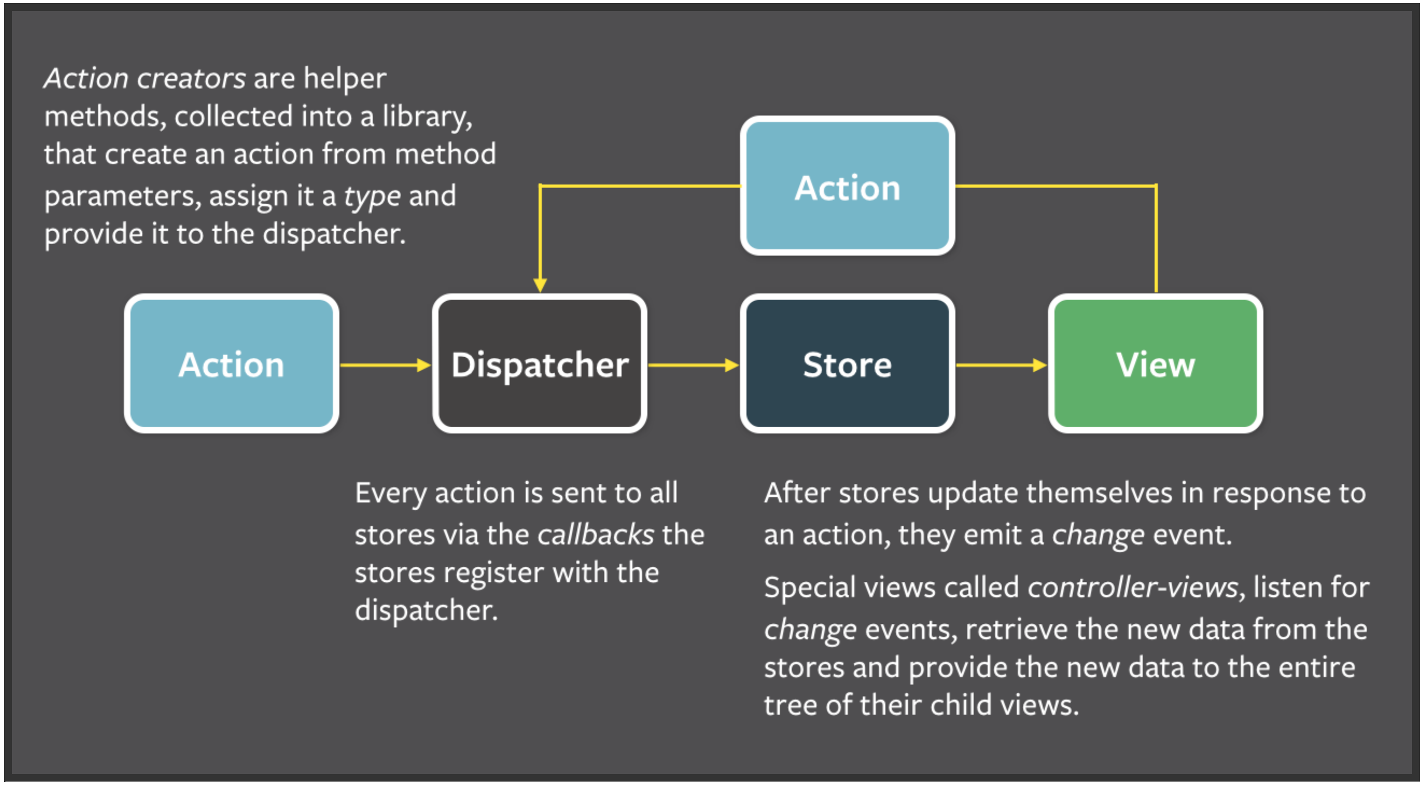
WELCOME TO DAY 5 OF REACTJS BOOTCAMP

AGENDA

- Introduction to Flux
- Actions, Dispatchers, Stores
- Overall Result of Flux Architecture

FLUX



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.

Flux is really just a fancy term for pub/sub architecture, i.e. data always flows one way through the application and it is picked up along the way by various subscribers (stores) who are listening to it.

Flux eschews MVC in favor of unidirectional data flow. What this means is that data enters through a single place (your actions) and then flow outward through their state manager (the store) and finally onto the view. The view can then restart the flow by calling other actions in response to user input.

WHY FLUX

- State is messy
- Unidirectional data makes for easy debugging
- Composable components favor reuse
- Stores as a single domain also simplifies debugging

DISPATCHER

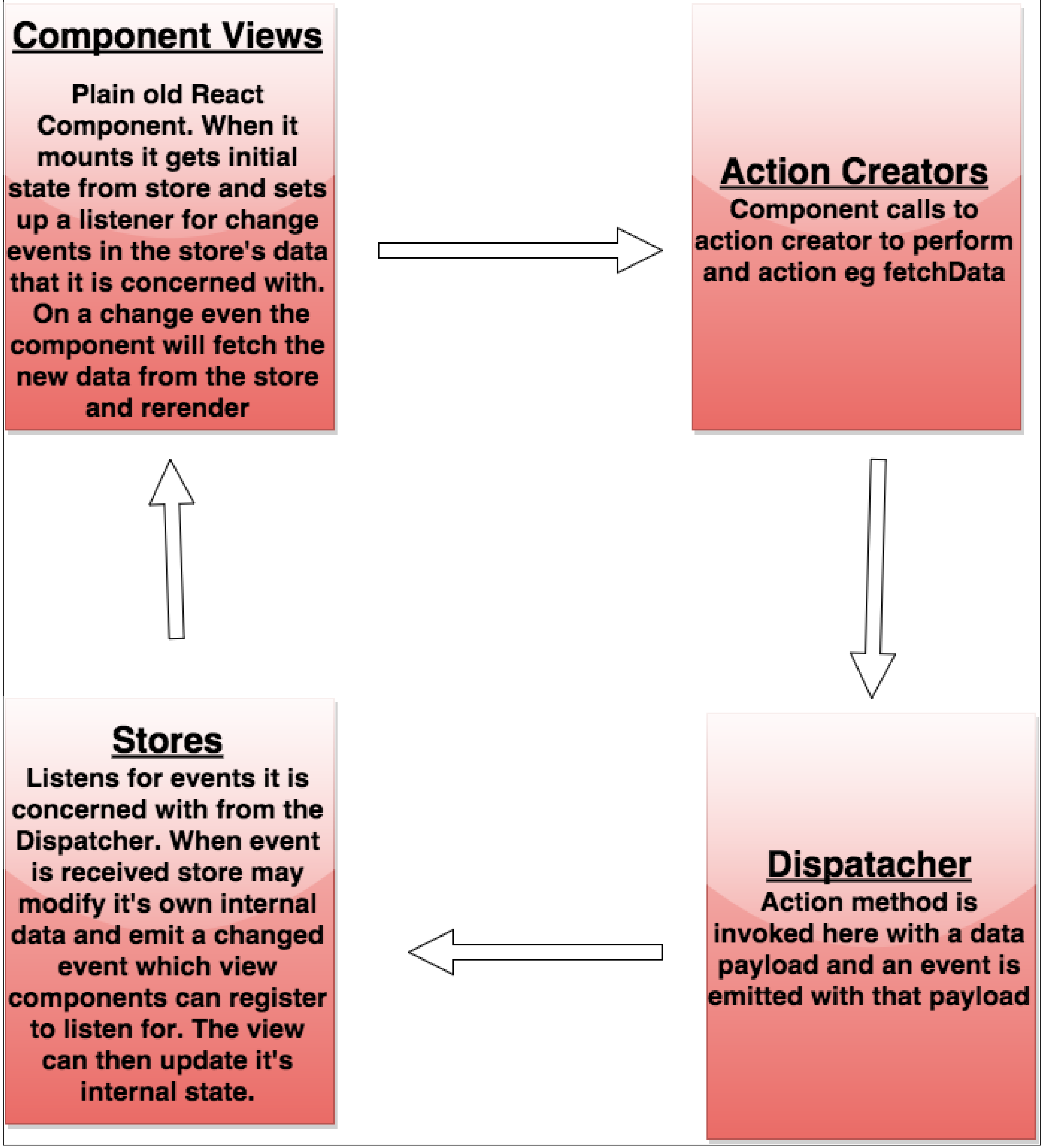
- *SINGLE* messaging hub in the application
- Registry of callbacks
- Has no logic
- Dispatcher recieves actions and fires corresponding callback
- Can manage dependencies between stores

STORES

- Contain the domain logic
- More than just a model
- All stores get the callback from the dispatcher and handle in a case statement
- Changes made through dispatcher -> Store's case statement on action type -> Data update -> Fire change event
- Views can query the store but they are treated as immutable from view's perspective

ACTIONS

- Views can start the communication intents via actions
- Actions are sent through the dispatcher and sent out to the stores
- We have had luck with past tense but it doesn't matter as long as you are consistent
- Try to keep actions generic where possible



Component Views

Plain old React Component. When it mounts it gets initial state from store and sets up a listener for change events in the store's data that it is concerned with. On a change even the component will fetch the new data from the store and rerender

Action Creators

Component calls to action creator to perform and action eg fetchData

Dispatacher

Action method is invoked here with a data payload and an event is emitted with that payload

Stores

Listens for events it is concerned with from the Dispatcher. When event is received store may modify it's own internal data and emit a changed event which view components can register to listen for. The view can then update it's internal state.