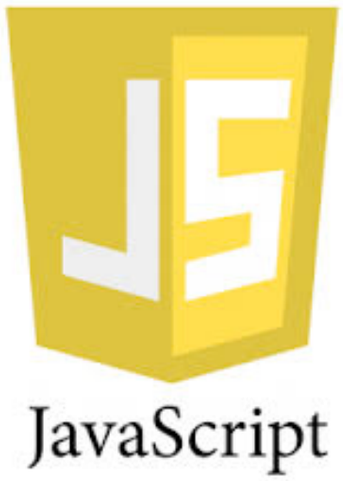


HTTP, conexiones, caches y REST

Juan Quemada, DIT - UPM

Índice: HTTP, conexiones, caches y REST

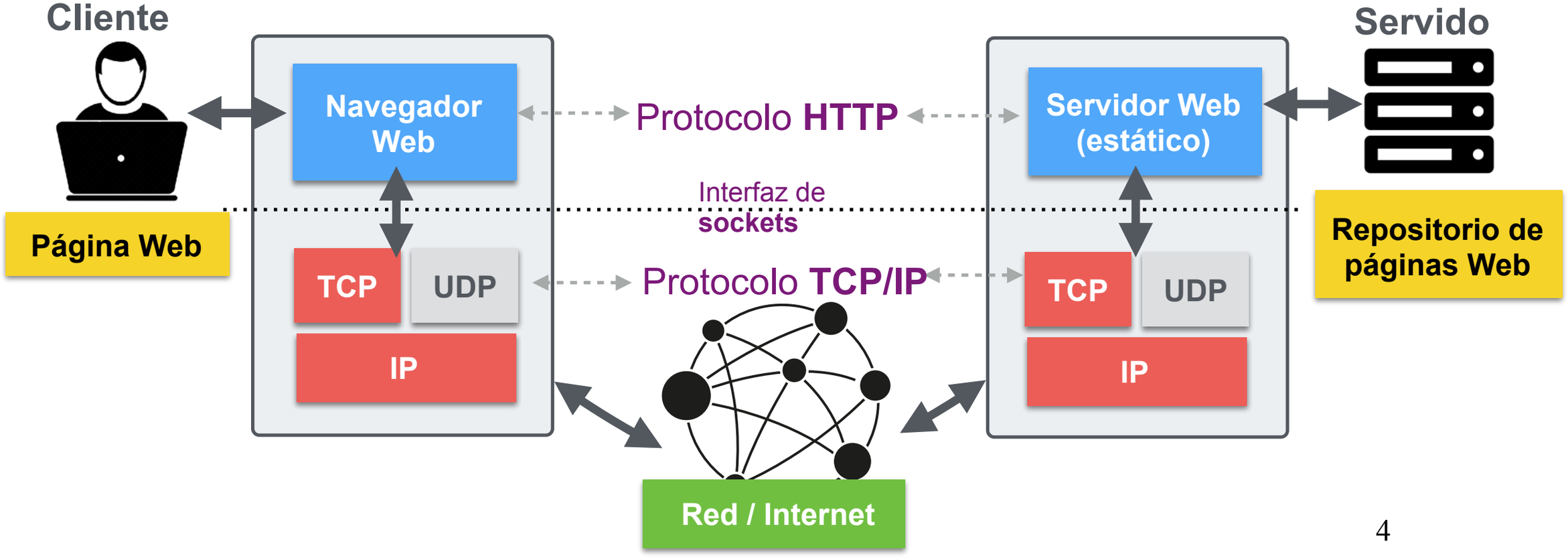
1. Publicación de Dos Web (I): URL, HTTP, servidor estático, transacción GET, código de respuesta y MIME …….. 3
2. Publicación de Dos Web (II): Ejemplos de servidor Web estático …….. 10
3. La plataforma Web actual: query, clientes y servidores programables y BBDDs …….. 14
4. HTTP 1.1 (I): Solicitudes, respuestas, interfaz uniforme, código de respuesta, seguridad, idempotencia y REST …….. 23
5. HTTP 1.1 (II): conexión HTTP, proxy, conexión persistente y paralela, respuesta chunked, caches y CDN …….. 30

Publicación de Docs Web (I):
URL, HTTP, servidor estático, transacción GET, código de respuesta y MIME

Juan Quemada, DIT - UPM

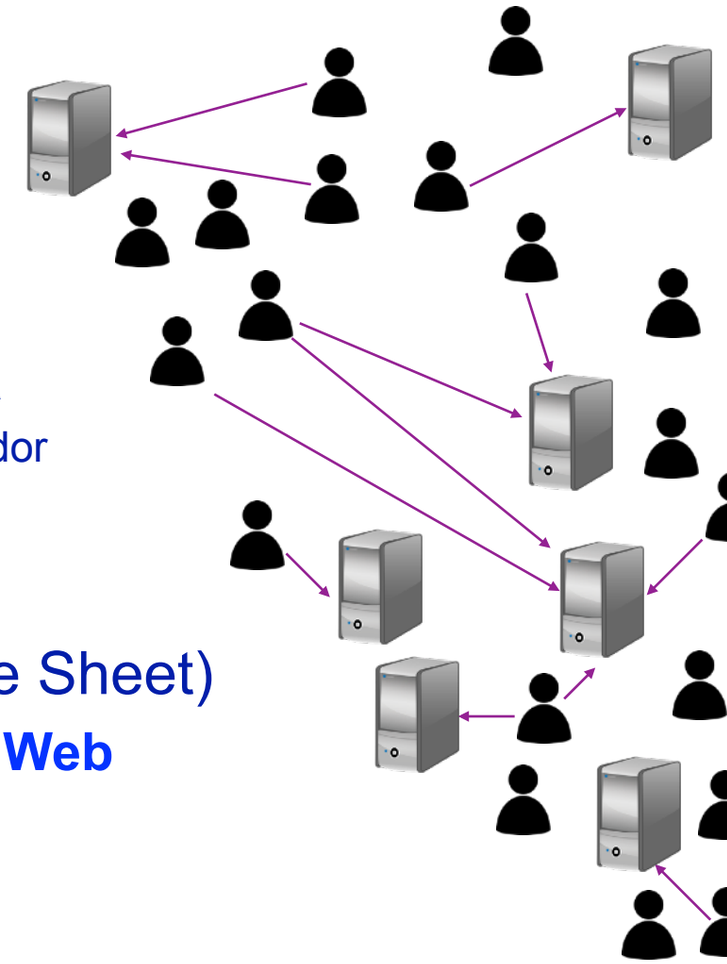
La Web

- Tim Berners Lee propone en 1989 una nueva aplicación: **la Web**
  - Es un servicio de publicación de documentos hipertexto
    - Publica y muestra documentos con facilidad utilizando un **Navegador** y un **Servidor Web estático**
- Es un servicio de publicación de documentos que Internet necesitaba
  - Su arquitectura permite crecer de forma descentralizada y escalable
    - La Web transforma Internet en una **"Red de distribución de contenidos"**



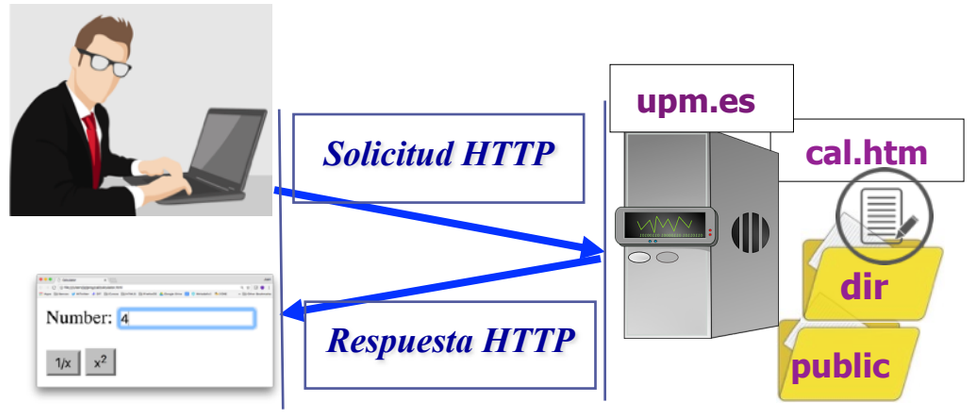
Publicación de documentos Web

- **Servidor Web estático**
  - Programa para publicar **páginas** Web (ficheros HTML), añadiendolas a su **repositorio de recursos**
    - El servidor envía las páginas a los clientes, cuando estos las solicitan con **HTTP**
- **Navegador**
  - Programa para **navegar** por páginas Web (en HTML) identificadas por **hiperenlaces** (URLs)
    - Las páginas Web se solicitan a los servidores a través del cajetín del navegador o con clics en hiperenlaces
- **URL** - Universal Resource Locator
  - **Dirección** de un fichero (recurso) o servicio en Internet
    - Nace para el Web, pero se generaliza para acceder a cualquier servicio
  - Tiene 3 partes principales
    - **Esquema o protocolo:** protocolo de acceso utilizado por el servídor
    - **Dirección del servidor:** dirección de dominio o IP del **servidor** con la página
    - **Ruta del recurso:** ruta al **fichero** en el directorio de recursos del servidor
  - Ejemplo: https://en.wikipedia.org/wiki/URL
    - Esquema (**https**), servidor (**en.wikipedia.es**), Ruta (**/wiki/URL**)



- **HTML** (HyperText Markup Language) y **CSS** (cascading Style Sheet)
  - **HTML** es el lenguaje para definir la estructura de un **documento Web**
    - **CSS** el lenguaje para definir el **aspecto visual** de un documento Web
- **HTTP -** HyperText Transfer Protocol
  - **Protocolo** para traer un **fichero** desde un **servidor remoto** a un cliente
    - Protocolo simple, **sin estado** y **muy escalable**

HTTP GET y el Servidor Web estático



HTTP

- Protocolo **transaccional** de tipo **solicitud-respuesta**
  - Las transacciones permiten realizar operaciones en los recursos en un servidor
    - El cliente debe establecer previamente una **conexión con el servidor**, normalmente una conexión TCP

Transacción HTTP GET:

- El cliente envía una **solicitud GET** del recurso al servidor (a través de la conexión)
  - El recurso (fichero) se identifica con la **ruta en el repositorio de recursos**
- El servidor envía el recurso (fichero) en la **respuesta** (a través de la conexión)
  - El **tipo de contenido** de la respuesta se identifica con un **tipo MIME**

Servidor Web estático

- Sirve ficheros solicitados por clientes con transacciones **GET** de **HTTP**
  - Los ficheros están en el **repositorio de recursos** (directorio con páginas y otros recursos Web)
    - El directorio con los recursos suele denominarse: **www**, **public**,

Tipo MIME y estado del servidor

Tipo MIME

- Define el **tipo** de un recurso o fichero en email, Web, ...
  - Tiene la siguiente estructura: "**tipo/subtipo",** por ejemplo
    - **text/plain**, **text/html**, ...
    - **image/gif**, **image/jpeg**, ...
  - Ver: https://developer.mozilla.org/ar/docs/Web/HTTP/Basics_of_HTTP/MIME_Types

Extensión de un fichero

- Indica su contenido tiene un tipo MIME asociado
  - Ver: https://www.file-extensions.org
- Por ejemplo
  - **text/plain** (*.txt, *.text, *.conf, ...), **text/html** (*.html, *.htm), **text/css** (*.css), ...
  - **image/gif** (*.gif), **image/jpeg** (*.jpeg, *.jpg, *.jpe), **image/png** (*.png), …..

Estado del servidor

- El estado del servidor respecto a la solicitud determina la respuesta enviada
  - **200 Ok**: El fichero solicitado está en el repositorio y se envía en la respuesta
  - **400 Bad Request**: El método (tipo de transacción HTTP) **NO** está no está soportado por el servidor
  - **404 Not Found**: El fichero solicitado **NO** está en el repositorio y no se envía

Solicitud y Respuesta HTTP

- **Solicitudes** y **respuestas HTTP** tienen una misma estructura, con dos partes
  - **Cabecera** (header): es un string terminado por una línea en blanco (\n\n), con 2 partes
    - **Primera línea**: tiene 3 parámetros separados por un espacio en blanco
    - **Bloque de parámetros**: cada parámetro ocupa una línea
  - **Cuerpo** (body): parte reservada para los datos, que están en el formato indicado por un tipo MIME

- **Primera línea**
  - **método** (o esquema) solicitado: **GET**
  - **ruta** al recurso en el servidor: **/me.htm**
  - **versión** de HTTP del cliente: **HTTP/1.1**
- **Parámetro** más importante
  - **Host: upm.es** dirección del servidor
  - **Accept: text/*, image/*** **tipos MIME** aceptados
  - **Accept-language: en, sp** acepta español o ingles
  - **User-Agent: Mozilla/5.0** identificador del tipo de cliente
- **Cuerpo**
  - vacío (no envía ninguna información)

Solicitud HTTP GET

1ª linea	**GET /me.htm HTTP/1.1**
parámetros	**Host: upm.es** **Accept: text/*, image/*** **Accept-language: en, sp** ..... **User-Agent: Mozilla/5.0**
Cuerpo	

- **Primera línea**
  - **versión** de HTTP del servidor: **HTTP/1.0**
  - **estado** del servidor: **200**
  - **mensaje** informativo: **Ok**
- **Parámetros** más importantes
  - **Server: Apache/1.3.6** identificador del tipo de servidor
  - **Content-type: text/html** el cuerpo lleva un documento **HTML**
  - **Content-length: 608** el cuerpo ocupa **608 octetos**
- **Cuerpo**
  - Lleva el contenido del fichero HTML solicitado

Respuesta HTTP GET

1ª linea	**HTTP/1.0 200 OK**
parámetros	**Server: Apache/1.3.6** **Content-type: text/html** ..... **Content-length: 608**
Cuerpo	**<html> …….. </html>**

Conexión TCP y carga de una página

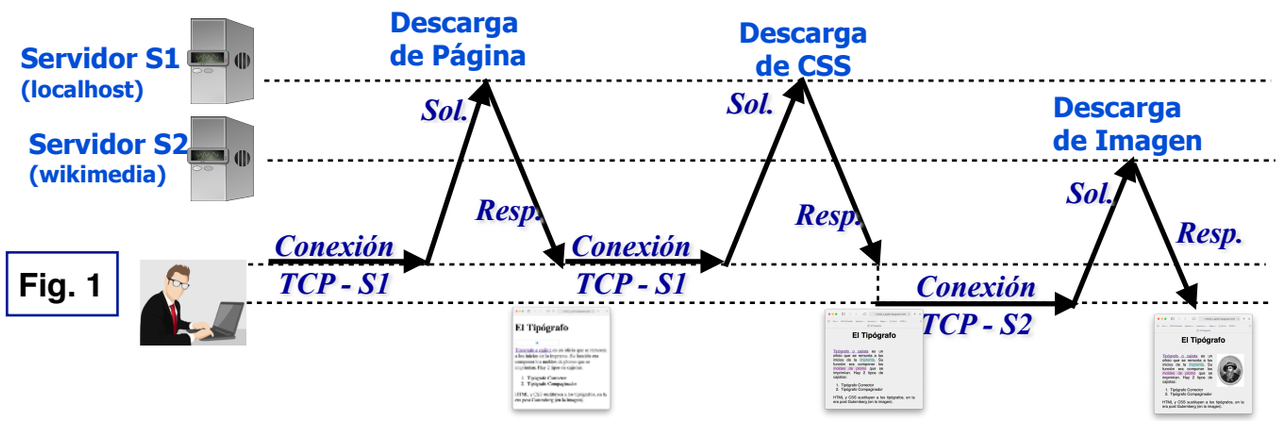


Fig. 1

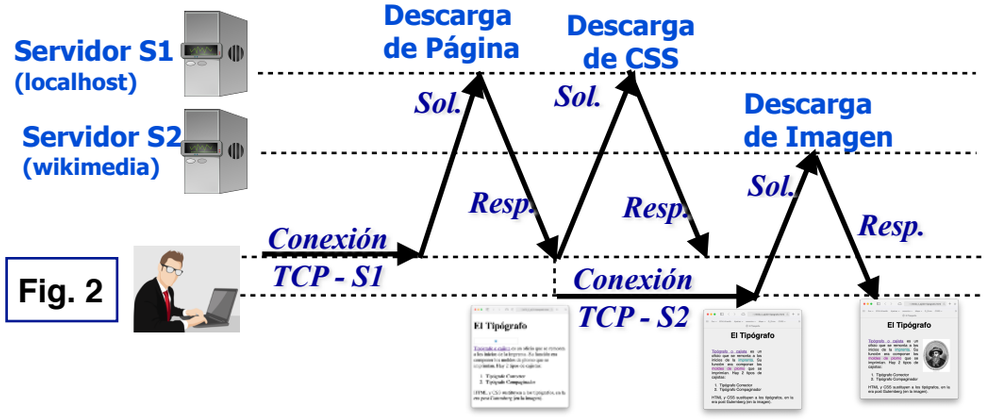
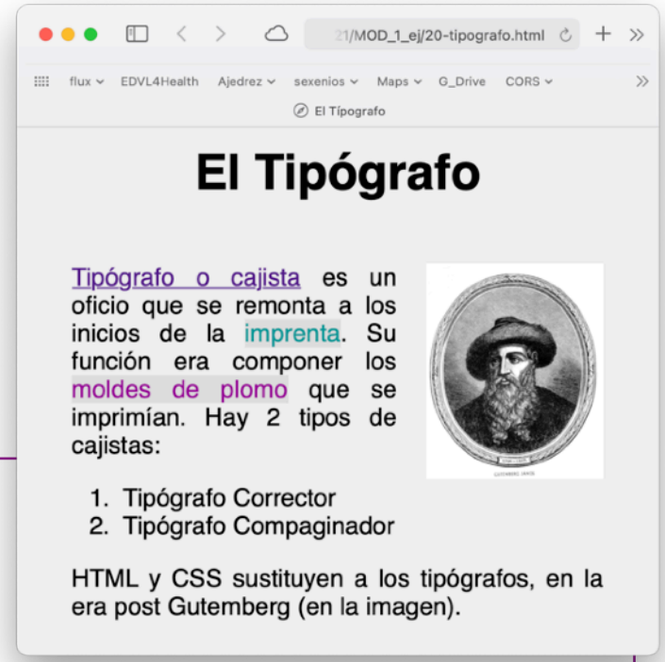


Fig. 2




```
<!DOCTYPE html><html>
  <head>
    <title>
      El Típografo
    </title>
    <meta charset="utf-8">
    <link rel="stylesheet"
          href="20-tipografo.css"
          type="text/css" />
  </head>

  <body>
    <h1>El Tipógrafo</h1>

    <img src="https://upload.wikimedia.org/wikipedia/commons/e/e1/
    Gutemberg.jpg" alt="Imagen de Gutemberg">

    <div class="texto">
      <a href="https://es.wikipedia.org/wiki/Cajista">Tipógrafo o
      cajista</a> es un oficio que se remonta a los inicios de la
      <span id="t1" class="bg"> imprenta</span>. Su función era
      componer los <span id="t2" class="bg">moldes de plomo</span>
      que se imprimían. Hay 2 tipos de cajistas:

      <ol>
        <li> Tipógrafo Corrector</li>
        <li> Tipógrafo Compaginador</li>
      </ol>

      HTML y CSS sustituyen a los tipógrafos, en la era post
      Gutemberg (en la imagen).
    </div>
  </body>
</html>
```

Publicación de Docs Web (II):
Ejemplos de servidor Web estático

Juan Quemada, DIT - UPM

Servidor Web estático realizado con sockets

```
let net = require('net');
let fs   = require('fs');
let mime = require('mime-types');
let port = (process.argv[2] || 8080);

function response (code, mime, file) {
    let body = (file) ? file : Buffer.from(`<html><body><h1>Error: ${code}<h1></body></html>`);
    let head = `HTTP/1.0 ${code}\nContent-type: ${mime}\nContent-length: ${body.length}\n\n`;
    return Buffer.concat([Buffer.from(head), body]);
}
      // Create server socket with TCP connection handler
let server = net.createServer( (socket) => {

    socket.on('data', function (data) {              // HTTP request handler
        let [method, path] = data.toString().split('\n')[0].trim().split(' ');
        path = 'public' + path;
        if (method.toLowerCase() === 'get') {
            fs.readFile(path, function (err, file) {
                if (err) {                    // Web page does not exist
                    socket.write(response(`404 Not Found <p>${err}`, 'text/html'));
                } else {
                  let f_mime = mime.lookup(path) || 'text/html';
                  socket.write(response(`200 OK`, f_mime, file));
                }                             // Sends requested page
            });
        } else {                          // Unsupported HTTP method
            socket.write(response(`400 Bad Request: ${method}`, 'text/html'));
        }
  });
});

server.listen(port);
console.log(`Static Web server at port: ${port}`);
```

Modulos utilizados
'**net':** Sockets TCP de cliente y servidor
'fs': gestión del sistema de ficheros
'mime-types': obtención tipos MIME de extensión
(debe instalarse con: $ npm install **'mime-types'**)

Función **response:** genera las respuestas al cliente. Estas se construyen con la clase **Buffer**: objetos creados con arrays de octetos que node utiliza para contenidos binarios.

La lectura de un fichero, devuelve su contenido en un **Buffer.** Las cabeceras se construyen con strings. El método **from** traduce strings de **'utf16le'**, su formato interno, a **'utf8'** el formato por defecto de strings binarios de un Buffer.

Extrae el método y la ruta de la primera línea y añade el prefijo **public** a la ruta, para que los ficheros se busquen en ese directorio. El directorio public es el repositorio de recursos Web del servidor.

Si el fichero solicitado no está en el repositorio, responde con: **404 Not Found**.

Si el fichero solicitado está en el repositorio, lo sirve al cliente: **200 Ok**.

Si la primitiva recibida no es GET, contesta **400 Bad Request**, porque es el único método soportado.

Ejemplos de otros 3 servidores estáticos

```
var HTTP = require('http');
var FS   = require('fs');

var server = HTTP.createServer(
  (request, response) => {
    FS.readFile(
      ('public' + request.url),
      function(err, data) {
        if (!err) {
          response.writeHead(
            200,
            { 'Content-Type': 'text/html',
              'Content-Length': data.length
            }
          );
          response.end(data);
        }
        else { response.end('error'); };
      }
    );
  }
)
server.listen(8080);
```

Crear servidor. Cada solicitud HTTP invoca el manejador de solicitudes.

El manejador recibe 2 parámetros:
- **request**: objeto con los parámetros de la solicitud HTTP recibida
- **response**: objeto preconfigurado para ensamblar la respuesta, que se envía con **response.send(..)**

Si el recurso está en el repositorio, lo sirve al cliente con **200 Ok**.

Si el recurso no está en el repositorio, responde al cliente con **'error'**

```
var express = require('express');
var path = require('path');

var app = express();

app.use(express.static(path.join(__dirname, 'public')));

app.listen(8080);
```

var path = require('path')
importa el módulo path de gestión de rutas.

var app = express()
crea la aplicación express.

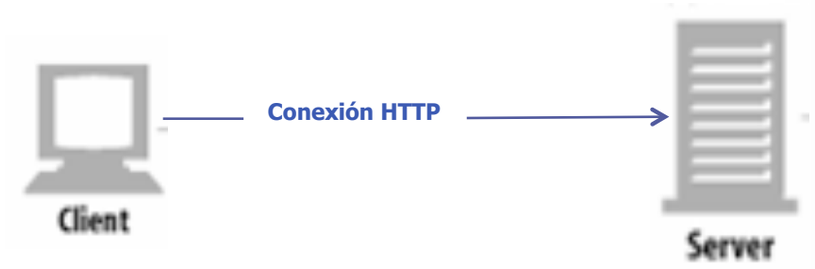
path.join(__dirname, 'public'):
crea ruta absoluta al directorio del repositorio de recursos public.

app.use(express.static(path.join(__dirname, 'public')))
instala el servidor estático de express la aplicación express **app**.

El módulo npm **serve** implementa un servidor estático que solo hay que instalar y arrancar para tenerlo operativo.

```
$
$ npm install serve
$
$ serve public # arranca servidor
$              # en puerto 3000 y
$              # configura repositorio
$              # en directorio public
.............
$
```

CURL



CURL: comando para interaccionar con servicios identificados por un URL.

Soporta muchos protocolos: HTTP, HTTPS, IMAP, SMTP, Kerberos, ...

La opción -v (verbosa) muestra todos los detalles del proceso.

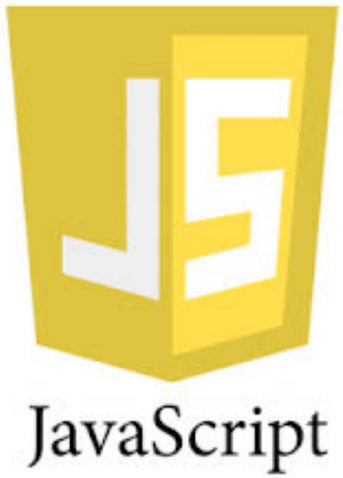
Ver opciones con:

$ curl --help

.....

$ man curl

```
jq — bash — 54×21
venus:~ jq$
venus:~ jq$ curl -v http://localhost:8000/mi_ruta
* Hostname was NOT found in DNS cache
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 8000 (#0)
> GET /mi_ruta HTTP/1.1
> User-Agent: curl/7.37.1
> Host: localhost:8000
> Accept: */*
>
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Content-Type: text/plain; charset=utf-8
< Content-Length: 43
< ETag: W/"2b-1b6a7631"
< Date: Mon, 27 Oct 2014 14:31:05 GMT
< Connection: keep-alive
<
<html><body><h1>Mi Ruta</h1></body></html>
* Connection #0 to host localhost left intact
venus:~ jq$
```

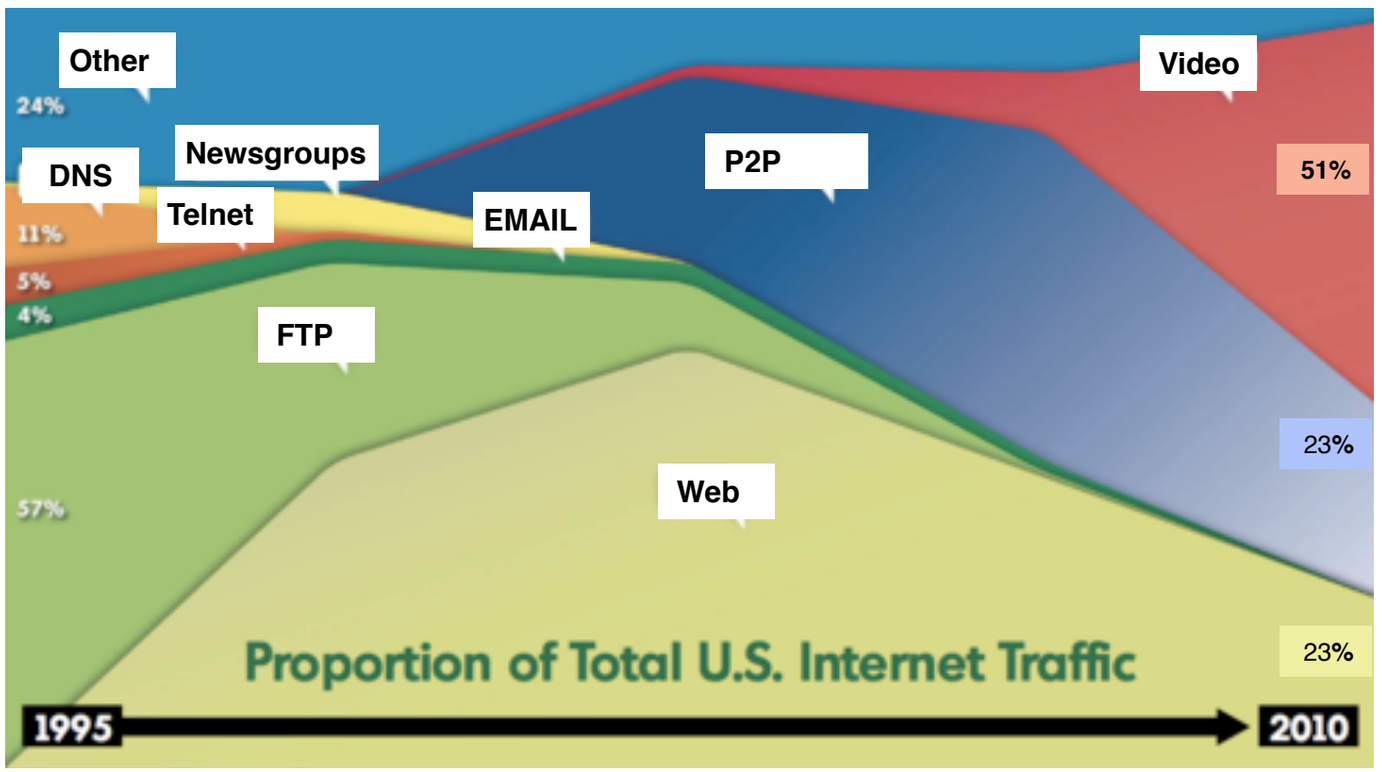


La plataforma Web actual:
query, clientes y servidores programables y BBDDs

Juan Quemada, DIT - UPM

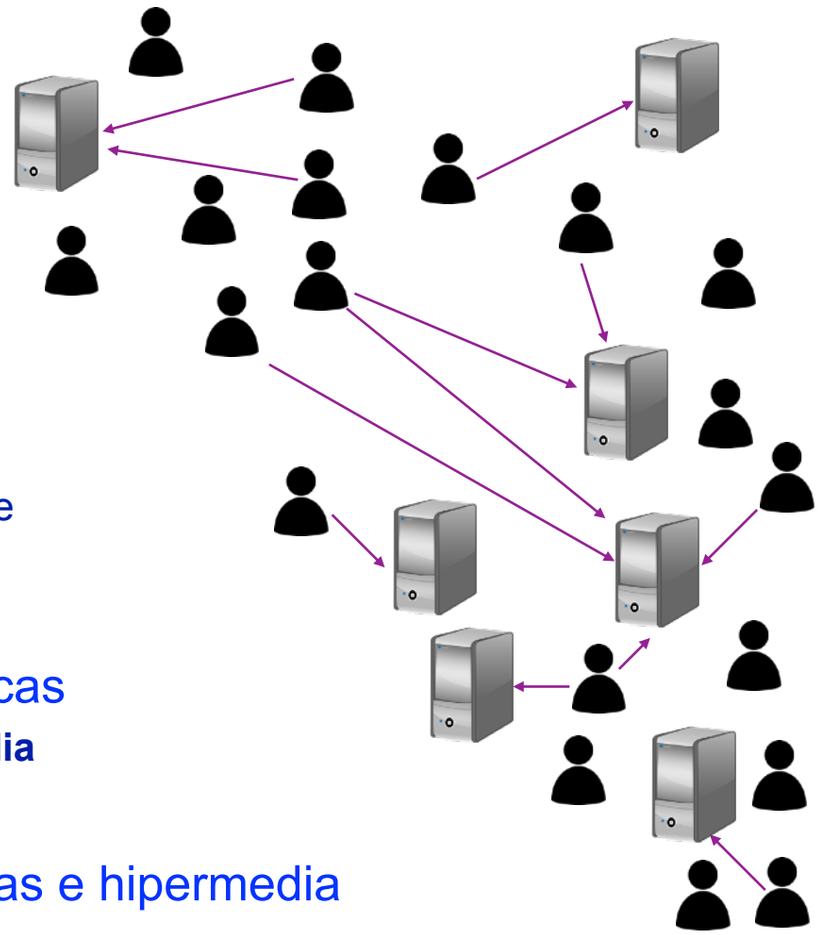
Computación distribuida y la plataforma Web

- Paradigma de **computación distribuida**
  - Partes de un programa cooperan a través de Internet con un objetivo común
    - Plantea múltiples retos relacionados con la concurrencia entre procesos y la comunicación entre ellos, transacciones seguras, sincronización de relojes, tolerancia a fallos de las partes, etc.
  - Existen muchas propuestas: Web, CORBA, Fractal, JavaBeans, NFS, AFS, ..
    - La **plataforma Web** se ha impuesto y es la solución dominante hoy en Internet
- La **plataforma Web**
  - Es la **plataforma HTML5** desplegada sobre TCP/IP (**interfaz de sockets)**
    - Permite diseñar cualquier tipo de aplicación de Internet



La plataforma Web actual

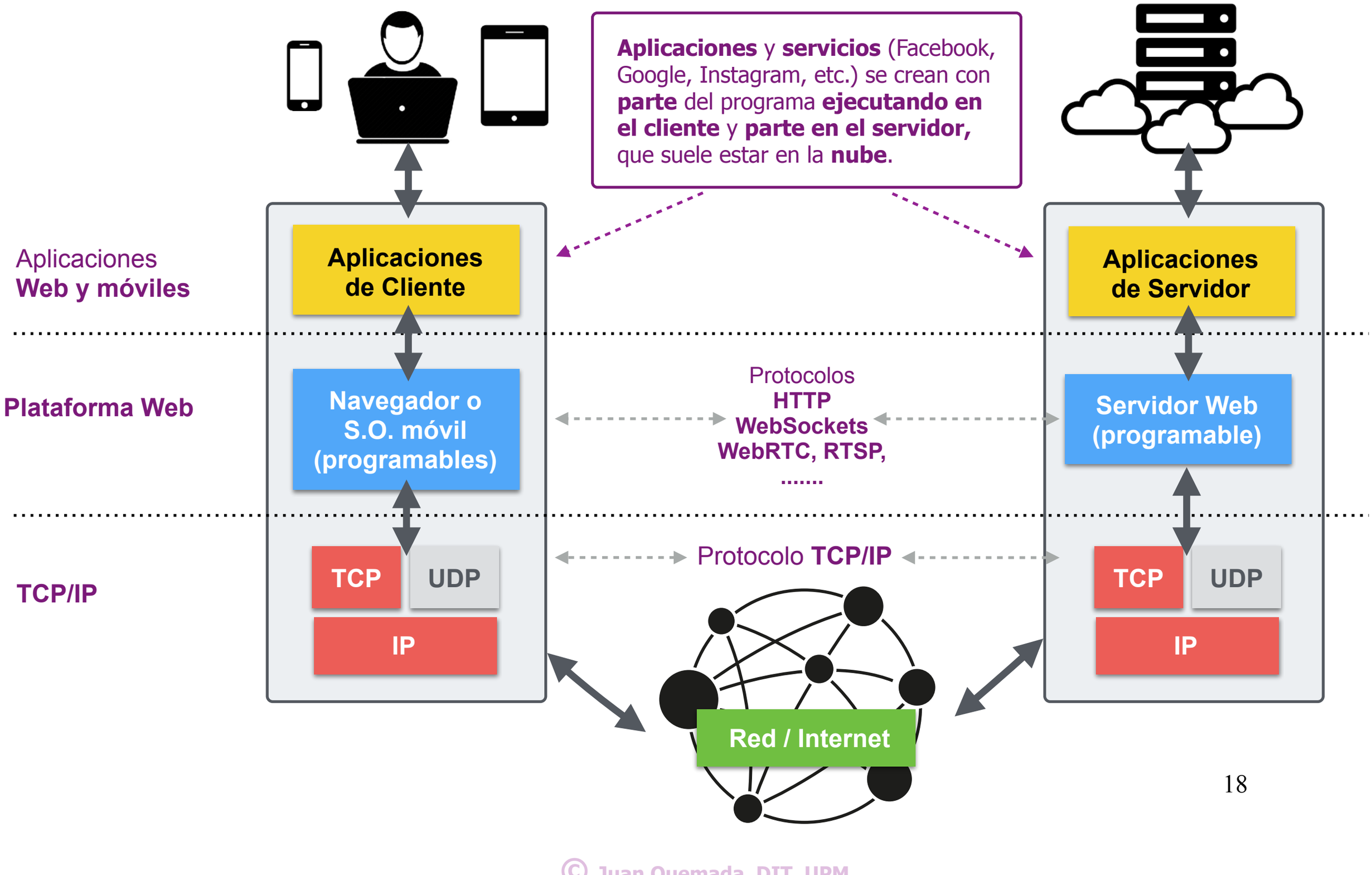
- **Cliente Web**
  - **Navegador**: se hace **programable** para mejorar la experiencia de usuario y descargar el servidor
    - Mejora el tiempo de respuesta de los servicios y disminuye el tráfico por Internet
  - Aparecen **los dispositivos móviles**: sustituyen el navegador por aplicaciones instalables
- **Servidor Web**
  - El servidor se hace **programable** y se especializa en gestionar accesos a **BBDDs** remotas
- **URL** - Universal Resource Locator
  - Se añade la **query** que envía **parámetros** del cliente al servidor
    - Por ejemplo: https://upm.es/registro**?nombre=José&apellido=Perez**
- **HTTP** - HyperText Transfer Protocol
  - HTTP incorpora **nuevas funcionalidades** y se hace **mas eficiente**
  - Se añaden **nuevos** protocolos: **WebSockets, RTSP**, **WebRTC, ...**
    - Permiten que la plataforma Web soporte **cualquier tipo de aplicación** en la nube
- **HTML** - HyperText Markup Language
  - **HTML5** redefine su función y se amplía con nuevas marcas semánticas
    - HTML solo debe describir la estructura de los documentos e interfaces **hipermedia**
  - Se añade **CSS** para definir el **aspecto visual**
  - Se añade **JavaScript** para incluir **aplicaciones de cliente** interactivas e hipermedia



Historia de la Plataforma Web

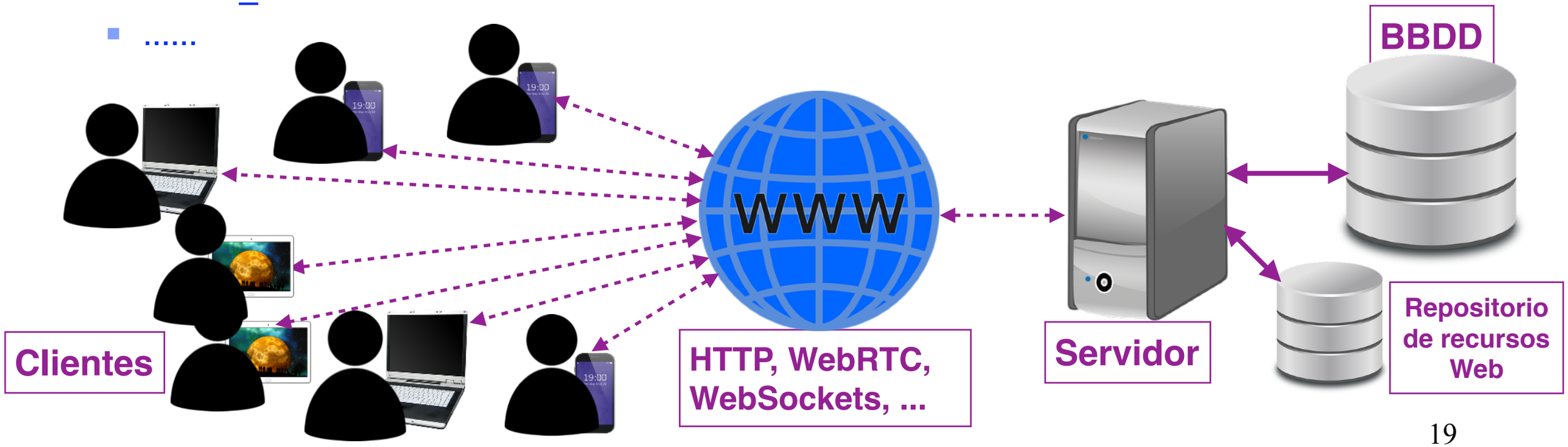
- Nacimiento de la **World Wide Web**
  - Aplicación para **publicar documentos en Internet** propuesta por Tim Berners Lee hacia 1989
    - Basada inicialmente en tres elementos: URL, HTTP y HTML
  - Tim Berners Lee demuestra hacia 1990 el primer navegador (textual) y el primer servidor
    - Enseguida aparecen otros y el navegador se convierte en la principal ventana de acceso Internet
  - En 1994 se crea el W3C (World Wide Web Consortium) para ordenar el desarrollo de la Web
- **Navegadores comerciales iniciales**
  - Netscape Navigator (1994), Microsoft Explorer (1995), Opera (1996), ..
- **Programación del navegador**
  - **Netscape** añade **JavaScript** (Brendan Eich) a Navigator en 1995
    - Netscape envía JavaScript a ECMA para su normalización en 1996, aparecen ECMASCRIPT 1.0 (1997), 3.0 (1999) y 5.0 (2009)
  - Microsoft añade **JScript** a Internet Explorer en 1996 (similar a JS pero con incompatibilidades)
  - Macromedia lanza **Flash** 1.0 en 1996 como un plugin de los navegadores existentes
- **CSS** - Cascading Style Sheets (1996)
  - HTML se centra en definir la **estructura del documento**, dejando a CSS el **diseño gráfico**
- **Evolución**
  - **Explorer** desplaza a **Netscape** (finales de los 90) y **Flash** se impone para contenidos multimedia (2000)
  - El W3C se centra en **XHTML** y **Web Services SOAP** y pierde protagonismo
- **Plataforma HTML5**
  - **Navegadores HTML5:** Firefox (2003), Safari (2003) Chrome (2008), Edge (2015), ...
  - Apple, Mozilla y Opera crean **WHATWG** (2004) para definir una **nueva plataforma Web**
    - Basada en **HTML5**, **CSS3**, **JavaScript5, HTTP** y mas protocolos

Arquitectura de la Plataforma Web



Arquitectura de 3 capas

- Los servicios y aplicaciones de Internet suelen tener estás 3 capas
  - **Cliente**: Capa de visualización y presentación con la interfaz del servicio
  - **Servidor**: Capa lógica con las reglas de orquestación de la respuesta a las peticiones
  - **BBDD**: Capa de persistencia que almacena los datos en una base de datos
- Cliente y servidor se comunican a través de múltiples protocolos:
  - HTTP: HTTP tiene hoy mas funcionalidad, mas eficiencia y menos latencia
  - Web Sockets: Para aplicaciones interactivas entre clientes
  - RTSP: Para streaming de vídeo
  - WebRTC: Para aplicaciones de voz y video sobre IP
  - Server_send events: Para notificación de eventos del servidor al cliente
  - .....



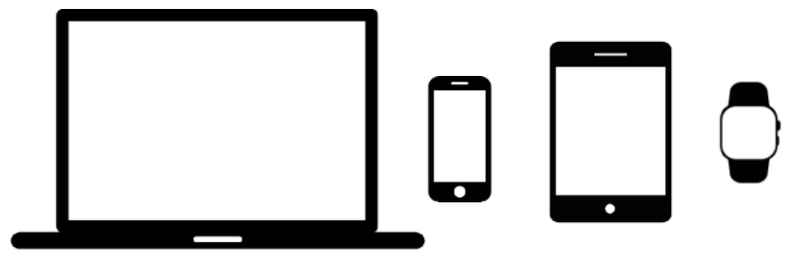
URI y URL

- **URL** (Uniform Resource Locator)
  - **Dirección** de acceso a cualquier **recurso** o **servicio** de Internet
    - Los **URLs** (RFC1738) son un caso particular de los **URIs** (Uniform Resource Identifiers, RFC3986)
      - https://www.ietf.org/rfc/rfc1738.txt y https://tools.ietf.org/html/rfc3986

scheme://user:password@host:port/path?query#fragment

- **http://upm.es/dir/pagina.html**
  - URL Web que identifica e la página Web **/dir/pagina.html** en el servidor **upm.es**
- http://**upm.es:8080**/dir/pagina.html
  - URL Web similar a la anterior, donde el servidor escucha en el **puerto 8080** y no en el 80 asignado a Web
- http://upm.es/dir/**pagina.html#p3**
  - URL igual al anterior pero con **fragment** o **anchor** (ancla), que identifica el elemento con **id='p3'** en **pagina.html**
- http://**felix@upm.es**/dir/pagina.html
  - URL Web de un recurso asociado al usuario **felix** en su cuenta en el servidor **upm.es**
    - Se recomienda enviar passwords en URLs solo con HTTPS y no con HTTP, porque es inseguro
- http://upm.es/**registro?id=23&nombre=José**
  - URL que envía dos parámetros en la query (parámetros **id** y **nombre)**
- **mailto:felix@upm.es**
  - URL de email que identifica el buzón del usuario **felix** en el servidor **upm.es**

El cliente y sus aplicaciones



- **Dispositivos** cliente de acceso a Internet
  - PCs, portátiles, tabletas, teléfonos y relojes inteligentes, etc.
- **Cliente: programa** que accede a servicios en Internet
  - El **navegador (browser)** es el principal cliente de acceso desde un PC
  - Las **apps** de los dispositivos móviles son hoy los clientes mas utilizados
- **Navegadores:** Apps se programan en **HTML, CSS y JavaScript**
  - Chrome, Firefox, Internet Explorer, Opera, Safari, ...
- **Aplicaciones nativas (apps):** Android, iOS-Apple, etc.
  - Se programan en entornos de desarrollo con lenguajes específicos
    - **Android** se programa en **Java**, **iOS** en **Swift**, etc
  - También se programan en **JavaScript** en entornos para aplicaciones **nativas**
    - **React Native**, **PWAs** (Progressive Web Apps), ... (reutilizan el código del navegador)

El servidor y sus aplicaciones

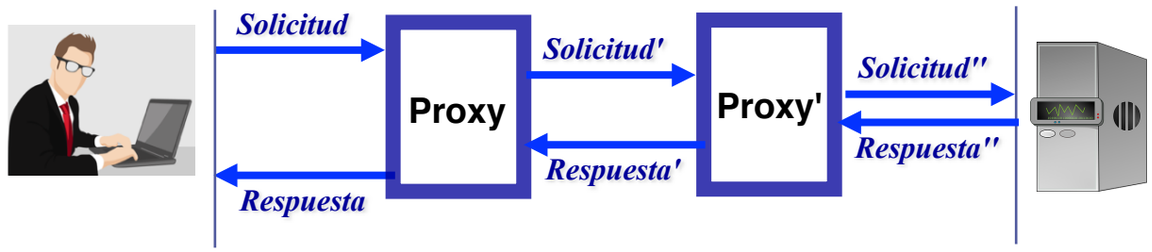
- **Servidor***
  - **Programa** proveedor de servicios a los clientes
    - Se conecta a un **puerto** de la **máquina servidora**, el servidor **Web** usa el **puerto 80** por defecto
      - *La **máquina servidora** se denomina también servidor, pero produce ambigüedad
- El programa **servidor** se ejecuta en una **máquina servidora**
  - Una máquina servidora tiene una dirección "**conocida**" en Internet
    - La dirección esta incluida en el URL de acceso: https://**en.wikipedia.org**/wiki/URL
      - Dirección de la máquina servidora: **en.wikipedia.org**
  - La máquinas servidoras pueden ser máquinas físicas o máquinas virtuales en la nube
- Servidores Web más usados: Apache, Nginx, Microsoft-IIS, etc.
  - Los servidores Web integran aplicaciones en múltiples lenguajes de programación
    - node.js + **JavaScript**
    - Ruby on Rails
    - Django + Python
    - Spring MVC + Java
    - Zend + PHP
    - etc

HTTP 1.1 (I):

Solicitudes, respuestas, interfaz uniforme, código de respuesta, seguridad, idempotencia y REST

Juan Quemada, DIT - UPM

HTTP 1.1



- Hoy se utiliza sobre todo **HTTP 1.1** (1997, 1999 y 2014 - RFCs 7230-7)
  - **HTTP/1.x** ha evolucionado mucho, aunque con pocas versiones 1.0, 1.1
  - **HTTP/2** (2015) está **aprobada** (en estado de transición) y **HTTP/3** está en desarrollo
    - Mejoran sobre todo la latencia, las prestaciones y el transporte basado en UDP
- **HTTP 1.1**
  - Soporta **páginas** Web **hipermedia** y acceso a **servicios** de Internet
    - Ver: https://developer.mozilla.org/en-US/docs/Web/HTTP, https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
  - Es un protocolo **transaccional** sin estado de **solicitud-respuesta**
    - Es bastante similar al inicial, pero con mas **funcionalidad**, **rapidez** y **eficiencia**
    - Solicitud y respuesta son **independientes**, como envíos de email
- **Solicitud** y **Respuesta**
  - Mantiene el **formato** de cabecera y cuerpo, añadiendo
    - **Primitivas** (además de **GET):** POST, PUT, DELETE, HEAD, OPTIONS, TRACE, ...
    - Nueva **funcionalidad** basada en nuevos **parámetros**
- **Conexión HTTP**
  - El cliente debe establecer una **conexión** al servidor para descargar recursos
    - Las **solicitudes** y **respuestas** se envían a través de dicha conexión
  - Se hace **mas rápida y eficiente** con persistencia, paralelismo (pipeline) y respuestas chunked
  - Puede atravesar **dispositivos intermedios** (Proxy), que añaden nueva funcionalidad (cache, proxy, **..**)
  - Soporta conexiones **simultáneas** con **todos los servidores** de los que descarga recursos

Solicitudes HTTP

POST ruta HTTP/1.1
< parámetros>
……. Cuerpo …….

GET ruta HTTP/1.1
< parámetros>
……. Cuerpo …….

PUT ruta HTTP/1.1
< parámetros>
……. Cuerpo …….

DELETE ruta HTTP/1.1
< parámetros>
……. Cuerpo …….

?????? ruta HTTP/1.1
< parámetros>
……. Cuerpo …….

Respuestas HTTP

HTTP/1.1 código y msg
< parámetros>
……. Cuerpo …….

Interfaz Uniforme y Aplicación de Servidor

- **Interfaz CRUD**
  - Interfaz de gestión del contenido de una BBDD con 4 primitivas
    - **C**reate: Crea un contenedor de datos en la BBDD
    - **R**ead: Lee el contenido del contenedor de la BBDD
    - **U**pdate: Actualiza el contenido del contenedor de la BBDD
    - **D**elete: Destruye el contenedor de datos de la BBDD
- Primitivas de HTTP (denominados métodos o verbos)
  - **Interfaz uniforme**: permite gestionar BBDDs remotamente con las 4 primitivas CRUD
    - **POST**: Crear un recurso en la BBDD del servidor
    - **GET**: Leer un recurso de la BBDD del servidor
    - **PUT**: Modificar un recurso de la BBDD del servidor
    - **DELETE**: Borrar un recurso de la BBDD del servidor
  - HTTP tiene además otros métodos o verbos
    - HEAD: Pide solo la cabecera al servidor
    - OPTIONS: Determinar qué métodos acepta un servidor
    - TRACE: Trazar proxies, caches, … hasta el servidor
    - CONNECT: Conectar a un servidor a través de un proxy
    - .....
- **Aplicación de servidor** o **del lado servidor** (server side application)
  - Programa con respuestas a medida de un interfaz basado en solicitudes HTTP
    - Basado fundamentalmente en las 4 primitivas del **interfaz uniforme**

Códigos de estado de respuesta del servidor

- Respuestas informativas (1xx)
  - 100 Continue: Continuar solicitud parcial
- Solicitud finalizada (2xx)
  - **200 OK:** Operación finalizada con éxito, recurso servido
  - **201 Created:** Recurso creado o actualizado con éxito
  - **206 Partial Content**: para uso con GET parcial
- Redirección (3xx)
  - 301 Moved Permanently: Recurso en otro servidor, actualizar el URL
  - 303 See Other: Envía la URI del documento de respuesta
  - 304 Not Modified: Cuando el cliente ya tiene los datos
- Error de cliente (4xx)
  - 400 Bad request: Comando enviado incorrecto
  - **404 Not Found**: Recurso no encontrado en repositorio
  - 405 Method Not Allowed: Método no soportado por este servidor
  - 409 Conflict: Conflicto con el estado del recurso
  - 410 Gone: Recurso ya no esta
- Error de Servidor (5xx)
  - **500 Internal Server Error**: El servidor tiene errores
- Ver: https://developer.mozilla.org/es/docs/Web/HTTP/Status

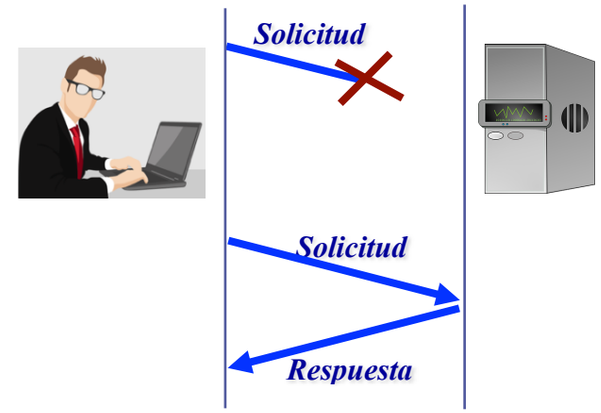
Tipos MIME

- **Tipo MIME**
  - Define el **tipo** de un recurso o fichero en email, Web, ...
    - Tiene la siguiente estructura: "**tipo/subtipo**"
      - Tipos MIME: https://developer.mozilla.org/ar/docs/Web/HTTP/Basics_of_HTTP/MIME_Types
      - Asociación de **extensiones de fichero** a tipos MIME: https://www.file-extensions.org
- Tipos MIME mas habituales para la Web
  - **Tipo text**
    - **text/plain**: Texto plano sin formato (**por defecto** si formato desconocido)
    - **text/html**: Texto plano con estructura definida con HTML
    - **text/css**: Fichero CSS
    - **text/javascript**: Script JavaScript
  - **Tipo image**
    - **image/gif**: Imagen en formato GIF
    - **image/jpeg**: Imagen en formato JPEG
    - **image/png**: Imagen en formato PNG
    - **image/svg**: Imagen en formato vectorial SVG
  - Envío de **parámetros de formularios** en el cuerpo (varios tipos)
    - **application/x-www-form-urlencoded**
      - Envío en body de parámetros de un formulario codificados en URL-encoded
    - **multipart/form-data**
      - Envío en body de parámetros de un formulario en múltiples formatos, incluso ficheros
  - Respuesta del servidor con **múltiples bloques y partes**
    - **multipart/byte-ranges**
      - El servidor envía la respuesta en varios bloques de octetos
  - .....

Seguridad e idempotencia

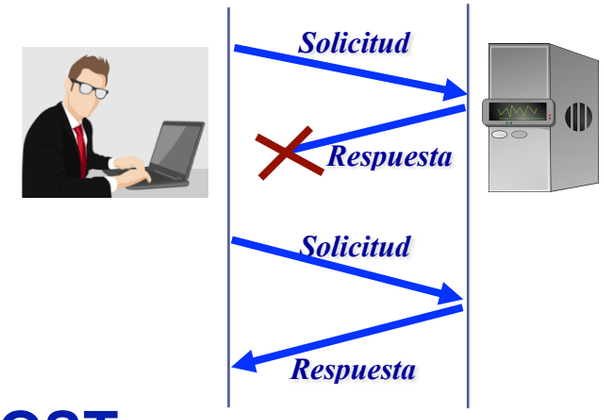
- Una transacción HTTP puede **ejecutarse** entre **1 o mas veces** en el servidor
  - Si todo va bien o si la **solicitud se pierde** y hay reenvío, la transacción **se ejecuta solo 1 vez**
    - Pero si la **respuesta se pierde** y hay reenvío, la transacción **se ejecuta mas de una vez** en el servidor

- **Idempotencia**
  - El resultado de ejecutar una primitiva es independiente del número de invocaciones
    - Por ejemplo: **x=2** es idempotente, pero **x=x+1** no es idempotente
  - Las operaciones asociadas a un interfaz REST deben ser idempotentes
    - Por ejemplo: **PUT /usuario/2007?edad=9**



- **Seguridad**
  - Un método es **seguro** si no modifica datos en el servidor
    - Eso permite servir la respuesta del servidor desde alguna **memoria cache** de la conexión

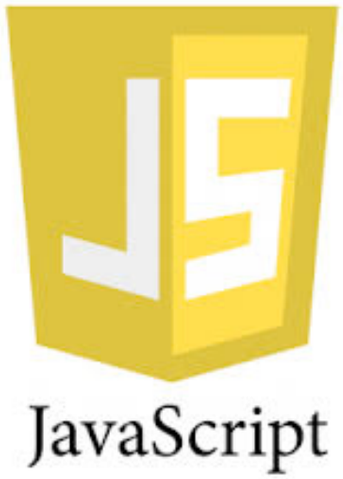
- Métodos del interfaz uniforme
  - POST: **El más peligroso** (puede duplicar recursos)
  - GET: **Seguro (cacheable) e idempotente**
  - PUT: **idempotente** (si está bien diseñado)
  - DELETE: **idempotente**



- Se debe tratar de minimizar el impacto de la **no idempotencia** de **POST**
  - Además, no se debe utilizar **nunca GET** para **modificar recursos** del servidor
    - Utilizar **POST**, **PUT** o **DELETE** para cada tipo de **modificación**, porque GET puede ser cacheado y no modificará los recursos

REST - **RE**presentational **S**tate **T**ransfer

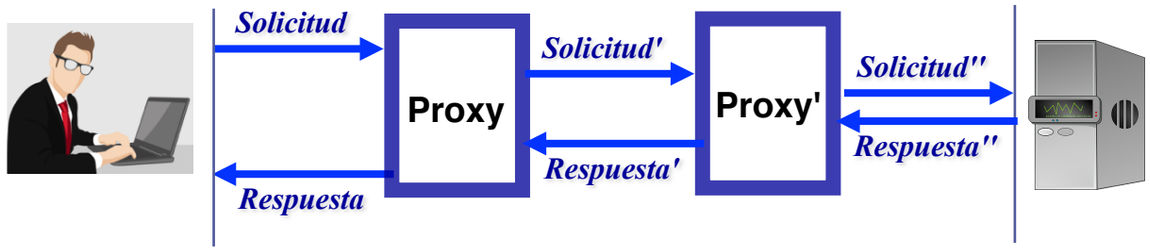
- **REST**
  - **Metodología** de desarrollo de **aplicaciones hipermedia distribuidas** (Roy Fielding 2000)
    - http://en.wikipedia.org/wiki/Representational_state_transfer, https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representaciona
- Aplicación **Web hipermedia** basada en **REST**
  - Utiliza una arquitectura orientada a **recursos** (guardados normalmente en un servidor)
    - Cada recurso tiene una **dirección (URL)** única (diferente de cualquier otra en Internet)
  - Utiliza un protocolo **cliente-servidor sin estado** (HTTP)
    - Cada solicitud tiene la información necesaria para elaborar la respuesta (sin necesidad de conocer las anteriores)
    - Las respuestas usan un lenguaje universal (HTML, JSON, XML) y contienen todo el estado del cliente
  - **Cliente** y **servidor** se comunican con **interfaces** basados en las **4 primitivas CRUD**
    - El **interfaz REST** son las primitivas HTTP utilizadas por el cliente para acceder a los recursos del servidor
  - La aplicación es **hipermedia** porque usuario interacciona con **clics** en **hiperenlaces** y **botones**
    - Los clics actualizan el recurso (indentificado por el URL) en la BBDD del servidor y devuelven el nuevo estado (página HTML, ...)
- Ejemplo de **Interfaz REST**
  - **GET /clientes** trae toda la lista de clientes del servidor
  - **GET /clientes/221** trae el registro del cliente 221
  - **PUT /clientes/221?name=Pepe+Díaz&age=23** actualiza el registro del cliente 221
  - **DELETE /clientes/221** borra el registro del cliente 22
  - **GET /productos** trae toda la lista de productos del servidor
  - **GET /productos/3** trae la descripción del producto 3
  - .....



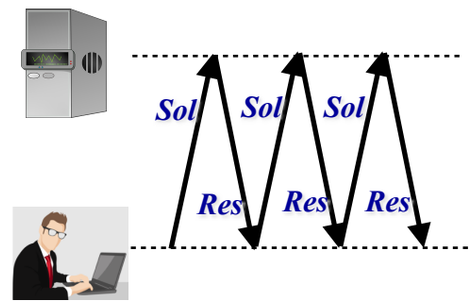
HTTP 1.1 (II):
conexión HTTP, proxy, conexión persistente y paralela, respuesta chunked, caches y CDN

Juan Quemada, DIT - UPM

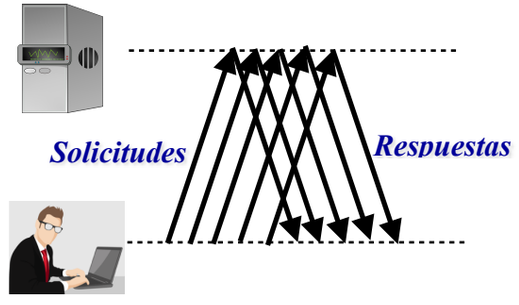
La Conexión HTTP



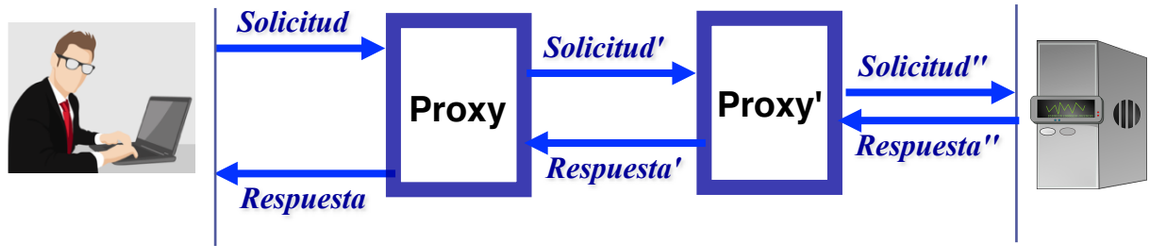
- Algunas **características** de la **conexión HTTP**
  - **Con proxy**: puede atravesar dispositivos intermedios
  - **Persistente**: puede mantenerse establecida a lo largo de varias transacciones
  - **Paralela** (pipelined): permite enviar solicitudes sin esperar a las respuestas
- **Dispositivo intermedio** (proxy)
  - Cada proxy recibe la solicitud y puede procesarla antes de reenviarla al siguiente
    - Esto permite crear nuevas funcionalidades, como **proxy Web**, **cache**, **firewall**, …
  - La conexión se establece con el **primer proxy**, este con el siguiente y así hasta llegar al servidor
    - HTTP no necesita hoy conectividad TCP extremo a extremo, algo que ya no existe en la Internet actual
      - Ver: https://en.wikipedia.org/wiki/Proxy_server
- **Persistencia**
  - El parámetro **Connection** solicita mantener la conexión o cerrarla
    - **Connection: keep-alive** solicita que la conexión permanezca abierta
    - **Connection: close** solicita cerrar la conexión HTTP al finalizar la transacción



- **Paralelismo** (pipelining)
  - Una solicitud puede enviarse sin haber recibido la respuesta a la anterior
    - Las respuestas deben llegar en el mismo orden en que se enviaron las solicitudes



Transfer-encoding: chunked



- **Transfer-encoding**
  - Parámetro HTTP que indica la codificación para el salto en curso
    - Ver: https://developer.mozilla.org/es/docs/Web/HTTP/Headers/Transfer-Encoding
  - Tipos de codificaciones
    - **Transfer-encoding: chunked** - Envío de body como secuencia de **fragmentos**
    - **Transfer-encoding: identity** - Envío de body **sin codificación**
    - **Transfer-encoding: gzip** - Envío de body codificado con **gzip**
    - ...
  - Ojo! Además existe el parámetro **Content-encoding**
    - Es el que indica la codificación extremo a extremo utilizada por HTTP

Ejemplo de respuesta chunked:

HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked

8\r\nEste es \r\n

3\r\nun \r\n

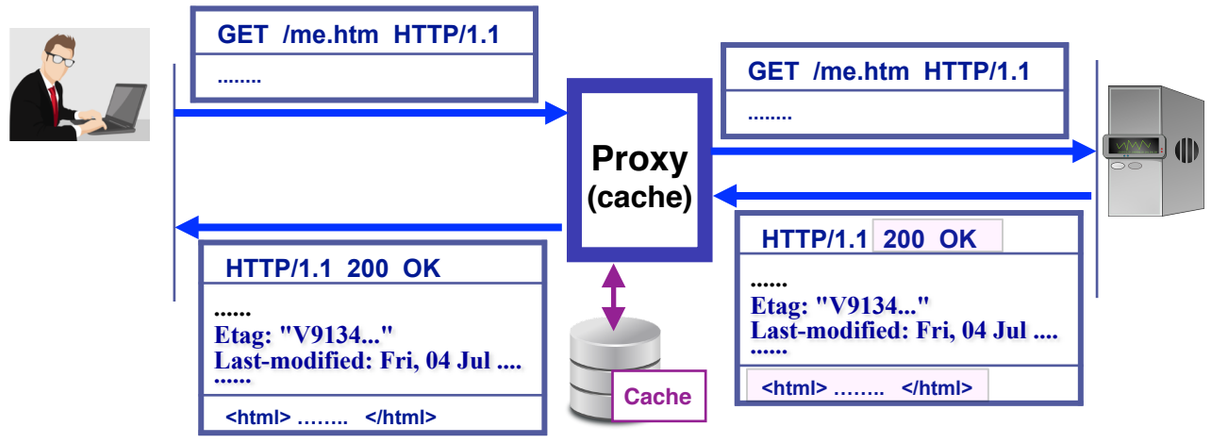
F\r\nmensaje partido \r\n

14\r\nen cuatro fragmentos\r\n

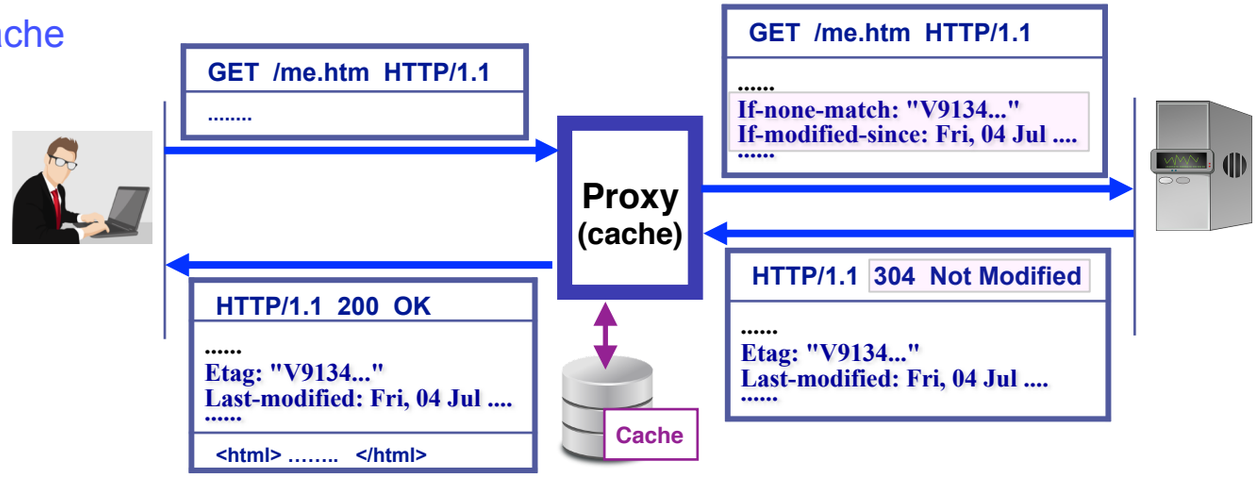
0\r\n\r\n

- **Transfer-encoding: chuncked**
  - Se usa para enviar ficheros, imágenes, ... de **longitud desconocida**
    - Permite enviar fragmentos de body, a medida que se van obteniendo
      - La respuesta termina cuando llega un fragmento vacío
  - Este modo **no** es compatible con el parámetro **Content-length**
- Cada fragmento envía primero su longitud en hexadecimal, seguida de **'\r\n'**
  - Después viene el contenido del fragmento, seguido también de **'\r\n'**
    - Puede haber tiempos de espera (sin envíos) entre fragmentos
- El final de envío se indica con un fragmento vacío
  - Es decir, **'0\r\n\r\n'**

La Cache



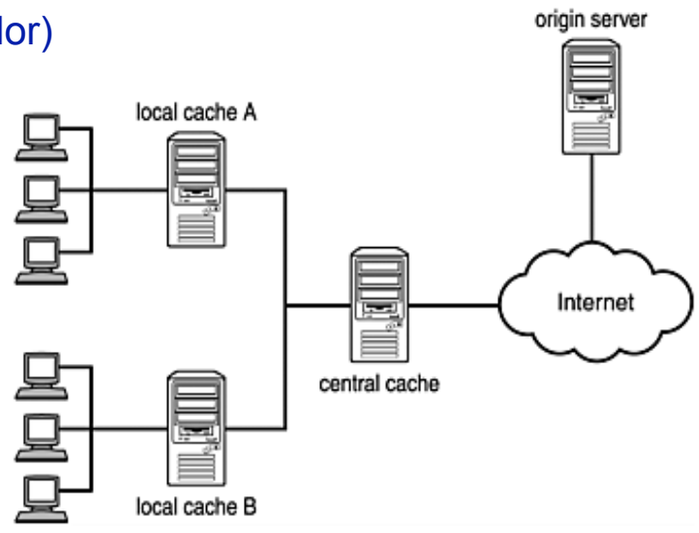
- Una **cache** almacena todos los **recursos cacheables** que bajan de un servidor
  - Guarda el **recurso**, su **URL** (servidor y ruta) y los parámetros **Etag y Last-modified**
    - Al llegar una **nueva solicitud** comprueban si la cache tiene el recurso a través del **URL** (servidor y ruta),
- Si el recurso **no está** en la cache, se reenvía la solicitud sin modificarla
  - La respuesta **traerá el recurso solicitado** (del servidor u otra cache)
    - La respuesta se reenvía al cliente sin modificar
    - El nuevo recurso se guarda en el repositorio de la cache



- Si el recurso **está** en la cache
  - La solicitud se reenvía **añadiendo**
    - **If-none-match: "V9134..."**
    - **If-modified-since: Fri, 04 Jul...**
- Si la respuesta es **304 Not Modified**
  - La cache envía el recurso guardado al cliente con **200 OK** sacándolo de su propio repositorio
    - La respuesta **304 Not Modified** no incluye el recurso (body) y reduce el tráfico de bajada del servidor a la cache
- Si la respuesta es **200 OK**, el recurso debe haber cambiado en el servidor
  - La respuesta se reenvía al cliente sin modificar y el recurso se guarda de nuevo en la cache

Caches y Distribución de Contenidos

- Internet es hoy un **gran repositorio de información** (Web)
  - El tráfico mayoritario son consultas (primitivas GET, ...) a los recursos mas populares
    - Videos, películas, grandes ficheros, fotos, librerias, páginas Web, …
- **Cache**
  - Proxy que almacena recursos cacheables o seguros (descargados con GET)
    - Sirve el recurso directamente al cliente (si este no ha sido modificado en el servidor)
- El tráfico Web se optimiza con **caches**
  - Las caches se sitúan en puntos estratégicos
    - En el navegador
    - En el punto de entrada a una organización
    - En los puntos de conexión entre redes o proveedores
    - Delante del servidor, para descargar su trabajo
    - .....
  - El recurso se envía al cliente desde la cache mas cercana
    - Así se evita tener que traer el recurso desde su servidor y se reduce el tráfico en la red

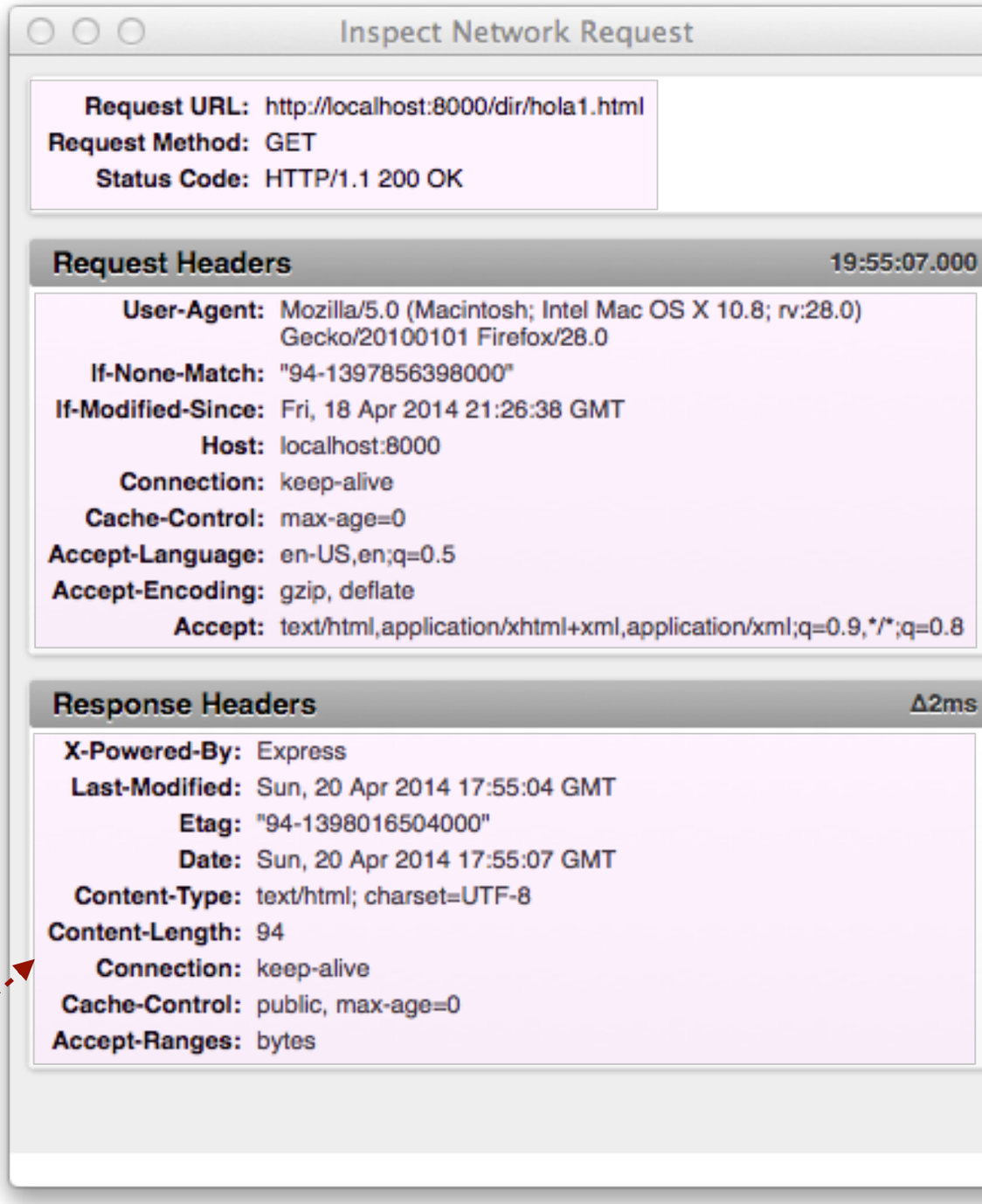


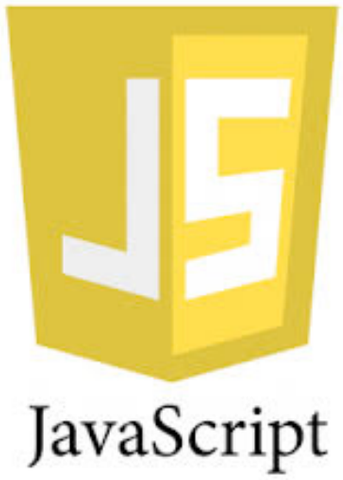
- **CDN - Content Distribution Network**
  - Internet tiene muchas caches en puntos estratégicos para reducir el tráfico
    - https://en.wikipedia.org/wiki/Content_delivery_network
  - Existen CDNs libres y empresas especializadas en distribuciones masivas
    - Por ejemplo, para estrenos de **nuevas películas**, para distribución de **nuevas versiones de iOS**, ...

Ejemplo de transacción

La consola de red de Firefox muestra una transacción GET.
Al hacer click en ella, el navegador muestra los detalles de la carga de una página Web desde un servidor static de express:

- **Status 200 OK**: respuesta incluye recurso
- **Content-Type: text/html; charset=UTF-8**
  -> indica página **HTML en UTF-8**
- **Content-Length: 95**
  -> Indica que el recurso descargado (body) tiene **95** octetos
- **Connection: keep-alive**
  -> Cliente y servidor solicitan mantener conexión
- **If-None-Match: .....**
- **If-Modified-Since: .....**
  -> Indican al servidor que la cache del navegador tiene el recurso
- **Etag: .....**
- **Last-Modified: .....**
  -> El servidor devuelve 200 OK, indicando Etag y fecha del nuevo recurso





Final del tema