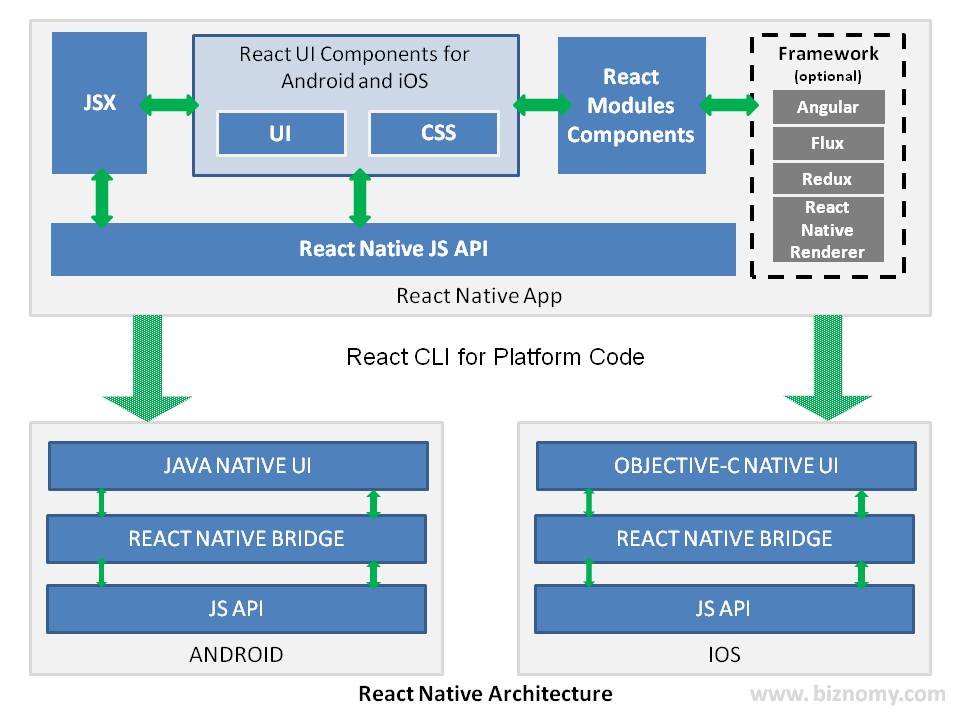
React Native

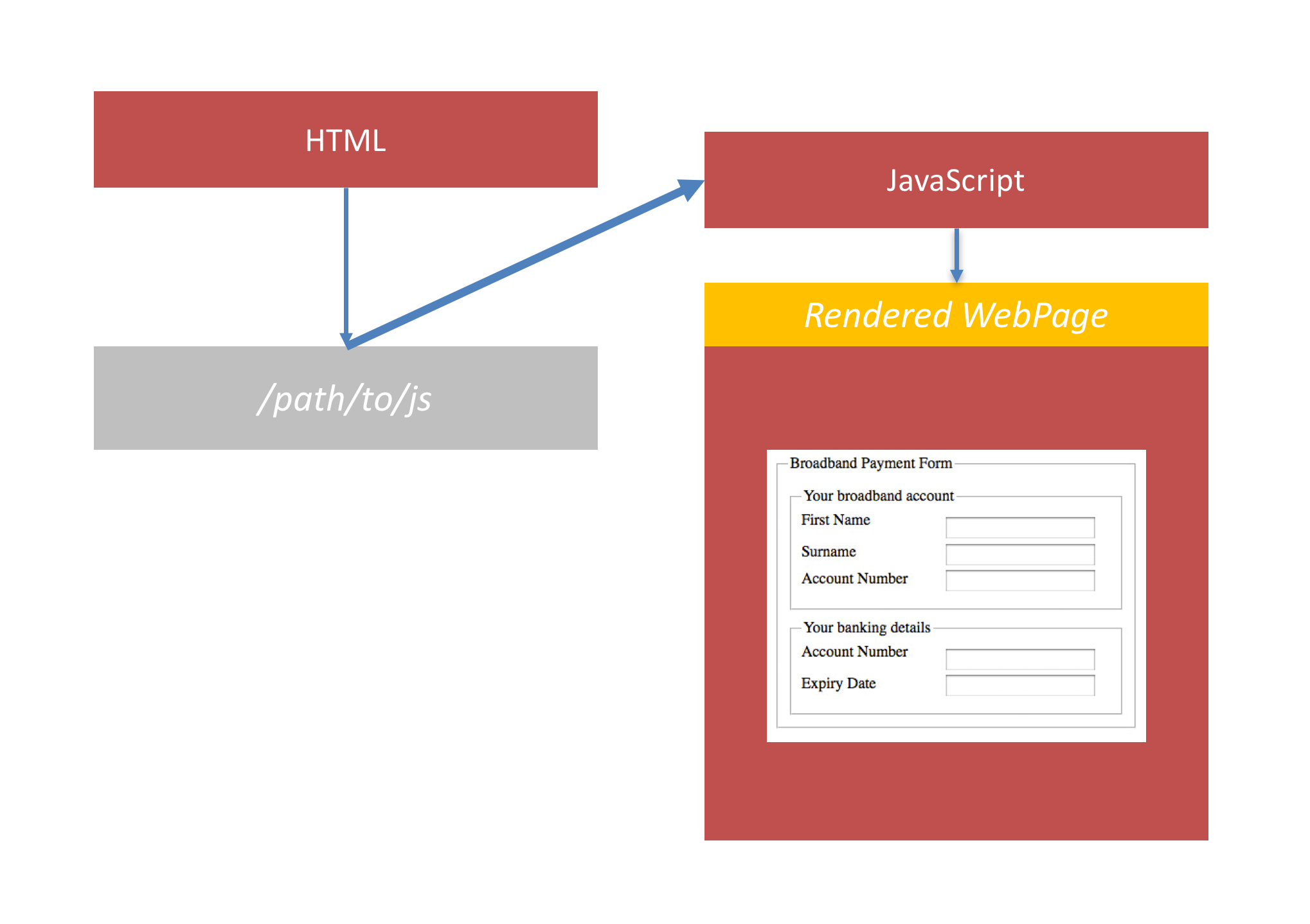
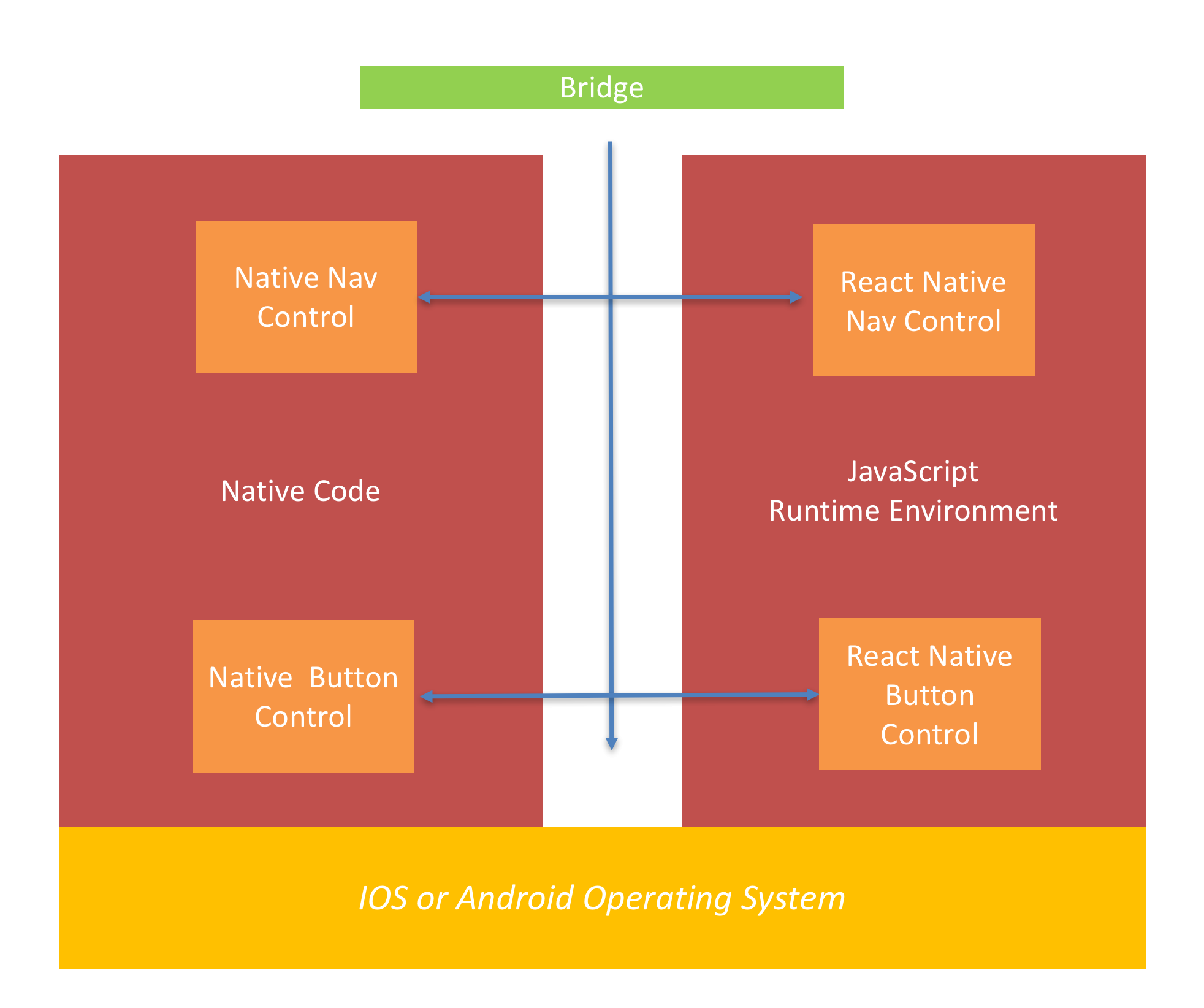
Background and motivation

The mobile development world is dominated worldwide by two well-known operating systems:Android and iOS. Regardless of (…), a mobile developer will always be looking for some aspects during his work: shorter development cycles, quicker time to deployment and better app performance.

The development of the React Native framework started as a hackathon effort in 2013, responding to the need of the developer community(….): it was looking for a mobile alternative that would combine the advantages of mobile app development with the power and agility of the native React environment. This resulted in React Native, a framework created by Facebook, and supported and continuously pushed forward by the development community.

One of its main advantages is code reuse, cost and time saving, reusing the code written for development for both Android and iOS.According to recent calculations, about 90% of the code can be reused between Android and iOS, but, on the other hand, it takes a while and some extra “resources” for environment settings(the setup) and practical modifications.



Technical references & Architecture

React Native lets you build mobile apps using only Javascript, allowing us to compose a rich mobile UI using declarative components.It uses the same fundamental UI building blocks as regular iOS and Android apps, but put together using Javascript and React. React native lets you reload the app, instead of compiling it again.With [Hot Reloading](http://facebook.github.io/react-native/blog/2016/03/24/introducing-hot-reloading.html), you can even run new code while retaining your application state.

Modular and intuitive architecture similar to React

Easy to start coding on someone else project( after cloning a project)=> flexibility among the devs of a team for example.

Easy to install library and updated, tricky at times, and the bugs are pretty tricky and some in debates(…)

Less time for testing, as the flow of the technical solution is easier to understand (moularity and component- based) in order to make test cases and scenarios.

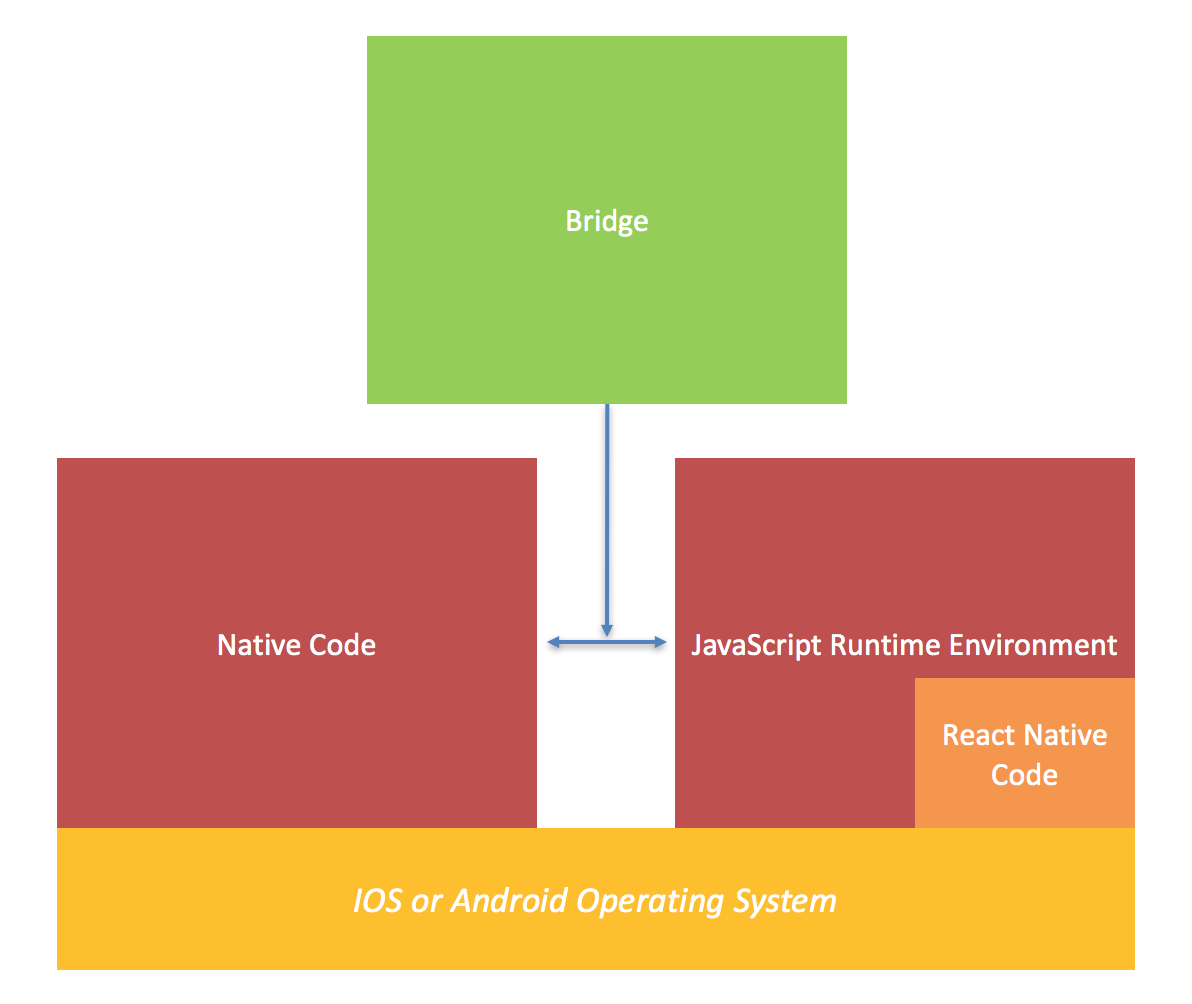
This leads to a valuable resource saving: the time, which is something and product owner or customer would like to hear

1. The runtime architecture - and how bridging connects our native code to the JavaScript code through the JavaScript runtime engine (core)
2. The build architecture - how React Native simply produces an output on a Node server called a bundle and the device downloads it, this is similar to how a SPA works
3. The debugging architecture - React Native also publishes debugging to a webpage (also hosted by our node server) which we can browse and debug using Google Chrome

JavaScript Web applications, known as Single Page Applications(SPAs) ruined in a browser.Here two things are loaded by the browser: the HTML document (which contains a reference to the JS) and Javascript itself. The final output of a SPAs is a rendered page having the expected UI, functionality, validations, forms, etc.React Native build process works in a very similar way, but instead of having a browser which downloads the js and html, the device or emulator will load the js and the markup language.

The JavaScript Runtime Environment Architecture (Bridging)

Because JavaScript is not the language that runs natively on the phone we must use a technique called bridging to allow JavaScript to run and to communicate with the phones processor.

the phone must run something called the ‘JavaScript core’; which is basically a ‘runtime environment’. 

The native code is actually operating on it’s own thread, meaning that if our React Native JavaScript code is taking a long time the UI won’t hang or slow down. It simply calls back across the bridge using events.

source=https://www.logicroom.co/react-native-architecture-explained/images/1\_image.png

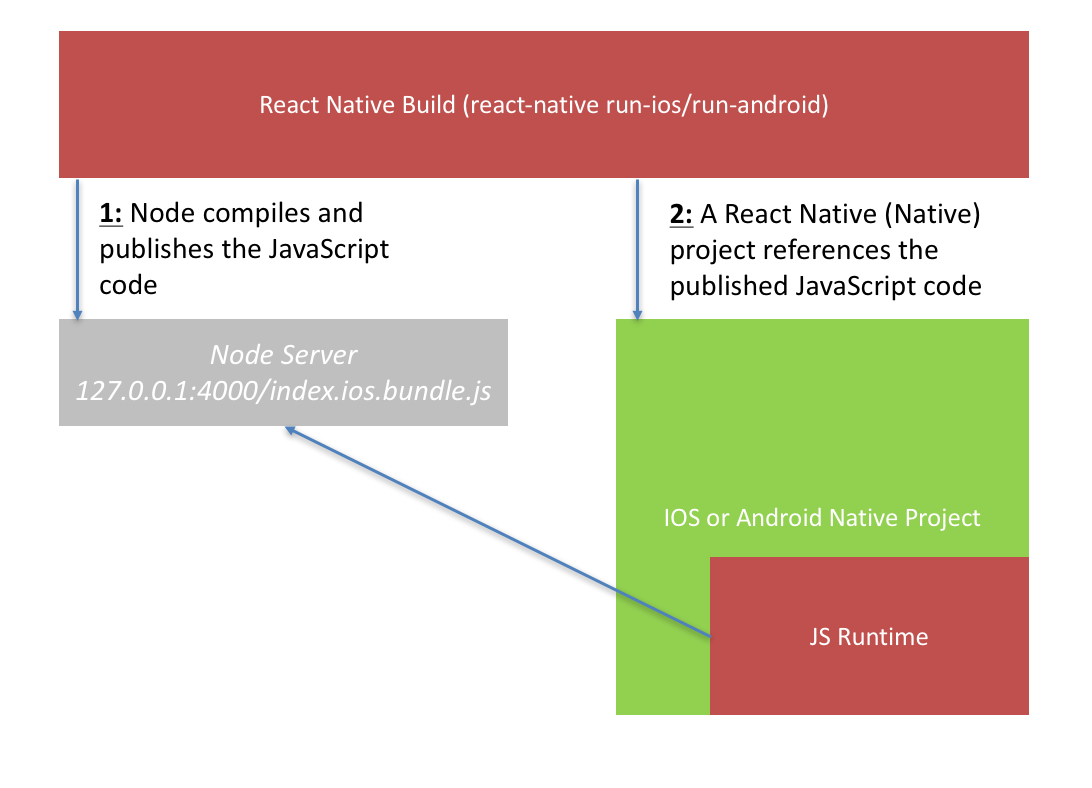
The React Native Build Process Architecture

react-native run-ios / react-native run-android

1. It will run a node web server locally which publishes a payload file index.platform.bundle. that holds ‘all’ of our React Native JavaScript code.

2.Then it will build a pure native application project (for IOS or Android) and deploy this to the phone.

Any React Native project has an embedded IOS and an Android project into it



Hot Reloading

Because it’s Javascript which does not need to be compiled and build, we can simply reload the React Native app, being able therefore to see the changes in real time.This ability to re-load is called ‘Hot Reloading’ and allows persisting the app’s state through reloads.

Debugging Architecture

In order to run a debugger we need to set manually the server ip:port addresses and to open a browser that the debugger can publish debug.

* Firstly, the app bundle is hosted on localhost:8089/index.platform.bundle Secondly, the debugger is hosted on localhost:8089/debugger-ui
* Thirdly, a process in terminal will stay open which reports the build process output.

For furthermore permissions and developer options, check the dev settings on the device.

Strong performance for mobile environments

The React Native architecture is very well tuned to mobile devices. It makes use of the GPU (Graphics Processing Unit), while native platforms are more ‘CPU (Central Processing Unit) intensive’. Compared to hybrid technologies – which was the only option for cross-platform in the past – React Native is very fast.

Simplified UI

DOM and Styling

React-Native doesn’t use HTML to render the app, but provides alternative components that work in a similar way. Those React-Native components map the actual real native iOS or Android UI components that get rendered on the app.Most components provided can be translated to something similar in HTML, where for example a View component is similar to a div tag, and a Text component is similar to a p tags.

To style the components, a stylesheet( usually an array of styles) in Javascript has to be created, which looks similar to CSS but its not CSS. As React-Native is not made from web elements, it cannot be styled in the same way and neither can override the style of a component for reuse. React Native also supports camel-case properties and FlexBox.

The animations and not CSS anymore, they have to be build with javascript and the recommended way to animate a component is to use [Animated](https://facebook.github.io/react-native/docs/animated.html) API provided by React-Native.

Platform Specific Code

One of React Natives main advantages is building a cross-platform app, so much of the code will be reused but, however, there may be needed to implement some different visual components for iOS and Android. It provides two options for organizing the code and separate it by platform: using the Platform module and platform-specific file extensions.

Certain components may be designed to work only on a specific platform and there props are annotated with @platform and have a small badge next to them on the website.

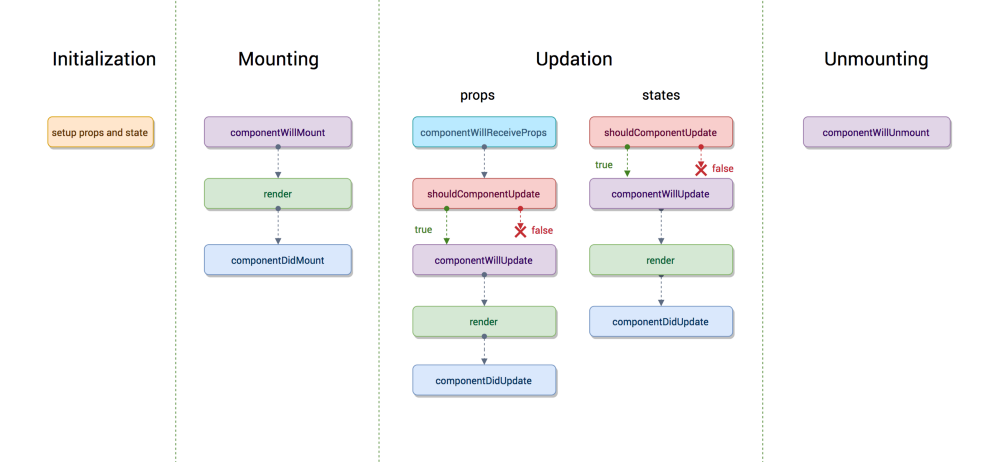
By including the Platform component from the react-native library the compiler detects the platform in which the app is running and the developer can use the detection logic to implement platform-specific code.The Platform.select method , returns the value for the platform you are currently running on.

When your platform-specific code is more complex, you should consider splitting the code out into separate files. React Native will detect when a file has a .ios. or .android. extension and load the relevant platform file when required from other components.

Components

There are two types of data that control a component: **props** and **state**.

Props are immutable and are set by the parent, while state property is immutable, it can be initialized in the constructor and updated by calling setState.The **Store** object holds the whole state tree of the application.The only way to change the state inside it is to dispatch an action on it.

Component Lifecycle

React-Native lifecycle phases and methods

The order of lifecycle should be: **constructor() -> componentWillMount() -> render() -> componentDidMount() -> [ runtime loop ] -> componentWillUnmount()**

The runtime loop has two possible situations: **[ new props coming update ] -> componentWillReceiveProps() -> shouldComponentUpdate() -> componentWillUpdate() -> render() -> componentDidUpdate() -> [ runtime loop ]**

The new props data will trigger the componentWillReceiveProps() function but new state data will not.

App State

AppState can tell you if the app is in the foreground or background, and notify you when the state changes and has three states: active, background or inactive.It is frequently used to determine the intent and proper behaviour when handling push notifications.

When user leave the app, the app become into background state, all Javascript code being halted and unloaded from memory and all timers (setTimeout) are unable to execute, because their don’t found the codes to run in the memory. Therefore, an external library is needed for running in the background.

Linking Libraries

All the libraries shipped with React Native live on the Libraries folder in the root of the repository. Some of them are pure Javascript and are used with the keyword **required**.Other libraries also rely on some native code, in that case the user has to add these files to the app, otherwise the app will throw an error as soon as the is gain access to the library.