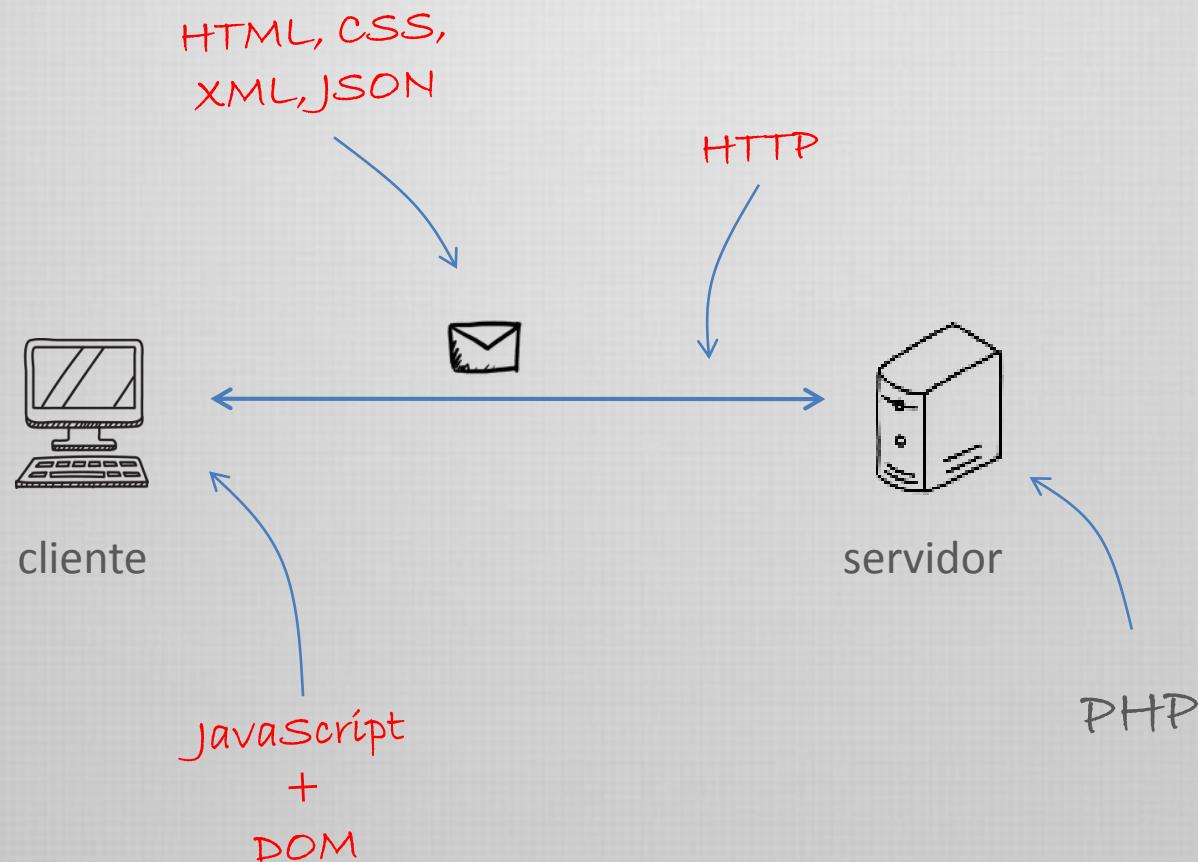


Aplicaciones Web

Diego C. Martínez

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Tecnologías web





modelo de **interacción** para aplicaciones web,
basado primordialmente en dos tecnologías:

JavaScript y XML

AJAX

Asynchronous JavaScript and XML

Es básicamente una técnica para implementar ciertos comportamientos en aplicaciones web.

La idea general es solicitar en background información a un servidor mientras el cliente opera sobre el navegador (por eso asincrónica).

*Antes
"Inner-Browsing"*

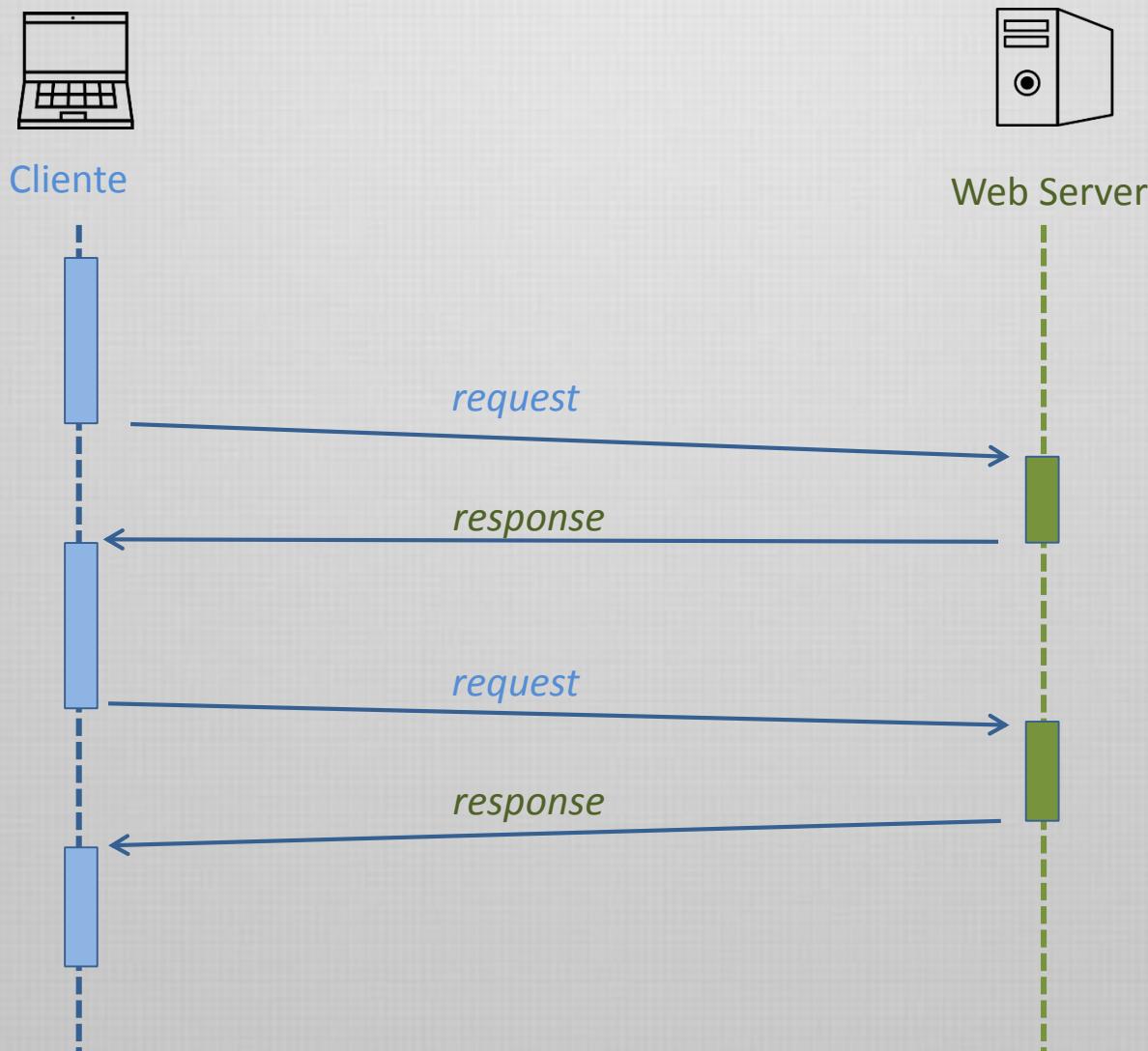
no es una tecnología, sino una agrupación de varias técnicas y tecnologías unidas para lograr un mismo efecto.



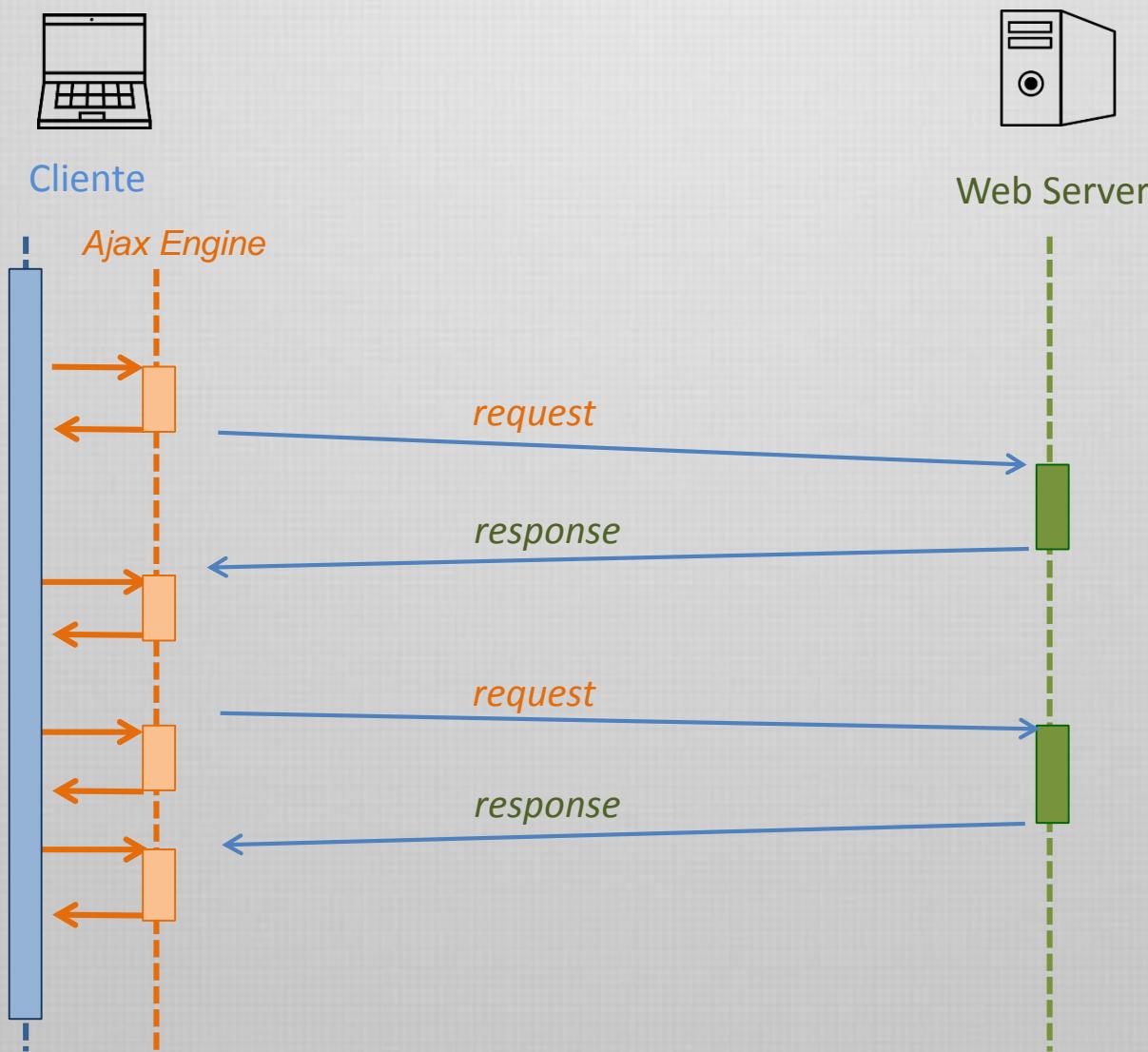
- Presentación basada en estándares: *XHTML, CSS*.
- Páginas dinámicas, utilizando *DOM*.
- Intercambio y manipulación de datos *XML* y *XSLT* (*y JSON!*)
- Recuperación asincrónica de datos, usando *XMLHttpRequest*



Modelo tradicional



Modelo Ajax



XMLHttpRequest

```
// IE7+, Firefox, Chrome, Opera, Safari, el mundo文明izado  
xmlhttp=new XMLHttpRequest();  
  
// IE6, IE5  
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
```

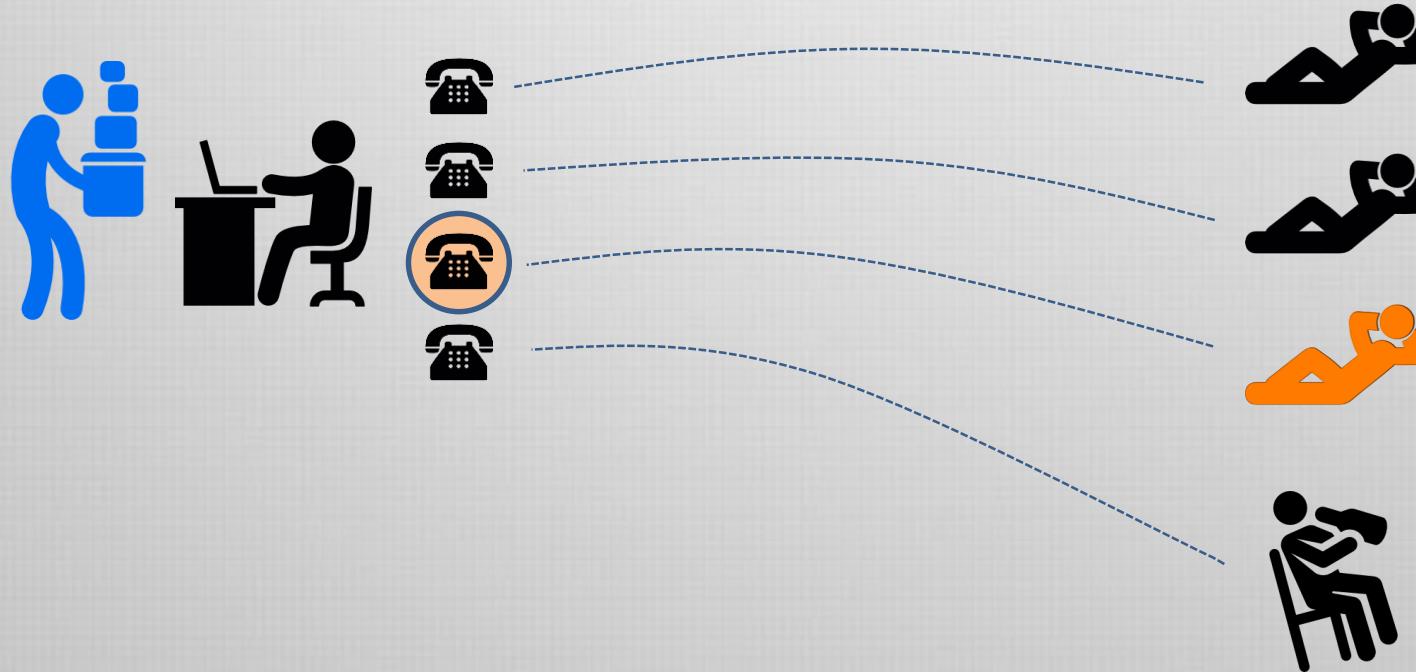


```
xmlhttp.open(metodo,URL,asincronico);  
xmlhttp.send();
```

Si el request es **asincrónico**, hay que indicar previamente qué sucederá cuando se complete la recepción de la respuesta del servidor...

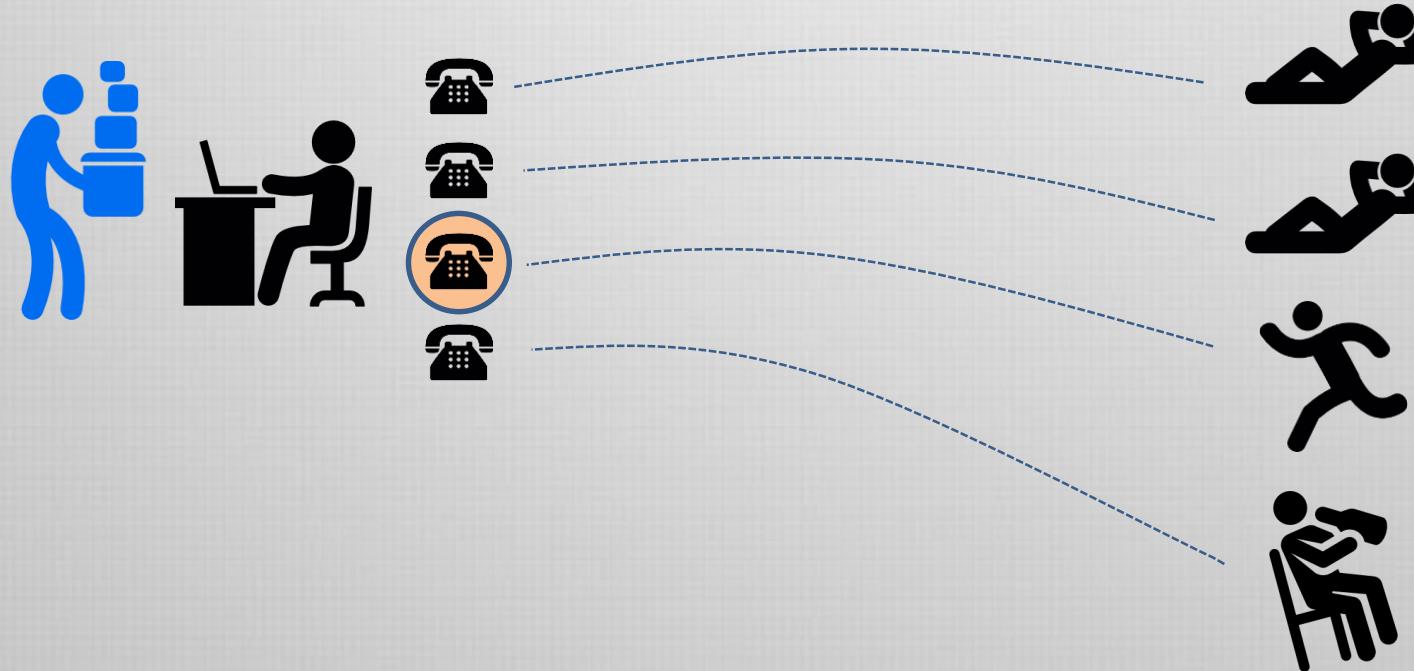
Call-back

Las funciones **call-back** son funciones que se invocan ante la ocurrencia de ciertos eventos.



Call-back

Las funciones **call-back** son funciones que se invocan ante la ocurrencia de ciertos eventos.



XMLHttpRequest en acción

```
req.onreadystatechange = processReqChange;  
req.open("GET", url, true);  
req.send();
```

onreadystatechange es una propiedad del objeto XMLHttpRequest que almacena la función que procesa la respuesta del servidor.

Es el oyente de cambio de estado: la función se invoca cada vez que cambia la propiedad **readyState**.

- después de invocar a *open*, **readyState** = 1.
- después de invocar a *send*, **readyState** = 2.
- mientras se recibe la respuesta, **readyState** = 3
- al finalizar la respuesta **readyState** = 4.

!La respuesta está lista para procesar!

XMLHttpRequest en acción

```
req.onreadystatechange = processReqChange;  
req.open("GET", url, true);  
req.send();
```

La respuesta del servidor se almacena finalmente en un atributo del objeto XMLHttpRequest.

Si la respuesta esperada es *texto plano*, se almacena en **responseText**

```
function processReqChange () {  
    if(req.readyState == 4){  
        document.getElementById(id).innerHTML = req.responseText;  
    }  
}
```

XMLHttpRequest en acción

```
req.onreadystatechange = processReqChange;  
req.open("GET", url, true);  
req.send();
```

El atributo **responseXML** es similar, pero nos permite encapsular la información para ser interpretada como XML.

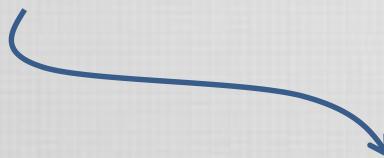
Al ser un XML, tiene una representación DOM y podemos usar operaciones para obtener los datos contenidos en ese documento!

```
function processReqChange () {  
    if(req.readyState == 4){  
        var xmldoc = req.responseXML;  
        var nodo = xmldoc.getElementsByTagName('root').item(0);  
    }  
}
```

XMLHttpRequest en acción

```
req.onreadystatechange = processReqChange;  
req.open("GET", url, true);  
req.send();
```

Las operaciones **open** y **send** inician el proceso de pedido al servidor.



recibe como parámetros:

el **método** (GET, POST, HEAD, etc)

+

el **URL** (que puede ser absoluto o relativo) y

+

un indicativo de si el pedido es **asincrónico o no**.

El valor “true” como último argumento indica que el script no esperará la respuesta luego de la invocación a la operación send, la cual comienza el pedido antes configurado

Ejemplo Ajax

Con lo visto hasta el momento es fácil crear una simple aplicación que utilice esta técnica.



Del lado cliente, haremos **pedidos HTTP** a un servidor por medio del objeto **XMLHttpRequest**. Recibiremos la información y la procesamos.



Del lado servidor, una aplicación (por ejemplo Java) o script (por ejemplo PHP) generará la respuesta solicitada por el cliente.

Los dos extremos colaboran creando el efecto deseado.

Es importante destacar que las tecnologías usadas tanto del lado cliente como del lado servidor son **independientes** y por lo tanto pueden cambiar.

Ejemplo Ajax – validación de datos



onkeyup

```
validateUserID () {  
    // controlar id  
}
```



JavaScript

XMLHttpRequest

```
function callback () {  
    //actualizar DOM  
}
```

validar.php?userid=...



Server de Validación

JSON

La tendencia es utilizar JSON como estructura de datos, en lugar de XML:

```
<albums>
  <album>
    <titulo>
      Master of Puppets
    </titulo>
    <artista>
      Metallica
    </artista>
  </album>
  <album>
    <titulo>
      The Division Bell
    </titulo>
    <artista>
      Pink Floyd
    </artista>
  </album>
<albums>
```

171 bytes

```
{
  'album':
  [
    {
      'titulo' : 'Master of Puppets',
      'artist' : 'Metallica',
    },
    {
      'titulo' : 'The Division Bell',
      'artist' : 'Pink Floyd',
    }
  ]
}
```

120 bytes

La principal ventaja es que al ser un formato de datos propio de JavaScript, es fácil interpretarlo. No hace falta parsing como en XML.

Algunos usos de Ajax

Validaciones de datos de formularios en tiempo real.

Identificaciones de usuario, números de serie, códigos postales, y otros datos pueden ser validados en el formulario antes del envío de los datos.

Autocompletamiento

Algunos datos pueden ser autocompletados mientras el usuario escribe.

Controles de interfaz de usuario

Arboles, menúes, barras de progreso se implementan sin necesidad de refresh de página completa.

Refresh de datos en la página.

Algunos datos son obtenidos del servidor y mostrados en la página.

Algunas desventajas de Ajax

Complejidad

Se necesita lógica de presentación en el cliente y generación y envío de documentos XML en el servidor. Requiere conocimientos de JavaScript.

Depuración

El proceso es complicado porque la lógica se distribuye en el cliente y en el servidor.

Código expuesto

Parte del código de la aplicación queda expuesto al ejecutarse en el cliente.

Indexado

Muchos web crawlers no encontrarán el contenido obtenido via AJAX.

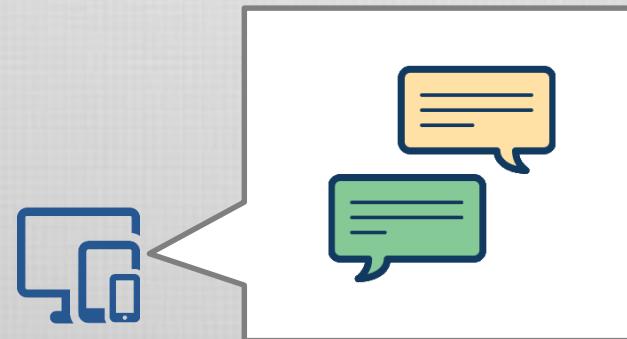
Bookmarks

Probablemente los *bookmarks* no registren el último estado de la página

Chat

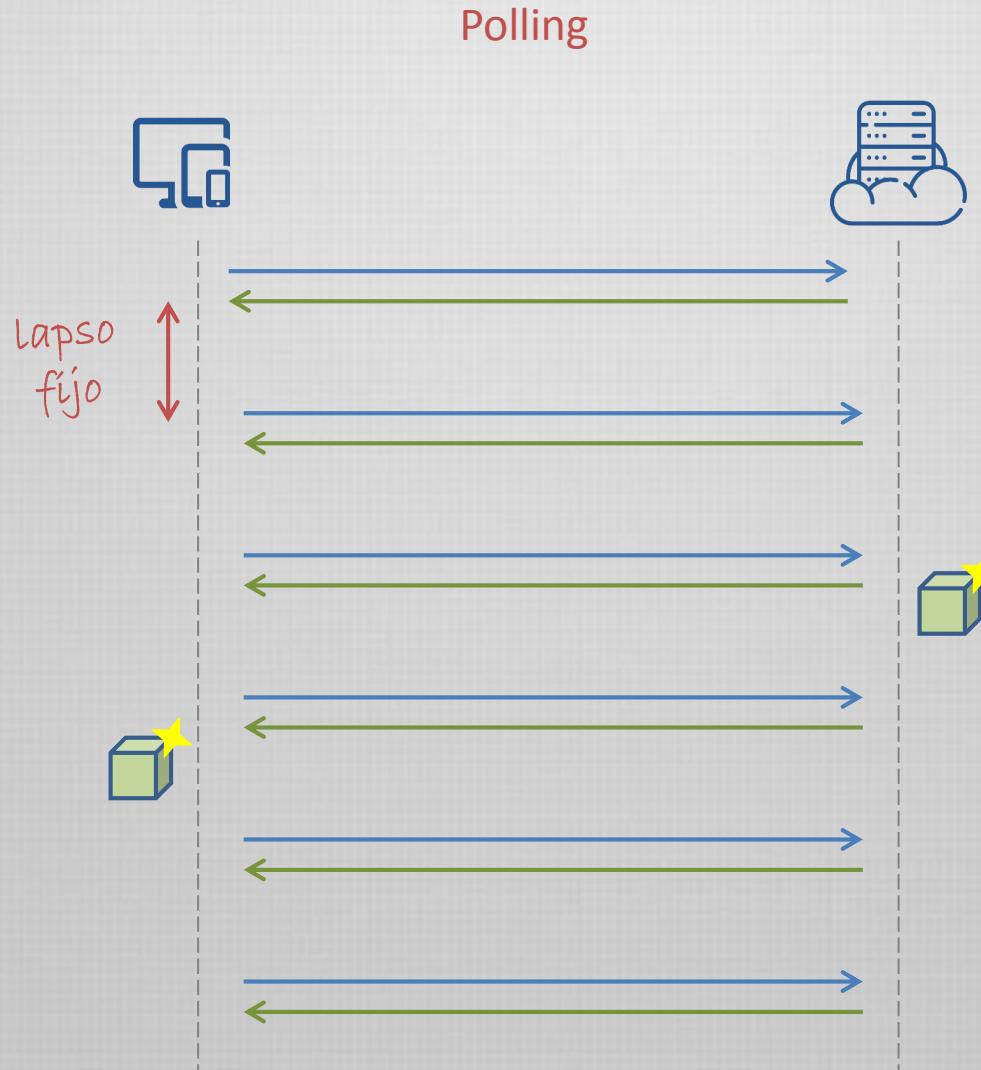


¿Como podríamos implementar un chat con esta técnica?

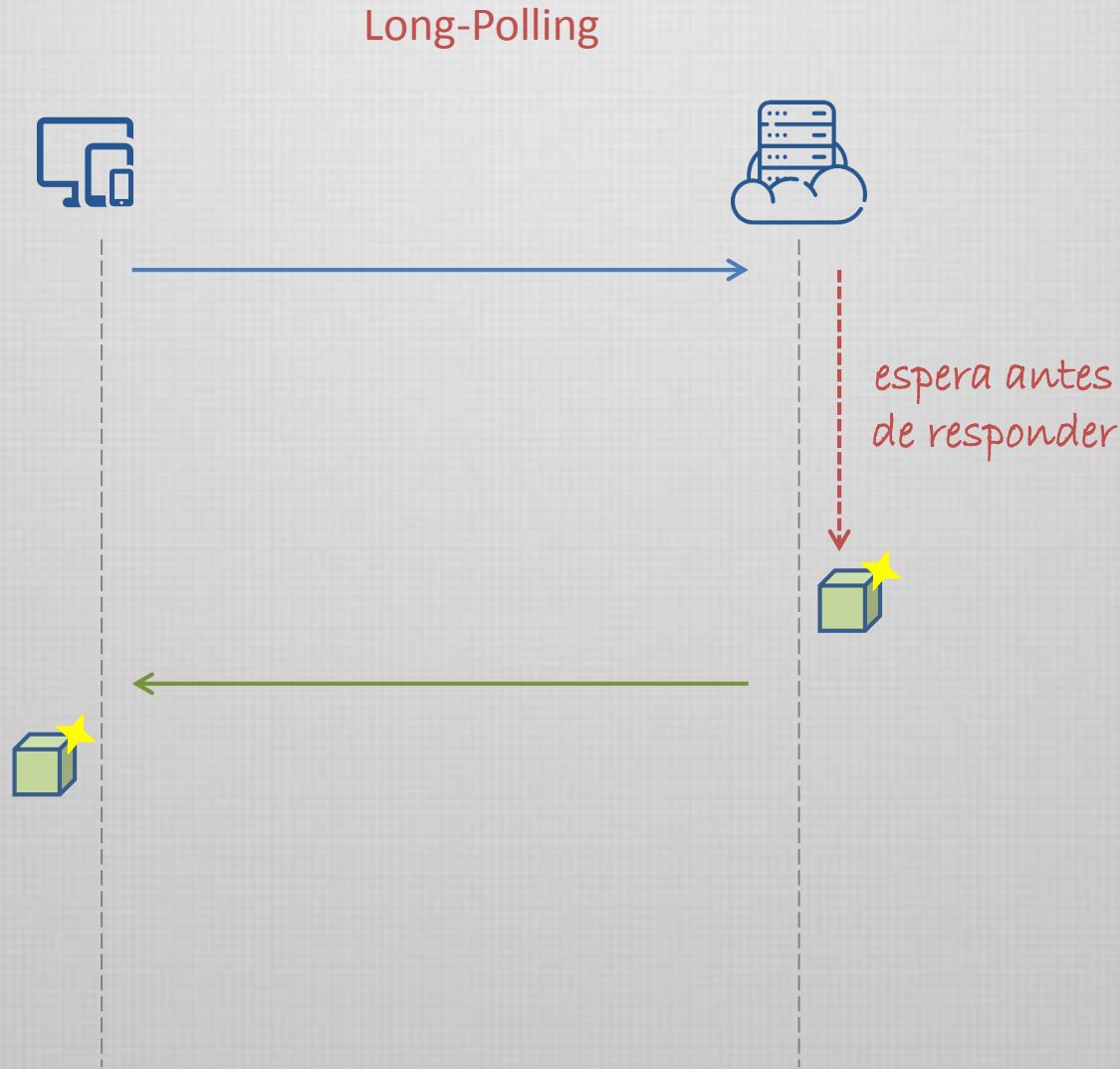


*Polling
Long Polling
Server-Sent Events*

Polling, Long-Polling, Server-Sent Events

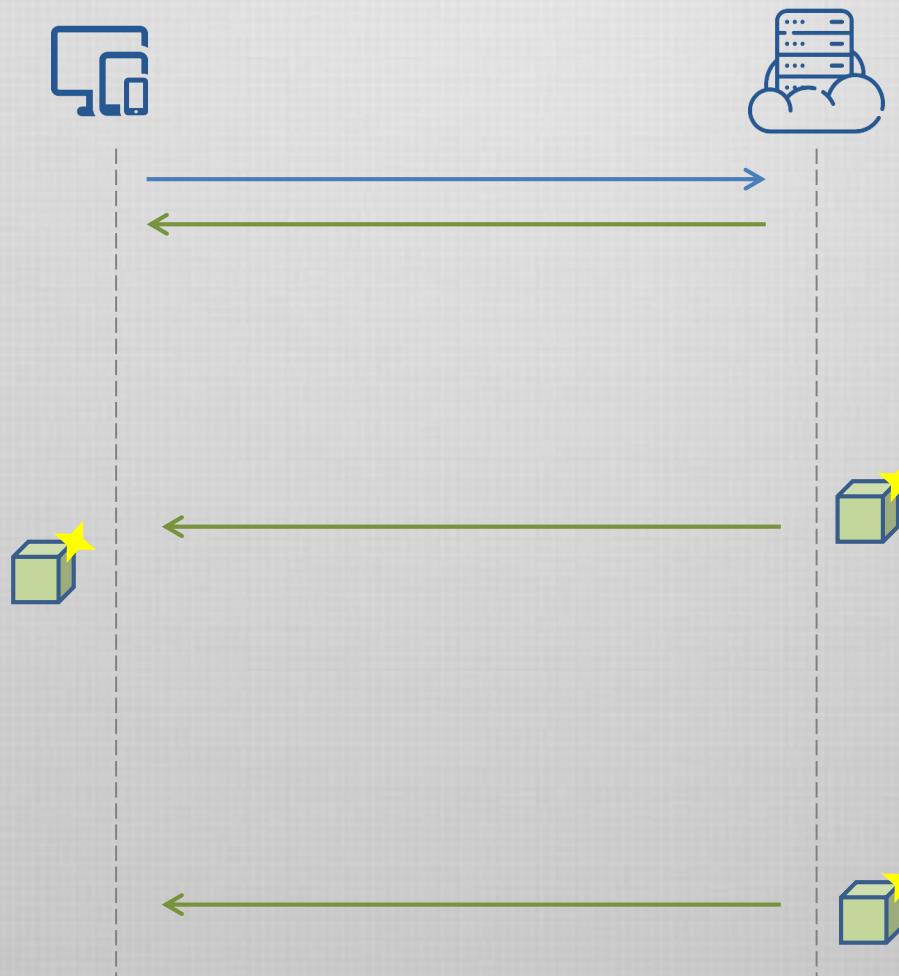


Polling, Long-Polling, Server-Sent Events



Polling, Long-Polling, Server-Sent Events

Server-Sent Event



Polling, Long-Polling, Server-Sent Events

Server-Sent Event



```
{ var source = new EventSource("server.php");
  source.onmessage = function(event) {
    document.getElementById("result").innerHTML += event.data + "<br>";
  };
}
```

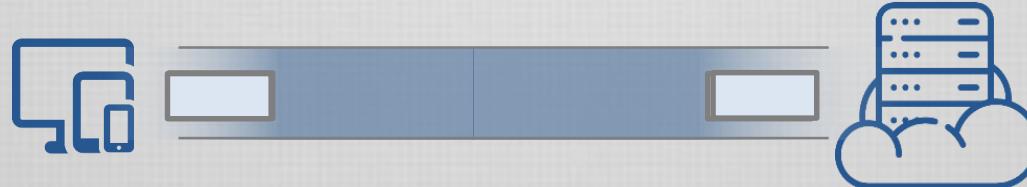


```
{ <?php
  header('Content-Type: text/event-stream');
  header('Cache-Control: no-cache');

  $time = date('r');
  echo "data: The server time is: {$time}\n\n";
  flush();
?>
```

Websockets

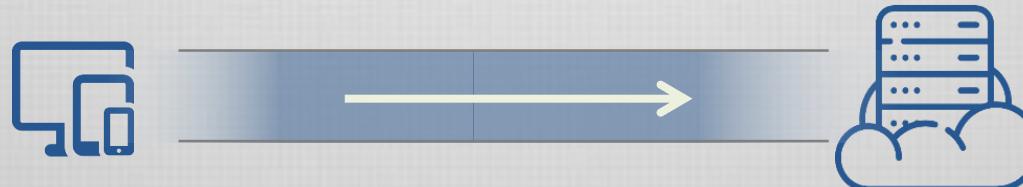
Tecnología que permite una comunicación **bidireccional** entre el navegador y el servidor



Representa una conexión aparte,
con otro protocolo de comunicación

Websockets

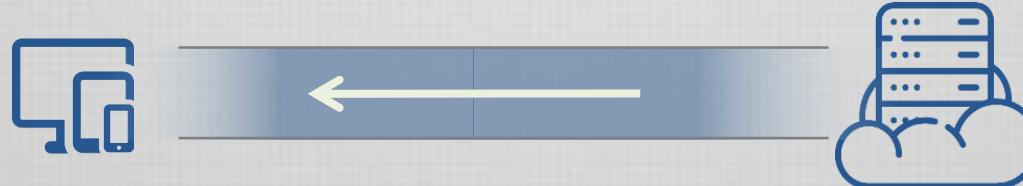
El primer paso es acordar la conexión websocket
“*handshake*”



GET /chat HTTP/1.1
Host: server.example.com
Upgrade: *websocket*
Connection: *Upgrade*
Sec-WebSocket-Key: *dfhJGg634gDGFH442fDUIK==*
Origin: *http://unsitio.com*
Sec-WebSocket-Protocol: *chat, superchat*
Sec-WebSocket-Version: 13

Websockets

El primer paso es acordar la conexión websocket
“handshake”



HTTP/1.1 101 Switching Protocols

Upgrade: *websocket*

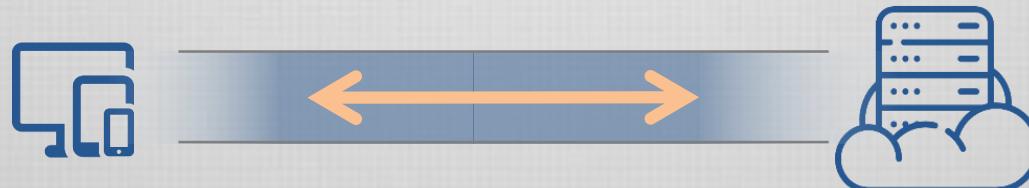
Connection: *Upgrade*

Sec-WebSocket-Accept: *sBiYGzTxaQ93pPLMkzhZRbK+xOo=*

Sec-WebSocket-Protocol: *chat*

Websockets

A partir del handshake se inicia la conexión full-duplex con websockets



Performance

Web Performance

Las aplicaciones web
son amenazadas por
ostentar una
performance
no óptima

- existe un procesamiento “remoto”
- comunicación cliente-servidor no bidireccional
- se solicitan solicita recursos adicionales al servidor
- el cliente debe procesar y mostrar parte de la información



La experiencia de uso se ve afectada naturalmente por
las tecnologías del cliente,
las tecnologías del servidor,
las condiciones de la red.

Web Performance

¿Por qué considerar
performance en
aplicaciones web?



Retención de usuarios

*La performance es esencial para
la experiencia del usuario*

*53% de los sitios móviles son
abandonados si la página tarda
mas de 3 segundos en cargar*

*El tiempo promedio de carga de sitios
móviles es de 19 segundos*

*Las que cargan en 5 segundos tienen
25% mas de publicidad observada
Sesiones un 70% más largas
35% menos de rebotes*

Web Performance

¿Por qué considerar performance en aplicaciones web?



Conversión de Usuarios

El usuario que se convierte en cliente



Algunos sitios incrementaron sus ventas al reducir el tiempo de carga

Para Mobify acelerar 100ms implica una ganancia de hasta \$380000 anuales

¿Por qué considerar
performance en
aplicaciones web?



Experiencia de Usuario (UX)

*Debe tenerse en cuenta la experiencia en general,
más allá de la oportunidad de negocio*

Diferentes dispositivos

Diferentes velocidades

Diferentes formas de uso del sitio



Web Performance

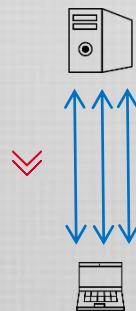
Si bien algunos de esos factores son difíciles (sino imposibles) de controlar, existen algunas reglas ampliamente aceptadas que se pueden seguir.

La mayoría de ellas centradas en el front-end y en la estructura de los datos.

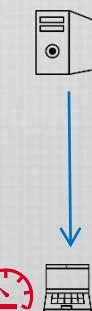
Diferentes autores identifican reglas variadas, aunque en general todas están destinadas a los mismos objetivos



Minimizar la cantidad de información transferida



Minimizar los actos de comunicación cliente-servidor



Mejorar el armado y visualización de los componentes en el navegador



Evitar usos excesivos de la red

Mas información...



Best Practices for Speeding Up Your Web Site
<http://developer.yahoo.com/performance/rules.html>



Web Performance Best Practices
http://code.google.com/speed/page-speed/docs/rules_intro.html



High Performance Web Sites
Steve Souders

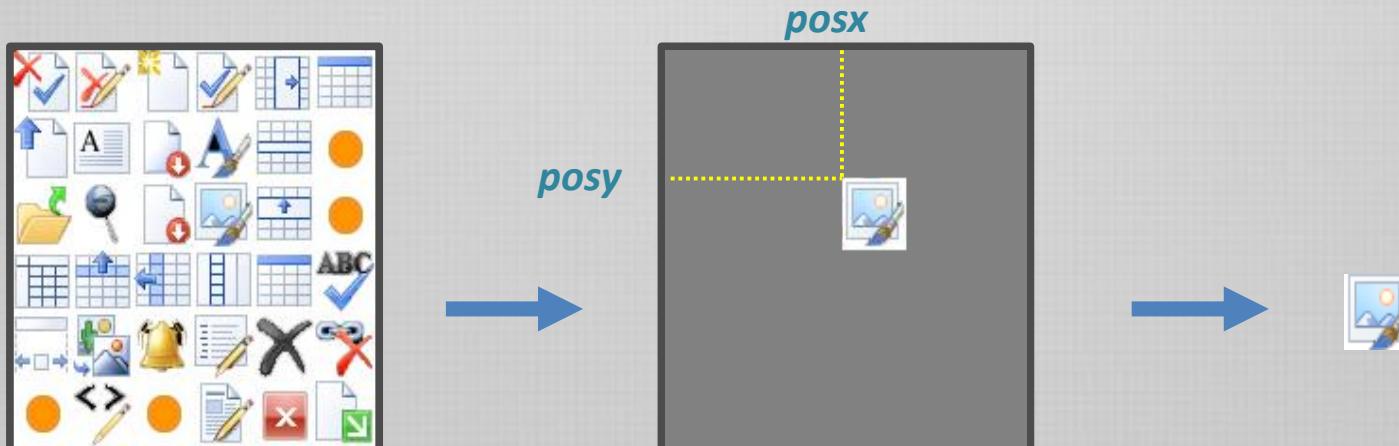
Regla: Minimizar pedidos HTTP

Hay varias técnicas que pueden ayudarnos a minimizar los pedidos HTTP

CSS Sprites

Una de las causas más comunes de la *abundancia de requests* es la carga de imágenes del documento.

Una técnica que minimiza estos pedidos es *CSS Sprites*. La idea es utilizar una sola imagen que es visualizada parcialmente.



Se deben especificar la posición de inicio (negativos) y el tamaño (positivos) como pares de pixels.

Regla: Minimizar pedidos HTTP

Imágenes inline

Es posible incluir el contenido de una imagen explícitamente en el documento HTML.
El atributo `src` del elemento `img` define la secuencia de bytes que conforman la imagen, en lugar de la dirección del recurso.

La imagen debe estar codificada en `base64`

Es especialmente útil cuando la imagen es pequeña.

```
<IMG SRC="data:image/gif;base64,R0lGODlhDAAMALMLAPN8ffBiYvWWlvrKy/FvcPewsO9VVfajo+w6O/z15estLv/8/AAAAAAAAAAAAAACH5BAEAAAsAIAAAAAAMAAwAAAQzcElZyryTEHyTUGknHd9xGV+qKsYirKkwDYiKDBiatt2H1KBLQRFIJAIKywRgmhwAII1EEADs=>
```

La imagen no podrá quedar en la cache y deberá ser retransmitida si se la necesita.

Puede localizarse también en el CSS

```
.img1 { background-image: url(data:image/gif;base64,AGEeVdFAdfASD...); }  
.img2 { background-image: url(data:image/gif;base64,GDEFghSAGJJJ...); }
```

Regla: Minimizar tiempos de carga

Ordenar apropiadamente los recursos CSS y Javascript

Los navegadores demoran el renderizado de algunas partes de la página (ya cargada) hasta que algunos scripts terminen de cargarse.

Depende de la estructura de la página.

```
<head>
<link rel="stylesheet" type="text/css" href="stylesheet1.css" />
<script type="text/javascript" src="scriptfile1.js" />
<script type="text/javascript" src="scriptfile2.js" />
<link rel="stylesheet" type="text/css" href="stylesheet2.css" />
<link rel="stylesheet" type="text/css" href="stylesheet3.css" />
</head>
```

```
<head>
<link rel="stylesheet" type="text/css" href="stylesheet1.css" />
<link rel="stylesheet" type="text/css" href="stylesheet2.css" />
<link rel="stylesheet" type="text/css" href="stylesheet3.css" />
<script type="text/javascript" src="scriptfile1.js" />
<script type="text/javascript" src="scriptfile2.js" />
</head>
```

Regla: Minimizar tiempos de carga

En general se desea que el navegador muestre progresivamente los contenidos de la página web solicitada.

funciona como "progress bar"



provee una referencia del sistema trabajando



provee una estimación de completitud de la tarea



provee algo para ver.

El *rendering progresivo* en el navegador se posterga hasta que los CSS estén completamente cargados.

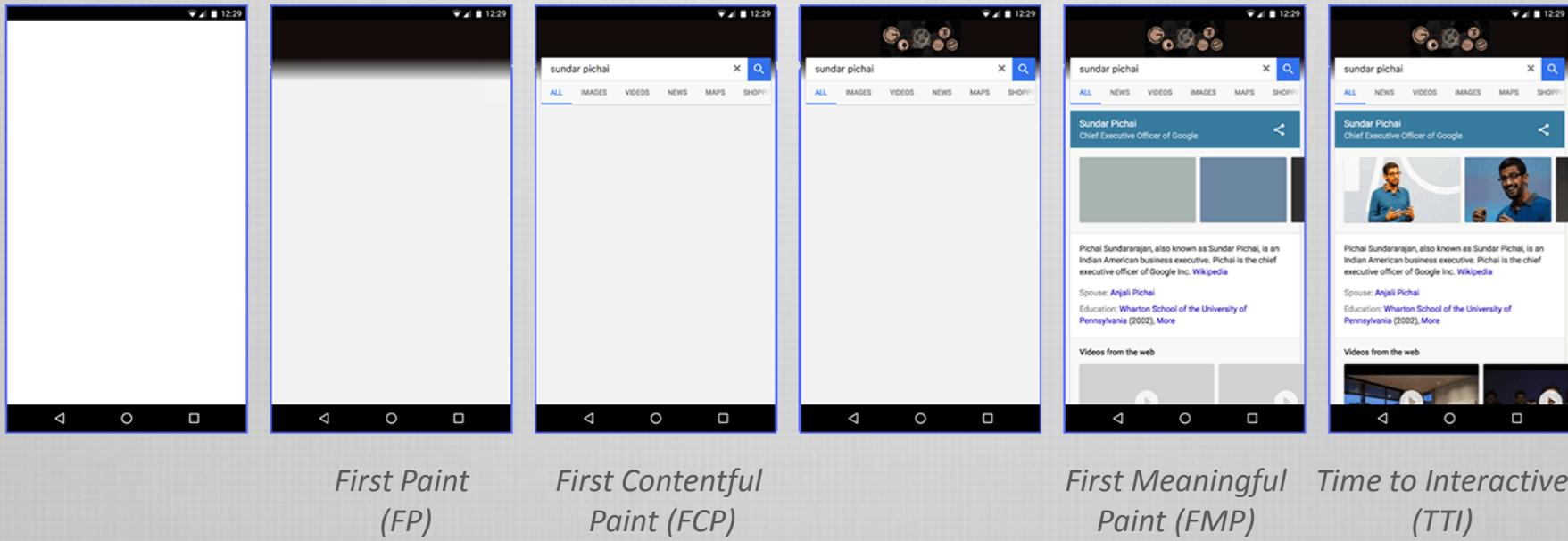
Si los stylesheets no están cargados todavía, es un desperdicio invertir en renderizar contenido pues deberá redibujarse nuevamente.

FOUC (Flash of Unstyled Content)

Por esa razón los CSS deben ser vinculados en el <head> del documento.

Regla: Minimizar tiempos de carga

*Time to interactive (TTI)
(renderizado y capaz de responder al input del usuario)*

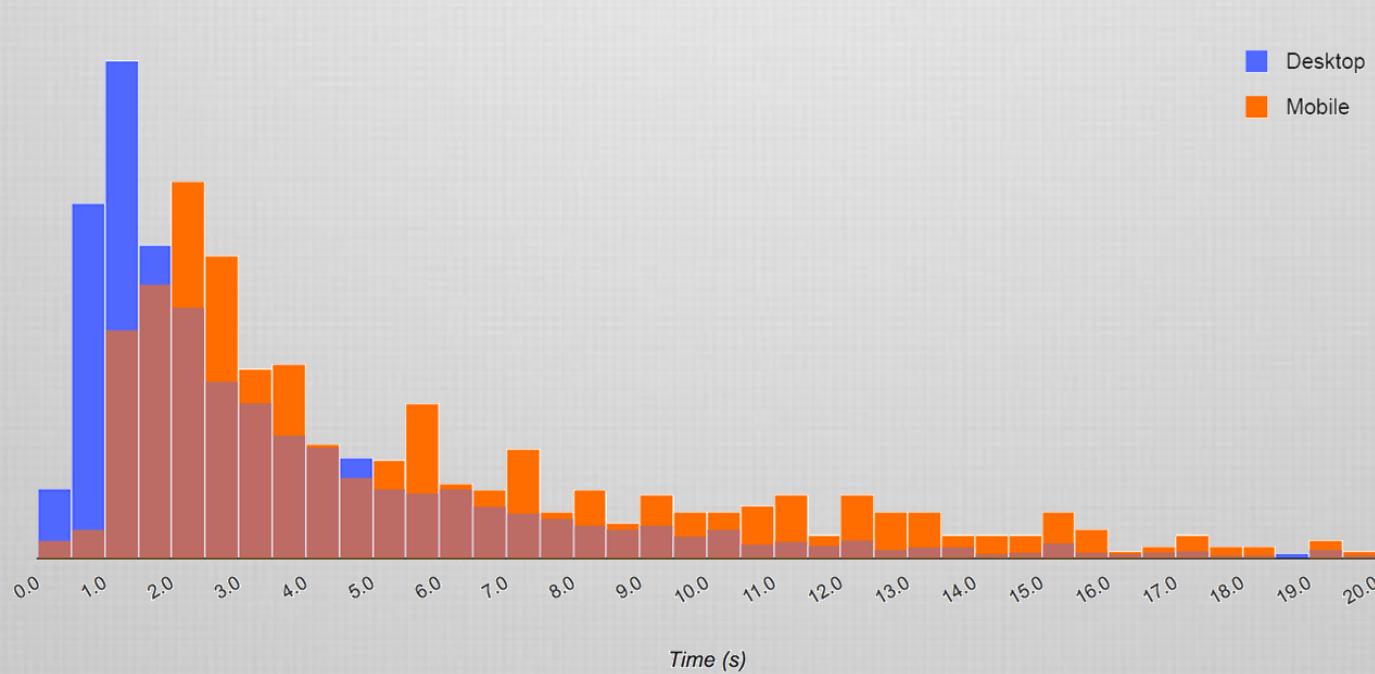


*First Paint
(FP)*

*First Contentful
Paint (FCP)*

*First Meaningful Time to Interactive
Paint (FMP) (TTI)*

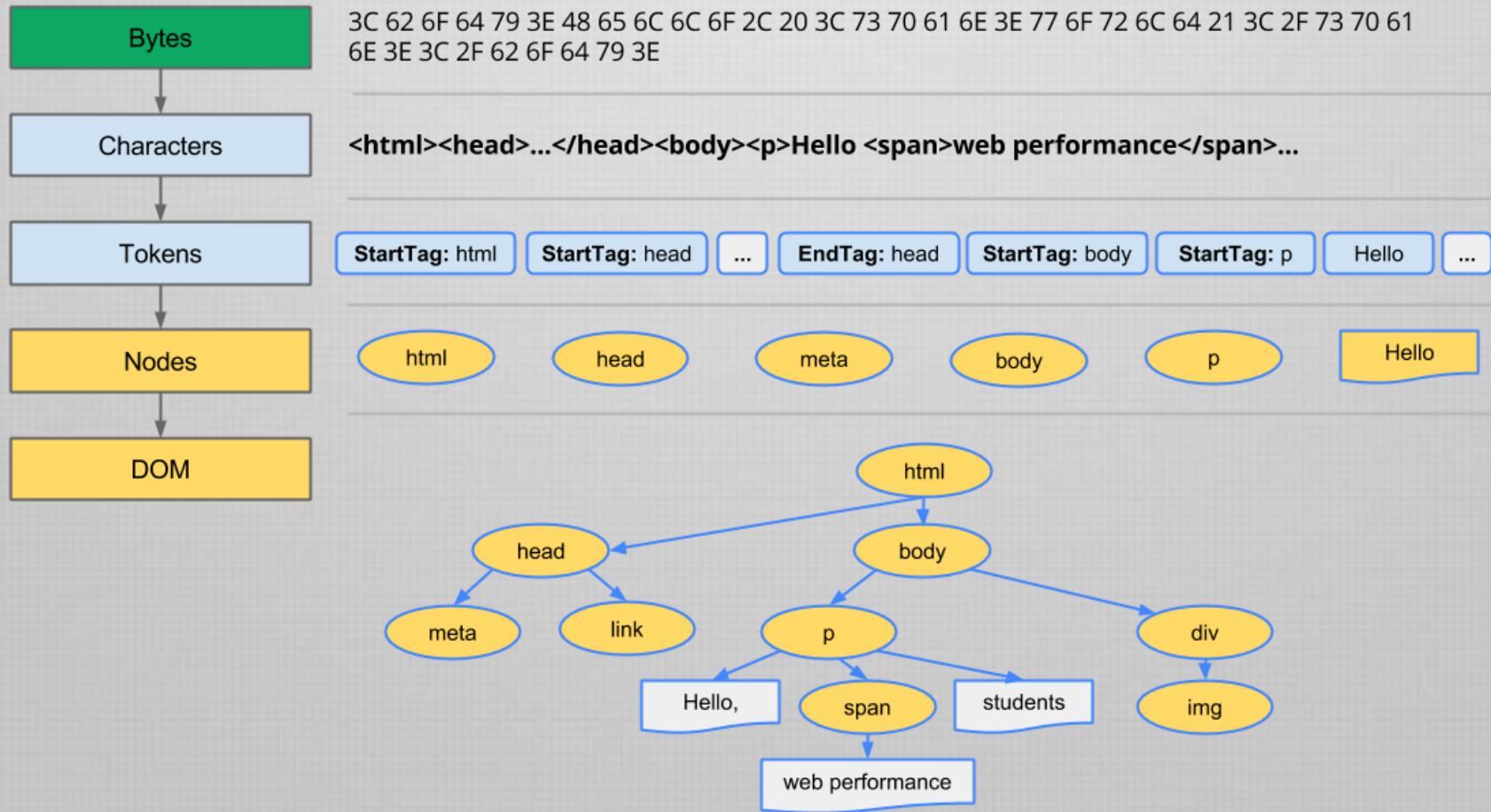
Regla: Minimizar tiempos de carga



*Time to interactive (TTI)
(renderizado y capaz de responder al input del usuario)*

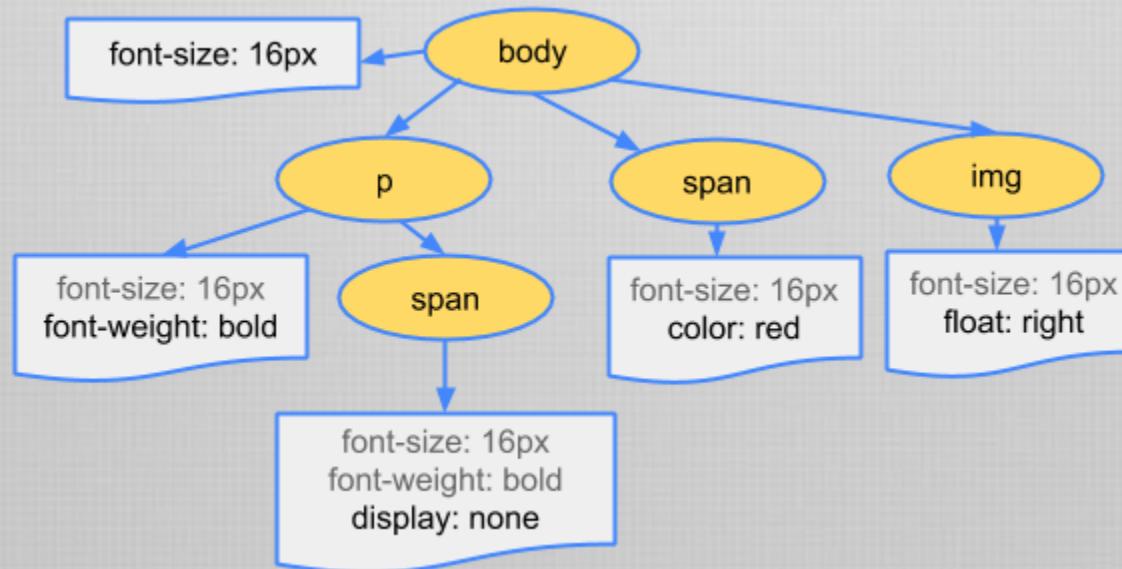
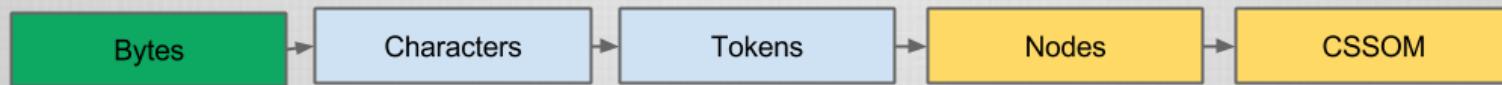
Critical Rendering Path

*Es el proceso que deriva en la renderización completa de la página
Crea dos estructuras básicas: DOM y CSSOM*



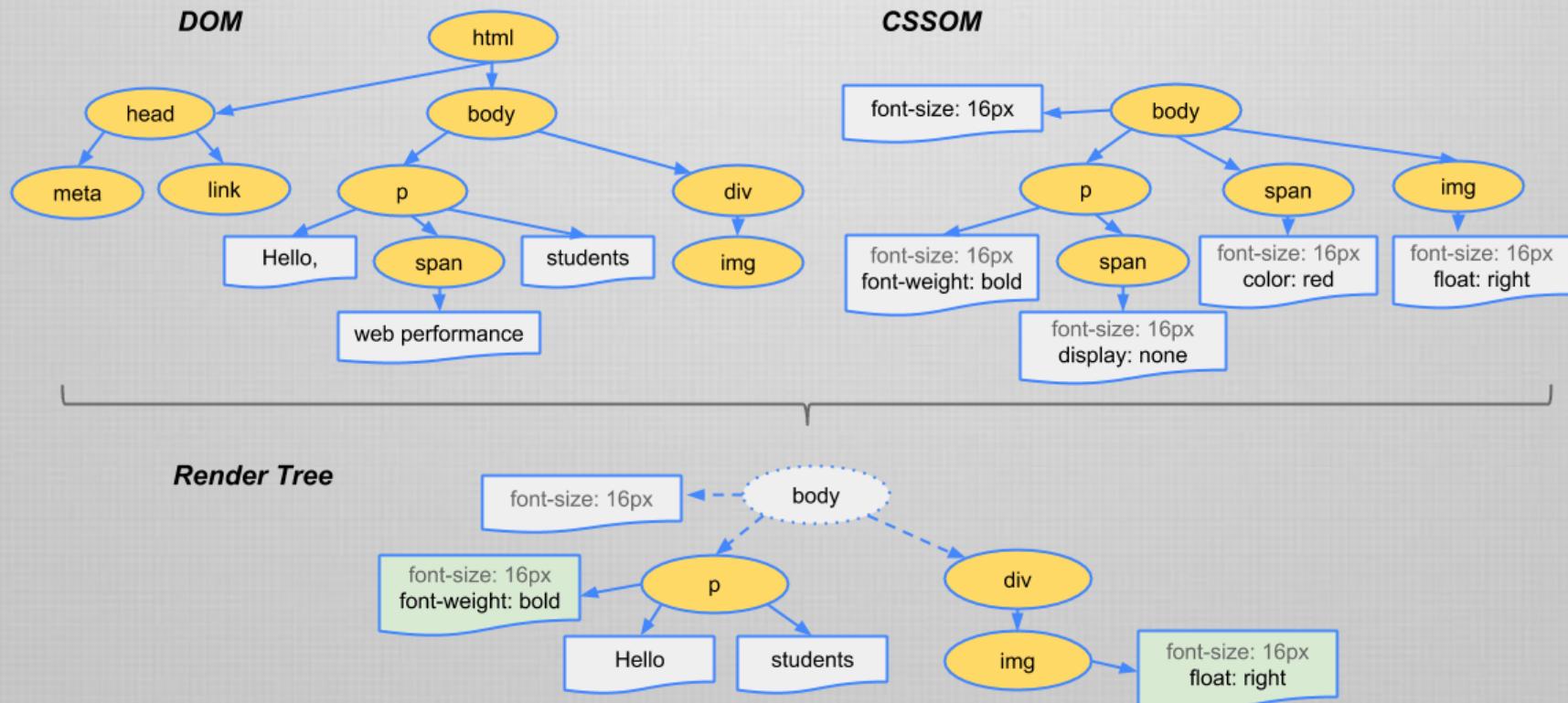
Critical Rendering Path

*Es el proceso que deriva en la renderización completa de la página
Crea dos estructuras básicas: DOM y CSSOM*



Critical Rendering Path

*Es el proceso que deriva en la renderización completa de la página
Crea dos estructuras básicas: DOM y CSSOM*



Regla: Minimizar tiempos de carga

En el caso de los scripts, el *rendering progresivo* se **posterga** para todo lo que esté “debajo” del script.

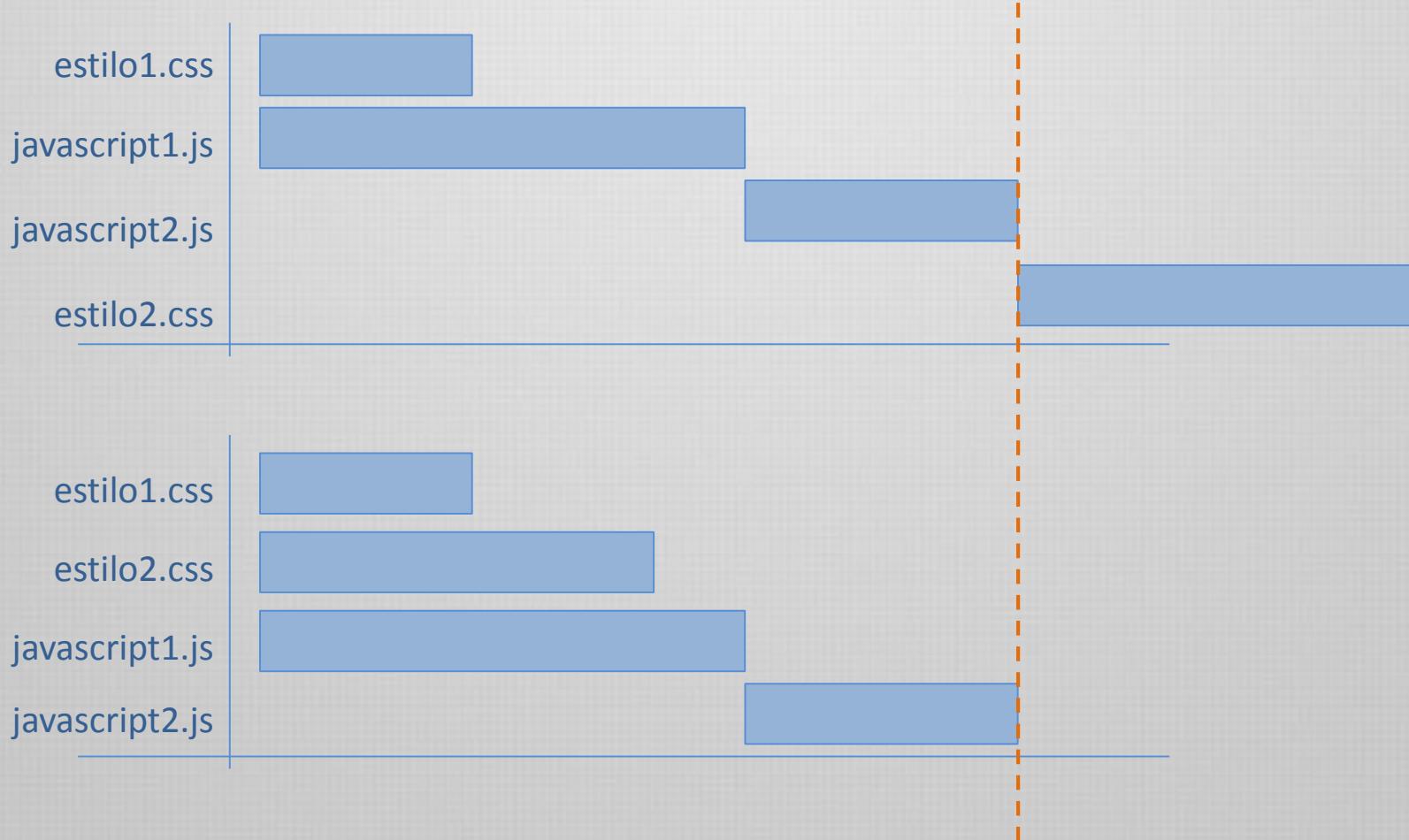
El estándar HTTP 1.1, *sugiere* que los clientes limiten la cantidad de conexiones simultáneas a un servidor:

A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy. (...) These guidelines are intended to improve HTTP response times and avoid congestion

Mas aún, cuando se **descarga un script**, no se abrirán nuevas conexiones.

Ubicar los CSS lo más arriba posible.
Ubicar los scripts lo más abajo posible.

Regla: Minimizar tiempos de carga



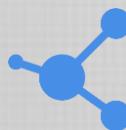
Regla: Minimizar tiempos de carga

El uso de JavaScript en una página web requiere



Cargar el código fuente

Implica transferir el recurso externo, si corresponde.



Parsear el código descargado

Esto ocurre aún cuando no es necesario ejecutar el código desde la IU.

Se realiza antes del disparo del evento `onload`.

Es posible **minimizar el tiempo de carga** de una página “difiriendo” el parsing del código JavaScript.

Utilizar **`defer`** o **`async`** (HTML5) en el elemento **`script`**
Estos atributos permiten la carga asincrónica de los scripts.

Iniciar la carga de scripts luego del evento **`onload`**
Una vez que la página ha sido cargada, una función modifica el DOM agregando elementos **<script>**.

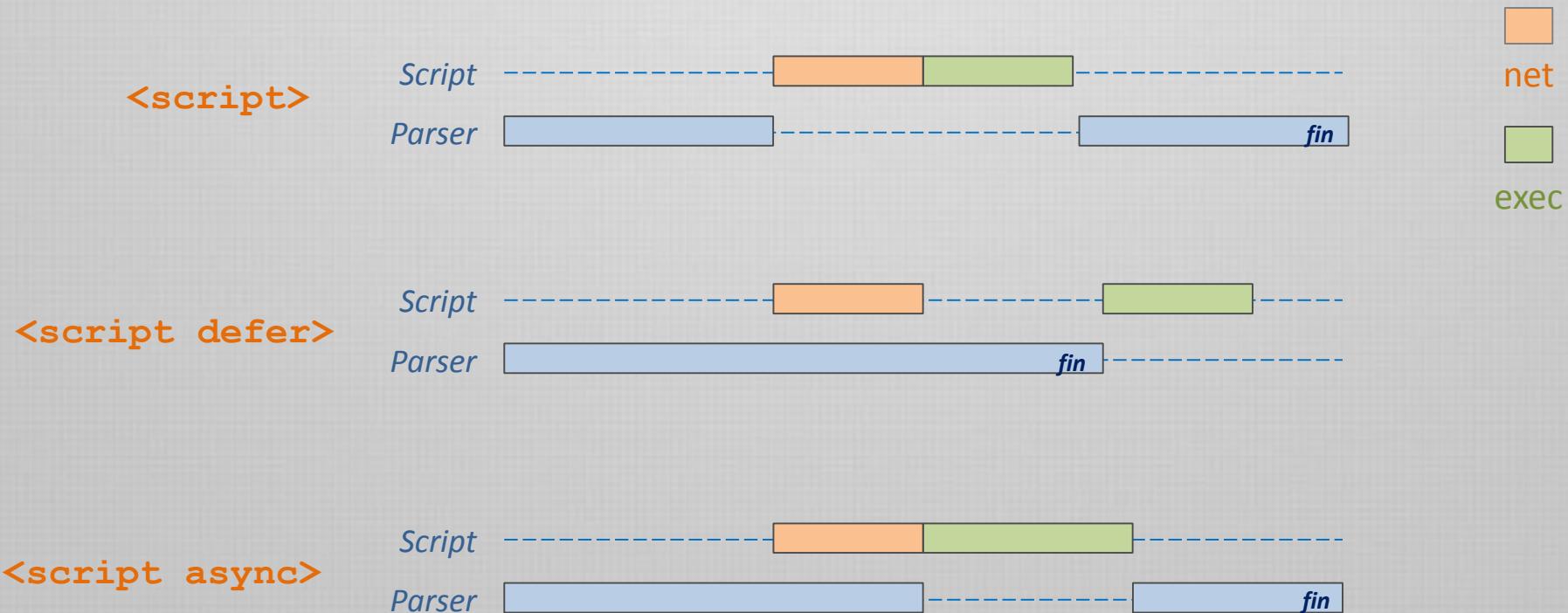
Cargar JavaScript en comentarios o strings
Luego se interpretan con la función **`eval()`**

Regla: Minimizar tiempos de carga

Utilizar **defer** o **async** (HTML5) en el elemento **script**

- El atributo **defer** es utilizado por IE hace varios años, de aceptación general paulatina.
- El atributo **async** es incluido en HTML5.

Los dos permiten cargar los scripts sin detener el parsing de HTML.



Regla: Minimizar tiempos de carga

Iniciar la carga de scripts luego del evento ***onload***

En cualquier lugar del documento puede agregarse esta porción de JavaScript que provoca la carga de otro script (lazy load)

```
<script>
  var node = document.createElement('script');
  node.type = 'text/javascript';
  node.src = 'example.js';
  // insertar en el DOM
</script>
```

Esto **no bloquea el resto del parsing**, aunque puede que la parte inferior del documento provoque errores de JavaScript antes de que finalice la carga.

Regla: Minimizar tiempos de carga

Cargar JavaScript en comentarios o strings

Puede realizarse la carga “a mano”, via AJAX para luego interpretarla con `eval()`

```
<script type="text/JavaScript">
    function loadFile(url) {
        function callback() {
            if (req.readyState == 4) { // 4 = Loaded
                if (req.status == 200) {
                    eval(req.responseText);
                } else {
                    // Error
                }
            }
        };
        var req = new XMLHttpRequest();
        req.onreadystatechange = callback;
        req.open("GET", url, true);
        req.send("");
    }
</script>
```

Regla: Minimizar tiempos de carga

Otra alternativa utilizada por Google:

```
<script id="lazy">
  /*
    Sentencias comentadas de JavaScript
  */
</script>

<script>
  function lazyLoad() {
    var lazyElement = document.getElementById('lazy');
    var lazyElementBody = lazyElement.innerHTML;
    var jsCode = stripOutCommentBlock(lazyElementBody);
    eval(jsCode);
  }
</script>

<div onclick=lazyLoad()> Lazy Load </div>
</html>
```

Regla: Minimizar pedidos HTTP

Evitar multiplicidad de recursos combinables

La existencia de varios recursos adicionales deriva en varios HTTP requests.

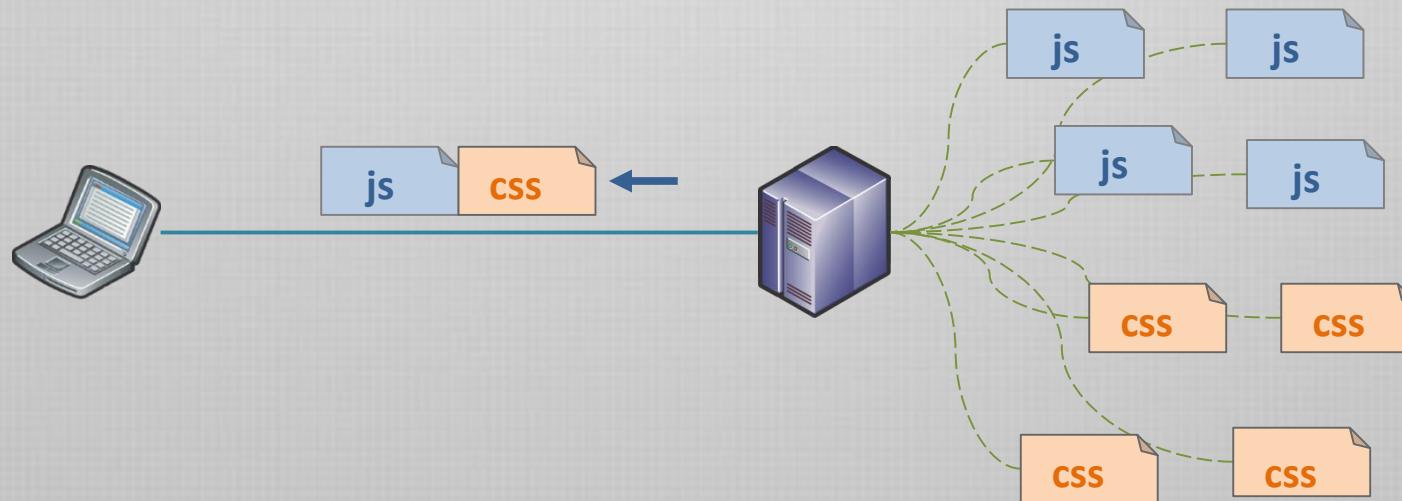
En particular los scripts y los css podrían combinarse en uno solo o en pocos.

Al fin y al cabo serán todos consolidados en el resultado final.

No siempre es factible la mezcla.

A veces los recursos provienen de diferentes fuentes.

Tal vez podrían ser combinados por un script del lado servidor.



Regla: Minimizar pedidos HTTP

En el objetivo de minimizar HTTP requests..

¿es preferible CSS y JavaScript *inline* o en *recursos externos*?

no generará requests adicionales.
La respuesta tendrá, sin embargo, mayor tamaño.

generará requests y respuestas adicionales.
Sin embargo, estos recursos podrán quedar en la caché del navegador.

¿cuando utilizar uno o el otro?

Si la frecuencia de un visitante es poca, preferiblemente *inline*.

Si la frecuencia de un visitante es mucha, preferiblemente *externo*.

Si la permanencia en el sitio es extensa, visitando varias páginas,
preferiblemente utilizar recursos *externos*.

Para home-pages como Google o Yahoo! preferiblemente *inline*.

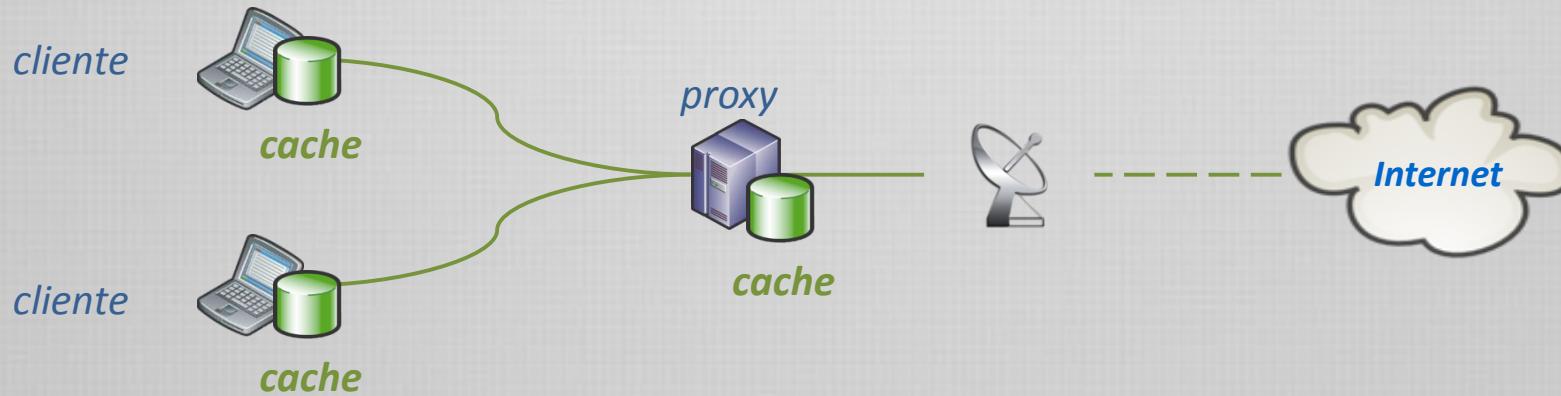
Regla: Optimizar el uso de la cache

Muchos de los recursos utilizados por una página no cambian frecuentemente.

CSS, imágenes de encabezados y pie de página, banners, etc

Los *navegadores* proveen una caché donde los recursos pueden ser **recuperados en lugar de solicitarlos** al servidor nuevamente.

Los *Internet Service Providers* (ISP) también proveen proxies con cache para los recursos transferidos.



El uso de cache (local o proxy) tiene varios beneficios:

- minimiza la cantidad de *HTTP requests*.
- minimiza la transferencia de información desde el servidor.
- libera ancho de banda para otras aplicaciones.

Regla: Optimizar el uso de la cache

Utilizar los encabezados *de caché*

Algunos navegadores utilizan heurísticas para decidir qué queda en la cache.
Pero también puede indicarse explícitamente en los *headers*.

Expires: Thu, 15 Jun 2019 20:00:00 GMT

La respuesta indica al cliente la fecha de expiración del recurso transportado

Cache-Control: max-age=315360000

La respuesta indica al cliente el período de validez del recurso transportado

El navegador NO solicitará el recurso (GET) mientras sea válido o no haya expirado.
Puede configurarse el servidor para que fije la *fecha* o el *max-age* automáticamente

Last-Modified: 15 Sep 2008 17:43:00 GMT

La respuesta indica al cliente la fecha de modificación del recurso

El cliente puede solicitar el recurso en el futuro con el tag:

If-Modified-Since: 15 Sep 2008 17:43:00 GMT

Regla: Optimizar el uso de la cache

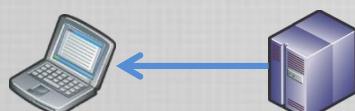
Pueden utilizarse también *Entity Tags (Etags)*

Son headers identificadores de recursos, independientes del tiempo.

Incluídos en HTTP 1.1



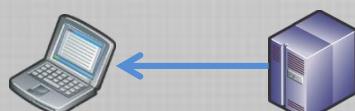
```
GET /imagenes/logo.gif HTTP/1.1  
Host: unsitio.com
```



```
HTTP/1.1 200 OK  
Last-Modified: Tue, 2 Nov 2008 05:06:00 GMT  
ETag: "10c24bc-4ab-457e1c1f"  
Content-Length: 1532
```



```
GET /imagenes/logo.gif HTTP/1.1  
Host: unsitio.com  
If-Modified-Since: Tue, 2 Nov 2008 05:06:00 GMT  
If-None-Match: "10c24bc-4ab-457e1c1f"
```



```
HTTP/1.1 304 Not Modified
```

Regla: Optimizar el uso de la cache

Los Etags son construidos con particularidades de cada servidor.



TM

*Para un mismo recurso, el Etag no será el mismo en IIS que en Apache.
No son los mismos incluso en servidores de la misma tecnología.*

Se utilizan datos como el *inode*, tamaño y timestamp del recurso en el servidor.

Si se provee *hosting* de multiples servidores, puede ser contraproducente.

Es posible configurar la conformación del etag, por ejemplo solo con el timestamp

Regla: Minimizar la cantidad de información del servidor

Las respuestas desde el servidor transportan mayor información que los requests.

El tamaño de la respuesta también causa un impacto en las demoras totales.

Aceptar información comprimida

Los navegadores pueden recibir información comprimida: *gzip* y *deflate*, indicando

Accept-Encoding: gzip, deflate

El servidor indica que la respuesta está comprimida utilizando el header

Content-Encoding: gzip

Es ventajoso comprimir HTML, CSS, JavaScript, XML

NO es ventajoso comprimir imágenes o documentos PDF.

Los servidores (como Apache) incluyen la funcionalidad de compresión y es posible indicar qué recursos deben ser comprimidos antes de enviarse al cliente.

Regla: Minimizar la cantidad de información del servidor

Compactar recursos de texto

Una técnica simple a observar es mantener en mínimo tamaño los recursos de texto, ya que usualmente contienen información “descartable”.

Esto se denomina a veces *minification (como consecuencia, uglyfication)*

En JavaScript es posible eliminar todos los espacios en blanco y los saltos de línea entre sentencias.

Existen herramientas para minificar JavaScript: JSMin, Closure Compiler

En CSS deben eliminarse los estilos no utilizados.

Google Page Speed permite detectar estilos innecesarios para una página

En HTML pueden eliminarse caracteres superfluos.

Aun falta un optimizador de HTML automatizado

Regla: Minimizar la cantidad de información del servidor

Usar apropiadamente las imágenes

Algunas aplicaciones gráficas agregan información adicional a las imágenes.

Algunos formatos son mejores que otros en cuanto a tamaño y definición



PNG es mejor que GIF en general.

Posee mayor grado de transparencia.

No se renderiza apropiadamente en navegadores viejos.



GIF es apropiado para imágenes pequeñas (iconos, bullets)



JPG debe utilizarse para imágenes grandes y fotografías.

No usar nunca BMP y TIFF, formatos sin compresión.

Ofrecer imágenes en el tamaño en el que se visualizarán.

Usar thumbnails y posponer la imagen en detalle si es necesario

Evitar fondos grandes. Aprovechar las propiedades de mosaico.

Usar y mantener pequeño el `favicon.ico`

Aunque no lo usemos, el browser igual lo pedirá!

Accesibilidad

Accesibilidad Web



Accesibilidad Web

“The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect.”

Tim Berners-Lee



naturalmente incluye a
Impedidos visualmente
Discapacitados motrices
Discapacidad auditiva
Dificultades del habla



...

Para muchas de estas personas, Internet tiene una importancia esencial por sus circunstancias de vida.



“For me being online is everything. It’s my hi-fi, my source of income, my supermarket, my telephone. It’s my way in”

Lynn Holdsworth, Digital Accessibility Specialist

Accesibilidad Web

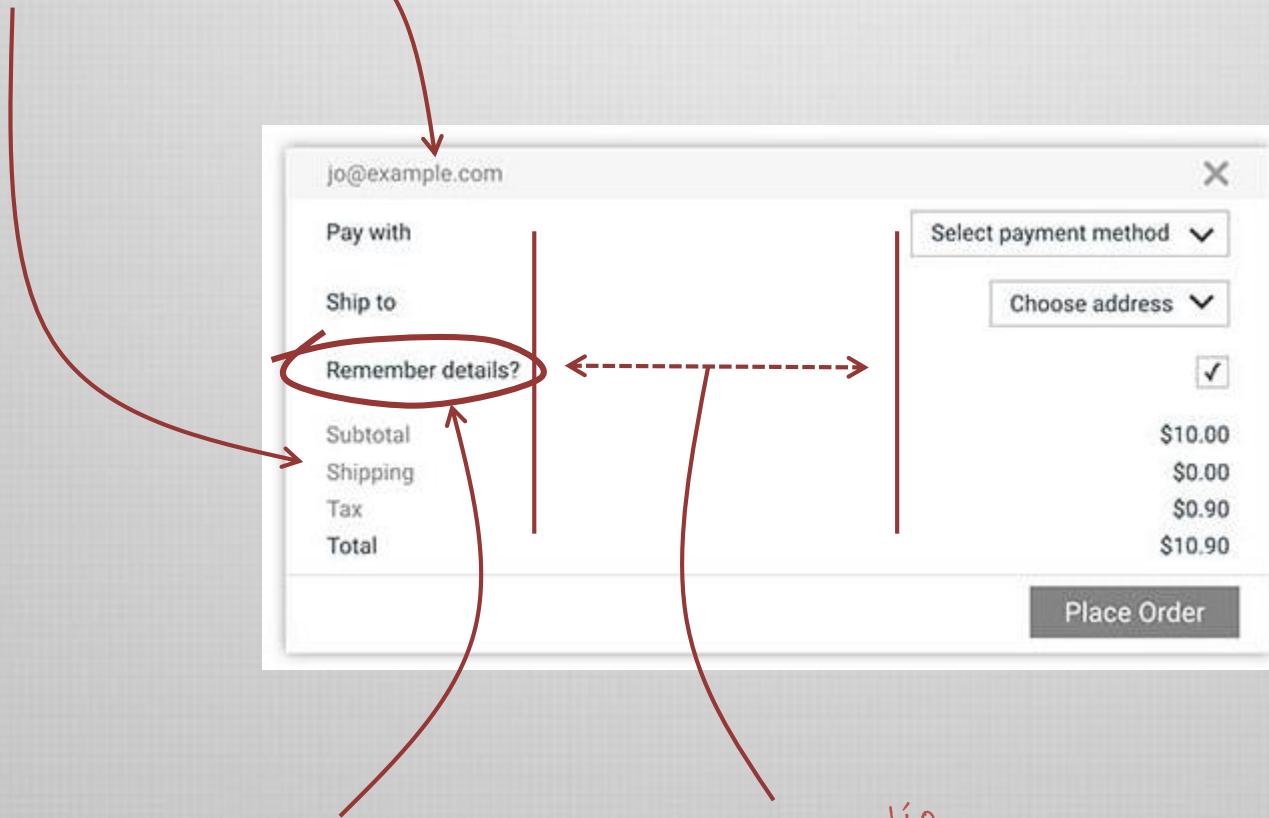
This video is not available in your country.

Sorry about that.



Accesibilidad Web

bajo contraste



espacio amplio
entre etiquetas
y componentes

Accesibilidad Web

jo@example.com ×

Subtotal	\$10.00
Shipping	\$0.00
Tax	\$0.90
Total	\$10.90

Pay with:

Select payment method ▾

Ship to:

Choose address ▾

Remember details?

Place Order



Accesibilidad web

Particularidades de algunas discapacidades

No videntes

- No utilizan monitor ni mouse
- Text-to-speech, speech-to-text



Impedidos visuales

- Rango limitado de visión
- Necesidad de mayor contraste de colores
- Zoom
- Confusión de colores



Discapacidad auditiva

- No reconocen señales auditivas (beeps, notificaciones sonoras, etc)
- Necesidad de videos con Closed-caption



Discapacidad motriz

- Mayor variación en las dificultades
- Imposibilidad o dificultad para usar mouse y teclado.

Discapacidad mental

- Mayor variación en las dificultades
- Requieren estructuras de navegación simples
- A veces, conviene gráficos en lugar de texto.

Accesibilidad Web



Accesibilidad web

¿Por qué crear sitios web *accesibles*?



Nos aseguramos que la información está disponible para personas con discapacidades

No formamos parte de la frontera que limita el acceso a personas con dificultades



Aumentamos la audiencia de nuestra información web

Llegamos a mas cantidad de personas



Es obligatorio hacerlo

Legalmente, algunas leyes se aplican

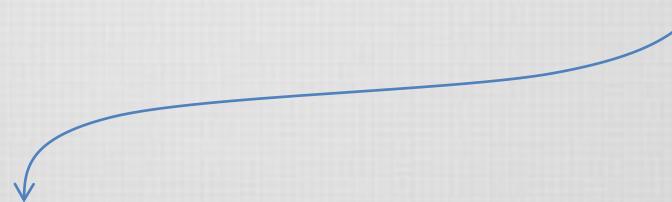
Éticamente también :)

Accesibilidad web –W3C

W3C interviene en este aspecto en forma activa.



WAI
Web Accessibility Initiative



establece guías aceptadas como **estándares internacionales** de accesibilidad web

soporta material para comprender e implementar accesibilidad web

desarrolla recursos a través de voluntarios

Grupos de Trabajo

Authoring Tool Accessibility Guidelines Working Group (AUWG)

Education and Outreach Working Group (EOWG)

Evaluation and Repair Tools Working Group (ERT WG)

Protocols & Formats Working Group (PFWG)

Research and Development Interest Group (RDIG)

User Agent Accessibility Guidelines Working Group (UAWG)

WAI Interest Group (WAI IG)

Web Content Accessibility Guidelines Working Group (WCAG WG)

WAI Coordination Group

Accesibilidad web –W3C

Documentos emitidos por la W3C sobre accesibilidad web

» Web Content Accessibility Guidelines (WCAG 2.0)

Procura definir páginas accesibles a todos los usuarios

Provee guías para la accesibilidad en sitios web.

POUR principles:

Percivable

Proveer alternativas de texto para contenido no-textual (large print, braille, voz, etc)

Proveer alternativas para time-based media (captions, audio tracks, etc)

Crear contenido que puede ser presentado de diferentes formas sin pérdida de información.

Facilitar a los usuarios ver y escuchar contenido incluyendo separación de foreground-background

Operable

Hacer que toda la funcionalidad esté disponible por teclado.

Ofrecer a los usuarios tiempo suficiente para leer y usar el contenido de la página

No diseñar contenido en formas que se conocen como causantes de ataques (e.g. epiléticos)

Proveer ayudas de navegación, búsqueda de contenido y locación dentro de la información

Understandable

Hacer que el contenido sea legible y comprensible.

Hacer que las páginas web sean predecibles en su forma de operar

Ayudar a los usuarios a evitar y corregir errores.

Robust

Maximizar la compatibilidad actual y futura con cualquier user-agent

Accesibilidad web –W3C

Documentos emitidos por la W3C sobre accesibilidad web

- » User Agent Accessibility Guidelines (UAAG)
Guías para desarrolladores de user-agents.
 - Acceso a todo contenido, incluidos los producidos por eventos del mouse o teclado
 - Control del usuario de cómo el contenido es renderizado.
 - Control del usuario sobre la interfaz de usuario.
 - Interfaces de programación estandarizadas
- » Authoring Tools Accessibility Guidelines (ATAG)
Guías para el software y servicios para la producción de contenido web.
 - Cómo crear herramientas de autoría accesibles
 - Cómo crear herramientas que producen contenido accesible.
- » Accessible Rich Internet Applications (WAI-ARIA)
Provee una ontología de roles, estados y propiedades para definir elementos de interfaces de usuario (widgets, estructuras, comportamientos)
 - Por ejemplo, crear regiones en una página, navegables desde el teclado en lugar de presionar “Tab” varias veces.
 - Por ejemplo, definir sectores que son actualizados via Ajax.

EEUU – Rehabilitation Act



Section 504 of the Rehabilitation Act of 1973

“No otherwise qualified individual with a disability in the United States . . . shall, solely by reason of her or his disability, be excluded from the participation in, be denied the benefits of, or be subjected to discrimination under any program or activity receiving Federal financial assistance.”

Es una ley de derechos civiles.

Prohíbe la discriminación basada en el estatus de discapacidad.



Section 508 of the Rehabilitation Act of 1973

Basado en la enmienda anterior, prohíbe al gobierno Federal proveer bienes y servicios electrónicos y de tecnologías de la información que no sean completamente accesibles a personas con discapacidades.

Es el primer estándar federal de accesibilidad web de los Estados Unidos

Todos los estados reciben financiamiento federal y están obligados por esta ley

Los proveedores del Estado deben cumplir esta ley

EEUU – Rehabilitation Act

Ejemplos de análisis de aplicación:

Se debe proveer un equivalente de texto para cada elemento no-texto.

Los documentos deben ser organizados de tal forma que sean legibles sin requerir ninguna hoja de estilo asociada.

Section 508 of the Rehabilitation Act of 1973 - Standard

Las páginas deben ser diseñadas de forma tal que la pantalla no parpadee con frecuencias mayores a 2hz y menores a 55hz

Cuando se espera una respuesta con límites de tiempo, el usuario debe ser alertado y ofrecer la posibilidad de requerir mas tiempo que el dado por defecto.

Cuando la página incluya contenido generado por un *applet*, plug-in u otra aplicación, debe proveerse un link para acceder al plug-in o applet.

Mitos sobre la accesibilidad web

1) Los sitios accesibles son aburridos.

Una idea preconcebida desde el diseño gráfico del sitio.

Los sitios accesibles NO son simples sitios de texto.

Las guías **no prohíben elementos**, sino que dicen cómo usarlos debidamente.

Ejemplo:

Uso del atributo “alt” en las imágenes.

Con eso es suficiente para hacer ese componente accesible.

2) Crear páginas accesibles es más costoso

Se usan las **mismas tecnologías** de siempre.

Si se planifica desde el principio, es tan costoso como cualquier otro sitio.

3) La tecnología de asistencia resuelve los problemas de accesibilidad

Hay grandes avances recientes en la **Tecnología de Asistencia**

(*sintetizadores, reconocimiento de voz, terminales braille, etc*)

Sin embargo, la información debe ser provista apropiadamente.

La accesibilidad no se reduce a poseer el “juguete tecnológico” adecuado.

Mitos sobre la accesibilidad web

4) Los no-videntes y los discapacitados no usan Internet.

No sólo lo usan: lo necesitan para mejorar su calidad de vida.

5) La accesibilidad web sólo beneficia a un conjunto reducido de usuarios

Usualmente un pretexto para la incapacidad de diseñar apropiadamente.

¿Qué significa “reducido” o “pocos” usuarios?

La accesibilidad web no sólo ayuda a personas con capacidades disminuidas.

6) La adhesión mínima a WCAG 2.0 garantiza la accesibilidad

No todas las necesidades de los usuarios están cubiertas por alguna guía.

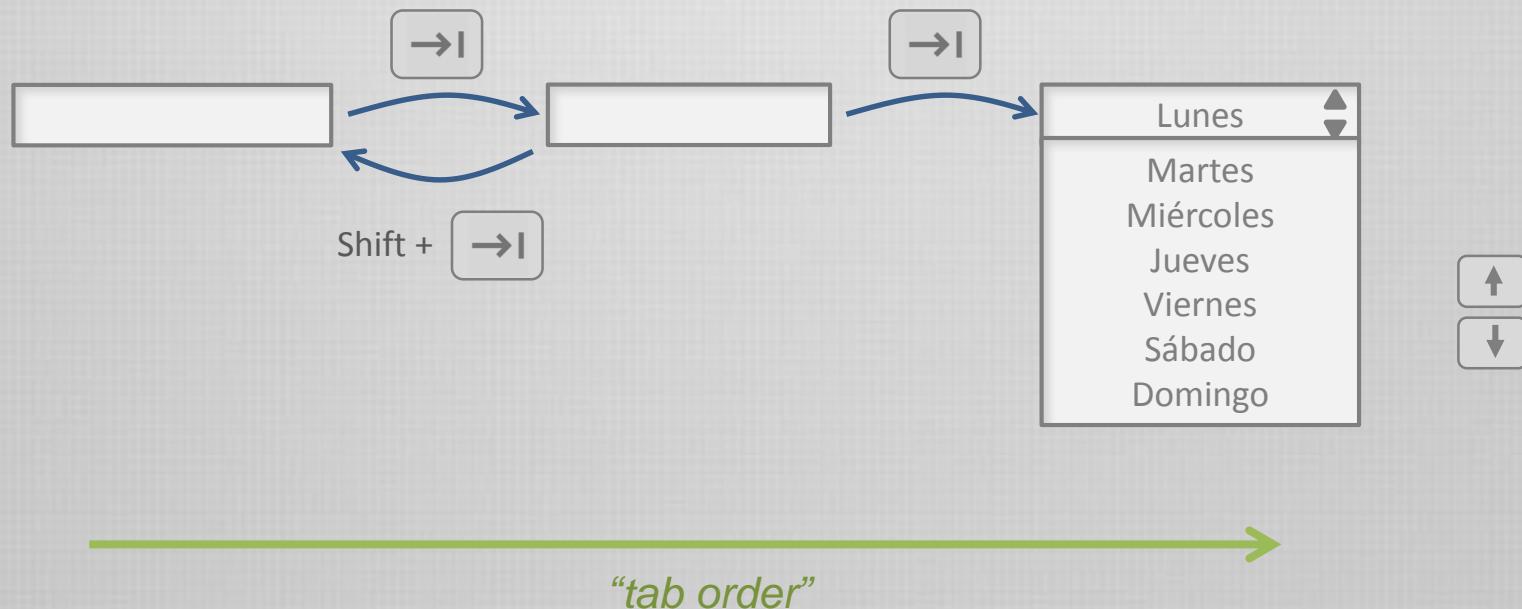
El objetivo de la WCAG 2.0 es eliminar barreras comunes.

El sitio requiere monitoreo y actualización tecnológica como es habitual.

Tips de accesibilidad

focus

*Determina en el documento el objeto que recibirá eventos del teclado
No todos los elementos son enfocables*

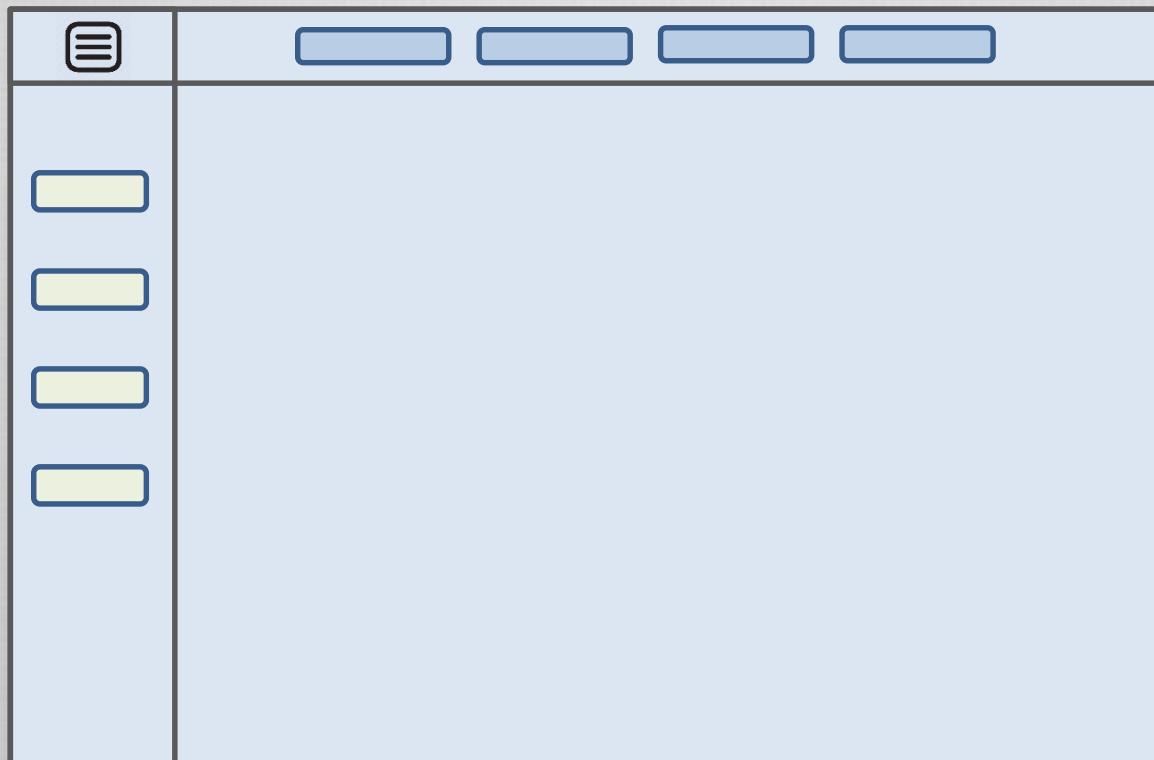


*Debe seguir un orden lógico
Es necesario observar el orden DOM*

Tips de accesibilidad

focus

*Determina en el documento el objeto que recibirá eventos del teclado
No todos los elementos son enfocables*



- ✓ Agregar el atributo `alt` a las imágenes

```

```

- ✓ Usar los headings adecuadamente

```
<h1>División de primer nivel</h1>  
<h2>División de segundo nivel</h2>
```

- ✓ Crear PDF accesibles

Los PDF varían en el formato de su contenido. Por ejemplo, una página escaneada. Conviene convertir a texto (OCR) y agregar metadatos (alt text, tags,etc)

- ✓ Saber cuándo usar PDF...

Para muchos documentos, HTML o Word es mejor.

- ✓ Agregar etiquetas a los campos de un formulario

Asocia explícitamente el campo de entrada con lo que representa.

- ✓ Identificar el lenguaje

```
<html lang="fr">
```

- ✓ Utilizar un verificador de contraste de colores

<http://webaim.org/resources/contrastchecker/>

- ✓ Identificar nombres de columnas en las tablas

```
<table>
  <thead>
    <tr>
      <th scope="col">id</th>
      <th scope="col">Nombre</th>
      <th scope="col">Apellido</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      ...
    </tr>
  </tbody>
</table>
```

✓ Evitar usar fonts pequeñas.

¡Oíd, mortales!, el grito sagrado: ¡libertad!, ¡libertad!, ¡libertad! Oíd el ruido de rotas cadenas ved en trono a la noble igualdad. Ya su trono dignísimo abrieron las Provincias Unidas del Sud! Y los libres del mundo responden: ¡Al gran Pueblo Argentino, salud!

✓ Proveer un indicador visible del foco de los componentes.

```
a {  
    color: black;  
    background-color: white;  
}  
a:hover a:focus{  
    color: white;  
    background-color: black;  
}
```

✓ Cuidar la redacción de los textos.

Sentencias complejas y palabras largas hacen al texto difícil de leer.
“Readability”

<https://www.webpagefx.com/tools/read-able/>



Caption video

Es simple y hace accesible los videos a personas con discapacidad auditiva. Además expande la posibilidad de búsquedas de video. Puede ser “on-the-fly”.



Crear menúes accesibles

TAB para entrar y salir del menú.

“←” y “→” para navegar en el menú

Abrir submenu con **ENTER, SPACE** o ”↓” y ”↑”

Salir del submenu con **ESCAPE**



Elegir widgets JavaScript que sustentan accesibilidad

JQuery, Dojo Toolkit, Yahoo YIU3

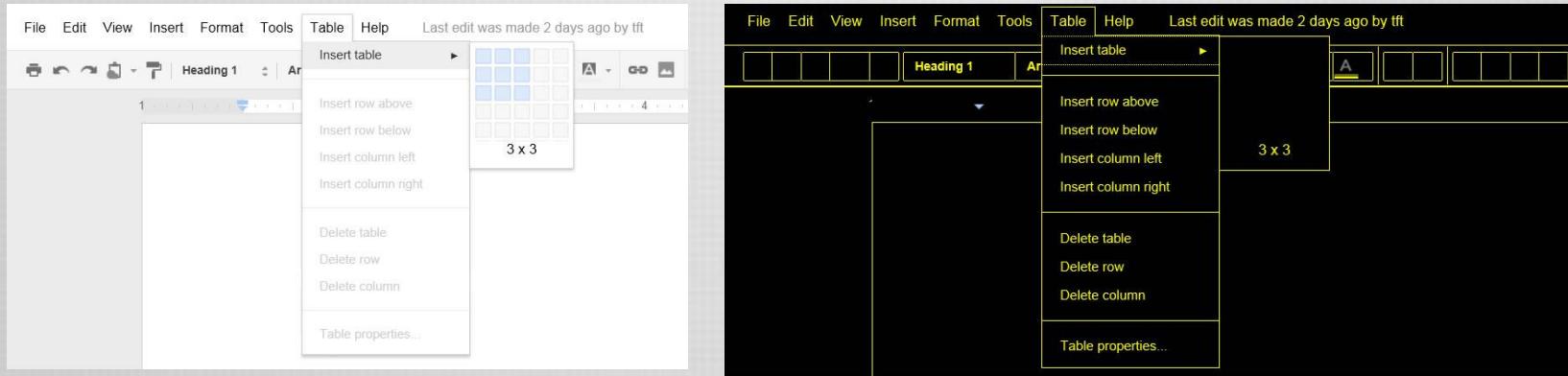


Elegir CMS que sustentan accesibilidad

Moodle, Drupal, Joomla!, Wordpress



Testear con alto contraste



(by Terrill Thompson)

Arquitecturas Web

Arquitecturas Web



¿cómo organizar todas estas tecnologías?

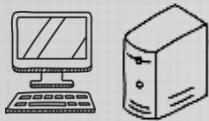


buscamos
escalabilidad
seguridad
confiabilidad
adaptabilidad

Arquitecturas Web



Arquitecturas web



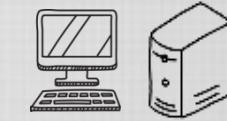
Static HTML

- 👍 bajo costo
- 👍 simple
- 👍 cacheable
- 👎 difícil de actualizar
- 👎 UI pobre



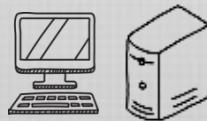
CGI

- 👍 Contenido dinámico
- 👍 Fácil de actualizar
- 👎 Alto costo computacional
- 👎 UI pobre



HTML + JavaScript

- 👍 Contenido dinámico
- 👍 UI Mejorada
- 👍 AJAX
- 👎 Lógica duplicada
- 👎 Riesgo de desbalance
- 👎 Poco cacheable



Client Side Apps

- 👍 Contenido dinámico
- 👍 Reduce carga del servidor
- 👍 Cacheable
- 👎 Contenido no indexable
- 👎 Requiere browser moderno

Frameworks

Básicamente

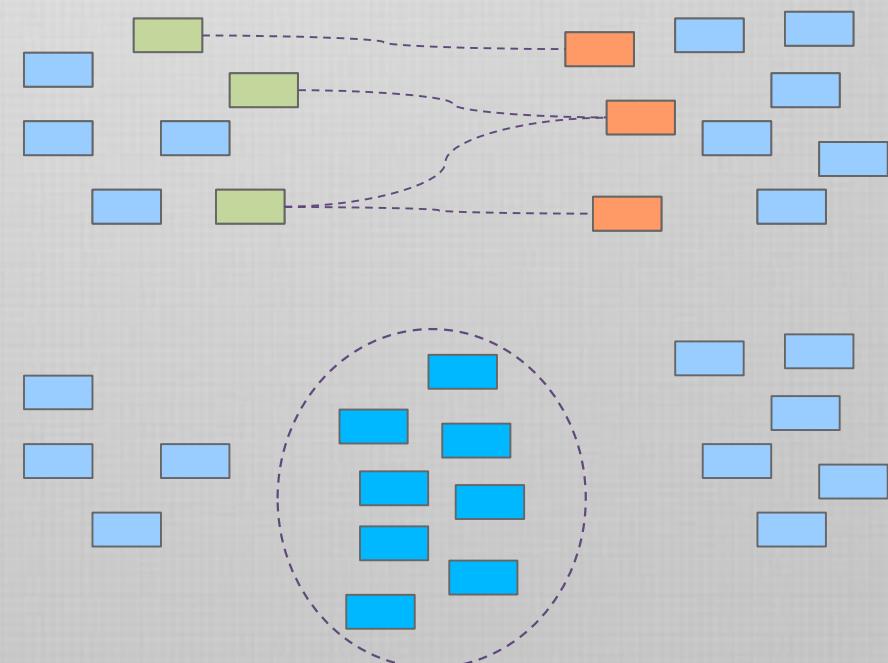
diseño e implementación parcial para una aplicación en un dominio específico

En cierto sentido es una aplicación *incompleta...*

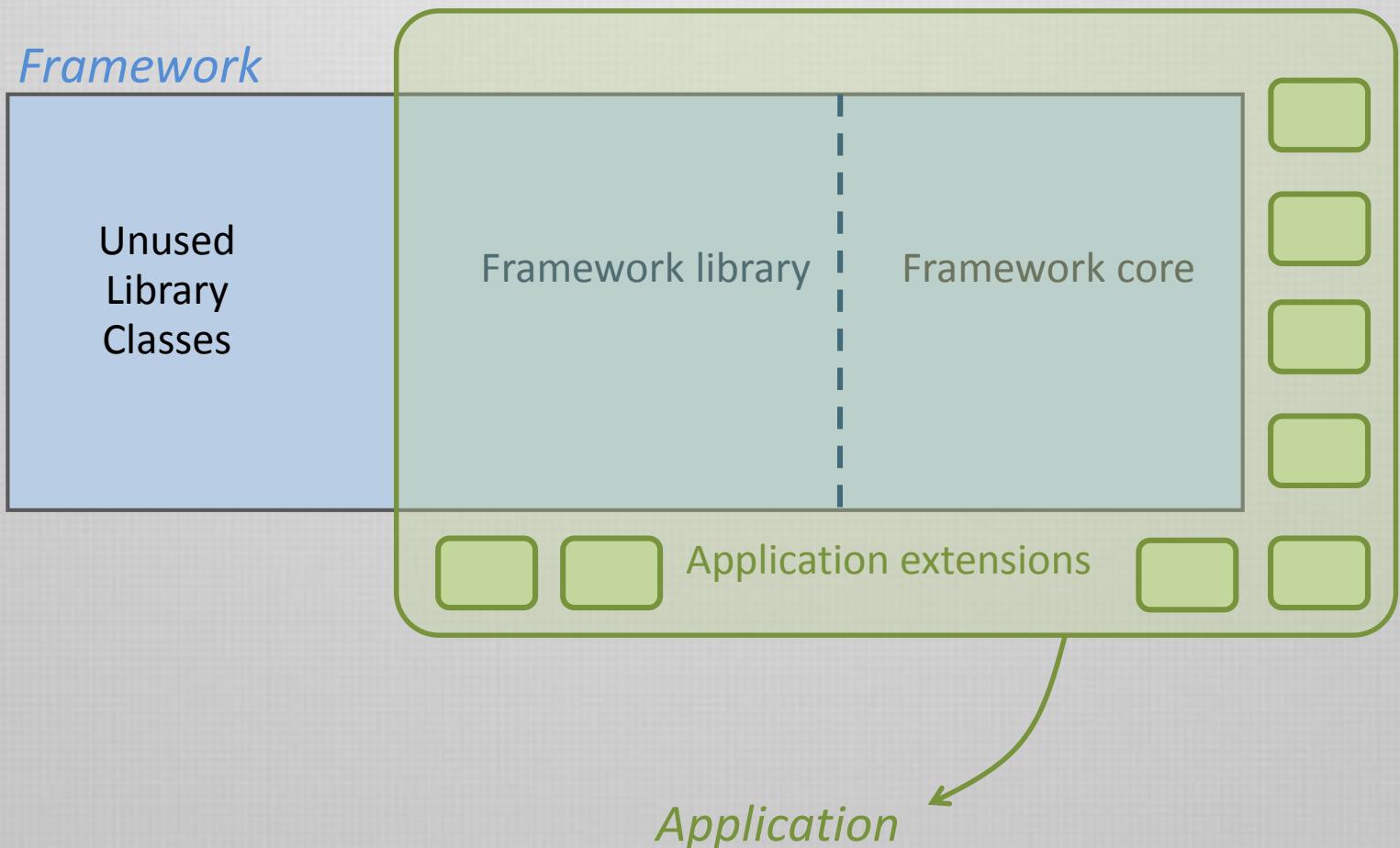
.. y lo faltante puede completarse de diversas formas, de modo tal de obtener diferentes aplicaciones específicas

Los frameworks son construídos cuando es necesario desarrollar varias aplicaciones *similares* (*en función y estructura*)

El framework implementa aquellos aspectos comunes a estas aplicaciones.



Partes de un framework



Beneficios del uso de frameworks



Reduce el tiempo de desarrollo.

No se inicia el proyecto desde cero.



Reduce los costos de desarrollo.

Menos tiempo implica menos costos.

El conocimiento del framework implica menos formación de RRHH.



Induce una organización arquitectónica.

Cada framework sigue una arquitectura particular, bastante rígida.

Obliga a una disciplina de trabajo ordenada.



Impregna calidad al proyecto.



Los frameworks son testeados y mantenidos constantemente.

Muchas instanciaciones actualmente en ejecución.



Aumenta el espectro de desarrolladores informados.

Un framework popular es conocido por muchos y por lo tanto ofrece más candidatos desarrolladores.



Facilita la integración y renovación de desarrolladores.

Un desarrollador familiarizado con un framework puede incorporarse rápidamente a proyectos de instancia.

Layering

La metáfora de las *capas* es la técnica común para particionar sistemas complejos.

Ejemplo mínimo → *cliente-servidor en bases de datos*



Cada capa descansa sobre la capa inferior.

Las capas inferiores son independientes de las superiores.

Ventajas:

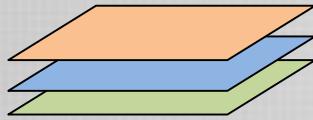
- Comprensión modular por capas*
- Abstracción de capas inferiores*
- Facilidad de cambios de capas*

Desventajas:

- Las capas no siempre encapsulan todo (cascading changes).*
- Performance en riesgo*

Las tres capas principales

Las tres **capas** principales usualmente distinguidas son



Presentación

Centrada en la interacción con el usuario.
Muestra información e interpreta comandos.

Dominio

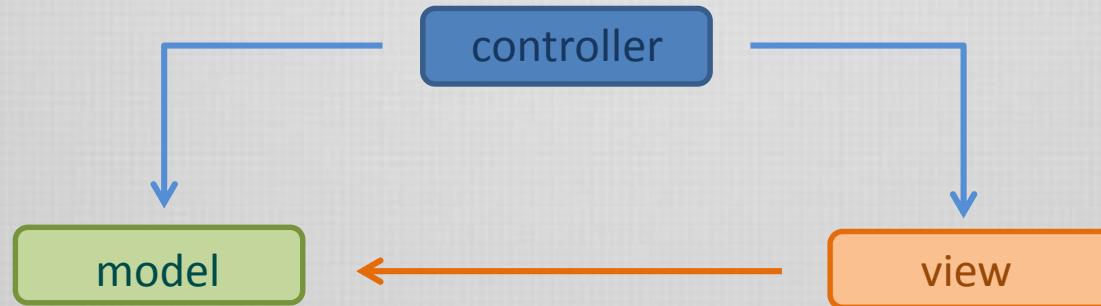
"Domain Logic" o "Business Logic"
Computaciones sobre los datos dependientes del dominio particular

Datos

Centrada en la comunicación con otros sistemas externos.
Típicamente un DBMS.

Patrón general de diseño: MVC

MVC es el patrón arquitectónico predominante en las aplicaciones web



El Controlador administra el Modelo y la Vista.

La Vista es responsable de observar el Modelo para exteriorizar los datos.

Patrón general de diseño: MVC

MVC es el patrón arquitectónico predominante en las aplicaciones web



Patrón general de diseño: MVC



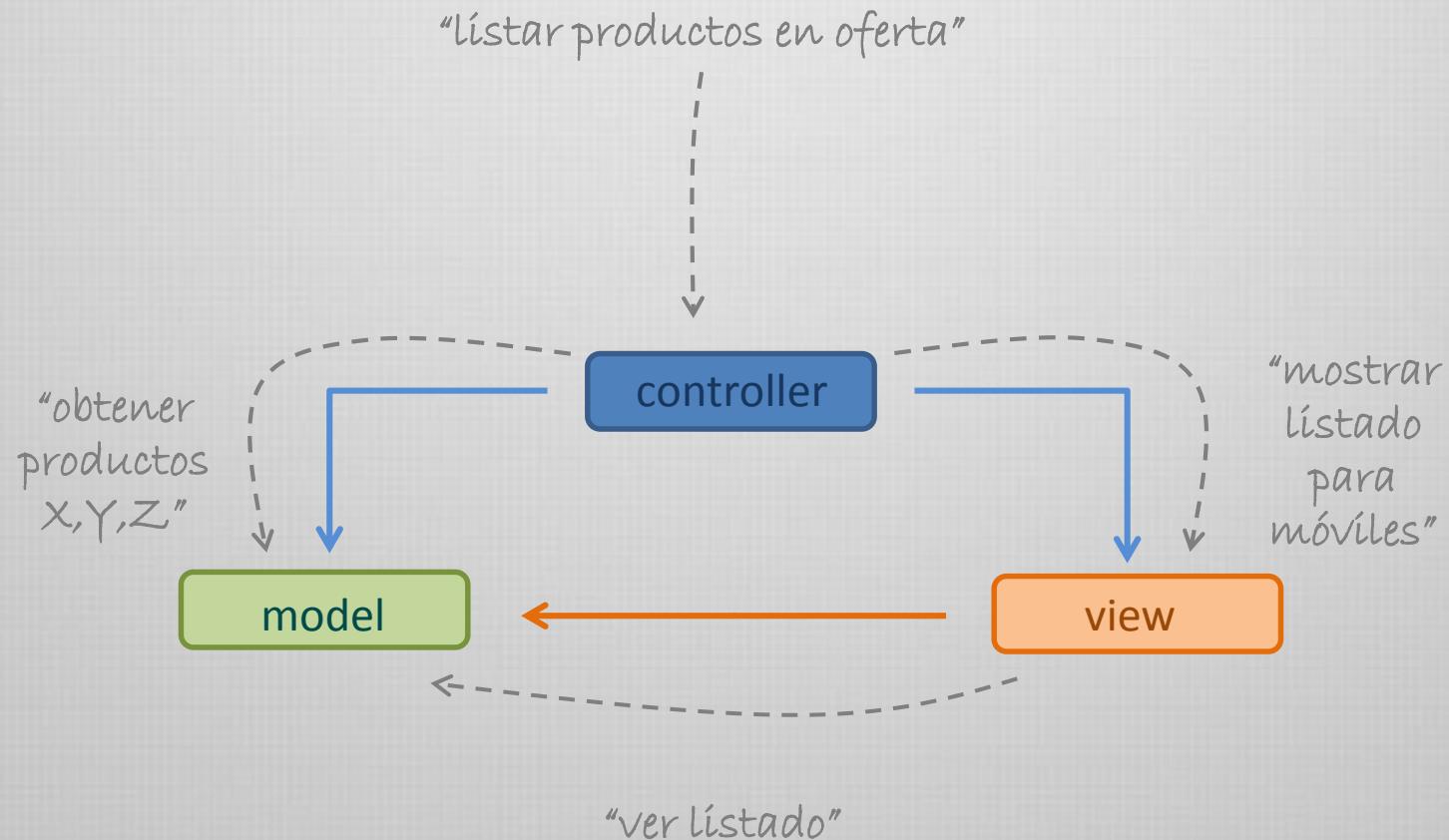
controller

model

view



Patrón general de diseño: MVC

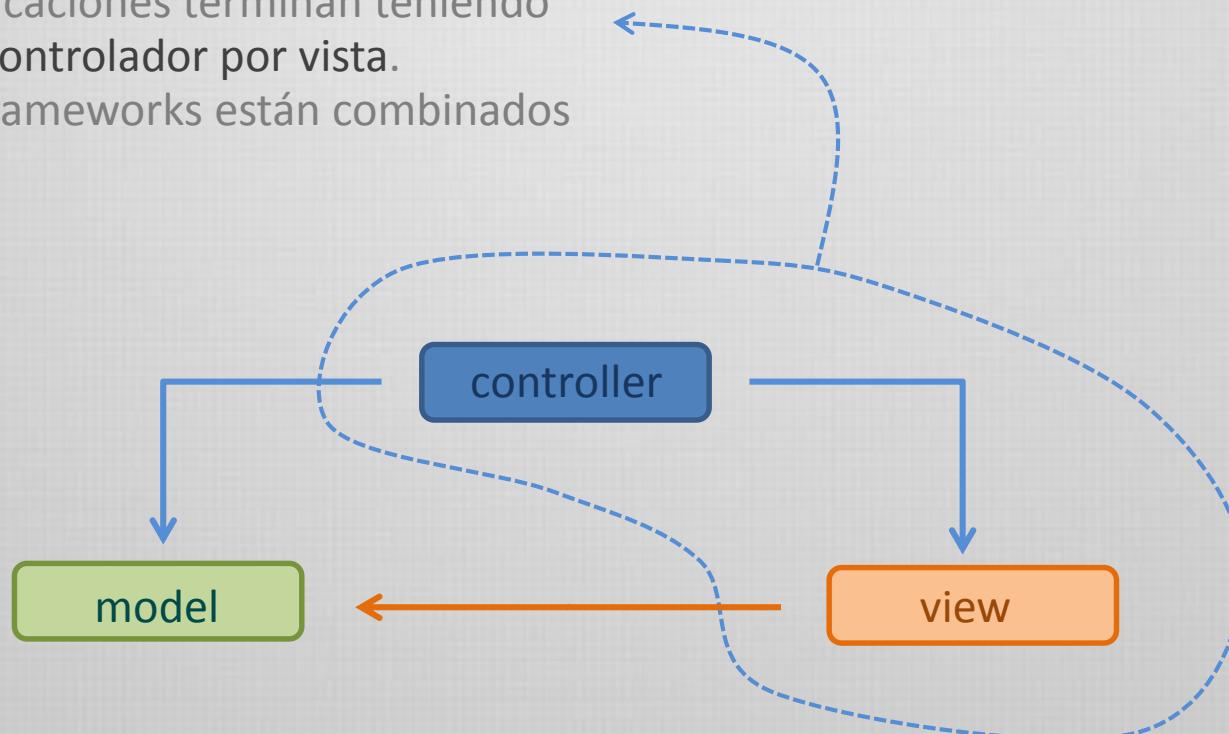


MVC: separaciones principales

Usualmente sobrevalorada.

Muchas aplicaciones terminan teniendo
un controlador por vista.

En algunos frameworks están combinados



MVC: separaciones principales

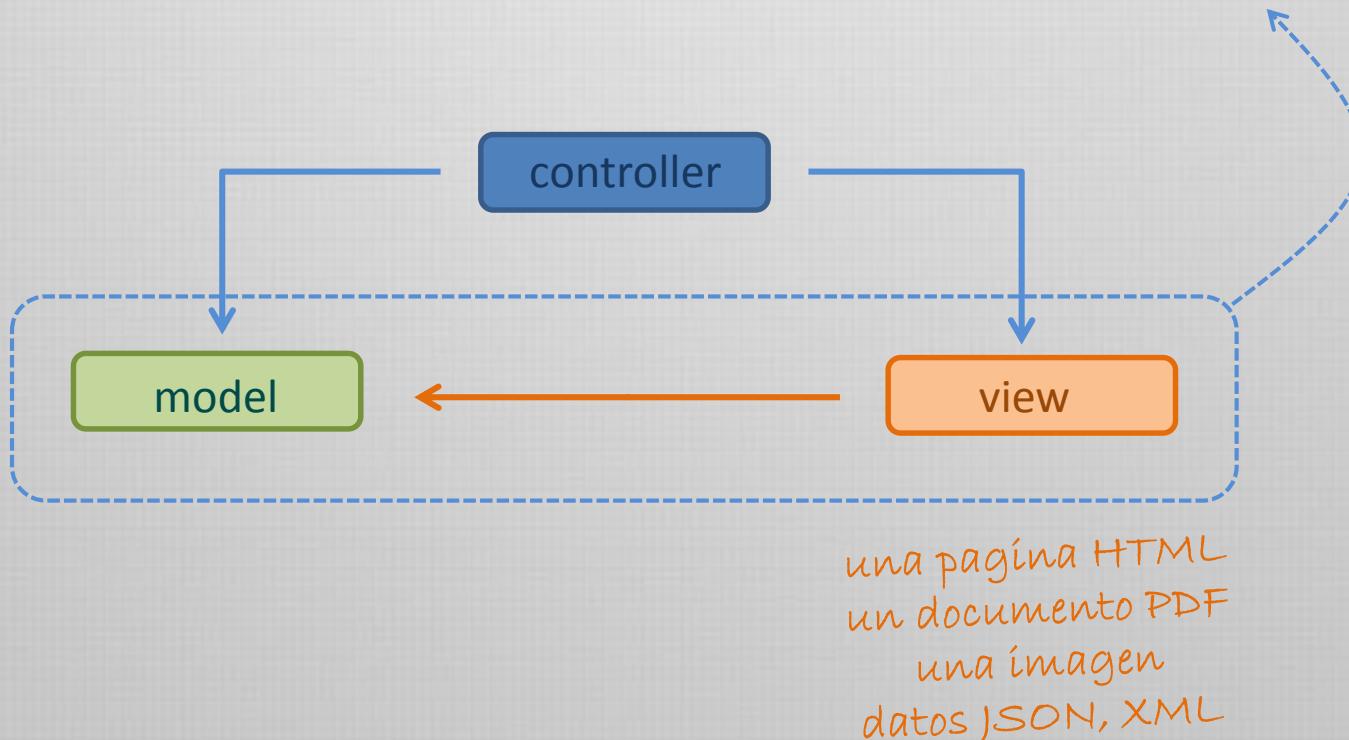
Separación principal

Las vistas y las presentaciones son problemas puntuales.

Los programadores del modelo no necesitan conocer nada de las vistas.

La misma información puede ser visualizada de maneras diferentes.

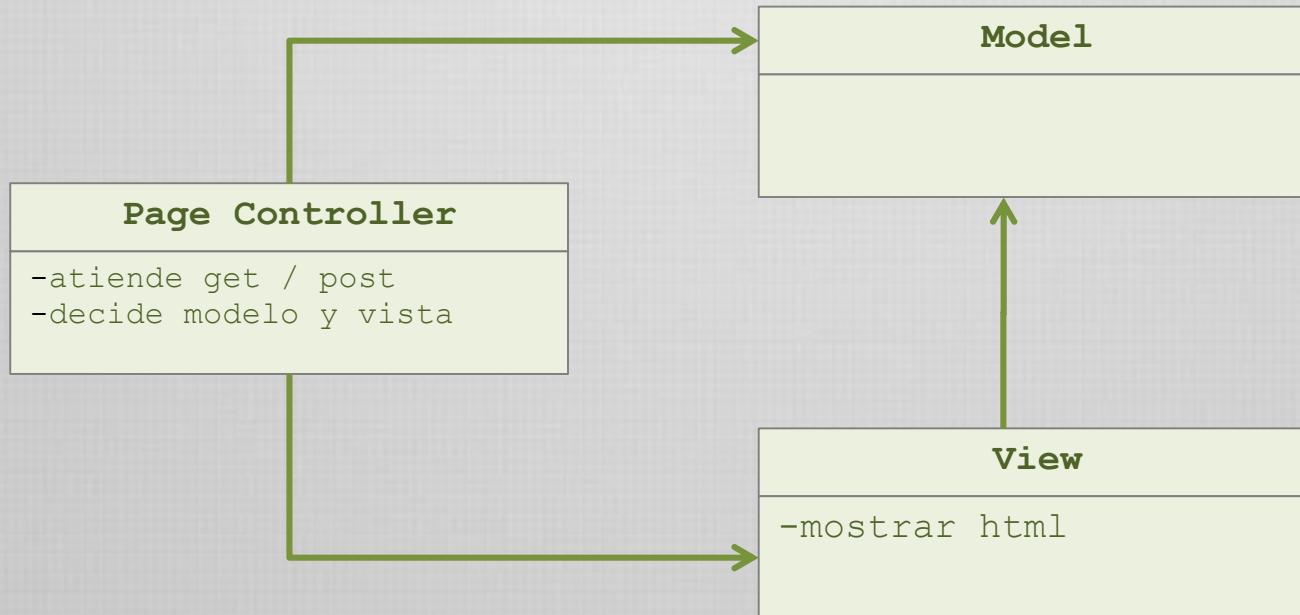
El testing puede despojarse de características visuales.



Page controller

Page-controller es un patrón de diseño congruente con el patrón arquitectónico MVC.

Aquí un objeto controla el *request* de una página específica del sitio (action)

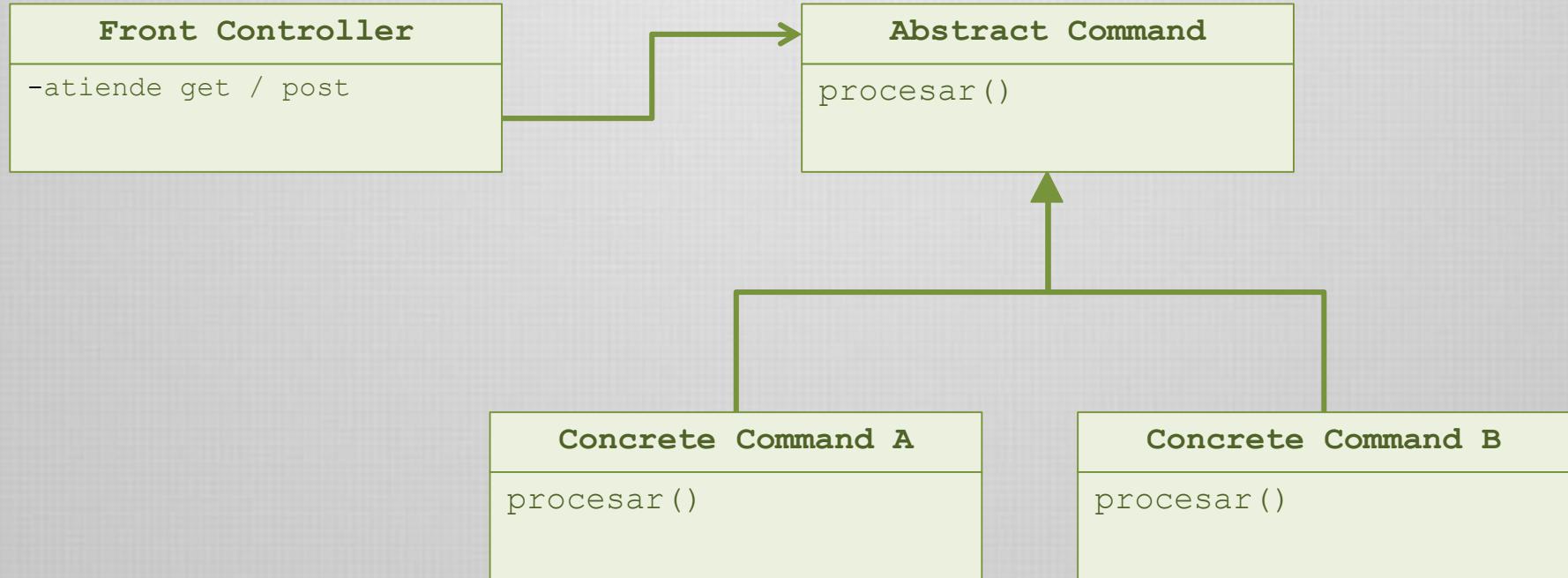


Hay un controlador por cada página lógica del sitio.
En algunos casos es la página misma.

Principalmente vinculado a una acción del sitio
Determina un conjunto de páginas parametrizables.

Front controller

Front-controller maneja **todos** los requests de un sitio web.



El objeto Front-controller recibe el request del cliente y
Recupera los datos correspondientes.
Prepara los datos para los comandos.
Decide qué comando ejecutar a continuación.

Template View

Template View es un patrón que renderiza información a HTML, incrustando marcas en una plantilla HTML.

```
{ marca }  
<? marca ?>  
<pre:marca/>  
<prefijo:marca> </prefijo:marca>
```

Las marcas son reemplazadas por datos computados.

Es en realidad el patrón habitual de los scripts PHP, JSP, ASP, aunque estos permiten la inclusión de lógica compleja.

Es el patrón usualmente elegido para la Vista del patrón MVC

Frameworks para la Web

¿por qué usar frameworks en la web?



menos
tiempo



menos
gastos



organización
del código



distribución
de tareas

¿qué suele ofrecer un framework para la web?

Uso de patrones de diseño efectivos

Arquitecturas definidas

Facilidades para la operatoria web
(formularios, validación, autenticación, soporte para UI, etc)

API

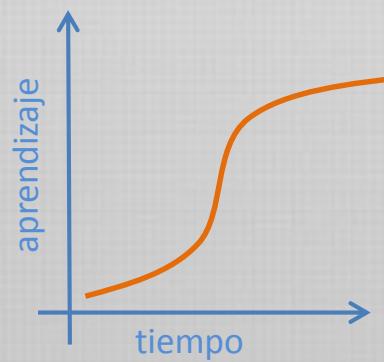
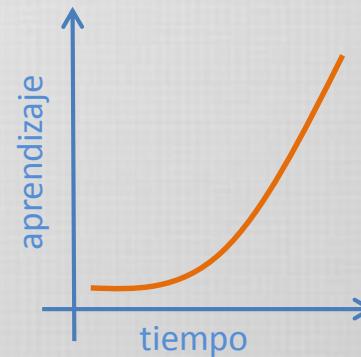
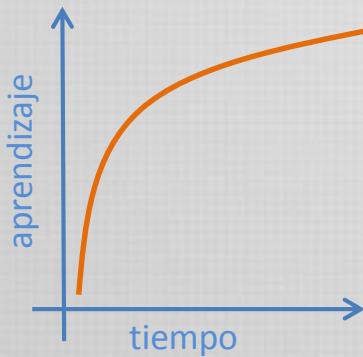
Administración de la capa de datos

Escalabilidad y performance

Comunidad de desarrolladores

Frameworks para la web

Curva de Aprendizaje



PHP - Frameworks para Web

Existe una gran variedad de frameworks, para diferentes lenguajes.
En el caso puntual de PHP, la oferta es extensa y variada.

PHP Framework	PHP4	PHP5	MVC	Multiple DB's	ORM	DB Objects	Templates	Caching	Validation	Ajax	Auth Module	Modules	EDP
Akelos 	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-
ash.MVC 	-	✓	✓	-	-	✓	✓	-	✓	-	✓	✓	-
CakePHP 	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	✓	✓	-
CodeIgniter 	✓	✓	✓	✓	-	✓	✓	✓	✓	-	-	-	-
DIY 	-	✓	✓	-	✓	✓	✓	✓	-	✓	-	-	-
eZ Components 	-	✓	-	✓	-	✓	✓	✓	✓	-	-	-	-
Fusebox 	✓	✓	✓	✓	-	-	-	✓	-	✓	-	✓	-
PHP on TRAX 	-	✓	✓	✓	✓	✓	✓	-	✓	✓	-	✓	-
PHPDevShell 	-	✓	-	-	-	-	✓	-	-	✓	✓	✓	-
PhpOpenbiz 	-	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	-	-
Prado 	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
QPHP 	✓	✓	✓	✓	✓	-	✓	✓	-	✓	✓	✓	✓
Seagull 	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-
Symfony 	-	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	✓	-
WACT 	✓	✓	✓	✓	✓	-	✓	✓	-	✓	-	✓	-
WASP 	-	✓	✓	-	-	✓	✓	-	✓	✓	✓	✓	-
Yii 	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Zend 	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-
ZooP 	✓	✓	✓	✓	-	✓	✓	✓	✓	✓	✓	-	-

ORM = object recorder mapper

DB Objects = objetos modelo de elementos de bases de datos

Auth Module = incluye módulo de autenticación de usuarios.

Modules = otros modulos de utilidad general: RSS, PDF

EDP = Programación orientada/guiada por eventos

PHP - Frameworks para Web

Lightweight vs Heavyweight

ofrecen funcionalidad básica y estructura simple.

mayor funcionalidad y dependencia del framework

Licencias

GPL, Open Source, BSD, MIT, etc

Estandarización

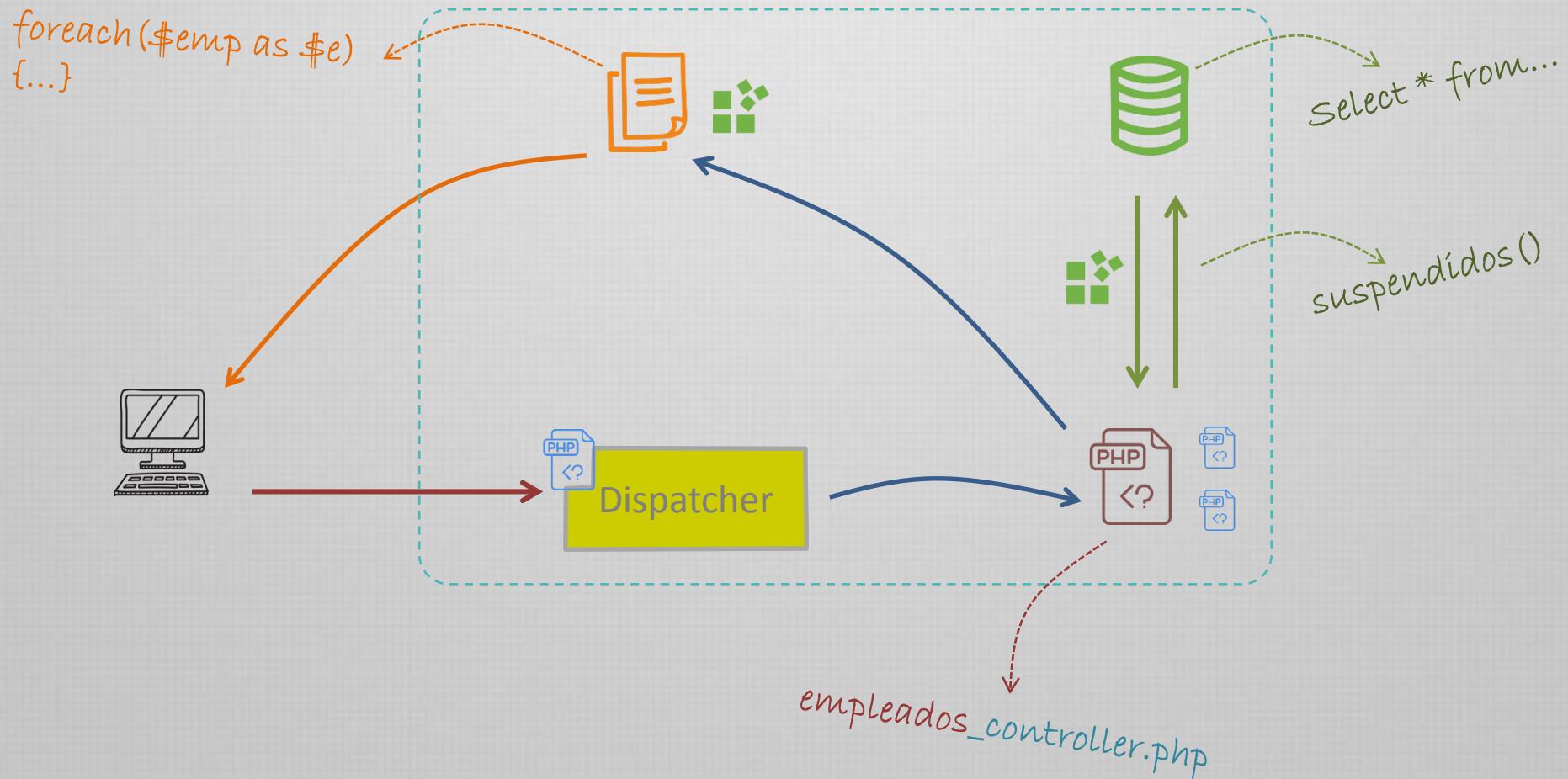
www.php-fig.org

Framework Interoperability Group

*Es un consorcio de desarrolladores PHP que determinan estándares generales
(coding style, logging, autoloading, etc)*

Web Frameworks: CakePHP

<http://nuestro.host.com/empleados/suspendidos>



Web Frameworks: CodeIgniter



Framework PHP, rápido y liviano

Más liberal que CakePHP en la aplicación del patrón MVC

Modelo

representa las estructuras de datos y es usualmente una interfaz de la base de datos

→ No mandatorio!

Vista

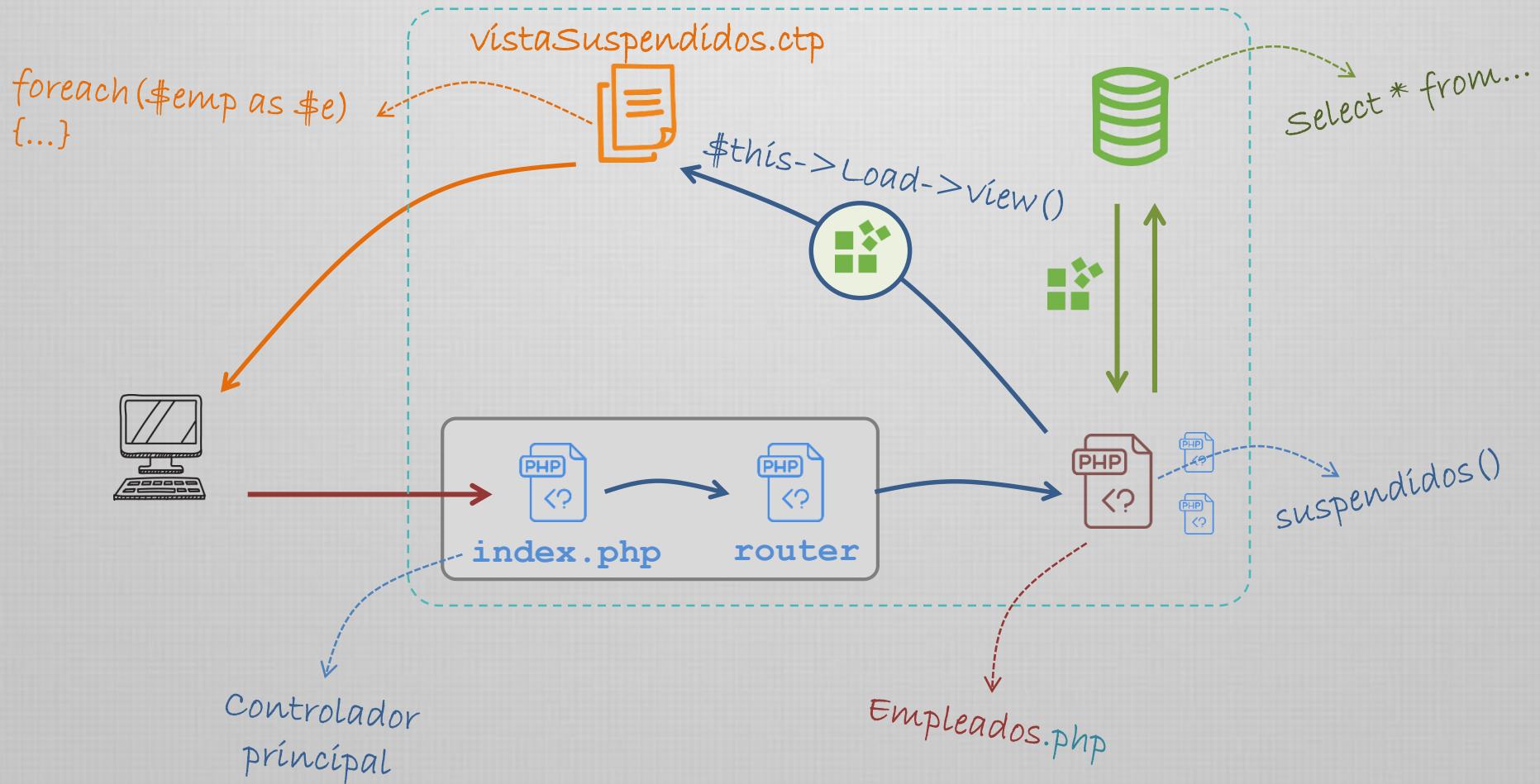
representa la información presentable al usuario (visual como en HTML o procesable como en RSS)

Controlador

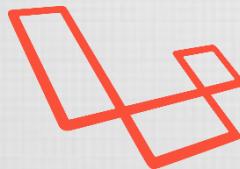
es el intermediario entre el modelo y la vista

Web Frameworks: Codeigniter

<http://nuestro.host.com/empleados/suspendidos>



Web Frameworks: Laravel



Framework PHP, basado en MVC
Actualmente uno de los frameworks mas populares



Creado por Taylor Otwell en 2011, como alternativa a CodeIgniter

Arquitectura bien definida

Rico ecosistema: Homestead, Valet, Lumen, etc

Comandos CLI

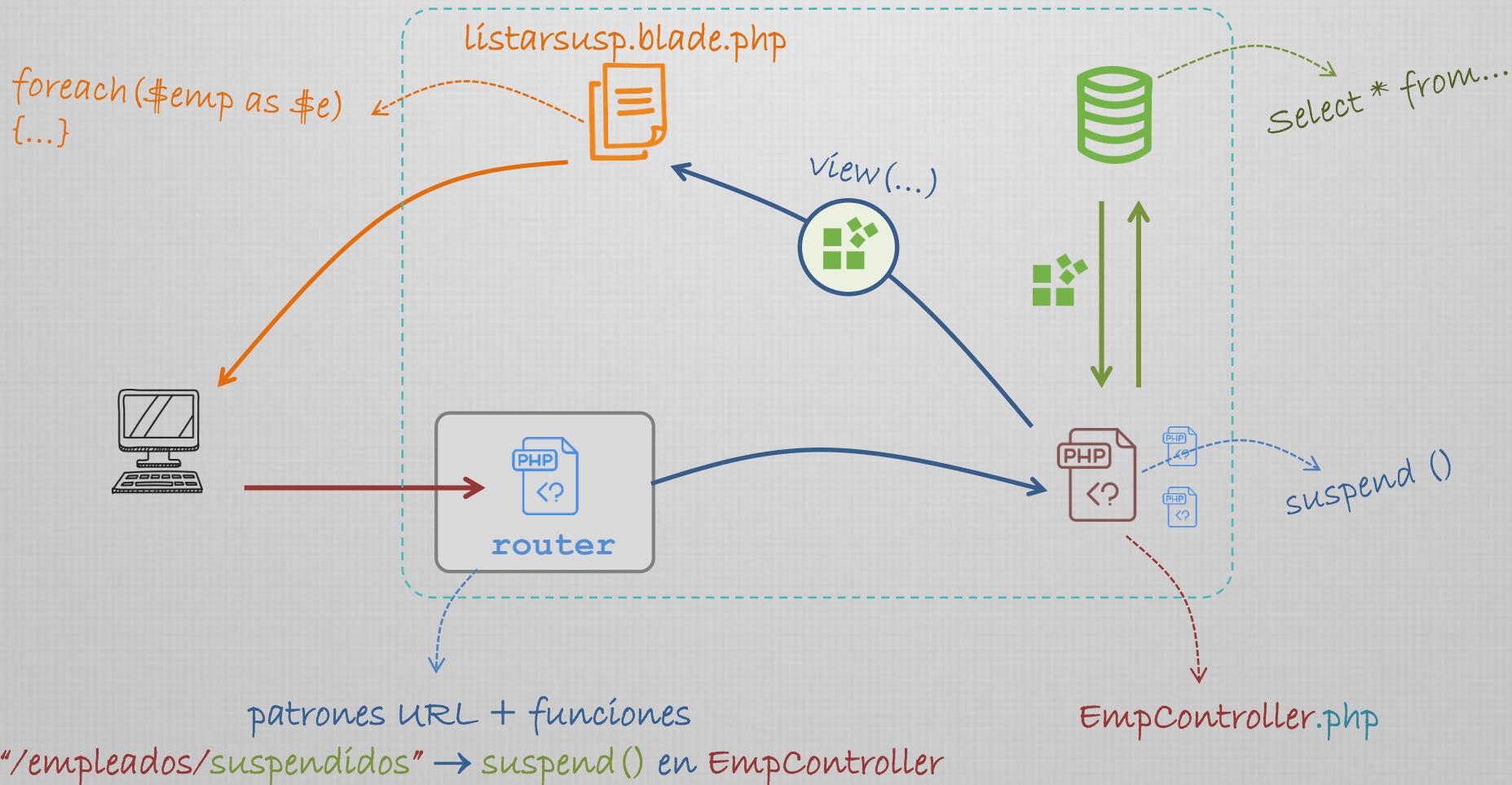
Mayor liberalismo en la implementación de modelos

Clara distribución del flujo de ejecución: route, middleware, controller, view

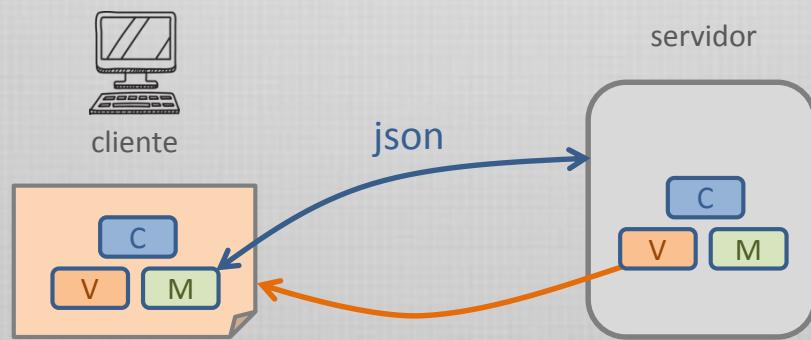
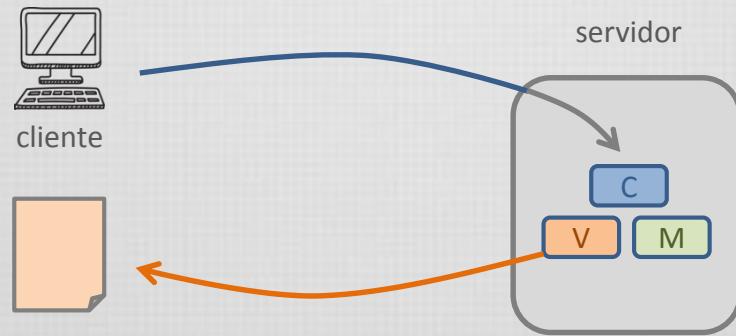
Templates propios: Blade

Web Frameworks: Laravel

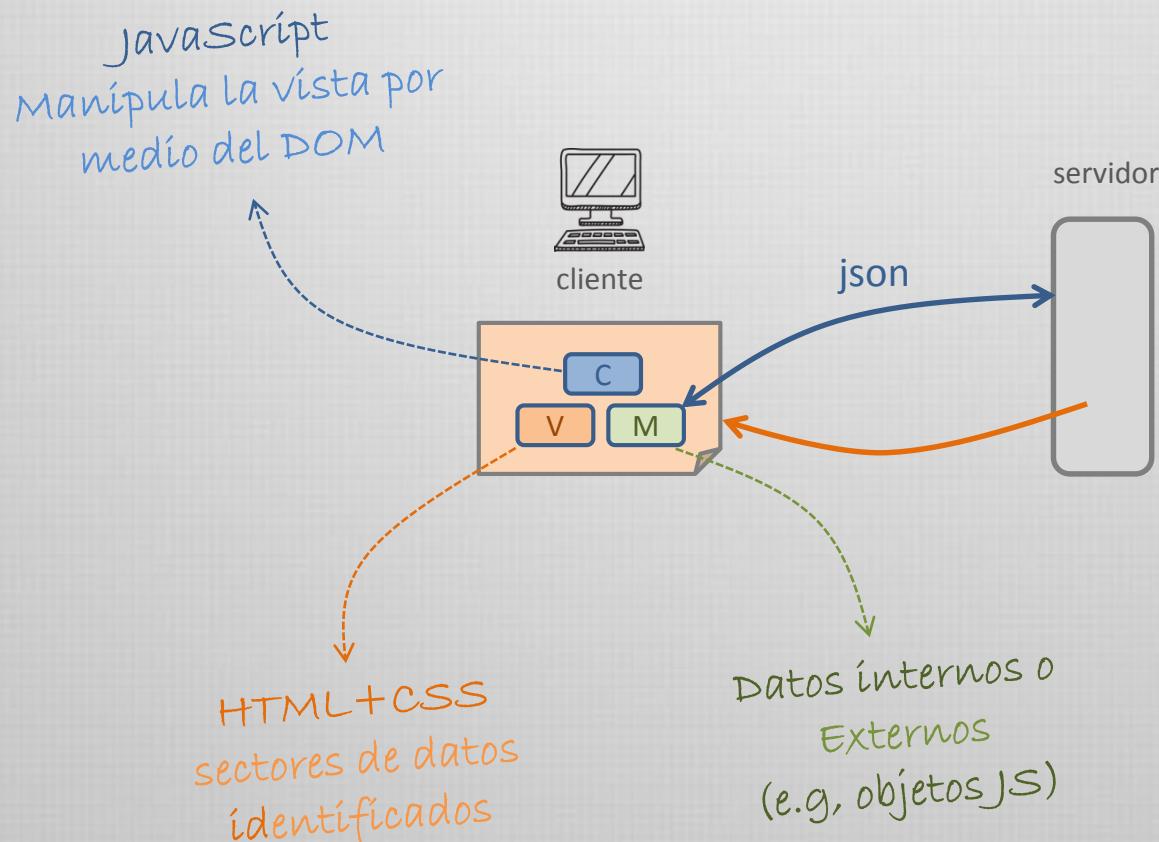
<http://nuestro.host.com/empleados/suspendidos>



MVC – Clientes y Servidores



MVC del lado cliente



MVC - Alternativas

Hay variaciones al modelo MVC tradicional, especialmente del lado cliente

MV* → El concepto de controlador dedicado no es mandatorio.

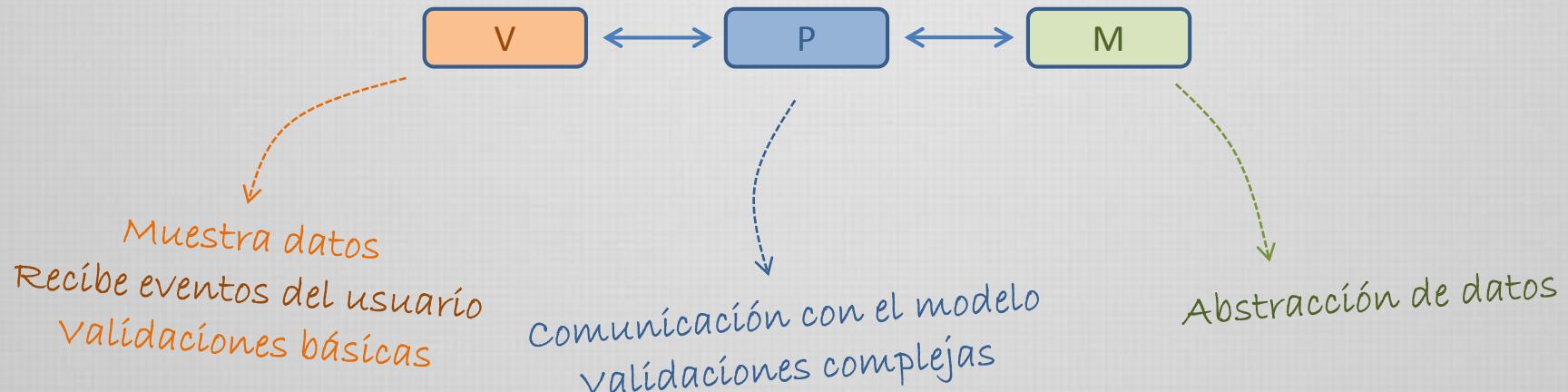
MVP → *Model-View-Presenter.*
La diferencia es la codificación del Presenter, que se comunica con la Vista por medio de una interfaz.
Usualmente no hay vínculo entre el Modelo y la Vista

MVVM → *Model-View-ViewModel*
Impulsado por Microsoft para su estructura de interfaces WPF-Silverlight
El ViewModel es una abstracción de la vista e implementa su comportamiento.

MVP

Persigue el mismo objetivo de separación de responsabilidades

La principal diferencia es **quién** controla los *inputs* del usuario



Variaciones

Passive View

La vista no tiene conocimiento del modelo.
El Presentador actualiza la vista reflejando
cambios en el modelo

Supervising Controller

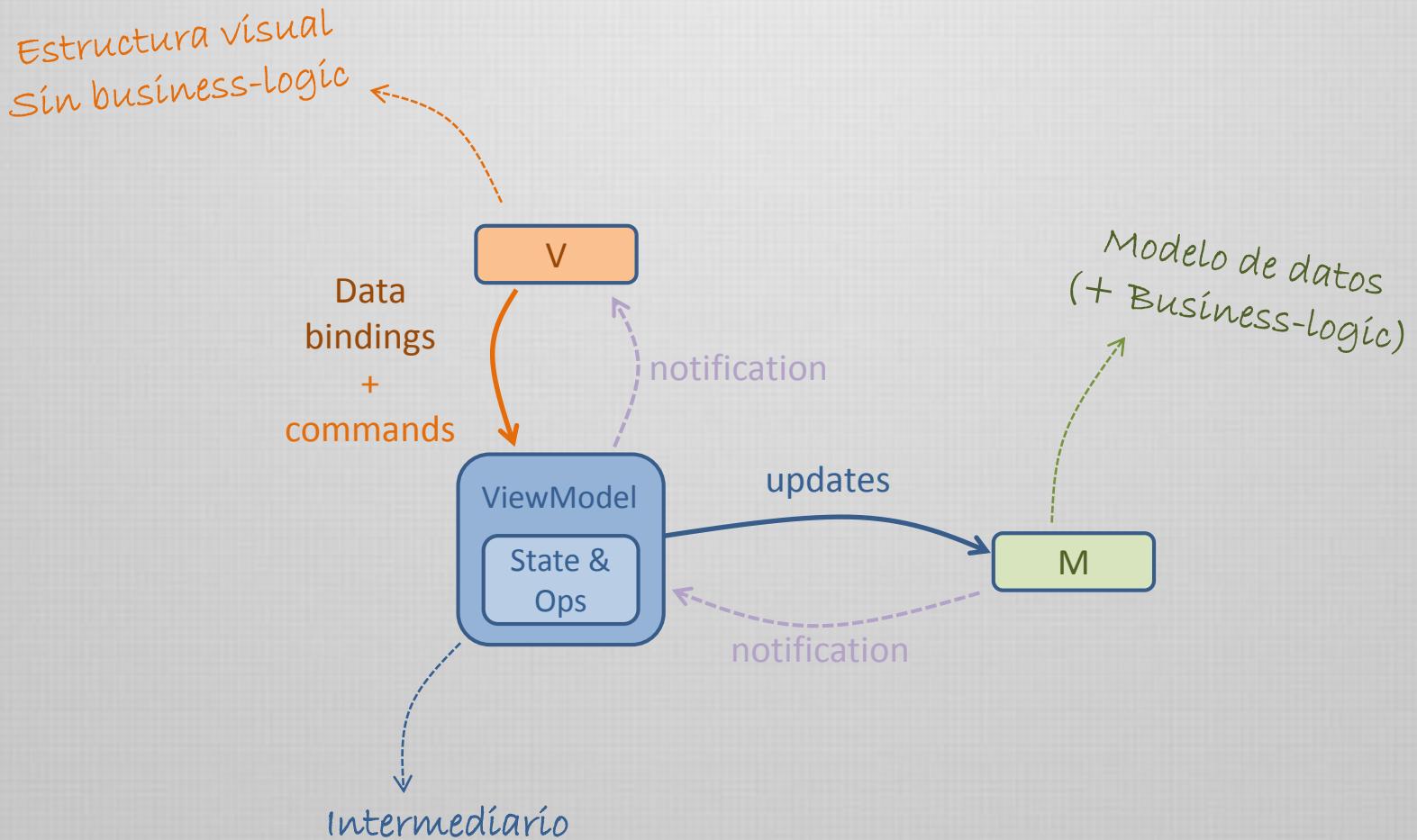
La vista interactúa con el modelo.
El Presentador actualiza el modelo y
manipula la vista sólo si es necesario.

MVP vs MVC

MVC	MVP
Separación de responsabilidades	Separación de responsabilidades
El controlador maneja los inputs del usuario y gestiona el modelo	La vista maneja los inputs del usuario e invoca al presentador si es necesario
La vista puede consultar al modelo directamente	La vista es completamente pasiva
No favorece <i>unit testing</i>	Favorece <i>unit testing</i>

Es un patrón más apropiado para aplicaciones *pesadas* del lado cliente
Preferido en aplicaciones con mucha interacción del usuario en la vista

MVMM



Recupera datos del modelo y los ofrece a la vista
La comunicación bidireccional se da por medio de
notificaciones (PropertyChanged)

Client-side Frameworks

Backbone.js

AngularJS

Ember.js

KnockoutJS

Dojo

YUI

Knockback.js

CanJS

Polymer

React

Mithril

Ampersand

Flight

Vue.js

MarionetteJS

TroopJS + RequireJS

...



JavaScript The MVC logo consists of the letters 'M', 'V', and 'C' each enclosed in a small green square.