

# Mạng máy tính

**TS. Phạm Tuấn Minh**

Khoa Công nghệ Thông tin, Đại học Thủy lợi

[minhpt@tlu.edu.vn](mailto:minhpt@tlu.edu.vn)

<http://netlab.tlu.edu.vn/~minhpt/>

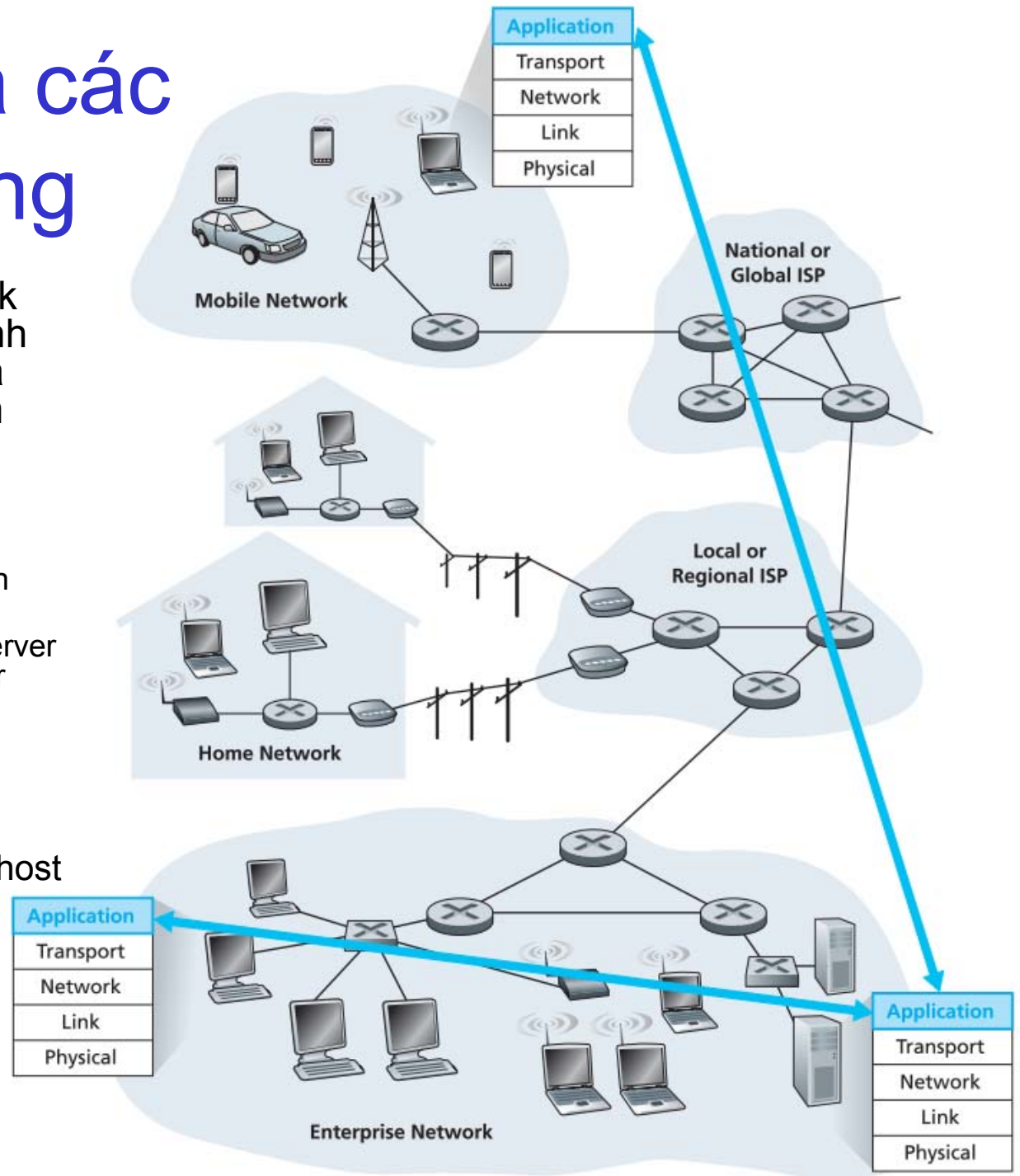


# Chương 2: Tầng ứng dụng

- ❑ Đặc điểm của ứng dụng mạng
- ❑ Web và HTTP
- ❑ Truyền tập tin: FTP
- ❑ Thư điện tử
- ❑ Hệ thống tên miền: DNS
- ❑ Ứng dụng ngang hàng
- ❑ Lập trình socket

# Giao tiếp giữa các ứng dụng mạng

- ứng dụng mạng (network application): chương trình chạy trên **end system** và **giao tiếp** với end system khác qua mạng
- ứng dụng Web
  - hai chương trình
    - trình duyệt chạy trên host của người dùng
    - chương trình Web server chạy trên Web server host
- hệ thống chia sẻ tập tin ngang hàng (P2P file-sharing system):
  - chương trình trên mỗi host tham gia vào hệ thống chia sẻ tập tin



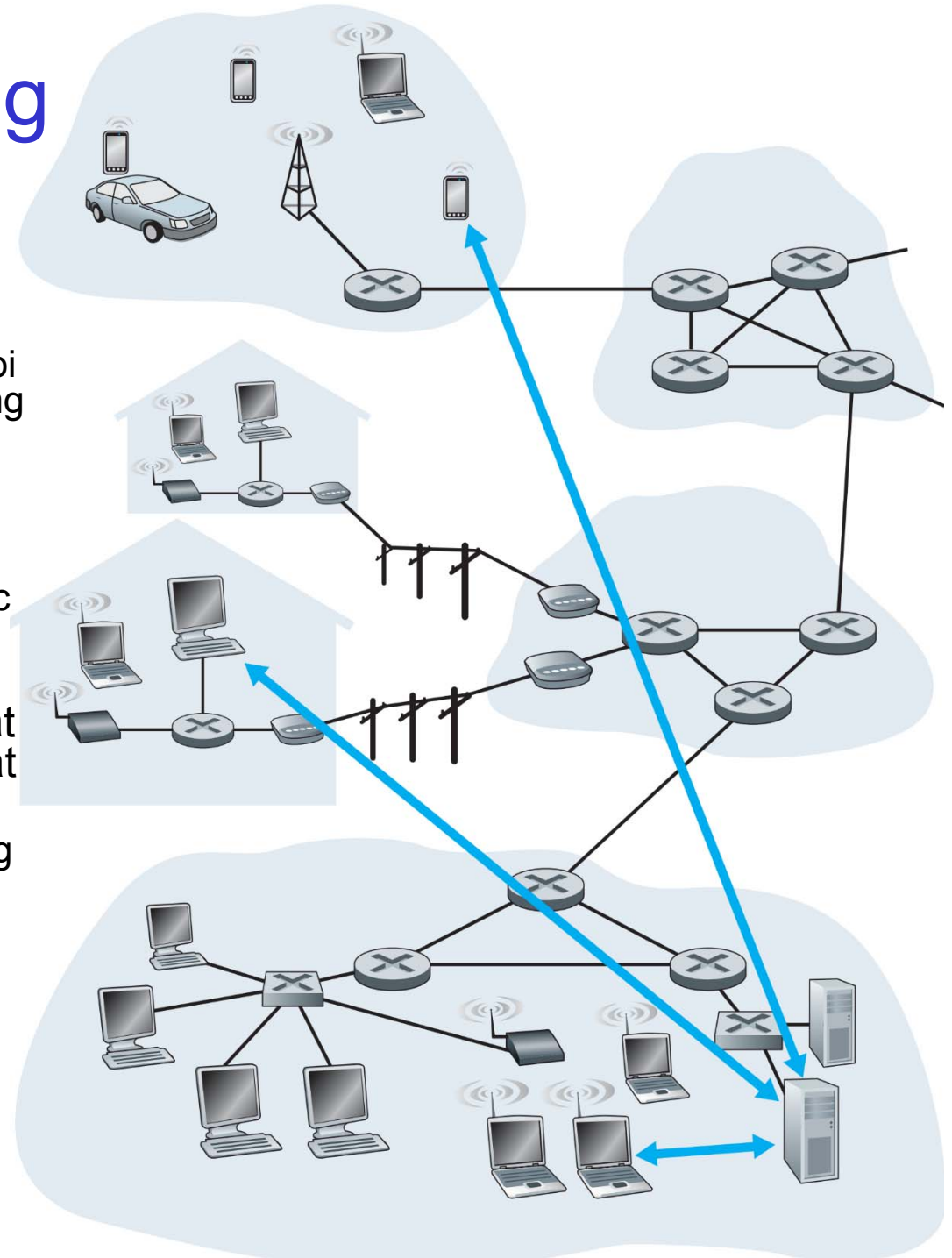
# Kiến trúc ứng dụng mạng

- ❑ kiến trúc ứng dụng và kiến trúc mạng
- ❑ hai kiểu kiến trúc dùng trong ứng dụng mạng:
  - kiến trúc khách chủ (client-server)
  - kiến trúc ngang hàng (peer-to-peer, P2P)



# Kiến trúc ứng dụng mạng

- ❑ kiến trúc client-server
  - có 1 chương trình luôn chạy gọi là server, phục vụ nhiều chương trình chạy trên các host khác, gọi là client
  - các client không trực tiếp giao tiếp với nhau
  - server có địa chỉ cố định và các client biết địa chỉ này
- ❑ trong ứng dụng client-server, server chạy trên 1 host duy nhất thường không đáp ứng được tất cả các yêu cầu từ các client
  - data center, chứa một số lượng lớn host, thường sử dụng để tạo ra 1 virtual server mạnh
- ❑ một số ứng dụng client-server: Web, FTP, Telnet, e-mail

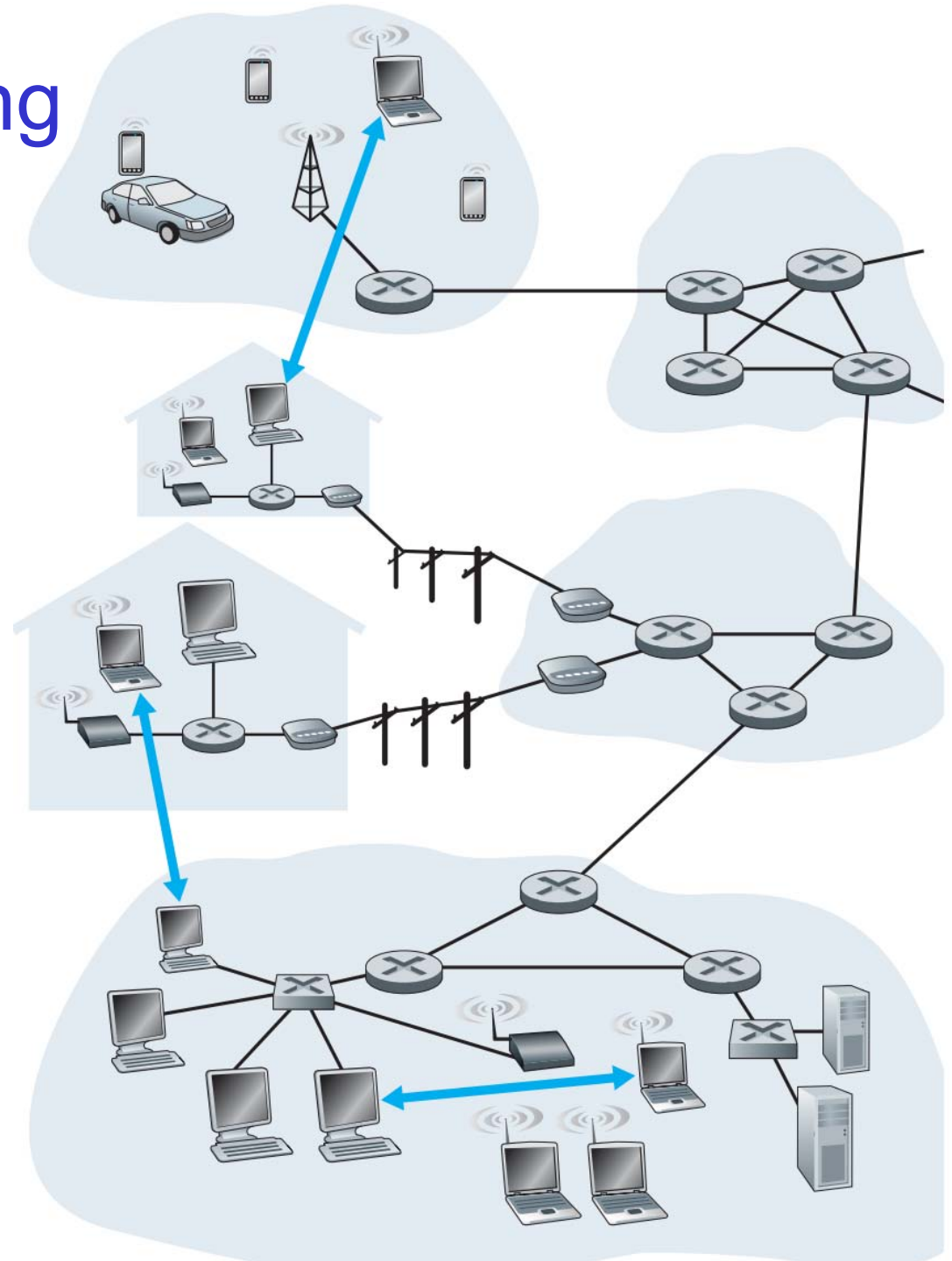


# Kiến trúc ứng dụng mạng

## □ kiến trúc P2P

- phụ thuộc ít hoặc không phụ thuộc vào server dành riêng trong data center
- ứng dụng khai thác trao đổi trực tiếp giữa cặp host khi có kết nối, gọi là peer
- Peer không do có nhà cung cấp dịch vụ làm chủ mà hầu hết là do người sử dụng làm chủ

- ## □ một số ứng dụng dùng kiến trúc: chia sẻ tập tin (như BitTorrent), Internet Telephony (như Skype), IPTV (như Kankan hay PPstream)



# Kiến trúc ứng dụng mạng

## □ Kiến trúc lai

- kết hợp cả client-sever và P2P
- ví dụ các ứng dụng nhắn tin, voip (như skype)



# Kiến trúc ứng dụng mạng

## □ Ưu điểm và thách thức của kiến trúc P2P

### ○ ưu điểm

- khả năng tự tùy biến về quy mô
- hiệu quả chi phí

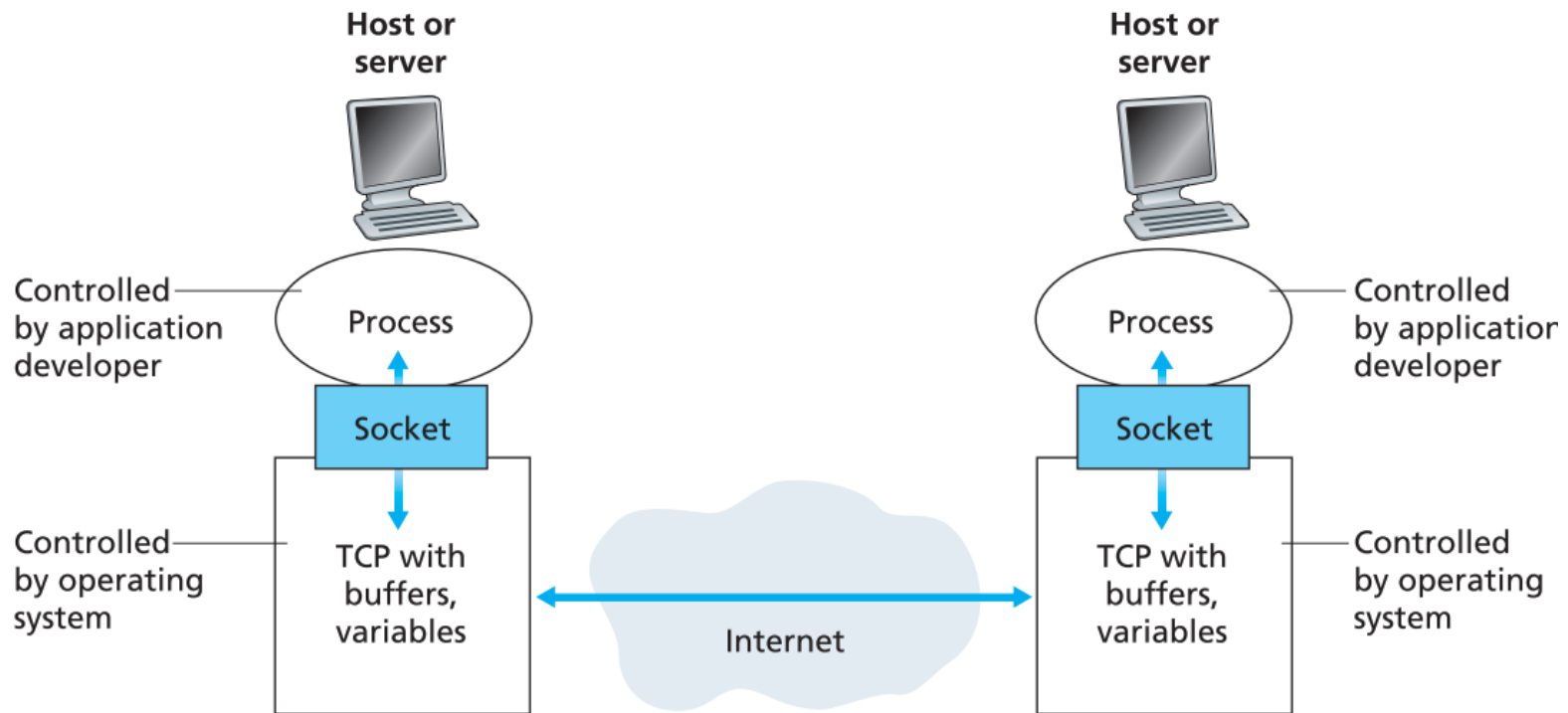
### ○ thách thức

- ISP Friendly
- an toàn bảo mật
- động cơ

# Giao tiếp tiến trình

- ❑ Theo hệ điều hành, tiến trình thực hiện các giao tiếp chứ không phải chương trình
- ❑ Tiến trình client và Server:
  - tiến trình khởi tạo giao tiếp (liên lạc với tiến trình khác vào bắt đầu một phiên) gọi là client
  - tiến trình chờ để tiến trình khác liên lạc thì gọi là server
  - Ví dụ: Trong Web: client? server? Trong ứng dụng chia sẻ tập tin P2P, khi Peer A liên lạc với Peer B để gửi một tập tin: client? server?

# Giao tiếp tiến trình



- Giao diện giữa tiến trình và mạng máy tính
  - Tiến trình gửi bản tin và nhận bản tin từ mạng qua một giao diện phần mềm gọi là socket
  - Socket là giao diện giữa tầng ứng dụng và tầng giao vận trong một host

# Giao tiếp tiến trình

## □ Đánh địa chỉ tiến trình

- để một tiến trình chạy trên một host gửi gói tin tới một tiến trình trên một host khác, tiến trình nhận cần có địa chỉ
  - (1) địa chỉ của host
  - (2) định danh xác định tiến trình nhận trong host đích
- trong Internet
  - host được xác định bởi **IP address**
  - tiến trình nhận (hay socket nhận) chạy trên host đích được xác định bởi **port number**. Tại sao lại cần thông tin này?

# Dịch vụ giao vận cung cấp cho ứng dụng

- ❑ Nhiều mạng, bao gồm Internet, cung cấp nhiều hơn một giao thức tầng giao vận. Khi phát triển ứng dụng, người phát triển phải chọn một trong các giao thức của tầng giao vận. Làm sao để chọn giao thức tầng giao vận?
- ❑ Giao thức tầng giao vận có thể cung cấp các dịch vụ nào cho ứng dụng?
  - truyền dữ liệu tin cậy
  - thông lượng
  - thời gian
  - an toàn bảo mật



# Dịch vụ giao vận cung cấp cho ứng dụng

- ❑ Truyền dữ liệu tin cậy (**reliable data transfer**)
  - Gói tin có thể bị thất lạc trong mạng máy tính. Khi nào?
  - Đối với nhiều ứng dụng, thất lạc dữ liệu có thể gây hậu quả không chấp nhận được. Ứng dụng nào? Tại sao?
  - Dịch vụ truyền dữ liệu đảm bảo (tin cậy): có cơ chế để đảm bảo dữ liệu được chuyển đúng và đầy đủ tới ứng dụng nhận
  - Khi giao thức tầng giao vận không cung cấp dịch vụ truyền tin cậy, dữ liệu gửi bởi tiến trình gửi có thể không tới tiến trình nhận. Điều này có thể chấp nhận được đối với các ứng dụng chấp nhận mất tin (**loss-tolerant application**). Ví dụ loss-tolerant application?

# Dịch vụ giao vận cung cấp cho ứng dụng

## □ Thông lượng

- Trong một phiên truyền thông giữa hai tiến trình dọc một đường đi trong mạng, thông lượng là tốc độ mà tiến trình gửi có thể chuyển các bit tới tiến trình nhận.
- Thông lượng khả dụng có thể thay đổi theo thời gian. Tại sao?
- Tầng giao vận có thể cung cấp dịch vụ đảm bảo thông lượng khả dụng
  - ứng dụng có thể yêu cầu thông lượng khả dụng  $r$  bit/giây và tầng giao vận sẽ đảm bảo rằng thông lượng khả dụng luôn thấp nhất là  $r$  bit/giây
- Ứng dụng cần đảm bảo yêu cầu thông lượng gọi là ứng dụng tác động bởi băng thông (**bandwidth-sensitive applications**)
  - Ví dụ, ứng dụng điện thoại Internet cần 32 kbps
- **Elastic application** có thể sử dụng băng thông ít hoặc nhiều tùy theo thông lượng khả dụng. Ví dụ elastic apps?

# Dịch vụ giao vận cung cấp cho ứng dụng

## □ Thời gian

- Đảm bảo thời gian: ví dụ mỗi bit gửi từ socket gửi tới socket nhận không chậm hơn 10msec
- Dịch vụ kiểu này cần cho các ứng dụng thời gian thực có tính tương tác, ví dụ Internet telephony, virtual environment, teleconferencing, và multiplayer game
- Trong trò chơi nhiều người chơi hoặc môi trường tương tác ảo, độ trễ lớn giữa hành động và thể hiện được hành động đó trong môi trường (ví dụ, người chơi khác quan sát thấy) sẽ làm giảm tính thực tế của ứng dụng

# Dịch vụ giao vận cung cấp cho ứng dụng

## ❑ An toàn bảo mật

- Giao thức tầng giao vận có thể cung cấp một hoặc nhiều dịch vụ an toàn bảo mật
- Ví dụ, tại nút gửi, giao thức tầng giao vận có thể mã hóa dữ liệu mà tiến trình gửi truyền đi, và tại nút nhận, giao thức tầng giao vận có thể giải mã dữ liệu trước khi chuyển tới tiến trình nhận
- Những dịch vụ như vậy có thể cung cấp tính bí mật giữa hai tiến trình, thậm chí nếu dữ liệu bị quan sát khi truyền từ tiến trình gửi tới tiến trình nhận
- Giao thức tầng giao vận có thể cung cấp các dịch vụ an toàn bảo mật khác ngoài tính bí mật, ví dụ như tính toàn vẹn dữ liệu, tính xác thực

# Dịch vụ tầng giao vận của Internet

- ❑ Internet có hai giao thức tầng giao vận là UDP và TCP
- ❑ Ứng dụng phải lựa chọn việc sử dụng UDP hay TCP



# Dịch vụ tầng giao vận của Internet

## □ Yêu cầu dịch vụ của một số ứng dụng

Application	Data Loss	Throughput	Time-Sensitive
File transfer/download	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	No loss	Elastic (few kbps)	No
Internet telephony/ Video conferencing	Loss-tolerant	Audio: few kbps–1Mbps Video: 10 kbps–5 Mbps	Yes: 100s of msec
Streaming stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps–10 kbps	Yes: 100s of msec
Instant messaging	No loss	Elastic	Yes and no

# Dịch vụ tầng giao vận của Internet

- ❑ Dịch vụ TCP: Mô hình dịch vụ TCP bao gồm dịch vụ hướng kết nối và dịch vụ truyền dữ liệu tin cậy
- ❑ Dịch vụ UDP: UDP là không hướng kết nối. UDP cung cấp dịch vụ truyền dữ liệu không tin cậy
- ❑ Các dịch vụ không được cung cấp bởi các giao thức tầng giao vận của Internet
  - An toàn bảo mật: TCP cung cấp truyền dữ liệu tin cậy giữa hai nút và có thể dễ dàng bổ sung SSL tại tầng ứng dụng để cung cấp dịch vụ an toàn bảo mật
  - Đảm bảo thông lượng và thời gian là dịch vụ không được cung cấp. Có phải các dịch vụ bị ảnh hưởng bởi thời gian như Internet telephony không thể chạy trong Internet?
  - Internet ngày nay có thể cung cấp các dịch vụ chấp nhận được cho các ứng dụng bị ảnh hưởng bởi thời gian nhưng không thể đảm bảo về thông lượng hoặc thời gian

# Dịch vụ tầng giao vận của Internet

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP [RFC 5321]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
File transfer	FTP [RFC 959]	TCP
Streaming multimedia	HTTP (e.g., YouTube)	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary (e.g., Skype)	UDP or TCP

# Giao thức tầng giao vận

- ❑ Các bản tin của tầng giao vận (message) được cấu trúc như thế nào? Vai trò của các trường trong bản tin? Khi nào các tiến trình gửi bản tin?
- ❑ Giao thức tầng giao vận định nghĩa:
  - Kiểu của bản tin, ví dụ bản tin yêu cầu (request message), bản tin trả lời (response message)
  - Cú pháp của các kiểu bản tin, ví dụ các trường thông tin trong bản tin và cách thức phân chia các trường
  - Ngữ nghĩa của các trường: thông tin trong trường này dùng để làm gì
  - Nguyên tắc để xác định một tiến trình gửi và trả lời gói tin khi nào và như thế nào?
- ❑ Một số giao thức tầng ứng dụng được đặc tả trong các RFC:
  - Giao thức tầng ứng dụng Web: HTTP (the HyperText Transfer Protocol [RFC 2616])
- ❑ Nhiều giao thức tầng ứng dụng là có bản quyền sở hữu
  - Giao thức tầng ứng dụng của Skype
- ❑ Phân biệt giữa ứng dụng mạng và giao thức tầng ứng dụng

# Chương 2: Tầng ứng dụng

- ❑ Đặc điểm của ứng dụng mạng
- ❑ Web và HTTP
- ❑ Truyền tập tin: FTP
- ❑ Thư điện tử
- ❑ Hệ thống tên miền: DNS
- ❑ Ứng dụng ngang hàng
- ❑ Lập trình socket

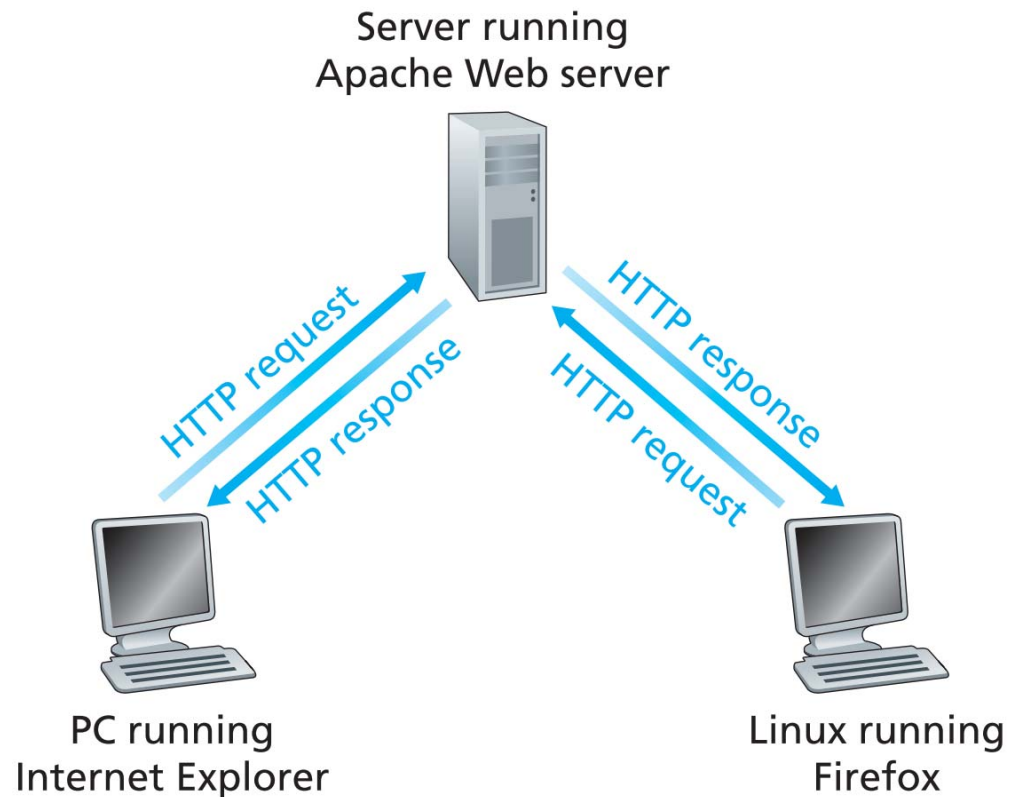


# Web

- ❑ World Wide Web: Berners-Lee 1994
- ❑ Đặc điểm mới thu hút người sử dụng là Web hoạt động theo yêu cầu (*on demand*)
- ❑ Mọi người có thể trở thành người xuất bản với chi phí rất thấp

# Giới thiệu về HTTP

- ❑ HyperText Transfer Protocol (HTTP): Giao thức tầng ứng dụng của Web
  - [RFC 1945] and [RFC 2616]
  - RFC 7540 (May 2015)
- ❑ HTTP sử dụng TCP làm giao thức tầng giao vận
- ❑ HTTP là giao thức không trạng thái (**stateless protocol**)



# Kết nối Non-Persistent và Persistent

## *non-persistent HTTP*

- ❑ một object gửi qua kết nối TCP
  - sau đó kết nối TCP đóng
- ❑ tải nhiều object cần nhiều kết nối

## *persistent HTTP*

- ❑ nhiều objects có thể gửi qua một kết nối TCP giữa client và server

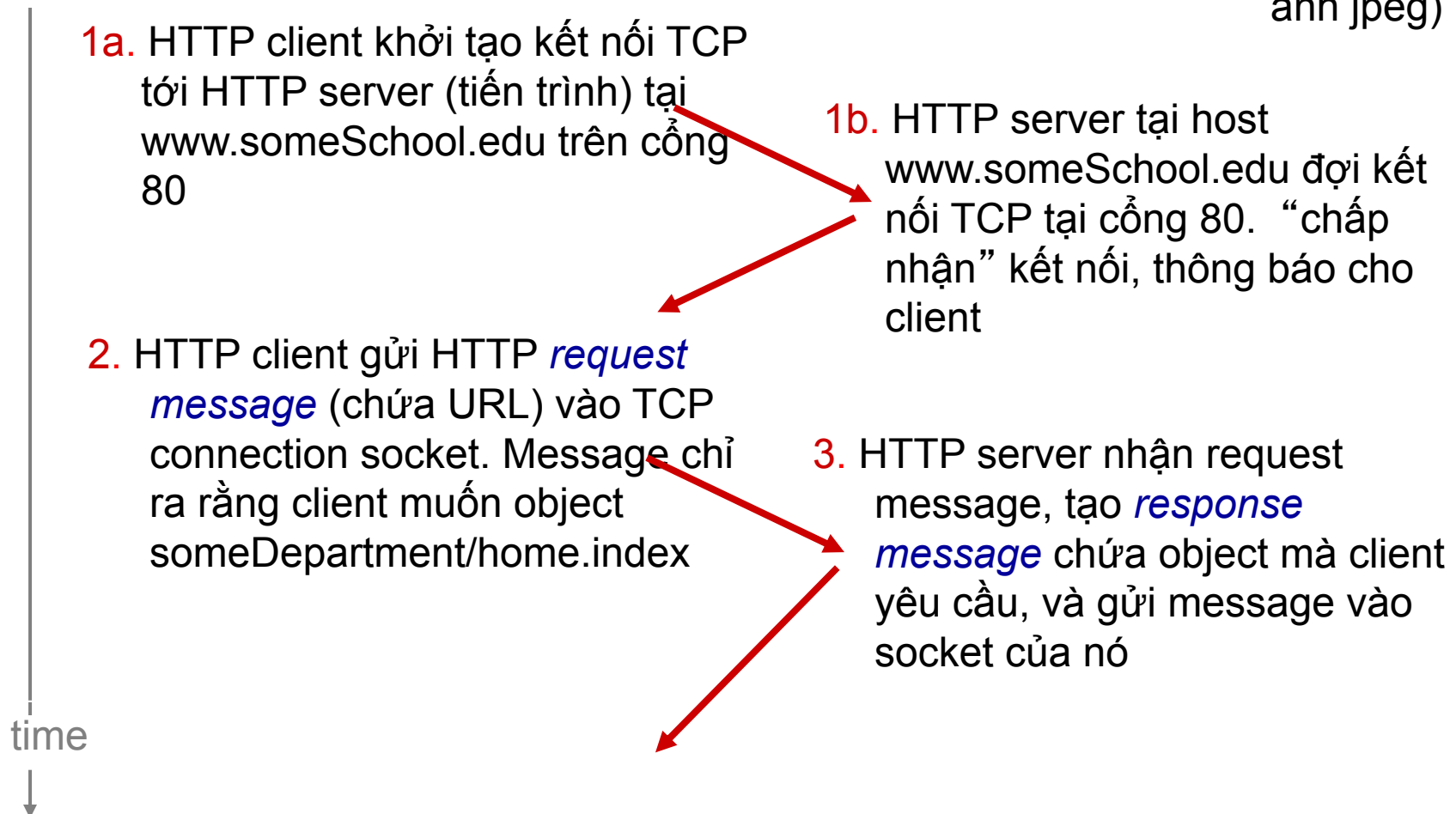
# Kết nối Non-Persistent và Persistent

## □ Non-Persistent HTTP

người dùng gõ URL:

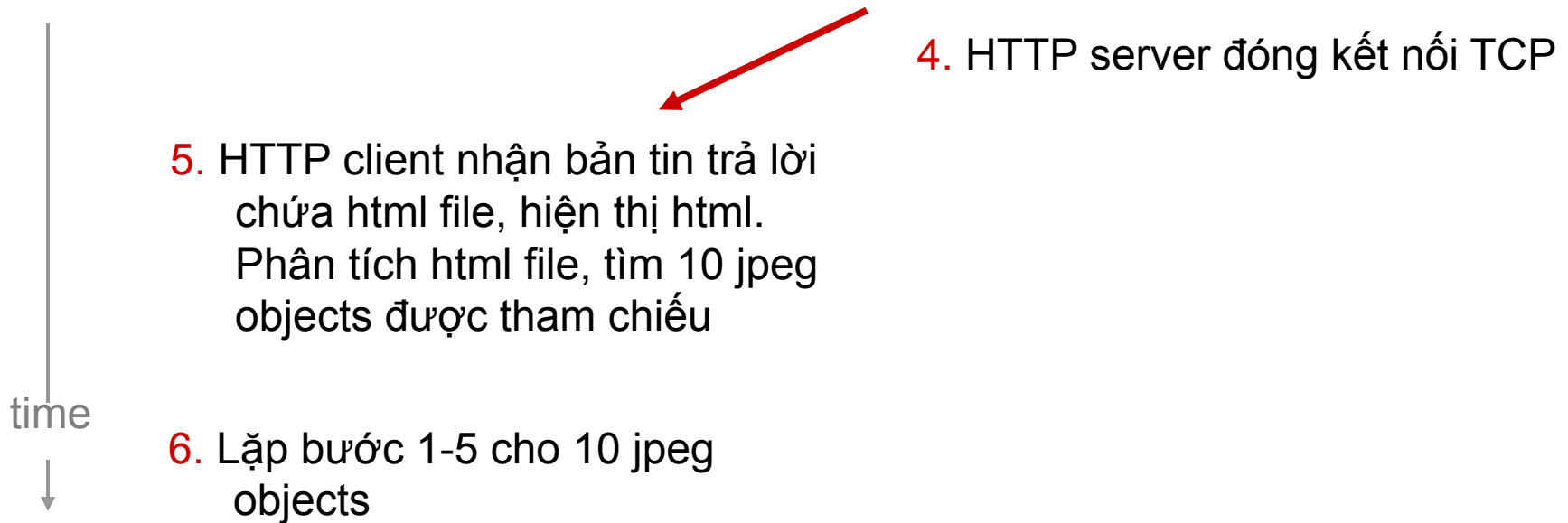
`www.someSchool.edu/someDepartment/home.index`

(chứa văn bản,  
và tham chiếu tới 10  
ảnh jpeg)



# Kết nối Non-Persistent và Persistent

## □ Non-Persistent HTTP





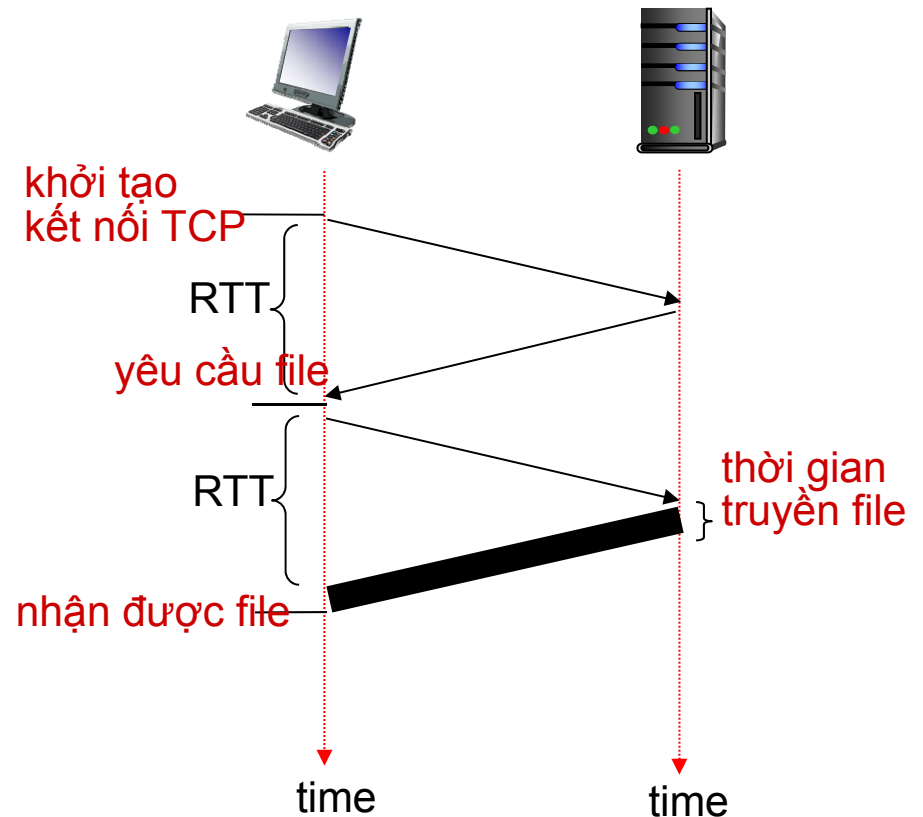
# Kết nối Non-Persistent và Persistent

- ❑ Non-persistent HTTP: thời gian đáp ứng (HTTP response time)

**RTT:** thời gian để một gói tin gửi từ client tới sever rồi từ server tới client

**HTTP response time:**

- ❑ 1 RTT để tạo kết nối TCP
- ❑ 1 RTT cho HTTP request và một số byte đầu tiên của HTTP response tới client
- ❑ thời gian truyền file
- ❑ non-persistent HTTP response time =  
2RTT + thời gian truyền file



# Kết nối Non-Persistent và Persistent

## *non-persistent HTTP:*

- ❑ cần 2 RTT cho 1 object
- ❑ overhead cho mỗi kết nối TCP
- ❑ trình duyệt thường mở đồng thời nhiều kết nối TCP song song để lấy về các object được tham chiếu

## *persistent HTTP:*

- ❑ server giữ kết nối TCP sau khi gửi response
- ❑ các HTTP message sau đó giữa client/server được gửi qua kết nối này
- ❑ client gửi request ngay khi thấy một object được tham chiếu
- ❑ 1 RTT cho tất cả object được tham chiếu

# Cấu trúc HTTP Message:

## HTTP request message

- ❑ 2 kiểu HTTP message: *request, response*
- ❑ HTTP request message:
  - ASCII

The diagram illustrates the structure of an HTTP request message. It shows a sequence of lines: a request line, followed by several header lines, and a final carriage return and line feed. Annotations with arrows point to specific parts of the message:

- request line (GET, POST, HEAD commands)**: Points to the first line: `GET /somedir/page.html HTTP/1.1`
- header lines**: Points to the subsequent lines: `Host: www.someschool.edu`, `Connection: close`, `User-agent: Mozilla/5.0`, and `Accept-language:`
- carriage return, line feed ở đầu dòng chỉ ra kết thúc phần header**: Points to the `\r\n` at the end of the header section.
- carriage return character**: Points to the `\r` in the final `\r\n`.
- line-feed character**: Points to the `\n` in the final `\r\n`.

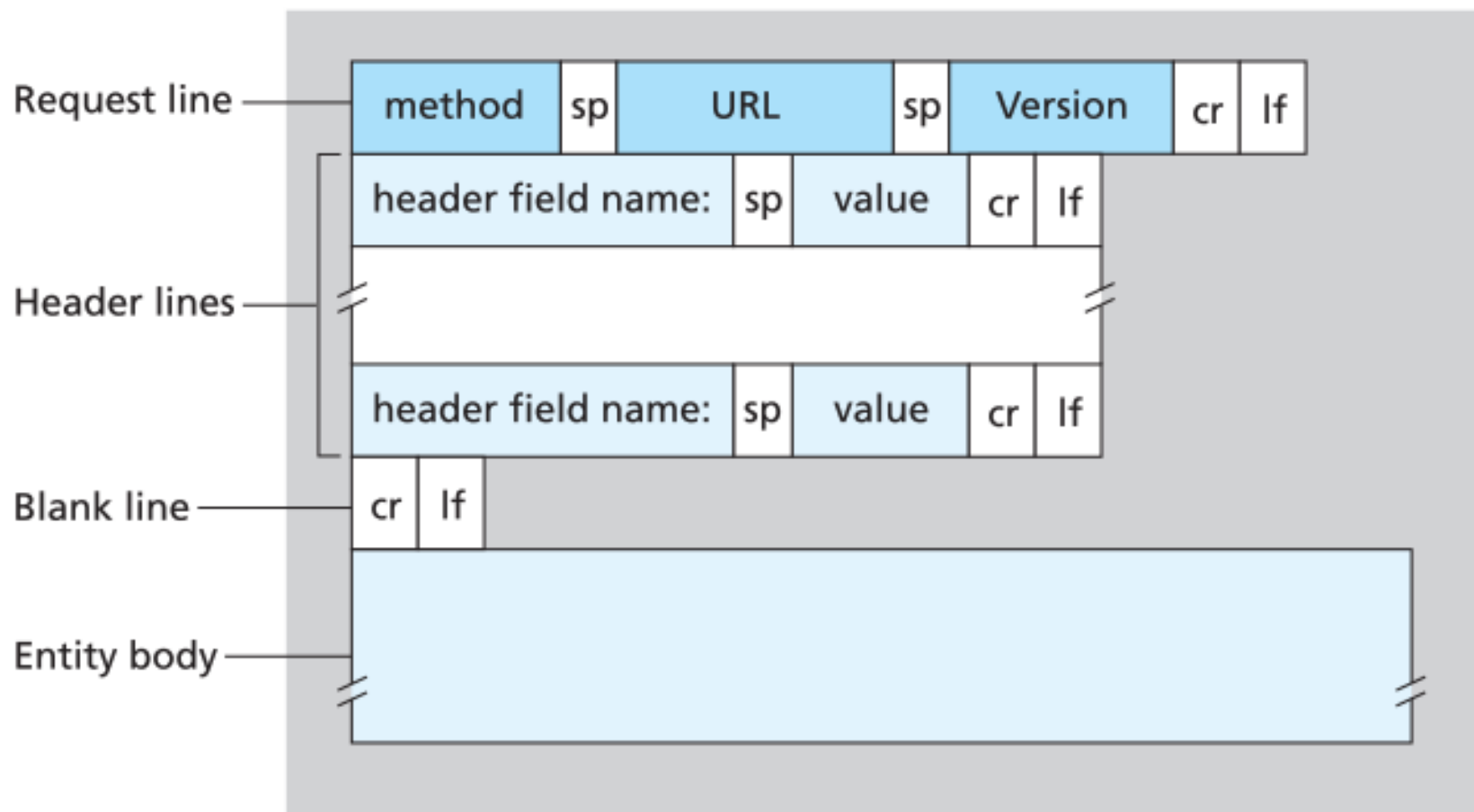
The full message structure shown is:

```
GET /somedir/page.html HTTP/1.1 \r\nHost: www.someschool.edu \r\nConnection: close \r\nUser-agent: Mozilla/5.0 \r\nAccept-language: \r\n\r\n
```

# HTTP Message Format:

## HTTP request message

- General format of an HTTP request message



# Cấu trúc HTTP Message: HTTP response message

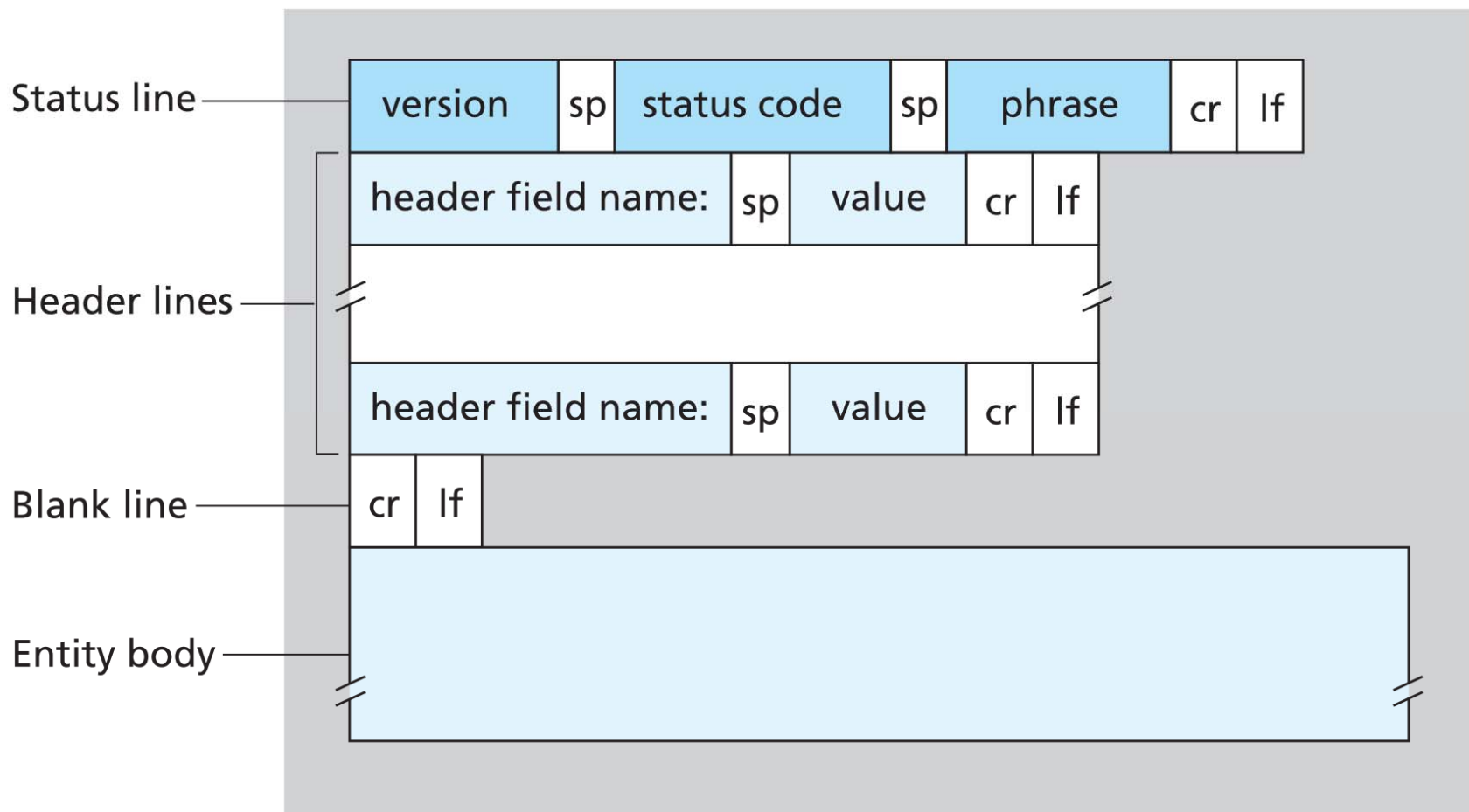
status line  
(protocol  
status code  
status phrase)

header  
lines

```
HTTP/1.1 200 OK \r\n
Connection: close \r\n
Date: Tue, 09 Aug 2011 15:44:04 GMT \r\n
Server: Apache/2.2.3 (CentOS) \r\n
Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT\r\n
Content-Length: 6821 \r\n
Content-Type: text/html \r\n
\r\n
data data data data data ...
```

data, e.g.,  
requested  
HTML file

# Cấu trúc HTTP Message: HTTP response message



# Thực hành HTTP

## 1. Telnet tới Web server:

```
telnet netlab.tlu.edu.vn 80
```

mở kết nối TCP connection tới cổng 80 (cổng mặc định của HTTP server) tại netlab.tlu.edu.vn  
nội dung gõ sẽ gửi tới cổng 80 tại netlab.tlu.edu.vn

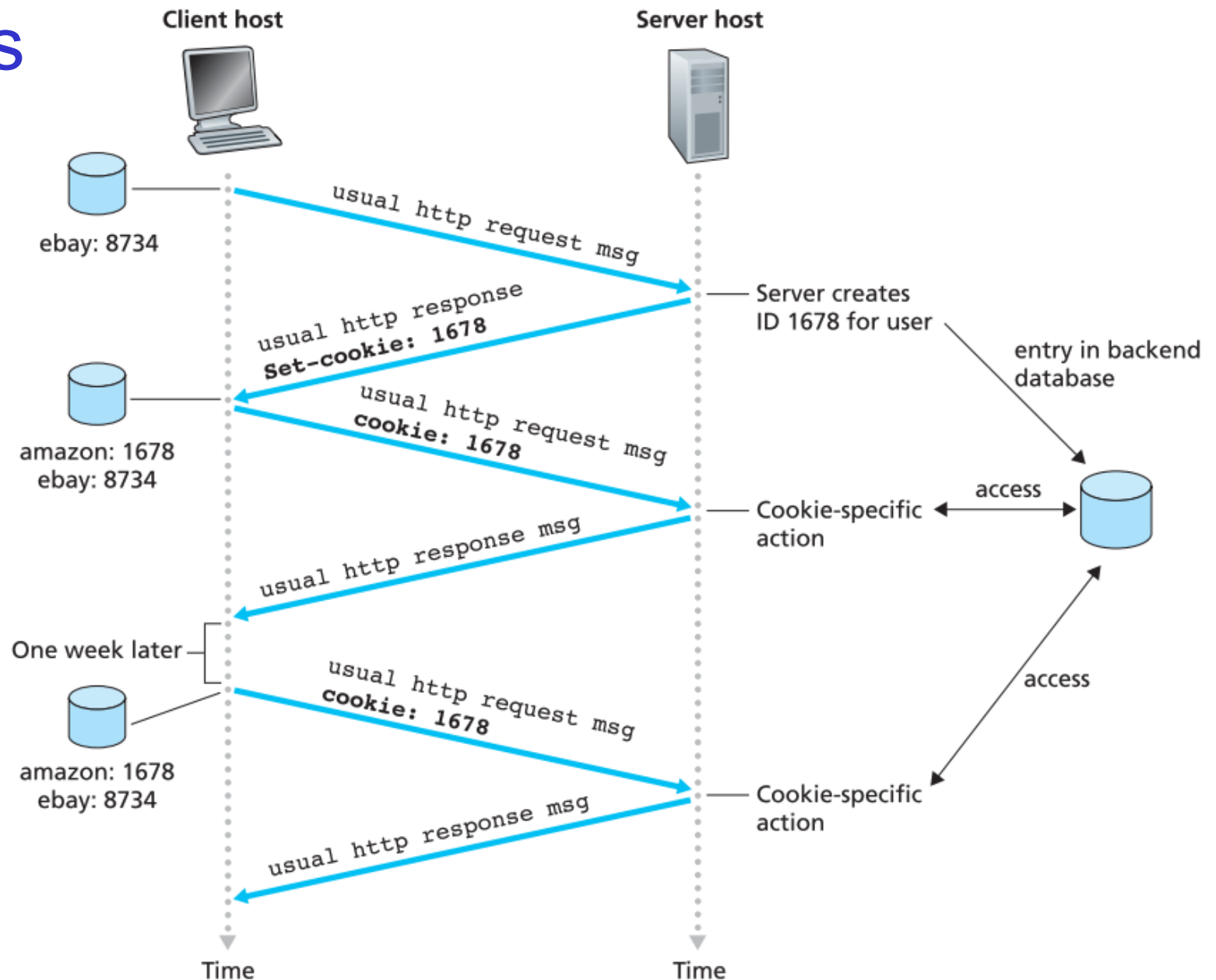
## 2. gõ GET HTTP request:

```
GET /~minhpt/ HTTP/1.1  
Host: netlab.tlu.edu.vn
```

gõ (bấm xuống dòng 2 lần),  
bạn đã gửi  
GET request tới HTTP server

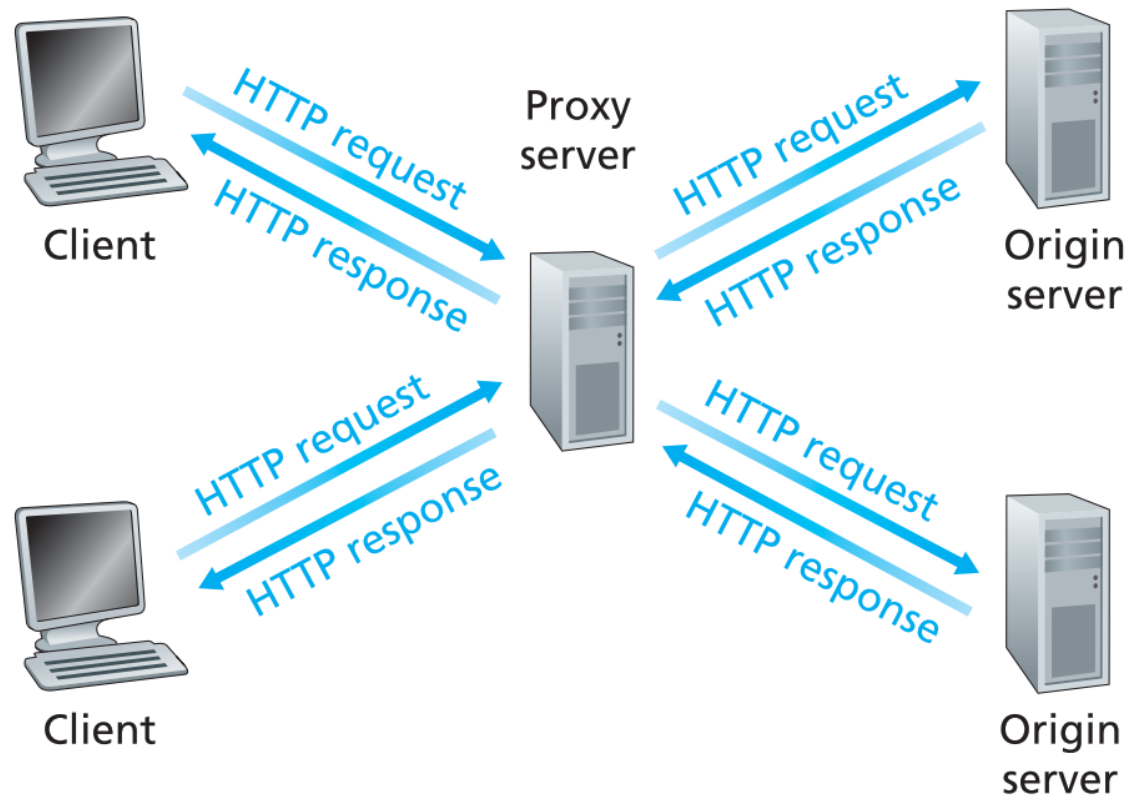
## 3. quan sát response message do HTTP server gửi lại (hoặc sử dụng Wireshark để xem HTTP request/response)

# Tương tác User-Server: Cookies

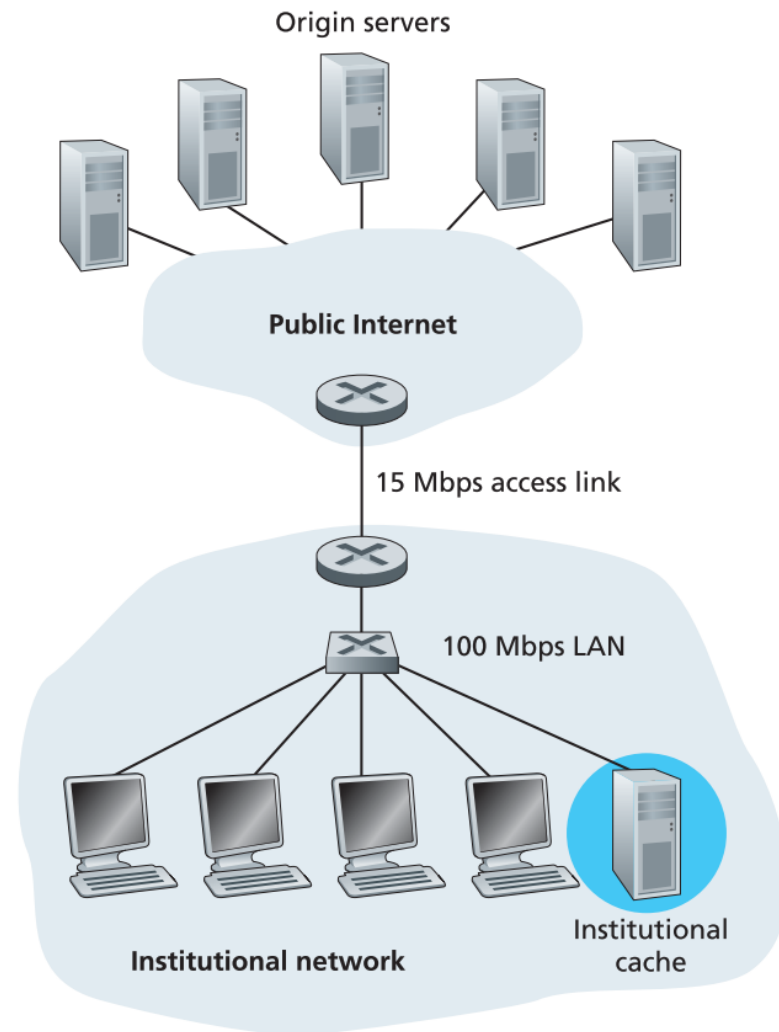
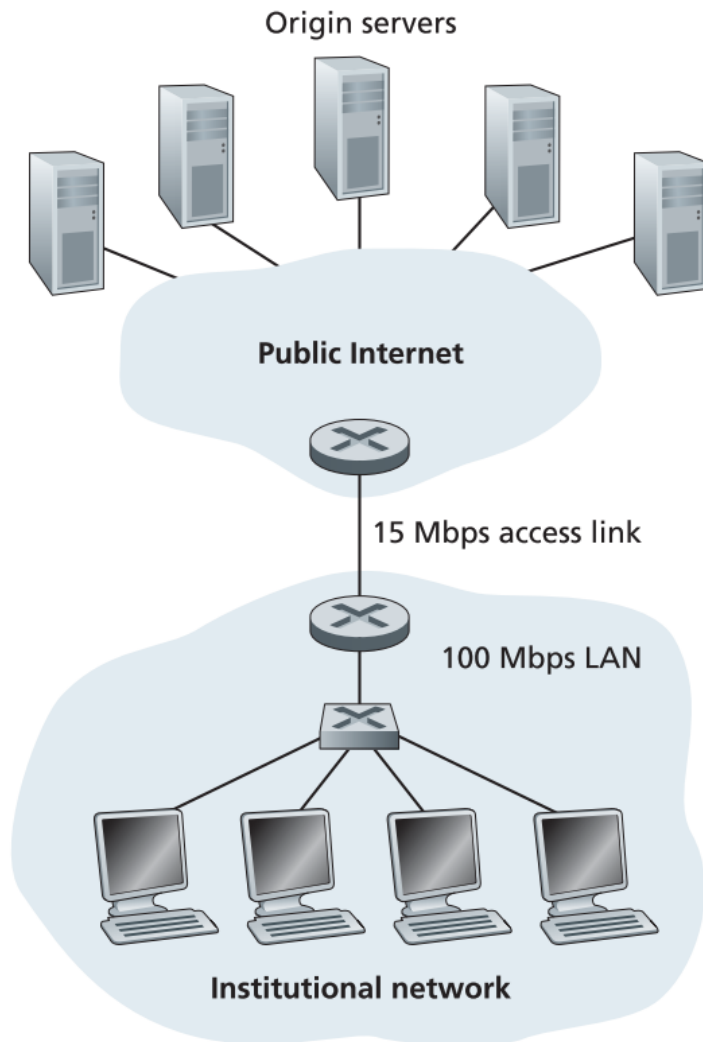




# Web Caching



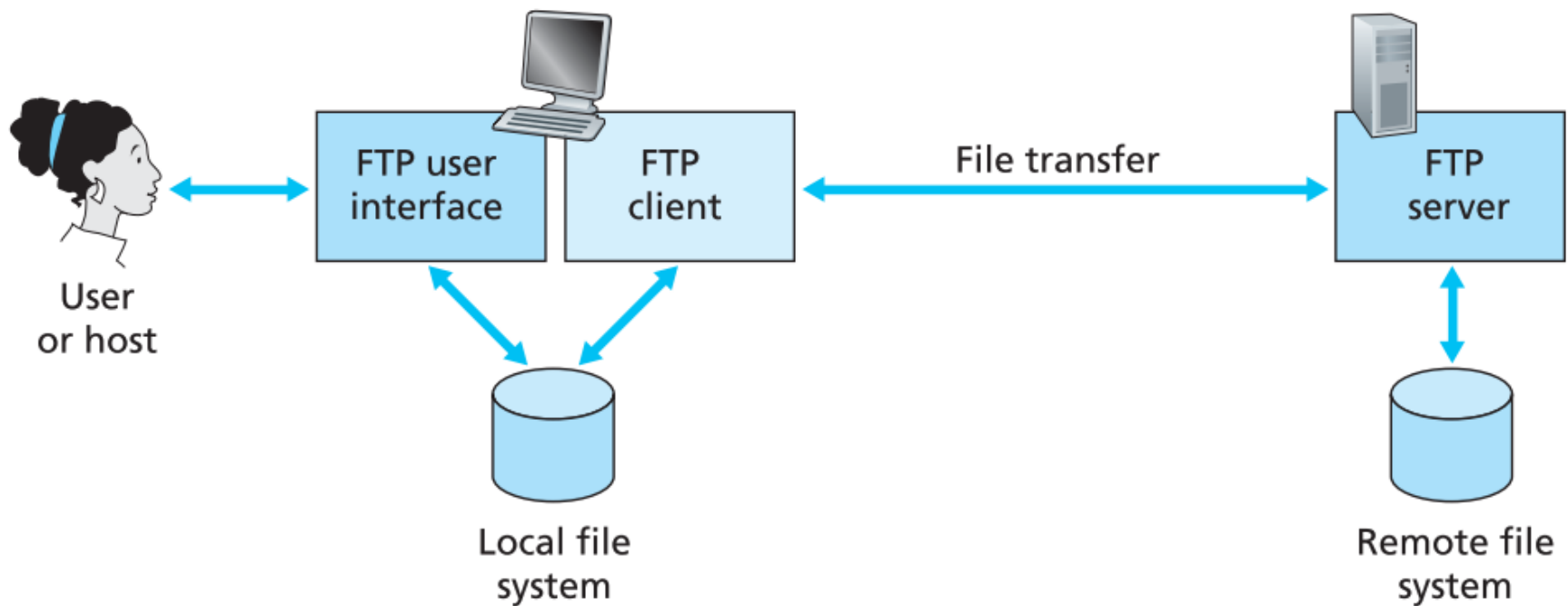
# Web caching: Lợi ích



# Chương 2: Tầng ứng dụng

- ❑ Đặc điểm của ứng dụng mạng
- ❑ Web và HTTP
- ❑ Truyền tập tin: FTP
- ❑ Thư điện tử
- ❑ Hệ thống tên miền: DNS
- ❑ Ứng dụng ngang hàng
- ❑ Lập trình socket

# Truyền tập tin: FTP



# Truyền tập tin: FTP



- ❑ kết nối điều khiển (control connection): dùng riêng
  - HTTP connection: dùng chung
- ❑ FTP server duy trì trạng thái: thư mục hiện tại, xác thực
  - HTTP không duy trì trạng thái

# FTP command, response

## ví dụ command:

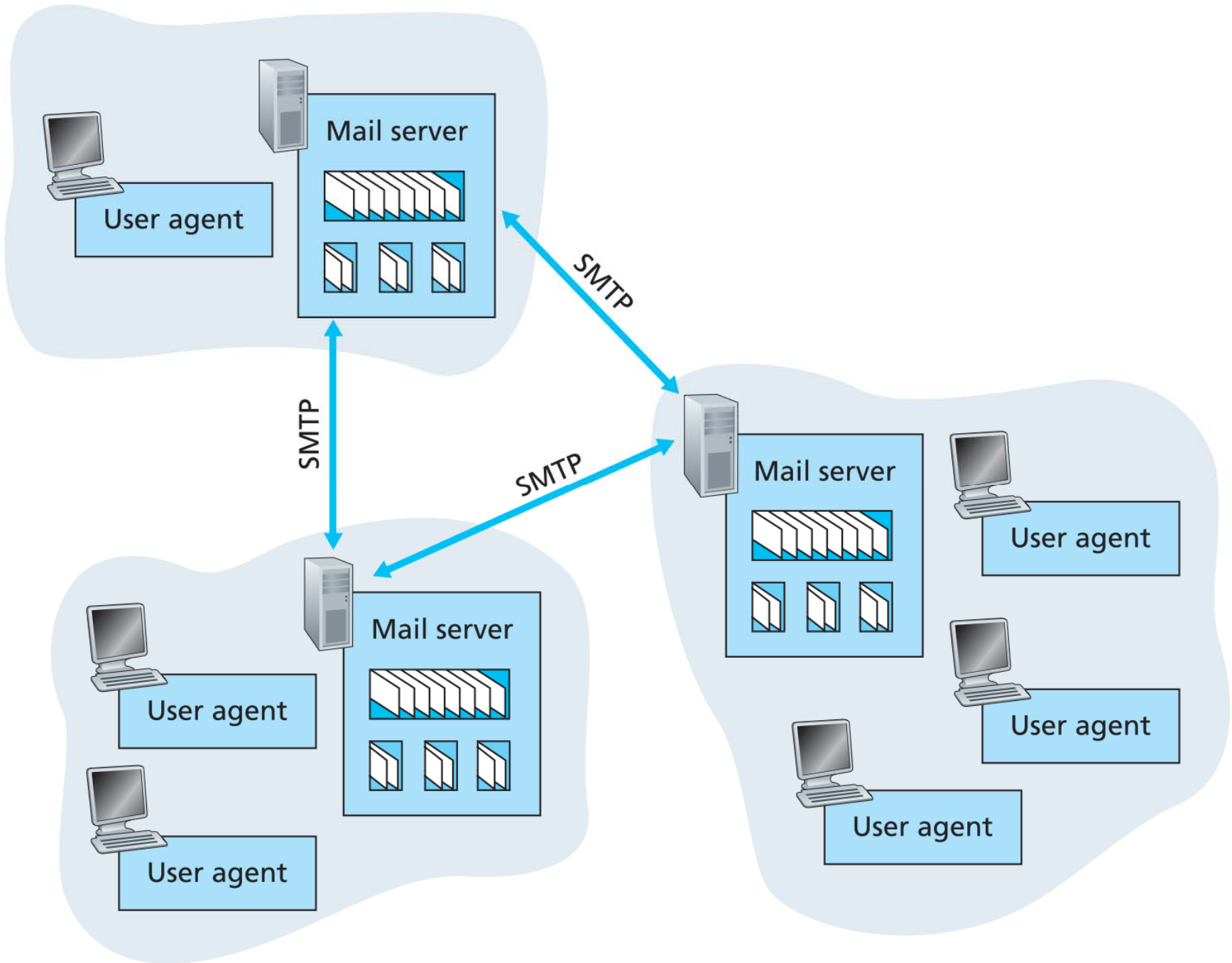
- ❑ gửi dạng ASCII text qua kết nối điều khiển
- ❑ **USER *username***
- ❑ **PASS *password***
- ❑ **LIST** trả về danh sách tập tin trong thư mục hiện tại
- ❑ **RETR *filename*** lấy (get) tập tin
- ❑ **STOR *filename*** lưu (put) tập tin lên remote host

## ví dụ return code

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

## Chương 2: Tầng ứng dụng

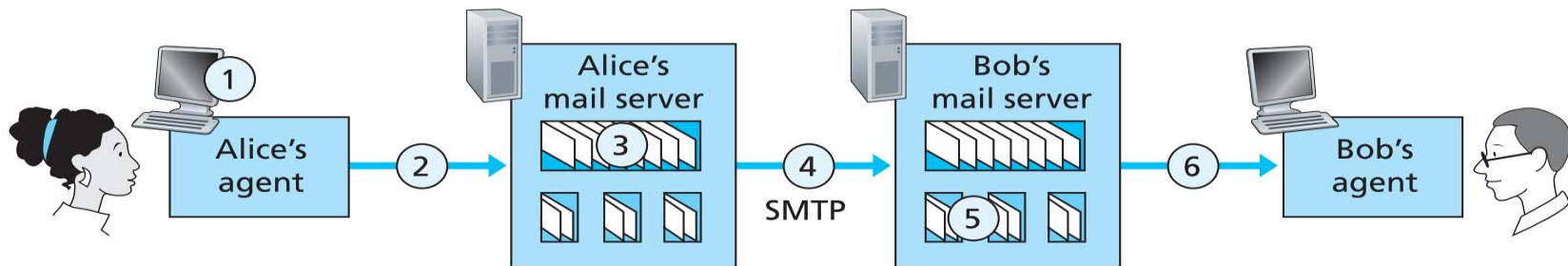
- ❑ Đặc điểm của ứng dụng mạng
- ❑ Web và HTTP
- ❑ Truyền tập tin: FTP
- ❑ **Thư điện tử**
- ❑ Hệ thống tên miền: DNS
- ❑ Ứng dụng ngang hàng
- ❑ Lập trình socket





# SMTP

- ❑ RFC 2821
- ❑ sử dụng TCP để truyền tin cậy bản tin email từ client tới server, cổng 25
- ❑ gửi trực tiếp: server gửi tới server nhận
- ❑ giao tiếp kiểu command/response (giống HTTP, FTP)
  - **commands**: ASCII text
  - **response**: mã trạng thái (status code) và thông điệp (phrase)
- ❑ bản tin phải là 7-bit ASCII



# SMTP

- ❑ SMTP sử dụng persistent connection
- ❑ SMTP yêu cầu bản tin (header & body) là 7-bit ASCII
- ❑ SMTP server sử dụng CRLF.CRLF để xác định kết thúc bản tin

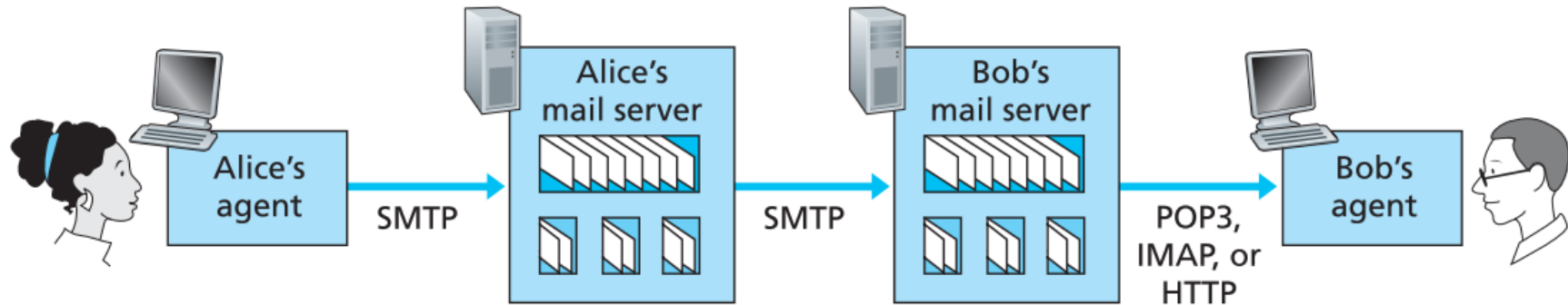
## so sánh với HTTP:

- ❑ HTTP: pull
- ❑ SMTP: push
- ❑ cùng giao tiếp dùng ASCII command/response, status code
- ❑ HTTP: mỗi object chứa trong bản tin riêng
- ❑ SMTP: nhiều object được gửi trong một bản tin có nhiều phần (multipart message)

# Ví dụ giao tiếp SMTP

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

# Giao thức truy cập thư điện tử

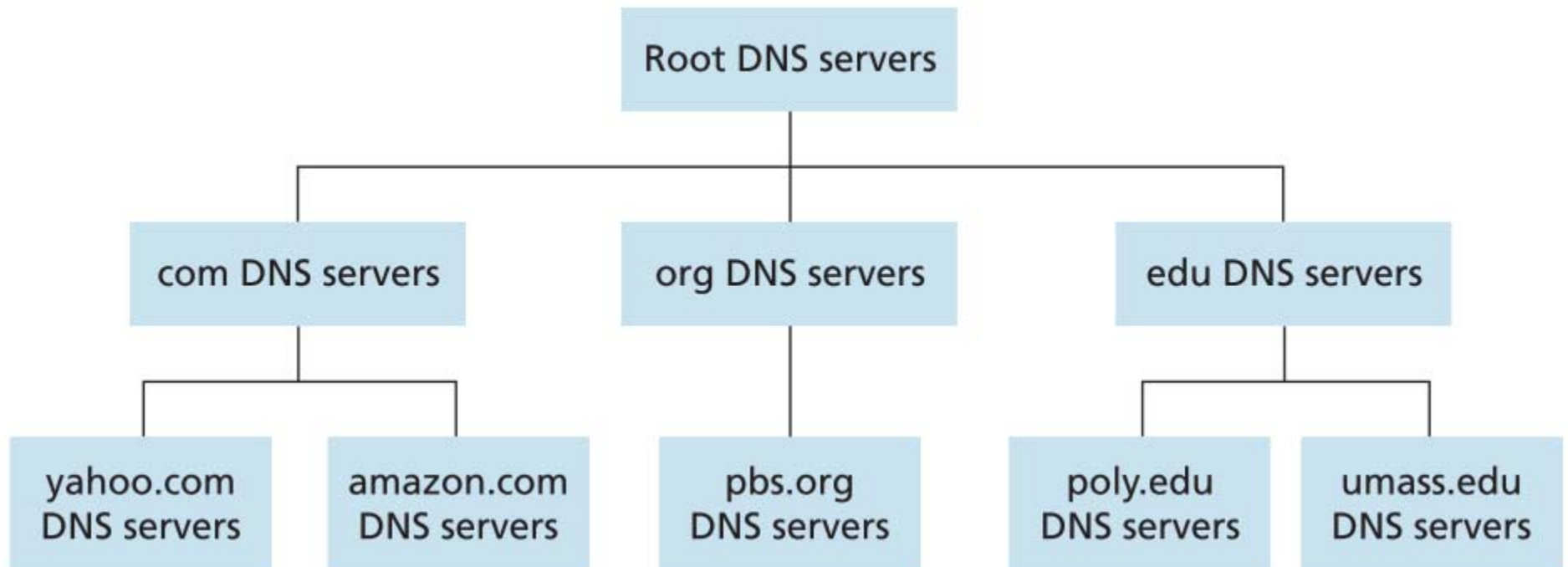


- ❑ **SMTP**: chuyển email tới server của người nhận
- ❑ Giao thức truy cập thư điện tử (mail access protocol): lấy email từ server
  - **POP**: Post Office Protocol [RFC 1939]: xác thực, tải về
  - **IMAP**: Internet Mail Access Protocol [RFC 1730]: có nhiều thao tác hơn, ví dụ thao tác với email trên server
  - **HTTP**: gmail, Hotmail, Yahoo! Mail,...

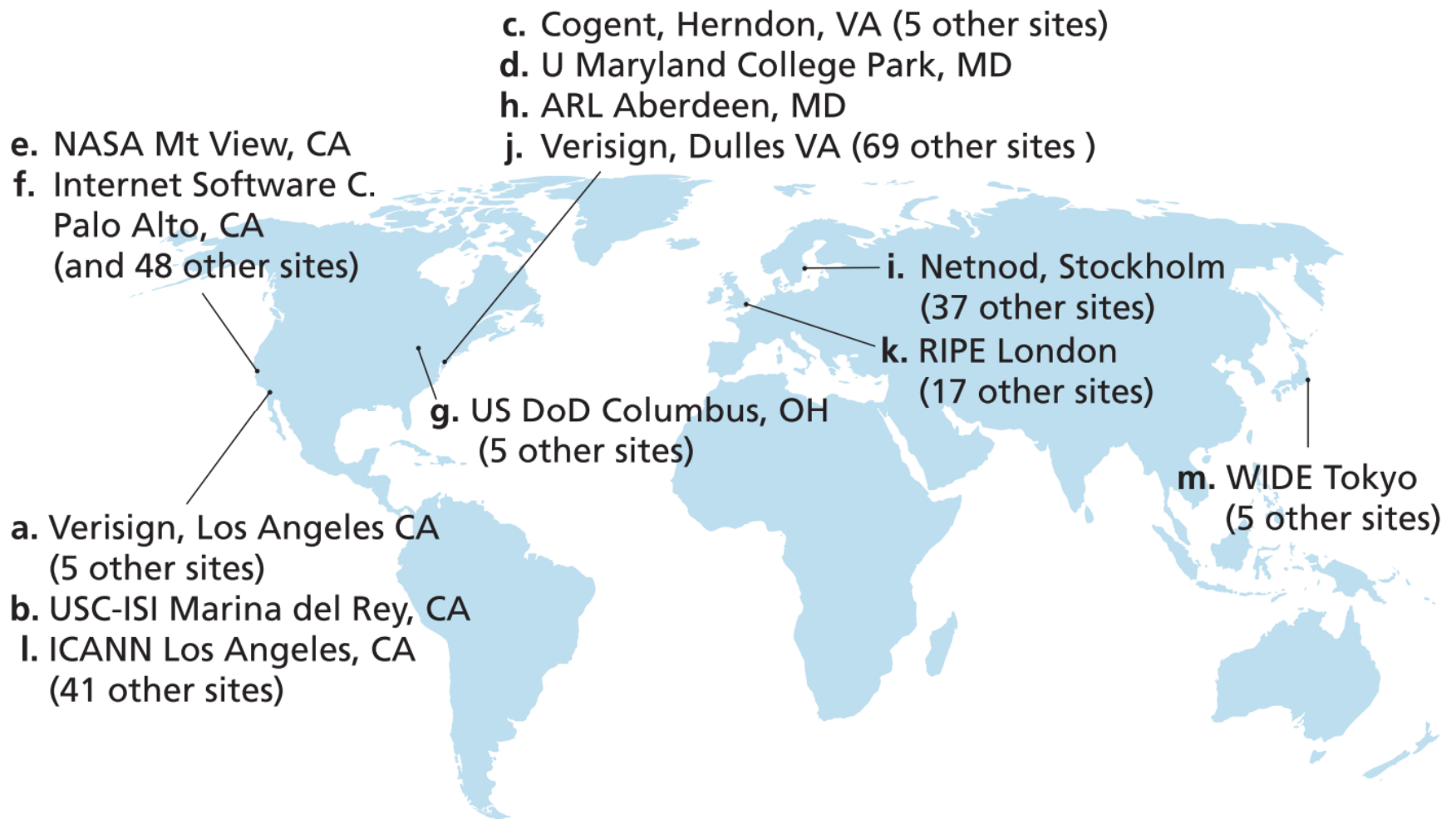
## Chương 2: Tầng ứng dụng

- ❑ Đặc điểm của ứng dụng mạng
- ❑ Web và HTTP
- ❑ Truyền tập tin: FTP
- ❑ Thư điện tử
- ❑ Hệ thống tên miền: DNS
- ❑ Ứng dụng ngang hàng
- ❑ Lập trình socket

# Cơ sở dữ liệu phân cấp và phân tán

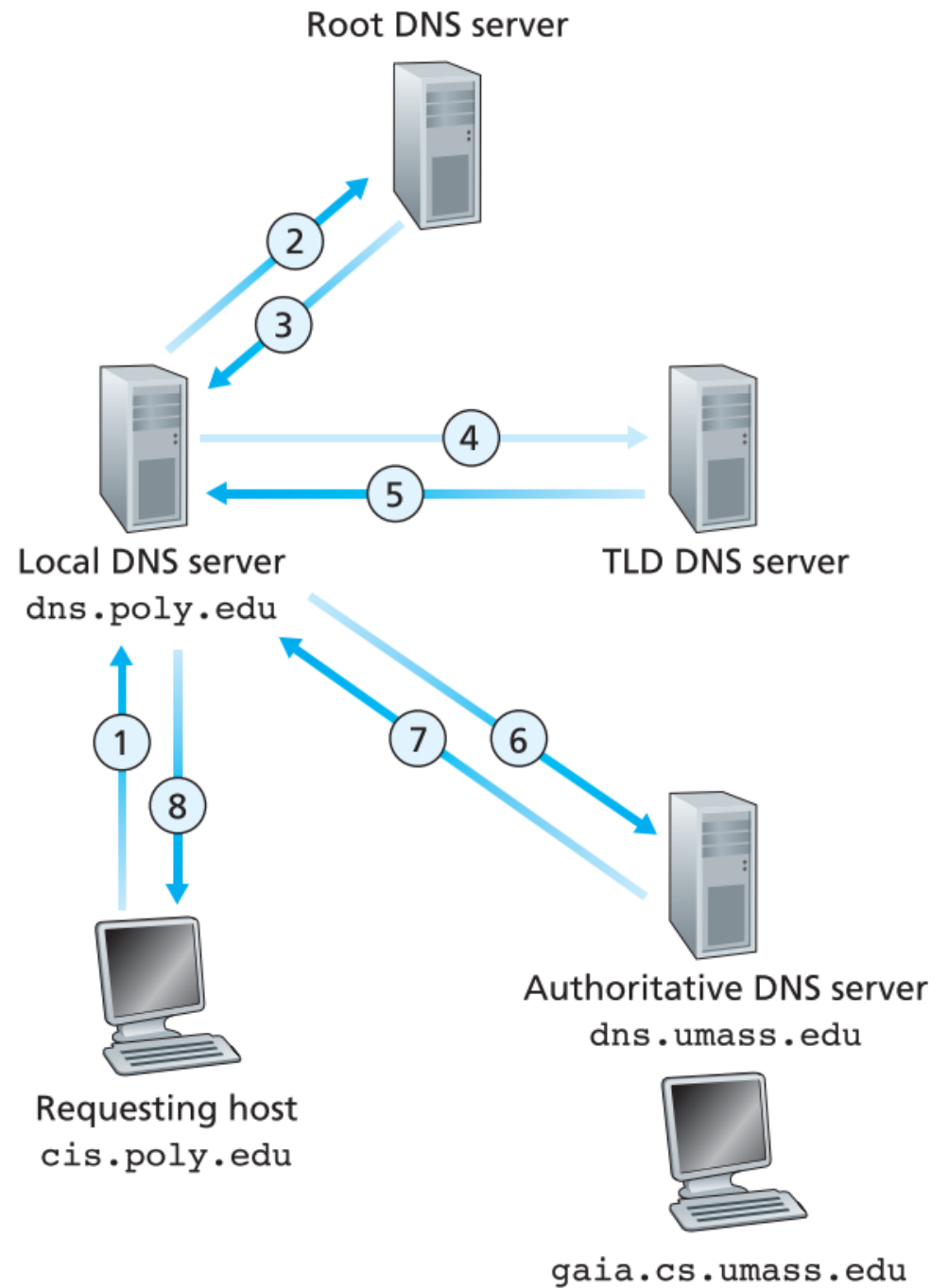


# Cơ sở dữ liệu phân cấp và phân tán



# Giao tiếp giữa DNS server

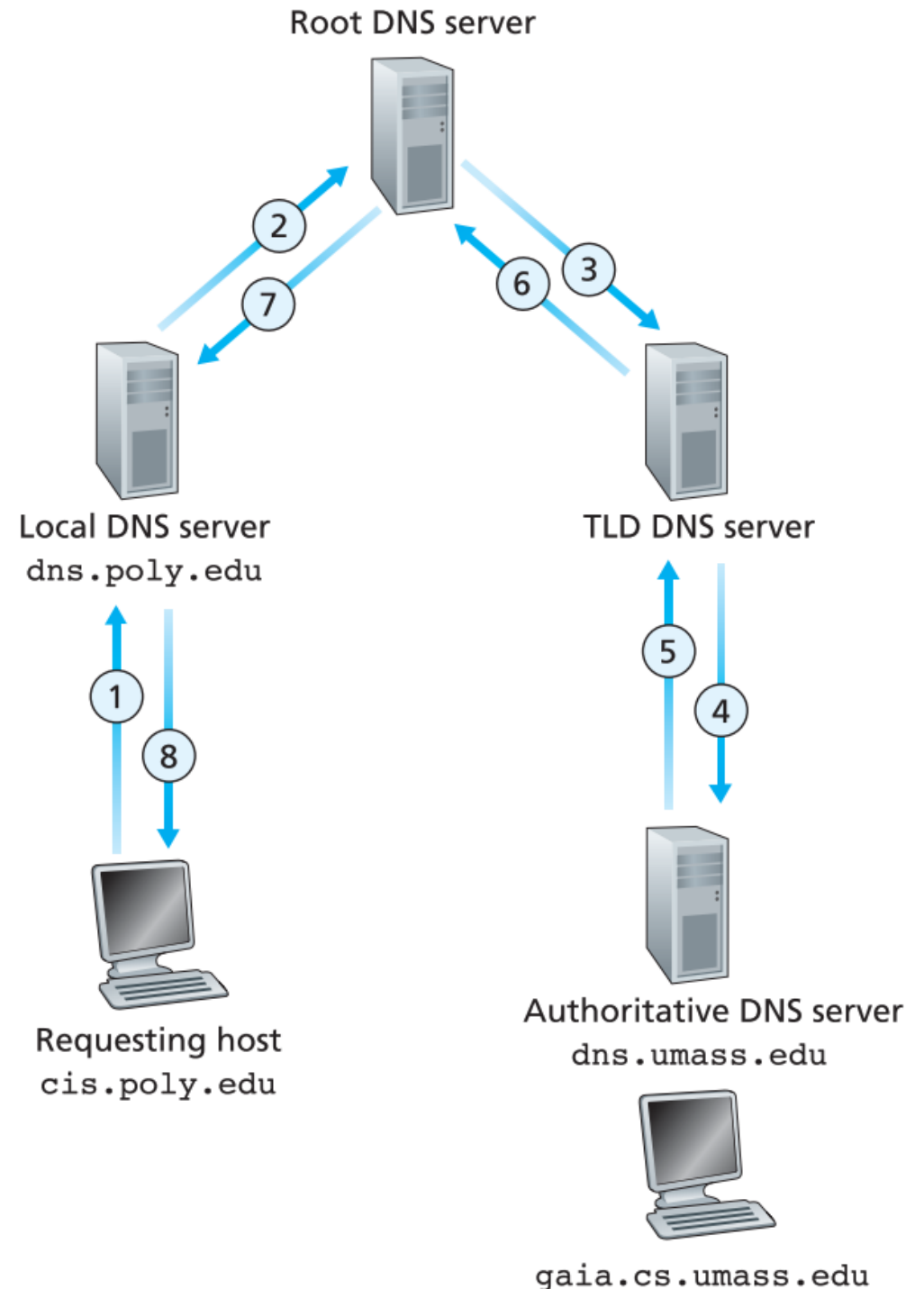
- ❑ Yêu cầu lặp (iterated query)





# Giao tiếp giữa DNS server

- ❑ Yêu cầu đệ quy  
(Recursive queries)



# Hoạt động của DNS

- ❑ khi name server nhận biết được mapping, name server sẽ lưu (*cache*) mapping
  - thông tin lưu trong cache entry, và bị xóa sau một khoảng thời gian (TTL)
  - TLD server thường được lưu trong các local name server
    - vì vậy root name server sẽ không thường xuyên được truy vấn
- ❑ các cached entry có không được cập nhật
  - nếu host name thay đổi địa chỉ IP của nó, địa chỉ IP này có thể không được cập nhật tới khi TTL quá hạn
- ❑ cơ chế cập nhật: RFC 2136

# Bản ghi DNS

**DNS:** cơ sở dữ liệu phân tán chứa các bản ghi tài nguyên (resource records **(RR)**)

RR format: (name, value, type, ttl)

## type=A

- **name** là hostname
- **value** là IP address

## type=NS

- **name** là domain (ví dụ: foo.com)
- **value** là hostname của authoritative name server của domain

## type=CNAME

- **name** là alias name cho một số “canonical” (real) name
- **www.ibm.com** thực tế là **servereast.backup2.ibm.com**
- **value** là canonical name

## type=MX

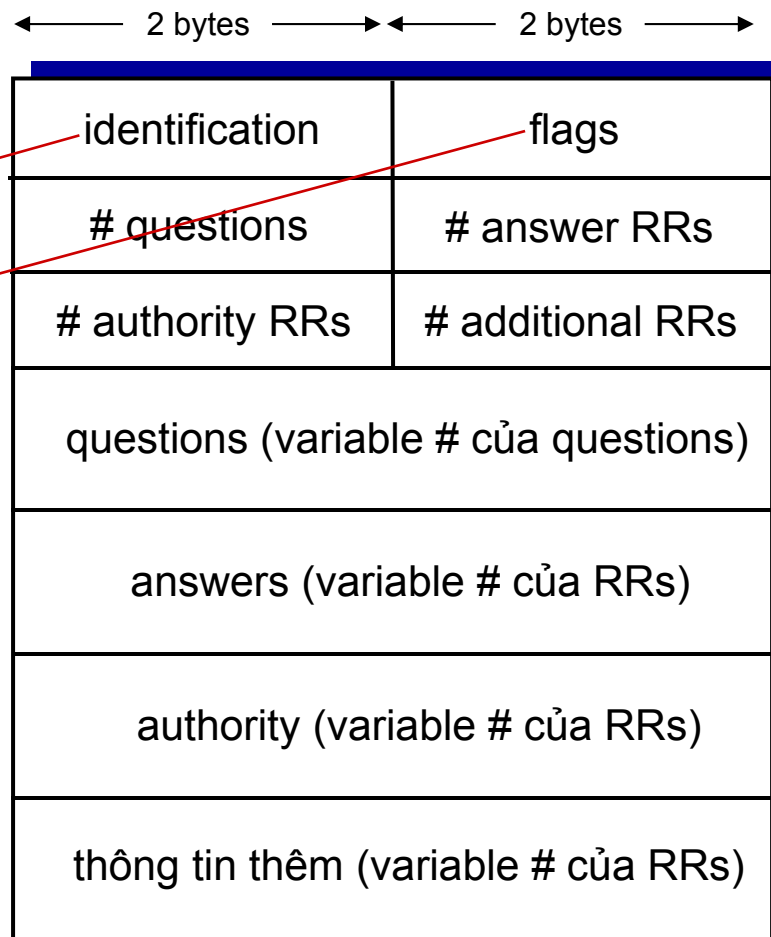
- **value** là tên của mailserver liên kết với **name**

# Giao thức DNS protocol, bản tin DNS

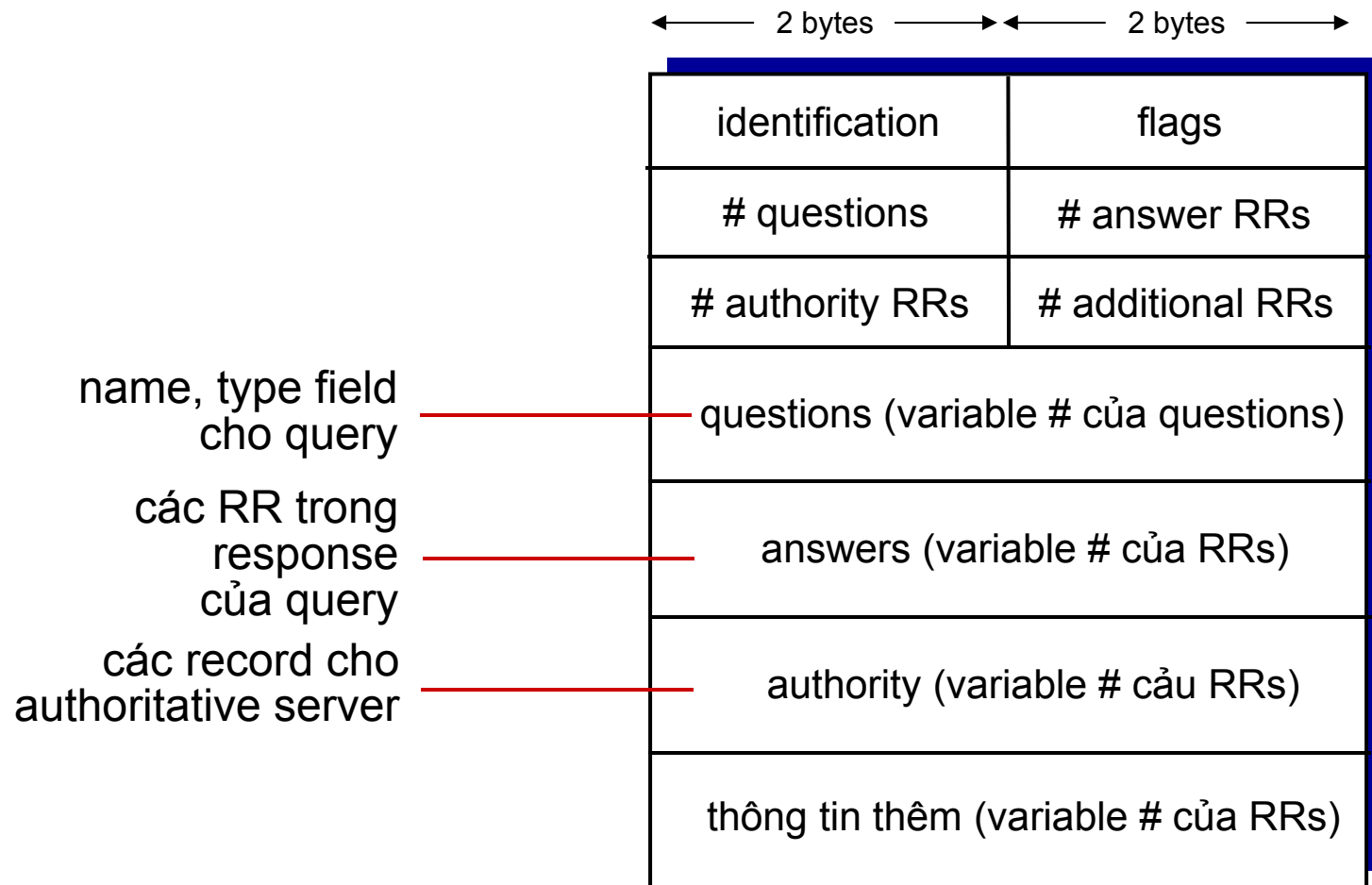
□ *query* message và *reply* message, có cùng cấu trúc

message header

- ❖ **identification**: 16 bit # cho query, reply của query có cùng giá trị id
- ❖ **flag**:
  - query hoặc reply
  - recursion desired
  - recursion available
  - authoritative reply



# Giao thức DNS protocol, bản tin DNS



# Chèn bản ghi vào DNS

- ❑ ví dụ: tổ chức mới “Network Utopia”
- ❑ đăng kí tên networkutopia.com tại *DNS registrar* (ví dụ, Network Solutions)
  - cung cấp các name, các IP address của authoritative name server (primary và secondary)
  - registrar chèn 2 RR vào .com TLD server:  
(networkutopia.com, dns1.networkutopia.com, NS)  
(dns1.networkutopia.com, 212.212.212.1, A)
- ❑ tạo authoritative server type A record cho www.networkutopia.com; type MX record cho networkutopia.com

# Tấn công DNS

## Tấn công DDoS

- ❑ làm root server quá tải bằng cách tăng lưu lượng
  - hiện tại đã bị ngăn chặn
  - Traffic Filtering
  - các Local DNS servers lưu các IP của các TLD server
- ❑ làm TLD server quá tải

## Tấn công Redirect

- ❑ Man-in-middle
  - chặn query
- ❑ DNS poisoning
  - gửi reply giả tới DNS server, which caches

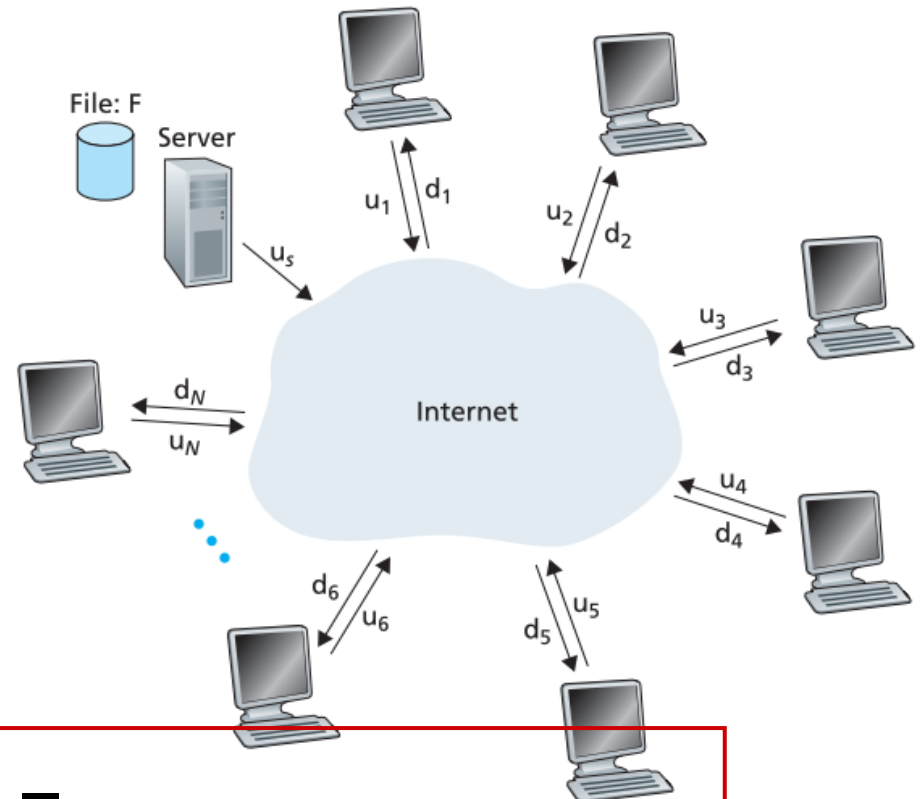
# Chương 2: Tầng ứng dụng

- ❑ Đặc điểm của ứng dụng mạng
- ❑ Web và HTTP
- ❑ Truyền tập tin: FTP
- ❑ Thư điện tử
- ❑ Hệ thống tên miền: DNS
- ❑ Ứng dụng ngang hàng
- ❑ Lập trình socket



# Khả năng tùy biến quy mô của kiến trúc P2P

- ❑ **server transmission:** phải tuần tự gửi (upload) N bản sao của 1 file:
  - time để gửi 1 file:  $F/u_s$
  - thời gian để gửi N bản sao:  $NF/u_s$
- ❑ **client:** mỗi client phải tải bản sao của file copy
  - $d_{\min}$  = tốc độ tải nhỏ nhất của client
  - min client download time:  $F/d_{\min}$



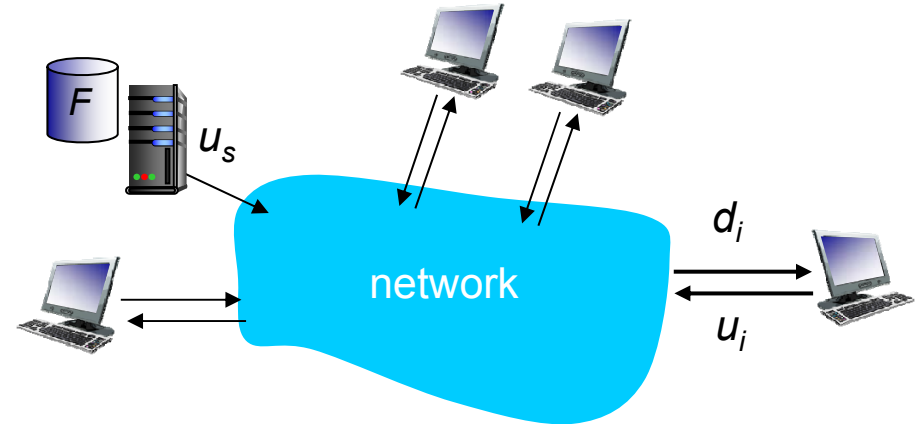
thời gian để gửi  $F$   
tới  $N$  client dùng  
kiến trúc client-server

$$D_{c-s} \geq \max\{NF/u_s, F/d_{\min}\}$$

tăng tuyến tính theo  $N$

# Khả năng tùy biến quy mô của kiến trúc P2P

- ❑ **server transmission:** phải upload ít nhất 1 bản sao
  - thời gian để gửi 1 bản sao:  $F/u_s$
- ❖ **client:** mỗi client phải tải bản sao của file
  - min client download time:  $F/d_{\min}$
- ❖ **clients:** cùng download  $NF$  bit
  - max upload rate (giới hạn max download rate):  $u_s + \sum u_i$



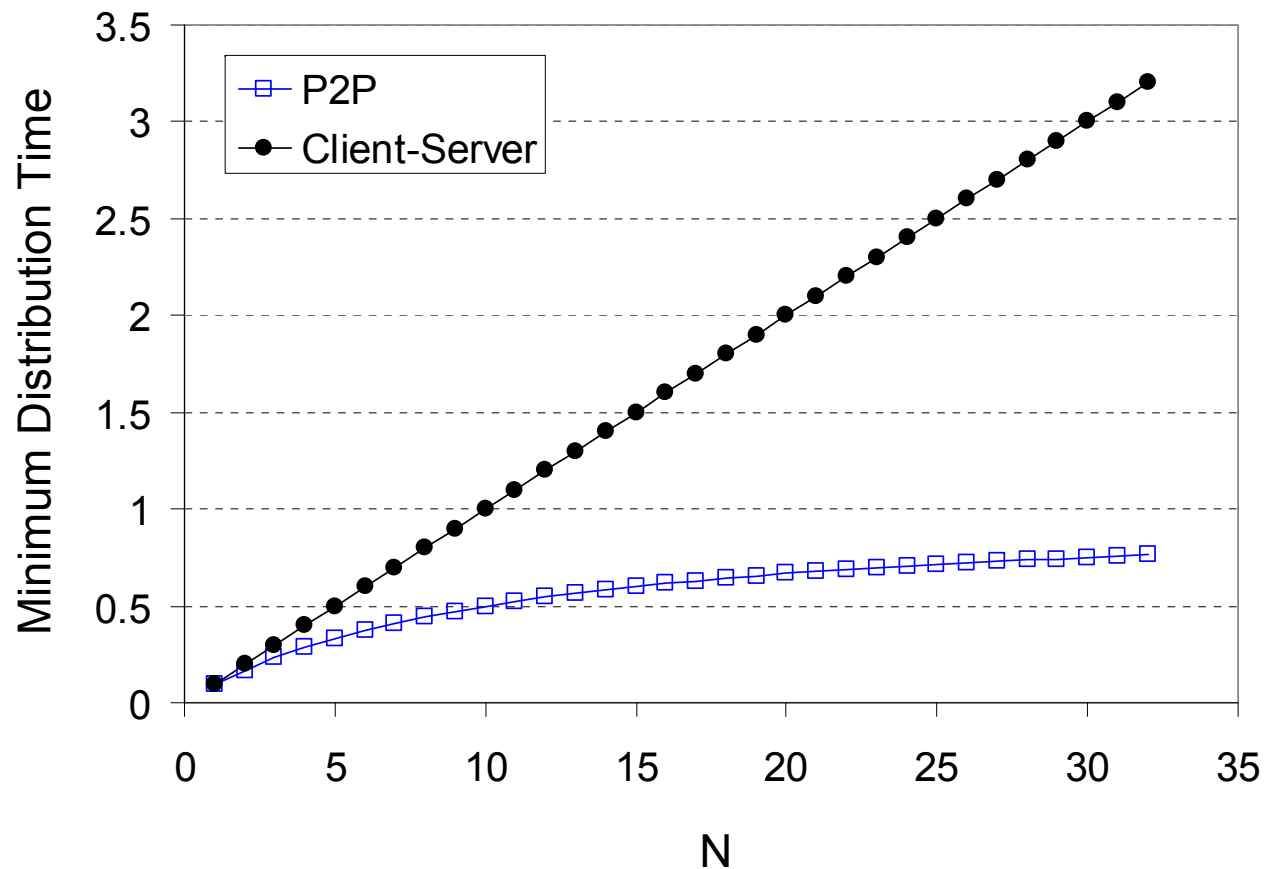
thời gian để chuyển  $F$   
tới  $N$  client sử dụng  
kiến trúc P2P

$$D_{P2P} \geq \max\{F/u_s, F/d_{\min}, NF/(u_s + \sum u_i)\}$$

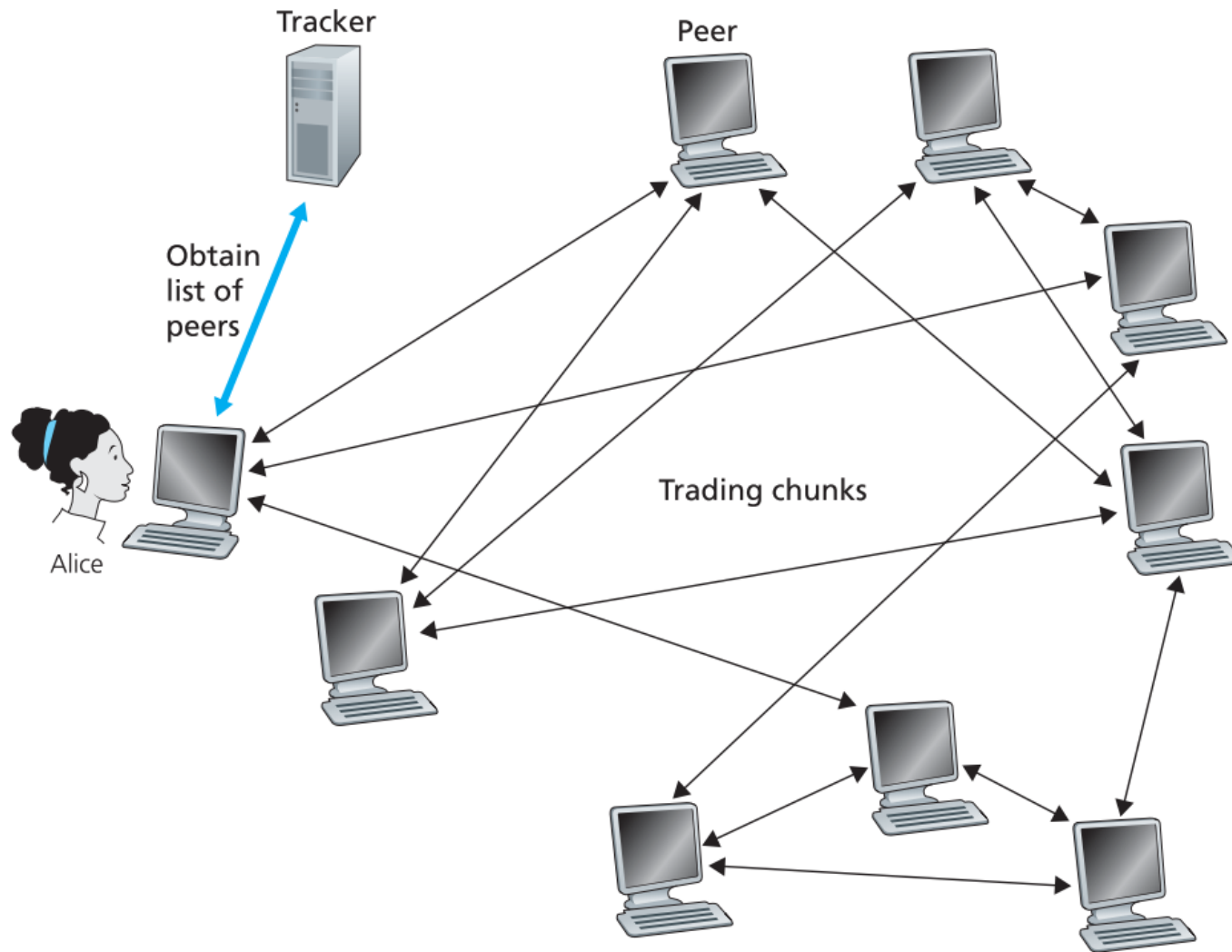
tăng tuyến tính với  $N$  ...  
... nhưng client cung cấp thêm capacity

# Khả năng tùy biến quy mô của kiến trúc P2P

tốc độ upload của client =  $u$ ,  $F/u = 1$  giờ,  $u_s = 10u$ ,  $d_{min} \geq u_s$



# Truyền file dùng BitTorrent



# Chương 2: Tầng ứng dụng

- ❑ Đặc điểm của ứng dụng mạng
- ❑ Web và HTTP
- ❑ Truyền tập tin: FTP
- ❑ Thư điện tử
- ❑ Hệ thống tên miền: DNS
- ❑ Ứng dụng ngang hàng
- ❑ Lập trình socket

# Lập trình Socket

Cách xây dựng một ứng dụng client/server giao tiếp dung socket

## Socket API

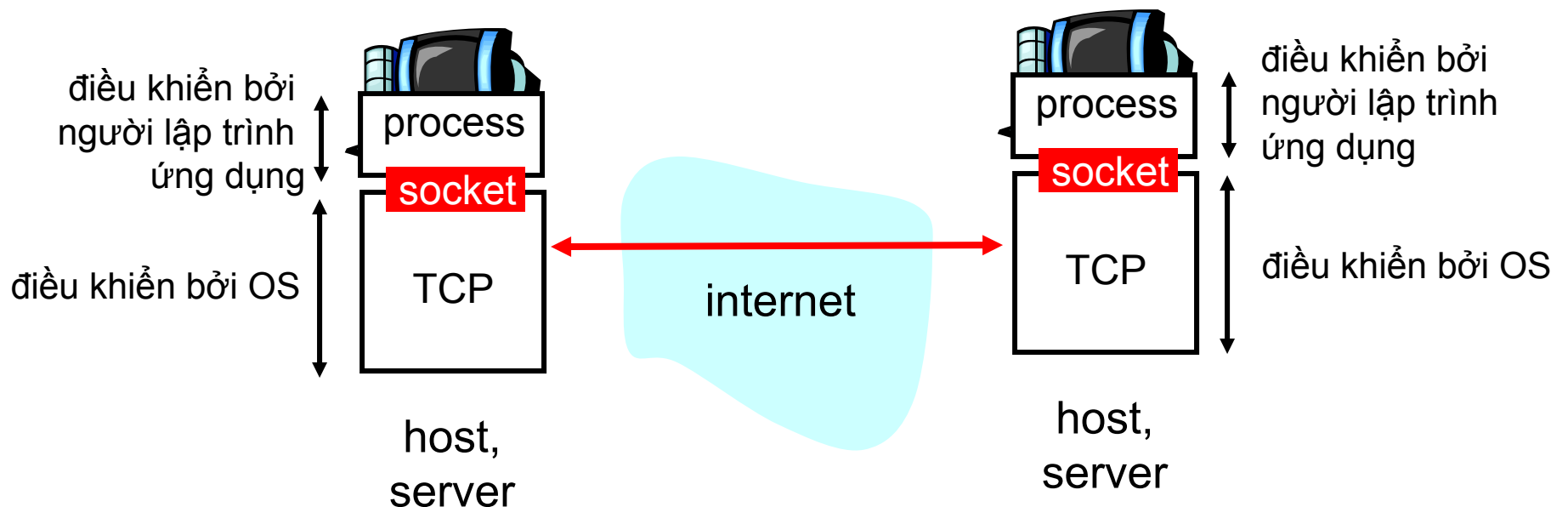
- ❑ xuất hiện trong BSD4.1 UNIX, 1981
- ❑ ứng dụng tạo, sử dụng và giải phóng
- ❑ mô hình client/server
- ❑ hai kiểu dịch vụ giao vận qua socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

## socket

giao diện *host-local*,  
*application-created*,  
*OS-controlled* : thông qua  
đó tiến trình ứng dụng có  
thể gửi và nhận các bản tin  
từ tiến trình ứng dụng khác

# Lập trình Socket dùng TCP

TCP service: truyền tin cậy các **byte** từ tiến trình này tới tiến trình khác



# Lập trình Socket dùng TCP

## Client phải liên lạc với server

- ❑ server process phải chạy trước
- ❑ server phải đã tạo socket (door) để chờ kết nối từ client

## Client liên lạc với server bằng cách

- ❑ tạo client-local TCP socket
- ❑ chỉ ra IP address, port number của server process
- ❑ khi **client tạo socket**: client TCP tạo kết nối tới server TCP

- ❑ khi có liên lạc từ client, **server TCP tạo socket mới** cho server process để giao tiếp với client

- cho phép server giao tiếp với nhiều client
- các source port number dùng để phân biệt các client (chi tiết hơn trong chương sau)

## phía ứng dụng

- *TCP cung cấp truyền tin cậy, đảm bảo thứ tự các byte giữa client và server*



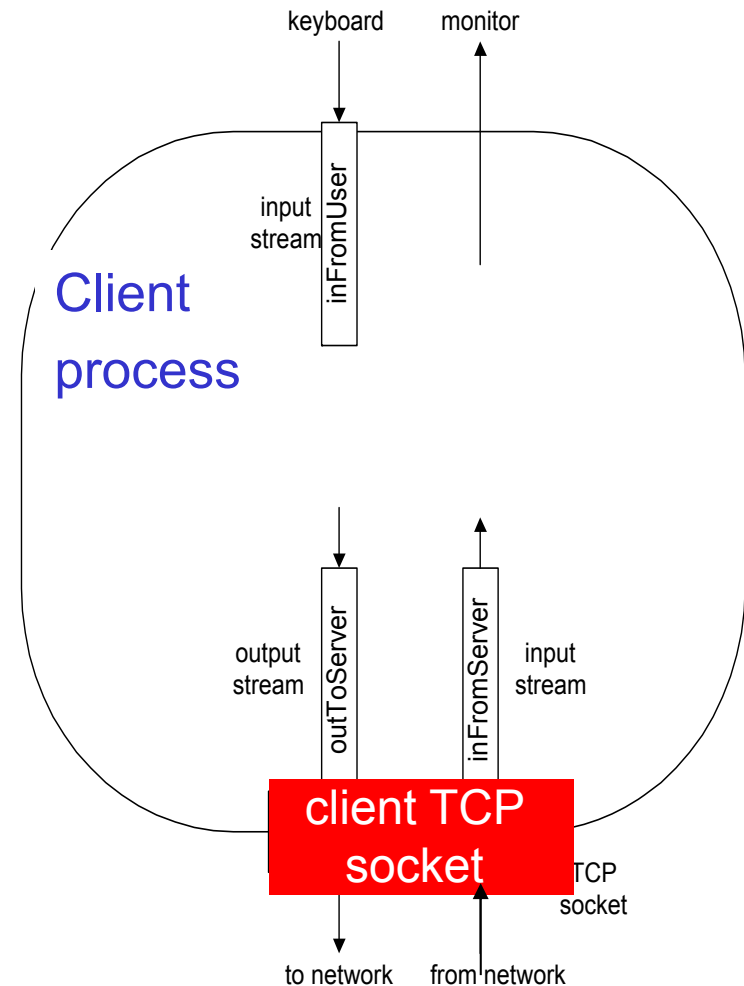
# Stream

- ❑ Một luồng (**stream**) là một chuỗi các kí tự ra vào của một tiến trình
- ❑ **input stream** gắn với input source đối với process, ví dụ: bàn phím, socket
- ❑ **output stream** được gắn với output source, ví dụ: màn hình, socket

# Lập trình Socket dùng TCP

## ví dụ ứng dụng client-server:

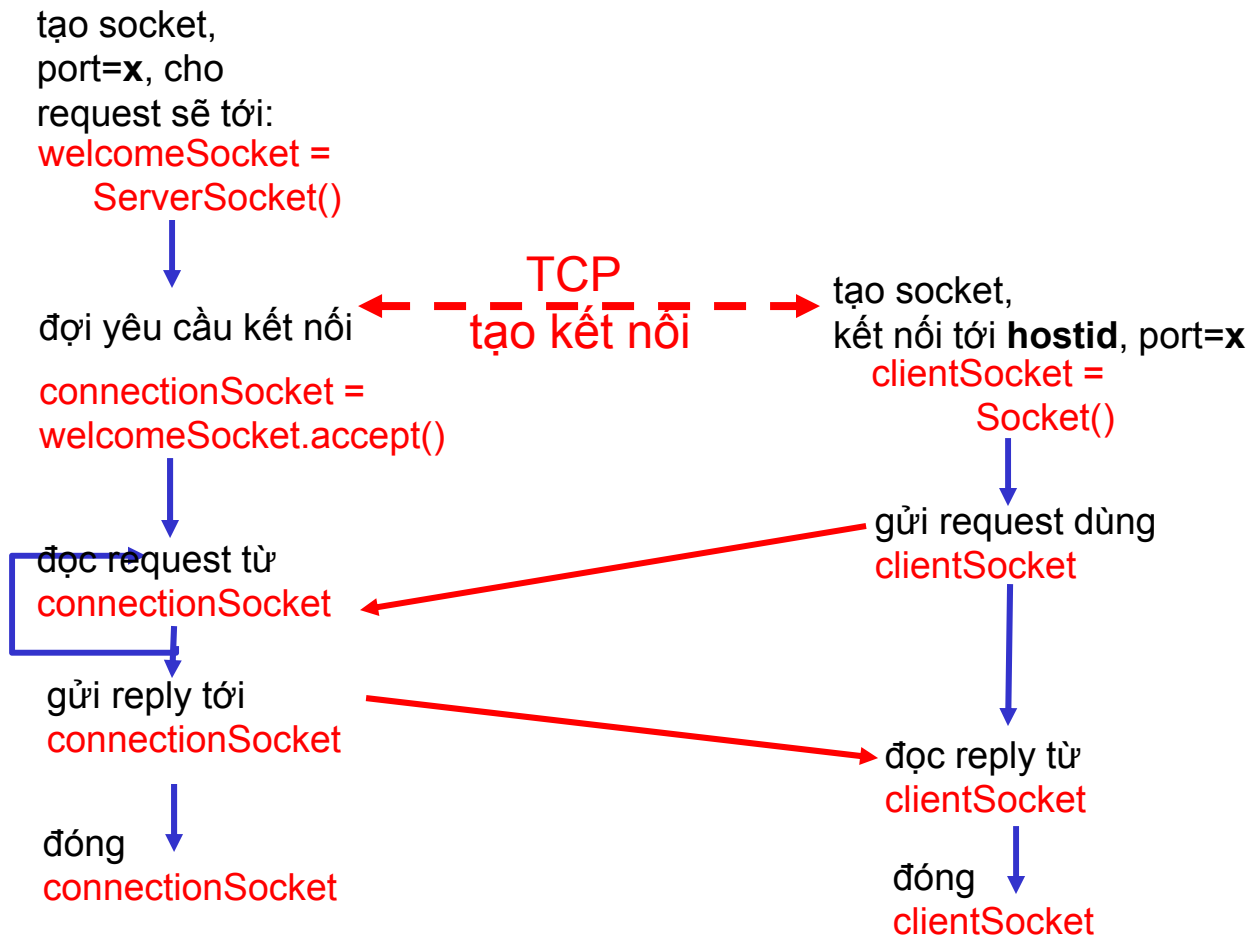
- 1) client đọc 1 dòng từ input stream (`inFromUser` stream) , gửi tới server qua socket (`outToServer` stream)
- 2) server đọc dòng từ socket
- 3) server chuyển dòng thành chữ hoa, gửi lại client
- 4) client đọc và hiện thị dòng nhận được từ socket (`inFromServer` stream)



# Tương tác giữa client socket và server socket: TCP

Server (chạy trên **hostid**)

Client





# Ví dụ: Java client (TCP)



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

```
    public static void main(String argv[]) throws Exception
    {
```

```
        String sentence;
        String modifiedSentence;
```

input stream  tạo  BufferedReader inFromUser =  
new BufferedReader(new InputStreamReader(System.in));

client socket,  
kết nối tới server  tạo  Socket clientSocket = new Socket("hostname", 6789);

output stream  
gắn với socket  tạo  DataOutputStream outToServer =  
new DataOutputStream(clientSocket.getOutputStream());

# Ví dụ: Java client (TCP)

tạo  
input stream  
gắn với socket

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

```
sentence = inFromUser.readLine();
```

gửi 1 dòng tới server

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

đọc 1 dòng từ server

```
modifiedSentence = inFromServer.readLine();
```

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

```
clientSocket.close();
```

```
}  
}
```

# Ví dụ: Java server (TCP)

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

```
class TCPServer {
```

```
    public static void main(String argv[]) throws Exception  
    {
```

```
        String clientSentence;  
        String capitalizedSentence;
```

tạo  
socket chờ  
tạo cổng 6789

```
        ServerSocket welcomeSocket = new ServerSocket(6789);
```

```
        while(true) {
```

đợi ở socket chờ,  
liên lạc từ client

```
            Socket connectionSocket = welcomeSocket.accept();
```

tạo input  
stream,  
gắn với socket

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

# Ví dụ: Java server (TCP)

tạo output stream, gắn với socket → `DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());`

đọc 1 dòng từ socket → `clientSentence = inFromClient.readLine();`

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

viết dòng đó ra socket → `outToClient.writeBytes(capitalizedSentence);`

}  
}  
}  
} → kết thúc while loop, lặp lại và đợi kết nối từ client khác

# Lập trình Socket dùng UDP

UDP: không kết nối giữa client và server

- ❑ không bắt tay (handshaking)
- ❑ nút gửi chỉ ra địa chỉ IP và cổng nhận trong mỗi gói tin
- ❑ nút nhận phải đọc địa chỉ IP và cổng từ gói tin nhận được

UDP: dữ liệu gửi có thể bị mất hoặc không theo thứ tự

phía ứng dụng

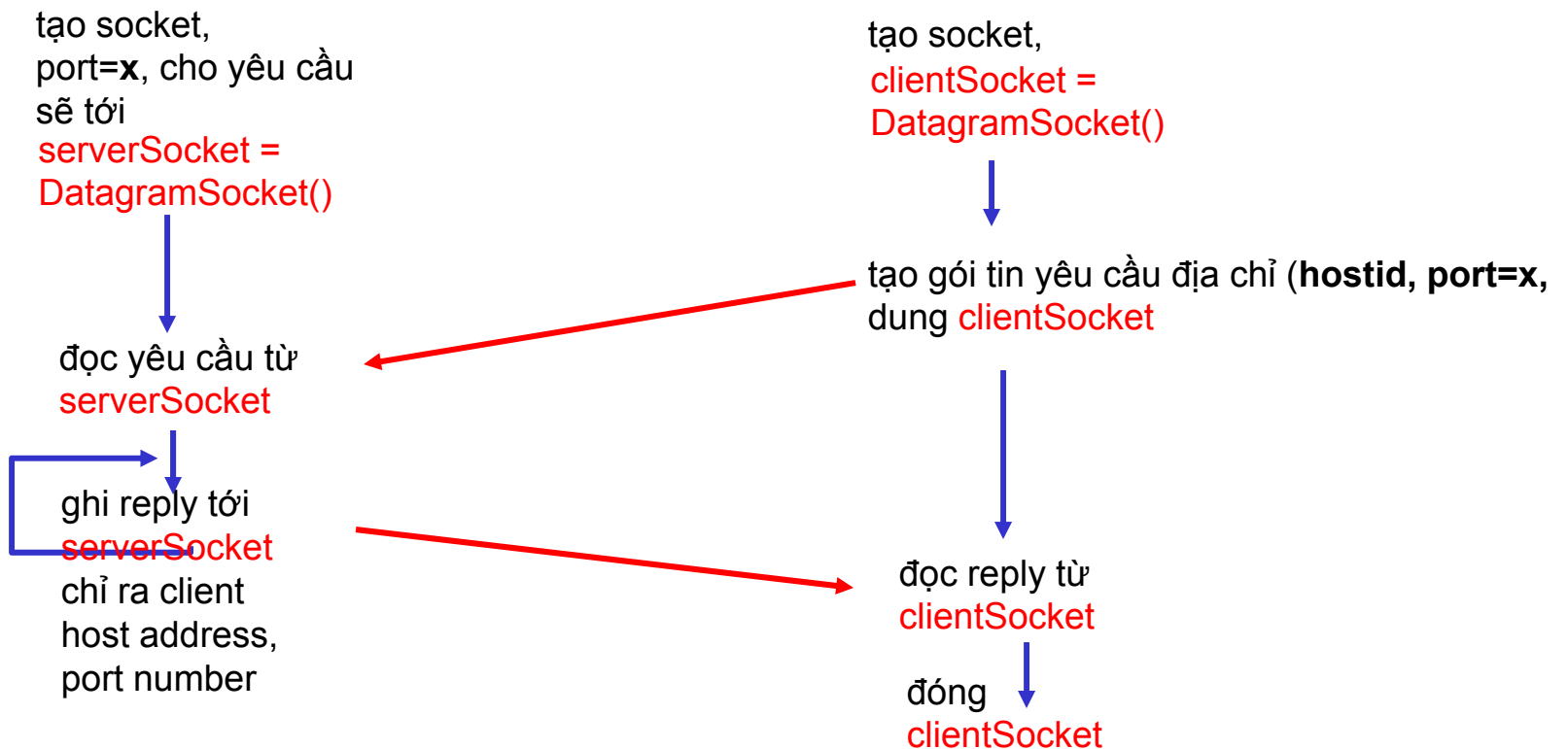
□ *UDP cung cấp truyền không tin cậy các byte giữa client và server*



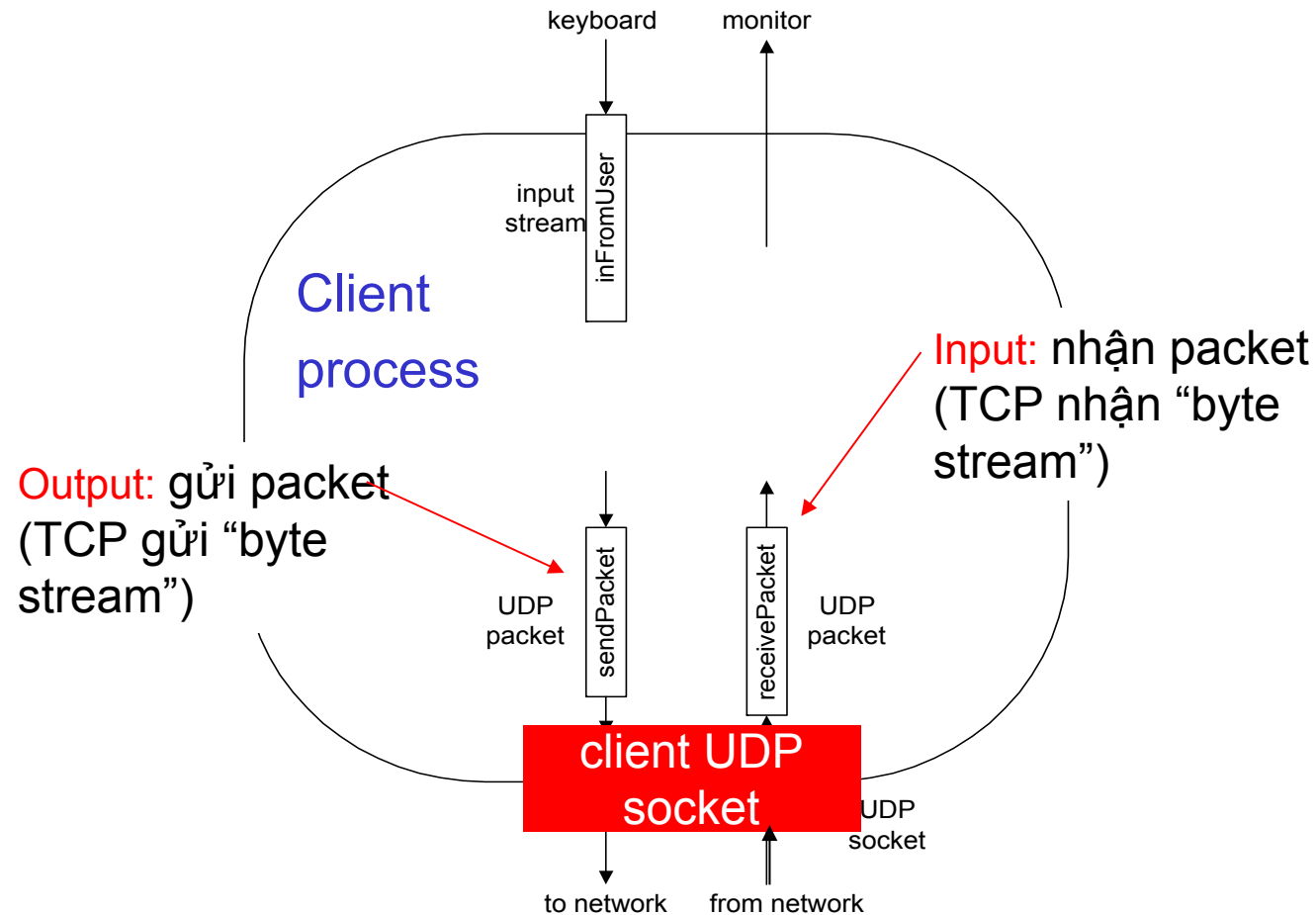
# Tương tác giữa client socket và server socket: UDP

Server (chạy trên **hostid**)

Client



# Ví dụ: Java client (UDP)



# Ví dụ: Java client (UDP)

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

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

tạo input stream → `BufferedReader inFromUser =  
 new BufferedReader(new InputStreamReader(System.in));`

tạo client socket → `DatagramSocket clientSocket = new DatagramSocket();`

chuyển hostname thành IP dung DNS → `InetAddress IPAddress = InetAddress.getByName("hostname");`  
`byte[] sendData = new byte[1024];`  
`byte[] receiveData = new byte[1024];`

```
String sentence = inFromUser.readLine();  
sendData = sentence.getBytes();
```

# Ví dụ: Java client (UDP)

tạo datagram với  
data-to-send,  
length, IP addr, port

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

gửi datagram  
tới server

```
clientSocket.send(sendPacket);
```

đọc datagram  
từ server

```
DatagramPacket receivePacket =  
    new DatagramPacket(receiveData, receiveData.length);  
  
clientSocket.receive(receivePacket);
```

```
String modifiedSentence =  
    new String(receivePacket.getData());  
  
System.out.println("FROM SERVER:" + modifiedSentence);  
clientSocket.close();  
}  
}
```

# Ví dụ: Java server (UDP)

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

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

tạo  
datagram socket  
tại cổng 9876

```
        DatagramSocket serverSocket = new DatagramSocket(9876);
```

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

```
        while(true)  
        {
```

tạo không gian  
chứa  
received datagram

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

nhận  
datagram

```
            serverSocket.receive(receivePacket);
```

# Ví dụ: Java server (UDP)

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

nhận IP addr  
port #, của nút gửi

```
→ InetAddress IPAddress = receivePacket.getAddress();  
→ int port = receivePacket.getPort();
```

```
String capitalizedSentence = sentence.toUpperCase();
```

```
sendData = capitalizedSentence.getBytes();
```

tạo datagram  
để gửi tới client

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

gửi datagram  
tới socket

```
→ serverSocket.send(sendPacket);  
    }  
    }  
    }
```

kết thúc while loop,  
lặp lại và đợi datagram khác

# Tóm tắt

- ❑ kiến trúc ứng dụng
  - client-server
  - P2P
- ❑ yêu cầu dịch vụ ứng dụng:
  - truyền tin cậy, độ trễ
- ❑ mô hình dịch vụ giao vận của Internet
  - tin cậy: TCP
  - không tin cậy: UDP
- ❑ giao thức:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P
- ❑ lập trình socket: TCP, UDP socket

# Tóm tắt

- ❑ trao đổi bản tin yêu cầu và bản tin trả lời:
  - client yêu cầu thông tin hoặc dịch vụ
  - server trả lời với dữ liệu và mã trạng thái
- ❑ cấu trúc của bản tin:
  - header: các trường cung cấp thông tin về dữ liệu (data)
  - data: thông tin giao tiếp

- ❑ *một số nguyên tắc*
  - bản tin dữ liệu và điều khiển: in-band, out-of-band
  - tập trung vs. phân tán
  - không trạng thái (stateless) vs. có trạng thái (stateful)
  - truyền tin cậy (reliable transfer) vs. truyền không tin cậy



# An interview with Marc Andreessen

- ❑ Marc Andreessen is the co-creator of Mosaic, the Web browser that popularized the World Wide Web in 1993. In 1994, Marc Andreessen and Jim Clark founded Netscape, whose browser was by far the most popular browser through the mid-1990s. He is now a co-founder and general partner of venture capital firm Andreessen Horowitz, overseeing portfolio development with holdings that include Facebook, Foursquare, Groupon, Jawbone, Twitter, and Zynga. He serves on numerous boards, including Bump, eBay, Glam Media, Facebook, and Hewlett-Packard. He holds a BS in Computer Science from the University of Illinois at Urbana-Champaign.
- ❑ What excites you about the future of networking and the Internet? What are your biggest concerns?
  - The most exciting thing is **the huge unexplored frontier of applications and services** that programmers and entrepreneurs are able to explore—the Internet has unleashed creativity at a level that I don't think we've ever seen before. My biggest concern is **the principle of unintended consequences**—we don't always know the implications of what we do, such as the Internet being used by governments to run a new level of surveillance on citizens.

# Mạng máy tính

- Hình ảnh và nội dung trong bài giảng này có tham khảo từ sách và bài giảng của TS. J.F. Kurose and GS. K.W. Ross