Lecture 8
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)

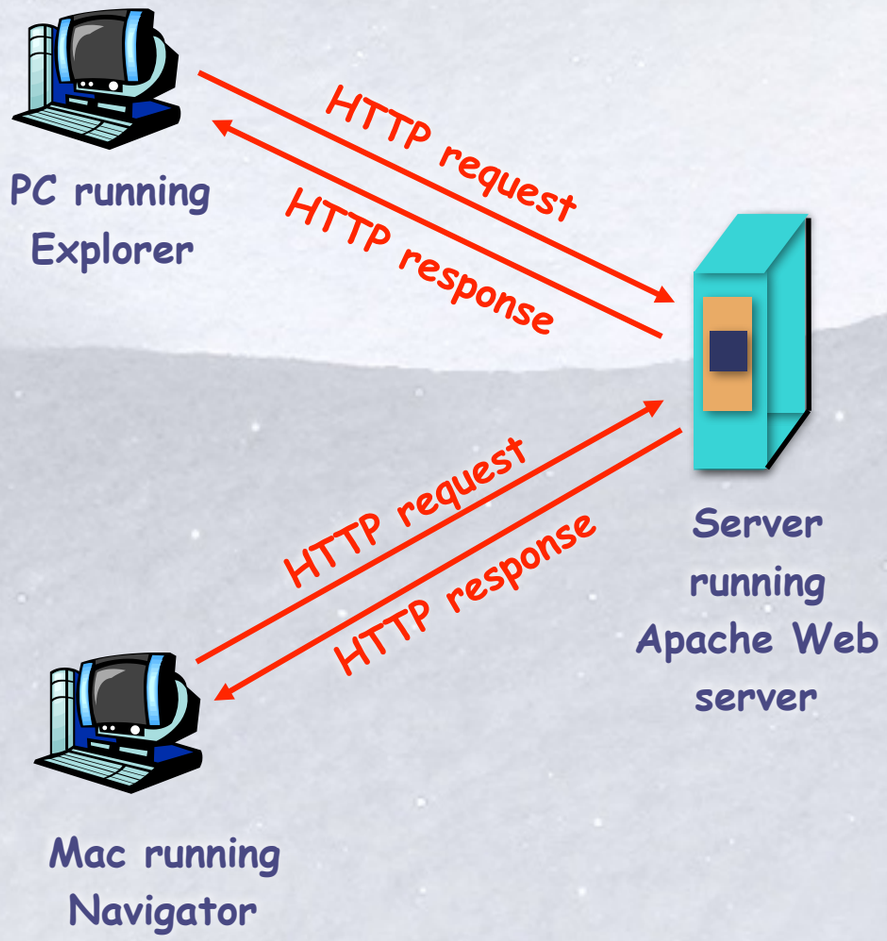
www.someschool.edu/someDept/pic.gif

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
- HTTP/2 (originally named HTTP/2.0)
  - RFC 7540 in May 2015

HTTPS

- HTTPS (also called HTTP over TLS, HTTP over SSL, and HTTP Secure) is a protocol for secure communication over a computer network which is widely used on the Internet.
- HTTPS consists of communication over Hypertext Transfer Protocol (HTTP) within a connection encrypted by Transport Layer Security or its predecessor, Secure Sockets Layer.
- The main motivation for HTTPS is authentication of the visited website and protection of the privacy and integrity of the exchanged data.

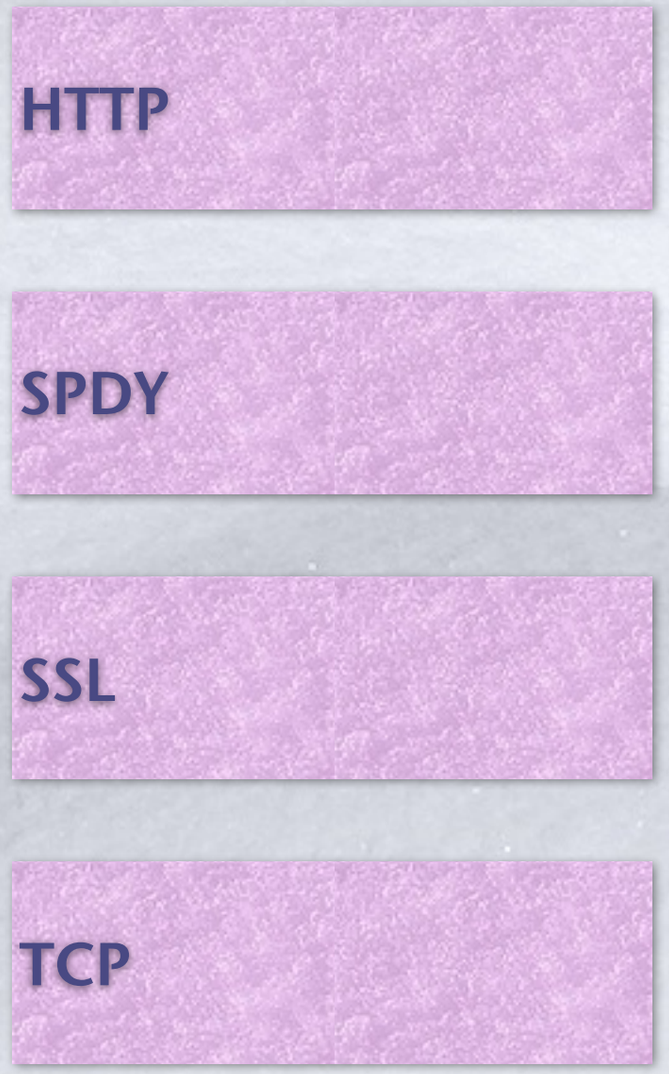
Difference from HTTP

- HTTPS URLs begin with "https://" and use port 443 by default, whereas HTTP URLs begin with "http://" and use port 80 by default.
- HTTP is not encrypted and is vulnerable to man-in-the-middle and eavesdropping attacks, which can let attackers gain access to website accounts and sensitive information, and modify webpages to inject malware or advertisements. HTTPS is designed to withstand such attacks and is considered secure against them (with the exception of older, deprecated versions of SSL).

Goals of SPDY

- Target a 50% reduction in page load time (PLT).
- Avoid the need for any changes to content by website authors.
- Minimize deployment complexity, avoid changes in network infrastructure.
- Develop this new protocol in partnership with the open-source community.
- Gather real performance data to (in)validate the experimental protocol.

SPDY



Relation to HTTP

- SPDY does not replace HTTP; it modifies the way HTTP requests and responses are sent over the wire. This means that all existing server-side applications can be used without modification if a SPDY-compatible translation layer is put in place.
- SPDY is effectively a tunnel for the HTTP and HTTPS protocols. When sent over SPDY, HTTP requests are processed, tokenized, simplified and compressed.
  - For example, each SPDY endpoint keeps track of which headers have been sent in past requests and can avoid resending the headers that have not changed; those that must be sent are compressed.
- The IETF working group for HTTPbis has released the draft of HTTP/2. SPDY was chosen as the starting point

New Functions

- multiplexing
- request prioritization
- header compression
- server push
- server hint

Result

Table 1: Average page load times for top 25 websites

	DSL 2 Mbps downlink, 375 kbps uplink		Cable 4 Mbps downlink, 1 Mbps uplink	
	Average ms	**Speedup**	**Average ms**	**Speedup**
HTTP	3111.916		2348.188	
SPDY basic multi-domain* connection / TCP	2242.756	27.93%	1325.46	43.55%
SPDY basic single-domain* connection / TCP	1695.72	45.51%	933.836	60.23%
SPDY single-domain + server push / TCP	1671.28	46.29%	950.764	59.51%
SPDY single-domain + server hint / TCP	1608.928	48.30%	856.356	63.53%
SPDY basic single-domain / SSL	1899.744	38.95%	1099.444	53.18
SPDY single-domain + client prefetch / SSL	1781.864	42.74%	1047.308	55.40%

https://www.chromium.org/spdy/spdy-whitepaper

RTT

Table 3: Average page load times for top 25 websites by RTT

	Average ms		**Speedup**
RTT in ms	**HTTP**	**SPDY basic (TCP)**	
20	1240	1087	12.34%
40	1571	1279	18.59%
60	1909	1526	20.06%
80	2268	1727	23.85%
120	2927	2240	23.47%
160	3650	2772	24.05%
200	4498	3293	26.79%

https://www.chromium.org/spdy/spdy-whitepaper

HTTP/2

- HTTP/2 (originally named HTTP/2.0) is a major revision of the HTTP network protocol used by the World Wide Web.
- It was developed from the earlier experimental SPDY protocol, originally developed by Google.
- HTTP/2 was developed by the Hypertext Transfer Protocol working group (httpbis, where bis means "second") of the Internet Engineering Task Force.

Differences from HTTP 1.1

- The proposed changes do not require any changes to how existing web applications work, but new applications can take advantage of new features for increased speed.
- HTTP/2 leaves most of HTTP 1.1's high-level syntax, such as methods, status codes, header fields, and URIs, the same. The element that is modified is how the data is framed and transported between the client and the server.
- Websites that are efficient minimize the number of requests required to render an entire page
- Additional performance improvements in the first draft of HTTP/2 (which was a copy of SPDY) come from multiplexing of requests and responses to avoid the head-of-line blocking problem in HTTP 1 (even when HTTP pipelining is used), header compression, and prioritization of requests.

Outline

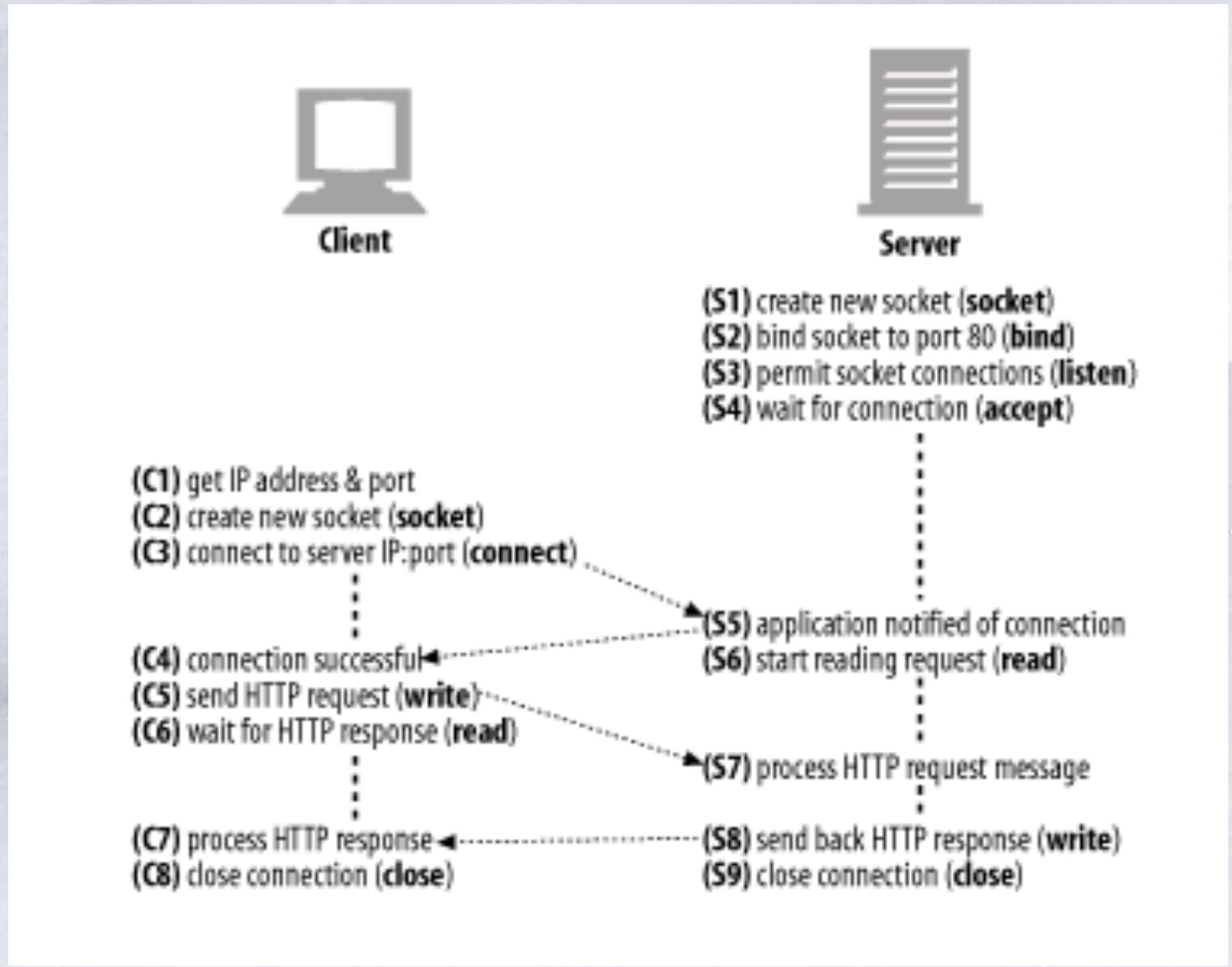
- HTTP Overview
- HTTP Request & Response
- Web Application Design

Request - Response

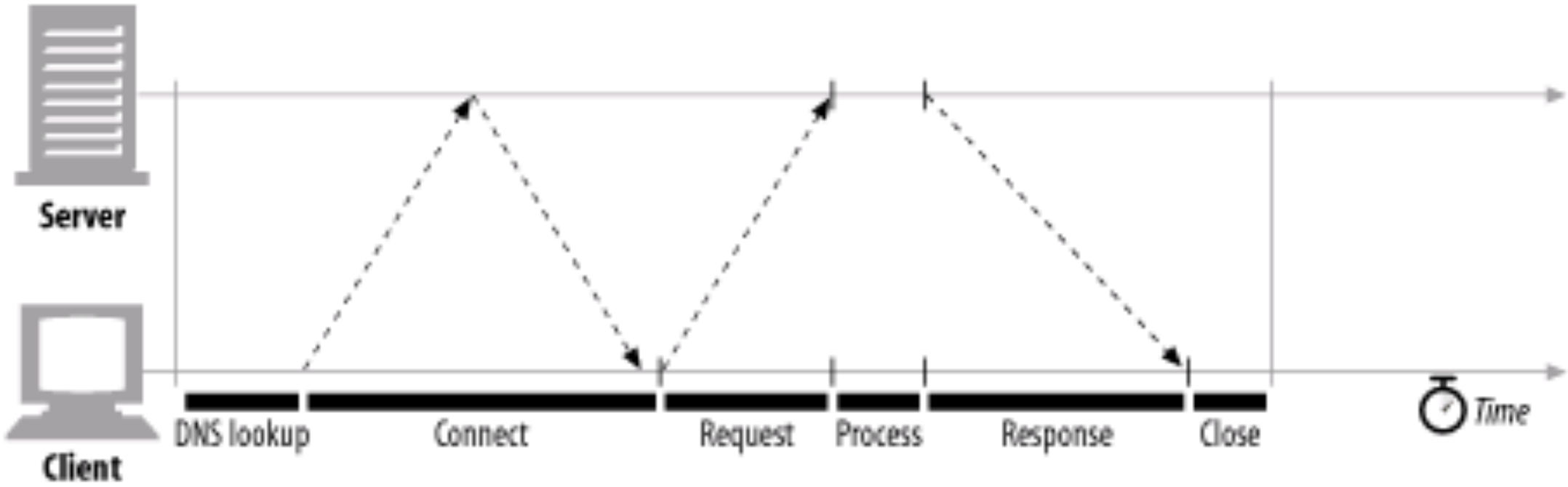
- HTTP has a simple structure:
  - client sends a request
  - server returns a reply.

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

How HTTP clients and servers communicate using the TCP sockets interface



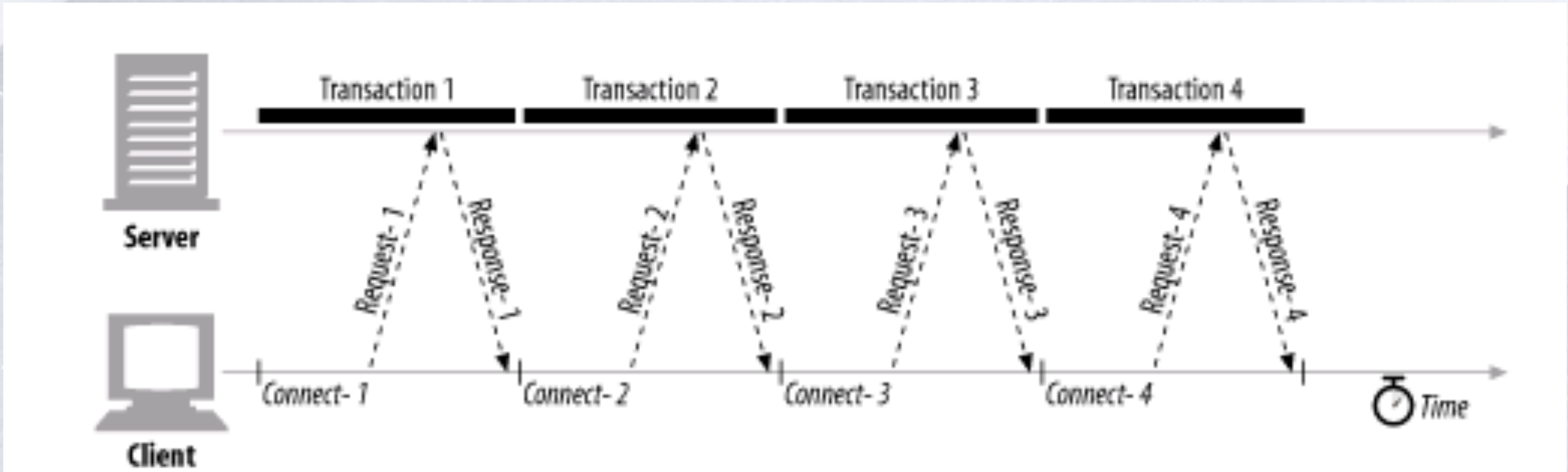
HTTP Transaction Delay



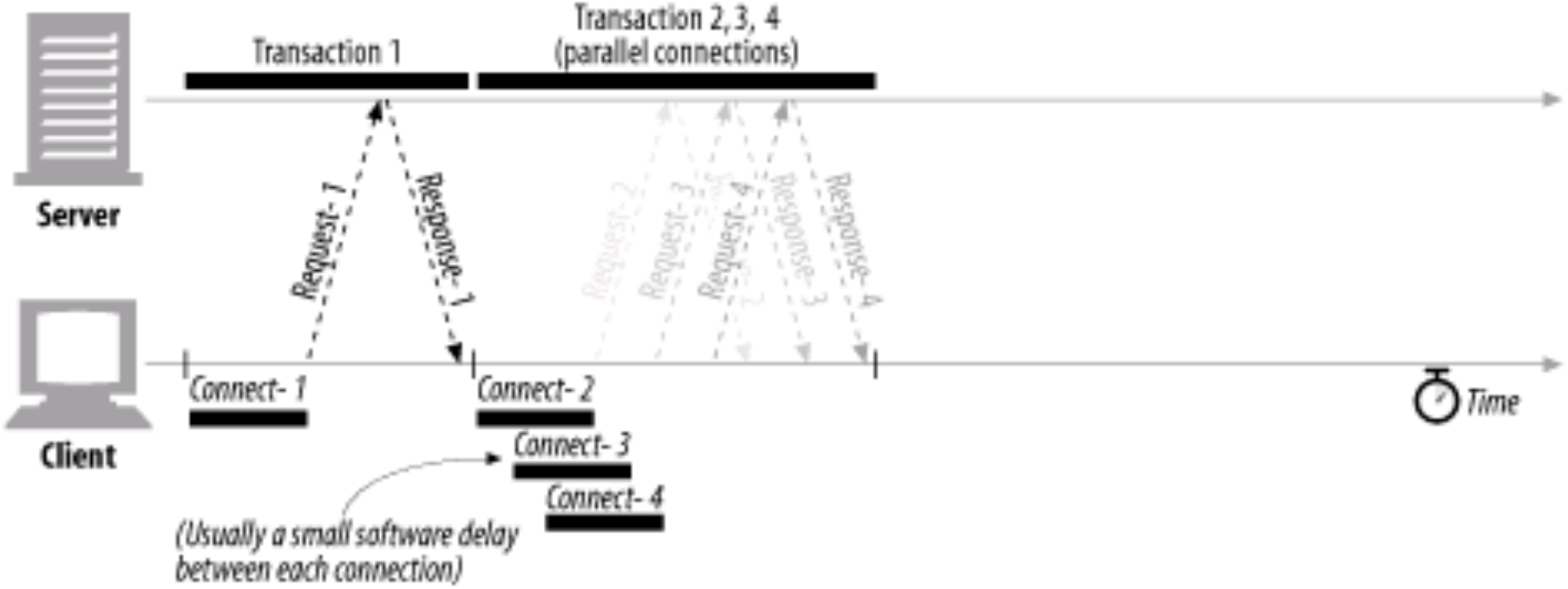
Common TCP-related delays affecting HTTP

- 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

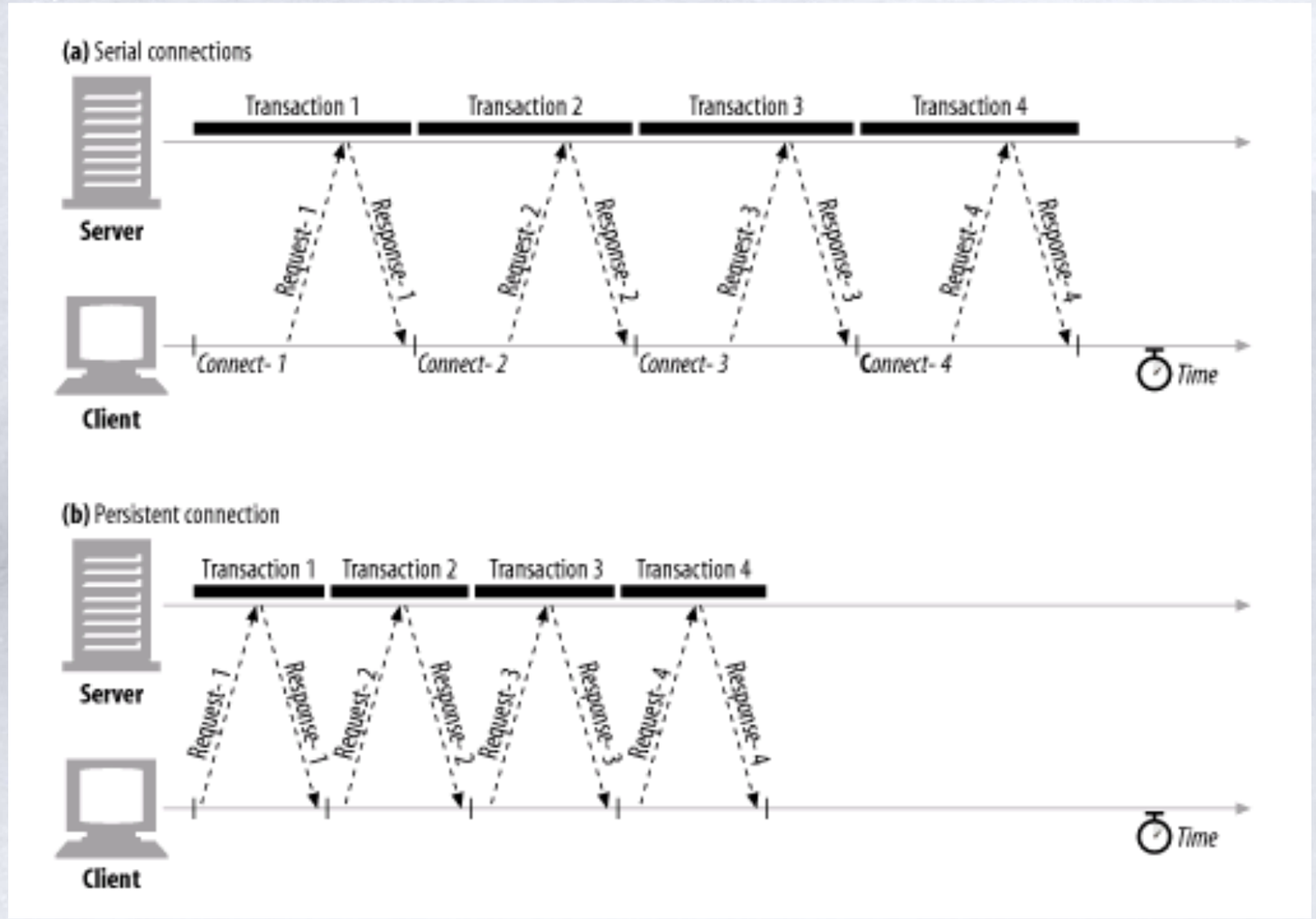


HTTP connections

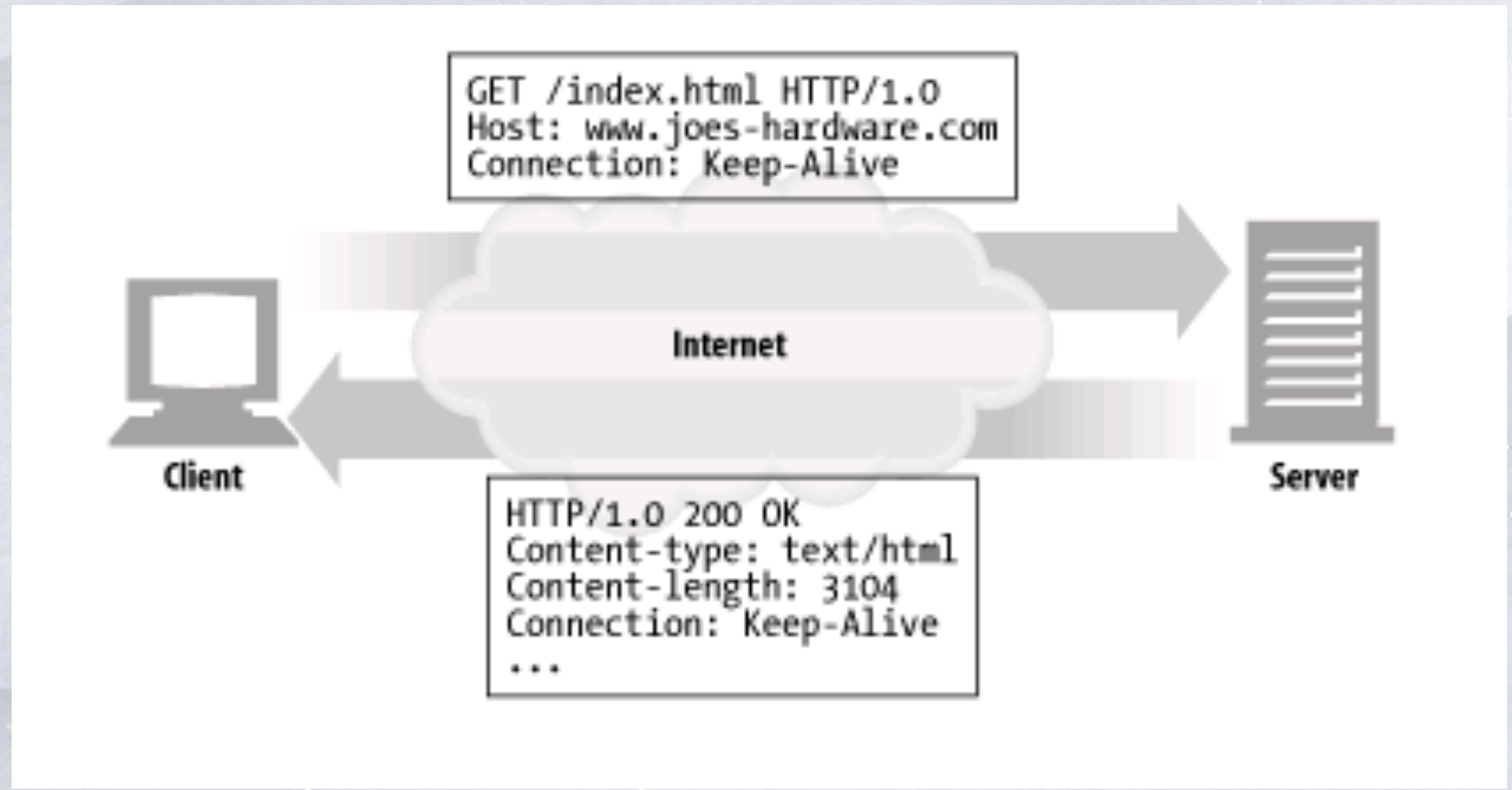
Nonpersistent HTTP	Persistent HTTP
At most one object is sent over a TCP connection.	Multiple objects can be sent over single TCP connection between client and server.
HTTP/1.0 uses nonpersistent HTTP	HTTP/1.1 uses persistent connections in default mode

Persistent Connections

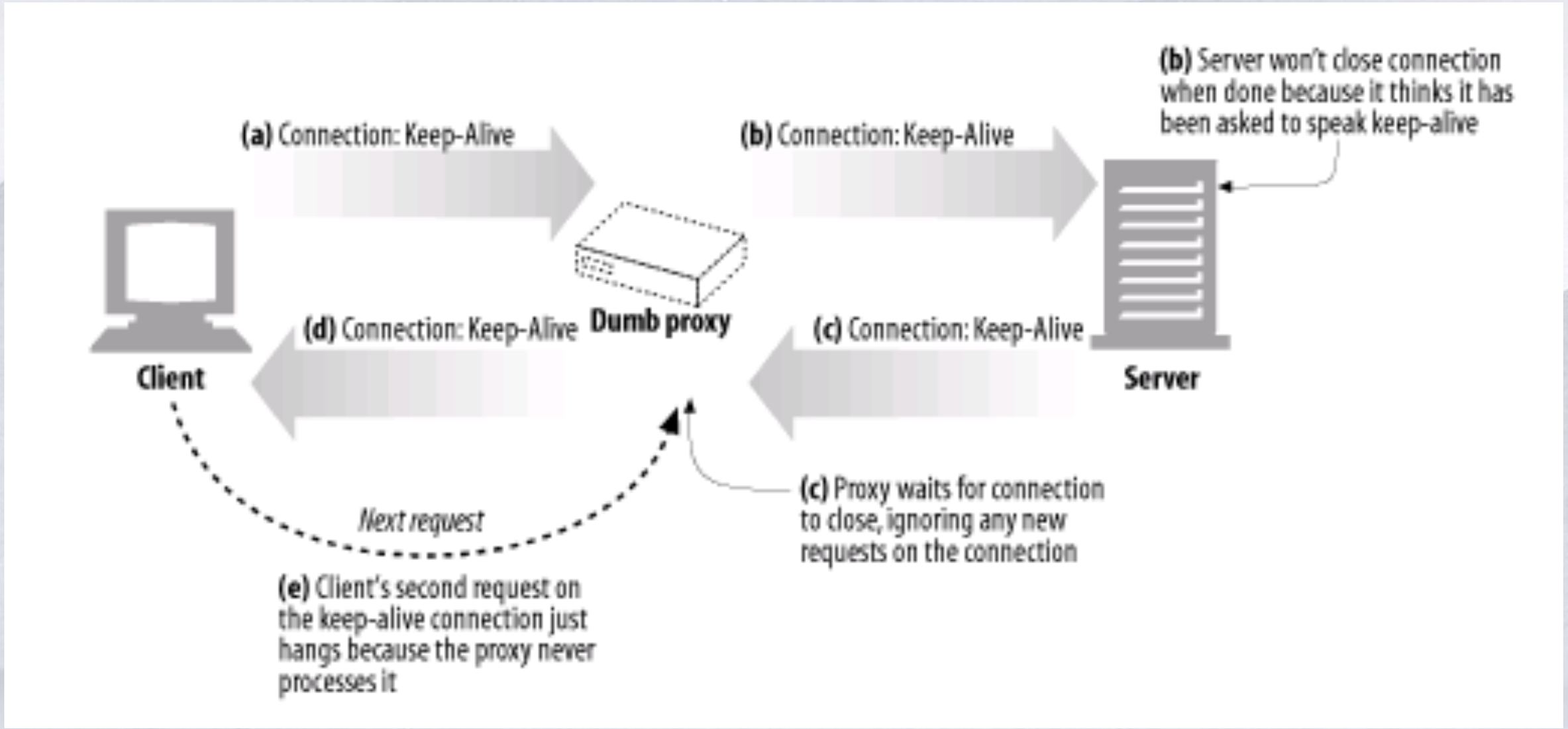
- TCP connections that are kept open after transactions complete are called persistent connections.



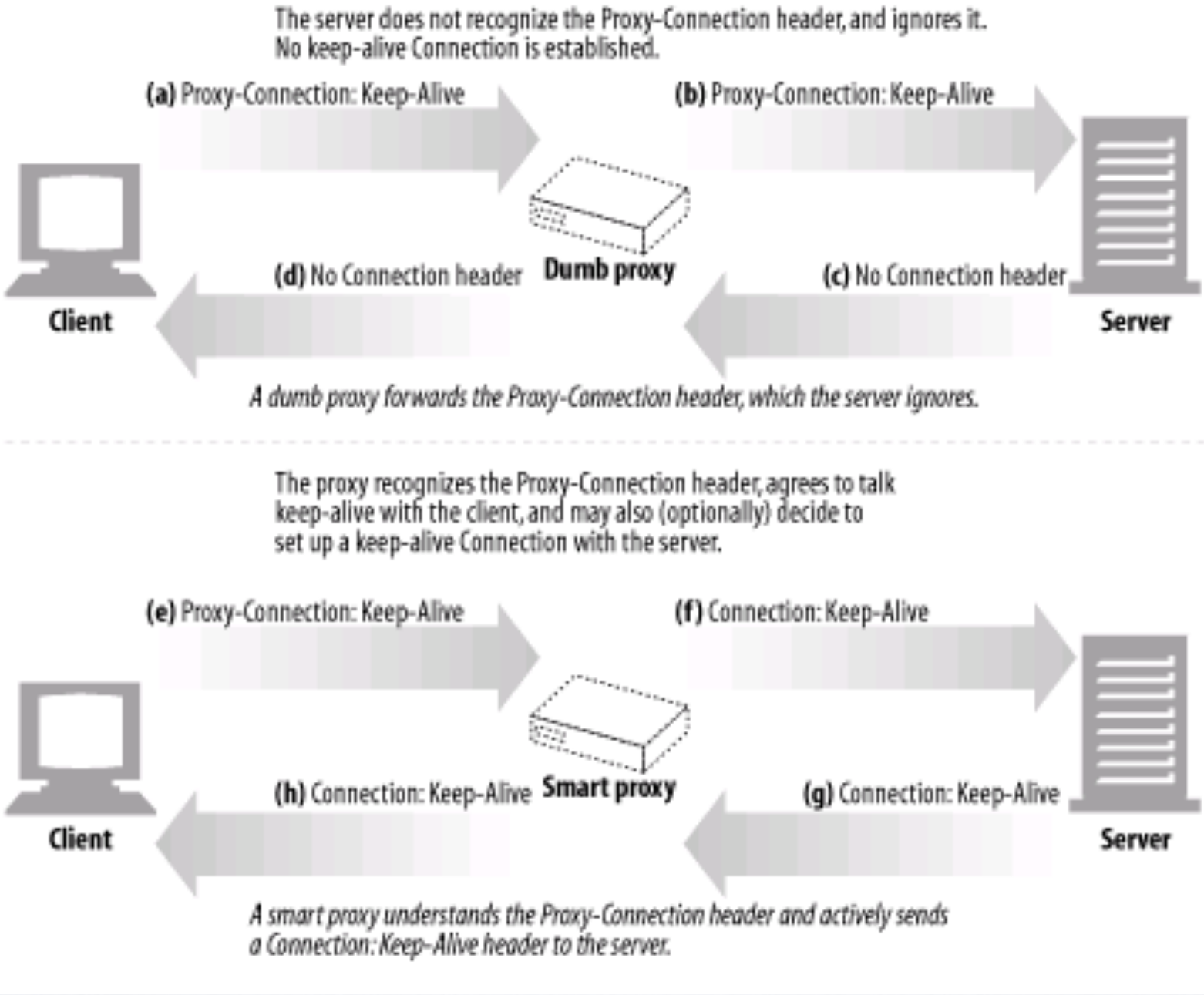
HTTP/1.0+ Keep-Alive Connections



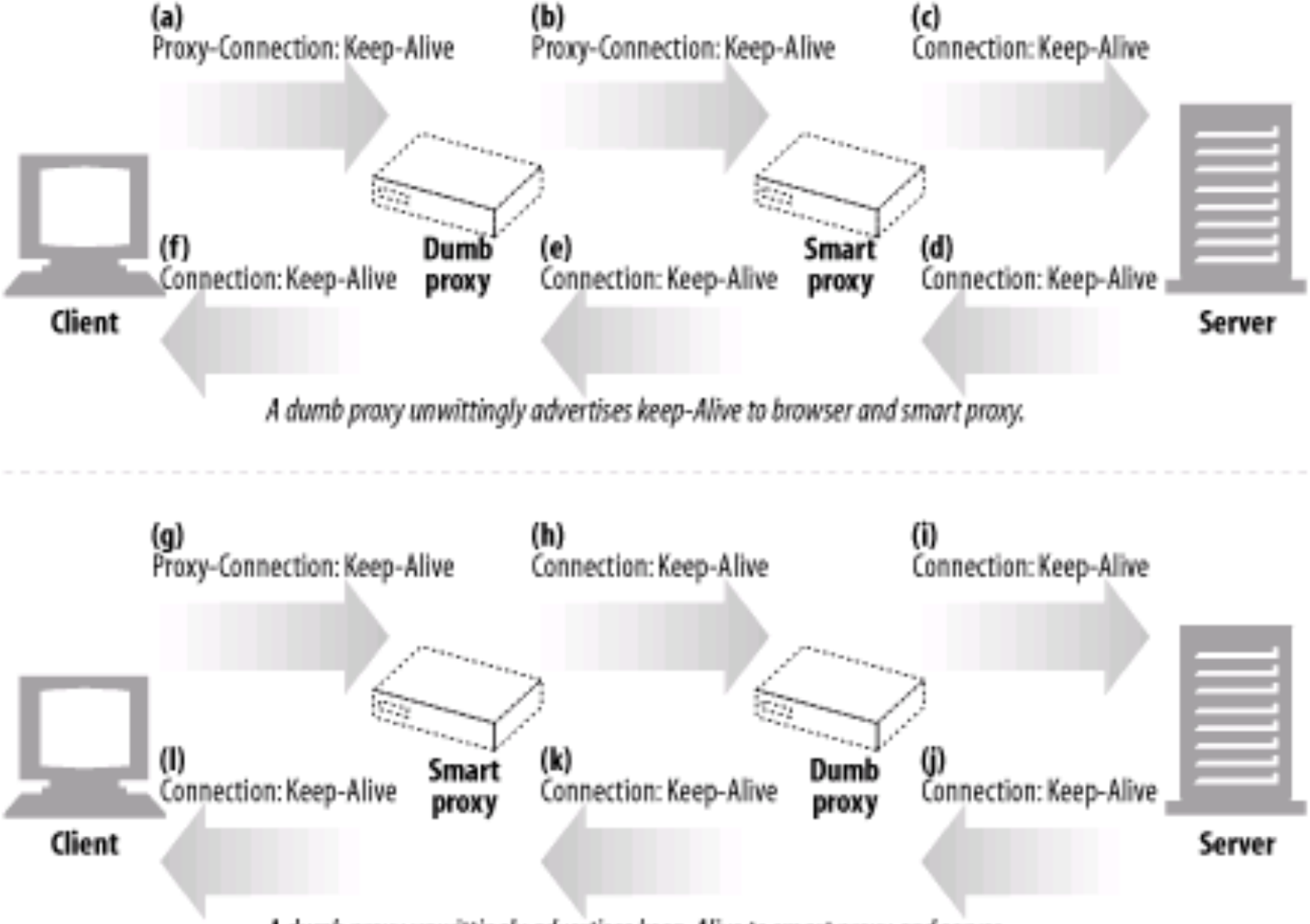
Keep-Alive and Dumb Proxy



Proxy-Connection Hack



Proxy-Connection's drawback



A dumb proxy unwittingly advertises keep-Alive to browser and smart proxy.

A dumb proxy unwittingly advertises keep-Alive to smart proxy and server.

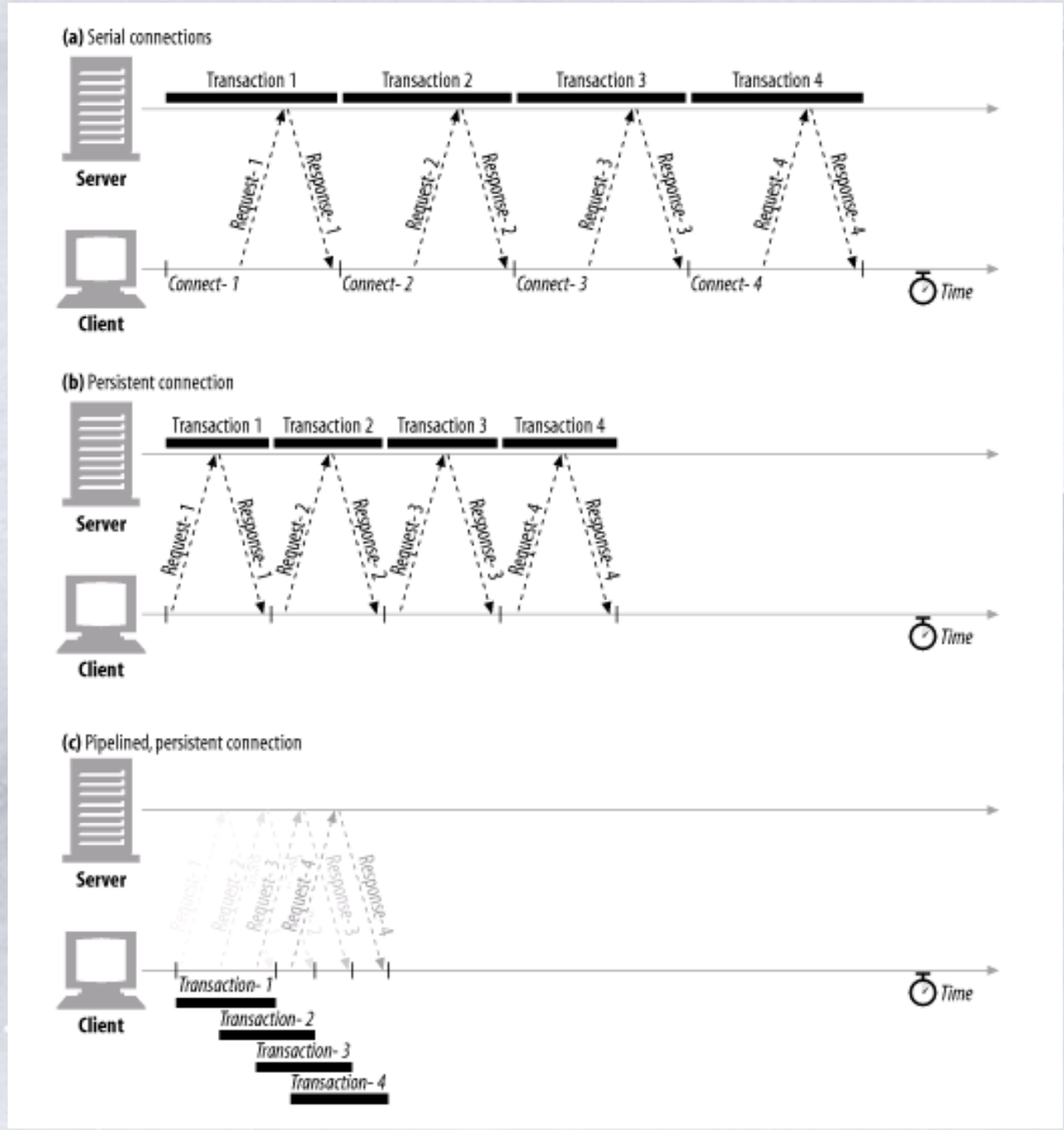
HTTP/1.1 Persistent Connections

- 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

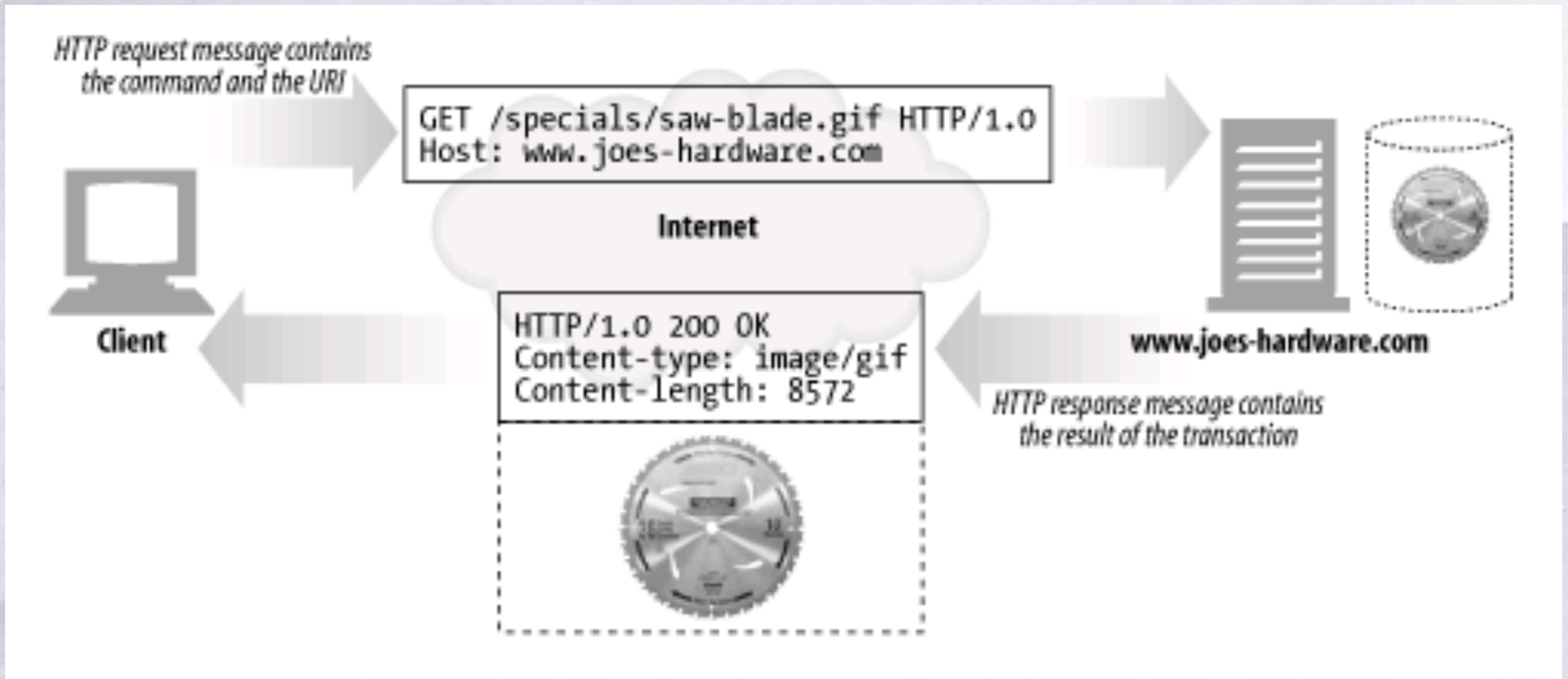
More on connection close

- 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.

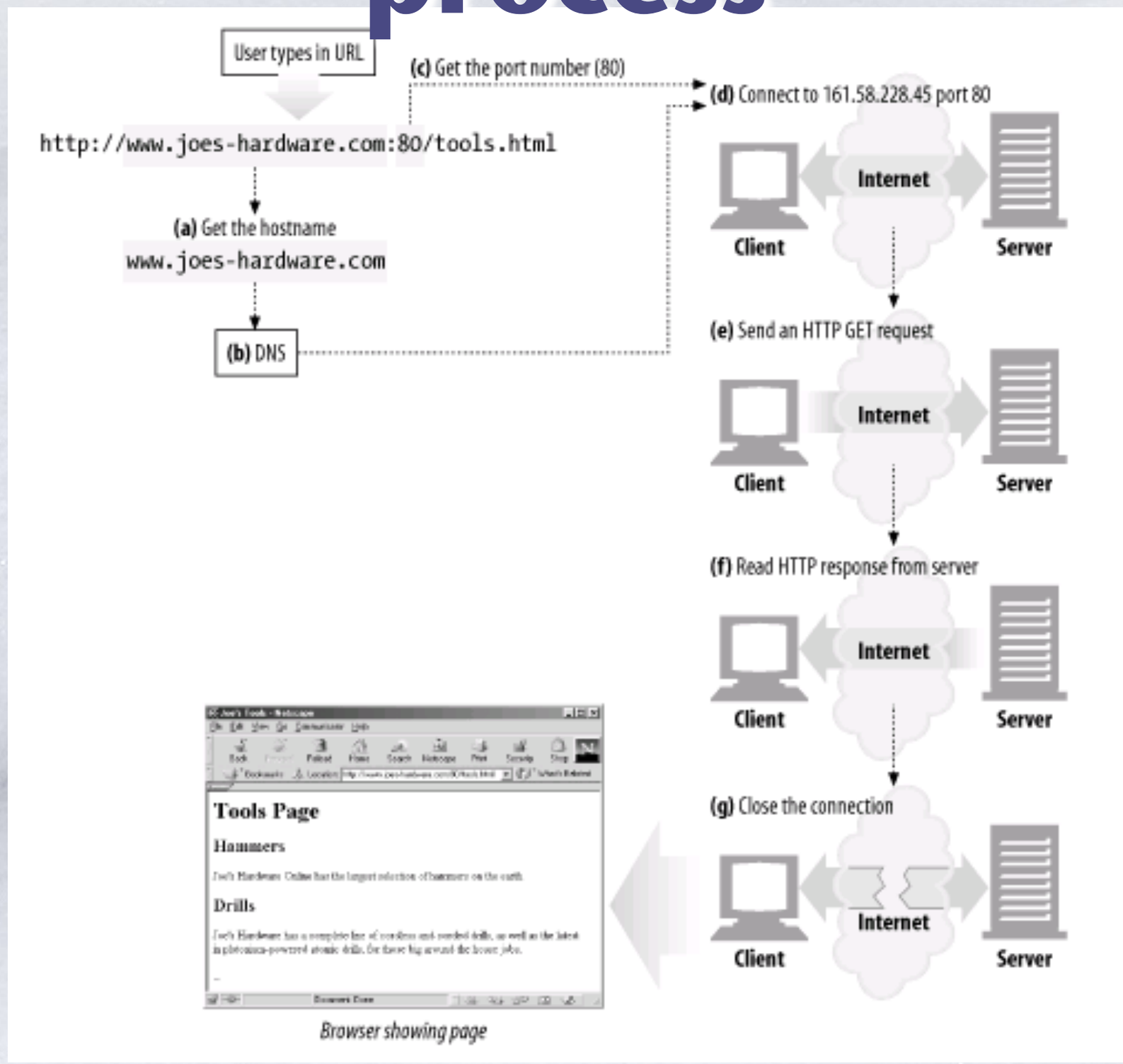
Graceful Connection Close

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

HTTP transaction



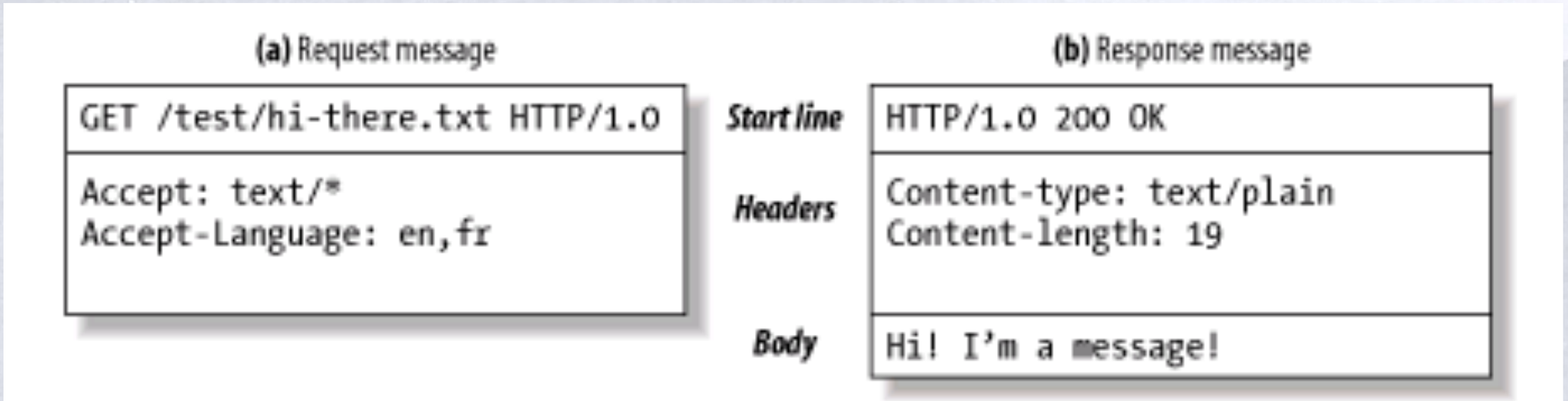
Basic browser connection process



Browser showing page

HTTP Messages

- two types of HTTP messages: request, response



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)

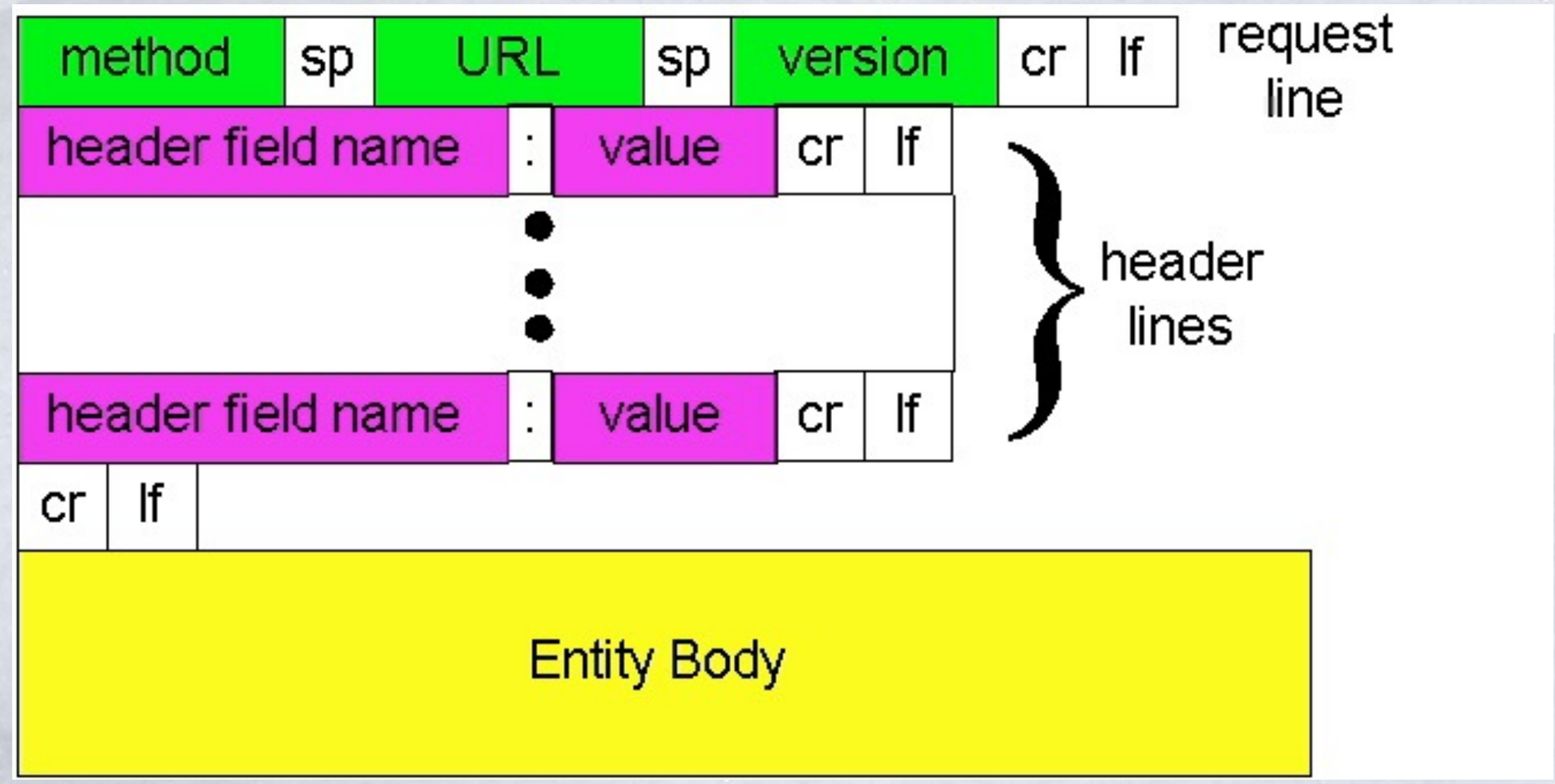
header lines

```
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

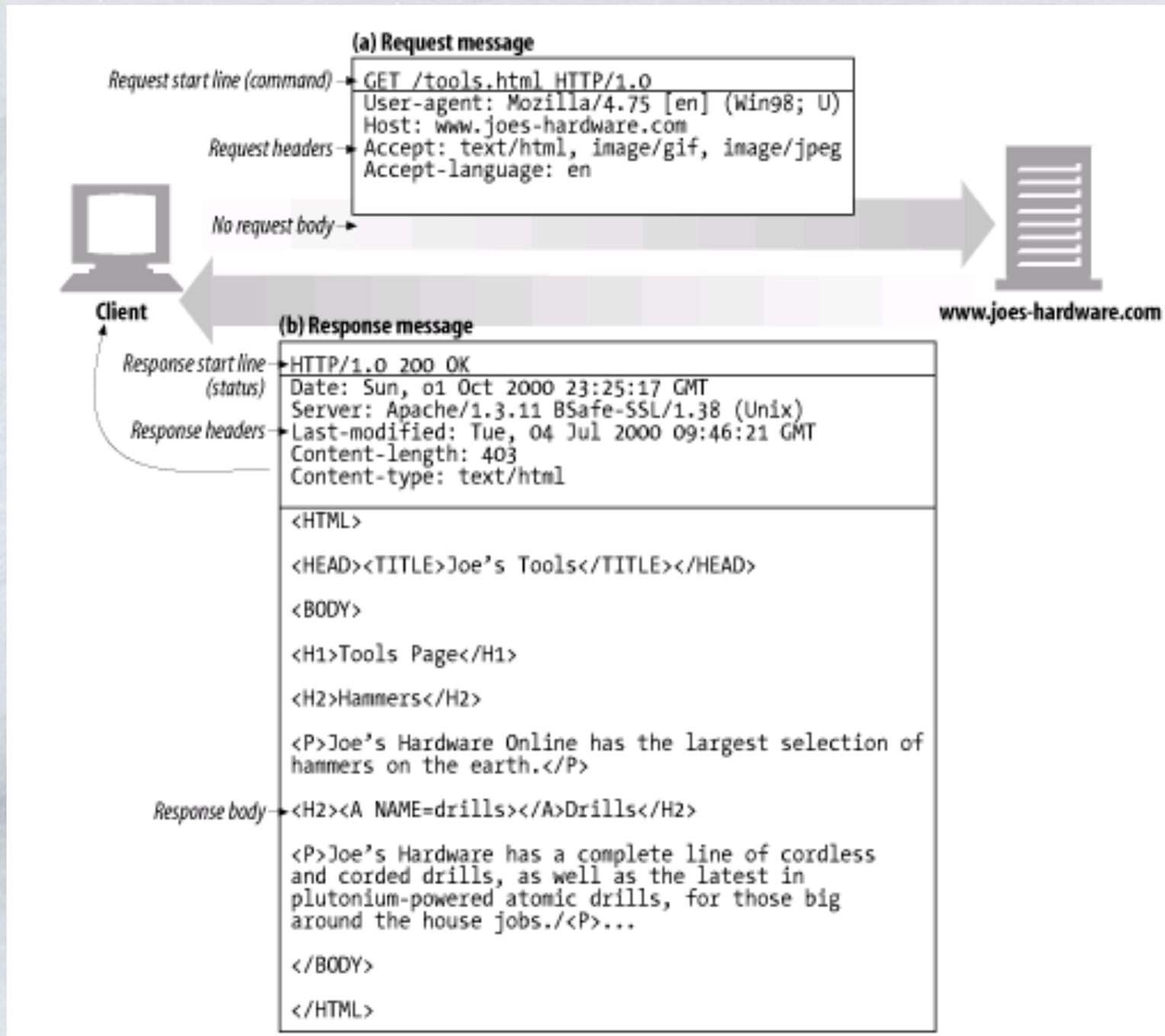
  OPTIONS

- future expansion is supported

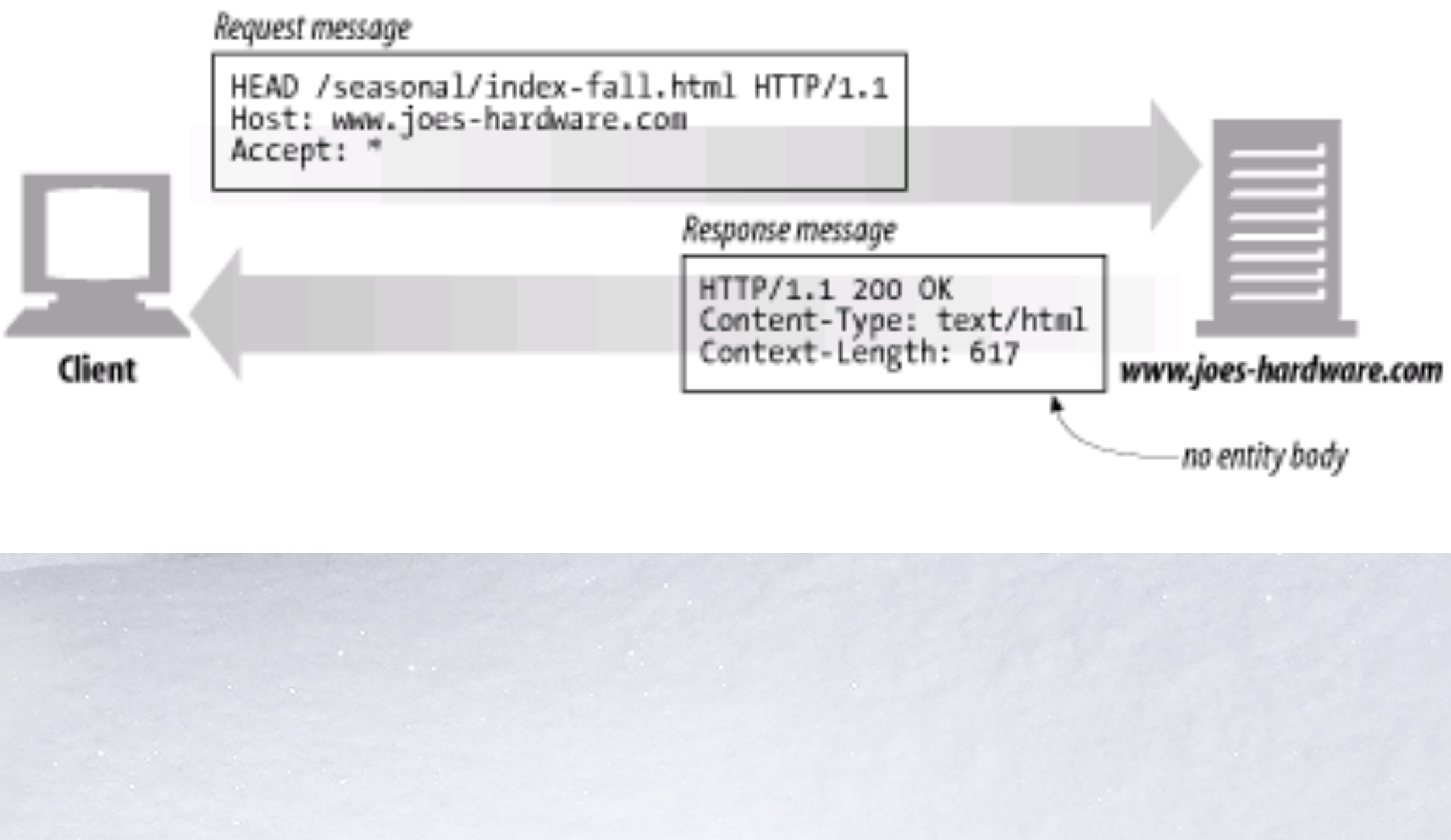
Methods

- 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

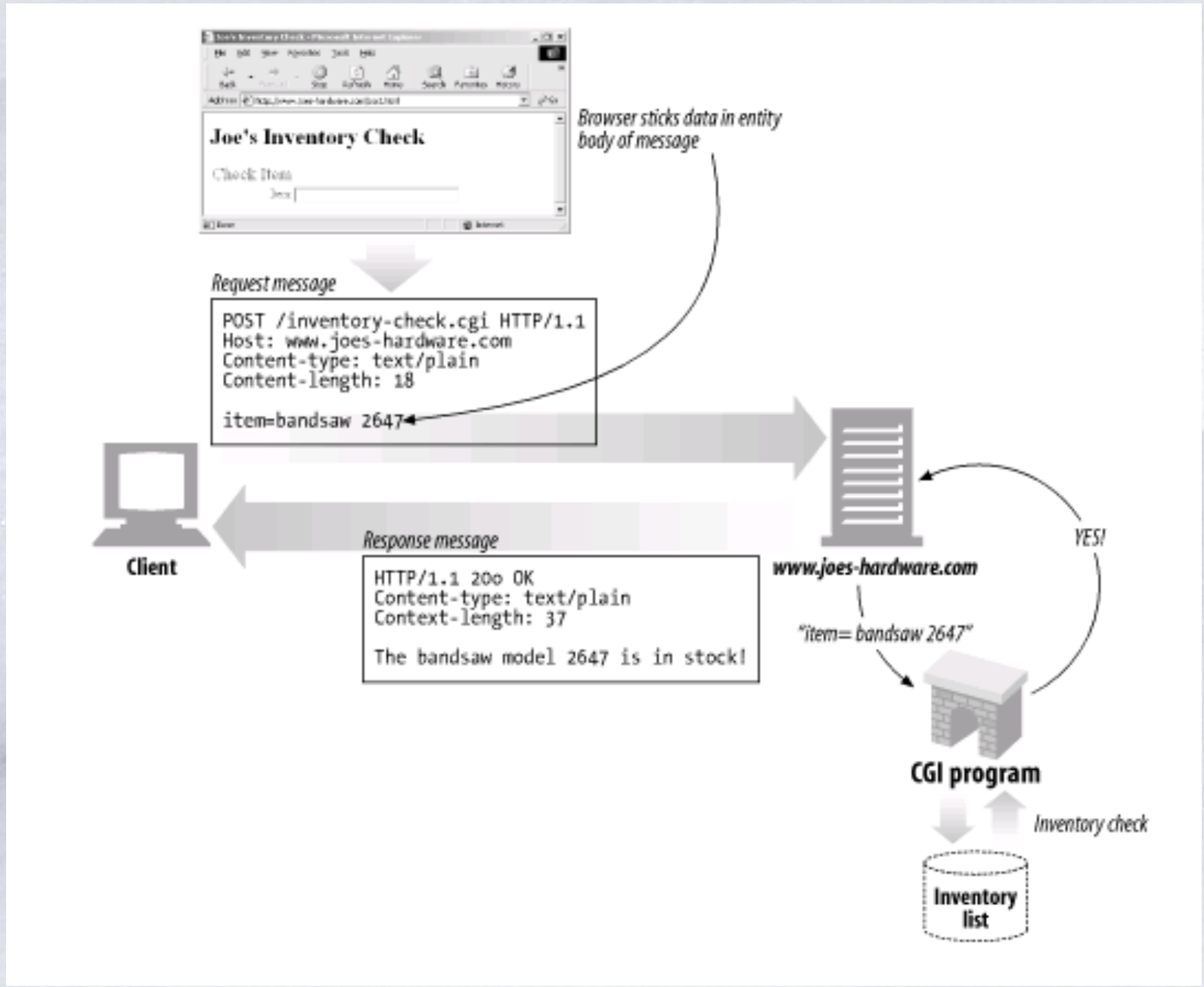
- A POST request includes some content (some data) after the headers (after the blank line).

- 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

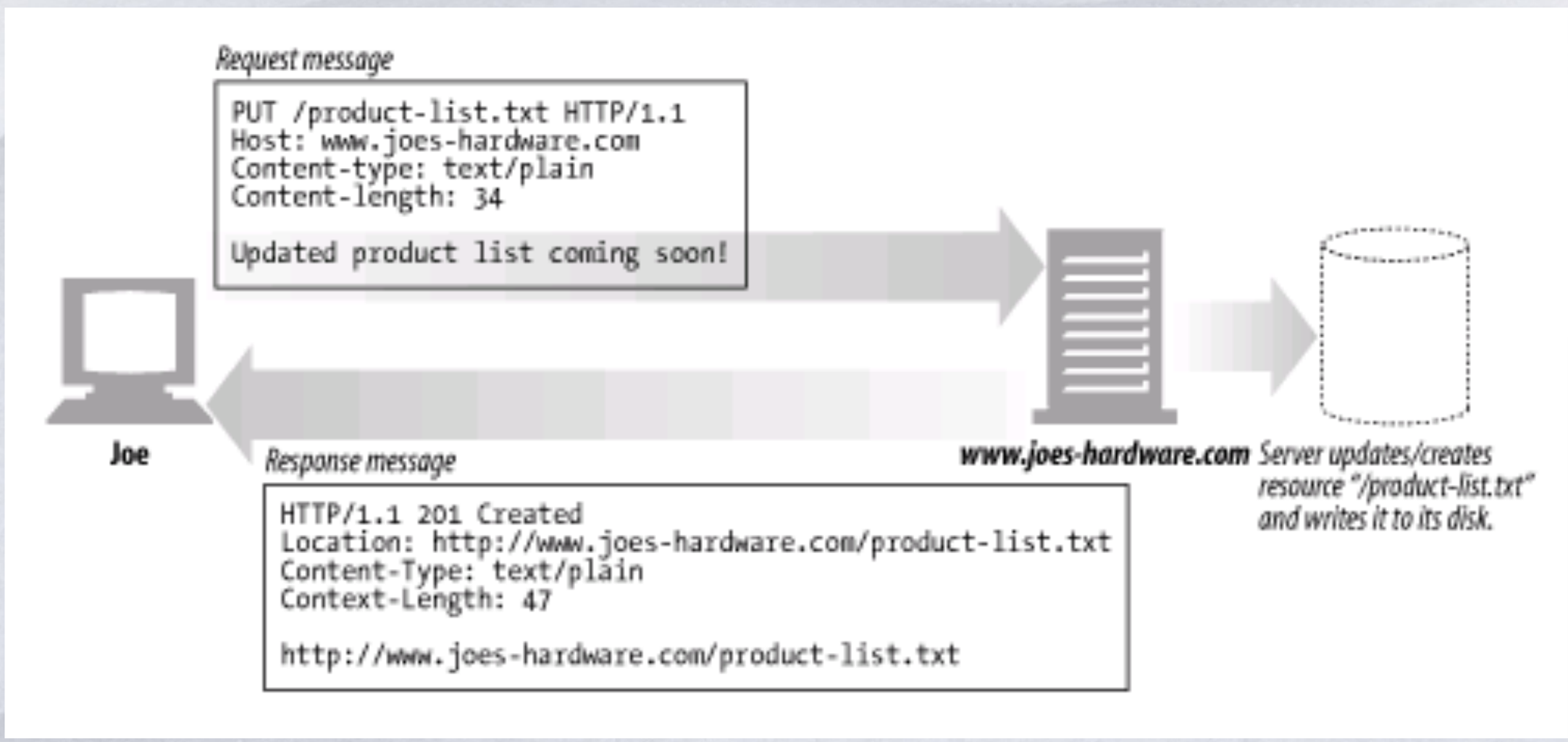
Example of POST



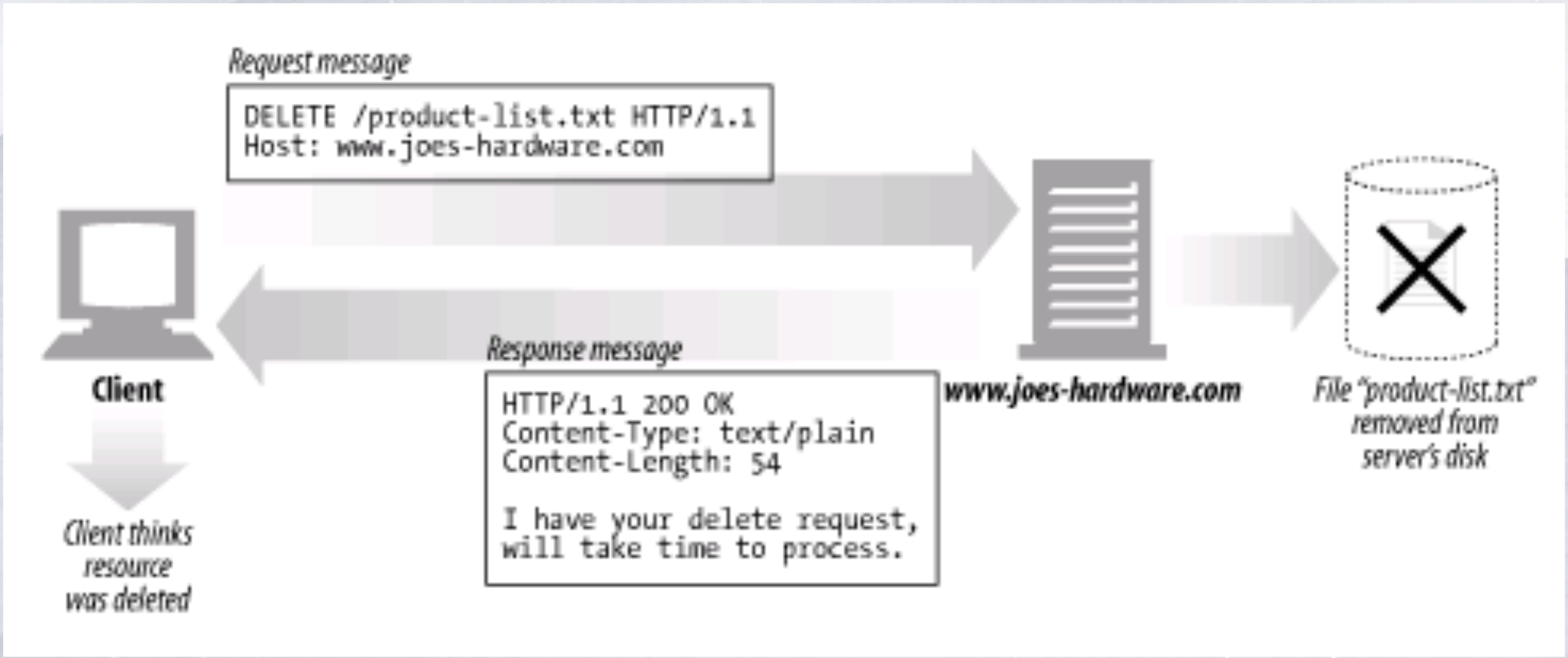
Methods (cont.)

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

Example of PUT



Example of DELETE

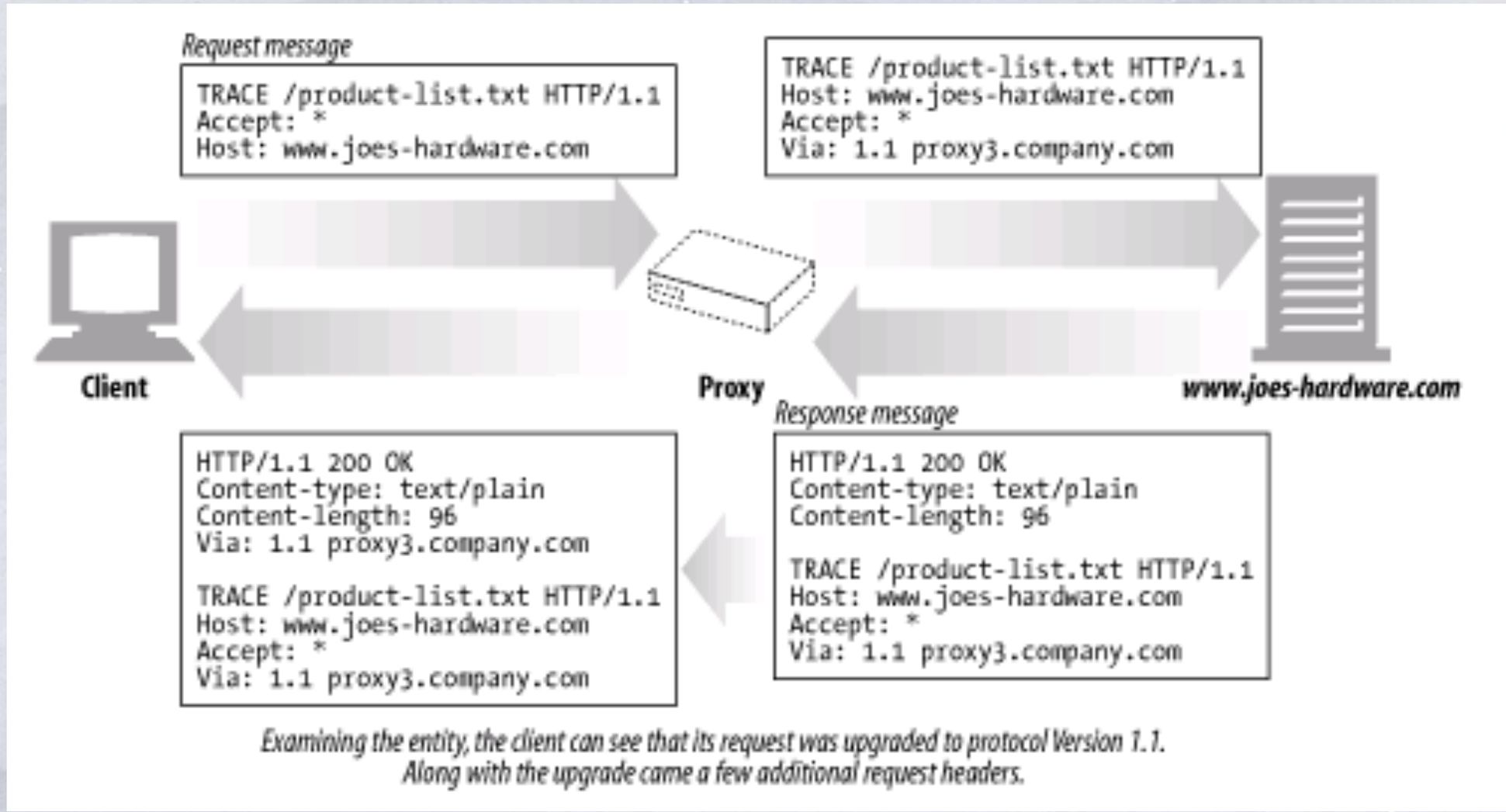


More Methods

- TRACE: used to trace HTTP forwarding through proxies, tunnels, etc.

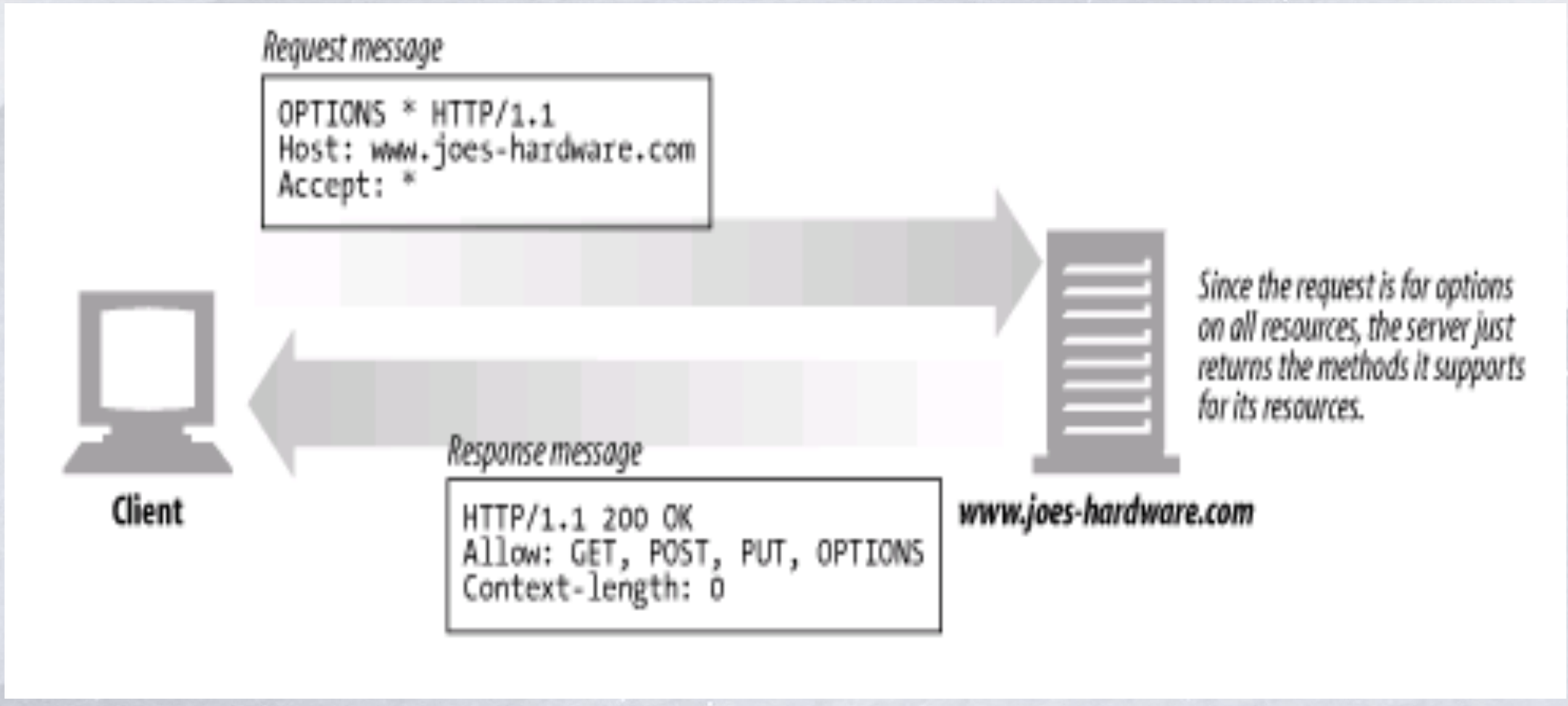
- OPTIONS: used to determine the capabilities of the server, or characteristics of a named resource.

Example of TRACE



Examining the entity, the client can see that its request was upgraded to protocol Version 1.1. Along with the upgrade came a few additional request headers.

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

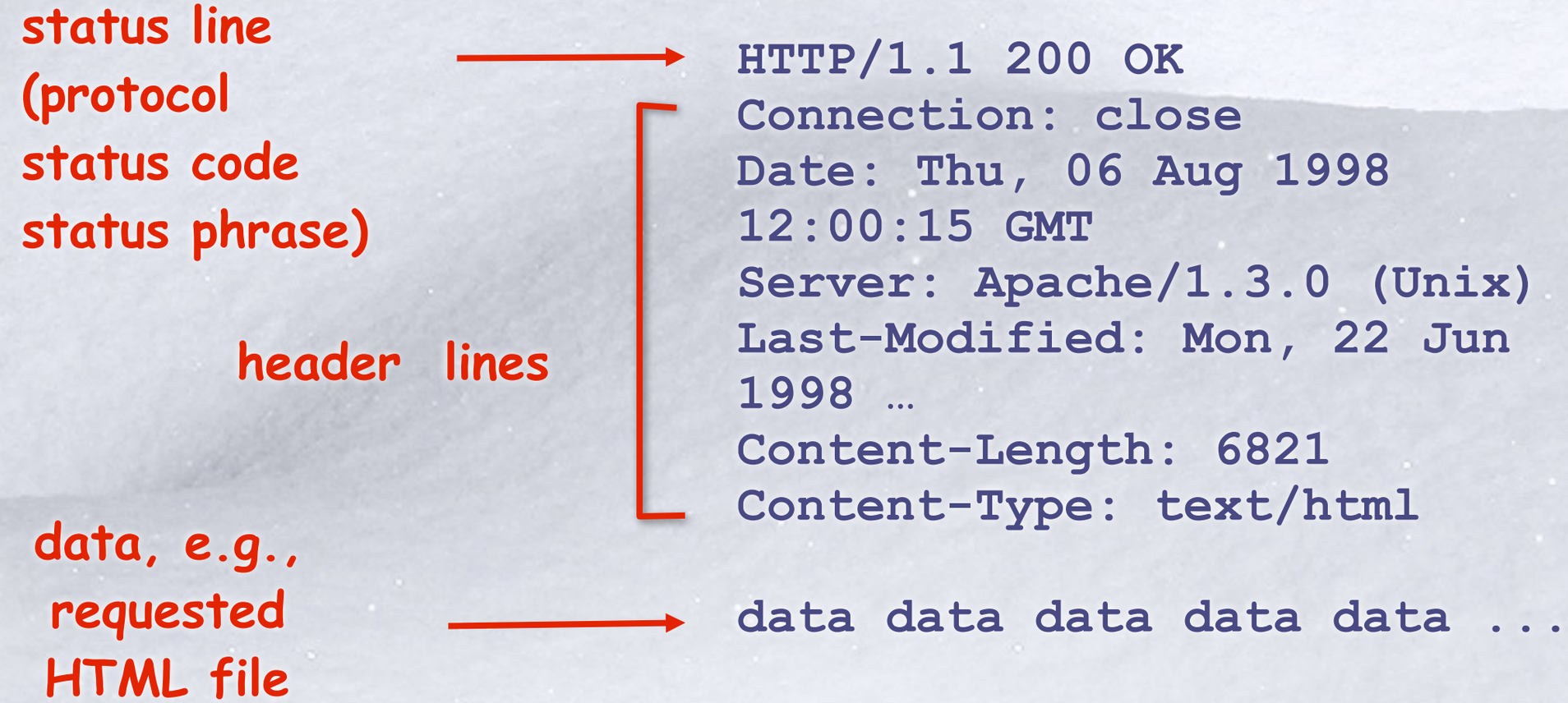
Header	Description
Max-Forwards	The maximum number of times a request should be forwarded to another proxy or gateway on its way to the origin server—used with the TRACE method
Proxy-Authorization	Same as Authorization, but used when authenticating with a proxy
Proxy-Connection	Same as Connection, but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

End of the Headers

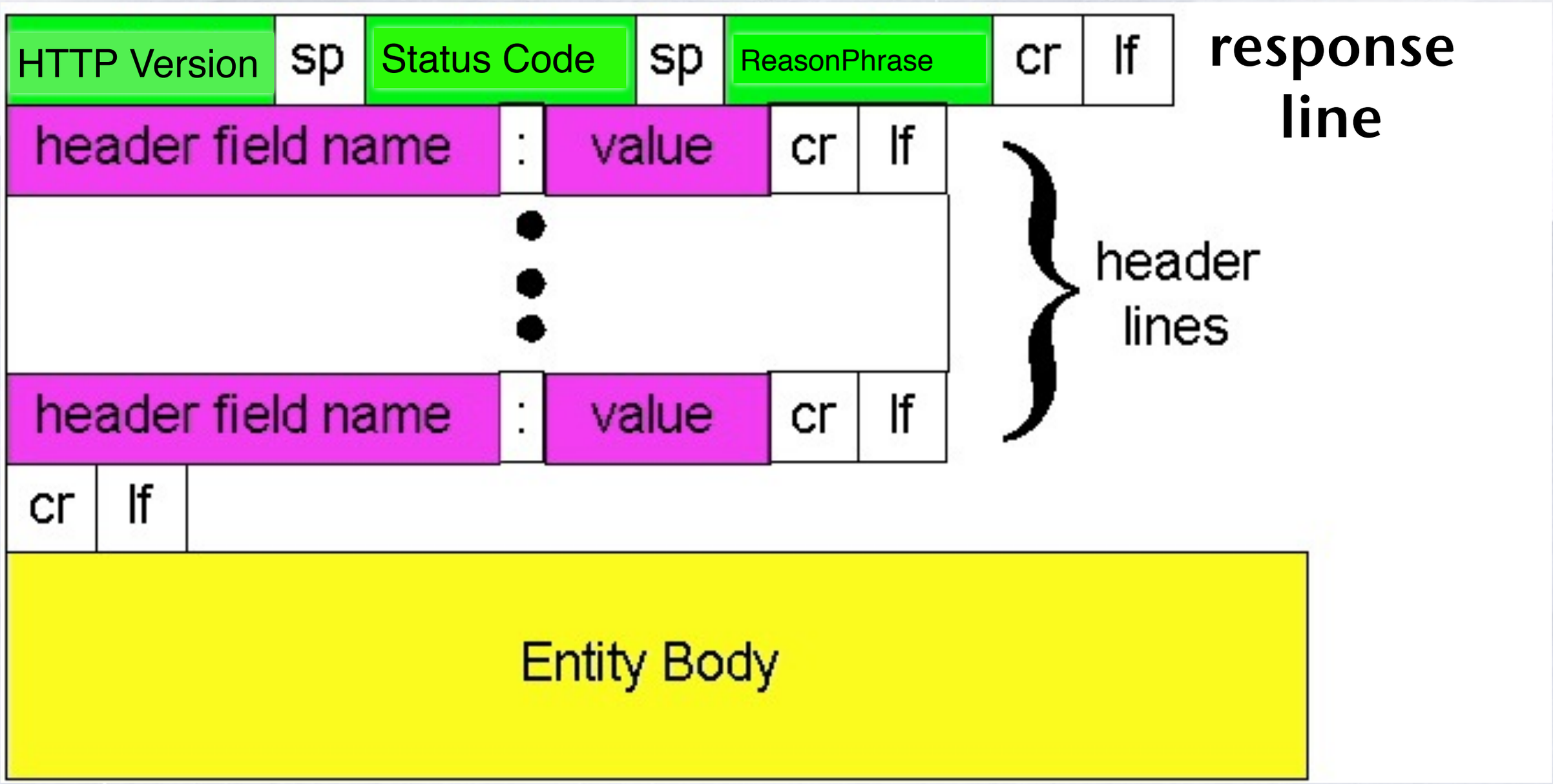
- Each header ends with a CRLF (\r\n)
- The end of the header section is marked with a blank line.
  - just CRLF
- For GET and HEAD requests, the end of the headers is the end of the request!

HTTP response message

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



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

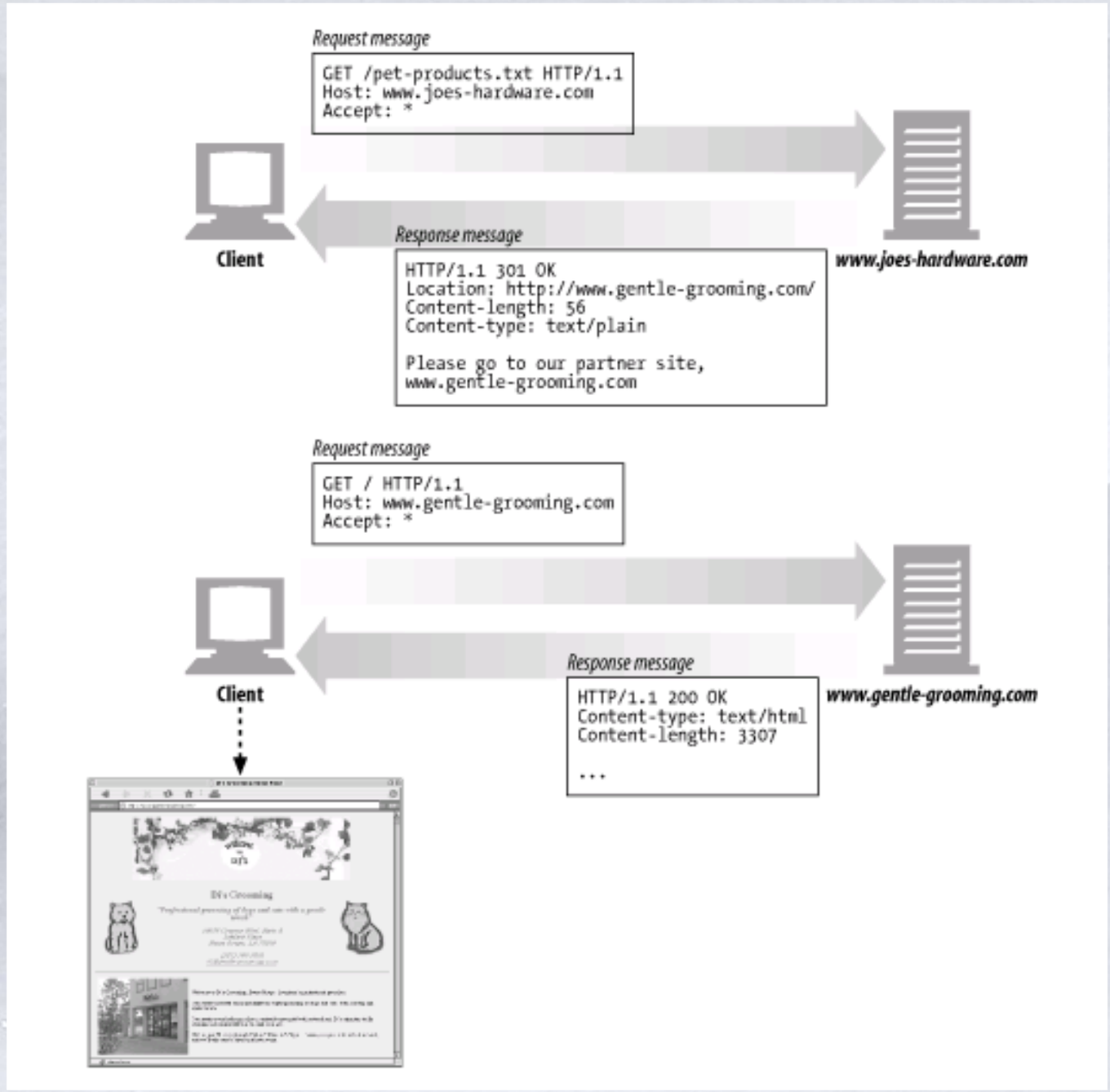
303 See Other

304 Not Modified

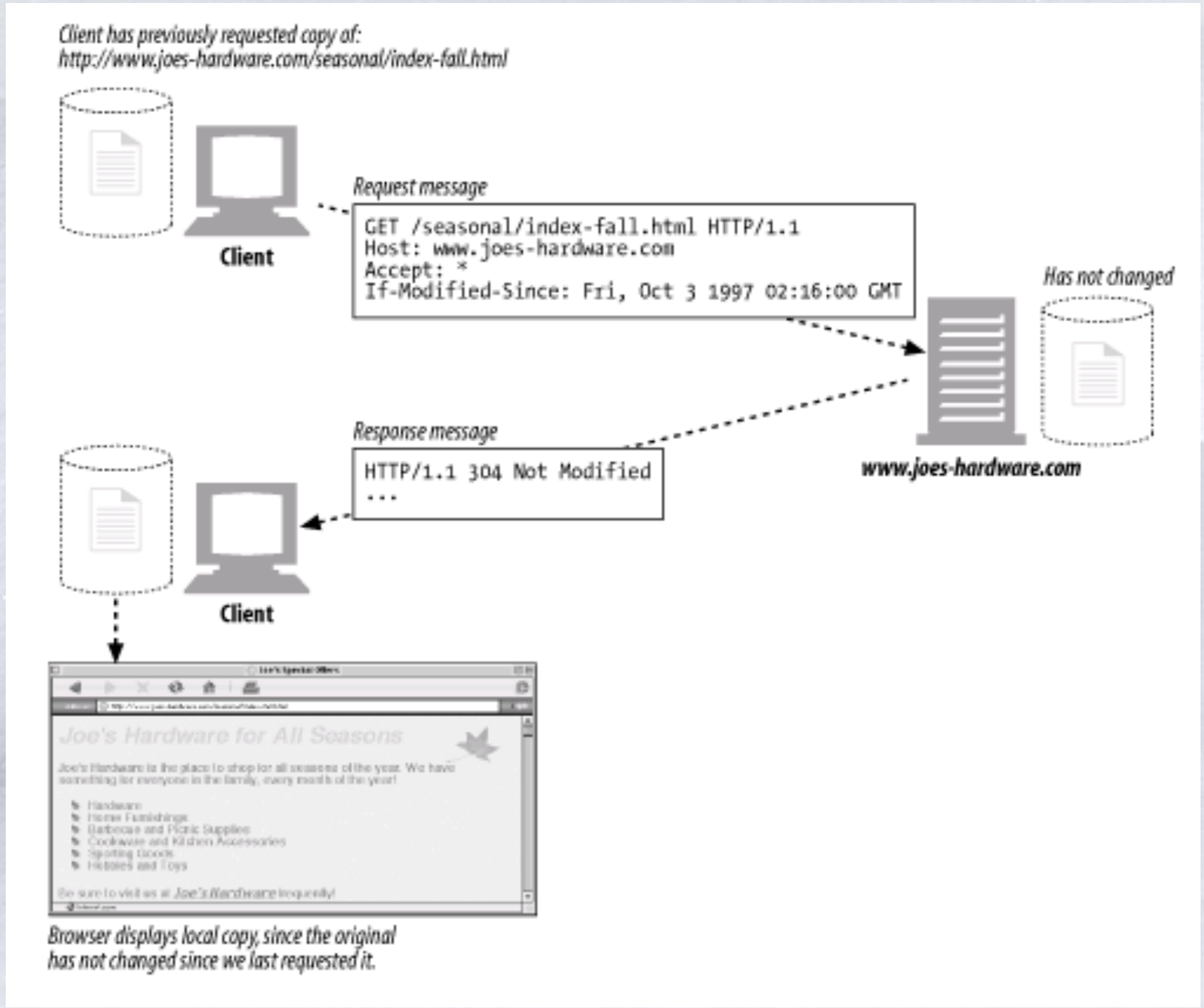
305 Use Proxy

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 used 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

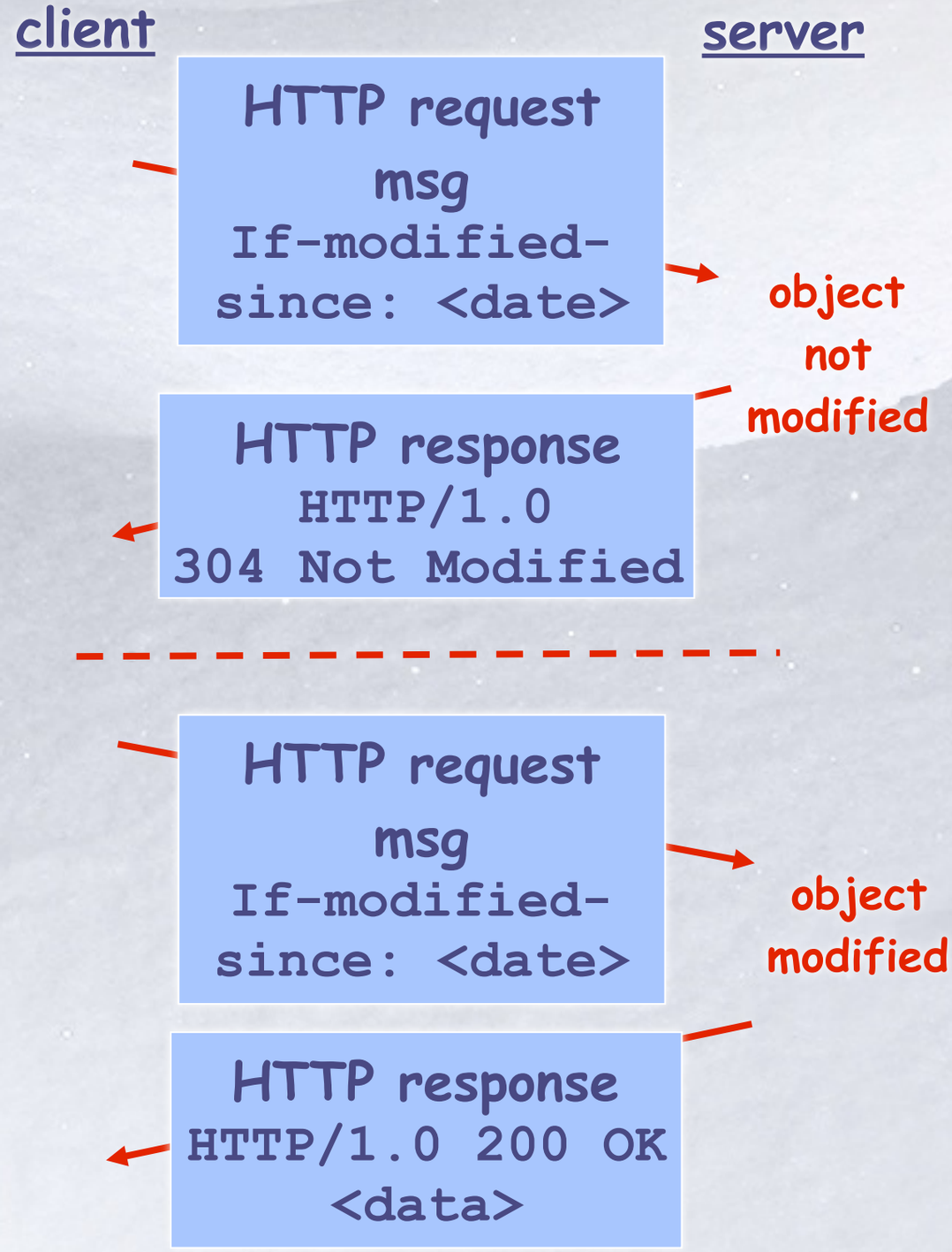
Header	Description
ETag	The entity tag associated with this entity
Expires	The date and time at which this entity will no longer be valid and will need to be fetched from the original source
Last-Modified	The last date and time when this entity changed

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
...
```

```
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 ...
```

```
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/1.1 304 Not Modified
Date: Thu, 8 Apr 2000 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
 (empty entity body)
```

Try it with telnet

> telnet www.xxx.edu.cn 80

GET / HTTP/1.0

Request-line

Blank Line
(end of headers)

HTTP/1.0 200 OK

Server: Apache

...

Response

Try it with telnet 1.1

> telnet www.xxx.edu.cn 80

GET / HTTP/1.1

Host: www.xxx.edu.cn

Required!

HTTP/1.0 200 OK

Server: Apache

...

Streams, Messages, and Frames

- **Stream**
  - A bidirectional flow of bytes within an established connection.
- **Message**
  - A complete sequence of frames that map to a logical message.
- **Frame**
  - The smallest unit of communication in HTTP 2.0, each containing a frame header, which at minimum identifies the stream to which the frame belongs.

Binary Format

Start Line

Header

Body

Length

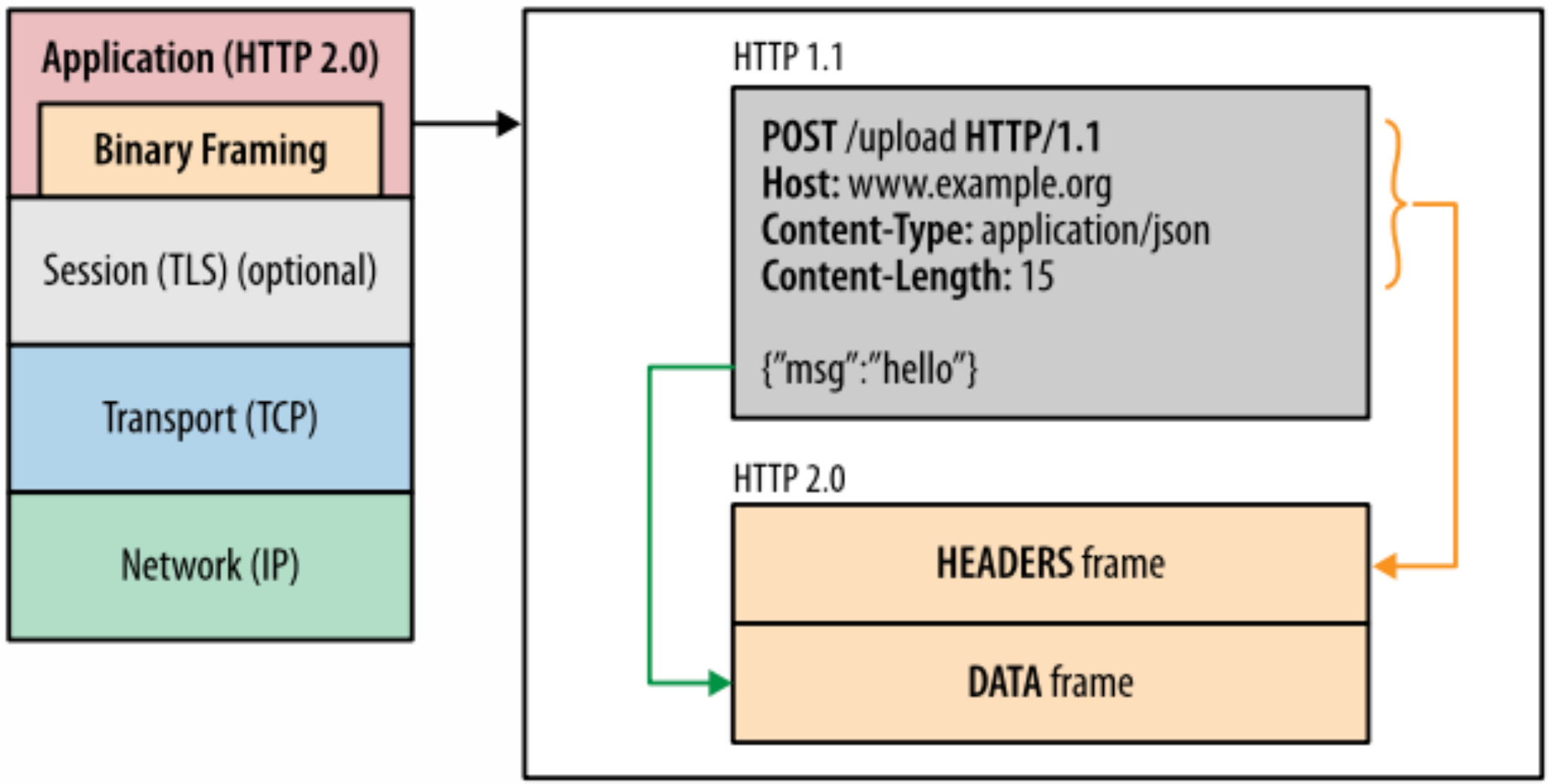
Type

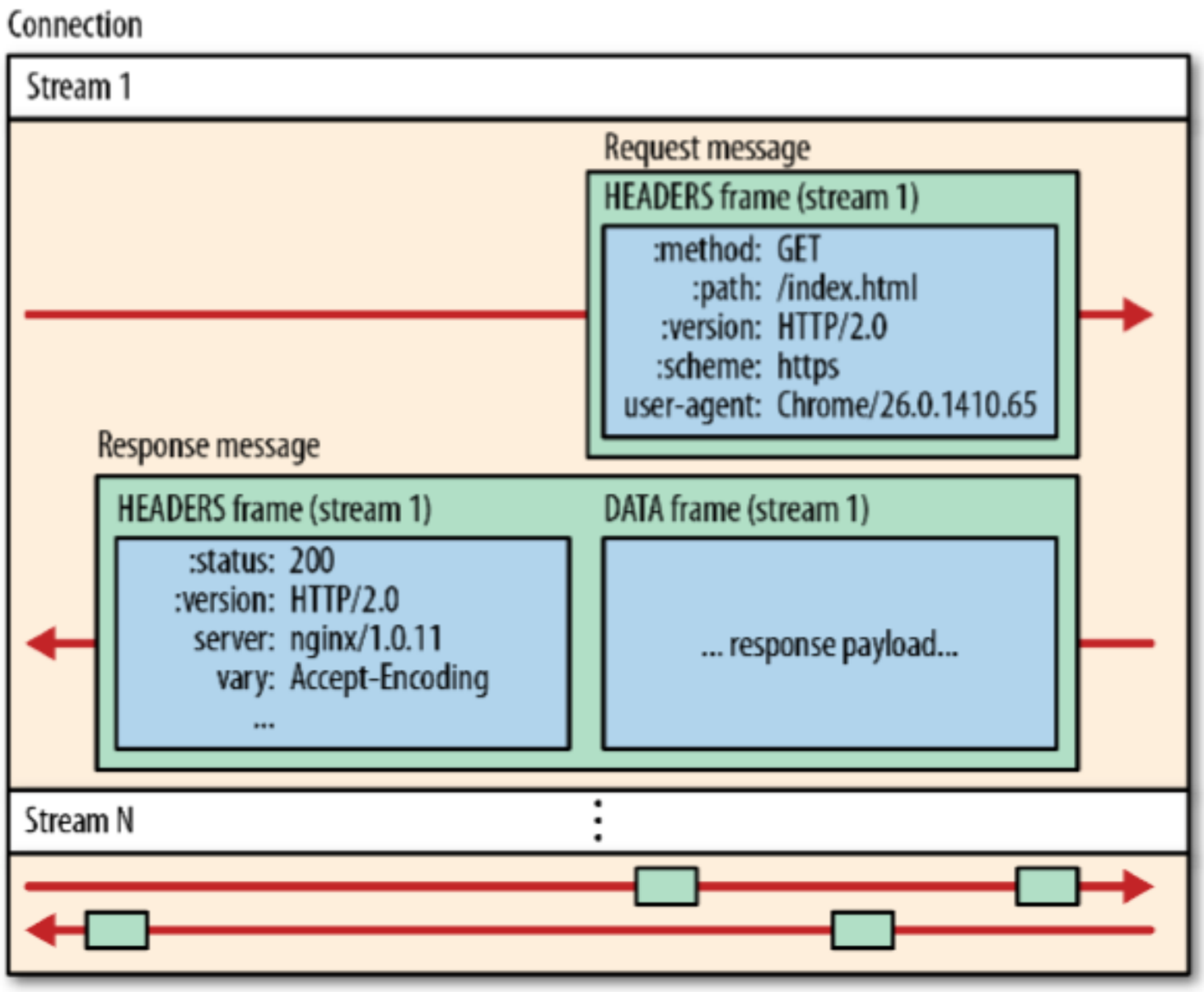
Flags

Stream ID

Payload

Binary Framing Layer

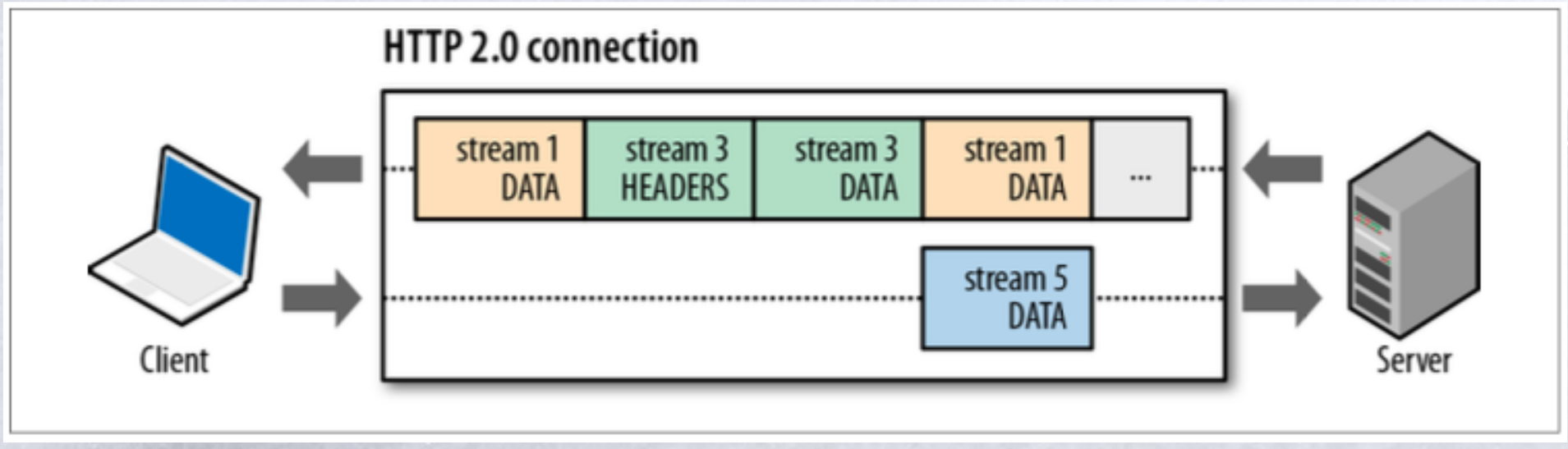


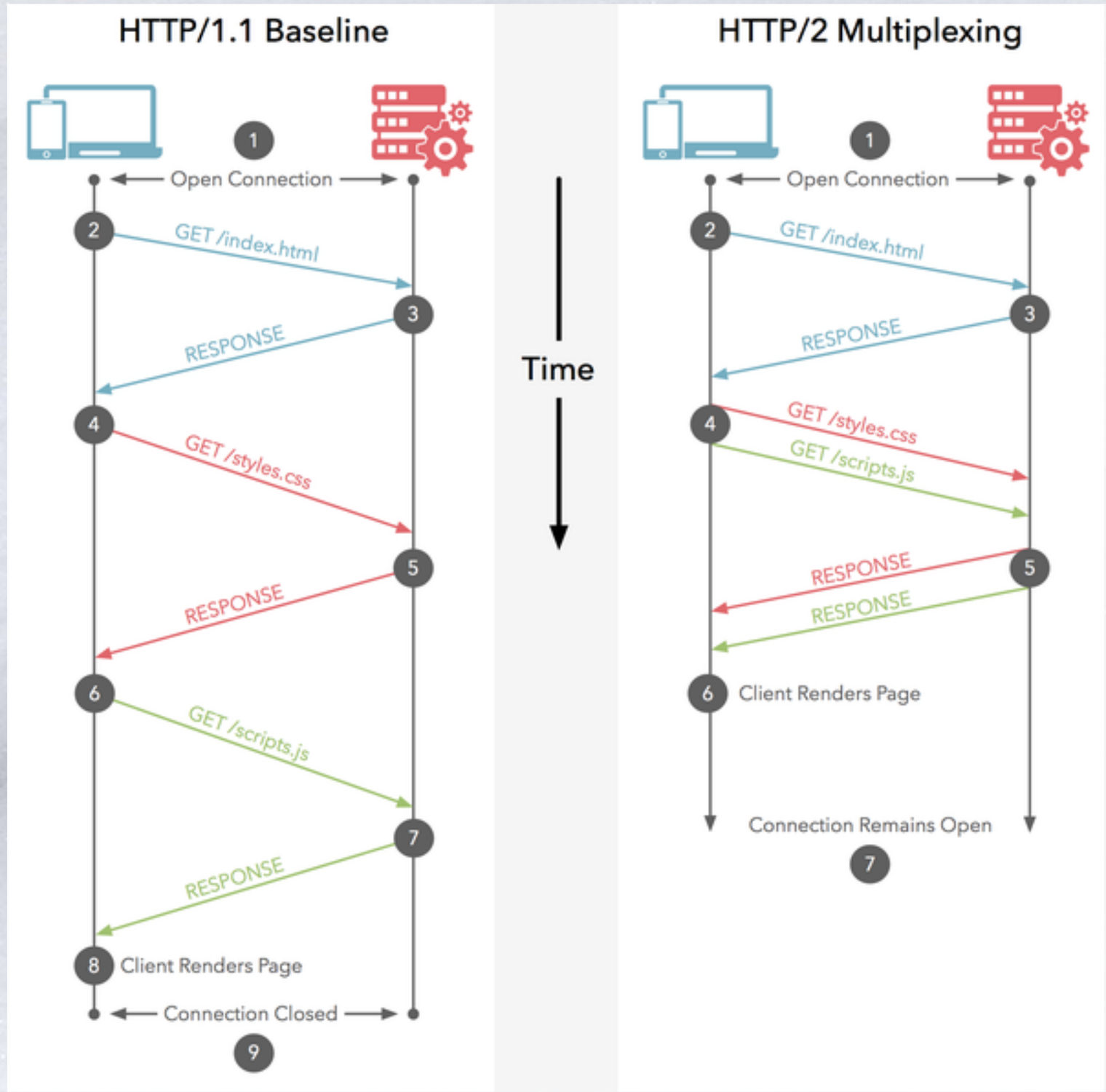

Connection
Stream 1
Request message
HEADERS frame (stream 1)
:method: GET
:path: /index.html
:version: HTTP/2.0
:scheme: https
user-agent: Chrome/26.0.1410.65
Response message
HEADERS frame (stream 1)
:status: 200
:version: HTTP/2.0
server: nginx/1.0.11
vary: Accept-Encoding
...
DATA frame (stream 1)
... response payload...
Stream N

Understanding HTTP 2.0

- All communication is performed with a single TCP connection.
- The stream is a virtual channel within a connection, which carries bidirectional messages. Each stream has a unique integer identifier (1, 2, ..., N).
- The message is a logical HTTP message, such as a request, or response, which consists of one or more frames.
- The frame is the smallest unit of communication, which carries a specific type of data—e.g., HTTP headers, payload, and so on.
  - The new binary framing layer in HTTP 2.0 resolves the head-of-line blocking problem found in HTTP 1.1 and eliminates the need for multiple connections to enable parallel processing and delivery of requests and responses. As a result, this makes our applications faster, simpler, and cheaper to deploy.

Request and Response Multiplexing




HTTP/1.1 Baseline
1
Open Connection
2
GET /index.html
3
RESPONSE
4
GET /styles.css
5
RESPONSE
6
GET /scripts.js
7
RESPONSE
8
Client Renders Page
Connection Closed
9
Time
HTTP/2 Multiplexing
1
Open Connection
2
GET /index.html
3
RESPONSE
4
GET /styles.css
GET /scripts.js
RESPONSE
RESPONSE
5
6
Client Renders Page
Connection Remains Open
7

Request Prioritization

- each stream can be assigned a 31-bit priority value:
  - 0 represents the highest priority stream.
  - $2^{31} - 1$ represents the lowest priority stream.
- HTTP 2.0 does not specify any specific algorithm for dealing with priorities, it just provides the mechanism by which the priority data can be exchanged between client and server.
- As such, priorities are hints, and the prioritization strategy can vary based on the implementation of client and server

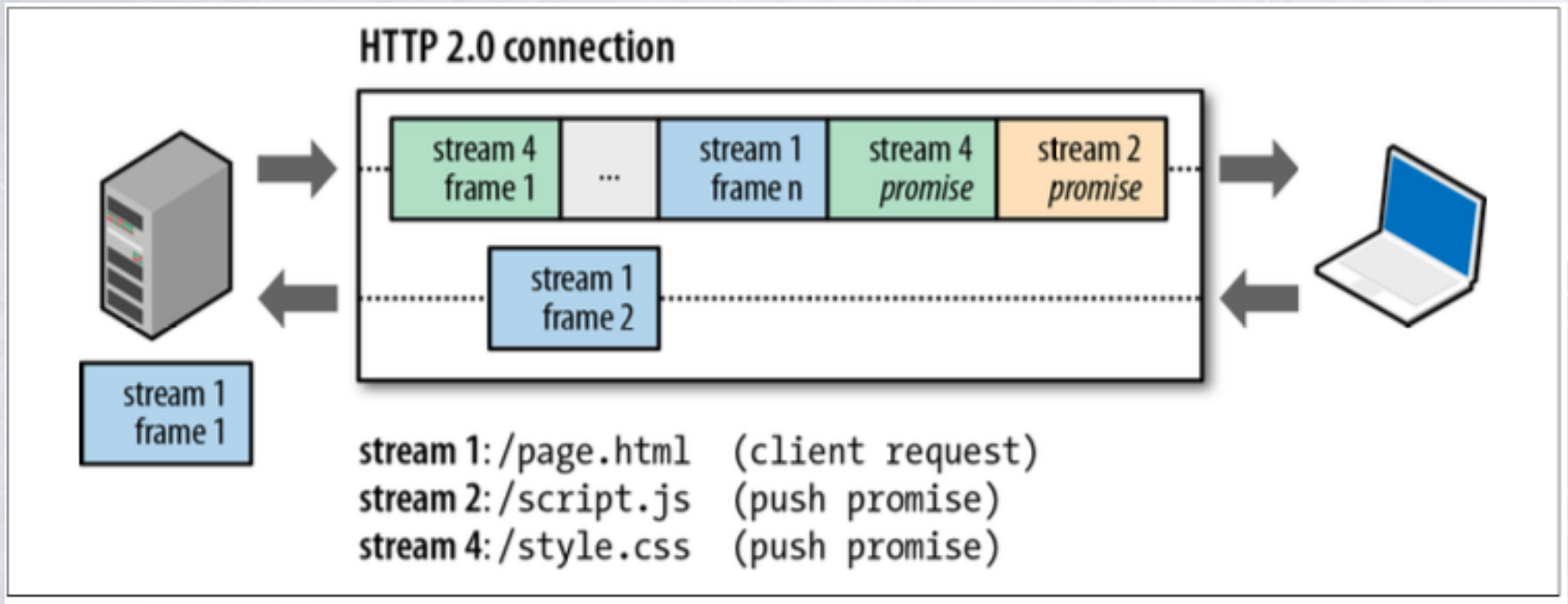
One Connection Per Origin

- With the new binary framing mechanism in place, HTTP 2.0 no longer needs multiple TCP connections to multiplex streams in parallel
- One connection per origin significantly reduces the associated overhead: fewer sockets to manage along the connection path, smaller memory footprint, and better connection throughput. Plus, many other benefits at all layers of the stack:
  - Consistent prioritization between all streams
  - Better compression through use of a single compression context
  - Improved impact on network congestion due to fewer TCP connections
  - Less time in slow-start and faster congestion and loss recovery

Flow Control

- Flow control is hop-by-hop, not end-to-end.
- Flow control is based on window update frames: receiver advertises how many bytes it is prepared to receive on a stream and for the entire connection.
- Flow control window size is updated by a WINDOW_UPDATE frame, which specifies the stream ID and the window size increment value.
- Flow control is directional: receiver may choose to set any window size that it desires for each stream and for the entire connection.
- Flow control can be disabled by a receiver, both for an individual stream or for the entire connection.

Server Push


HTTP 2.0 connection
stream 4 frame 1
...
stream 1 frame n
stream 4 promise
stream 2 promise
stream 1 frame 2
stream 1 frame 1
stream 1: /page.html (client request)
stream 2: /script.js (push promise)
stream 4: /style.css (push promise)

important benefits

- Pushed resources can be cached by the client.
- Pushed resources can be declined by the client.
- Pushed resources can be reused across different pages.
- Pushed resources can be prioritized by the server.

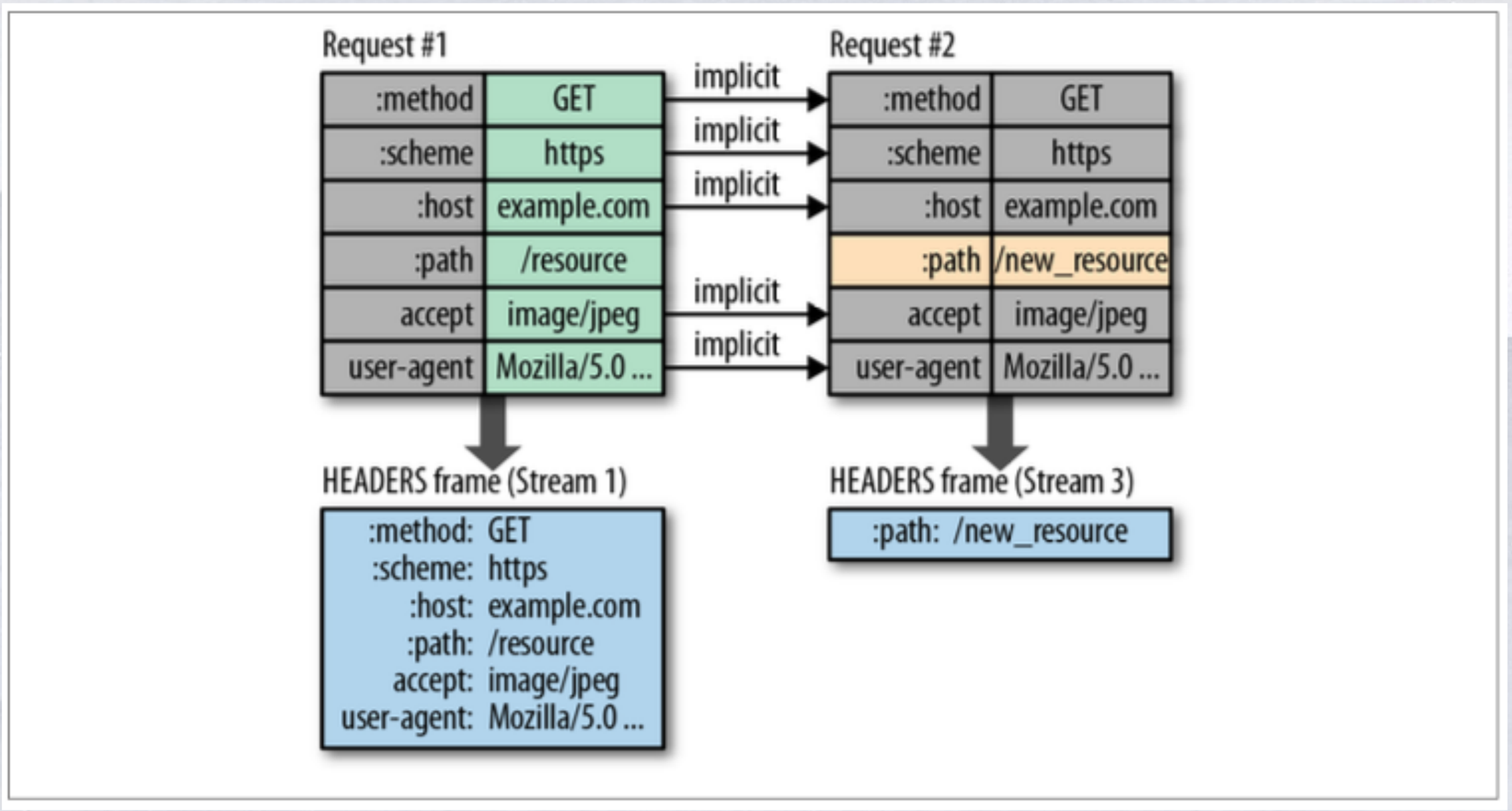
Header Compression

- Instead of retransmitting the same data on each request and response, HTTP 2.0 uses “header tables” on both the client and server to track and store previously sent key-value pairs.

- Header tables persist for the entire HTTP 2.0 connection and are incrementally updated both by the client and server.

- Each new header key-value pair is either appended to the existing table or replaces a previous value in the table.

Differential coding of HTTP 2.0 headers



Efficient HTTP 2.0 Upgrade and Discovery

```
GET /page HTTP/1.1
  Host: server.example.com
  Connection: Upgrade, HTTP2-Settings
  Upgrade: HTTP/2.0
  HTTP2-Settings: (SETTINGS payload)
  HTTP/1.1 200 OK
  Content-length: 243
  Content-type: text/html
  (... HTTP 1.1 response ...)
          (or)
  HTTP/1.1 101 Switching Protocols
  Connection: Upgrade
  Upgrade: HTTP/2.0
  (... HTTP 2.0 response ...)
```

```
const http2 = require('http2');
const fs = require('fs');

const server = http2.createSecureServer({
  key: fs.readFileSync('localhost-privkey.pem'),
  cert: fs.readFileSync('localhost-cert.pem')
});
server.on('error', (err) => console.error(err));

server.on('stream', (stream, headers) => {
  // stream is a Duplex
  stream.respond({
    'content-type': 'text/html',
    ':status': 200
  });
  stream.end('<h1>Hello World</h1>');
});

server.listen(8443);
```

https://nodejs.org/api/http2.html#http2_server_side_example

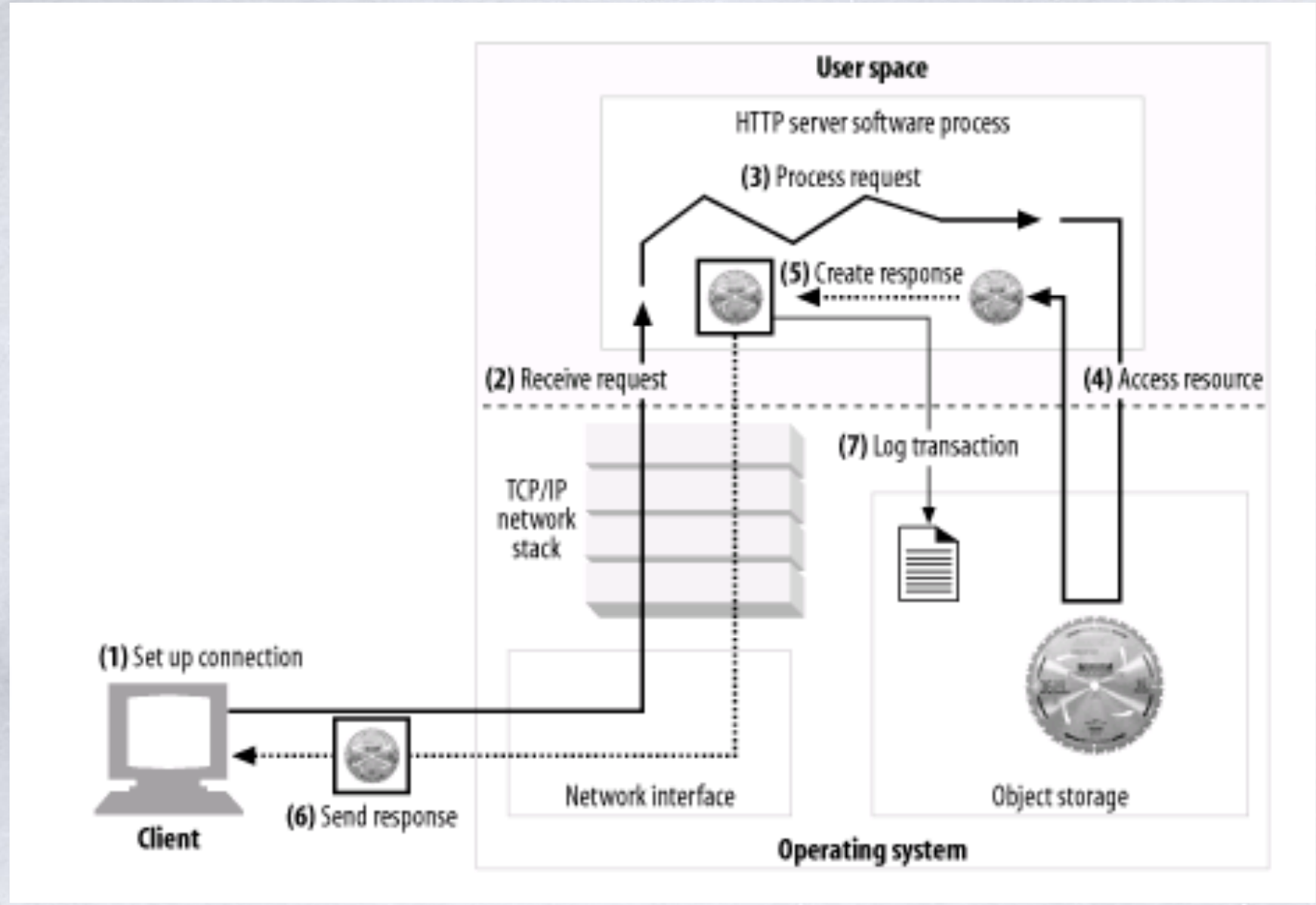
Outline

- 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



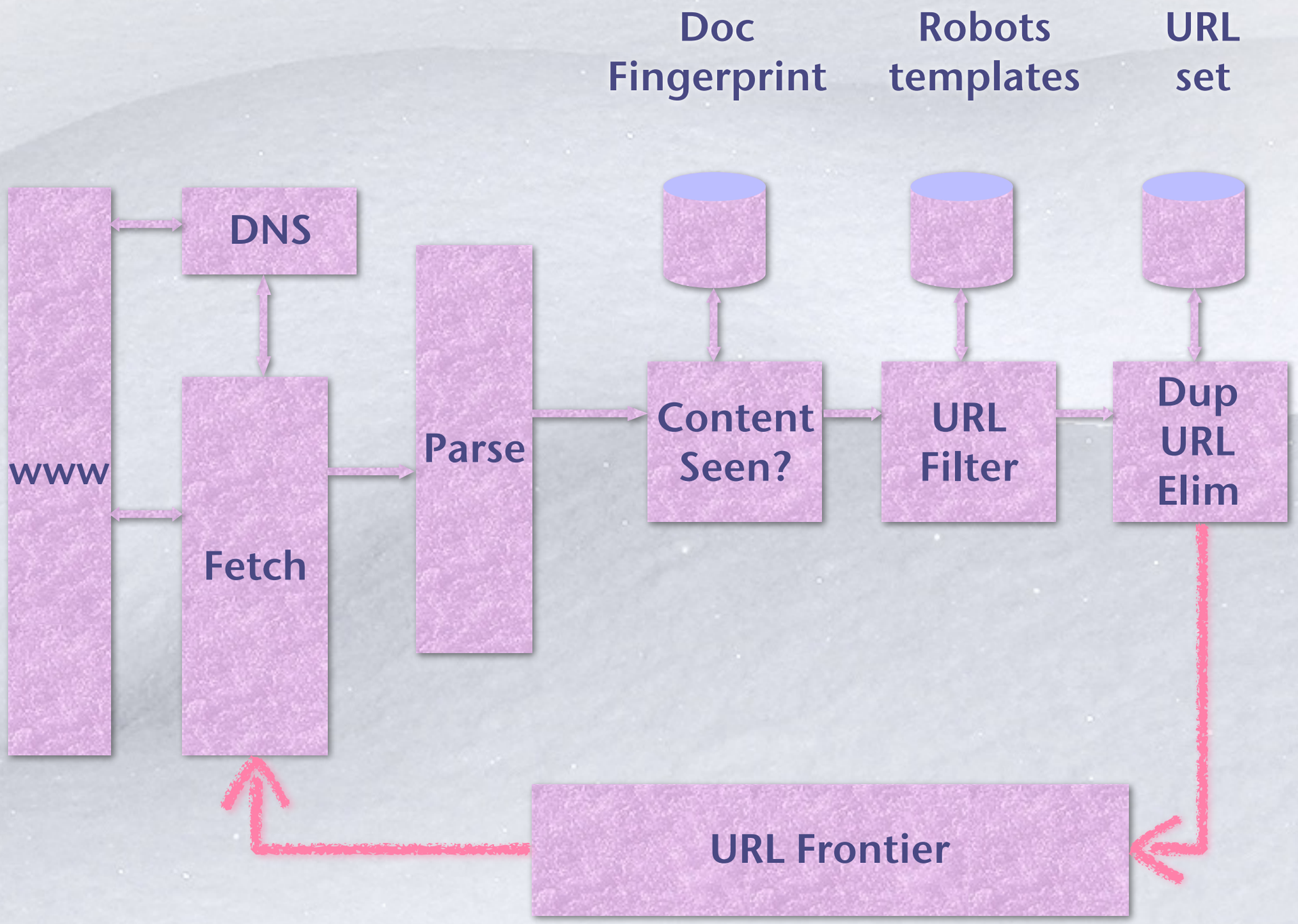
Web Client design

- 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

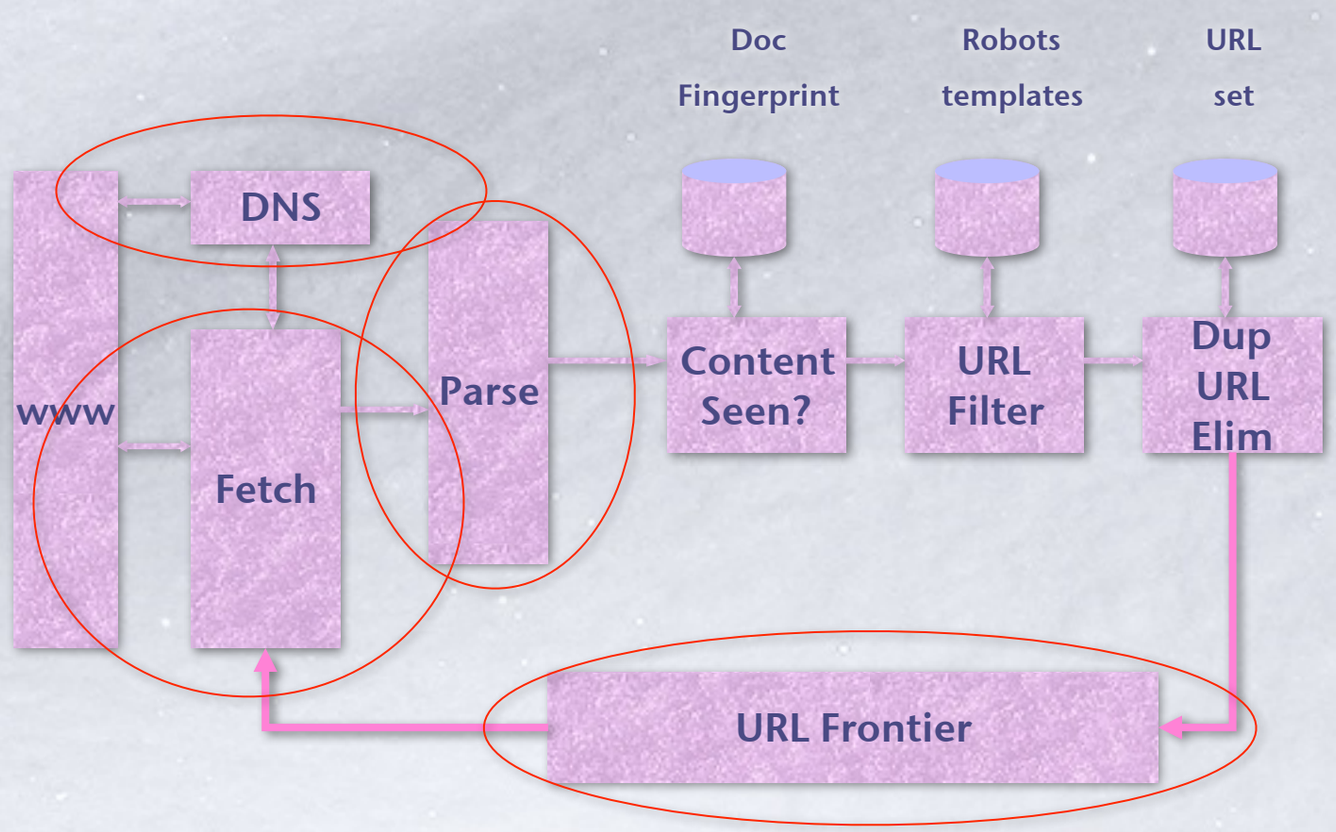
- Should provide:
  - Distributed
  - Scalable
  - Performance and efficiency
  - Quality
  - Freshness
  - Update
  - Extensible

Architecture of a crawler



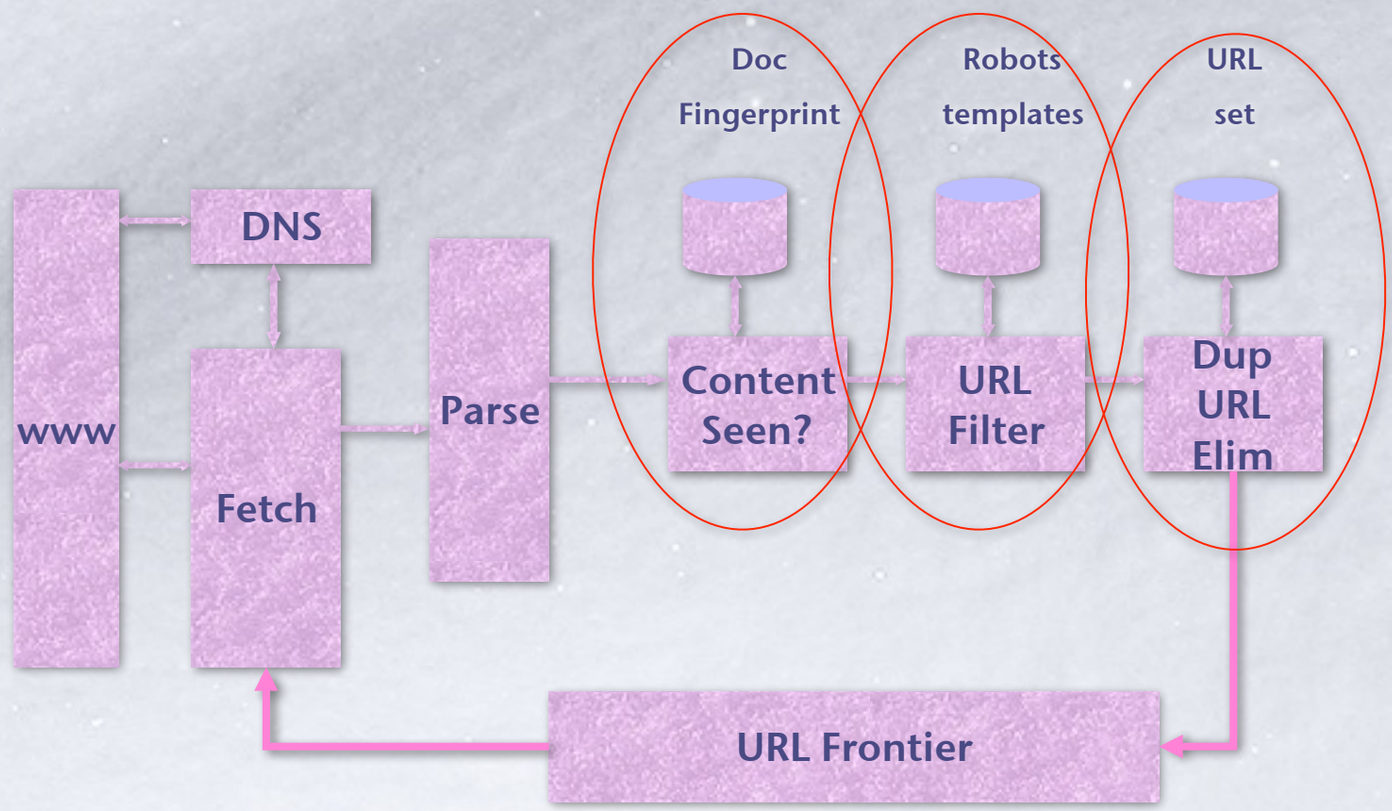
Architecture of a crawler (Cont'd)

- URL Frontier: containing URLs yet to be fetches 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 “IxeBot":

```
User-agent: *
Disallow: /yoursite/temp/

User-agent: IxeBot
Disallow:
```

References

- High Performance Browser Networking
- http://httpwg.org/specs/rfc7540.html
- RFC2616 HTTP1.1
- RFC1945 HTTP1.0
- David Gourley,etc. HTTP: The Definitive Guide

Thanks!!!