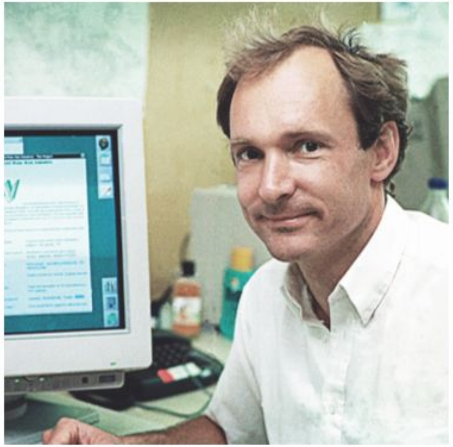
The Web

The Web as we know it was founded in ~1990, by Tim Berners-Lee, physicist at CERN



Tim Berners-Lee Photo: CERN

His goal:

provide distributed access to data

The World Wide Web (WWW):

a distributed database of “pages”
linked together via the
Hypertext Transport Protocol (HTTP)

Why was the web so successful?

- Had networks in mind from the beginning

- What made it successful in the beginning is what makes it successful now
  - It gives a lot of leeway for how websites work (didn’t over-specify)
  - Not tied to any one underlying system
  - No central authority — you can just add your own server/content
  - The ability to quickly navigate information from different sources

The web: basic requirements

- Something to represent content with links: **HTML**

- Client program to access/navigate/display content (e.g. HTML): **Web browser**

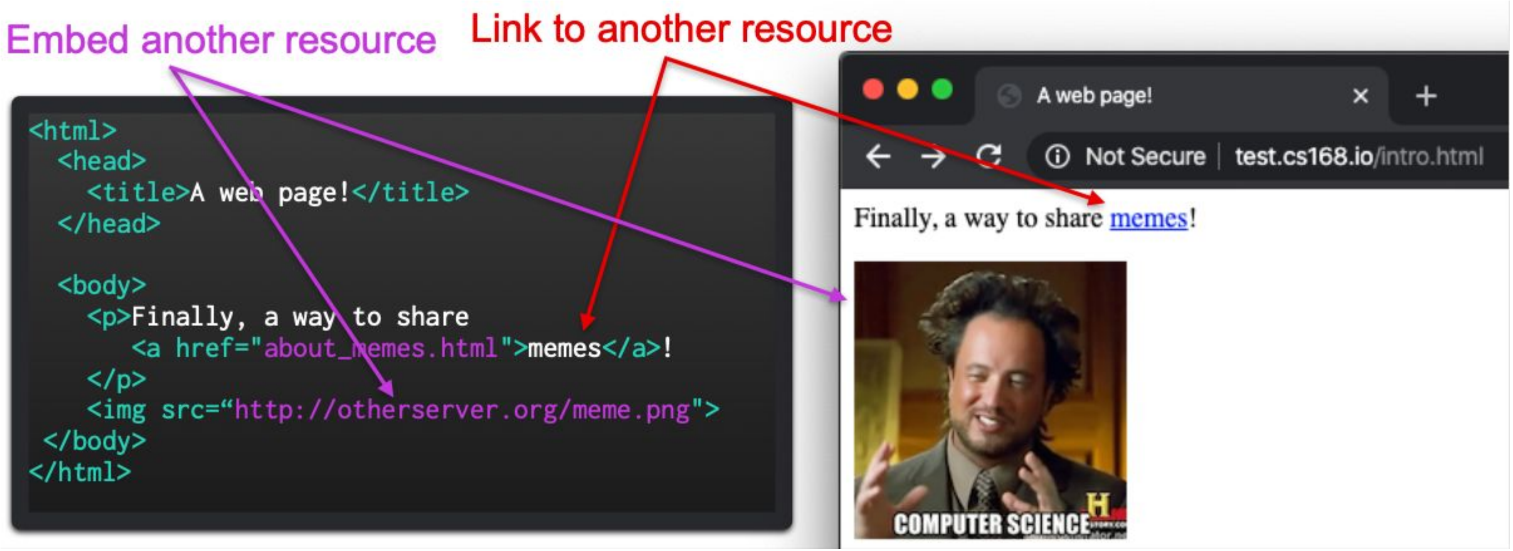
- A way to reference content: **URLs**
    - It’s how you link/embed content to/in other content across a network
    - First general “handle” for arbitrary Internet content
    - Not just naming a host/processes (address/port)

- Something to host content: **Web servers**

- A protocol to get content from server to client: **HTTP**
    - Turns web URLs into TCP connections

Web basics

- HTML: HyperText Markup Language - Represent content with links
- Browser: Access/navigate/display content
- Provide integrated interface to scattered information



Embed another resource

Link to another resource

```
<html>
  <head>
    <title>A web page!</title>
  </head>

  <body>
    <p>Finally, a way to share
       <a href="about_memes.html">memes</a>!
    </p>
    <img src=“http://otherserver.org/meme.png">
 </body>
</html>
```

Finally, a way to share memes!

Web basics: URL syntax

scheme: //host[:port]/path/resource

scheme	Typically a protocol: http, ftp, https, smtp, rtsp, etc.
host	DNS hostname or IP address
port	Defaults to protocol’s standard port e.g. http: 80 https: 443
path	Traditionally reflecting file system
resource	Identifies the desired resource (traditionally a file)

The web: basic requirements

- Something to represent content with links: **HTML**

- Client program to access/navigate/display content (e.g. HTML): **Web browser**

- A way to reference content: **URLs**
  - It’s how you link/embed content to/in other content across a network
  - First general “handle” for arbitrary Internet content
  - Not just naming a host/processes (address/port)

- Something to host content: **Web servers**

- A protocol to get content from server to client: **HTTP**
  - Turns web URLs into TCP connections

HyperText Transfer Protocol (HTTP)

- Focusing our discussion on common/current versions of HTTP:
  - HTTP 1.0 (1996) and HTTP 1.1 (1997)
  - These are (significant) outgrowth of original “HTTP 0.9”
- HTTP 2 published in 2015
  - Largely based on work by Google
  - As of 2020, 44% of websites use it
  - Significant departure; largely performance optimizations
- HTTP 3 forthcoming standard
  - Largely based on work by Google
  - As of 2020, 5% of websites use it (more or less Google and Facebook?)
  - Significant departure; largely performance optimizations

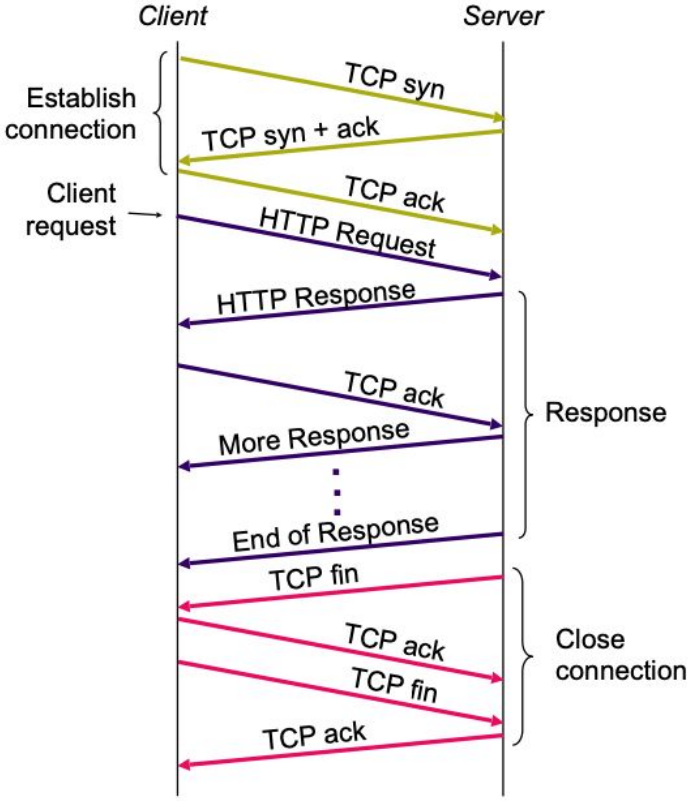
HyperText Transfer Protocol (HTTP)

- (Simple HTTP 1.0 “GET” request)

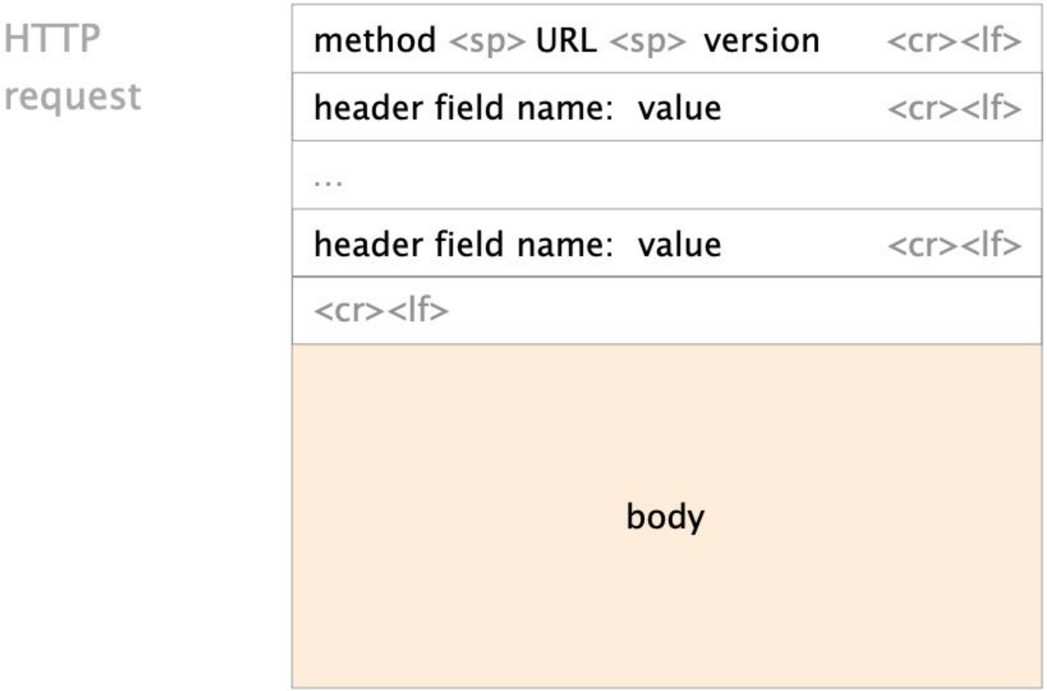
- Client creates TCP connection (port 80)
- Client sends request

- Server sends response packets
- Client ACKs them

- Server closes connection



HTTP client requests



HTTP request

method <sp> URL <sp> version <cr><lf>

header field name: value <cr><lf>

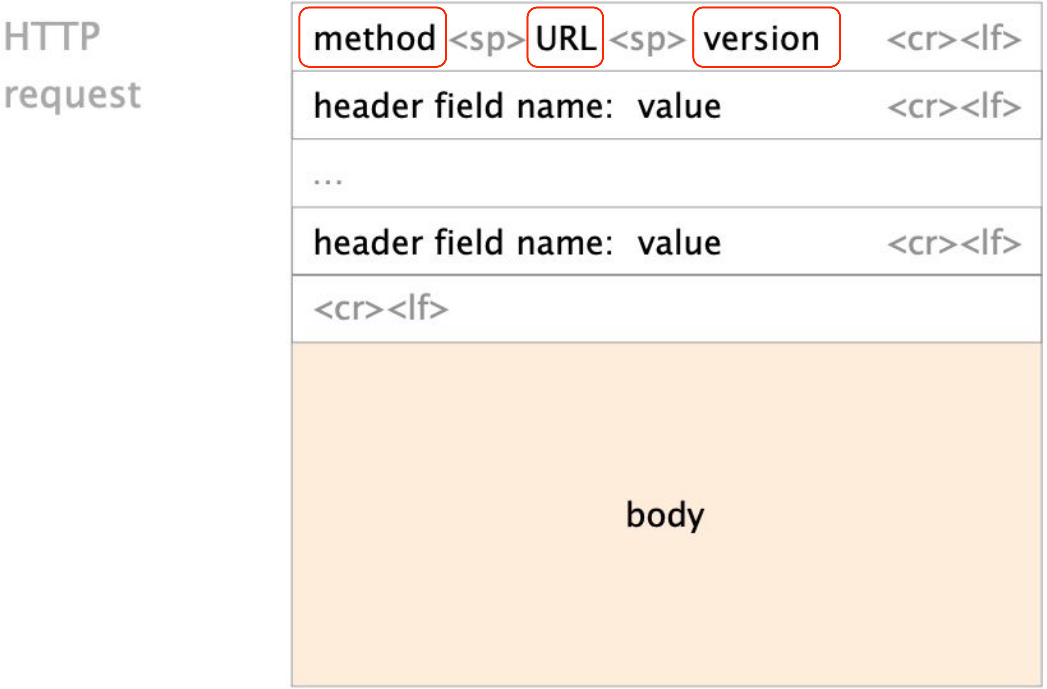
...

header field name: value <cr><lf>

<cr><lf>

body

HTTP client requests



HTTP request

method <sp> URL <sp> version <cr><lf>

header field name: value <cr><lf>

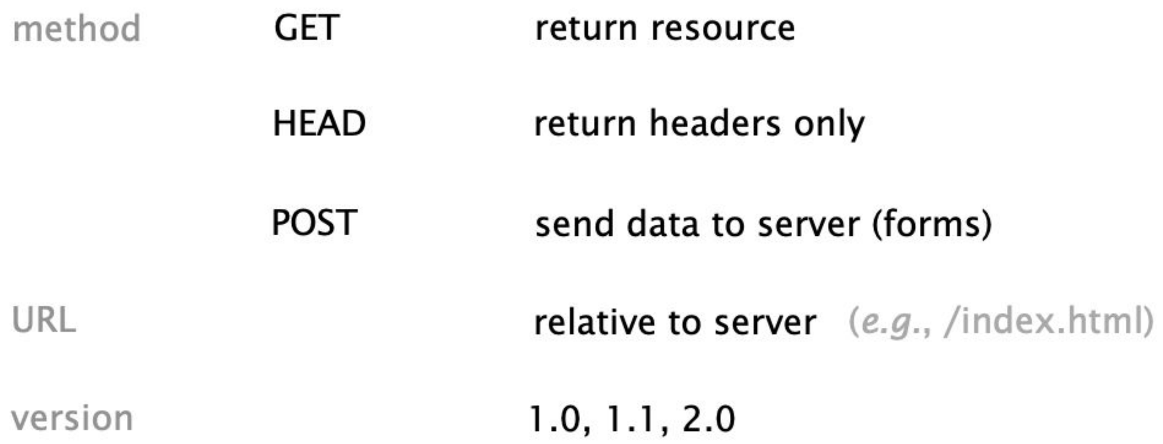
...

header field name: value <cr><lf>

<cr><lf>

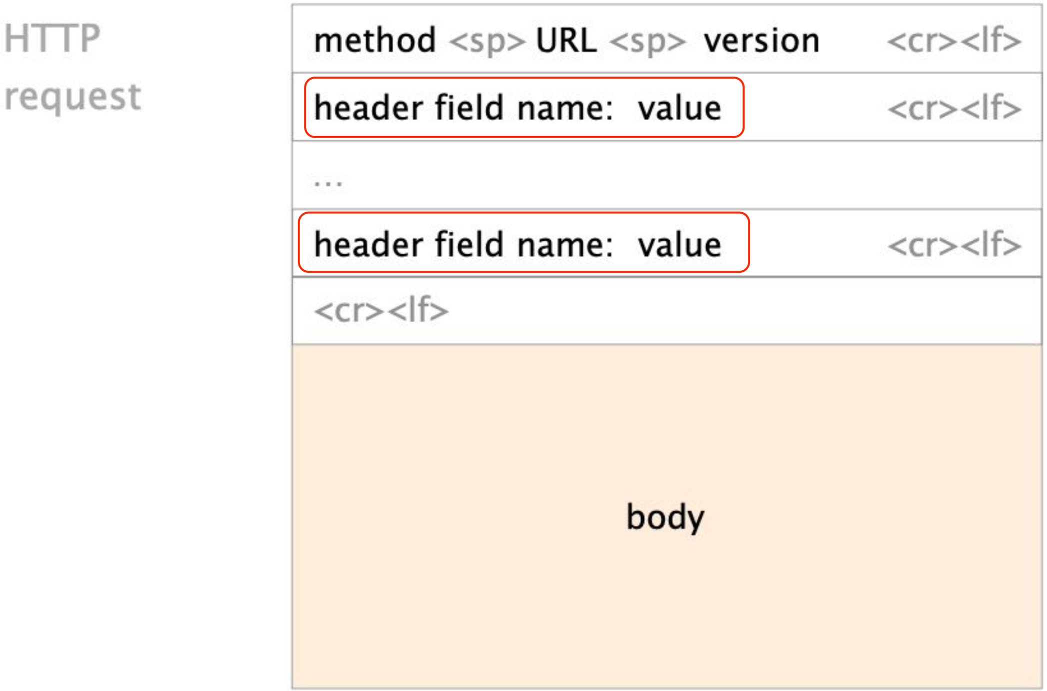
body

HTTP client requests



method	GET	return resource
	HEAD	return headers only
	POST	send data to server (forms)
URL		relative to server (*e.g.*, /index.html)
version		1.0, 1.1, 2.0

HTTP client requests



HTTP request

method <sp> URL <sp> version <cr><lf>

header field name: value <cr><lf>

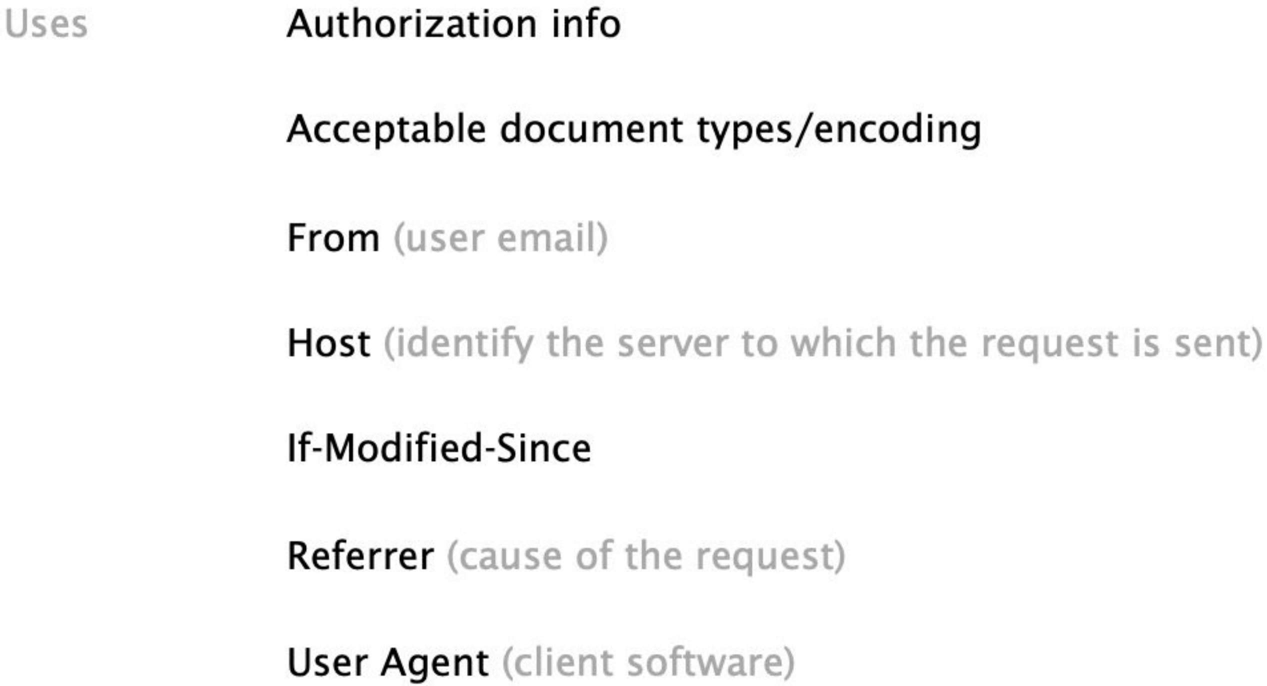
...

header field name: value <cr><lf>

<cr><lf>

body

Request headers are variable length but still human readable



Uses

Authorization info

Acceptable document types/encoding

From (user email)

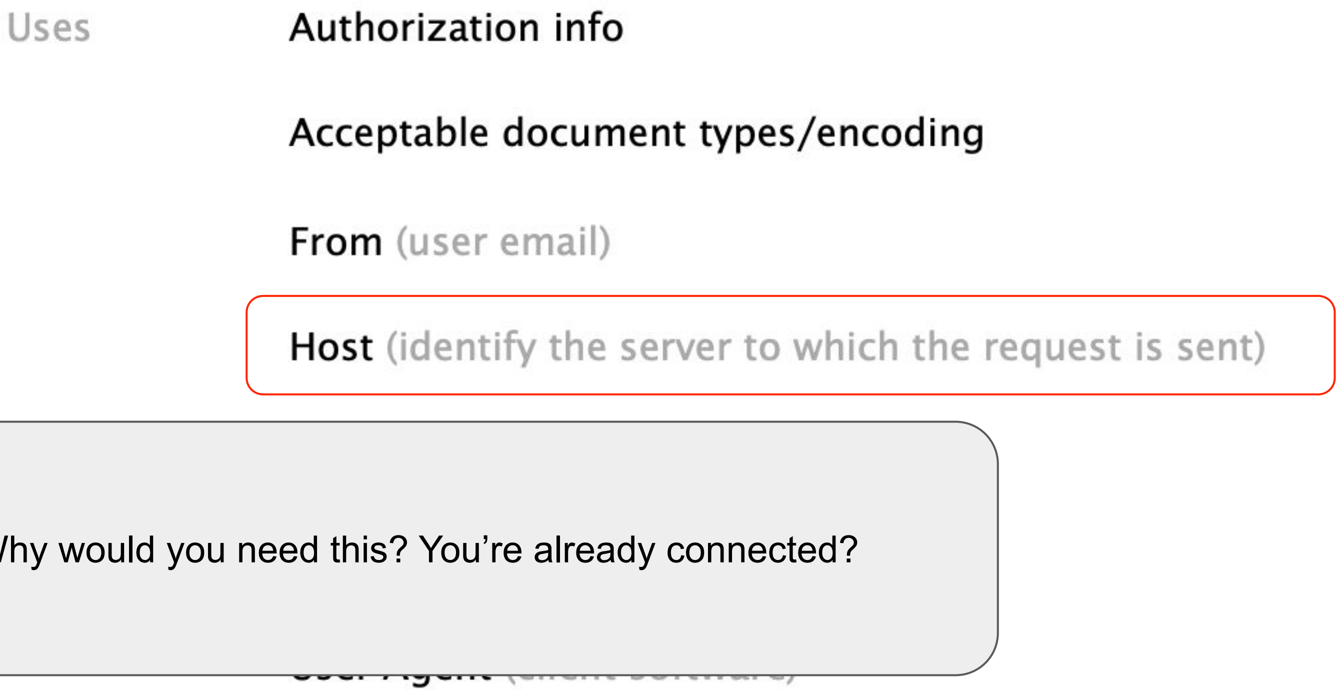
Host (identify the server to which the request is sent)

If-Modified-Since

Referrer (cause of the request)

User Agent (client software)

Request headers are variable length but still human readable



Uses

Authorization info

Acceptable document types/encoding

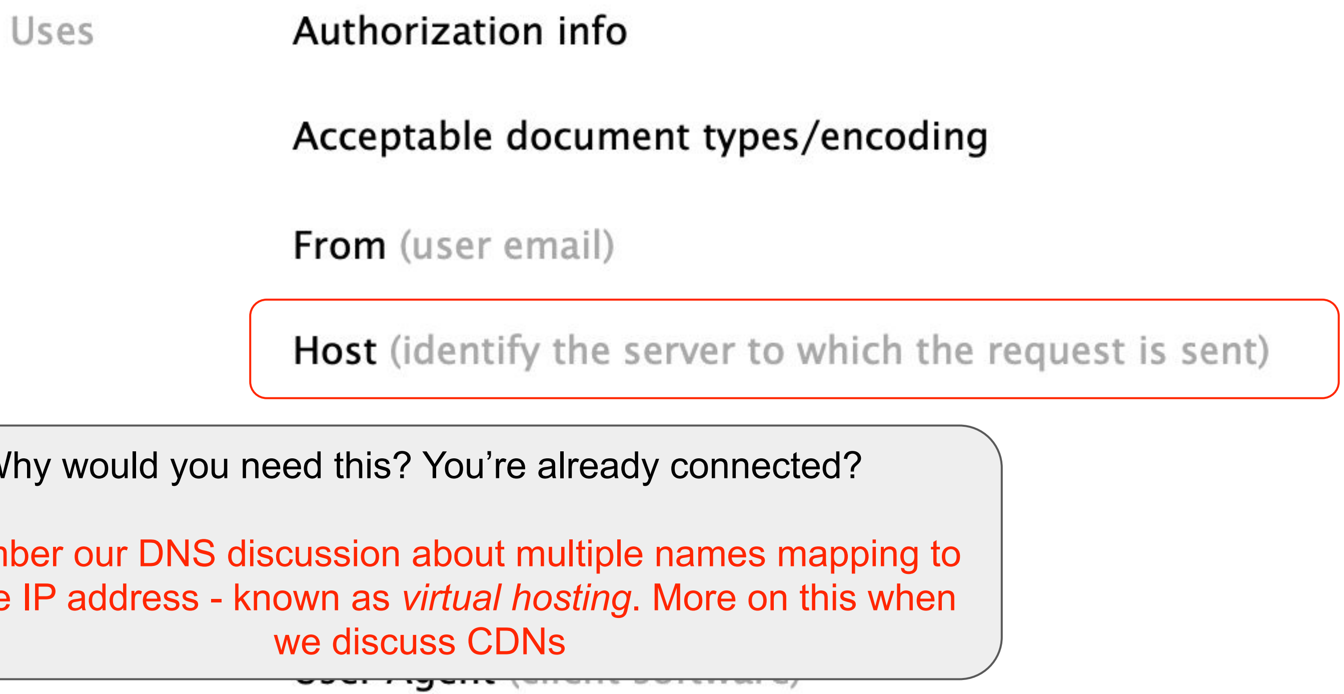
From (user email)

Host (identify the server to which the request is sent)

User-Agent (client software)

Why would you need this? You’re already connected?

Request headers are variable length but still human readable



Uses

Authorization info

Acceptable document types/encoding

From (user email)

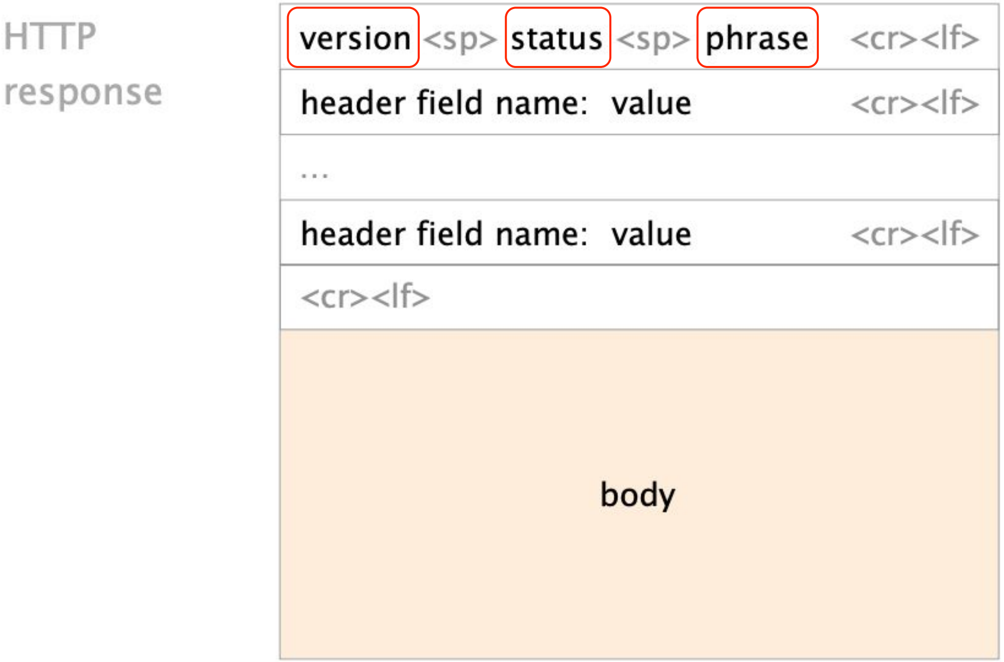
Host (identify the server to which the request is sent)

User-Agent (client software)

Why would you need this? You're already connected?

Remember our DNS discussion about multiple names mapping to a single IP address - known as *virtual hosting*. More on this when we discuss CDNs

HTTP server responses



HTTP response

version <sp> status <sp> phrase <cr><lf>

header field name: value <cr><lf>

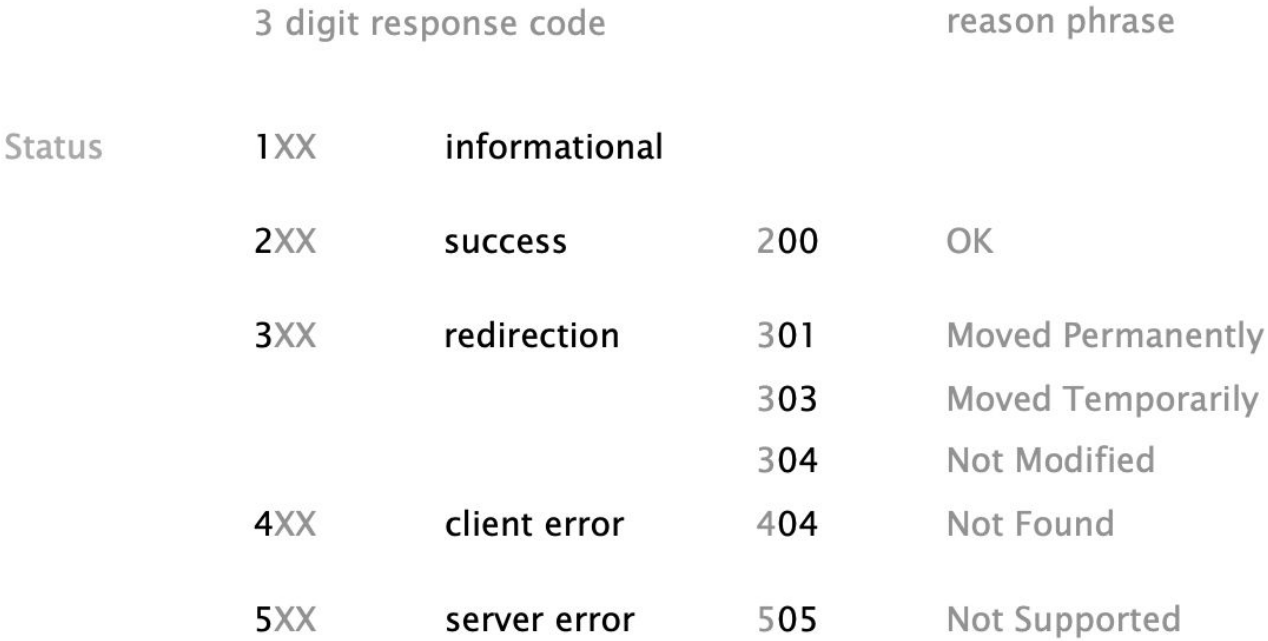
...

header field name: value <cr><lf>

<cr><lf>

body

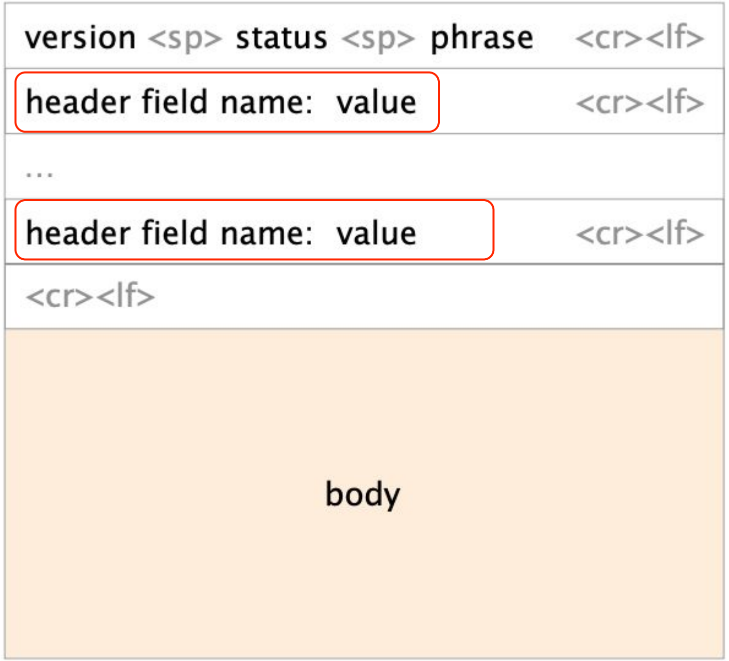
HTTP server responses



	3 digit response code			reason phrase
Status	1XX	informational		
	2XX	success	200	OK
	3XX	redirection	301	Moved Permanently
			303	Moved Temporarily
			304	Not Modified
	4XX	client error	404	Not Found
	5XX	server error	505	Not Supported

HTTP server responses

HTTP response



version <sp> status <sp> phrase <cr><lf>

header field name: value <cr><lf>

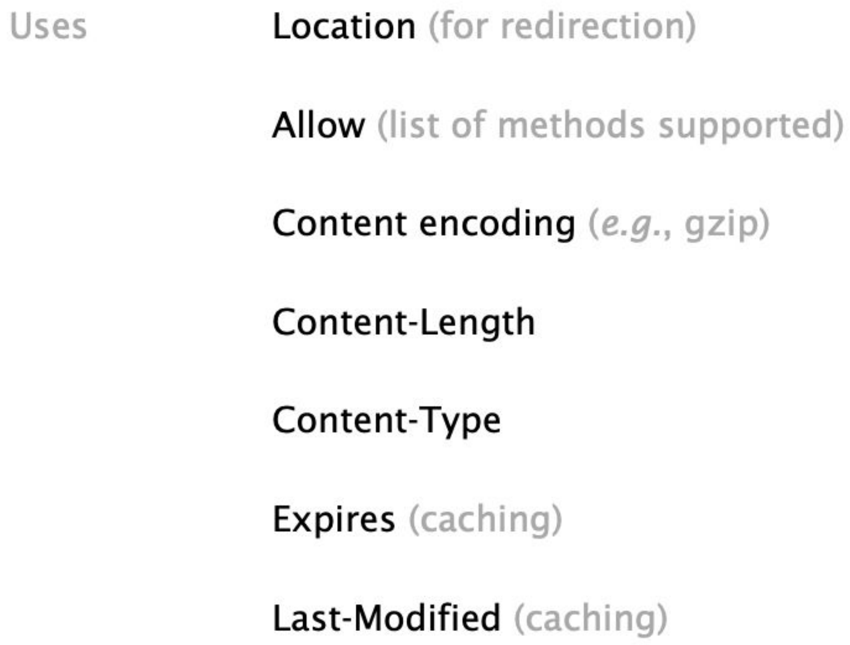
...

header field name: value <cr><lf>

<cr><lf>

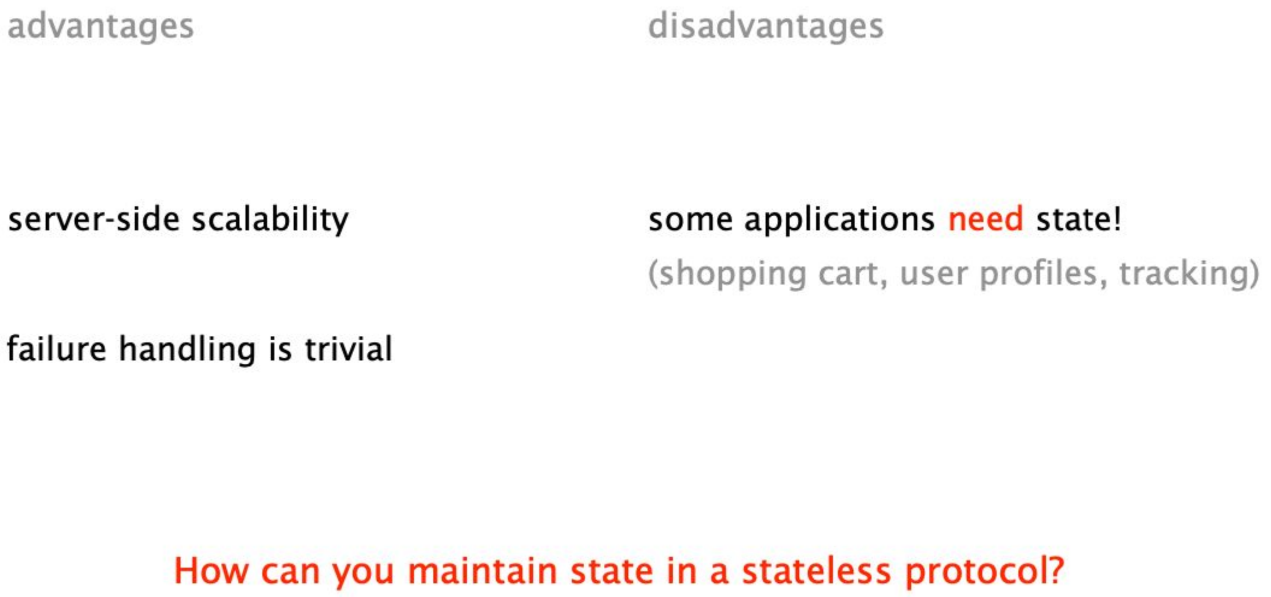
body

Like request headers, response headers are of variable lengths and human-readable



Uses	**Location** (for redirection)
	Allow (list of methods supported)
	Content encoding (*e.g.*, gzip)
	Content-Length
	Content-Type
	Expires (caching)
	Last-Modified (caching)

HTTP is a stateless protocol, meaning each request is treated independently



advantages

server-side scalability

failure handling is trivial

disadvantages

some applications need state!

(shopping cart, user profiles, tracking)

How can you maintain state in a stateless protocol?

HTTP makes the client maintain the state. This is what cookies are for

client stores small state

on behalf of the server *X*

client sends state

in all future requests to *X*

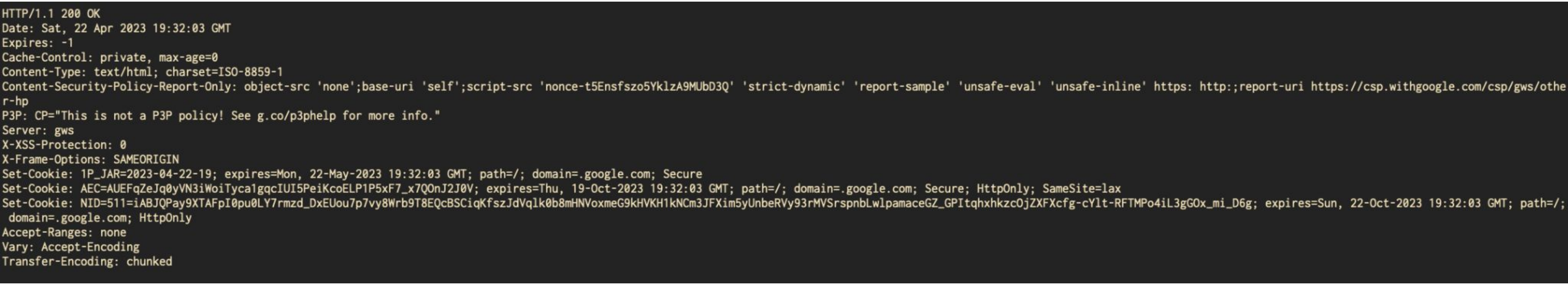
can provide authentication

Demo

telnet google.com 80

Request GET / HTTP/1.1

Host: www.google.com



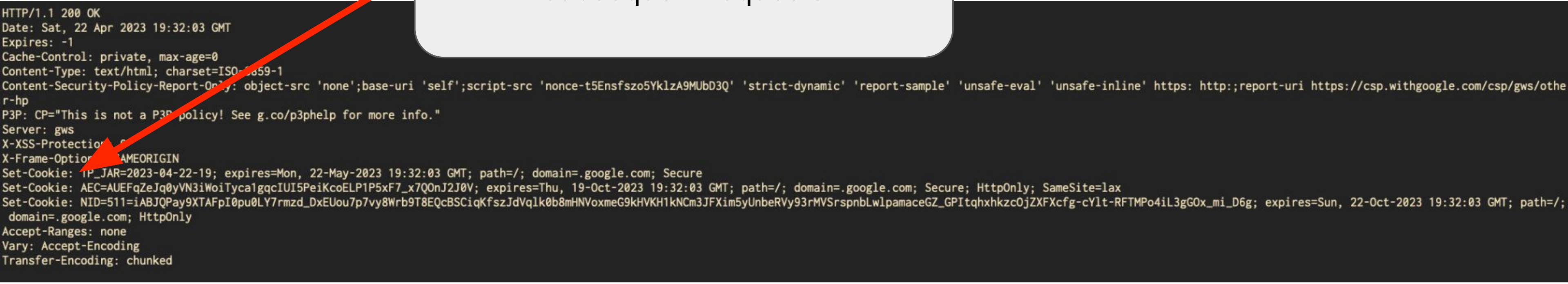
```
HTTP/1.1 200 OK
Date: Sat, 22 Apr 2023 19:32:03 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Content-Security-Policy-Report-Only: object-src 'none';base-uri 'self';script-src 'nonce-t5Ensfszo5YklzA9MUbD3Q' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https: http:;report-uri https://csp.withgoogle.com/csp/gws/othe
r-hp
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2023-04-22-19; expires=Mon, 22-May-2023 19:32:03 GMT; path=/; domain=.google.com; Secure
Set-Cookie: AEC=AUEFqZeJq0yVN3iWoiTyca1gqcIUI5PeiKcoELP1P5xF7_x7QOnJ2J0V; expires=Thu, 19-Oct-2023 19:32:03 GMT; path=/; domain=.google.com; Secure; HttpOnly; SameSite=lax
Set-Cookie: NID=511=iABJQPay9XTAFpI0pu0LY7rmzd_DxEUou7p7vy8Wrb9T8EQcBSCiqKfszJdVqlk0b8mHNVoxmeG9kHVKH1kNCm3JFXim5yUnbeRVy93rMVSrspnbLwlpamaceGZ_GPItqhxhkzcOjZXFXcfg-cYlt-RFTMPo4iL3gGOx_mi_D6g; expires=Sun, 22-Oct-2023 19:32:03 GMT; path=/;
 domain=.google.com; HttpOnly
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked
```

Demo

telnet google.com 80

Request GET / HTTP/1.1
Host:

Browser will relay this value in subsequent requests



```
HTTP/1.1 200 OK
Date: Sat, 22 Apr 2023 19:32:03 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Content-Security-Policy-Report-Only: object-src 'none';base-uri 'self';script-src 'nonce-t5Ensfszo5YklzA9MUbD3Q' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https: http:;report-uri https://csp.withgoogle.com/csp/gws/othe
r-hp
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2023-04-22-19; expires=Mon, 22-May-2023 19:32:03 GMT; path=/; domain=.google.com; Secure
Set-Cookie: AEC=AUEFqZeJq0yVN3iWoiTyca1gqcIUI5PeiKcoELP1P5xF7_x7QOnJ2J0V; expires=Thu, 19-Oct-2023 19:32:03 GMT; path=/; domain=.google.com; Secure; HttpOnly; SameSite=lax
Set-Cookie: NID=511=iABJQPay9XTAFpI0pu0LY7rmzd_DxEUou7p7vy8Wrb9T8EQcBSCiqKfszJdVqlk0b8mHNVoxmeG9kHVKH1kNCm3JFXim5yUnbeRVy93rMVSrspnbLwlpamaceGZ_GPItqhxhkzcOjZXFXcfg-cYlt-RFTMPo4iL3gGOx_mi_D6g; expires=Sun, 22-Oct-2023 19:32:03 GMT; path=/;
 domain=.google.com; HttpOnly
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked
```

What now? What about performance? Goals depend on who you're talking about

	User	Network operators	Content provider
wish	fast downloads high availability	no overload	happy users cost-effective infrastructure
solution	Improve HTTP to compensate for TCP weakspots	Caching and Replication	