Swenug Linköping 2015-MAR-25

React + Flux

Kristofer Palmvik

Kristofer Palmvik

JS

BACKBONE.JS

Knockout.

jQuery
write less, do more.

ANGULARJS
by Google

Mithril

UNDERSCORE.JS

ember

“Studies show that a todo list is the most complex JavaScript app you can build before a newer, better framework is invented.”

–Allen Pike

http://www.allenpike.com/2015/javascript-framework-fatigue/

React

A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES

Decrease coupling
Increase cohesion

Most frameworks separate logic and presentation

How do we keep all views synchronized and updated?

It worked in 1998!

Let’s render everything from scratch.
Every time!

That didn’t work :(

Because state will be lost and browser DOM manipulation is slow

“The Document Object Model (DOM) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML, and XML documents. The nodes of every document are organized in a tree structure, called the DOM tree. Objects in the DOM tree may be addressed and manipulated by using methods on the objects.”

– http://en.wikipedia.org/wiki/Document_Object_Model

JavaScript is fast.
Browser DOM manipulation is slow.

- Create a virtual DOM in JS
- Update virtual DOM on changes
- Compute minimum change set
- Batch execute the DOM manipulation (in a clever way)

Well, that’s React

!important

Virtual DOM $\neq$ Shadow DOM

React =

Virtual DOM

+

Diff algorithm

+

React components (your code)

Idempotent function generating virtual DOM objects from current state

Reactive programming

Think: Excel

My first React Component

Code examples

Just add react.js (600kB) or react.min.js (120 kB)

Props and state

Used by render()

Prop(ertie)s

```
React.createElement(myComponent,
     { myValue: 'Special Value' });
```

Set by the owner, cannot be modified inside the component

State

User input + other events

Only modified by the component

Avoid state, or keep it as simple as possible!

JSX

A fancy way of writing HTML-ish code in JavaScript

```
React.render(
 <HelloMessage name="John" />, mountNode);

React.render(
  React.createElement(
   HelloMessage, {name: "John"}), mountNode);
```

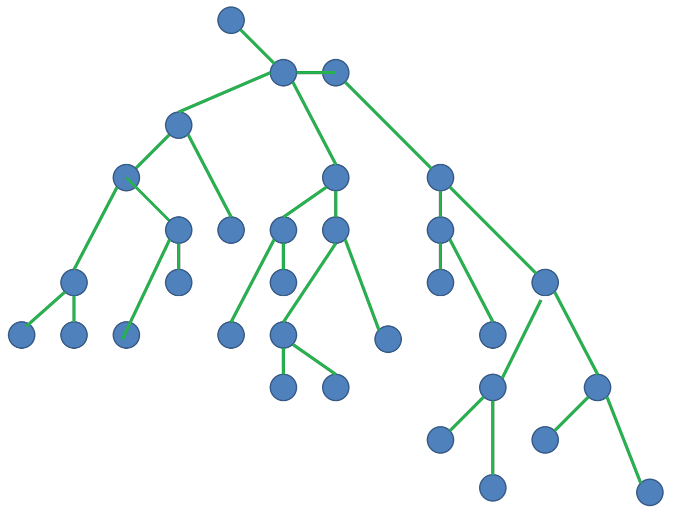
```
return (
      <div>
        <h3>TODO</h3>
        <TodoList items={this.state.items} />
        <form onSubmit={this.handleSubmit}>
          <input onChange={this.onChange} value={this.state.text} />
          <button>{'Add #' + (this.state.items.length + 1)}</button>
        </form>
      </div>
    );

return (
 React.createElement("div", null,
  React.createElement("h3", null, "TODO"),
  React.createElement(TodoList, {items: this.state.items}),
  React.createElement("form", {onSubmit: this.handleSubmit},
   React.createElement("input", {onChange: this.onChange, value:
this.state.text}),
   React.createElement("button", null, 'Add #' + (this.state.items.length + 1))
  )
 )
);
```

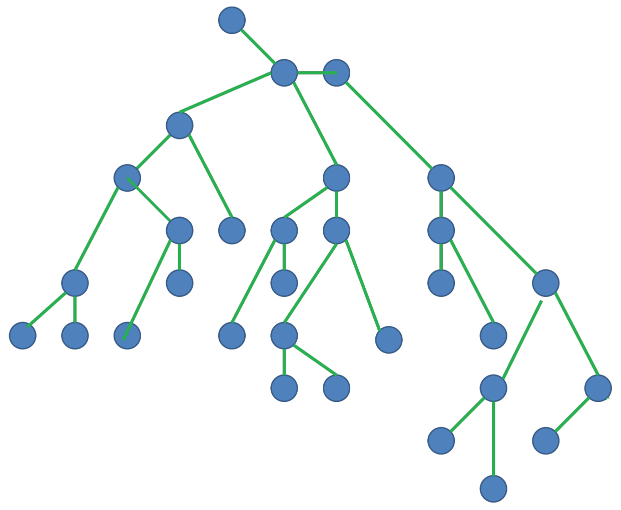
Will it render?

Render when a component has changed its state

Use a similar strategy as Doom 3

But slightly less fun to play





Differences between two trees: $O(n^3)$

React is cheating to make it in O(n)

Assumptions

- Two components of the same class will generate similar trees.
- Two components of different class will generate different trees.
- Every element have unique ID (key) that is stable between render() calls

Diff between two nodes

1. Different type or different components:
   - throw away
2. DOM-node (like <div>):
   - Pairwise match their properties in O(n)
3. Components of same type:
   - Give props to the first and render
   - Diff the resulting subtree.

Diff between two lists (naive)

List

```
<li>Bytecode</li>
<li>Changelog</li>
```

List’

```
<li>Array</li>
<li>Bytecode</li>
<li>Changelog</li>
```

Diff between two lists (with keys)

List

```
<li key=”B”>Bytecode</li>
<li key=”C”>Changelog</li>
```

List’

```
<li key=”A”>Array</li>
<li key=”B”>Bytecode</li>
<li key=”C”>Changelog</li>
```

Batch execute the changes to browser DOM

Read first, write later to avoid layout thrashing

“It is important to remember that the reconciliation algorithm is an implementation detail.”

– https://facebook.github.io/react/docs/reconciliation.html

“React is an amazing abstraction, but very few abstractions come without at least some overhead. In the case of Atom’s text editor, it’s worth the effort to avoid this overhead by hand coding all DOM updates.”

– https://github.com/atom/atom/pull/5624

Great things about React

- Everything is JavaScript
  - Write code and test it as usual
- Everything is in the Virtual DOM
  - Test without a browser/Selenium/PhantomJS…
  - Server side rendering
- Everything is in one place
  - Logic and presentation together in the component
- Everything is escaped
  - XSS protection by default

Code example

// Todo: Write React Todo app

You can still use jQuery*

Use lifecycle hooks like componentDidMount()

* But you don’t want to

OK, so what is Flux?

“Flux is more of a pattern than a full-blown framework, and you can start using it without a lot of new code beyond React.”

– https://facebook.github.io/react/blog/2014/07/30/flux-actions-and-the-dispatcher.html

“Not a framework, but a set of guiding principles for building scalable, maintainable applications with React.”

– https://medium.com/@garychambers108/understanding-flux-f93e9f650af7

“Flux is, on simple terms, a glorified pub/sub architecture.”

– http://ryanclark.me/getting-started-with-flux

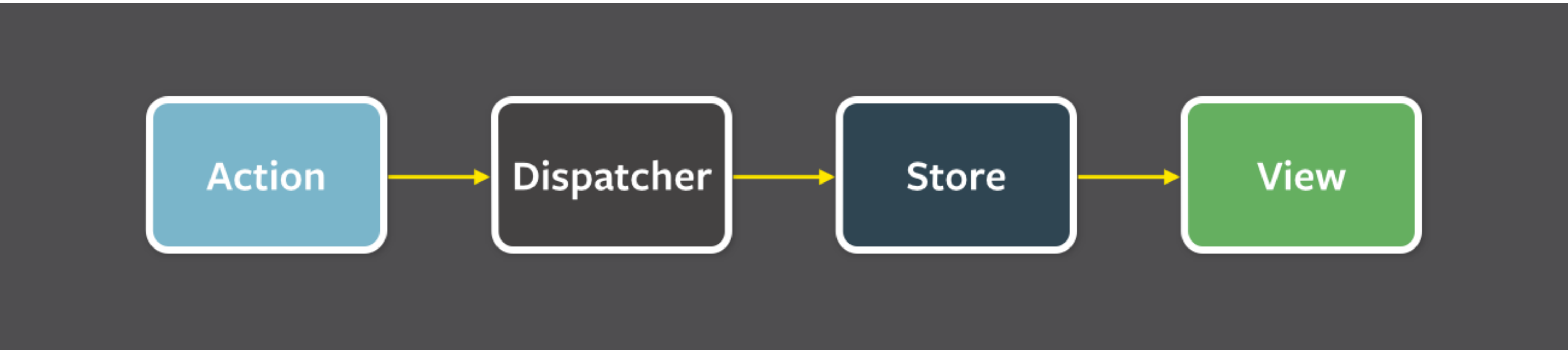
Flux helps you with the data flow

It is not a framework, the only thing you get is the Dispatcher

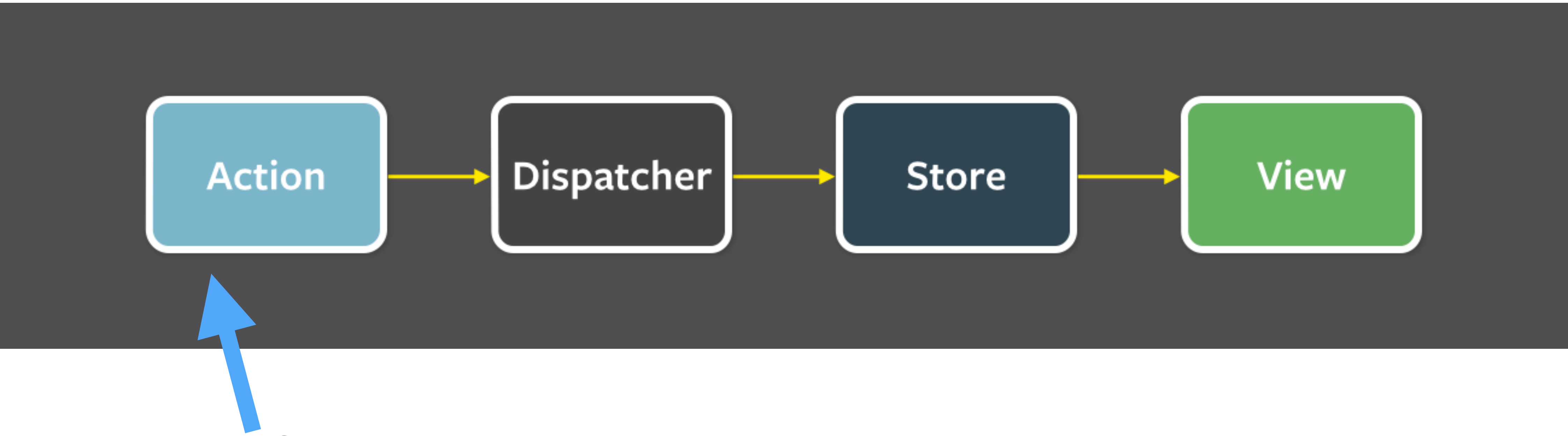
Replace MVC with Flux

And combine it with React if you like

It’s all about the data flow

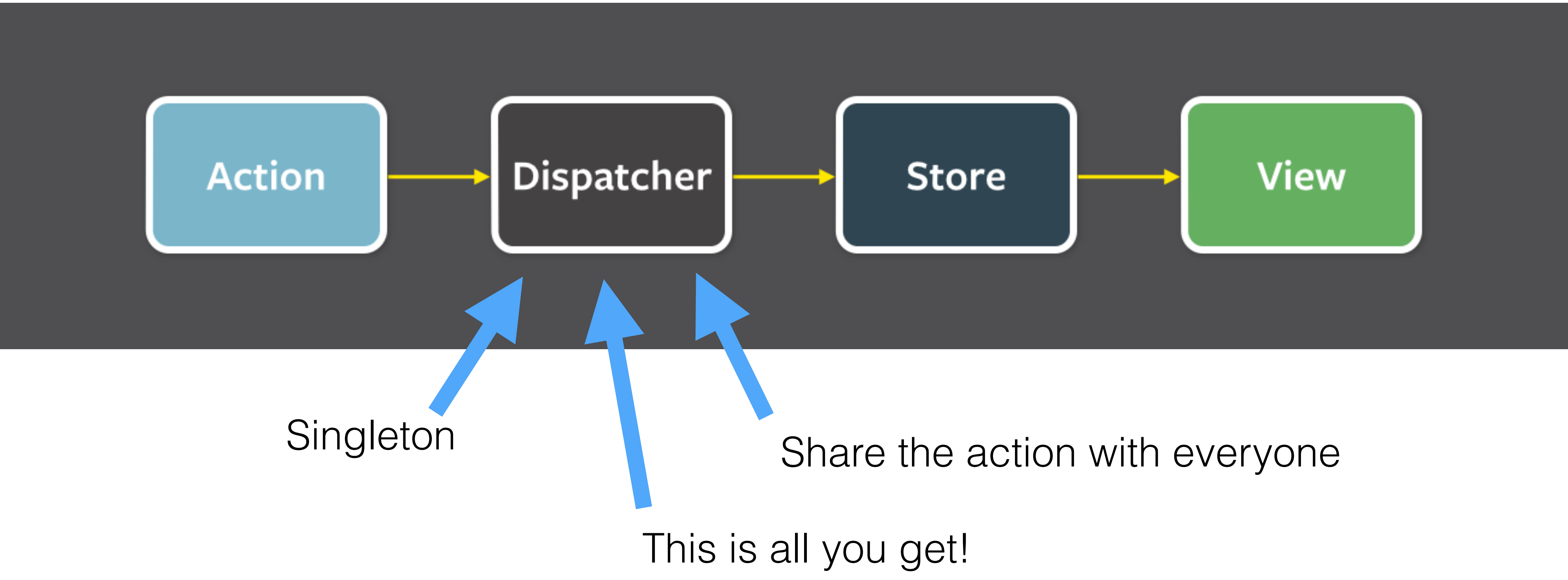


It’s all about the data flow

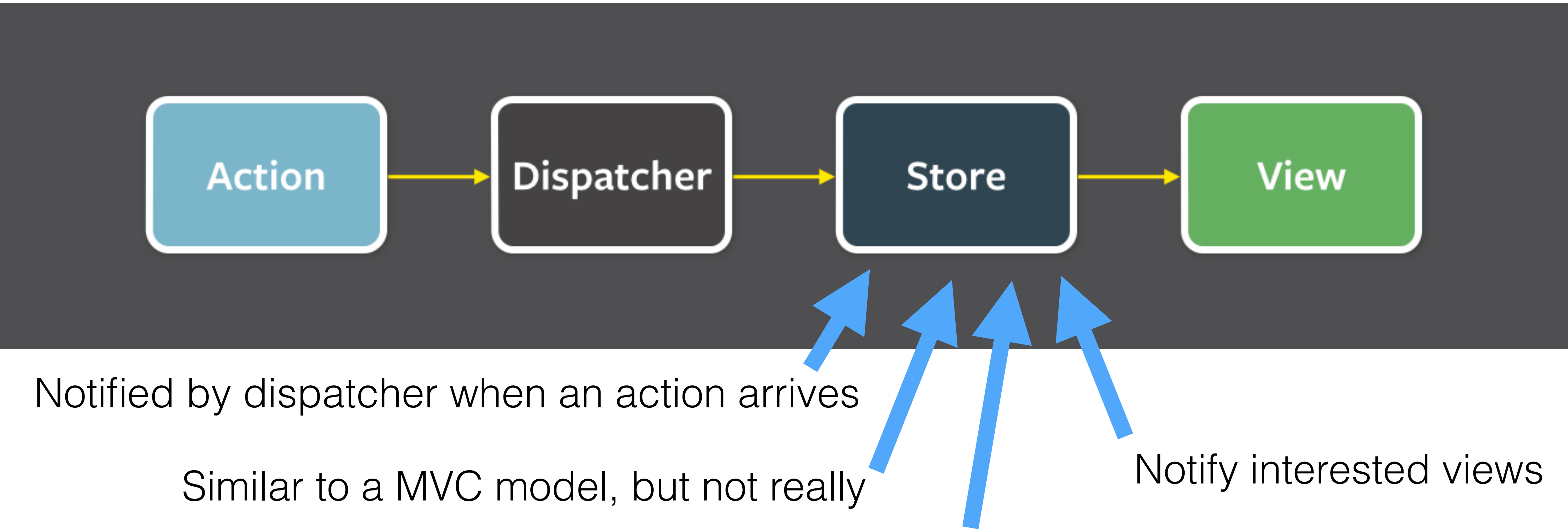


Object created when something happens
in the application, like a click

It’s all about the data flow



It's all about the data flow



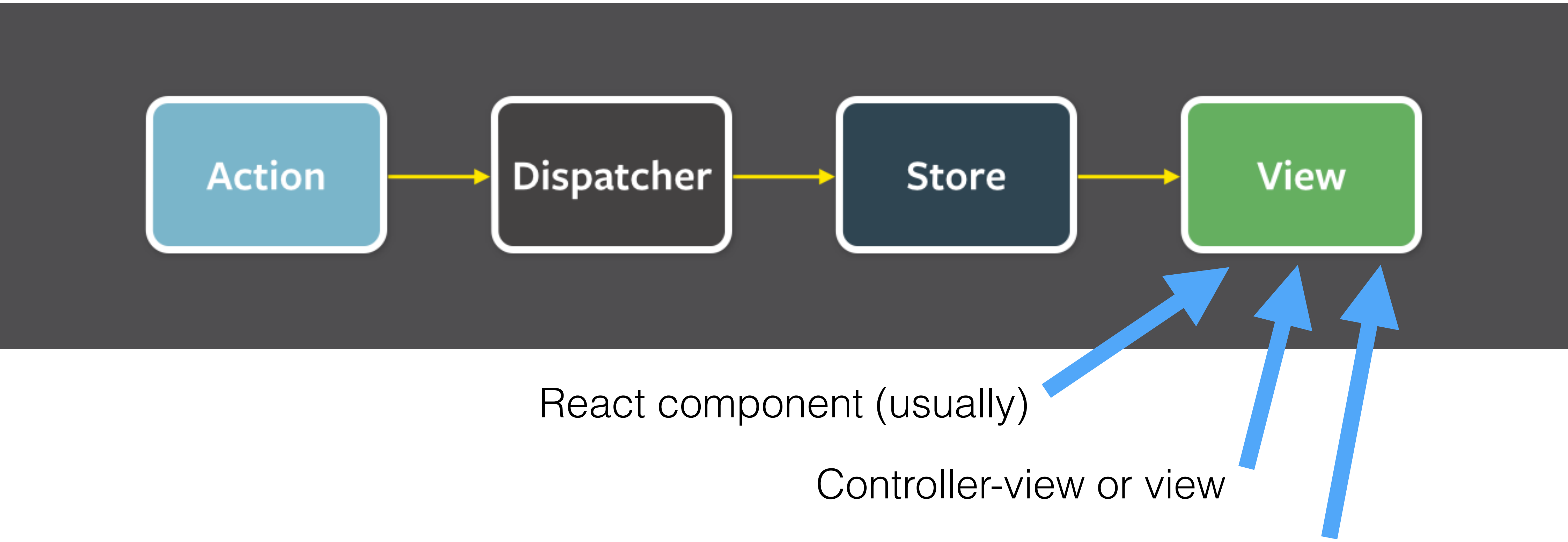
Notified by dispatcher when an action arrives

Similar to a MVC model, but not really

Notify interested views

Reactive state change

It’s all about the data flow

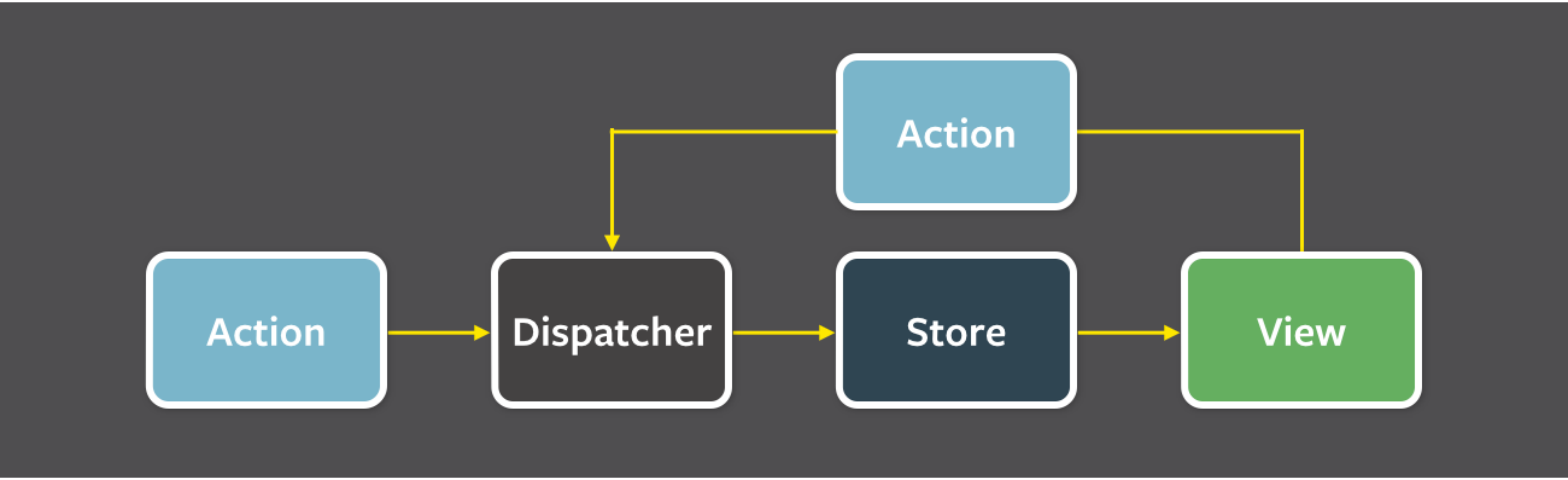


React component (usually)

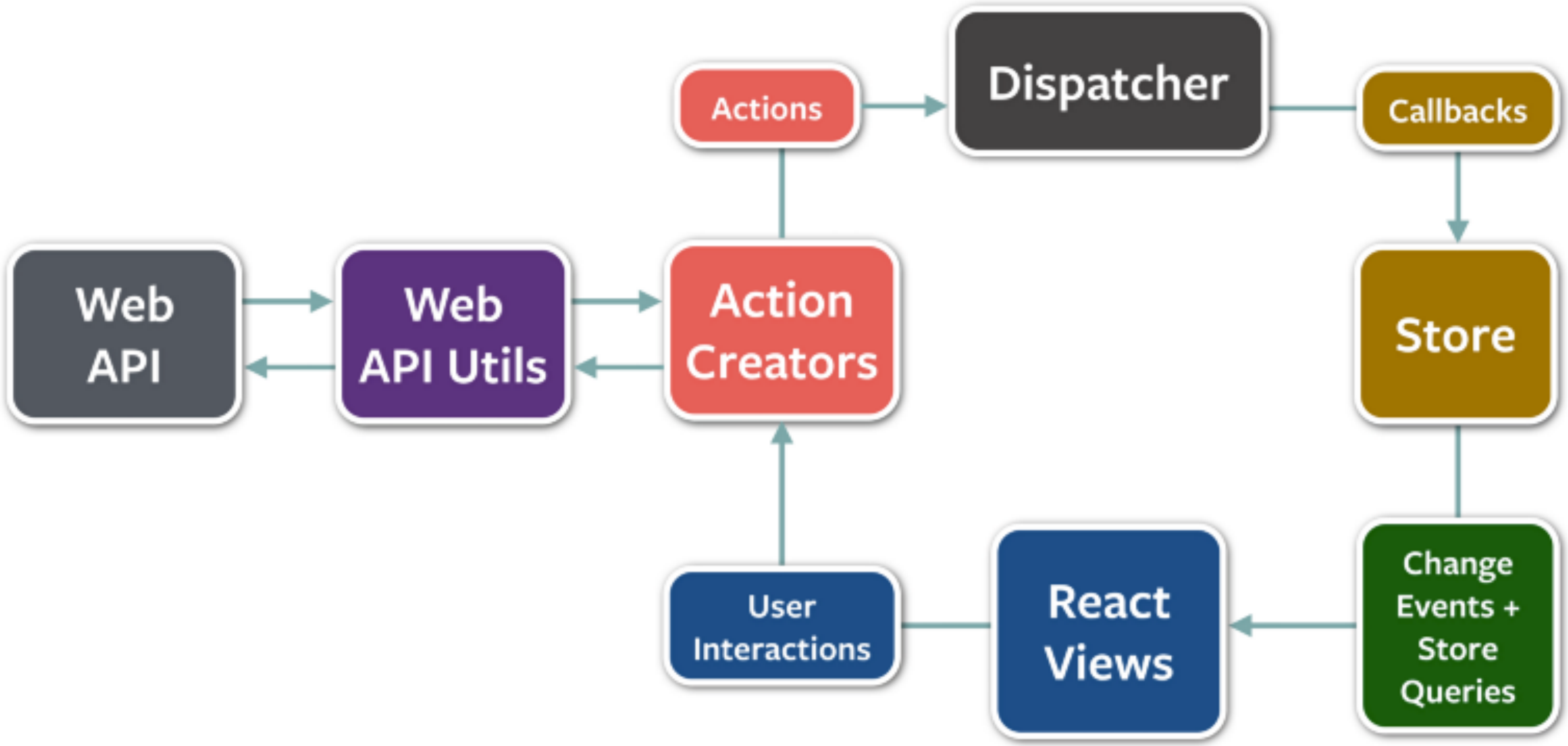
Controller-view or view

Copy state from store

A little less conversation



Round and round…



Easy to reason about

- All change is initiated by an action
- All actions are going through the dispatcher
- All state in the store(s)

No loops, please!

The dispatcher is looking out for circular dependencies

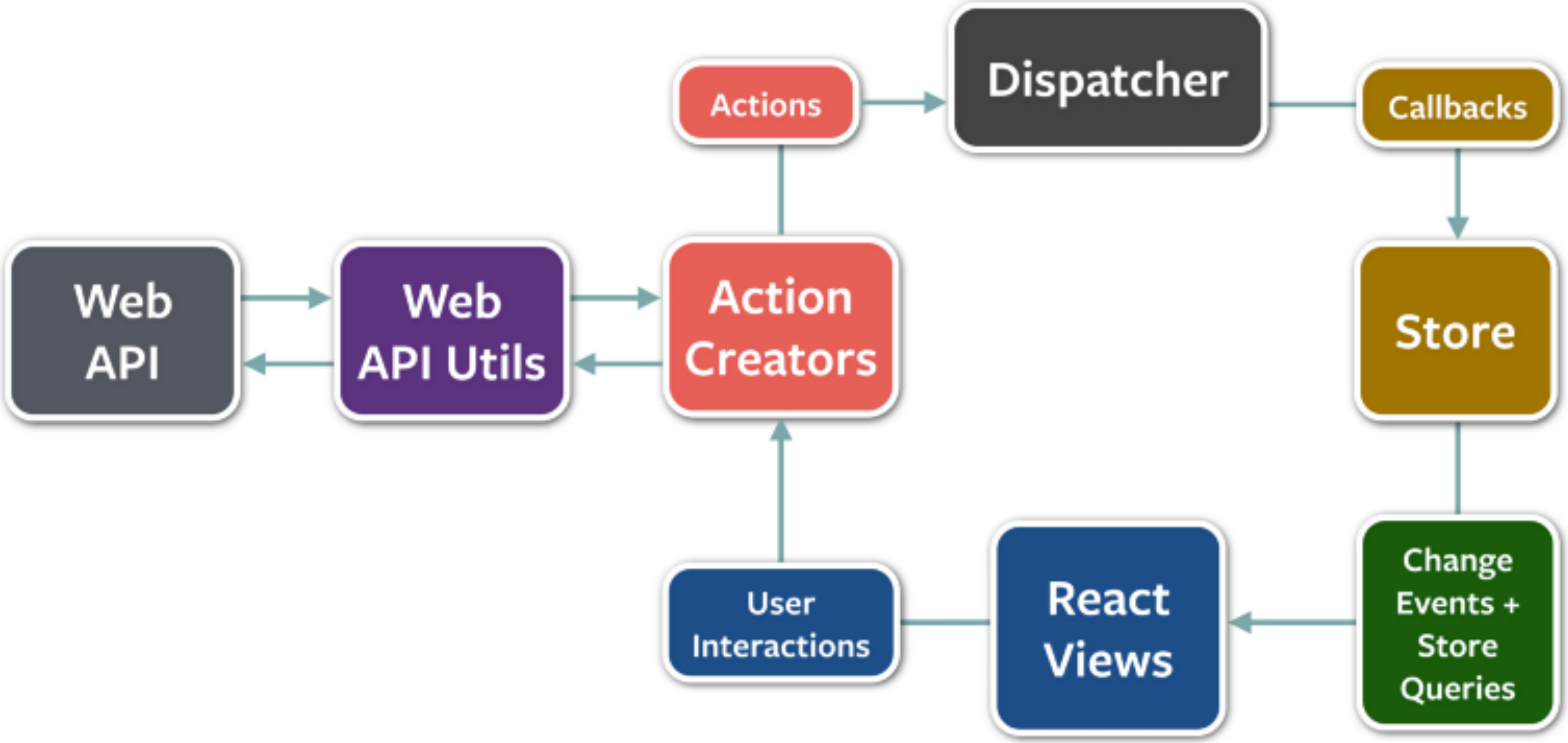
Code example

So much todo, so little done.

My experience so far…

Don’t think you get it all

Where do we fetch data?



Where do you store data?

Less dependencies between stores. More controller-views.

How does it scale?

Easy to understand, fun, and great performance

How do we actually do it?

Facebook usually knows best…

Add-ons

…routing, css, canvas, animations, immutability, testing, functional programming…

Reflux

If you like Functional Reactive programming
and want to avoid comparing strings

Reflux

```
// Creating action
var toggleGem = Reflux.createAction();

// Listening to action
var isGemActivated = false;

toggleGem.listen(function() {
  isGemActivated = !isGemActivated;
  var strActivated = isGemActivated ?
    'activated' :
    'deactivated';
  console.log('Gem is ' + strActivated);
});
```

Relay and GraphQL

Declare your data dependency and let magic handle the rest

React Native

Use native controls instead of a web browser

Learn once. Write anywhere.

TL;DL

Code examples

https://github.com/kpalmvik/react-flux-swenug

kristofer palmvik.

kristofer.palmvik.se kristofer@palmvik.se

@kpalmvik kpalmvik