A Holistic view of Improving Website Performance

Rene Churchill - Astute Computing
rene@astutecomputing.com

What does "*Performance*" mean?

1. Client side - How fast does the site render in MY personal browser.
2. Server side - How many clients can I support with my server?

vs.

Website performance can be viewed from many different perspectives.

This talk will cover some of the things you can do as a webmaster to improve how quickly your site renders on your users machines.

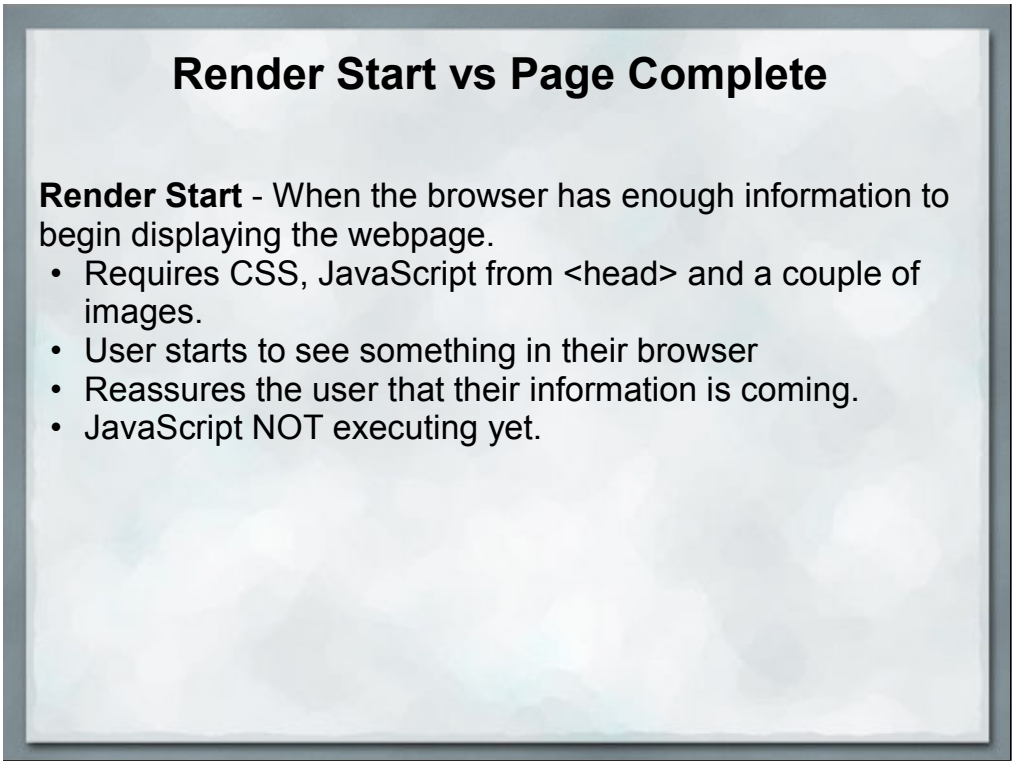
This means only making changes on the server since you do not have control over the end-users browser settings.

Why should I give a gnats arse?

Potentially a MAJOR impact on site revenue:

"Experiments at Amazon.com showed that every 100-ms increase in the page load time decreased sales by 1 percent, while similar work at Google revealed that a 500-ms increase in the search results display time reduced revenue by 20 percent."

IEEE Computer magazine, Kohavi and Longbotham - 2007
http://exp-platform.com/Documents/IEEEComputer2007OnlineExperiments.pdf



Render start vs Page Complete, which is more important depends on your website.

A table of statistics would care about render start because the user can start reviewing the data before the page is done.

A e-commerce site may care more for page complete because that is when the "Buy This" buttons appear.

Render Start vs Page Complete

Page Complete - When everything visible has been loaded.

- User has all the information they requested.
- Buttons/forms active and clickable
- This is when the **onLoad()** event fires.
- JavaScript may still be loading elements in the background.

At this point people can start interacting with the webpage (in theory)

Render Start vs Page Complete

Page Really Complete - When *everything* has been loaded.

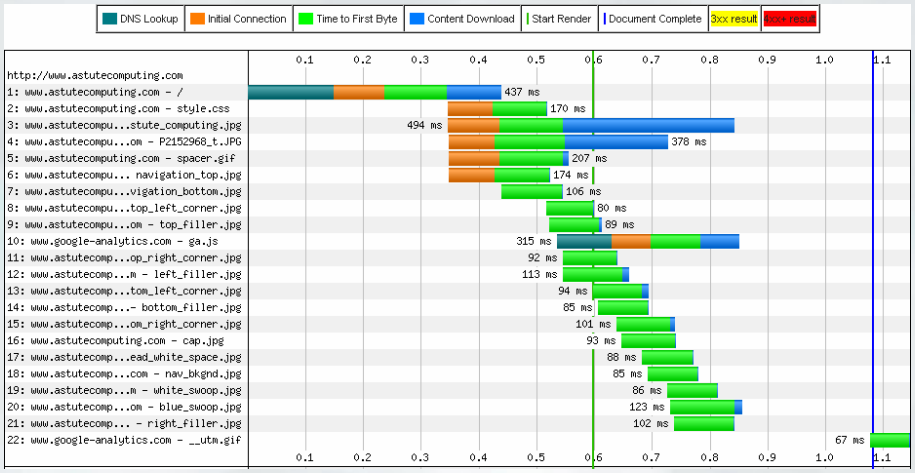
- All of the widgets have loaded
- All of the initial AJAX calls have been made
- Facebook, Google Analytics, AddThis, Google AdWords, etc., etc., etc.

Some websites get to be very frustrating because the browser locks up when **onLoad()** fires because SO much JavaScript is launched.

Steps to resolving an HTML request

1. DNS lookup
2. Initialize Connection
3. Load & Compile PHP
4. Execute PHP & Database Queries
5. Return resulting HTML
6. Fetch CSS, JavaScript and Images
7. Render page
8. Execute JavaScript
9. Profit!

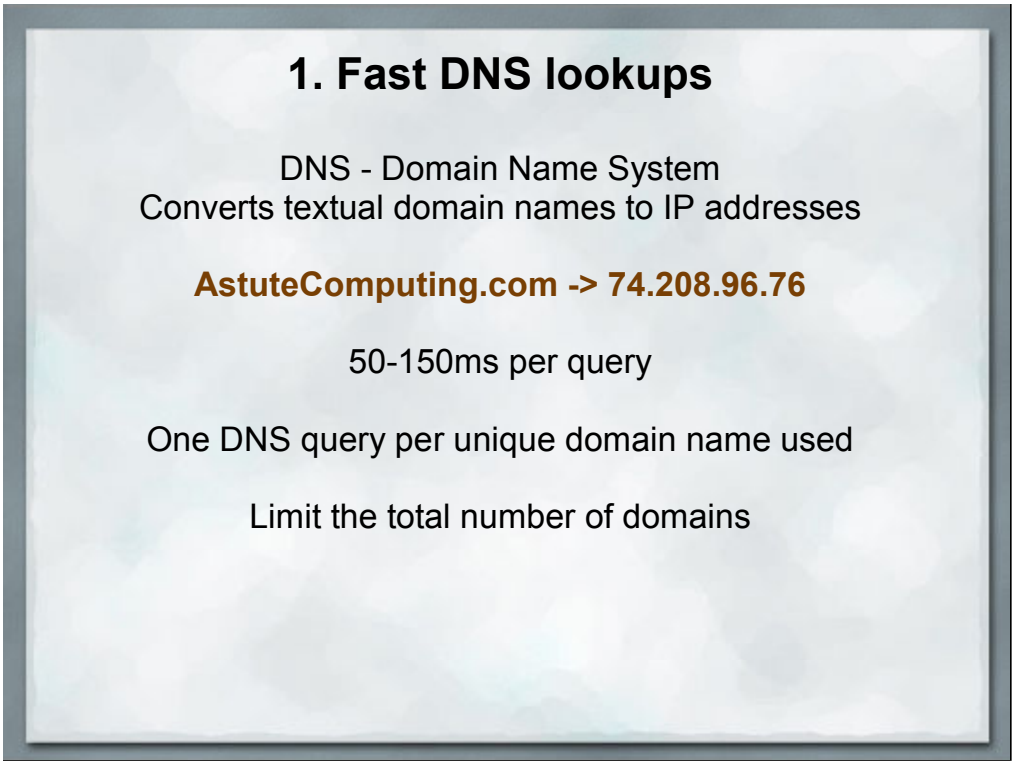
Waterfall view of Webpage Performance



http://webpagetest.org/

This is a waterfall graph from the site WebPageTest.org, which is a huge help in understanding the entirety of how long it takes your webpage to load.

We'll be covering the information shown on this graph in detail throughout this presentation.



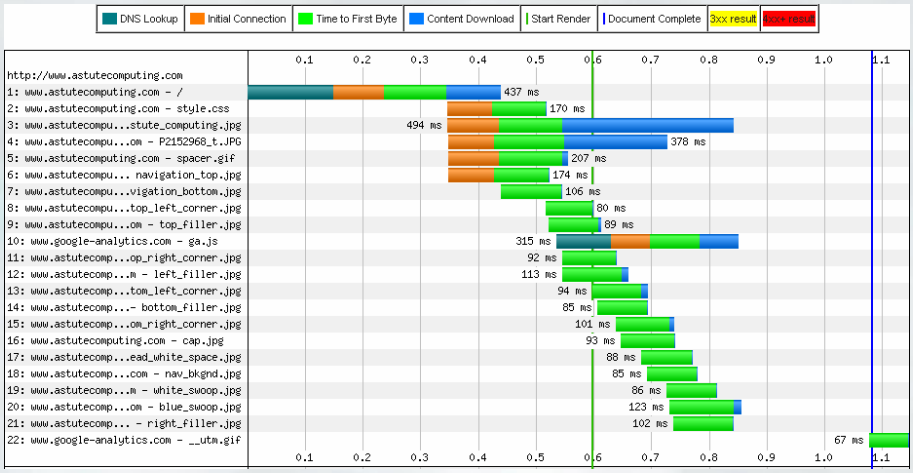
Not much you can really do to speed up a DNS server.

They're amazingly fast already, almost 100% of the time involved in the query is network lag between your browser and the DNS servers.

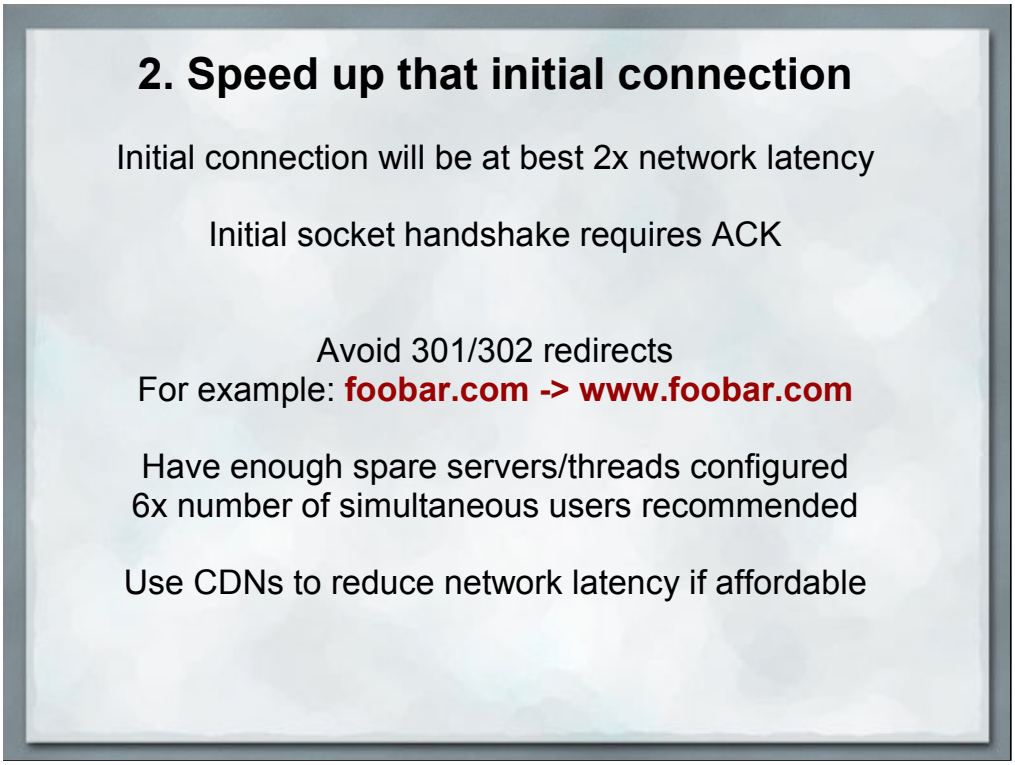
Keep in mind that every widget that you add to your webpage adds at least one more domain name. AddThis.com, Facebook like buttons, Google Analytics, etc., etc., etc.

DNS queries are cached by the browser but the cache doesn't help on that all-important first pageview.

Waterfall view of Webpage Performance



http://webpagetest.org/



Socket communication requires an acknowledgement before the connection is established, so a minimum of 2x the network latency.

Not all that much that you can do to speed this up. Content Delivery Networks can help reduce the network lag, but not many of us are in a financial position to take advantage of this.

Avoid redirects, especially on that all-important home page. Yes, it makes handling your cookies easier, but it can easily add a half-second to your first and most important pageview. Redirects add another DNS lookup and another connection request if the domain name changes.
CDN - Content Delivery Network. A collection of servers placed around the country. DNS tricks are employed so that the server closed to the user is picked.

Better User Experience

Some things you can do to improve your users experience:

- Register variations on your domain name - plurals, with and without dashes, etc. Do not redirect the user on the first page.
- Analytic programs can handle multiple domain names
- Use wildcard cookies to avoid domain name redirects:
  `setcookie($name, $value, $time, '/', '.example.com');`
- Put your preferred domain name into the Canonical link to avoid Google penalties for duplicate content:
  `<link rel="canonical" href="http://www.foo.com/bar.php" />`

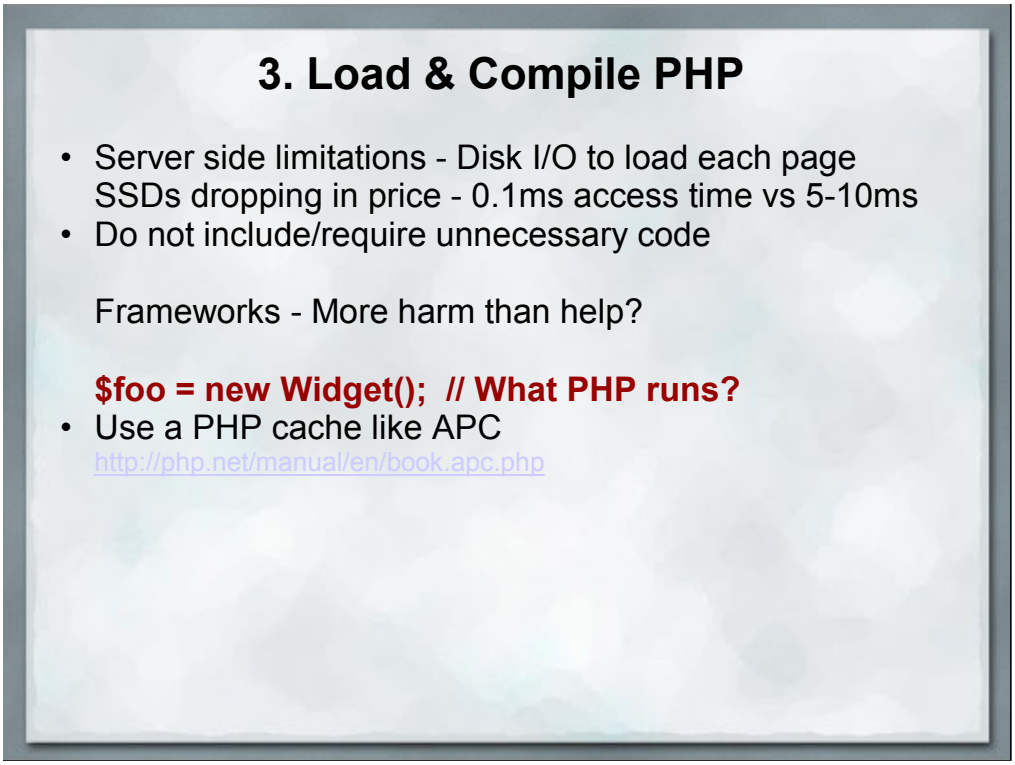
Variations on the domain name doesn't help performance wise, but it's a nice touch and avoids the misspelling squatters.

Redirects & Spare Servers/Threads

IE8 allows up to 6 simultaneous connections per domain name

Here you can see the cost of redirecting to a specific version of the domain name. It includes another DNS lookup and a second socket connection.

Also notice that IE8 allows up to 6 simultanious connections, each taking a thread/server.



The controversial item on this page is the question about frameworks.

When that constructor fires, how much is actually being included/compiled/run? How do you know?

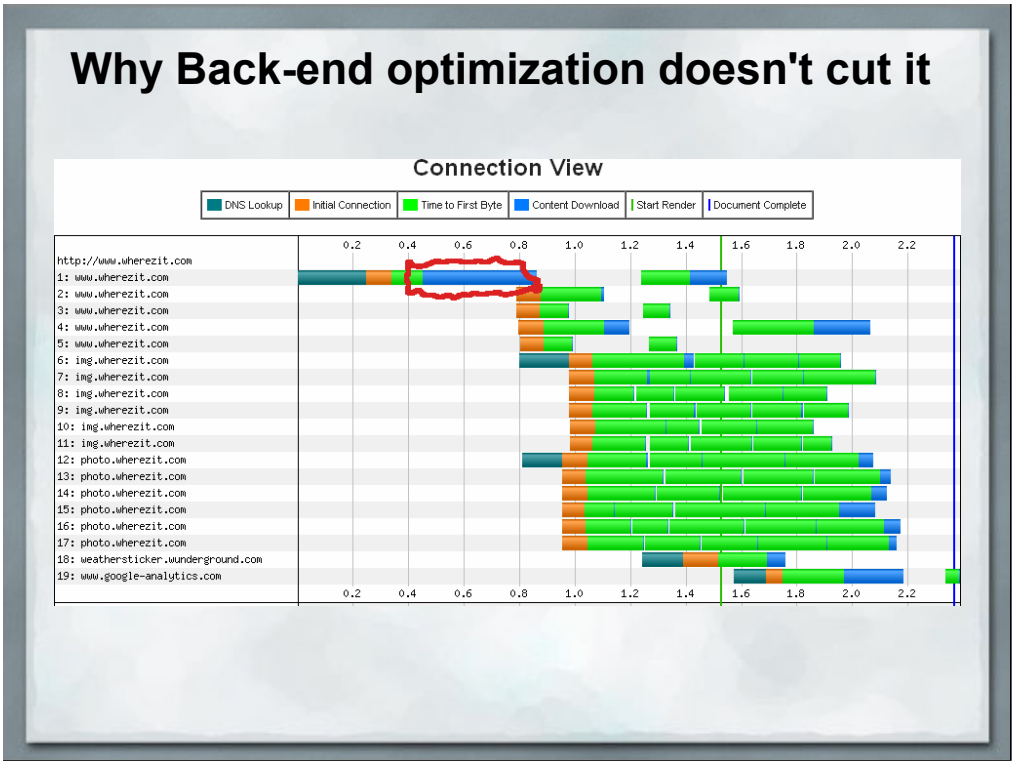
4. Execute PHP & Database Queries

Steve Souders: *"For years when developers started focusing on the performance of their websites, they would start on the back end, optimizing C++ code or database queries. Then we discovered that about 10% or 20% of the overall page load time was spent on the back end. So if you cut that in half, you only improve things 5%, maybe 10%. In many cases, you can reduce the back end time to zero and most users won't notice."*

http://radar.oreilly.com/2011/06/steve-souders-optimization-mobile-http-archive.html

Since we're a programming group, most of what we talk about is here in this section.

And we're pretty good at it, to the point where it's not a major portion of the problem.



Pardon the sloppy drawing, it's hard to draw nice curves with a trackball

Even if I eliminated the back-end PHP time to generate this page, the vast majority of the time that the user is kept waiting is NOT involved in PHP.

Other options for load/execute PHP

- memcache - http://memcached.org/
- Varnish Cache Server - http://www.varnish-cache.org/
- MySQL query caching
- nginX - http://nginx.org/ (less memory usage)
- php-fpm - http://php-fpm.org/ (less memory usage)
- Many, many others

What really needs to be Dynamic?

- Use PHP (or any other language) and cron to generate static HTML pages on a schedule
- Use AJAX to pull in user specific content after page has loaded.

User specific content could be the contents of their shopping cart, their username, etc. All of the little personalization touches could be done via AJAX.

5. Return the resulting HTML

Enable gzip in Apache - Internet vs Intranet - gzip helps most in low-bandwidth environments

```
/etc/httpd/conf/httpd.conf
 LoadModule deflate_module modules/mod_deflate.so
 ...
 AddOutputFilterByType DEFLATE text/html text/plain text/xml
 ...
 <VirtualHost ...>
    SetOutputFilter DEFLATE
 </VirtualHost>
```

Balance the bandwidth reduction via gzip against the CPU time used to compress things. If the server is heavily loaded, the CPU time may be more valuable than the bandwidth.

5. Return the resulting HTML

Every modern browser supports partial page rendering so:

- Flush output buffer early - flush(), after </head>, after the fold, after writing a major div/table of the page, etc.

```
<html>
<head>
  ... stuff ...
</head>
<body class="body_foobar">
<?php
   flush();
?>
```

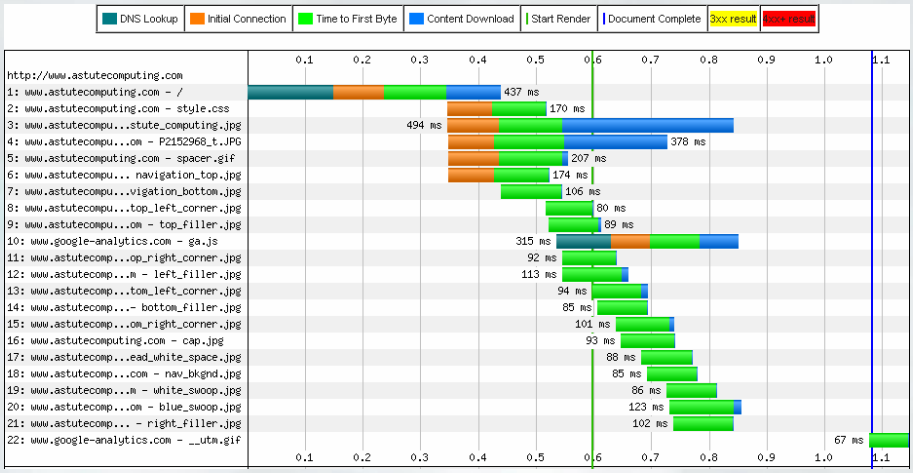
Note, flush() conflicts with gzip.

Because gzip encodes data in chunks and is implemented as an output filter, flush() cannot arbitrarily push partial blocks out the pipe.

So if the HTML that you generate is fairly small, skip gzip and see if flush() makes more of a different in pageload time.

Personally I get better results from gzip/deflate than flush()

Example of Flush() helping



http://webpagetest.org/

Stylesheets reduce HTML size

- Use CSS to avoid repeating style information

```
<table cellpadding=5 cellspacing=0 border=1
bordercolor="#ff0000" bgcolor="#009900"
width="90%">
```

vs.

```
<table class="gross">
```

- Shorten CSS tags

```
<p class="a23r7">
```

vs.

```
<p class="paragraph_style_alpha23_rev7">
```

Why nested tables are bad:

Example stolen from a great presentation by Bill Merikallio & Adam Pratt: http://www.hotdesign.com/seybold/15inteadofthis.html

A pleasant enough looking table

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.	Sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper.
Suscipit lobortis nisl ut aliquip ex ea commodo consequat.	Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis. • At vero eros et accumsan et iusto odio dignissim qui blandit • Praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.
Qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.	Diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. • Epsum factorial non deposit quid pro quo hic escorol. • Olypian quantels et gomilla congolium sic ad nauseum. Souvlaki ignitus carborundum e pluribus unum. Defacto lingo est igpay atinlay.

Take a look beneath the surface

This could really be done much more simply.	Guess how much markup there is in this little table? 13.7k. There are 17 rows and 9 columns in this thing. And did I mention all of the spacer GIFs?
There are way too many table cells and spacers in here.	And all of the dotted borders are done with a `background` attribute on table cells, which won't validate. • A nested table? What for? • To make a bulleted list? You're kidding, right?
This could all be done with 8 table cells and 4 CSS rules.	Seriously. 8 cells and 4 css rules, that's all it takes. • Oh no, another table masquerading as a bulleted list. • Just mark up your bulleted lists as bulleted lists and let CSS do the rest.

6. Fetch CSS, JavaScript, Images

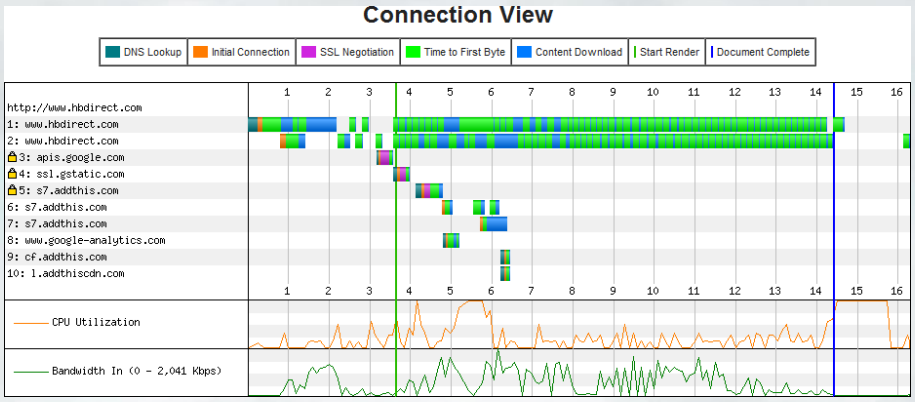
This is all the "extra" stuff that your page needs.

This is where your largest speed gains will be found.

Everything we've discussed so far is minor. The major gains are right here.

All of the extra stuff listed on your page. Any external reference that you make in a page gets pulled from the servers during this step.

Image Overload



Test done with IE 7, 2 connections allowed

Increase Parallel Downloads

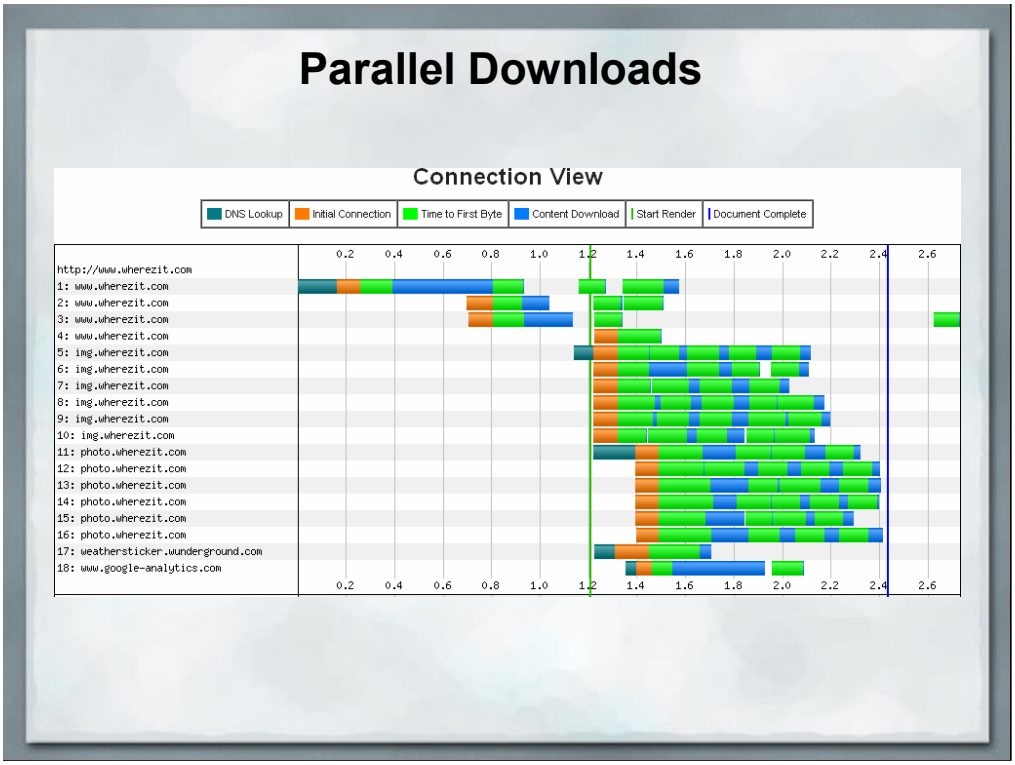
Older browsers restricted to 2 connections / hostname, 6 total

Newer browsers allow 4 connections per hostname,
IE 8 & 9 allow 6.

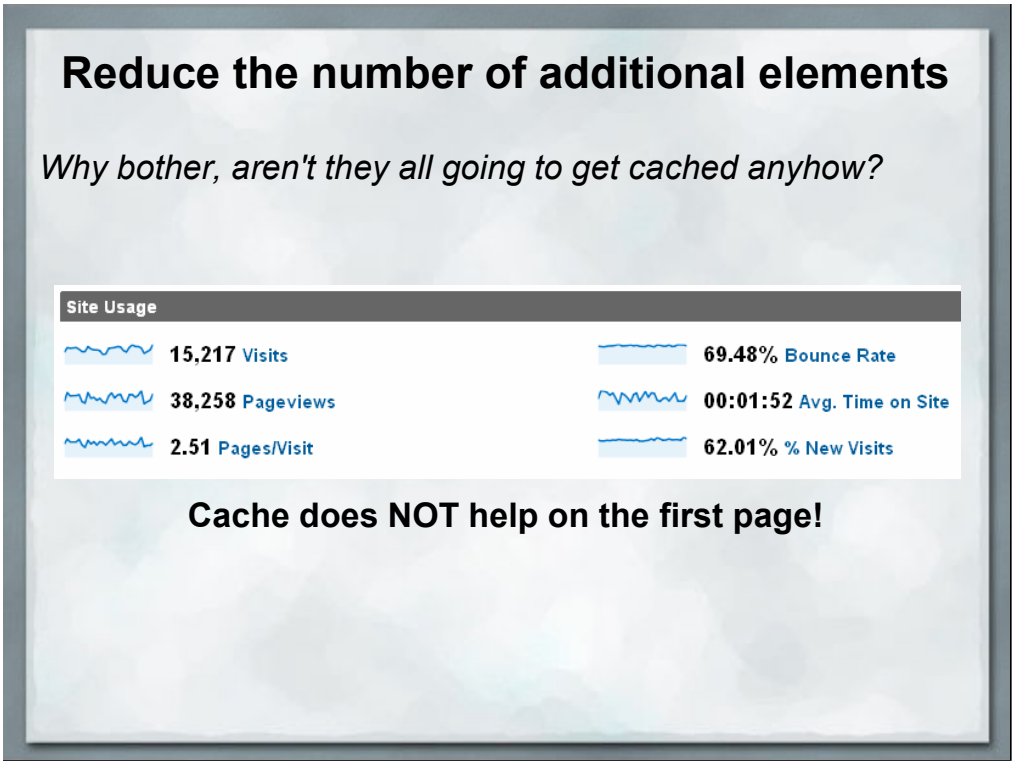
Use multiple domain names:

- photo.example.com
- images.example.com
- etc.

Note that using multiple domain names does increase the DNS resolution time, but it's a tradeoff between the number of downloads that you can simultaniously get rolling.



Reusing the image here.



The VAST majority of users to your website are NOT going to have anything in their cache for that all-important first-page.

Google Analytics released a newsletter recently showing that the average bounce rate over all the sites they analyze (that opted into sharing data) was 47%.

Reduce the number of additional elements

- Combine CSS into a single file
- Combine JavaScript into a single file
- Minify to remove white space and comments

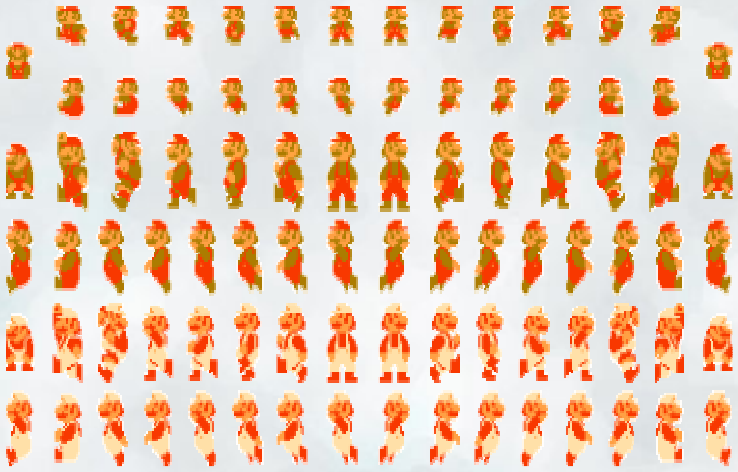
Note that combining all of your javascript into a single file means ALL of your JavaScript, even those libraries that you're using from somebody else.

This can cause conflicts when you're using something like GoogleMaps and they change/improve their libraries.

Also, this can cause havoc when many people are changing the website because you're all making changes to the same file. So save this step until the end, just before the site is pushed live.

CSS Sprites

Blast from the past.
Sprites used heavily in old 8bit video games.



CSS Sprites

Many images can be combined into a single larger image.

- Menu tabs
- Rollover images
- Ratings (stars, dots, whatever)
- Thumbs up
- Thumbs down
- Rounded corners
- Buttons
- etc.

This one larger image results in a single HTTP request rather than a horde of them.

Since bandwidth is less of an issue these days, the larger image size is less of a problem than the multiple requests.

CSS Sprites

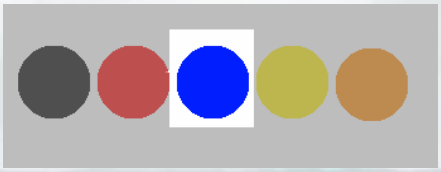
CSS cannot edit/crop images directly

However, background images CAN be shifted.

So...

```
<img class="blue_circle" src="transparent.gif" height=30 width=30>

<style>
img.blue_circle {
    background: url('/images/circles.png') 0px -90px no-repeat;
}
</style>
```



This one larger image results in a single HTTP request rather than a horde of them.

Since bandwidth is less of an issue these days, the larger image size is less of a problem than the multiple requests.

CSS Sprites

This trick also works well with other effects:

```
<a id="buyme" href="add2cart.php">....</a>

<style>
a#buyme {
   background: url('sprite.png') 0px 0px no-repeat;
}
a#buyme:hover {
   background: url('sprite.png') 0px -50px no-repeat;
}
a#buyme:visited {
   background: url('sprite.png') 0px -100px no-repeat;
}
```

CSS Sprites

Generating/Updating sprite image is a pain

A Solution: HTML Table & Screenshot

Borders help to show boundaries

Obviously, creating these sprite images can be quite time-consuming, so this step is often best left until the site is almost ready to go live, after the customer has signed off on the design look & feel.

Hint: To make creating these a little easier, create an HTML table with all of the image in it and then do a screenshot.

If you do the table trick try adding borders, it'll make later edits much easier and you can just crop out the border when you are positioning the sprite.

Data URL - Encode images into your HTML

```
<img src="/images/larry.gif" alt="Larry">
```

Becomes:

```
<img src="
ywAAAAAMAAwAAAC8IyPqcvt3wCcDkiLc7C0qwyGHhSWpjQu5yqmCYsa
py
uvUUlvONmOZtfzgFzByTB10QgxOR0TqBQejhRNzOfkVJ+5YiUqrXF5Y5l
Kh/DeuNcP5yLWGsEbtLiOSpa/TPg7JpJHxyendzWTBfX0cxOnKPjgBzi4
diinWGdkF8kjdfnycQZXZeYGejmJlZeGl9i2icVqaNVailT6F5iJ90m6m
vuTS4OK05M0vDk0Q4XUtwvKOzrcd3iq9uisF81M1OlcR7lEewwcLp7tuN
NkM3uNna3F2JQFo97Vriy/Xl4/f1cf5VWzXyym7PHhhx4dbgYKAAA7">
```

http://tools.ietf.org/html/rfc2397

Data URL - Encode images into your HTML

Pros:

- No additional HTTP request required
- Bandwidth usually smaller problem than network latency

Cons:

- Images cannot be cached
- Not supported in IE 6 & 7 (Workaround: Mime HTML: http://www.phpied.com/mhtml-when-you-need-data-uris-in-ie7-and-under/)
- IE8 requires encoded data to be less than 32kb

More ways to reduce external elements

Don't use images in the first place.
Text nav buttons with the same CSS background image.

CSS with background images and text overlay to make menus

Lots of JavaScript dropdown menus scripts available that make textual menus.

7. Render HTML

- CSS in <head>
- JavaScript at the bottom of the page
- Defer loading images below the fold so JavaScript event **onLoad()** can fire.

  http://developer.yahoo.com/yui/imageloader/

JavaScript blocks other downloads, so any external script references in your header will block images from downloading.

document.write() calls cannot be moved to the bottom of the page, but they can be deferred.

8. Execute JavaScript

We have no control over which JavaScript engine is used.

Keep the gee-whiz factor under control.

Do frameworks really help? (Node.js, jQuery, etc.)

The end user gets to control which JavaScript engine is used by their choice of browser. We could restrict the site to a specific browser, but that's just slitting our own throats.

Keep the "Gee-Whiz" factor under control. Yes, that subtly changing background gradient may be really cool, but does it actually help move product?

Again I'm bringing up frameworks. I've seen major chunks (like 250k worth) of JavaScript included in order to use one or two 20-line JavaScript utility functions. There's no reason to abuse bandwidth like that.

Apache settings - speed tweaks

/etc/httpd/conf/httpd.conf
KeepAlive On
MaxKeepAliveRequests 5000
KeepAliveTimeout 15

Keeping around lots of spare threads/server processes for quick connections.

Don't resolve DNS, let post-processing analytic tool handle it

HostnameLookups Off

KeepAlive - tradeoff between server resources and browser speed.

Recommend a minimum of 5-15 seconds, a maximum of a little more than the average time spent on a page for your site, but no more than 5 minutes.

On a high volume site, you may not be able to afford the server resources to keep that many connections alive.

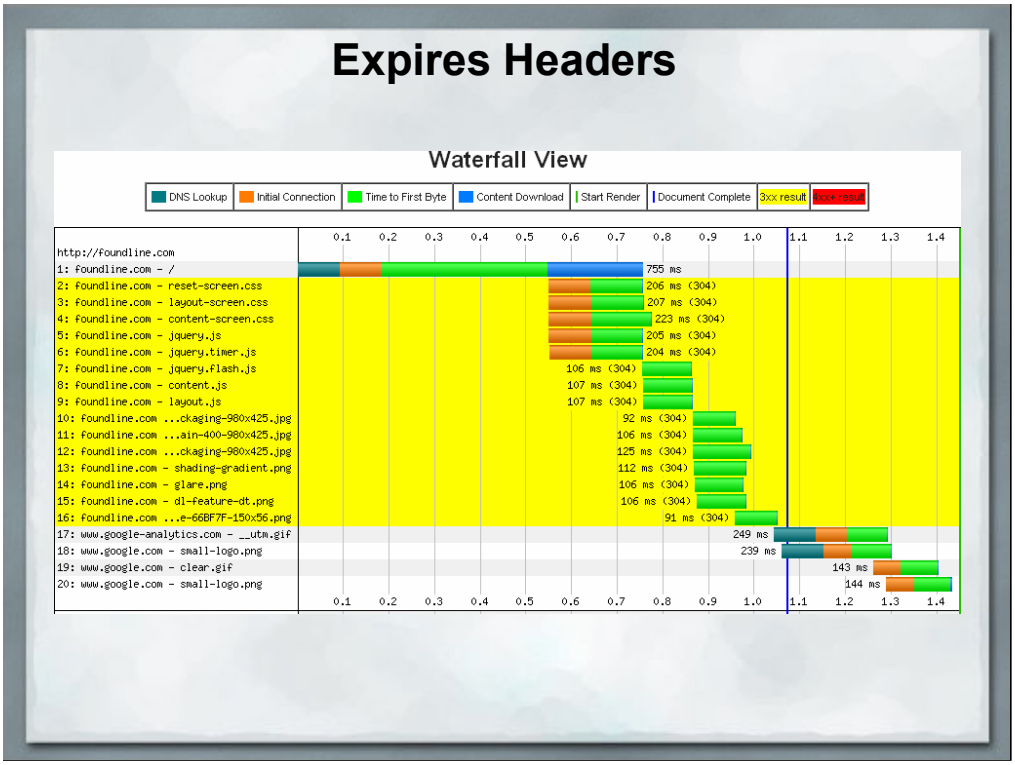
Expires Headers

Eliminates HTTP HEAD requests on subsequent pages to check modification times.

```
/etc/httpd/conf/httpd.conf
 LoadModule expires_module modules/mod_expires.so
 ...
 <VirtualHost ...>
   ExpiresActive On
   ExpiresByType image/* "access plus 4 weeks"
   ExpiresByType text/css "access plus 4 weeks"
   ExpiresByType application/x-javascript A2592000
 </VirtualHost>
```

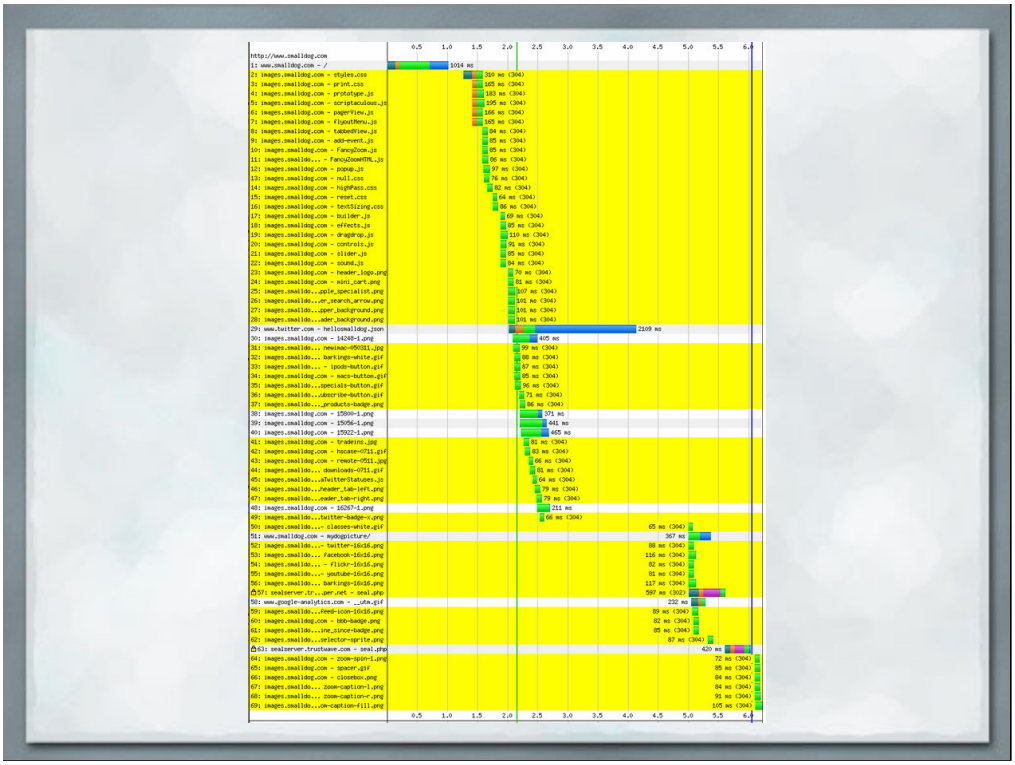
The HEAD requests are done on subsequent page loads to see if a page/image/file on the server has changed. Setting the Expires header can eliminate these checks, speeding up 2nd page load times.

Expires Headers



I'm not picking on FoundLine in particular here, their page reload time is quite good, but they were the first site I could find that wasn't using the Expires headers.

A 304 result means "Not Modified" in response to a HEAD query.



I know this is pretty much unreadable, but here is a more extreme example, smalldog.com.

Lots of graphics all of which haven't changed. Only the white lines are changed elements.

Additional Reading

http://developer.yahoo.com/performance/rules.html
http://developer.yahoo.com/yslow/

Additional Tools

http://webpagetest.org
http://pagespeed.googlelabs.com/
http://spritegen.website-performance.org/ - CSS Sprite generator

This presentation published at:

https://docs.google.com/present/view?id=ddd3c4t5_16hh2k9wgj