TTI

Time To Interactive

The time at which a page becomes interactive (events wired up, etc).

Total Blocking Time (TBT)

Nov 7, 2019 • Updated May 4, 2020

Appears in: Metrics

Philip Walton
Twitter · GitHub · Blog

★ Total Blocking Time (TBT) is an important, user-centric metric for measuring load responsiveness because it helps quantify the severity of how non-interactive a page is prior to it becoming reliably interactive—a low TBT helps ensure that the page is usable.

What is TBT?

https://web.dev/tbt/

Login | Register

WEBPAGETEST

HOME | TEST RESULT | TEST HISTORY | FORUMS | DOCUMENTATION | ABOUT

Web Page Performance Test for

https://cnn.com

From: Dulles, VA - **Moto G4 - Chrome - 3GFast**
3/1/2020, 2:27:41 PM

Need

F	A	A	A	F
First Byte Time	Keep-alive Enabled	Compress Transfer	Compress Images	Cache static content

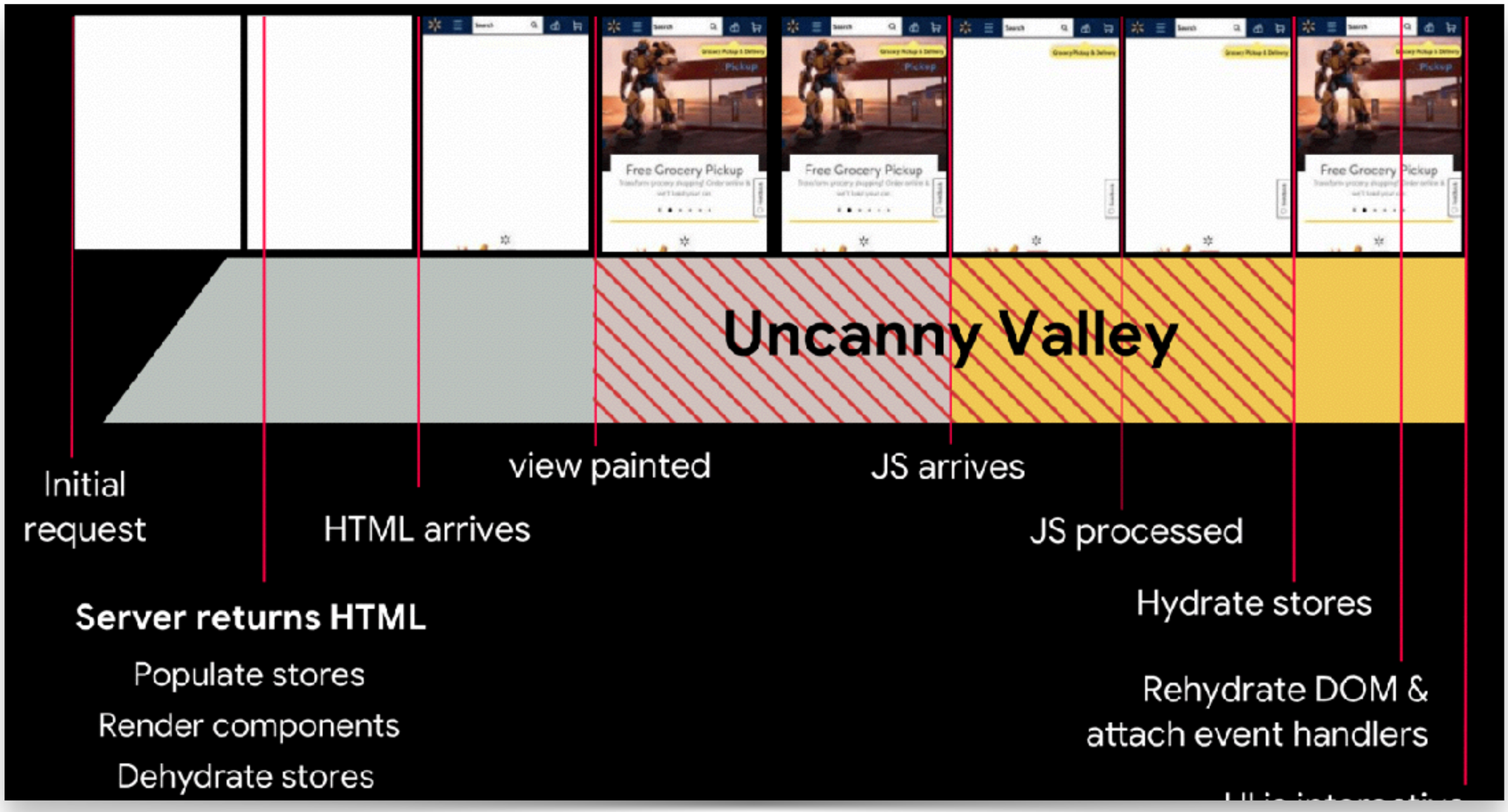
Summary | Details | Performance Review | Content Breakdown | Domains | Processing Breakdown | Screenshot | Image Analysis | Re

Tester: MotoG4_05-192.168.1.105
Test runs: **9**

Export HTT

	First Byte	Start Render	First Contentful Paint	Speed Index	Last Painted Hero	Result (error code)	Web Vitals: Largest Contentful Paint	Web Vitals: Cumulative Layout Shift	Web Vitals: Total Blocking Time	Document Complete: Time	Document Complete: Requests	Document Complete: Bytes I
First View (Run 1)	3.685s	20.300s	20.294s	45.775s	83.013s	0	36.159s	0.404	≥ 12.833s	105.739s	624	5,386 K

https://www.webpagetest.org/result/200301_H1_196aafce6ffc7b3695064e8002f837d8/1/details/#waterfall_view_step1

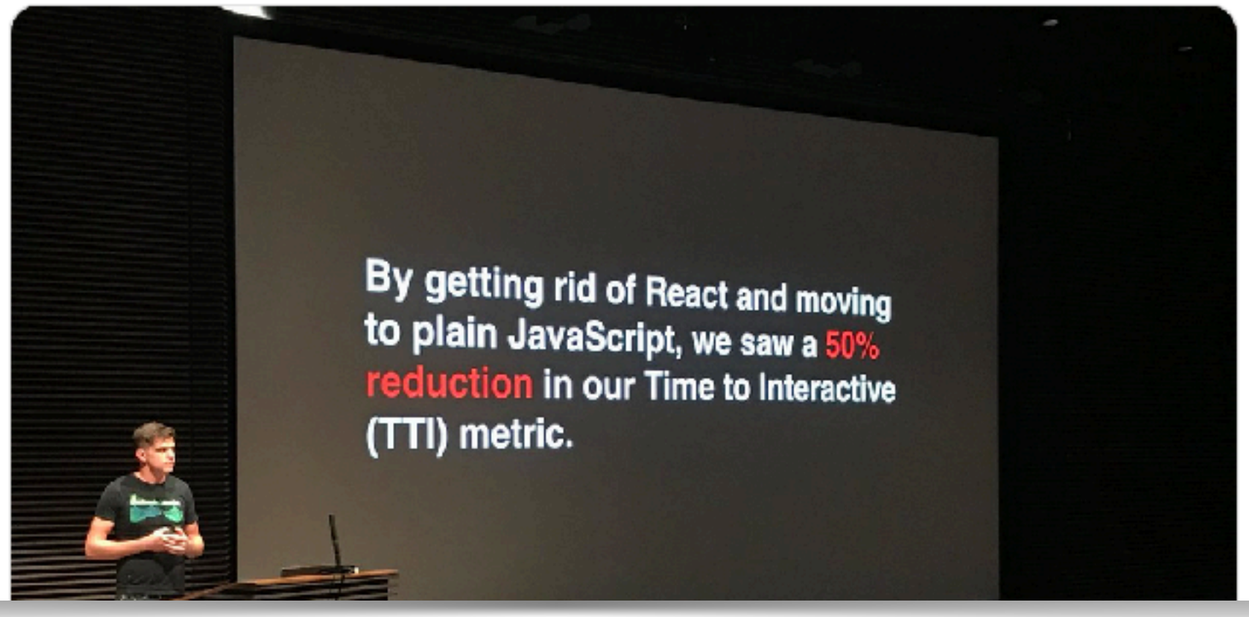


https://addyosmani.com/blog/rehydration/

Netflix UI Engineers
@NetflixUIE

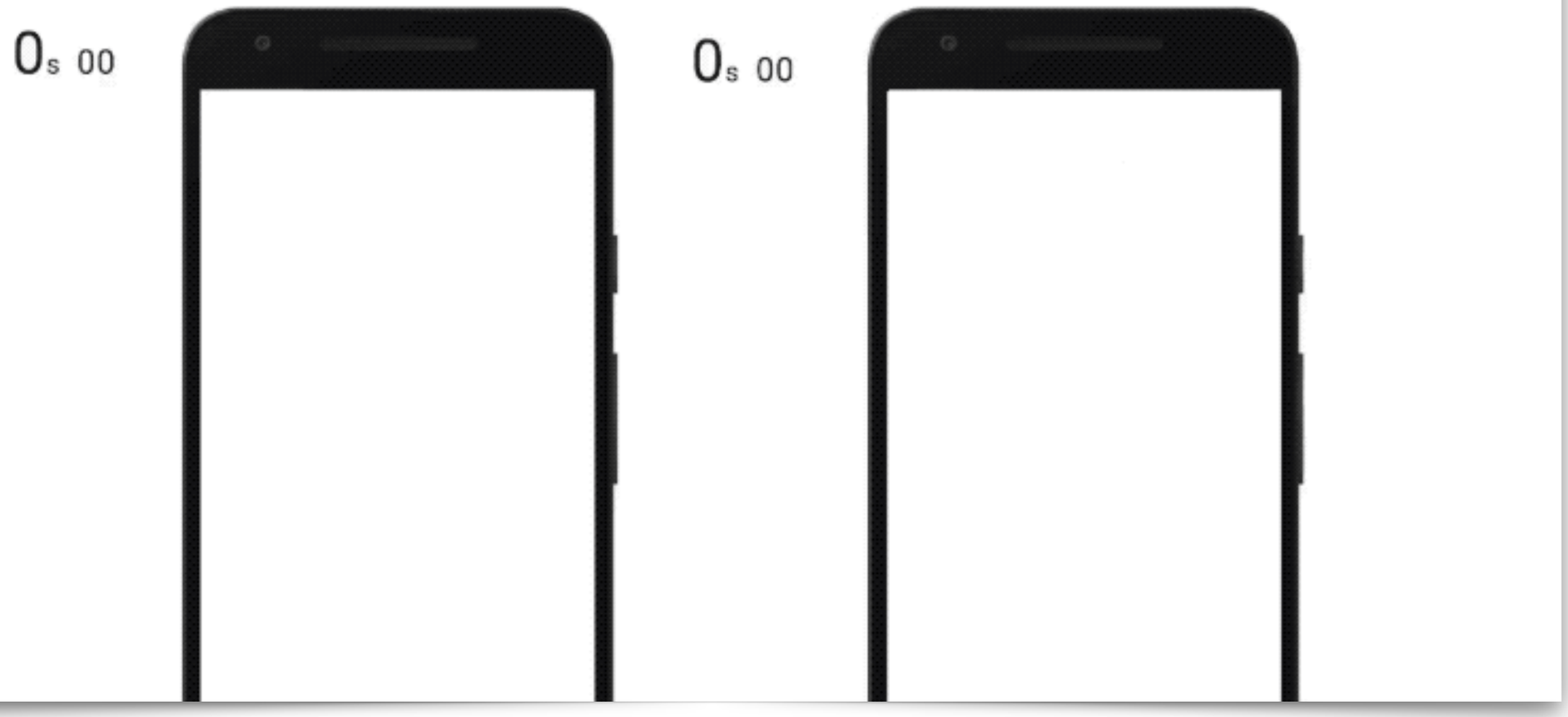
Follow

Removing client-side React.js (but keeping it on the server) resulted in a 50% performance improvement on our landing page



https://twitter.com/NetflixUIE/status/923374215041912833

TIME TO INTERACTIVE



https://medium.com/@addyosmani/the-cost-of-javascript-in-2018-7d8950fbb5d4

“

we feel your baseline should be getting interactive in under 5 seconds on a slow 3G connection on a median mobile

Addy Osmani

https://medium.com/@addyosmani/the-cost-of-javascript-in-2018-7d8950fbb5d4

The Vanilla JS Toolkit

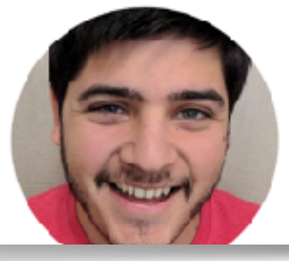
Methods & APIs Helpers Plugins Boilerplates Polyfills Learning

The Vanilla JavaScript Toolkit

A collection of JavaScript methods, helper functions, plugins, boilerplates, polyfills, and learning resources.

***Vanilla JS** is a term for coding with native JavaScript features and browser APIs instead of frameworks and libraries.*

⏰ ***Last chance to join!** A new session of the Vanilla JS Academy starts on Monday. Register today and save 30%. Click here to learn more.*



> *Can you actually create a website without JS frameworks? Yes! Here's a collection of resources to get started.*
>
> *- Ed Rivas*

https://vanillajstoolkit.com/reference/

Reference Guide

A quick reference for commonly used JavaScript methods and browser APIs.

Unless otherwise noted, these work in all modern browsers, and IE9+. You can extend support back further with polyfills.

On this page

Styling a Select Like It's 2019

Posted by **Scott** 12/18/2018

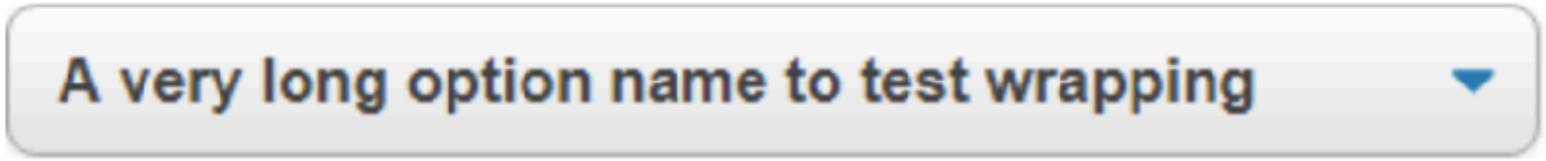
Update 12/19 The select now has totally consistent appearance in Internet Explorer 11 and 10, thanks to **a nice tip** from **Jelmer de Maat**

The `select` element has long been difficult to style consistently across browsers. To avoid its shortcomings in the past, we have used workarounds like styling a parent element, adding pseudo-elements, and even using JavaScript to construct a select-like control out of different elements that are easier to style. But workarounds are hard to maintain and use, not to mention the accessibility challenges that custom elements bring.

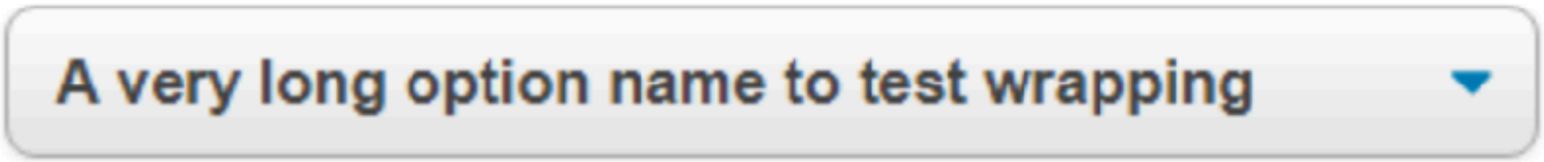
Recently, we'd seen some articles suggest that things haven't changed a great deal with `select`'s styling limitations, but I decided to return to the problem and tinker

https://www.filamentgroup.com/lab/select-css.html

- Firefox



- Chrome



- Safari

https://www.filamentgroup.com/lab/select-css.html

Basic Snapper example

A snapper carousel with some thumbnail links.

Thumbnails are just regular links to a slide's ID attribute. The scrollbar is cropped from sight using the optional `snapper_pane_crop` div (only recommended when thumbnails or next/prev navigation is in play).

rollup.js

Introduction

Overview

Rollup is a module bundler for JavaScript which compiles small pieces of code into something larger and more complex, such as a library or application. It uses the new standardized format for code modules included in the ES6 revision of JavaScript, instead of previous idiosyncratic solutions such as CommonJS and AMD. ES modules let you freely and seamlessly combine the most useful individual functions from your favorite libraries. This will eventually be possible natively everywhere, but Rollup lets you do it today.

Installation

https://rollupjs.org/guide/en/#

Code Splitting

This guide extends the examples provided in Getting Started and Output Management. Please make sure you are at least familiar with the examples provided in them.

Code splitting is one of the most compelling features of webpack. This feature allows you to split your code into various bundles which can then be loaded on demand or in parallel. It can be used to achieve smaller bundles and control resource load prioritization which, if used correctly, can have a major impact on load time.

There are three general approaches to code splitting available:

- Entry Points: Manually split code using `entry` configuration.
- Prevent Duplication: Use the `SplitChunksPlugin` to dedupe and split chunks.
- Dynamic Imports: Split code via inline function calls within modules.

Entry Points

This is by far the easiest and most intuitive way to split code. However, it is more manual and has some

https://webpack.js.org/guides/code-splitting/

Was $60.00 Now $50.00

105 Reviews / Write a Review

3.4 / 5

66% Recommend this item

Overall Fit:

RUNS SMALL | TRUE TO SIZE | RUNS LARGE

Filter By

Body Type | Height

Sort By

Highest Rated

JWall

Atlanta

Overall Fit:

★★★★★

Yes, I recommend this product.

Great fit and good value

June 26, 2018

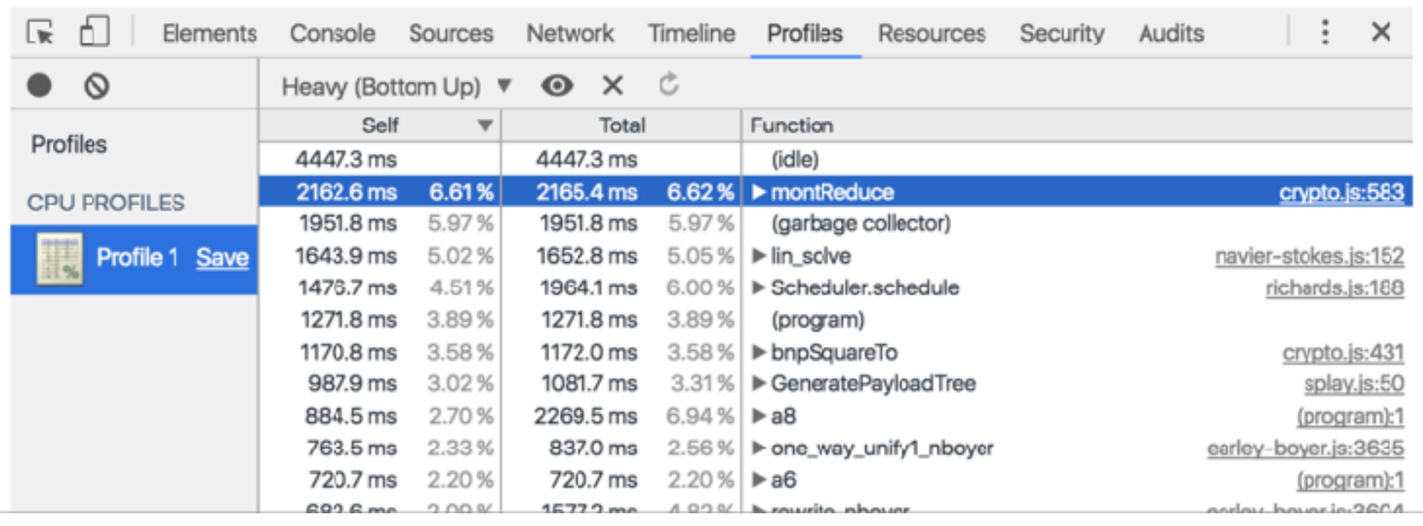
Good T-shirts can be hard to

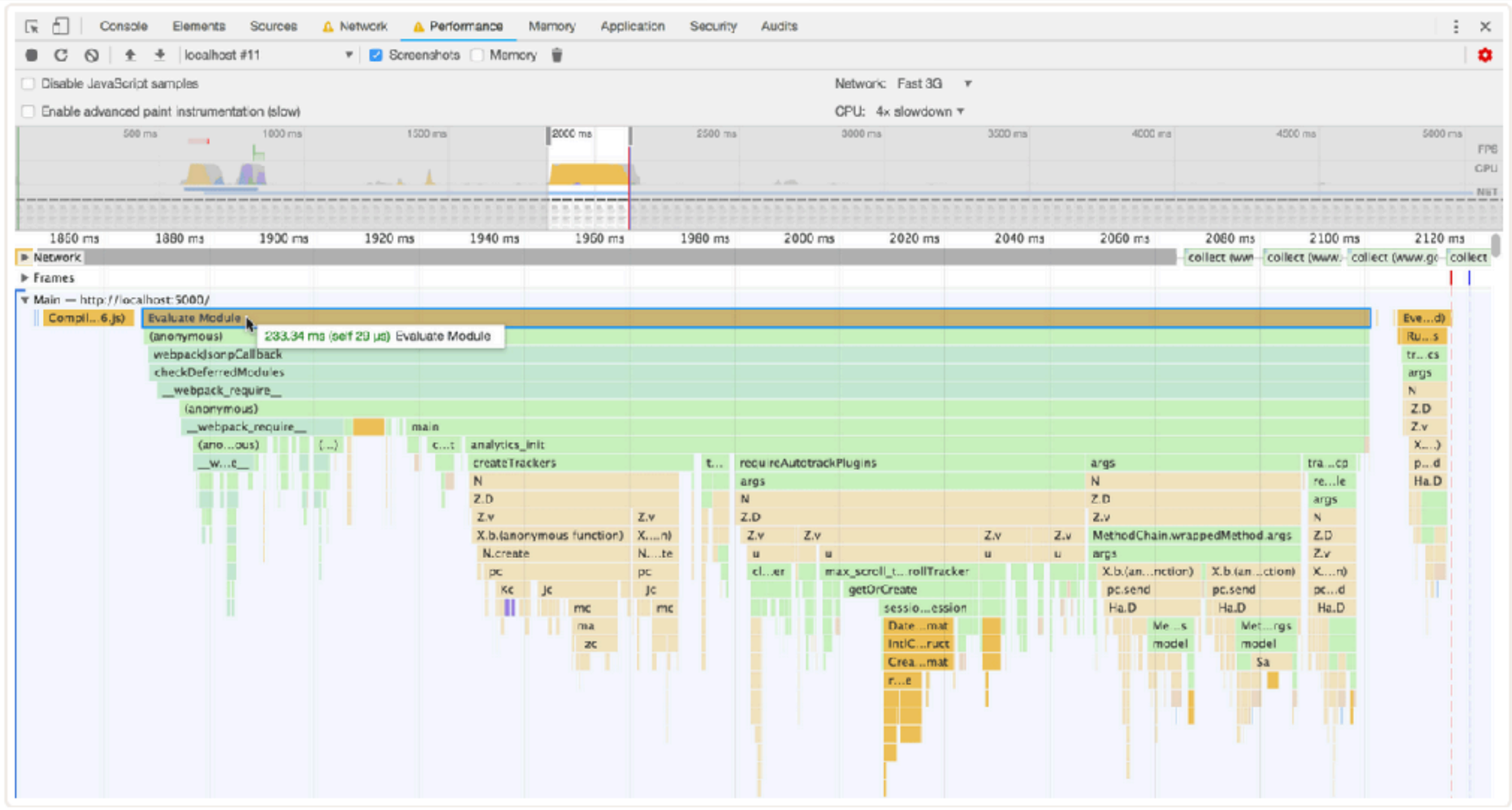
Speed Up JavaScript Execution

By Kayce Basques
Technical Writer, Chrome DevTools & Lighthouse

By Meggin Kearney
Meggin is a Tech Writer

Identify expensive functions using the Chrome DevTools CPU Profiler.





A performance trace of my site's JavaScript while loading (with network/CPU throttling enabled).

https://philipwalton.com/articles/idle-until-urgent/

First Input Delay (FID)

Nov 7, 2019 • Updated May 4, 2020

Appears in: Metrics

Philip Walton
Twitter · GitHub · Blog

★ First Input Delay (FID) is an important, user-centric metric for measuring load responsiveness because it quantifies the experience users feel when trying to interact with unresponsive pages—a low FID helps ensure that the page is usable.

We all know how important it is to make a good first impression. It's important when meeting new people, and it's also important when building experiences on the web

https://web.dev/fid/

window.requestIdleCallback()

This is an experimental technology
Check the Browser compatibility table carefully before using this in production.

The `window.requestIdleCallback()` method queues a function to be called during a browser's idle periods. This enables developers to perform background and low priority work on the main event loop, without impacting latency-critical events such as animation and input response. Functions are generally called in first-in-first-out order; however, callbacks which have a `timeout` specified may be called out-of-order if necessary in order to run them before the timeout elapses.

You can call `requestIdleCallback()` within an idle callback function to schedule another callback to take place no sooner than the next pass through the event loop.

A `timeout` option is strongly recommended for required work, as otherwise it's possible multiple seconds will elapse before the callback is fired.

https://developer.mozilla.org/en-US/docs/Web/API/Window/requestIdleCallback

Window.requestAnimationFrame()

The `window.requestAnimationFrame()` method tells the browser that you wish to perform an animation and requests that the browser calls a specified function to update an animation before the next repaint. The method takes a callback as an argument to be invoked before the repaint.

> **Note:** Your callback routine must itself call `requestAnimationFrame()` if you want to animate another frame at the next repaint.

You should call this method whenever you're ready to update your animation onscreen. This will request that your animation function be called before the browser performs the next repaint. The number of callbacks is usually 60 times per second, but will generally match the display refresh rate in most web browsers as per W3C recommendation. `requestAnimationFrame()` calls are paused in most browsers when running in background tabs or hidden `<iframe>`s in order to improve performance and battery life.

The callback method is passed a single argument, a `DOMHighResTimeStamp`, which indicates the current time (based on the number of milliseconds since time origin). When

https://developer.mozilla.org/en-US/docs/Web/API/Window/requestAnimationFrame

Intersection Observer v1 is one of those APIs that's probably universally loved, and, now that Safari supports it as well, it's also finally universally usable in all major browsers. For a quick refresher of the API, I recommend watching Surma's Supercharged Microtip on Intersection Observer v1—also embedded below for your viewing pleasure—or reading Surma's in-depth article. People have used Intersection Observer v1 for a wide range of use cases like lazy loading of images and videos, being notified when elements reach `position: sticky`, fire analytics events, and many more.

For the full details, check out the Intersection Observer docs on MDN, but as a short reminder, this is what the Intersection Observer v1 API looks like in the most basic case:

https://developers.google.com/web/updates/2019/02/intersectionobserver-v2?hl=en

TTI goals recap...

- Lean harder on native browser features to get you farther along with less (or no!) JavaScript
- Break apart scripts to load only what you need when you need it
- Optimize how the scripts you must run are running in the browser so they're gentler on the CPU