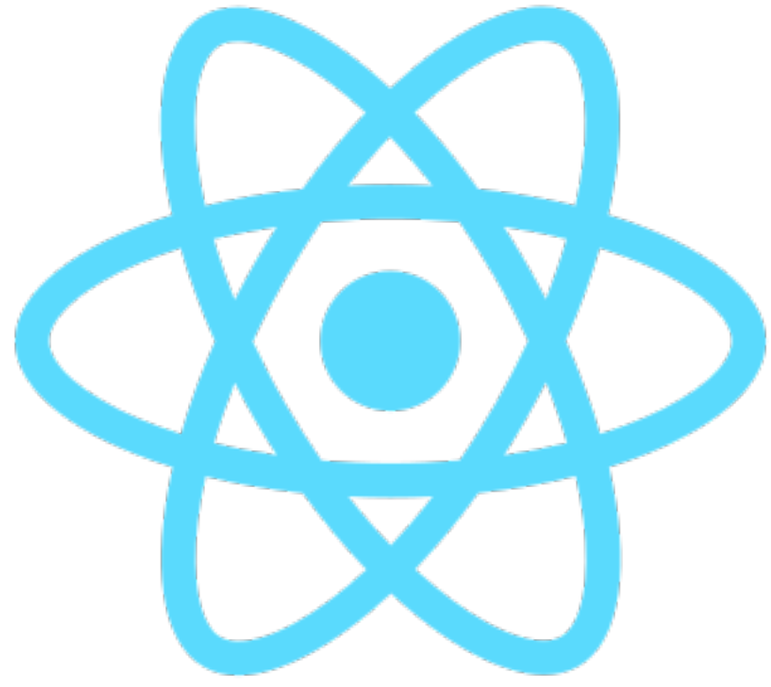
Introduction ReactJS

ReactJS

- From Facebook
- Open Source Library
- Public since 03/13



ReactJS

- View-Layer to build User-Interfaces

- Library not a Framework

Use Cases

- Stateful Single Page Applications
- Render DOM (HTML) on Server-Side
- Transpile to Native Apps
- To build your own UI-Component-Framework
- Highly User-Interaction / Real-Time Apps

Fits perfectly for …

- Adaptive integration into legacy Web-Apps
- Integration into custom Web-Stacks
- Universal Code-Base for Web/Mobile/Desktop
- Decoupled Client / Server

Unfits for …

- Not well architectured frontend Web-Apps
- Low-Budget / Quick & Dirty Web-Apps
- Very small Web-Apps / Web-Sites
- Need a All-in-One UI-Framework

ReactJS Architecture

- It’s all about UI-Components
- Component internal UI-State-Management
- Has a hierarchical component structure like HTML (XML)
- Stateful & Stateless UI-Components
- One-Way **react**ive data flow
    - Immutable-Data passing down with props
    - Lifting UI-State changes up by calling props functions
- Abstracting DOM with in-memory Virtual-DOM
    - Rerending only when internal state change is triggered

JavaScript **S**yntax E**X**tensions

- No Template-Engine - It's **JavaScript**
- Declarative UI approach
- First-Class-Citizen in ReactJS
- Translation-Pipeline
  - JSX -> JavaScript (functions) -> Virtual-DOM -> DOM
- Mental Model is view = f(props, state)

JSX

```
import React from 'react';

const todos = ['Learn', ,Code'];

const TodoList = () =>
  <ul>
    {
      todos.map((todo, i) => <li key={i}>{todo}</li>)
    }
  </ul>

export default TodoList
```

JSX

```
{
  type: 'ul',
  props: {
    children: [{
      type: 'li',
      props: {
        children: 'Learning'
      }
    }, /* ... */]
  }
}
```

```
<ul>
  <li>Learning</li>
  <li>Coding</li>
</ul>
```

First ReactJS App

$ yarn create vite

$ cd my-first-app

$ yarn

$ yarn dev

Modern JavaScript

- Arrow Functions (Lambdas) [() **=>** ({})]
- Promise [new **Promise**(resolve, reject) => {})]
- Block Scoping [**const**, **let**]
- String templates [`foo ${bar}`]
- Object literal enhancements [{foo, **do() {}**}]
- Classes {class Foo intends Bar {}}
- Module imports / exports [**export** default, **import** Foo from './foo']
- Destructing [const **{a, b}** = this.props]
- Function Parameters
    - default [function foo (a **= 'Demo'**) {}]
    - rest [function foo (a, **...rest**) {}]
    - spread [foo(**...['D', 'B', 'C']**)]

More JavaScript

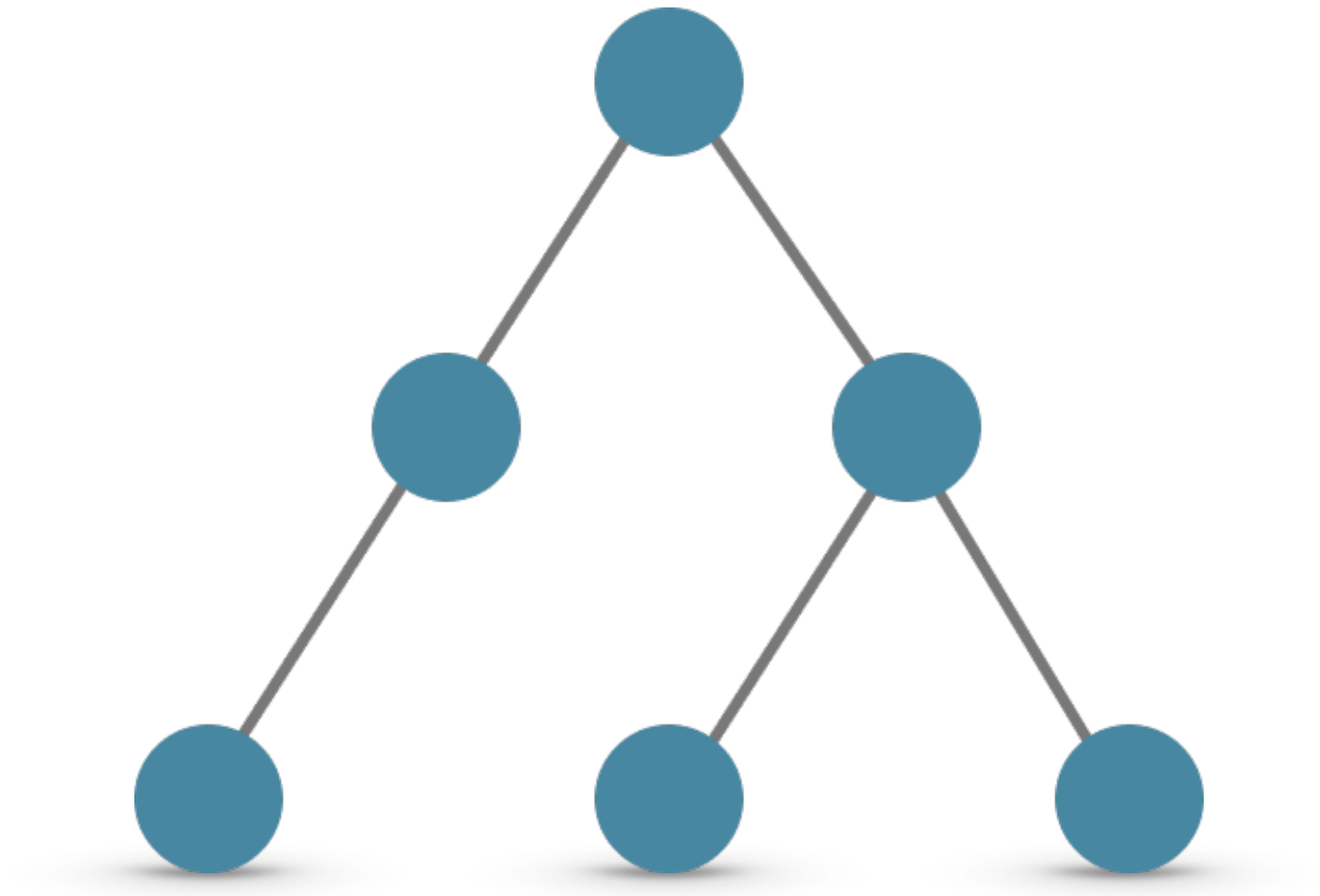
- Generators [function* generation () {}]
- Iterations [for (const i **of** generation()) {}]
- Async/Await [**async** function () { return **await** do() }]

It’s all about Components

Components

- Large Apps are composed of many small components
- loose coupled components are
  - composable
  - reusable
  - isolated
  - testable

Hierarchical structure



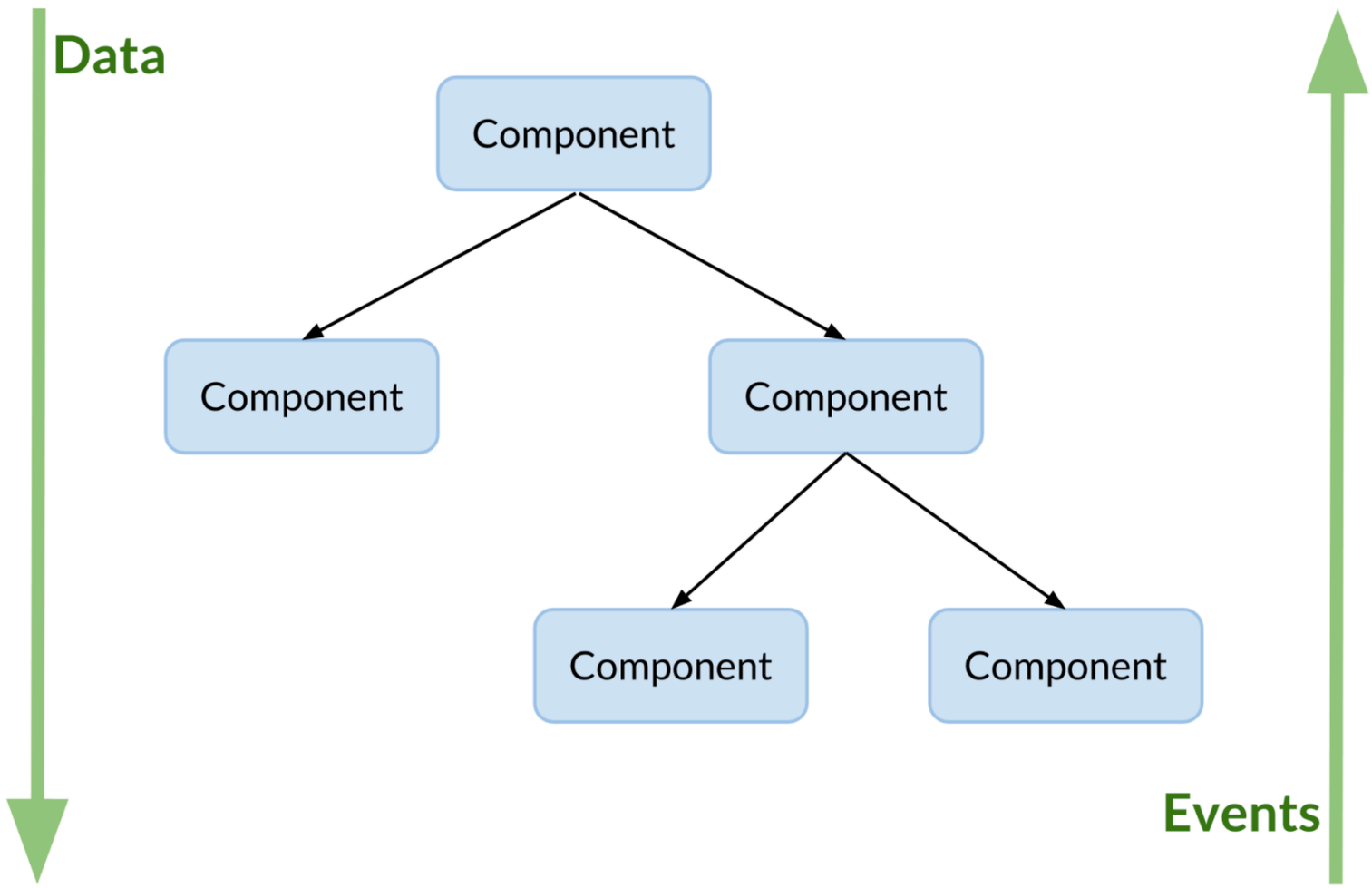
Components

- **Co-location** of UI representation and behavior
- **High cohesion** - group what changes together
- **Autonomous** - Isolation to maximize reusability

Unidirectional Data Flow

- Receive props from parent component as immutable data
- Immutable data promotes loose coupling
    - Data flows down via **props**
    - Events flows up via callback **functions**

Unidirectional Data Flow



Unidirectional Data Flow

```
import React from 'react';

export const TodoList = props =>
  <ul>
    {
      props.todos.map(todo => <li>{todo}</li>)
    }
  </ul>

export default TodoList
```

Unidirectional Data Flow

```
import React, { Component } from 'react';
import TodoList from './TodoList'

const todos = ['Learn', 'Code'];

class App extends Component {
  render() {
    return <TodoList todos={todos} />
  }
}

export default App;
```

Explicit
State Mutations

- „dumb“ components are pure functions
- „smart“ components have mutable local state
- state change only through an explicit interface

```
this.state.todos = ['Learning'] // WRONG

// CORRECT
this.setState({
  todos: ['Learning']
})
```

Explicit
State Mutations

- ReactJS doesn’t do dirty checking
- No DOM handling (e.g. add, remove, update DOM nodes)
- only state changes triggers re-render of the component

DOM update in depth

- Build a new virtual DOM subtree
- Diff it with the previous one
- Compute necessary mutations
- Batch-execute all mutations

Build & Dev Tools

Remote Data Access

Fetch URL and apply data to state

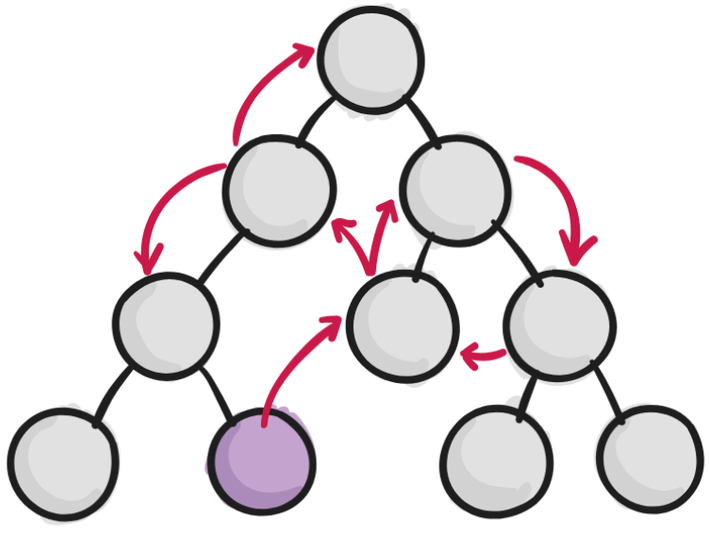
```
componentDidMount() {
   fetch('http://localhost:8080/posts')
   .then(response => response.json())
   .then(data => this.setState({content: data.content}))
   .catch(error => this.setState({error: error.message}))
}

// Similar to componentDidMount and componentDidUpdate:
useEffect(() => {
  fetch('http://localhost:8080/posts')
    .then(response => response.json())
    .then(data => this.setState({content: data.content}))
    .catch(error => this.setState({error: error.message}))
});
```

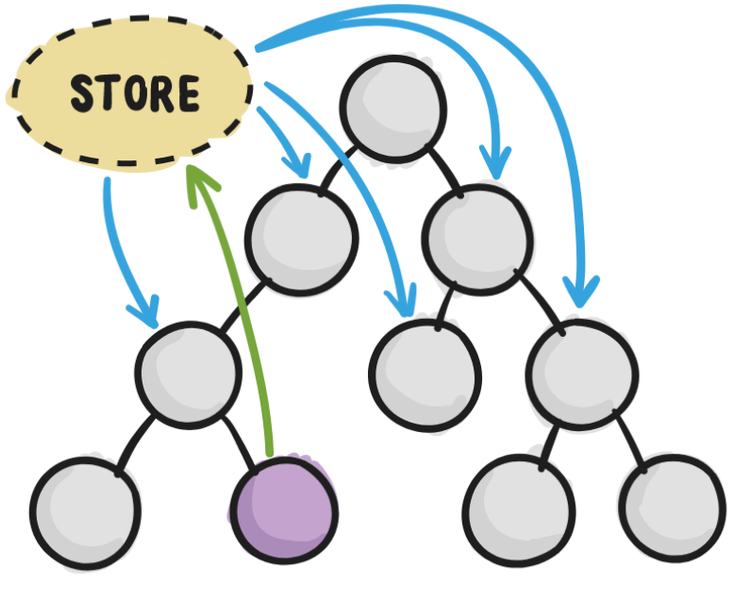
State Management

- „Smart“ Parent & „Dumb“ Children
- Central-State with Context / Reducer Pattern

without Context/Reducer

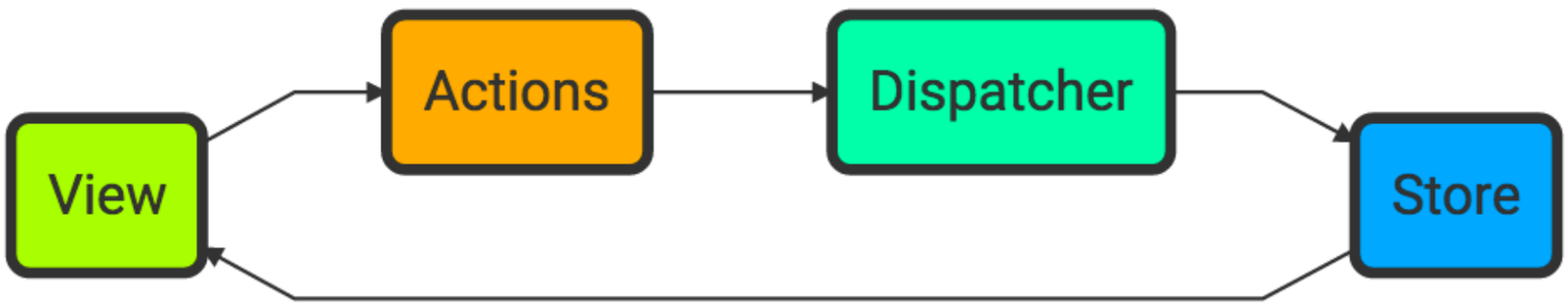


with Context/Reducer

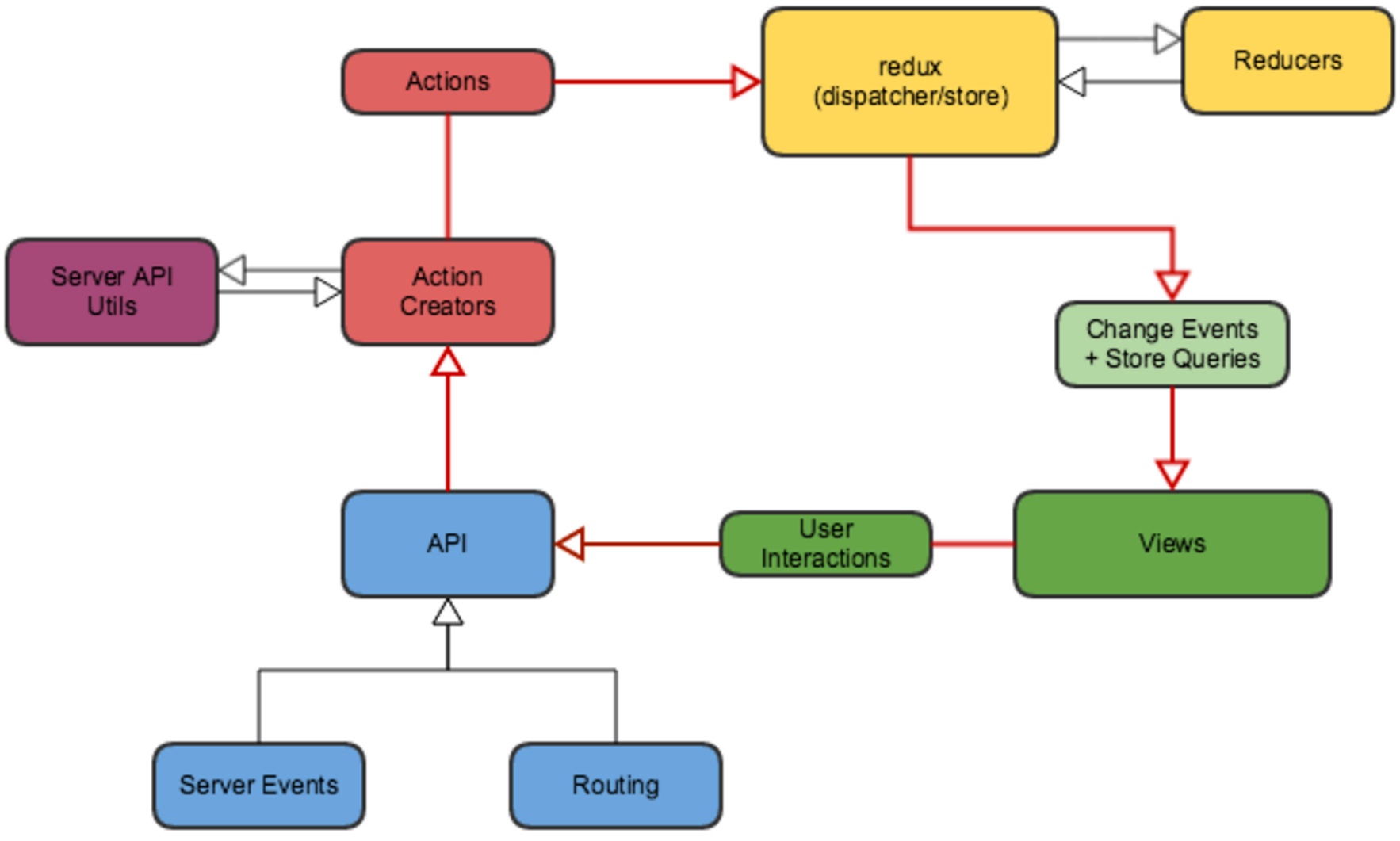


Redux

- „Single Source of Truth“ for the hole app
- State is Read-Only
- Changes are made with pure functions



Redux/Flux Architecture



Redux

```
// Actions
export const UPDATE_GREETING = 'UPDATE_GREETING'

// Action creators
export function updateGreeting(greeting) {
  return {
    type: UPDATE_GREETING,
    greeting,
  }
}

// Reducer
export function greetingReducer(state = 'World', action) {
  switch (action.type) {
    case UPDATE_GREETING:
      return action.greeting
    default:
      return state
  }
}
```

React-Redux

```
import {Provider} from 'react-redux'
import {createStore, combineReducers} from 'redux'

const store = createStore(combineReducers({
  greeting: greetingReducer,
}))

ReactDOM.render(
  <Provider store={store}>
    <div>
      <HelloWorld />
    </div>
  </Provider>,
  document.getElementById('root')
)
```

more ReactJS

- React-Router - Map URLs to components
- Recompose - Compose components to Higher-Order-Components
- RxJS - Reactive Extensions
- GraphQL - Declarative Data Fetching APIs
- React-Native - Transpile to Mobile/Desktop
- React-Bootstrap - Bootstrap UI
- React-Material - Material UI

more JS Tools

- mocha - test runner
- nodemon - watch for changes
- es-lint - style-code and syntax checking
- prettier - lintish syntax formatter
- pm2 / forever - process supervisor