@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

HELLO! PLEASE:
DOWNLOAD CHROME CANARY
...AND HAVE A LOCAL RAILS APP W/A DECENTLY COMPLEX FRONTEND RUNNING

FRONT END PERFORMANCE FOR FULL-STACK DEVELOPERS

@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

Understanding Ruby GC through GC.stat

by **Nate Berkopec** (@nateberkopec) of ***speedshop*** (who?), a Rails performance consultancy.

Summary: Have you ever wondered how the heck Ruby's GC works? Let's see what we can learn by reading some of the statistics it provides us in the GC.stat hash. *(1560 words/8 minutes)*

Most Ruby programmers don't have any idea how garbage collection works in their runtime - what triggers it, how often it runs, and what is garbage collected and what isn't. That's not entirely a bad thing - garbage collection in dynamic languages like Ruby is usually pretty complex, and Ruby programmers are better off just focusing on writing code that matters for their users.

I call that an object leak.

But, occasionally, you get bitten by GC - either it's running too often or not enough, or your process is using tons of memory but you don't know why. Or maybe you're just curious about how GC works!

@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

Reservations

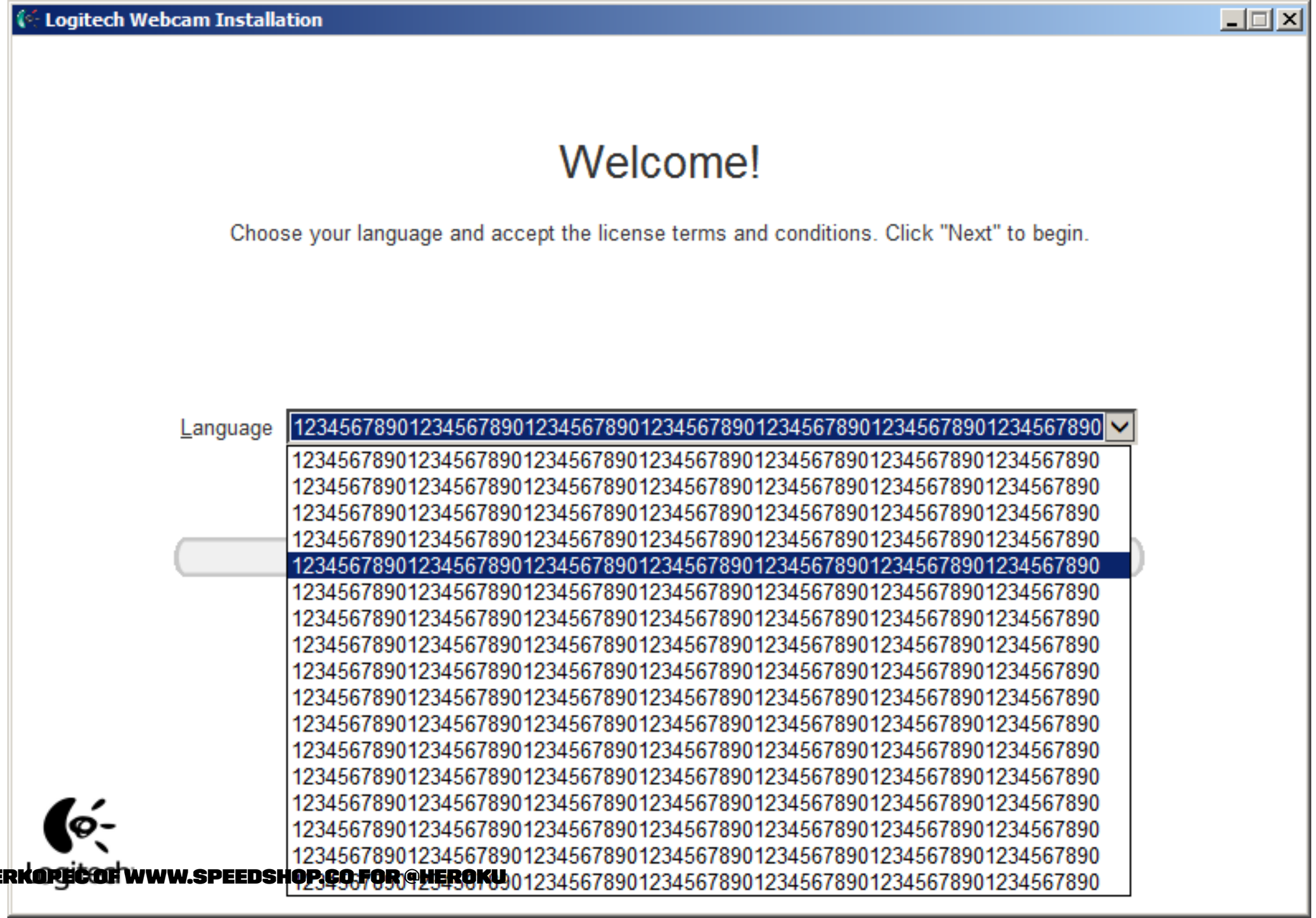
Date/ Time

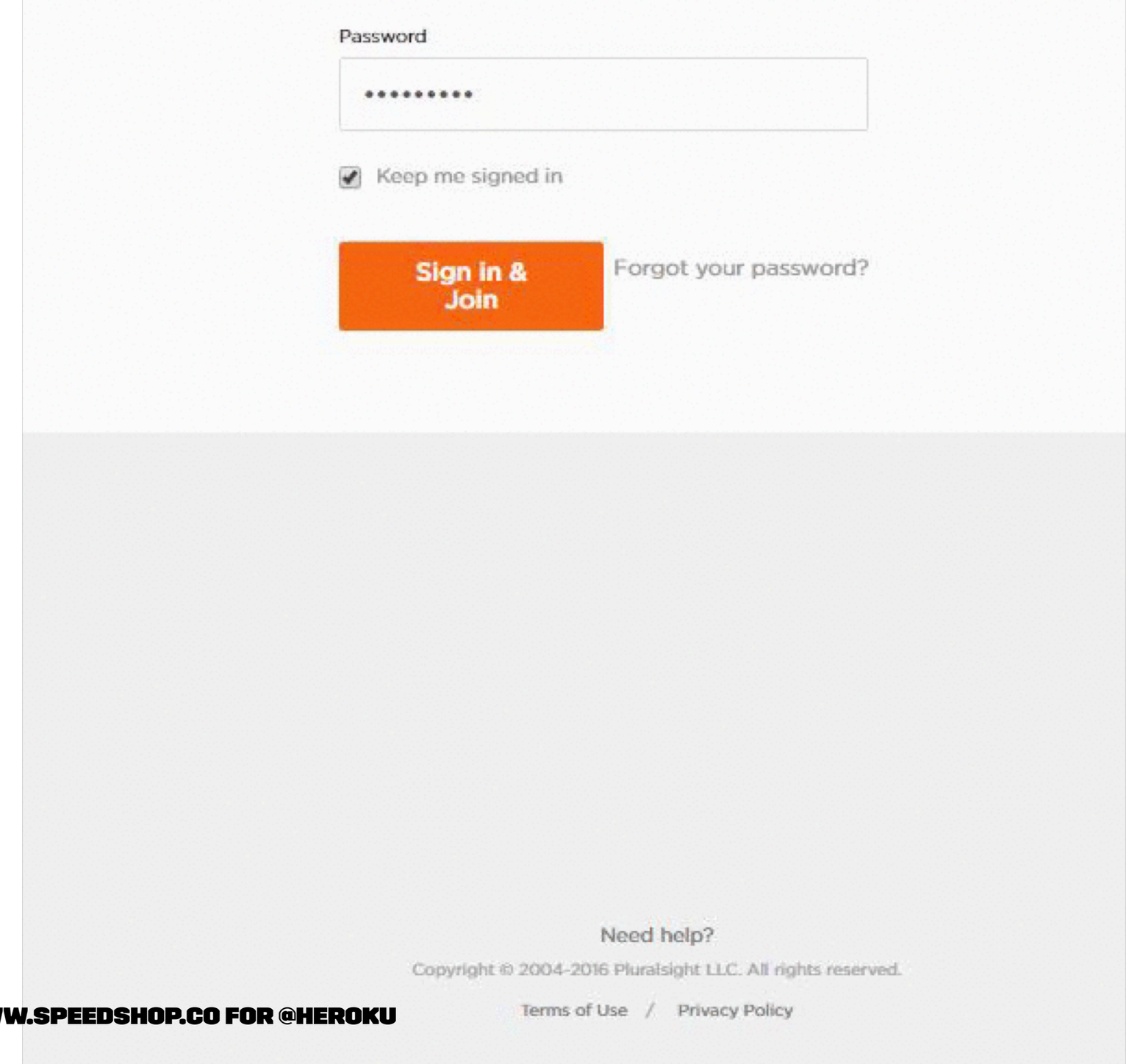
2.04.2017, 13:15

No. persons

8

Please enter a valid email address





@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

CHROME DEVTOOLS

@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

ASSUMPTIONS

USING CANARY

@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

TURN ON CHROME DEVTOOLS EXPERIMENTS

chrome://flags/

TURN OFF ANY EXTENSIONS

LOAD TIMES

THE THIN ORANGE LINE

“First Meaningful Paint is essentially the paint after which the biggest above-the-fold layout change has happened, and web fonts have loaded.”

1000 MILLISECONDS

100 MILLISECONDS FROM INPUT

NETWORK SPEEDS

» US: ~25% of users still on 3G

» US: ~1.4 Megabytes/Second

» Worldwide: ~625 KB/s

AVERAGE PAGE IS 2.5 MB

@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

LOAD PERFORMANCE IS (MOSTLY) NETWORK PERFORMANCE

CRITICAL RENDERING PATH

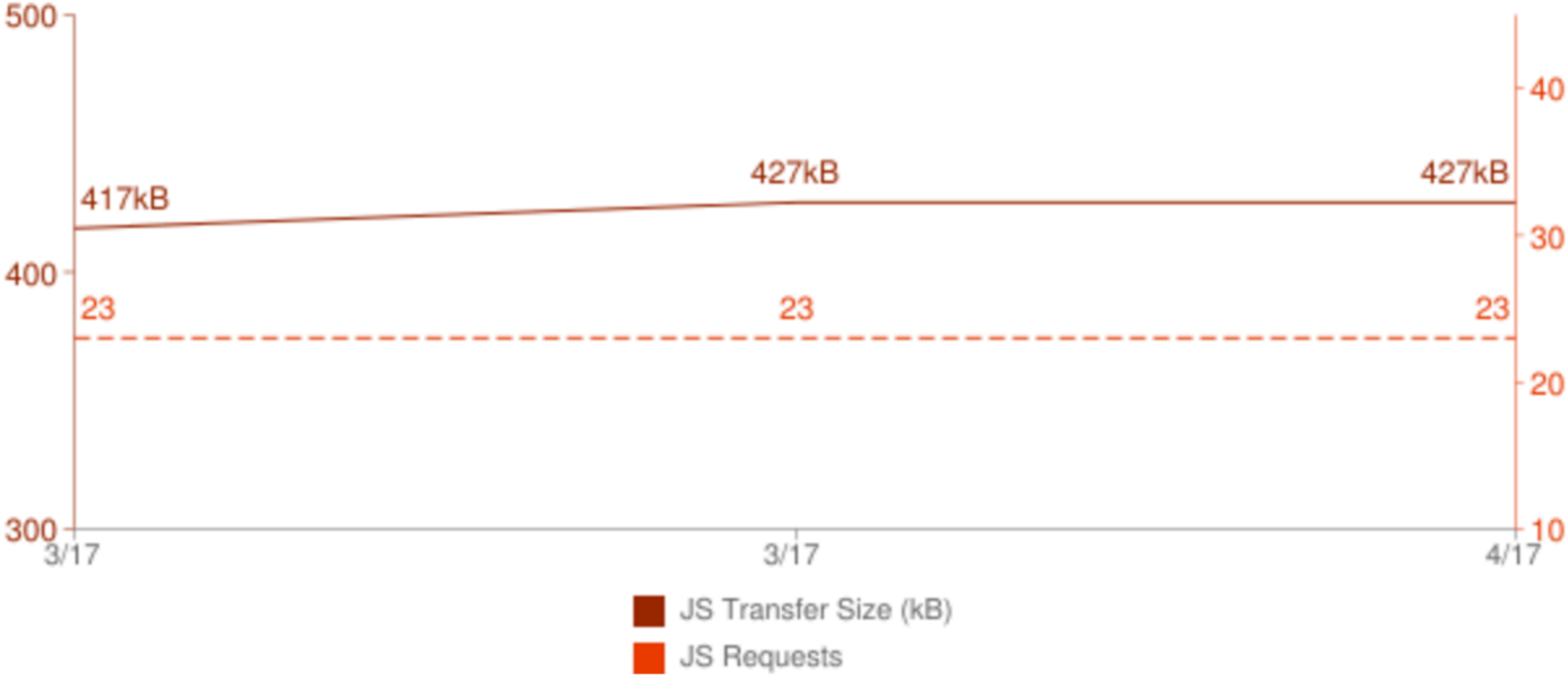
@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

1. Get HTML document
2. Start speculative preloading
3. Build DOM and CSSOM
4. Combine into render tree
5. Layout
6. Paint

JAVASCRIPT IS THE ENEMY

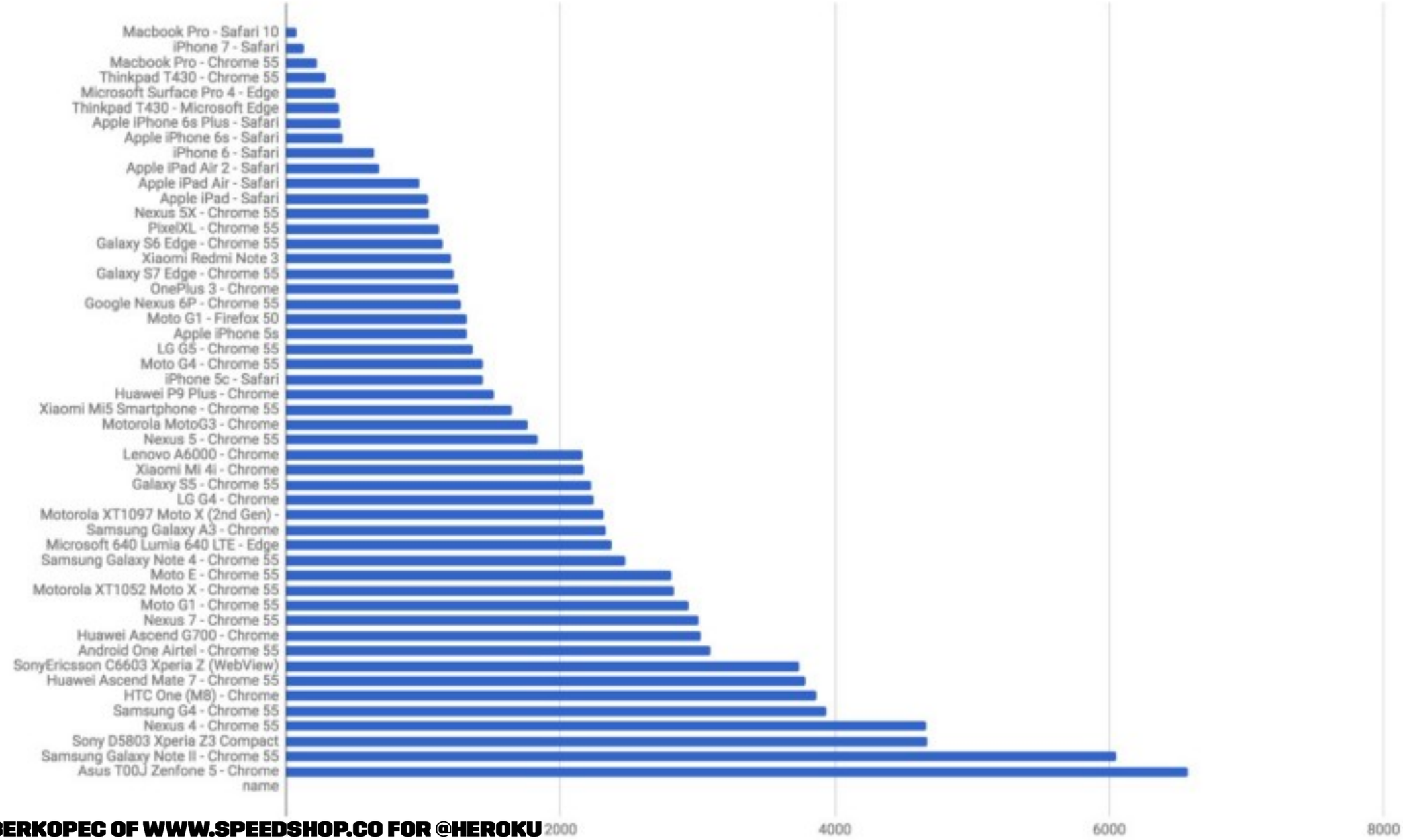
1. Wait for script to download
2. If all the CSS on this page hasn't downloaded yet, wait again.
3. Parse and execute this script

JS Transfer Size & JS Requests

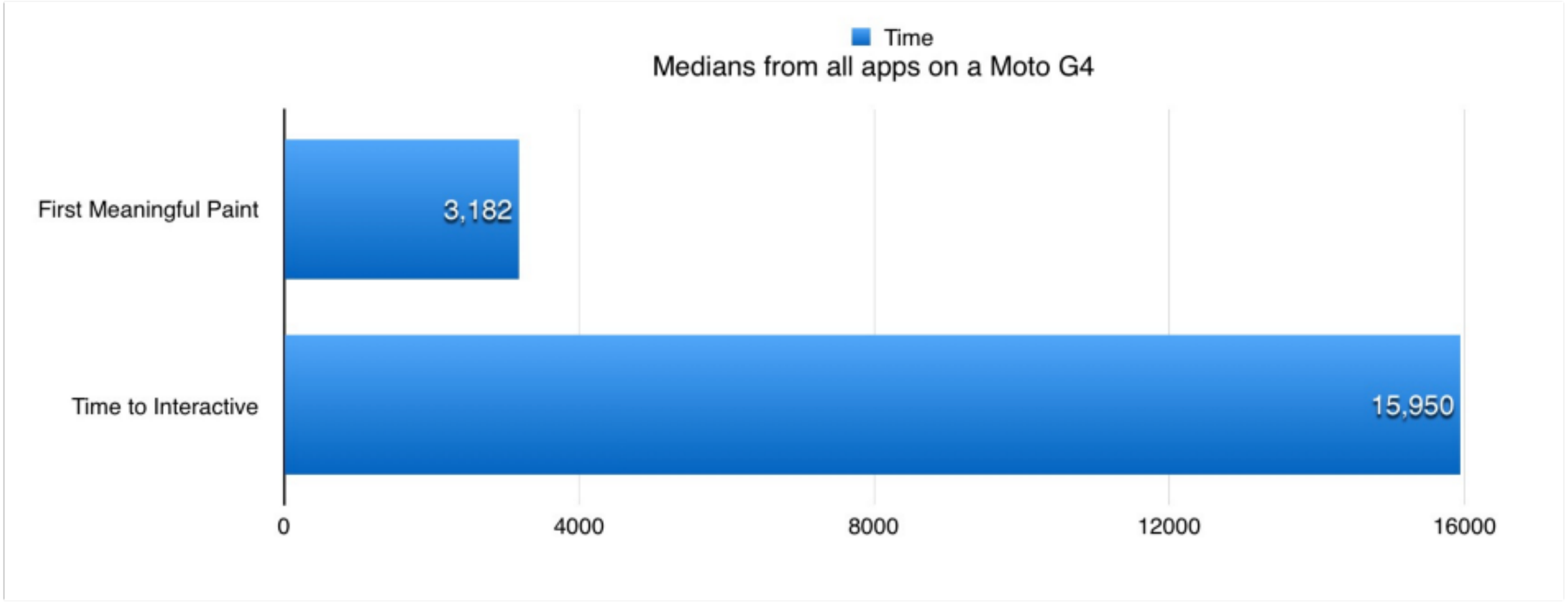


NOT JUST GZIPPED SIZE

1MB ~= 1 SEC JUST TO PARSE



@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU



JS apps became interactive in 16s on average mobile hardware over 3G

On desktop, most took 8 seconds to be fully usable on a cable connection.

EXERCISE 1: LET'S LOOK AT THE CRP OF A PAGE

Discourse

HOW TO HALVE LOAD TIMES WITH ONE WEIRD TRICK: NO JAVASCRIPT

@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

async

defer

LAST RESORT
BOTTOM OF THE PAGE

MAKING JS SMALLER

» Webpack

» Google's Closure Compiler

» Using smaller libraries

EXERCISE 2: IDENTIFYING JS BLOCKING

CODERWALL

@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

OPENSTREETMAP

INLINING CSS

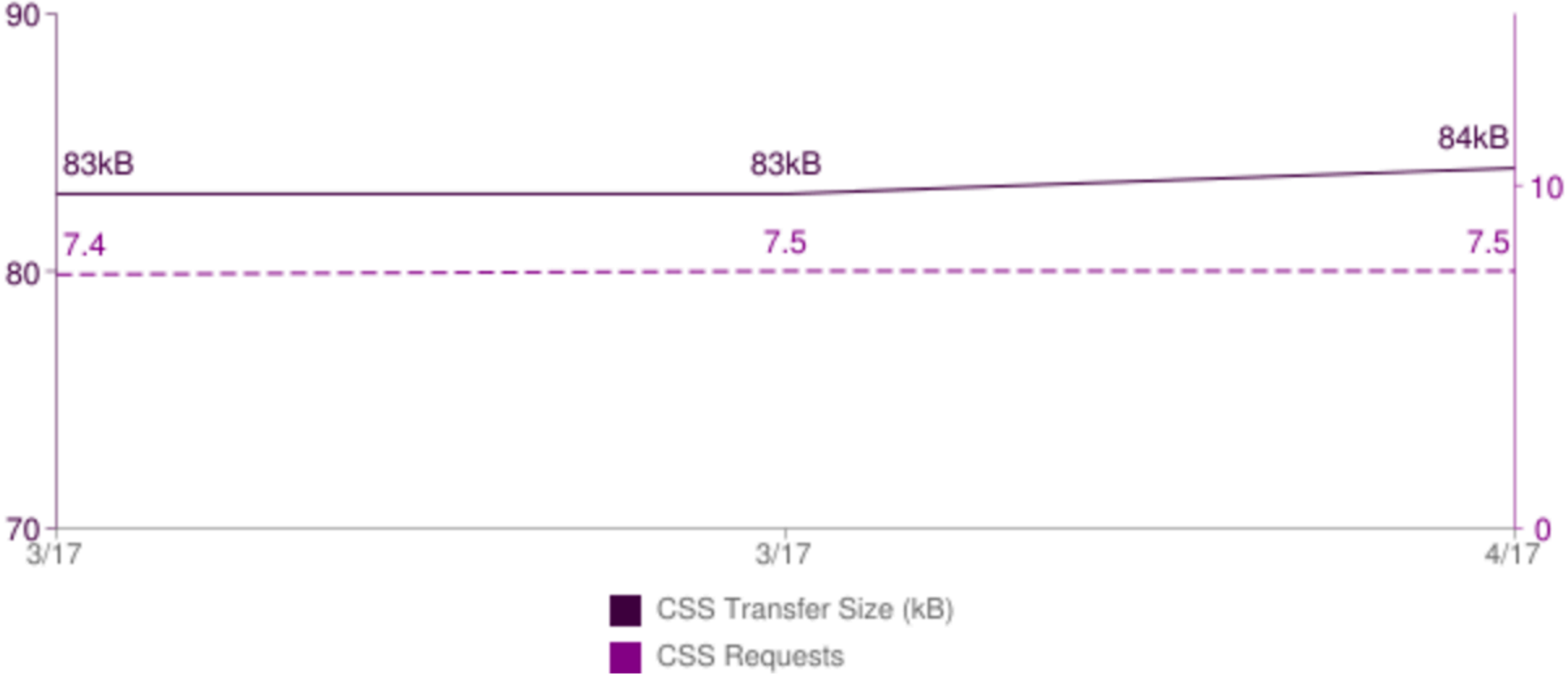
@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

CSS IN THE BODY

@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

Demo

CSS Transfer Size & CSS Requests



EXERCISE 3: LOOKING FOR BLOCKING CSS

HN

HTTP/2

@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

PARALLELIZING MANY RESOURCES

SIDEKIQ.ORG

@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

SERVER PUSH

LINK HEADERS

```
Link: </css/style.css>; rel=preload;
```

1. application.js and application.css
2. Things which cannot be cached
3. Next pages
4. Redirects

RACK_HTTP_PRELOAD

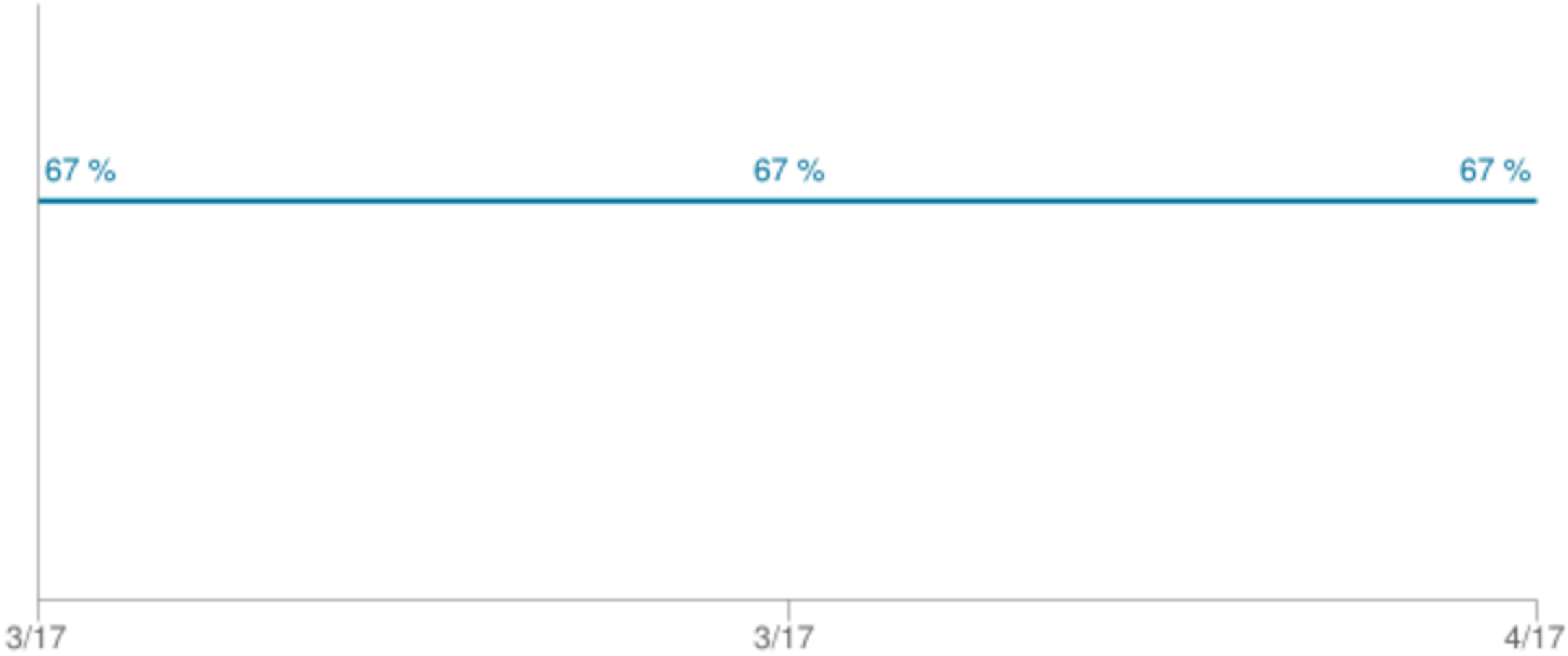
STREAMING RESPONSES

DOESN'T ACTUALLY WORK (LOL)

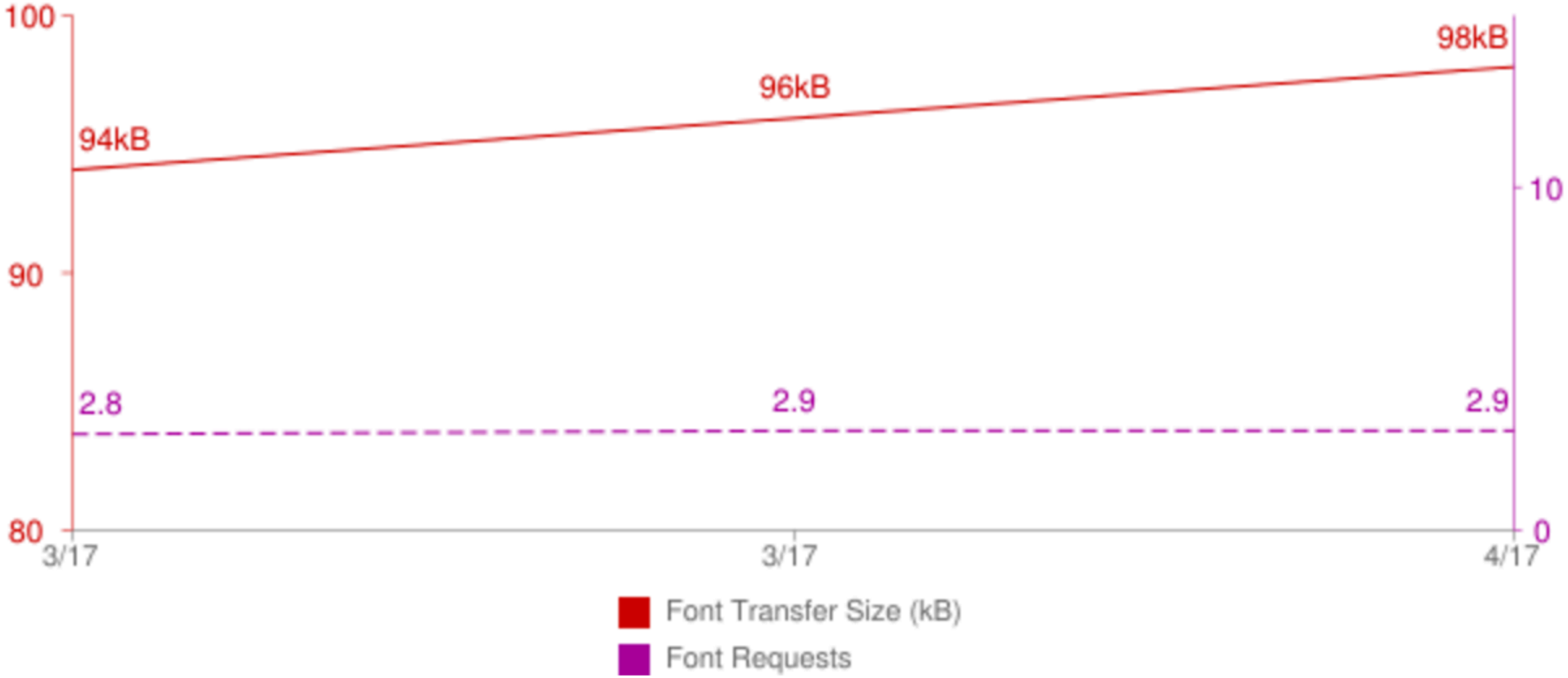
WEBFONTS

@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

Sites with Custom Fonts



Font Transfer Size & Font Requests



BASECAMP

RUBYGEMS.ORG

» Use Webfonts as spice, not filler.

» Don't use Typekit (blocking JS)

» Just use Google WebFonts

HTTP CACHING

@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

```
$(document).ready();
```

BIG DOMS

@NATEBERKOPEC OF WWW.SPEEDSHOP.CO FOR @HEROKU

100 Avoids an excessive DOM size**: 212 nodes** (target: 1,500 nodes)

▼ More information

Total DOM Nodes	DOM Depth	Maximum Children
212	11	37
target: < 1,500 nodes	*target: < 32*	*target: < 60 nodes*

IMAGE OPTIMIZATION

1. Find blocking JavaScript
2. Find blocking CSS and WebFonts
3. Optimize images
4. Add server push and/or HTTP/2
5. Analyze startup performance (document.ready)
6. Check HTTP caching