

Lecture 10

HTTP Protocol



Outline

- ❄ HTTP Overview
- ❄ HTTP Request & Response
- ❄ Web Application Design

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 (Uniform Resource Locator)

www.someschool.edu/someDept/pic.gif

host name path name

- ❖ Browsers

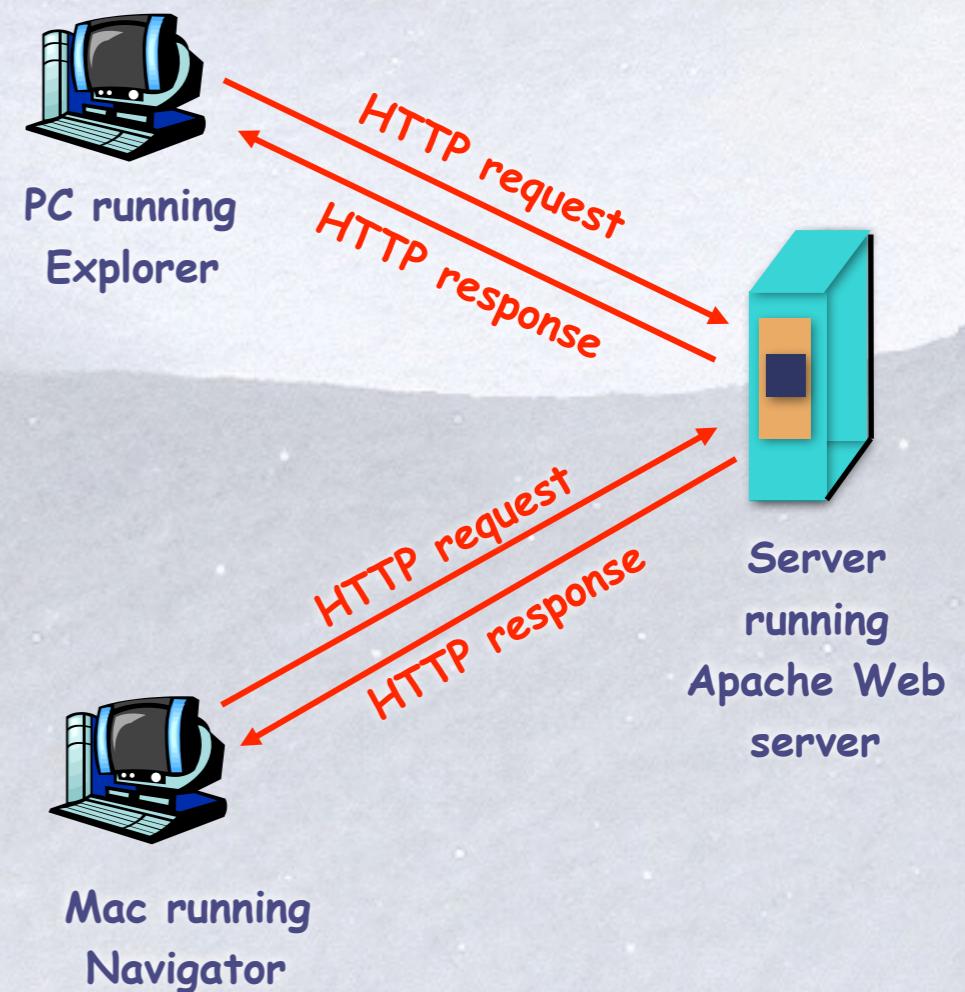
HTTP overview

❄ 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 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

HTTP Usage

- ❖ HTTP is the protocol that supports communication between web browsers and web servers.
- ❖ A “Web Server” is a HTTP server

From the RFC

- ❖ “HTTP is an application-level protocol with the lightness and speed necessary for distributed, hypermedia information systems.”

Transport Independence

- ❖ The RFC states that the HTTP protocol generally takes place over a TCP connection, but the protocol itself is not dependent on a specific transport layer.

HTTP Versions

- ❖ The original version now goes by the name “HTTP Version 0.9”
 - * HTTP 0.9 was used for many years.
- ❖ Starting with HTTP 1.0 the version number is part of every request.
 - * tells the server what version the client can talk (what options are supported, etc).
- ❖ HTTP 1.0: RFC 1945
- ❖ HTTP 1.1: RFC 2616
- ❖ HTTPS
- ❖ HTTP 2.0

Outline

- ❄ HTTP Overview
- ❄ **HTTP Request & Response**
- ❄ Web Application Design

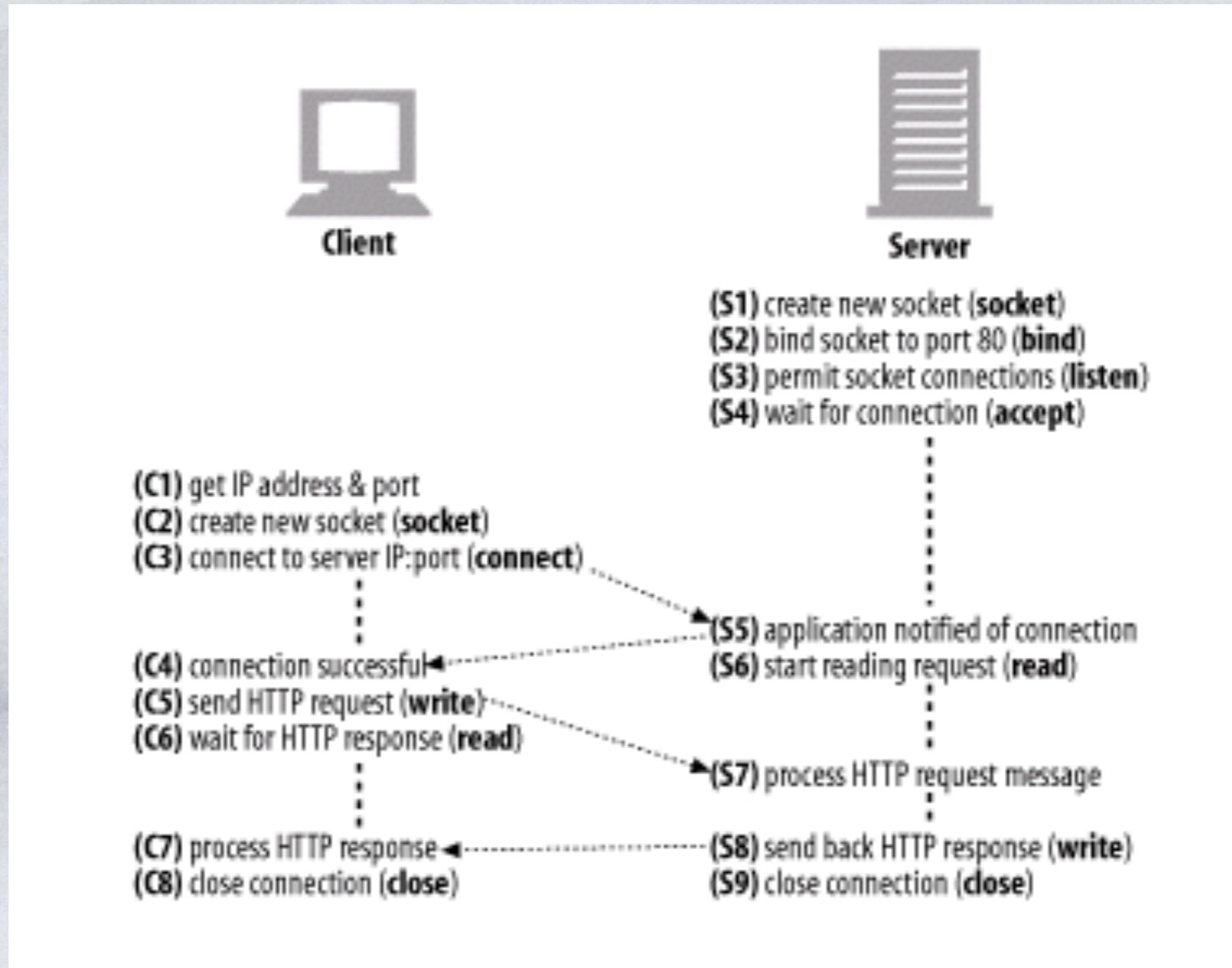
Request - Response

❄️ HTTP has a simple structure: 

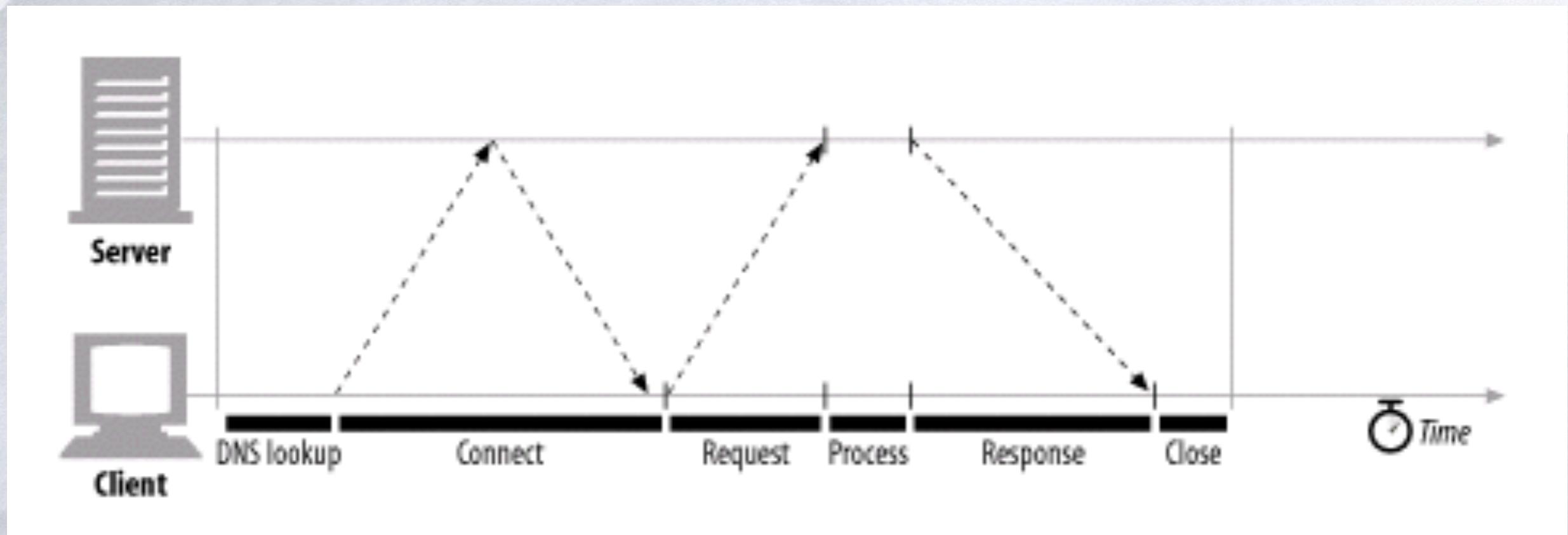
- * client sends a request
- * server returns a reply.

❄️ HTTP can support multiple request-reply exchanges over a single TCP connection.

How HTTP clients and servers communicate using the TCP sockets interface



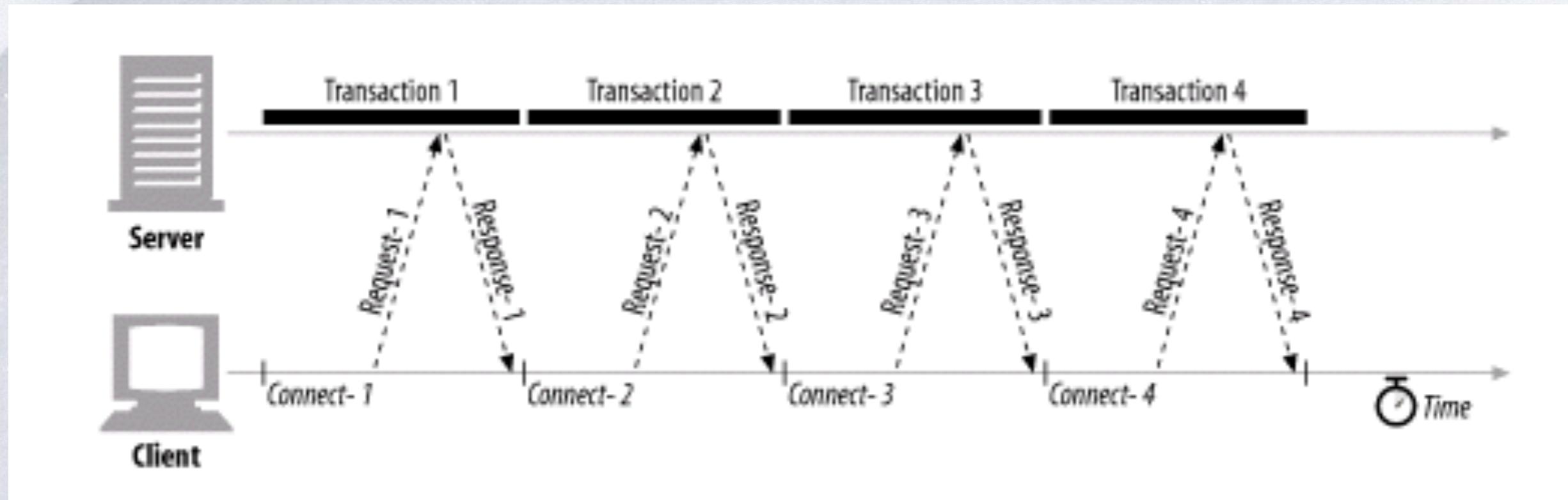
HTTP Transaction Delay



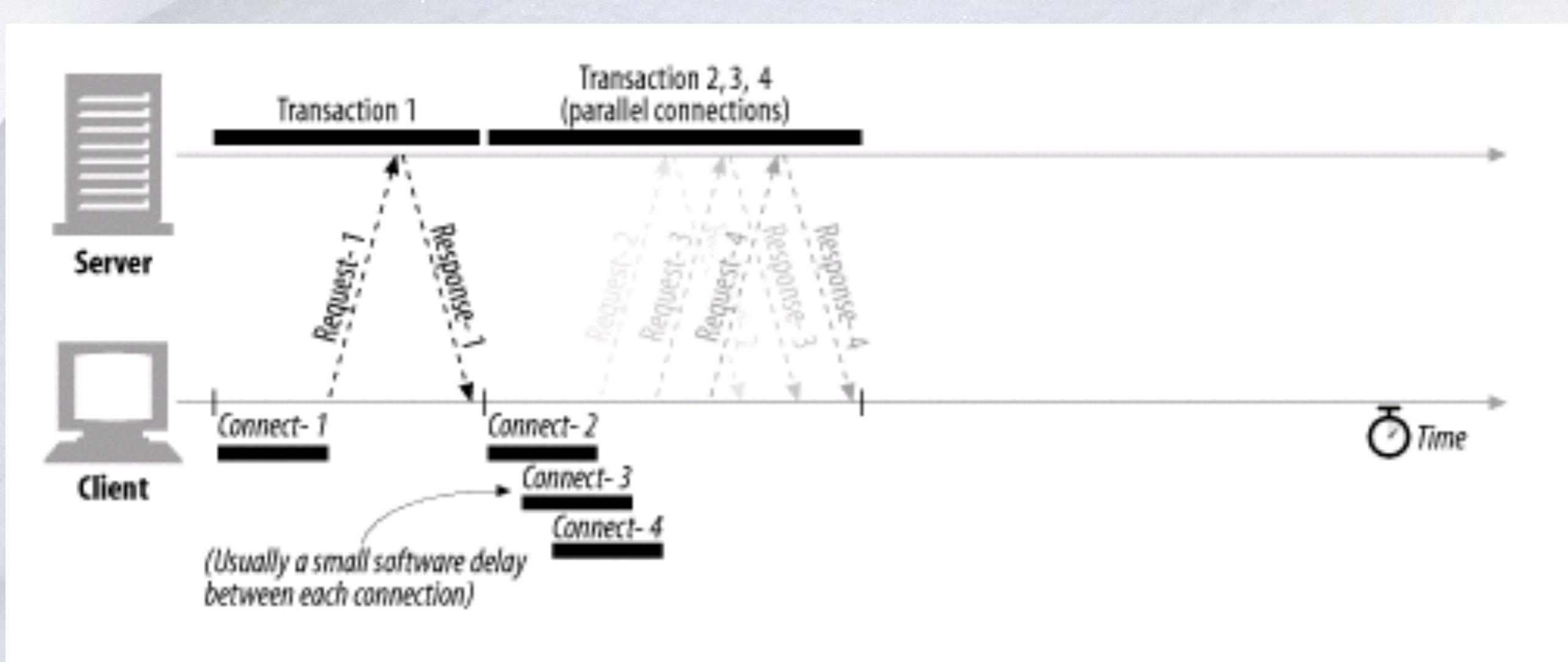
Common TCP-related delays affecting HTTP

- ❄ The TCP connection setup handshake
- ❄ TCP slow-start congestion control
- ❄ Nagle's algorithm for data aggregation
- ❄ TCP's delayed acknowledgment algorithm for piggybacked acknowledgments
- ❄ TIME_WAIT delays and port exhaustion

Serial Transaction Delays



Parallel connections

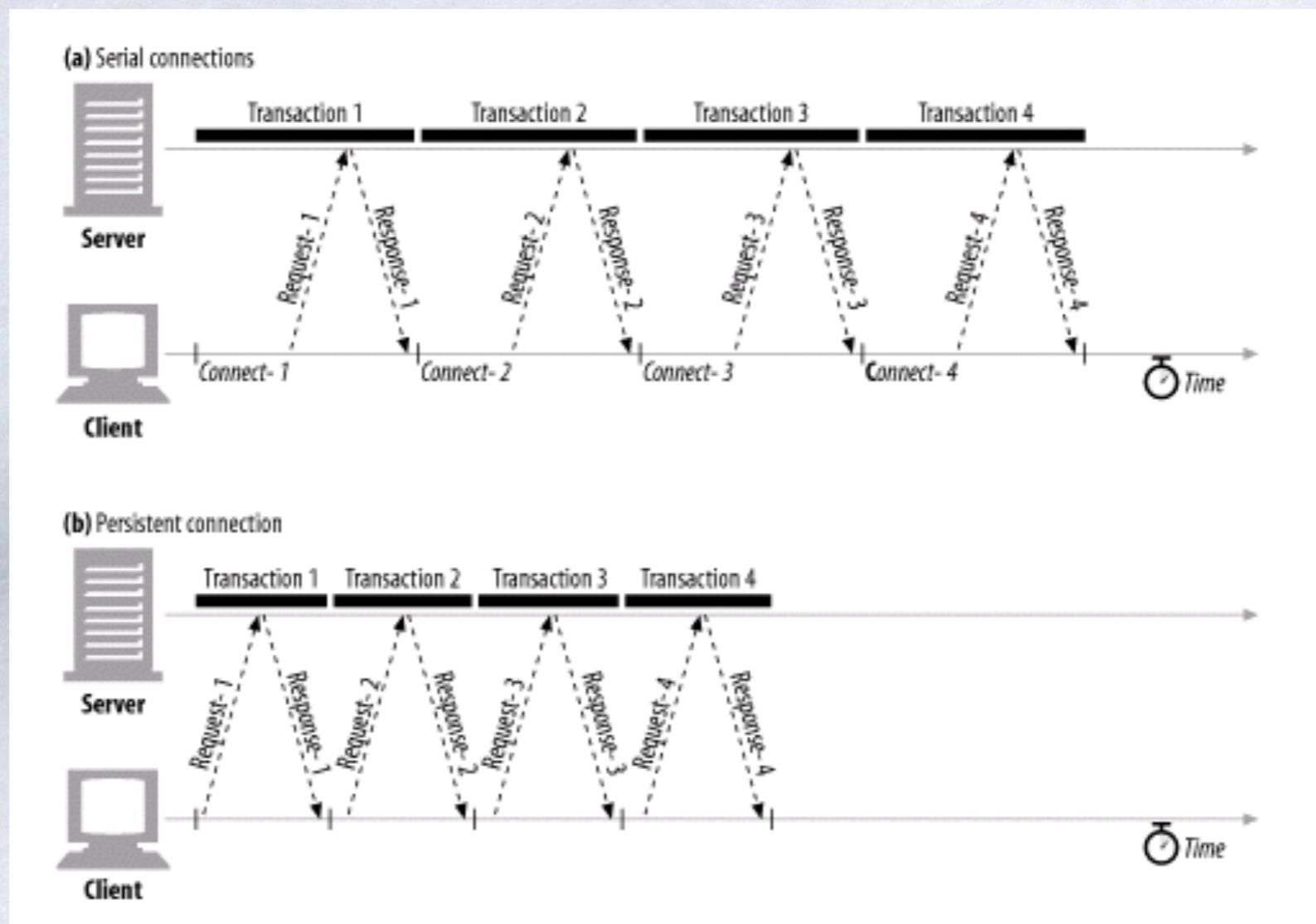


HTTP connections

Nonpersistent HTTP	Persistent HTTP
At most one object is sent over a TCP connection.	Multiple objects can be sent over single TCP connection between client and server.
HTTP/1.0 uses nonpersistent HTTP	HTTP/1.1 uses persistent connections in default mode

Persistent Connections

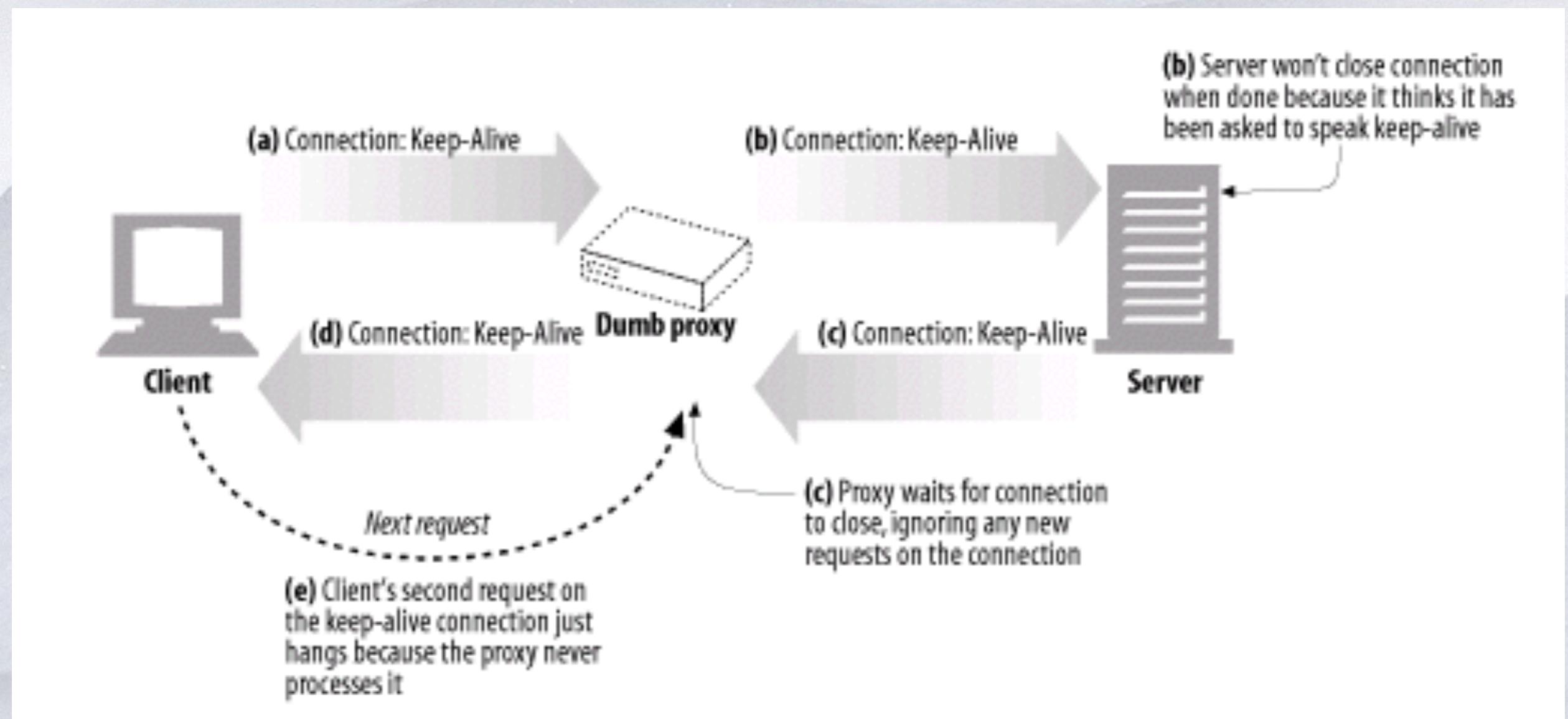
- ✿ TCP connections that are kept open after transactions complete are called persistent connections.



HTTP/1.0+ Keep-Alive Connections

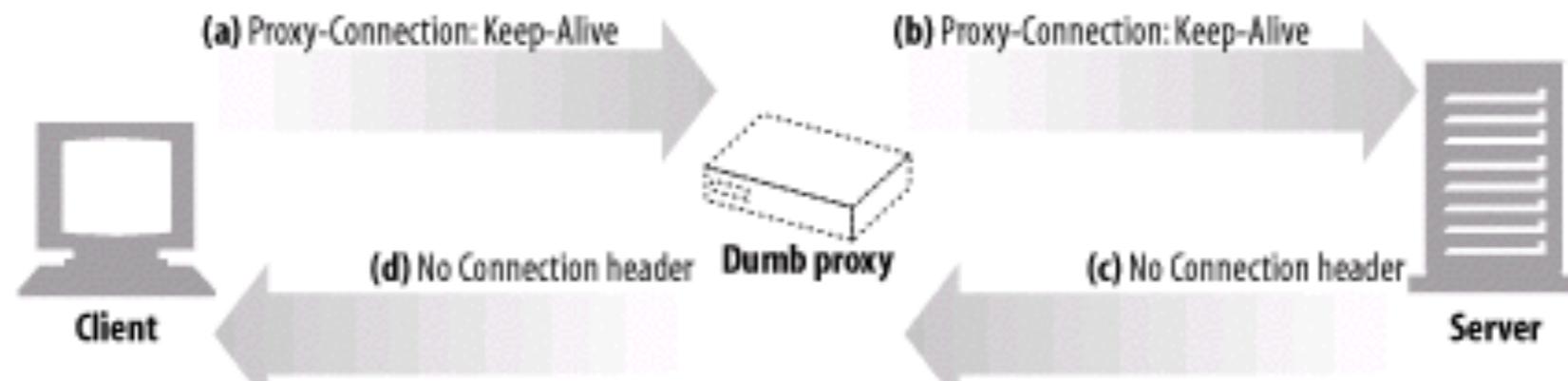


Keep-Alive and Dumb Proxy



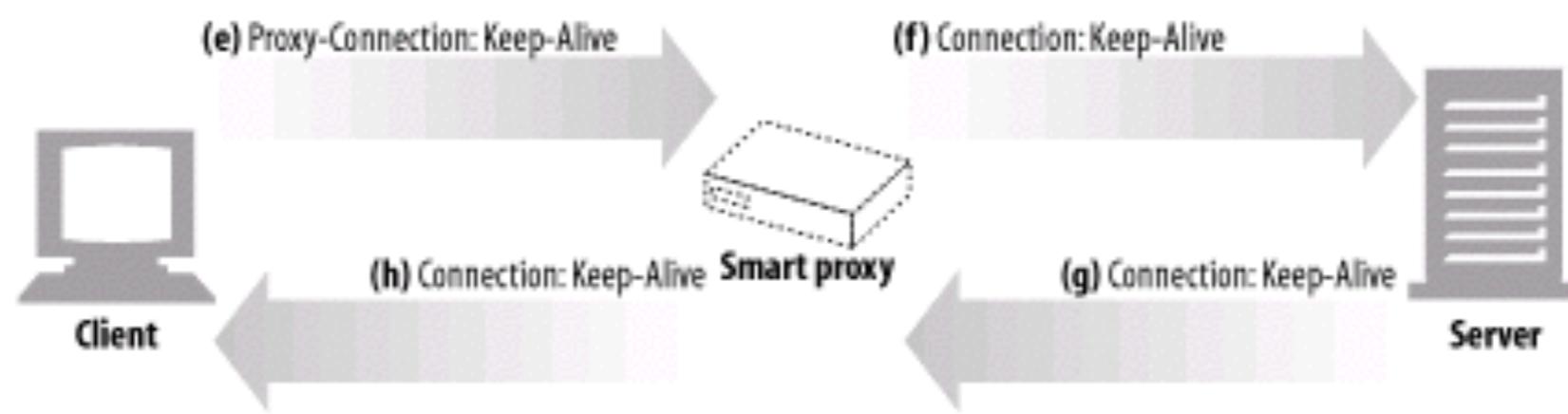
Proxy-Connection Hack

The server does not recognize the Proxy-Connection header, and ignores it.
No keep-alive Connection is established.



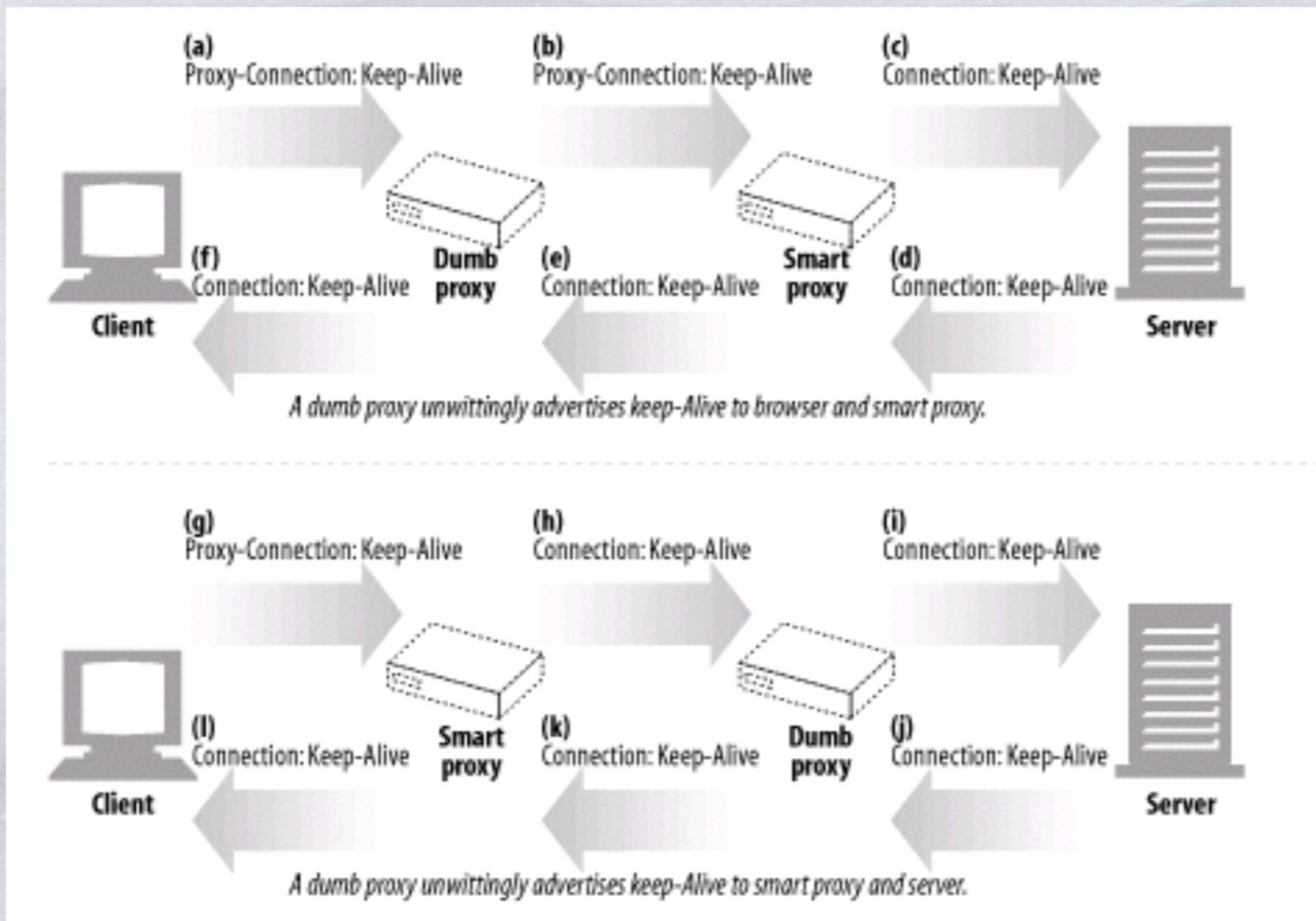
A dumb proxy forwards the Proxy-Connection header, which the server ignores.

The proxy recognizes the Proxy-Connection header, agrees to talk keep-alive with the client, and may also (optionally) decide to set up a keep-alive Connection with the server.



A smart proxy understands the Proxy-Connection header and actively sends a Connection: Keep-Alive header to the server.

Proxy-Connection's drawback



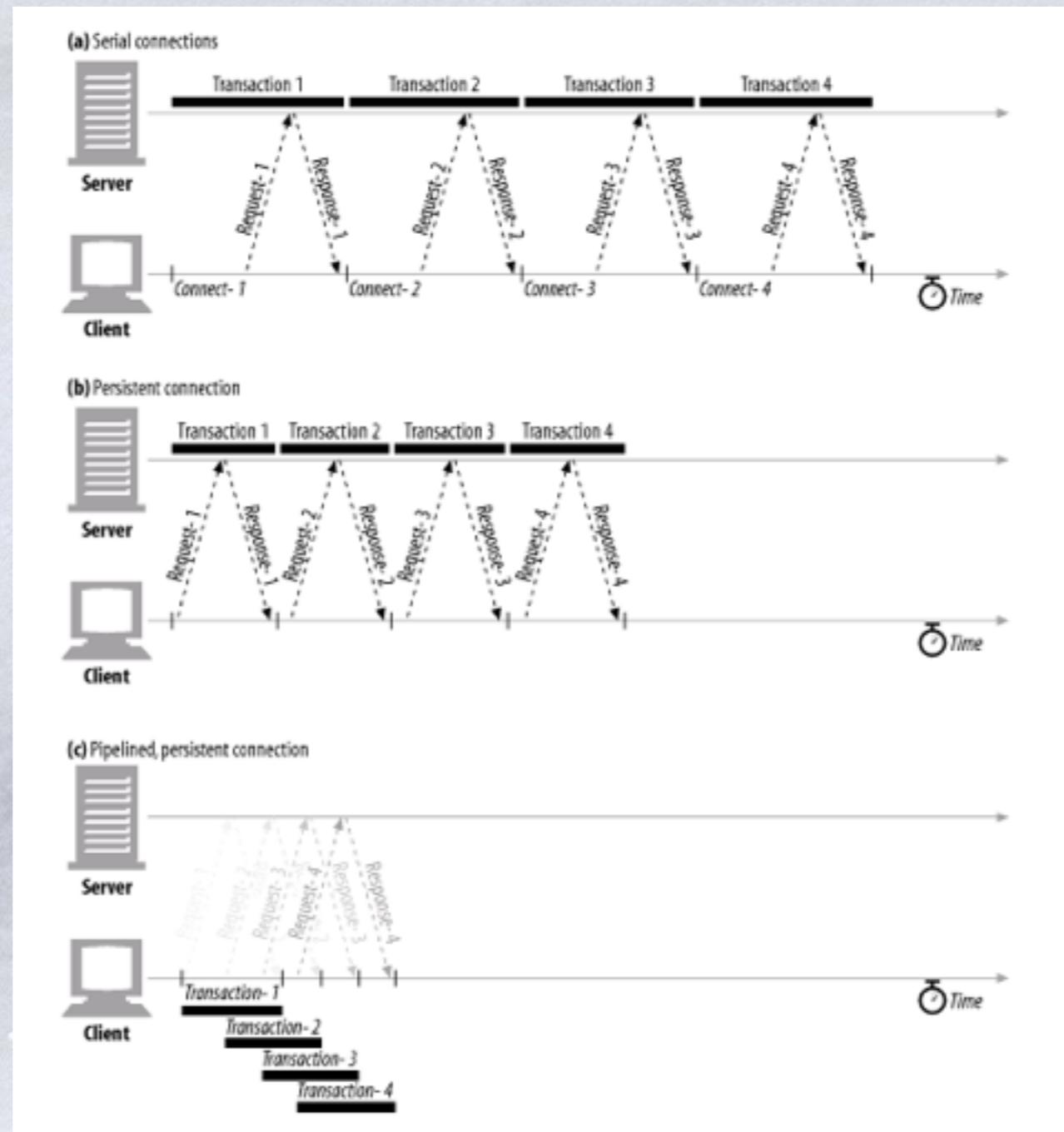
HTTP/1.1 Persistent Connections

- ❖ Unlike HTTP/1.0+ keep-alive connections, HTTP/1.1 persistent connections are active by default. HTTP/1.1 assumes all connections are persistent unless otherwise indicated by `Connection: close` header.

Pipelined Connections

- ❖ HTTP/1.1 permits optional request pipelining over persistent connections.
- ❖ Multiple requests can be enqueued before the responses arrive.

Serial vs persistent vs pipeline



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

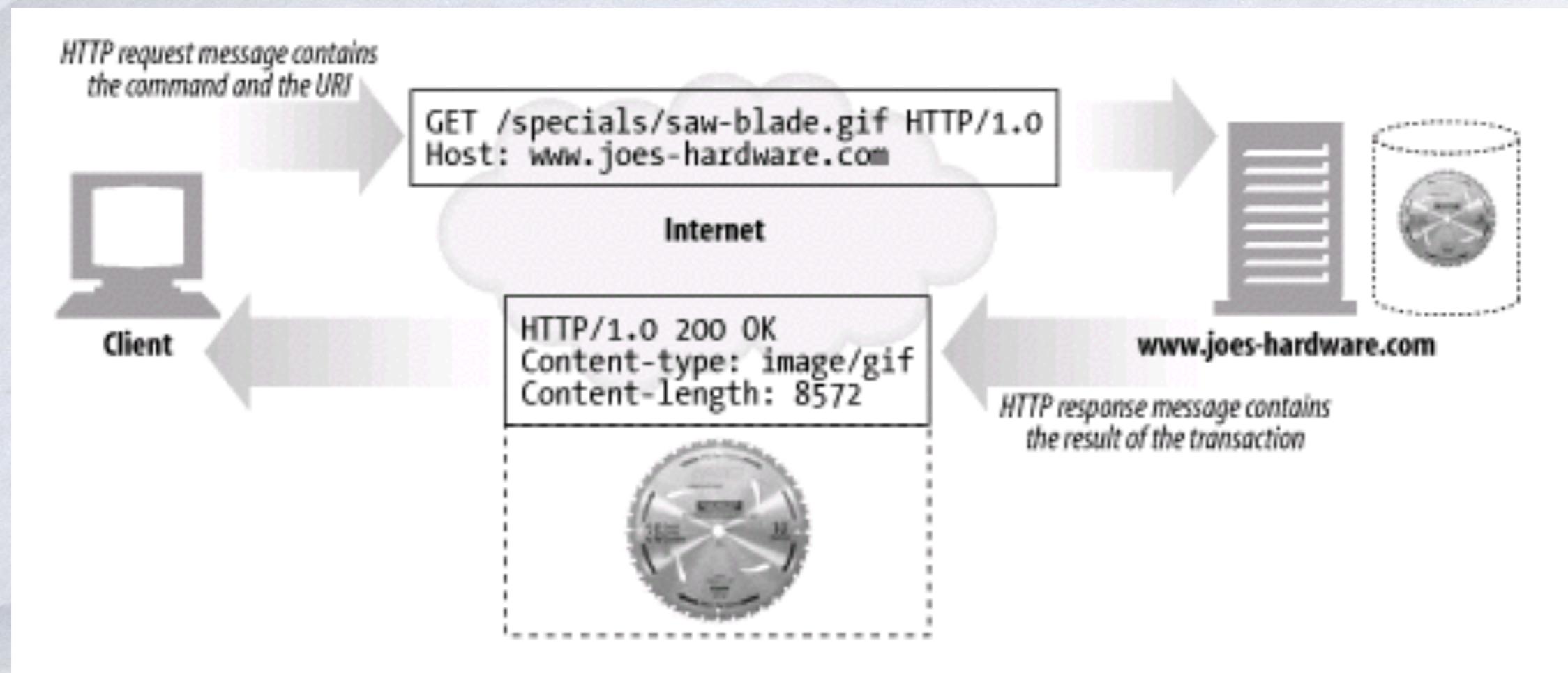
More on connection close

- ❖ Any HTTP client, server, or proxy can close a TCP transport connection at any time.
- ❖ Each HTTP response should have an accurate Content-Length header to describe the size of the response body.

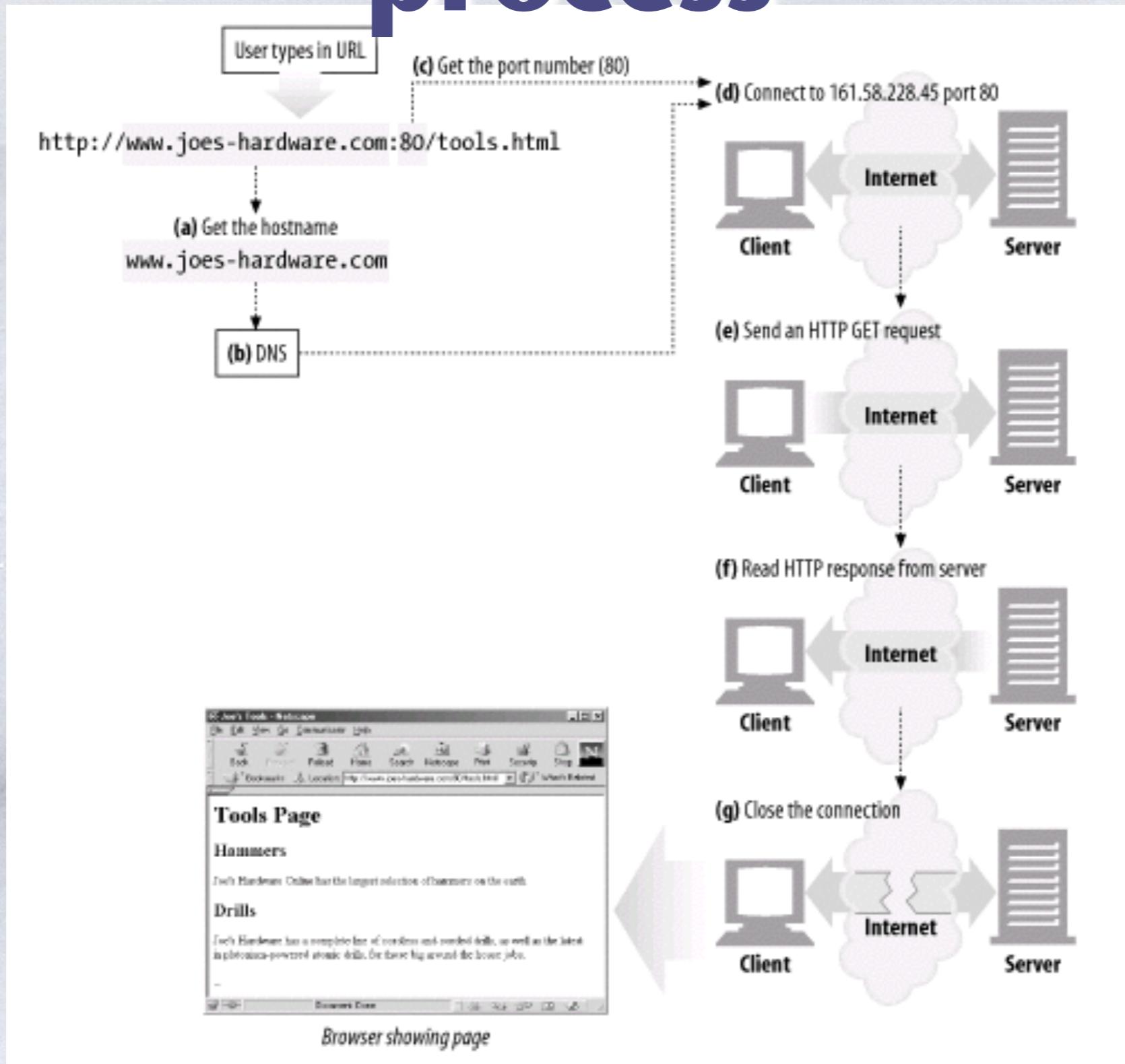
Graceful Connection Close

- ❖ First close output channel, then wait for the peer to close its output channel, finally, the connection can be closed fully.

HTTP transaction



Basic browser connection process



HTTP Messages

- ❖ two types of HTTP messages: request, response

(a) Request message

```
GET /test/hi-there.txt HTTP/1.0
```

```
Accept: text/*
Accept-Language: en,fr
```

(b) Response message

Start line

```
HTTP/1.0 200 OK
```

Headers

```
Content-type: text/plain
Content-length: 19
```

Body

```
Hi! I'm a message!
```

HTTP request message



HTTP request message:

- * ASCII (human-readable format)
- * Lines end with CRLF “\r\n”
- * First line is called “Request-Line”

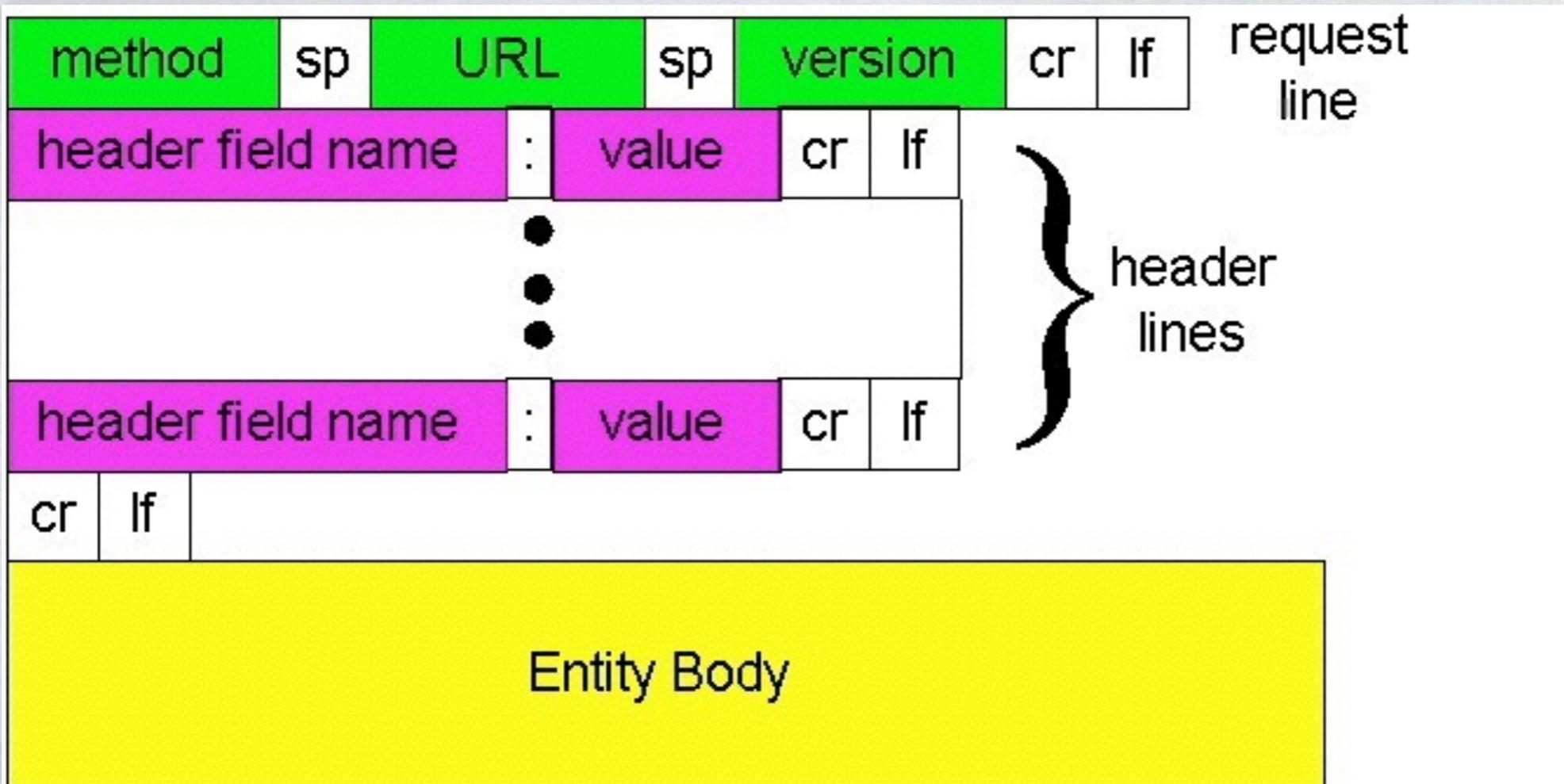
request line
(GET, POST,
HEAD commands)

header lines

→ GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr

Carriage return,
line feed
indicates end
of message → (extra carriage return, line feed)

HTTP request message general format



Request Line

Method URI HTTP-Version\r\n

- ❖ The request line contains 3 tokens (words).
- ❖ space characters “ ” separate the tokens.
- ❖ Newline (\n) seems to work by itself (but the protocol requires CRLF)

Request Method

* The Request Method can be:

GET HEAD PUT

POST DELETE TRACE

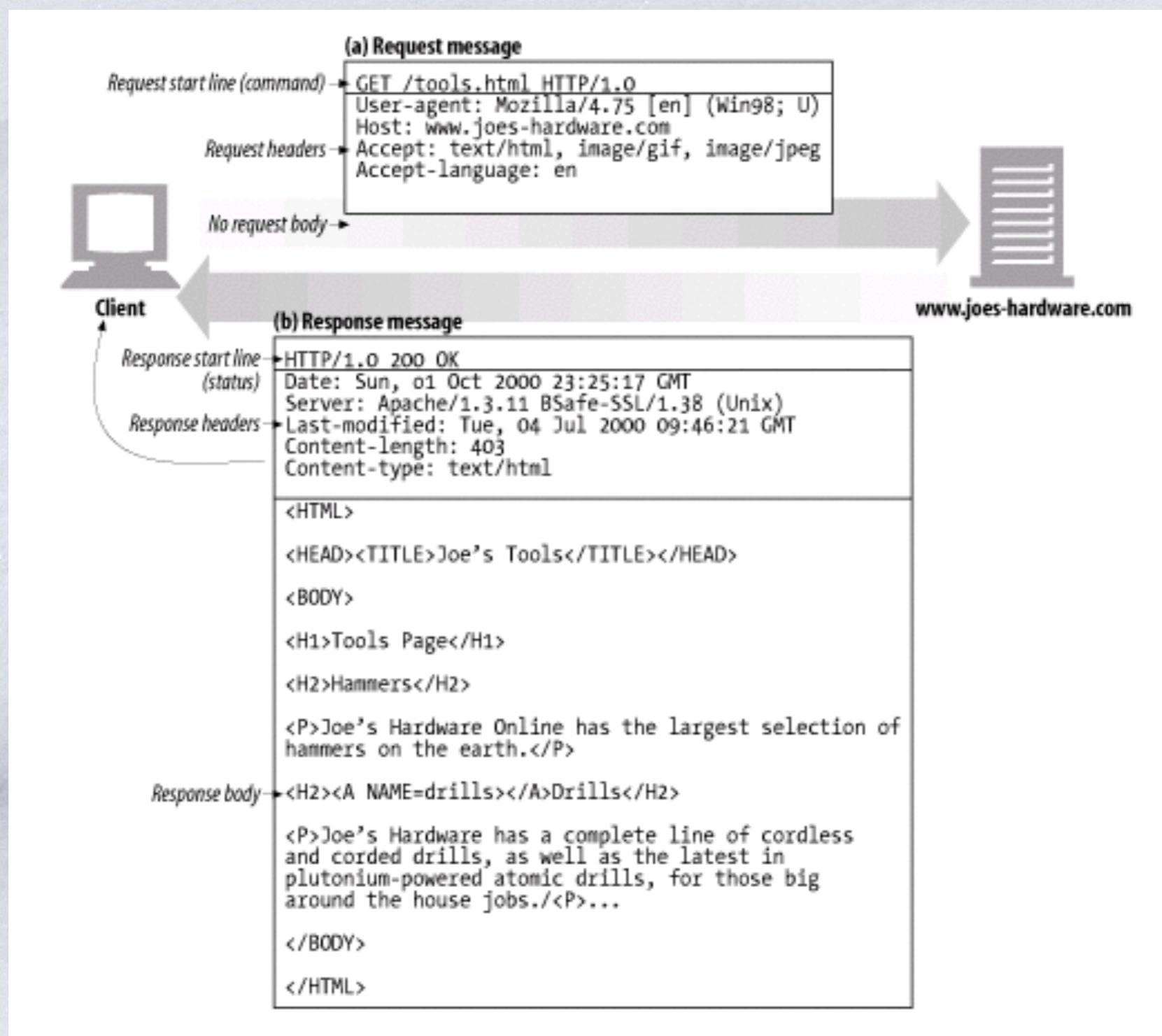
OPTIONS

* future expansion is supported

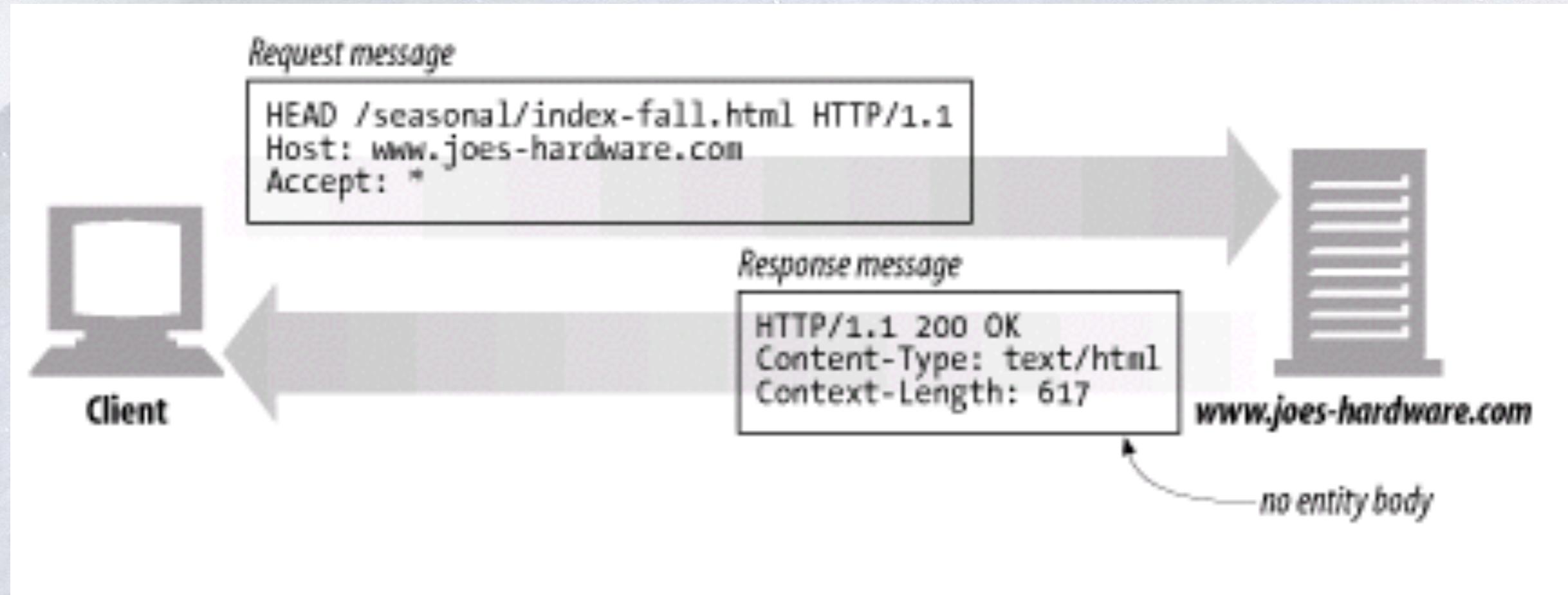
Methods

- ❖ GET: retrieve information identified by the URI.
- ❖ HEAD: retrieve meta-information about the URI.
- ❖ POST: send information to a URI and retrieve result.

Example of GET



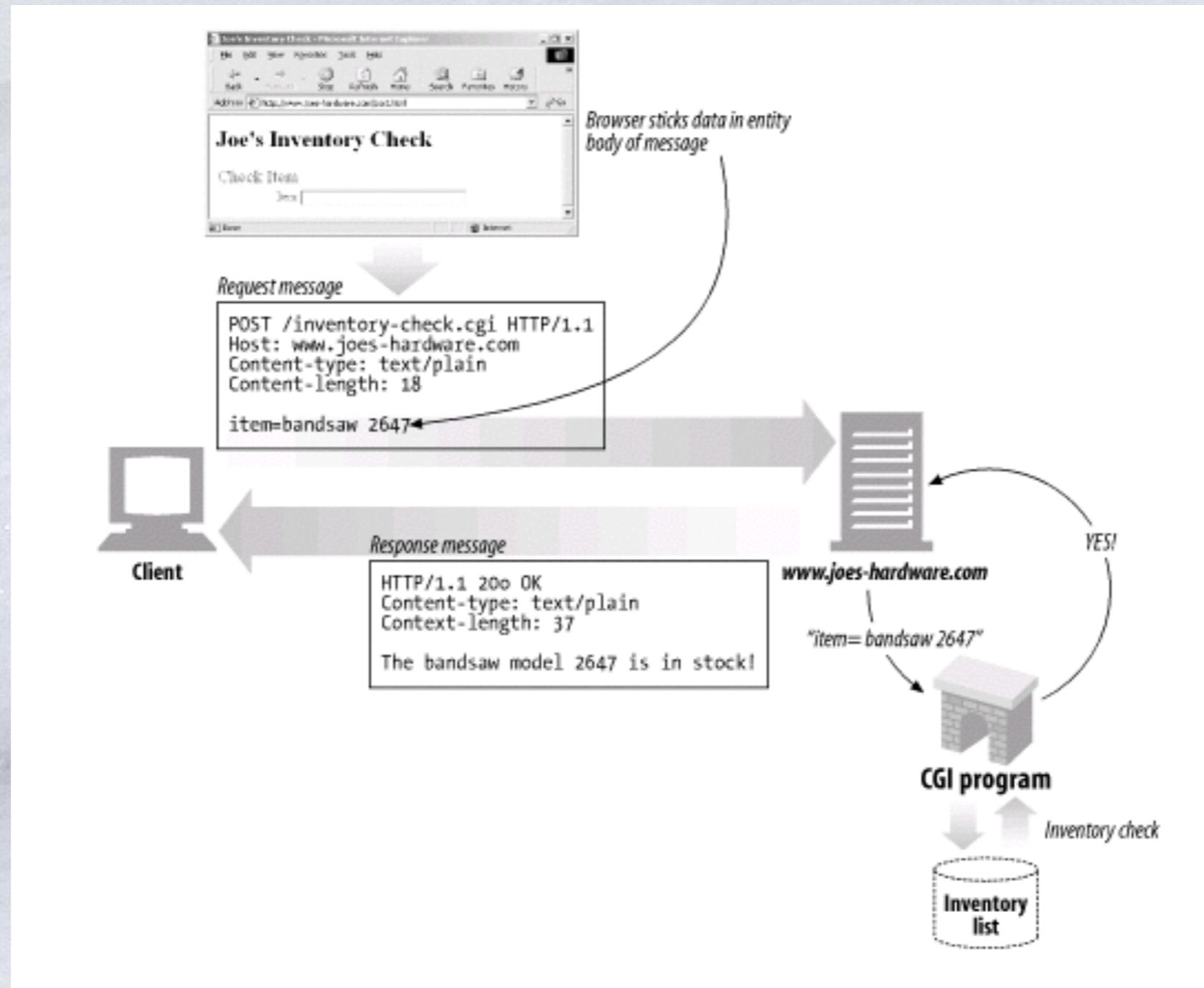
Example of HEAD



POST

- ❖ A POST request includes some content (some data) after the headers (after the blank line).
- ❖ There is no format for the data (just raw bytes).
- ❖ A POST request must include a Content-Length line in the headers:
Content-length: 267

Example of POST



Methods (cont.)

- ❖ PUT: Store information in location named by URI.
- ❖ DELETE: remove entity identified by URI.

Example of PUT



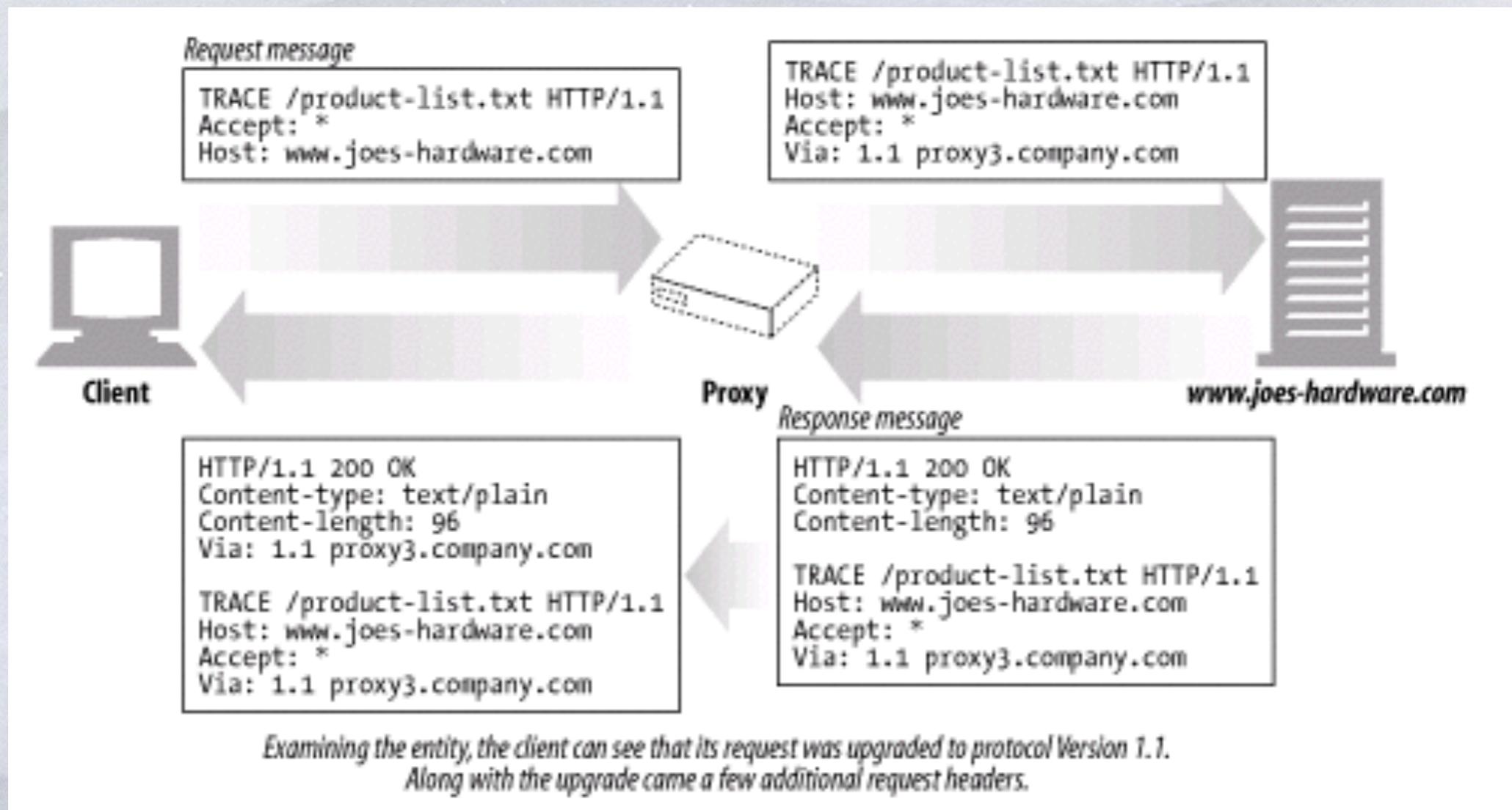
Example of DELETE



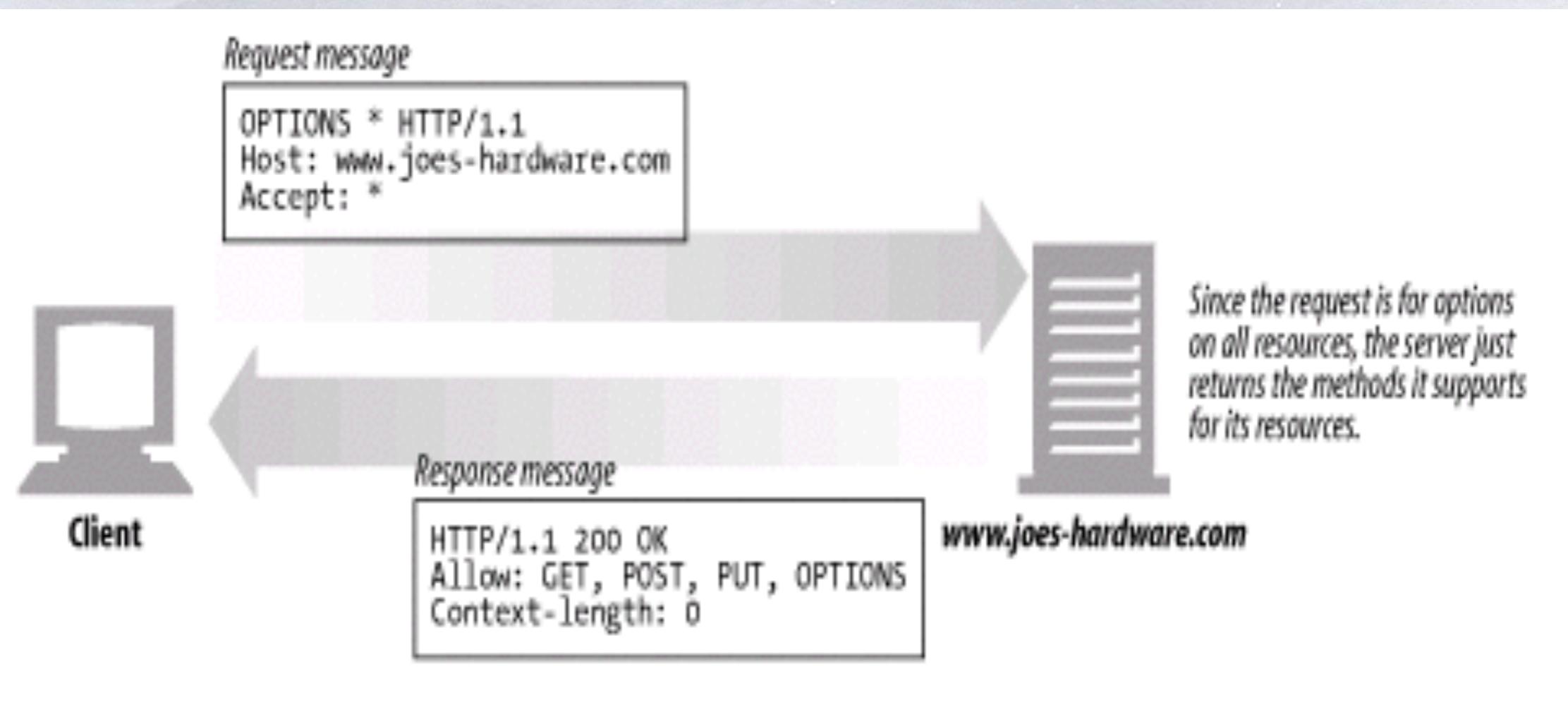
More Methods

- ❖ TRACE: used to trace HTTP forwarding through proxies, tunnels, etc.
- ❖ OPTIONS: used to determine the capabilities of the server, or characteristics of a named resource.

Example of TRACE



Example of OPTIONS



Method Common Usage

- ❖ GET, HEAD and POST are supported everywhere.
- ❖ HTTP 1.1 servers often support PUT, DELETE, OPTIONS & TRACE.

HTTP Version Number

- ❖ “HTTP/1.0” or “HTTP/1.1”
- ❖ HTTP 0.9 did not include a version number in a request line.
- ❖ If a server gets a request line with no HTTP version number, it assumes 0.9

The Header Lines

- ❖ After the Request-Line come a number (possibly zero) of HTTP header lines.
- ❖ Each header line contains an attribute name followed by a “:” followed by a space and the attribute value.
- ❖ The Name and Value are just text.

Headers

- ❖ Request Headers provide information to the server about the client
 - * what kind of client
 - * what kind of content will be accepted
 - * who is making the request
- ❖ There can be 0 headers (HTTP 1.0)
- ❖ HTTP 1.1 requires a Host: header

Example HTTP Headers

Accept: text/html

Host: www.nju.edu.cn

From: abc@nju.edu.cn

User-Agent: Mozilla/4.0

Referer: <http://test.com/abc>

General informational headers

Header	Description
Connection	Allows clients and servers to specify options about the request/response connection
Date	Provides a date and time stamp telling when the message was created
MIME-Version	Gives the version of MIME that the sender is using
Trailer	Lists the set of headers that are in the trailer of a message encoded with the chunked transfer encoding
Transfer-Encoding	Tells the receiver what encoding was performed on the message in order for it to be transported safely
Upgrade	Gives a new version or protocol that the sender would like to "upgrade" to using
Via	Shows what intermediaries (proxies, gateways) the message has gone through

General caching headers

Header	Description
Cache-Control	Used to pass caching directions along with the message
Pragma	Another way to pass directions along with the message, though not specific to caching

Request informational headers

Header	Description
Client-IP	Provides the IP address of the machine on which the client is running
From	Provides the email address of the client's user
Host	Gives the hostname and port of the server to which the request is being sent
Referer	Provides the URL of the document that contains the current request URI
UA-Color	Provides information about the color capabilities of the client machine's display
UA-CPU	Gives the type or manufacturer of the client's CPU
UA-Disp	Provides information about the client's display (screen) capabilities
UA-OS	Gives the name and version of operating system running on the client machine
UA-Pixels	Provides pixel information about the client machine's display
User-Agent	Tells the server the name of the application making the request

Request Accept headers

Header	Description
Accept	Tells the server what media types are okay to send
Accept-Charset	Tells the server what charsets are okay to send
Accept-Encoding	Tells the server what encodings are okay to send
Accept-Language	Tells the server what languages are okay to send
TE	Tells the server what extension transfer codings are okay to use

Conditional request headers

Header	Description
Expect	Allows a client to list server behaviors that it requires for a request
If-Match	Gets the document if the entity tag matches the current entity tag for the document
If-Modified-Since	Restricts the request unless the resource has been modified since the specified date
If-None-Match	Gets the document if the entity tags supplied do not match those of the current document
If-Range	Allows a conditional request for a range of a document
If-Unmodified-Since	Restricts the request unless the resource has not been modified since the specified date
Range	Requests a specific range of a resource, if the server supports range requests

Request security headers

Header	Description
Authorization	Contains the data the client is supplying to the server to authenticate itself
Cookie	Used by clients to pass a token to the server—not a true security header, but it does have security implications
Cookie2	Used to note the version of cookies a requestor supports

Proxy request headers

Header	Description
Max-Forwards	The maximum number of times a request should be forwarded to another proxy or gateway on its way to the origin server—used with the TRACE method
Proxy-Authorization	Same as Authorization, but used when authenticating with a proxy
Proxy-Connection	Same as Connection, but used when establishing connections with a proxy

End of the Headers

- ❖ Each header ends with a CRLF (\r\n)
- ❖ The end of the header section is marked with a blank line.
 - * just CRLF
- ❖ For GET and HEAD requests, the end of the headers is the end of the request!

HTTP response message

- * ASCII Status Line
- * Headers Section
- * Content can be anything (not just text)
- * typically an HTML document or some kind of image.

status line
(protocol
status code
status phrase)

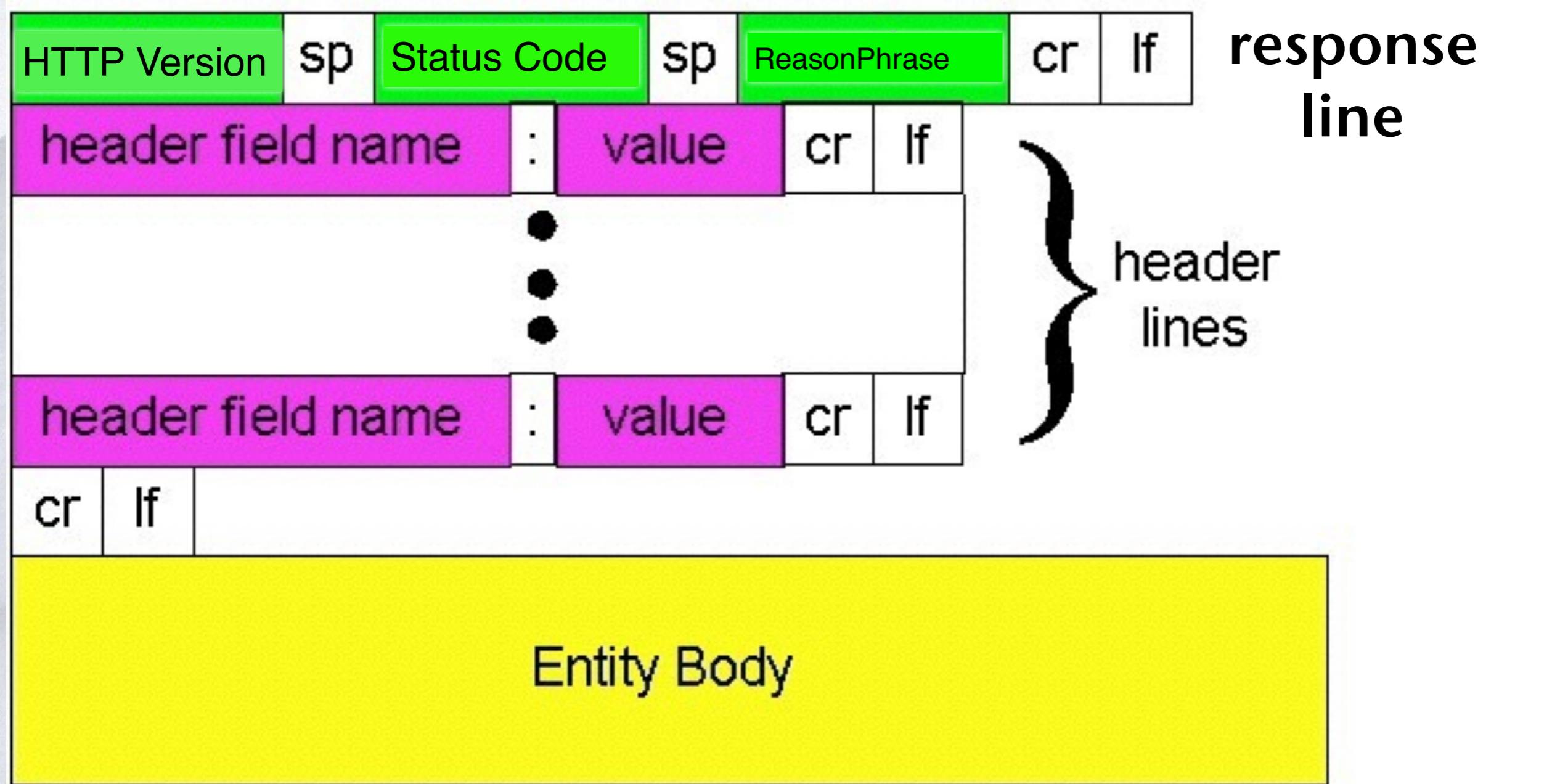
→ HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998
12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun
1998 ...
Content-Length: 6821
Content-Type: text/html

header lines

data, e.g.,
requested
HTML file

→ data data data data data ...

HTTP response message: general format



Response Status Line

HTTP-Version Status-Code Reason-Phrase

- ❖ Status Code is 3 digit number (for computers)
- ❖ Reason-Phrase is text (for humans)

Status Codes

1xx Informational

2xx Success

3xx Redirection

4xx Client Error

5xx Server Error

Common Status Codes

200 OK

301 Moved Permanently

400 Bad Request

401 Unauthorized

403 forbidden

404 Not Found

500 Internal Server Error

Informational status codes

100 Continue

101 switching protocols

Success status codes

200 OK

201 Created

202 Accepted

203 Non-Authoritative Information

204 No Content

205 Reset Content

206 Partial Content

Redirection status codes

300 multiple choices

301 Moved Permanently

302 Found

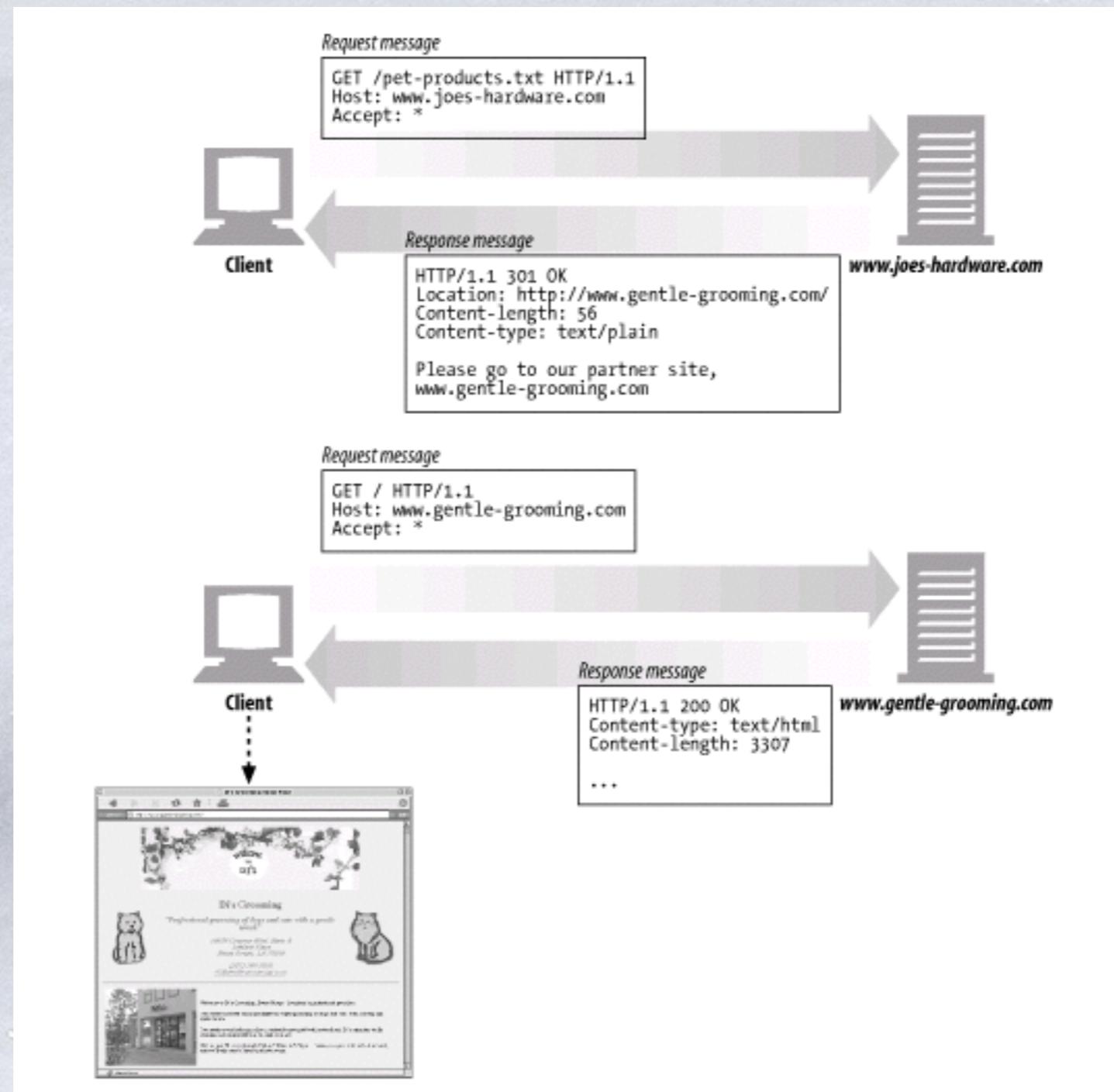
303 See Other

304 Not Modified

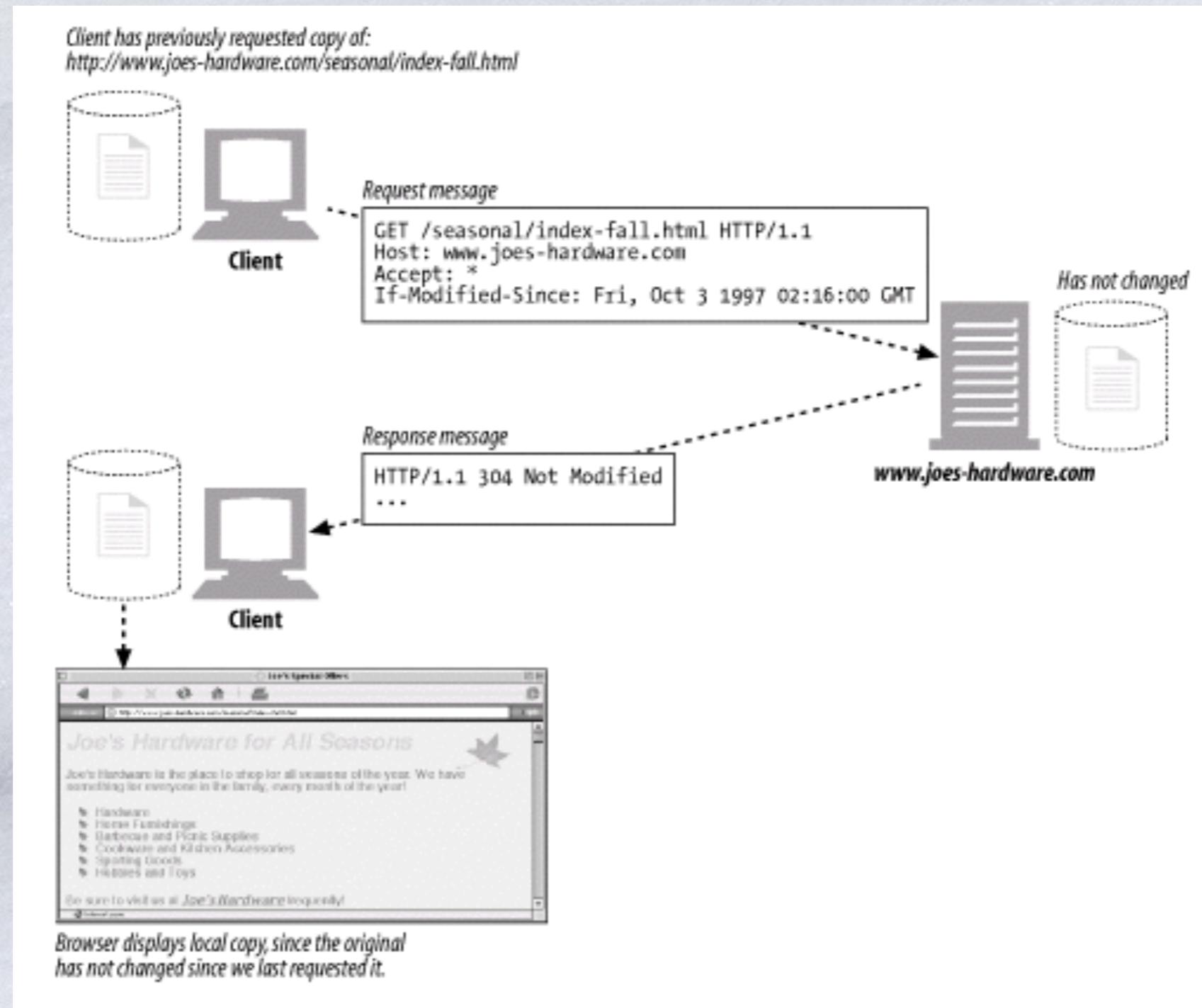
305 Use Proxy

307 Temporary Redirect

301 status code example



304 status code example



Client Error Status Codes

400 Bad Request

401 Unauthorized

403 Forbidden

404 Not Found

405 Method Not Allowed

406 Not Acceptable

407 Proxy Authentication Required

408 Request Timeout

409 Conflict

410 Gone

Client Error Status Codes

(cont.)

411 Length Required

412 Precondition Failed

413 Request Entity Too Large

414 Request URI Too Long

415 Unsupported Media Type

416 Requested Range Not Satisfiable

417 Expectation Failed

Server Error Status Codes

500 Internal Server Error

501 Not Implemented

502 Bad Gateway

503 Service Unavailable

504 Gateway Timeout

505 HTTP Version Not Supported

Response Headers

❖ Provide the client with information about the returned entity (document).

- * what kind of document
- * how big the document is
- * how the document is encoded
- * when the document was last modified

❖ Response headers end with blank line

Response Header Examples

Date: Wed, 30 Jan 2002 12:48:17 EST

Server: Apache/1.17

Content-Type: text/html

Content-Length: 1756

Content-Encoding: gzip

Response informational headers

Header	Description
Age	How old the response is
Public	A list of request methods the server supports for its resources
Retry-After	A date or time to try back, if a resource is unavailable
Server	The name and version of the server's application software
Title	For HTML documents, the title as given by the HTML document source
Warning	A more detailed warning message than what is in the reason phrase

Negotiation headers

Header	Description
Accept-Ranges	The type of ranges that a server will accept for this resource
Vary	A list of other headers that the server looks at and that may cause the response to vary; i.e., a list of headers the server looks at to pick which is the best version of a resource to send the client

Response security headers

Header	Description
Proxy-Authenticate	A list of challenges for the client from the proxy
Set-Cookie	Not a true security header, but it has security implications; used to set a token on the client side that the server can use to identify the client
Set-Cookie2	Similar to Set-Cookie, RFC 2965 Cookie definition
WWW-Authenticate	A list of challenges for the client from the server

Entity informational headers

Header	Description
Allow	Lists the request methods that can be performed on this entity
Location	Tells the client where the entity really is located; used in directing the receiver to a (possibly new) location (URL) for the resource

Entity content headers

Header	Description
Content-Base	The base URL for resolving relative URLs within the body
Content-Encoding	Any encoding that was performed on the body
Content-Language	The natural language that is best used to understand the body
Content-Length	The length or size of the body
Content-Location	Where the resource actually is located
Content-MD5	An MD5 checksum of the body
Content-Range	The range of bytes that this entity represents from the entire resource
Content-Type	The type of object that this body is

Entity caching headers

Header	Description
ETag	The entity tag associated with this entity
Expires	The date and time at which this entity will no longer be valid and will need to be fetched from the original source
Last-Modified	The last date and time when this entity changed

Content

- ❖ Content can be anything (sequence of raw bytes).
- ❖ Content-Length header is required for any response that includes content.
- ❖ Content-Type header also required.

Conditional GET: client-side caching

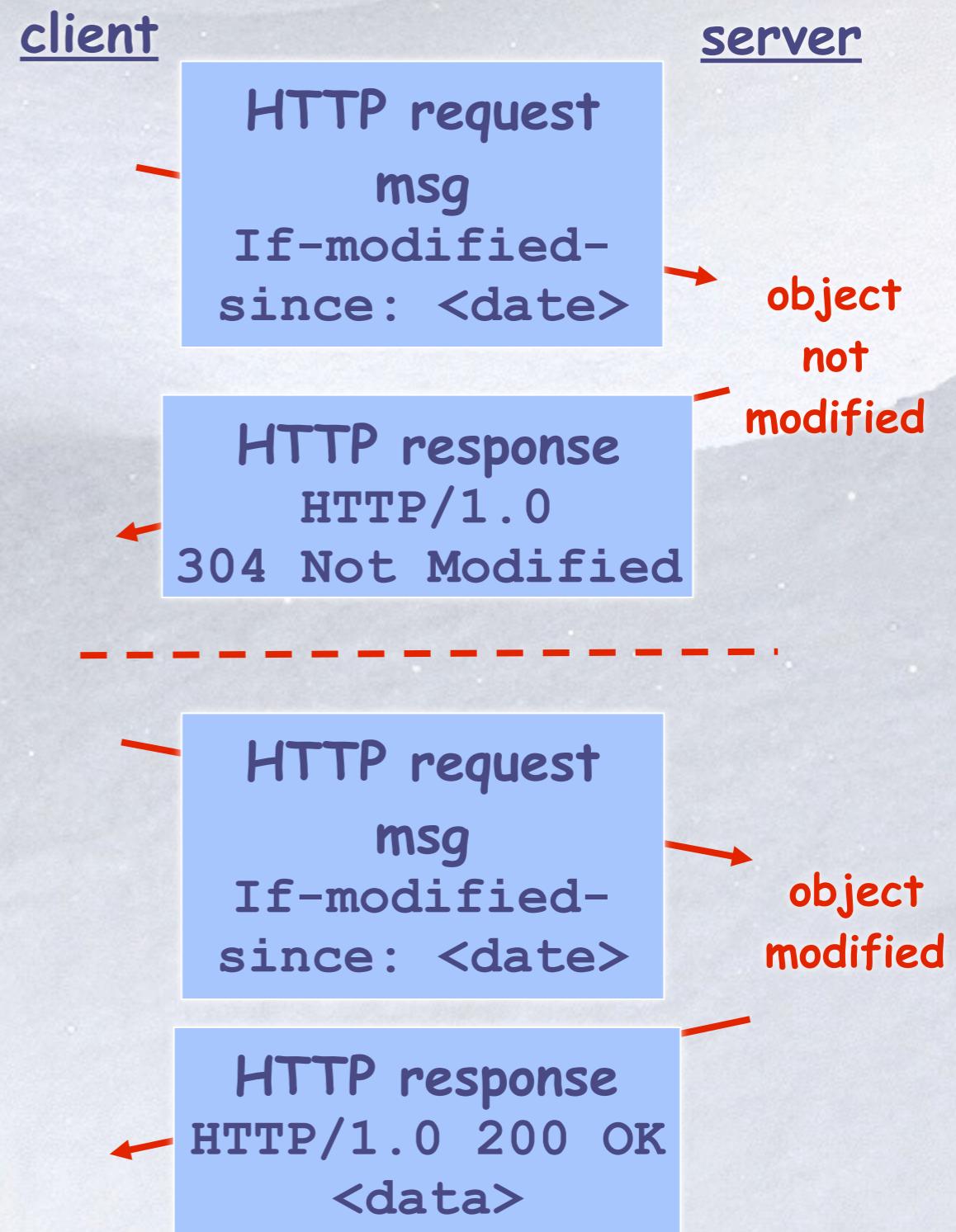
❖ Goal: don't send object if client has up-to-date cached version

❖ 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



Conditional Get

❄ ask to receive only if current object stale

```
GET /instruction/default.html HTTP/1.1  
...  
User-agent: Mozilla/4.0  
Accept: text/html, image/gif  
If-modified-since: Sat, 1 April 2000  
06:23:19
```

```
HTTP/1.1 200 OK  
Date: Thu, 6 April 2000 12:00:15 GMT  
Server: Apache/1.3.0 (Unix)  
Last-Modified: Sat, 1 Apr 2000 06:23:19 GMT  
Content-Length: 6821  
Content-Type: text/html  
data data data data data ...
```

Try it with telnet

```
> telnet www.xxx.edu.cn 80
```

```
GET / HTTP/1.0
```

```
HTTP/1.0 200 OK
```

```
Server: Apache
```

```
...
```

Request-line

Blank Line
(end of headers)

Response

Try it with telnet 1.1

```
> telnet www.xxx.edu.cn 80
```

```
GET / HTTP/1.1
```

```
Host: www.xxx.edu.cn
```

```
HTTP/1.0 200 OK
```

```
Server: Apache
```

```
...
```

Required!

Outline

- ❄ HTTP Overview
- ❄ HTTP Request & Response
- ❄ Web Application Design

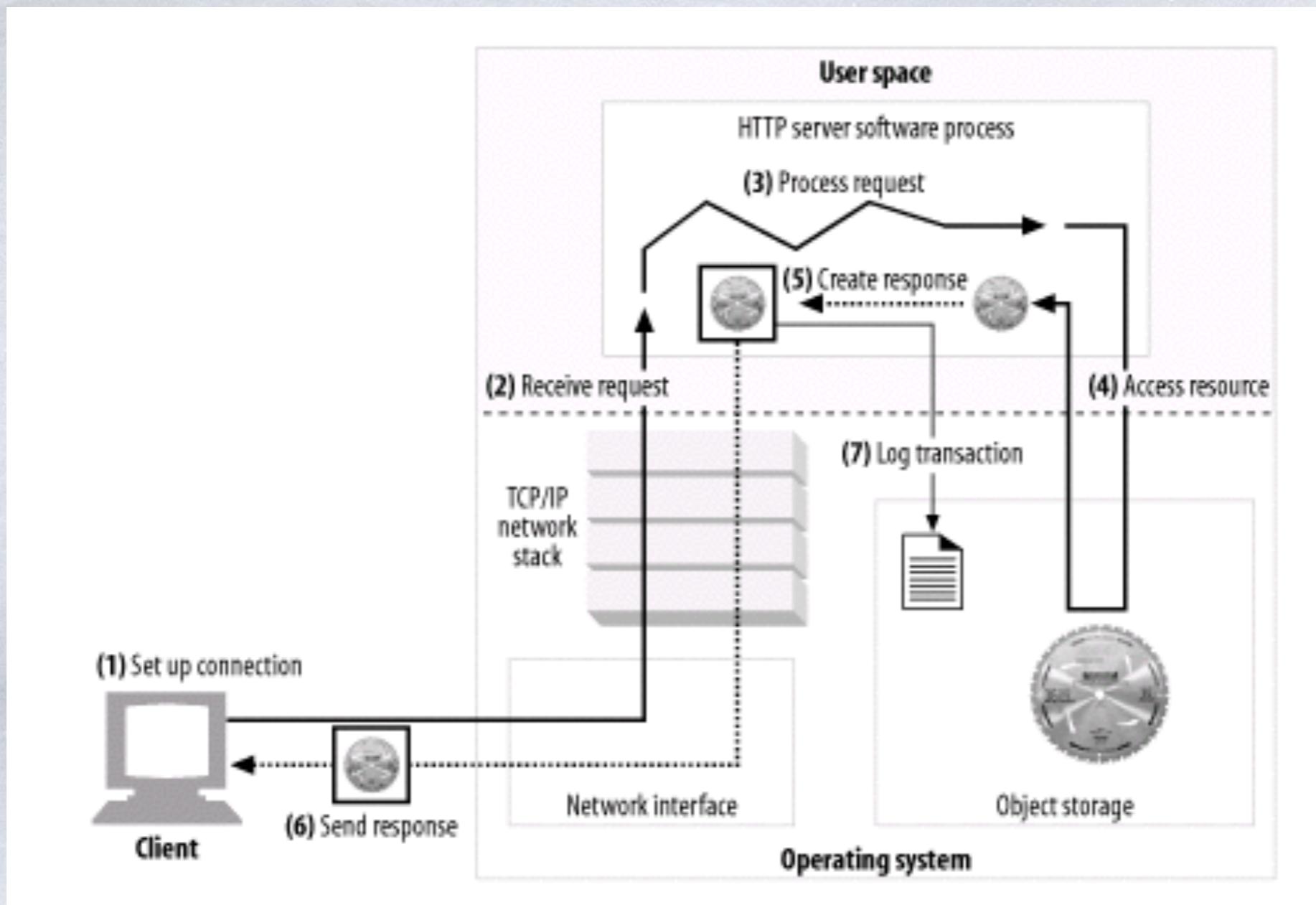
WEB Server design

❖ Web Server market

- * apache vs Microsoft web server vs Sun iPlanet servers

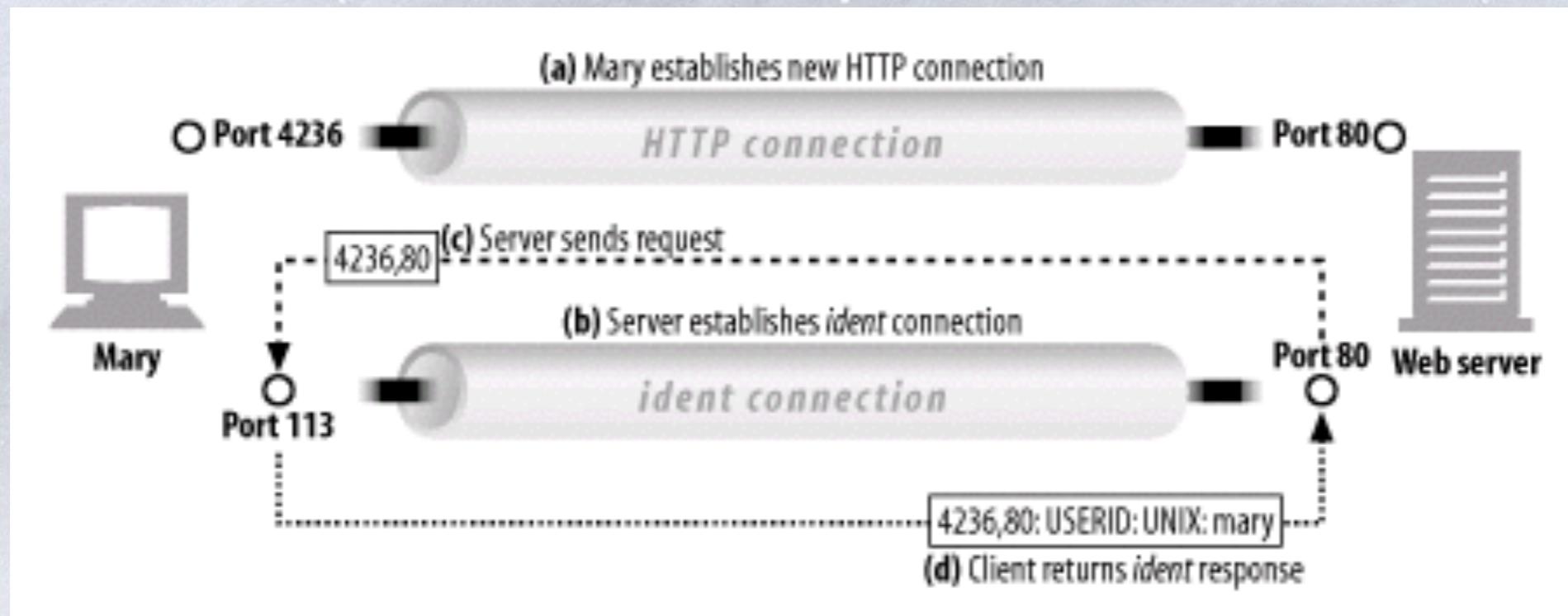
❖ Web Server size from 30 lines Perl script to 50MB secure commerce engines

Real Web Servers Do



Step 1: Accepting Client Connections

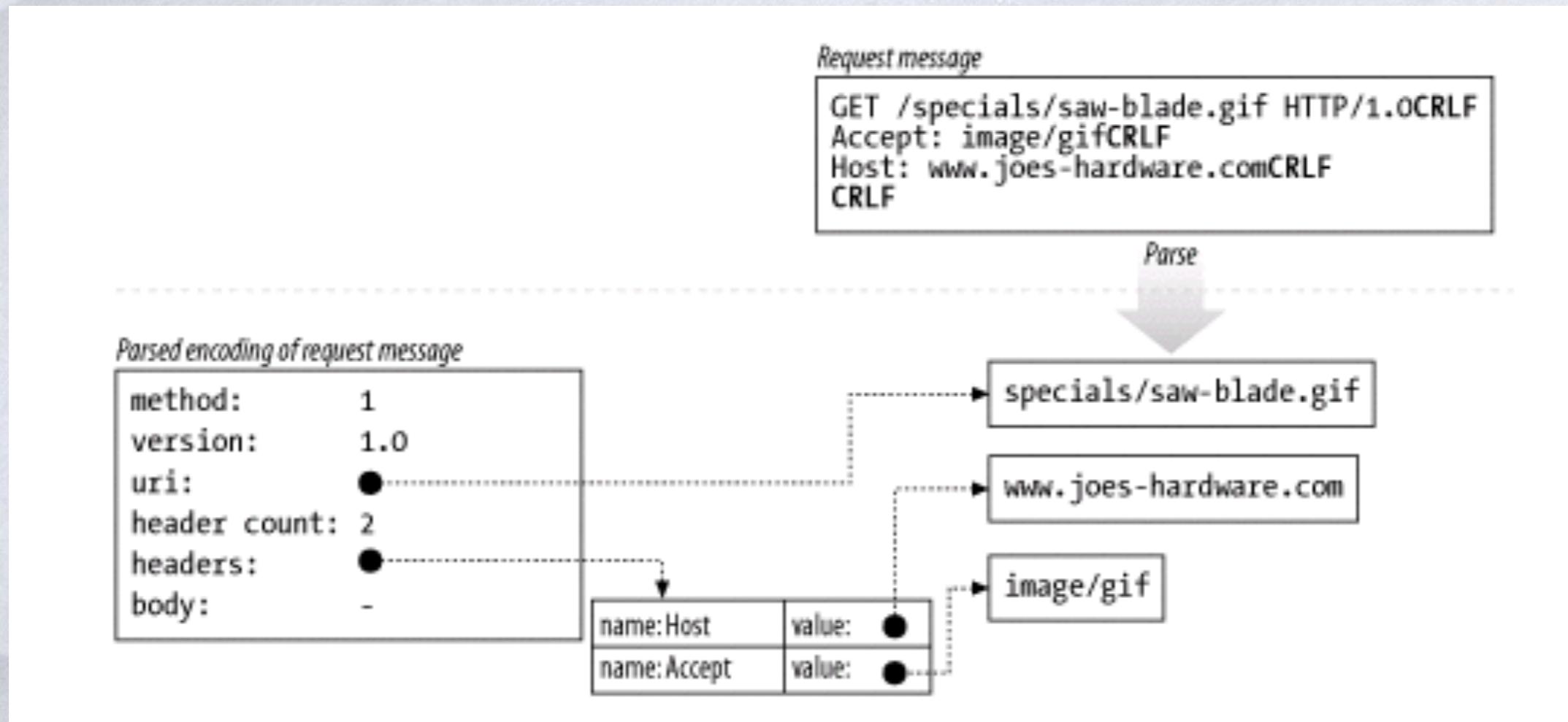
- ❖ Handling new connections
- ❖ Client hostname identification
- ❖ Determining the client user through ident



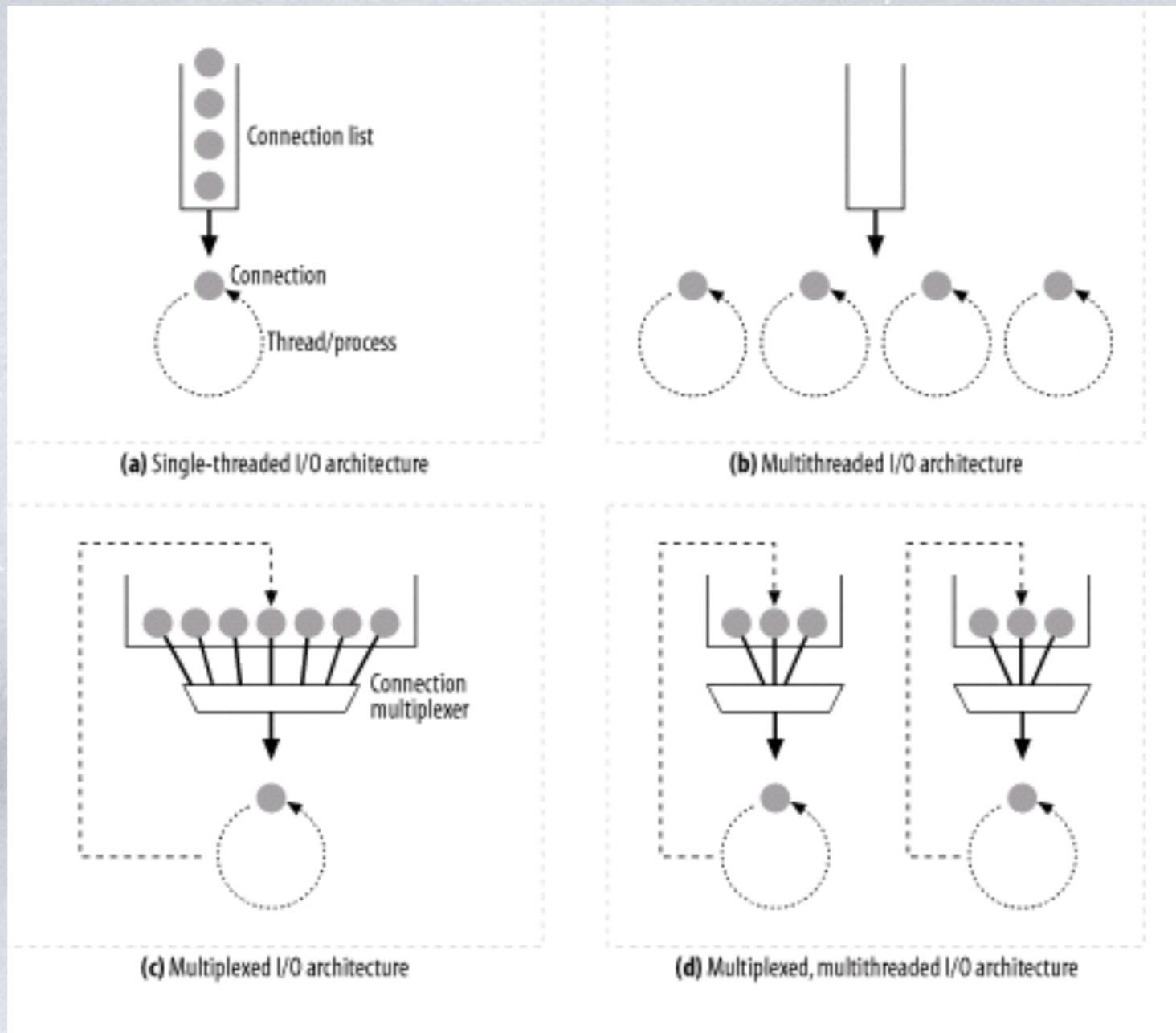
Step 2: Receiving Request Messages

- ❖ Parses the request line looking for the request method, the specified resource identifier (URI), and the version number, each separated by a single space, and ending with a carriage-return line-feed (CRLF) sequence
- ❖ Reads the message headers, each ending in CRLF
- ❖ Detects the end-of-headers blank line, ending in CRLF (if present)
- ❖ Reads the request body, if any (length specified by the Content-Length header)

Parsing a request message into a convenient internal representation



Connection processing architectures



Step 3: Processing Requests

- ❖ Once the web server has received a request, it can process the request using the method, resource, headers, and optional body.

Step 4: Mapping and Accessing Resources

Docroots

- * Virtually hosted docroots
- * User home directory docroots

Directory Listings

Dynamic Content Resource Mapping

Server-Side Includes(SSI)

Access Control

Step 5: Building Responses

❖ Response Entities

- * A Content-Type header, describing the MIME type of the response body
- * A Content-Length header, describing the size of the response body
- * The actual message body content

❖ redirection

- * Permanently moved resources
- * Temporarily moved resources
- * URL augmentation
- * Load balancing
- * Canonicalizing directory names

Step 6: Sending Responses

- ❖ For nonpersistent connections, the server is expected to close its side of the connection when the entire message is sent.
- ❖ For persistent connections, the server needs to be extra cautious to compute the Content-Length header correctly, or the client will have no way of knowing when a response ends

Web Client design

❖ Web crawler, spider, Web Robot

❖ Definition:

- * A Web crawler is a computer program that browses the World Wide Web in a methodical, automated manner. (Wikipedia)

❖ Utilities:

- * Gather pages from the Web.
- * Support a search engine, perform data mining and so on.

❖ Object:

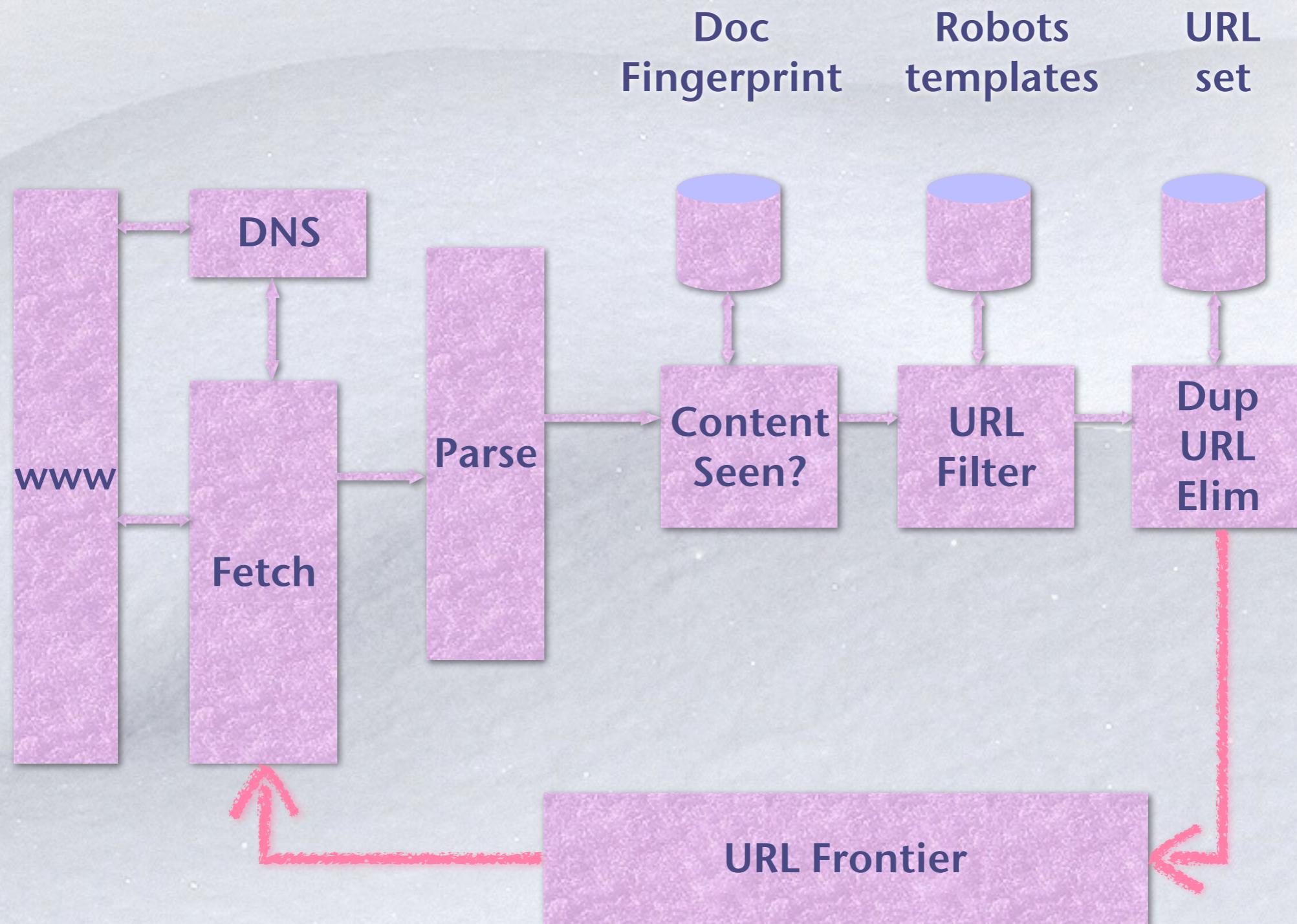
- * Text, video, image and so on.
- * Link structure.

Features of a crawler

❖ Should provide:

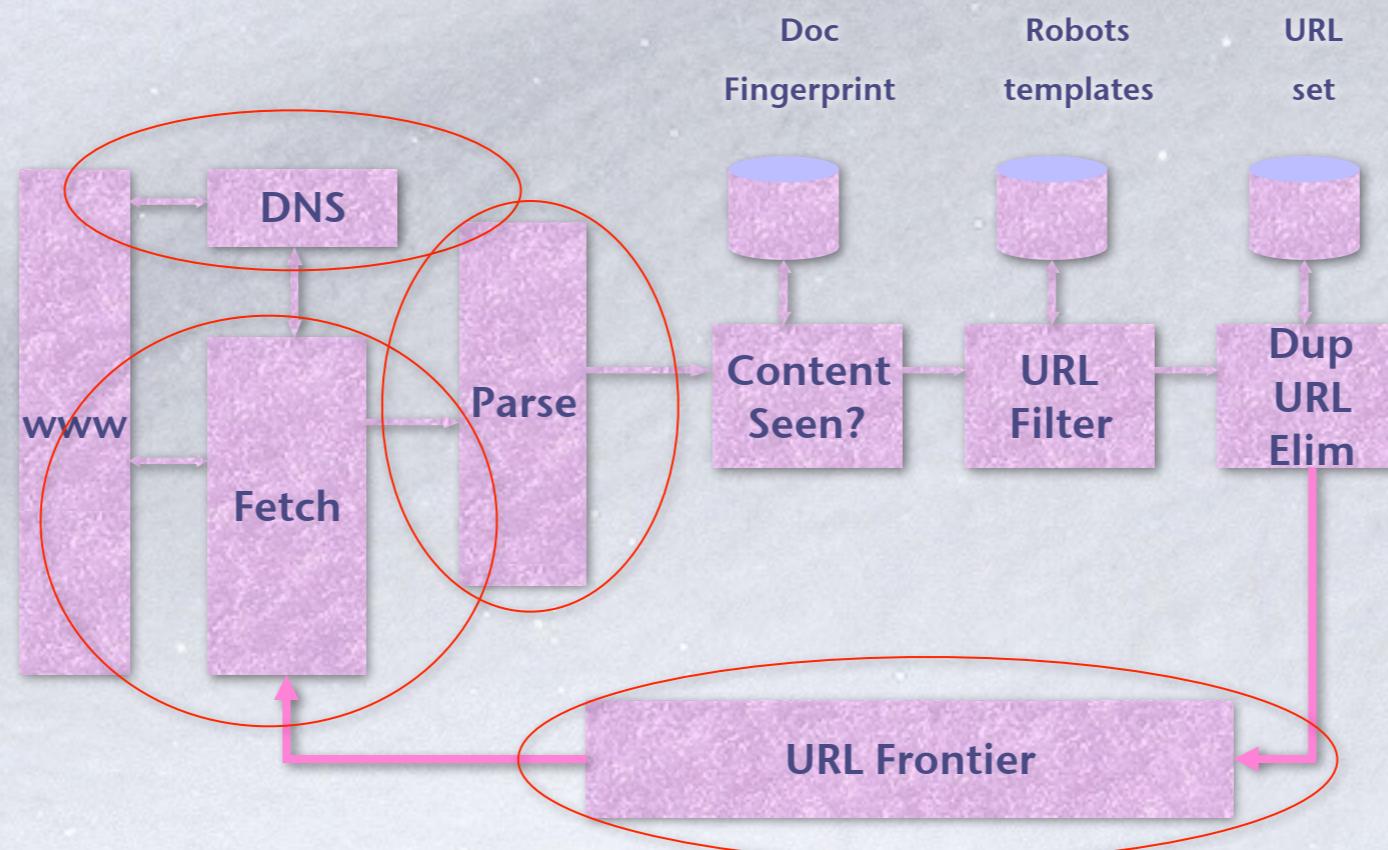
- * Distributed
- * Scalable
- * Performance and efficiency
- * Quality
- * Freshness
- * Update
- * Extensible

Architecture of a crawler



Architecture of a crawler (Cont'd)

- * URL Frontier: containing URLs yet to be fetched in the current crawl. At first, a seed set is stored in URL Frontier, and a crawler begins by taking a URL from the seed set.
- * DNS: domain name service resolution. Look up IP address for domain names.
- * Fetch: generally use the http protocol to fetch the URL.
- * Parse: the page is parsed. Texts (images, videos, and etc.) and Links are extracted.



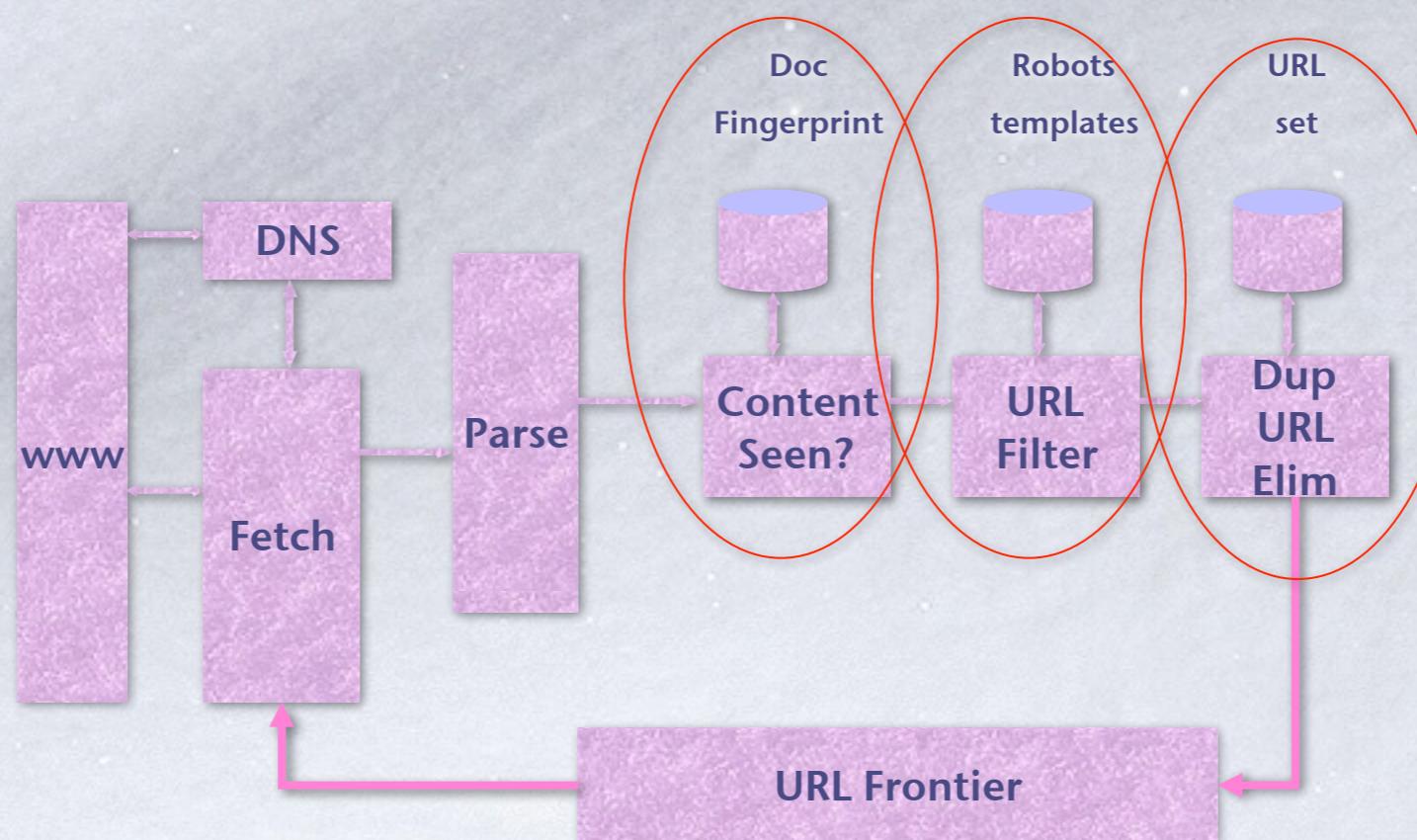
Architecture of a crawler (Cont'd)

* **Content Seen?:** test whether a web page with the same content has already been seen at another URL. Need to develop a way to measure the fingerprint of a web page

* **URL Filter:** Whether the extracted URL should be excluded from the frontier (`robots.txt`).

- * URL should be normalized (relative encoding).
- * `en.wikipedia.org/wiki/Main_Page`
- * `Disclaimers`

* **Dup URL Elim:** the URL is checked for duplicate elimination.



Crawlers

- ❖ Where to Start: The "Root Set"
- ❖ Extracting Links and Normalizing Relative Links
- ❖ Avoiding Loops and Dups
 - * Canonicalizing URLs
 - * Breadth-first crawling
 - * Throttling
 - * Limit URL size
 - * URL/site blacklist
 - * Pattern detection
 - * Content fingerprinting

Robots.txt

- ❖ Protocol for giving spiders (“robots”) limited access to a website, originally from 1994
 - * www.robotstxt.org/wc/norobots.html
- ❖ Website announces its request on what can(not) be crawled
 - * For a URL, create a file URL/robots.txt
 - * This file specifies access restrictions

Robots.txt example

- ❖ No robot should visit any URL starting with "/yoursite/temp/", except the robot called “Ix BOT”:

User-agent: *

Disallow: /yoursite/temp/

User-agent: Ix BOT

Disallow:

References

- ❄ **HTTP History**
 - * <http://www.apacheweek.com/features/http11>
- ❄ **RFC2616 HTTP1.1**
- ❄ **RFC1945 HTTP1.0**
- ❄ **David Gourley,etc. HTTP: The Definitive Guide**
- ❄ **Tunneling TCP based protocols through Web proxy servers**
 - * <http://tools.ietf.org/id/draft-luotonen-web-proxy-tunneling-01.txt>

Thanks!!!

