# REACT.JS UND REDUX WORKSHOP

Maximilian Schempp (schempp.xyz)

24.05.2017 - Entwicklertag Karlsruhe - Tutorial Day

# ZEITPLAN

- 09:00 Uhr: Beginn
- Bis 11:00 Uhr: React.js Präsentation + Übung
- 11:00 - 12:00 Uhr: Redux Präsentation + Übung
- 12:00 - 13:00 (12:30) Uhr: Pause
- Ab 13:00 (12:30) Uhr: Redux Übung

- Kleinere, ~10 Minuten Pausen nach Gefühl :)

- Ende: c.a. 14:00 - 15:00 Uhr - je nach Lust und Laune

# WAS IST REACT?

*javascript library for building user interfaces*

*Declarative*

*Component-Based*

# WAS IST REACT?

```
// funktionale Komponente
const Fancy = ({name}) => <p>FANCY {name}</p>

class App extends React.Component {
  // lifecycle hooks

  render() {
    return (<div>
                <Fancy name="Bob" />
                <Fancy name="Alice" />
            </div>);
  }
}

ReactDOM.render(<App />, document.getElementById("some-id"));
```

```
<div id="some-id"></div> // Anwendung wird in diesem div gerendert
<script src="bundle.js"></script> // Anwendung wird geladen
```

# JSX (JAVASCRIPT SYNTAX EXTENSION)

syntactic sugar:

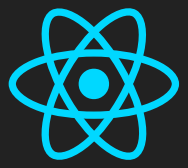
```
const A = <MyButton color="blue" shadowSize={2}>Click Me</MyButton>
const B = <MyButton color="blue" shadowSize={2} />
```

wird zu

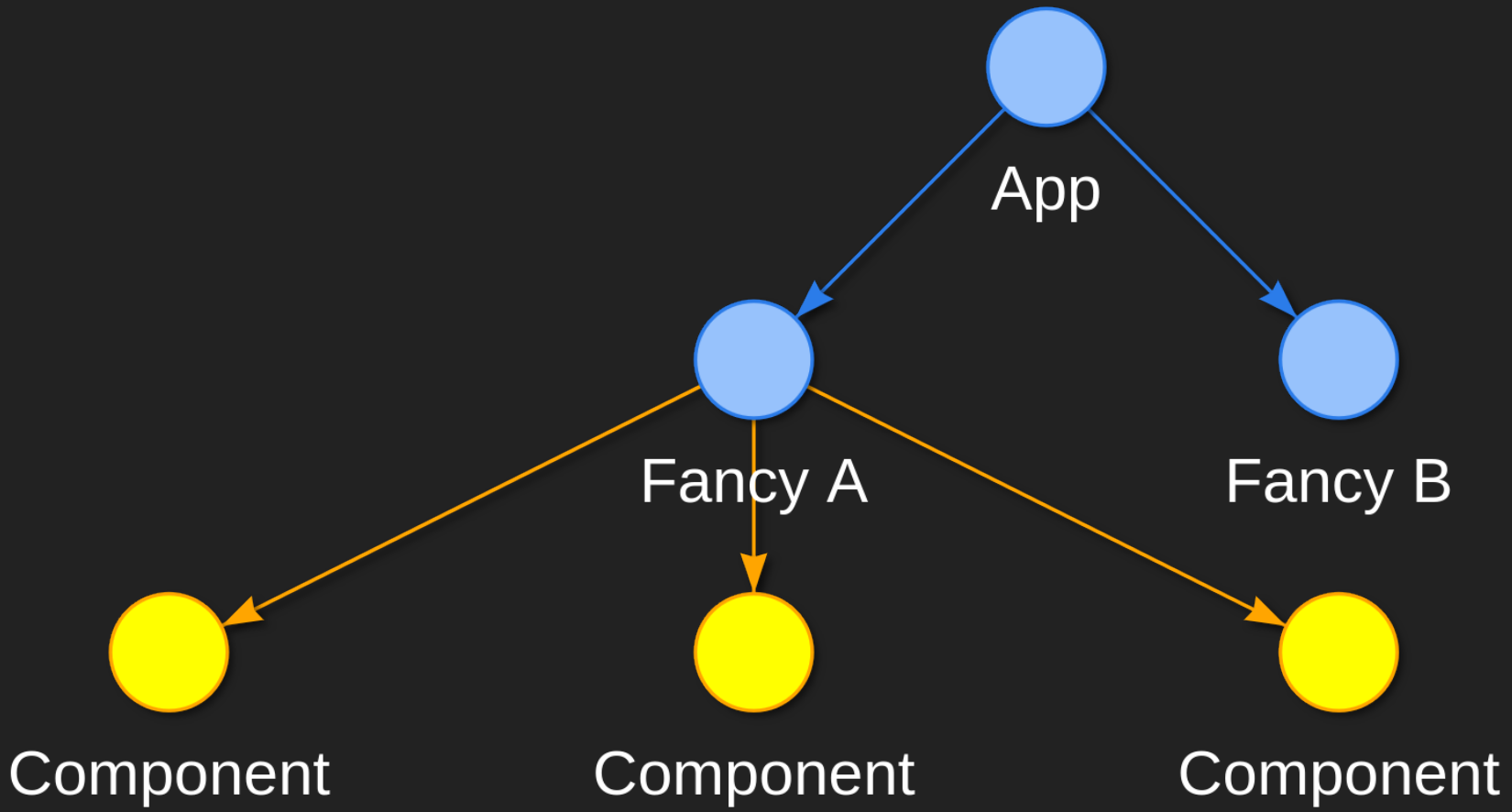
```
const A = React.createElement(
  MyButton, // element name, e.g. 'div', 'span', ...
  {color: 'blue', shadowSize: 2}, // properties
  'Click Me' // children
)

const B = React.createElement(
  MyButton, // element name
  {color: 'blue', shadowSize: 2}, // properties
  null)
```

# WAS IST REACT?



## HIERARCHISCH STRUKTURIERT


App
Fancy A
Fancy B
Component
Component
Component

# REACT - DO IT YOURSELF 

https://github.com/andrena/react-redux-workshop

## // TODO:

> $ git clone
> $ cat README.md
> $ git checkout stage-1
> $ cat STAGE-1.md
> $ have-fun

# REACT.JS CHEAT SHEET

## Funktionale Komponente

```
const Foo = ({propertyA, propertyB}) => <div>{propertyA}</div>
```

## Klassenbasierte Komponente

```
class Bar extends React.Component {
    // https://facebook.github.io/react/docs/react-component.html
    constructor() {
        this.state = {
            barbar: 'test'
        }
    }

    render() {
        return (
            <div>
                {this.props.propertyA}
                {this.state.barbar}
            </div>)
    }
}
```

# WAS IST REDUX?

*Redux is a predictable **state** container for JavaScript apps.*

*It helps you write applications that behave consistently, run in different environments (client, server, and native), and are easy to test.*

# AUS WAS BESTEHT REDUX?

- store
- actions
- reducers
- (selectors)

# REDUX STORE

*A store holds the whole state tree of your application. The only way to change the state inside it is to dispatch an action on it.*

```
import { createStore } from 'redux'
import reducer from './myReducer'

// reducer is needed to create the store
const store = createStore(reducer)
```

Wichtige Methoden:

```
const state = store.getState() // the whole state-tree
store.dispatch(action)
```

# ACTIONS 

*An action is a plain object that represents an intention to change the state. Actions are the only way to get data into the store.*

```
const action = {
    type: 'MY_ACTION_TYPE', // a pattern, but not necessary
    ... // anything you want
}
```

Pattern action-creator:

```
const doStuff = (stuff) => ({type: 'DO_STUFF_ACTION', data: stuff})
```

# REDUCERS

*A **reducer** is a function that accepts a state and a value and returns a new state.*

***Actions** describe the fact that something happened, but don't specify how the application's state changes in response. This is the job of **reducers**.*

# REDUCERS 

Reducer **müssen** immer einen **neuen** State zurück geben.
Wann immer eine Action 'dispatched' wird, erfahren **alle** reducer davon und können entsprechend reagieren

```
const someReducer = (state = initialState, action) => {
  switch (action.type) {
    case SOME_ACTION_TYPE:
      return Object.assign({}, state, {message: 'bar'})
    case SOME_OTHER_ACTION_TYPE:
      return Object.assign({}, state, {info: 'foo'})
    default:
      return state
  }
}
```

# SELECTORS 

Actions: Daten **in** den store schreiben (setter).

Selectors: Daten **aus** dem store beziehen (getter).

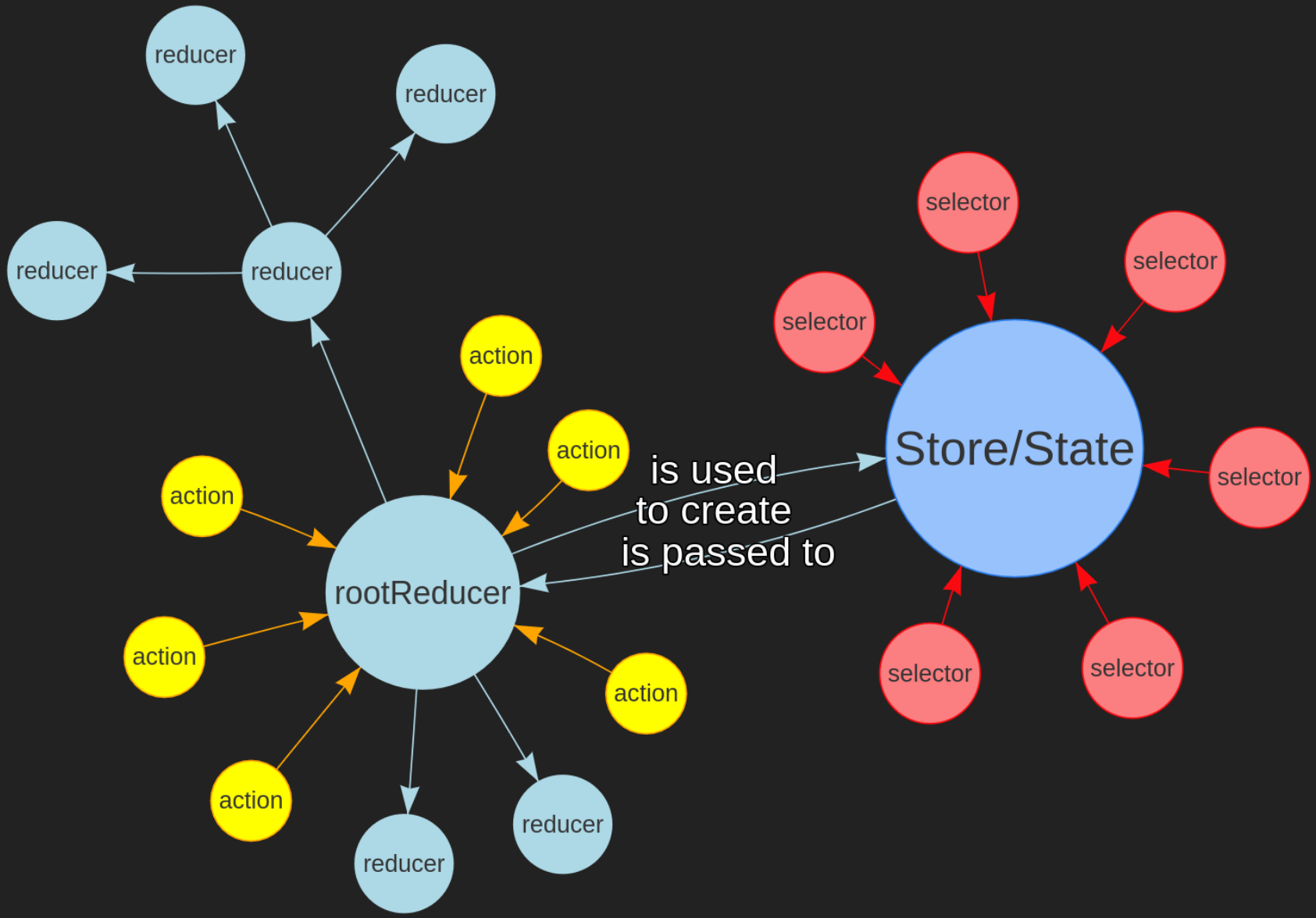
```
const getStuff = (state) => state.myStuff
const getMyStuff = (state, stuffType) => state.myStuff[stuffType]
```

# REDUCERS KOMBINIEREN

- Jeder Reducer verwaltet seinen eigenen State
- Reducer kennen sich nicht gegenseitig
- Reducer können zusammen geführt werden:

```
const rootReducer = combineReducers({
    theDefaultReducer,
    firstNamedReducer,
    combineReducers({
        subReducer,
        anotherReducer,
        theLastReducer
    })
})
```

# ZUSAMMENSPIEL



# REACT UND REDUX  

https://github.com/reactjs/react-redux

## React Komponente: `<Provider store>`

```
const ReduxApp = <Provider store={store}><App /></Provider>
```

## Funktion (Higher Order Komponente): `connect(...)`

```
const FooComponent = ({myStuff, doStuff}) => ...

const mapStateToProps = (state) => ({myStuff: state.stuff.myStuff})
const mapActionsToProps = {
    // wird zu (...args) => dispatch(actions.doStuff(...args))
    doStuff: actions.doStuff
}

const connector = connect(mapStateToProps, mapActionsToProps)

// connector injects myStuff and doStuff into FooComponent
export default connector(FooComponent)
```

# REDUX - DO IT YOURSELF 

https://github.com/andrena/react-redux-workshop

## // TODO:

> $ git checkout stage-2
> $ cat STAGE-2.md
> $ have-fun

# REDUX CHEAT SHEET

```
import {Provider} from 'react-redux'
import {createStore} from 'redux'
import reducer from './reducer'

const store = createStore(reducer)
ReactDOM.render(
    <Provider store={store}><App /></Provider>,
    document.getElementById('root'))
```

```
const Foo = ({myProp}) => ...

const mapStateToProps = (state) => ({
    myStuff: state.stuff.myStuff
})

const mapActionsToProps = {
    someAction: actions.someAction
}

export default connect(mapStateToProps, mapActionsToProps)(Foo)
```

# ZUSAMMENFASSUNG

- React.js ist eine javascript user-interface library, kein Framework
- React.js ermöglicht es wiederverwendbare Komponenten zu schreiben - komplett in javascript
- Redux ist eine javascript state container library, die es ermöglicht den Zustand der Anwendung zu verwalten
- Beide libraries fügen sich sehr gut in den neuen ES6-Standart ein
- Durch die Entwicklertools für React.js und Redux wird das Entwickeln nochmal deutlich vereinfacht

> $ start-workshop
> doing workshop...
> ...done

> questions? [Y\n]
> feedback? [Y\n]

> thank you for your participation :)