Decoupled

Why we're building a worse wheel

デカップルド
なぜ悪い車輪を構築してしまうのか

Josh Waihi

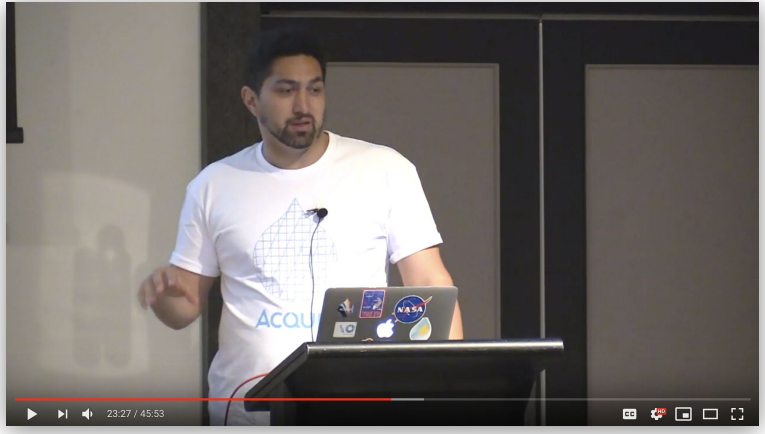
November 2019

ACQUIA

Drupal South Hobart 2019

で、Josh Waihiが話した内容を日本語で発表します

https://drupalsouth.org/



https://www.youtube.com/watch?v=6GwTClYxFmo

Acquia

Profile

自己紹介

Hikaru Maruyama 丸山ひかる

Acquia Technical Translator

Developer Relations

Drupaler Since 2019.08

Ruby / Rails / Web API / Docker / AWS

Acquia

HISTORY

LET'S LOOK AT DRUPAL'S EVOLUTION

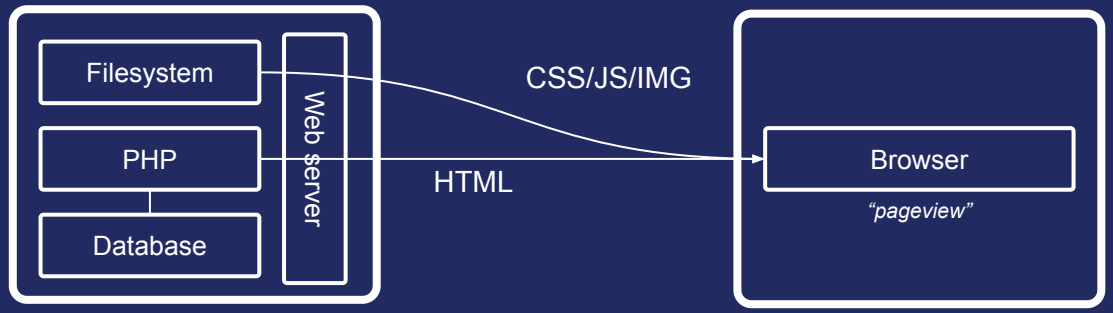
Drupalの進化をみてみましょう！

ACQUIA

Traditional Drupal

The static web

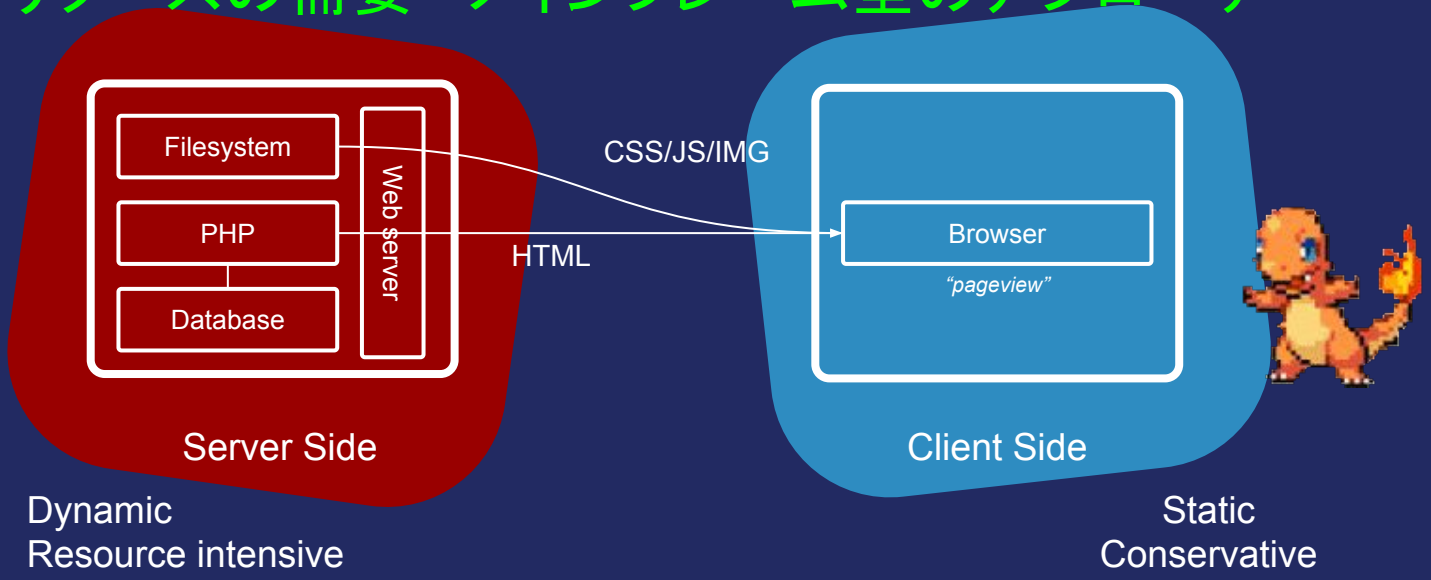
従来のDrupal - 静的ウェブサイト



Resource demand

A “mainframe” approach

リソースの需要 - メインフレーム型のアプローチ



Dynamic
Resource intensive

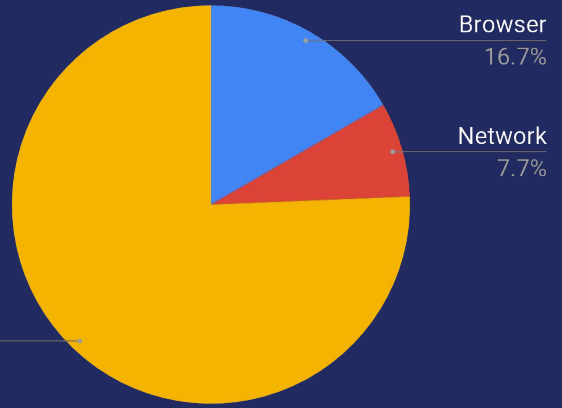
Static
Conservative

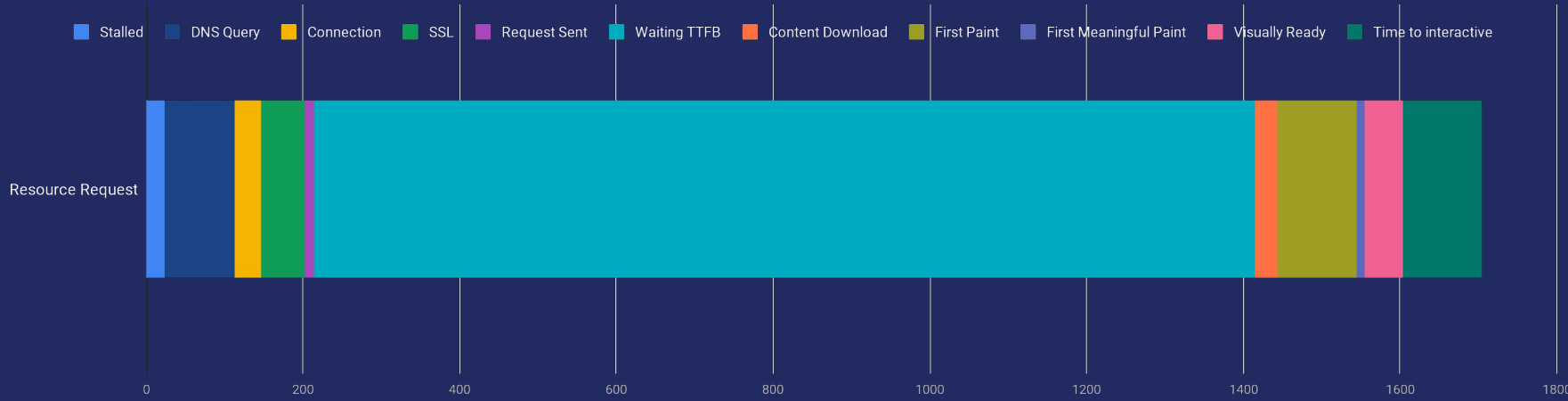
ACQUIA

Loading Journey

Drupal bound performance

読込時間から見るDrupalの性能限界



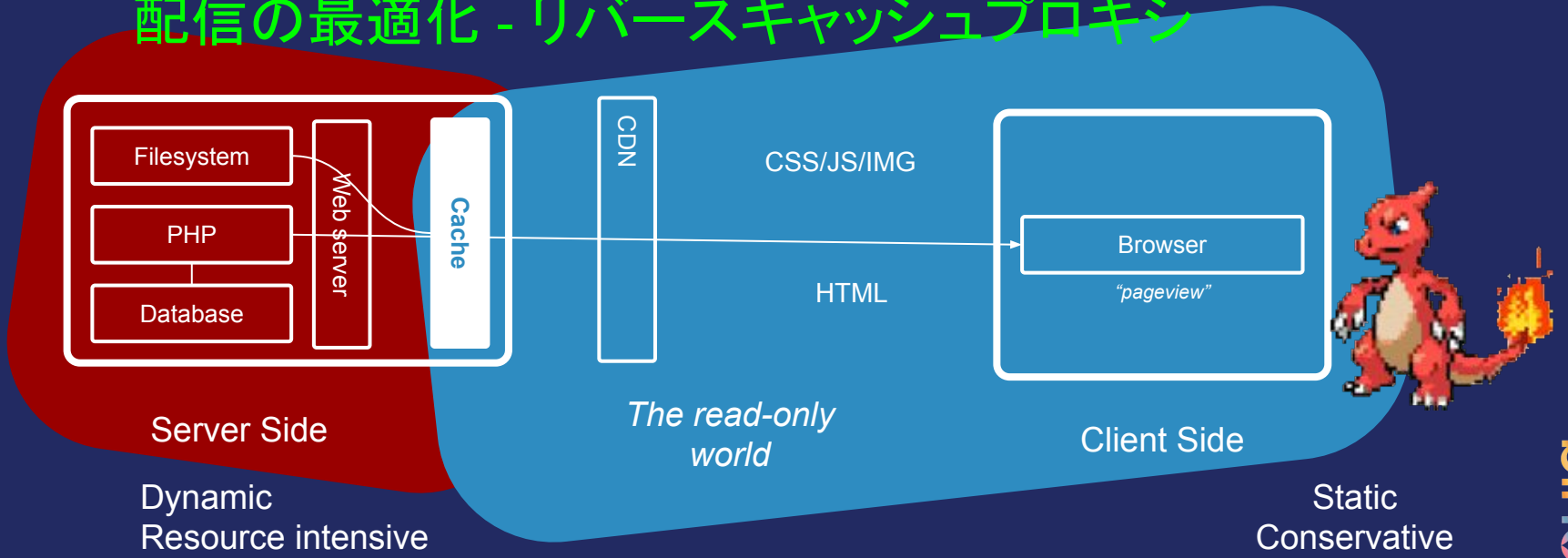


ACQUIA

Delivery Optimisation

Reverse cache proxies

配信の最適化 - リバースキャッシュプロキシ

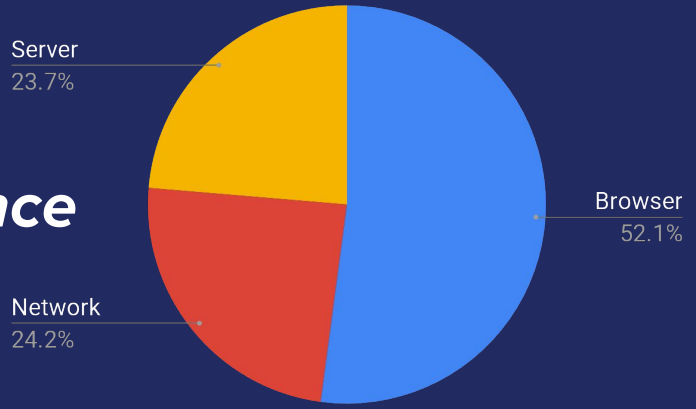


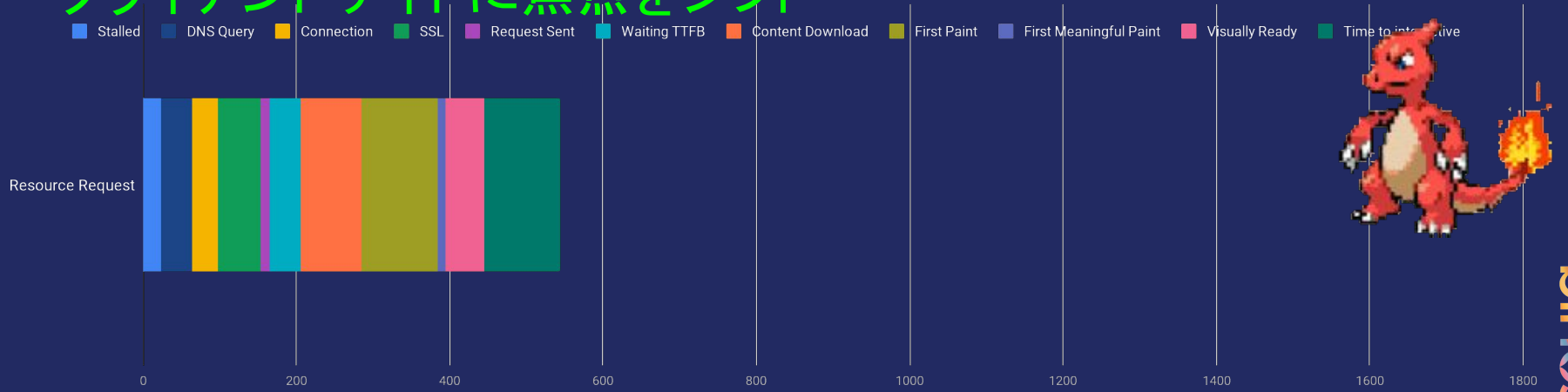
ACQUIA

Loading Journey

Server-side caching shifts performance focus to client-side

サーバーサイドキャッシュにより、
クライアントサイドに焦点をシフト





ACQUIA

Performance
example 10k pageviews

1万ページビューの場合のパフォーマンス

- Site = 500 pages
- Average Drupal Response Time = 1.8 seconds
- 10,000 pageviews

サイト = 500ページ数
平均レスポンスタイム = 1.8秒
1万ページビュー

ページ配布	Page Distr	20% (100)	30% (150)	35% (175)	15% (75)
トラフィック	Traffic Distr	70% (7k)	20% (2k)	7% (700)	3% (300)
キャッシュヒット率	Cache Hit Rate	98.57%	92.5%	75%	75%

- Cache Hit Rate Avg 85.27%
- 14.73% of traffic > 1.8s TTFB
- Assuming no cache invalidation or expiry takes place

平均キャッシュヒット率 85.27%
14.73%のトラフィック > 1.8秒のTTFB(TIme To First Byte)
キャッシュ無効化なし and 有効期限なしを想定

ACQUIA

Performance
example 5k pageviews

5千ページビューの場合のパフォーマンス

- Site = 500 pages
- Average Drupal Response Time = 1.8 seconds
- 5,000 pageviews

サイト = 500ページ数
平均レスポンスタイム = 1.8秒
5,000ページビュー

ページ配布	Page Distr	20% (100)	30% (150)	35% (175)	15% (75)
トラフィック	Traffic Distr	70% (7k)	20% (2k)	7% (700)	3% (300)
キャッシュヒット率	Cache Hit Rate	97.14%	85%	58.33%	50%

- Cache Hit Rate Avg 72.62%　平均キャッシュヒット率 72.62%
- 27.38% of traffic > 1.8s TFFB　27.38%のトラフィック > 1.8秒のTTFB(TIme To First Byte)
- Assuming no cache invalidation or expiry takes place

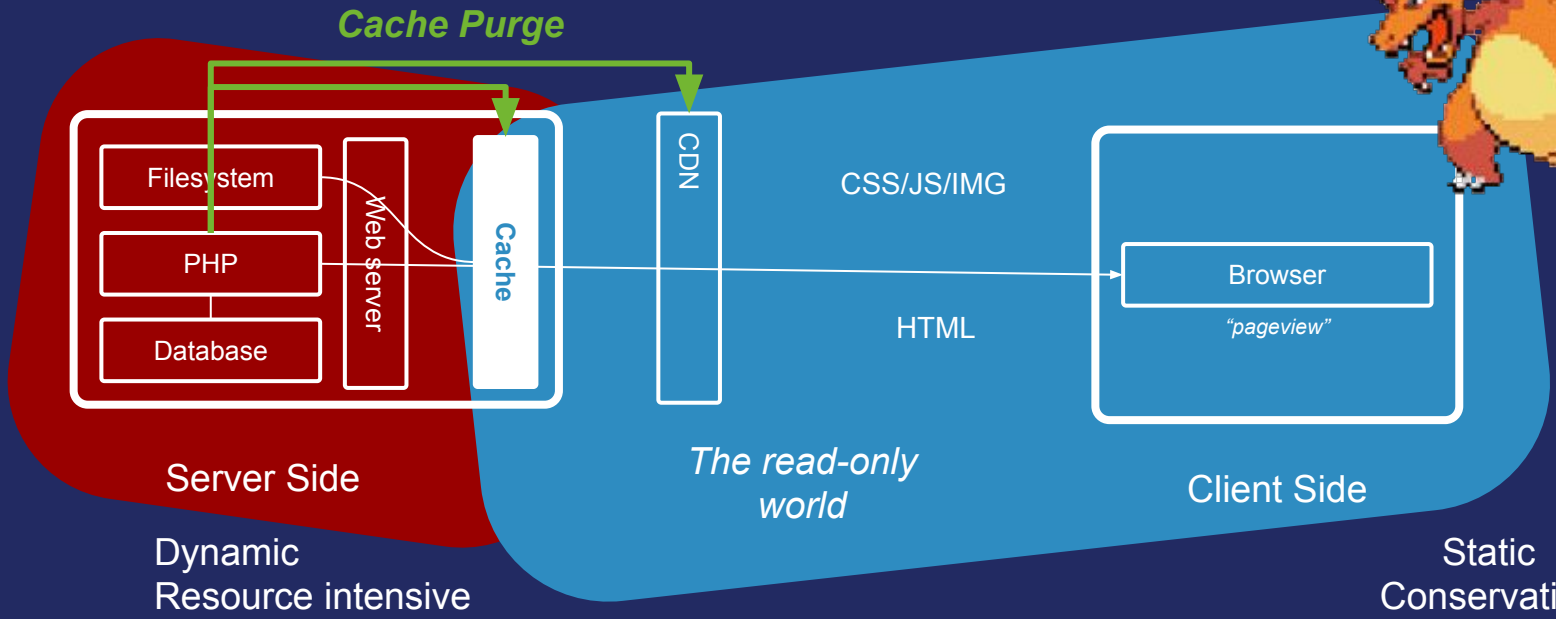
キャッシュ無効化なし and 有効期限なしを想定

ACQUIA

コンテンツの完全性 - キャッシュの無効化

Content Integrity

Cache invalidation



ACQUIA

現代のパラダイムシフト

Modern Paradigm Shifts

In the Web

- Data refresh, not page refresh ページの更新ではなくデータの更新
- Personalised experience パーソナライズ体験
- Event-driven data push イベントドリブンなデータプッシュ

JavaScriptは現代の要求に適している

Javascript is naturally suited to modern demands of the web

- Pageless DOM manipulation ページレスなDOM操作
- Client-side personalisation processing power クライアント側の処理能力
- Event-driven backend supports push (websockets) and pull (req)
  イベントドリブンなバックエンドは
  プッシュ(websockets)とプル(リクエスト)をサポート

ACQUIA

従来のDrupalのメリット/デメリット

Traditional Drupal

Pros/Cons

Pros メリット

- Content Management
  コンテンツマネジメント
- Optimised Content Delivery
  コンテンツ配信の最適化
- Centralised processing
  集中処理
- Scale simple content
  シンプルなコンテンツのスケール

Cons デメリット

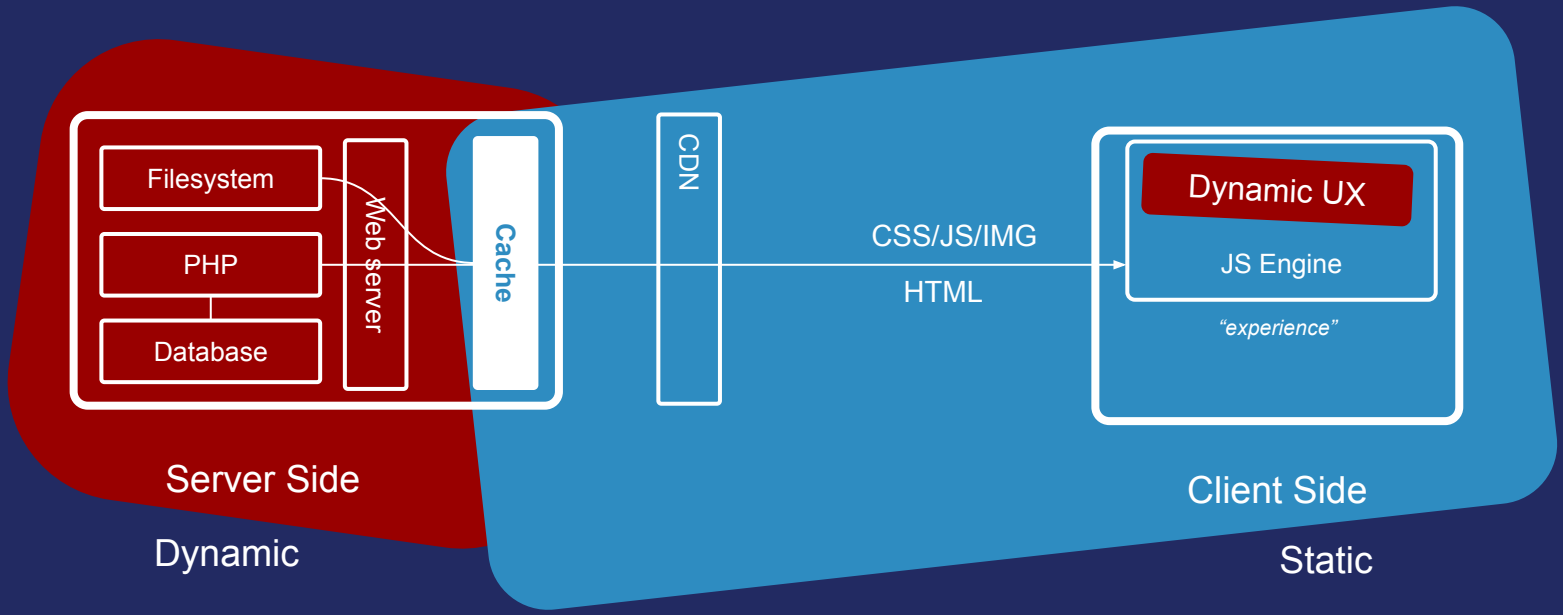
- Monolithic
  モノリシック
- Complexity with personalisation
  パーソナライズの複雑さ
- Limited frontend leverage
  フロントエンドレバレッジの制限
- Slow TTFB
  遅いTTFB

ACQUIA

動的クライアント - ローカライズされた処理

Dynamic Client

Localised processing

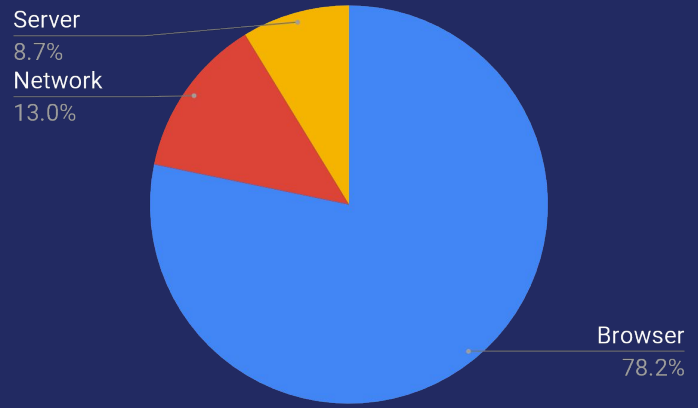


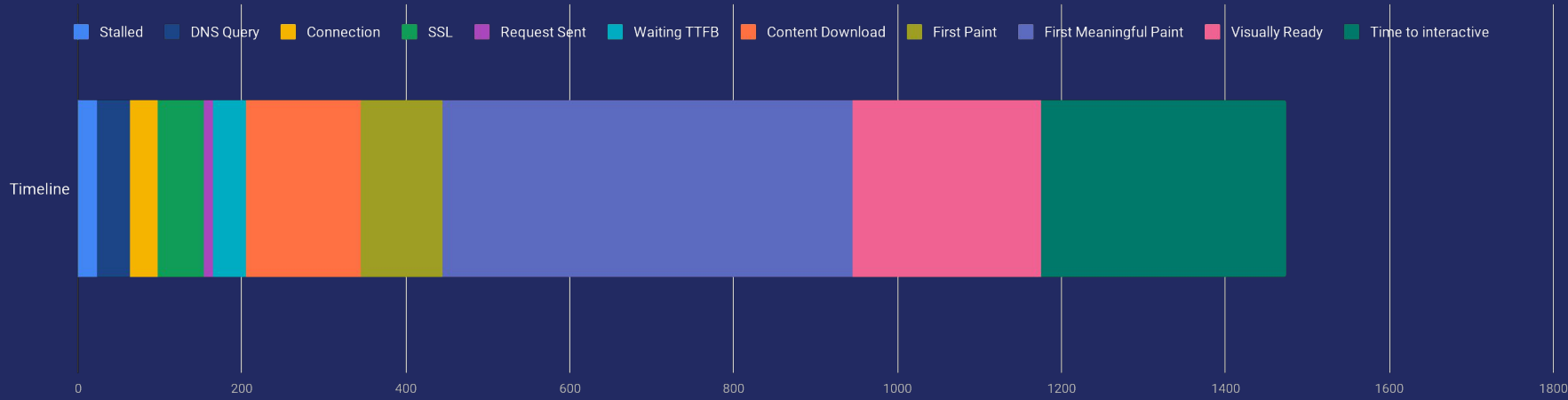
ACQUIA

Loading Journey

Client-side complexity

クライアント側の複雑さ



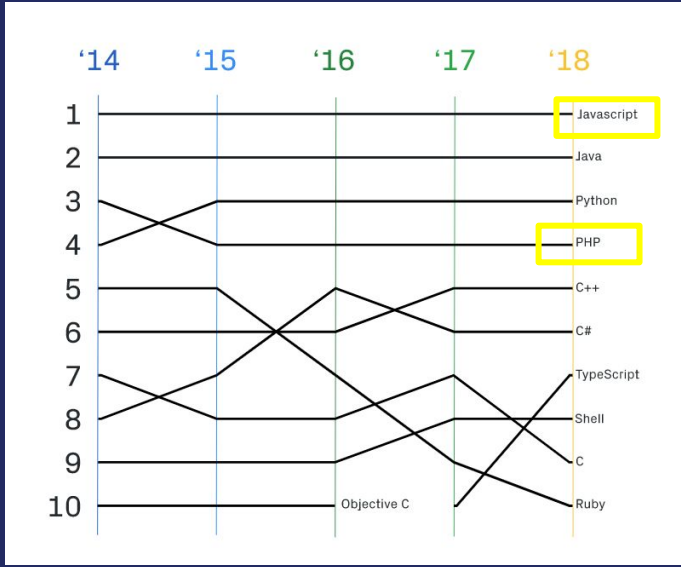


ACQUIA

POWER TO THE PEOPLE

JavaScriptは人気

JavaScript - the popularity of



The State of Octoverse results for 2018. Category: languages.
Source: Octoverse

JavaScriptは求人でも人気

JavaScript - the job market

Cloud computing, ecommerce tech skill searches growing fast

Year-over-year growth

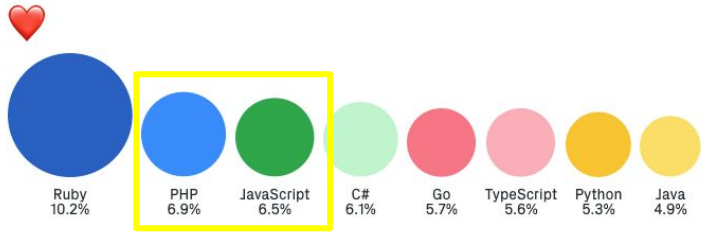
Rank	Skill	Job search growth	Rank	Skill	Job search growth
1	Kubernetes	173%	11	Azure	53%
2	Magento	116%	12	Spanish	46%
3	Verilog	89%	13	Django	45%
4	Golang	81%	14	PHP	45%
5	Ansible	72%	15	Blockchain	44%
6	Autocad	71%	16	A+	44%
7	Laravel	66%	17	Siebel	43%
8	React	61%	18	Chinese	38%
9	Node.js	57%	19	CCNA	32%
10	C	54%	20	HTML	30%

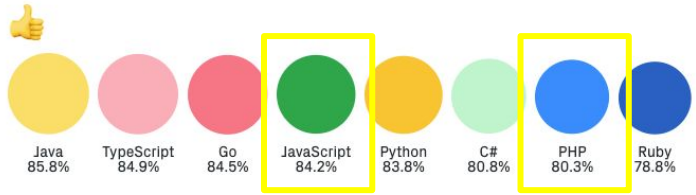
indeed

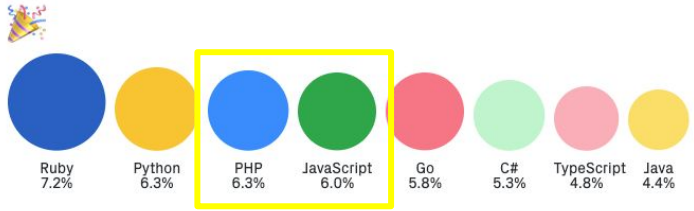
https://www.hiringlab.org/2018/11/29/hottest-skills-tech-job-searches1/

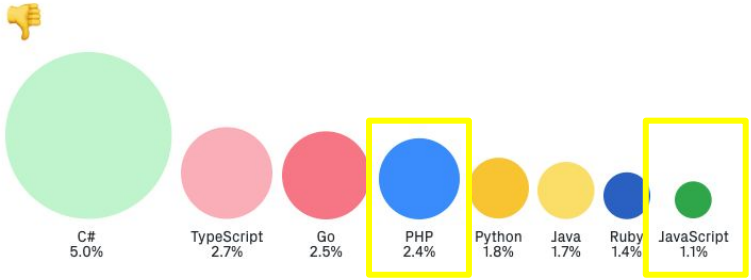
ACQUIA

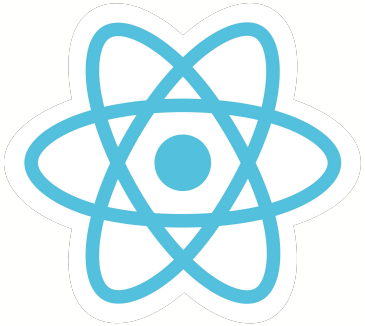
GitHub Emojis







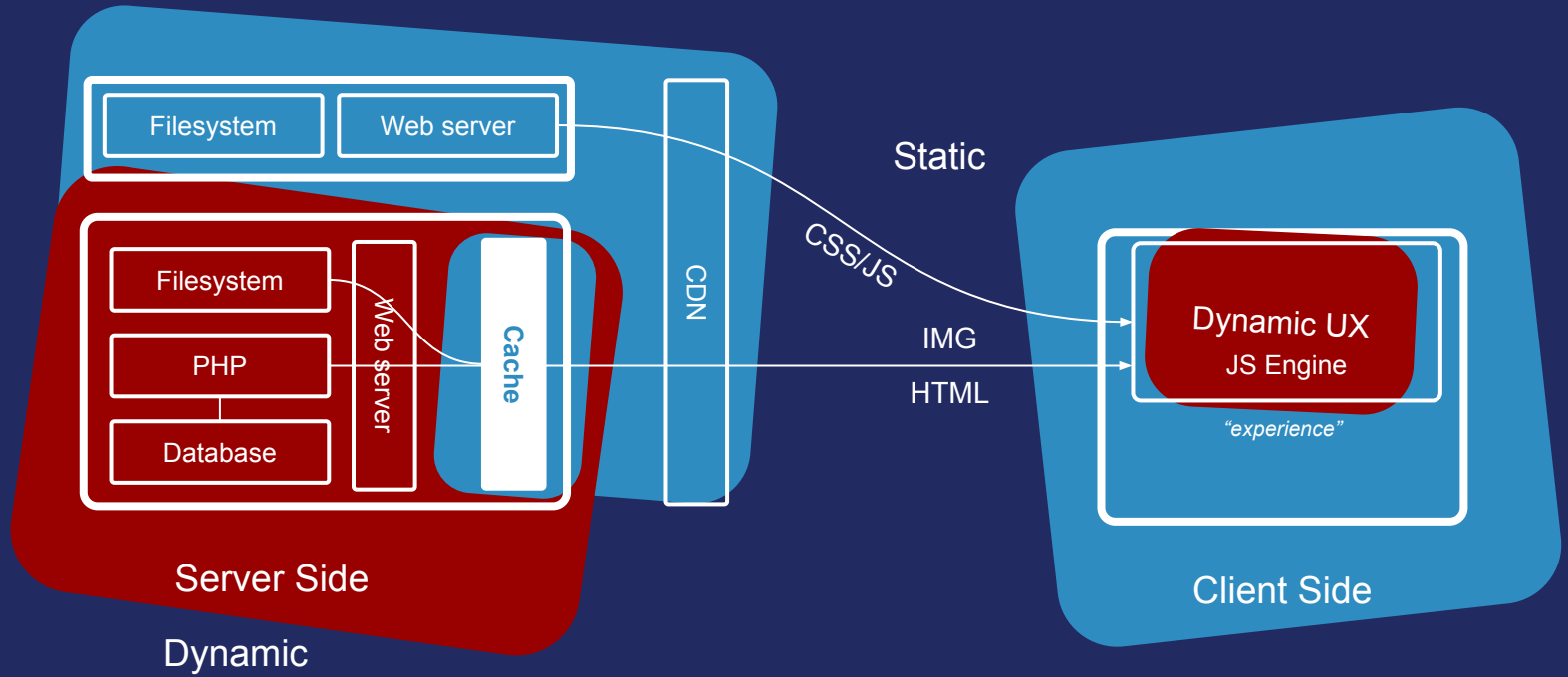




マイクロサービス - ポリリシックなサービス提供

Microservices

Polylithic service delivery

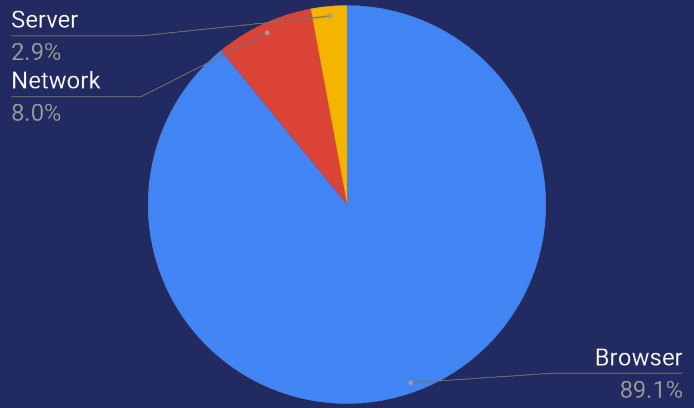


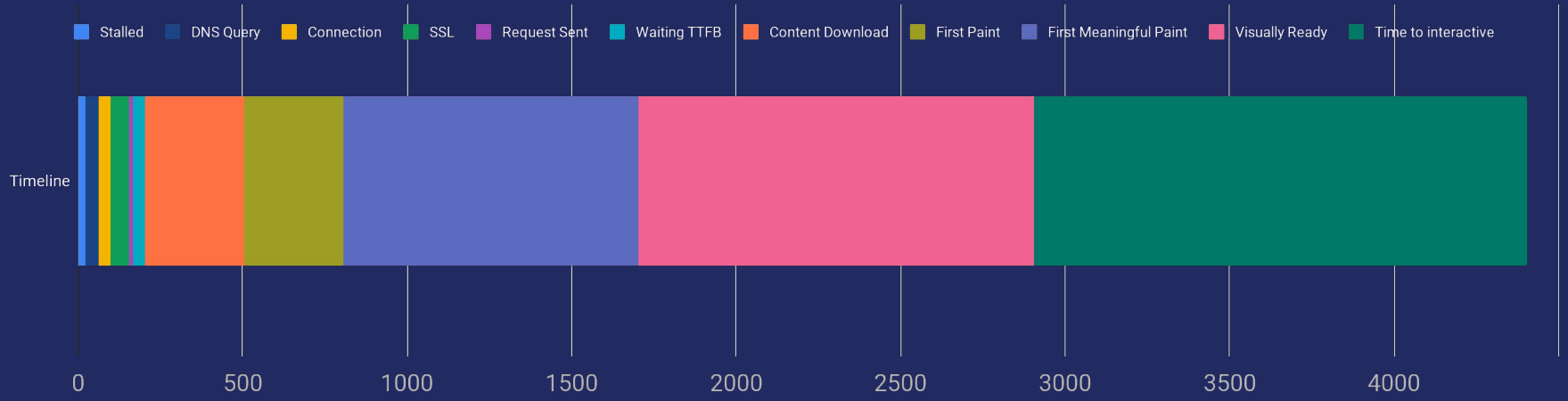
ACQUIA

Loading Journey

Client-side App loading

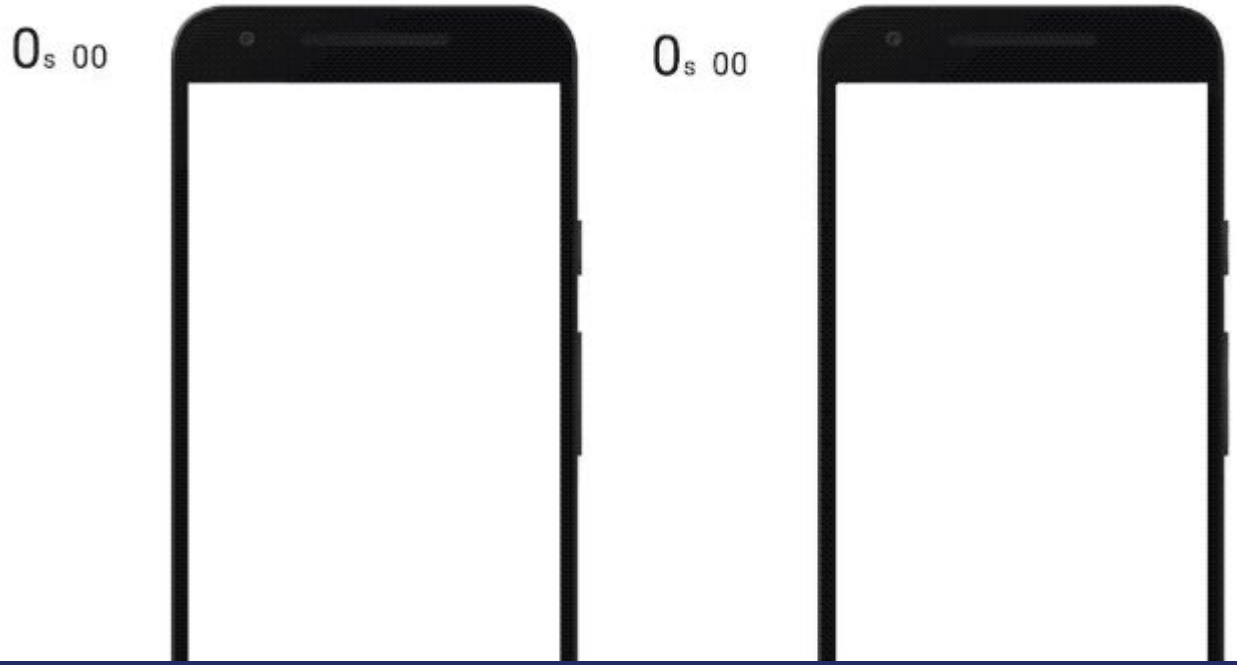
クライアントサイドの
アプリケーション読み込み





ACQUIA

TIME TO INTERACTIVE



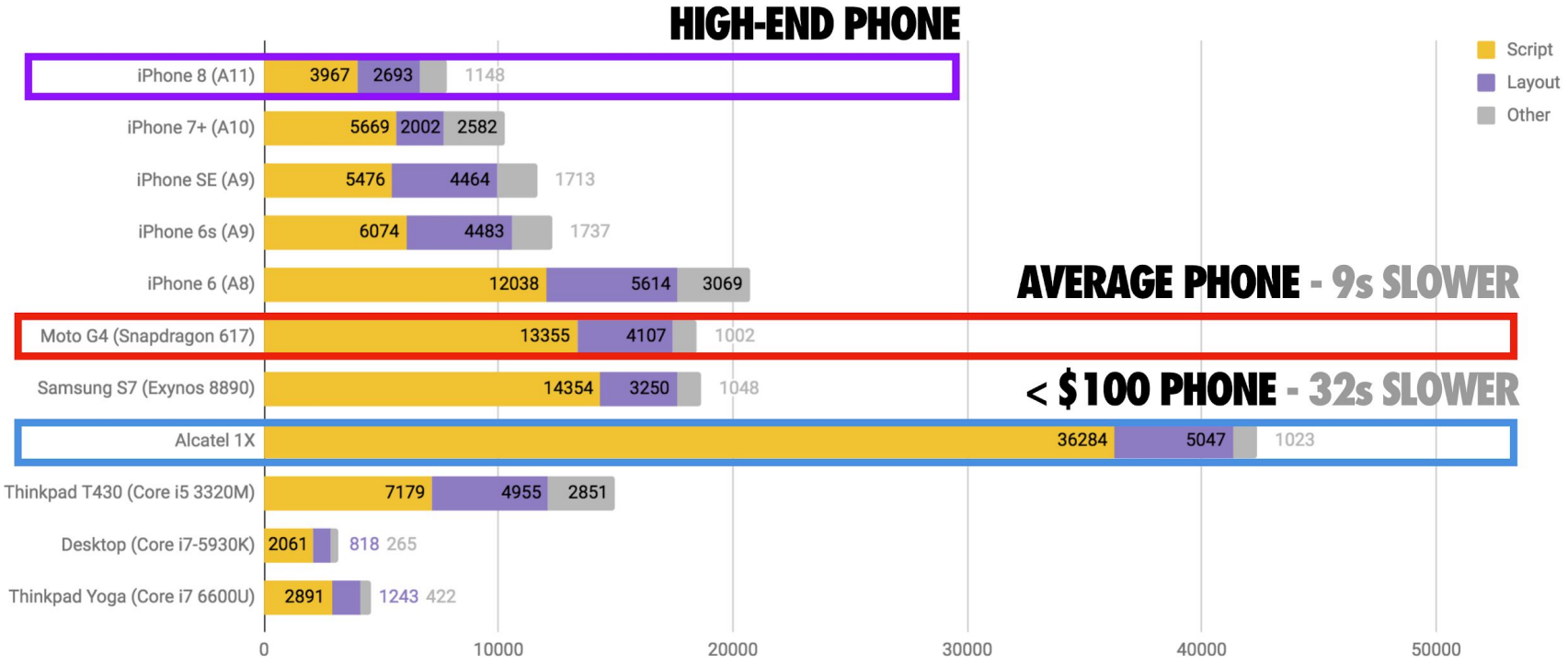
https://medium.com/@addyosmani/the-cost-of-javascript-in-2018-7d8950fbb5d4

ACQUIA

THE COST OF JAVASCRIPT

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

JS PROCESSING FOR CNN.COM



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

ACQUIA

THE MEDIAN WEBPAGE

350KB
JAVASCRIPT
>1MB UNCOMPRESSED

15s
UNTIL INTERACTIVE

Statistics from httparchive.org

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

ACQUIA

MOBILE IS A SPECTRUM

$30

LOW-END

< $200

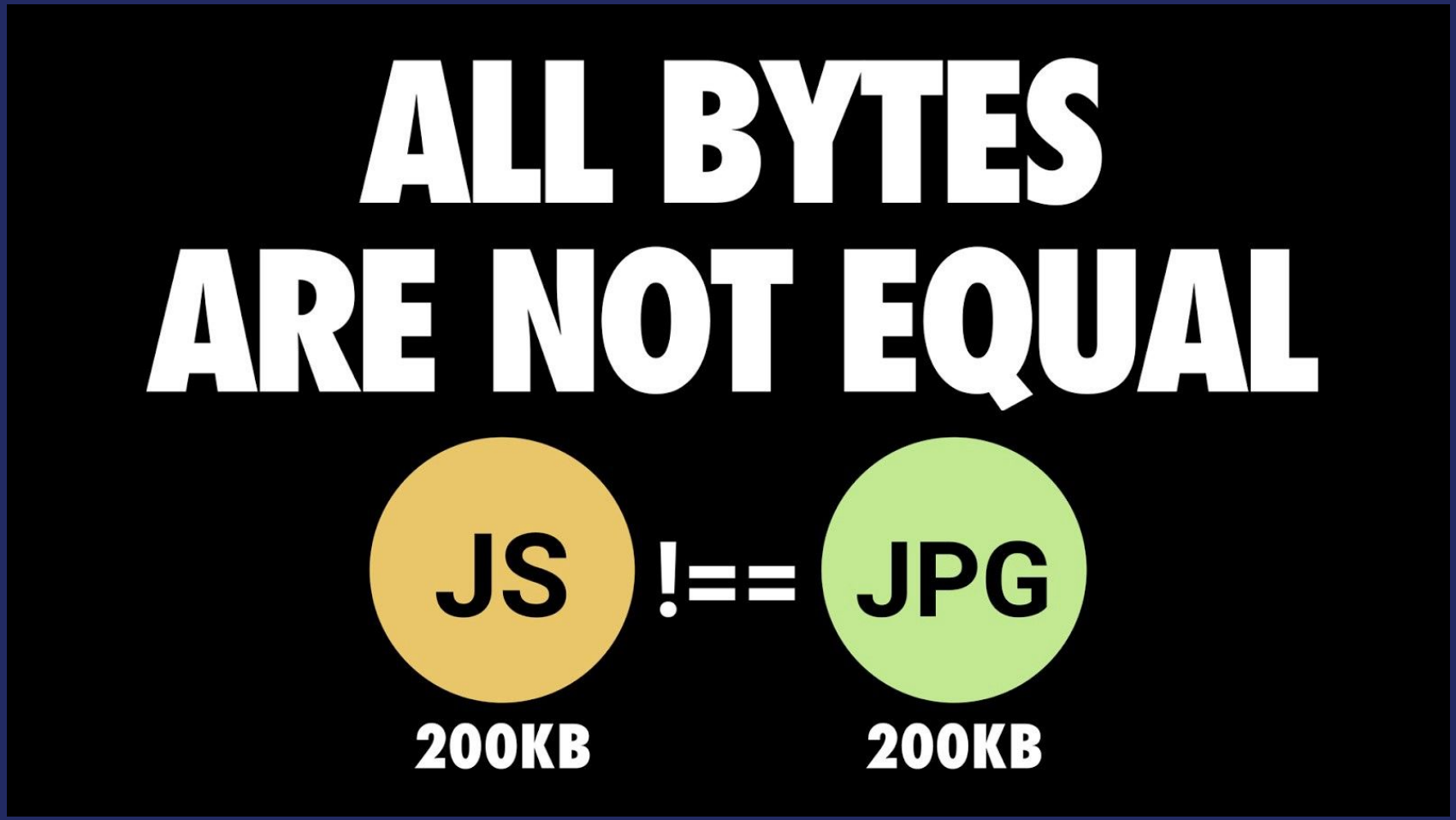
MEDIAN

$1K

HIGH-END

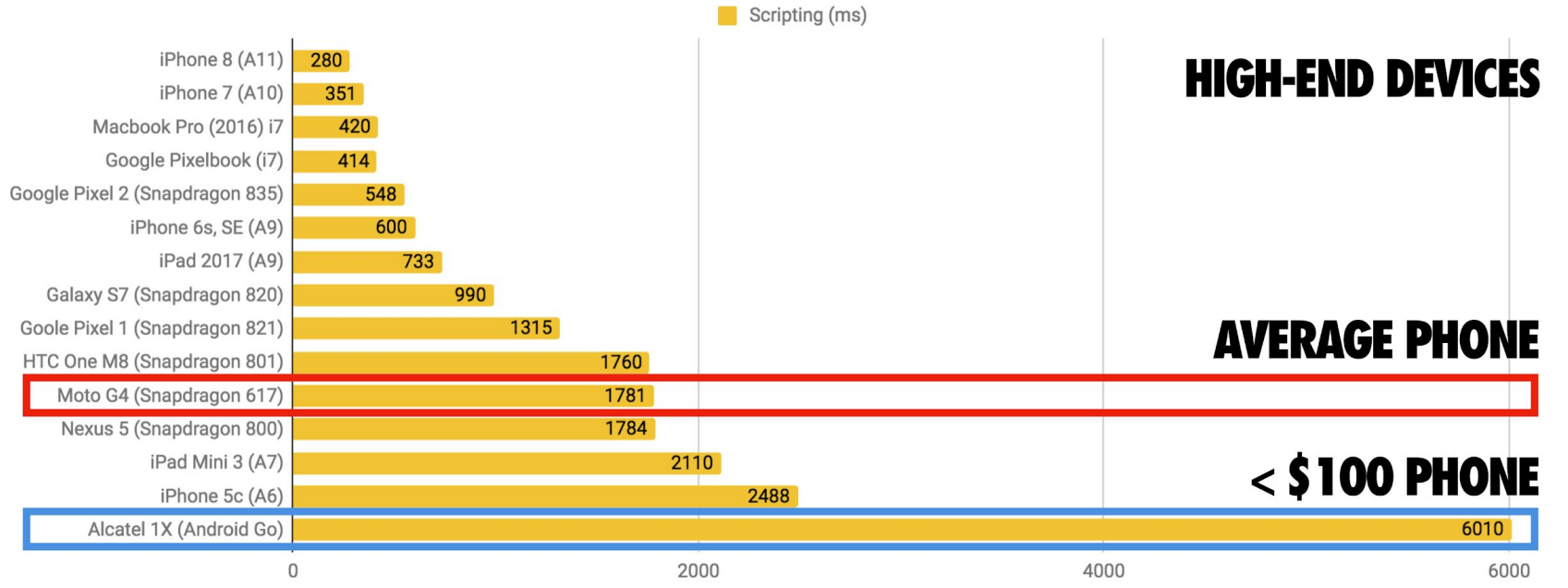
https://medium.com/@addyosmani/the-cost-of-javascript-in-2018-7d8950fbb5d4

ACQUIA



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

2018 JAVASCRIPT PROCESSING TIMES



Tests run during July, 2018 on hardware running the latest versions of Android and iOS available

1MB JS UNCOMPRESSED (200KB min/compressed)

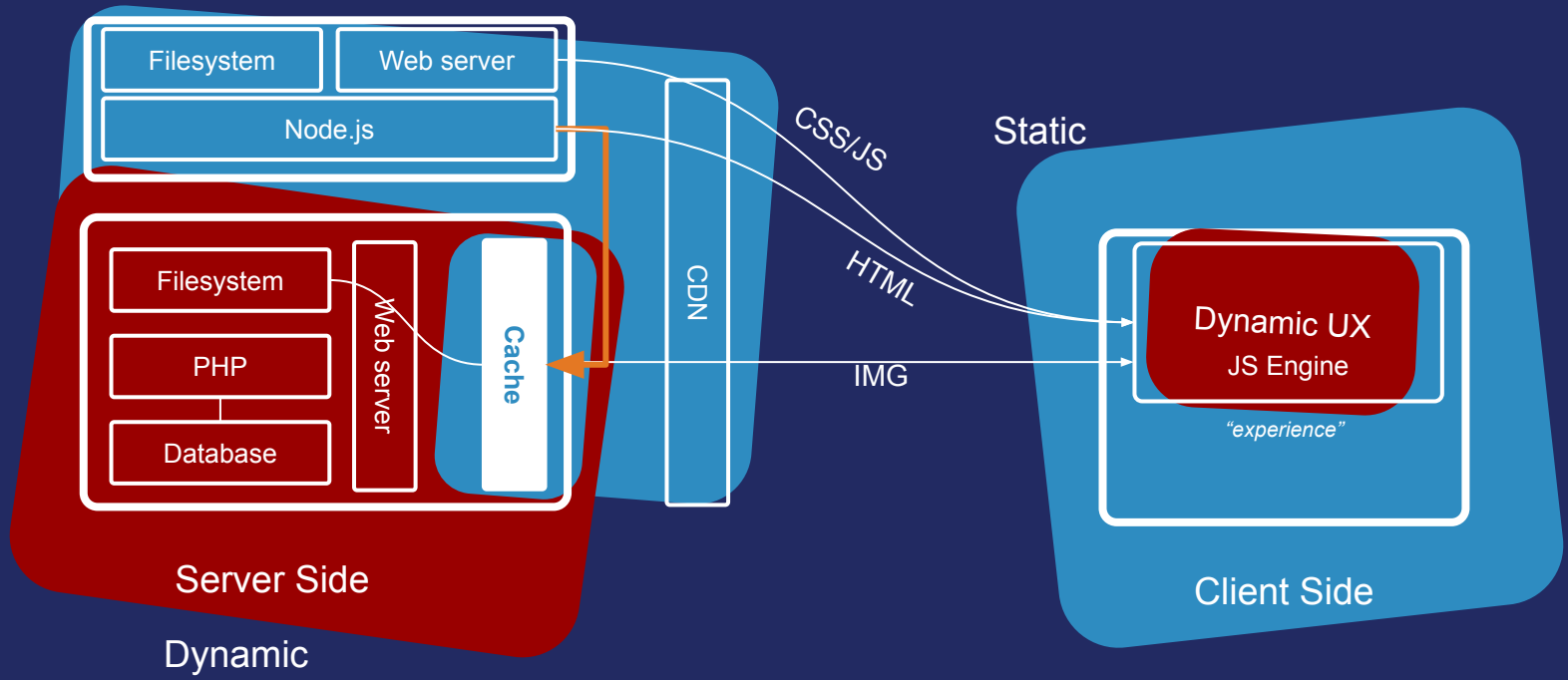
https://medium.com/@addyosmani/the-cost-of-javascript-in-2018-7d8950fbb5d4

ACQUIA

再発明 - サーバーサイドレンダリング(SSR)

Reinventing

Server-side rendering

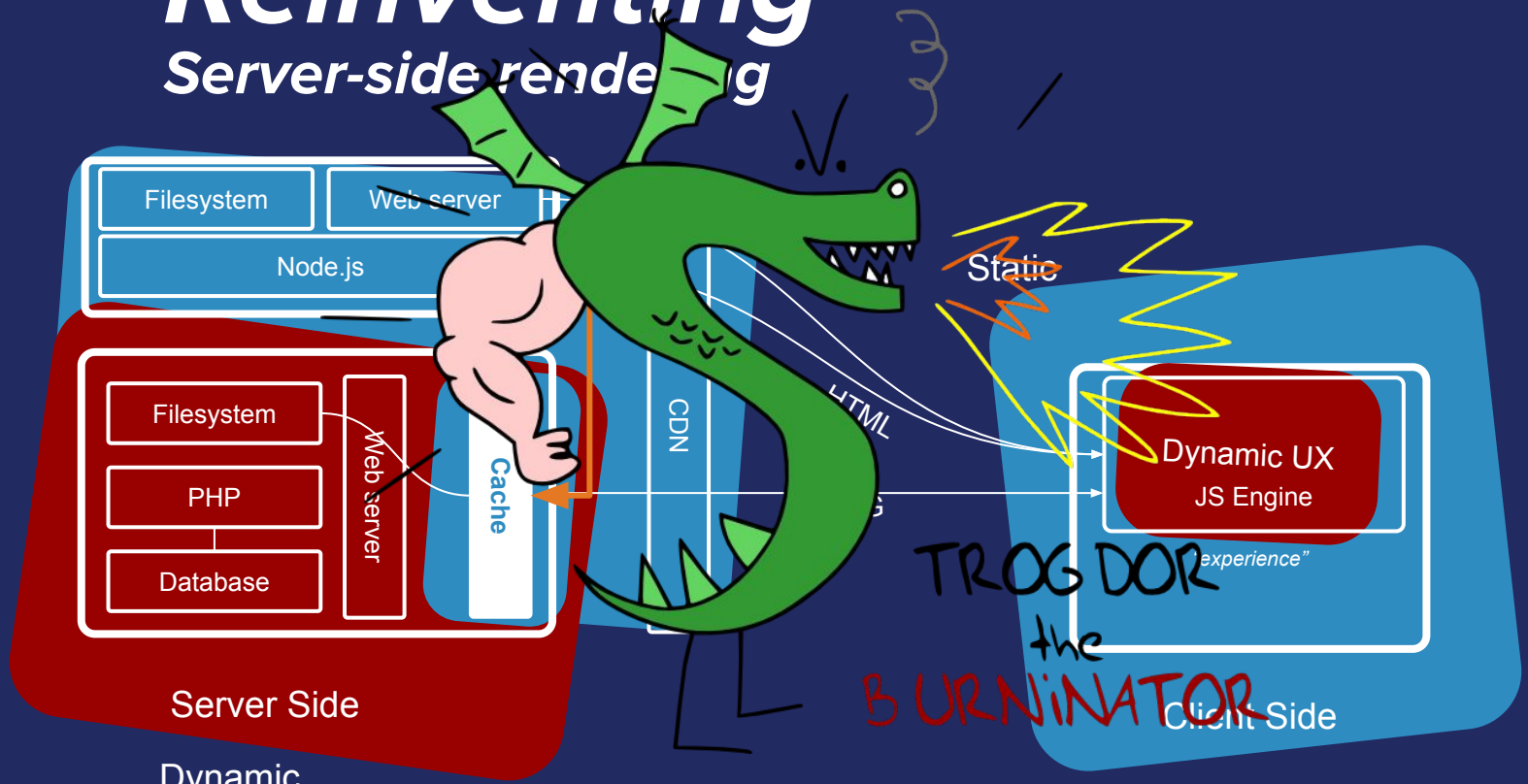


ACQUIA

再発明 - サーバーサイドレンダリング(SSR)

Reinventing

Server-side rende g

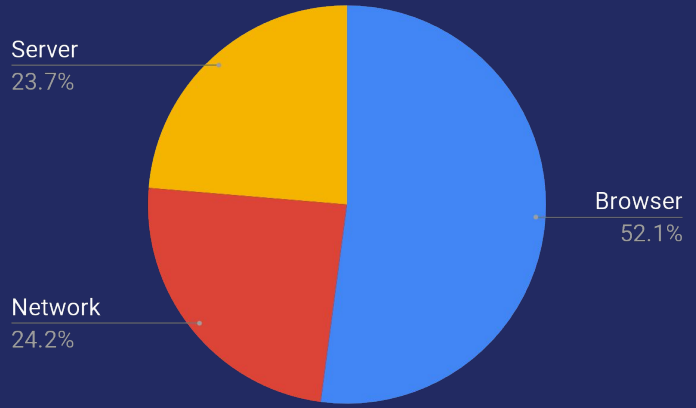


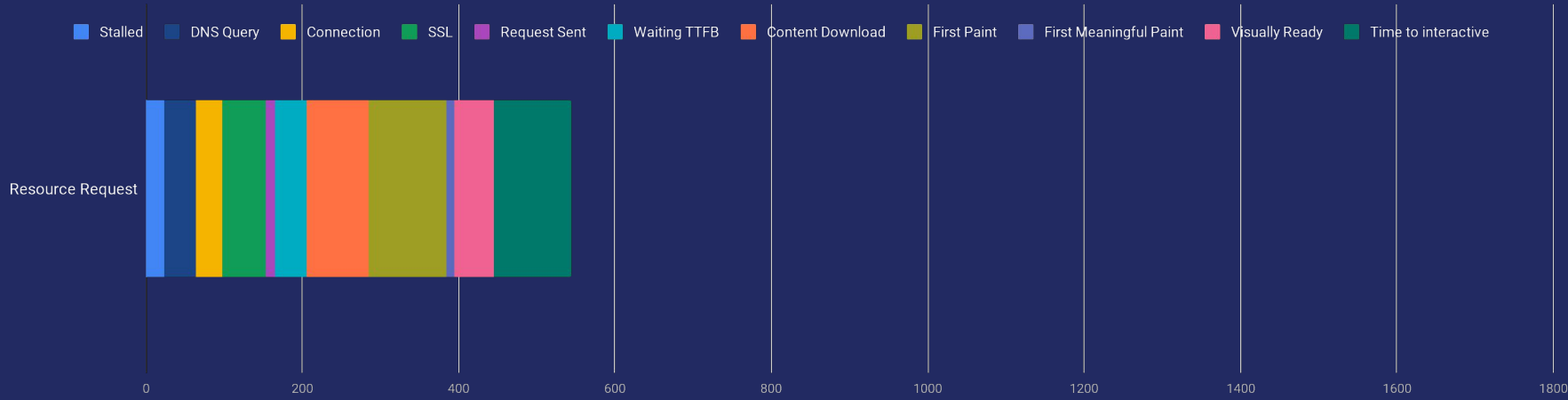
ACQUIA

Loading Journey

This looks suspiciously familiar...

これは非常に疑わしい...





ACQUIA

完全なデカップルド設計 - メリット/デメリット

Fully Decoupled

Pros/Cons

Pros メリット

- Content Management
  コンテンツマネジメント
- Optimised Content Delivery
  コンテンツ配信の最適化
- Centralised processing
  集中処理
- Scale simple content
  シンプルなコンテンツのスケール

Cons デメリット

- Polylithic
  ポリリシック
- Backend Complexity
  バックエンドの複雑さ
- No cache invalidation
  キャッシュ無効化なし
- Slow TTFB?
  遅いTTFB？

ACQUIA

車輪の再発明 - サーバーサイドのコンテンツ配信

Reinventing the Wheel

For Server-side Content Delivery

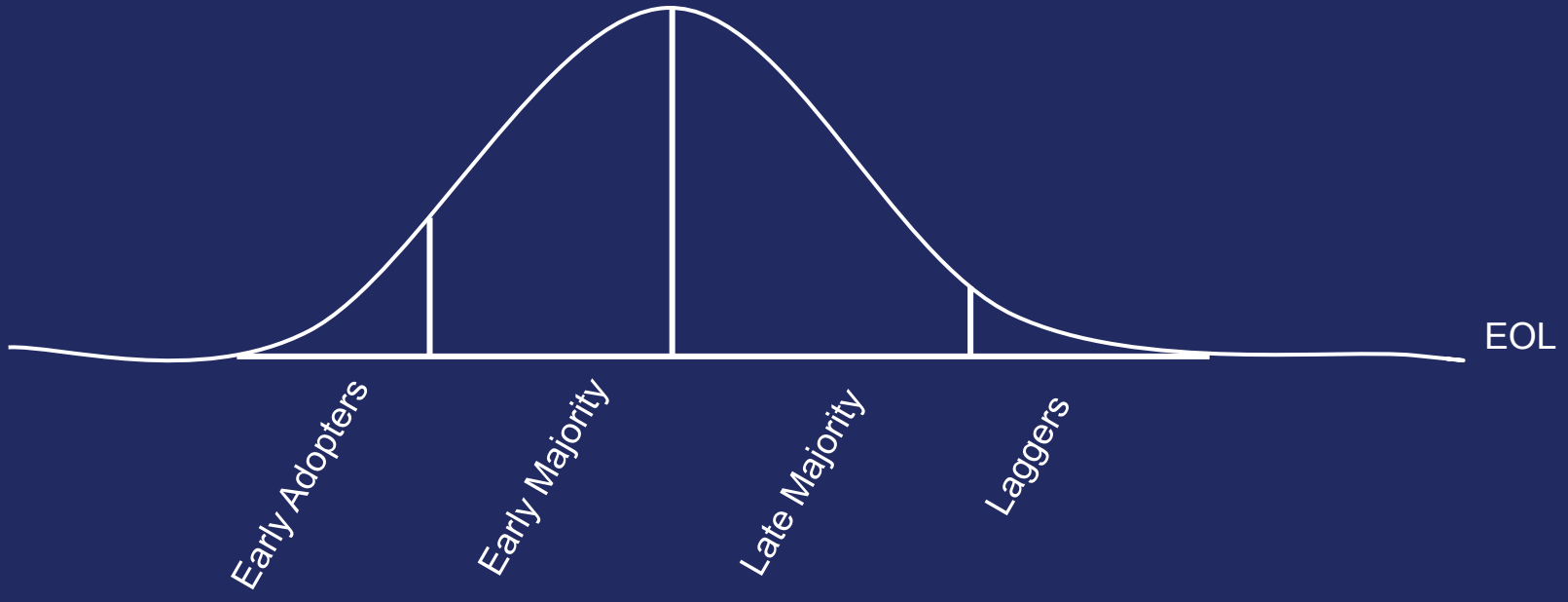
PHP

Node.js

Acquia

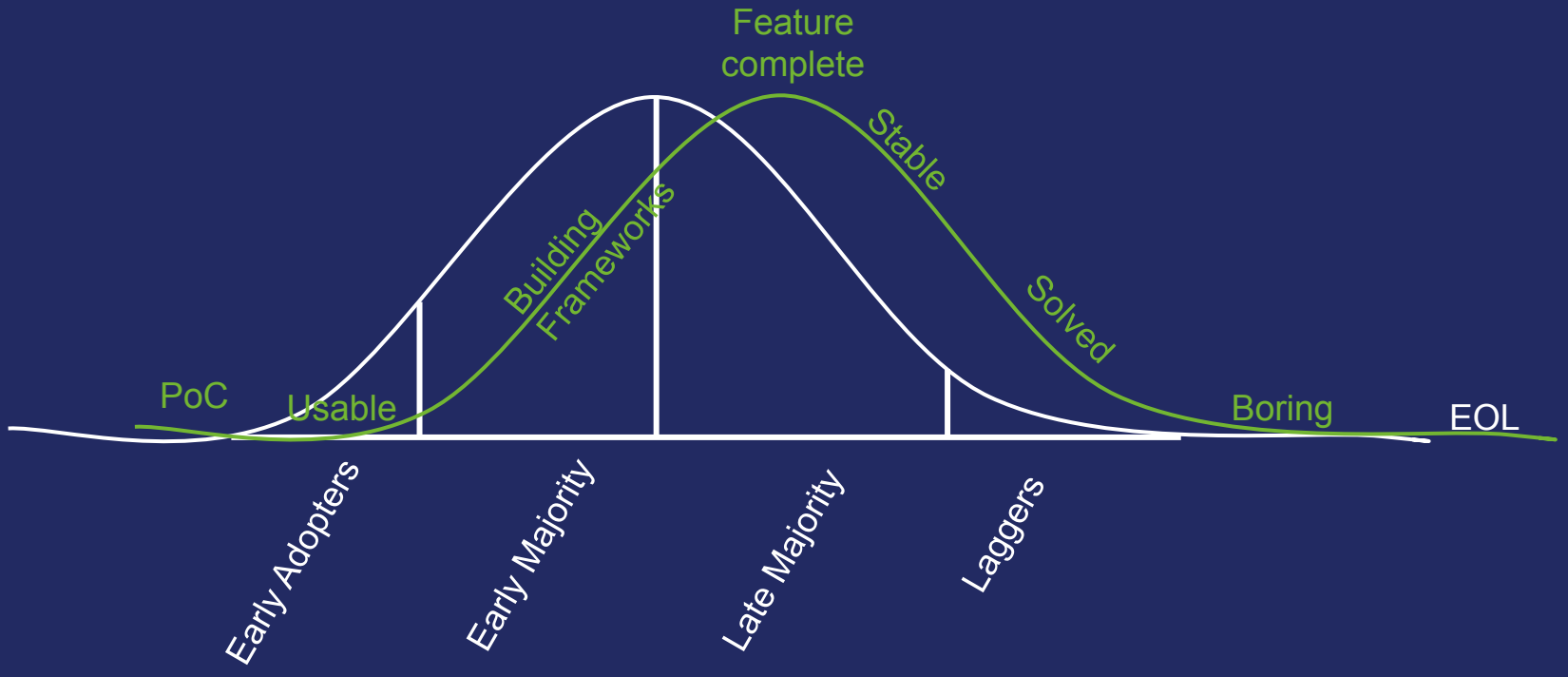
Technology Maturity

テクノロジーの成熟度



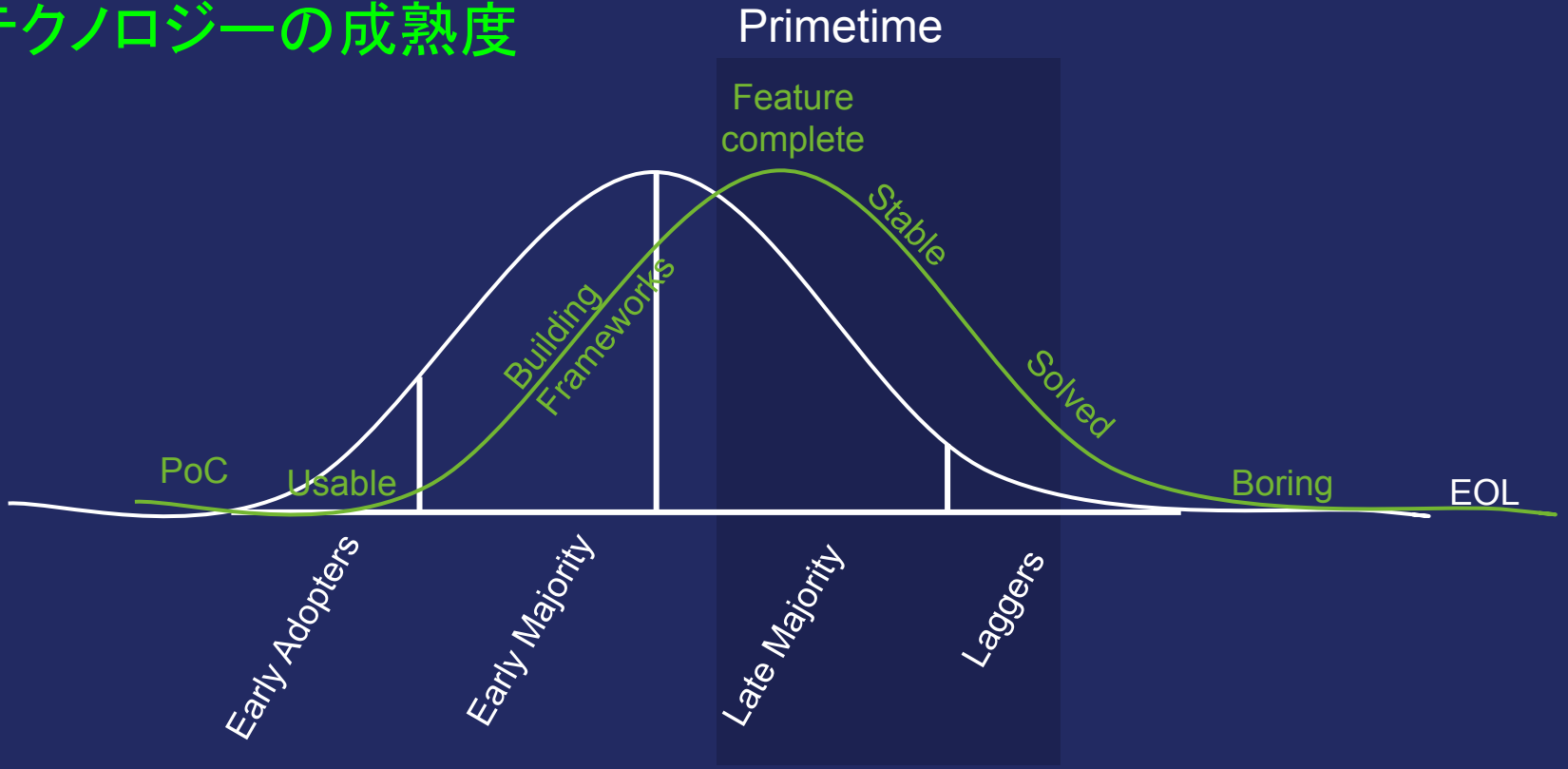
Technology Maturity

テクノロジーの成熟度



Technology Maturity

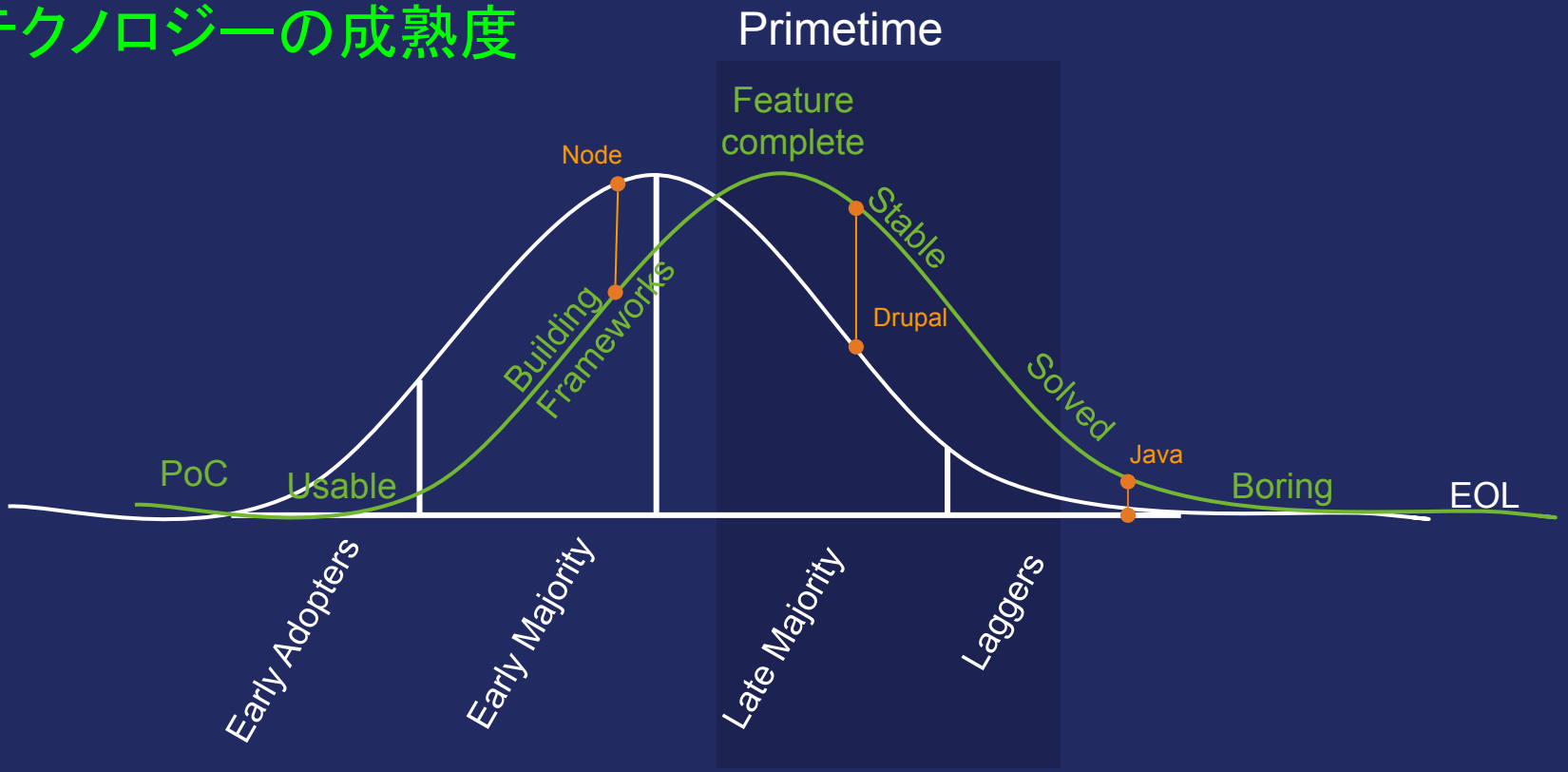
テクノロジーの成熟度



ACQUIA

Technology Maturity

テクノロジーの成熟度



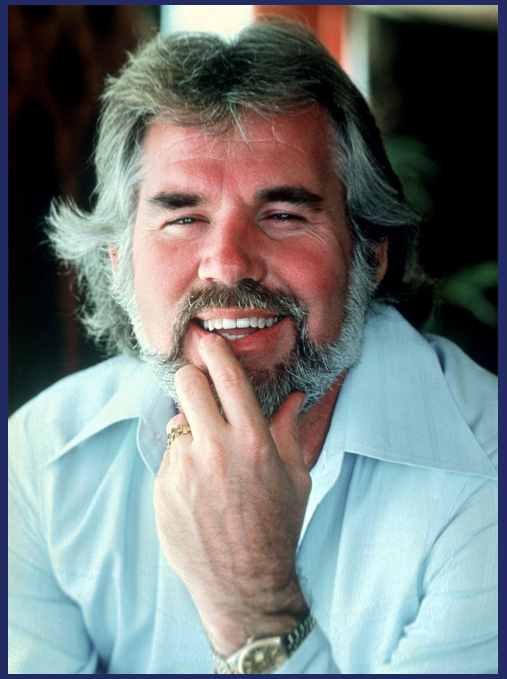
ACQUIA

より良い車輪を再発明する

REINVENT A BETTER WHEEL

(To the tune of the gambler)

You gotta know when to decouple	いつ分離するか、いつ結合するか
know when to couple	いつフロントエンド ダイナミズムが
know when you need FE dynamecy	必要になるのか
know when you don't	知らない時に知っておく必要がある
You gotta count your node_modules	PRがマージされる時
when your setting on the PR merge	node_modulesを数える必要がある
They'll be time enough for rendering	コンパイルが完了したら
when the compiling's done	レンダリングに十分な時間になります



ACQUIA

デカップリングのヒント

Decoupling tips

- *Performance is worse in a fully decoupled architecture.*
  完全に分離された(デカップルド)設計では、パフォーマンスが低下
- *Start coupled and validate reasons to decouple.*
  まずはカップルド設計ではじめて、分離する理由を検証する
- *Polylithic delivery increases operational costs exponentially.*
  ポリリシック配信は、運用コストを指数関数的に増加させる
- *Clarify editorial requirements up front.*
  編集要件を前もって明確にする
  - *Editorial layout management is hard in a decoupled architecture.*
    デカップルド設計では、編集レイアウトの管理は困難
- *DO NOT DIY API. Use JSON:API or GraphQL.*
  DIY(自家製)APIはやめて、JSON:APIまたはGraphQLを使う

ACQUIA


Acquia