# Computer Networks

## 5. Application layer (HTTP)

Boni García

http://bonigarcia.github.io/

boni.garcia@urjc.es

Departamento de Teoría de la Señal y Comunicaciones y Sistemas Telemáticos y Computación
Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad Rey Juan Carlos

2019/2020

Escuela Técnica Superior
de Ingeniería de Telecomunicación

## Table of contents

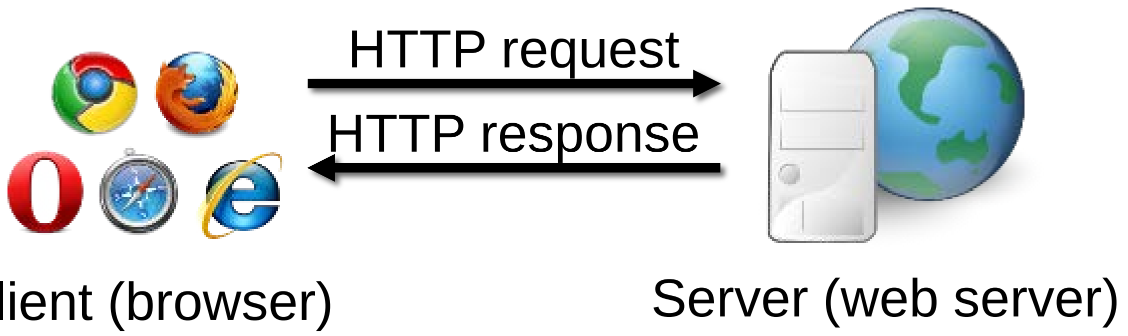
## Table of contents

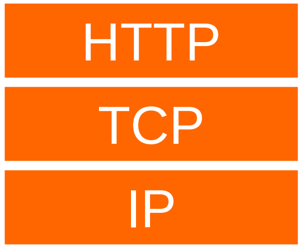
## 1. Introduction - What is the Web?

- The Web (World Wide Web) is a service for the distribution of hypertext content accessible via the Internet
- As we already know, the Internet is a decentralized set of interconnected networks that use the TCP/IP protocol stack interconnecting hundreds of millions of hosts around the world
- Therefore, we should not confuse the Web with the Internet
- Web pages are documents written in HTML (Hypertext Markup Language) and interconnected through links (links)

## 1. Introduction - What is the Web?

- The Web is based on the client-server model
- The application-level protocol for communicating clients and servers on the Web is **HTTP** (Hypertext Transfer Protocol)
- Web resources are identified with a unique name called web address or **URL** (Uniform Resource Locator)
- Default port for web servers: 80





## 1. Introduction - What is the Web?

- A web server sends the files it has stored on its hard drive to HTTP clients requesting it
- Any type of file can be used, although the usual are the files that a browser recognizes (html, jpg, png, pdf ...)
- When it receives a request, it returns the file of the hard disk that fits the route indicated in the URL

$$\underbrace{\text{http}}_{\text{Schema (protocol)}}\text{:}\underbrace{\text{//www.myserver.com}}_{\text{Server (host name with or IP address)}}\underbrace{\text{:port}}_{\text{Server port}}\underbrace{\text{/file/path}}_{\text{Path}}\underbrace{\text{/file.txt}}_{\text{Resource}}\underbrace{\text{?key=value}}_{\text{Query}}\underbrace{\text{\#fragment}}_{\text{Anchor}}$$

Depending on the server, the URLs may be case sensitive. For example:
https://en.wikipedia.org/wiki/Iron_Maiden -- https://en.wikipedia.org/wiki/Iron_maiden

## 1. Introduction - What is the Web?

- Most used browsers:
  - Google Chrome
  - Firefox
  - Internet Explorer → Edge
  - Safari
  - Opera
- Most used web servers:
  - Apache
  - Internet Information Server (IIS)

## 1. Introduction - HTML

- Hypertext Markup Language (HTML) is the standard markup language for creating web pages
- Web browsers receive HTML documents from a web server or local storage and render the documents into multimedia web pages
- HTML elements are the building blocks of HTML pages
  - Images, links, paragraphs, tables, forms, etc. may be embedded into a we page
  - HTML elements are delineated by tags, written using angle brackets, for example:
    - `<a href="https://urjc.es/">Link to URJC</a>`
    - `<img src="my-picture.png">`
- The current version of HTML is 5

# 1. Introduction - HTML

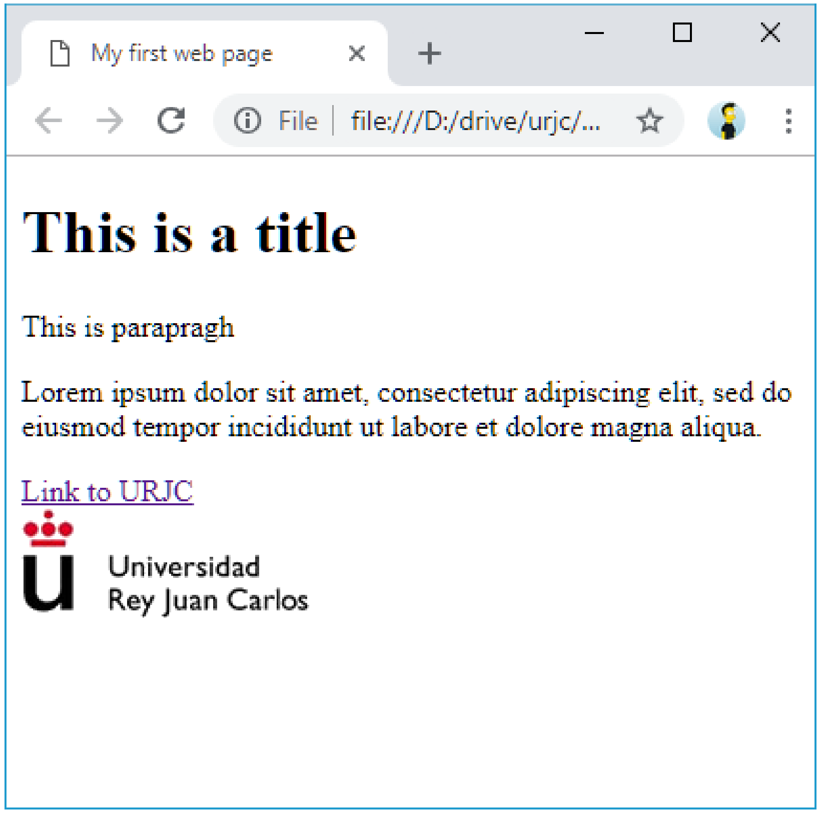
- Simple example of a web page:

```html
<!DOCTYPE html>
<html>
<head>
<title>My first web page</title>
</head>
<body>
    <h1>This is a title</h1>

    <p>This is parapragh</p>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing
elit, sed do eiusmod tempor incididunt ut labore et
dolore magna aliqua.</p>

    <a href="https://urjc.es/">Link to URJC</a>
    <br>

    <!-- This is a picture (with absolute path) -->
    <img src="https://www.urjc.es/images/Logos/logo-urjc-
negro.png">
    </a>
</body>
</html>
```



# This is a title

This is parapragh

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

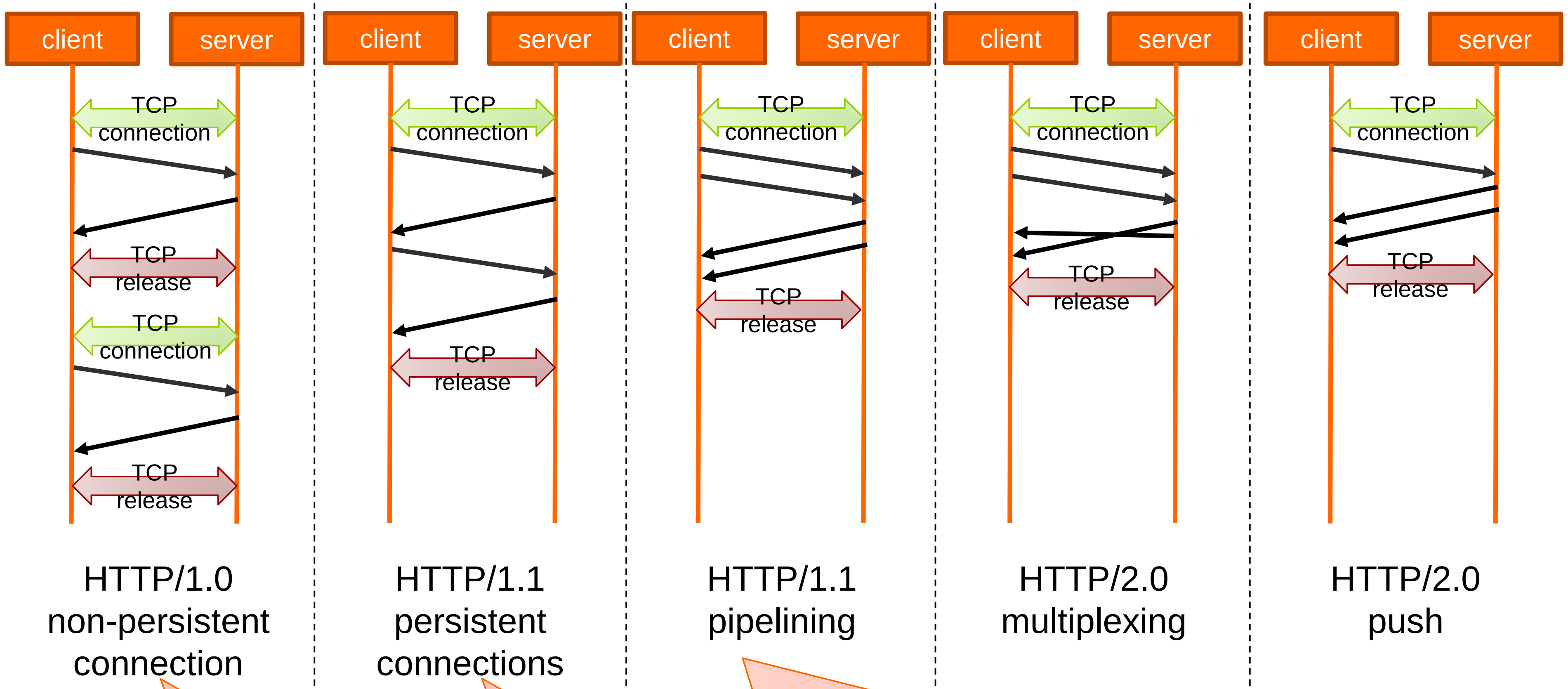
Link to URJC

Universidad
Rey Juan Carlos

## 1. Introduction - HTTP versions

- HTTP/0.9: Original version. Currently obsolete
- HTTP/1.0 (RFC 1945): Old version but still used
  - The connections are non-persistent (multiple TCP connections will be used, one for each requested object)
- HTTP/1.1 (RFC 2616): Current version
  - The connections are persistent (the server keeps a TCP connection open so that the following requests and responses are transmitted through that connection)
  - Allows successive requests (pipelining), i.e. making several requests to the server without waiting for the response
- HTTP/2.0 (RFC 7540): Supported in browsers since 2015
  - The answers can be processed asynchronously (multiplexing)
  - The server can send resources to the client before there is a request (push)

## 1. Introduction - HTTP versions



By default in HTTP 1.0

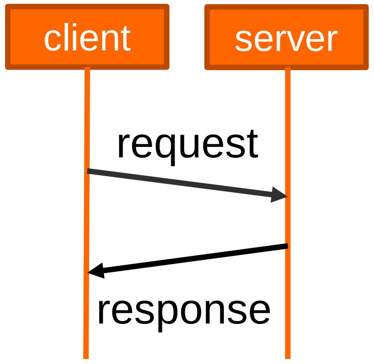
By default in HTTP 1.1

As of 2018, HTTP pipelining is not enabled by default in modern browsers, due to several issues (multiplexing is used instead)

## Table of contents

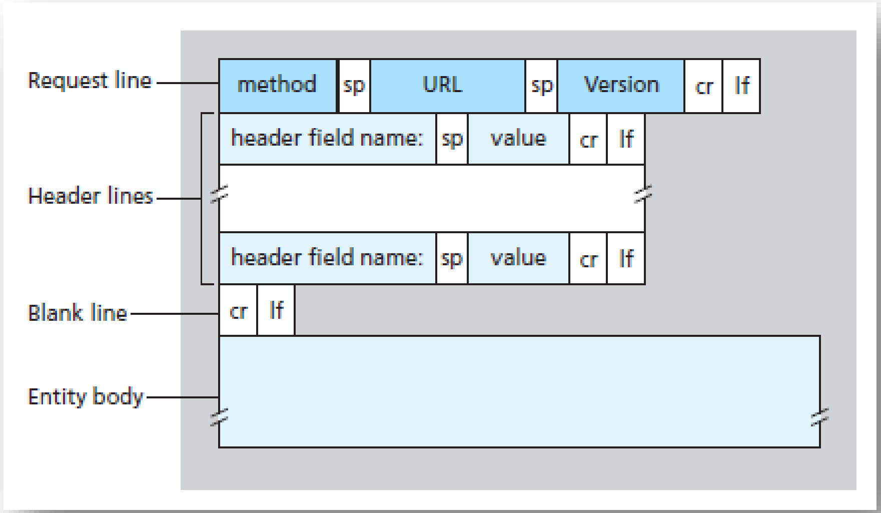
## 2. HTTP 1.1 - Messages

- There are two types of HTTP messages:
  - **Request**: from client to server
  - **Response**: from server to client (in reply to a request)



## 2. HTTP 1.1 - Messages

- The format of **requests** is as follows:



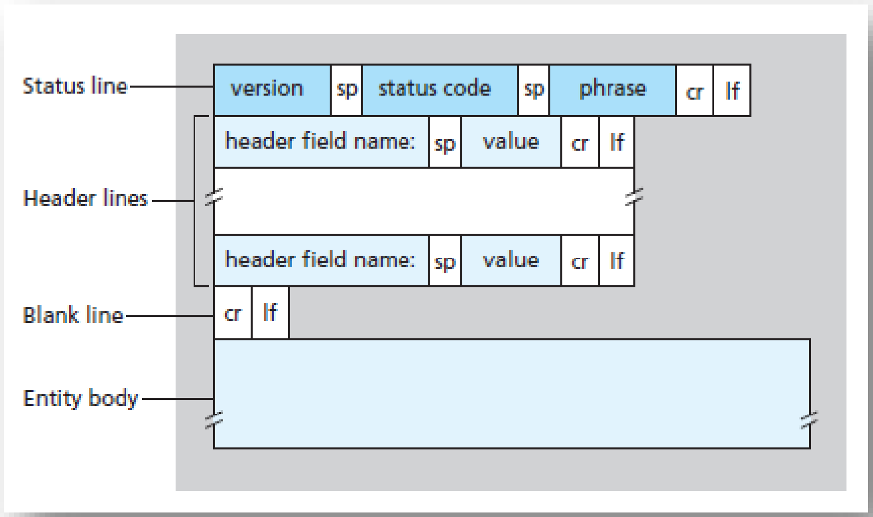
For example:

```
GET /index.html  HTTP/1.1
Host: www.example.com
User-Agent:  Mozilla/4.0
Accept: text/html
```

sp=space
cr=carriage return
lf=line feed

## 2. HTTP 1.1 - Messages

- The format of **responses** is as follows:



sp=space
cr=carriage return
lf=line feed

For example:

```
HTTP/1.1 200 OK
Date: Fri, 29 Nov 2019 11:30:00 GMT
Server: Apache/2.0.54 (Fedora)
Content-Type: text/html
Content-Length: 119

<!DOCTYPE html>
<html>
<head>
<title>My web page</title>
</head>
<body>
    <p>Hello world!</p>
</body>
</html>
```

## 2. HTTP 1.1 - Request methods

- HTTP **methods** (sometimes referred to as "verbs") indicate the action that clients take on the web resource
- HTTP 1.1 (RFC 2616) defines 8 methods:

GET POST PUT DELETE HEAD TRACE OPTIONS CONNECT

- There is an extension to HTTP 1.1 (RFC 5789) that defines a new method:

PATCH

## 2. HTTP 1.1 - Request methods

Used in REST services

- **`GET`**: Reads of a specific resource (URL)
- **`POST`**: Sends data that is processed by a resource (URL)

Main methods

- `PUT`: Create a resource
- `DELETE`: Delete a resource

Methods normally disabled on web servers

- `PATCH`: Requests the server to partially modify a resource

- `HEAD`: Request an identical response to the one that would correspond to a GET request, but without the body of the response. This is useful to know the headers of the answer but without transporting all the content
- `TRACE`: Requests the server to send back a response message with the request sent (echo service). It is used for checking and diagnostic purposes
- `OPTIONS`: Requests the server for the supported methods for a given resource. The answer is obtained in the `Allow` header
- `CONNECT`: Used to tell a web proxy to establish a secure connection (TLS) with a remote client

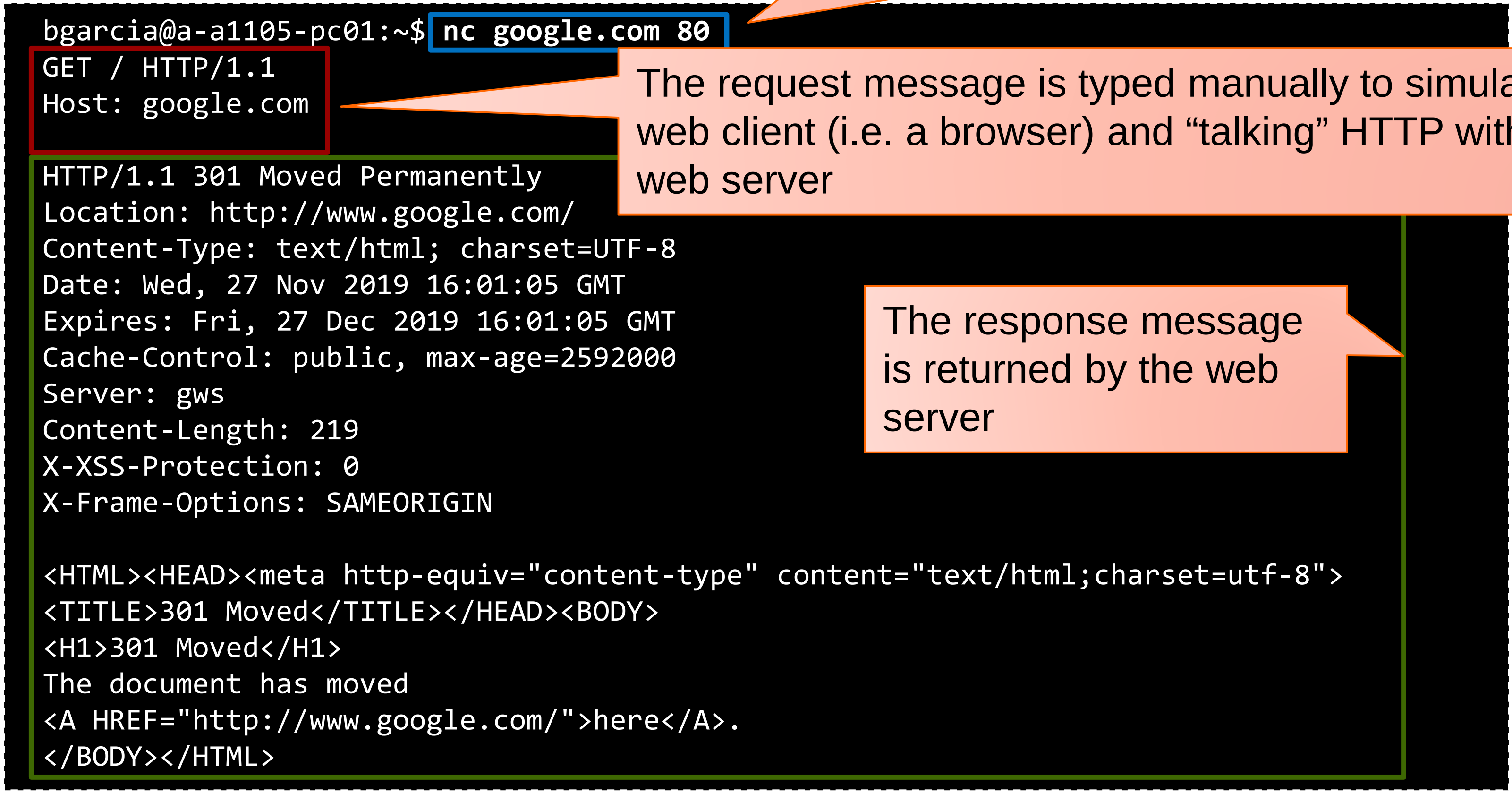
## 2. HTTP 1.1 - Response codes

- **1xx** Informative response. For example:
  - `101 Protocol Change`
- **2xx** Successful operation. For example:
  - `200 OK`
- **3xx** Redirection. For example:
  - `301 Moved Permanently`
  - `304 Not Modified`
- **4xx** Error on the part of the client. For example:
  - `401 Unauthorized`
  - `404 Not Found`
- **5xx** Server error. For example:
  - `500 Internal Error`

https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

# 2. HTTP 1.1 - Manual example

In this example we use the Linux command nc to open a TCP connection to the google.com web server, which is listening to port 80



```
bgarcia@a-a1105-pc01:~$ nc google.com 80
GET / HTTP/1.1
Host: google.com

HTTP/1.1 301 Moved Permanently
Location: http://www.google.com/
Content-Type: text/html; charset=UTF-8
Date: Wed, 27 Nov 2019 16:01:05 GMT
Expires: Fri, 27 Dec 2019 16:01:05 GMT
Cache-Control: public, max-age=2592000
Server: gws
Content-Length: 219
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
```

The request message is typed manually to simulate a web client (i.e. a browser) and “talking” HTTP with the web server

The response message is returned by the web server

## 2. HTTP 1.1 - Manual example

```
bgarcia@a-a1105-pc01:~$ nc google.com 80
GET / HTTP/1.1
Host: www.google.com

HTTP/1.1 200 OK
Date: Wed, 27 Nov 2019 15:58:06 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
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=2019-11-27-15; expires=Fri, 27-Dec-2019 15:58:06 GMT; path=/;
domain=.google.com
Set-Cookie: NID=192=wtYxfr-G98yu532khddxODSmJ9xs9Dxa6a-
S9gHLmGOUCTvTVOifAR9CNhuC_TWEvgHGdF3M2pTDIPHLakdOzOshsgKHETrehAgFZxP88kidGh-
O1zMEmxvCbGyVrA2NzyZ06FxOk3sC1Gs2OMCPLtVXi_DTxyHNOnch6NgqRfY; expires=Thu, 28-May-2020
15:58:06 GMT; path=/; domain=.google.com; HttpOnly
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked

<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="es">
...
</html>
```

*mandatory header

## 2. HTTP 1.1 - Basic headers

- Some of the most common headers in **requests**:

| Header | Description | Example |
|---|---|---|
| `Host`* | Server host name | `Host: en.wikipedia.org` |
| `User-Agent` | Client identification | `User-Agent: Mozilla/5.0 …` |
| `Accept` | MIME type | `Accept: text/plain` |
| `Accept-Charset` | Character encoding | `Accept-Charset: utf-8` |

- Some of the most common headers in **responses**:

| Header | Description | Example |
|---|---|---|
| `Date` | Date in which the response was made | `Date: Tue, 15 Nov 1994 08:12:31 GMT` |
| `Server` | Server identification | `Server: Apache/2.4.1 (Unix)` |
| `Content-Type` | MIME type (and optionally, encoding) | `text/html; charset=UTF-8` |
| `Content-Length` | Body size (in bytes) | `Content-Length: 348` |

## 2. HTTP 1.1 - MIME types

- MIME (Multipurpose Internet Mail Extensions) indicates the type of content of a message

| MIME type | Typical file extension(s) | Description |
|---|---|---|
| `text/plain` | `.txt` | Plain text |
| `text/html` | `.html .htm` | Web page |
| `image/jpeg` | `.jpg .jpeg` | JPEG image |
| `image/gif` | `.gif` | GIF image |
| `image/png` | `.png` | PNG image |
| `application/pdf` | `.pdf` | PDF file |
| `audio/mpeg3` | `.mp3` | Audio in MP3 format |
| `video/mpeg` | `.mpg .mpeg` | Video in MPEG format |

https://www.iana.org/assignments/media-types/media-types.xhtml

## 2. HTTP 1.1 - Connection types

- By default, connections are **non-persistent** in HTTP/1.0 and **persistent** in HTTP/1.1
- This behavior can be changed using the header `Connection`:
  - Persistent connection (`Connection: Keep-Alive`).

```
GET / HTTP/1.1
Host: www.example.com
```

```
GET / HTTP/1.0
Host: www.example.com
Connection: Keep-Alive
```

  - Non-persistent (`Connection: Close`)

```
GET / HTTP/1.1
Host: www.example.com
Connection: Close
```

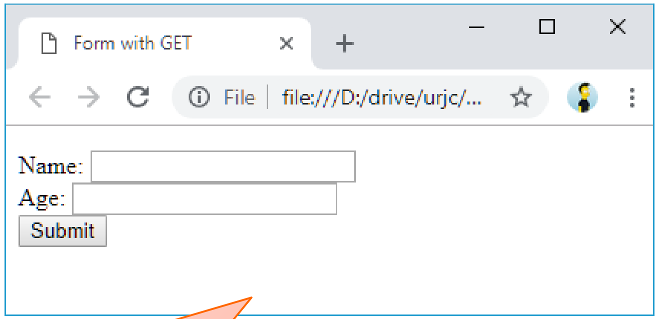
```
GET / HTTP/1.0
Host: www.example.com
```

## 2. HTTP 1.1 - HTML forms

- A common way to send data from client to server is using **HTML forms**:

Web page with form in HTML using **GET**

```
<!DOCTYPE html>
<html>
<head>
<title>Form with GET</title>
</head>
<body>
    <form action="http://pc2.emp2.net/form.php" method="get">
        <p>
            Name: <input type="text" name="name"><br>
            Age: <input type="text" name="age"><br>
            <input type="submit">
        </p>
    </form>
</body>
</html>
```



Rendered web page in a web browser

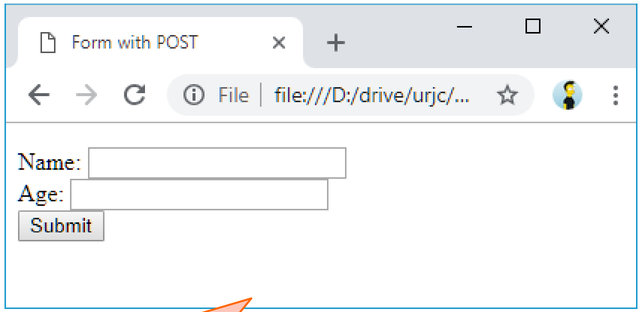
When clicking on the button "Submit", browser will send a HTTP request to the server with the following message

```
GET /form.php?name=John+Smith&age=24 HTTP/1.1
Host: pc2.emp2.net
User-Agent: Mozilla/4.5 [en]
Accept: image/jpeg, image/gif, text/html
Accept-language: en
Accept-Charset: iso-8859-1
```

## 2. HTTP 1.1 - HTML forms

```
<!DOCTYPE html>
<html>
<head>
<title>Form with POST</title>
</head>
<body>
    <form action="http://pc2.emp2.net/form.php" method="post">
        <p>
            Name: <input type="text" name="name"><br>
            Age: <input type="text" name="age"><br>
            <input type="submit">
        </p>
    </form>
</body>
</html>
```

Web page with form in HTML using **POST**



Rendered web page in a web browser

```
POST /form.php HTTP/1.1
Host: pc2.emp2.net
User-Agent: Mozilla/4.5 [en]
Accept: image/jpeg, image/gif, text/html
Accept-language: en
Accept-Charset: iso-8859-1
Content-Type: application/x-www-form-urlencoded
Content-Length: 23

name=John+Smith&age=24
```

When clicking on the button “Submit”, browser will send a HTTP request to the server with the following message

## 2. HTTP 1.1 - Cookies

- HTTP is a stateless protocol, i.e. it does not save any information about clients
- To save information about the status of clients, HTTP implements the **cookies** mechanism
- The technology of cookies is implemented using a couple of HTTP headers:
  - **`Set-cookie`** header in responses (from server to client)
  - **`Cookie`** header on requests (from client to server)
    - There are a couple of obsolete headers regarding to cookies: `Cookie2` and `Set-Cookie2`
- Information of cookies is stored both in client and server:
  - Cookie file stored in browser
  - Cookies database in web server

## 2. HTTP 1.1 - Cookies

- **`Set-cookie`** example:

```
Set-Cookie: status=deleted; Path=/; Domain=mysite.com; Expires=Wed, 31-Dec-2030
23:59:59 GMT; Secure
```

  - `status=deleted` → Cookie **name** (`status`) and **value** (`deleted`). This field is **mandatory** in Set-Cookie
  - `Path=/` → Client must sent the cookie back for further requests **starting** with this path (this value is a **prefix path**)
  - `Domain=mysite.com` → Client must sent the cookie back for further requests to the **same site** (if not present, it will be the target server host name or IP address)
  - `Expires=Wed, 31-Dec-2030 23:59:59 GMT` → Client must sent the cookie back **until** this date
    - If the date is expired, the cookies is deleted in the client
    - If not present, the cookie is not persistent as a file (stored in memory)
  - `Secure` → Cookie can only be sent using HTTPS (TLS)

## 2. HTTP 1.1 - Cookies

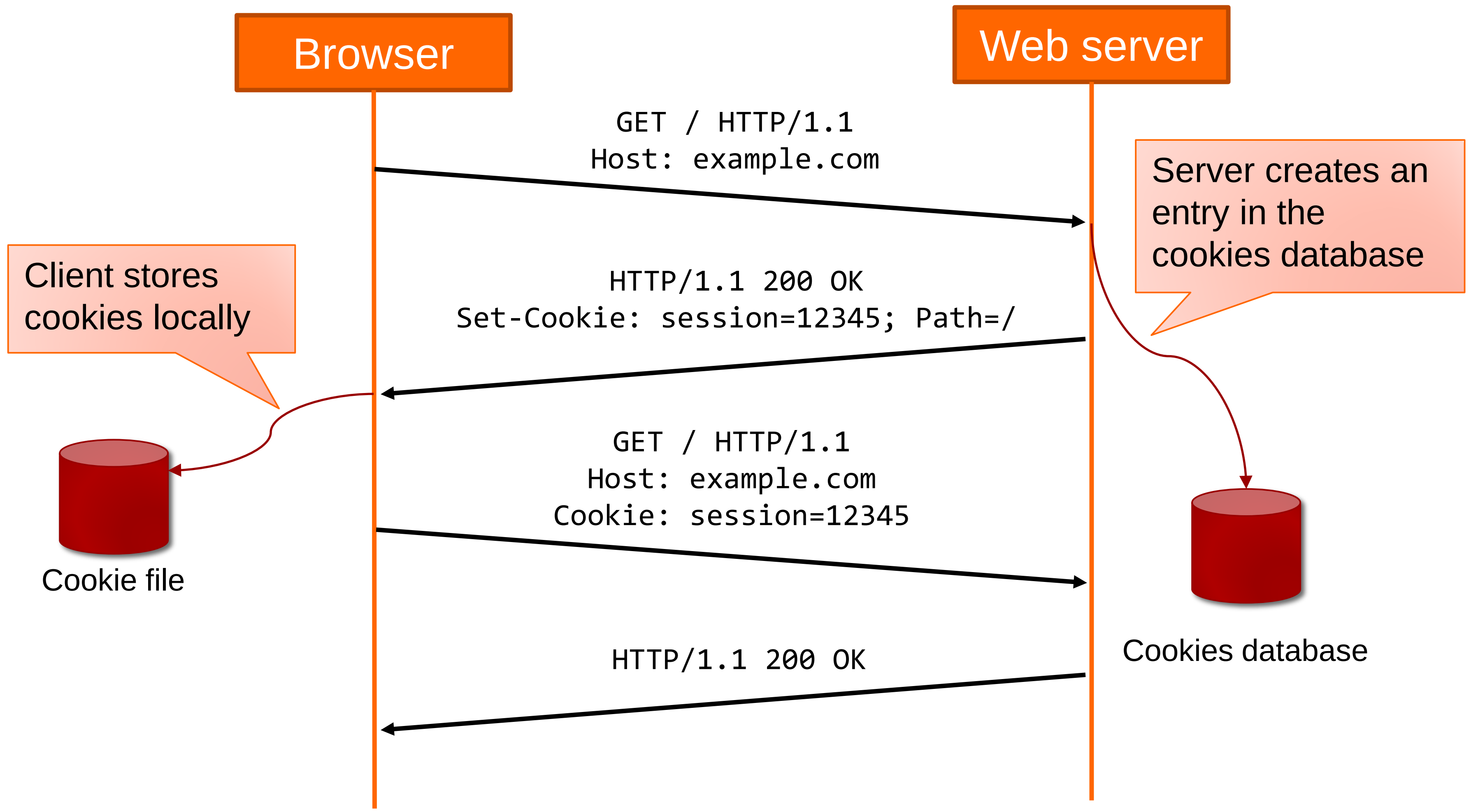
- **`Cookie`** example:

```
Cookie: status=deleted; user=john; edited=false
```

  - Previously received pairs of `name=value` separated by `;` are sent by client to server
  - Stored cookies are sent from client to server when:
    1. Cookie not expired (`Expires`)
    2. Requests has same domain (`Domain`)
    3. URL stars with the same prefix (`Path`)

## 2. HTTP 1.1 - Cookies

- Example:


Browser
Web server
Server creates an entry in the cookies database
Client stores cookies locally
Cookie file
Cookies database

```
GET / HTTP/1.1
Host: example.com
```

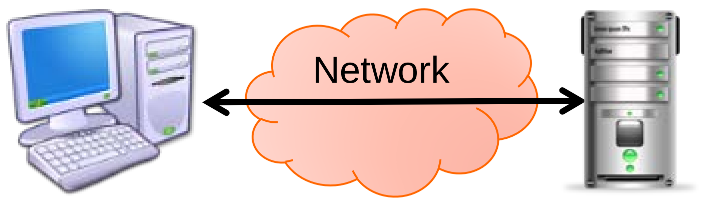
```
HTTP/1.1 200 OK
Set-Cookie: session=12345; Path=/
```

```
GET / HTTP/1.1
Host: example.com
Cookie: session=12345
```

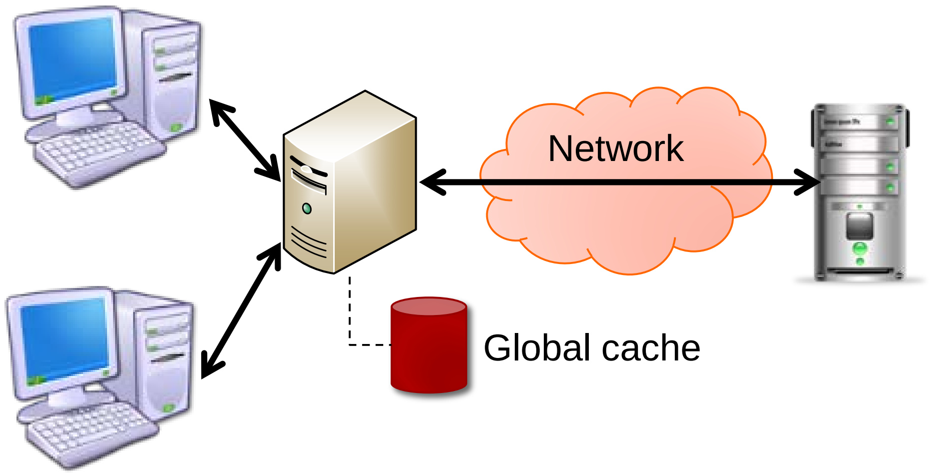
```
HTTP/1.1 200 OK
```

## 2. HTTP 1.1 - Web proxy

- In networking, a **proxy** is a server that intercepts network connections made from client to servers
  - Motivation for the use of proxies: performance improvement (cache), monitoring, filtering
- A **web proxy** is a type of proxy that intercepts HTTP traffic

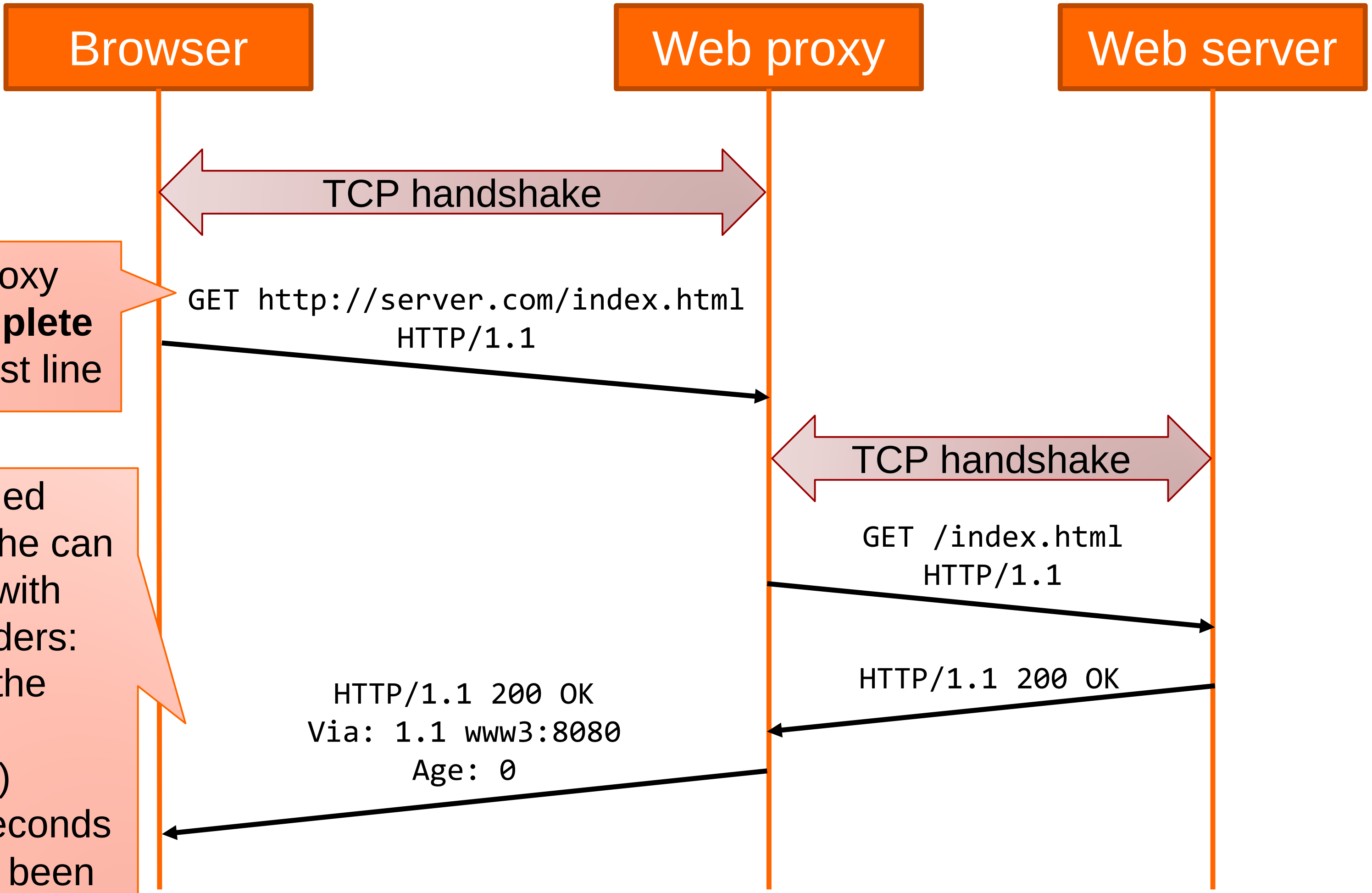


Without proxy



With proxy

# 2. HTTP 1.1 - Web proxy



```
GET http://server.com/index.html
HTTP/1.1
```

```
GET /index.html
HTTP/1.1
```

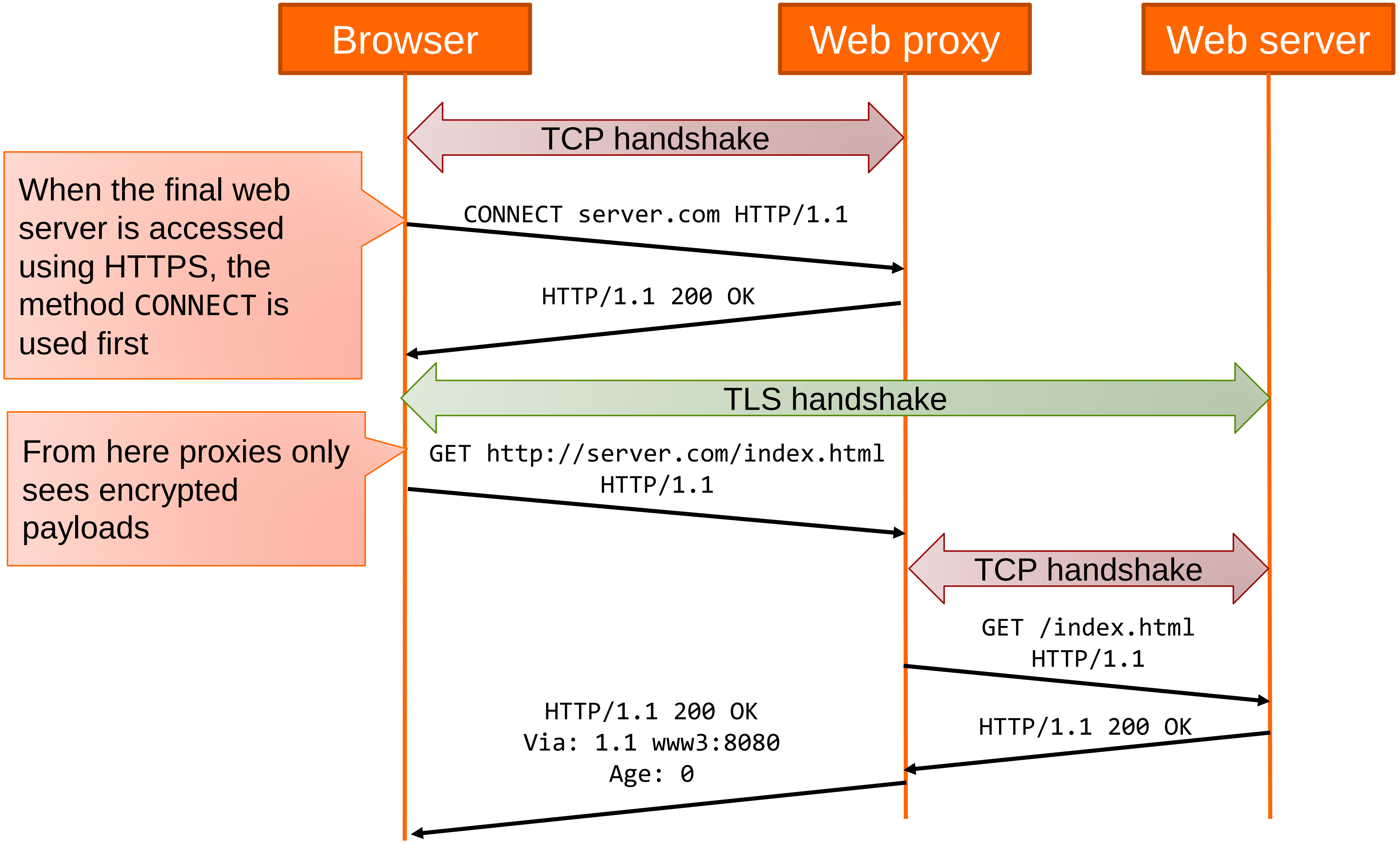
```
HTTP/1.1 200 OK
Via: 1.1 www3:8080
Age: 0
```

Request using proxy includes the **complete URL** in the request line

Resources obtained from a global cache can be sent together with the following headers:

- **`Via`**: name of the proxy (in this example `www3`)
- **`Age`**: time in seconds the object has been in a proxy cache (in this example `0`)

## 2. HTTP 1.1 - Web proxy

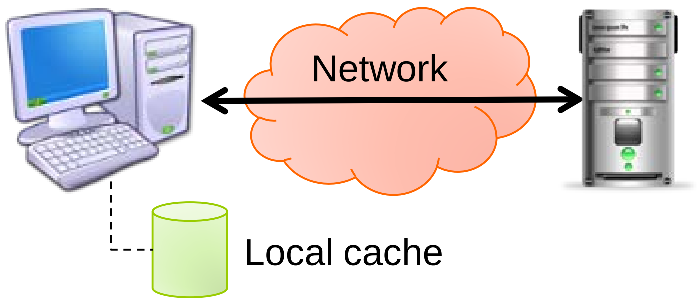


When the final web server is accessed using HTTPS, the method `CONNECT` is used first

From here proxies only sees encrypted payloads

## 2. HTTP 1.1 - Web cache

- In addition to global caches (in proxies), browsers also maintains another cache for web resources (known as **local cache**)
- The objective is to improve performance and reduce latency by avoiding HTTP transfers of resources that have not changed



## 2. HTTP 1.1 - Web cache

- Resource are stored in a cache together with an **expiration time**
  - Before this expiration time, the resource is **fresh** and can be used without requesting again (this called *cache hint*)
  - After the expiration time, the resource is **stale**, and cannot be used again without revalidation (forwarding the request to check if it is in fact still fresh)
    - Revalidation can also be triggered when the user presses the reload button
- Caches have finite storage so items are periodically removed. This process is called *cache eviction*

## 2. HTTP 1.1 - Web cache

- The **freshness** (in seconds) of a resource is calculated based on several headers:
  1. If the header `Cache-control: max-age=N` is specified:
     ```
     freshness = N
     ```
  2. If this header is not present, the headers `Expires` and the `Date` are used:
     ```
     freshness = value(Expires) – value(Date)
     ```
  3. If neither header is present, the headers `Last-Modified` and the `Date` are used:
     ```
     freshness = [value(Date) – value(Last-Modified)] / 10
     ```
- Finally, the **expiration time** of a resource is computed as follows:
  ```
  expirationTime = responseTime + freshness - currentAge
  ```

## 2. HTTP 1.1 - Web cache

- **`Cache-Control`** is a general-header used to specify directives for caching mechanisms (in both requests and responses)
- The main values of `Cache-Control` in **responses** are the following (can be concatenated with `,`):
  - `public`: resource can be cached anywhere
  - `private`: resource can be cached only in private caches (option by default)
  - `max-age=X`: valid time in seconds
  - `no-store`: resource cannot be cached
  - `no-cache`: a resource is cached but needs to be revalidated always (equivalent to `max-age=0`)
  - `must-revalidate`: equivalent to `no-cache` but servers are supposed to send the error code 504 if revalidation is not possible

```
Cache-Control: private, max-age=86400
```

Example

## 2. HTTP 1.1 - Web cache

- The header **`Expires`** (introduced in HTTP 1.0) is used to establish the absolute time in which a resource is valid. For example:

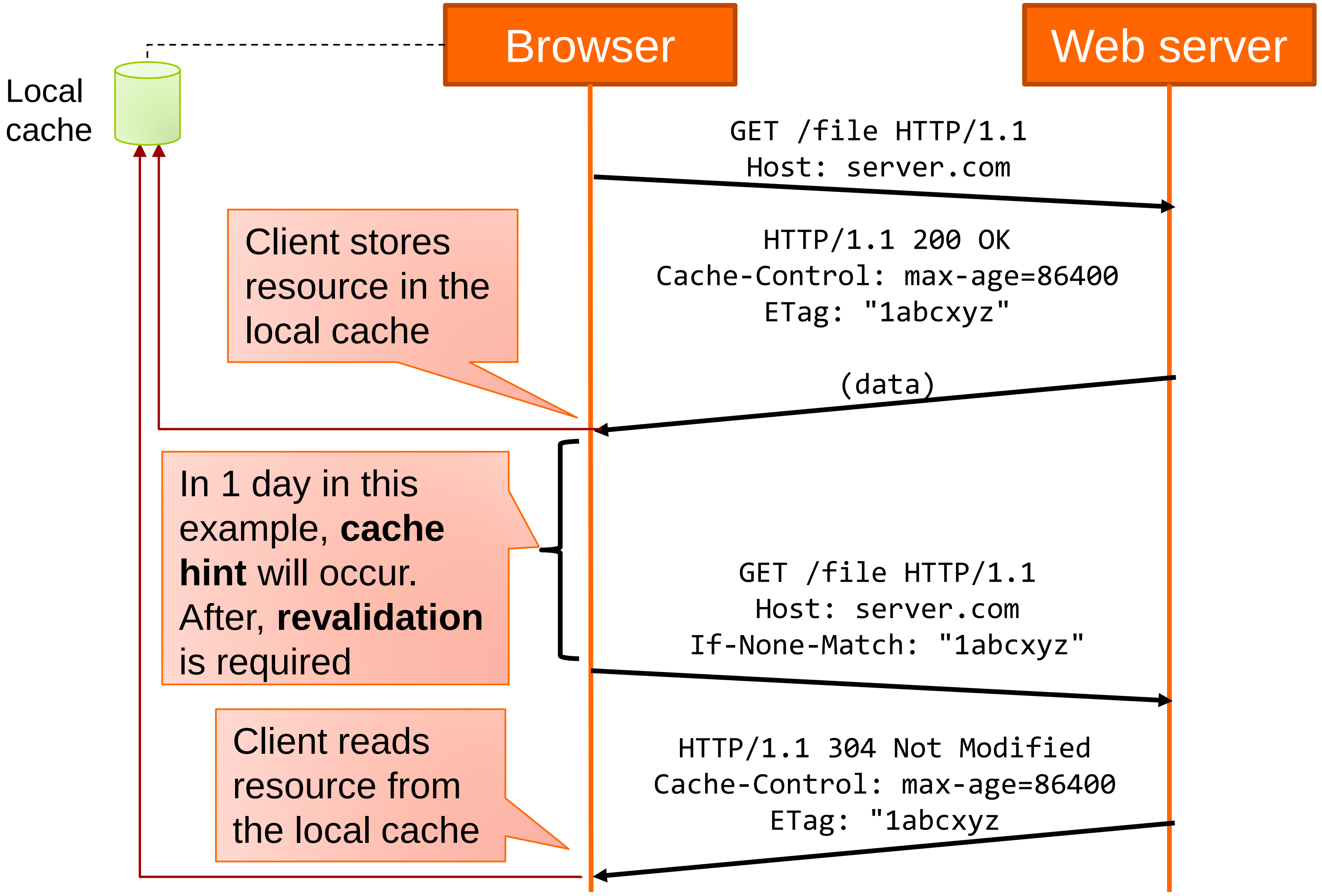
```
Expires: Wed, 21 Oct 2020 07:28:00 GMT
```

- If it is sent together with `Cache-Control`, `Cache-Control` has priority

## 2. HTTP 1.1 - Web cache

- The revalidation process can be done in two ways. The **strong validation** mechanism use the following headers:
  - **`ETag`**: Unique identifier of a resource
    - Typically it is the hash of the resource
    - Every time the resource changes, the `ETag` is changed as well in the server-side
  - **`If-None-Match`**: `ETag` value used by a client to **revalidate** a resource (find out if a cached resource has changed or not)

# 2. HTTP 1.1 - Web cache


Browser
Web server
Local cache
Client stores resource in the local cache
In 1 day in this example, **cache hint** will occur. After, **revalidation** is required
Client reads resource from the local cache
(data)

```
GET /file HTTP/1.1
Host: server.com
```

```
HTTP/1.1 200 OK
Cache-Control: max-age=86400
ETag: "1abcxyz"
```

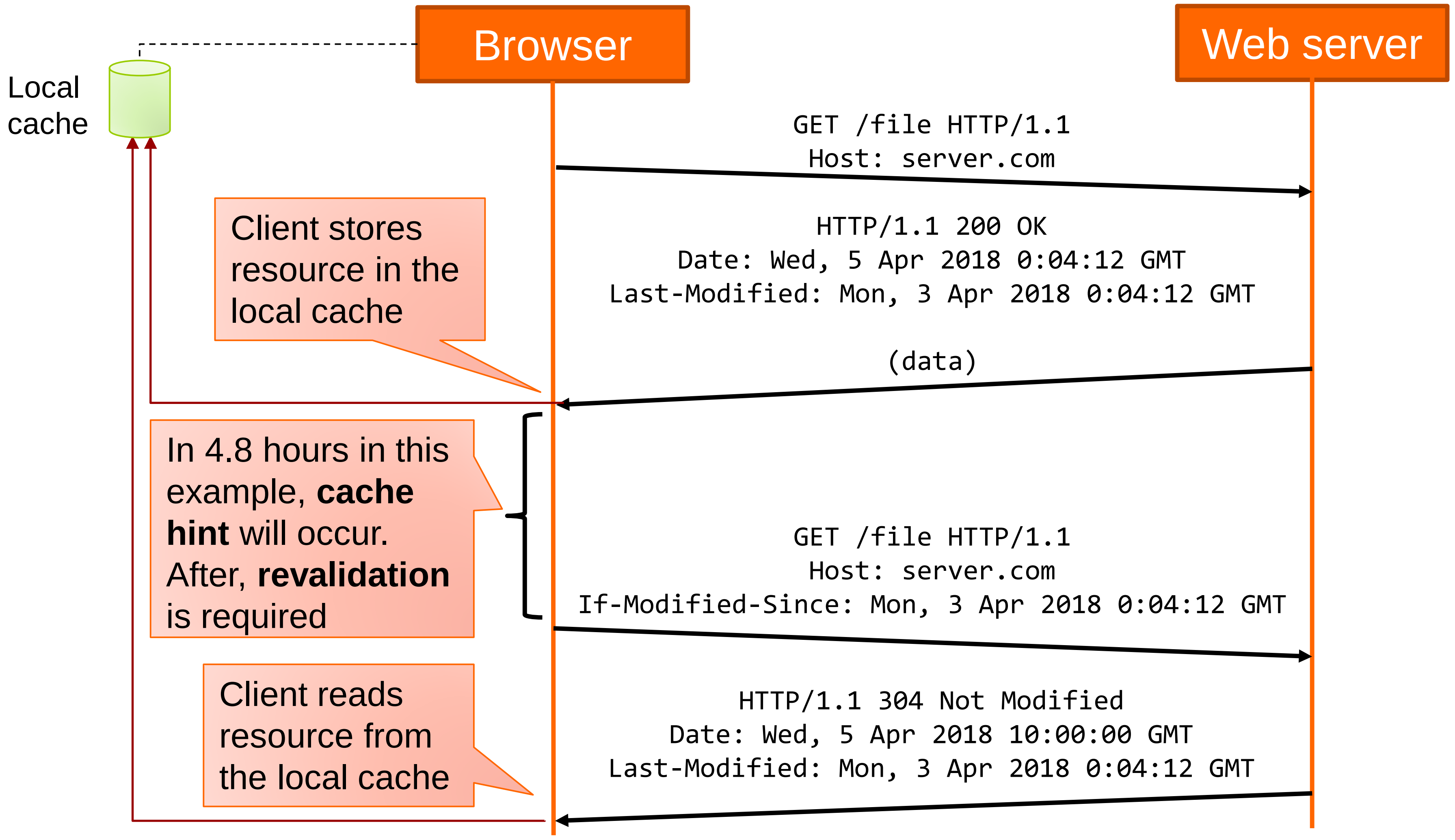
```
GET /file HTTP/1.1
Host: server.com
If-None-Match: "1abcxyz"
```

```
HTTP/1.1 304 Not Modified
Cache-Control: max-age=86400
ETag: "1abcxyz
```

## 2. HTTP 1.1 - Web cache

- The **weak validation** mechanism use the following headers (it is considered weak because it only has 1 second resolution):
  - **`Last-Modified`**: date and time at which the server believes the resource was last modified
  - **`If-Modified-Since`**: date and time used by a client to **revalidate** a resource (find out if a cached resource has changed or not)
- If both strong and weak validation is used in request (), strong is preferred

## 2. HTTP 1.1 - Web cache


Browser
Web server
Local cache
Client stores resource in the local cache
In 4.8 hours in this example, **cache hint** will occur. After, **revalidation** is required
Client reads resource from the local cache

```
GET /file HTTP/1.1
Host: server.com
```

```
HTTP/1.1 200 OK
Date: Wed, 5 Apr 2018 0:04:12 GMT
Last-Modified: Mon, 3 Apr 2018 0:04:12 GMT

(data)
```

```
GET /file HTTP/1.1
Host: server.com
If-Modified-Since: Mon, 3 Apr 2018 0:04:12 GMT
```

```
HTTP/1.1 304 Not Modified
Date: Wed, 5 Apr 2018 10:00:00 GMT
Last-Modified: Mon, 3 Apr 2018 0:04:12 GMT
```

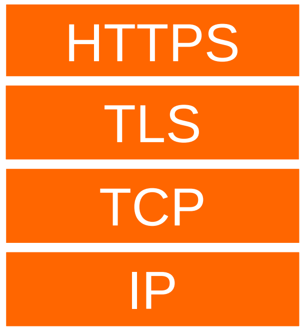
## 2. HTTP 1.1 - Web cache

- The header `Cache-Control` can be used also in **requests**. In this case, their values (can be concatenated) are interpreted as follows:
  - `no-cache`: resource cannot proceed from a cache without being revalidated
  - `no-store`: resource cannot be cached (in a proxy for example)
  - `max-age=X`: client requires an cached answer only if the resource age is less or equal than `X` (seconds)

## Table of contents

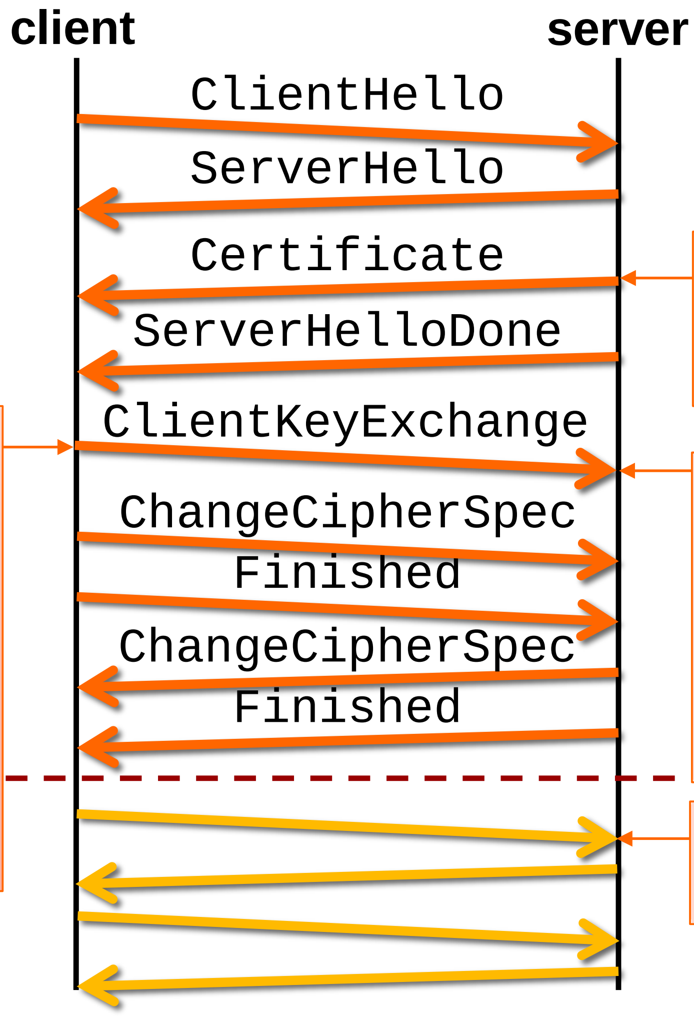
## 3. HTTPS

- HTTPS (Hypertext Transfer Protocol Secure) is the secure version of HTTP
- With HTTPS you get that sensitive information (keys, etc.) can not be intercepted by an attacker, since all you will get will be an encrypted data flow that will be impossible to decipher
- TLS (Transport Layer Security) is a protocol that provides encryption over TCP connections
- Default port for web servers that use HTTPs: 443



# 3. HTTPS

- TLS handshake in HTTPS:



The client generates a pre-master key that will be used to generate the MS master key with which all session data will be encrypted. This key is sent encrypted with the server's public key, obtained from the certificate

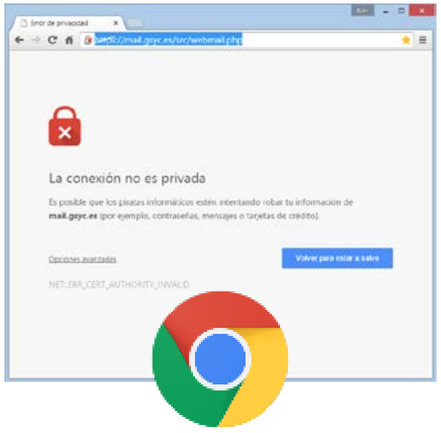
When sending the certificate the server is sending its public key to the client

The server obtains the pre-master key by deciphering the message sent by the client, using its private key. The MS is obtained from the pre-master

Data exchange through symmetric encryption

## 3. HTTPS

- A secure web server must have a certificate issued by a certification authority (CA)
- Browsers have a list of known CAs
- Upon receiving an invalid certificate, it shows a security alert to the user. This occurs when:
  - The certificate signed by an unknown CA
  - The certificate has expired

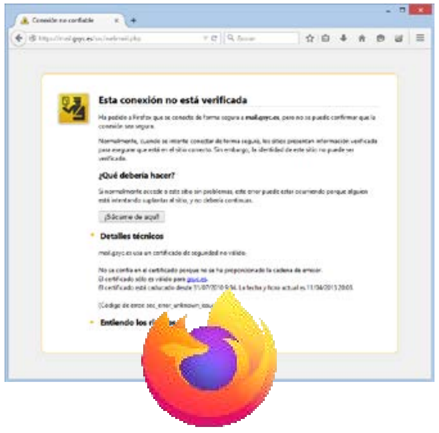

Error de privacidad
https://mail.gsyc.es/src/webmail.php

La conexión no es privada

Es posible que los piratas informáticos estén intentando robar tu información de **mail.gsyc.es** (por ejemplo, contraseñas, mensajes o tarjetas de crédito).

Opciones avanzadas
Volver para estar a salvo

NET::ERR_CERT_AUTHORITY_INVALID



**Esta conexión no está verificada**

Ha pedido a Firefox que se conecte de forma segura a **mail.gsyc.es**, pero no se puede confirmar que la conexión sea segura.

Normalmente, cuando se intente conectar de forma segura, los sitios presentan información verificada para asegurar que está en el sitio correcto. Sin embargo, la identidad de este sitio no puede ser verificada.

**¿Qué debería hacer?**

Si normalmente accede a este sitio sin problemas, este error puede estar ocurriendo porque alguien está intentando suplantar al sitio, y no debería continuar.

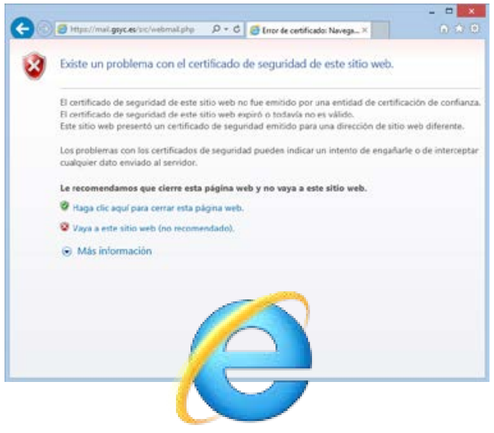
**Detalles técnicos**

mail.gsyc.es usa un certificado de seguridad no válido.

No se confía en el certificado porque no se ha proporcionado la cadena de emisor.
El certificado sólo es válido para gsyc.es.
El certificado está caducado desde 11/07/2010 9:56. La fecha y hora actual es 11/04/2013 20:05.

(Código de error: sec_error_unknown_issu[illegible]

**Entiendo los r[illegible]**


https://mail.gsyc.es/src/webmail.php
Error de certificado: Navega...

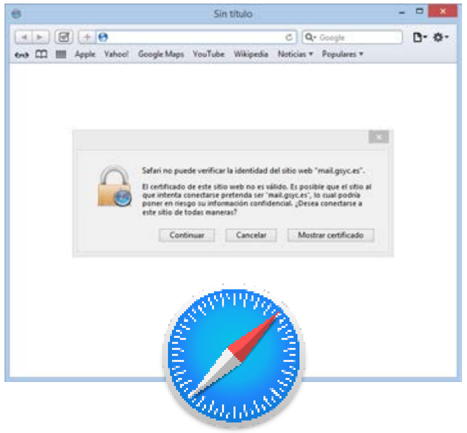
Existe un problema con el certificado de seguridad de este sitio web.

El certificado de seguridad de este sitio web no fue emitido por una entidad de certificación de confianza.
El certificado de seguridad de este sitio web expiró o todavía no es válido.
Este sitio web presentó un certificado de seguridad emitido para una dirección de sitio web diferente.

Los problemas con los certificados de seguridad pueden indicar un intento de engañarle o de interceptar cualquier dato enviado al servidor.

**Le recomendamos que cierre esta página web y no vaya a este sitio web.**

Haga clic aquí para cerrar esta página web.
Vaya a este sitio web (no recomendado).
Más información



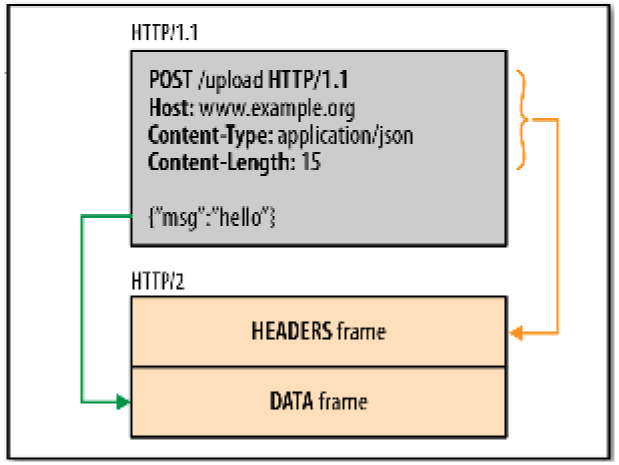
Safari no puede verificar la identidad del sitio web "mail.gsyc.es".

El certificado de este sitio web no es válido. Es posible que el sitio al que intenta conectarse pretenda ser "mail.gsyc.es", lo cual podría poner en riesgo su información confidencial. ¿Desea conectarse a este sitio de todas maneras?

## Table of contents

# 4. HTTP 2

- HTTP 2.0 introduces improvements aimed at decreasing latency:
  - Multiplexing requests (responses can be processed asynchronously)
  - Push service (the server can send resources before there is an explicit request from the client)
  - Header compression (through a mechanism called HPACK, defined in RFC 7541)
  - HTTP 2 talks all the semantics of HTTP 1.1 (verbs, headers, responses) but the protocol becomes binary instead of textual



## Table of contents

## 5. Takeaways

- **HTTP** (Hypertext Transfer Protocol) is the application layer protocol of the Web
  - **Clients**: browsers (Chrome, Firefox, Safari, Edge, Opera)
  - **Servers**: web servers (Apache, IIS)
  - The most used version of HTTP is **1.1** (migration to 2.0)
  - HTTP over TLS is known as **HTTPS**
- Clients send **requests** to servers:
  - First line includes **method** (e.g. GET, POST) and **URL**
  - Header examples: `Host`, `If-None-Match`, ...
- Servers sends **responses** to clients:
  - First line includes status code (e.g. 200, 404, …)
  - Headers in responses: `Content-Type`, `Content-Length`, `Cache-Control`, ...