

Threading Model

CAUTION

This document refers to the architecture of the new renderer, [Fabric](#), that is in active roll-out.

The React Native renderer distributes the work of the render pipeline across multiple threads.

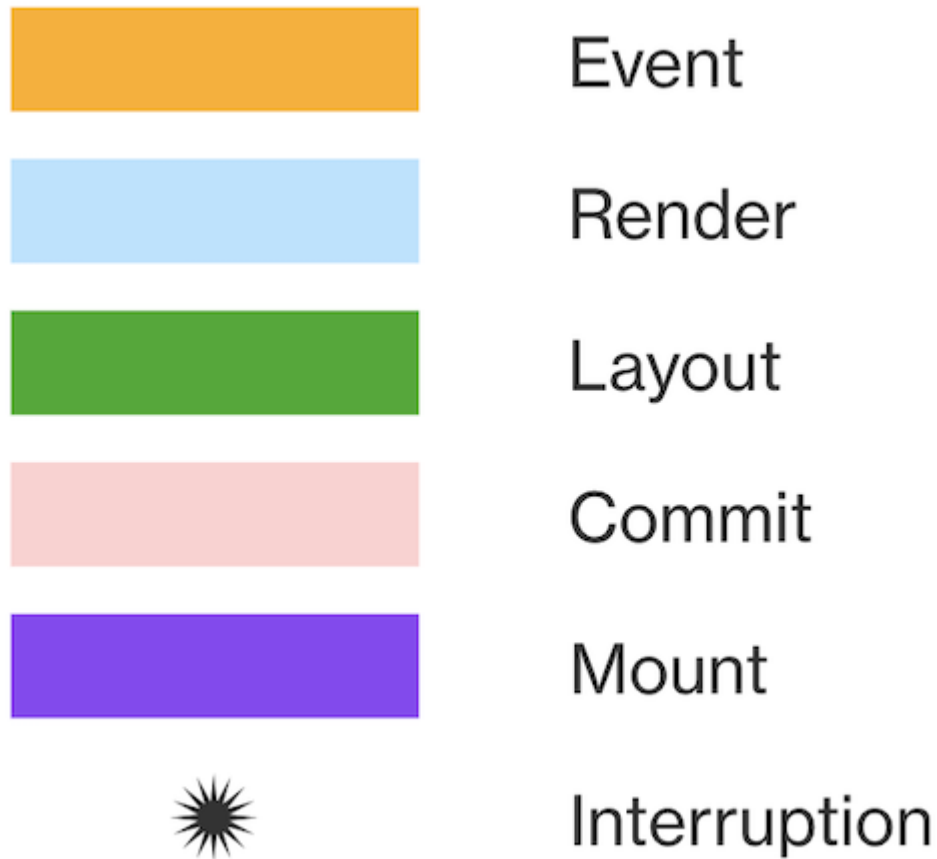
Here we define the threading model and provide some examples to illustrate thread usage of the render pipeline.

React Native renderer is designed to be thread safe. At a high level thread safety is guaranteed by using immutable data structures in the internals of the framework (enforced by C++ “const correctness” feature). This means that every update in React creates or clones new objects in the renderer instead of updating data structures. This allows the framework to expose thread safe and synchronous APIs to React.

The renderer uses three different threads:

- **UI thread** (often called main): The only thread that can manipulate host views.
- **JavaScript thread**: This is where React’s render phase is executed.
- **Background thread**: Thread dedicated to layout.

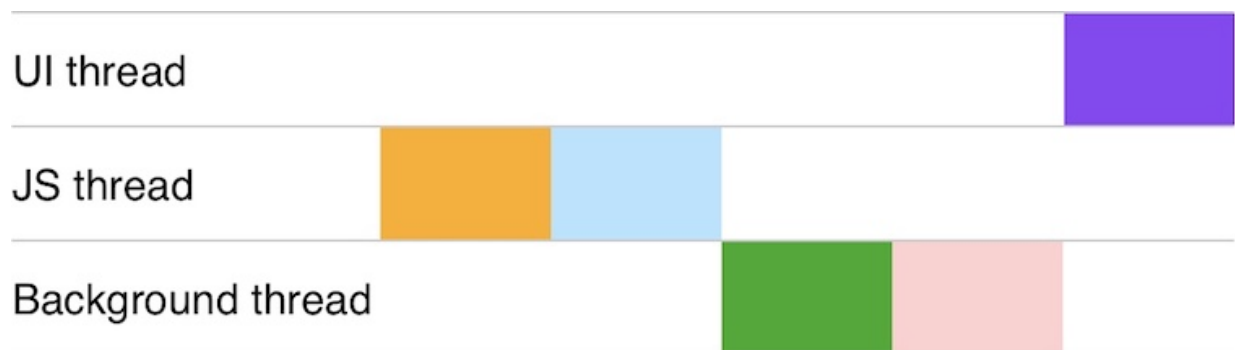
Let’s review the supported scenarios of execution for each phase:



Render Scenarios

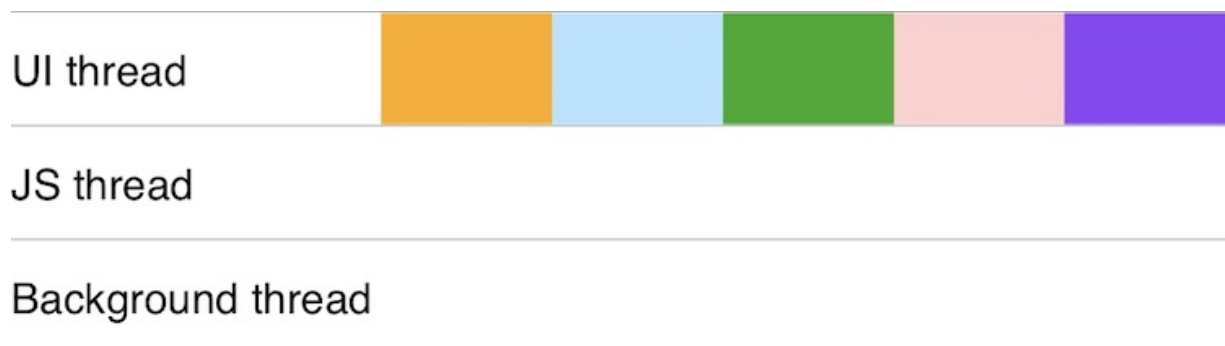
Render in a Background Thread

This is the most common scenario where most of the render pipeline happens on JavaScript and background thread.



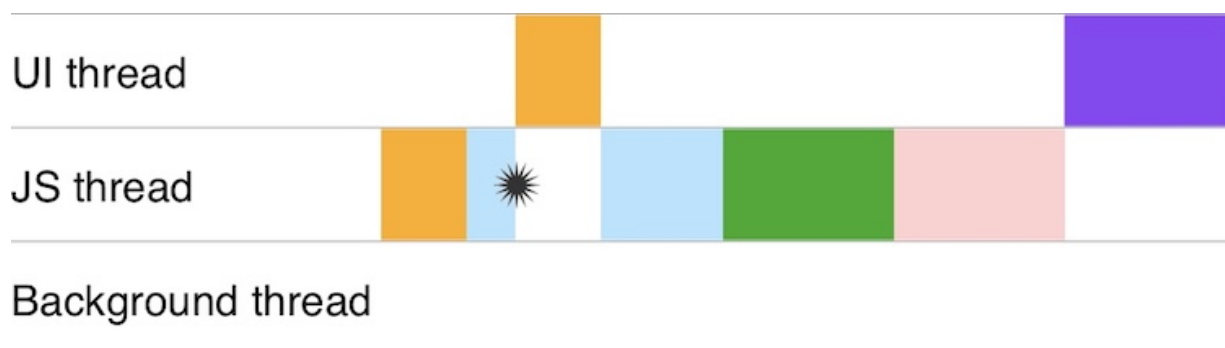
Render in the UI Thread

When there is a high priority event on the UI Thread, the renderer is able to execute all the render pipeline synchronously on the UI thread.



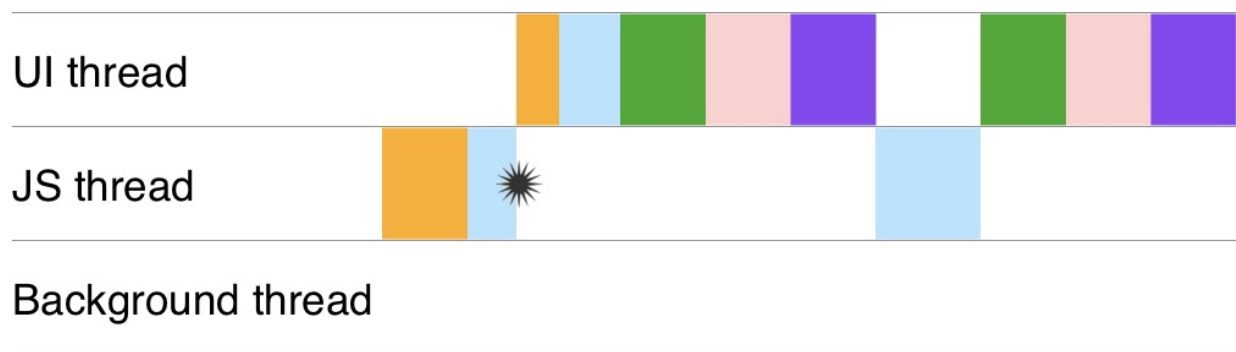
Default or continuous event interruption

This scenario shows the interruption of the render phase by a low priority event in the UI thread. React and the React Native renderer are able to interrupt the render phase and merge its state with a low priority event that is executed on the UI thread. In this case the render process continues executing in the background thread.



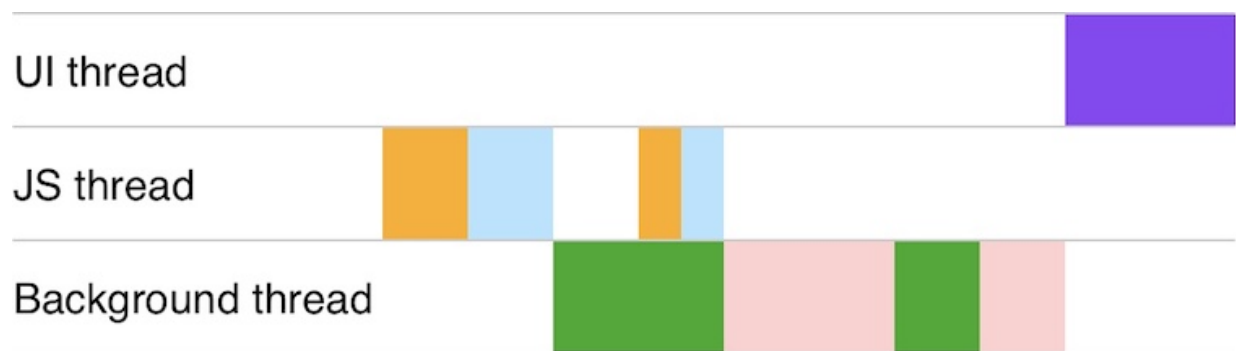
Discrete event interruption

The render phase is interruptible. This scenario shows the interruption of the render phase by a high priority event in the UI thread. React and the renderer are able to interrupt the render phase and merge its state with a high priority event that was executed on the UI thread. The render phase executes synchronously on the UI thread.



Background thread batches updates from JavaScript

Before background thread dispatches update to UI thread, it checks if a newer update hasn't come in from JavaScript. This way, the renderer doesn't render stale state when it knows a newer state is coming in.




C++ State update

Update originating on UI thread and skips rendering phase. See [React Native Renderer State Updates](#) for more details.



Is this page useful?  

 Edit this page

*Last updated on **Apr 22, 2022***