CSS and the Critical Path

Patrick Hamann
September 2013

Who am I?

- @patrickhamann
- Client-side developer
- CSS and performance geek

Who I work for

Making the next generation of the Guardian

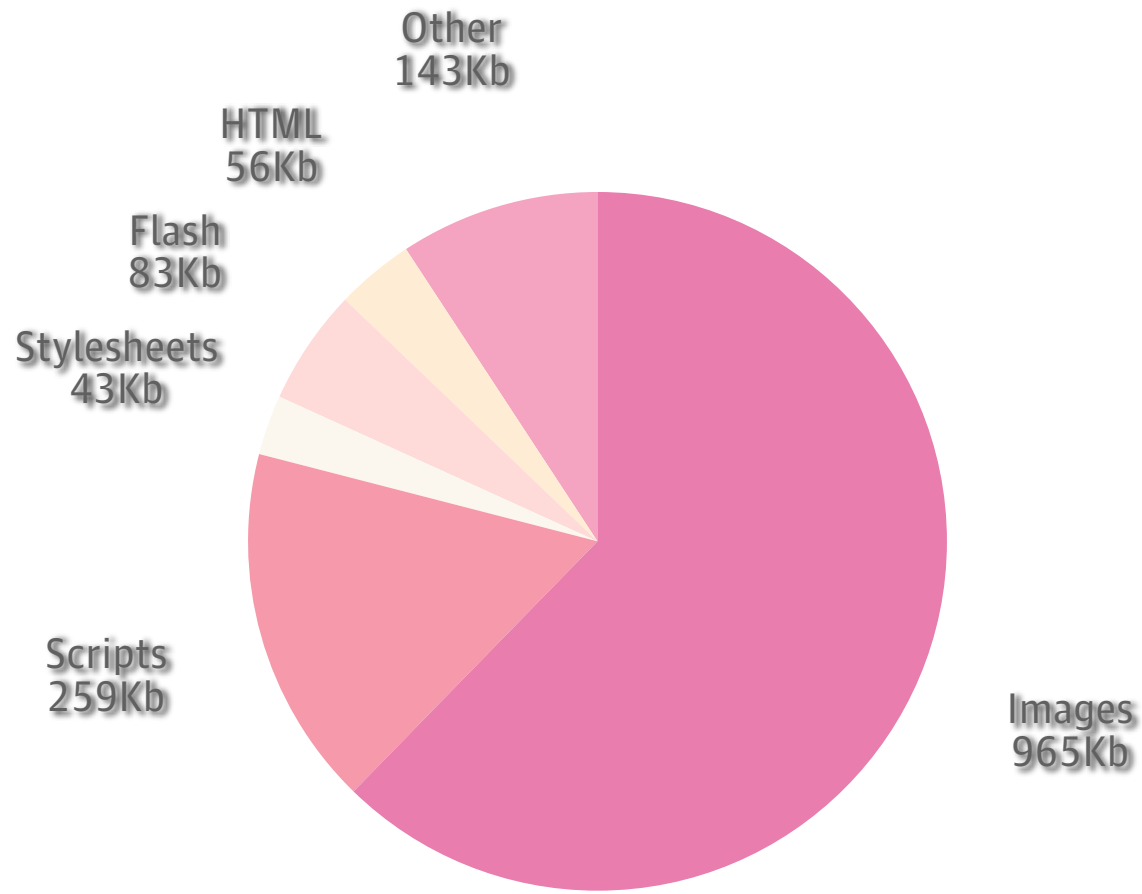
What I'm going to cover today

- The problem
- Browser rendering 101
- CSS and the critical path
- CSS loading techniques & real world examples
- The future
- Questions

Disclaimer

The problem

Page weight is increasing year on year

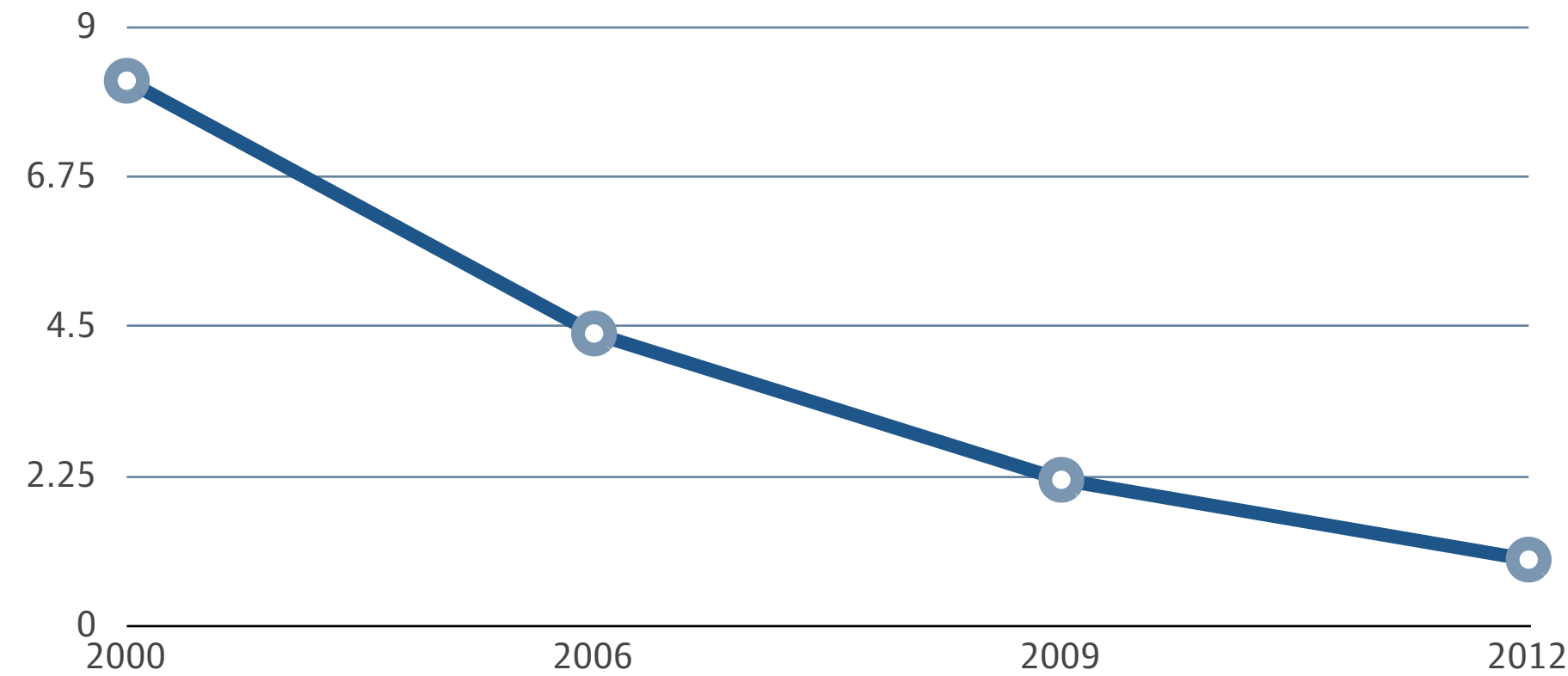


Source: HTTPArchive - Sept 2013

A new device landscape

http://www.flickr.com/photos/brad_frost/7387823392

User load time expectations are decreasing



Source: Web Performance Today - March 2013

Why performance matters: $$$

- Amazon found that it increased revenue by **1%** for every **100 milliseconds** of improvement.

- Shopzilla sped up its average page load time from 6 seconds to **1.2 seconds** and experienced a **12%** increase in revenue and a **25%** increase in page views.

- Mozilla shaved **2.2 seconds** off their landing pages, thereby increasing download conversions by **15.4%**, which they estimate will result in **60 million** more

Source: Strangeloop Networks case-studies

Guardian Maslow's hierarchy of user needs

- We asked **3000** digital news consumers how important 17 key product drivers were for them
- Consumers rated **speed** (fast page load time) as their **second most important** driver
- Only after easy to find well organised content
- The attribute that had the highest percentage of people saying it was **VERY important** to them

Time and user perception

Delay	User perception
0–100 ms	Instant
100–300 ms	Small perceptible delay
300–1000 ms	Machine is working
1,000+ ms	Likely mental context switch
10,000+ ms	Task is abandoned

Source: Ilya Grigorik – High-Performance Browser Networking

“For an application to feel instant, a perceptible response to user input must be provided within **hundreds of milliseconds**. After a second or more, the user’s flow and engagement with the initiated task is broken, and after 10 seconds have passed, unless progress feedback is provided, the task is frequently abandoned.

Ilya Grigorik
High-Performance Browser Networking

1000ms

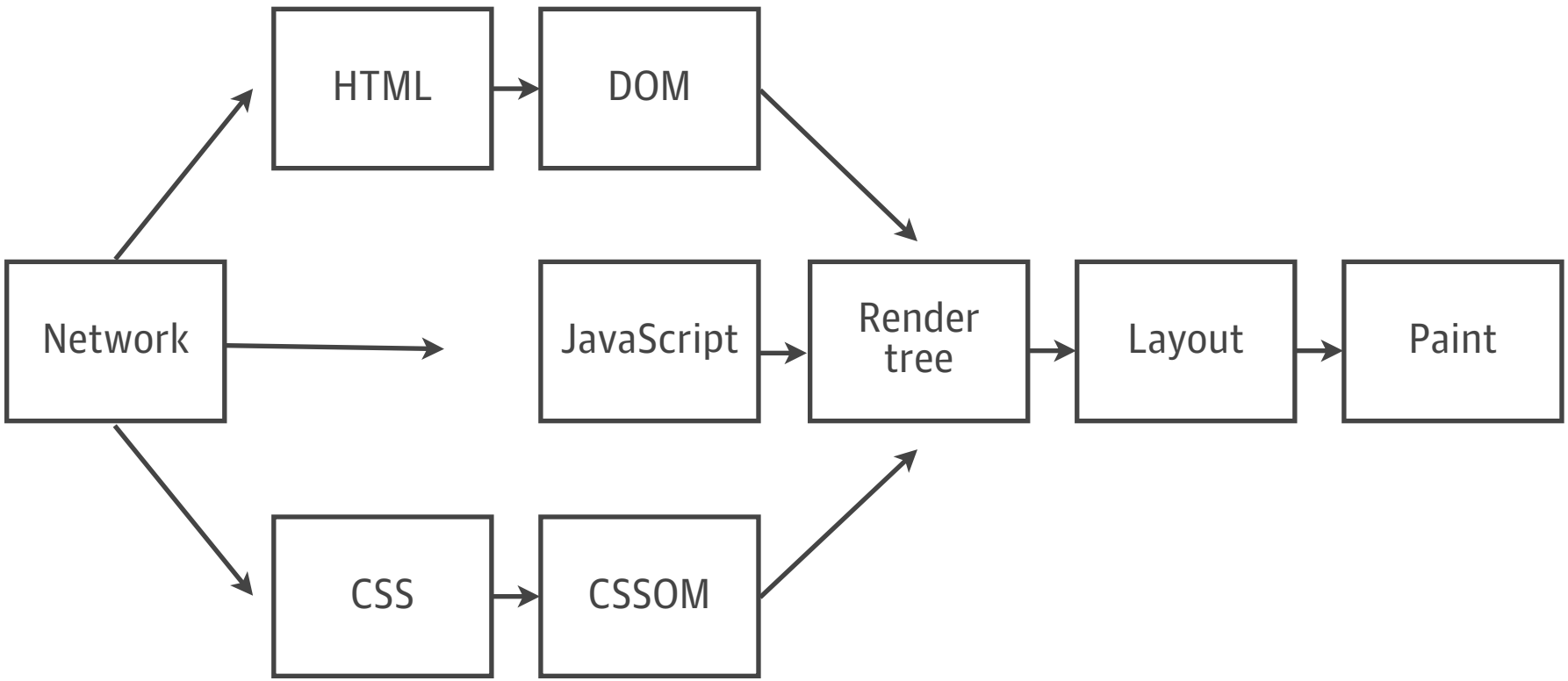
Browser rendering 101

“

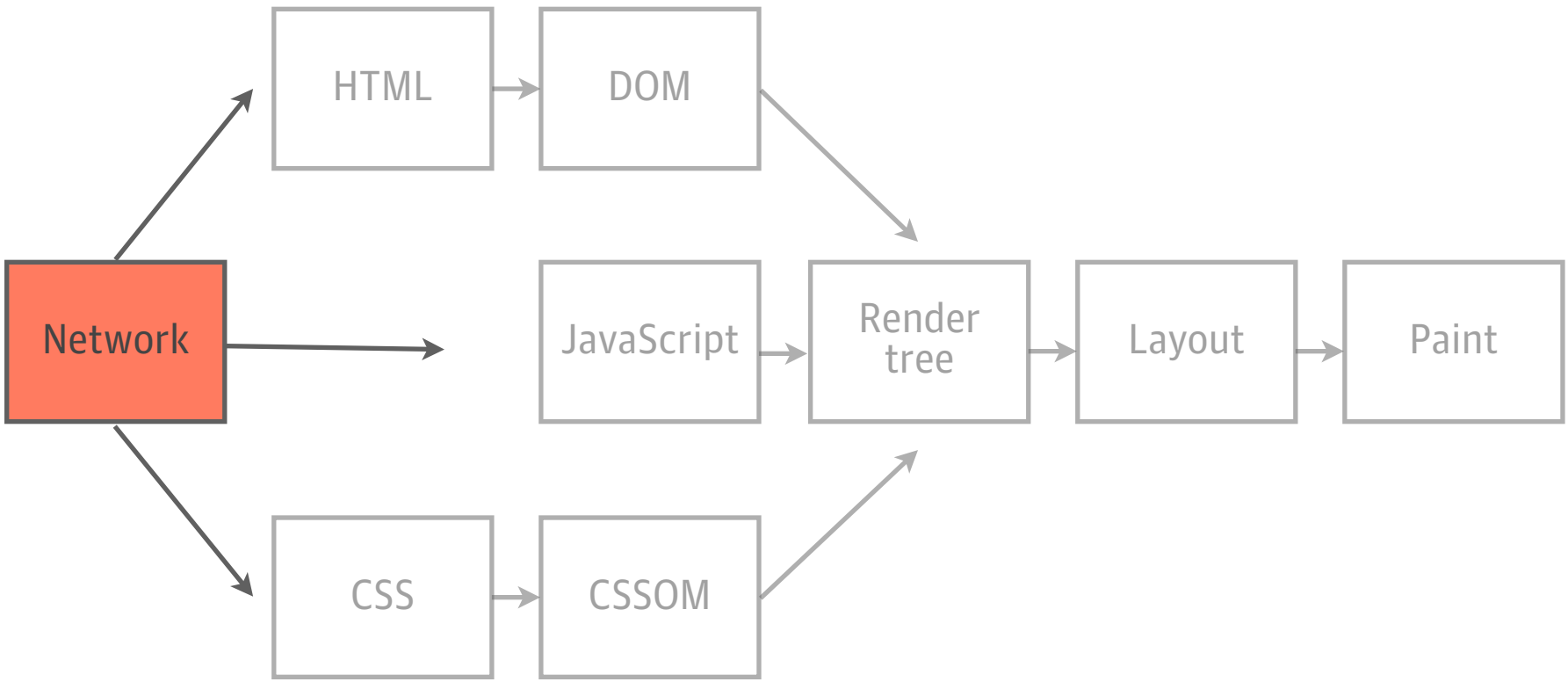
As a web developer, learning the internals of browser operations **helps you make better decisions** and know the justifications behind development best practices.

Paul Irish
Chrome Developer Relations

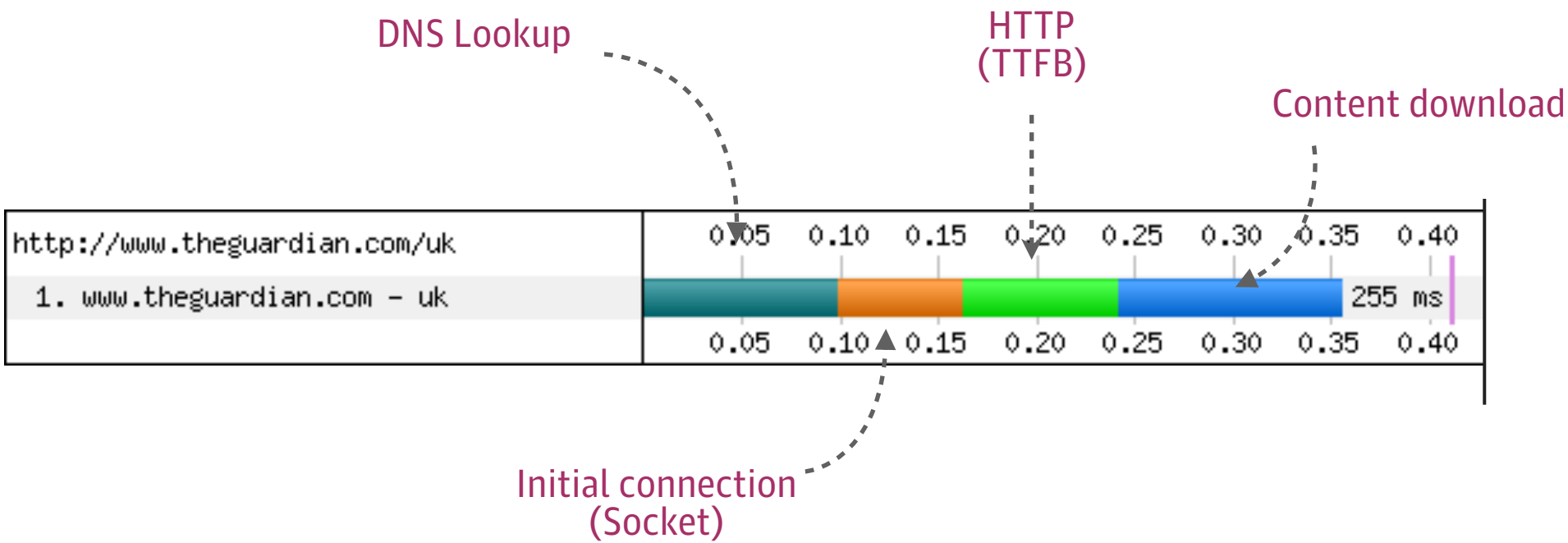
Rendering engine flow



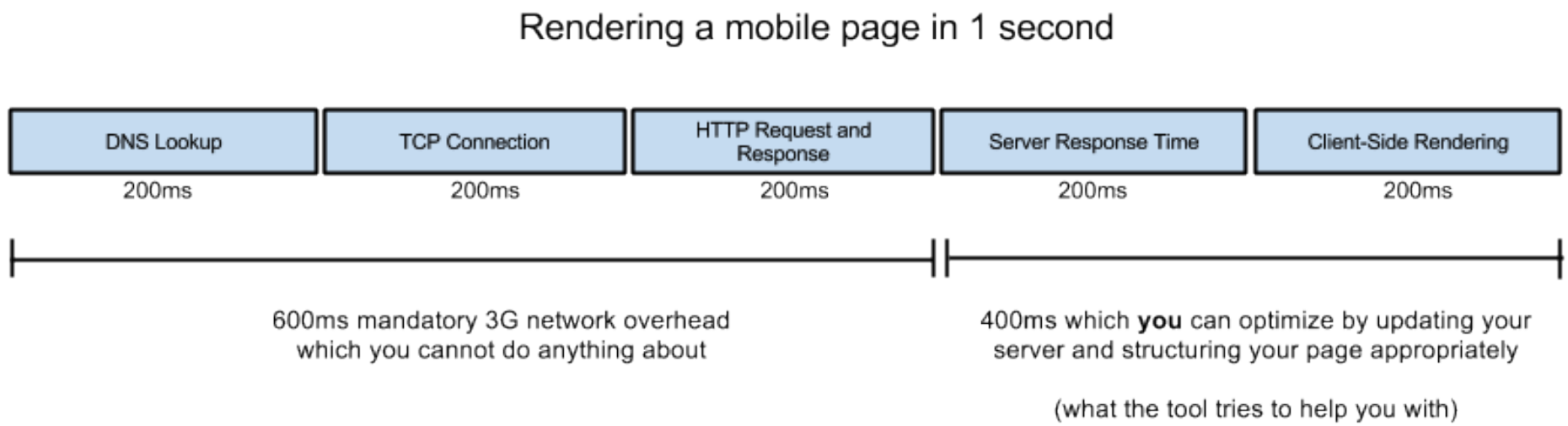
Network


HTML
DOM
Network
JavaScript
Render tree
Layout
Paint
CSS
CSSOM

Anatomy of a network request

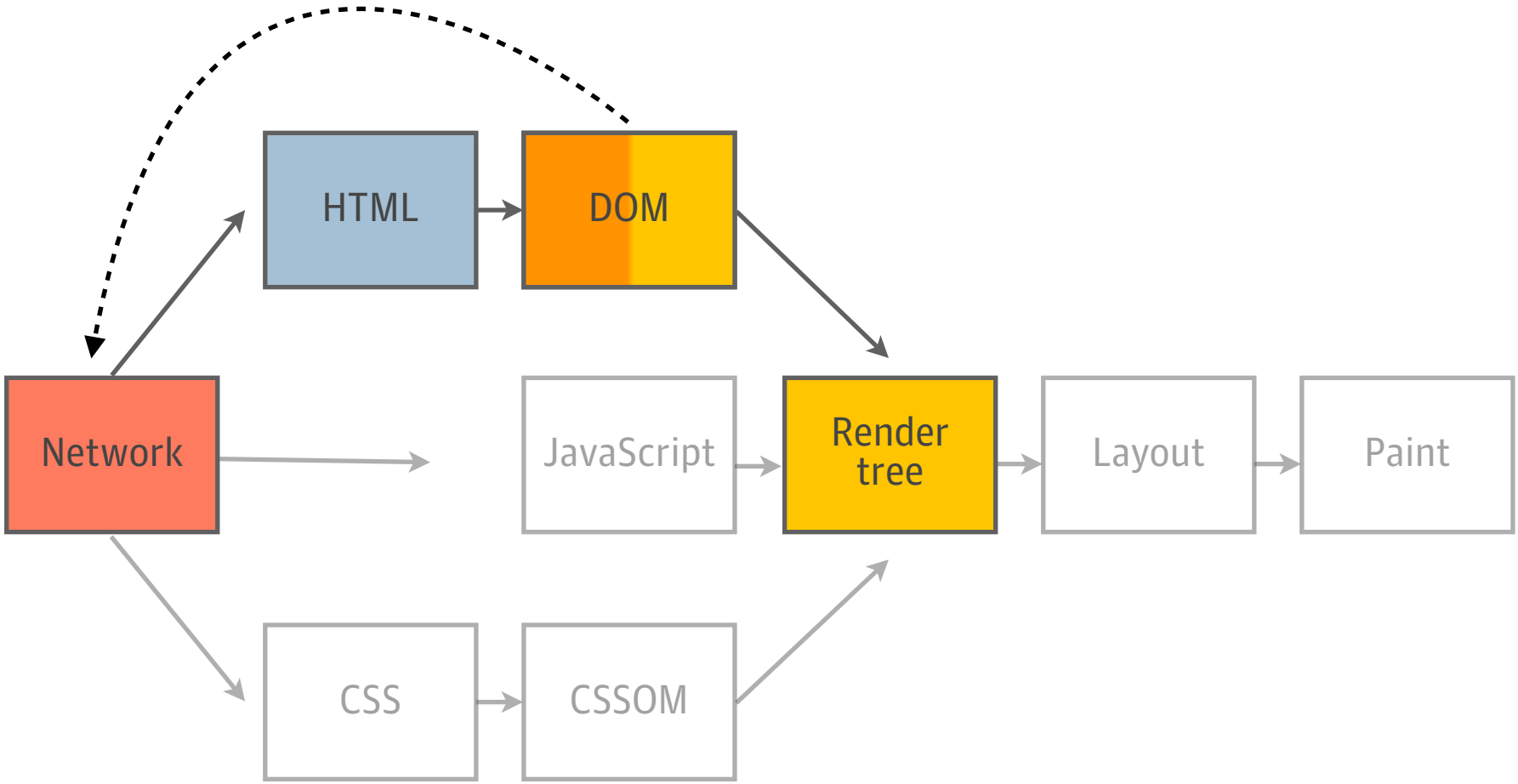


Anatomy of a network request

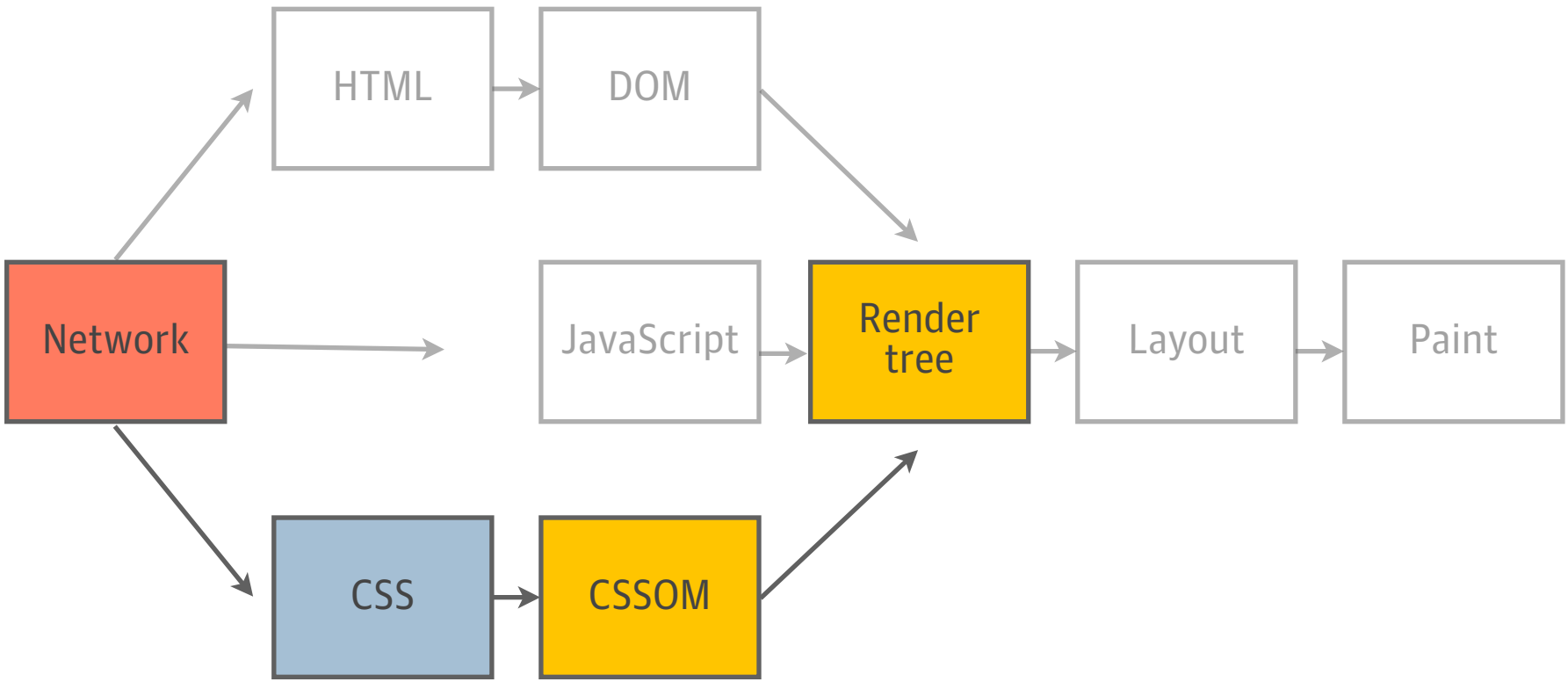


Source: Google - Pagespeed Insights

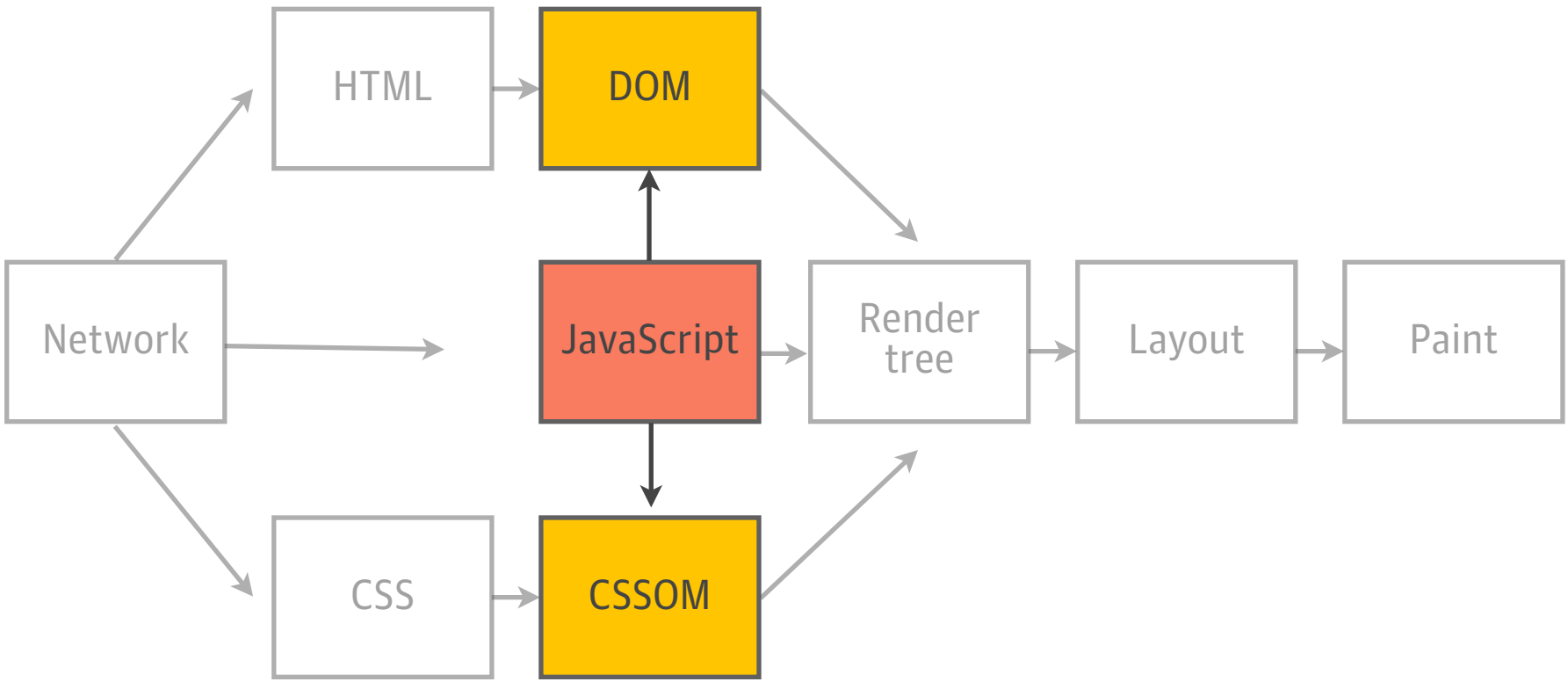
DOM: Document Object Model



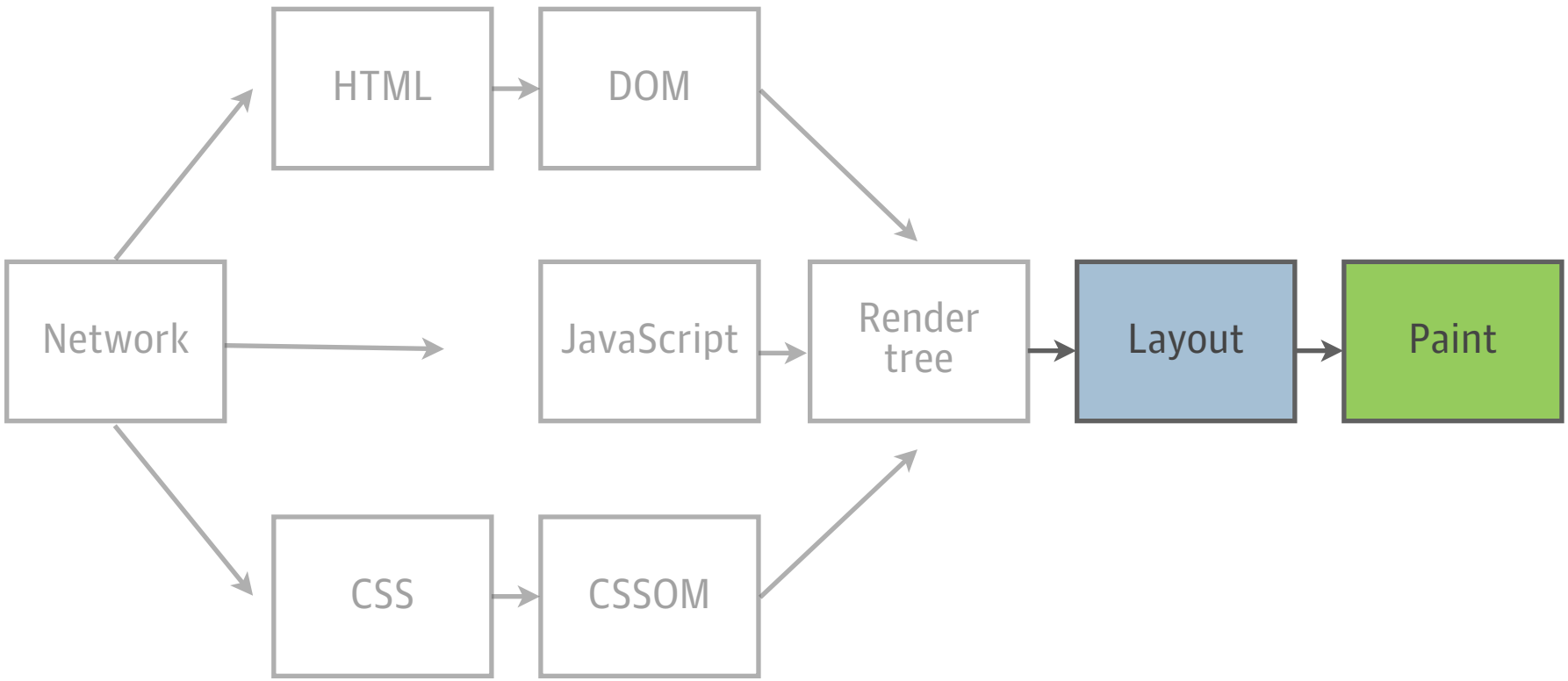
CSSOM: CSS Object Model



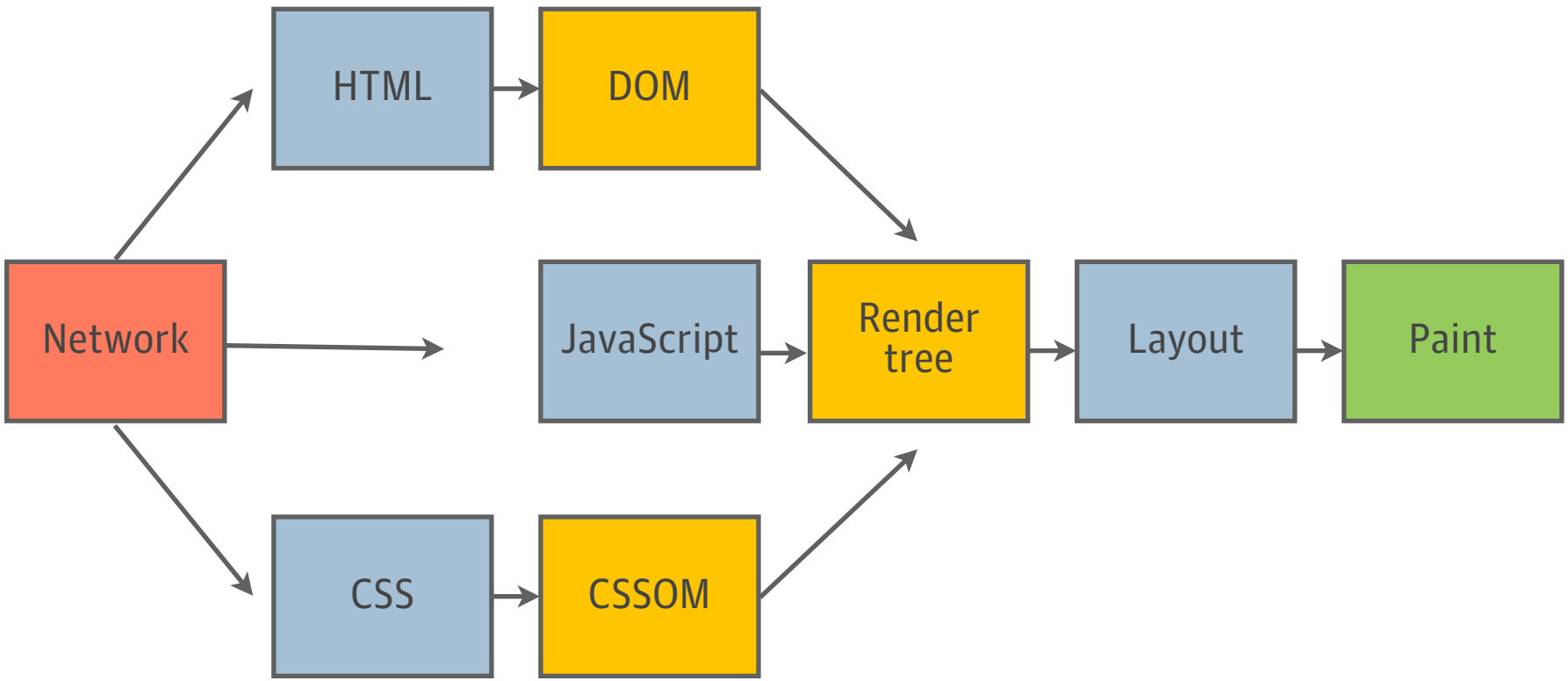
JavaScript our friend and foe



Rendering engine flow



Rendering engine flow



The Critical Path

“

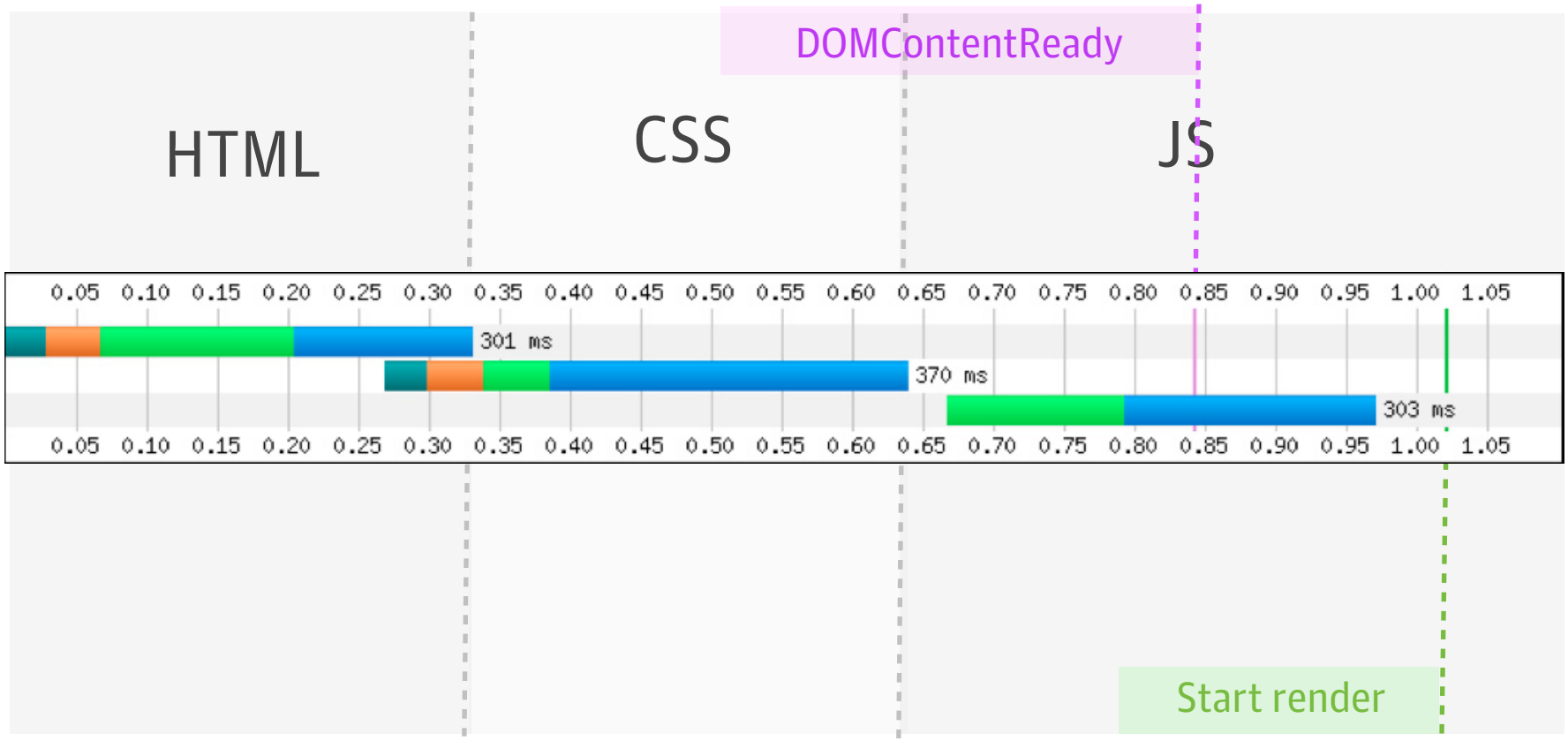
When building high-performance pages we want to stay off the critical path. **Critical** is the path from the user following a link to the first impression and then the working experience.

Stoyan Stefanov

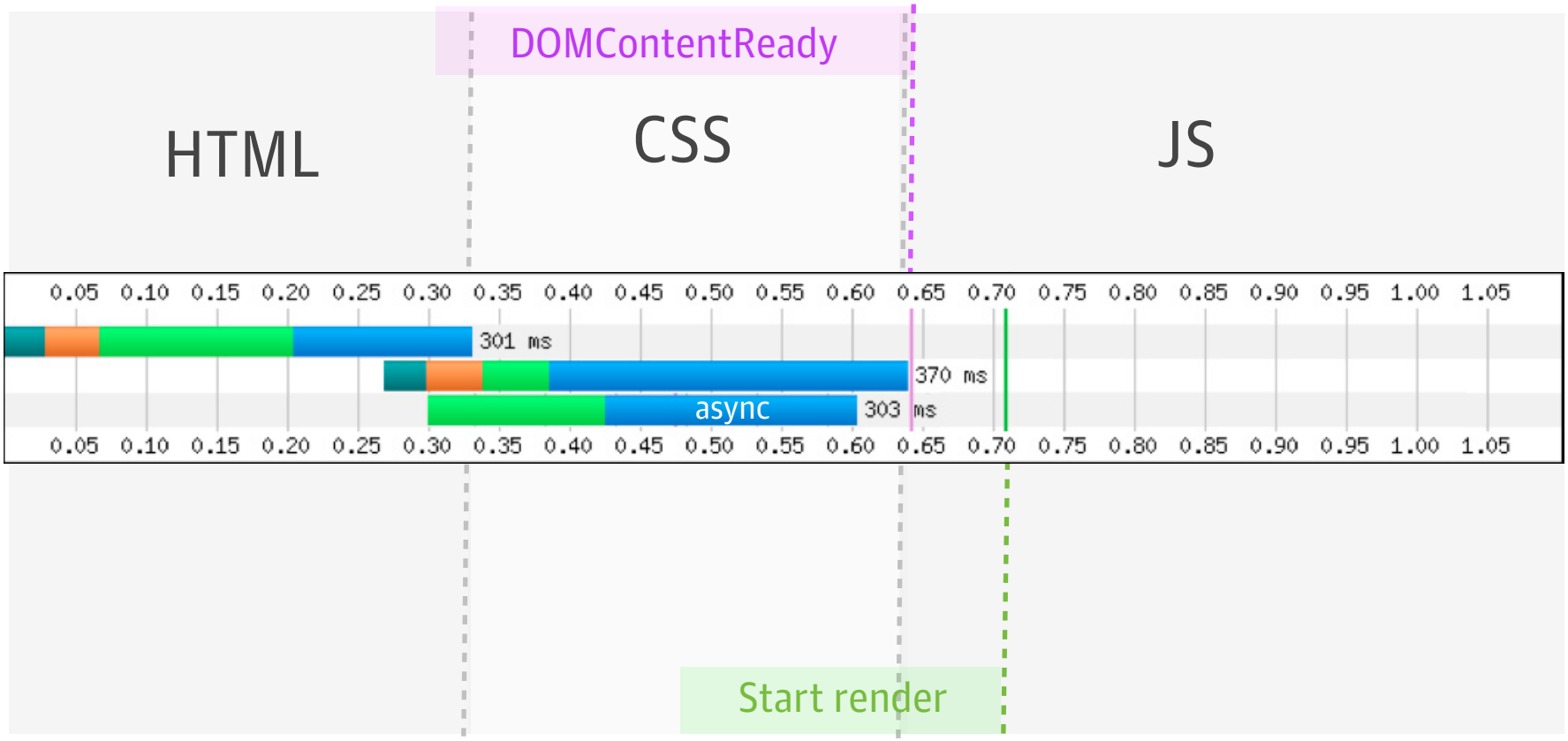
CSS and the critical path - 2012

g

The critical path - traditional waterfall



The critical path - traditional waterfall



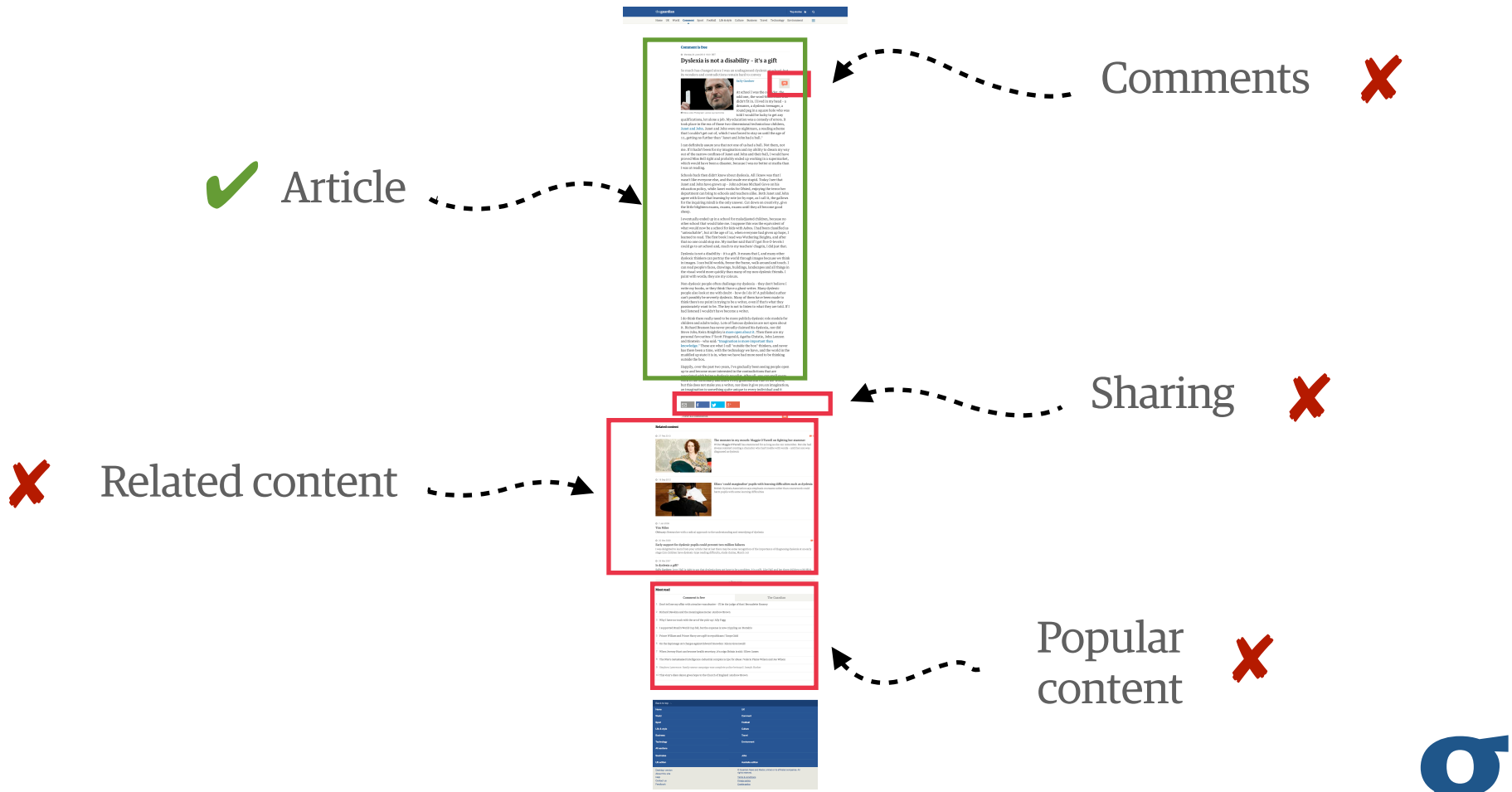
The critical path

- Latency on mobile is greatest barrier to 1000ms
- Core CSS matching current media blocks rendering
- Defer all non-critical assets, especially all JavaScript

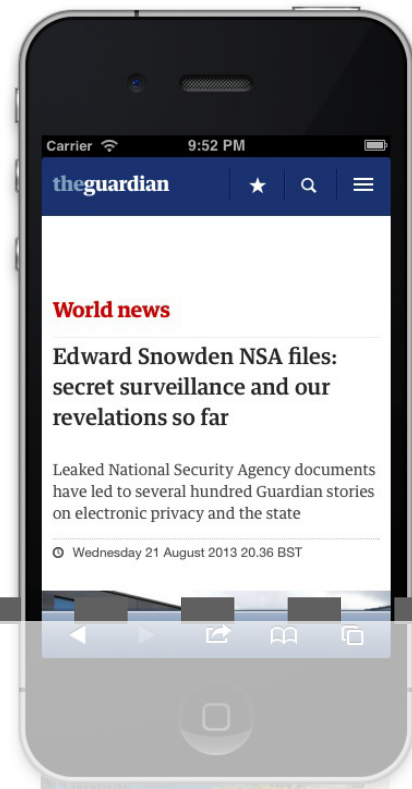
Get the CSS down as soon as possible

CSS and the critical path

What is your critical CSS?



What is your critical CSS?



Inline

g

Inline critical CSS in the <head>

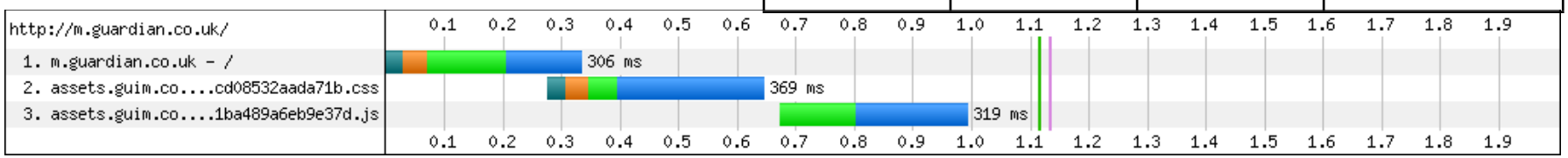
```
<!DOCTYPE html>
<html>
    <head>
        <title>The Guardian</title>

        <style type="text/css">
            /* Above the fold styles */
            h1 { color: blue; }
        </style>

    </head>
    <body>
        <h1>Academic criticise NSA and GCHQ for weakening online encryption</h1>
        ...

        /* Global stylesheets and JavaScript */
        <link rel="stylesheet" href="global.min.7885a401.css" />
        <script type="text/javascript" src="app.0708d36d8.js" defer></script>

    </body>
</html>
```

WTF a style element $%@!

Inline results

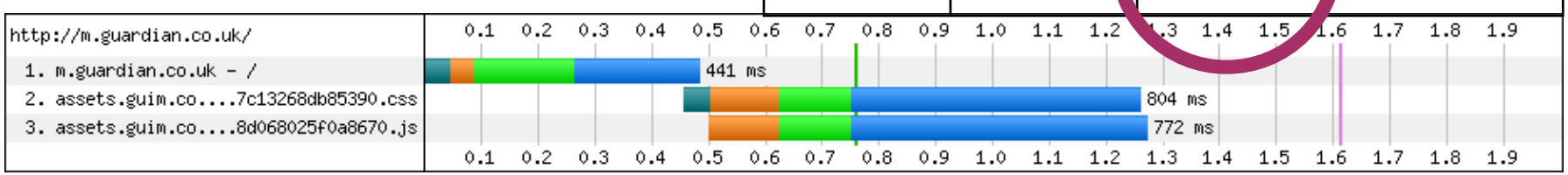
Before

Load time	First byte	Start render	Visually complete
3.366s	0.204s	1.113s	3.700s



After

Load time	First byte	Start render	Visually complete
3.491s	0.261s	0.759s	3.100s



Inline filmstrip

Inline pros

- Single HTTP request to view content
- Resilience
- Shave ~600-700ms off start render and DOMContentReady
- Browsers look ahead pre-parser still prioritises the CSS in the footer

Inline cons

- FOUC
- Having separate CSS can break the cascade
- "Above the fold" is a tricky concept on a responsive website
- With great power comes great responsibility
- Cache invalidation with CSS updates

LocalStorage

g

“

Bing and Google Search make extensive use of localStorage for stashing SCRIPT blocks that are used on subsequent page views. None of the other top sites from my previous post use localStorage in this way. **Are Bing and Google Search onto something?**

Yes, definitely

Steve Souders

Storager case study: Bing, Google

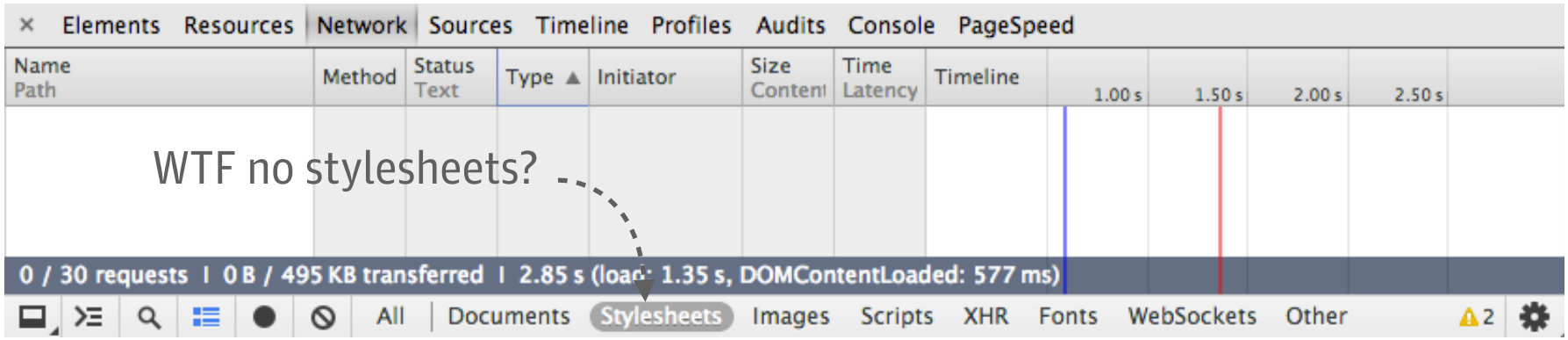
g

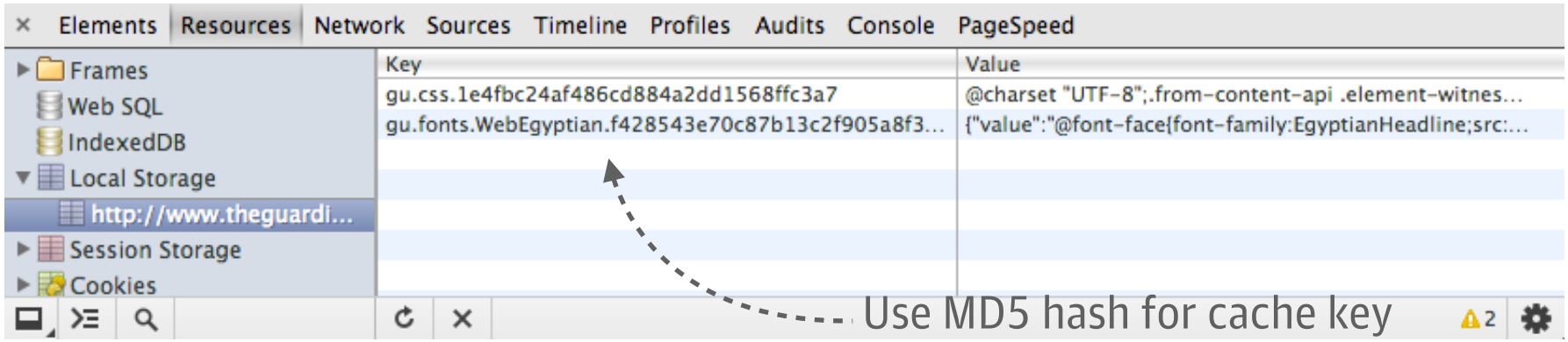
Load CSS from storage and inline

```
<!DOCTYPE html>
<html>
    <head>
        <title>The Guardian</title>

        <style type="text/css">
            /* Above the fold styles */
            h1 { color: blue; }
        </style>

        <script type="text/javascript">
            /* load cached styles */
            var css = localStorage.getItem("gu.css");
            if(css) {
                var style = document.createElement('style'),
                    script = document.getElementsByTagName('script')[0];
                style.innerHTML = css;
                script.parentNode.insertBefore(s, sc);
            }
        </script>

    </head>
    <body>
        <h1>Edward Snowden's not the story. The fate of the internet is</h1>
        ...
```

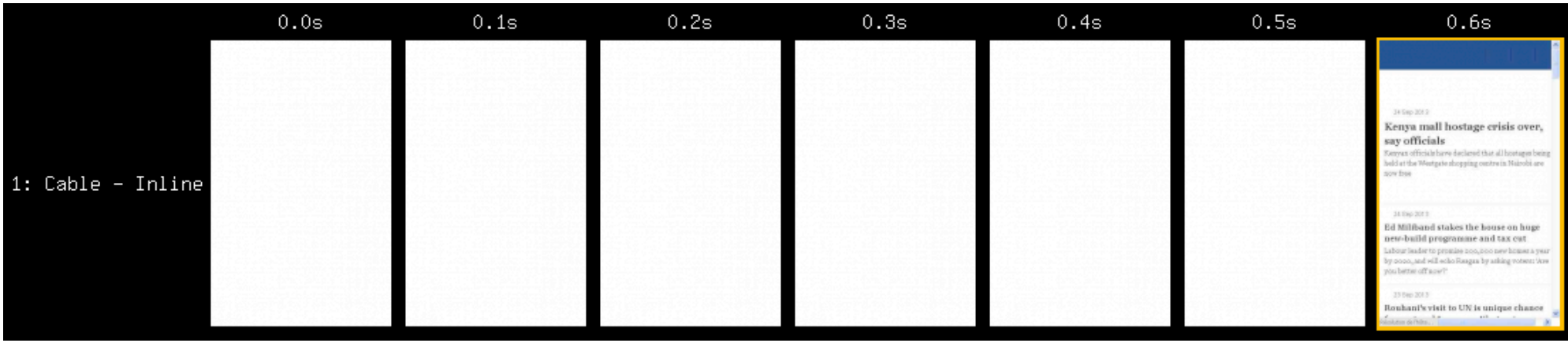
```
function storeCss(c) {
    Object.keys(localStorage ).forEach(function(key) {
        if(key.match(/^gu.css./g)) { localStorage.removeItem(key); }
    });
    try {
        localStorage.setItem('gu.css.@Static("stylesheets/global.min.css").md5Key', c);
    } catch(e) {

    }
}

function loadCssWithAjax() {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', '@Static("stylesheets/global.min.css")');
    xhr.onreadystatechange = function() {
        if (xhr.readyState === 4) {
            if(xhr.status === 200) {
                inlineCss(xhr.responseText);
                storeCss(xhr.responseText);
            } else {
                loadCssWithLink();
            };
        }
    }
    setTimeout( function () {
        if(xhr.readyState < 4) {
            xhr.abort();
            loadCssWithLink();
        }
    }, 5000);
    xhr.send();
}

if(guardian.isModernBrowser && !guardian.css.loaded) {
    loadCssWithAjax();
} else if(!guardian.isModernBrowser){
    loadCssWithLink();
}
```

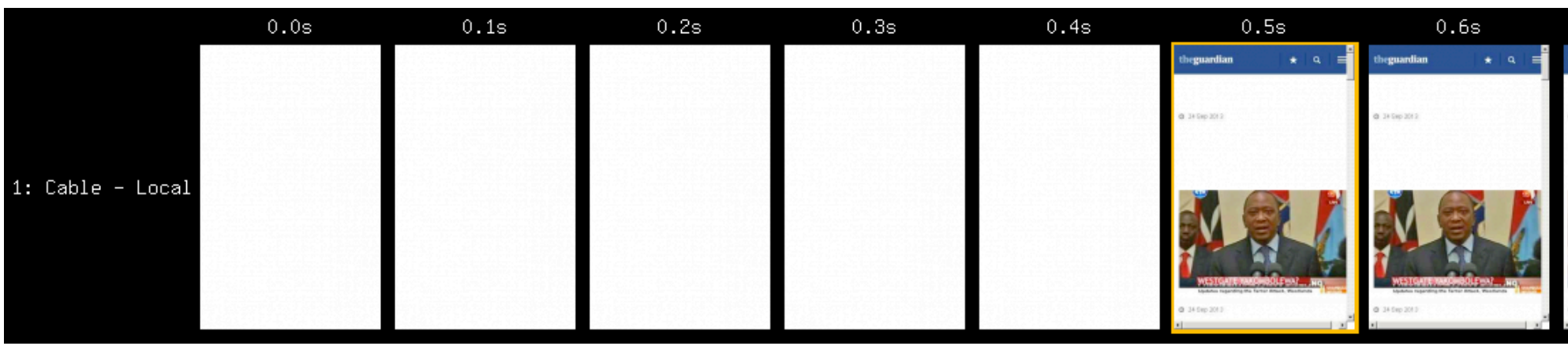
Look mum! No stylesheets





LocalStorage filmstrip





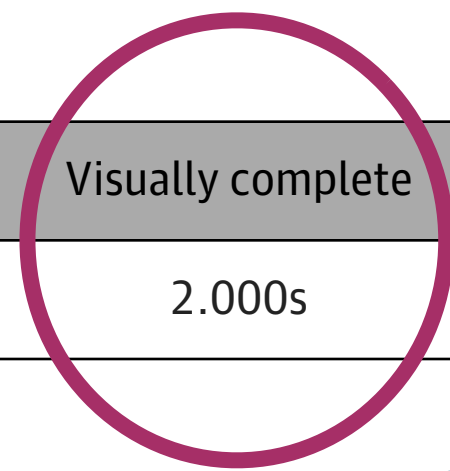
LocalStorage results

Before

Load time	First byte	Start render	Visually complete
1.307s	0.433s	0.466s	2.700s

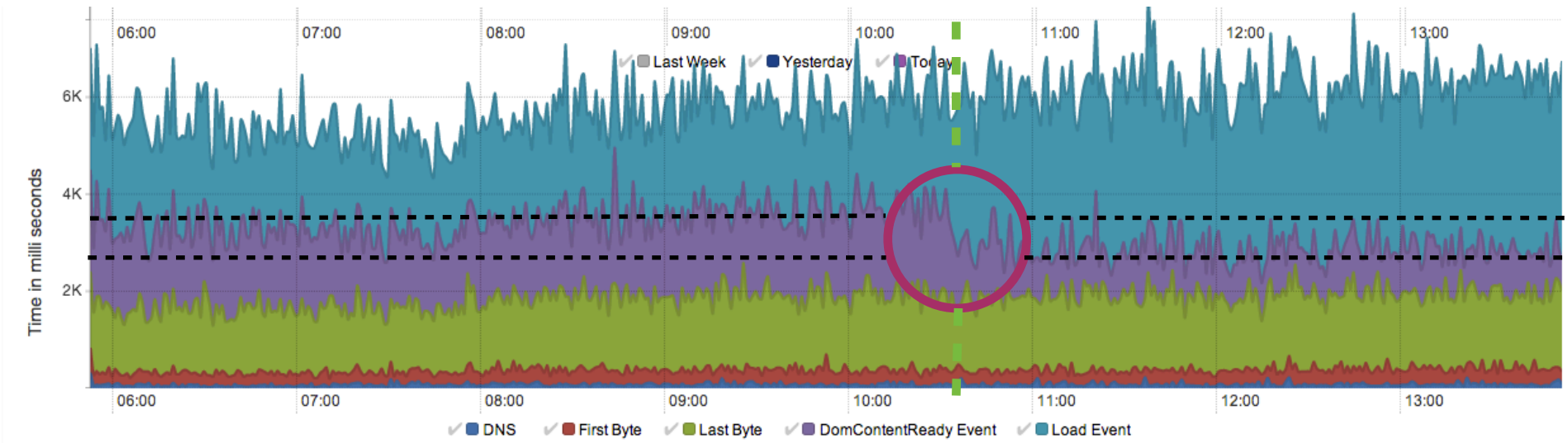
After

Load time	First byte	Start render	Visually complete
1.067s	0.426s	0.462s	2.000s



* Repeat view median run

Results from real user metrics



~600ms decrease in user DOMContentReady

Basket.js

- A simple (proof-of-concept) script loader that caches scripts with localStorage

Source: http://addyosmani.github.io/basket.js/

But...

g

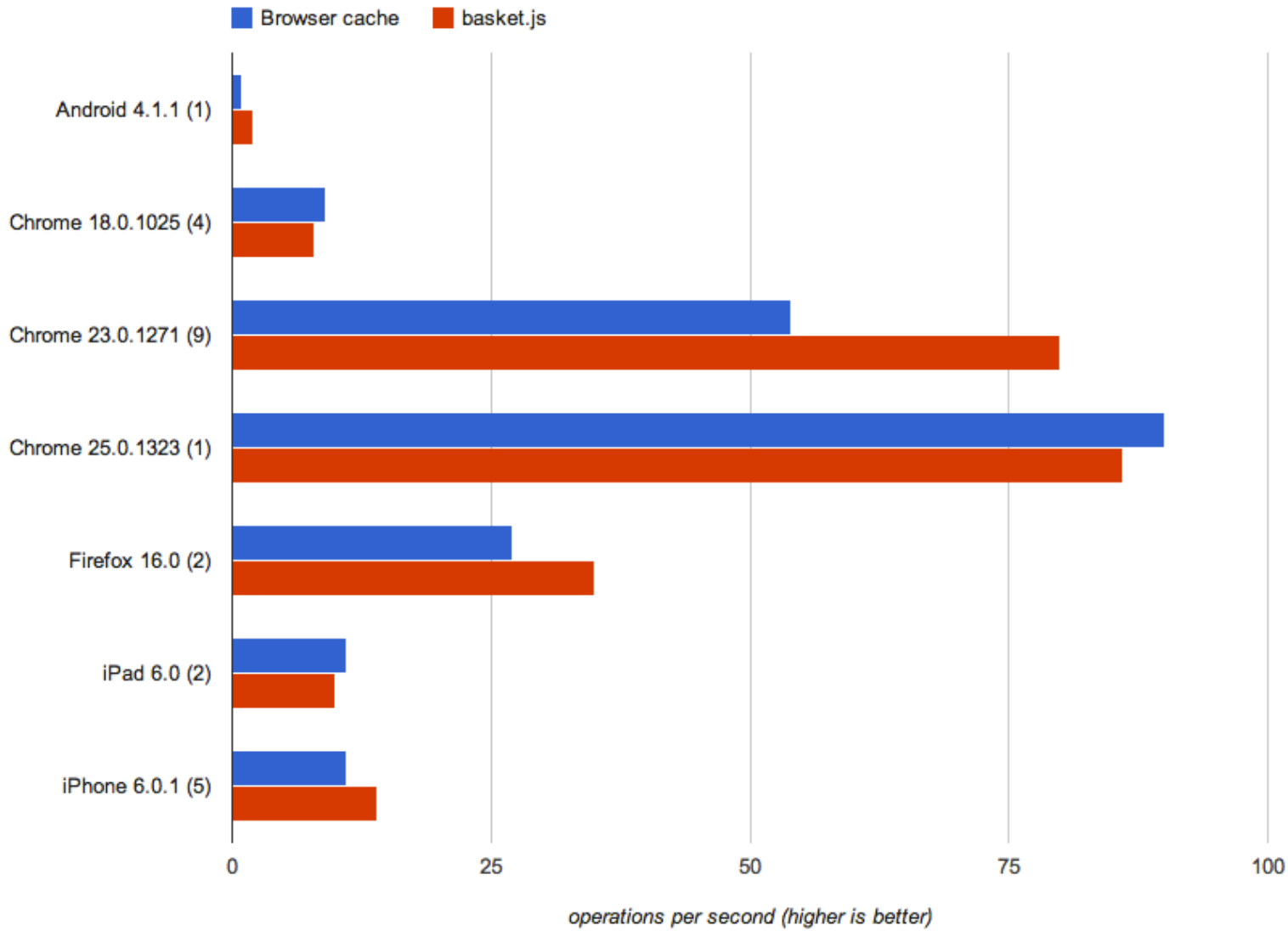
Speculative parsing

- DNS prefetching
- Preload scanner parses references to external resources

```
<!-- blocking stylesheet, nothing renders until it is downloaded and parsed -->
<link href="main.css" rel="stylesheet">

<!-- non-blocking, low download priority because of the evaluated media query -->
<link href="i-want-a-monitor-of-this-size.css" rel="stylesheet" media="(min-width: 4000px)">

<!-- won't be downloaded at all, because it is marked as disabled -->
<link href="noop.css" rel="stylesheet" disabled>

<!-- print stylesheet is non-blocking -->
<link href="noop.css" rel="stylesheet" media="print">
```

Source: Ilya Grigorik – Debunking responsive css performance myths

LocalStorage vs browser cache



Source: http://addyosmani.github.io/basket.js/

The Future

Resource Priorities API

- W3C draft spec
- Allows developers to programmatically give the User Agent hints on the download priority of a resource.
- Unblock non-critical assets

Yay!

```
<!DOCTYPE html>
<html>
 <head>
    <link rel="stylesheet" type="text/css" src="core.css" />
    <link rel="stylesheet" type="text/css" src="global.css" lazyload />
 </head>
```

g

Client hints

- IETF draft spec
- New “CH” client hint HTTP header field
- Device width, height and pixel density variables
- Cache friendly via Vary: CH

```
HTTP/1.1 200 OK
Content-Encoding: gzip
CH: dh=598, dw=384, dpr=2.0
Content-Type: text/html; charset=utf-8
Expires: Wed, 25 Sep 2013 18:36:37 GMT
```

SPDY/HTTP 2.0

- Final draft by end of 2014
- Each single connection can carry any number of bidirectional streams, thus allowing multiplexing
- One connection per origin
- Request prioritisation
- Lowers page load times by eliminating unnecessary latency

RESS

- Responsive Design + Server Side Components
- Use client hints to make decision on which CSS to serve

```
<style type="text/css">
    /* Inline above the fold styles */
</style>

@if(isWideDevice) {
    <link href="desktop.css" rel="stylesheet"
        media="screen and (min-width: 600px)" />
}
```

Takeaways

- Users do care about speed
- Inline critical (above the fold) CSS
- Defer all other non-critical assets to avoid blocking of render tree
- Where possible store downloaded assets in localStorage

“

Performance first.

Thank you!

We're hiring!

@patrickhamann
http://github.com/guardian/frontend
September 2013