Webpagetestから始める継続的パフォーマンス改善

ページロードタイム編 ⏳

自己紹介

- Name : **azu**
- Twitter : @azu_re
- Website: Web scratch, JSer.info
- Create: textlint, Almin

アジェンダ

- パフォーマンス改善は指標を決めて行わないと迷子になる
- パフォーマンス改善を行うには継続的な計測を行う
  - 今回はページロードについて、ランタイムは範囲外
- パフォーマンス改善のアプローチ
- 継続的なパフォーマンス計測とリグレッションの検知

「パフォーマンス」の定義

- 今回は「ページロードタイム」についての「パフォーマンス」の話
  - 「ページロード」 = URLにアクセスしてページが表示されるまで
- 「ランタイム」の話ではありません
  - 「ランタイム」= ページ表示後のUI操作やアニメーションなど
- 「パフォーマンス」と言った場合は大体「ページロードタイムにおけるパフォーマンス」と解釈

「ページロードのパフォーマンスが良い」の定義

- ページが表示されるまでに時間を早くするのが目的
- 真っ白のページじゃなくて意味あるコンテンツをユーザー(ブラウザ)に早く表示させる
  - => 「ページロードのパフォーマンス」を良くする
- onLoad is not 表示速度
  - onLoadの速さは表示速度とは直接関係ない
  - 間接的にはonLoadが早いのはいいことだけど、これが目的ではない


0.0s
0.3s
0.8s
1.2s
1.5s
Optimized (progressive) rendering
Unoptimized rendering
lorem ipsum lore
lorem ipsum arhusr progressive render is awesome!

継続的に改善するために、計測も継続的に行う

- 継続的にパフォーマンスを改善するには、パフォーマンス計測も継続的に
- パフォーマンスはいつのまにかリグレッションが起きやすい
- 人間は0.1秒の変化は勝手に補完するので変化に鈍感
- パフォーマンスを数値化して問題に気づけるようにする
- => 定期的に計測を行うためにも自動化が必要

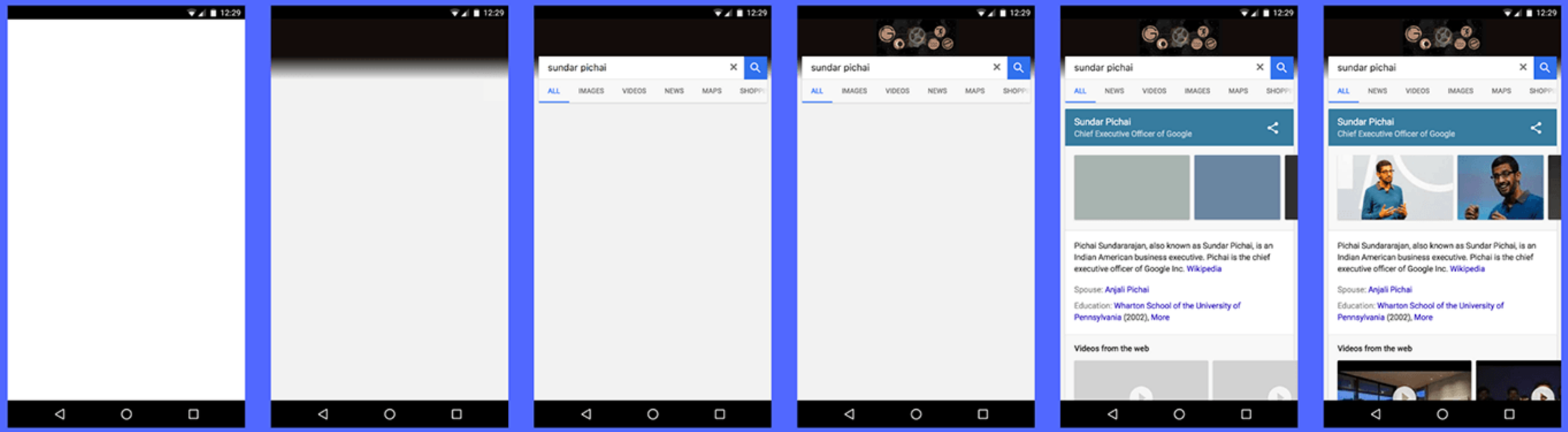
なぜ計測を自動化するの？

- 継続的に改善するには、**同じ条件で計測した比較できる値**が必要
  - 仮想環境で同じ条件を作って定期的に計測を行う
- 手元のDevToolsで図ることもできる
  - ただし、高性能な開発者端末での結果の1つに過ぎない
  - ただし、他の作業を並行しがちなので結果が安定しない
  - 同じ条件で計測した値じゃないので比較しにくい

改善には指標が必要

- 改善する方向 = 指標（メトリクス）がないと迷子になる
- アプリケーションにはパフォーマンスだけじゃなくて様々に要素が混在する
- 他の機能を改善したつもりがパフォーマンスが低下することもある
- 感覚ではなくて数値として改善を確認する手段が必要

ページロードに関する指標



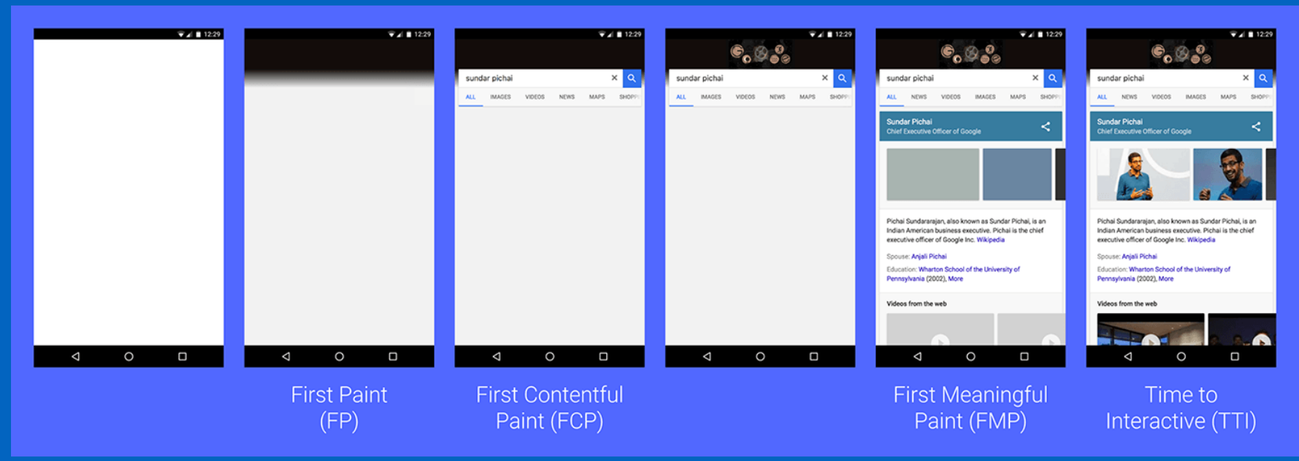
First Paint (FP)

First Contentful Paint (FCP)

First Meaningful Paint (FMP)

Time to Interactive (TTI)

- First Paint(FP)
  - 最初の1pxが描画されるまでの時間
- First Contentful Paint(FCP)
  - コンテンツっぽいものが描画されるまでの時間
- First Meaningful Paint(FMP)
  - 意味のあるものが描画されるまでの時間
- Time to Interactive(TTI)
  - ユーザーの操作に反応できるまでにかかった時間



ページロードに関する指標

- ブラウザが提供するイベントの値を指標として利用できる
- Speed Indexも定番
  - ファーストビューが見えるまでにかかった時間
- User-centric Performance Metrics | Web Fundamentals | Google Developers
- Web クライアントサイドのパフォーマンスメトリクス – Speed Index、 Paint Timing、 TTI etc... ::ハブろぐ

アプリケーションによって使うべき指標は異なる

- アプリよってページロードの仕組みも違うため指標の選択肢も異なる
  - 例) CSR(クライアントサイトレンダリング): 真っ白な時間が長いのでFMPを指標に
  - 例) SSR(サーバサイドレンダリング): 真っ白な時間は短いが、操作できるまで(JSを読む)の時間を測るためTTIを指標に
- コンテンツによって**意味のある表示**(FMP)が本当に意味のあるものかは別
  - `performance.mark` APIを使いプロダクトごとに指標を作るのがベスト
- これらの値をパフォーマンス計測サービスで記録する

どうやって計測するか

どうやって指標となる値を記録するか

合成モニタリングとリアルユーザーモニタリング

- 合成モニタリング(Synthetic Monitoring)
  - 計測用の仮想環境などから、同じ条件で定期的に繰り返し計測
  - 環境が同じ条件なら計測結果のゆらぎが少ない
- リアルユーザーモニタリング(RUM)
  - ユーザーがページを開いたときに記録を取ってログとして送る
  - ユーザー環境はバラバラなので計測結果のゆらぎは大きい
  - 一方でリアルなデータなので、KPIと値を組み合わせて分析といった用途に使える

表示速度と指標

- 表示速度を改善したいので、指標は合成モニタリングで集める
- 合成モニタリングなら同じ環境で継続的に計測できる
- => 値同士を比較して差分を見られる
- => パフォーマンスは相対で見たほうがわかりやすい

合成モニタリングを行えるサービス

Priceは2018年8月時の参考値、プランによりPriceは異なります。

- WebPagetest(無料 - 制限200チェック/day)
  - OSSなのでSelf Hoistingもできる
- SpeedCurve(有料 - $20+/month)
- Calibre(有料 – $29+/month)
- New Relic Synthetics(有料 – $69/month)
- CatchPoint(有料 – $899/month)

サービスによって計測できるものは違う（読まなくていい）

- サービスによって計測する対象、仕方が違うので比較はしにくい
- Webpagetestなどのデータは丸め込まれたデータなので分析用途ではない(Real Dataとしては捉えない)
- スコアは人間にとってわかりやすい値であるだけど、実データとは異なる
- WebPagetestは計測するだけで継続的に変化を見るダッシュボードはない
  - 計測したデータは1ヶ月だけ保持される
- SpeedCurve、Calibreは計測 + ダッシュボード + 通知など
  - 計測は大体AWS EC2のリージョン/マシン
  - https://www.webpagetest.org/getLocations.php?f=html&k=A
  - http://support.speedcurve.com/synthetic-settings/test-agent-locationsregions
  - https://calibreapp.com/docs/site/agent-locations
- CatchPointはISPとかネットワーク周りが詳細にとれる
  - Synthetic Monitoring を活用したグローバルサービスのネットワークレイテンシの測定と改善 - クックパッド開発者ブログ
  - 統計的に扱いやすいデータが集めやすい
- ユーザー行動と紐づけて分析するならRUMも必要

システムは複雑系なので、計測したもので分析できるとは限らない

A system is never the sum of its parts. It is the product of the interactions of itsparts.

— *Dr. Russel Ackof*

- 確かに色々な値を取れるけど、システムのすべての値が取れるとは限らないということは覚えておく
- 実験計画法などを参照


WebPagetest - Website Perfor
保護されていない通信 | www.webpagetest.org
Login | Register | Login with Google
WEBPAGETEST
100 Gbps $0.13/Mbps - IP Transit For Your Network
Global Internet Backbone 21,000+ BGP sessions, 7200+ networks, 185+ IXPs Join Us he.net
HOME
TEST HISTORY
FORUMS
DOCUMENTATION
ABOUT
Test a website's performance
Advanced Testing
Simple Testing
Visual Comparison
Traceroute
START TEST
Test Location
Dulles, VA USA (Desktop, Android, iOS)
Select from Map
Browser
Advanced Settings
3 runs, First View only, Cable connection, private
WebPagetest
Run a free website speed test from multiple locations around the globe using real browsers (IE and Chrome) and at real consumer connection speeds. You can run simple tests or perform advanced testing including multi-step transactions, video capture, content blocking and much more. Your results will provide rich diagnostic information including resource loading waterfall charts, Page Speed optimization checks and suggestions for improvements.
If you have any performance/optimization questions you should visit the Forums where industry experts regularly discuss Web Performance Optimization.
Recent Industry Blog Posts
Refresh Stale DNS Records on 1.1.1.1
Internet Native Applications
Integrating redirection.io with Cloudflare Workers
How Cloudflare protects customers from cache poisoning
Edge-Side-Includes with Cloudflare Workers
more...
Recent Discussions
Browser Cache and WebPagetest
Cache headers detection
CSSOM Ready metric
child host override
Chrome extension: 4 Website Performance Tests with one click
more...
WebPagetest Partners
プライバシー - 利用規約

WebPagetest

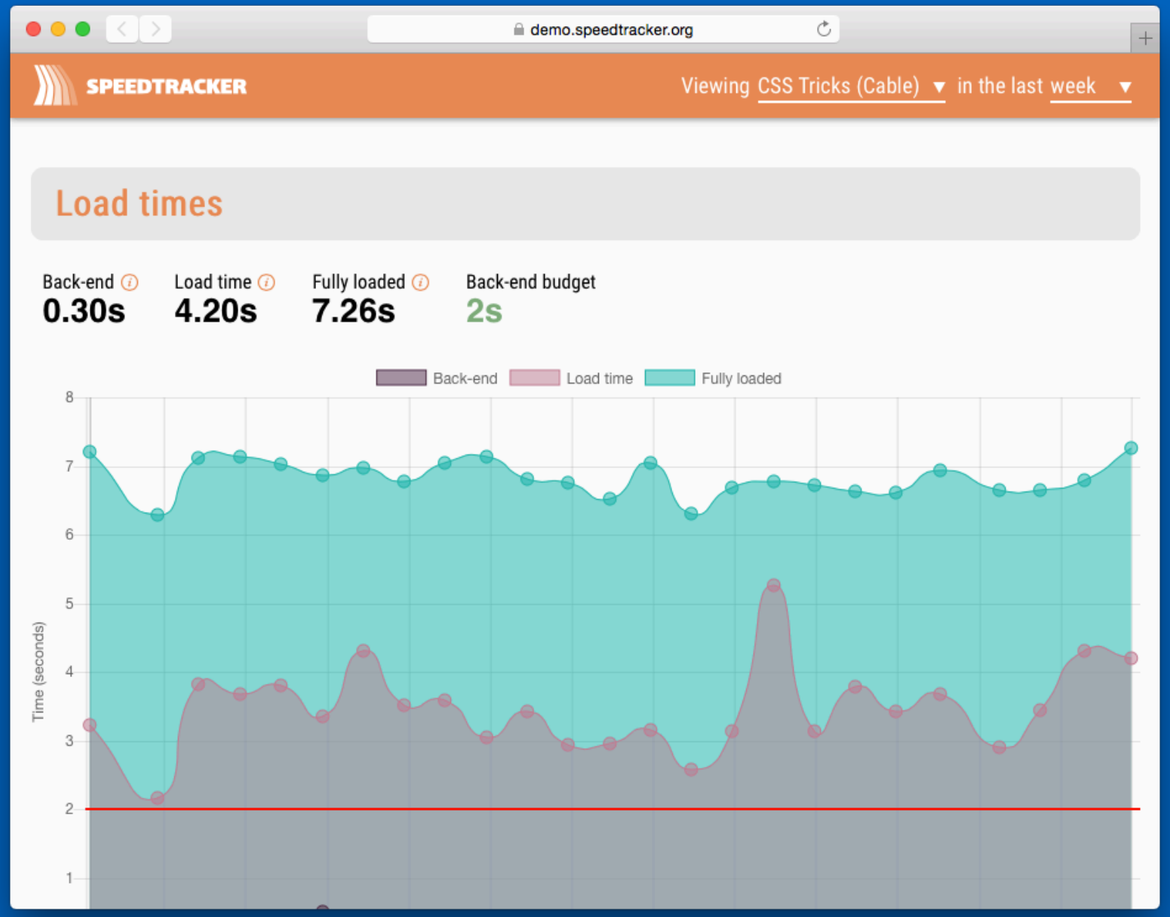
- WebPagetestは一度計測してその結果を返すだけサービス
  - 無料で200回/日利用でき、OSSなのでSelf Hostもできる
- WebPagetest自体には定期的に計測したりそのデータをグループ管理する仕組みはない
  - これを補ってくれるサービスやツールが多く存在する

WebPagetest単体では足りないとところ

- WebPagetestで継続的に計測するには次のことが必要
  - 計測頻度や計測を行うタイミングの管理
  - 計測結果を保存
  - 結果をグラフなどにして可視化できるダッシュボード
  - 結果をもとにアラートを行う

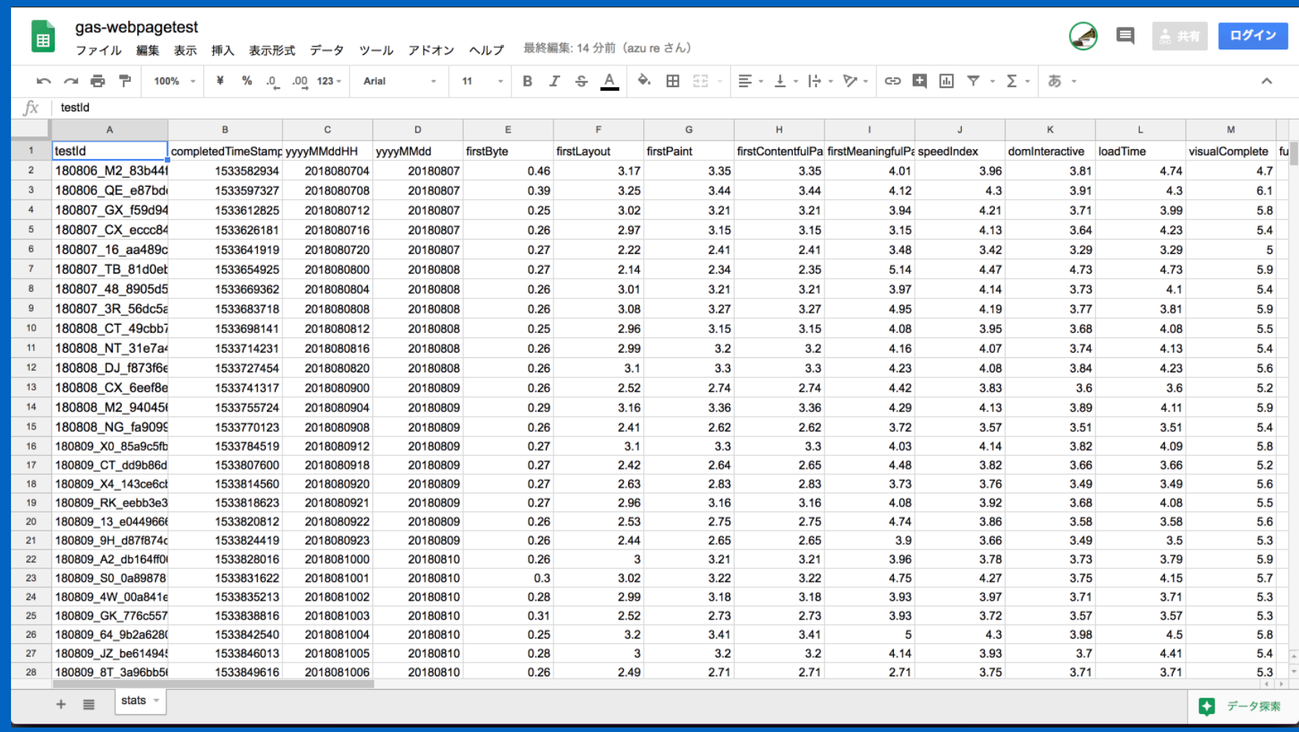
SpeedTracker

- WebPagetestで計測して、その結果をGitHubリポジトリに保存する
  - GitHubアカウントだけが必要
- 計測の柔軟性や計測回数は少ない(2回/1日)
  - 具体的には認証があるサイトなどは扱えない
- 個人で公開してるサイトを計測するにはお手軽



gas-webpagetest

- WebPagetestで計測しその結果をGoogle SpreadSheetに記録するGoogle Apps Scripts
  - clasp + TypeScriptで書き直された
  - Google Apps ScriptsはCronや、Webサーバ、SpreadSheetの操作などいろいろできる
- コードで書かれてるのでWebPagetestでできる範囲のことは大抵できる



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	testId	completedTimeStamp	yyyyMMddHH	yyyyMMdd	firstByte	firstLayout	firstPaint	firstContentfulPa	firstMeaningfulPa	speedIndex	domInteractive	loadTime	visualComplete
2	180806_M2_83b44f	1533582934	2018080704	20180807	0.46	3.17	3.35	3.35	4.01	3.96	3.81	4.74	4.7
3	180806_QE_e87bd	1533597327	2018080708	20180807	0.39	3.25	3.44	3.44	4.12	4.3	3.91	4.3	6.1
4	180807_GX_f59d94	1533612825	2018080712	20180807	0.25	3.02	3.21	3.21	3.94	4.21	3.71	3.99	5.8
5	180807_CX_eccc84	1533626181	2018080716	20180807	0.26	2.97	3.15	3.15	3.15	4.13	3.64	4.23	5.4
6	180807_16_aa489c	1533641919	2018080720	20180807	0.27	2.22	2.41	2.41	3.48	3.42	3.29	3.29	5
7	180807_TB_81d0eb	1533654925	2018080800	20180808	0.27	2.14	2.34	2.35	5.14	4.47	4.73	4.73	5.9
8	180807_48_8905d5	1533669362	2018080804	20180808	0.26	3.01	3.21	3.21	3.97	4.14	3.73	4.1	5.4
9	180807_3R_56dc5a	1533683718	2018080808	20180808	0.26	3.08	3.27	3.27	4.95	4.19	3.77	3.81	5.9
10	180808_CT_49cbb7	1533698141	2018080812	20180808	0.25	2.96	3.15	3.15	4.08	3.95	3.68	4.08	5.5
11	180808_NT_31e7a4	1533714231	2018080816	20180808	0.26	2.99	3.2	3.2	4.16	4.07	3.74	4.13	5.4
12	180808_DJ_f873f6e	1533727454	2018080820	20180808	0.26	3.1	3.3	3.3	4.23	4.08	3.84	4.23	5.6
13	180808_CX_6eef8e	1533741317	2018080900	20180809	0.26	2.52	2.74	2.74	4.42	3.83	3.6	3.6	5.2
14	180808_M2_94045f	1533755724	2018080904	20180809	0.29	3.16	3.36	3.36	4.29	4.13	3.89	4.11	5.9
15	180808_NG_fa9095	1533770123	2018080908	20180809	0.26	2.41	2.62	2.62	3.72	3.57	3.51	3.51	5.4
16	180809_X0_85a9c5fb	1533784519	2018080912	20180809	0.27	3.1	3.3	3.3	4.03	4.14	3.82	4.09	5.8
17	180809_CT_dd9b86d	1533807600	2018080918	20180809	0.27	2.42	2.64	2.65	4.48	3.82	3.66	3.66	5.2
18	180809_X4_143ce6cb	1533814560	2018080920	20180809	0.27	2.63	2.83	2.83	3.73	3.76	3.49	3.49	5.6
19	180809_RK_eebb3e3	1533818623	2018080921	20180809	0.27	2.96	3.16	3.16	4.08	3.92	3.68	4.08	5.5
20	180809_13_e044966f	1533820812	2018080922	20180809	0.26	2.53	2.75	2.75	4.74	3.86	3.58	3.58	5.6
21	180809_9H_d87f874c	1533824419	2018080923	20180809	0.26	2.44	2.65	2.65	3.9	3.66	3.49	3.5	5.3
22	180809_A2_db164ff0	1533828016	2018081000	20180810	0.26	3	3.21	3.21	3.96	3.78	3.73	3.79	5.9
23	180809_S0_0a89878	1533831622	2018081001	20180810	0.3	3.02	3.22	3.22	4.75	4.27	3.75	4.15	5.7
24	180809_4W_00a841e	1533835213	2018081002	20180810	0.28	2.99	3.18	3.18	3.93	3.97	3.71	3.71	5.3
25	180809_GK_776c557	1533838816	2018081003	20180810	0.31	2.52	2.73	2.73	3.93	3.72	3.57	3.57	5.3
26	180809_64_9b2a628(	1533842540	2018081004	20180810	0.25	3.2	3.41	3.41	5	4.3	3.98	4.5	5.8
27	180809_JZ_be61494	1533846013	2018081005	20180810	0.28	3	3.2	3.2	4.14	3.93	3.7	4.41	5.4
28	180809_8T_3a96bb5	1533849616	2018081006	20180810	0.26	2.49	2.71	2.71	2.71	3.75	3.71	3.71	5.3

gas-webpagetest + Google Data Studio

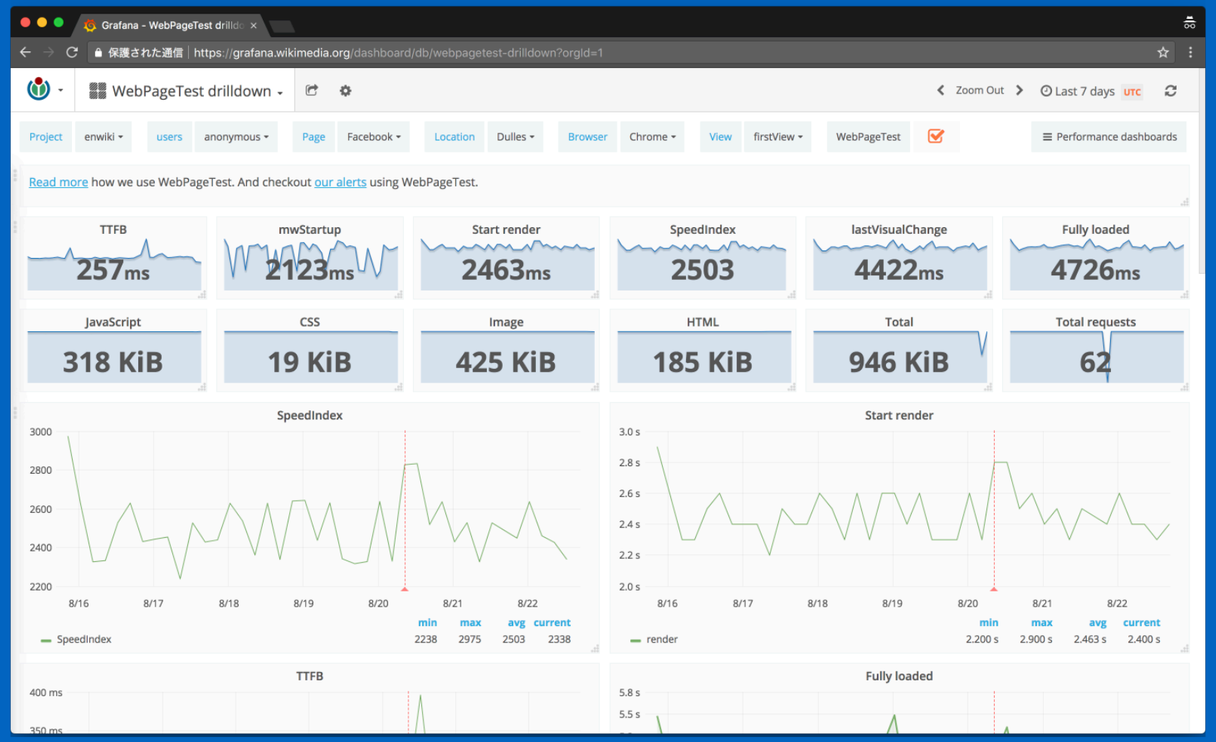
- Google Data Studioは任意のリソースをもとにしたダッシュボードを作れるサービス
  - いわゆるBI(Business Intelligence)ツールで無料で利用できる
  - Google SpreadSheet、MySQL、GitHubなどをデータソースにできる
  - 最近複数のデータリソースを混ぜることもできるようになった
- Spreadsheetに蓄積したデータのビジュアライズをGoogle Data Studioで行う

SpeedCurve

- SpeedCurveの合成モニタリングはWebPagetestをベースにしている
  - SpeedCurve | Synthetic: WebPageTestでWebPagetest自体には計測の追跡や分析がないのでSpeedCurveが登場したという話が書かれている
  - WebPagetestのいいUIという感じの有償サービス
- RUMも計測できるサービスもやってる

Sitespeed.io

- 自前でSpeedCurveみたいなものを作るツールキット
  - Docker(Private HostingのWebPagetest含む) + AWSで運用
  - Web Performance Dashboards with sitespeed.io
- WikiPedia(WikiMedia)の人が開発している
  - Performance/WebPageTest - Wikitech
  - https://grafana.wikimedia.org/dashboard/db/webpagetest



好きなものを使ってとりあえず計測を始める

- まずは計測を始めよう
  - gas-webpagetest + 好きなSaaSの2種類で計測を開始しよう
- 1-2週間ぐらい計測をしながら現状を把握しよう
  - 「ブラウザの開発者ツールのパフォーマンス計測を自動化」から初めてみよう
- 多くの計測データにはバイアスがかかっているので、分析には気をつけよう
  - どんなに優れたモニタリングツールではすべての因子を見ることができない
- パフォーマンス計測を継続することは偉いこと
  - パフォーマンス計測自体も徐々に改善していけばいい

計測してどうする?

改善する

改善するために計測する

- ただ単に計測したデータを貯めてるだけではしょぼい死活監視にしかならない
- 貯めたデータはパフォーマンス改善の指標に使える
- ダッシュボードから問題を見つけることは難しい
  - Lighthouseとかを使ったほうが具体的な指示がでる
  - その指示とデータを照らし合わせてみる
- 比較対象をもつことが大事
  - 似た機能を持つサイトと比較してみる

まずは大きな改善から始めよう

- 局所的な改善はページロードには小さな影響しかない
  - 遅いサイトは秒単位の遅い問題を持っている
  - まずは秒単位の問題を解決してから、ミリ秒単位の問題を解決していく
- ウェブページのロードはウォータフォール
  - 一つでも重たいものがあるとそこで詰まる = クリティカル
  - 重たいものを減らしていく = パフォーマンス改善
  - 既にあるものを早くするのではなく、ブロッカーを取り除く

クリティカルレンダリングパスの改善

- ページロードから表示までの一連の流れ = クリティカルレンダリングパス
  - これを改善することがページロードタイムの改善につながる
- クリティカルパスの改善の考え方
  - ファイルサイズを小さくする
  - リクエスト数を減らす
  - 待機時間が長いリソース取得の改善

改善のパターン（読まなくていい）

- ファイルサイズを小さくする
  - The Cost Of JavaScript In 2018 – Addy Osmani – Medium
  - ページロードに不要なリソースの分解、削除
- リクエスト数を減らす
  - The Critical Request - Speaker Deck
  - HTTP/2だと並列リクエストができるのでアプリケーション次第
  - リクエストごとに表示がプログレッシブに進むならキャッシュを考えて分けたほうがいい
  - 結局全部取ってからじゃないないと何もできないなら細かく刻む必要はあまりない
  - リソース間の依存関係を把握する必要があるということ
  - リソースの依存関係をみてボトルネックを発見し解決する
  - LightHouseのクリティカルリクエストチェーンを減らす
- 待機時間が長いリソースの改善
  - リソースを返すのに時間がかかってる配信するサーバ側の問題
  - サードパーティスクリプトなど直接改善できないものは、遅延ロードさせるなどクリティカルパスから外す
  - Resource Hintsを使って先に読んでおく
- メインコンテンツに必要ないリソースは後で読む
  - サードパーティスクリプト
  - script async属性
  - lazy load

改善のセオリー

- 大きなボトルネックのパフォーマンス改善にはある程度セオリーがある
  - HTTP/2みたいにセオリーが変わる場合もあるけど、その場合も継続的な計測が役立つ
- 以下を読んで
  - Make the Web Faster | Google Developers
  - 超速! Webページ速度改善ガイド (WEB+DB PRESS plus)
  - High Performance Browser Networking (O'Reilly)
  - Webフロントエンド ハイパフォーマンス チューニング｜技術評論社
  - Performance Calendar

細かくなっていく改善

何を改善するのかが難しい問題

- ダッシュボードを見ても何が問題がわからずに何もできない
  - Note: SpeedCurveはいろんなヒントを出してくれる
- 速くするのではなく遅くしないことの方が重要
  - ボトルネックを取り除いていく => パフォーマンス改善
  - と考えたほうが行動しやすい
- ボトルネックはいろいろなツールで見つけられる
  - How To Think About Speed Tools | Web Fundamentals | Google Developers

改善していく例

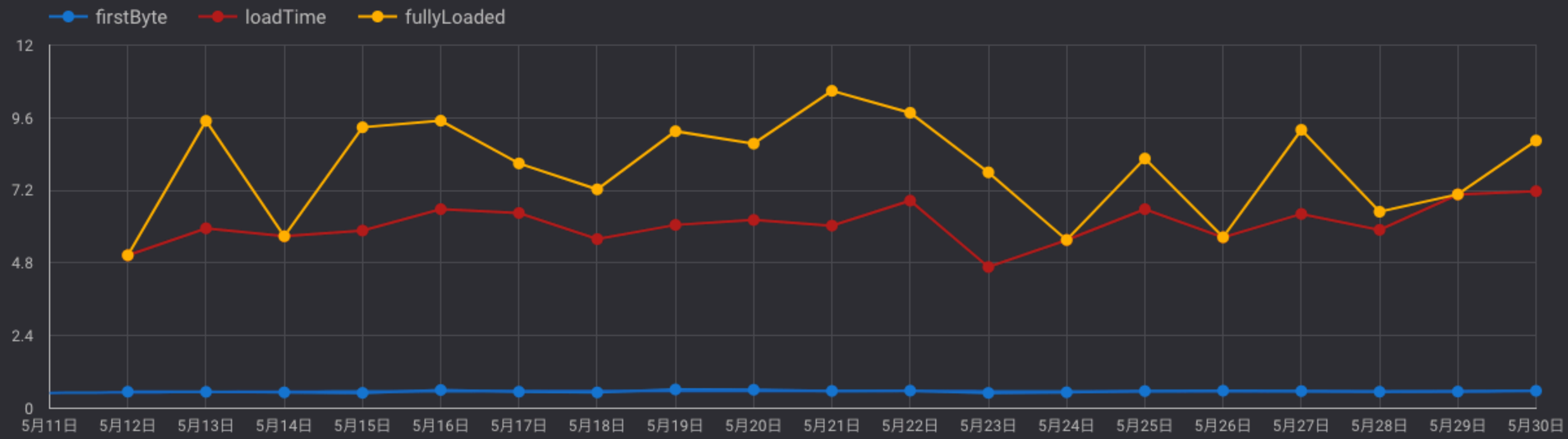
サイトA: 比較して問題を見つける例

- とりあえず計測してとりあえず可視化してみた
- ダッシュボードを見てもいまいち何が問題がわからない

PAGE LOAD TIMES

firstByte: The time it takes for the server to respond with the first byte of the response
loadTime: The time between the initial request and the browser load event
fullyLoaded: The time at which the page has fully finished loading content

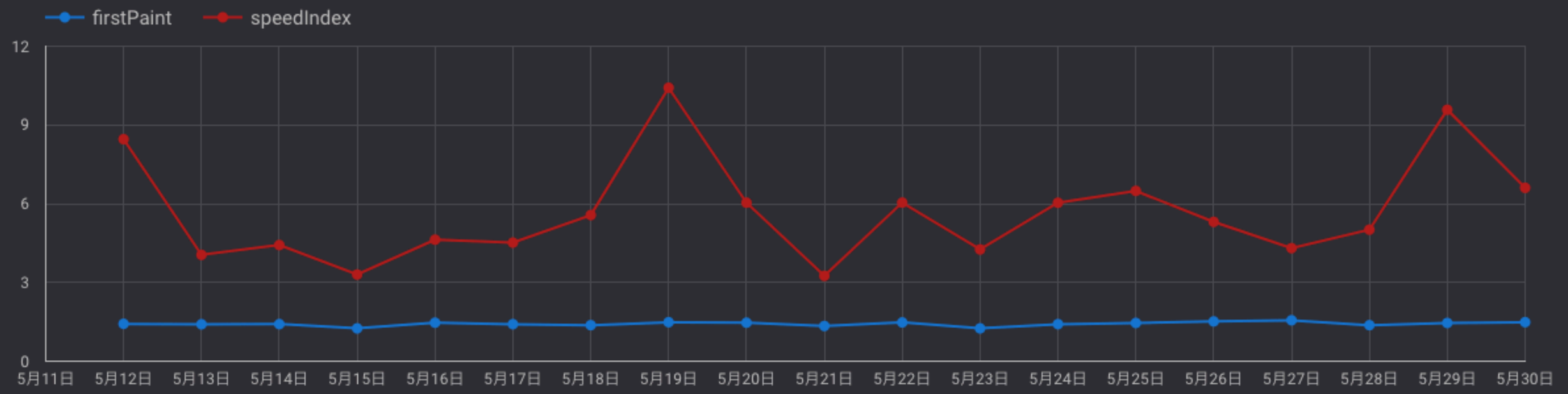
firstByte	loadTime	fullyLoaded
0.44	6.67	9.19
⬆ 5.2%	⬇ -12.5%	⬆ 3.8%



RENDERING TIMES

firstPaint: The time until the browser starts painting content to the screen
speedIndex: A custom metric introduced by WebPageTest to rate pages based on how quickly they are visually populated
visualComplete: The time it takes for the page to be fully visually populated

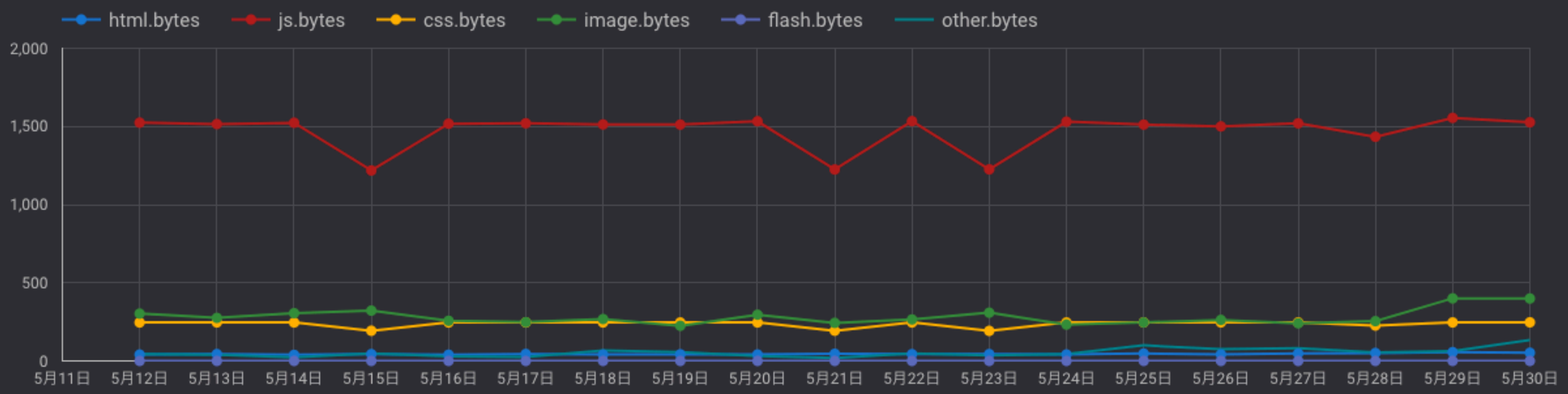
firstPaint	speedIndex
1.35	3.01
⬆ 4.6%	⬆ 8.9%



Content breakdown (size)

Content file size by type

html.bytes	js.bytes	css.bytes	image.bytes	font.bytes	other.bytes
83.78	1,497.35	148.6	607.85	20.6	10.95
↑ 0.6%	↓ -0.2%	↓ -0.3%	↓ -7.1%	↑ 0.0%	↑ 3.6%

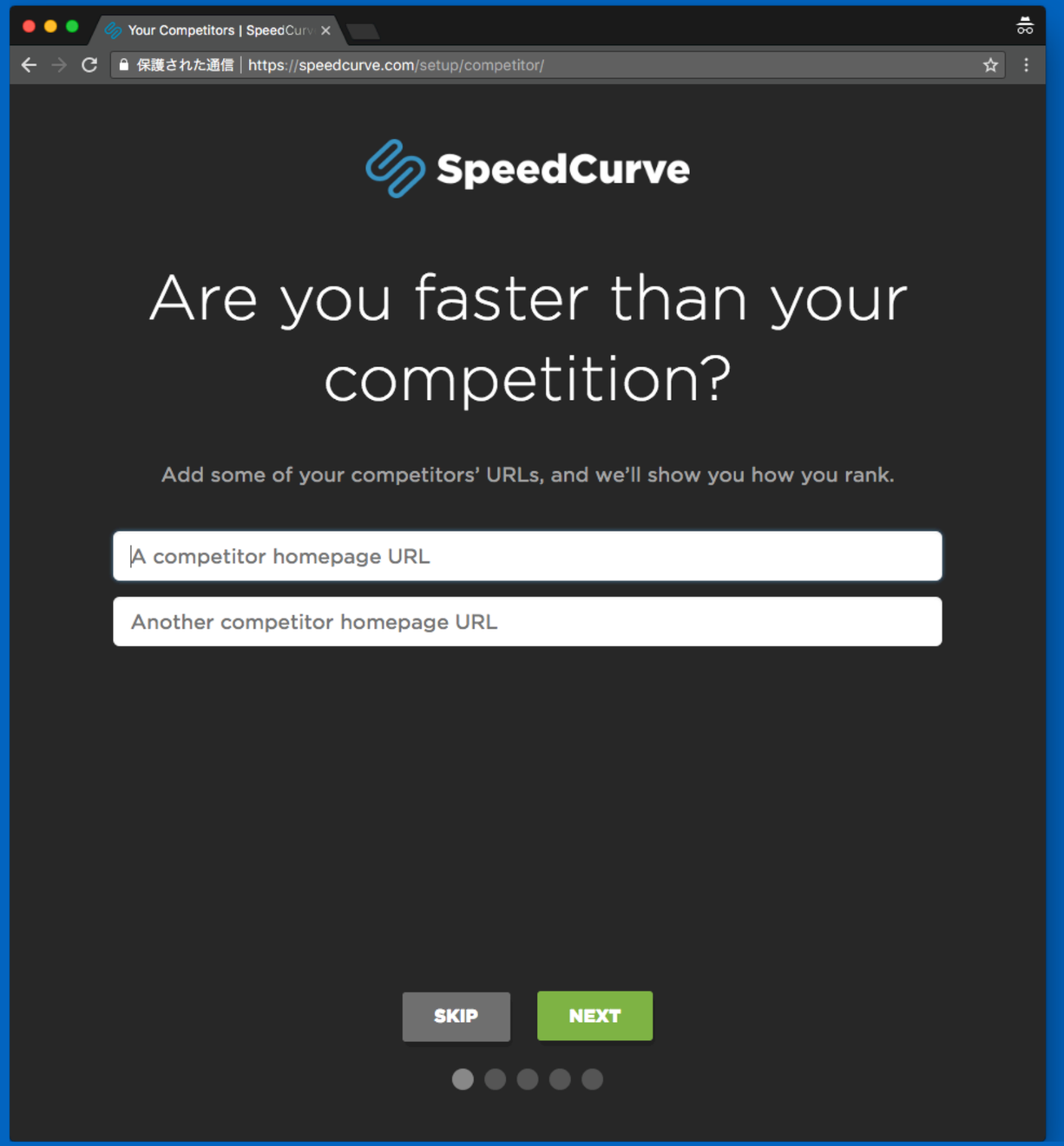


サイトA: 何が問題化を比較して見つける

- 比較対象があると問題が見つけやすい

Competitorと比較して計測

- WebPagetestでは、結果のtestId同士を比較できる
- SpeedCurveでは、登録する時に"比較対象のURL"を入力する



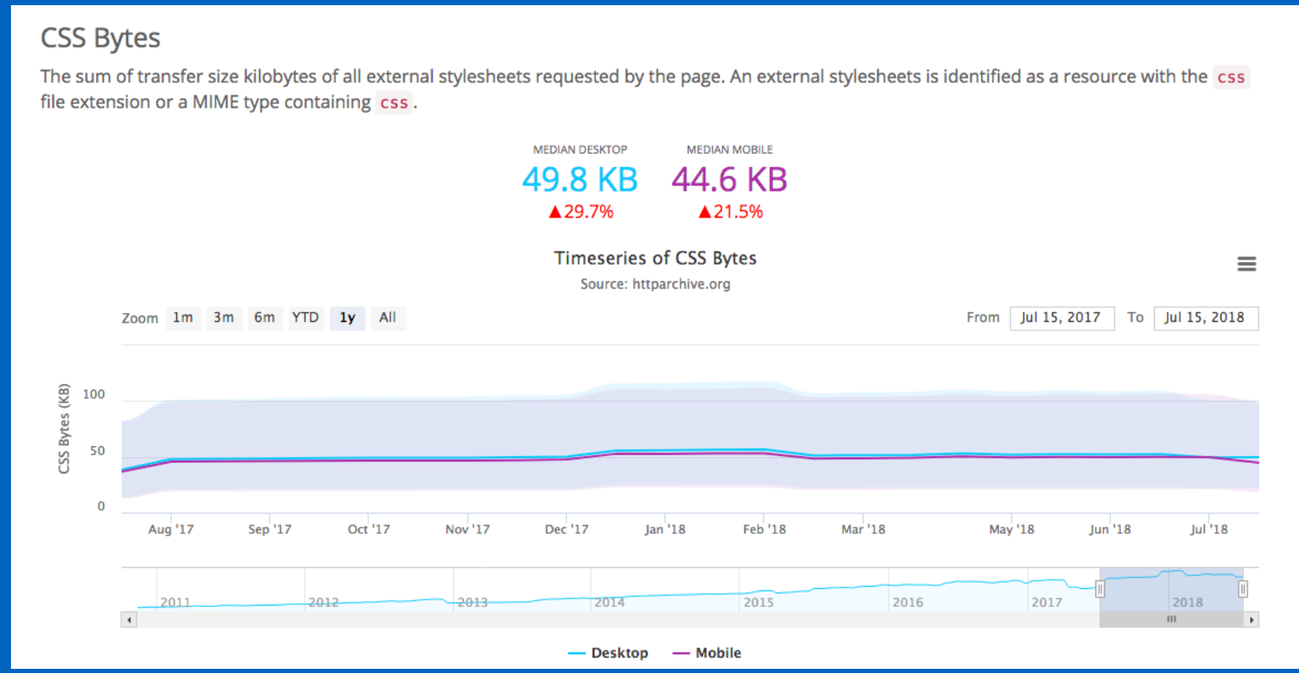
サイトA: サイトを比較してみるとCSSのサイズに問題

下記はgzipしたCSSのファイルサイズ

- **サイトA: 230kb** – 明らかにでかい！
- サイトX: 20kb
- サイトY: 33kb
- サイトZ: 37kb

一般的なサイトと比較する

- HTTP ArchiveでPage Weightで有名なサイトのHTML, CSS, JSなどのファイルサイズを調査
- CSSは**50Kb**弱が一般的なサイズ

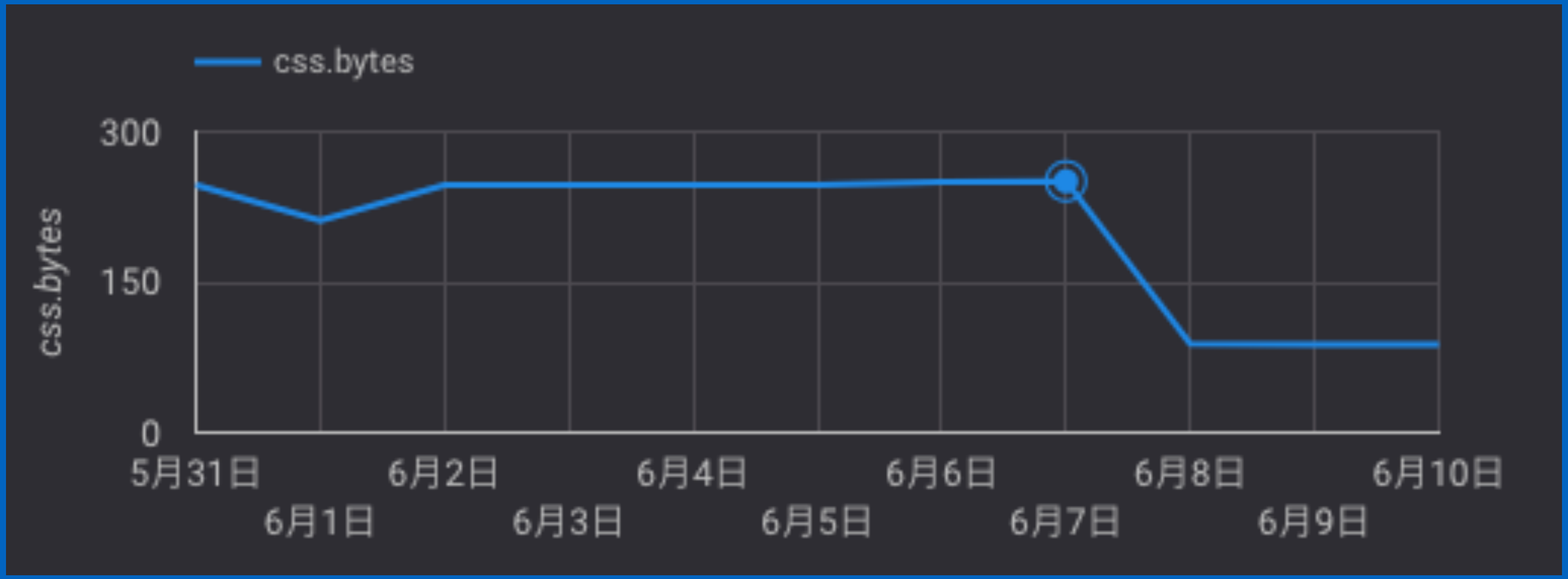


サイトA: CSSの問題を分析する

- TestMyCSSというサービスではCSSに含まれるBase64のサイズやセレクタの複雑度、重複、空のルールなどを一覧できる
- サイトAのCSSはBase 64が**150kb**ぐらいあり異常に大きかった
- CSSにフォントがBase64化したものが含まれていた

サイトA: CSSからフォントを取り除く

- CSSからフォントを取り除いた: **230kb** -> **80kb**



1つの改善がパフォーマンスに与える影響は大きくない

- 普通のウェブアプリの機能はそこまで極端なボトルネックを持っていない
- ページロードに関係する機能を一つだけ早くしても、全体への影響は小さい
  - リソースのロードなどは影響範囲が広め
- 継続的に計測し改善を続ける必要がある

例) サイトB: SpeedIndexを指標にちょっとづつ改善した

- サイトA: Reactで作られたサイト
- クライアントサイドレンダリングのみ
  - JavaScriptが実行されるまで真っ白なのでファイルサイズをへらすことが、そのままページロードタイムへ影響
- どれだけ初期表示に必要なJavaScriptを小さくしていくか

改善項目の一部

- Babel 7 -> ランタイムコストが小さくなる
- babel-plugin-external-helpers -> helper関数がまとまってファイルサイズが小さくなる
- webpack 4 + "module"フィールドの対応
  - TreeShaking/Scope Hoistingでサイズとランタイムコストの減少
  - `es-lodash`の対応 - Tree Shakingで最終的にはファイルサイズが小さくなる
- 初期表示に不要なコンポーネントをCode Splitting
  - `import()`で動的にロードすることで、初期表示のbundleファイルサイズが小さくなる
- React 16へのアップデート -> ファイルサイズの減少
- サードパーティのCSSやJSの非同期ロード -> 初期表示には不要なので画面に表示されたタイミングでロード

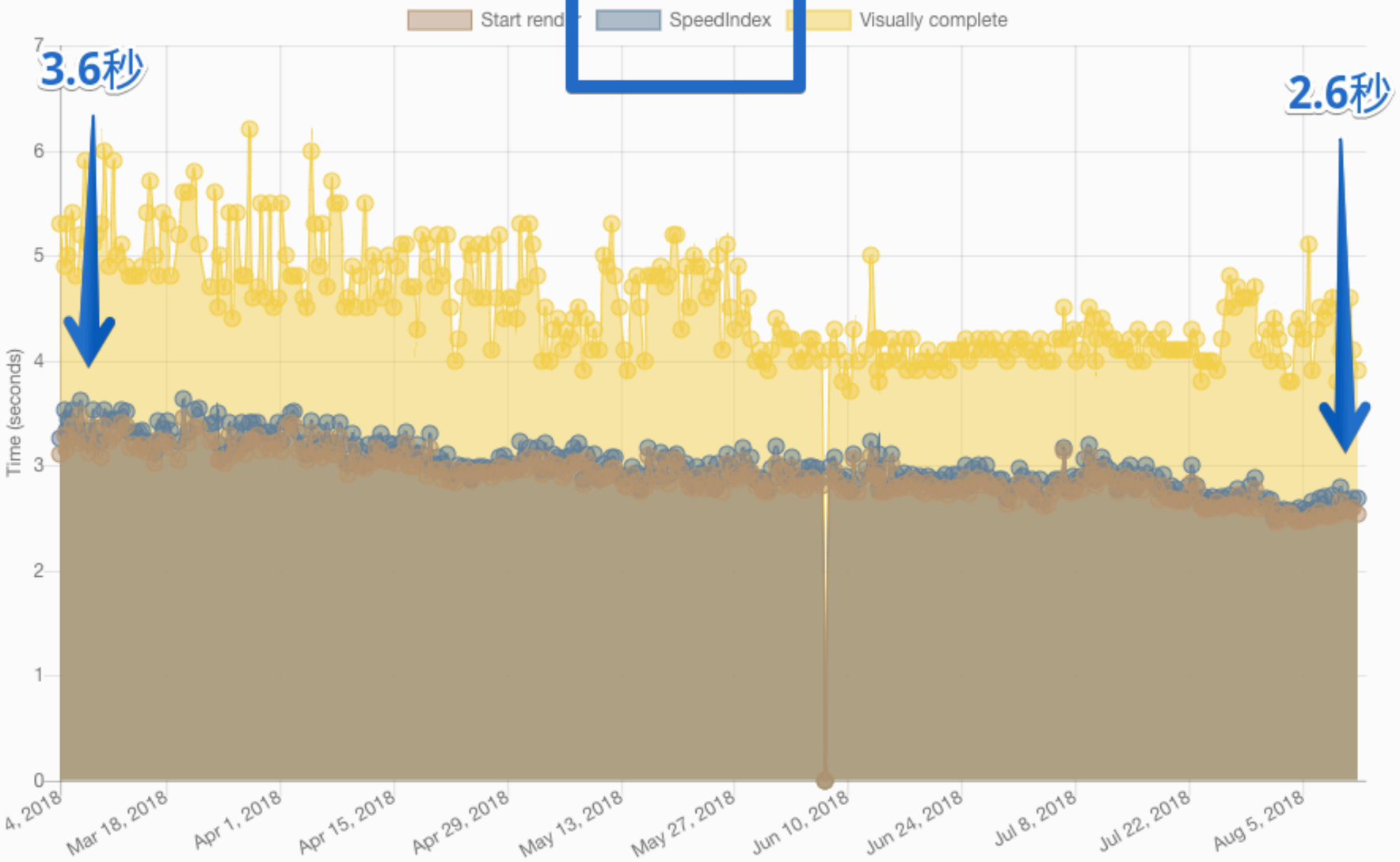
Rendering

半年の改善とSpeedIndexの変化

Start render 2.54s

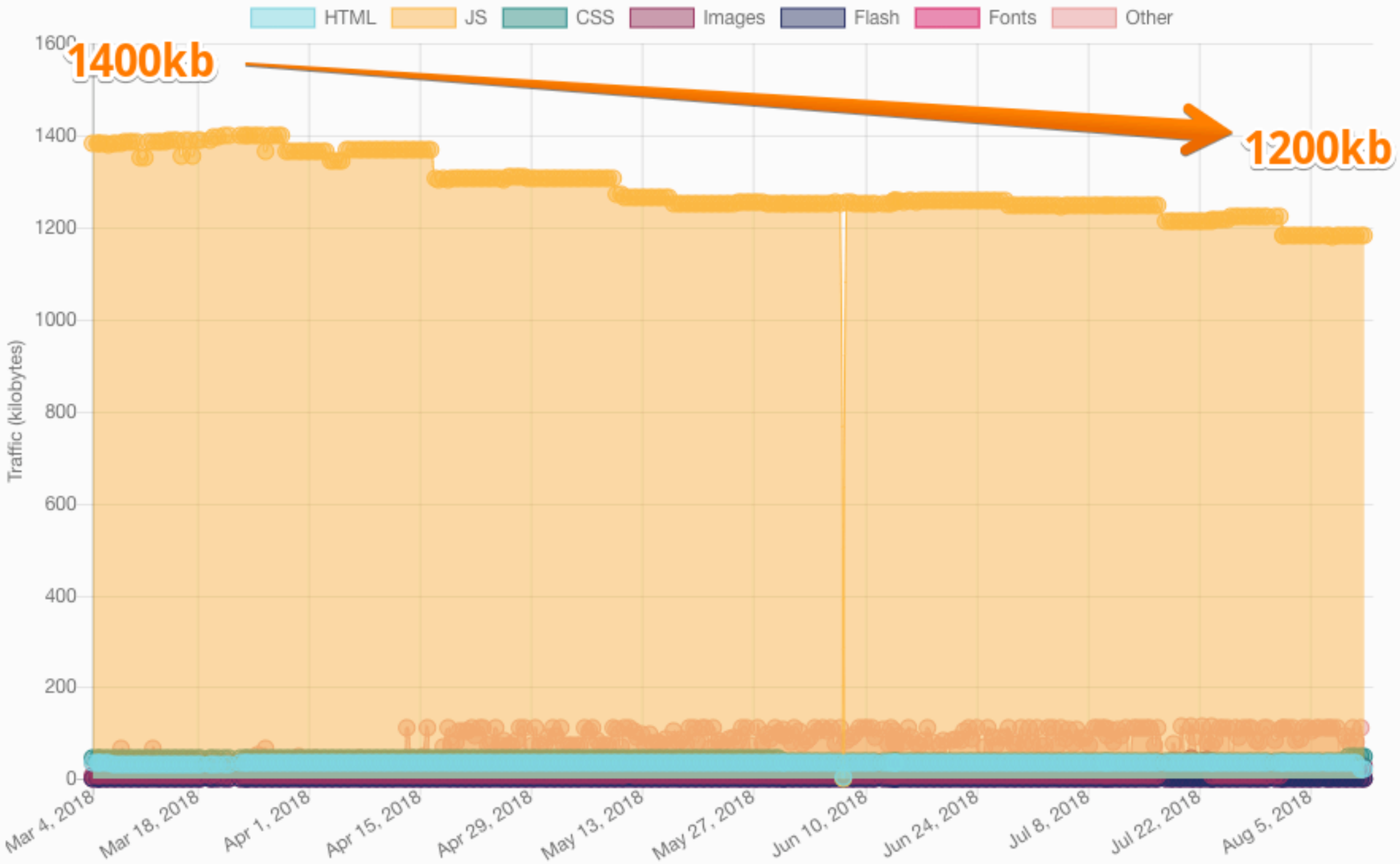
SpeedIndex 2.69s

Visually complete 3.90s



Content breakdown (size)

HTML	JS	CSS	Images	Flash	Fonts	Other
21.3KB	1183.4KB	48.2KB	28.6KB	0.0KB	0.0KB	14.0KB



劇的な変化を望む場合はアーキテクチャも変化を

- 劇的な変化を得るには全体的/根本的な変化が必要になる

例) クライアントサイドレンダリング(CSR)だけだったのをサーバサイドレンダリング(SSR)もするように変更

- インターネットテレビ局「AbemaTV」の表示速度が従来比3倍にアップ | 株式会社サイバーエージェント
- AbemaTVはただのSSR じゃねぇんだよ - Speaker Deck

アーキテクチャを変化できるような土壌を持つ

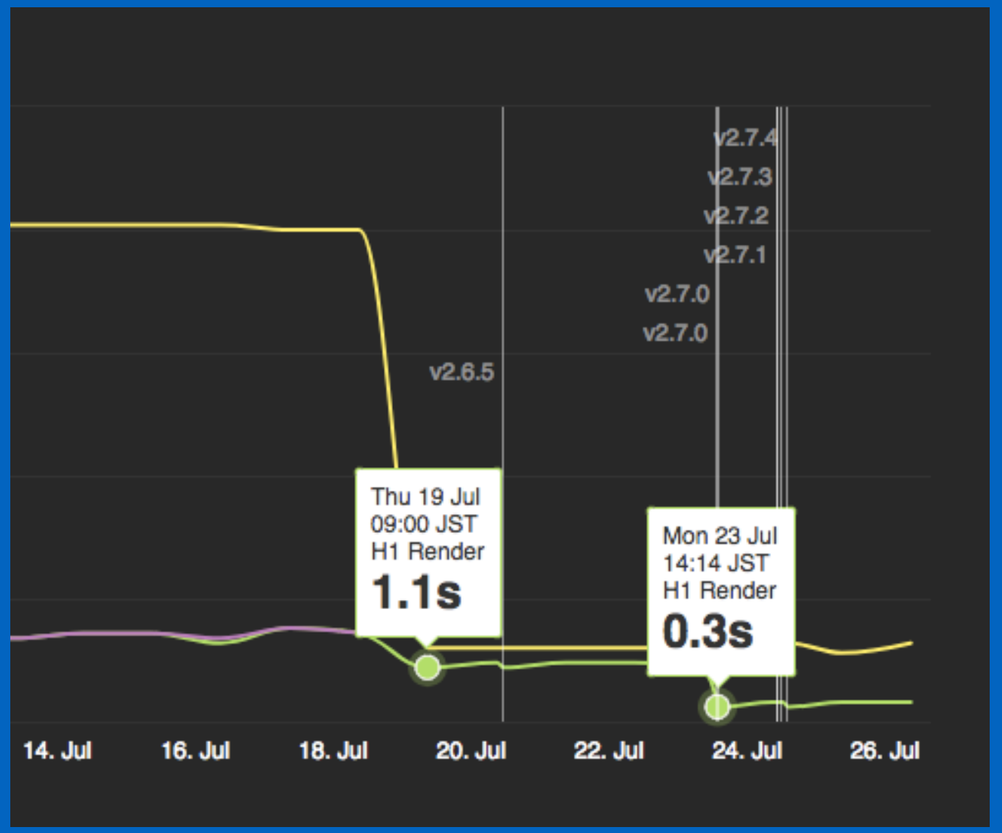
- 泥団子Big ball of mudのアーキテクチャは変更することが難しい
  - 計測は続けつつも、まずは泥団子を解体して正常な形にする
  - 正常な形にしていくと大抵リソースの扱いも正常できるため、結果的にはパフォーマンスが改善される
- アーキテクチャについては以前書いた
  - 複雑なJavaScriptアプリケーションを考えながら作る話

Big ball of mud 密結合でリファクタリングできないようなアーキテクチャの比喩。大きな泥だんごともいわれる

改善したら確認する

- 改善をするリリースをしたら、計測を実行して確認するという習慣は重要
  - 習慣化する = リリースしたらslackに通知をするとか、
  - 数値を出して興味を引く
- パフォーマンスに興味を持たないと徐々に悪化しやすい
  - パフォーマンスに関心を持つチームは必然的にパフォーマンスを改善する傾向
  - パフォーマンスに関心を持たないチームはパフォーマンスのリグレッションを起こしやすい
- かっこいいダッシュボードはモチベーションの維持のためという側面もある

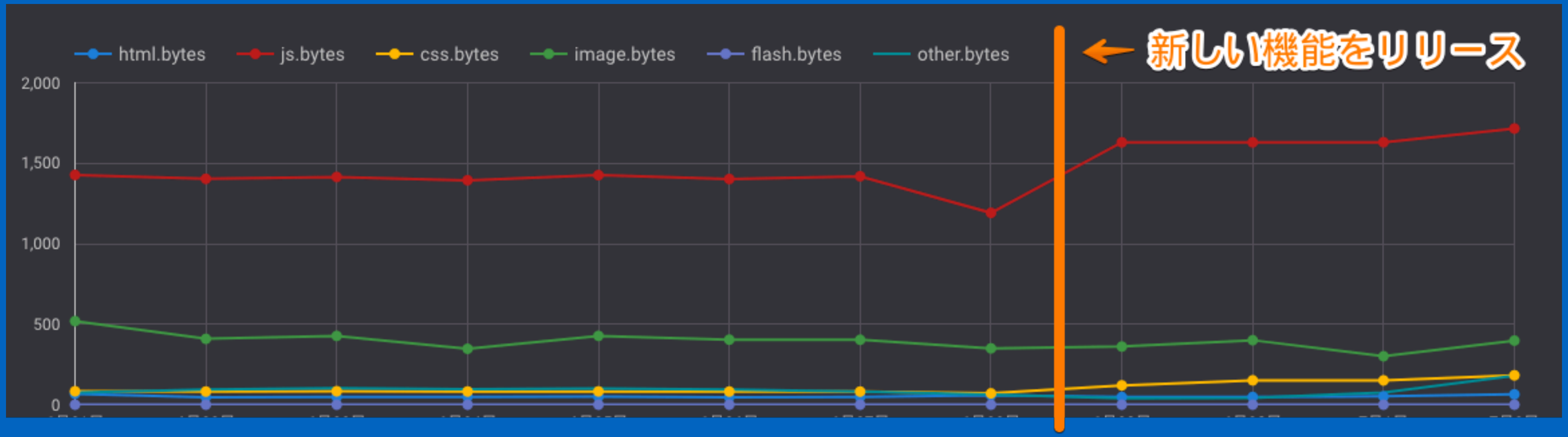
リリースにラベルをつけてわかりやすく表示してる例



パフォーマンスのリグレッションを検知する

- 人間は慣れてしまうので0.1秒とかの変化には気づきにくい
- 新しい機能を追加したときにパフォーマンスがリグレッションを起こしてもすぐには体感できない
- 合成モニタリングでパフォーマンスリグレッションを検知する
  - 結果が値として比較できる

例) サイトC: 新しい機能を追加したらサイズが増えた



新しい機能を追加したらサイズが増えた

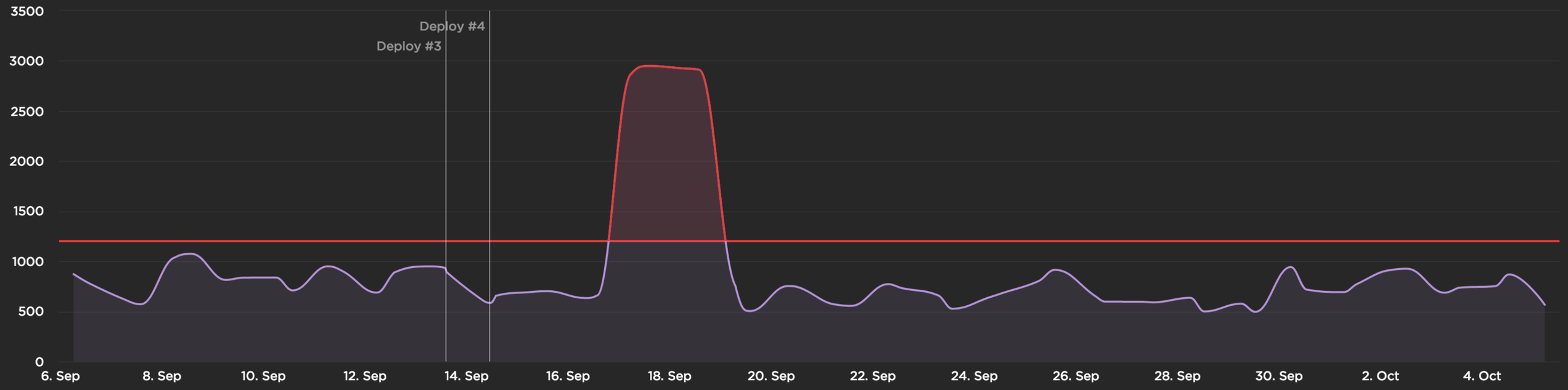
- 新しい機能をリリースしたらjsのbundleサイズが200kbも増えた
  - 原因: ライブラリの依存してるライブラリがでかかった
- Wifiだと違いを体感できないが、モバイルなどでは影響が大きい
  - 開発者の環境だと気づくのが難しい
- このようなリグレッションに気づくのにも継続的にパフォーマンス計測が必要

パフォーマンスの予算を決めて維持する

- パフォーマンスの予算(Performance Budget)を決めてその**水準を維持**する
  - プロダクトごとに決める必要がある(アプリによって指標は異なるため)
  - 基準値を決めるには常時計測して現状を把握する必要がある
  - 少なくても1週間ぐらいは計測結果を貯めてから基準値を決める
  - 例) 画像サイズの合計1200KBというPerformance Budgetを決める、そのしきい値を超えたらslackに通知する
  - Monitor your performance budgets | SpeedCurve Support

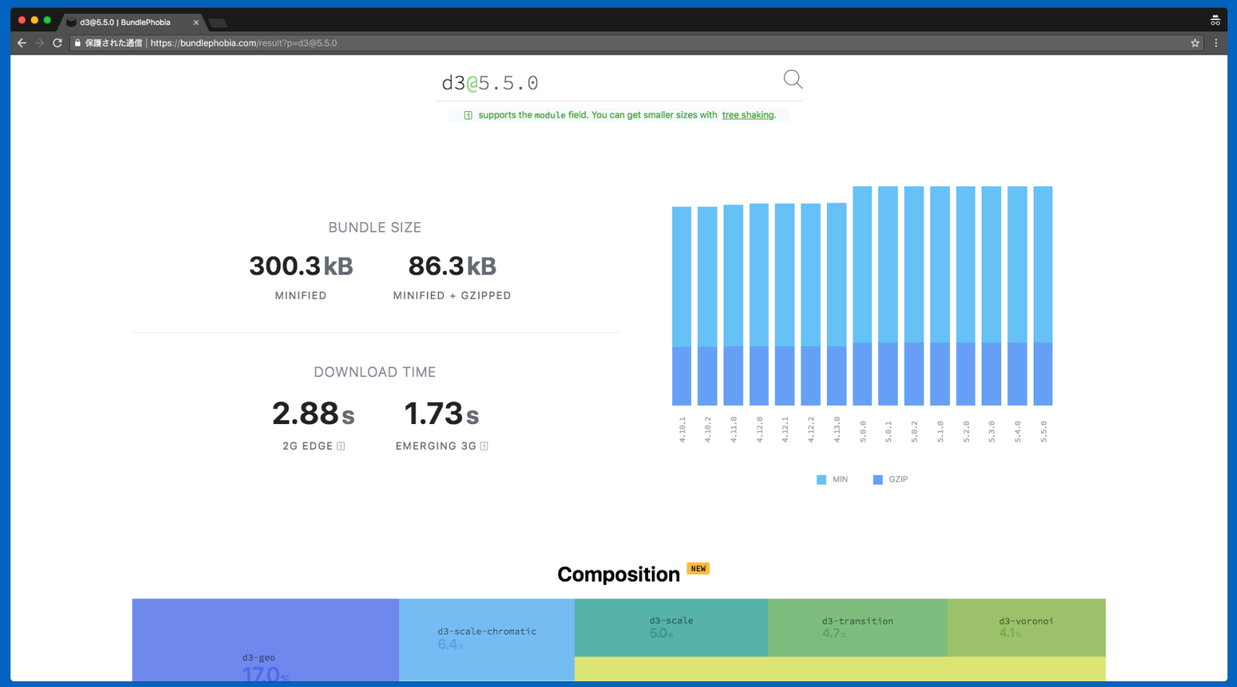
IMAGE SIZE BUDGET

Current Image Size	Image Size Budget	Decrease over 30 days (-35%)	Remaining budget (53%)
565KB	**1200KB**	**-306KB**	**635KB**



サイズの問題を予防する

- ライブラリを導入時にファイルサイズを気にする
  - npmならBundlePhobiaが便利
- bundleを分析するならwebpack-bundle-analyzerなど
  - webpackはBuild Analysisなどを参照
- ビルド時に検知する
  - webpackのperformance.hintsオプション
  - size-limitでファイルサイズをCIで計測する



新しく作るときの予算

- RAIL モデル
  - ページを使用する準備が整うまで 1,000 ミリ秒以内
- Performance Budget Calculator
  - 期待するロード時間から許容できるHTML,JS,CSSなどのファイルサイズを計算してくれる
- が、現実的にRAILモデルなどは予算としては厳しいアプリケーションも多いので柔軟にやりましょう
  - アーキテクチャ編: SSR と CDN (Fastly) とユーザー依存情報の分離（新規開発のメモ書きシリーズ4） ::ハブろぐ

パフォーマンス

- パフォーマンスは相対的な指標を扱う
  - 年々ウェブのサイズは大きくなっている
  - 平均サイズ: 2011年は500KB、2018年は1500KB
  - State of the Web
- 決めた水準以下を維持することを目標にする
  - 水準を変えたいとなったとき、計測した値は役立つ

まとめ

- 改善するために計測していく
- 継続的に改善するためには指標をちゃんと持つ
- 速くするのではなく遅くしない[not-slow]
- 基準値(Performance Budget)を決めてそこより遅くならないことをベースにする
- 計測方法も徐々に改善していく

[not-slow] システム全体で見ればパフォーマンスは1因子でしかない。問題の正常化にまず取り組む

参考

- thedaviddias/Front-End-Performance-Checklist: 🎮 The only Front-End Performance Checklist that runs faster than the others
- Loading Third-Party JavaScript | Web Fundamentals | Google Developers
- 超速！ Webページ速度改善ガイド ──使いやすさは「速さ」から始まる：書籍案内｜技術評論社
- WebパフォーマンスとプロダクトKPIの相関を可視化する話
- Performance/WebPageTest - Wikitech
- Using WebPageTest - O'Reilly Media
- speedtracker/speedtracker: 📉 Visualisation layer and data store for SpeedTracker
- Webサイトパフォーマンス管理の基礎知識
- Synthetic vs Real User Monitoring : What to use when? | Tezify Blog
- 最近の Web パフォーマンス改善について知っておきたいコト -HTML5 Conference 2017- - YouTube
- Web クライアントサイドのパフォーマンスメトリクス — Speed Index、Paint Timing、TTI etc... ::ハブろぐ
- 日経電子版 サイト高速化とPWA対応 / nikkei-high-performance-pwa - Speaker Deck
- Introducing a faster BBC News front page | Wildly Inaccurate

FAQ

- Lighthouseとかではダメなの?
  - WebPagetestはLighthouseの結果も含んでいる
  - パフォーマンス改善に必要なのは連続的値
  - スコアじゃなくて時間を計測する(スコアはルールが変わることがある)

個人的なパフォーマンス計測のスタートポイント

- まずは計測を始めよう
  - gas-webpagetest + 好きなSaaSの2種類で計測を開始しよう
  - WebPageTestは無料で使えてすごい、有料の計測サービスも一緒に体験しよう(Trialがある)
- 1-2週間ぐらい計測をしながら現状を把握しよう
  - ブラウザの開発者ツールのパフォーマンス計測を自動化すると考えるならある程度でも形がでてくる
- 多くの計測データにはバイアスがかかっているので、分析には気をつけよう
  - パフォーマンスは様々な因子がでてくる総合的なもの
  - どんなに優れたモニタリングツールではすべての因子を見ることができない
  - スコアやグラフは丸められたデータであることは意識しよう
- パフォーマンス計測を継続することは偉いこと
  - その行為自体は否定はされるものではない
  - パフォーマンス計測自体も徐々に改善していけばいい

メモ

- CatchPointは明らかに他のサービスより広い範囲のデータを取っている
- その分明らかに値段も高めの設定になってる
- CDNがakamai中心だった時代が終わるように、モニタリングも多様化して変化していく
  - 国内CDNシェア(2018年4月) | J-Stream CDN情報サイト
- 必要性に応じてモニタリングも変化していくと考えられる
  - まだまだ敷居の高さがある

使わなかったスライド

ウェブアプリには様々な要素が混在する

- ウェブアプリのアーキテクチャはさまざまな要素からできてる
  - メンテナンス性、パフォーマンス、セキュリティ、スケーラビリティ、
- すべてを実現することはできない
- 例) 高パフォーマンスと高スケールの両方を実現することは簡単ではない
- そのようなアーキテクチャを決めるときにも指標をちゃんと数値として決める
- 指標や方向を持って変化して、それを確認するすべを持つ

> システムは決して部分の総和ではない。システムは部分の相互作用の成果だ。
>
> — *Building Evolutionary Architectures*(進化的アーキテクチャ)

システムはバランス

- フロントが遅いかバックエンドが遅いかネットワークが遅いか
- 結局はバランスの問題になる
- システムはそれぞれのコンポーネントの足し算じゃなくて、相互作用で成り立っている
  - フロントが遅いのを直せば、バックエンドへアクセスが増え負荷がかかるといったように
- それぞれをちゃんと計測しておいて、(あらゆる)修正ごとに何が変化したのを見られるような状態が正常

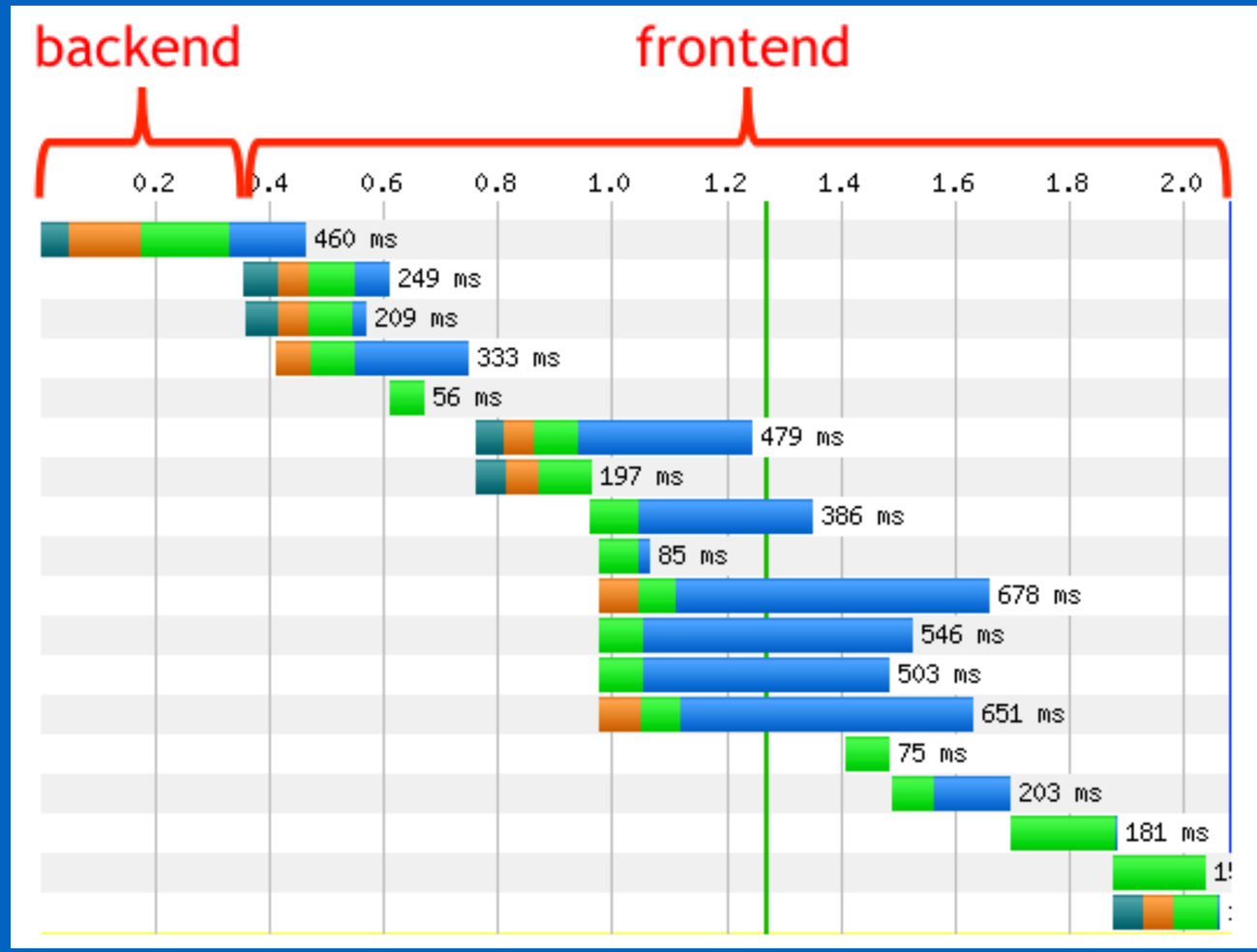
ページロードに関するパフォーマンスはフロントの裁量が大きい

- フロント側だけをいくら早くしてもバックエンド側が遅いとダメ(逆も同じ)
- しかし、バックエンドの役割のほとんどはリソース配信
- 特にSPAなどのJavaScriptでUIを作る場合はフロント側の比率が高くなる傾向

エンドユーザーに対するレスポンスタイムのうち80%〜90%はWebフロントエンドで発生する

80-90% of the end-user response time is spent on the frontend.
Start there. [note]

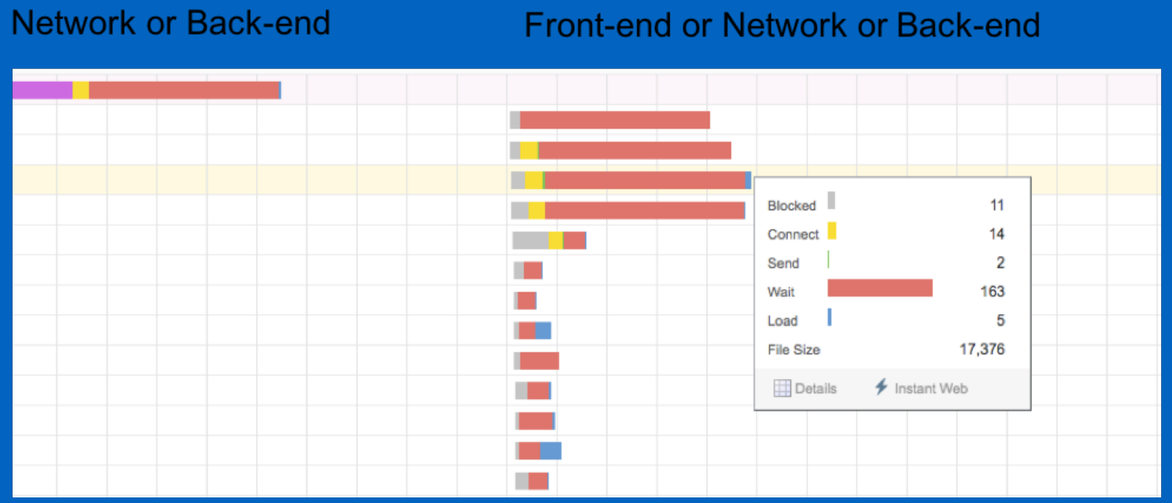
— *the Performance Golden Rule | High Performance Web Sites*



[note] この分類はフロントエンドがコントロールできる領域が80-90%という意味で考えている

+ ネットワーク

- フロントが8-9割は言い過ぎなので、もうすこし細かく見ると次の分類
  - フロントエンド
  - バックエンド
  - **ネットワーク**
- Front-end vs Back-end vs Network Performance | Web Performance



Metric	Classification
DNS	Network
TCP Connection	Network
SSL	Network
Page Load Time	End-to-end
TTFB/Wait	Back-end, Network
Re-directions	Back-end, Network
Content Download	Back-end, Network
DOM Interactive	Front-end
Document Complete	End-to-end
Speed Index	End-to-end
Total Time to First Byte	End-to-end

フロントエンドのコントロール領域は広い

- Network
  - Resource Hintsでの投機的取得
- Content Download
  - 不要なものを取捨選択、遅延ロード、Service Workerでのキャッシュ
- フロントエンドはどんなリソースを必要とするか知っていて、それをコントロールできる
- ページロードにおいてフロントエンドがコントロールできる領域が8割から9割と考える[ex]

[ex] もちろんバックエンドやネットワークが絶望的に遅いとかだとこの比率は変わってしまう

パフォーマンス改善のセオリーも変わる

- パフォーマンス改善の"いい方法"というのもどんどん変わってる
  - パフォーマンス計測サービスも新しい機能が次々と増えている
  - SpeedCurveやCalibreなどは機能追従が早い
  - Chrome自体にも新しいメトリクスが増えている
- HTTP/1とHTTP/2でセオリーも違う

使うべき指標

- 使うべき指標を決めたらパフォーマンス予算を決めてそれを超えないようにする
- その予算を決めるためにも、ある程度の量(少なくても1週間分ぐらい)は計測して傾向を掴む必要がある

- ページの表示速度を改善したい(ページロード)
  - URLにアクセスしてからできるだけコンテンツを早く表示したい => First MeaningFul Paint
  - URLにアクセスしてからできるだけ早くコンテンツを操作できるようにしたい => TTI
- 動作を高速にしたい(ランタイム)
  - 重たい操作があるのを軽くしたい -> FPSをチェックする
  - レスポンスのいい検索、メニューがすぐ切り替わる、ページ移動が早い
- 安定した動作をしてほしい
  - ストリーミング再生が安定して行える -> Long Taskが発生してないか
  - いきなりクラッシュしないで -> エラーハンドリングをちゃんと
- モバイルでも動作が安定してほしい
  - ネットワーク帯域に気を配る -> ファイルサイズのコスト

計測結果のかたよりをへらすために

- 計測結果のかたよりをへらすために
  - 環境などコントロールできるところはコントロール
  - 無作為に選んだ条件で繰り返し計測する
  - 反復回数はできるだけ多く
  - 1日2回とかではなく、30分前後に1回とか、1度の計測で3回リトライするとか
- 実験計画法 - Wikipedia
- gas-webpagetestは30分に1度、3回のリトライをしている

ブラウザとパフォーマンス

- ブラウザ自体のパフォーマンスについて
- WebKitはパフォーマンスリグレッションはrevertする
  - Performance Testing | WebKit
- JavaScriptエンジンのベンチマークを比較するサイトをMozillaが管理している
  - ARE WE FAST YET?
- パフォーマンスリグレッションはバグ扱い