REACT NATIVE

Anna Maziejuk

Kamil Jankowski

React Native

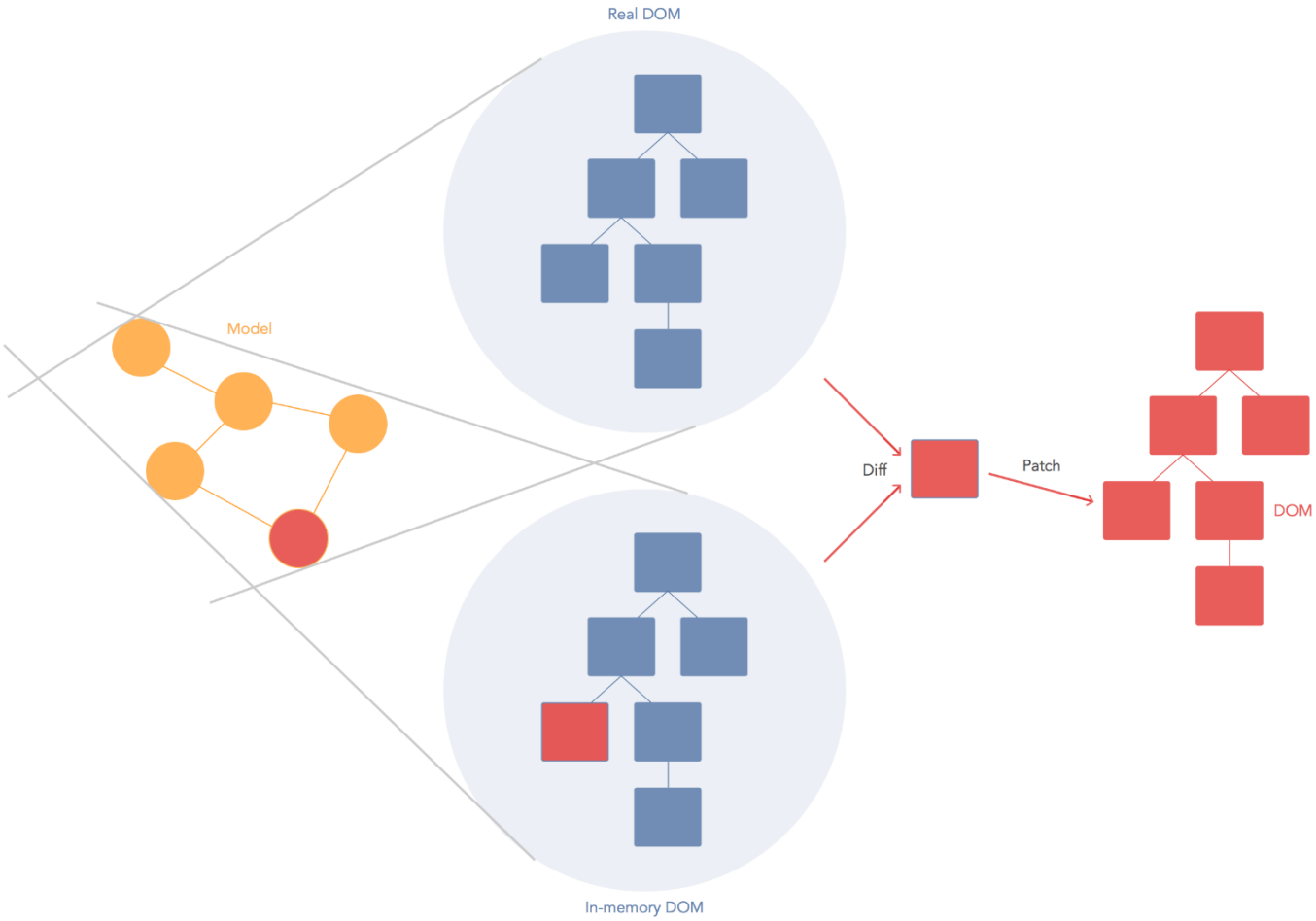
- Framework/biblioteka bazujący na React
- Pozwala na tworzenie aplikacji tylko za pomocą języka JavaScript
- Wspiera platformy iOS i Android

REACT

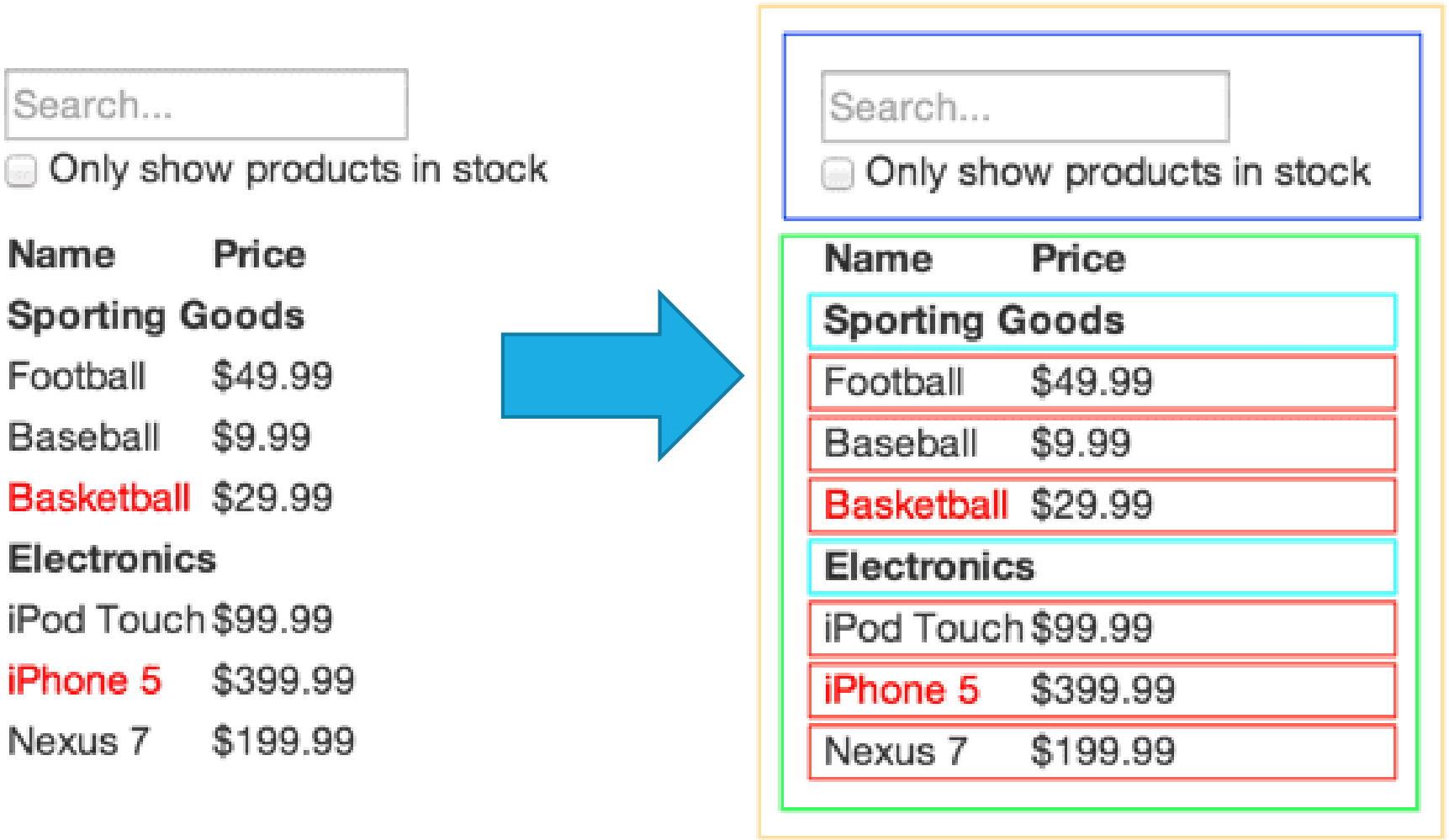
React

- Framework webowy/biblioteka stworzony przez Facebooka
- JavaScript, ECMAScript 6, **JSX**, TypeScript
- **Wirtualy DOM**
- HotReload
- **Podział na komponenty**
- **State**, **props** i cykl życia komponentu

Wirtualny DOM



Myślenie w React



JSX

- Rozszerzenie języka JavaScript wzorowane na XML
- Pozwala na umieszczanie znaczników HTML w kodzie JS
- Transpiluje się do JavaScript
- Małe litery – tagi HTML/SVG
- Wielkie litery – „własne” komponenty

```jsx
const LinksList = (
    <ul id="list-id">
        <li><a href="http://www2.cs.put.poznan.pl">Instytut Informatyki</a></li>
        <li><a href="https://fc.put.poznan.pl">Wydział Informatyki</a></li>
        <li><a href="https://www.put.poznan.pl/">Politechnika Poznańska</a></li>
    </ul>
    );
```

- Instytut Informatyki
- Wydział Informatyki
- Politechnika Poznańska

```jsx
const LinksList = React.createElement(
    "ul",
    { id: "list-id" },
    React.createElement(
      "li",
      null,
      React.createElement(
        "a",
        { href: "http://www2.cs.put.poznan.pl" },
        "Instytut Informatyki"
      )
    ),
    React.createElement(
      "li",
      null,
      React.createElement(
        "a",
        { href: "https://fc.put.poznan.pl" },
        "Wydział Informatyki"
      )
    ),
    React.createElement(
      "li",
      null,
      React.createElement(
        "a",
        { href: "https://www.put.poznan.pl/" },
        "Politechnika Poznańska"
      )
    )
  );
```

Props

- Jest przekazywany jako obiekt do komponentu i używany do obliczenia zwracanej wartości
- Zmiany w props spowodują ponowne obliczenie zwracanej wartości
- Może być dowolną wartością (string, numer, obiekt, funkcja…)

State

- Wewnętrzny stan komponentu
- Umożliwia wewnętrzne zarządzanie konfiguracją komponentu
- `this.state` jest właściwością klasy dla instancji komponentu
- Jest nie mutowalny, ale może być uaktualniony przez wywołanie `this.setState()`
- Jakiekolwiek zmiany w state powodują ponowny render

REACT NATIVE VS REACT.JS

Podobieństwa

- JavaScript i JSX
- State i props
- Cykl życia komponentów
- Możliwość wykorzystania TypeScript lub Flow
- Wiele bibliotek
- Możliwość użycia Redux

Różnice

React.js

- Web
- Transpilacja do commonJs
- Tagi HTML
- CSS
- Nawigacja poprzez linki i dodatkowe biblioteki
- Podstawowe komponenty (tagi HTML), reszta z zewnętrznych bibliotek

React Native

- Aplikacje mobilne (Android, iOS)
- Transpilacja na natywny kod (Java,Swift)
- Komponenty bazowe
- Style (jako obiekty JSON)
- Natywna nawigacja
- Bardzo wiele komponentów z biblioteki React
- Bardzo duża liczba zewnętrznych bibliotek
- Możliwość tworzenia komponentów w natywnych językach i dalsze używanie ich w JS

REACT NATIVE

Jak działa RN

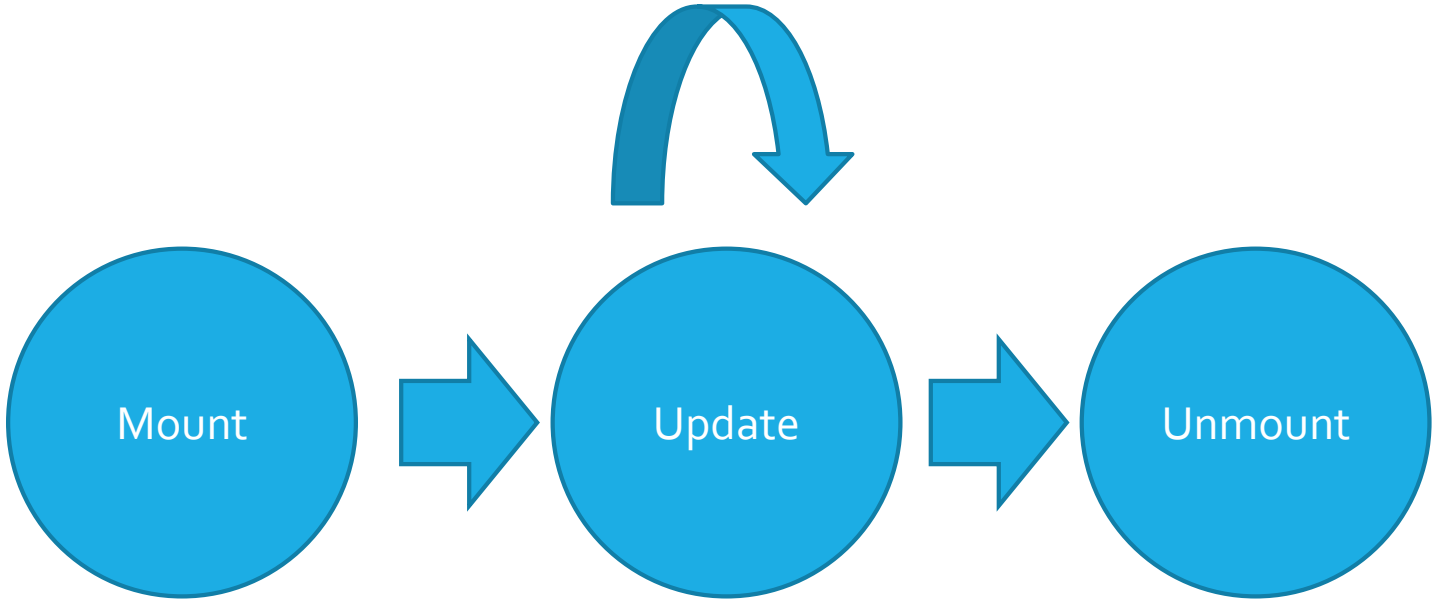
- Kod JavaScript jest transpilowany do natywnego kodu
- Oddzielne wątki dla UI i dla JavaScript
- Asynchroniczna komunikacja przez „most”
  - Wątek JavaScript może nie odpowiadać , jednak UI nadal będzie działał

Komponenty RN

- Należy je zaimportować z 'react-native'
- div → View
- span → Text
  - Wszystkie kawałki tekstu muszą być opatrzone tagiem
- button → Button
- ScrollView
- …

https://facebook.github.io/react-native/docs/components-and-apis.html

Cykl życia komponentów



Mount

- **constructor(props)**
  - Inicjalizuje State
- **componentWillMount()**
  - w praktyce nieużywana
- **render()**
  - Główna funkcja, która renderuje komponent
  - Zwraca korzeń (node)
- **componentDidMount()**
  - Robi wszystko, co nie jest potrzebne UI (np. liczniki)
  - Można użyć w celu pobrania danych z zewnętrznego API lub subskrybcji

Update

- **componentWillReceiveProps(nextProps)**
  - Uaktualnia wszystkie pola zawarte w state, które zależą od props
- **shouldComponentUpdate(nextProps, nextState)**
  - Porównuje zmienione wartości i zwraca true jeśli komponent powinien zostać przerenderowany
  - Jeśli zwróci false, to cykl uaktualniania zostaje przerwany
- **componentWillUpdate(nextProps, nextState)**
  - Rzadko wykorzystywany
- **render()**
- **componentDidUpdate(prevProps, prevState)**
  - Robi wszystko, co nie jest potrzebne UI (np. żądania sieci)

Unmount

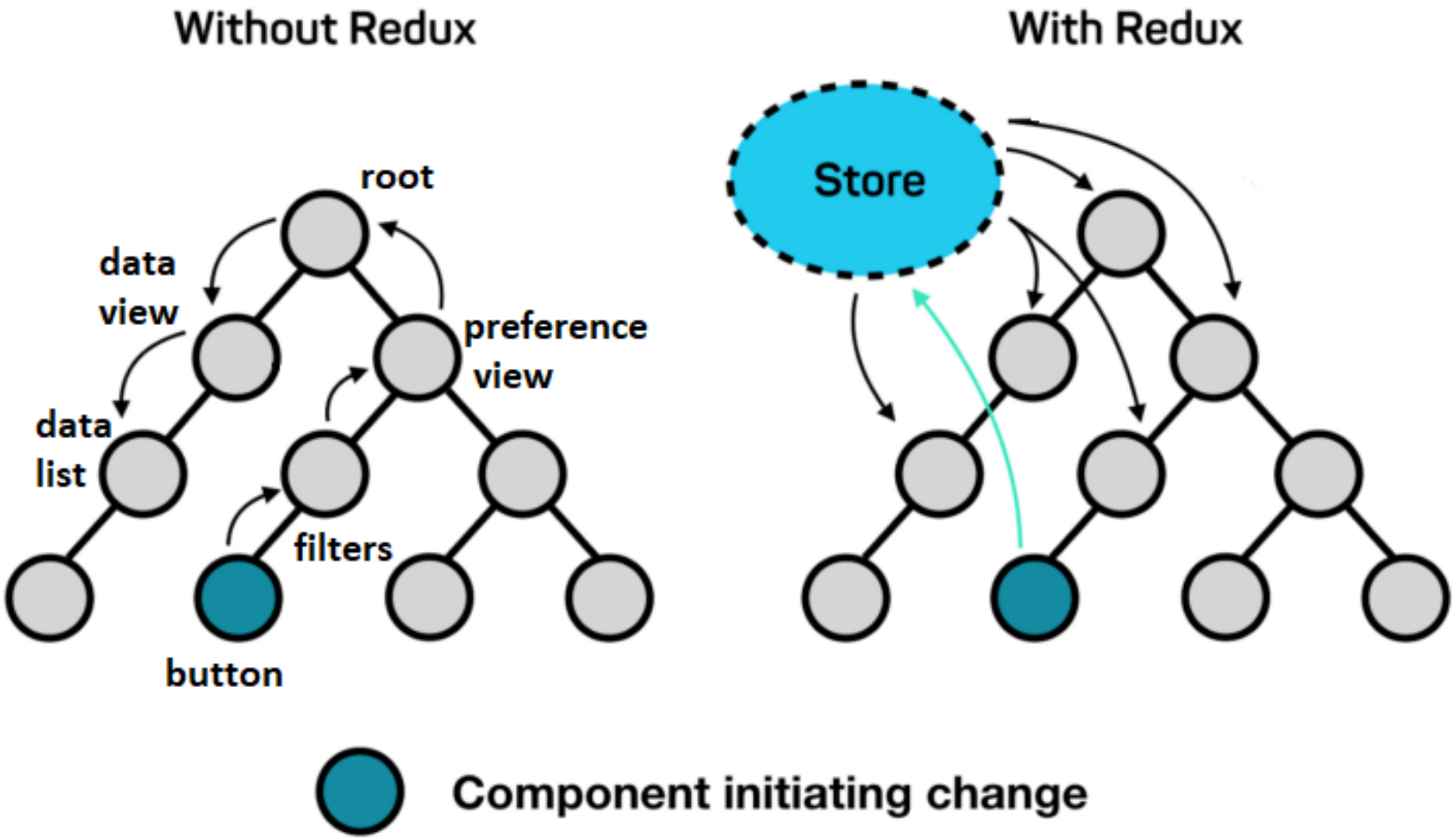
- **componentWillUnmount()**
  - „Sprząta”
    - Usuwa event listeners
    - Unieważnia żądania sieci
    - Czyści timeouts/intervals
  - Dobre miejsce na usunięcie subskrybcji

Expo i snack

- „Najszybszy sposób na zbudowanie aplikacji”
- Zestaw narzędzi przyspieszający i ułatwiający tworzenie aplikacji przy użyciu technologii React Native
- Snack - pozwala uruchomić React Native w przeglądarce

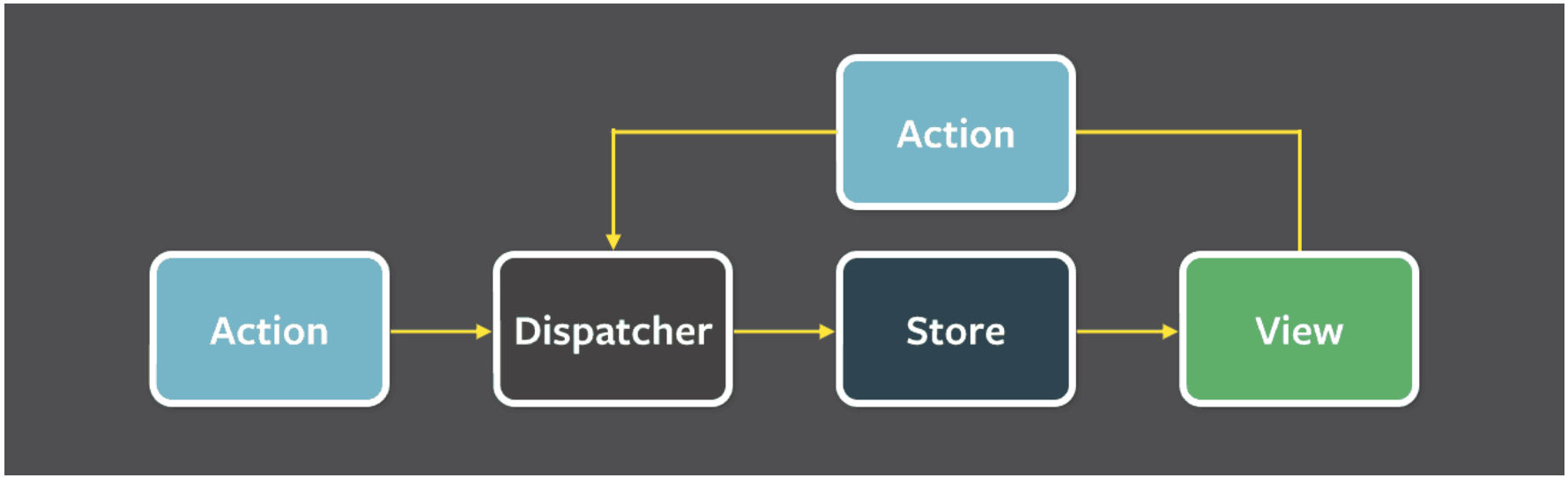
REDUX

Przepływ danych



Flux

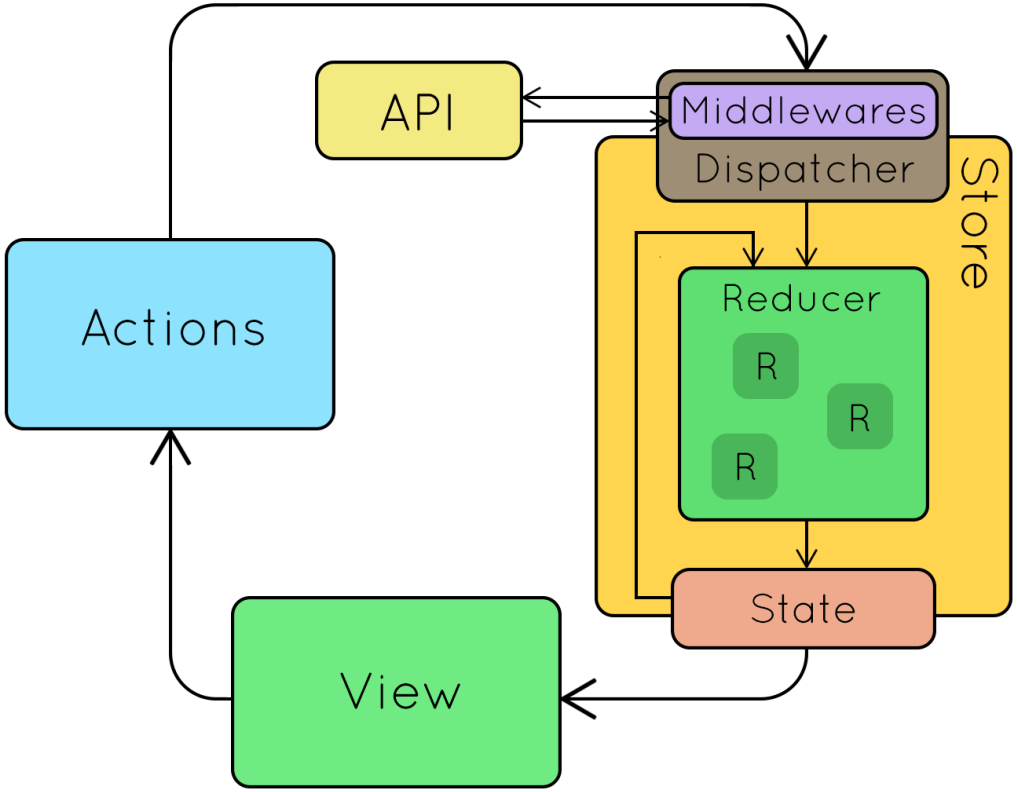
- Redux to implementacja architektury Flux
- **Dispatcher** – odpowiedzialny jest za odbieranie akcji i rozsyłanie ich do odpowiednich Store'ów
- **Store** – odpowiadają za przechowywanie informacji
- **View** – widok, źródło akcji
- informacje przepływają tylko w jednym, zawsze tym samym kierunku



3 główne zasady Reduxa

- „Single source of truth”
  - Stan aplikacji jest przechowywany w drzewie w jednym storze
- „State is read-only”
  - Nie można modyfikować stanu bezpośrednio, wysyłane są za to akcje, które są kolejno analizowane
- „Changes are made with pure functions”
  - Akcja wpływa na stan poprzez reducer
  - Reducer to „czysta funkcja”, która przyjmuje poprzedni stan i akcję, a następnie zwraca zupełnie nowy stan

Cykl działania Redux



Redux w praktyce

- Reducer
- Store
- Actions
- Container (metoda „connect”) + (Selector)
- Komponent

Reducer

```
import * as Actions from '../Actions/ActionTypes'

const CounterReducer = (state = {count: 0}, action) => {
    switch (action.type) {
        case Actions.COUNTER_INCREMENT:
            return Object.assign({}, state, {
                count: state.count + 1
            });
        case Actions.COUNTER_DECREMENT:
            return Object.assign({}, state, {
               count: state.count - 1
            });
        default:
            return state;
    }
}

export default CounterReducer;
```

Store

```
import { combineReducers, createStore } from 'redux';

import counterReducer from './CounterReducer'

const AppReducers = combineReducers({
    counterReducer,
});

const rootReducer = (state, action) => {
    return AppReducers(state,action);
}

let store = createStore(rootReducer);

export default store;
```

Actions

```
export const counterIncrement=()=>({
      type:Actions.COUNTER_INCREMENT
})
export const counterDecrement=()=>({
      type:Actions.COUNTER_DECREMENT
})
```

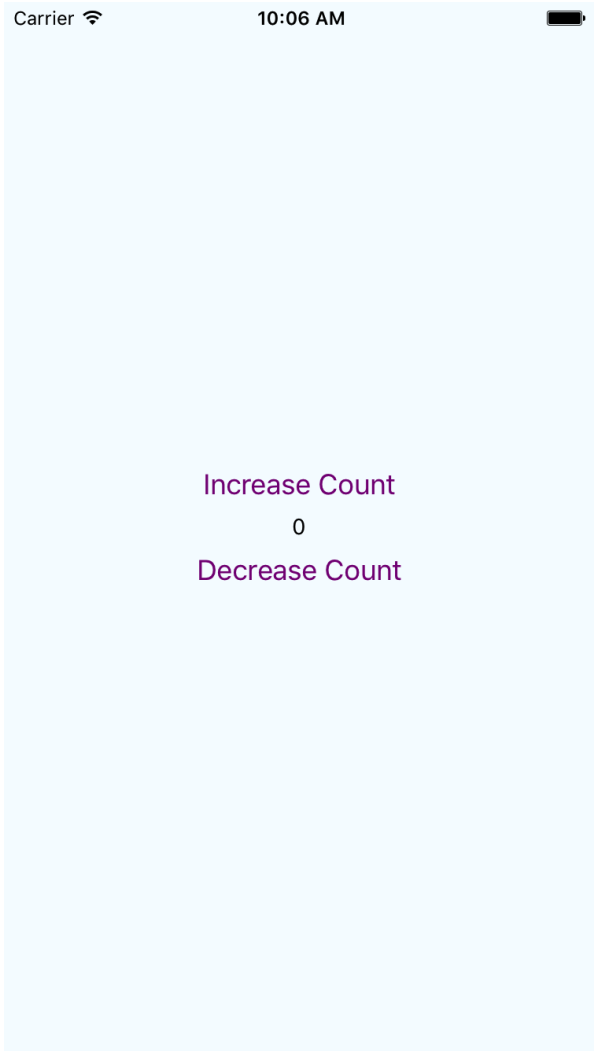
Container

```
const mapStateToProps = (state) => ({
     count: state.counterReducer.count
});

const mapDispatchToProps = (dispatch) => ({
    increment: () => dispatch(counterIncrement),
    decrement: () => dispatch(counterDecrement),
});

export default connect(mapStateToProps, mapDispatchToProps)(CounterComponent);
```

Komponent




```
export default class CounterApp extends Component {

    constructor(props) {
        super(props)
    }

    render() {
        return (
            <View style={styles.container}>
                <Button
                    onPress={this.props.increment}
                    title="Increase Count"
                    color="#841584"
                    accessibilityLabel="Increase Count"
                />

                <Text>{this.props.count}</Text>

                <Button
                    onPress={this.props.decrement}
                    title="Decrease Count"
                    color="#841584"
                    accessibilityLabel="Decrease Count"
                />
            </View>
        );
    }
}
```

PODSUMOWANIE

DZIĘKUJEMY ZA UWAGĘ