**Q1. Execution process of react native code.**

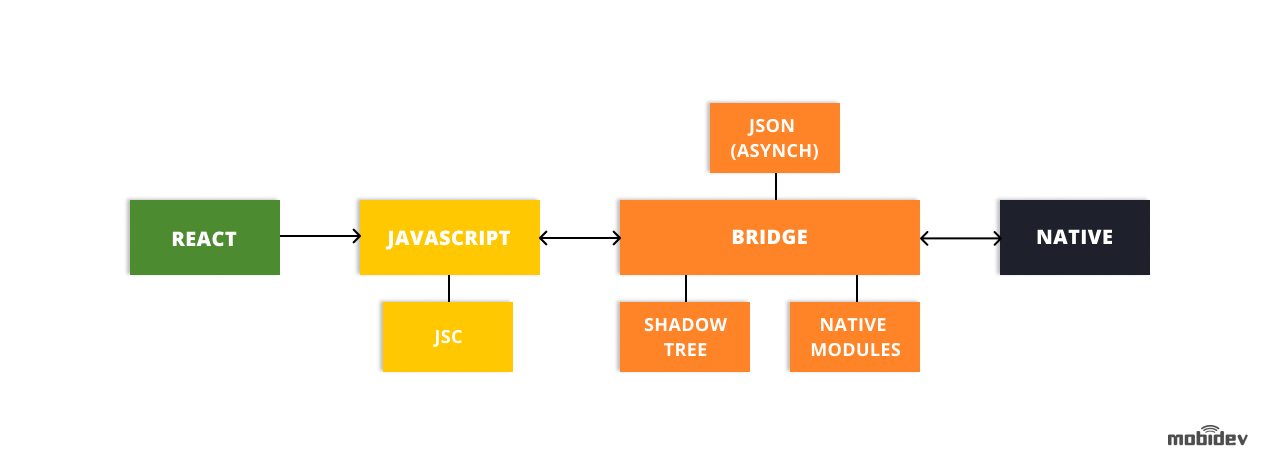
**Ans.**

If we consider the big picture, there are three parts to the RN platform:

* **Native Code/Modules**: Most of the native code in case of iOS is written in Objective C or Swift, while in the case of Android it is written in Java or Kotlin. But for writing our React Native app, we would hardly ever need to write native code for iOS or Android.
* **Javascript VM**: The JS Virtual Machine that runs all our JavaScript code. In the case of iOS, React Native uses the JavaScriptCore provided by the iOS platform. In the case of Android, React Native bundles the JavaScriptCore along with the application. This increases the app size.
* **React Native Bridge:** React Native bridge is a C++/Java bridge which is responsible for communication between the native and Javascript thread. A custom protocol is used for message passing.

Bridge work as Asynchronous, Batched, Serializable.

Now, whenever the React Native app is launched, the first item to be loaded is the native entry point. The Native thread spawns the JS thread which runs the bundled JS code. The JS code has all the business logic of the application. The Native thread now sends messages via the RN Bridge to start the JS application. Now, the spawned JS thread starts issuing instructions to the native thread via the RN Bridge. The instructions include what views to load, what information is to be retrieved from the hardware, etc.



When a React Native application is launched, it spawns up the following threading queues.

**Main thread (Native Queue)** - This is the main thread which gets spawned as soon as the application launches. It loads the app and starts the JS thread to execute the Javascript code. The native thread also listens to the UI events like 'press', 'touch', etc. These events are then passed to the JS thread via the RN Bridge.

Once the Javascript loads, the JS thread sends the information on what needs to be rendered onto the screen. This information is used by a shadow node thread to compute the layouts. The shadow thread is basically like a mathematical engine which finally decides on how to compute the view positions. These instructions are then passed back to the main thread to render the view.

**Javascript thread (JS Queue)** - The Javascript Queue is the thread queue where the main bundled JS thread runs. The JS thread runs all the business logic, i.e., the code we write in React Native.

**Custom Native Modules** - Apart from the threads spawned by React Native, we can also spawn threads on the custom native modules we build to speed up the performance of the application. For example - Animations are handled in React Native by a separate native thread to offload the work from the JS thread.

**Q2. Which is better for react native app - Javascript vs typescript**

**Ans.**

There is no simple yes and no answer to this question. Whether to use JavaScript or TypeScript depends on the project we are working on.

Let’s see the advantages of TypeScript over JavaScript-

* TypeScript always points out the compilation errors at the time of development only. Because of this at the run-time, the chance of getting errors are very less whereas JavaScript is an interpreted language.
* TypeScript has a feature which is strongly-typed or supports static typing. That means it allows for checking type correctness at compile time. This is not available in JavaScript. Results in early bug catching and structured code.
* Brings new features like type annotation, classes and modules.

**JavaScript Vs TypeScript**

* typescript is an Object-oriented programming language whereas JavaScript is a scripting language.
* Typescript code needs to be compiled while JavaScript code doesn’t need to compile. TypeScript takes time to compile the code making it a bit slow.
* TypeScript makes the code more consistent, clean, simple and reusable. So it is better to use TypeScript for large projects whereas JavaScript is preferable to use in small coding projects.

Use cases for TypeScript:

* We have a large codebase.
* We need Static typing and speed is a priority (since it catches bugs in real-time).

Use cases for JavaScript

* Extra time to compile the code isn’t an option.
* Flexibility is a priority (as it offers dynamic typing)

Typescript is just compiled to javascript. Think of Typescript as a tool like eslint that helps us write better code, just in a different way. There will be no performance change to the user, only difference is in whatever step we do to build our application will now have to have a typescript parser that compiles it.

**Q3. Can I do native changes in case of expo?**

**Ans.**

The downside to using Expo is that we cannot extend the native functionality while writing our app. Which means, if we wish to integrate a third-party native library or extend app functionality with native code, then we simply cannot do it. This is because Expo projects do not reveal the native iOS and Android projects that react-native CLI does.

**Q4. Can we convert a react expo app to react-native CLI?**

**Ans.**

Yes, Expo provides a way for us to eject the app to React-Native-CLI at any stage of the development process. So, we can integrate any native Java/ iOS code into the project or bring in any functionalities that haven’t been supported by Expo.

**Q5. What are the limitations of react-native expo?**

**Ans.**

* Native Java or iOS code can’t be connected with Expo. Whereas with React-Native-CLI, this can be done.
* Expo is a third-party service, it may take time to get up to date with the React-Native-CLI features which are developed by the React Native Team and the open-source community.
* Not all iOS and Android APIs are available yet - Bluetooth, In-App Purchases are the most painful.
* When our app needs background code execution - one of the problematic topics. That is about the situation when the app works in the background, plays audio, handles push-notifications in the background are in development for several years.
* Huge app size for simple applications - That is because you have to store all Expo SDK modules in managed workflow even when they are not in use.
* The only supported push notification service goes from Expo called Expo Push Notification service/API.
* Expo only supports Android 5+ and iOS 10+ OS versions.