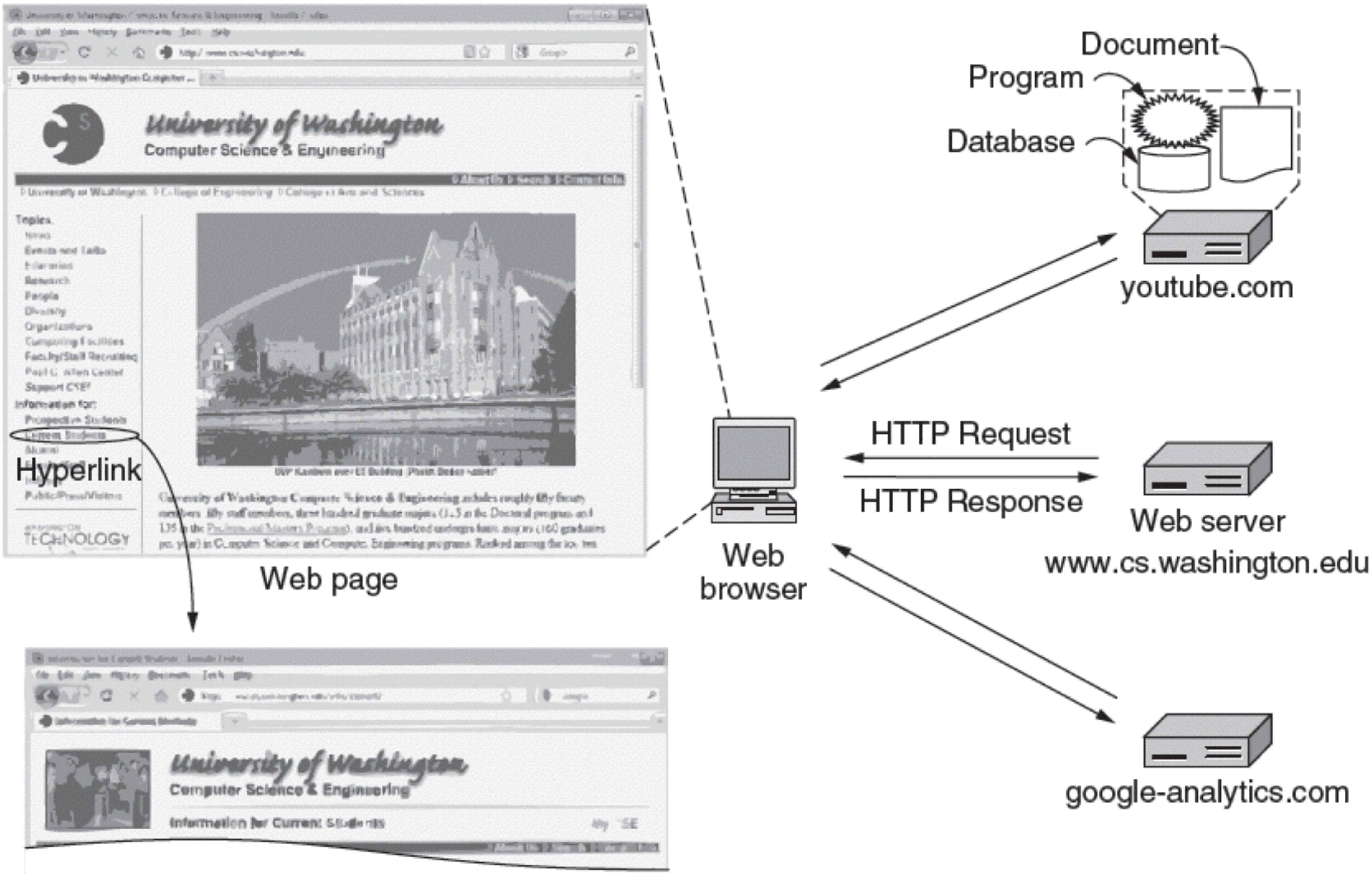
Couche application

HTTP

Computer Networks. Tanenbaum
Computer Networking. Kurose&Ross

Carole Delporte

Http: Principes



Web et HTTP

- *Une page web* contient des *objets*
- Objet : fichier HTML, images JPEG, applet, fichiers audio,…
- Une page web page consiste en un fichier de base *HTML* contenant des objets référencés
- Chaque objet est adressable par une *URL (Uniform Resource Locator)*

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

URL:

- Example: http://www.phdcomics.com/comics.php

Protocol Server Page on server

Our focus

Name	Used for	Example
http	Hypertext (HTML)	http://www.ee.uwa.edu/~rob/
https	Hypertext with security	https://www.bank.com/accounts/
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
file	Local file	file:///usr/suzanne/prog.c
mailto	Sending email	mailto:JohnUser@acm.org
rtsp	Streaming media	rtsp://youtube.com/montypython.mpg
sip	Multimedia calls	sip:eve@adversary.com
about	Browser information	about:plugins

Common URL protocols

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

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

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

Overview

Steps a client (browser) takes to follow a hyperlink:

- Determine the protocol (HTTP)
- Ask DNS for the IP address of server
- Make a TCP connection to server
- Send request for the page; server sends it back
- Fetch other URLs as needed to display the page
- Close idle TCP connections

Steps a server takes to serve pages:

- Accept a TCP connection from client
- Get page request and map it to a resource (e.g., file name)
- Get the resource (e.g., file from disk)
- Send contents of the resource to the client.
- Release idle TCP connections

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)

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

Carole Delporte

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

time

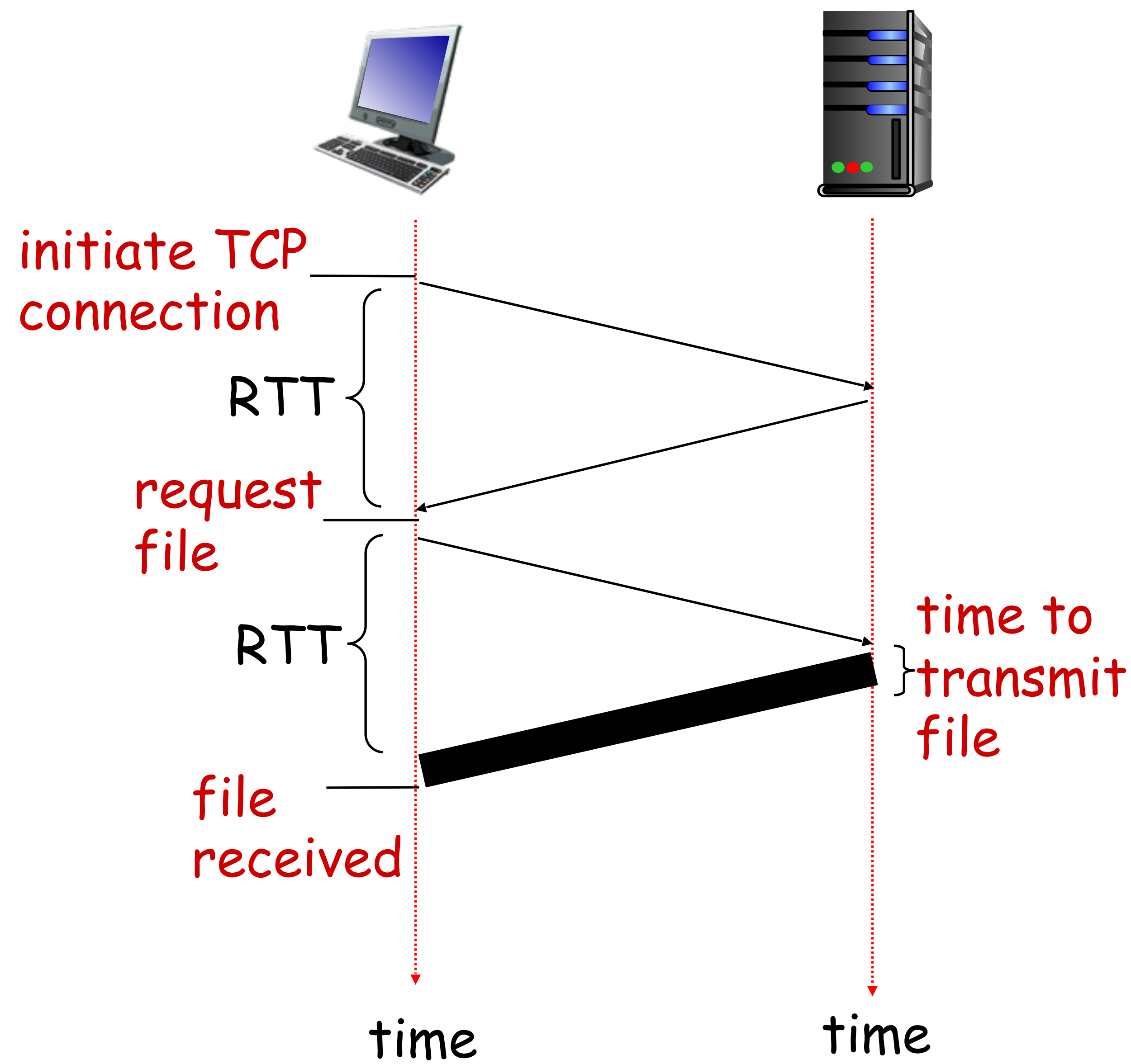
6. Steps 1-5 repeated for each of 10 jpeg objects

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)

carriage return character

line-feed character

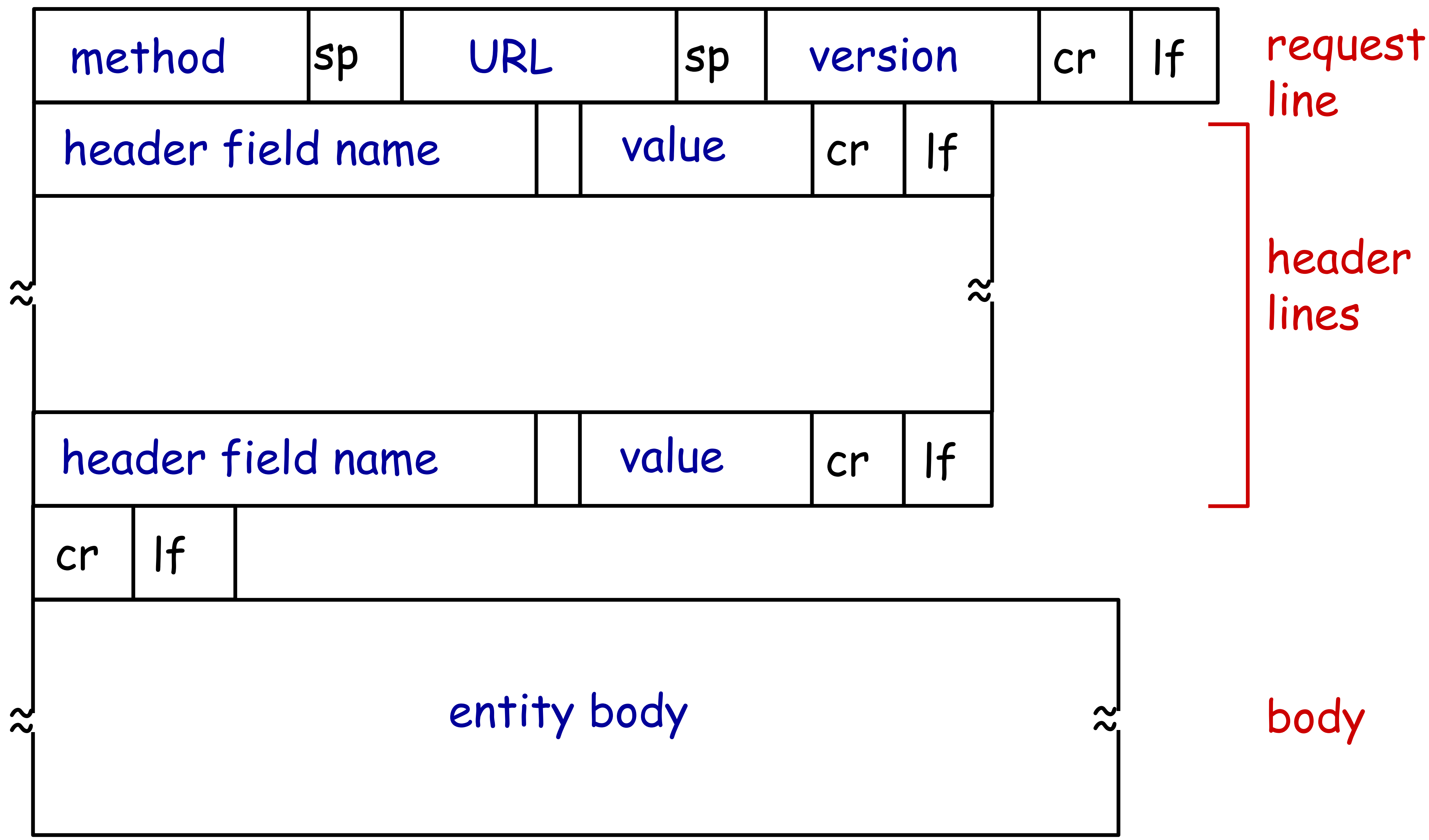
request line (GET, POST, HEAD commands)

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
```

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

HTTP request message: general format



HTTP

Headers:

Function	Example Headers
Browser capabilities (client → server)	User-Agent, Accept, Accept-Charset, Accept-Encoding, Accept-Language
Caching related (mixed directions)	If-Modified-Since, If-None-Match, Date, Last-Modified, Expires, Cache-Control, ETag
Browser context (client → server)	Cookie, Referer, Authorization, Host
Content delivery (server → client)	Content-Encoding, Content-Length, Content-Type, Content-Language, Content-Range, Set-Cookie

Uploading form input

POST method:

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

URL method:

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

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

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

Request methods.

Fetch a page → GET

Used to send input data to a server program → POST

Method	Description
GET	Read a Web page
HEAD	Read a Web page's header
POST	Append to a Web page
PUT	Store a Web page
DELETE	Remove the Web page
TRACE	Echo the incoming request
CONNECT	Connect through a proxy
OPTIONS	Query options for a page

HTTP response message

status line
(protocol
status code
status phrase)

```
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 ...
```

header
lines

data, e.g.,
requested
HTML file

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`

HTTP

Response codes tell the client how the request fared:

Code	Meaning	Examples
1xx	Information	100 = server agrees to handle client's request
2xx	Success	200 = request succeeded; 204 = no content present
3xx	Redirection	301 = page moved; 304 = cached page still valid
4xx	Client error	403 = forbidden page; 404 = page not found
5xx	Server error	500 = internal server error; 503 = try again later

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

telnet www.irif.fr 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  /bla  HTTP/1.1
Host: www.irif.fr
```

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!

```
$ telnet www.irif.fr 80
Trying 81.194.27.176...
Connected to www.irif.fr.
Escape character is '^]'.
GET   /blaa  HTTP/1.1
Host: www.irif.fr

HTTP/1.1 301 Moved Permanently
Date: Mon, 10 Oct 2015 10:32:29 GMT
Server: Apache/2.4.23 ( FreeBSD) PHP/5.6.24 OpenSSL/0.9.8sz-freebsd
Location: https://www.irif.fr/bla
Content-Length: 231
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE html PUBLIC « -//IETF//DTD HTML 2.0 EN » >
<html><head>…..
```

User-server state: cookies

many Web sites use cookies

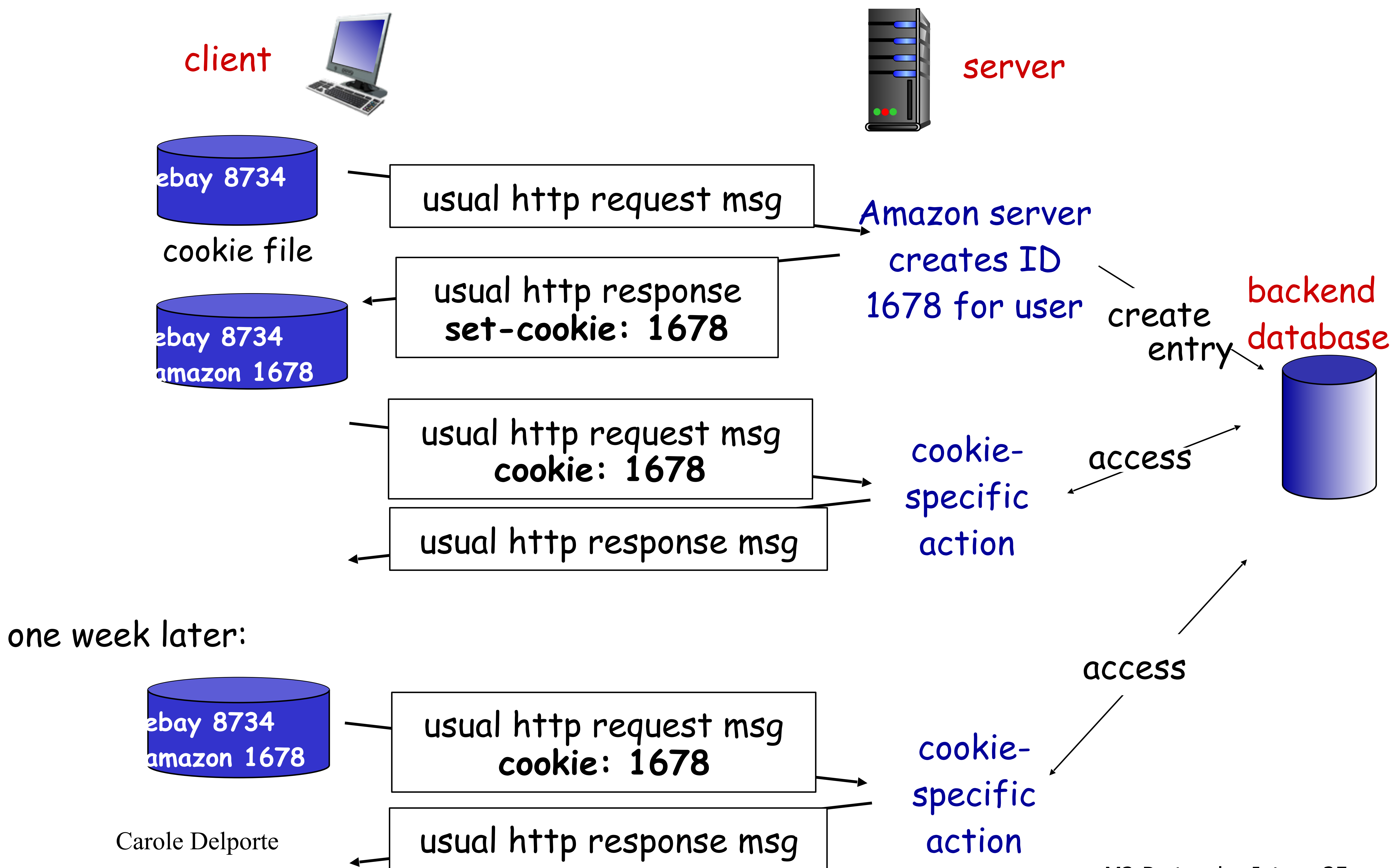
four components:

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

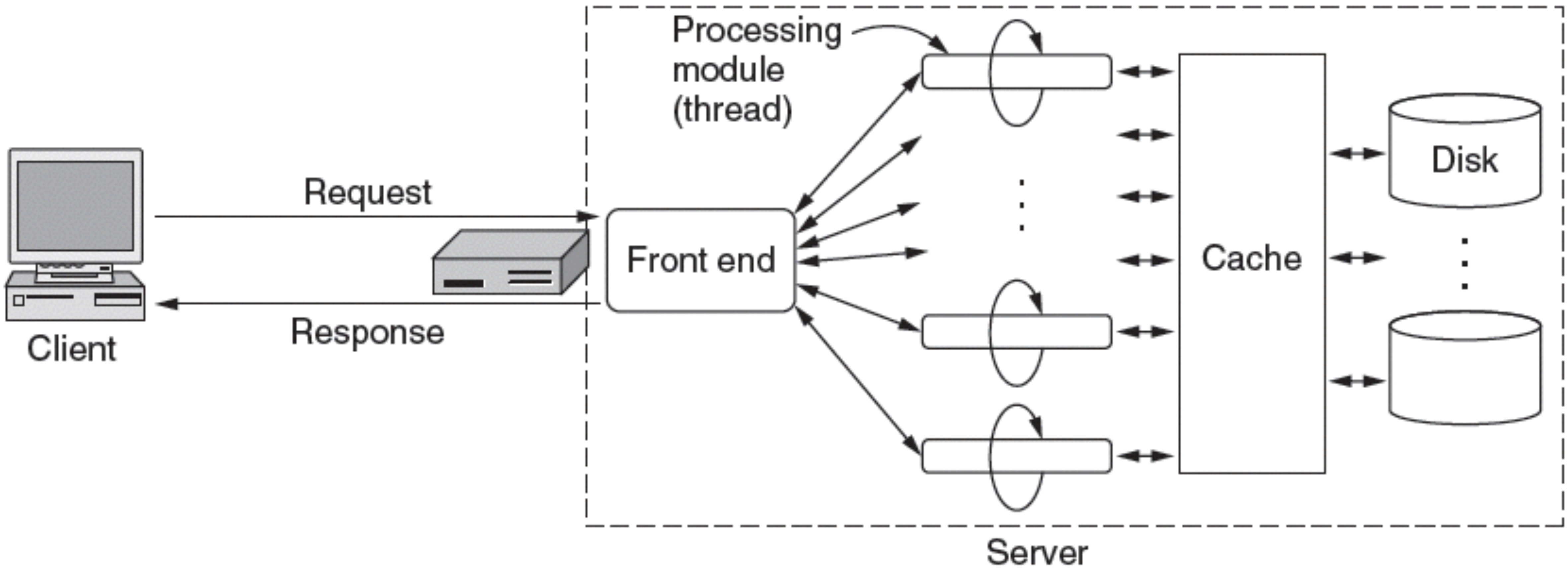
how to keep "state":

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

Caching

To scale performance, Web servers can use:

- Caching, multiple threads, and a front end



Caching…

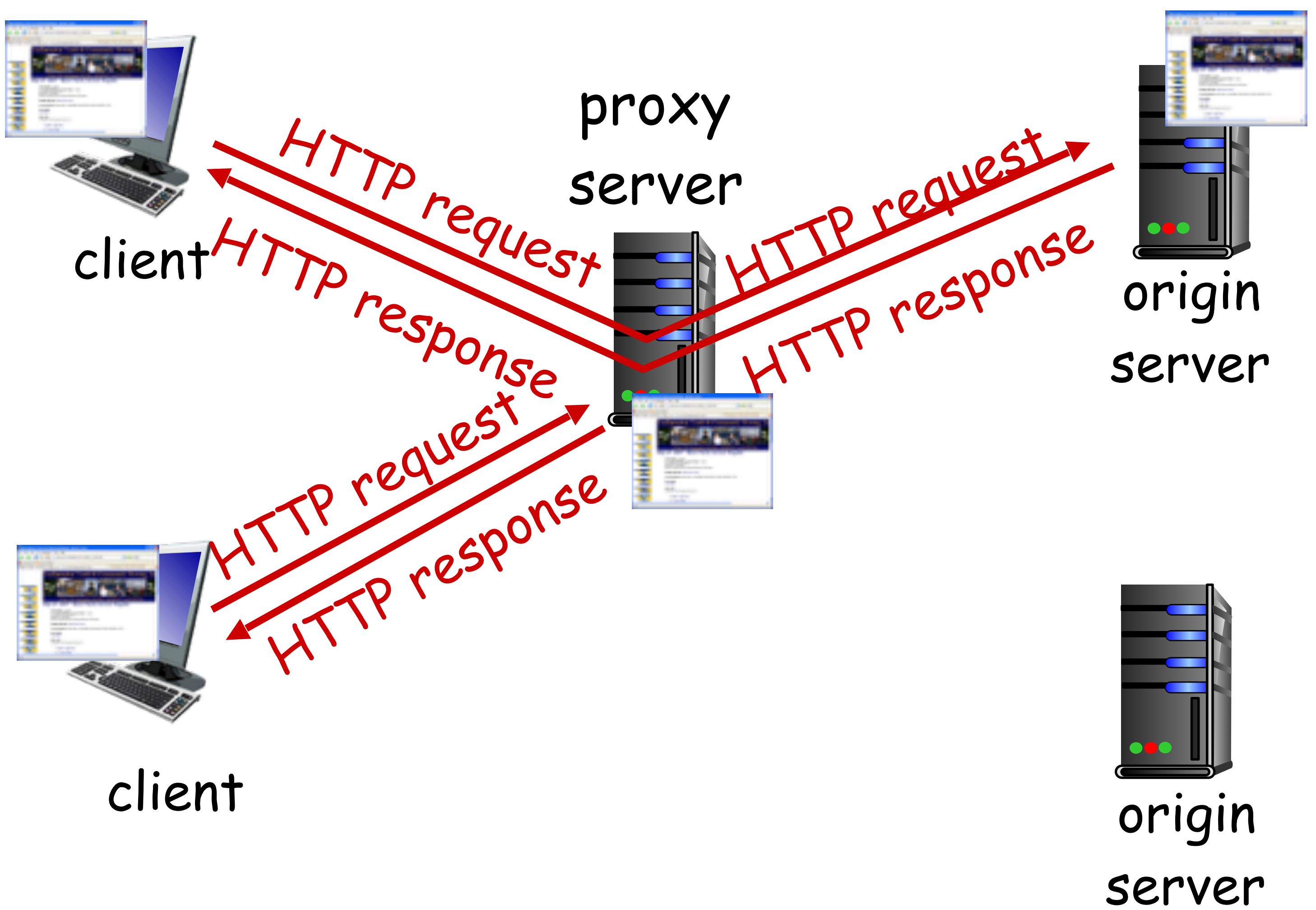
Server steps, revisited:

- Resolve name of Web page requested
- Perform access control on the Web page
- Check the cache
- Fetch requested page from disk or run program
- Determine the rest of the response
- Return the response to the client
- Make an entry in the server log

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



Carole Delporte

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:

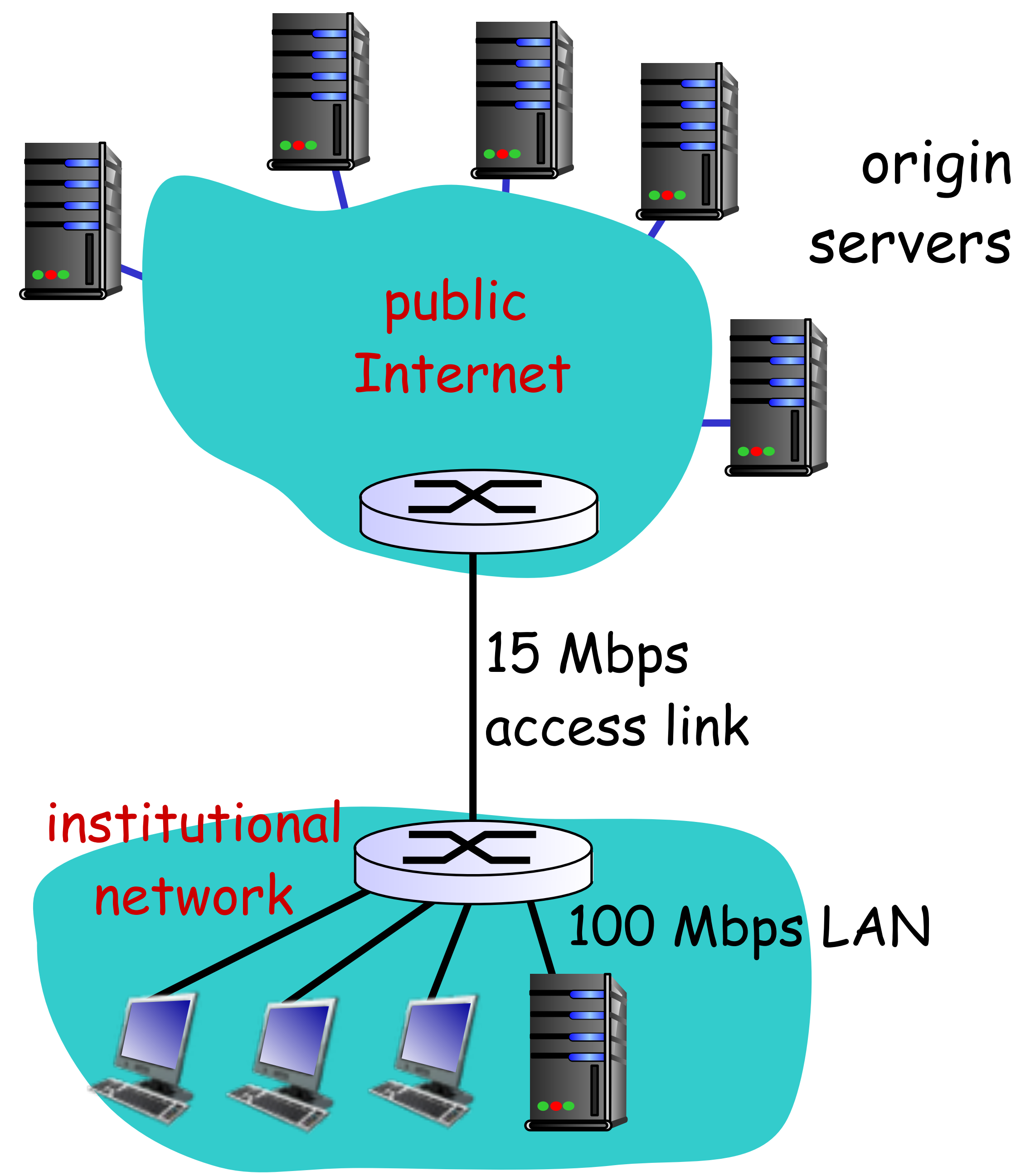
assumptions:

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

consequences:

- Traffic intensity on the LAN : 15%
- access link utilization = 100% problem!
- total delay = Internet delay + access delay + LAN delay

  = 2 sec + minutes + millisecs



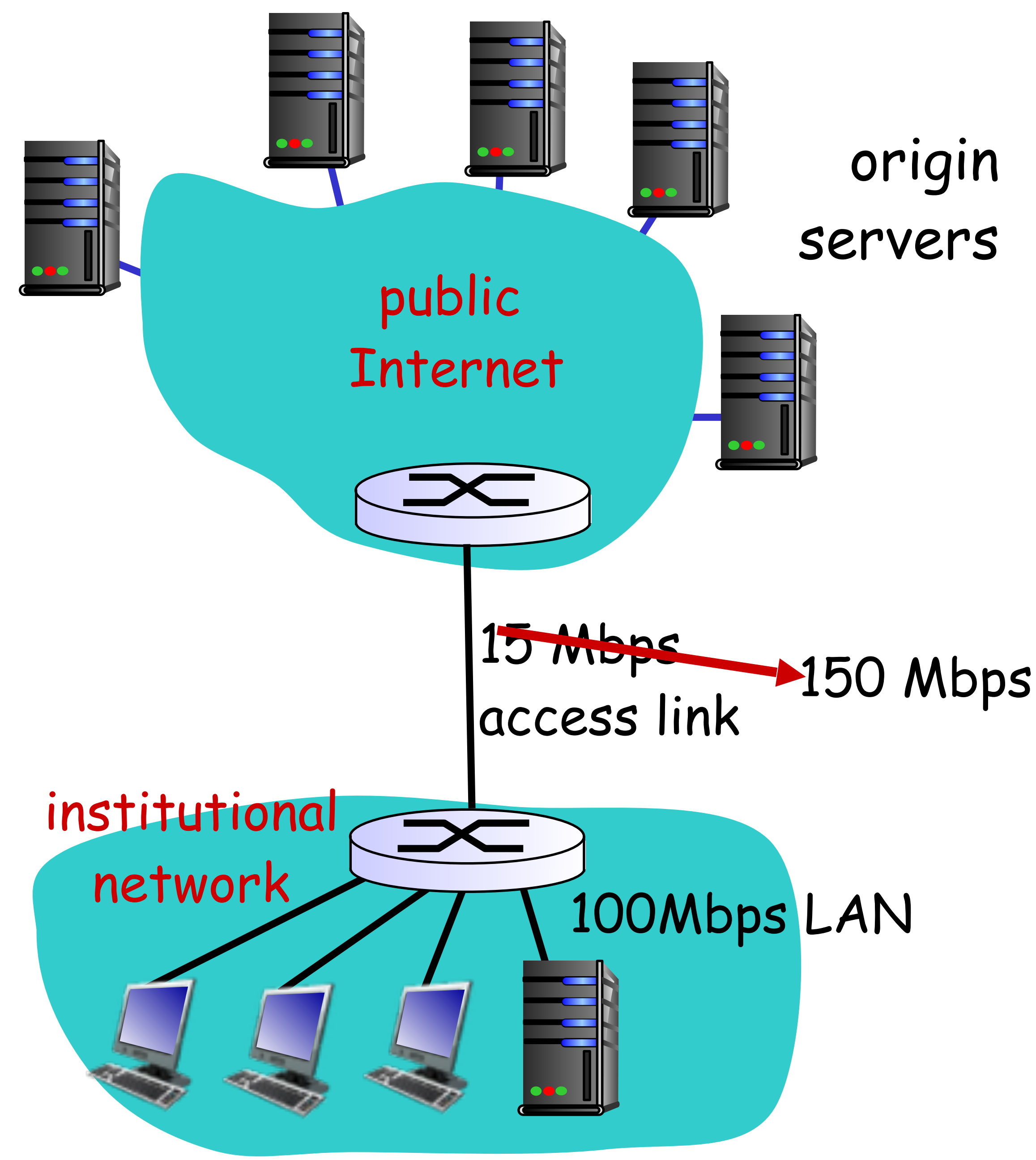
Caching example: fatter access link

assumptions:

- avg object size: 1Mbits
- avg request rate from browsers to origin servers:15/sec
- RTT from institutional router to any origin server: 2 sec
- access link rate: ~~15 Mbps~~ 150 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = ~~100%~~ 10%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + ~~minutes~~ msecs + msecs



Cost: increased access link speed (not cheap!)

Caching example: install local cache

assumptions:

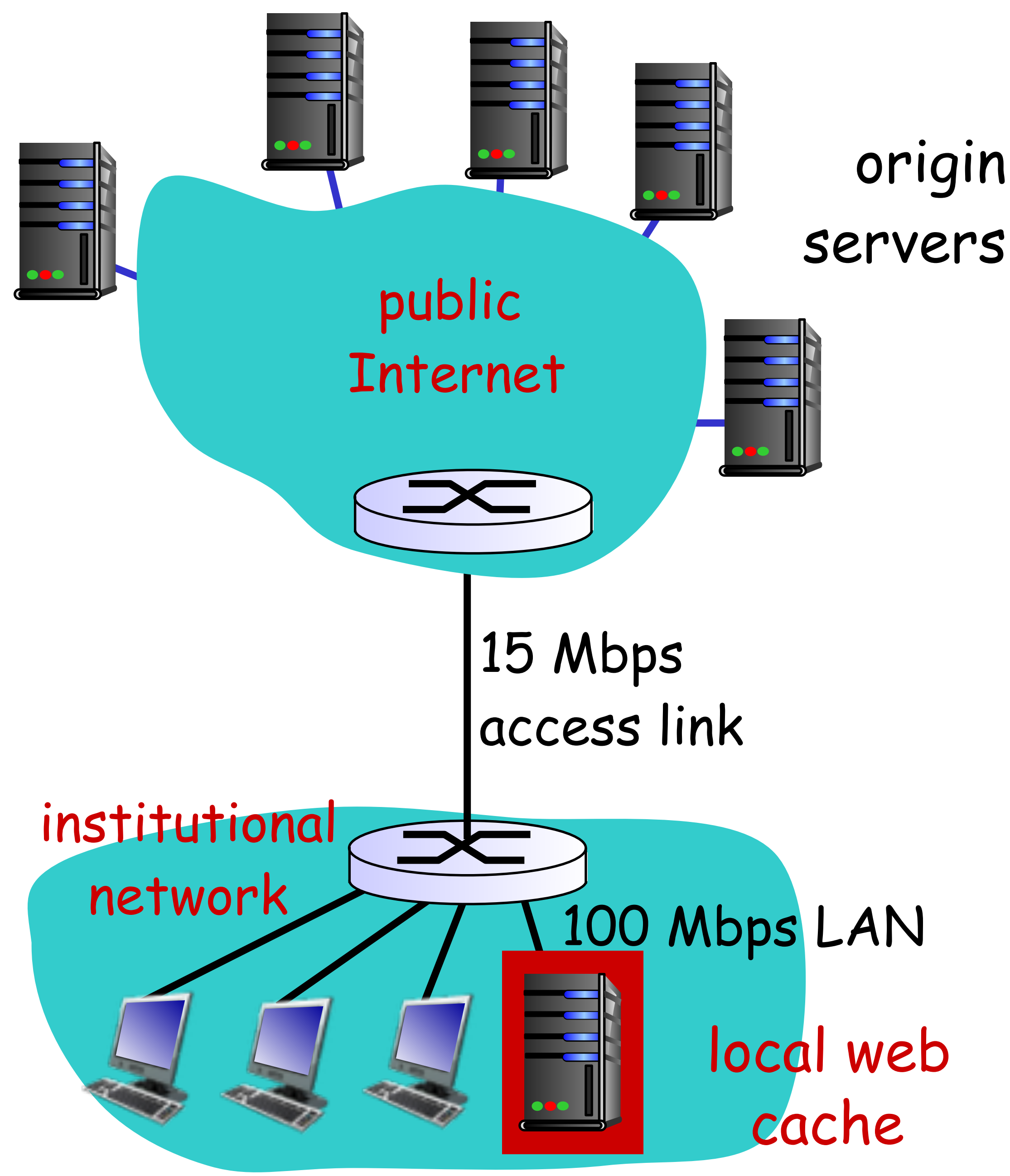
- avg object size: 1Mbits
- avg request rate from browsers to origin servers:15/sec
- RTT from institutional router to any origin server: 2 sec
- access link rate: 15 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = 100%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + usecs

?

?



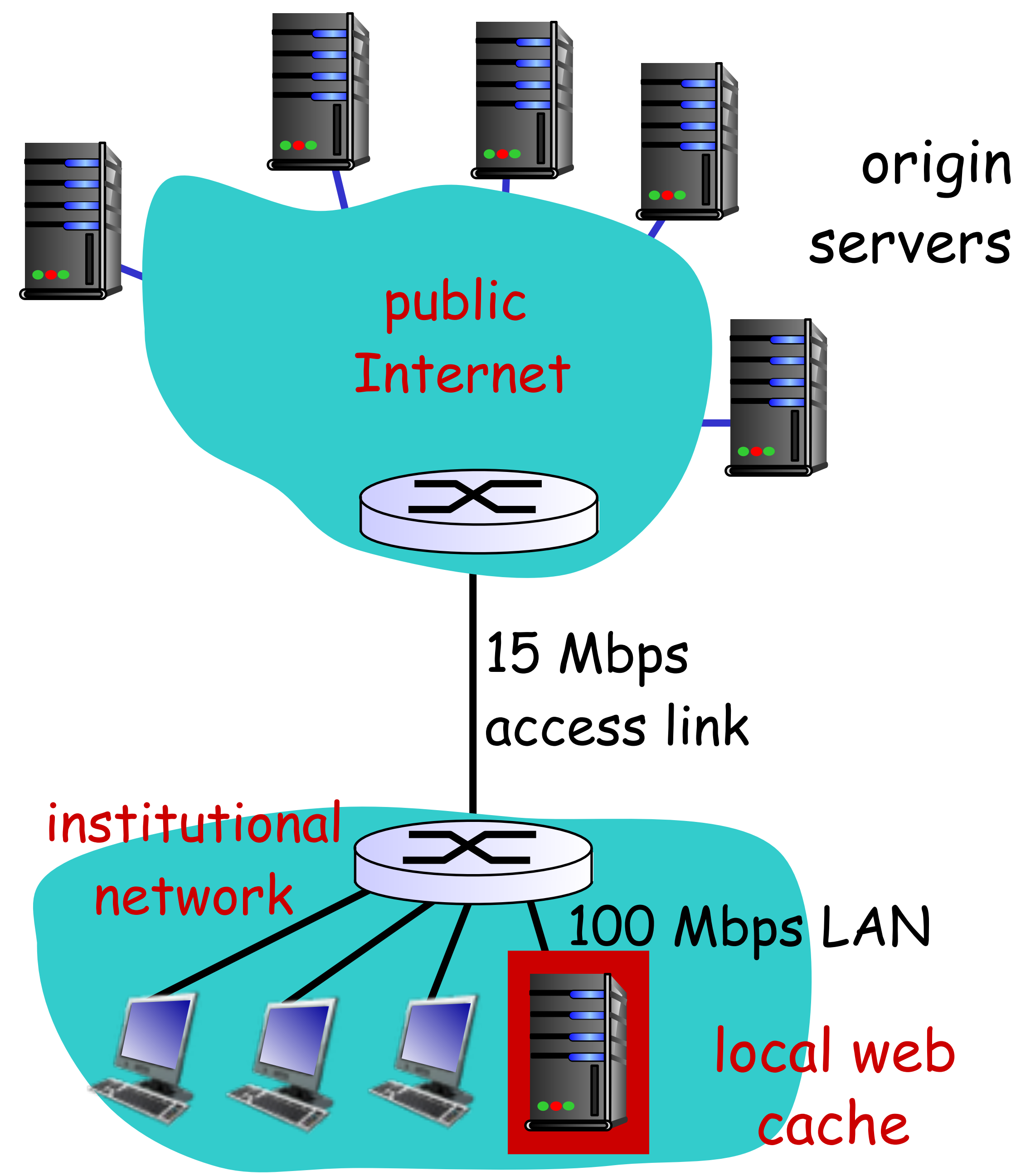
How to compute link utilization, delay?

Cost: web cache (cheap!)

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
  - access link utilization =60%
- 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
  - less than with 150 Mbps link (and cheaper too!)

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