Hello, Barcelona!
Who Am I?
Hello, Barcelona!

**Harry** Roberts
**Consultant** Front-end Architect.
**CSS, Performance, Scalability, Architecture.**
@csswizardry
#faCSS
CSS ❤ Performance
What We Know Already
What We Know Already

*Styles at the top*; scripts at the bottom.

*Styles block rendering*; scripts block downloads.

HTTP *requests are expensive*—minimise them.

Minify, concatenate, and Gzip—*reduce transfer count and size*. 
The Critical Path
The Critical Path

Critical in our fight for performance.

Journey between a user requesting a page and them seeing something. Most of it is spent on CSS.

The Critical Path is dead time for your users.

Make this journey as short and light as we possibly can.
We've started asking for a page.

The HTML has made it over the wire (see <title>).

Now we're waiting for some CSS.

You can see the Critical Path happening!
The Critical Path

**HTML and CSS** are on the Critical Path.

**DOM and CSSOM are needed** before anything can be rendered.

Thus, they are **render-blocking** resources.

Users **see nothing** until both are on the client.

Get these over the wire **as quickly as we possibly can**.

Everything else can **load progressively** (images, video, audio, even JS).

CSS pretty much **is the Critical Path**.
Optimise the Critical Path.
Make it as short as possible.
Do not put anything unnecessary onto it.
Don’t make the Critical Path carry too much.
CSS Is Your Biggest Performance Bottleneck*
CSS Bottleneck

Browsers will **not render anything** until they have all the CSS for the current view/state.

Browsers **download all CSS**: even CSS that **doesn’t match** the current media type/query.
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, minimum-scale=1.0">
    <title>Test</title>
    <link rel="stylesheet" href="all.css">
    <link rel="stylesheet" href="screen.css" media="screen">
    <link rel="stylesheet" href="print.css" media="print">
    <link rel="stylesheet" href="bs.css" media="bs">
</head>
But they all get downloaded?!
tl;dr: Get CSS onto the Client ASAP
Smaller file sizes are better.
Keep things off of your Critical Path.
Make the Critical Path as short as you possibly can.
Just get CSS over the wire as fast as you possibly can.
Prioritise CSS above all else for first render.
Things We Should Do
The Three Cs.
The Three Cs

**Concatenate** CSS into as few files as possible to reduce HTTP requests.

**Compress** transfer over the wire using Gzip to reduce file sizes.

**Cache** stylesheets to prevent them being re-requested unnecessarily.
Remove Unused CSS
Remove Unused CSS

CSS file sizes generally shouldn’t be that problematic. But still, waste is waste. If it’s not used, try get rid of it.
Welcome to the BBC
Thursday, 14 January

Customise your Homepage

Actor Alan Rickman dies aged 69 after cancer battle
ENTERTAINMENT & ARTS

South Africa v England: Third Test, day one
CRICKET

Oscar nominations: The films and stars out in front
ENTERTAINMENT & ARTS

Web Page Performance
- Optimize the order of styles and scripts (23)
- Remove unused CSS rules (2229)
  2229 rules (75%) of CSS not used by the current page.
  - 40% is not used by the current page.
  - main.css: 0% is not used by the current page.
  - main.min.css: 90% is not used by the current page.
  - site.css: 80% is not used by the current page.
  - sb.min.css: 49% is not used by the current page.
  - share.css: 100% is not used by the current page.
- Use normal CSS property names instead of vendor-prefixed ones (3)
Remove Unused CSS

DevTools » Audits » Web Page Performance » Audit Present State » Run

UnCSS automates this (Grunt, Gulp, etc.).
Remove unused styles from CSS

XHmiko91 Update dependencies.
- bin
- src
- tests
- .jstrc
- .gitattributes
- .gitignore
- .travis.yml
- Gruntfile.js
- LICENSE.md
- README.md
- appveyor.yml
- package.json

UnCSS

npm v5.2.1
- Linux build: passing
- Windows build: passing
- coverage: 100%
- dependencies: out-of-date
- devDependencies: out-of-date
Inline Critical/Above the Fold CSS
A **huge** performance boost.

Send the (critical) **CSS back with the first response**.

Cut out **an entire round trip**—fewer HTTP requests.

**CSS arrives with HTML**, not after it.

Begin **rendering immediately**.

Lazy load the rest of the **CSS when we get chance**.

More on this **later**—sort of.
Google don’t send any separate CSS files.
They inline it all.
Important to lazy load the rest of the CSS and then cache it.
If it’s a single-page site then we don’t need to lazy load a CSS file.
Send it over with the HTML every time.
Things We Should Avoid
Do Not @import Stylesheets
Do Not `@import` Stylesheets

`@import` completely **kills performance**.

Causes **so much** extra work.

Adds **more round trips**.

**Delays** downloads.

**Increases time to render** dramatically.
The @import Process

**Client asks** for HTML.

**Server sends** HTML back.

**HTML asks** for CSS.

**Server sends** CSS back.

**CSS asks** for more CSS.

**Server sends** more CSS back.
Don’t let CSS request other CSS.
All requests for CSS should be sent out at the same time.
Basically, use multiple `<link />`s.
The `<link />` Process

- **Client asks** for HTML.
- **Server sends** HTML back.
- **HTML asks** for CSS and more CSS.
- **Server sends** CSS and more CSS back.
Working on a Sotheby’s site with a client.

Very complex mechanism for producing per-page CSS.

Complex Sass dependencies (anything can import anything).

Compiling Sass on the server as-per the CMS.

All in the pursuit of performance.

Then they were @importing a CSS file!
@import on Sotheby’s

All the hard work of **trying to create** the tiniest possible CSS file...
Was being **completely undone** by @importing their Google Fonts CSS.
Let me show you something crazy...
25kb? Nice!
Wait. What?

More CSS!!!
Why is that all the way down there?

Either lazy-loaded via JS, or...

It’s being @imported.
Whilst the browser was downloading, unpacking, parsing, and executing the first stylesheet, it had time to download 15 other assets. Then it was told to go off and get some more CSS. Which it had to download, unpack, parse, and execute. Then the browser could start rendering. @import kills performance by delaying everything.
Ideal fix: *concatenate* into one file*

Quick fix: serve via two `<link />` elements**
Avoid Base64 Encoding
Avoid Base64 Encoding

We’re told **Base64 reduces HTTP** requests—a good thing. But it actually creates something **much more expensive**...
Avoid Base64 Encoding

Base64 moves progressive assets onto your Critical Path.

Things that could have loaded as they were ready are now forced to download immediately, and on the Critical Path. Made the journey longer.

Making the browser do more work before it can render.
Avoid Base64 Encoding

Don’t put *non-essential assets* onto your Critical Path.

Pages *can render without images*.

You’re making the CSS heavier for unnecessary assets—let images *load in when ready*.

**Never put fonts** on the Critical Path—can add hundreds of KB.

All just *delays that first render*. 
Don’t Use an Asset Domain or CDN
Don’t Use an Asset Domain or CDN

Adds more work onto the Critical Path.
Makes the browser look up new DNS.
Can add 120ms before the CSS even starts downloading.
120ms that we don’t need to spend at all.
Don’t Use an Asset Domain or CDN

Always serve CSS from the host domain.
Host domain’s DNS is already warmed up—zero extra cost.
We can start sending CSS back immediately.
Asset Domains on m.skybet.com

Started off with **everything on host domain**.

**More requests** as site grew larger.

**Lack of parallelisation** was costing us.

Created a story for **moving assets onto asset domains** (stX.skybet.com).

Can now download **same amount of assets in less time**, right?
Asset Domains on m.skybet.com

Not quite.

Nightly WebPage Test showed us that performance had gotten worse!
Visually complete was almost ¼ second slower.
How on earth had this happened?!
Asset Domains on m.skybet.com

We’d started incurring **new DNS lookups**.

The **cost of going to a new domain was higher** than the cost of lack of parallelisation.

We moved our CSS **back onto the host domain**.

**Left other assets** on asset domains.

Performance **was better than ever**.
HTTP/2?
HTTP/2 Will Change Things

HTTP/2 brings in **new changes**.
Some will really **help performance**.
Requests get **cheaper**.
**Preemptively send assets** over the wire.
Employ **better caching** strategies.
Multiplexing
HTTP/1.1 is like buying a single item at a grocery store, taking it back home, **going back to the store for the next item you need, and repeating** until your pantry is fully stocked.

Multiplexing gives you a shopping cart so you can pick up **everything you need in one trip**.

—cloudflare.com/http2/what-is-http2
HTTP requests become cheaper.

More requests can be made per TCP connection.

Concatenation and spriting become unnecessary!

Practically, this means...
Better Caching Strategies
Better Caching Strategies

We can now split CSS into many different files. Split them based on rate-of-change. Only need to invalidate cache for smaller parts of your UI. Send your reset/Normalize.css styles in one file: cache it forever. Send your app styles in another that can change more frequently. Per-page CSS becomes marginally easier.
Hardly ever changes. Send as one file and cache for a long time.

Changes more frequently. Send as its own file and cache-bust it when needed.

```html
<link rel="stylesheet" href="core.css" />
<link rel="stylesheet" href="app.css" />
<link rel="stylesheet" href="video-player.css" />

Only need this on the video page. Becomes cheap to send as an extra request.
```
Server Push
Server Push

“Oh, you asked for an HTML page? Take this CSS as well—you’re gonna need it!”

Send assets to the browser before it’s even asked for them.

Remove round trips completely.

Like native inlined CSS (in a very roundabout way).
URL

index.html, 01.css, 02.css

Browser

Server
No longer waiting for the HTML to request the CSS. We can send the CSS back *in anticipation*. Saves us so much time—*no need to go back* to the server!
Lessons
Lessons

CSS is **expensive**.
Optimise your **Critical Path**.
Understand where all your **CSS is coming from**.
Avoid **Base64**.
Don’t use **@import**.
Avoid **CDNs/asset domains** for CSS.
Thank You
Thank You

Harry Roberts

csswizardry@gmail.com

@csswizardry