Building *Faster* Mobile Websites

the nuts and bolts of hitting the ***1000 millisecond*** *"time to glass" target ...*

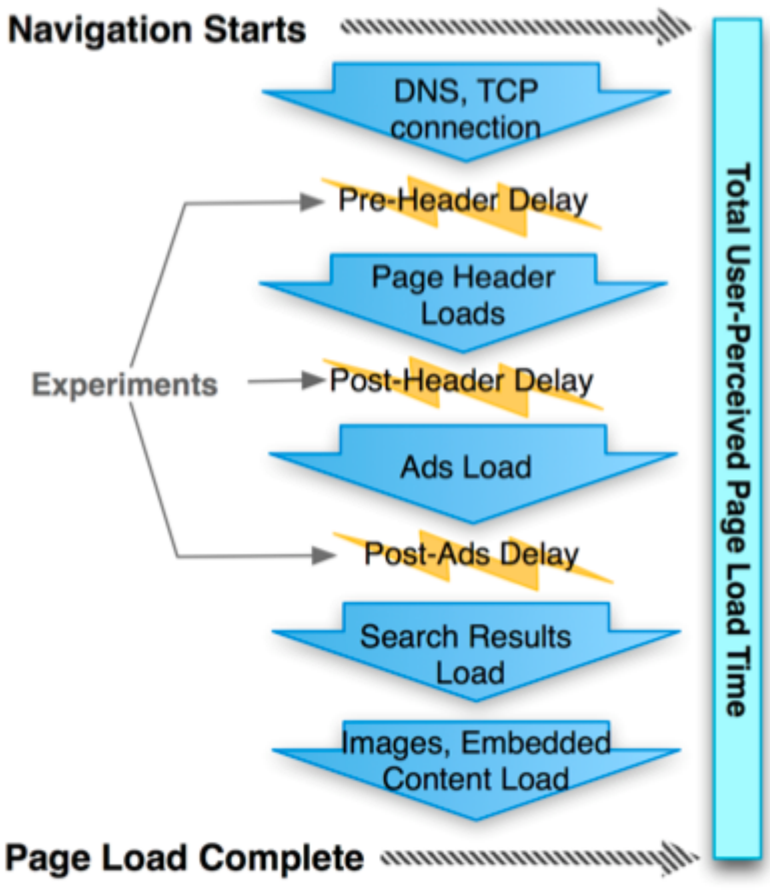
+Ilya Grigorik **@igrigorik**
Make The Web Faster, Google

Video of the talk: http://bit.ly/12GFKDE

What's the impact of slow sites?

Lower conversions and engagement, higher bounce rates...

Web Search Delay Experiment



Type of Delay	Delay (ms)	Duration (weeks)	Impact on Avg. Daily Searches
Pre-header	50	4	Not measurable
Pre-header	100	4	**-0.20%**
Post-header	200	6	**-0.59%**
Post-header	400	6	**-0.59%**
Post-ads	200	4	**-0.30%**

- The cost of delay increases over time and persists
- Delays under half a second impact business metrics
- "Speed matters" is not just lip service

Performance Related Changes and their User Impact

bing Server Delays Experiment

	Distinct Queries/User	Query Refinement	Revenue/User	Any Clicks	Satisfaction	Time to Click (increase in ms)
50ms	-	-	-	-	-	-
200ms	-	-	-	-0.3%	-0.4%	500
500ms	-	-0.6%	-1.2%	-1.0%	-0.9%	1200
1000ms	-0.7%	-0.9%	-2.8%	-1.9%	-1.6%	1900
2000ms	-1.8%	-2.1%	-4.3%	-4.4%	-3.8%	3100

- Means no statistically significant change

- Strong negative impacts
- Roughly linear changes with increasing delay
- Time to Click changed by roughly double the delay

[Performance Related Changes and their User Impact]

How speed affects bounce rate

$y = 0.6517x + 33.682$

$R^2 = 0.91103$

Error pages

Every second = 0.65 increase in bounce rate

So, how are we doing today?

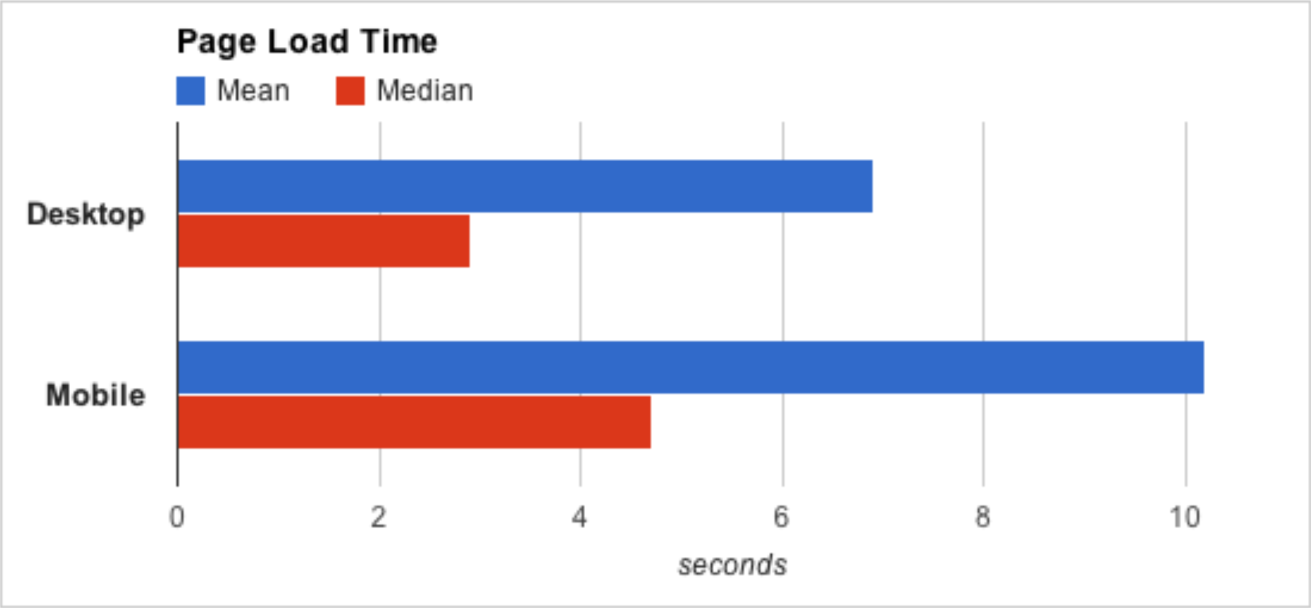
Okay, I get it, speed matters... but, are we there yet?

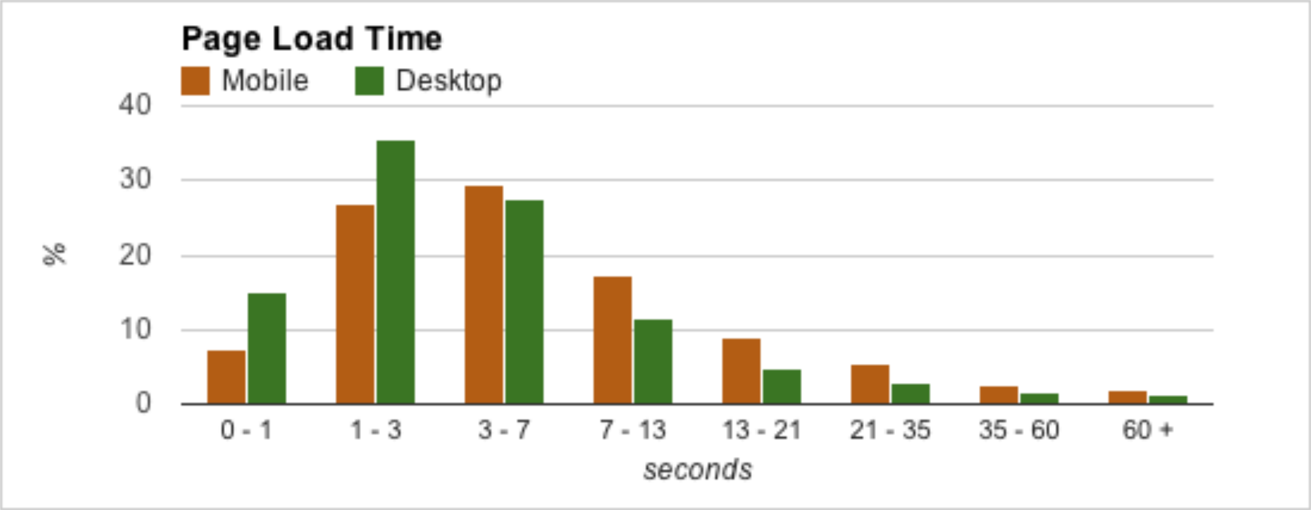
Usability Engineering 101

Delay	User reaction
0 - 100 ms	Instant
100 - 300 ms	*Feels sluggish*
300 - 1000 ms	Machine is working...
1 s+	Mental context switch
10 s+	I'll come back later...

Stay under 250 ms to feel "fast".

Stay under 1000 ms to keep users attention.





Desktop

Median: ~2.7s

Mean: ~6.9s

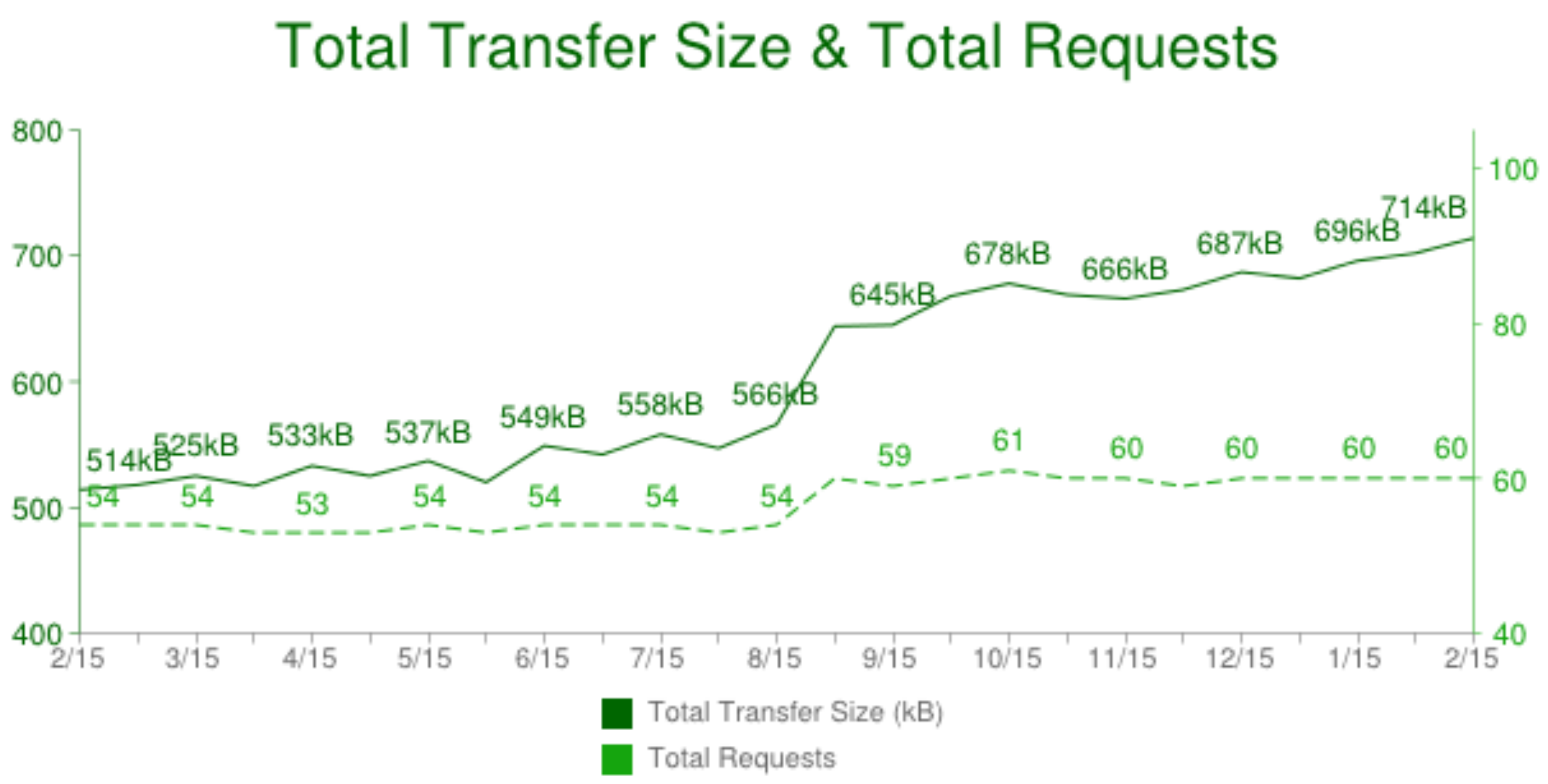
Mobile *

Median: ~4.8s

Mean: ~10.2s

* *optimistic*

How Fast Are Websites Around The World? - Google Analytics Blog



Content Type	Avg # of Requests	Avg size
HTML	6	39 kB
Images	**39**	**490 kB**
Javascript	**10**	**142 kB**
CSS	3	27 kB

HTTP Archive - Mobile Trends (Feb, 2013)

For many, mobile is the one and only internet device!

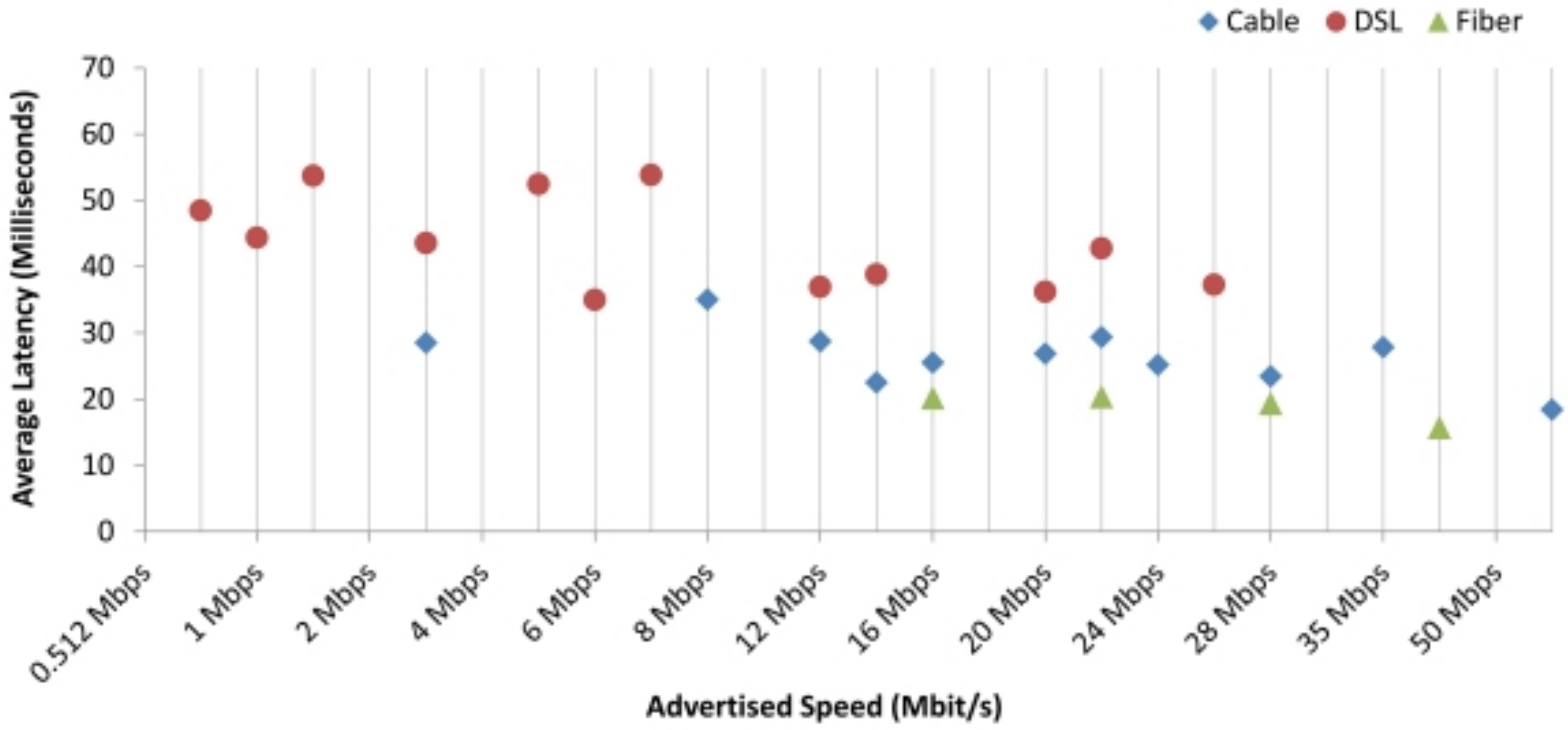
Country	Mobile-only users
Egypt	70%
India	59%
South Africa	57%
Indonesia	44%
United States	25%

onDevice Research

The network will save us!

1000 ms is plenty of time.. 4G will fix everything! Right, right?

* Nope.



***Fiber-to-the-home** services provided **18 ms** round-trip latency on average, while **cable-based** services averaged **26 ms**, and **DSL-based** services averaged **43 ms**. This compares to 2011 figures of 17 ms for fiber, 28 ms for cable and 44 ms for DSL.*

Mobile, oh Mobile...

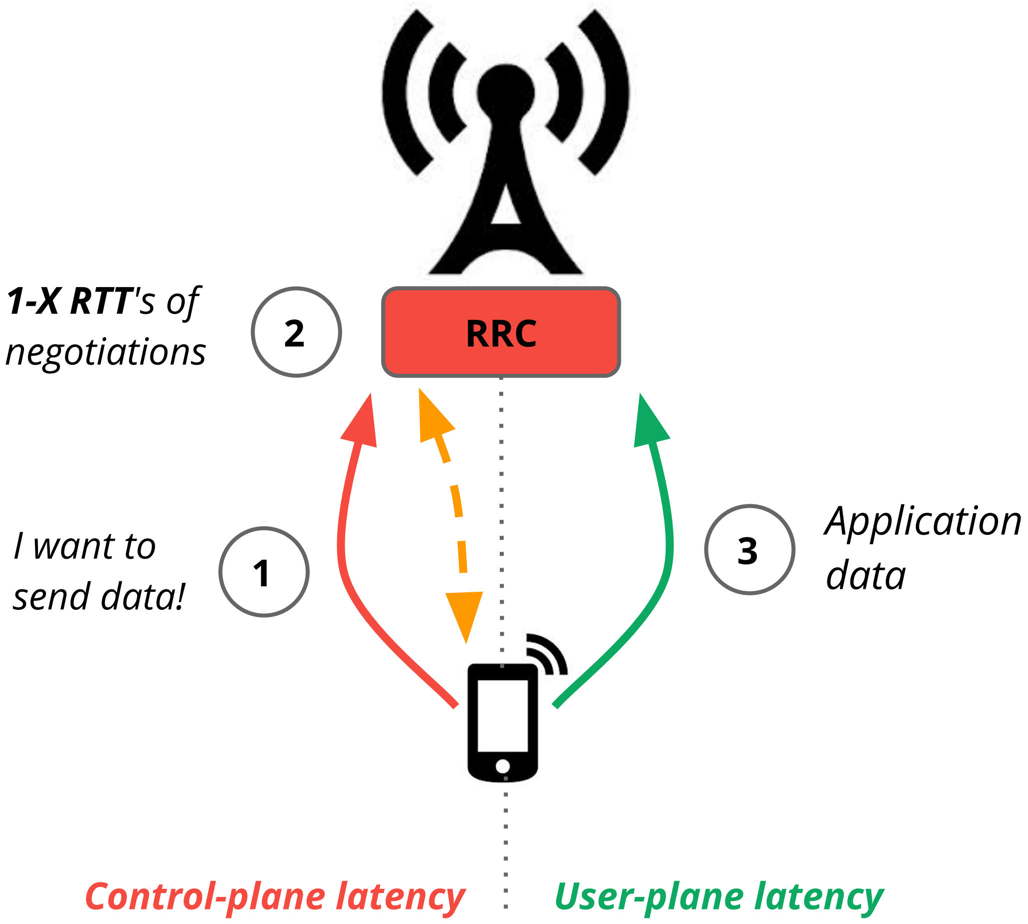
*"Users of the **Sprint 4G network** can expect to experience average speeds of 3 Mbps to 6 Mbps download and up to 1.5 Mbps upload with an **average latency of 150 ms**. On the **Sprint 3G** network, users can expect to experience average speeds of 600 Kbps - 1.4 Mbps download and 350 Kbps - 500 Kbps upload with an **average latency of 400 ms**."*

	3G	4G
Sprint	400 ms	150 ms
AT&T	150 - 400 ms	100 - 200 ms

Mobile design constraint: Battery life

- Radio is the **second most expensive** component (after screen)
- Limited amount of available power (as you well know...)

Control and User plane latencies

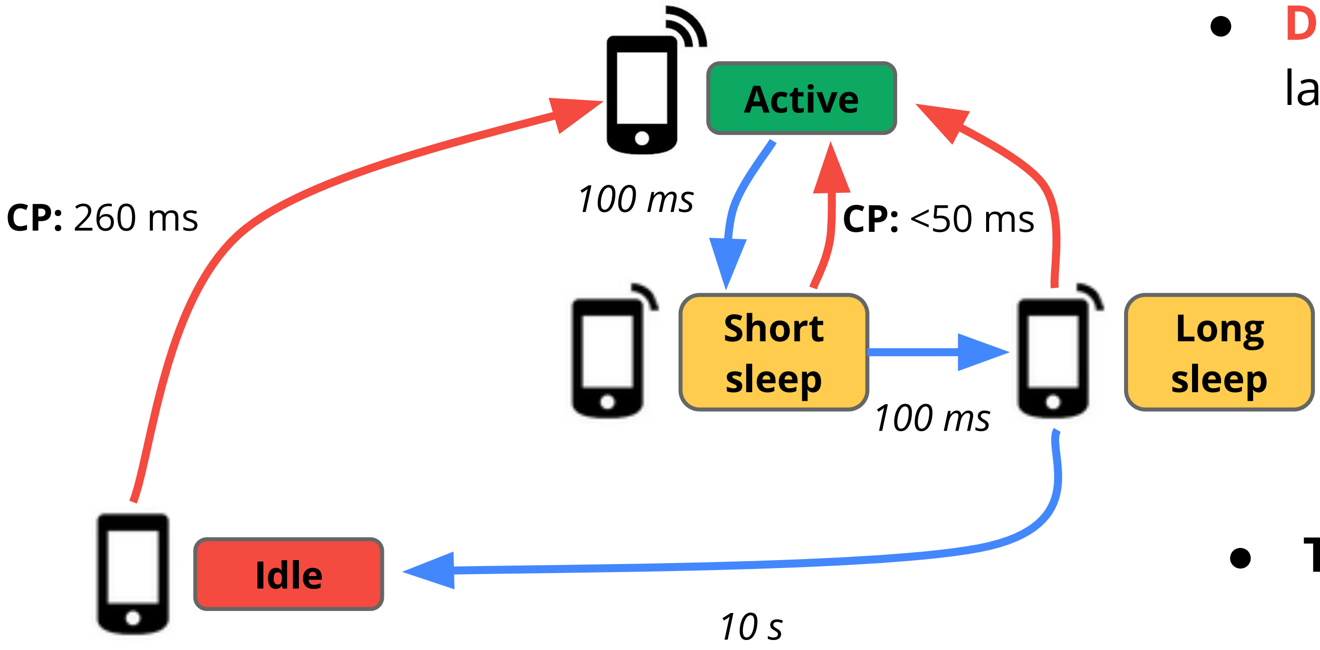


- There is a **one time** cost for control-plane negotiation
- **User-plane latency** is the one-way latency between packet availability in the device and packet at the base station

	LTE	HSPA+	3G
Idle to connected latency	< 100 ms	< 100 ms	**< 2.5 s**
User-plane one-way latency	< 5 ms	< 10 ms	< 50 ms

Same process happens for incoming data, just reverse steps 1 and 2

LTE power state transitions (AT&T)

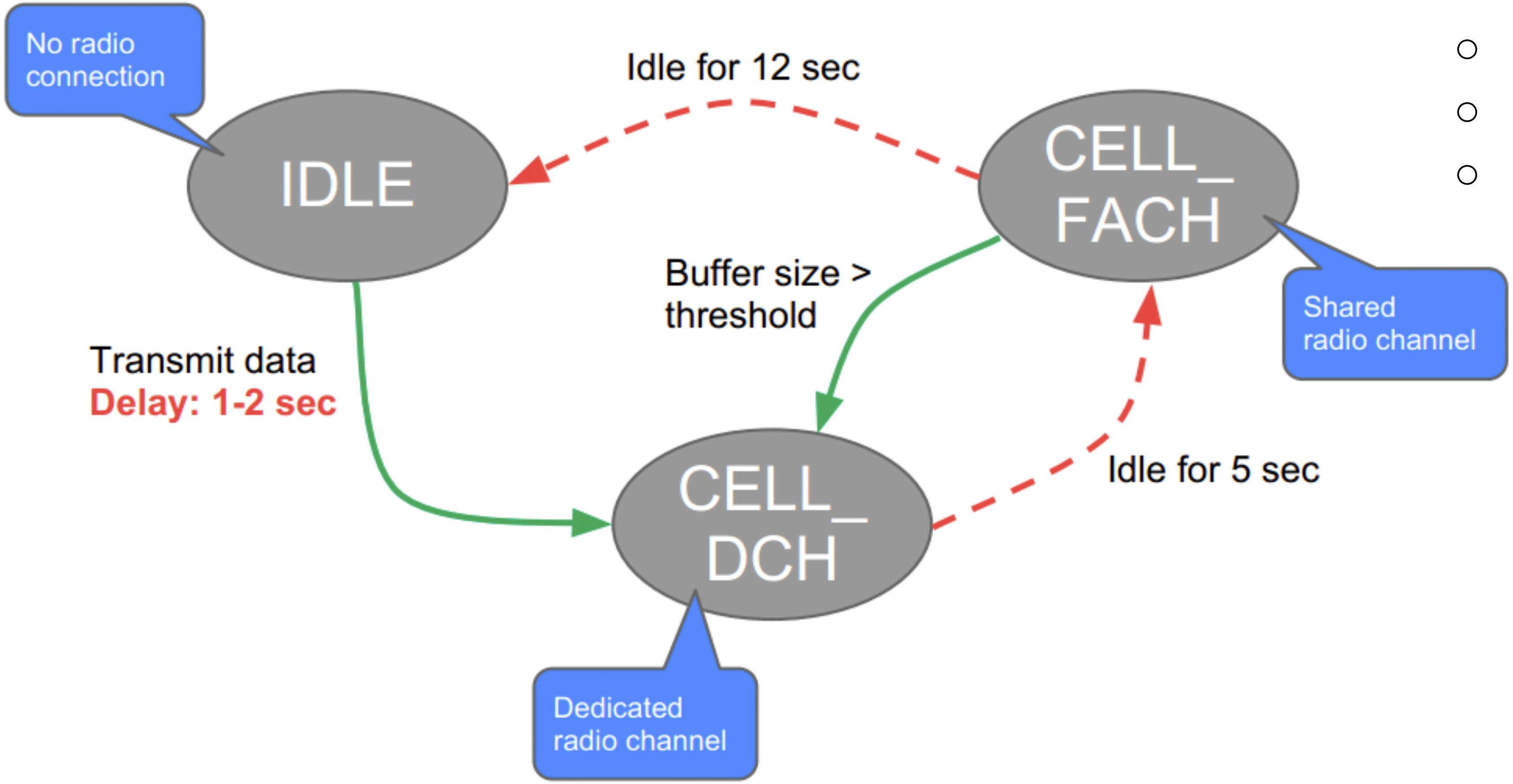


- **Idle to Active:** 260 ms control-plane latency
- **Dormant to Active:** <50 ms control-plane latency (spec)

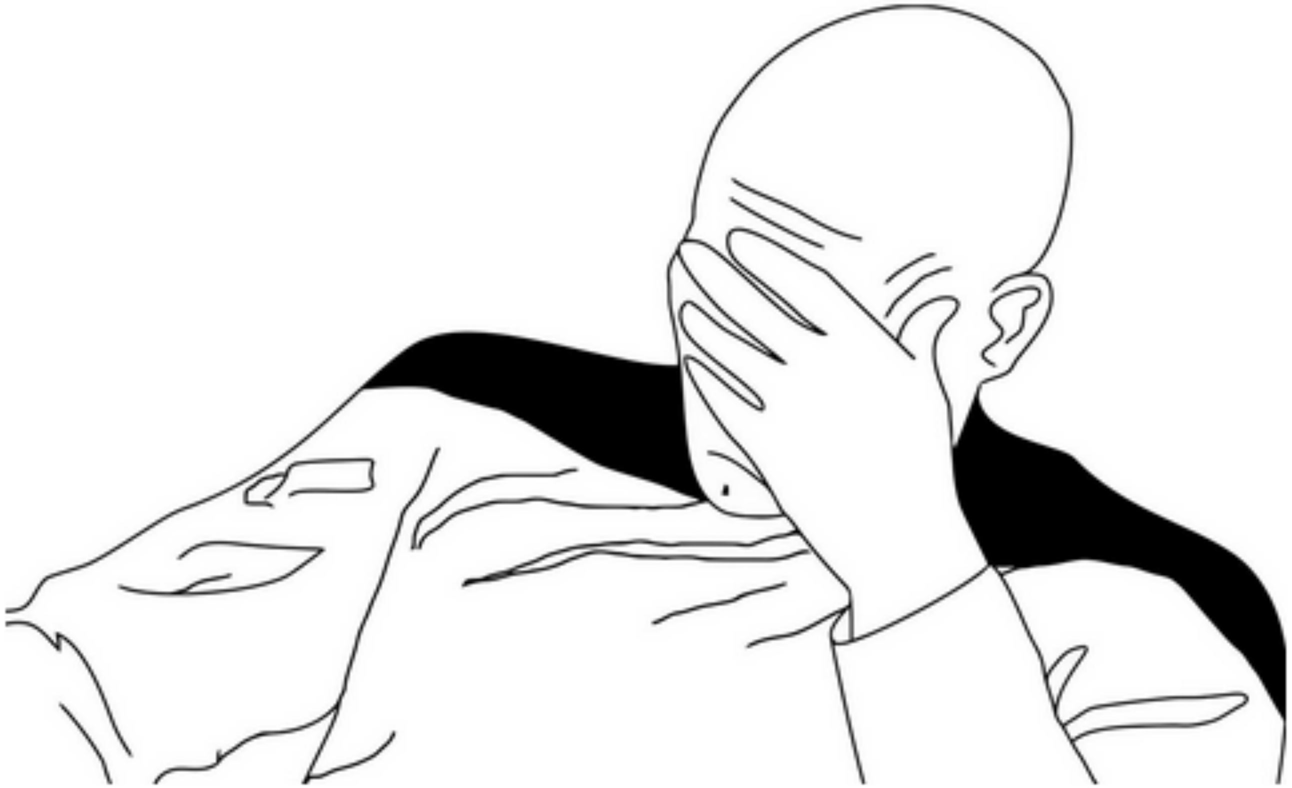
- **Timeout driven** state transitions back to idle
  - 100 ms, 100 ms, 10 s > Idle
- Similar state machine for 3G devices
  - Except CP latencies are ***much higher***

https://github.com/attdevsupport/ARO/blob/master/ARODataAnalyzer/src/lte.conf

3G power state transitions (AT&T)



- *Radio cycles between 3 states*
  - ***Idle***
  - *Low TX power (**FACH**)*
  - *High TX power (**DCH**)*



*I just wanted to make a **fast** mobile app.....*

Uh huh... Yeah, tell me more...

1. **Latency variability can be very high on mobile networks**

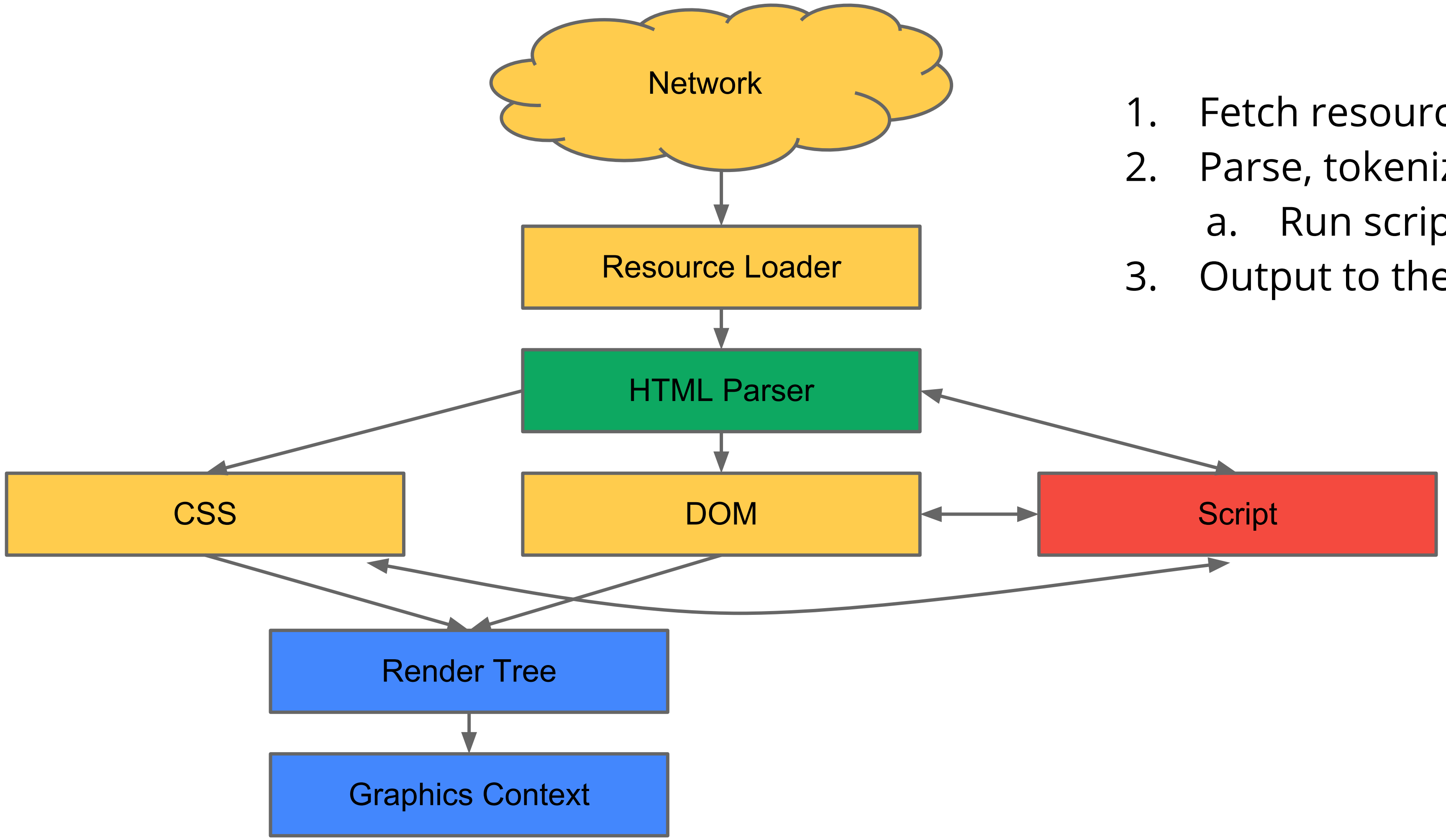
2. **4G networks will improve latency, but...**
   a. We still have a long way to go until everyone is on 4G - *a decade!*
   b. And 3G is definitely not going away anytime soon
   c. Ergo, latency and variability in latency ***is a problem***

3. **What can we do about it?**
   a. Re-use connections
   b. Download resources in bulk, avoid waking up the radio
   c. Compress resources
   d. Cache

How do we render the page?

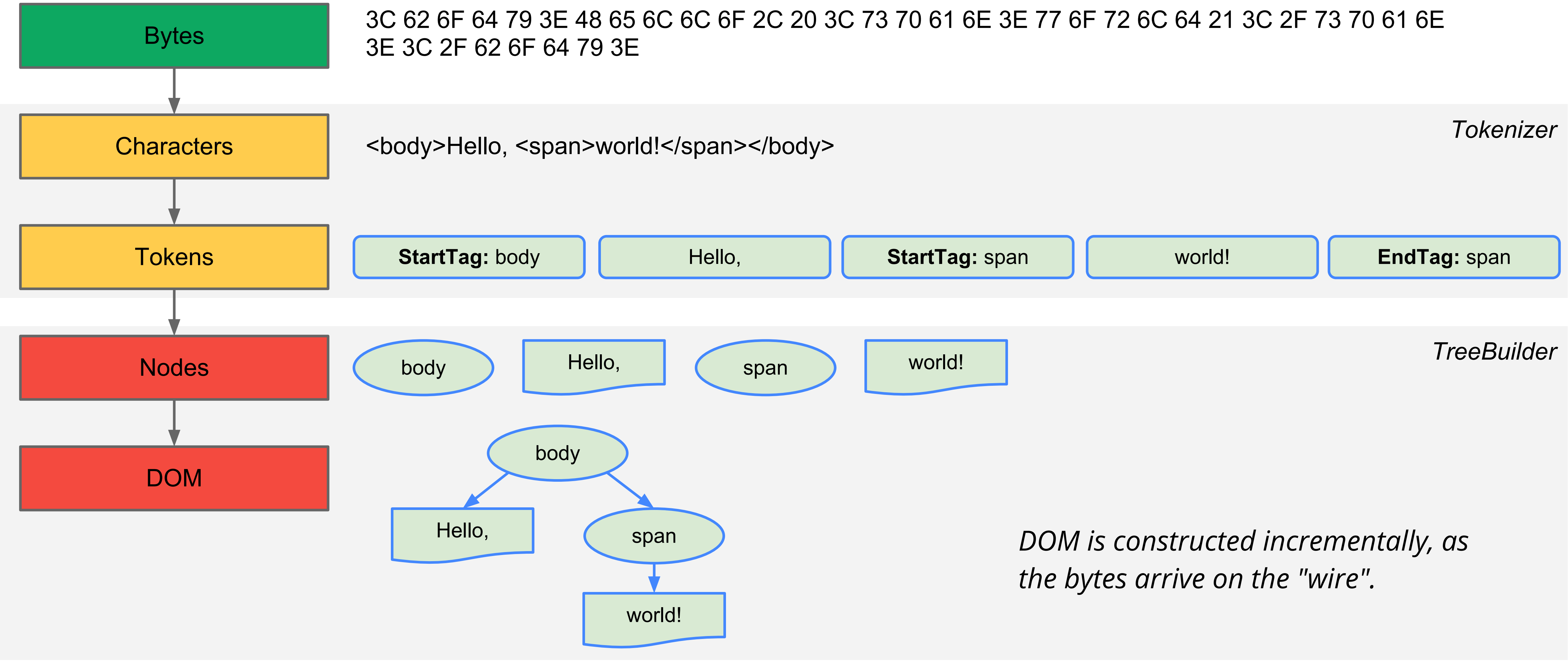
we're getting bytes off the wire... and then what?

Life of a web-page in the browser...

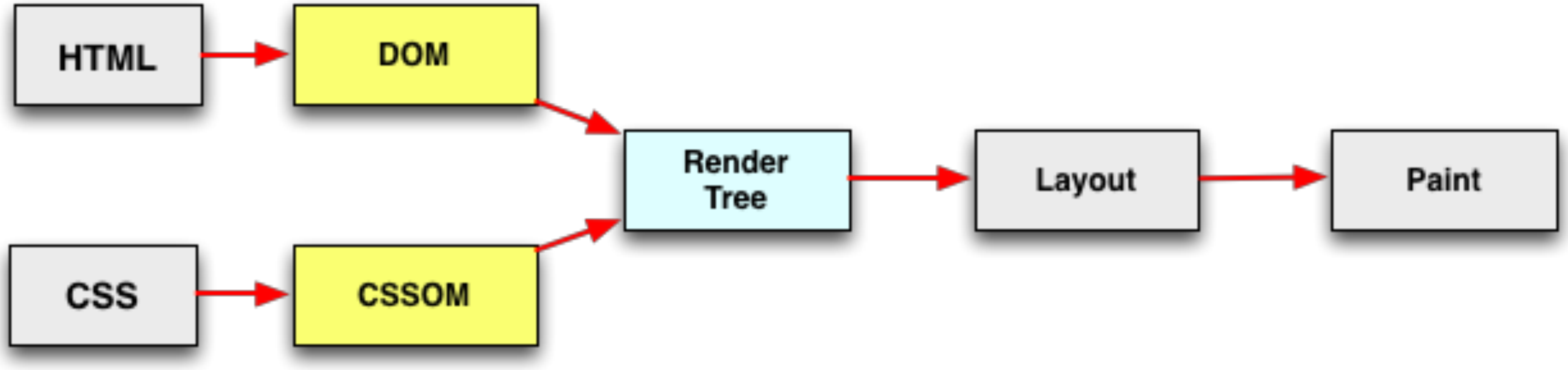


1. Fetch resources from the network
2. Parse, tokenize, construct the DOM
    a. Run scripts...
3. Output to the screen

The HTML5 parser at work...



Deciphering the Critical Rendering Path



- HTML > Document Object Model - incremental parsing
- CSS > CSS Object Model

- Rendering is blocked on **CSSOM and DOM**

The HTML5 parser at work…

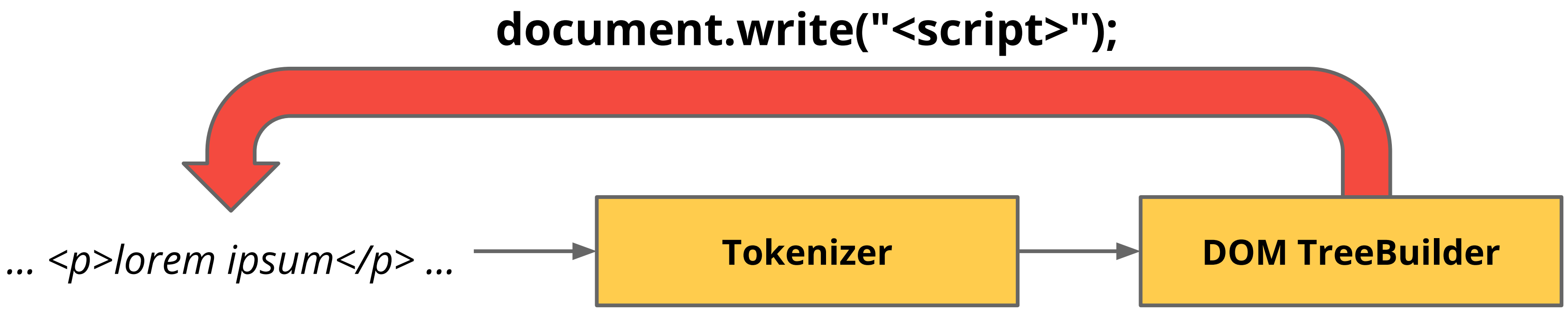
```
<!doctype html>
<meta charset=utf-8>
<title>Awesome HTML5 page</title>

<script src=application.js></script>
<link href=styles.css rel=stylesheet />

<p>I'm awesome.
```

HTMLDocumentParser begins parsing the received data …

```
HTML
  - HEAD
    - META charset="utf-8"
    - TITLE
      #text: Awesome HTML5 page
    - SCRIPT src="application.js"
      ** stop **
```

Stop. Dispatch request for application.js. Wait…

(1) Scripts can block the document parser...



JavaScript can **block** DOM construction.

Script execution can change the input stream.

Hence we **must wait for script to execute**.

Sync scripts block the parser...

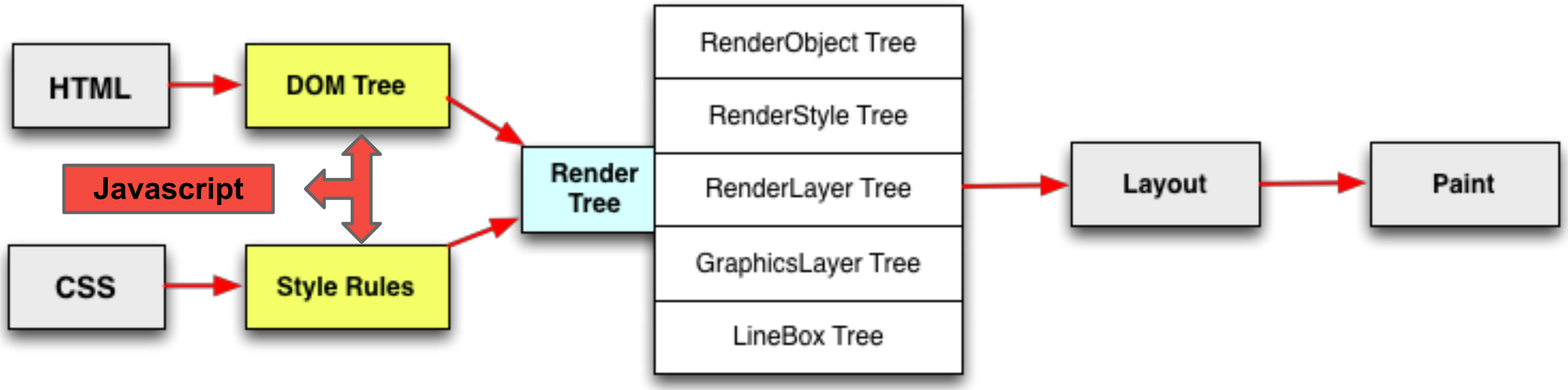
Sync script **will block** the rendering of your page:

```
<script type="text/javascript"
        src="https://apis.google.com/js/plusone.js"></script>
```

Async script **will not block** the rendering of your page:

```
<script type="text/javascript">
  (function() {
    var po = document.createElement('script'); po.type = 'text/javascript';
    po.async = true; po.src = 'https://apis.google.com/js/plusone.js';
    var s = document.getElementsByTagName('script')[0];
    s.parentNode.insertBefore(po, s);
  })();
</script>
```

(2) Javascript can query CSS, which means...



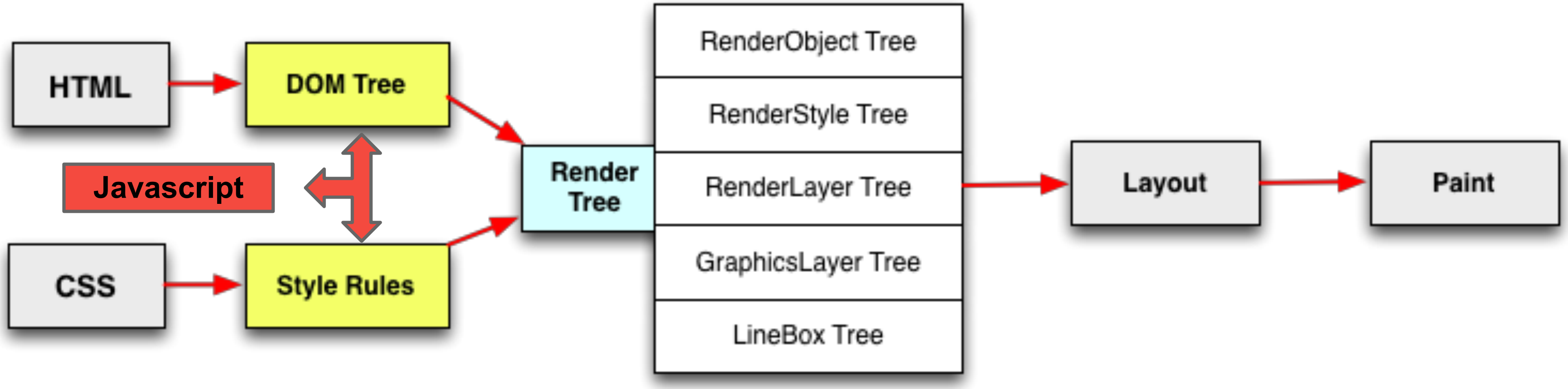
JavaScript can **block on CSS**.

DOM construction can be blocked on Javascript, which can be blocked on CSS

- *ex: asking for computed style, but stylesheet is not yet ready...*

At least CSS can't query javascript.. phew!

(3) Rendering is blocked on CSS...



CSS must be fetched & parsed before Render tree can be painted.

Otherwise, the user will see "flash of unstyled content" + reflow and repaint when CSS is ready

At least CSS can't query javascript.. phew!

Performance rules to keep in mind...

(1) JavaScript can **block the DOM** construction
(2) JavaScript can **block on CSS**
(3) Rendering is **blocked on CSS**...

Which means...

(1) Get CSS down to the client as fast as you can

- *Unblocks paints, removes potential JS waiting on CSS scenario*

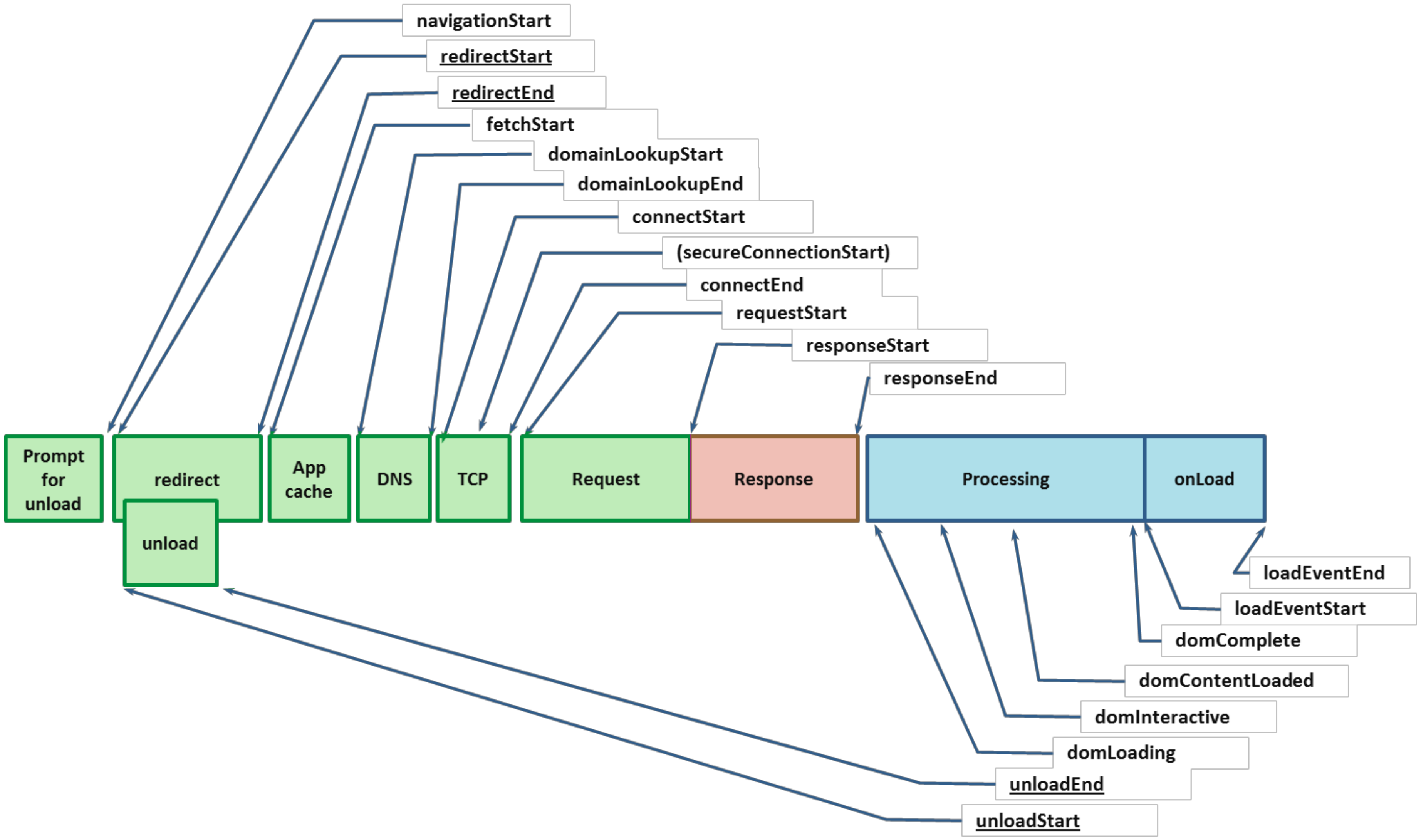
(2) If you can, use async scripts + avoid doc.write at all costs

- *Faster DOM construction, faster DCL and paint!*
- *Do you need scripts in your critical rendering path?*

Let's put it all together now

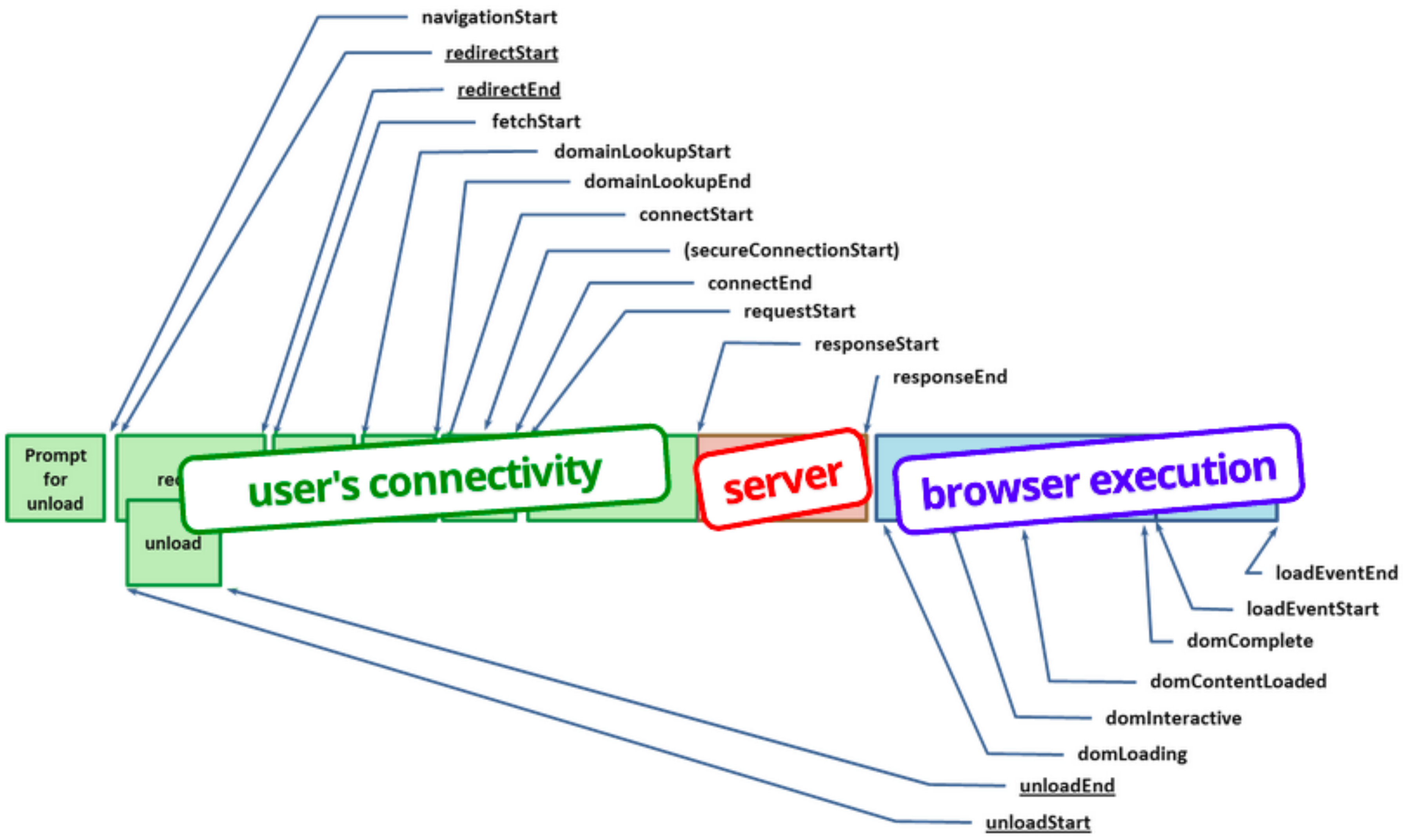
network, browser rendering pipeline, and the rest...

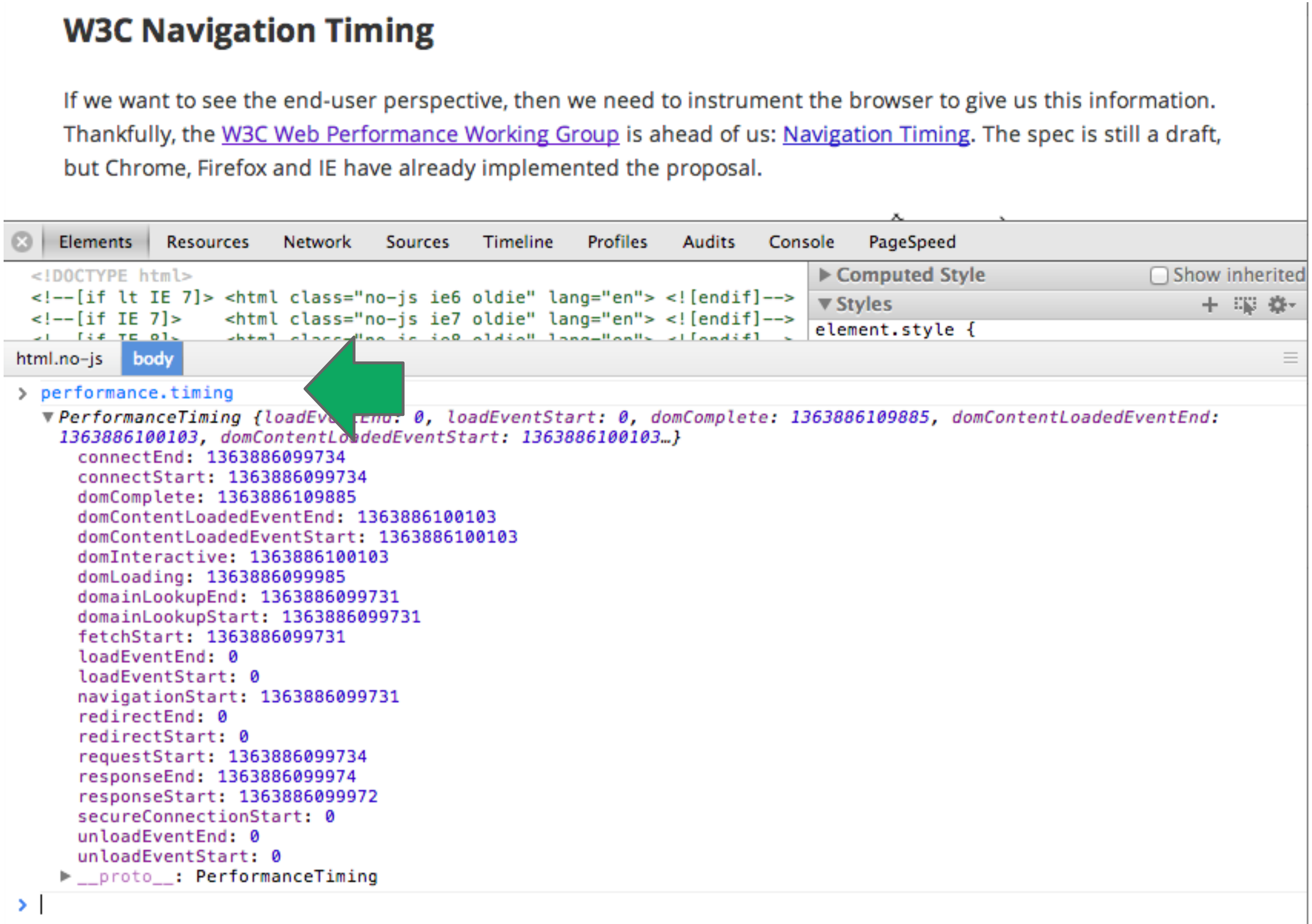
Navigation Timing (W3C)



[Navigation Timing spec](#)

Navigation Timing (W3C)



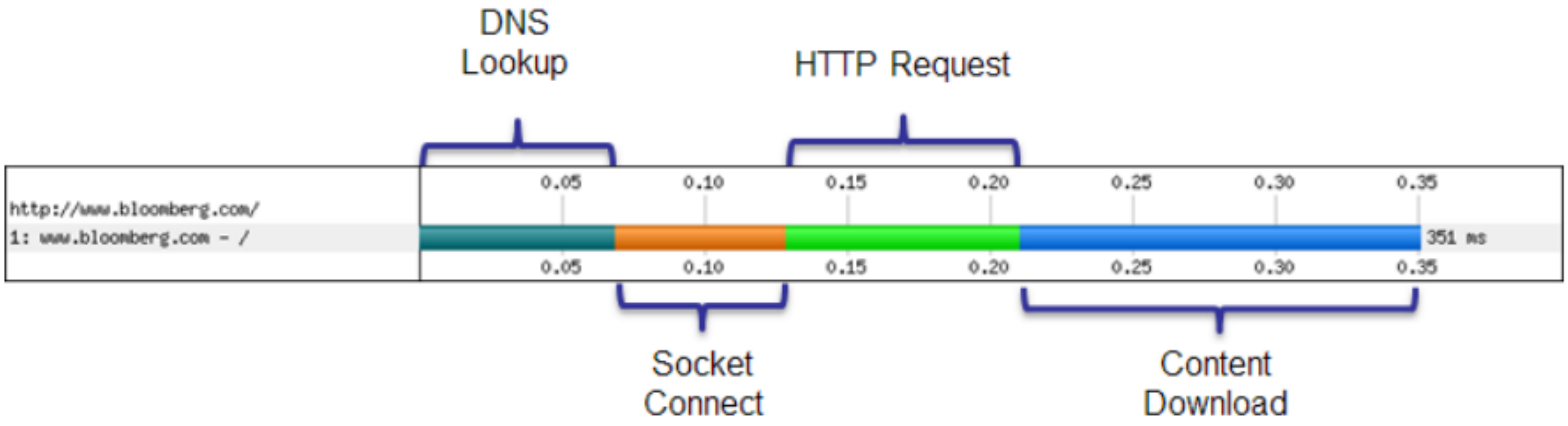


Available in...

- IE 9+
- Firefox 7+
- Chrome 6+
- Android 4.0+

caniuse.com/nav-timing

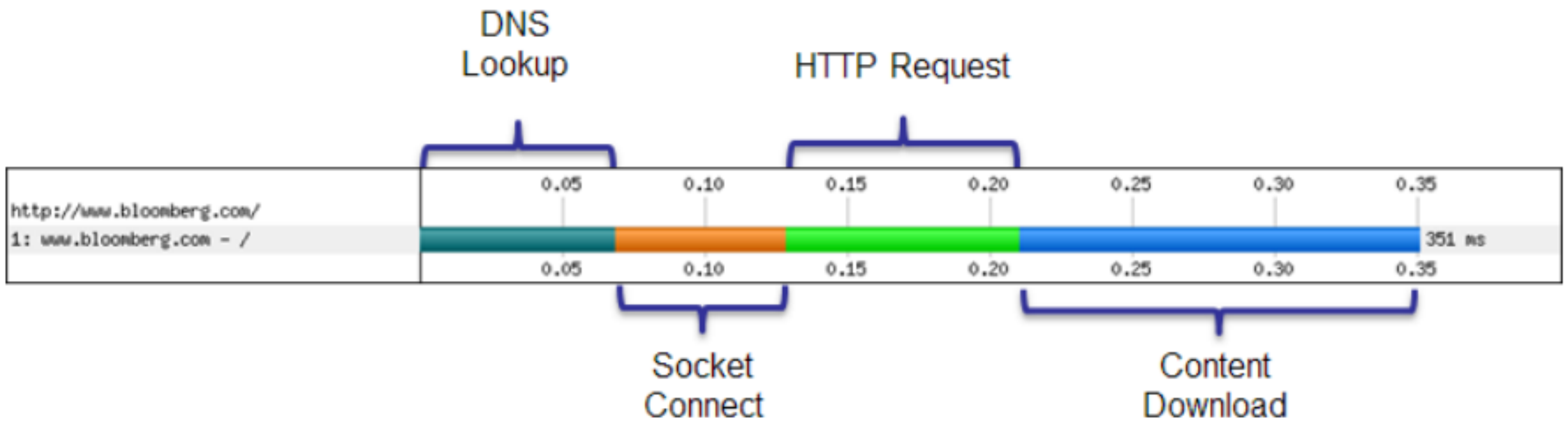
The *(short)* life of a web request



- *(Worst case)* **DNS lookup** to resolve the hostname to IP address
- *(Worst case)* **New TCP connection**, requiring a full roundtrip to the server
- *(Worst case)* **TLS handshake** with up to two extra server roundtrips!

- **HTTP request**, requiring a full roundtrip to the server
- **Server processing time**

The (short) life of our *1000 ms budget*



	3G *(200 ms RTT)*	**4G** *(80 ms RTT)*
Control plane	*(200-2500 ms)*	*(50-100 ms)*
DNS lookup	200 ms	80 ms
TCP Connection	200 ms	80 ms
TLS handshake	*(200-400 ms)*	*(80-160 ms)*
HTTP request	200 ms	80 ms
Leftover budget	**0-400 ms**	**500-760 ms**

Network overhead of one HTTP request!

@igrigorik

Our mobile apps and pages are not single HTTP requests... are they?

But, perhaps they {could, should} be?

	3G *(200 ms RTT)*	**4G** *(80 ms RTT)*
Leftover budget	**0-400 ms**	**500-760 ms**

***~400 ms** of budget left for...*

Should be **<100 ms**

- **Server processing time**
  - *what is your server processing time?*
- **Client-rendering**
  - *what does it take to render a page?*

Reserve **100 ms** for layout, rendering

200 ms

JavaScript execution and an extra request if we're lucky!

*Breaking the **1000 ms** time to glass mobile barrier... **hard facts:***

1. **Majority of time is in network overhead**
   - *Leftover budget is ~400 ms on average*
2. **Fast server processing time is a must**
   - *Ideally below 100 ms*
3. **Must allocate time for browser parsing and rendering**
   - *Reserve at least 100 ms of overhead*

Therefore...

*Breaking the **1000 ms** time to glass mobile barrier... **implications:***

1. **Inline just the required resources for above the fold**
    - *No room for extra requests... unfortunately!*
    - *Identify and inline critical CSS*
    - *Eliminate JavaScript from the critical rendering path*
2. **Defer the rest until after the above the fold is visible**
    - *Progressive enhancement...*
3. ...
4. *Profit*

A simple example in action...

network, browser rendering pipeline, and the rest...

```
<html>

<head>
 <link rel="stylesheet" href="all.css">
 <script src="application.js"></script>
</head>

<body>
 <div class="main">
   Here is my content.
 </div>
 <div class="leftnav">
   Perhaps there is a left nav bar here.
 </div>
 ...
</body>
</html>
```

1. Split **all.css**, inline AFT styles
2. Do you need the JS at all?
   - Progressive enhancement
   - Inline AFT JS code
   - Defer the rest

```html
<html>
<head>

 <style>
   .main { ... }
   .leftnav { ... }
   /* ... any other styles needed for the initial render here ... */
 </style>

 <script>
   // Any script needed for initial render here.
   // Ideally, there should be no JS needed for the initial render
 </script>

</head>
<body>
 <div class="main">
   Here is my content.
 </div>
 <div class="leftnav">
   Perhaps there is a left nav bar here.
 </div>

 <script>
    function run_after_onload() {
      load('stylesheet', 'remainder.css')
      load('javascript', 'remainder.js')
    }
 </script>

</body>
</html>
```

Above the fold CSS

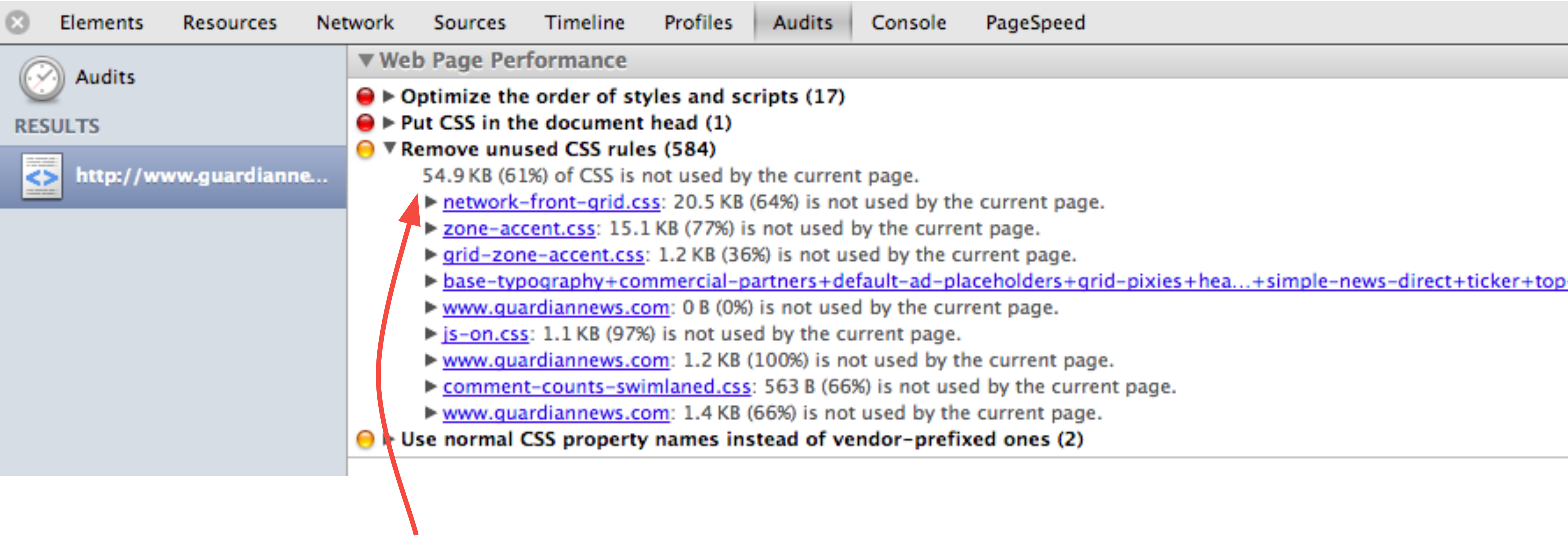
Above the fold JS
(ideally, none)

Paint the above the fold,
then fill in the rest

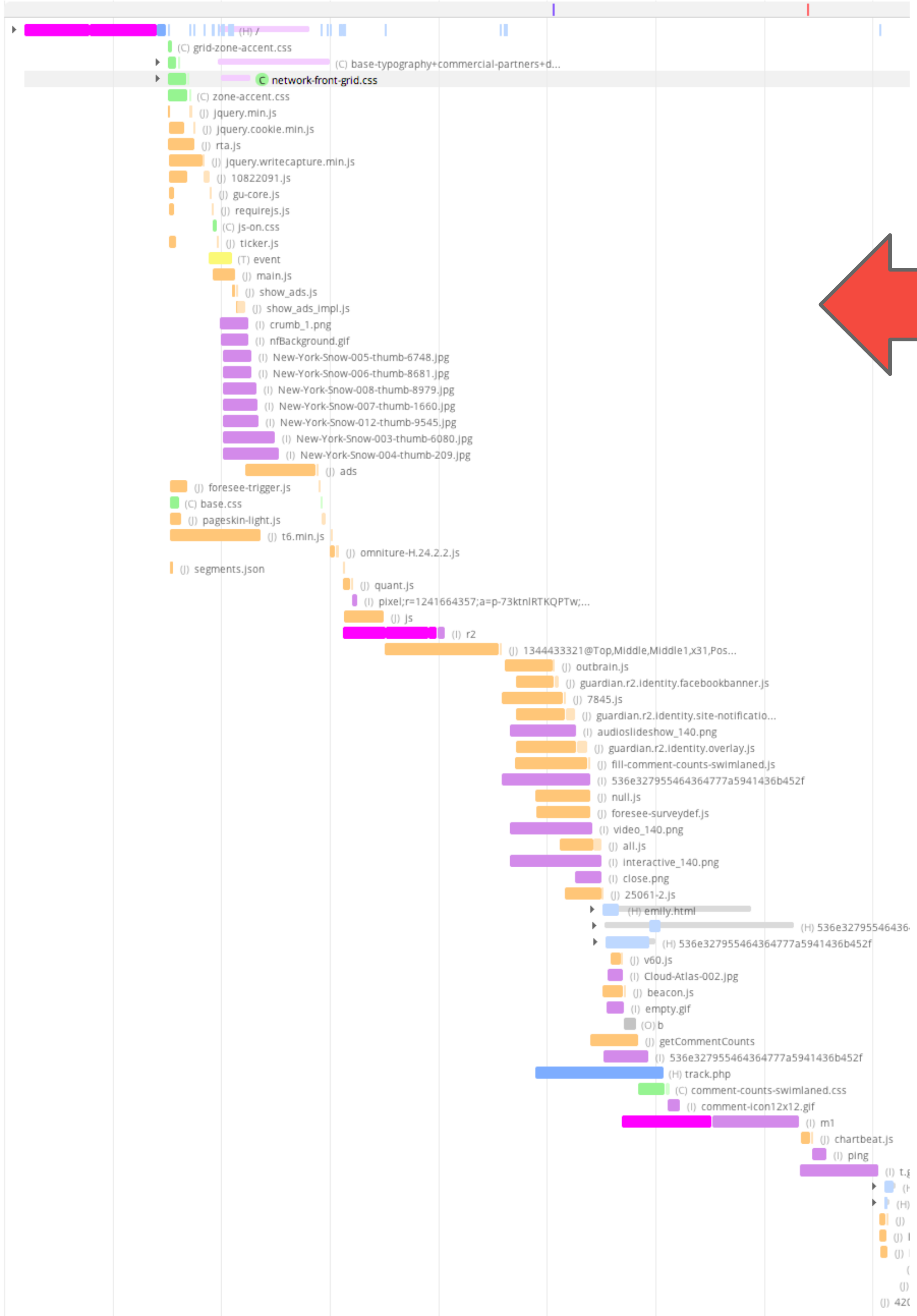
A few tools to help you...

How do I find "critical CSS" and my critical rendering path?

Identify critical CSS via an Audit



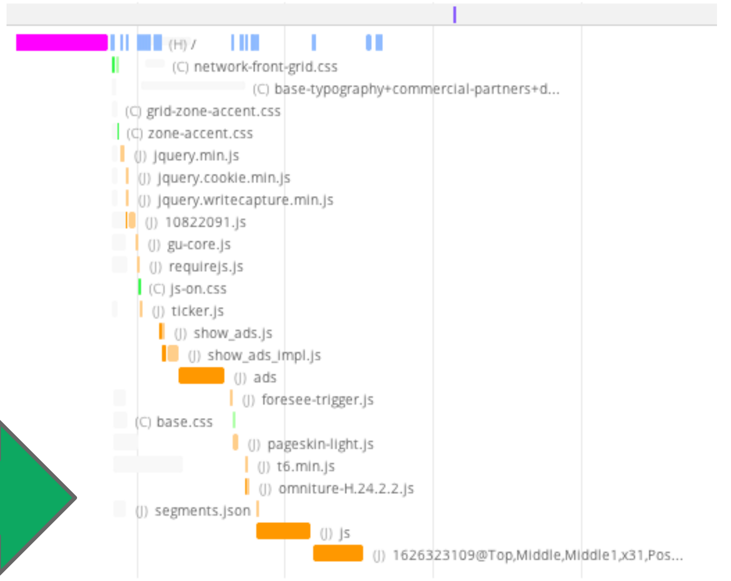
DevTools > Audits > Web Page Performance



guardian.co.uk

Full Waterfall

Critical Path



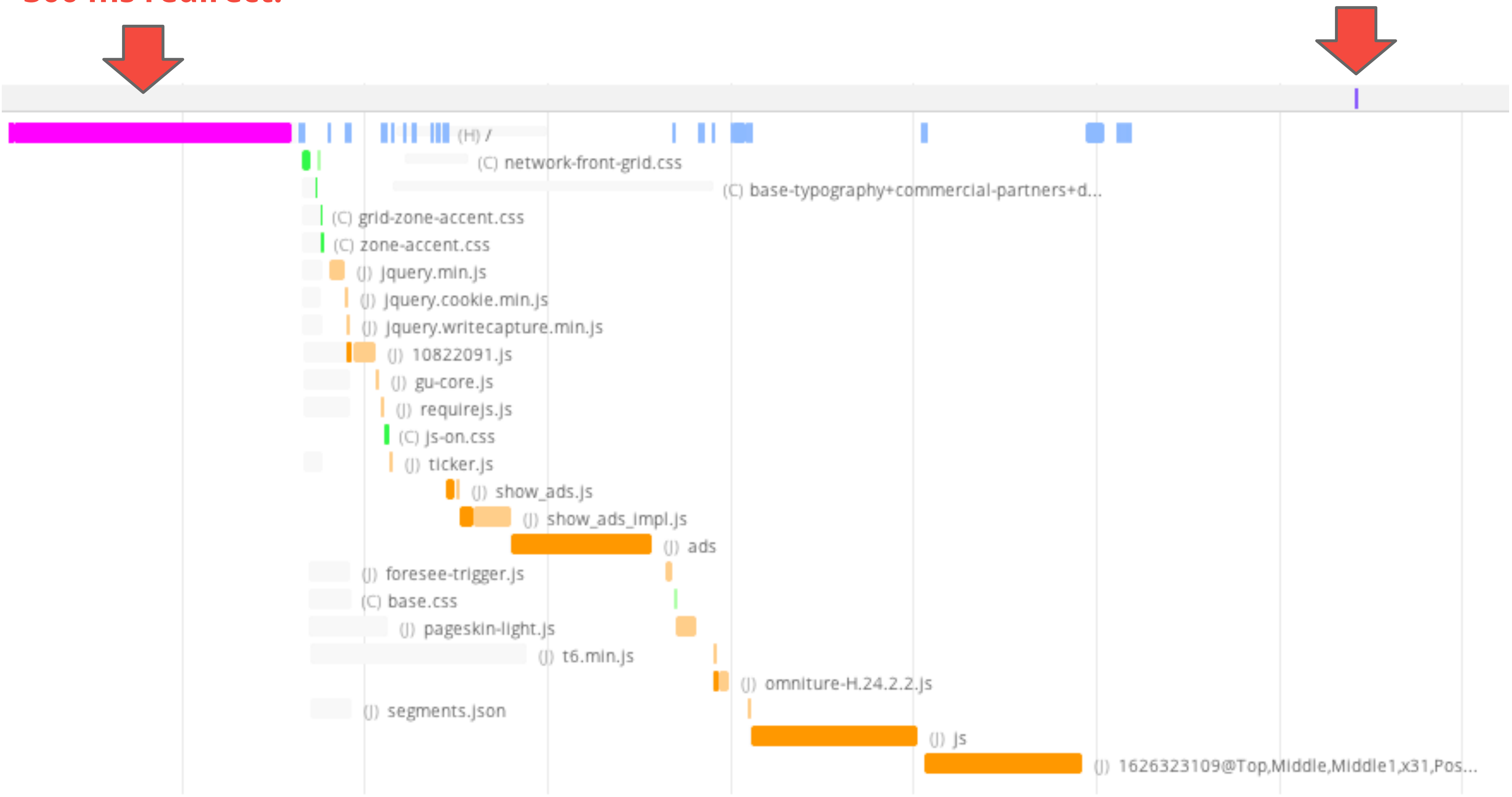
Critical Path Explorer* extracts the subtree** *of the waterfall that is in the* ***"critical path" *of the* ***document parser and the renderer.***

(automation for the win!)

@igrigorik

300 ms redirect!

DCL.. no defer



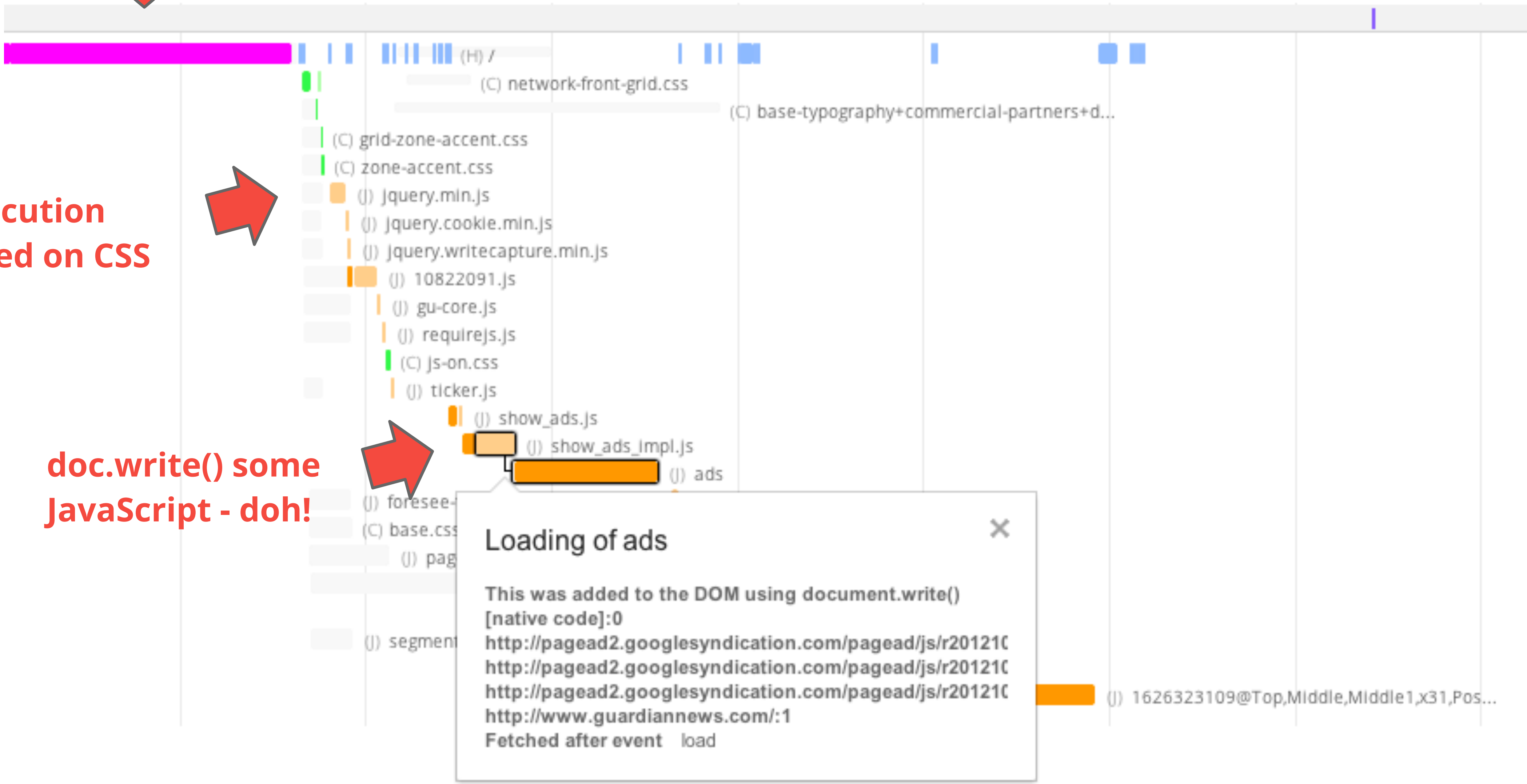
300 ms redirect!

JS execution blocked on CSS

300 ms redirect!

JS execution blocked on CSS

doc.write() some JavaScript - doh!



@igrigorik

300 ms redirect!
(H) /
(C) network-front-grid.css
(C) base-typography+commercial-partners+d...
(C) grid-zone-accent.css
(C) zone-accent.css
(J) jquery.min.js
(J) jquery.cookie.min.js
(J) jquery.writecapture.min.js
(J) 10822091.js
(J) gu-core.js
(J) requirejs.js
(C) js-on.css
(J) ticker.js
(J) show_ads.js
(J) show_ads_impl.js
(J) ads
(J) foresee-trigger.js
(C) base.css
(J) pageskin-light.js
(J) t6.min.js
omniture-H.24.2.2.js
(J) segments.json
J js
(J) 1626323109@Top,Middle,Middle1,x31,Pos...
JS execution blocked on CSS
doc.write() some JavaScript - doh!
long-running JS

@igrigorik

One request. Inline. Defer the rest.

*It's not as crazy, or as hard as it sounds: **investigate your critical rendering path**.*

Thanks! Questions?

- *1000 ms total budget*
  - 600 ms in network overhead
  - 400 ms for server processing and browser rendering
    - **aim for <100 ms server response**
    - **reserve 100 ms for browser rendering**

- **To beat 1000 ms time to glass barrier**
  - Inline critical CSS (no room for other requests)
  - Eliminate JavaScript from critical rendering path

Slides @ bit.ly/mobile-barrier
Video @ bit.ly/12GFKDE

bit.ly/browser-networking

+Ilya Grigorik - igrigorik@google.com - @igrigorik