Why a New Architecture

⚠ **CAUTION**

This documentation is still **experimental** and details are subject to changes as we iterate. Feel free to share your feedback on the discussion inside the working group for this page.

Moreover, it contains several **manual steps**. Please note that this won't be representative of the final developer experience once the New Architecture is stable. We're working on tools, templates and libraries to help you get started fast on the New Architecture, without having to go through the whole setup.

The goal of the New Architecture is to solve some of the issues that afflicted the Old Architecture in terms of performance and flexibility. This section provides the basic context to understand the Old Architecture's limitations and how it has been possible to overcome them with the New Architecture.

This is not a technical deep dive: for further technical information, refer to the Architecture tab of the website.

Old Architecture's Issues

The Old Architecture used to work by serializing all the data that had to be passed from the JS layer to the native layer using a component called *The Bridge*. *The Bridge* can be imagined as a bus where the producer layer sends some data for the consumer layer. The consumer could read the data, deserialize it and execute the required operations.

The Bridge had some intrinsic limitations:

- **It was asynchronous:** one layer submitted the data to the bridge and asynchronously "waited" for the other layer to process them, even when this was not really necessary.
- **It was single-threaded:** JS used to work on a single thread; therefore, the computation that happened in that world had to be performed on that single thread.

- **It imposed extra overheads:** every time one layer had to use the other one, it had to serialize some data. The other layer had to deserialize them. The chosen format was JSON for its simplicity and human-readability, but despite being lightweight, it was a cost to pay.

New Architecture's Improvements

The New Architecture dropped the concept of *The Bridge* in favor of another communication mechanism: the *JavaScript Interface (JSI)*. The *JSI* is an interface that allows a JavaScript object to hold a reference to a C++ and vice-versa.

Once an object has a reference to the other one, it can directly invoke methods on it. So, for example, a C++ object can now ask a JavaScript object to execute a method in the JavaScript world and viceversa.

This idea allowed the unlocking of several benefits:

- **Synchronous execution:** it is now possible to execute synchronously those functions that should not have been asynchronous in the first place.
- **Concurrency:** it is possible from JavaScript to invoke functions that are executed on different threads.
- **Lower overhead:** the New Architecture doesn't have to serialize/deserialize the data anymore; therefore there are no serialization taxes to pay.
- **Code sharing:** by introducing C++, it is now possible to abstract all the platform agnostic code and to share it with ease between the platforms.
- **Type safety:** to make sure that JS can properly invoke methods on C++ objects and vice-versa, a layer of code automatically generated has been added. The code is generated starting from some JS specification that must be typed through Flow or TypeScript.

These advantages are the foundations of the New Native Module System and a jumping stone to further enhancements. For example, it has been possible to develop a new renderer which offers faster and more performant Native Components.

Further Reading

For a technical overview of the New Architecture, read the Architecture tab.

For more information on the Fabric Renderer, read the Fabric section.