

# High Performance Web Sites

Much of the material derives from  
Steve Souders (SpeedCurve) and Tenni  
Theurer (Visa), formerly at Yahoo!

# Performance



(See Firebug's Net tab  
"timeline")

# Where Is The Most Of The Time Spent

A study of popular web pages and the time to download them showed that the vast majority of time is spent on the client side;

This is true even if the page has been cached

	Empty Cache	Full Cache
amazon.com	82%	86%
aol.com	94%	86%
cnn.com	81%	92%
ebay.com	98%	92%
google.com	86%	64%
msn.com	97%	95%
myspace.com	96%	86%
wikipedia.org	80%	88%
yahoo.com	95%	88%
youtube.com	97%	95%

percentage of time spent on the front-end

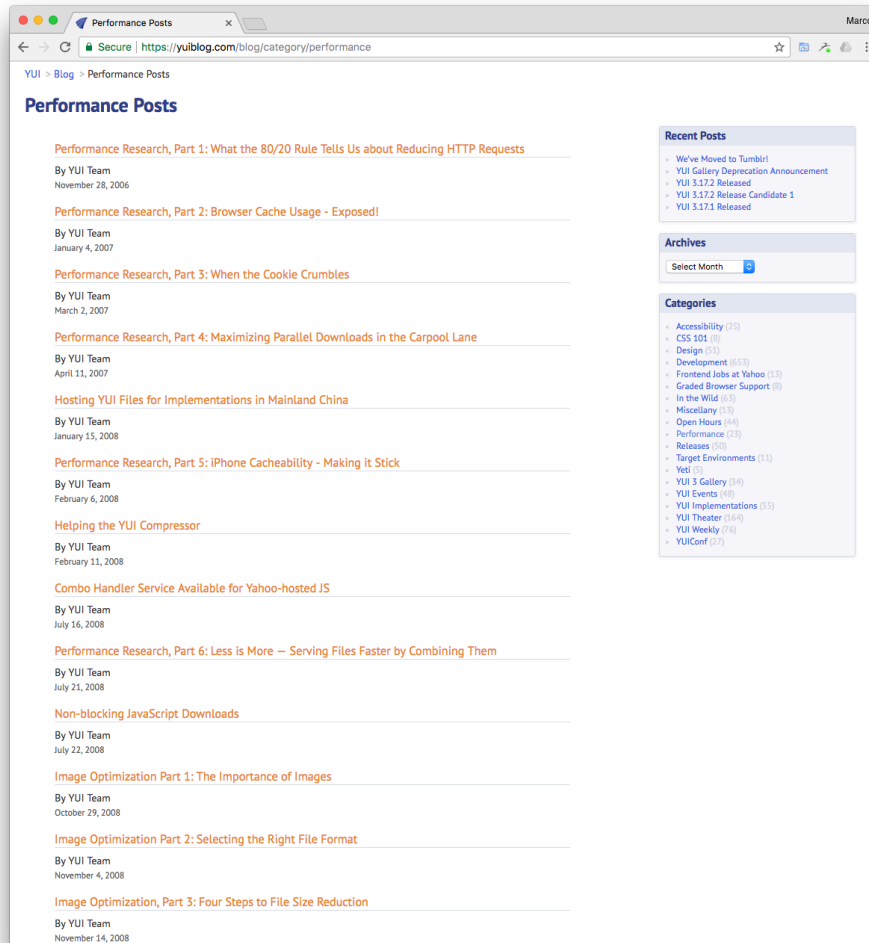
# The Performance Golden Rule

- 80-90% of the end-user response time is spent on the front-end. So start there
  - Greater potential for improvement
  - Simpler to optimize
  - Proven to work

# 80/20 Performance Rule

- Vilfredo Pareto:
  - “80% of the consequences come from 20% of the causes”
  - See:  
<https://yuiblog.com/blog/2006/11/28/performance-research-part-1/>
- Software Engineering: “80% of time spent in 20% of the code”
- Focus on the 20% that affects 80% of the end-user response time
- Web pages: 10% fetch HTML, 90% spent on fetching Images, scripts and stylesheets
- I.e. Start at the front-end

# Yahoo Interface / Engineering Blogs



Most of the work on how to improve the performance of Web sites was done by Steve Souders and Tenni Theurer at Yahoo.com. Original blog site contains performance posts (2006-2011)

<http://yuiblog.com/blog/category/performance>

Steve Souders became Head Performance Engineer at Google, then to Fastly, a CDN, and now is at SpeedCurve. See his work at:

<http://stevesouders.com>

“SpeedCurve provides insight into the interaction between performance and design to help companies deliver fast and engaging user experiences.”

Tenni Theurer moved to VISA.

# Browser Cache Experiment

- An “empty cache” means the browser bypasses the disk cache and has to request all the components to load the page.
- A “full cache” means all (or at least most) of the components are found in the disk cache and the corresponding HTTP requests are avoided
- Experiment: Try to determine what the percentage of people is who load a home page when there are no elements of the page in the user’s cache?
- Solution: add a new image (a pixel) to your page, e.g.

```

```

- With the following response headers:

```
Expires: thu, 15 Apr 2008 20:00:00 GMT          (an earlier date than today)
```

```
Last-Modified: Wed, 28 Sep 2009 23:49:57 GMT   (today’s date)
```

- The Expires makes sure the page is not cached; the Last-Modified makes sure the server will have to check if blank.gif has changed
- Requests from a browser will produce one of these response status codes
  - 200 – the server is sending back the image implying the browser does not have the image in its cache
  - 304 – the browser has the image in its cache, and the server responds saying it has not been modified
- Compute the following numbers:
  - Percentage of users who view with an empty cache ::= 
$$\frac{(\# \text{ unique users with at least one 200 response})}{(\text{total } \# \text{ unique users})}$$
  - Percentage of page views that are done with an empty cache ::= 
$$\frac{(\text{total } \# \text{ of 200 responses})}{(\# \text{ of 200} + \# \text{ of 304 responses})}$$
- See: <https://yuiblog.com/blog/2007/01/04/performance-research-part-2/>

# Surprising Results Lessons:

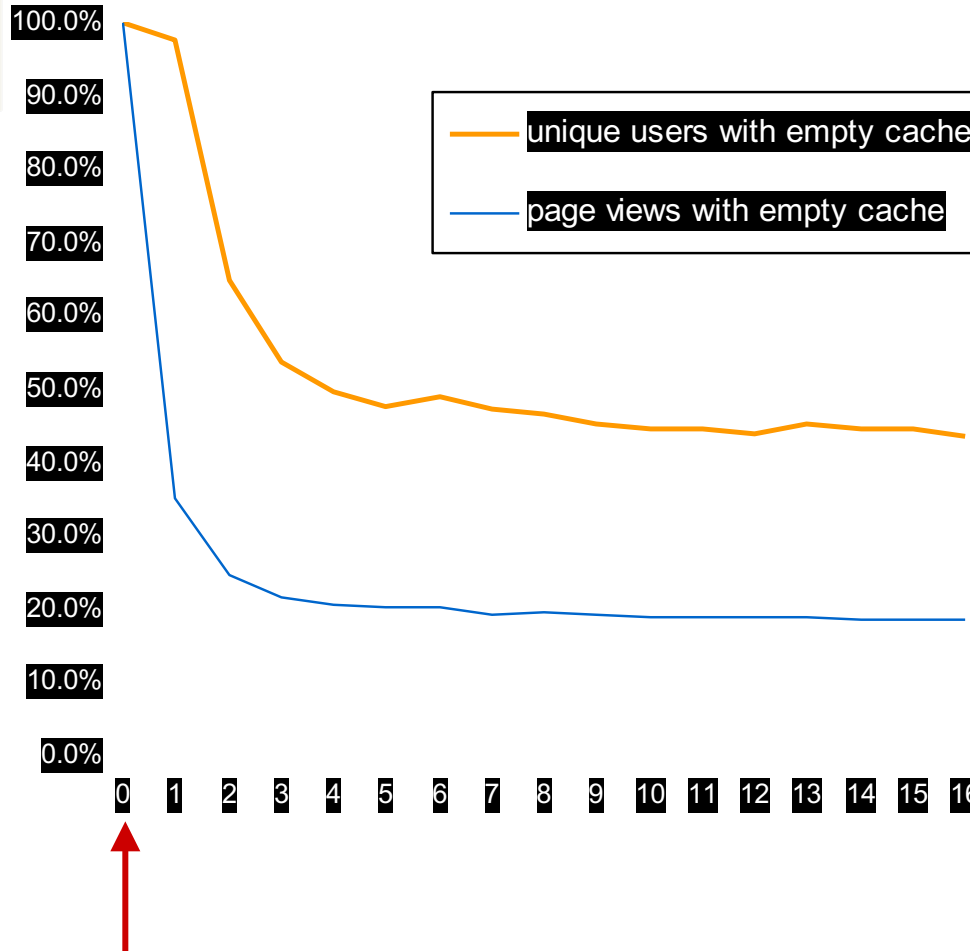
The empty cache user is more prevalent than one might think

users with  
empty cache

40-60%

page views with  
empty cache

~20%



On the first day no one has the images cached, but over time more people have the image until a steady state is reached; Result: 40-60% of yahoo users have an empty cache experience and 20% of page views are done with an empty cache



# Impact of Cookies on Response Time

<b>Cookie Size</b>	<b>time</b>	<b>Delta</b>
0 bytes	78ms	0ms
500 bytes	79ms	+1 ms
1000 bytes	94ms	+16ms
1500 bytes	109ms	+31ms
2000 bytes	125ms	+47ms
2500 bytes	141ms	+63ms
3000 bytes	156ms	+78ms

**ms = millisecond (at ~800kbps, DSL speed)**

(Note: today's cookies are much bigger in size)

See: <http://yuiblog.com/blog/2007/03/01/performance-research-part-3/>

# Analysis of Cookie Sizes Across the Web

## Website total Cookie Size (2007)

Amazon	60 bytes
Google	72 bytes
Yahoo	122 bytes
Cnn	184 bytes
Youtube	218 bytes
msn	268 bytes
eBay	331 bytes
MySpace	500 bytes

## Lessons

- eliminate unnecessary cookies
- keep cookie sizes low
- set cookies at appropriate domain level
- set Expires date appropriately an earlier date or none removes the cookie sooner
- Unfortunately today's cookie sizes are much bigger

*Experiment in Firefox:* click on Tools | Preferences | Privacy | Accept cookies from sites | select Keep until: "ask me every time" for cookies.  
Then load [www.amazon.com](http://www.amazon.com)

# The “initial” 14 Rules

1. Make fewer HTTP requests
2. Use a CDN (content distribution network)
3. Add an Expires header
4. Gzip components
5. Put stylesheets at the top
6. Move scripts to the bottom
7. Avoid CSS expressions
8. Make JS and CSS external
9. Reduce DNS lookups
10. Minify JS
11. Avoid redirects
12. Remove duplicate scripts
13. Configure Etags
14. Make AJAX cacheable

See: <https://developer.yahoo.com/performance/rules.html>

Details on all of the rules to follow

# Rule 1: Make fewer HTTP requests

- Most browsers download only two resources at a time from a given hostname, as suggested in the HTTP/1.1 specification
  - However, some browsers open more than two connections per hostname
- To reduce the number of HTTP requests
  - Combine scripts
  - Combine style sheets
  - Combine images into an image map

```

<map name="map1">
<area shape="rect" coords="0,0,31,31" href="home.html"
      title="Home">
. . .
</map>
```

- Combine images using “sprites” (see next slide)
- Drawbacks
  - Images must be contiguous
  - Defining area coordinates is error prone

# CSS Sprites

- An image sprite is a collection of images put into a single image
- Using images sprites reduces the number of server requests and saves bandwidth
- Consider `img_navsprites.gif`

which includes 3 separate images



## The code

```
<!DOCTYPE html>
<html><head><style>
img.home { width:46px; height:44px;
            background:url(img_navsprites.gif) 0 0;  }
img.next { width:43px ; height:44px;
            background:url(img_navsprites.gif) -91px 0;  }
</style></head>
<body>

<br><br>

</body></html>
```



- `width:46px;height:44px;` - Defines the portion of the image we want to use
- `background:url(img_navsprites.gif) 0 0;` - Defines the background image and its position (left 0px, top 0px)
- Check [google.com](http://google.com) with Firebug's Net tab. See the image `images/nav_logo242.png`

# Rule 2: Use a CDN

amazon.com	Akamai
aol.com	Akamai
cnn.com	cdn.turner.com
ebay.com	Akamai, Mirror Image
google.com	Google CDN
msn.com	SAVVIS
myspace.com	Akamai, Limelight
wikipedia.org	- not using CDN --
yahoo.com	Akamai
youtube.com	Google CDN, Akamai
apple.com	Akamai

- Content Distribution  
Networks have servers around the world
- They distribute your content so downloads can come from a nearby location
- Major CDN providers are
  - Akamai
    - Distributes OS X and iOS updates
  - SAVVis
  - Limelight
  - OnApp
  - BityGravity
  - Amazon CloudFront
- Free CDNs
  - CloudFlare
  - BootstrapCDN
  - Incapsula

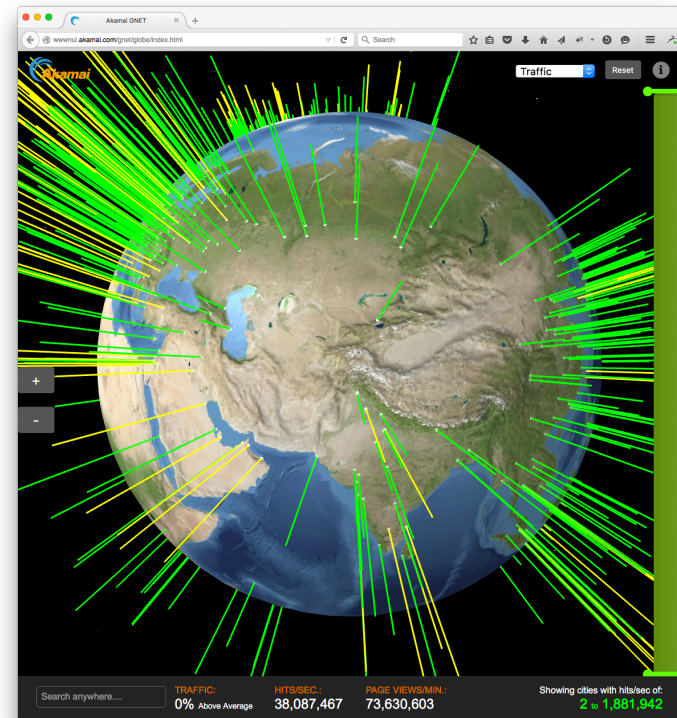
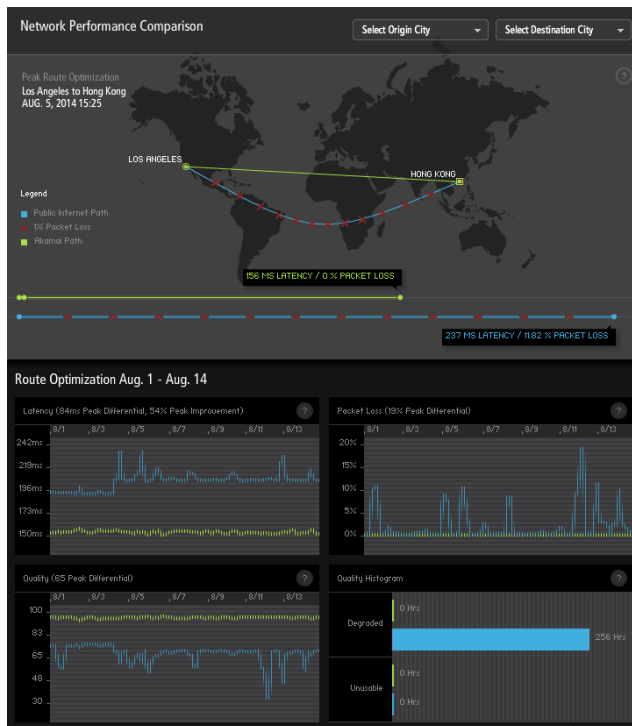
# More on Use a CDN: Akamai

Distribute your static content before distributing your dynamic content

see <http://www.akamai.com/html/technology/dataviz2.html>

(Follow visualizing the internet (twice))

<http://wwwnui.akamai.com/gnet/globe/index.html>



# Rule 3: Add an Expires Header

- All caches use a set of rules to determine whether to deliver an object from the cache or request a new version
  - If the object's headers tell the cache not to keep the object, it won't
  - If the object is authenticated or secure, it won't be cached
  - A cached object is "fresh" (able to be sent to a client without checking with the server) if
    - It has an expiry time or other age-controlling directive that is set and still within the fresh period
    - If a browser cache has already seen the object and has been set to check once a session
    - If a proxy cache has seen the object recently, and it was modified long ago
  - If an object is "stale", the origin server will be asked to validate the object or tell the cache whether the copy that is has is still good
- As part of HTTP protocol there is a Cache-Control header
  - Cache-Control is an alternative to Expires
  - When `cache-control: max-age` is present, the response is stale if its current age is greater than the age value given (in seconds) at the time of a new request for the resource
  - The max-age directive on a response implies that the response is cacheable
- HTTP headers are sent by the server before the HTML, and only seen by the browser and any intermediate caches. Typical HTTP 1.1 response headers might look like this:

```
HTTP/1.1 200 OK
Date: Fri, 30 Oct 1998 13:19:41 GMT
Server: Apache/1.3.3 (Unix)
Cache-Control: max-age=3600, must-revalidate
Expires: Fri, 30 Oct 1998 14:19:41 GMT
Last-Modified: Mon, 29 Jun 1998 02:28:12 GMT
ETag: "3e86-410-3596fbbc"
Content-Length: 1040
Content-Type: text/html
```



# More on Adding Expire Headers

- Here is a way to add an Expires header to files using the Apache httpd.conf file; add the lines:

```
<FilesMatch "\.(ico|pdf|flv|jpg|jpeg|png|gif|js|css|swf)$">  
Header set Expires "Thu, 15 Apr 2010 20:00:00 GMT"  
</FilesMatch>
```

- An Apache module enables / modifies expire headers:
- Apache Module mod\_expire:
  - [http://httpd.apache.org/docs/2.2/mod/mod\\_expires.html](http://httpd.apache.org/docs/2.2/mod/mod_expires.html)
  - This module controls the setting of the Expires HTTP header and the max-age directive of the Cache-Control HTTP header in server responses. The expiration date can set to be relative to either the time the source file was last modified, or to the time of the client access.
  - ExpiresDefault "access plus 1 month"
  - ExpiresByType image/gif "modification plus 5 hours 3 minutes"
- For information on adding Expire headers and cache control to IIS 7, see:
  - <http://www.iis.net/configreference/system.webserver/staticcontent/clientcache>
  - <http://stackoverflow.com/questions/10825497/iis-7-5-how-do-you-add-a-dynamic-http-expire-header>

# Rule 4: Gzip Components

	HTML	Scripts	Stylesheets
amazon.com	x		
aol.com	x	some	some
cnn.com			
ebay.com	x		
froogle.google.com	x	x	x
msn.com	x	deflate	deflate
myspace.com	x	x	x
wikipedia.org	x	x	x
yahoo.com	x	x	x
youtube.com	x	some	some

gzip scripts, stylesheets, XML, JSON (not images, PDF)

- Compression works when a web server like Apache is set up to "compress" resources and when a client's browser accepts such compressed resources.
- During the initial negotiation, if both browser and server support at least one "common" compression method (gzip, compress, etc) then the transfer is compressed.
- Client:  
GET / HTTP/1.1  
**Accept-Encoding:** gzip,deflate
- Server:  
HTTP/1.1 200 OK  
Vary: Accept-Encoding  
**Content-Encoding:** gzip
- 90% of browsers support compression

<http://websites tips.com/optimization/#server-side-compression>

# More on Gzip'ing Components

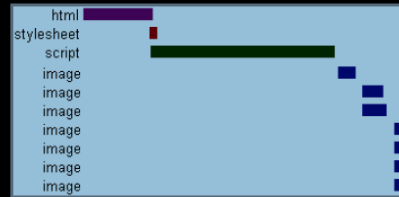
- A couple of Apache modules to enable / modify compression:
- Apache Module mod\_deflate (supports gzip and deflate):
  - [http://httpd.apache.org/docs/2.0/mod/mod\\_deflate.html](http://httpd.apache.org/docs/2.0/mod/mod_deflate.html)
  - Allows output from the server to be compressed before being sent to the client
  - `AddOutputFilterByType DEFLATE text/html text/plain text/xml`
- Apache Module mod\_gzip:
  - <http://sourceforge.net/projects/mod-gzip/>
  - This is the older compression module for Apache
- See “Compressing Web Content with mod\_gzip and mod\_deflate”:
  - <http://www.linuxjournal.com/article/6802>
- Compression can be configured in IIS 7, using the IIS Manager:
  - <https://technet.microsoft.com/en-us/library/cc771003%28v=WS.10%29.aspx>

# Rule 5: Put Stylesheets at the top

- **Stylesheets block rendering in IE**; IE will examine all stylesheets before starting to render, so its best to
  - **Put stylesheets in the <head>**
- Firefox doesn't wait, it renders objects immediately and re-draws if it finds a stylesheet; that causes flashing during loading;
  - Put stylesheets in the <head>
- Use <link> (not @import)
  - There are two ways to load stylesheets
  - <link> includes the stylesheet in the web page  
`<link href="styles.css" type="text/css" />`
  - @import allows you to import one style sheet into another  
`<style type="text/css">@import url("styles.css");</style>`
- General Rule for using @import
  - Link to a stylesheet for a specific page, but import a stylesheet that applies to all pages

# Rule 6: Move Scripts to the Bottom

scripts block parallel downloads across all hostnames



scripts block rendering of everything below them in the page

`script defer` attribute is not a solution

- blocks rendering and downloads in FF
- slight blocking in IE

- As the loading of JavaScript can cause the browser to stop rendering the page until the JavaScript is fully loaded, one can avoid the delay by moving scripts to the bottom

- Example: move jQuery and Bootstrap libraries reference **right before** `</BODY>`

- A second option is the defer attribute `<script type="text/javascript" defer="defer"> some script .... </script>`

- “defer” script attribute indicates that the script is not going to generate any document content. The browser can continue parsing and drawing the page

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script> and scroll down to "defer"

# Rule 7: Avoid CSS Expressions

- Used to set CSS properties dynamically in IE

```
Width: expression(document.body.clientWidth < 600 ? "600px" : "auto");
```

- Problem: expression may execute many times
  - Mouse moves, key press, resize, scroll, etc
- Alternatives
  - One-time expressions
  - Event handlers
- Expression overwrites itself
- Note: fixed in later versions of IE

```
<style>
P { background-color: expression(altBgcolor(this)); }
</style>
<script>
function altBgcolor(elem) {
    elem.style.backgroundColor =
    (new Date()).getHours()%2 ? "F08A00" : "#B8D4FF"; }
</script>
```

# Rule 8: Make JS and CSS External

- JavaScript can be placed inline, e.g

```
<script type="text/javascript">var foo = "bar"; </script>
```

- Or as an external script, e.g.

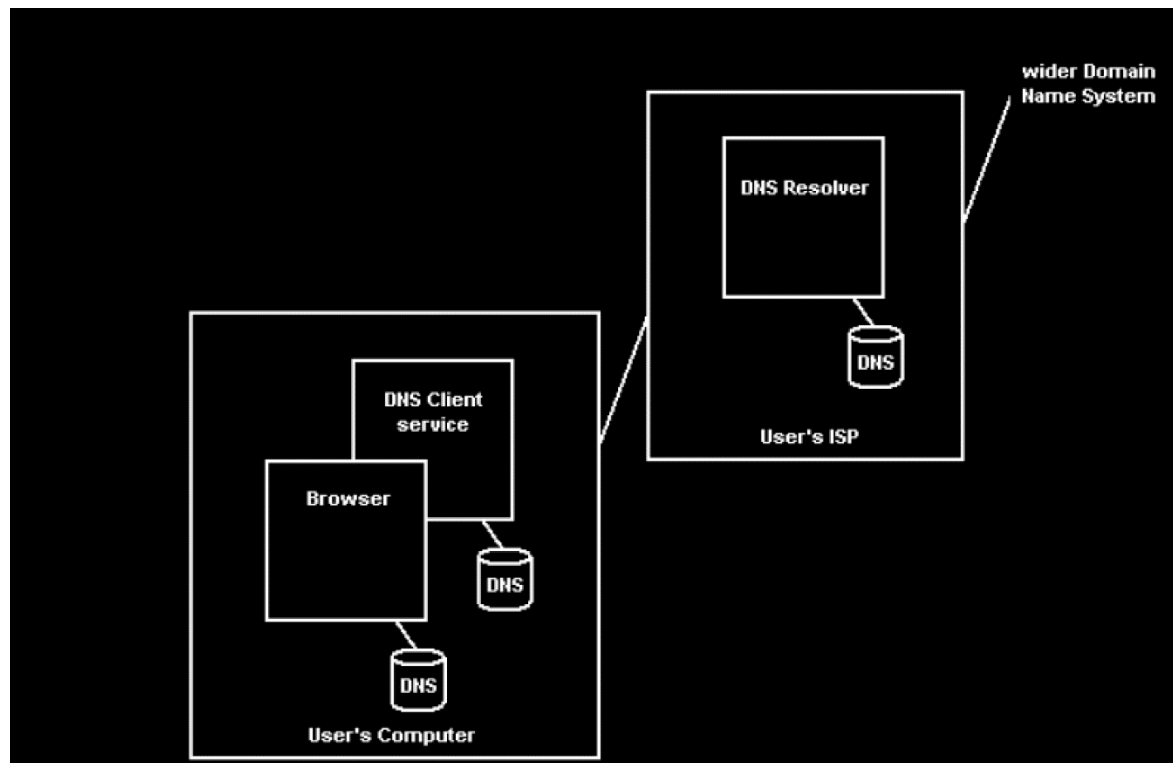
```
<script src="foo.js" type="text/javascript"></script>
```

- Placing JavaScript and CSS inline makes the document bigger
- Making JavaScript and CSS external implies more HTTP requests, but
  - As HTML documents are not typically cached, so inlining JavaScript code will cause the same bytes to be downloaded on every page view
  - **External JS and CSS can be cached**
- Defining JavaScript externally is typically better
- Download external files after onload

```
window.onload = downloadComponents;  
function downloadComponents ( ) {  
    var elem = document.createElement("script");  
    elem.src = http://.../file1.js;  
    document.body.appendChild(elem);  
    . . . . .  
}
```

# Rule 9: Reduce DNS lookups

- Typically each look up takes 20 – 120 milliseconds
- DNS lookups will block parallel downloads
- The operating system and the browser both have DNS caches
- As a general rule it is best to reduce the number of unique hostnames used in a web page





# Rule 10: Minify JavaScript

	Minify External?	Minify Inline?
www.amazon.com	no	no
www.aol.com	no	no
www.cnn.com	no	no
www.ebay.com	yes	no
froogle.google.com	yes	yes
www.msn.com	yes	yes
www.myspace.com	no	no
www.wikipedia.org	no	no
www.yahoo.com	yes	yes
www.youtube.com	no	no

minify inline scripts, too

- Minification, or minify, is the process of removing all unnecessary characters from source code, without changing its functionality
  - Unnecessary characters usually include: white space characters, new line characters, comments, and sometimes block delimiters used to add readability to code, but are not required for execution
  - Minified code is especially useful for interpreted languages because it reduces the amount of data that must be transferred
- There are many programs that minify JavaScript code, see <http://en.wikipedia.org/wiki/Minify> (JSmIn, packer, Google Closure Compiler)
- See these websites for examples of minification:  
[www.google.com](http://www.google.com)  
[purecss.io](http://purecss.io)

# Minified JavaScript

- Example of "minified code" is the Google Maps engine at:  
<http://maps.google.com/>
- An example of "minified library" is Bootstrap at:  
<https://maxcdn.bootstrapcdn.com/bootstrap/3.3.0/js/bootstrap.min.js>
- You can use PHP code from Google (HTTP server for minification) to do the job of minifying CSS & JavaScript :  
<http://code.google.com/p/minify/>
- The original minifier from Crockford:  
<http://www.crockford.com/javascript/jsmin.html>
- An on-the-fly minifier of JavaScript/CSS for IIS can be found here:  
[http://highoncoding.com/Articles/777\\_Minify\\_CSS\\_and\\_JavaScript\\_in\\_ASP\\_N  
ET.aspx](http://highoncoding.com/Articles/777_Minify_CSS_and_JavaScript_in_ASP_NET.aspx)

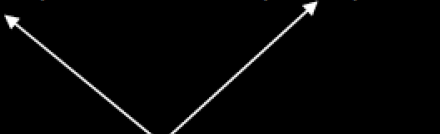
# Minify vs. Obfuscate

An obfuscator minifies but also makes modifications to the program, changing variable names, function names, etc. making it much harder to understand; JSMIn and Dojo are two minifiers

	Original	JSMIn Savings	Dojo Savings
www.amazon.com	204K	31K (15%)	48K (24%)
www.aol.com	44K	4K (10%)	4K (10%)
www.cnn.com	98K	19K (20%)	24K (25%)
www.myspace.com	88K	23K (27%)	24K (28%)
www.wikipedia.org	42K	14K (34%)	16K (38%)
www.youtube.com	34K	8K (22%)	10K (29%)
Average	85K	17K (21%)	21K (25%)

minify - it's safer

not much difference



# Rule 11: Avoid Redirects

- Redirects are used to map users from one URL to another
    - However, redirects insert an extra HTTP round-trip between user and origin server
  - **PHP Redirect**

```
<?
Header( "HTTP/1.1 301 Moved Permanently" );
Header( "Location: http://www.new-url.com" );
?>
```
  - **JSP (Java) Redirect**

```
<%
response.setStatus(301);
response.setHeader( "Location", "http://www.new-url.com/" );
response.setHeader( "Connection", "close" );
%>
```
  - **CGI PERL Redirect**

```
$q = new CGI;
print $q->redirect("http://www.new-url.com/");
```
  - **Redirect Old domain to New domain ([htaccess redirect](#))**
    - Create a .htaccess file with the below code, it will ensure that all your directories and pages of your old domain will get correctly redirected to your new domain.  
The .htaccess file needs to be placed in the root directory of your old website (i.e the same directory where your index file is placed)
- Options +FollowSymLinks  
RewriteEngine on  
RewriteRule (.\*?) http://www.newdomain.com/\$1 [R=301,L]
- REPLACE www.newdomain.com in the above code with your actual domain name.
  - contact every backlinking site to modify their backlink to point to your new website.
  - **Note\*** This .htaccess method of redirection works ONLY on Linux servers having the Apache Mod-Rewrite module enabled.

# Rule 11: Avoid Redirects

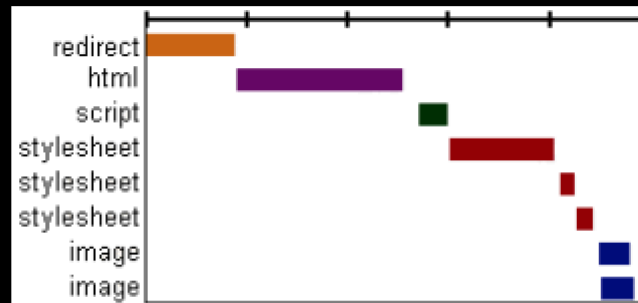
3xx status codes – mostly 301 and 302

HTTP/1.1 301 Moved Permanently

Location: <http://stevesouders.com/newuri>

add Expires headers to cache redirects

worst form of blocking



## Rule 12: Remove Duplicate Scripts

- Hurts performance
  - Extra HTTP requests (IE only)
  - Extra executions
- Is it atypical
  - 2 of the top 10 websites contain duplicate scripts (JS and CSS)

# Rule 13: Configure ETags

- Etags are used by clients and servers to verify that a cached resource is valid
  - To check that the resource (image, script, stylesheet, etc.) in the browser's cache matches the one on the server
  - If there is a match, the server returns a 304

- Unique identifier returned in response

Etag: "c8897e-ae-4165acf0"

Last-Modified: Thu, 07 Oct 2008 20:54:08 GMT

- Used in conditional GET requests

If-None-Match: "c8897e-ae-4165acf0"

If-Modified-Since: Thu, 07 Oct 2008 20:54:08 GMT

- If Etag doesn't match, can't send 304
- Etag format varies across web servers
  - Apache: inode-size-timestamp
  - IIS: FileTimestamp:ChangeNumber
- Use 'em or lose 'em
  - Apache: FileETag none
  - IIS: <http://support.microsoft.com/kb/922703/>
  - <http://stackoverflow.com/questions/477913/how-do-i-remove-etag-headers-from-iis7>

# Rule 14: Make AJAX Cacheable and small

- The URL of an AJAX request is included inside the HTML; as it is not bookmarked or linked to, the requesting page can be cached by the browser
- The AJAX request URL should include a dynamic variable, e.g. dateAndtime, so if the page is changed at the server, the new page will be downloaded
- As long as the AJAX request page has not changed, e.g. an address book would change infrequently, it is best to have the browser cache it
- See <http://blog.httpwatch.com/2009/08/07/ajax-caching-two-important-facts/> for more details. In general, use of these response headers that make AJAX response cacheable:
  - Expires, Last-Modified, Cache-Control
  - Example: stock quote that updates every 10 seconds

```
GET /yab/[...]&r=0.5289571 HTTP/1.1
```

```
Host: us.xxx.mail.yahoo.com
```

```
HTTP/1.1 200 OK
```

```
Date: thu, 12 Apr 2007 19:39:09 GMT
```

```
Cache-Control: private, max-age=0
```

```
Last-Modified: Sat, 31 Mar 2007 01:17:17 GMT
```

```
Content-type: text/xml; charset=utf-8
```

```
Content-Encoding: gzip
```



# Updated Yahoo Best Practices (2011)

Yahoo updated their list of best practices in 2011. See:

<http://developer.yahoo.com/performance/rules.html>

- 1.Flush the buffer early
- 2.Use GET for AJAX requests
- 3.Post-load components
- 4.Preload Components
- 5.Reduce the number of DOM Elements
- 6.Split Components Across Domains
- 7.Minimize the number of iframes
- 8.No 404s
- 9.Reduce cookie size
- 10.Use cookie-free domains for components
- 11.Minimize DOM access
- 12.Develop smart event handlers
- 13.Avoid filters
- 14.Optimize images
- 15.Optimize CSS sprites
- 16.Don't scale images in html
- 17.Make favicon.ico small and cacheable
- 18.Keep components under 25K
- 19.Pack components into a multipart document
- 20.Avoid Empty image src

# Some New Rules (2011)

- **Avoid empty src or href**

You may expect a browser to do nothing when it encounters an empty image src.

However, it is not the case in most browsers. IE makes a request to the directory in which the page is located; Safari, Chrome, Firefox make a request to the actual page itself. This behavior could possibly corrupt user data, waste server computing cycles generating a page that will never be viewed, and in the worst case, cripple your servers by sending a large amount of unexpected traffic.

- **Use GET for AJAX requests**

When using the XMLHttpRequest object, the browser implements POST in two steps: (1) send the headers, and (2) send the data. It is better to use GET instead of POST since GET sends the headers and the data together (unless there are many cookies). IE's maximum URL length is 2 KB, so if you are sending more than this amount of data you may not be able to use GET.

- **Reduce the number of DOM elements**

A complex page means more bytes to download, and it also means slower DOM access in JavaScript. Reduce the number of DOM elements on the page to improve performance.

# Some New Rules (cont' d)

- **Avoid HTTP 404 (Not Found) error**

Making an HTTP request and receiving a 404 (Not Found) error is expensive and degrades the user experience. Some sites have helpful 404 messages (for example, "Did you mean ...?"), which may assist the user, but server resources are still wasted.

- **Reduce cookie size**

HTTP cookies are used for authentication, personalization, and other purposes. Cookie information is exchanged in the HTTP headers between web servers and the browser, so keeping the cookie size small minimizes the impact on response time.

- **Use cookie-free domains**

When the browser requests a static image and sends cookies with the request, the server ignores the cookies. These cookies are unnecessary network traffic. Make sure that static component requests are cookie-free (i.e., use static.mydomain.com to serve static content).

- **Do not scale images in HTML**

Web page designers sometimes set image dimensions by using the width and height attributes of the HTML image element. Avoid doing this since it can result in images being larger than needed. For example, if your page requires image myimg.jpg which has dimensions 240x720 but displays it with dimensions 120x360 using the width and height attributes, then the browser will download an image that is larger than necessary. (This rule conflicts with Responsive design patterns)

# Some New Rules (cont' d)

- **Make favicon small and cacheable**

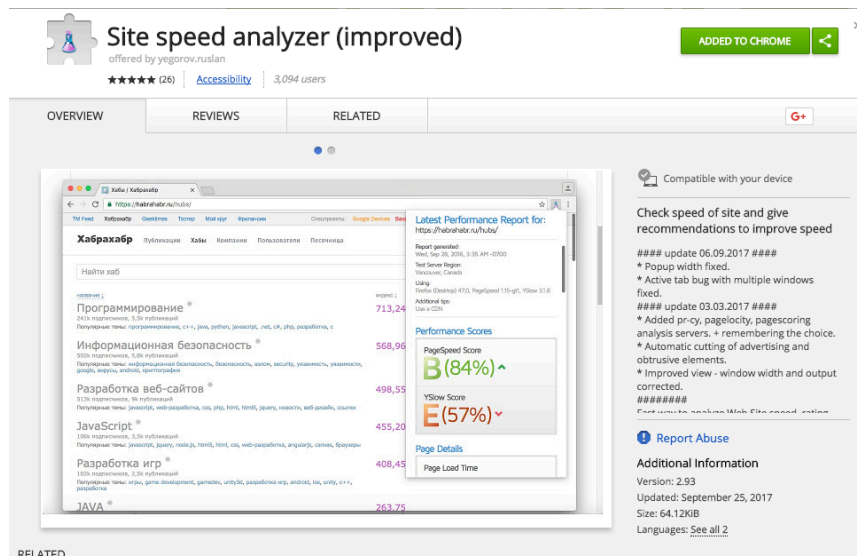
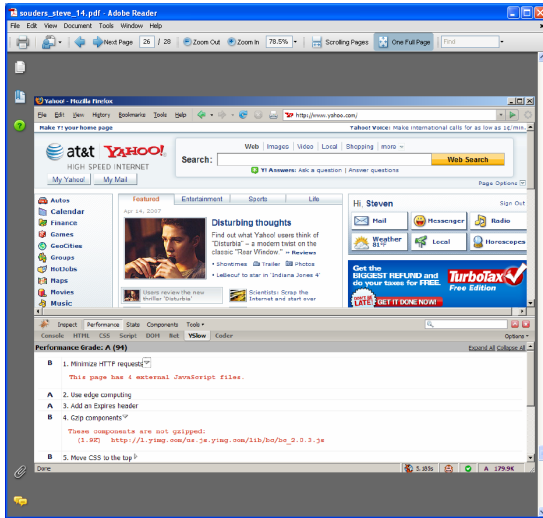
A favicon is an icon associated with a web page; this icon resides in the favicon.ico file in the server's root. Since the browser requests this file, it needs to be present; if it is missing, the browser returns a 404 error (see "Avoid HTTP 404 (Not Found) error" above). Since favicon.ico resides in the server's root, each time the browser requests this file, the cookies for the server's root are sent. Making the favicon small and reducing the cookie size for the server's root cookies improves performance for retrieving the favicon. Making favicon.ico cacheable avoids frequent requests for it. Set expires header to a few months into the future, and limit size to 1K.

# Updated Yahoo Best Practices (2012)

1. Minimize HTTP Requests
2. Use a Content Delivery Network
3. Add an Expires or a Cache-Control Header
4. Gzip Components
5. Put Stylesheets at the Top
6. Put Scripts at the Bottom
7. Avoid CSS Expressions
8. Make JavaScript and CSS External
9. Reduce DNS Lookups
10. Minify JavaScript and CSS
11. Avoid Redirects
12. Remove Duplicate Scripts
13. Configure Etags
14. Make Ajax Cacheable
15. Flush the Buffer Early
16. Use GET for AJAX Requests
17. Post-load Components
18. Preload Components
19. Reduce the Number of DOM Elements
20. Split Components Across Domains
21. Minimize the Number of iframes
22. No 404s
23. Reduce Cookie Size
24. Use Cookie-free Domains for Components
25. Minimize DOM Access
26. Develop Smart Event Handlers
27. Choose <link> over @import
28. Avoid Filters
29. Optimize Images
30. Optimize CSS Sprites
31. Don't Scale Images in HTML
32. Make favicon.ico Small and Cacheable
33. Keep Components under 25K
34. Pack Components into a Multipart Document
35. Avoid Empty Image src

Only 23 rules  
can be tested  
with YSlow (no  
longer available)

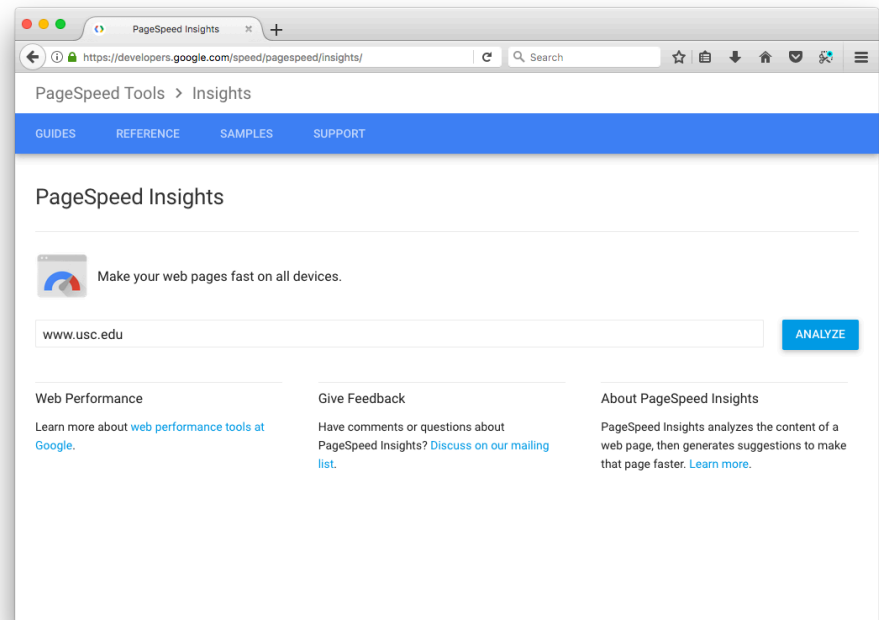
# Demo Yslow



- <http://developer.yahoo.com/yslow>
- Grades web pages for each rule described earlier
- Firefox add-on integrated with Firebug
- Minimize HTTP requests
- Add an expires header
- Gzip components
- Tests 23 rules (Yslow V2) or original 14 rules (Classic V1)
- Original version stopped working with Firefox 46.
- Extensions available for Chrome (Site speed analyzer)

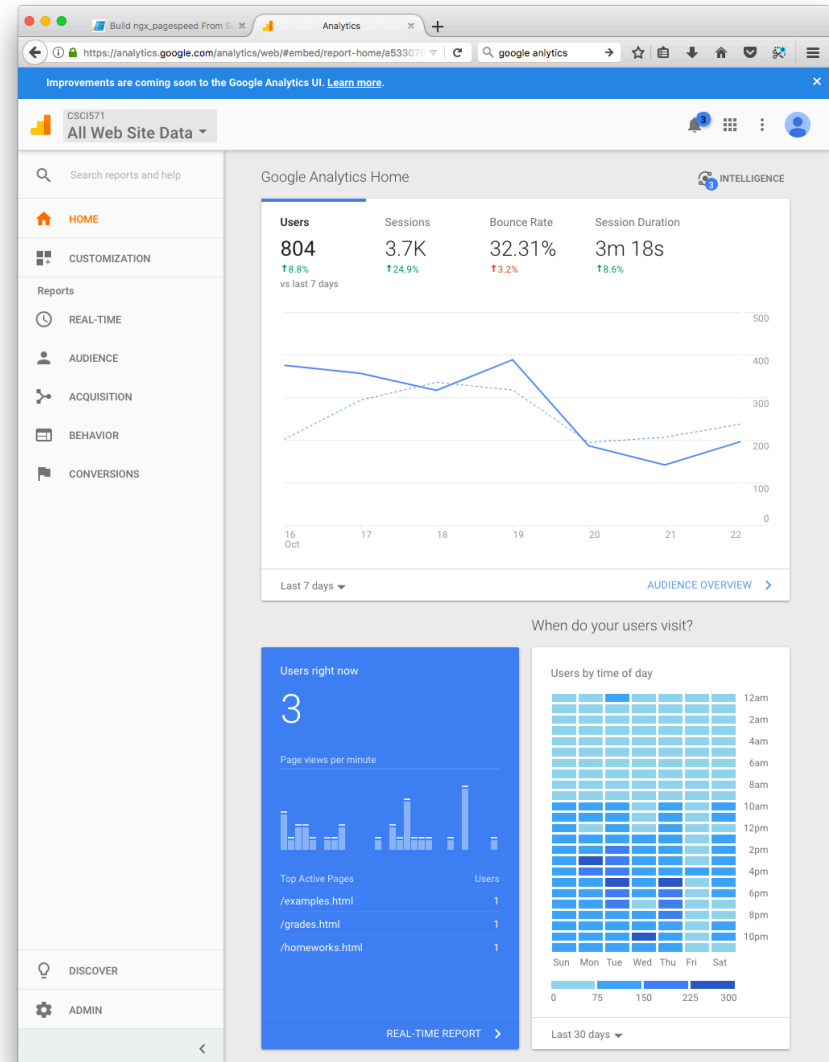
# Demonstrate Google's PageSpeed Insights

- Google offers a similar service to Yahoo's YSlow called PageSpeed Insights. See:  
<http://developers.google.com/speed/pagespeed/>
- Enter the URL to test at:  
<https://developers.google.com/speed/pagespeed/insights/>
- For Chrome info go to:  
[https://developers.google.com/speed/pagespeed/insights\\_extensions](https://developers.google.com/speed/pagespeed/insights_extensions)
- Page Speed is also available in Google Analytics
- PageSpeed can integrate with Apache and Nginx web server to automatically optimize your site



# Demonstrate Google Analytics

- Google offers PageSpeed Insights inside Google Analytics.  
<https://analytics.google.com>
- Select Left Navigation – Behavior – Site Speed – Speed Suggestions – click “x total” under “PageSpeed Suggestions” columns





# References

**book:** <http://www.oreilly.com/catalog/9780596514211/>

**examples:** <http://stevesouders.com/examples/>

**image maps:** <http://www.w3.org/TR/html401/struct/objects.html#h-13.6>

**CSS sprites:** <http://alistapart.com/articles/sprites>

**inline images:** <http://tools.ietf.org/html/rfc2397>

**jsmin:** <http://crockford.com/javascript/jsmin>

**dojo compressor:** <http://dojotoolkit.org/docs/shrinksafe>

**HTTP status codes:** <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>

**Fasterfox:** <http://fasterfox.mozdev.org/>

**LiveHTTPHeaders:** <http://livehttpheaders.mozdev.org/>

**YUIBlog:** <http://yuiblog.com/blog/2006/11/28/performance-research-part-1/>

<http://yuiblog.com/blog/2007/01/04/performance-research-part-2/>

<http://yuiblog.com/blog/2007/03/01/performance-research-part-3/>

<http://yuiblog.com/blog/2007/04/11/performance-research-part-4/>

**YDN:** [http://developer.yahoo.net/blog/archives/2007/03/high\\_performanc.html](http://developer.yahoo.net/blog/archives/2007/03/high_performanc.html)

[http://developer.yahoo.net/blog/archives/2007/04/rule\\_1\\_make](http://developer.yahoo.net/blog/archives/2007/04/rule_1_make)

# Efficient Loading of JavaScript

- Loading JavaScript takes time, but one possible approach is to split the initial payload.
- Problems:
  - - JavaScript execution in the browser is single threaded, so while you are loading the modules in the background, the rest of your app becomes non-responsive to user actions while the modules load.
  - - It is difficult to decide when, and in what order, to load the modules.
- The Google Solution
  - write each module into a separate script tag and hide the code inside a comment block (`/* */`). When the resource first loads, none of the code is parsed since it is commented out. To load a module, find the DOM element for the corresponding script tag, strip out the comment block, and `eval()` the code.
  - Implications: Once the code arrives, modules are `eval`'ed on an as-needed basis, without the delay of actually downloading the script. The `eval` does take time, but this is minimal and is tied to the user's actions, so it makes sense from the user's perspective.
- Here is an example of how Google loads its JavaScript (see <http://googlecode.blogspot.com/2009/09/gmail-for-mobile-html5-series-reducing.html>)

```
<html>...
<script id="lazy">
// Make sure you strip out (or replace) comment blocks in your JavaScript first.
/*  JavaScript of lazy module    */
</script>
<script>
function stripOutCommentBlock(code) {    return code.replace(/^[\s\xA0]+\//\/*|\/[\s\xA0]+$/g, "");
    }
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>
```