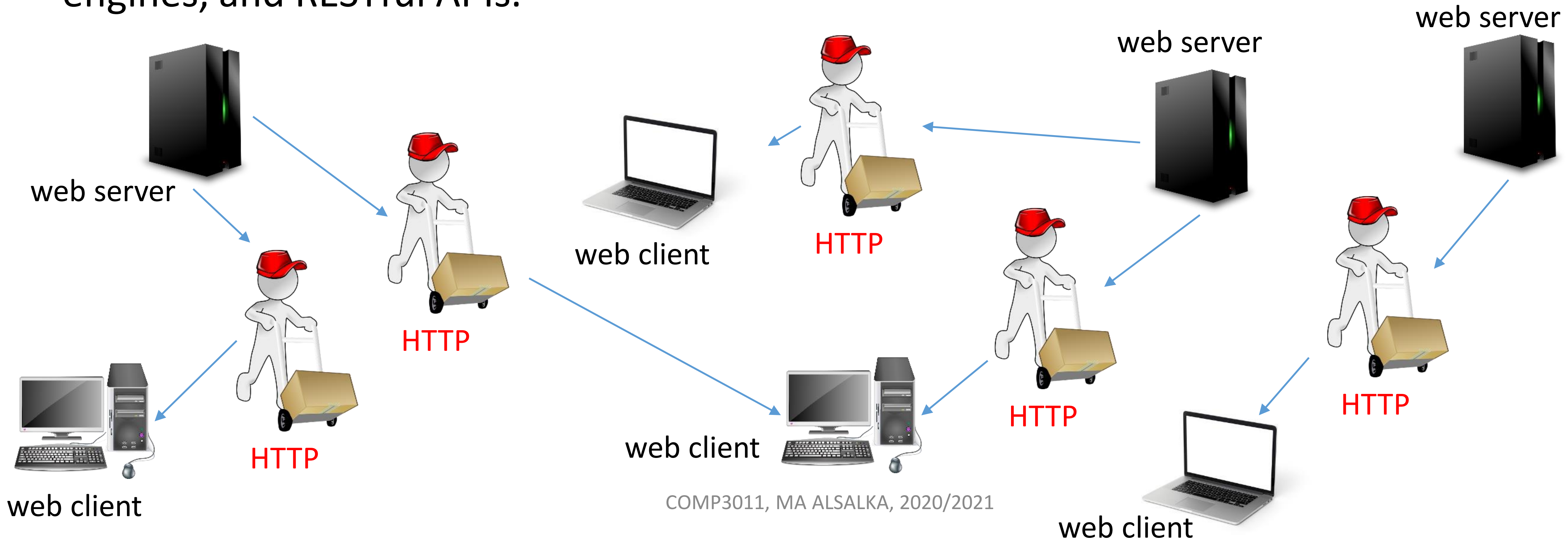
Web Services and Web Data
COMP3011

Unit 2. HTTP – The Workhorse of the Web

UNIVERSITY OF LEEDS
School of Computing

The Hypertext Transfer Protocol (HTTP) *The Multimedia Courier of the Web*

- Every day, billions of multimedia content (e.g. JPEG images, HTML pages, text files, MPEG movies, WAV audio files, Java applets) cruise through the Internet using the HTTP protocol.
- HTTP moves information quickly and reliably from web servers all around the world to web browsers on people's desktops.
- HTTP is an application layer protocol that dates back to 1991, but is still the workhorse of the world wide web.
- A good understanding of this protocol is essential for building web applications, search engines, and RESTful APIs.

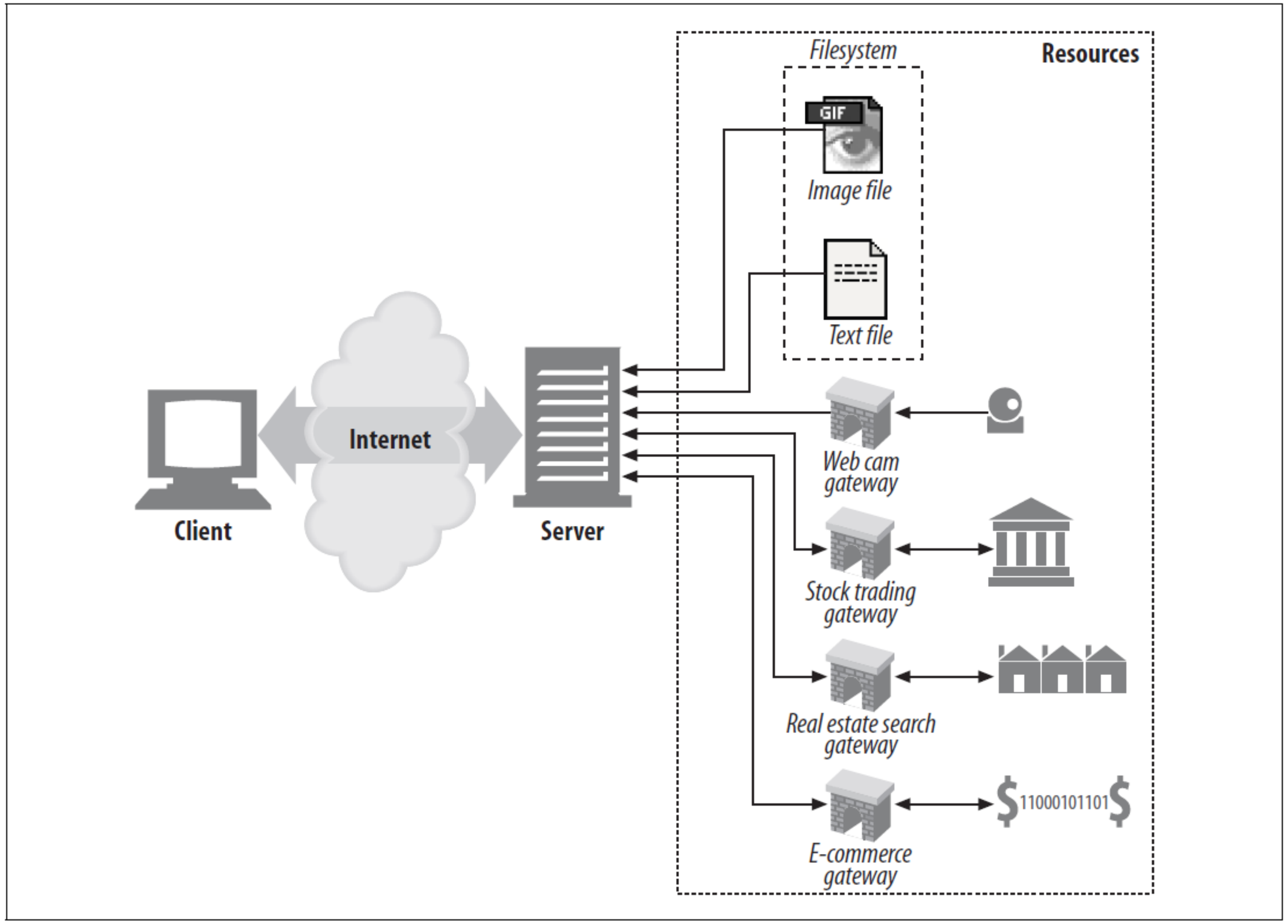


A Quick Introduction to the HTTP Protocol

Web Resources

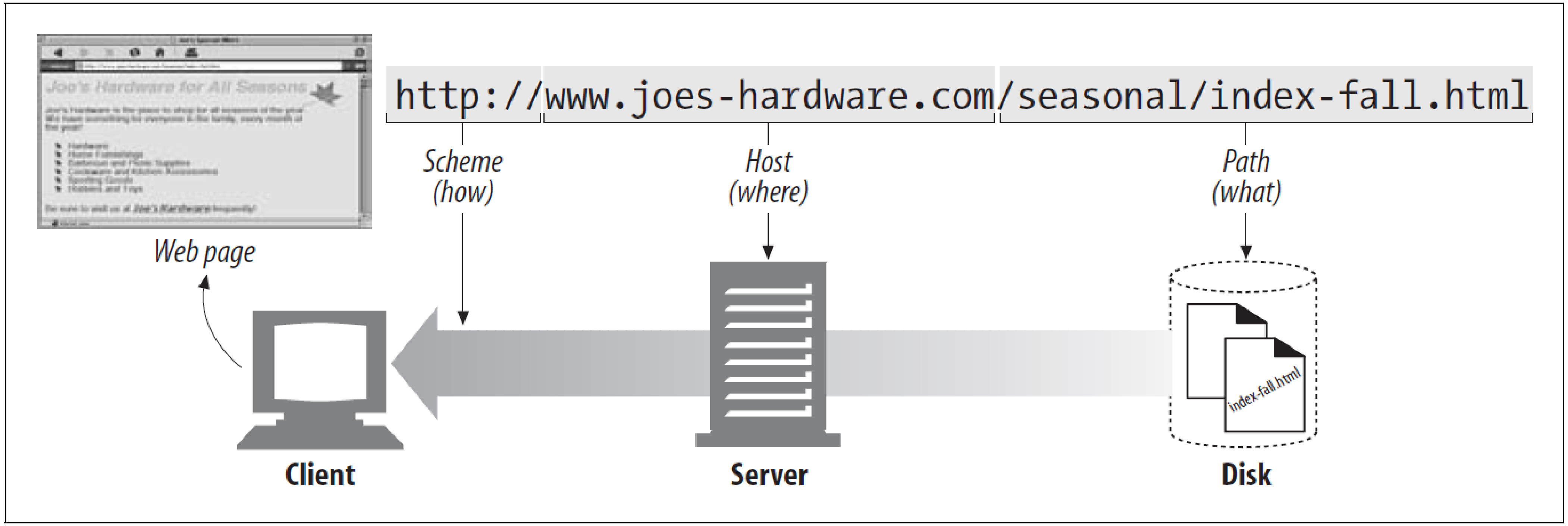
COMP3011, MA ALSALKA, 2020/2021

- Web servers host *web resources*.
- A web resource is the source of web content.
- The simplest kind of web resource is a static file on the web server's filesystem (text, HTML, Microsoft Word, Adobe Acrobat, JPEG image, AVI movie, .. etc.).
- However, resources do NOT have to be static files. Resources can also be software programs that generate content on demand.

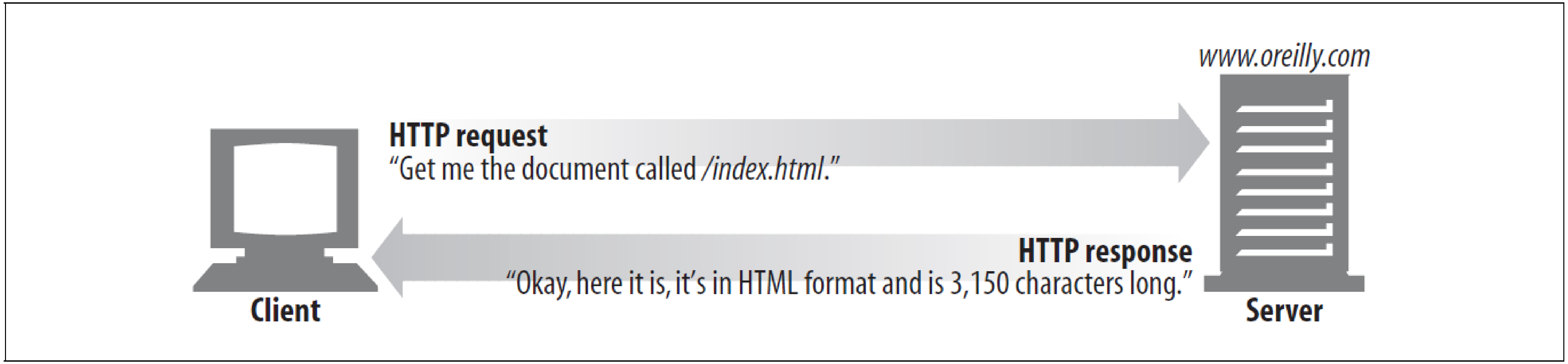


URIs

- Each web resource has a name, so clients can point out what resources they are interested in.
- The resource name is called a *uniform resource identifier*, or URI.
- URIs have the general form: “scheme://server location/path”



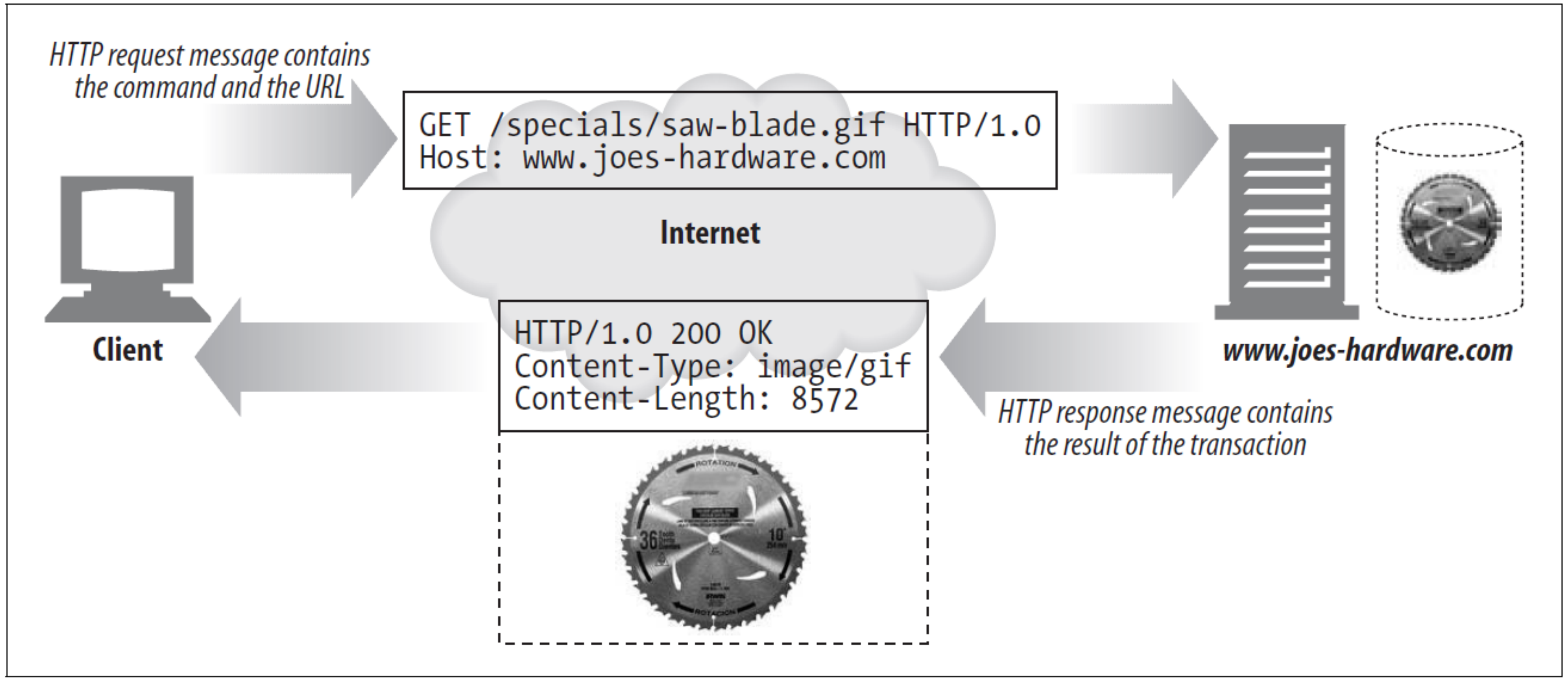
HTTP uses a Client Server Model


www.oreilly.com
HTTP request
"Get me the document called /index.html."
HTTP response
"Okay, here it is, it's in HTML format and is 3,150 characters long."
Client
Server

HTTP Message Types

All HTTP messages fall into two types: request messages and response messages.

- Request messages request an action from a web server.
- Response messages carry results of a request back to a client. Both request and response messages have the same basic message structure



HTTP request message contains the command and the URL
Internet
Client
www.joes-hardware.com
HTTP response message contains the result of the transaction

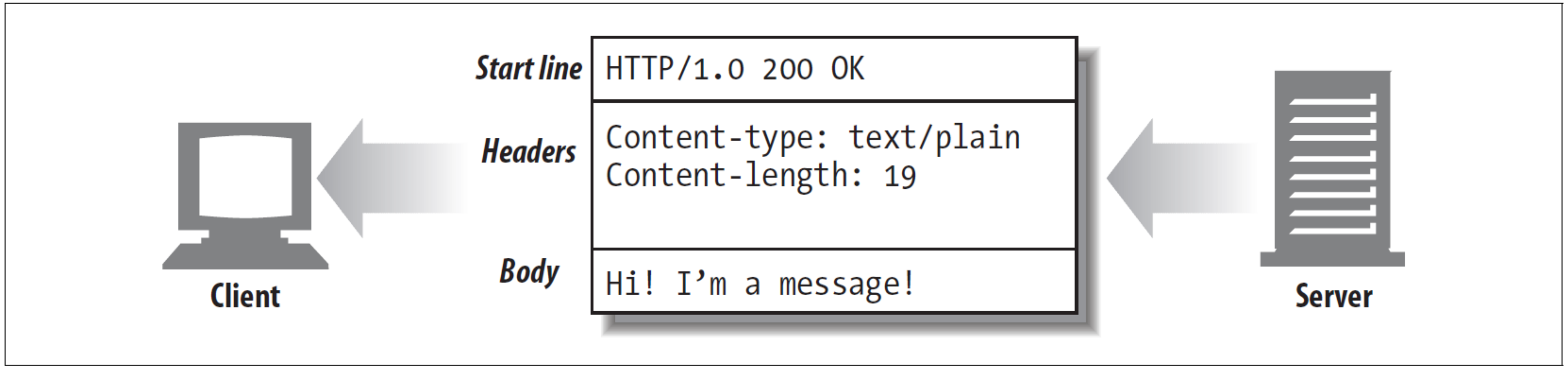
```
GET /specials/saw-blade.gif HTTP/1.0
Host: www.joes-hardware.com
```

```
HTTP/1.0 200 OK
Content-Type: image/gif
Content-Length: 8572
```

The Parts of an HTTP Message

- A start line describing the message.
- A block of headers containing some attributes.
- An optional body containing data.

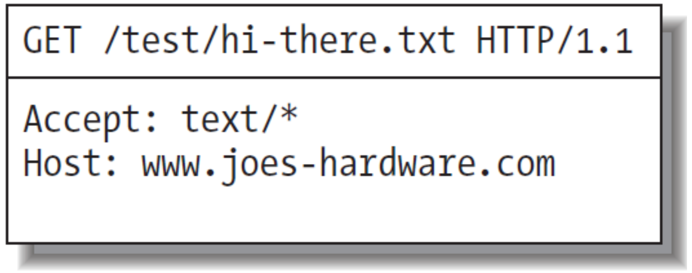
Each line ends with a two-character end-of-line sequence, consisting of a carriage return (ASCII 13) and a line-feed character (ASCII 10), called CRLF (or \r\n in C like languages).



```
HTTP/1.0 200 OK
Content-type: text/plain
Content-length: 19

Hi! I’m a message!
```

Format of a Request Message

```
<method> <request-URL> <version>
<headers>
<entity-body>
```



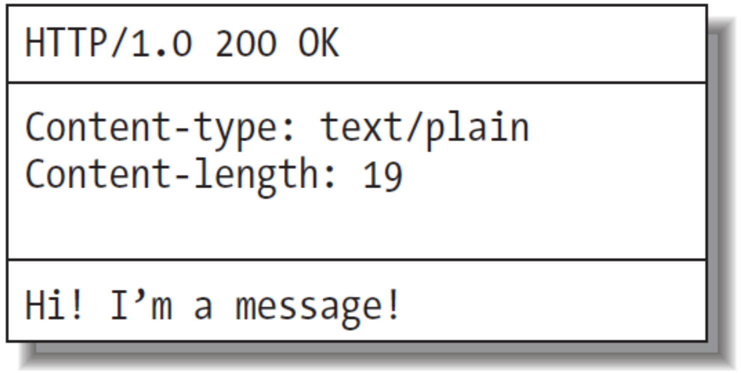
```
GET /test/hi-there.txt HTTP/1.1

Accept: text/*
Host: www.joes-hardware.com
```

Example

- <method> The action that the client wants the server to perform on the resource. It is a single verb such as “GET,” “HEAD,” or “POST”
- <request-URL> A complete URL naming the requested resource, or the path component of the URL.
- <version> The version of HTTP that the message is using. Its format looks like: HTTP/<major>.<minor>
- <headers> Zero or more headers, each of which is a name, then a colon (:), then a value.
- Headers are terminated by a blank line (CRLF), marking the end of headers and the beginning of the entity body.
- <entity-body> The entity body contains a block of arbitrary data. Not all messages contain entity bodies, so sometimes a message terminates with a bare CRLF.

Format of a Response Message

```
<version> <status> <reason-phrase>
<headers>
<entity-body>
```



```
HTTP/1.0 200 OK

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

Hi! I'm a message!
```

Example

- <status> A three-digit number describing what happened during the request. Remember, for example, the notorious 404 (Not Found) status code.
- The first digit describes the general class of status (“success,” “error,” etc.).
- <reason-phrase> A human-readable version of the numeric status code, consisting of all the text until the end-of-line sequence.

Status Code Classes

Overall range	Defined range	Category
100-199	100-101	Informational
200-299	200-206	Successful
300-399	300-305	Redirection
400-499	400-415	Client error
500-599	500-505	Server error

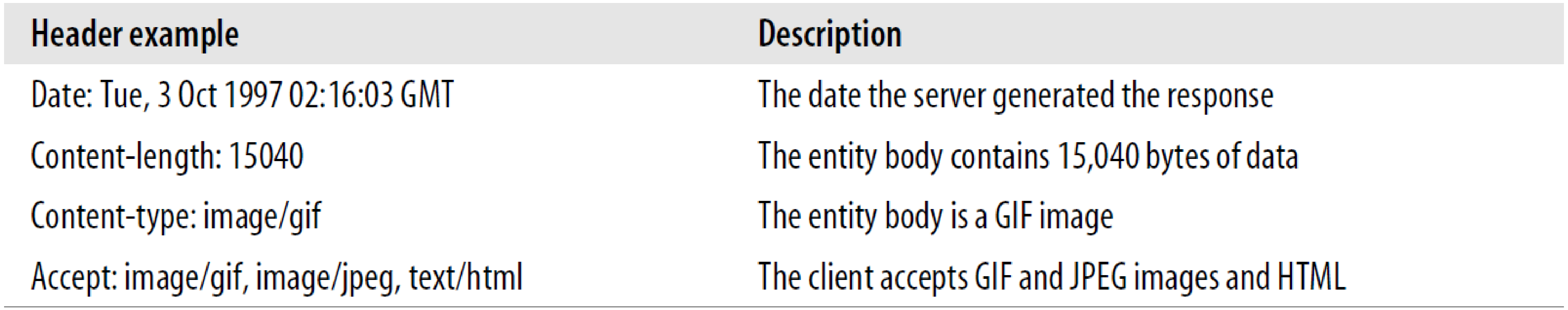
Examples

Status code	Reason phrase	Meaning
200	OK	Success! Any requested data is in the response body.
401	Unauthorized	You need to enter a username and password.
404	Not Found	The server cannot find a resource for the requested URL.

The Header Lines

Each HTTP header has a simple syntax: a name, followed by a colon (:), followed by the field value, followed by a CRLF

Examples



Header example	Description
Date: Tue, 3 Oct 1997 02:16:03 GMT	The date the server generated the response
Content-length: 15040	The entity body contains 15,040 bytes of data
Content-type: image/gif	The entity body is a GIF image
Accept: image/gif, image/jpeg, text/html	The client accepts GIF and JPEG images and HTML

Header Types

- General headers used by both clients and servers. For example, the Date header:

  Date: Tue, 3 Oct 1974 02:16:00 GMT
- Request headers specific to request messages. They provide extra information to servers, such as what type of data the client is willing to receive. For example, to accept any media type we use:

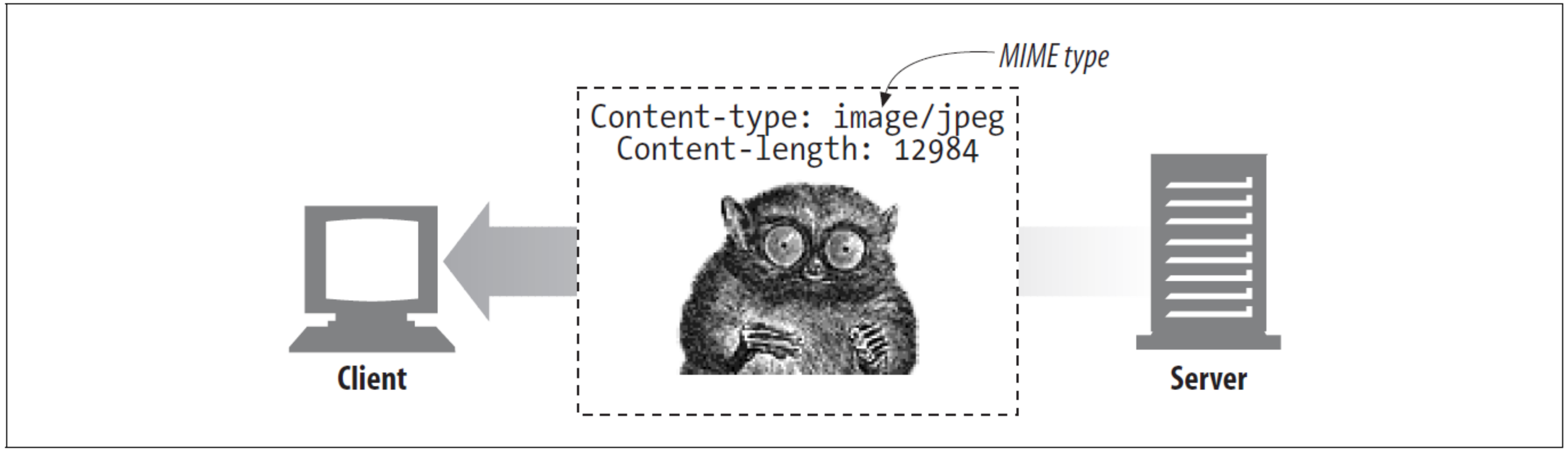
  Accept: */*
- Response headers provide information to the client. For example, to tell the client that it is talking to a Version 1.0 Tiki-Hut server:

  Server: Tiki-Hut/1.0
- Entity headers refer to headers that deal with the entity body. For example, the following Content-Type header lets the application know that the data is an HTML document in the iso-latin-1 character set:

  Content-Type: text/html; charset=iso-latin-1

Media Types

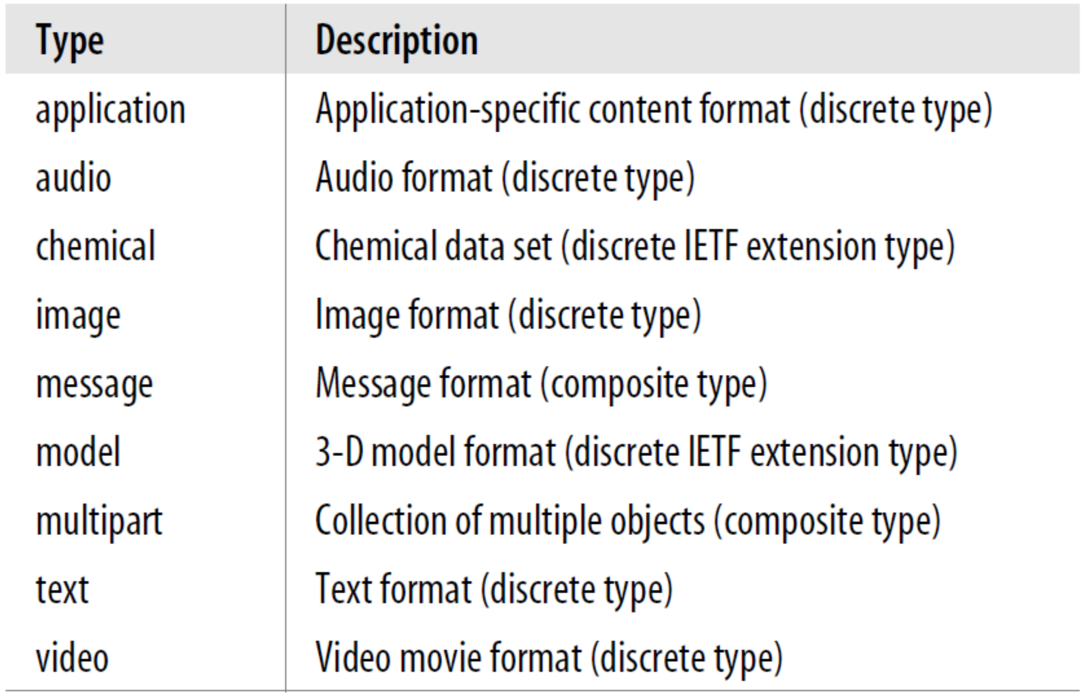
- In HTTP, MIME media types are widely used in Content-Type and Accept headers.
- HTTP tags each object being transported through the Web with a data format label (content type header) called a *MIME type*.
- MIME (Multipurpose Internet Mail Extensions) was originally designed for moving messages between different electronic mail systems.
- MIME worked very well, so HTTP adopted it to describe and label its own multimedia content.



MIME types structure

- Each MIME media type consists of a primary type, a subtype, and a list of optional parameters.
- The type and subtype are separated by a slash, and the optional parameters begin with a semicolon.
- The primary type can be a predefined type, an IETF-defined (Internet Engineering Task Force) extension token, or an experimental token (beginning with “x-”). Here are a few examples:

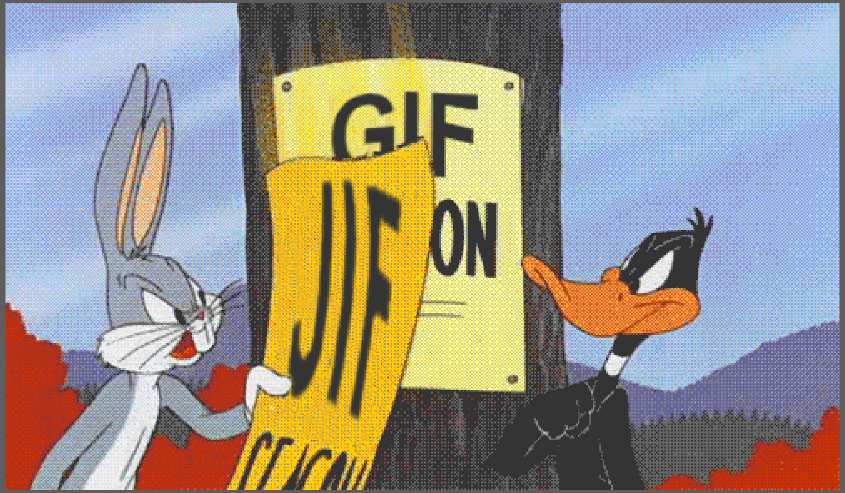
Examples



Type	Description
application	Application-specific content format (discrete type)
audio	Audio format (discrete type)
chemical	Chemical data set (discrete IETF extension type)
image	Image format (discrete type)
message	Message format (composite type)
model	3-D model format (discrete IETF extension type)
multipart	Collection of multiple objects (composite type)
text	Text format (discrete type)
video	Video movie format (discrete type)

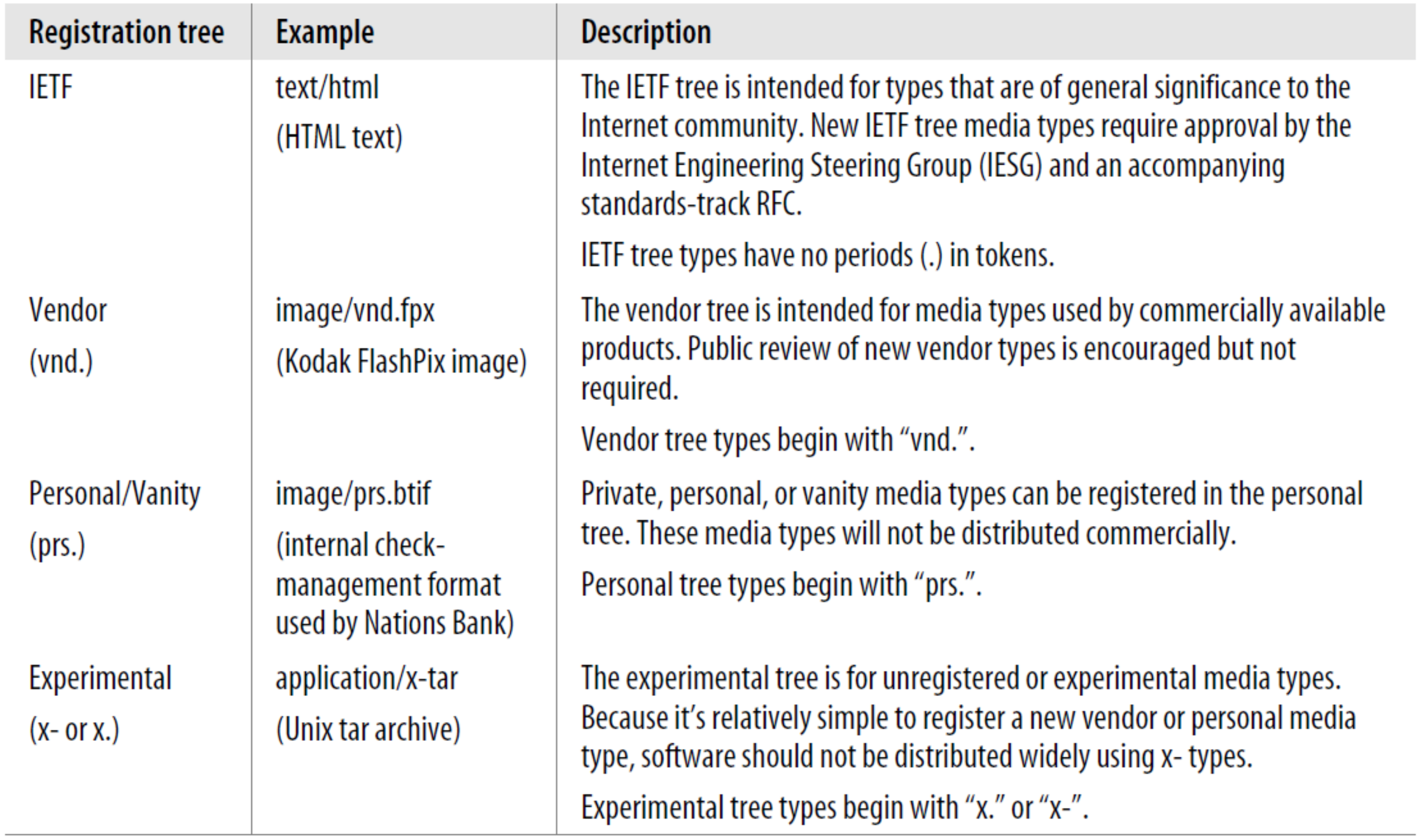
MIME type examples:

- An HTML-formatted text document would be text/html.
- A plain ASCII text document would be text/plain.
- A JPEG version of an image would be image/jpeg.
- An Apple QuickTime movie would be video/quicktime.
- A Microsoft PowerPoint presentation would be application/vnd.ms-powerpoint.
- A GIF-format image would be image/gif.



MIME Type IANA Registration

- MIME types should be registered with IANA (Internet Assigned Numbers Authority)
- Registration is simple and open to all.
- MIME type tokens are split into four classes, called “registration trees,” each with its own registration rules.



Registration tree	Example	Description
IETF	text/html (HTML text)	The IETF tree is intended for types that are of general significance to the Internet community. New IETF tree media types require approval by the Internet Engineering Steering Group (IESG) and an accompanying standards-track RFC. IETF tree types have no periods (.) in tokens.
Vendor (vnd.)	image/vnd.fpx (Kodak FlashPix image)	The vendor tree is intended for media types used by commercially available products. Public review of new vendor types is encouraged but not required. Vendor tree types begin with “vnd.”.
Personal/Vanity (prs.)	image/prs.btif (internal check-management format used by Nations Bank)	Private, personal, or vanity media types can be registered in the personal tree. These media types will not be distributed commercially. Personal tree types begin with “prs.”.
Experimental (x- or x.)	application/x-tar (Unix tar archive)	The experimental tree is for unregistered or experimental media types. Because it’s relatively simple to register a new vendor or personal media type, software should not be distributed widely using x- types. Experimental tree types begin with “x.” or “x-”.

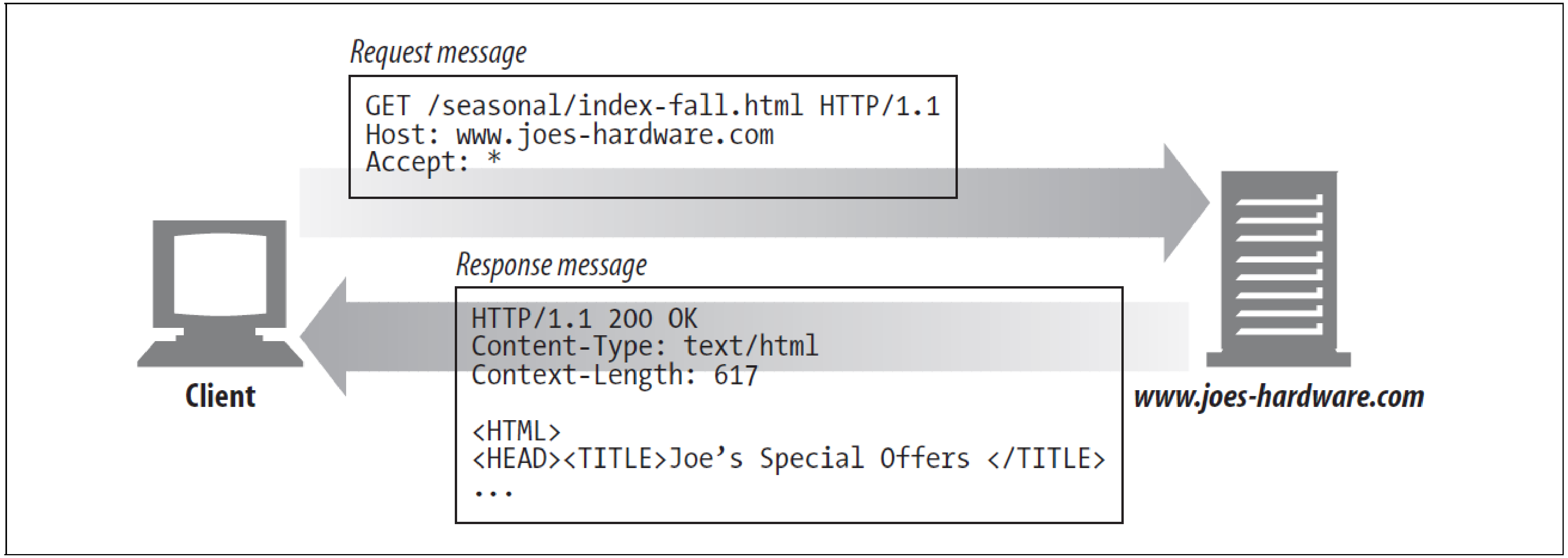
Common HTTP Methods (verbs)

Method	Description	Message body?
GET	Get a document from the server.	No
HEAD	Get just the headers for a document from the server.	No
POST	Send data to the server for processing.	Yes
PUT	Store the body of the request on the server.	Yes
TRACE	Trace the message through proxy servers to the server.	No
OPTIONS	Determine what methods can operate on a server.	No
DELETE	Remove a document from the server.	No

Note that not all methods are implemented by every server. To be compliant with HTTP Version 1.1, a server need implement only the GET and HEAD methods for its resources.

GET

GET is the most common method. It usually is used to ask a server to send a resource.

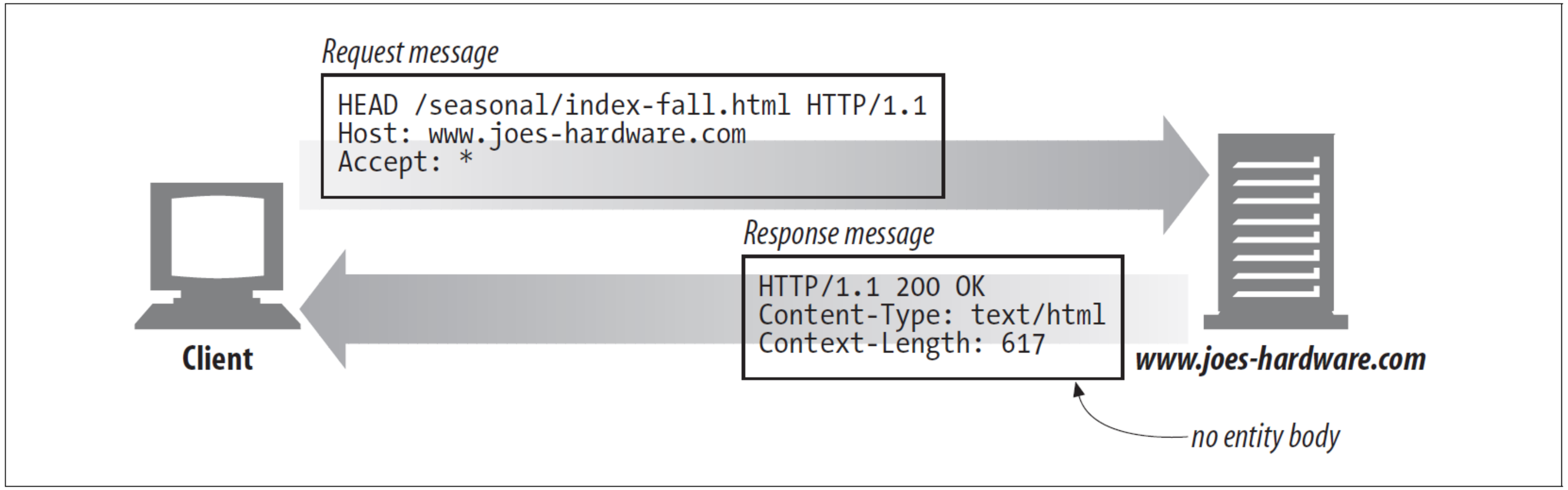


```
GET /seasonal/index-fall.html HTTP/1.1
Host: www.joes-hardware.com
Accept: *
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Context-Length: 617

<HTML>
<HEAD><TITLE>Joe’s Special Offers </TITLE>
...
```

HEAD

- The HEAD method behaves exactly like the GET method, but the server returns only the headers in the response. No entity body is ever returned.
- This allows a client to inspect the headers for a resource without having to actually get the resource.
- Using HEAD, you can:
    - Find out about a resource (e.g., determine its type) without getting it.
    - See if an object exists, by looking at the status code of the response.
    - Test if the resource has been modified, by looking at the headers (Last-Modified).

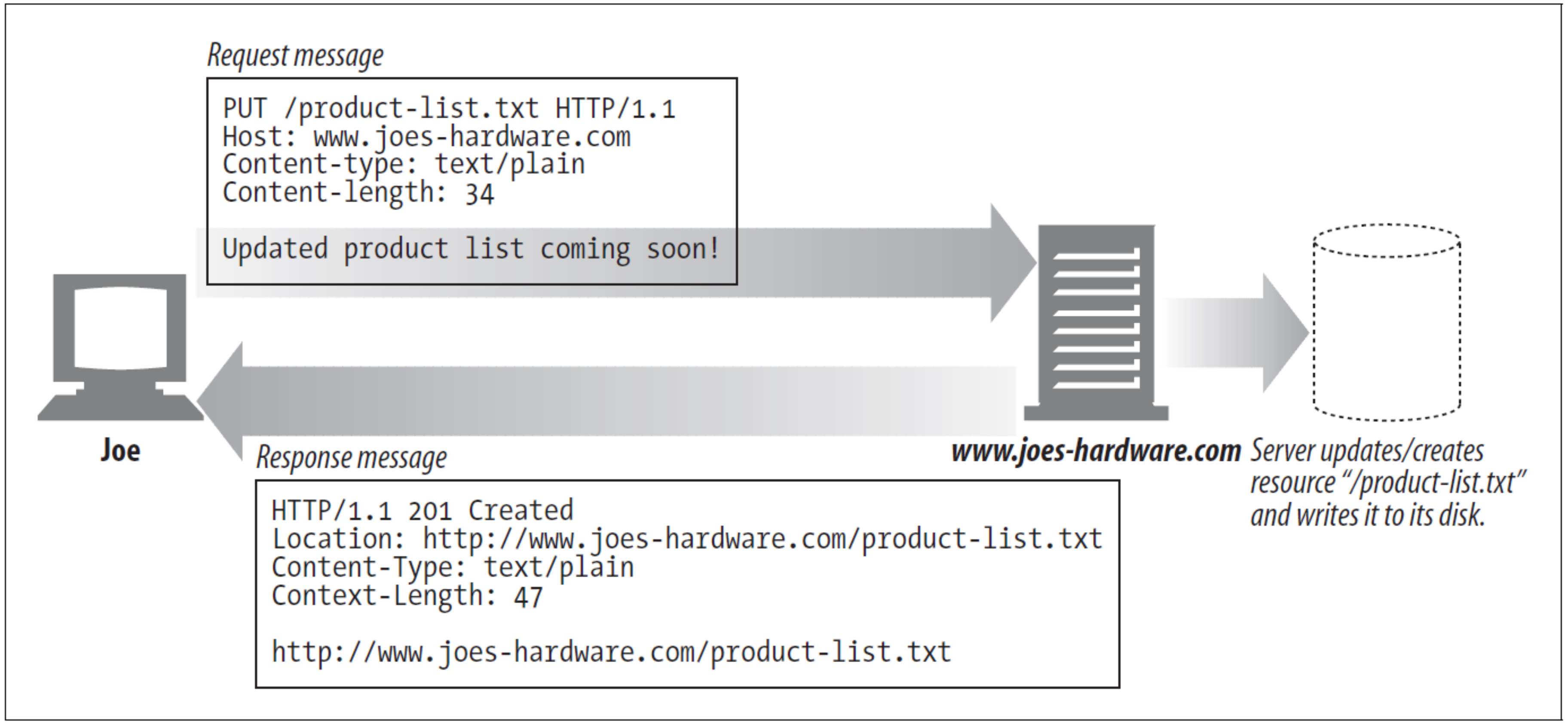


```
HEAD /seasonal/index-fall.html HTTP/1.1
Host: www.joes-hardware.com
Accept: *
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Context-Length: 617
```

PUT

- The PUT method writes documents to a server, in the inverse of the way that GET reads documents from a server.
- Some publishing systems let you create web pages and install them directly on a web server using PUT

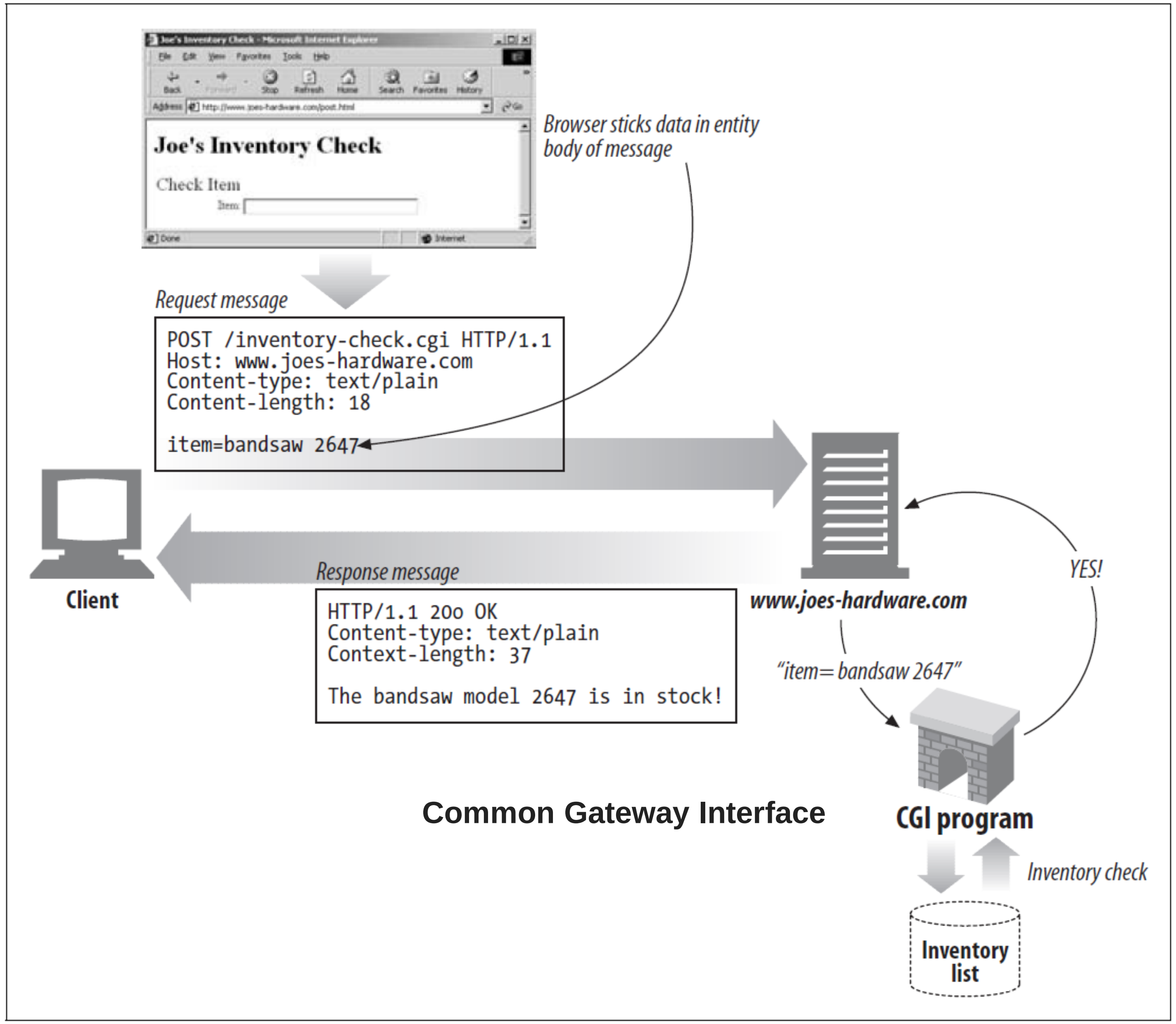

Request message
Joe
Response message
www.joes-hardware.com
Server updates/creates resource "/product-list.txt" and writes it to its disk.

```
PUT /product-list.txt HTTP/1.1
Host: www.joes-hardware.com
Content-type: text/plain
Content-length: 34

Updated product list coming soon!
```

```
HTTP/1.1 201 Created
Location: http://www.joes-hardware.com/product-list.txt
Content-Type: text/plain
Context-Length: 47

http://www.joes-hardware.com/product-list.txt
```

POST

- The POST method sends input data to the server. For example, it is often used to support HTML forms. The data from a filled-in form is sent to the server, which then processes it.

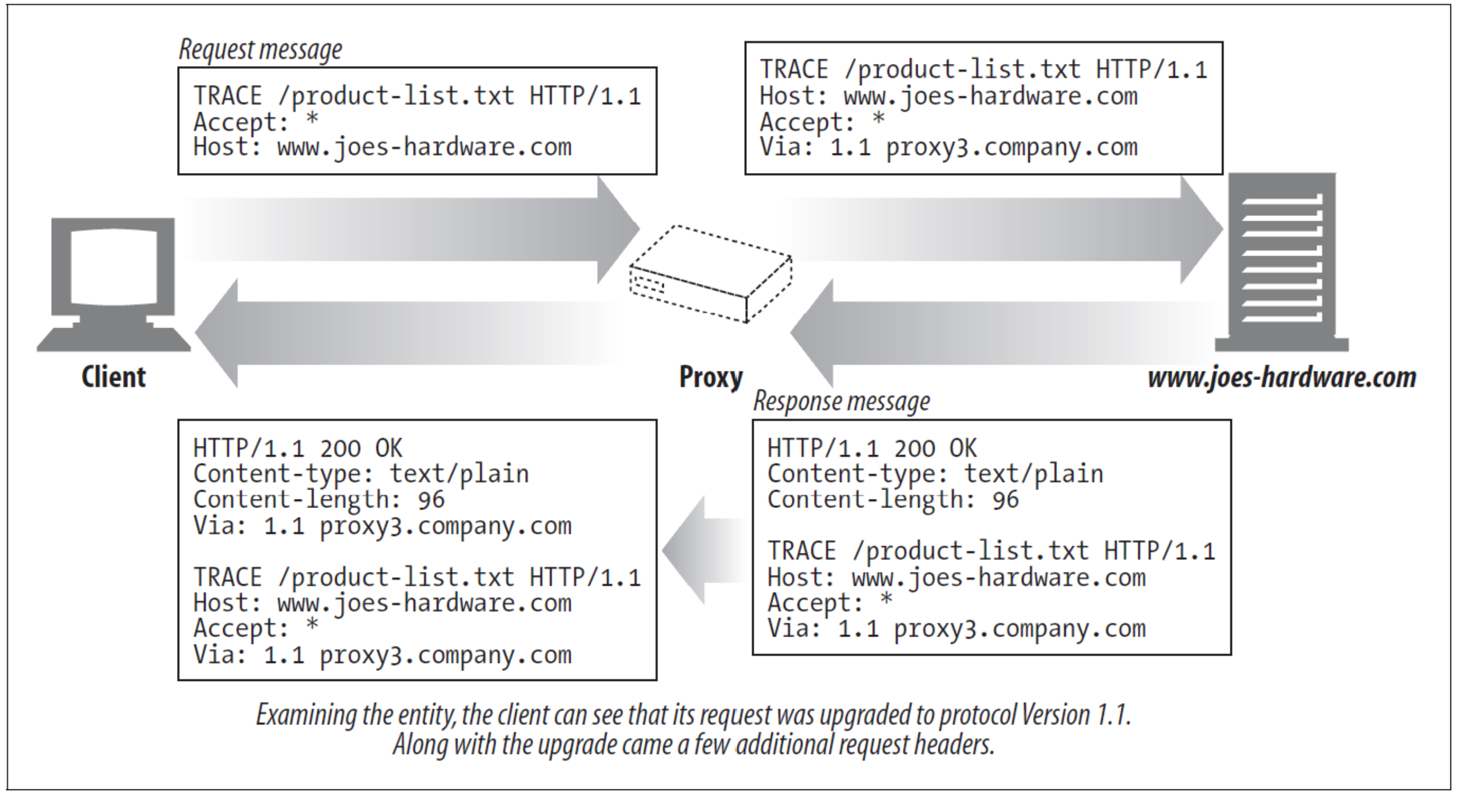


```
POST /inventory-check.cgi HTTP/1.1
Host: www.joes-hardware.com
Content-type: text/plain
Content-length: 18

item=bandsaw 2647
```

```
HTTP/1.1 200 OK
Content-type: text/plain
Context-length: 37

The bandsaw model 2647 is in stock!
```

TRACE

- When a client makes a request, that request may have to travel through firewalls, proxies, gateways, or other applications.
- Each of these has the opportunity to modify the original HTTP request.
- The TRACE method allows clients to see how its request looks when it finally makes it to the server.
- A TRACE request initiates a "loopback" diagnostic at the destination server.
- The server at the final leg of the trip bounces back a TRACE response, with the virgin request message it received in the body of its response.



```
TRACE /product-list.txt HTTP/1.1
Accept: *
Host: www.joes-hardware.com
```

```
TRACE /product-list.txt HTTP/1.1
Host: www.joes-hardware.com
Accept: *
Via: 1.1 proxy3.company.com
```

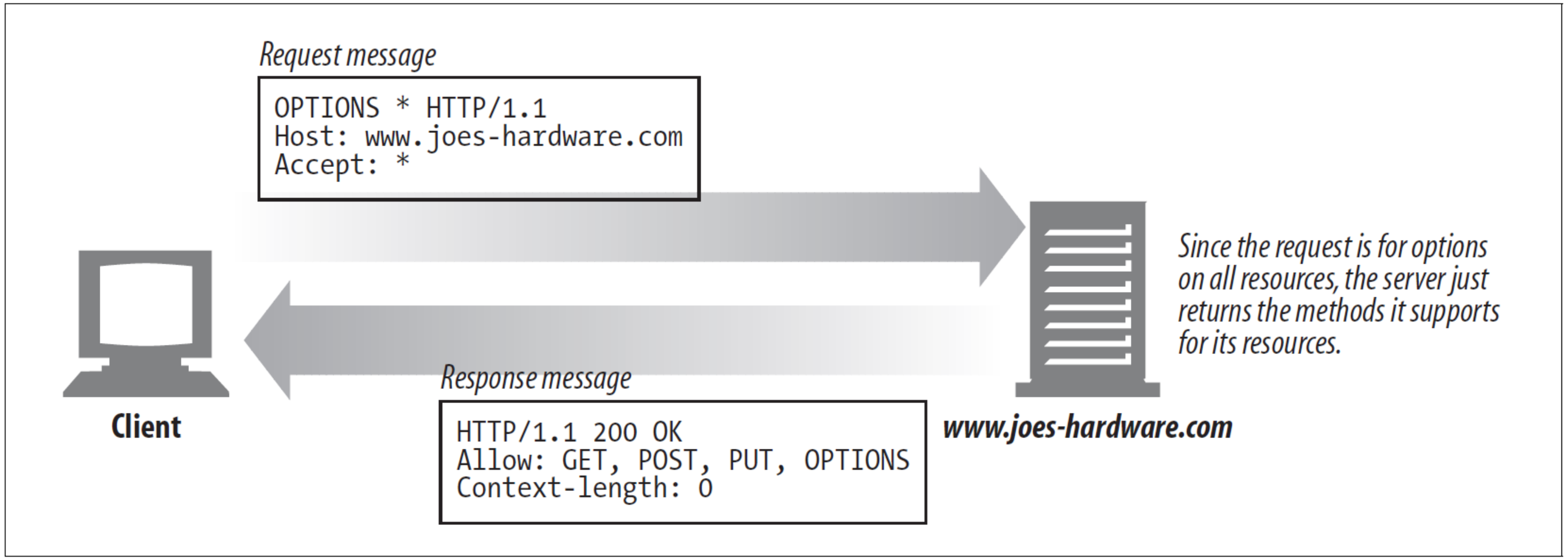
```
HTTP/1.1 200 OK
Content-type: text/plain
Content-length: 96

TRACE /product-list.txt HTTP/1.1
Host: www.joes-hardware.com
Accept: *
Via: 1.1 proxy3.company.com
```

```
HTTP/1.1 200 OK
Content-type: text/plain
Content-length: 96
Via: 1.1 proxy3.company.com

TRACE /product-list.txt HTTP/1.1
Host: www.joes-hardware.com
Accept: *
Via: 1.1 proxy3.company.com
```

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.

OPTIONS

- The OPTIONS method asks the server to tell us about the various supported capabilities of the web server.
- You can ask a server about what methods it supports in general or for particular resources (Some servers may support particular operations only on particular kinds of objects).
- This provides a means for client applications to determine how best to access various resources without actually having to access them.

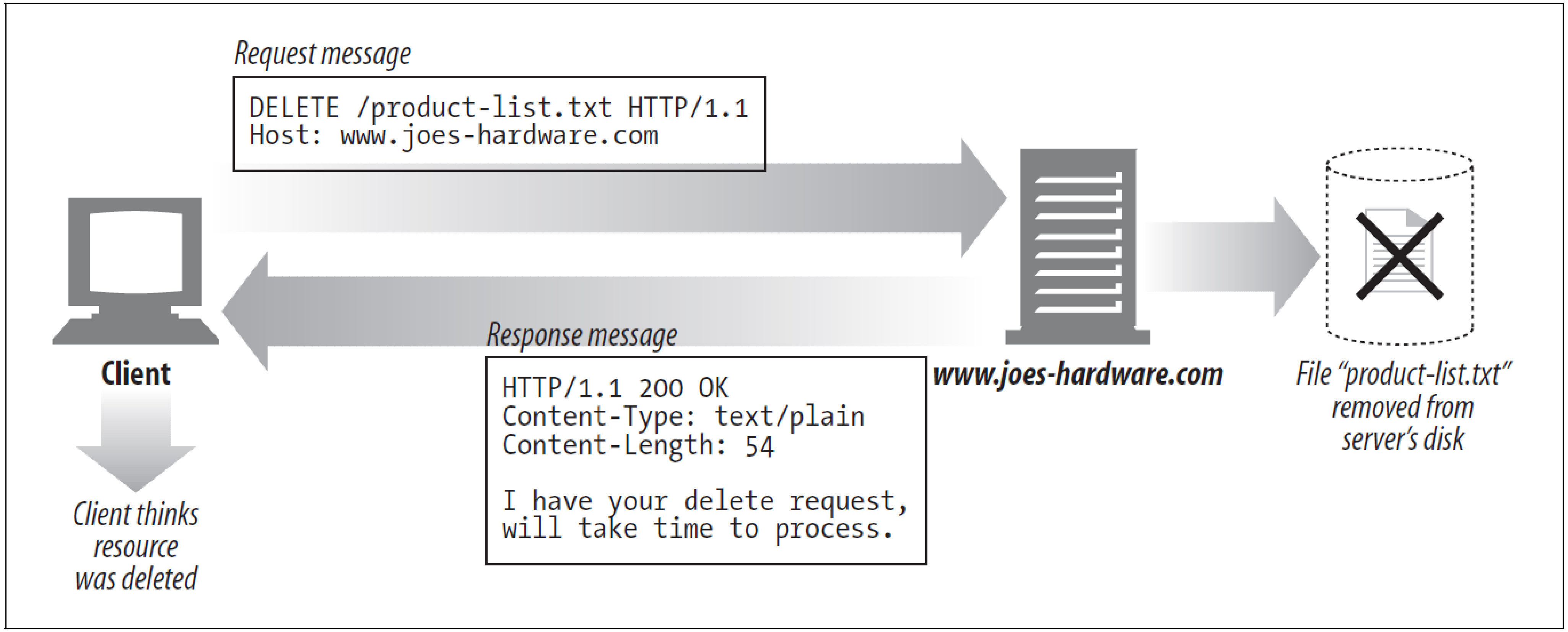

Request message
Client
Since the request is for options on all resources, the server just returns the methods it supports for its resources.
www.joes-hardware.com
Response message

```
OPTIONS * HTTP/1.1
Host: www.joes-hardware.com
Accept: *
```

```
HTTP/1.1 200 OK
Allow: GET, POST, PUT, OPTIONS
Context-length: 0
```

DELETE

- The DELETE method asks the server to delete the resources specified by the request URL.
- However, the client application is not guaranteed that the delete is carried out.
- The HTTP specification allows the server to override the request without telling the client.


Request message
Client
Client thinks resource was deleted
Response message
www.joes-hardware.com
File "product-list.txt" removed from server's disk

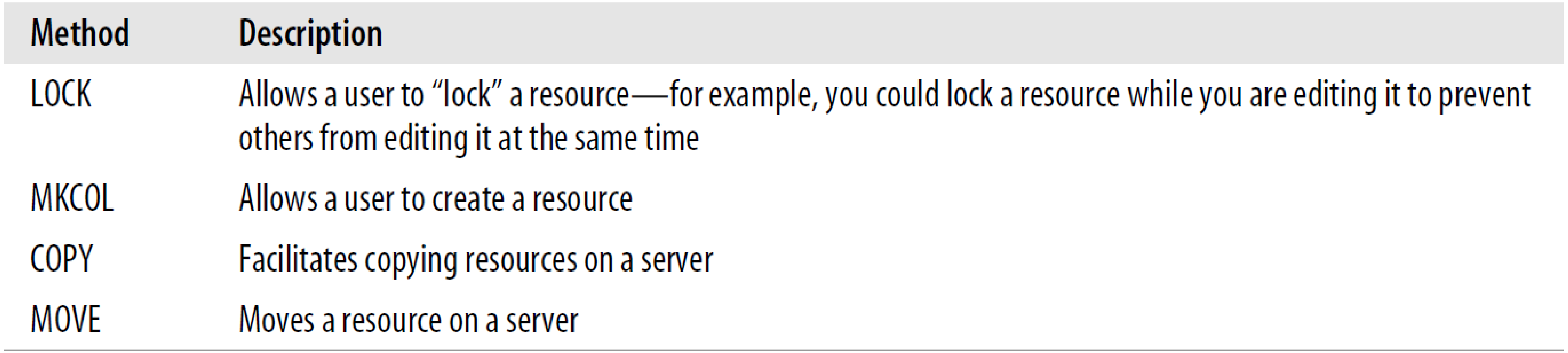
```
DELETE /product-list.txt HTTP/1.1
Host: www.joes-hardware.com
```

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 54

I have your delete request,
will take time to process.
```

Extension Methods

- HTTP was designed to be field-extensible, so new features wouldn’t cause older software to fail.
- Extension methods are methods that are not defined in the HTTP/1.1 specification.
- They provide developers with a means of extending the capabilities of the HTTP services their servers implement on the resources that the servers manage.

Example: the WebDAV HTTP extension that support publishing of web content to web servers over HTTP.



Method	Description
LOCK	Allows a user to “lock” a resource—for example, you could lock a resource while you are editing it to prevent others from editing it at the same time
MKCOL	Allows a user to create a resource
COPY	Facilitates copying resources on a server
MOVE	Moves a resource on a server

That's it Folks

Further Reading

HTTP: The Definitive Guide by Brian Totty and David Gourley