New React API: enabling better patterns

New React API features

- New Context API
- Suspense and Concurrent rendering
- lazy, memo, ...
- Hooks API

Broad picture...

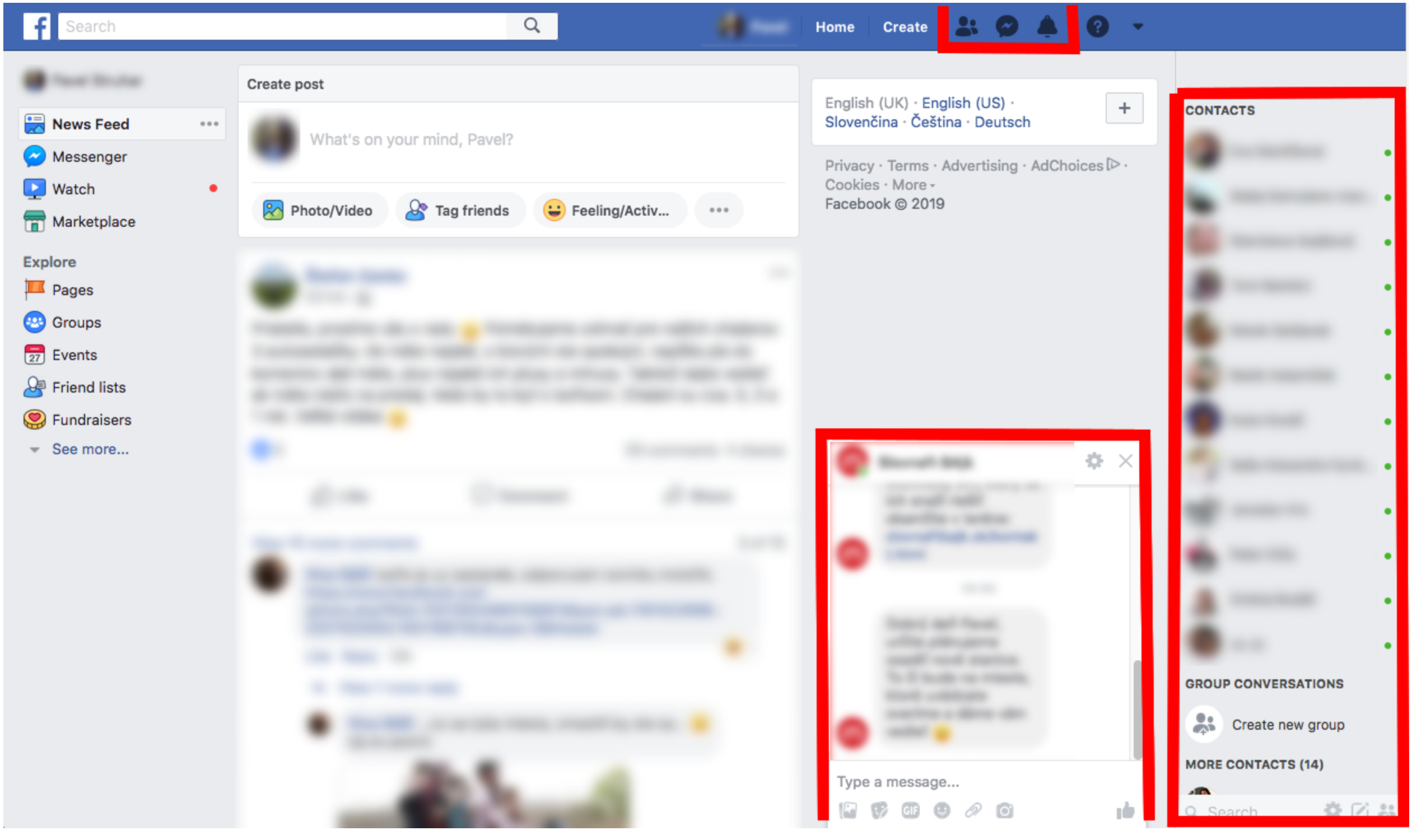
What is the greatest contribution of React?

The most important thing that React brought to the community was the idea of unidirectional data flow.

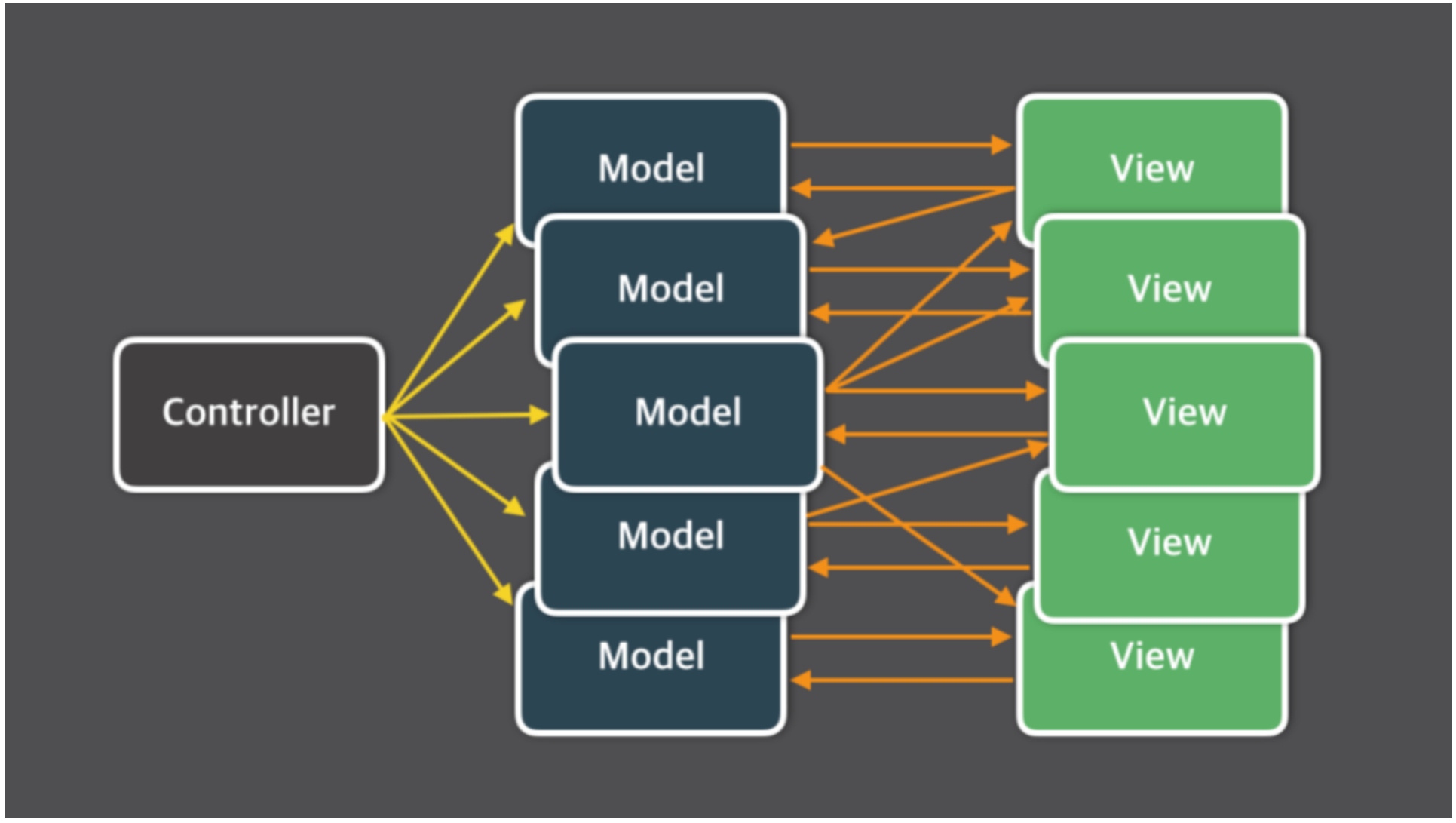
by Igor Minar, Angular tech lead, ReactiveConf 2016

Why? Long story short...

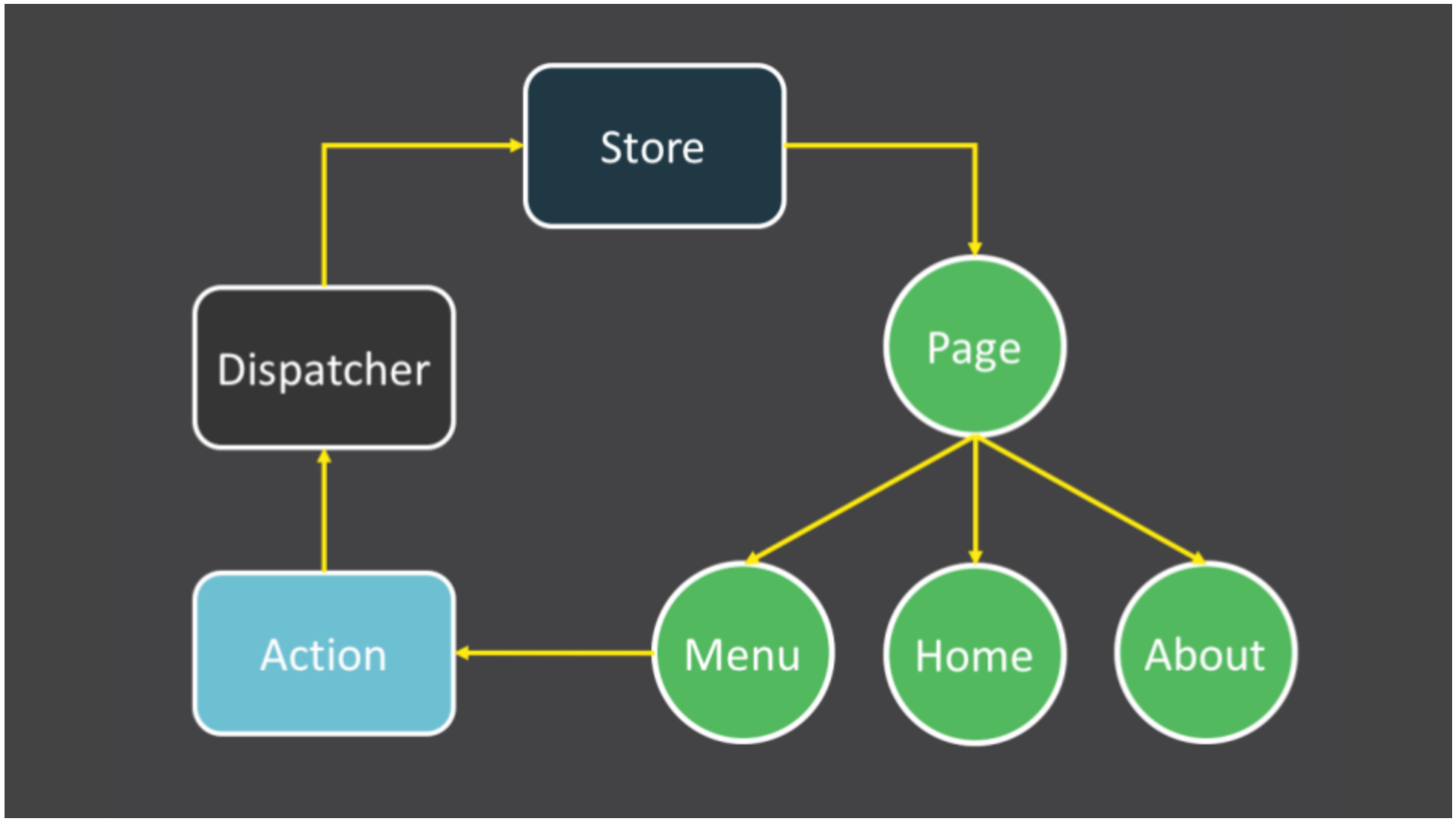
Bugs in Facebook chat



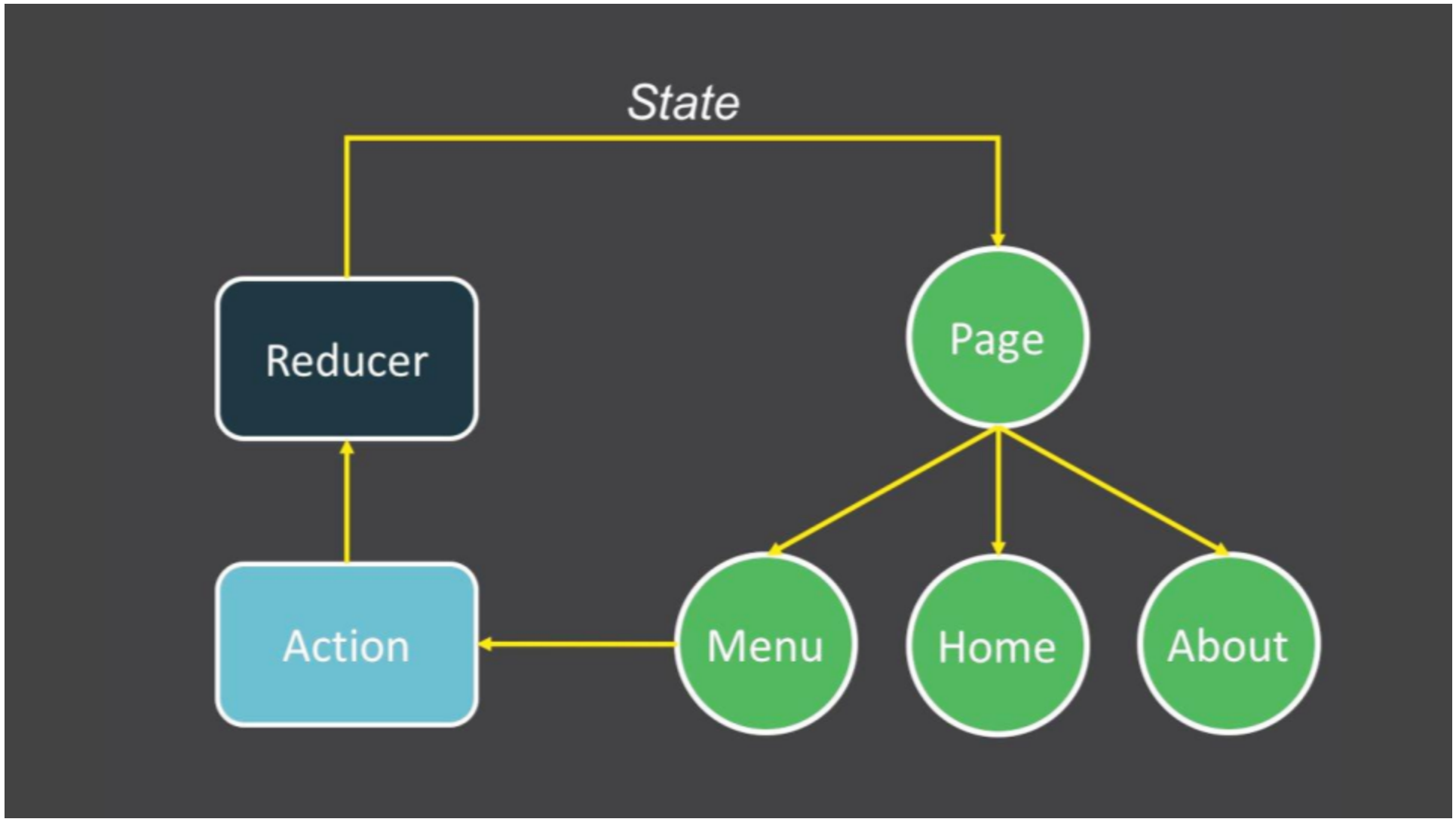
MVC


Model
Model
Model
Model
Model
Controller
View
View
View
View
View

FLUX



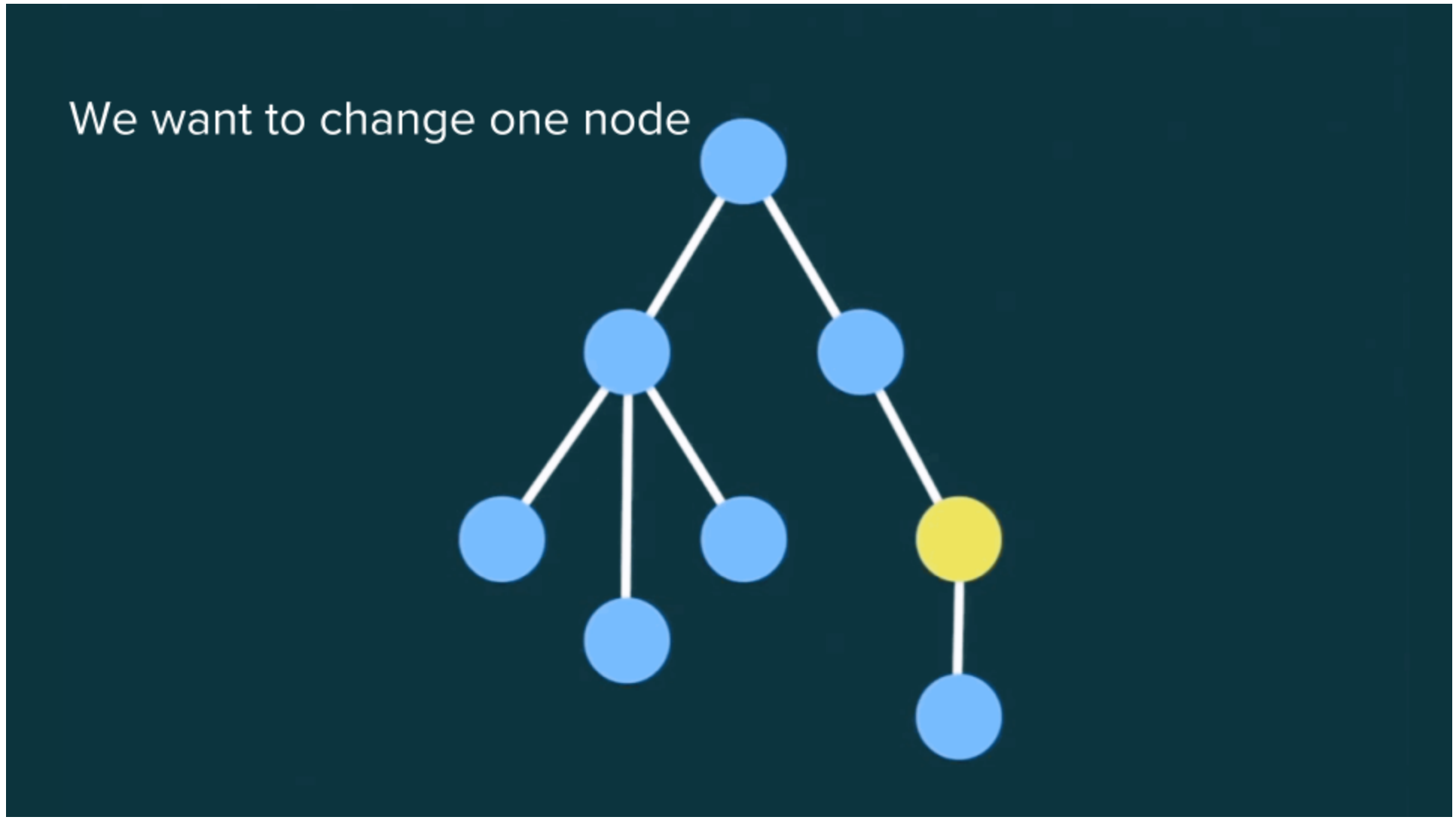
Redux



Asynchronicity + mutation

http://redux.js.org/docs/introduction/Motivation.html

Immutable data operations


We want to change one node

Immutable App Architecture

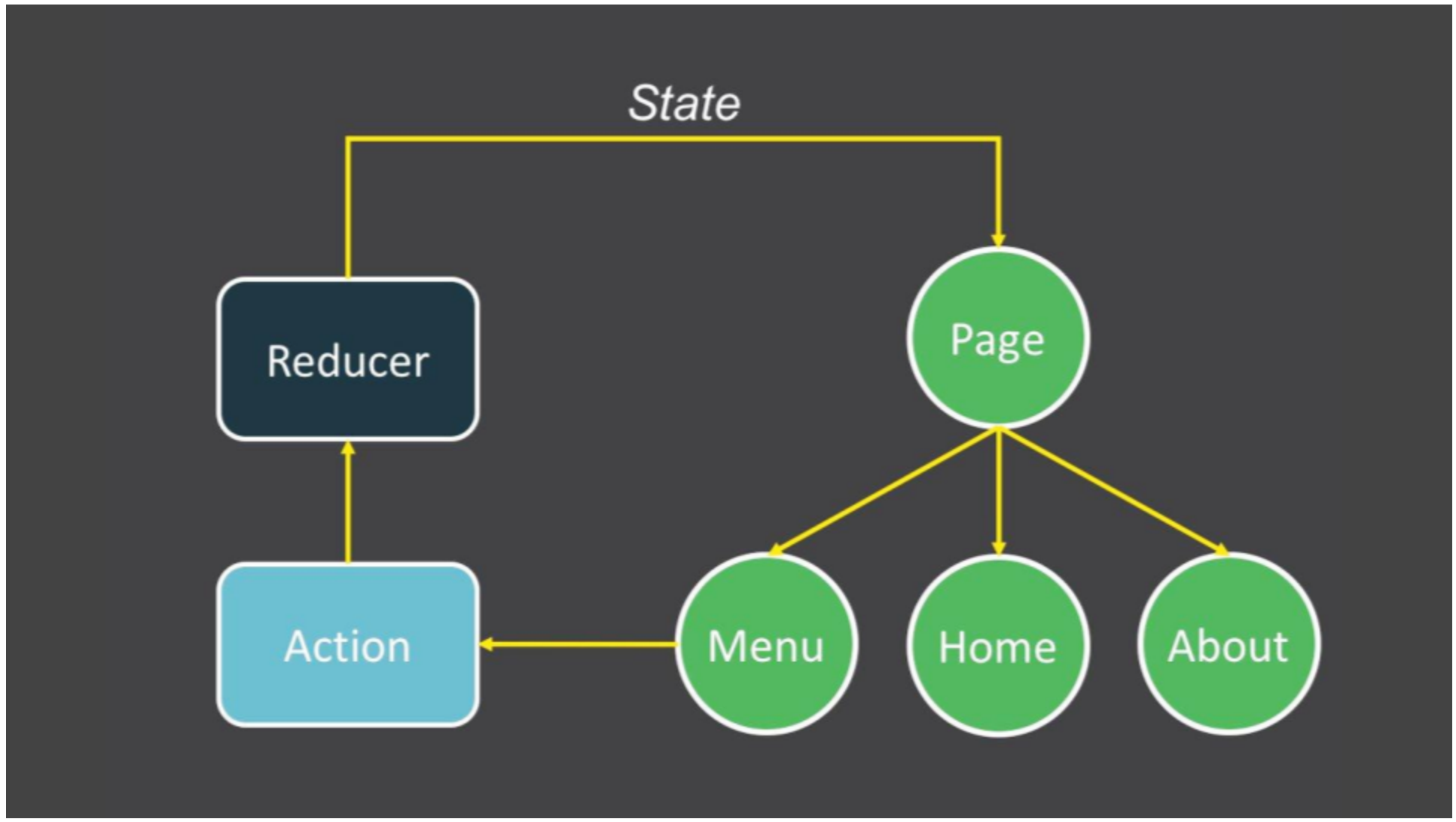
by Lee Byron

https://youtu.be/pLvrZPSzHxo

❤️ Redux

since 2015

Looks simple



Reality...

[React / Redux cheat sheet]

Too much boilerplate

Reducers
Action type constats
Action creators
mapStateToProps
mapDispatchToProps
...

💔 you might not need Redux

2016 - ?

You Might Not Need Redux

Dan Abramov

Sep 19, 2016 · 3 min read

People often choose Redux before they need it. “What if our app doesn’t scale without it?” Later, developers frown at the indirection Redux introduced to their code. “Why do I have to touch three files to get a simple feature working?” Why indeed!

People blame Redux, React, functional programming, immutability, and many other things for their woes, and I understand them. It is natural to compare Redux to an approach that doesn’t require “boilerplate” code to update the state, and to conclude that Redux is just complicated. In a way it is, and by design so.

https://medium.com/@dan_abramov/you-might-not-need-redux-be46360cf367

https://twitter.com/acdlite/status/1024852895814930432

Thinking in React

Why take Redux if you don't need it? Start with React alone. Add more stuff later.

https://reactjs.org/docs/thinking-in-react.html

Redux benefits

- Single state tree + immutable operations (inside reducers)
- **connect(<YourComponent />)** to state anywhere

```
<Counter increment={2} />
```

0

```
import React from 'react';

export class Counter extends React.Component {
  state = {
    counter: 0
  }
  handleClick = () => {
    this.setState({counter: this.state.counter + this.props.increment})
  }
  render() {
    return <button onClick={this.handleClick}>{this.state.counter}</button>
  }
}
```

Using object (may fail due to asynchronicity)

```
this.setState(
  { counter: this.state.counter + this.props.increment }
);
```

Using function

```
this.setState(
  (state) => ({
    ...state,
    counter: state.counter + this.props.increment
  })
);
```

Extract action

```
incrementAction = (state) => ({
  ...state,
  counter: state.counter + this.props.increment
})

...

this.setState(this.incrementAction);
```

Action creator

```
incrementAction = (increment) => (state) => ({
  ...state,
  counter: state.counter + increment
})

...

this.setState(incrementAction(this.props.increment));
```

Familiar? (redux)

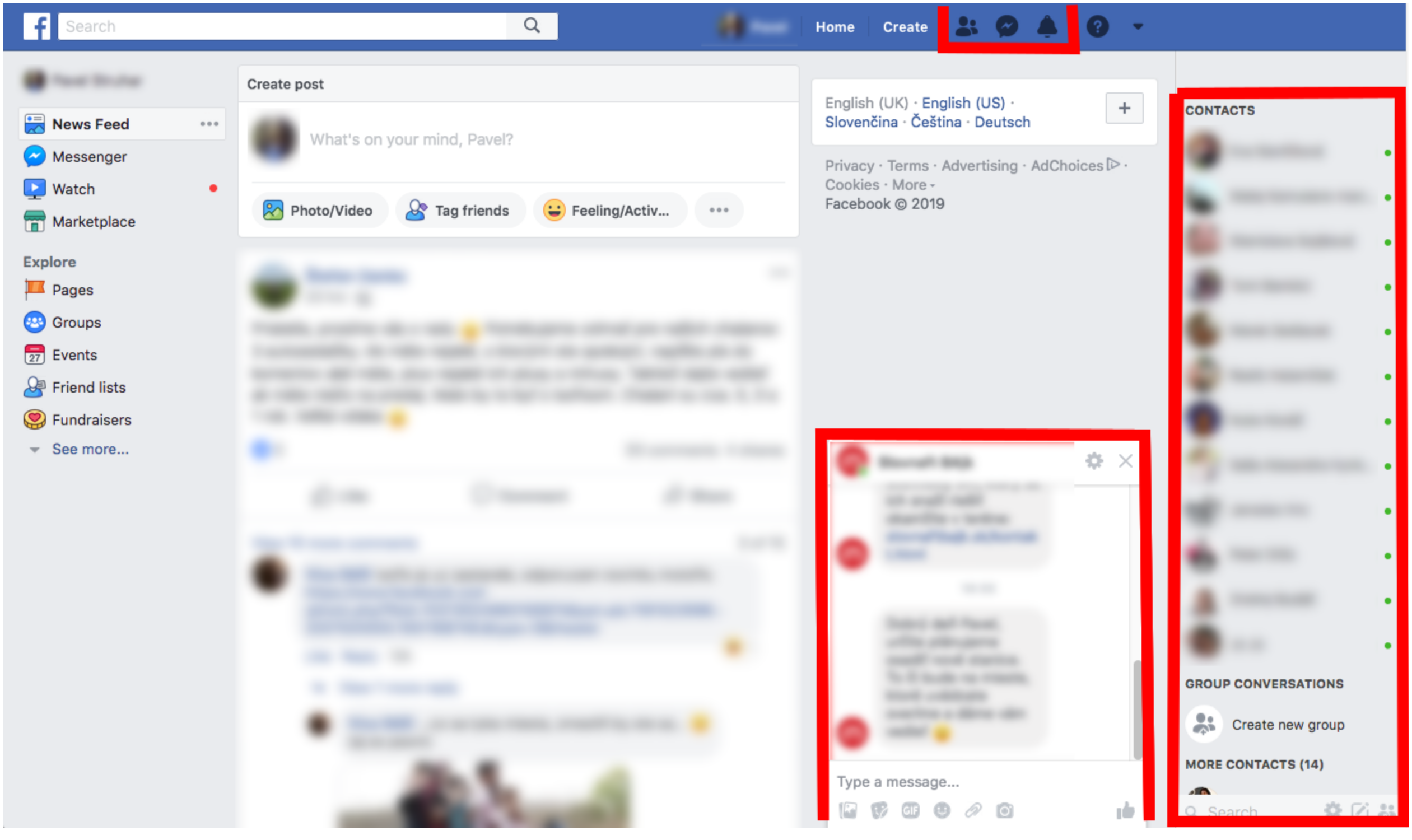
```
const reducer = (state, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return {
        ...state,
        counter: state.counter + 1
      };
      ...
    default:
      return state;
  }
}
```

Redux benefits without Redux

✅ Single state tree + immutable operations (inside reducers)

❓ **connect(<YourComponent />)** to state anywhere

connect() to the same state



React Context

this.context - experimental

Stable Context API - React 16.3.0 (March 29, 2018)

When to use Context

- Share data that can be considered "global"
- Avoid passing props through multiple elements

```
const UserContext = React.createContext({
  user: 'Joe',
  ...
});

class App extends React.Component {
  render() {
    return (
      <UserContext.Provider value={{ user: 'Mike' }}>
        <Main />
      </UserContext.Provider>
    );
  }
}
```

```
class Toolbar extends React.Component {
  render() {
    return (
      <div>
        ...
        <UserContext.Consumer>
          {
            ({ user }) => (
              <div>Hi {user}!</div>
            )
          }
        </UserContext.Consumer>
      </div>
    )
  }
}
```

```
export const UserContext = React.createContext();

class App extends React.Component {
  state = { user: 'Mike' }
  handleLogout = () => {
    this.setState( state => ({...state, user: null}))
  }
  render() {
    return (
      <UserContext.Provider
        value={{
          user: this.state.user,
          onLogout: this.handleLogout
        }}
      >
        <Main />
      </UserContext.Provider>
    );
  }
}
```

```
class Toolbar extends React.Component {
  render() {
    return (
      <div>
        ...
        <UserContext.Consumer>
          {
            ({ user, onLogout }) => (
              <div>Hi {user}!
                <button onClick={onLogout}>Logout</button>
              </div>
            )
          }
        </UserContext.Consumer>
      </div>
    )
  }
}
```

Extracting the "global" state

```
// UserContext.js
import React from 'react';
const { Consumer, Provider } = React.createContext();

class UserProvider extends React.Component {
  state = { user: 'Mike' }
  handleLogout = () => {
    this.setState( state => ({...state, user: null}))
  }
  render() {
    <Provider value={
      user: this.state.user,
      onLogout: this.handleLogout
    }>{this.props.children}</Provider>
  }
}

export { UserProvider, Consumer as UserConsumer }
```

```
import { UserProvider } from './UserContext'

class App extends React.Component {
  render() {
    return (
      <UserProvider>
        <Main />
      </UserProvider>
    );
  }
}
```

```
import { UserConsumer } from './UserContext'
class Toolbar extends React.Component {
  render() {
    return (
      <div>
        ...
        <UserConsumer>
          {
            ({ user, onLogout }) => (
              <div>Hi {user}!
                <button onClick={onLogout}>Logout</button>
              </div>
            )
          }
        </UserConsumer>
      </div>
    )
  }
}
```

Exercise

Both buttons are re-rendered every time - they don't need to be.

0

-1 +1

How?

- shouldComponentUpdate()
- PureComponent - but needs to be a class component
- React.memo - wrap function component

React.memo

```
React.memo(
  (props) => <div>{props.greeting}</div>
);
```

React Suspense

Suspense + React.lazy

```
import React, {lazy, Suspense} from 'react';
const OtherComponent = lazy(() => import('./OtherComponent'));

function MyComponent() {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <OtherComponent />
    </Suspense>
  );
}
```

React Hooks

slides by @mariouhrin

Thank you

Made with MDX Deck

https://github.com/jxnblk/mdx-deck