CMPE 150: Introduction to Computer Networks

Dr. Chane L. Fullmer

chane@cse.ucsc.edu

Homework Assignments

Homework assignment #5

Chapter two

Due June 5th – This Thursday

(Optional) Class Project

- Network programming project
  - In lieu of taking final examination
  - Or, just a wildcard – for mid-term or final
- Goal:
  - Build an FTP client/server from scratch
  - Using 'C' language
- Details on web page.. now…

Due by tomorrow

(June 4th)

Class Final Exam

- Final exam
  - Three hours
  - 40 – 50 questions -- comprehensive
  - Multiple choice as the midterm
  - Scantron – bring your sheet and pencils

  - **Wednesday – June 11th** 8:00 – 11:00am
    - Eight short days.. Tick, tick, tick…

Class Review Lecture

- Thursday – June 5th
  - (last class meeting)
- Coverage of high points of the quarter
  - Much of what might appear on the final exam…..

CMPE 150: Introduction to Computer Networks Set 18:

Application-level Protocols – continued

HTTP

Background

- Hypertext Transport Protocol.
- Created by Tim Berners-Lee at CERN.
  - Physicists, not computer scientists!
  - Share data from physics experiments.
- Standardized and much expanded by the IETF.

Web and HTTP

- 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
- Example URL:

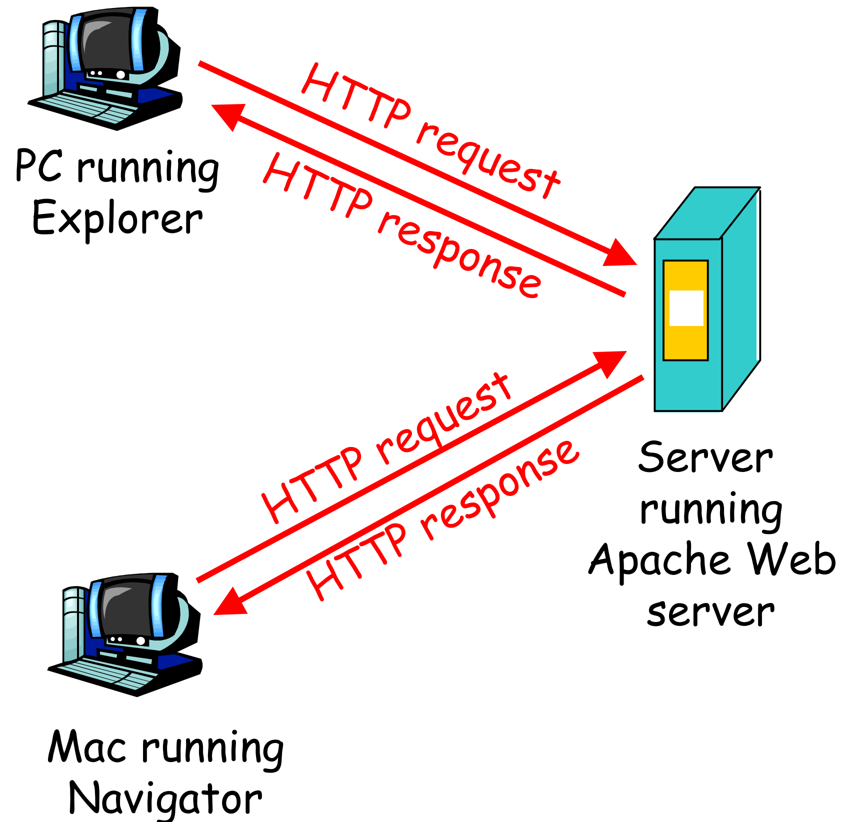
```
www.someschool.edu/someDept/pic.gif
```

host name — path name

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, "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:

- 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

<u>Nonpersistent HTTP</u>

- At most one object is sent over a TCP connection.
- HTTP/1.0 uses nonpersistent HTTP

<u>Persistent HTTP</u>

- Multiple objects can be sent over single TCP connection between client and server.
- HTTP/1.1 uses persistent connections in default mode

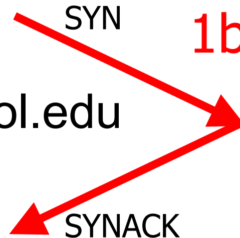
Nonpersistent HTTP

Suppose user enters URL

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

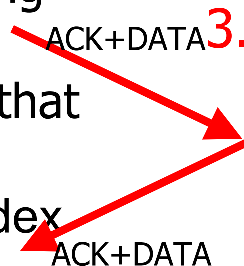
(contains text, references to 10 jpeg images)

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

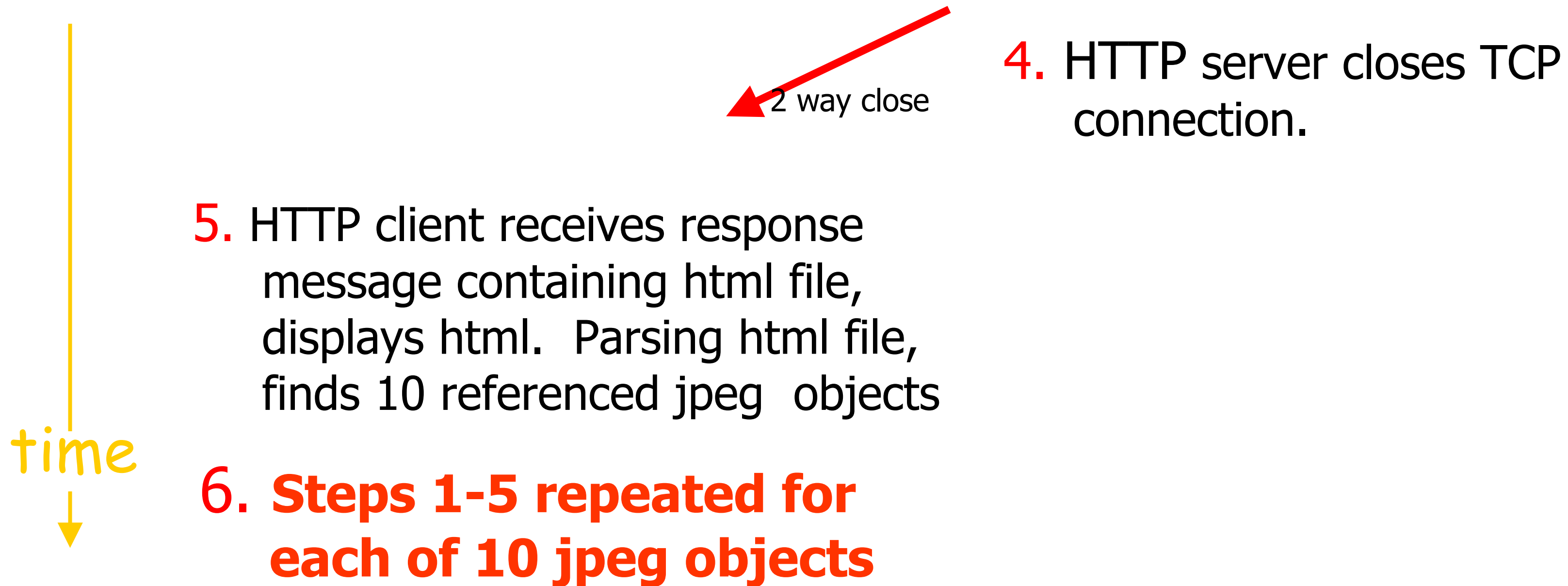
2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index



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

time

Nonpersistent HTTP (cont.)



Persistent HTTP

Nonpersistent HTTP issues:

- requires 2 RTTs per object
- OS must work and allocate host resources for each TCP connection
- but 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 are sent over connection

Persistent without pipelining:

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

Persistent with pipelining:

- 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

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

request line (GET, POST, HEAD commands)

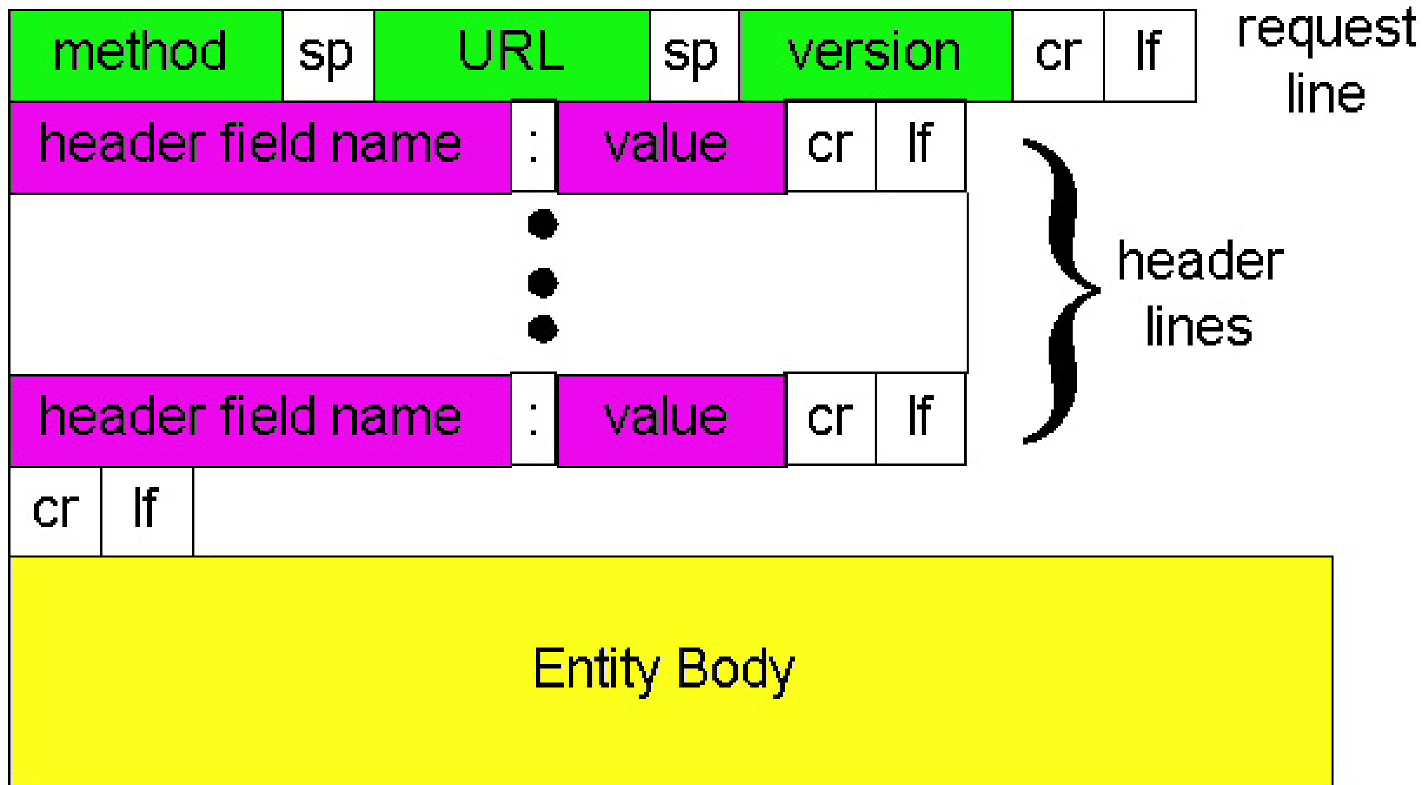
header lines

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:sp
```

(extra carriage return, line feed)

HTTP request message: general format



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

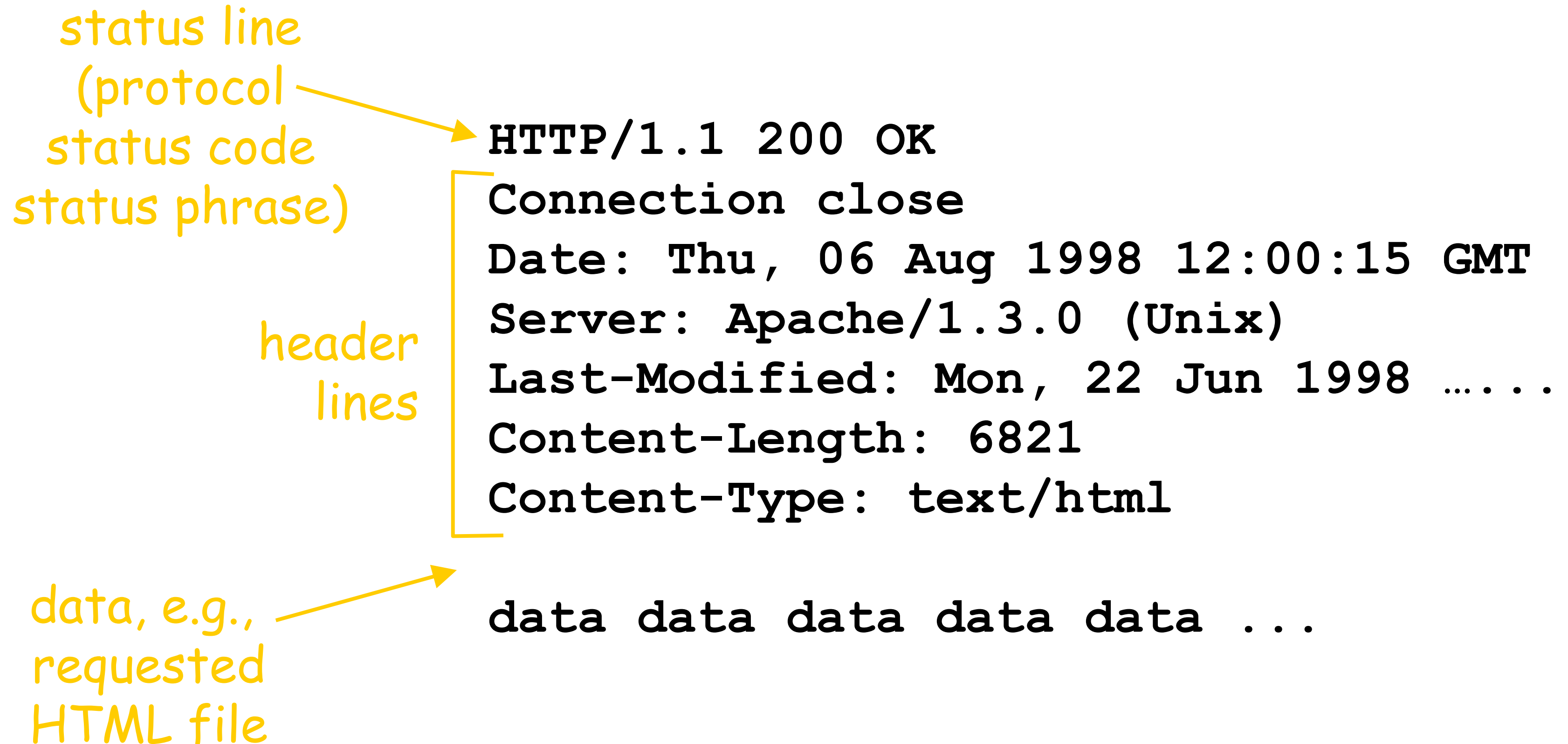
HTTP/1.0

- 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 response message



HTTP response status codes

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

1. Telnet to your favorite Web server:

```
telnet www.eurecom.fr 80
```

Opens TCP connection to port 80 (default HTTP server port) at www.eurecom.fr. Anything typed in sent to port 80 at www.eurecom.fr

2. Type in a GET HTTP request:

```
GET /~ross/index.html HTTP/1.0
```

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!

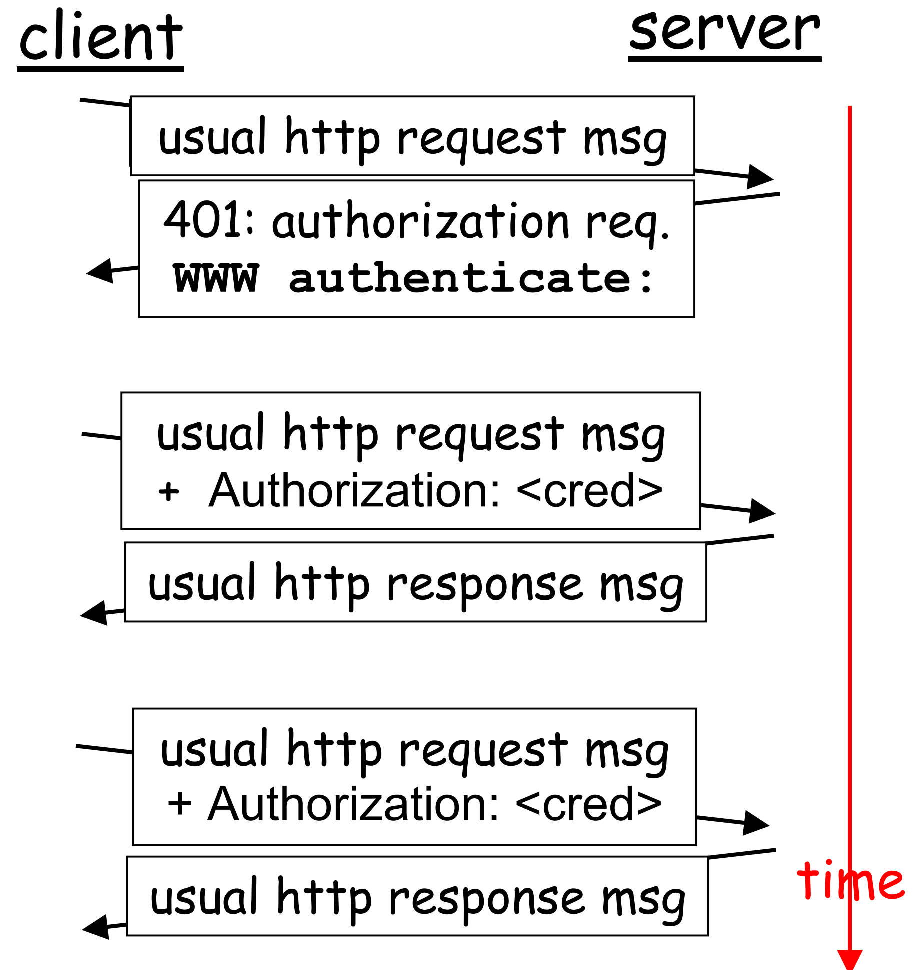
User-server interaction: authorization

Authorization : control access to server content

- authorization credentials: typically name, password
- stateless: client must present authorization in *each* request
  - authorization: header line in each request
  - if no authorization: header, server refuses access, sends

    `WWW authenticate:`

    header line in response



Cookies: keeping "state"

Many major Web sites use cookies

Four components:

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

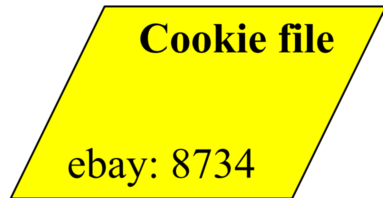
Example:

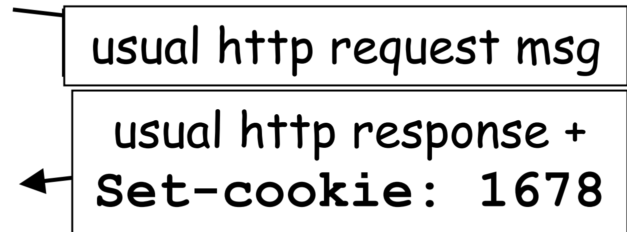
- Susan access Internet always from same PC
- She visits a specific e-commerce site for first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID

Cookies: keeping "state" (cont.)

client

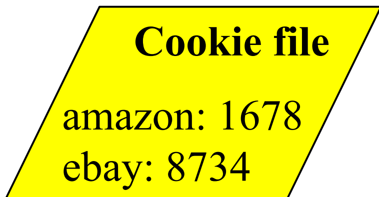
server





server creates ID 1678 for user

entry in backend database

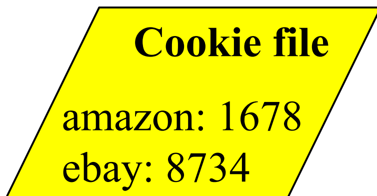


cookie-specific action

access

access

one week later:



cookie-spectific action

Cookies (continued)

What cookies can bring:

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

aside

Cookies and privacy:

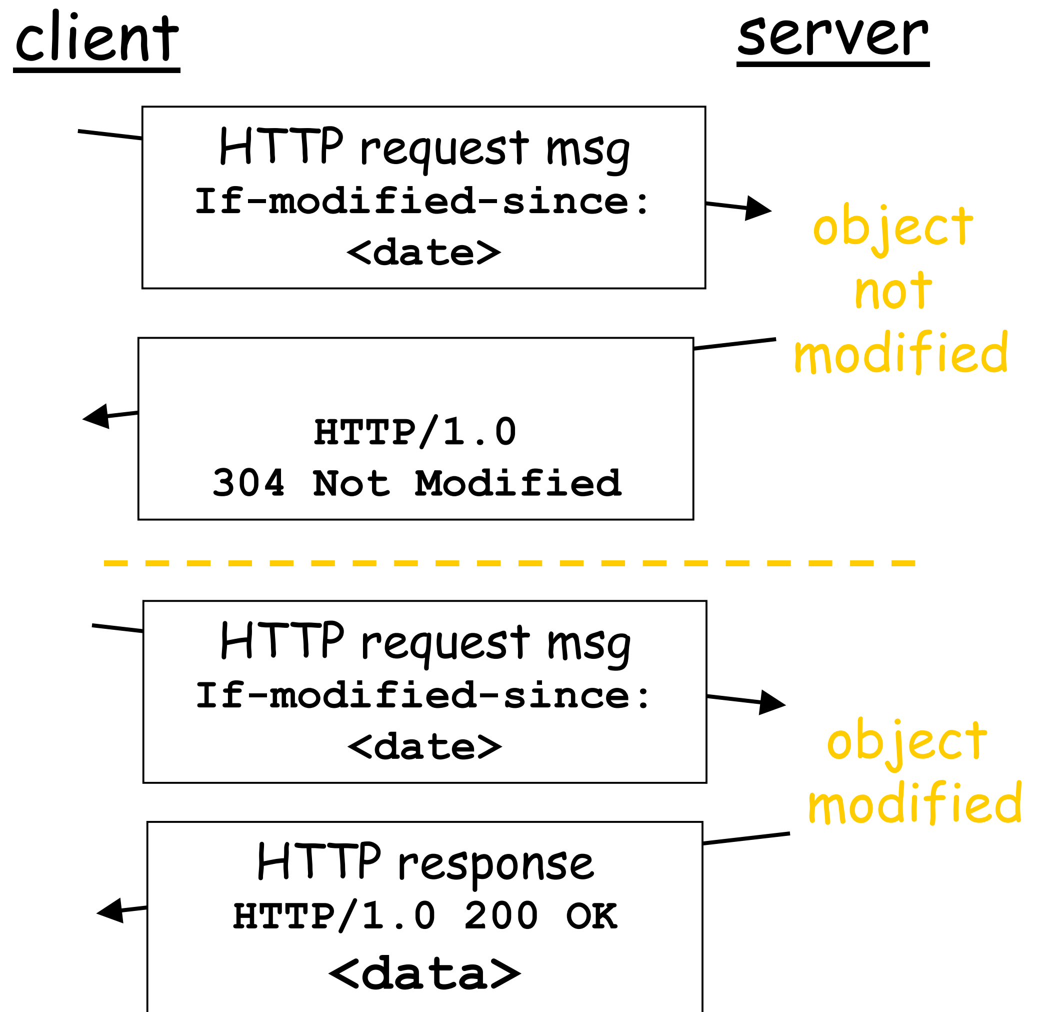
- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites
- search engines use redirection & cookies to learn yet more
- advertising companies obtain info across sites

Conditional GET: client-side caching

- Goal: don't send object if client has up-to-date cached version
- client: 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



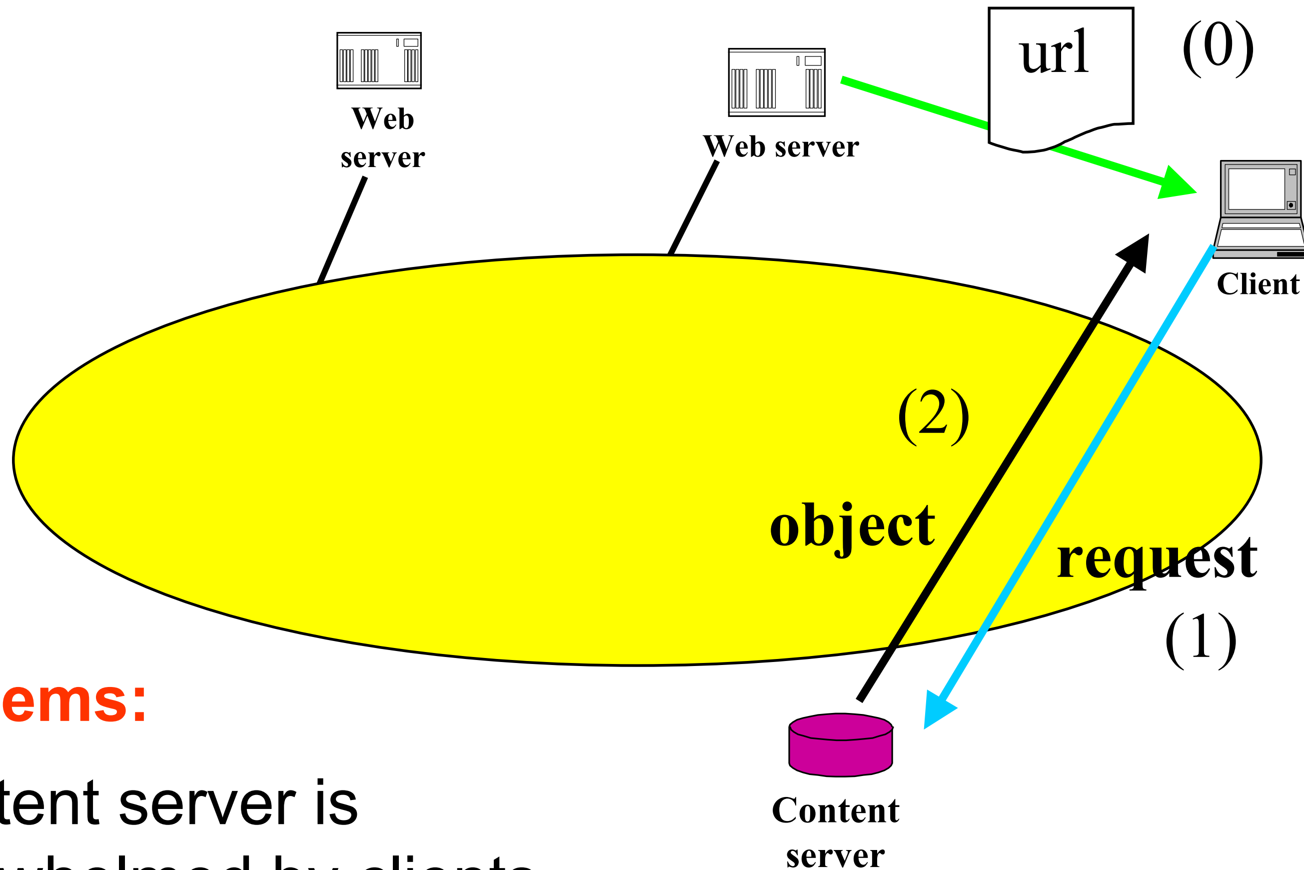
Review Thursday

Web caching

Caching architectures

- Proxy Caches
  - Serve a specific client population.
  - Can store recently accessed documents.
    - Lower latency for the end user.
    - Better use of wide area bandwidth by avoiding repeated transfers of recently-used information.
- Cooperative caching.
  - Multiple communicating caches.
  - … to increase hit rate and reduce access latency.
  - Approaches to cooperative caching
    - Hierarchical caches.
    - Non-hierarchical routable cache systems.

Why Web Caching



Problems:

- Content server is overwhelmed by clients.
- Content server may be very far from client.

Web Caching with Client-based Hashing

(0)

url

Web server

Web server

Client

hash

(2)

object

request

(1)

WebCache

WebCache

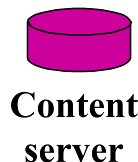
WebCache

WebCache

WebCache

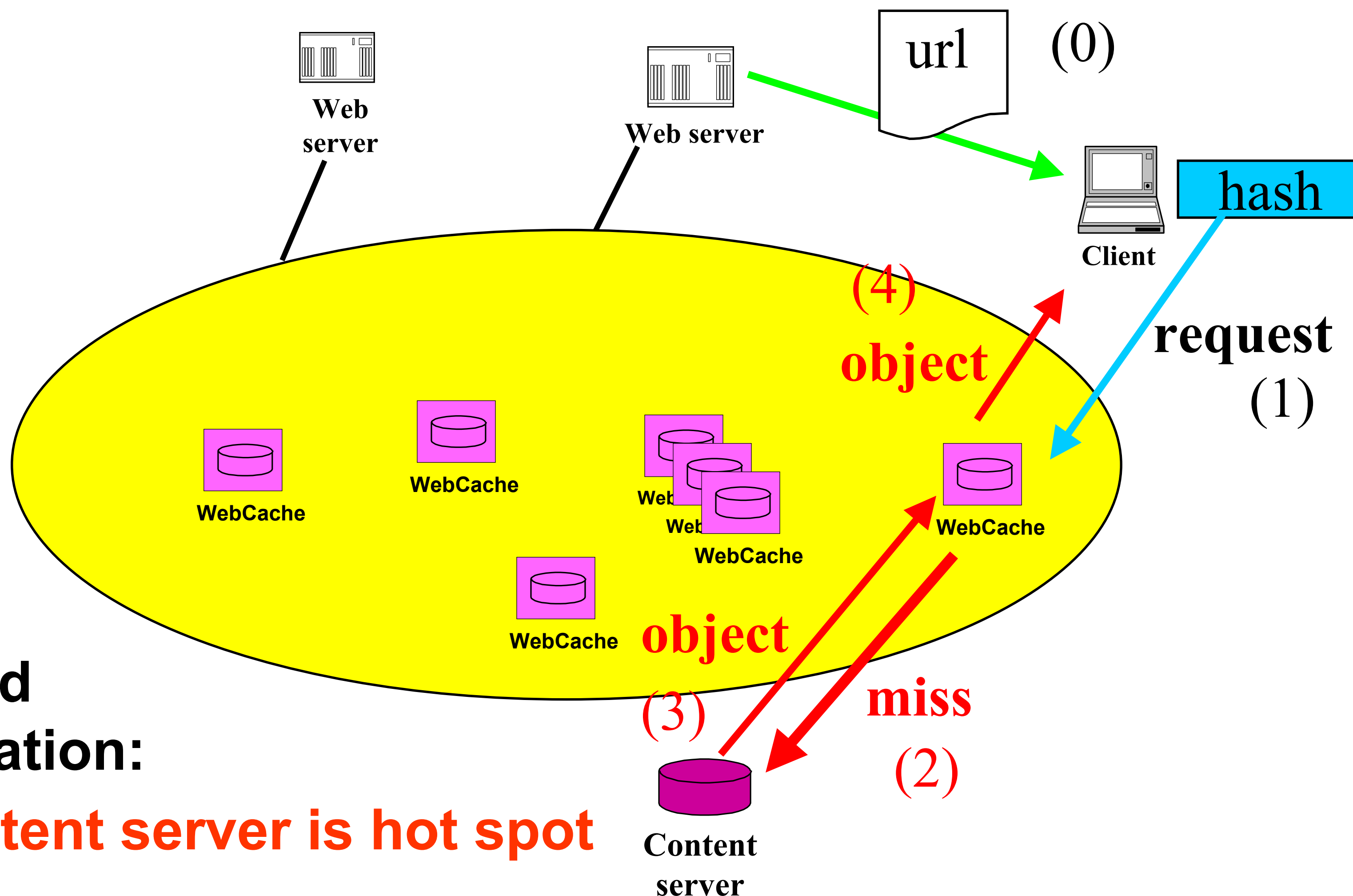
Limitations:

- Client is modified with hash
- One cache per object
- Search uses URLs



Content server

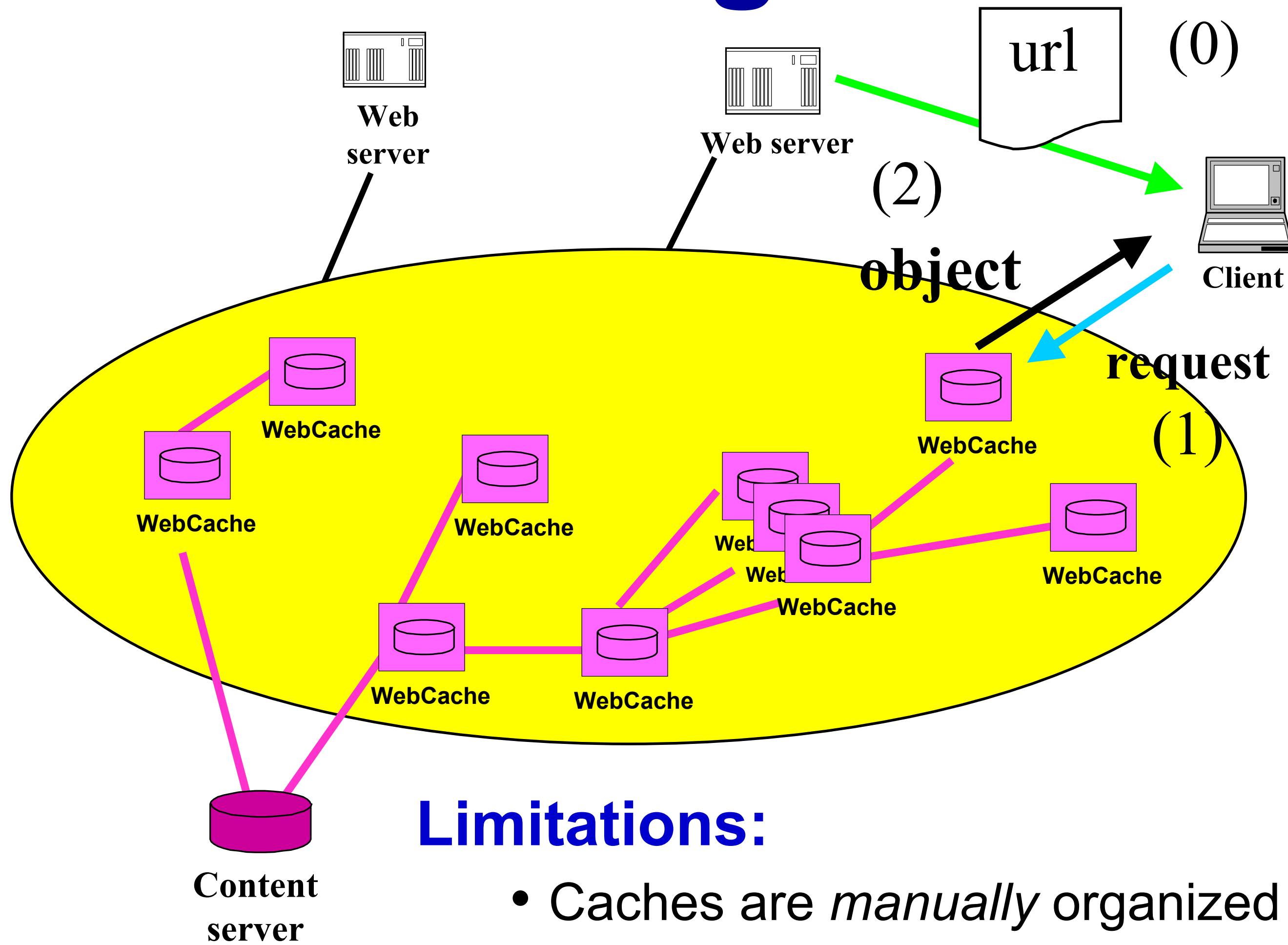
Web Caching with Client-based Hashing



Added Limitation:

- Content server is hot spot

Web Caching with Hierarchies

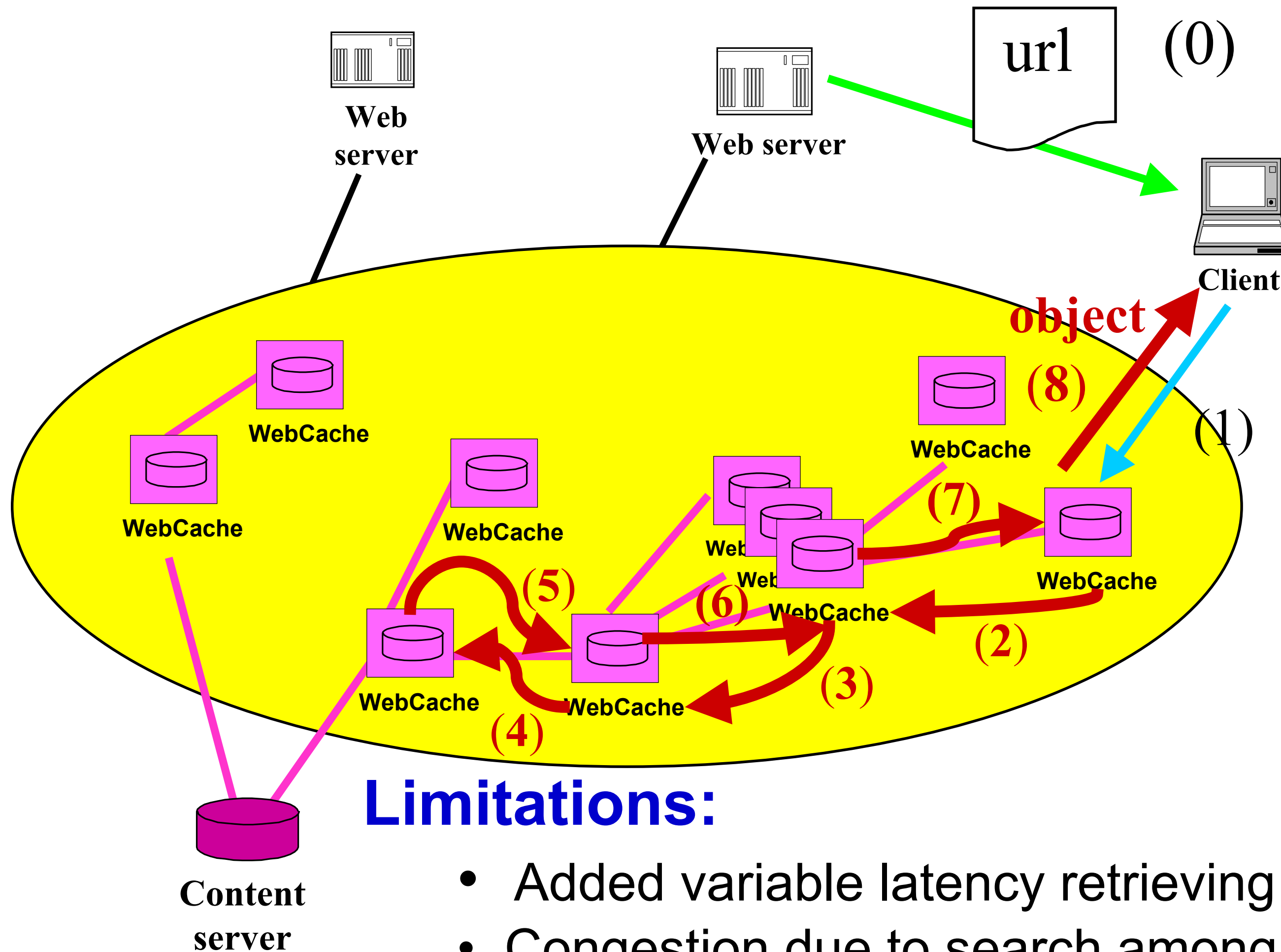


Advantage:
Content server is buffered from requests

Limitations:

- Caches are *manually* organized
- Hierarchy defined per system or per URL
- Hierarchy is independent of congestion and load

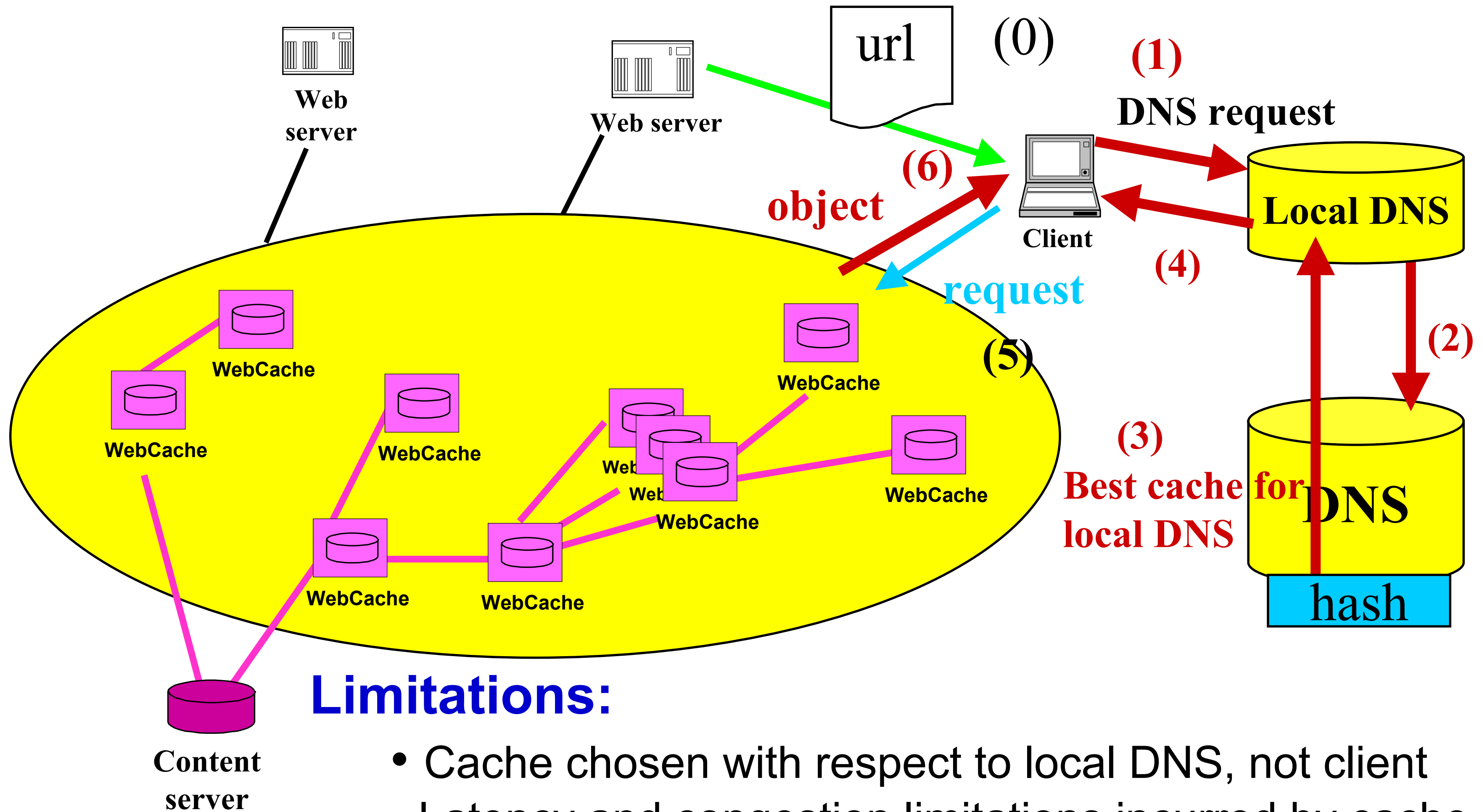
Web Caching with Hierarchies



Limitations:

- Added variable latency retrieving object
- Congestion due to search among siblings
- Client must choose the cache (static choice or require performance data at client)

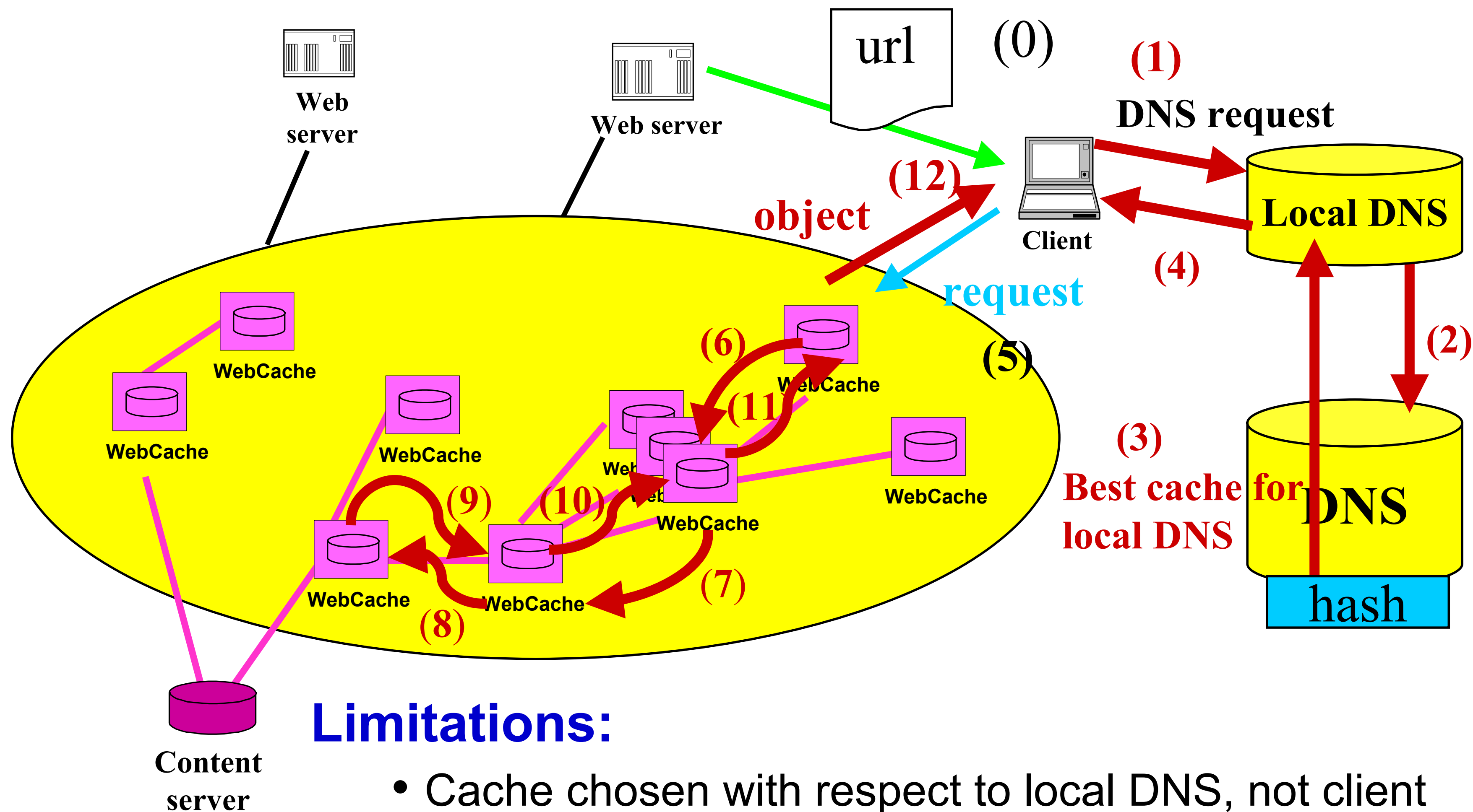
Web Caching with DNS-based Hashing



Limitations:

- Cache chosen with respect to local DNS, not client
- Latency and congestion limitations incurred by cache hierarchy

Web Caching with DNS-based Hashing



Limitations:

- Cache chosen with respect to local DNS, not client
- Latency and congestion limitations incurred by cache hierarchy

Summary of Limitations

- Special DNS servers or modified browsers needed
- Special DNS servers receive all requests
- Search for cache-client match must be done at the time the request is received
- Either hashing is independent of network congestion or performance information must be sent to special DNS servers or clients
- Caches must be (manually) organized into a hierarchy
- DNS-based hashing optimizes with respect to local DNS, not the client
- Latencies incurred getting responses can be long
- Hierarchy of caches incurs additional congestion

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