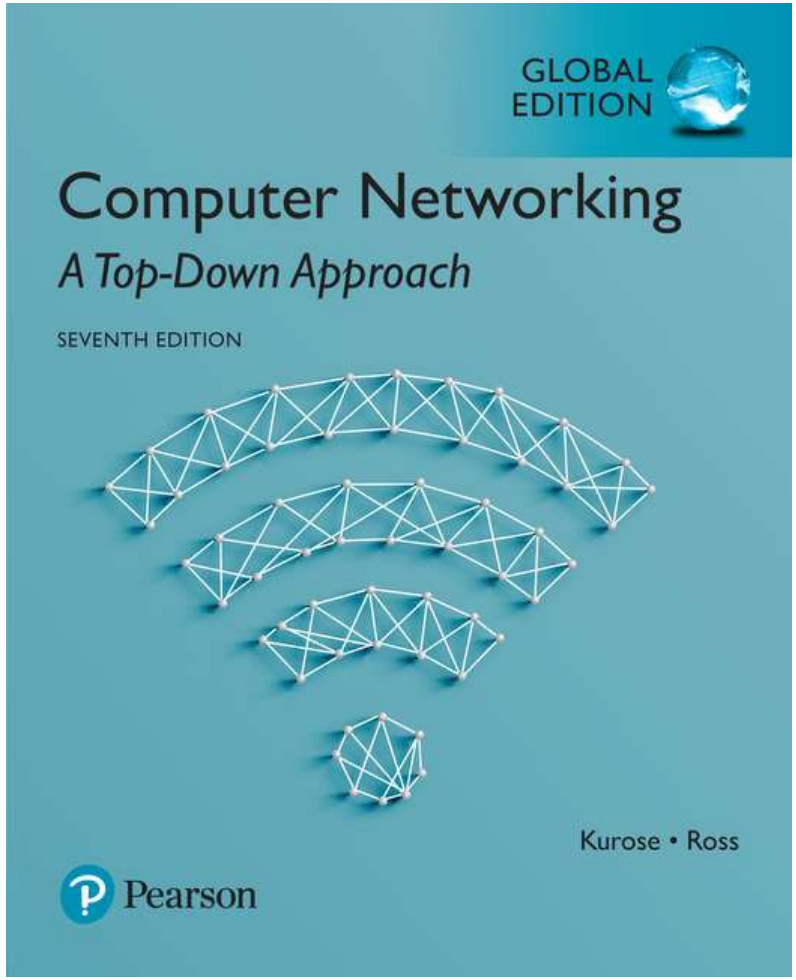
제8강
Web과 HTTP

Computer Networking: A Top Down Approach

컴퓨터 네트워크
(2019년 1학기)

박승철교수

한국기술교육대학교
컴퓨터공학부



Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks

2.7 socket programming with UDP and TCP

Pre-study Test :

1) 다음 중 웹 페이지를 식별하는 주소는?

① IP 주소
② Port 번호
③ URL
④ Domain 주소

2) 다음 중 URL의 주소 구조는?

① 네트워크 주소 + 호스트 주소
② IP 주소 + Port 번호
③ Domain 주소 + Port 번호
④ IP 주소 + Path 이름

3) 다음 중 HTTP에 대한 설명 중 틀린 것은?

① 서버와 클라이언트는 Port 번호 80을 사용한다.
② 서버는 항상 동작하고 있다.
③ 클라이언트에 의해 서비스가 개시된다.
④ 비상태형(stateless) 프로토콜이다.

4) 비지속성(non-persistent) TCP를 사용하는 HTTP의 웹 페이지(파일) 다운로드에 걸리는 시간은 얼마인가?

① RTT + 파일 전송 시간
② 2RTT + 파일 전송 시간
③ 3RTT + 파일 전송 시간
④ 4RTT + 파일 전송 시간

5) 하나의 파일에서 라인의 끝은 무엇으로 표현할까?

① EOL(End Of Line)을 표시하는 그래픽 문자
② EOL(End Of Line)을 표시하는 특수 문자
③ EOL(End Of Line)을 표시하는 제어 문자
④ EOL(End Of Line)을 표시하는 “EOL”

6) 다음 중 웹 페이지의 제어 정보만 다운로드할 때 사용하는 메소드는?

① HEAD
② GET
③ POST
④ PUT

7) 다음 중 웹 쿠키(cookies)에 대한 설명 중 틀린 것은?

① 클라이언트에 의해 할당된다.
② 상태 정보 유지를 위해 사용된다.
③ 클라이언트에 의해 제공된다.
④ 사용자 인증 및 인가(authorization)에 사용된다.

8) 웹 프락시 서버(proxy server)의 장점이 아닌 것은?

① 응답 시간을 단축시킨다.
② 네트워크 트래픽을 축소시킨다.
③ 최신 정보를 제공한다.
④ 보안 서비스 제공을 용이하게 한다.

Web and HTTP

First, a review…

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

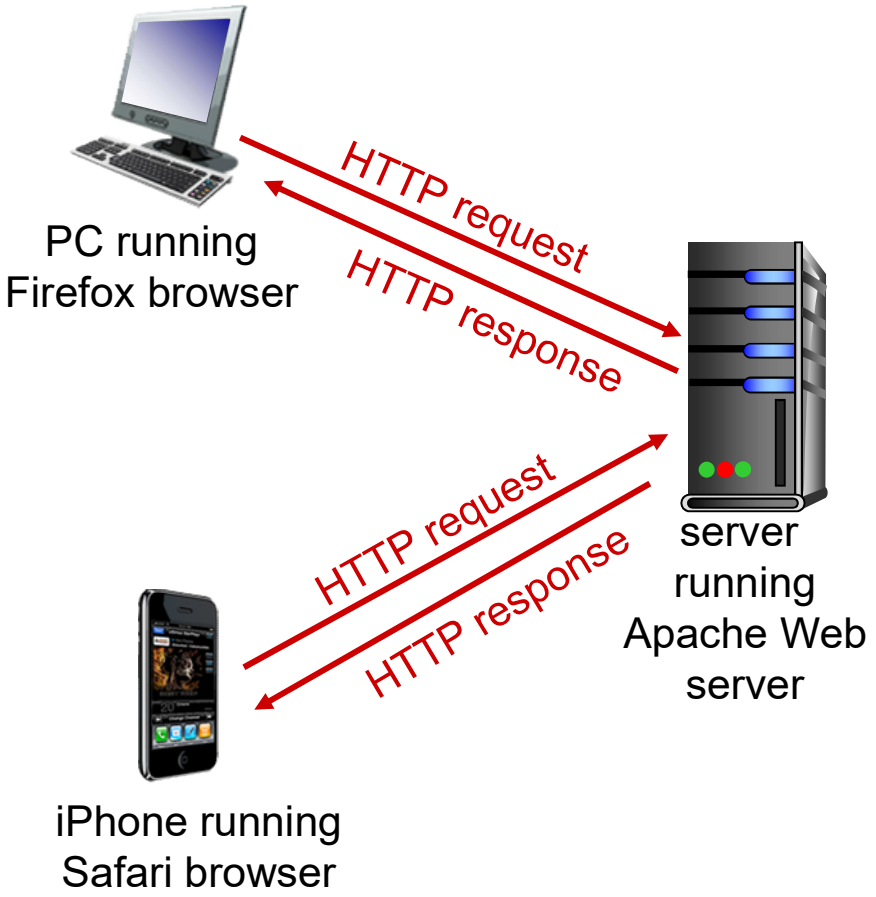
`www.someschool.edu/someDept/pic.gif`

host name | 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)

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

non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

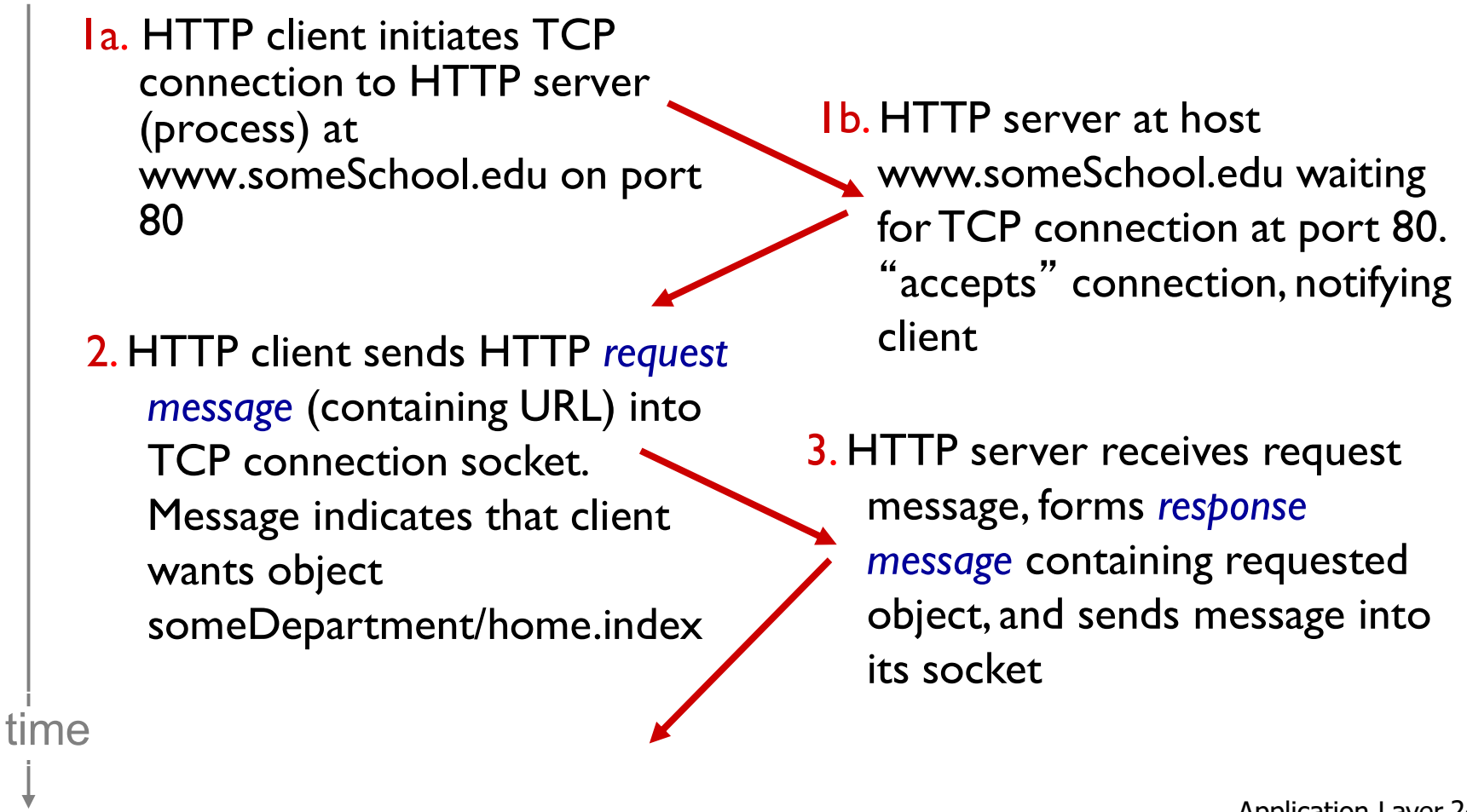
persistent HTTP

- multiple objects can be sent over single TCP connection between client, server

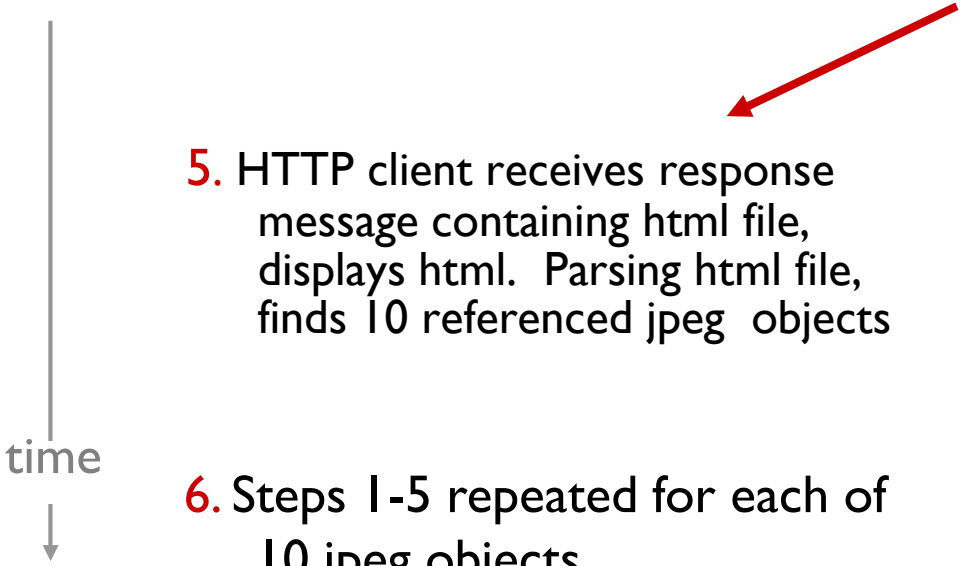
Non-persistent HTTP

suppose user enters URL:
`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)



Non-persistent HTTP (cont.)



4. HTTP server closes TCP connection.

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

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

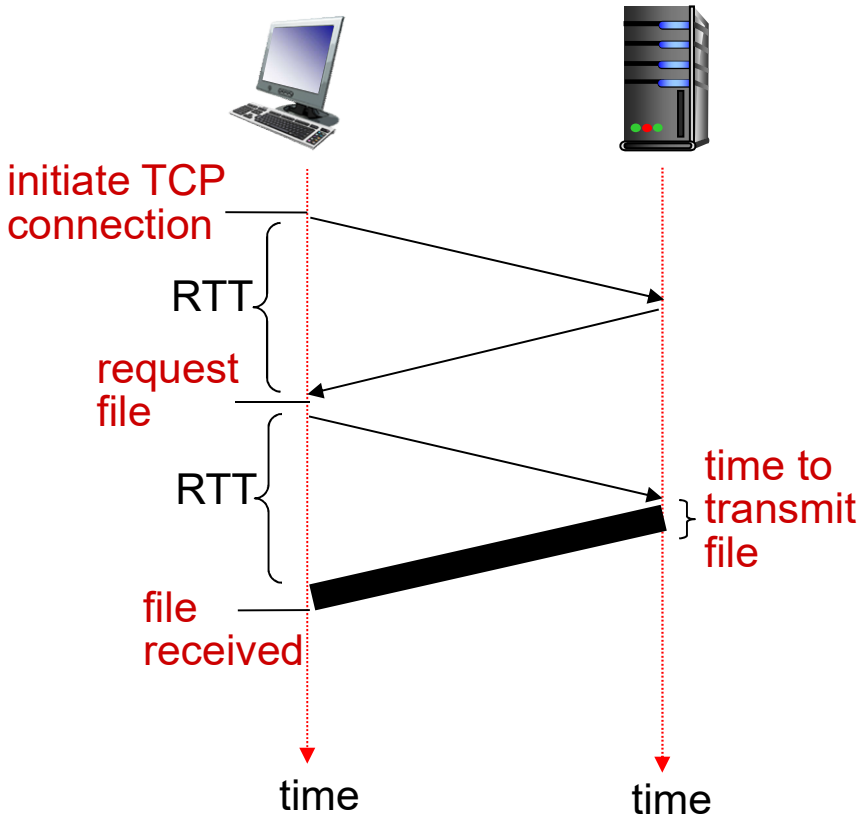
Non-persistent HTTP: response time

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

HTTP 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
- non-persistent HTTP response time =

  2RTT+ file transmission time



Persistent HTTP

non-persistent HTTP issues:

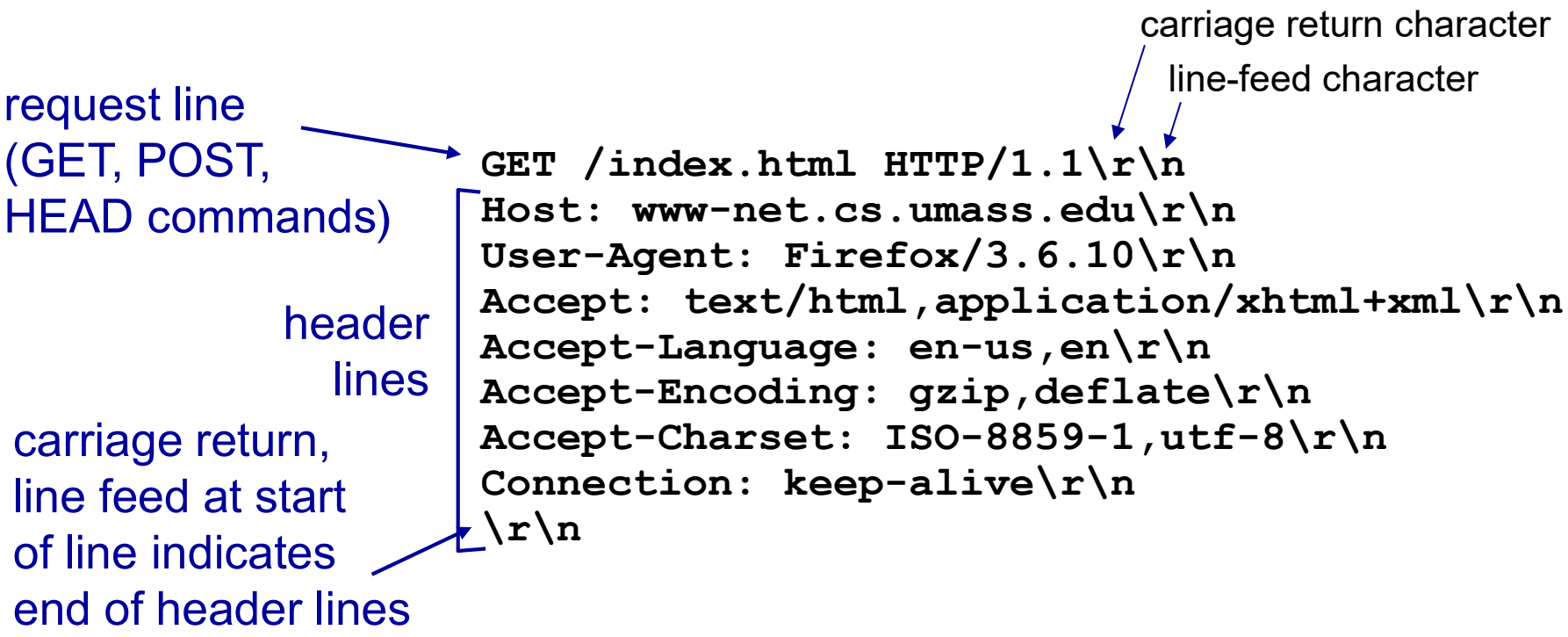
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

persistent HTTP:

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

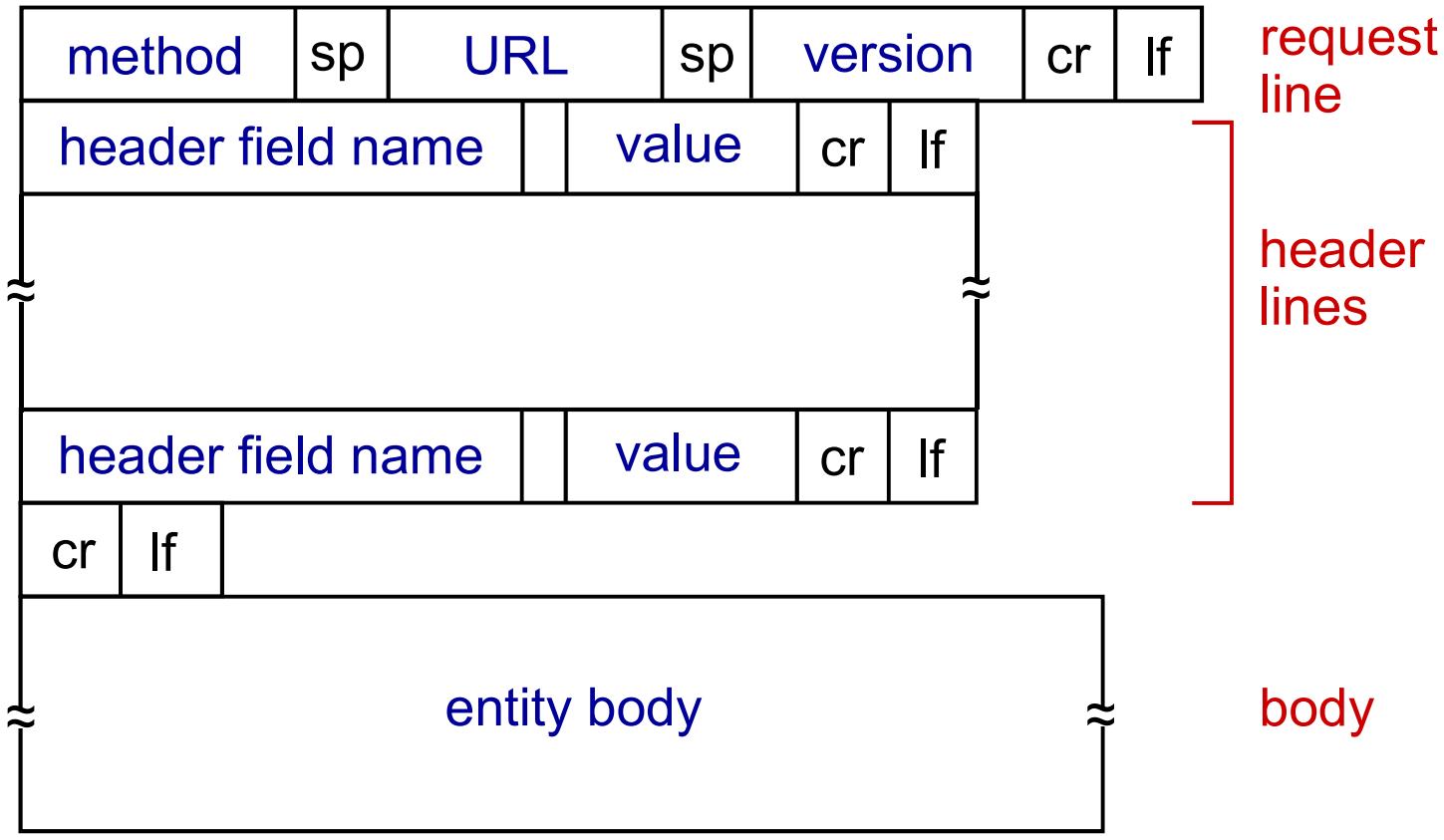
HTTP request message

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



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

HTTP request message: general format



ASCII 문자 코드

Bit Positions			7	0	0	0	0	1	1	1	1
			6	0	0	1	1	0	0	1	1
			5	0	1	0	1	0	1	0	1
4	3	2	1								
0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	STX	DC2	“	2	B	R	b	r
0	0	1	1	ETX	DC3	#	3	C	S	c	s
0	1	0	0	EOT	DC4	$	4	D	T	d	t
0	1	0	1	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	ACK	SYN	&	6	F	V	f	v
0	1	1	1	BEL	ETB	‘	7	G	W	g	w
1	0	0	0	BS	CAN	(	8	H	X	h	x
1	0	0	1	HT	EM	)	9	I	Y	i	y
1	0	1	0	LF	SUB	*	:	J	Z	j	z
1	0	1	1	VT	ESC	+	;	K	[	k	{
1	1	0	0	FF	FS	,	<	L	₩	l	|
1	1	0	1	CR	GS	-	=	M	]	m	}
1	1	1	0	SO	RS	.	>	N	^	n	~
1	1	1	1	SI	US	/	?	O	_	o	DEL

Uploading form input(동적 문서)

POST method:

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

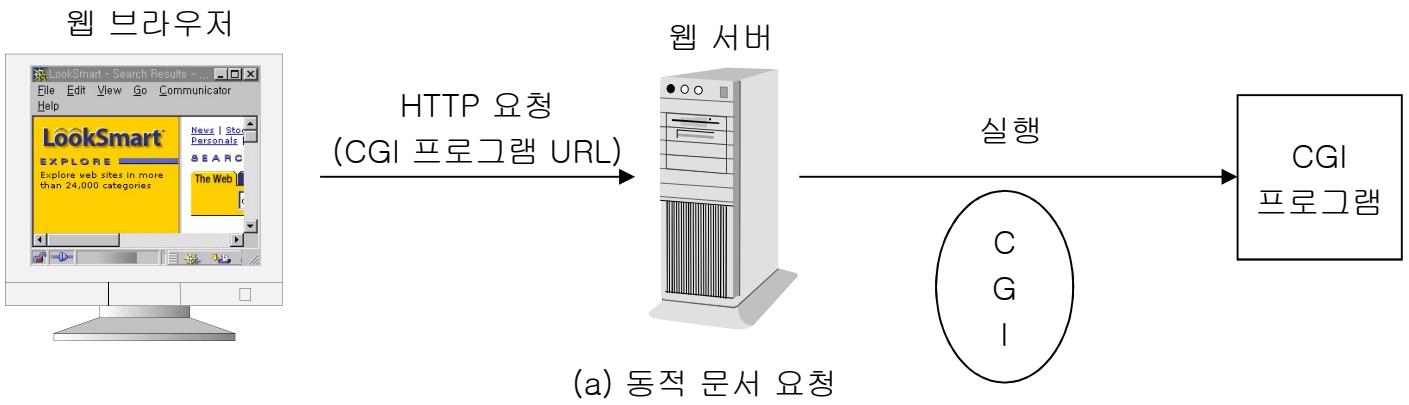
URL method:

- uses GET method
- input is uploaded in URL field of request line:

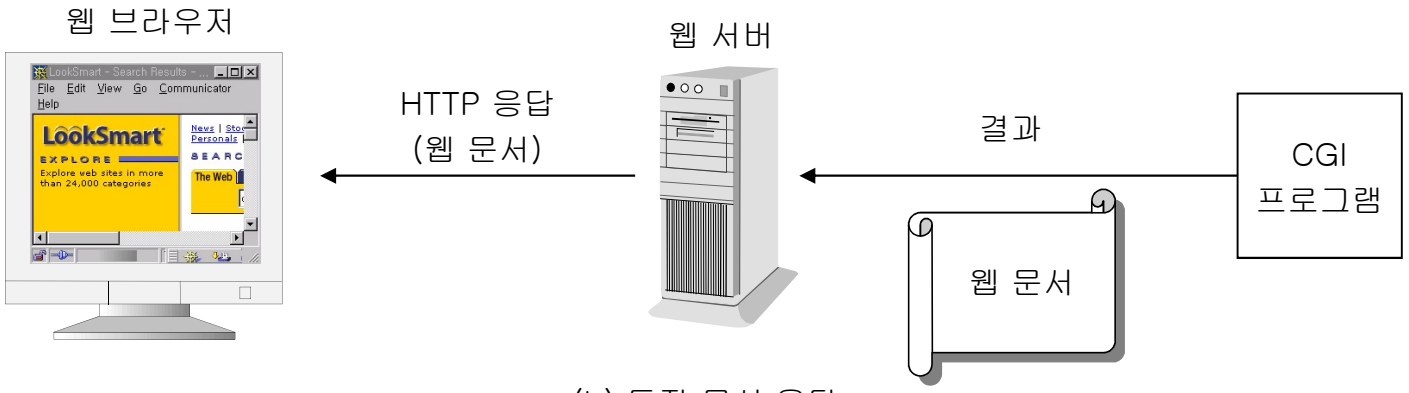
```
www.somesite.com/animalsearch?monkeys&banana
```

Uploading form input

- 동적 문서(dynamic document) 처리 절차



(a) 동적 문서 요청



(b) 동적 문서 응답

Uploading form input

- 웹 브라우저-CGI 프로그램간 데이터 전달 방식
  - GET 방식
  - POST 방식

Uploading form input

- GET 방식
  - 사용자가 입력한 데이터를 CGI 프로그램의 URL 정보에 연결하여 웹 서버에게 전달
  - 웹 서버는 이 정보를 환경 변수(QUERY_STRING)에 저장하여 CGI 프로그램이 수행될 때 사용
  - URL 정보와 사용자 입력 데이터의 구분은 물음표(?)에 의함
  - "name"과 "age" 변수를 가지는 HTML의 폼 태그를 이용하여 CGI 프로그램에 데이터를 전달하는 예
    - http://test.kut.ac.kr/cgi-bin/test-cgi?name=xxx&age=yy

Uploading form input

- GET 방식의 문제점
  - QUERY_STRING 환경변수의 크기가 한정되어 있기 때문에 CGI 프로그램으로 전달할 수 있는 데이터의 크기에 제한
  - 전달되는 데이터가 사용자들이 눈으로 확인할 수 있는 URL에 포함되어 전달되기 때문에 보안상 문제 발생 가능

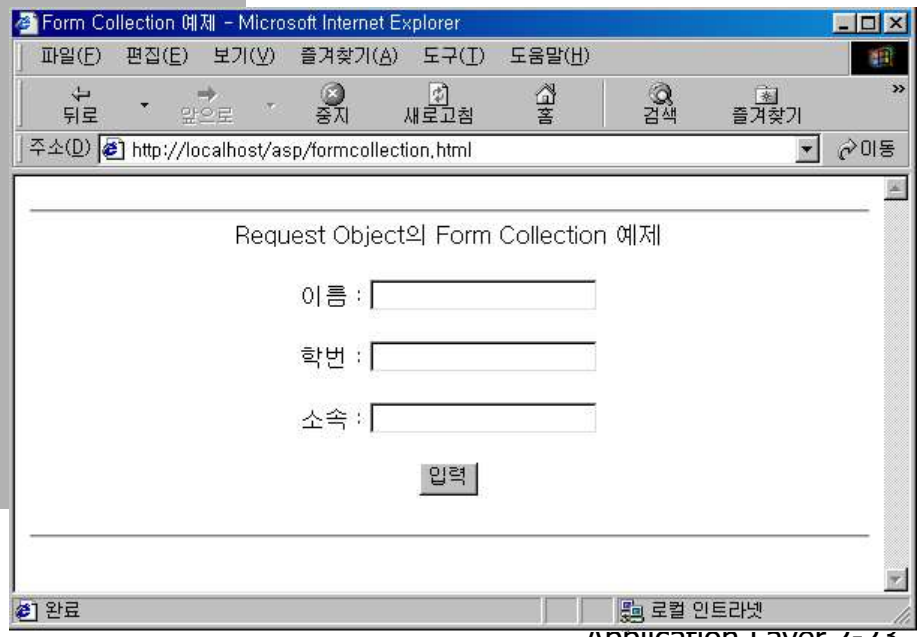
Uploading form input

- POST 방식
  - 표준 입력 방식과 같은 방식으로 사용자 데이터를 CGI 프로그램에 전달
  - 사용자 입력 데이터를 별도의 폼(Form)에 담아 HTTP 요청 메시지를 통해 웹 서버에 전달
  - CGI 프로그램은 폼에 있는 데이터를 표준 입력으로 사용
  - 사용하는 폼의 크기에 제한이 없기 때문에 크기가 큰 사용자 데이터를 CGI 프로그램에 전달하기에 적합
  - 보안 측면에서 GET 방식보다 우수
  - CGI 프로그램 전용 언어를 사용하여 CGI 프로그램을 작성하는 대부분의 경우 POST 방식을 사용

Uploading form input

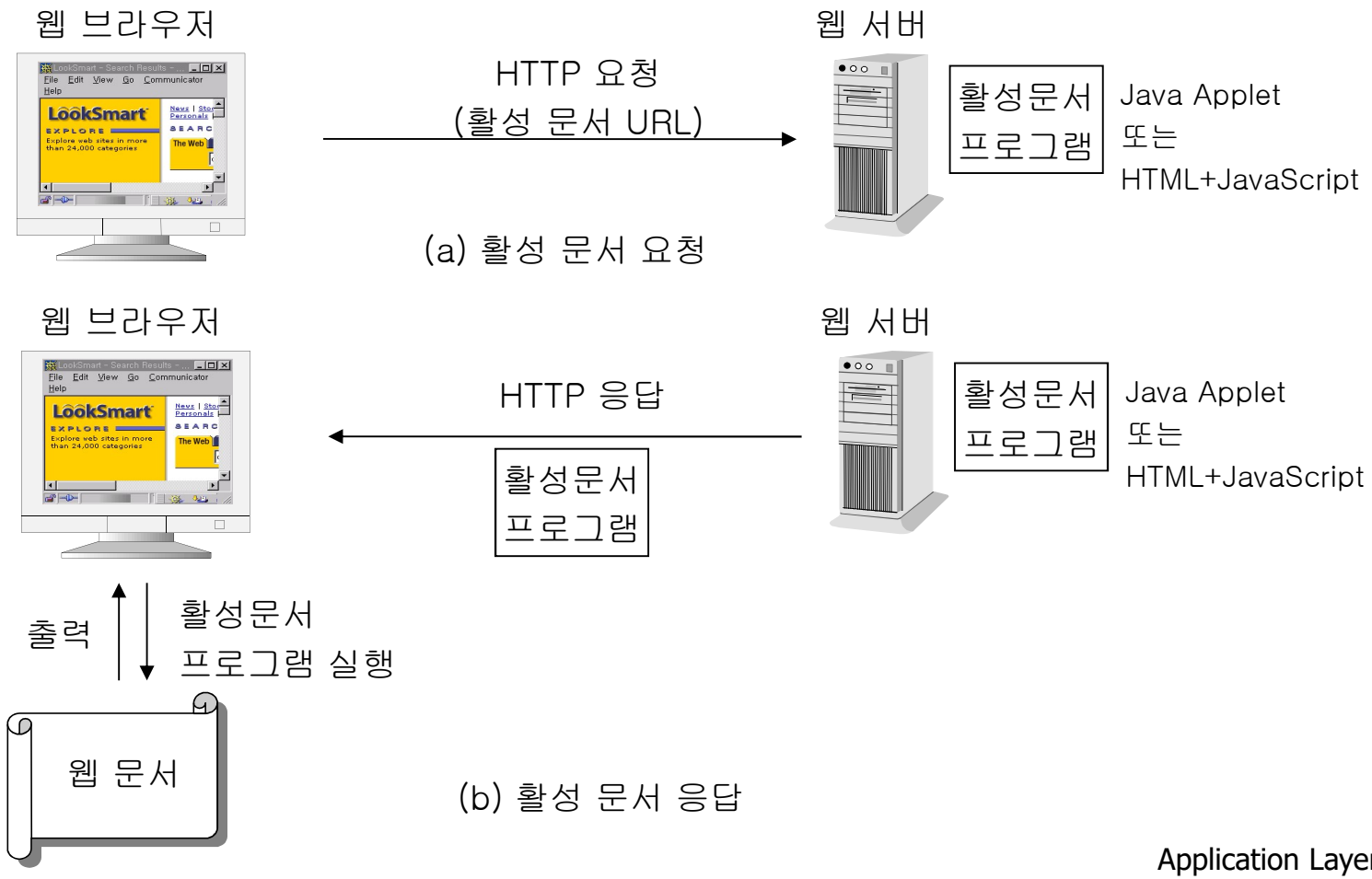
- POST 방식 CGI 프로그램 호출을 포함하는 HTML 예제

```
 1 : <html>
 2 :   <head><title>Form Collection 예제</title></head>
 3 :   <body  bgcolor="white">
 4 :     <center>
 5 :
 6 :     <hr>
 9 :      Request Object의 Form Collection 예제<br><br>
 8 :     <form method = “post”  action = “FormCollection.asp”>
 9 :     이름 :
10 :     <input type="text" name="이름">
11 :     <br><br>
12 :     학번 :
13 :     <input type="text" name="학번">
14 :     <br><br>
15 :     소속 :
16 :     <input type="text" name="소속">
17 :    <br><br>
18 :    <input type="submit" value="입력">
19 :    </form>
20 :    <hr>
21 :   </body>
22 : </html>
```



활성 문서(Active Document) 처리

- 활성 문서 처리 절차



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

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
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

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 msg

301 Moved Permanently

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

400 Bad Request

- request msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

User-server state: cookies

many Web sites use cookies

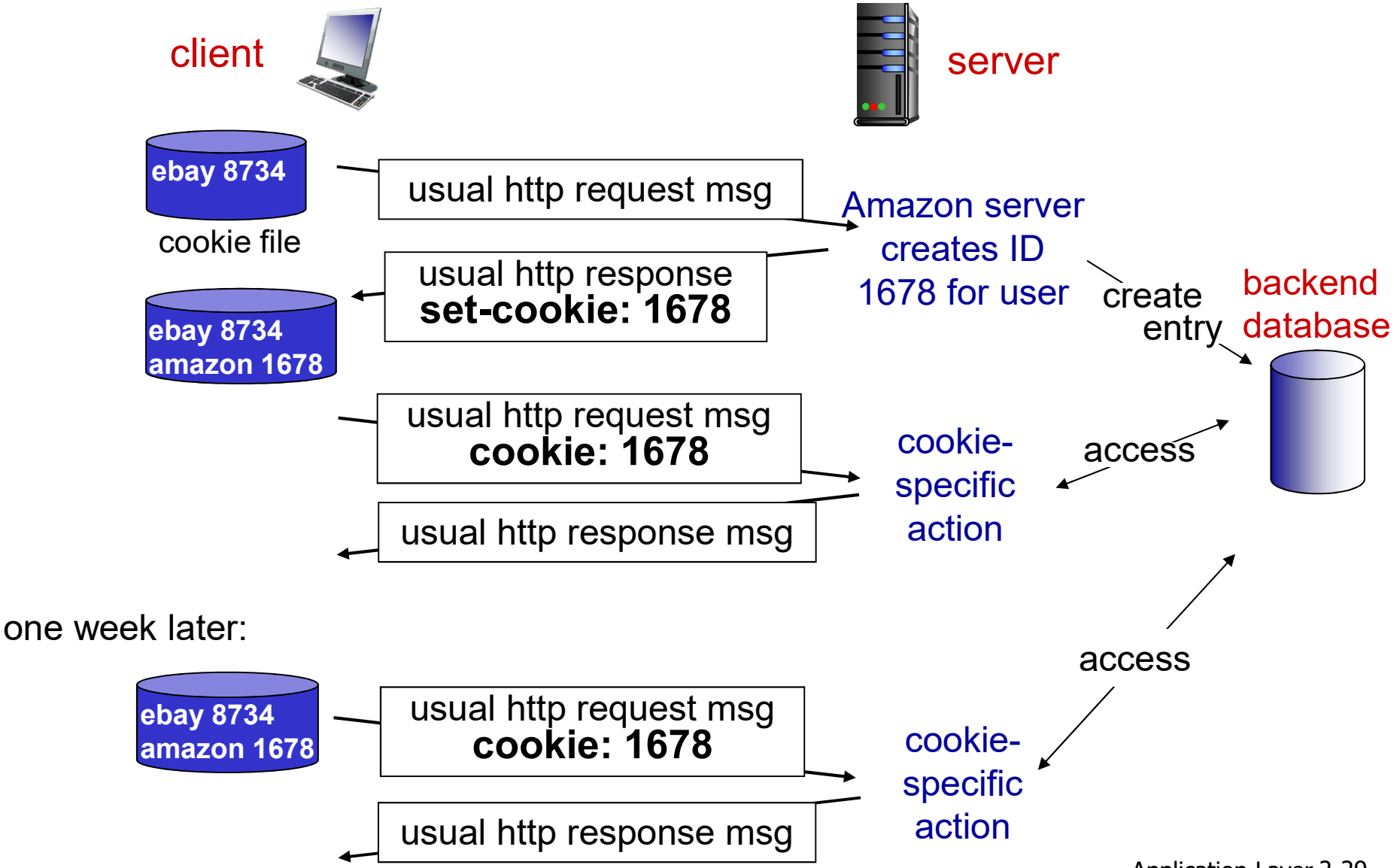
four components:

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 always access Internet 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.)



Cookies (continued)

what cookies can be used for:

- 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

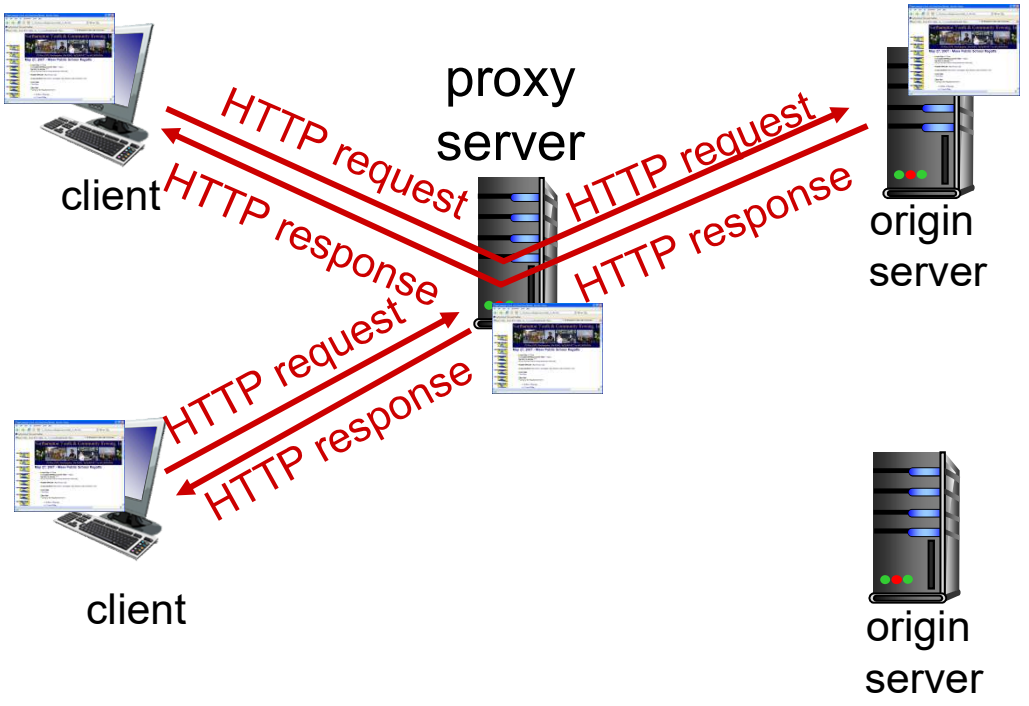
how to keep "state":

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

Web caches (proxy server)

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

- 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
- reduce traffic on an institution's access link
- Internet dense with caches: enables "poor" content providers to effectively deliver content (so too does P2P file sharing)

Caching example:

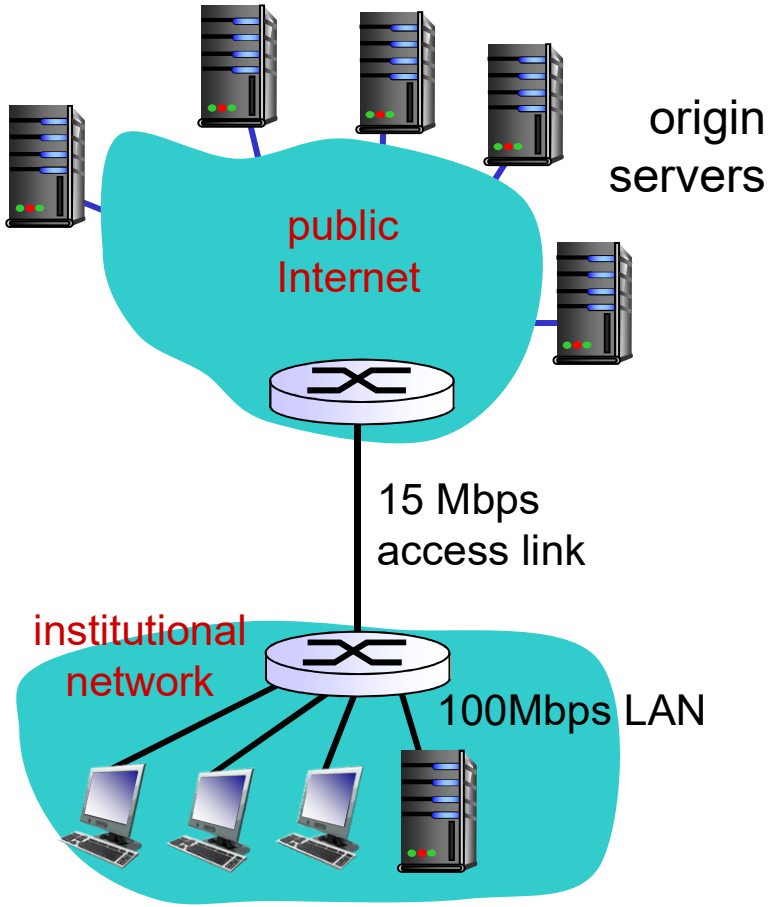
assumptions:

- avg object size: 1M bits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 15 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 15 Mbps

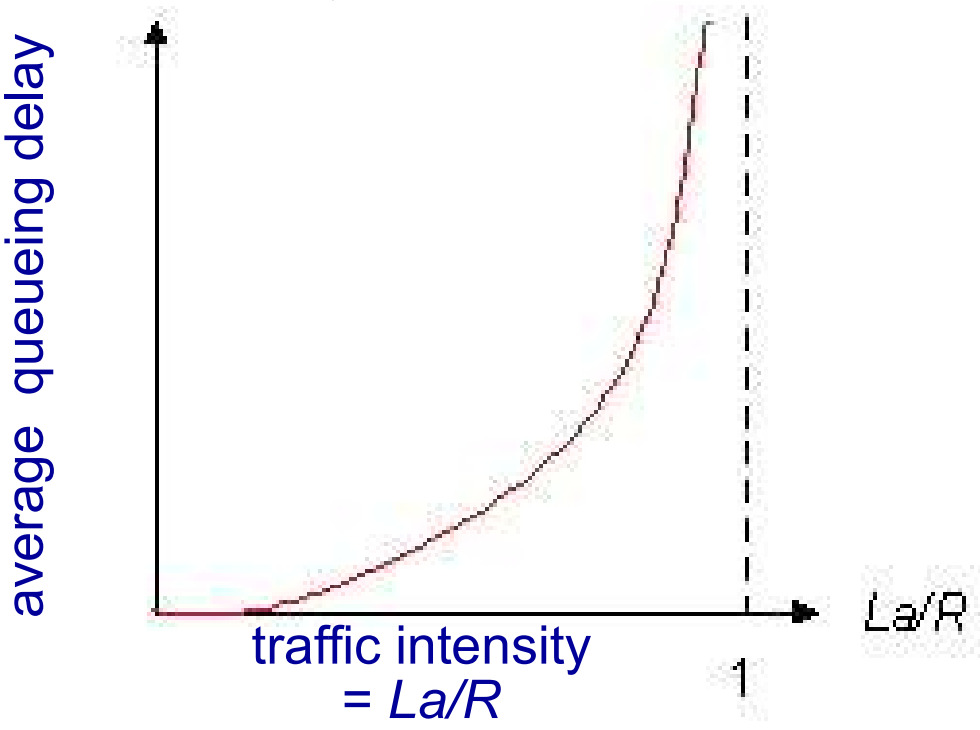
consequences:

- LAN utilization: 15(intensity 0.15)%
- access link utilization = 100%
- total delay = Internet delay + access delay + LAN delay

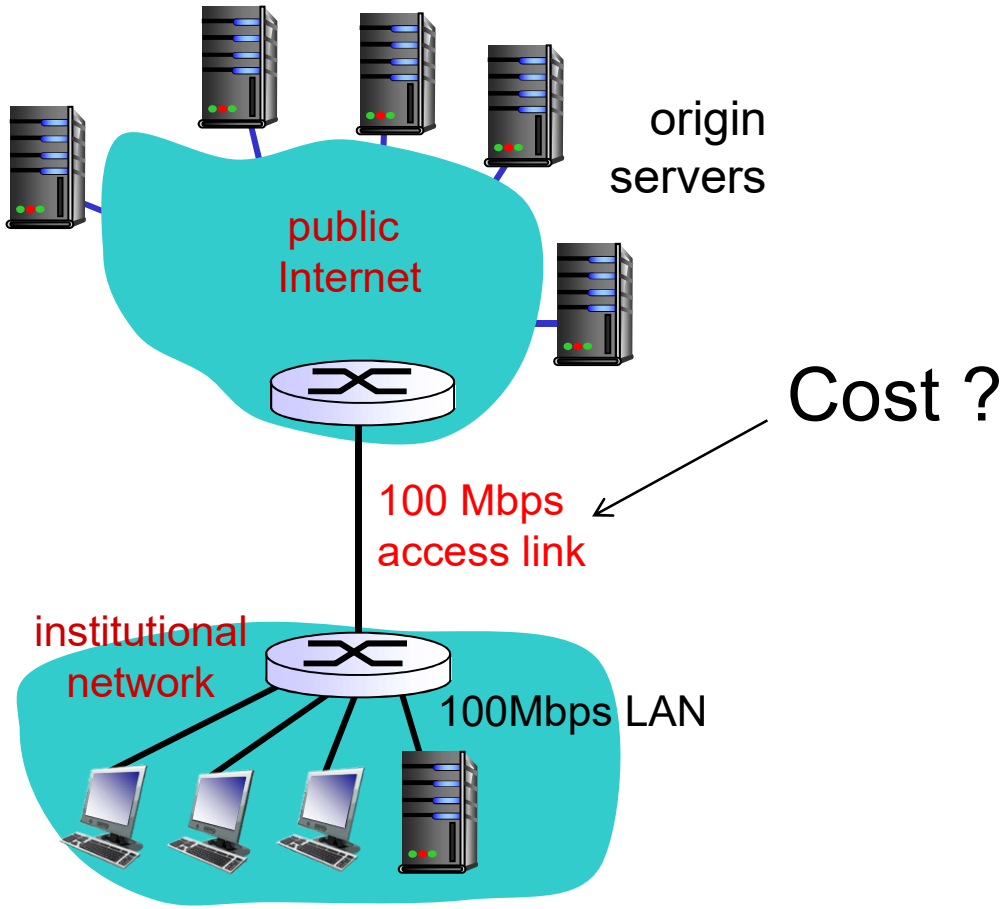
  = 2 sec + minutes + usecs



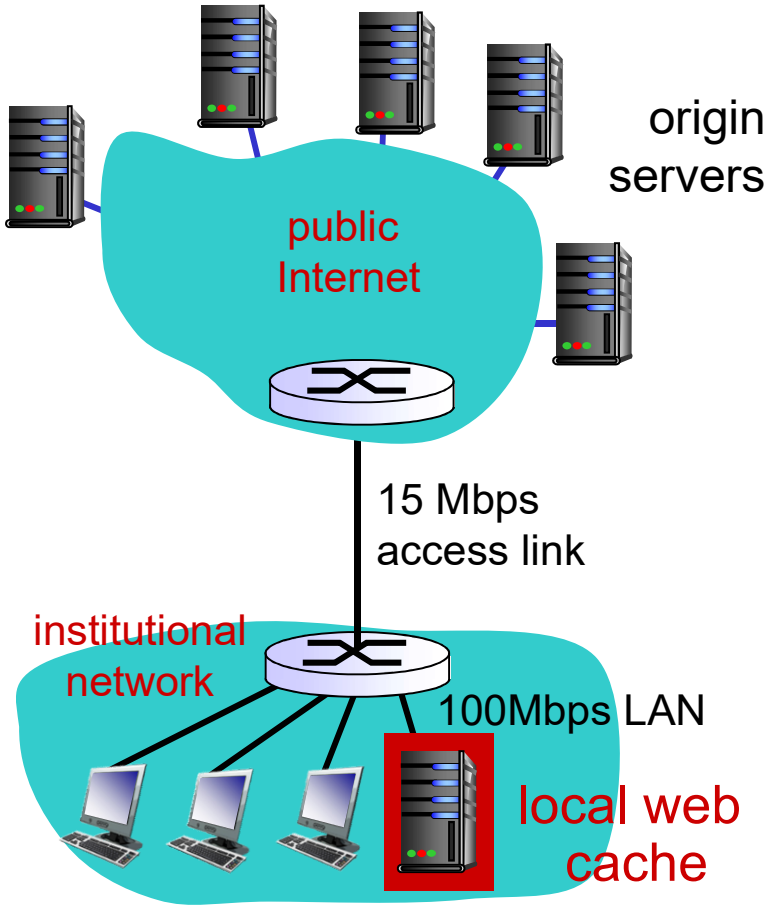
Caching example:



Caching example:



Caching example: install local cache

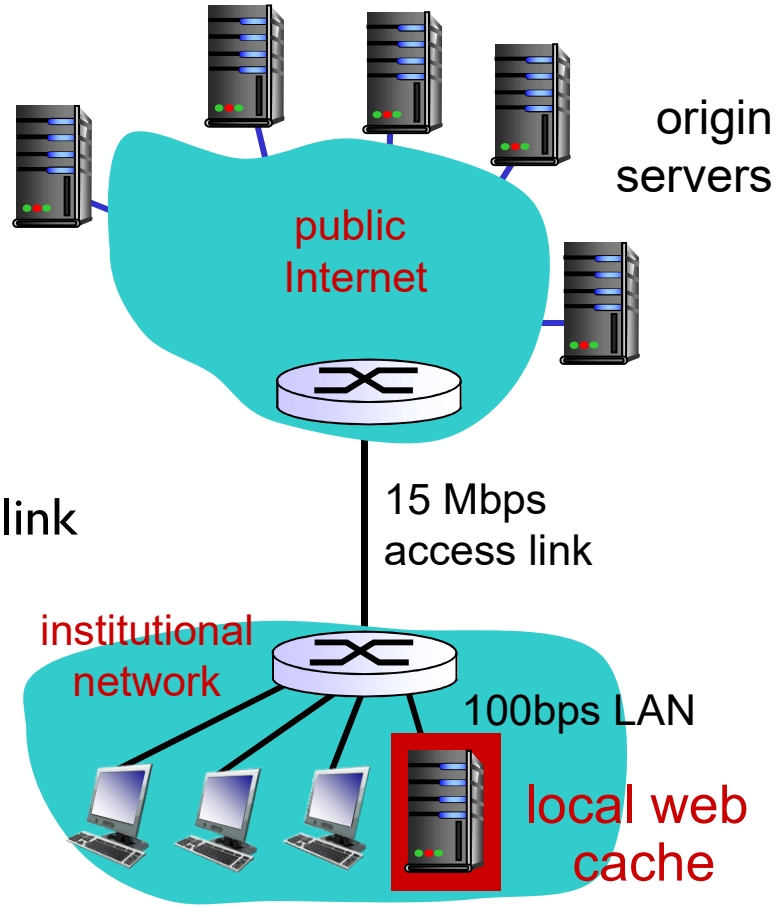


Caching example: install local cache

Calculating access link utilization, delay with cache:

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

  = 0.6*15 Mbps = 9 Mbps
  - utilization = 9/16 = 0.6
- total delay
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs

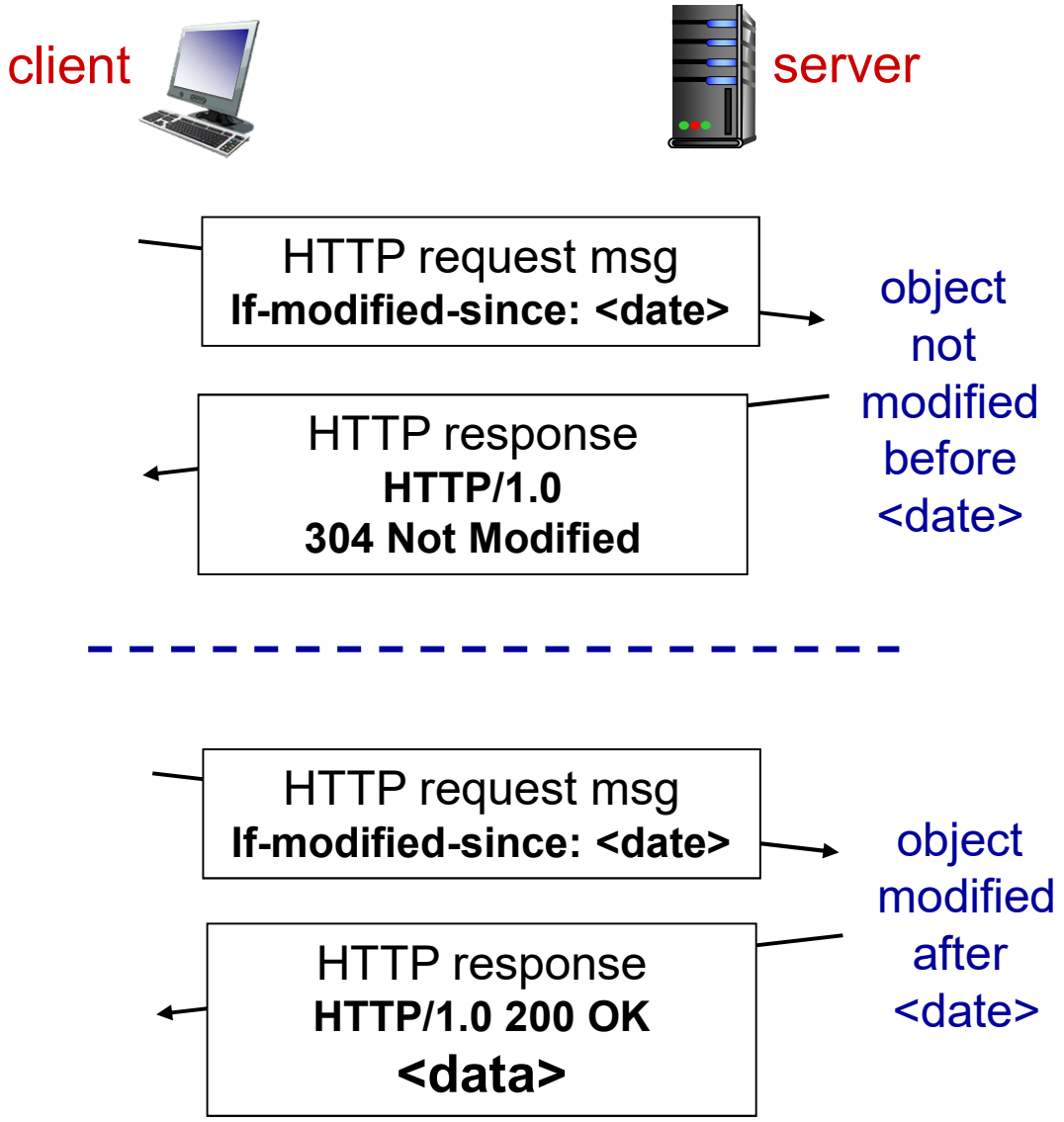


Conditional GET

- *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`



Pre-study Test :

1) 다음 중 웹 페이지를 식별하는 주소는?

① IP 주소
② Port 번호
③ URL
④ Domain 주소

2) 다음 중 URL의 주소 구조는?

① 네트워크 주소 + 호스트 주소
② IP 주소 + Port 번호
③ Domain 주소 + Port 번호
④ IP 주소 + Path 이름

3) 다음 중 HTTP에 대한 설명 중 틀린 것은?

① 서버와 클라이언트는 Port 번호 80을 사용한다.
② 서버는 항상 동작하고 있다.
③ 클라이언트에 의해 서비스가 개시된다.
④ 비상태형(stateless) 프로토콜이다.

4) 비지속성(non-persistent) TCP를 사용하는 HTTP의 웹 페이지(파일) 다운로드에 걸리는 시간은 얼마인가?

① RTT + 파일 전송 시간
② 2RTT + 파일 전송 시간
③ 3RTT + 파일 전송 시간
④ 4RTT + 파일 전송 시간

5) 하나의 파일에서 라인의 끝은 무엇으로 표현할까?

① EOL(End Of Line)을 표시하는 그래픽 문자
② EOL(End Of Line)을 표시하는 특수 문자
③ EOL(End Of Line)을 표시하는 제어 문자
④ EOL(End Of Line)을 표시하는 “EOL”

6) 다음 중 웹 페이지의 제어 정보만 다운로드할 때 사용하는 메소드는?

① HEAD
② GET
③ POST
④ PUT

7) 다음 중 웹 쿠키(cookies)에 대한 설명 중 틀린 것은?

① 클라이언트에 의해 할당된다.
② 상태 정보 유지를 위해 사용된다.
③ 클라이언트에 의해 제공된다.
④ 사용자 인증 및 인가(authorization)에 사용된다.

8) 웹 프락시 서버(proxy server)의 장점이 아닌 것은?

① 응답 시간을 단축시킨다.
② 네트워크 트래픽을 축소시킨다.
③ 최신 정보를 제공한다.
④ 보안 서비스 제공을 용이하게 한다.