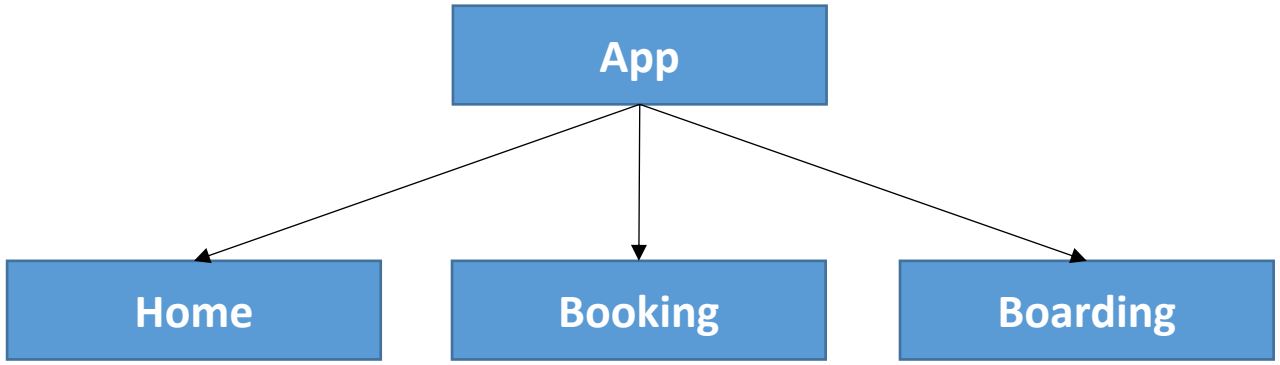
# State Management with Redux und @ngrx/store

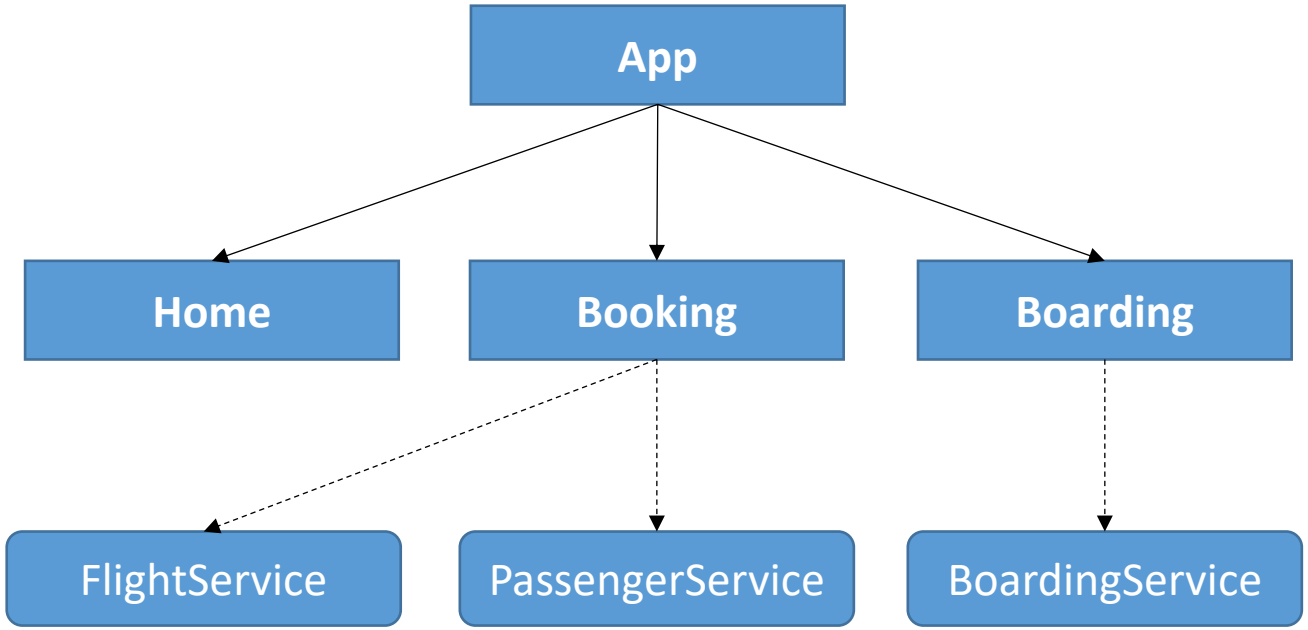
**Alex Thalhammer**

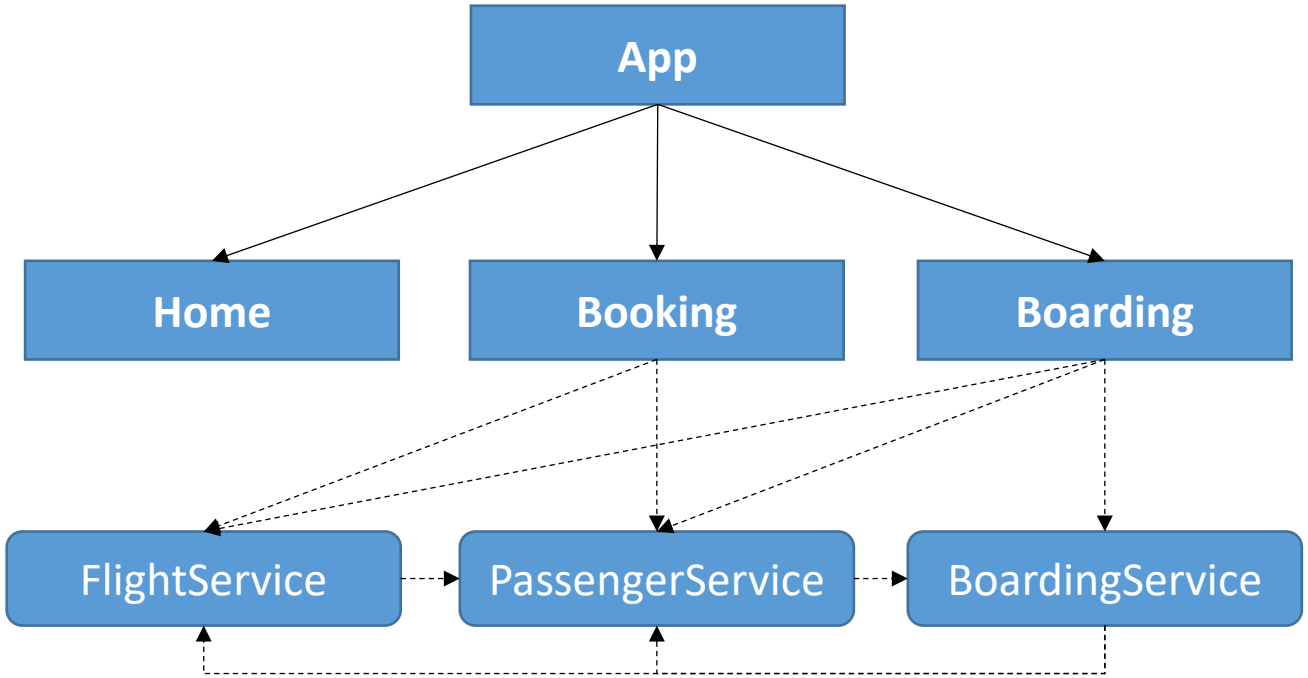
# Contents

- Motivation
- State
- Actions
- Reducer
- Store
- Selectors
- Effects
- Entities


Motivation


App
Home
Booking
Boarding


App
Home
Booking
Boarding
FlightService
PassengerService
BoardingService


App
Home
Booking
Boarding
FlightService
PassengerService
BoardingService

# Redux

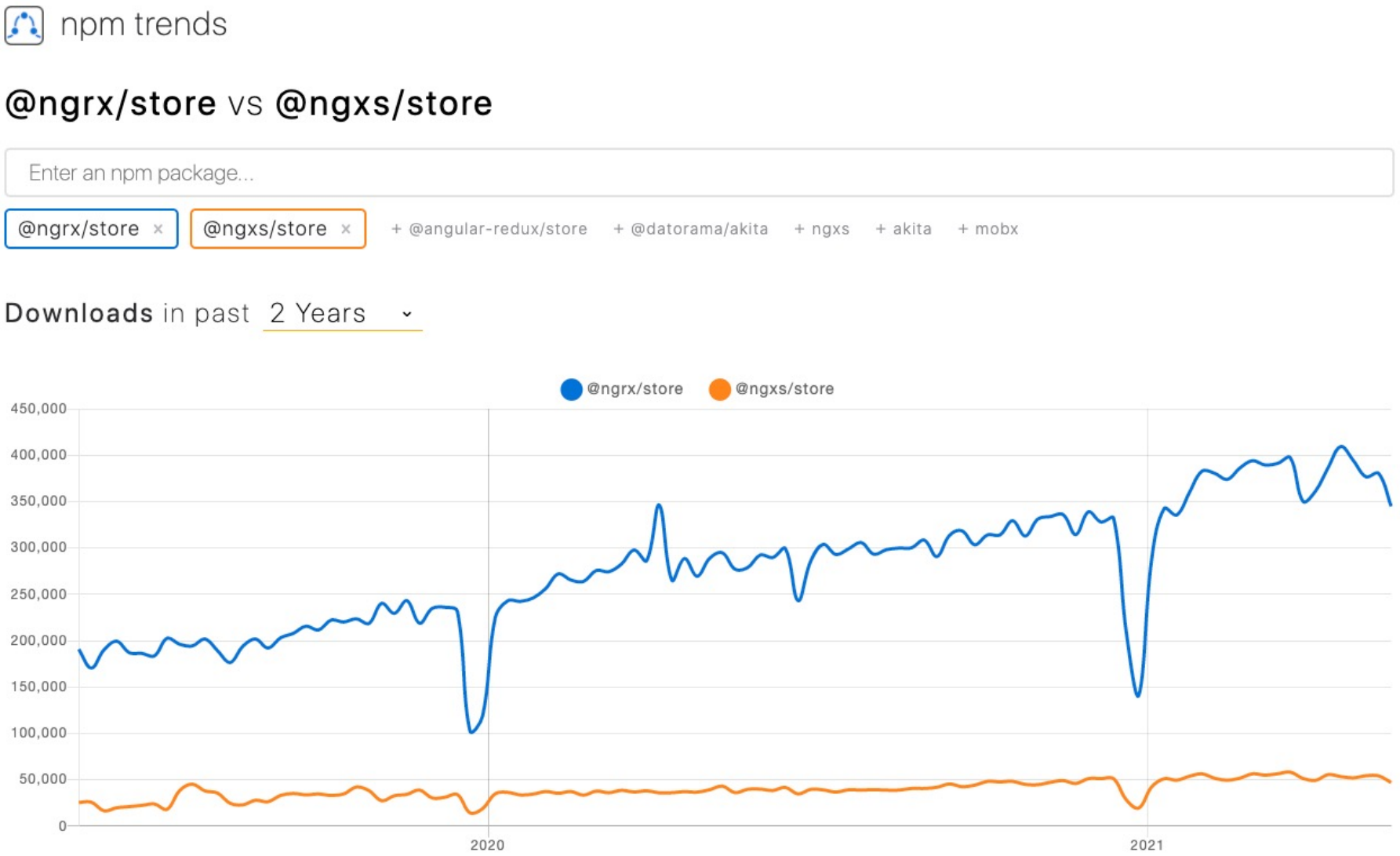
- Redux makes complex UI manageable
- Origin: React Ecosystem
- Implementation used here: @ngrx/store

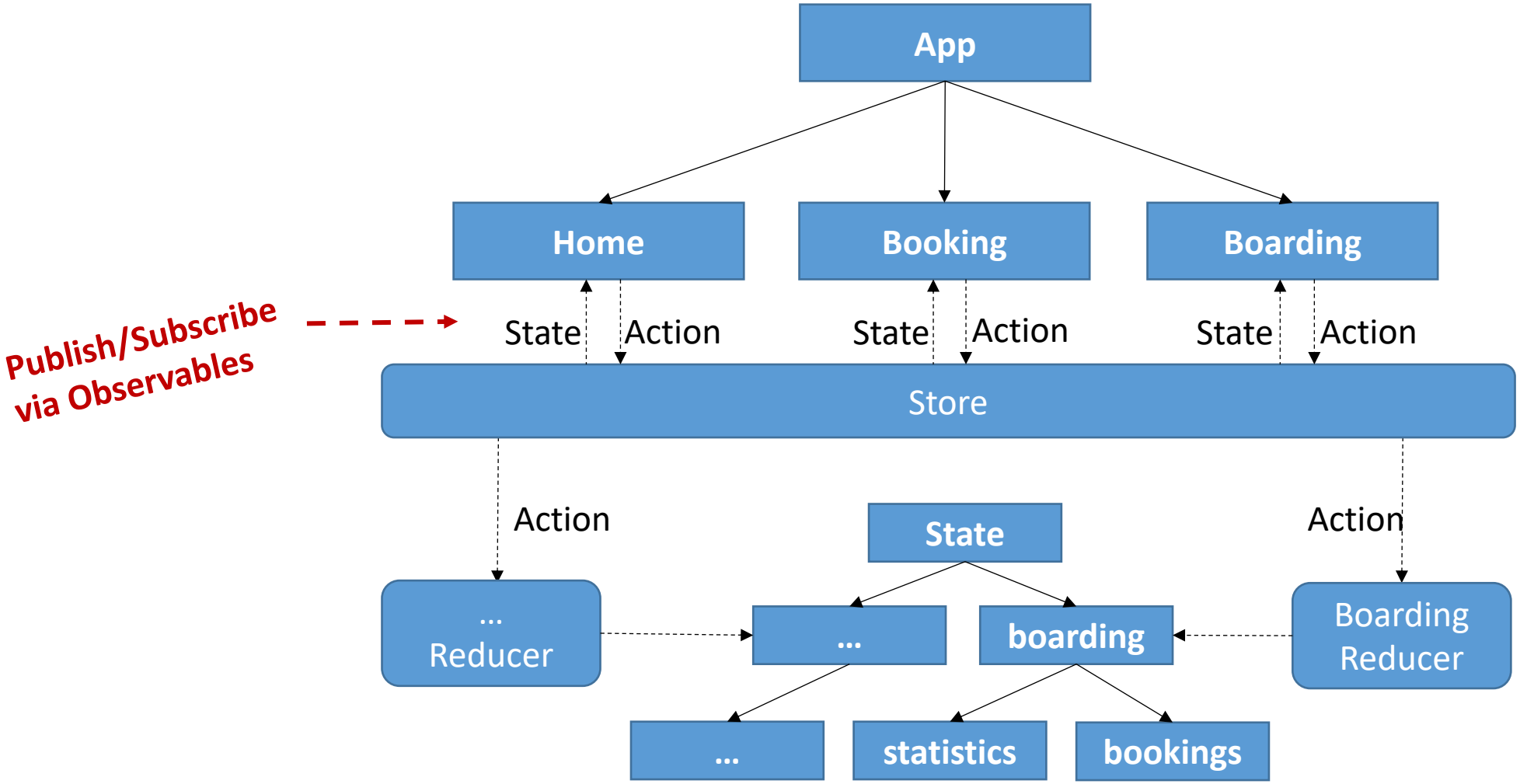
**npm install @ngrx/store --save**

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

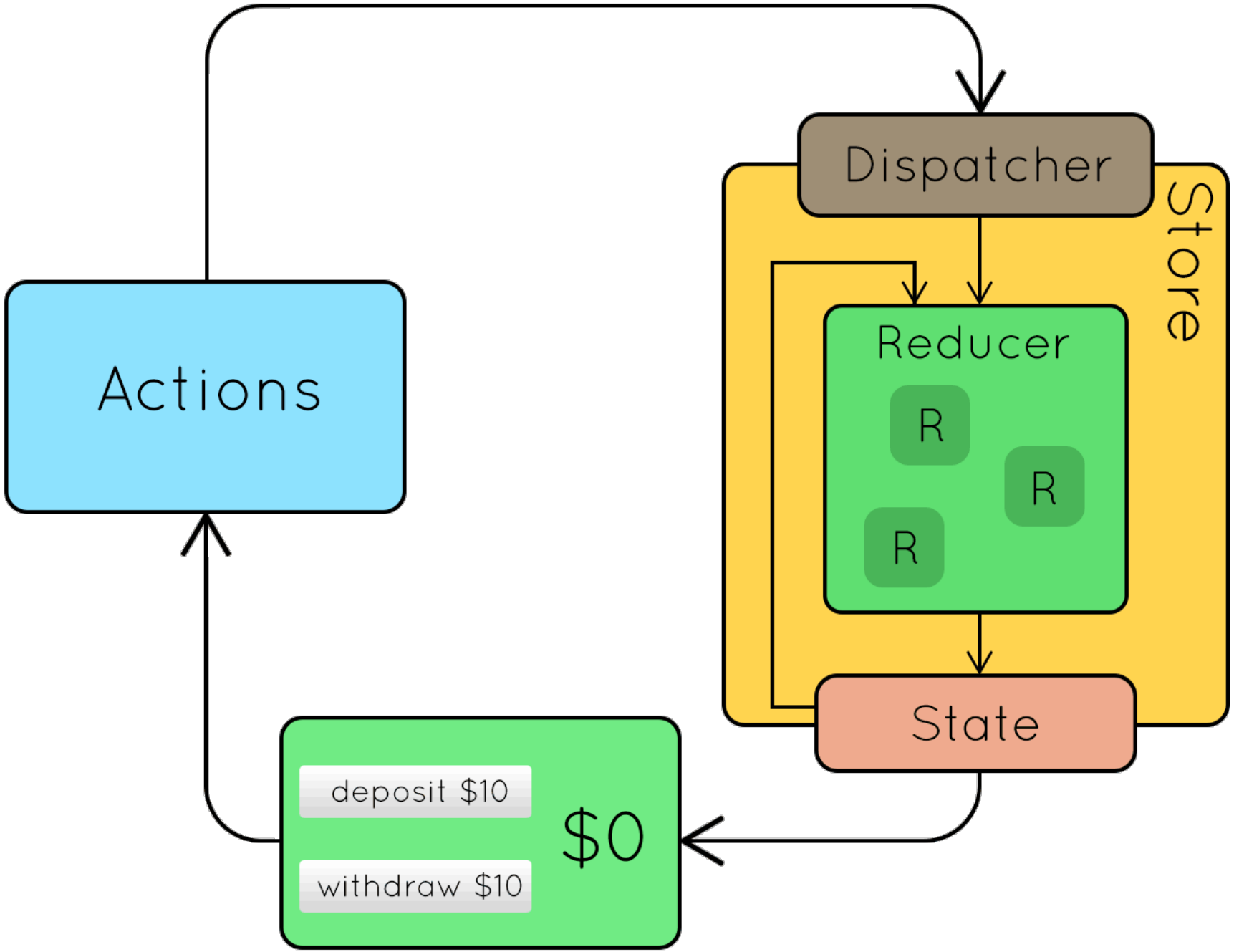
SOFTWARE ARCHITECT

# Alternatives






Single Immutable State Tree


Actions
Dispatcher
Store
Reducer
R
R
R
State
deposit $10
withdraw $10
$0

# State

# State

```
export interface FlightBookingState {
  flights: Flight[];
  statistics: FlightStatistics;
  basket: object;
}
```

# State

```
export interface FlightBookingState {
  flights: Flight[];
  statistics: FlightStatistics;
}

export interface FlightStatistics {
  countDelayed: number;
  countInTime: number;
}
```

# AppState

```
export interface AppState {
  flightBooking: FlightBookingState;
  currentUser: UserState;
}
```


7DAYZ
Allianz
Allianz
Allianz
Title:
Direction:
Scene:
Camera:

# Actions

# Parts of an Action

- Type
- Payload

# Actions

- Actions express *events* that happen throughout your application

- dispatch(flightsLoaded({ flights }))

# Defining an Action

```
export const flightsLoaded = createAction(
    '[FlightBooking] FlightsLoaded',
    props<{flights: Flight[]}>()
);
```

# Reducer

# Reducers

- Reducers are responsible for handling transitions from one state to the next state in your application

- Receive Actions

# Reducer

**(currentState, action) => newState**

# Reducer

- Function that executes Action
- Pure function (stateless, etc.)
- Each Reducer gets each Action
  - Check whether Action is relevant
  - This prevents cycles

# Reducer for FlightBookingState

```
export const flightBookingReducer = createReducer(
    initialState,

    on(flightsLoaded, (state, action) => {
        const flights = action.flights;
        return { ...state, flights };
    })
)
```


Store

# Store

- Manages state tree
- Allows to read state (via Selectors / Observables)
- Allows to modify state by dispatching actions


# Registering @ngrx/store

# Registering @ngrx/Store

```
@NgModule({
   imports: [
      […]
      StoreModule.forRoot(reducers)
   ],
   […]
})
export class AppModule { }
```

# Registering @ngrx/Store

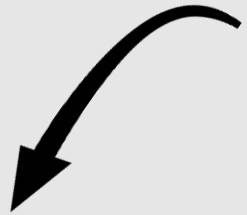
```
@NgModule({
   imports: [
      […]
      StoreModule.forRoot(reducers),
      !environment.production ? StoreDevtoolsModule.instrument() : []
   ],
   […]
})
export class AppModule { }
```

**@ngrx/store-devtools**

# ngrx and Feature Modules

# Registering @ngrx/Store

```
@NgModule({
   imports: [
      […]
      StoreModule.forFeature('flightBooking', flightBookingReducer)
   ],
   […]
})
export class FlightBookingModule { }
```



State branch for feature

# DEMO

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

# Lab

NgRx Store

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

# Selectors

- Selectors are pure functions used for obtaining slices of store state

- select(tree => tree.flightBooking.flights): Observable<Flight[]>

# DEMO

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

# Lab

NgRx Selectors

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

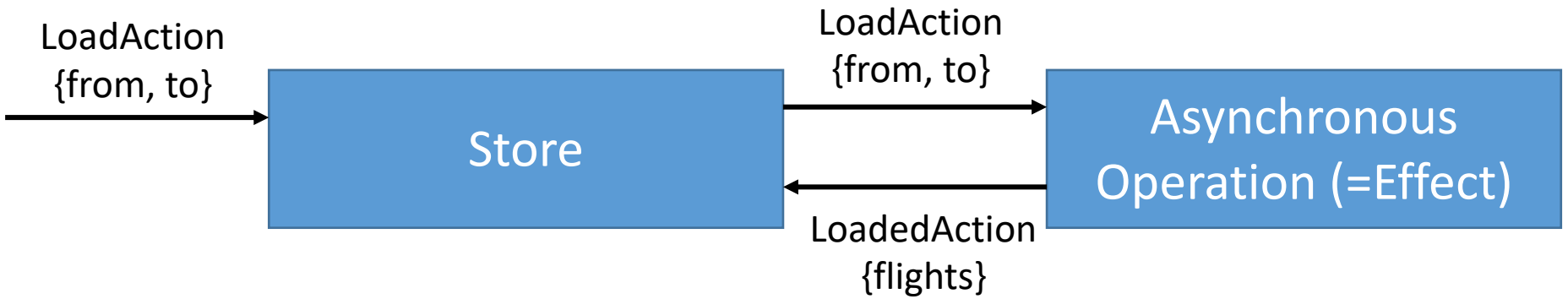
SOFTWARE
ARCHITECT

# Effects

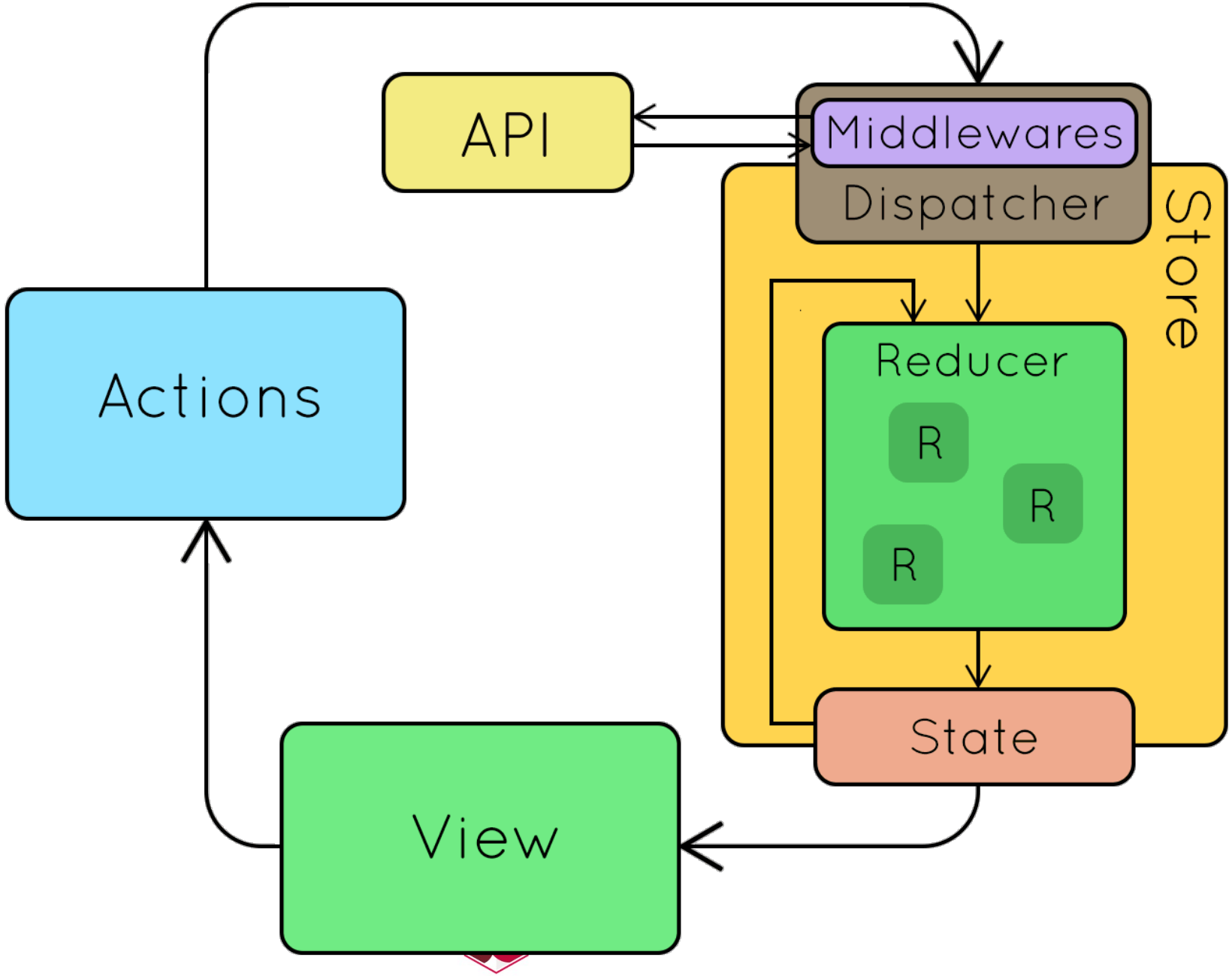
# Challenge

- Reducers are synchronous by definition
- What to do with asynchronous operations?

# Solution: Effects




**ng add @ngrx/effects**


API
Middlewares
Dispatcher
Store
Actions
Reducer
R
R
R
State
View

# Effects are Observables

# Implementing Effects

```
@Injectable()
export class FlightBookingEffects {

    […]

}
```

# Implementing Effects

```
@Injectable()
export class FlightBookingEffects {

  constructor(
    private flightService: FlightService, private actions$: Actions) {
  }

  […]

}
```

# Implementing Effects

```
@Injectable()
export class FlightBookingEffects {

  constructor(
    private flightService: FlightService, private actions$: Actions) {
  }

  myEffect$ = createEffect(() => this.actions$.pipe(
                  ofType(loadFlights));
}
```

# Implementing Effects

```
@Injectable()
export class FlightBookingEffects {

  constructor(
    private flightService: FlightService, private actions$: Actions) {
  }

  myEffect$ = createEffect(() => this.actions$.pipe(
                  ofType(loadFlights),
                  switchMap(a => this.flightService.find(a.from, a.to, a.urgent))));
}
```

# Implementing Effects

```
@Injectable()
export class FlightBookingEffects {

  constructor(
    private flightService: FlightService, private actions$: Actions) {
  }

  myEffect$ = createEffect(() => this.actions$.pipe(
                  ofType(loadFlights),
                  switchMap(a => this.flightService.find(a.from, a.to, a.urgent)),
                  map(flights => flightsLoaded({flights}))));

}
```

# Implementing Effects

```
@NgModule({
  imports: [
    StoreModule.provideStore(appReducer, initialAppState),
    EffectsModule.forRoot([SharedEffects]),
    StoreDevtoolsModule.instrument()
  ],
  [...]
})
export class AppModule { }
```

# Implementing Effects

```
@NgModule({
  imports: [
    […]
    EffectsModule.forFeature([FlightBookingEffects])
  ],
  [...]
})
export class FeatureModule {
}
```

# DEMO

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

# Lab

NgRx Effects

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE ARCHITECT

# @ngrx/entity and @ngrx/schematics

- ng add @ngrx/entity
- ng add @ngrx/schematics
- ng g module passengers
- ng g entity Passenger --module passengers.module.ts

# DEMO

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

# Lab

NgRx Entities & Schematics

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

# @ngrx/store-devtools

- Add Chrome / Firefox extension to use Store Devtools
  - Works with Redux & NgRx
  - https://ngrx.io/guide/store-devtools

- Also available for MobX
  - MobX Developer Tools

# DEMO

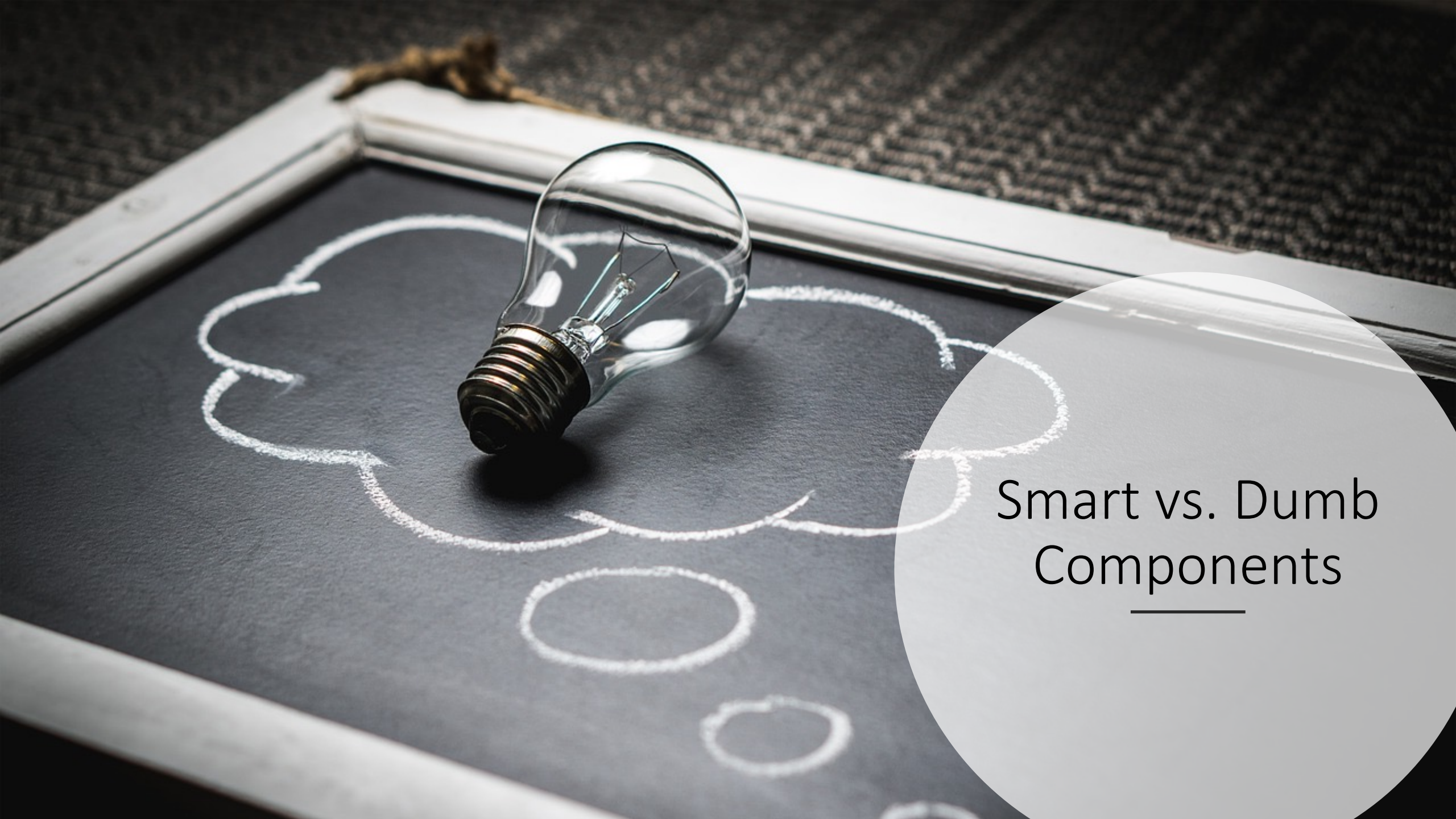
ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

# Like this topic?

Check out the NgRx Guide

https://ngrx.io/guide/



# Smart vs. Dumb Components

# Thought experiment

- What if <flight-card> would directly talk with the store?
  - Querying specific parts of the state
  - Triggering effects
- Traceability?
- Performance?
- Reuse?

# Smart vs. Dumb Components

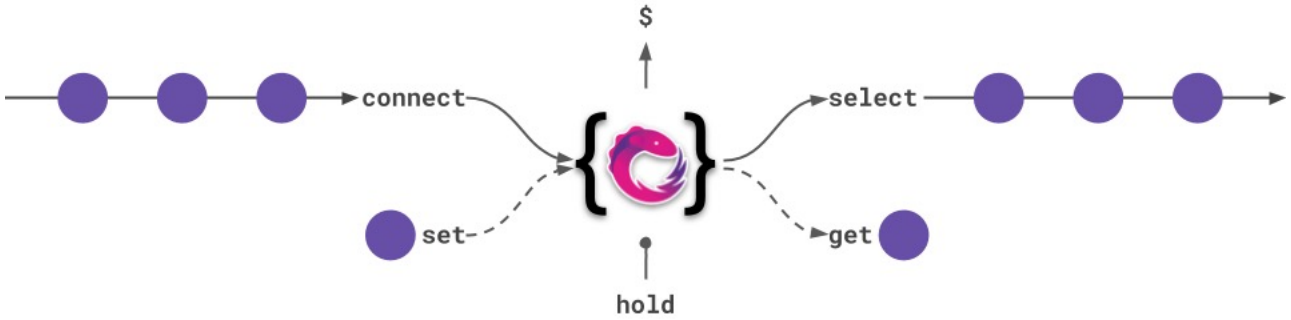
| Smart Component | Dumb |
| --- | --- |
| • Drives the "Use Case"<br>• Usually a "Container" | • Independent of Use Case<br>• Reusable<br>• Usually a "Leaf" |

# Challenges task in Angular components

- reacting to events from different sources
- transforming and composing state
- handling state lifetime
- handling subscriptions

# @rx-angular/state I

- It's a
  - lightweight
  - flexible
  - strongly typed




- tool for managing component state in Angular
- with focus on runtime performance and template rendering

# @rx-angular/state II

- merge global into local state
- shared state selections
- subscription-less interaction
- hook into imperative functions
  - component lifecycle
  - HostBindings

# @rx-angular/state III

- ng add @rx-angular/state

- For getting started play with the tutorials: https://github.com/rx-angular/rx-angular/tree/master/apps/demos/src/app/features/tutorials/basics

- There is also another npm package for reactive rendering called @rx-angular/template