Chapter 07
Application Layer

Associate Prof. Hong Zheng （郑宏）
Computer School
Beijing Institute of Technology

Key Points

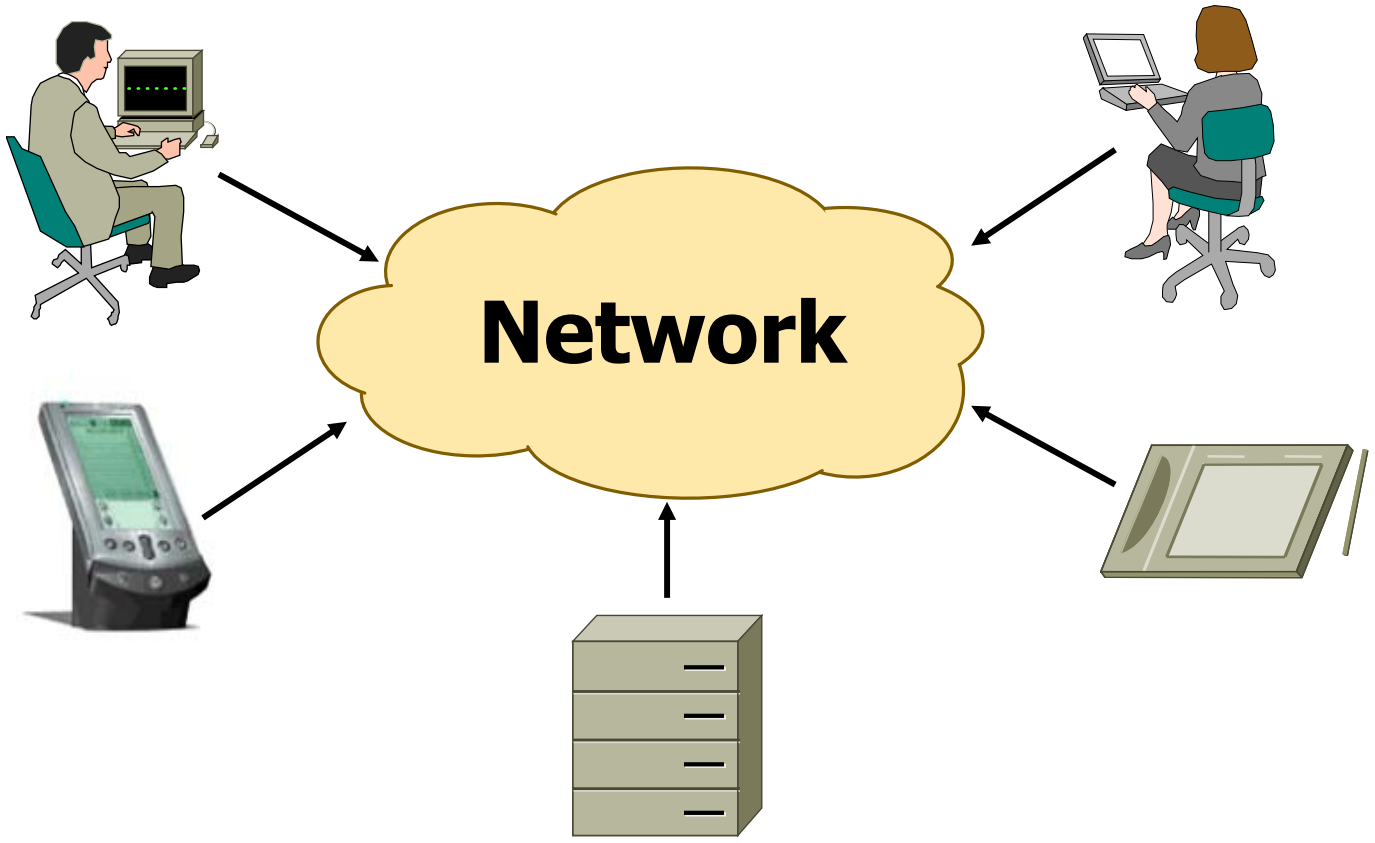
DNS	Name space, Name Servers, Name Resolution.	熟练掌握
FTP	Two connections, two processes, and two ports.	熟练掌握
E-MAIL	Components, SMTP, POP3, IMAP, Mail format, MIME, Web-based E-Mail.	熟练掌握
Web and HTTP	Web Page, Hyperlink, URL, HTTP, Nonpersistent and persistent HTTP, HTTP message format,	熟练掌握
DHCP	Message Type, Operations	熟练掌握

Chapter 7: Roadmap

- Application and Application Layer
- C/S and P2P Application Architectures
- DNS
- FTP
- E-Mail and SMTP/POP/IMAP
- WWW and HTTP
- DHCP

Network Applications

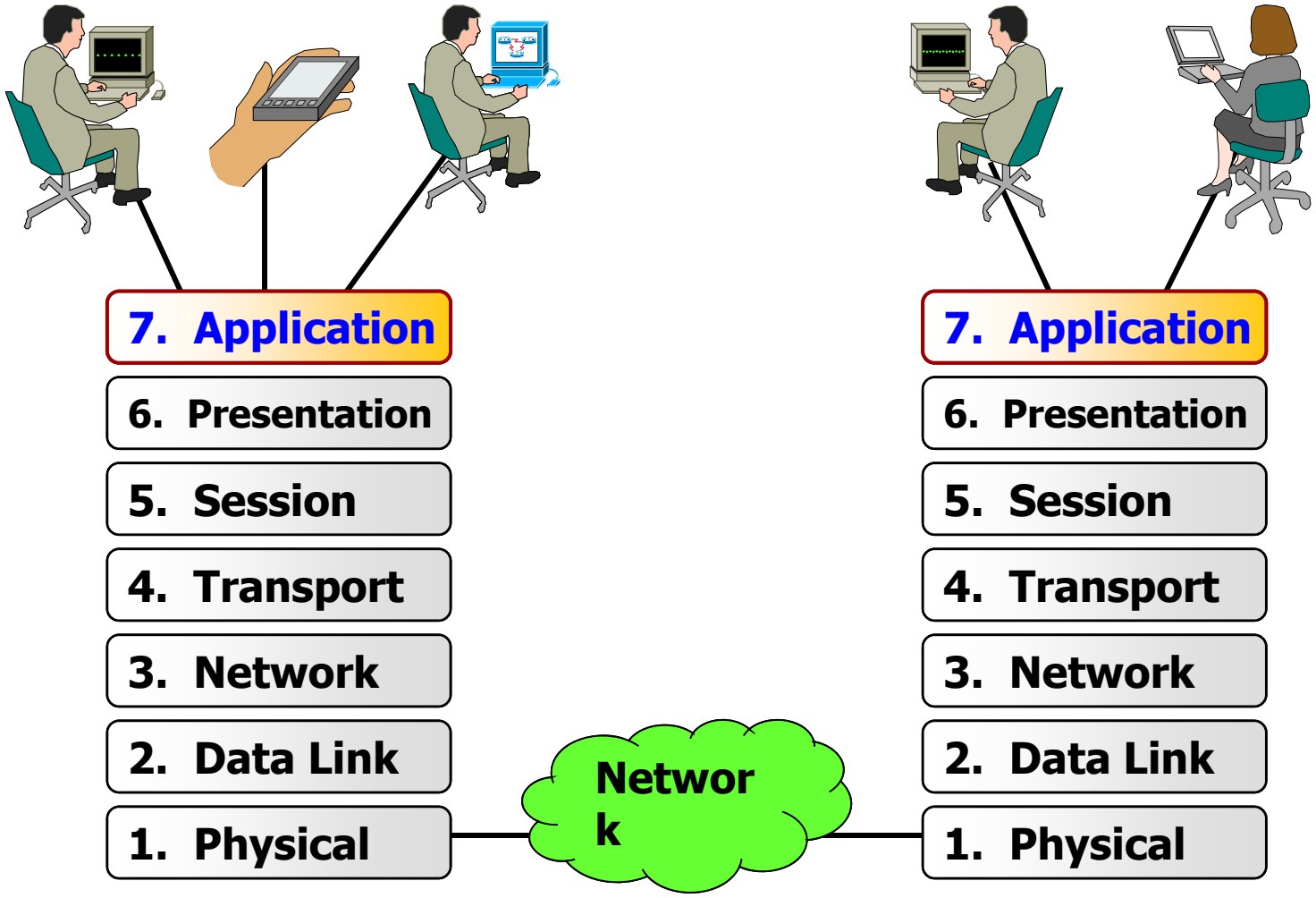
- Communicating, distributed processes
  - Run on (different) end systems
  - Communicate over network



Application Layer

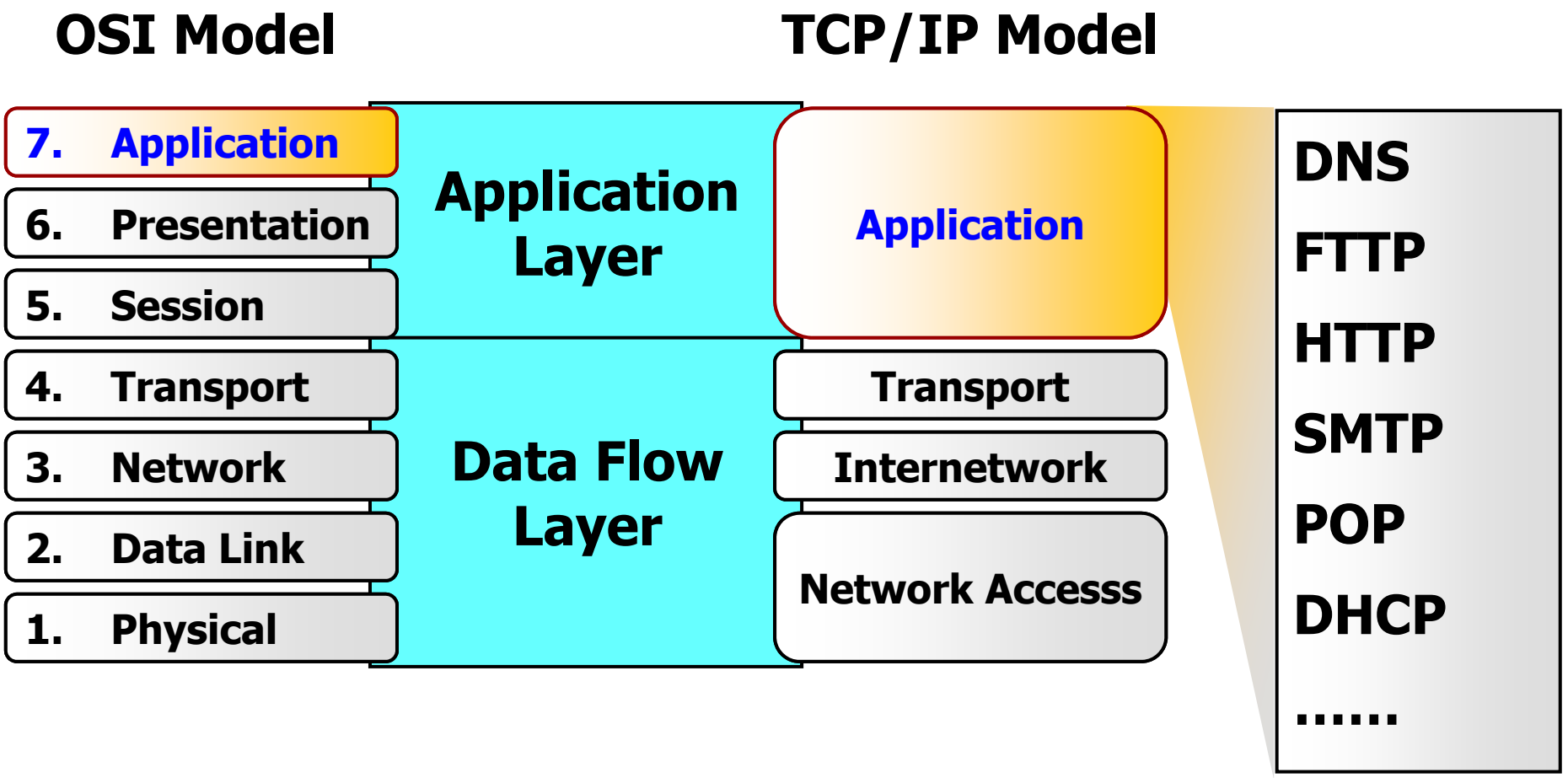
- **Support network applications**
  - Define messages exchanged by apps and actions taken
  - Implementing services by using the service provided by the lower layer, i.e., the transport layer
- **The Interface Between Application (human) and Data Networks**

Application Layer



Application layer provide the interface to the network

Application Layer



Application Protocols Tailored to the Application

- **Telnet:** interacting with account on remote machine
- **DNS:** resolving Internet domain names to IP addresses
- **FTP:** copying files between accounts
- **SMTP:** sending e-mail to a remote mail server
- **HTTP:** satisfying requests based on a global URL

Comparing the Protocols

- **Commands and replies**
  - Telnet sends commands in binary, whereas the other protocols are text based
  - Many of the protocols have similar request methods and response codes
- **Data types**
  - Telnet, FTP, and SMTP transmit **text data** in standard U.S. 7-bit ASCII
  - FTP also supports transfer of data in **binary** form
  - SMTP uses **MIME** standard for sending non-text data
  - HTTP incorporates some key aspects of **MIME** (e.g., classification of data formats)

Comparing the Protocols (Cont.)

- **Transport**
  - Telnet, FTP, SMTP, and HTTP all depend on reliable transport protocol
  - Telnet, SMTP, and HTTP use a single TCP connection
  - … but FTP has separate control and data connections
- **State**
  - In Telnet, FTP, and SMTP, the server retains information about the session with the client
  - E.g., FTP server remembers client's current directory
  - In contrast, HTTP servers are stateless

Reflecting on Application-Layer Protocols

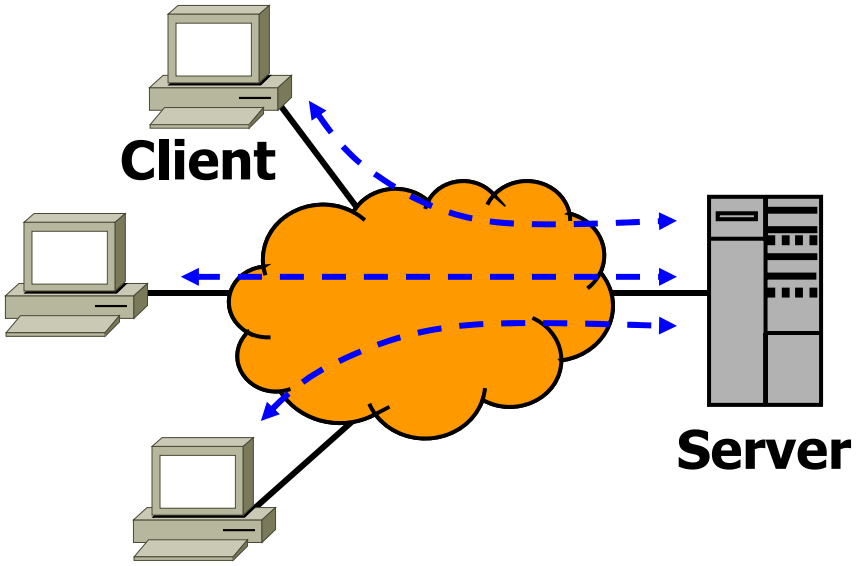
- **Protocols are tailored to the applications**
  - Each protocol is customized to a specific need
- **Protocols have many key similarities**
  - Each new protocol was influenced by the previous ones
  - New protocols commonly borrow from the older ones
- **Protocols depend on same underlying substrate**
  - Ordered reliable stream of bytes (i.e., TCP)
  - Domain Name System (DNS)
- **Relevance of the protocol standards process**
  - Important for interoperability across implementations
  - Yet, not necessary if same party writes all of the software
  - …which is increasingly common (e.g., P2P software)

Chapter 7: Roadmap

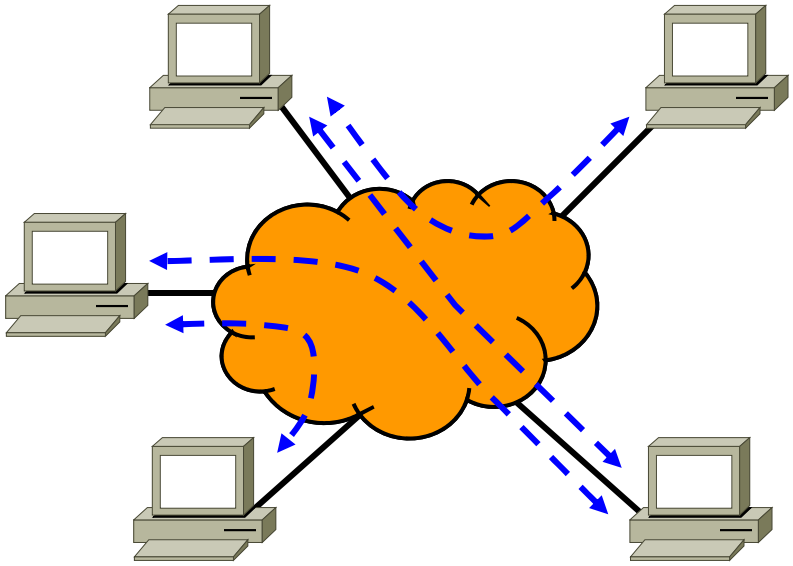
- Application and Application Layer
- C/S and P2P Application Architectures
- DNS
- FTP
- E-Mail and SMTP/POP/IMAP
- WWW and HTTP
- DHCP

Application Architectures

- Client-Server (C/S)
- Peer-to-Peer (P2P)
- Hybrid of client-server and P2P



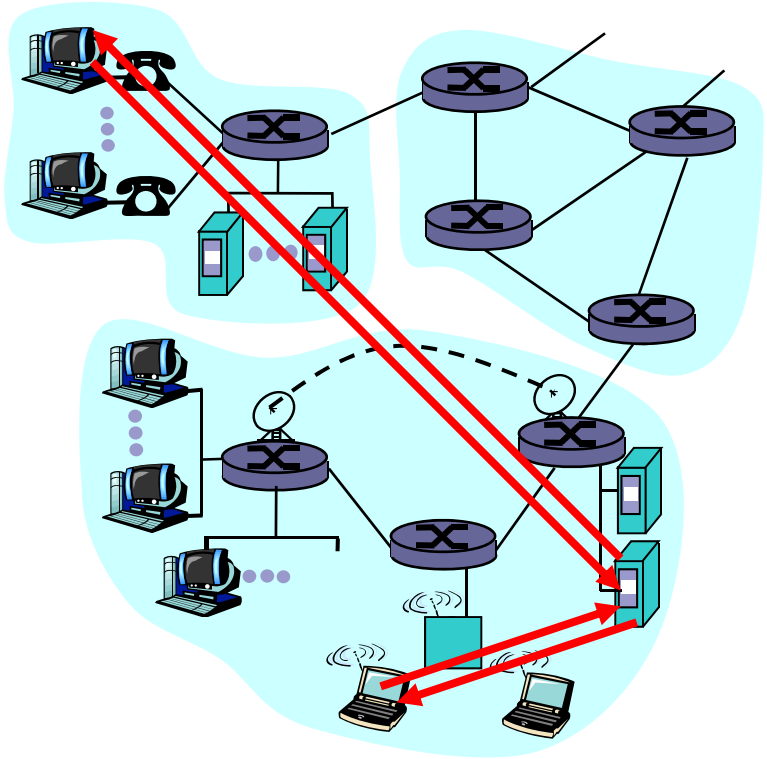
Client/Server



Peer-to-Peer

Client-Server Architecture

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



server:

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

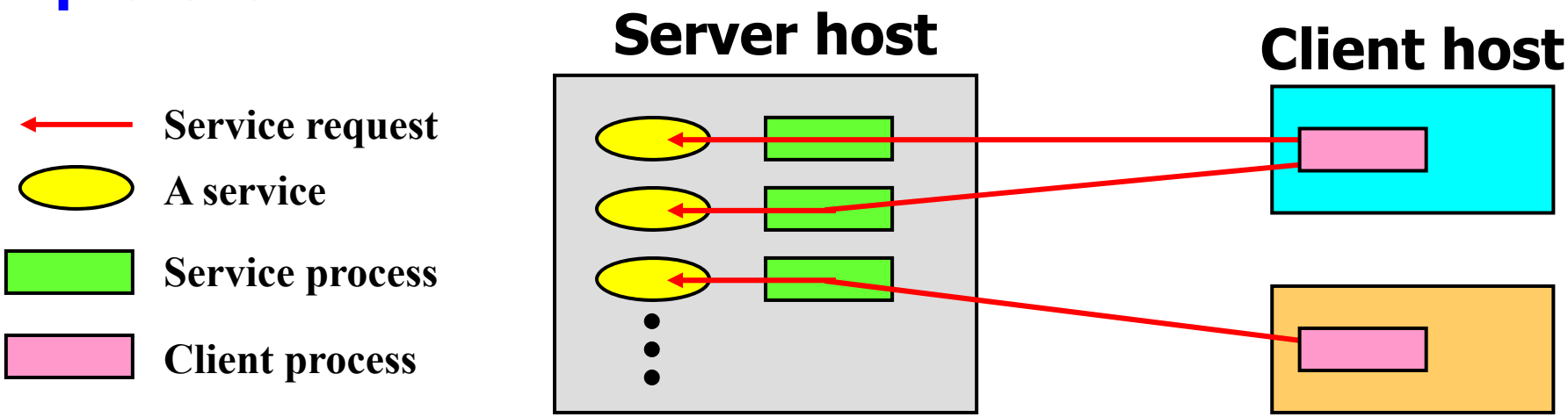
clients:

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

The Client-Server Architecture

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

- A server process, running on a server host, provides access to a service.
- A client process, running on a client host, accesses the service via the server process.
- The interaction of the process proceeds according to a protocol.

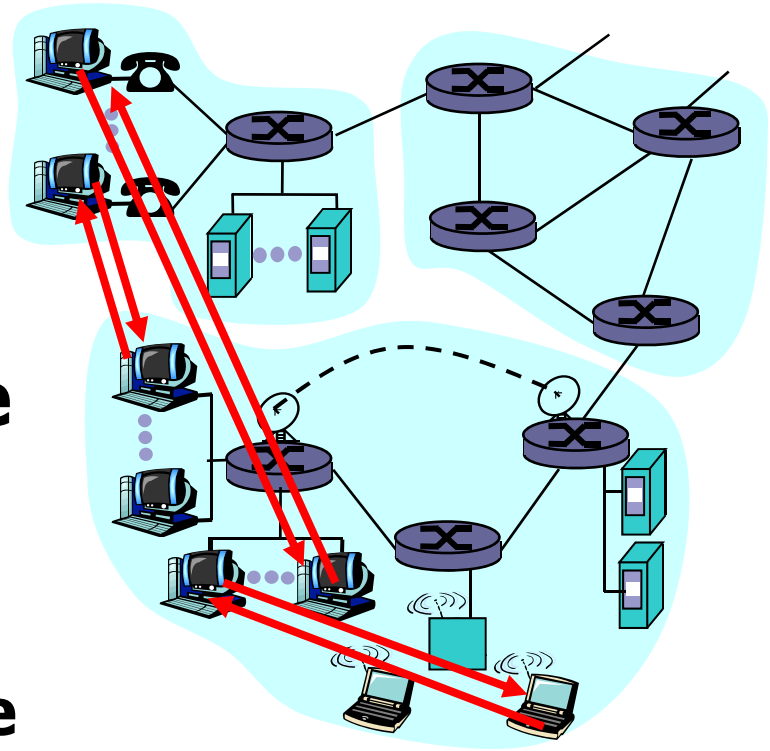


The Client-Server Architecture

- Clients request data or services from servers
- Very successful model
  - WWW (HTTP), FTP, Web services, etc.
- Limitations:
  - Scalability is hard to achieve
  - Presents a single point of failure
  - Requires administration
  - Unused resources at the network edge

P2P Architecture

- All nodes are both clients and servers
  - Provide and consume data
  - Any node can initiate a connection
- No centralized data source
  - "The ultimate form of democracy on the Internet"
  - "The ultimate threat to copy-right protection on the Internet"



Highly scalable but difficult to manage

P2P Architecture

- P2P networks generate more traffic than any other internet application
- 2/3 of all bandwidth on some backbones

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)

Hybrid of Client-Server and P2P

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 Ccommunicating

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

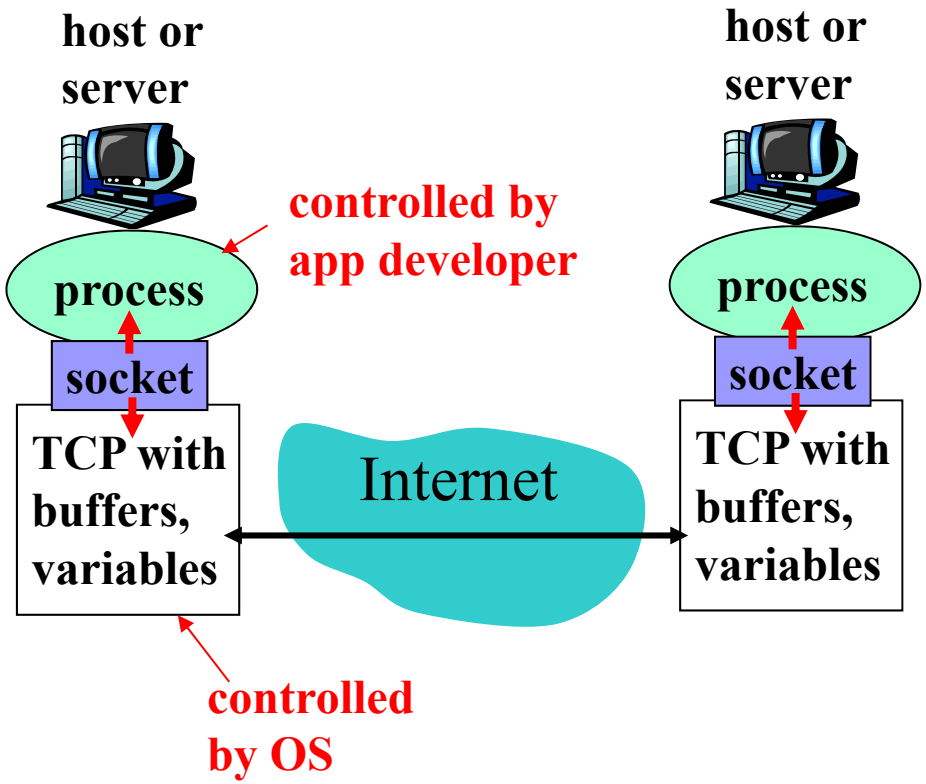
Server process: process that waits to be contacted

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

Sockets

- Process sends/receives messages to/from its socket

- A socket consists of a host IP address and a port



How does an Application Access the Transport Service?

API: application programming interface

- Defines interface between application and transport layer (lots more on this later)
  - (1) choice of transport protocol;
  - (2) ability to fix a few parameters
- Example: Socket API
  - sometimes called "Berkeley sockets" acknowledging their heritage from Berkeley Unix
  - two processes communicate by sending data into socket, reading data out of socket

App-layer protocol defines

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

Public-domain protocols:

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

Proprietary protocols:

- e.g., Skype

What transport service does an app need?

Data loss

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

Timing

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

Bandwidth

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

Transport service requirements of common apps

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

Internet transport protocols services

TCP service:

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

UDP service:

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

Q: why bother?
Why is there a UDP?

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	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Vonage,Dialpad)	typically UDP

Chapter 7: Roadmap

- Application and Application Layer
- C/S and P2P Application Architectures
- DNS
- FTP
- E-Mail and SMTP/POP/IMAP
- WWW and HTTP
- DHCP

DNS

- Internet has one global system of addressing: IP
- And one global system of naming: DNS
- DNS:

  a hierarchical, domain-based naming scheme and a distributed database system, for mapping host names and e-mail destinations to IP addresses.

Host Names vs. IP addresses

- Host names
  - Mnemonic name appreciated by humans
  - Provide little (if any) information about location
  - Examples: www.cnn.com and ftp.eurocom.fr
- IP addresses
  - Numerical address appreciated by routers
  - Fixed length, binary number
  - Hierarchical, related to host location
  - Examples: 64.236.16.20 and 193.30.227.161

Host Names vs. IP addresses

- **Name could map to**
  - **multiple IP addresses.**
    - **www.cnn.com to multiple replicas of the Web site**
  - **different addresses in different places**
    - **Address of a nearby copy of the Web site**
- **Multiple names for the same address**
  - **aliases like ee.myit.edu and cs.myit.edu**

Mapping from Names to Addresses

- **Solution 1: Local File (Original name to address mapping)**
  - **/etc/hosts.txt:** simply listed the name and IP address of every host on the network.
  - **SRI kept main copy**
  - **Downloaded regularly**
  - **Flat namespace**
- **Problem: doesn't scale**
  - **Many more downloads**
  - **Many more updates (manually)**

Mapping from Names to Addresses

- **Solution 2: Central server**
  - One place where all mappings are stored
  - All queries go to the central server
- **Many practical problems:**
  - Single point of failure
  - High traffic volume
  - Distant centralized database
  - Single point of update
  - Does not scale

Need a distributed, hierarchical collection of servers

Mapping from Names to Addresses

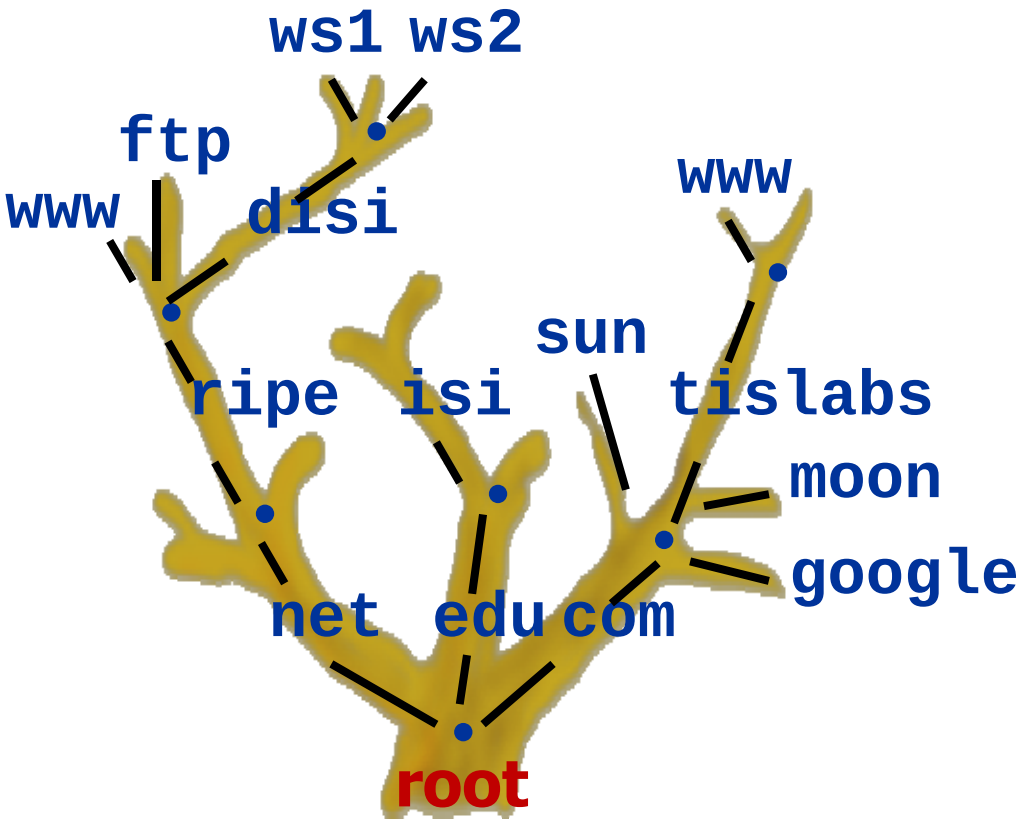
- **Solution 3: Domain Name System (DNS)**
  - **distributed database** implemented in hierarchy of many **name servers**
  - **application-layer protocol** host, routers, name servers to communicate to **resolve** names (name/address translation)
    - complexity at network's "edge"
    - **Core** Internet function, implemented as application-layer protocol

DNS: Domain Name System

- RFC 1034, 1035
- Comprised of 3 components
  - Name space
  - Servers making that name space available
  - Resolvers (clients) which query the servers about the name space

The DNS Name Space

Hierarchical naming

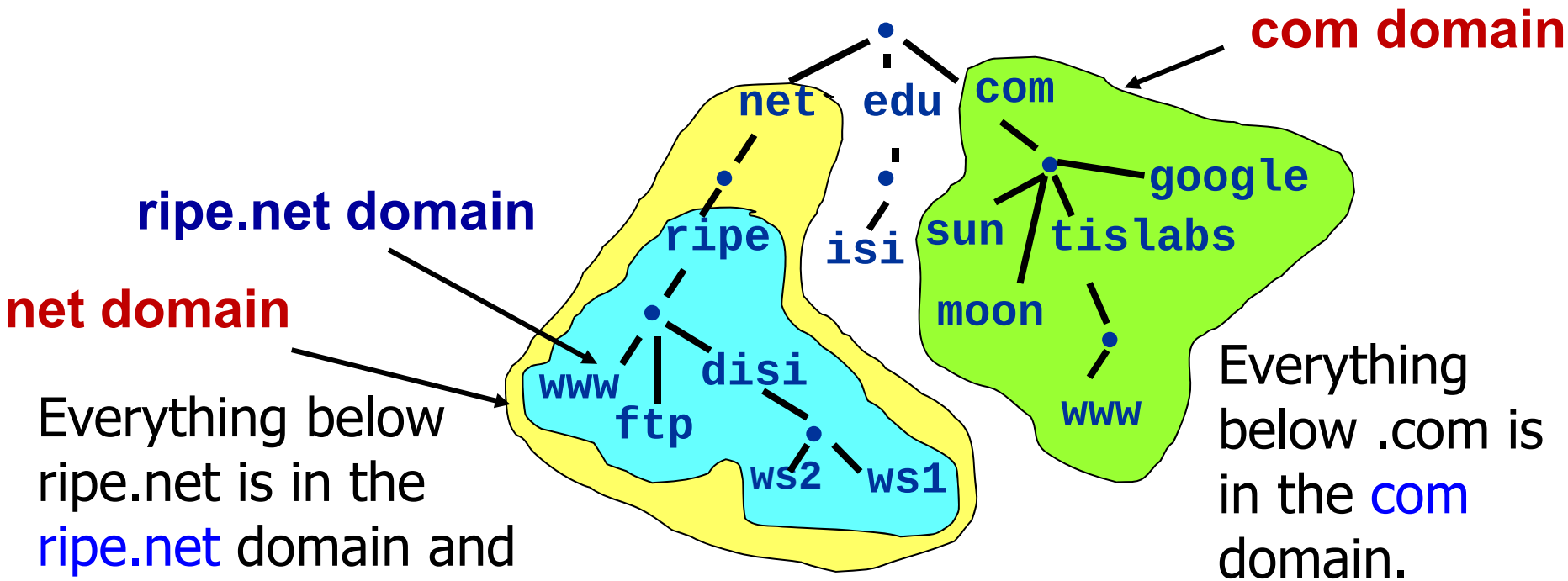


- Domain names can be mapped to a tree.
- A **domain name** is a path from a leaf node up to the root.
- A **domain** is a sub-tree in the domain name space.
- New branches at the ‘dots’
- No restriction to the amount of branches.

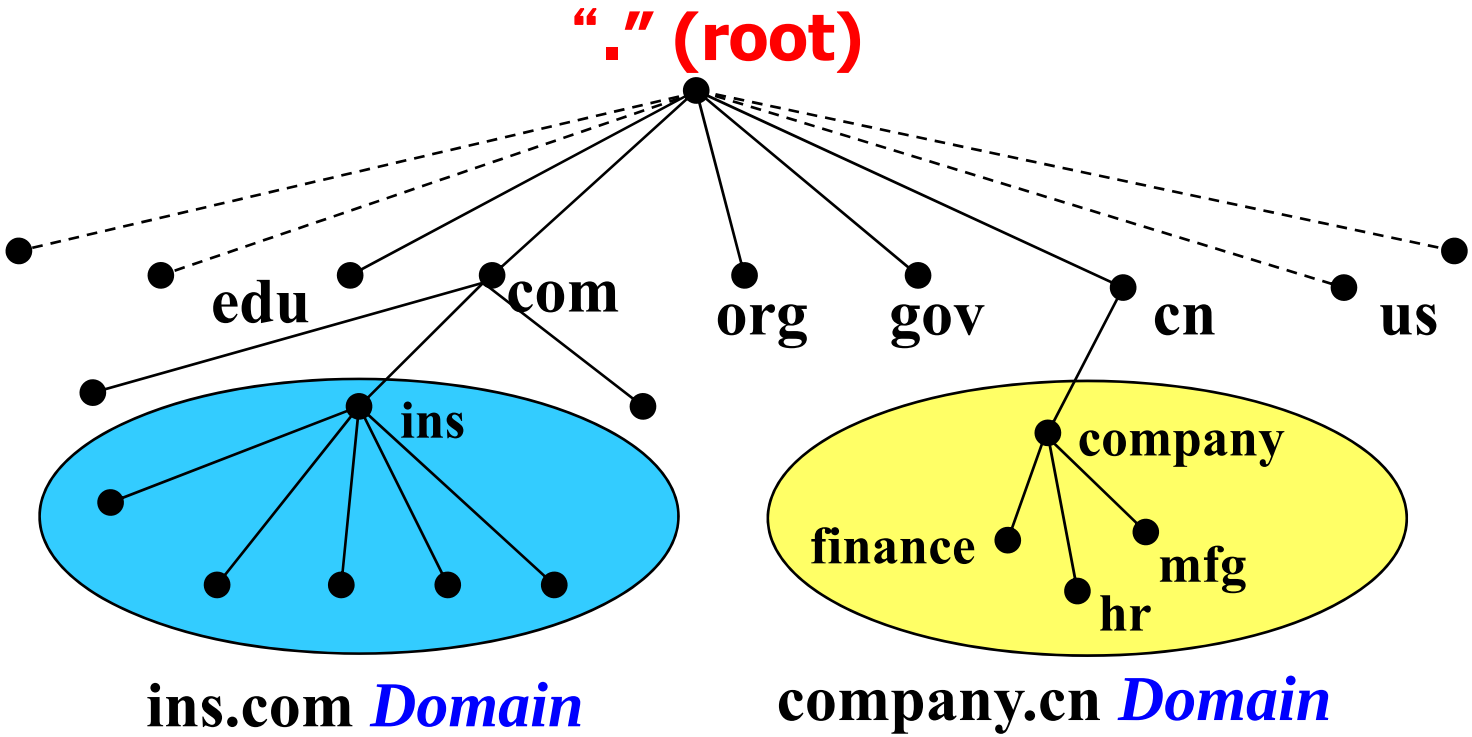
Concept: Domains

- **Domains are "namespaces"**

A domain is a sub-tree in the domain name space.



Name Space Structure



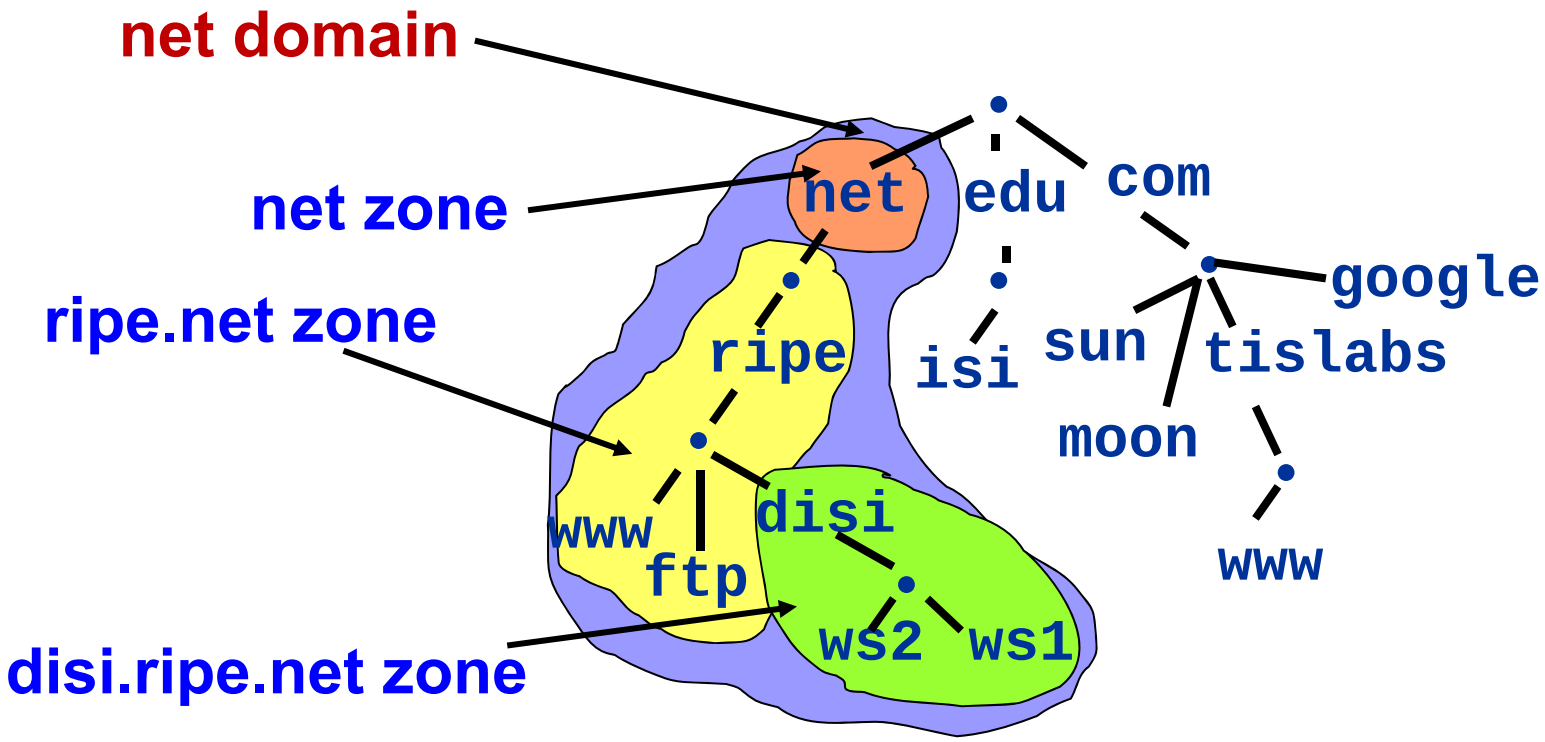
A domain name is a path from a leaf node up to the root -- a unique name.

Fully Qualified Domain Name (FQDN)

www.mybit.edu.cn

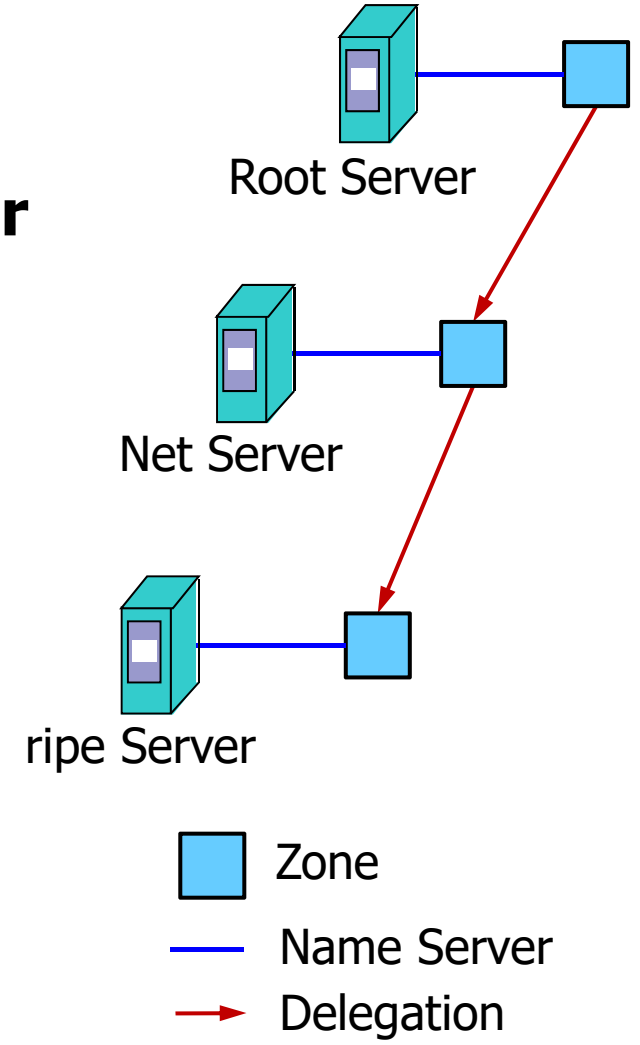
Zones and Delegations

- Zones: administrative namespaces



Zones and Delegations

- **Delegation:**
  - **Delegating a zone to a legal entity who are responsible for maintaining the DNS zone.**
  - **Setting a responsible name server for a zone.**
- **After delegating, the name servers can independently manage host names and subdomains.**
- **Authority is delegated from a parent and to a child**

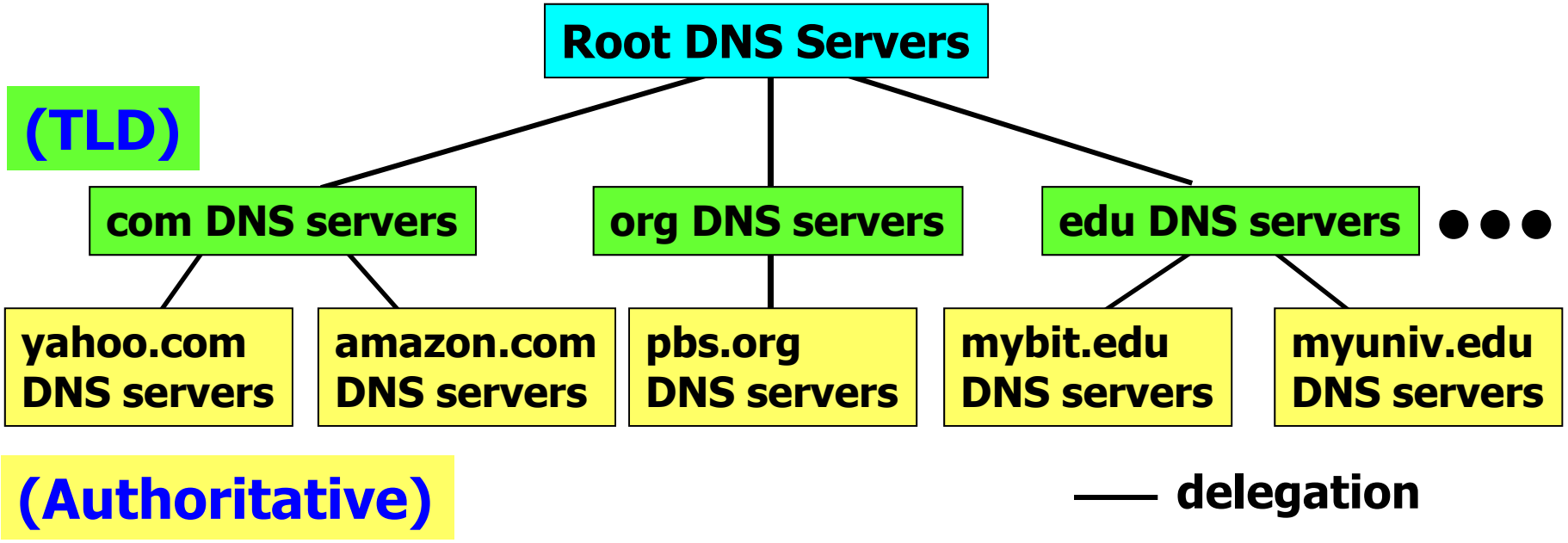


Name Server: DNS Server

- Maintains information (in a zone file) about Domains in Zones.
- Answer 'DNS' questions.
  - Questions are called "queries"
  - Answers are called "responses"
- A single DNS server can be configured with many zones

Name Server

- Hierarchy of DNS servers
  - Root servers
  - Top-level domain (TLD) servers
  - Authoritative DNS servers

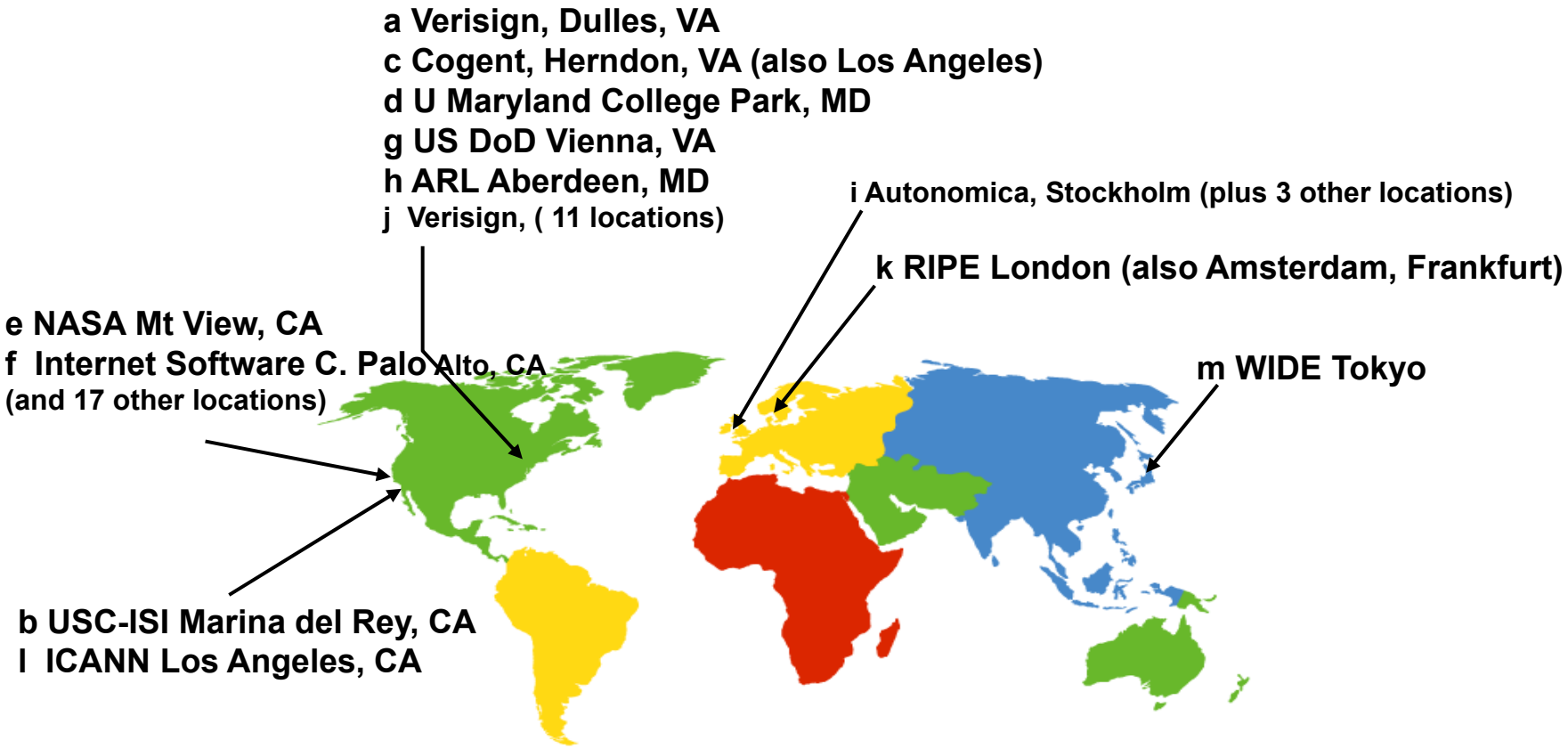


Name Server

- Each name server knows the addresses of the root servers
- Each name server knows the addresses of its immediate children (i.e., those it delegates)

Root Servers

- The root zone is managed by the root name servers
  - 13 root name servers worldwide



Root Servers

Operated by 12 different organizations.

- 1) Verisign
- 2) University of Southern California
- 3) Cogent
- 4) University of Maryland
- 5) NASA AMES Research Center
- 6) Internet Systems Consortium
- 7) US Department of Defense
- 8) US Army Research Lab
- 9) Netnod
- 10) RIPE
- 11) ICANN
- 12) WIDE

TLD Servers

- **Responsible for:**
  - **top-level domains such as com, org, net, edu, etc,**

  AND
  - **all top-level country domains such as uk, fr, ca, cn, etc.**
- **Stores the IP address of the second-level domain (e.g. microsoft.com) within the TLD Name (.com).**

Authoritative Servers

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

Using DNS

- Two components
  - Local DNS servers
  - Resolver software on hosts

Using DNS

- **Local DNS Server**
  - Usually near the end hosts that use it.
  - Does not strictly belong to hierarchy.
  - Each ISP (residential ISP, company, university) has one.
    - Also called "default name server"
  - When a host makes a DNS query, query is first sent to its local DNS server
    - Acts as a proxy, forwards query into hierarchy.

Using DNS

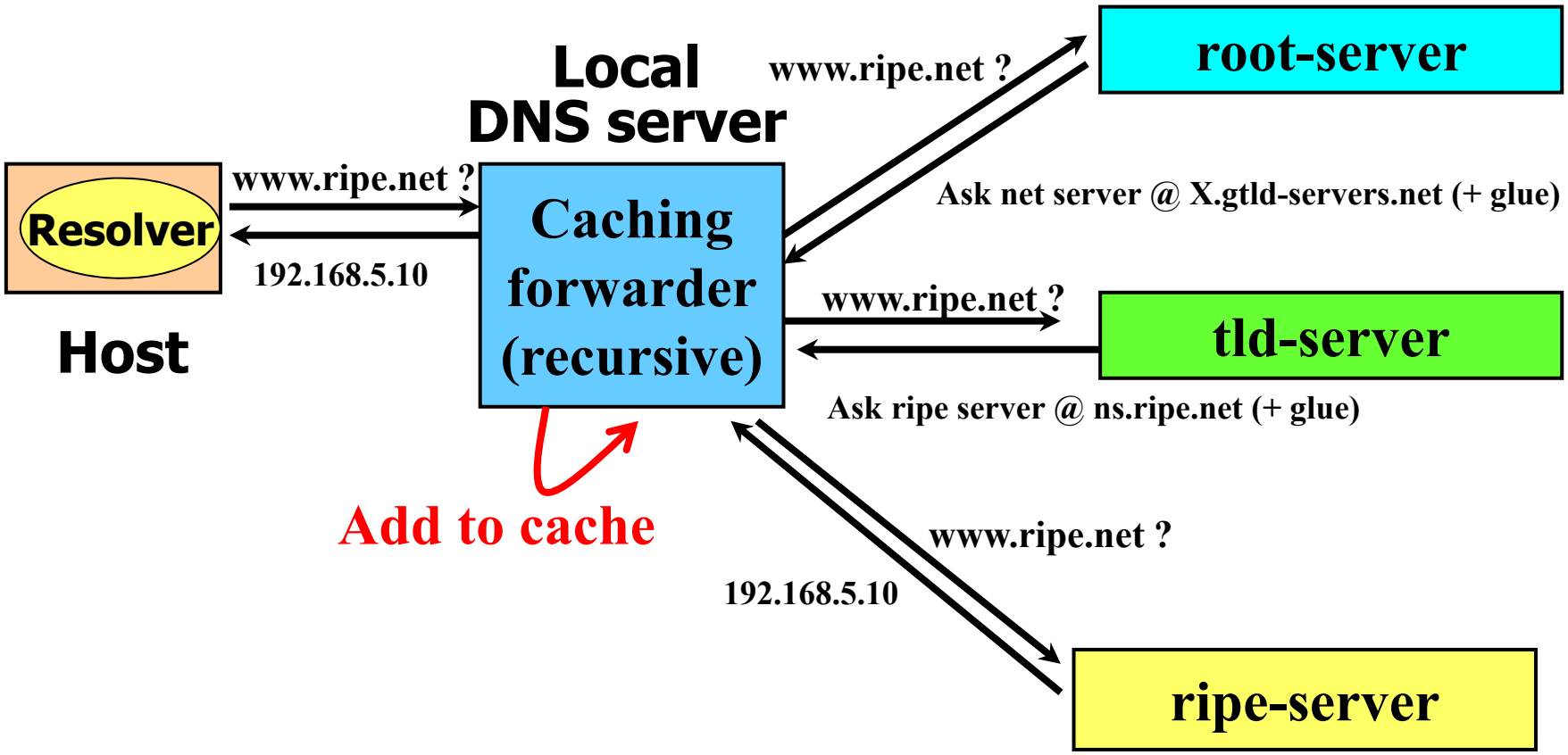
- **Resolvers**
  - Ask the questions to the DNS system on behalf of the application.
  - Normally implemented in a system library (e.g, libc)

```
gethostbyname(char *name);
gethostbyaddr(char *addr, int len, type);
```

will trigger the resolver.

Typical Resolution

Ask question: www.ripe.net



Lookup Methods

Two Types of Queries

Recursive query:

- Server goes out and searches for more info (recursive)
- Only returns final answer or "not found"

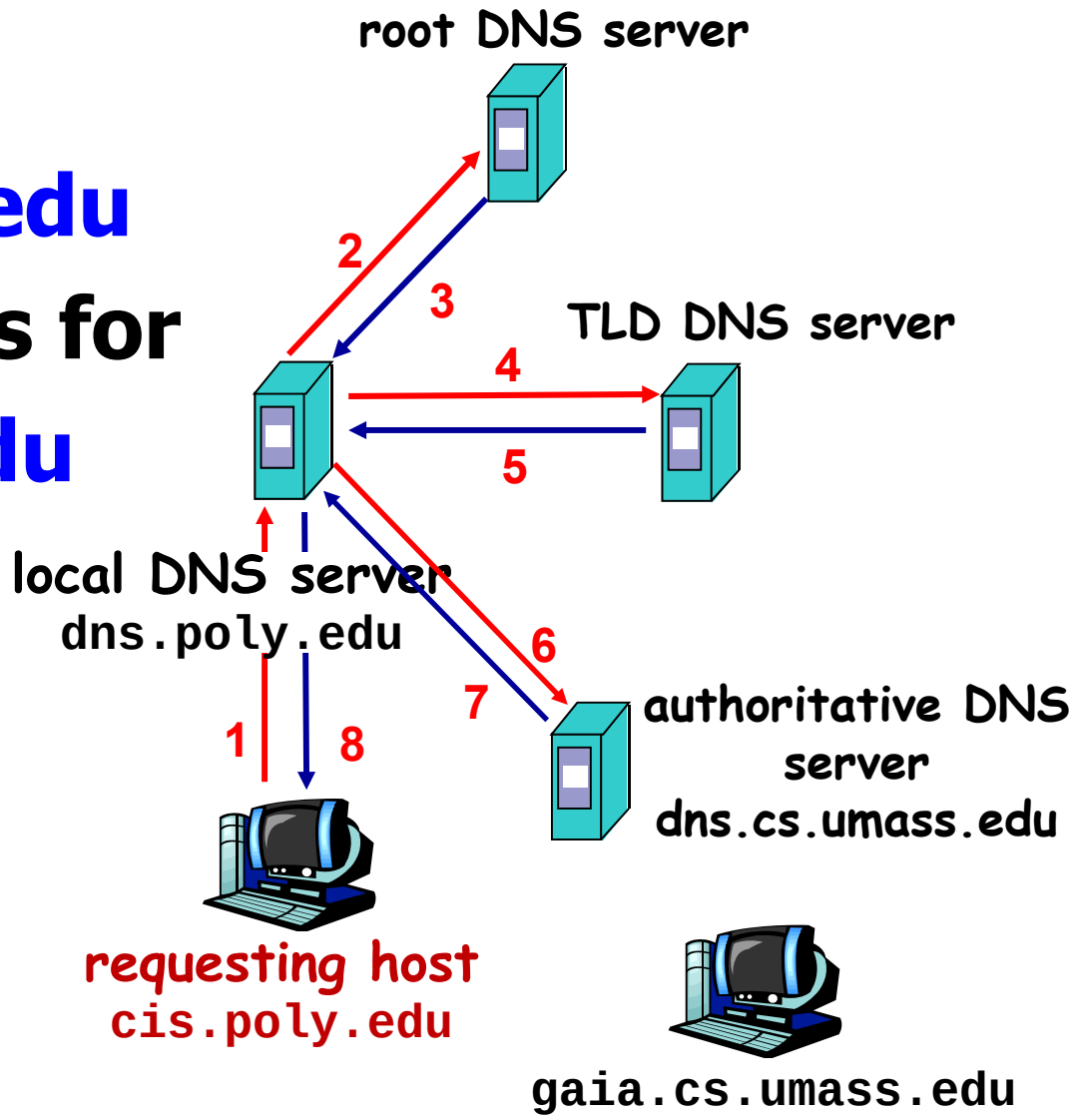
Iterative query:

- Server responds with as much as it knows (iterative)
- "I don't know this name, but ask this server"

Example

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

iterated query

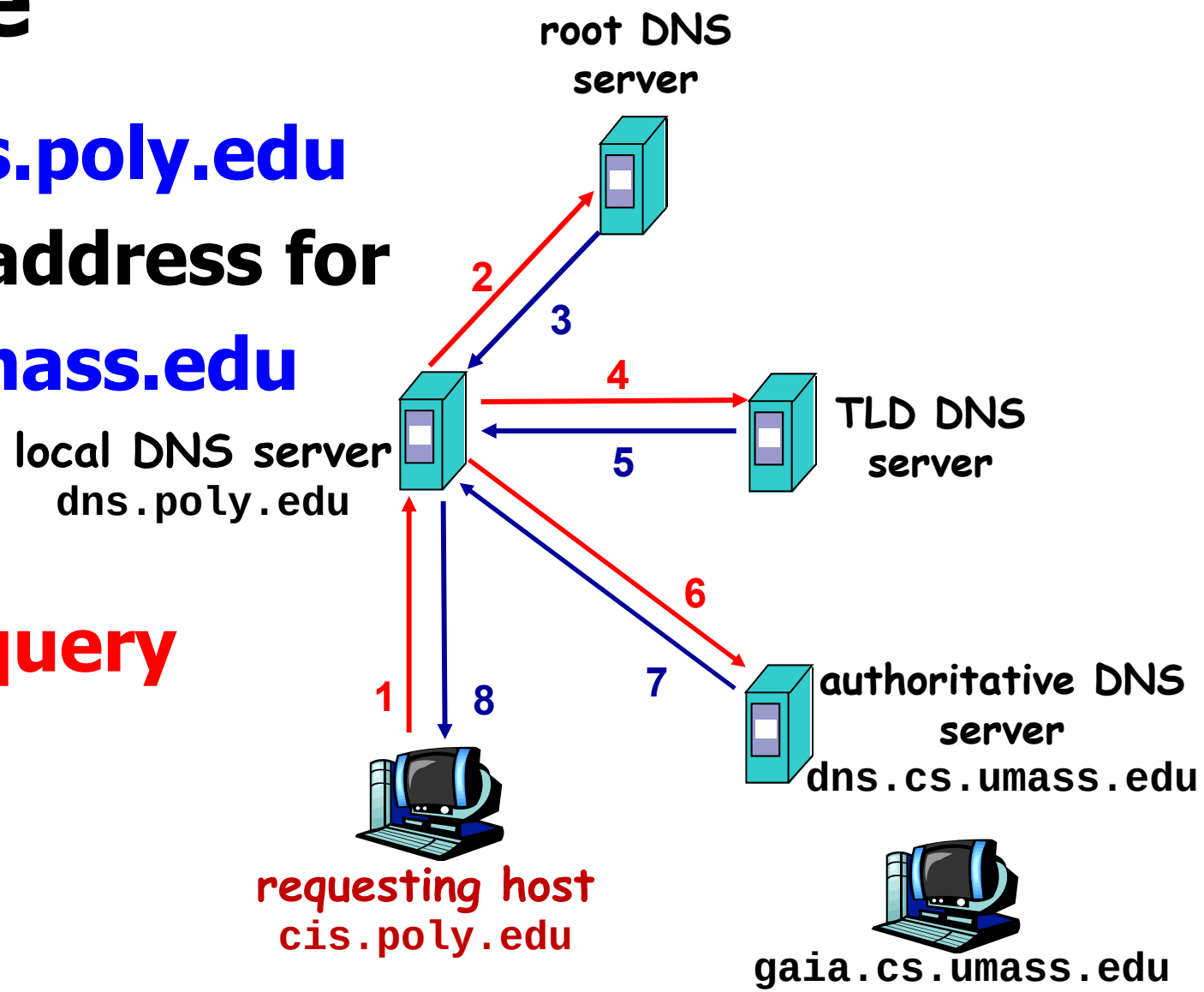


Example

Host at cis.poly.edu

wants IP address for

gaia.cs.umass.edu



iterated query

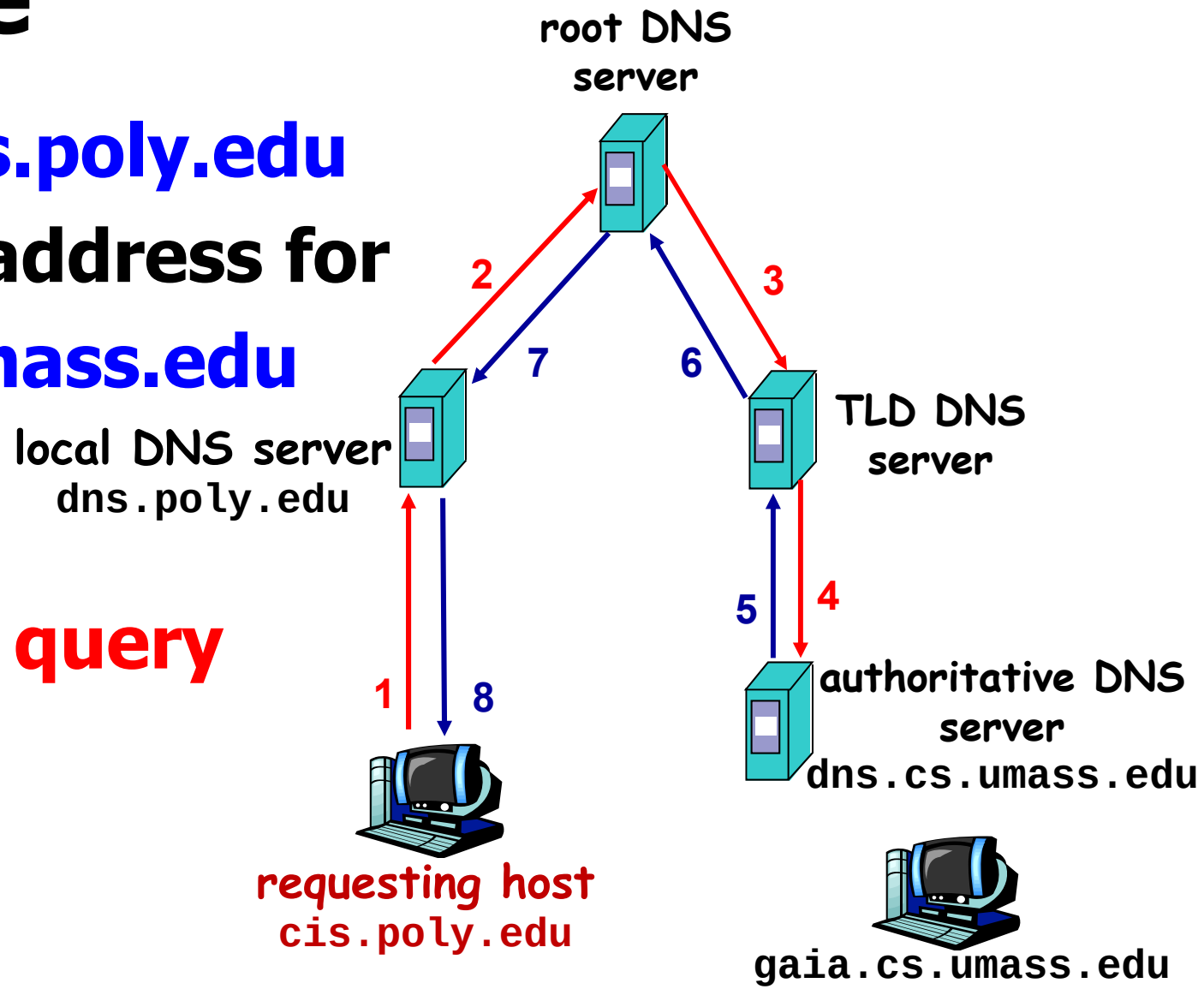
Example

Host at cis.poly.edu

wants IP address for

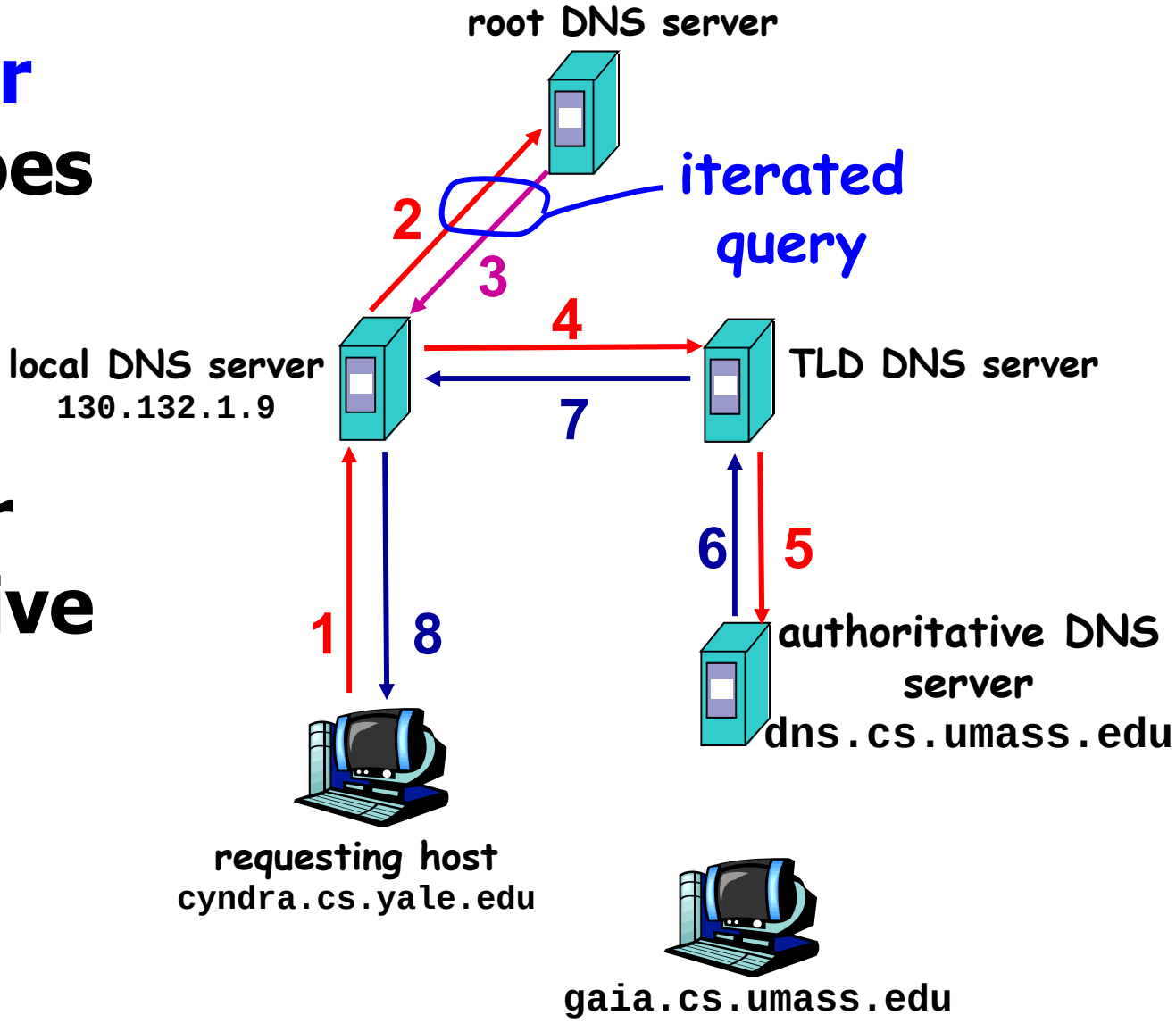
gaia.cs.umass.edu

recursive query



Example: The Hybrid Case

- **Local server** typically does recursive
- **Root** server does iterative



DNS: Caching and Updating records

- **Performing all these queries take time**
  - And all this before the actual communication takes place.
- **Caching can substantially reduce overhead**
  - The top-level servers very rarely change
  - Popular sites (e.g., www.cnn.com) visited often
  - Local DNS server often has the information cached
- **How DNS caching works**
  - DNS servers cache responses to queries
  - Responses include a "time to live" (TTL) field
  - Server deletes the cached entry after TTL expires

DNS: Caching and Updating records

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

域名解析过程

- ①客户机提出域名解析请求，并将该请求发送给本地域名服务器
- ②本地域名服务器收到请求后，先查询本地的缓存，如果有该纪录项，则本地域名服务器就直接把查询的结果返回
- ③如果本地的缓存中没有该纪录，则本地域名服务器就直接把请求发给根域名服务器，根域名服务器再返回给本地域名服务器一个所查询域(根的子域)的顶级域名服务器的地址

域名解析过程

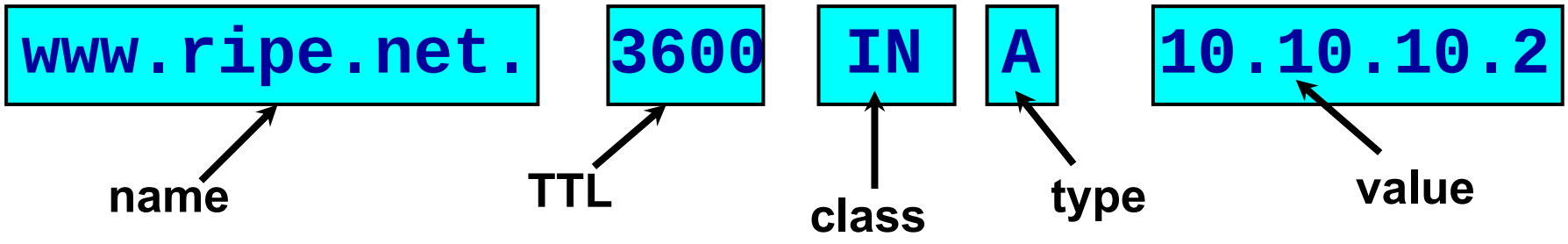
- ④本地服务器再向上一步返回的顶级域名服务器发送请求。接受请求的服务器查询自己的缓存，如果没有该纪录，则返回相关的下级(权威)域名服务器的地址
- ⑤重复第四步，直到找到正确的纪录
- ⑥本地域名服务器把返回的结果保存到缓存，以备下一次使用，同时还将结果返回给客户机

DNS Resource Records

- The DNS data is stored in the database in the form of resource records (RR).
- The RRs are directly inserted in the DNS messages.
- The RRs are a 5 tuple that consist of:

{name, TTL, class, type, value}

Class = Internet (IN), Chaosnet (CH), etc.

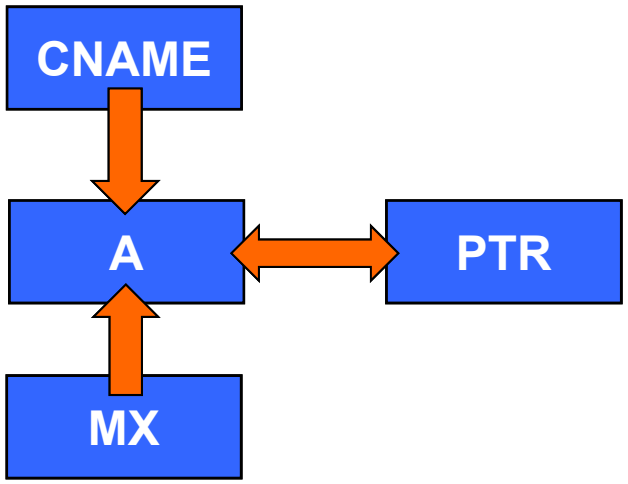


DNS Resource Records

- There are a few dozen different RRs; the most common are:
  - SOA
  - NS
  - A
  - CNAME
  - MX
  - PTR

The SOA and NS records are used to provide information about the DNS itself

Resource Records can have relationships with each other



DNS Resource Records

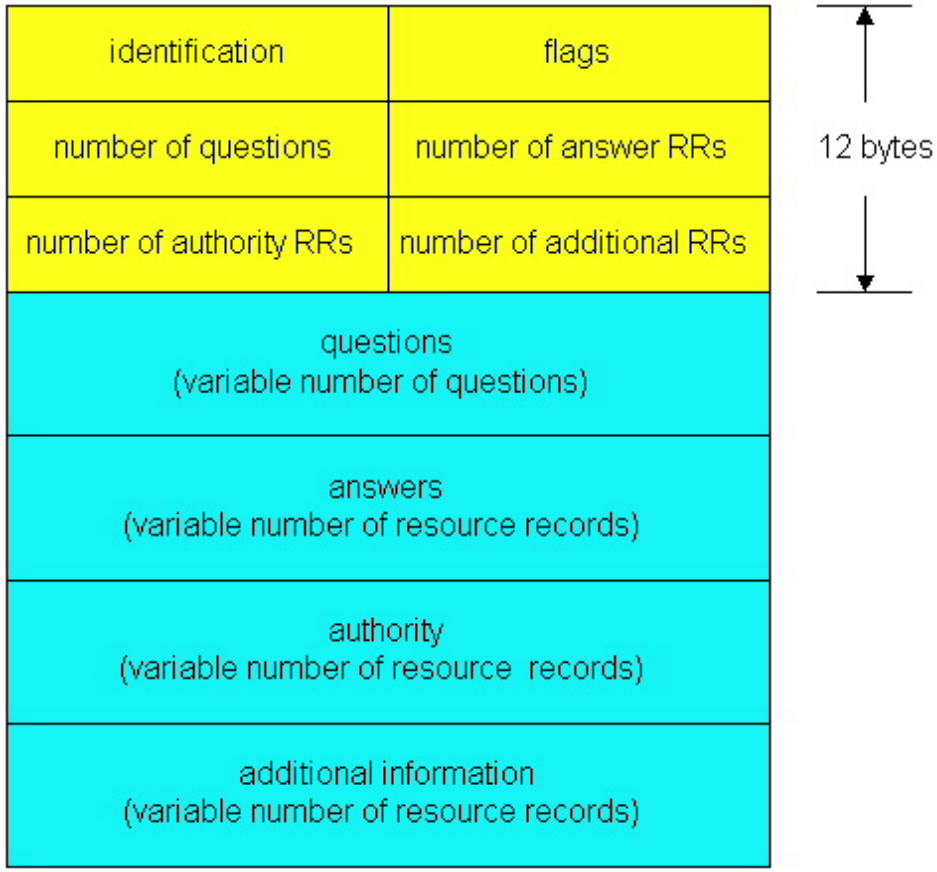
Type	Description	Function
SOA	Start of [a zone of] Authority record	Specifies authoritative information about a DNS zone. Appears at the beginning of a DNS zone file
NS	Name Server record	Delegates a DNS zone to use the given authoritative name servers
A	Address record	Most commonly used to map hostnames to an IP address of the host
CNAME	Canonical Name record	Alias of one name to another
MX	mail exchange record	Maps a domain name to a list of mail servers for that domain
PTR	pointer record.	The most common use is for implementing reverse DNS lookups

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**



DNS and Security

- **No way to verify answers**
  - Opens up DNS to many potential attacks
  - DNSSEC fixes this
- **Most obvious vulnerability: recursive resolution**
  - Using recursive resolution, host must trust DNS server
  - When at Starbucks, server is under their control
  - And can return whatever values it wants
- **More subtle attack: Cache poisoning**
  - Those "additional" records can be anything!

DNS Summary

- DNS provides a mechanism for maintaining the user friendliness of the Internet
- Motivations → large distributed database
  - Scalability
  - Independent update
  - Robustness
- Hierarchical database structure
  - Zones
  - Name servers. How is a lookup done?
- Caching and TTLs
- What are the steps to resolve a domain name?

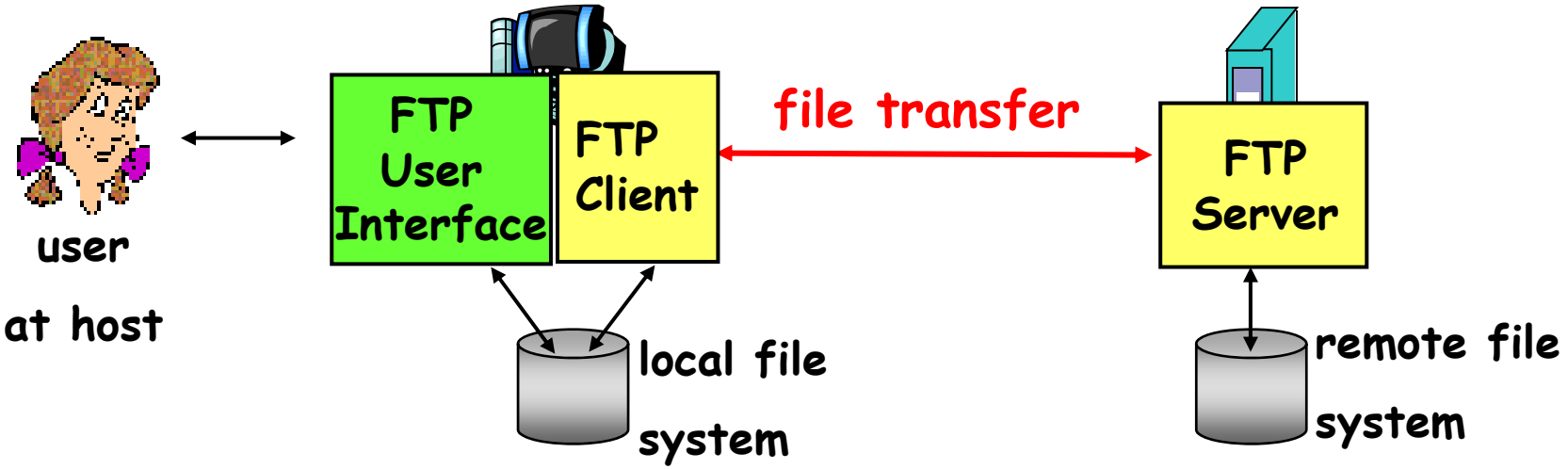
What DNS did Right?

- **Hierarchical delegation** avoids central control, improving manageability and scalability
- **Redundant servers** improve robustness
- **Caching** reduces workload and improve robustness

Chapter 7: Roadmap

- Application and Application Layer
- C/S and P2P Application Architectures
- DNS
- FTP
- E-Mail and SMTP/POP/IMAP
- WWW and HTTP
- DHCP

FTP: File Transfer Protocol



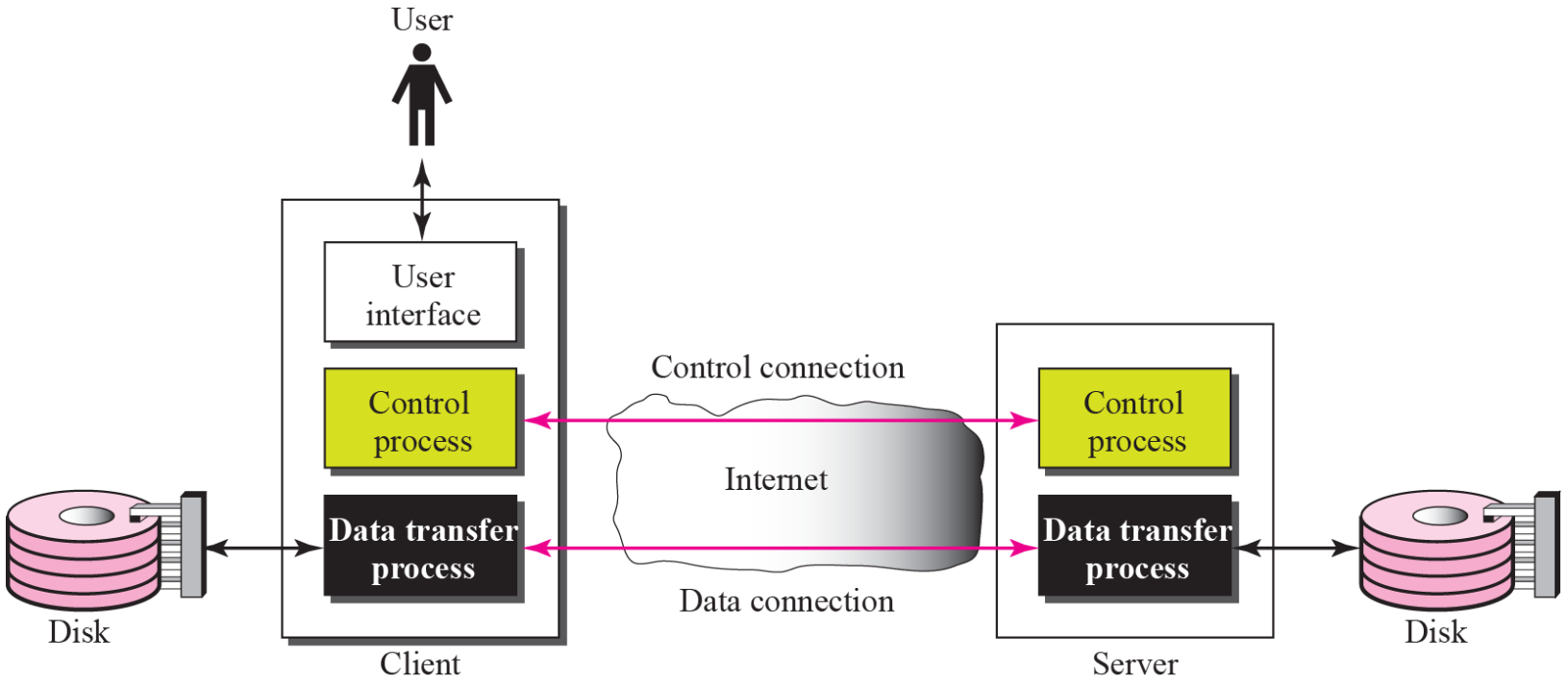
- transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- RFC 959

FTP: File Transfer Protocol

- Can you transfer files from one system to another straightforward?
- What problems should be dealt with at first?
- Some problems:
  - Different file name conventions.
  - Different ways to represent text and data.
  - Different directory structures.

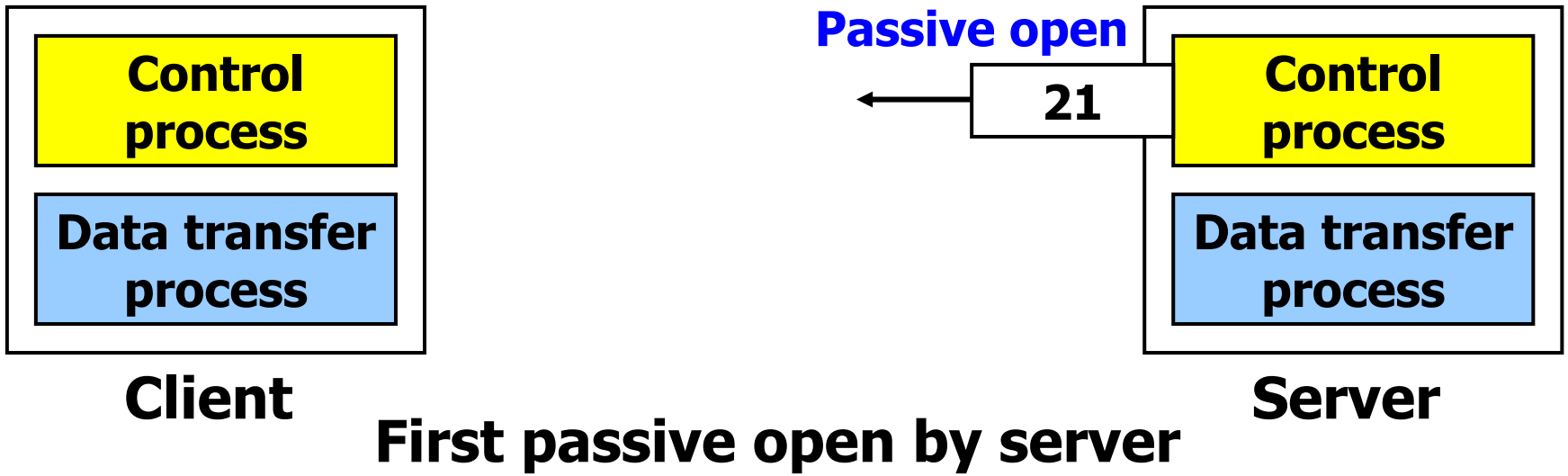
All of these problems have been solved by FTP in a very simple and elegant approach.

Separate control, data connections

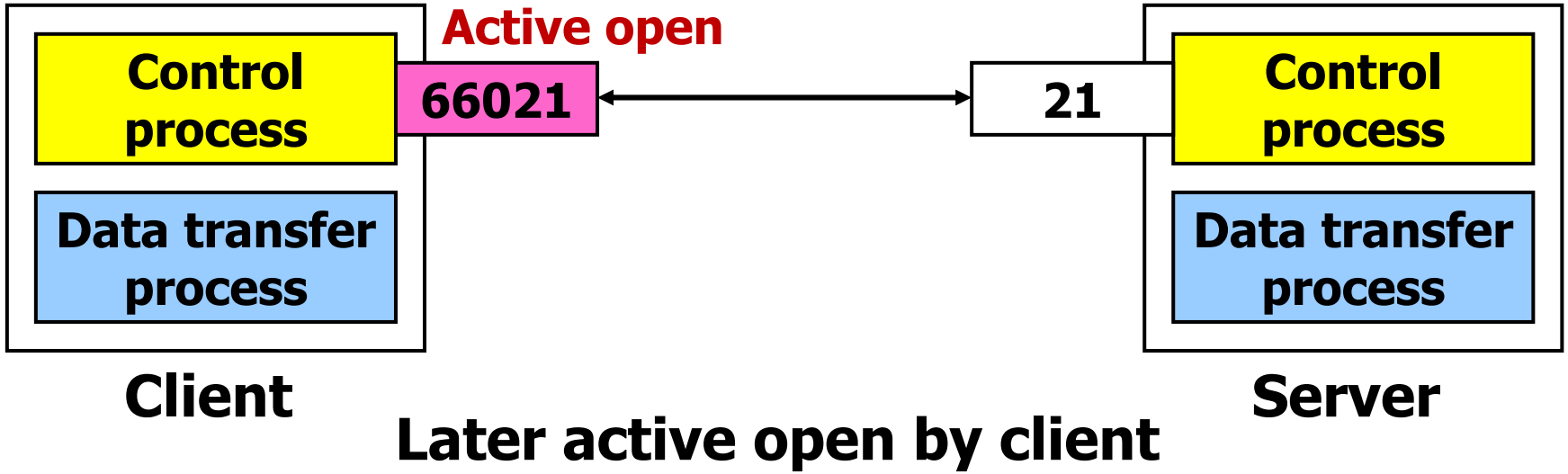


- FTP uses the services of TCP. It needs two TCP connections.
- The well-known port 21 is used for the control connection.
- The well-known port 20 for the data connection.

The control connection



First passive open by server

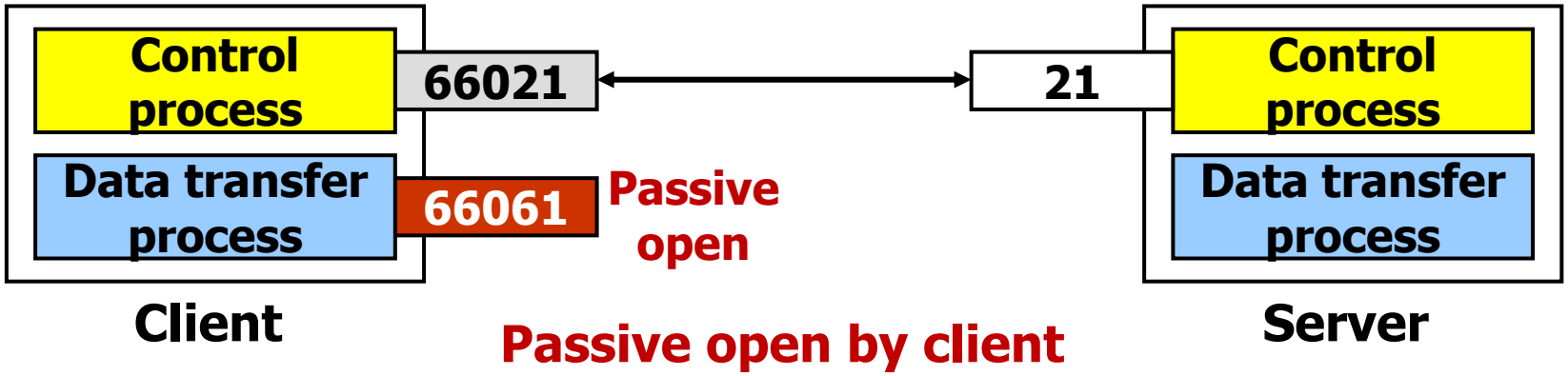


Later active open by client

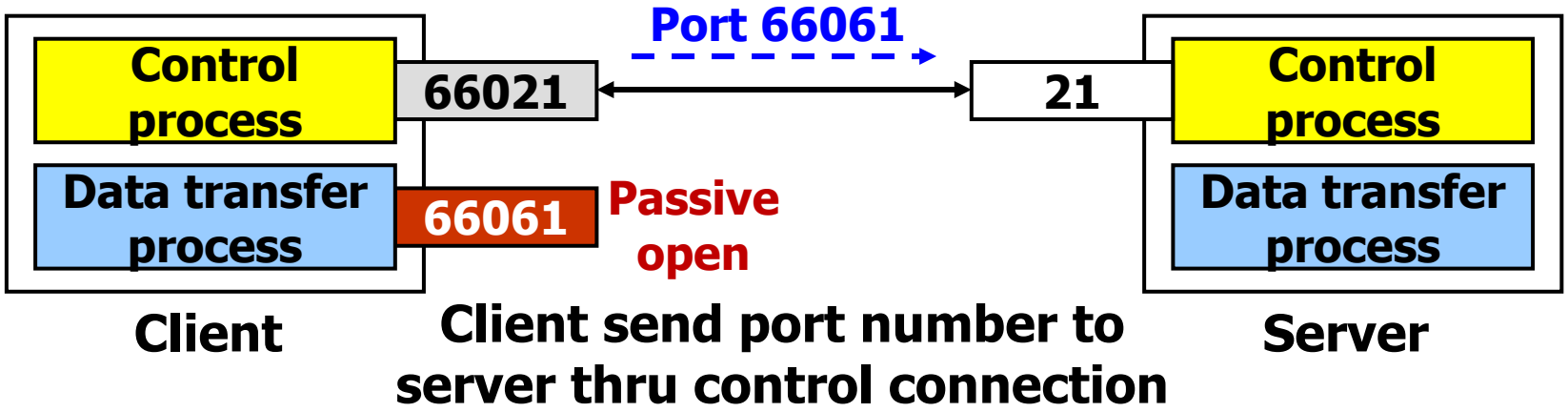
The Data Connection

- Uses Server's well-known port 20

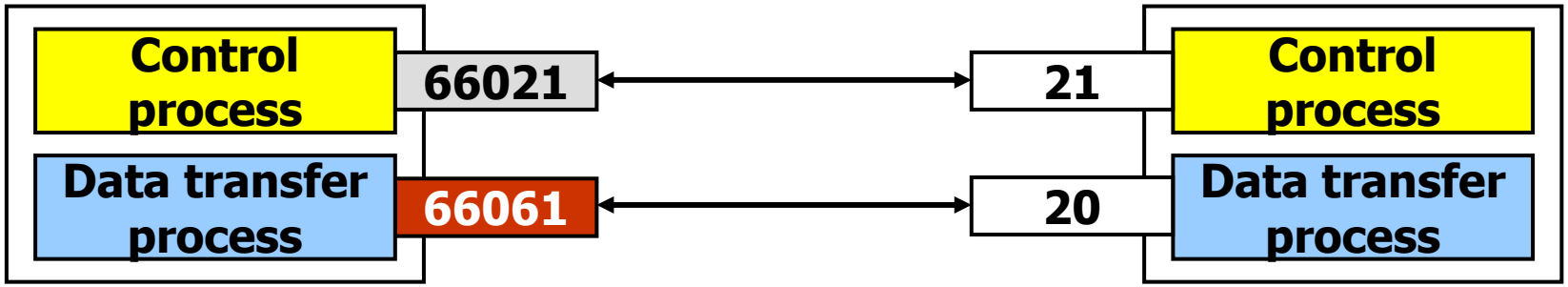
1. Client issues a passive open on an ephemeral port, say x.
2. Client uses PORT command to tell the server about the port number x.
3. Server issues an active open from port 20 to port x.
4. Server creates a child server/ephemeral port number to serve the client



Passive open by client

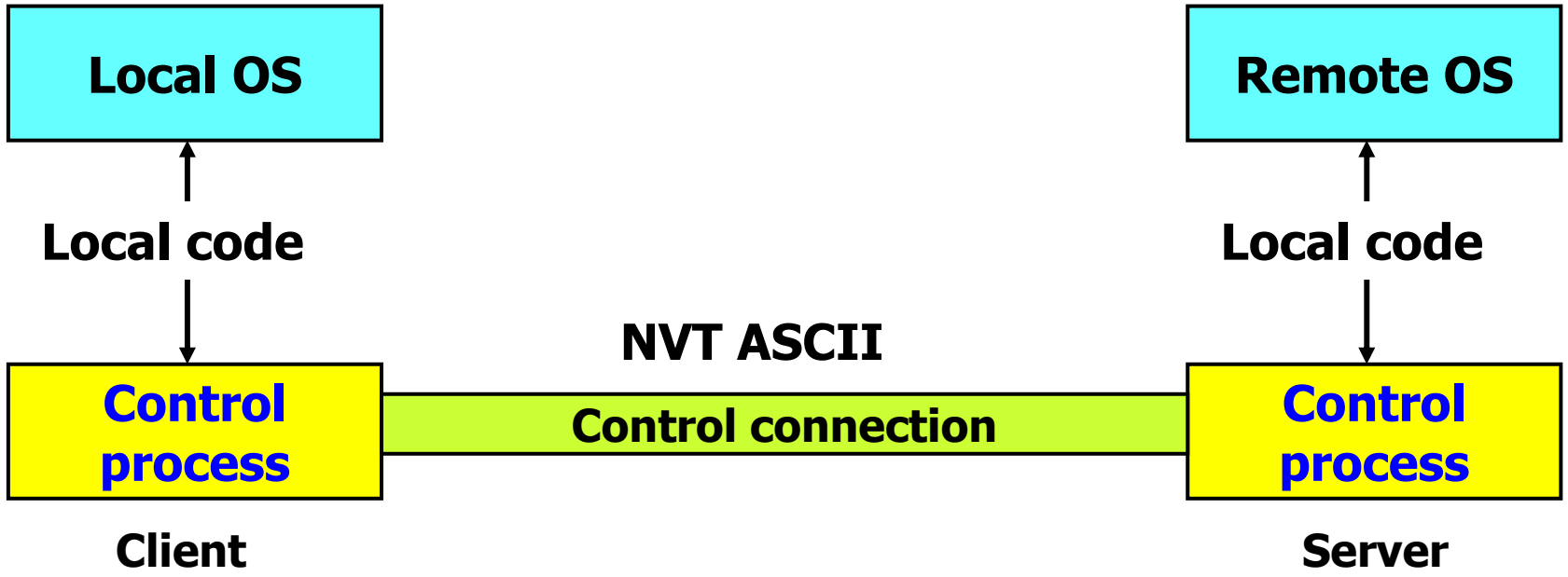


Client send port number to server thru control connection



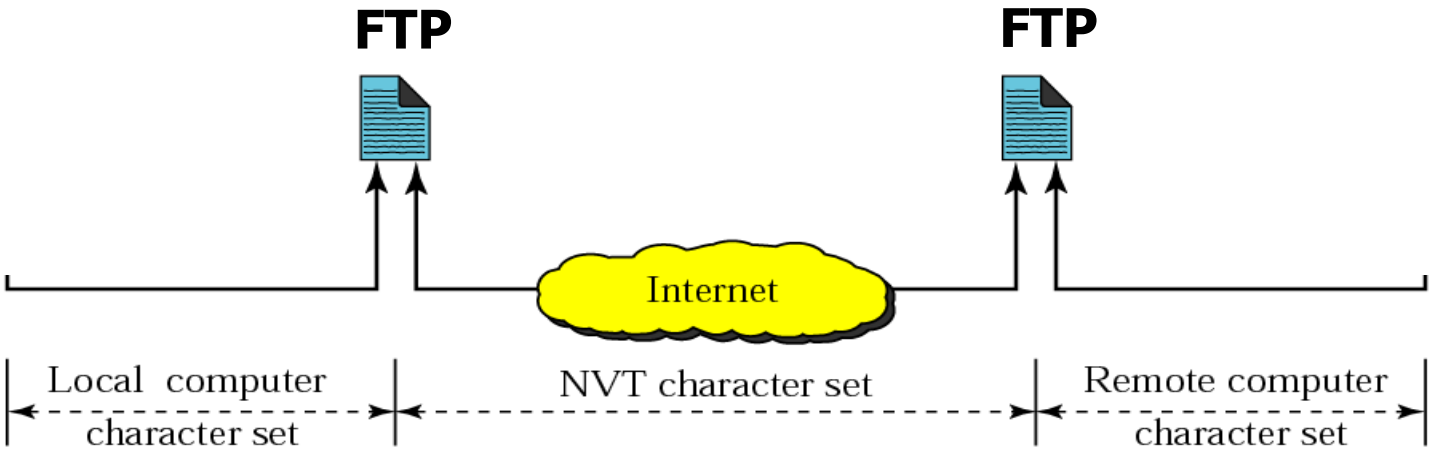
Active open by server

Using the control connection

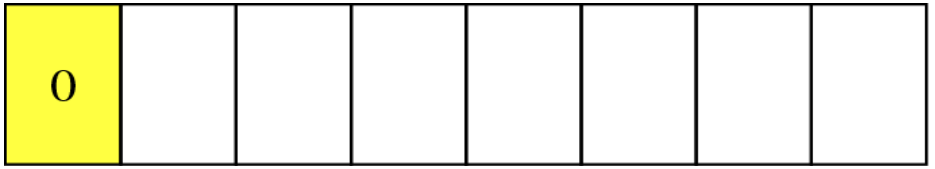


NVT ASCII: Network Virtual Terminal ASCII

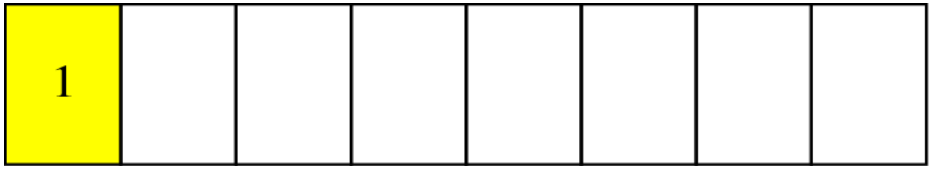
NVT ASCII: Network Virtual Terminal ASCII



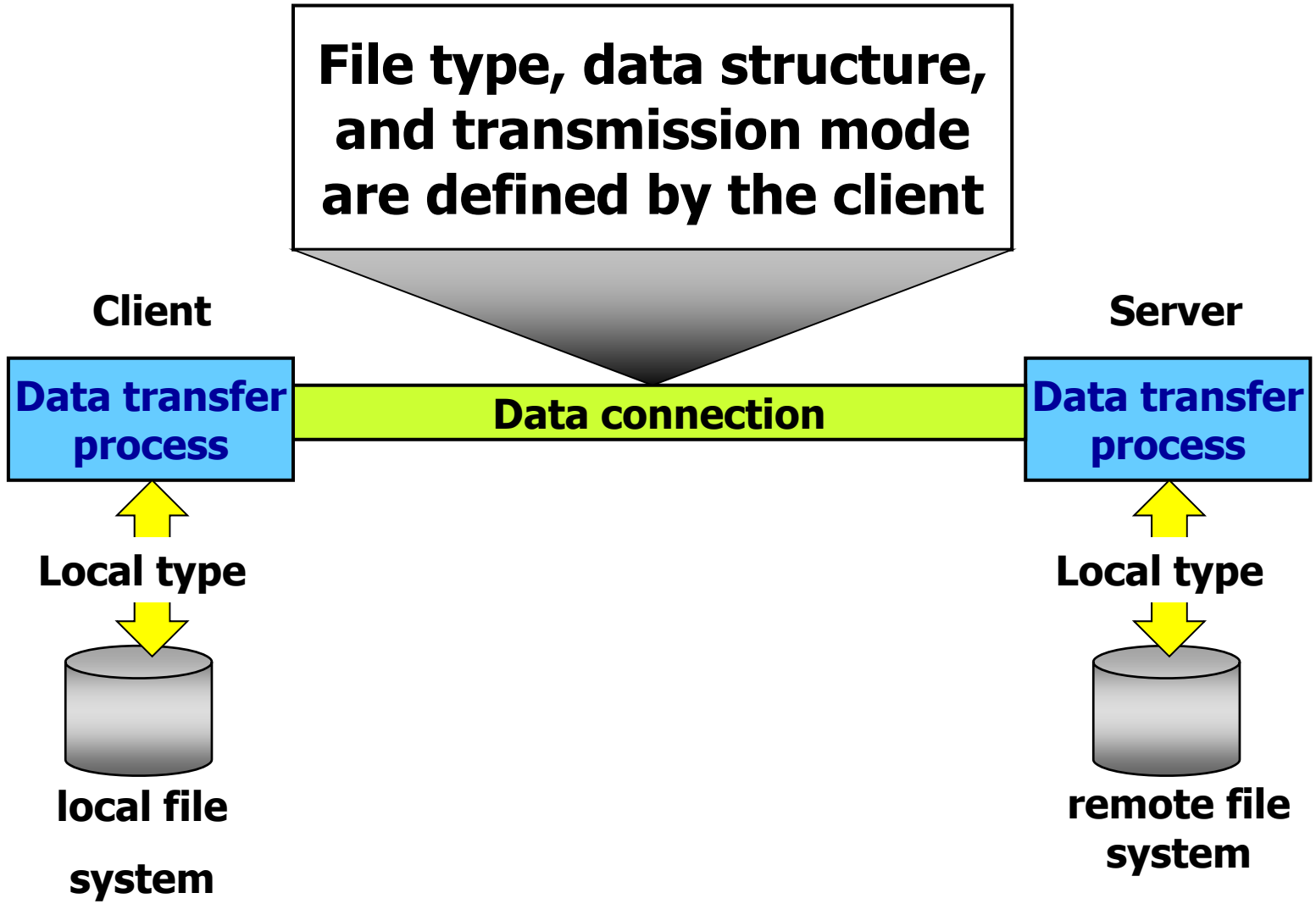
Format of NVT ASCII characters



Format of NVT control characters



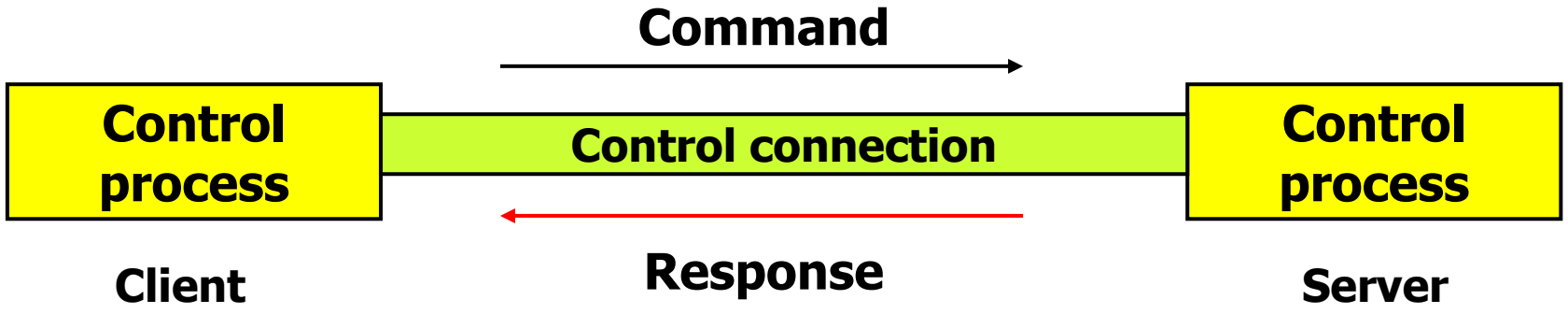
Using the data connection



Using the data connection

- **File Type**
  - ASCII
  - EBCDIC
  - Image
- **Data Structure**
  - File Structure
  - Record Structure
  - Page Structure
- **Transmission Mode**
  - Stream mode
  - Block mode
  - Compressed mode

FTP Command processing



- Access Commands
- File Management
- Data Formatting
- Port defining
- File transfer
- Miscellaneous

FTP Command processing

Access commands

Command	*Argument(s)*	*Description*
USER	User id	User information
PASS	User password	Password
ACCT	Account to be charged	Account information
REIN		Reinitialize
QUIT		Log out of the system
ABOR		Abort the previous command

FTP Command processing

File management commands

Command	*Argument(s)*	*Description*
CWD	Directory name	Change to another directory
CDUP		Change to parent directory
DELE	File name	Delete a file
LIST	Directory name	List subdirectories or files
NLIST	Directory name	List subdirectories or files without attributes
MKD	Directory name	Create a new directory
PWD		Display name of current directory
RMD	Directory name	Delete a directory
RNFR	File name (old)	Identify a file to be renamed
RNTO	File name (new)	Rename the file
SMNT	File system name	Mount a file system

FTP Command processing

Data formatting commands

Command	*Argument(s)*	*Description*
TYPE	**A** (ASCII), **E** (EBCDIC), **I** (Image), **N** (Nonprint), or **T** (TELNET)	Define file type
STRU	**F** (File), **R** (Record), or **P** (Page)	Define organization of data
MODE	**S** (Stream), **B** (Block), or **C** (Compressed)	Define transmission mode

FTP Command processing

Port defining commands

Command	*Argument(s)*	*Description*
PORT	6-digit identifier	Client chooses a port
PASV		Server chooses a port

FTP Command processing

File transfer commands

Command	*Argument(s)*	*Description*
RETR	File name(s)	Retrieve files; file(s) are transferred from server to client
STOR	File name(s)	Store files; file(s) are transferred from client to server
APPE	File name(s)	Similar to STOR, but if file exists, data must be appended to it
STOU	File name(s)	Same as STOR, but file name will be unique in the directory
ALLO	File name(s)	Allocate storage space for files at the server
REST	File name(s)	Position file marker at a specified data point
STAT	File name(s)	Return status of files

FTP Command processing

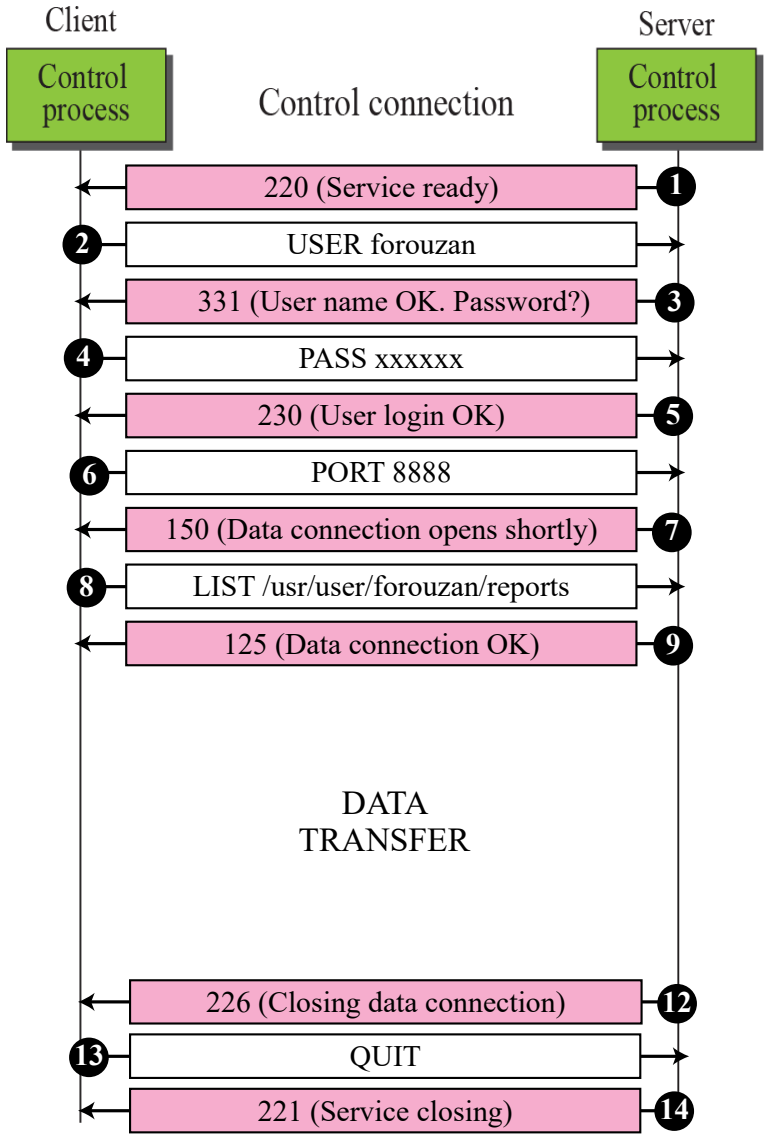
Miscellaneous commands

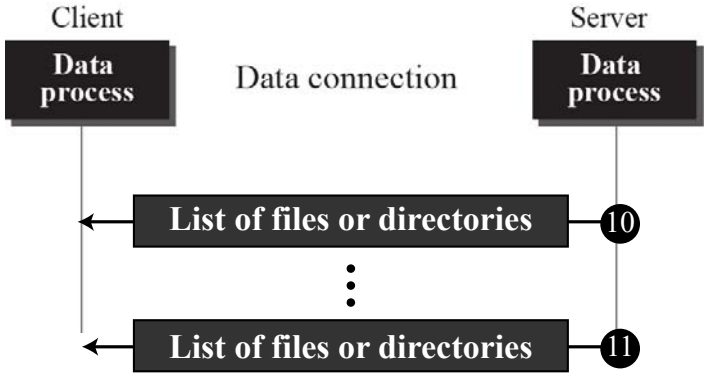
Command	*Argument(s)*	*Description*
HELP		Ask information about the server
NOOP		Check if server is alive
SITE	Commands	Specify the site-specific commands
SYST		Ask about operating system used by the server

FTP Command processing

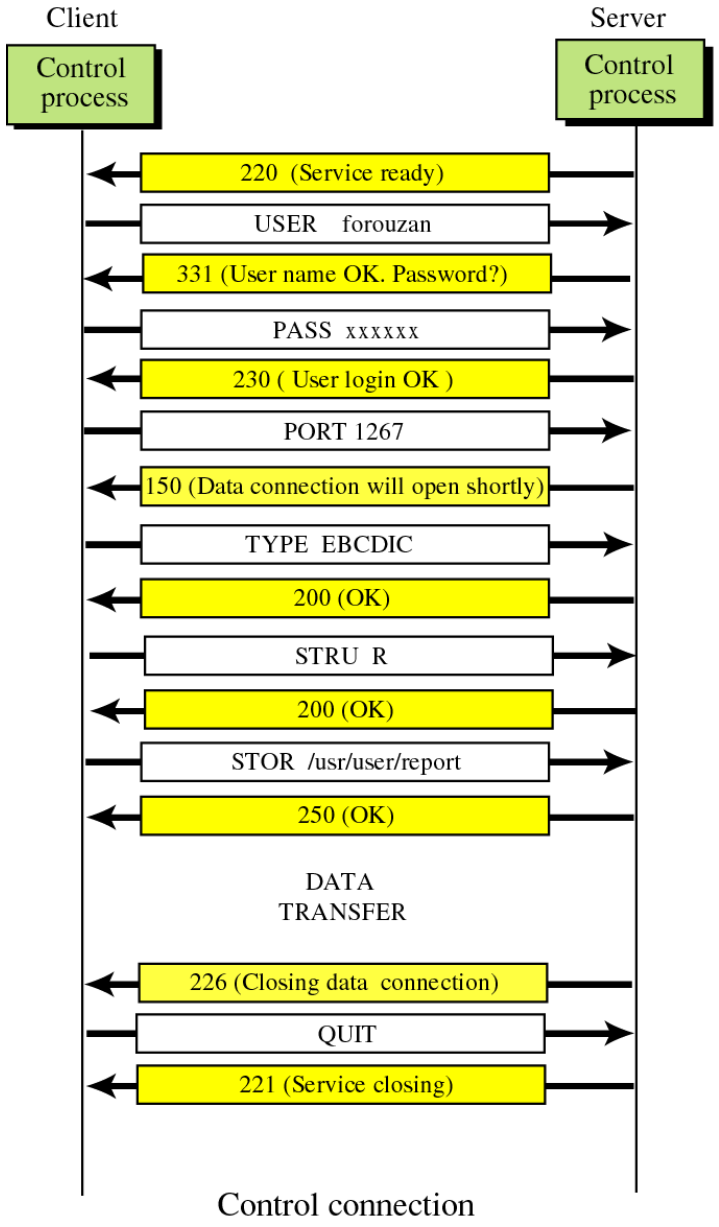
Code	*Description*
Positive Preliminary Reply	
120	Service will be ready shortly
125	Data connection open; data transfer will start shortly
150	File status is OK; data connection will be open shortly
Positive Completion Reply	
200	Command OK
211	System status or help reply
212	Directory status
213	File status
214	Help message
215	Naming the system type (operating system)
220	Service ready
221	Service closing
225	Data connection open
226	Closing data connection
227	Entering passive mode; server sends its IP address and port number
230	User login OK
250	Request file action OK
Positive Intermediate Reply	
331	User name OK; password is needed
332	Need account for logging
350	The file action is pending; more information needed

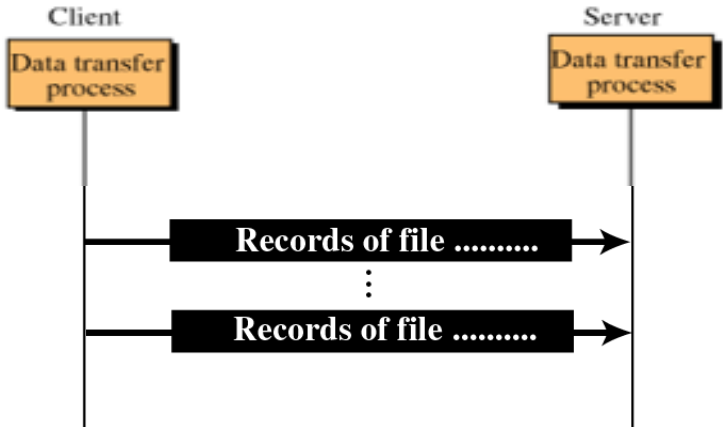
Example 1


Client
Server
Control process
Control connection
Control process
220 (Service ready)
1
2
USER forouzan
331 (User name OK. Password?)
3
4
PASS xxxxxx
230 (User login OK)
5
6
PORT 8888
150 (Data connection opens shortly)
7
8
LIST /usr/user/forouzan/reports
125 (Data connection OK)
9
DATA TRANSFER
226 (Closing data connection)
12
13
QUIT
221 (Service closing)
14


Client
Server
Data process
Data connection
Data process
List of files or directories
10
List of files or directories
11

Example 2


Client
Server
Control process
Control process
220 (Service ready)
USER forouzan
331 (User name OK. Password?)
PASS xxxxxx
230 (User login OK)
PORT 1267
150 (Data connection will open shortly)
TYPE EBCDIC
200 (OK)
STRU R
200 (OK)
STOR /usr/user/report
250 (OK)
DATA TRANSFER
226 (Closing data connection)
QUIT
221 (Service closing)
Control connection


Client
Server
Data transfer process
Data transfer process
Records of file .....
Records of file .....

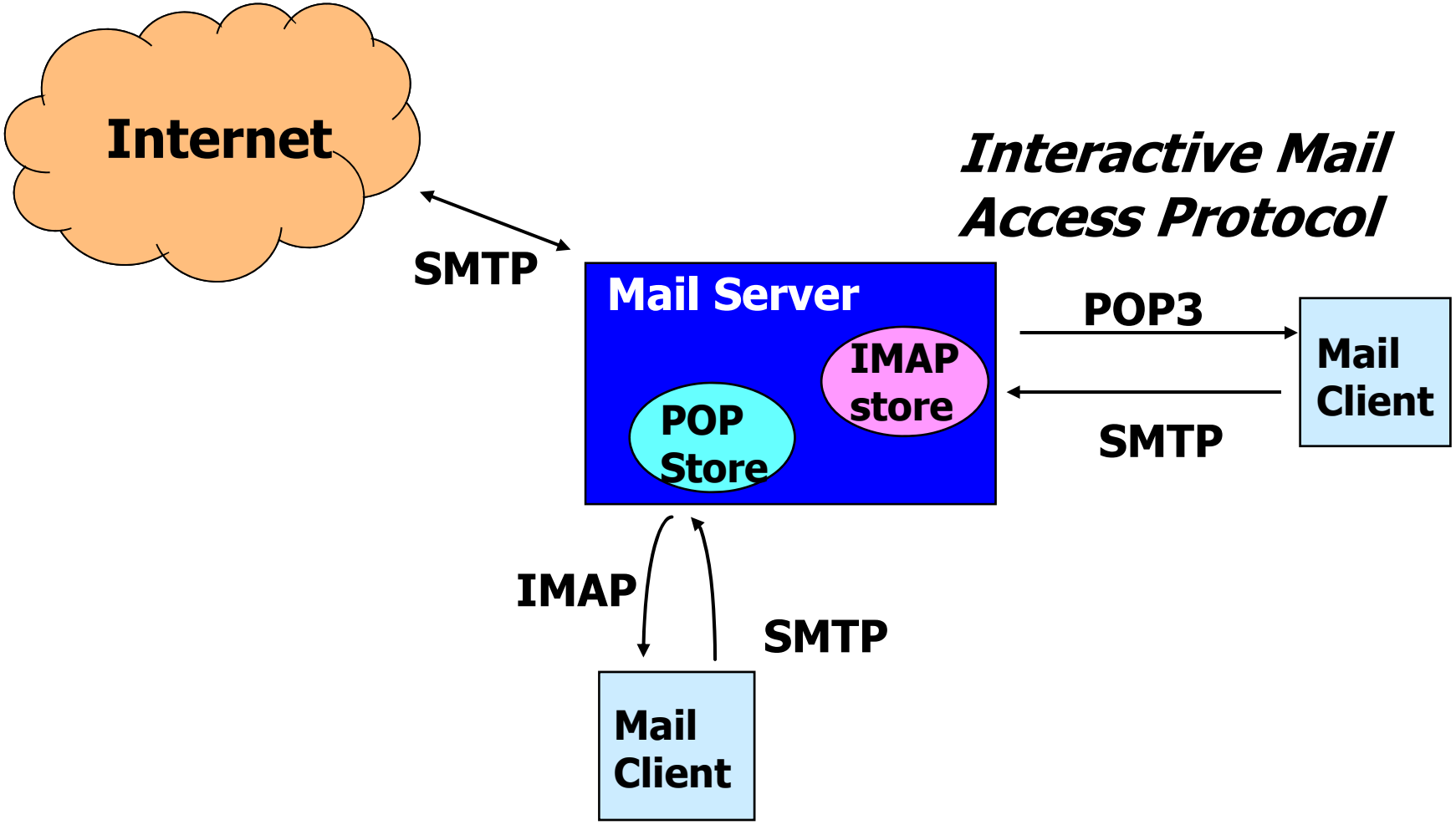
匿名FTP

- 某些场所提供一组公共文件
- 此时可以使用匿名FTP登录
  - 用户名：annonymous
  - 口令：guest或邮件地址
- 限制：只允许用户使用命令的一个子集。例如：允许用户复制文件，但不能查找文件

Chapter 7: Roadmap

- Application and Application Layer
- C/S and P2P Application Architectures
- DNS
- FTP
- E-Mail and SMTP/POP/IMAP
- WWW and HTTP
- DHCP

Electronic Mail

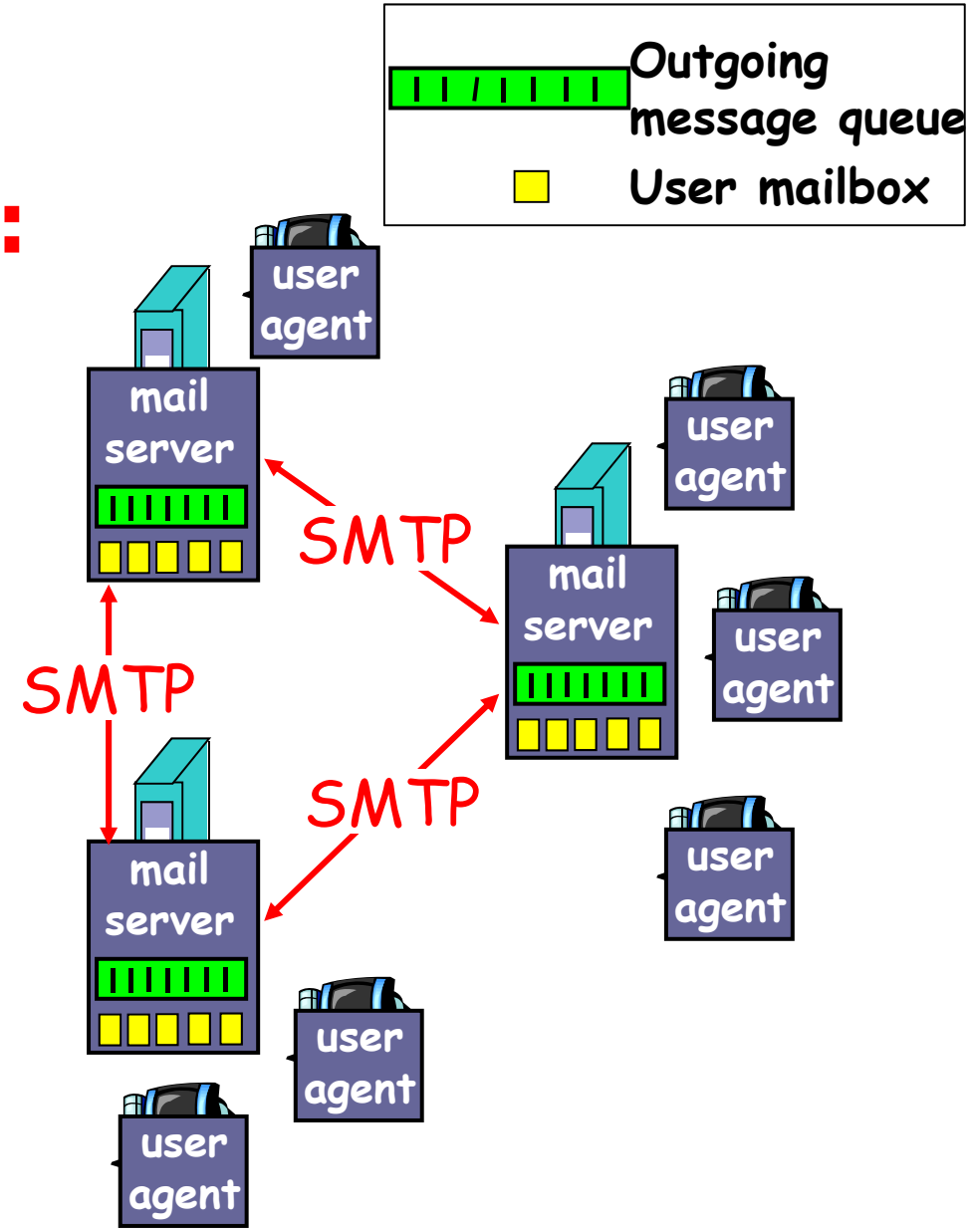


A Typical Mail Environment

Electronic Mail

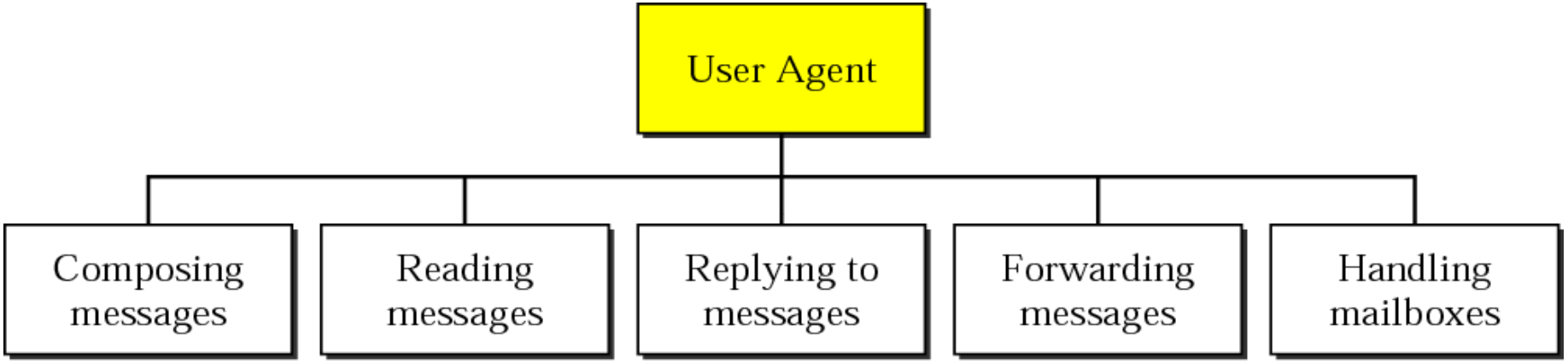
3 major components:

- user agents (**UA**)
- mail transfer agents (**MTA**)
  - e.g mail servers
  - SMTP
- mail access agents (**MAA**)
  - POP3
  - IMAP4



User Agent

- **Provides services to the user to make the process of sending and receiving a message easier.**

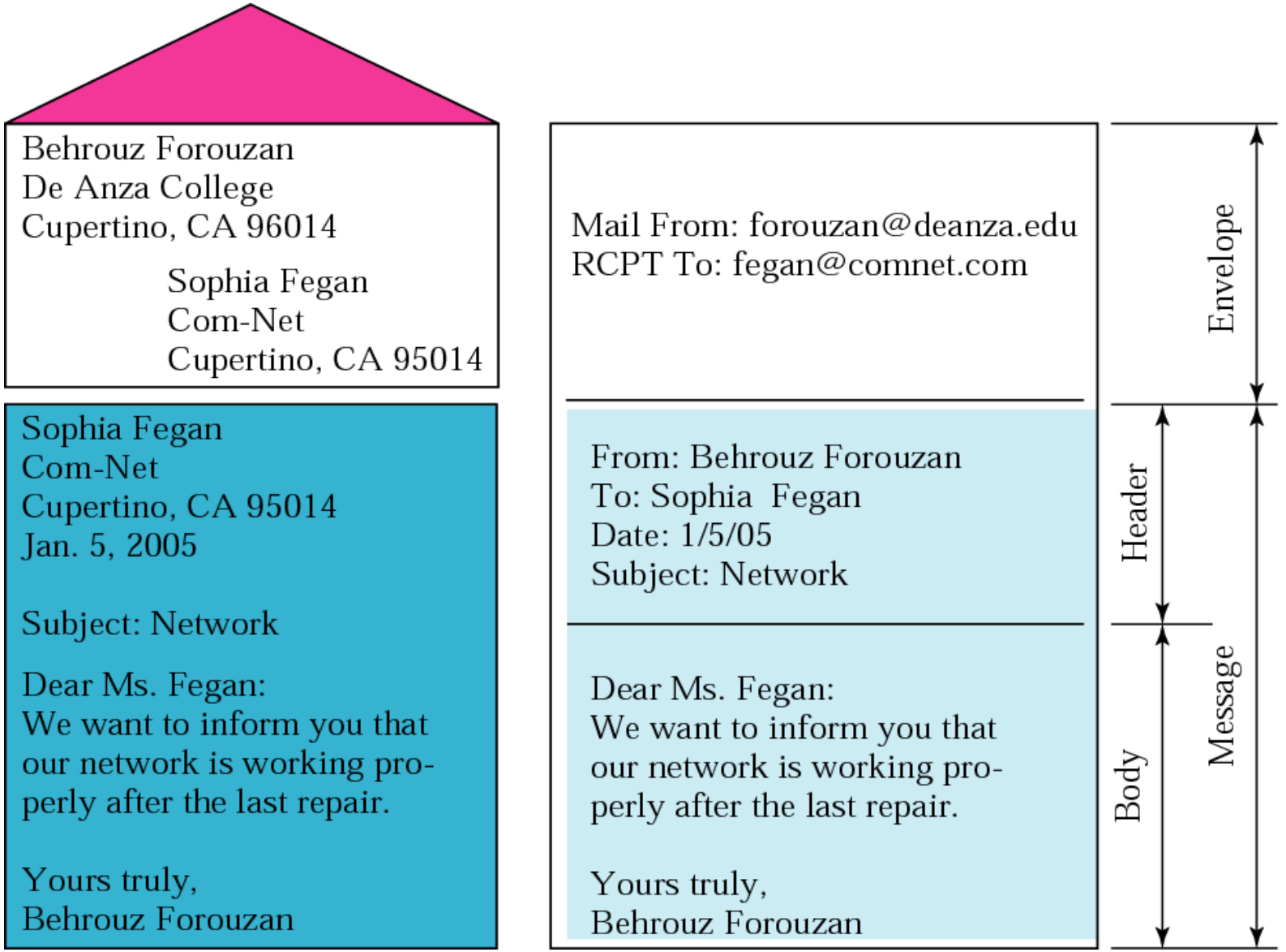


Services Provided by a User Agent

User Agent

- Type
  - command-driven
    - E.g. mail, pine, and elm
  - GUI-based
    - E.g. Outlook, foxmail, Eudora

Format of an email



Email address

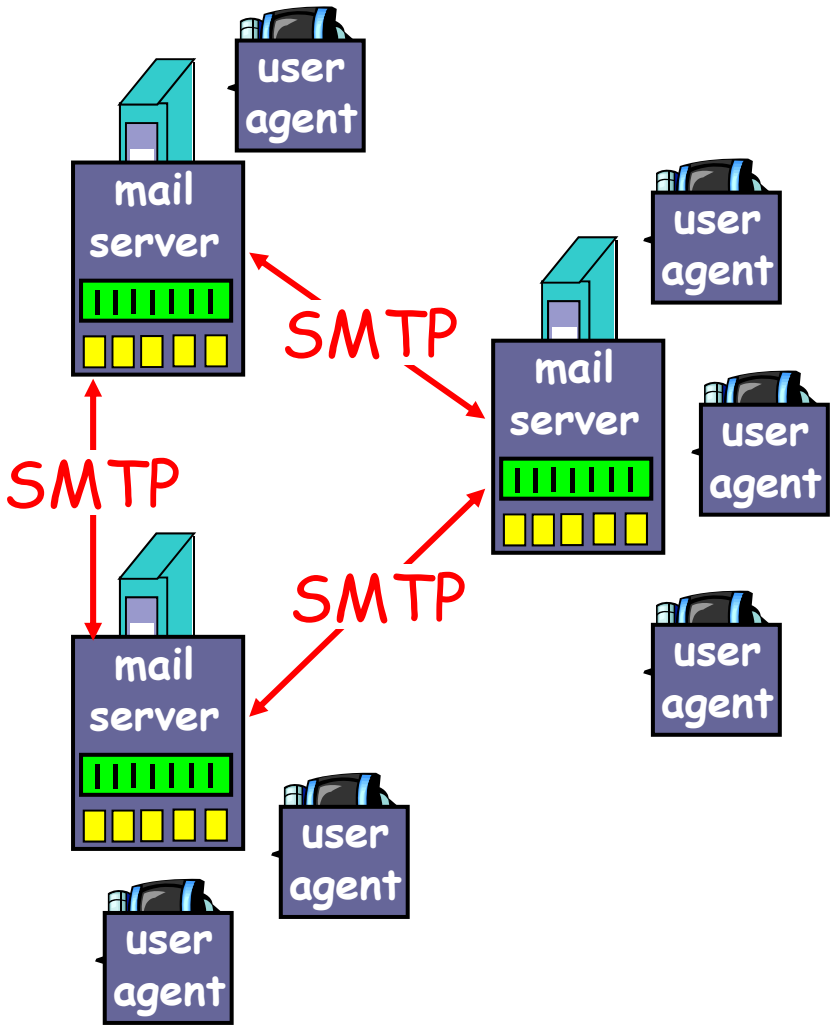
Mail Transfer Agent (MTA)

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

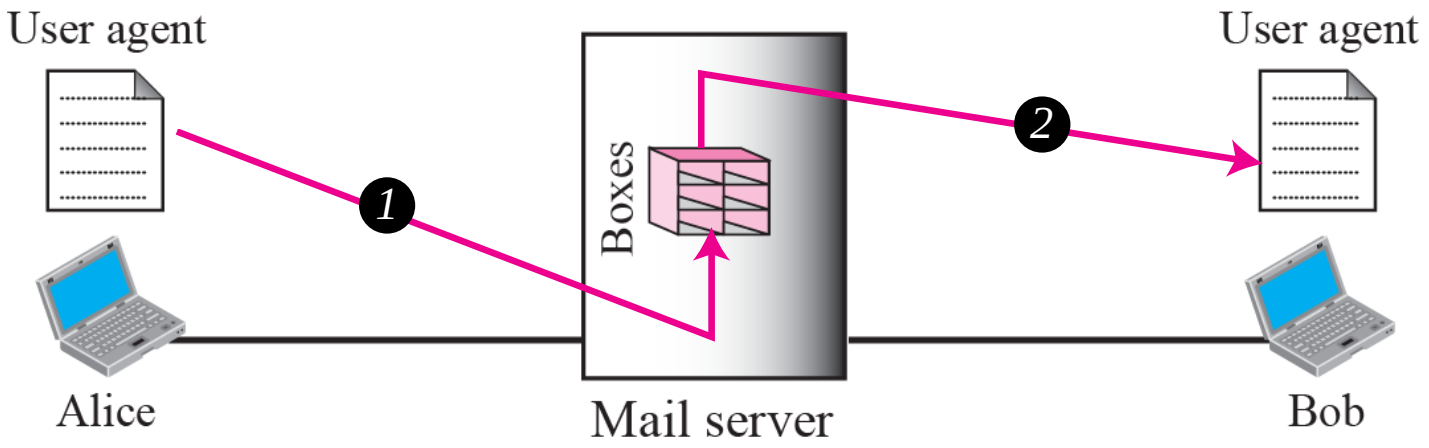
Mail Transfer Agent

Mail Servers (MTA)

- **mailbox** contains incoming messages for user
- **message queue** of outgoing (to be sent) mail messages



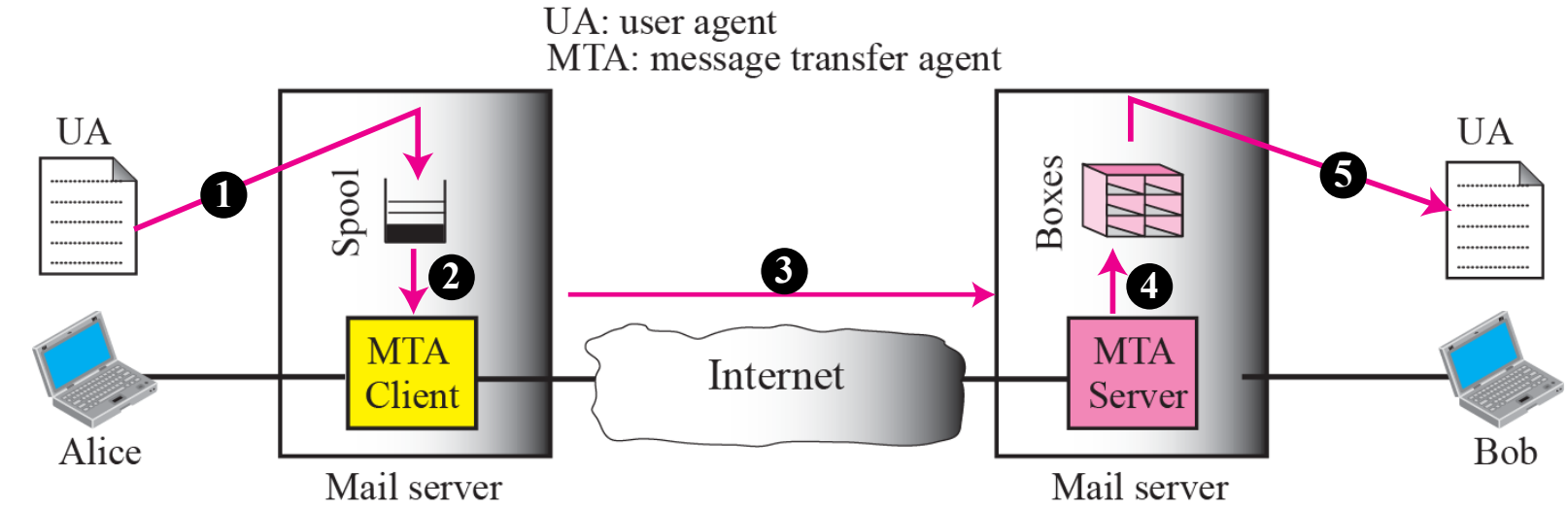
First scenario



Note

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

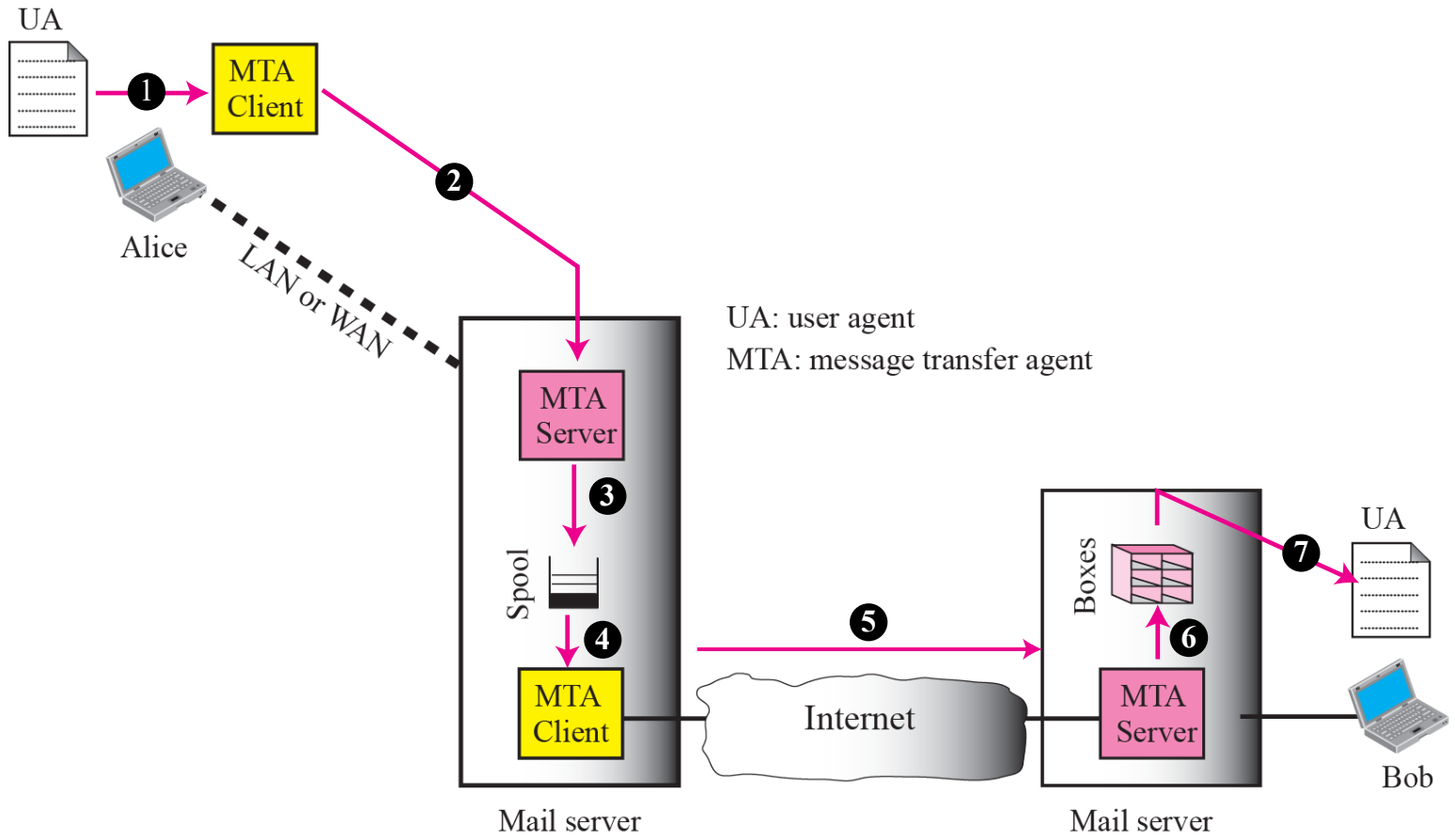
Second scenario



Note

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).

Third scenario

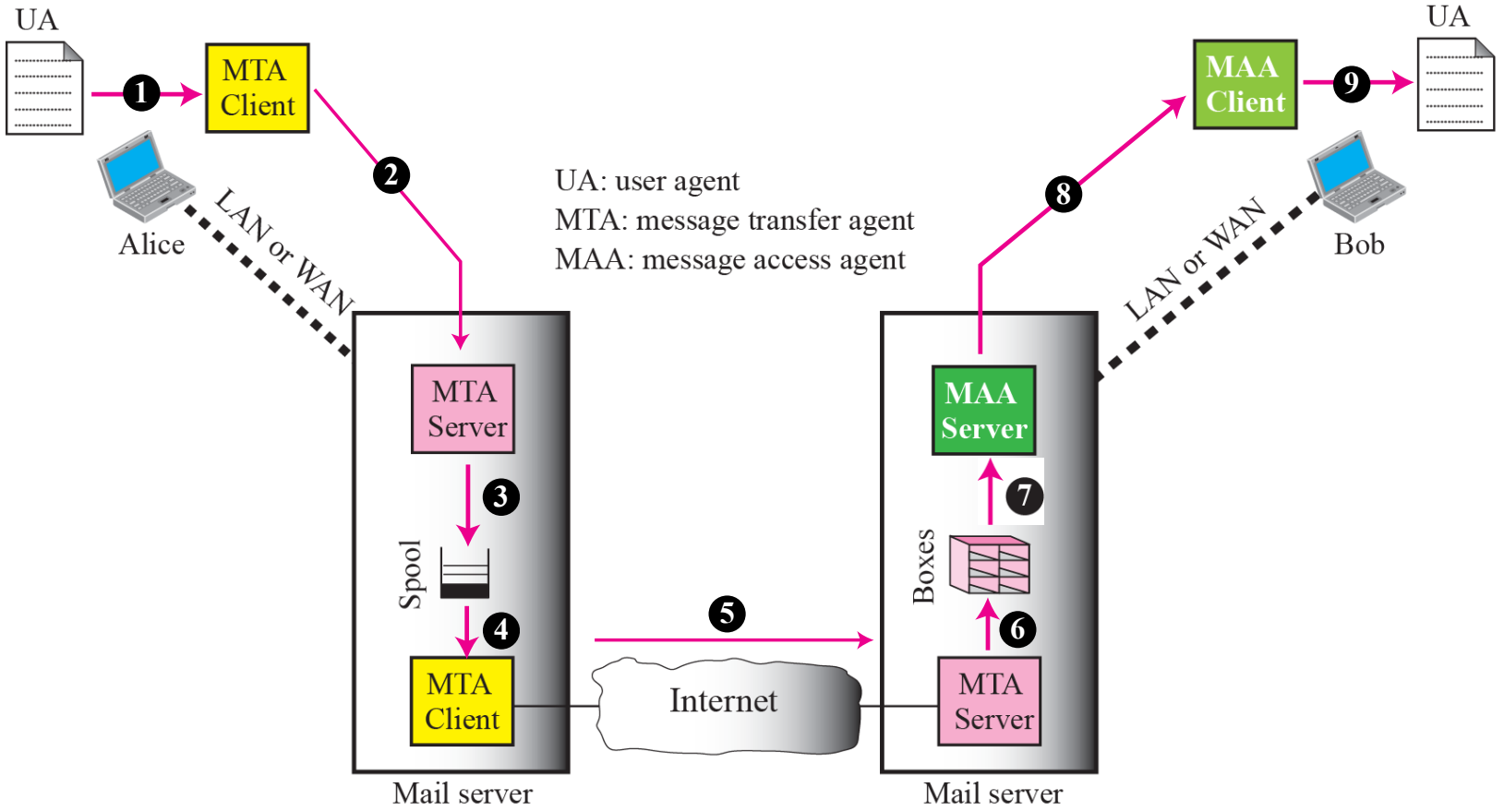


Third scenario

Note

When the sender is connected to the mail server via a LAN or a WAN, we need two UAs and two pairs of MTAs (client and server).

Fourth scenario



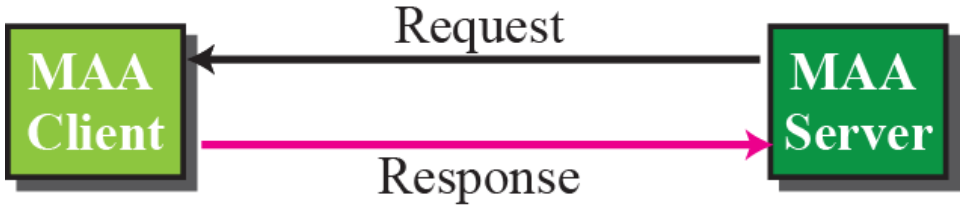
Fourth scenario

Note

When both sender and receiver are connected to the mail server via a LAN or a WAN, we 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.

Push *vs.* Pull

a. Client pushes messages

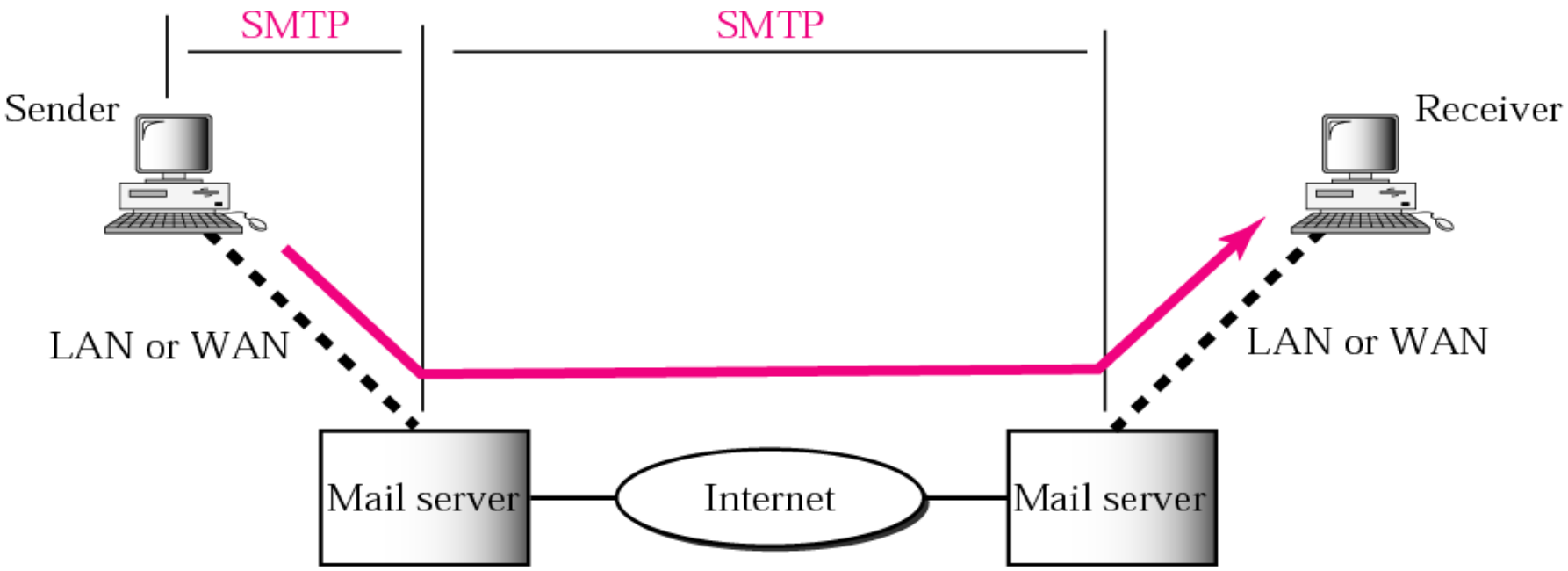


b. Client pulls messages

SMTP

- Simple Mail Transfer Protocol
- The standard for e-mail transmissions across the Internet.
- RFC 821 defines the SMTP protocol
- RFC 822 defines what a mail message looks like
- The protocol used today is ESMTP
  - Defined in RFC 2821.

SMTP



SMTP defines how MTAs exchange message

SMTP

- Relatively simple text-based protocol
- Client-server protocol
  - The client transmits an email message to the server
- SMTP uses TCP port 25 or 2525
- SMTP is a "push" protocol that does not allow one to "pull" messages from a remote server on demand
  - To do a pull (i.e. receive) a mail client must use POP3 or IMAP

SMTP

- command/response interaction
  - commands: ASCII text
  - response: status code and phrase

SMTP

SMTP Commands

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

SMTP

SMTP Responses

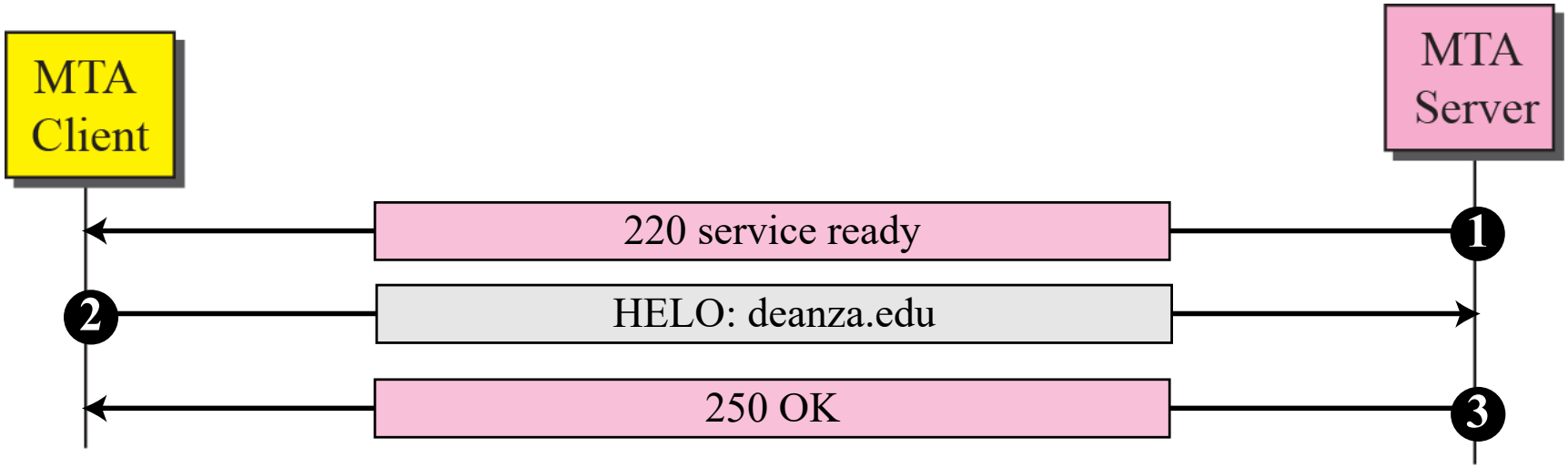
Code	*Description*
	Positive Completion Reply
211	System status or help reply
214	Help message
220	Service ready
221	Service closing transmission channel
250	Request command completed
251	User not local; the message will be forwarded
	Positive Intermediate Reply
354	Start mail input
	Transient Negative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

SMTP

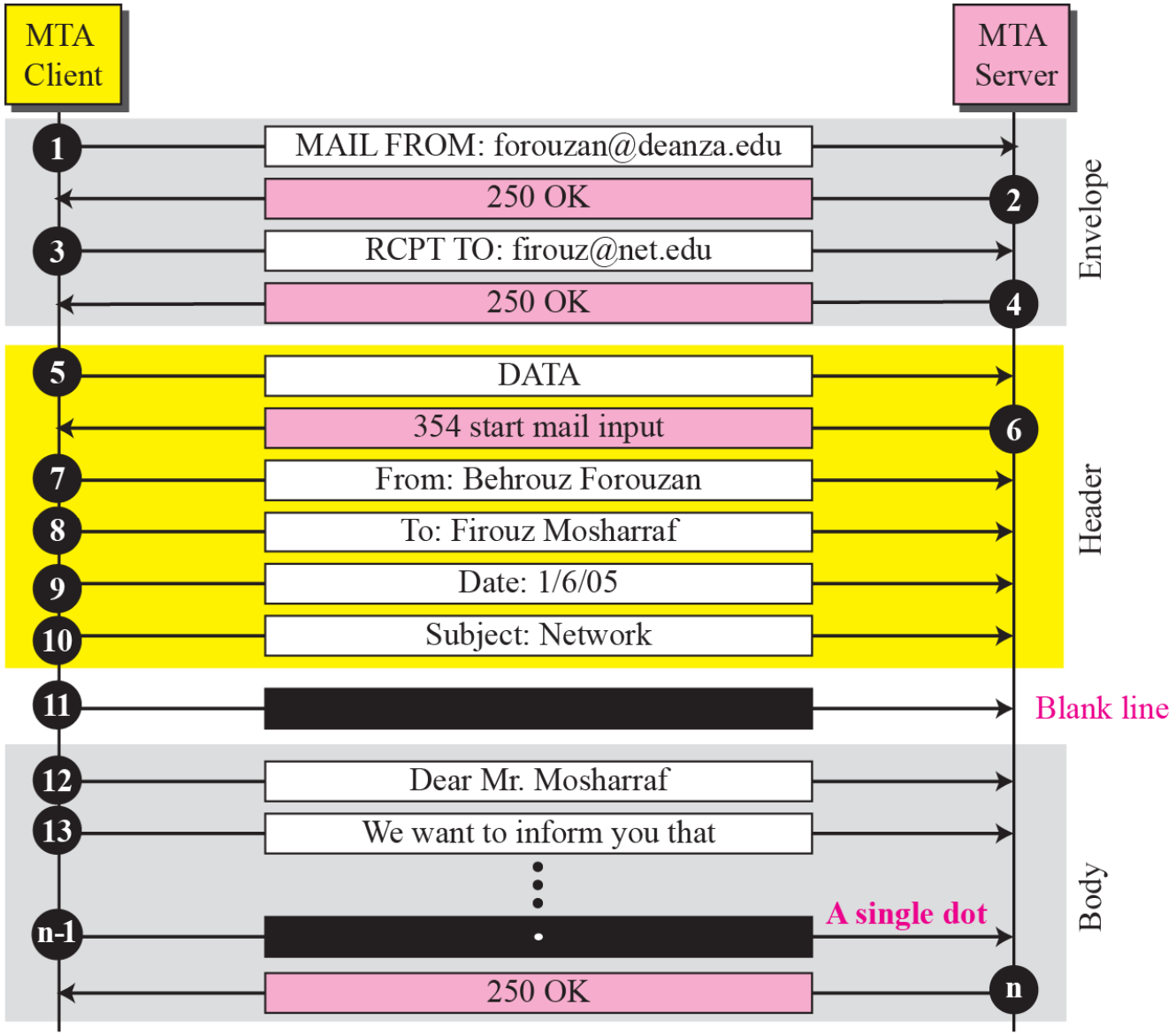
- **Three phases of transfer:**
  - **Handshaking (greeting)**
  - **Transfer of messages**
  - **Closure**

Handshaking

Connection establishment

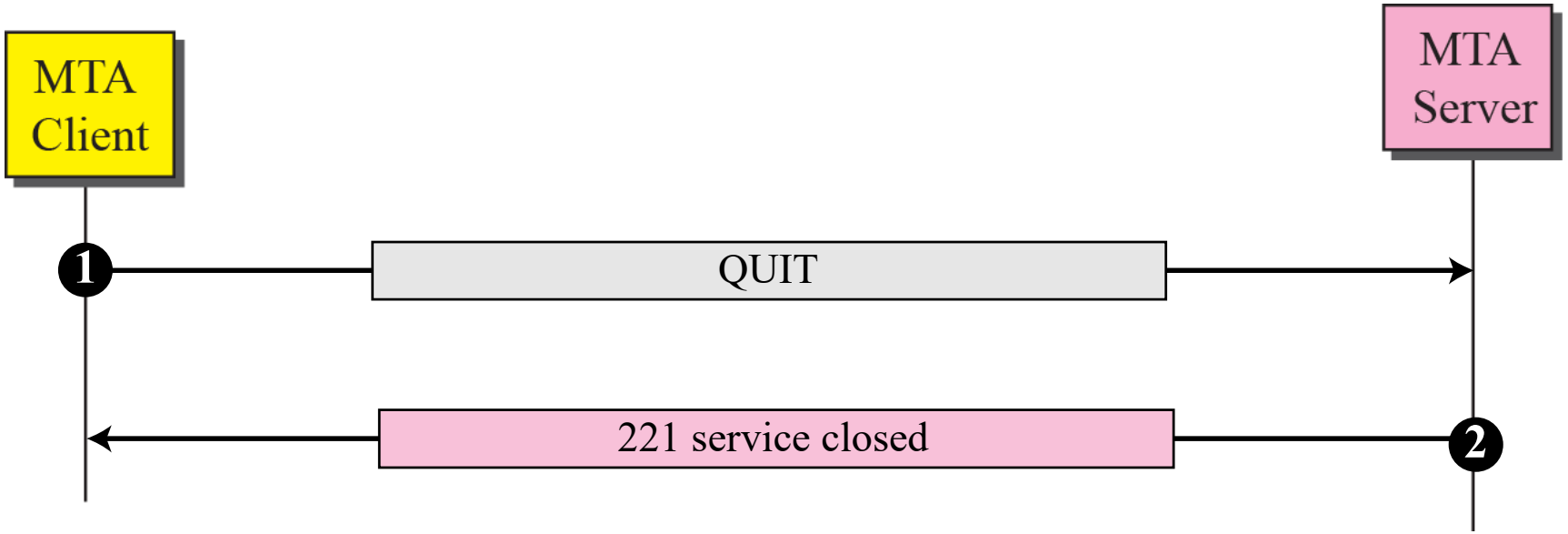


Message Transfer



Closure

Connection termination



Example

```
$ telnet mail.adelphia.net 25
Trying 68.168.78.100...
Connected to mail.adelphia.net (68.168.78.100).
```

Example

========== Connection Establishment ==========

220 mta13.adelphia.net SMTP server ready Fri, 6 Aug 2004 . . .

HELO mail.adelphia.net

250 mta13.adelphia.net

========== Envelope ==========

MAIL FROM: forouzanb@adelphia.net

250 Sender <forouzanb@adelphia.net> Ok

RCPT TO: forouzanb@adelphia.net

250 Recipient <forouzanb@adelphia.net> Ok

========== Header and Body ==========

DATA

354 Ok Send data ending with <CRLF>.<CRLF>

From: Forouzan

TO: Forouzan

This is a test message

to show SMTP in action.

.

Example

========== Connection Termination==========

250 Message received: adelphia.net@mail.adelphia.net

QUIT

221 mta13.adelphia.net SMTP server closing connection

Connection closed by foreign host.

SMTP: final words

- SMTP uses **persistent** TCP connections
- SMTP requires message (header & body) to be in **7-bit ASCII**
- SMTP server uses `CRLF.CRLF` to determine end of message

Comparison with HTTP:

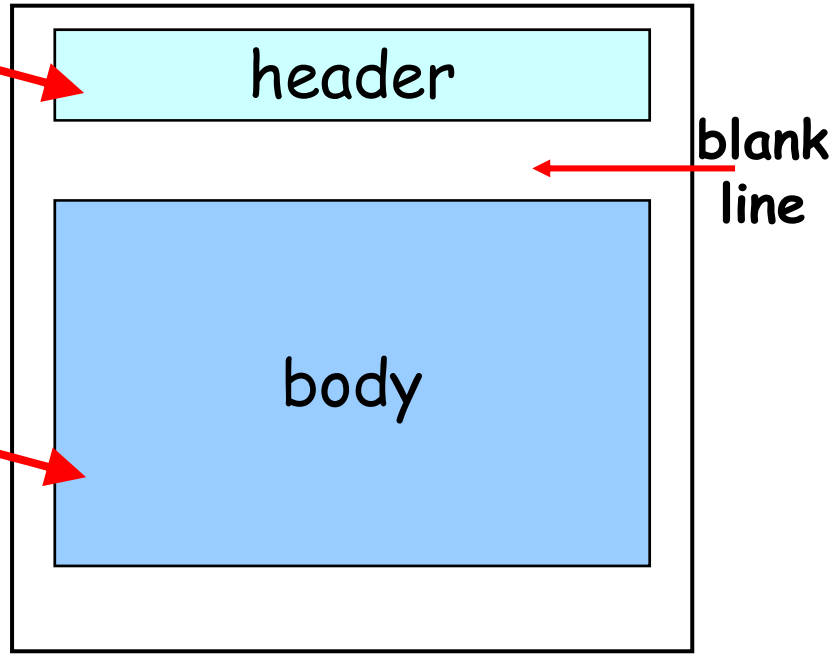
- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

Mail message format

- E-mail messages have two parts
  - A header, in 7-bit U.S. ASCII text
  - A body, also represented in 7-bit U.S. ASCII text
- Header
  - Lines with "type: value"
  - "From: teacher@mybit.edu.cn"
  - "To: jrex@mybit.edu.cn"
  - "Subject: Go Tigers!"

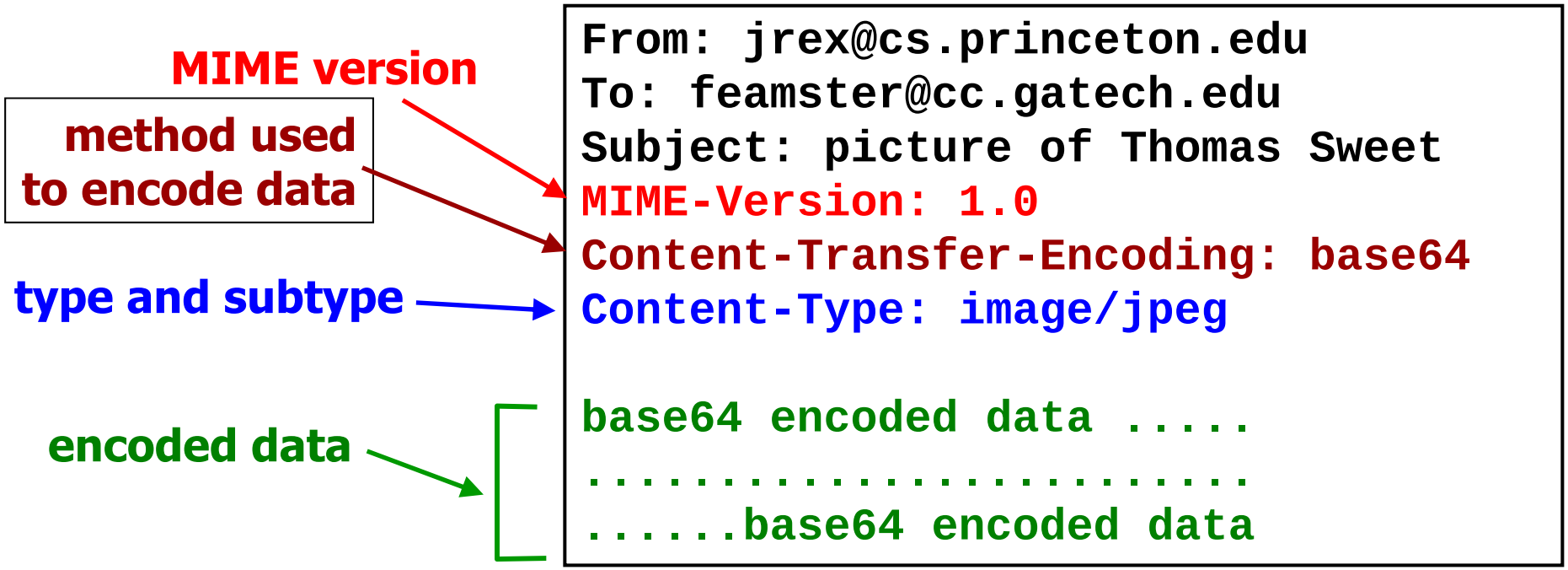
different from SMTP commands!

- Body
  - The text message
  - No particular structure or meaning



MIME

- **MIME: Multipurpose Internet Mail Extensions**
  - Sending Non-Text Data
  - multimedia mail extension, RFC 2045, 2056
- **Additional lines in message header declare MIME content type**

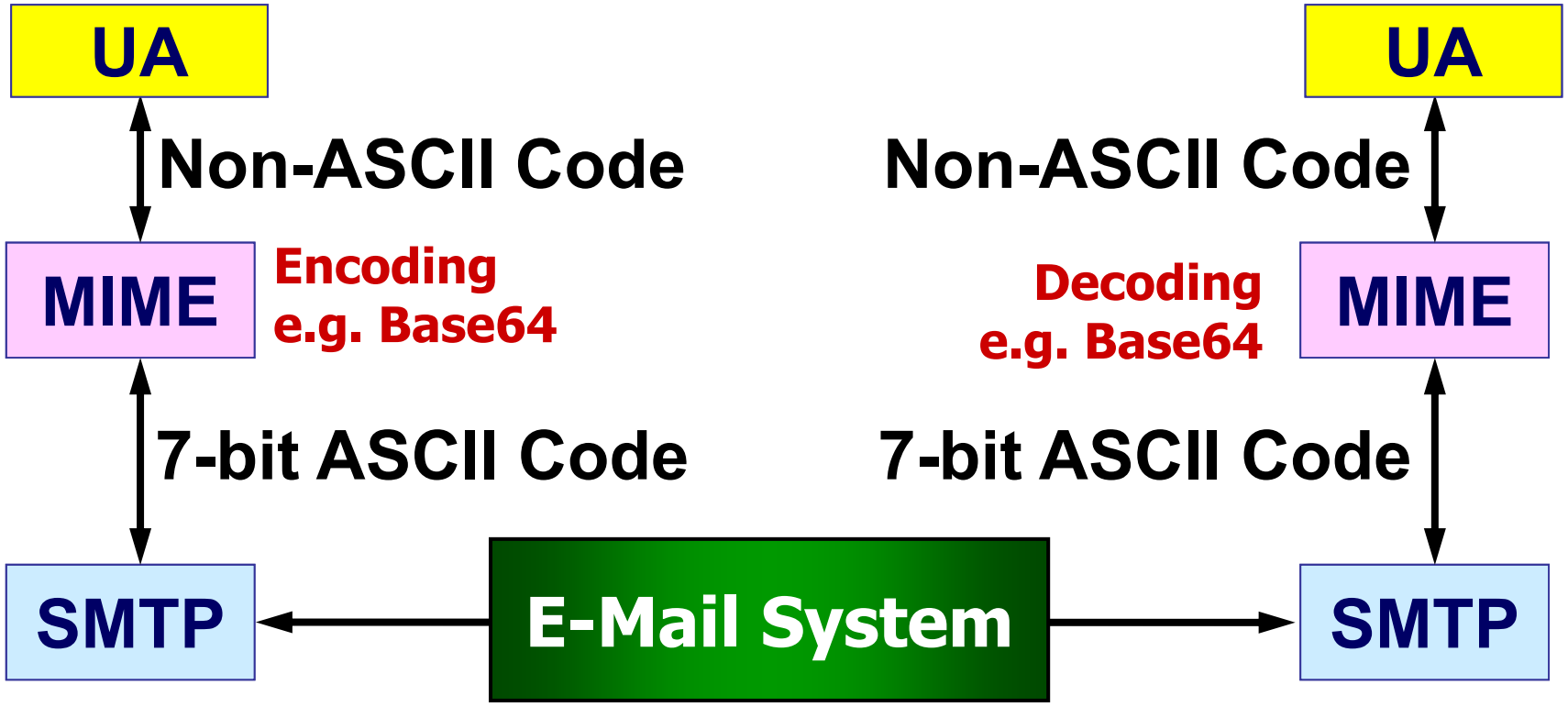


MIME

The MIME types and subtypes defined in RFC 2045

Type	*Subtype*	*Description*
Text	Plain	Unformatted
	HTML	HTML format (see Appendix E)
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)

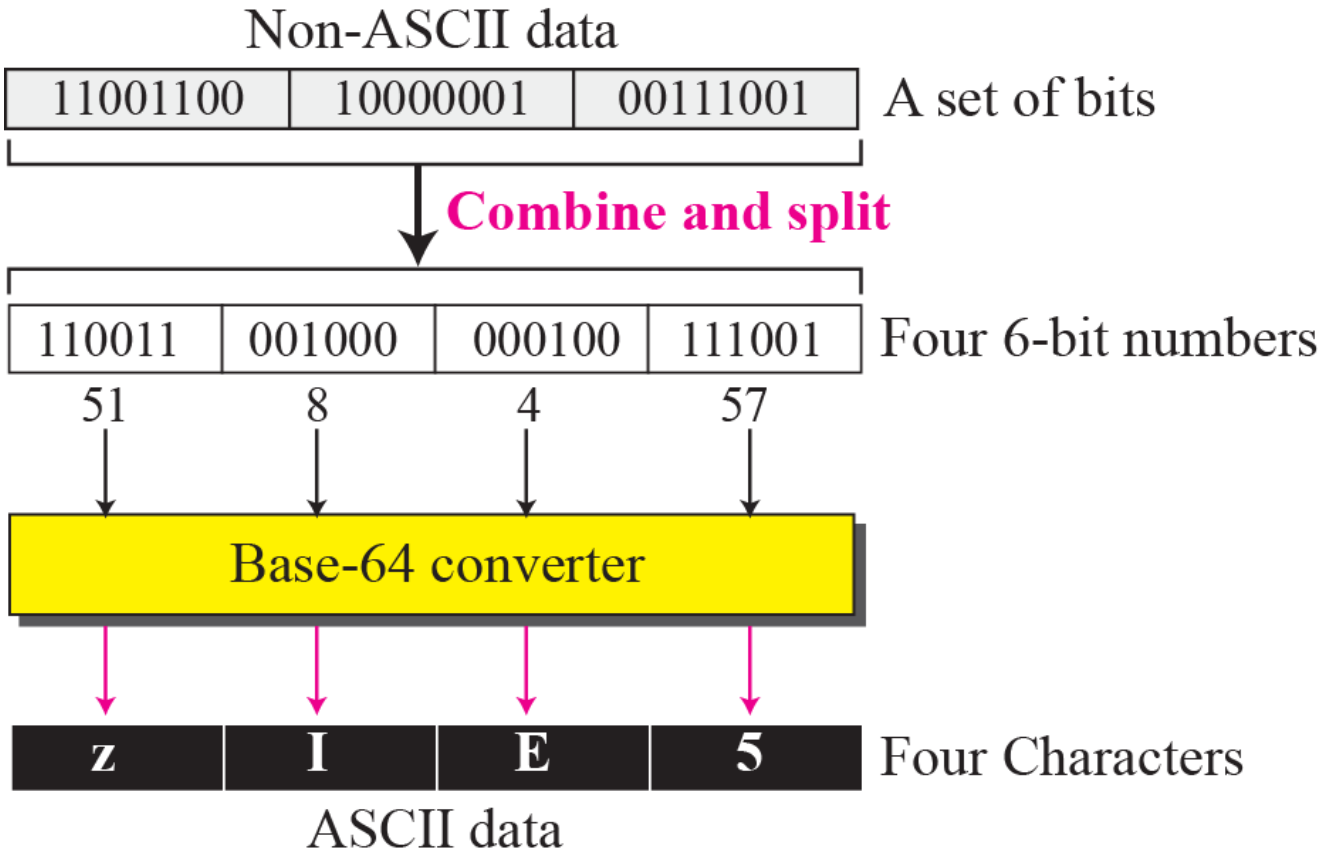
MIME


UA
UA
Non-ASCII Code
Non-ASCII Code
MIME
Encoding
e.g. Base64
Decoding
e.g. Base64
MIME
7-bit ASCII Code
7-bit ASCII Code
SMTP
E-Mail System
SMTP

Content-Transfer-Encoding

Type	*Description*
7bit	NVT ASCII characters and short lines
8bit	Non-ASCII characters and short lines
Binary	Non-ASCII characters with unlimited-length lines
Base64	6-bit blocks of data are encoded into 8-bit ASCII characters
Quoted-printable	Non-ASCII characters are encoded as an equal sign plus an ASCII code

Content-Transfer-Encoding

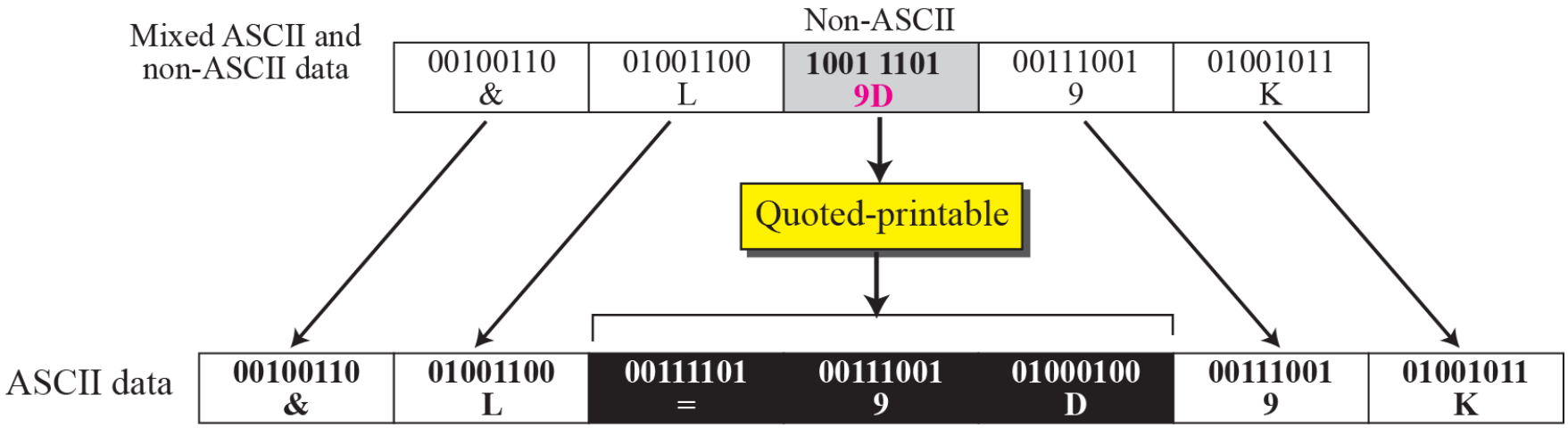


Base64

Content-Transfer-Encoding

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**		

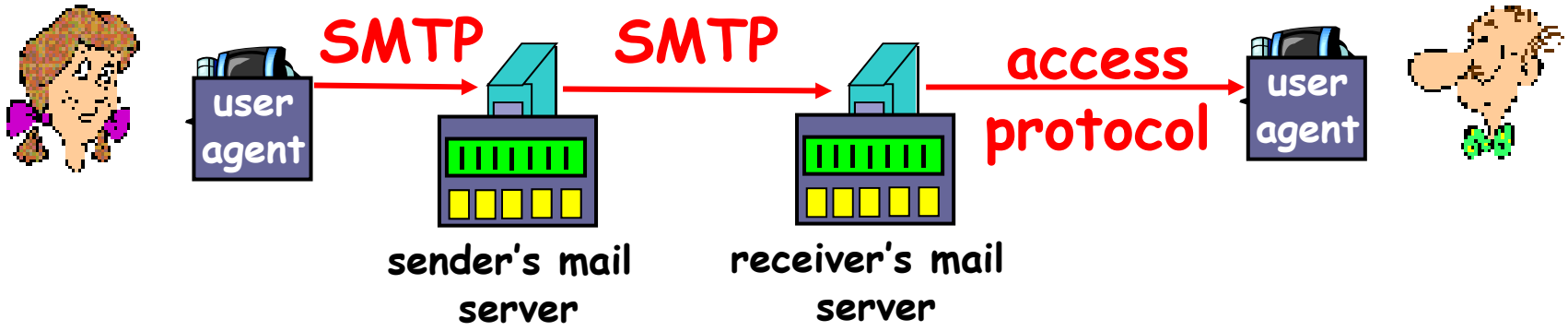
Content-Transfer-Encoding



Quoted printable

Mail Access Agent (MAA)

- The client must **pull** messages from the server.
- Two message access protocols available:
  - Post Office Protocol, version 3 (**POP3**)
  - Internet Mail Access Protocol, version 4 (**IMAP 4**)



POP 3

- Local e-mail clients use the Post Office Protocol version 3 (POP 3)
  - Retrieve (read) e-mail from a remote server over a TCP/IP connection
  - An application-layer Internet standard protocol
- Two phases:
  - Authorization (agent <-->server)
  - Download

POP3 protocol

Authorization phase

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

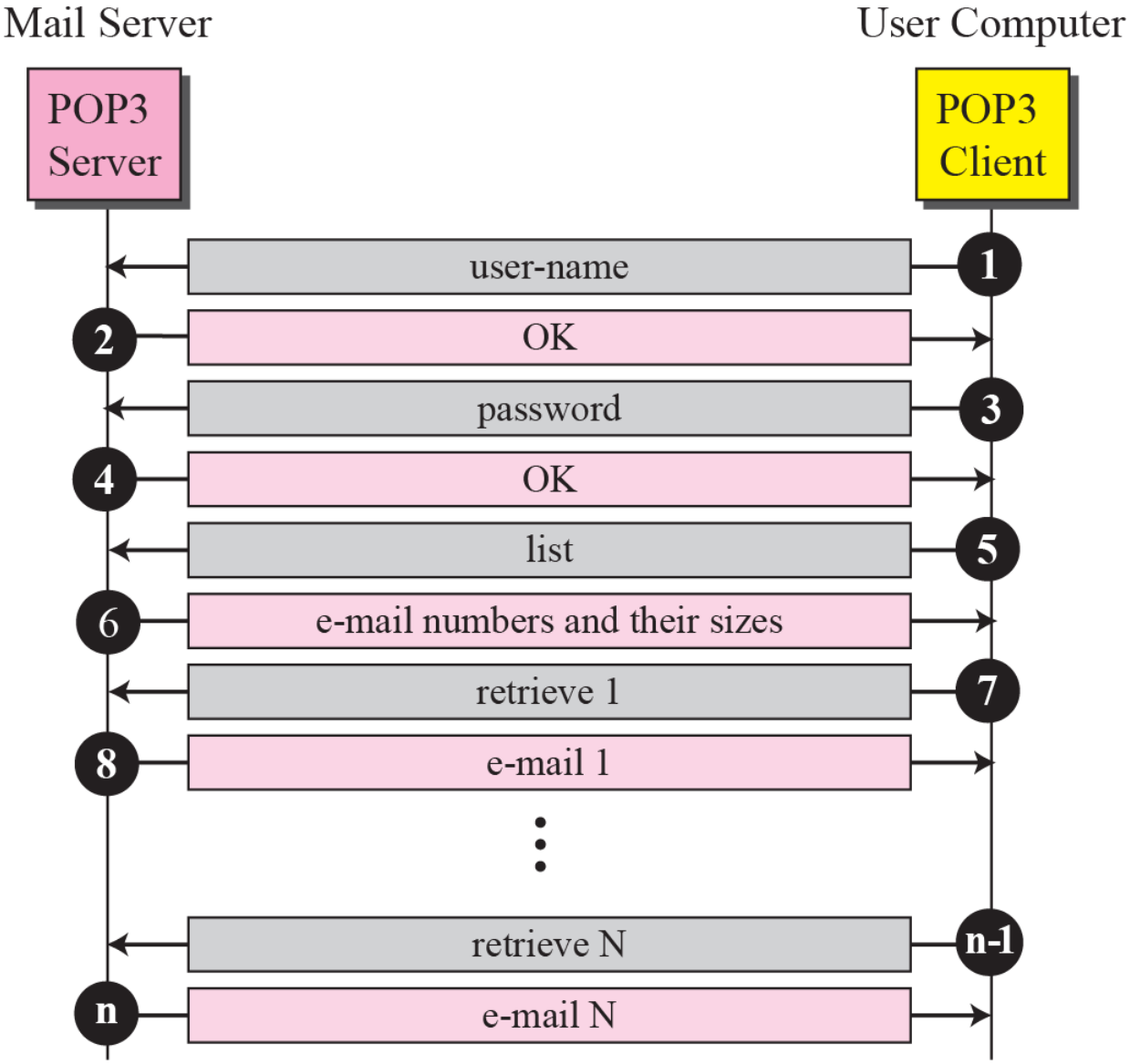
Transaction phase, client:

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

POP3 protocol



POP3 Commands

命令	说明
User <用户邮箱名>	给出用户邮箱名
Pass <口令串>	给出用户口令
Quit	退出会话
Stat	询问邮件总数及长度
List	列出邮件
Retr	检索邮件
Dele	给邮件做“删除”标记
Noop	空操作
Rset	复位，取出所有带“删除”标记的邮件

POP

- The design of POP3 and its procedures supports end-users with intermittent connections (such as dial-up connections)
  - Allows users to retrieve e-mail when connected
  - View and manipulate the retrieved messages without needing to stay connected
- Although most clients have an option to *leave mail on server*, e-mail clients using POP3 generally:
  - Connect
  - Retrieve all messages
  - Store them on the user's PC as new messages
  - Delete them from the server
  - Disconnect.

IMAP

- **Internet Mail Access Protocol**
- **supports both *connected* and *disconnected* modes of operation.**
  - **E-mail clients using IMAP generally leave messages on the server until the user explicitly deletes them.**
- **This and other facets of IMAP operation allow multiple clients to access the same mailbox**

POP3 and IMAP

Feature	POP3	IMAP
Where is protocol defined?	RFC 1939	RFC 2060
Which TCP port is used?	110	143
Where is e-mail stored?	User's PC	Server
Where is e-mail read?	Off-line	On-line
Connect time required?	Little	Much
Use of server resources?	Minimal	Extensive
Multiple mailboxes?	No	Yes
Who backs up mailboxes?	User	ISP
Good for mobile users?	No	Yes
User control over downloading?	Little	Great
Partial message downloads?	No	Yes
Are disk quotas a problem?	No	Could be in time
Simple to implement?	Yes	No
Widespread support?	Yes	Growing

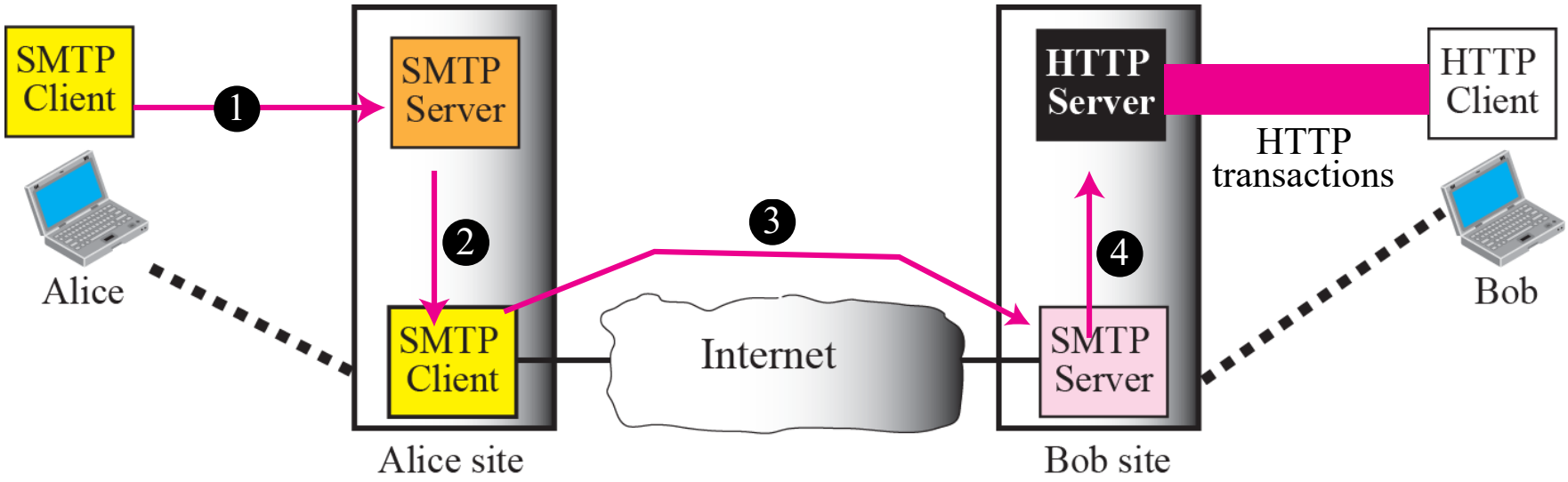
Web-Based E-Mail

- Some websites such as Hotmail, Yahoo, and Google provide email service to anyone who accesses the site.
- Mail transfer and retrieval requires the use of HTTP.

Web-Based E-Mail

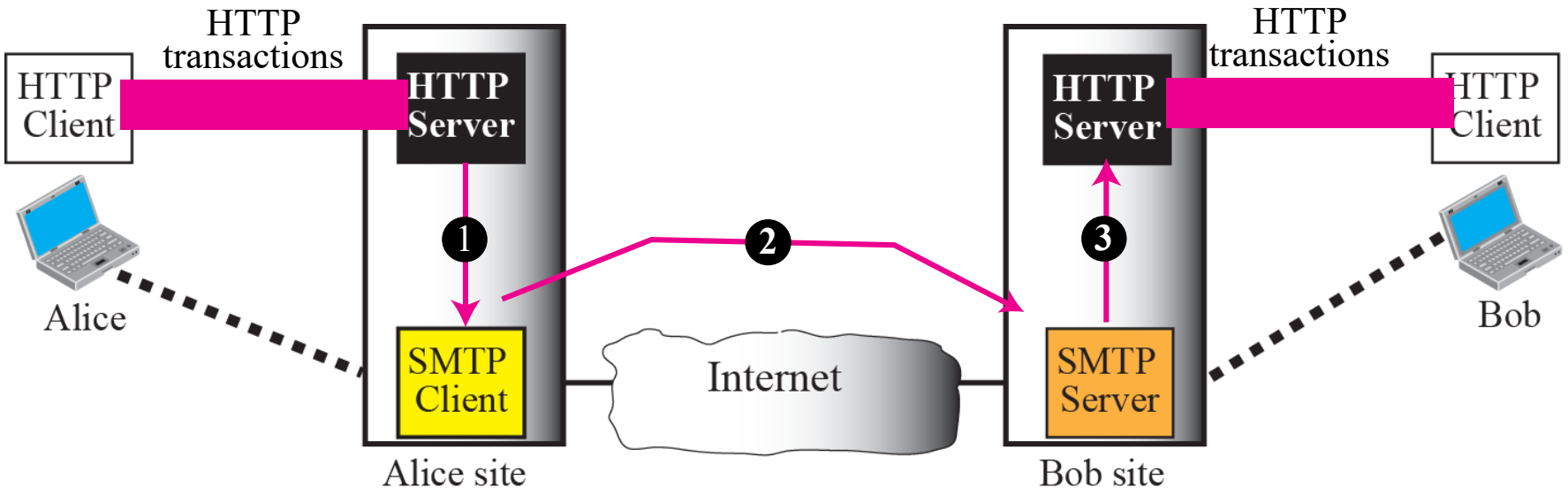
- User agent is an ordinary Web browser
  - User communicates with server via HTTP
  - E.g., Gmail, Yahoo mail, and Hotmail
- Reading e-mail
  - Web pages display the contents of folders
  - … and allow users to download and view messages
  - "GET" request to retrieve the various Web pages
- Sending 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)

Web-Based E-Mail



Web-based e-mail, case 1

Web-Based E-Mail



Web-based e-mail, case 2

E-Mail Security

- Does not provide any security provisions
- However, e-mail exchanges can be secured using two application-layer securities designed in particular for e-mail systems.
  - Pretty Good Privacy (PGP)
  - Secure MIME (SMIME)

Chapter 7: Roadmap

- Application and Application Layer
- C/S and P2P Application Architectures
- DNS
- FTP
- E-Mail and SMTP/POP/IMAP
- WWW and HTTP
- DHCP

WWW and HTTP

- World Wide Web, "Web"
- Tim Berners-Lee at CERN in 1991

Tim Berners-Lee

- World Wide Web (WWW): a distributed database of "pages" linked through Hypertext Transport Protocol (HTTP)
- First HTTP implementation – 1990
- Tim Berners-Lee at CERN
- HTTP/0.9 – 1991
  Simple GET command for the Web
- HTTP/1.0 –1992
  Client/Server information, simple caching
- HTTP/1.1 - 1996

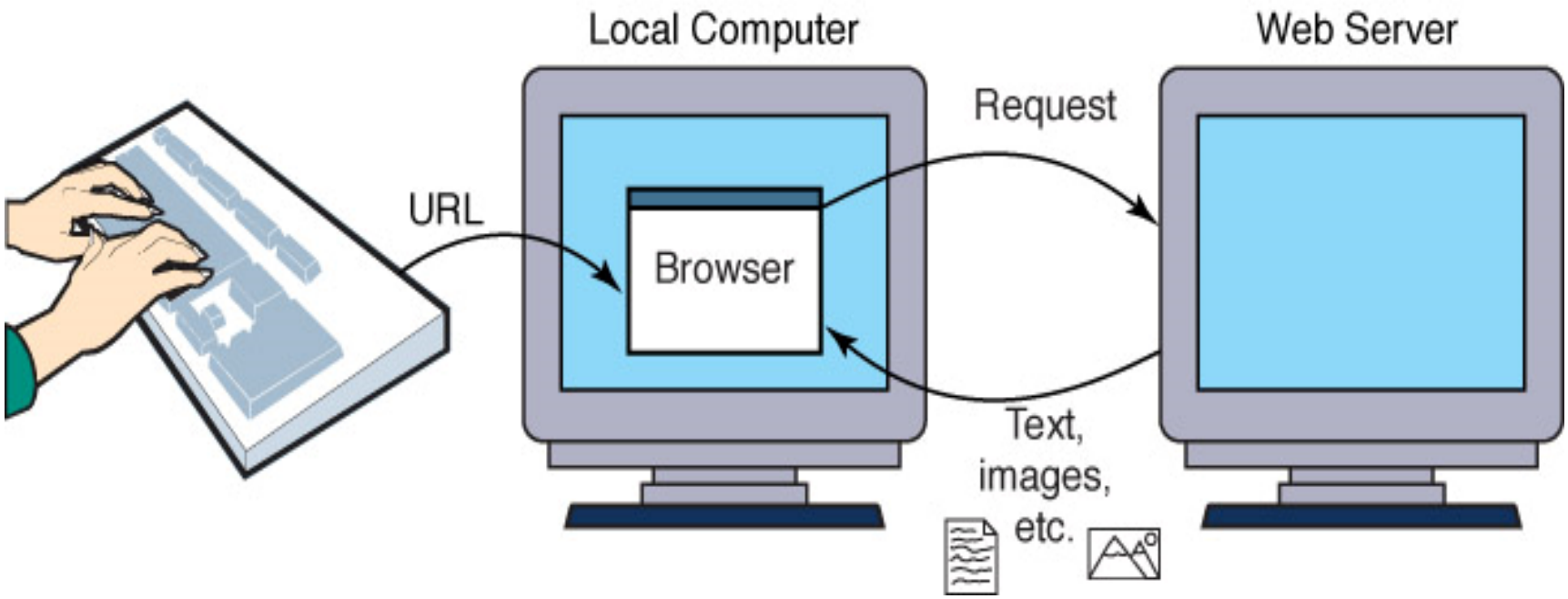
The World Wide Web

- **The Web**
  - An infrastructure of information combined and the network software used to access it
- **Web page**
  - A document that contains or references various kinds of data
- **Website**
  - A collection of related web pages
- **Web browser**
  - a software tool that retrieves and displays Web pages
- **Web server**
  - A computer set up to respond to requests for Web pages

The World Wide Web

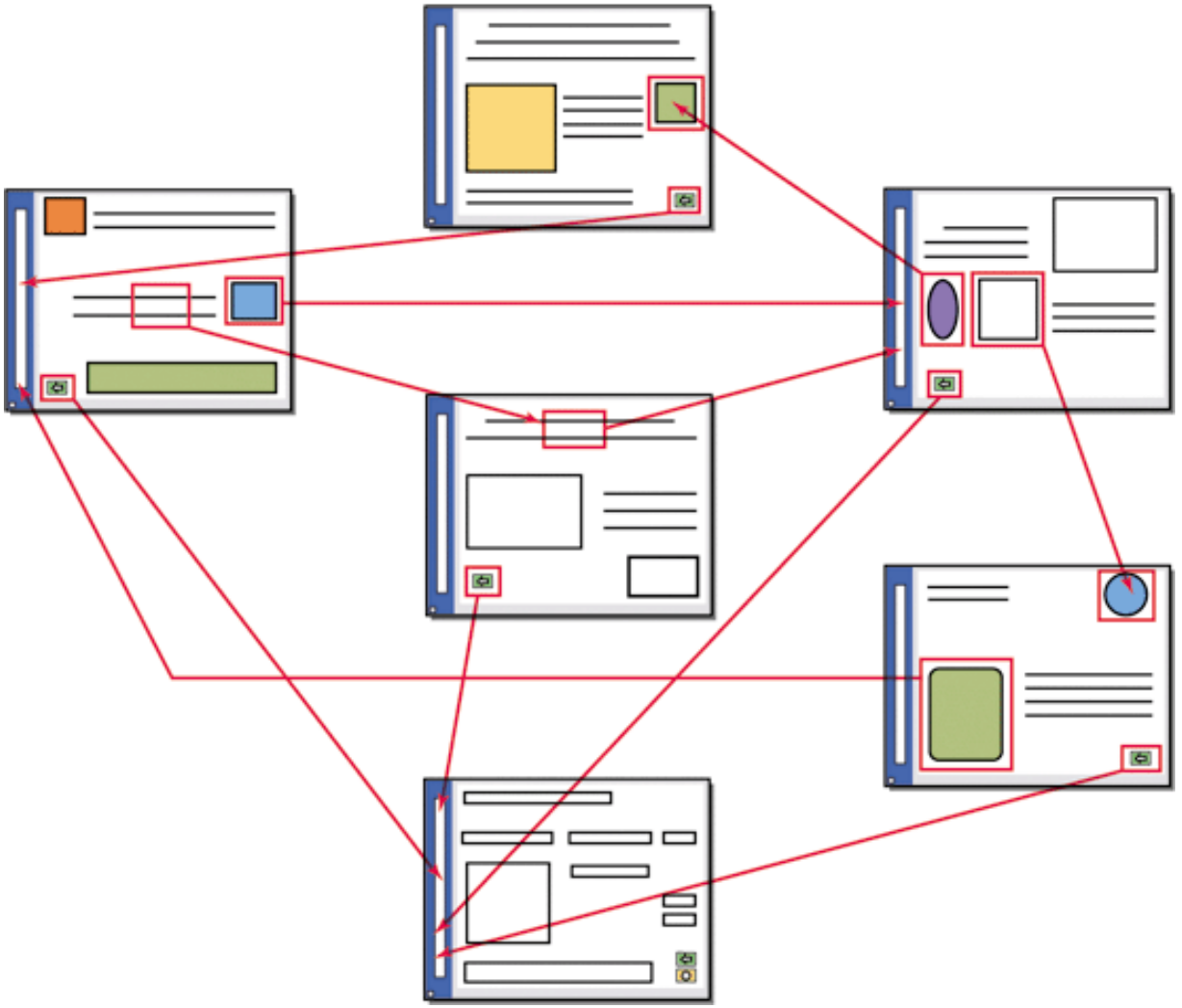
- **Uniform Resource Locator (URL)**
  - A standard way of specifying the location of a Web page, containing the hostname, "/", and a file
- **Links**
  - A connection between one web page and another
- **Hyperlink**
  - Element in an electronic document —a word, phrase, or image—that when clicked, opens a related document
  - Hypertext, hypermedia
- **Hypertext Transfer Protocol (HTTP)**
  - Protocol controls communication between Web clients and servers

The World Wide Web



A browser retrieving a Web page

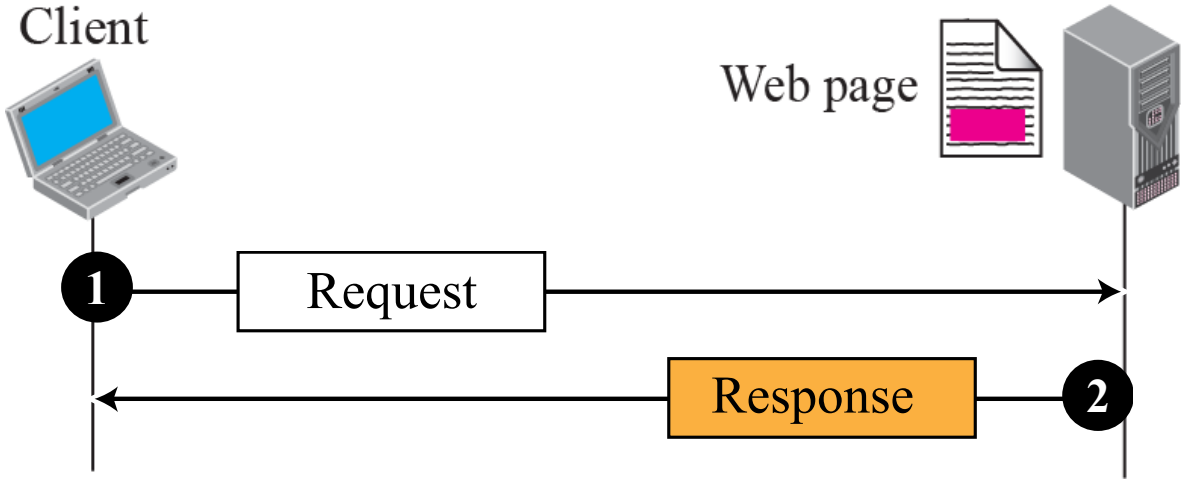
The World Wide Web



The hyperlinks

Example 1

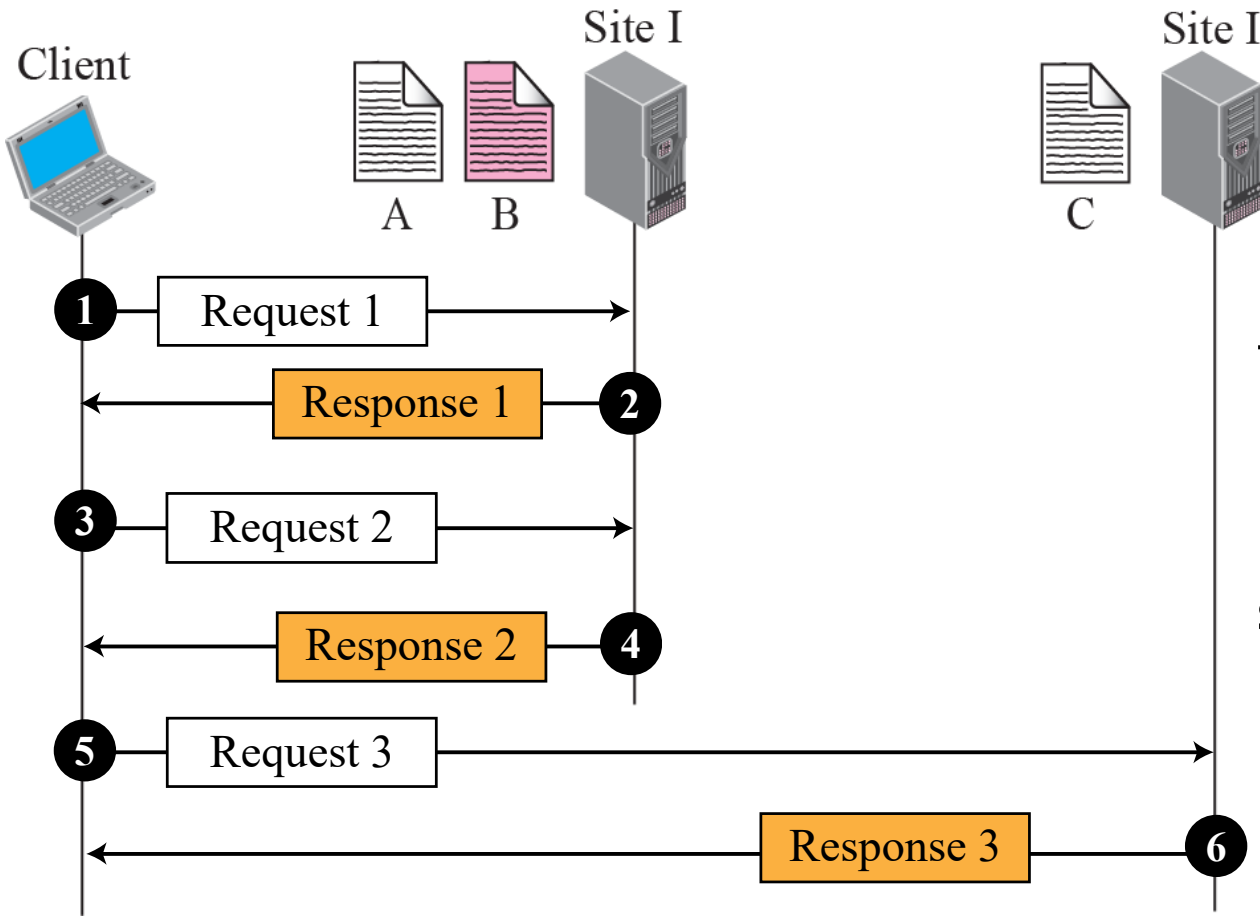
- To retrieve a Web page that contains the biography of a famous character with some pictures, which are embedded in the page itself.
- Since the whole document is a simple Web page, it can be retrieved using one single request/ response transaction



Example 2

- Now assume we need to retrieve a scientific document that contains one reference to another text file and one reference to a large image.
- The main document and the image are stored in two separate files in the same site (file A and file B); the referenced text file is stored in another site (file C).
- Since we are dealing with three different files, we need three transactions if we want to see the whole document. The first transaction (request/response) retrieves a copy of the main document (file A), which has a reference (pointer) to the second and the third files.

Example 2



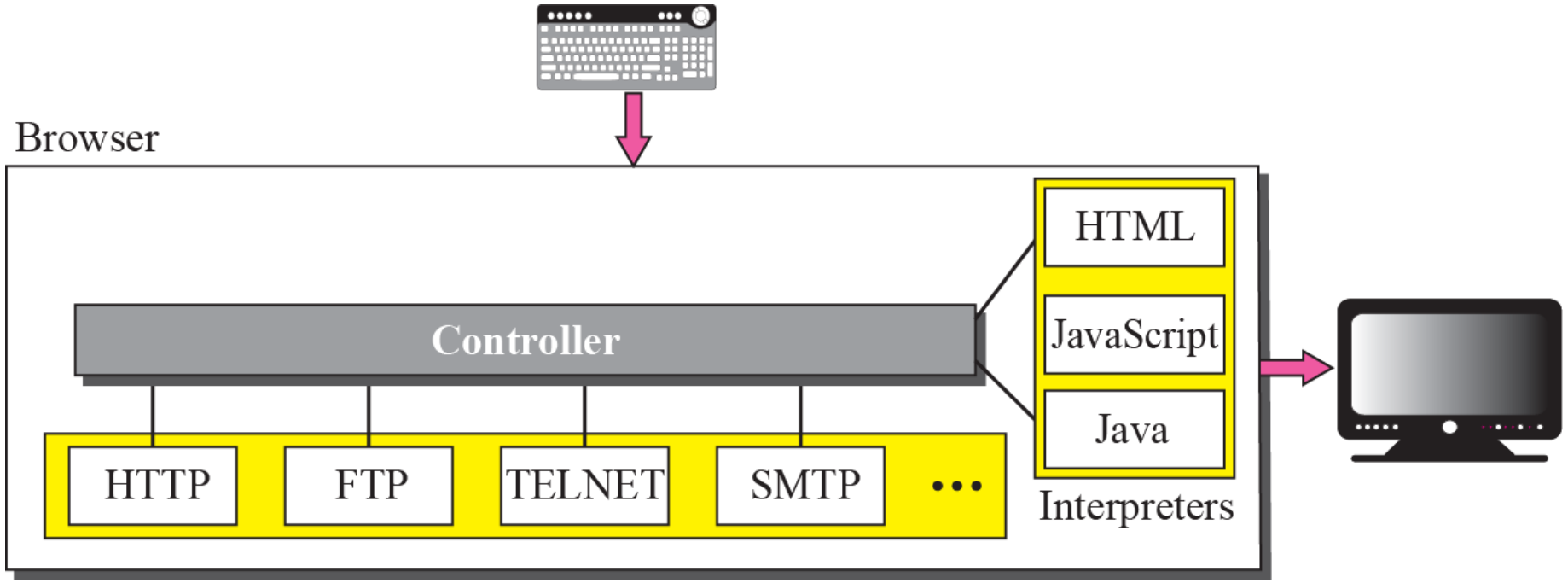
file A and file B are in the same site, the referenced text file C is stored in another site.

We need three transactions if we want to see the whole document.

Example 3

- A very important point:
  - file A, file B, and file C are independent Web pages, each with independent names and addresses.
  - Although references to file B or C are included in file A, it does not mean that each of these files cannot be retrieved independently.
  - The second user can retrieve file B with one transaction. The third user can retrieve file C with one transaction.

Web Browser


Browser
Controller
HTTP
FTP
TELNET
SMTP
...
HTML
JavaScript
Java
Interpreters

Web Components

- **Infrastructure:**
  - Clients, Servers, Proxies
- **Contents:**
  - Individual objects (files, etc.)
  - Web sites (coherent collection of objects)
- **Implementation**
  - **HTML:** formatting content
  - **URL:** naming content
  - **HTTP:** protocol for exchanging content

HTML: HyperText Markup Language

- A Web page has:
  - Base HTML file
  - Referenced objects (e.g., images)

- HTML has several functions:
  - Format text
  - Reference images
  - Embed hyperlinks (HREF)

Sample Webpage HTML Structure

```
<HTML>
  <HEAD>
    <TITLE>The title of the webpage</TITLE>
  </HEAD>
  <BODY>
    <P>Body of the webpage</P>
  </BODY>
</HTML>
```

Example Web Page

HTML code

```
<center>
<img src="exclamation.gif"><br>
<p>This is an example of <em>interpreted</em> <b>HTML</b> code.</p>
</center>
```

Browser display

This is an example of *interpreted* **HTML** code.

Example Web Page

page.html

Harry Potter Movies

As you all know, the new HP book will be out in June and then there will be a new movie shortly after that…

hpface.jpg

"Harry Potter and the Bathtub Ring"

castle.gif

URL

- **URL:** Uniform Resource Locators
- **URI:** Uniform Resource Identifiers
- **URN:** Uniform Resource Names

Note

Every resource available on the Web has an address that may be encoded by a URI

URL

protocol://hostname[:port]/directorypath/resource

protocol	**http, ftp, https, smtp, rtsp, *etc*.**
hostname	**DNS name, IP address**
port	**Defaults to protocol's standard port. *e.g.* http: 80 https: 443**
directory path	**Hierarchical, reflecting file system**
resource	**Identifies the desired resource** **Can also extend to program executions:** `http://us.f413.mail.yahoo.com/ym/ShowLetter?box=%40B%40Bulk&MsgId=2604_1744106_29699_1123_1261_0_28917_3552_1289957100&Search=&Nhead=f&YY=31454&order=down&sort=date&pos=0&view=a&head=b`

URL: Example

Components of a URL

http://www.course.com/cca/ch1/index1.html

Protocol	Web server	Domain name	Location on server	Requested file
http://	www	course.com/	cca/ch1	Index1.html

Where is the port?

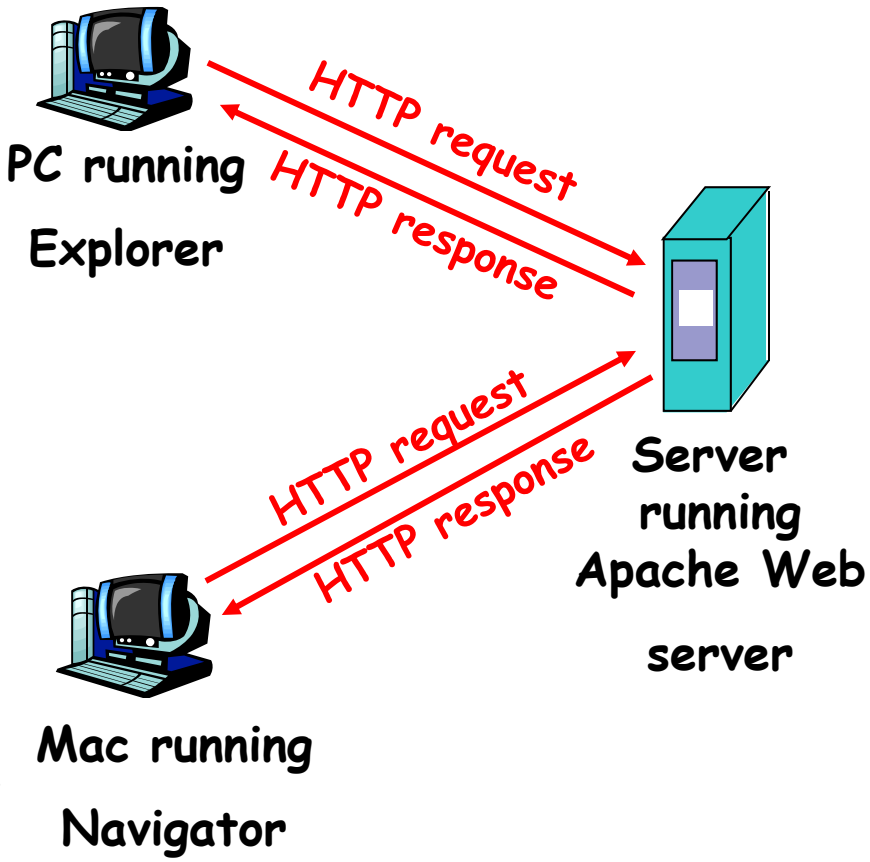
The port is optional when it is the default port for the given scheme or service/protocol.

HTTP: HyperText Transfer Protocol

- HTTP is a protocol used mainly to access data on the World Wide Web
- Request-response protocol
- Reliance on a global namespace
- Resource *metadata*
- Stateless
- ASCII format

HTTP

- **Client/Server model**
  - ***client:*** **browser that requests, receives, "displays" Web objects**
  - ***server:*** **Web server sends objects in response to requests**
- **HTTP 1.0: RFC 1945**
- **HTTP 1.1: RFC 2068**



HTTP

Uses TCP on port 80:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP 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

Nonpersistent HTTP

- At most one object is sent over a TCP connection.
- HTTP/1.0 uses nonpersistent HTTP

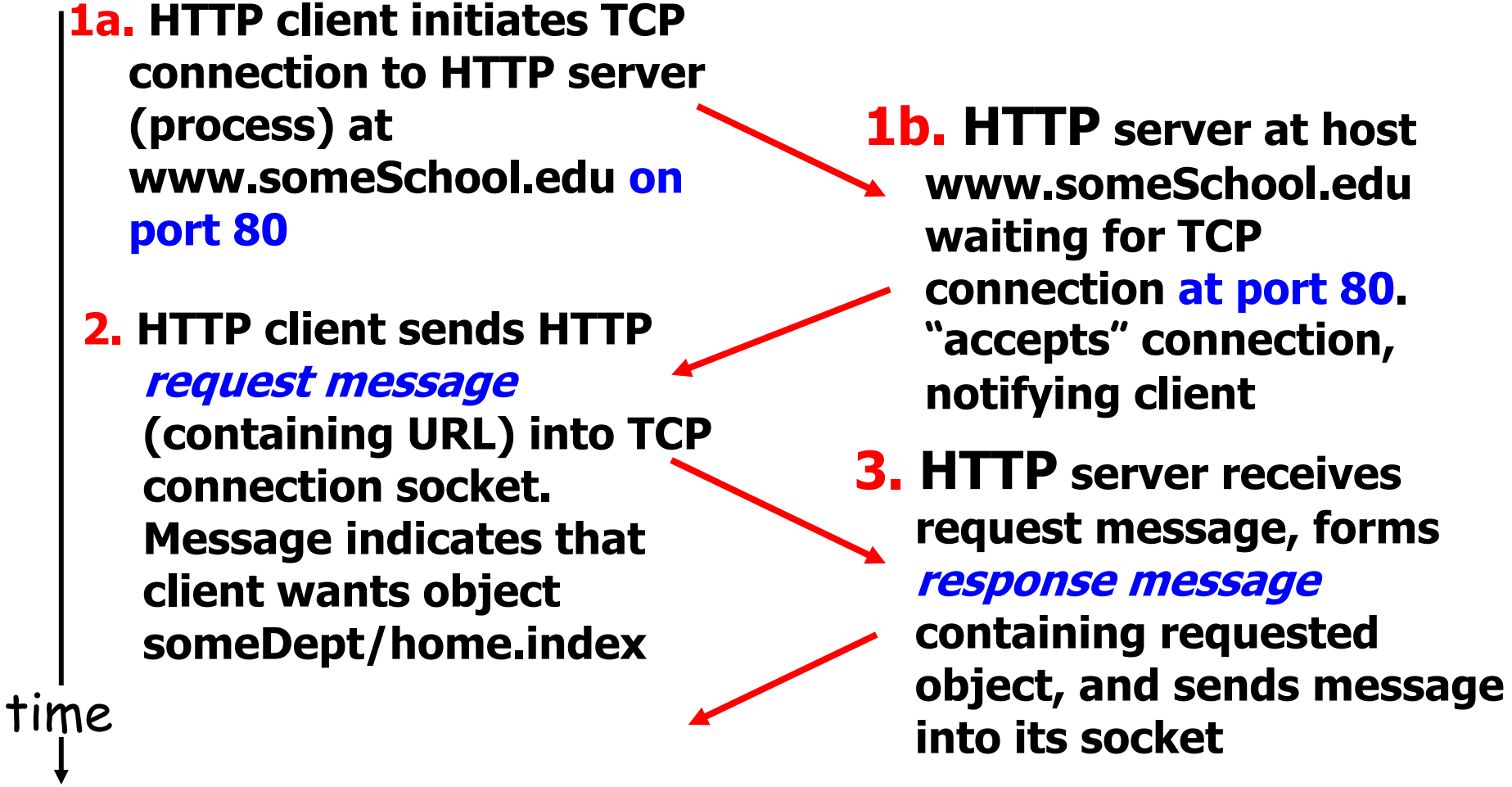
Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server.
- HTTP/1.1 uses persistent connections in default mode

Nonpersistent HTTP

Suppose user enters URL
www.someSchool.edu/someDept/index.html

(contains text, references to 10 jpeg images)



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

time

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

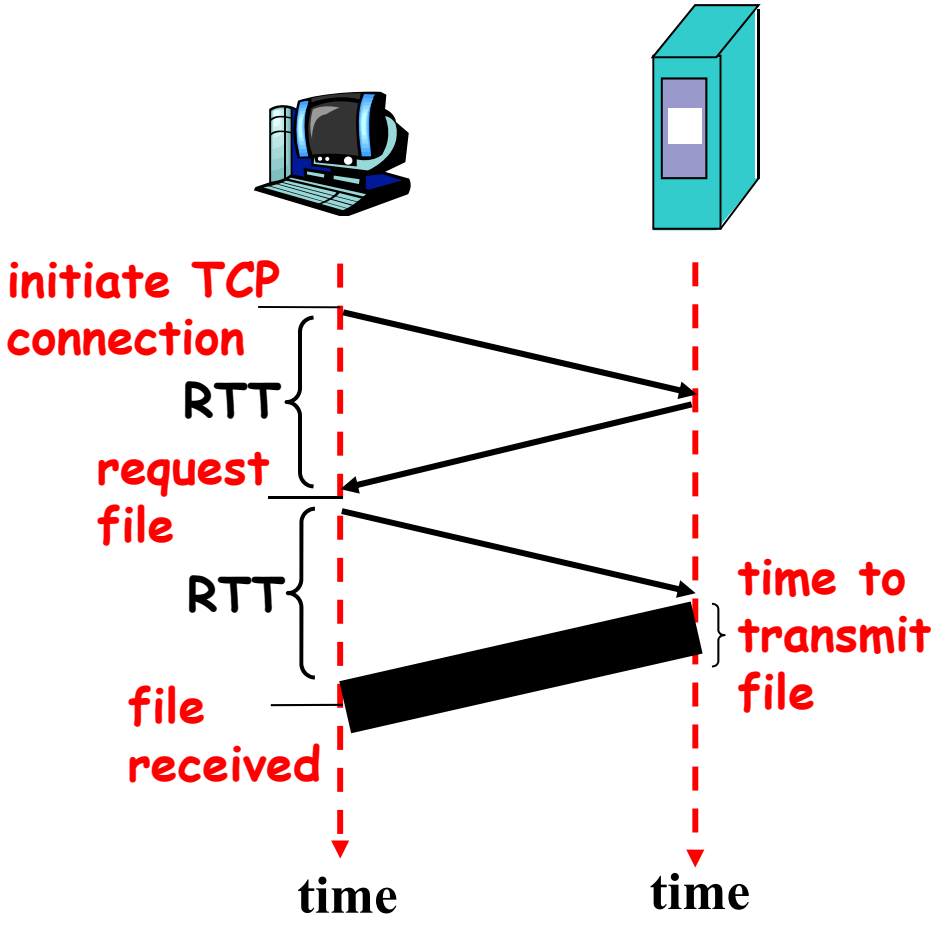
Non-Persistent HTTP: Response time

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

Response time:

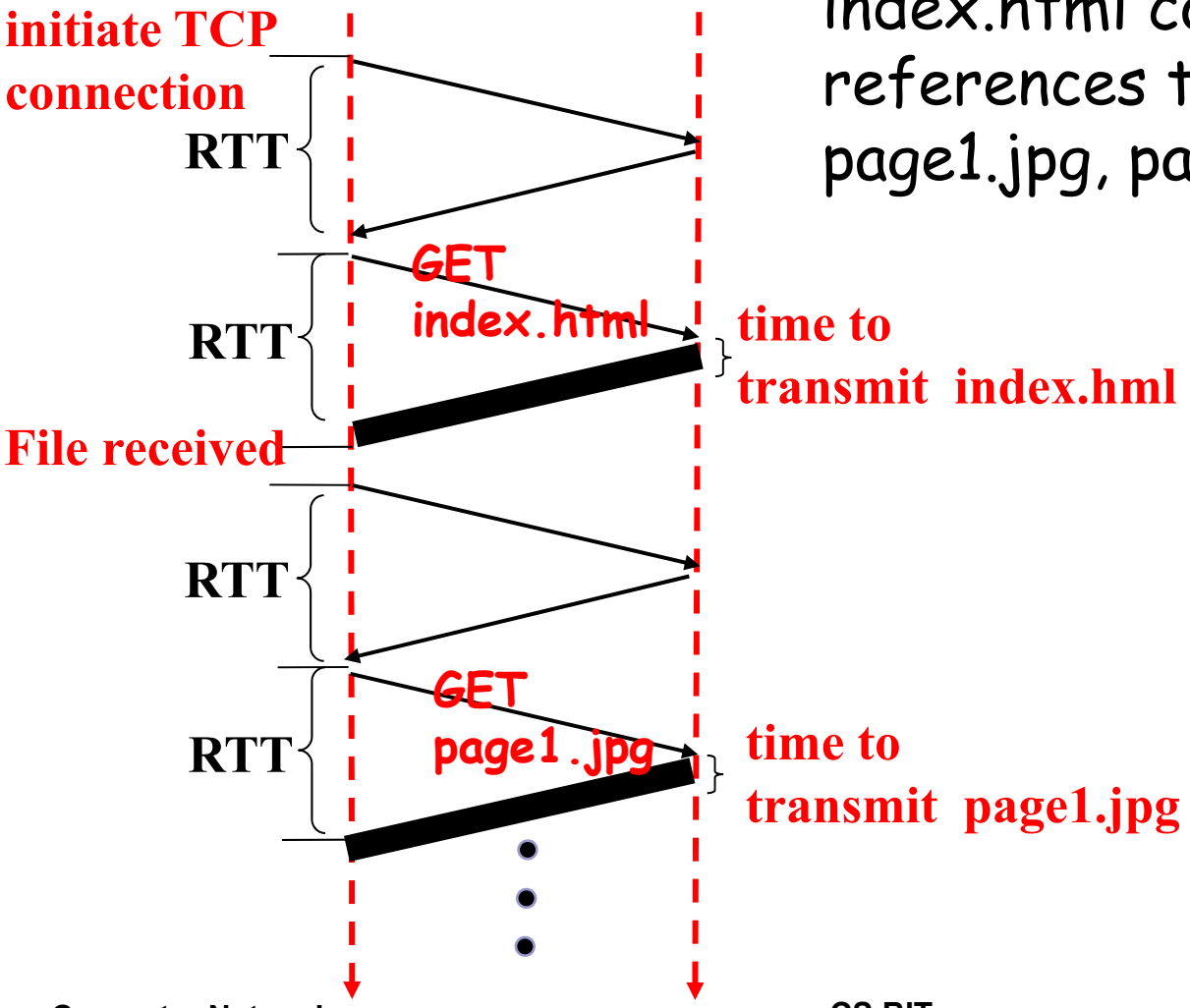
- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time

total = 2 RTT+transmit time



Classical HTTP/1.0

www.someSchool.edu/someDeptt/index.html



index.html contains text, references to 10 jpeg images: page1.jpg, page2.jpg, page3.jpg …

HTTP/1.0 Delay

- For each object:
  - TCP handshake --- 1 RTT
  - client request and server responds --- at least 1 RTT (if object can be contained in one packet)
- Discussion: how to reduce delay?
  - Parallel TCP connections
  - Persistent HTTP
  - Cache and conditional GET

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

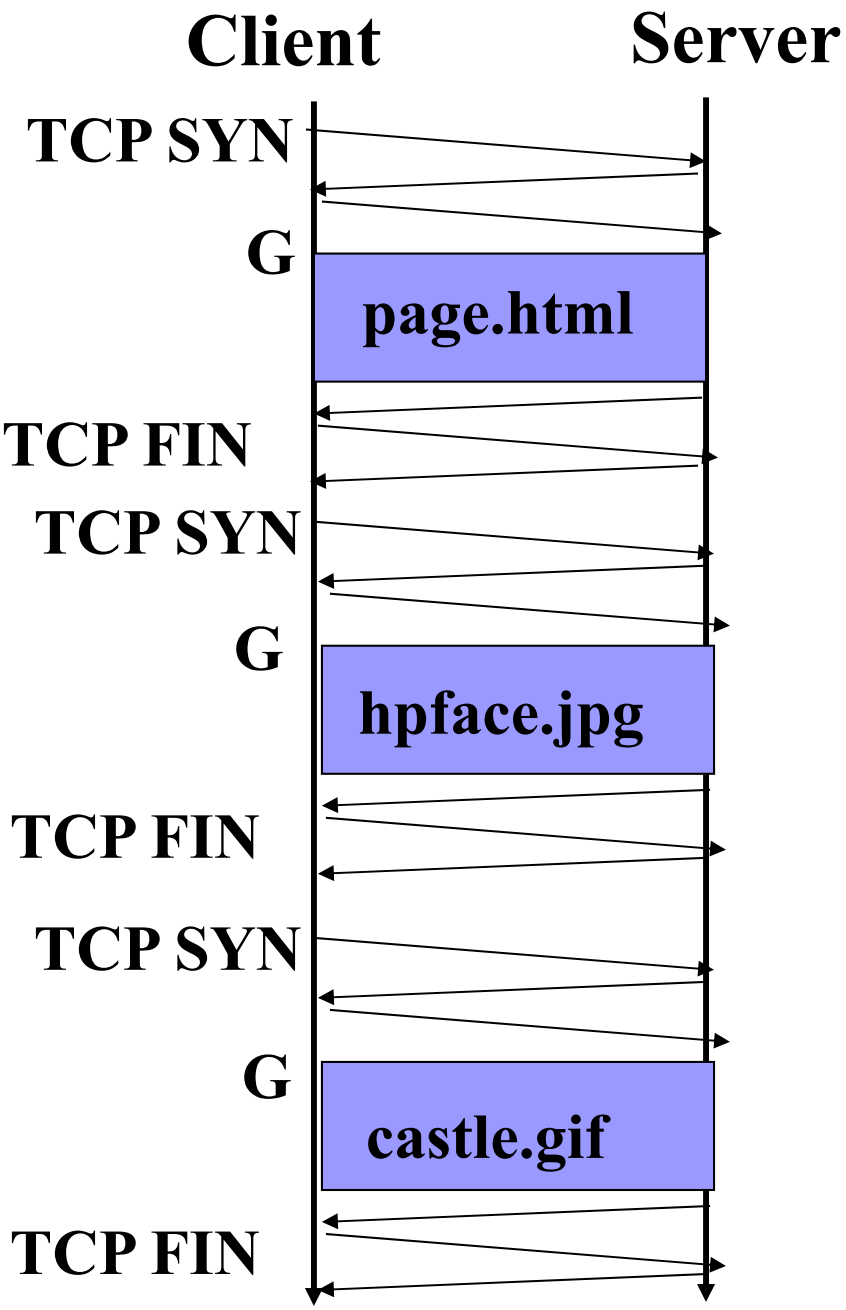
- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection

Persistent *without* pipelining:

- client issues new request only when previous response has been received
- one RTT for each referenced object

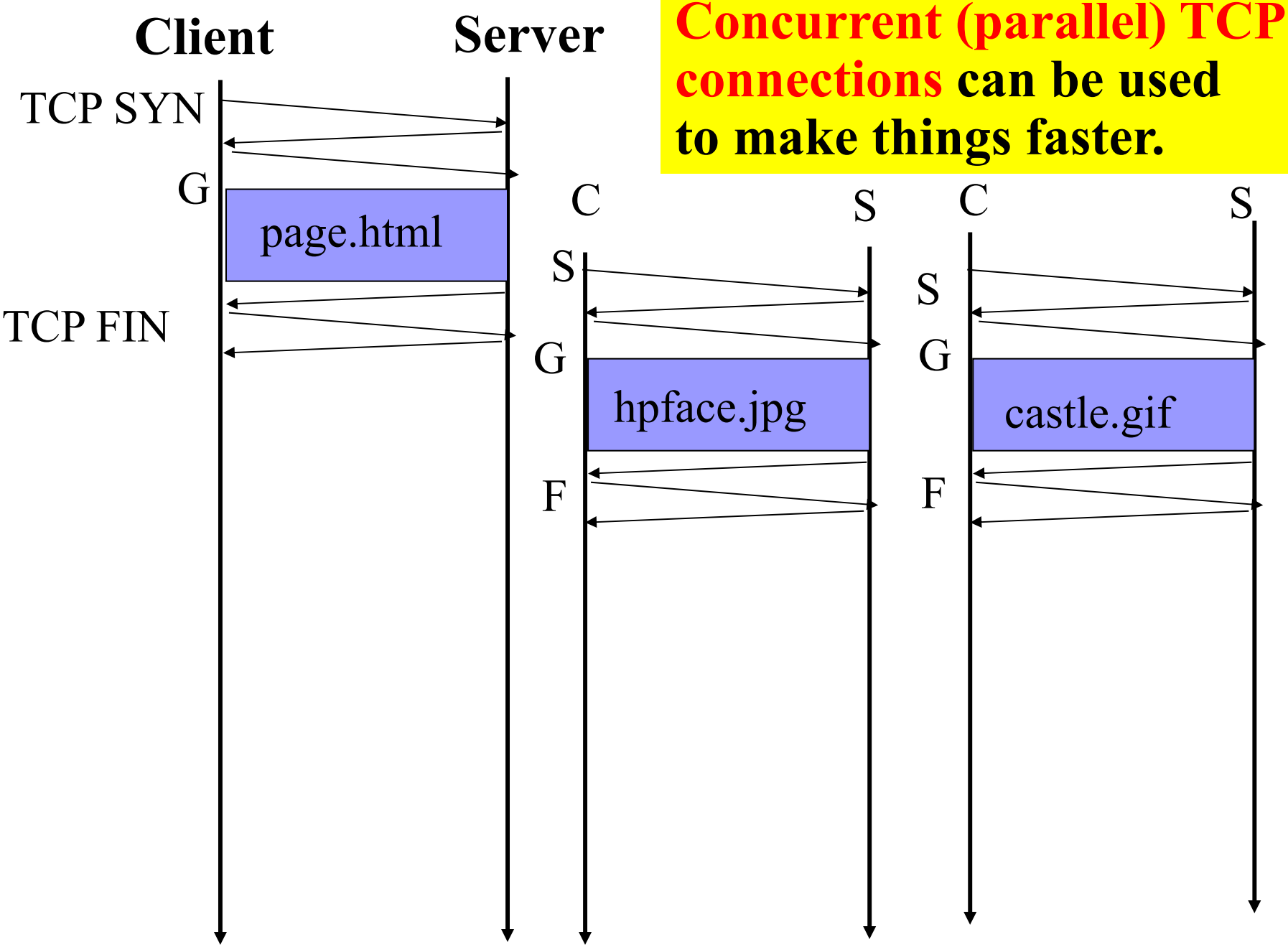
Persistent *with* pipelining:

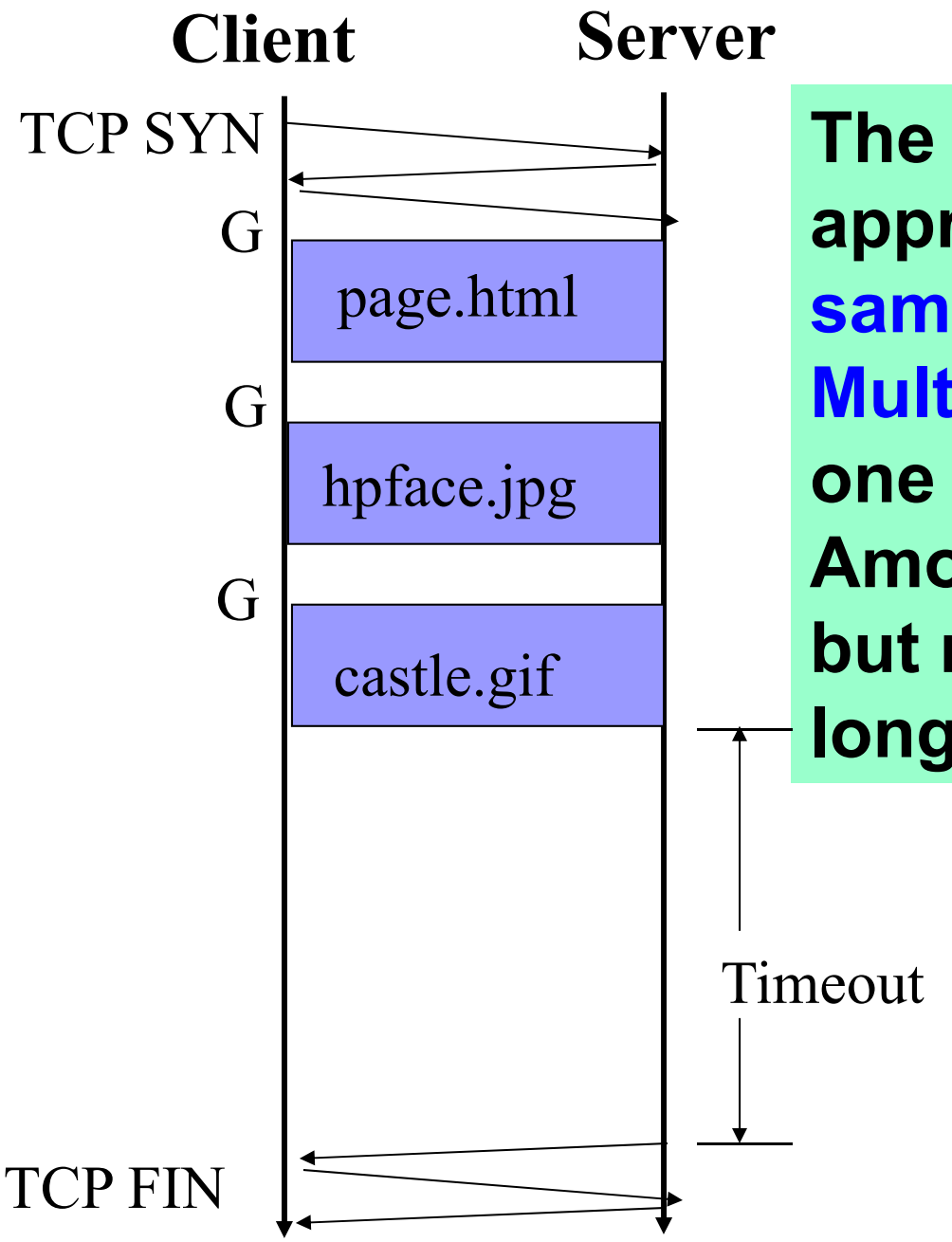
- default in HTTP/1.1
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects



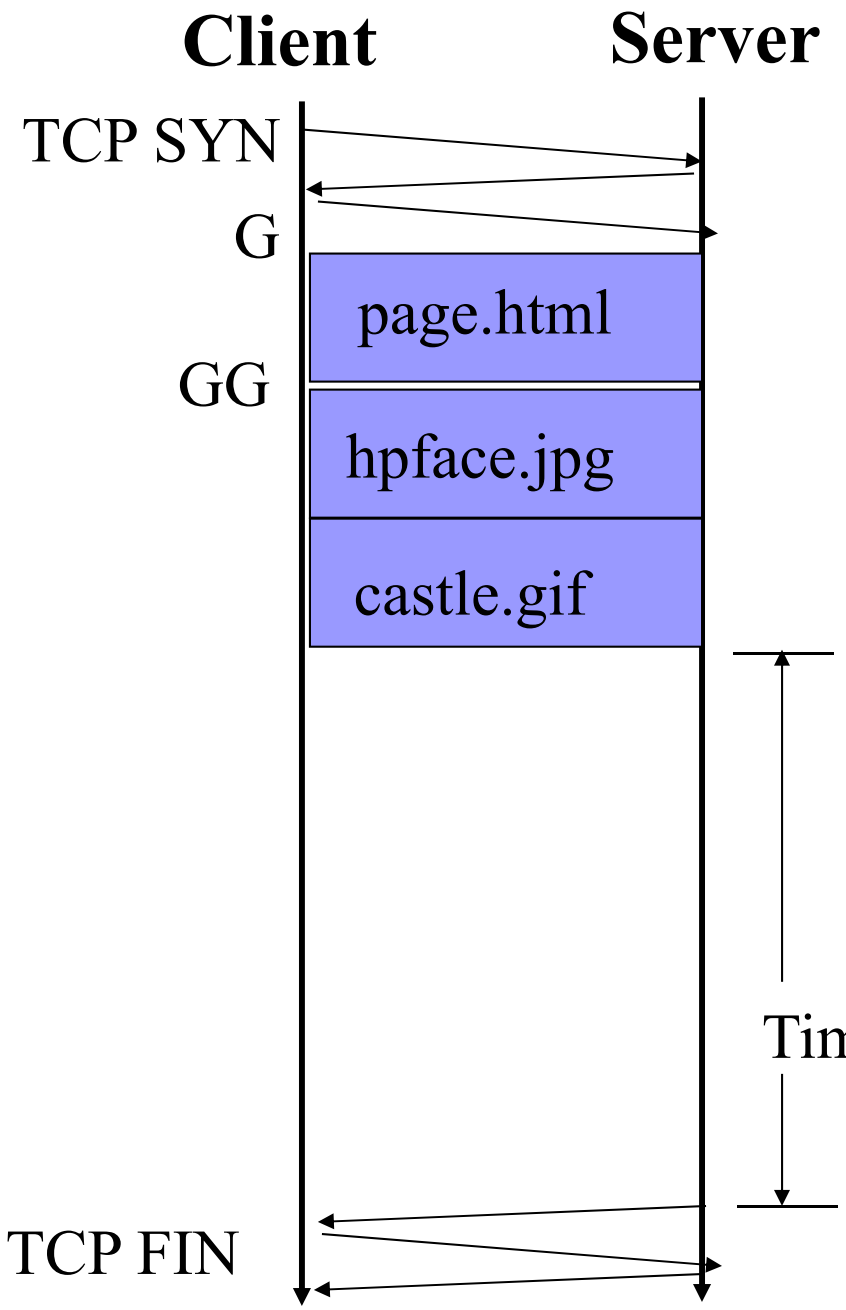
The "classic" approach in HTTP/1.0 is to use one HTTP request per TCP connection, serially.

Concurrent (parallel) TCP connections can be used to make things faster.





The "persistent HTTP" approach can re-use the same TCP connection for Multiple HTTP transfers, one after another, serially. Amortizes TCP overhead, but maintains TCP state longer at server.



The "pipelining" feature in HTTP/1.1 allows requests to be issued asynchronously on a persistent connection. Requests must be processed in proper order.
Can do clever packaging.

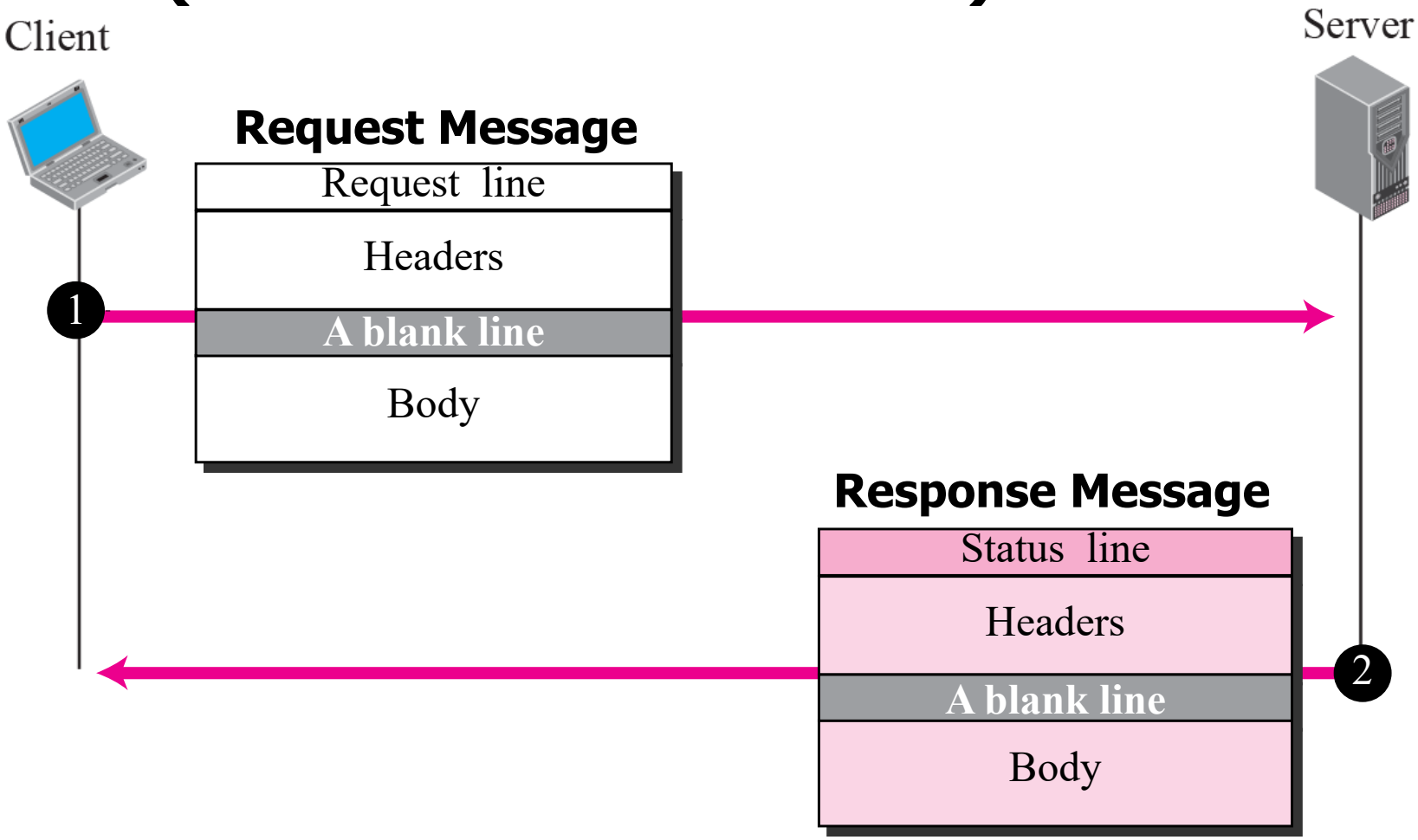
HTTP Message Flow: Persistent HTTP

- **Default for HTTP/1.1**
- **On same TCP connection:** server parses request, responds, parses new request, …
- Client sends requests for all referenced objects as soon as it receives base HTML
- Fewer RTTs

HTTP transaction

Two types of HTTP messages: *request, response*

ASCII (human-readable format)



HTTP request message

request line (GET, POST, HEAD commands)

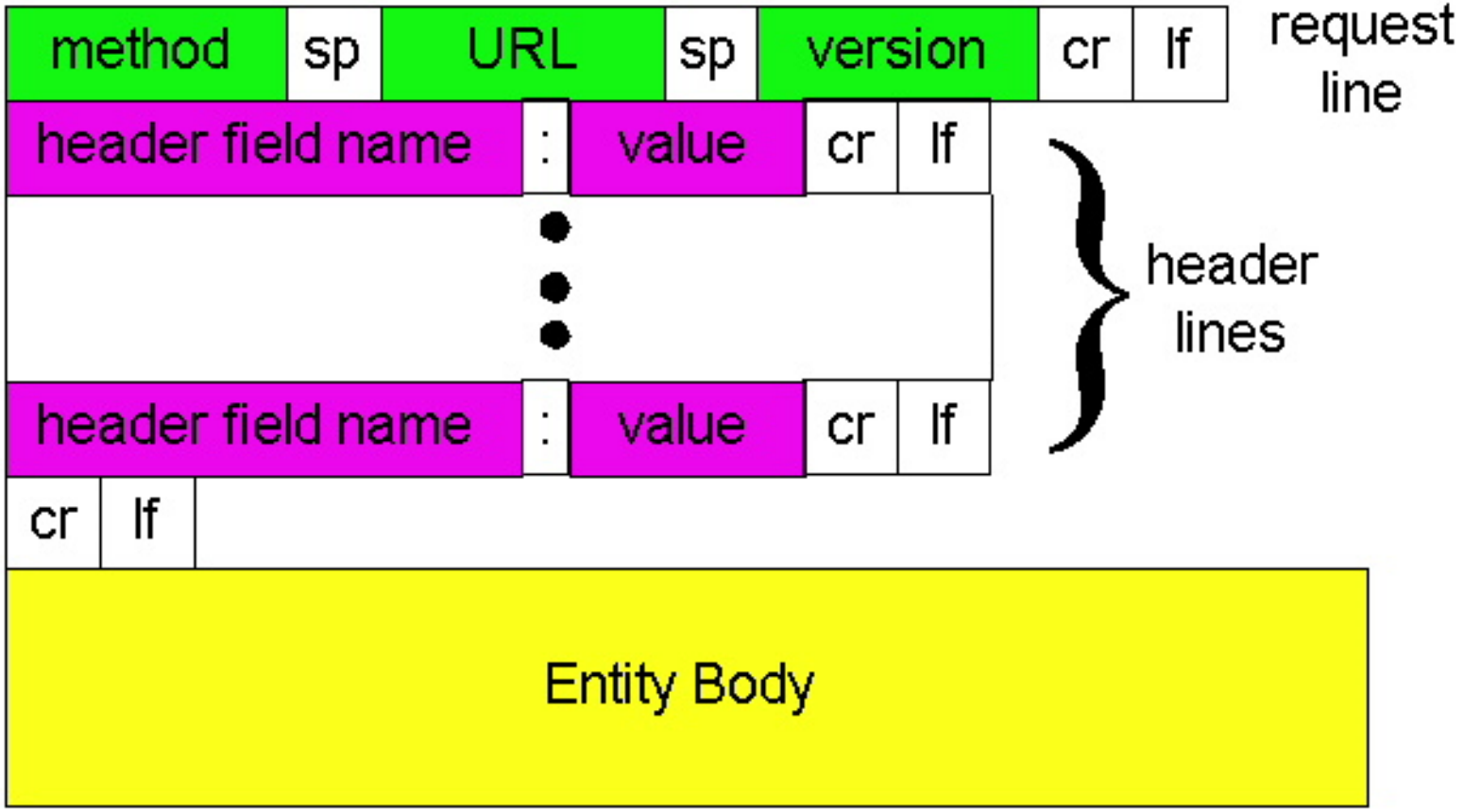
header lines

Carriage return, line feed indicates end of message

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

(extra carriage return, line feed)

HTTP request message: general format



HTTP Methods

- GET: retrieve a file (95% of requests)
- HEAD: just get meta-data (e.g., mod time)
- POST: submitting a form to a server
- PUT: store enclosed document as URI
- DELETE: removed named resource
- LINK/UNLINK: in 1.0, gone in 1.1
- TRACE: http “echo” for debugging (added in 1.1)
- CONNECT: used by proxies for tunneling (1.1)
- OPTIONS: request for server/proxy options (1.1)

Method types

HTTP/1.0

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

HTTP/1.1

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

Uploading form input

Post method:

- Web page often includes form input
- Input is uploaded to server in entity body

URL method:

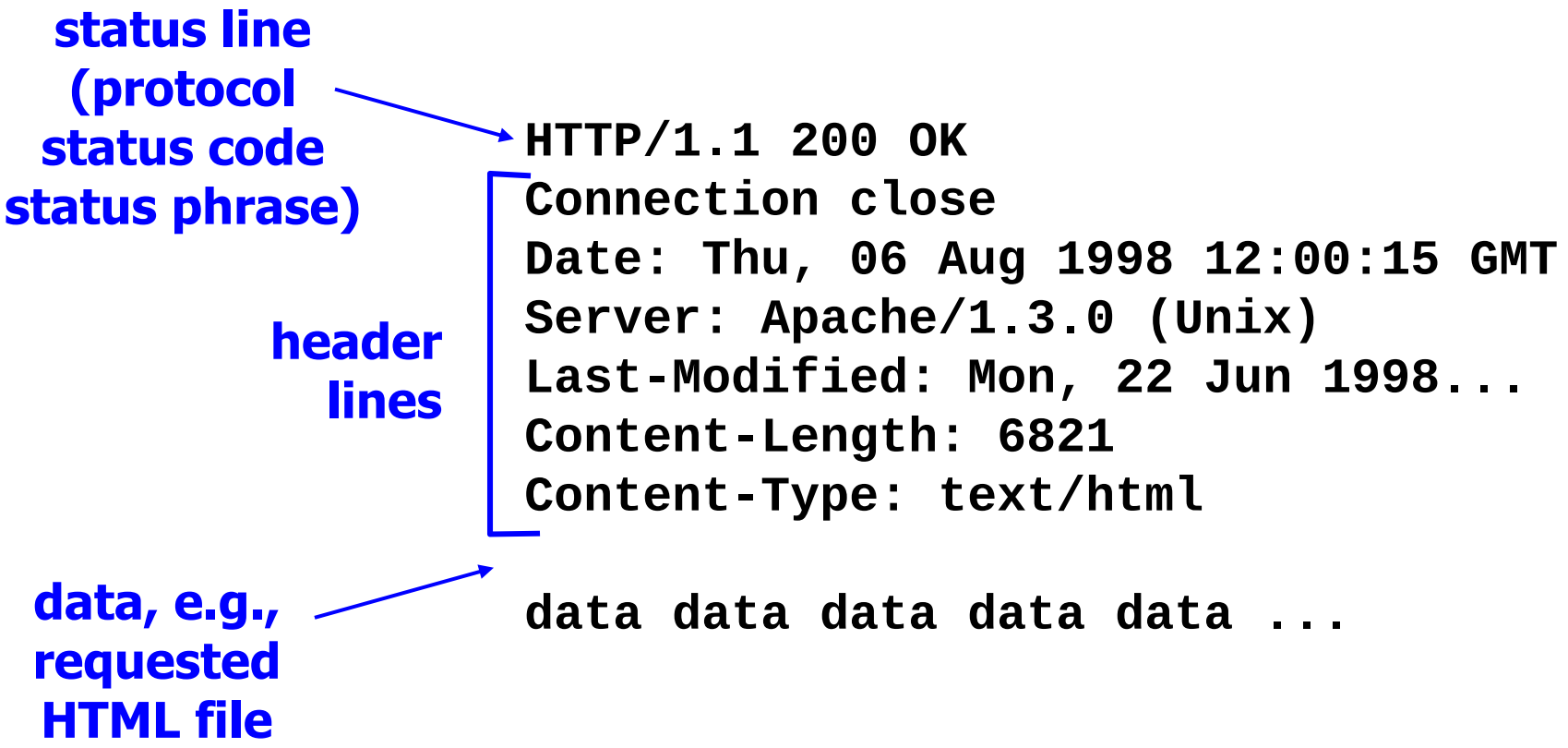
- Uses GET method
- Input is uploaded in URL field of request line

www.somesite.com/animalsearch?monkeys&banana

HTTP request headers

Header	*Description*
User-agent	Identifies the client program
Accept	Shows the media 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
Host	Shows the host and port number of the client
Date	Shows the current date
Upgrade	Specifies the preferred communication protocol
Cookie	Returns the cookie to the server
If-Modified-Since	Returns the cookie to the server

HTTP response message



HTTP response status codes

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

HTTP response headers

Header	*Description*
Date	Shows the current date
Upgrade	Specifies the preferred communication protocol
Server	Gives information about the server
Set-Cookie	The server asks the client to save a cookie
Content-Encoding	Specifies the encoding scheme
Content-Language	Specifies the language
Content-Length	Shows the length of the document
Content-Type	Specifies the media type
Location	To ask the client to send the request to another site
Accept-Ranges	The server will accept the requested byte-ranges
Last-modified	Gives the date and time of the last change

Different Forms of Server Response

- **Return a file**
  - URL matches a file (*e.g.,* /www/index.html)
  - Server returns file as the response
  - Server generates appropriate response header
- **Generate response dynamically**
  - URL triggers a program on the server
  - Server runs program and sends output to client
- **Return meta-data with no body**

User-Server Interaction: Cookies

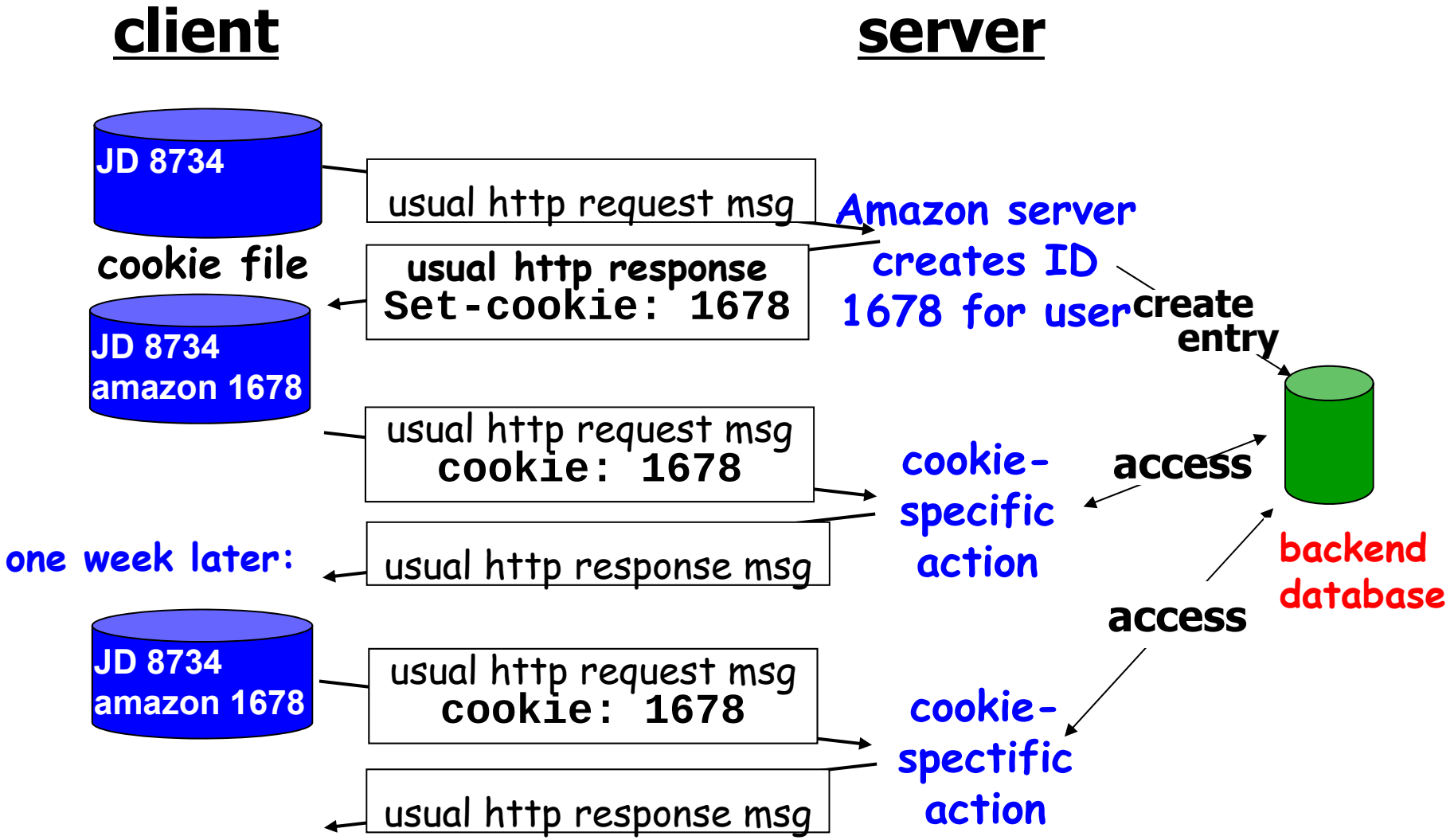
Many major Web sites use cookies

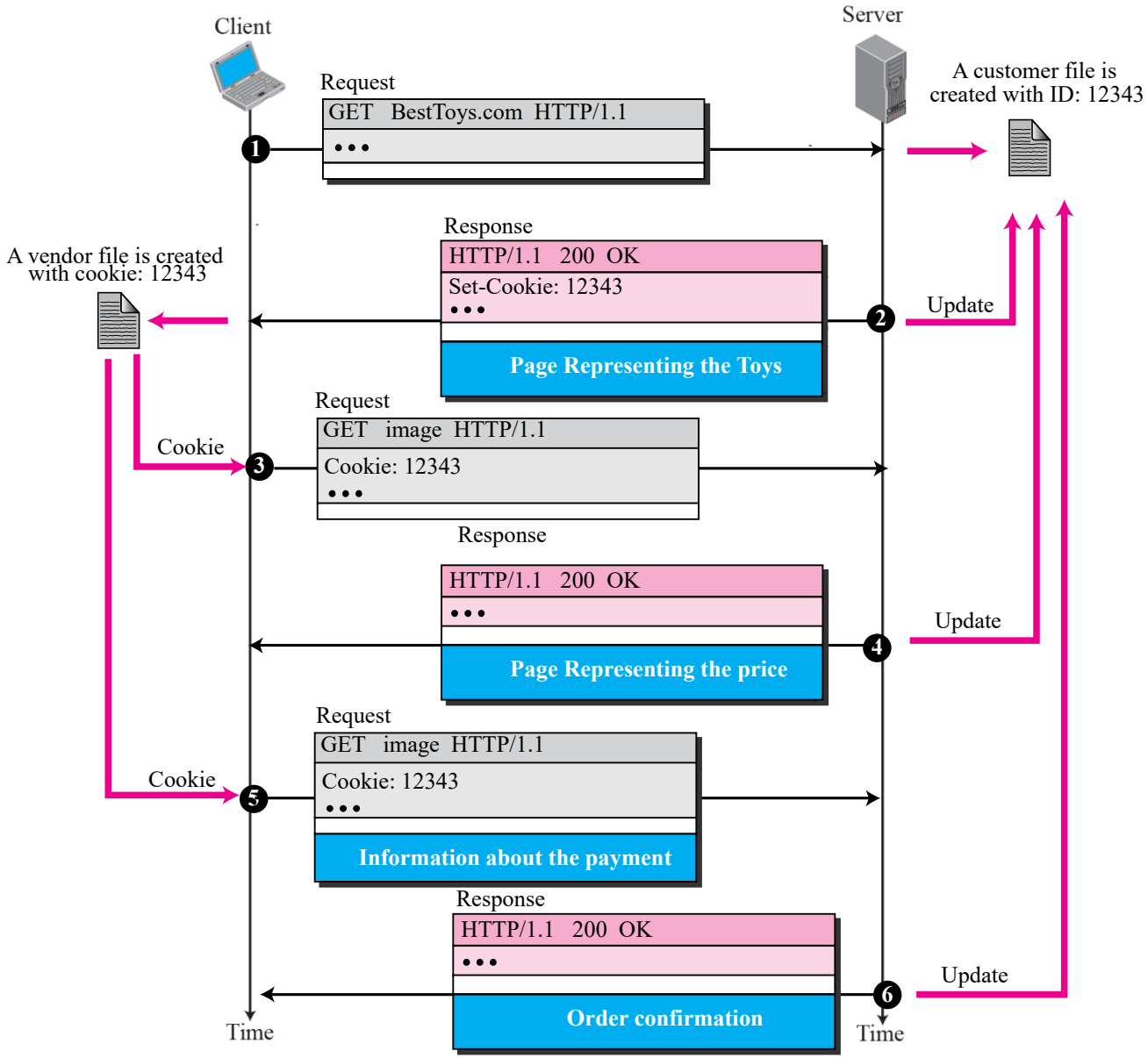
keeping user-server "state"

Four components:

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

User-Server Interaction: Cookies




Client
Server
Request
GET BestToys.com HTTP/1.1
A customer file is created with ID: 12343
Response
HTTP/1.1 200 OK
Set-Cookie: 12343
A vendor file is created with cookie: 12343
Update
Page Representing the Toys
Request
GET image HTTP/1.1
Cookie: 12343
Cookie
Response
HTTP/1.1 200 OK
Update
Page Representing the price
Request
GET image HTTP/1.1
Cookie: 12343
Cookie
Information about the payment
Response
HTTP/1.1 200 OK
Update
Order confirmation
Time
Time

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

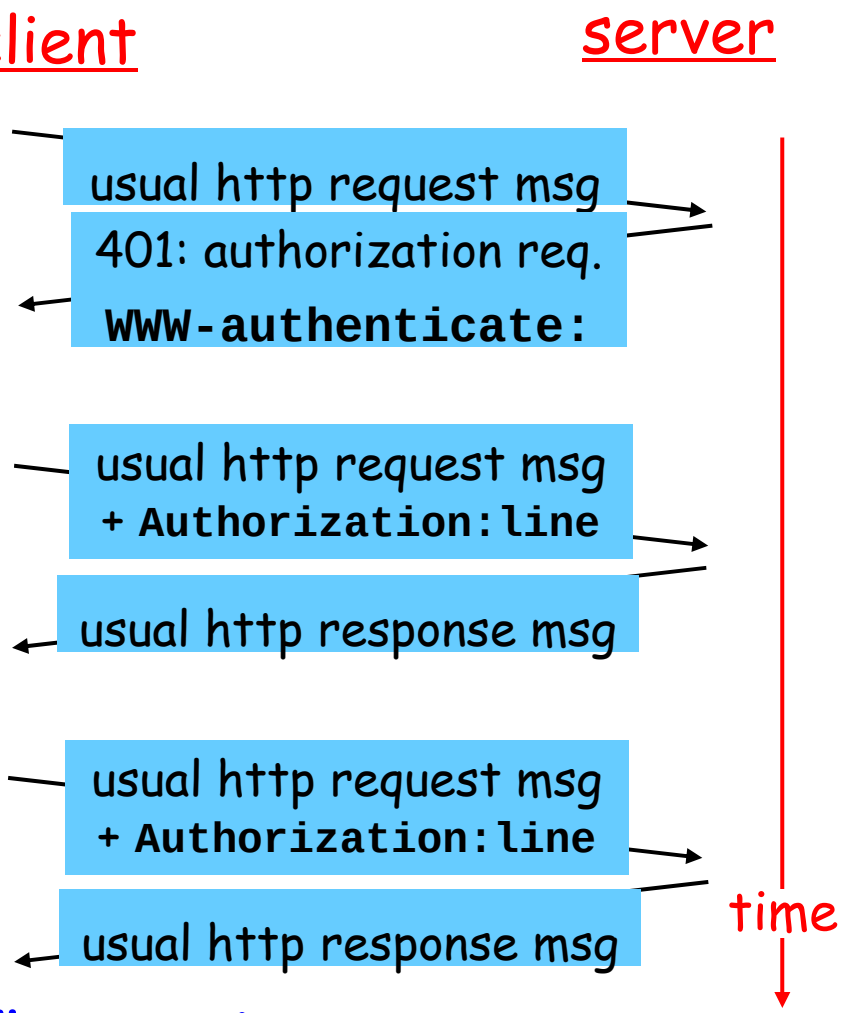
User-Server Interaction: Authentication

Authentication goal: control access to server documents

- **stateless:** client must present authorization in each request
- **authorization:** typically name, password
  - `Authorization:` header line in request
  - if no authorization presented, server refuses access, sends

    `WWW-authenticate:`

    header line in response

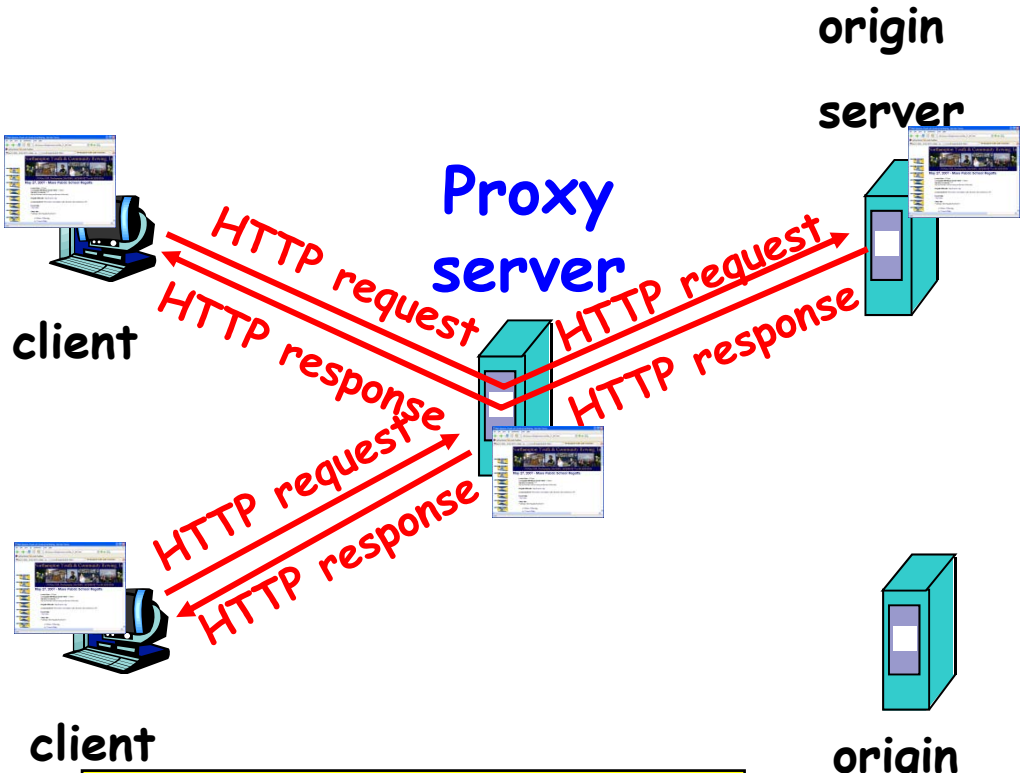


Browser caches name & password so that user does not have to repeatedly enter it.

Web caches (proxy server)

Goal: satisfy client request without involving origin server

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



Cache acts as both client and server

More about Web caching

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

Why Web caching?

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

Caching example

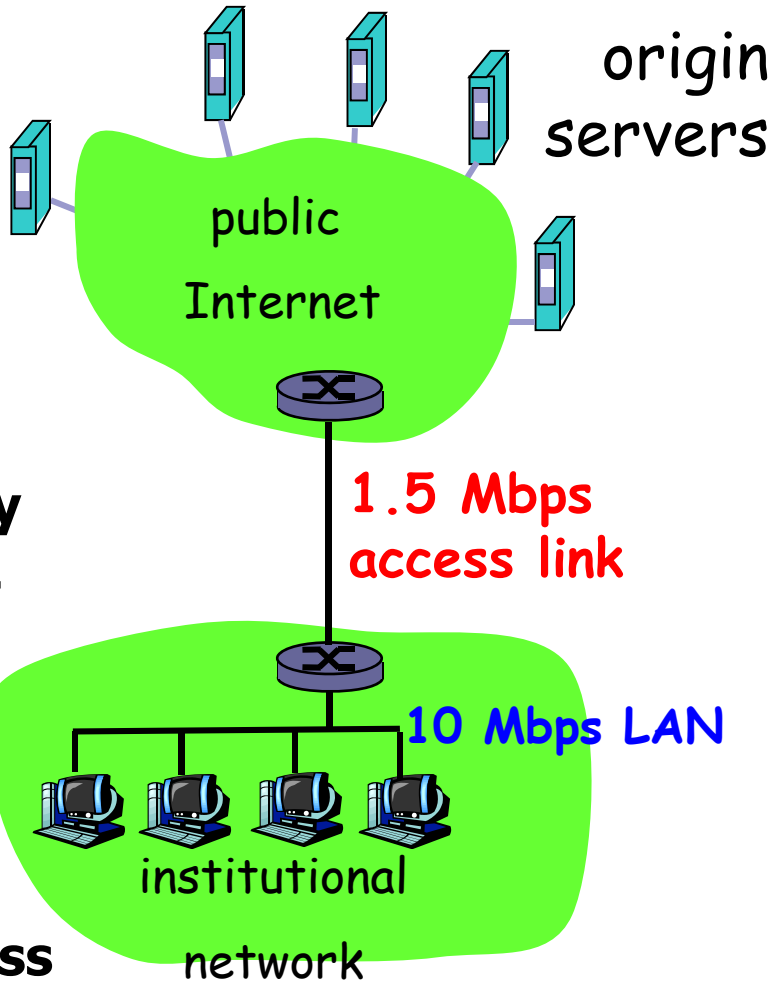
Assumptions

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

Consequences

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

= 2 sec + minutes + milliseconds



Caching example

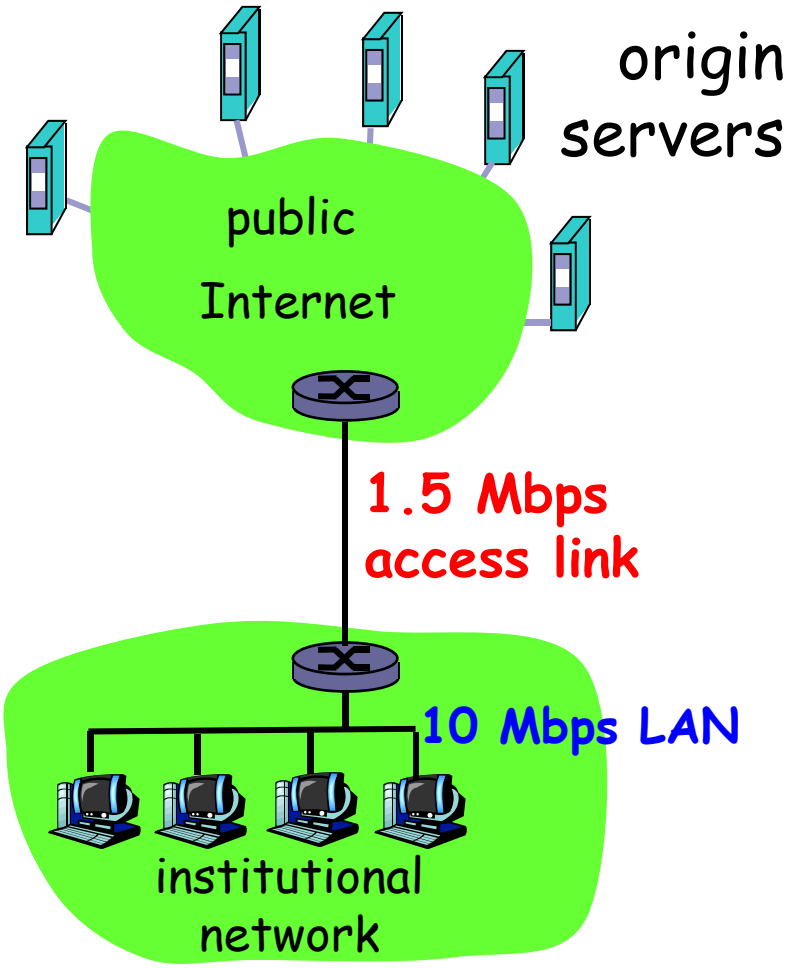
Assumptions

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

Consequences

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

  = 2 sec + minutes + milliseconds



Caching example (cont)

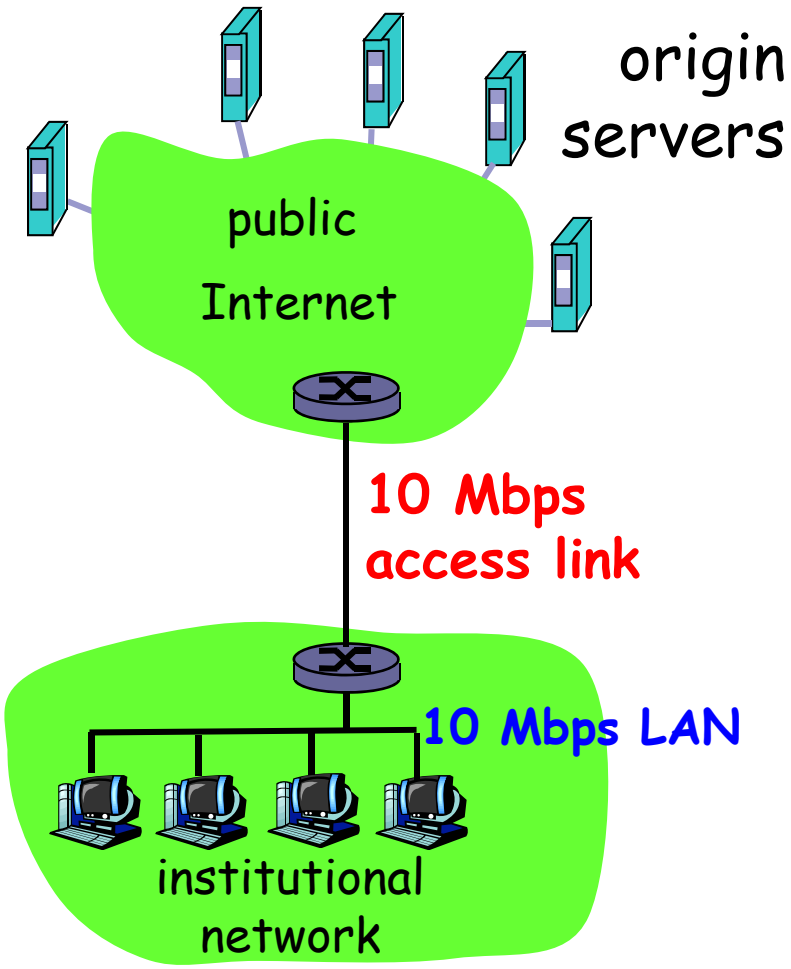
Possible solution

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

Consequence

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

  = 2 sec + msecs + msecs
- often a costly upgrade



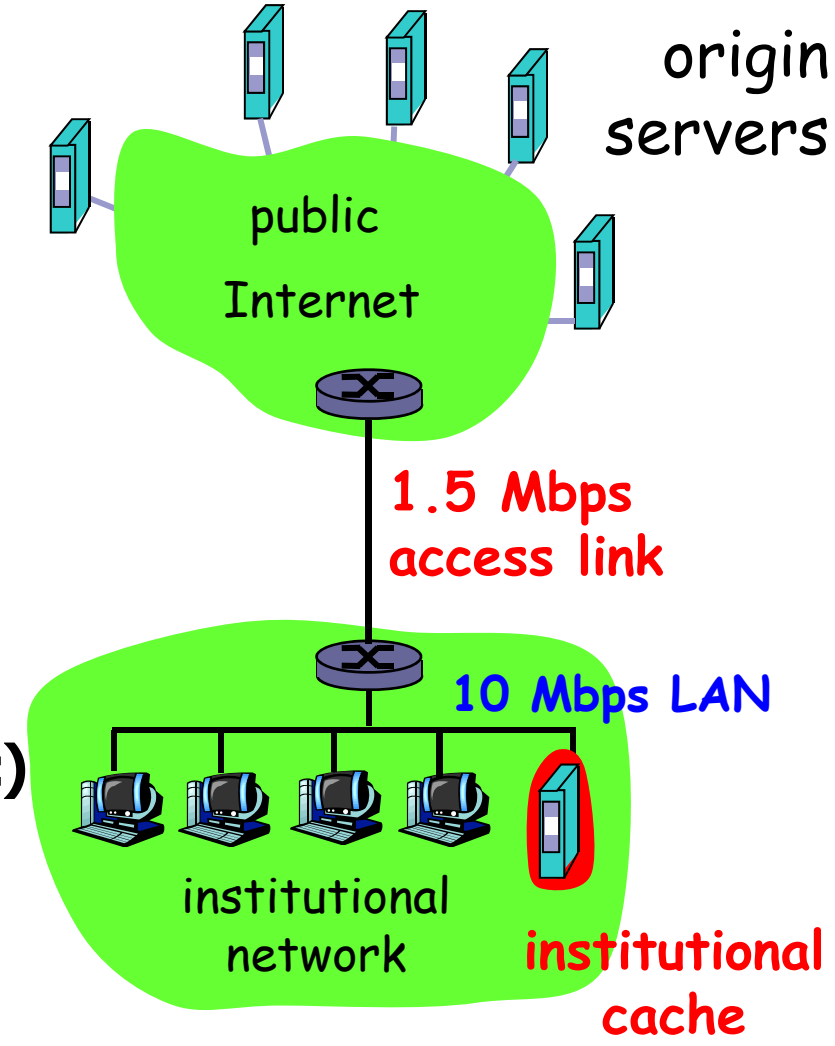
Caching example (cont)

Possible solution: install cache

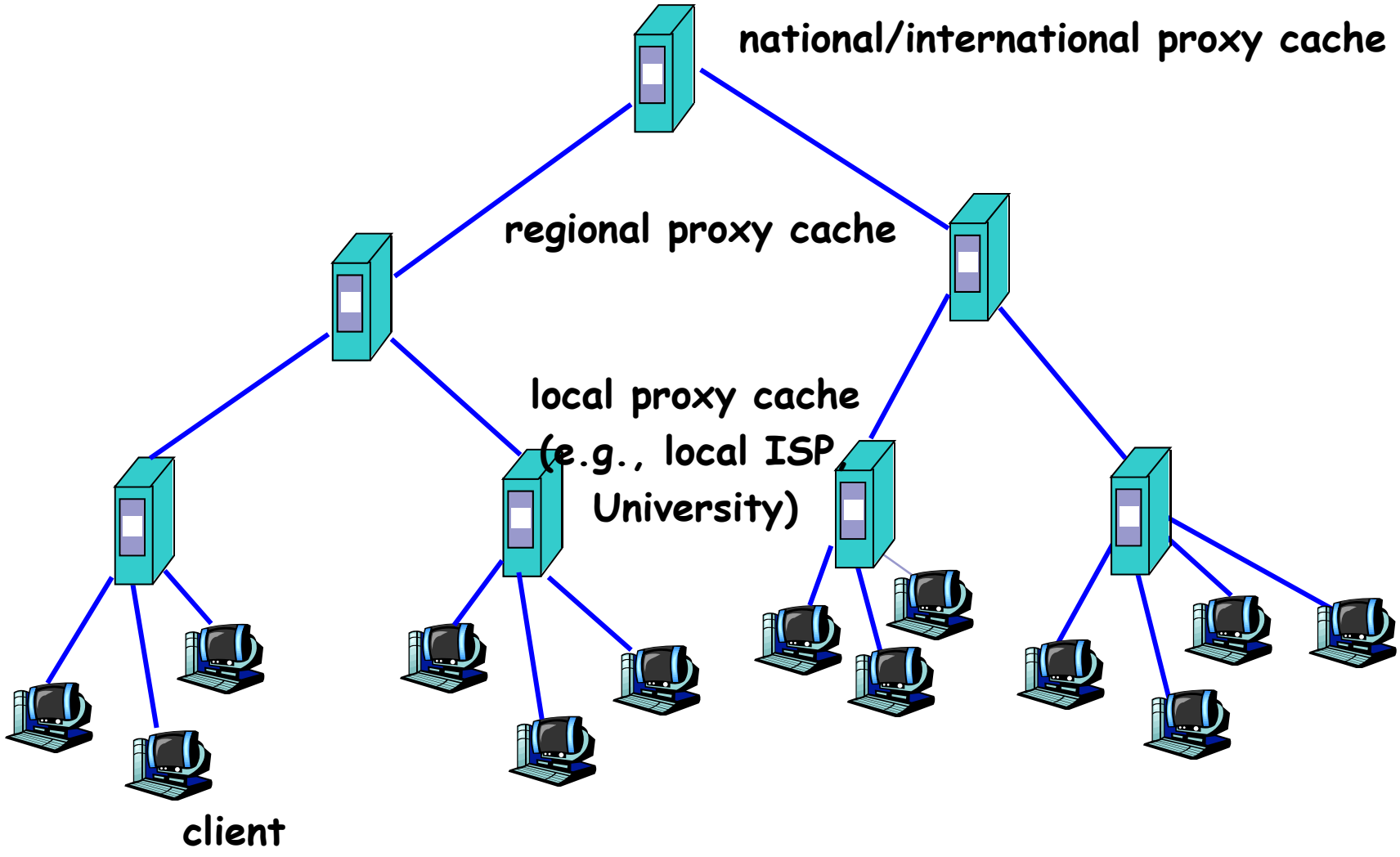
- suppose hit rate is 0.4

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 = 0.6*(2) secs + 0.4*milliseconds < 1.4 secs



Web Caching Hierarchy



Some Issues

- **Not all objects can be cached**
  - E.g., dynamic objects, copyrighted material
- **Cache consistency**
  - strong
  - weak
- **Cache Replacement Policies**
  - Variable size objects
  - Varying cost of not finding an object (a "miss") in the cache
- **Prefetch?**
  - A large fraction of the requests are one-timers

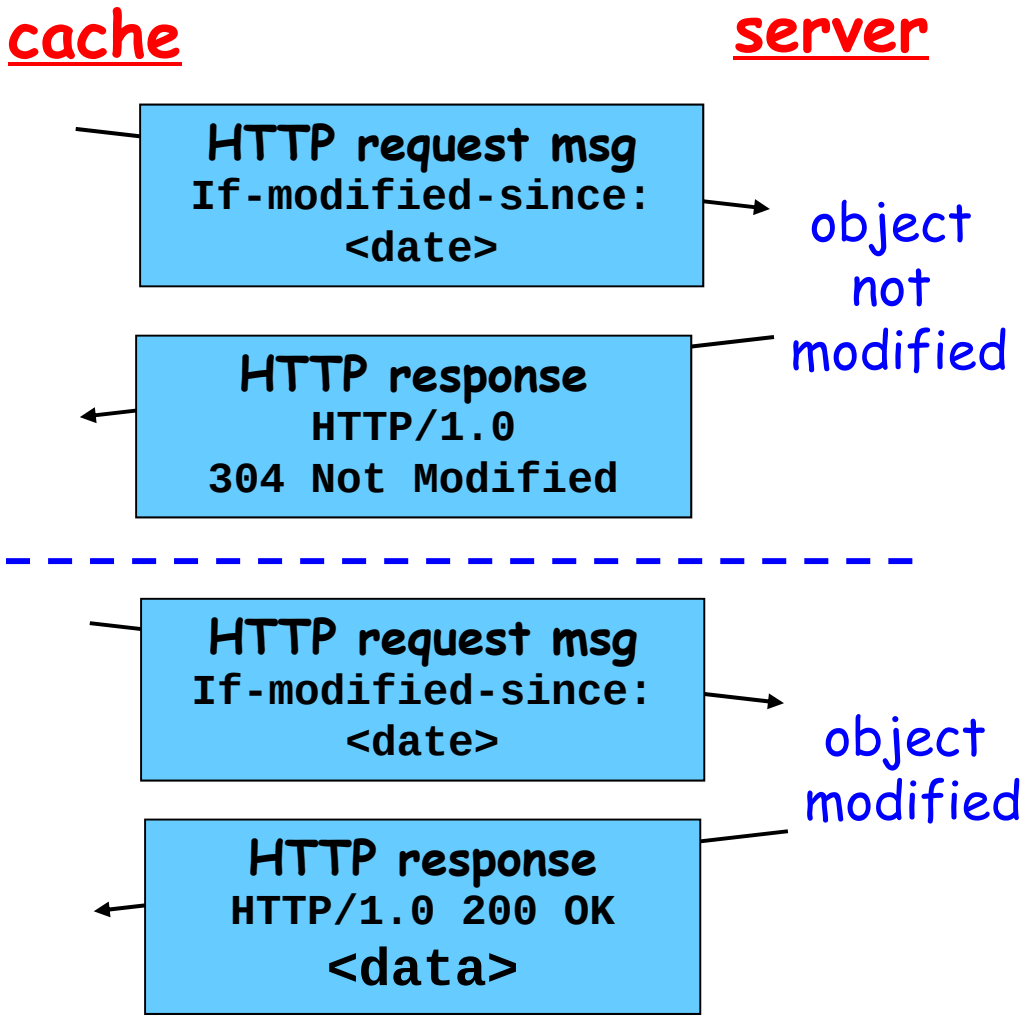
Conditional GET: client-side caching

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

  `If-modified-since: <date>`

- **server:** response contains no object if cached copy is up-to-date:

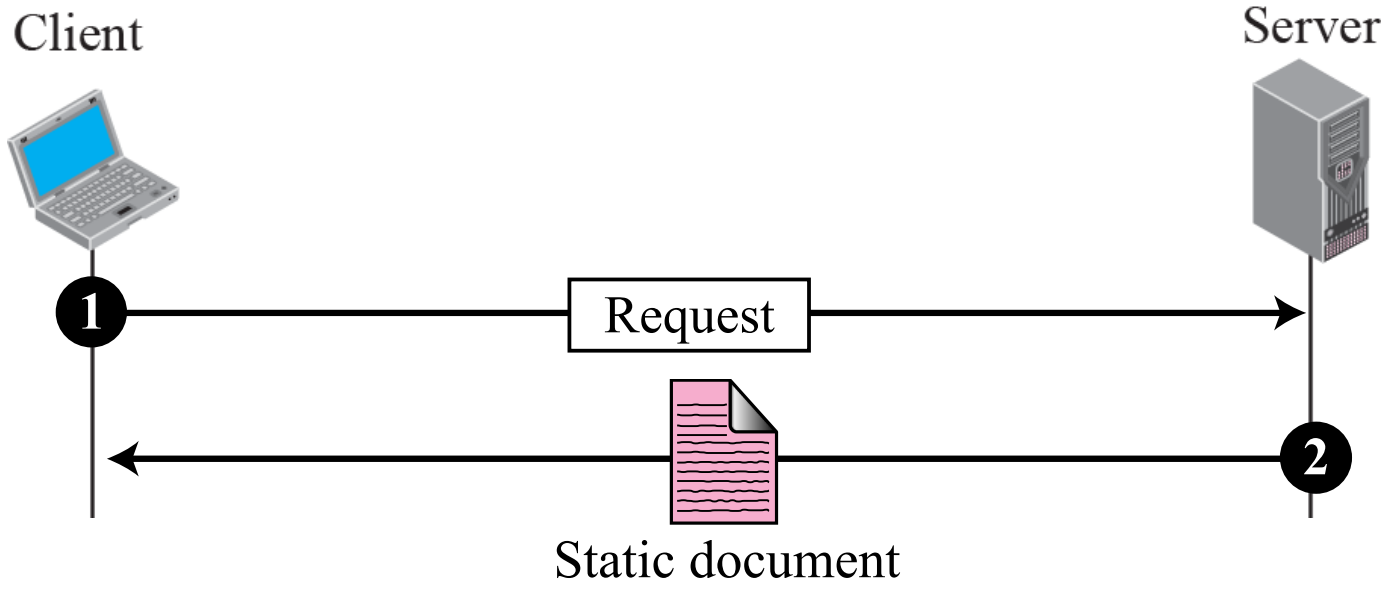
  `HTTP/1.0 304 Not Modified`



Web Documents

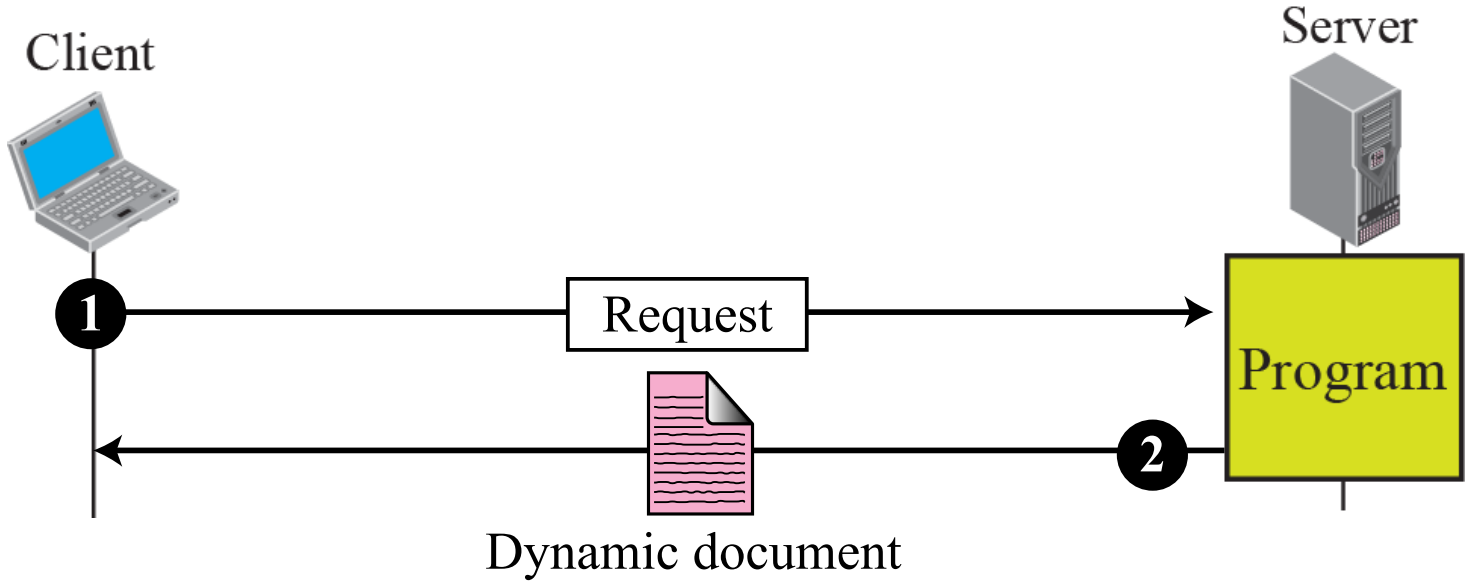
- The documents in the WWW can be grouped into three broad categories:
  - static
  - dynamic
  - active
- The category is based on the time the contents of the document are determined.

Static document



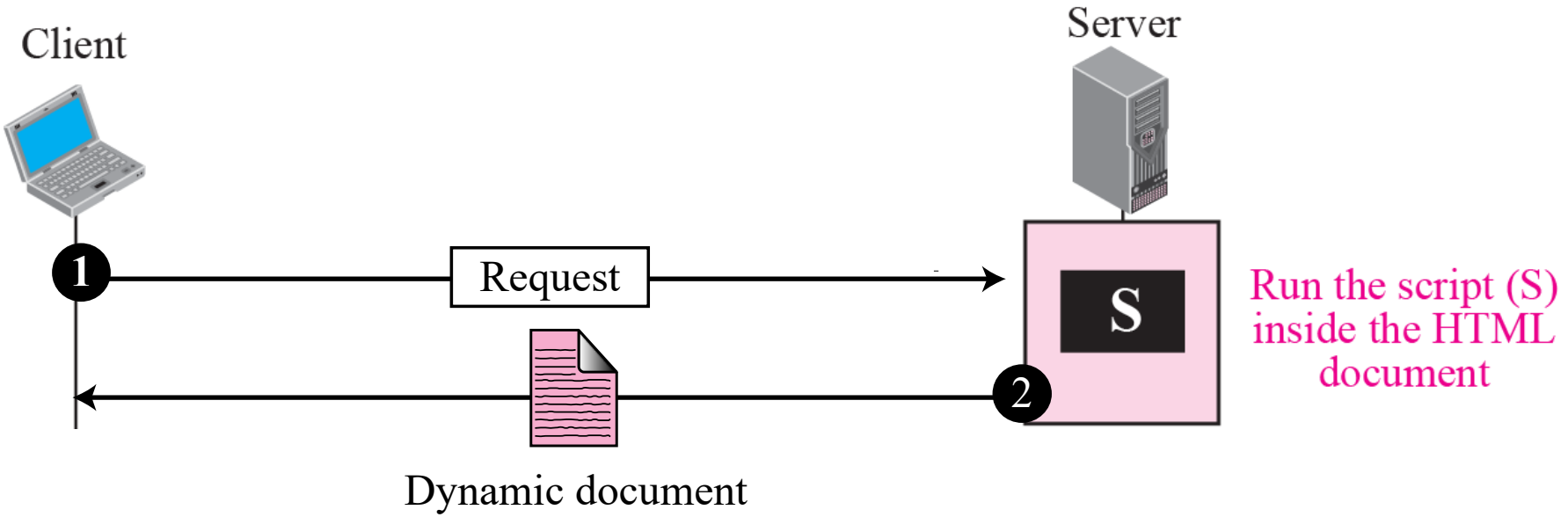
Static document

Dynamic document



Dynamic document using CGI

Dynamic document



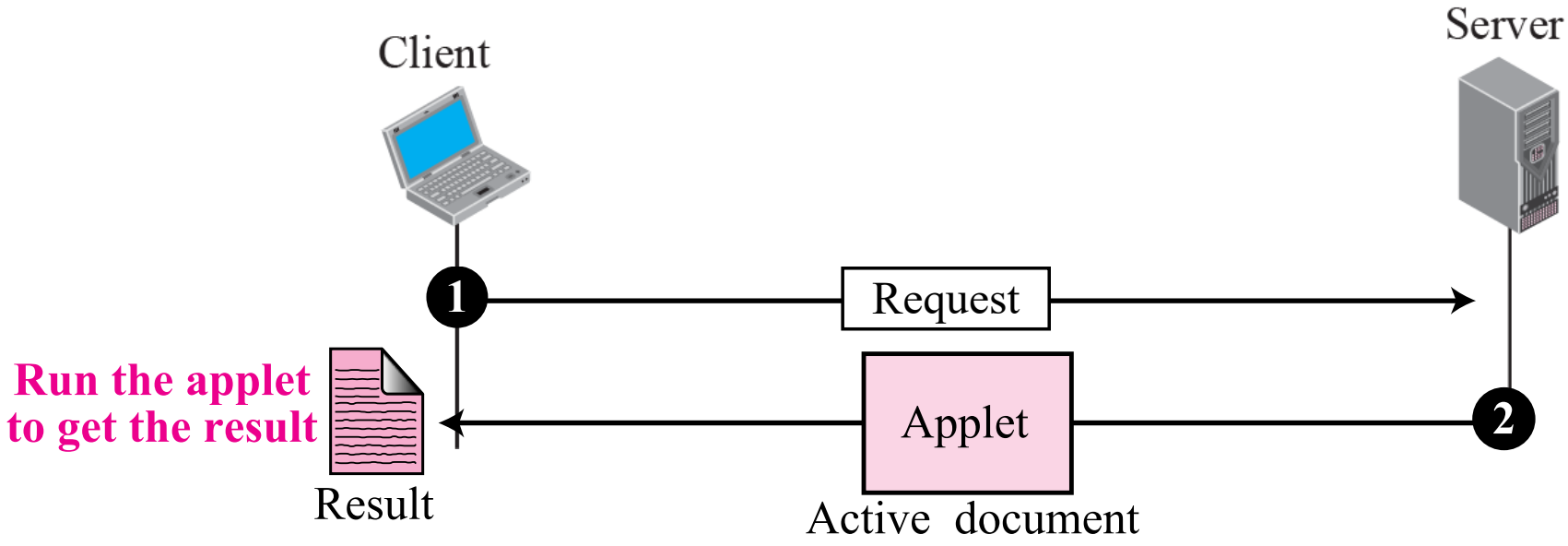
Dynamic document using server-site script

Dynamic document

Note

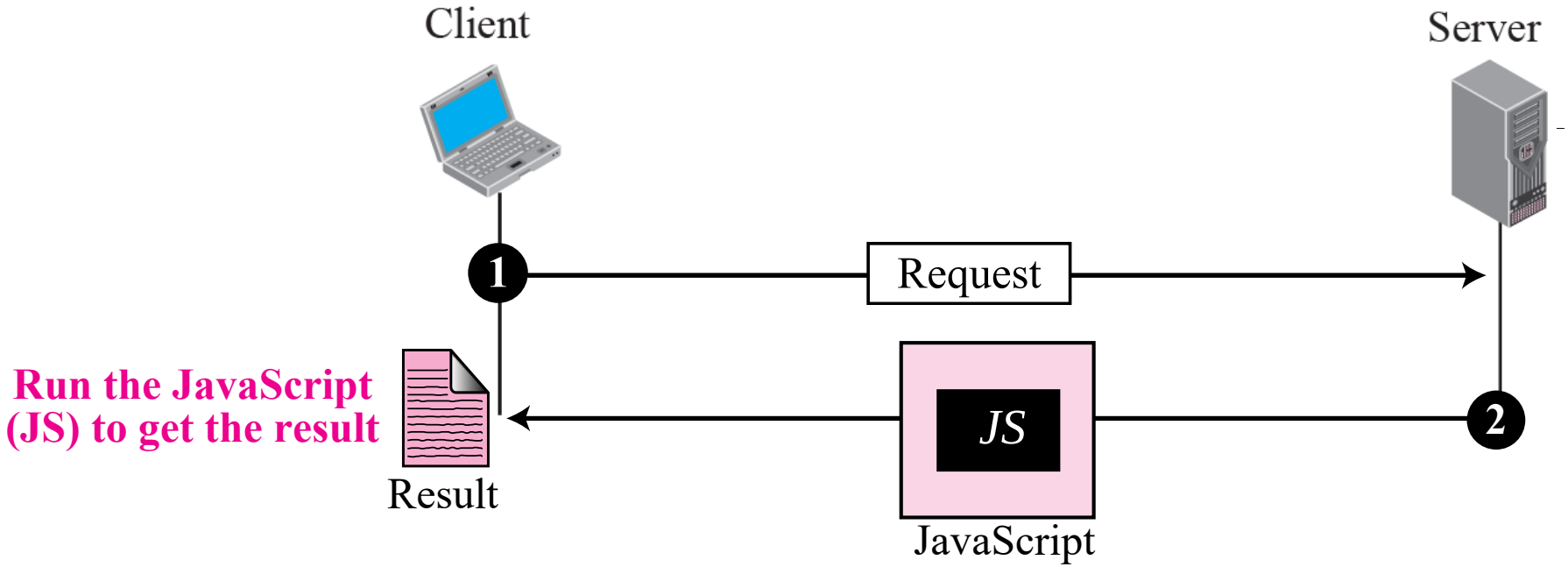
Dynamic documents are sometimes referred to as server-site dynamic documents.

Active document



Active document using Java applet

Active document



Active document using client-site script

Active document

Note

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

Web Server

- **Web site vs. Web server**
  - **Web site:** collections of Web pages associated with a particular host name
  - **Web server:** program that satisfies client requests for Web resources
- **Handling a client request**
  - Accept the TCP connection
  - Read and parse the HTTP request message
  - Translate the URL to a filename
  - Determine whether the request is authorized
  - Generate and transmit the response

Web Server: Generating a Response

- **Returning a file**
  - URL corresponds to a file (e.g., /www/index.html)
  - … and the server returns the file as the response
  - … along with the HTTP response header
- **Returning meta-data with no body**
  - Example: client requests object “if-modified-since”
  - Server checks if the object has been modified
  - … and simply returns a “HTTP/1.1 304 Not Modified”
- **Dynamically-generated responses**
  - URL corresponds to a program the server needs to run
  - Server runs the program and sends the output to client

Hosting: Multiple Sites Per Machine

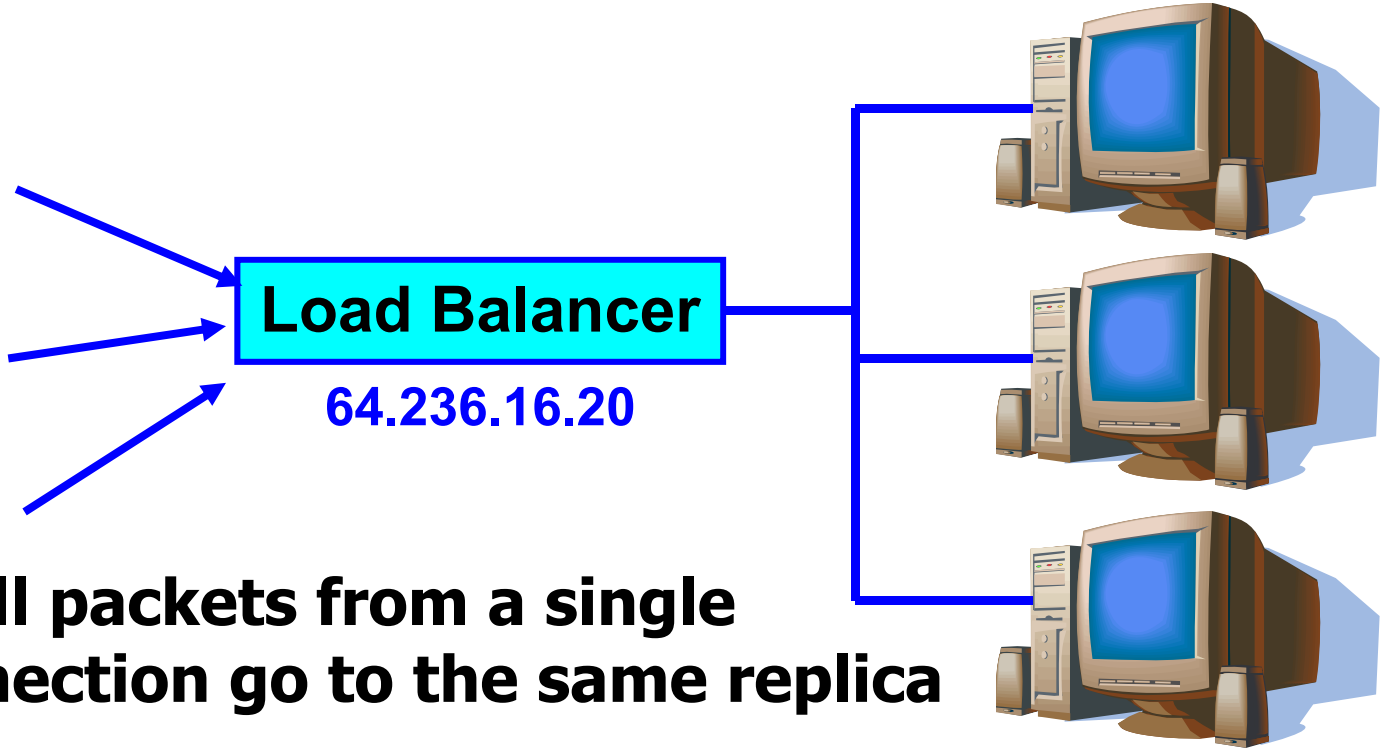
- **Multiple Web sites on a single machine**
  - Hosting company runs the Web server on behalf of multiple sites (e.g., www.foo.com and www.bar.com)
- **Problem: returning the correct content**
  - www.foo.com/index.html vs. www.bar.com/index.html
  - How to differentiate when both are on same machine?
- **Solution #1: multiple servers on the same machine**
  - Run multiple Web servers on the machine
  - Have a separate IP address for each server
- **Solution #2: include site name in the HTTP request**
  - Run a single Web server with a single IP address
  - … and include "Host" header (e.g., "Host: www.foo.com")

Hosting: Multiple Machines Per Site

- **Replicating a popular Web site**
  - Running on multiple machines to handle the load
  - … and to place content closer to the clients
- **Problem:** directing client to a particular replica
  - To balance load across the server replicas
  - To pair clients with nearby servers
- **Solution #1:** manual selection by clients
  - Each replica has its own site name
  - A Web page lists the replicas (e.g., by name, location)
  - … and asks clients to click on a hyperlink to pick

Hosting: Multiple Machines Per Site

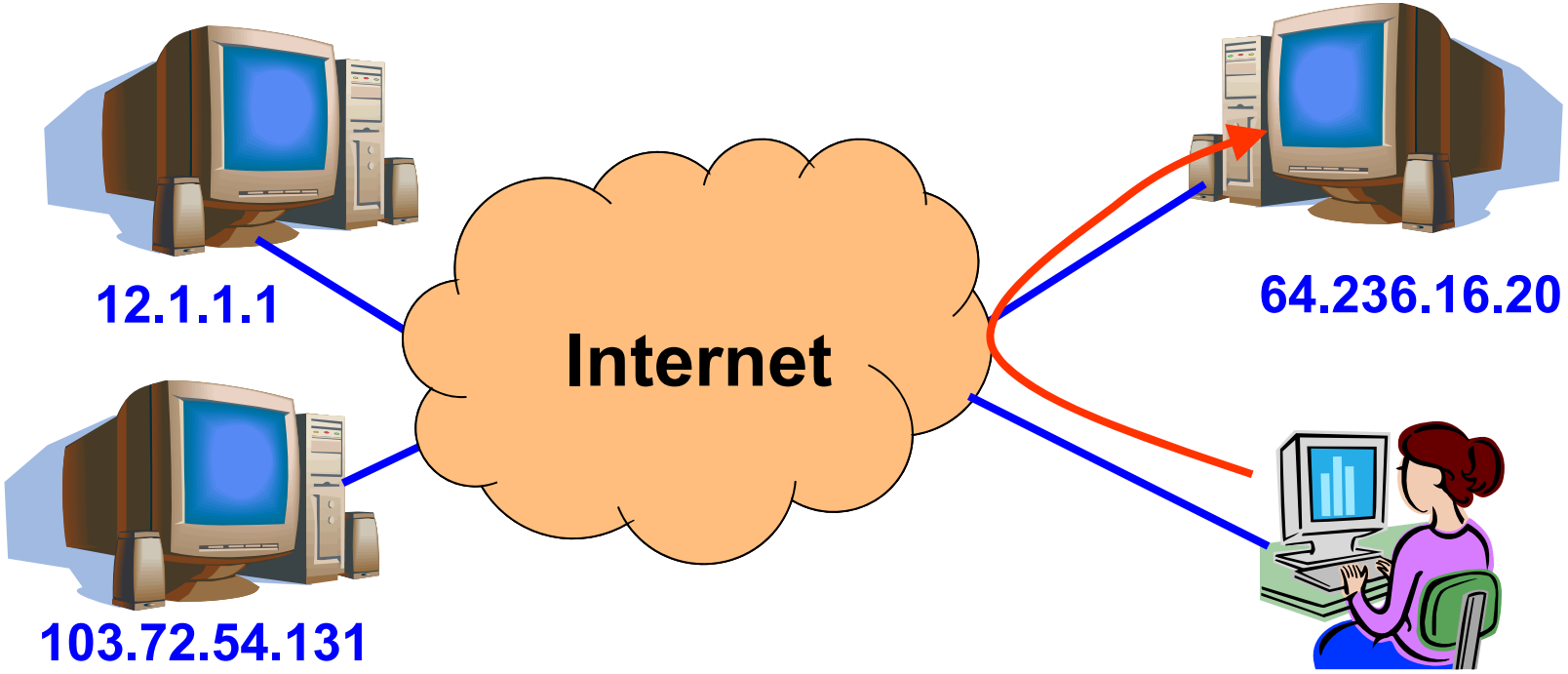
- **Solution #2:** single IP address, multiple machines
  - Same name and IP address for all of the replicas
  - Run multiple machines behind a single IP address



  - Ensure all packets from a single TCP connection go to the same replica

Hosting: Multiple Machines Per Site

- **Solution #3:** multiple addresses, multiple machines
  - Same name but different addresses for all of the replicas
  - Configure DNS server to return different addresses

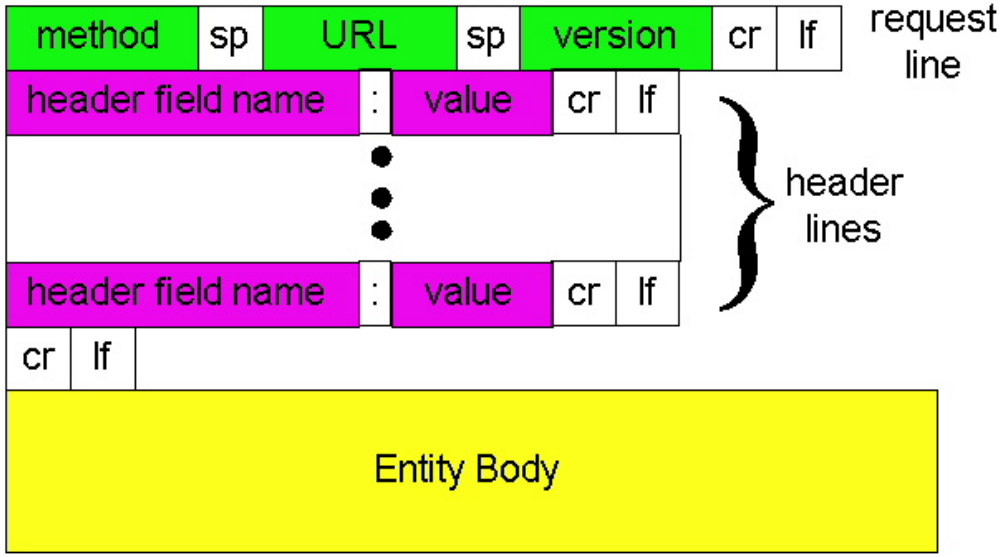


Summary of Web and HTTP

- **The major application on the Internet**
  - Majority of traffic is HTTP (or HTTP-related)
- **Client/server model:**
  - Clients make requests, servers respond to them
  - Done mostly in ASCII text (helps debugging!)
- **Various headers and commands**
  - Too many to go into detail here
  - Many web books/tutorials exist

Summary: HTTP

- **HTTP message format**
  - **ASCII requests, header lines, entity body, and responses line**



- **How does a client locate a server?**
- **Is the application extensible, robust, scalable?**

Summary: HTTP

- HTTP message flow
  - stateless server
    - each request is self-contained; thus cookie and authentication are needed in each message
  - reducing latency
    - persistent HTTP
    - conditional GET reduces server/network workload and latency
    - cache and proxy reduce traffic and latency

Chapter 7: Roadmap

- Application and Application Layer
- C/S and P2P Application Architectures
- DNS
- FTP
- E-Mail and SMTP/POP/IMAP
- WWW and HTTP
- DHCP

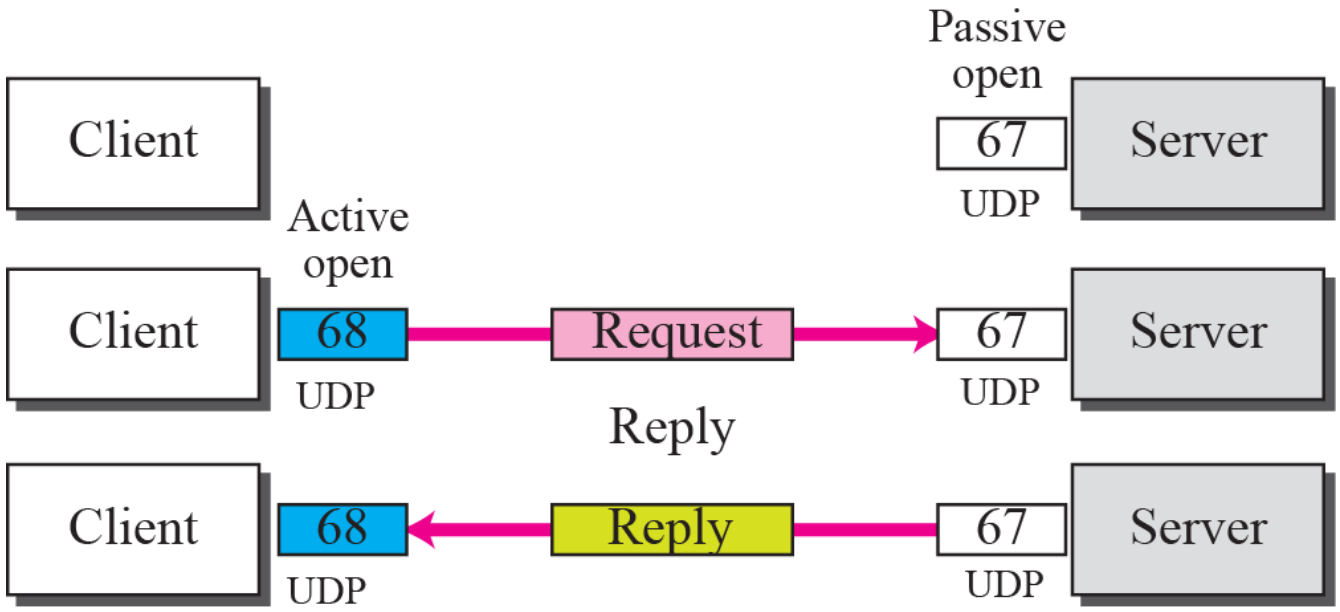
DHCP

- Dynamic Host Configuration Protocol
- Standard protocol
- Defined by RFC 1541 (superseded by RFC 2131)
- Runs over UDP, utilizing 2 ports:
  - 67 – connections to server
  - 68 – connections to client
- Uses client–server model
- Extension of BOOTP

DHCP

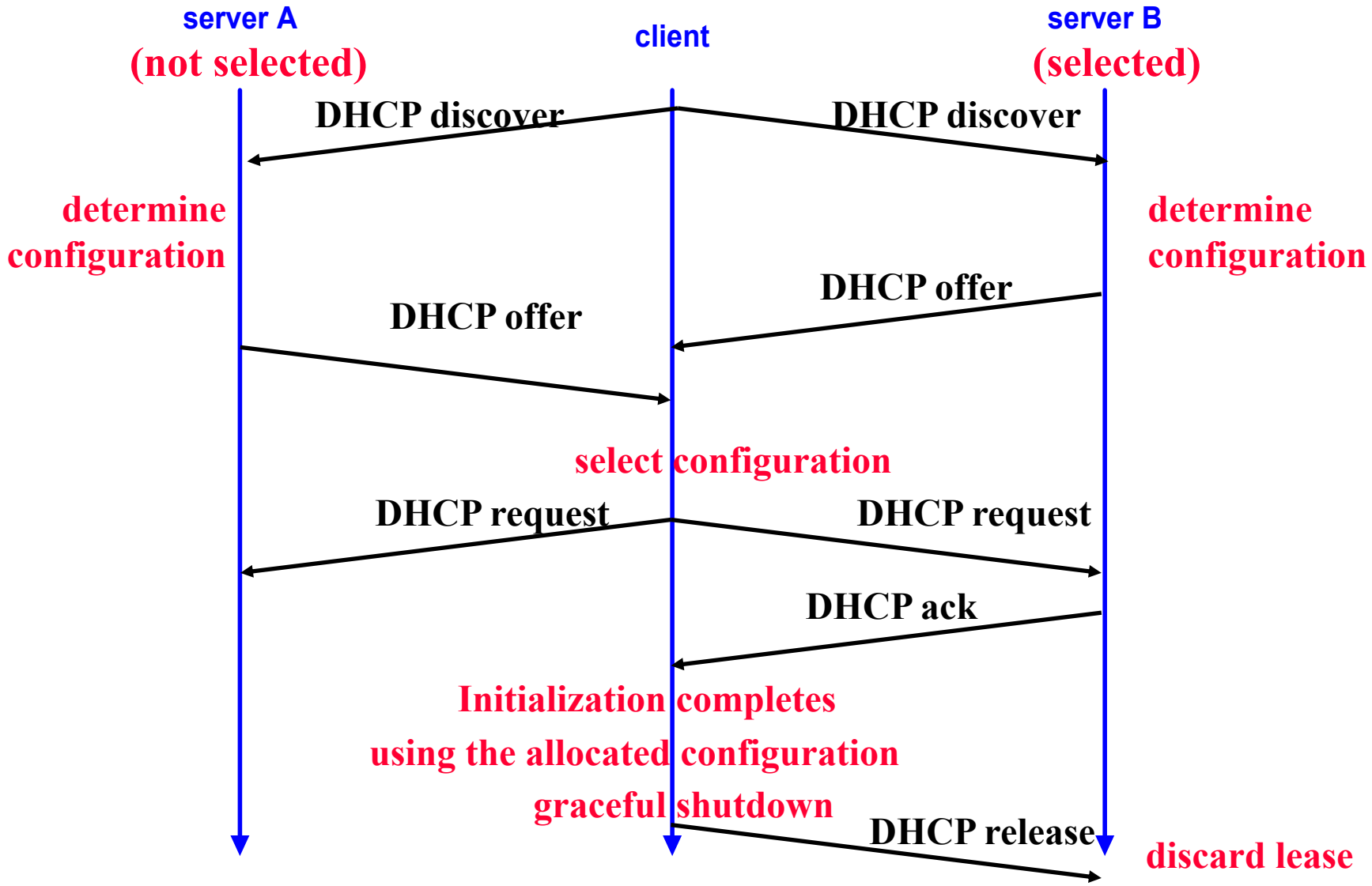
- Provides automatic configuration of the host connected to network or booted
- Provides hosts with initial configuration information upon boot up:
  - IP address with subnet mask,
  - default gateway,
  - IP address of the DNS server

DHCP



Use of UDP ports

DHCP Interaction



BOOTP/DHCP Message Format

OpCode	Hardware Type	Hardware Address Length	Hop Count
Number of Seconds		Unused (in BOOTP) Flags (in DHCP)	
Transaction ID			
Client IP address			
Your IP address			
Server IP address			
Gateway IP address			
Client hardware address (16 bytes)			
Server host name (64 bytes)			
Boot file name (128 bytes)			
Options			

(There are >100 different options)

DHCP Message Type

Value	Message Type
1	DHCPDISCOVER
2	DHCPOFFER
3	DHCPREQUEST
4	DHCPDECLINE
5	DHCPACK
6	DHCPNAK
7	DHCPRELEASE
8	DHCPINFORM

Message Types

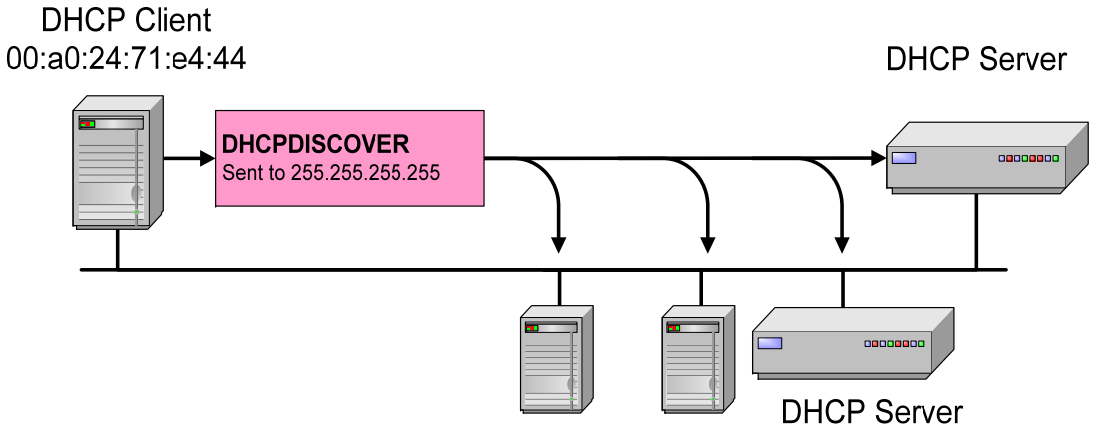
- DHCPDISCOVER: Broadcast by a client to find available DHCP servers.
- DHCPOFFER: Response from a server to a DHCPDISCOVER and offering IP address and other parameters.
- DHCPREQUEST: Message from a client to servers that does one of the following:
  - Requests the parameters offered by one of the servers and declines all other offers.
  - Verifies a previously allocated address after a system or network change (a reboot for example).
  - Requests the extension of a lease on a particular address.

Message Types (Contd.)

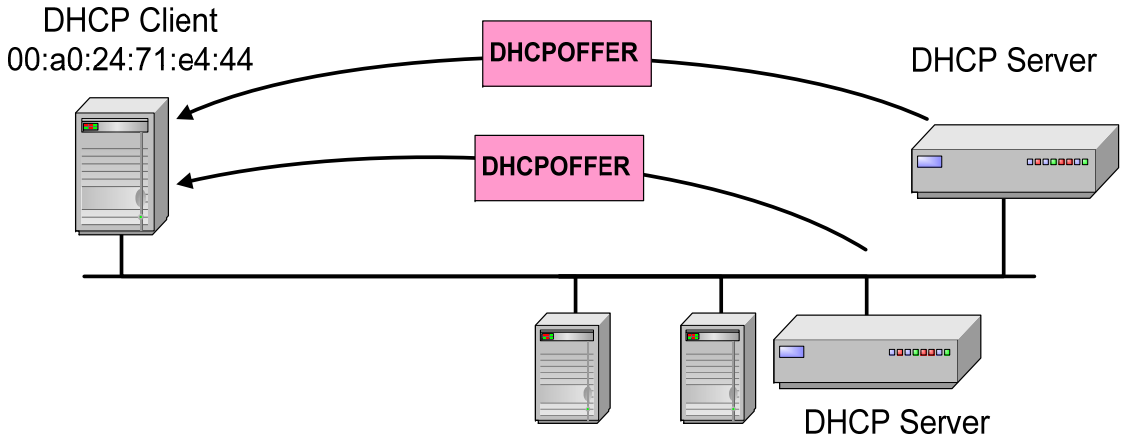
- DHCPACK: Acknowledgement from server to client with parameters,
- including IP address.
- DHCPNACK: Negative acknowledgement from server to client, indicating that the client's lease has expired or that a requested IP address is incorrect.
- DHCPDECLINE: Message from client to server indicating that the offered address is already in use.
- DHCPRELEASE: Message from client to server canceling remainder of a lease and relinquishing network address.
- DHCPINFORM: Message from a client that already has an IP address (manually configured for example), requesting further configuration parameters from the DHCP server.

DHCP Operations

- **DCHP DISCOVER (broadcast)**



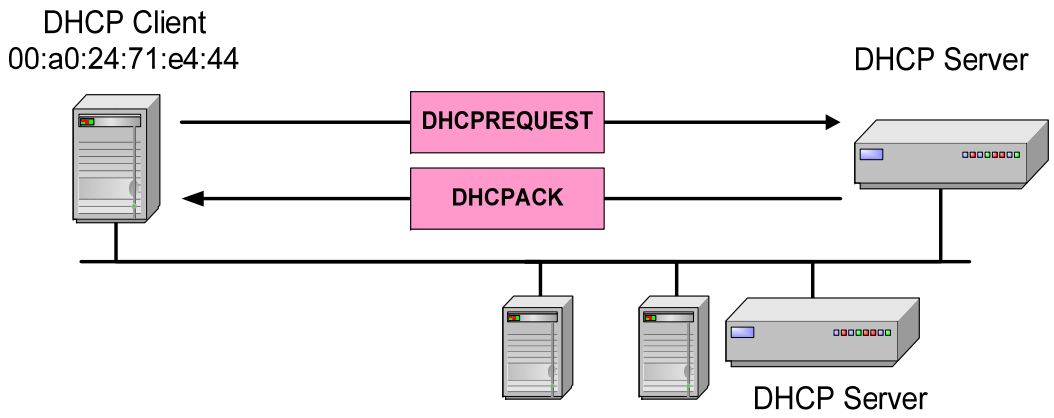
- **DCHP OFFER**



DHCP Operations

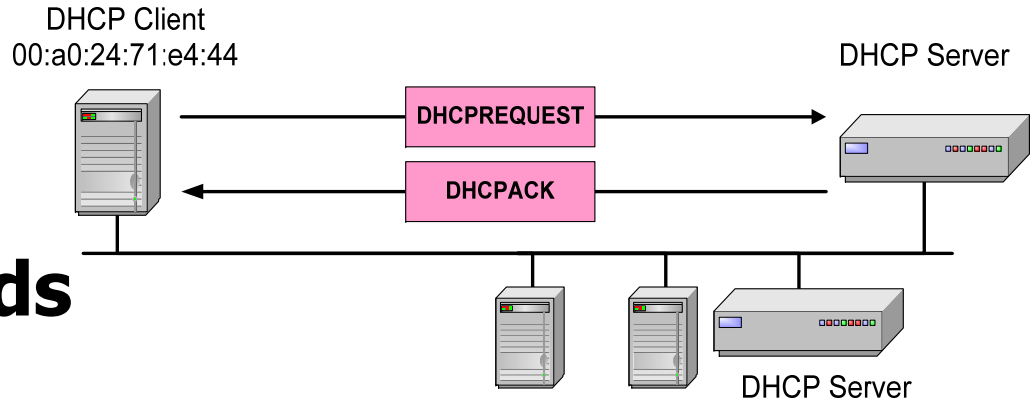
- **DCHP REQUEST**

  At this time, the DHCP client can start to use the IP address



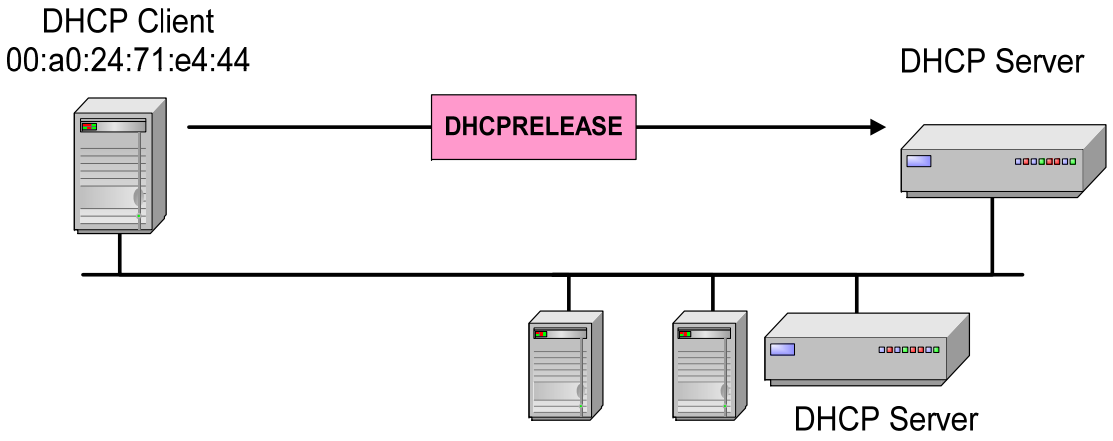
- **Renewing a Lease (sent when 50% of lease has expired)**

  If DHCP server sends DHCPNACK, then address is released.



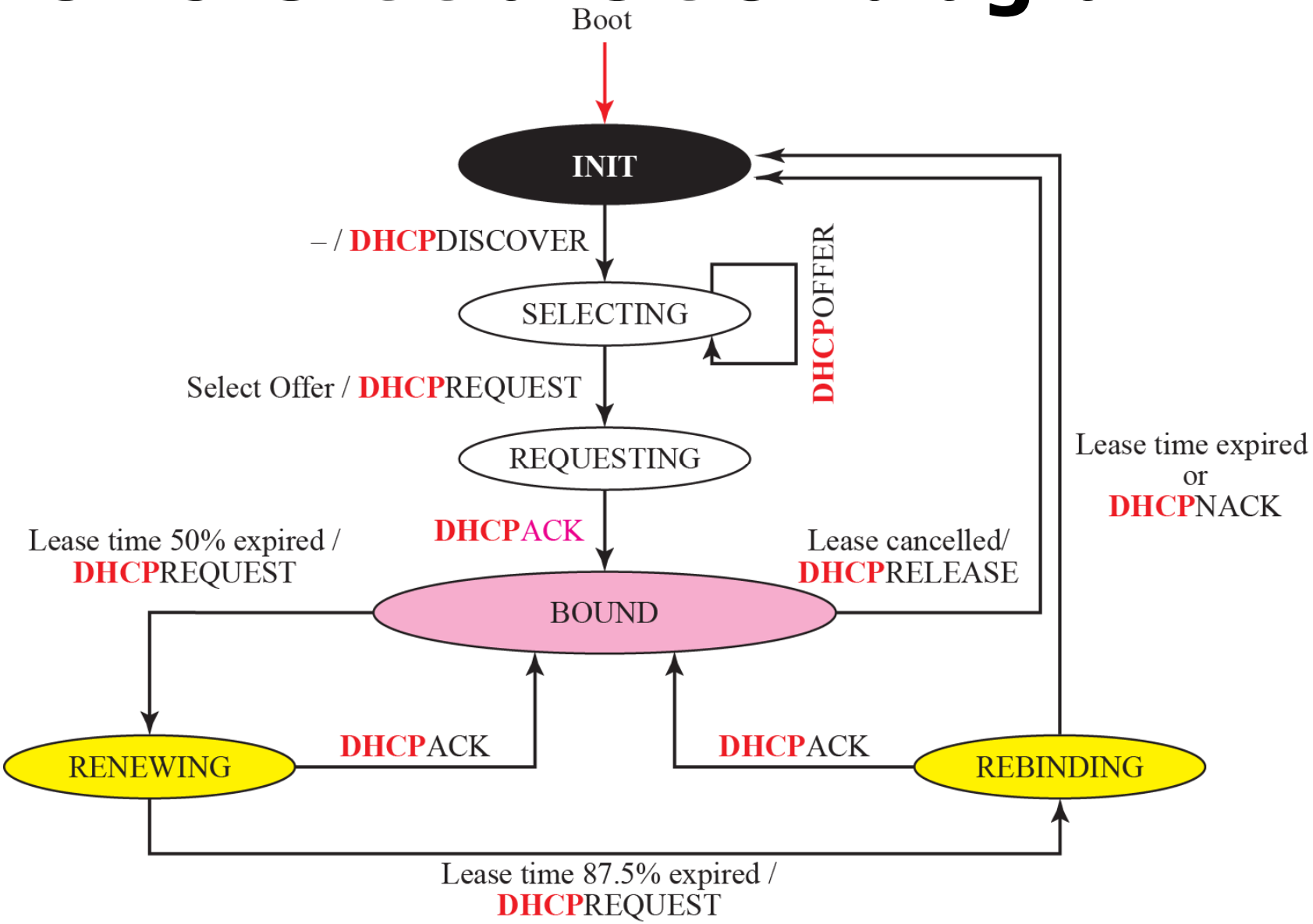
DHCP Operations

- **DCHP RELEASE**

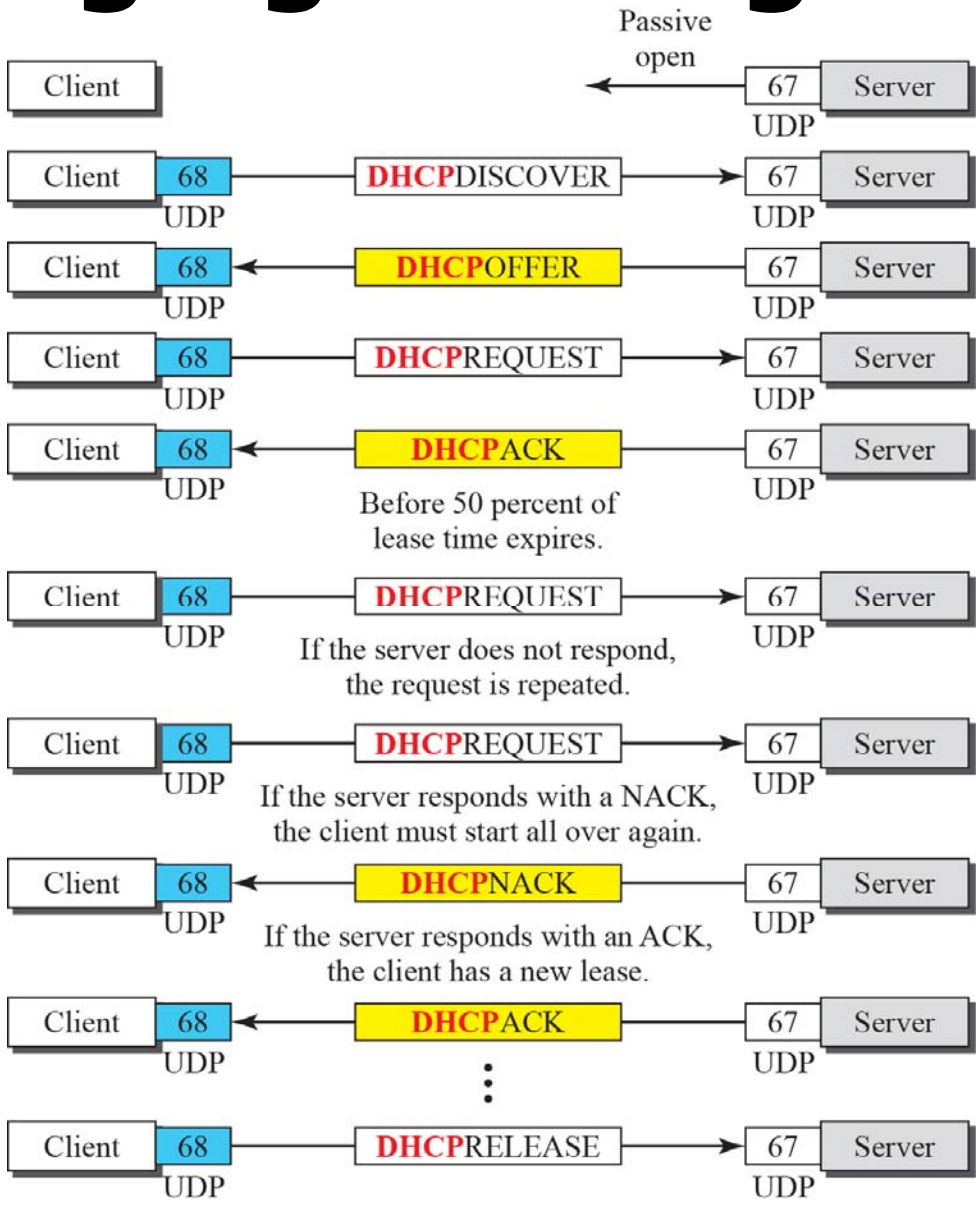


At this time, the DHCP client has released the IP address

DHCP client transition diagram



Exchanging messages



DHCP

- Static/Manual allocation
  - Server allocates IP chosen by the Admin
  - Server configuration includes
    - IP-Address -and-
    - MAC-Address

    for every client
  - DHCP only used to convey assigned address to client

DHCP

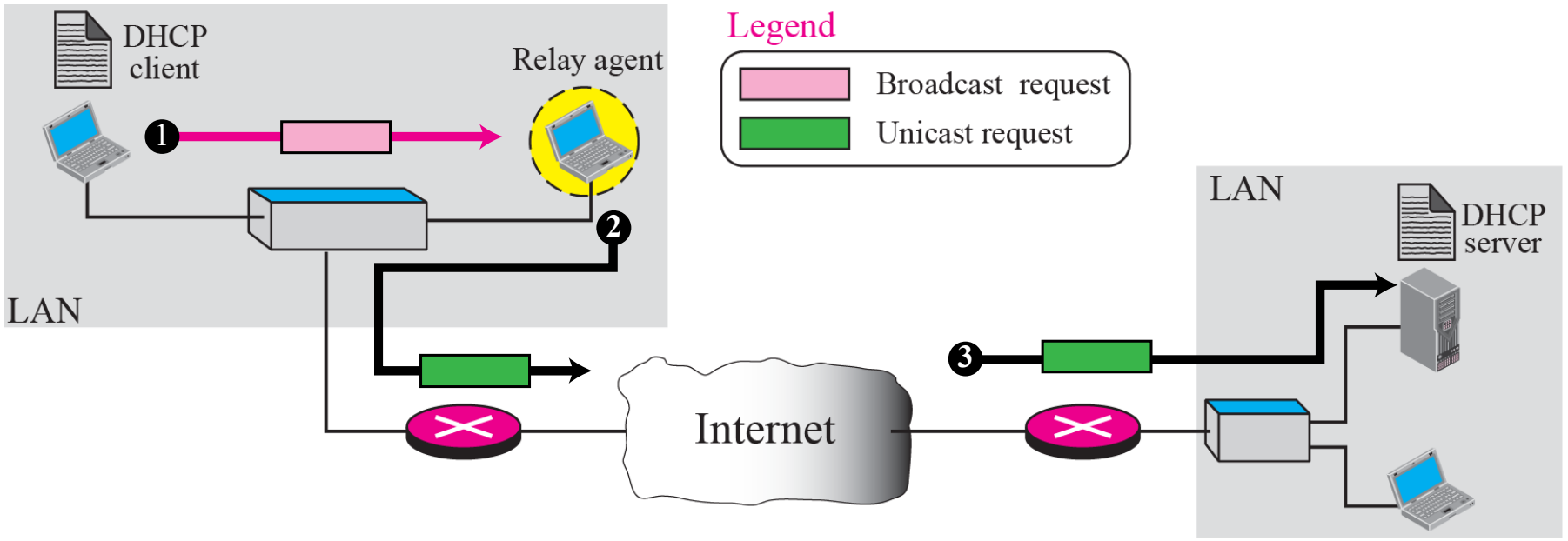
- Automatic allocation
  - IP address is permanently associated with a MAC address
  - till administrator intervenes the infinite lease
- Dynamic allocation
  - Like Automatic allocation
  - Except:
    - server tracks leases
    - give IP addresses whose lease has expired to other DHCP clients

end

DHCP

- Client is responsible to renew/release IP
- Lease timestamps:
  - Total lease duration
  - T1 (0.5 * duration_of_lease)
    - client enters the RENEWING state
    - contacts the server that originally issued network address
  - T2 (0.875 * duration_of_lease)
    - client enters the REBINDING state
    - attempts to contact any server

Client and Server on two different networks



DHCP Conclusion

- works well if you have to manage a lot of mobile users:
  - people with laptops working in and out of the office
  - hosts coming and going with a great amount of frequency
- perfect when Network parameters have changed
  - only one point you have to work at
- sharing a limited pool of IP addresses
- Mixture of allocation types can be used

DHCP Conclusion

- Insecurity of UDP
- Broadcast of messages
- Risk of wrong configuration causing
  - High traffic
  - Waste of IP Addresses
  - A lot more work than expected
- Detailed and precise analysis before implementation needed