IF3210 Pengembangan Aplikasi Peranti Bergerak

(IF3210 Mobile Application Development)

Mobile Development Cross Platform Technology Approaches

Tim Dosen IF3210

STEI ITB

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Outline

1. Definition of Mobile App Development
2. The Key Challenge of Mobile App Dev
3. Mobile App Development Approaches
4. Cross-Platform Architecture (RN & Flutter)
5. Framework Choice Guidelines

Mobile App Development (IBM)

Mobile application development is the process of making software for smartphones, tablets and digital assistants, most commonly for the Android and iOS operating systems.

Mobile App Development (AWS)

Mobile application development is the process of creating software applications that run on a mobile device, and a typical mobile application utilizes a network connection to work with remote computing resources. Hence, the mobile development process involves creating installable software bundles (code, binaries, assets, etc.) , implementing backend services such as data access with an API, and testing the application on target devices.

Mobile App Development (Microsoft)

The definition of mobile application development can be broad—encompassing all the processes involved in the creation and release of a mobile app—or specific, referring to the coding and deployment of the app.

The Key Challenge

In Mobile Apps Development

Fragmentation

Each mobile platform has different interfaces, standards, programming languages, APIs, capabilities, and SDKs

Within the same platform, each device has differences, for example: memory, screen resolution, etc.

Ahmad, A., Li, K., Feng, C., Asim, S. M., Yousif, A., dan Ge, S. (2018): An Empirical Study of Investigating Mobile Applications Development Challenges, *IEEE Access*, **6**, 17711–17728. https://doi.org/10.1109/ACCESS.2018.2818724

3 Approaches

In Mobile Apps Development

Kotlin

Native

Use platform-specific programming languages and tools, such as Kotlin for Android, Swift for iOS, etc.

JS

Mobile Web

Use web technologies to create apps that run on mobile browsers.

Cross-platform

Write in a single code base and target multiple platforms.

Native Tools

In Mobile Apps Development

Programming Languages	Java, **Kotlin**, C/C++	Objective-C, **Swift**
IDE	Android Studio	Xcode
Development Operating System	Windows, macOS, Linux	macOS
Dependency Manager	Gradle	CocoaPods
Note: green indicates a recommendation	https://roadmap.sh/android	https://roadmap.sh/ios

Mobile Web Tools

In Mobile Apps Development

Browser Engine	Chromium	WebKit
Default Browser	Google Chrome	Safari
Languages (Programming, Styling, and Markup)	JavaScript, HTML, CSS	
Popular Tools	**React**, Vue, Svelte, Angular, jQuery, etc.	
Popular IDE	**VSCode**, Notepad++, Sublime Text, etc.	
Dependency Manager	**npm**, yarn, bower, etc.	

Note: green indicates a recommendation

Cross-Platform Tools

Top 2 cross-platform tools in mobile app development (Stackoverflow Developer Survey 2023)

React Native

An open-source framework by Facebook for creating native apps for Android and iOS using React (A best-in-class JavaScript library for building UIs)

Flutter

An open-source framework by Google for building beautiful, natively compiled, multi-platform applications from a single codebase.

Top 2 Cross-Platform Tools

In Mobile Apps Development

	React Native	Flutter
Programming Languages	JavaScript	Dart
Installation	npm	binary
Community Support	Very high	Rapidly growing
Build & Release	Manual or EAS (Expo)	CLI
DevOps (CI/CD)	EAS (Expo) or third parties	Rich Default CLI for easy CI/CD

Why Do We Need Cross-Platform Tools?

In Mobile Apps Development

- Reusable code
- Rapid time to market
- Robust performance
- Close-to-native UX
- Broad audience reach
- Greater cost efficiencies

The decision whether to build a cross-platform, native, or web app depends on its audience, complexity, and functionality. Budget limitations, time to market, developer expertise, and other business priorities also play a role in decision-making.

https://www.microsoft.com/en-gb/power-platform/products/power-apps/topics/app-development/native-vs-cross-platform-apps

Case Study: Starlink App

https://www.youtube.com/watch?v=tsR9eqmlrjs

Show Case: Xiaomi

https://www.youtube.com/watch?v=wfD7ZQhwACU

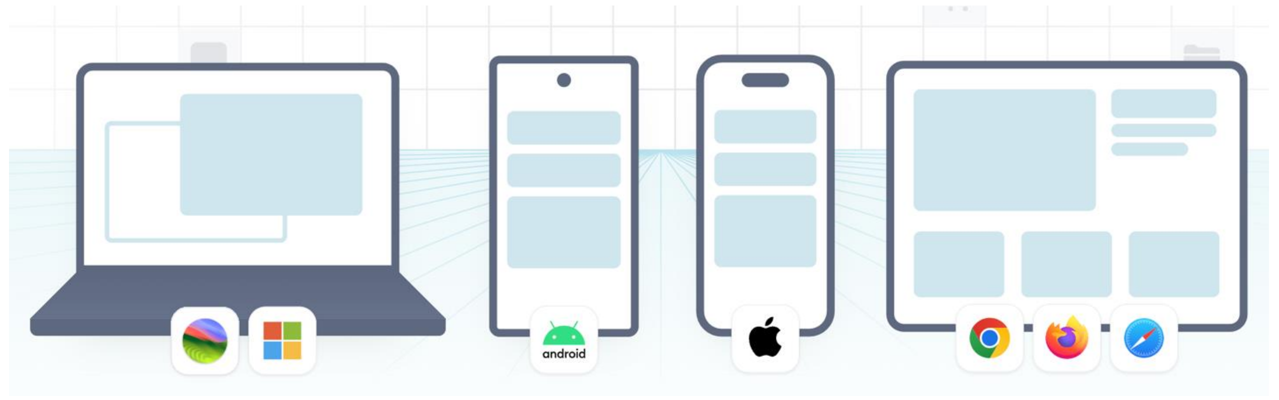
React Native (react + react-native)

- React Native is an open-source framework for building native apps for Android, iOS, and more using React.

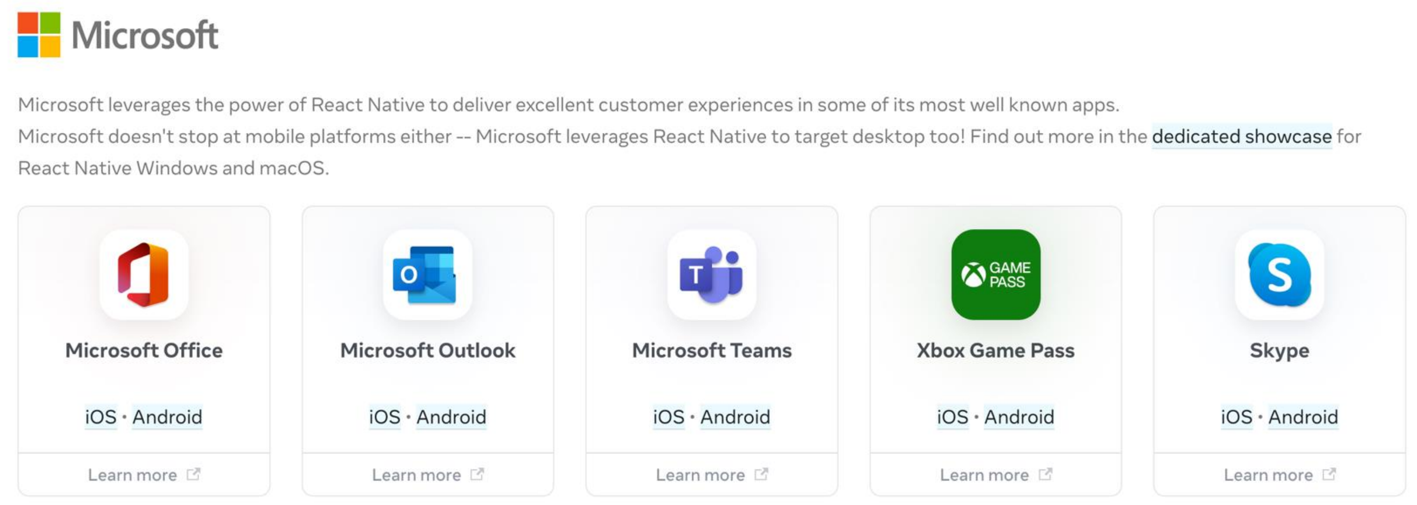
- React Native was first developed in 2013 as an internal Facebook project, and then released to the public in 2015.

- **We use JavaScript** to access **native platform's APIs** as well as to describe the appearance and behavior of your UI using React components.

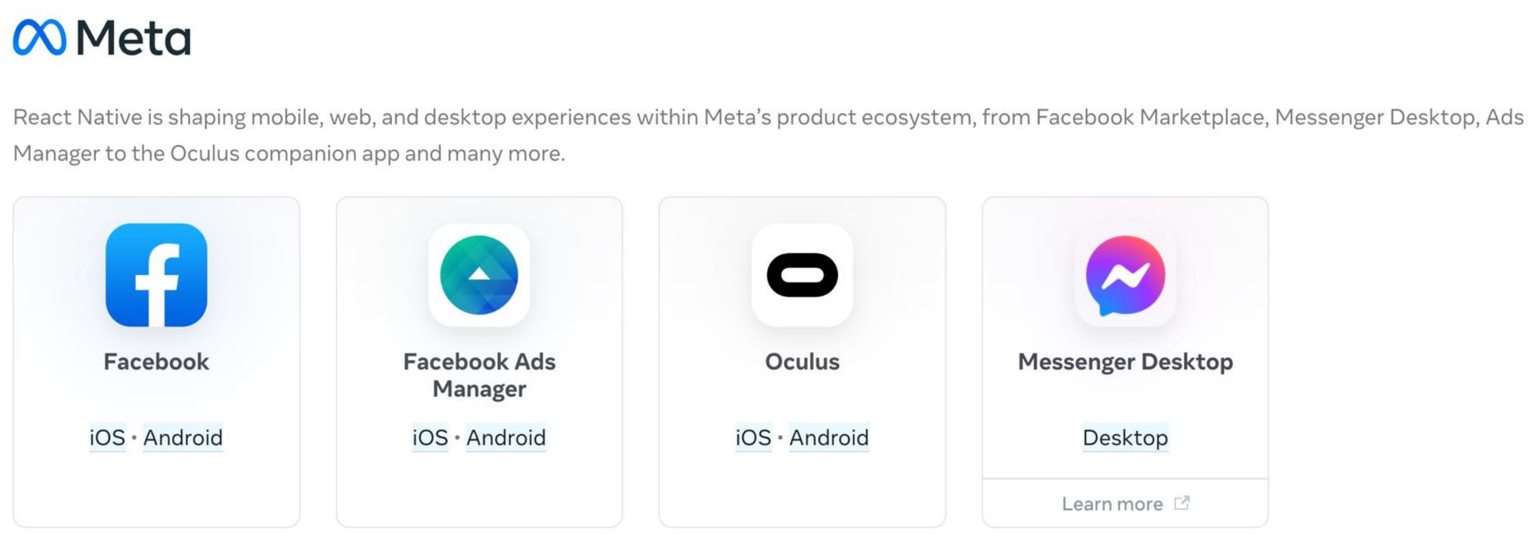
- **Fast JavaScript/Native Interfacing** using JSI (JavaScript Interface): A lightweight API to embed a JavaScript engine in a C++ application. Fabric uses it to communicate between Fabric's C++ core and React.



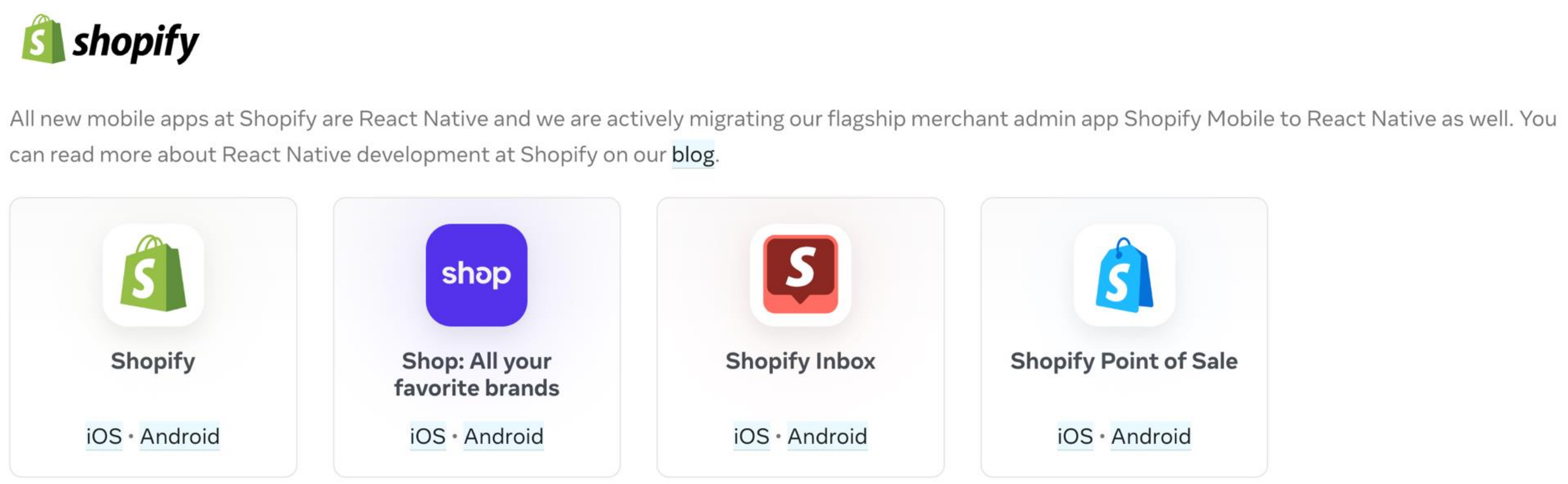
• Who is Using React Native ?


Microsoft

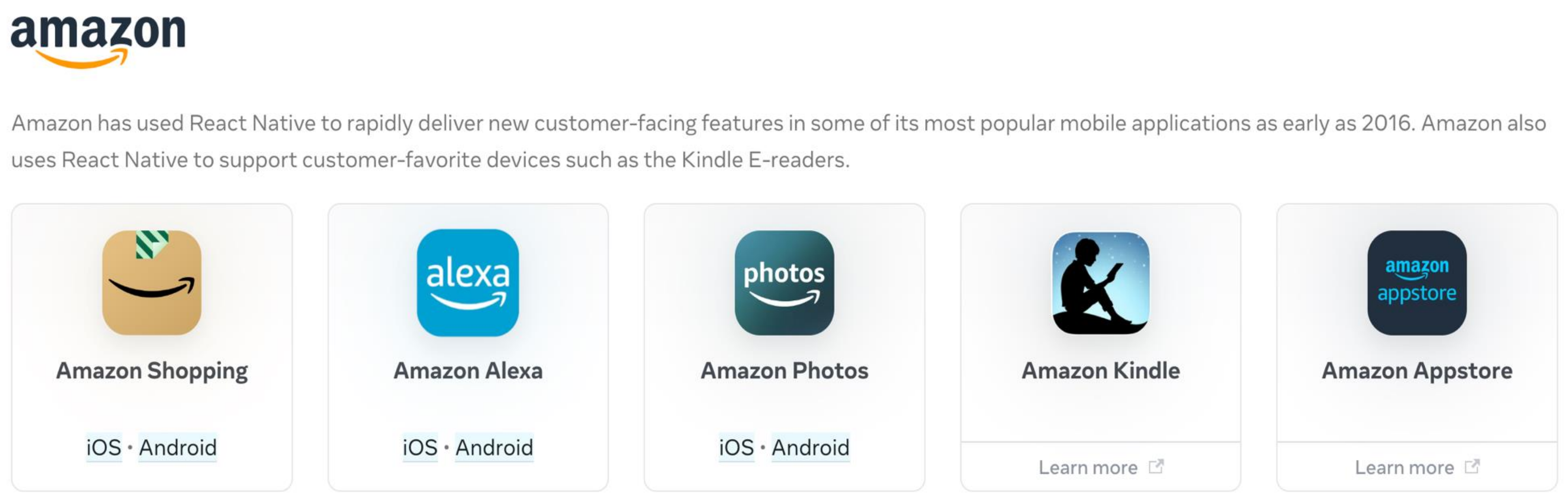
Microsoft leverages the power of React Native to deliver excellent customer experiences in some of its most well known apps.
Microsoft doesn't stop at mobile platforms either -- Microsoft leverages React Native to target desktop too! Find out more in the dedicated showcase for React Native Windows and macOS.


Meta

React Native is shaping mobile, web, and desktop experiences within Meta's product ecosystem, from Facebook Marketplace, Messenger Desktop, Ads Manager to the Oculus companion app and many more.


shopify

All new mobile apps at Shopify are React Native and we are actively migrating our flagship merchant admin app Shopify Mobile to React Native as well. You can read more about React Native development at Shopify on our blog.

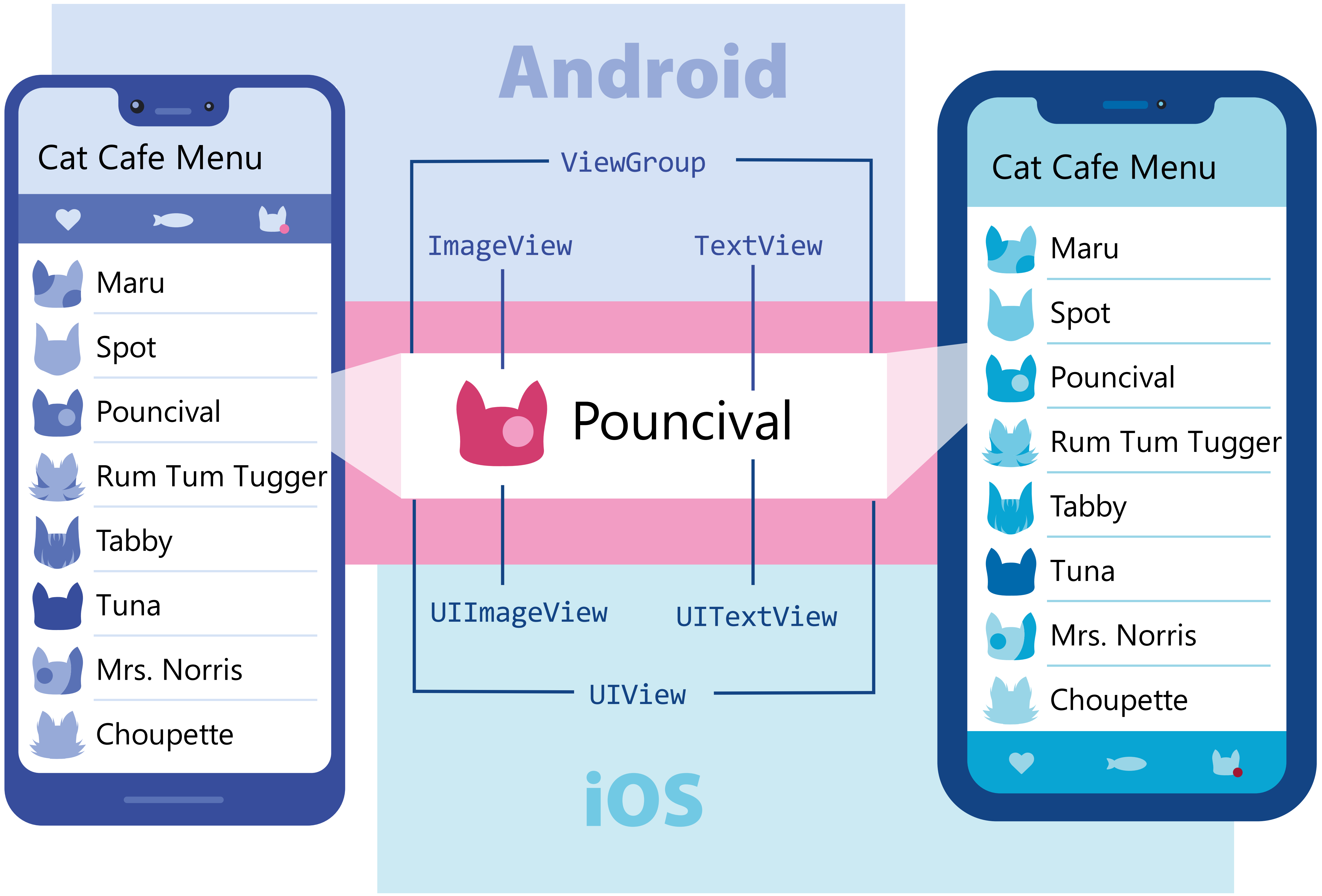

amazon

Amazon has used React Native to rapidly deliver new customer-facing features in some of its most popular mobile applications as early as 2016. Amazon also uses React Native to support customer-favorite devices such as the Kindle E-readers.

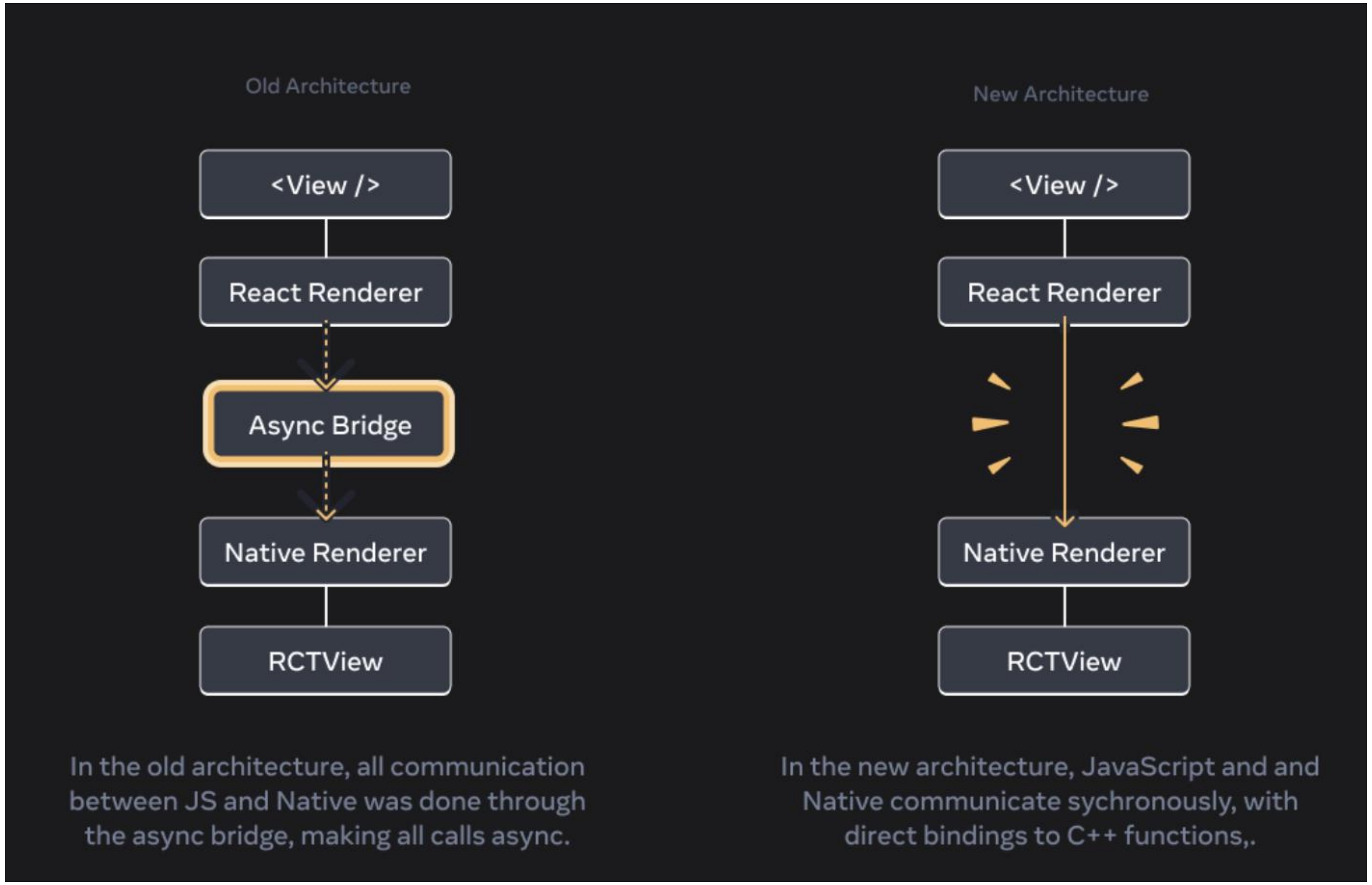
View in Native Android and iOS Development

A view is the basic building block of UI: A rectangular element on the screen which can be used to display:

- Text
- Image
- Text Input
- Other Views
- Etc..



React Native Architecture (Late 2024-onwards)



In the old architecture, all communication between JS and Native was done through the async bridge, making all calls async.

In the new architecture, JavaScript and and Native communicate sychronously, with direct bindings to C++ functions,.

Delivering the New Architecture to the OSS community has been a huge effort that took several years of research and development.

https://reactnative.dev/blog/2024/10/23/the-new-architecture-is-here

JSI, Fabric, TurboModules & Yoga

JSI is an interface that allows JavaScript to hold a reference to a C++ object and vice-versa. With a memory reference, you can directly invoke methods without serialization costs.

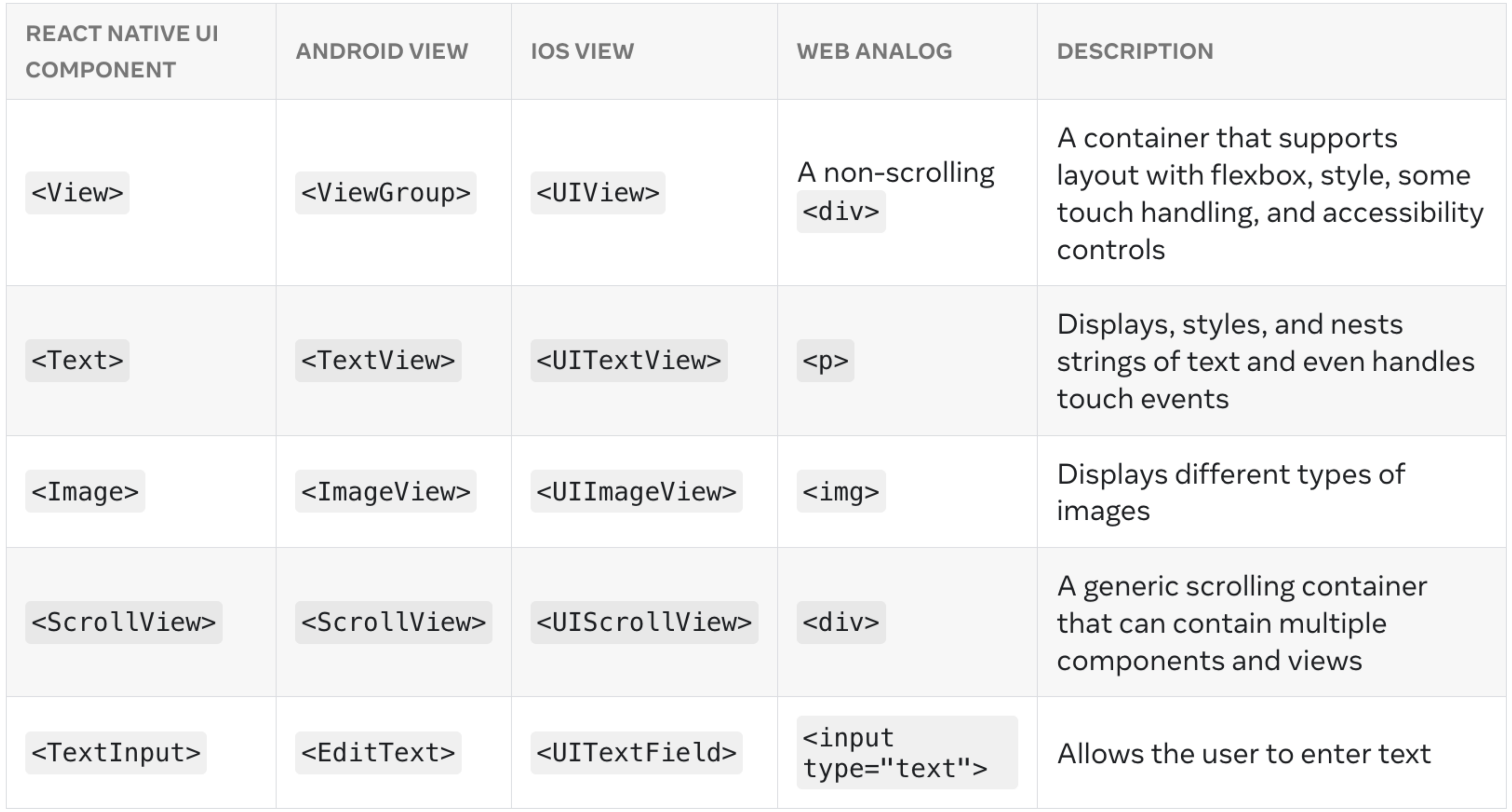
Fabric is React Native's new rendering system, a conceptual evolution of the legacy render system. The core principles are to unify more render logic in C++, improve interoperability with host platforms, and to unlock new capabilities for React Native. Using **JSI** to control the UI directly on the native side.

Your React Native application code may need to interact with native platform APIs that aren't provided by React Native or an existing library. You can write the integration code yourself using a **Turbo Native Module**. Using **JSI** to access native modules.

Yoga is an embeddable layout system used in popular UI frameworks like React Native. Yoga supports a familiar subset of CSS, mostly focused on Flexbox. This gives users a familiar model and enables sharing code between native platforms and the browser. Used by **Fabric** to position UI elements.

Native Components

In Android development, we write views in Kotlin or Java; in iOS development, you use Swift or Objective-C. With React Native, we can invoke these views with JavaScript using React components.

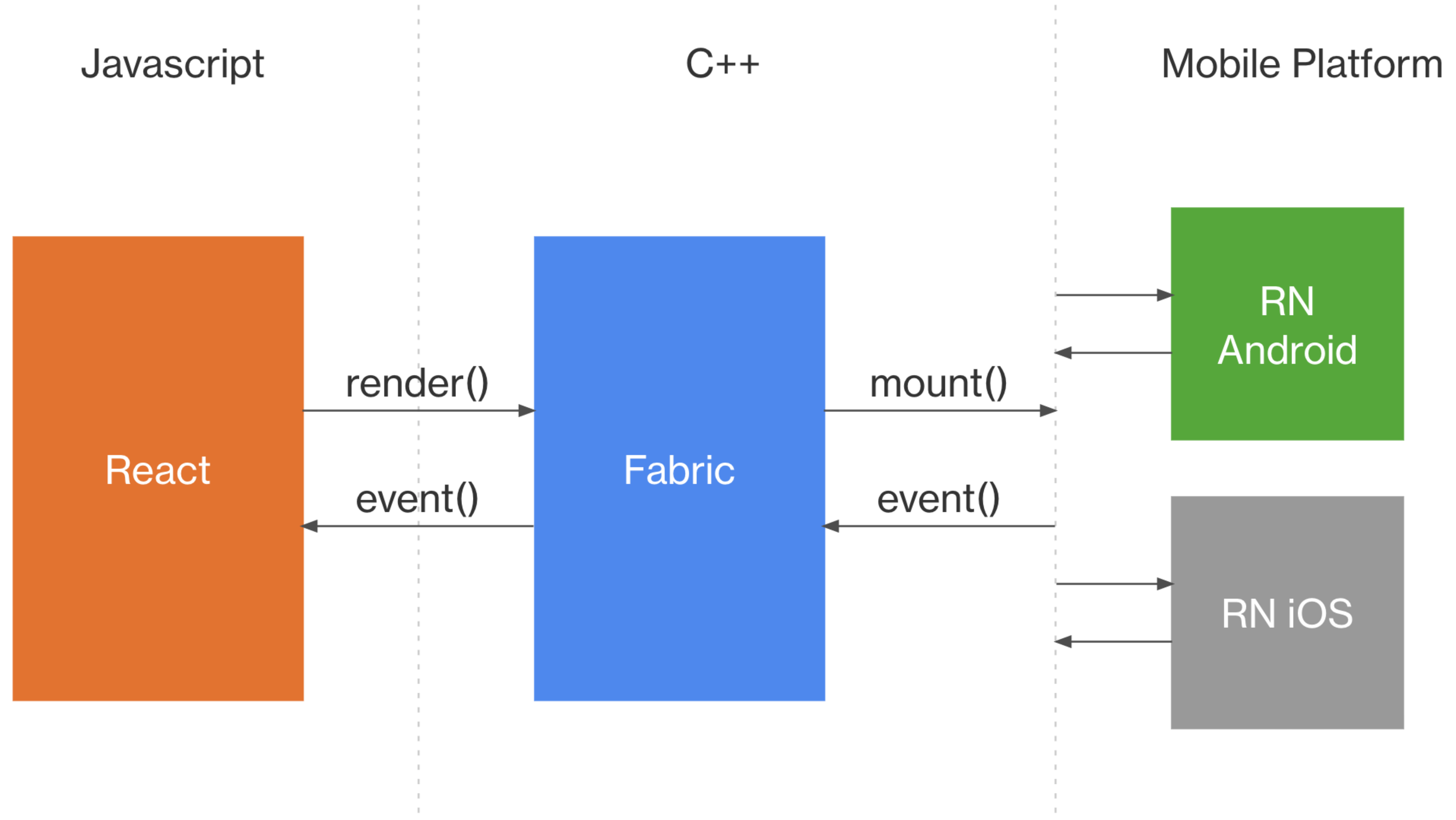


REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
`<View>`	`<ViewGroup>`	`<UIView>`	A non-scrolling `<div>`	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
`<Text>`	`<TextView>`	`<UITextView>`	`<p>`	Displays, styles, and nests strings of text and even handles touch events
`<Image>`	`<ImageView>`	`<UIImageView>`	`<img>`	Displays different types of images
`<ScrollView>`	`<ScrollView>`	`<UIScrollView>`	`<div>`	A generic scrolling container that can contain multiple components and views
`<TextInput>`	`<EditText>`	`<UITextField>`	`<input type="text">`	Allows the user to enter text

Example of Ready-to-Use Native Components (Core Components)

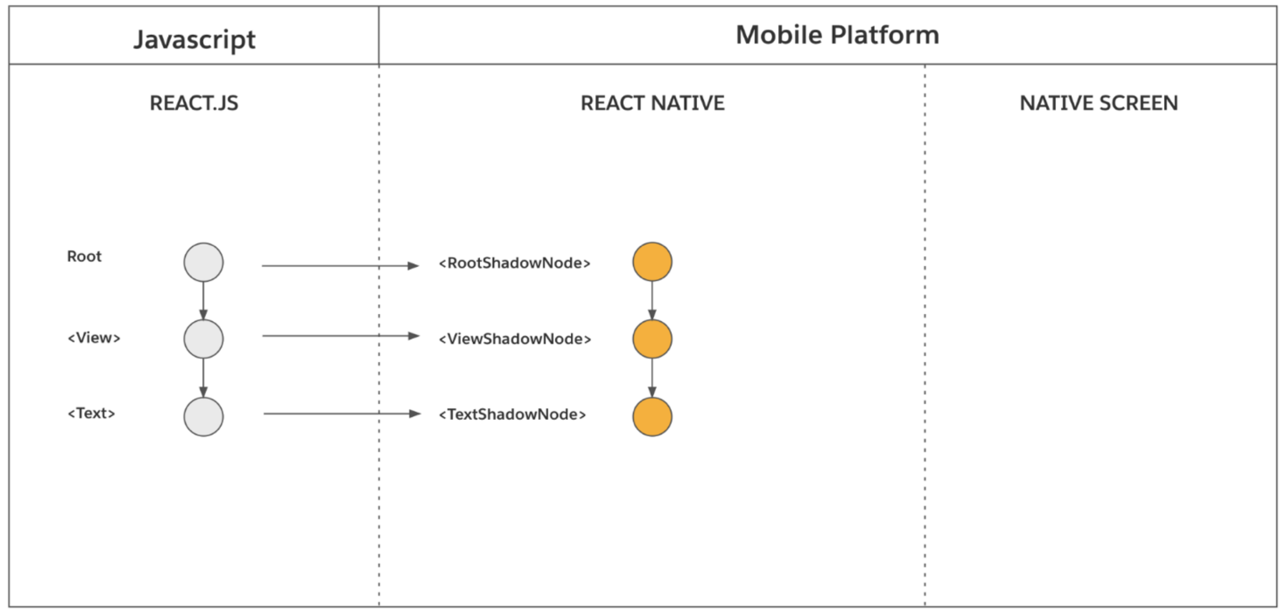
Cross Platform Implementation

The React Native renderer utilizes a core render implementation to be shared across platforms (the current renderer is implemented in C++)

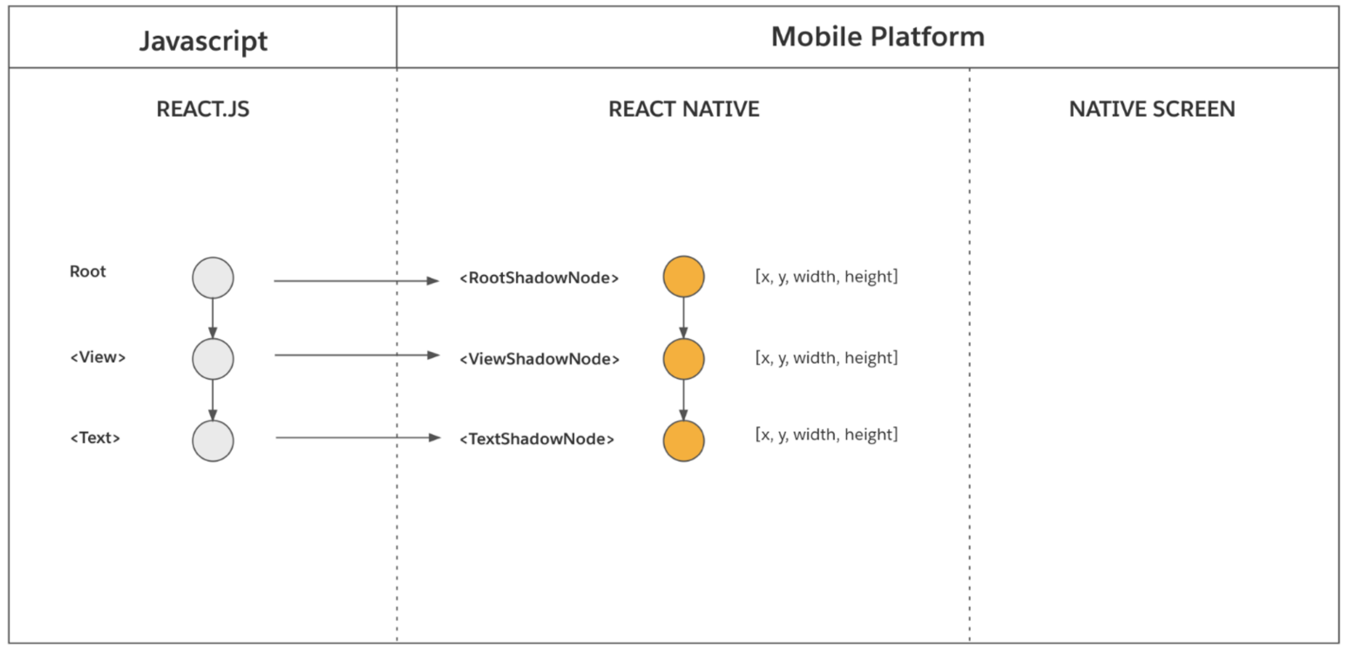


Render, Commit, and Mount

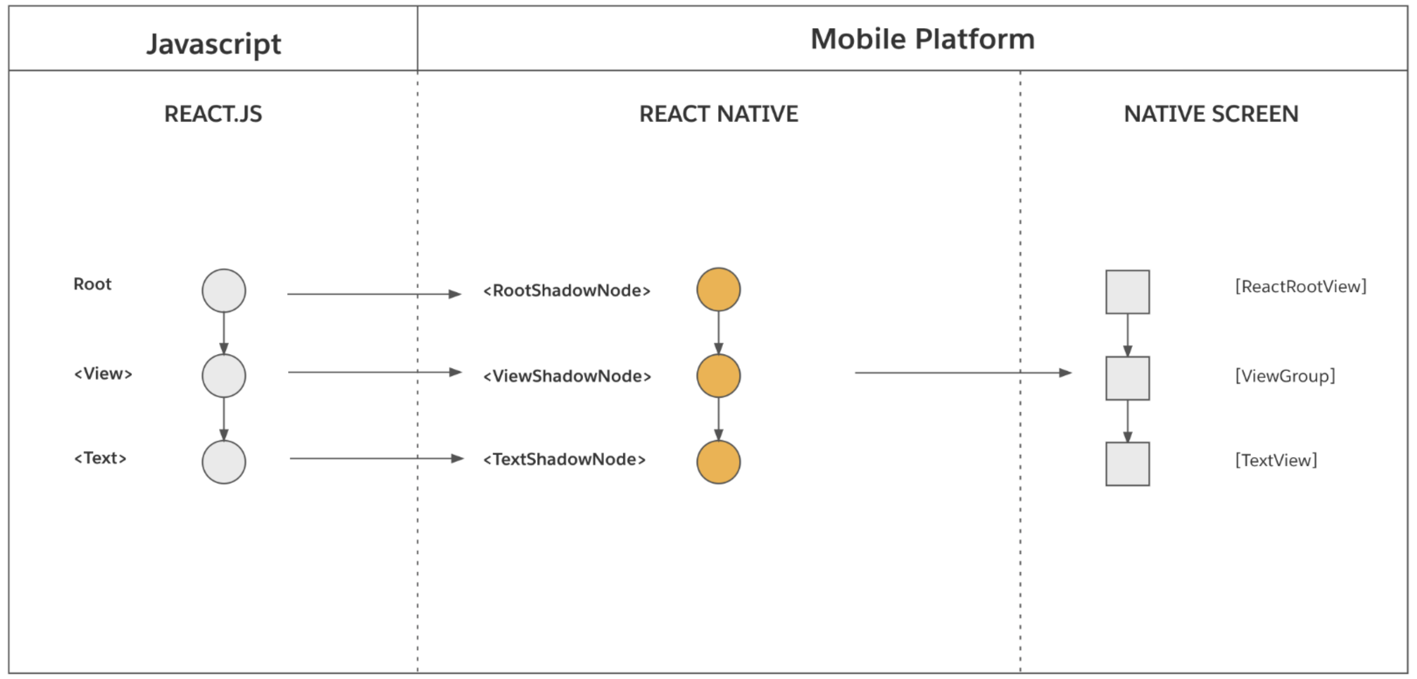
The React Native renderer goes through a sequence of work to render React logic to a host platform. This sequence of work is called the render pipeline.



Render



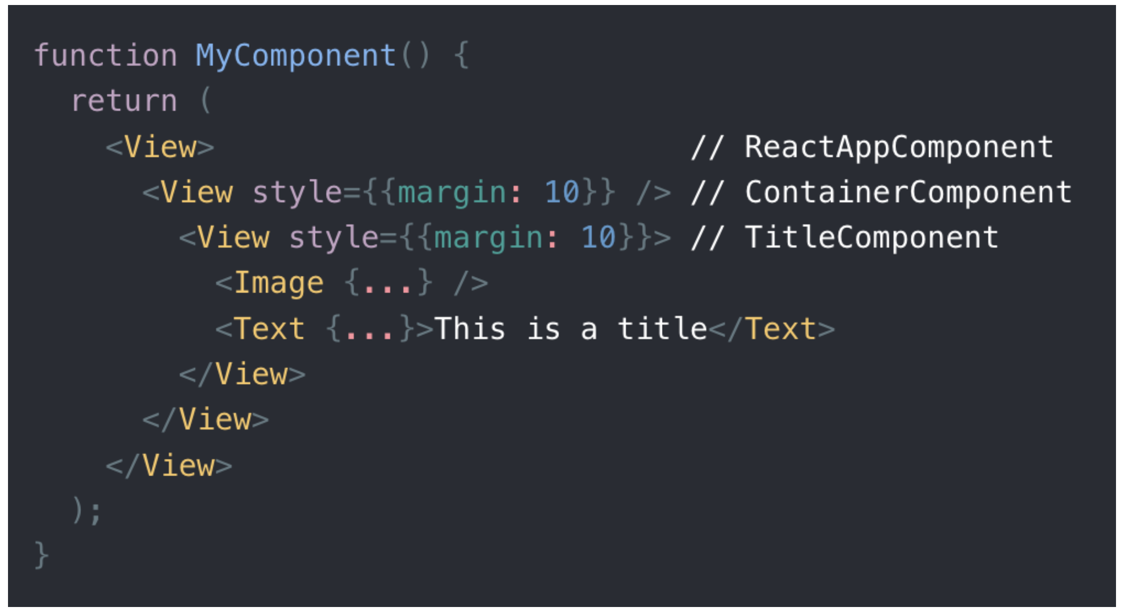
Commit



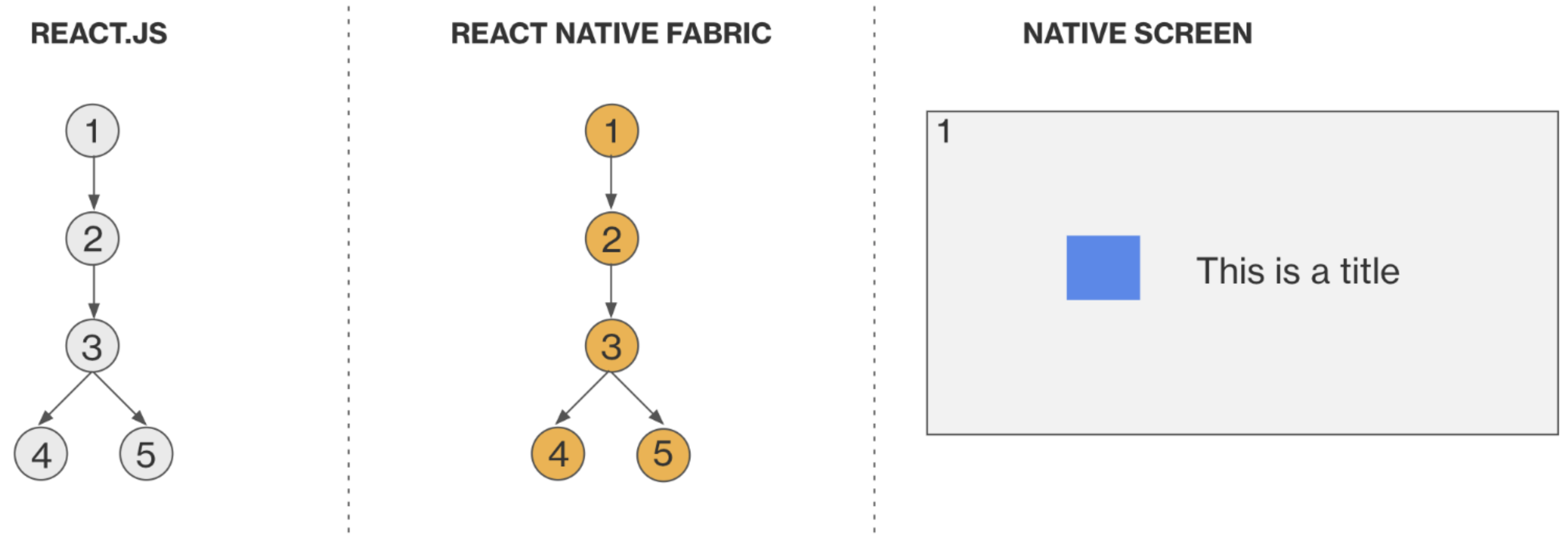
Mount

View Flattening

View Flattening is an optimization by the React Native renderer to avoid deep layout trees. The View flattening algorithm is implemented in C++ and its benefits are shared by default on all supported platforms.



```
function MyComponent() {
  return (
    <View>                          // ReactAppComponent
      <View style={{margin: 10}} /> // ContainerComponent
        <View style={{margin: 10}}> // TitleComponent
          <Image {...} />
          <Text {...}>This is a title</Text>
        </View>
      </View>
    </View>
  );
}
```



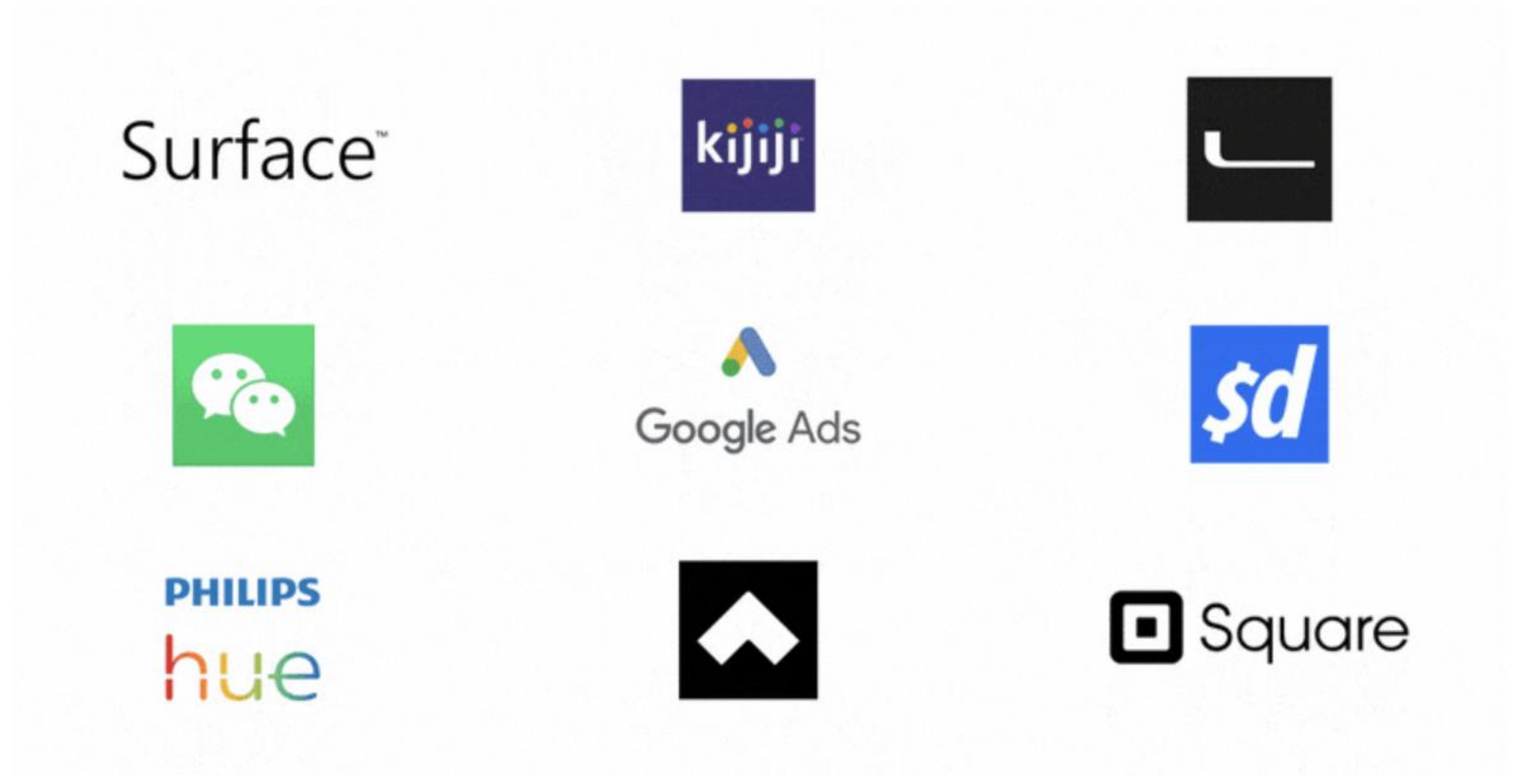
the Views (2) and (3) would be flattened as part of the "diffing algorithm" and as a result their styles will be merged into the View (1)

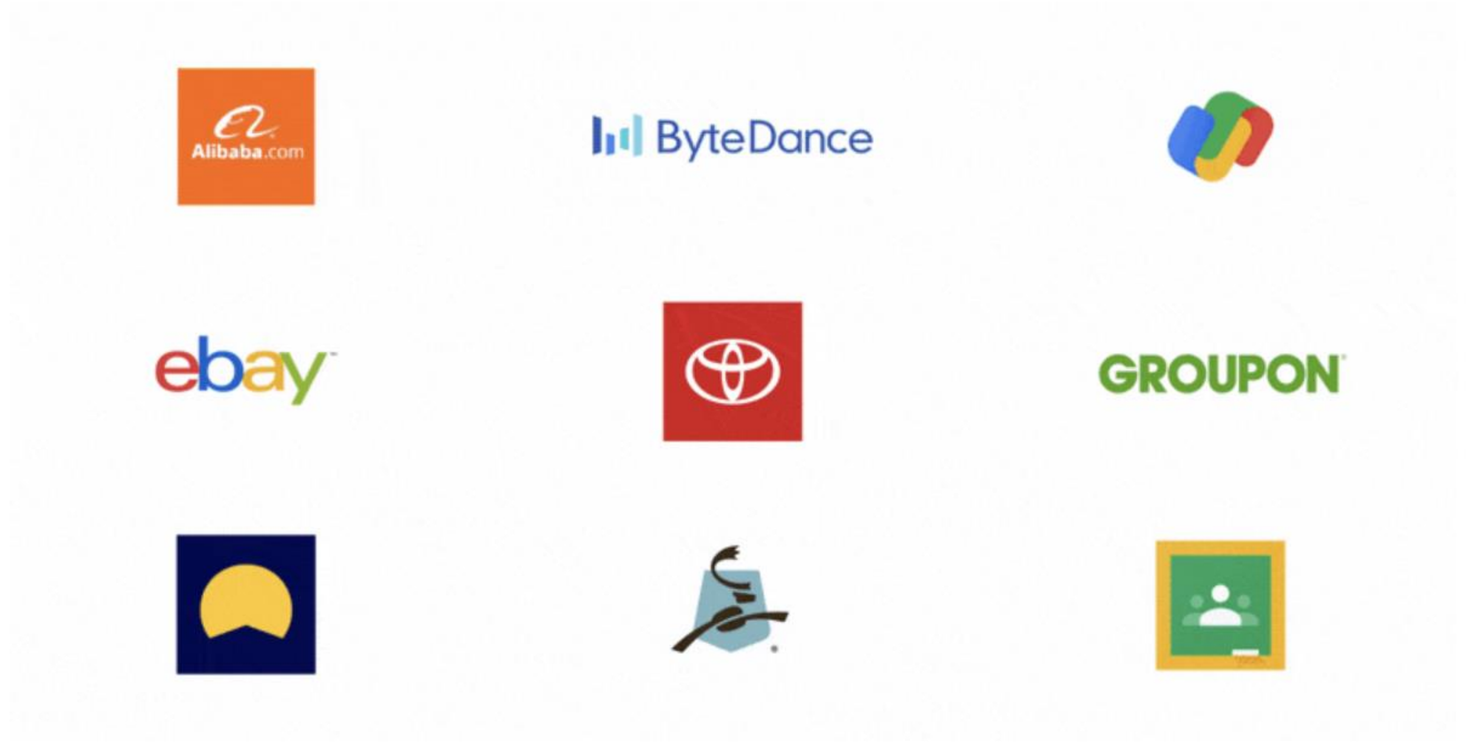
Flutter

- Flutter is an open-source framework for building beautiful, natively compiled, multi-platform applications from a single codebase.

```
MediaQuery.of(context).size.width
```

- Who is Using Flutter

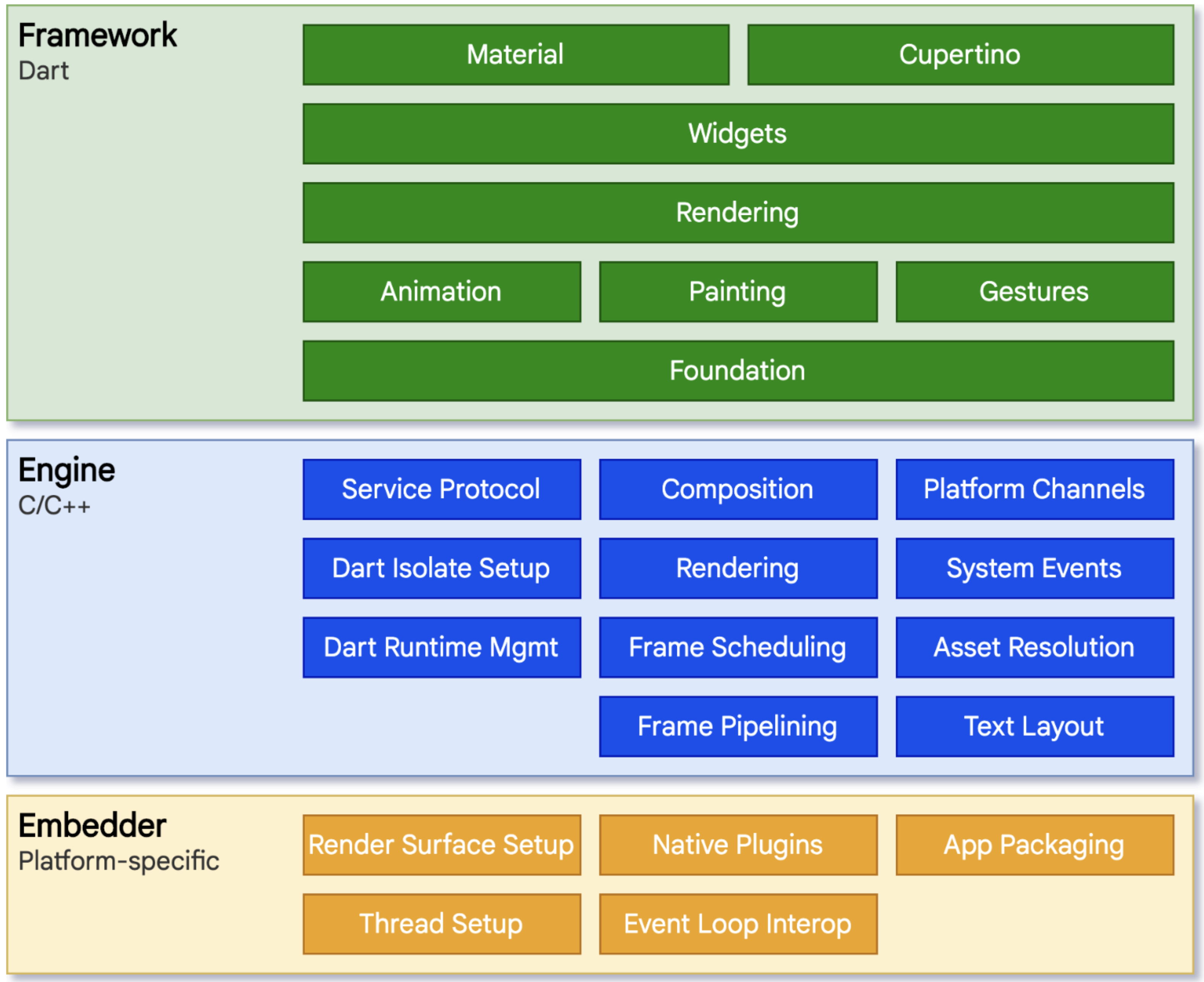




Flutter Background

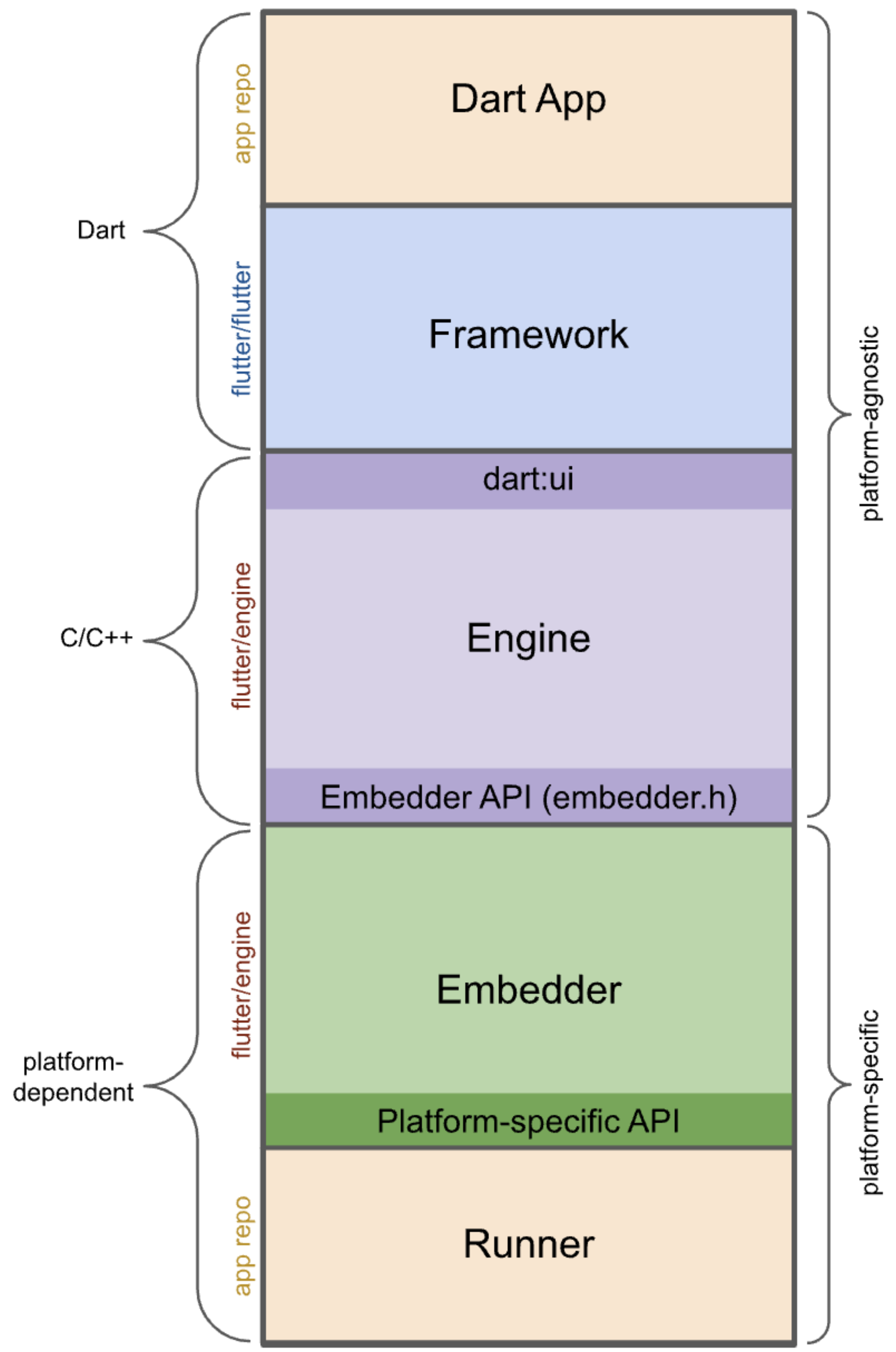
- Released as *Open Source* project by Google in 2017
- Approach to UI
  - Reactive framework (as with Jetpack Compose, and react native)
  - Entire UI built using composition of widgets, creating highly customizable and flexible UI
  - App look and feel the same across different devices.

Flutter Architecture



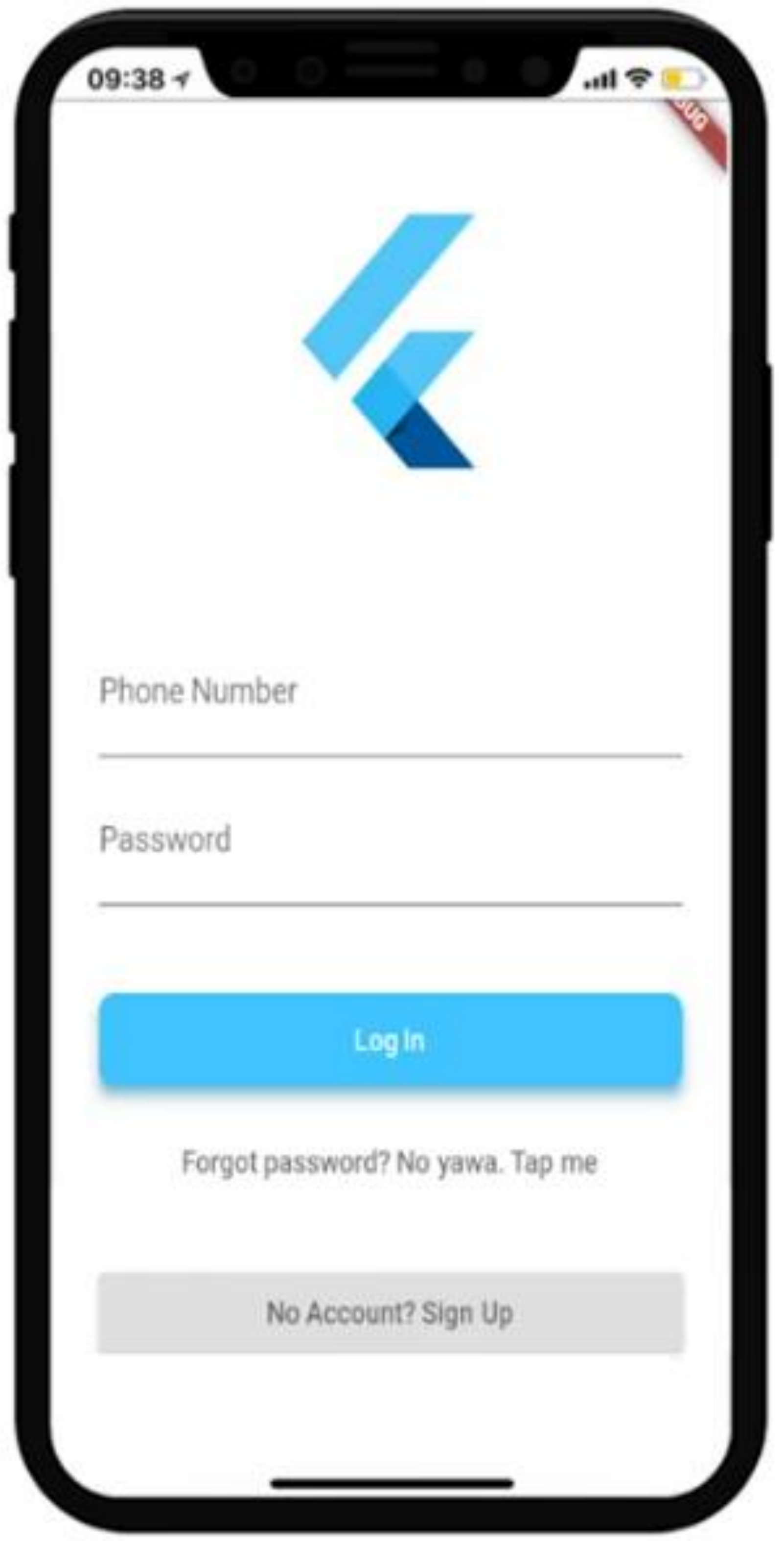
- extensible, layered system design
- It exists as a series of independent libraries that each depend on the underlying layer. No layer has privileged access to the layer below, and every part of the framework level is designed to be optional and replaceable.
- A platform-specific embedder provides an entrypoint; coordinates with the underlying operating system for access to services like rendering surfaces, accessibility, and input; and manages the message event loop.
- Renderer (by default Impeller on iOS and Android / Skia on other platforms)
- The embedder is written in a language that is appropriate for the platform: currently Java and C++ for Android, Swift and Objective-C/Objective-C++ for iOS and macOS, and C++ for Windows and Linux.

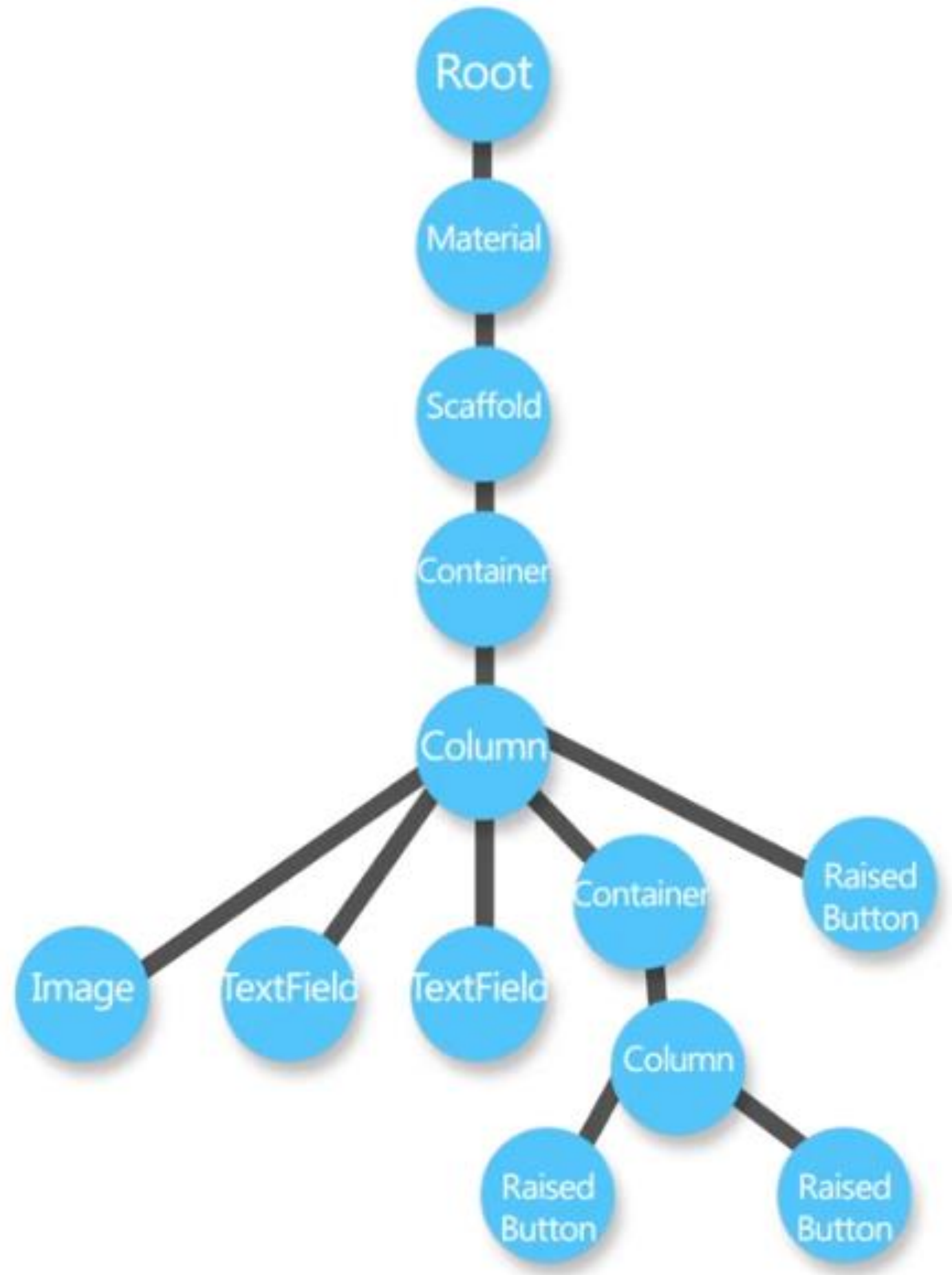
Anatomy of an App



- **Dart App**
  - Composes widgets into the desired UI.
  - Implements business logic.
  - Owned by app developer.
- **Framework** (source code)
  - Provides higher-level API to build high-quality apps (for example, widgets, hit-testing, gesture detection, accessibility, text input).
  - Composites the app's widget tree into a scene.
- **Engine** (source code)
  - Responsible for rasterizing composited scenes.
  - Provides low-level implementation of Flutter's core APIs (for example, graphics, text layout, Dart runtime).
  - Exposes its functionality to the framework using the **dart:ui API**.
  - Integrates with a specific platform using the Engine's **Embedder API**.
- **Embedder** (source code)
  - Coordinates with the underlying operating system for access to services like rendering surfaces, accessibility, and input.
  - Manages the event loop.
  - Exposes **platform-specific API** to integrate the Embedder into apps.
- **Runner**
  - Composes the pieces exposed by the platform-specific API of the Embedder into an app package runnable on the target platform.
  - Part of app template generated by flutter create, owned by app developer.

Widget Tree

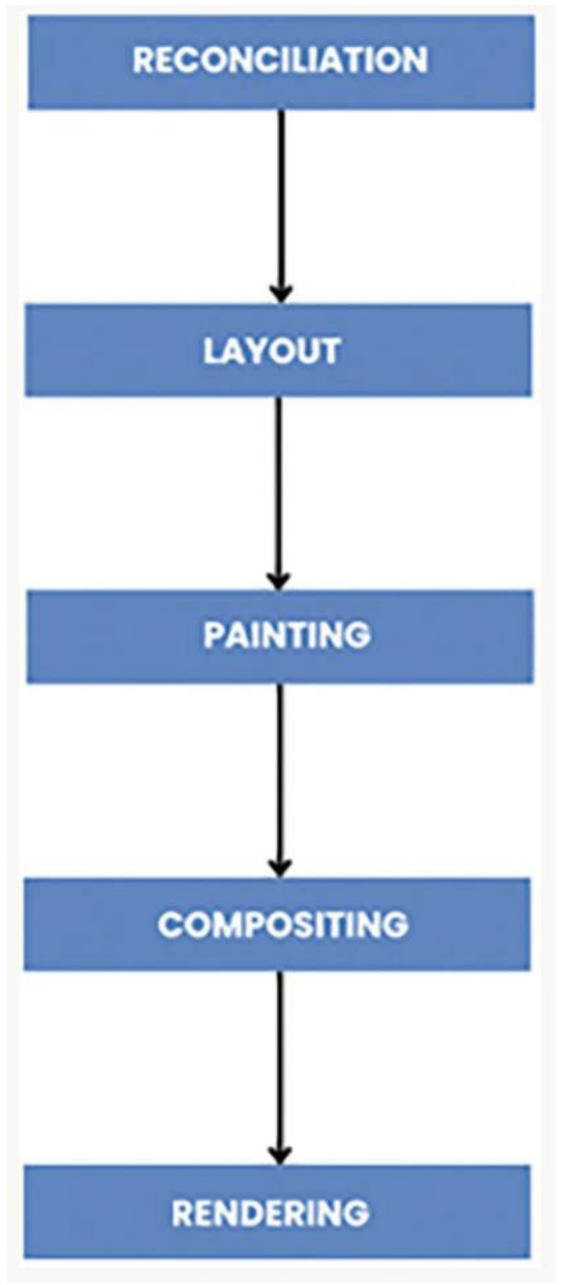




- As in View in Android, everything is Widget in Flutter
- Stored hierarchicaly in Tree data structure
- Widget may be stateless or stateful.

https://blog.logrocket.com/complete-guide-flutter-architecture/#widget-tree

Flutter Rendering Process



- Render Process
  - Building UI using Widget: creating tree-like data structure where each widget represent visual element
  - Widget Tree is representation of the visual aspect, but it is not directly visible to user
  - During render, Flutter takes the widget tree and runs several steps to transform it into actual pixels as follows:
- Reconciliation
  - Perform linear reconciliation (unlike React's tree diff) to find which widgets needs to be updated
- Layout
  - Once the differences are identified, Flutter performs layout process to determine size and position of each widget.
- Painting
  - After the layout determined, Flutter assigns colors, gradients, and other visual attributes to each widget. It paints each widgets onto canvas.
- Compositing:
  - Taking painted widgets and combines them to create final image that will be displayed on screen. Taking account of transparency , overlapping and layering
- Rendering:
  - Final process that takes the composite image and coverts onto platform specific graphcs instruction that are sent to GPU.

Framework Choice Guidelines

- Choosing React Native is recommended for companies/developers who have a strong JavaScript or Web background since it is easy to reuse available skills/resources that will lead to reducing development cost and time.
- Choosing Flutter is recommended for companies who have enough budget and resources, since Dart is not widely used, learning Flutter might take more time.
- Both Flutter and React Native have their strengths and weaknesses in mobile application development (Fentaw, A. L., 2020, G Tollin,2023)
- Both Flutter and React Native are under continuous improvements and soon both frameworks might support loads more near-native functionalities than they support now.

Fentaw, A. L., 2020, "Cross platform mobile application development: a comparison study of React Native Vs Flutter", Master's Thesis, University Of JYVÄSKYLÄ

Tollin, G., & Marcus, L. (2023). React Native vs. Flutter: A performance comparison between cross-platform mobile application development frameworks.

Referensi

- Ahmad, A., Li, K., Feng, C., Asim, S. M., Yousif, A., dan Ge, S. (2018): An Empirical Study of Investigating Mobile Applications Development Challenges, *IEEE Access*, **6**, 17711–17728. https://doi.org/10.1109/ACCESS.2018.2818724
- Fentaw, A. L., 2020, "Cross platform mobile application development: a comparison study of React Native Vs Flutter", Master's Thesis, University Of JYVÄSKYLÄ
- StackOverflow Developer Survey 2023, https://survey.stackoverflow.co/2023/
- React Native View Flattening Algorithm, https://reactnative.dev/architecture/view-flattening
- React Native Cross Platform Implementation, https://reactnative.dev/architecture/xplat-implementation
- React Native Cross Rendering Pipeline, https://reactnative.dev/architecture/render-pipeline
- Mobile Application Development Definition, https://www.ibm.com/think/topics/mobile-application-development
- Mobile Application Development Definition, https://aws.amazon.com/mobile/mobile-application-development/
- Mobile Application Development Definition, https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-mobile-app-development
- https://roadmap.sh/android
- https://roadmap.sh/ios

THANK YOU