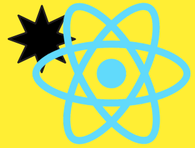
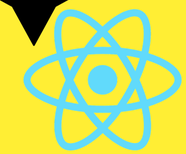








State Management and Reusable Code in React

A Brief History and Introduction

Daria Caraway
React Denver Jan 2021

@dariacaraway

Full-Stack Developer

React
TypeScript
Kotlin

workday.

Boulder

A Daria Timeline

2015

First **React** app built and shipped with CoffeeScript + ES5

2016-2018

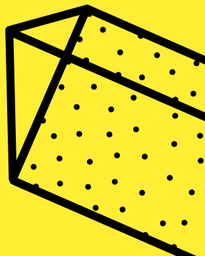
Quick detour to the magical land of Angular

2018

Started at Workday as a **React** Developer

2019

Switched teams to work on brand new React Application



@dariacaraway

A Daria Timeline

2015

First **React** app built and shipped with CoffeeScript + ES5

2016-2018

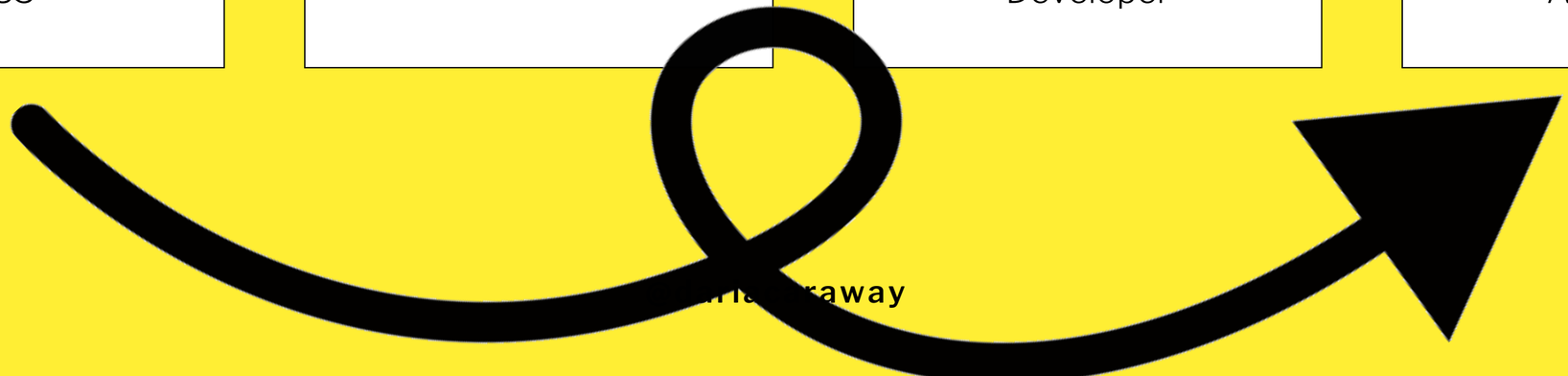
Quick detour to the magical land of Angular

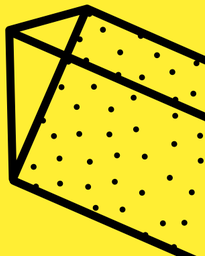
2018

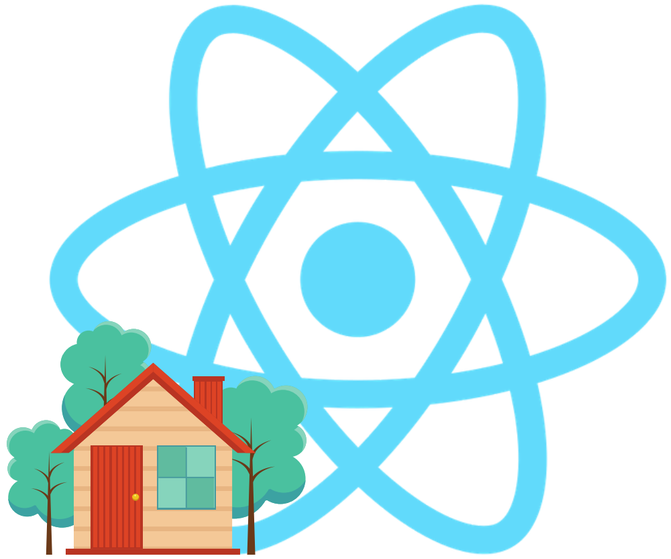
Started at Workday as a Front-End **React** Developer

2019

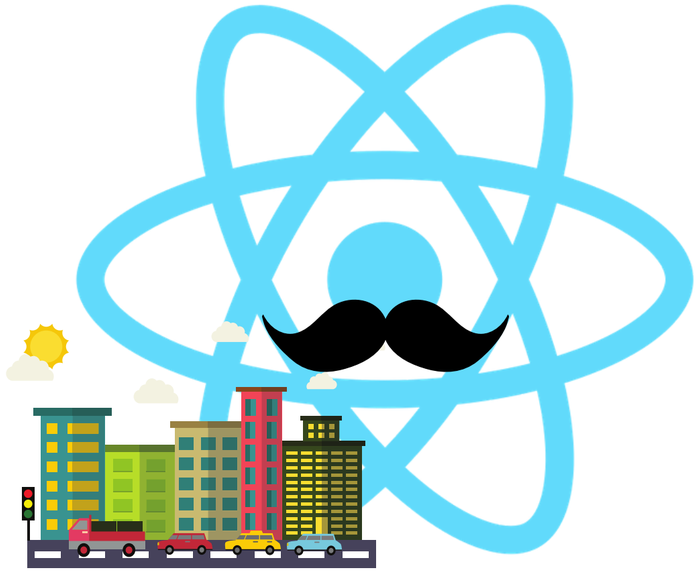
Switched teams to work on brand new React Application





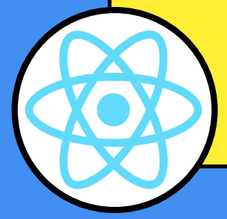


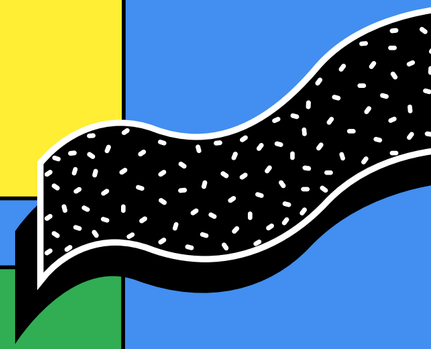
2015



2018

The Same React Underneath

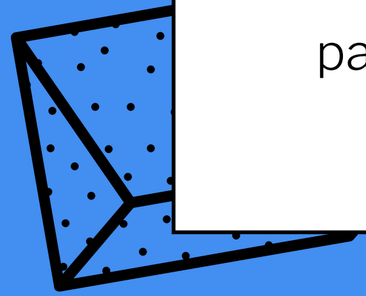




2021 Best Practices === 2015 Findings

Many of the best practices in 2021 were evolutions of patterns the community was trying out in 2015

We can understand how best to use React if we take some time to learn about the past iterations



EVOLVING QUESTIONS

As people start using React, two major problem themes are identified and continue today...

State Management

How do you manage data that your components need to know about to render?

Code Reuse

How do you share code across components that need access to the same type of data?

A (rough) React Timeline

of State Management and Code Reuse

2013-2015

✦

Facebook recommends the "Flux" data pattern and React supports Mixins

2015-2017

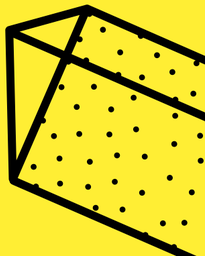
✦

Redux comes onto the scene and Higher Order Components replace Mixins

2018-2019

✦

React releases Hooks and a stable Context API. Community leans away from larger solutions

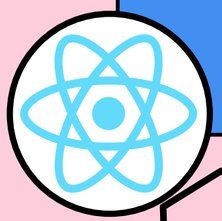


@dariacaraway

2013 Facebook

React is Publicly Released

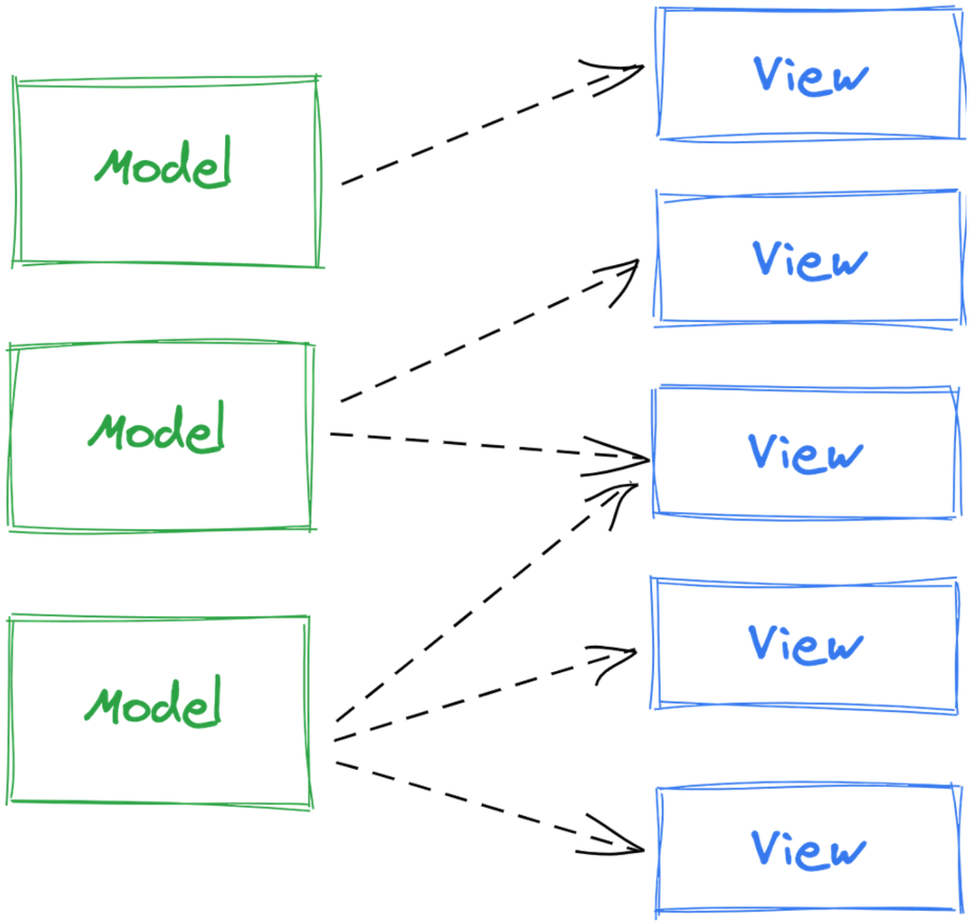
"React isn't an MVC framework. React is a library for building composable user interfaces. It encourages the creation of reusable UI components which present data that changes over time." [1]



2013 JS State Managment

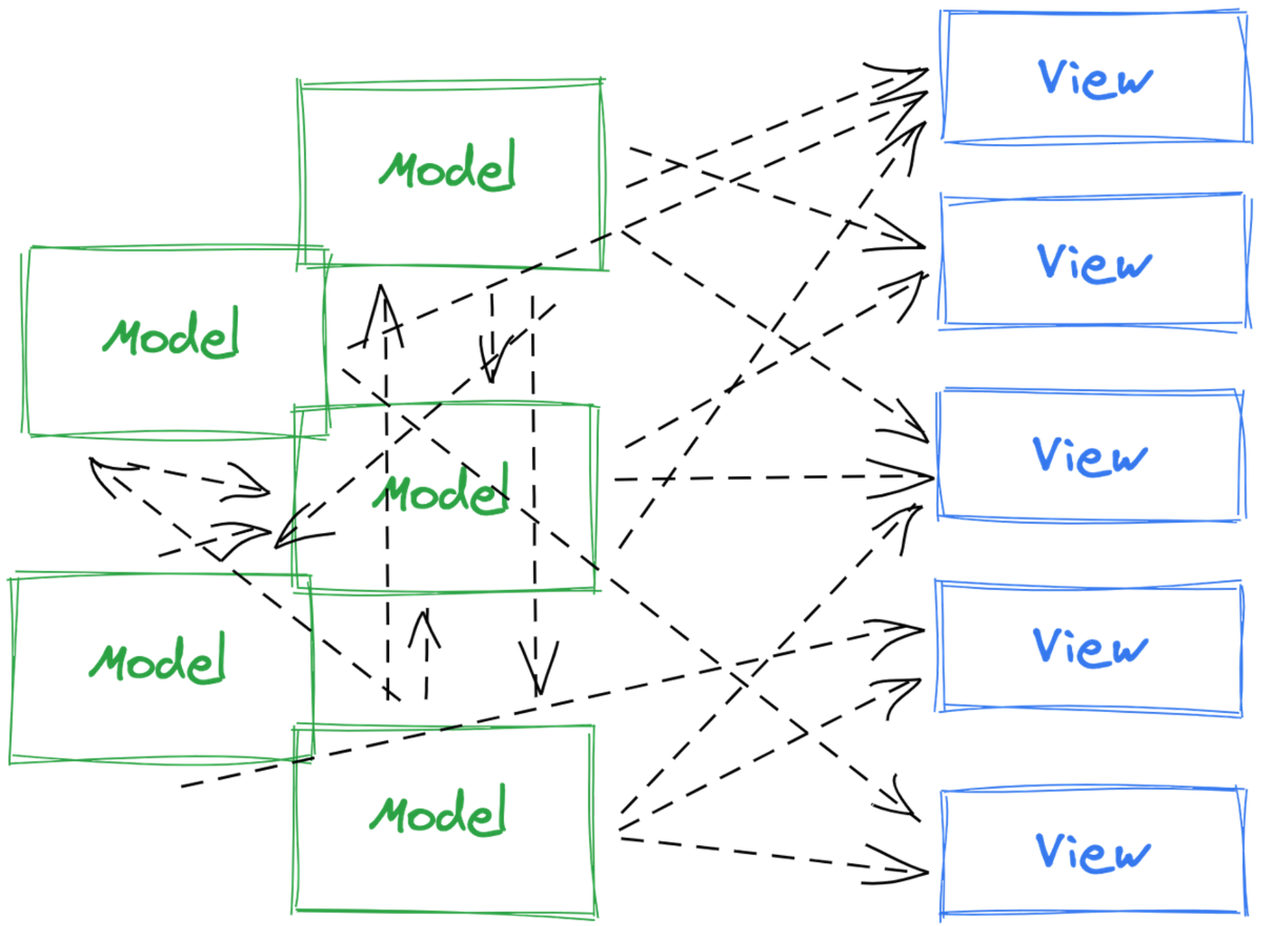
Frameworks like backbone.js and angular.js were popular for their Model-View-Controller data flows

The Model-View Data Flow



@dariacaraway

The Model-View Data Problem

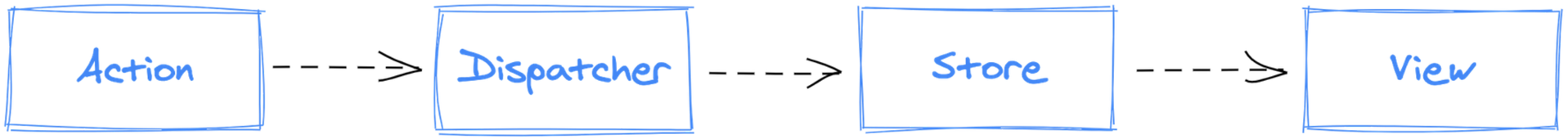


2013
Flux

A guide on how to architect and manage unidirectional data flow through an application that complements React's declarative style and one-way data binding. Not a framework.

@dariacaraway

The Flux Data Flow



Single Dispatcher

Logic to update the stores is managed in a single location. All data flows through the dispatcher via actions and is easily traced.

Multiple Stores

Stores manage application state for a particular domain.

The Flux Data Solution

2013
Mixins

An object (collection of functions) that has access to React lifecycle methods and state. React components are created with a mixin and have access to the object's functionality.

Mixin Definition

```
var FetchSearchResultsMixin = {
  getInitialState: function() {
    return {
      results: Store.getResults()
    }
  },

  componentDidMount: function() {
    Store.addChangeListener(this.handleChange)
  },

  componentWillUnmount: function() {
    Store.removeChangeListener(this.handleChange)
  },

  handleDataChange: function() {
    this.setState({
      results: Store.getResults()
    })
  }
}
```


@dariacaraway

Mixin Usage

```
var SearchResults = React.createClass({
  mixins: [FetchSearchResultsMixin],

  render: function() {
    var results = this.state.results;
    return (
      <div>
        {results.map(function(result) {
          return <Result result={result} key={result.id} />
        })}
      </div>
    )
  }
})
```

Re-Usable Functions

Mixins can be added to any React component and allow for components to build up combinations of shared logic by using multiple mixins

Access to State & React Lifecycle

Unlike plain JS functions, mixins have access to component state and class component lifecycle methods

Mixins For Code Reuse

A (rough) React Timeline

of State Management and Code Reuse

2013-2015

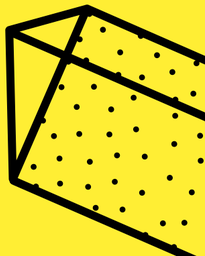
Facebook recommends the "Flux" data pattern and React supports Mixins

2015-2017

Redux comes onto the scene and Higher Order Components replace Mixins

2018-2019

React releases Hooks and a stable Context API. Community leans away from larger solutions



@dariacaraway

optimizely / nuclear-js

rpominov / fluce

~2015

glenjamin / fluctuations

goatslacker / alt

yahoo / fluxible

facebook / flux

goatslacker / microflux

threepointone / disto

acdlite / flummox

vigetlabs / microcosm

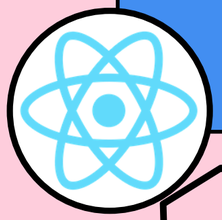
martyjs / marty

@dariacaraway

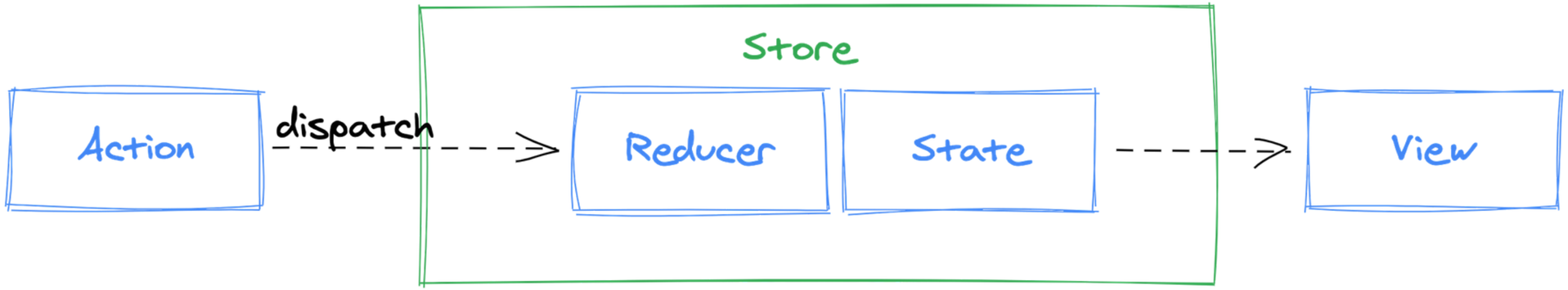
2015
Dan Abramov & Andrew Clark

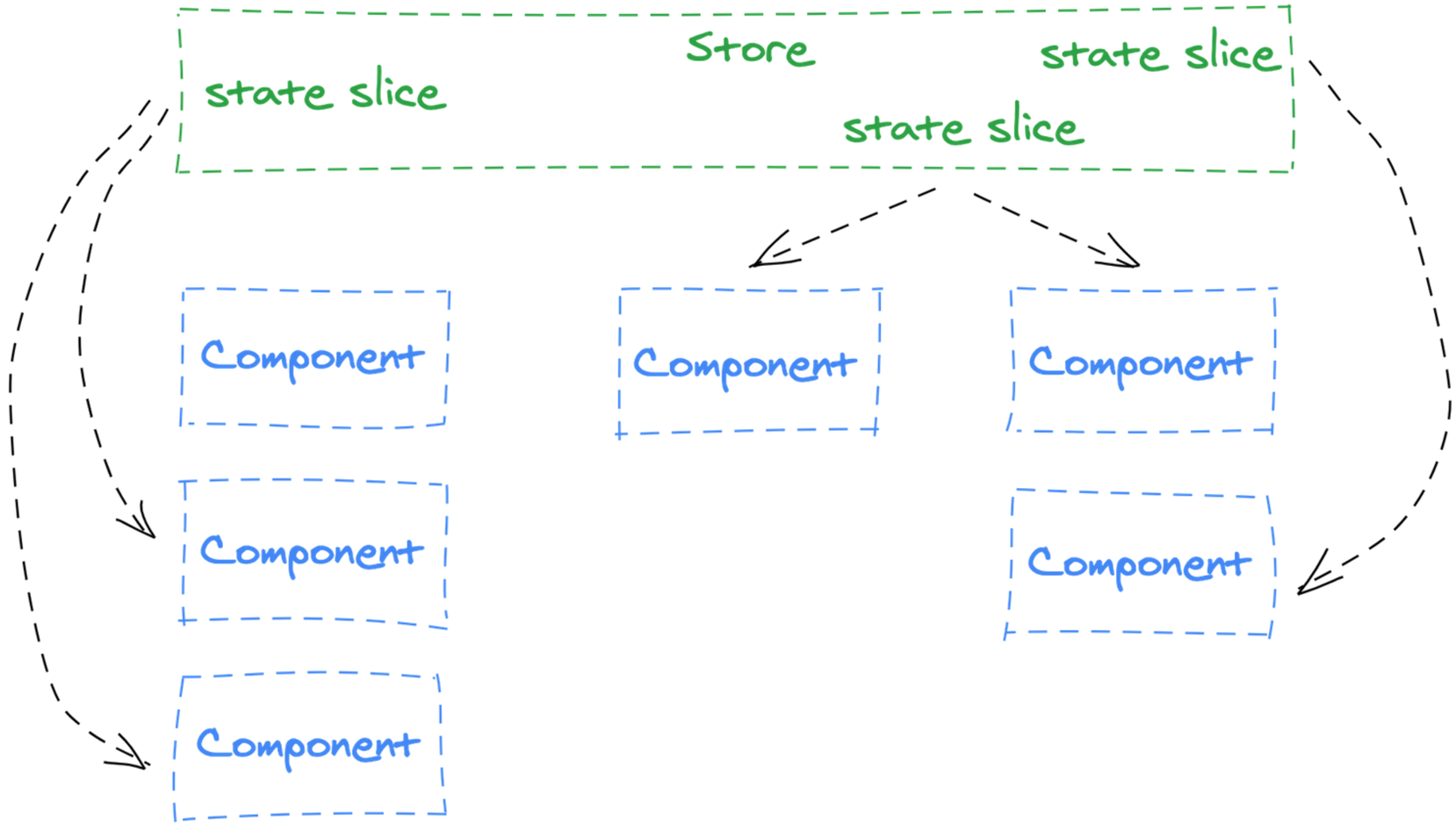
Redux is Released

A Flux+ implementation wrapped up in a well-defined, opinionated set of tools that allowed for hot-reloading and time-travel debugging.



The Redux Data Flow




Store
state slice
state slice
state slice
Component
Component
Component
Component
Component
Component

Single Immutable State Tree

State is never modified directly: immutable state allows for predictability, and replay-ability

Reducers & Middleware

Pure reducer functions take the actions and update the state. Third party plugins can be added to the redux data cycle pre or post reducer

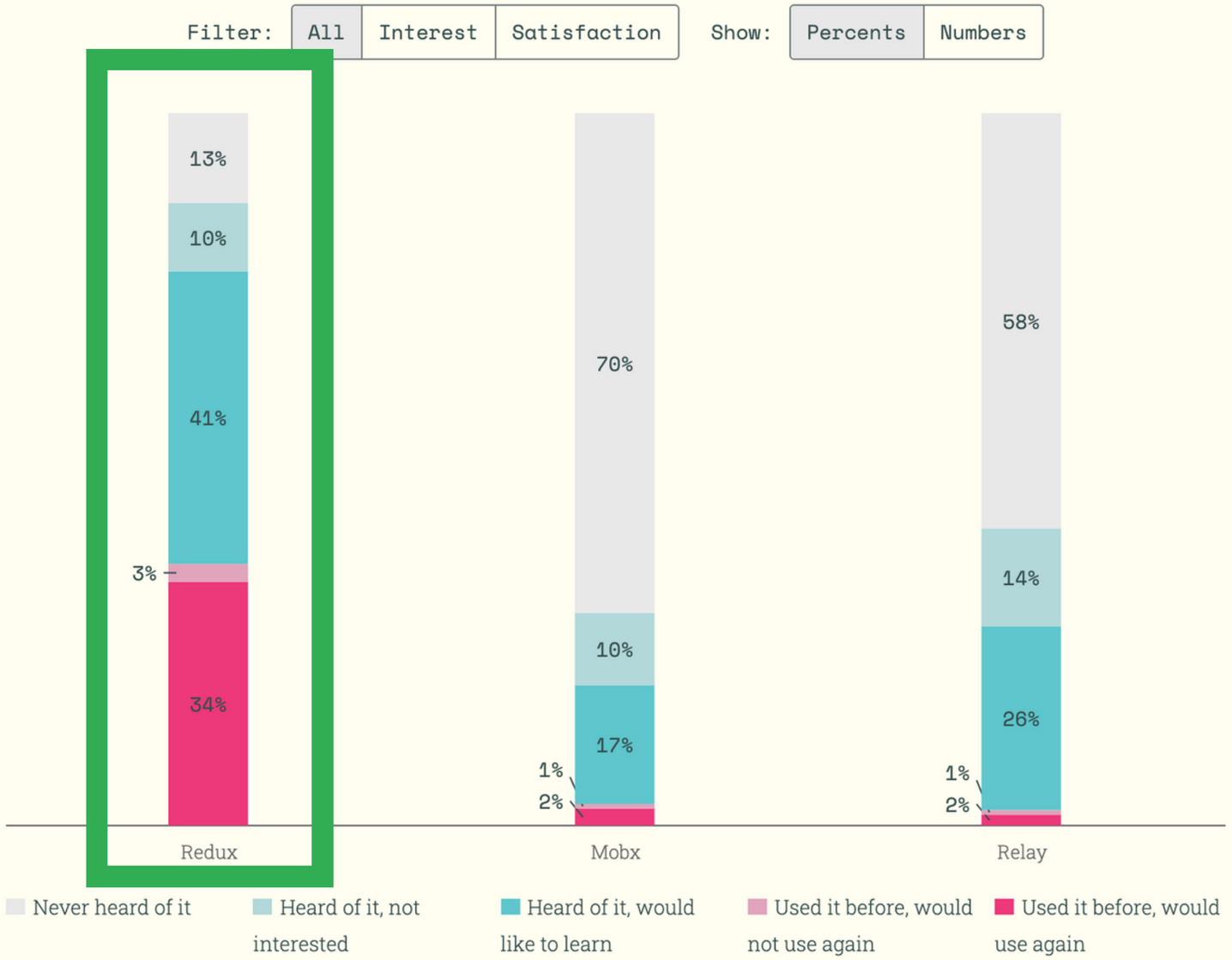
The React-Redux Solution

2016

Redux is emerging as a clear leader for React state management.

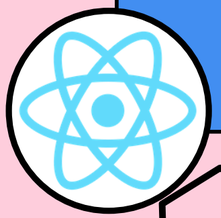
State Management

2016.STATEOFJS.COM/2016/STATEMANAGEMENT/



Mixins Considered Harmful

2016
Facebook



Mixins Considered Harmful

July 13, 2016 by Dan Abramov

"How do I share the code between several components?" is one of the first questions that people ask when they learn React. Our answer has always been to use component composition for code reuse. You can define a component and use it in several other components.

It is not always obvious how a certain pattern can be solved with composition. React is influenced by functional programming but it came into the field that was dominated by object-oriented libraries. It was hard for engineers both inside and outside of Facebook to

Mixin Concerns

Hard To Follow Dependencies

If you want to find where *this.function* is implemented, you have to search across many mixin files

Mixins Become a Confusing Web

Complex interdependencies can arise between mixins and become a tightly coupled web making your components difficult to reuse

ES6 Ships Without Mixins

React mixins were built on top of JavaScript mixins. ECMAScript 2015 (JavaScript ES6) did not initially support mixins. React mixins *could not* be used with ES6

2016

Higher Order Components (HOC)

An existing code reuse pattern that takes advantage of React Component composition. The component equivalent of a "higher order function".

Higher Order Component Definition

```
function getFetchSearchResultsComponent(BaseComponent) {
  return class extends React.Component {
    constructor(props) {
      super(props)
      this.handleChange = this.handleChange.bind(this)
      this.state = {
        data: Store.getResults()
      }
    }

    componentDidMount() {
      Store.addChangeListener(this.handleChange)
    }

    componentWillUnmount() {
      Store.removeChangeListener(this.handleChange)
    }

    handleChange() {
      this.setState({
        data: Store.getResults()
      })
    }

    render() {
      return <BaseComponent data={this.state.data} {...this.props} />
    }
  }
}
```


@dariacaraway

Higher Order Component Usage

```
const SearchResultsWithFetch =
      getFetchSearchResultsComponent(SearchResults)
```

Supports All React Components

HOCs can wrap React.createClass, ES6 class, or function components

Explicit Dependencies

Dependencies are managed by passing props clearly between React components and are easy to follow

View vs. Logic Components

Wrapper logic components keeps views separate and re-usable.

Higher Order Component Solution

A (rough) React Timeline

of State Management and Code Reuse

2013-2015

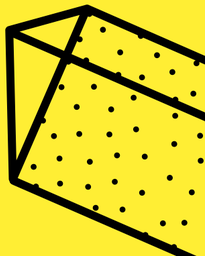
Facebook recommends the "Flux" data pattern and React supports Mixins

2015-2017

Redux comes onto the scene and Higher Order Components replace Mixins

2018-2019

React releases Hooks and a stable Context API. Community leans away from larger solutions



@dariacaraway


2018
Facebook
Stable Context API
"Introduce a new version of context that addresses existing limitations."[3]

Stable Context Definition

```
const SearchThemeContext = React.createContext('light')
class SearchResults extends React.Component {

  render() {
    const children = this.props.results.map((result) =>
      <SearchResult searchResult={result} />
    )
    return (
      <SearchThemeContext.Provider value={'blue'}>
        <div>{children}</div>
      </SearchThemeContext.Provider>
    )
  }
}
```

Stable Context Usage

```
class SearchResultButton extends React.Component {
  render() {
    return (
      <SearchThemeContext.Consumer>
          {color => (
             <button style={{background: color}}>
             {this.props.children}
           </button>
          )}
      </SearchThemeContext.Consumer>
    )
  }
}

class SearchResult extends React.Component {
  render() {
    return (
      <div>
        {this.props.description}
        <SearchResultButton>Link to Result</SearchResultButton>
      </div>
    )
  }
}
```

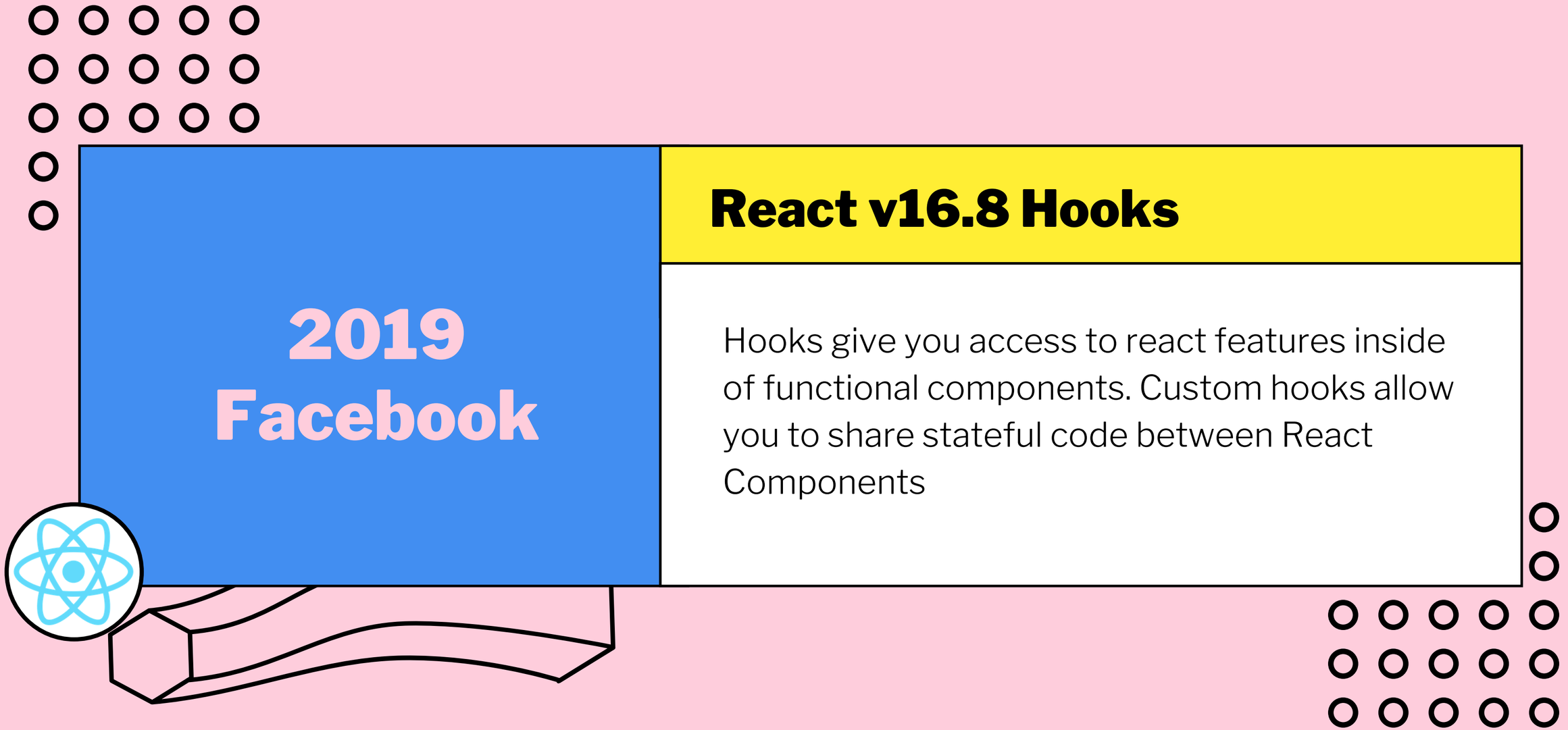
More Intuitive Developer Experience

Provider - consumer interaction is explicit and easier to follow

Updates Consistantly

Context updates will trigger a re-render in all components consuming the context regardless of shouldComponentUpdate. Equality is checked by reference to ensure updates on object changes

Stable Context Solution


2019
Facebook
React v16.8 Hooks
Hooks give you access to react features inside of functional components. Custom hooks allow you to share stateful code between React Components

Custom Hook Definition

```
import { useState, useEffect } from 'react';
function useFecthSearchResults() {
  const [searchResults, setSearchResults] = useState(null)

  useEffect(() => {
    function handleChange(results) {
      setSearchResults(results)
    }

    Store.addChangeListener(handleChange)

    return () => {
      Store.removeChangeListener(handleChange)
    }
  })

  return searchResults
}
```

Custom Hook Usage

```
function SearchResults() {
  const results = useFecthSearchResults()
  return (
    <div>
      {results.map(function(result) {
        return <Result result={result} key={result.id} />
      })}
    </div>
  )
}
```

Explicit Dependencies

Unlike mixins, hooks are called explicitly from components and cannot have function name collisions

Composability

Hooks are composable and can be built on top of eachother.

Access to State & React Lifecycle

Functions have access to state and react lifecycle hooks

Hooks Solution

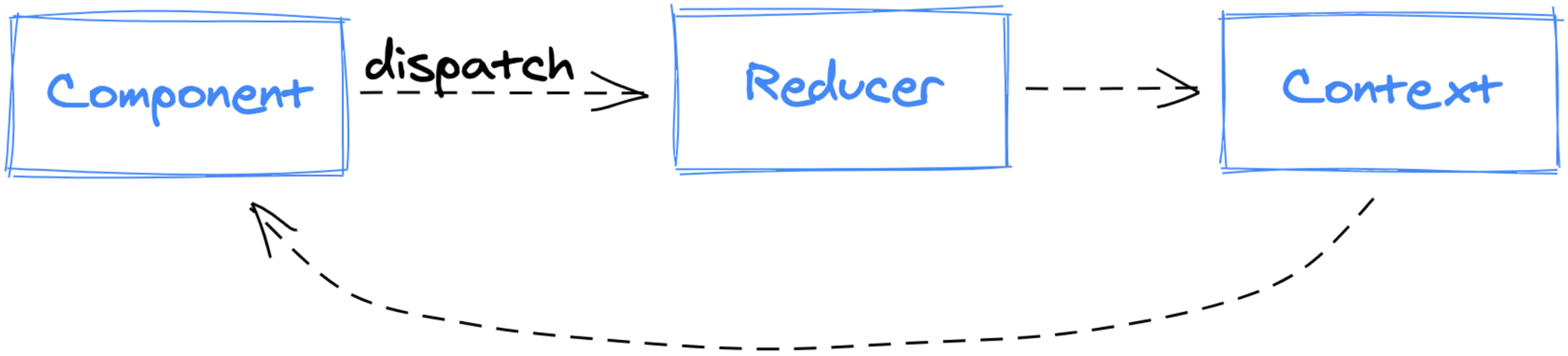
2019

Movement towards a lighter state management solution.

2019
React *is* State Management

Context and hooks makes React itself a state management solution that the community can use to scale.

Redux With React



Freely Localize State

State can be kept at the lowest possible level with component state or contexts. All state can be updated with the same means

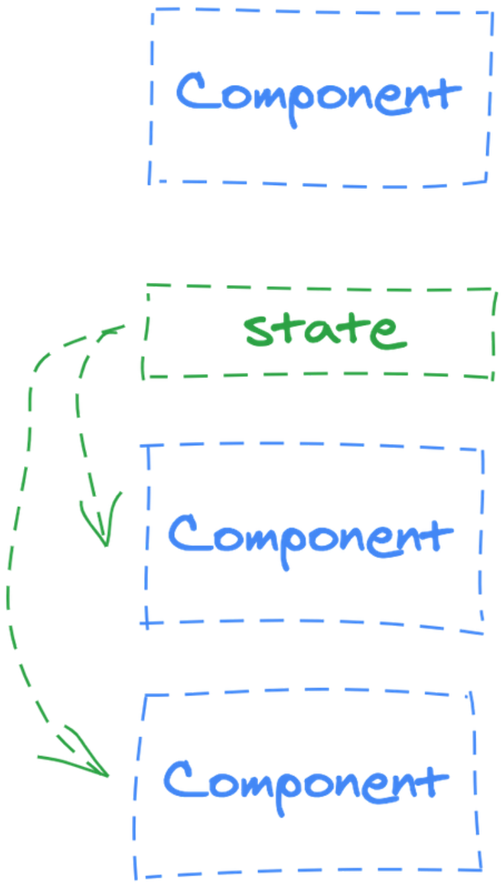
No Extra Libraries

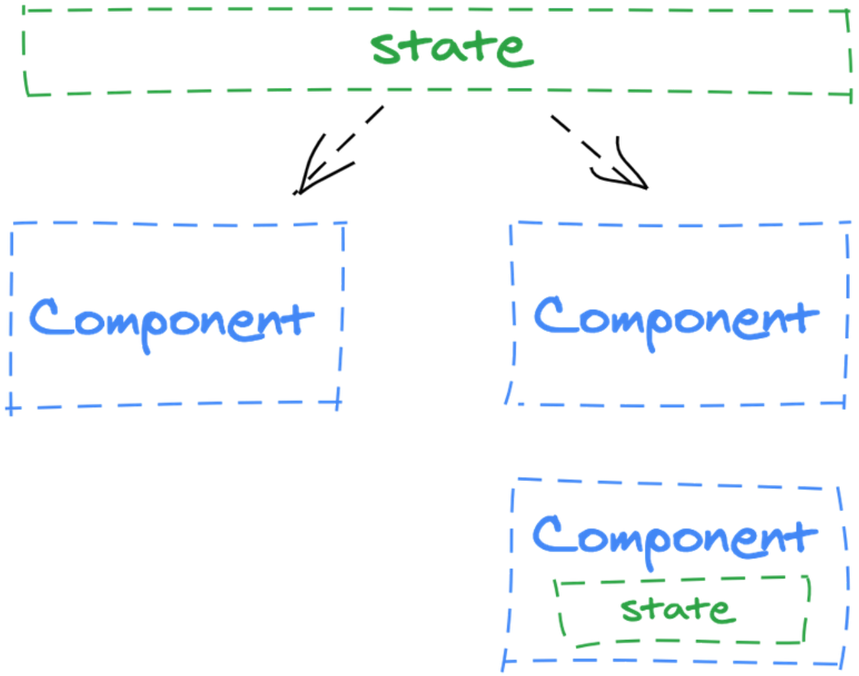
No need to install a small or large library and learn a separate tool

First Class Maintenence

State management with react tools will likely never be left behind by a future react release

React State Management Solution


Component
state
Component
Component


state
Component
Component
Component
state

Freely Localize State

State can be kept at the lowest possible level with component state or contexts. All state can be updated with the same means

No Extra Libraries

No need to install a small or large library and learn a separate tool

First Class Maintenence

State management with react tools will likely never be left behind by a future react release

React State Management Solution

2021.. How did we get here?

Key takeaways from historical React

Flux & Mixins

What did we learn from 2013-2015 React?

- Unidirectional data flow helps cut down on complexity
- Explicit can be easier to maintain that implicit
- Complex interdependencies between shared code makes components less re-usable

Redux & Higher Order Components

What did we learn from 2016-2018 React?

- Immutable state leads to predictability
- Everything in global state can be hard to maintain and un-necessary for every application
- Keeping logic components separate from view components helps re-usability

TO BE CONTINUED...

Hooks & Contexts

Building block state management and code re-use solution

- Keeps state as local as possible, option for global state
- Prop drilling can be avoided with React contexts & every consumer component when context data changes by reference

Hot Third Party Options

There are still many other solutions out there, each with a unique value proposition

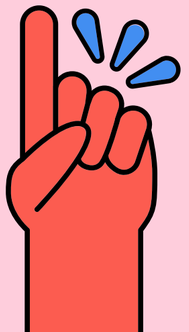
- Flux-y: Redux, Mobx
- GraphQL: Apollo, Relay
- X-State
- Facebook Recoil

@dariacaraway

Learning From the Past

Mixins + Flux ➡ Hooks + Context

React has gone through quite a few iterations, but it is still very much the same React underneath. We can apply learnings from every iteration to build better applications today.



Thank You!

@dariacaraway

1. https://reactjs.org/blog/2013/06/05/why-react.html
2. https://reactjs.org/blog/2016/07/13/mixins-considered-harmful.html
3. https://github.com/acdlite/rfcs/blob/1e25818e47037a7de80e95793c4129e9887d6f2b/text/0000-new-version-of-context.md
4. https://reactjs.org/docs/legacy-context.html

https://facebook.github.io/flux/docs/in-depth-overview
https://reactjs.org/blog/2014/05/06/flux.html
https://code-cartoons.com/a-cartoon-guide-to-flux-6157355ab207
https://reactjs.org/docs/react-without-es6.html
https://reactjs.org/blog/2016/07/13/mixins-considered-harmful.html
https://reactjs.org/docs/higher-order-components.html
https://reactjs.org/docs/legacy-context.html
https://reactjs.org/docs/context.html
https://medium.com/@dan_abramov/you-might-not-need-redux-be46360cf367
https://medium.com/@dan_abramov/the-evolution-of-flux-frameworks-6c16ad26bb31

https://reactjs.org/blog/2018/03/29/react-v-16-3.html
http://2016.stateofjs.com/2016/statemanagement/
https://kentcdodds.com/blog/reacts-new-context-api
https://reactjs.org/docs/legacy-context.html
https://blog.logrocket.com/pitfalls-of-overusing-react-context/