

Web Performance

Steve Kinney

**Hi, I'm Steve.
(@stevekinney)**





SendGrid Marketing Campaign x Steve

Secure | https://sendgrid.com/marketing_campaigns/ui/campaigns/1917514/edit

DESIGN PREVIEW Save Draft Send Campaign

Settings Build Tags A/B Testing T T Formatted Font Size B I U T

From: Subject: Preheader:

BUTTON

Button Color: #000000

Border Color: #000000

Font Color: #FFFFFF

Width: AUTO %

Height: 16 px

Padding: + 12 px - 18 px + 12 px - 18 px

Border Radius: 6 px

Border Width: 1 px

Font Family: Arial

Font Size: 16 px

Font Weight: normal

This is my awesome button

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam tincidunt elementum sem non luctus. Ut dolor nisl, facilisis non magna quis, elementum ultricies tortor. In mattis, purus ut tincidunt egestas, ligula nulla accumsan justo, vitae bibendum orci ligula id ipsum. Nunc elementum tincidunt libero, in ullamcorper magna volutpat a.

mobx (1).svg react.svg mobx.svg flux.svg Show All X

Prologue

What is performance and why
does it matter?

Let's answer those in the
opposite order, actually.

**Why does performance
matter?**

Allow me to use some slides with
too many words and then read them
to you. *#thoughtleadership101*

“
0.1 second is about the limit for having the user feel that the system is reacting instantaneously, meaning that no special feedback is necessary except to display the result. —**Jakob Nielsen**

”

“
1.0 second is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. Normally, no special feedback is necessary during delays of more than 0.1 but less than 1.0 second, but the user does lose the feeling of operating directly on the data. —**Jakob Nielsen**



10 seconds is about the limit for keeping the user's attention focused on the dialogue. For longer delays, users will want to perform other tasks while waiting for the computer to finish, so they should be given feedback indicating when the computer expects to be done. Feedback during the delay is especially important if the response time is likely to be highly variable, since users will then

If your user interface takes 10 seconds or *more* to respond to an interaction, then we should talk.

And now: I'll show you some statistics
to make myself sound smart.

Aberdeen Group found that a 1 second slow down resulted 11% fewer page views, 7% less conversion. [Source](#).

Akamai found that two-second delay in web page load time increase bounce rates by 103 percent. [Source](#).

A 400 millisecond improvement in performance resulted in a **9% increase** in traffic at Yahoo. [Source](#).

Google found that a 2% slower page resulted in 2% fewer searches, which means 2% fewer ads shown. [Source](#).

100 millisecond improvement in performance results in 1% increase in overall revenue at **Amazon**. [Source](#).

53% of users will leave a mobile site if it takes more than 3 secs to load. [Source](#).

(One more thing...)

According to research, if you want user to feel like your site is faster than your competitors, you need to be **20% faster**.

At the same time...

Our applications are
getting bigger.



Source: <https://twitter.com/xbs/status/626781529054834688>

MEDIAN DESKTOP
1699.0 KB
▲251.4%

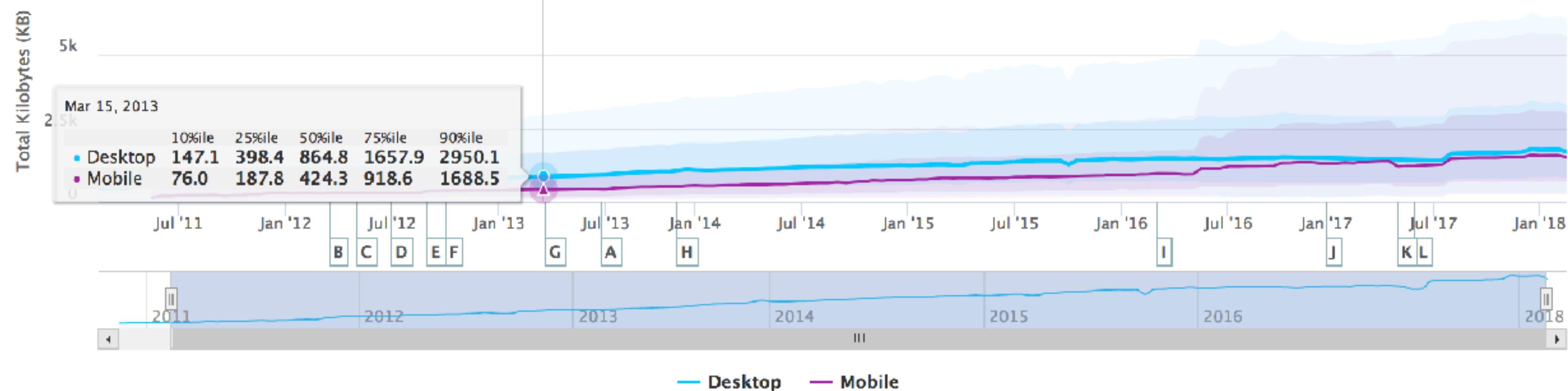
MEDIAN MOBILE
1524.1 KB
▲952.6%

Timeseries of Total Kilobytes

Source: httparchive.org

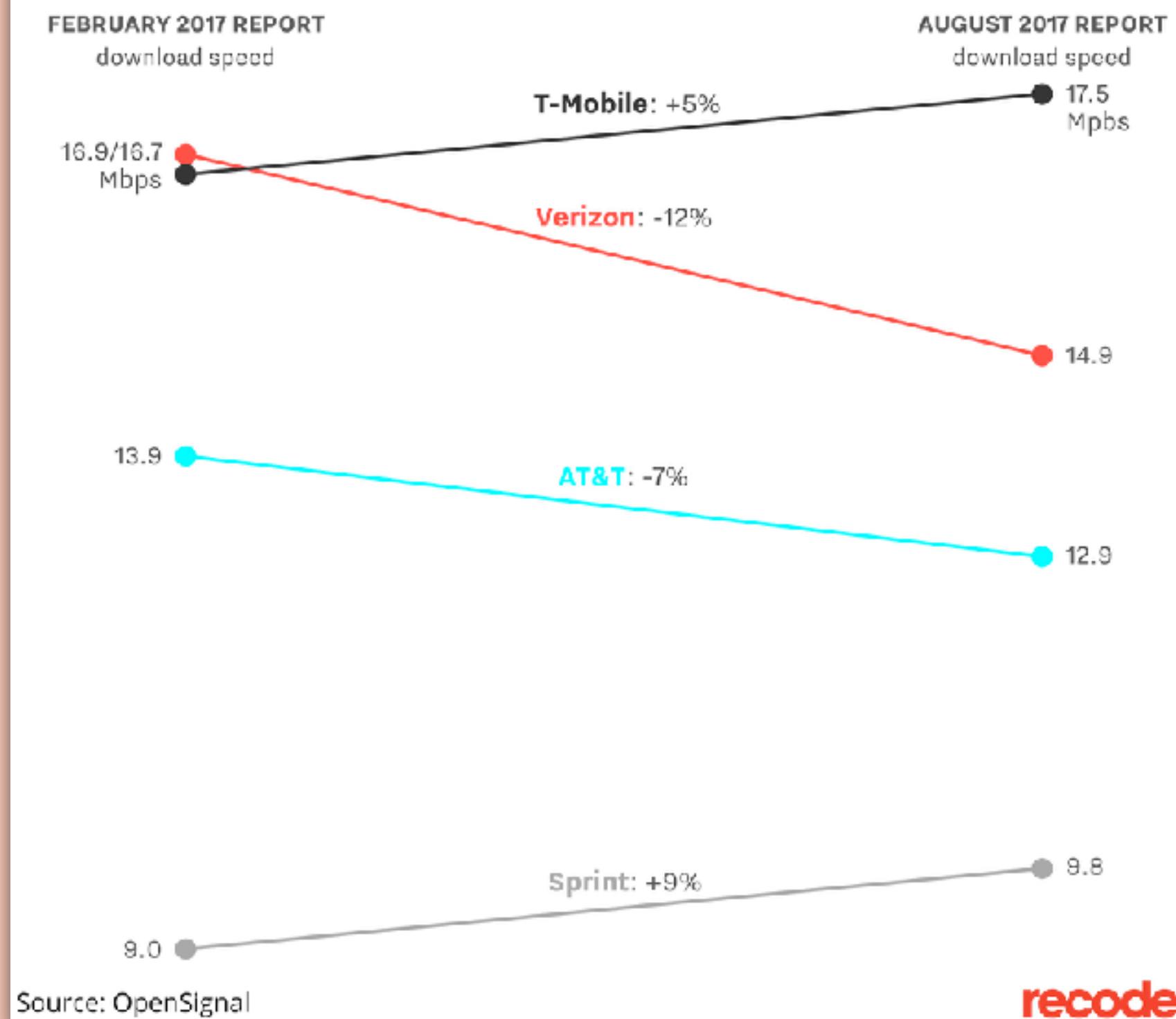
Zoom **1m** 3m 6m YTD 1y All

From **Feb 11, 2011** To **Feb 15, 2018**



LTE is actually getting
slower.

AT&T and Verizon LTE mobile download speeds have declined thanks to unlimited data plans



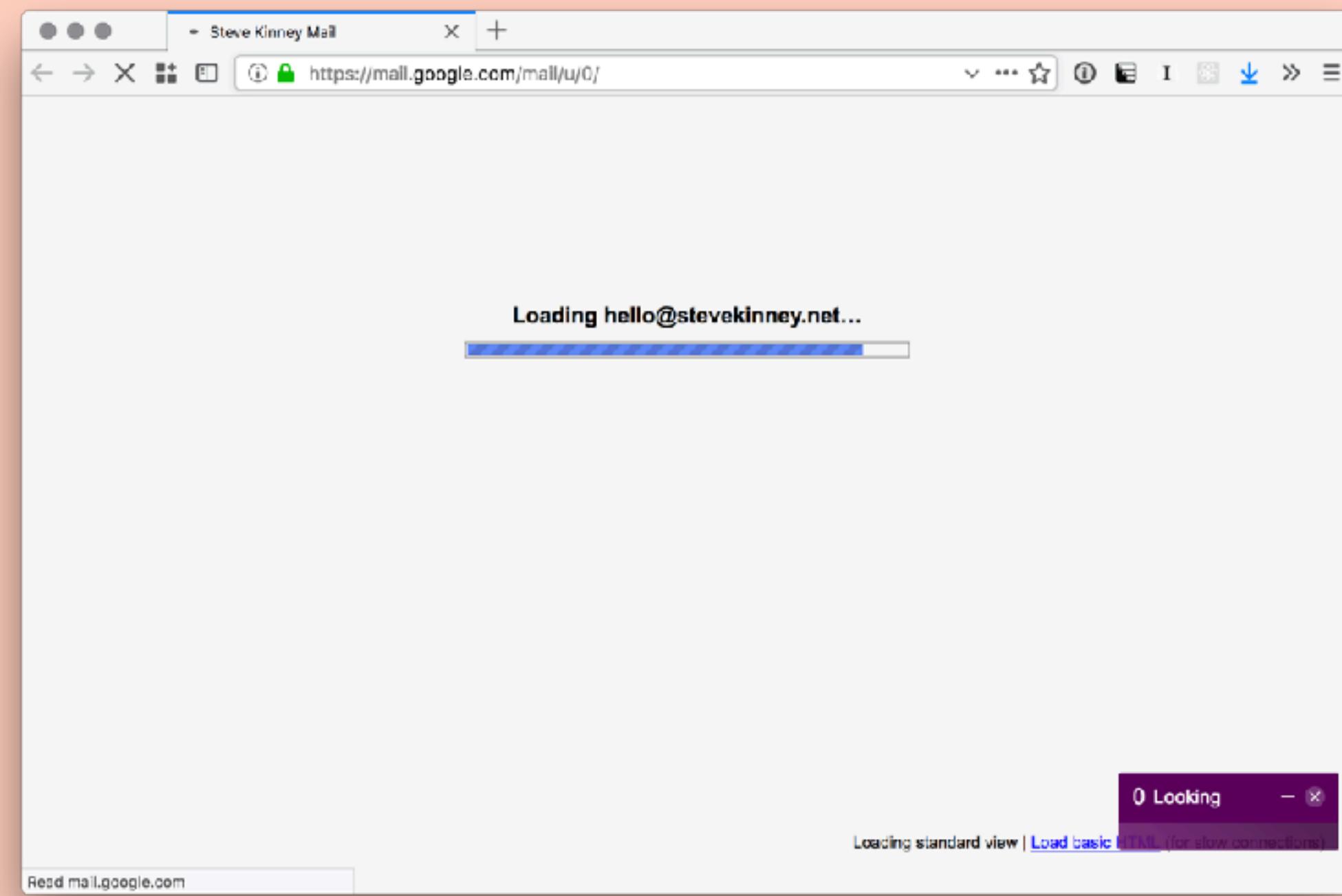
Source: <https://www.recode.net/2017/8/2/16069642/verizon-att-tmobile-sprint-mobile-customers-slow-speeds-unlimited-data-plan>

Thinking About Performance

**Are all of our needs the
same?**

The New York Times

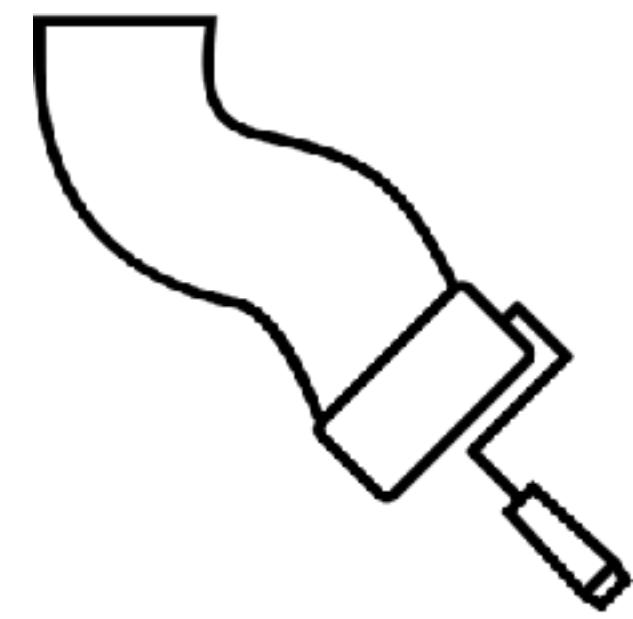




There are different kinds
of performance.



</>



RAIL: Some numbers to
think about.

Wetware perception-reaction times

Delay	User reaction
0 - 100 ms	Instant
100 - 300 ms	Slight perceptible delay
300 - 1000 ms	Task focus, perceptible delay
1000+ ms	Mental context switch
10,000+ ms+	I'll come back later...

Perception-reaction times vary by task, environment, and other factors. That said, above numbers are a good reference for (web) app use cases.



[Speed, performance and human perception](#)

Response	Animation	Idle	Load
Tap/click to paint in less than 100 milliseconds.	Each frame completes in less than 16 milliseconds.	Use idle time to proactively schedule work.	Satisfy the “response” goals during full load.
	Drag to paint in less than 16 milliseconds.	Complete that work in 50-millisecond chunks.	Get first meaningful paint in 1,000 milliseconds.

**Disclaimer: We're not going
to obsess over numbers.**

It's about getting 10%
better.

“*Strategies for Optimizing Web Performance When, Honestly, You Have Like 5 Meetings Today and You Need to Choose the Correct Hills Upon Which to Strategically Die*” – **Romeeka Gayhart, @CCandUC**



What matters to you?

- The New York Times might care about time to first headline.
- Twitter might care about time to first tweet.
- Chrome might care about time to inspect element.
- What does your product or project care about?

Measure First

“
Measure. Don’t tune for speed until you’ve measured, and even then don’t unless one part of the code overwhelms the rest. —**Rob Pike**

”

Do not go just blindly applying
performance optimizations.

There is a cost to every abstraction
—and everything has a trade off.

I'm not a fan of premature optimization, but performance is one of those things where if we're not keeping an eye on it, it has a chance of getting away from us.

Some things to think about while measuring

- Are we testing performance on fancy MacBook Pros or consumer-grade hardware?
- Are we simulating less-than-perfect network conditions.
- What is our performance budget?

Don't get carried away
with measuring, either.

Thinking deeply about the architecture
and design of your application is a better
use of your time than micro-benchmarks.

Three Tiers of Advice

- Definitely do this.
- Maybe do this, but measure before and after.
- Only do this if you find a performance problem that needs solving.

**And now: Steve's Golden
Rule of Performance**

**Doing Less Stuff Takes
Less Time.**

**What does that even
mean?**

Steve's Second Rule of Performance:
If you can do it later. Do it later.

Rough Outline

- **JavaScript performance:** Write code that runs faster, later, or not at all.
- **Rendering performance:** It turns out most of our JavaScript happens in the browser, which has its own performance concerns.
- **Load performance:** Until the user actually gets the page, there isn't much to optimize.

Let's get into it.

Chapter One

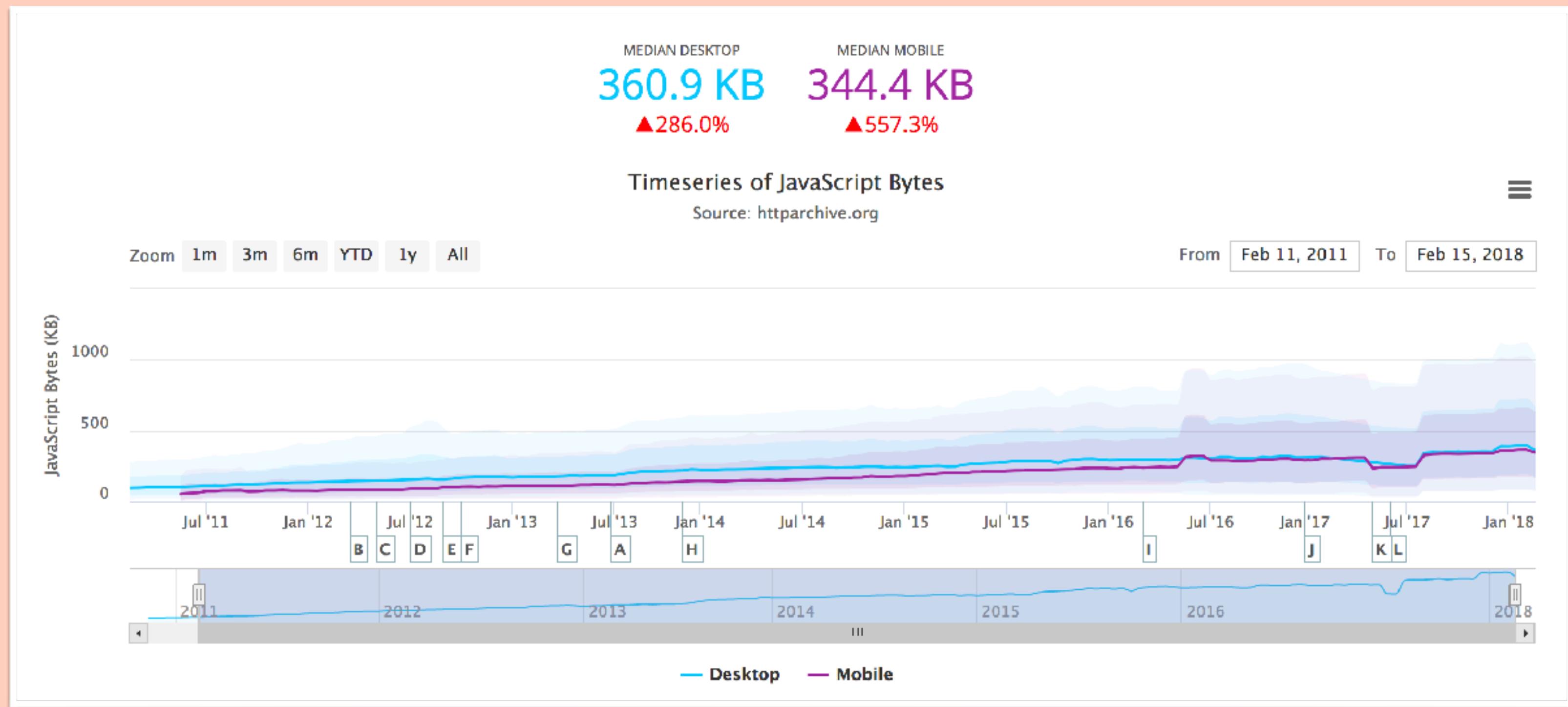
JavaScript Performance.

Problem: You literally can't buy faster servers to improve performance of client-side applications.

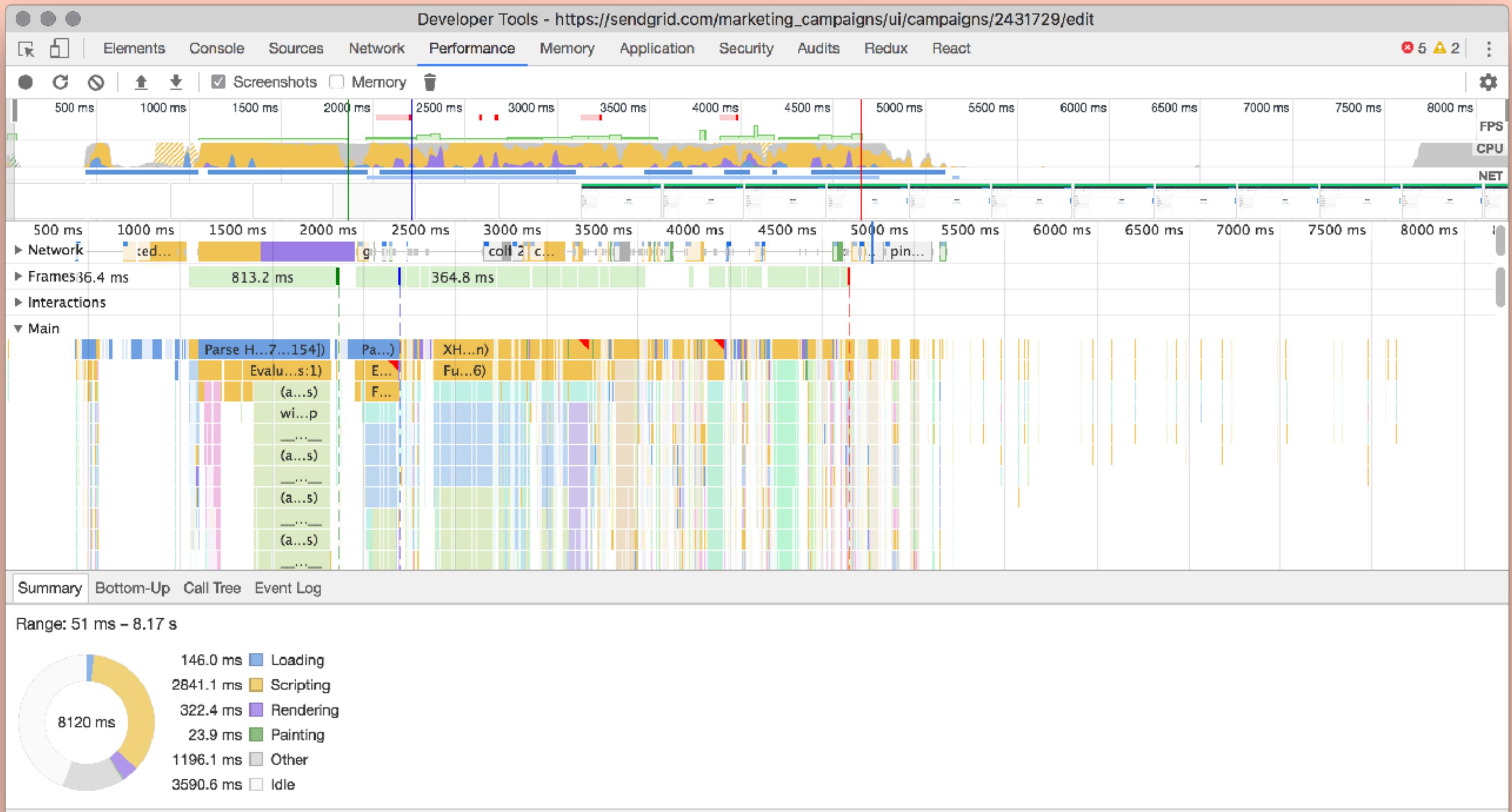
I mean, you could buy all of your customers faster computers, I guess.



A lot of time and energy is spent compressing assets, removing requests, and reducing latency, but what about once the application is running?



Sometimes, parsing and
compiling is the real culprit.



**Okay, so how does
JavaScript even work?**

Fun fact: JavaScript is a
compiled language.

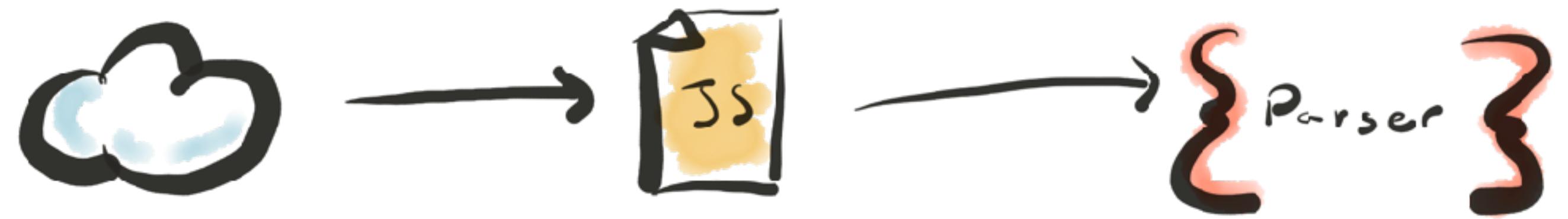
Most browsers use something called just-in-time (JIT) compilation.

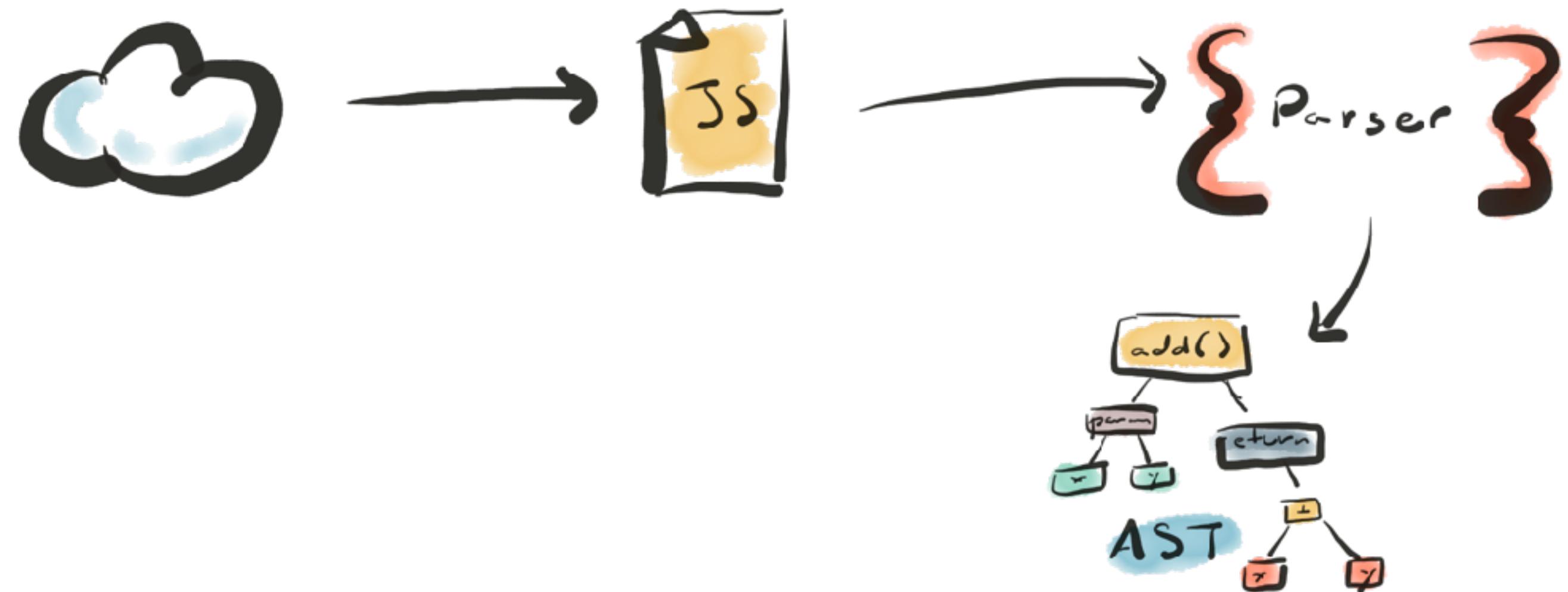
Things to know about JIT compilation

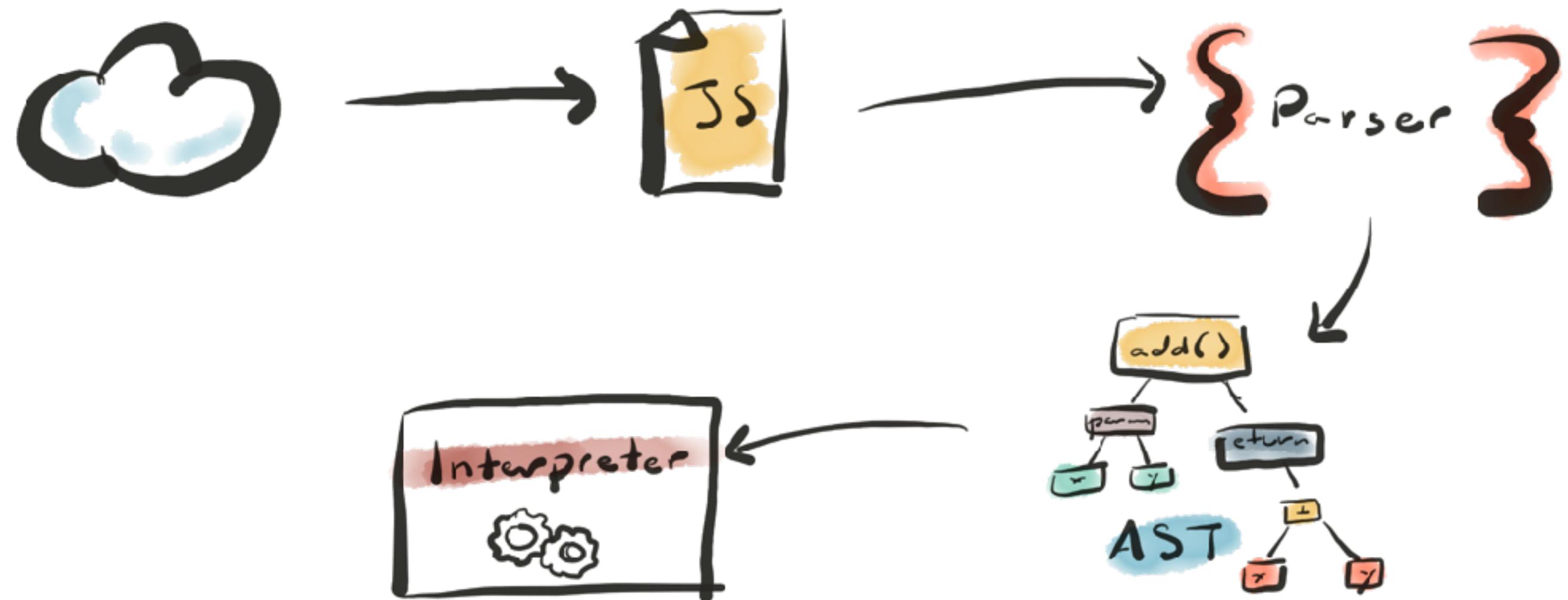
- It means that there is compilation step.
- It means that it happens moments before execution.
- That means it happens on our client's machine.
- That means they're paying the cost and/or doing the hard work for us.

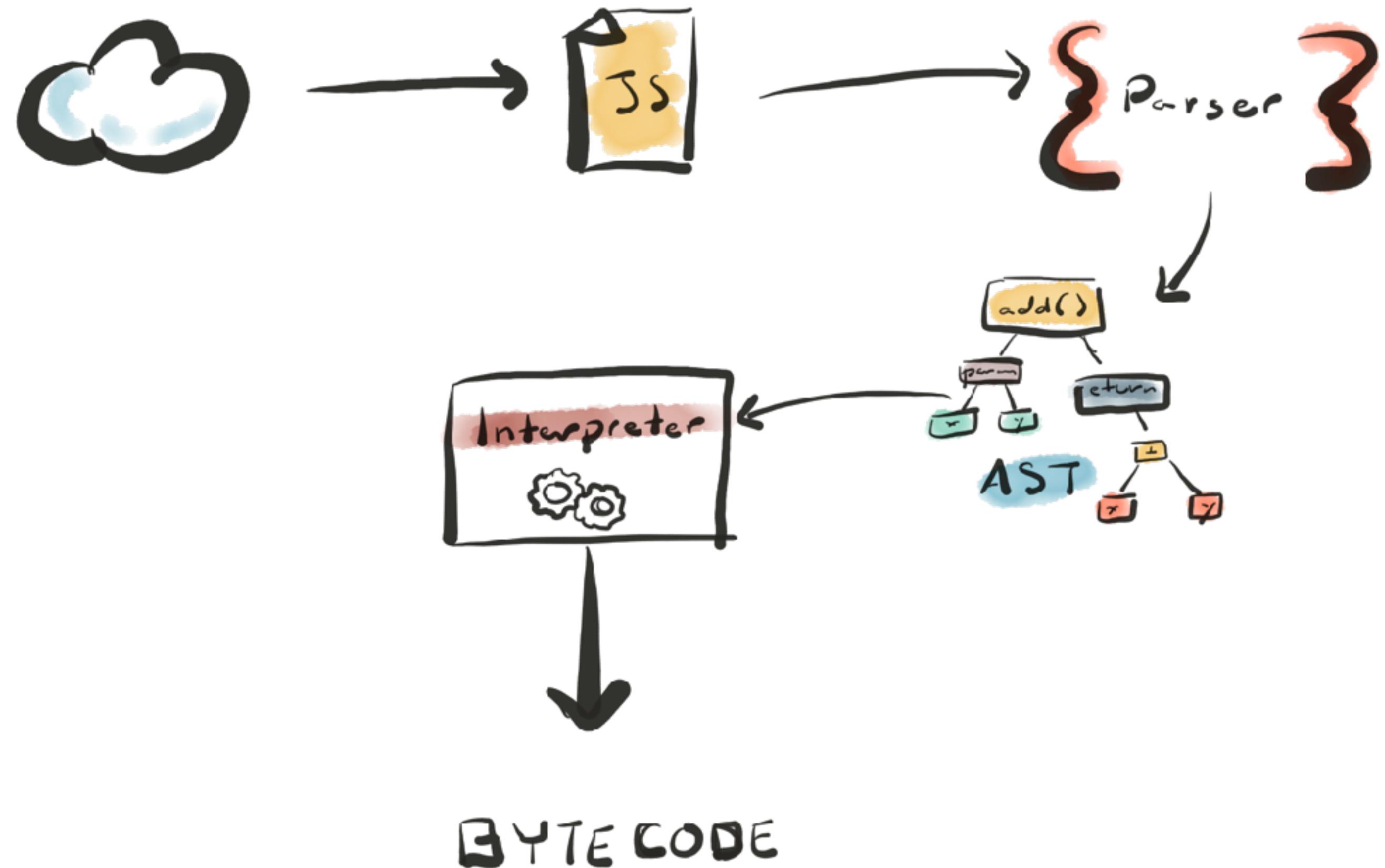
Let's look at your code's journey
through V8 at a high level.

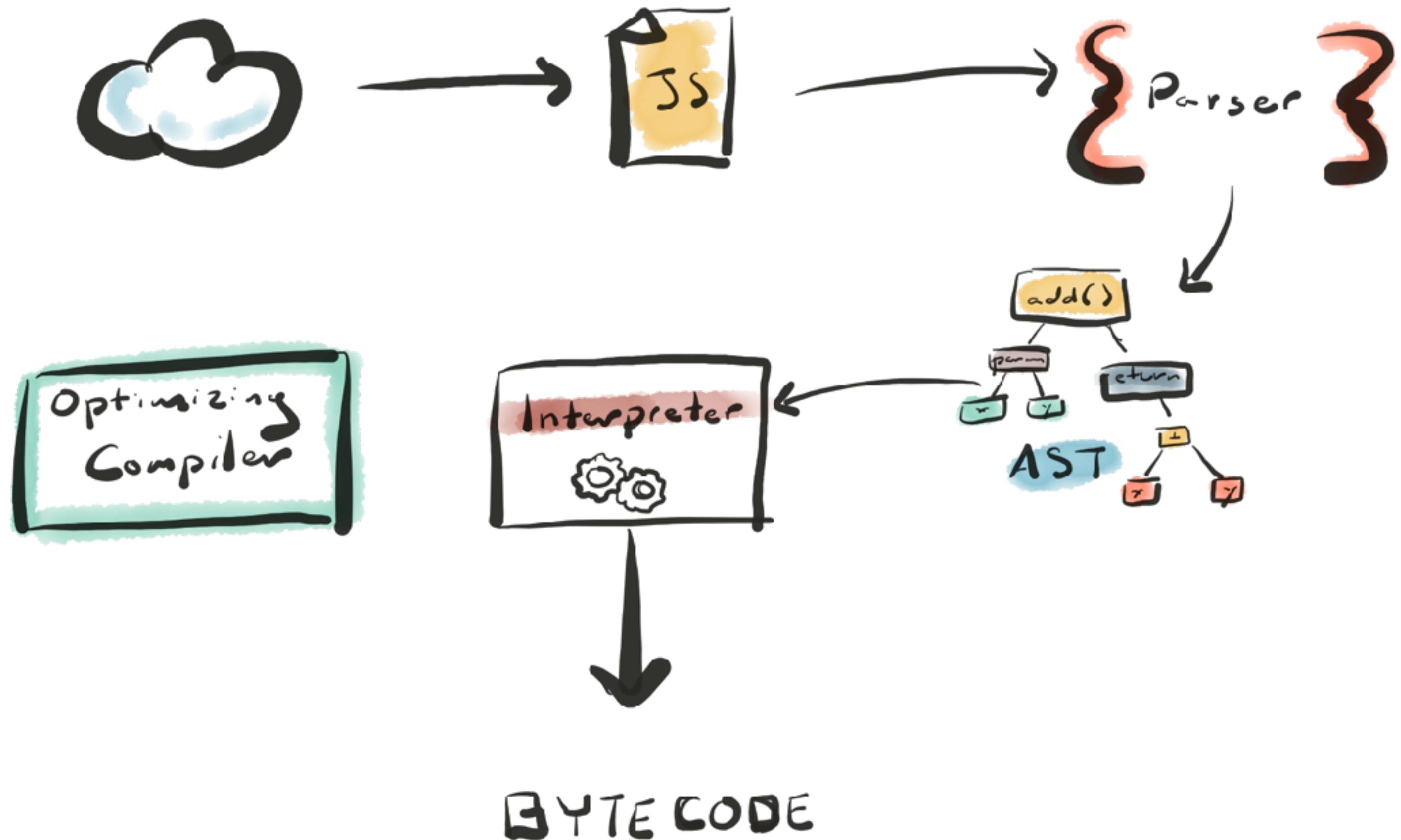


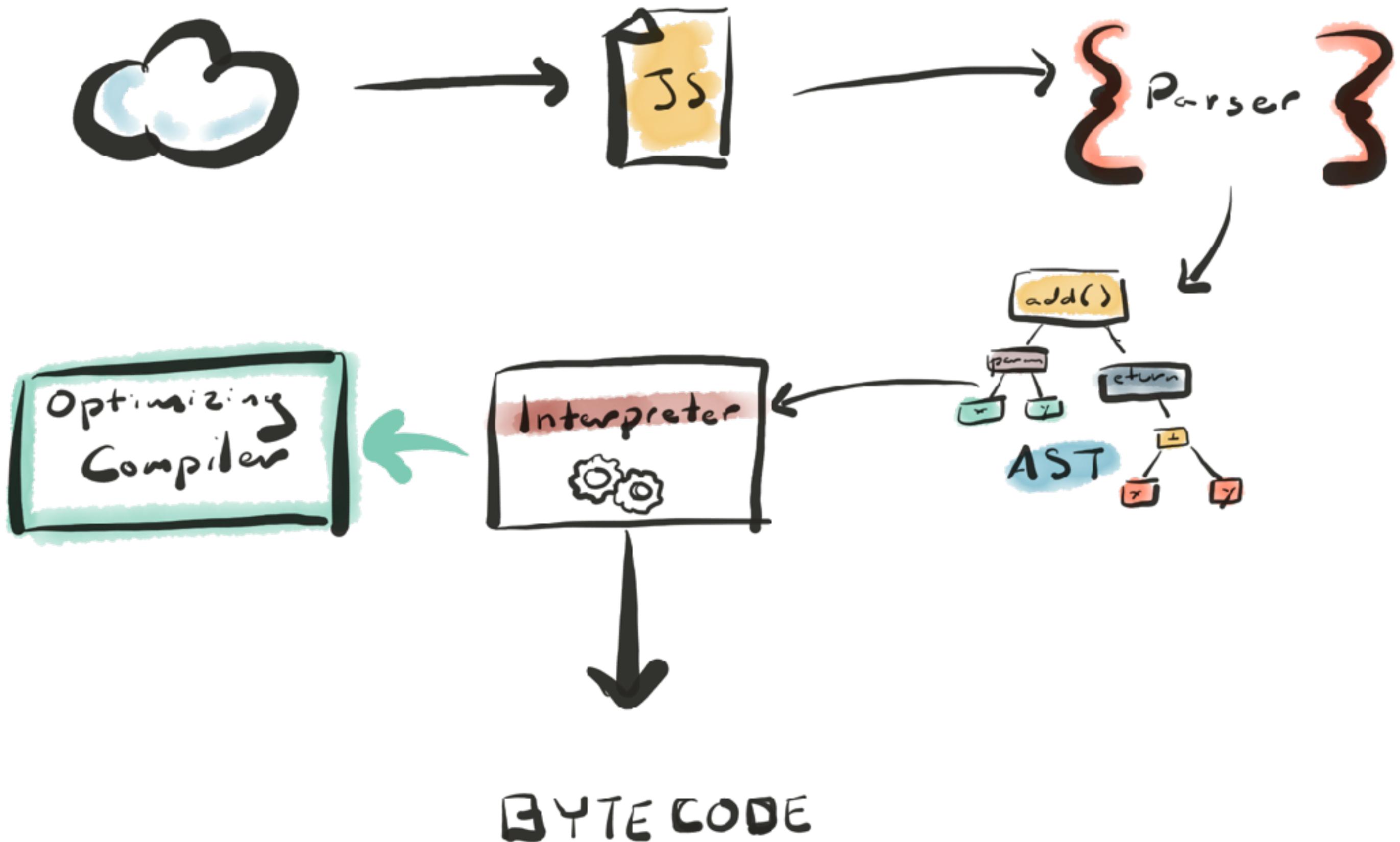


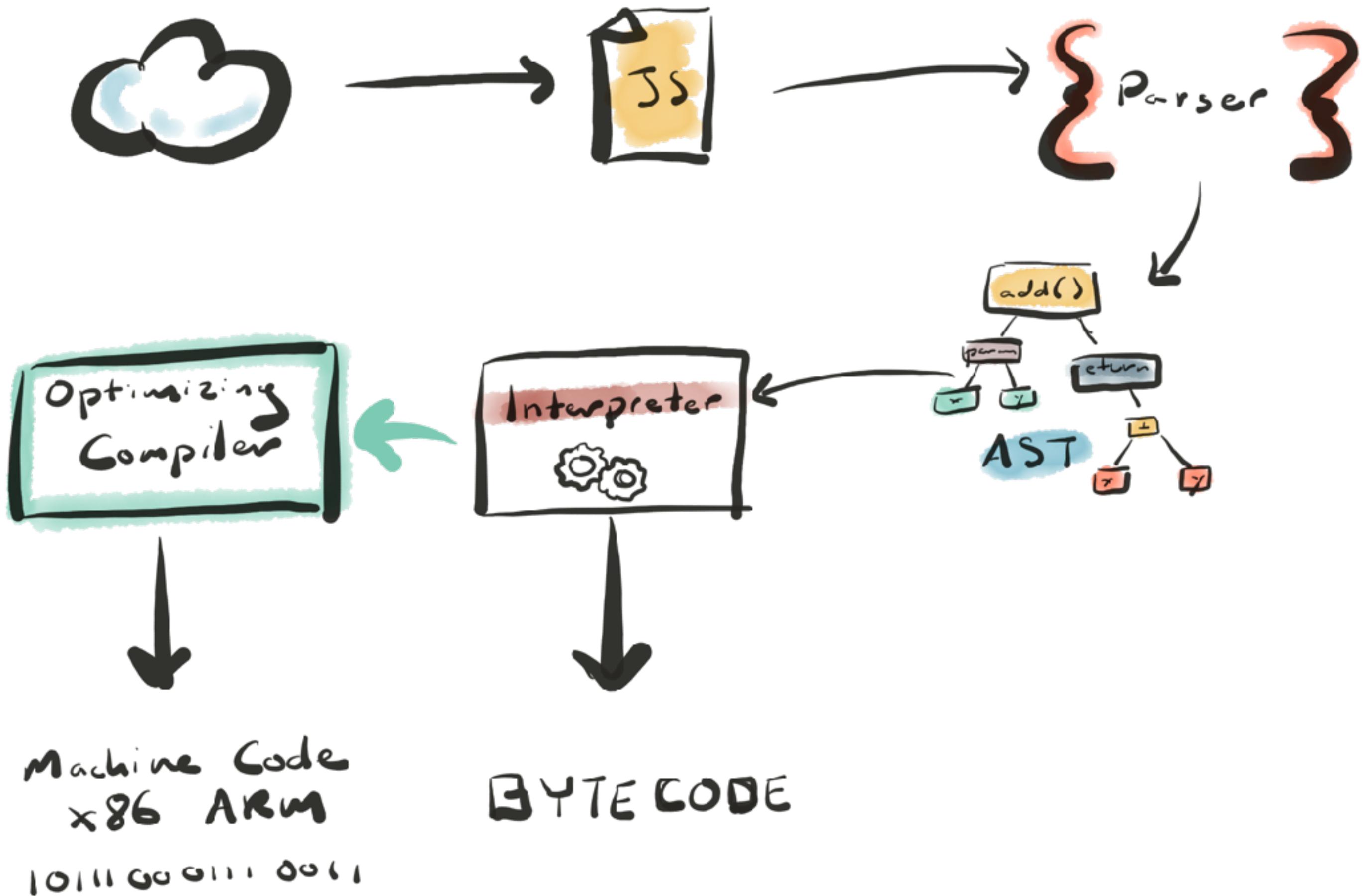


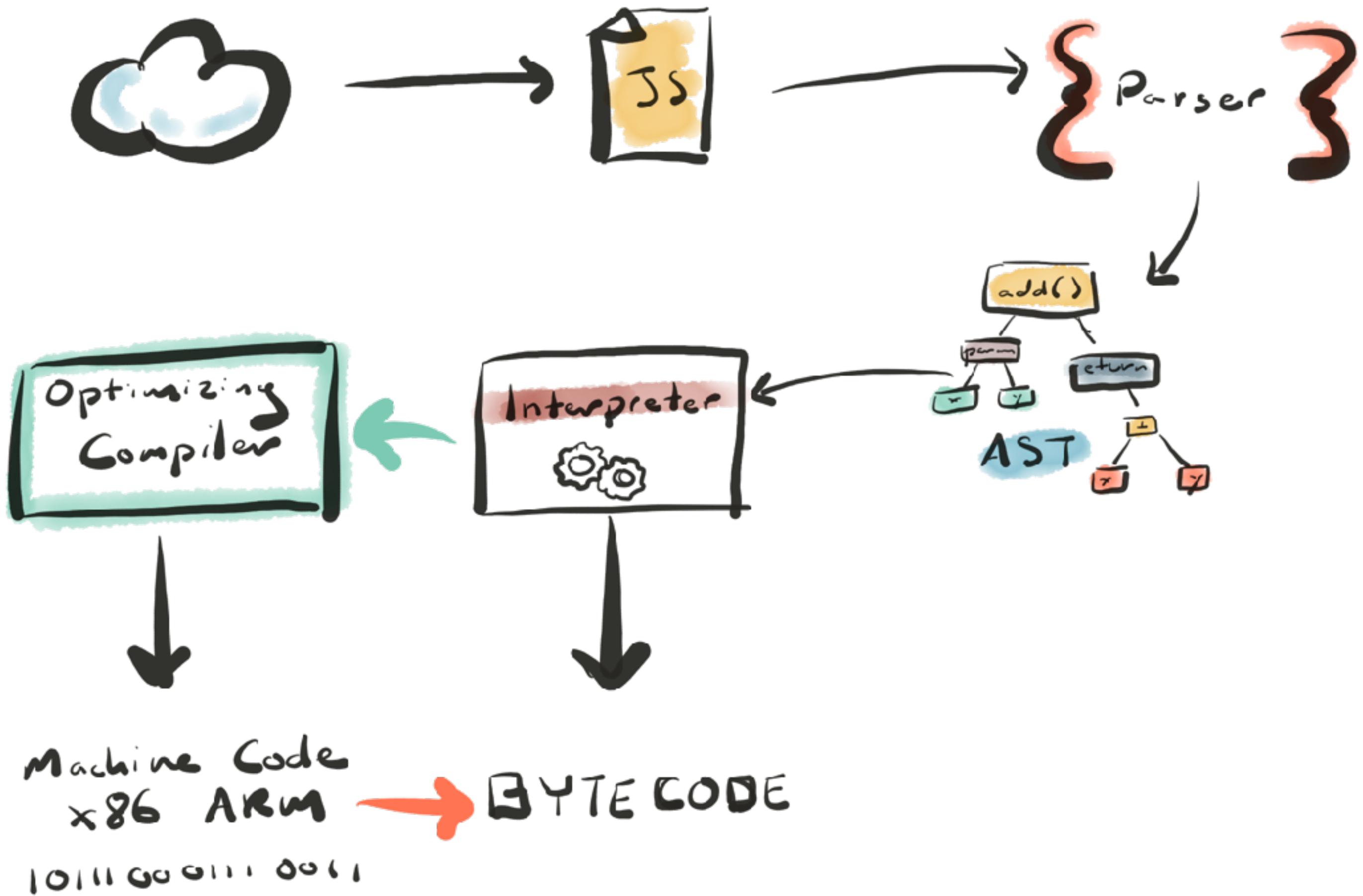












Parsing

The source code is the true intention
of the application, but the engine
needs to figure out what this means.

Parsing can be slow.

As slow as **1MB/s** on mobile.

One way to reduce parsing time
is to have less code to parse.

Another way is to do as much parsing as you need and as little as you can get away with.

Parsing happens in two phases

- **Eager** (full parse): This is what you think of when you think about parsing.
- **Lazy** (pre-parse): Do the bare minimum now. We'll parse it for realies later.

Generally speaking, this is
a *good thing*™.

Doing less work is faster
than doing work, right?

The basic rules

- Scan through the top-level scope. Parse all the code you see that's actually doing something.
- Skip things like function declarations and classes for now. We'll parse them when we need them.

*This could bite you. But,
how?*

```
// These will be eagerly-parsed.  
const a = 1;  
const b = 2;  
  
// Take note that there a function here,  
// but, we'll parse the body when we need it.  
function add(a, b) {  
    return x + y;  
}  
  
add(a, b); // Whoa. Go back and parse add()!
```

**Do you see the problem
here?**

Corollary: Doing stuff twice
is slower than doing it once.

```
const a = 1;  
const b = 2;
```

```
// Parse it now!  
(function add(a, b) {  
    return x + y;  
});
```

```
add(a, b);
```

It's definitely helpful to
know how this works, but...

micro-optimization (*noun*): Thing you read about one time and you know pester your co-works about in code reviews, even though it has an almost unnoticeable impact at scale.

“

“But, Steve! These little things add up!”

— Me (pretending to be you), just now.

”

nolanlawson/optimize-js: Optim... Steve

GitHub, Inc. [US] | https://github.com/nolanlawson/optimize-js

This repository Search Pull requests Issues Marketplace Explore

nolanlawson / optimize-js Watch 70 Star 3,557 Fork 110

Code Issues 15 Pull requests 6 Projects 0 Wiki Insights

Optimize a JavaScript file for faster initial load by wrapping eagerly-invoked functions
<https://nolanlawson.github.io/optimiz...>

98 commits 4 branches 4 releases 12 contributors Apache-2.0

Branch: master New pull request Create new file Upload files Find file Clone or download

nolanlawson update readme ... Latest commit e8ceaid on Feb 27, 2017

benchmarks	(#37) - update benchmarks to more accurately include compilation time	a year ago
bin	(#37) - update benchmarks to more accurately include compilation time	a year ago
lib	Browserify-specific fix for #29 (#36)	a year ago
test	Add more browserify tests (#39)	a year ago
.gitignore	fix code coverage	2 years ago
.travis.yml	update travis file	2 years ago
LICENSE	initial commit	2 years ago
README.md	update readme	a year ago

**And now: An exploration of
why measuring is important.**

Lab

[https://nolanlawson.github.io/
test-optimize-js/](https://nolanlawson.github.io/test-optimize-js/)

Chrome 55, Windows 10 RS1, Surfacebook i5

Script	Original	Optimized	Improvement	Minified	Min+Optimized	Improvement
Create React App	55.39ms	51.71ms	6.64%	26.12ms	21.09ms	19.26%
ImmutableJS	11.61ms	7.95ms	31.50%	8.50ms	5.99ms	29.55%
jQuery	22.51ms	16.62ms	26.18%	19.35ms	16.10ms	16.80%
Lodash	20.88ms	19.30ms	7.57%	20.47ms	19.86ms	3.00%
PouchDB	43.75ms	20.36ms	53.45%	36.40ms	18.78ms	48.43%
ThreeJS	71.04ms	72.98ms	-2.73%	54.99ms	39.59ms	28.00%
Overall improvement: 20.63%						

Edge 14, Windows 10 RS1, SurfaceBook i5

Script	Original	Optimized	Improvement	Minified	Min+Optimized	Improvement
Create React App	32.46ms	24.85ms	23.44%	26.49ms	20.39ms	23.03%
ImmutableJS	8.94ms	6.19ms	30.74%	7.79ms	5.41ms	30.55%
jQuery	22.56ms	14.45ms	35.94%	16.62ms	12.99ms	21.81%
Lodash	22.16ms	21.48ms	3.05%	15.77ms	15.46ms	1.96%
PouchDB	24.07ms	21.22ms	11.84%	39.76ms	52.86ms	-32.98%
ThreeJS	43.77ms	39.99ms	8.65%	54.00ms	36.57ms	32.28%
Overall improvement: 13.52%						

Safari 10, macOS Sierra, 2013 MacBook Pro i5

Script	Original	Optimized	Improvement	Minified	Min+Optimized	Improvement
Create React App	31.60ms	31.60ms	0.00%	23.10ms	23.50ms	-1.73%
ImmutableJS	5.70ms	5.60ms	1.75%	4.50ms	4.50ms	0.00%
jQuery	12.40ms	12.60ms	-1.61%	10.80ms	10.90ms	-0.93%
Lodash	14.70ms	14.50ms	1.36%	11.10ms	11.30ms	-1.80%
PouchDB	14.00ms	14.20ms	-1.43%	11.70ms	12.10ms	-3.42%
ThreeJS	35.60ms	36.30ms	-1.97%	27.50ms	27.70ms	-0.73%
Overall improvement: -1.04%						

Ninjas Practicing Multidimensionality

npm Enterprise features pricing documentation support

find packages

sign up or log in

Share your code. npm Orgs help your team discover, share, and reuse code. [Create a free org »](#)

★ **optimize-js-plugin** public

Webpack plugin that uses [optimize-js](#)

Thanks to [@nolanlawson](#) for his awesome work.

Install

```
npm i --save-dev optimize-js-plugin
```

[npm i optimize-js-plugin](#)
[how? learn more](#)

[vignesh.shanmugam](#) published a year ago

0.0.4 is the latest of 4 releases

[github.com/vigneshshanmugam/optimize-js-p...](#)

[MIT](#)

Collaborators [list](#)

Usage

```
// webpack.config.js
```

LTE Speeds.png

High Performance Java...pdf

Show All

Stats

Try to avoid nested functions

```
function sumOfSquares(x, y) {  
    // 👉 This will repeatedly be parsed.  
    function square(n) {  
        return n * n;  
    }  
  
    return square(x) + square(y);  
}
```

Better...

```
function square(n) {  
    return n * n;  
}
```

```
function sumOfSquares(x, y) {  
    return square(x) + square(y);  
}
```

**Okay, cool—so it's parsed.
Now what?**

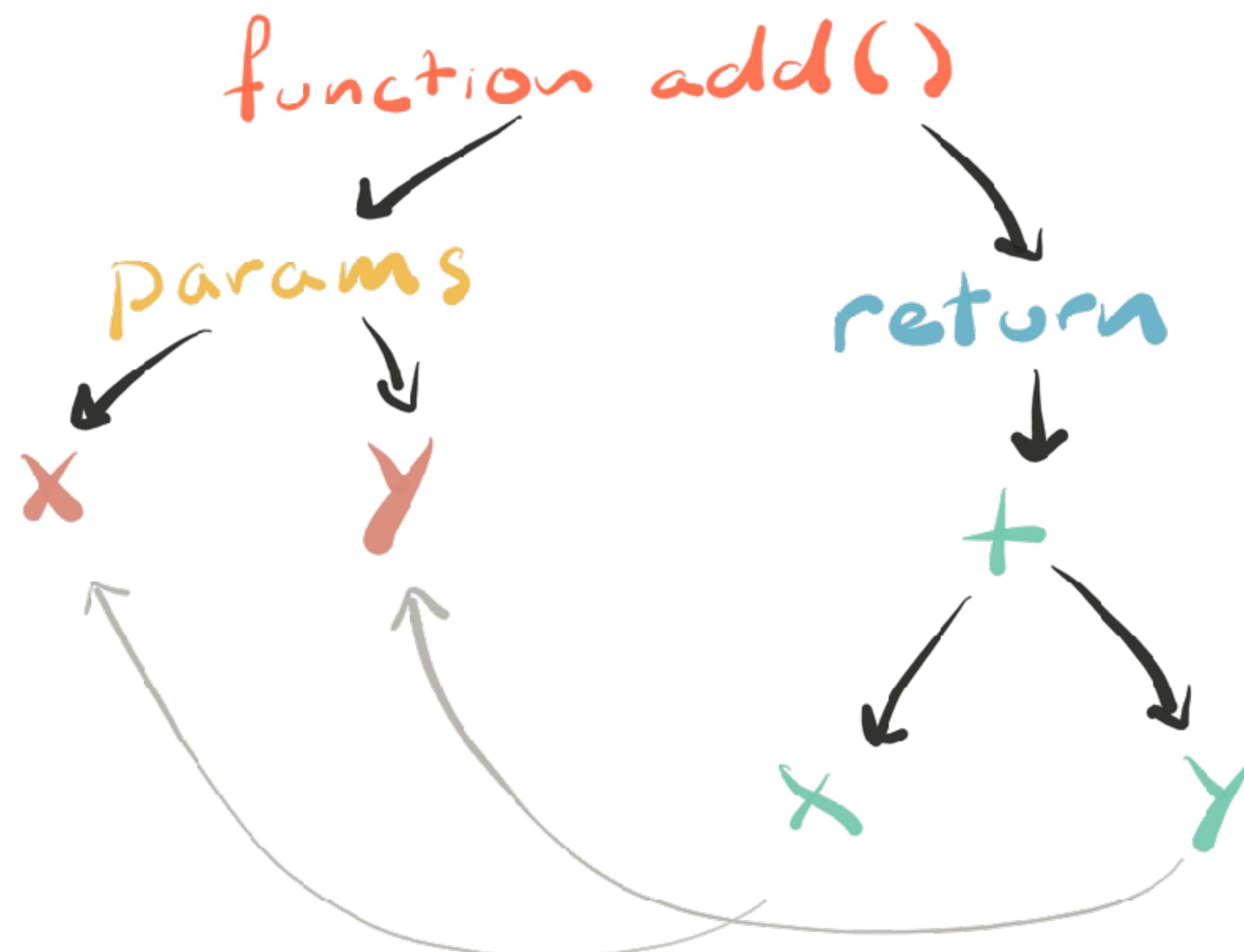
It's turned into an abstract
syntax tree.

“
In computer science, an abstract syntax tree (AST) [...] is a tree representation of the abstract syntactic structure of source code written in a programming language. —

Wikipedia

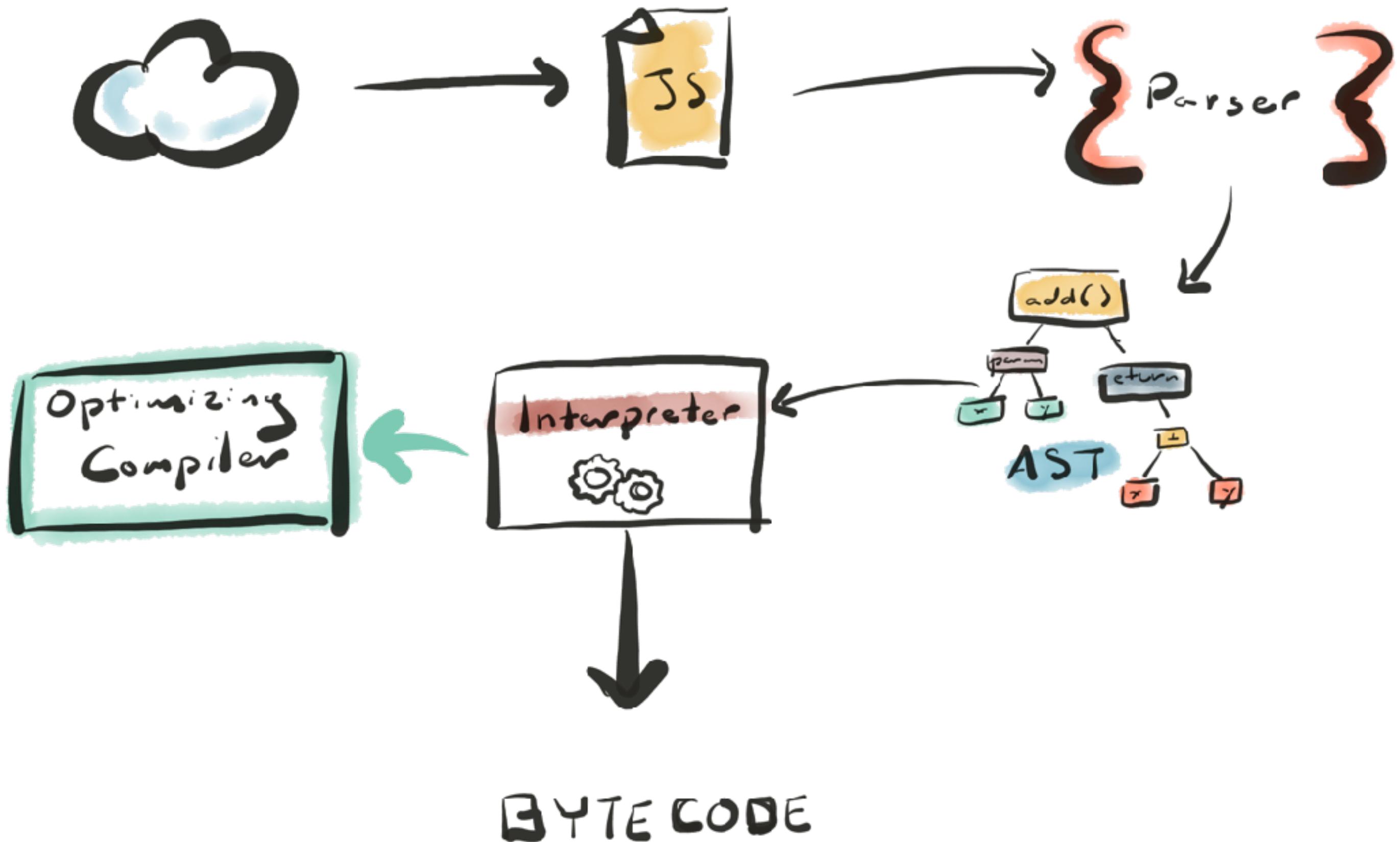
”

Essential, we've gone from a big long string of text to an actual data structure representing our code.



**With our AST, we now have everything
we need to make byte code!**

The baseline compiler takes the AST
and starts to execute our code as
we wrote it.



Three things the engine does to help you out

- Speculative optimization
- Hidden classes for dynamic lookups
- Function inlining

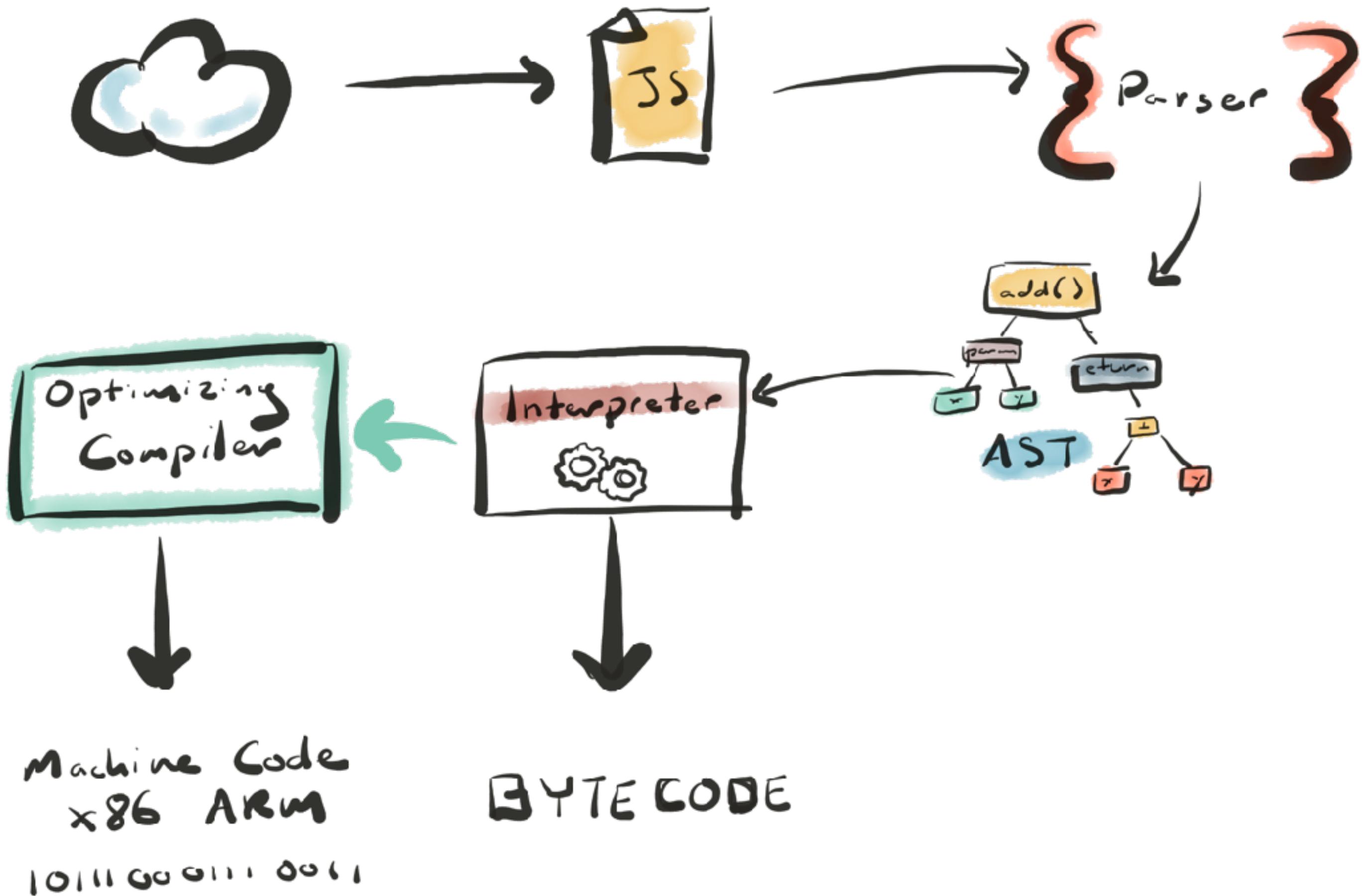
It turns out that JavaScript
is hard.

It also turns out that
JavaScript is dynamic.

But, what if we made some assumptions based on what we've seen in the past?

Play Time

```
function add(a, b) {  
    return x + y;  
}
```



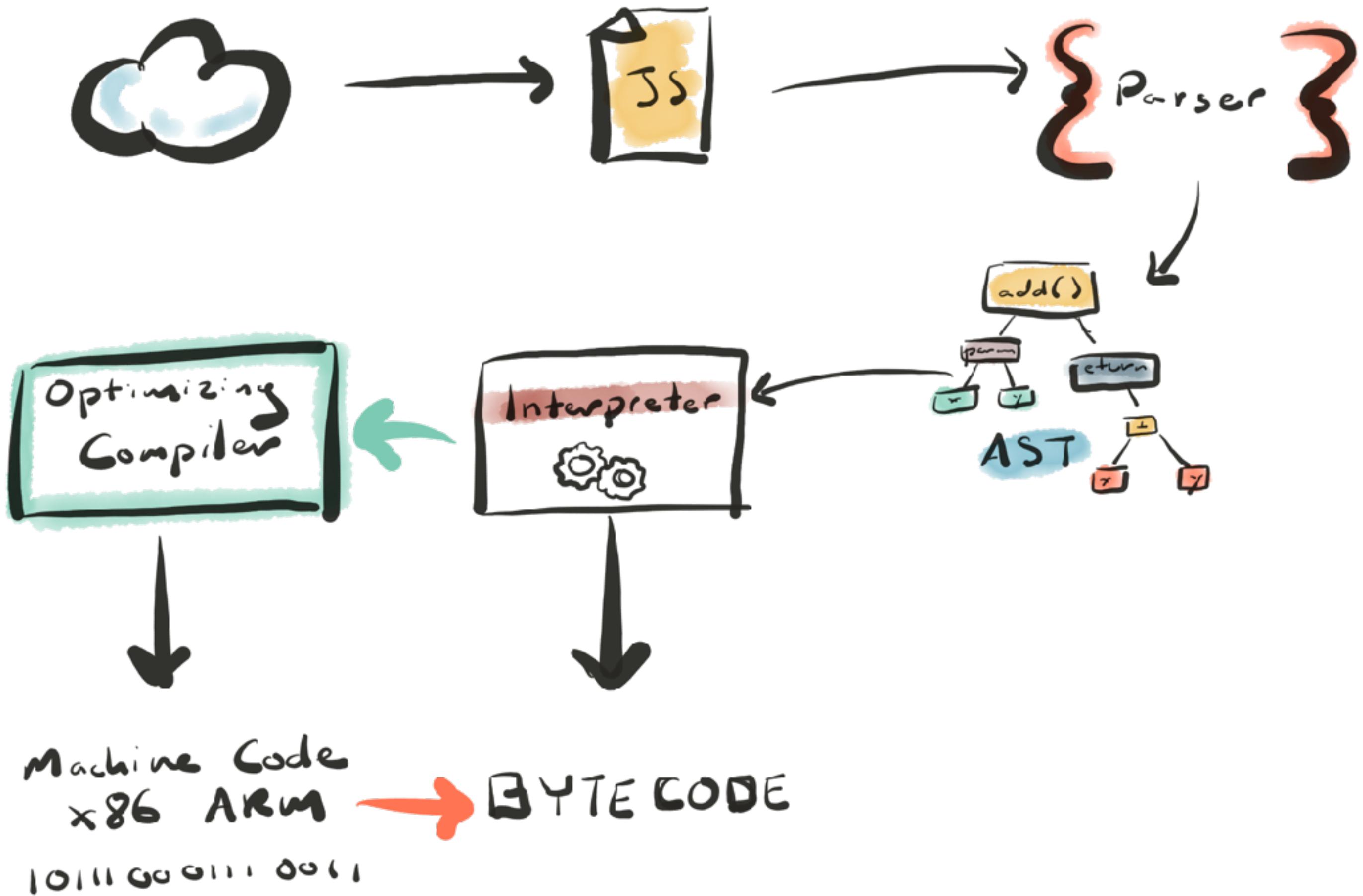
We use a system called
speculative optimization.

How does this work?

- We use an interpreter because the optimizing compiler is slow to get started.
- Also: it needs some information before it knows what work it can either optimize or skip out on all together.
- So, the interpreter starts gathering feedback about what it sees as the function is used.

But what if a string slips in
there?

Play Time



The optimizing compiler optimizes
for what it's seen. If it sees
something new, that's problematic.

But first...

Play Time



Monomorphism.

Polymorphism.

Megamorphism.

This is not just for objects.

$*$ -morphism.

- **Monomorphic:** This is all I know and all that I've seen. I can get incredibly fast at this one thing.
- **Polymorphic:** I've seen a few shapes before. Let me just check to see which one and then I'll go do the fast thing.
- **Megamorphic:** I've seen things. A lot of things. I'm not particularly specialized. Sorry.

**So, how does the browser figure
out what type something is?**

Play Time

Dynamic lookup: This object could be anything, so let me look at the rule book and figure this out.

Sure, computers are good at looking stuff up repeatedly, but they're also good at remembering things.

It turns out there is a secret
type system behind your back.

const obj = { x: 1 };

cφ

const obj = { x: 1 };

obj.y = 2;

cφ
c!

const obj = { x: 1 };

obj.y = 2;

cp
ci

const another = { x: 1, y: 2 };

const obj = { x: 1 };

c1

obj.y = 2;

c2

const another = { x: 1, y: 2 };

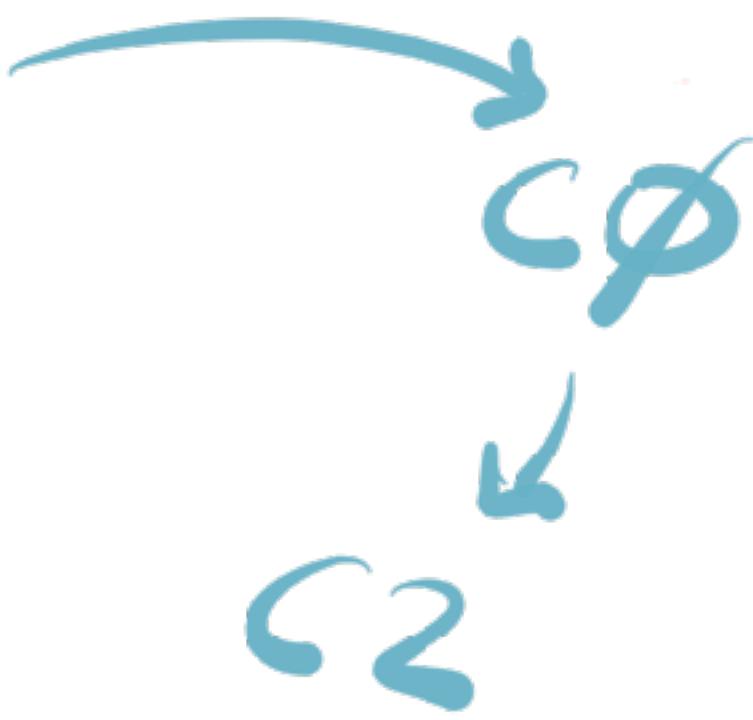
You can only move forward
along the chain.

const obj = { x: 1 };

 cφ

```
const obj = { x: 1 };
```

```
obj.y = 2;
```

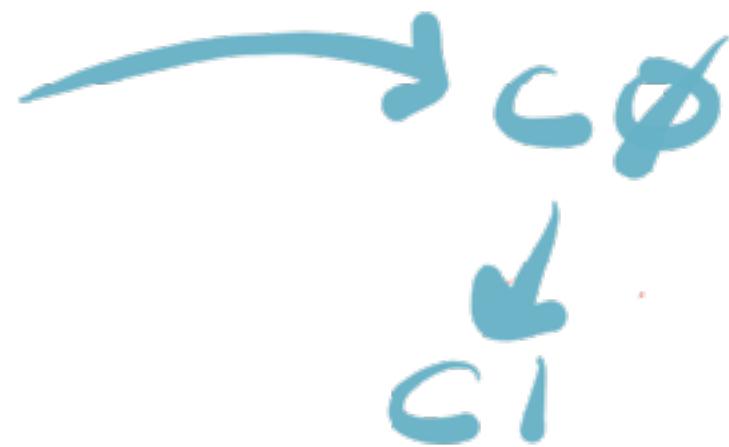


const obj = { x: 1 };  c0

obj.y = 2;  c1

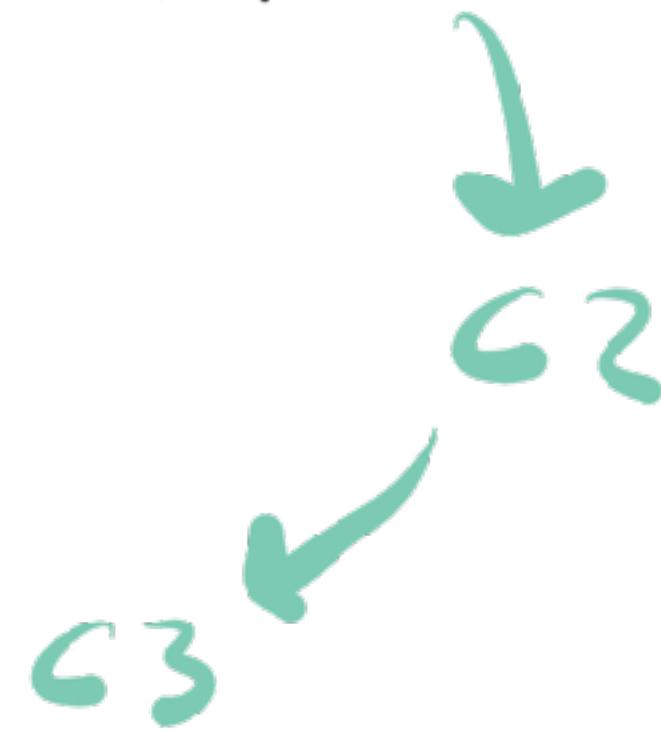
const another = { x: 1, y: 2 };  c2

```
const obj = { x: 1 };
```



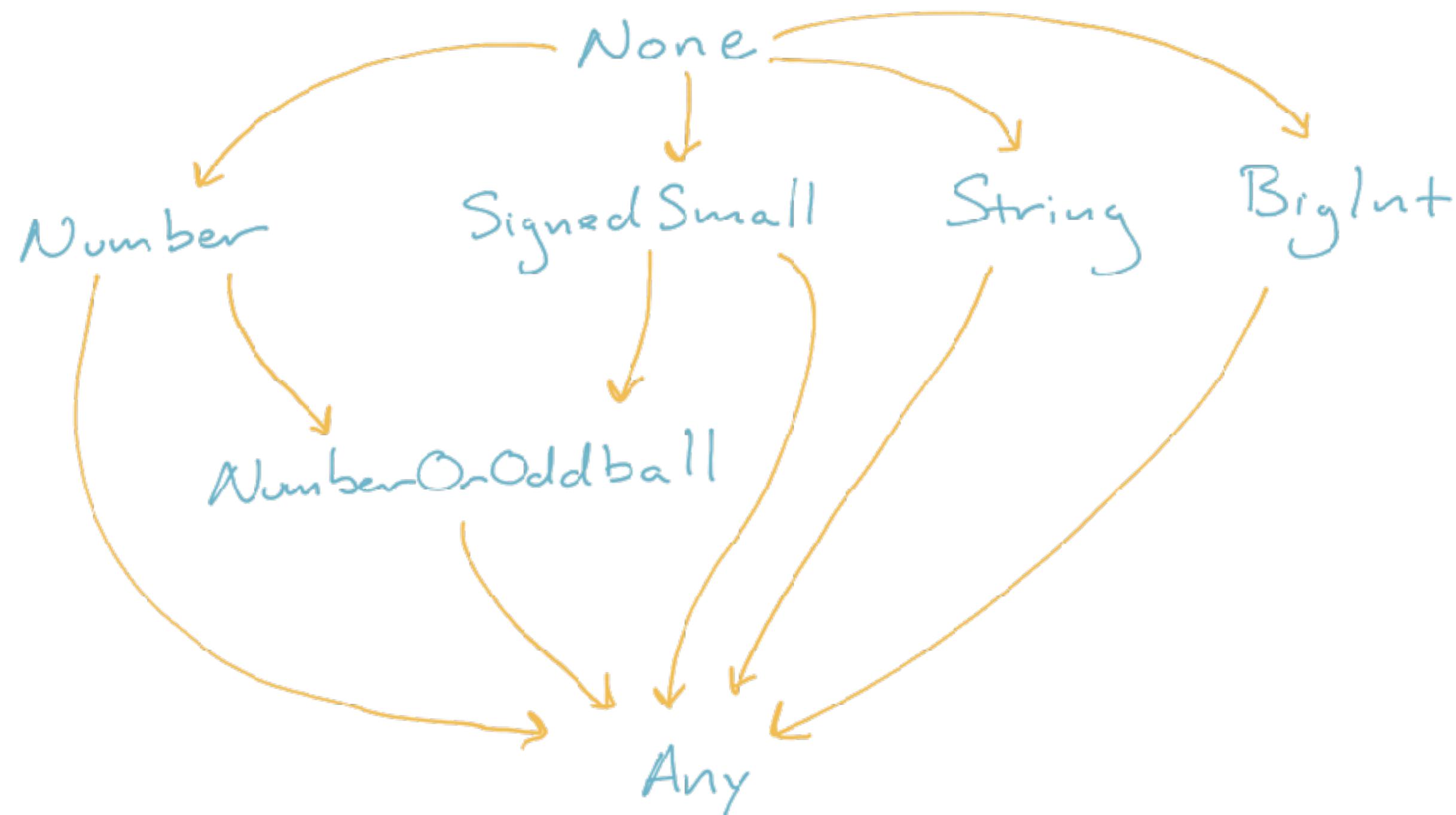
```
const another = { x: 1, y: 2 };
```

```
delete another.y;
```



Okay, let's look at this
again with some real code.

FEEDBACK LATTICE



Play Time

Takeaways

- Turbofan is able to optimize your code in substantial ways if you pass it consistent values.

Takeaways

- Initialize your properties at creation.
- Initialize them in the same order.
- Try not to modify them after the fact.
- Maybe just use TypeScript or Flow so you don't have to worry about these things?

Function Inlining

Play Time

Larger Takeaways

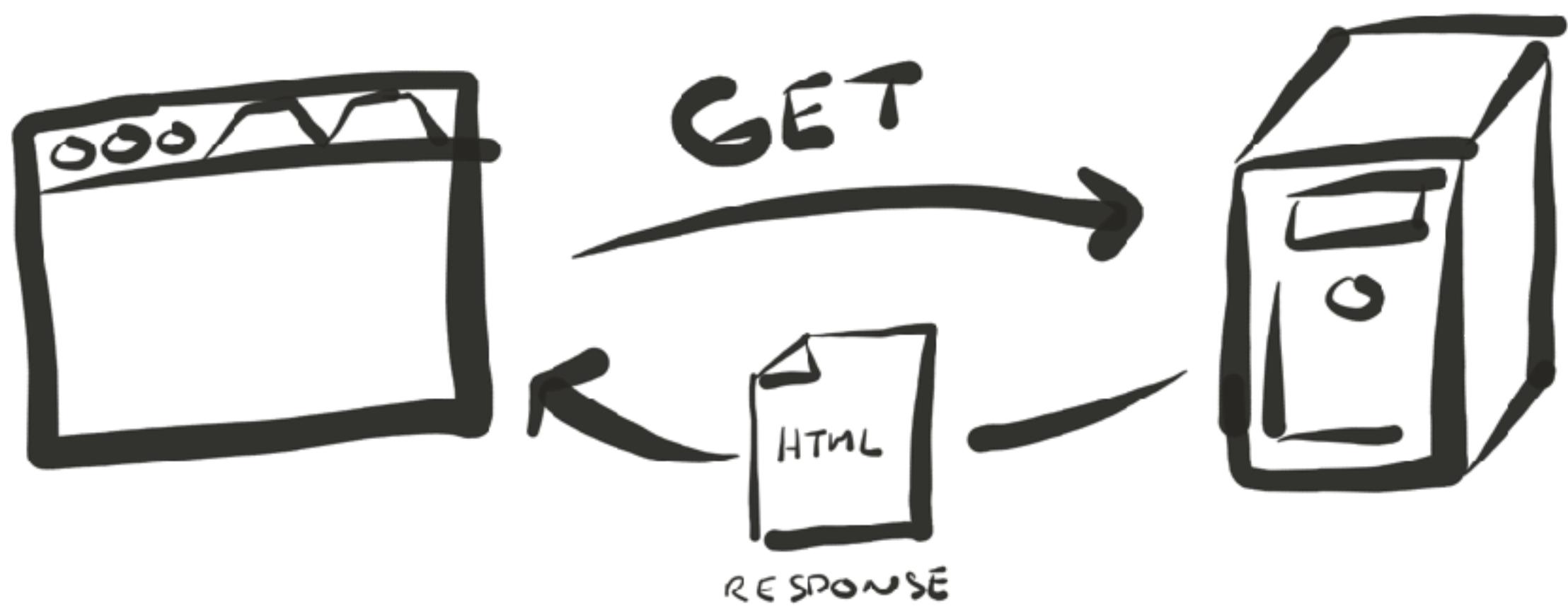
- The easiest way to reduce parse, compile, and execution times is to ship less code.
- Use the **User Timing API** to figure out where the biggest amount of hurt is.
- Consider using a type system so that you don't have to think about all of the stuff I just talked about.

Chapter Two

Rendering Quickly Now and Over Time.

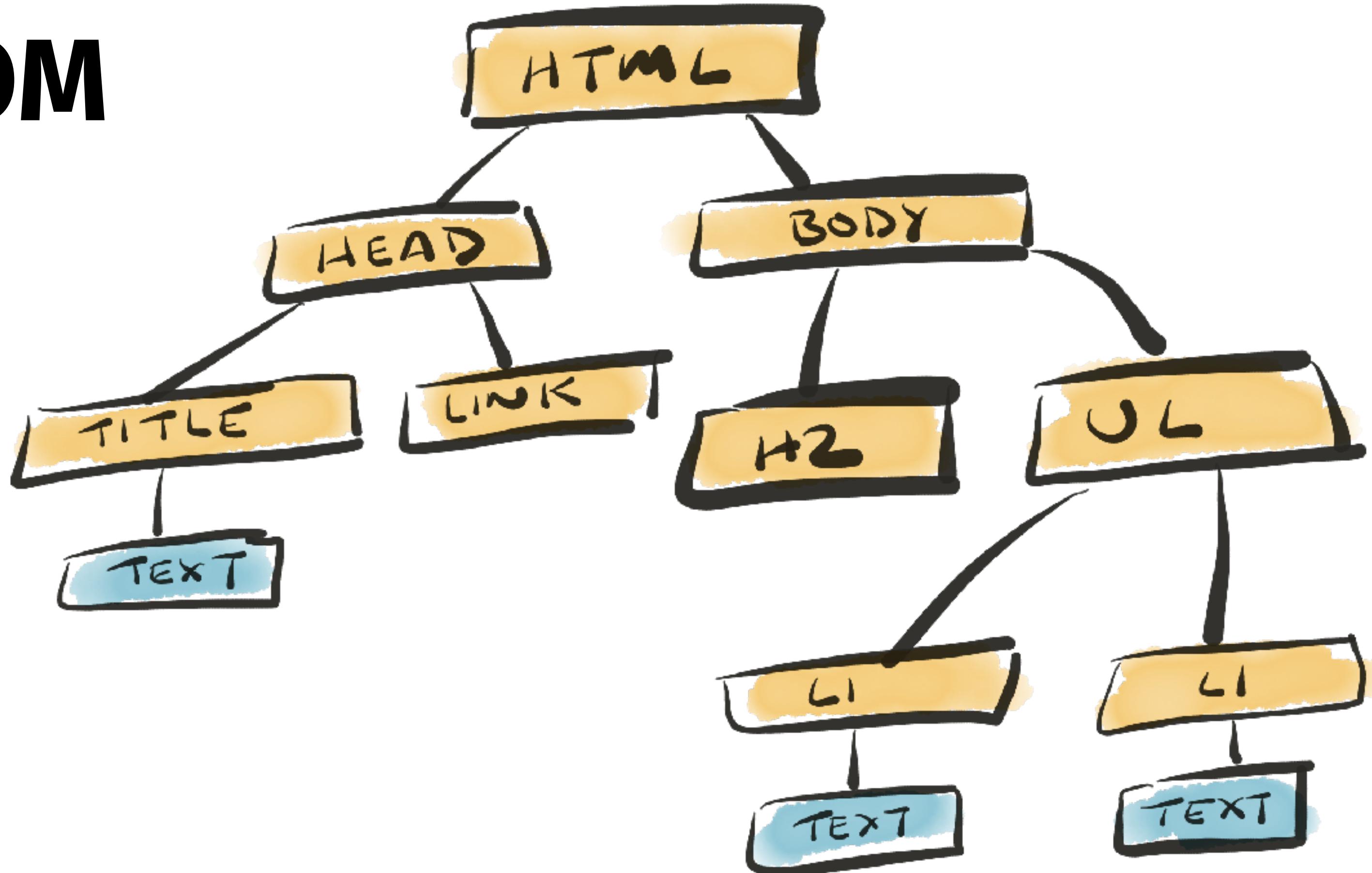
How Web Pages Are Born





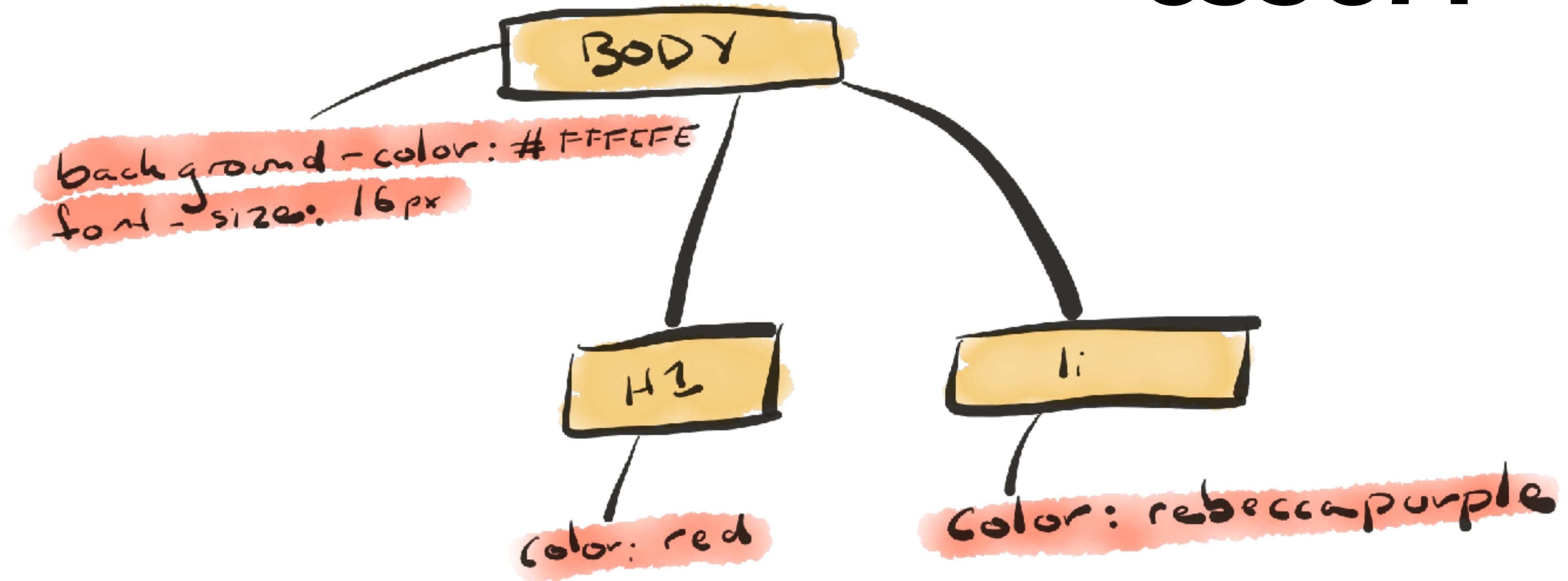
```
<html>
  <head>
    <title>Awesome Webpage</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Grocery List</h1>
    <ul>
      <li>Apples</li>
      <li>Vegan Spam</li>
    </ul>
  </body>
</html>
```

DOM

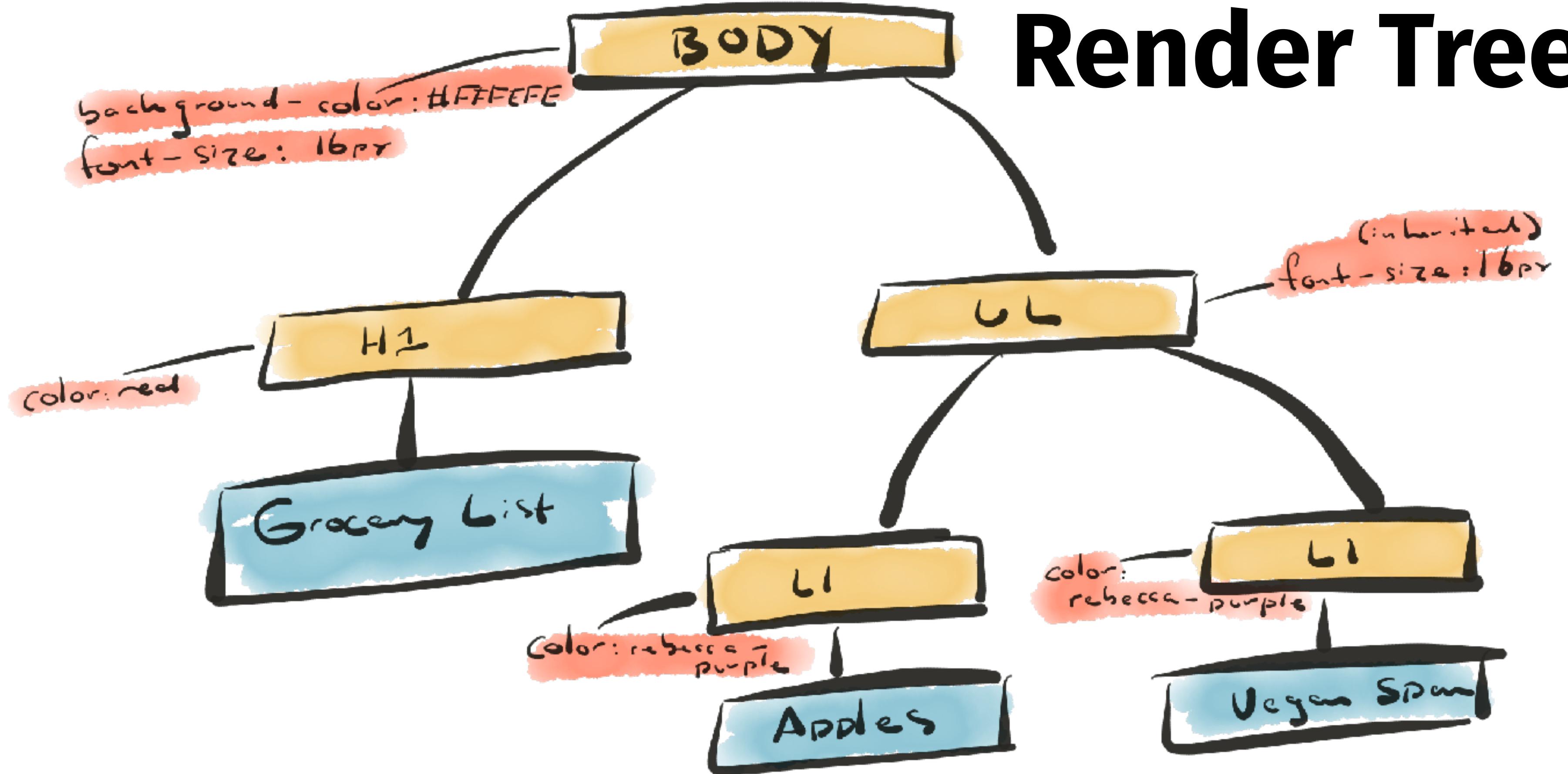


```
body {  
    background-color: #FFFFFF;  
    font-size: 16px;  
}  
  
h1 {  
    color: red;  
}  
  
li {  
    color: rebeccapurple;  
}
```

CSSOM



Render Tree



The Render Tree

- The Render Tree has a one-to-one mapping with the *visible* objects on the page.
 - So, not hidden object.
 - Yes, to pseudo elements (e.g. :after, :before).
- There might be multiple rules that apply to a single element. We need to figure that all out here.

Style calculation: The browser figures out all of the styles that will be applied to a given element.

This involves two things:

- Figuring out which rules apply to which elements.
- Figuring out how what the end result of an element with multiple rules is.

Styling Elements: Selector Matching

This is the process of figuring out what styles apply to an element.

The more complicated you
get, the longer this takes.

Class names are super
simple.



.sidebar > .menu-item:nth-child(4n + 1)

Free Advice: Stick to simple class names whenever possible. Consider using BEM.

Browsers read selectors
from right to left.

The less selectors you use,
the faster this is going to be.

Takeaways

- Use simple selectors whenever possible.
 - Consider using BEM or some other system.
- Reduce the effected elements.
 - This is really a way to get to the first one.
 - A little bit of code—either on the server or the client—can go a long way.

Styling Elements: Calculating Render Styles

Selector matching tries to figure out what selectors apply to an element.

When multiple selectors apply to an element. The browser needs to figure out who wins.

The easiest way to make
this faster is to not do it.

Free Advice (again): Stick to simple
class names whenever possible.
Consider using BEM.

**A quick note on style invalidation: It
doesn't matter as much in newer
browsers.**

Some Takeaways

- Reduce the amount of unused CSS that you're shipping.
 - The less styles you have, the less there is to check.
- Reduce the number of styles that effect a given element.

Layout (a.k.a Reflow): Look at the elements and figure out where they go on the page.

Paint: We know what things should look like and where they should go.
Draw some pixels to the screen.

Composite Layers: You might end up painting on multiple layers, but you'll eventually need to combine them.

Profit.



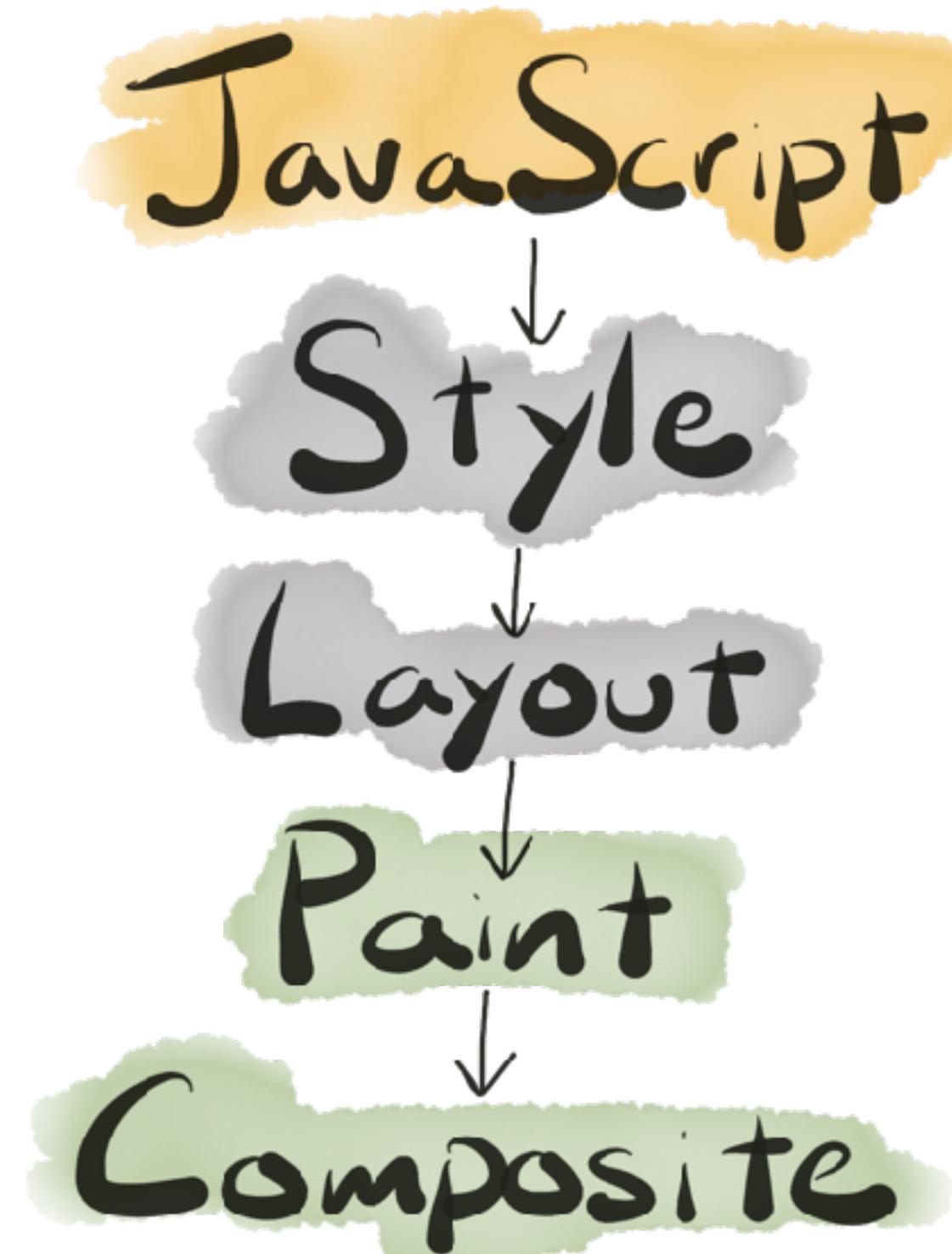
JavaScript gives you the ability to change all of this after the initial load, which means you might have to do all of the above *again*.



Things JavaScript can do: An incomplete list™

- Change the class on an object.
- Change the inline styles on an object.
- Add or remove elements from the page.

The Render Pipeline



**Okay, so let's say you change a
class or inline style on an element.**

The computed styles could have changed—so, we better recalculate those and rebuild the render tree.

That may or may not have changed
the geometry of the objects. We
should probably re-layout the page.

Things are different. I guess we
need to paint some new images.

Send those images off to
the GPU to be composited.

To be clear: You don't need to do
all of these things every time.

And, that's what this is
about.

**Reminder: Steve's golden
rule of performance.**

Layouts and Reflows

“
Reflows are very expensive in terms of performance, and is one of the main causes of slow DOM scripts, especially on devices with low processing power, such as phones. In many cases, they are equivalent to laying out the entire page again.

—Opera



Whenever the geometry of an element changes, the browser has to reflow the page.

(Browser implementations have different ways of optimizing this, so there is no point sweating the details in this case.)

Tasting Notes

- A reflow is a blocking operation. Everything else stops.
- It consumes a decent amount of CPU.
- It will definitely be noticeable by the user if it happens often (e.g. in a loop).

A reflow of an element causes a reflow of its parents and children.

Okay, so what causes a reflow?

- Resizing the window
- Changing the font
- Content changes
- Adding or removing a stylesheet
- Adding or removing classes
- Adding or removing elements
- Changing orientation
- Calculating size or position
- Changing size or position
- (Even more...)

Generally speaking, a reflow is followed by a repaint, which is also expensive.

How can you avoid reflows?

- Change classes at the lowest levels of the DOM tree.
- Avoid repeatedly modifying inline styles.
- Trade smoothness for speed if you're doing an animation in JavaScript.
- Avoid table layouts.
- Batch DOM manipulation.
- Debounce window resize events.

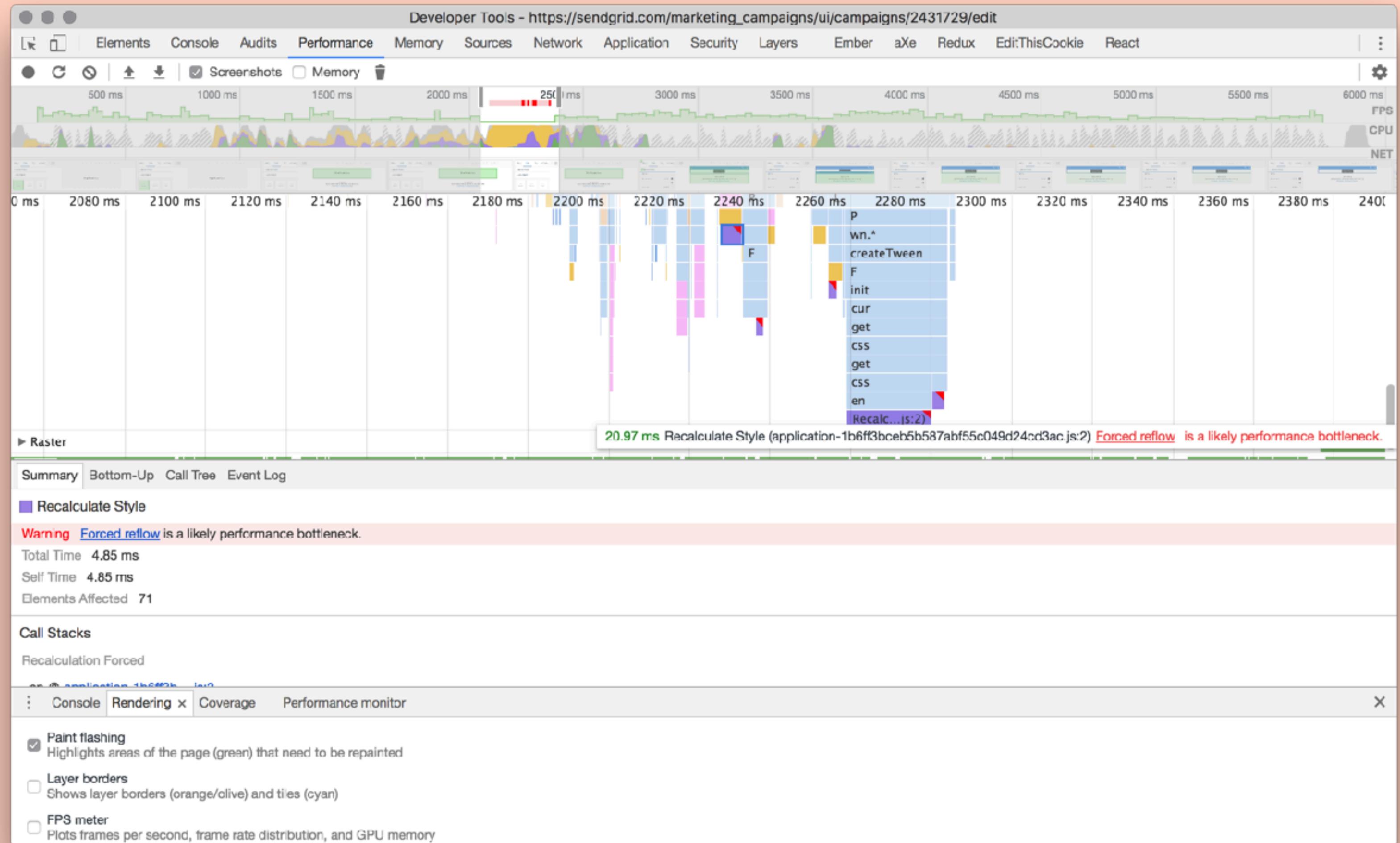
Lab

[https://codepen.io/
stevekinney/full/eVadLB/](https://codepen.io/stevekinney/full/eVadLB/)

Layout Thrashing

**Less cool names: Forced
synchronous layout.**

There are a set of things you can do
that cause the browser to stop what it's
doing and calculate style and layout.



“
Layout Thrashing occurs when JavaScript violently writes, then reads, from the DOM, multiple times causing document reflows.

—Winston Page

”

```
const height = element.offsetHeight;
```

```
const height = element.offsetHeight;
```

The browser wants to get you the most up to date answer, so it goes and does a style and layout check.

```
firstElement.classList.toggle('bigger');
const firstElementWidth = firstElement.width;
secondElement.classList.toggle('bigger');
const secondElementWidth = secondElement.width;
```

```
firstElement.classList.toggle('bigger'); // Change!
const firstElementWidth = firstElement.width; // Calculate
secondElement.classList.toggle('bigger'); // Change!
const secondElementWidth = secondElement.width; // Calculate
```

The browser knew it was going to have to change stuff after that first line.

Then you went ahead and asked it
for some information about the
geometry of another object.

So, it stopped your JavaScript and
reflowed the page in order to get
you an answer.

Solution: Separate reading
from writing.

```
firstElement.classList.toggle('bigger'); // Change!
secondElement.classList.toggle('bigger'); // Change!
const firstElementWidth = firstElement.width; // Calculate
const secondElementWidth = secondElement.width; // 🌴
```

Play Time

It sounds like we could use
a better abstraction, right?

GitHub - wilsonpage/fastdom: Eliminates layout thrashing by batching DOM measurement and mutation tasks

<https://github.com/wilsonpage/fastdom>

Preventing layout thrashing! | GitHub - wilsonpage/fastdom...

Features Business Explore Marketplace Pricing This repository Search Sign in or Sign up

wilsonpage / fastdom

Code Issues 7 Pull requests 3 Projects 0 Insights

Watch 118 Star 4,328 Fork 186

Eliminates layout thrashing by batching DOM measurement and mutation tasks

266 commits 12 branches 27 releases 24 contributors

Branch: master New pull request Find file Clone or download

wilsonpage 1.0.8 Latest commit 78d59bb 6 days ago

examples	V1	2 years ago
extensions	Fix promise not being deleted from Map after clear (fixes #102) (#105)	11 months ago
src	Fix fastdom-strict callbacks not being called with context (fixes #108)...	10 months ago
test	Fix fastdom-strict callbacks not being called with context (fixes #108)...	10 months ago
.gitignore	V1	2 years ago
.jshintrc	V1	2 years ago
.npmignore	V1	2 years ago
travis.yml	Fixed incorrect context in fastdom-promised (closes #81)	2 years ago

```
fastdom.measure(() => {
  console.log('measure');
});
```

```
fastdom.mutate(() => {
  console.log('mutate');
});
```

```
fastdom.measure(() => {
  console.log('measure');
});
```

```
fastdom.mutate(() => {
  console.log('mutate');
});
```

Play Time

Lab

[https://codepen.io/
stevekinney/full/eVadLB/](https://codepen.io/stevekinney/full/eVadLB/)

React to the rescue?

```
class App extends Component {
  state = { widths: [50, 100, 150], }

  doubleSize = () => {
    const widths = this.state.widths.map(n => n * 2);
    this.setState({ widths });
  };

  render() {
    const [firstWidth, secondWidth, thirdWidth] = this.state.widths;
    return (
      <div>
        <button onClick={this.doubleSize}>Double Sizes</button>
        <div style={{ width: firstWidth }}>
          <div style={{ width: secondWidth }}>
            <div style={{ width: thirdWidth }}>
              </div>
            </div>
          </div>
        </div>
      );
    }
}
```

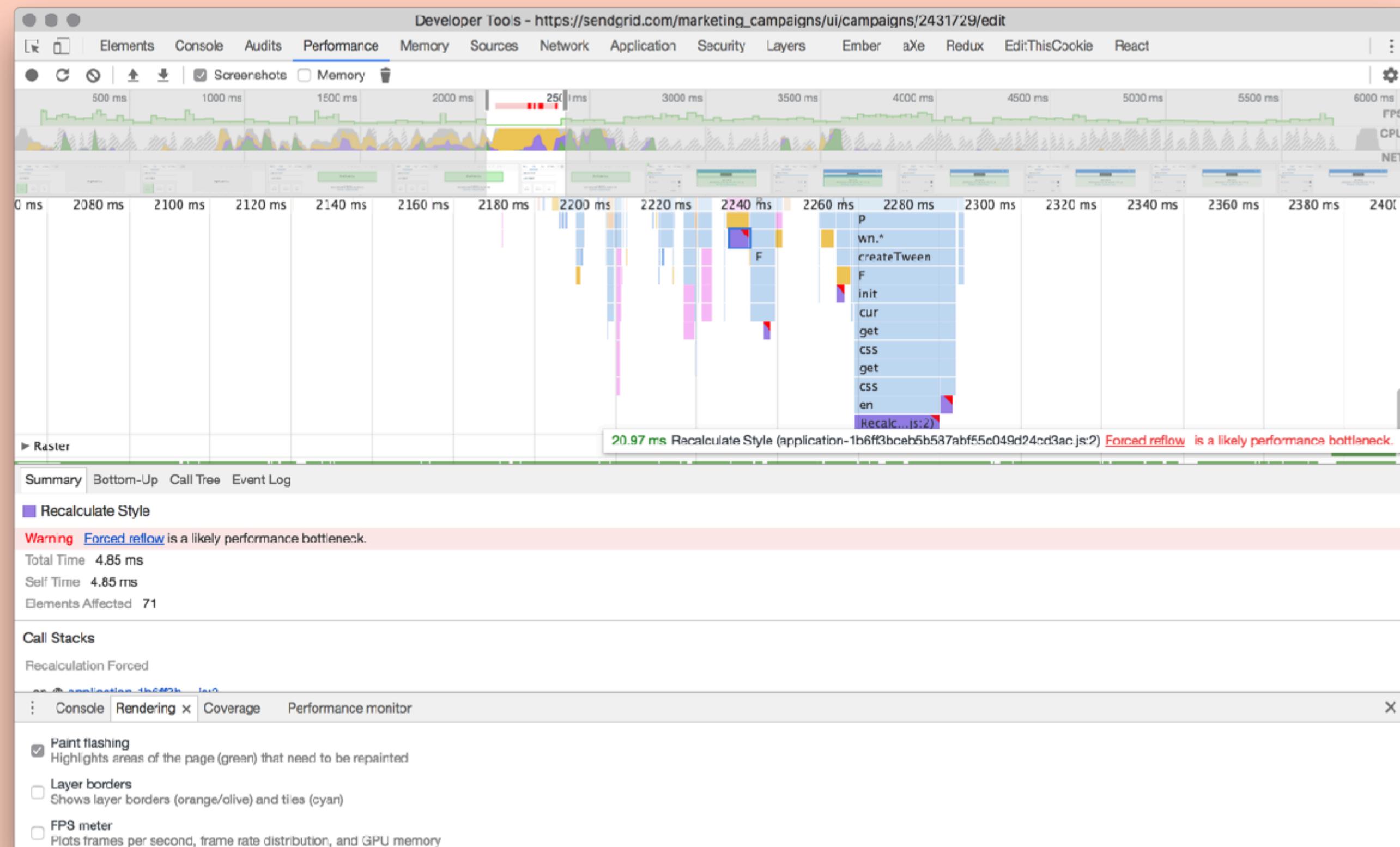
Lab

[https://
2z379l1pjr.codesandbox.io/](https://2z379l1pjr.codesandbox.io/)

Friendly fact: Production mode is important in React!

Play Time

Er, maybe not.



Some Takeaways

- Don't mix reading layout properties and writing them—you'll do unnecessary work.
- If you can change the visual appearance of an element by adding a CSS class. Do that, you'll avoid accidental trashing.

Some Takeaways

- Storing data in memory—as opposed to the DOM—means we don’t have to check the DOM.
- Frameworks come with a certain amount of overhead.
- You don’t *need* to use a framework to take advantage of this.
- You *can* do bad things even if you use a framework.
- You may not know you’re layout thrashing—so, measure!



Painting, Layers, the Profiling Thereof

Anytime you change something other
than opacity or a CSS transform...
you're going to trigger a paint.

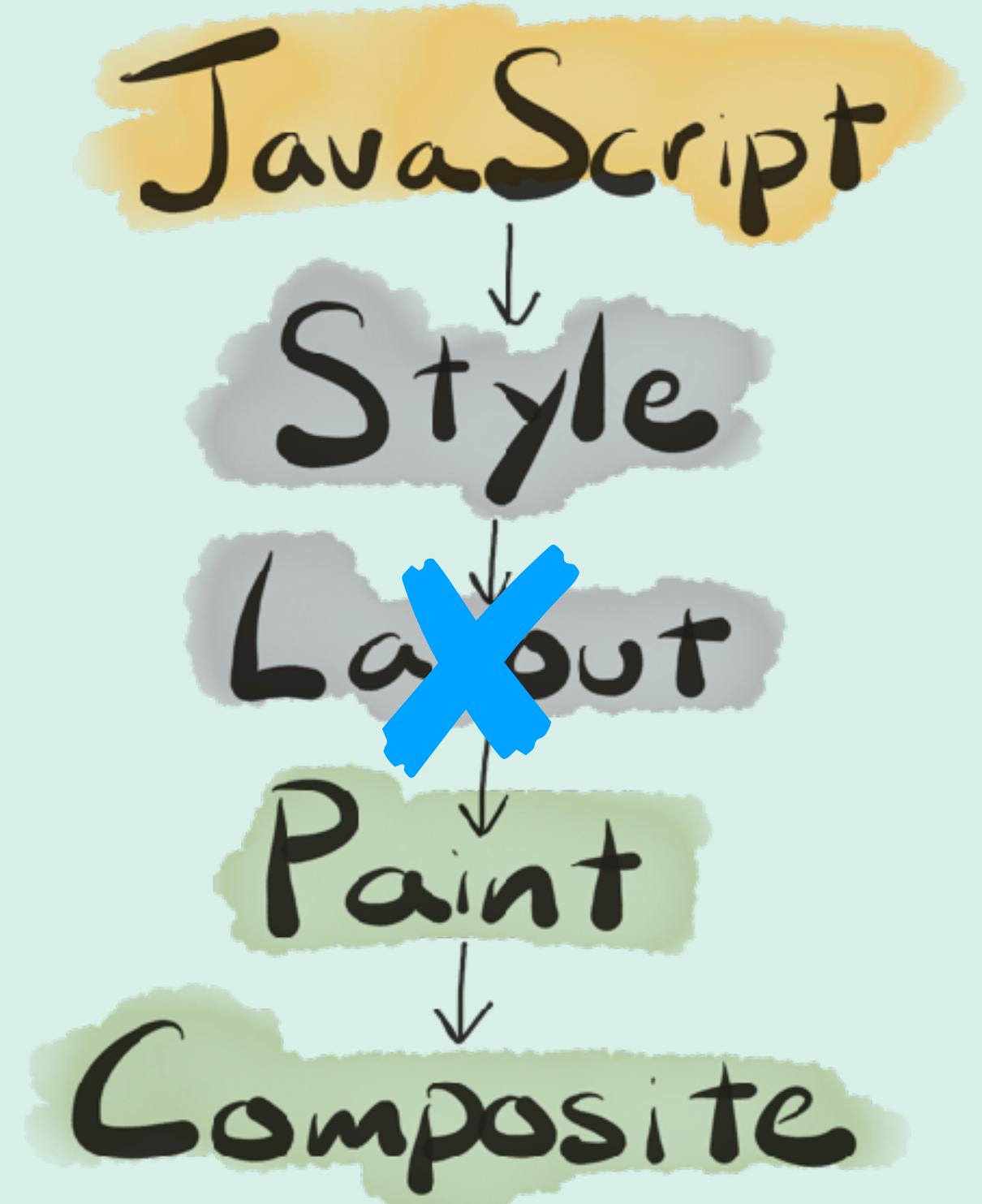


When we do a paint, the browser tells every element on the page to draw a picture of itself.

It has all of this information form
when we constructed the render
tree and did the layout.

Triggering a layout will
always trigger a paint.

But, if you're just changing colors or something—then you don't need to do a reflow. Just a repaint.



Use your tools to see if
you're painting.

Play Time

Rule of Thumb: Paint as much as you need and as little as you can get away with.

An Aside: The Compositor Thread

Nice threads

- **The UI thread:** Chrome itself. The tab bar, etc.
- **The Renderer thread:** We usually call this the main thread.
 - This is where all JavaScript, parsing HTML and CSS, style calculation, layout, and painting happens. There are one of these per tab.
- **The Compositor Thread:** Draws bitmaps to the screen via the GPU.

The Compositor Thread

- When we paint, we create bitmaps for the elements, put them onto layers, and prepare shaders for animations if necessary.
- After painting, the bitmaps are shared with a thread on the GPU to do the actual compositing.
- The GPU process works with OpenGL to make magic happen on your screen.

**The Main Thread is CPU-
intensive.**

**The Compositor Thread is
GPU-intensive.**

It can go off and work on some super hard JavaScript computation and the animations will still chug along.

This is cool, because it frees up the main thread to do all of the work it's responsible for.

Managing Layers

Again: Painting is super expensive and you should avoid it whenever possible.

“
But, Steve—how do I avoid painting? Isn’t that just a fact of life when it comes to getting pixels on the screen? –**Your inner monologue**

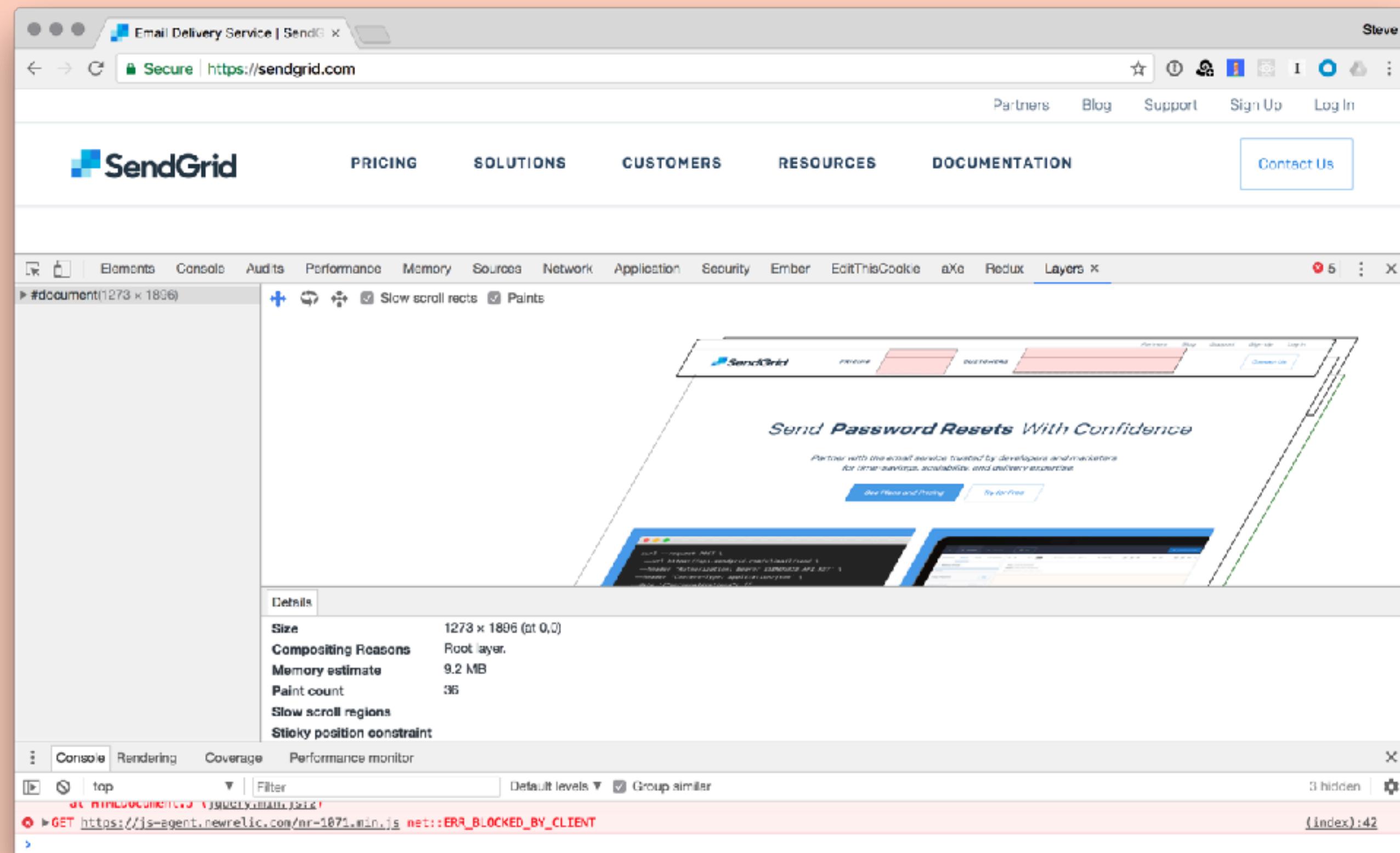
”

“*Let the Compositor Thread handle
this stuff!*” – *Me, in response*

Things the compositor thread is really good at:

- Drawing the same bitmaps over and over in different places.
- Scaling and rotating bitmaps.
- Making bitmaps transparent.
- Applying filters.
- Mining Bitcoin.

If you want to be fast, then offload
whatever you can to the less-busy
thread.



**Disclaimer: Compositing is
kind of a hack.**



CODE

Save Draft

Send Campaign



HTML



T

```
1 <html>
2 <head>
3   <title></title>
4 </head>
5 <body></body>
6 </html>
7
8
```

From:

Subject:

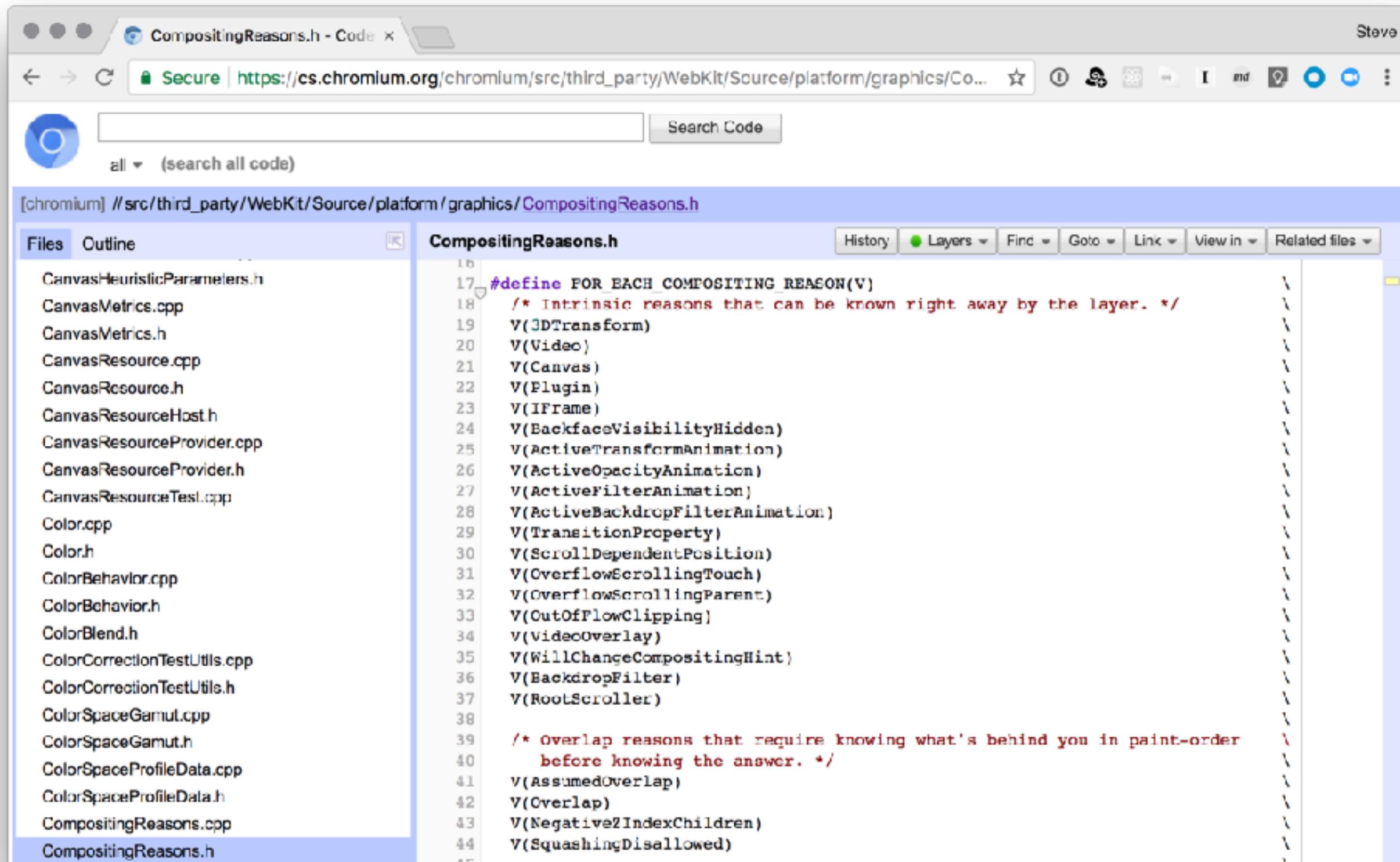
ONE DOES NOT SIMPLY

USE LAYERS

Layers are an optimization that the browser does for you under the hood.

What kind of stuff gets its own layer?

- The root object of the page.
- Objects that have specific CSS positions.
- Objects with CSS transforms.
- Objects that have overflow.
- (Other stuff...)



The screenshot shows a web-based code viewer for the Chromium project. The URL is https://cs.chromium.org/chromium/src/third_party/WebKit/Source/platform/graphics/CompositingReasons.h. The code editor displays the file `CompositingReasons.h`. The left sidebar lists other files in the directory, such as `CanvasHeuristicParameters.h`, `CanvasMetrics.cpp`, and `Color.cpp`. The main pane shows the content of `CompositingReasons.h`, which defines various reasons for compositing. The code is annotated with comments explaining the reasoning behind each reason.

```
#define FOR_EACH_COMPOSITING_REASON(V)  
/* Intrinsic reasons that can be known right away by the layer. */  
V(3DTransform)  
V(Video)  
V(Canvas)  
V(Plugin)  
V(IFrame)  
V(BackfaceVisibilityHidden)  
V(ActiveTransformAnimation)  
V(ActiveOpacityAnimation)  
V(ActiveFilterAnimation)  
V(ActiveBackdropFilterAnimation)  
V(TransitionProperty)  
V(ScrollDependentPosition)  
V(OverflowScrollingTouch)  
V(OverflowScrollingParent)  
V(CutOfFlowClipping)  
V(VideoOverlay)  
V(WillChangeCompositingHint)  
V(BackdropFilter)  
V(RootScroller)  
  
/* Overlap reasons that require knowing what's behind you in paint-order  
   before knowing the answer. */  
V(AssumedOverlap)  
V(Overlap)  
V(Negative2IndexChildren)  
V(SquashingDisallowed)
```

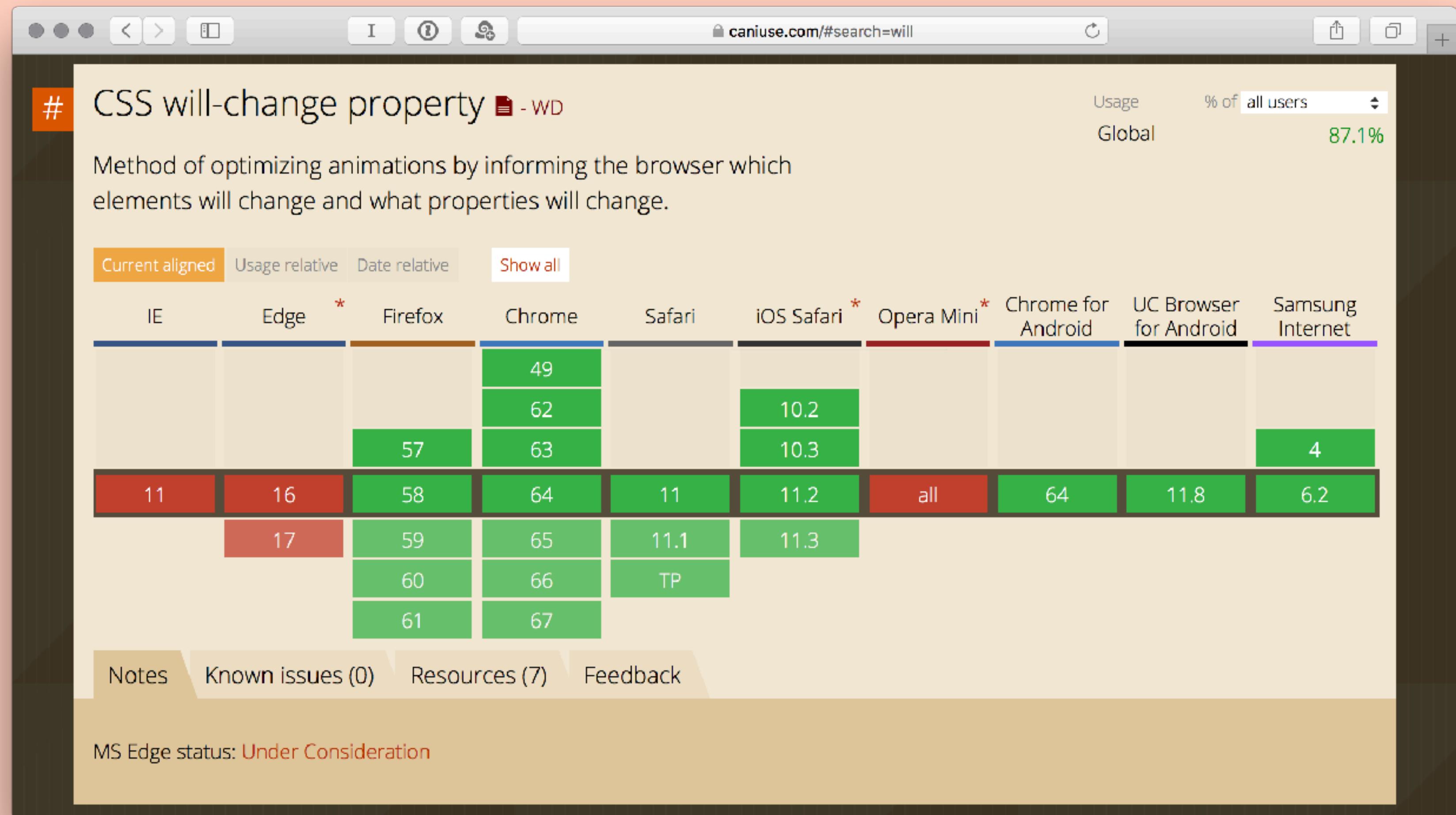
<http://bit.ly/paint-layers>

Objects that don't fall under one of these reasons will be on the same element as the last one that did.

(Hint: The root object is always its own layer.)

**You can give the browser hints
using the will-change property.**

```
.sidebar {  
  will-change: transform;  
}
```



```
.sidebar {  
    transform: translateZ(0);  
}
```

```
.sidebar {  
  will-change: transform;  
}
```

Play Time

Developer Tools - https://linen-copper.glitch.me/

Elements Console Audits Performance Memory Sources Network Application Security Layers X

#document(800 x 1206) Slow scroll rects Paints

Mr. Awesome's Message
Hello World!

Details

Size	800 x 1206 (at 0,0)
Compositing Reasons	Root layer.
Memory estimate	3.7 MB
Slow scroll regions	
Sticky position constraint	

Console Rendering Coverage Performance monitor X

Paint flashing
Highlights areas of the page (green) that need to be repainted

Layer borders
Shows layer borders (orange/olive) and tiles (cyan)

FPS meter
Plots frames per second, frame rate distribution, and GPU memory

Developer Tools - https://linen-copper.glitch.me/

Elements Console Audits Performance Memory Sources Network Application Security Layers X

#document(800 x 1206) Slow scroll rects Paints

Mr. Anderson's Message

Slow Scroll

Slow Scroll Regions

Details

Size	800 x 1206 (at 0,0)
Compositing Reasons	Root layer.
Memory estimate	3.7 MB
Paint count	635
Slow scroll regions	
Sticky position constraint	

Console Rendering Coverage Performance monitor

Paint flashing
Highlights areas of the page (green) that need to be repainted

Layer borders
Shows layer borders (orange/olive) and tiles (cyan)

FPS meter
Plots frames per second, frame rate distribution, and GPU memory



Using layers is a trade
off.

Managing layers takes a certain amount of work on the browser's behalf.

Each layer needs to be kept in the shared memory between the main and composite threads.

This is a terrible idea.

```
* {  
  will-change: transform;  
}
```

The browser is already trying to
help you out under the hood.

Pro Tip: will-change is for things
that will change. (Not things that are
changing.)

Promoting an object to its own layer
takes a non-zero amount of time.

```
.sidebar.is-opening {  
  will-change: transform;  
  transition: transform 0.5s;  
  transform: translate(400px);  
}
```

```
.sidebar {  
  will-change: transform;  
  transition: transform 0.5s;  
}  
  
.sidebar:hover {  
  will-change: transform;  
}  
  
.sidebar.open {  
  transform: translate(400px);  
}
```

will-change is tricky because
while it's a CSS property, you'll
typically access it using JavaScript.

```
element.addEventListener('mouseenter', () => {  
  element.style.willChange = 'transform';  
});
```

If it's something that the user is interacting with constantly, add it to the CSS. Otherwise, do it with JavaScript.

Clean up after yourself. Remove will-change when it's not going to change anymore.

```
element.addEventListener('mouseenter', () => {  
  element.style.willChange = 'transform';  
});
```

```
element.addEventListener('animationEnd', () => {  
  element.style.willChange = 'auto';  
});
```

Exercise

- Let's look at the “Paint Storming” example.
- Don't be surprised if you find a paint storm.
- Can you swap out that jQuery animation for a CSS transition?
- Can you put the `will-change` on before the transition?
- Can you remove it after?

Chapter Three

Getting What You Need

Latency and Bandwidth: A Journey of Self-Discovery

“
Networks, CPUs, and disks all hate you. On the client, you pay for what you send in ways you can't easily see. —**Alex Russell**

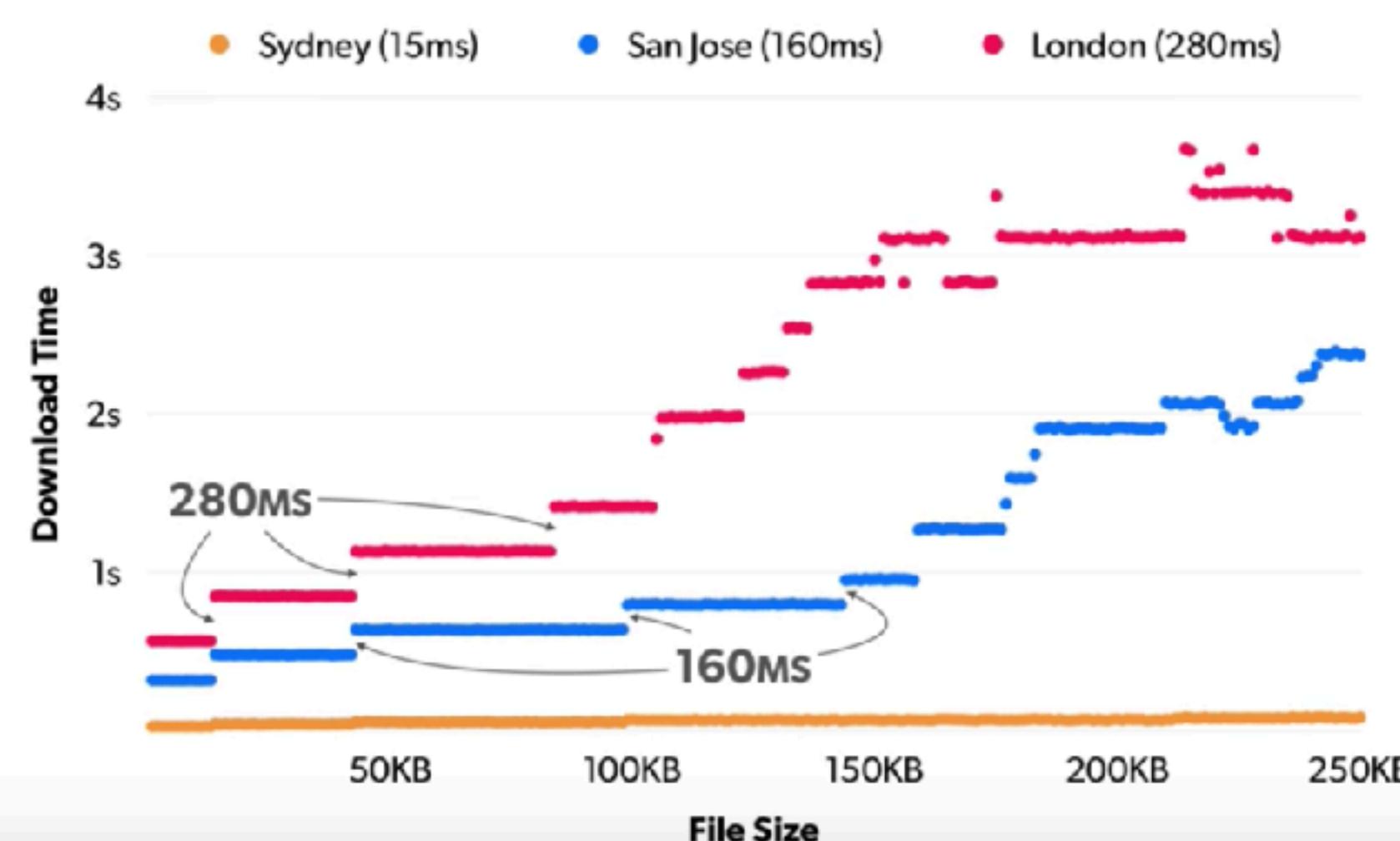
”



Bandwidth vs. Latency

- **Bandwidth** is how much stuff you can fit through the tube per second.
- **Latency** is how long it takes to get to the other end of the tube.

LATENCY vs LOAD TIME



▶ ▶! 🔊 4:24 / 12:14

CC HD

<https://www.youtube.com/watch?v=ak4EZQB4Ylg>



TCP focuses on reliability

- We keep checking in with the server to make sure that everything is going well.
- Packets are delivered in the correct order.
- Packets are delivered without errors.
- Client acknowledges each packet.
- Unreliable connections are handled well.
- Will not overload the network.

TCP *starts* by sending a small amount of data and then starts sending more and more as we find out that things are being successful.

Fun fact: This is why things feel so much worse on a slow Internet connection.

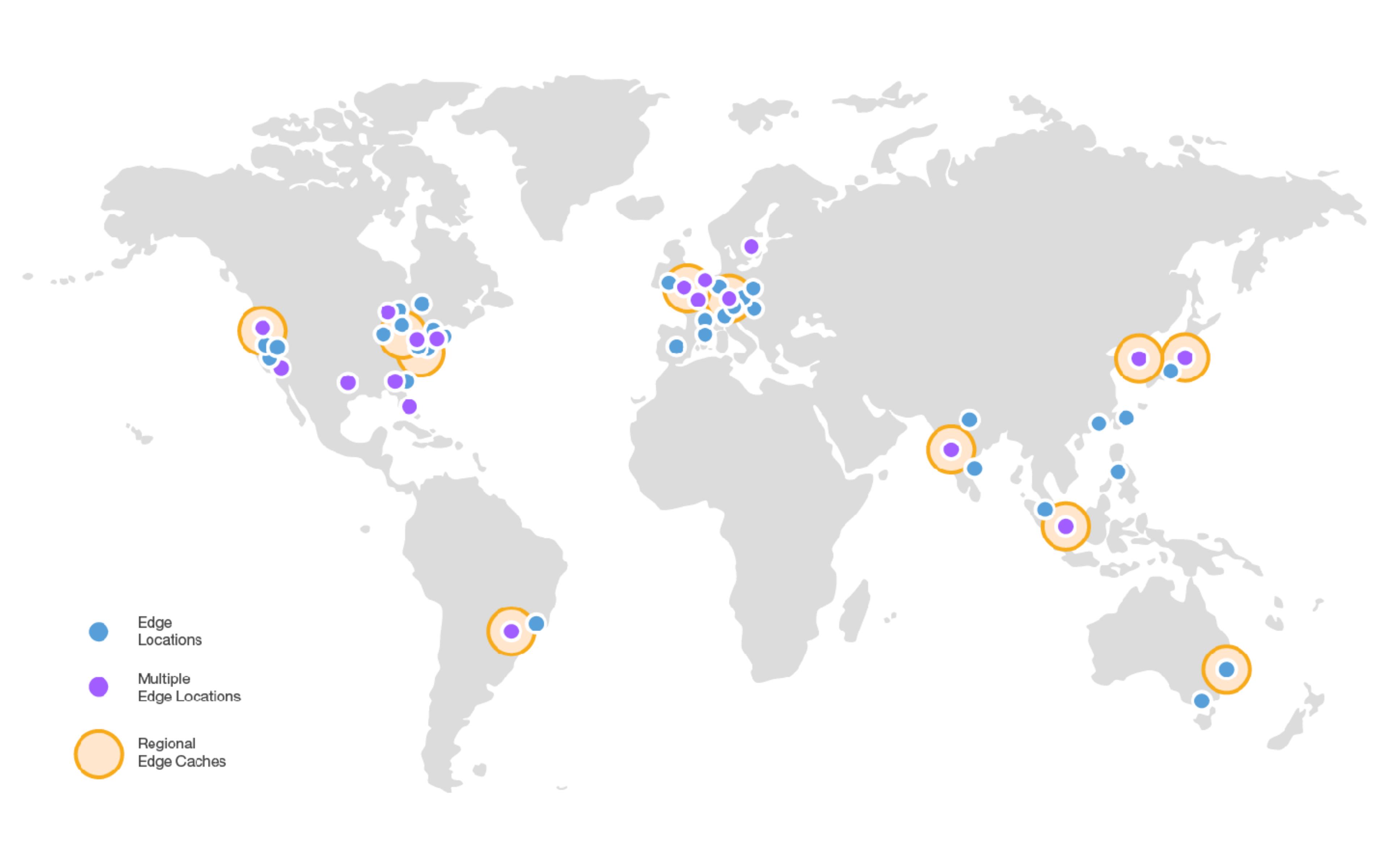
Pro tip: The initial window size is 14kb. So, if you can get files under 14kb, then it means you can get everything through in the first window. Very cool.

Lab

<http://www.cloudping.info/>

Hmm... So, where is the *optimal*
place to put our assets?

Answer: Everywhere.



Cache Money

The year is 1997.

PARENTAL
ADVISORY
EXPLICIT CONTENT



HTTP/1.1 added the Cache-Control response header.

Caching only affects the "safe" HTTP methods.

- GET
- OPTIONS
- HEAD

It doesn't support ... because how would it?

- PUT
- POST
- DELETE
- PATCH

Cache-Control headers

- no-store
- no-cache
- max-age
- s-maxage
- immutable

Three over-simplified possibilities

- **Cache Missing:** There is no local copy in the cache.
- **Stale:** Do a Conditional GET. The browser has a copy but it's old and no longer valid. Go get a new version.
- **Valid:** We have a thing in cache and its good—so, don't even bother talking to the server.

```
const express = require('express');
const serveStatic = require('serve-static');

const app = express();

app.use(serveStatic(__dirname, {
  setHeaders(response, path) {
    response.setHeader('Cache-Control', 'no-store');
  }
}));

const port = process.env.port || 3000;
app.listen(port, () => console.log(`⚓ Ahoy! The server is listening on port ${port}!`));
```

no-store: The browser gets
a new version every time.

no-cache: This means you can store a copy, but you can't use it without checking with the server.

max-age: Tell the browser not to bother if whatever asset it has is less than a certain number of seconds old.

Caching is great *unless* you
mess it up.

We can say "Yo, cache this for a long time!"

But, what if we ship some bunk assets? Oh no.

How will the user know to do a hard refresh to get the new ones?

**Another solution: Content-
Addressable Storage**

main.567eea7aa72b3ee48649.js

Caching for CDNs

CDNs respect the max-age header just like browsers. But this opens up a new can of worms.

- We want CSS and JavaScripts to be cached by the browser.
- We would like the CDN to cache the HTML that it serves up. But we don't want the browser to (because that ends us up in our earlier problem).

s-maxage is for CDNs only. Tell the CDN to keep it forever. But don't tell the browser to do it.

To reiterate: We have no way to reach into all of our customers browsers and tell them to purge their caches of our assets, but we can tell the CDN to.

Lazy-loading and pre-loading with React and webpack

**And now: A review of Steve's
golden rules for performance.**

**Not doing stuff is faster
than doing stuff.**

Doing stuff later is a way to
not do stuff now. So, it's faster.



Sean Thomas Larkin
@TheLarkInn

Following

My [#webpack](#) challenge for the world. Do not ever load (lazily or not) a bundle more than 300kb. Let me know what the perf result is. 😎

Jason Miller 🐈⚙️ @_developit

I cannot stress enough how important this is:

👉 add split points to your Webpack builds!

It's not okay to ship megabytes of JS on mobile 😭

5:17 PM - 7 Nov 2016

35 Retweets 116 Likes



10

35

116



I

Play Time

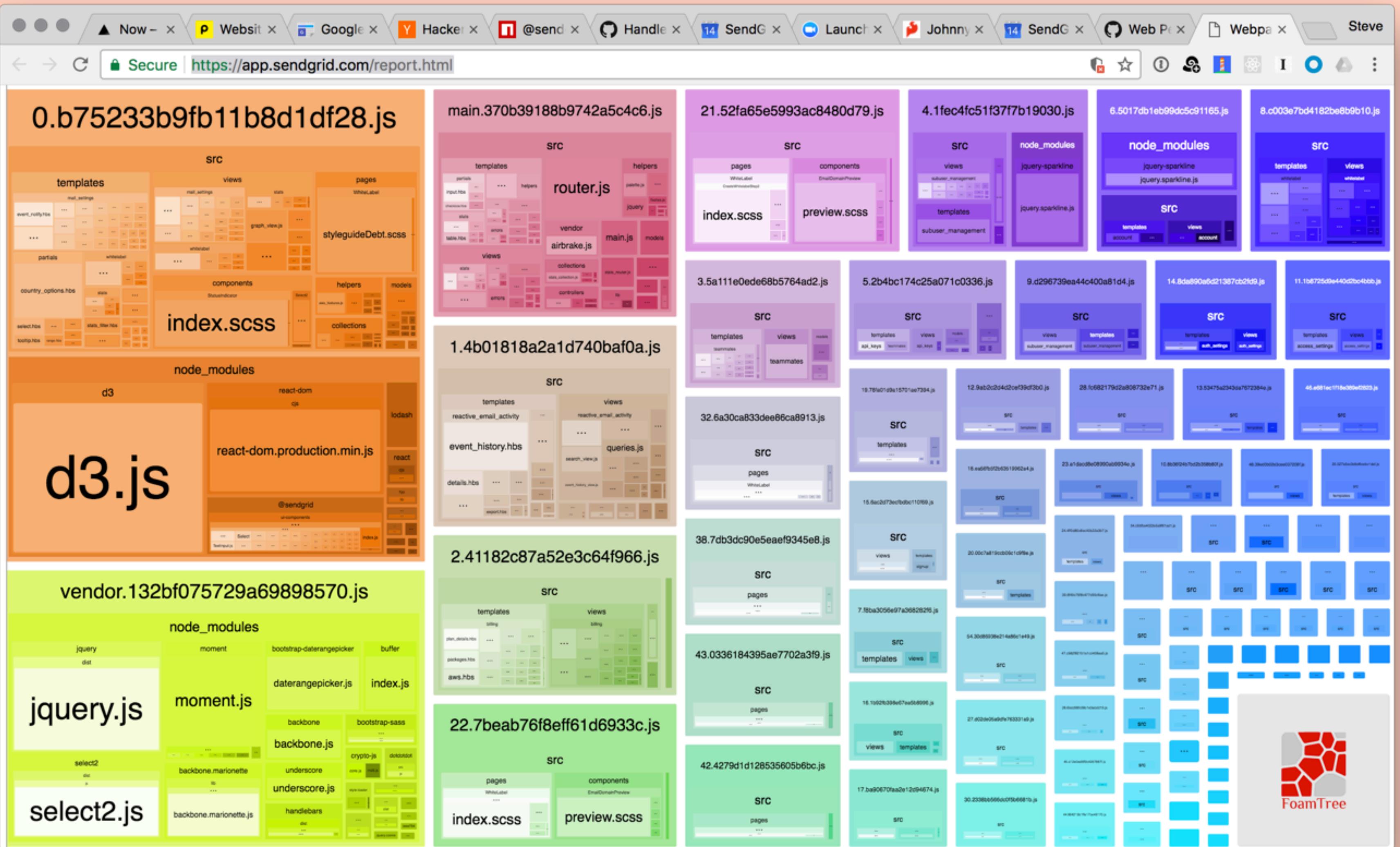
Exercise

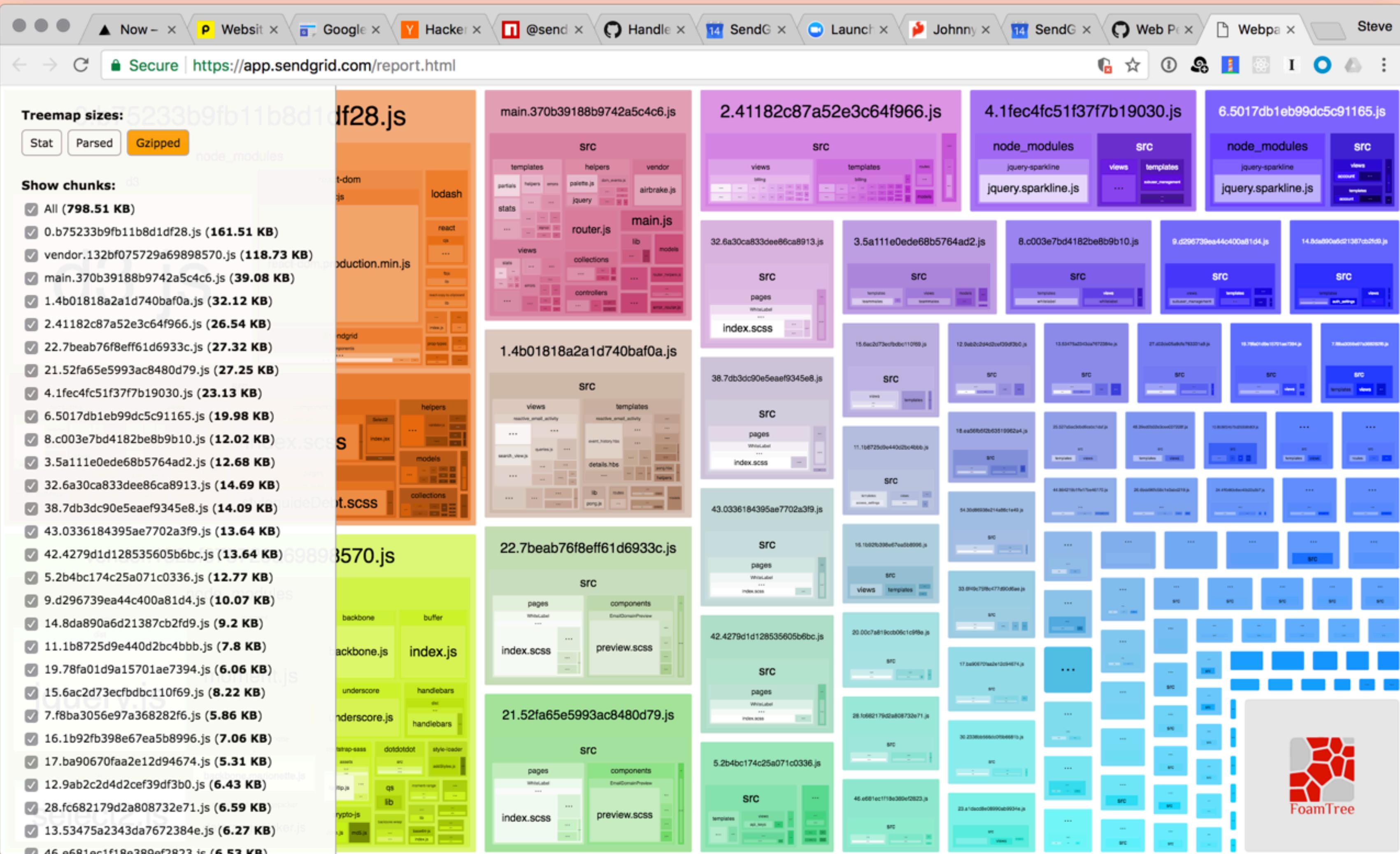
- So, I implemented lazy-loading for the Codemirror editor.
- But, it looks like the Markdown component is taking up quite a bit of space as well.
- Could you lazy-load that too for me?

Takeaways

- Some libraries have code you don't need. See if you can get that out of your build.
- Get the code you need now now.
- Get the code you need later later.
- Your tools can help you do this.

Please allow me to hop on
my soap box for a moment.





Some Words on HTTP/2

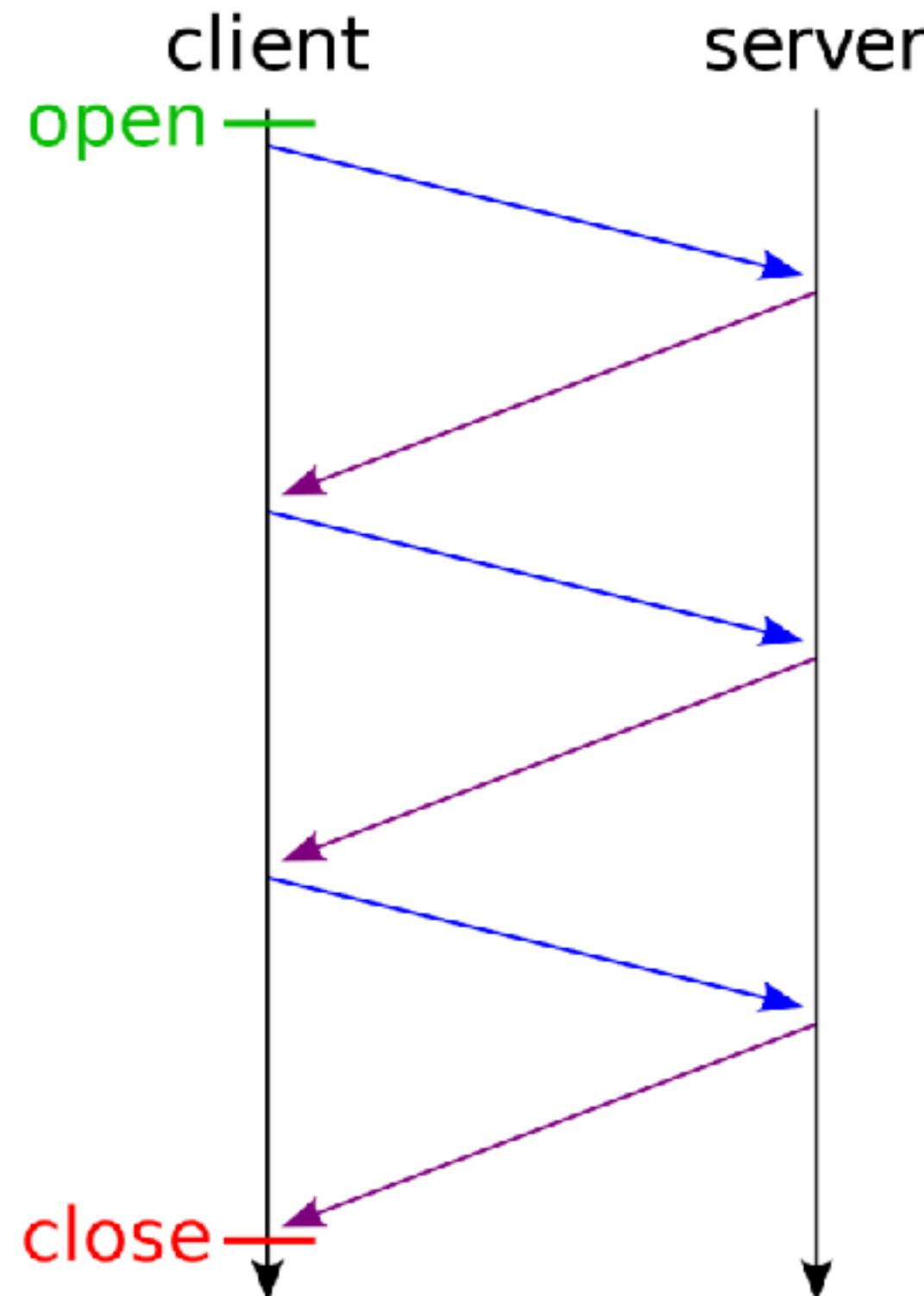
HTTP/2: What even are you?

- An upgrade to the HTTP transport layer.
- Fully multiplexed—send multiple requests in parallel.
- Allows servers to proactively push responses into client caches.

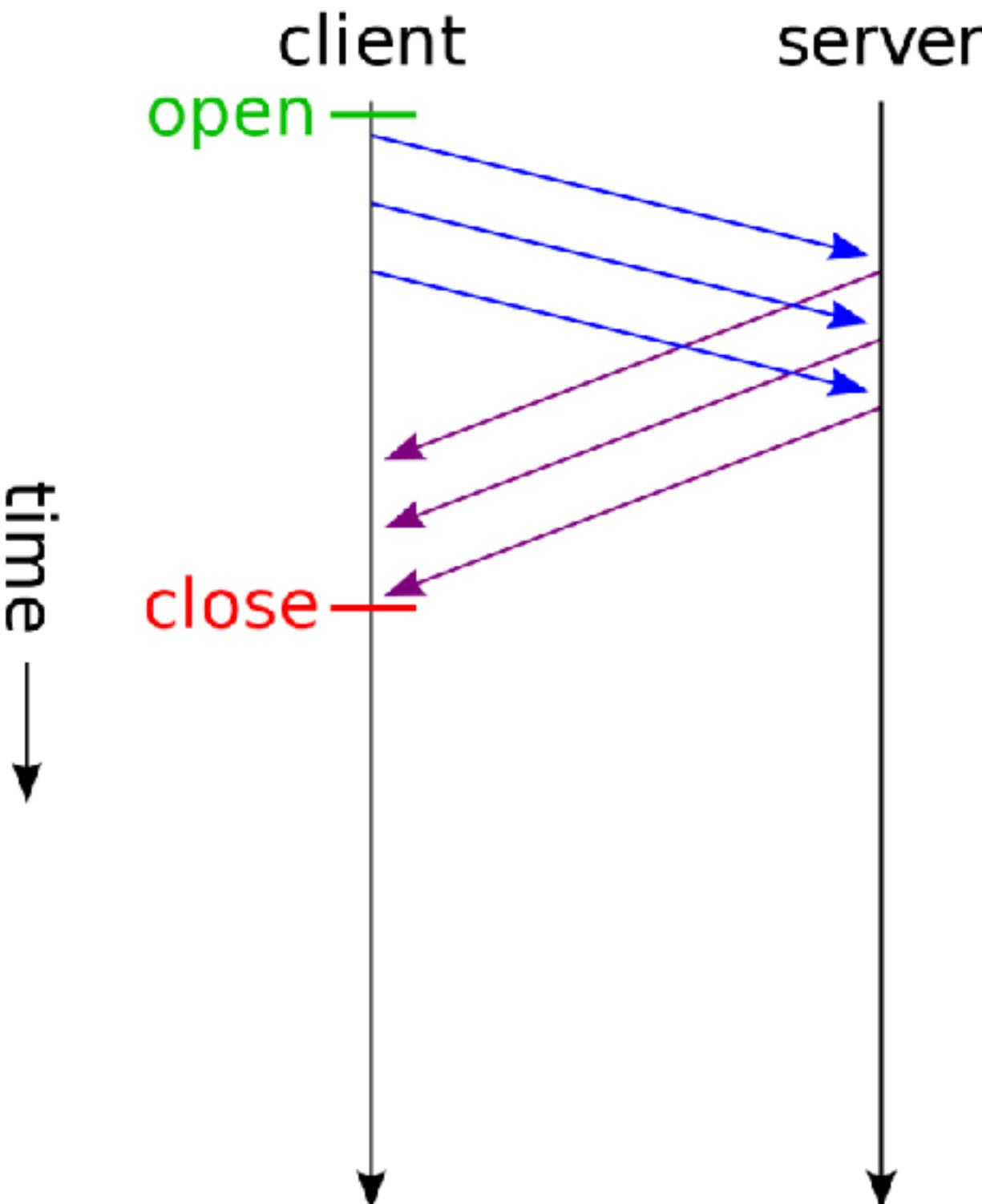
HTTP/1.1: What's wrong with you?

- Websites are growing: more images, more JavaScript
- Sure, bandwidth has gotten a lot better, but roundtrip time hasn't
- It takes just as long to ping a server now as it did 20 years ago.
- That's right: one file at a time per connection
- No big deal. It's not like we are building websites that request 100 files to something.

no pipelining



pipelining



The weird thing is that once you have this in place some “best practices” become not-so-good.

Is concatenating all of your JS and
CSS into large, single files still
useful?

What about inlining images
as data URLs in our CSS?

**Measure, measure,
measure.**

Getting HTTP/2: The easy
way.

The screenshot shows the Now website interface. At the top, there's a navigation bar with links for About, Blog, Now, World, Domains, API, OSS, Docs, Pricing, Download, Day, TY, Chat, and Login. Below the navigation is a main header with the word "Now" and a sub-header "Now – Realtime Global Deployments". The central part of the page displays two terminal-like windows. The left window shows a deployment progress bar at 38% completion. The right window shows a command-line interface with commands like 'ls', 'now', and 'curl' being run, along with a progress bar for an upload. Below these windows, there are three sections: 'Docker', 'Node.js', and 'Static Websites', each with a brief description and a command-line example.

The screenshot shows the Amazon CloudFront website. At the top, there's a navigation bar with links for 'Menu', 'Contact Sales', 'Products', 'Solutions', 'Pricing', 'Getting Started', 'More', 'English', 'My Account', and 'Sign Up'. The main title 'Amazon CloudFront' is prominently displayed in large white letters on an orange background. Below it, a sub-headline reads 'Highly secure global content delivery network (CDN)'. A yellow button labeled 'Try Amazon CloudFront For Free' is centered below the headline. At the bottom of the page, there are links for 'Amazon CloudFront Home', 'Product Details', 'Pricing', 'Getting Started', 'Resources', and 'Case Studies'. To the right, there's a video player showing two men speaking at an event, with the word 'Invent' visible.

Netlify All-in-one platform for [build, deploy & deliver](#)

Secure <https://www.netlify.com>

Features Pricing Docs Blog Support [Search](#)

 netlify

Log in Sign Up

Write frontend code. Push it. We handle the rest.

All the features developers need right out of the box: Global CDN, Continuous Deployment, one click HTTPS and more...

[Get started for free](#)

Deploy for free with no credit card required.

AM169255 - All teams

Human Circle
Empty team

Dallas
Abstract
Netlify
Create a new team

gen.com
gen.com

Claimed by Netlify
Last update at 11:06 pm (an hour ago)

easp.org
easp.org

Claimed by Netlify
Last update at 11:24 pm (2 hours ago)

measuredscms.org
measuredscms.org

Claimed by Netlify

A screenshot of the Cloudflare website. The header features the Cloudflare logo, navigation links for Products, Solutions, Resources, Pricing, Support, Login, Sign Up, and a button for 'Under Attack?'. The main headline reads 'Cloudflare makes your ecommerce stores faster & safer.' Below it is a call-to-action button 'Get Started Today' and a sign-up form with fields for Email and Password, and a 'Sign Up' button. A purple banner at the bottom introduces 'Cloudflare Workers' with a 'LEARN MORE' button.

This slide is for you to locate the services mentioned in the previous slide—because I ❤️ you.

- <https://now.sh>
- <https://aws.amazon.com/cloudfront/>
- <https://cloudflare.com>
- <https://netlify.com>

Chapter Four

Build It Faster. Build It Smarter.

When in doubt, let your
tools help you.

Steve

GitHub, Inc. [US] | https://github.com/purifycss/purifycss

This repository Search Pull requests Issues Marketplace Explore

purifycss / purifycss Watch 179 Star 8,241 Fork 320

Code Issues 49 Pull requests 11 Projects 0 Wiki Insights

Remove unused CSS. Also works with single-page apps.

302 commits 1 branch 8 releases 16 contributors MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

Illyism committed on Jun 6, 2017 v1.2.6 Latest commit d8a29b4 on Jun 6, 2017

__tests__	revert modify tests	9 months ago
bin	Add whitelist option	9 months ago
config	revert prepack	9 months ago
lib	revert prepack	9 months ago
src	revert prepack	9 months ago
.eslintrc	Remove files from package.json	10 months ago
.gitignore	Tests Jest	11 months ago
.travis.yml	Remove editor config, change travis.yml	11 months ago
LICENSE	2016 License	2 years ago
README.md	Update README: whitelist cli	9 months ago

RABBI

Lab

Paying the Babel Tax™

Let's file these under
“maybe use.”

**babel-preset-env is your
friend.**

```
{  
  "presets": ["env"]  
}
```

```
{  
  "presets": [  
    ["env", {  
      "targets": {  
        "browsers": ["last 2 versions", "safari >= 7"]  
      }  
    }]  
  ]  
}
```

**npm install @babel/plugin-
transform-react-remove-prop-
types**

```
import React from 'react';
import PropTypes from 'prop-types';

class Announcement extends React.Component {
  render() {
    return (
      <article>
        <h1>Important Announcement</h1>
        <p>{this.props.content}</p>
      </article>
    )
  }
}

Announcement.propTypes = {
  content: PropTypes.string,
}

export default Announcement;
```

```
import React from 'react';
import PropTypes from 'prop-types';

class Announcement extends React.Component {
  render() {
    return <article>
      <h1>Important Announcement</h1>
      <p>{this.props.content}</p>
    </article>;
  }
}

export default Announcement;
```

```
{  
  "plugins": [  
    "syntax-jsx",  
    [ "transform-react-remove-prop-types",  
      {  
        "removeImport": true  
      }  
    ]  
  ]  
}
```

```
import React from 'react';

class Announcement extends React.Component {
  render() {
    return <article>
      <h1>Important Announcement</h1>
      <p>{this.props.content}</p>
    </article>;
  }
}

export default Announcement;
```

```
{  
  "plugins": [  
    "syntax-jsx",  
    [ "transform-react-remove-prop-types",  
      {  
        "mode": "wrap"  
      }  
    ]  
  ]  
}
```

```
Announcement.propTypes = process.env.NODE_ENV !== "production" ? {  
  content: PropTypes.string  
} : {};
```

npm install babel-plugin-transform-react-pure-class-to-function

This is another one of those plugins
that does what it says on the tin.

```
import React from 'react';

class Announcement extends React.Component {
  render() {
    return (
      <article>
        <h1>Important Announcement</h1>
        <p>{this.props.content}</p>
      </article>
    )
  }
}

export default Announcement;
```

```
import React from 'react';

function Announcement(props) {
  return <article>
    <h1>Important Announcement</h1>
    <p>{props.content}</p>
  </article>;
}

export default Announcement;
```

What if the component has state?

```
class Counter extends React.Component {  
  constructor() {  
    super();  
    this.state = { count: 0 };  
  }  
  
  render() {  
    return <p>{this.state.count}</p>;  
  }  
}
```

```
class Counter extends React.Component {  
  constructor() {  
    super();  
    this.state = { count: 0 };  
  }  
  
  render() {  
    return <p>{this.state.count}</p>;  
  }  
}
```

Why is this cool?

```
npm install @babel/plugin-  
transform-react-inline-elements
```

What is this and why is it
fast?

This...

```
<div className="important">Hello World</div>
```

...becomes...

```
React.createElement(  
  "div",  
  { className: "important" },  
  "Hello World"  
);
```

...becomes...

```
{  
  $$typeof: [object Symbol] { ... },  
  _owner: null,  
  _store: [object Object] { ... },  
  key: null,  
  props: {  
    children: "Hello World",  
    className: "important"  
  },  
  ref: null,  
  type: "div"  
}
```

Not doing something is
faster than doing it.

But, if you have to do it, then you
might as well only do it once.

Do it at build time.

Admission: what actually comes out of this plugin is a little grosser.

```
var _jsx = function () { var REACT_ELEMENT_TYPE = typeof Symbol
  === "function" && Symbol.for && Symbol.for("react.element") ||
  0xeac7; return function createRawReactElement(type, props, key,
  children) { var defaultProps = type && type.defaultProps; var
  childrenLength = arguments.length - 3; if (!props &&
  childrenLength !== 0) { props = {}; } if (props && defaultProps) {
  for (var propName in defaultProps) { if (props[propName] === void
  0) { props[propName] = defaultProps[propName]; } } } else if (!
  props) { props = defaultProps || {};} if (childrenLength === 1)
  { props.children = children; } else if (childrenLength > 1) { var
  childArray = Array(childrenLength); for (var i = 0; i <
  childrenLength; i++) { childArray[i] = arguments[i + 3]; }
  props.children = childArray; } return { $$typeof:
  REACT_ELEMENT_TYPE, type: type, key: key === undefined ? null : ''
  + key, ref: null, props: props, _owner: null }; }; }();

_jsx("div", {}, void 0, "Hello World");
```

```
npm install @babel/plugin-  
transform-react-constant-  
elements
```

```
import React from 'react';

class Article extends React.Component {
  render() {
    return (
      <article>
        <h1>Important Announcement</h1>
        <p>{this.props.content}</p>
        <footer>—The Management</footer>
      </article>
    )
  }
}

export default Announcement;
```

```
import React from 'react';

var _ref = <h1>Important Announcement</h1>

var _ref2 = <footer>—The Management</footer>

class Announcement extends React.Component {
  render() {
    return <article>
      {_ref}
      <p>{this.props.content}</p>
      {_ref2}
    </article>;
  }
}

export default Announcement;
```

The screenshot shows a web browser window with the URL "prepack.io" in the address bar. The page has a yellow header bar with the word "Prepack" on the left and navigation links for "ABOUT", "GETTING STARTED", "FAQS", "TRY IT OUT", and "GITHUB" on the right. Below the header is a large yellow section containing the title "Prepack" in a large, bold, sans-serif font. Underneath the title is the subtitle "A tool for making JavaScript code run faster." in a smaller white sans-serif font. A note below the subtitle states: "*Prepack is still in an early development stage and not ready for production use just yet. Please try it out, give feedback, and help fix bugs." At the bottom of this section are two buttons: "Getting Started" and "Try It Out", both in white text on a dark background. The main content area below this is white and features the heading "What does it do?" followed by a paragraph explaining the tool's purpose. At the very bottom of the page is a thin white footer bar with the text "Add "https://prepack.io/getting-started.html" to Reading List".

Prepack

ABOUT GETTING STARTED FAQS TRY IT OUT GITHUB

Prepack

A tool for making JavaScript code run faster.

*Prepack is still in an early development stage and not ready for production use just yet. Please try it out, give feedback, and help fix bugs.

Getting Started Try It Out

What does it do?

Prepack is a tool that optimizes JavaScript source code: Computations that can be done at compile-time instead of run-time get eliminated. Prepack replaces the global code of a JavaScript bundle with equivalent code that is a simple sequence of assignments. This gets rid of most intermediate computations and object allocations.

Add "https://prepack.io/getting-started.html" to Reading List

```
for (let x = 0; x < 10; x++) {  
    console.log(x);  
}
```

```
console.log(0);
console.log(1);
console.log(2);
console.log(3);
console.log(4);
console.log(5);
console.log(6);
console.log(7);
console.log(8);
console.log(9);
```

```
(function () {
  function fibonacci(x) {
    return x <= 1 ? x : fibonacci(x - 1) + fibonacci(x - 2);
  }
  global.x = fibonacci(10);
})());
```

The screenshot shows the Prepack.io web application interface. At the top, there is a navigation bar with links for ABOUT, GETTING STARTED, FAQS, TRY IT OUT, and GITHUB. Below the navigation bar, the main content area has a title "Fibonacci" with a dropdown arrow, an "OPTIONS" button, a preview panel, and a toolbar with "SAVE" and "DELETE" buttons.

Code Editor:

```
1 - (function () {
2 -   function fibonacci(x) {
3 -     return x <= 1 ? x : fibonacci(x - 1) + fibonacci(
4 -   }
5 -   global.x = fibonacci(10);
6 - })();
```

Preview Panel:

```
1 x = 55;
```

Prepack is not production ready, but
it's an interesting idea and you
should be thinking along these lines.

Epilogue

Closing Thoughts, Recommendations,
and Further Research

Free tip (repeated): Make sure you're running in “Production Mode”.

What is production mode?

- PropTypes? Nah.
- “Helpful” warnings? No, thanks.
- Performance metrics? Nope.
- Not having this stuff relieves React of a bunch of work and allows it to run fast.

SendGrid Marketing Campaign x Steve

192.168.99.100:3000/marketing_campaigns/ui/campaigns

first name last name

Marketing Campaigns

Dashboard

Marketing

Marketing Campaigns

Tour

Overview

Campaigns

Contacts

This page is using the development build of React. 🚧

Note that the development build is not suitable for production.
Make sure to [use the production build](#) before deployment.

Open the developer tools, and the React tab will appear to the right.

All Sent Draft In Progress Canceled Scheduled

Search By Name

%

<input type="checkbox"/> CAMPAIGN NAME	DELIVERED	UNIQUE OPENS	UNIQUE CLICKS	
	DELIVERED/REQUESTS	UNIQUE OPENS/DELIVERED	UNIQUE CLICKS/DELIVERED	UN

This page is using the production build of React. ✓
Open the developer tools, and the React tab will appear to the right.

Steve Kinney

Your trial expired. You must upgrade to continue using this feature.

Campaigns

Create Campaign

Filter By Status

All Sent Draft In Progress Canceled Scheduled

Marketing Campaigns

Search By Name

%

Feedback

Steve

Secure | https://sendgrid.com/marketing_campaigns/ui/campaigns

Steve Kinney

Dashboard

Marketing

Marketing Campaigns

Experiments

Some super important things we didn't talk about today.

- Server-side rendering.
- Image performance.
- Loading web fonts.
- Progressive web applications.



Avoid Render Blocking

Render blocking is anything that keeps the browser from painting to the screen—or, umm, rendering.

Quick Tip: Make sure your CSS
`<link>`s are in the `<head>`.

Inlining has trade offs.

- Sure, it saves you a network request.
- But, you can't cache the styles.
- So, while it might work better for single-page applications, you wouldn't want to include it every HTML page on multi-page applications.
- With HTTP/2, you can actually just avoid this problem all together.