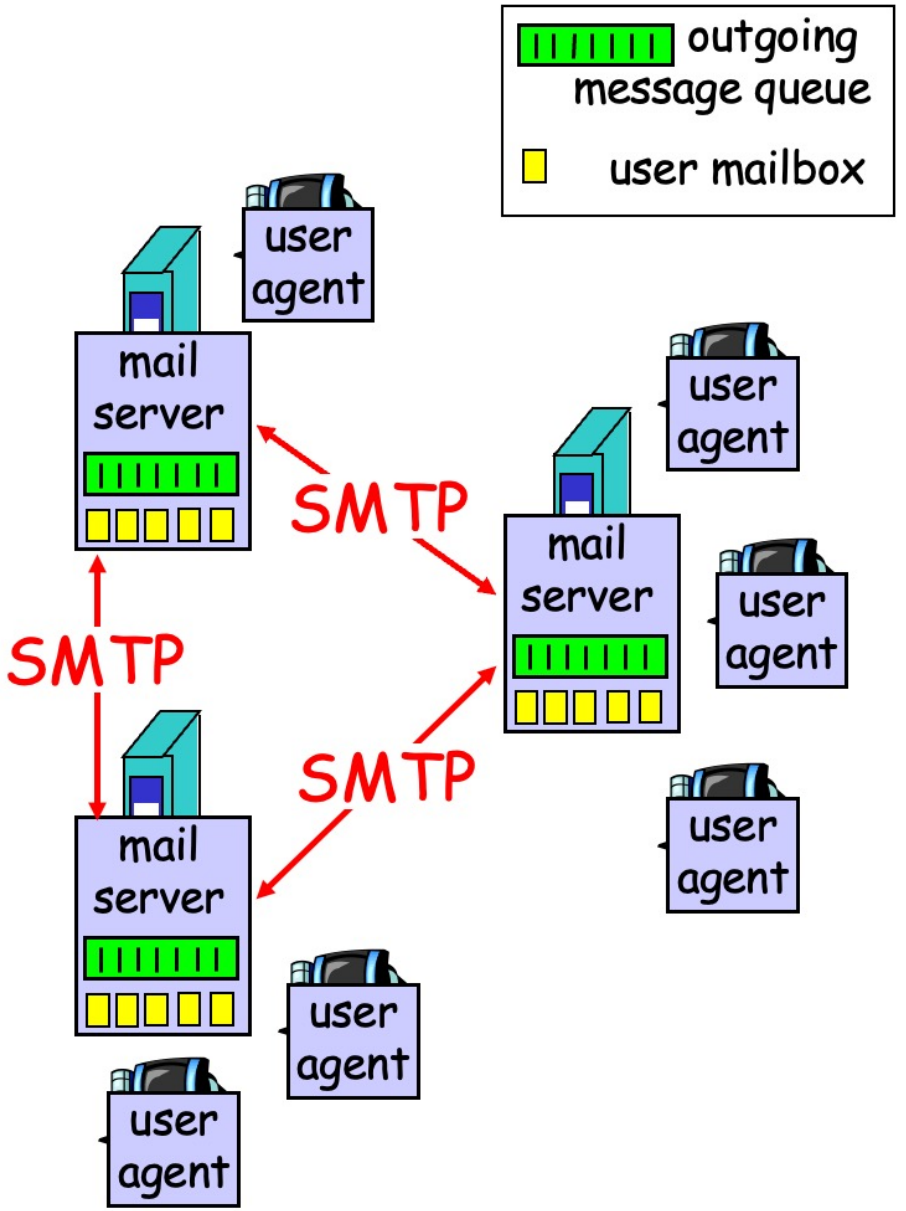
Electronic Mail

Three major components:

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

User Agent

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



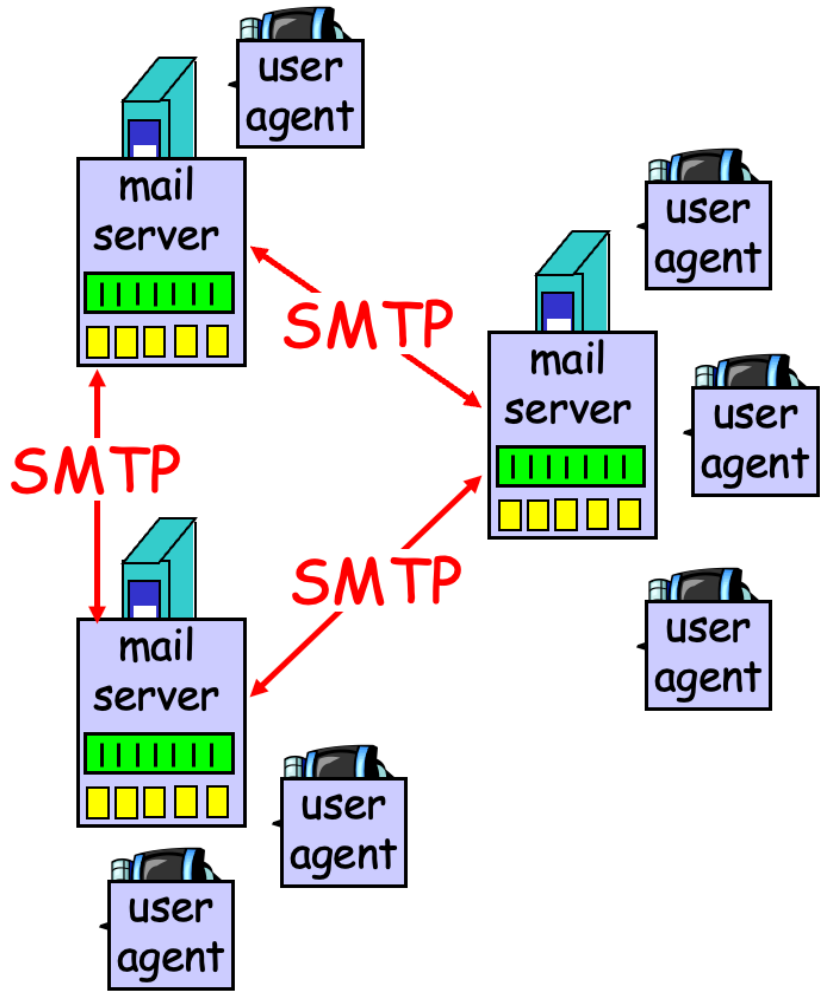
Electronic Mail

(E-mail)

Electronic Mail: mail servers

Mail Servers

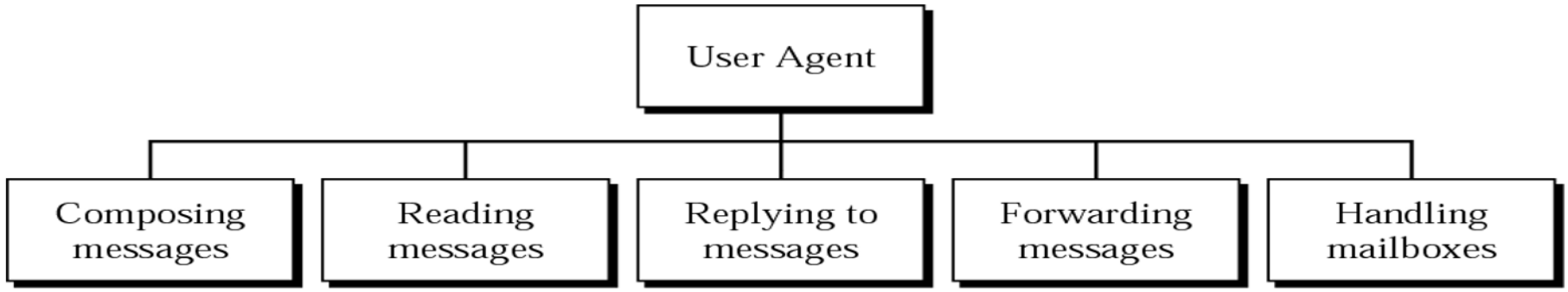
- mailbox contains incoming messages for user
- message queue of outgoing (to be sent) mail messages
- SMTP protocol between mail servers to send email messages
  - client: sending mail server
  - "server": receiving mail server



Electronic Mail

(E-mail)

User Agent



Note:

Some examples of command-driven user agents are mail, pine, and elm.

Note:

Some examples of GUI-based user agents are Eudora, Outlook, and Netscape.

Electronic Mail

(E-mail)

SMTP
(Simple Mail
Transfer Protocol)

Port Number: 25

Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCII

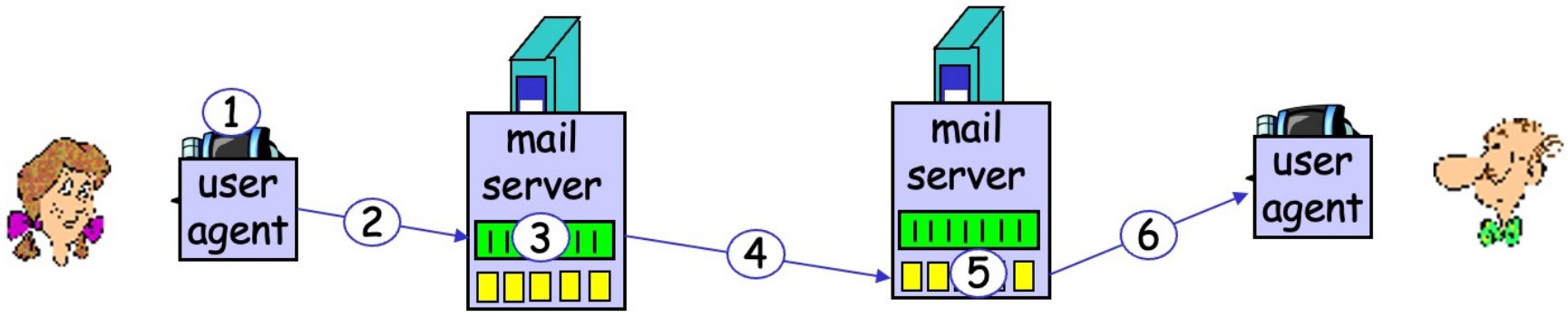
Electronic Mail

(E-mail)

SMTP
(Simple Mail
Transfer Protocol)

Scenario: Alice sends message to Bob

1) Alice uses UA to compose message and "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



Electronic Mail

(E-mail)

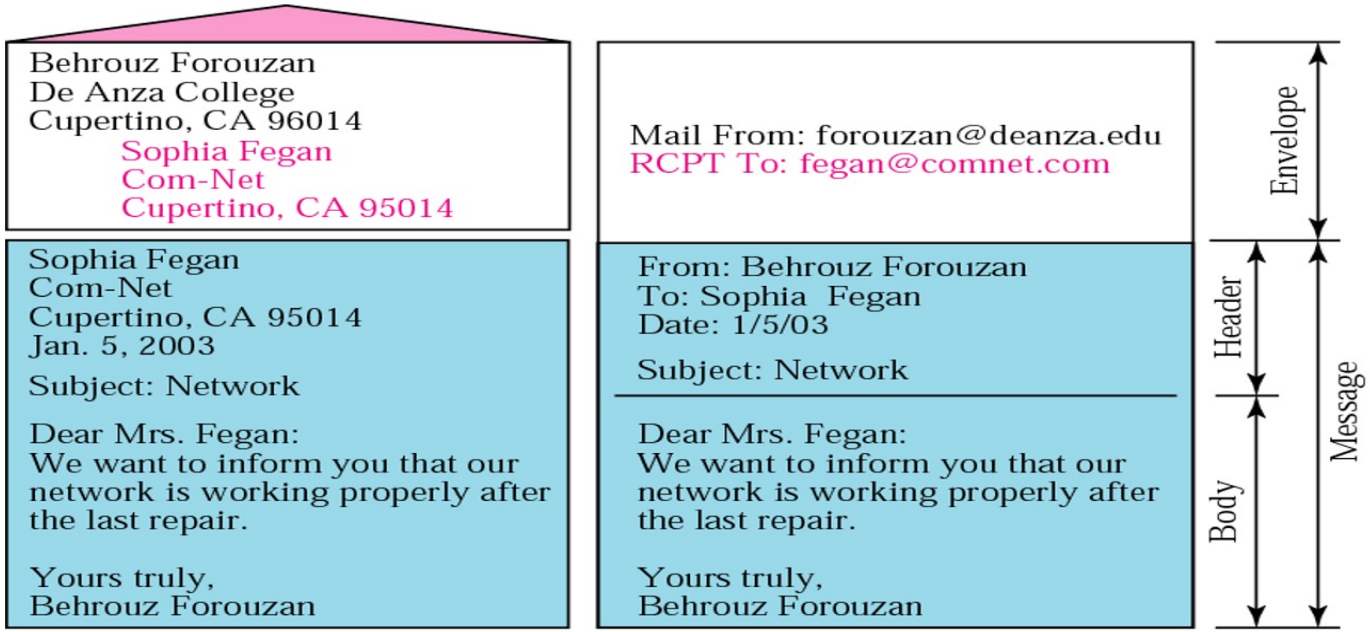
SMTP (Simple Mail Transfer Protocol)

- First, e-mail is considered a one-way transaction.
- When Alice sends an e-mail to Bob, she may expect a response, but this is not a mandate.
- Bob may or may not respond. If he does respond, it is another one-way transaction.
- Second, it is neither feasible nor logical for Bob to run a server program and wait until someone sends an e-mail to him.
- Bob may turn off his computer when he is not using it.
- This means that the idea of client/ server programming should be implemented in another way: using some intermediate computers (servers).

Electronic Mail

(E-mail)

Format



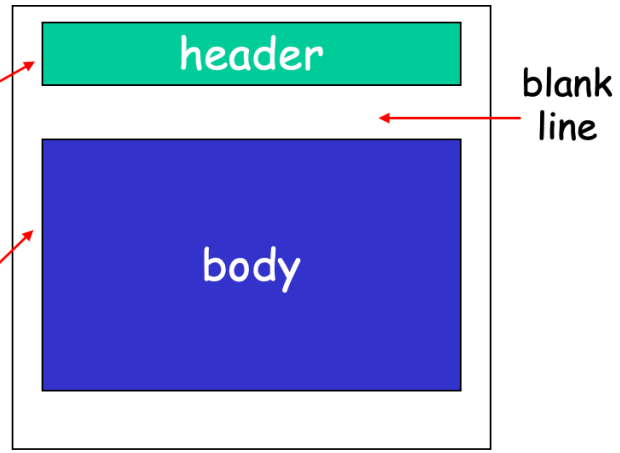
Mail message format

SMTP: protocol for exchanging email msgs

RFC 822: standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:

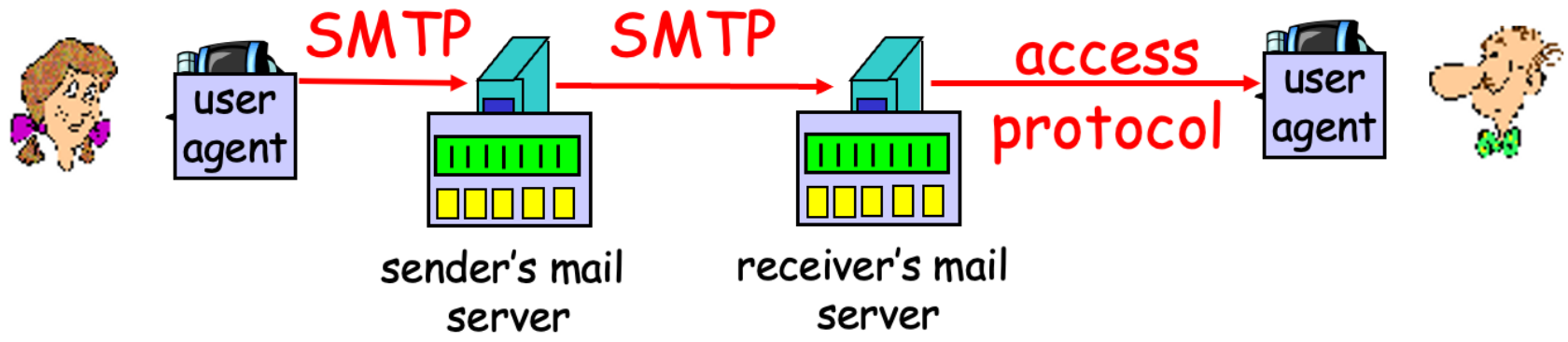
  different from SMTP commands!
- body
  - the "message", ASCII characters only



Electronic Mail

(MAC- Media Access Control)

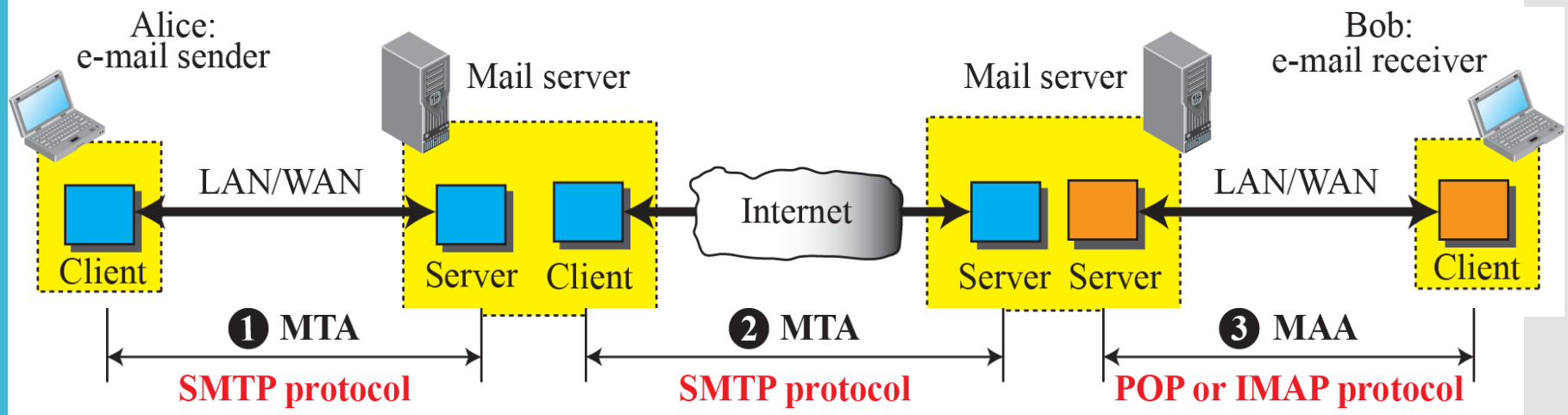
Mail access protocols



- 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: gmail, Hotmail, Yahoo! Mail, etc.

Electronic Mail

(MAC- Media Access Control)



SMTP Commands

Keyword	*Argument(s)*	*Description*
HELO	Sender's host name	Identifies itself
MAIL FROM	Sender of the message	Identifies the sender of the message
RCPT TO	Intended recipient	Identifies the recipient of the message
DATA	Body of the mail	Sends the actual message
QUIT		Terminates the message

SMTP 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
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

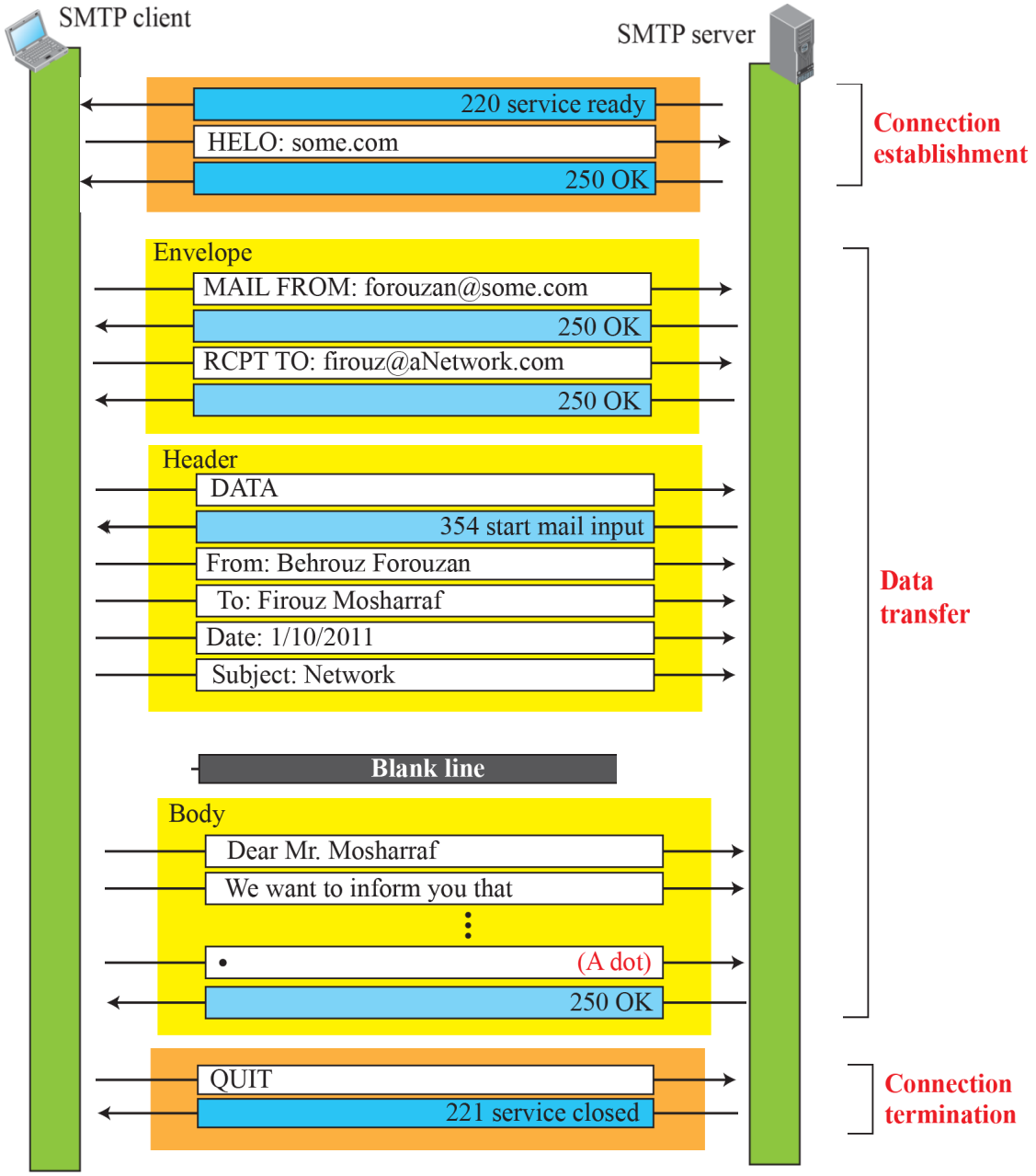
Example

(Working SMTP)

- To show the three mail transfer phases, we show all of the steps described above using the information depicted in Figure.
- In the figure, we have separated the messages related to the envelope, header, and body in the data transfer section.
- Note that the steps in this figure are repeated two times in each e-mail transfer:
  - one from the e-mail sender to the local mail server
  - one from the local mail server to the remote mail server.
  - The local mail server, after receiving the whole e-mail message, may spool it and send it to the remote mail server at another time.

Example

(Working SMTP)

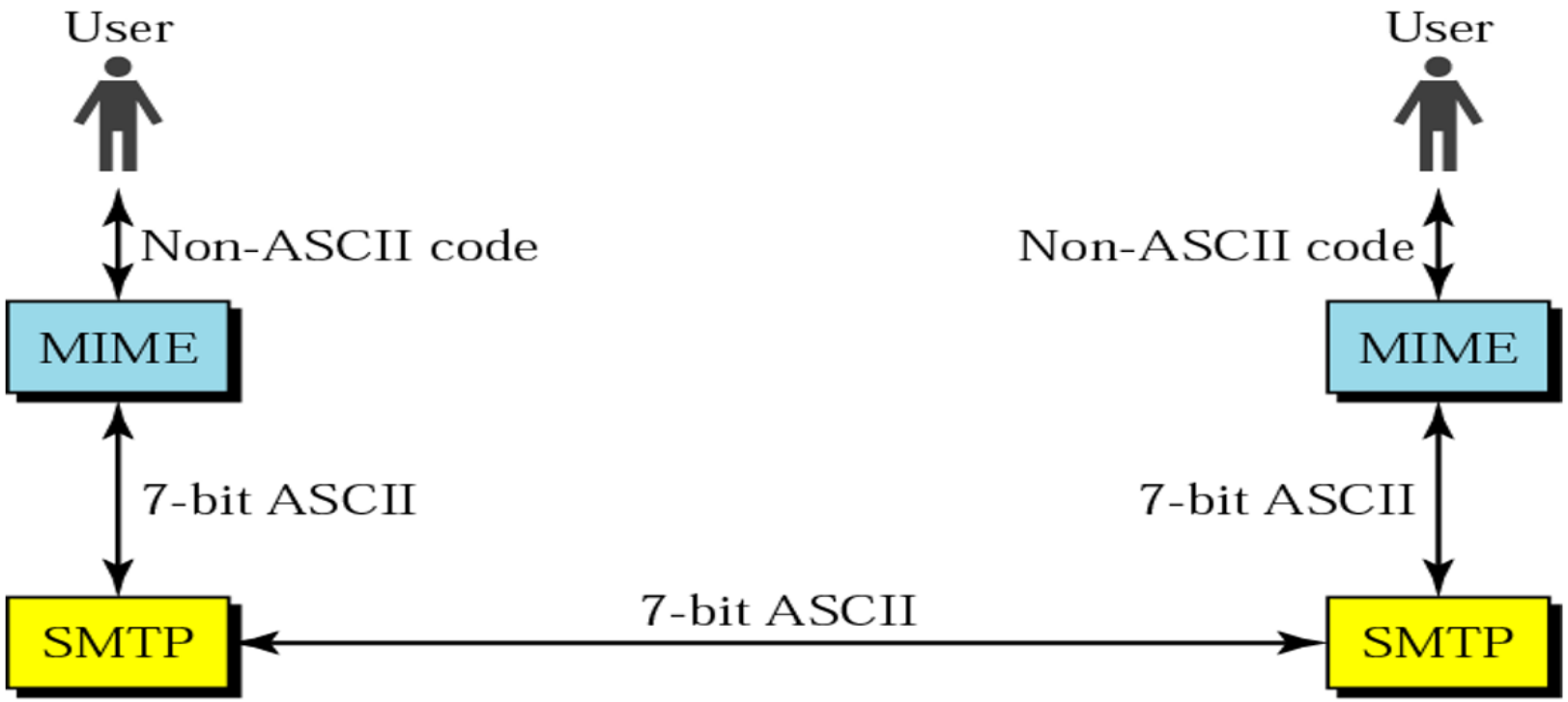


Multipurpose Internet Mail Extensions (MIME)

Multipurpose Internet Mail Extensions:

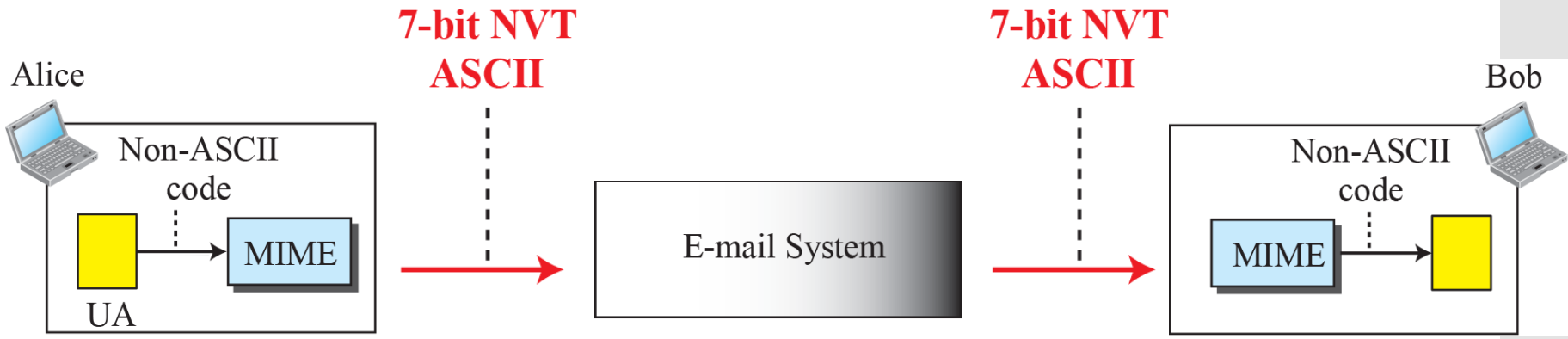
(Extension of SMTP)

- MIME is a specification for formatting non-ASCII messages so that they can be sent over the Internet.
- Many e-mail clients now support MIME, which enables them to send and receive graphics, audio, and video files via the Internet mail system.



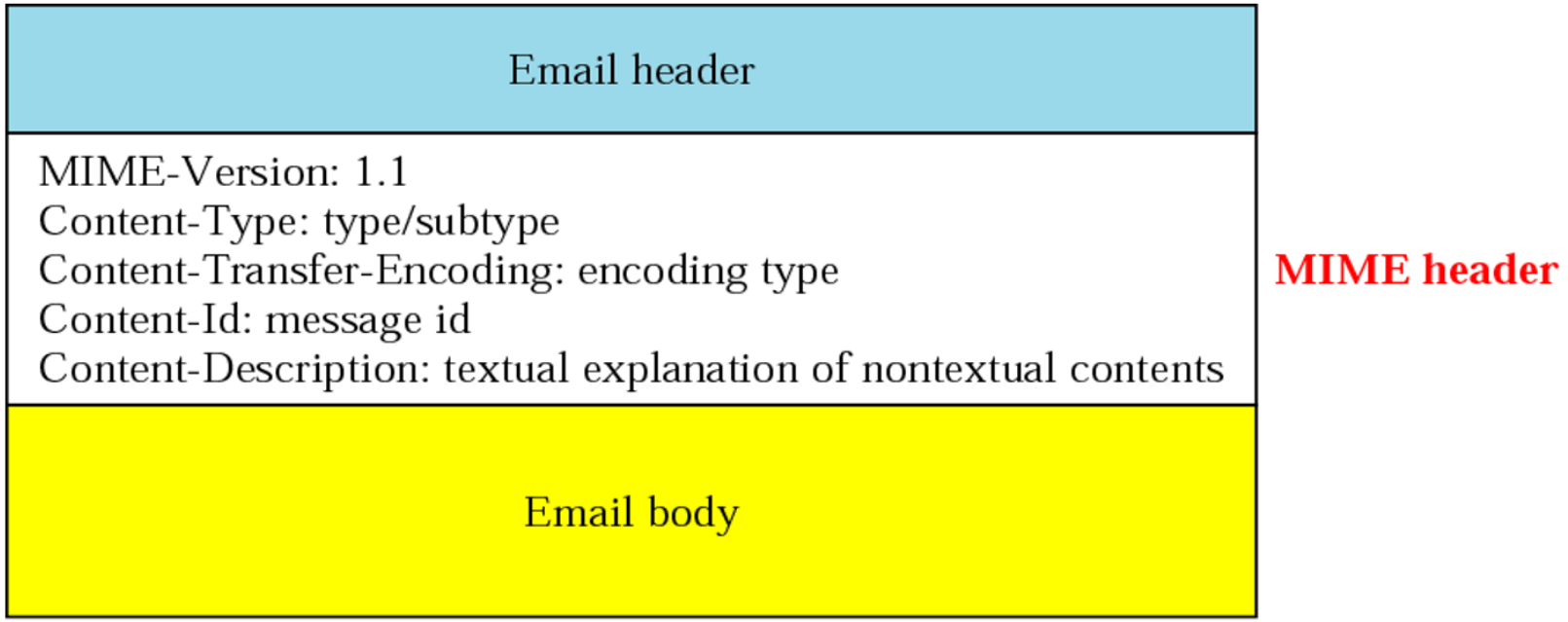
Multipurpose Internet Mail Extensions (MIME)

- Servers insert the MIME header at the beginning of any Web transmission.
- Clients use this header to select an appropriate "player" application for the type of data the header indicates.
- Some of these players are built into the Web client or browser (for example, all browsers come with GIF and JPEG image players as well as the ability to handle HTML files); other players may need to be downloaded.



Multipurpose Internet Mail Extensions (MIME)

Header



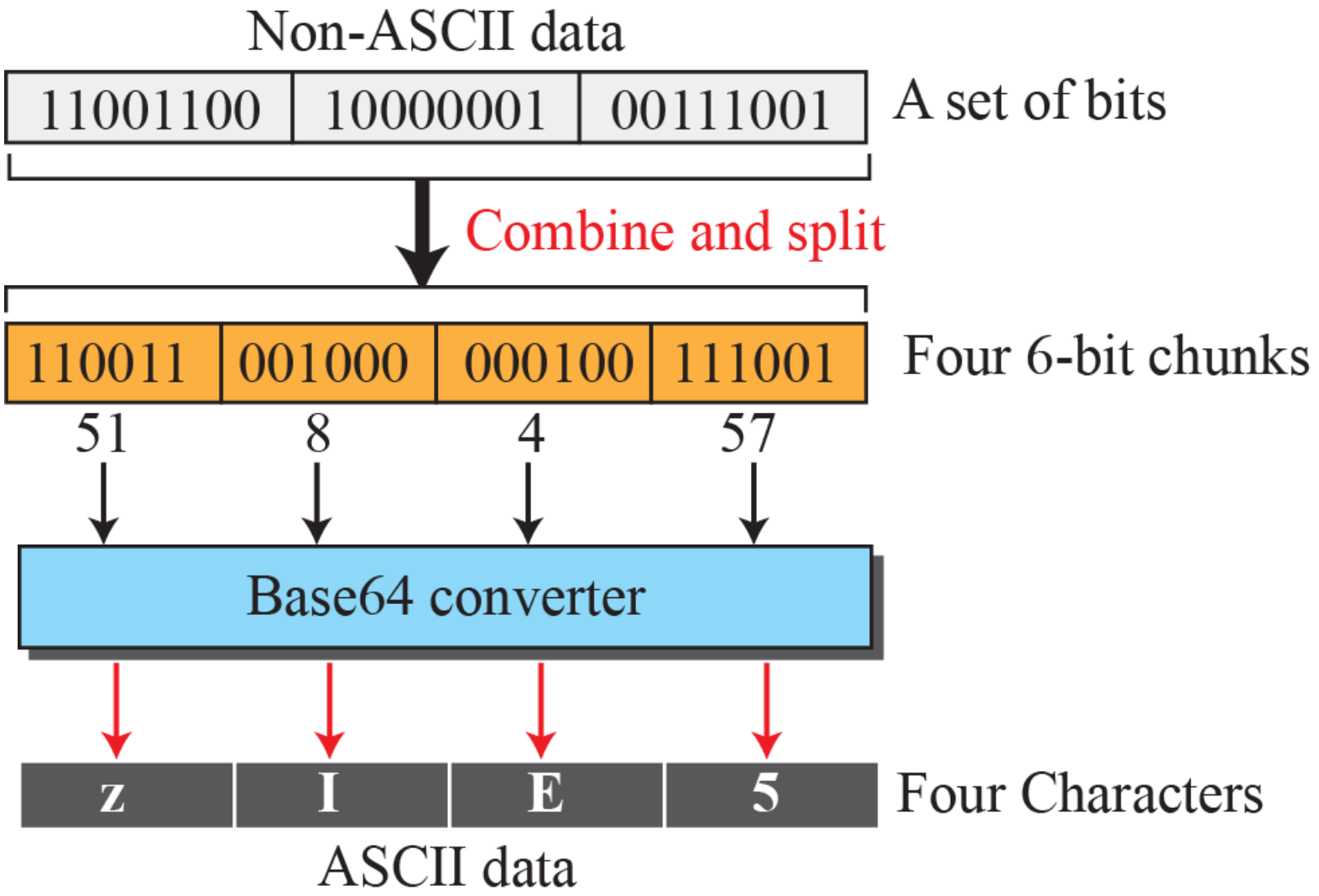
Data Types and Subtypes in MIME

Type	*Subtype*	*Description*
Text	Plain	Unformatted
	HTML	HTML format (see Appendix C)
Multipart	Mixed	Body contains ordered parts of different data types
	Parallel	Same as above, but no order
	Digest	Similar to Mixed, but the default is message/RFC822
	Alternative	Parts are different versions of the same message
Message	RFC822	Body is an encapsulated message
	Partial	Body is a fragment of a bigger message
	External-Body	Body is a reference to another message
Image	JPEG	Image is in JPEG format
	GIF	Image is in GIF format
Video	MPEG	Video is in MPEG format
Audio	Basic	Single channel encoding of voice at 8 KHz
Application	PostScript	Adobe PostScript
	Octet-stream	General binary data (eight-bit bytes)

Methods for Content-Transfer-Encoding

Type	*Description*
7-bit	NVT ASCII characters with each line less than 1000 characters
8-bit	Non-ASCII characters with each line less than 1000 characters
Binary	Non-ASCII characters with unlimited-length lines
Base64	6-bit blocks of data encoded into 8-bit ASCII characters
Quoted-printable	Non-ASCII characters encoded as an equal sign plus an ASCII code

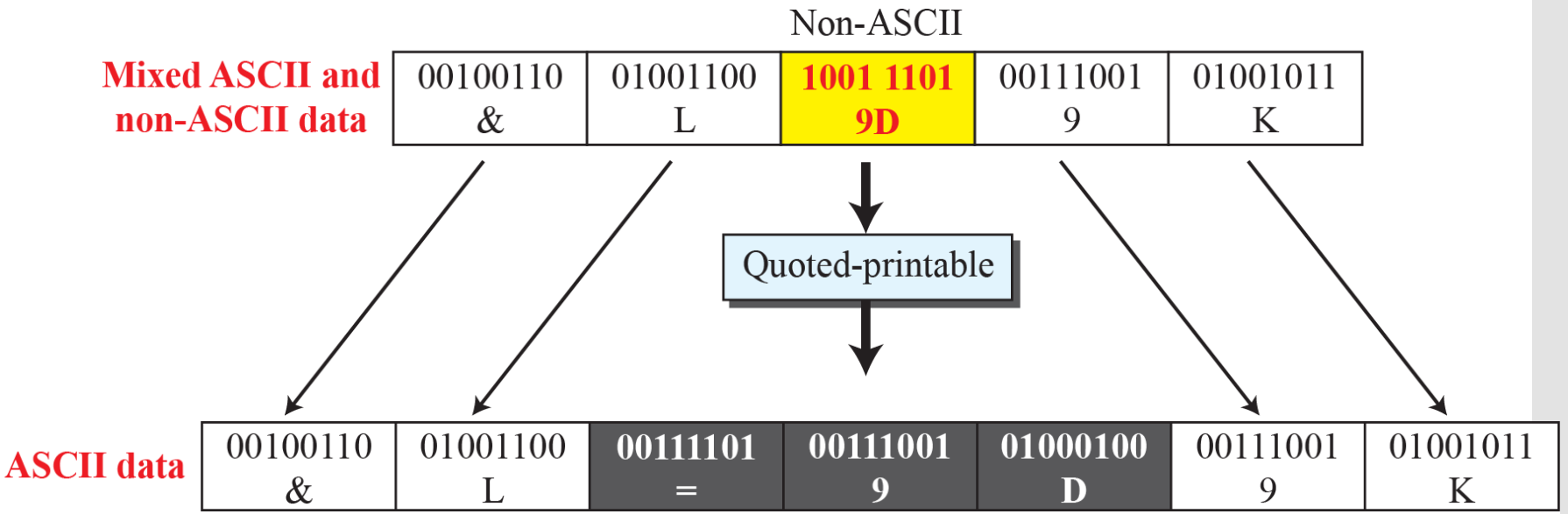
Base64 conversion



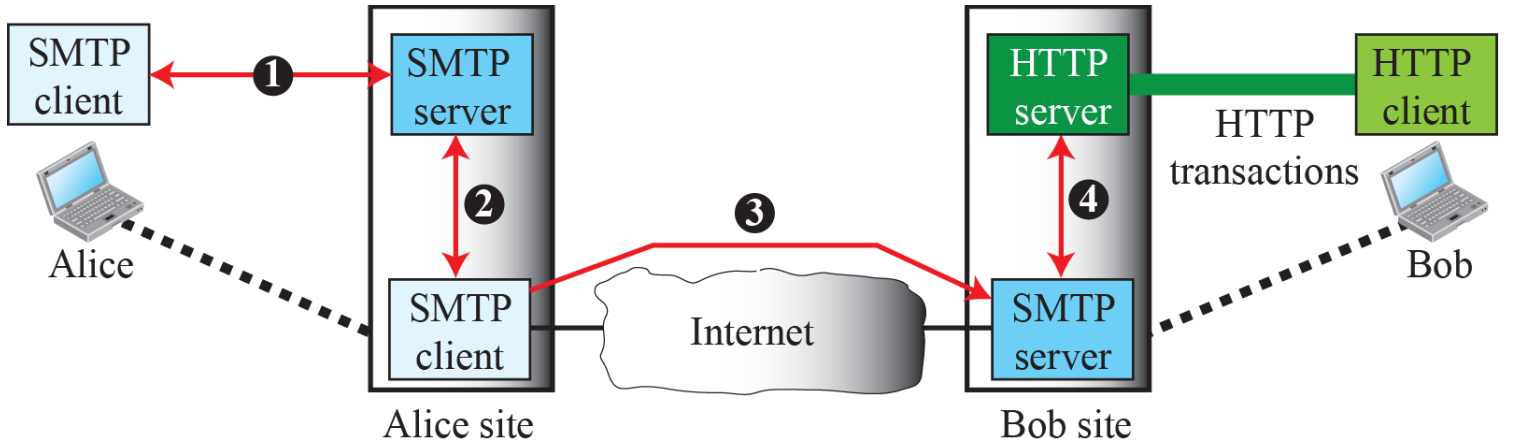
Base64 Converting Table

Value	*Code*	*Value*	*Code*	*Value*	*Code*	*Value*	*Code*	*Value*	*Code*	*Value*	*Code*
0	**A**	11	**L**	22	**W**	33	**h**	44	**s**	55	**3**
1	**B**	12	**M**	23	**X**	34	**i**	45	**t**	56	**4**
2	**C**	13	**N**	24	**Y**	35	**j**	46	**u**	57	**5**
3	**D**	14	**O**	25	**Z**	36	**k**	47	**v**	58	**6**
4	**E**	15	**P**	26	**a**	37	**l**	48	**w**	59	**7**
5	**F**	16	**Q**	27	**b**	38	**m**	49	**x**	60	**8**
6	**G**	17	**R**	28	**c**	39	**n**	50	**y**	61	**9**
7	**H**	18	**S**	29	**d**	40	**o**	51	**z**	62	+
8	**I**	19	**T**	30	**e**	41	**p**	52	**0**	63	/
9	**J**	20	**U**	31	**f**	42	**q**	53	**1**		
10	**K**	21	**V**	32	**g**	43	**r**	54	**2**		

Quoted-printable

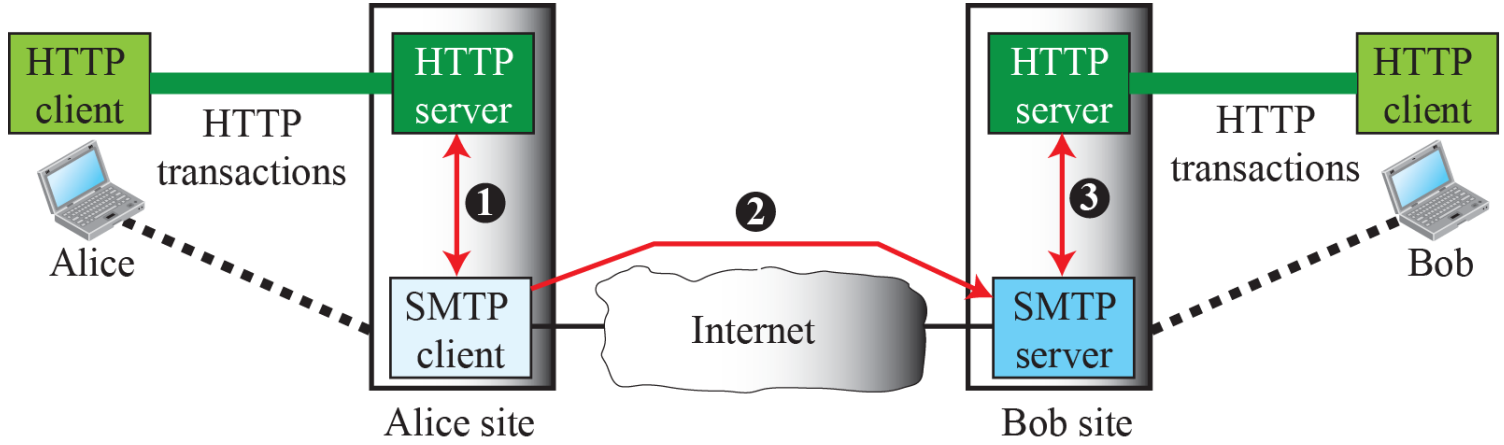


Web-based e-mail, cases I and II



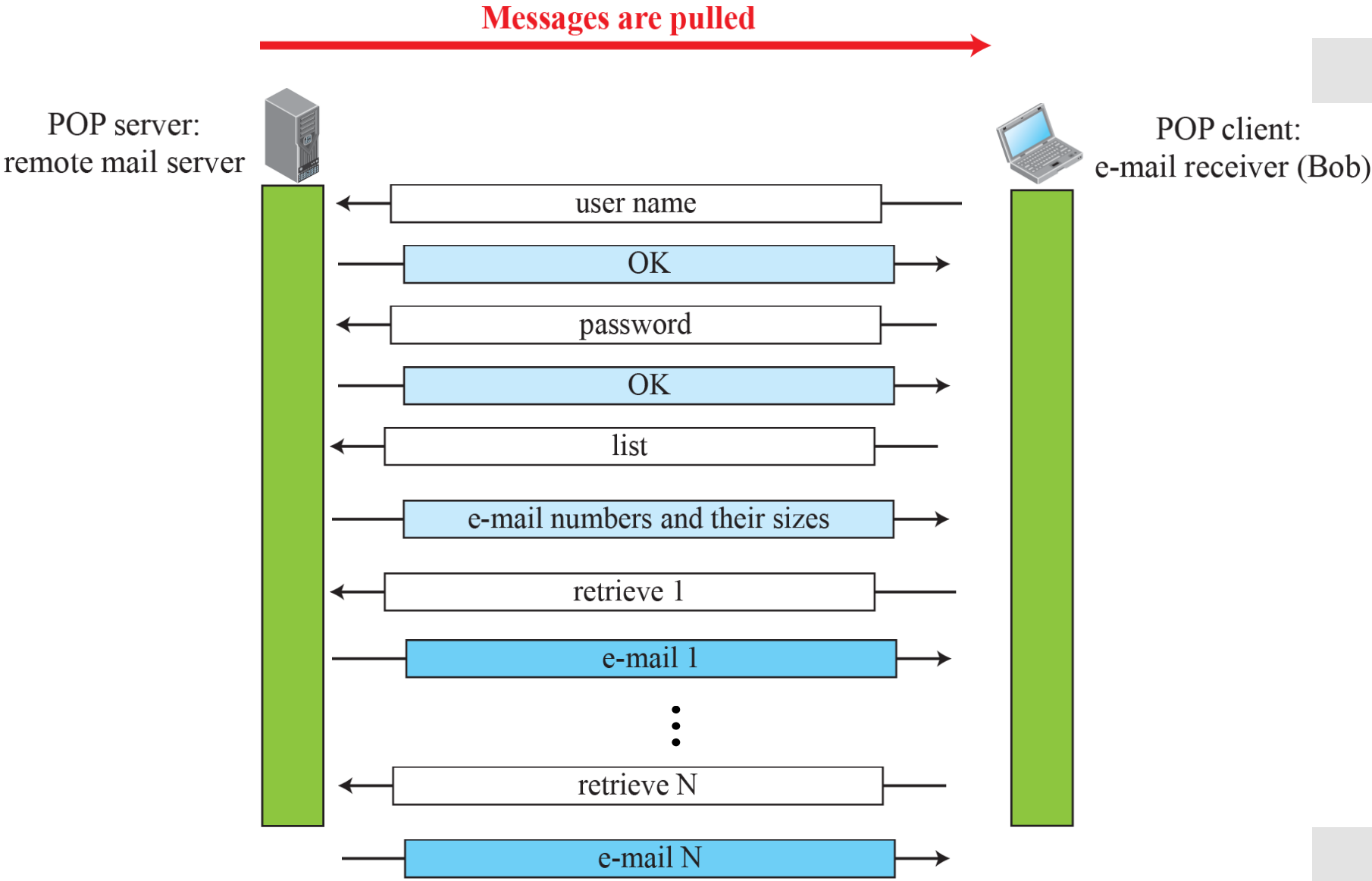
Case 1: Only receiver uses HTTP

HTTP Protocol: This is not a dedicated protocol for email communications, but it can be used for accessing your mailbox also called web based email, this can be used to compose or retrieve emails from an your account. Hotmail is a good example of using HTTP as an email protocol.



Case 2: Both sender and receiver use HTTP

POP3



Port Number: 110

POP3 (more) and IMAP

POP3 and IMAP

More about POP3

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

IMAP

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

- POP3(Post Office Protocol) and IMAP(Internet Message Access Protocol) are two different protocols (methods) used to access email.
- IMAP is the better option - and the recommended option - when you need to check your emails from multiple devices, such as a work laptop, a home computer, or a tablet, smart phone, or other mobile device.

Port Number: 143

Domain Name System (DNS):

- To identify an entity, TCP/IP protocols use the IP address, which uniquely identifies the connection of a host to the Internet.

  - However, people prefer to use names instead of numeric addresses.
  - Therefore, the Internet needs to have a directory system that can map a name to an address.
  - This is analogous to the telephone network.
  - A telephone network is designed to use telephone numbers, not names.
  - People can either keep a private file to map a name to the corresponding telephone number or can call the telephone directory to do so.

Domain Name System (DNS): (continued)

- Name Space
  - Domain Name Space
  - Domain
  - Distribution of Name Space
  - Zone
  - Root Server
- DNS in the Internet
  - Generic Domains
  - Country Domains
- Resolution
  - Recursive Resolution
  - Iterative Resolution
  - Caching
- Resource Records
- DNS Messages
- Encapsulation
- Registrars
- DDNS
- Security of DNS

Domain Name System (DNS): (continued)

DNS: Domain Name System

People: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

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

Q: map between IP addresses and name ?

Domain Name System:

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

Domain Name System (DNS): (continued)

DNS

DNS services

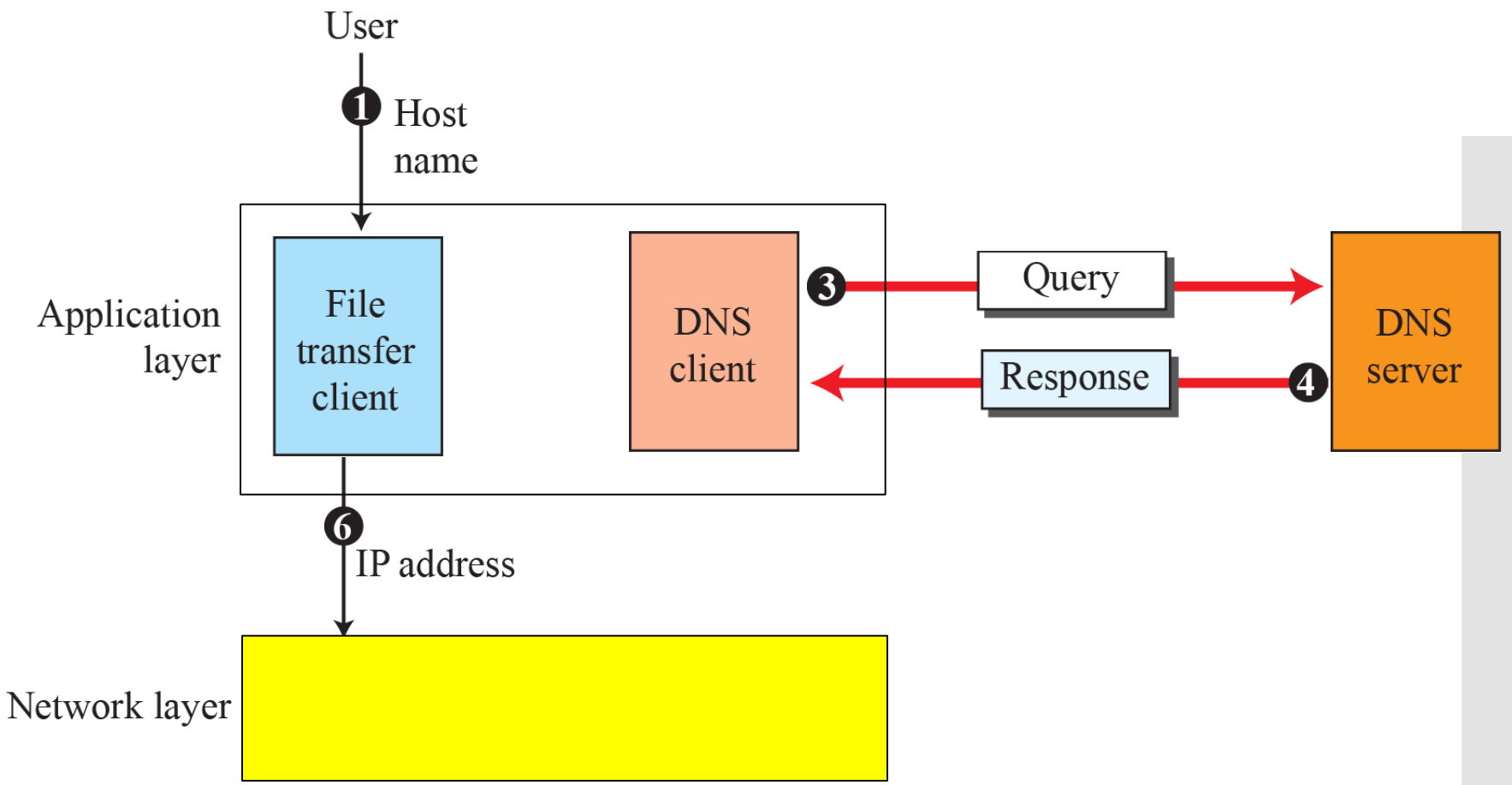
- hostname to IP address translation
- host aliasing
  - Canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: set of IP addresses for one canonical name

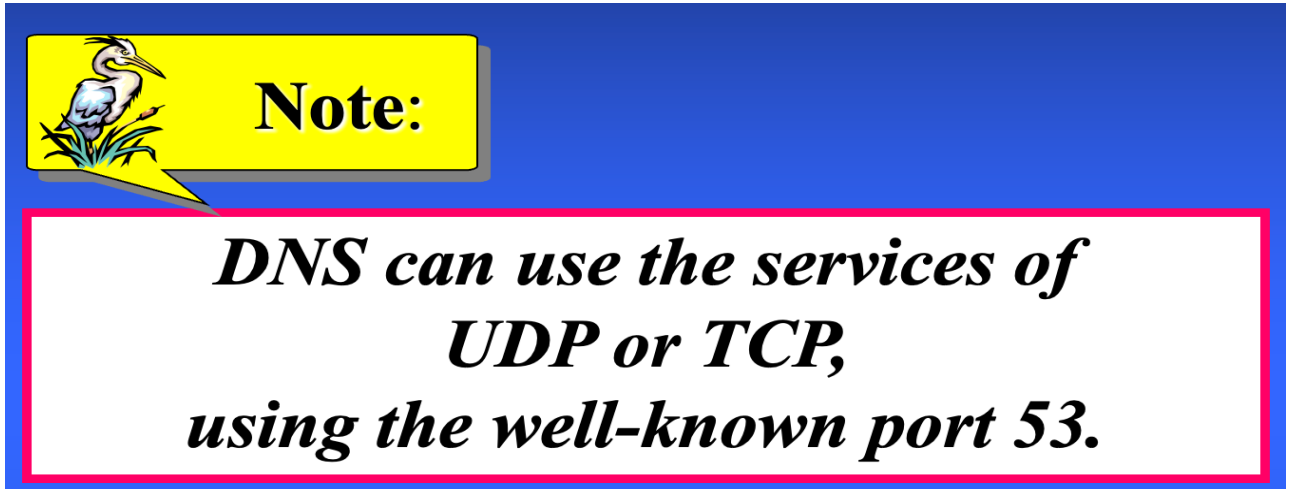
Why not centralize DNS?

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

doesn't *scale!*

Purpose of DNS

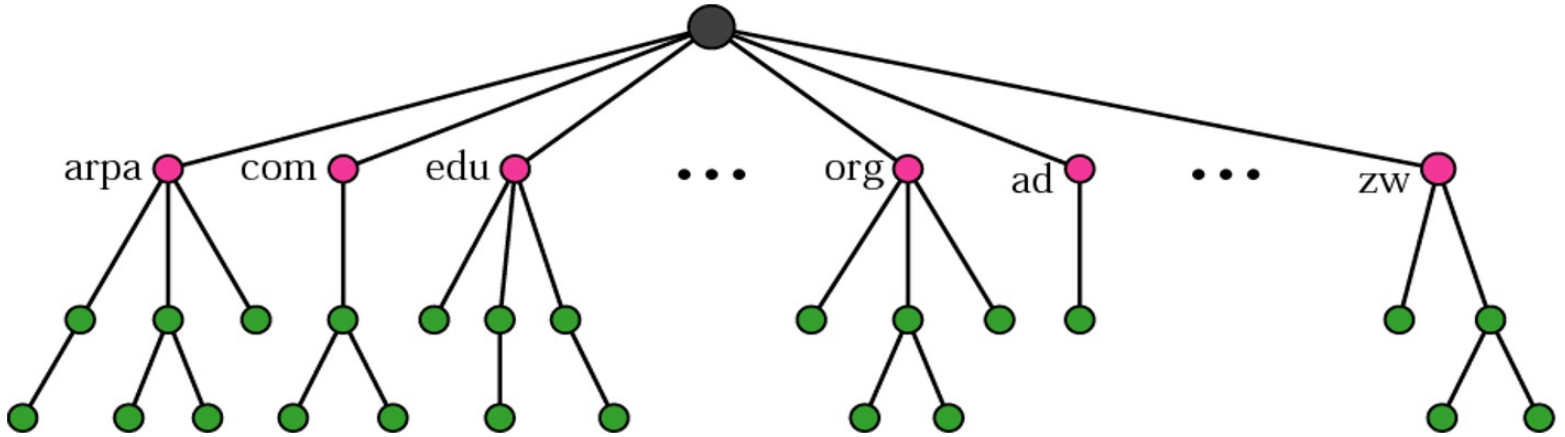




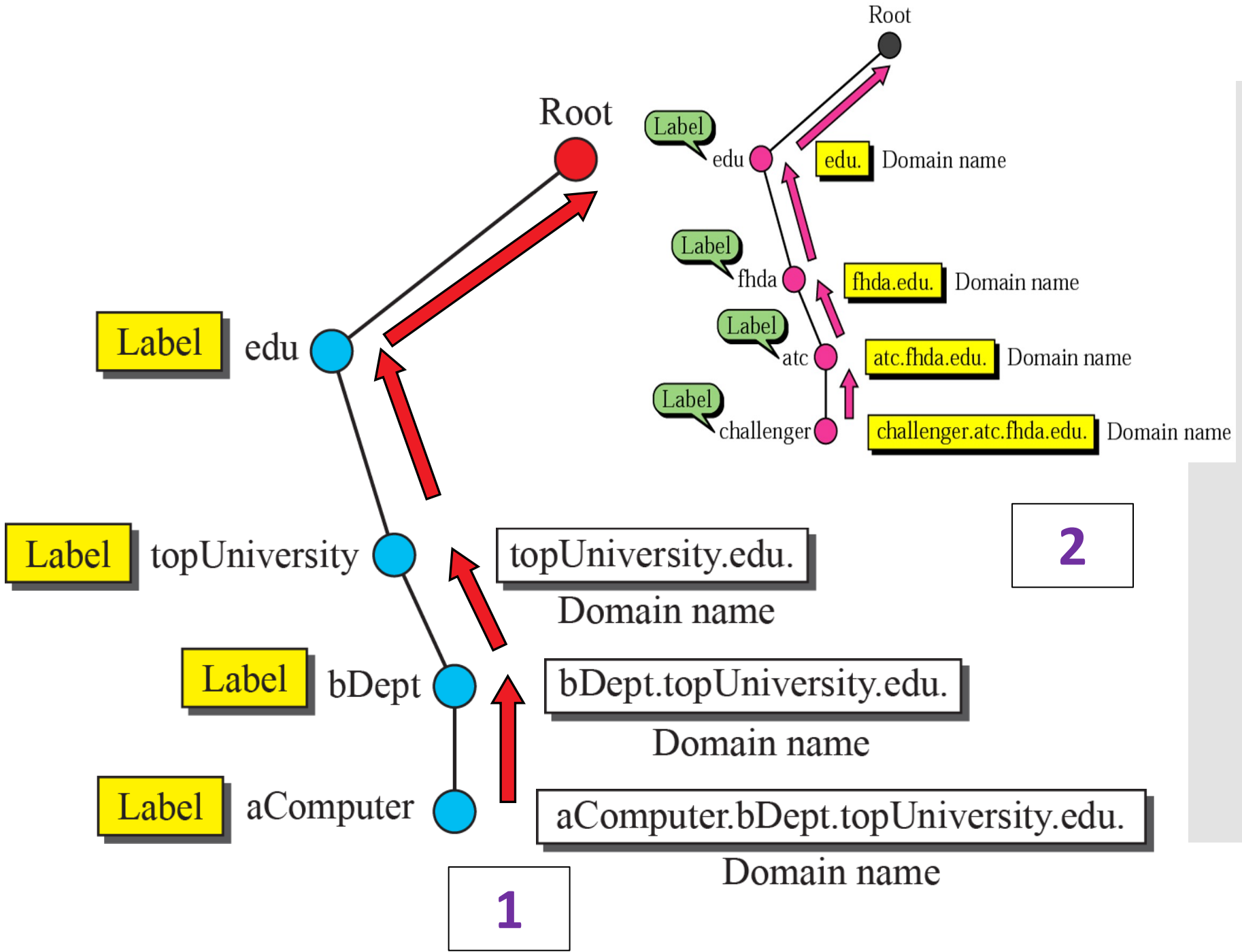
Domain Name System (DNS): Domain name space

Domain name space:

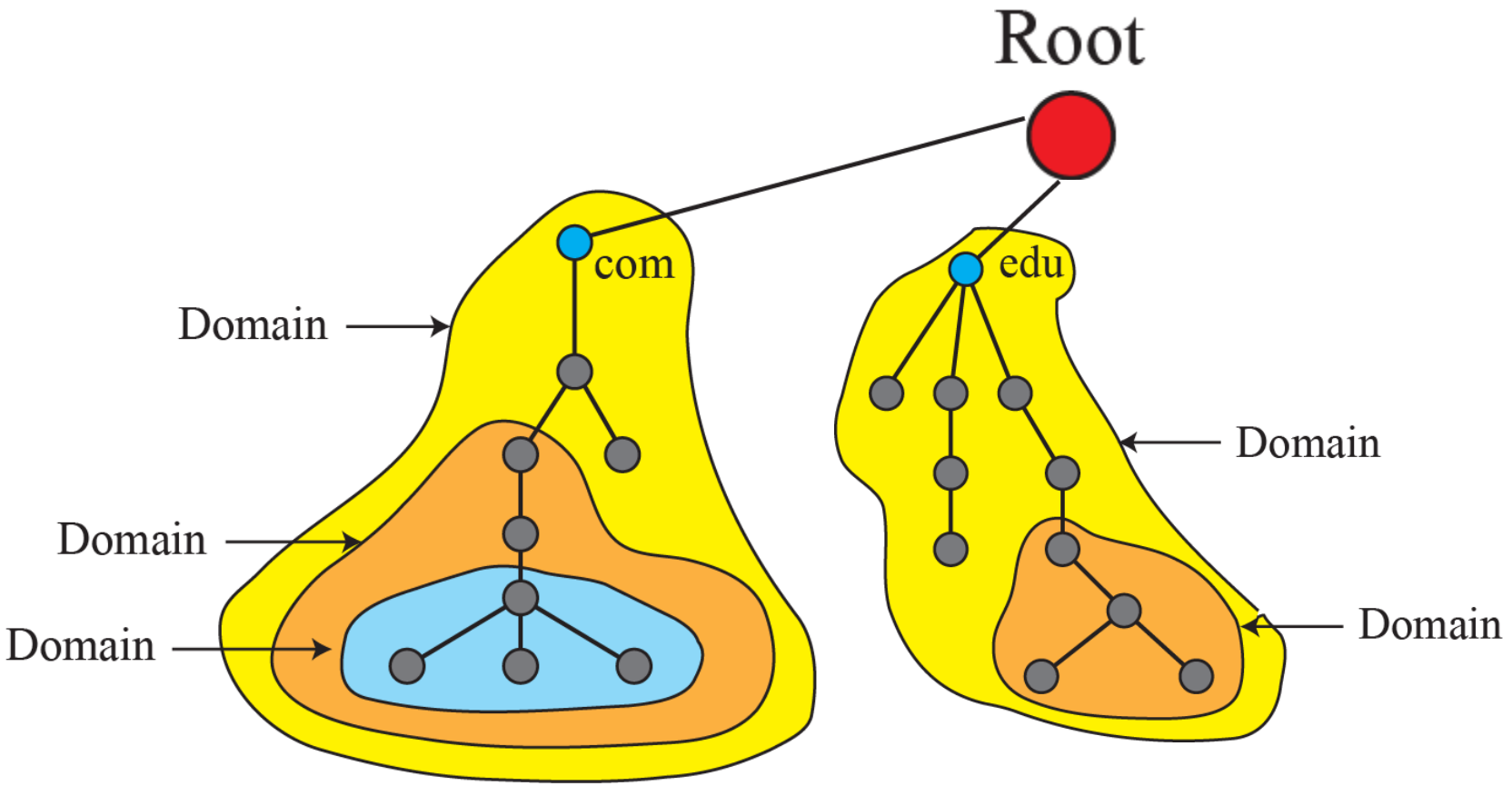
- Alternatively referred to as a namespace, a domain namespace is a name service provided by the Internet for Transmission Control Protocol and Networks/Internet Protocol (TCP/IP).
- DNS is broken up into domains, a logical organization of computers that exist in a larger network.

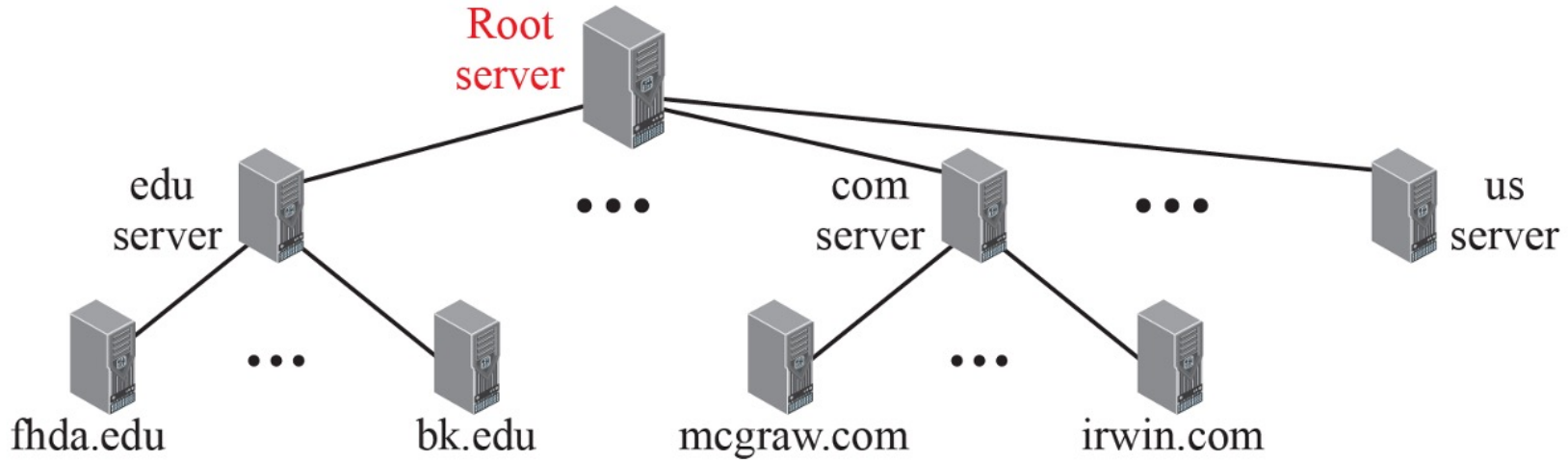


Domain Name System (DNS): Domain names and labels



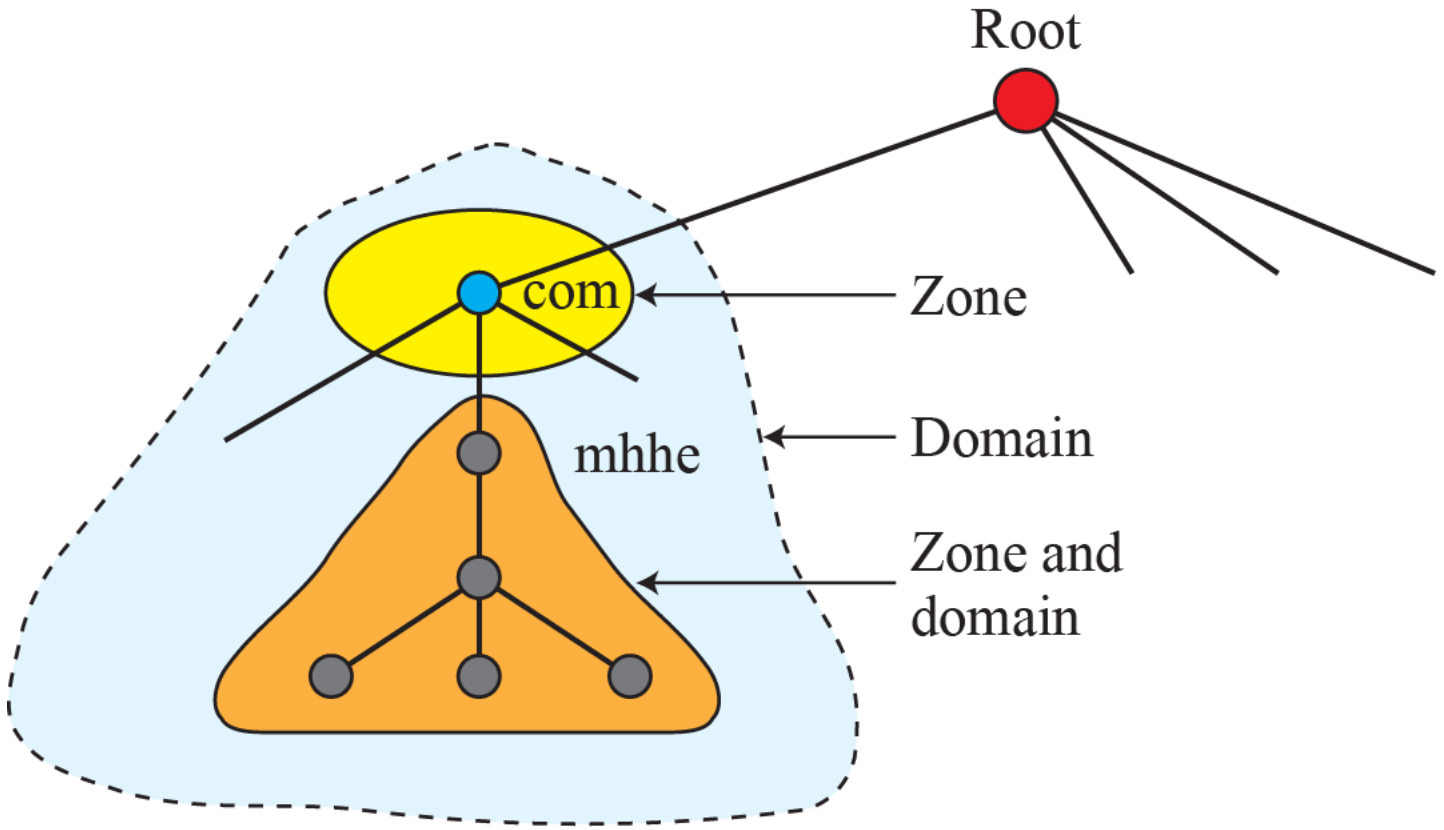
Domain Name System (DNS): Domains and Hierarchy of Server



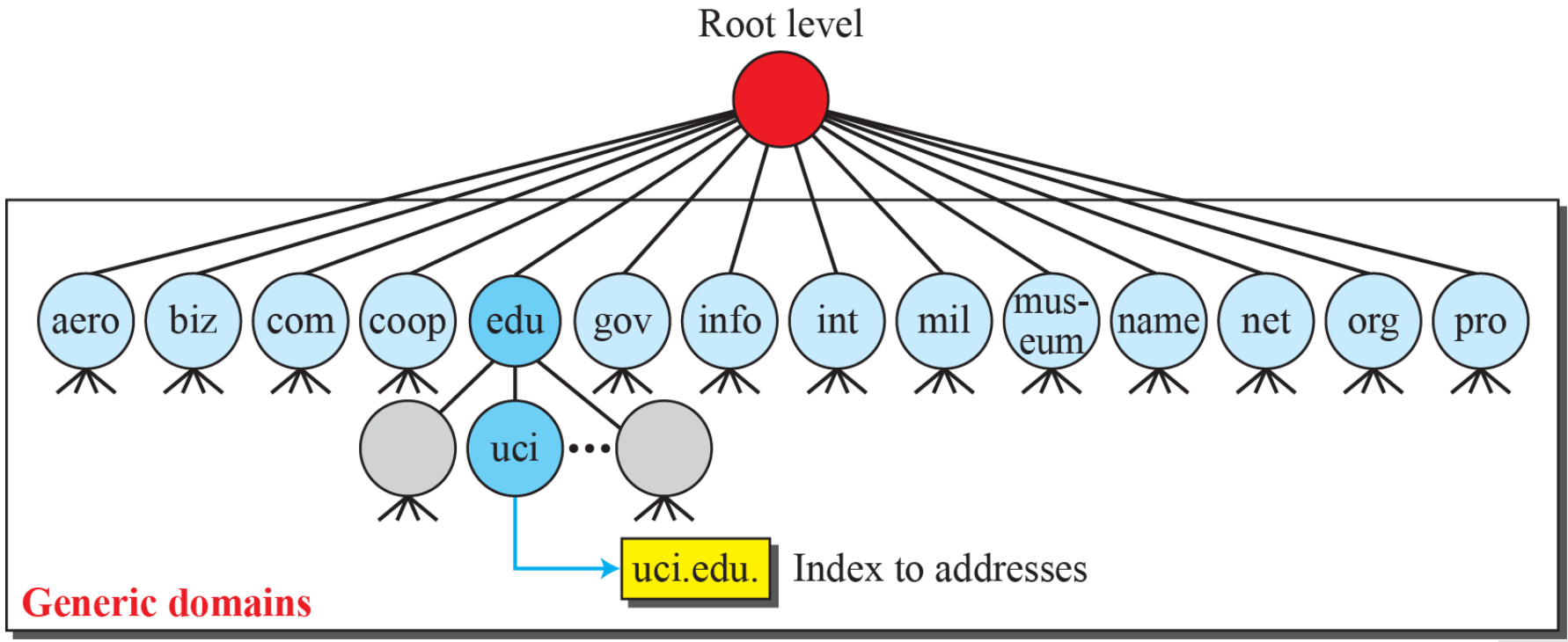


Domain Name System (DNS): Zone : A zone is simply a portion of a domain

- For example, the Domain Microsoft.com may contain all of the data for Microsoft.com, Marketing.microsoft.com and Development.microsoft.com. However, the zone Microsoft.com contains only information for Microsoft.com and references to the authoritative name servers for the sub-domains.



Domain Name System (DNS): Generic domains



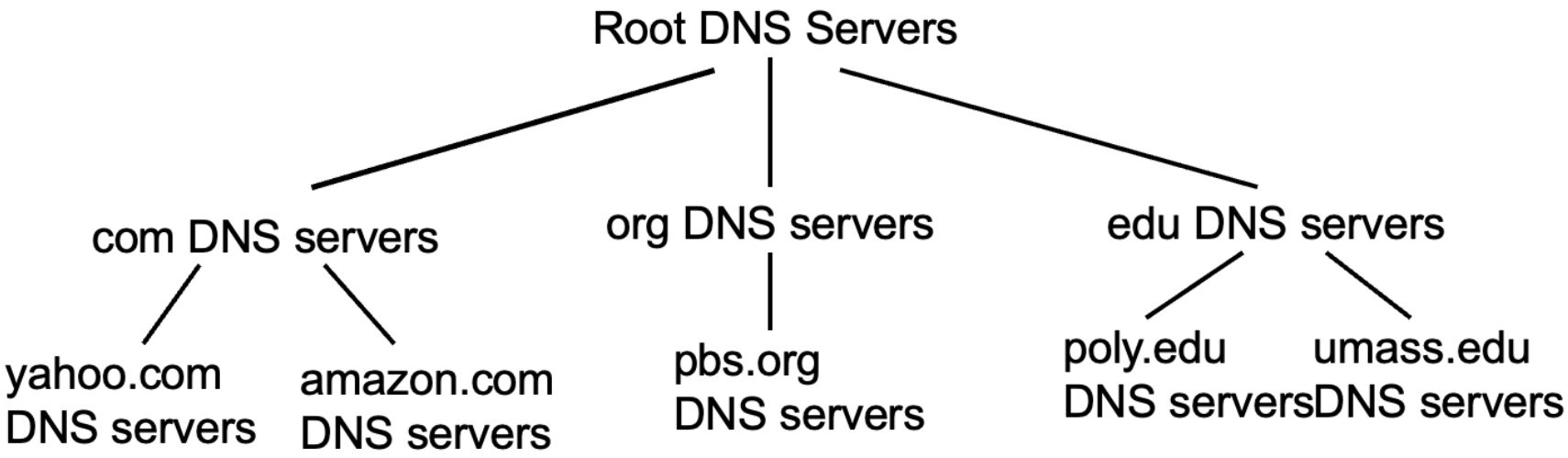
A primary server loads all information from the disk file; the secondary server loads all information from the primary server.

Domain Name System (DNS): Generic domain labels

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

Domain Name System (DNS): Distributed, Hierarchical Databases

Distributed, Hierarchical Database



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

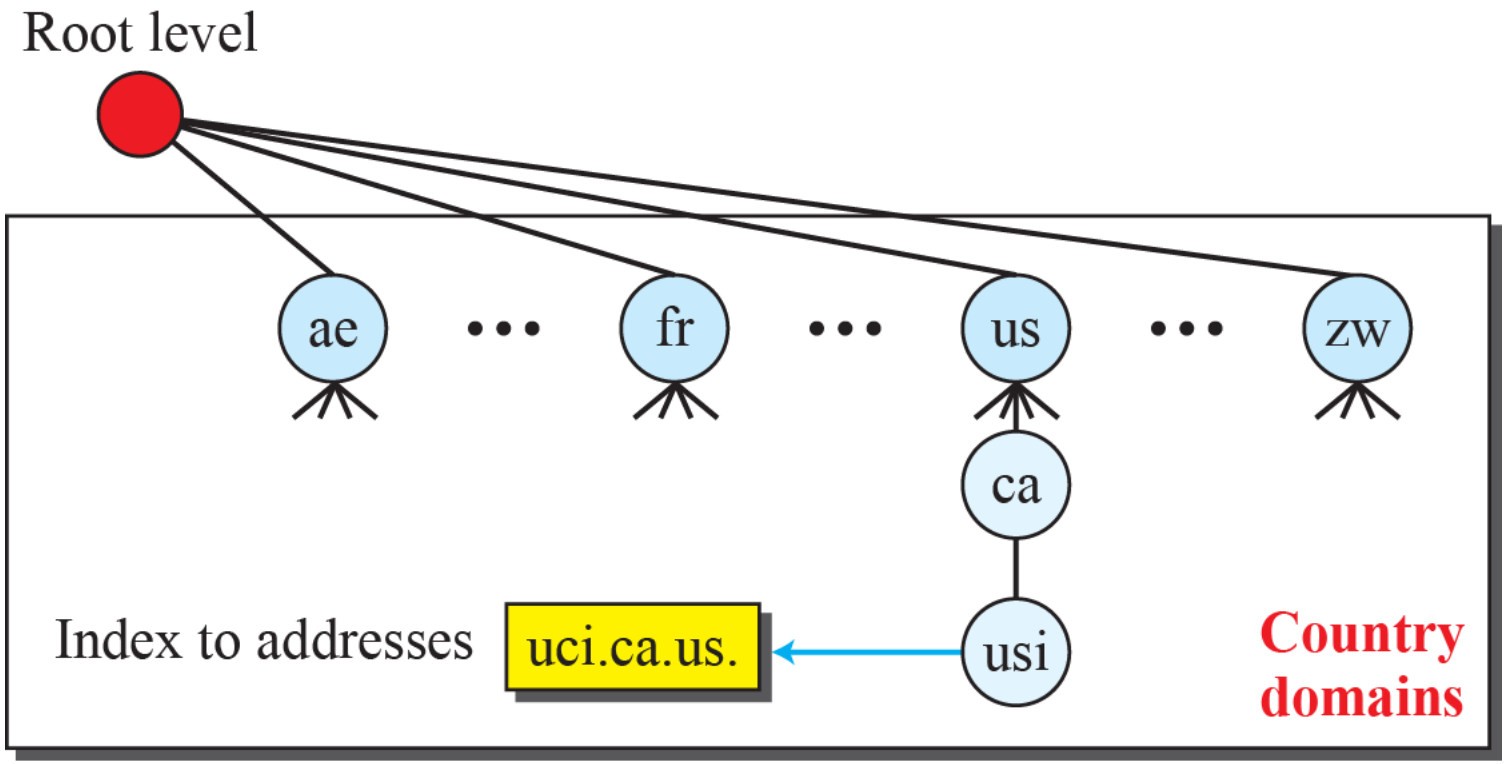
- client queries a root server to find com DNS server
- client queries com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

Domain Name System (DNS): TLD and Authoritative Servers

TLD and Authoritative Servers

- Top-level domain (TLD) servers:
  - responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
  - Network Solutions maintains servers for com TLD
  - Educause for edu TLD
- Authoritative DNS servers:
  - organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web, mail).
  - can be maintained by organization or service provider

Domain Name System (DNS): Country domain



Domain Name System (DNS): Local DNS name server

Local Name Server

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

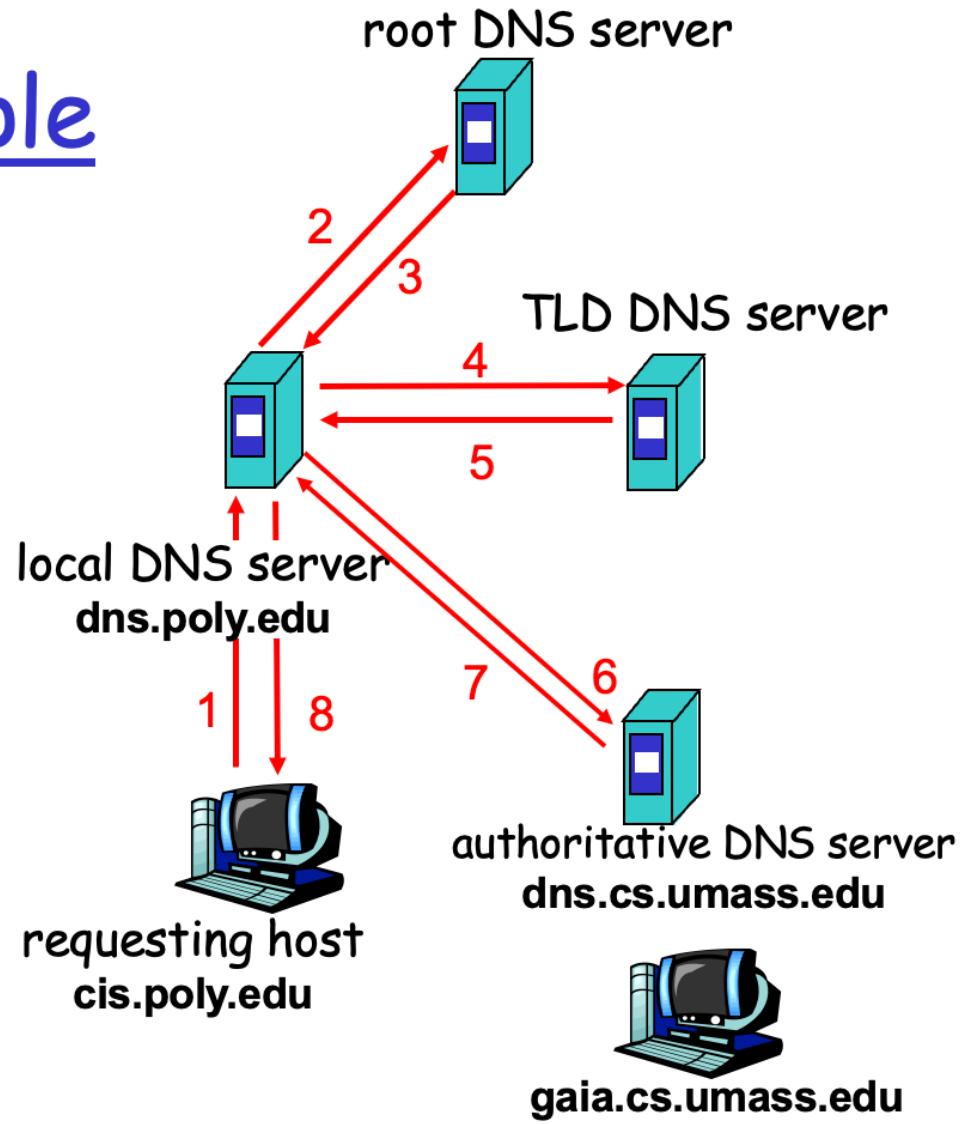
Domain Name System (DNS): Resolution Example

DNS name resolution example

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

iterated query:

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

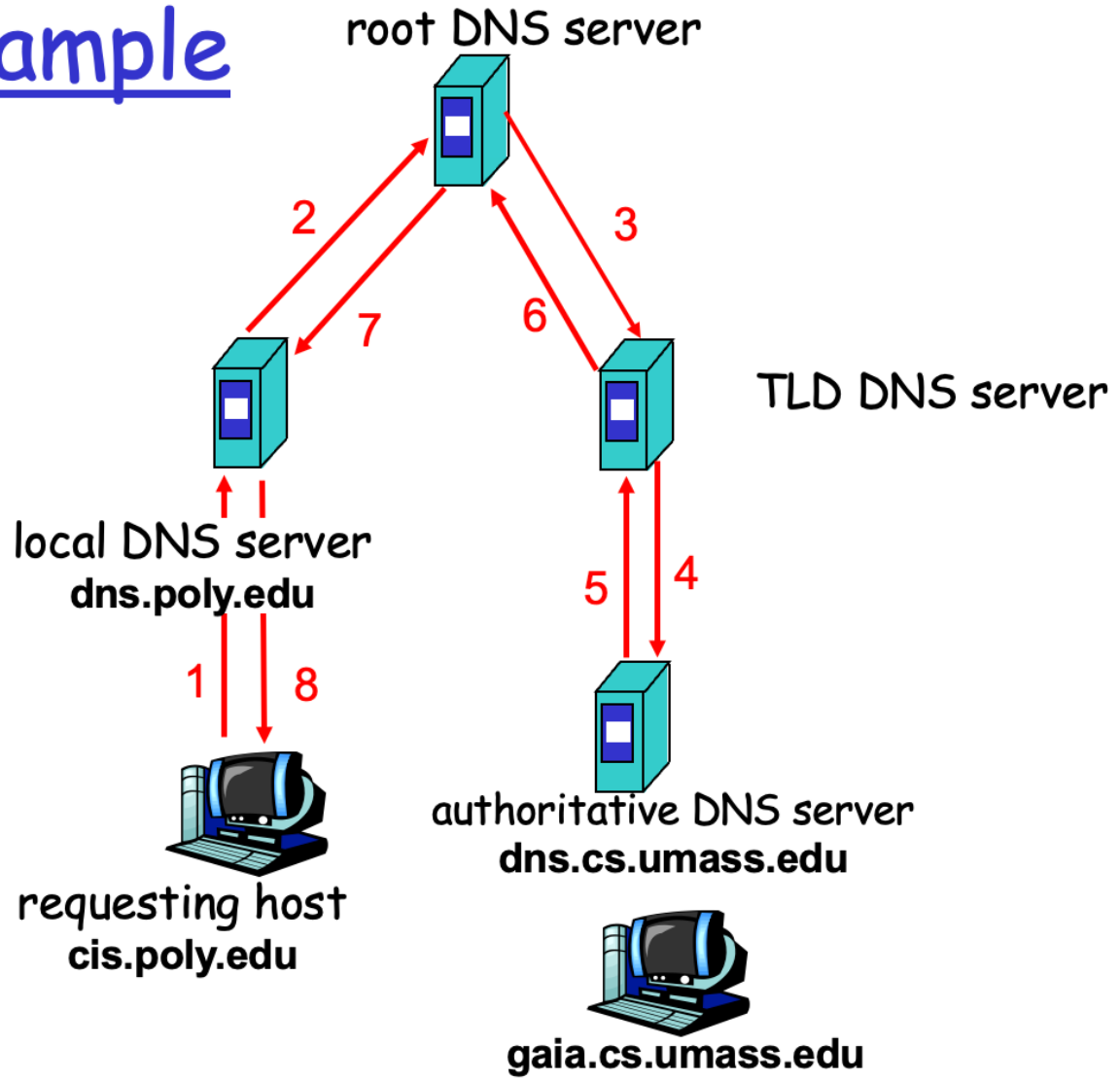


Domain Name System (DNS): Resolution Example

DNS name resolution example

recursive query:

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



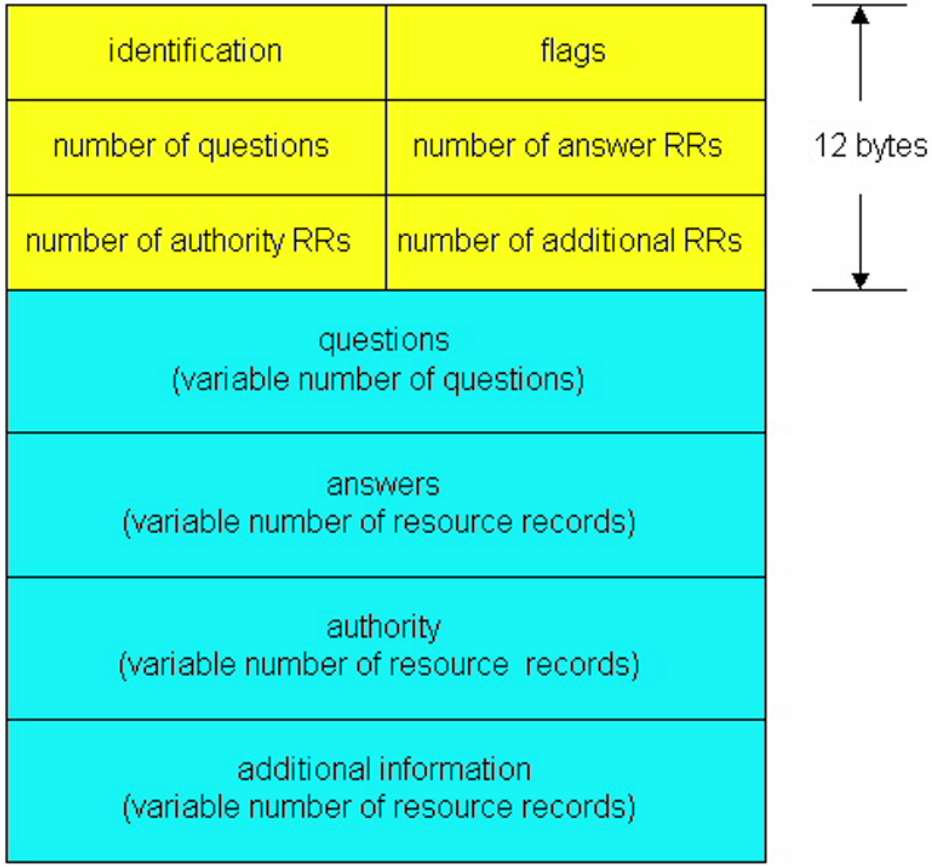
Domain Name System (DNS): Protocol, Messages

DNS protocol, messages

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

msg header

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



Domain Name System (DNS): Protocol, Messages

Header Format

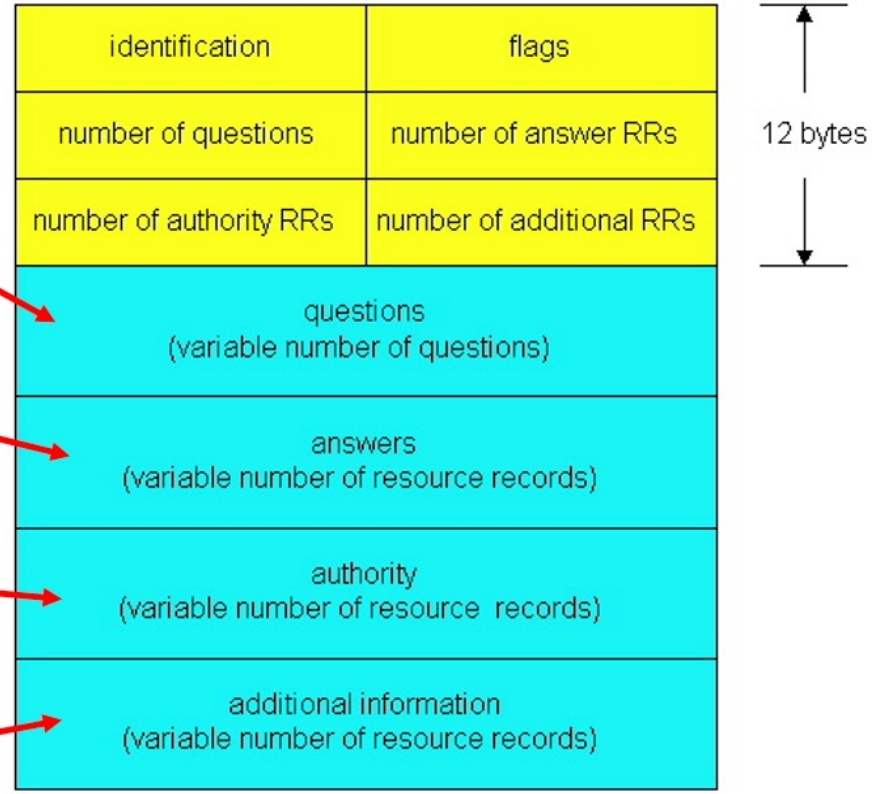
DNS protocol, messages

Name, type fields for a query

RRs in response to query

records for authoritative servers

additional "helpful" info that may be used



TELNET

(Terminal Network)

(RFC-854)

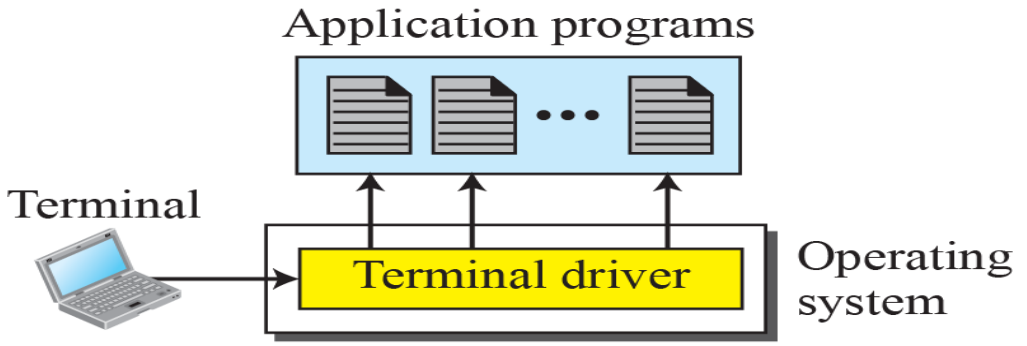
- A server program can provide a specific service to its corresponding client program.
- However, it is impossible to have a client/server pair for each type of service we need.
- Another solution is to have a specific client/server program for a set of common scenarios, but to have some generic client/server programs that allow a user on the client site to log into the computer at the server site and use the services available there.
- We refer to these generic client/server pairs as remote logging applications.
- One of the original remote logging protocols is TELNET (Terminal Network or Teletype Network).

- Port Number-----23.

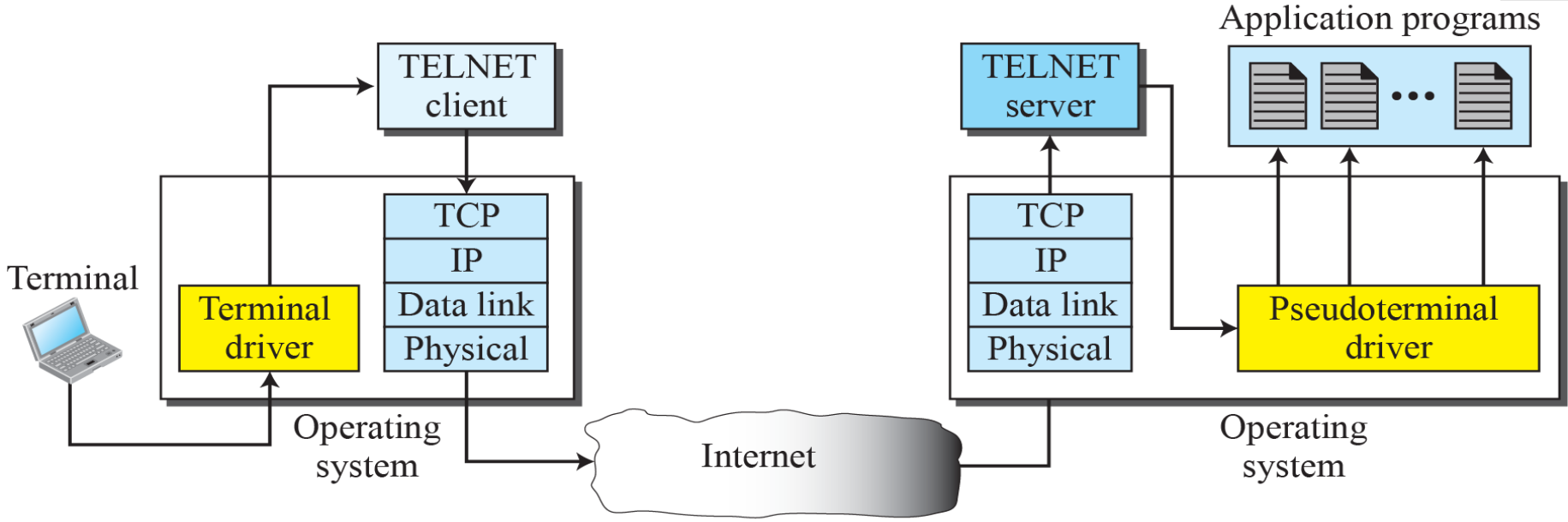
TELNET (continued)

- Local versus Remote Logging
- Network Virtual Terminal (NVT)
- Options
- User Interface

Local versus remote logging

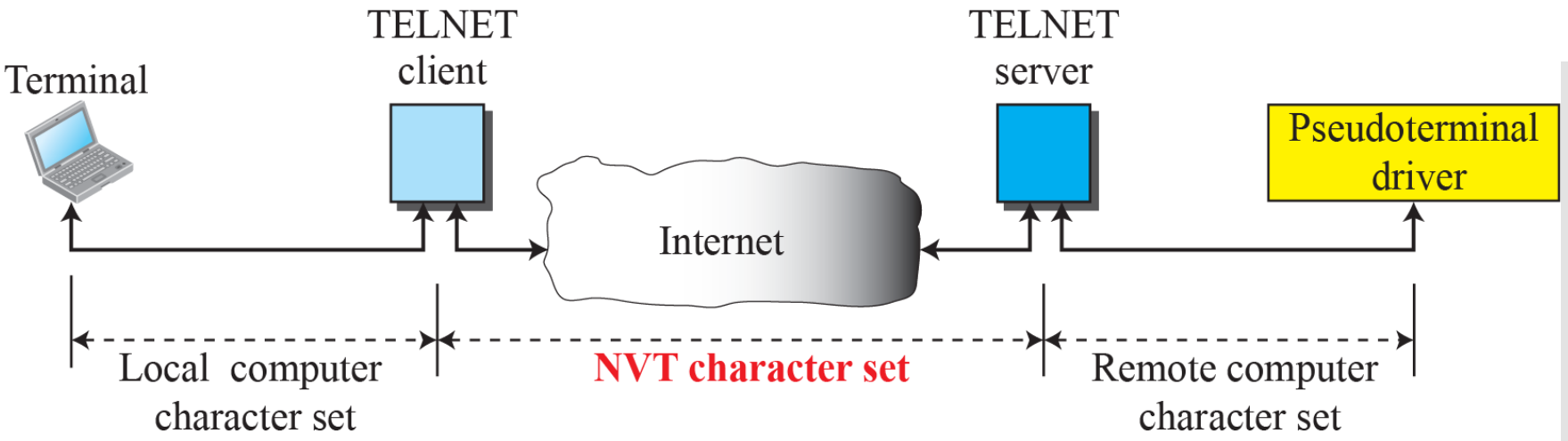


a. Local logging



b. Remote logging

NVT (Network Virtual Terminal)



NVT character format

- NVT is an acronym for Network Virtual Terminal.
- NVT is a subset of the Telnet protocol, i.e., it is as if the Telnet protocol consists of two layers: the lower layer called NVT and the upper layer called the main Telnet protocol.

Examples of interface commands

(TELNET Commands)

Command	*Meaning*	*Command*	*Meaning*
open	Connect to a remote computer	**set**	Set the operating parameters
close	Close the connection	**status**	Display the status information
display	Show the operating parameters	**send**	Send special characters
mode	Change to line or character mode	**quit**	Exit TELNET

Secure Shell (SSH) / (SSh)

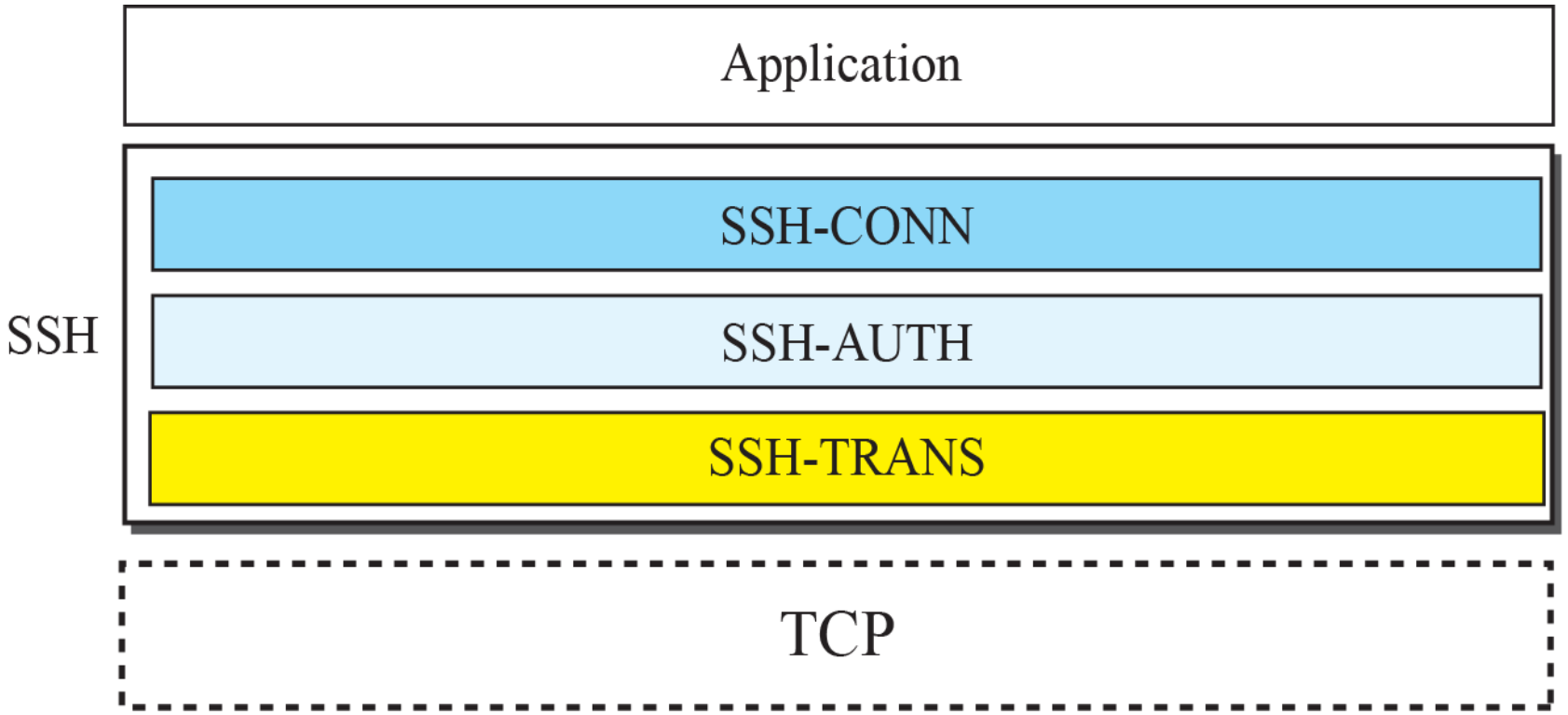
- Secure Shell (SSH) is a secure application program that can be used today for several purposes such as remote logging and file transfer, it was originally designed to replace TELNET.
- There are two versions of SSH: SSH-1 and SSH-2, which are totally incompatible. The first version, SSH-1, is now deprecated(not-approved) because of security flaws in it.
- In this section, we discuss only SSH-2.

Secure Shell (SSH) / (SSh) (continued)

- Components
  - SSH Transport-Layer Protocol (SSH-TRANS)
  - SSH Authentication Protocol (SSH-AUTH)
  - SSH Connection Protocol (SSH-CONN)
- Applications
  - SSH for Remote Logging (TErminaL NETwork)
  - SSH for File Transfer
- Port Forwarding
- Format of the SSH Packets

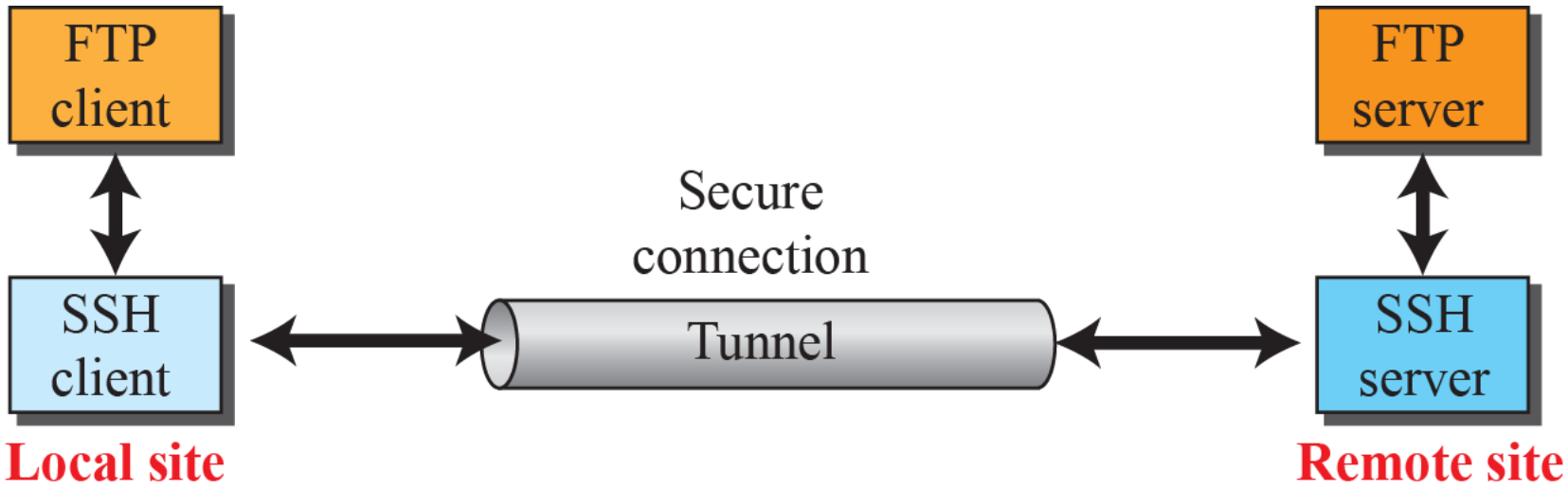
Secure Shell (SSH) / (SSh)

(Components of SSH)



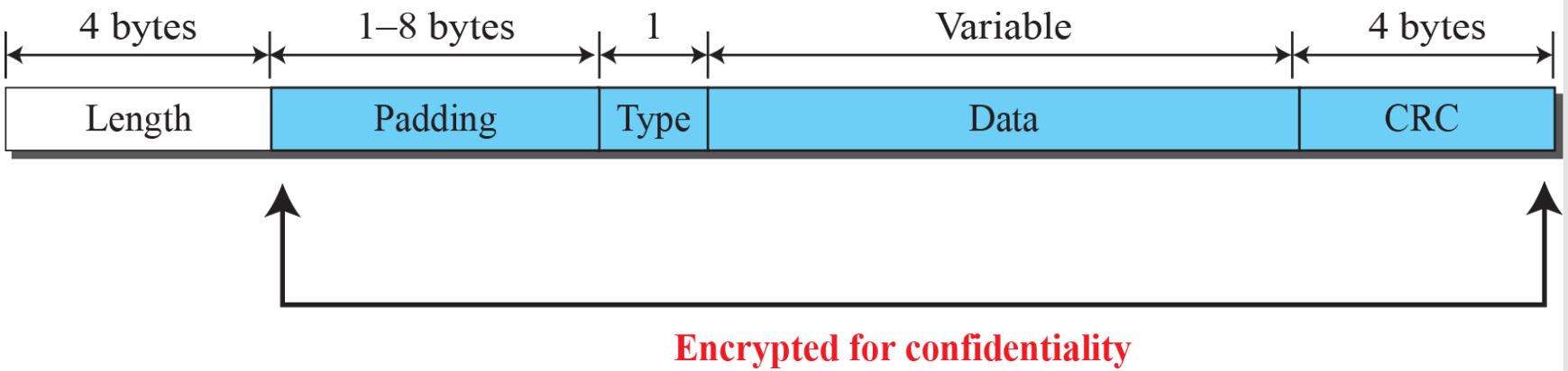
Secure Shell (SSH) / (SSh) Port Forwarding

- SFTP is not the FTP protocol running over SSH, but a different file transfer protocol developed as an extension for SSH-2.



Secure Shell (SSH) / (SSh)

SSH Packet Format

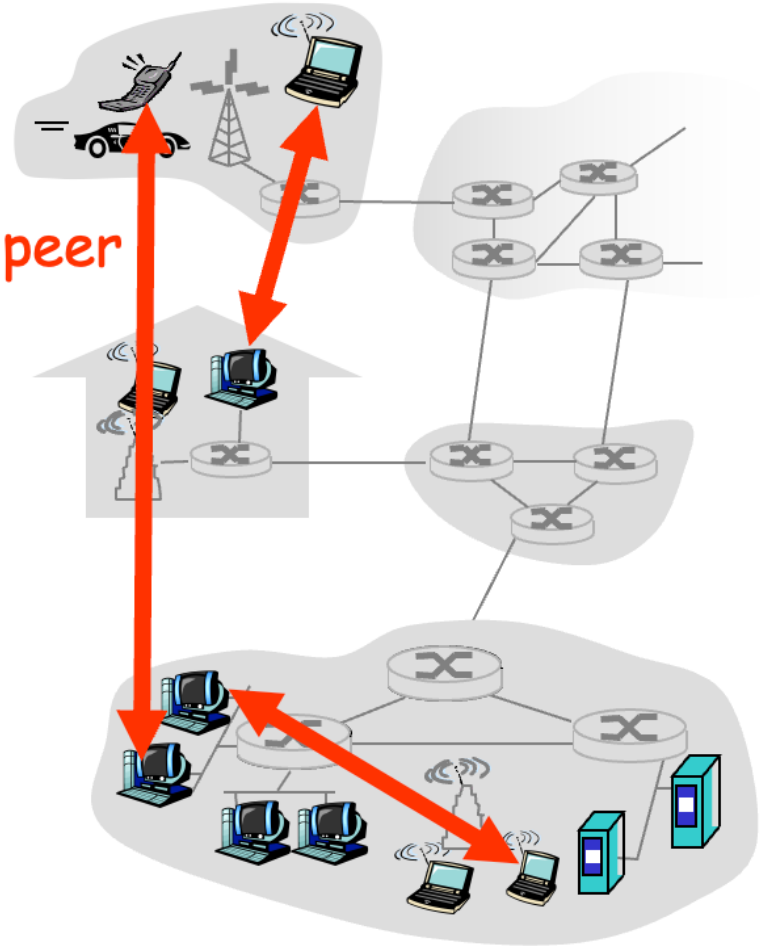


P2P Applications

Pure P2P architecture

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

- Three topics:
  - File distribution
  - Searching for information
  - Case Study: Skype

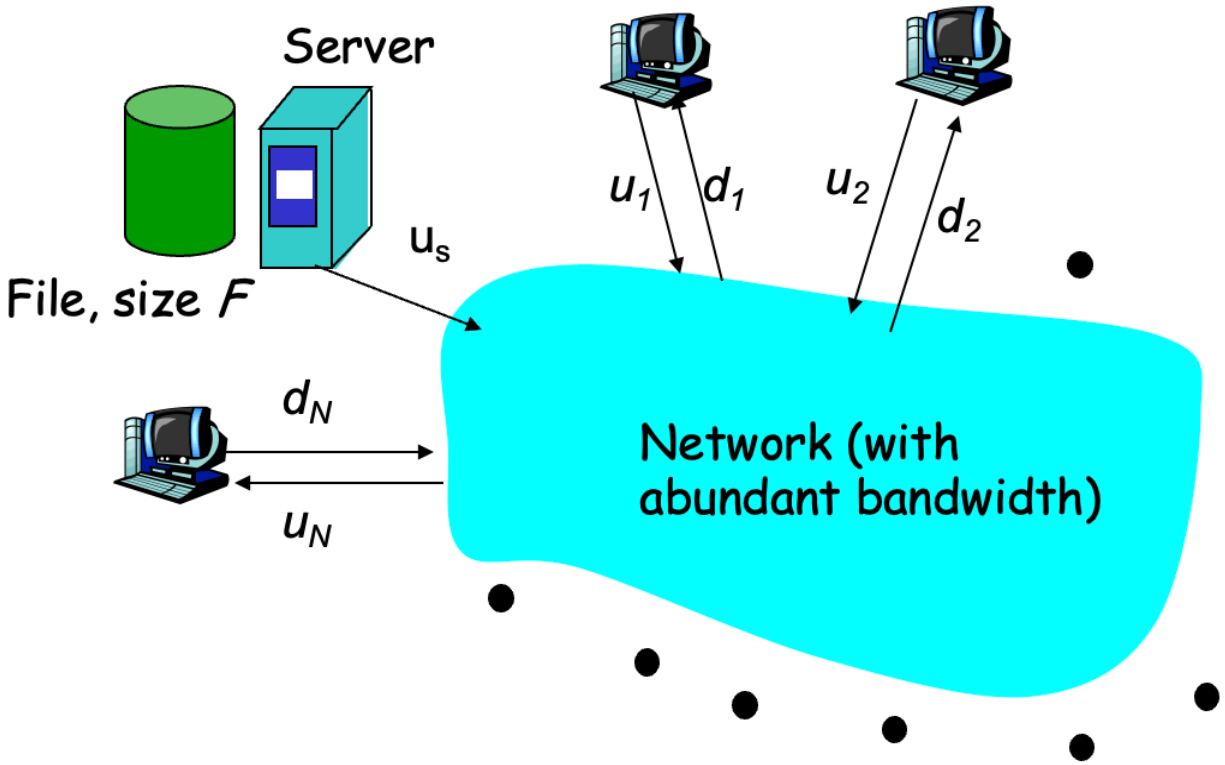


P2P Applications

(File Distribution)

File Distribution: Server-Client vs P2P

Question: How much time to distribute file from one server to *N peers*?



u_s: server upload bandwidth

u_i: peer i upload bandwidth

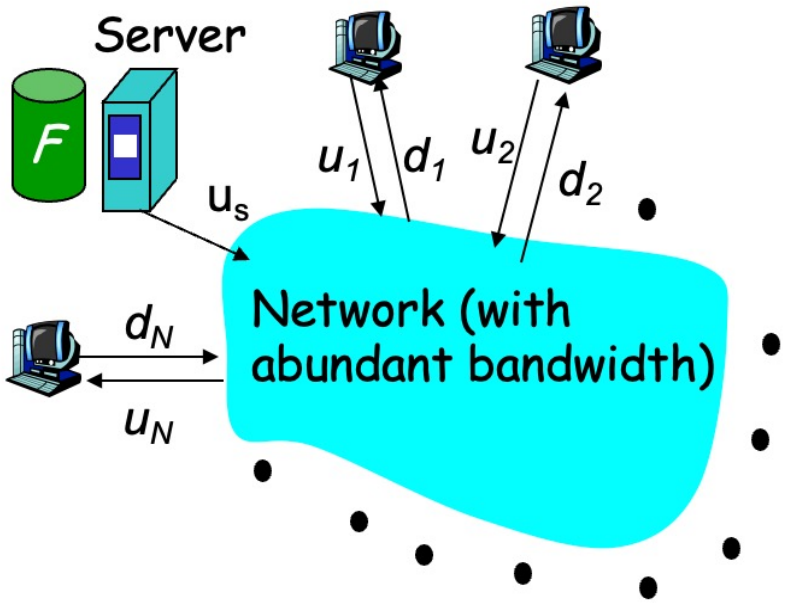
d_i: peer i download bandwidth

P2P Applications

(File Distribution)

File distribution time: server-client

- server sequentially sends N copies:
  - NF/u_s time
- client i takes F/d_i time to download



Time to distribute F to N clients using client/server approach $= d_{cs} = \max\{ NF/u_s, F/\min_i(d_i) \}$

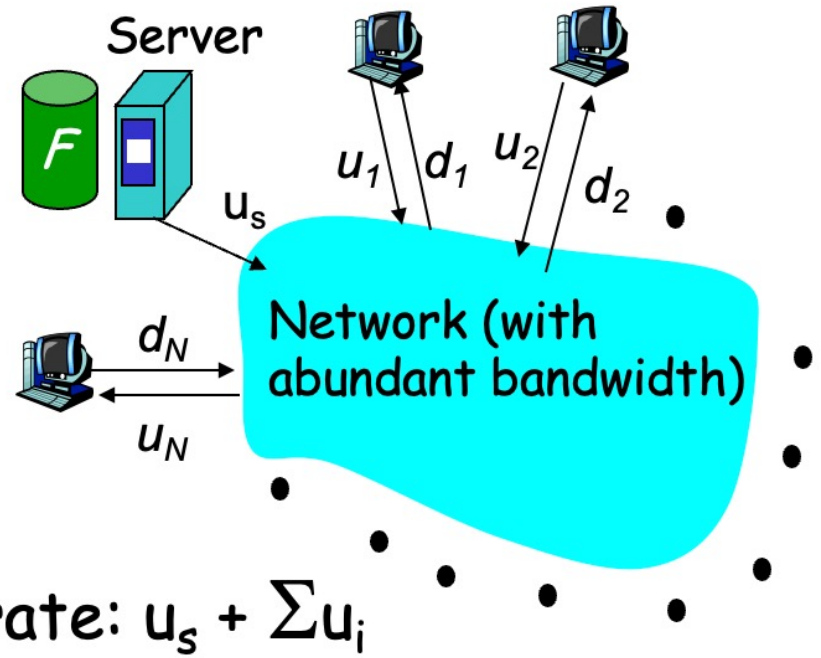
increases linearly in N (for large N)

P2P Applications

(File Distribution)

File distribution time: P2P

- server must send one copy: F/u_s time
- client i takes F/d_i time to download
- NF bits must be downloaded (aggregate)
  - fastest possible upload rate: $u_s + \Sigma u_i$



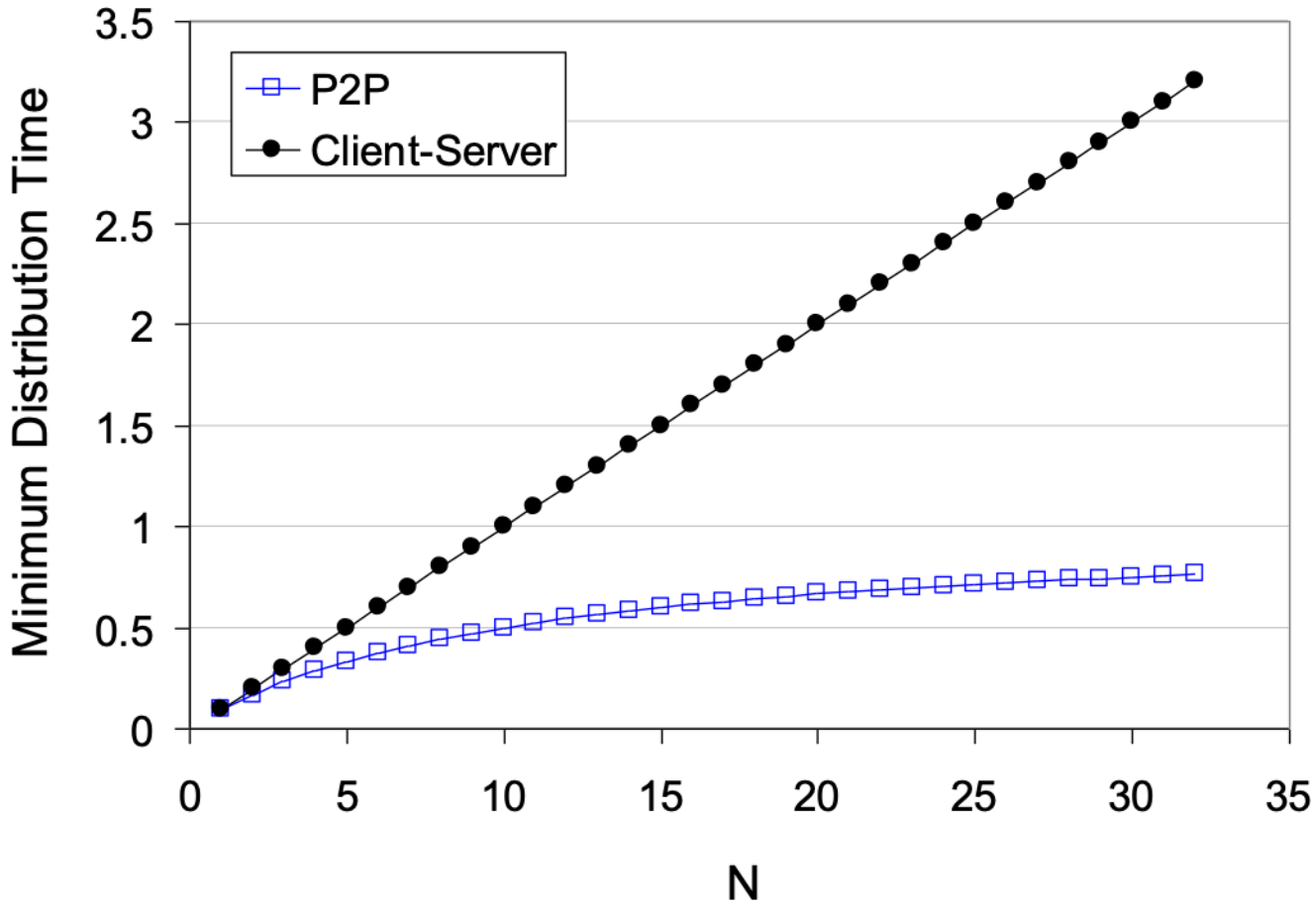
$$d_{P2P} = \max \{ F/u_s, F/\min(d_i)_i, NF/(u_s + \Sigma u_i) \}$$

P2P Applications

(File Distribution)

Server-client vs. P2P: example

Client upload rate = u, F/u = 1 hour, u_s = 10u, $d_{min} \geq u_s$

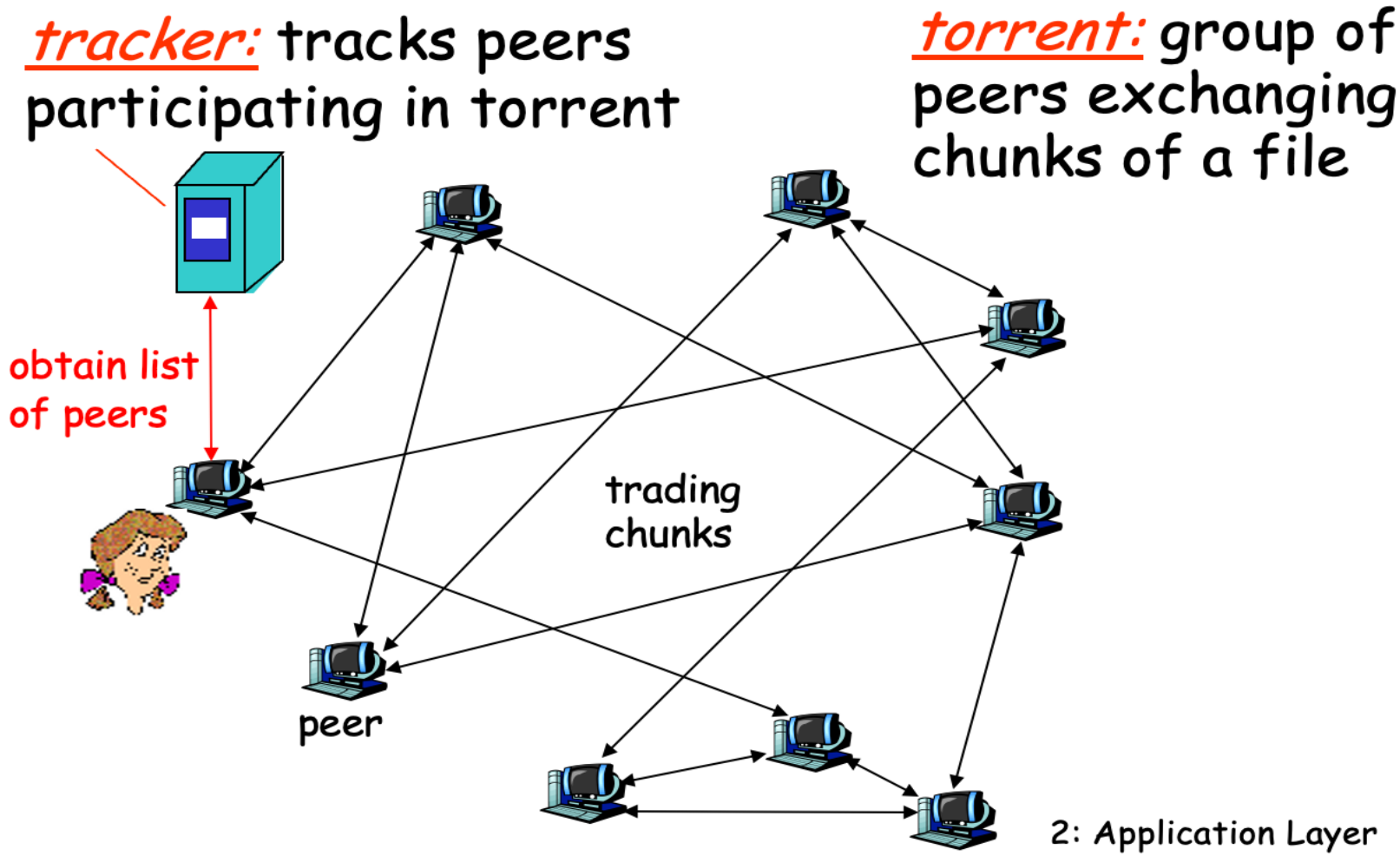


P2P Applications

(Bit Torrent)

File distribution: BitTorrent

- P2P file distribution

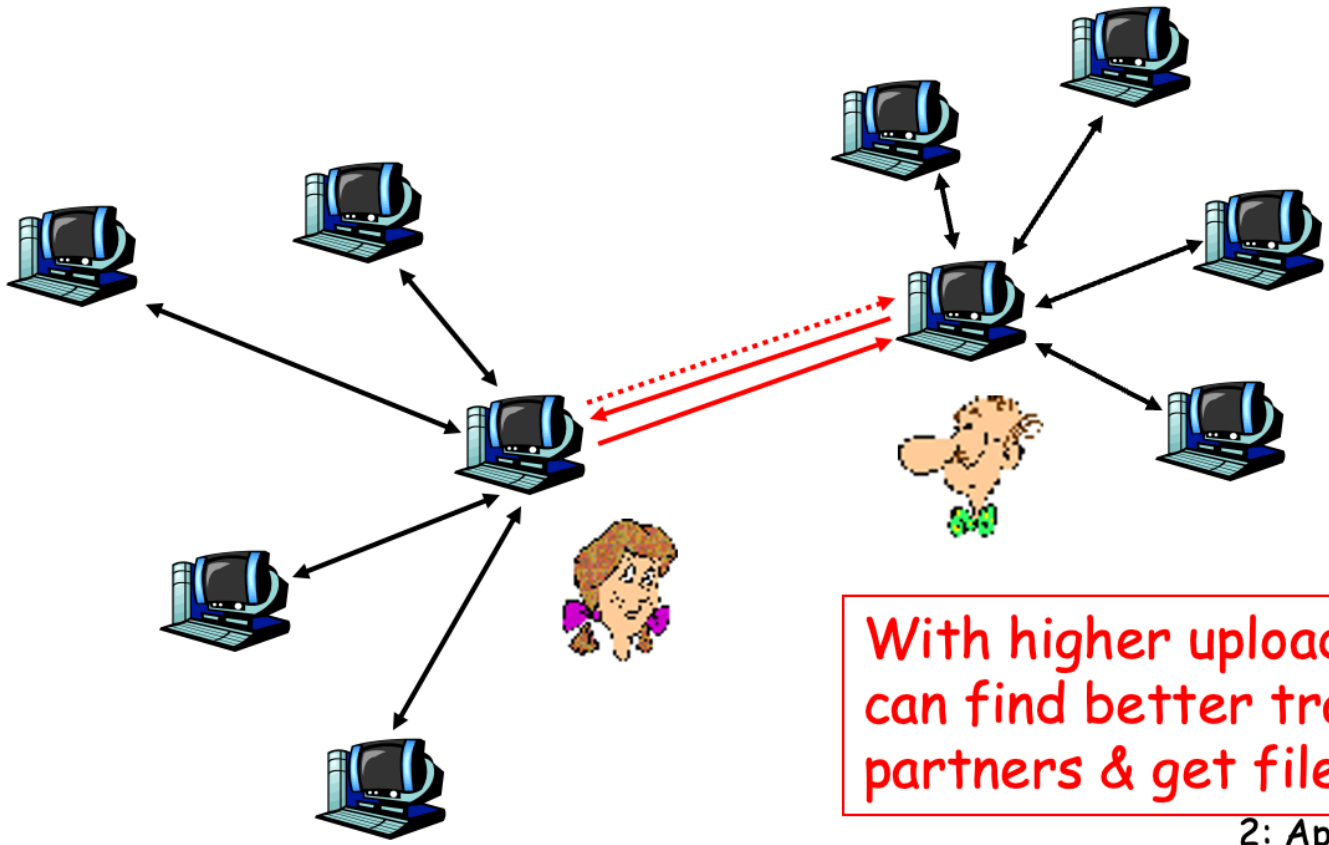


P2P Applications

(Bit Torrent)

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



With higher upload rate, can find better trading partners & get file faster!

P2P Applications

(Distributed Hash Table)

Distributed Hash Table (DHT)

- DHT = distributed P2P database
- Database has (key, value) pairs;
  - key: ss number; value: human name
  - key: content type; value: IP address
- Peers query DB with key
  - DB returns values that match the key
- Peers can also insert (key, value) peers

P2P Applications

(Distributed Hash Table)

DHT Identifiers

- Assign integer identifier to each peer in range $[0,2^n-1]$.
  - Each identifier can be represented by n bits.
- Require each key to be an integer in same range.
- To get integer keys, hash original key.
  - eg, key = h("Led Zeppelin IV")
  - This is why they call it a distributed "hash" table

P2P Applications

(Distributed Hash Table)

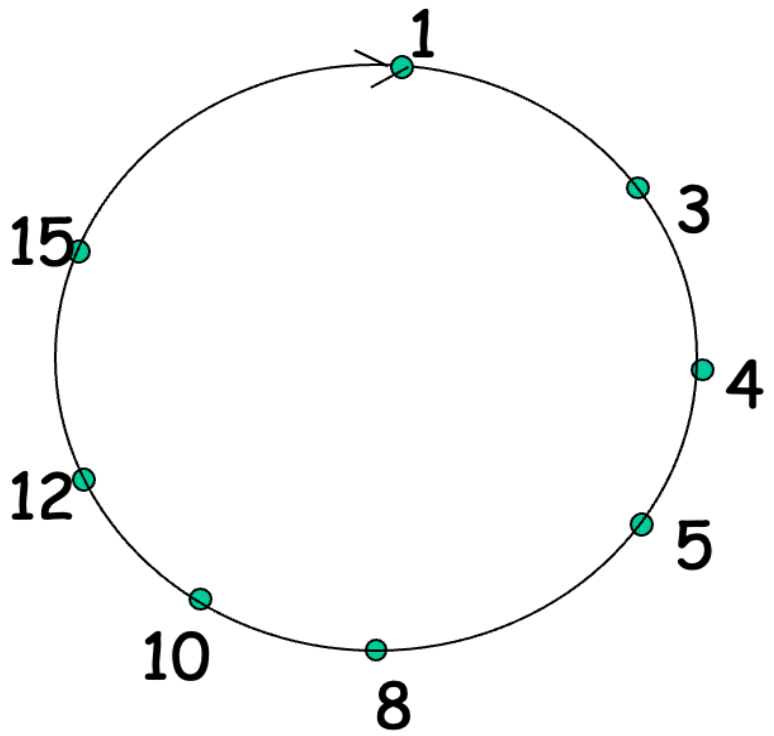
How to assign keys to peers?

- Central issue:
  - Assigning (key, value) pairs to peers.
- Rule: assign key to the peer that has the closest ID.
- Convention in lecture: closest is the immediate successor of the key.
- Ex: n=4; peers: 1,3,4,5,8,10,12,14;
  - key = 13, then successor peer = 14
  - key = 15, then successor peer = 1

P2P Applications

(Distributed Hash Table)

Circular DHT (1)



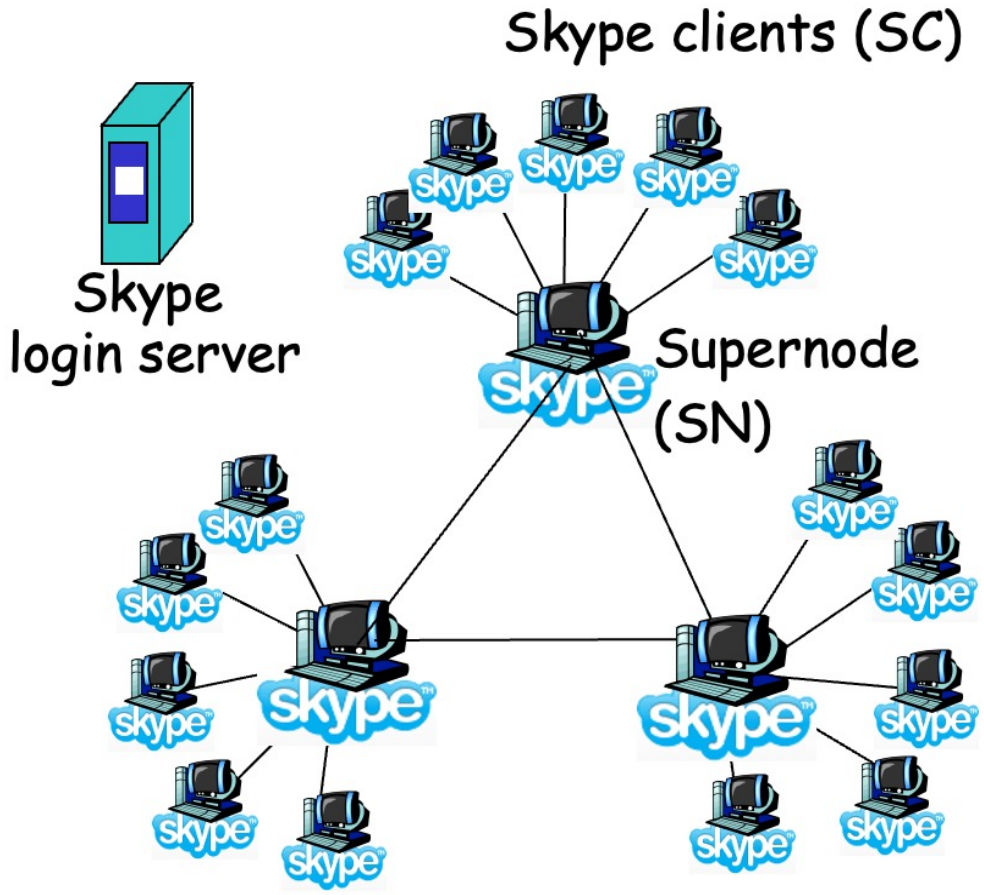
- Each peer *only* aware of immediate successor and predecessor.
- "Overlay network"

P2P Applications

(Distributed Hash Table)

P2P Case study: Skype

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SNs
- Index maps usernames to IP addresses; distributed over SNs

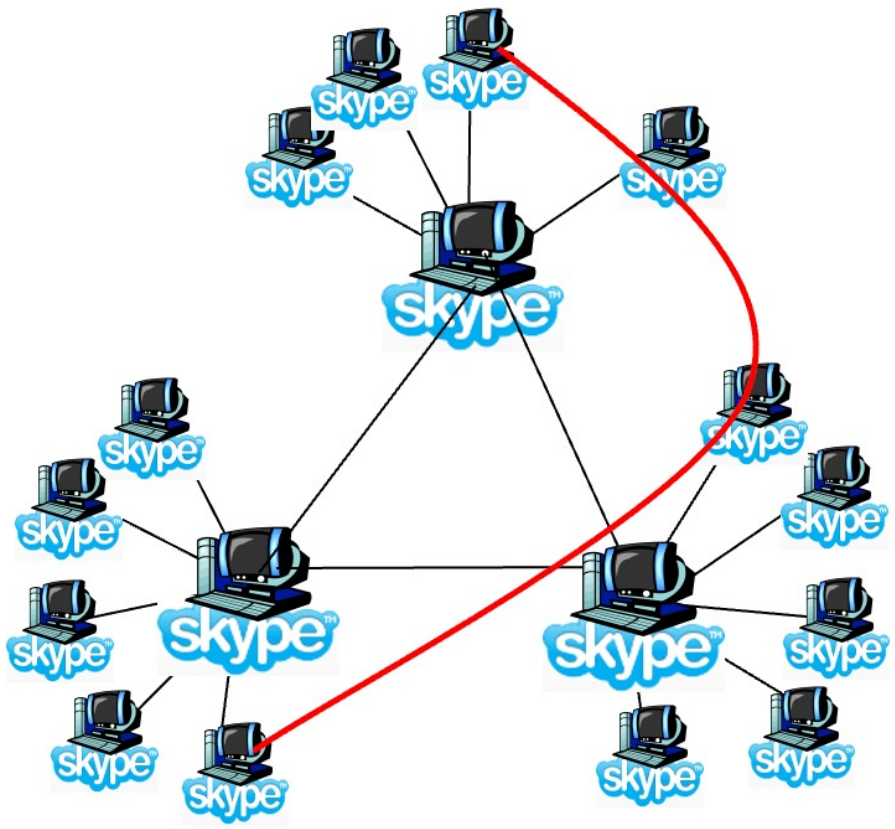


P2P Applications

(Peers as Relay)

Peers as relays

- Problem when both Alice and Bob are behind "NATs".
  - NAT prevents an outside peer from initiating a call to insider peer
- Solution:
  - Using Alice's and Bob's SNs, Relay is chosen
  - Each peer initiates session with relay.
  - Peers can now communicate through NATs via relay

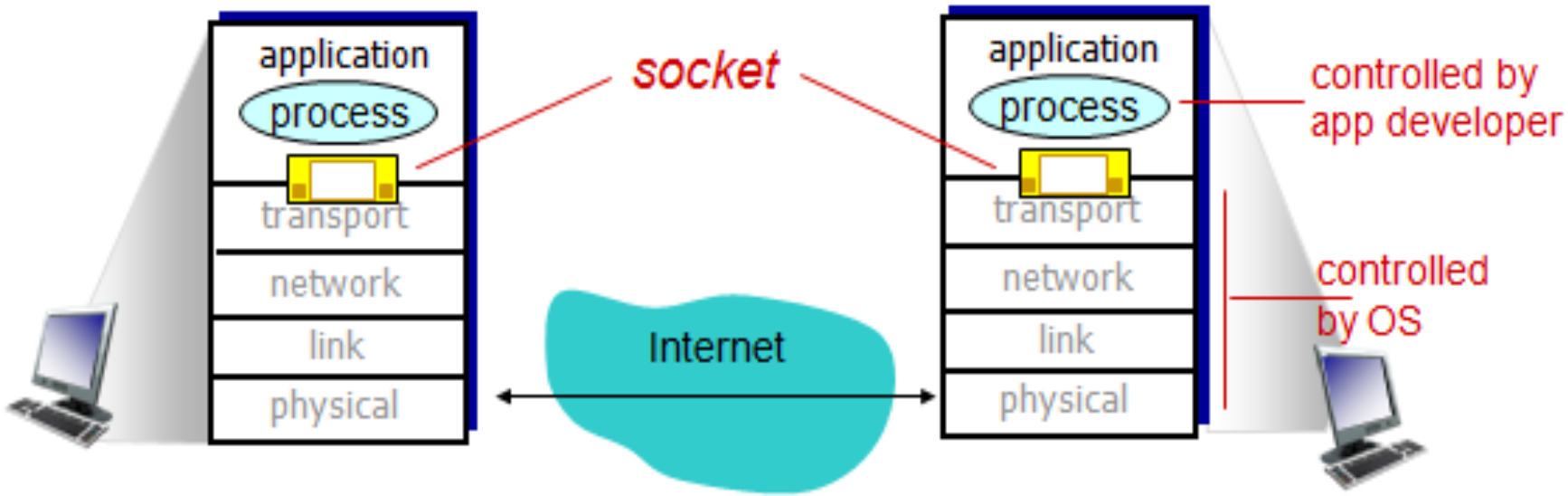


Sockets

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



Socket programming

Two socket types for two transport services:

- *UDP:* unreliable datagram
- *TCP:* reliable, byte stream-oriented

Application Example:

1. Client reads a line of characters (data) from its keyboard and sends the data to the server.
2. The server receives the data and converts characters to uppercase.
3. The server sends the modified data to the client.
4. The client receives the modified data and displays the line on its screen.

Socket programming

with TCP

Socket programming

Goal: learn how to build client/server application that communicate using sockets

Socket API

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

socket

a *host-local*, *application-created*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another application process

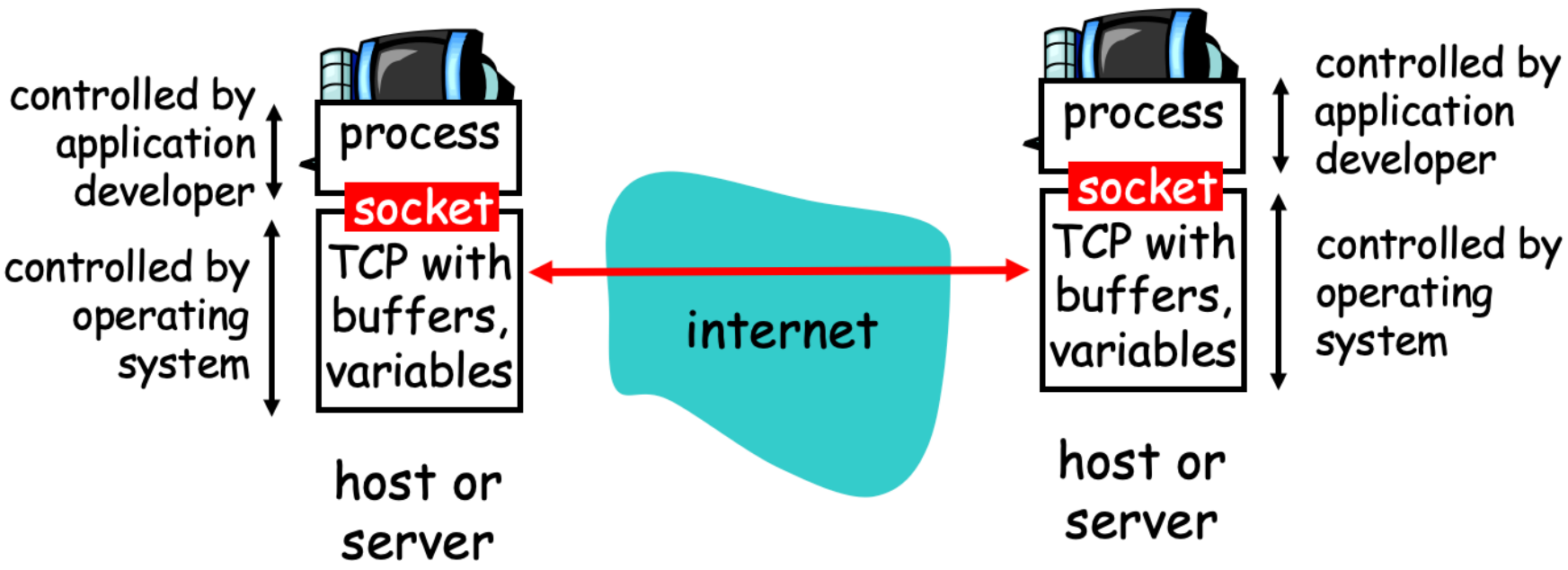
Socket programming

with TCP

Socket-programming using TCP

Socket: a door between application process and end-end-transport protocol (UCP or TCP)

TCP service: reliable transfer of **bytes** from one process to another



Socket programming

with TCP

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 client-local 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 client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

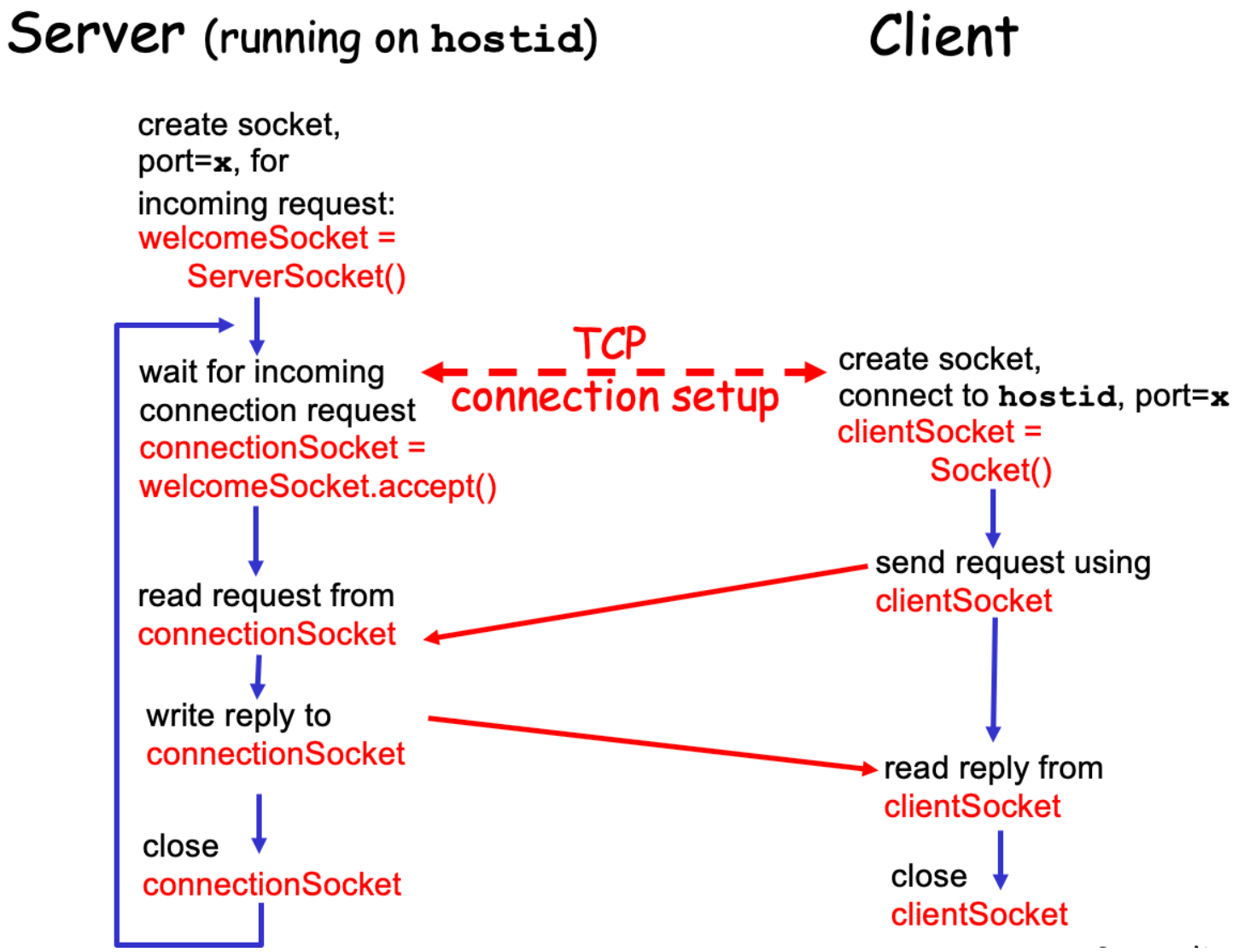
application viewpoint

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

Socket programming

with TCP

Client/server socket interaction: TCP

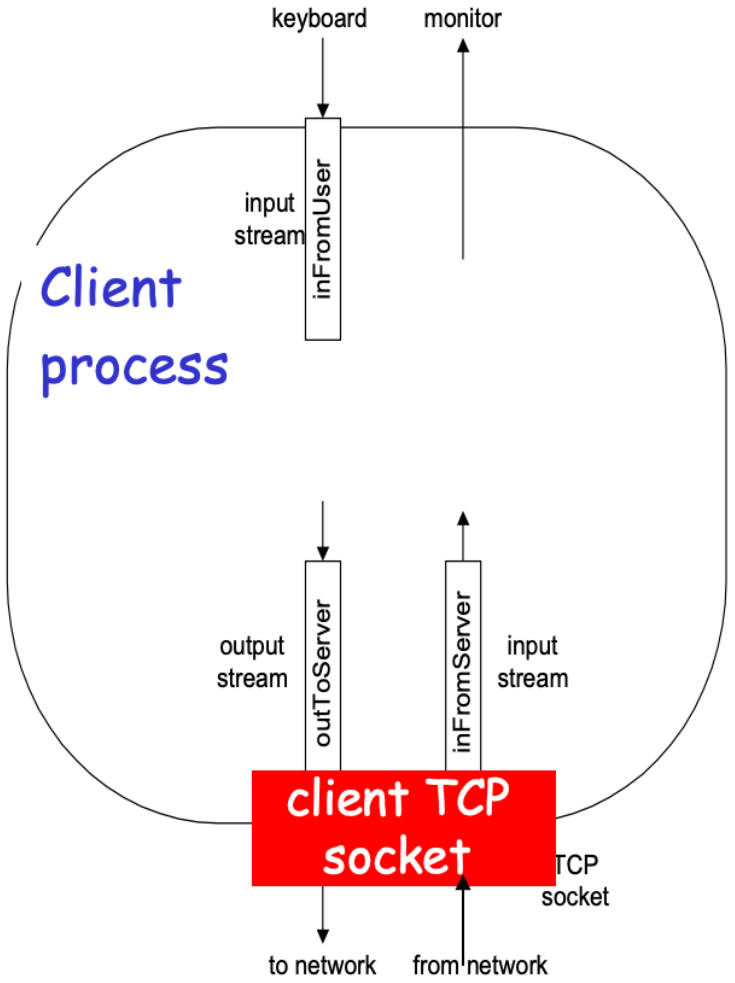


Socket programming

with TCP

Stream jargon

- A stream is a sequence of characters that flow into or out of a process.
- An input stream is attached to some input source for the process, e.g., keyboard or socket.
- An output stream is attached to an output source, e.g., monitor or socket.



Socket programming

with TCP

Socket programming with TCP

Example client-server app:

1) client reads line from standard input (`inFromUser` stream) , sends to server via socket (`outToServer` stream)
2) server reads line from socket
3) server converts line to uppercase, sends back to client
4) client reads, prints modified line from socket (`inFromServer` stream)

Socket programming

with TCP

Example: Java client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream

Create client socket, connect to server

Create output stream attached to socket

Socket programming

with TCP

Example: Java client (TCP), cont.

Create input stream attached to socket

```
BufferedReader inFromServer =
  new BufferedReader(new
  InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();
```

Send line to server

```
outToServer.writeBytes(sentence + '\n');
```

Read line from server

```
modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

    }
}
```

Socket programming

with TCP

Example: Java server (TCP)

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

       Socket connectionSocket = welcomeSocket.accept();

      BufferedReader inFromClient =
        new BufferedReader(new
        InputStreamReader(connectionSocket.getInputStream()));
```

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

Socket programming

with TCP

Example: Java server (TCP), cont

Create output stream, attached to socket

Read in line from socket

Write out line to socket

```
DataOutputStream  outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());

clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';

outToClient.writeBytes(capitalizedSentence);
            }
        }
    }
```

End of while loop, loop back and wait for another client connection

Socket programming

with UDP

Socket programming *with UDP*

UDP: no "connection" between client and server

- no handshaking
- sender explicitly attaches IP address and port of destination to each packet
- server must extract IP address, port of sender from received packet

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

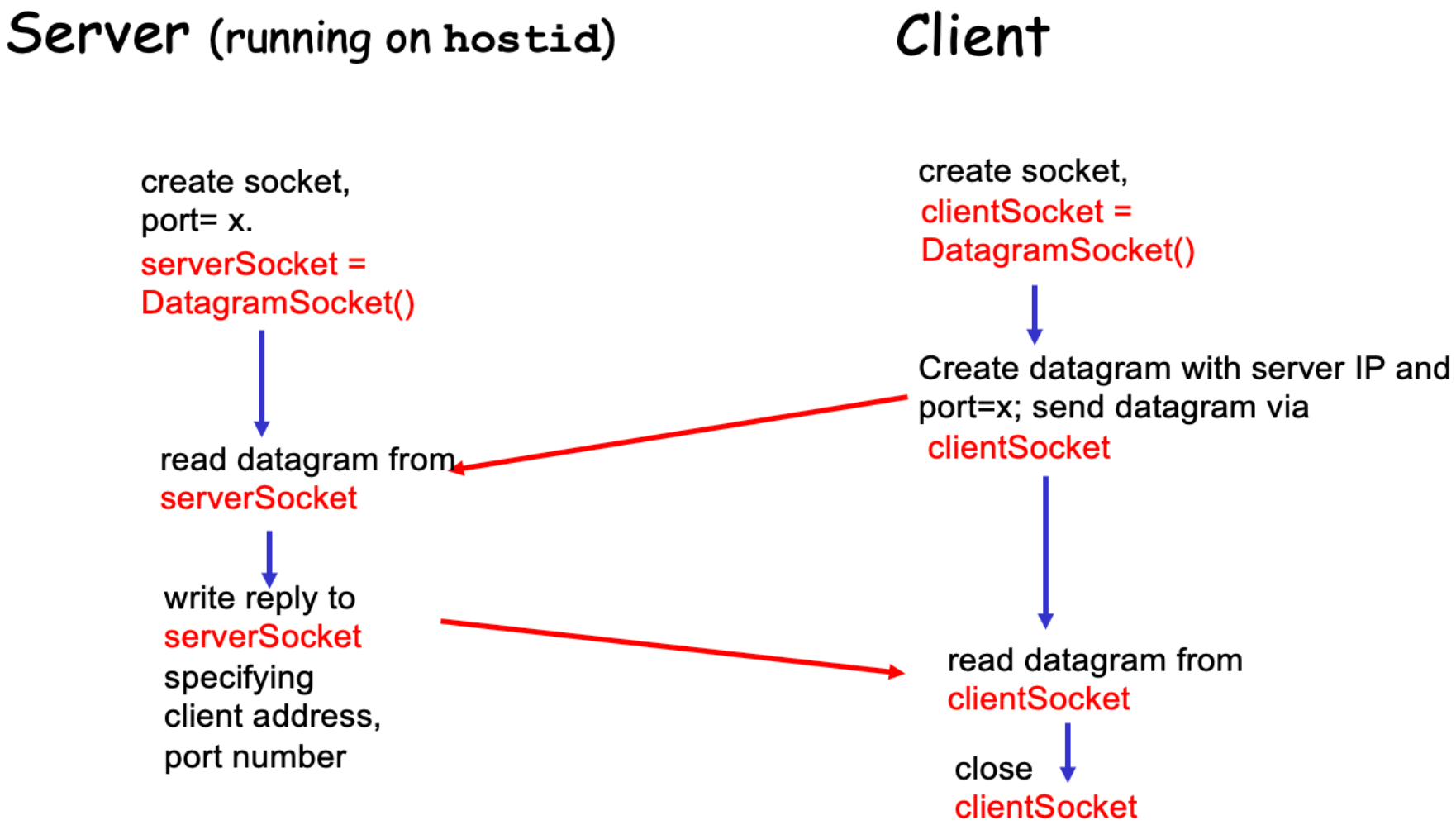
application viewpoint

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

Socket programming

with UDP

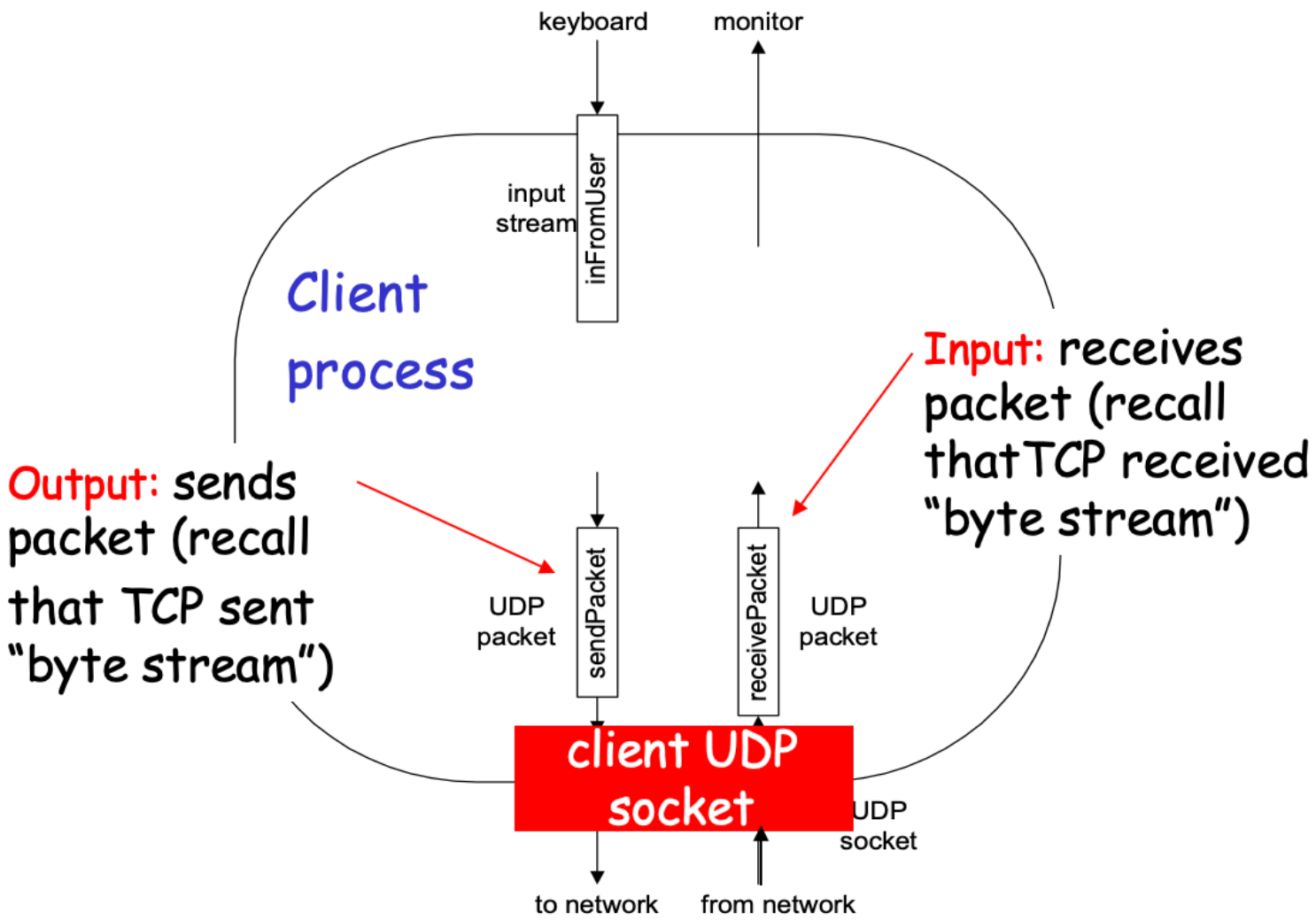
Client/server socket interaction: UDP



Socket programming

with UDP

Example: Java client (UDP)



Socket programming

with UDP

Example: Java client (UDP)

Create input stream

Create client socket

Translate hostname to IP address using DNS

```
import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String args[]) throws Exception
    {

BufferedReader inFromUser =
  new BufferedReader(new InputStreamReader(System.in));

DatagramSocket clientSocket = new DatagramSocket();

InetAddress IPAddress = InetAddress.getByName("hostname");

byte[] sendData = new byte[1024];
byte[] receiveData = new byte[1024];

String sentence = inFromUser.readLine();

sendData = sentence.getBytes();
```

Socket programming

with UDP

Example: Java client (UDP), cont.

Create datagram with data-to-send, length, IP addr, port

Send datagram to server

Read datagram from server

```
DatagramPacket sendPacket =
  new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

clientSocket.send(sendPacket);

DatagramPacket receivePacket =
   new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

String modifiedSentence =
    new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
}

}
```

Socket programming

with UDP

Example: Java server (UDP)

```
import java.io.*;
import java.net.*;

class UDPServer {
  public static void main(String args[]) throws Exception
   {
     DatagramSocket serverSocket = new DatagramSocket(9876);

     byte[] receiveData = new byte[1024];
     byte[] sendData  = new byte[1024];

     while(true)
      {
        DatagramPacket receivePacket =
           new DatagramPacket(receiveData, receiveData.length);
        serverSocket.receive(receivePacket);
```

Create datagram socket at port 9876

Create space for received datagram

Receive datagram

Socket programming

with UDP

Example: Java server (UDP), cont

```
                String sentence = new String(receivePacket.getData());

Get IP addr
port #, of
sender          InetAddress IPAddress = receivePacket.getAddress();

                int port = receivePacket.getPort();

                        String capitalizedSentence = sentence.toUpperCase();

                sendData = capitalizedSentence.getBytes();

Create datagram
to send to client  DatagramPacket sendPacket =
                     new DatagramPacket(sendData, sendData.length, IPAddress,
                                        port);

Write out
datagram
to socket       serverSocket.send(sendPacket);
                  }
                }
                }
```

End of while loop, loop back and wait for another datagram