Angular with Redux

Manuel Mauky
@manuel_mauky

JUG Görlitz

Saxonia Systems
So geht Software.

Manuel Mauky

@manuel_mauky
www.lestard.eu
github.com/lestard

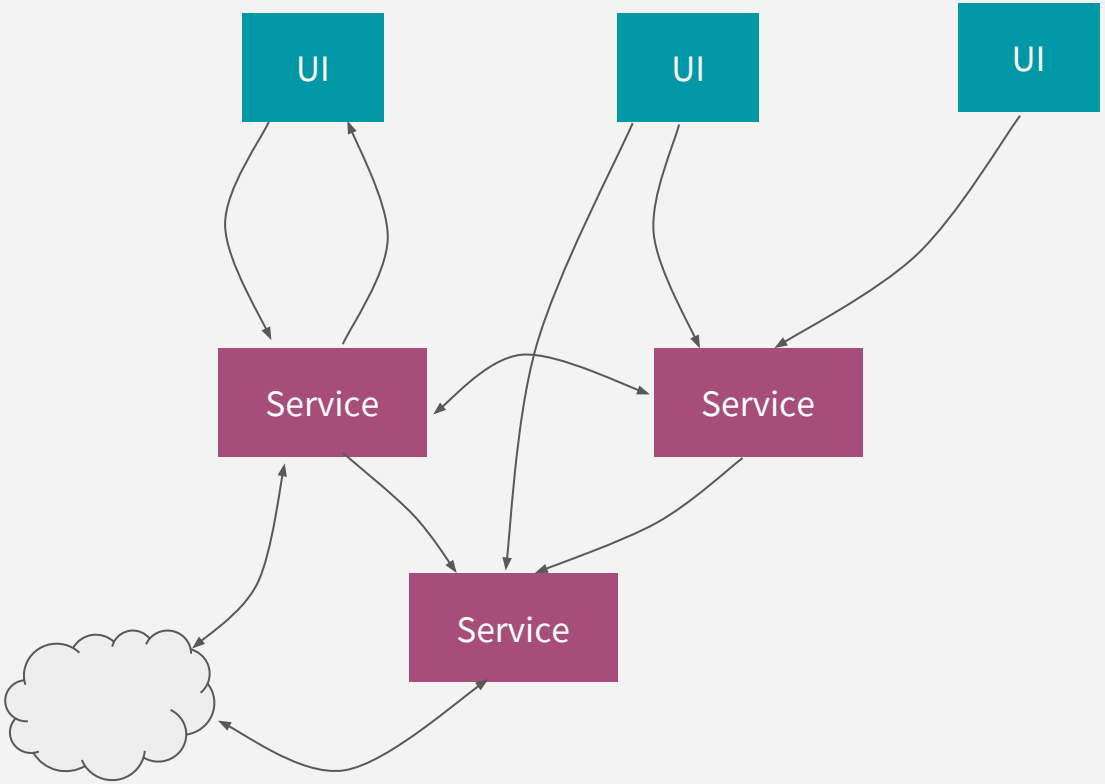
Saxonia Systems
So geht Software.

JUG Görlitz

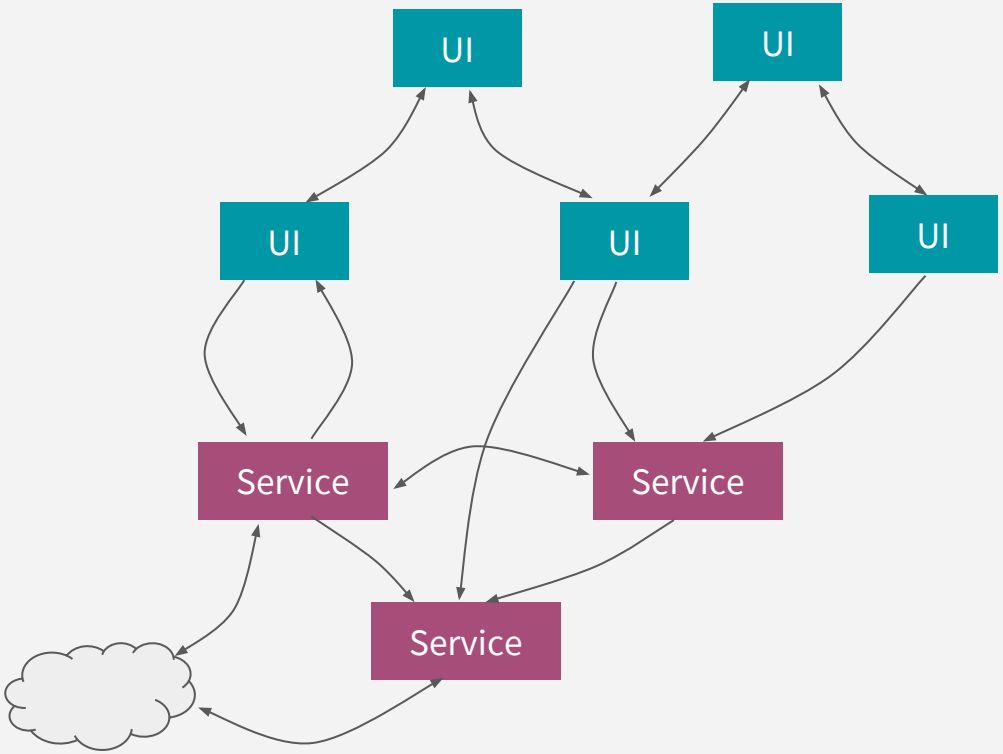
Why Redux?

What can go wrong with pure Angular?

client logic + dataflow with Angular



client logic + dataflow with Angular



- What is the current state of my app?
- Where are the parts of the state located?
- Why is the state the way it is?
- How did it happen?
- Something is wrong → Where do I have to look for the bug?

- What is the current state of my app?
- Where are the parts of the state located?
- Why is the state the way it is?
- How did it happen?
- Something is wrong → Where do I have to look for the bug?
- Object Orientation:
    - Combine "State" and "Behaviour"

- What is the current state of my app?
- Where are the parts of the state located?
- Why is the state the way it is?
- How did it happen?
- Something is wrong → Where do I have to look for the bug?
- Object Orientation:
    - ~~Combine~~ **Mix up** "State" and "Behaviour"

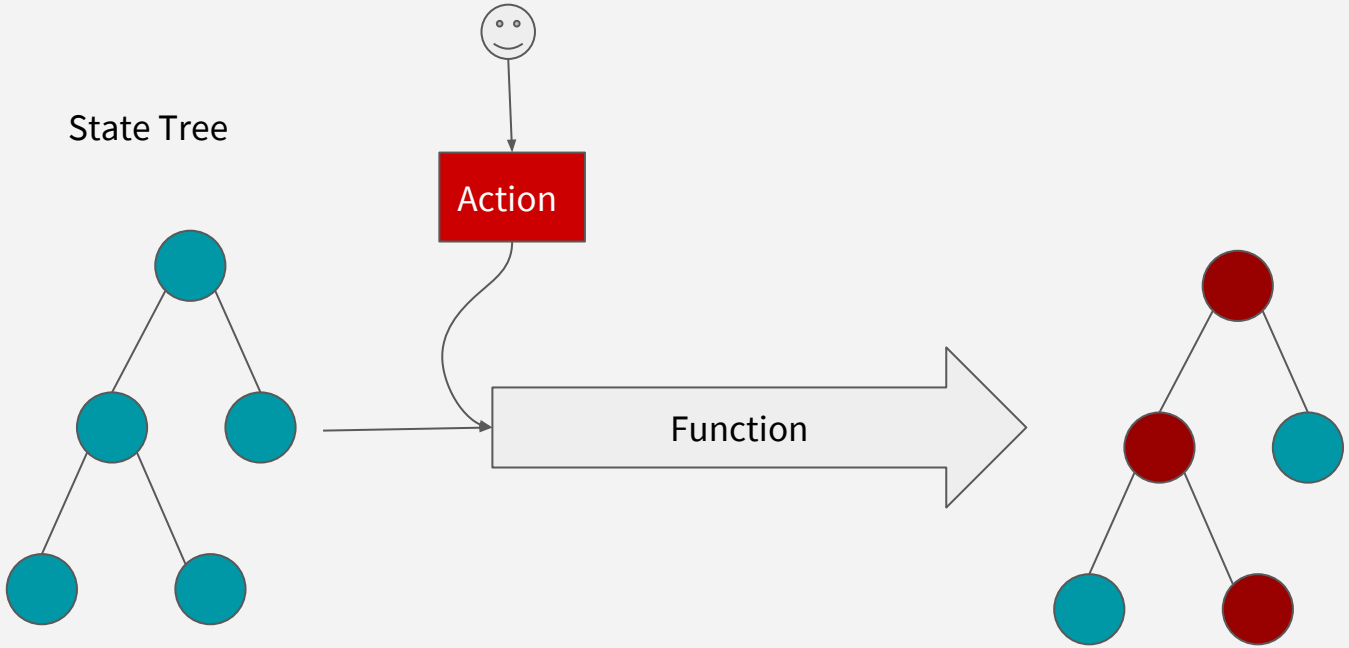
- What is the current state of my app?
- Where are the parts of the state located?
- Why is the state the way it is?
- How did it happen?
- Something is wrong → Where do I have to look for the bug?
- Object Orientation:
  - ~~Combine~~ **Mix up** "State" and "Behaviour"
  - time is implicit

Redux

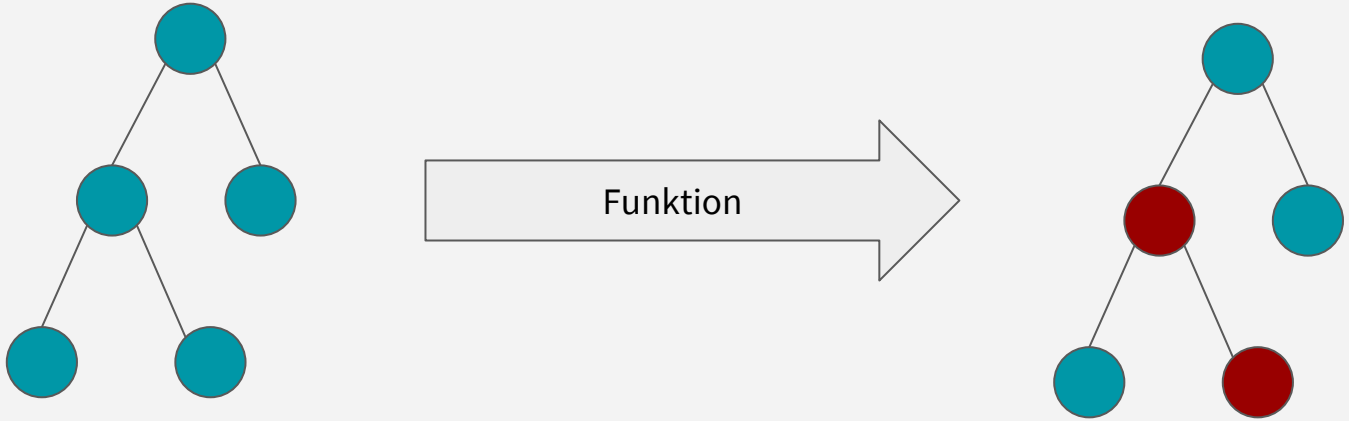
A functional approach to state-management

Functional Programming

- Pure Functions (no side-effects)
- Immutable Data (create new data instead of modifing existing data)


State Tree
Action
Function

Zustandsbaum



Zustands-Automat mit Überführungsfunktion

Where does Redux come from?
How does it work?

Before Redux: **Flux-Architecture**

Flux is a frontend architecture pattern (Alternative to MVC*)

Redux is a derivative of **Flux** + Implementation

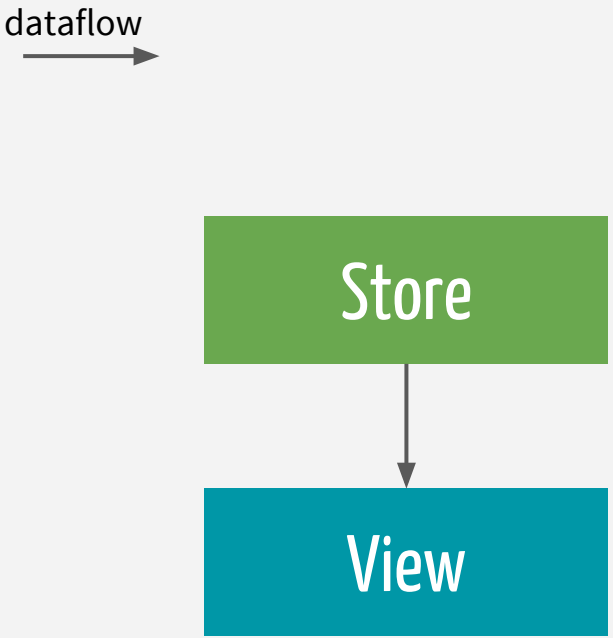
Flux is object-oriented + some functional ideas

Redux is (almost) fully functional

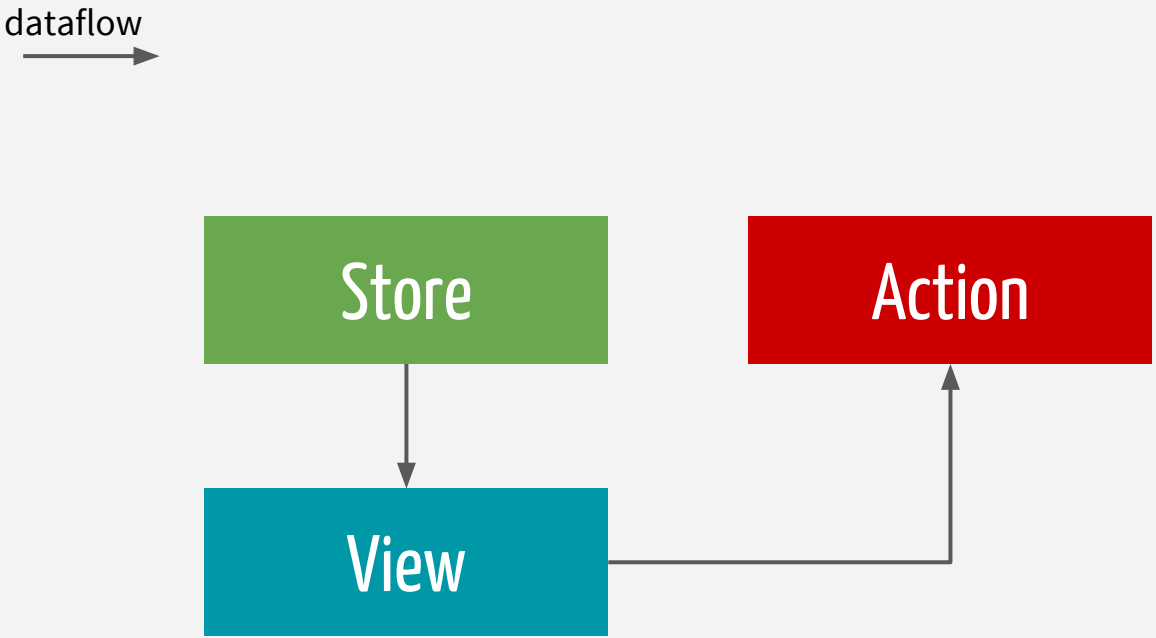
How does Flux work?

Store

- application state
- logic
- represents a business domain



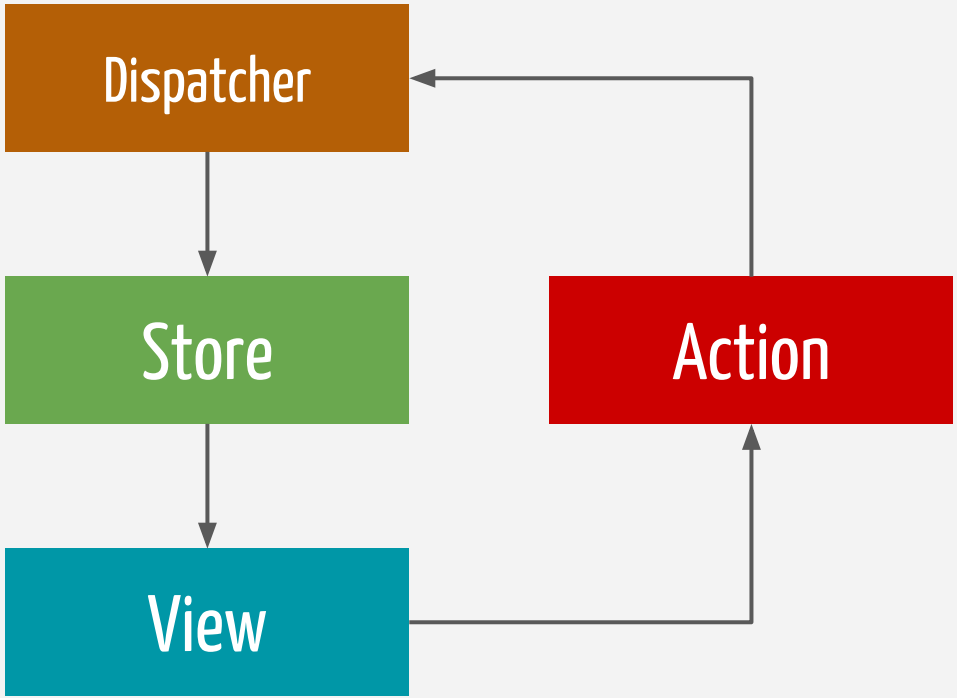
- View shows data of one or more Stores
- when the Store updates the View will update itself too



- View creates “Actions” based on user interaction
- an Action represents a business action
- comparable to *command pattern*

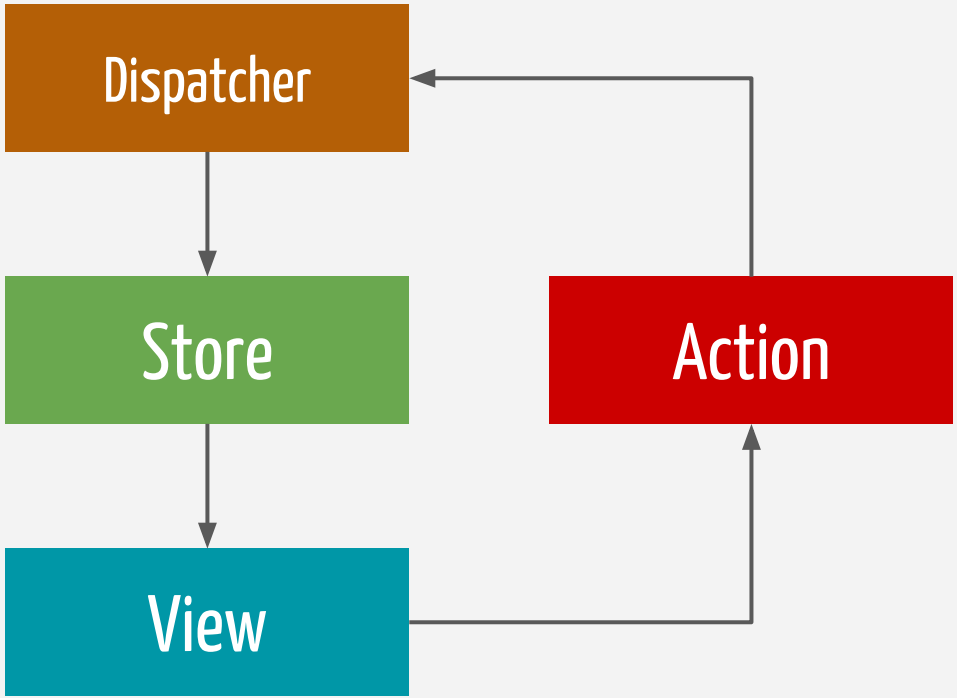
Example: Action

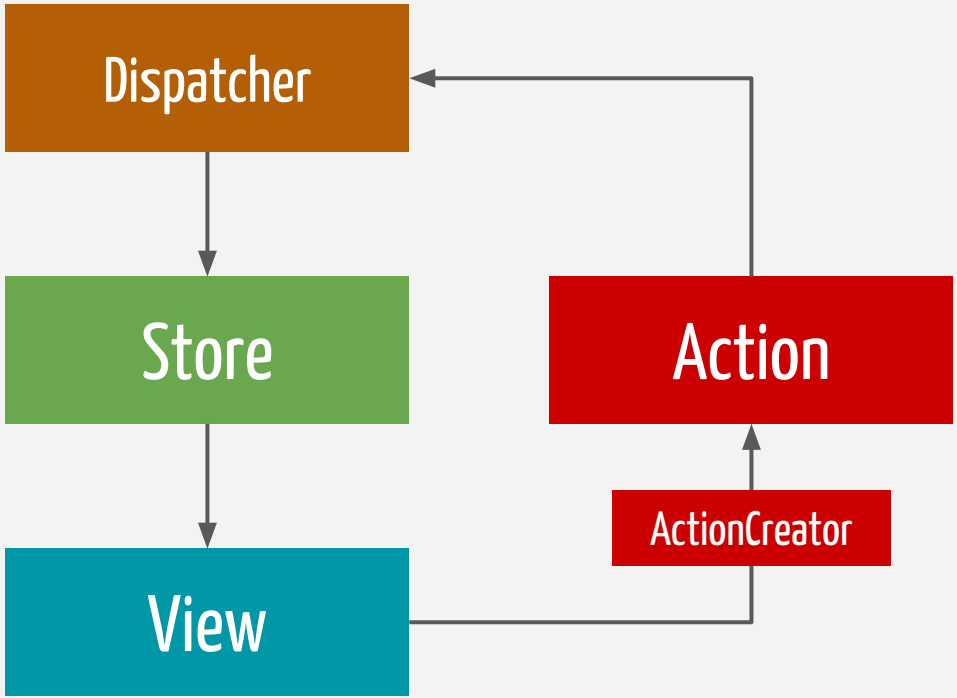
```
{
    type: "CREATE_USER_ACTION",
    payload: {
        username: "Luise",
        email: "luise@example.org"
    }
}
```



- Dispatcher takes **all** Actions and passes them to **all** Stores
- Stores decide on their own if and how they like to react to Actions

Single Directional Dataflow



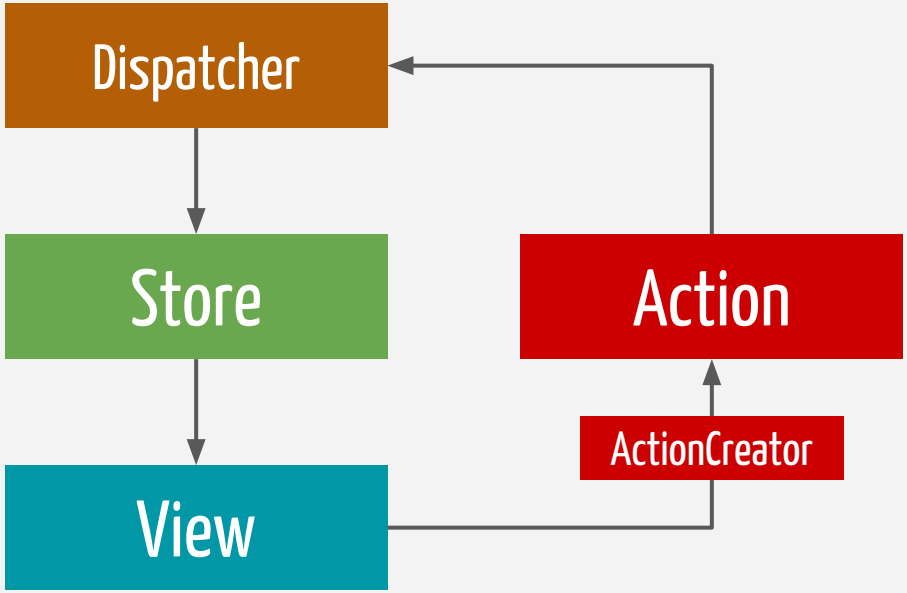


- a function that creates Actions
- decouples the View from the actual creation of Actions

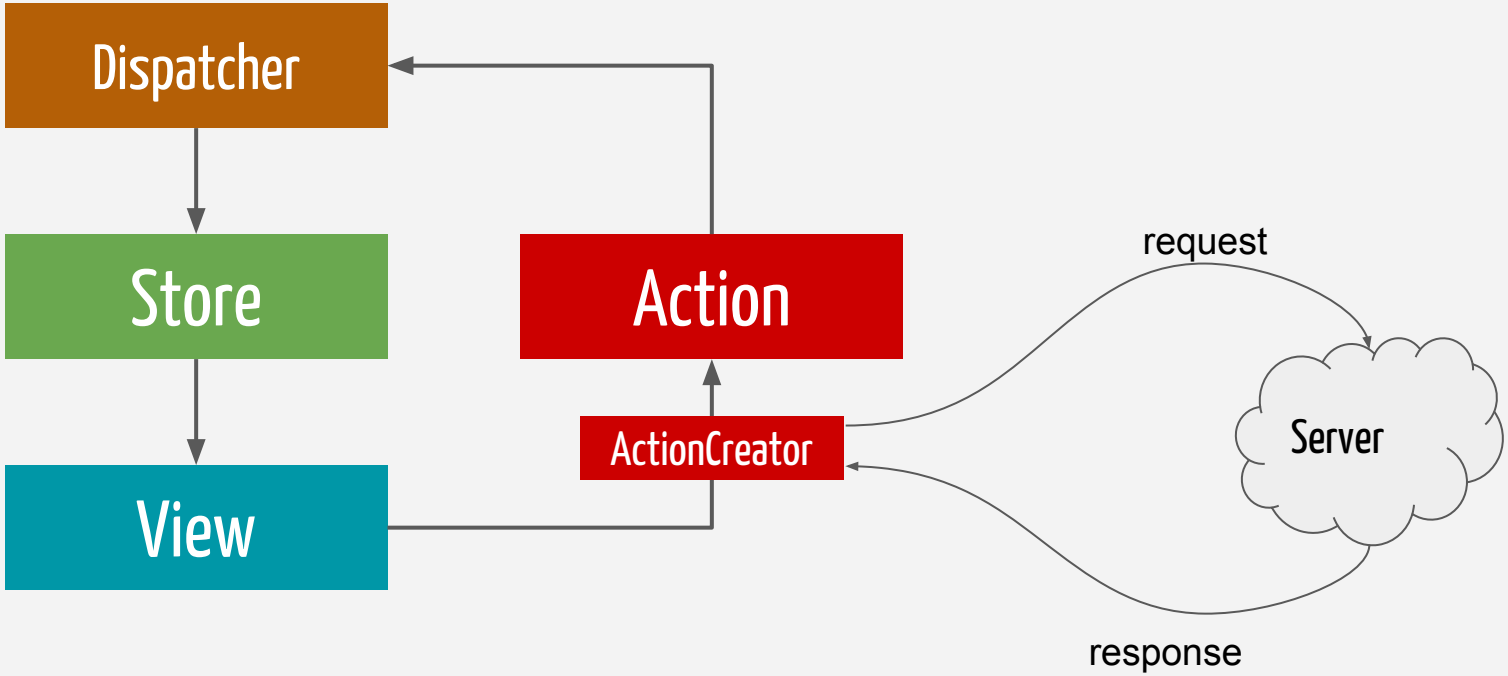
Example: ActionCreator

```
const createUser = (username, email) => {
    dispatch({
        type: "CREATE_USER_ACTION",
        payload: {
            username: username,
            email: email
        }
    });
};
```

How to do async operations? REST-Requests?



How to do async operations? REST-Requests?

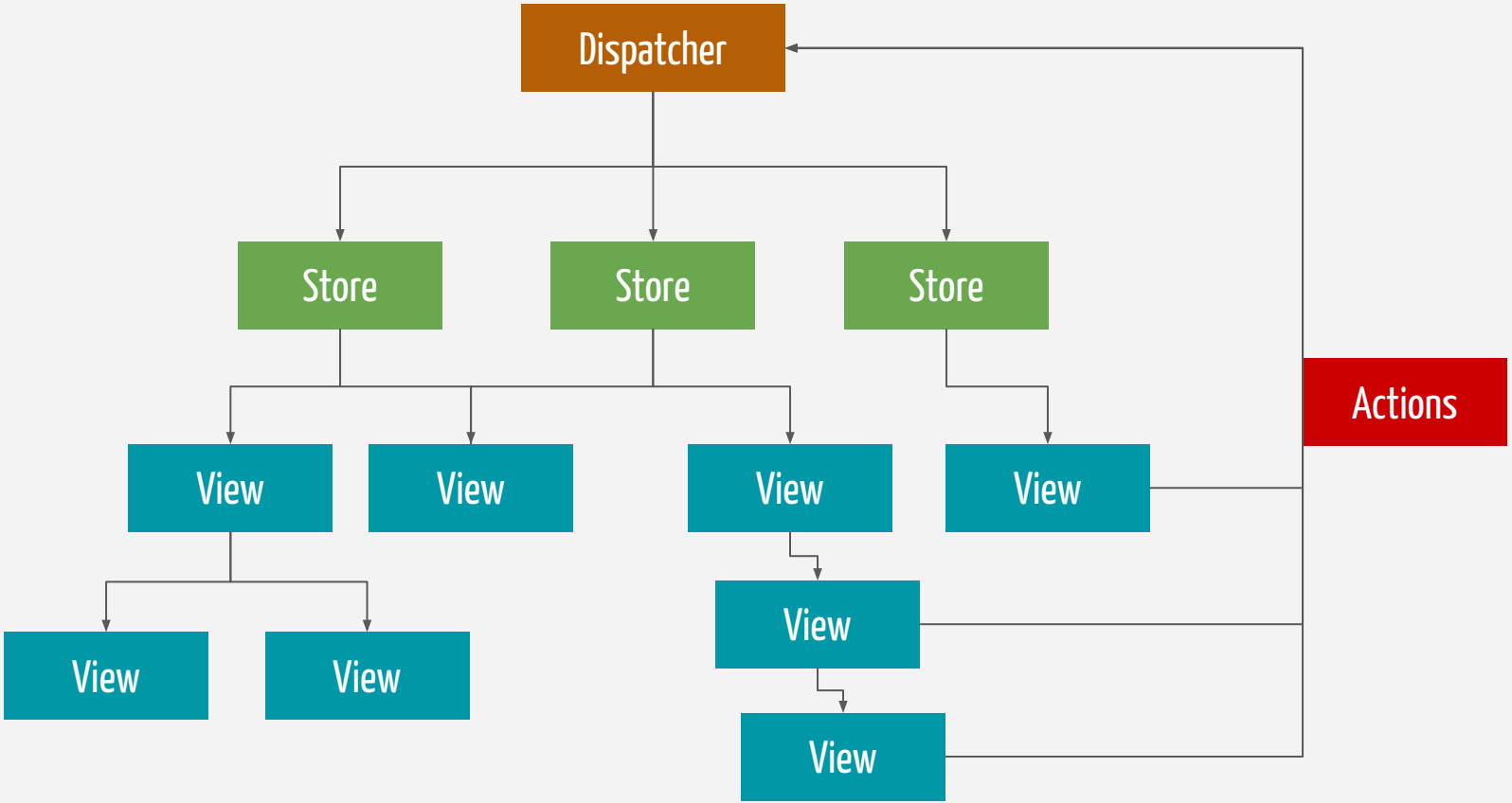


How to do async operations? REST-Requests?

ActionCreator can be asynchronous and create multpile Actions

1. Dispatch Action "FETCH_DATA_STARTED"
2. Request data from the server
3. When the data arrives → dispatch Action: "FETCH_DATA_SUCCESSFUL"
4. On timeout or error → dispatch Action: "FETCH_DATA_FAIL_TIMEOUT"

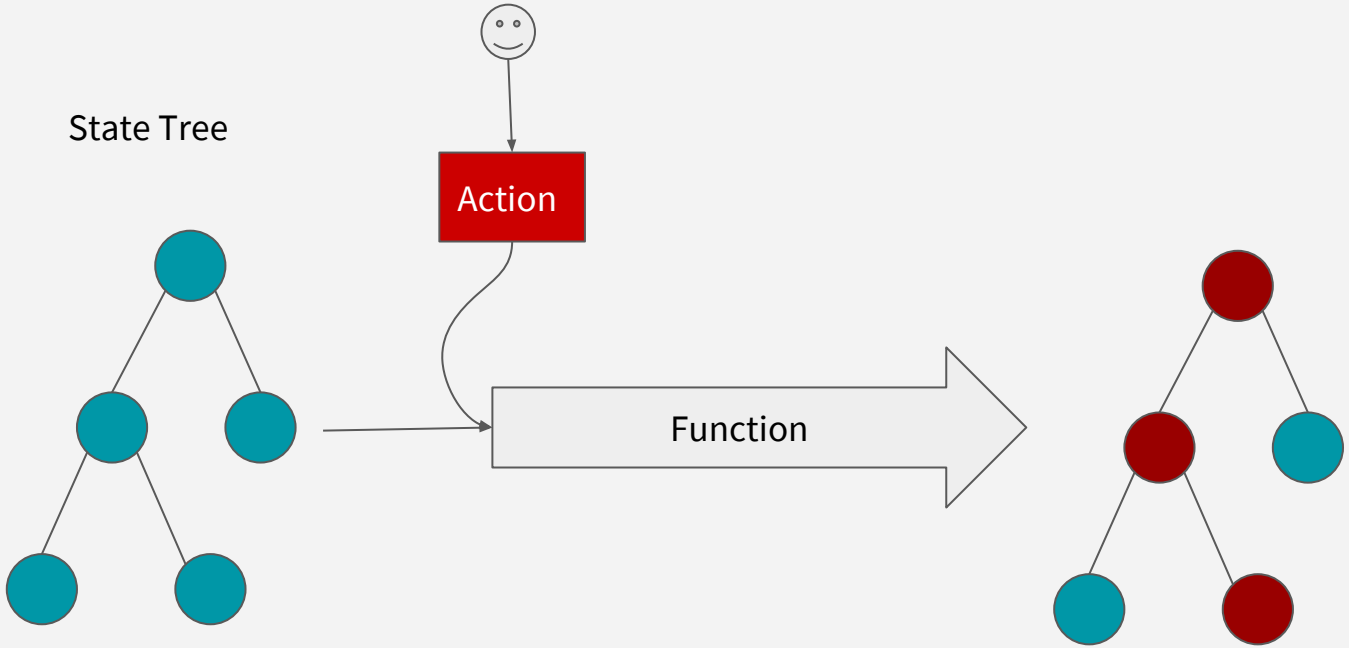
```
const fetchUsers = () => {
    dispatch({type: "FETCH_USERS_STARTED"});

    fetch("http://my.api.example.com/users")
        .then(response => response.json)
        .then(json => dispatch({
                type: "FETCH_USERS_SUCCESSFUL",
                payload: json
            }),
            error => dispatch({
                type: "FETCH_USERS_FAILED"
            })
        );
};
```

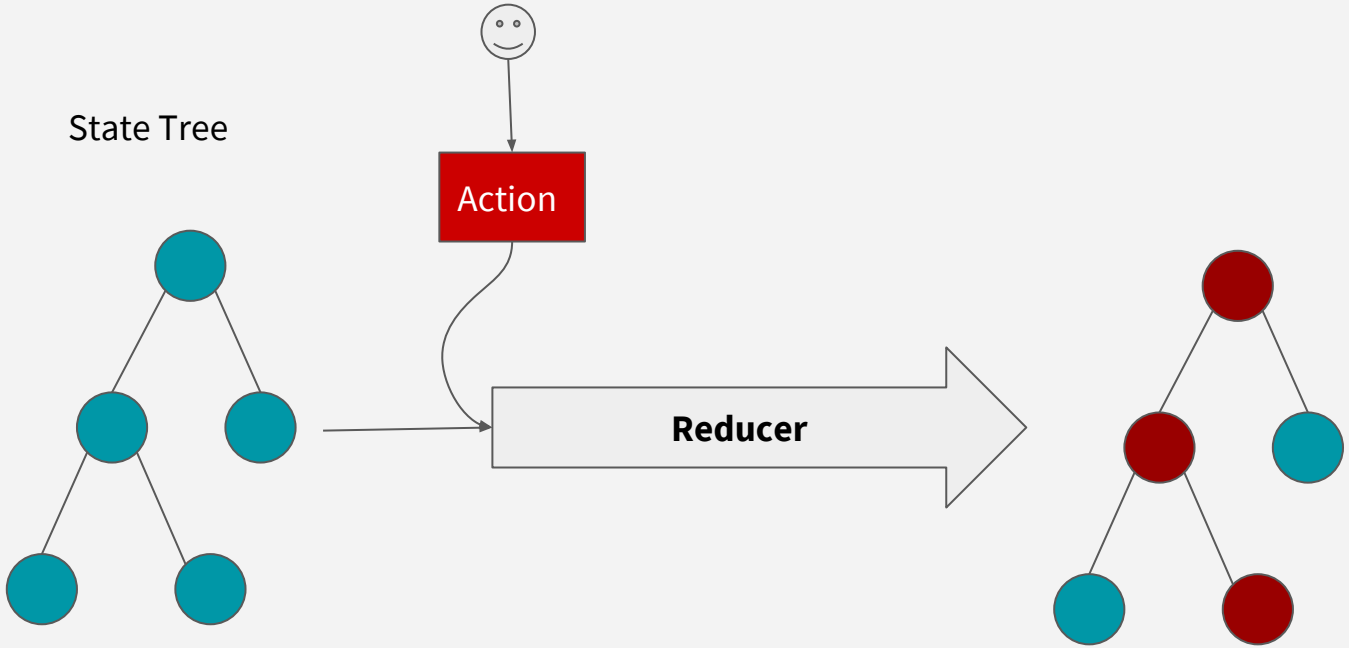

Dispatcher
Store
Store
Store
Actions
View
View
View
View
View
View
View
View

Flux in Functional: **Redux**

Redux

- Only 1 Store
- immutable State-Tree
- Reducer (transformation function):
  - signature: (state, action) → state
  - pure function
  - composable


State Tree
Action
Function


State Tree
Action
Reducer

Why the term "Reducer"?

```
var arr = [0,1,2,3];
var sum = arr.reduce(function(acc, val) {
                        return acc + val;
                    }, 0);
// sum is 6
```

Why the term "Reducer"?

Reducer Function

```
var arr = [0,1,2,3];
var sum = arr.reduce(function(acc, val) {
                        return acc + val;
                    }, 0);
// sum is 6
```

Why the term "Reducer"?

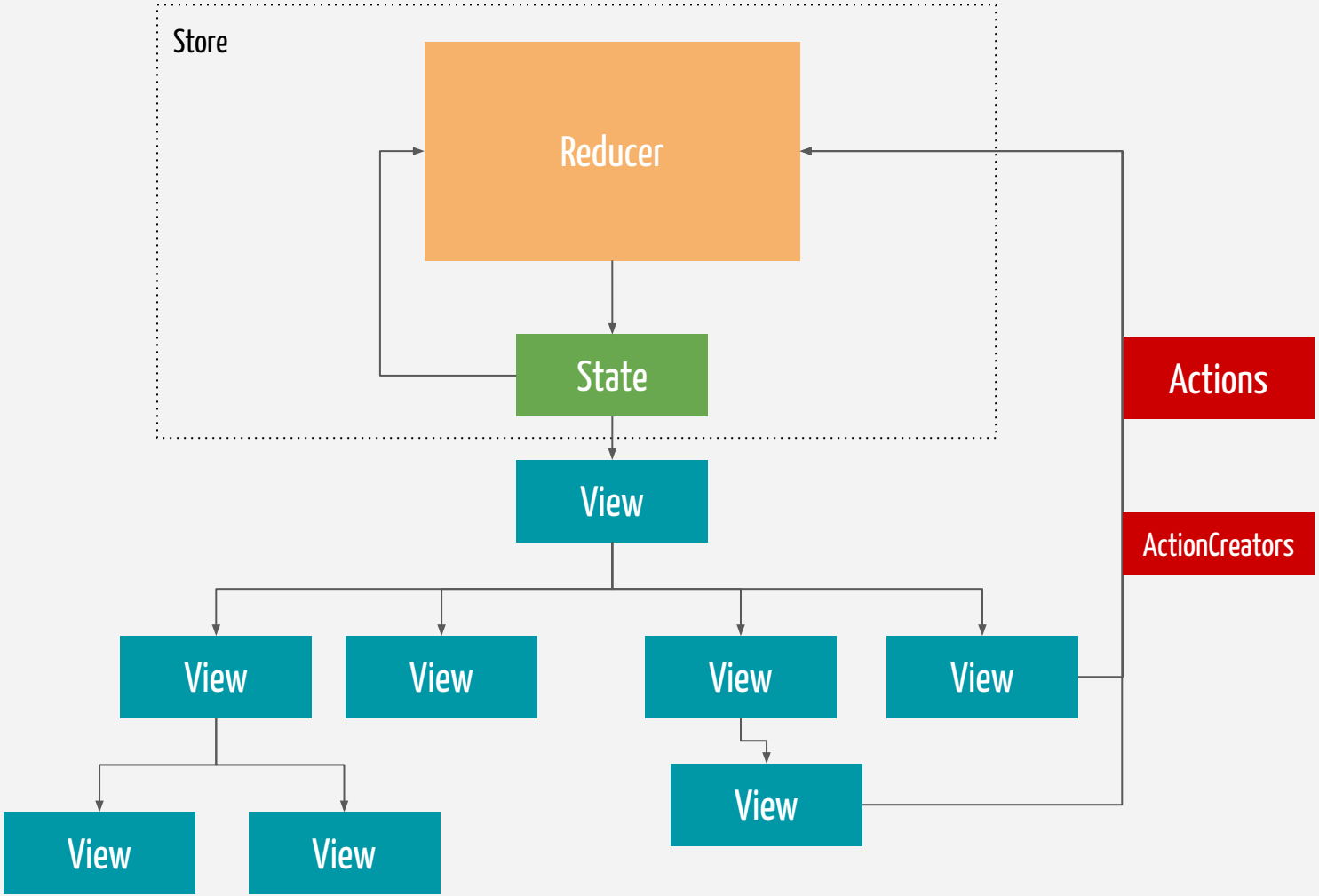
```
var arr = [0,1,2,3];
var sum = arr.reduce(function(acc, val) {
                        return acc + val;
                    }, 0);
// sum is 6
```

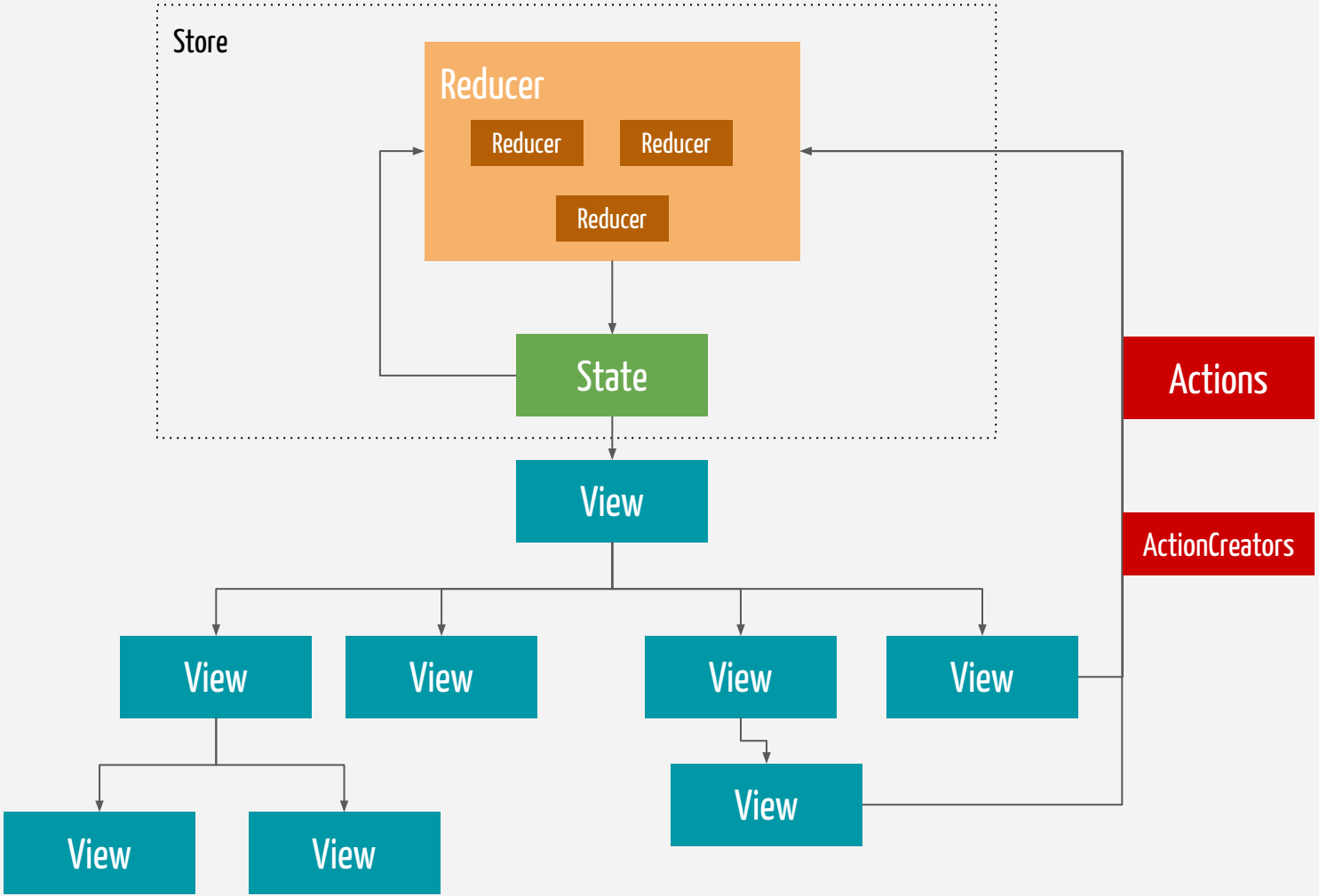
initialer Wert

Why the term "Reducer"?

```
var actions = [action1, action2, action3,...];

var currentState = actions.reduce(function(state, action) {
                        // calculate new state based
                        return newState;
                    }, initialState);
```

Redux = Flux + Reducer


Store
Reducer
State
Actions
View
ActionCreators
View
View
View
View
View
View
View


Store
Reducer
Reducer
Reducer
Reducer
State
Actions
ActionCreators
View
View
View
View
View
View
View
View

Redux in Code?

```
// action types
const LOAD_START = 'LOAD_START'
const LOAD_FINISHED = 'LOAD_FINISHED'
```

```
const LOAD_START = 'LOAD_START'
const LOAD_FINISHED = 'LOAD_FINISHED'

type State = {
     items: Array<string>
     loading: boolean
}
```

```
const LOAD_START = 'LOAD_START'
const LOAD_FINISHED = 'LOAD_FINISHED'

type State = {
     items: Array<string>
     loading: boolean
}

function reducer(state: State, action: Action): State {

}
```

```
const LOAD_START = 'LOAD_START'
const LOAD_FINISHED = 'LOAD_FINISHED'

type State = {
     items: Array<string>
     loading: boolean
}

function reducer(state: State, action: Action): State {
     switch(action.type) {


          default: return state;
     }
}
```

```
const LOAD_START = 'LOAD_START'
const LOAD_FINISHED = 'LOAD_FINISHED'

type State = {
     items: Array<string>
     loading: boolean
}

function reducer(state: State, action: Action): State {
     switch(action.type) {
          case LOAD_START:
               // todo: set loading=true

               return ? // state should be immutable, we need a copy

          default: return state;
     }
}
```

Immutability in JavaScript / TypeScript?

```
const state = {
     items: ["hallo", "welt"],
     loading: false
}


let newState = Object.assign({}, state); // creates a copy
```

Immutability in JavaScript / TypeScript?

```
const state = {
     items: ["hallo", "welt"],
     loading: false
}


let newState = Object.assign({}, state);
newState.loading = true;
```

Immutability in JavaScript / TypeScript?

```
const state = {
     items: ["hallo", "welt"],
     loading: false
}


let newState = Object.assign({}, state);
newState.loading = true;


const newState = Object.assign({}, state, { loading: true});
```

Immutability in JavaScript / TypeScript?

```
const state = {
     items: ["hallo", "welt"],
     loading: false
}


let newState = Object.assign({}, state);
newState.loading = true;


const newState = Object.assign({}, state, { loading: true});

// ES7 / TypeScript 2.1
const newState = {...state, {loading: true}};
```

Immutability in JavaScript / TypeScript?

```
// ES7 / TypeScript 2.1
const newState = {...state, {looading: true}};  // no compile error in TypeScript
```

Immutability in JavaScript / TypeScript?

```
// ES7 / TypeScript 2.1
const newState = {...state, {looading: true}};  // no compile error in TypeScript


// npm install tassign

const newState = tassign(state, {looading: true});     // compile error

const newState = tassign(state, {loading: true});      // no compile error
```

```
const LOAD_START = 'LOAD_START'
const LOAD_FINISHED = 'LOAD_FINISHED'

type State = {
     items: Array<string>
     loading: boolean
}

function reducer(state: State, action: Action): Reducer<State> {
     switch(action.type) {
          case LOAD_START:
               // todo: set loading=true

               return ? // state should be immutable, we need a copy

          default: return state;
     }
}
```

```
const LOAD_START = 'LOAD_START'
const LOAD_FINISHED = 'LOAD_FINISHED'

type State = {
     items: Array<string>
     loading: boolean
}

function reducer(state: State, action: Action): Reducer<State> {
     switch(action.type) {
          case LOAD_START:
               return tassign(state, { loading: true });

          default: return state;
     }
}
```

```
const LOAD_START = 'LOAD_START'
const LOAD_FINISHED = 'LOAD_FINISHED'

type State = {
     items: Array<string>
     loading: boolean
}

function reducer(state: State, action: Action): Reducer<State> {
     switch(action.type) {
          case LOAD_START:
               return tassign(state, { loading: true });

          case LOAD_FINISHED:
               let newItems = action.payload.items;

               return tassign(state, {
                    items: [...state.items, ...newItems],
                    loading: false
               });
          default: return state;
     }
```

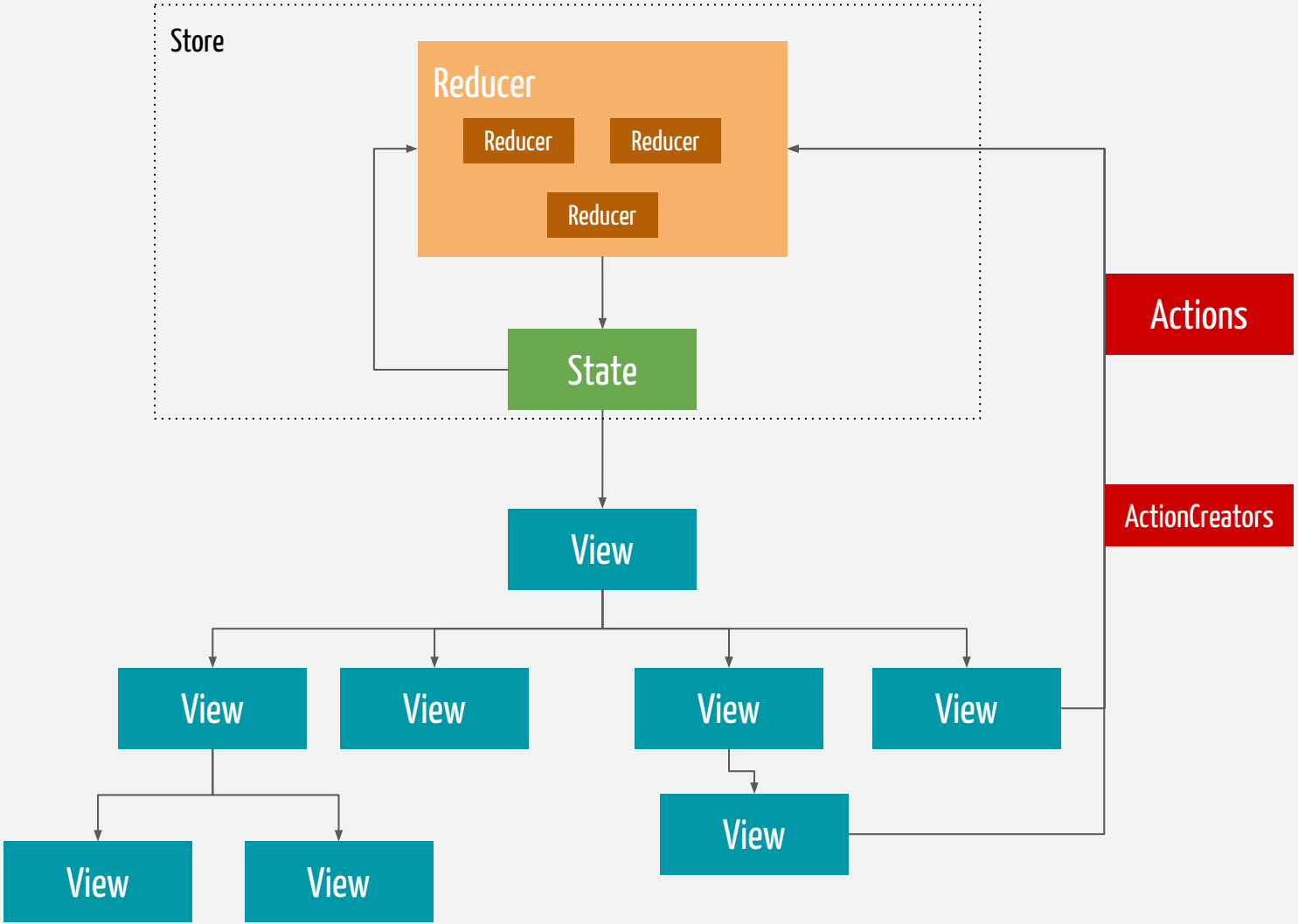
- The whole state in one single data structure?
- One reducer contains all logic?

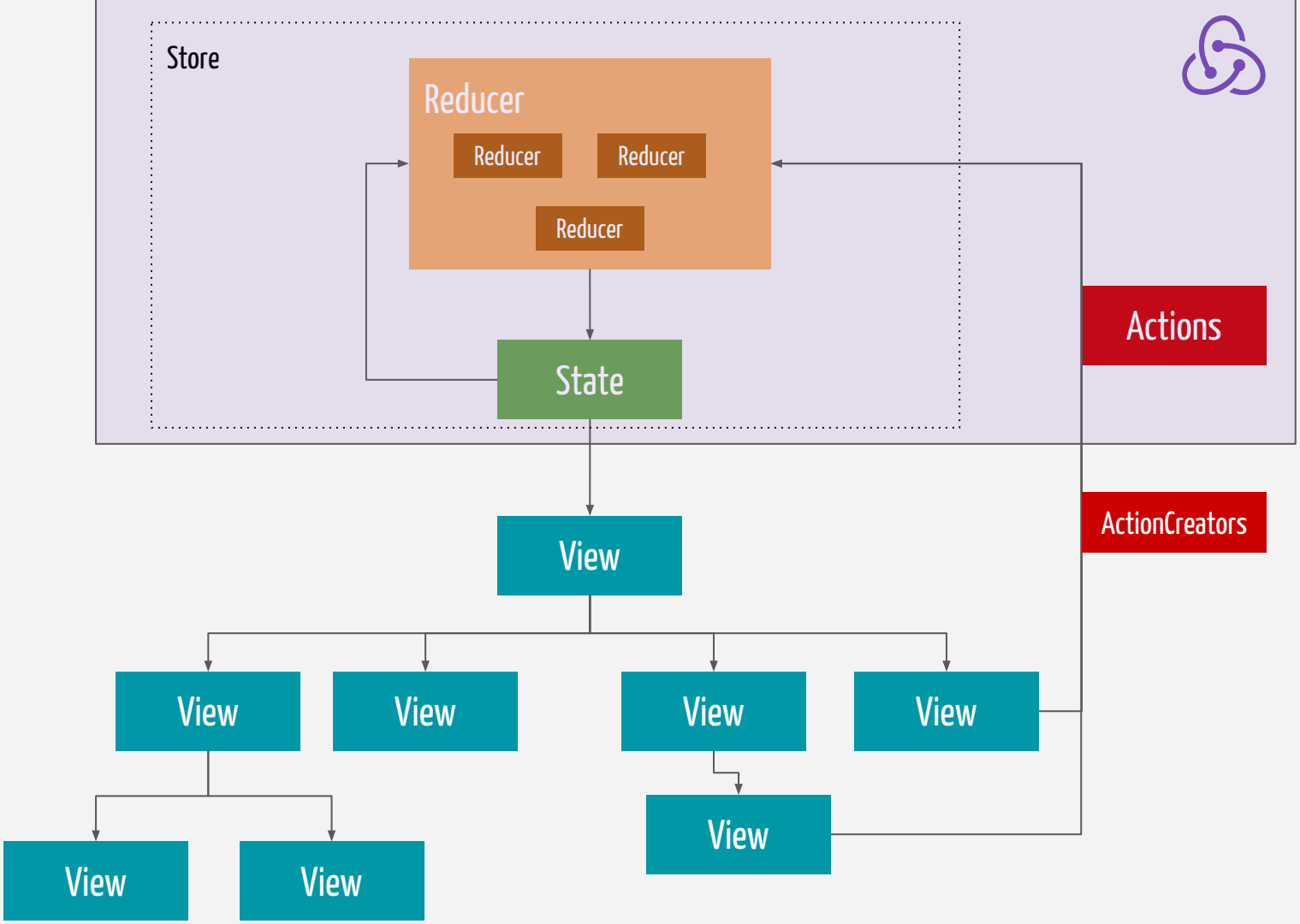
Reducer Composition

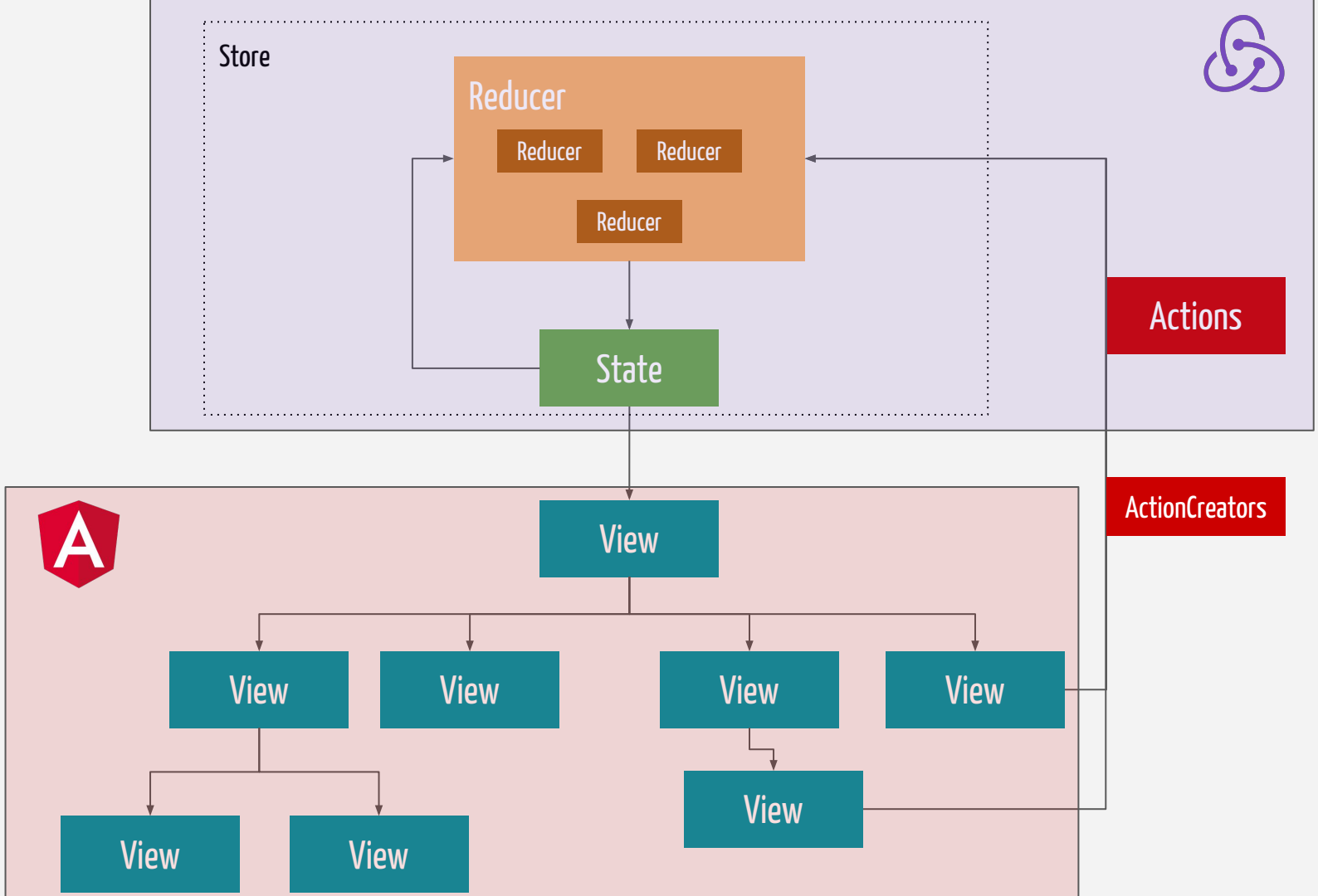
- The whole state in one single data structure?
- One reducer contains all logic?

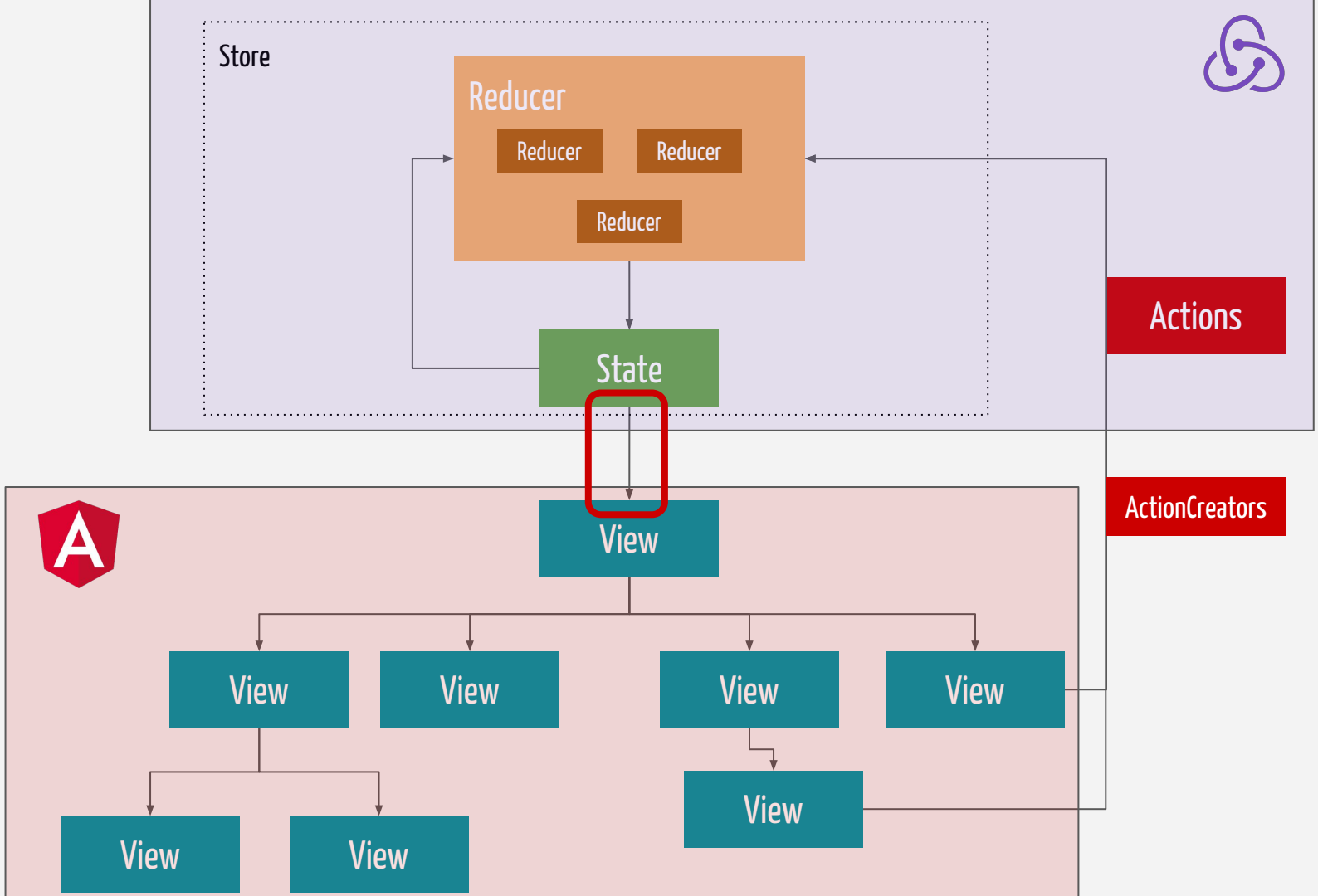
```
import { combineReducers } from 'redux'

import userReducer from '../users'
import productsReducer from '../products'
import categoriesReducer from '../categories'

var rootReducer = combineReducers({
    users: userReducer,
    products: productsReducer,
    categories: categoriesReducer,
})
```

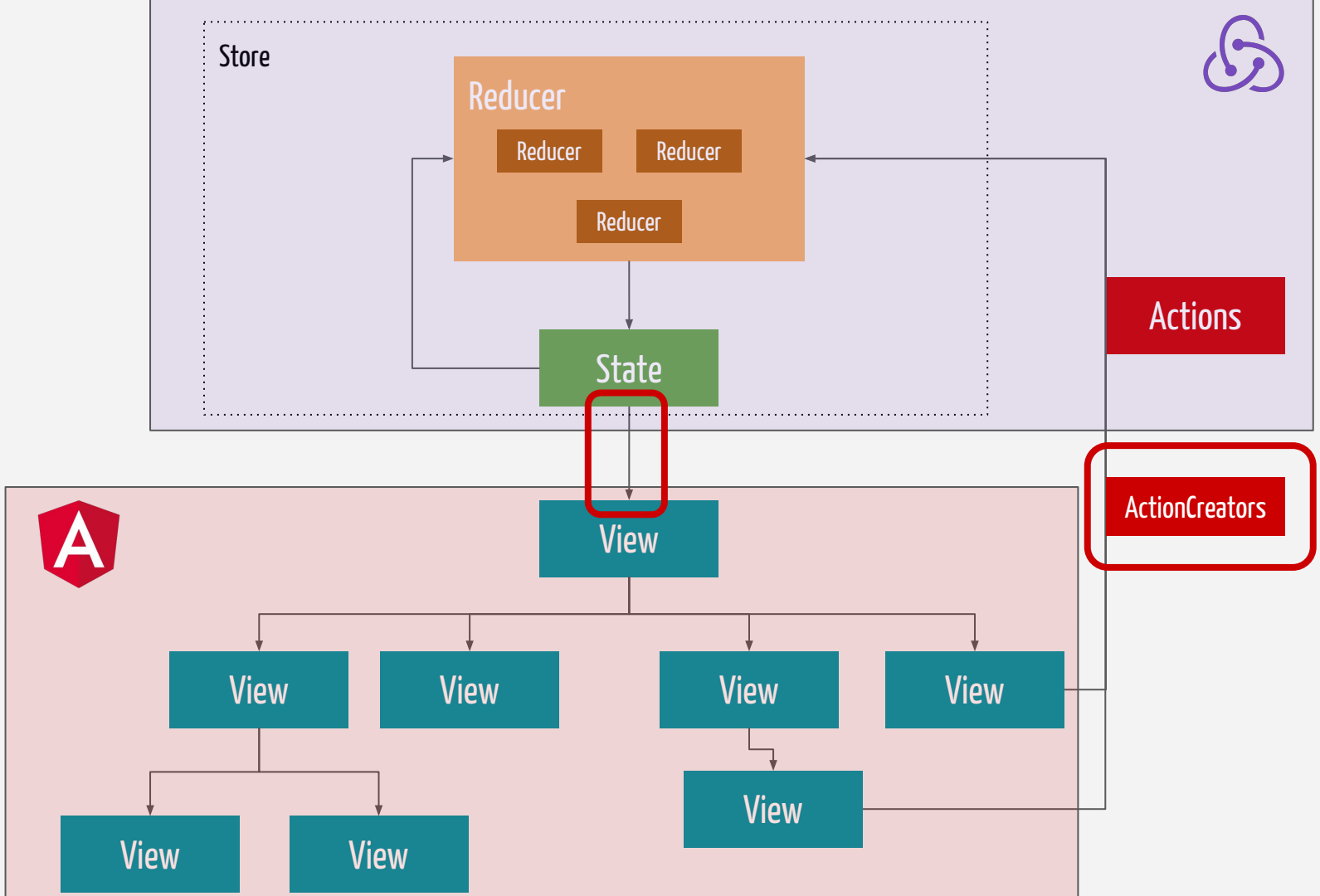
Angular + Redux


Store
Reducer
Reducer
Reducer
Reducer
State
Actions
ActionCreators
View
View
View
View
View
View
View
View


Store
Reducer
Reducer
Reducer
Reducer
State
Actions
ActionCreators
View
View
View
View
View
View
View
View


Store
Reducer
Reducer
Reducer
Reducer
State
Actions
ActionCreators
View
View
View
View
View
View
View
View


Store
Reducer
Reducer
Reducer
Reducer
State
Actions
ActionCreators
View
View
View
View
View
View
View
View


Store
Reducer
Reducer
Reducer
Reducer
State
Actions
ActionCreators
View
View
View
View
View
View
View
View

angular-redux

Angular + Redux?

- https://github.com/angular-redux
- combines original redux lib with Angular
- pending questions:
  - ActionCreators needs dependency jnjection
  - How to get the data from the store into the UI components?

ActionCreators

```
import { Injectable } from '@angular/core'

@Injectable()
export class ProductsActionCreators {
}
```

```
import { NgRedux } from '@angular-redux/store'
import { AppState } from '../appstate'

@Injectable()
export class ProductsActionCreators {
    constructor(private ngRedux: NgRedux<AppState>) {}
}
```

```
import { Http } from '@angular/http'

@Injectable()
export class ProductsActionCreators {
    constructor(private ngRedux: NgRedux<AppState>, private http: Http) {}
}
```

```
@Injectable()
export class ProductsActionCreators {
    constructor(private ngRedux: NgRedux<AppState>, private http: Http) {}

    public loadProducts() {
    }
}
```

```
@Injectable()
export class ProductsActionCreators {
    constructor(private ngRedux: NgRedux<AppState>, private http: Http) {}

    public loadProducts() {
        this.ngRedux.dispatch({
            type: 'LOAD_PRODUCTS_START'
        })
    }
}
```

```
@Injectable()
export class ProductsActionCreators {
    constructor(private ngRedux: NgRedux<AppState>, private http: Http) {}

    public loadProducts() {
        this.ngRedux.dispatch({
            type: 'LOAD_PRODUCTS_START'
        })

        this.http.get('/api/products')
            .map(resp => resp.json())
    }
}
```

```
@Injectable()
export class ProductsActionCreators {
    constructor(private ngRedux: NgRedux<AppState>, private http: Http) {}

    public loadProducts() {
        this.ngRedux.dispatch({
            type: 'LOAD_PRODUCTS_START'
        })

        this.http.get('/api/products')
            .map(resp => resp.json())
            .subscribe(res => {

             });
    }
}
```

```
@Injectable()
export class ProductsActionCreators {
    constructor(private ngRedux: NgRedux<AppState>, private http: Http) {}

    public loadProducts() {
        this.ngRedux.dispatch({
            type: 'LOAD_PRODUCTS_START'
        })

        this.http.get('/api/products')
            .map(resp => resp.json())
            .subscribe(res => {
                this.ngRedux.dispatch({
                    type: 'LOAD_PRODUCTS_FINISHED',
                    payload: {
                        json: res
                    }
                })
             });
    }
}
```

How to get the data
from the store
into the UI components?

Selector function

- concept is well-tested in react-redux community
- Selector: a pure function that queries some data from the State
- `(state) → T`

```
function isLoading (state: AppState): boolean {

    return state.products.loadingFlag;

}
```

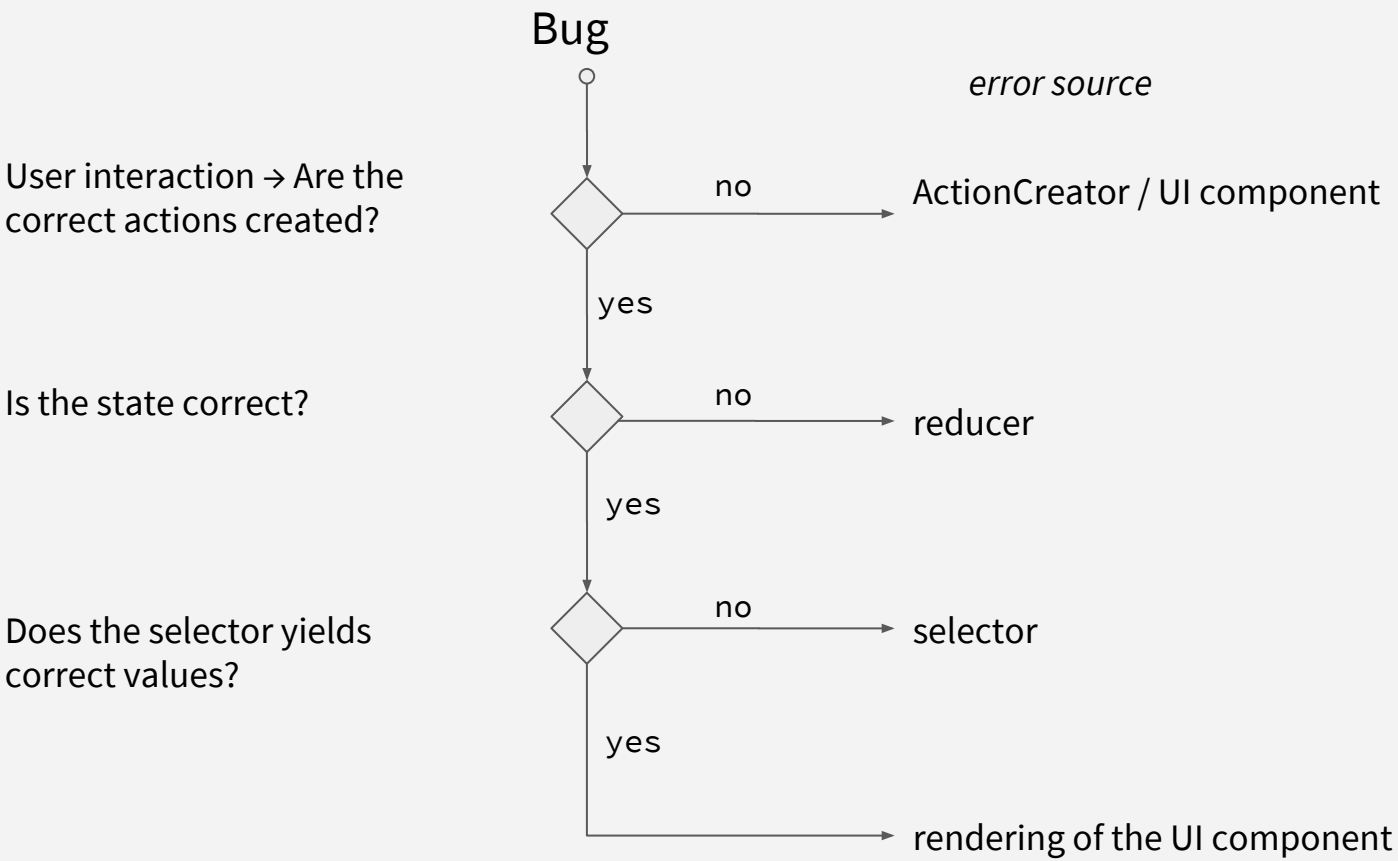
```
import { select } from '@angular-redux/store'
import { isLoading } from '../products-selectors'

@Component({
    selector: 'app-product-overview',
    templateUrl: '...'
})
export class ProductOverviewComponent {

    @select(isLoading)
    public loading: Observable<boolean>

}
```

```
// products-overview.component.html
<div>
    <h1>Products</h1>
    …
    <p *ngIf="loading | async">Loading...</p>
</div>
```

Conclusion

Debugging

- Time-Travel-Debugging
- Whole application state is visible in one place
- clear solution process to find and fix bugs


Bug
error source
User interaction → Are the correct actions created?
no
ActionCreator / UI component
yes
Is the state correct?
no
reducer
yes
Does the selector yields correct values?
no
selector
yes
rendering of the UI component

More files

- Reducer, Selectors, ActionTypes, ActionCreators…
- however: files have clear responsibility
- learing curve: new devs may have problems in the beginning

testability

- Reducer and Selectors are pure functions
- asynchronous behaviour is encapsulated in ActionCreators
- UI components are conceptually similar to functions:
  - New data comes in → What is rendered?
  - Interaction by the user → Are correct ActionCreators invoked (Mocking)

Thinking functional, modelling state

- functional way of thinking may be unfamiliar to OOP developers
  - Pure Functions
  - Function Composition
  - Immutability
  - Reducer Functions

- How to model the state?
- How to compose the reducers?
- Asynchronous operations?

Simpler Transition React ⟵ ⟶ Angular

- Same basics for react and angular projects
- a lot of code can be reused without modification
- developer know how.
- concept is usable on other platforms too:
  - React-Native/NativeScript → Mobile
  - Java Desktop → JavaFX
  - ...

Q&A

@manuel_mauky
github.com/lestard
www.lestard.eu

JUG Görlitz

Saxonia Systems
So geht Software.