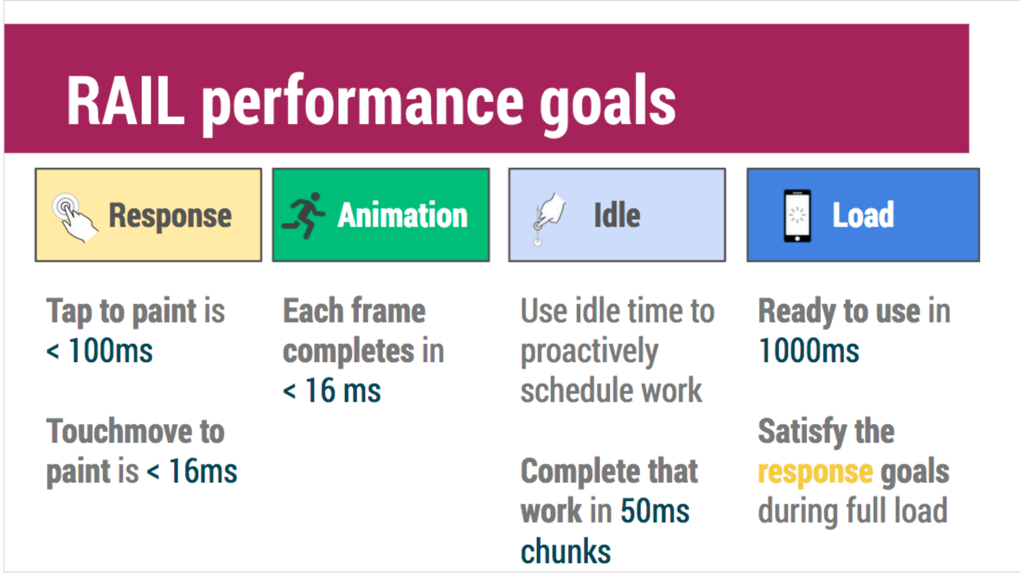
Cafepress performance audit

paul irish, PM devtools & performance, google chrome
april 28, 2015

Setup: 2013 Moto X. Chrome Canary (44.0.2384.0). On wifi.

We are evaluating the site using RAIL as a model for best user experience performance.
More details on RAIL: How Users Perceive the Speed of The Web

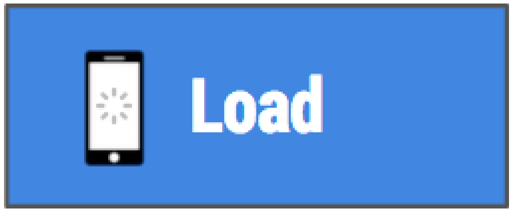


RAIL performance goals

Response	Animation	Idle	Load
Tap to paint is < 100ms Touchmove to paint is < 16ms	Each frame completes in < 16 ms	Use idle time to proactively schedule work Complete that work in 50ms chunks	Ready to use in 1000ms Satisfy the response goals during full load

Scenarios investigated

Scenario: Loading the homepage. Empty cache.
Scenario: Loading a product page. Empty cache.
Scenario: Scrolling the homepage.
Scenario: Tapping on the hamburger menu.
Scenario: Add item to the cart.



Scenario: Loading the homepage. Empty cache.

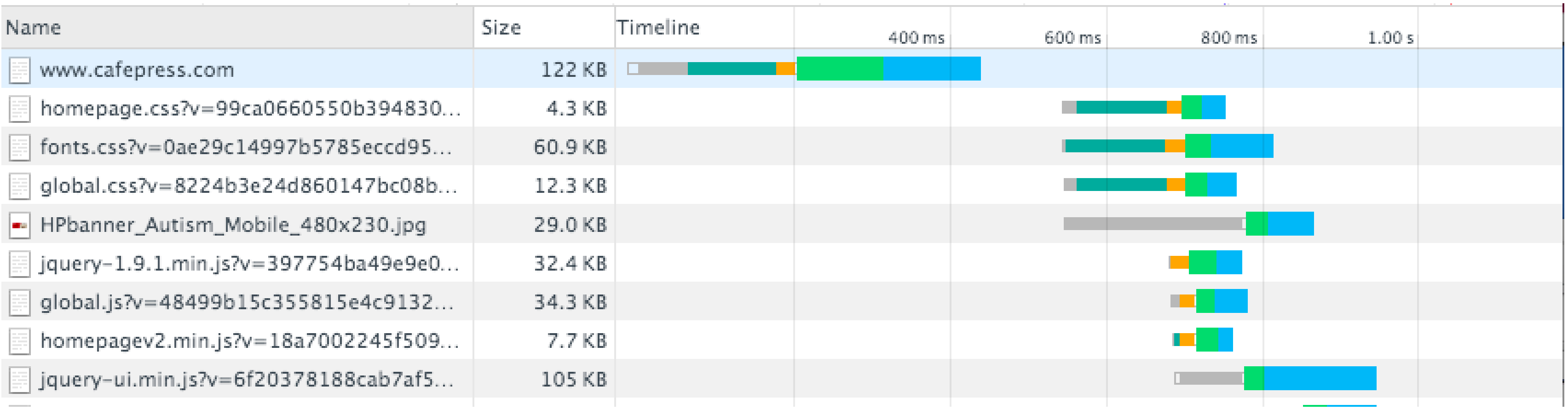
Goal: Ready to use in 1000ms

Result: Ready to use in 3200ms

Initial Render

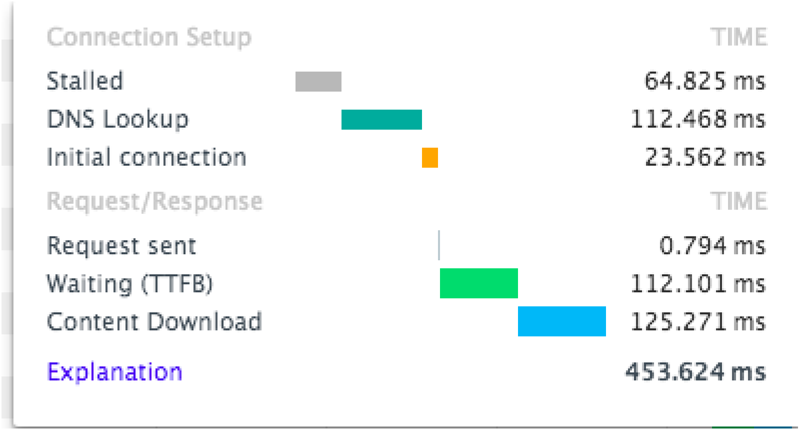
We get an initial render after 1.12s. This is on Wifi.

These are all the render blocking requests sitting between the request and the first paint:



Name	Size	Timeline
www.cafepress.com	122 KB	
homepage.css?v=99ca0660550b394830...	4.3 KB	
fonts.css?v=0ae29c14997b5785eccd95...	60.9 KB	
global.css?v=8224b3e24d860147bc08b...	12.3 KB	
HPbanner_Autism_Mobile_480x230.jpg	29.0 KB	
jquery-1.9.1.min.js?v=397754ba49e9e0...	32.4 KB	
global.js?v=48499b15c355815e4c9132...	34.3 KB	
homepagev2.min.js?v=18a7002245f509...	7.7 KB	
jquery-ui.min.js?v=6f20378188cab7af5...	105 KB	

The first request (for the HTML) is is within expected bounds.

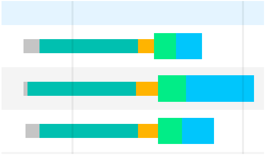


```
Connection Setup                        TIME
Stalled                            64.825 ms
DNS Lookup                        112.468 ms
Initial connection                 23.562 ms
Request/Response                        TIME
Request sent                        0.794 ms
Waiting (TTFB)                    112.101 ms
Content Download                  125.271 ms
Explanation                       453.624 ms
```

112ms of waiting is a little long, indicating the backend serving HTML probably has a few more caching opportunities. Compare it to the waiting of the static requests.

But after the HTML we immediately have 7 new render blocking requests.

They come from a new domain so we have a fresh DNS lookup, which is what the big green delay on these are:



Had all of this CSS and JS been two requests instead of 7, I would expect them to complete maybe 30-40% faster. Taking the first paint from 1120ms to about 1020ms.

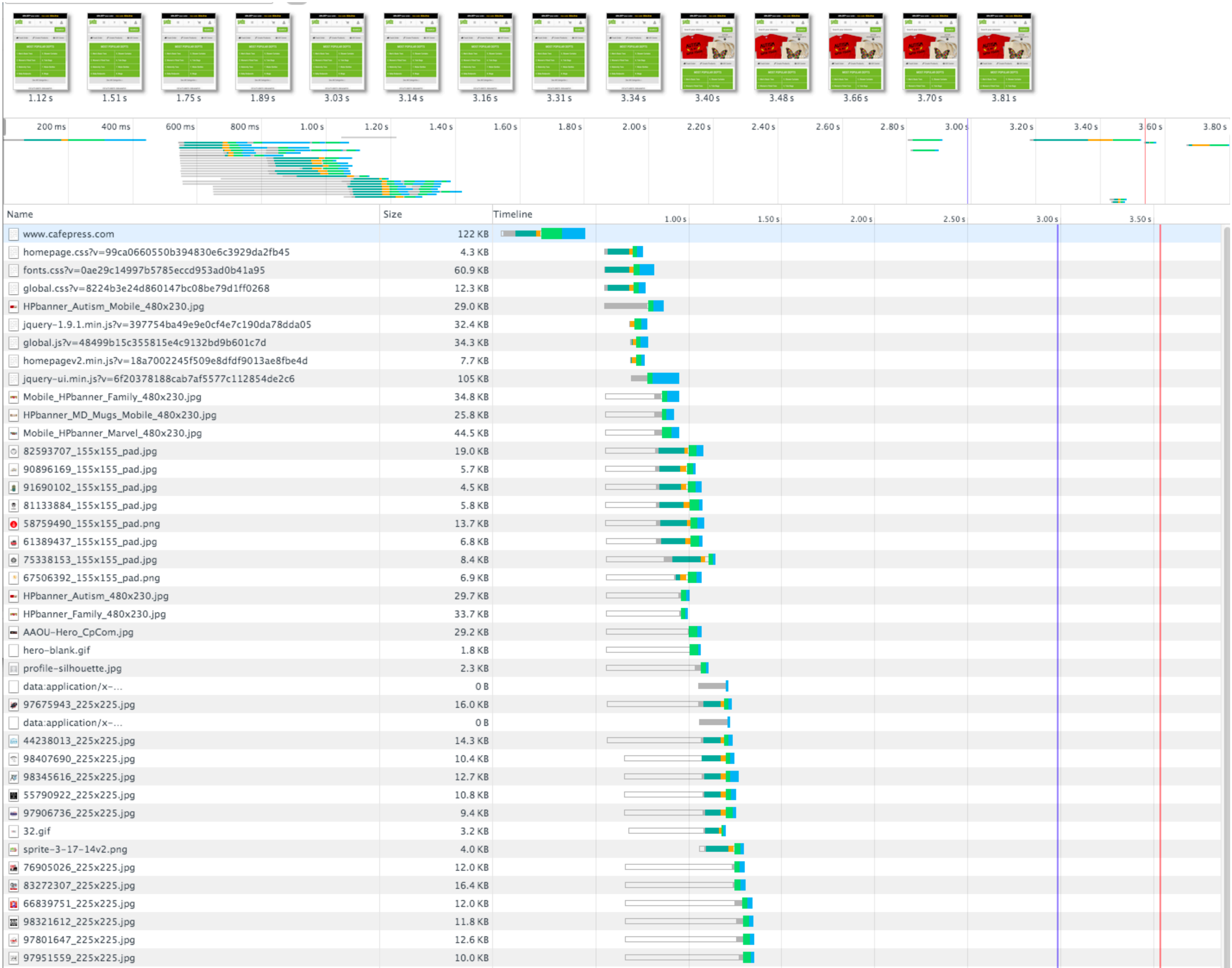
The big issue here is that require new assets to render a first paint. jQuery and jQuery UI are not required to be executed before the user sees pixels.

The CSS is scoped to just “homepage.css”, which is excellent. This should be included inline in the HTML, following critical path guidelines. https://developers.google.com/speed/docs/insights/OptimizeCSSDelivery

Visually complete

While the user gets some green boxes at 1120ms in, they don’t get text and our hero image until 3400ms.

Here’s the full waterfall with filmstrip. Explanation below



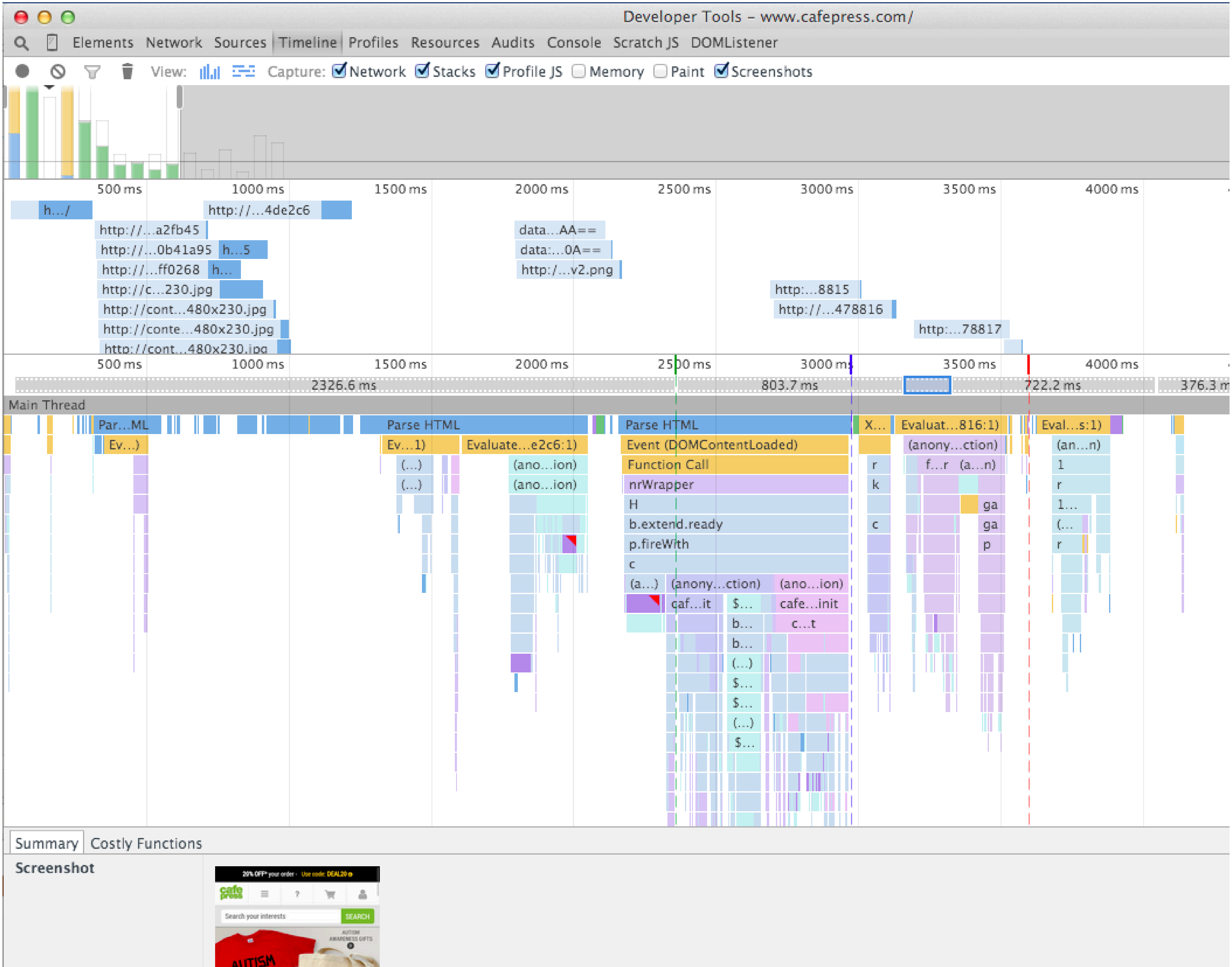
Name	Size	Timeline
www.cafepress.com	122 KB	
homepage.css?v=99ca0660550b394830e6c3929da2fb45	4.3 KB	
fonts.css?v=0ae29c14997b5785eccd953ad0b41a95	60.9 KB	
global.css?v=8224b3e24d860147bc08be79d1ff0268	12.3 KB	
HPbanner_Autism_Mobile_480x230.jpg	29.0 KB	
jquery-1.9.1.min.js?v=397754ba49e9e0cf4e7c190da78dda05	32.4 KB	
global.js?v=48499b15c355815e4c9132bd9b601c7d	34.3 KB	
homepagev2.min.js?v=18a7002245f509e8dfdf9013ae8fbe4d	7.7 KB	
jquery-ui.min.js?v=6f20378188cab7af5577c112854de2c6	105 KB	
Mobile_HPbanner_Family_480x230.jpg	34.8 KB	
HPbanner_MD_Mugs_Mobile_480x230.jpg	25.8 KB	
Mobile_HPbanner_Marvel_480x230.jpg	44.5 KB	
82593707_155x155_pad.jpg	19.0 KB	
90896169_155x155_pad.jpg	5.7 KB	
91690102_155x155_pad.jpg	4.5 KB	
81133884_155x155_pad.jpg	5.8 KB	
58759490_155x155_pad.png	13.7 KB	
61389437_155x155_pad.jpg	6.8 KB	
75338153_155x155_pad.jpg	8.4 KB	
67506392_155x155_pad.png	6.9 KB	
HPbanner_Autism_480x230.jpg	29.7 KB	
HPbanner_Family_480x230.jpg	33.7 KB	
AAOU-Hero_CpCom.jpg	29.2 KB	
hero-blank.gif	1.8 KB	
profile-silhouette.jpg	2.3 KB	
data:application/x-…	0 B	
97675943_225x225.jpg	16.0 KB	
data:application/x-…	0 B	
44238013_225x225.jpg	14.3 KB	
98407690_225x225.jpg	10.4 KB	
98345616_225x225.jpg	12.7 KB	
55790922_225x225.jpg	10.8 KB	
97906736_225x225.jpg	9.4 KB	
32.gif	3.2 KB	
sprite-3-17-14v2.png	4.0 KB	
76905026_225x225.jpg	12.0 KB	
83272307_225x225.jpg	16.4 KB	
66839751_225x225.jpg	12.0 KB	
98321612_225x225.jpg	11.8 KB	
97801647_225x225.jpg	12.6 KB	
97951559_225x225.jpg	10.0 KB	

The visual complete paint comes between the blue & red lines, which is significantly later than when most of the network activity finished.

Why?

The first clue is globalDeferredMobile.js being one of those late requests.

And why is it requested so late, and delaying our paint?
To answer that we go to the Timeline.

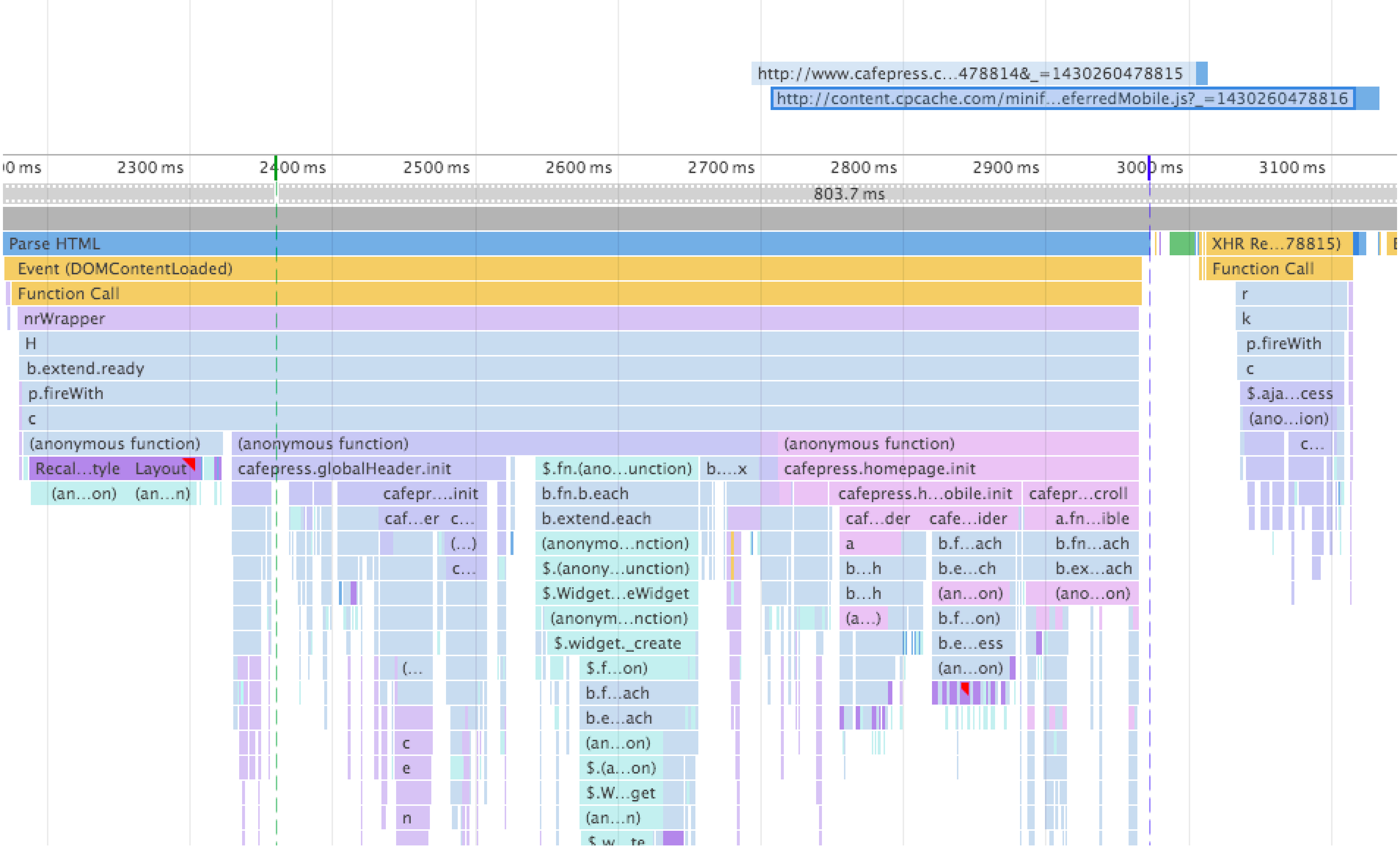


Developer Tools – www.cafepress.com/

Capture: ☑ Network ☑ Stacks ☑ Profile JS □ Memory □ Paint ☑ Screenshots

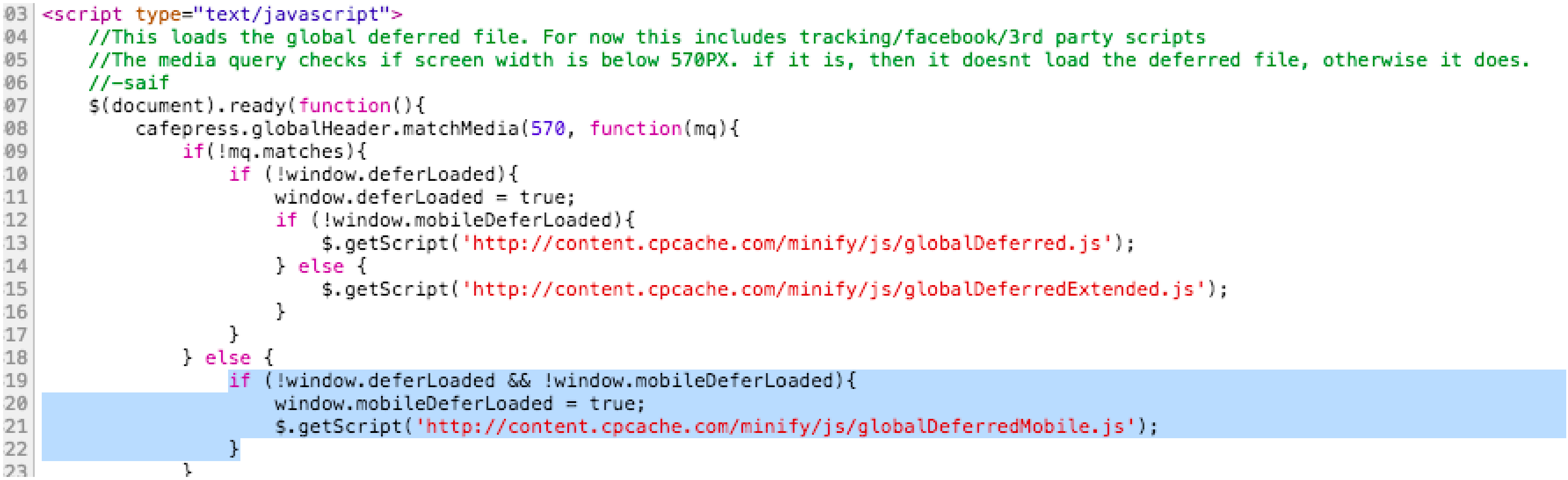
Zoom in a bit and focus on those late requests.

1. the pencilbanner JSONP request. Pretty harmless.
2. globalDeferredMobile.js



It was requested during a pretty massive handler for document ready -- almost 1000ms in total.

We can find the request in the source:

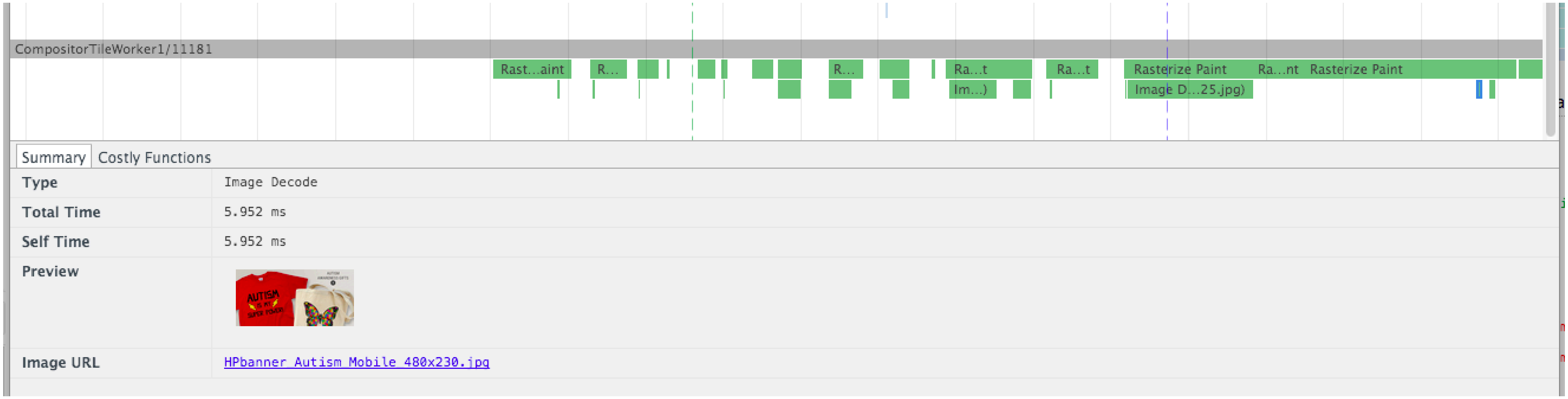


```
<script type="text/javascript">
    //This loads the global deferred file. For now this includes tracking/facebook/3rd party scripts
    //The media query checks if screen width is below 570PX. if it is, then it doesnt load the deferred file, otherwise it does.
    //-saif
    $(document).ready(function(){
        cafepress.globalHeader.matchMedia(570, function(mq){
            if(!mq.matches){
                if (!window.deferLoaded){
                    window.deferLoaded = true;
                    if (!window.mobileDeferLoaded){
                        $.getScript('http://content.cpcache.com/minify/js/globalDeferred.js');
                    } else {
                        $.getScript('http://content.cpcache.com/minify/js/globalDeferredExtended.js');
                    }
                }
            } else {
                if (!window.deferLoaded && !window.mobileDeferLoaded){
                    window.mobileDeferLoaded = true;
                    $.getScript('http://content.cpcache.com/minify/js/globalDeferredMobile.js');
                }
            }
```

It's unclear to me if globalDeferredMobile.js is required for finishing the rendering of the homepage.

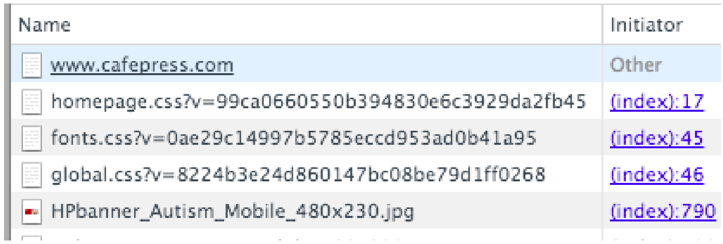
- **If it is required:** It's request doesn't' start until ⅔ of the way through the DOMContentLoaded (DCL) handler, but more importantly… If this file is required for the first view, it should be started WAY before DCL, because at this point we're already 2600ms after the page was requested.
- **If it's not required:** Requesting in DCL is fine, you may even want to delay until window.load, as things are still fairly busy.

Of all the images to be downloaded, most of them are decoded before our hero image.



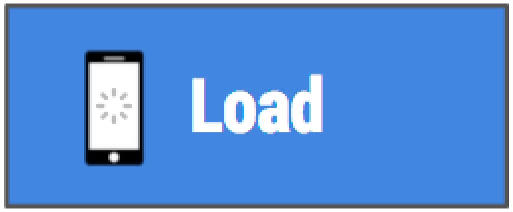
Type	Image Decode
Total Time	5.952 ms
Self Time	5.952 ms
Preview	
Image URL	HPbanner_Autism_Mobile_480x230.jpg

However it was the first image to be downloaded.



Name	Initiator
www.cafepress.com	Other
homepage.css?v=99ca0660550b394830e6c3929da2fb45	(index):17
fonts.css?v=0ae29c14997b5785eccd953ad0b41a95	(index):45
global.css?v=8224b3e24d860147bc08be79d1ff0268	(index):46
HPbanner_Autism_Mobile_480x230.jpg	(index):790

Very curious.
Something to investigate.

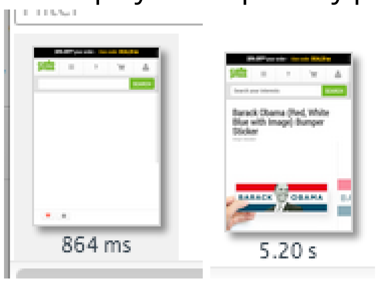


Scenario: Loading a product page. Empty cache.

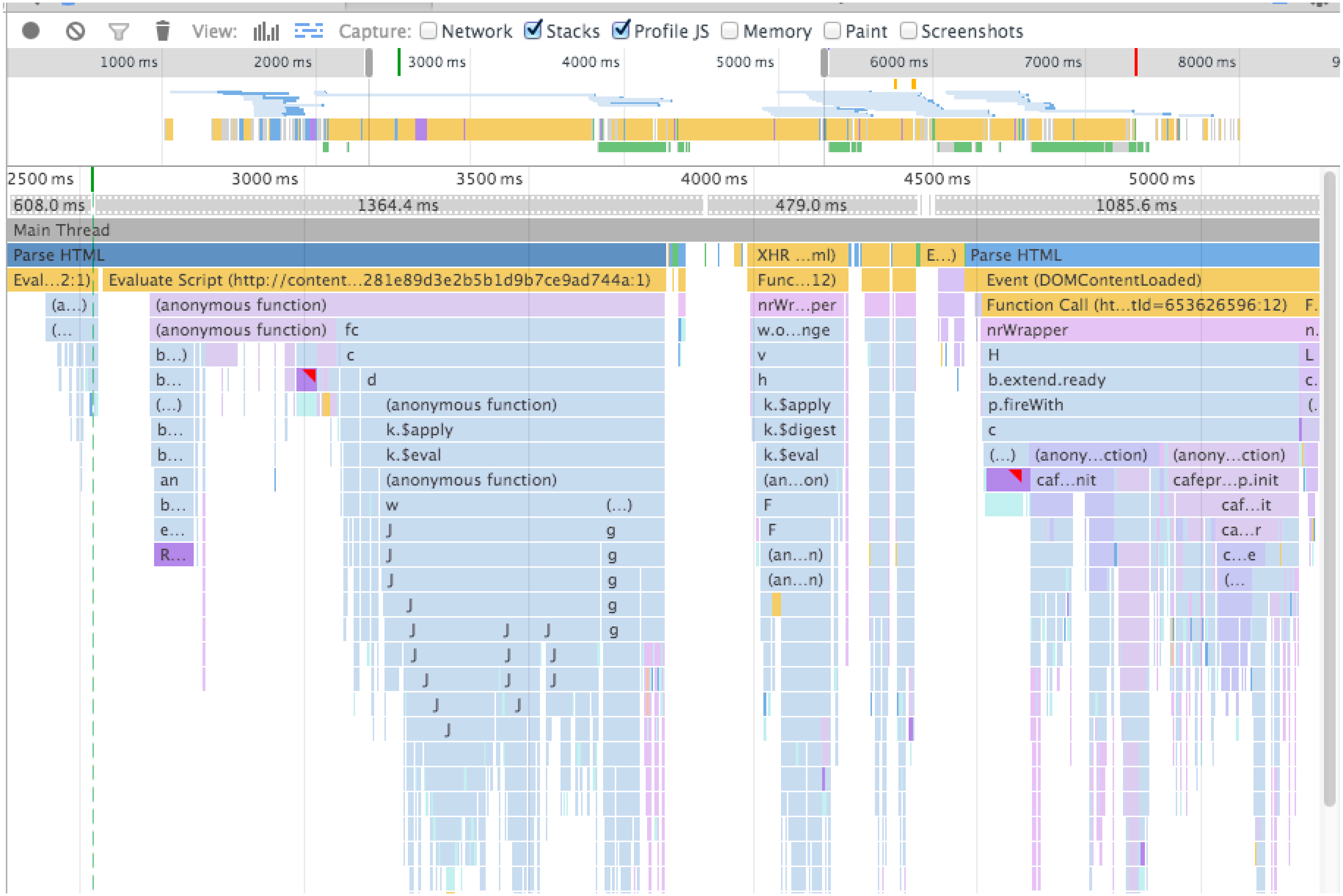
Goal: Ready to use in 1000ms

Result: Ready to use in 5000ms

The display of the primary product image is massively delayed. Initial render in under a second but takes 5 more to get the product up.



What's happening in those 5 seconds?



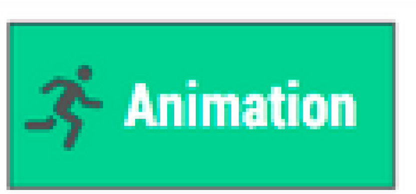
Capture: □ Network ☑ Stacks ☑ Profile JS □ Memory □ Paint □ Screenshots

What do we see?

- 1.5s of it is angular booting up
- .7s of it is the init() stuff for cafepress
- Then another 1s of it is handling the product details in JS and doing clientside templating.

Ideally the product page wouldn't rely on javascript to render its first view. Perhaps explore serverside rendering.

Removing angular, handlebars and relying on the HTML to render the view would reduce 5000ms to ~2200ms.



Scenario: Scrolling the homepage.

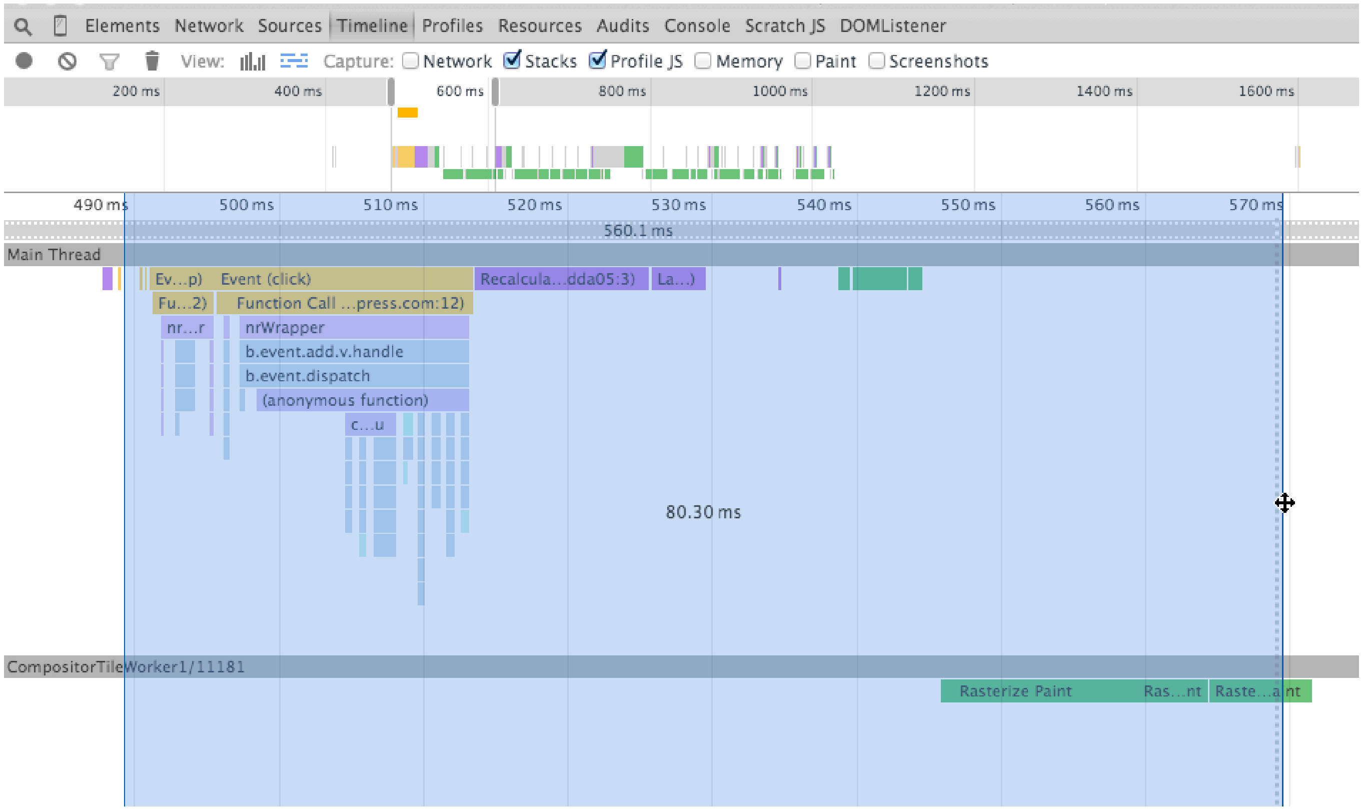
Goal: 60fps scrolling
Result: ~60fps scrolling.

Looks good!

Scenario: Tapping on the hamburger menu.

Goal: Tap to paint is < 100ms
Result: Tap to paint is ~ 80ms

Tapping on the hamburger menu.

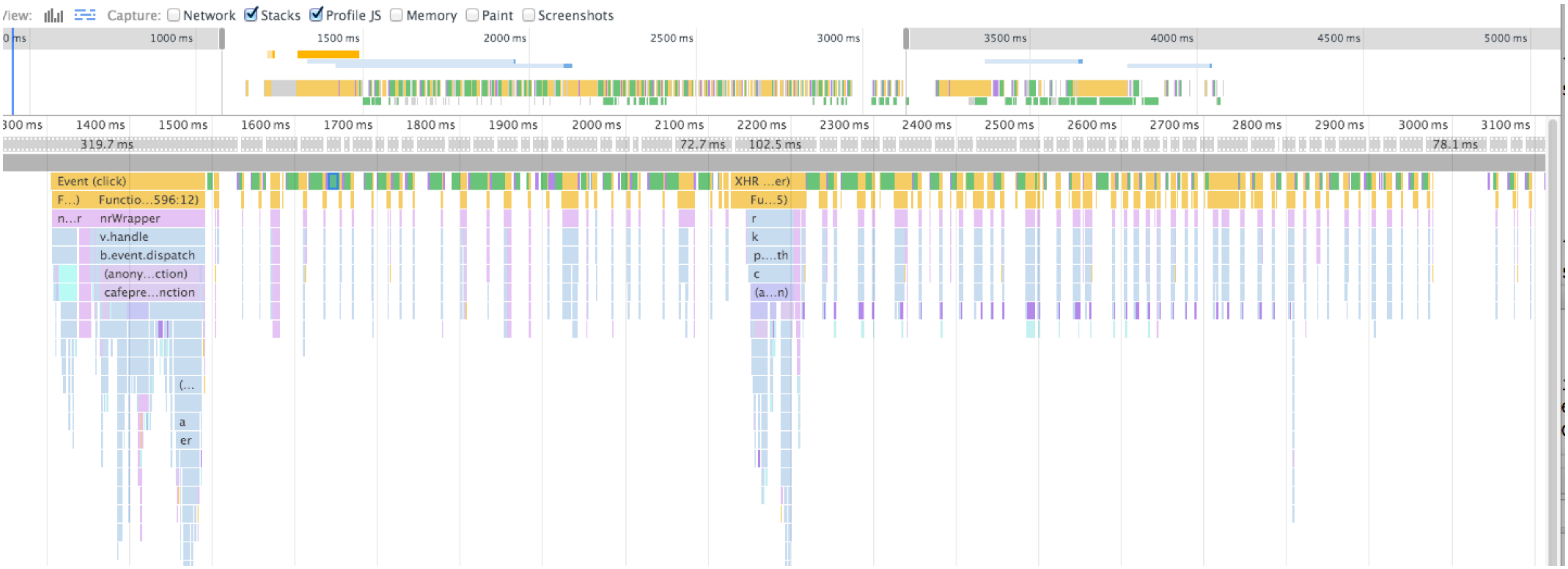


Seems good!

Scenario: Add item to the cart.
Adding an item to the cart does an animation to place it in the topnav cart icon.
We will break it into two parts, the initial response and the subsequent animation.

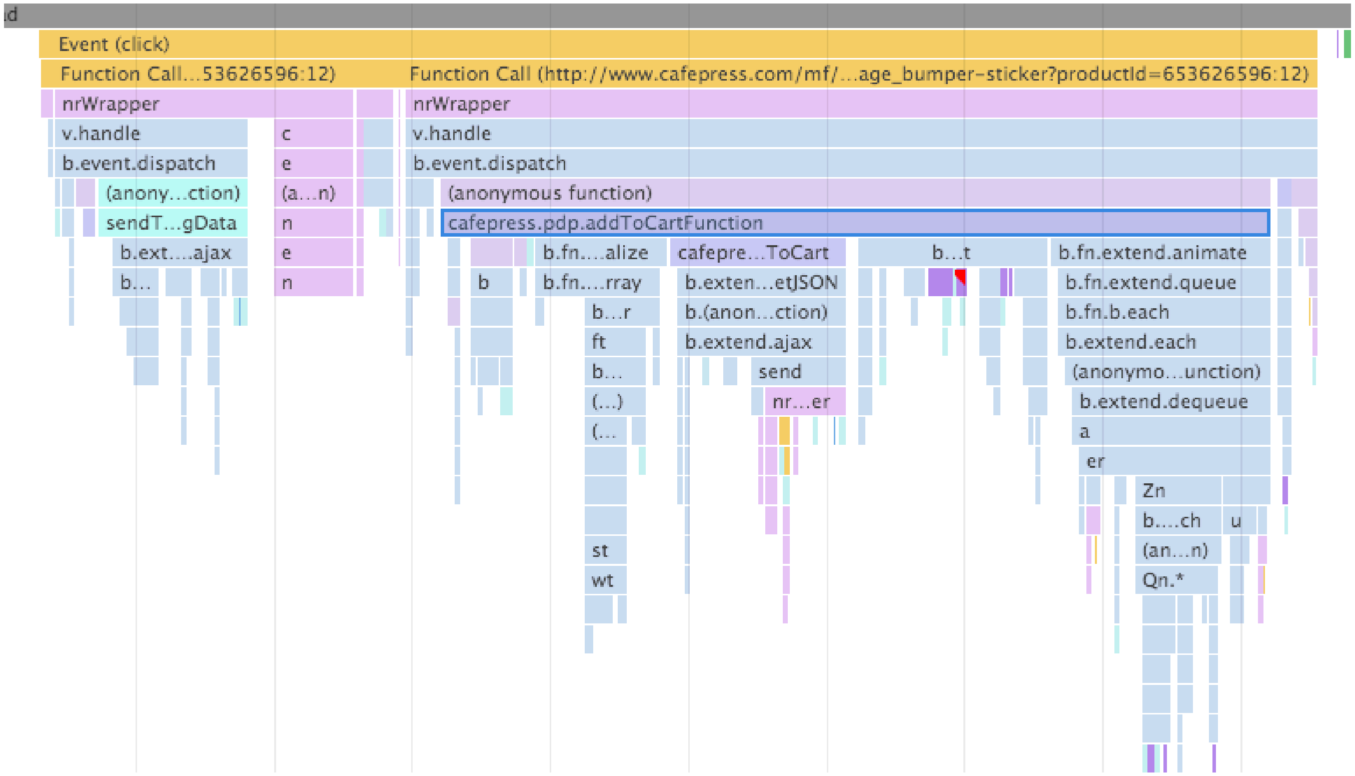
Goal: Tap to paint is < 100ms, then 60fps animation thereafter

Result: Tap to paint is ~ 180ms, then ~33fps animation.

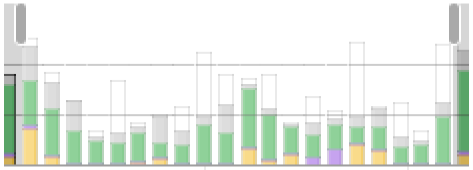


View: Capture: □ Network ☑ Stacks ☑ Profile JS □ Memory □ Paint □ Screenshots

The addToCart takes 180ms. No low-hanging fruit, just a lot of work that happens before anything is shown to the user:

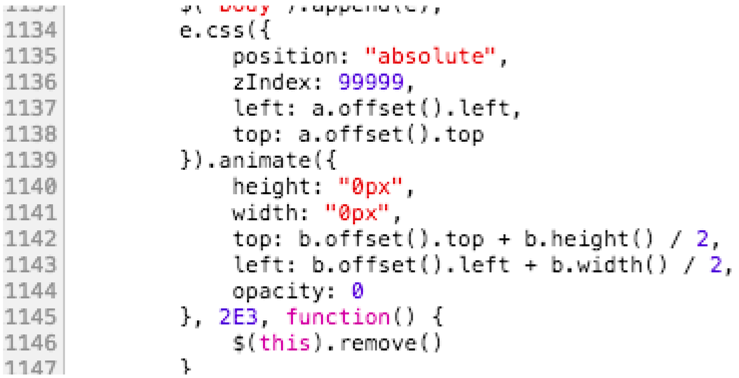


After that we animate the product across the screen.



The bottom gray bar is 60fps and we’re rarely hitting it. We’re missing the frame budget mostly due to heavy painting.

In addToCartFunction we can see why:



```
        $('body').append(e);
        e.css({
            position: "absolute",
            zIndex: 99999,
            left: a.offset().left,
            top: a.offset().top
        }).animate({
            height: "0px",
            width: "0px",
            top: b.offset().top + b.height() / 2,
            left: b.offset().left + b.width() / 2,
            opacity: 0
        }, 2E3, function() {
            $(this).remove()
        }
```

The product is animated using left/top which causes paint storms. It should animate using transforms instead. High Performance Animations - HTML5 Rocks
Additionally, jQuery should not be used for this animation, and it should be set up using CSS transitions

Cafepress inights

- Critical path CSS for a quick first paint.
- Defer homepage JS to bottom of <body>. Put [async] on them.
- Explore why the hero image of the homepage carousel is decoded so late.
- Explore serverside rendering to get rid of depending on JS & Handlebars to render product page
- Don’t use jQuery for animation, use CSS transitions.
- Animate transforms and never top/left
- Consider deferring some addToCartLogic until later, so the feedback can begin sooner.

Blink insights

- Investigate why homepage hero image is decoded so late
- Natural animations ftw!
- Some decodes were massively descheduled. This is a two-core phone. Anything we can do there?