Tumblr dashboard loading perf

3/2/2016 - by @samccone and @paul_irish

🔥 10 second TL;DR 🔥

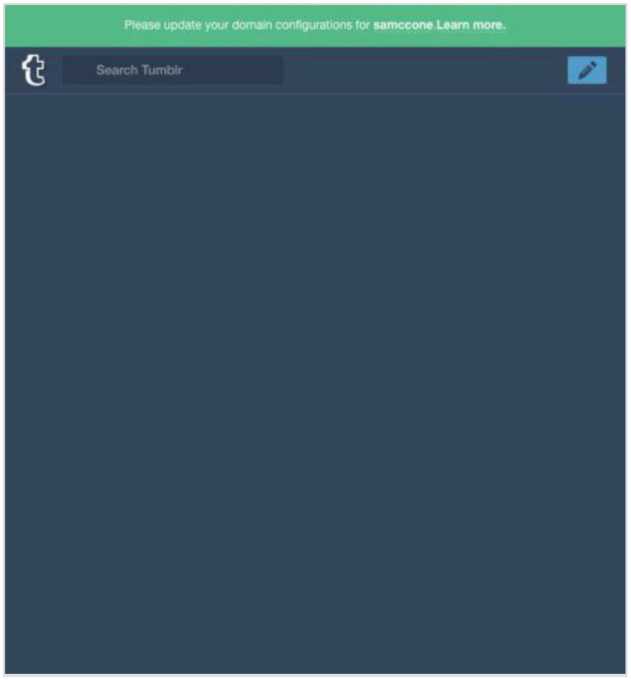
Do not intermix <script> tags within your body html markup, throttle your reflow causing code, and be aware of the overhead browserify causes on large codebases.

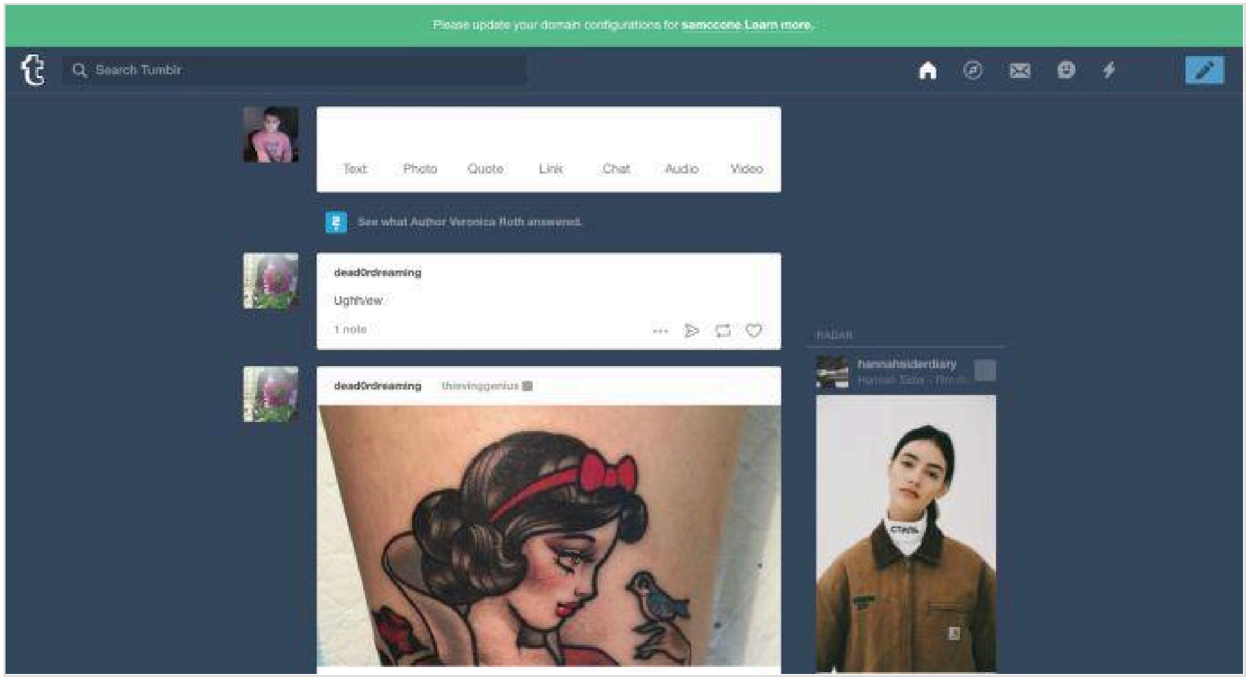
If you have ever used tumblr before as a logged in user you have most likely have experienced loading jank when your dashboard loads in. As a user you see a flash of the page with no posts and then you see the posts but can not interact with the page for around a second. It is easy to write off this behavior as just how the load in works, but that is a poor rationalization; let's take a deeper look into how we can prevent this flash and how we can make the site boot fast.

Table of contents:

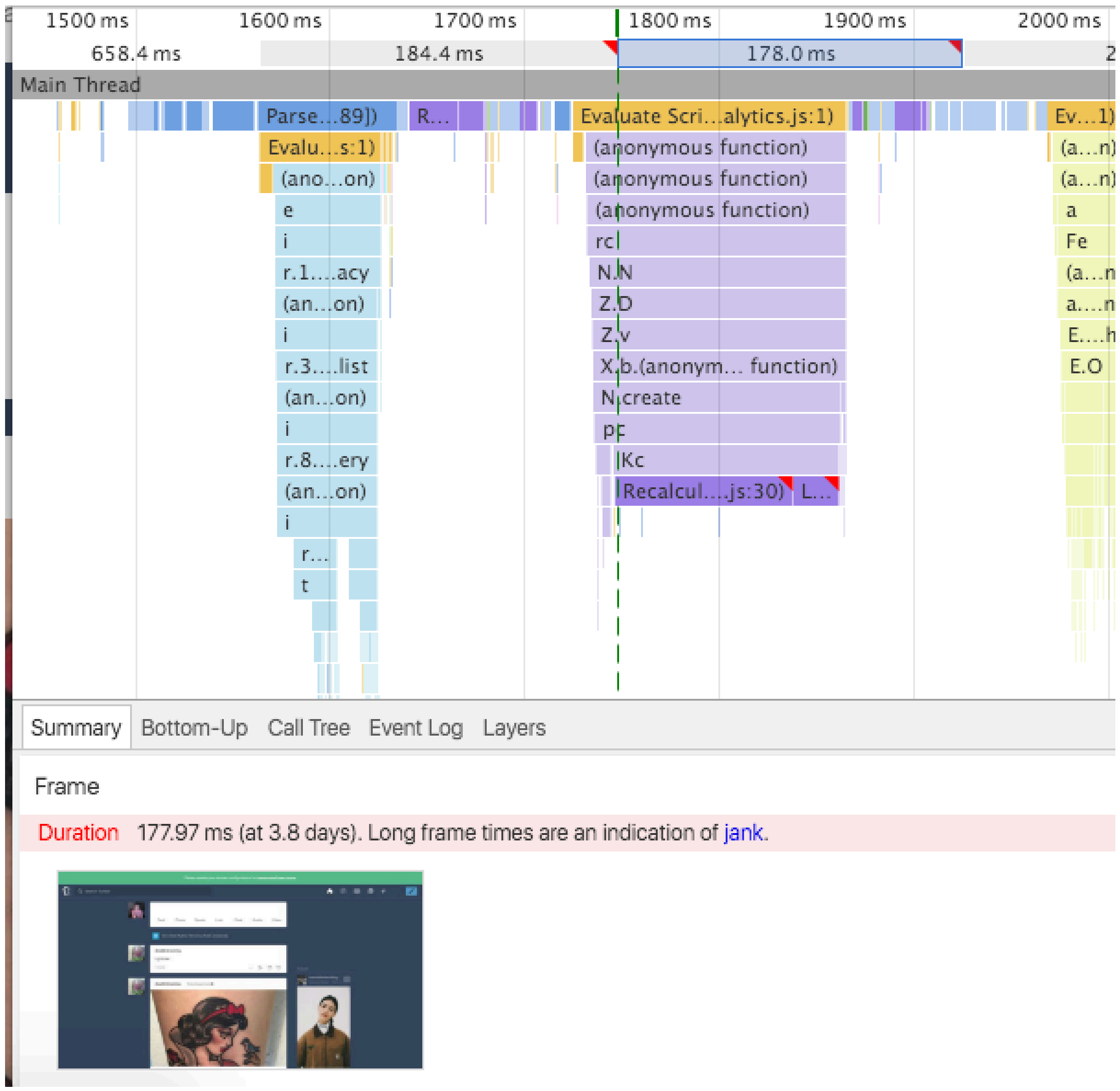
The Flash

The initial paint of tumblr is really interesting, we go from an emptyish document into a mostly filled in page that is still loading a few assets.


Please update your domain configurations for samccone.Learn more.
Search Tumblr



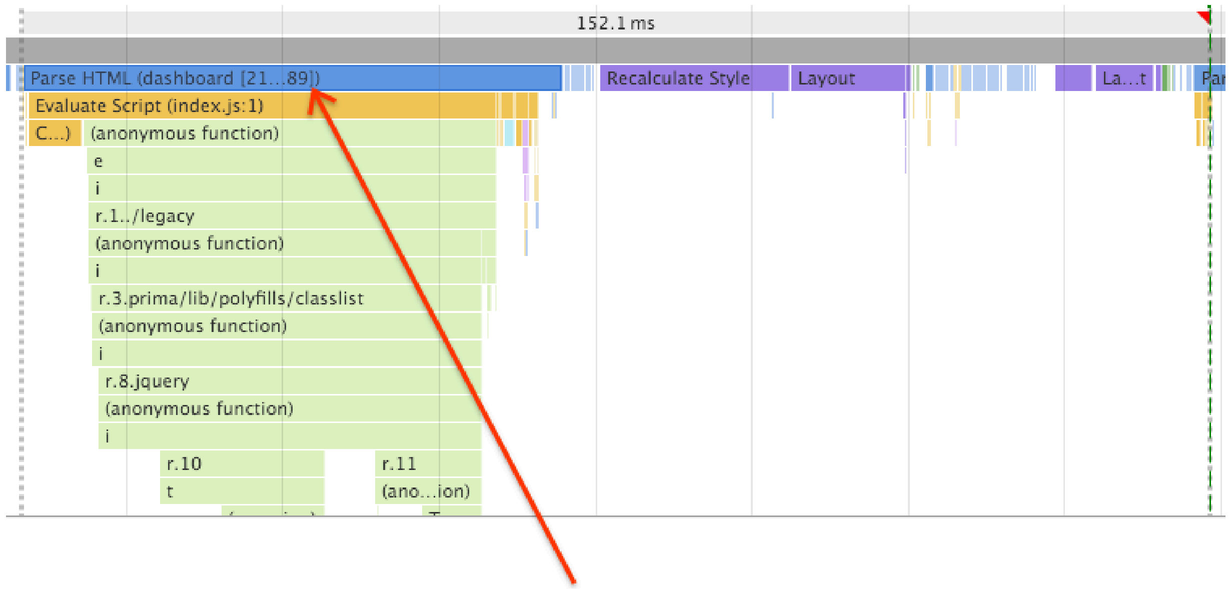
Normally this kind of paint in is a pretty expected thing, and nothing to really take as something being "off". However as a user when you see this load in, you experience it as a "flash", meaning that the first frame that the browser paints is persisted for well under a second. This kind of experience sticks out to me as something being wrong. By using the timeline in chrome devtools we can see if we can confirm this issue and figure out a potential fix to remove this flicker of content.



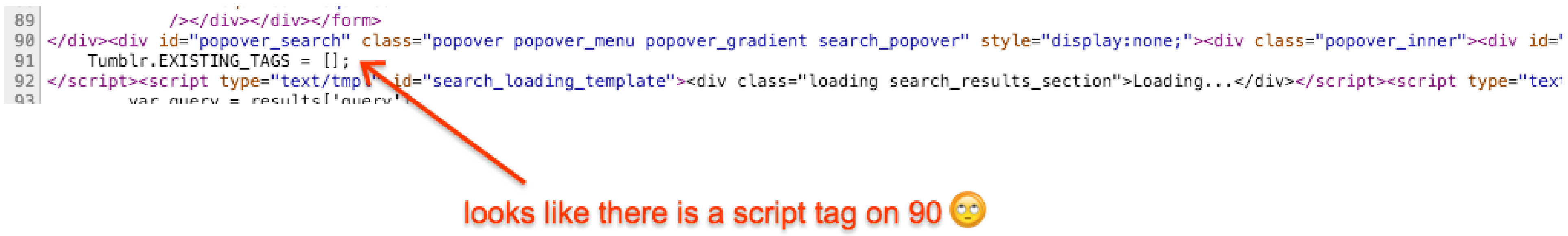
Frame

Duration 177.97 ms (at 3.8 days). Long frame times are an indication of jank.

We can confirm this flash via the above display in devtools, as you can see the first paint at around **1600ms** is the first frame that the browser paints and then roughly **200ms** after the initial paint we get out next frame which is the ideal lo-fi version of the page. The outstanding question is why is there this delay, and how can we fix it?



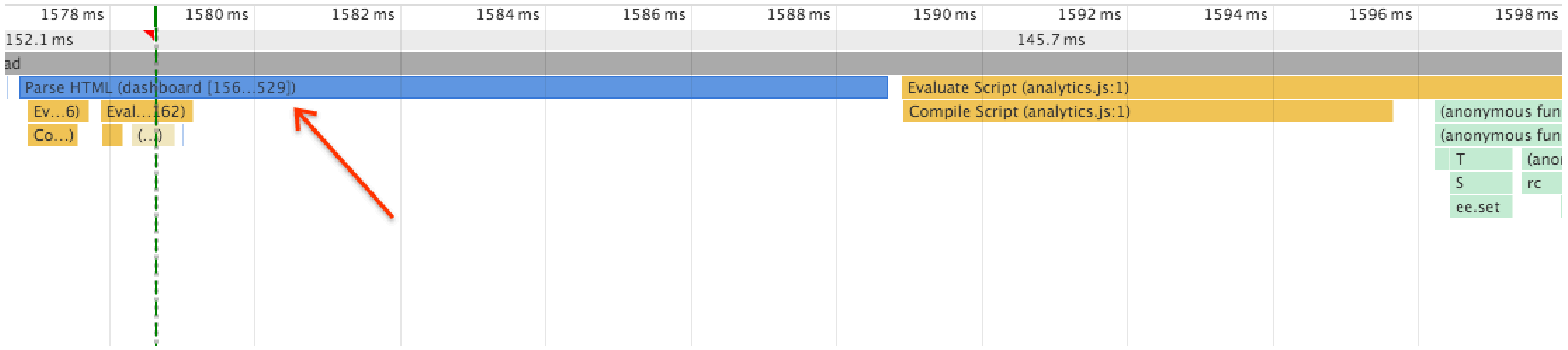
Zooming into the initial frame stack in the timeline we can see that we are parsing part of the html as the response comes down from the server, the browser then recalcs the style and then does a layout and a paint, but it is strange how we are stopping at line 89 in the html document... I wonder why…

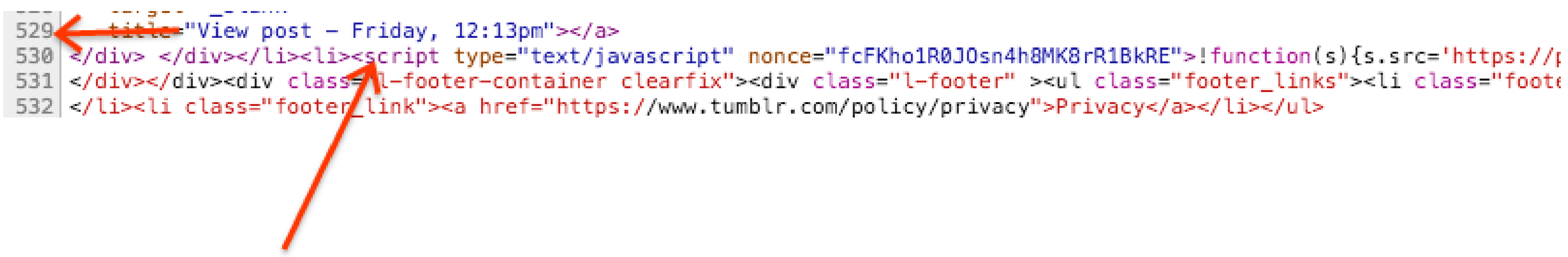


```
89                />< /div></div></form>
90 </div><div id="popover_search" class="popover popover_menu popover_gradient search_popover" style="display:none;"><div class="popover_inner"><div id="
91     Tumblr.EXISTING_TAGS = [];
92 </script><script type="text/tmpl" id="search_loading_template"><div class="loading search_results_section">Loading...</div></script><script type="text
93          var query = results['query'
```

Line 90 in the devtools source panel explains it. The HTML parser has no idea how much script there is to execute, or how long it will take, so the browser bails our and schedules a recalc and layout.… which fills our frame budget and pushes the next chunk of work into the following frame.

The same pattern repeats itself on the next frame where we parse the html up until line 529 where we encounter another script tag mixed within our content

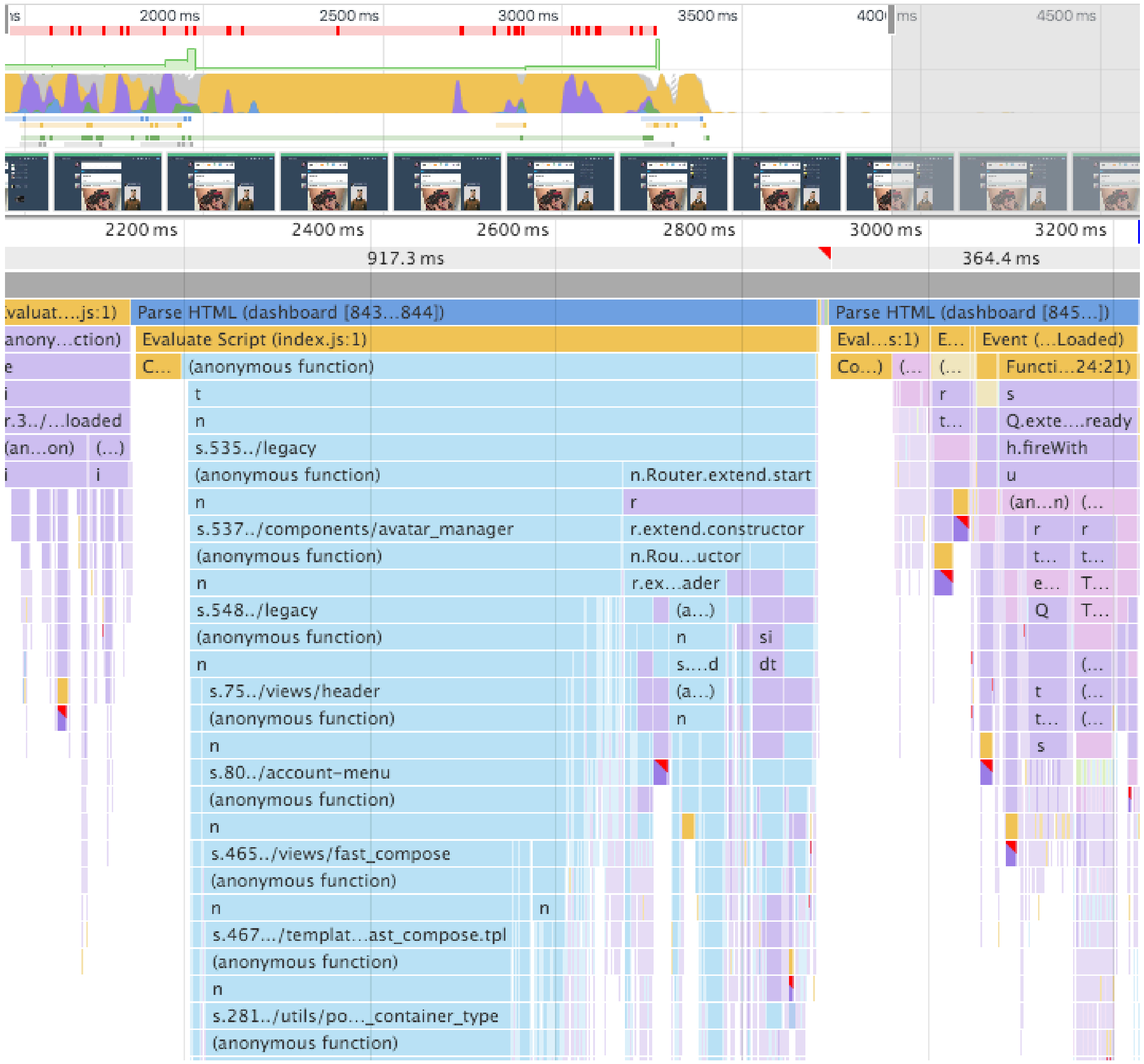




```
    title="View post - Friday, 12:13pm"></a>
</div> </div></li><li><script type="text/javascript" nonce="fcFKho1R0JOsn4h8MK8rR1BkRE">!function(s){s.src='https://p
</div></div><div class="l-footer-container clearfix"><div class="l-footer" ><ul class="footer_links"><li class="foote
</li><li class="footer_link"><a href="https://www.tumblr.com/policy/privacy">Privacy</a></li></ul>
```

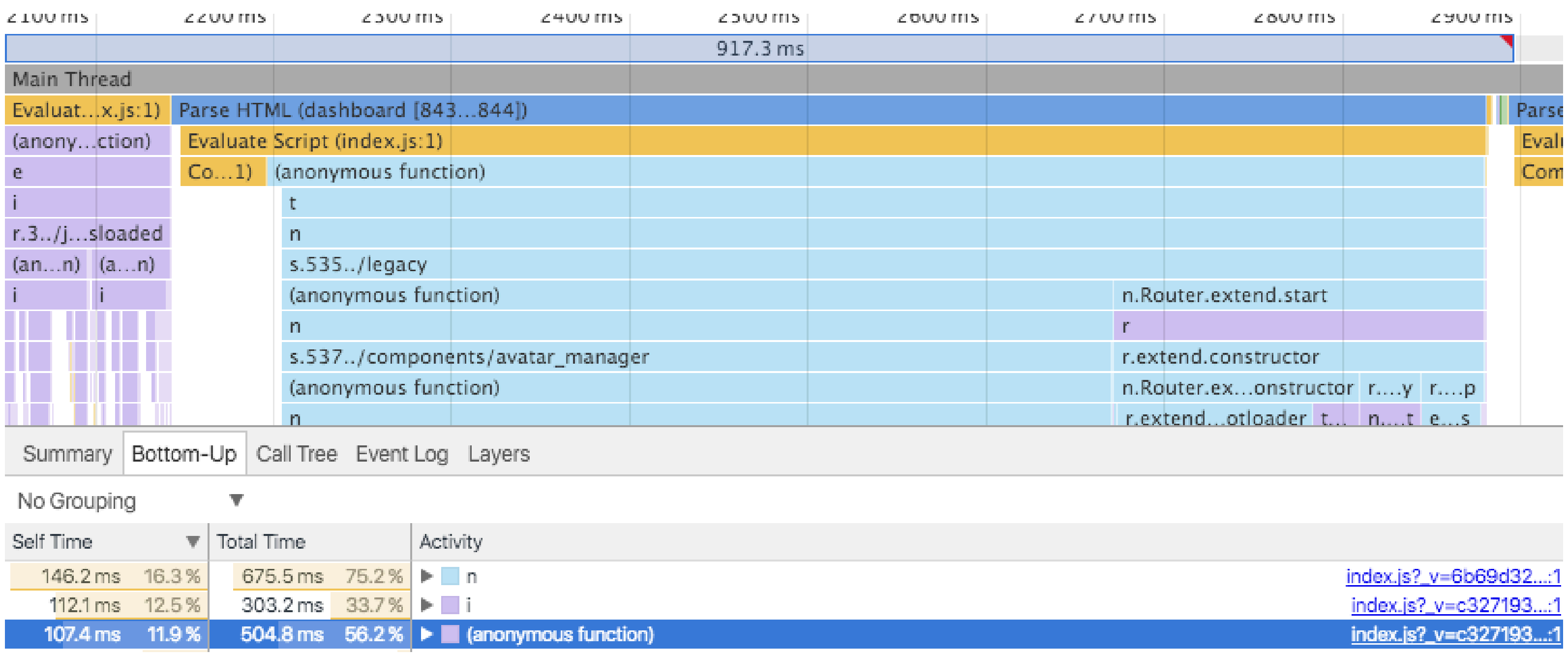
When taken together both of these breaks in the html parser account for the flash and delay for the initial light paint of the document. By removing or moving the inline script tags on the page, or by adding the defer attribute to the script tags, the initial paint experience should no longer have a flicker and should provide a much nicer experience to the user. #win

Mr Freeze


2000 ms
2500 ms
3000 ms
3500 ms
4000 ms
4500 ms
2200 ms
2400 ms
2600 ms
2800 ms
3000 ms
3200 ms
917.3 ms
364.4 ms
Parse HTML (dashboard [843…844])
Parse HTML (dashboard [845…])
Evaluate Script (index.js:1)
Event (…Loaded)
(anonymous function)
Functi…24:21)
Q.exte….ready
h.fireWith
s.535../legacy
n.Router.extend.start
r.extend.constructor
s.537../components/avatar_manager
n.Rou…uctor
r.ex…ader
s.548../legacy
s.75../views/header
s.80../account-menu
s.465../views/fast_compose
s.467…/templat…ast_compose.tpl
s.281../utils/po…_container_type
r.3../…loaded

Once tumblr paints the initial light version of the page, it bootstraps the DOM with some backbone views and other JS enlivened elements. This is pretty typical behavior, and quite expected. What is undesirable about the current behavior is that it drops the user down to 1 frame per second and essentially locks them out of using the page during that time.

As you can see from the above flame chart it looks like the browser is doing quite a bit of work and causing quite a few reflow/recalc actions. The ideal logic here would be to throttle the work and reflow causing paths to allow the browser to iteratively do this JS work without blocking the main thread (check out requestIdleCallback).



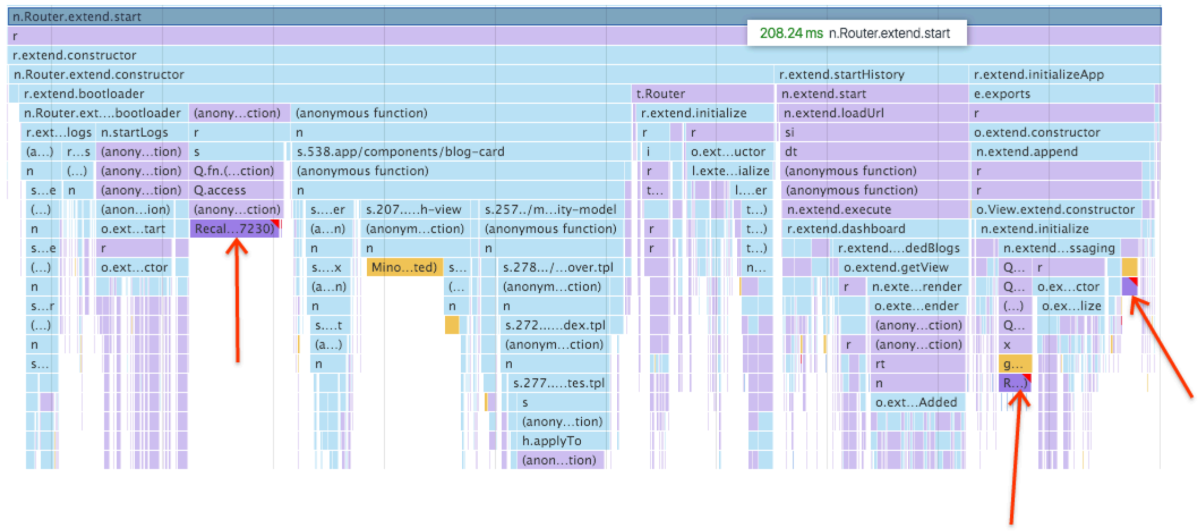
Self Time		Total Time		Activity	
146.2 ms	16.3 %	675.5 ms	75.2 %	▶ n	index.js?_v=6b69d32…:1
112.1 ms	12.5 %	303.2 ms	33.7 %	▶ i	index.js?_v=c327193…:1
107.4 ms	11.9 %	504.8 ms	56.2 %	▶ (anonymous function)	index.js?_v=c327193…:1

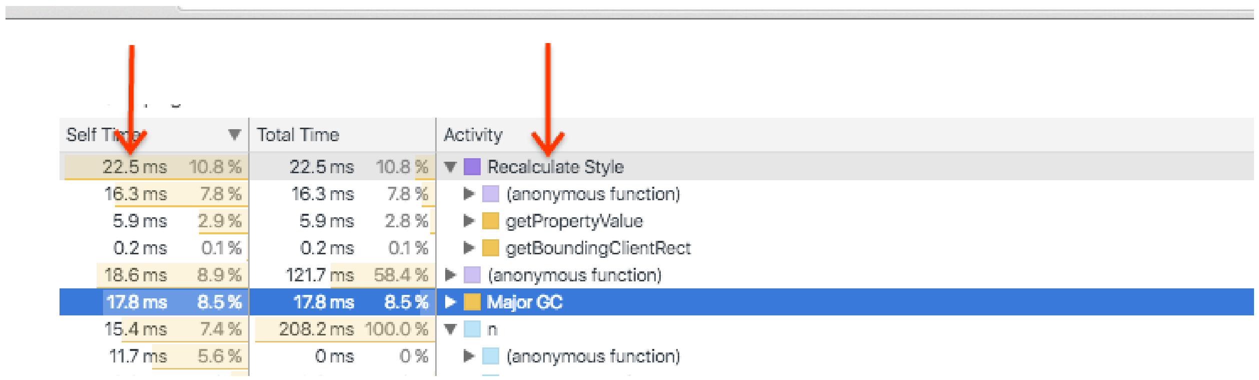
By selecting the frame and looking at the bottom-up call stack and sorting by self time we can identify the heavy methods and try to decipher exactly what things are eating up so much time.

In this case both "n" and "i" are the browserify load in code which points to the fact that over 400ms is being spent simply walking the browserify tree and importing all of the code for browserify. It might be worth investigating something like topological sorting of the dependencies to remove the need for this dynamic require behavior, which should significantly drop this initial boot up time. Take a look at a tool like [rollup]() or [closure compiler.]()

The reflow thrash

The next bit of work in the initial frame is spent kicking off the backbone router. When this happens





Self Time ▼		Total Time		Activity
22.5 ms	10.8 %	22.5 ms	10.8 %	▼ Recalculate Style
16.3 ms	7.8 %	16.3 ms	7.8 %	▶ (anonymous function)
5.9 ms	2.9 %	5.9 ms	2.8 %	▶ getPropertyValue
0.2 ms	0.1 %	0.2 ms	0.1 %	▶ getBoundingClientRect
18.6 ms	8.9 %	121.7 ms	58.4 %	▶ (anonymous function)
17.8 ms	8.5 %	17.8 ms	8.5 %	▶ Major GC
15.4 ms	7.4 %	208.2 ms	100.0 %	▼ n
11.7 ms	5.6 %	0 ms	0 %	▶ (anonymous function)

We are spending a bit of time in recalc style. In the flame chart you can see this by items marked with the red flags. The ideal solution here would be to defer and batch these reflow actions until the next animation frame via something like requestAnimationFrame.

In the end…

In the end, there are 3 major points to look at to improve the initial tumblr load in experience.

1. Remove inline script tags from being intermixed with the initial markup
2. Investigate replacing / augmenting browserify with a tool like rollup to reduce the cost of dynamic requires.
3. Look into batching / deferring reflow causing actions.