

# Ingeniería de Aplicaciones Web

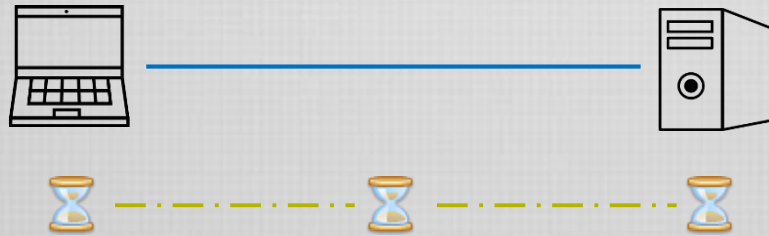
*Diego C. Martínez*

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

# Web Performance

Las aplicaciones web  
son amenazadas por  
ostentar una  
performance  
**no óptima**

- existe un procesamiento “*remoto*”
- comunicación cliente-servidor *no bidireccional*
- se solicitan *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*

# Web Performance

¿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](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

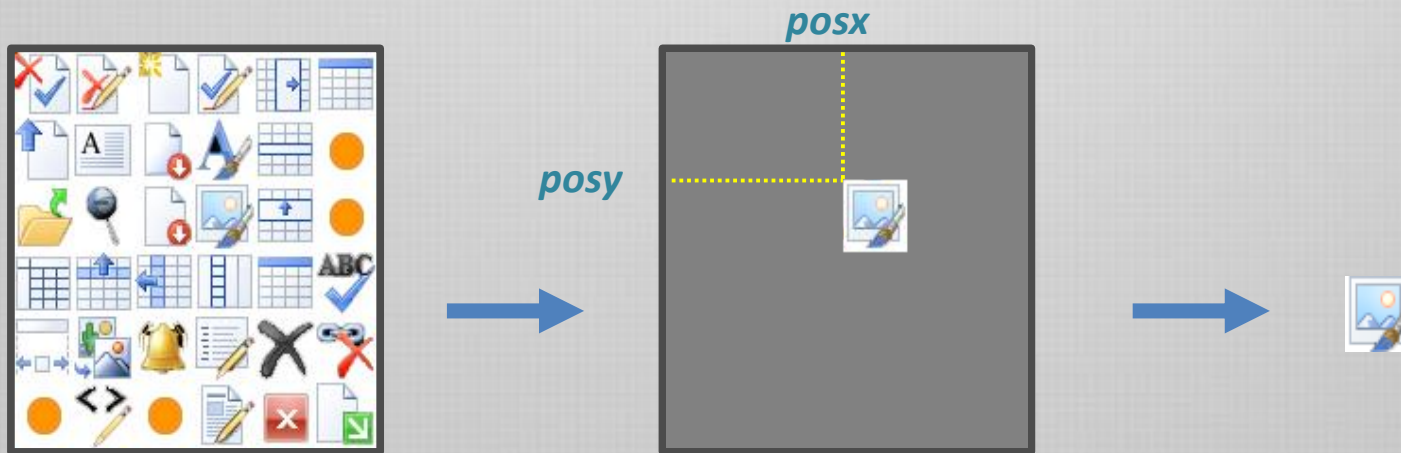
## CSS Sprites

---

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,R0lGODlhDAAMALMLAPN8ffBiYvWWlvrK
y/FvcPewsO9VVfajo+w6O/zl5estLv/8/AAAAAAAAAAAAAAAAACH5BAEAAAsALAAAA
AAMAAwAAAQzcElZyryTEHyTUgknHd9xGV+qKsYirKkwDYiKDBiatt2H1KBLQRFIJA
IKywRgmhwAI1EEADs=">
```

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"*



*proporciona una referencia del sistema trabajando*



*proporciona una estimación de completitud de la tarea*



*proporciona 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.*

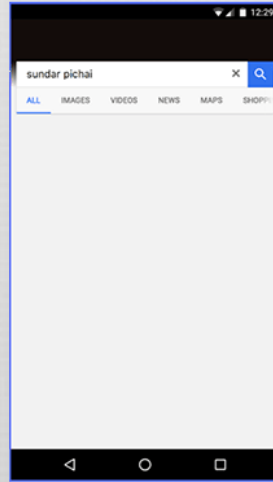
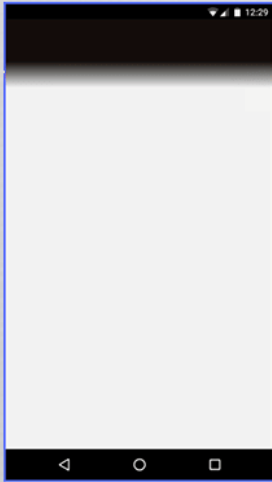
**FOUC (Flash of Unstyled Content)**

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

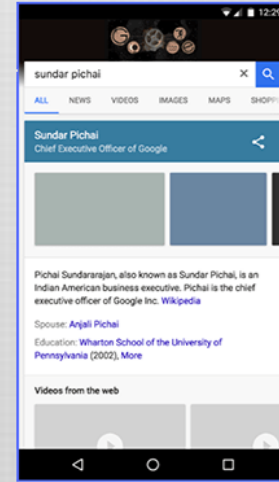
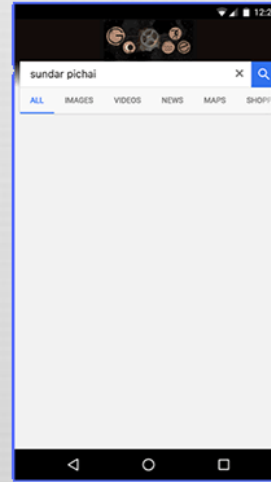
# Regla: Minimizar tiempos de carga



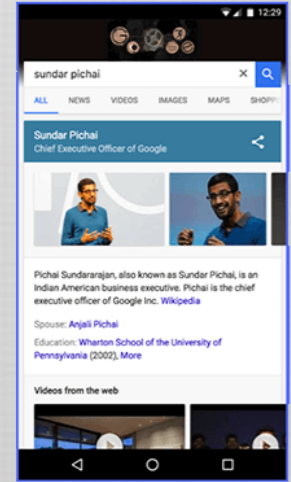
*First Paint  
(FP)*



*First Contentful  
Paint (FCP)*



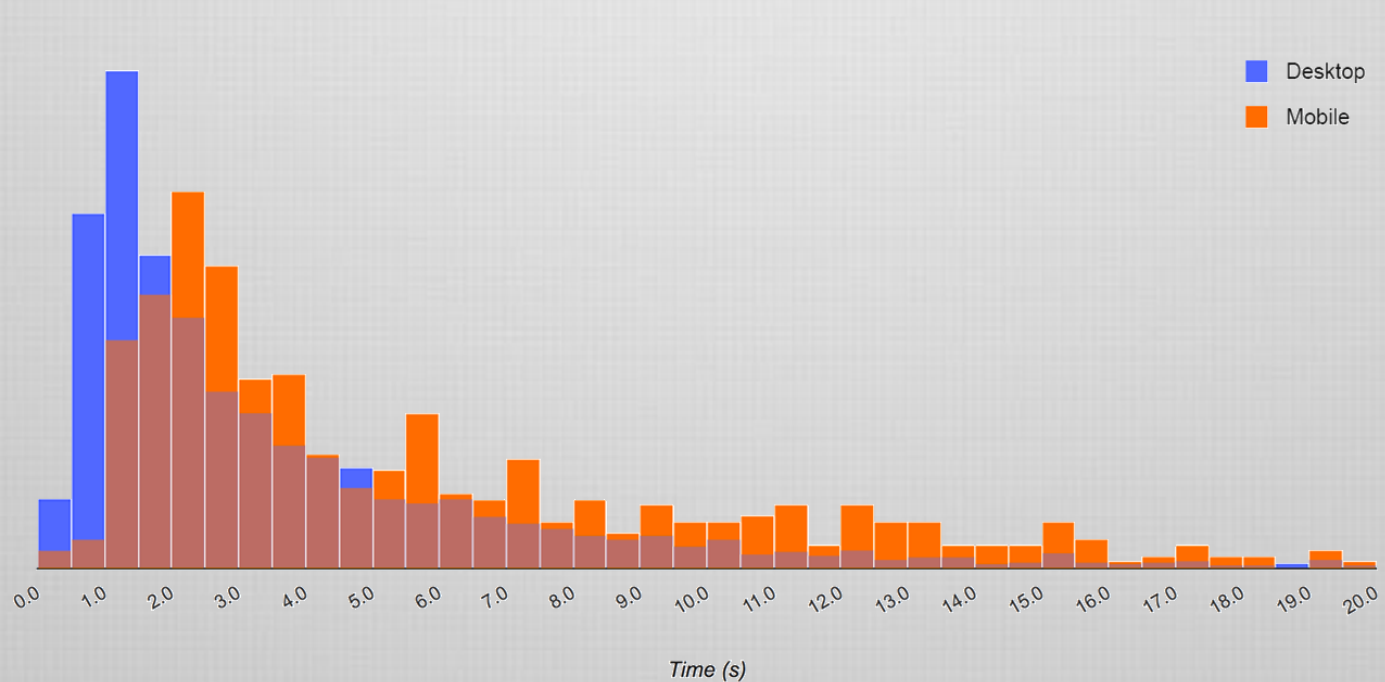
*First Meaningful  
Paint (FMP)*



*Time to Interactive  
(TTI)*

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

# Regla: Minimizar tiempos de carga

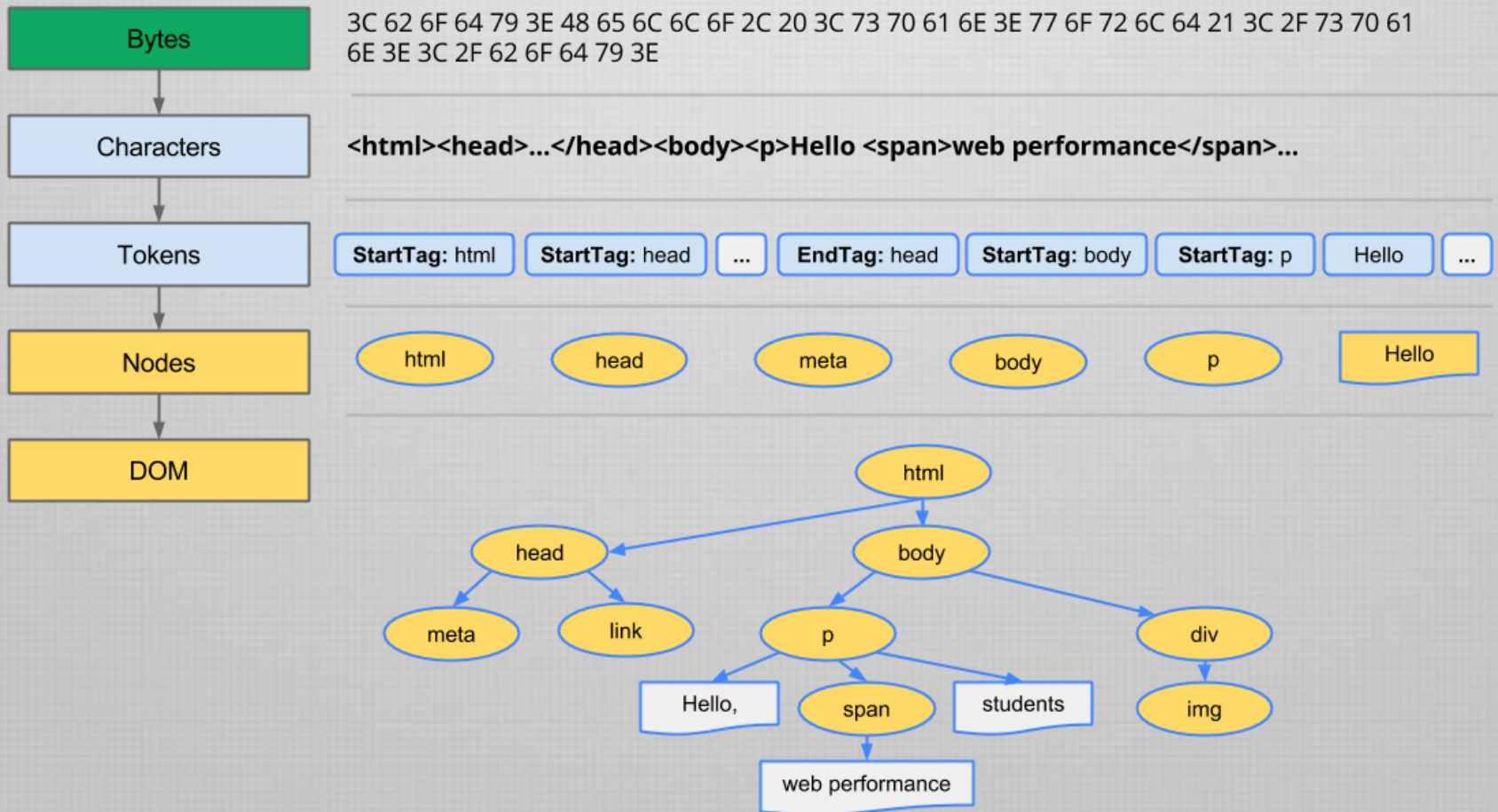


*Time to interactive (TTI)*  
*(renderizado y capaz de responder al input del usuario)*



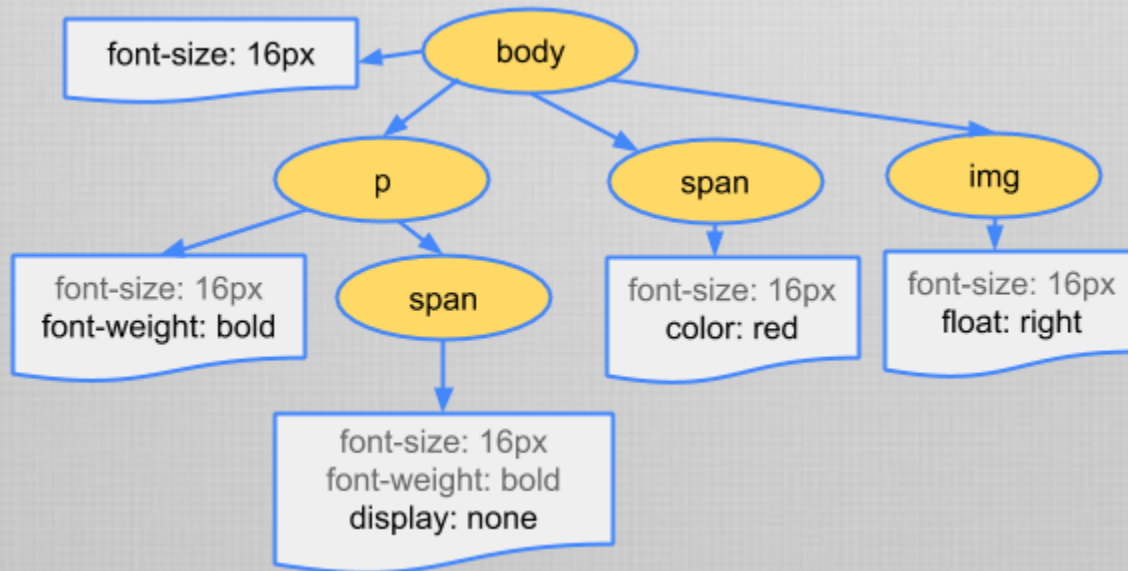
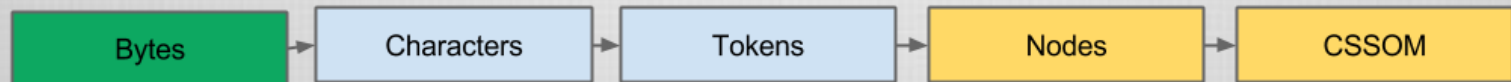
# Critical Rendering Path

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



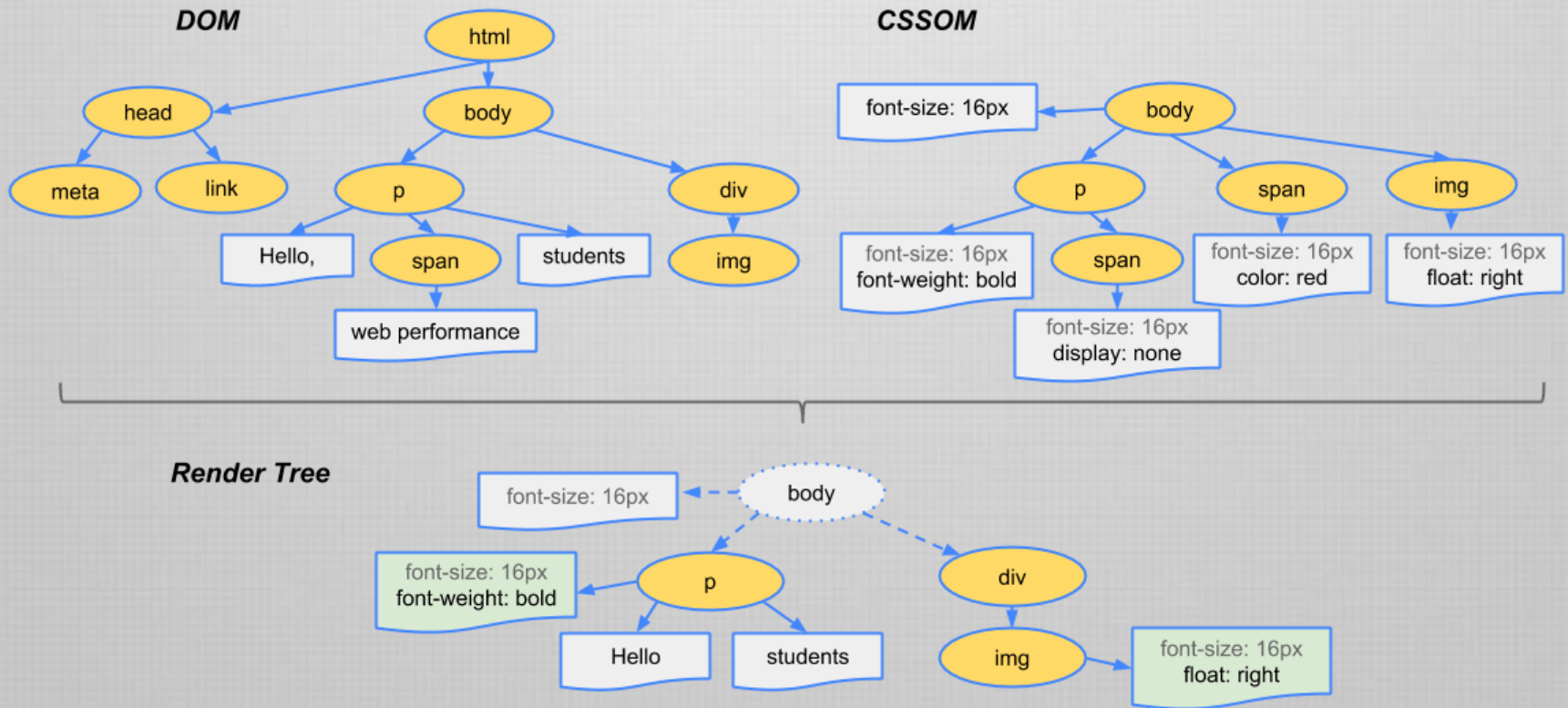
# Critical Rendering Path

*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.

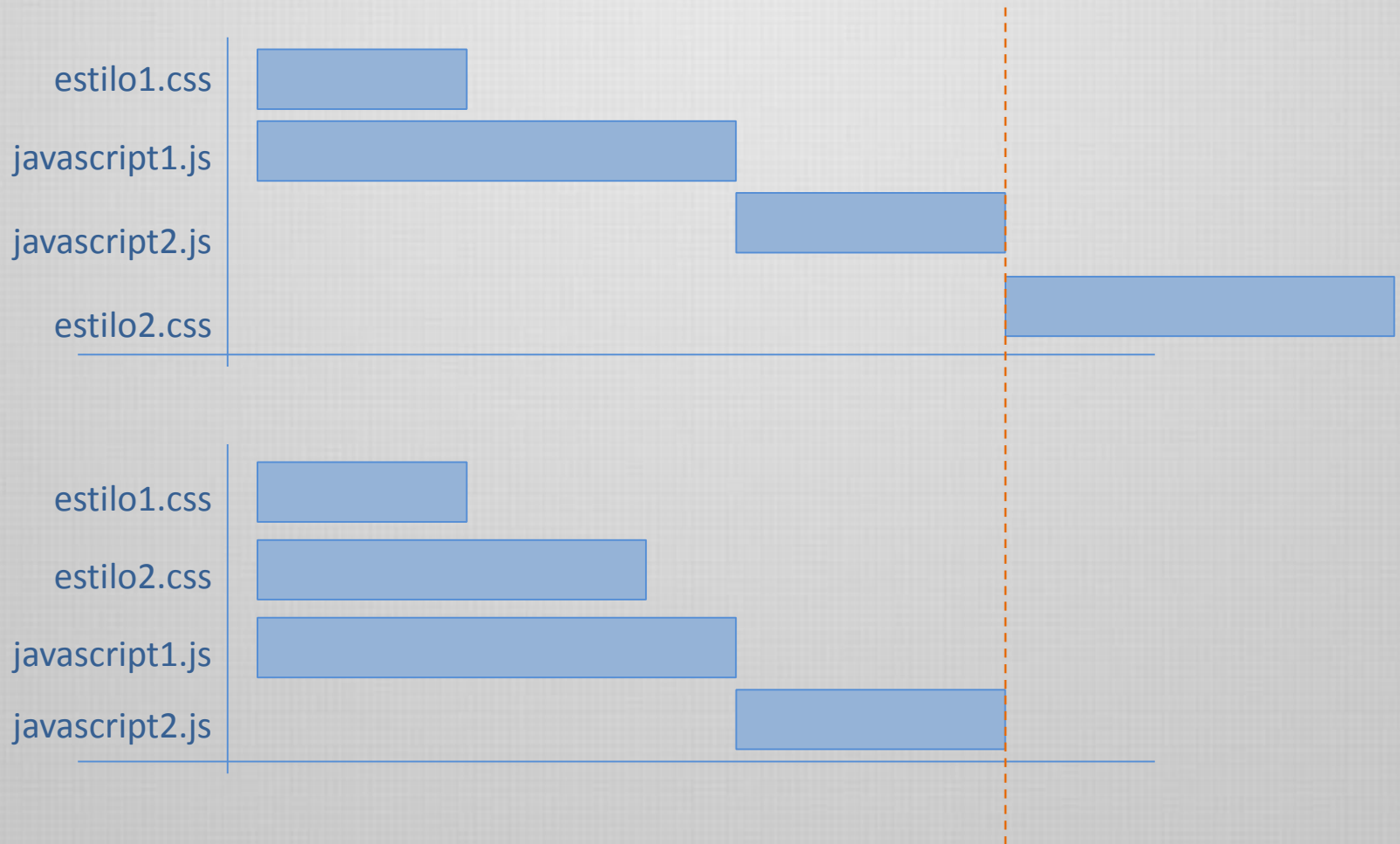
El estándar HTTP 1.1, *sugiere* que los clientes limiten la cantidad de conexiones simultáneas a un servidor:

*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 “diferiendo” 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>**.

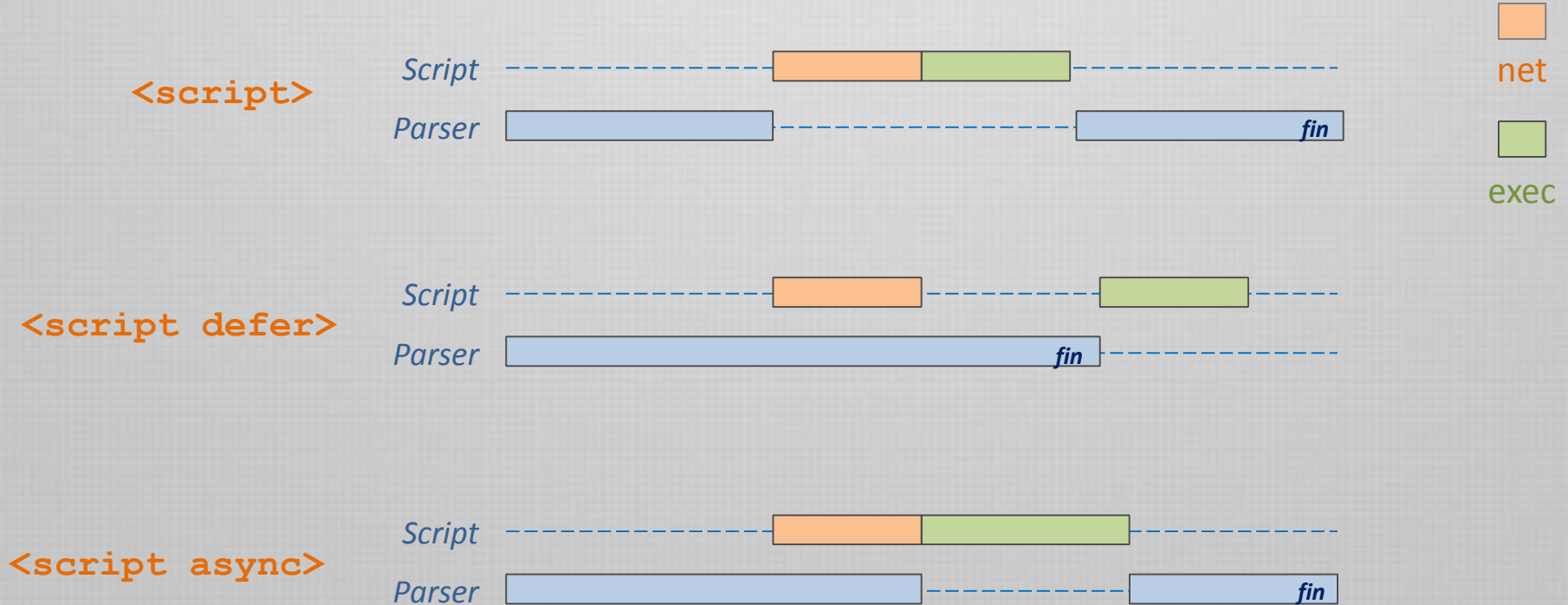
Cargar JavaScript en comentarios o strings  
Luego se interpretan con la función **eval()**

# 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.

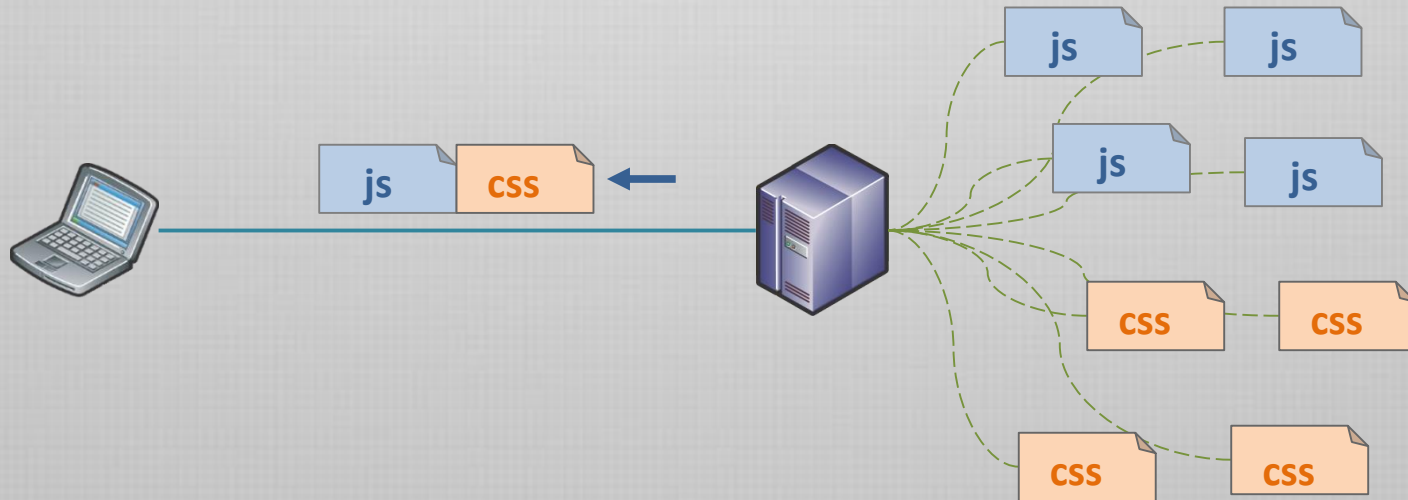
En particular los scripts y los css podrían combinarse en uno solo o en pocos.

*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.*

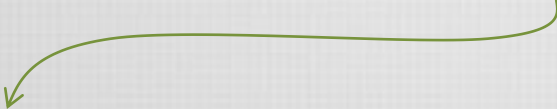
*Tal vez podrían ser combinados por un script del lado servidor.*



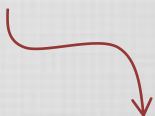
# Regla: Minimizar pedidos HTTP

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*.

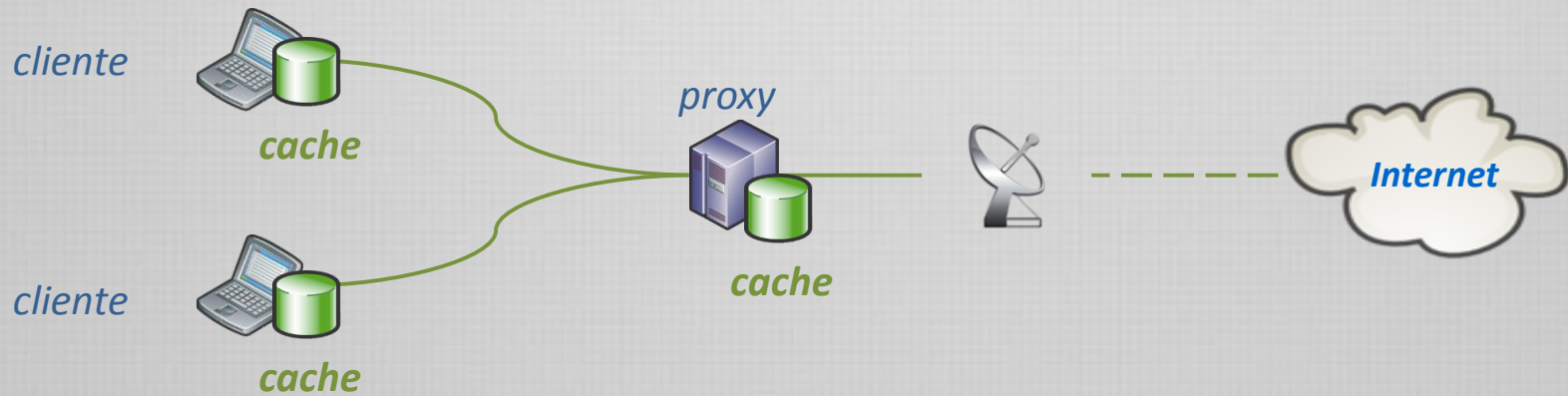
# Regla: Optimizar el uso de la cache

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 2014 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

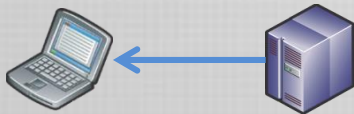
Pueden utilizarse también *Entity Tags (Etags)*.

*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 construídos con particularidades de cada servidor.



*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: Optimizar el uso de la cache

Los navegadores pueden variar en el tratamiento de la cache.



Firefox, por ejemplo, guarda TODO en la cache.

- Inclusive aquello que se recibe con el header **Cache-control: no-cache**.
- Asigna a todos los recursos una fecha de expiración *(posiblemente en el pasado)*
- Cuando un recurso en la cache alcanza la expiración, es *revalidado* o *recargado* nuevamente.
  - la *revalidación* **confirma** la validez del recurso.
  - método fuerte: **If-non-match** (Etags)
  - método débil: **Last-modified**  
respuesta posible: **304 Not modified**
  - la *recarga* implica solicitar completamente el recurso al servidor.

# 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: JSTMin, 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!*



# Regla: Minimizar redirecciones

A veces es necesario realizar redirecciones en los pedidos del cliente.

Posibles motivos:

- *los recursos fueron movidos a otra dirección*
- *se mantiene alguna convención sobre la estructura de las URLs*
- *interconexion entre partes de la aplicación web*
- *capturar URLs mal escritas o alias previsibles.*

Hay dos formas de hacer redirecciones:

- **del lado del servidor**, indicando un mensaje  
*301 Moved permanently*  
*302 Moved temporarily*
- **del lado cliente**, enviando un mensaje *200 OK* con un HTML con el meta-tag

```
<meta http-equiv="refresh" content="seg;url">
```

un JavaScript con

```
window.setLocation
```

En la medida de lo posible, evitar redirecciones.

De no ser posible, preferir las del lado servidor.