

What Is Flux?



A pattern

Centralized dispatcher

Unidirectional data flows

They call it **Flux** for a reason too.

Flux Implementations



Facebook's Flux

Alt

Reflux

Flummox

Marty

Fluxxor

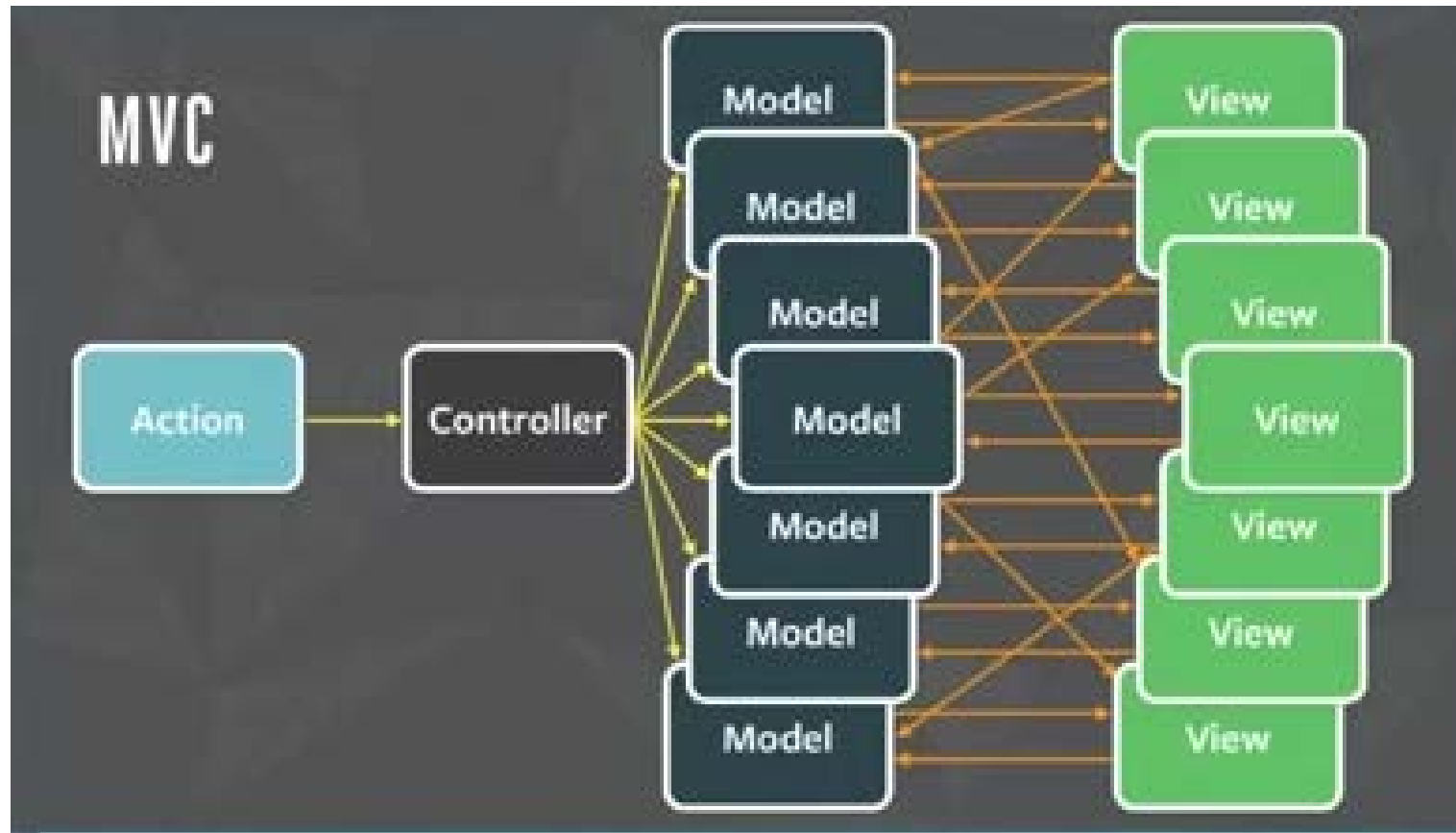
Delorean

Redux

NuclearJS

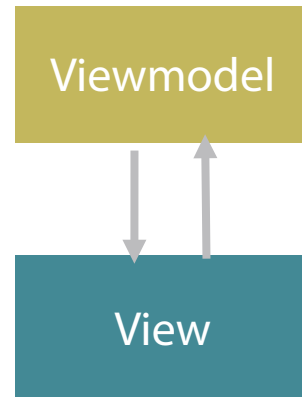
Fluxible

Good Luck Debugging This



Two-way Binding vs. Unidirectional

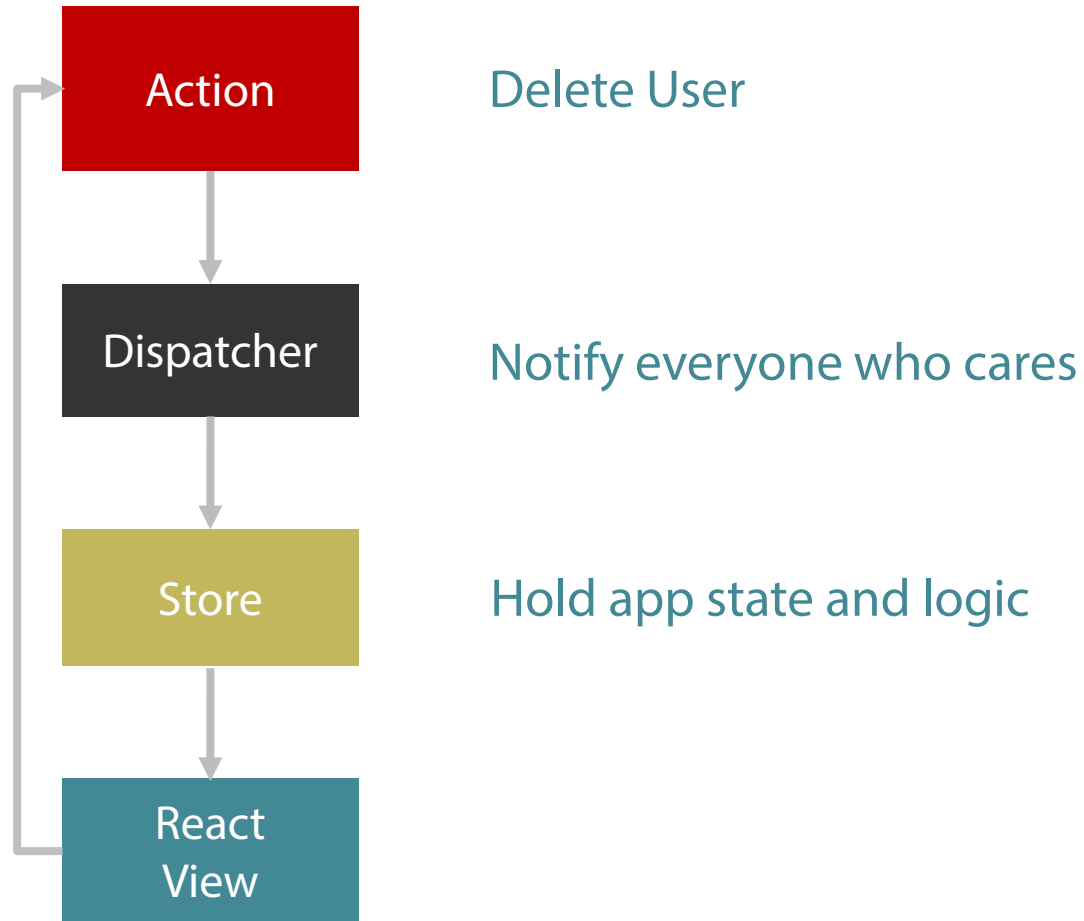
Two-way binding



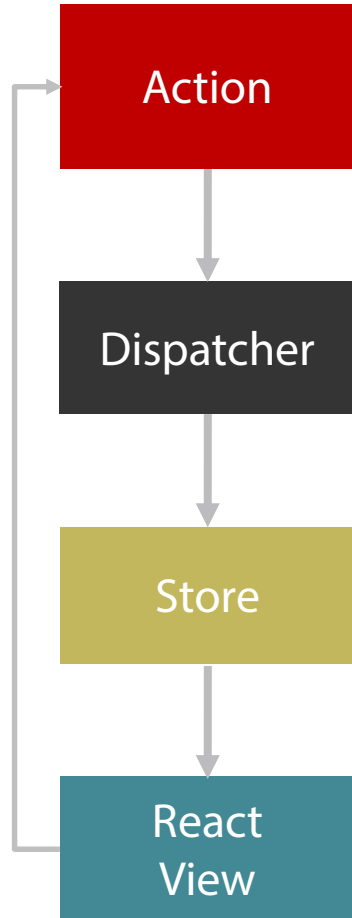
Unidirectional



Flux: 3 Parts



Actions

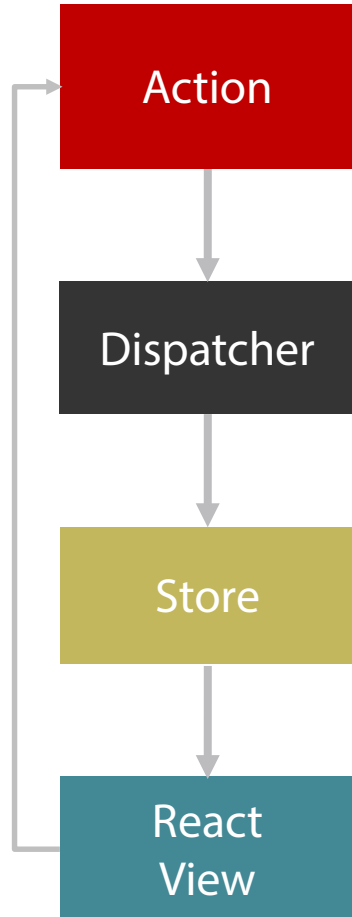


Encapsulate events

Triggered by user interactions and server

Passed to dispatcher

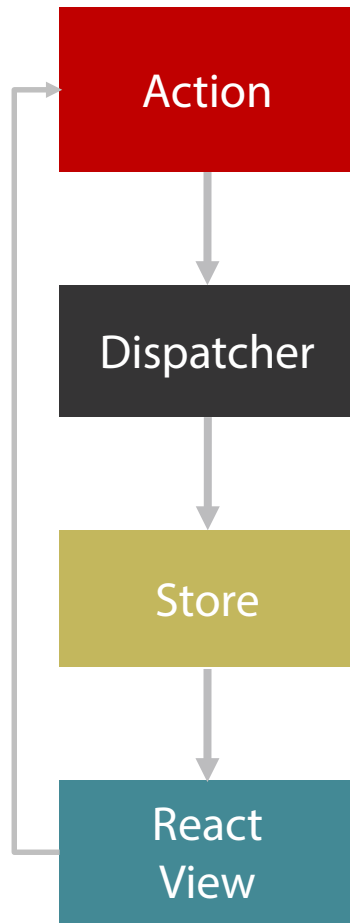
Actions



Payload has type and data

```
{  
  type: USER_SAVED  
  data: {  
    firstName: 'Cory',  
    lastName: 'House';  
  }  
}
```


Dispatcher



Central Hub – There's only one

Holds list of callbacks

Broadcasts payload to registered callbacks

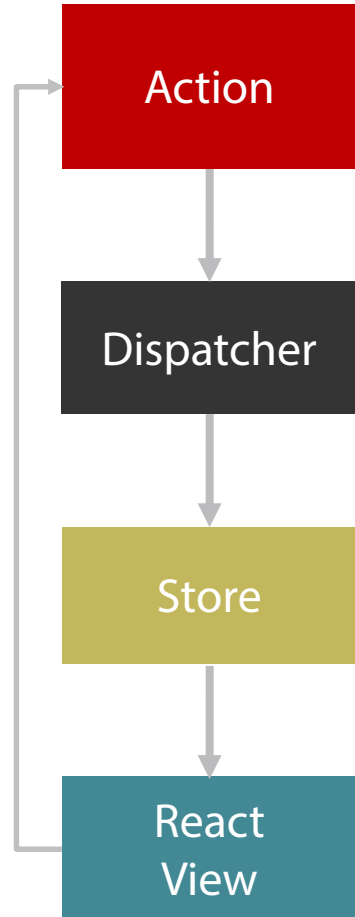
Sends actions to stores

Constants

Keeps things organized

Provides high level view of what the app actually does

Store



Holds app state, logic, data retrieval
Not a model - *Contains* models.

One, or many

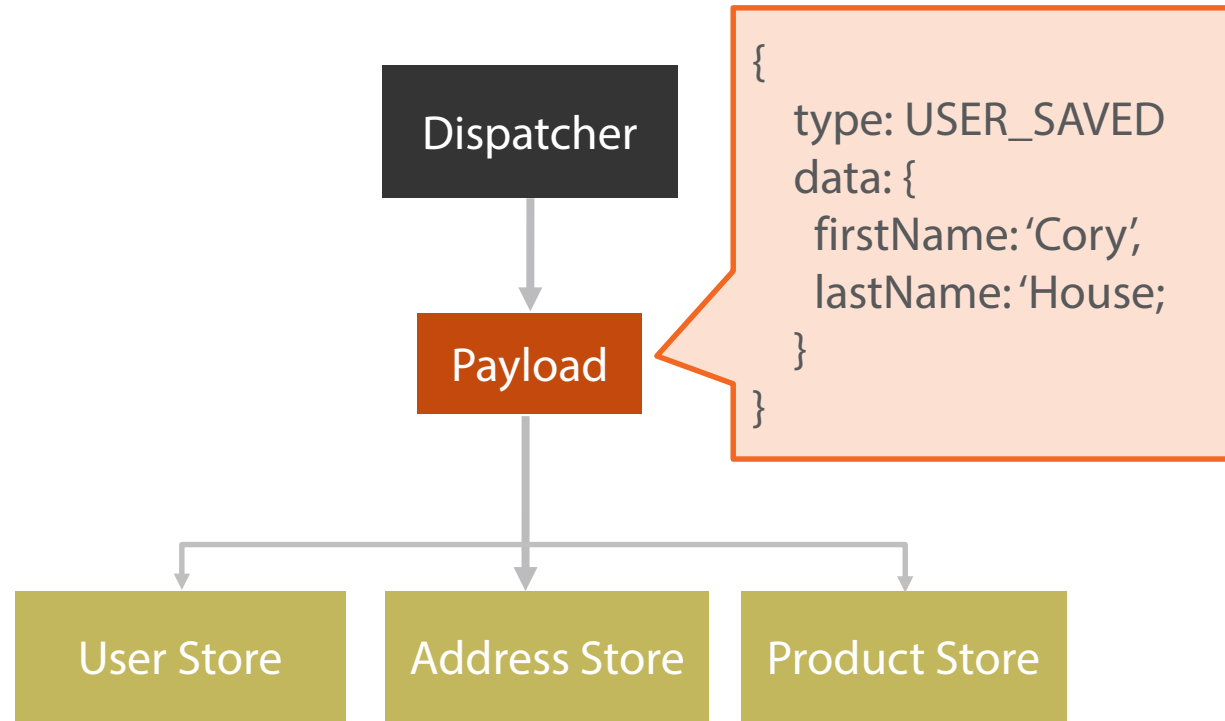
Registers callbacks with dispatcher

Uses Node's EventEmitter

The Structure of a Store

Every store has these common traits (aka interface)

1. Extend EventEmitter
2. addChangeListener and removeChangeListener
3. emitChange



As an application grows, the dispatcher becomes more vital, as it can be used to **manage dependencies between the stores** by invoking the registered callbacks in a specific order. Stores can declaratively wait for other stores to finish updating, and then update themselves accordingly.

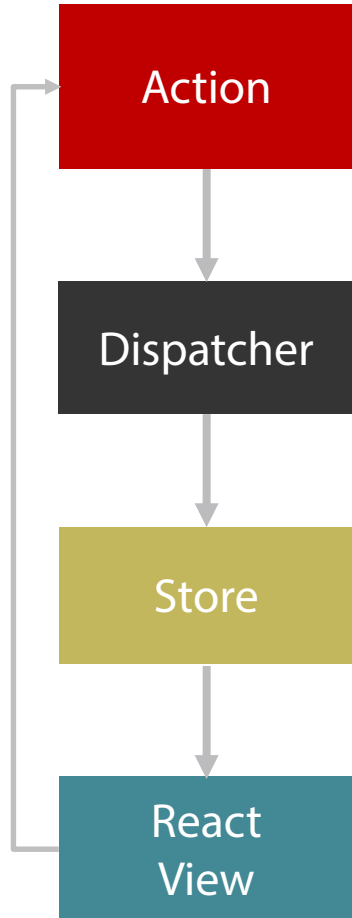
— Flux Documentation

I usually view ActionCreators as something that is used for writes to the server. For example, the user added a new message by clicking the Send button, so the responsible view calls into ActionCreator.

On the other hand, if a store needs to fetch data from the server, then that doesn't have to use an ActionCreator. Instead, the store can directly hit an endpoint to load its data, and then direct the response through the dispatcher.

— Jing Chen, Facebook

Controller Views



Top level component

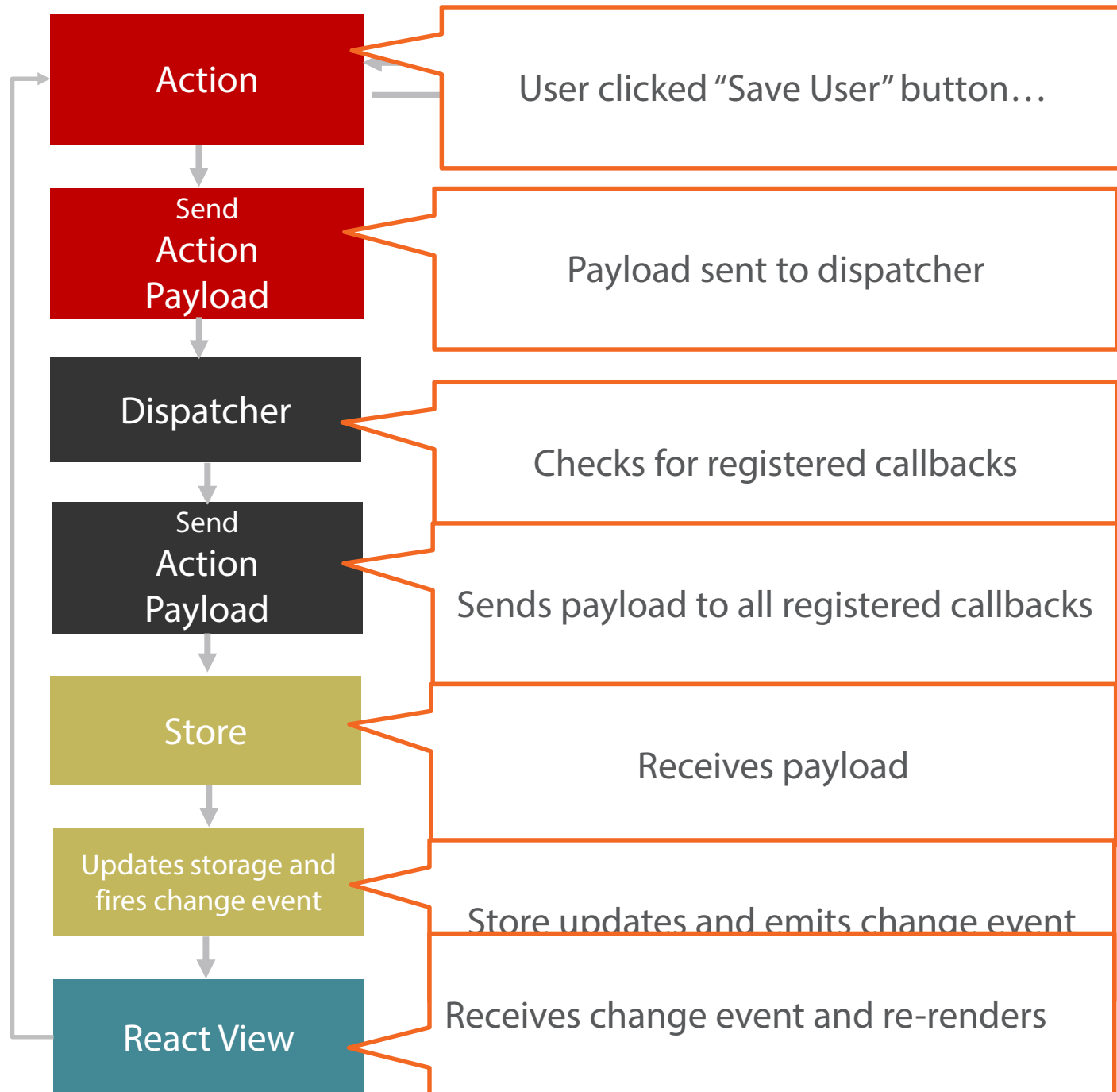
Interacts with Stores

Holds data in state

Sends data to children as props

Payload:

```
{  
  type: USER_SAVED  
  data: {  
    firstName: 'Cory',  
    lastName: 'House';  
  }  
}
```



A Chat with Flux

- React** Hey CourseAction, someone clicked this “Save Course” button.
- Action** Thanks React! I registered an action creator with the dispatcher, so the dispatcher should take care of notifying all the stores that care.
- Dispatcher** Let me see who cares about a course being saved. Ah! Looks like the CourseStore has registered a callback with me, so I’ll let her know.
- Store** Hi dispatcher! Thanks for the update! I’ll update my data with the payload you sent. Then I’ll emit an event for the React components that care.
- React** Ooo! Shiny new data from the store! I’ll update the UI to reflect this!

Flux API

`register(function callback)` – “Hey dispatcher, run me when actions happen. - Store”

`unregister(string id)` – “Hey dispatcher, stop worrying about this action. - Store”

`waitFor(array<string> ids)` – “Update this store first. – Store”

`dispatch(object payload)` - “Hey dispatcher, tell the stores about this action. - Action”

`isDispatching()` – “I’m busy dispatching callbacks right now.”

So Flux Is a Publish-Subscribe Model?

Not quite.

Differs in two ways:

1. Every payload is dispatched to all registered callbacks.
2. Callbacks can wait for other callbacks

Summary

Flux is a pattern for unidirectional data flows

Actions encapsulate events

Dispatcher is a central hub that holds callbacks

Stores hold app state

Many implementations

