React JS

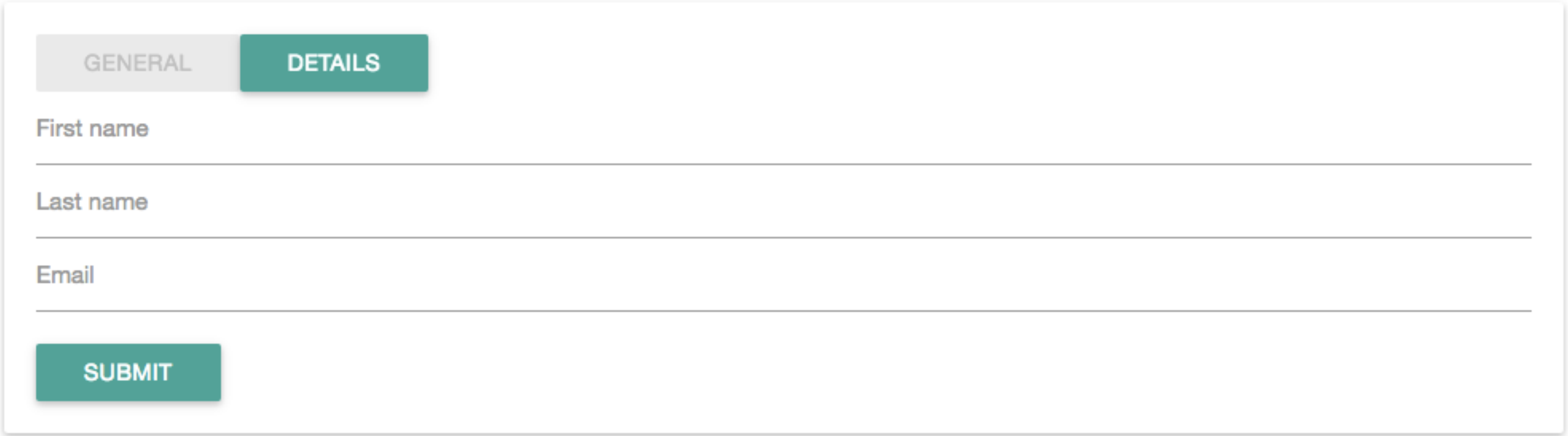
Flux Pattern: Introduction

Saikat Bhattacharya
Technology Lead | DEP

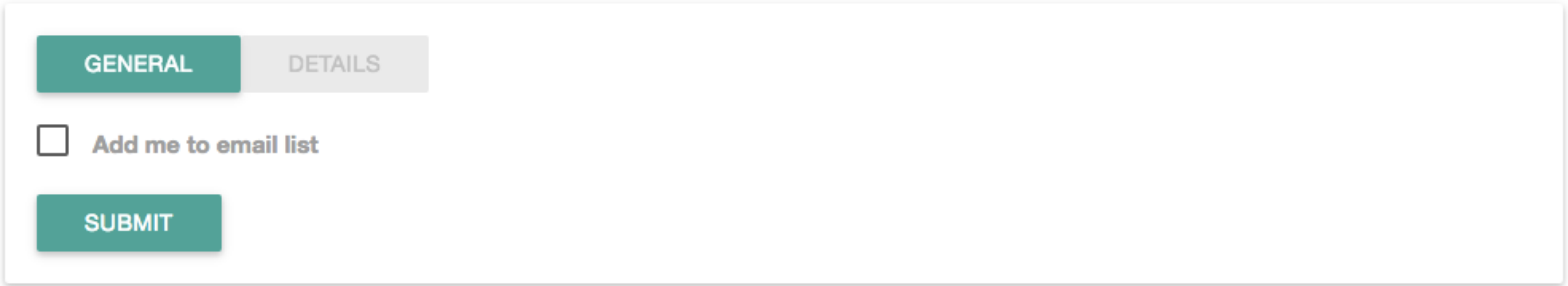
Recap

- React JS is a ~~framework~~ library
- React JS plays mostly the role of 'View' in MVC
- React JS uses JSX, most commonly for rendering
- React JS believes in Virtual DOM
- State and Props: the two objects that carry information
- **`react-router`**: A module for switching between components

Task



Task



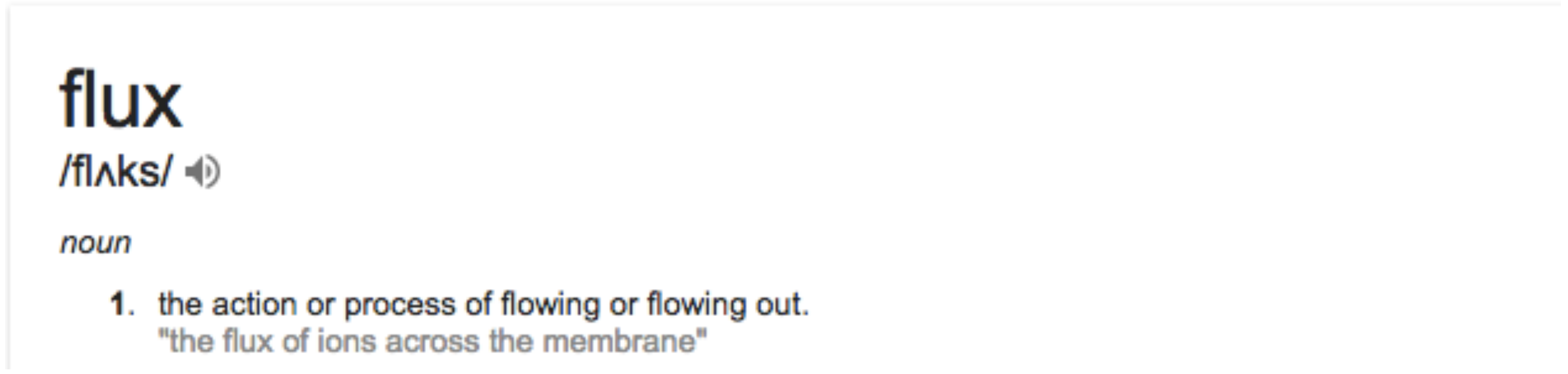
Task

First Name: Saikat
Last Name: Bhattacharya
Email: saikat.bhattacharya5@cognizant.com
optIn: true

Today’s Plan

- What is Flux?
- Difference between Flux and MVC
- Action, Dispatcher, Store and View

What is ‘Flux’?



flux
/flʌks/

noun

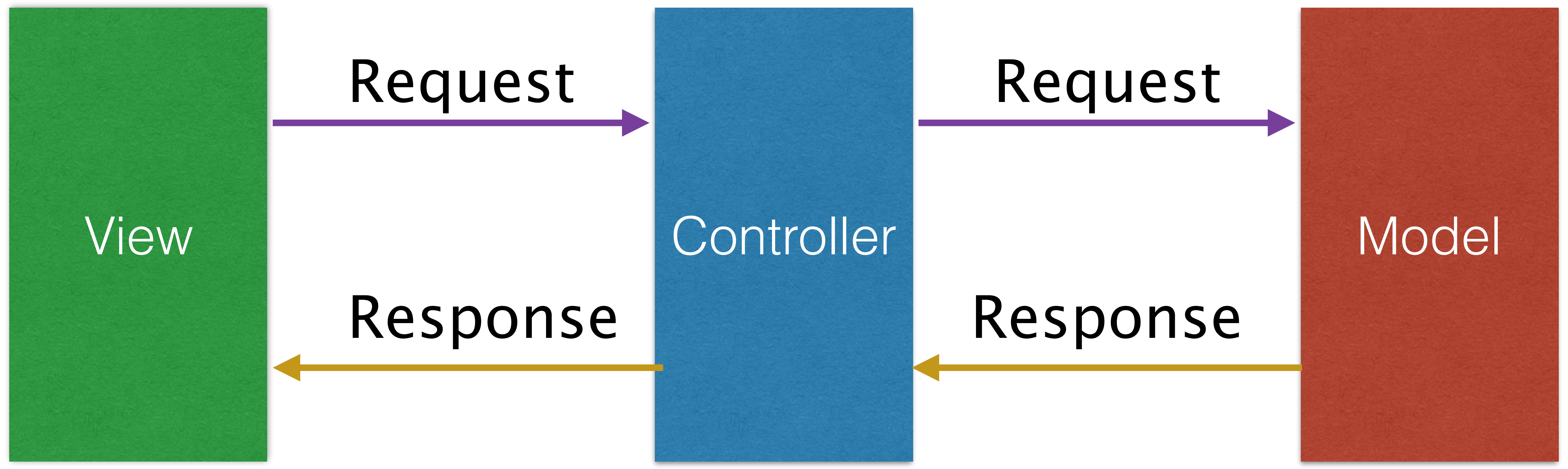
1. the action or process of flowing or flowing out.
   "the flux of ions across the membrane"

This term was introduced into field of science by Isaac Newton

What is ‘Flux’?

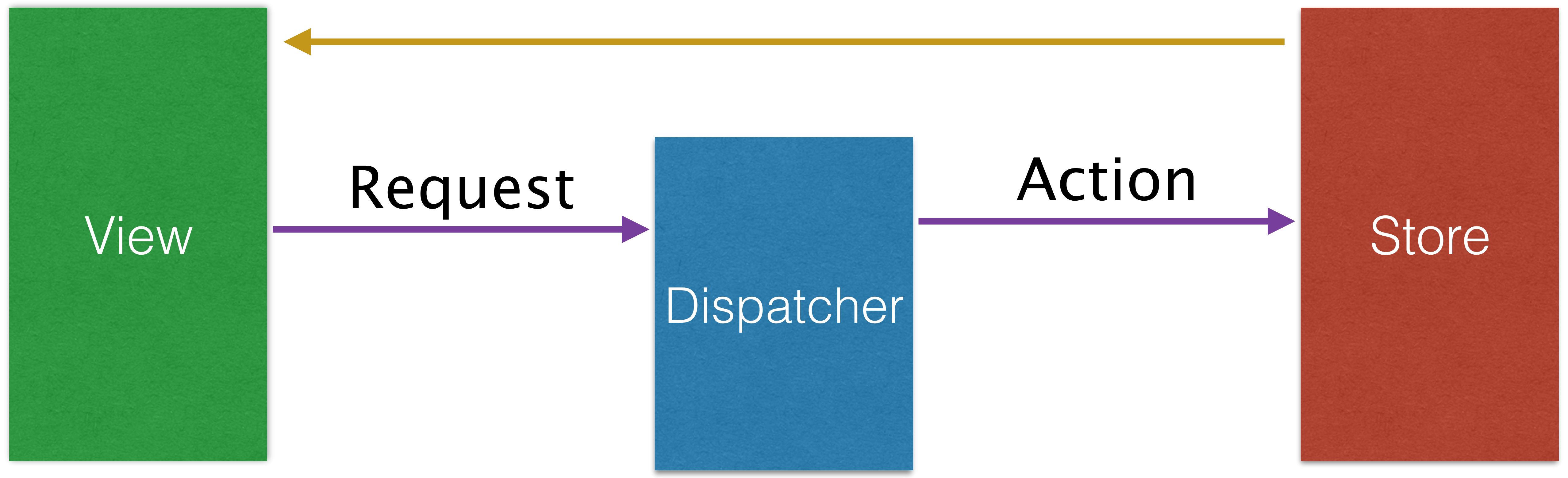
- It's more of a pattern rather than a formal framework
- It complements React's composable view components by utilizing a unidirectional data flow
- Three major parts: the dispatcher, the stores, and the views

MVC vs Flux



Traditional MVC data flow

MVC vs Flux



Flux data flow

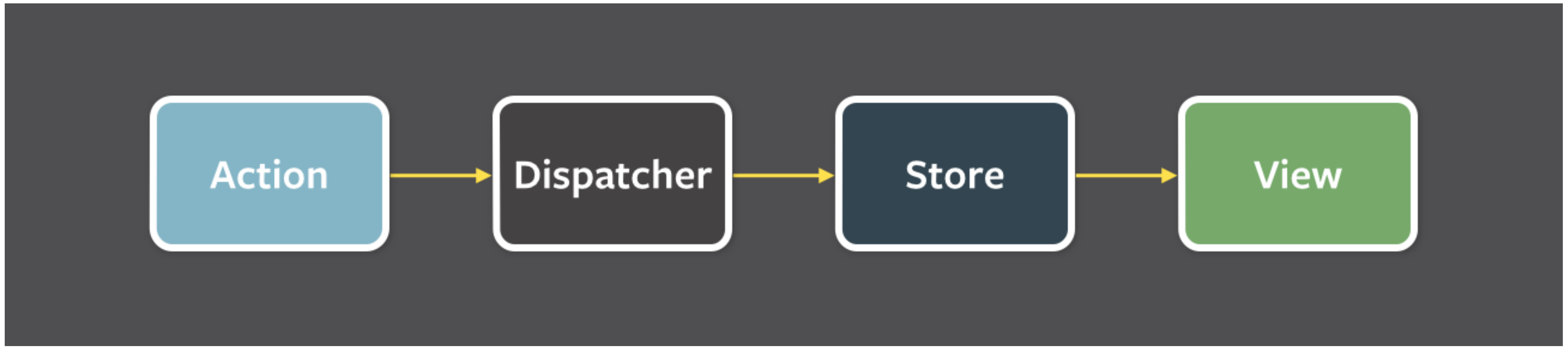
MVC vs Flux

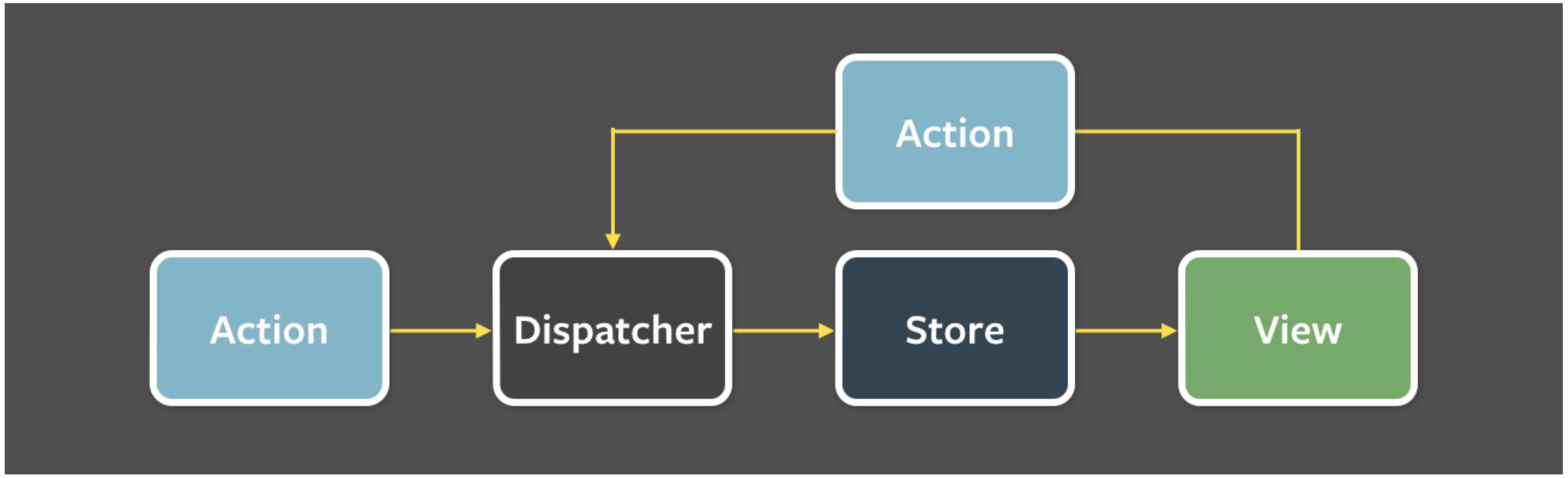
- The Flow of the app is essential to Flux and there are very strict rules that are enforced by the Dispatcher. In MVC the flow isn’t enforced and most MVC patterns implement it differently

- Unidirectional flow: Every change goes through the dispatcher. A store can’t change other stores directly. Same applies for views and other actions. Changes must go through the dispatcher via actions. In MVC it’s very common to have bidirectional flow

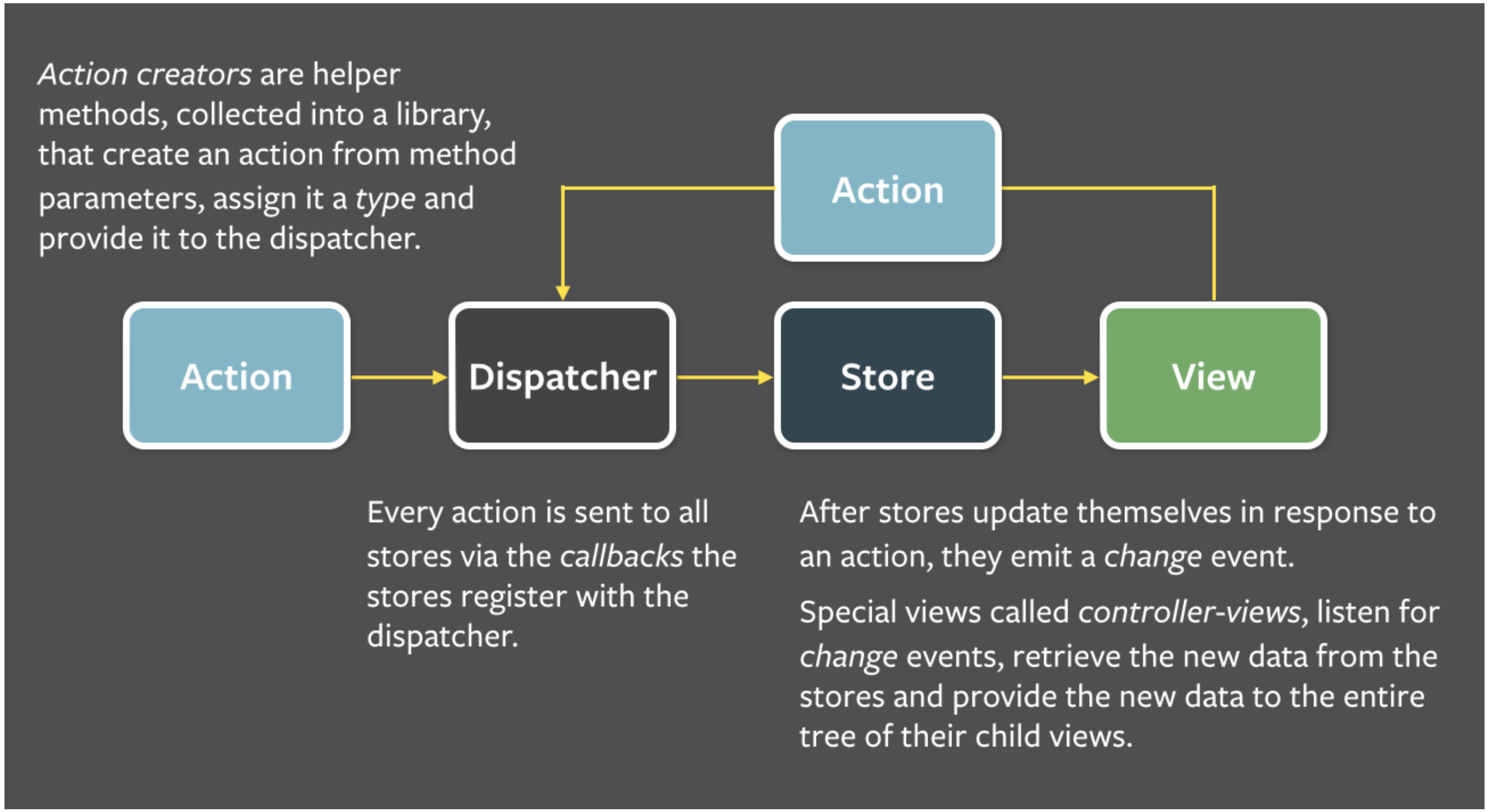
- Stores don’t need to model anything and can store any application related state. In MVC models try to model something, usually single objects

Action, Dispatcher, Store and View





Action, Dispatcher, Store and View



Action creators are helper methods, collected into a library, that create an action from method parameters, assign it a *type* and provide it to the dispatcher.

Every action is sent to all stores via the *callbacks* the stores register with the dispatcher.

After stores update themselves in response to an action, they emit a *change* event.

Special views called *controller-views*, listen for *change* events, retrieve the new data from the stores and provide the new data to the entire tree of their child views.