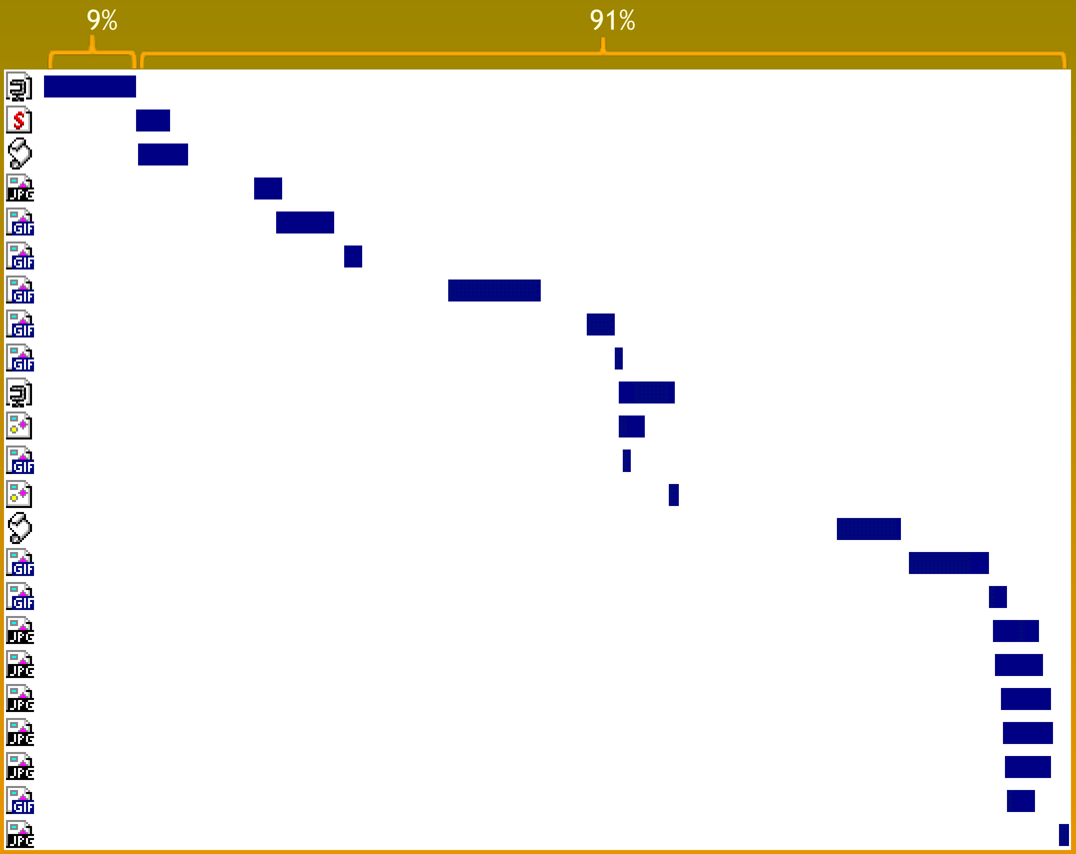
Even Faster Web Sites

Flushing the Document Early
Simplifying CSS Selectors
Avoiding @import

Steve Souders
souders@google.com

http://stevesouders.com/docs/web20expo-20090402.ppt

the importance of frontend performance

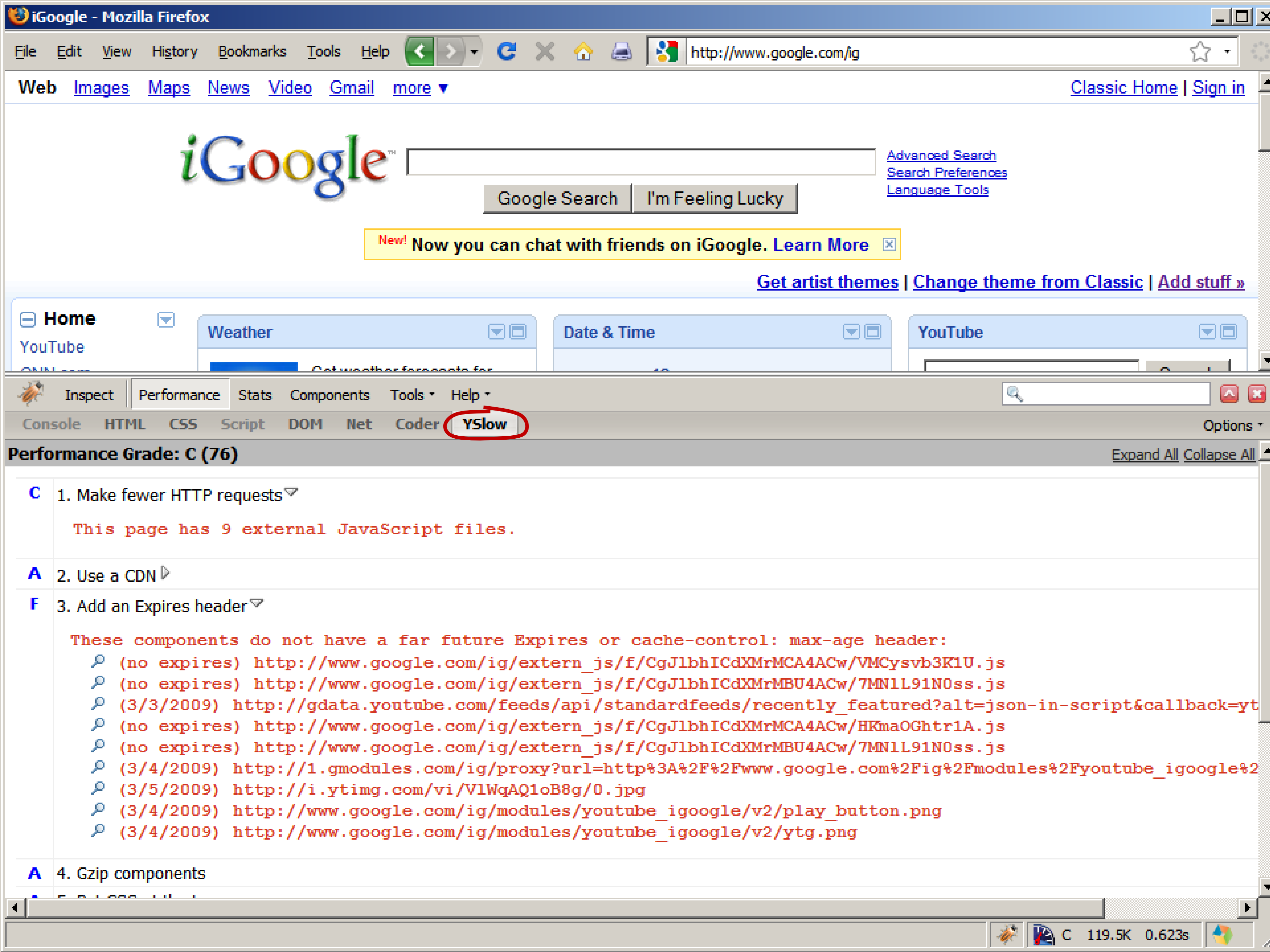


iGoogle, empty cache

time spent on the frontend

	Empty Cache	Primed Cache
www.aol.com	97%	97%
www.ebay.com	95%	81%
www.facebook.com	95%	81%
www.google.com/search	47%	0%
search.live.com/results	67%	0%
www.msn.com	98%	94%
www.myspace.com	98%	98%
en.wikipedia.org/wiki	94%	91%
www.yahoo.com	97%	96%
www.youtube.com	98%	97%

April 2008


iGoogle - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.google.com/ig
Web Images Maps News Video Gmail more ▾
Classic Home | Sign in
iGoogle
Google Search
I'm Feeling Lucky
Advanced Search
Search Preferences
Language Tools
New! Now you can chat with friends on iGoogle. Learn More
Get artist themes | Change theme from Classic | Add stuff »
Home
YouTube
Weather
Date & Time
YouTube
Inspect Performance Stats Components Tools Help
Console HTML CSS Script DOM Net Coder YSlow
Options
Expand All Collapse All
C 119.5K 0.623s

Performance Grade: C (76)

C 1. Make fewer HTTP requests

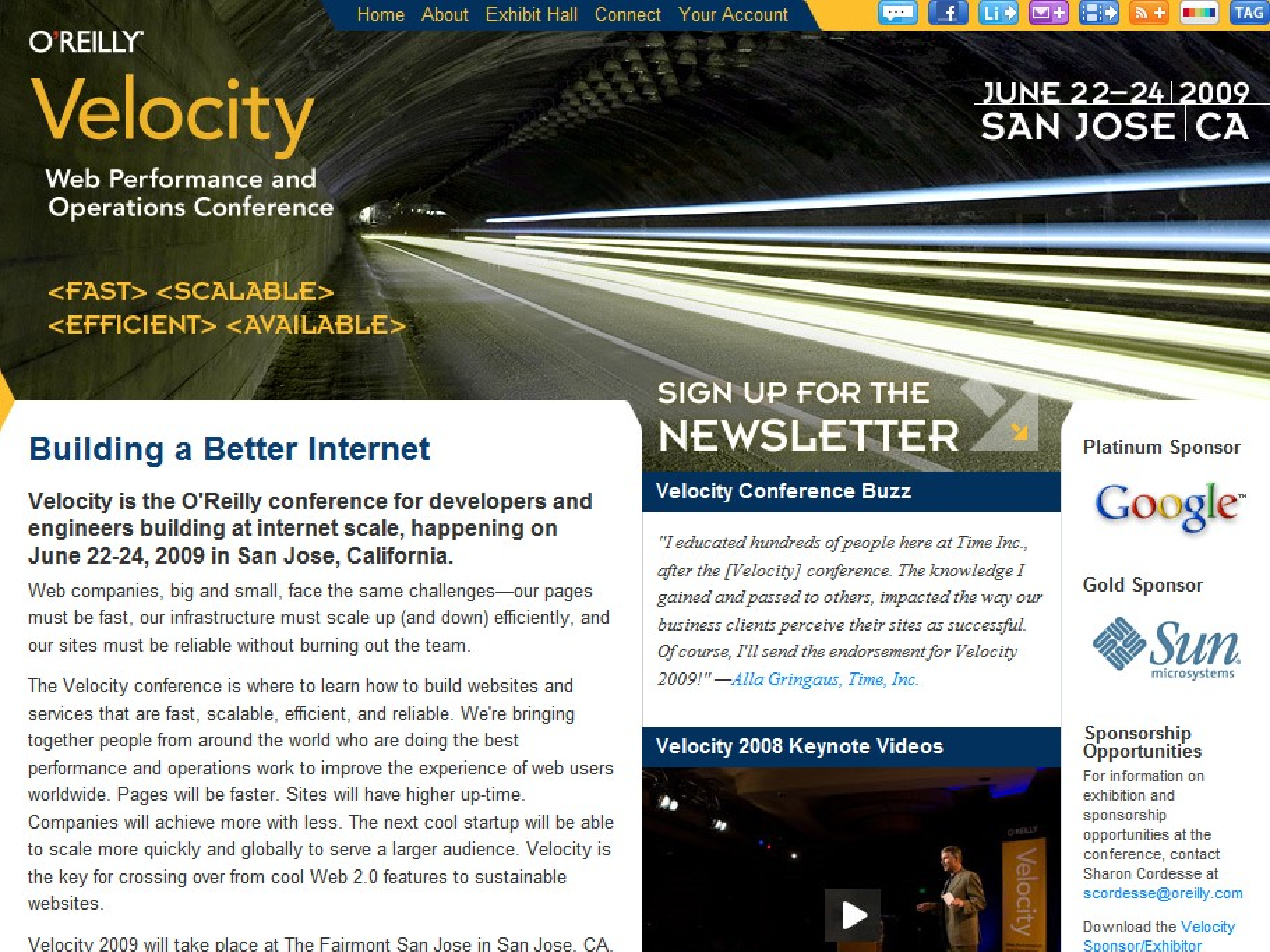
```
This page has 9 external JavaScript files.
```

A 2. Use a CDN

F 3. Add an Expires header

```
These components do not have a far future Expires or cache-control: max-age header:
  (no expires) http://www.google.com/ig/extern_js/f/CgJlbhICdXMrMCA4ACw/VMCysvb3K1U.js
  (no expires) http://www.google.com/ig/extern_js/f/CgJlbhICdXMrMBU4ACw/7MNlL91N0ss.js
  (3/3/2009) http://gdata.youtube.com/feeds/api/standardfeeds/recently_featured?alt=json-in-script&callback=yt
  (no expires) http://www.google.com/ig/extern_js/f/CgJlbhICdXMrMCA4ACw/HKmaOGhtr1A.js
  (no expires) http://www.google.com/ig/extern_js/f/CgJlbhICdXMrMBU4ACw/7MNlL91N0ss.js
  (3/4/2009) http://1.gmodules.com/ig/proxy?url=http%3A%2F%2Fwww.google.com%2Fig%2Fmodules%2Fyoutube_igoogle%2
  (3/5/2009) http://i.ytimg.com/vi/VlWqAQ1oB8g/0.jpg
  (3/4/2009) http://www.google.com/ig/modules/youtube_igoogle/v2/play_button.png
  (3/4/2009) http://www.google.com/ig/modules/youtube_igoogle/v2/ytg.png
```

A 4. Gzip components



Building a Better Internet

Velocity is the O'Reilly conference for developers and engineers building at internet scale, happening on June 22-24, 2009 in San Jose, California.

Web companies, big and small, face the same challenges—our pages must be fast, our infrastructure must scale up (and down) efficiently, and our sites must be reliable without burning out the team.

The Velocity conference is where to learn how to build websites and services that are fast, scalable, efficient, and reliable. We're bringing together people from around the world who are doing the best performance and operations work to improve the experience of web users worldwide. Pages will be faster. Sites will have higher up-time. Companies will achieve more with less. The next cool startup will be able to scale more quickly and globally to serve a larger audience. Velocity is the key for crossing over from cool Web 2.0 features to sustainable websites.

Velocity 2009 will take place at The Fairmont San Jose in San Jose, CA.

Velocity Conference Buzz

"I educated hundreds of people here at Time Inc., after the [Velocity] conference. The knowledge I gained and passed to others, impacted the way our business clients perceive their sites as successful. Of course, I'll send the endorsement for Velocity 2009!" —Alla Gringaus, Time, Inc.

Velocity 2008 Keynote Videos

Platinum Sponsor

Google

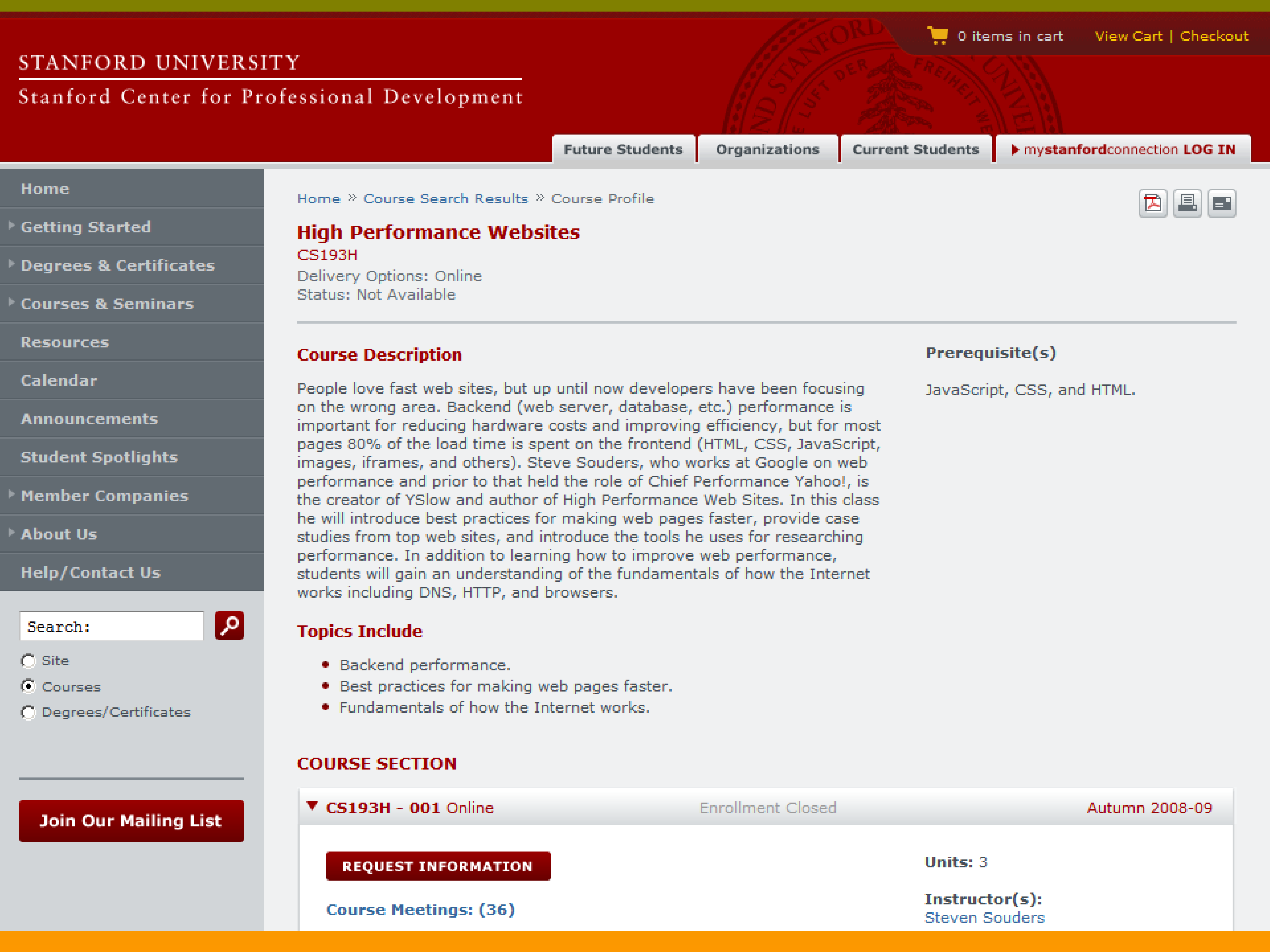
Gold Sponsor

Sun microsystems

Sponsorship Opportunities

For information on exhibition and sponsorship opportunities at the conference, contact Sharon Cordesse at scordesse@oreilly.com

Download the Velocity Sponsor/Exhibitor



STANFORD UNIVERSITY
Stanford Center for Professional Development

0 items in cart | View Cart | Checkout

Future Students | Organizations | Current Students | mystanfordconnection LOG IN

- Home
- Getting Started
- Degrees & Certificates
- Courses & Seminars
- Resources
- Calendar
- Announcements
- Student Spotlights
- Member Companies
- About Us
- Help/Contact Us

Search:

Site
[✓ Courses]
Degrees/Certificates

Join Our Mailing List

Home » Course Search Results » Course Profile

High Performance Websites

CS193H
Delivery Options: Online
Status: Not Available

Course Description

People love fast web sites, but up until now developers have been focusing on the wrong area. Backend (web server, database, etc.) performance is important for reducing hardware costs and improving efficiency, but for most pages 80% of the load time is spent on the frontend (HTML, CSS, JavaScript, images, iframes, and others). Steve Souders, who works at Google on web performance and prior to that held the role of Chief Performance Yahoo!, is the creator of YSlow and author of High Performance Web Sites. In this class he will introduce best practices for making web pages faster, provide case studies from top web sites, and introduce the tools he uses for researching performance. In addition to learning how to improve web performance, students will gain an understanding of the fundamentals of how the Internet works including DNS, HTTP, and browsers.

Prerequisite(s)

JavaScript, CSS, and HTML.

Topics Include

- Backend performance.
- Best practices for making web pages faster.
- Fundamentals of how the Internet works.

COURSE SECTION

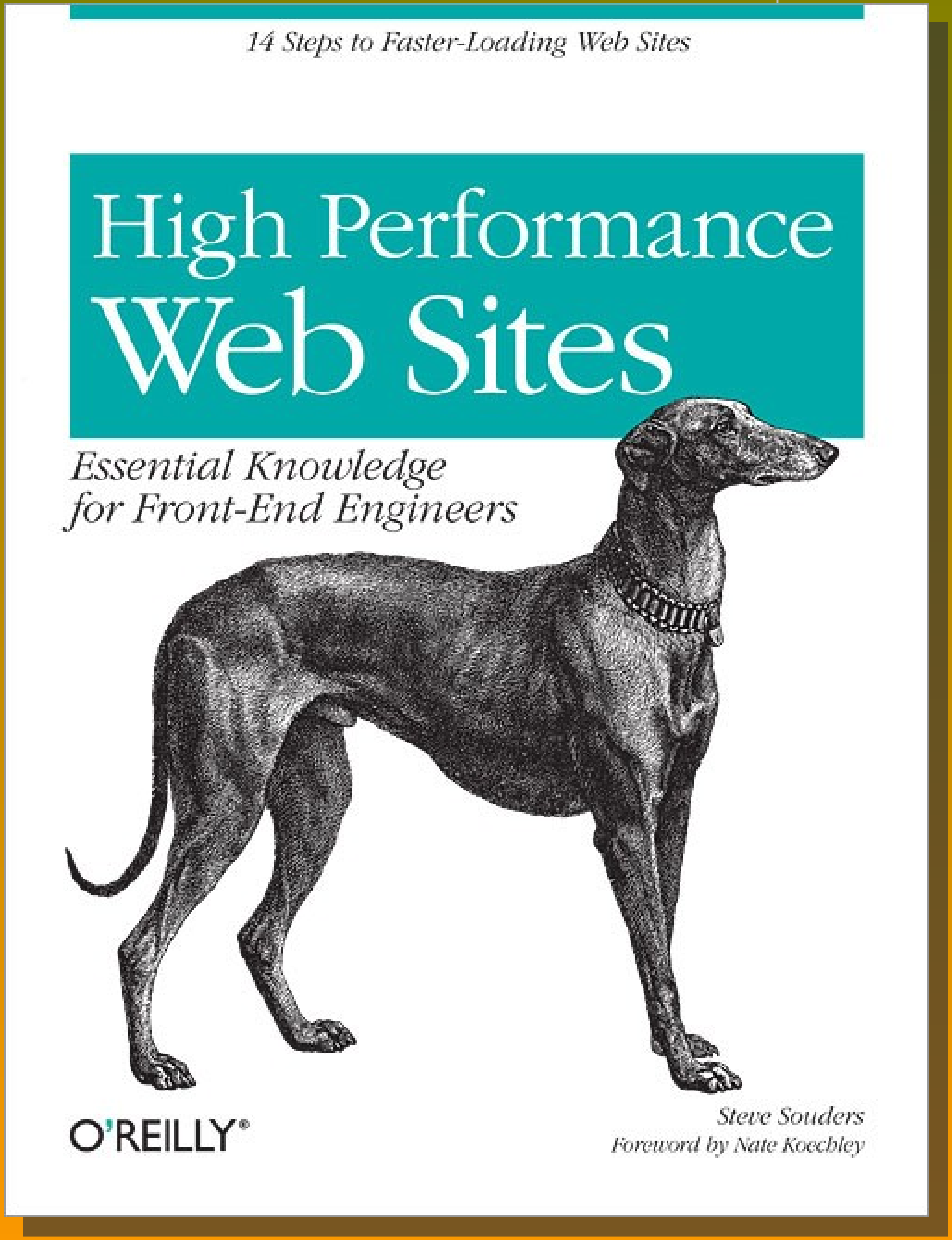
CS193H - 001 Online | Enrollment Closed | Autumn 2008-09

REQUEST INFORMATION

Course Meetings: (36)

Units: 3

Instructor(s):
Steven Souders



14 Steps to Faster-Loading Web Sites

High Performance Web Sites

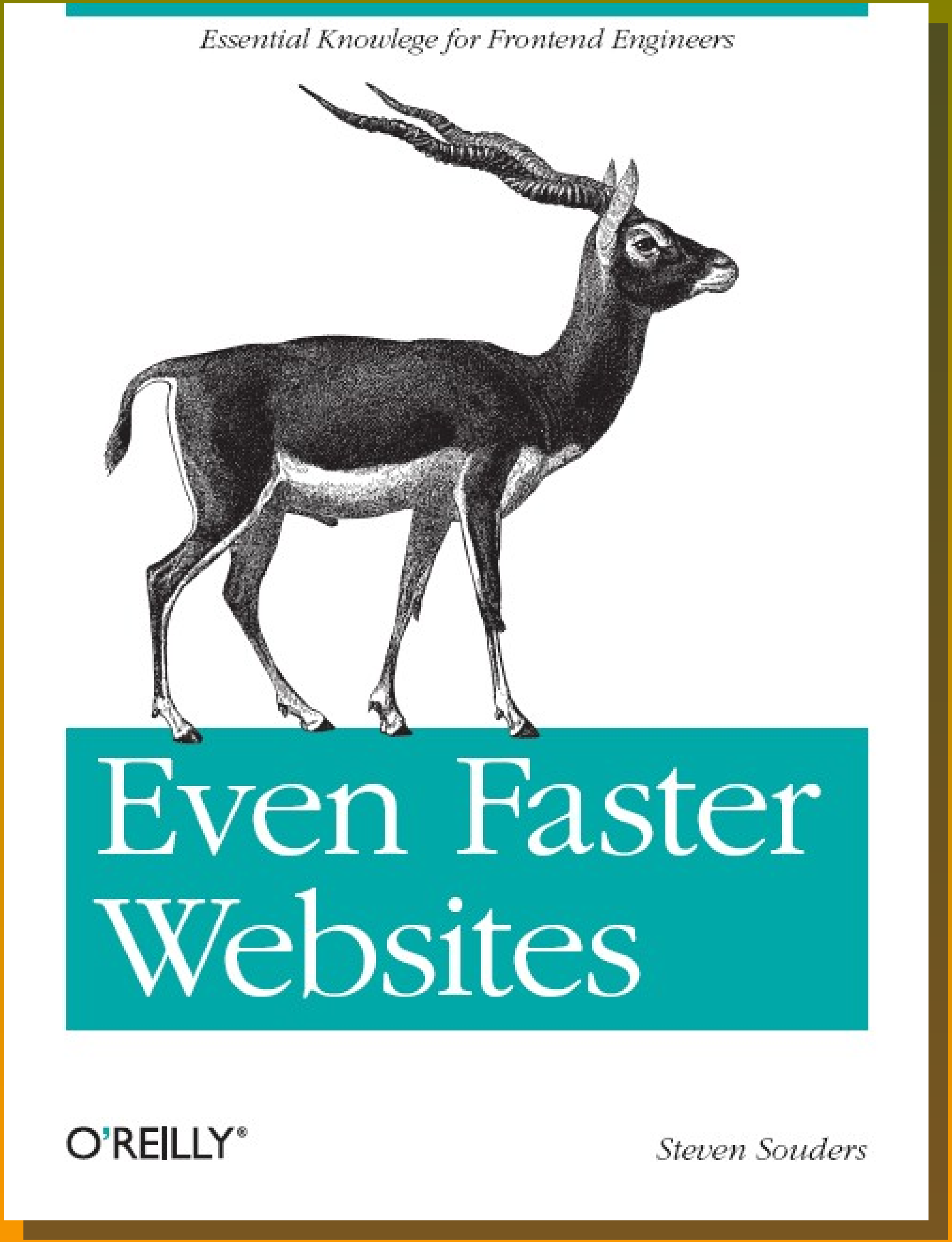
Essential Knowledge for Front-End Engineers

O'REILLY®

Steve Souders

Foreword by Nate Koechley

Sept 2007



Essential Knowlege for Frontend Engineers

Even Faster Websites

O'REILLY®

Steven Souders

June 2009

Even Faster Web Sites

Splitting the initial payload

Loading scripts without blocking

Coupling asynchronous scripts

Positioning inline scripts

Sharding dominant domains

Flushing the document early

Avoiding @import

Using iframes sparingly

Simplifying CSS Selectors

Understanding Ajax performance...*Doug Crockford*

Creating responsive web apps......*Ben Galbraith, Dion Almaer*

Writing efficient JavaScript...........*Nicholas Zakas*

Scaling with Comet......................*Dylan Schiemann*

Going beyond gzipping...............*Tony Gentilcore*

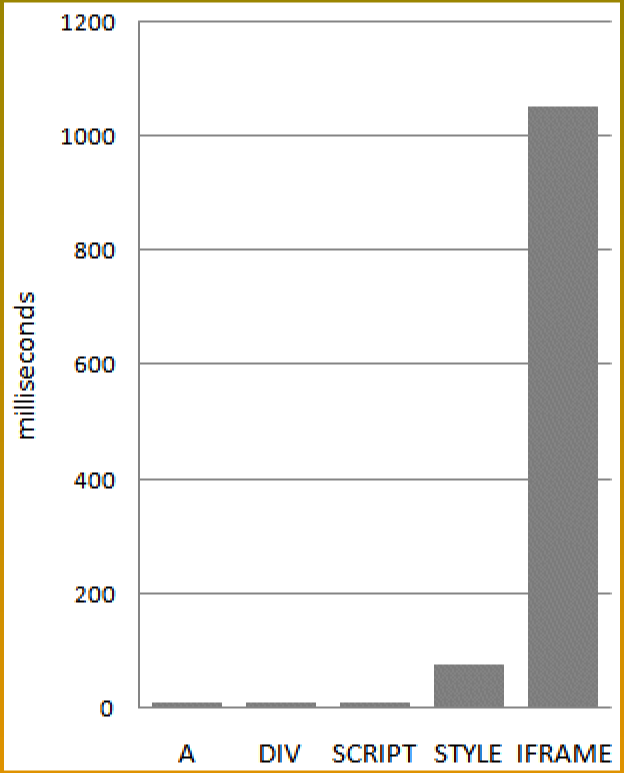
Optimizing images.....................*Stoyan Stefanov, Nicole Sullivan*

iframes:

most expensive DOM element

load 100 *empty* elements of each type

tested in all major browsers1



1IE 6, 7, 8; FF 2, 3.0, 3.1b2; Safari 3.2, 4; Opera 9.63, 10; Chrome 1.0, 2.0

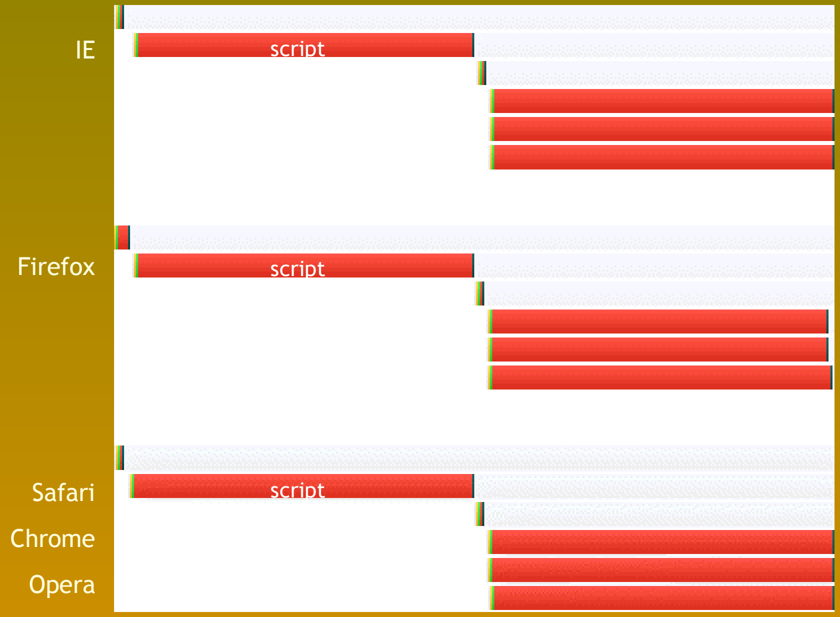
iframes block onload

parent's onload doesn't fire until iframe and all its components are downloaded

workaround for Safari and Chrome: set iframe src in JavaScript

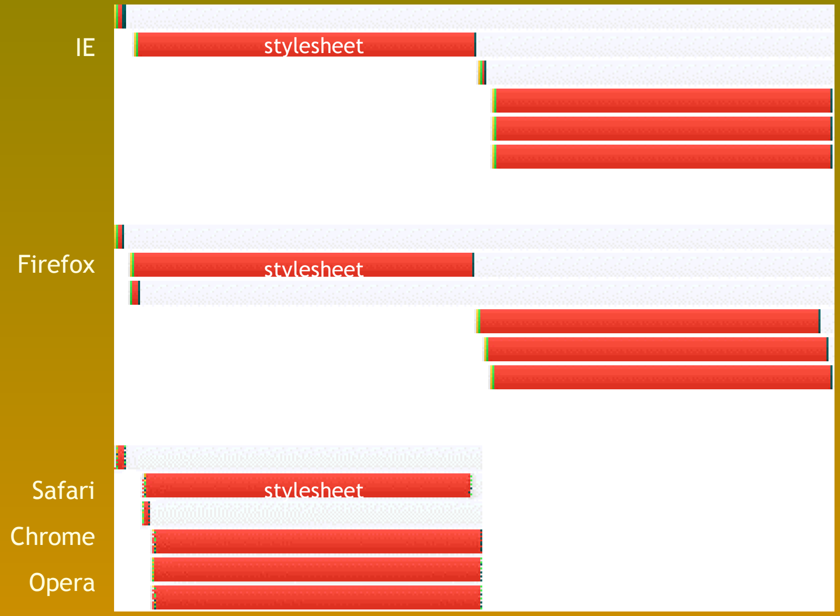
```
<iframe id=iframe1 src=""></iframe>
<script type="text/javascript">
document.getElementById('iframe1').src="url";
</script>
```

scripts block iframe



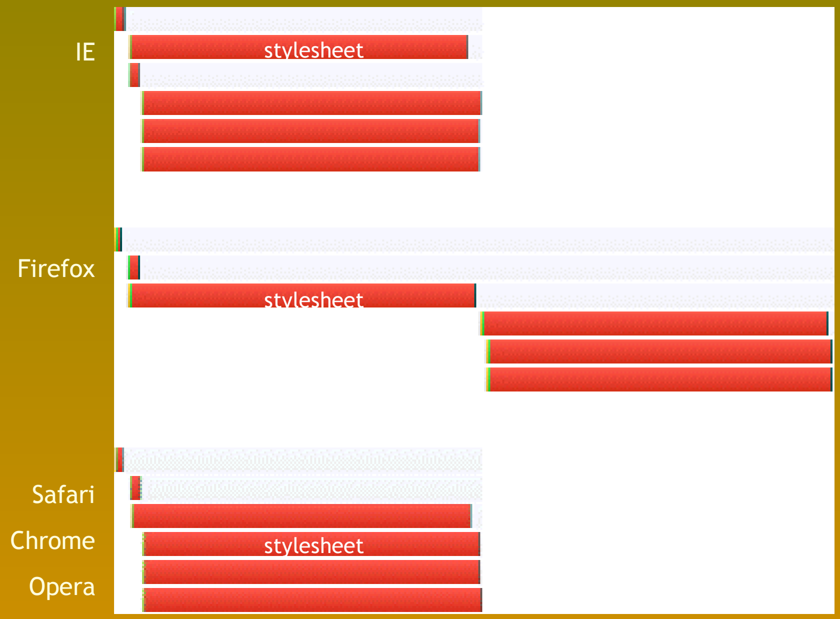
no surprise – scripts in the parent block the iframe from loading

stylesheets block iframe (IE, FF)



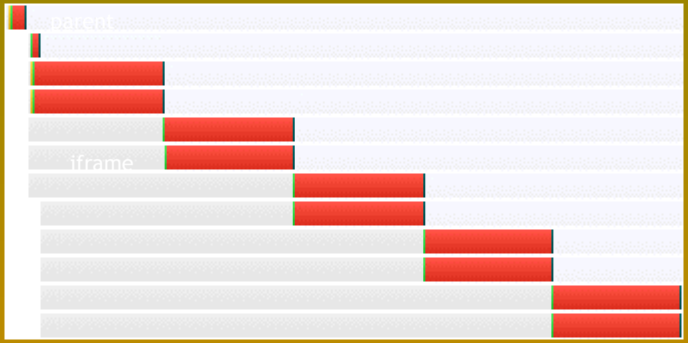
surprise – stylesheets in the parent block the iframe or its resources in IE & Firefox

stylesheets after iframe still block (FF)



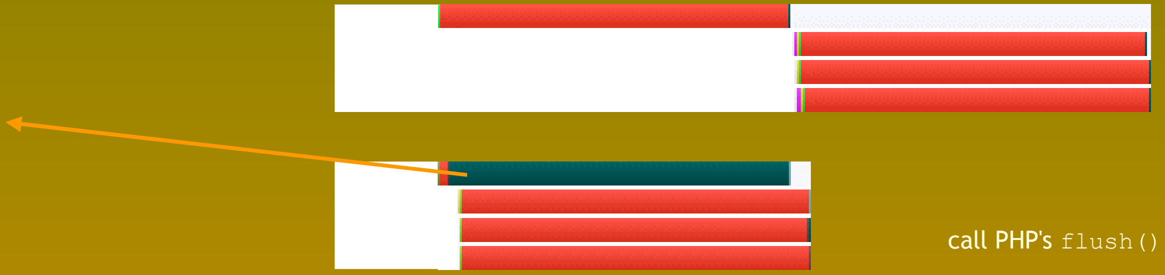
surprise – even moving the stylesheet *after* the iframe still causes the iframe's resources to be blocked in Firefox

iframes: no free connections



iframe shares connection pool with parent (here – 2 connections per server in IE 7)

flushing the document early



gotchas:

PHP output_buffering - `ob_flush()`

Transfer-Encoding: chunked

gzip - Apache's DeflateBufferSize before 2.2.8

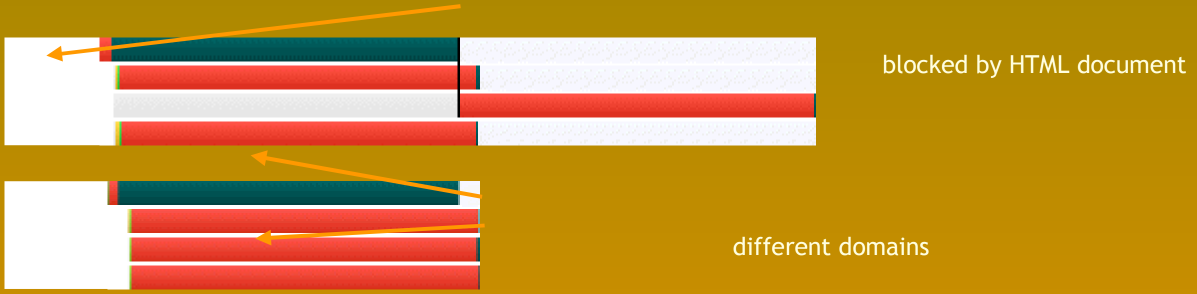
proxies and anti-virus software

browsers - Safari (1K), Chrome (2K)

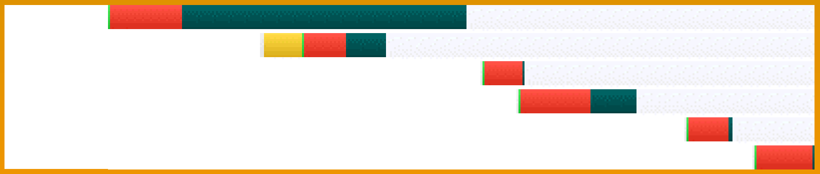
other languages:

flushing and domain blocking

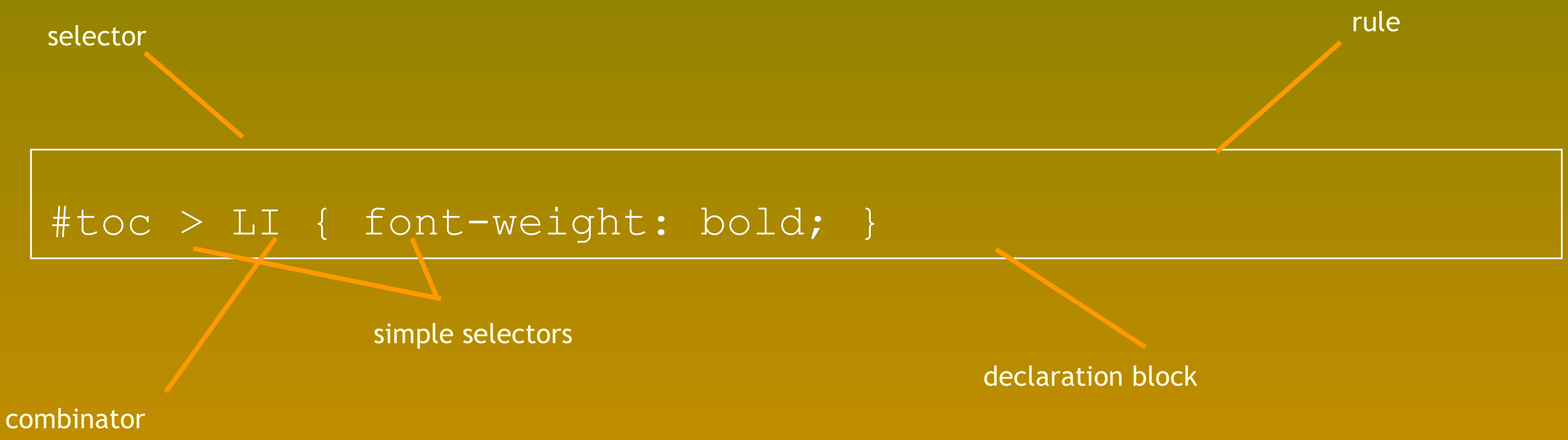
you might need to move flushed resources to a domain different from the HTML doc



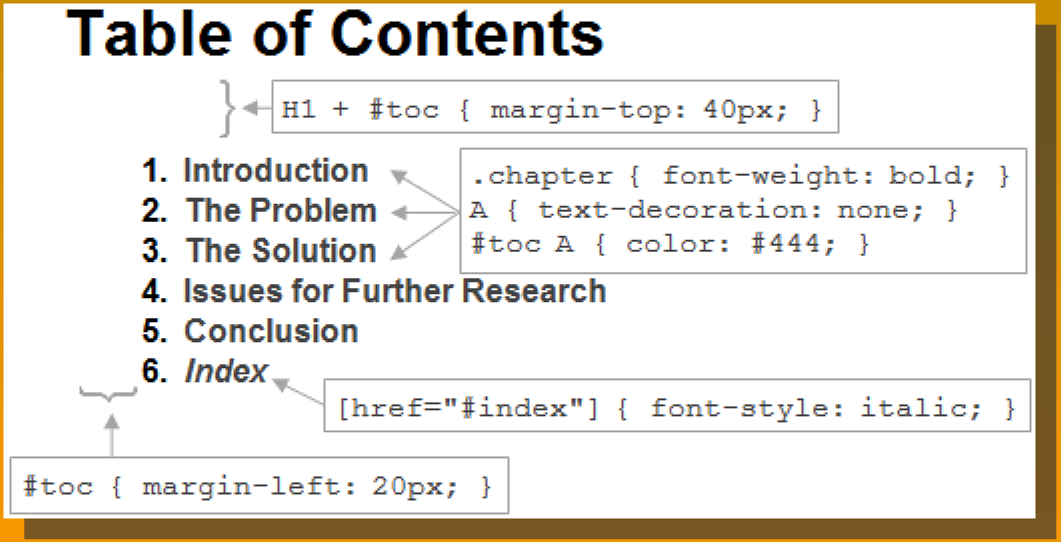
case study: Google search



Simplifying CSS Selectors



```
#toc > LI { font-weight: bold; }
```



```
H1 + #toc { margin-top: 40px; }
```

```
.chapter { font-weight: bold; }
A { text-decoration: none; }
#toc A { color: #444; }
```

```
[href="#index"] { font-style: italic; }
```

```
#toc { margin-left: 20px; }
```

types of CSS selectors

ID selectors

```
#toc { margin-left: 20px; }
```

element whose ID attribute has the value "toc"

class selectors

```
.chapter { font-weight: bold; }
```

elements with class=chapter

type selectors

```
A { text-decoration: none; }
```

all A elements in the document tree

types of CSS selectors

adjacent sibling selectors

```
H1 + #toc { margin-top: 40px; }
```

an element with ID=toc that immediately follows an H1

child selectors

```
#toc > LI { font-weight: bold; }
```

all LI elements whose parent has id="toc"

descendant selectors

```
#toc A { color: #444; }
```

all A elements that have id="toc" as an ancestor

types of CSS selectors

universal selectors

```
* { font-family: Arial; }
```

all elements

attribute selectors

```
[href="#index"] { font-style: italic; }
```

all elements where the href attribute is "#index"

psuedo classes and elements

```
A:hover { text-decoration: underline; }
```

non-DOM behavior

others: `:visited :link :active :focus :first-child :before :after`

writing efficient CSS

https://developer.mozilla.org/en/Writing_Efficient_CSS

"The style system matches a rule by starting with the rightmost selector and moving to the left through the rule's selectors. As long as your little subtree continues to check out, the style system will continue moving to the left until it either matches the rule or bails out because of a mismatch."

```
#toc > LI { font-weight: bold; }
```

find every LI whose parent is id="toc"

```
#toc A { color: #444; }
```

find every A and climb its ancestors until id="toc" or DOM root (!) is found

writing efficient CSS

1. avoid universal selectors
2. don't qualify ID selectors

bad: `DIV #navbar {}`

good: `#navbar {}`

1. don't qualify class selectors

bad: `LI .tight {}`

good: `.li-tight {}`

1. make rules as specific as possible

bad: `#navbar A {}`

https://developer.mozilla.org/en/Writing_Efficient_CSS

writing efficient CSS

1. avoid descendant selectors

bad: `UL LI A {}`

better: `UL > LI > A {}`

1. avoid tag-child selectors

bad: `UL > LI > A {}`

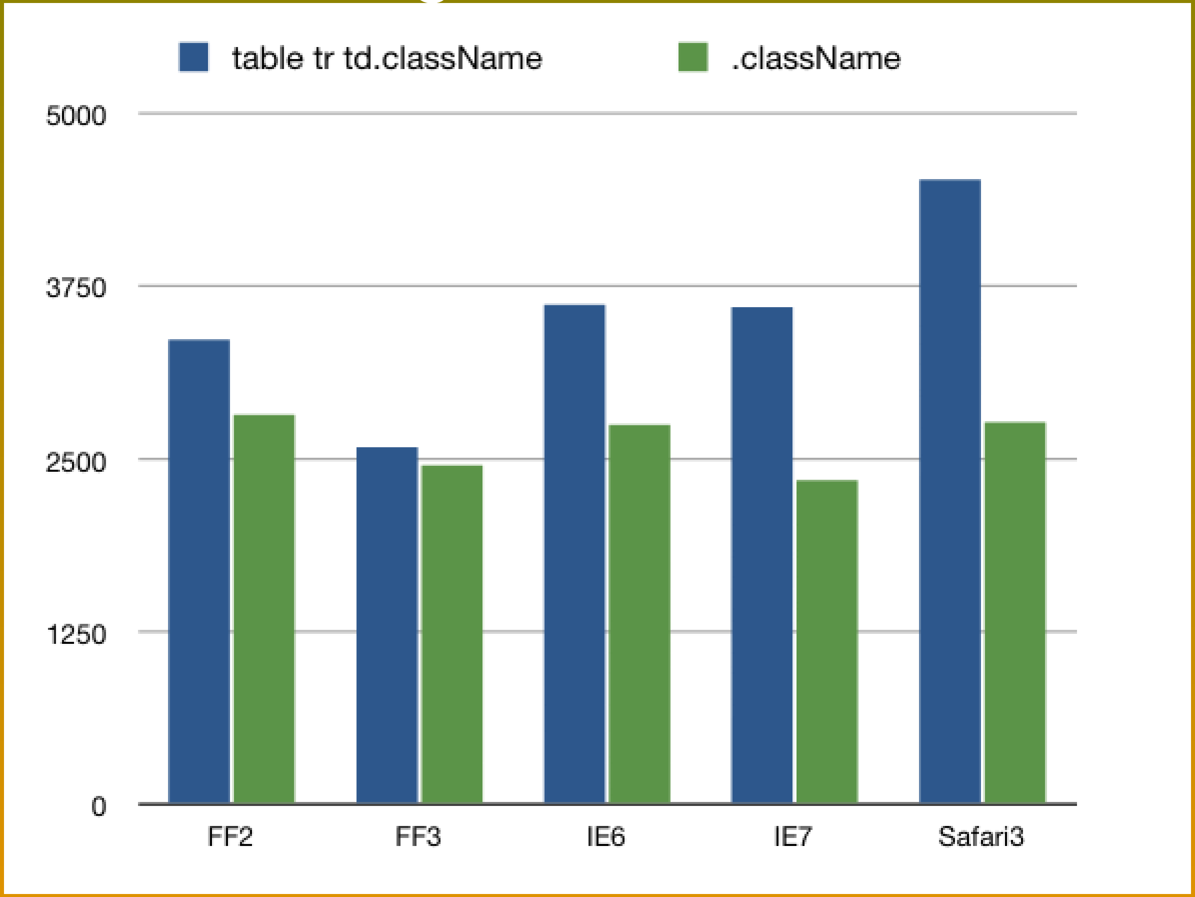
best: `.li-anchor {}`

1. be wary of child selectors
2. rely on inheritance

https://developer.mozilla.org/en/Writing_Efficient_CSS

http://www.w3.org/TR/CSS21/propidx.html

Testing CSS Performance



20K TD elements

http://jon.sykes.me/152/testing-css-performance-pt-2

testing massive CSS

20K `A` elements

no style: control

tag:

```
A {}
```

class:

```
.a00001 {}
```

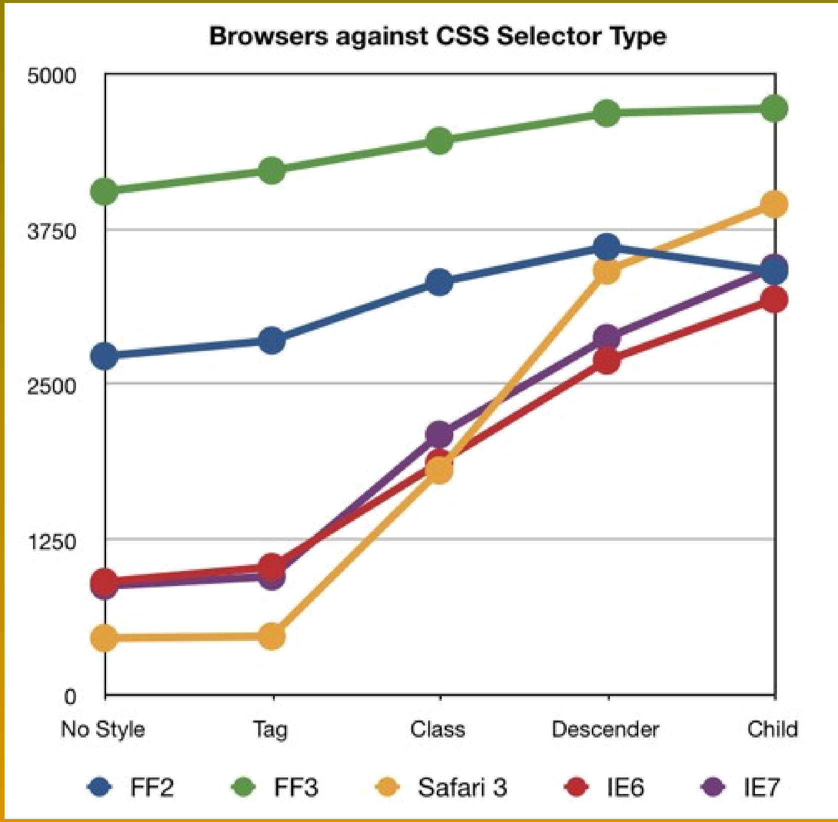
```
.a20000 {}
```

descender:

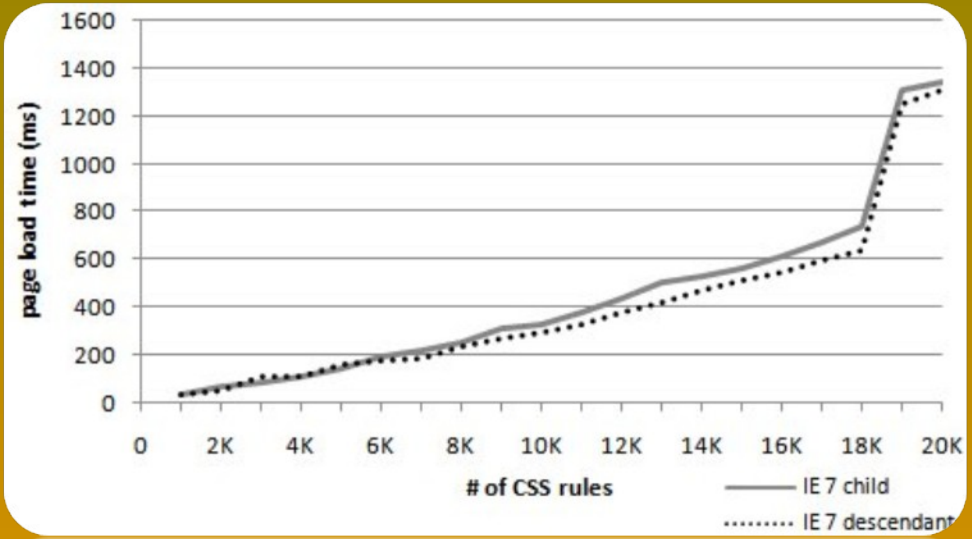
```
DIV DIV DIV P A.a00001 {}
```

child:

```
DIV > DIV > DIV > P > A.a00001 {}
```



CSS performance isn't linear



IE 7 "cliff" at 18K rules

real world levels of CSS

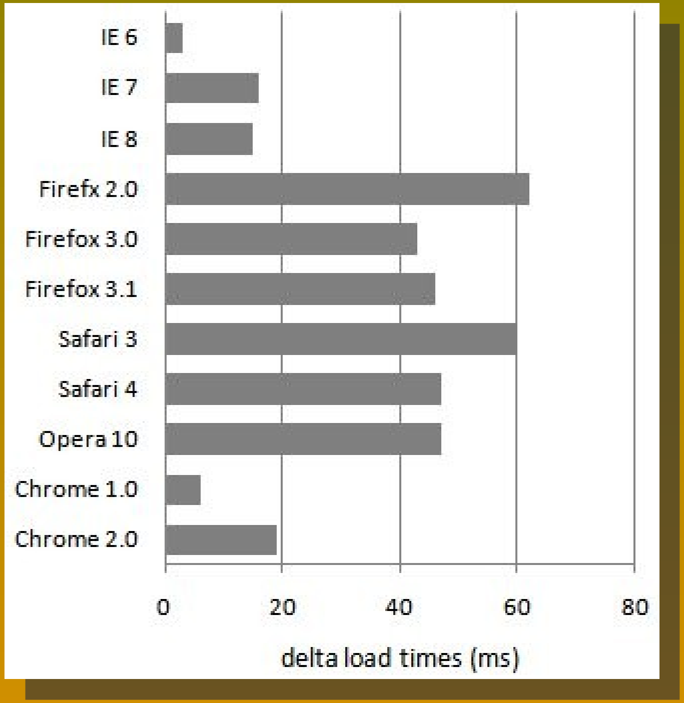
	# Rules	# elements	Avg Depth
AOL	2289	1628	13
eBay	305	588	14
Facebook	2882	1966	17
Google Search	92	552	8
Live Search	376	449	12
MSN.com	1038	886	11
MySpace	932	444	9
Wikipedia	795	1333	10
Yahoo!	800	564	13
YouTube	821	817	9
average	**1033**	**923**	**12**

testing typical CSS

1K rules (vs. 20K)

same amount of CSS in all test pages

30 ms avg delta



"costly"selectors aren't always costly (at typical levels)

are these selectors "costly"?

```
DIV DIV DIV P A.class0007 { ... }
```

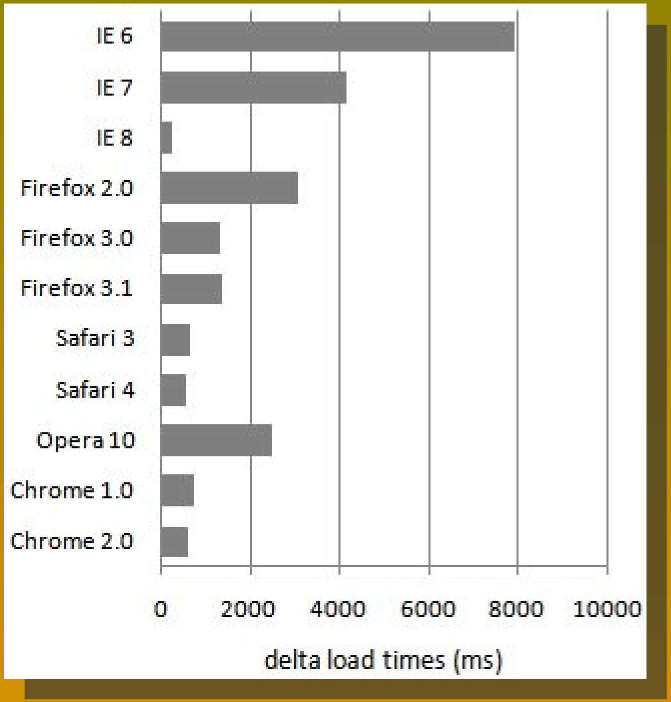
http://www.stevesouders.com/blog/2009/03/10/performance-impact-of-css-selectors/

testing expensive selectors

1K rules (vs. 20K)

same amount of CSS in all test pages

2126 ms avg delta!



truly expensive selector

```
A.class0007 * { ... }
```

compare to:

```
DIV DIV DIV P A.class0007 { ... }
```

the key is the *key selector* – the rightmost argument

CSS3 selectors (bad)

more David Hyatt:

"The sad truth about CSS3 selectors is that they really shouldn’t be used at all if you care about page performance. Decorating your markup with classes and ids and matching purely on those while avoiding all uses of sibling, descendant and child selectors will actually make a page perform significantly better in all browsers."

http://shauninman.com/archive/2008/05/05/css_qualified_selectors#comment_3942

selectors to avoid

```
A.class0007 DIV { ... }

#id0007 > A { ... }

.class0007 [href] { ... }

DIV:first-child { ... }
```

reflow time vs. load time

reflow – time to apply CSS, re-layout elements, and repaint

triggered by DHTML:

```
elem.className = "newclass";

elem.style.cssText = "color: red";

elem.style.padding = "8px";

elem.style.display = "";
```

reflow can happen multiple times for long-lasting Web 2.0 apps

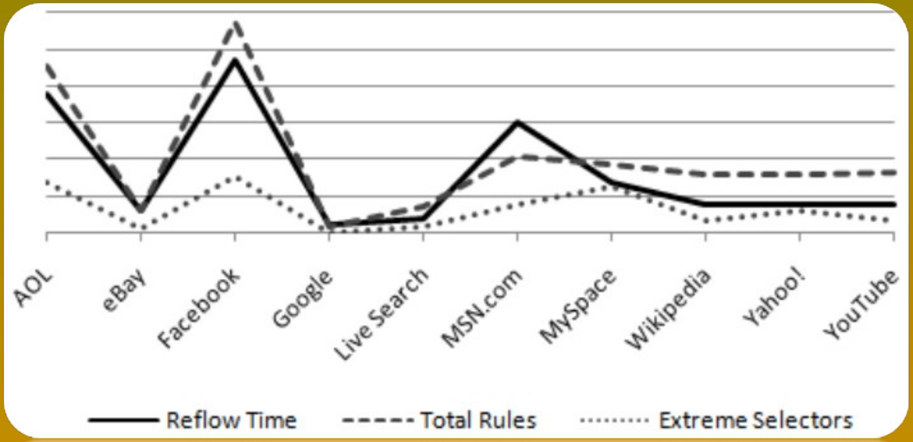
reflow time by browser

DHTML action	Chr1	Chr2	FF2	FF3	IE6,7	IE 8	Op	Saf3	Saf4
className	1x	1x	1x	1x	1x	1x	1x	1x	1x
display none	-	-	-	-	1x	-	-	-	-
display default	1x	1x	1x	2x	1x	1x	-	1x	1x
visibility hidden	1x	1x	1x	1x	1x	1x	-	1x	1x
visibility visible	1x	1x	1x	1x	1x	1x	-	1x	1x
padding	-	-	1x	2x	4x	4x	-	-	-
width length	-	-	1x	2x	1x	1x	-	1x	-
width percent	-	-	1x	2x	1x	1x	-	1x	-
width default	1x	-	1x	2x	1x	1x	-	1x	-
background	-	-	1x	1x	1x	-	-	-	-
font-size	1x	1x	1x	2x	1x	1x	-	1x	1x

reflow performance varies by browser and action

"1x" is 1-6 seconds depending on browser (1K rules)

Simplifying CSS Selectors



efficient CSS comes at a cost – page weight

focus optimization on selectors where the *key selector* matches many elements

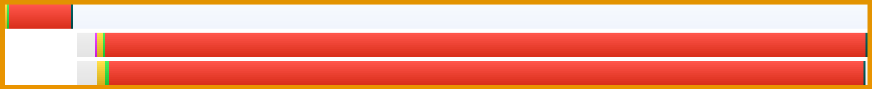
reduce the number of selectors

Avoiding @import - @import @import

```
<style>
@import url('stylesheet1.css');
@import url('stylesheet2.css');
</style>
```

no blocking

in fact, improves progressive rendering in IE



http://stevesouders.com/tests/atimport/import-import.php

link @import

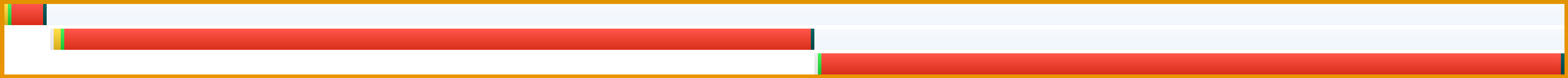
```
<link rel='stylesheet' type='text/css' href='stylesheet1.css'>

<style>

@import url('stylesheet2.css');

</style>
```

blocks in IE



http://stevesouders.com/tests/atimport/link-import.php

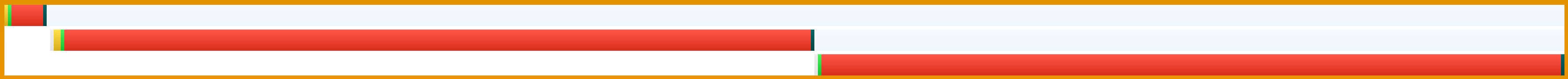
link with @import

```
<link rel='stylesheet' type='text/css' href='stylesheet1.css'>
```

includes

```
@import url('stylesheet2.css');
```

blocks in all browsers!



http://stevesouders.com/tests/atimport/link-with-import.php

link blocks @import

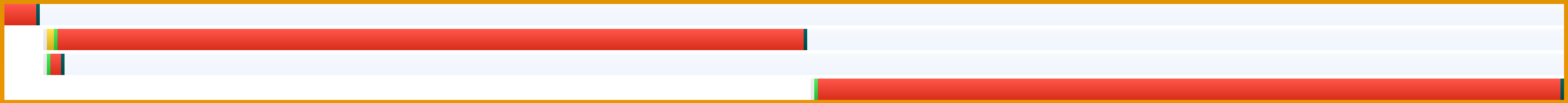
```
<link rel='stylesheet' type='text/css' href='stylesheet1.css'>

<link rel='stylesheet' type='text/css' href='proxy.css'>
```

includes

blocks in IE

```
@import url('stylesheet2.css');
```

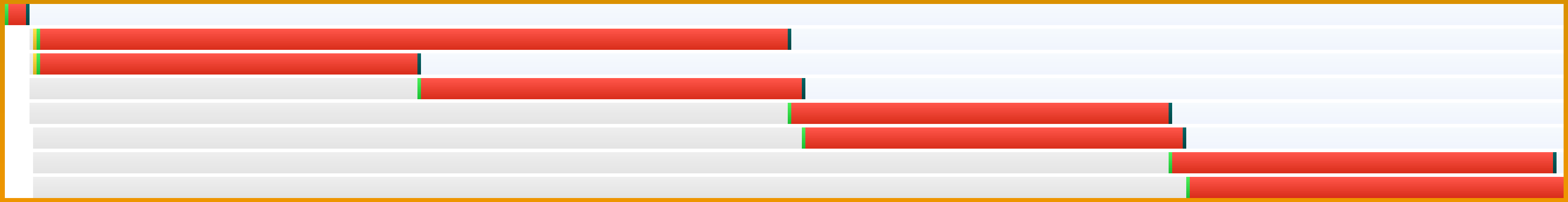


http://stevesouders.com/tests/atimport/link-blocks-import.php

many @imports

```
<style>
@import url('stylesheet1.css');
...
@import url('stylesheet6.css');
</style>
<script src='script1.js'></script>
```

loads script before stylesheets in IE

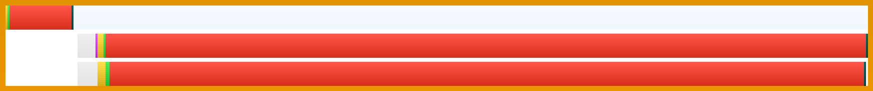


http://stevesouders.com/tests/atimport/many-imports.php

link link

```
<link rel='stylesheet' type='text/css' href='stylesheet1.css'>
<link rel='stylesheet' type='text/css' href='stylesheet2.css'>
```

no blocking in all browsers



http://stevesouders.com/tests/atimport/link-link.php

takeaways

focus on the frontend

run YSlow: http://developer.yahoo.com/yslow

speed matters

impact on revenue

Google:	+500 ms → -20% traffic[1]
Yahoo:	+400 ms → -5-9% full-page traffic[2]
Amazon:	+100 ms → -1% sales[1]

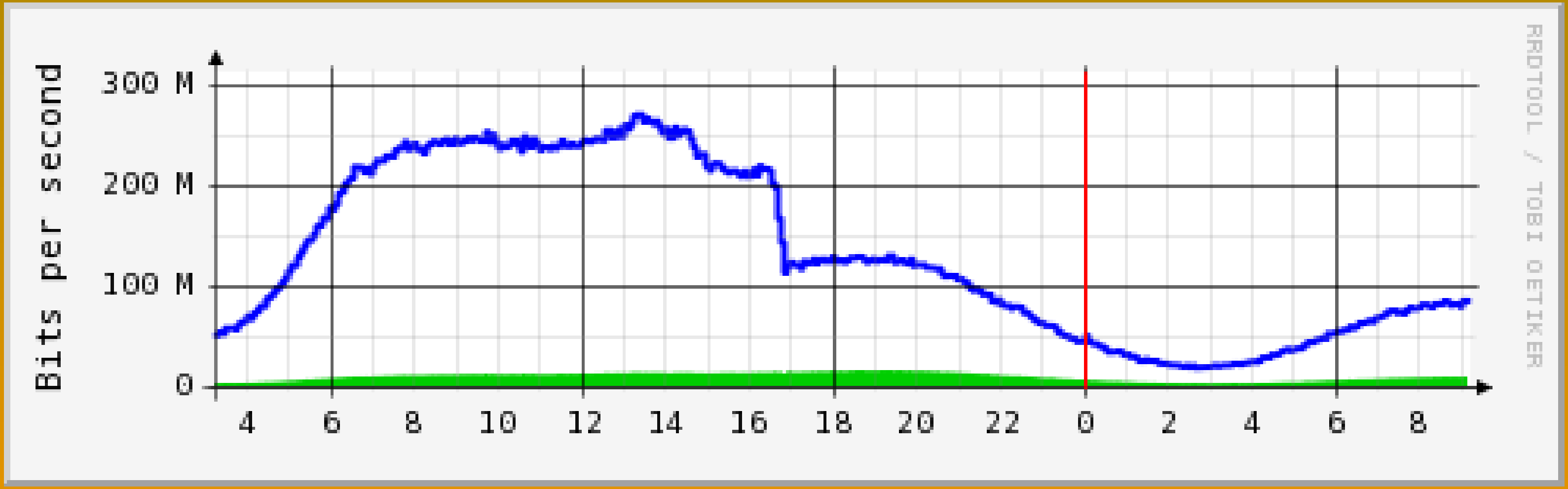
1 http://home.blarg.net/~glinden/StanfordDataMining.2006-11-29.ppt

2 http://www.slideshare.net/stoyan/yslow-20-presentation

cost savings

hardware – reduced load

bandwidth – reduced response size



http://billwscott.com/share/presentations/2008/stanford/HPWP-RealWorld.pdf if you want

better user experience

more revenue

reduced operating expenses

the strategy is clear

Steve Souders

souders@google.com

http://stevesouders.com/docs/web20expo-20090402.ppt