

Deep Dive into Browser Performance

Ilia Alshanetsky

@iliaa

Me, Myself and I

PHP Core Developer

Author of
Guide to PHP Security

CIO at Centah Inc.



Why Browser Performance Matters?

Browser rendering 3.342s

Back-end
Processing
0.128s

PHPConference.com - Total Page Load - 3.47s

Browser rendering 1.347s

Back-end
Processing
0.103s

PHP.net - Total Page Load - 1.45s

Browser rendering 1.43s

Back-end
Processing
0.058.6

Github - Total Page Load - 1.43s

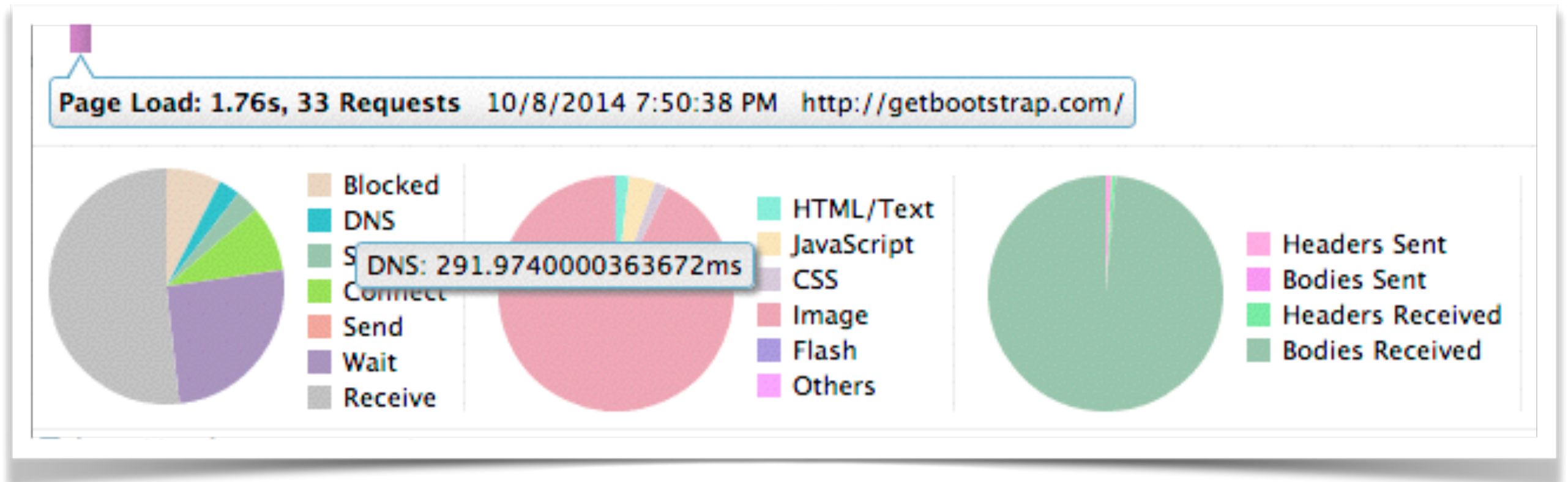
What Takes All This Time?

1. **DNS**
2. **HTTP + SSL Negotiation**
3. **JavaScript Processing**
4. **CSS Rendering**
5. **Image Rendering**
6. **DOM Rendering**



by James Balderson

DNS



DNS may take up-to
20% of 1st page load!

DNS Prefetching

Instruct the browser to pre-resolve (sub)domain from which you intend to retrieve resources.

```
<link rel="dns-prefetch" href="//mydomain.tld" />
```

Supported By:

Firefox 3.5+

Chrome

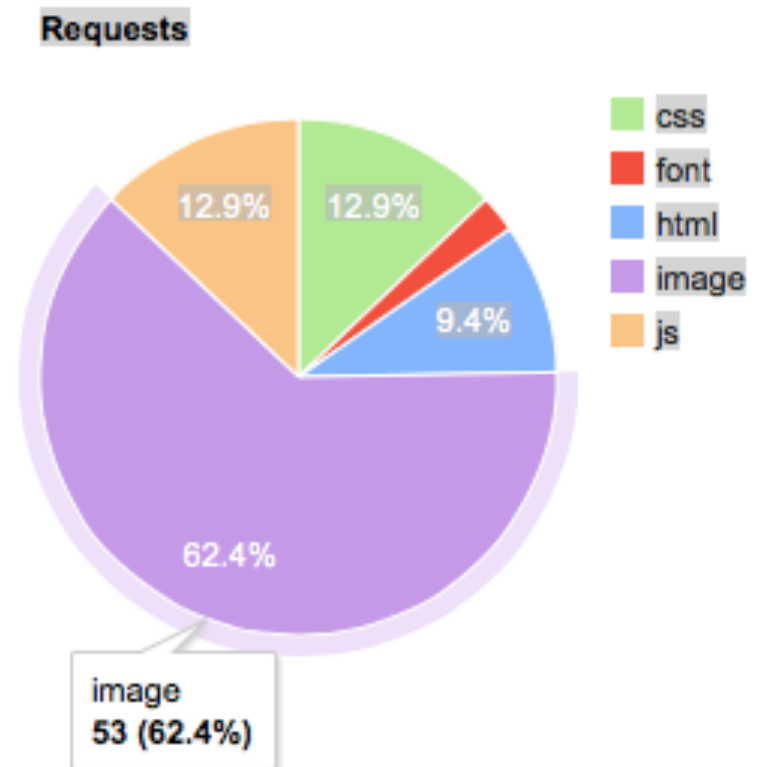
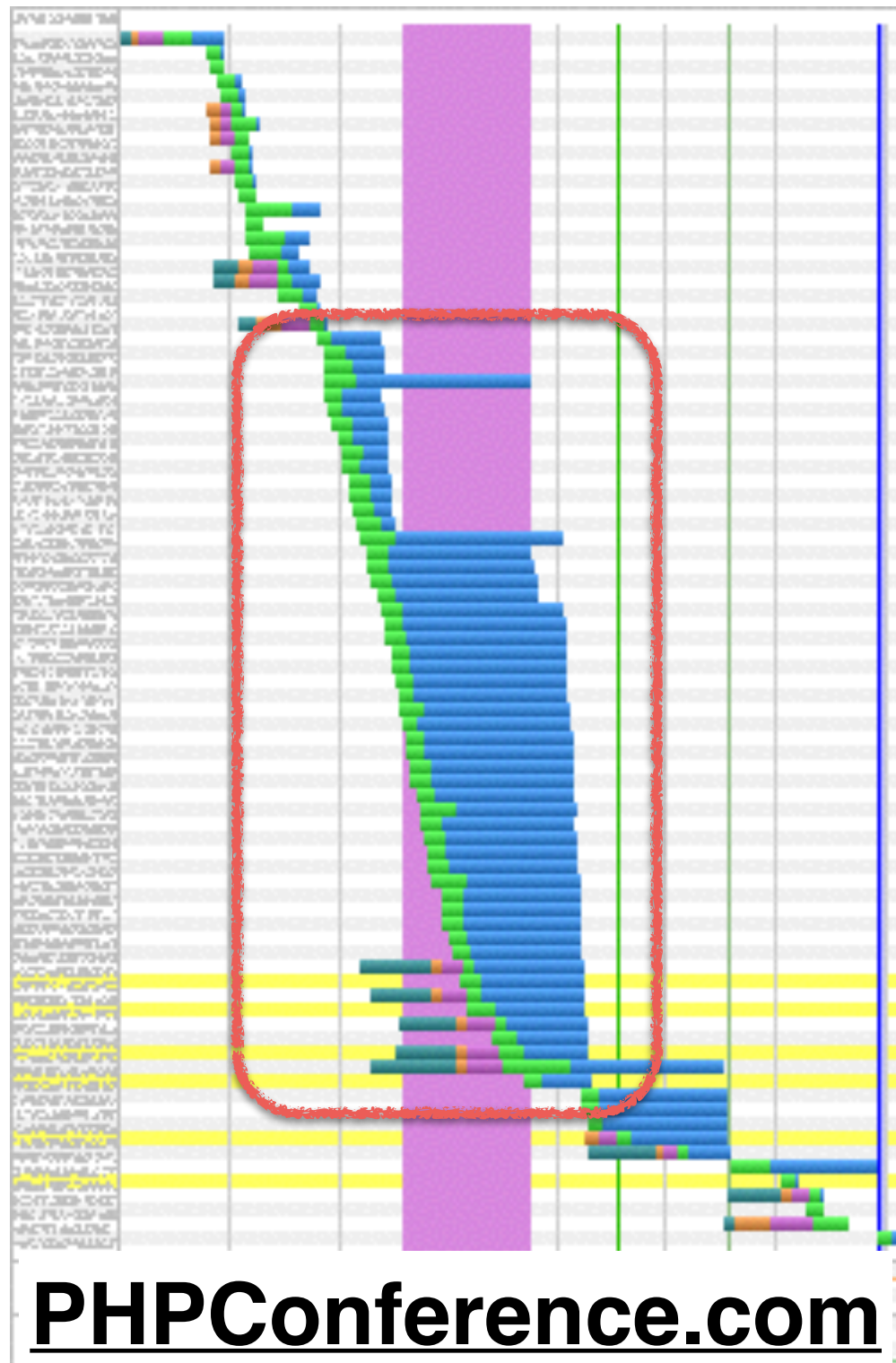
Safari 5+

IE 9+

DNS Based Optimizations

1. Use Embedded images via data:image
2. Defer loading of external resources
3. Avoid multi-domain CDNs
4. Sub-domains still need to be resolved

Use Sprites!



40 images = 1 Request

17.5% reduction in size (74kb saving) !

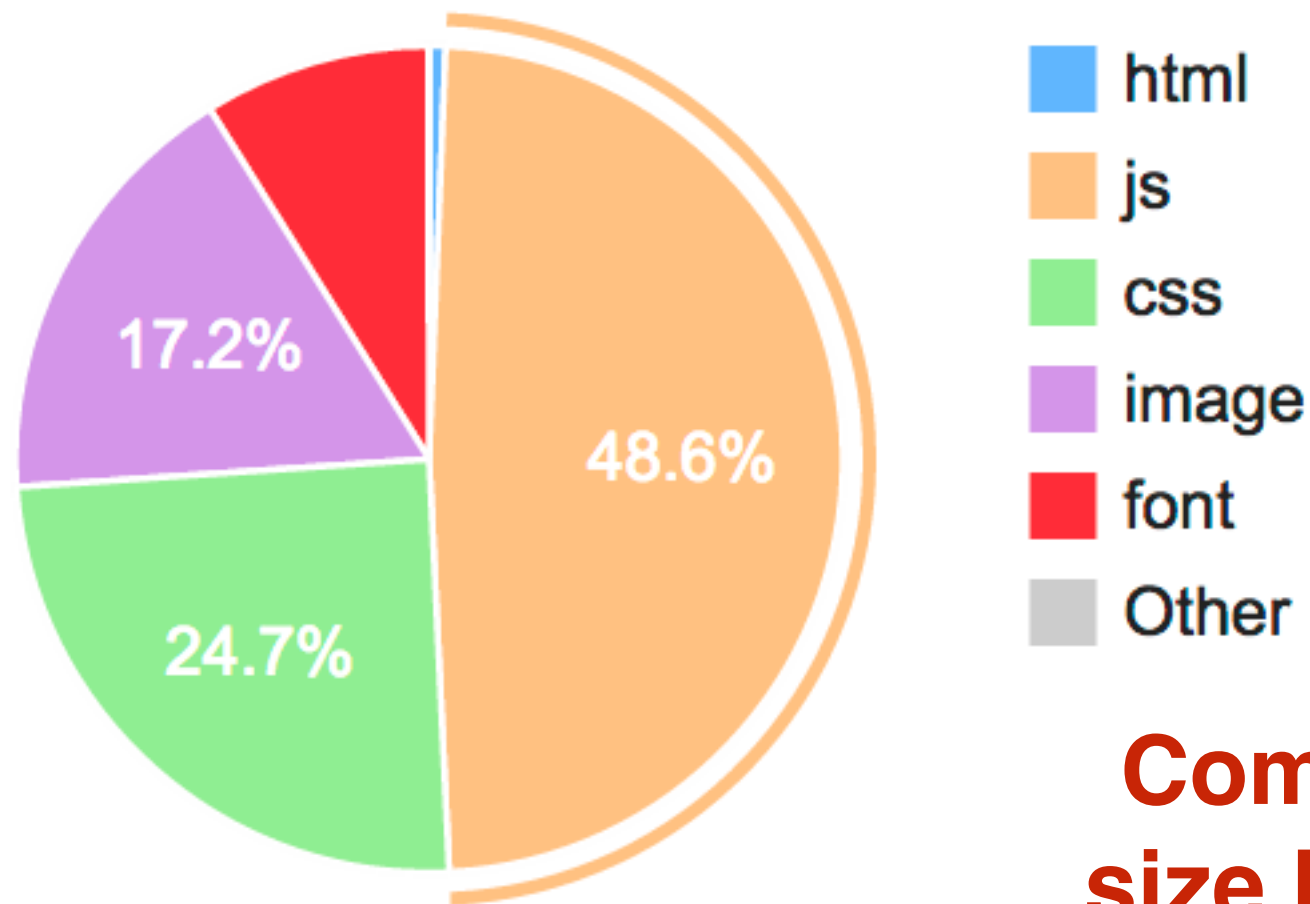
<http://www.spritebox.net/>

<http://spritepad.wearekiss.com/>

Profile Page Loading

- Use Your Browser
 - * **Developer Tools or Equivalent**
- Do Remote Tests
 - * <http://www.webpagetest.org/>
 - * <https://developers.google.com/speed/pagespeed/>
 - * <https://www.modern.ie/en-us>
- Actual User Profiling
 - * <http://www.lognormal.com/boomerang/doc/>
 - * **Use Web-Timing API directly**

Compression For The Win!



1,394 KB
59 requests,
4.63 seconds to load

Compression Reduces data-size by >50% and makes page load in 2.1 seconds!

Use gzip compression

965.8 KB total in compressible text, savings = 695.2 KB

Compress Images

171.6 KB total in images, savings = 51.8 KB

PHPConference.com via http://www.webpagetest.org

					Document Complete			Fully Loaded		
	Load Time	First Byte	Start Render	DOM Elements	Time	Requests	Bytes In	Time	Reque sts	Bytes In
First View	3.467s	0.314s	2.280s	1985	3.467s	84	711 KB	3.547s	85	712 KB
Repeat View	1.641s	0.630s	1.262s	1985	1.641s	10	145 KB	1.641s	10	145 KB

Cache, Cache, Cache

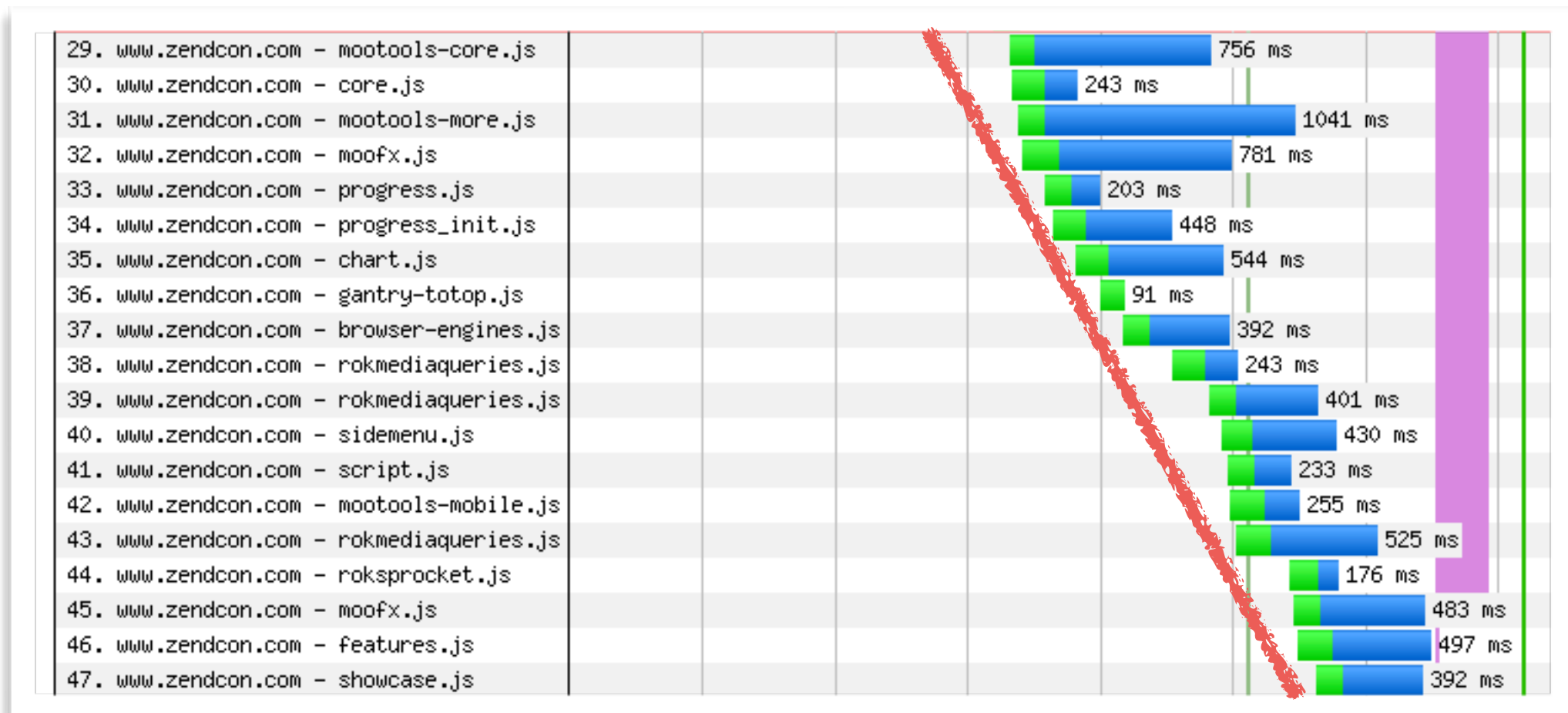
Set **max-age** or **expires** headers

Value should be at least **30 days**

To prevent stale content, **use unique file names** on new deployments for changed files.

Your goal is that 2nd page load only queries the server for the dynamic content!

JavaScript



**JavaScript is loaded synchronously,
so compact your files into a single
compressed file!**

JavaScript

Combination & minifying of JS files is best achieved with:

- * Closure Compiler

- * <http://goo.gl/8MVOIJ>

- * YUI Compressor

- * <http://refresh-sf.com/yui/>

- * <http://yui.github.io/yuicompressor/>

- * PHP Based

- * <https://github.com/tedious/JShrink>

JavaScript

Don't over-do combining of JS Files!

- ▶ Unnecessary data loading
- ▶ Decompression Overhead
- ▶ Extra JS Compilation



Micro-Case Study: SlashDot.org

One “BIG” JavaScript file

71kb compressed, 251kb actual size

199ms to receive

37ms to process

21.3% of total page load, 16% of total page size

< 10% of loaded JS code is executed

JavaScript

Only load up-front what you absolutely need

Defer loading of everything else via RequireJS

```
<head>  
  <script src="scripts/require.js"></script>  
</head>
```

```
require.config({  
  baseUrl: 'js/lib',  
  paths: { jquery: 'jquery-1.11.1' }  
});
```

```
define(['lib/jquery'], function ($) {...});
```

<http://requirejs.org/>

If you can't win, cheat!



```
$(document).ready(function() {  
    setTimeout(function() {  
        $.get( "your-file.js" );  
    }, 2000);  
});
```

General JS Tips

- 1. Avoid Xpath, reference/search by ID**
- 2. Setup events pre-load as opposed to post-load**

`onkeyup="js_function()"` vs `$("input").each(function() {});`

- 3. For Grids only load the data to be displayed**
- 4. innerHTML is not always faster than DOM**

<http://jsperf.com/dom-vs-innerhtml/37>

General JS Tips

- Most browsers leak memory with JS, avoid the common culprits:
 - ◆ Use closures
 - ◆ Avoid passing objects (can result in circular references)
 - ◆ Avoid global variables

General JS Tips

- Help browser to make use of multiple CPUs by using iframes to embed complex components such as grids.



CSS

* Minimize

* Combine

* Compress



* Don't fear inlined (`<style>`) CSS

Avoid Repaints & Reflows

- Changes to DOM nodes
- Hiding DOM nodes
- Actions that extend the page (causes scroll)
- Changes to colour, background and outline properties

Merge Style Changes

```
// slowest  
el.style.left = "10px";  
el.style.top  = "10px";
```

```
// getting better  
el.className += " top-left-class";
```

```
// best  
el.style.cssText += "; left: 10px; top: 10px;";
```

Peekaboo Trick

```
var me = $("#e1");  
me.hide();
```

```
// make various changes to DOM/Content
```

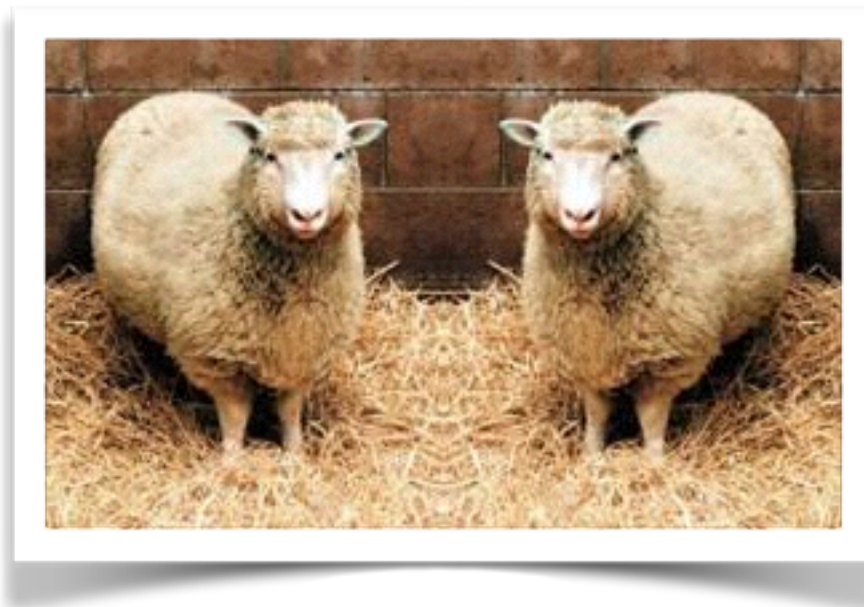
```
me.show();
```

Dolly Trick

```
var $dolly = el.clone();
```

```
// make changes to the copy
```

```
el.replaceWith($dolly);
```



Good Reference Points

<http://www.phpied.com/rendering-repaint-reflowrelayout-restyle/>

<http://www-archive.mozilla.org/newlayout/doc/reflow.html>

<https://developers.google.com/speed/articles/reflow>

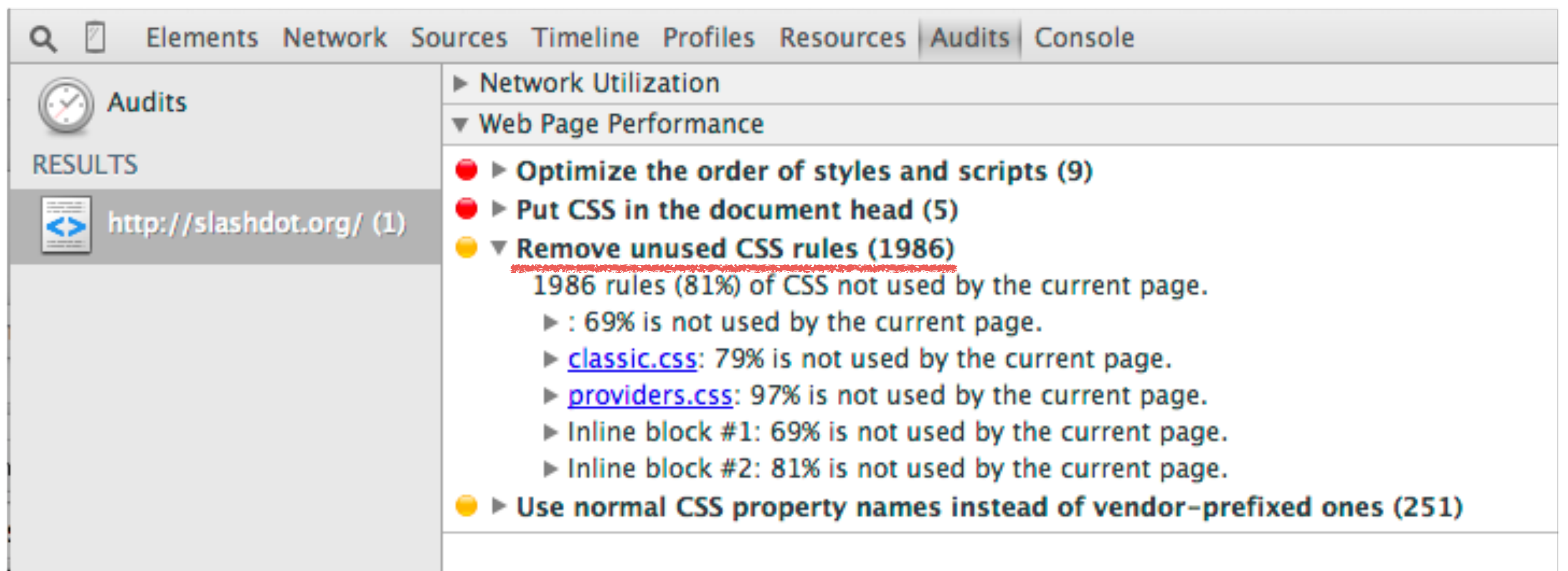
More CSSery

- Reference by element ID
- Be specific, but avoid child selectors
- Avoid @import()
- Avoid multi-class css rule (.foo.bar.baz)

More CSSery

- Pseudo selectors are slow
- Name space attribute selectors
(`type="..."` vs `input[type="..."]`)
- Eliminate un-used rules
- Avoid browser specific extensions
(`-webkit`, `-opera`, `-moz`, etc...)

Micro-Case Study: SlashDot.org



The screenshot shows the Chrome DevTools interface with the 'Audits' tab selected. The left sidebar shows the 'Audits' section with a clock icon and the word 'Audits'. Below it, the 'RESULTS' section shows a document icon and the URL 'http://slashdot.org/ (1)'. The main panel displays a list of audits under the 'Web Page Performance' category. The first three audits are: 'Optimize the order of styles and scripts (9)' (red dot), 'Put CSS in the document head (5)' (red dot), and 'Remove unused CSS rules (1986)' (yellow dot). The 'Remove unused CSS rules' audit is expanded, showing a summary: '1986 rules (81%) of CSS not used by the current page.' followed by a list of specific rules and their usage percentages: ': 69% is not used by the current page.', 'classic.css: 79% is not used by the current page.', 'providers.css: 97% is not used by the current page.', 'Inline block #1: 69% is not used by the current page.', and 'Inline block #2: 81% is not used by the current page.' The final audit is 'Use normal CSS property names instead of vendor-prefixed ones (251)' (yellow dot).

Elements Network Sources Timeline Profiles Resources Audits Console

Audits

RESULTS

<http://slashdot.org/> (1)

- ▶ Network Utilization
- ▼ Web Page Performance
 - ▶ **Optimize the order of styles and scripts (9)**
 - ▶ **Put CSS in the document head (5)**
 - ▼ **Remove unused CSS rules (1986)**
 - 1986 rules (81%) of CSS not used by the current page.
 - ▶ : 69% is not used by the current page.
 - ▶ [classic.css](#): 79% is not used by the current page.
 - ▶ [providers.css](#): 97% is not used by the current page.
 - ▶ Inline block #1: 69% is not used by the current page.
 - ▶ Inline block #2: 81% is not used by the current page.
 - ▶ **Use normal CSS property names instead of vendor-prefixed ones (251)**

CSS Tools

<https://github.com/Cerdic/CSSTidy> **PHP**

<http://devilo.us/> **Web-based**

THANK YOU FOR
LISTENING

Slides: <http://ilia.ws>

@iliaa