SWCON201
Opensource & Software
Development Methods and Tools

Problem Definition

Department of
Software Convergence

경희대학교
KYUNG HEE UNIVERSITY

Contents

- Agenda
- Class
- Summary
- Reference

Problem Resolving in SWCON

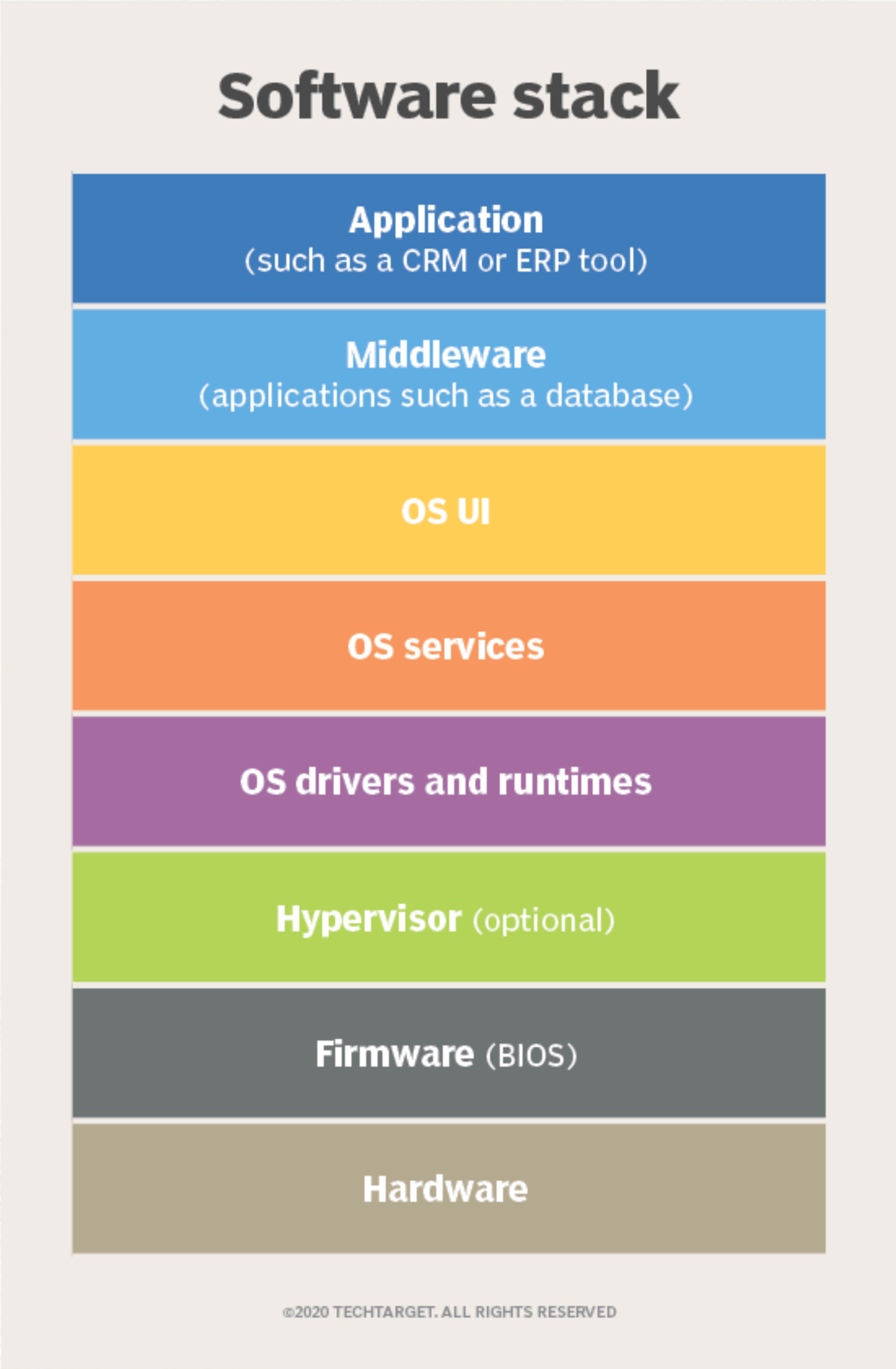
- 웹파이선 프로그래밍 (1Y-1H)
- 디자인적 사고 (1Y-2H)
- 소프트웨어융합개론 (1Y-2H)
- 트랙별 캡스톤디자인 (3Y-2H)
- 소프트웨어융합 캡스톤디자인 (4Y-1H)
- 졸업논문 (4Y-2H)
- & 프로젝트 과제(들)

이번 챕터는 소프트웨어를 개발함에 있어서 고려해야하는 "문제"를 다루는 측면에서 "문제 해결" 교과목과 다름

Problem Definition (software layer)

- System software
- Application software

System software vs Application software



Reference: https://www.techtarget.com/whatis/definition/system-software

System software vs Application software

System software vs. application software

System software	Application software
General-purpose software that manages basic system resources and processes	Software that performs specific tasks to meet user needs
Written in low-level assembly language or machine code	Written in higher-level languages, such as Python and JavaScript
Must meet specific hardware needs; interacts closely with hardware	Does not take hardware into account and doesn't interact directly with hardware
Installed at the same time as the OS, usually by the manufacturer	User or admin installs software when needed
Runs any time the computer is on	User triggers and stops the program
Works in the background and users don't usually access it	Runs in the foreground and users work directly with the software to perform specific tasks
Runs independently	Needs system software to run
Is necessary for the system to function	Isn't needed for the system to function

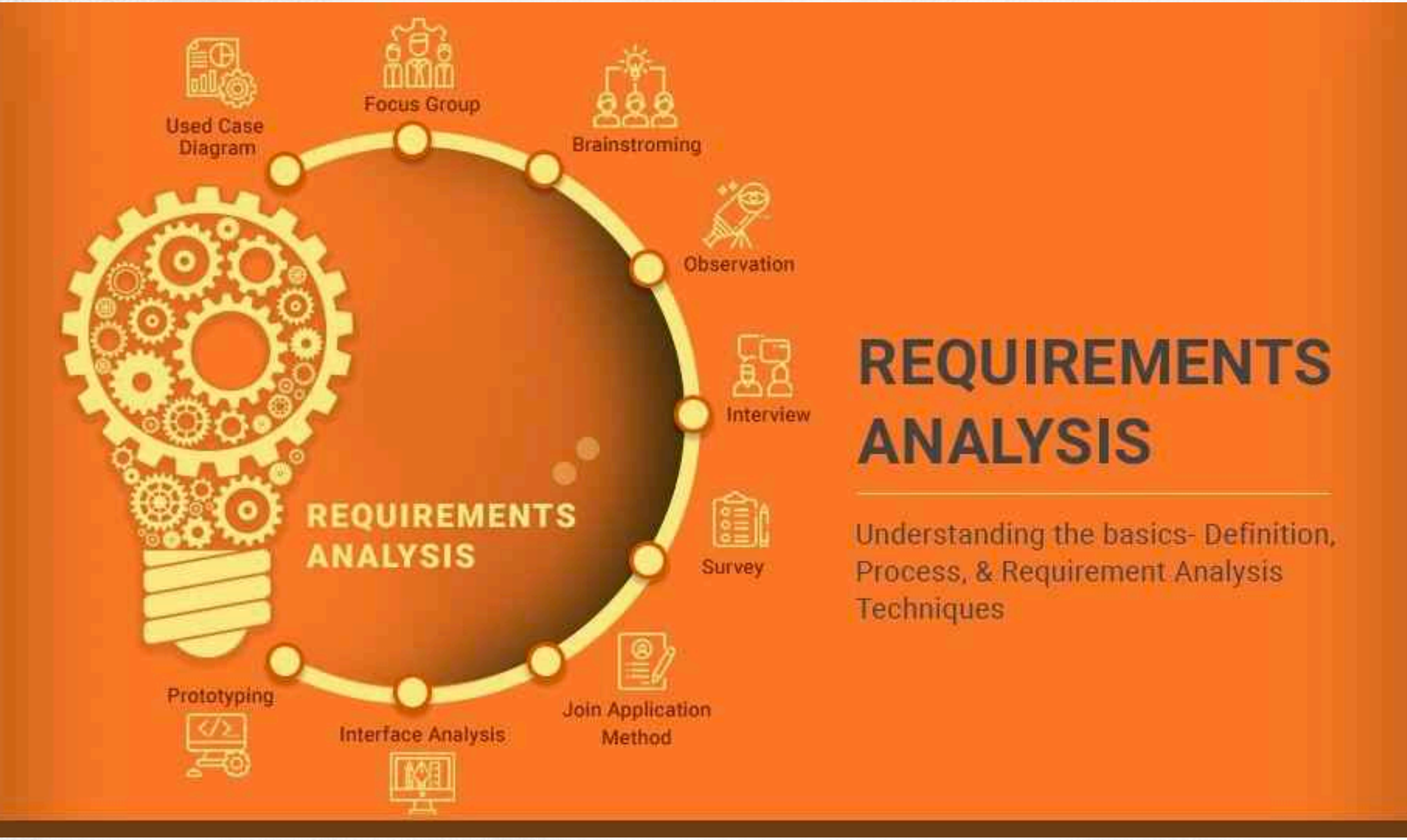
©2021 TECHTARGET. ALL RIGHTS RESERVED TechTarget

Reference: https://www.techtarget.com/whatis/definition/system-software

Problem Definition (purpose)

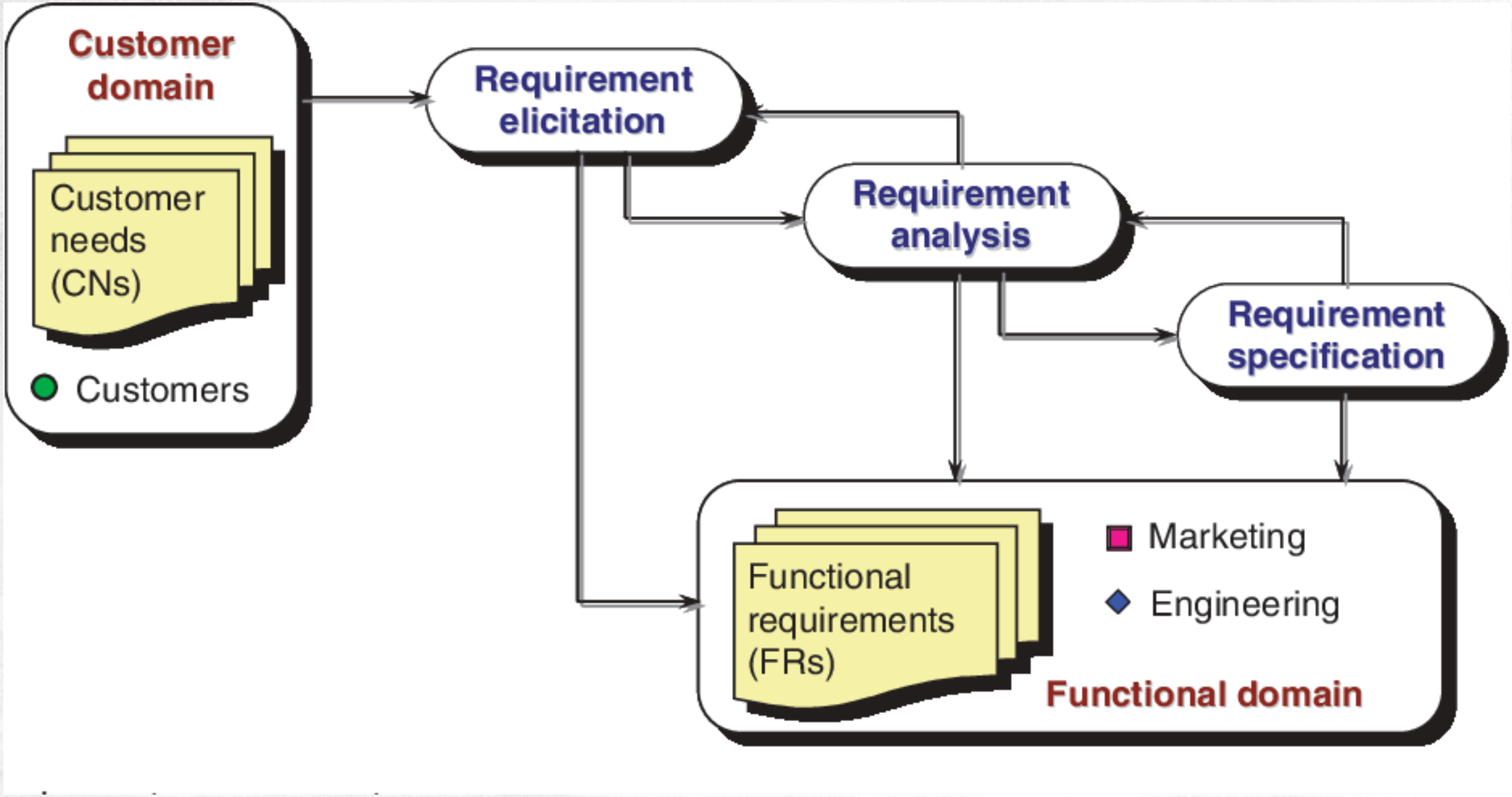
- Purely your own purpose
- Based on other person's request

Customer Requirement Analysis



Reference: https://reqtest.com/requirements-blog/requirements-analysis/

Customer Requirement Analysis

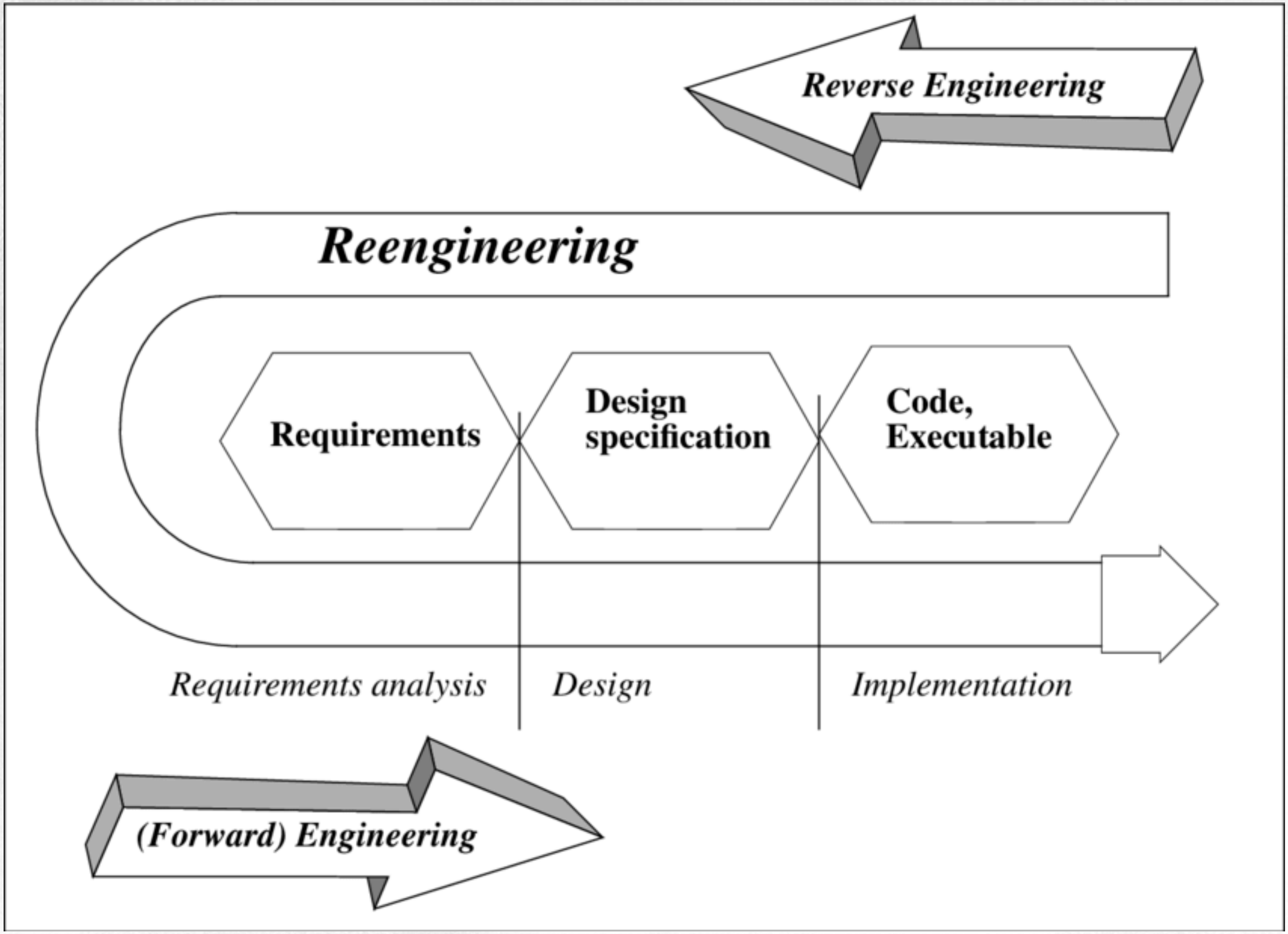


Reference: https://www.semanticscholar.org/paper/Customer-Requirement-Management-in-Product-A-Review-Jiao-Chen/4e4b12e567a65761f8f7bdc73ea8b9165430c9fb/figure/0

Problem Definition (legacy)

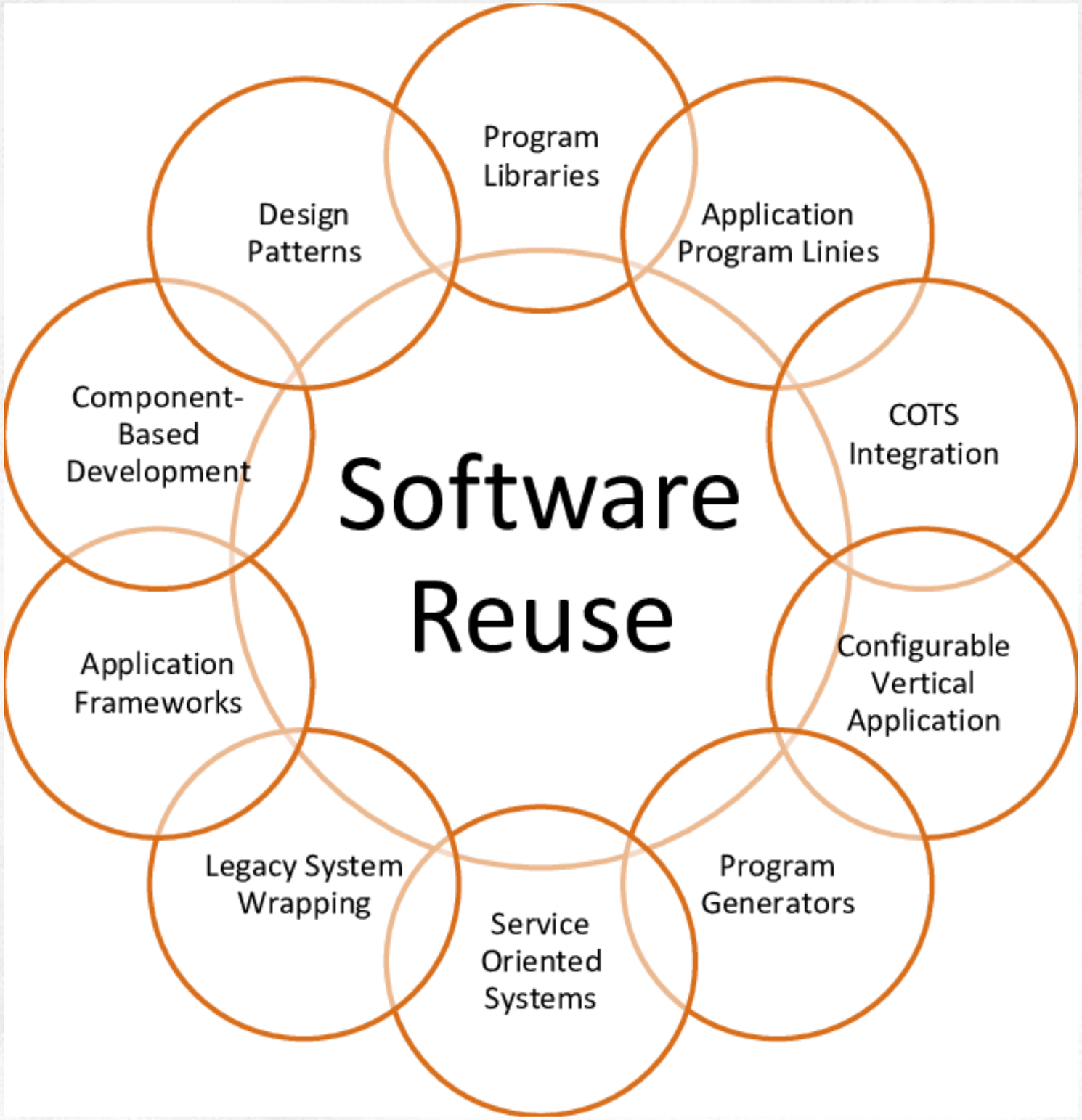
- From scratch
- Based on existing platform(s)

Reverse Engineering (Reengineering)



Reference: https://www.researchgate.net/figure/Software-engineering-reverse-engineering-and-reengineering_fig1_2799789

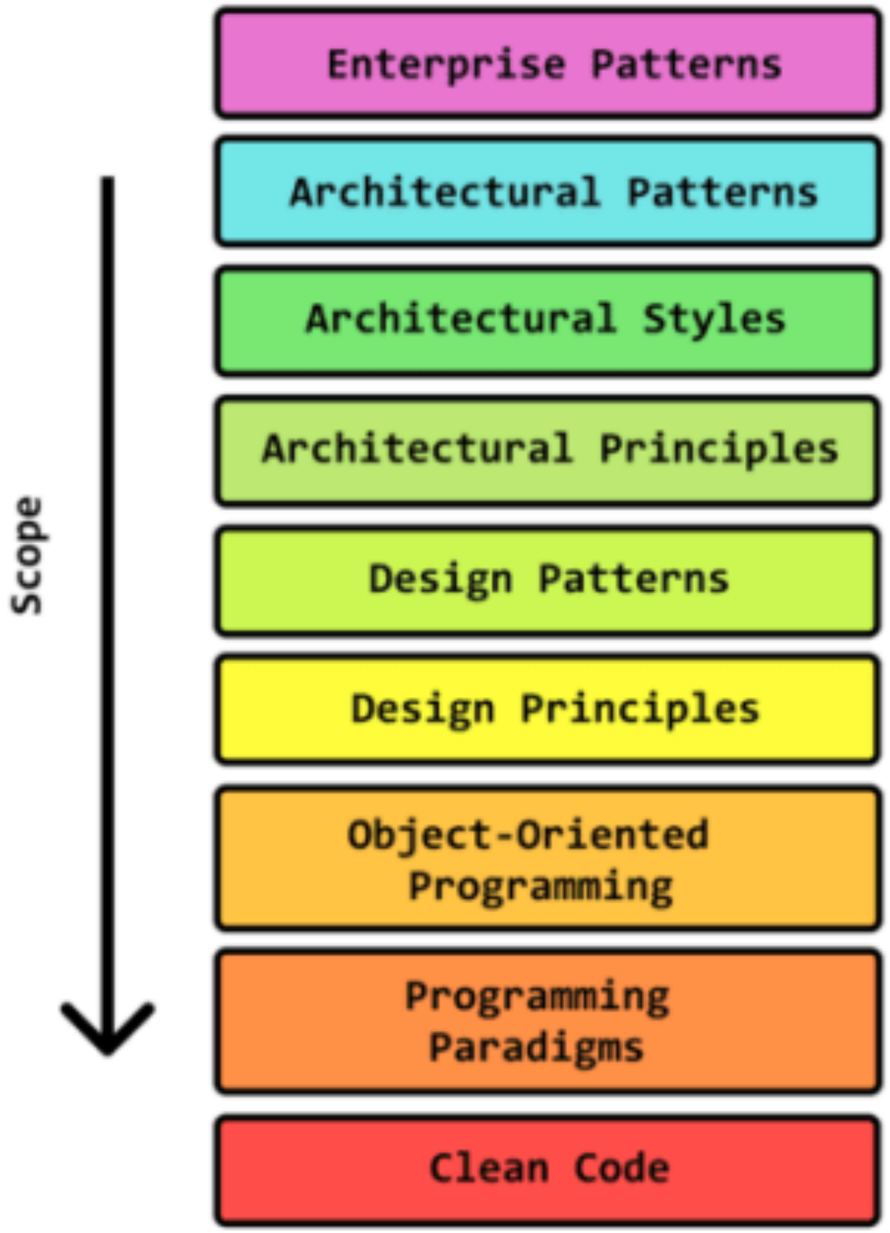
Software Reuse



Reference: https://www.researchgate.net/figure/Software-engineering-reverse-engineering-and-reengineering_fig1_2799789

Design Patterns

The Software Design & Architecture Stack



DTOs, Domain-Models, Transaction Scripts, Repositories, Mappers, Value Objects

Model-View-Controller, Domain-Driven Design

Layered, Client-Server, Monolithic, Component-based

Policy vs. details, Coupling & cohesion, dependencies, boundaries

Observer, Strategy, Factory, etc

Composition Over Inheritance, Hollywood Principle, encapsulate what varies, SOLID, DRY, YAGNI

Inheritance, Polymorphism, Encapsulation, Abstraction

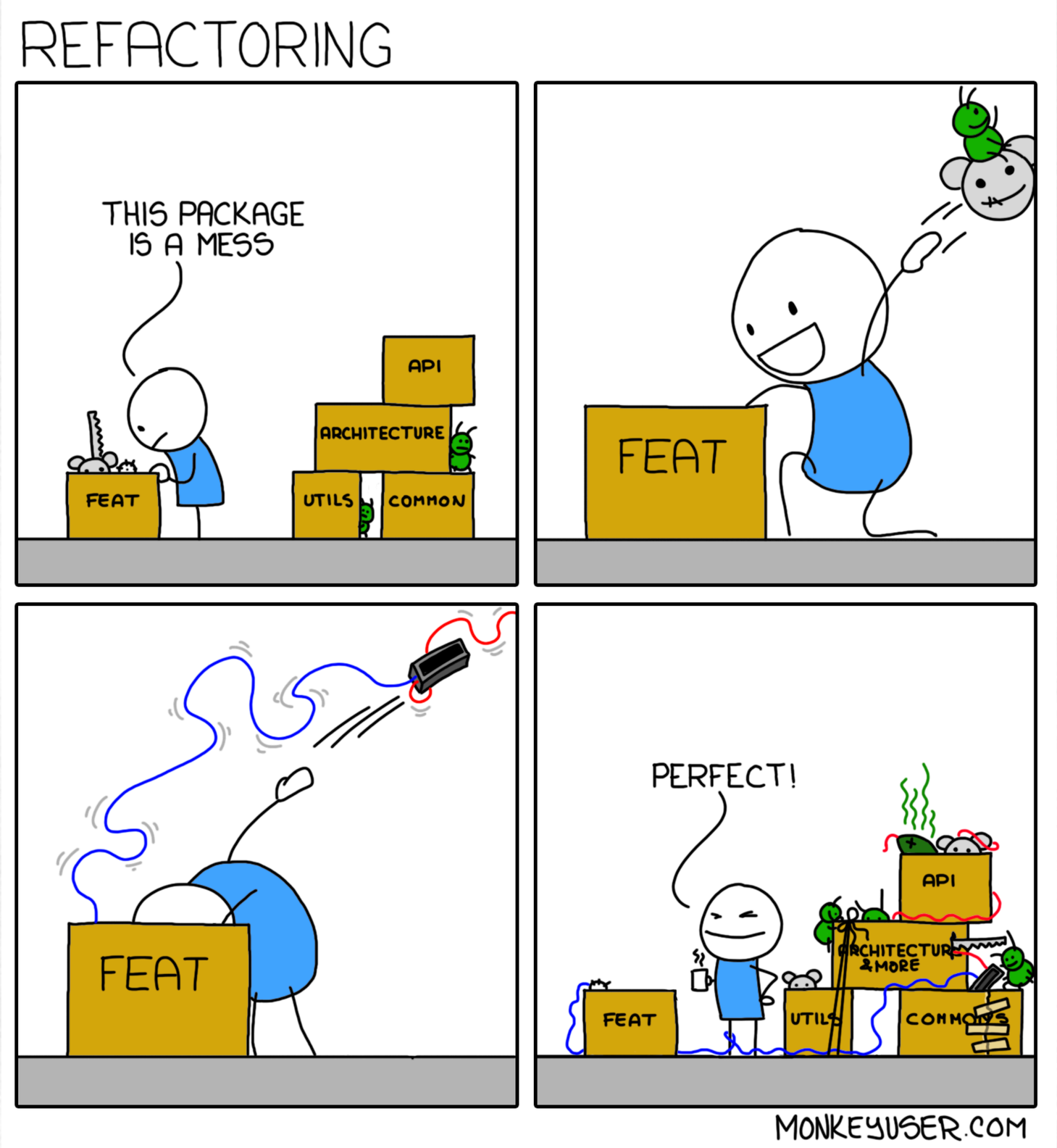
Structured, Object-Oriented, Functional

Name, construct, structure, style, readability

Image Credits: Khalil Stemmler

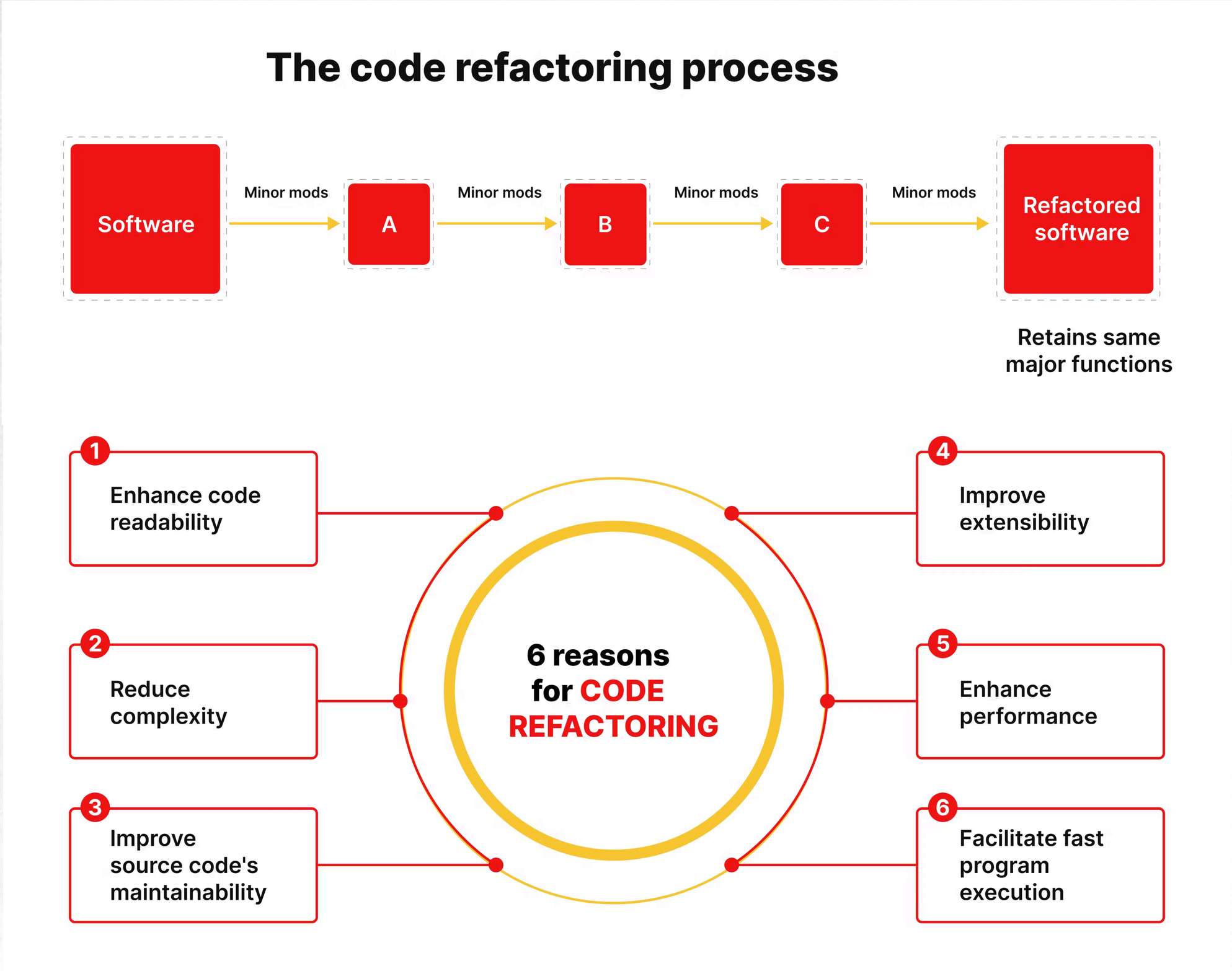
Reference: https://dzone.com/articles/what-is-design-pattern

Refactoring



Reference: https://www.monkeyuser.com/2018/refactoring/

Refactoring



Reference: https://maddevs.io/blog/code-refactoring/

Problem Definition (project characteristics)

- Due (Service deployment date)
- Performance (Capacity, Response Time, etc.)
- Resource (Number of developers, Developer skill, Money, etc)

Performance indicator (or KPI)

- A performance indicator or key performance indicator (KPI) is a type of performance measurement.
- KPIs evaluate the success of an organization or of a particular activity (such as projects, programs, products and other initiatives) in which it engages.
- KPIs provides a focus for strategic and operational improvement, create an analytical basis for decision making and help focus attention on what matters most.

Reference: https://en.wikipedia.org/wiki/Performance_indicator

KPI example

KPI GROUP	METRICS	TARGETS PER MONTH
CODE QUALITY	Maintainability index	Between 20 and 100: higher values are better
	Cyclomatic complexity	1 < target < 10
	Depth of inheritance	0
	Class coupling	0
	Lines of code	Depends on an app's complexity but the lower the better
TESTING QUALITY	Test coverage	80% or more + track the general trend
	Defects found in UAT	30 or less + track the general trend
	Defects found in production	No more than 10% out of all bugs + track the general trend
SOLUTION AVAILABILITY	MTBF	Track the general trend, aim to increase numbers
	MTTR	Track the general trend, aim to decrease numbers
	Page load time	Less than 2 seconds
SOLUTION SECURITY	Number of unavailability cases	1-5 (including deployment unavailability)
	Number of vulnerabilities found by regular penetration testing	0-3 + track the general trend
	Number and severity of security incidents	0-1 for High severity incidents and 0-2 for Medium and Low severity incidents
USER SATISFACTION (a survey with rankings from 1 (completely not satisfied) to 5 (completely satisfied)	Meeting user expectations on functionality	Not less than 5
	User interface convenience	Not less than 5
	Stability of software performance	Not less than 4

Reference: https://www.scnsoft.com/blog/software-development-kpis

KPI example

A SAMPLE KPI SYSTEM

Costs

- Budget planned
- Budget spent
- Over-budget approved and spent (if any)
- Time spent by your employees fulfilling

Features (by business value & priority)

- Implemented
- In backlog
- Backlog waiting time expired

Application reliability

- Service availability
- Response time
- Defects found in production

Application security

- Number of vulnerabilities found during regular security testing
- Number of security incidents (if any)

User satisfaction & engagement

- User satisfaction score
- Percentage of active users daily, weekly and monthly
- Average time spent on an application per day

Software quality

- Coverage by unit tests
- Share of passed test cases
- Maintainability index, etc.

Reference: https://www.mediacategory.com/

Problem Definition (hardware dependency)

- Hardware dependent
- Hardware in-dependent

Hardware dependent problem

- Case Study
  - Arduino (ATmega, C, Sketch IDE)
  - RaspberryPi (ARM Processor, GPIO, USB)
  - CUDA (NVIDIA GPU, C/C++/Fortran)

Hardware independent problem

- Case Study: Web server development
  - C++ & Proxygen
  - Python & Django
  - Javascript & Node.js
  - Apache/Nginx (tools, non-programming)

- Generally, KPI (Key Performance Index) is considered, such as:
  - Due (Service deployment date)
  - Performance (Capacity, Response Time, etc)
  - Resource (Number of developers, Developer skill, Money, etc)

Case Study: Web server development

- C++ & Proxygen

Proxygen: Facebook's C++ HTTP Libraries

Support Ukraine | linux passing | mac passing

This project comprises the core C++ HTTP abstractions used at Facebook. Internally, it is used as the basis for building many HTTP servers, proxies, and clients. This release focuses on the common HTTP abstractions and our simple HTTPServer framework. Future releases will provide simple client APIs as well. The framework supports HTTP/1.1, SPDY/3, SPDY/3.1, HTTP/2, and HTTP/3. The goal is to provide a simple, performant, and modern C++ HTTP library.

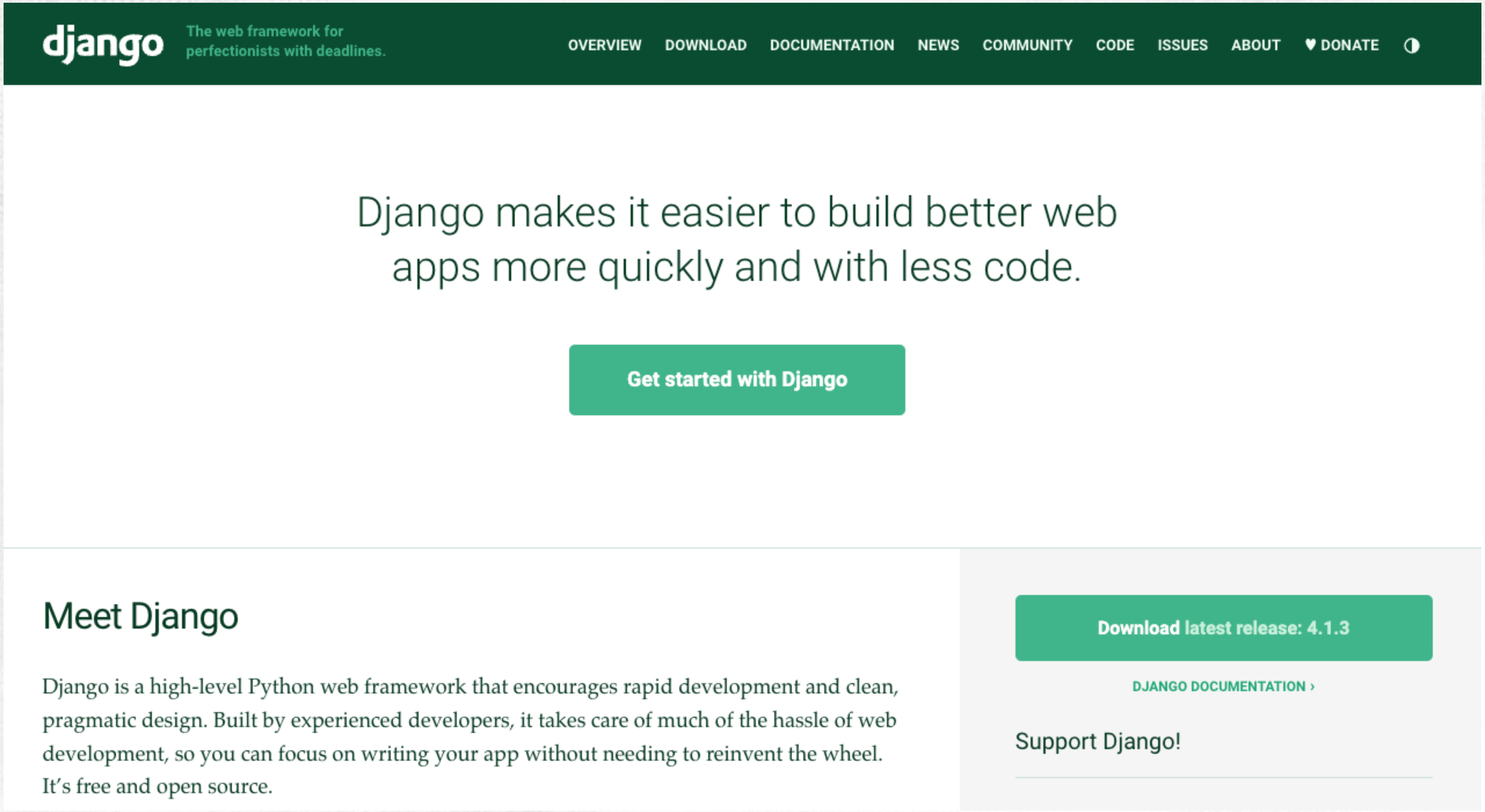
We have a Google group for general discussions at https://groups.google.com/d/forum/facebook-proxygen.

The original blog post also has more background on the project.

Reference: https://github.com/facebook/proxygen

Case Study: Web server development

- Python & Django

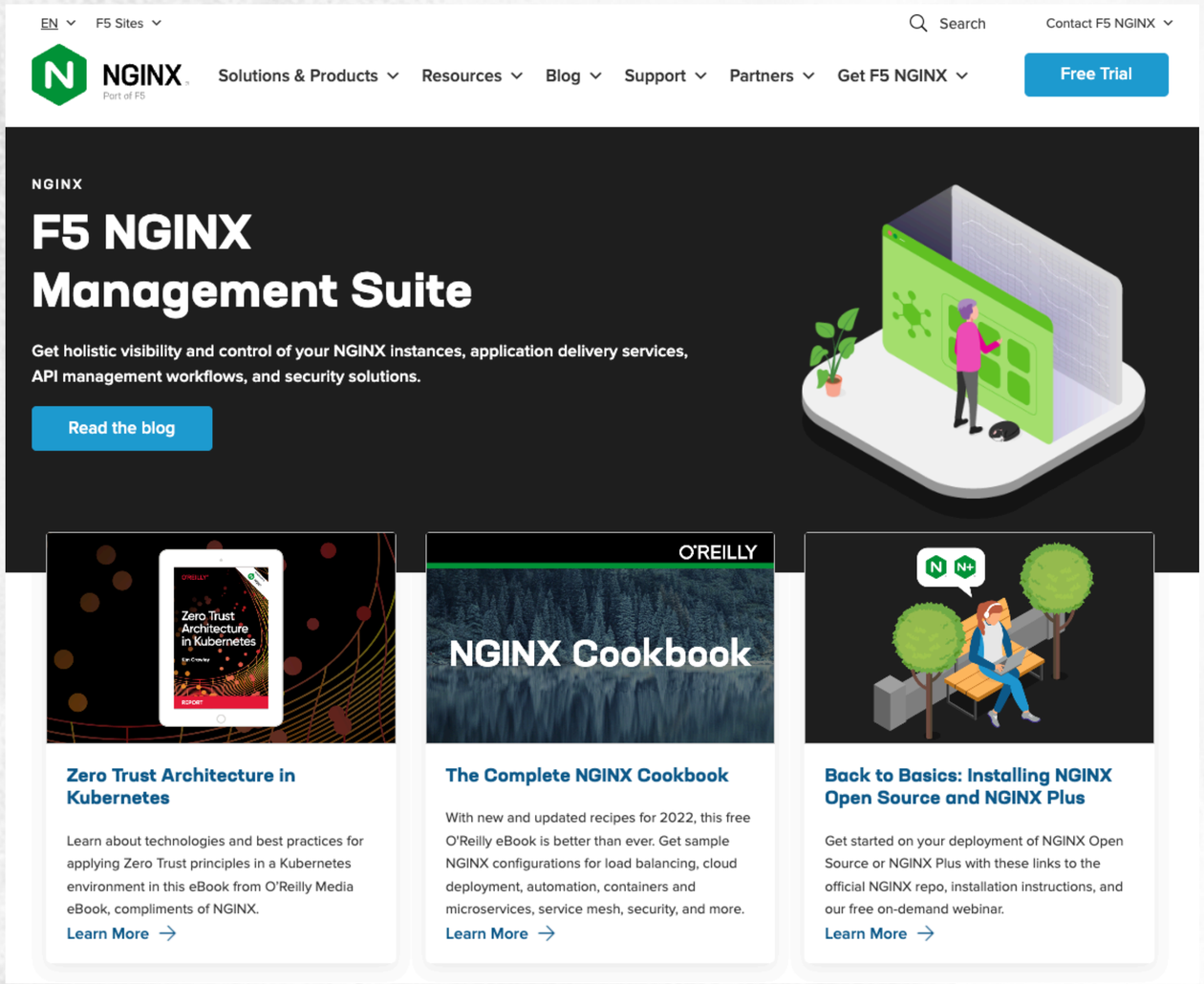


Case Study: Web server development

- Javascript & Node.js

Case Study: Web server development

- Apache/Nginx (tools, non-programming)

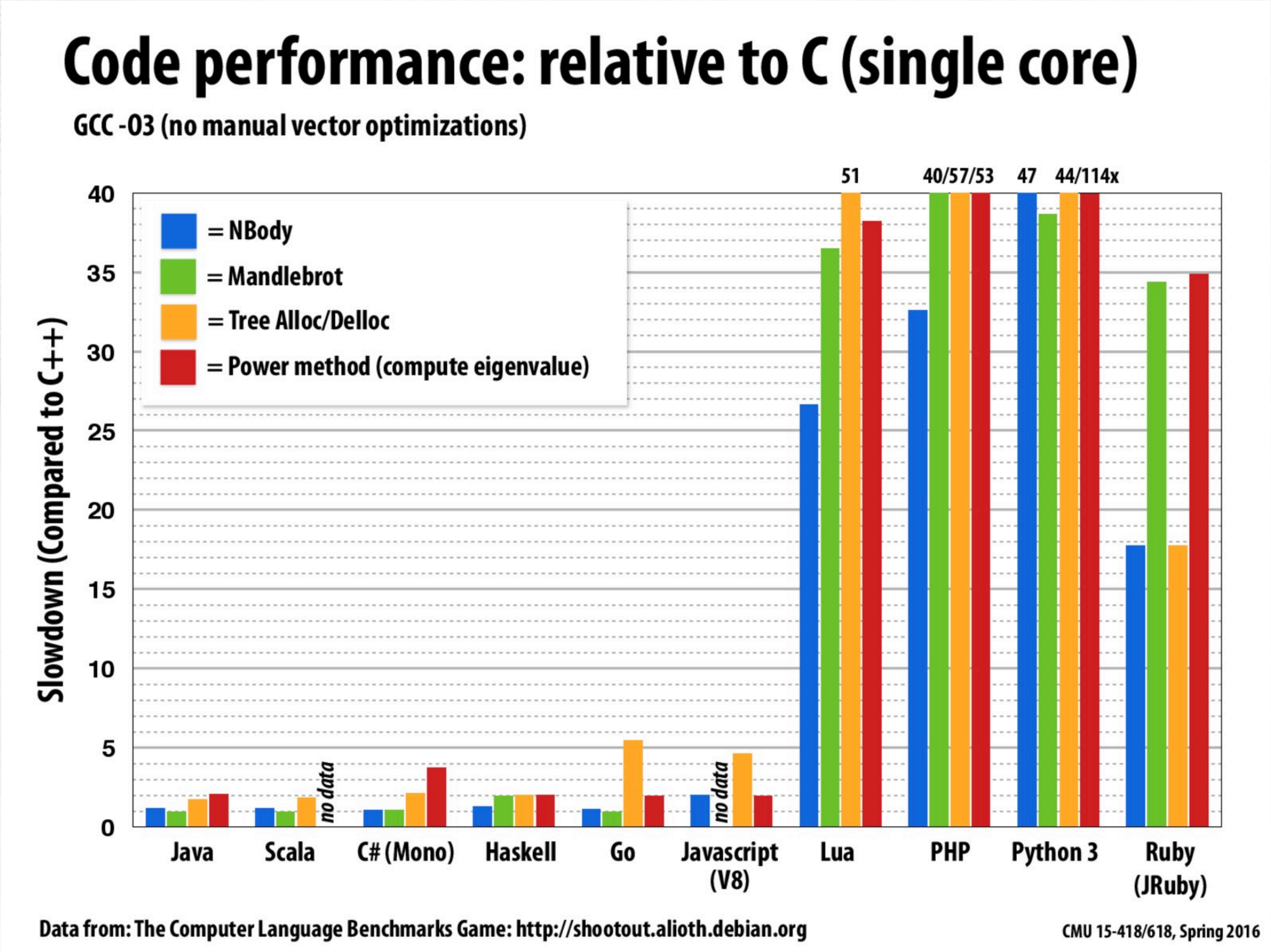


Hardware independent problem

- Case Study: Web server development
  - C++ & Proxygen
  - Python & Django
  - Javascript & Node.js
  - Apache/Nginx (tools, non-programming)

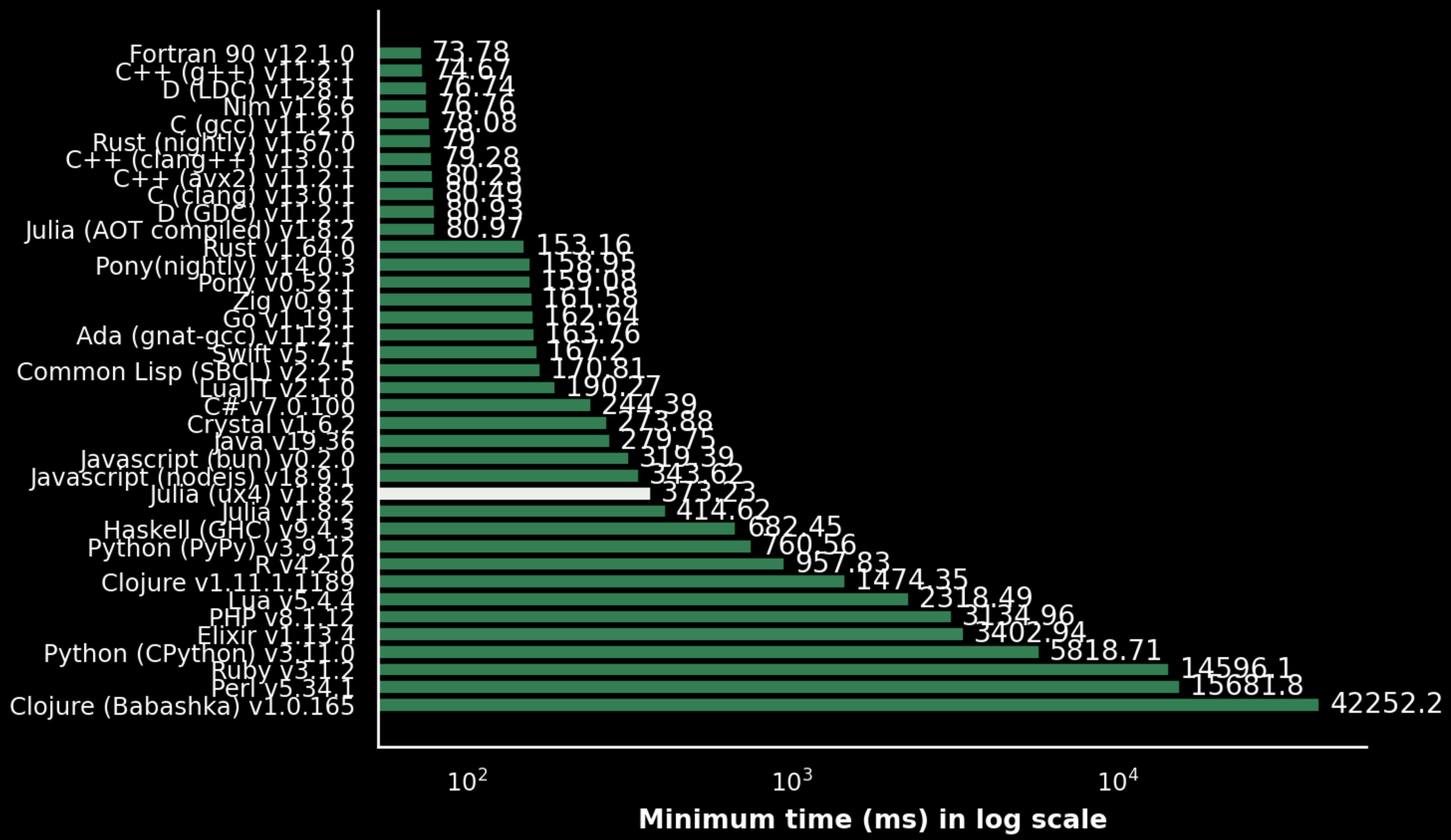
- **Talk about PROs and CONs about:**
  - Due (Service deployment date)
  - Performance (Capacity, Response Time, etc)
  - Resource (Number of developers, Developer skill, Money, etc)

Consideration on performance (time)



Reference: http://15418.courses.cs.cmu.edu/spring2016/lecture/dsl/slide_006

Consideration on performance (time)



Reference: https://github.com/niklas-heer/speed-comparison/blob/master/README.md

Consideration on performance (time)

	Go	Python	C++
fasta	1.26 secs	37.32 secs	0.77 secs
special-norm	1.43 secs	120.99 secs	0.72 secs
n-body	6.38 secs	567.56 secs	2.12 secs

Source: https://benchmarksgame-team.pages.debian.net

codilime
CREATING VALUE

Reference: https://codilime.com/blog/go-vs-python-and-cpp-main-differences/

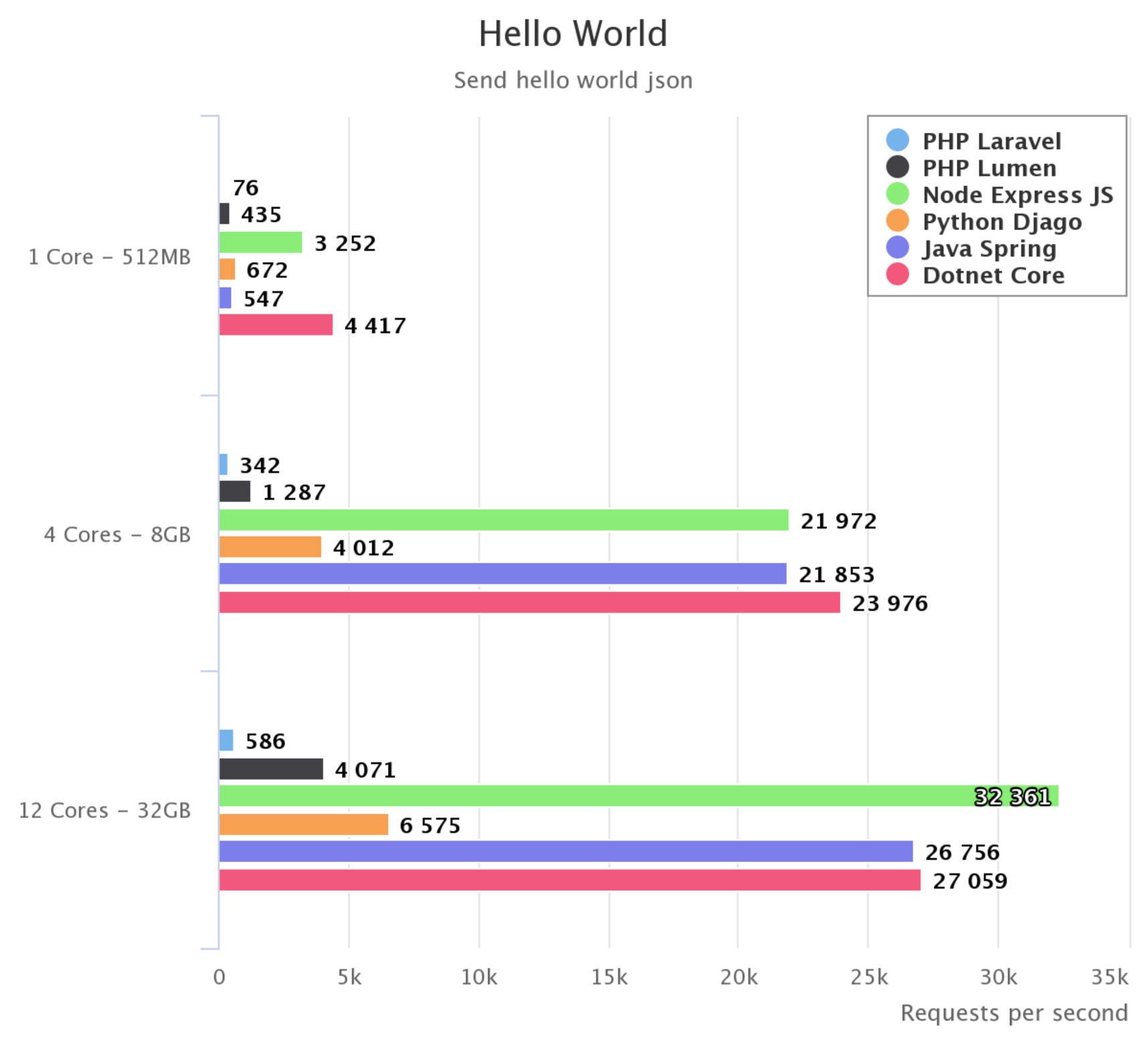
Consideration on performance (overall)

Table 4. Normalized global results for Energy, Time, and Memory

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

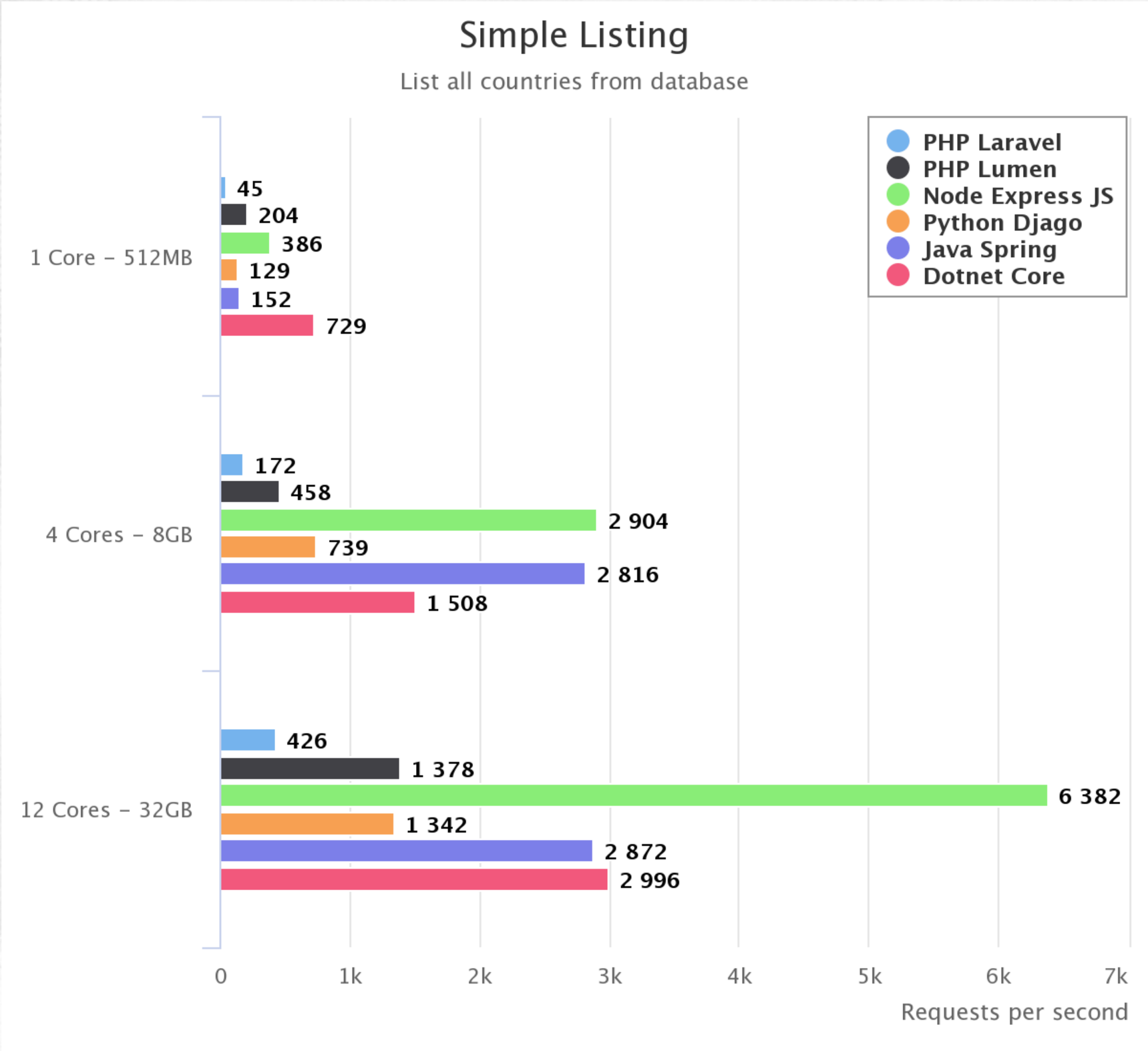
Reference: https://icing.space/2019/programming-language-%E2%9A%A1%EF%B8%8F-energy-efficiency-compared/

Consideration on performance (http capacity)



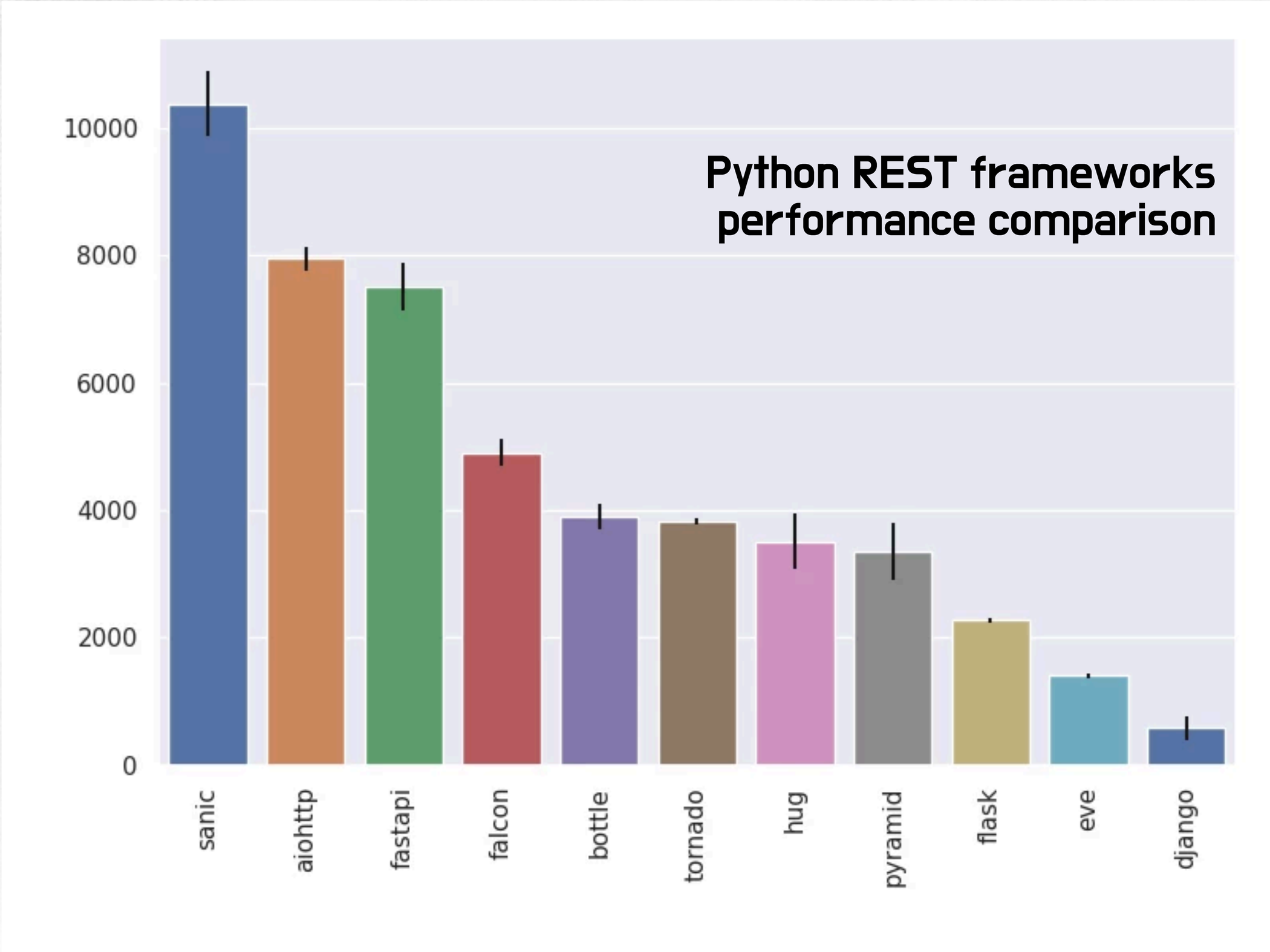
Reference: https://medium.com/@mihaigeorge.c/web-rest-api-benchmark-on-a-real-life-application-ebb743a5d7a3

Consideration on performance (http capacity)



Reference: https://medium.com/@mihaigeorge.c/web-rest-api-benchmark-on-a-real-life-application-ebb743a5d7a3

Consideration on performance (http capacity)



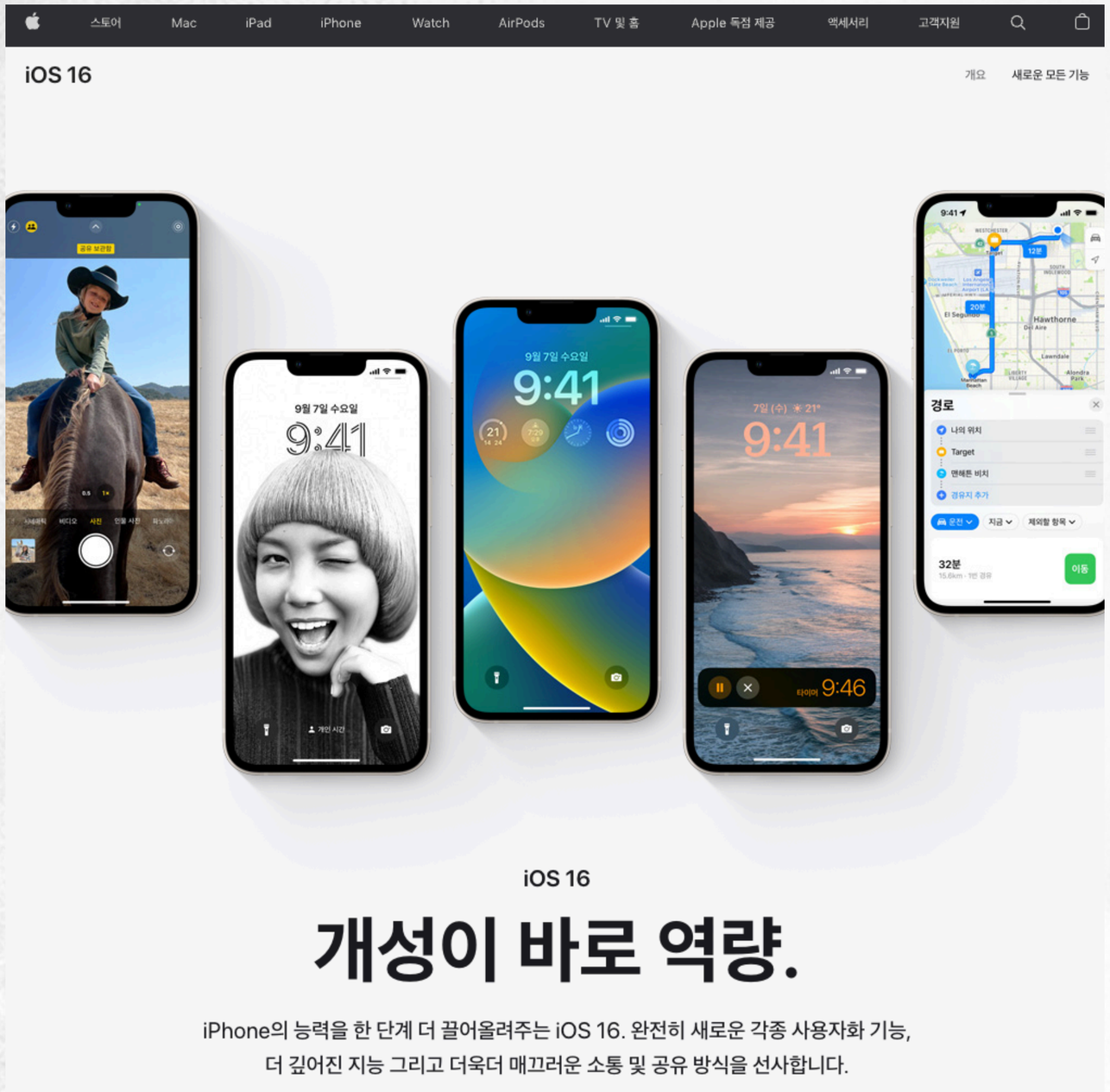
Reference: https://www.grandmetric.com/2020/08/10/python-rest-frameworks-performance-comparison/

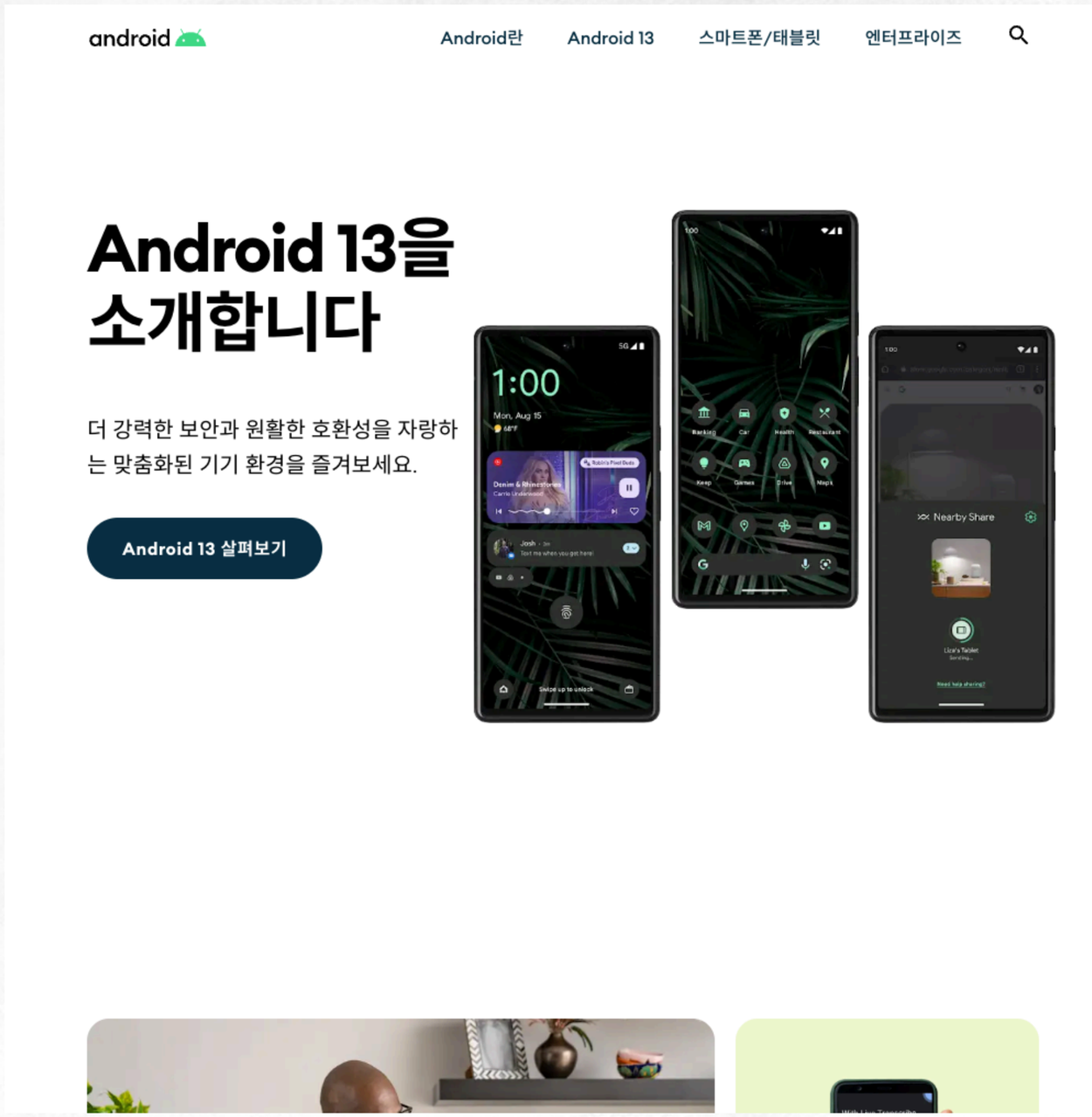
Hardware independent problem

- Case Study: Mobile application development
  - Native platform (iOS, Android)
  - Cross platform (HTML/CSS/JavaScript, Dart/Flutter)

Case Study: Mobile application development

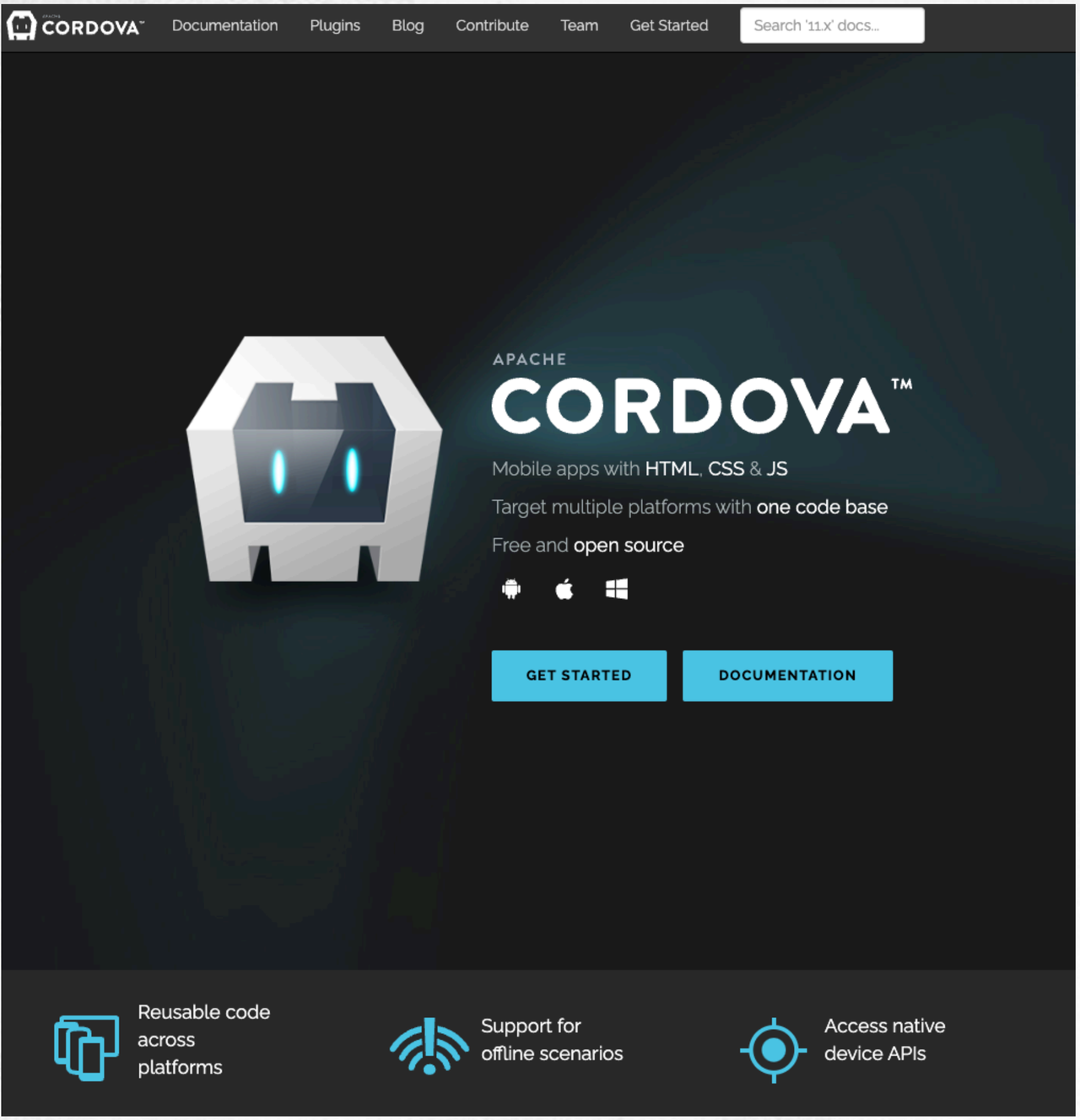
- Native platform (iOS, Android)

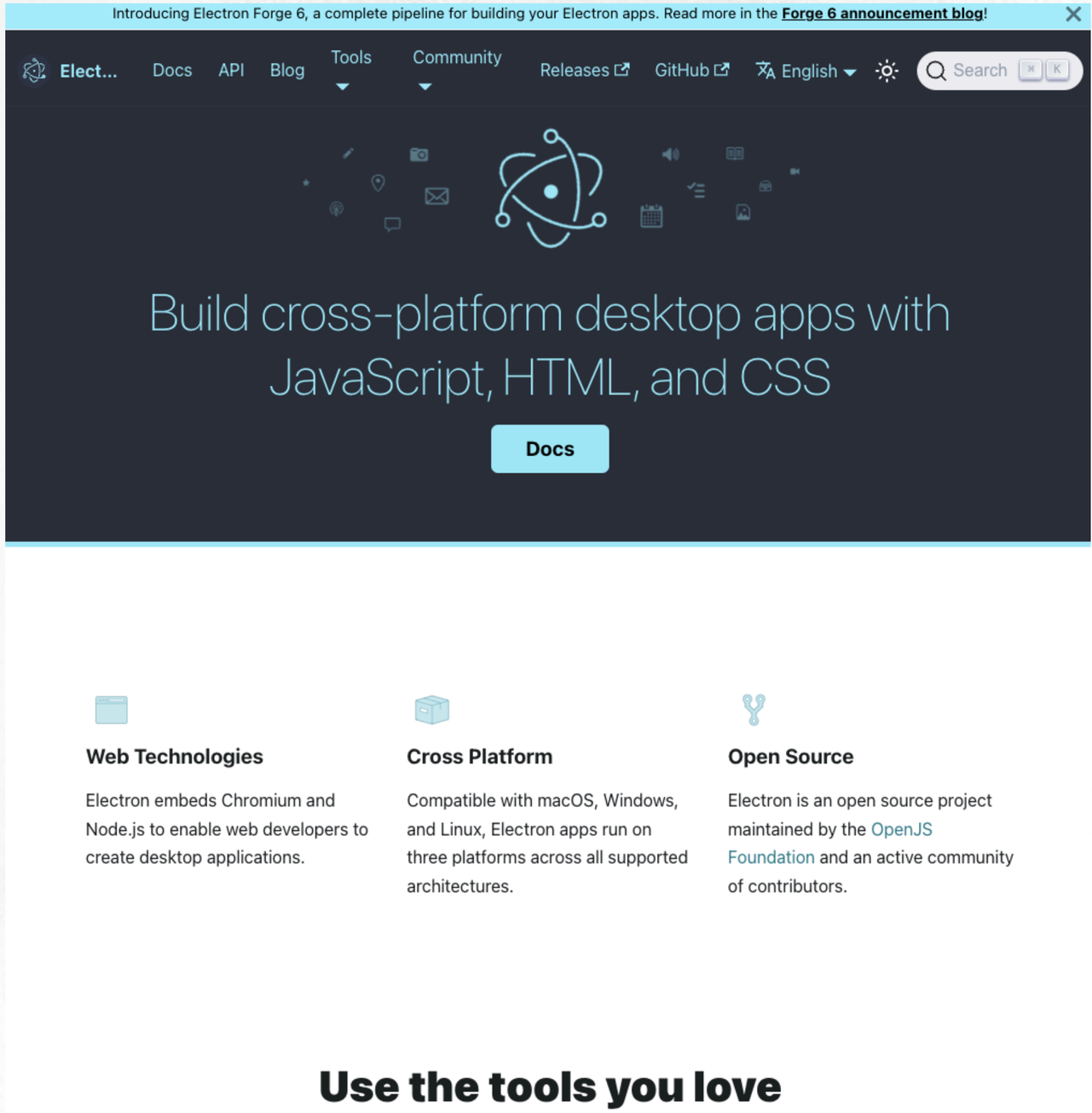




Case Study: Mobile application development

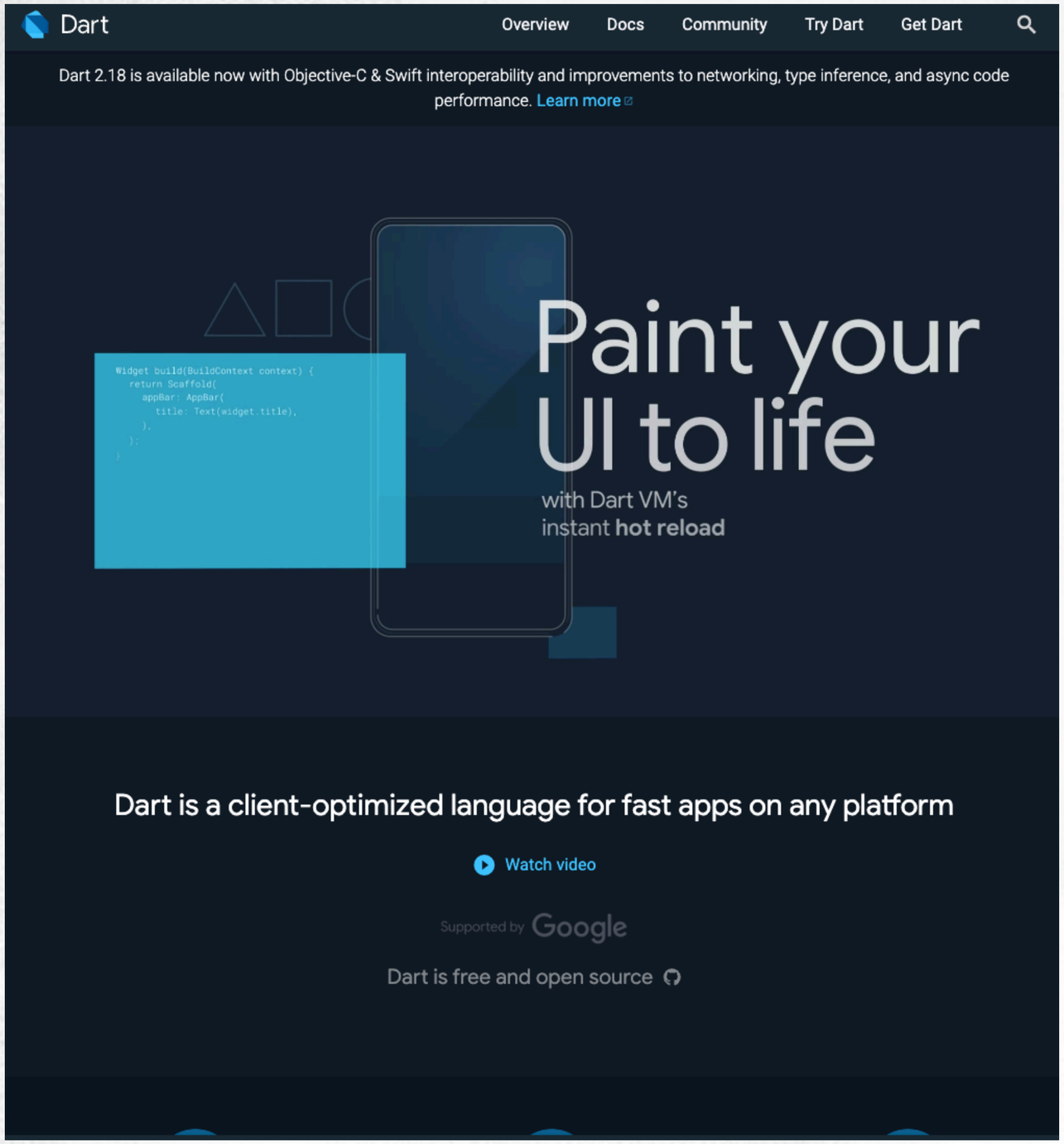
- Cross platform (HTML/CSS/JavaScript)

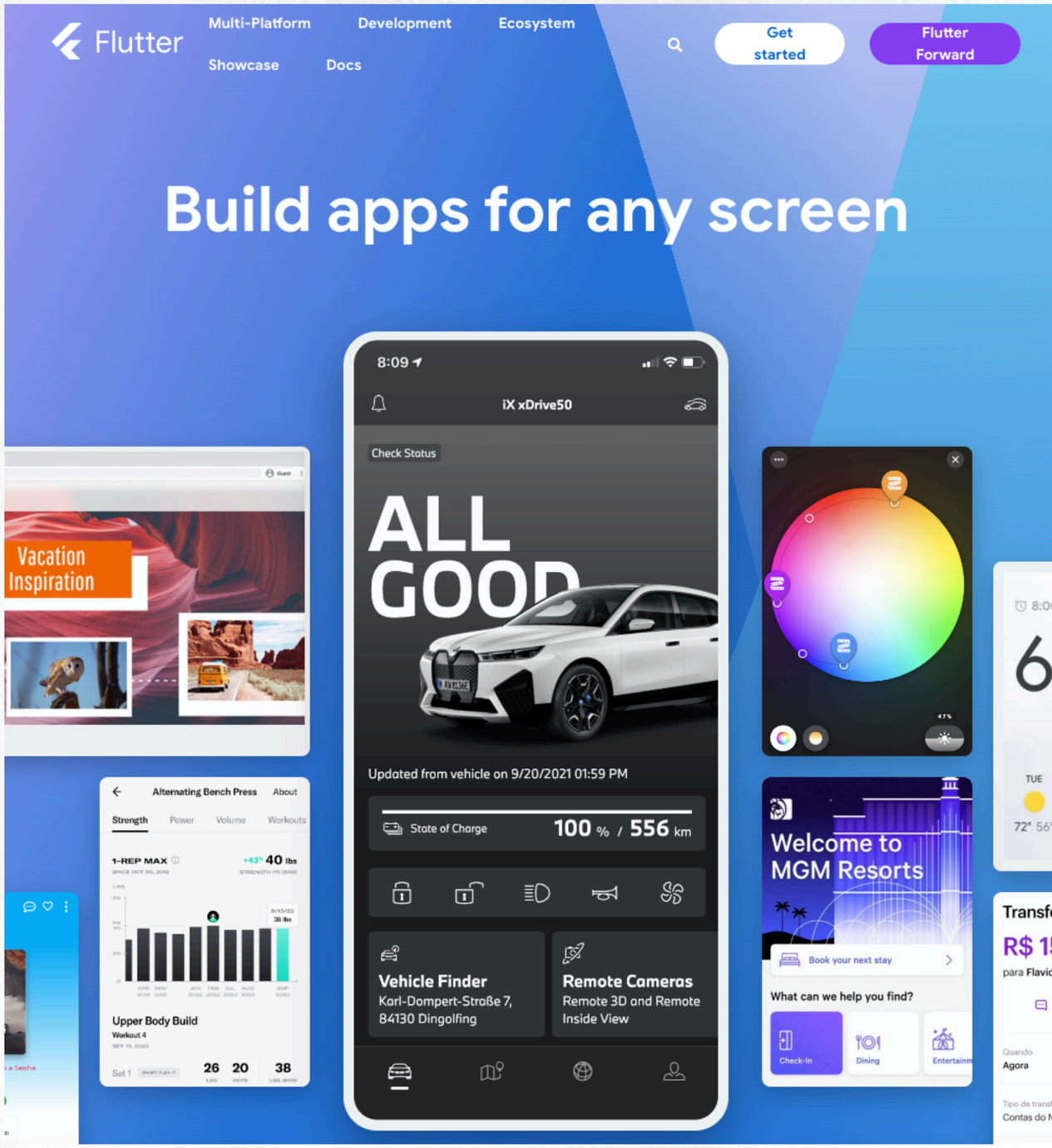




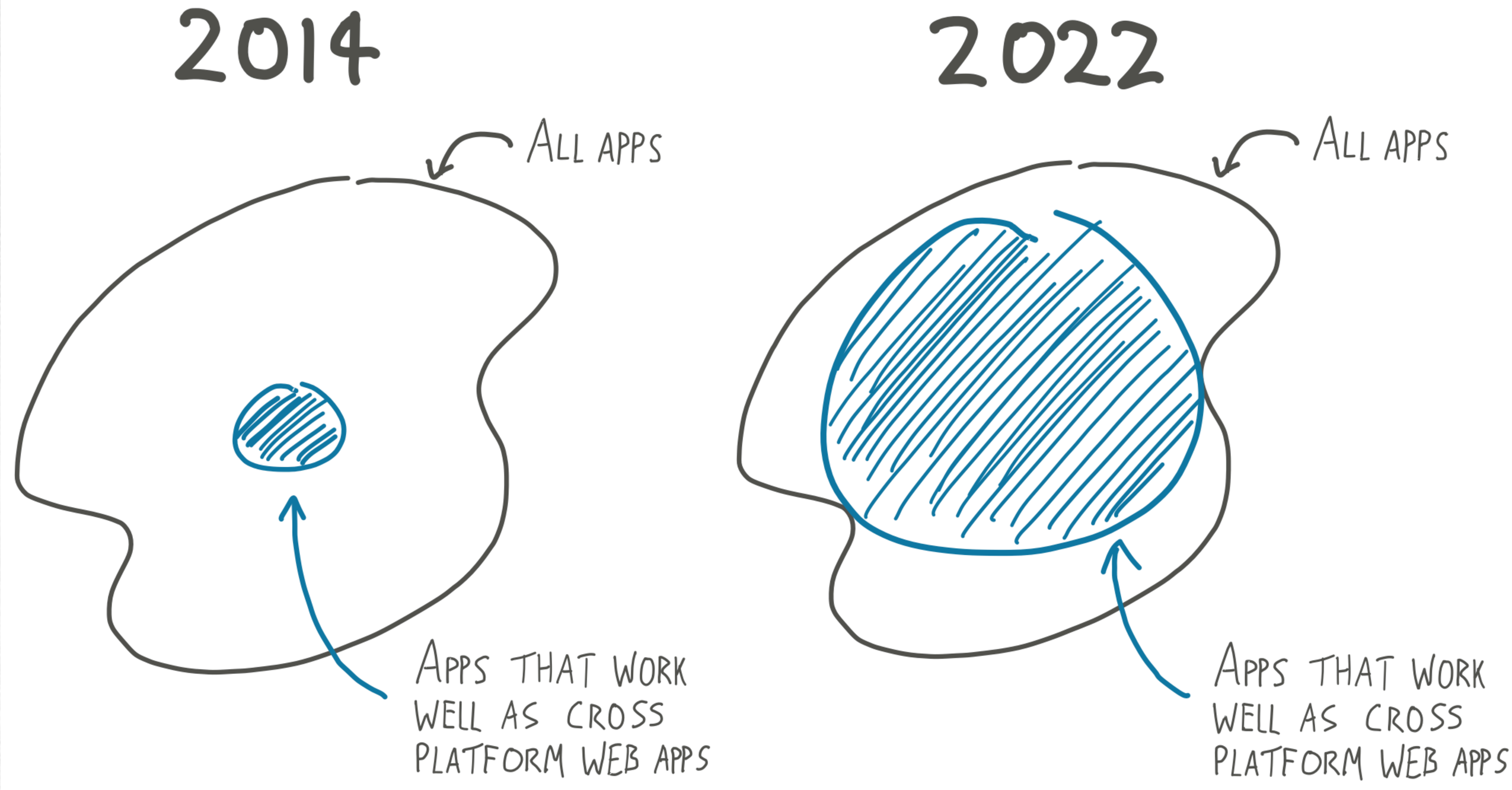
Case Study: Mobile application development

- Cross platform (Dart/Flutter)





Consideration on coverage



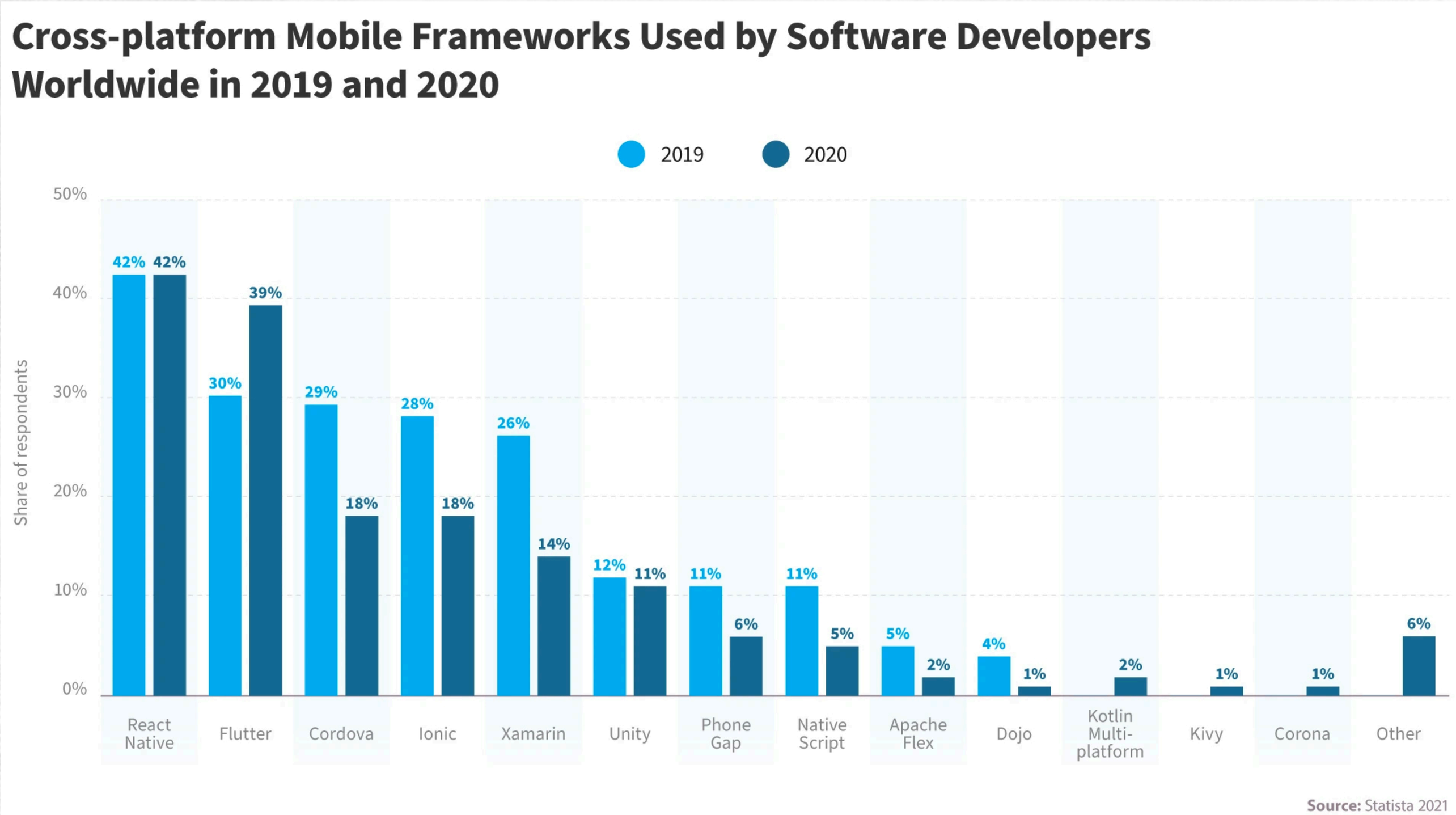
Reference: https://javascript.plainenglish.io/i-replaced-my-native-ios-app-with-a-cross-platform-web-app-and-no-one-noticed-1653901ce244

Consideration on general criteria

Criteria	Native	Cross-Platform
Architecture	Different apps for different platform	One app for multiple platforms
Performance	Fully native and higher	Native-like and performance issues
Development Time	Requires time as need to write different code	Significantly reduced development time
Cost	Not cost-effective	Lower costs
Target audience	Limited to a specific platform	Reach a large number of users
User experience	Superior	Less reliable as unified UX for all platforms
Code reusability	Almost none	60% reuse
Hardware accessibility	Complete hardware support	Limited

Reference: https://medium.com/tech-in-200-words/cross-platform-vs-native-apps-comparison-in-200-words-2498acd2480b

Consideration on platform



Reference:https://nix-united.com/blog/flutter-vs-react-native/

Consideration on platform

	Flutter	React Native
Date of release	May, 2017	March, 2015
Vendor	Google	Facebook
Programming language	Dart	JavaScript
Type	Software Development Kit	Software Framework
Positioning	"Beautiful native apps in record time"	"A framework for building native apps using React"
Licensing	Free, open-source	

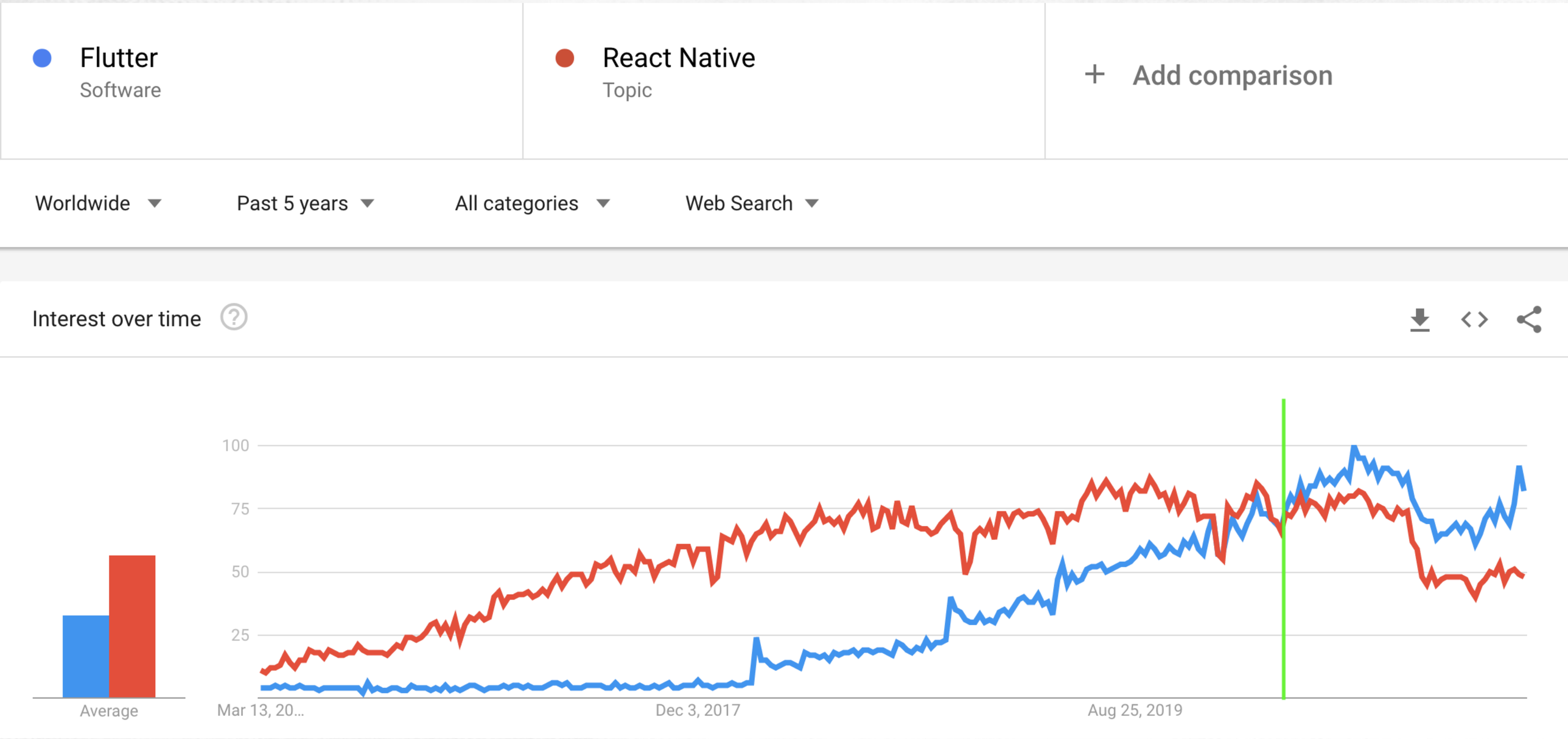
Reference: https://nix-united.com/blog/flutter-vs-react-native/

Consideration on platform

	Flutter	React Native
Github stars	116,000	93,900
Github commits	23,022	22,113
Stackoverflow watchers	44,400	63,500
Stackoverflow questions	79,604	95,662

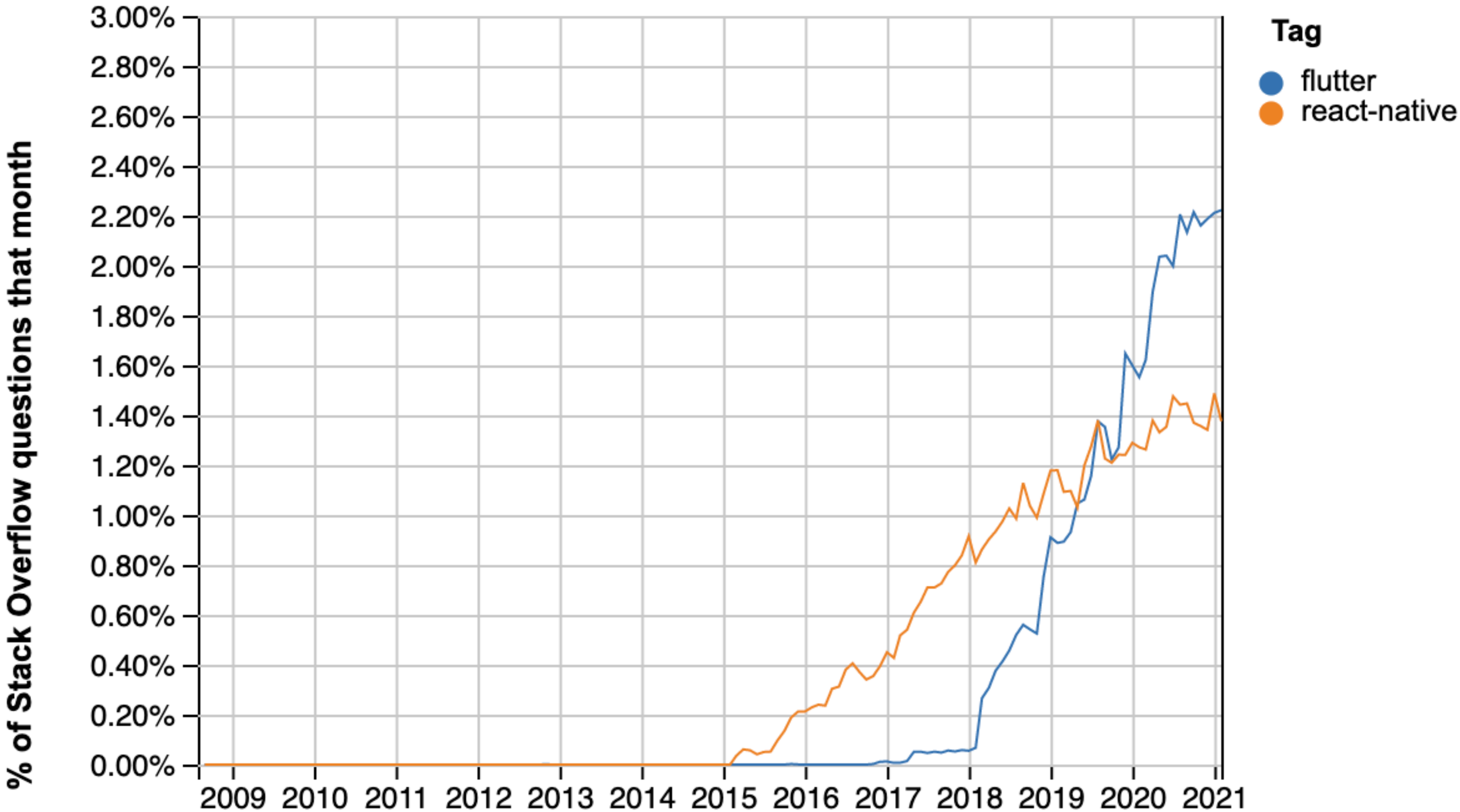
Reference: https://nix-united.com/blog/flutter-vs-react-native/

Consideration on platform



Reference: https://nix-united.com/blog/flutter-vs-react-native/

Consideration on platform



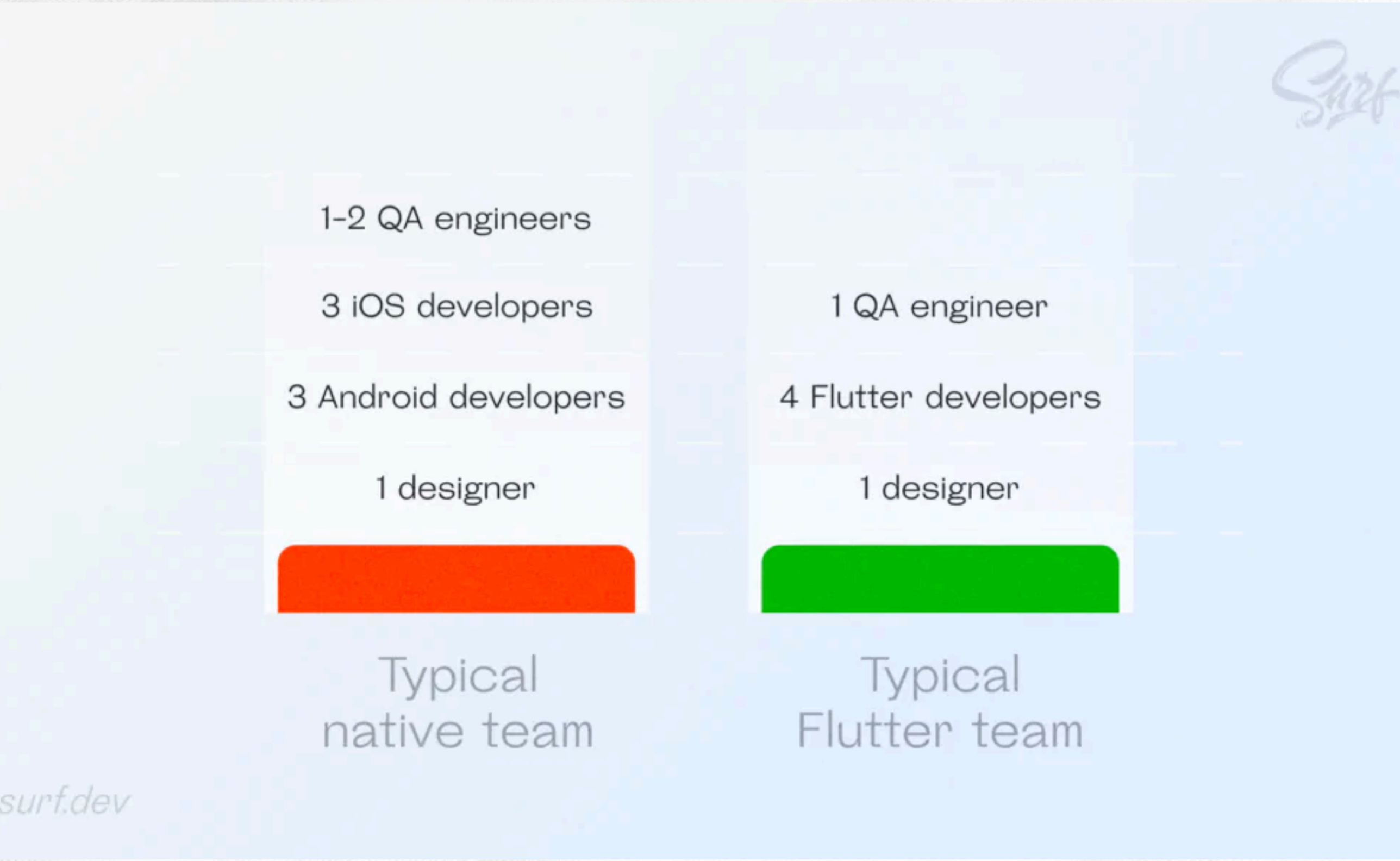
Reference: https://nix-united.com/blog/flutter-vs-react-native/

Consideration on performance (time)

Type of programming task for benchmark	iOS (ms)			Android (ms)		
	React Native	Flutter	Native	React Native	Flutter	Native
CPU intensive test with Borwein's algorithm task	582	180	26	822	285	144
Memory intensive test with Gauss–Legendre algorithm task	2'992	189	173	3'289	273	223

Reference: https://nix-united.com/blog/flutter-vs-react-native/

Consideration on resource



Consideration on coverage

Flutter	*Native Technologies*
Advanced OS plugins with simple logic	AR/VR apps
Apps with material design	Advanced UI apps
Apps with Skia rendering engine	Apps with iOS-level features
Flexible UI with high-level widgets	AI-triggered apps with big data
MVP mobile apps	Heavy computation apps
Rich-animations apps	Gaming apps
GPS-centric apps	IoT apps connecting with TV and Wallets
	GPS-centric apps
	Rich-animations apps
	Standalone native apps

surf.dev

Reference: https://surf.dev/flutter-vs-native/

Consideration on coverage

When to choose native technologies?	*When to avoid native technologies?*
Create advanced native apps with flawless performance and speed	When you have high maintenance and updates' costs
Create niche MVP apps	Global device compatibility development and related issues
Develop top-performance apps with a strong OS embedded connection between the app and the device	Need to deploy the app into the market quickly
Regular app updates based on the release of every new OS version	App development without a high investment plan, with a less skilled team
Need for sophisticated features with top reliability and security	Development of a native app with fewer in-app payment options
MVP mobile apps	
Reactive apps with big data integration	

surf.dev

Reference: https://surf.dev/flutter-vs-native/

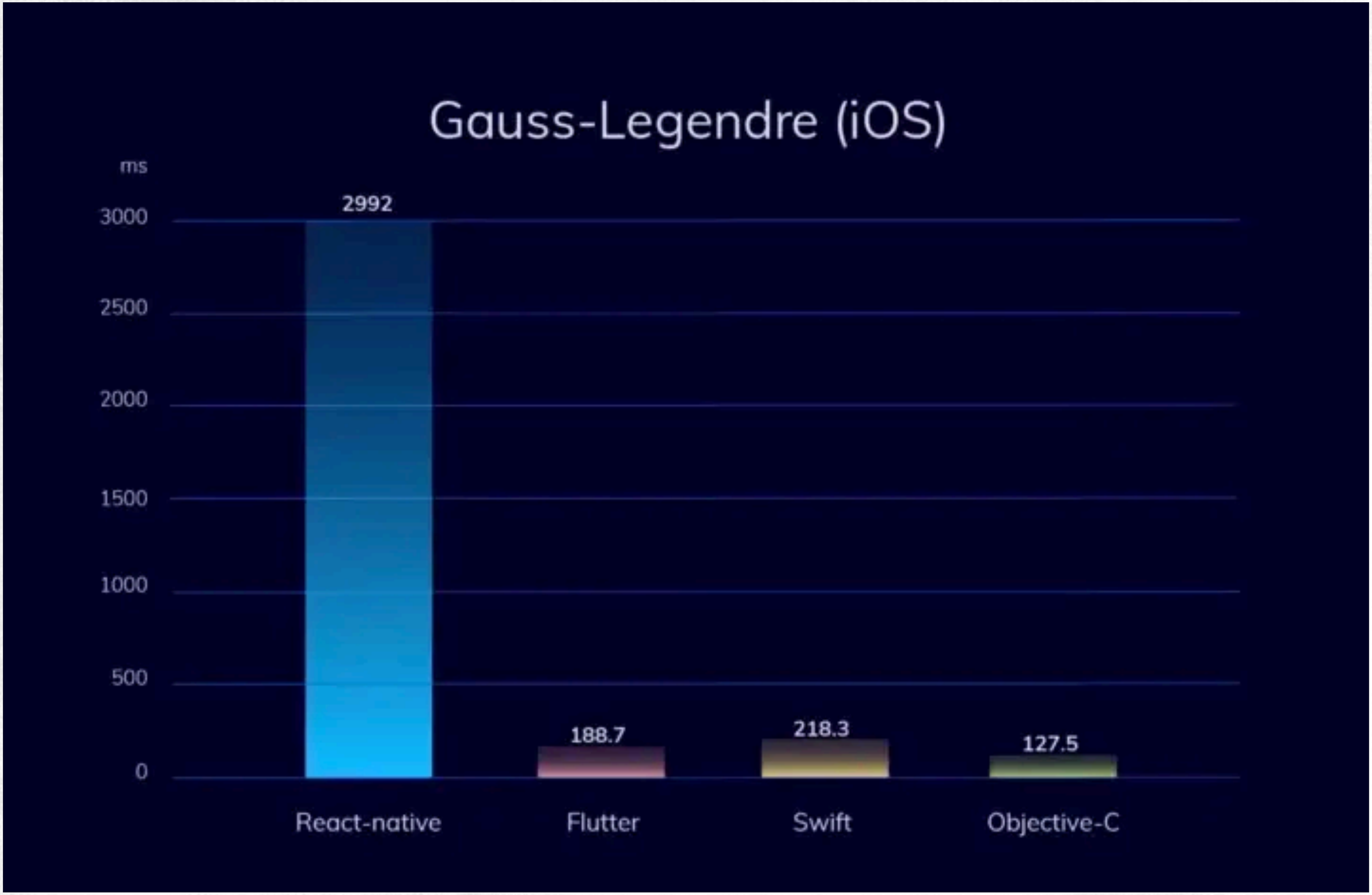
Consideration on coverage

When to choose Flutter?	*When to avoid Flutter?*
Build MVP with on-demand mobile features and reactive widgets	For brand-specific and feature-specific apps for a standalone business
Create native apps with modern cloud services	For mobile apps with universally accepted MVP, MVVM, and MVC architectures
Create a smooth design experience	For mobile apps with a particular focus on native functionality and performance
Develop a single code with front-end and back-end functionalities	For native apps without functional limitations or UI
Quickly reach wider target audience and reduce time-to-market	

surf.dev

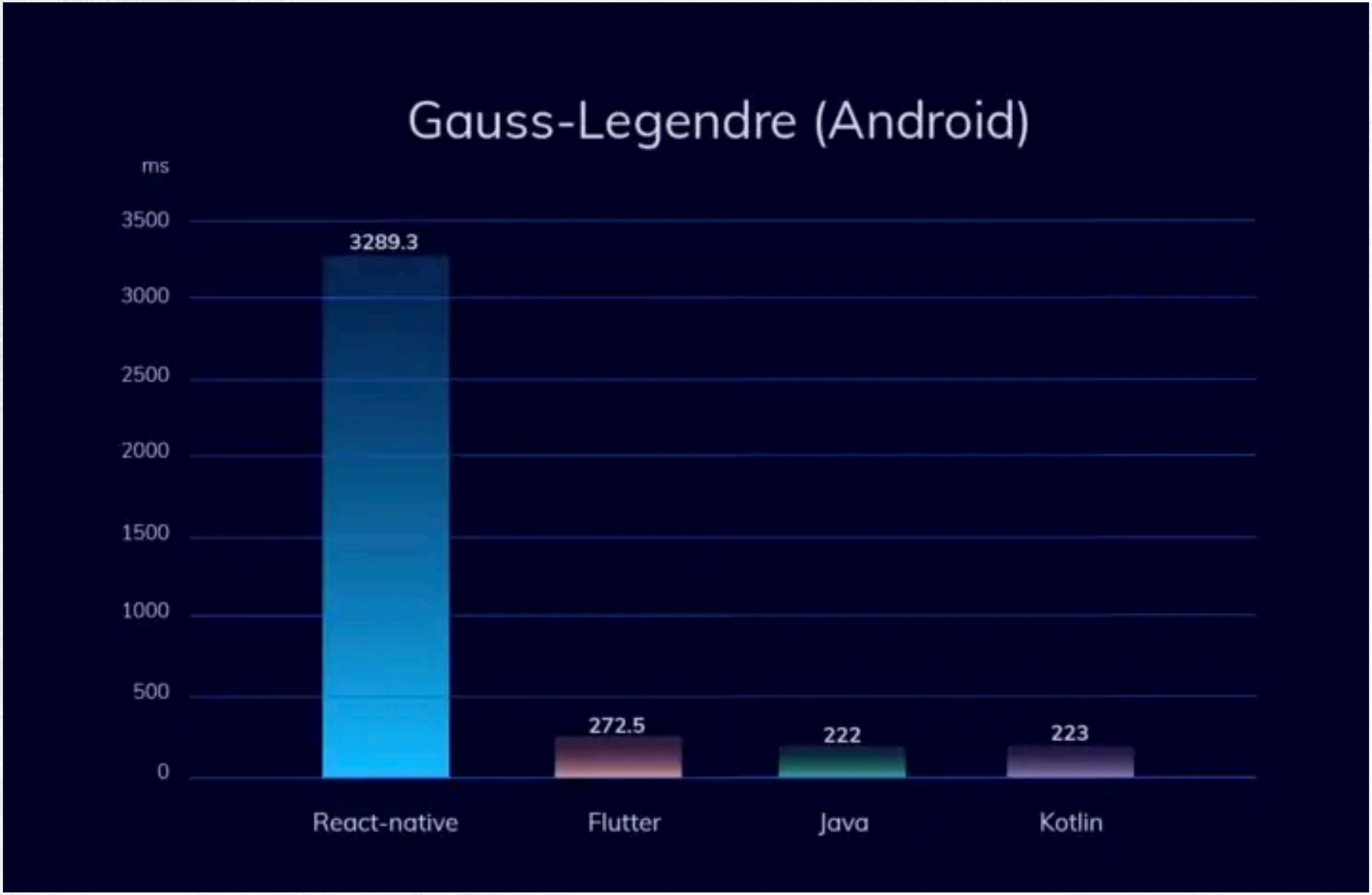
Reference: https://surf.dev/flutter-vs-native/

Consideration on performance (time)



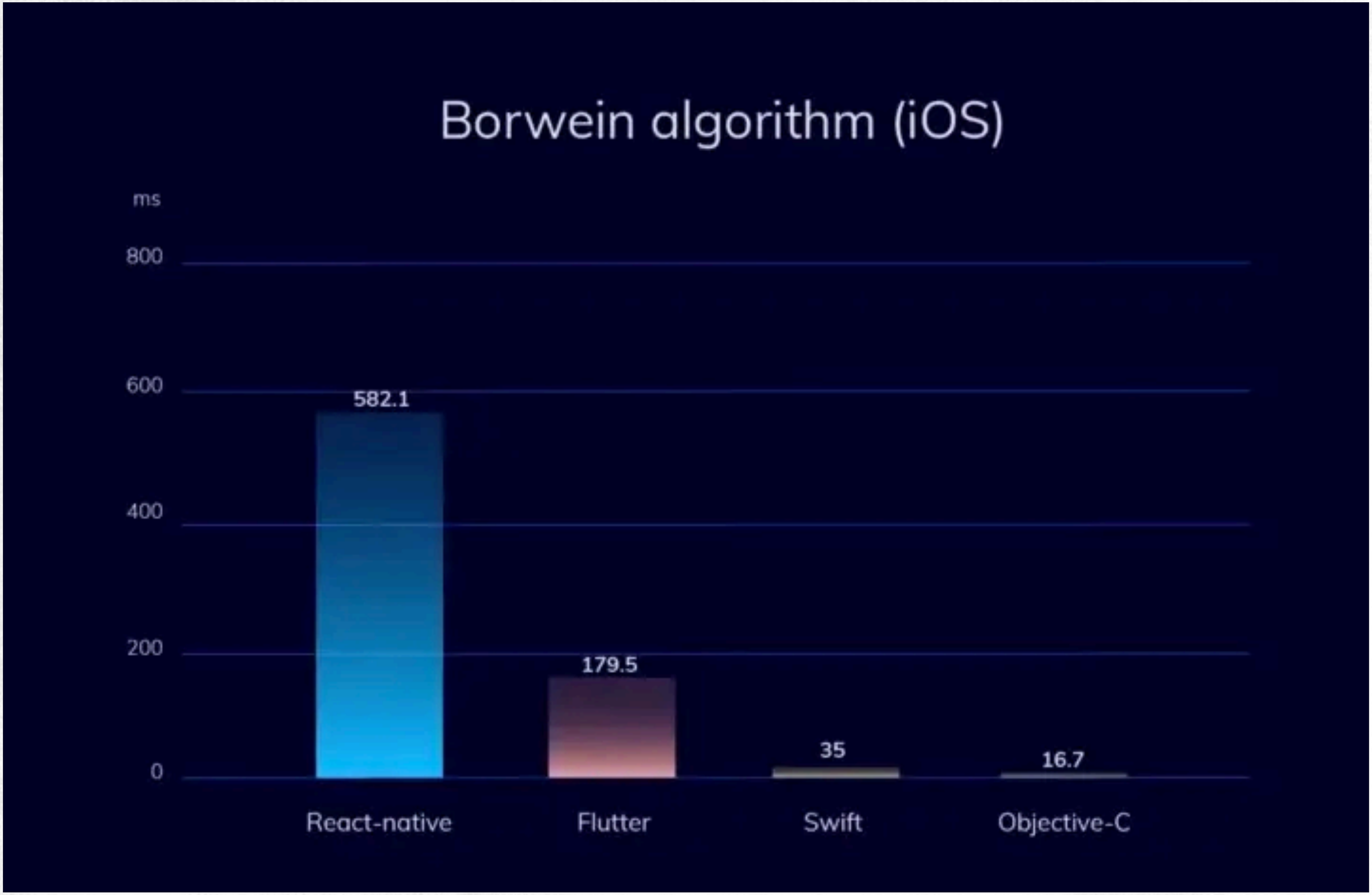
Reference: https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980

Consideration on performance (time)



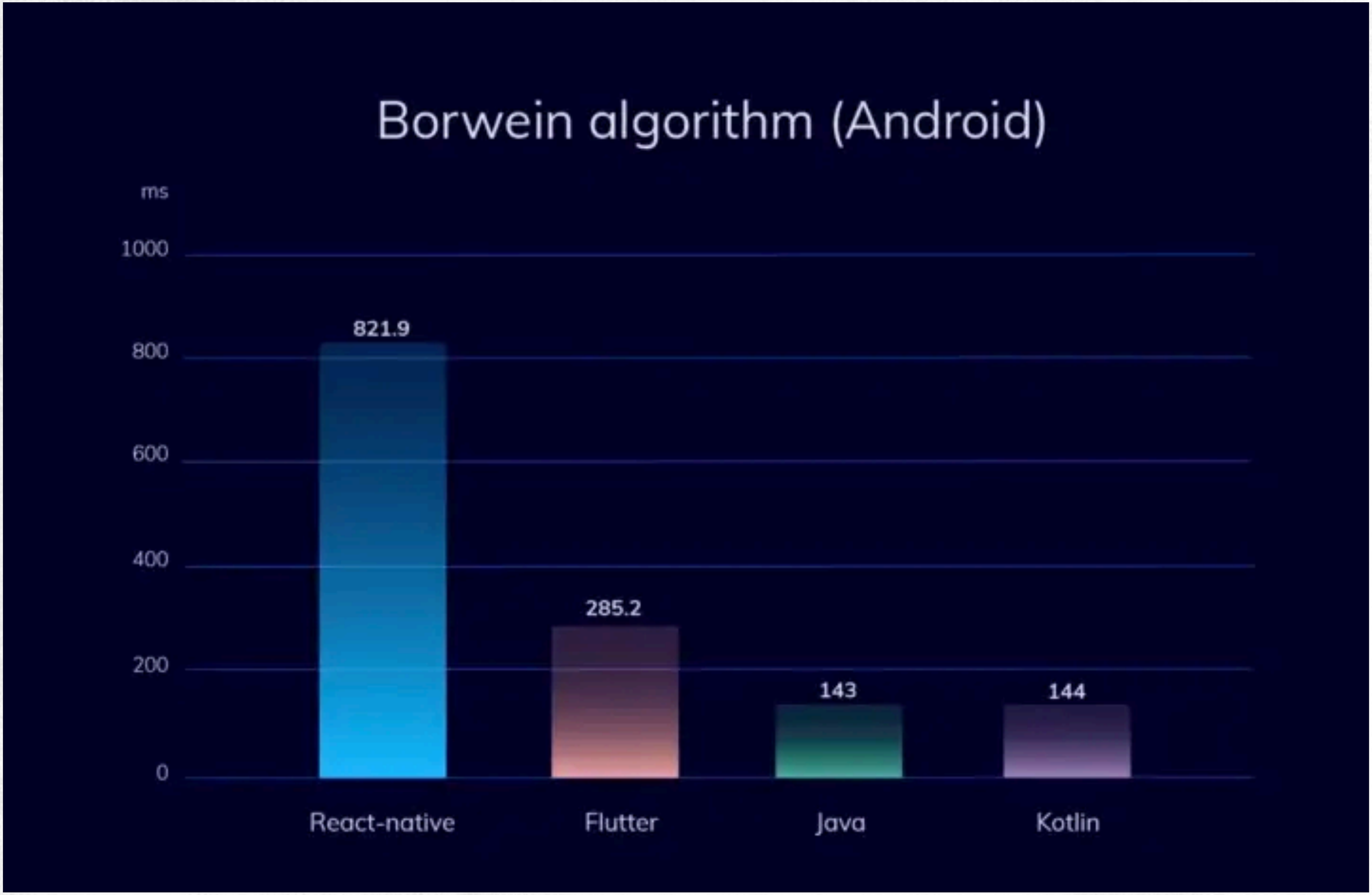
Reference: https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980

Consideration on performance (time)



Reference: https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980

Consideration on performance (time)



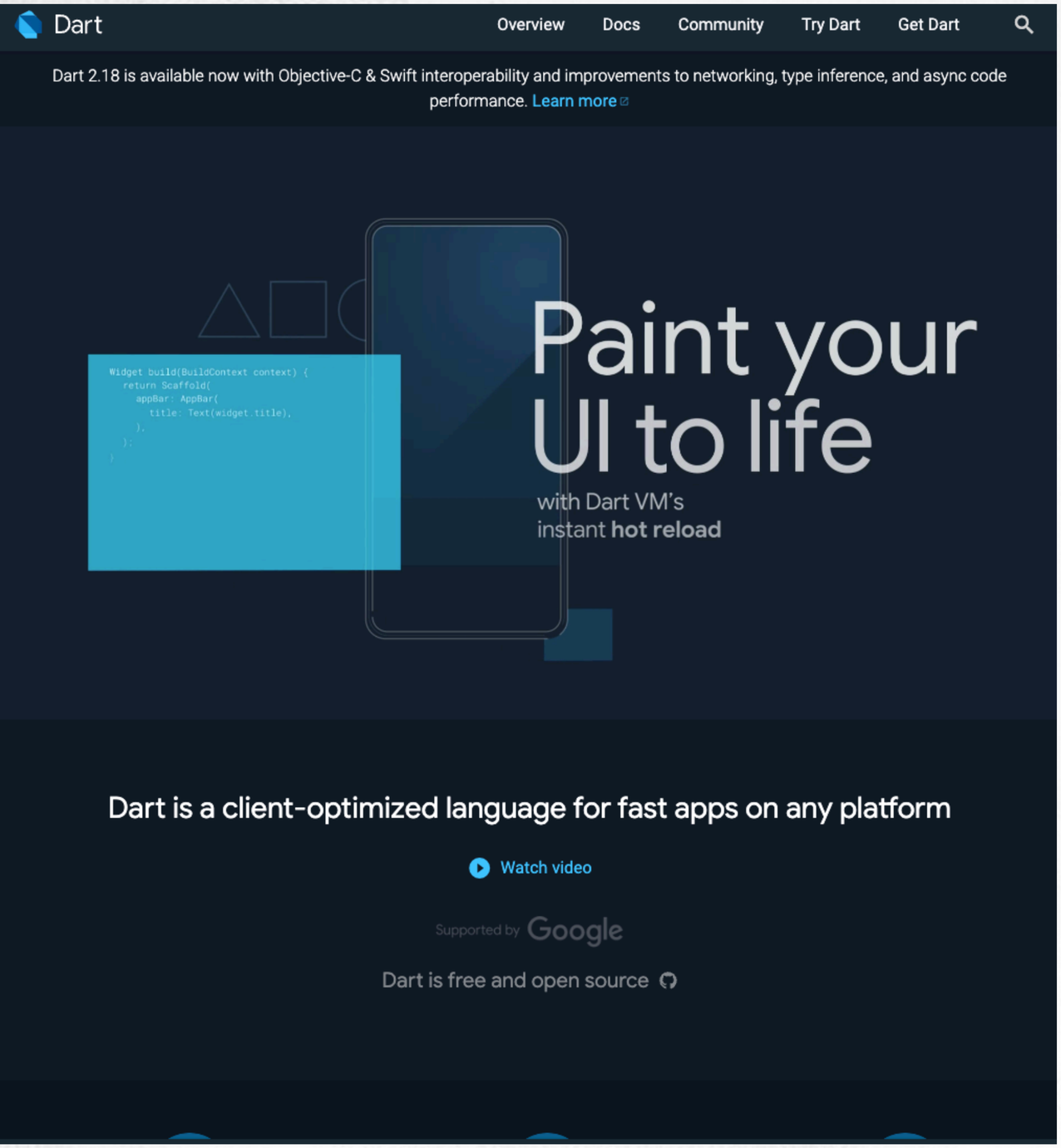
Reference: https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980

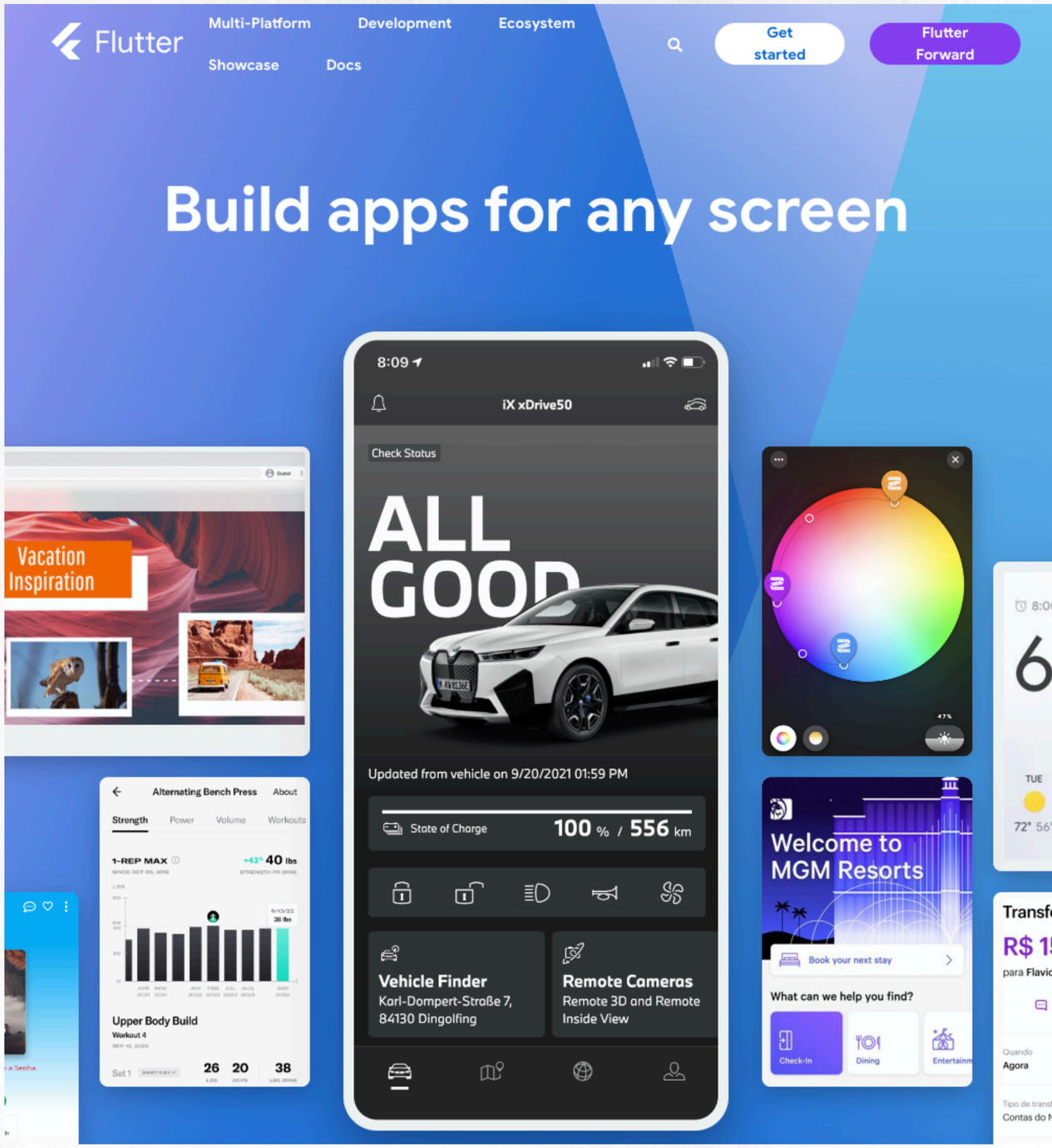
Summary

- Define your problem.
  - Hardware dependent
  - Hardware in-dependent
- Define your approach:
  - From the scratch [making new software]
  - Evolving conventional software
- Define your KPIs (Key Performance Indexes):
  - Due
  - Performance
  - Cost
  - Man-power

Reference

- Dart & Flutter official sites





Thank you

경희대학교
KYUNG HEE UNIVERSITY