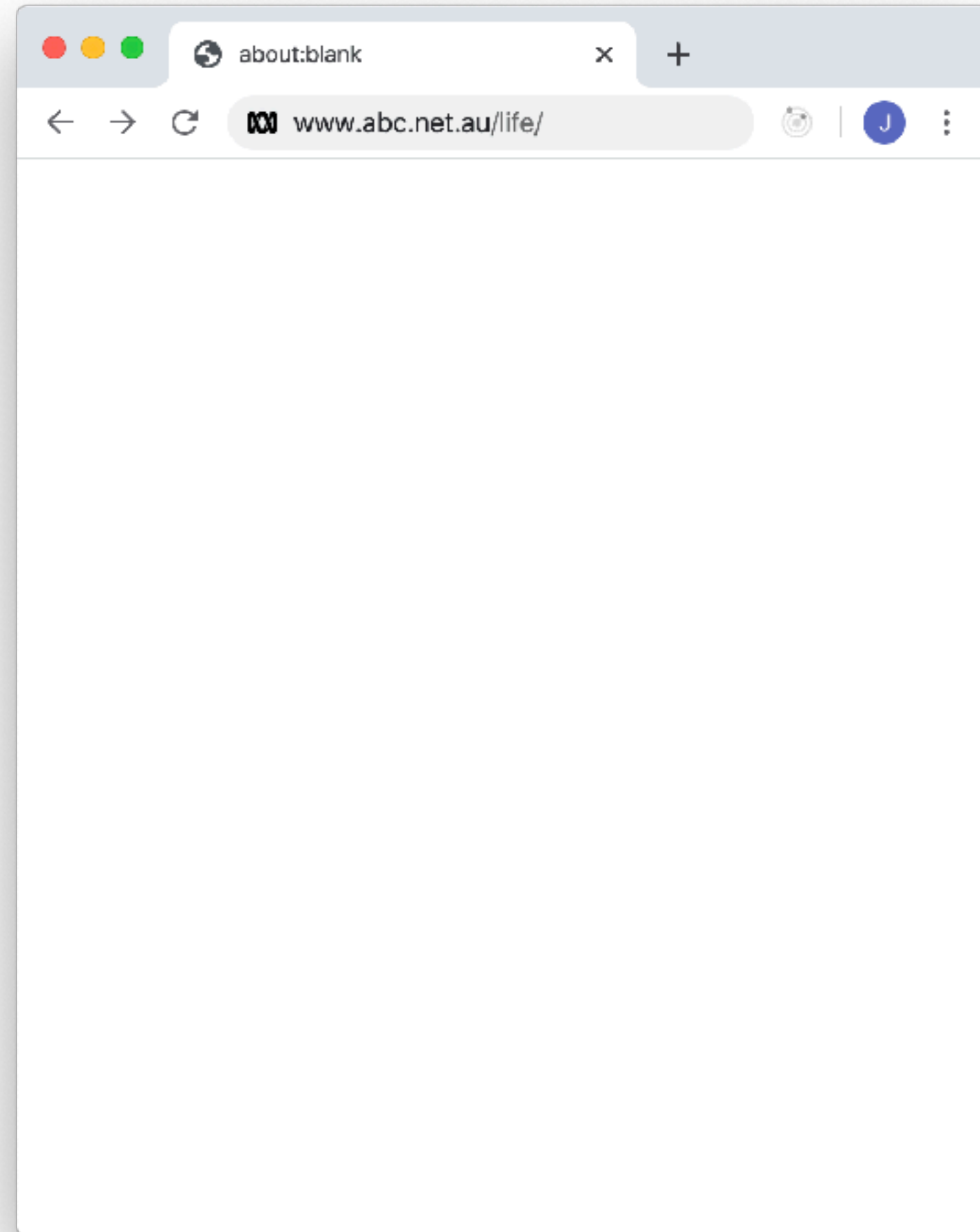


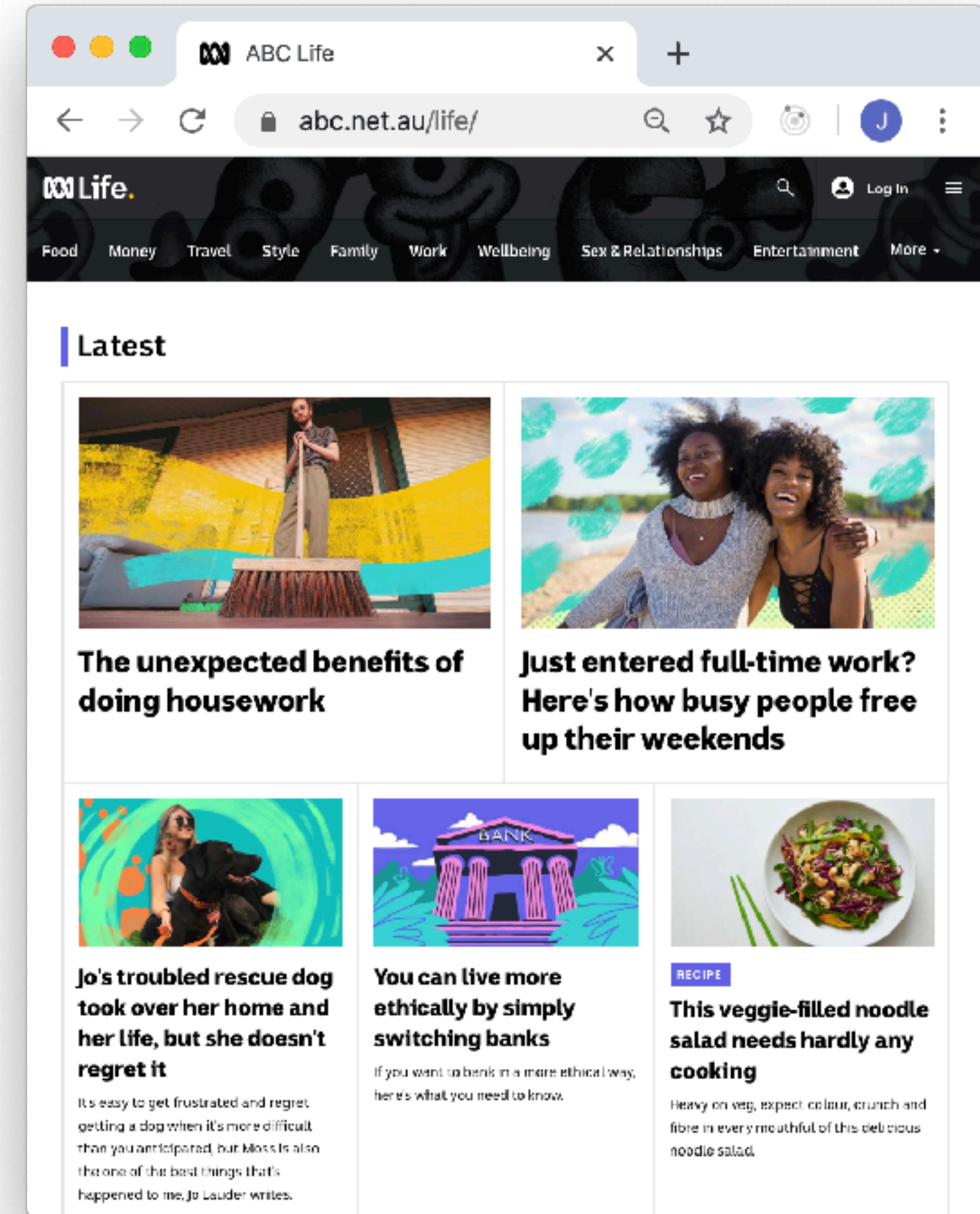
THE FIRST TWO SECONDS

FASTER PAGE  
LOADS FOR REACT

@joshduck



— Stuff happens —>



# React is just JavaScript

Browsers have twenty years of  
optimisations for rendering and JS



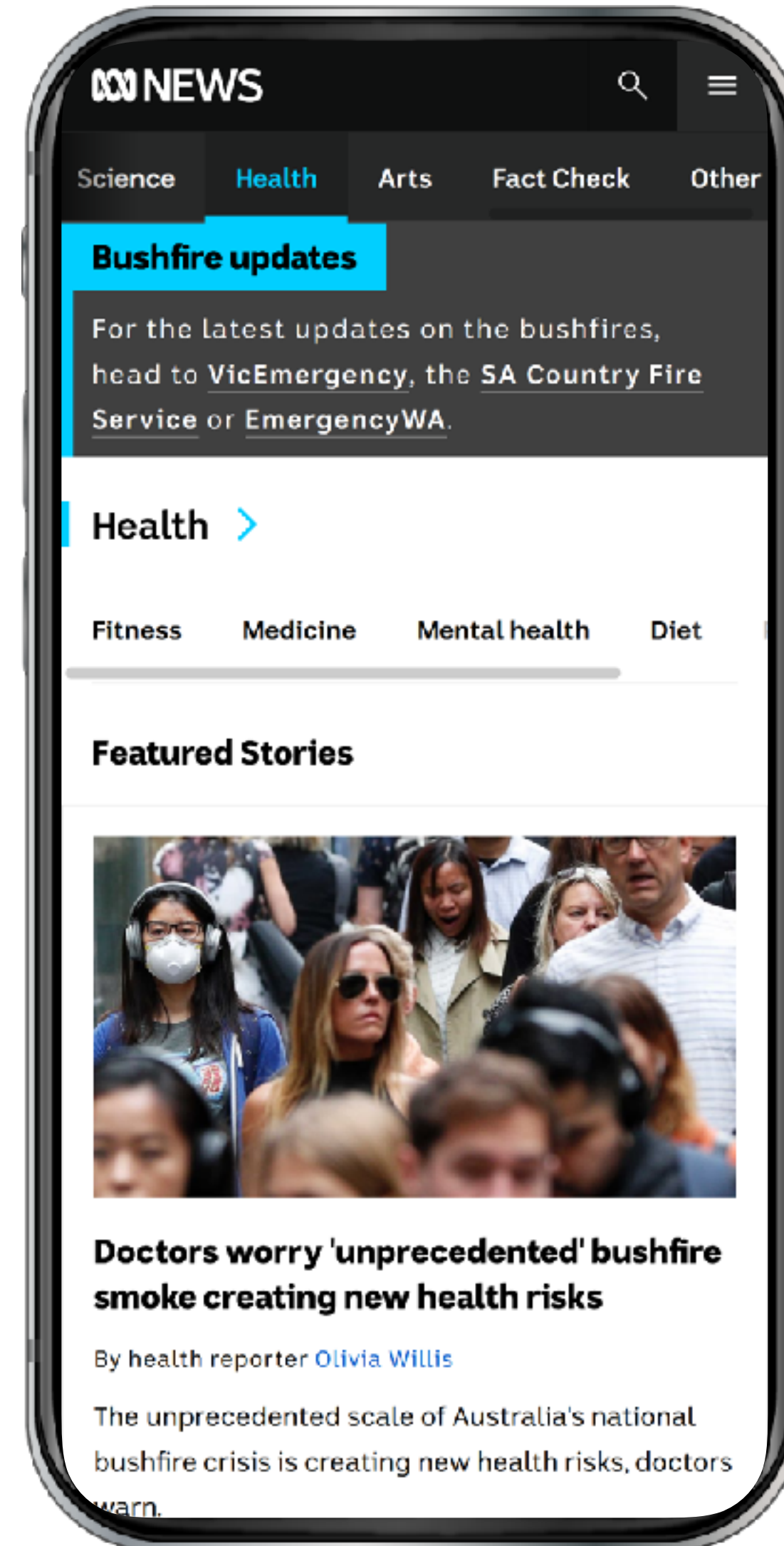
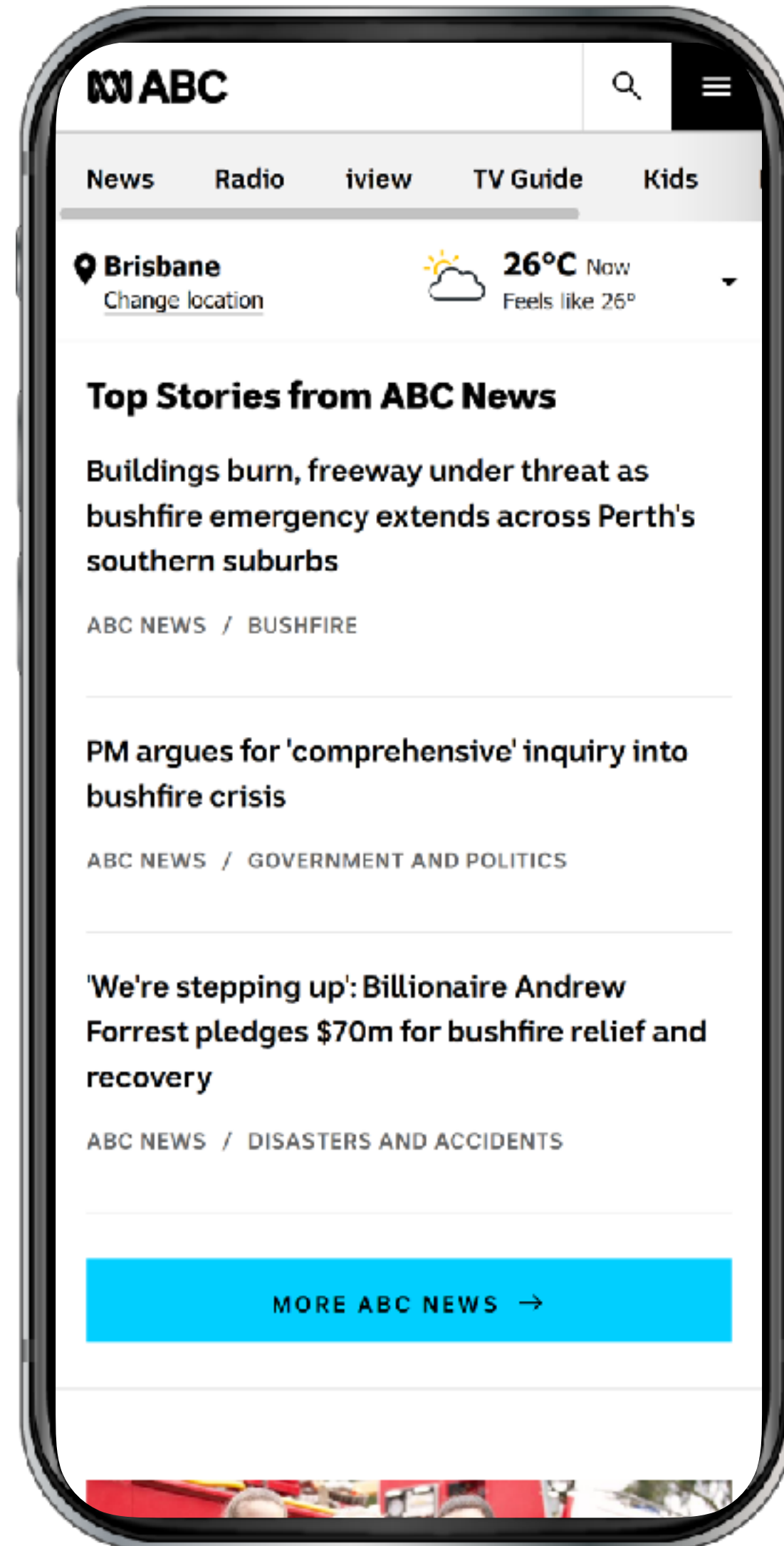
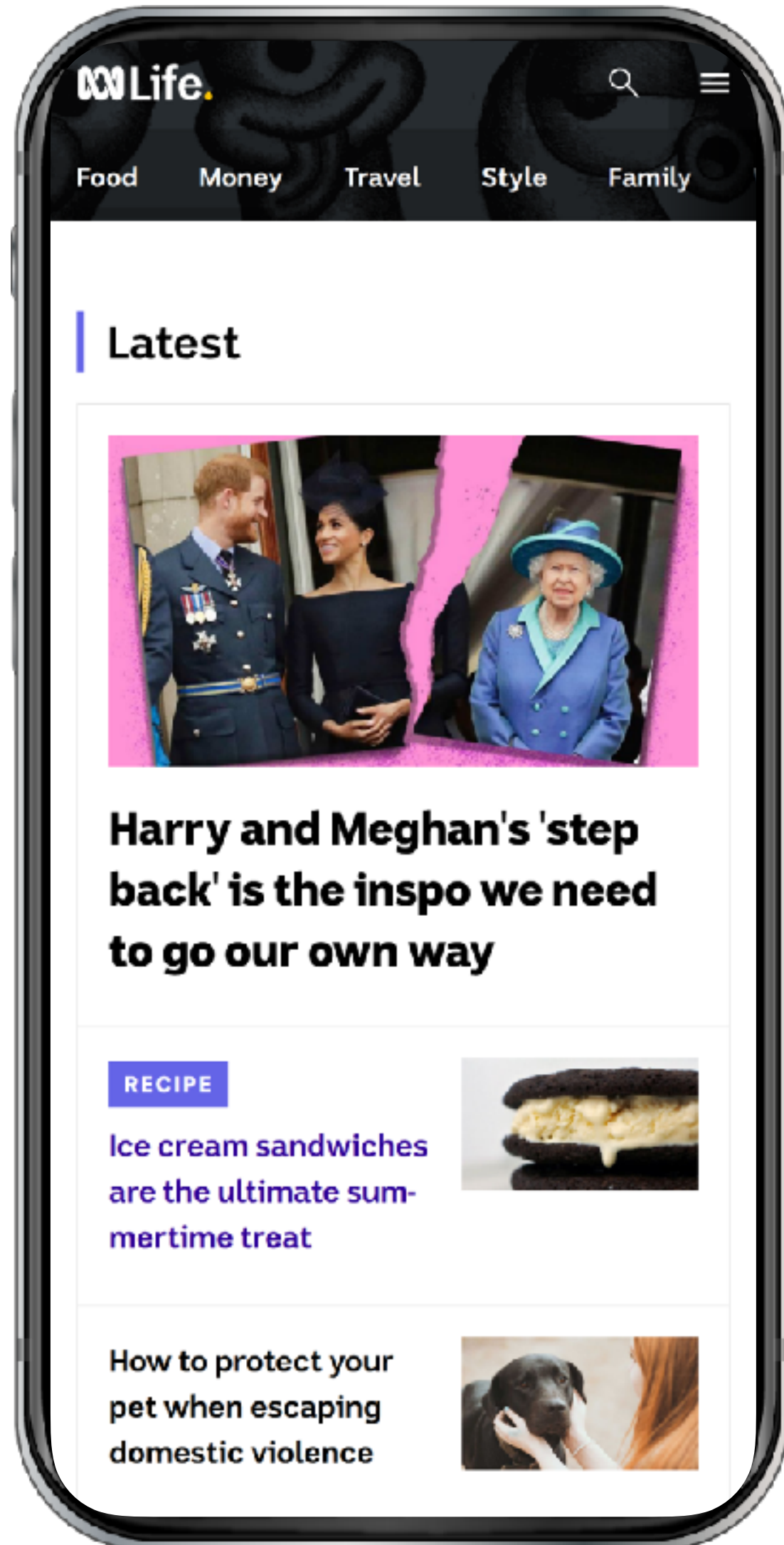
Josh Duck  
@joshduck

Engineering manager at:

clipchamp

WABC

facebook



# Today we'll cover

---

Key metrics, and our guiding model

Our key architectural decisions

Fixing performance bottlenecks

Our take-aways from the journey

LET'S START AT THE VERY BEGINNING

# Defining the architecture

---

# Server Side Rendering

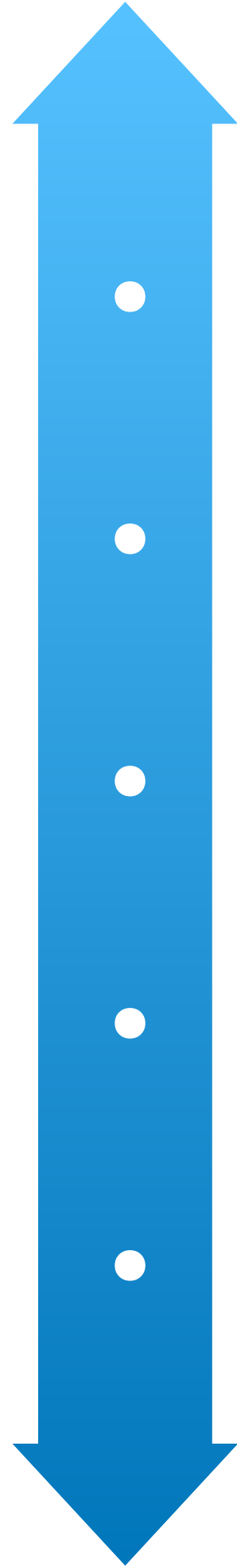
Components are rendered to markup on the server

+

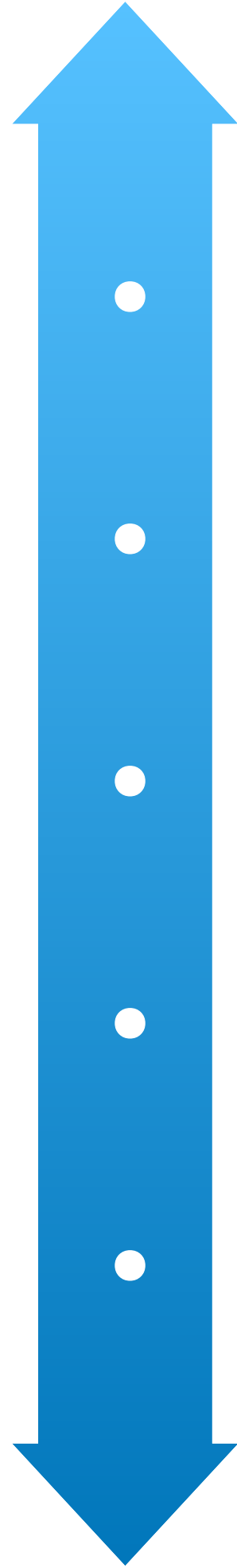
# Data Hydration

Initial data is included in the HTML payload (fetch-then-render)



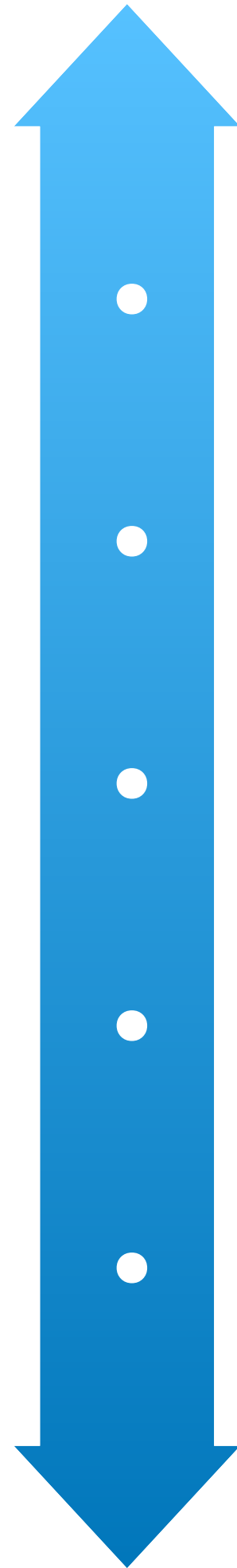


- Client rendering (like create-react-app)
- **Server render with client hydration (like Next.js)**
- Server render with progressive hydration
- Server render with partial hydration
- Server render with no client JS



- Client rendering (like create-react-app)
- Server render with client hydration (like Next.js)
- Server render with progressive hydration
- Server render with partial hydration
- **Server render with no client JS**





- Client rendering (like create-react-app)
- Server render with client hydration (like Next.js)
- **Server render with progressive hydration**
- Server render with partial hydration
- Server render with no client JS

Load Sidebar JS code

Load Content JS code

👉 Press these buttons to simulate JS code loading over the network!

## React Progressive Hydration Demo\*

*\* very experimental — likely contains bugs.*

This app is server-rendered to HTML. Concurrent Mode lets us hydrate parts of UI without waiting for *all* JS to load.

Sidebar (Initial HTML)

Clicked on sidebar 0 times

Content (Initial HTML)

Clicked on content 0 times

# Server render with CDN and caching



- Static site generation (like Gatsby)
- **Server rendering with CDN/caching**
- Server rendering per request (like Express)

# CSS media queries

(Yes, really)



# Our architecture

---

Server side rendering

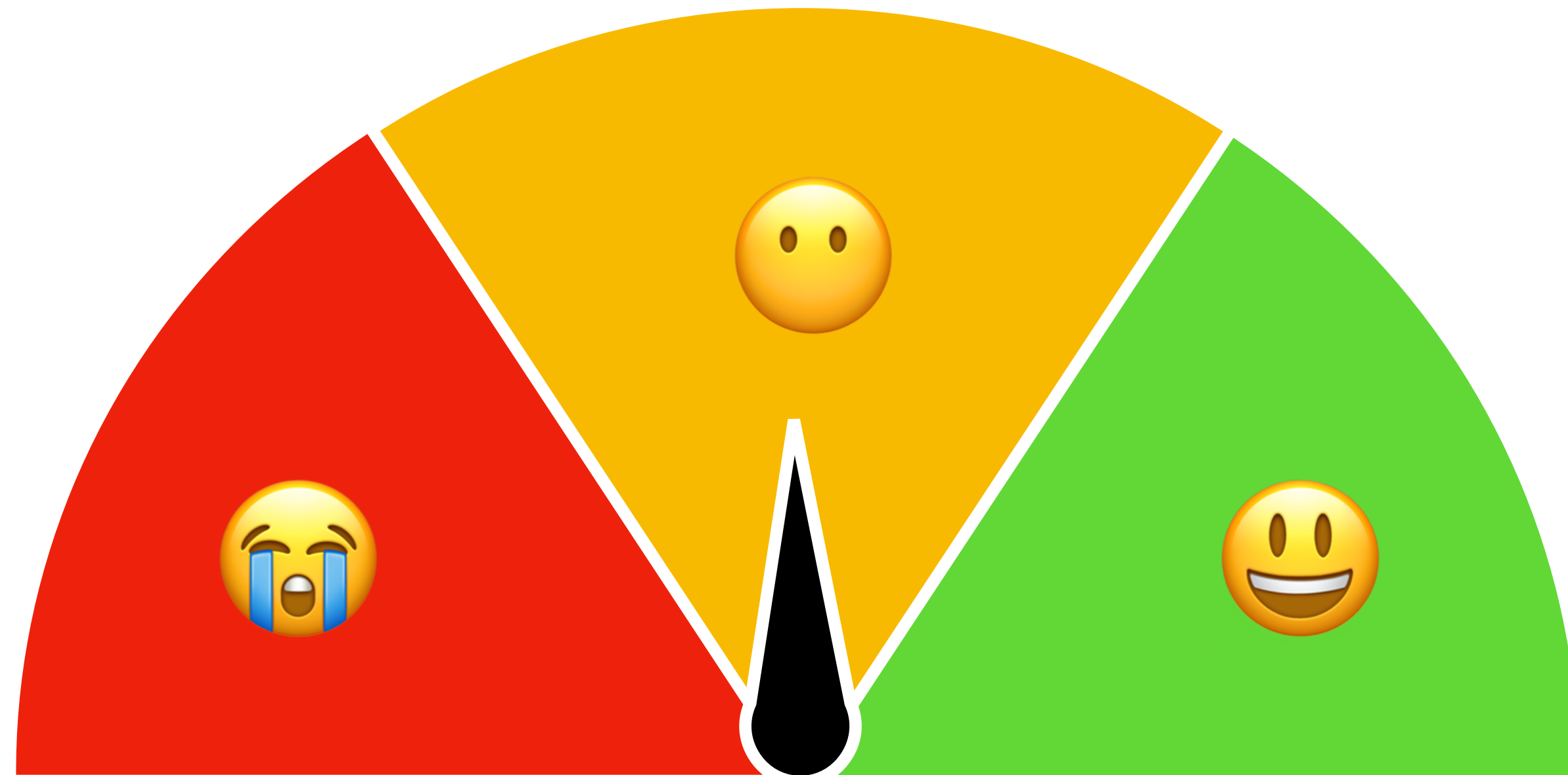
Client hydration

Caching via CDN

CSS media queries

# Our performance was...

---



# Performance matters

---

Better user experience

Better conversions

Better SEO

# Google punishes slow sites

---

The “Speed Update,” as we’re calling it, will only affect pages that deliver the slowest experience to users and will only affect a small percentage of queries. It applies the same standard to all pages, regardless of the technology used to build the page. The intent of the search query is still a very strong signal, so a slow page may still rank highly if it has great, relevant content.



A. Developer in Intellectually Dishonest

Jan 22 · 8 min read



## We made our pages faster and increased revenue

We completely redesigned the site and checkout too, but that part of the story won't get us an awesome like count

Read more...

1.2K

4 responses

# Response time in man-computer conversational transactions

by ROBERT B. MILLER

Fall Joint Computer Conference, 1968

*International Business Machines Corporation*  
Poughkeepsie, New York

## INTRODUCTION AND MAJOR CONCEPTS

The literature concerning man-computer transactions abounds in controversy about the limits of "system response time" to a user's command or inquiry at a terminal. Two major semantic issues prohibit resolving this controversy. One issue centers around the question of "Response time to what?" The implication is that different human purposes and actions will have different acceptable or useful response times.

This paper attempts a rather exact and definition of different classes and purpose at terminals of various types. It will be shown that "two-second response time" is a universal requirement.

The second semantic question is "What is a need or requirement?" In the present discussion, the reader is asked to accept the following definition: "A need or requirement is some demonstrably better alternative in a set of competing known alternatives that enable a human purpose or action to be implemented." This definition intentionally ignores the problem of value versus cost. It is not

## Operating needs and psychological needs

An example of an operating need is that unless a given airplane's velocity exceeds its stall speed, the airplane will fall to earth. Velocity above stall speed is an undebatable operating need. In a superficially different context, it is a "fact" (let's assume we know the numbers) that when airline customers make reservations over a telephone, any delays in completing transactions above five min-

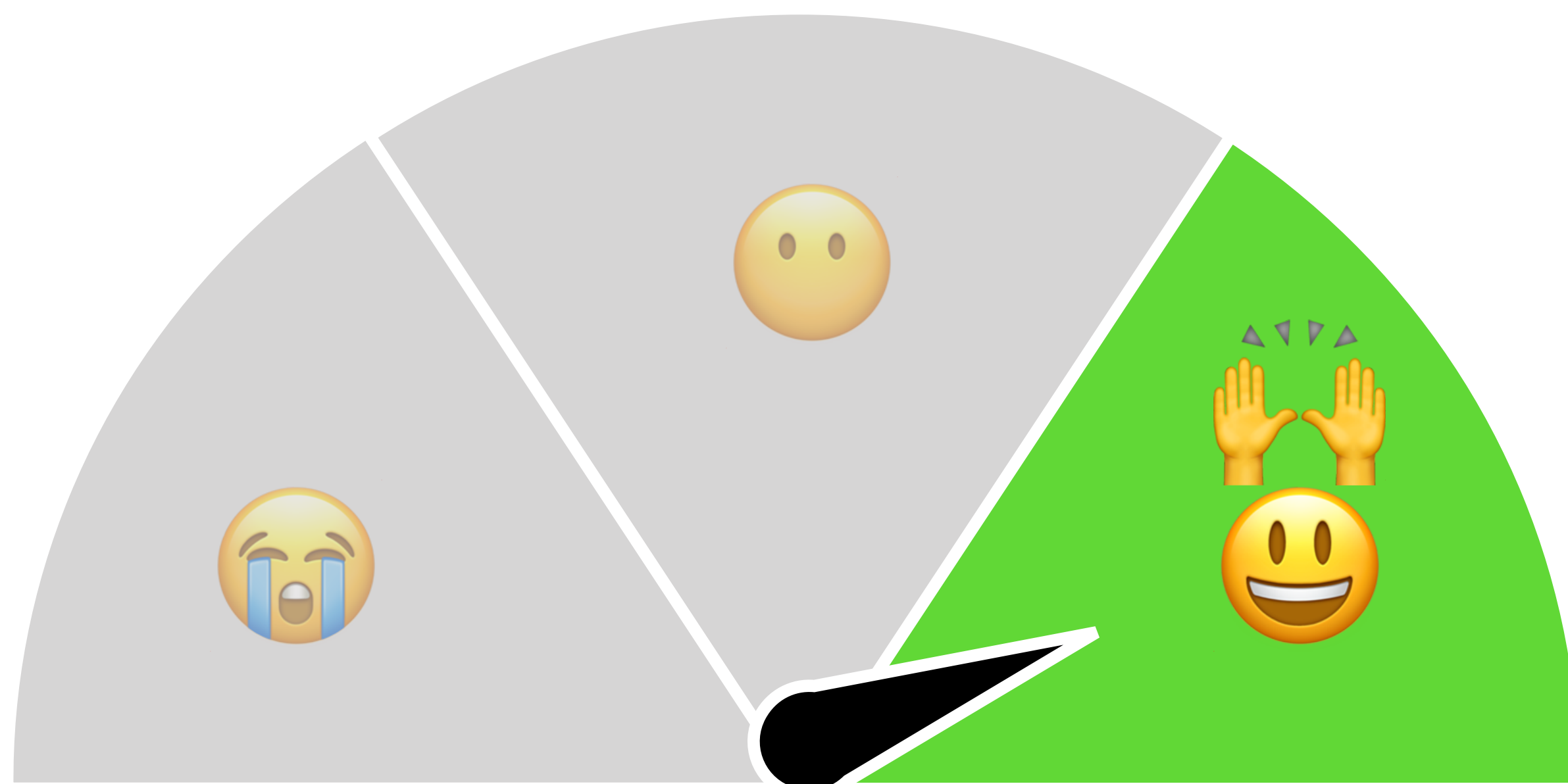
**That is why the tasks which humans can and will perform with machine communications will seriously change their character if response delays are greater than two seconds, with some possible**

into the problems of operating needs except to mention when they may be more significant than a psychological need. The following topics address psychological needs.

## Response to expectancies

# What we wanted

---



The right architecture and  
right technical solutions  
alone won't make you fast





Blazing fast, simple and co

Blazing fast and accurate glob

A blazing fast js bundle



Blazing fast 1kb search library



Blazing fast, zero config



Blazing fast blog

A blazing fast library for viewin

A blazing fast React alterna



A blazing-fast Single Pag

[DEPRECATED] Blazing fast tile base

Performance comes from  
measuring and improving  
and iterating

HOW CAN WE BE FAST?

Learning what to measure

---

# Lighthouse performance



Performance

## Metrics



● First Contentful Paint	2.2 s	● First Meaningful Paint	2.2 s
● Speed Index	2.4 s	▲ First CPU Idle	8.8 s
▲ Time to Interactive	10.5 s	▲ Max Potential First Input Delay	1,320 ms



# Standard point-in-time metrics

---

First Byte

FCP

First Contentful Paint  
Any DOM paint

FMP

First Meaningful Paint  
User defined hero element

LCP

Largest Contentful Paint  
Largest node was rendered

TTI

Time to Interactive  
Large JS chunks have finished

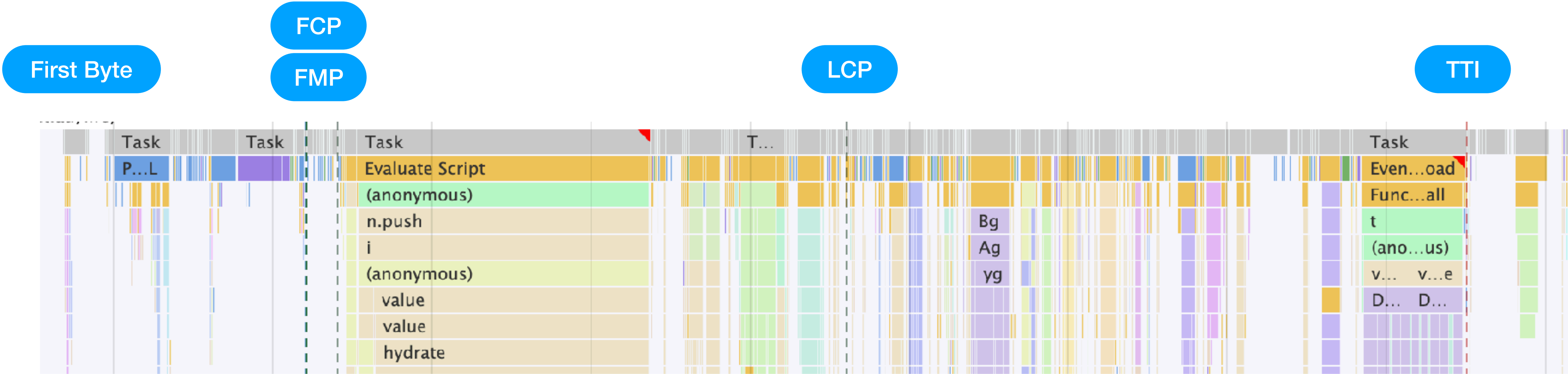


Surma  
@DasSurma

We have a new 3 letter acronym performance metrics.

This is not a drill. I repeat:  
New 3 letter acronym.

# Standard point-in-time metrics



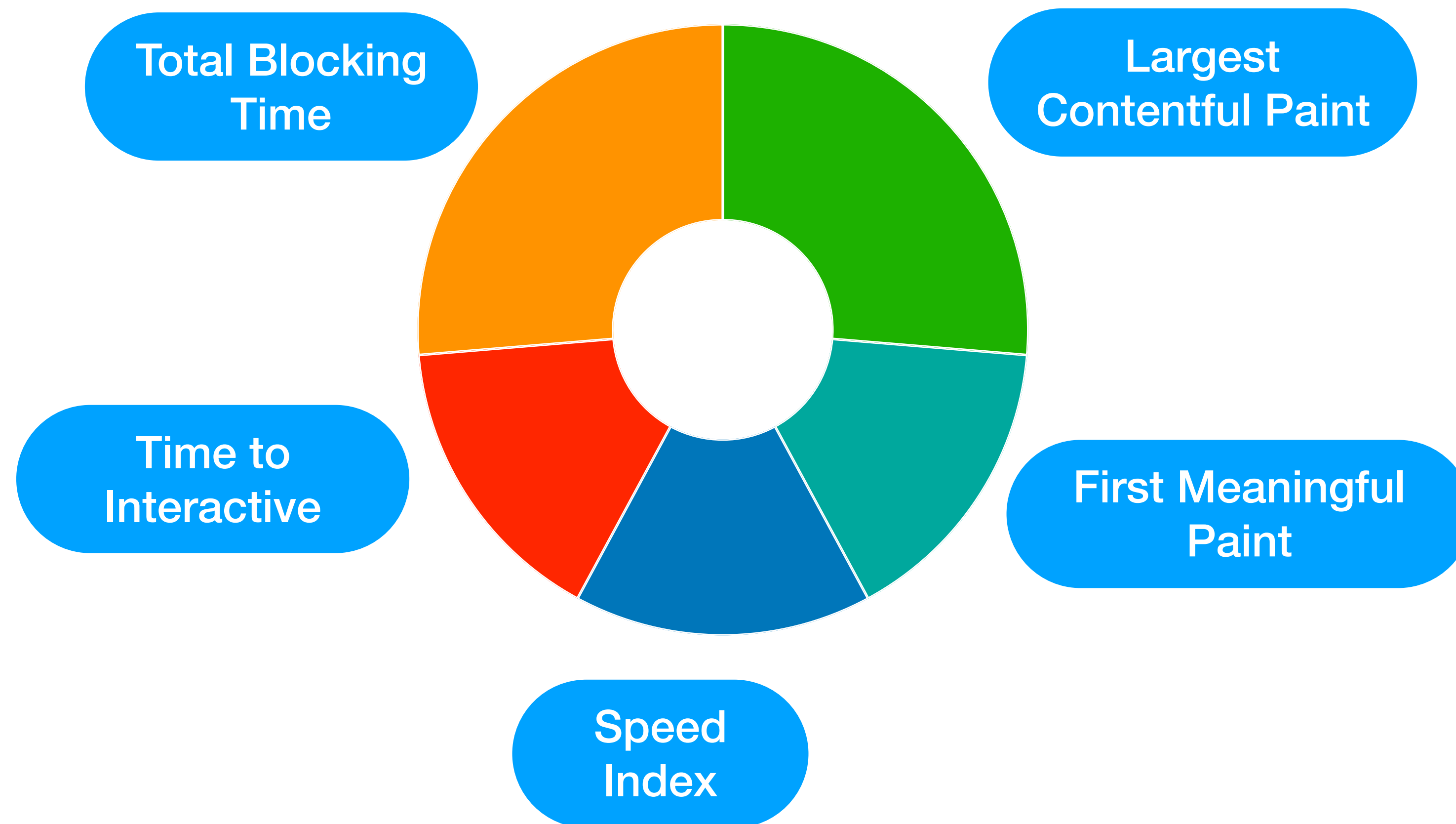
# Speed index score

---



# Lighthouse performance

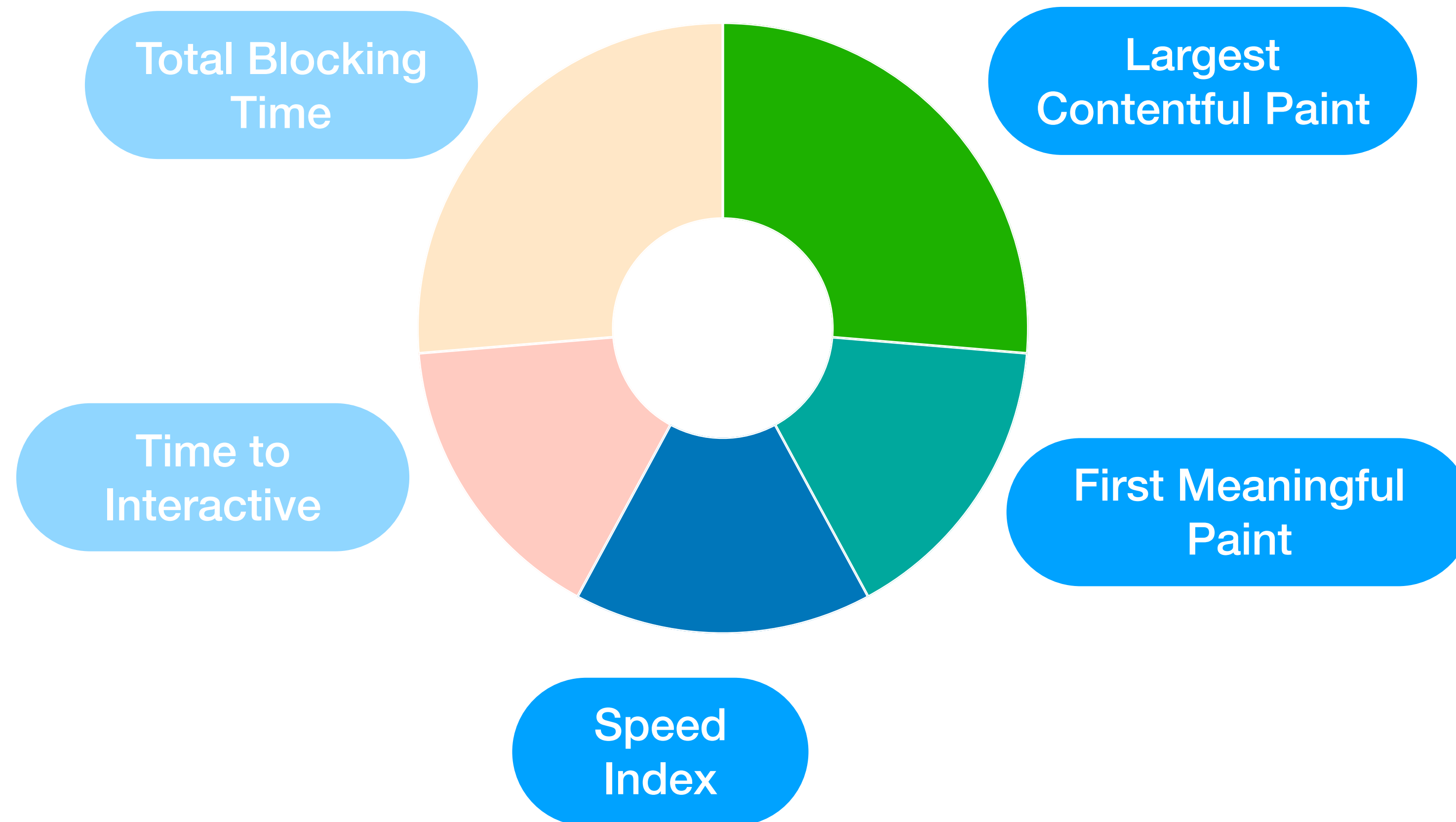
---





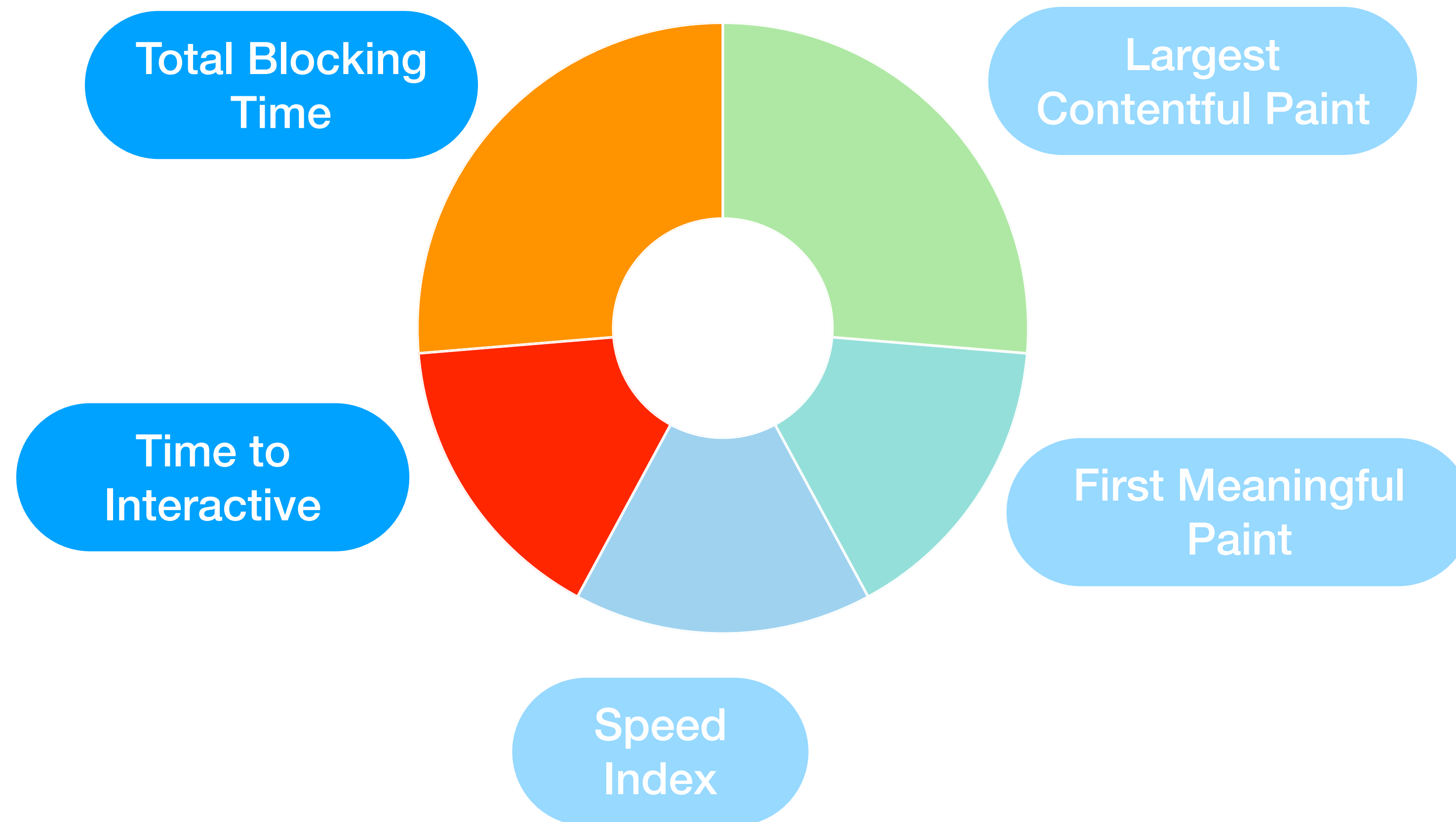
# Lighthouse performance

---

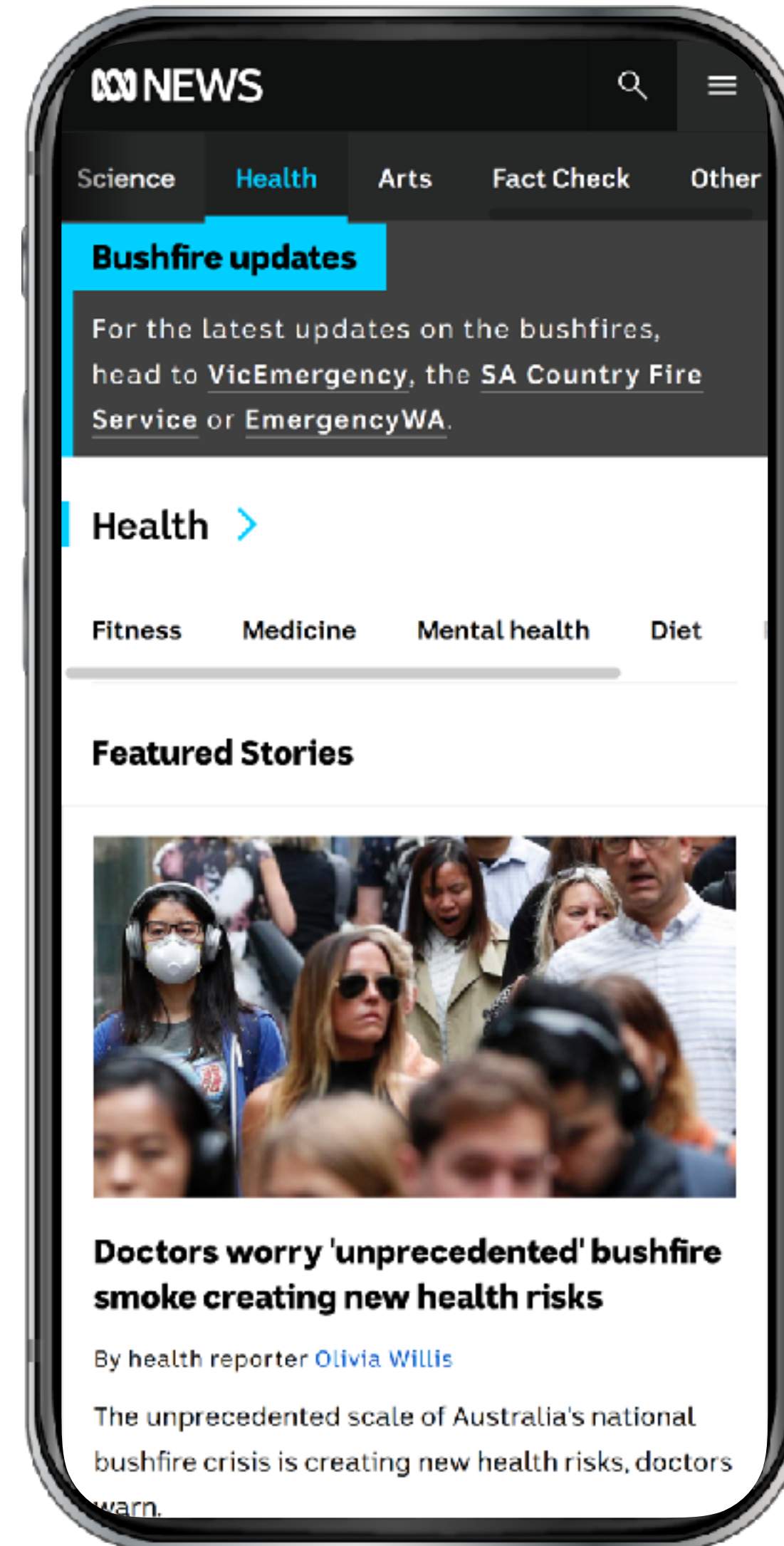
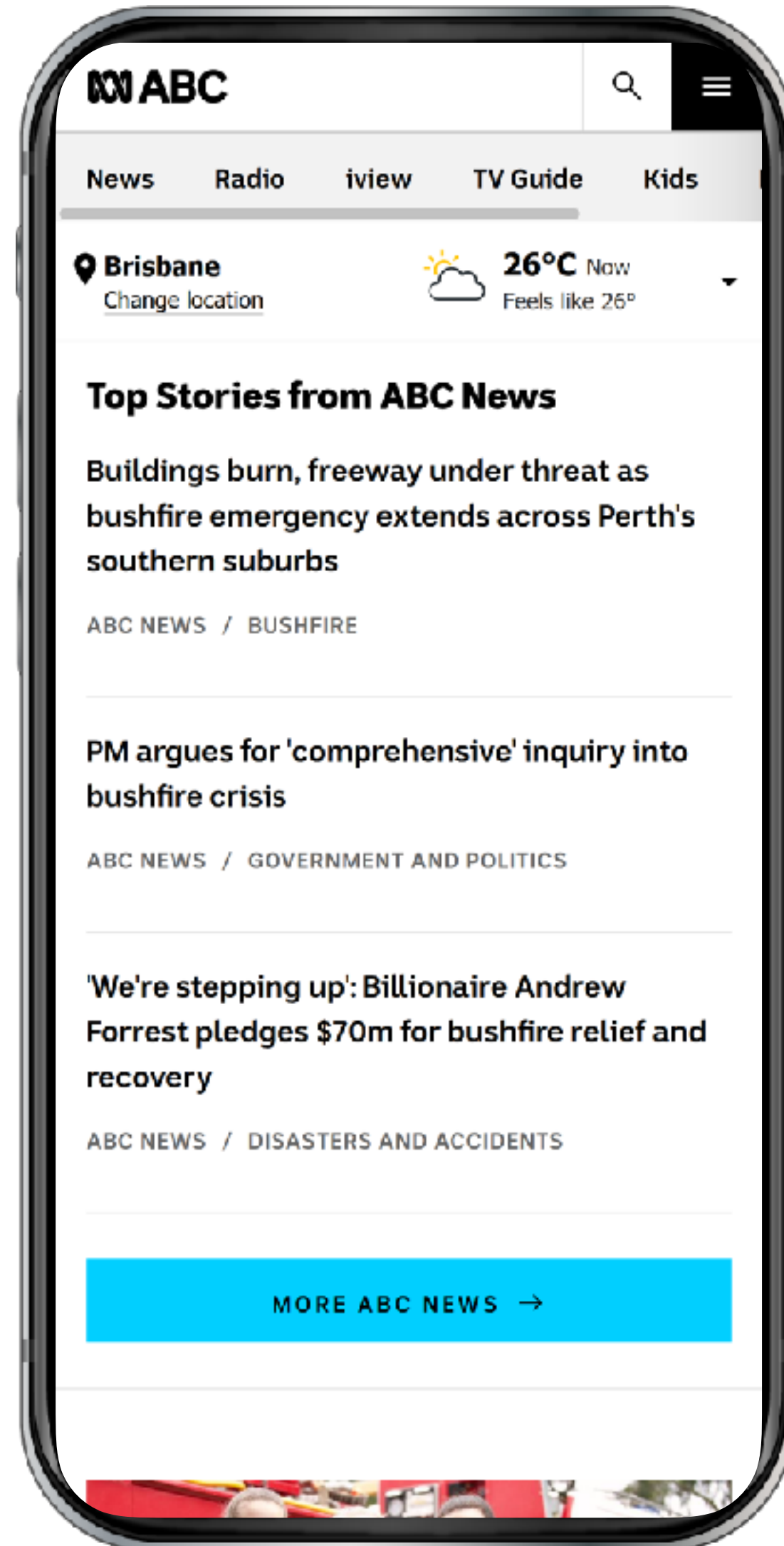
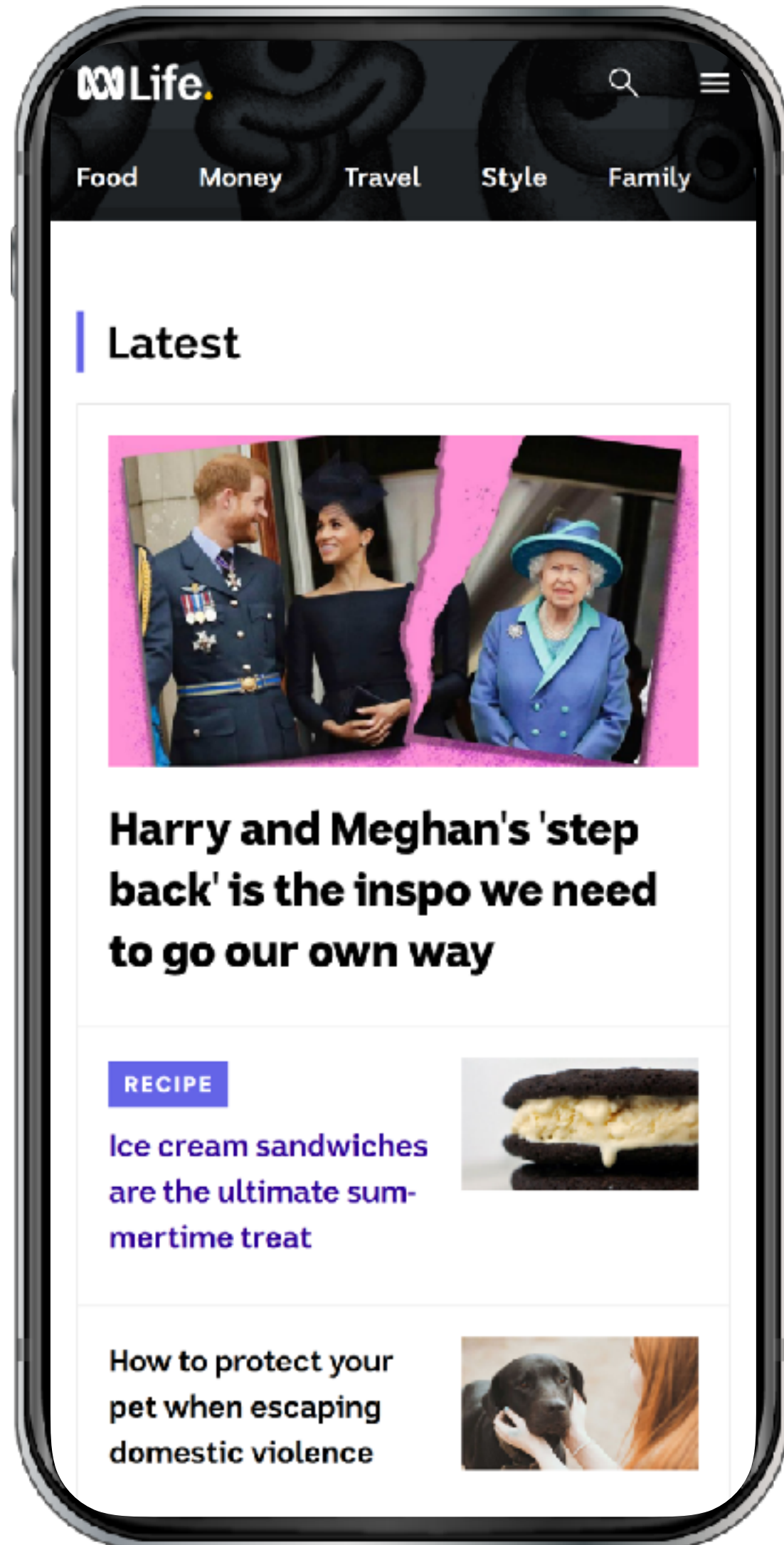


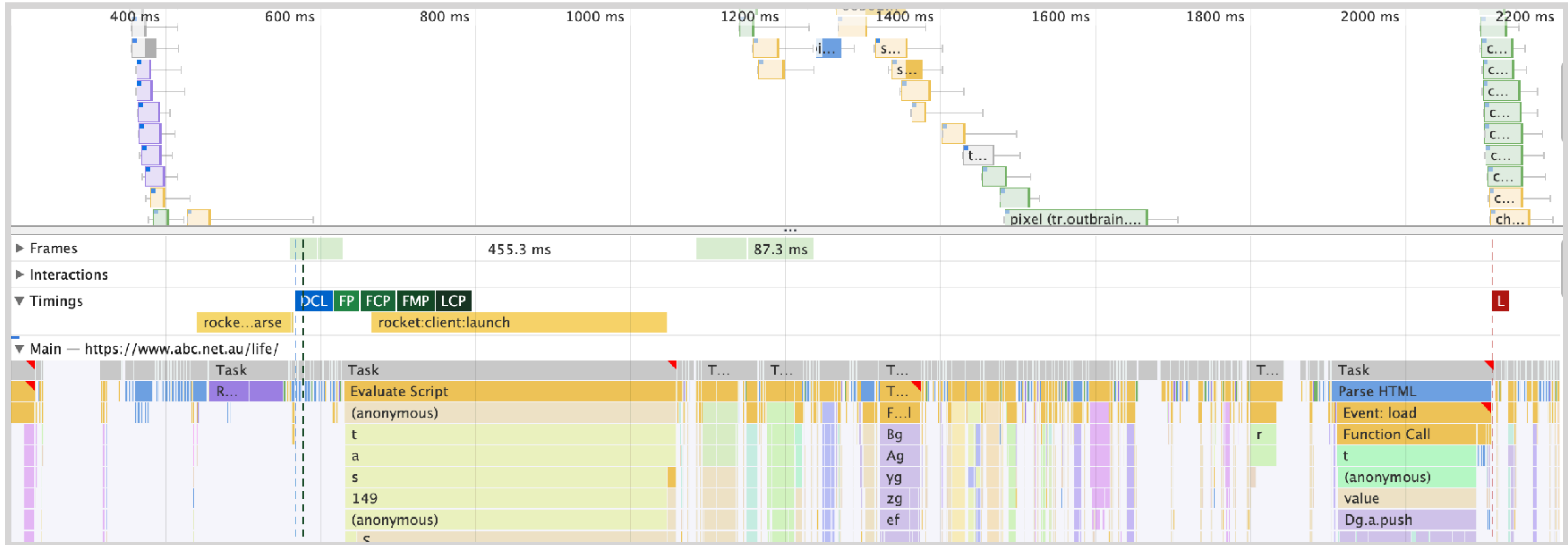
# Lighthouse performance

---



Choose metrics that  
fit your needs





Not React

(mostly)

React

Not React

(mostly)

# Our page load model

---



## Phase 1

Styling complete  
Layout complete  
Images visible  
Not interactive



## Phase 2



## Phase 3

# Our page load model

---



## Phase 1

Styling complete  
Layout complete  
Images visible  
Not interactive



## Phase 2

React hydration  
Page is interactive



## Phase 3

# Our page load model

---

## Phase 1

Styling complete  
Layout complete  
Images visible  
Not interactive

## Phase 2

React hydration  
Page is interactive

## Phase 3

Non-critical JS  
Analytics  
Ads



# Understanding what's happening

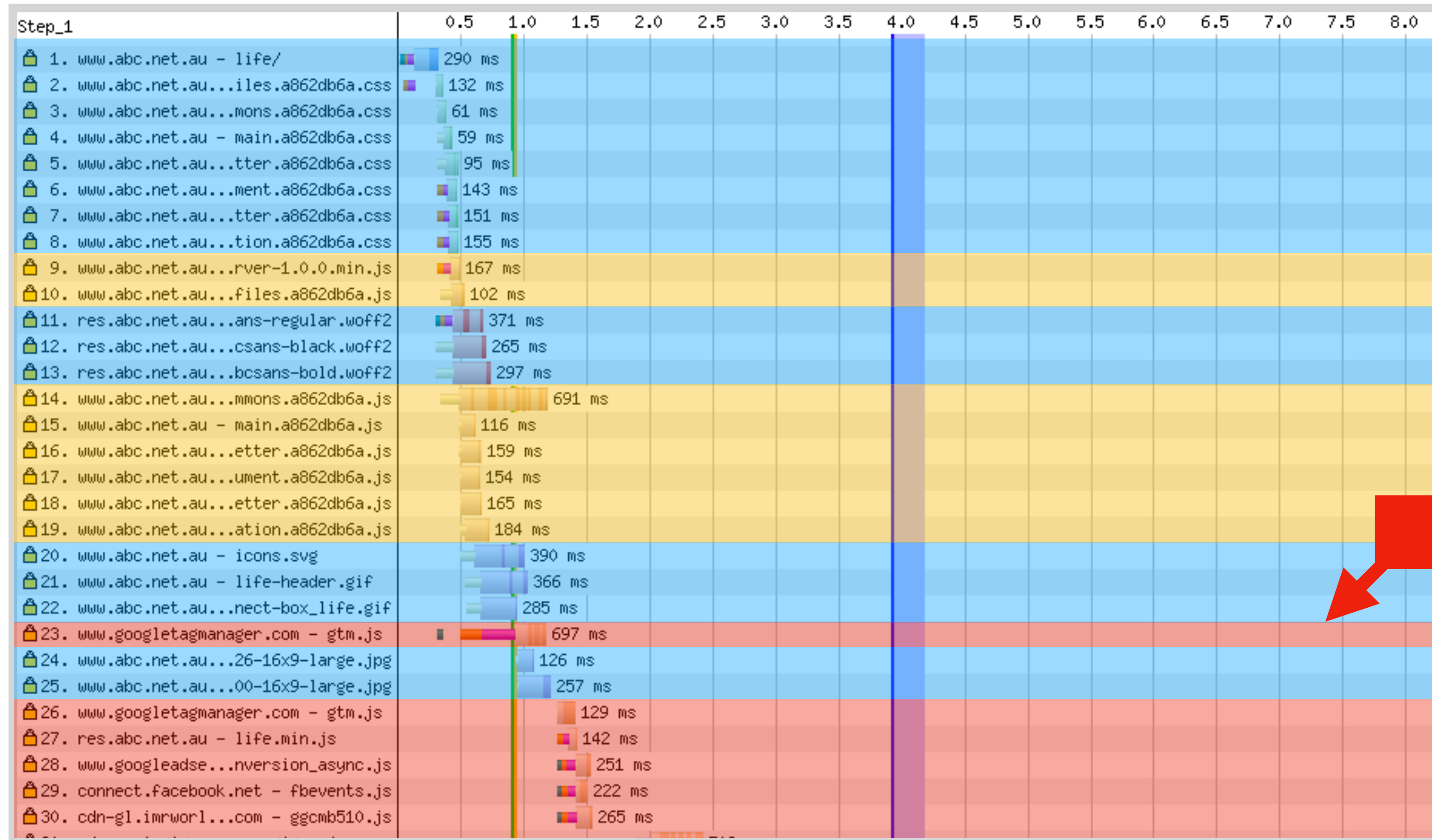


# Understanding what's happening

 Phase 1

 Phase 2

 Phase 3



Whoops!



MEASURE AND IMPROVE

# Improving performance

---



## Phase 1

Styling complete

Layout complete

Images visible

Not interactive



## Phase 2

React hydration

Page is interactive



## Phase 3

Non-critical JS

Analytics

Ads

Fix our cache headers

# Font swapping

# Font swapping

---

First Meaningful Paint

↓ 45%

4.0 sec → 2.2 sec



# Preloading resources

# Preloading resources

---

```
<link rel="preload" href="/abc-regular.woff2" as="font" type="font/woff2" />  
<link rel="preload" href="/abc-bold.woff2" as="font" type="font/woff2" />  
<link rel="preload" href="/abc-black.woff2" as="font" type="font/woff2" />
```

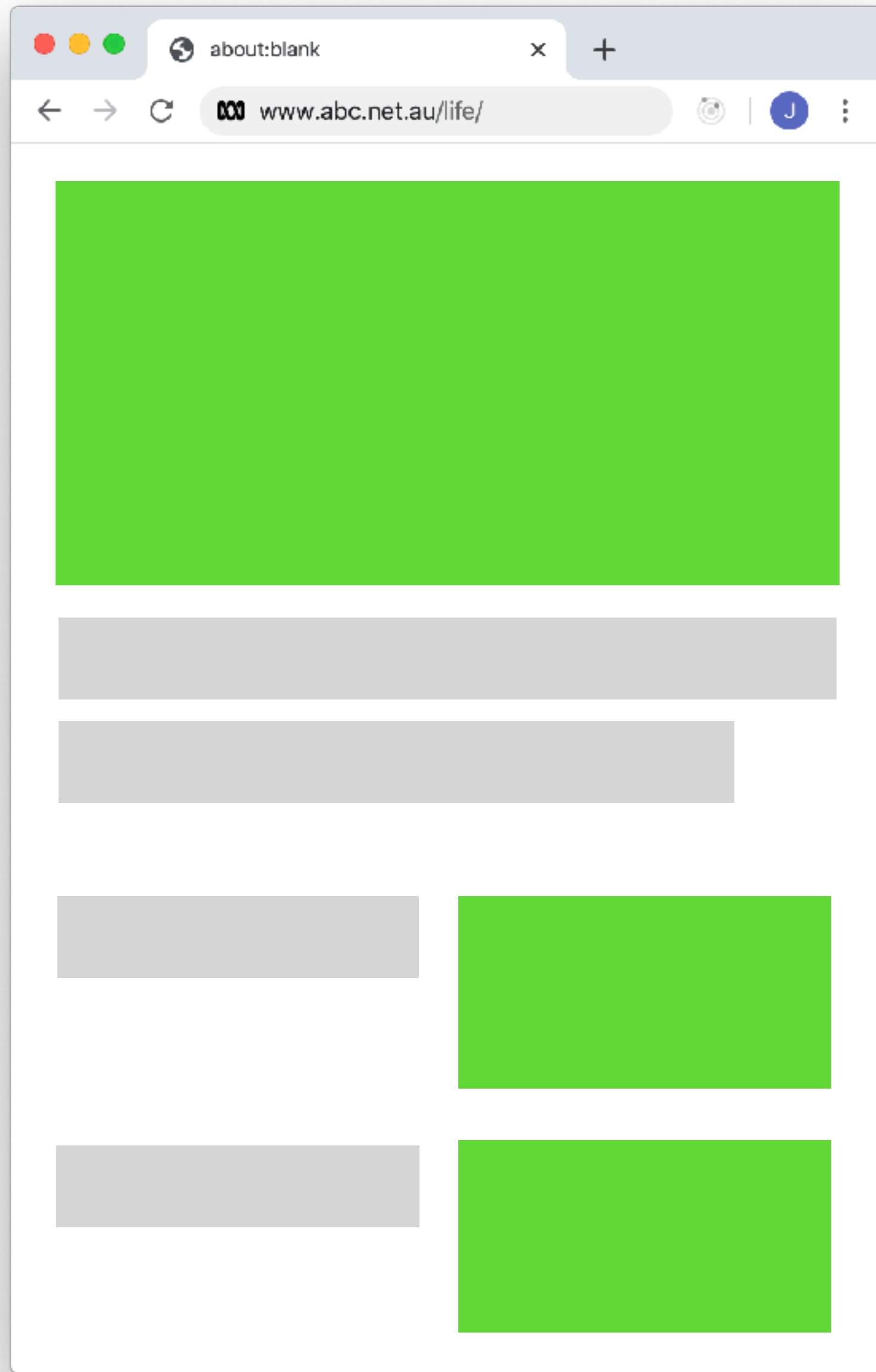
# Lazy loading images

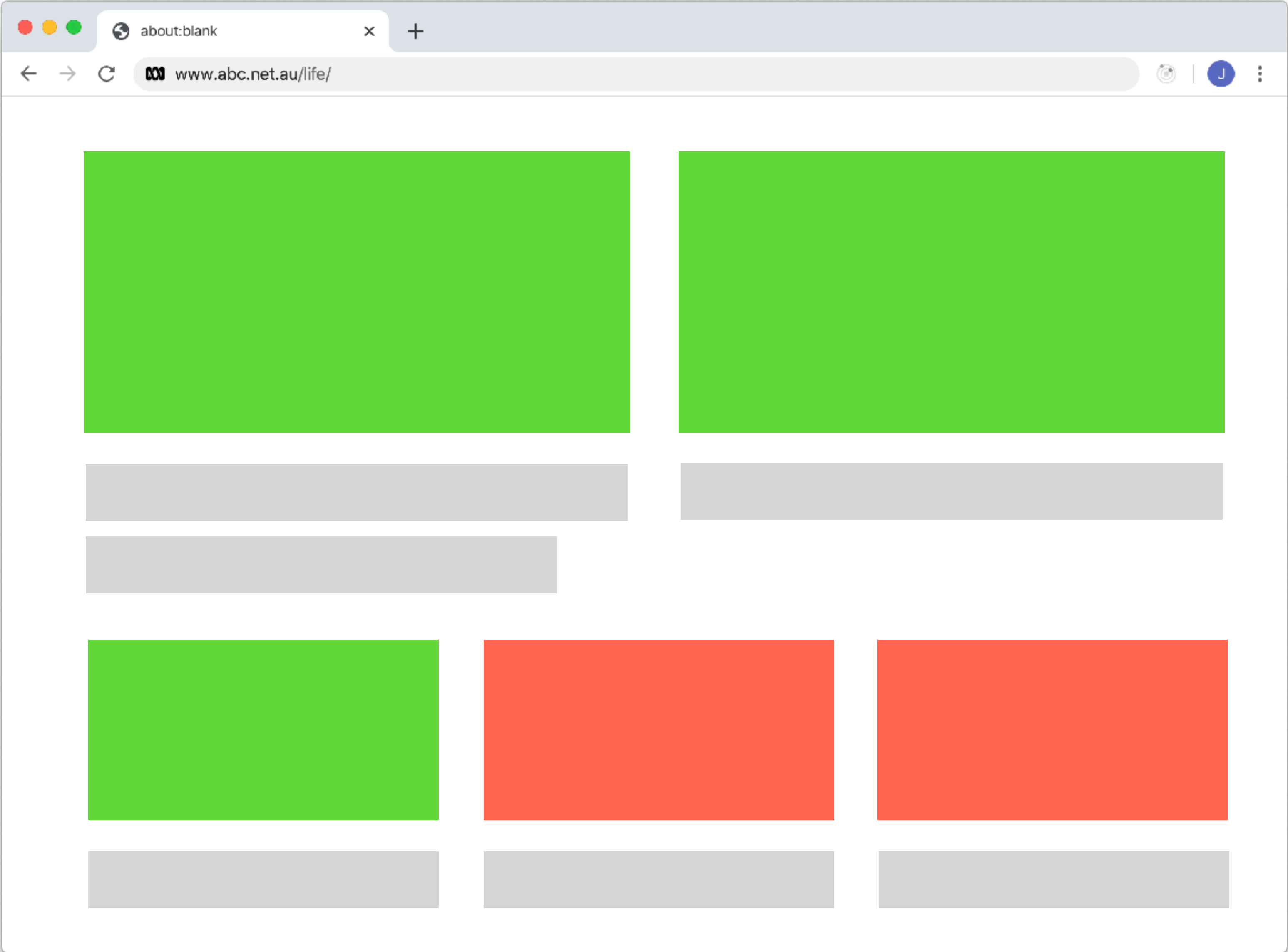
# Intersection observer

---

```
function Image(props) {
  const ref = React.createRef();
  const visible = useIntersectionObserver(ref);

  return (
    <img
      ref={ref}
      src={visible ? props.src : undefined}
    />
  );
}
```





# Bootstrapping images

---

```
<script data-react-helmet="true" type="text/javascript" nonce src="/core/assets/abc-polyfill-observer-1.0.0.min.js"></script>
▼<script data-react-helmet="true" type="text/javascript">
  !function(){var t;t=function(){for(var t=document.querySelectorAll("img"),e=[],n=0;n<t.length;n++){var a=t[n],o=window.scrollY||window.pageYOffset||0;a.parentElement&&a.parentElement.getBoundingClientRect().top<window.innerHeight+o&&e.push(a)}for(n=0;n<e.length;n++){(a=e[n]).hasAttribute("data-nojs")&&a.removeAttribute("data-nojs");for(n=0;n<e.length;n++){var r=(a=e[n]).dataset.src,s=a.dataset.srcset,d=a.dataset.sizes;r&&(a.src=r),d&&s&&(a.srcset=s,a.sizes=d)}},"loading"!==document.readyState?t():document.addEventListener("DOMContentLoaded",t)}();
</script>
<script id="LOADABLE_REQUIRED_CHUNKS" type="application/javascript">
```

## First Image

↓ 44%

7.8 sec → 4.3 sec



# Native support...?

---

```

```

# Native support...?

## Lazy loading via attribute for images & iframes UNOFF

Usage % of all users ?  
Global 62.68%

The `loading` attributing on images & iframes gives authors control over when the browser should start loading the resource.

Current aligned Usage relative Date relative Apply filters Show all ?

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baid Brow:
			4-74												
			75												
6-10	12-17	2-70	76-77	3.1-12.1	10-63	3.2-13.1		2.1-4.4.4	12-12.1				4-9.2		
11	18	71	78	13	64	13.2	all	76	46	78	68	12.12	10.1	1.2	7.1
	76	72-73	79-81	TP		13.3									



## Phase 1

Styling complete

Layout complete

Images visible

Not interactive



## Phase 2

React hydration

Page is interactive



## Phase 3

Non-critical JS

Analytics

Ads

# No module hack for polyfills

```
<script src="/polyfill.min.js" nomodule></script>
```

# No module hack

---

First Contentful Paint

↓ 22%

1.8s → 1.4s  
-20kb gzipped

# Avoid transpiling for modern browsers

---

JavaScript Bytes

↓ **7 to 22%**

Estimated savings

# Dynamic imports and bundle splitting

# Dynamic imports

---

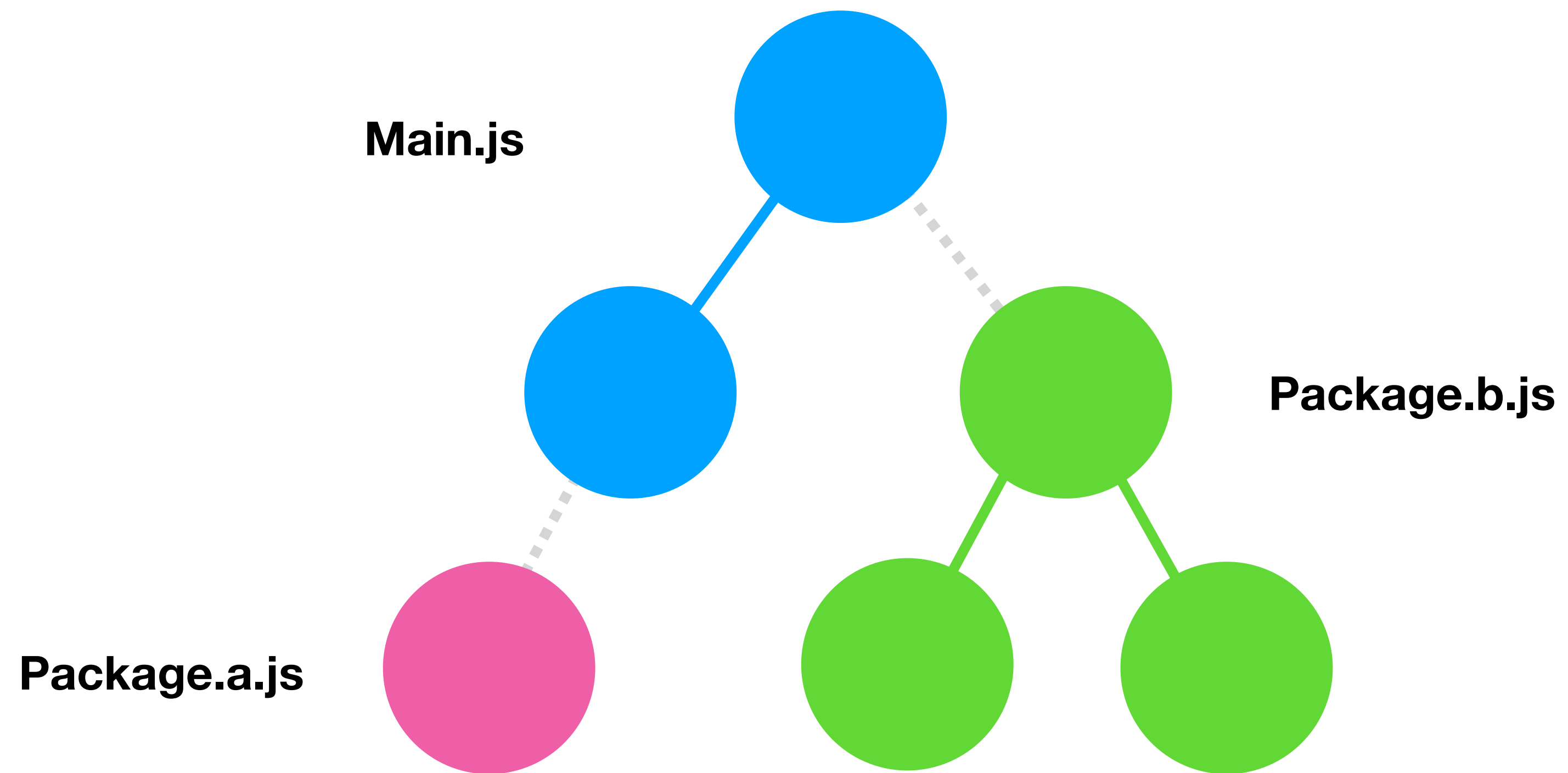
```
// Before
import sum from './sum';
console.log(sum(1, 2));

// After
import('./sum').then(sum => {
  console.log(sum(1, 2));
});
```

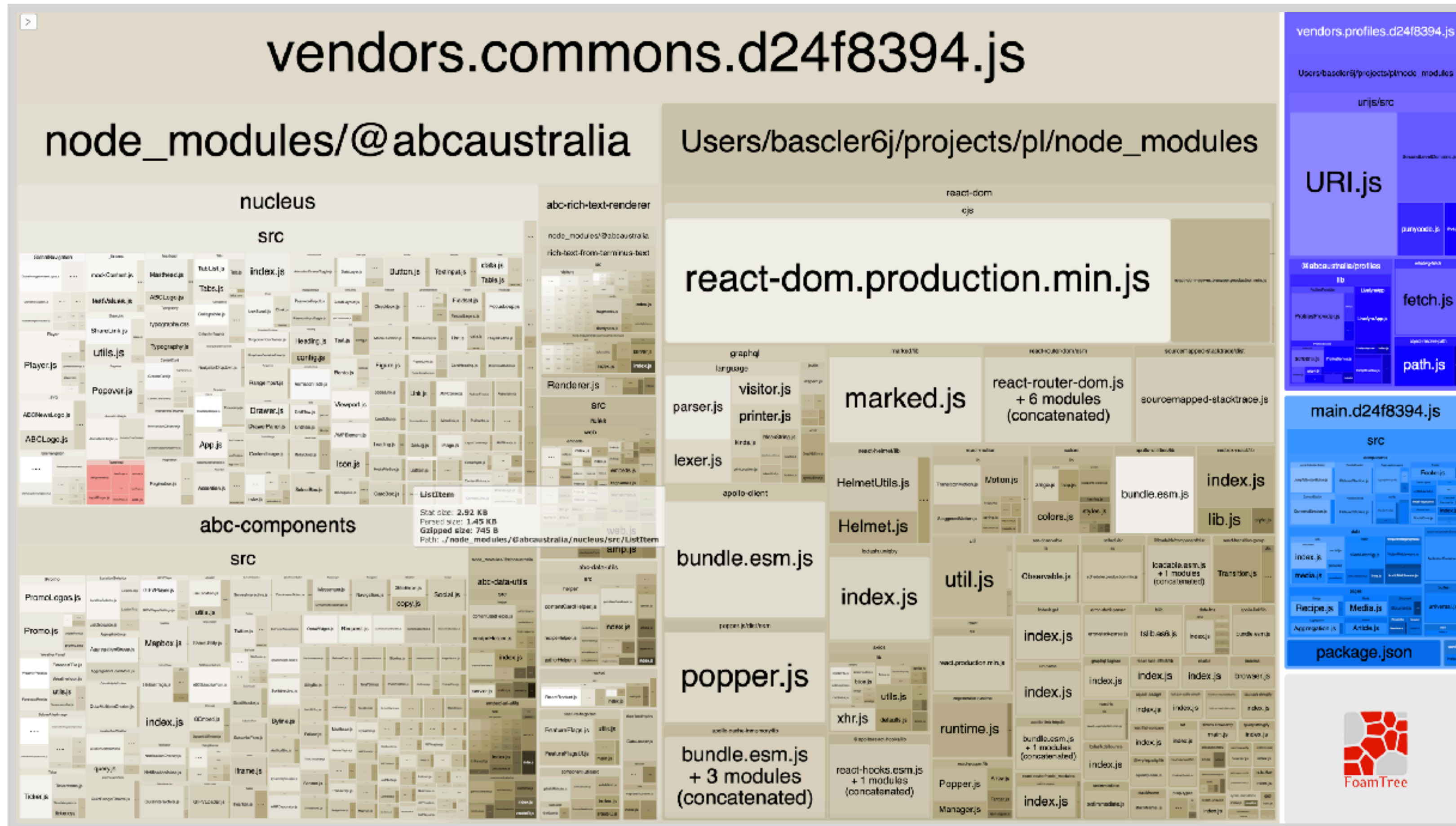


# Dynamic imports

---



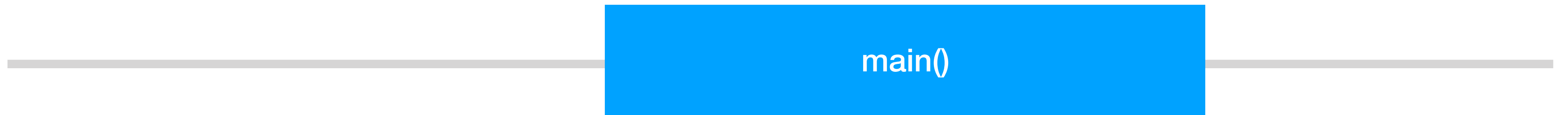
# Webpack analyse

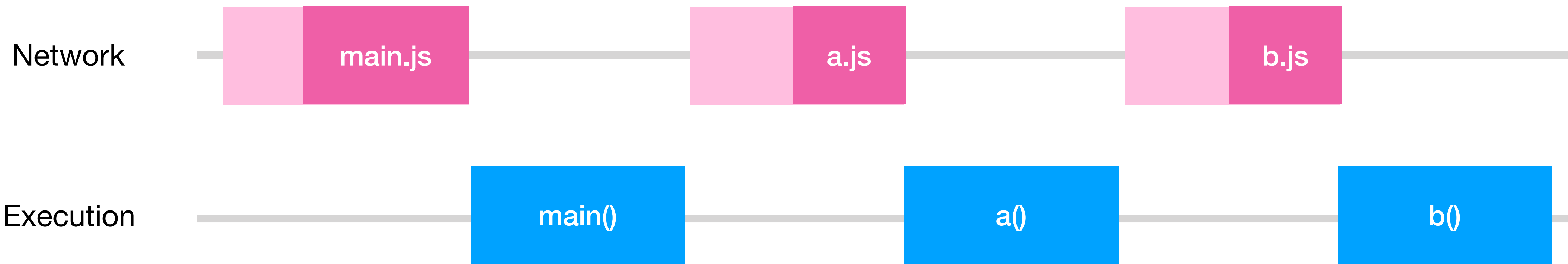


Network



Execution

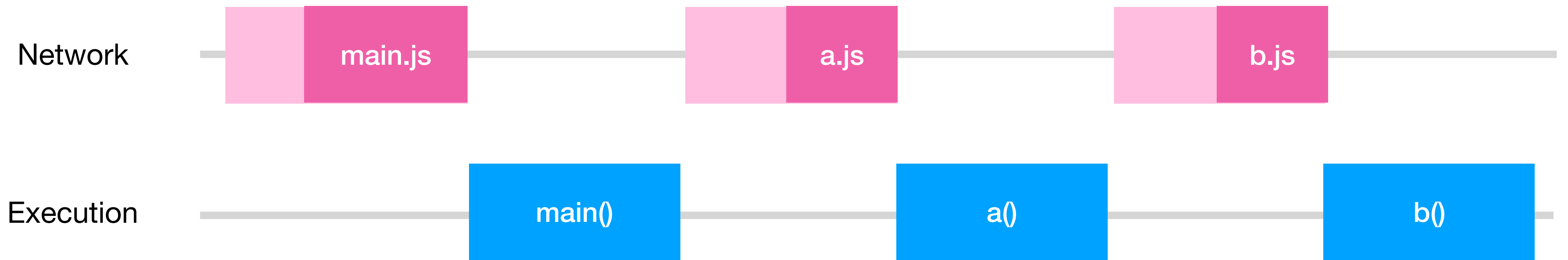


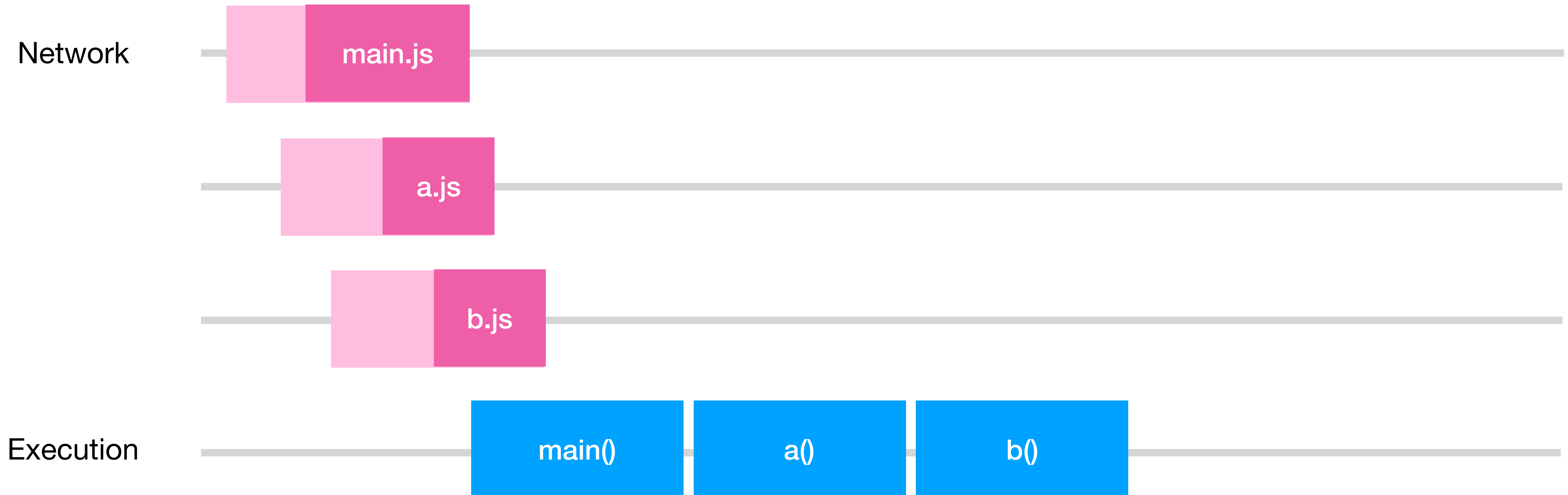




**loadable  
components**

1. Make dynamic `import()` work on server render
2. Generate a list of packages to preload on the client

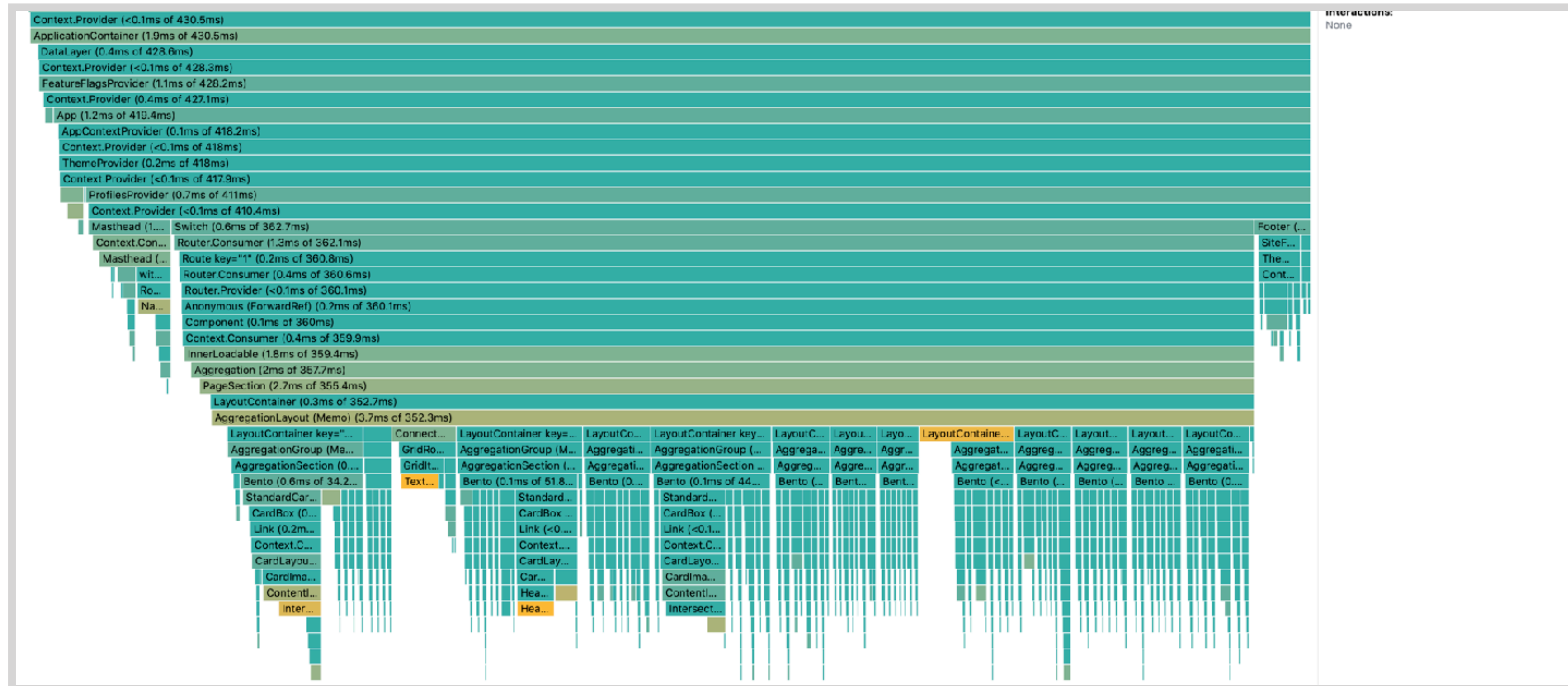




# React Profiling



# React profiling





## Phase 1

Styling complete  
Layout complete  
Images visible  
Not interactive



## Phase 2

React hydration  
Page is interactive



## Phase 3

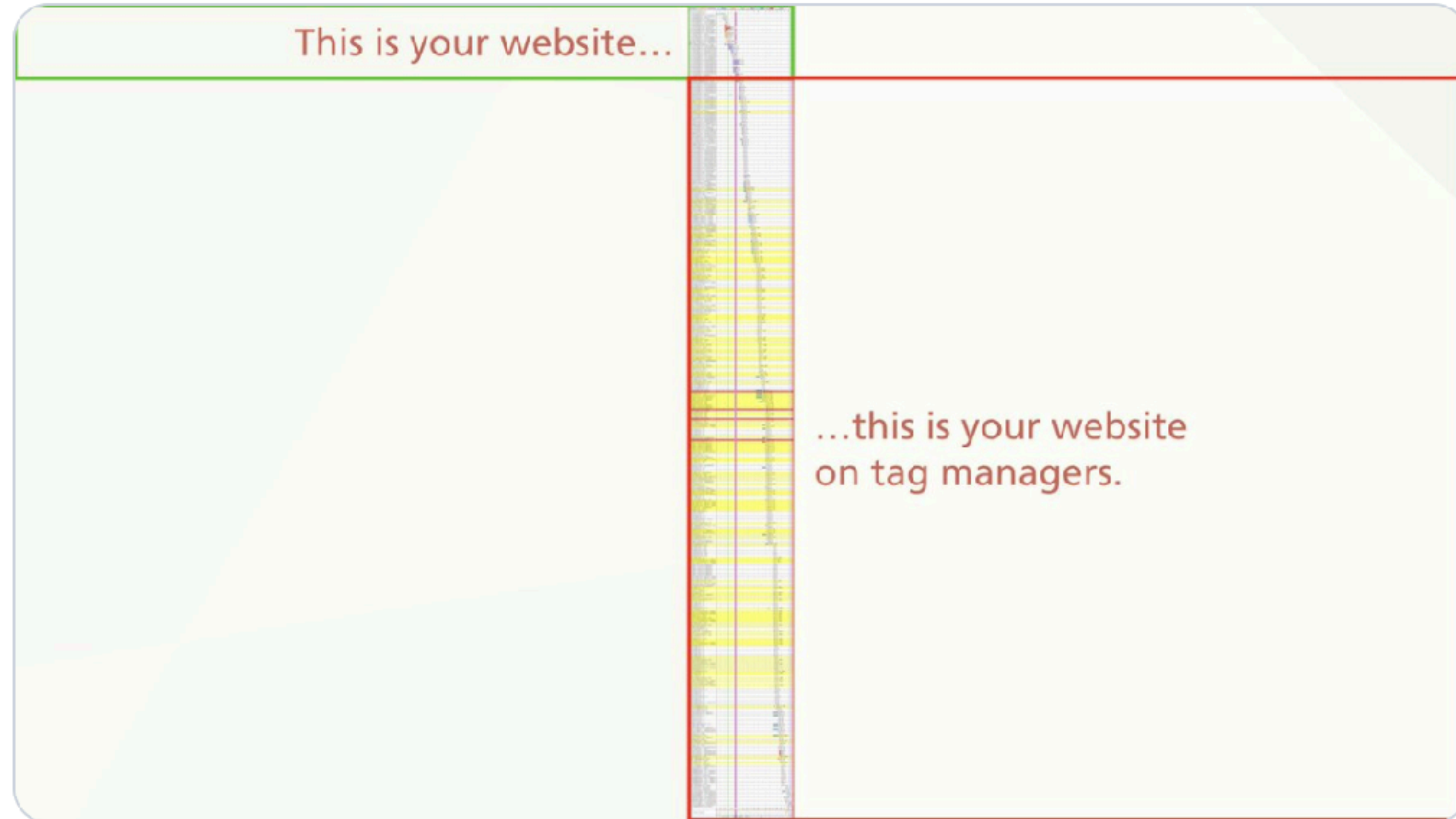
Non-critical JS  
Analytics  
Ads



**Harry Roberts** ✓  
@csswizardry

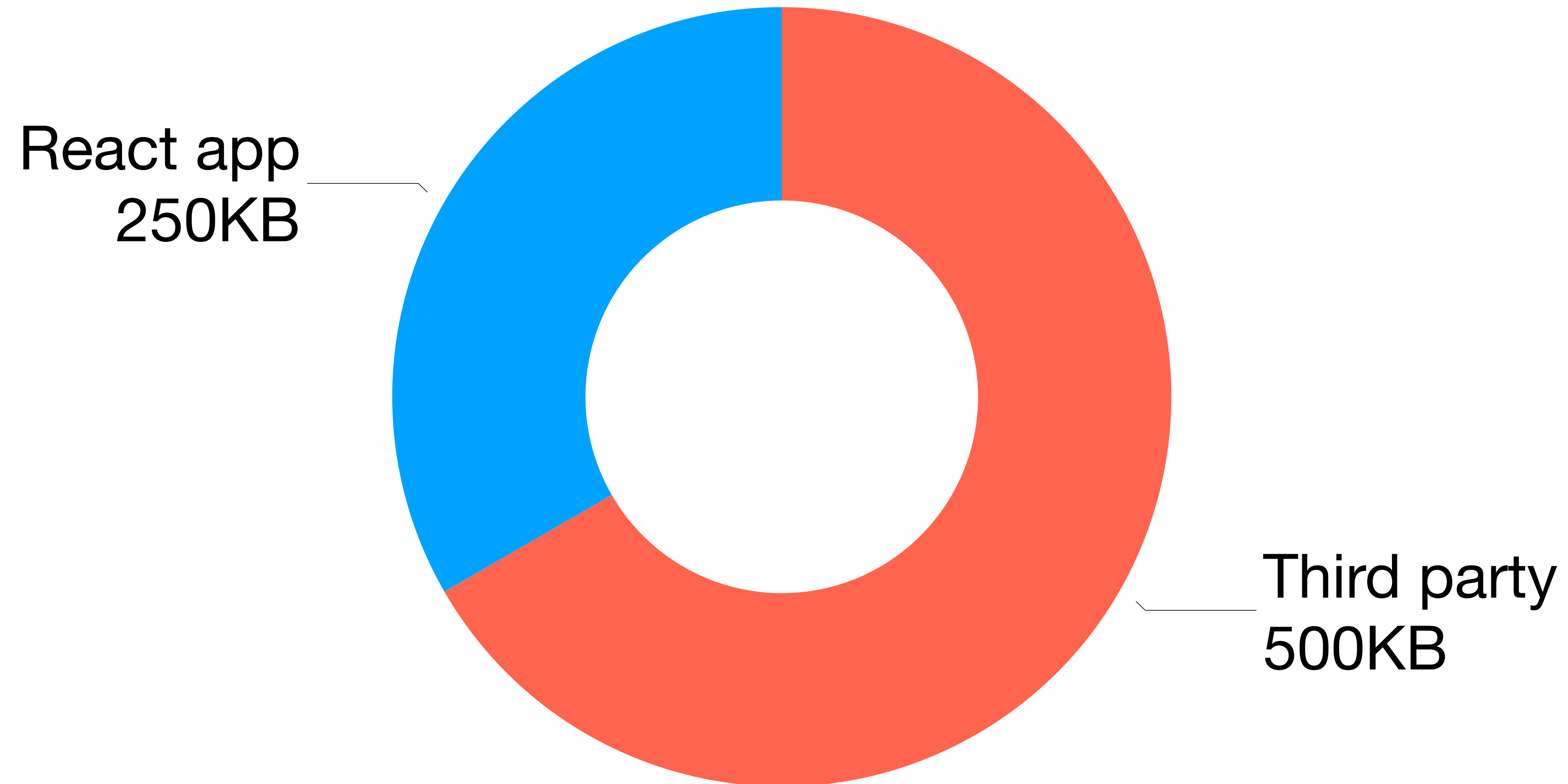


This is your website. This is your website on tag managers.



# JS breakdown

---



# Lite YouTube Embed

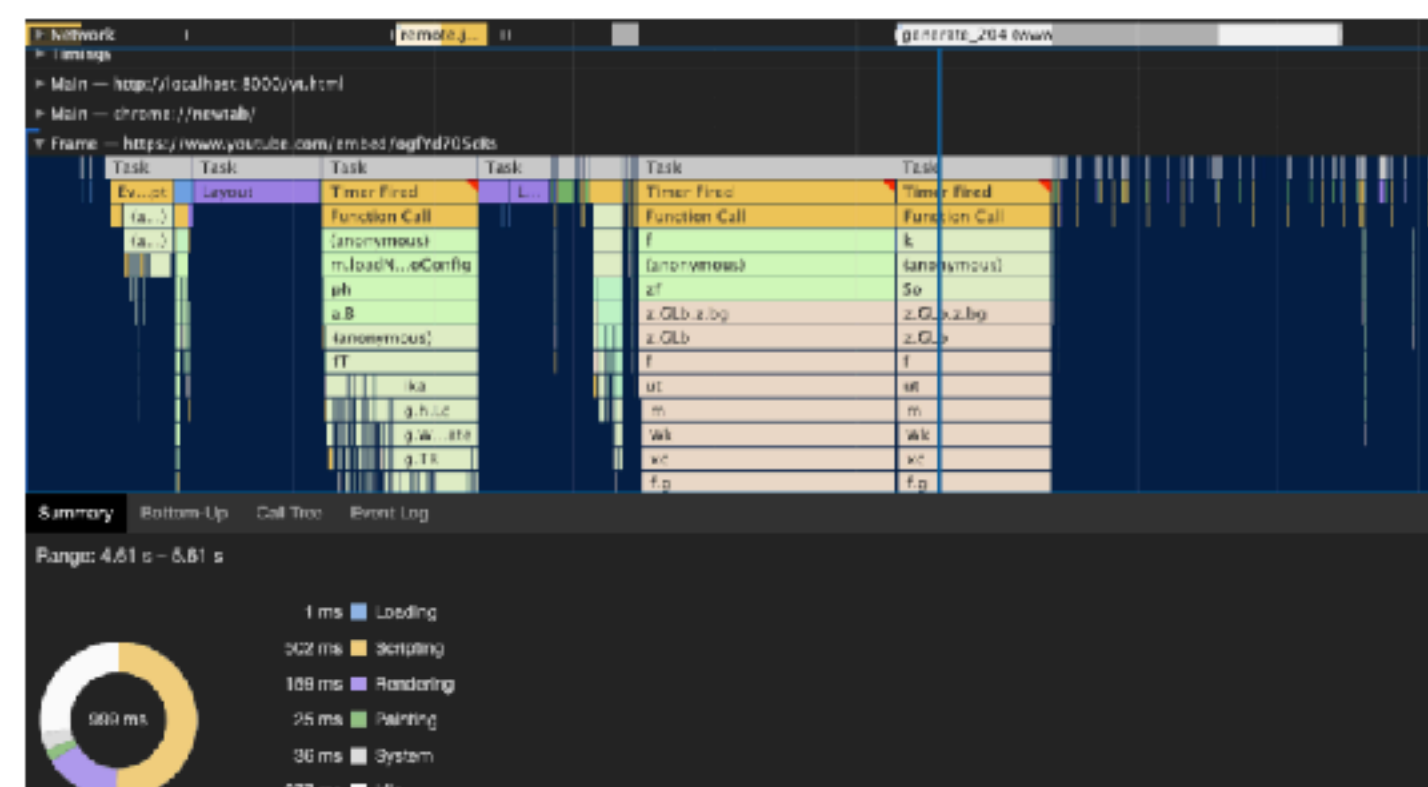
## Lite YouTube Embed

Renders faster than a sneeze.

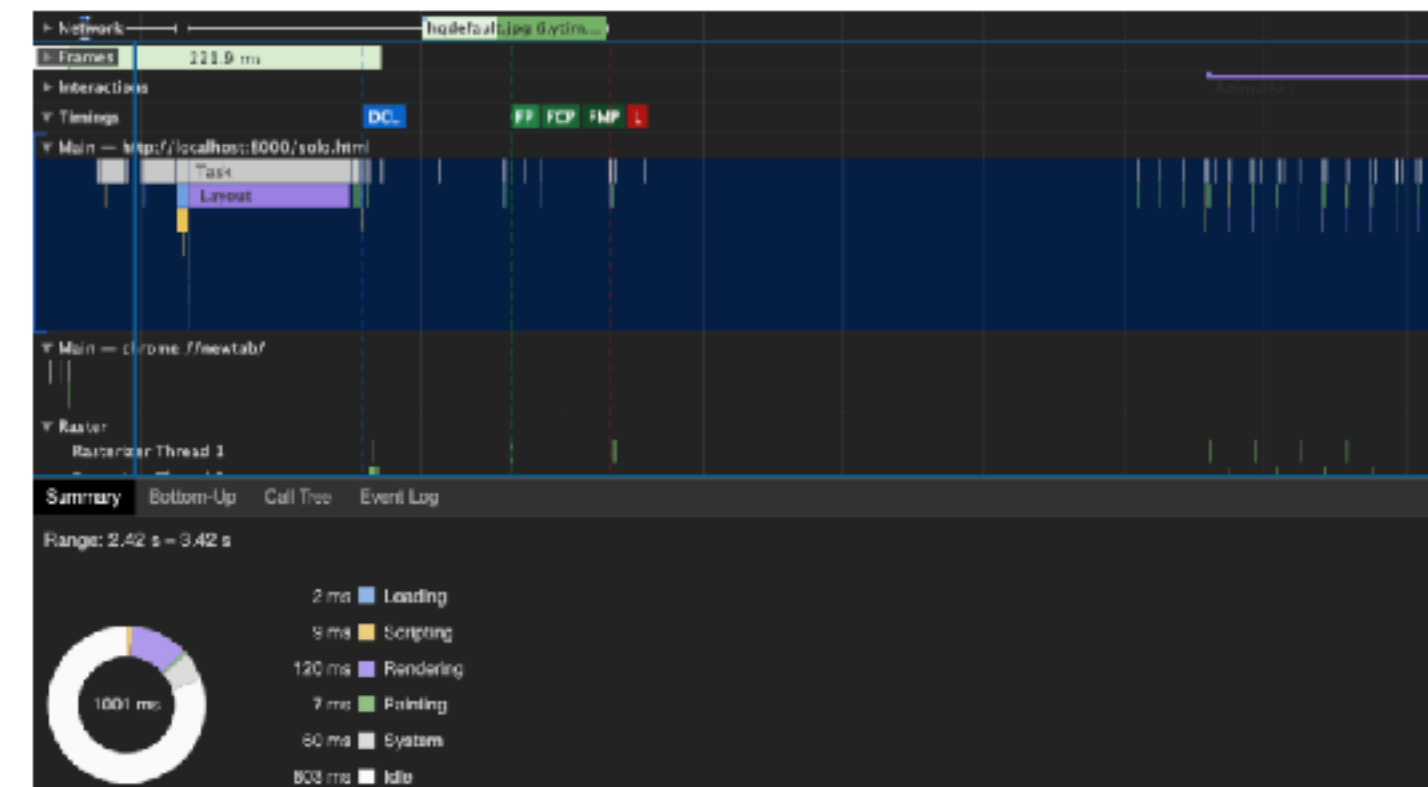
Provide videos with a supercharged focus on visual performance. This custom element renders just like the real thing but approximately 224X faster.

## Comparison

Normal `<iframe>` YouTube embed



lite-youtube



# Lighthouse third party report

● Third-Party Usage — 9 Third-Parties Found ^

Third-party code can significantly impact load performance. Limit the number of redundant third-party providers and try to load third-party code after your page has primarily finished loading. [Learn more](#)

Third-Party	Size	Main Thread Time
<a href="#">Google Tag Manager</a>	72 KB	697 ms
Gigya	105 KB	467 ms
<a href="#">Google/DoubleClick Ads</a>	13 KB	540 ms
<a href="#">Nielsen NetRatings SiteCensus</a>	82 KB	409 ms
<a href="#">Facebook</a>	142 KB	181 ms
Chartbeat	32 KB	132 ms
<a href="#">Google Analytics</a>	18 KB	112 ms

# Scorecard

---

## First Contentful Paint

↓ **54%**

MotoG4 4.3 sec → 2.0 sec

↓ **30%**

Desktop 0.56 sec → 0.39 sec

# Scorecard

---

Visually Complete

↓ **25%**

MotoG4 16.7 sec → 12.6 sec

↓ **32%**

Desktop 1.7 sec → 1.2 sec



# Scorecard

---

## Parse and Compile

↓ **37%**

MotoG4 6.4 sec → 4.0 sec

↓ **42%**

Desktop 1.2 sec → 0.7 sec

# Scorecard

---

## Blocking Execution

↓ 45%

MotoG4 4.9 sec → 2.7 sec

↓ 55%

Desktop 0.79 sec → 0.36 sec

What did we learn?

---

Lighthouse audit is not a  
silver bullet

**Opportunities** — These suggestions can help your page load faster. They don't [directly affect](#) the Performance score.

Opportunity	Estimated Savings
▲ Defer offscreen images	 2.25 s ▼
▲ Eliminate render-blocking resources	 1.11 s ▼
■ Avoid multiple page redirects	 0.78 s ▼
■ Preconnect to required origins	 0.31 s ▼

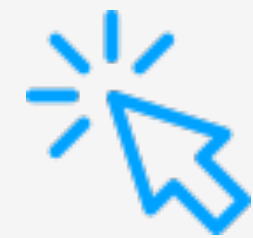
**Diagnostics** — More information about the performance of your application. These numbers don't [directly affect](#) the Performance score.

- ▲ Serve static assets with an efficient cache policy — 93 resources found ▼
- ▲ Avoid an excessive DOM size — 2,477 elements ▼
- ▲ Minimize main-thread work — 5.6 s ▼



## Phase 1

Styling complete  
Layout complete  
Images visible  
Not interactive



## Phase 2

React hydration  
Page is interactive



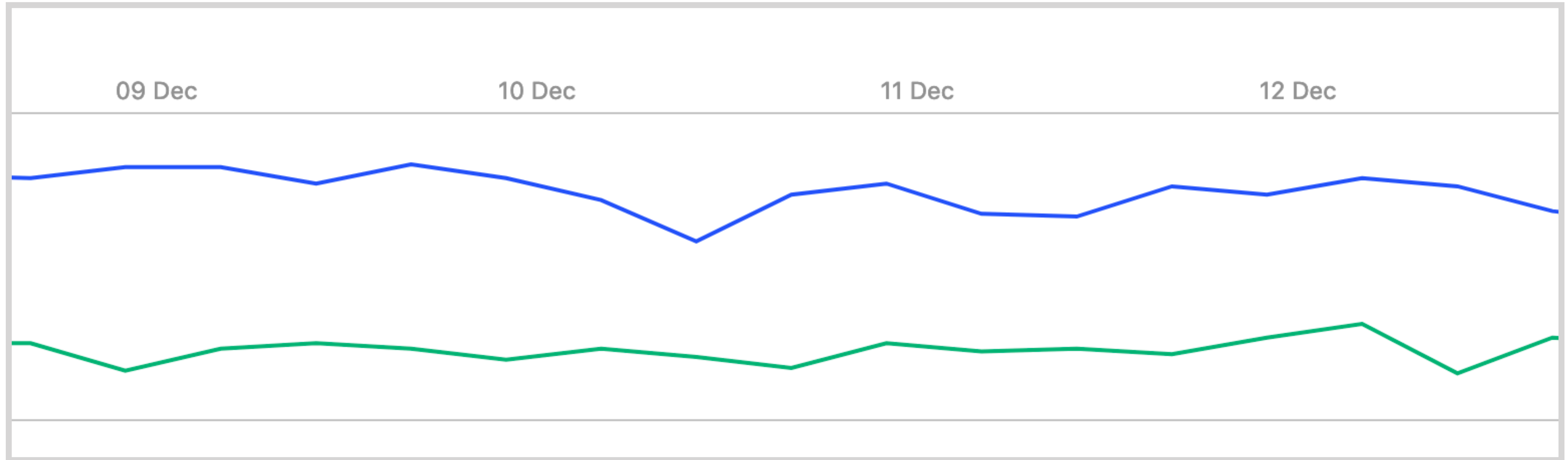
## Phase 3

Non-critical JS  
Analytics  
Ads

Measuring impact of  
changes is hard

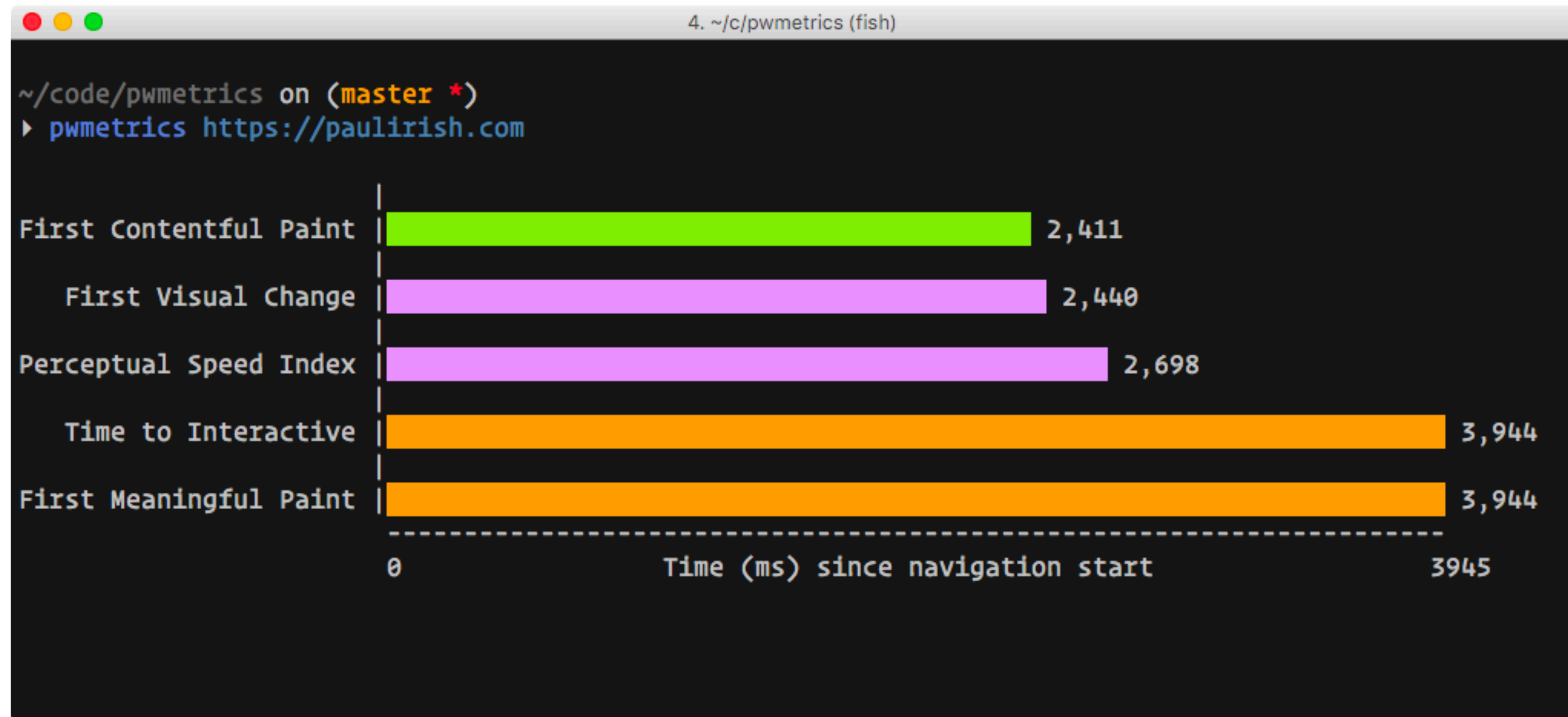
# Lighthouse is super noisy

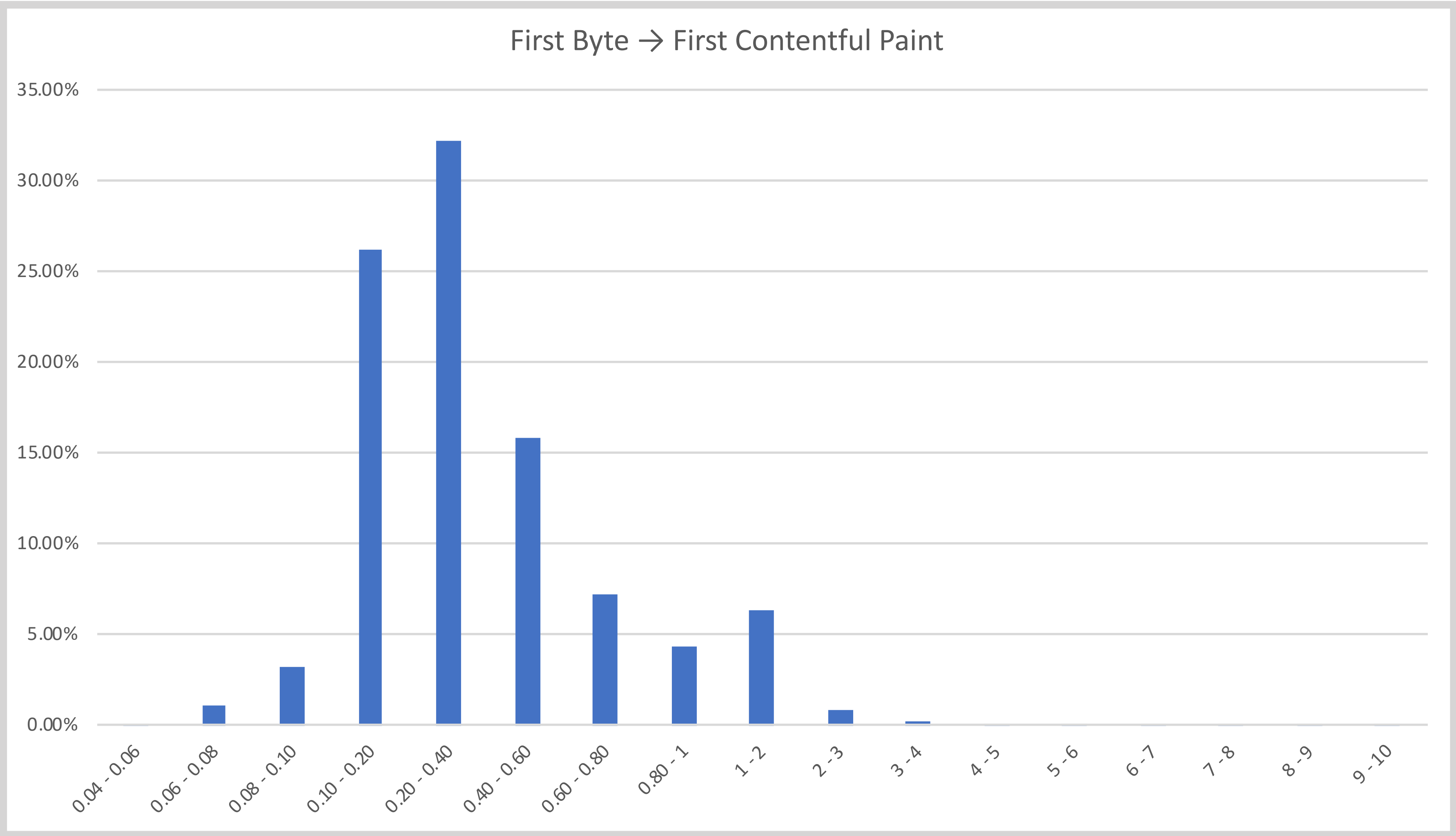
---





# PWMetrics





# Performance API

---

```
// Introspection
performance.getEntriesByType("mark");
performance.getEntriesByType("resource");

entry.transferSize;
entry.startTime;
entry.duration;

// Reporting
performance.mark("render-start");
```

# Record your wins and losses

I always forget this

# Calibre

6 months ▾

● Chrome Desktop ● MotoG4, 3G connection ● iPhone6 - Good 3G

Jul 2019

Aug 2019

Sep 2019

Oct 2019

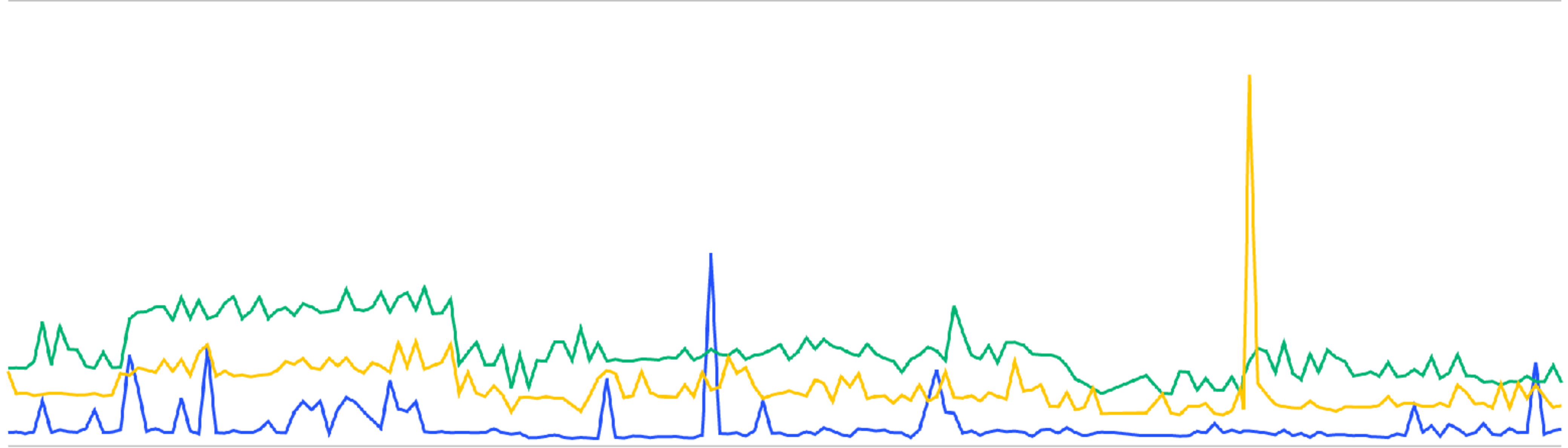
Nov 2019

Dec 2019

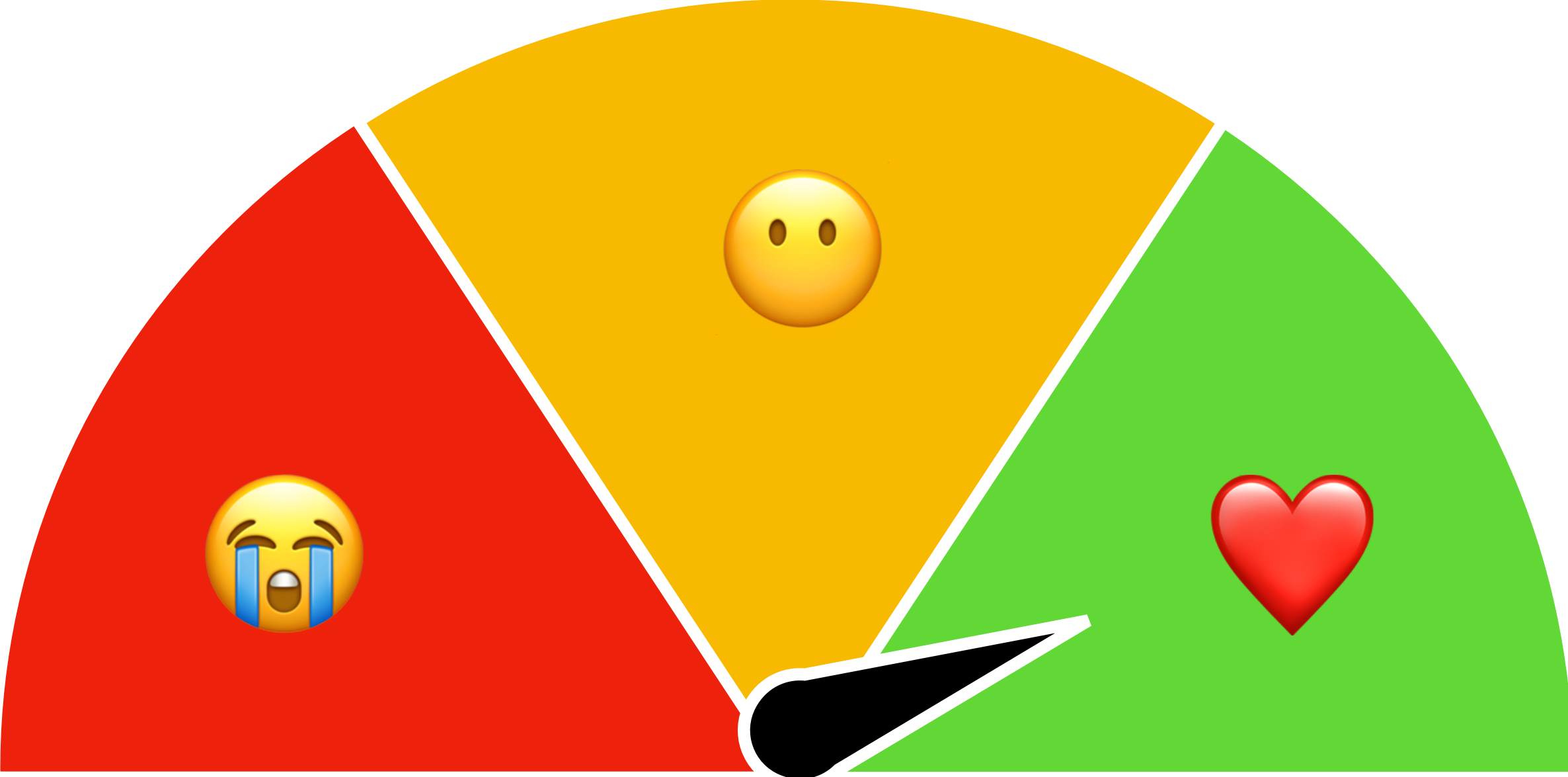
10.00 sec

5.00 sec

0 ms



React complements  
the platform



THE FIRST TWO SECONDS

# FASTER PAGE LOADS FOR REACT

[joshduck.com](https://joshduck.com) or @joshduck for slides.