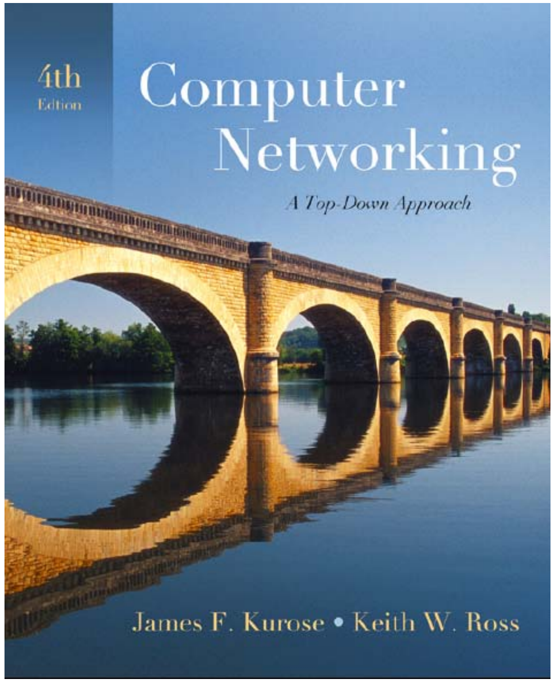
Chapter 2
Application Layer
第二章 應用層



Computer Networking: A Top Down Approach, 4th edition.
Jim Kurose, Keith Ross
Addison-Wesley, July 2007.

Chapter 2: Application layer

- 2.1 Principles of network applications 應用層原理
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P Applications
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP

Chapter 2: Application Layer

Our goals:

- conceptual, implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm 主從式架構
  - peer-to-peer paradigm 點對點架構
- learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS

Some network apps 網路應用

- ❒ e-mail 電子郵件
- ❒ Web 網站
- ❒ instant messaging 即時訊息
- ❒ remote login 遠端登入
- ❒ P2P file sharing 檔案分享
- ❒ multi-user network games 多人網路遊戲
- ❒ streaming stored video clips 串流多媒體
- ❒ voice over IP 網路電話
- ❒ real-time video conferencing 即時會議
- ❒ grid computing 網格運算

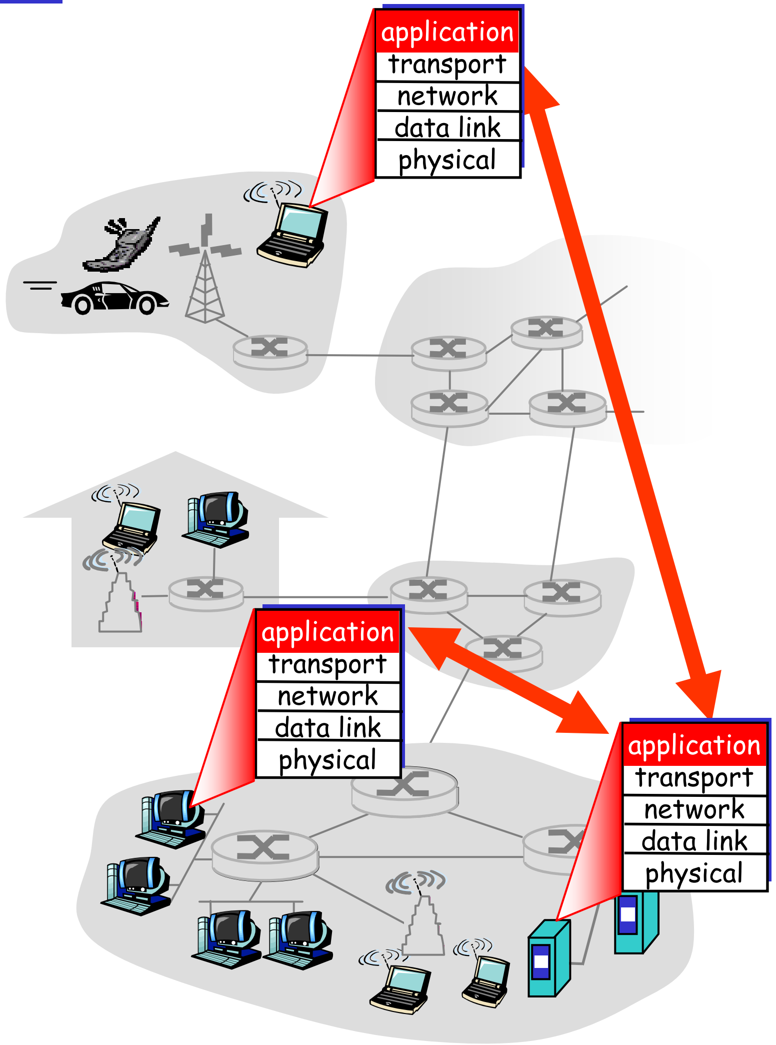
Creating a network app
建立網路應用程式

write programs that

- run on (different) *end systems* 在不同的終端系統上執行
- communicate over network 透過網路相互溝通
- e.g., web server software communicates with browser software

little software written for devices in network core

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



Chapter 2: Application layer

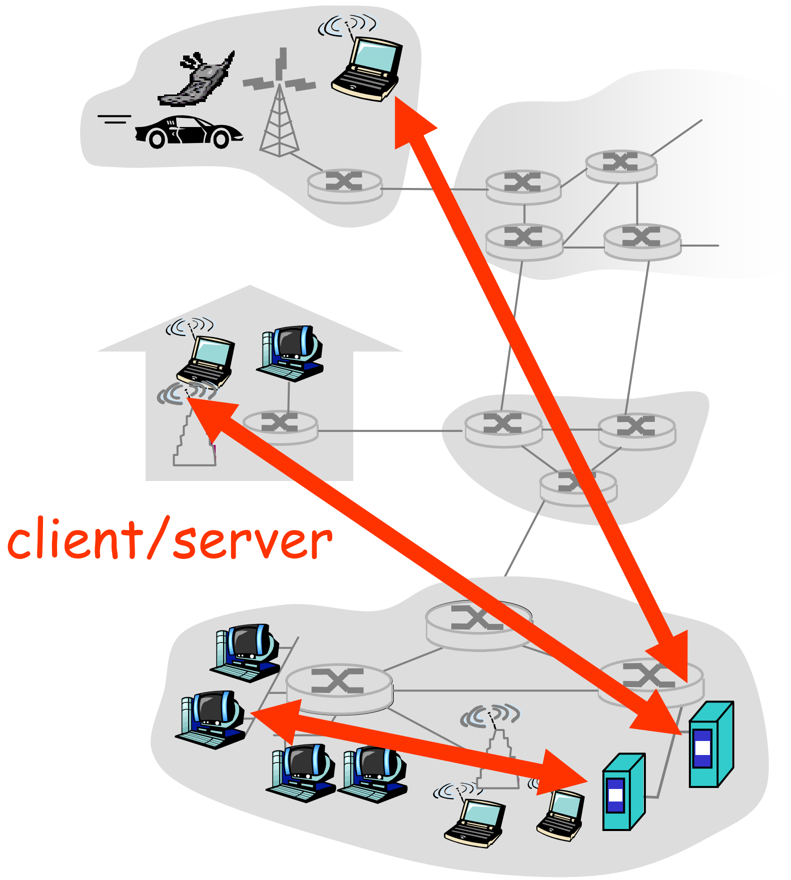
- 2.1 Principles of network applications 應用層原理
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
- 2.9 Building a Web server

Application architectures
應用程式架構

- Client-server 主從式架構
- Peer-to-peer (P2P) 點對點架構
- Hybrid of client-server and P2P 混合式架構

Client-server architecture

主從式架構



server: 伺服器端

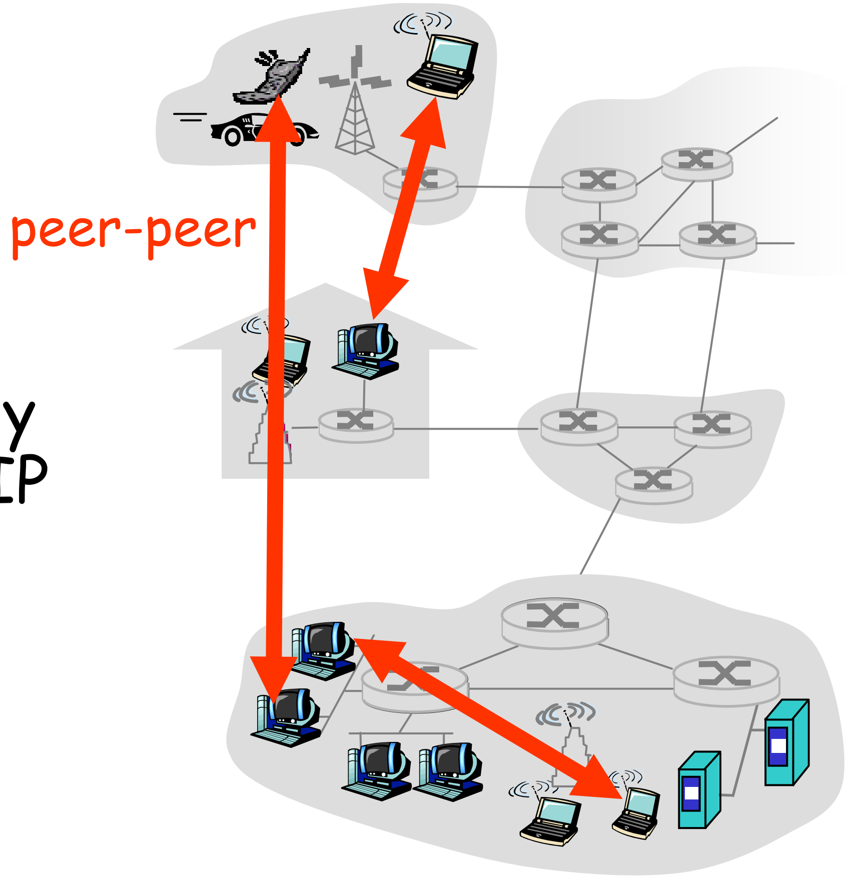
- always-on host
- permanent IP address
  固定位置
- server farms for scaling 多台機器同時服務

clients: 用戶端

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses 不固定位置
- do not communicate directly with each other
  用戶間不會直接溝通

Pure P2P architecture
點對點架構

- *no* always-on server
  沒有固定的伺服器
- arbitrary end systems directly communicate
  用戶間直接溝通
- peers are intermittently connected and change IP addresses



Highly scalable but difficult to manage
具高擴充性但難以管理

Hybrid of client-server and P2P
混和式架構（主從式+點對點）

Skype

- voice-over-IP P2P application
- centralized server: finding address of remote party: 先從伺服器找到欲通話對象的位址
- client-client connection: direct (not through server) 直接與通話對象通話

Instant messaging 即時通訊 MSN、AOL、Yahoo

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

Processes communicating 行程通訊

Process 行程: program running within a host.

- within same host, two processes communicate using inter-process communication (defined by OS). 行程間通訊
- processes in different hosts communicate by exchanging messages 透過交換“訊息”通訊

用戶端及伺服器端行程的分別

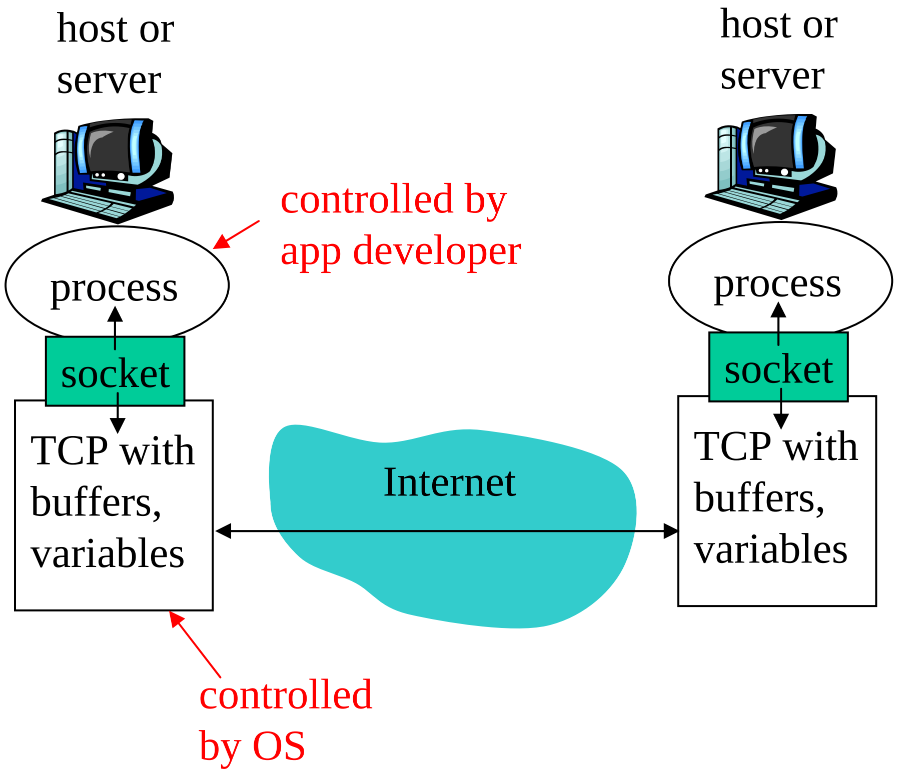
Client process: process that initiates communication

Server process: process that waits to be contacted

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

Sockets 行程通訊的 “大門”

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door which brings message to socket at receiving process


host or server
process
socket
TCP with buffers, variables
Internet
host or server
process
socket
TCP with buffers, variables
controlled by app developer
controlled by OS

- API: (1) choice of transport protocol; (2) ability to fix a few parameters (lots more on this later)

Addressing processes 行程定址

- to receive messages, process must have *identifier*（獨一無二）
- host device has unique 32-bit IP address
- Q: does IP address of host on which process runs suffice for identifying the process?
  只有IP Address是否足夠？

Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?
  - *A:* No（否）, *many* processes can be running on same host 可同時在同一個主機上執行多個行程
- *identifier* includes both IP address and port numbers associated with process on host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - Port number: 80

App-layer protocol defines
應用層協定定義下列格式：

- Types of messages exchanged,
  交換的訊息種類
  - e.g., request, response
- Message syntax:
  訊息語法
  - what fields in messages & how fields are delineated
- Message semantics
  訊息語意
  - 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?
應用程式所需的傳輸層服務

Data loss 是否容忍資料遺失

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

Timing 是否要求即時到達

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

Bandwidth 是否需要最小頻寬

- 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 video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

Internet transport protocols services

網際網路上的傳輸層協定

TCP service:

- *connection-oriented 連結導向:* setup required between client and server processes
- *reliable transport 可信賴傳輸* between sending and receiving process
- *flow control 流量控制:* sender won't overwhelm receiver
- *congestion control 擁塞控制:* 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 2: Application layer

- 2.1 Principles of network applications
  - app architectures
  - app requirements
- 2.2 Web and HTTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP

Web and HTTP

First some jargon 術語

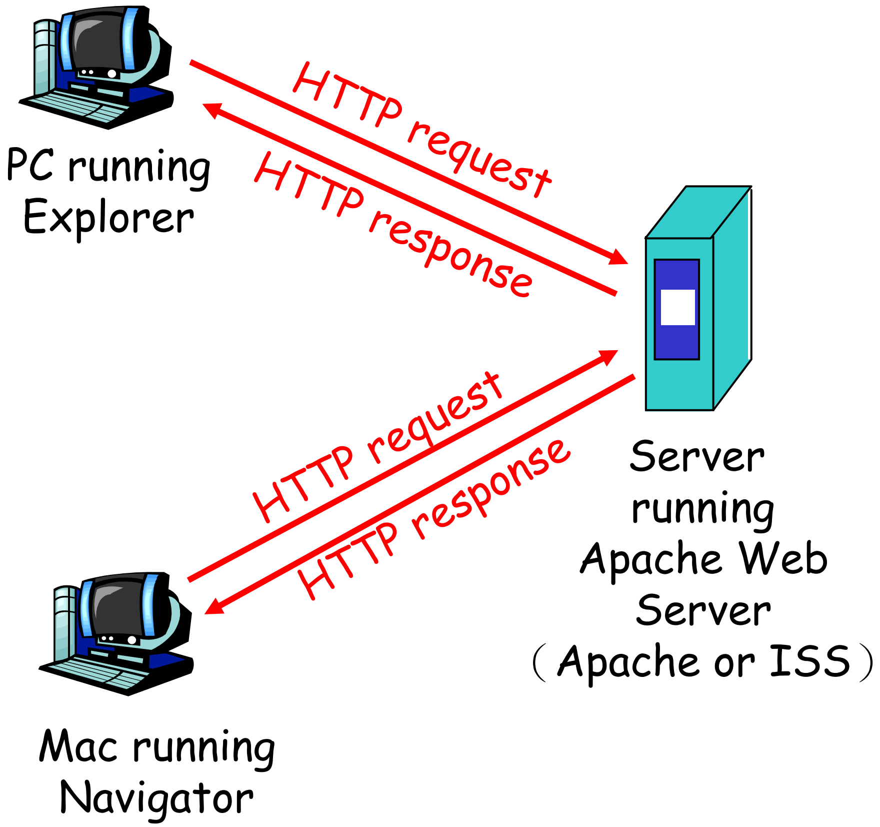
- Web page （網頁） consists of objects （物件）
- Object can be HTML file, JPEG image, Java applet, audio file,…
- Web page consists of base HTML-file （基本HTML 檔案） which includes several referenced objects
- Each object is addressable by a URL（Uniform Resource Locator ，全球資源定址）
- Example URL:

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

HTTP overview 超文件傳輸協定

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser 瀏覽器 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 overview (continued)

Uses TCP: 使用TCP協定

- 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"

無狀態協定

- server maintains no information about past client requests

aside

Protocols that maintain "state" are complex!

- past history (state) must be maintained
  需保留歷史資料
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP connections （HTTP連線）

Nonpersistent HTTP
非持續性連線

- At most one object is sent over a TCP connection. 一個TCP連線只傳輸一個物件
- HTTP/1.0 uses nonpersistent HTTP

Persistent HTTP
持續性連線

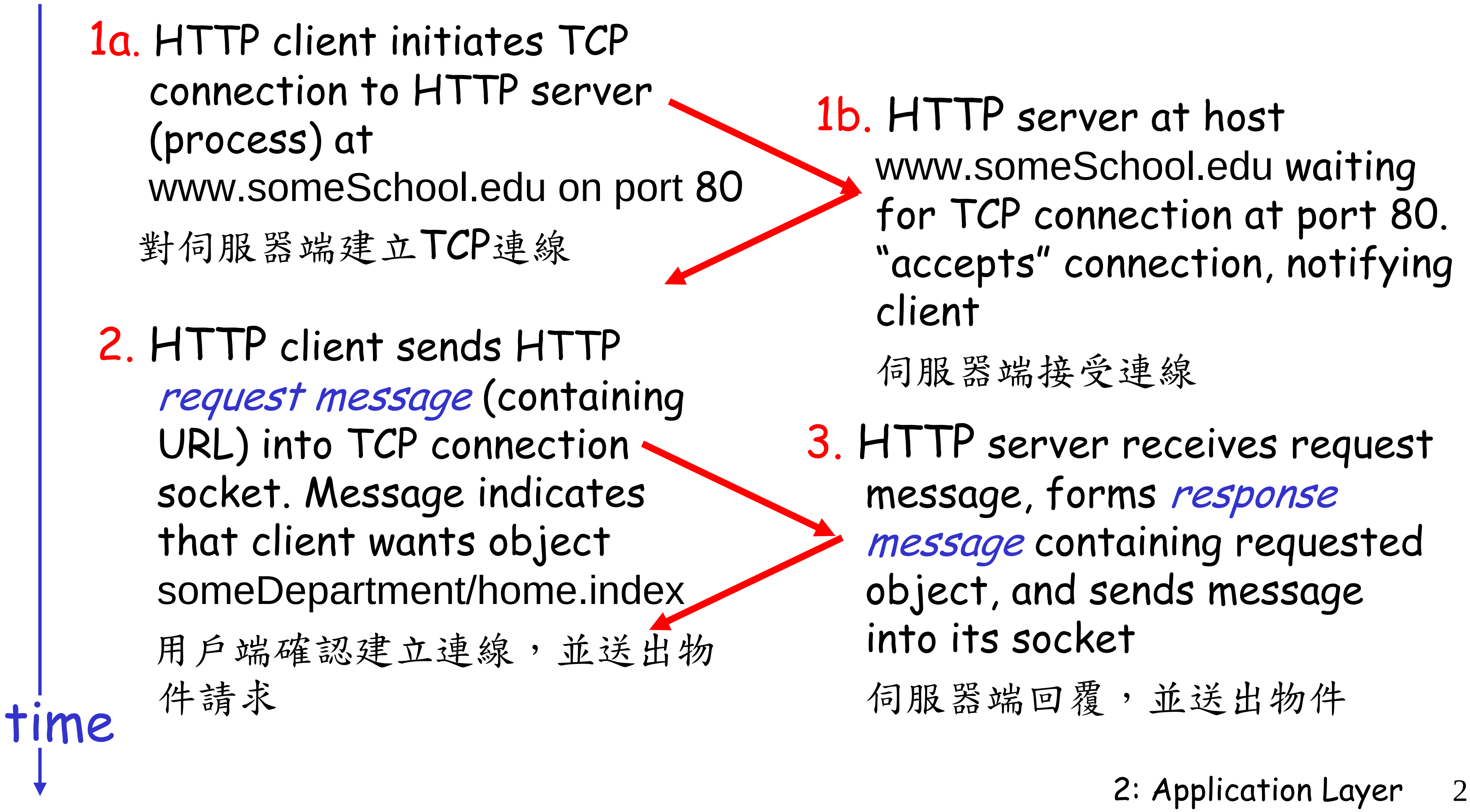
- Multiple objects can be sent over single TCP connection between client and server. 可在一個TCP連線傳輸多個物件
- HTTP/1.1 uses persistent connections in default mode

Nonpersistent HTTP 非持續性連線

Suppose user enters URL

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

(contains text,
references to 10
jpeg images)



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

對伺服器端建立TCP連線

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

伺服器端接受連線

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

用戶端確認建立連線，並送出物件請求

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

伺服器端回覆，並送出物件

Nonpersistent HTTP (cont.)

4. HTTP server closes TCP connection.

   伺服器端關閉TCP連線

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

   用戶端收到html檔案，並同時發現需要10個JPG物件

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

   針對每一個JPG物件重覆1-5步驟

time

Non-Persistent HTTP: Response time

非持續性連線：回覆時間

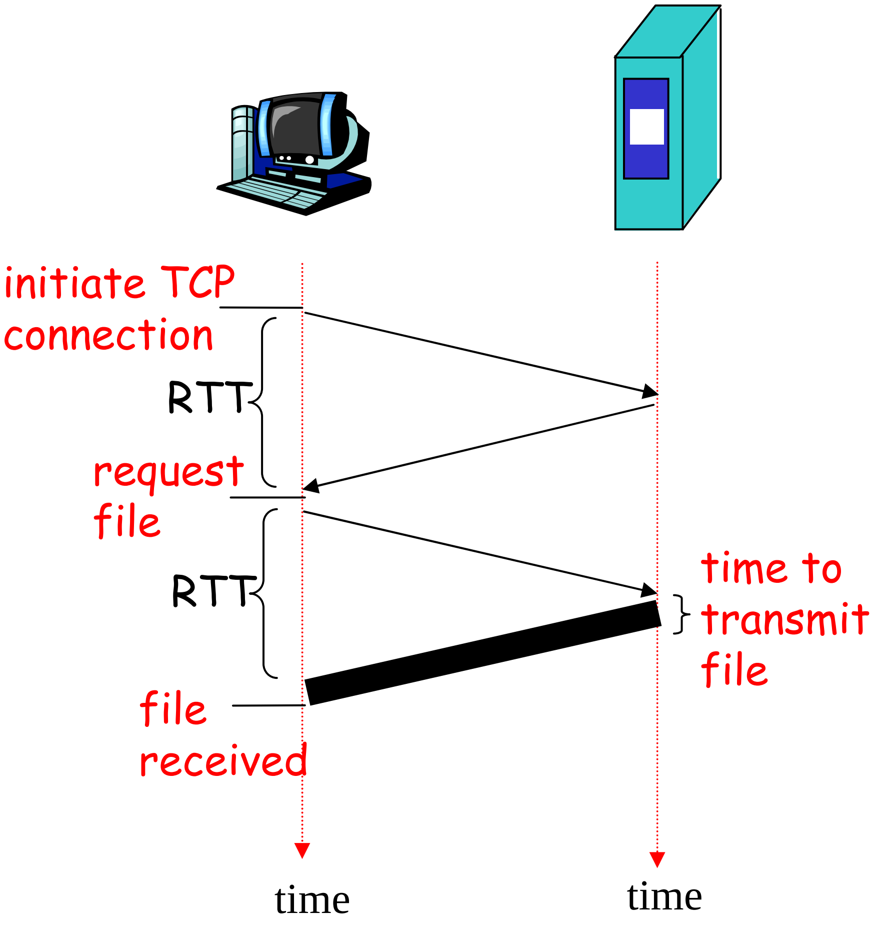
Definition of RTT 來回時間

time to send a small packet to travel from client to server and back.

Response time: 回覆時間

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

total = 2RTT+transmit time



Persistent HTTP 持續性連線

Nonpersistent HTTP issues:

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

Persistent HTTP

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

Persistent *without* pipelining:

不平行傳送的作法

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

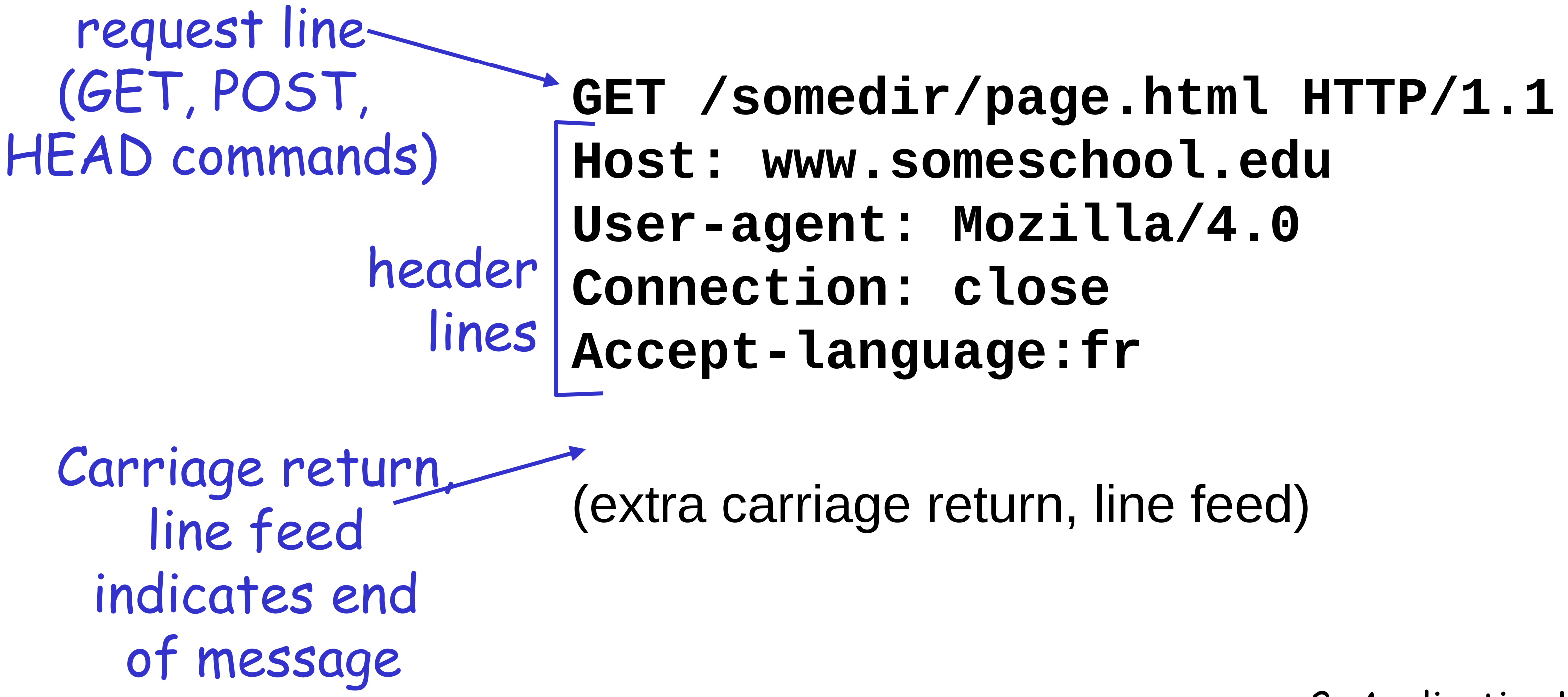
Persistent *with* pipelining:

平行傳送的作法

- 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

HTTP request message 請求訊息

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



request line (GET, POST, HEAD commands)

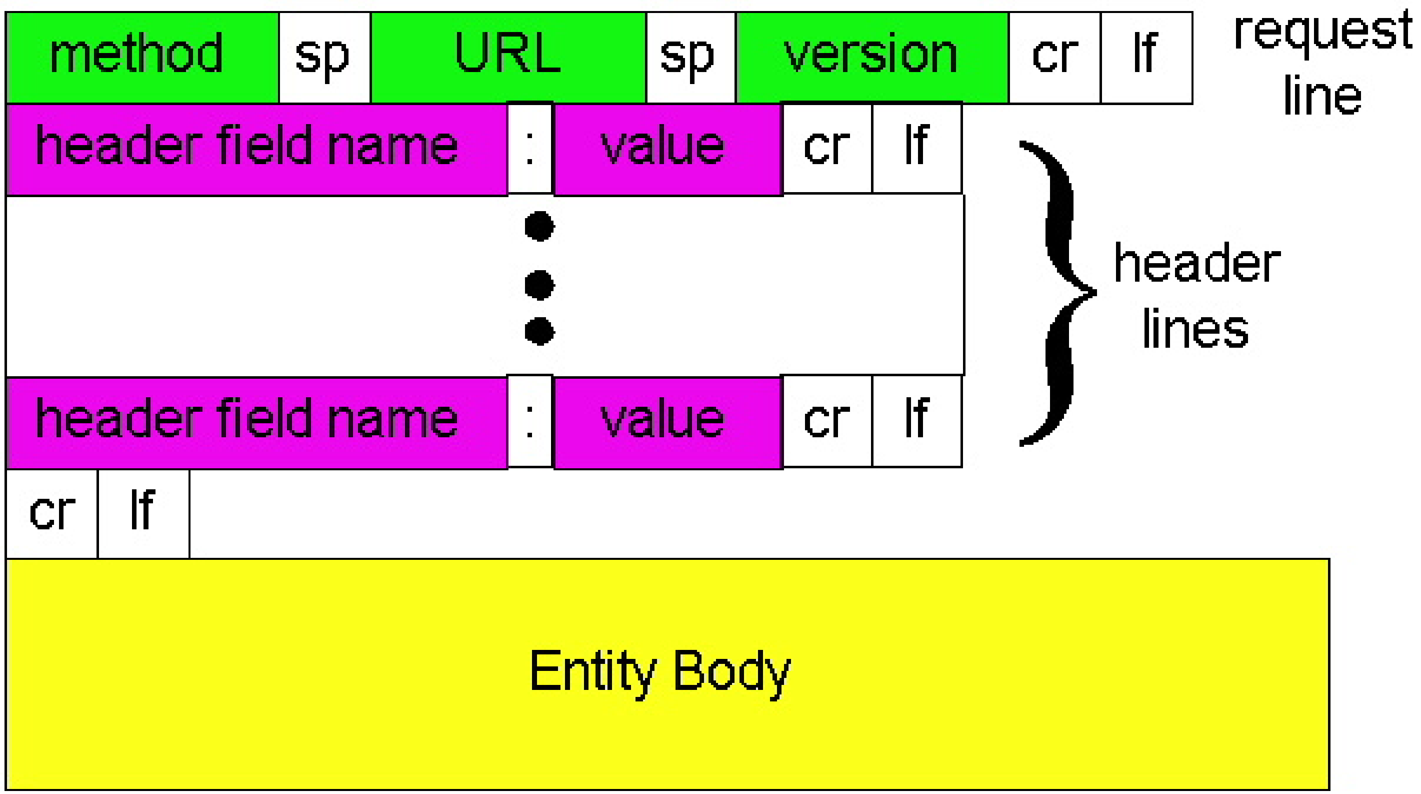
header lines

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



Uploading form input 輸入資料上傳

Post method: 表單輸入法

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

URL method: 附帶在URL

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

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

Method types

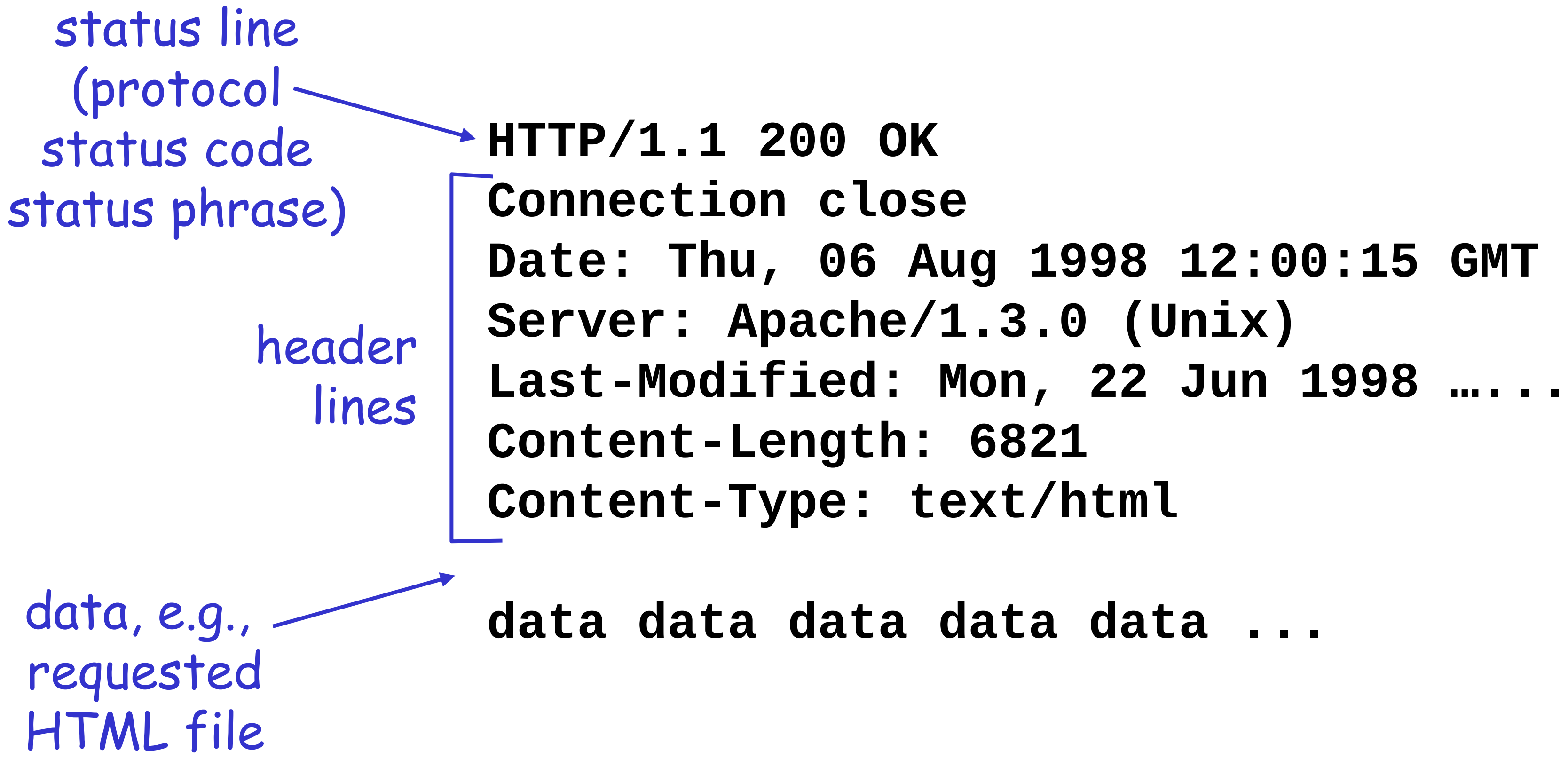
HTTP/1.0

- GET
- POST
- HEAD
  - asks server to leave requested object out of response
    不回應請求的物件（除錯用）

HTTP/1.1

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

HTTP response message 回應訊息



status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

HTTP response status codes 狀態碼

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

Trying out HTTP (client side) for yourself

試試看!!!

1. Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

Opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. Anything typed in sent to port 80 at cis.poly.edu

2. Type in a GET HTTP request:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.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!

User-server state: cookies

Many major Web sites use cookies

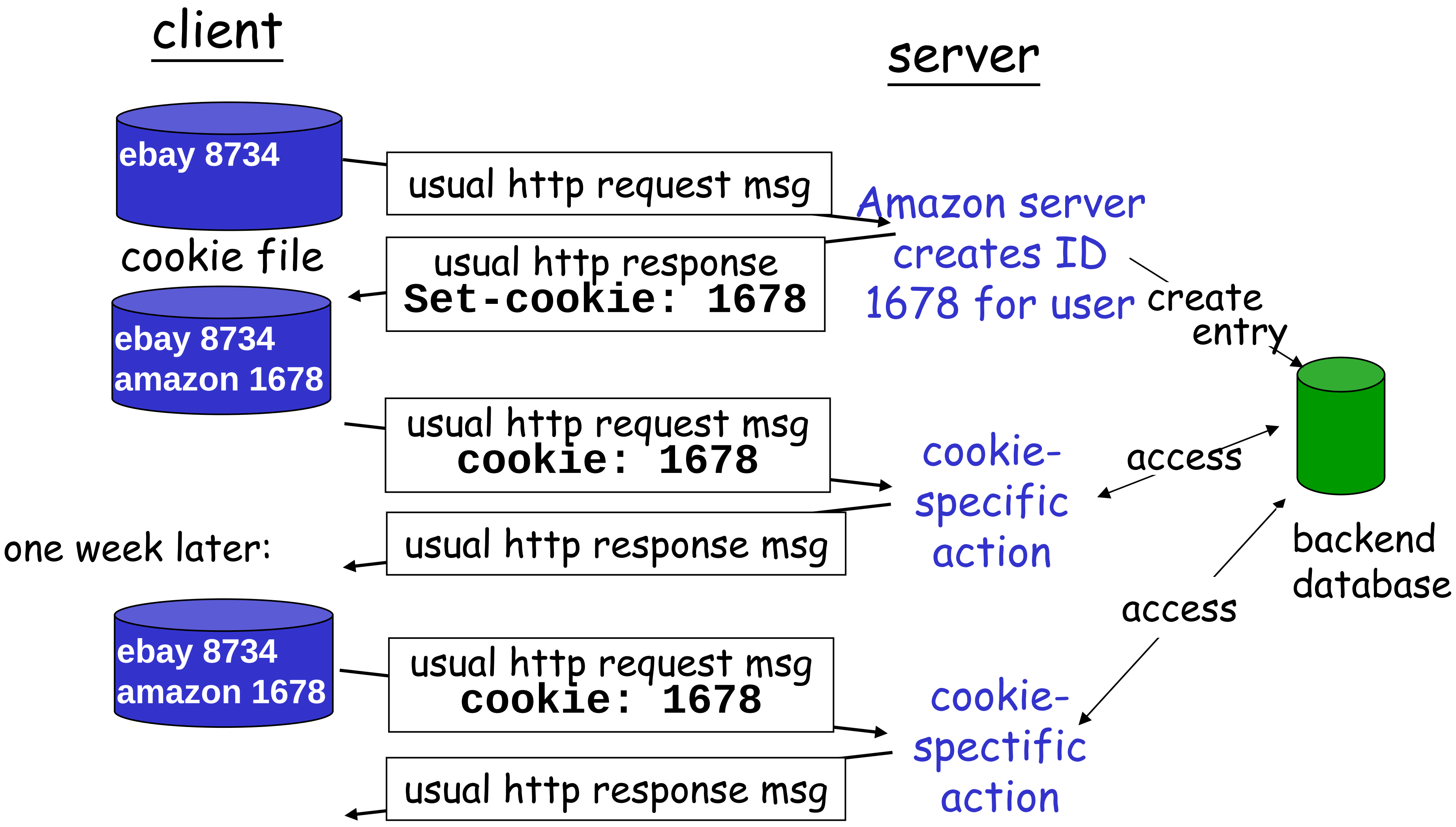
Four components: 四元件

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

Example:

- Susan always access Internet always from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies: keeping "state" (cont.) 保留狀態



Cookies (continued)

What cookies can bring:

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

aside

Cookies and privacy:

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

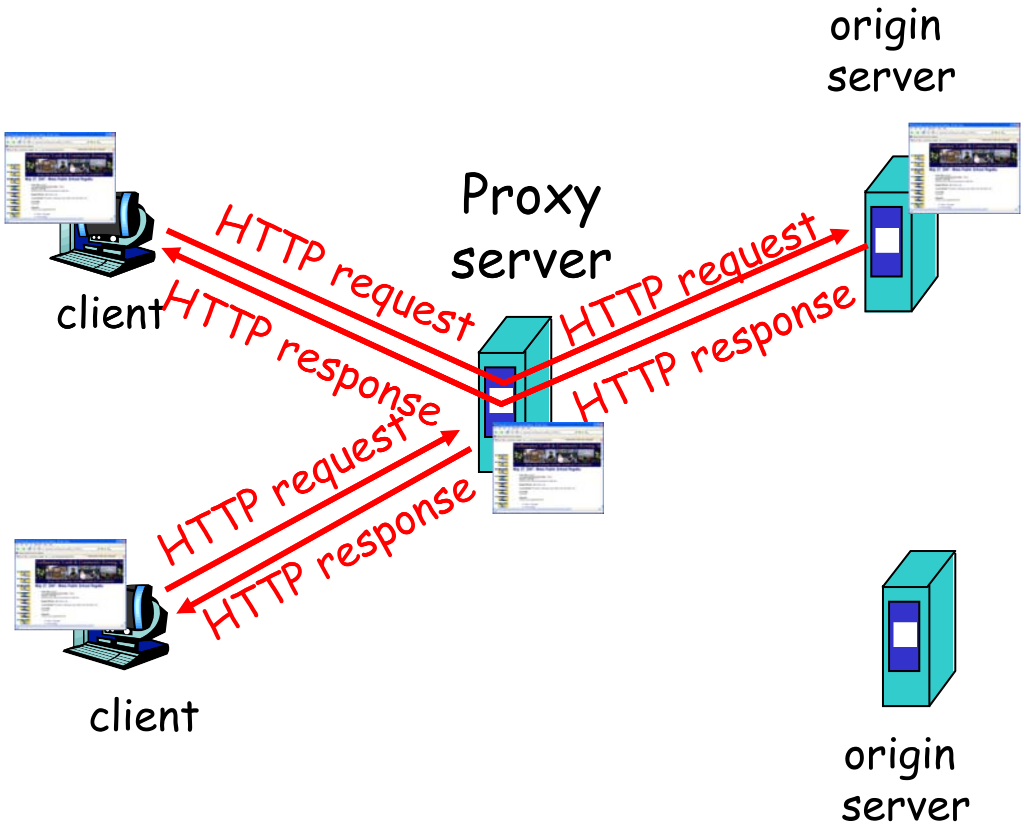
How to keep "state":

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

Web caches (proxy server) 代理伺服器

Goal: satisfy client request without involving origin server

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



More about Web caching 快取

- 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 減少回應時間
- reduce traffic on an institution's access link.減少網路流量
- Internet dense with caches: enables "poor" content providers to effectively deliver content (but so does P2P file sharing)

Caching example 快取的例子

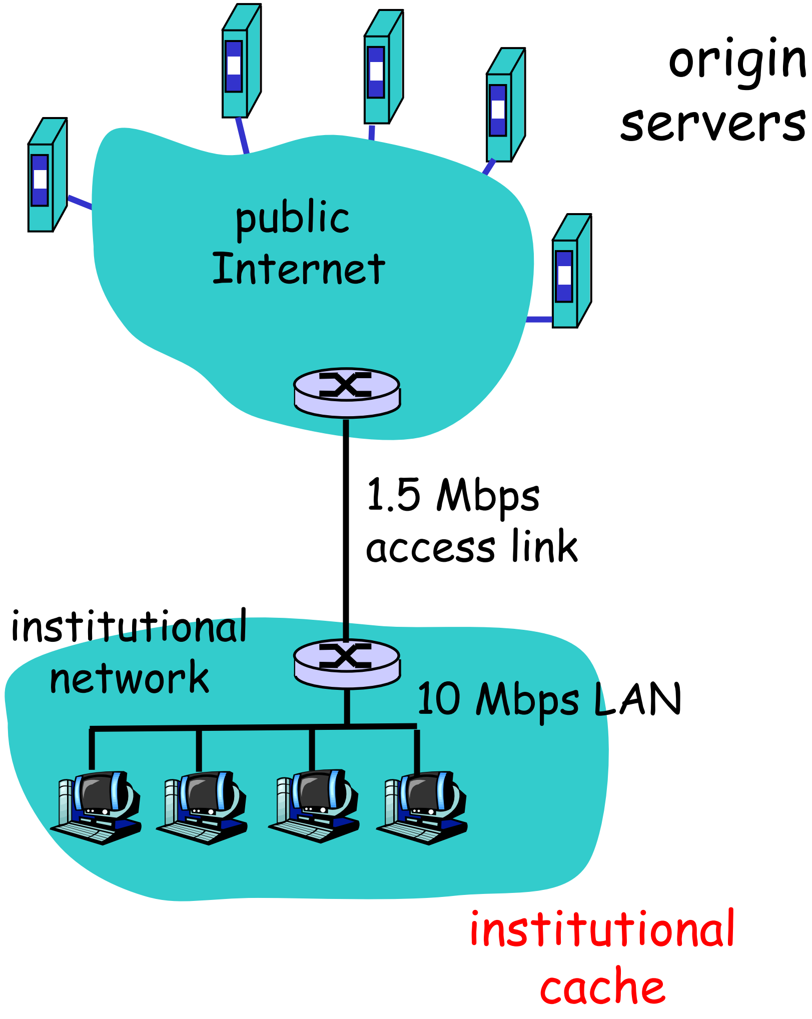
Assumptions 假設

- 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 結果

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

= 2 sec + minutes + milliseconds



Caching example (cont)

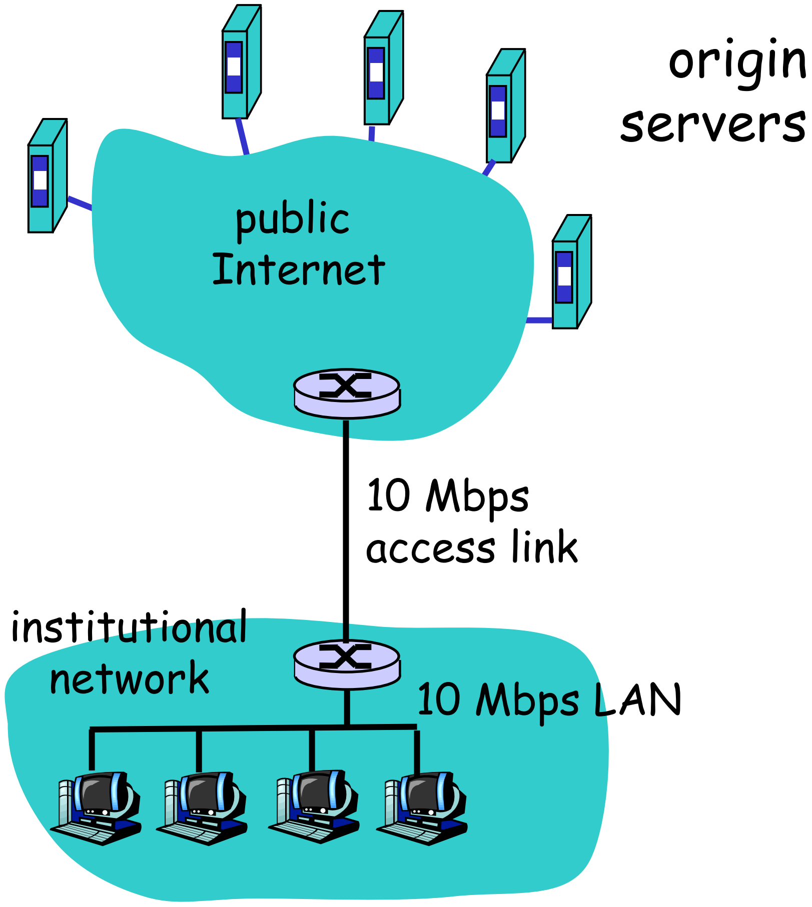
possible solution

- 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



institutional cache

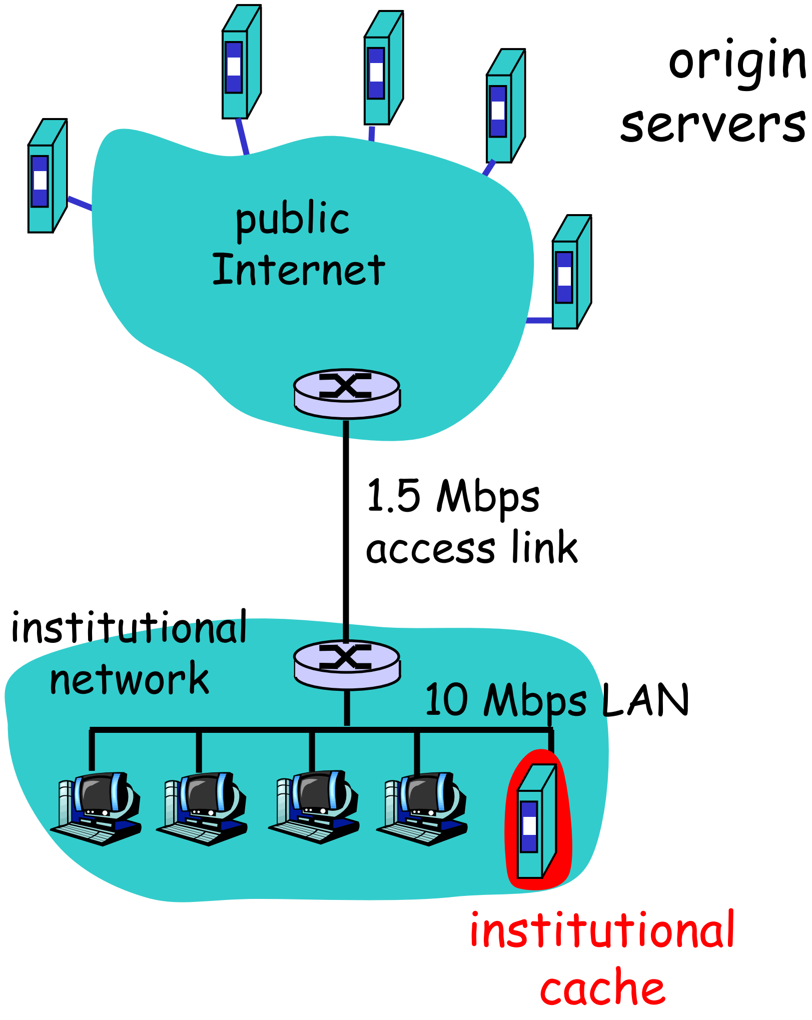
Caching example (cont)

possible solution: install cache

- suppose hit rate is 0.4

consequence

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

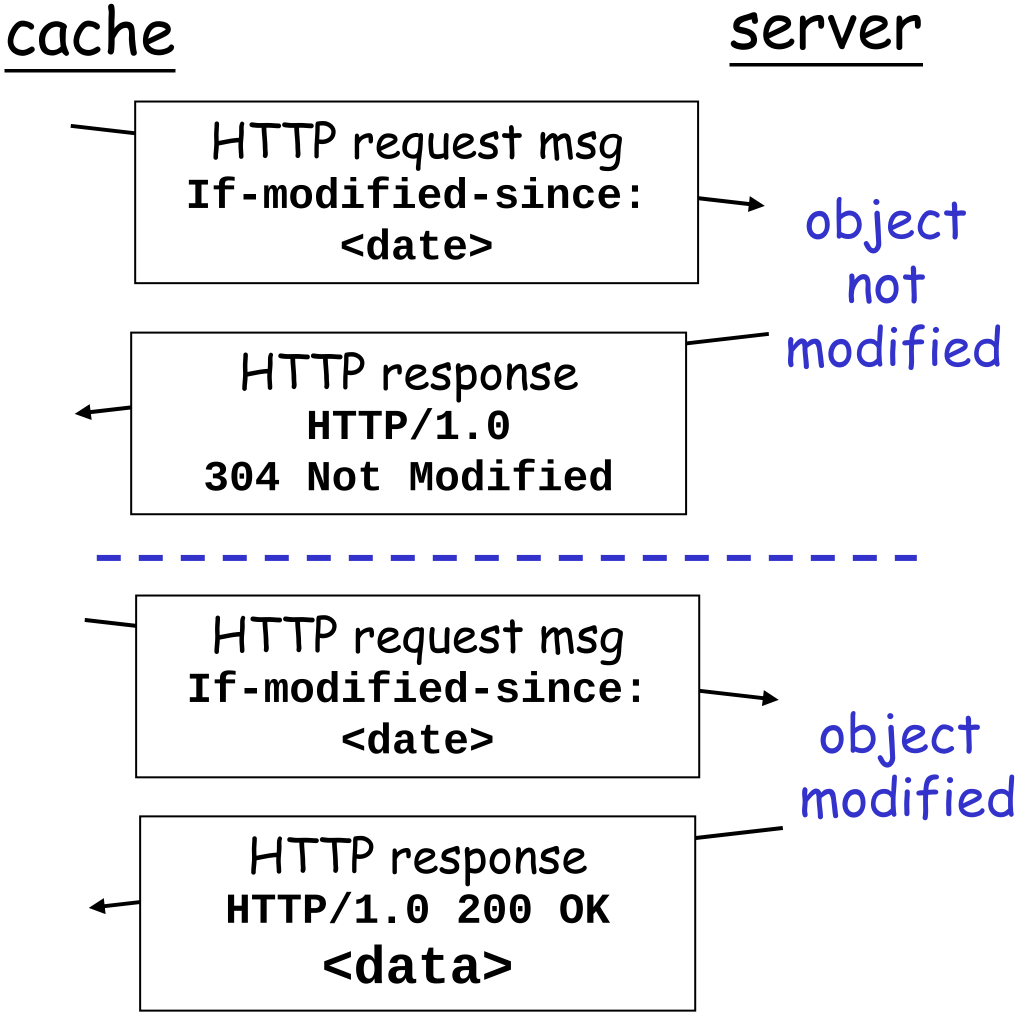


Conditional GET 條件式的GET

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

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

  `HTTP/1.0 304 Not Modified`



cache
server
HTTP request msg
If-modified-since: <date>
object not modified
HTTP response
HTTP/1.0 304 Not Modified
HTTP request msg
If-modified-since: <date>
object modified
HTTP response
HTTP/1.0 200 OK
<data>