



# Angular mit Redux



Manuel Mauky  
@manuel\_mauky

JUG  
Görlitz



**Saxonia** Systems  
So geht Software.

# Warum Redux?

# Warum Redux?

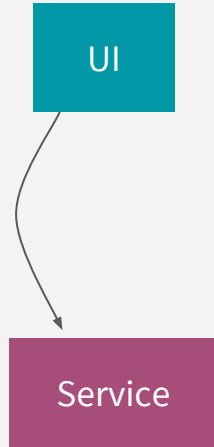
Was kann bei Angular schief gehen?

# Client-Logik + Datenfluss bei Angular

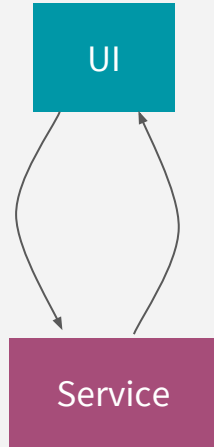


UI

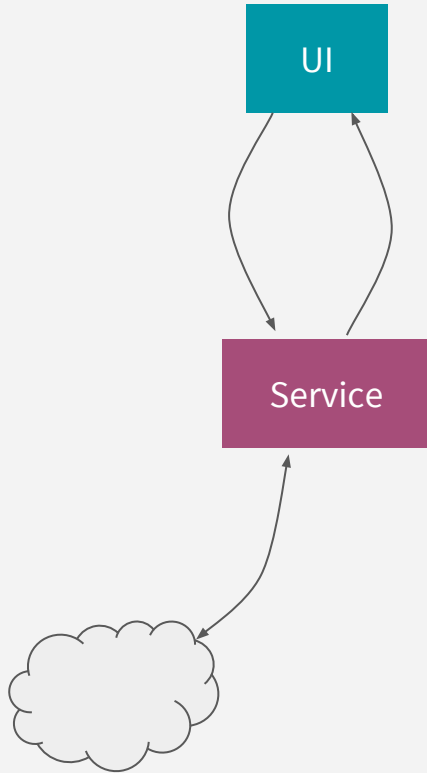
# Client-Logik + Datenfluss bei Angular



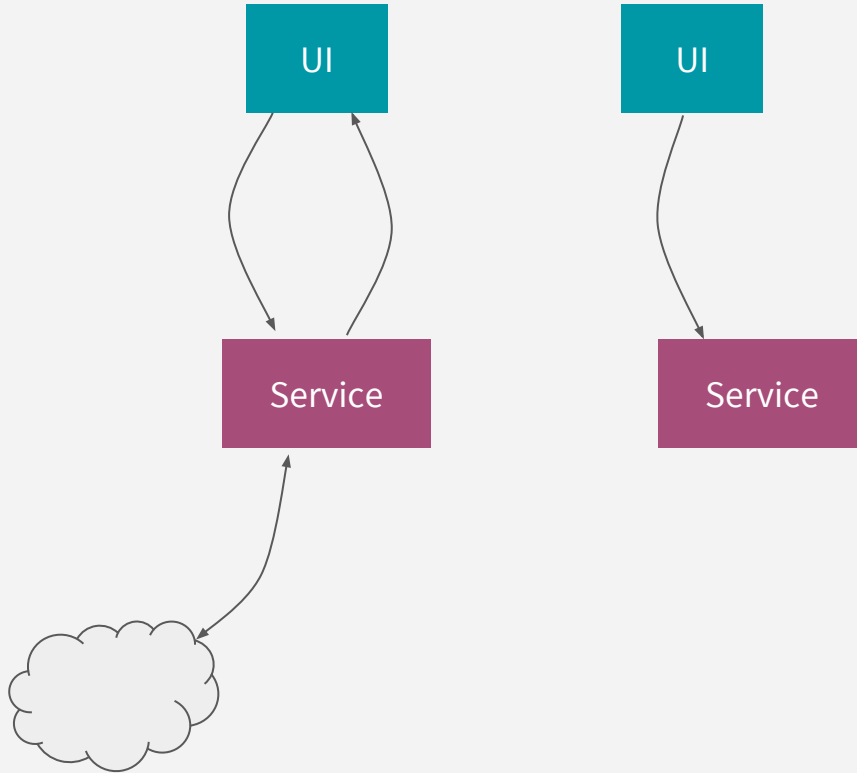
# Client-Logik + Datenfluss bei Angular



# Client-Logik + Datenfluss bei Angular

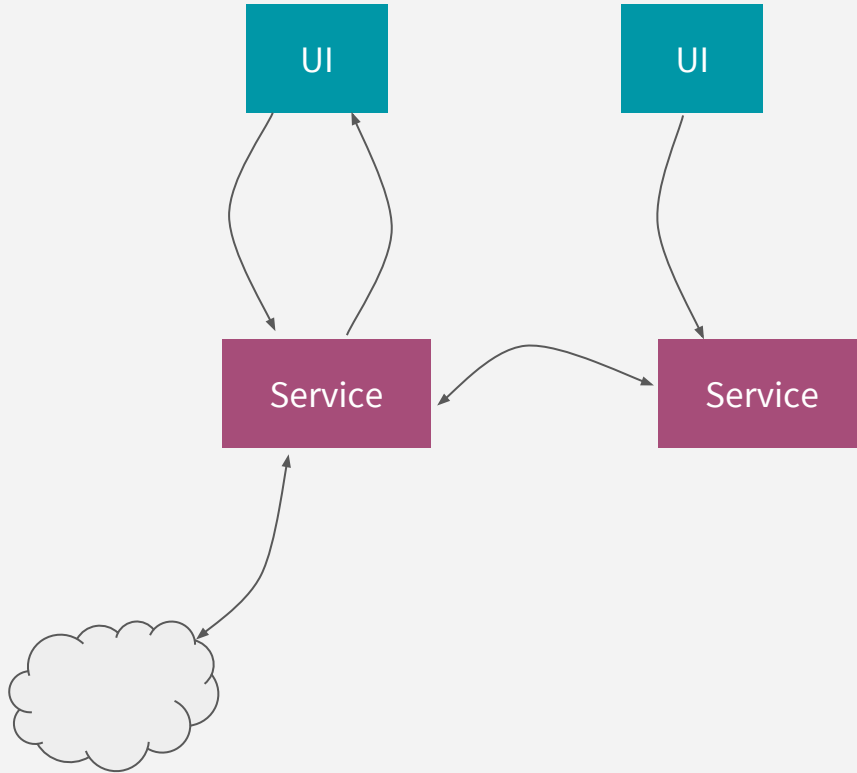


# Client-Logik + Datenfluss bei Angular

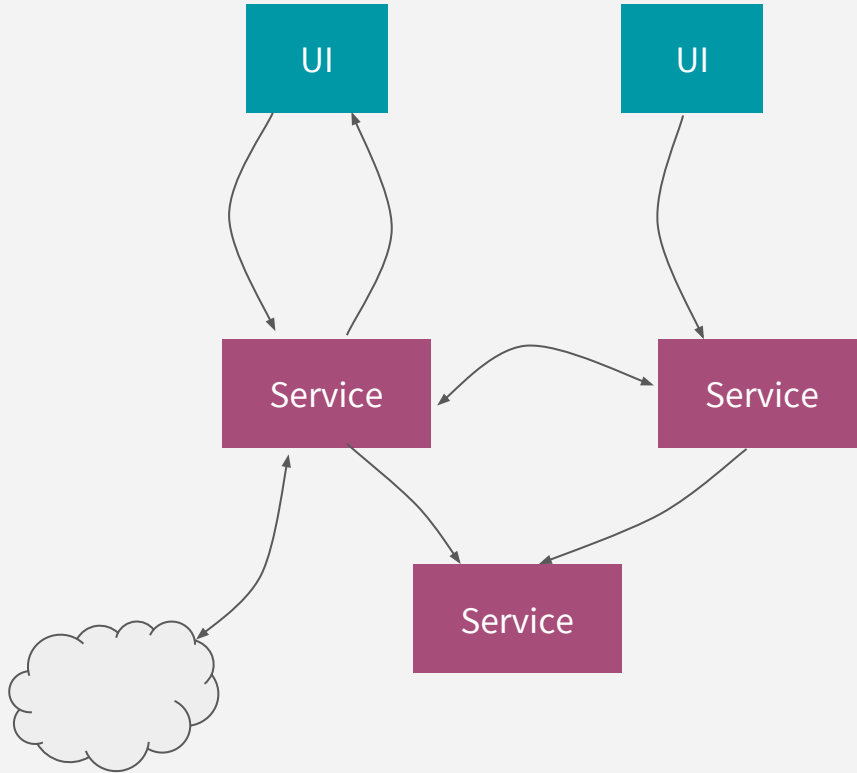




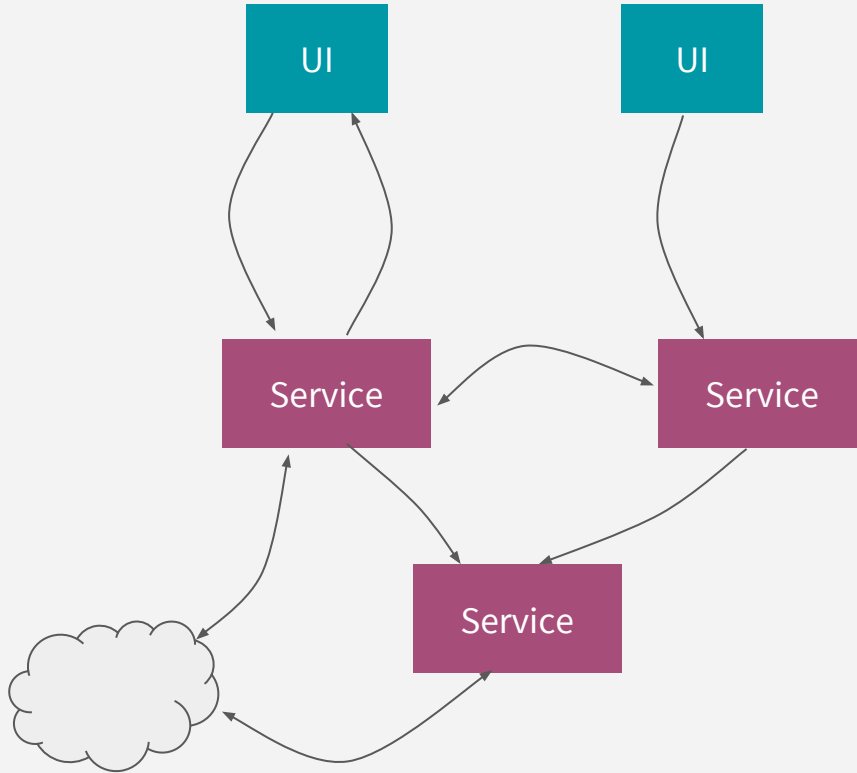
# Client-Logik + Datenfluss bei Angular



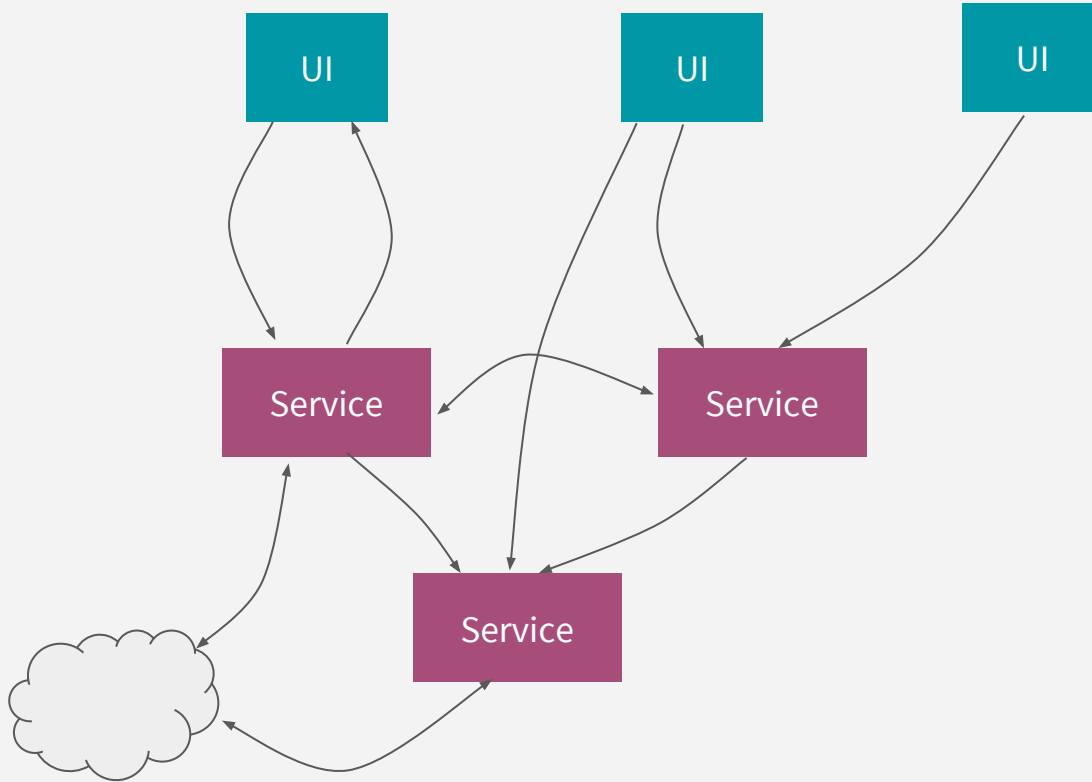
# Client-Logik + Datenfluss bei Angular



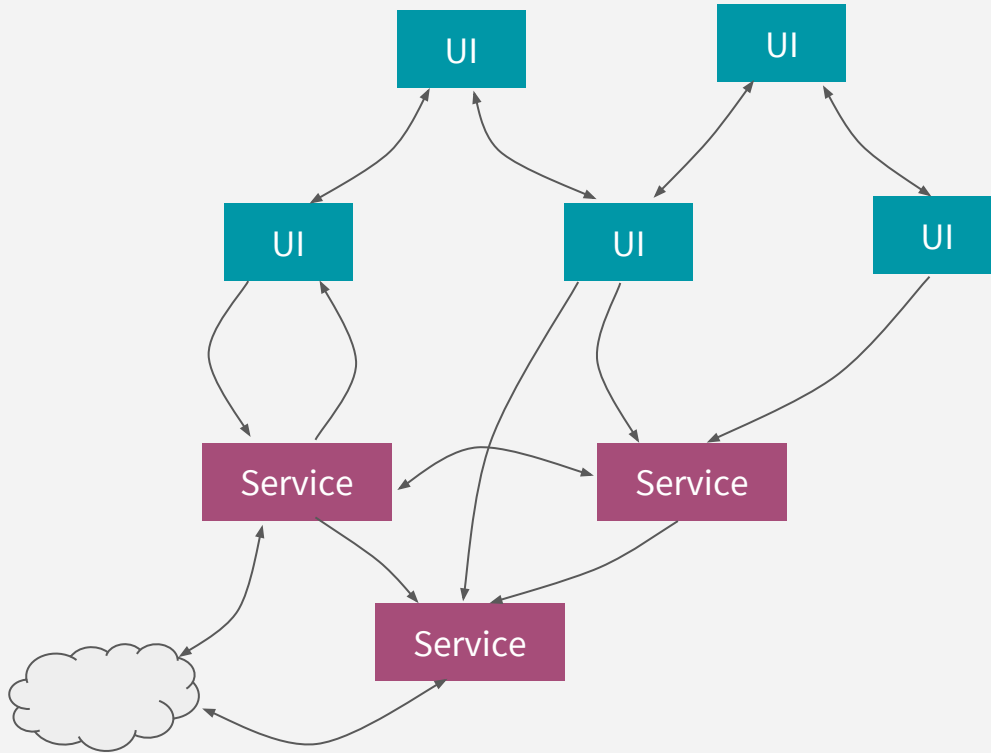
# Client-Logik + Datenfluss bei Angular



# Client-Logik + Datenfluss bei Angular



# Client-Logik + Datenfluss bei Angular



- Was ist der aktuelle Zustand meiner Anwendung?
- Wo liegen einzelne Teile des Zustands?
- Warum ist der Zustand so?
- Wie kam es zu diesem Zustand?
- Ein Bug tritt auf → Wo muss ich suchen?
- Objekt-Orientierung:
  - Kombination der Aspekte "Daten" und "Verhalten"

- Was ist der aktuelle Zustand meiner Anwendung?
- Wo liegen einzelne Teile des Zustands?
- Warum ist der Zustand so?
- Wie kam es zu diesem Zustand?
- Ein Bug tritt auf → Wo muss ich suchen?
- Objekt-Orientierung:
  - ~~Kombination~~ **Vermischung** der Aspekte "Daten" und "Verhalten"

- Was ist der aktuelle Zustand meiner Anwendung?
- Wo liegen einzelne Teile des Zustands?
- Warum ist der Zustand so?
- Wie kam es zu diesem Zustand?
- Ein Bug tritt auf → Wo muss ich suchen?
- Objekt-Orientierung:
  - ~~Kombination~~ **Vermischung** der Aspekte "Daten" und "Verhalten"
  - Abstraktion über Zeit

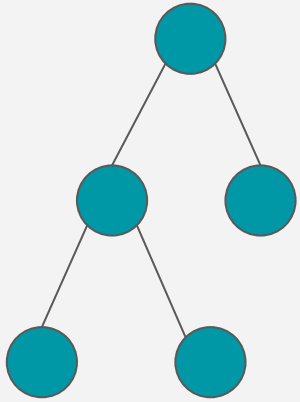


Funktionale Programmierung?

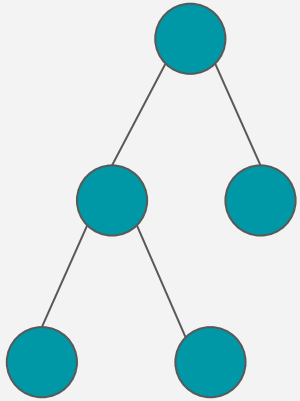
# Funktionale Programmierung

- Pure Funktionen (keine Seiteneffekte)
- Immutable Data (unveränderlich)

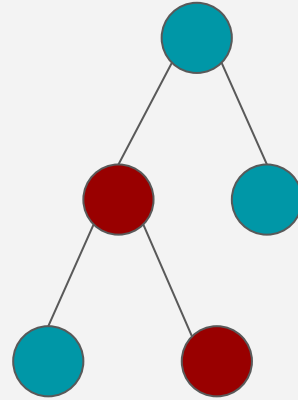
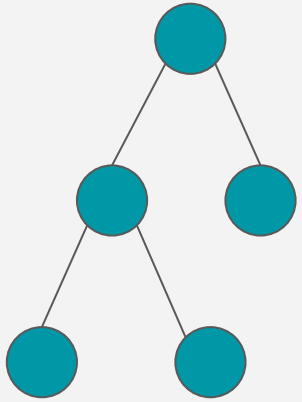
## Zustandsbaum



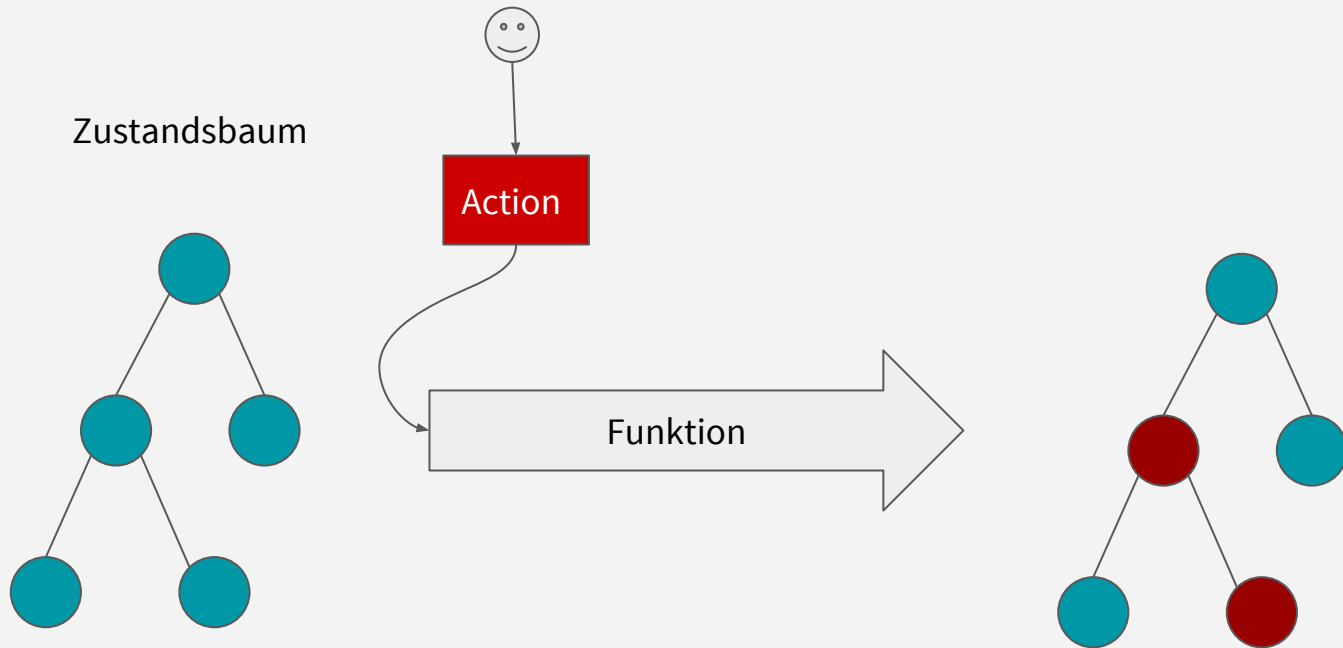
Zustandsbaum



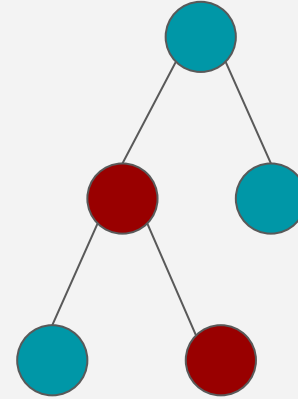
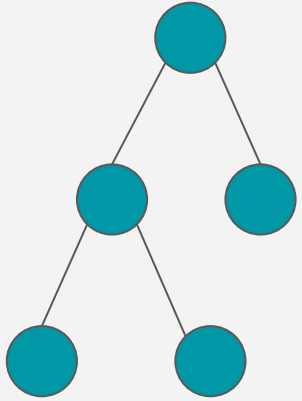
## Zustandsbaum



Zustandsbaum



Zustandsbaum



Zustands-Automat mit Überföhrungsfunktion

Wo kommt Redux her?  
Wie funktioniert Redux genau?





Redux-Vorläufer: **Flux-Architektur**

**Flux** ist ein Architektur-Muster (Alternative zu MVC\*)

**Redux** ist eine Weiterentwicklung von **Flux** + Implementierung

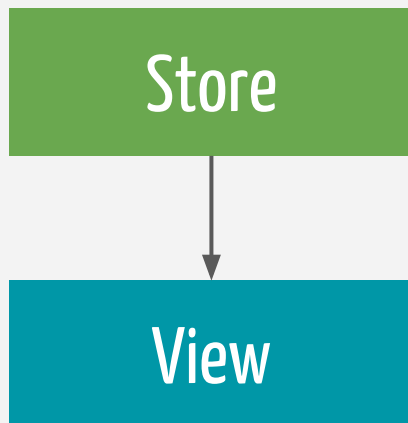
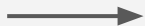
**Flux** ist Objekt-Orientiert + funktionale Einflüsse

**Redux** ist (fast) vollständig funktional

# Store

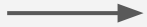
- Applikationszustand
- Logik
- repräsentiert eine fachliche Einheit

Datenfluss



- View zeigt Daten von Stores an
- Wenn sich Store ändert, wird View geupdated

Datenfluss



Store



View

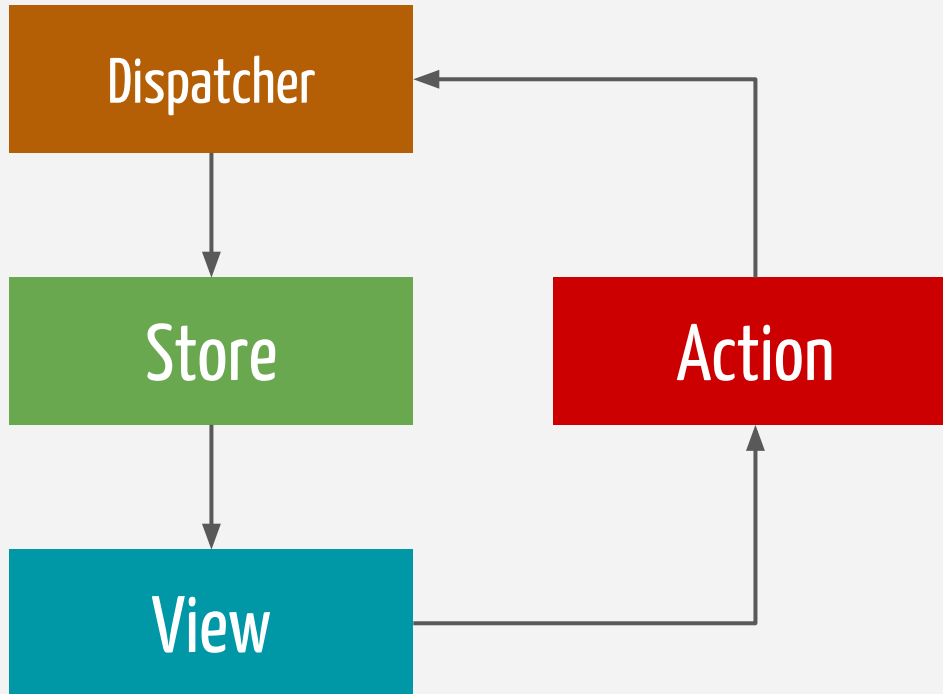


Action

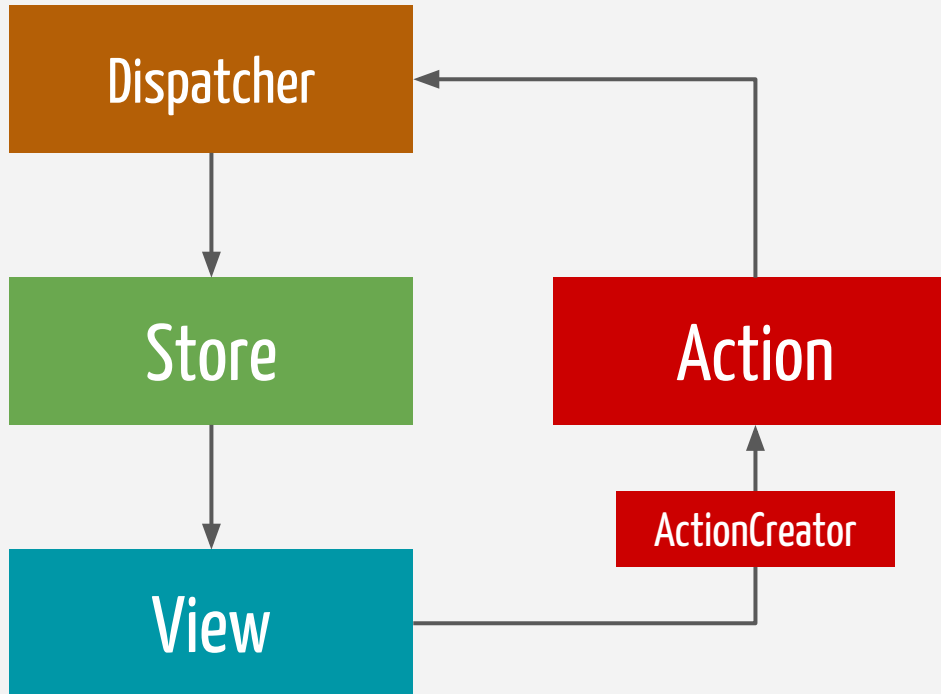
- View erzeugt “Actions” bei Userinteraktion
- Action repräsentiert fachliche Aktion
- vergleichbar Command-Pattern

## Beispiel: Action

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



- Dispatcher leitet **alle** Actions an **alle** Stores weiter
- Stores entscheiden selbst, ob und wie sie auf Actions reagieren



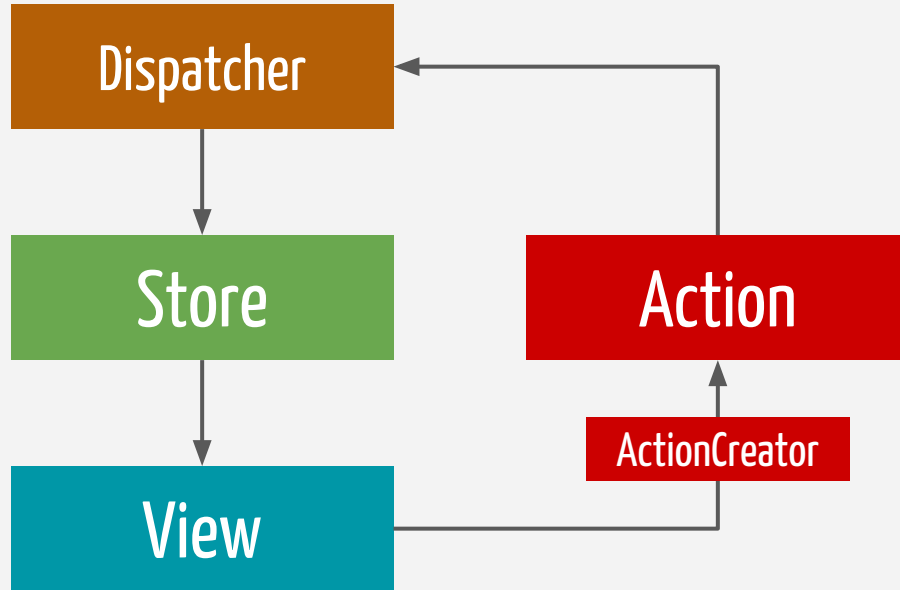
- Funktion, die Actions erzeugt
- Entkopplung von konkreter Action-Erzeugung



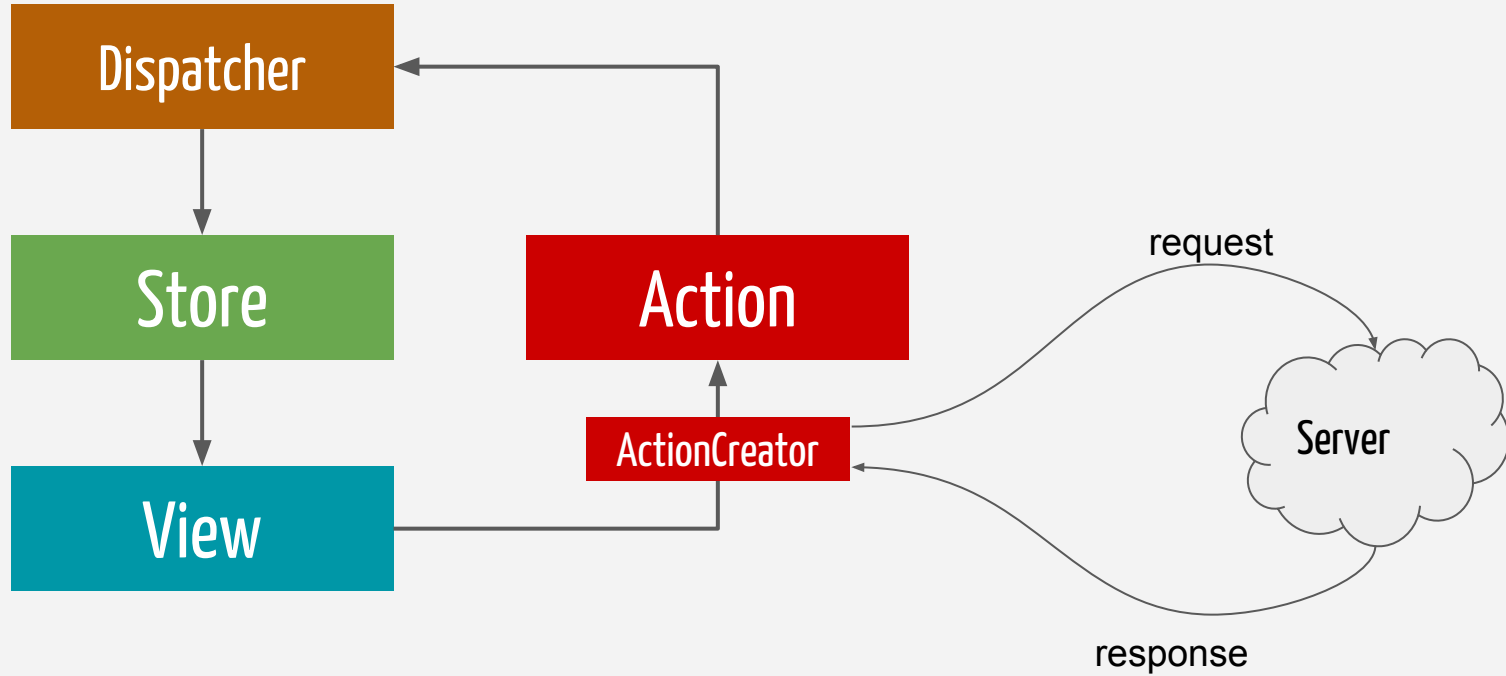
## Example: ActionCreator

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

# Asynchronität? REST-Requests?



# Asynchronität? REST-Requests?

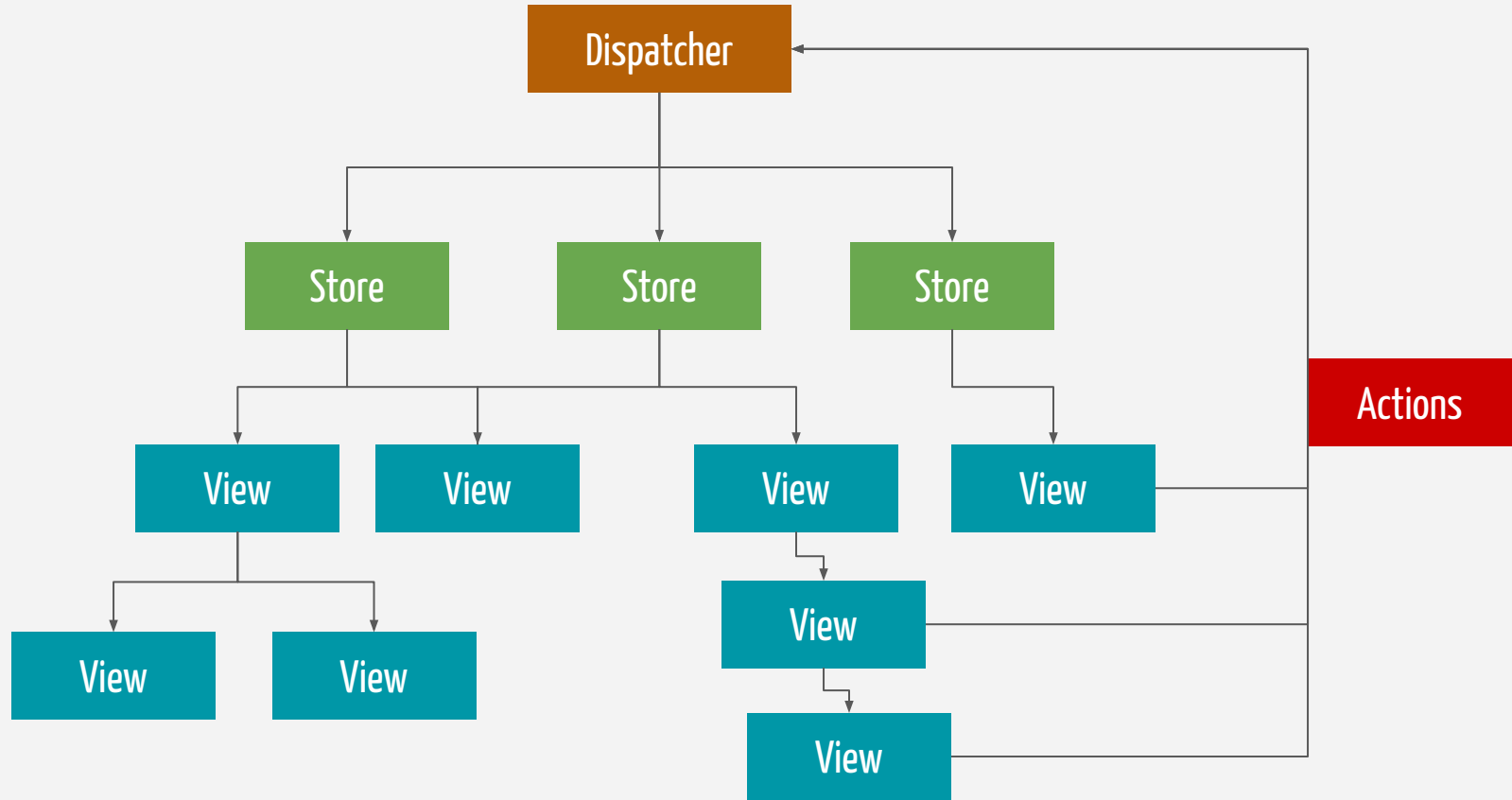


# Asynchronität? REST-Requests?

ActionCreator können Asynchron sein und mehrere Actions erzeugen

1. Dispatch Action "FETCH\_DATA\_STARTED"
2. Hole Daten vom Server
3. Wenn Daten ankommen → dispatch Action: "FETCH\_DATA\_SUCCESSFUL"
4. Bei Timeout oder Fehler → 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"  
    })  
  );  
};
```





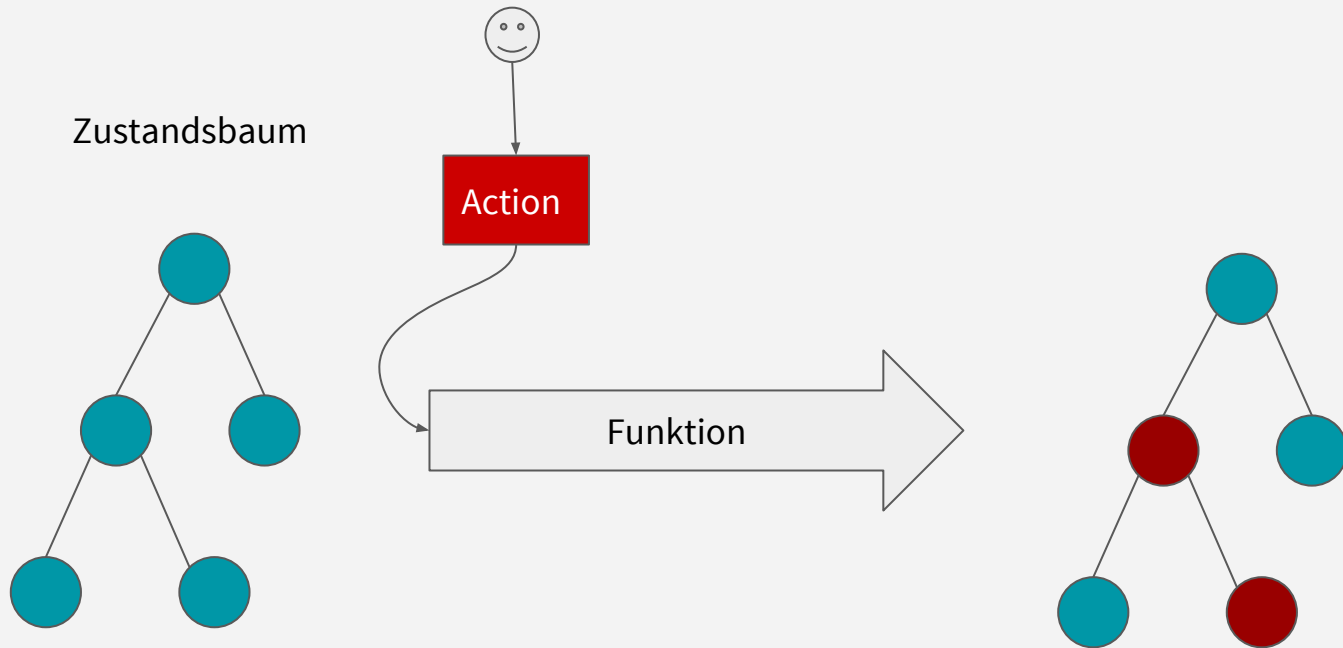
Flux in Funktional: **Redux**

# Redux

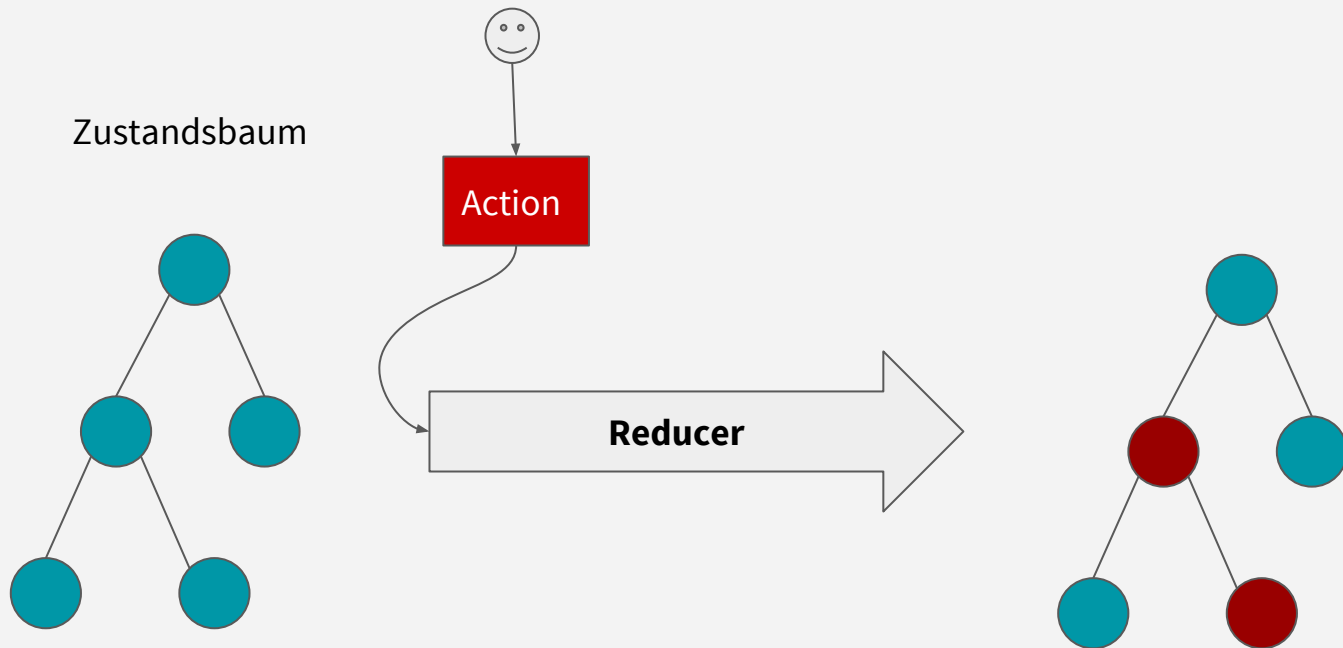
- Nur 1 Store
- immutable State-Tree
- Reducer (Überföhrungsfunktion):
  - Signatur:  $(state, action) \rightarrow state$
  - Pure Function
  - composable



Zustandsbaum



Zustandsbaum




## Begriff "Reducer"?

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

# Begriff "Reducer"?

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

Reducer-Funktion



## Begriff "Reducer"?

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

initialer Wert

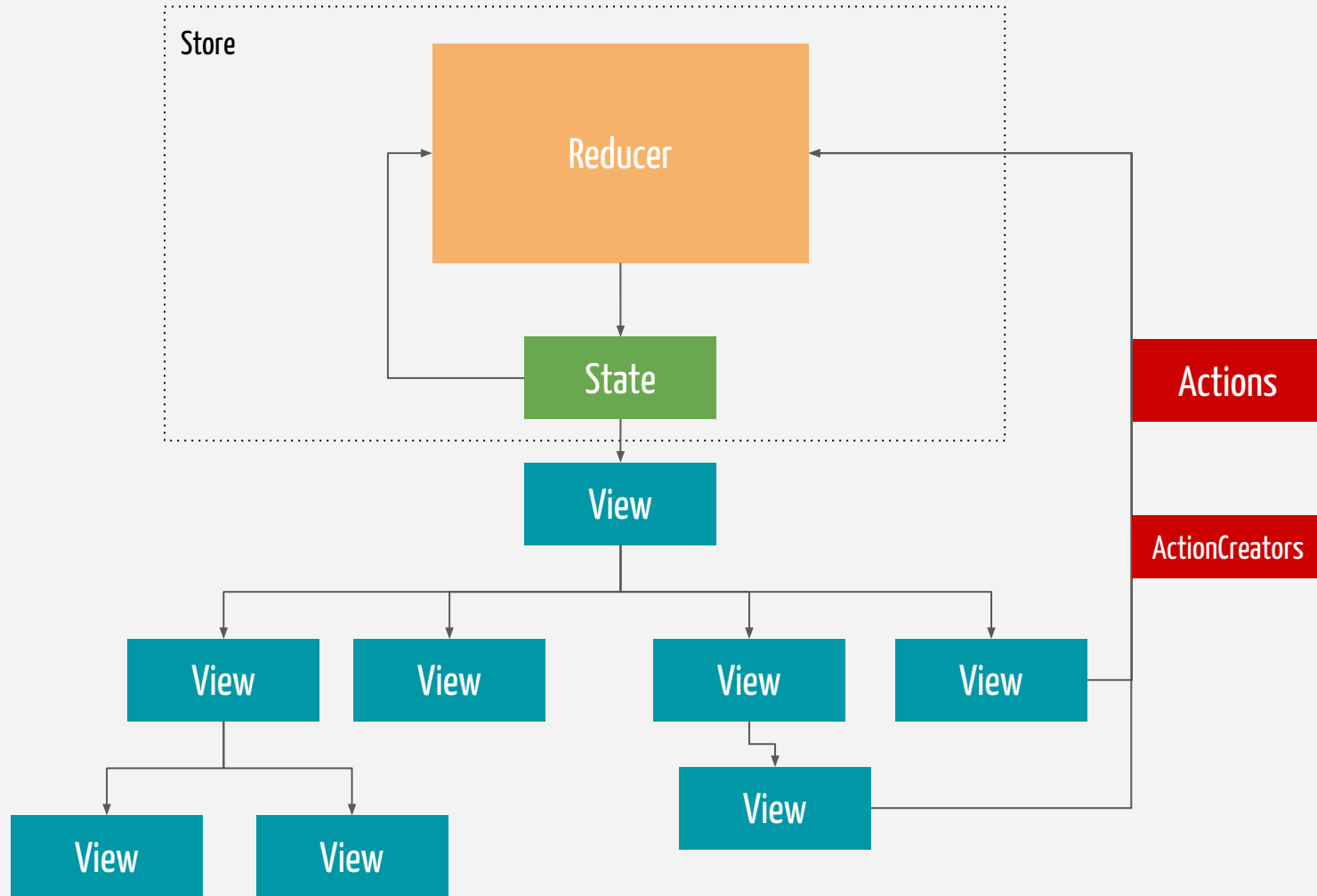


## Begriff "Reducer"?

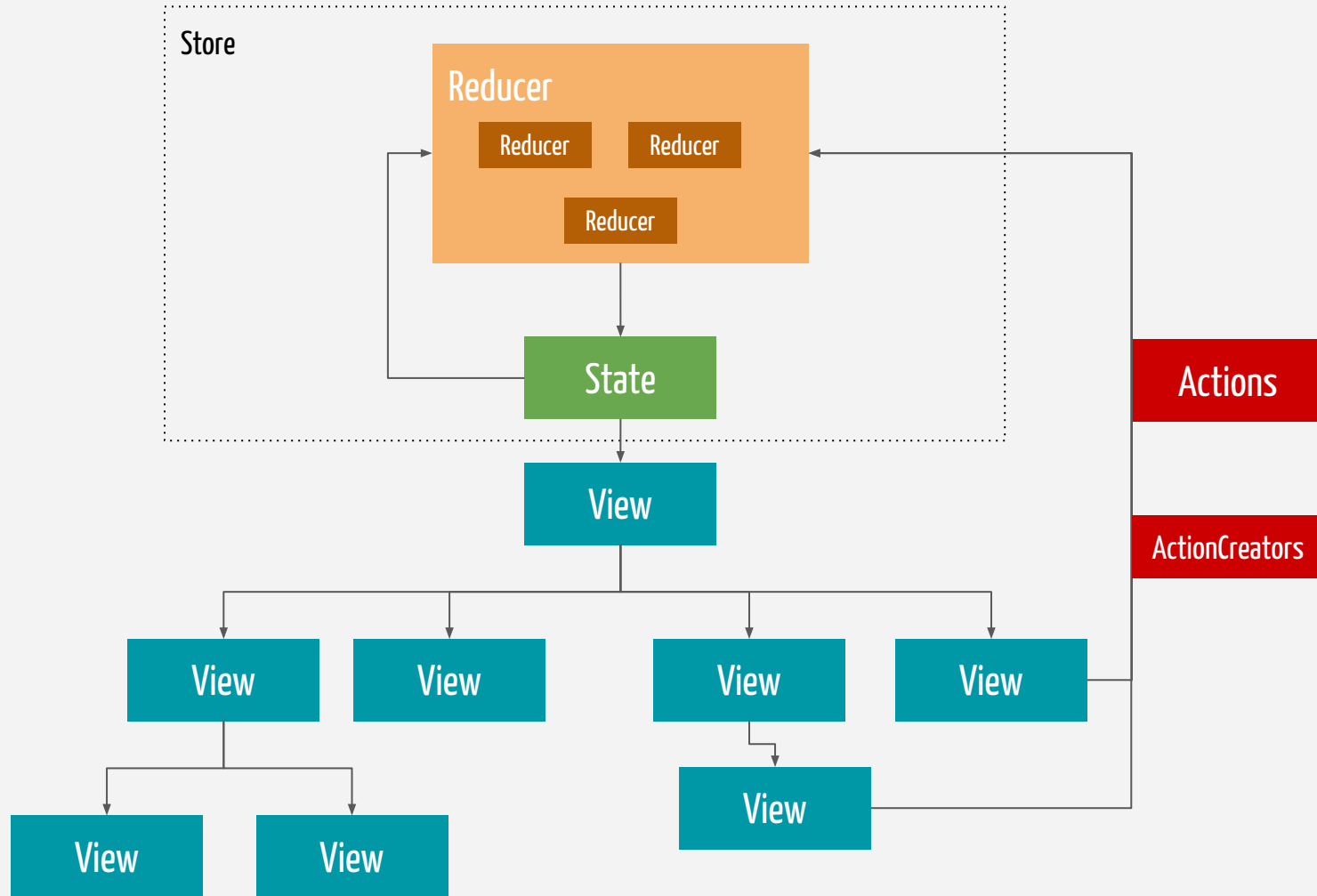
```
var actions = [action1, action2, action3,...];
```

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

**Redux = Flux + Reducer**







# Redux im 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): Reducer<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) {  
  
    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:
      // 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); // erzeugt "Kopie"
```

# 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, {loading: true}}; // no compile error in TypeScript
```

# Immutability in JavaScript / TypeScript?

```
// ES7 / TypeScript 2.1
```

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

```
// npm install tassign
```

```
const newState = tassign(state, {loading: 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;
  }
}
```

- Der gesamte Zustand in einer einzigen Datenstruktur?
- Ein Reducer enthält sämtliche Logik?



# Reducer Composition

- Der gesamte Zustand in einer einzigen Datenstruktur?
- Ein Reducer enthält sämtliche Logik?

```
import { combineReducers } from 'redux'

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

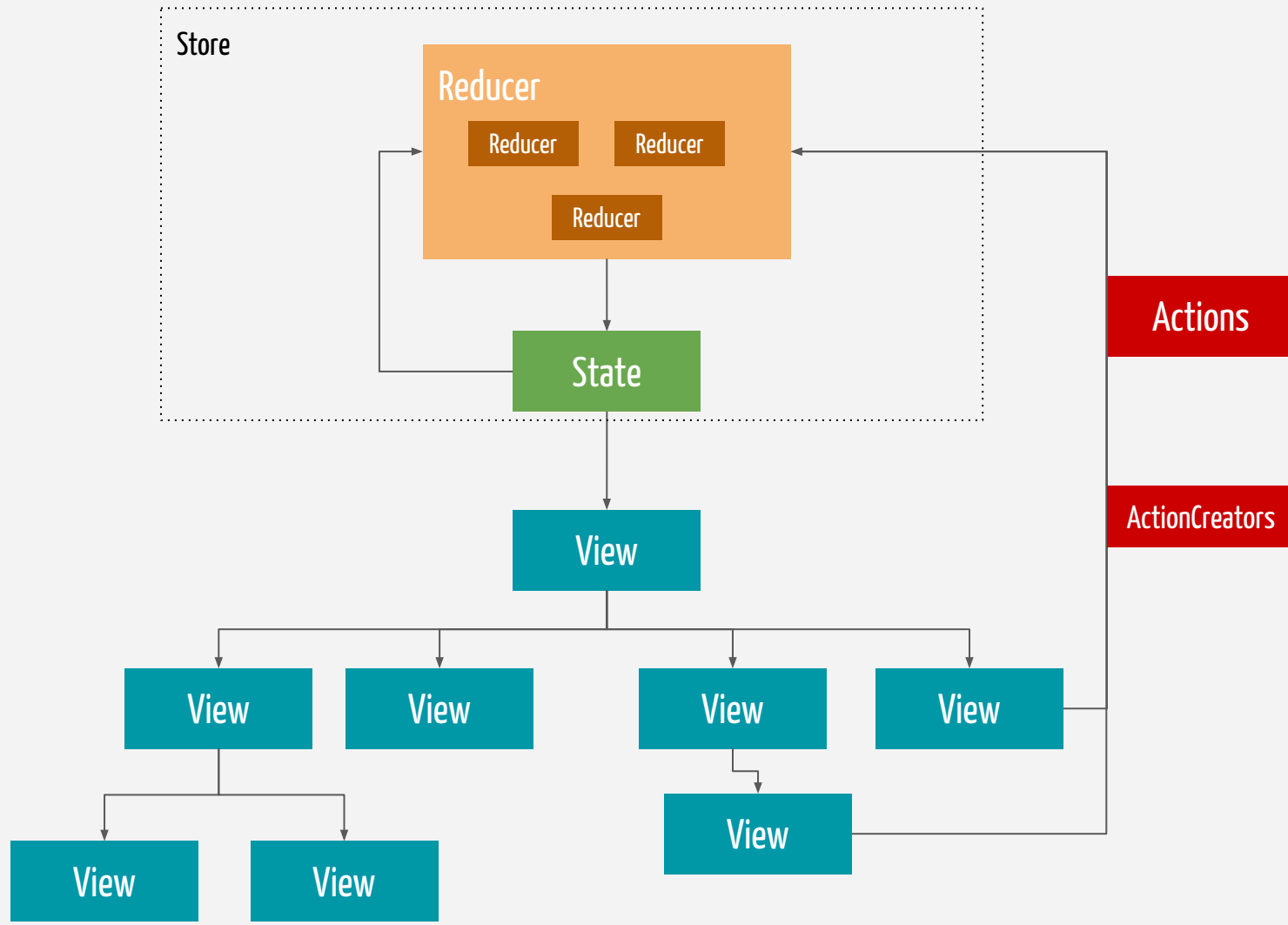
var rootReducer = combineReducers({
  users: userReducer,
  products: productsReducer,
  categories: categoriesReducer,
})
```

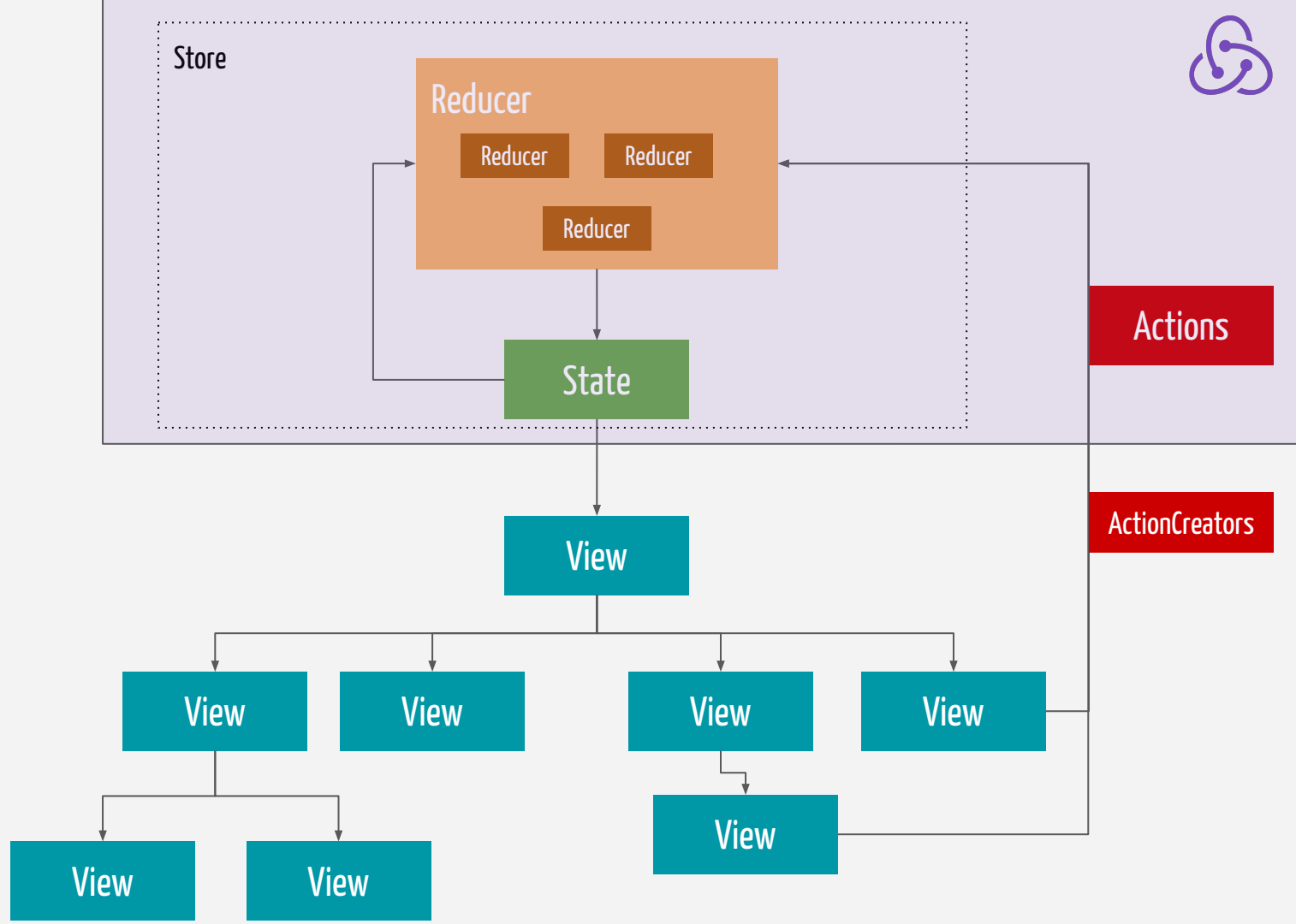


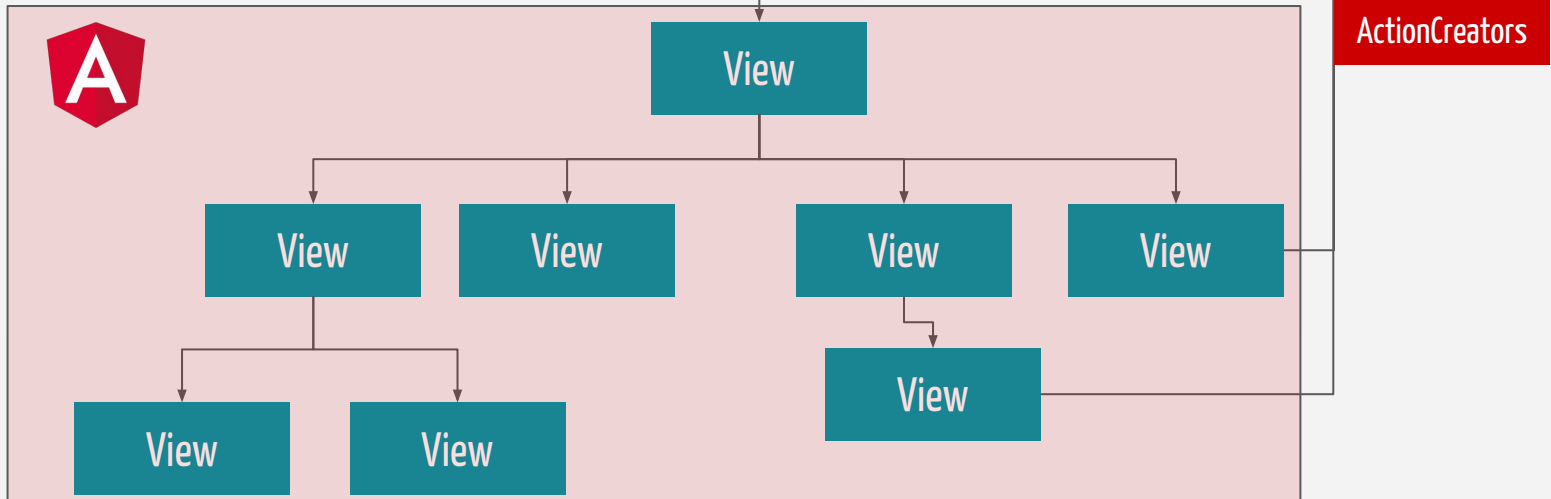
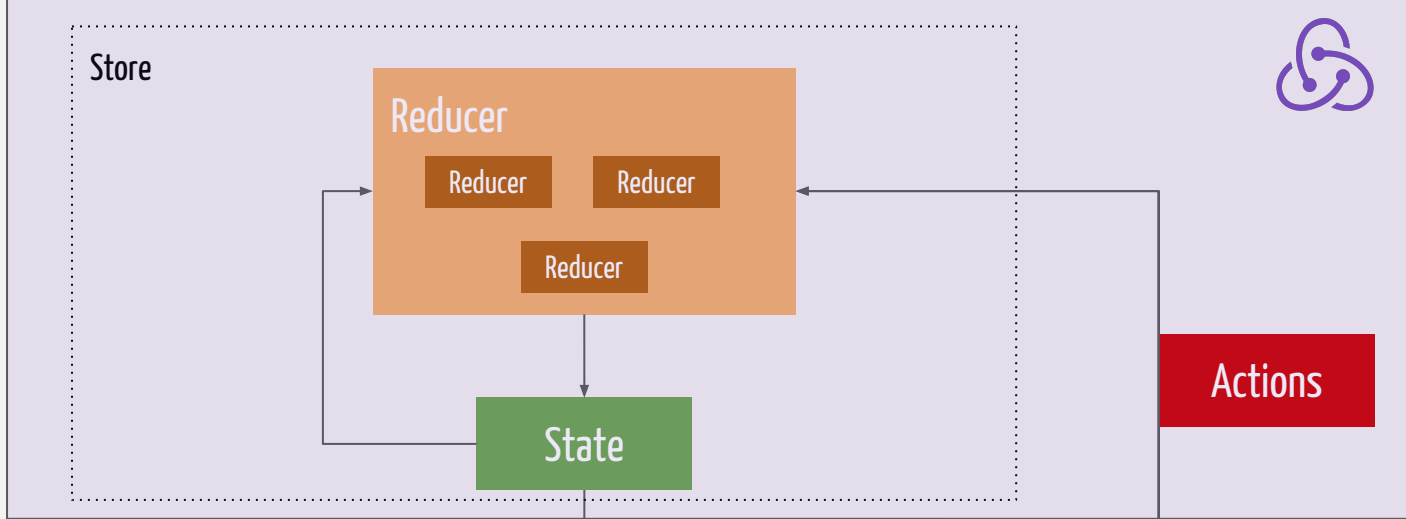
+

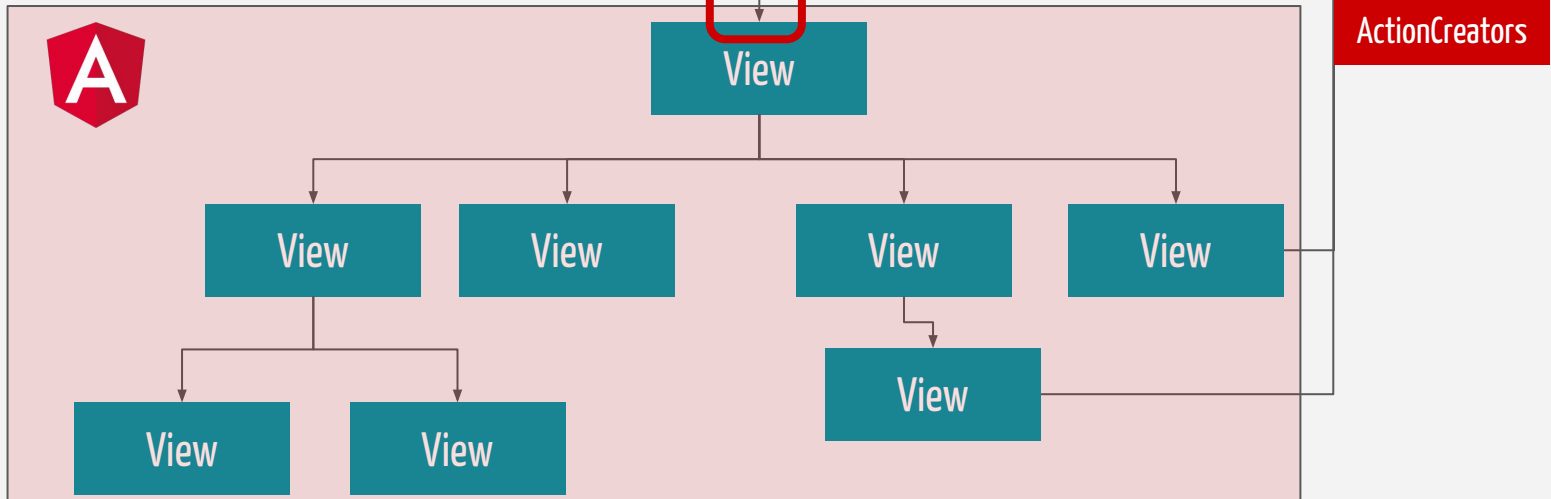
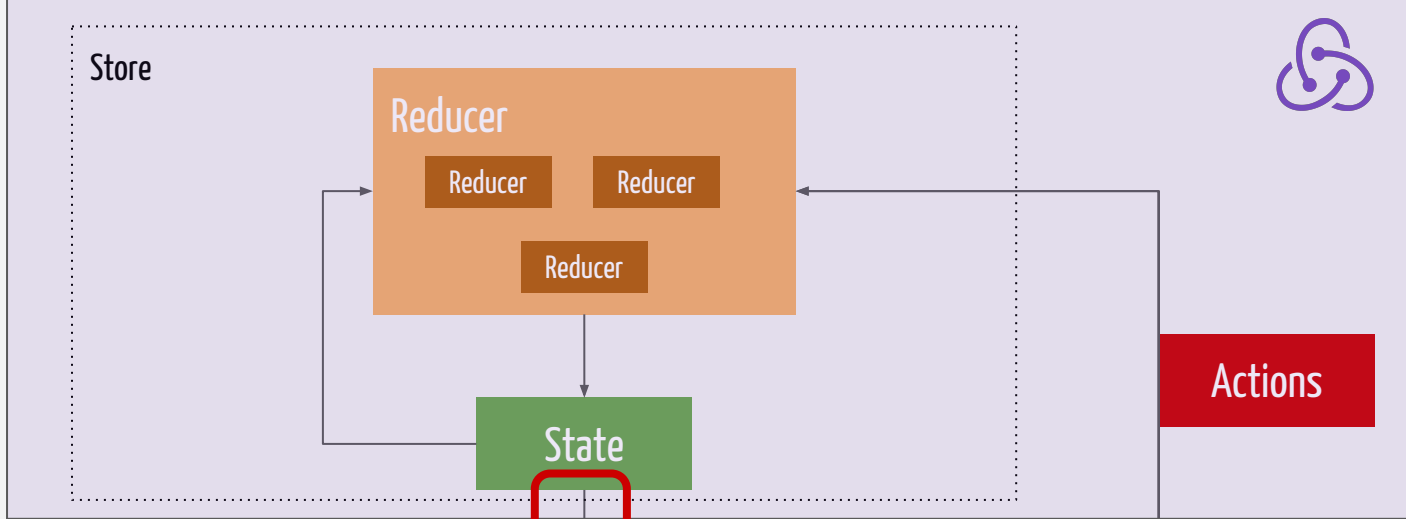


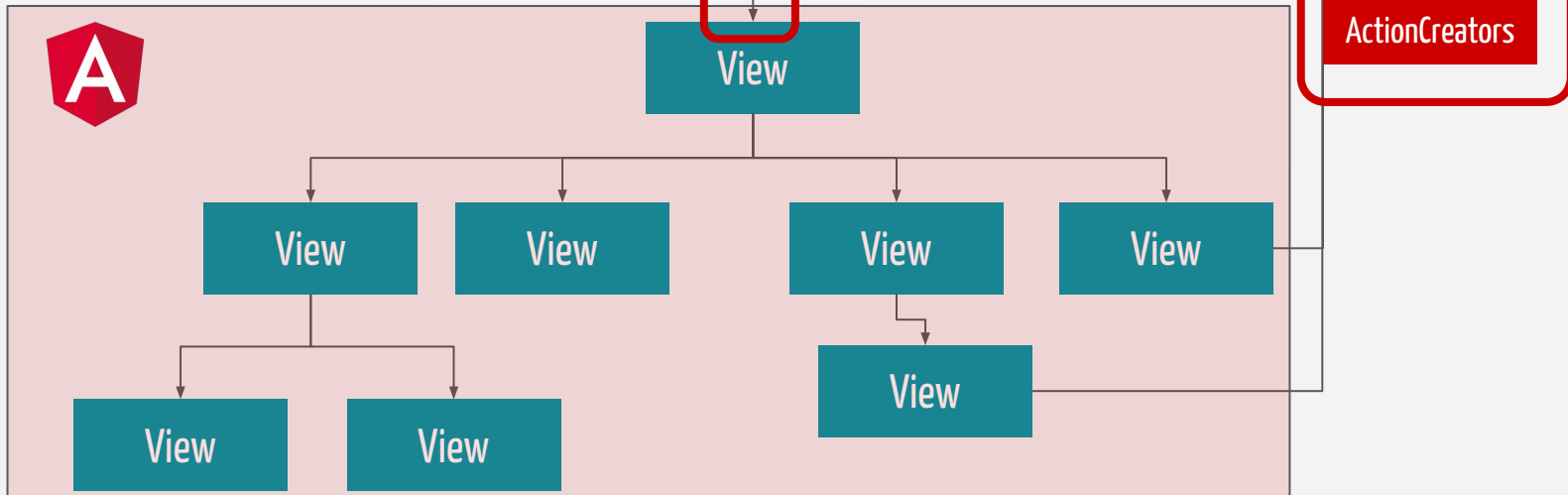
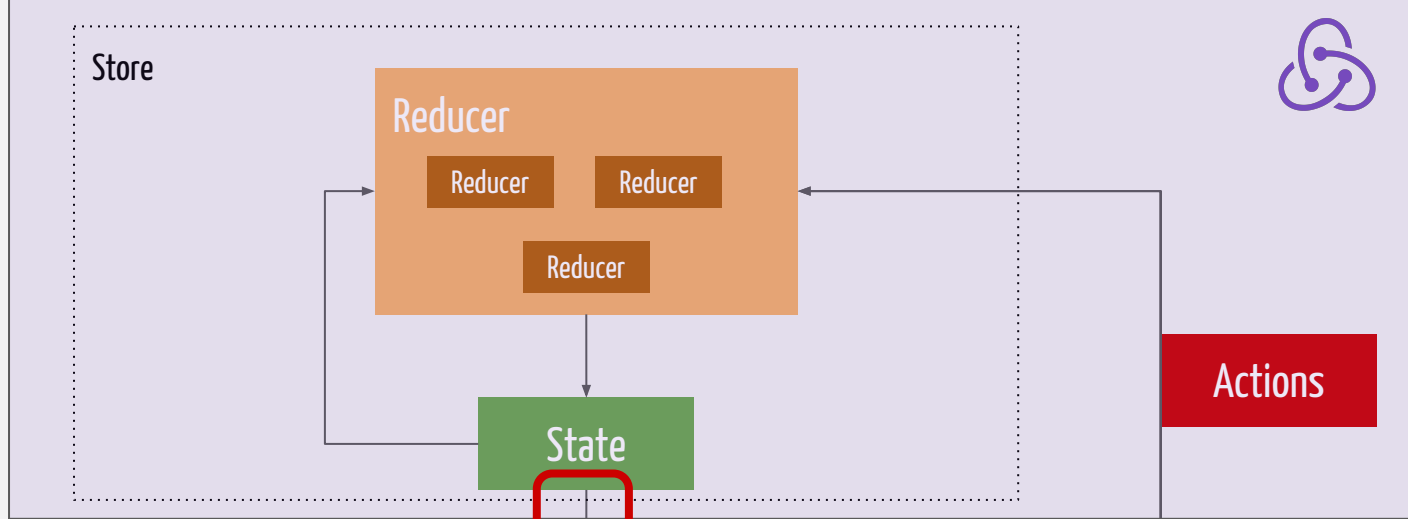
Angular + Redux















angular-redux

# Angular + Redux?

- <https://github.com/angular-redux>
- Verbindet original Redux-Bibliothek mit Angular
- Offene Punkte:
  - ActionCreators benötigen Dependency-Injection
  - Wie kommen die Daten aus dem Store in die UI-Komponenten?

# 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
          }
        });
      });
  }
}
```

Wie kommen Daten aus dem Store  
in die  
UI-Komponenten?

# Selector

- Konzept bei react-redux erprobt
- Selector: Funktion, die aus dem State bestimmte Daten herausholt (selektiert)
- $(state) \rightarrow T$

```
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>
```

# Fazit



# Debugging

- Time-Travel-Debugging
- Gesamter Anwendungszustand auf einem Blick sichtbar
- Klarer Lösungsweg um Bugs zu finden

Bug



*Fehlerquelle*

Bug



Nutzerinteraktion → Werden  
die richtigen Actions erzeugt?

*Fehlerquelle*

Nutzerinteraktion → Werden  
die richtigen Actions erzeugt?

Bug



*Fehlerquelle*

ActionCreator / UI-Komponente

Bug

*Fehlerquelle*

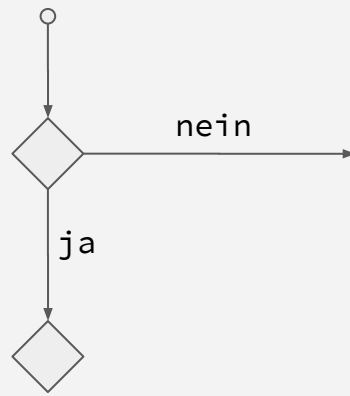
Nutzerinteraktion → Werden  
die richtigen Actions erzeugt?

nein

ActionCreator / UI-Komponente

ja

Ist der neue State korrekt?



Bug

*Fehlerquelle*

Nutzerinteraktion → Werden  
die richtigen Actions erzeugt?

nein

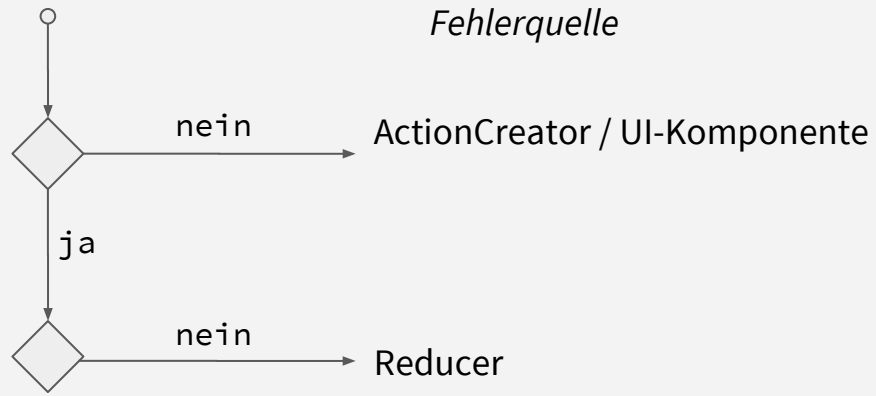
ActionCreator / UI-Komponente

ja

Ist der neue State korrekt?

nein

Reducer



Bug

*Fehlerquelle*

Nutzerinteraktion → Werden  
die richtigen Actions erzeugt?

nein

ActionCreator / UI-Komponente

ja

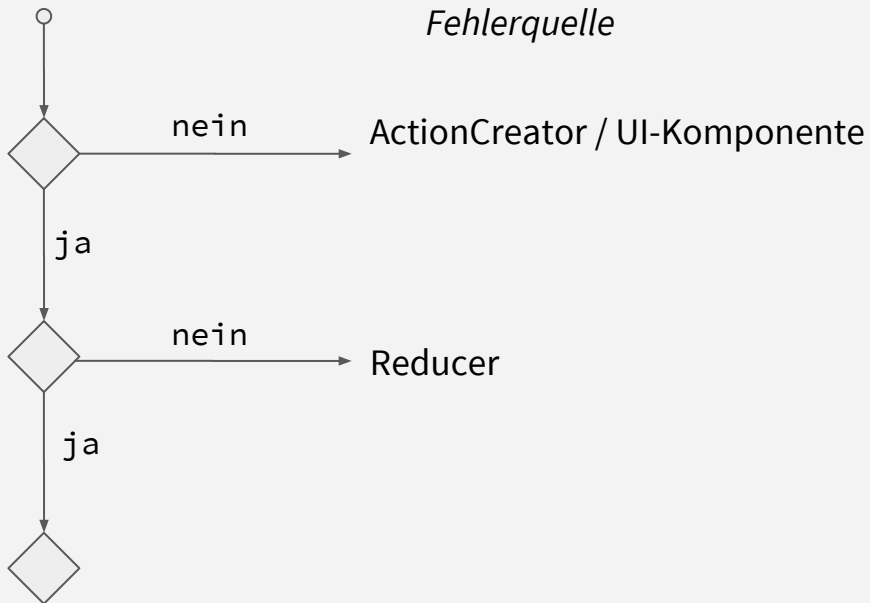
Ist der neue State korrekt?

nein

Reducer

ja

Liefert Selector korrekte  
Daten?



Bug

*Fehlerquelle*

Nutzerinteraktion → Werden  
die richtigen Actions erzeugt?

nein

ActionCreator / UI-Komponente

ja

Ist der neue State korrekt?

nein

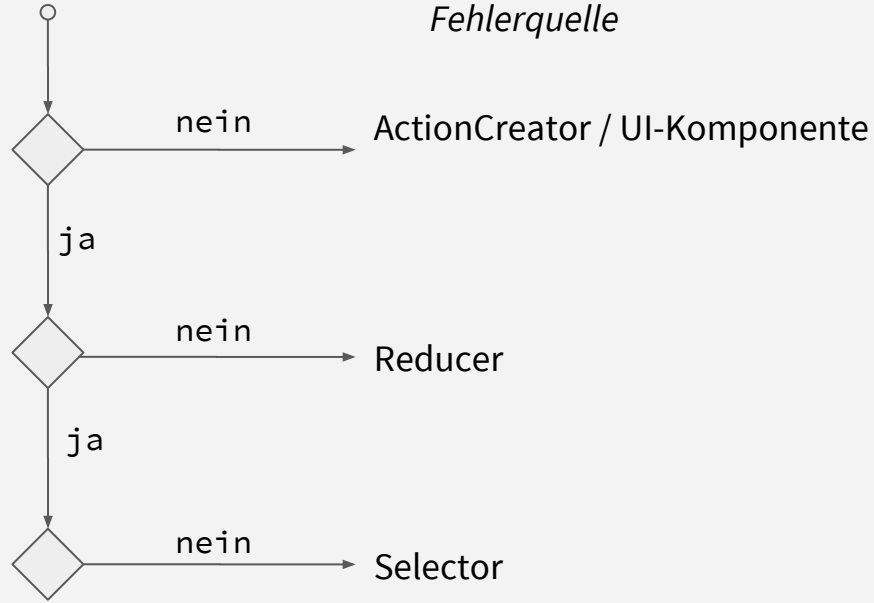
Reducer

ja

Liefert Selector korrekte  
Daten?

nein

Selector





Bug

*Fehlerquelle*

Nutzerinteraktion → Werden  
die richtigen Actions erzeugt?

nein

ActionCreator / UI-Komponente

ja

Ist der neue State korrekt?

nein

Reducer

ja

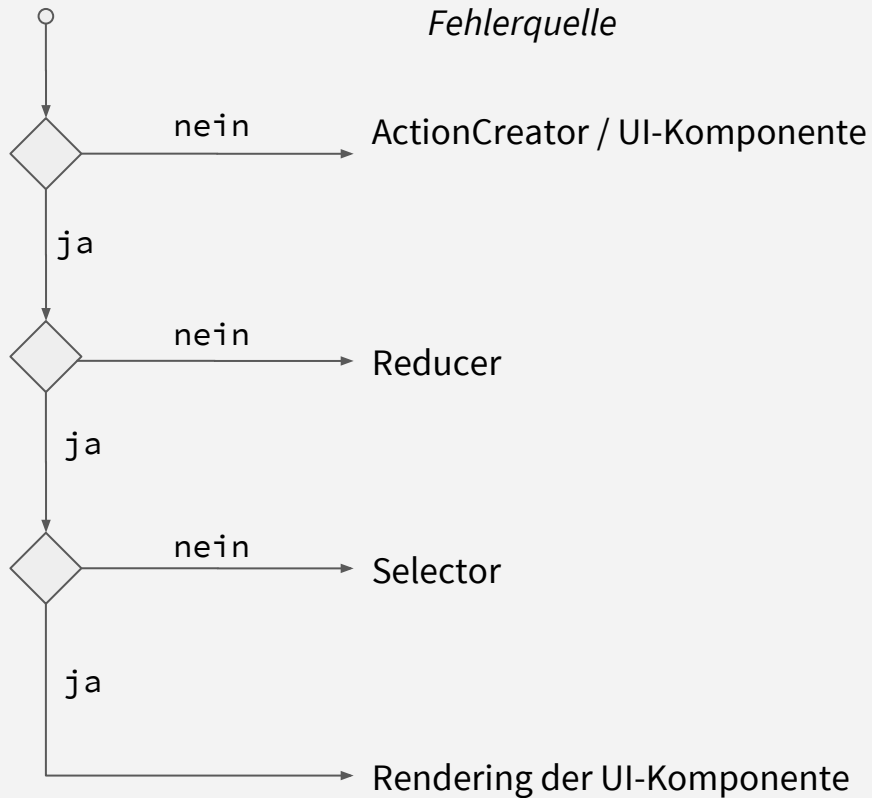
Liefert Selector korrekte  
Daten?

nein

Selector

ja

Rendering der UI-Komponente



# Mehr Dateien

- Reducer, Selectors, ActionTypes, ActionCreators...
- Einstiegshürde: Neue Entwickler finden sich schlechter zurecht
- Klare Zuständigkeiten
- Mit etwas Übung findet man sich sogar besser zurecht

# Testbarkeit

- Reducer und Selectors sind pure Funktionen
- Asynchronität isoliert in ActionCreators
- UI-Komponenten sind konzeptionell ähnlich zu Funktionen:
  - Daten kommen rein → Wie sieht das Rendering aus?
  - Interaktion durch Nutzer → Werden korrekte ActionCreators aufgerufen (Mocking)

# Denkweise + Modellierung

- Funktionale Denkweise für OOP-Entwickler ungewohnt
  - Pure Functions
  - Function Composition
  - Immutability
  - Reduce-Functions
- Wie modelliere ich den State?
- Wie komponiere ich die Reducer?
- Sämtliche Interaktionen nur per Actions
- Asynchronität mittels ActionCreators

# Einfacher Umstieg React $\longleftrightarrow$ Angular

- Gemeinsame Basis für React- und Angular-Projekte
- viel Code kann unverändert übernommen werden
- Entwickler-Know-How
- Auch auf anderen Plattformen anwendbar:
  - React-Native/NativeScript  $\rightarrow$  Mobile
  - Java Desktop  $\rightarrow$  JavaFX

# Fragen?

 @manuel\_mauky

 [github.com/lestard](https://github.com/lestard)

[www.lestard.eu](http://www.lestard.eu)

**JUG**  
**Görlitz** 



**Saxonia Systems**  
So geht Software.