Lecture 9: Application layer

Reading Chapter 7

Computer networks, Tanenbaum

Contents

- Application layer
  - Fundamental concepts
  - Case study: HTTP, Mail, FTP…

Fundamental concepts

Application layer in OSI model

Application (HTTP, Mail, …)
Transport (UDP, TCP …)
Network (IP, ICMP…)
Datalink (Ethernet, ADSL…)
Physical (bits…)

Protocols communication between parties of the application

Transmission data between application

Application and service?

MUSIC ONLINE

VoIP

GAME
ON LINE

CHAT

VoD

e-Office

e-BANK

SMS

MAIL

E-learning

WEB

YOUTUBE

VIDEO
CONFERENCE

FTP

EBAY

GOOGLE

SKYPE

Social
networks

SSH

NEWS

BITTORENT

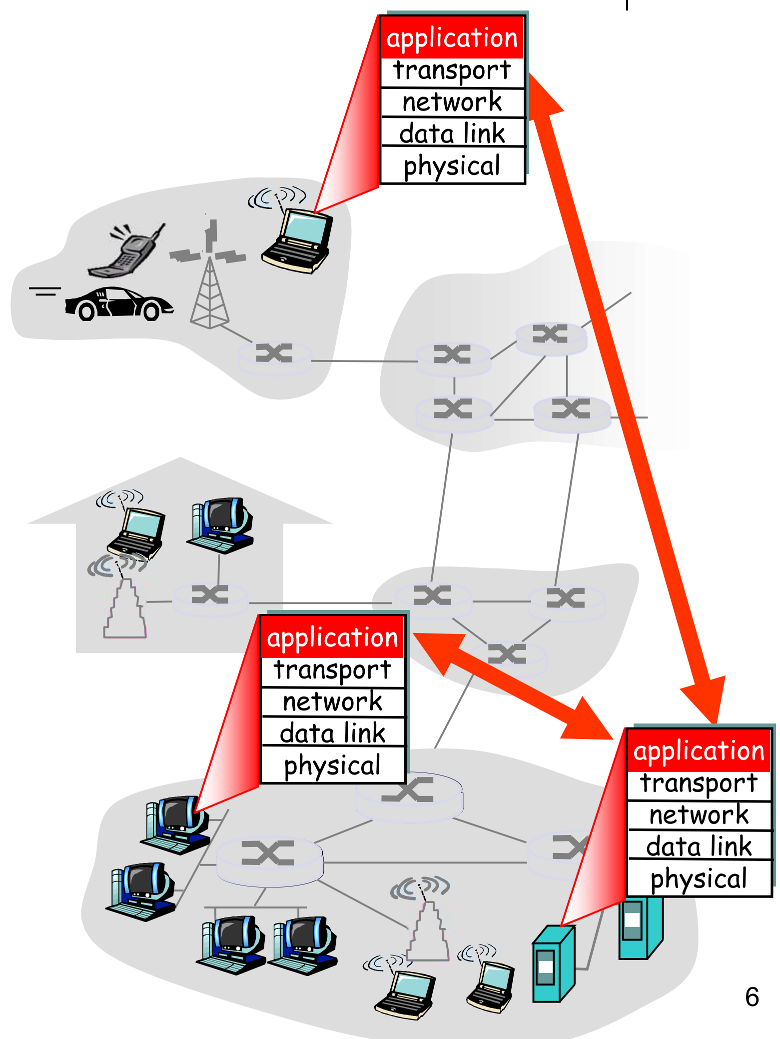
E-COMMERCE

GRID

e-Goverment

Application and application protocol

- Application protocol
  - Define communication rule
  - Use service of transport layer (TCP/UDP…)
- Application:
  - Is a process on the internet. They communicate to each other by exchanging messages.
  - Runs on end systems
  - Use application protocol for providing service
- Example of application/protocol:
  - Web (HTTP)
  - Mail (SMTP/POP/IMAP) …



Components of an application

- Application software is compose of
  - User interface:
    - Interfacing with users,
    - e.g. Web browser (Firefox, IE), mail reader(Thunderbird, Outlook,..)
    - Implement one part of application protocol
  - Server program:
    - Cung cấp dịch vụ cho người sử dụng
- Application process: the application software running on an OS

Communication between application processes

- Socket: an interface between application process and transport layer
  - Processes use services provided by transport layer to exchange information
- Socket is identified by port number and IP address

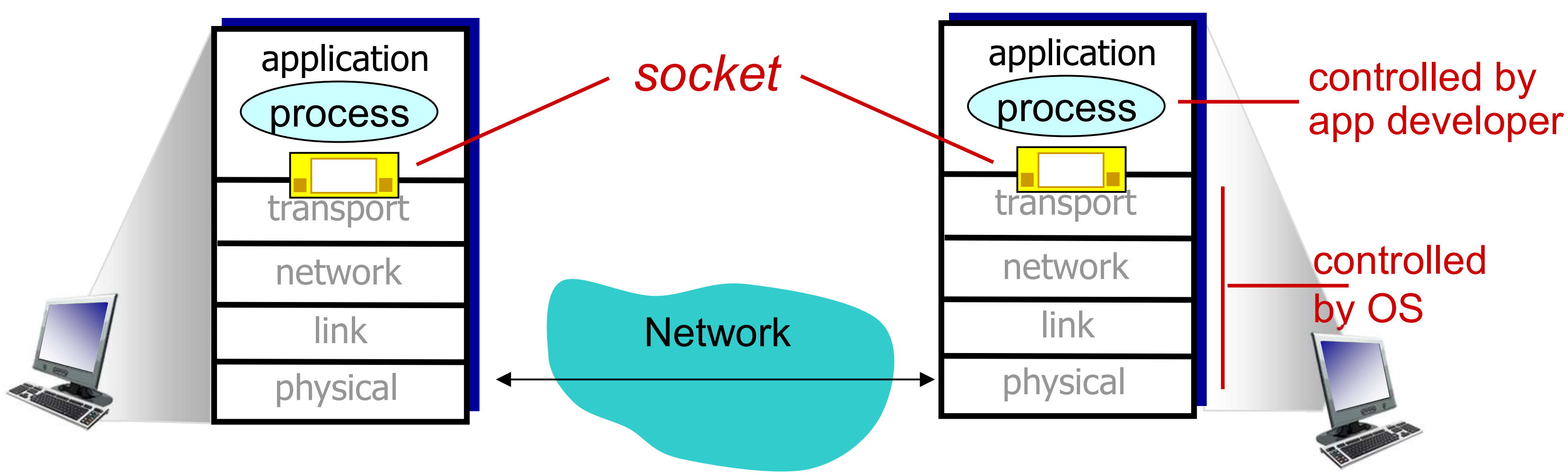
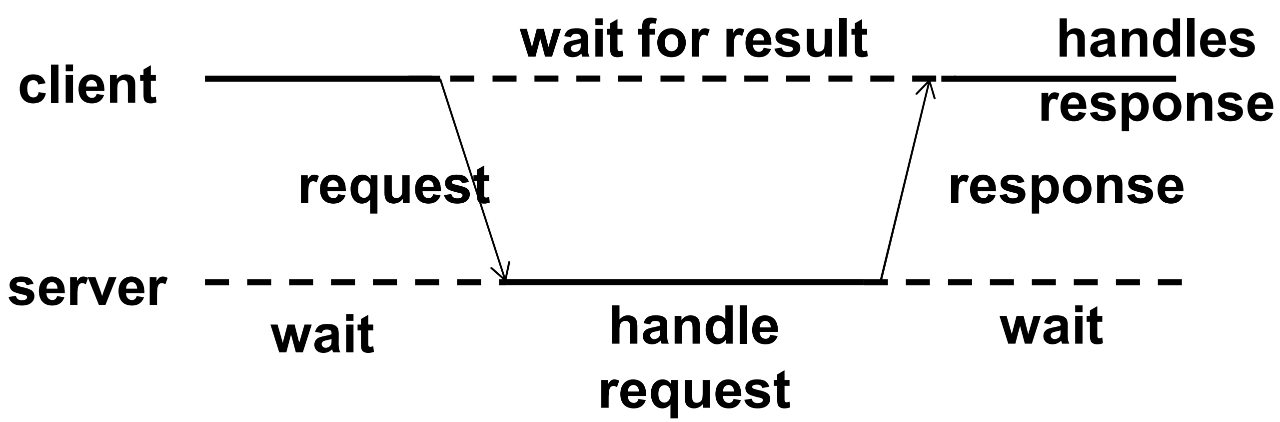


Image from: "Computer Networking: A Top Down Approach", Jim Kurose

Communication between processes

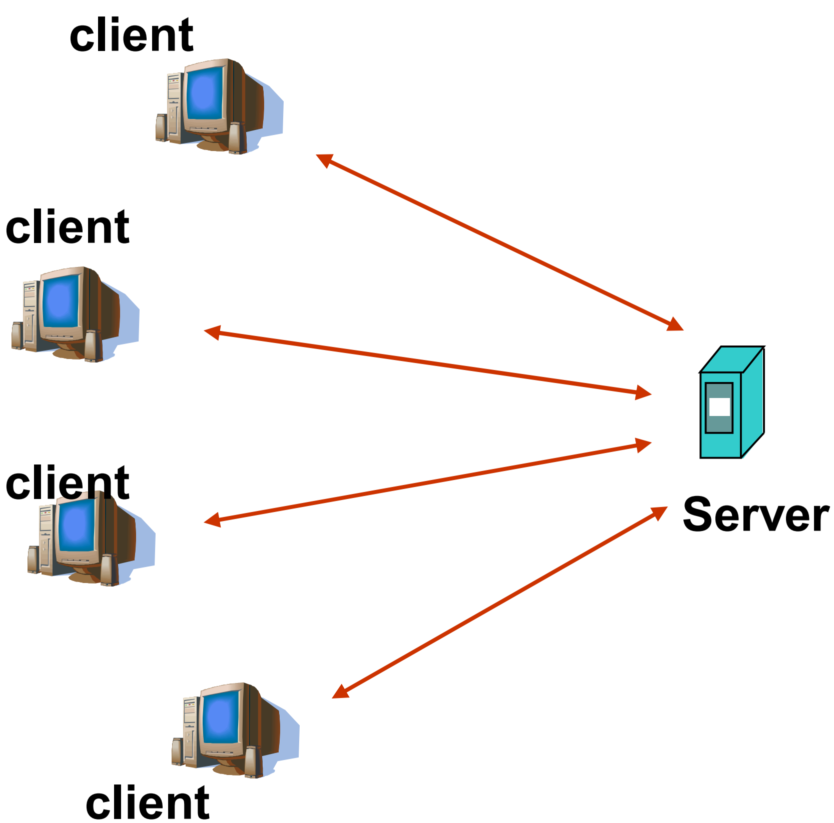
- Client process: send requests
- Server process: response requests
- Standard model: 1 server – many clients
- Clients need to know server address: IP address and port number



Application architecture

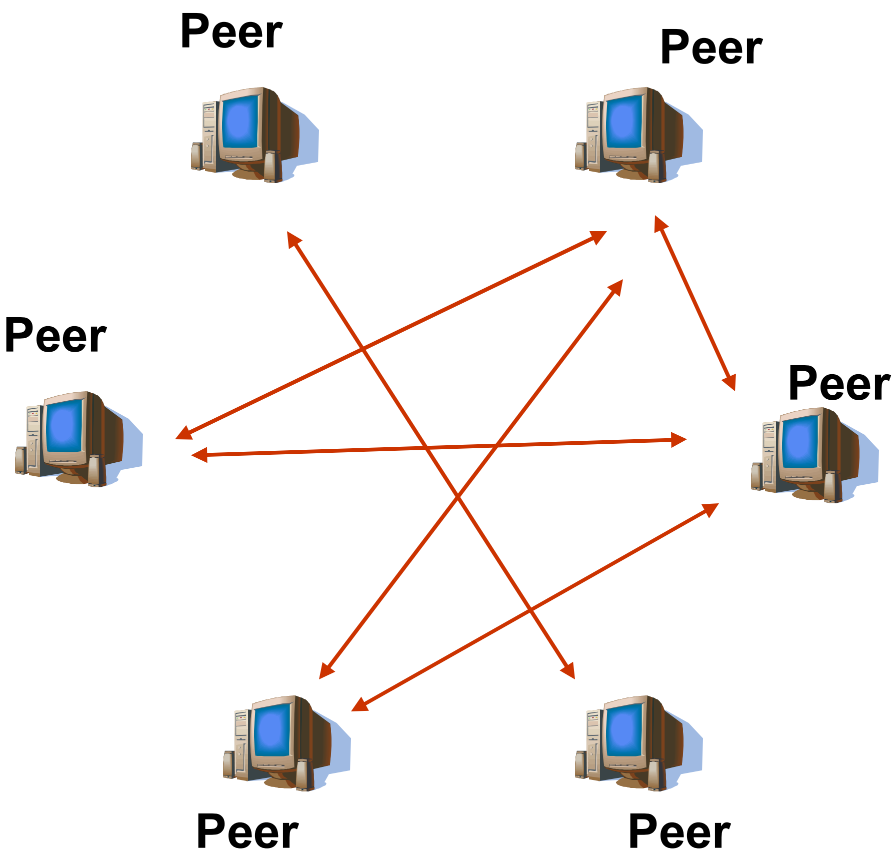
- Client-server
- P2P
- Hybrid

Client-server



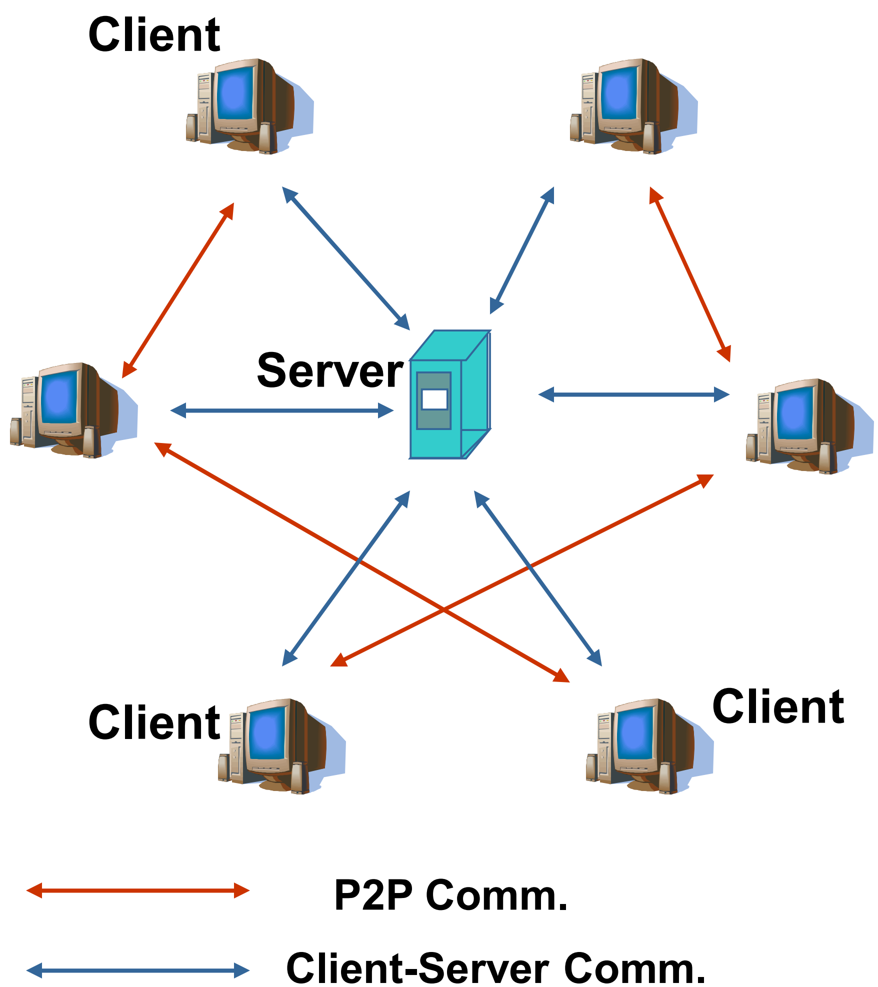
- Two kind of components: client and server
- Client
  - Client sends requests for service to server
  - Clients do not contact directly to each other
- Server
  - Always online waiting for service requests from clients
  - There may be backup servers for assuring high availability in failures
- e.g. Web, Mail, …

Pure Peer-to-peer architecture



- No center server, only peers as components
- Peers have equal role in the system
- Any two peers can communicate directly to each other but only when both are online.
- Peer does not need to be online all the time
- E.g. Gnutella, Bittorent

Hybrid architecture



- A center server for user management, indexing for search purpose.
- Clients communicate directly to each other after authentication process with server.
- E.g. Skype (before 2016)
  - Skype server manage user lists, authentification
  - After authentification users communicate directly to each other

Domain name service

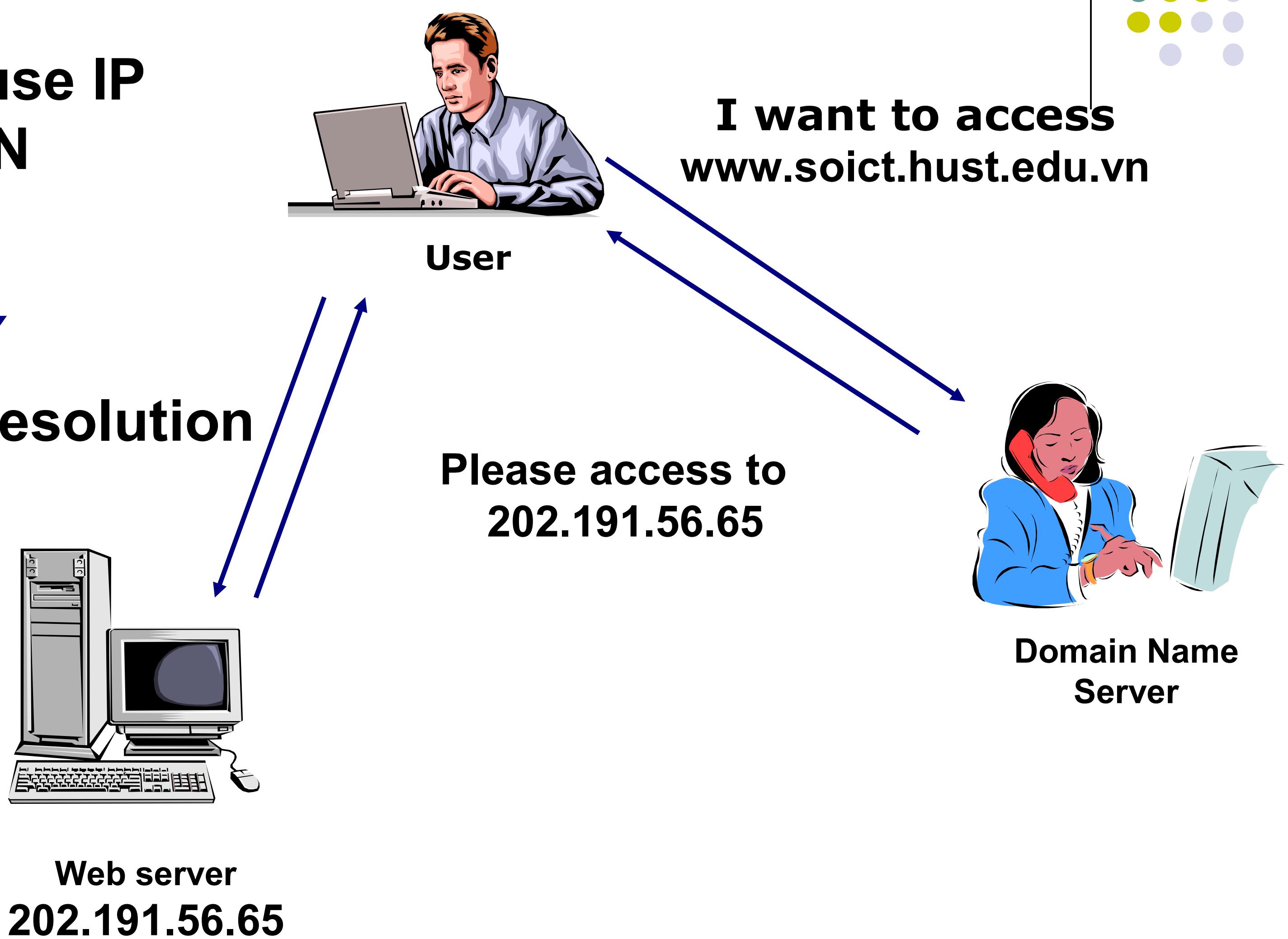
Introduction

- Domain name: identifier on application layer for network node
  - Internet management should be centralised
  - International: ICANN
  - Vietnam: VNNIC
- DNS(Domain Name System): the Internet's system for mapping alphabetic names to numeric Internet Protocol (IP) addresses
- Address resolution
  - Users/ Clients use domain name to access services
  - Computers and network devices cannot use domain name but IP address
- How to translate domain name to IP address and reverse?

Example of address resolution

- Computers use IP
- Users use DN

Need address resolution



DNS Server system

- Root server
  - Answer local DNS servers
  - Manage zone and decentralize the management to lower-level servers
  - There are 13 root servers (http://www.root-servers.org)

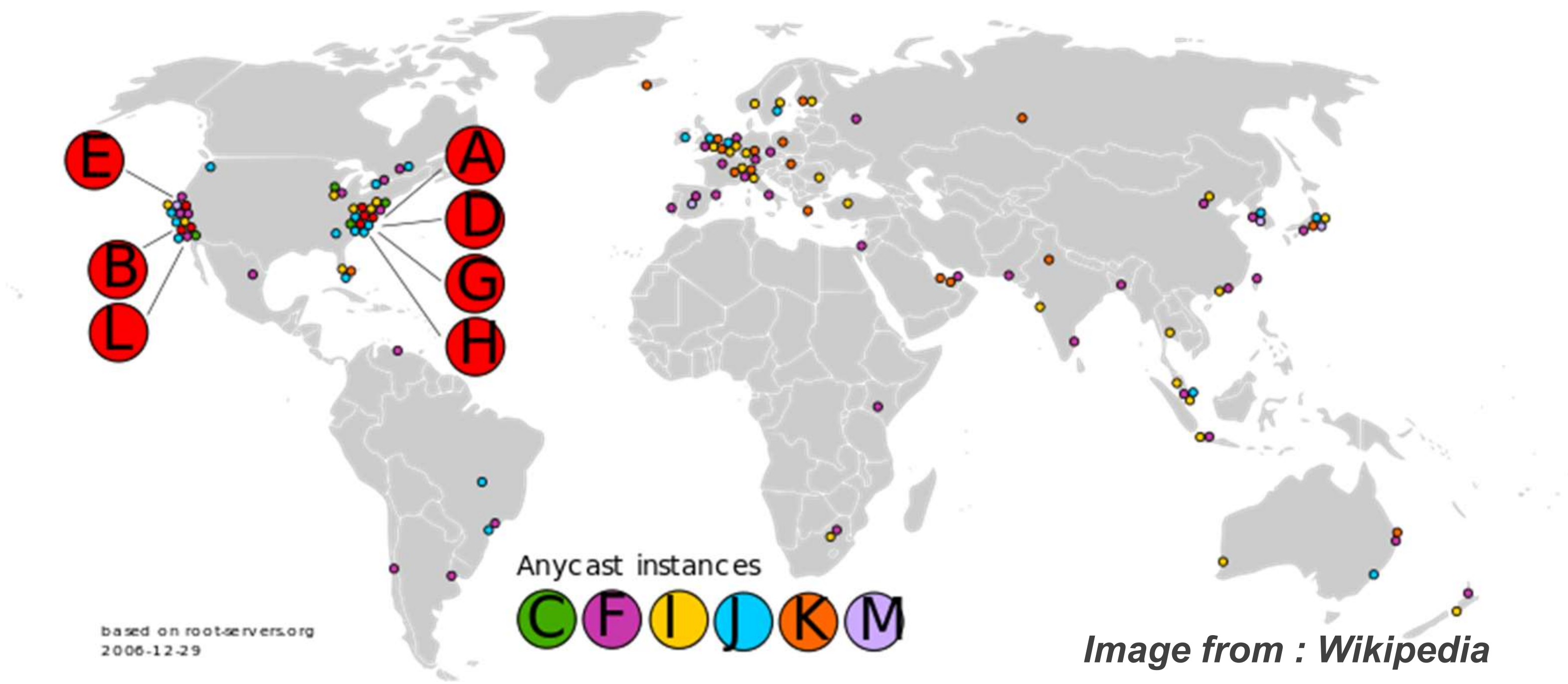


Image from : Wikipedia

DNS Server system (cont.)

- Top Level Domain servers
  - Manage domain level 1
- Authoritative DNS servers
  - Manage lower-level domains
- Servers of organisations: ISP
  - Not belong to DNS hierarchy
- Local server: for private network of institutes
  - Not belong to DNS hierarchy

Address resolution

- Self-resolution
  - File HOST:
    - Windows: C:\WINDOWS\system32\drivers\etc\
    - Linux: /etc/hosts
  - Application cache
- DNS service: client/server
  - Application protocol: DNS
  - Use UDP/TCP with the port 53
  - Recursive Query
  - Interactive Query

DNS Message

- DNS Query and DNS Reply: same format
- Identification
  - Response must have the same identification of the request
- Flags: control flags
- #Question: number of domain names requested
- QUESTION: requested domain names

Identification	**Flags**
#Question	**#Answer RRs**
#Authority RRs	**#Additional RRs**
QUESTION	
ANSWER	
AUTHORITY	
ADDITIONAL	

DNS Message

- #Answer RRs: Number of answered records
- ANSWER: Answered records
- # Authority RRs: Number of records that servers are authorized
- AUTHORITY: Records of authorized servers
- #Additional RRs: Number of additional records
- ADDITIONAL: additional records

Identification	**Flags**
#Question	**#Answer RRs**
#Authority RRs	**#Additional RRs**
QUESTION	
ANSWER	
AUTHORITY	
ADDITIONAL	

Example: dig linux.com

```
; <> DiG 9.9.2-P1 <> linux.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21655
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2,
ADDITIONAL: 3
;; QUESTION SECTION:
;linux.com. IN A
;; ANSWER SECTION:
linux.com. 1786 IN A 140.211.167.51
linux.com. 1786 IN A 140.211.167.50
;; AUTHORITY SECTION:
linux.com. 86386 IN NS ns1.linux-foundation.org.
linux.com. 86386 IN NS ns2.linux-foundation.org.
;; ADDITIONAL SECTION:
ns1.linux-foundation.org. 261 IN A 140.211.169.10
ns2.linux-foundation.org. 262 IN A 140.211.169.11
```

TTL: timing in cache

Example: dig linux.com

```
; <> DiG 9.9.2-P1 <> linux.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21655
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2,
ADDITIONAL: 3
;; QUESTION SECTION:
;linux.com. IN A
;; ANSWER SECTION:
linux.com. 1786 IN A 140.211.167.51
linux.com. 1786 IN A 140.211.167.50
;; AUTHORITY SECTION:
linux.com. 86386 IN NS ns1.linux-foundation.org.
linux.com. 86386 IN NS ns2.linux-foundation.org.
;; ADDITIONAL SECTION:
ns1.linux-foundation.org. 261 IN A 140.211.169.10
ns2.linux-foundation.org. 262 IN A 140.211.169.11
```

Names of DNS servers answered the request
If ANSWER is empty, DNS Resolver sends the request to these DNS servers

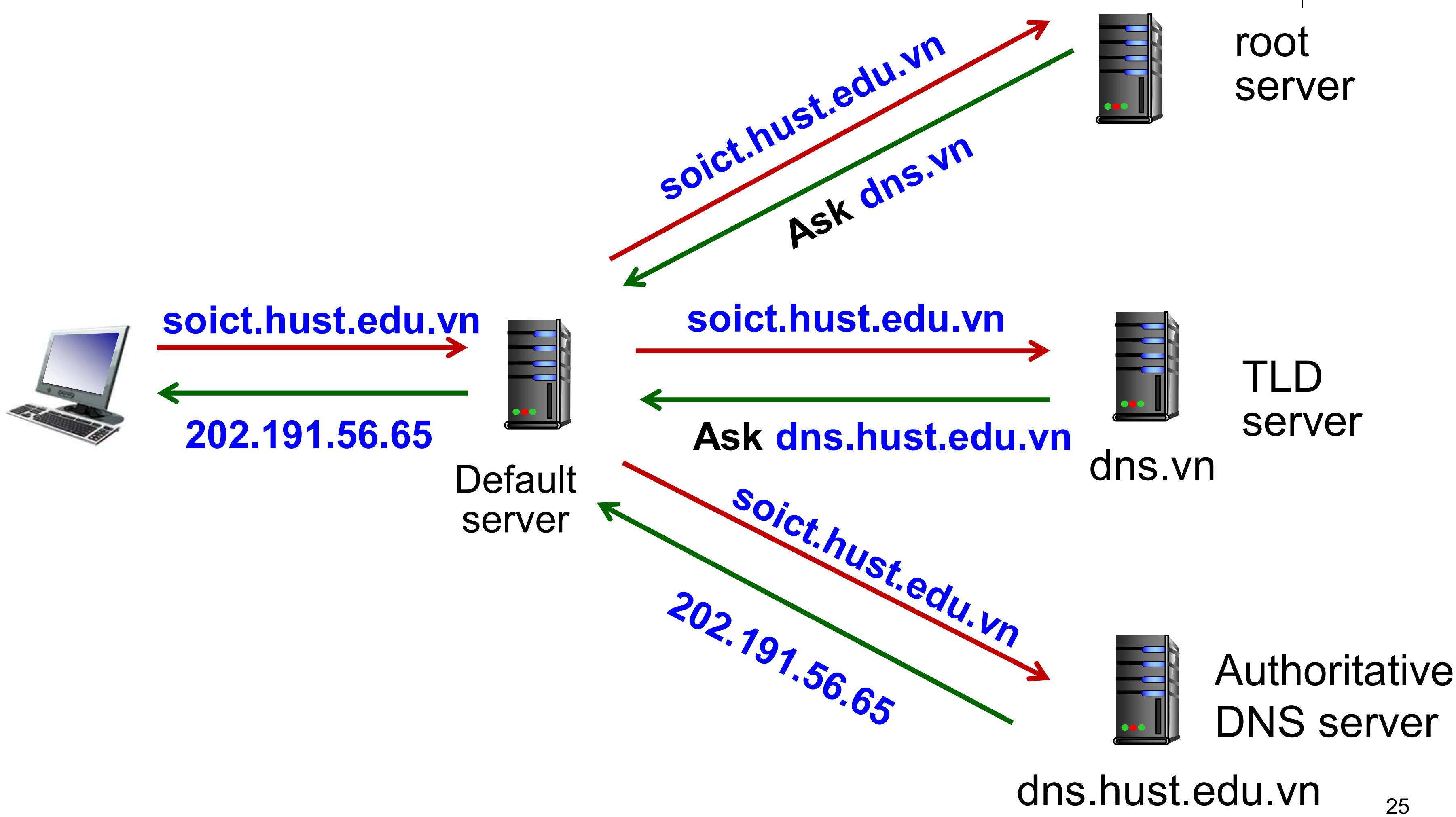
Example : dig linux.com

```
; <> DiG 9.9.2-P1 <> linux.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21655
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2,
ADDITIONAL: 3
;; QUESTION SECTION:
;linux.com. IN A
;; ANSWER SECTION:
linux.com. 1786 IN A 140.211.167.51
linux.com. 1786 IN A 140.211.167.50
;; AUTHORITY SECTION:
linux.com. 86386 IN NS ns1.linux-foundation.org.
linux.com. 86386 IN NS ns2.linux-foundation.org.
;; ADDITIONAL SECTION:
ns1.linux-foundation.org. 261 IN A 140.211.169.10
ns2.linux-foundation.org. 262 IN A 140.211.169.11
```

IP address of DNS servers.
Information will be stored in cache

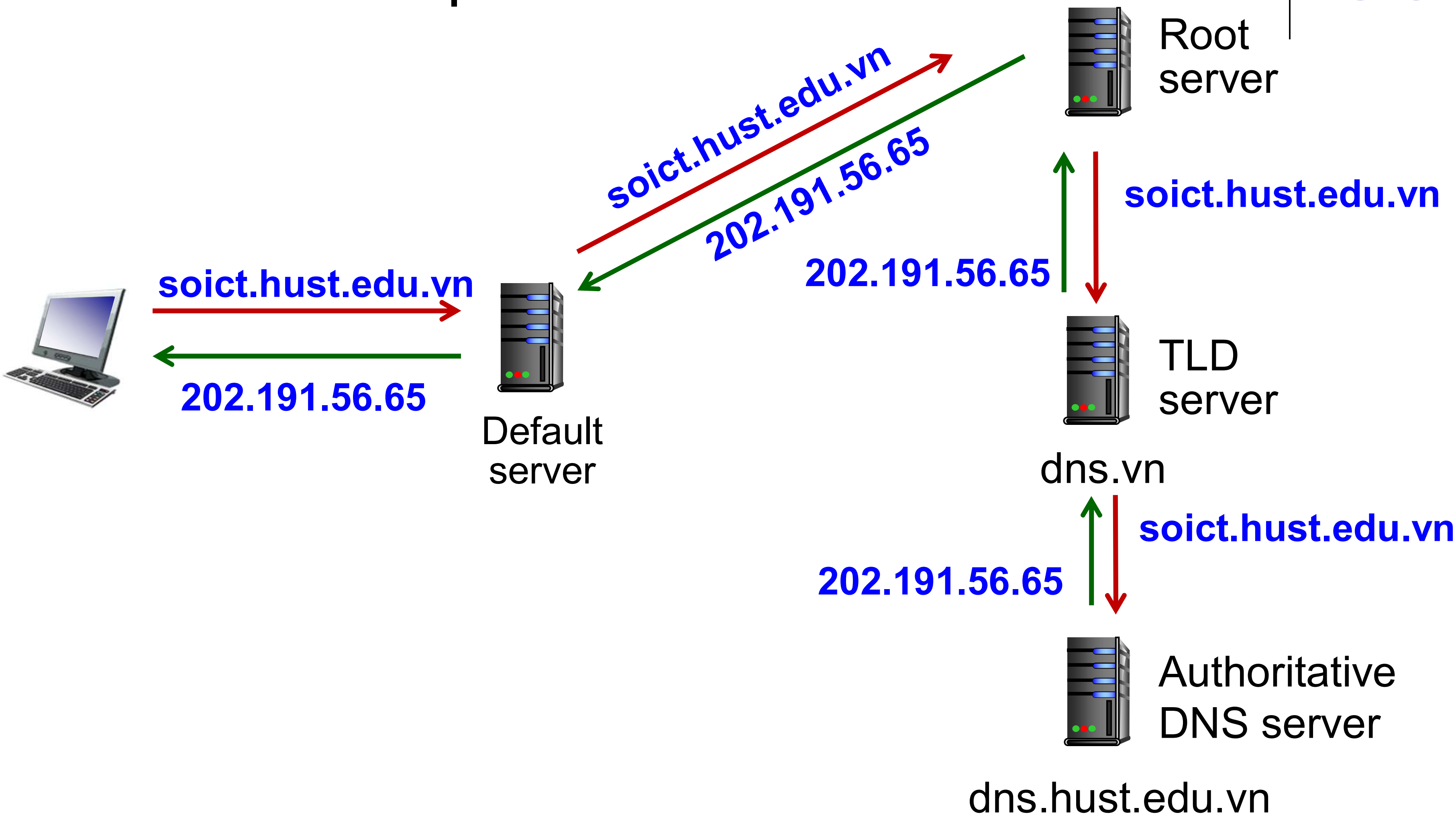
Interactive Query

- Default mechanism on DNS



Recursive Query

- Extensible option



HTTP and WWW

Reading 7.3

Computer Networks, Tanenbaum

HTTP và Web

- Internet before 1990s:
  - Limited using for government institutes, research centers, ...
  - Email or FPT services were not suitable for public data sharing
  - No effective mechanism to link scattered resources in the Internet
- In 1990, Tim Berners-Lee introduce World Wide Web:
  - Exchange information as hypertext using HTML (Hypertext Markup Language)
  - Objects are not needed to be packed as “all in one” as previous ones
  - Hypertexts only need to contain links to other objects (located by URL)

Uniform Resource Locator

- Areference to a web resource that specifies its location on a computer network and a mechanism for retrieving it

```
protocol://hostname[:port]/directory-path/resource
```

- *protocol*: http, ftp, https, smtp, rtsp…
- *hostname*: domain name or IP address
- *port:* port number (might not need)
- *directory path*: path to the resource
- *resource*: name of the resource

Web and HTTP

First, a quick review…

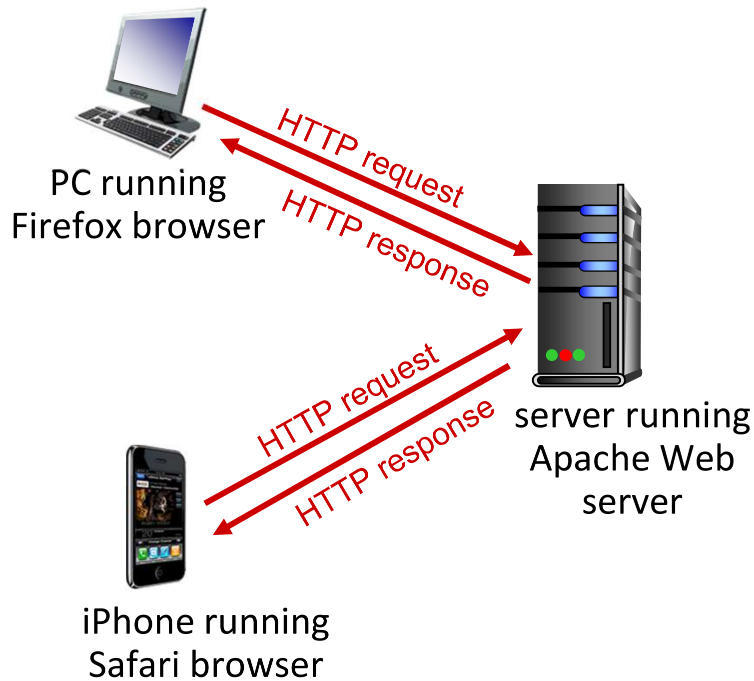
- web page consists of *objects,* each of which can be stored on different Web servers
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects, each* addressable by a *URL,* e.g.,

$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

HTTP overview

HTTP: hypertext transfer protocol

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



HTTP overview (continued)

HTTP uses TCP:

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

HTTP is "stateless"

- server maintains *no* information about past client requests

aside

protocols that maintain "state" are complex!

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

HTTP connections: two types

Non-persistent HTTP

1. TCP connection opened
2. at most one object sent over TCP connection
3. TCP connection closed

downloading multiple objects required multiple connections

Persistent HTTP

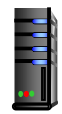
- TCP connection opened to a server
- multiple objects can be sent over *single* TCP connection between client, and that server
- TCP connection closed

Non-persistent HTTP: example

User enters URL: **`www.someSchool.edu/someDepartment/home.index`**

(containing text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80



1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80 "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

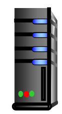
3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

Non-persistent HTTP: example (cont.)

User enters URL: **`www.someSchool.edu/someDepartment/home.index`**

(containing text, references to 10 jpeg images)



4. HTTP server closes TCP connection.

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

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

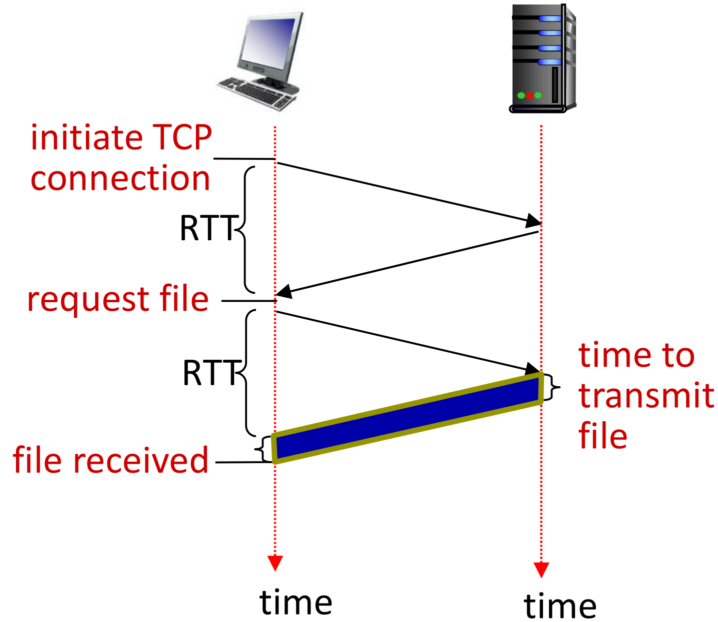
time

Non-persistent HTTP: response time

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

HTTP response time (per object):

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



Non-persistent HTTP response time = 2RTT+ file transmission time

Persistent HTTP (HTTP 1.1)

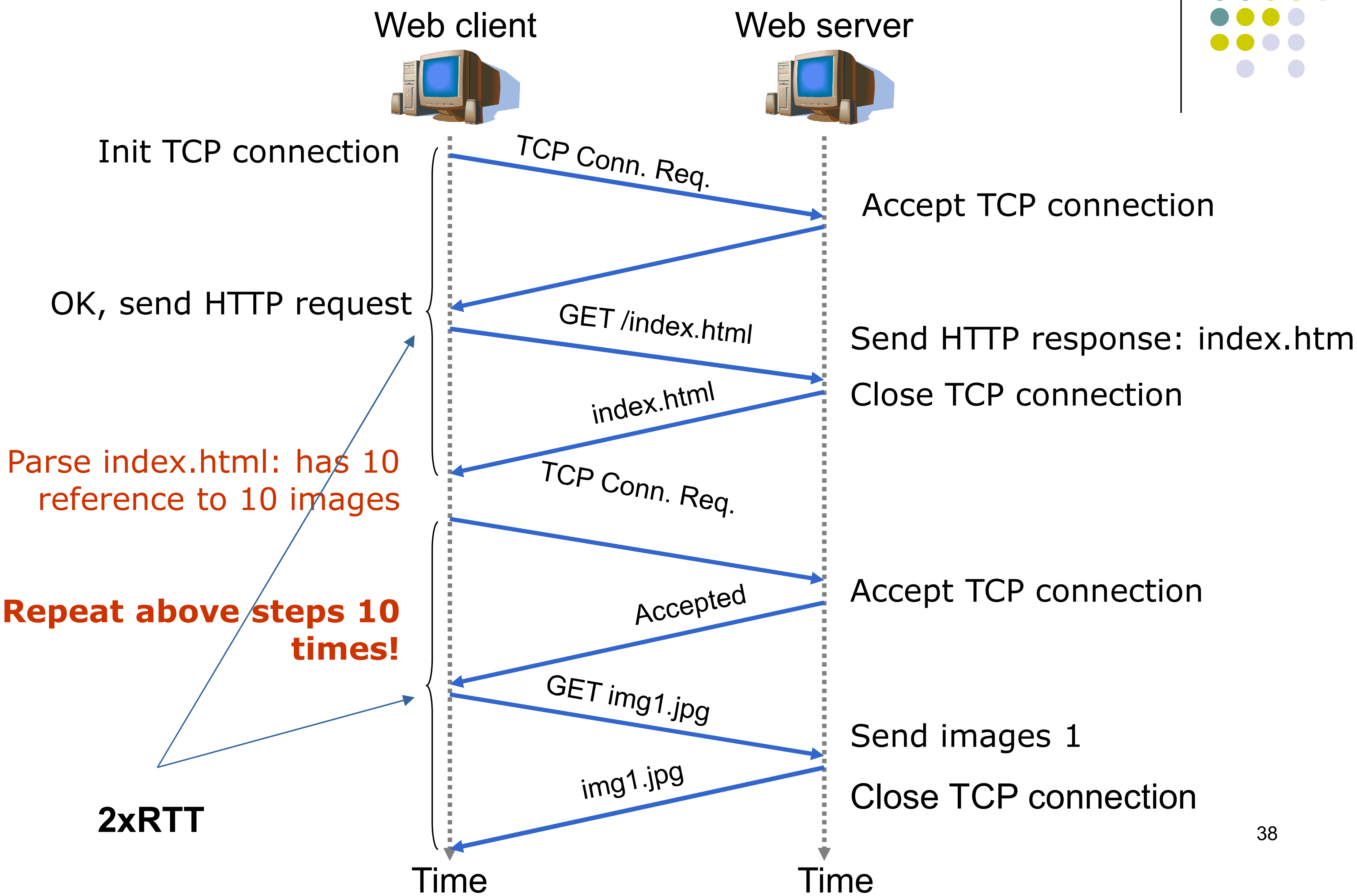
Non-persistent HTTP issues:

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

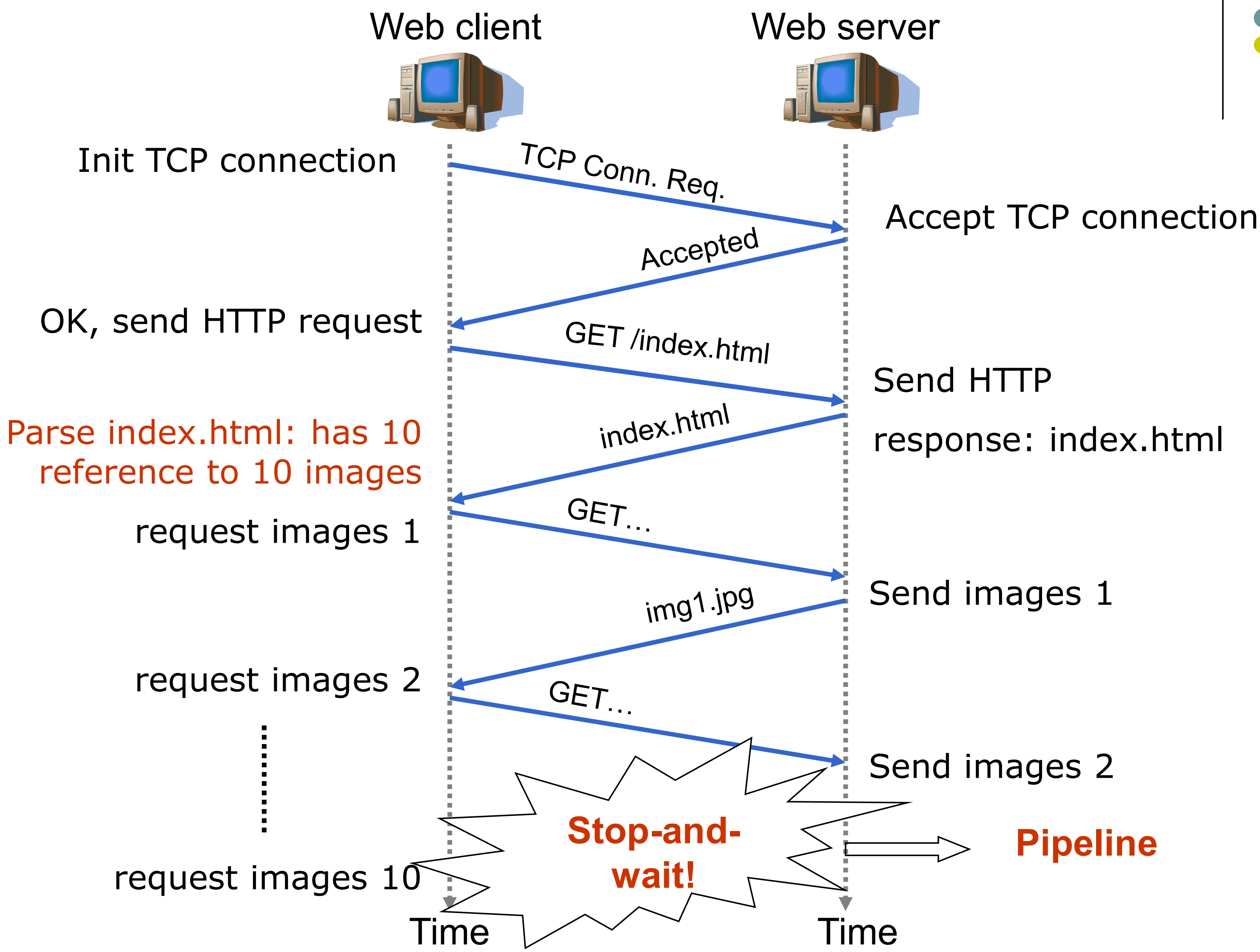
Persistent HTTP (HTTP1.1):

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

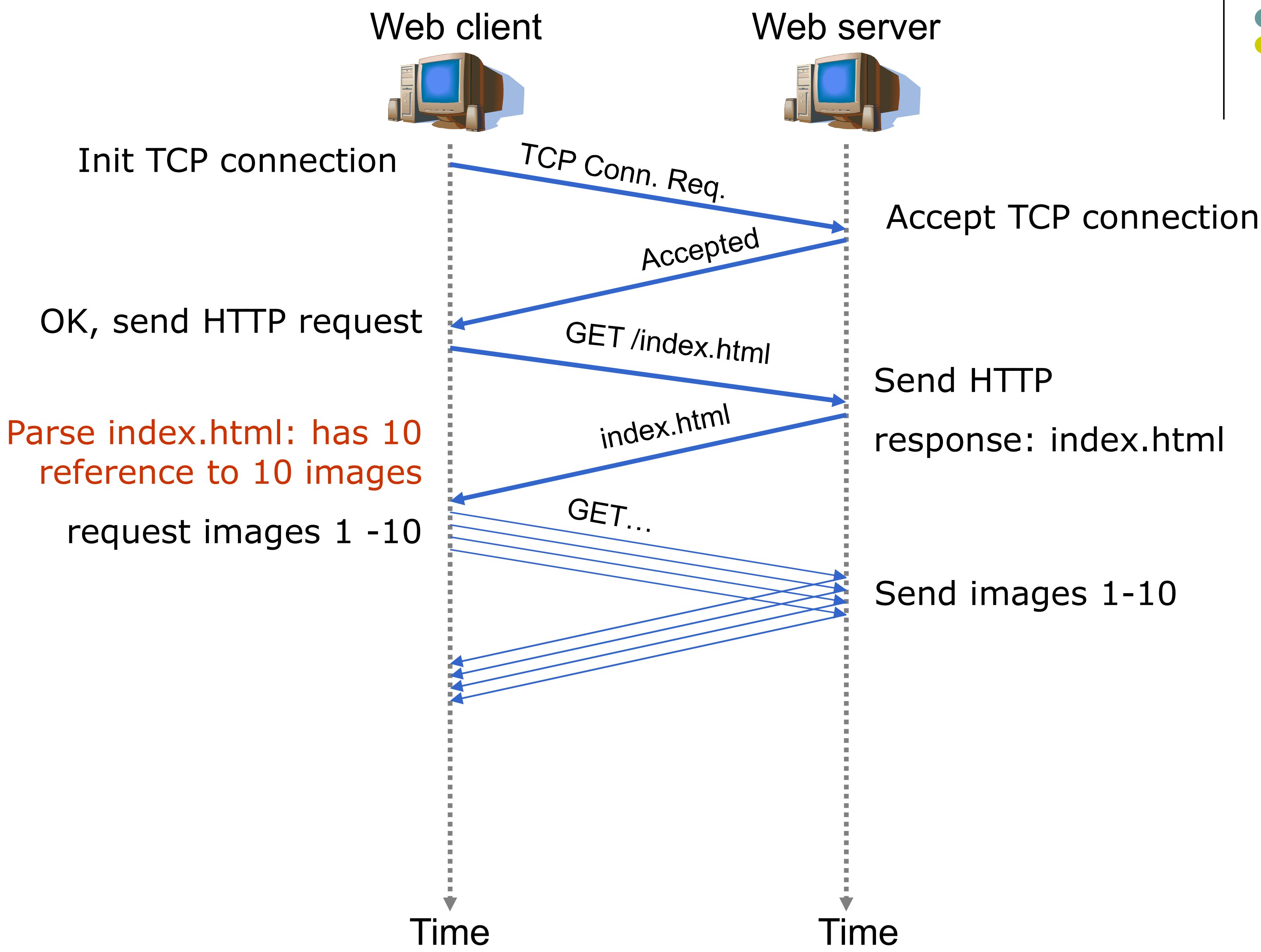
Operation of HTTP/1.0



Operation of HTTP/1.1



HTTP/1.1 with pipeline



HTTP request message

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

carriage return character

line-feed character

request line (GET, POST, HEAD commands)

header lines

carriage return, line feed at start of line indicates end of header lines

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

Other HTTP request messages

POST method:

- web page often includes form input
- user input sent from client to server in entity body of HTTP POST request message

GET method (for sending data to server):

- include user data in URL field of HTTP GET request message (following a ‘?’):

`www.somesite.com/animalsearch?monkeys&banana`

HEAD method:

- requests headers (only) that would be returned *if* specified URL were requested with an HTTP GET method.

PUT method:

- uploads new file (object) to server
- completely replaces file that exists at specified URL with content in entity body of POST HTTP request message

HTTP response message

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some 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 (in Location: field)

  400 Bad Request
  - request msg not understood by server

  404 Not Found
  - requested document not found on this server

  505 HTTP Version Not Supported

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet gaia.cs.umass.edu 80
```

- opens TCP connection to port 80 (default HTTP server port) at gaia.cs.umass. edu.
- anything typed in will be sent to port 80 at gaia.cs.umass.edu

2. type in a GET HTTP request:

```
GET /kurose_ross/interactive/index.php HTTP/1.1
Host: gaia.cs.umass.edu
```

- by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!

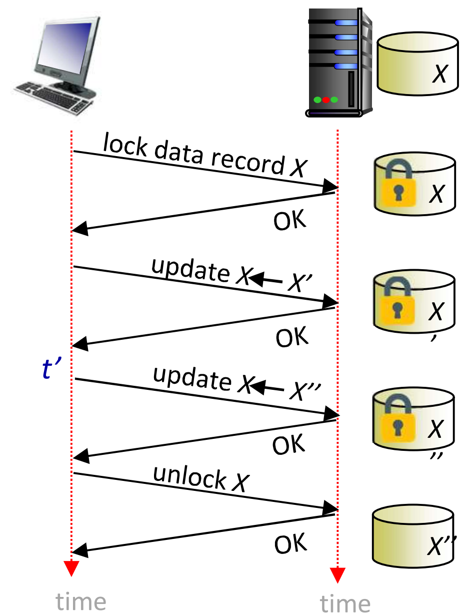
(or use Wireshark to look at captured HTTP request/response)

Maintaining user/server state: cookies

Recall: HTTP GET/response interaction is *stateless*

- no notion of multi-step exchanges of HTTP messages to complete a Web "transaction"
  - no need for client/server to track "state" of multi-step exchange
  - all HTTP requests are independent of each other
  - no need for client/server to "recover" from a partially-completed-but-never-completely-completed transaction

a stateful protocol: client makes two changes to X, or none at all



Q: what happens if network connection or client crashes at *t′* ?

Maintaining user/server state: cookies

Web sites and client browser use *cookies* to maintain some state between transactions

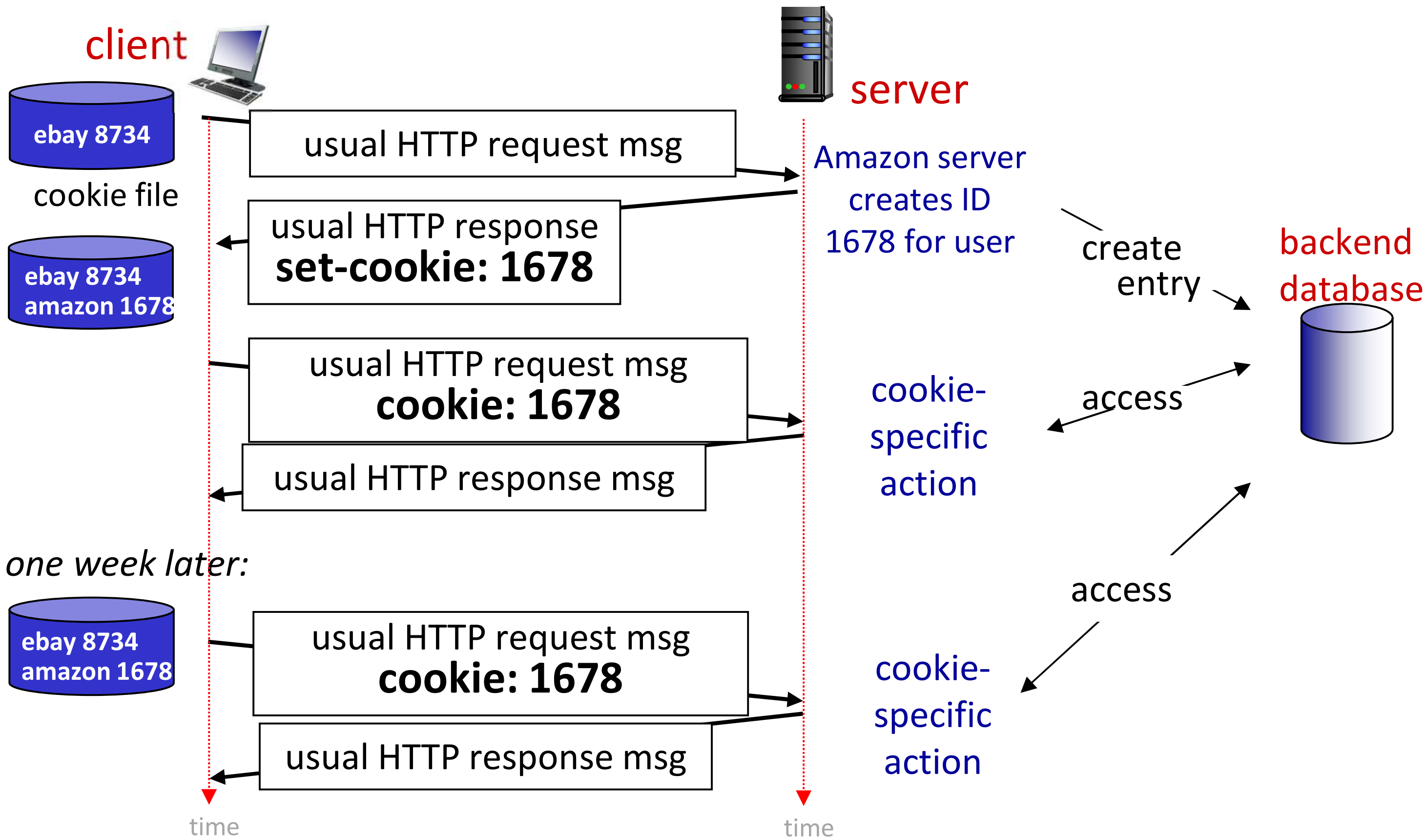
four components:

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

Example:

- Susan uses browser on laptop, visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID (aka "cookie")
  - entry in backend database for ID
- subsequent HTTP requests from Susan to this site will contain cookie ID value, allowing site to "identify" Susan

Maintaining user/server state: cookies



HTTP cookies: comments

What cookies can be used for:

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

Challenge: How to keep state:

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

aside

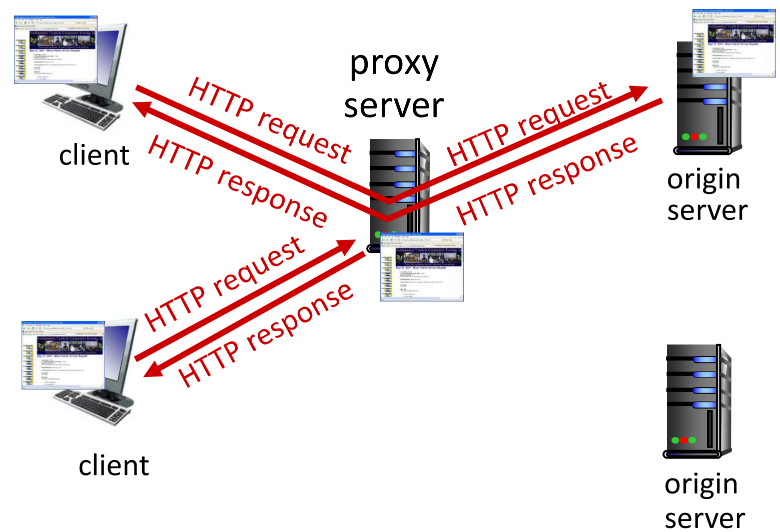
cookies and privacy:

- cookies permit sites to *learn* a lot about you on their site.
- third party persistent cookies (tracking cookies) allow common identity (cookie value) to be tracked across multiple web sites

Web caches (proxy servers)

Goal: satisfy client request without involving origin server

- user configures browser to point to a *Web cache*
- browser sends all HTTP requests to cache
  - *if* object in cache: cache returns object to client
  - *else* cache requests object from origin server, caches received object, then returns object to client



Web caches (proxy servers)

- Web cache acts as both client and server
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)

Why Web caching?

- reduce response time for client request
  - cache is closer to client
- reduce traffic on an institution's access link
- Internet is dense with caches
  - enables "poor" content providers to more effectively deliver content

Caching example

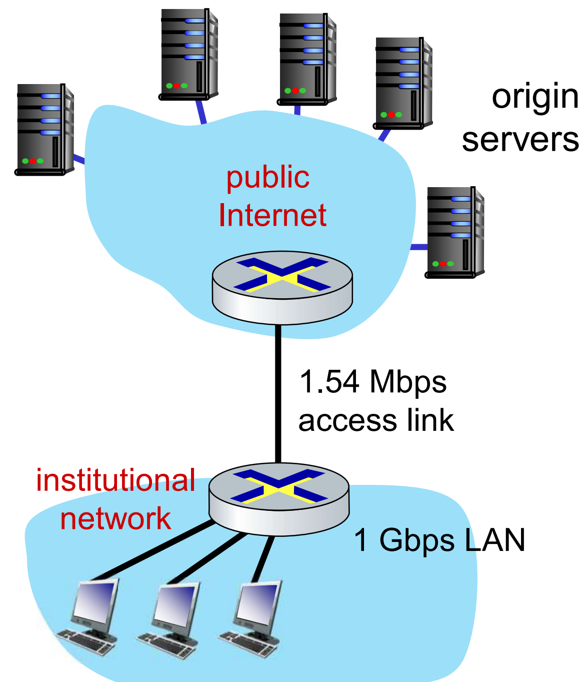
Scenario:

- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- Web object size: 100K bits
- Average request rate from browsers to origin servers: 15/sec
  - average data rate to browsers: 1.50 Mbps

Performance:

- LAN utilization: .0015
- access link utilization = .97
- end-end delay = Internet delay + access link delay + LAN delay
  = 2 sec + minutes + usecs

problem: large delays at high utilization!



Caching example: buy a faster access link

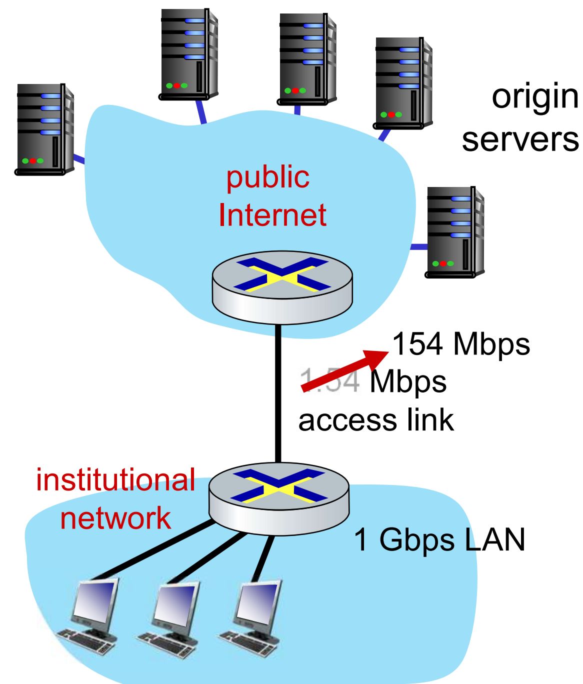
Scenario:

- access link rate: ~~1.54~~ 154 Mbps
- RTT from institutional router to server: 2 sec
- Web object size: 100K bits
- Avg request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Performance:

- LAN utilization: .0015
- access link utilization = ~~.97~~ → .0097
- end-end delay = Internet delay + access link delay + LAN delay
  = 2 sec + ~~minutes~~ msecs + usecs

Cost: faster access link (expensive!)



Caching example: install a web cache

Scenario:

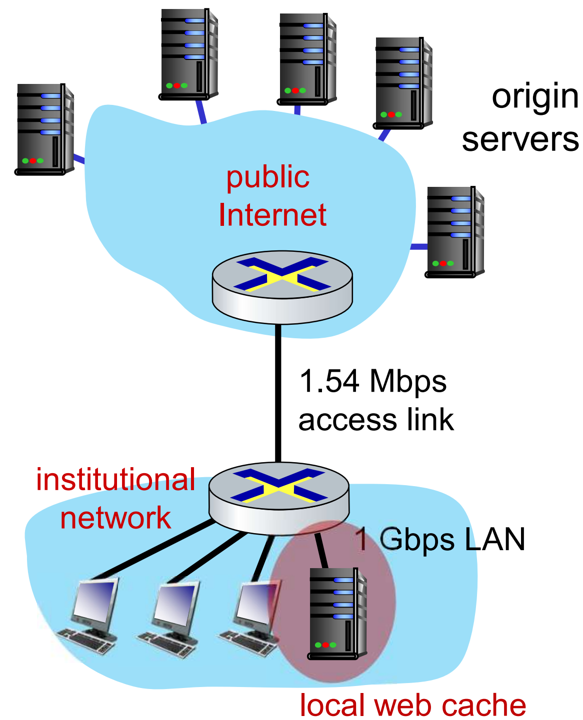
- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- Web object size: 100K bits
- Avg request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Performance:

- LAN utilization: .?
- access link utilization = ?
- average end-end delay = ?

How to compute link utilization, delay?

Cost: web cache (cheap!)



Caching example: install a web cache

Calculating access link utilization, end-end delay with cache:

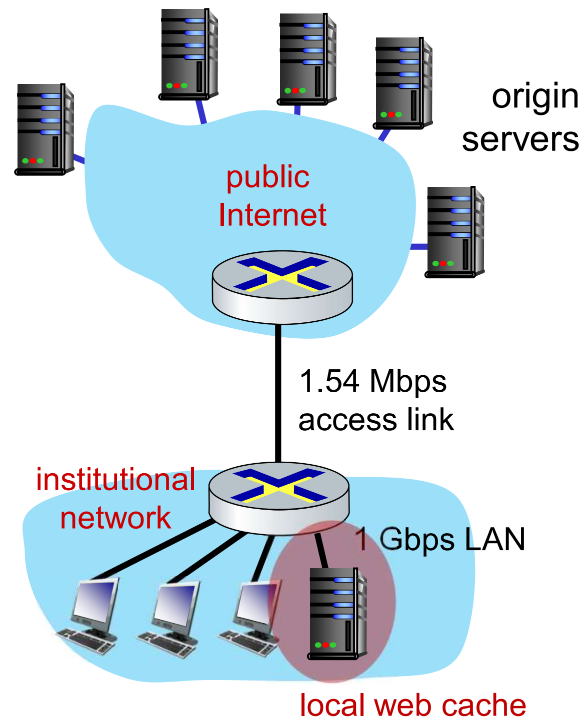
- suppose cache hit rate is 0.4: 40% requests satisfied at cache, 60% requests satisfied at origin
- access link: 60% of requests use access link
- data rate to browsers over access link

  = 0.6 * 1.50 Mbps = .9 Mbps
- utilization = 0.9/1.54 = .58
- average end-end delay

  = 0.6 * (delay from origin servers)

  + 0.4 * (delay when satisfied at cache)

  = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs



lower average end-end delay than with 154 Mbps link (and cheaper too!)

Local cache

- Web pages could be stored in local server (local cache)
- Using local cache for
  - Reading web offline
  - Improve performance in accessing web pages

Conditional GET

Goal: don’t send object if cache has up-to-date cached version

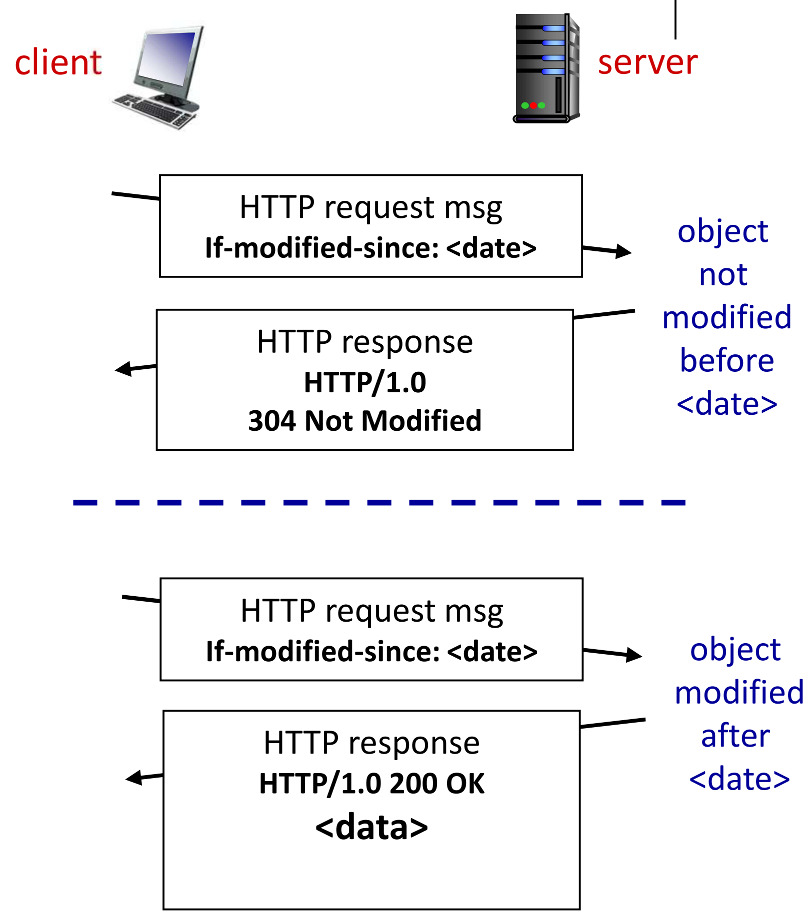
- no object transmission delay
- lower link utilization

- *cache:* specify date of cached copy in HTTP request

  If-modified-since: <date>

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

  HTTP/1.0 304 Not Modified



HTTP/2

Key goal: decreased delay in multi-object HTTP requests

HTTP1.1: introduced multiple, pipelined GETs over single TCP connection

- server responds *in-order* (FCFS: first-come-first-served scheduling) to GET requests
- with FCFS, small object may have to wait for transmission (head-of-line (HOL) blocking) behind large object(s)
- loss recovery (retransmitting lost TCP segments) stalls object transmission

HTTP/2

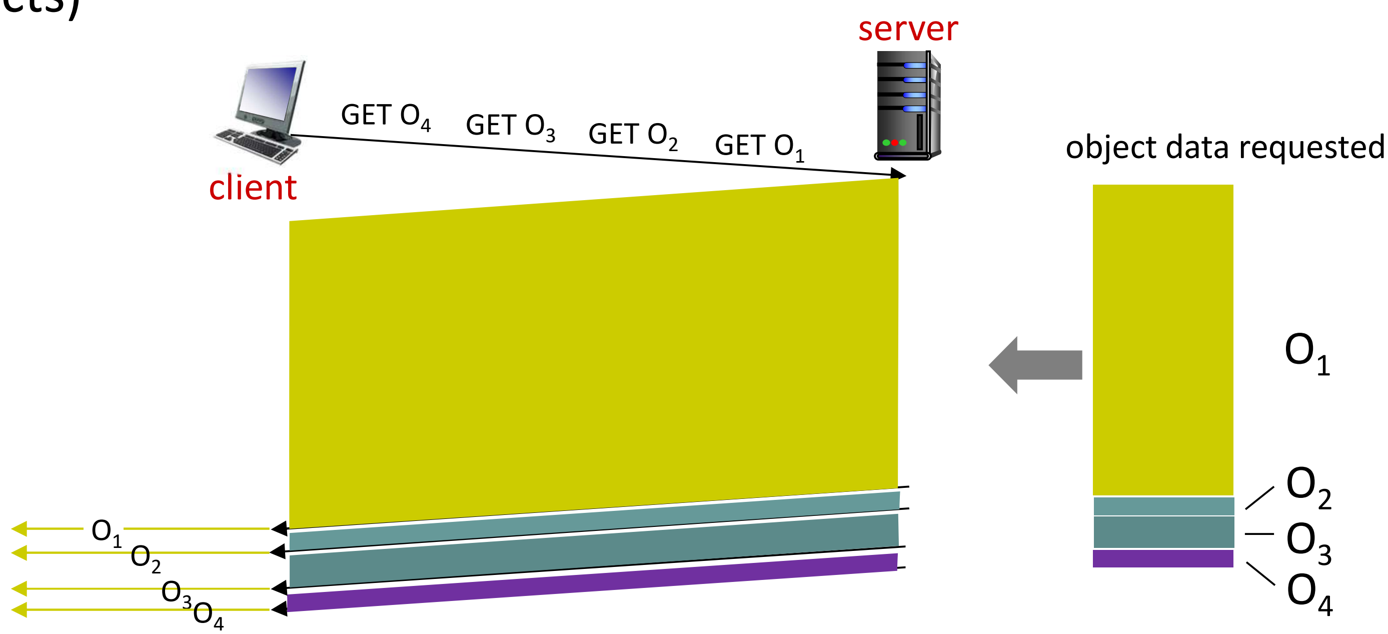
Key goal: decreased delay in multi-object HTTP requests

HTTP/2: [RFC 7540, 2015] increased flexibility at *server* in sending objects to client:

- methods, status codes, most header fields unchanged from HTTP 1.1
- transmission order of requested objects based on client-specified object priority (not necessarily FCFS)
- *push* unrequested objects to client
- divide objects into frames, schedule frames to mitigate HOL blocking

HTTP/2: mitigating HOL blocking

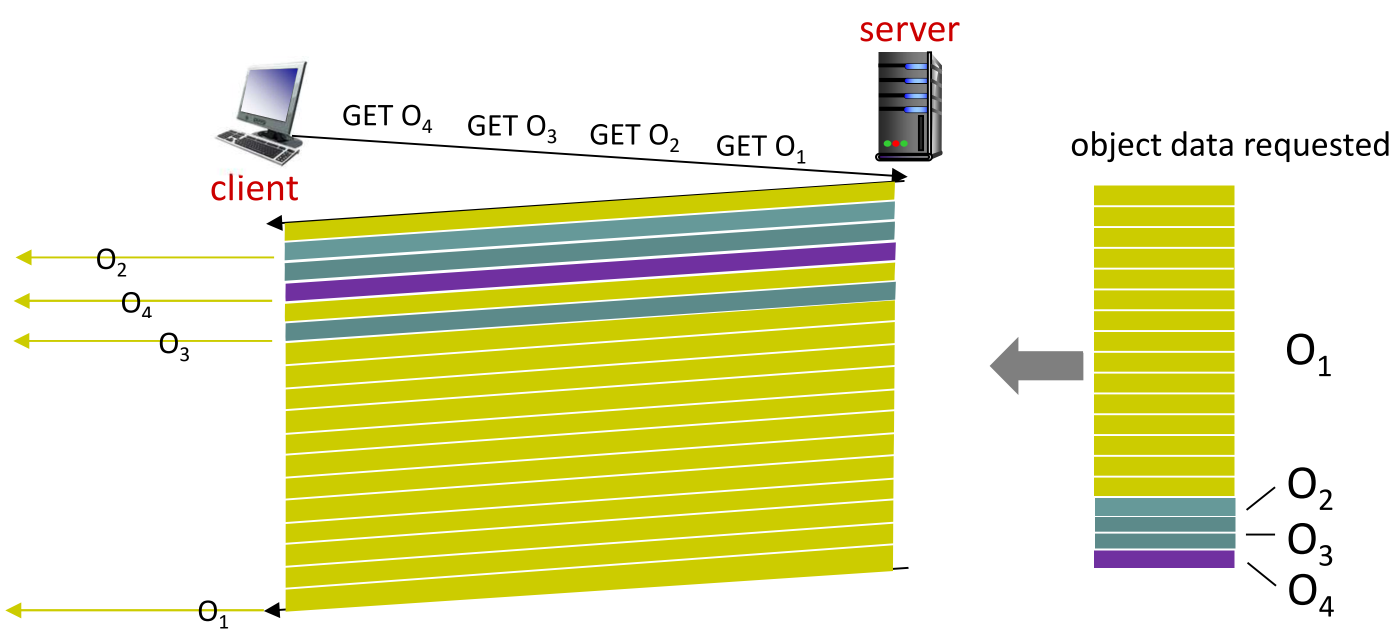
HTTP 1.1: client requests 1 large object (e.g., video file, and 3 smaller objects)



objects delivered in order requested: O_2, O_3, O_4 wait behind O_1

HTTP/2: mitigating HOL blocking

HTTP/2: objects divided into frames, frame transmission interleaved



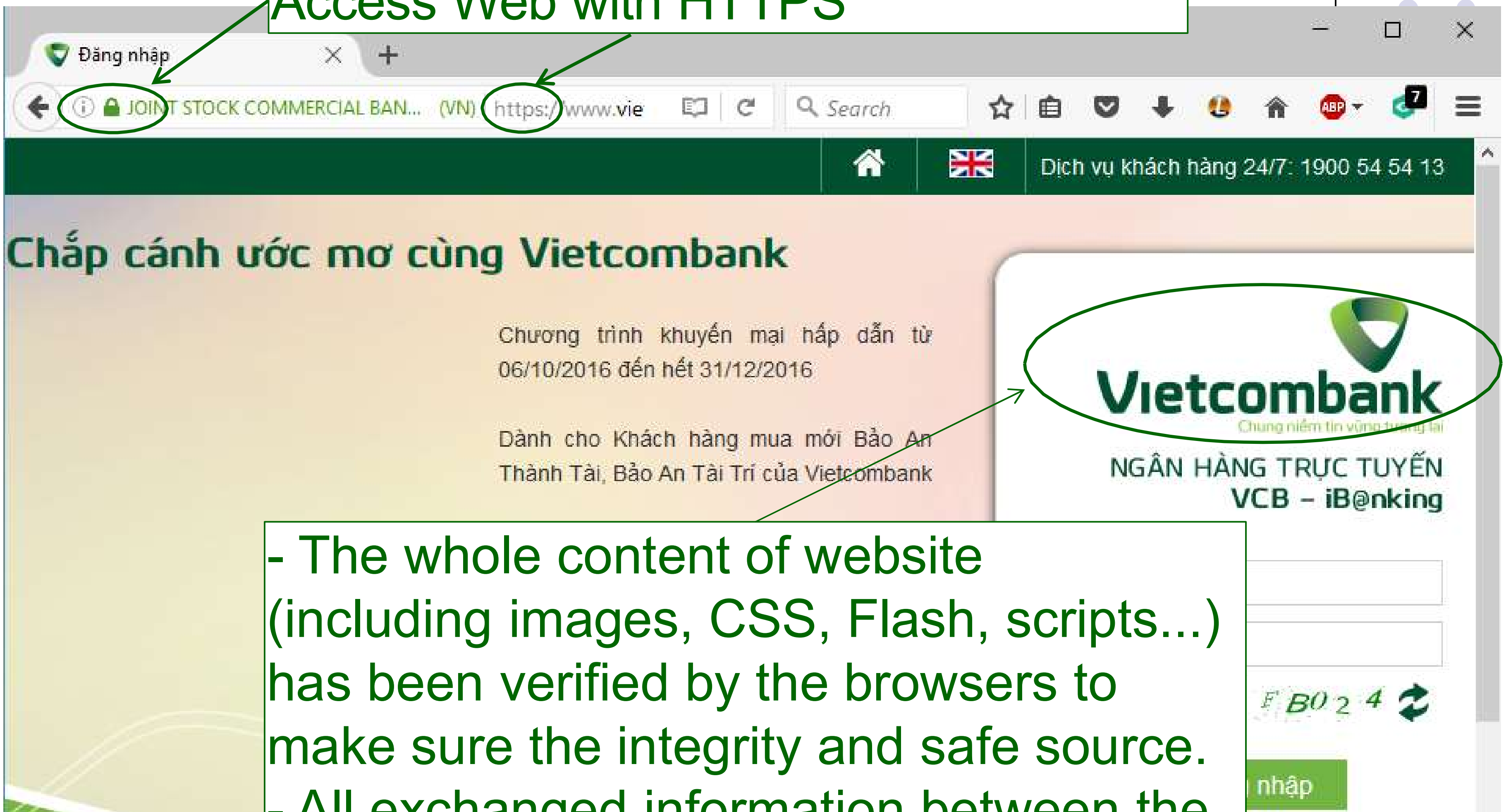
O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

HTTPS

- Limitation of HTTP:
  - No mechanism for users to check the reliability of web server → Không có cơ chế để người dùng kiểm tra tính tin cậy của Web server → security vulnerability for imposters or embed malicious code to HTML
  - No mechanism for data encryption → security vulnerability for attackers to sneak and steal sensitive information
- Secure HTTP: use SSL/TLS instead of TCP to send HTTP messages
  - Authentication:
    - Users can access to the correct website
    - Communication data won't be changed
  - Security: data are kept secretly during data transmission
- Port: 443

HTTPS on the Web

Access Web with HTTPS

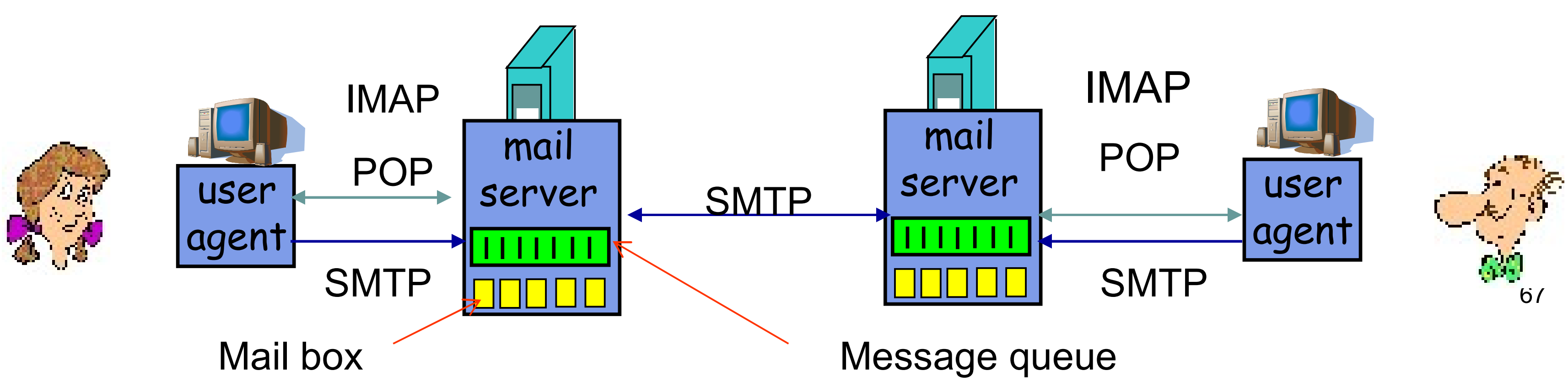


- The whole content of website (including images, CSS, Flash, scripts...) has been verified by the browsers to make sure the integrity and safe source.
- All exchanged information between the broser and Vietcombank is kept secret.

Email

Electronic mail (E-mail)

- MUA (Mail User Agent)
  - Get emails from servers, send emails to servers
  - e.g. Outlook, Thunderbird…
- MTA (Mail Transfer Agent): :
  - Contain the mail boxes of user
  - Queue to send emails
  - e.g. Sendmail, MS Exchange…
- Protocols:
  - Send emails: SMTP-Simple Mail Transfer Protocol
  - Receive emails
    - POP – Post Office Protocol
    - IMAP – Internet Mail Access Protocol



SMTP

- RFC 2821
- TCP, port 25: send emails from client to server and between servers
- Interactive request/response
  - Request: Command with ASCII
  - Response: state code and data

Web Mail

- Use Web browser as MUA
- MUA and MTA exchange information through HTTP
- Mails are stored on servers
- E.g.
  - Gmail,
  - Hotmail,
  - Yahoo! Mail, etc.
- Today, there are many MTA accessible through web interface
  - http://mail.hust.edu.vn
  - http://mail.soict.hust.edu.vn

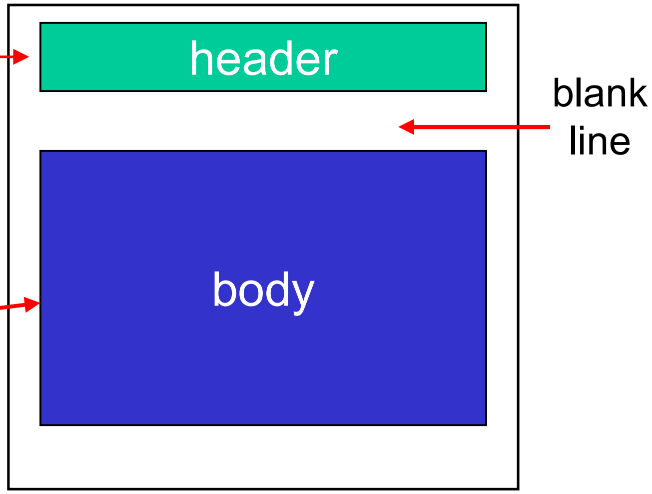
Mail message format

SMTP: protocol for exchanging e-mail messages, defined in RFC 531 (like HTTP)

RFC 822 defines *syntax* for e-mail message itself (like HTML)

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

  these lines, within the body of the email message area different from SMTP MAIL FROM:, RCPT TO: commands!
- Body: the “message” , ASCII characters only



MIME standard

- Represent email content with multimedia data
- MIME: multimedia mail extension, RFC 2045, 2056
- Add one line in the header to specify the sending data type

MIME version

method used
to encode data

multimedia data
type, subtype,
parameter declaration

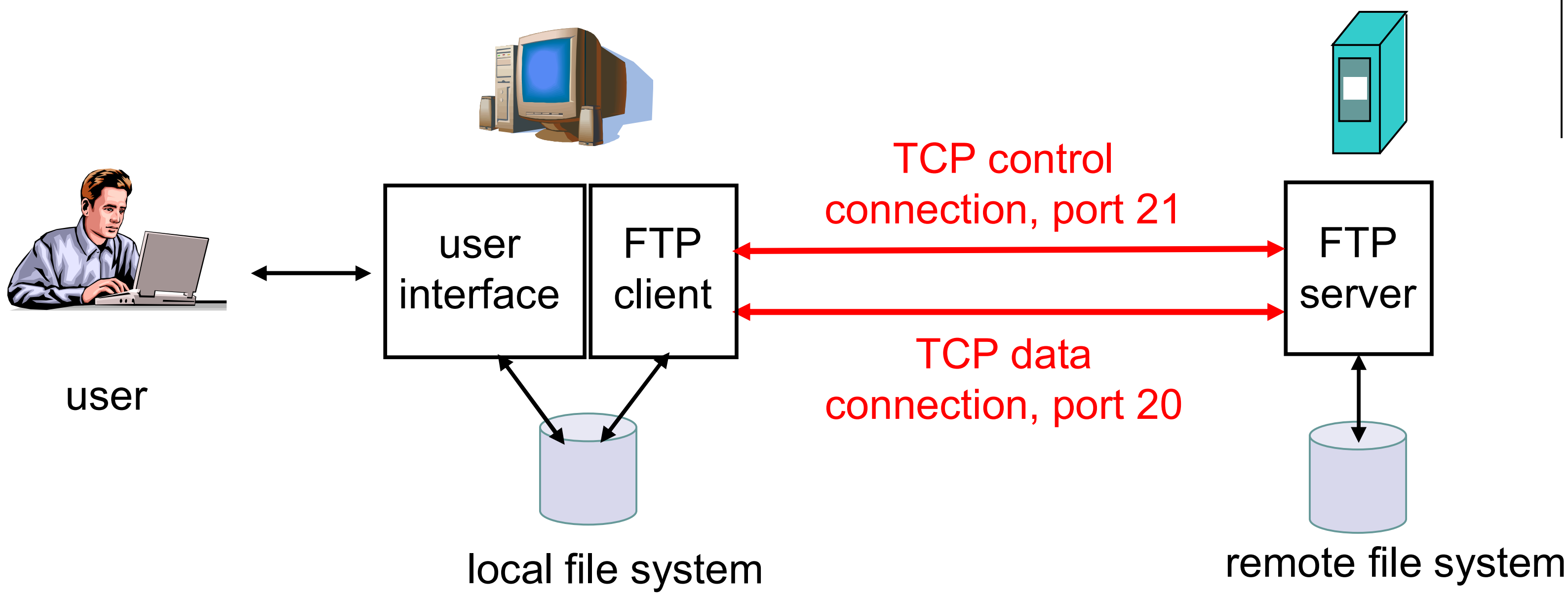
encoded data

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

base64 encoded data .....
....
......base64 encoded data

File Transfer Protocol

FTP: File Transfer Protocol



- Client-server model
- File transfer between two hosts
- RFC 959
- Use TCP, port 20, 21
- Out-of-band control:
  - FTP command : port 21
  - Data: port 20
- Need user to log-in before data transfer
- Some servers allow anonymous user

FTP commands, responses

Sample commands:

- **USER *username***
- **PASS *password***
- **LIST** return list of file in current directory
- **RETR filename** retrieves (gets) file
- **STOR filename** stores (puts) file onto remote host

Sample return codes

- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

FTP client

Command line

C:\Documents and Settings\hongson>ftp
ftp> ?
Commands may be abbreviated. Commands are:

!	**delete**	literal	prompt	send
?	debug	ls	**put**	status
append	dir	mdelete	pwd	trace
ascii	disconnect	mdir	quit	type
bell	**get**	mget	quote	user
binary	glob	mkdir	recv	verbose
bye	hash	mls	remotehelp	
cd	help	mput	rename	
close	lcd	**open**	rmdir	

GUI FTP clients: IE, Firefox, GFTP, ….