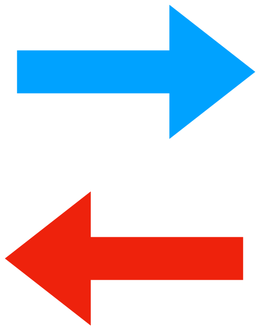
COM 1001

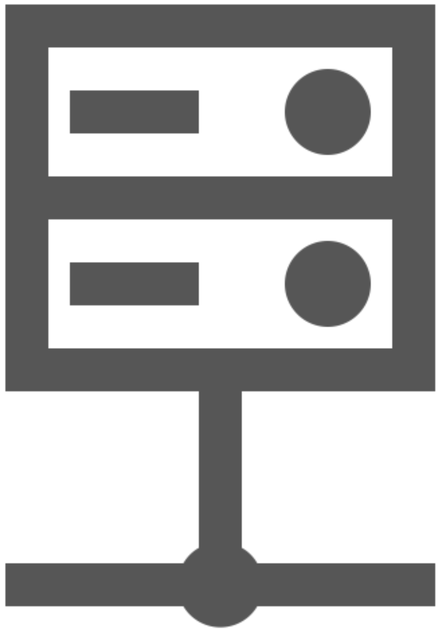
Introduction to Software Engineering

Professor Phil McMinn

The HyperText Transfer Protocol (HTTP)

What Happens When Your Browser Requests a Web Page?





Device with web browser

Server hosting web site
somewhere on the Internet
(web server)

The Path of Browser Request

1. The **browser** performs DNS Lookup for the **web server** based on its **domain name** (e.g. `www.sheffield.ac.uk`)
2. The **browser** sends an **HTTP Request** to the **web server**
3. The **web server** sends a **HTTP Response**, with the requested HTML file
4. The **browser** begins to render HTML
5. The **browser** sends additional requests for objects embedded in the HTML file (CSS files, images, JavaScript, etc.)

1 DNS Lookup

A **domain name** like “`www.sheffield.ac.uk`” is easy for humans to remember.

An **Internet Protocol (IP)** address like `143.167.2.102` is not, but this is how servers on the Internet are located.

So the first step in requesting a page is converting the domain to an IP address.

This information is managed by one of the 13 “root” **domain name servers (DNS)** that are located around the world.

However, if each browser in the world sent requests to these root servers every time they needed to request a web page, they'd be quickly overwhelmed!

The browser therefore consults several caches in the following order:

1. Its **own** cache
2. The cache on the browser's **operating system**
3. The cache of the **router** on which the browser's machine lives
4. The **ISP**'s DNS cache

Finally, your ISP gets its DNS information from one of the root servers.

2 Browser sends HTTP Request

Once it has the IP address of the web server, a browser can now send that server a **HTTP Request**.

It doesn't have to literally be **HTTP** – it could be **HTTPS**, or **HTTP/2**, or even **HTTP/3**.

These later versions of the HTTP protocol following HTTP/1.1 do not really have any implications for web applications, they just make HTTP…

… more secure (HTTPS)

… faster (HTTP/2 and HTTP/3)

```
GET /dcs HTTP/1.1
Host: www.sheffield.ac.uk
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.16; rv:84.0) Gecko/20100101 Firefox/84.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Cookie: _ga_global=GA1.3.34348778.1604339527; _ga=GA1.3.34348778.1604339527;
_fbp=fb.2.1604339527540.412012585; _hjid=0582874f-bc89-43b5-b8a7-7b7043e7fc80;
_ga_global_gid=GA1.3.279550765.1610359615; _ga_studrec=GA1.3.34348778.1604339527;
_ga_studrec_gid=GA1.3.521051581.1610359615; _gid=GA1.3.768165880.1610359615;
_hjIncludedInSessionSample=1; _hjTLDTest=1; _hjAbsoluteSessionInProgress=0; _gat_UA-2341502-28=1
Upgrade-Insecure-Requests: 1
```

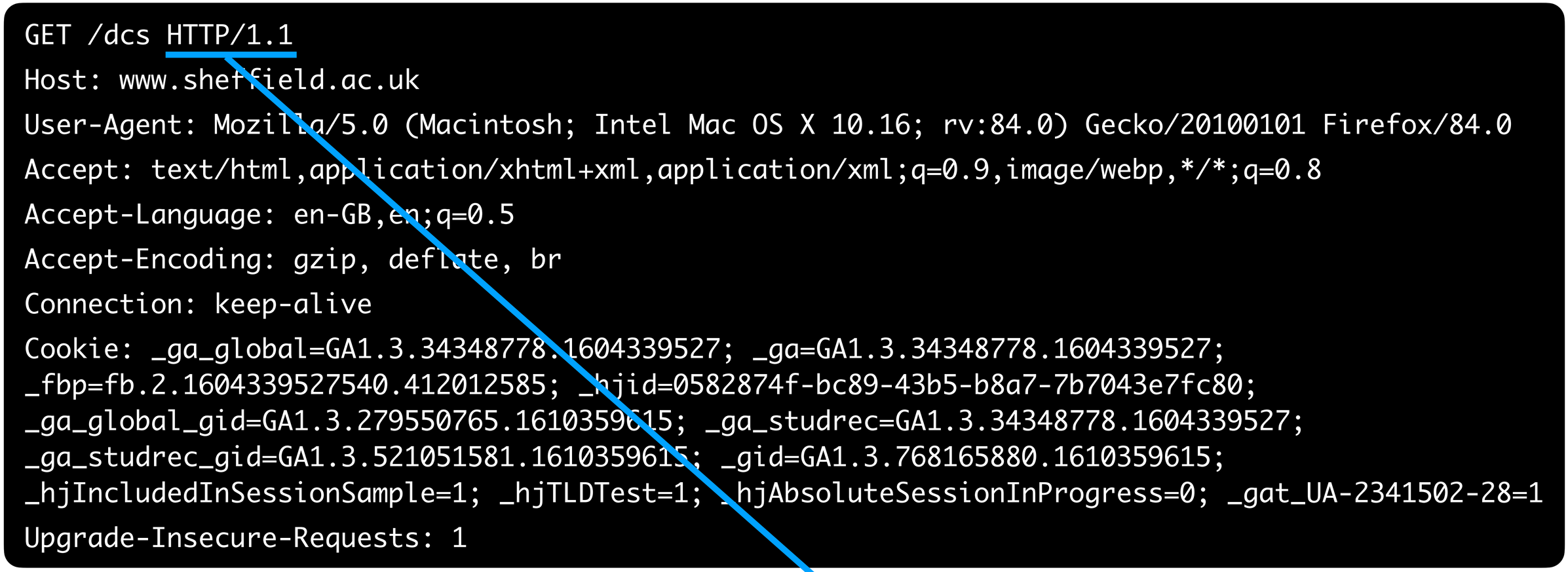
The first part of the first line of a **HTTP Request** is the HTTP **method**.

The **method** defines the type of request being made and therefore how the server will interpret it. The most important **methods** are GET and POST.

More on methods later in the module.

```
GET /dcs HTTP/1.1
Host: www.sheffield.ac.uk
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.16; rv:84.0) Gecko/20100101 Firefox/84.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Cookie: _ga_global=GA1.3.34348778.1604339527; _ga=GA1.3.34348778.1604339527;
_fbp=fb.2.1604339527540.412012585; _hjid=0582874f-bc89-43b5-b8a7-7b7043e7fc80;
_ga_global_gid=GA1.3.279550765.1610359615; _ga_studrec=GA1.3.34348778.1604339527;
_ga_studrec_gid=GA1.3.521051581.1610359615; _gid=GA1.3.768165880.1610359615;
_hjIncludedInSessionSample=1; _hjTLDTest=1; _hjAbsoluteSessionInProgress=0; _gat_UA-2341502-28=1
Upgrade-Insecure-Requests: 1
```

Secondly, we have the **resource identifier** of the **resource** being requested, which could be a web page, or any type of file (such an image, script, or document).



The resource identifier is followed by the **HTTP protocol** being used.

```
GET /dcs HTTP/1.1
Host: www.sheffield.ac.uk
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.16; rv:84.0) Gecko/20100101 Firefox/84.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Cookie: _ga_global=GA1.3.34348778.1604339527; _ga=GA1.3.34348778.1604339527;
_fbp=fb.2.1604339527540.412012585; _hjid=0582874f-bc89-43b5-b8a7-7b7043e7fc80;
_ga_global_gid=GA1.3.279550765.1610359615; _ga_studrec=GA1.3.34348778.1604339527;
_ga_studrec_gid=GA1.3.521051581.1610359615; _gid=GA1.3.768165880.1610359615;
_hjIncludedInSessionSample=1; _hjTLDTest=1; _hjAbsoluteSessionInProgress=0; _gat_UA-2341502-28=1
Upgrade-Insecure-Requests: 1
```

Finally, the **HTTP Request** contains a series of **Request Headers**

Most of these are not particularly important for this module.

```
GET /dcs HTTP/1.1
Host: www.sheffield.ac.uk
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.16; rv:84.0) Gecko/20100101 Firefox/84.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Cookie: _ga_global=GA1.3.34348778.1604339527; _ga=GA1.3.34348778.1604339527;
_fbp=fb.2.1604339527540.412012585; _hjid=0582874f-bc89-43b5-b8a7-7b7043e7fc80;
_ga_global_gid=GA1.3.279550765.1610359615; _ga_studrec=GA1.3.34348778.1604339527;
_ga_studrec_gid=GA1.3.521051581.1610359615; _gid=GA1.3.768165880.1610359615;
_hjIncludedInSessionSample=1; _hjTLDTest=1; _hjAbsoluteSessionInProgress=0; _gat_UA-2341502-28=1
Upgrade-Insecure-Requests: 1
```

Finally, the **HTTP Request** contains a series of **Request Headers**

Most of these are not particularly important for this module.

Note the `Cookie` header, however – we will be returning to the topic of **cookies** in a later lecture.

3 Web Server sends HTTP Response

Once the server receives a **HTTP Request**, it will respond in the form of a **HTTP Response**. How it does depends on a number of things.

For example, the resource (i.e., a web page) may not actually exist.

The response consists of a series of headers (like the request did), and a body, which contains the resource requested.

Here's the initial part of the **HTTP response** Sheffield University's web server sent for the **HTTP request** for `http://www.sheffield.ac.uk/dcs`

```
HTTP/1.1 200 OK
X-Drupal-Dynamic-Cache: MISS
X-Varnish: 341153 402883
X-Grace: none
Vary: Accept-Encoding,X-Forwarded-Host
Cache-Control: max-age=900, public
X-Cache: HIT (drupal-cache-live1)
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Strict-Transport-Security: max-age=31536000; preload
Link: <https://www.sheffield.ac.uk/dcs>; rel="canonical"
…
```

The most important part of the response headers is the **status code**.
Ideally it sends a `200 OK`, which means success.

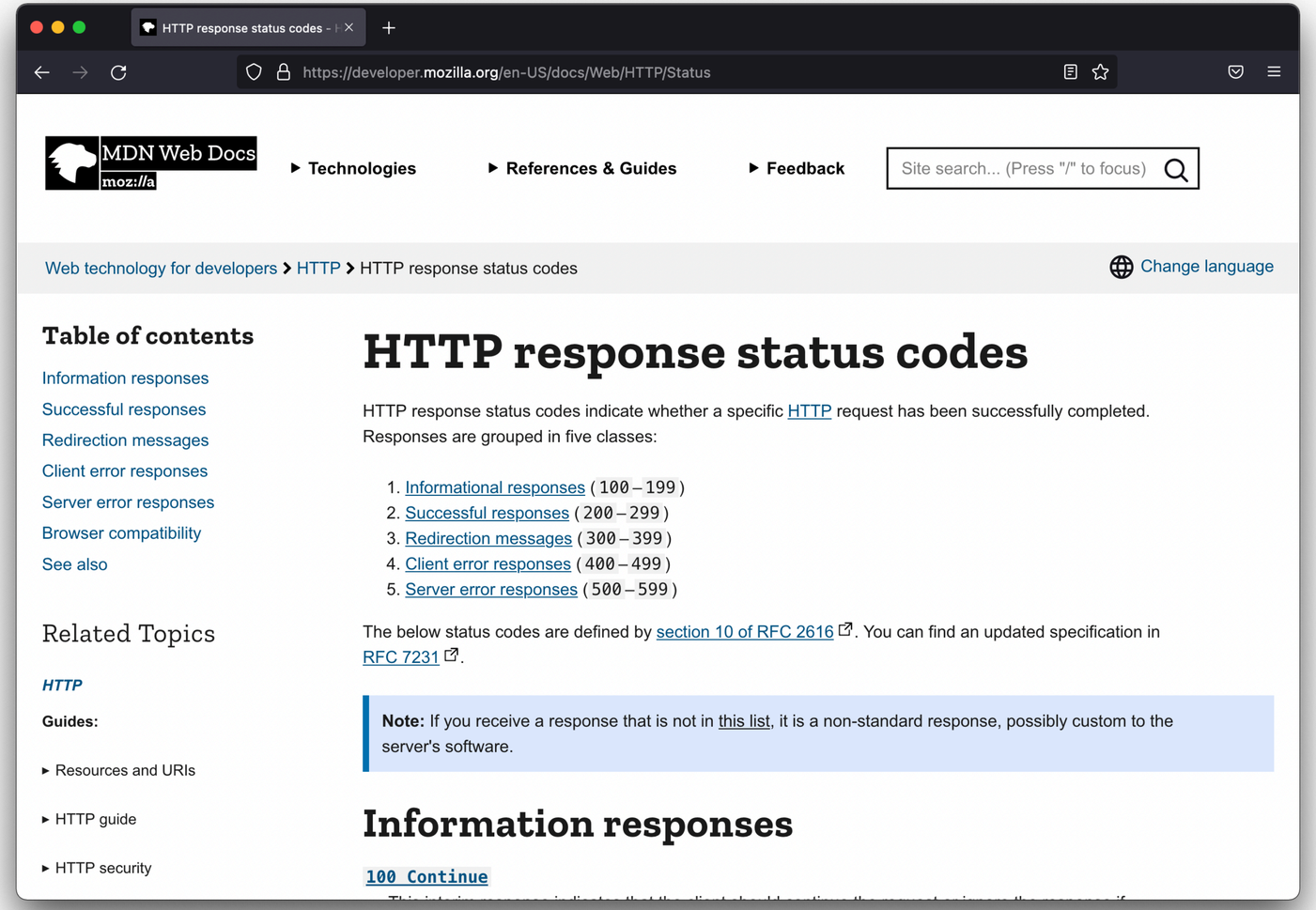
But it may send a `404 Not Found` or a `500 Internal Server Error`.

Both of these mean the requested resource cannot be sent.

```
HTTP/1.1 200 OK
X-Drupal-Dynamic-Cache: MISS
X-Varnish: 341153 402883
X-Grace: none
Vary: Accept-Encoding,X-Forwarded-Host
Cache-Control: max-age=900, public
X-Cache: HIT (drupal-cache-live1)
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Strict-Transport-Security: max-age=31536000; preload
Link: <https://www.sheffield.ac.uk/dcs>; rel="canonical"
…
```

More on Status Codes

developer.mozilla.org/en-US/docs/Web/HTTP/Status



More on Status Codes

developer.mozilla.org/en-US/docs/Web/HTTP/Status/418

418 I'm a teapot

The HTTP `418 I'm a teapot` client error response code indicates that the server refuses to brew coffee because it is, permanently, a teapot. A combined coffee/tea pot that is temporarily out of coffee should instead return 503. This error is a reference to Hyper Text Coffee Pot Control Protocol defined in April Fools' jokes in 1998 and 2014.

Some websites use this response for requests they do not wish to handle, such as automated queries.

Status

```
418 I'm a teapot
```

The Body of the HTTP Response
Static vs Dynamic Content Generation

Depending on the nature of the resource, there may be more work for the web server to do in generating the body of the HTTP response.

If the resource is a **static file** (e.g., an image), the server just needs to locate it, and encode it into the response.

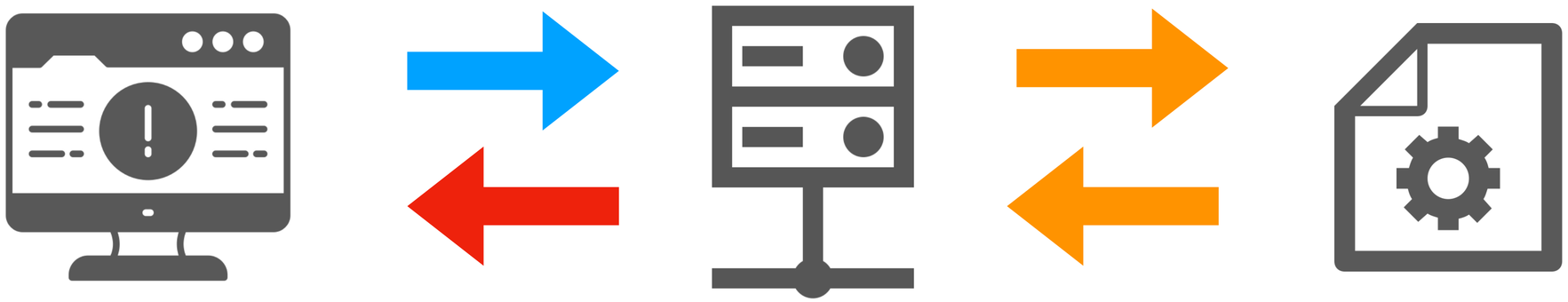
If the resource is **dynamically generated**, the **server needs to run a program** to generate the content.

Dynamic Content Generation

If we can dynamically generate content, the web pages can respond to user actions and change the information appearing in the web page. This information could come from a **database**, for example.

In other words, we can write **web applications**.

In this module, we will be dynamically generating web content using **Ruby** and the help of a framework called **Sinatra**.



browser

web server

Content generated using
Ruby + Sinatra

4 The Browser Renders the HTML

Once the browser receives an HTML file, as part of the body of the HTTP Response, it can then process it and render it onto the screen.

5 The Browser Sends Additional Requests

During processing of the HTML file, the browser may find that it needs to request additional files (e.g., images and scripts, etc.) and will send additional HTTP Requests for those.