WEB PERFORMANCE

BASIC CONCEPTS & OPTIMIZATIONS

Aurimas Likas
alikas@kayak.com

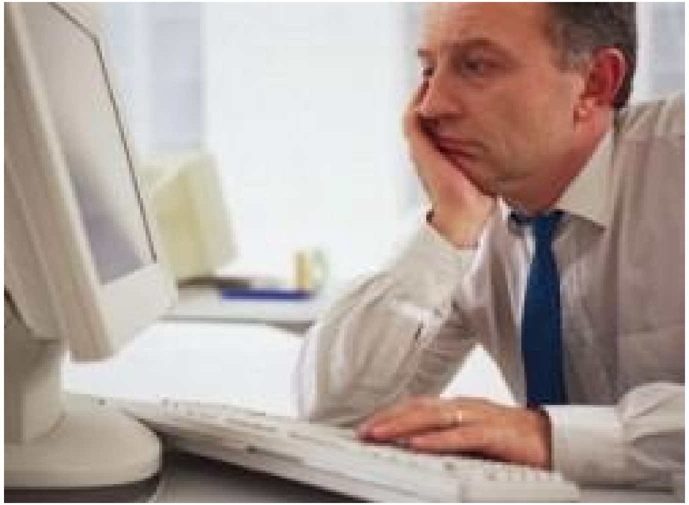
Overview

1. Is it that important and why?
2. Trends
3. Critical rendering path
4. Measure & analyze
5. Optimizing

(1) IS IT IMPORTANT AND WHY?

Simply put, people hate waiting in lines...

People **hate** waiting for web pages to load...



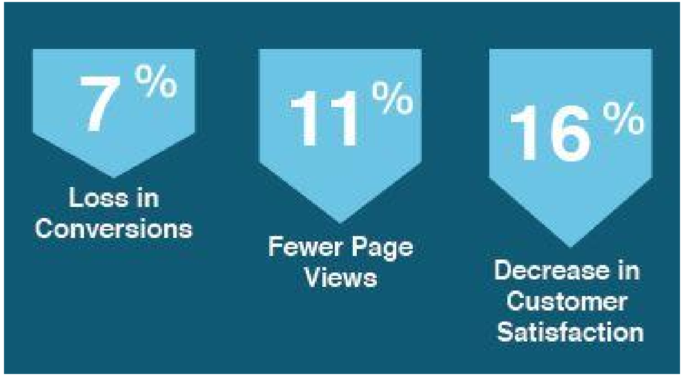
It's so easy to *navigate to a competitor's site* ☹

Performance matters!

It affects:

- **user behavior & UX** (engagement, bounce rates)
- **business metrics** (conversion, reputation, revenue)
- **SEO**

1 second matters. *A lot.*



* studies by Akamai and Gomez ~5 years ago so it's likely that web users' expectations are even higher now!

- **50%** of people expect page to load in <**2s**
- **40%** ***abandon a website*** that takes >**3s** to load
- **50%** rate a quick page loading as a key component to site loyalty
- **80%** of shoppers having trouble with web site performance ***won't return to the site*** to buy again
- More than **33%** will ***share poor experience*** with friends and colleagues

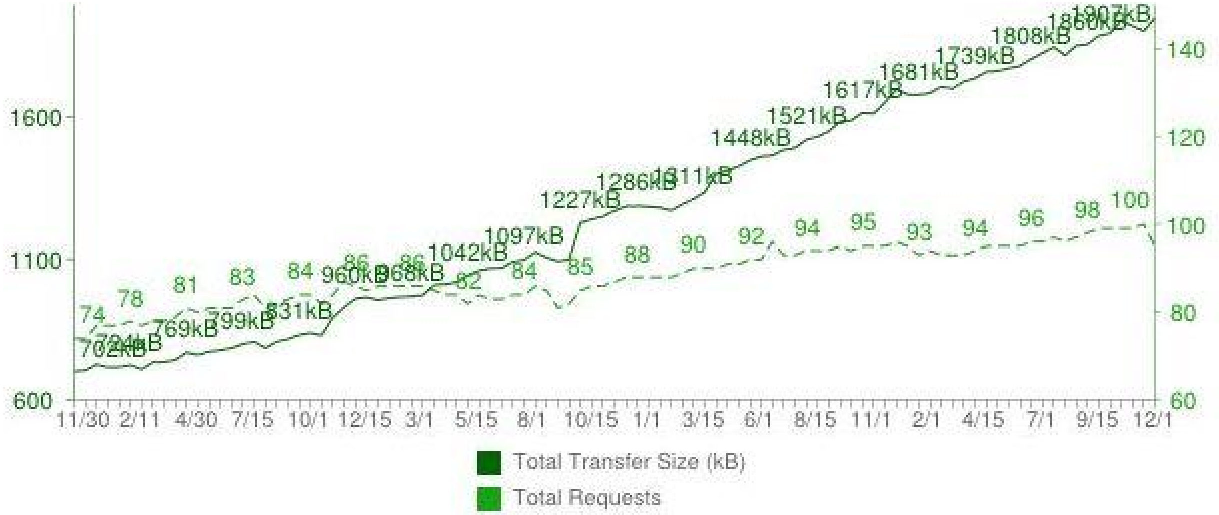
So what is fast?

Delay	User reaction
0 - 100 ms	Instant
100 - 300 ms	Slight perceptible delay
300 - 1000 ms	Task focus, perceptible delay
1 s+	Mental context switch
10 s+	I'll come back later...

- *To keep the user engaged, the task must complete within 1000 milliseconds.*
- *Unofficial rule of big companies like Google, Amazon, etc. is load the page under 250 ms*

(2) TRENDS

Applications are getting more complex and growing...



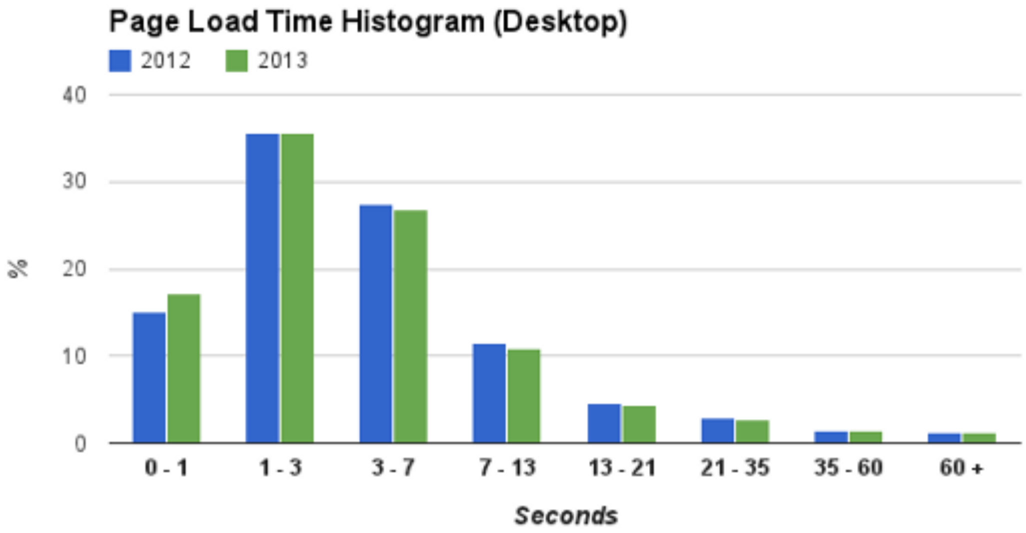
- Total requests count on Dec 2010 was 70+. Now it’s 100.
- Total transfer size increased from 720 kB to 1950 kB!

Distribution of content types

Content Type	**Desktop**		**Mobile**	
	Avg # of requests	**Avg size**	**Avg # of requests**	**Avg size**
HTML	9.8	59 Kb	7	50 Kb
Images	**53**	**1243 Kb**	**38**	**697 Kb**
Javascript	**18**	**295 Kb**	**15**	**239 Kb**
CSS	6.1	57 Kb	3.5	46 Kb
Total	**86,9**	**1654 Kb**	**63,5**	**1032 Kb**

Stats from HttpArchive.org which currently tracks almost 500k sites.

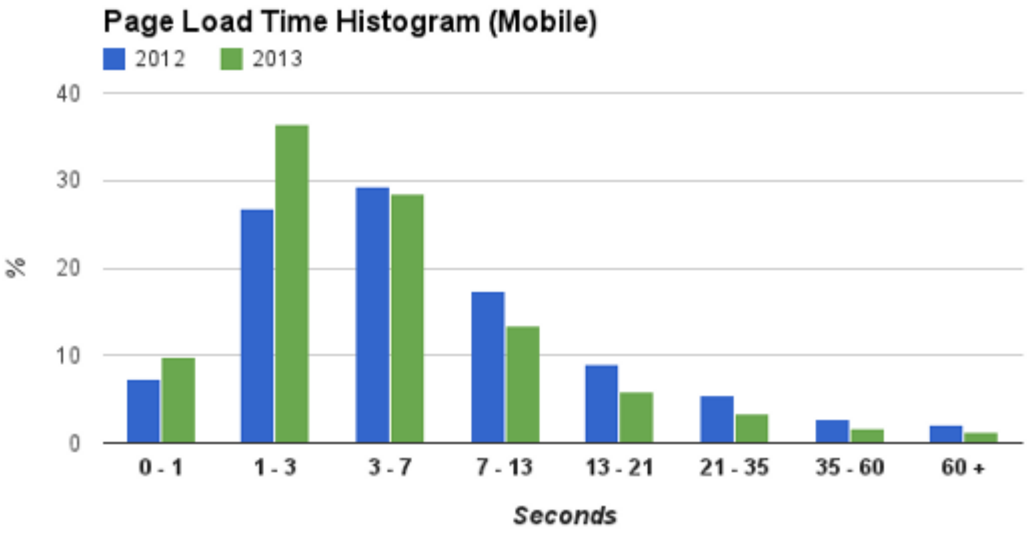
Is the web getting faster?



Desktop: ~3.1 s

"While access from desktop is only a bit faster, it is still impressive given that the size of the web pages have increased by over 56% during this period."

Is the web getting faster?



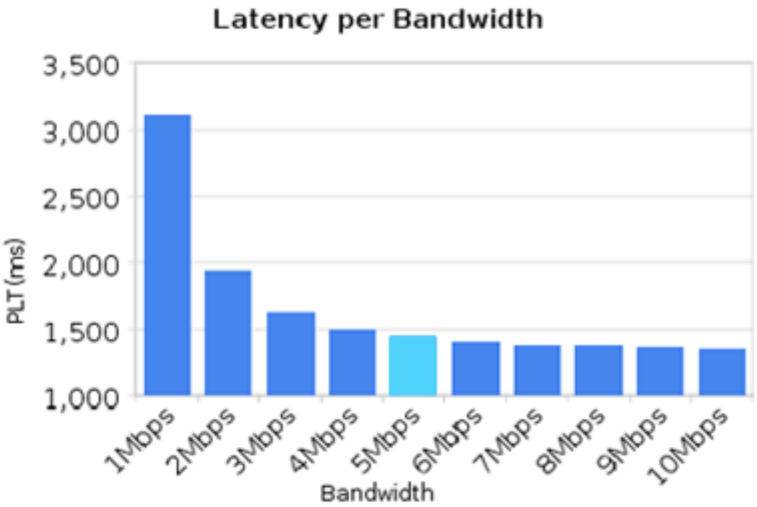
Mobile: ~3.5 s

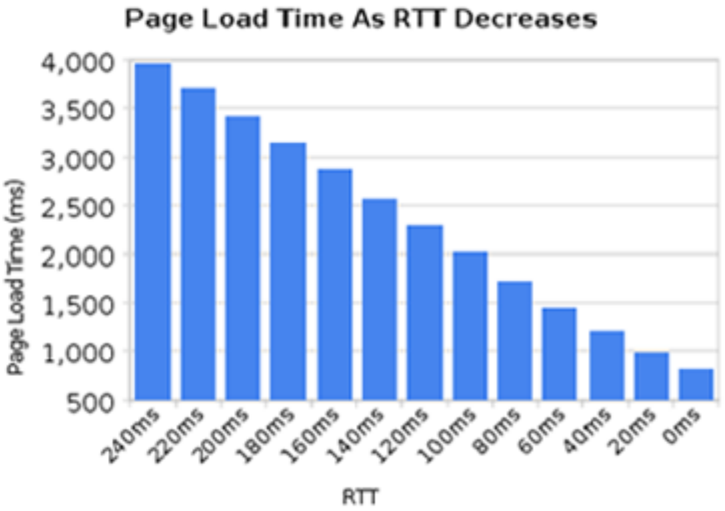
"It’s great to see access from mobile is around 30% faster compared to last year."

Will new technologies save us?

- Internet speed is increasing
- Computers and phones getting faster
- Web browsers have gotten faster too

More Bandwidth Doesn’t Equate to Better Performance



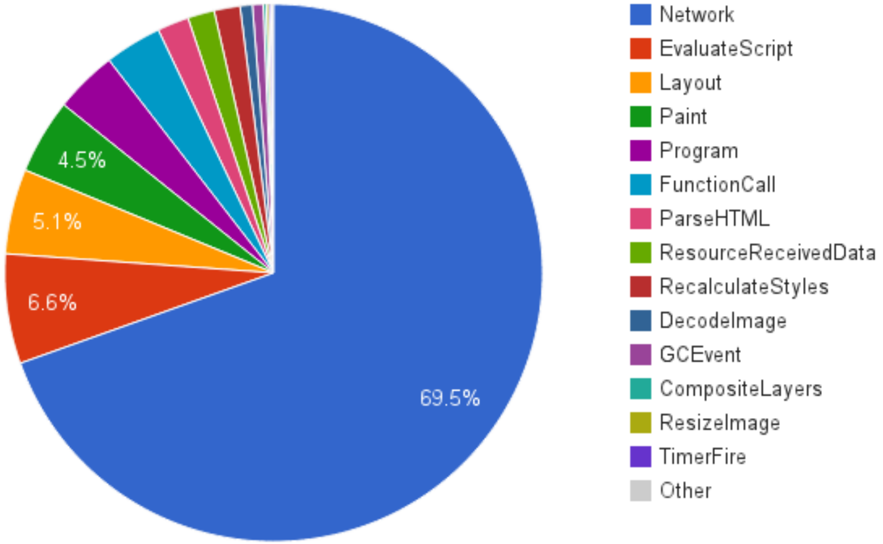


- upgrading from 5Mbps to 10Mbps results in a mere 5% improvement in page loading times
- for every 20ms improvement in latency, we have a linear improvement in page loading times

Latency is the bottleneck

- Lots of small transfers
- New TCP connections are expensive
- High latency overhead on mobile networks

Top 1 million Alexa sites



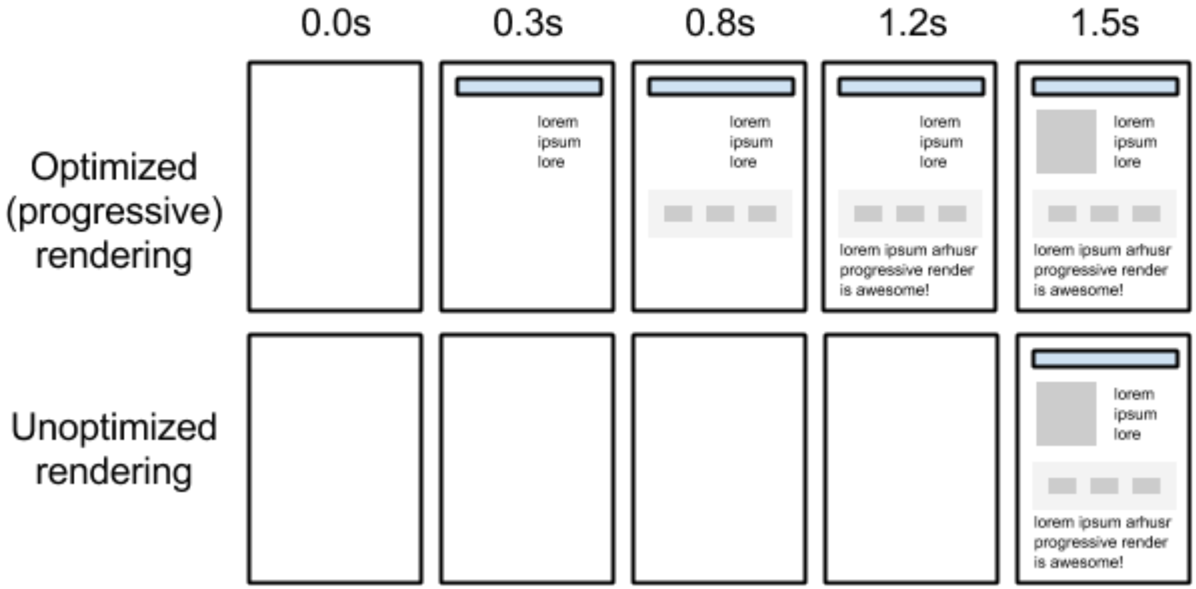
- 69.5% of time blocked on network
- 6.6% of time blocked JavaScript
- 5.1% blocked on Layout
- 4.5% blocked on Paint
- ...

No surprises here... First page load is network (latency) bound!

(3) CRITICAL RENDERING PATH

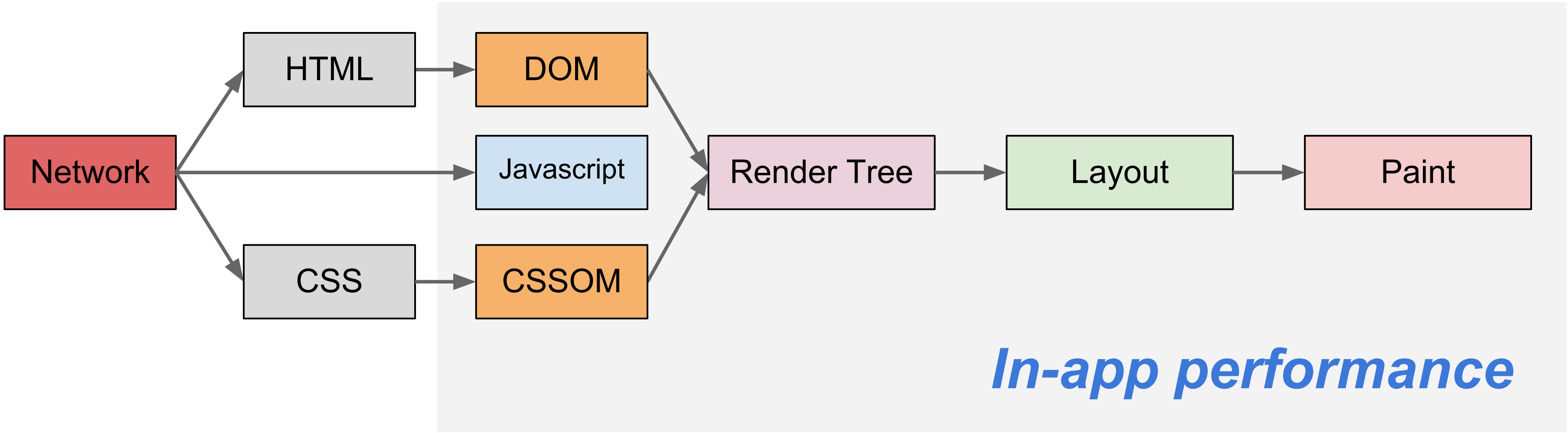
Critical rendering path

Optimizing for performance is about understanding what happens in between receiving the HTML, CSS, and JavaScript bytes and the required processing to turn them into rendered pixels - that's the critical rendering path.



0.0s
0.3s
0.8s
1.2s
1.5s
Optimized (progressive) rendering
Unoptimized rendering
lorem ipsum lore
lorem ipsum arhusr progressive render is awesome!

Critical rendering path



Simple example

index.html

```html
<!doctype html>
<meta charset=utf-8>
<title>Performance!</title>

<link href=styles.css rel=stylesheet />

<p>Hello <span>world!</span></p>
```

style.css

```css
p { font-weight: bold; }
span { display: none;
```

- Simple (valid) HTML file
- External CSS stylesheet

First HTML bytes arrive...

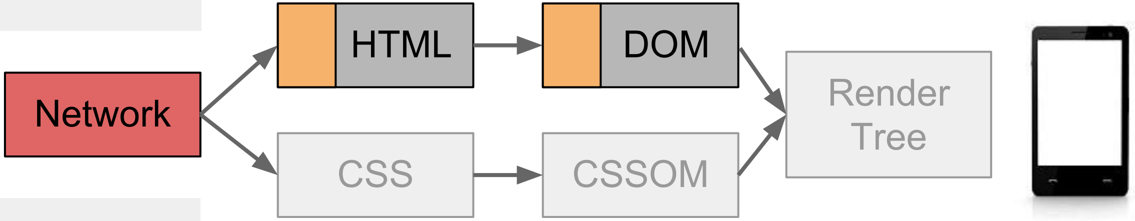
index.html

```
<!doctype html>
<meta charset=utf-8>
<title>Performance!</title>

<link href=styles.css rel=stylesheet />

<p>Hello <span>world!</span></p>
```

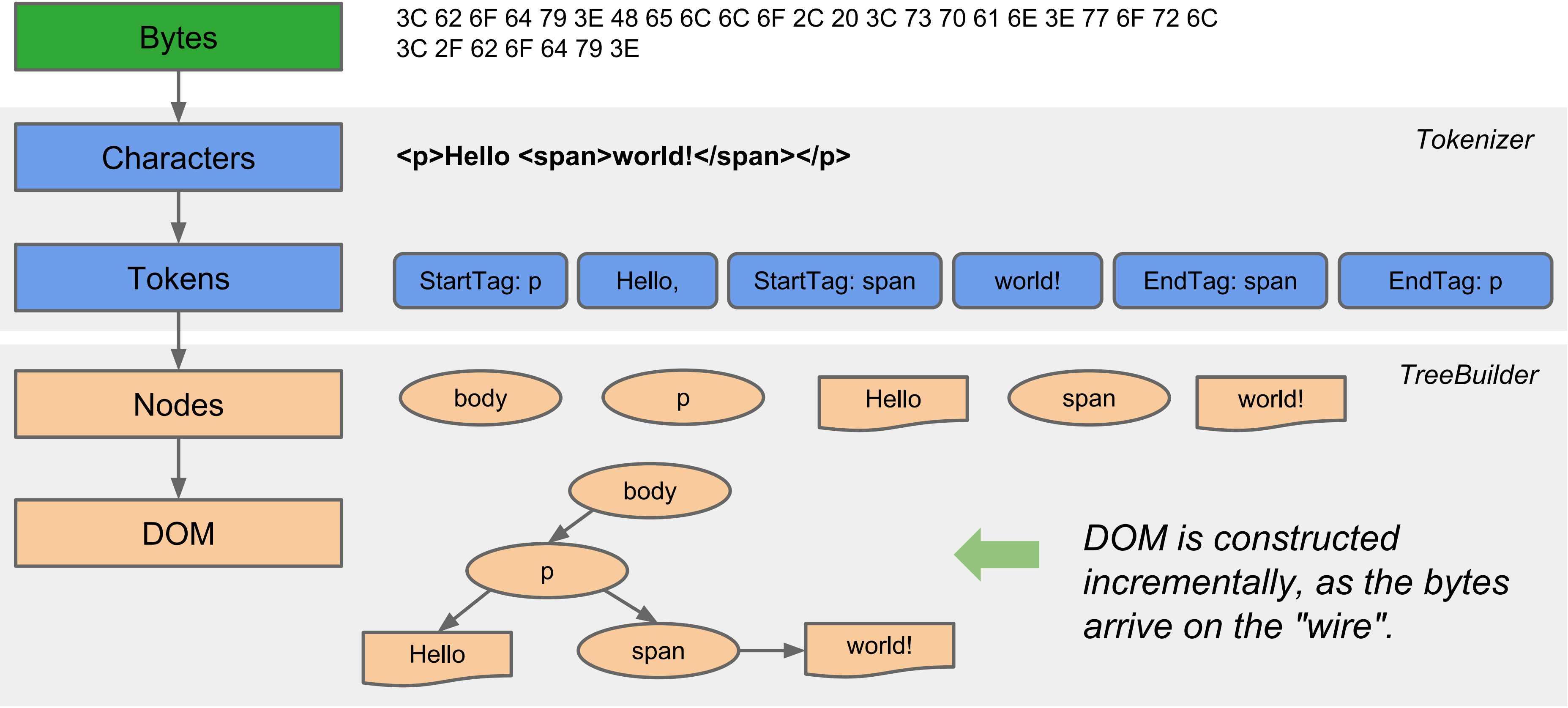
style.css

```
p { font-weight: bold; }
span { display: none;
```

- first bytes of index.html response packet
- we have not discovered the CSS yet...



The HTML5 parser starts it's work...



Bytes

3C 62 6F 64 79 3E 48 65 6C 6C 6F 2C 20 3C 73 70 61 6E 3E 77 6F 72 6C
3C 2F 62 6F 64 79 3E

Tokenizer

Characters

<p>Hello <span>world!</span></p>

Tokens

StartTag: p | Hello, | StartTag: span | world! | EndTag: span | EndTag: p

TreeBuilder

Nodes

body | p | Hello | span | world!

DOM

body
p
Hello
span
world!

DOM is constructed incrementally, as the bytes arrive on the "wire".

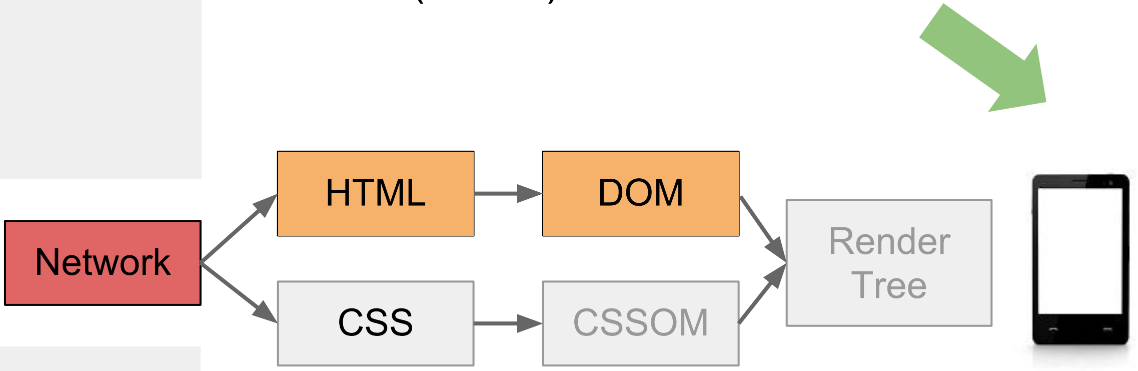
DOM construction is INCREMENTAL!

DOM is ready... waiting on CSS!

index.html

```
<!doctype html>
<meta charset=utf-8>
<title>Performance!</title>

<link href=styles.css rel=stylesheet />

<p>Hello <span>world!</span></p>
```

- <link> discovered, network request sent
- screen is empty, blocked on CSS
  - otherwise, flash of unstyled content (FOUC)



style.css

```
p { font-weight: bold; }
span { display: none;
```

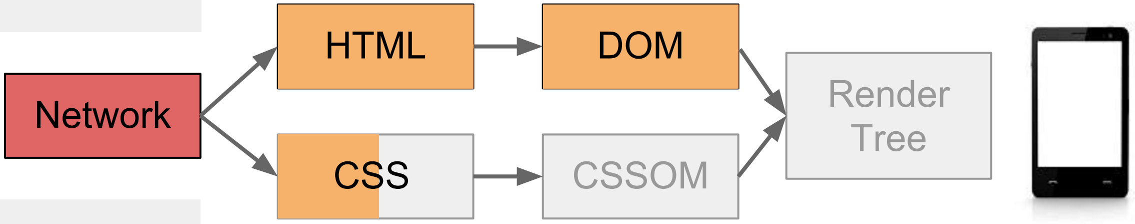
First CSS bytes arrive... still waiting on CSS!

index.html

```
<!doctype html>
<meta charset=utf-8>
<title>Performance!</title>

<link href=styles.css rel=stylesheet />

<p>Hello <span>world!</span></p>
```

We must wait for the entire file...

Unlike HTML parsing, **CSS is NOT incremental**



style.css

```
p { font-weight: bold; }
span { display: none; }
```

CSS download has finished - woohoo!

index.html

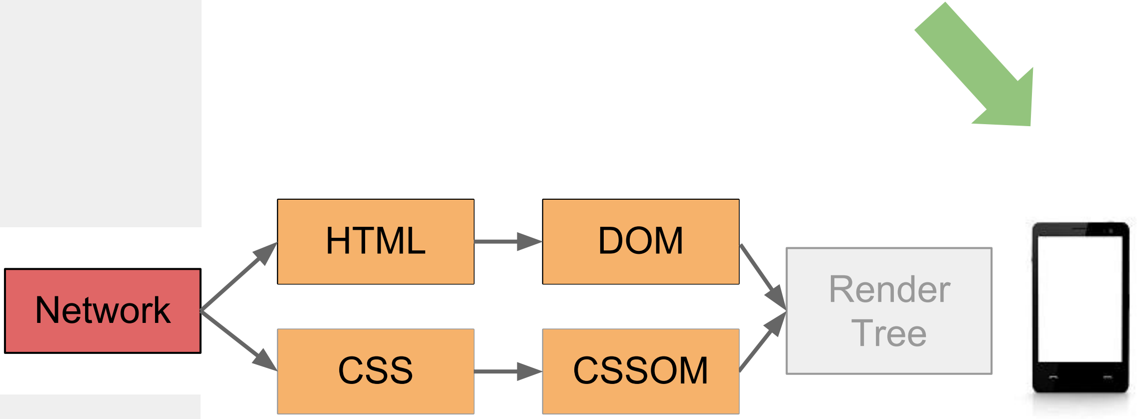
```html
<!doctype html>
<meta charset=utf-8>
<title>Performance!</title>

<link href=styles.css rel=stylesheet />

<p>Hello <span>world!</span></p>
```

style.css

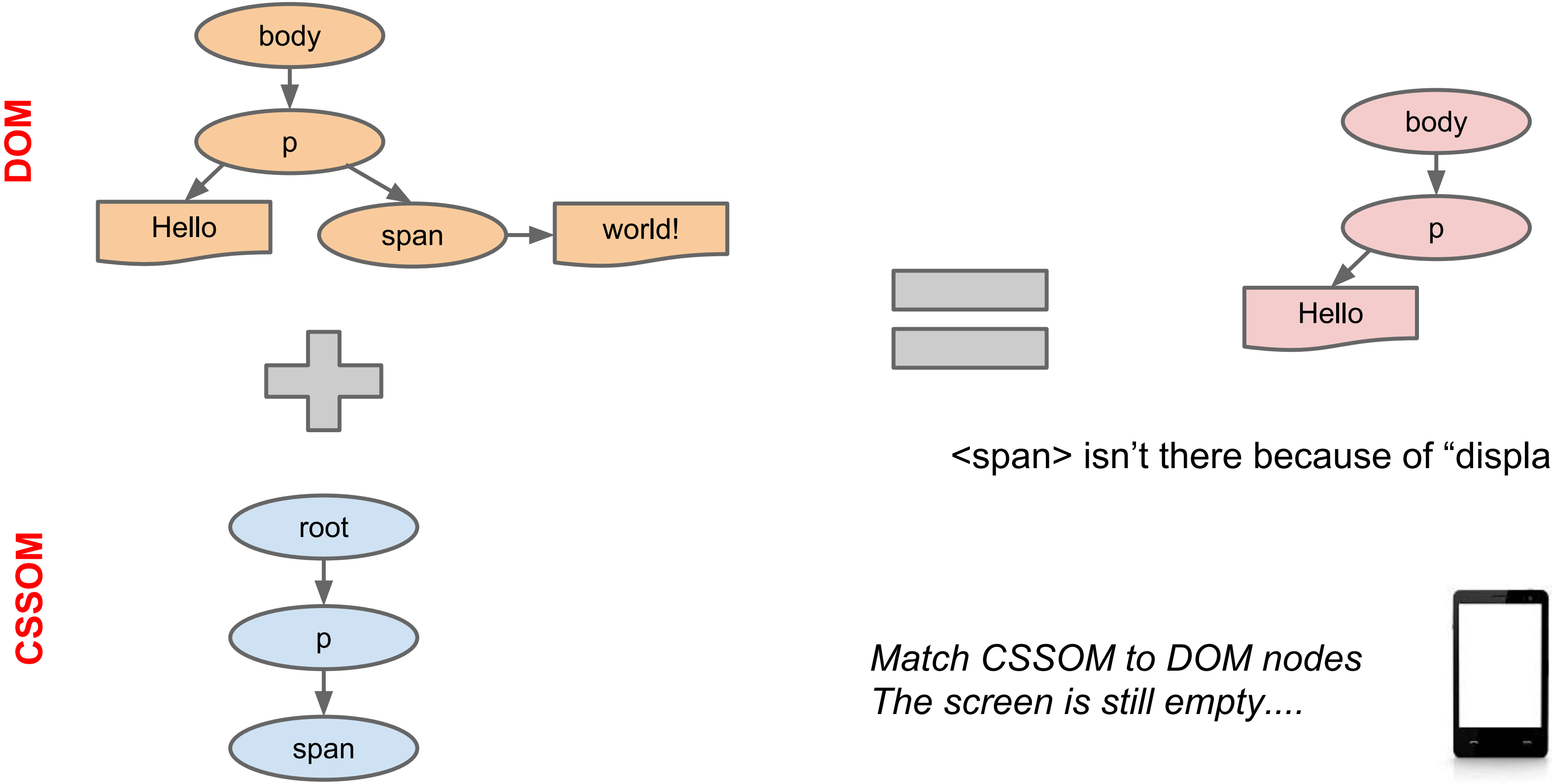
```css
p { font-weight: bold; }
span { display: none; }
```

We can construct the CSSOM!

But... the screen is still blank :((

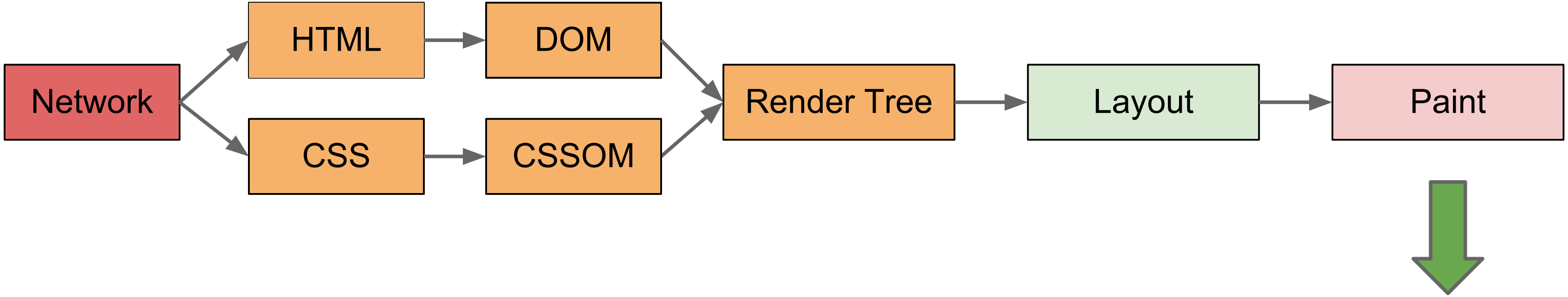


DOM + CSSOM = Render Tree

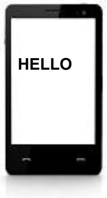


<span> isn't there because of "display: none"

Match CSSOM to DOM nodes
The screen is still empty....



- Once render tree is ready, perform layout (computes the exact position and size of each object)
- Once layout is complete, paint step renders pixels to the screen

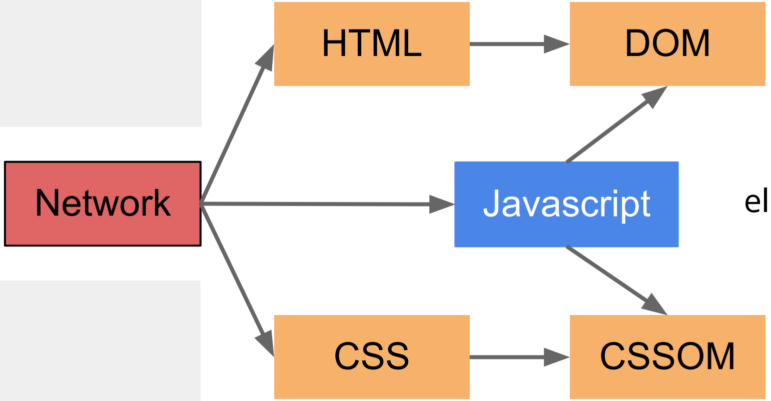


How about Javascript?

index.html

```
<!doctype html>
<meta charset=utf-8>
<title>Performance!</title>

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

<p>Hello <span>world!</span></p>
```

JavaScript can query (and modify) DOM, CSSOM!



elem.style.width = "500px"

style.css

```
p { font-weight: bold; }
span { display: none; }
```

JavaScript performance pitfalls

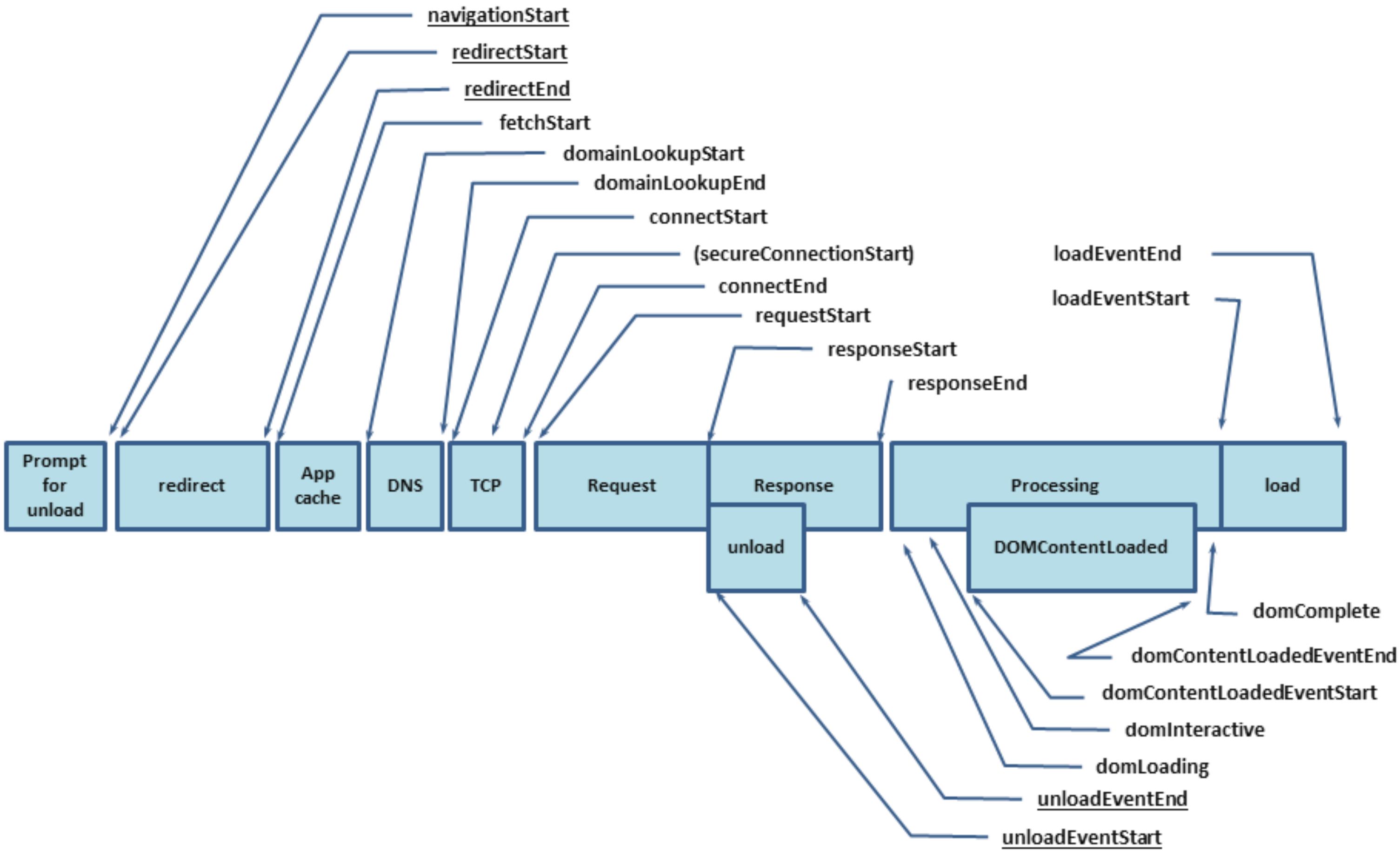
```
<script>
var old_width = elem.style.width;
elem.style.width = "300px";
document.write("I'm awesome")
</script>
```

- JavaScript can **query CSSOM**
- JavaScript can **block on CSS**
- JavaScript can **modify CSSOM**

- JavaScript can **query DOM**
- JavaScript can **block DOM construction**
- JavaScript can **modify DOM**

(4) MEASURE & ANALYZE

Navigation Timing (W3C)



Measure, analyze, optimize

1. Measure user perceived network latency with Navigation Timing
2. Analyze RUM data to identify performance bottlenecks
3. Use GA's advanced segments or similar solution
4. Setup (daily, weekly, ...) reports

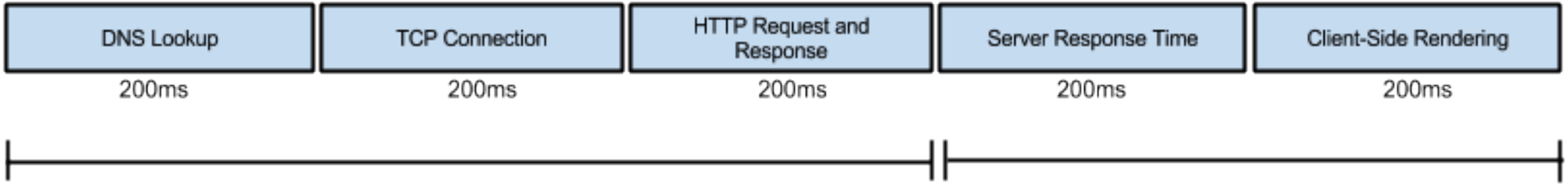
(5) OPTIMIZING

The foundation of the optimization

To deliver the fastest first render we need to optimize three variables:

- *Minimize* the **number of critical resources**
- *Minimize* the **number of critical bytes**
- *Minimize* the **critical path length**

Rendering a mobile page in 1s



600 ms mandatory 3G network overhead which you cannot do anything about

400 ms which you can optimize

It’s challenging..

Optimize server response (<200ms)

Reduce DNS lookups

- 130 ms average lookup time and much slower on mobile.. Examples: *fonts, analytics, social buttons, etc*.

Avoid redirects

- Often results in new handshake (and maybe even DNS). Avoid “m dot”

Make fewer HTTP requests

- No request is faster than no request.
- Avoid external blocking JavaScript and CSS in ATF content (inline it)
- AFT content <14Kb (only 1 RTT if server’s updated to a new TCP standard IW10)

Optimize JS execution and rendering time

```
<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 critical styles
2. Do you need the JS at all?
    - ○ Progressive enhancement
    - ○ Inline critical JS code
    - ○ Defer the rest

```
<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

Enable compression

- Apache: Use mod_deflate
- Nginx: Use ngx_http_gzip_module

Leverage browser caching

- **Cache-Control** defines how, and for how long the individual response can be cached by the browser and other intermediate caches.
- **ETag** provides a revalidation token that is automatically sent by the browser to check if the resource has changed since the last time it was requested.

Minify and combine resources

- For **HTML**, you can use i.e. PageSpeed Insights Chrome Extension
- For **CSS**, you can try YUI Compressor and cssmin.js
- For **JavaScript**, try the Closure Compiler, JSMin or the YUI Compressor

Optimize Images

- Prefer vector formats
- Minify and compress SVG assets
- Pick best raster image format
- Experiment with optimal quality settings for raster formats
- Remove unnecessary image metadata
- Serve scaled images

Optimize CSS Delivery

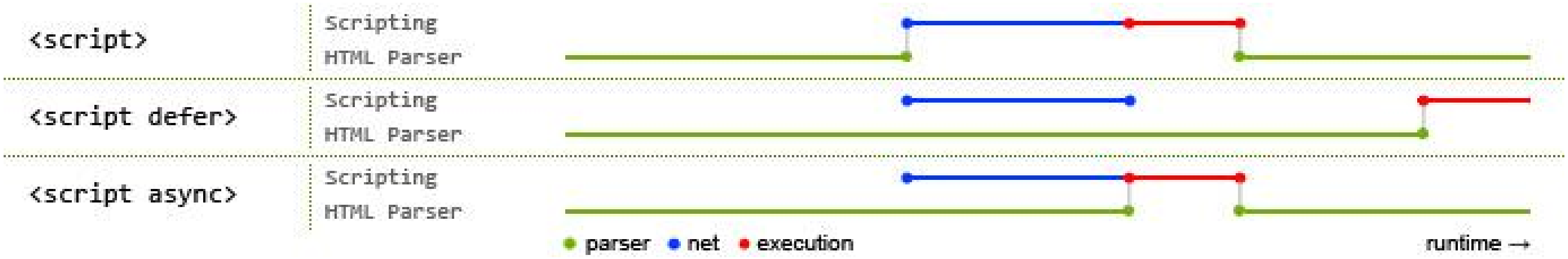
- Inline small css
- Don't inline large data URIs
- Don’t inline CSS attributes

```
var cb = function() {
  var l = document.createElement('link'); l.rel = 'stylesheet';
  l.href = 'small.css';
  var h = document.getElementsByTagName('head')[0]; h.parentNode.insertBefore(l, h);
};
var raf = requestAnimationFrame || mozRequestAnimationFrame ||
    webkitRequestAnimationFrame || msRequestAnimationFrame;
if (raf) raf(cb);
else window.addEventListener('load', cb);
```

Prioritize visible content

- Structure your HTML to load the critical, above-the-fold content first
- Reduce the amount of data used by your resources

Use asynchronous scripts



- regular - block on HTTP request, parse, execute, proceed
- defer - delaying script execution until the HTML parser has finished
- async - download in background, execute when ready

`Asyc` *is supported in IE10+ only*
`Defer` *has issues in IE8/9*

Optimize in-app performance

- **Performance == 60 FPS**
  - 16.6 ms budget per frame
  - Shared budget for your code, GC, layout, and painting
  - Use frames view to hunt down and eliminate jank
- **Profile and optimize your code**
  - Profile your JavaScript code
  - Profile the cost of layout and rendering!
- **Eliminate JS and DOM memory leaks**
  - Monitor and diff heap usage to identify memory leaks
- **Test on mobile devices**
  - Emulators won't show you true performance on the device

SPEED IS A FEATURE.

IT’S A DISCIPLINE.

IT’S A CONTINUOUS EFFORT.