



State Management with Redux und @ngrx/store

Alex Thalhammer

Contents


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



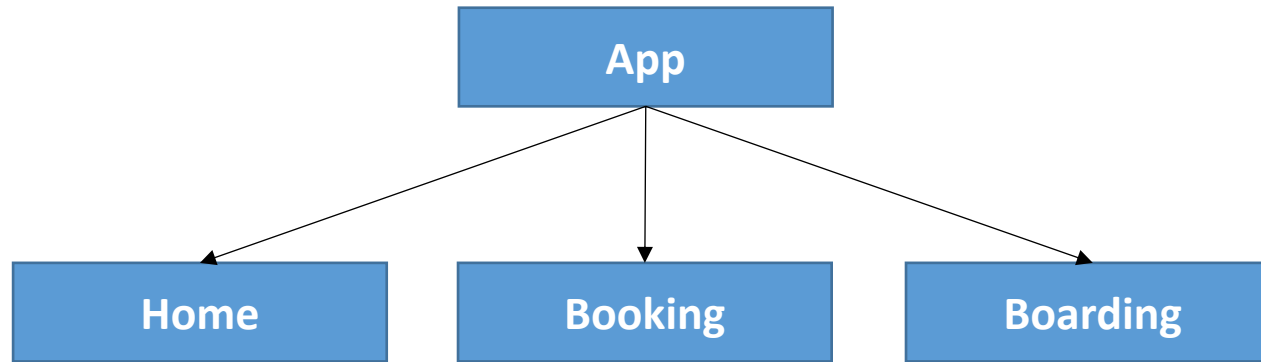
ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

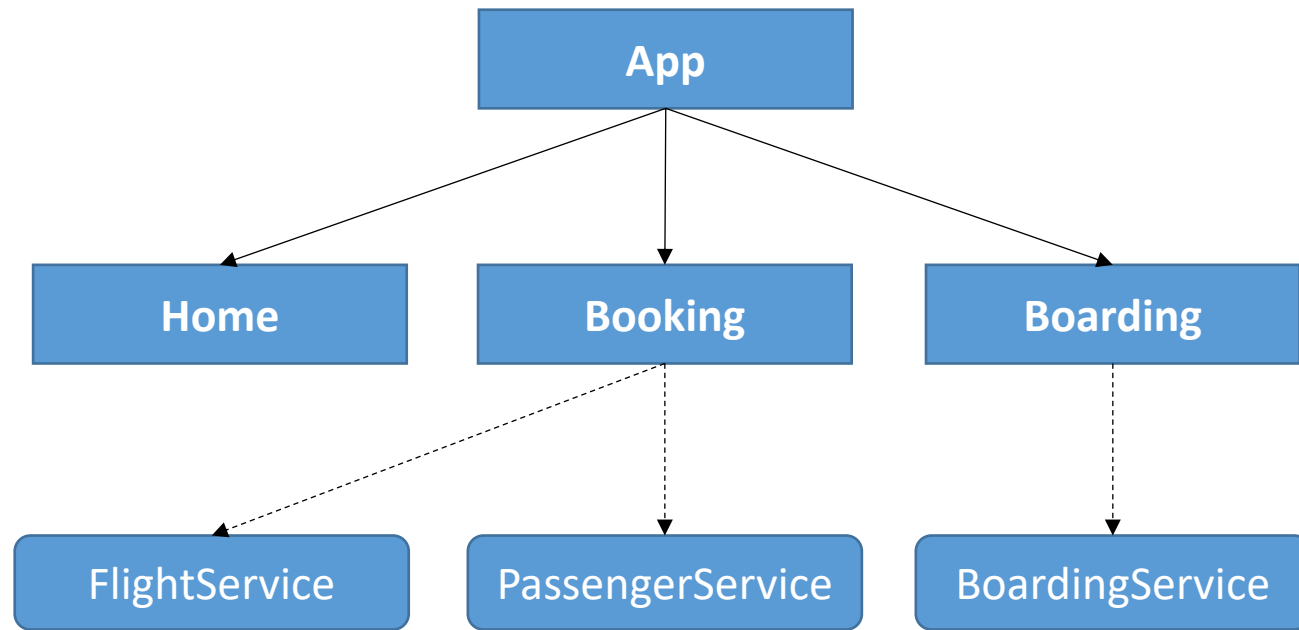


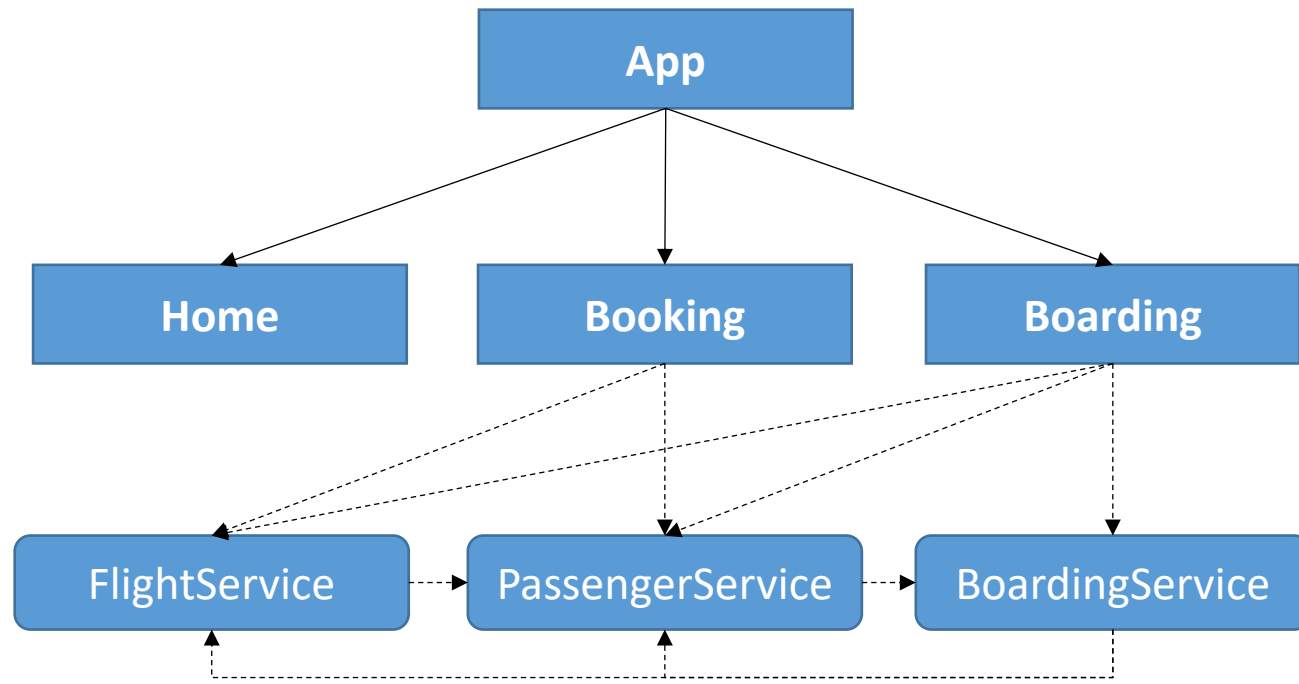
SOFTWARE
ARCHITECT

A full-page background image featuring a silhouette of a person standing on a rock in the middle of the ocean at sunset. The person's arms are outstretched horizontally, mirroring the horizon line. The sky is filled with dramatic, colorful clouds in shades of orange, red, and blue. The sun is a bright point on the horizon, and its reflection is clearly visible in the calm water. The overall mood is one of tranquility and inspiration.

Motivation







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



@ngrx/store vs @ngxs/store

Enter an npm package...

@ngrx/store x

@ngxs/store x

+ @angular-redux/store

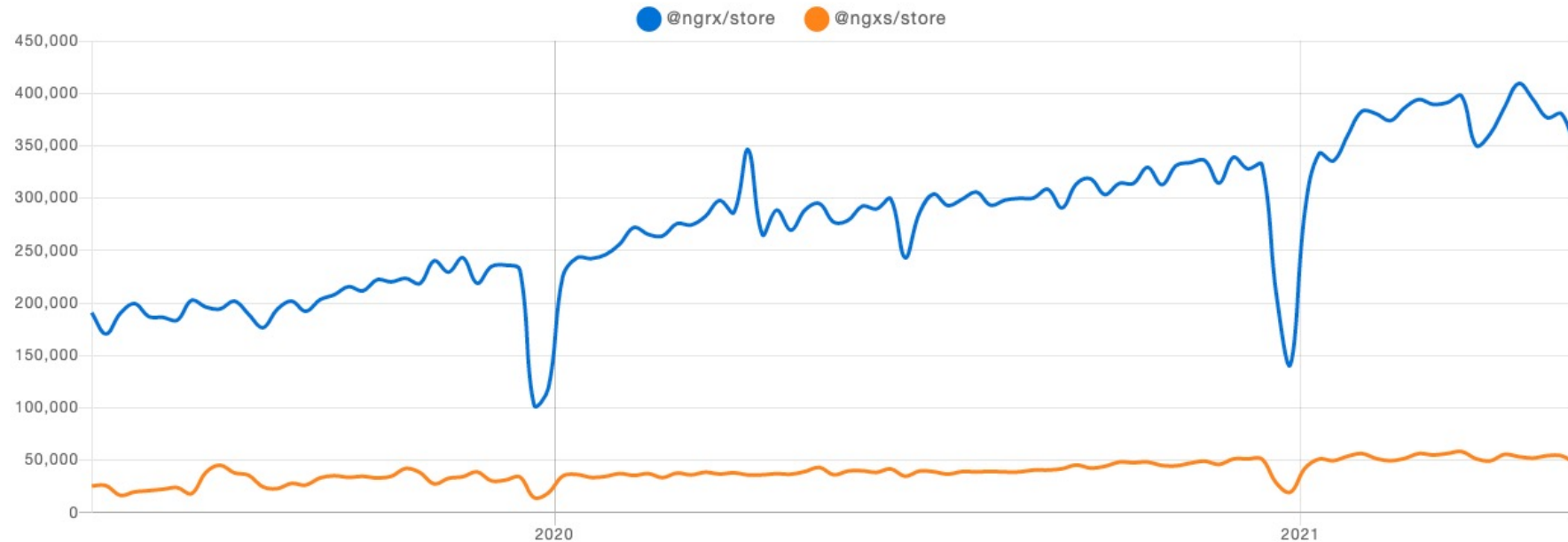
+ @datorama/akita

+ ngxs

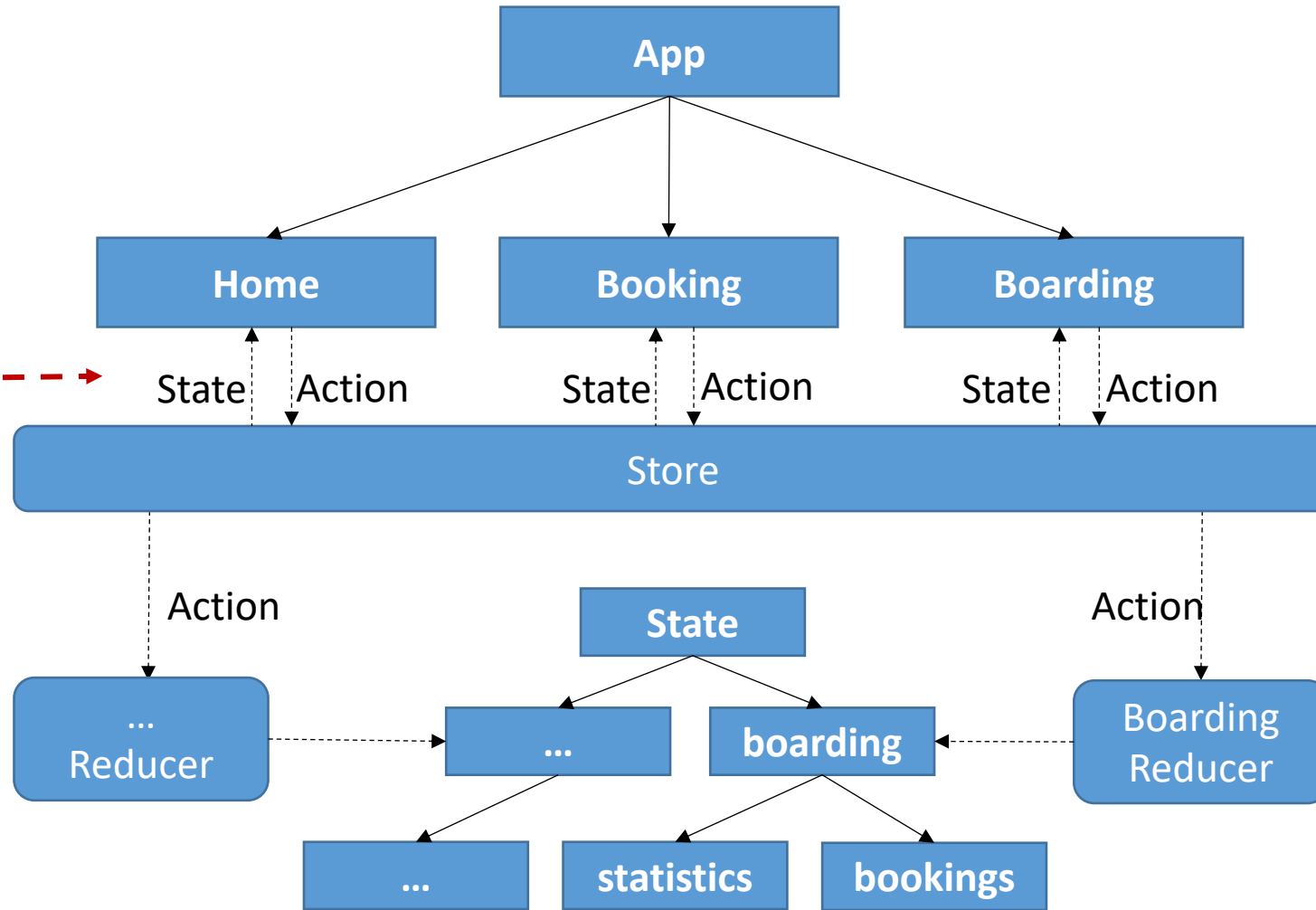
+ akita

+ mobx

Downloads in past 2 Years ▾



*Publish/Subscribe
via Observables*



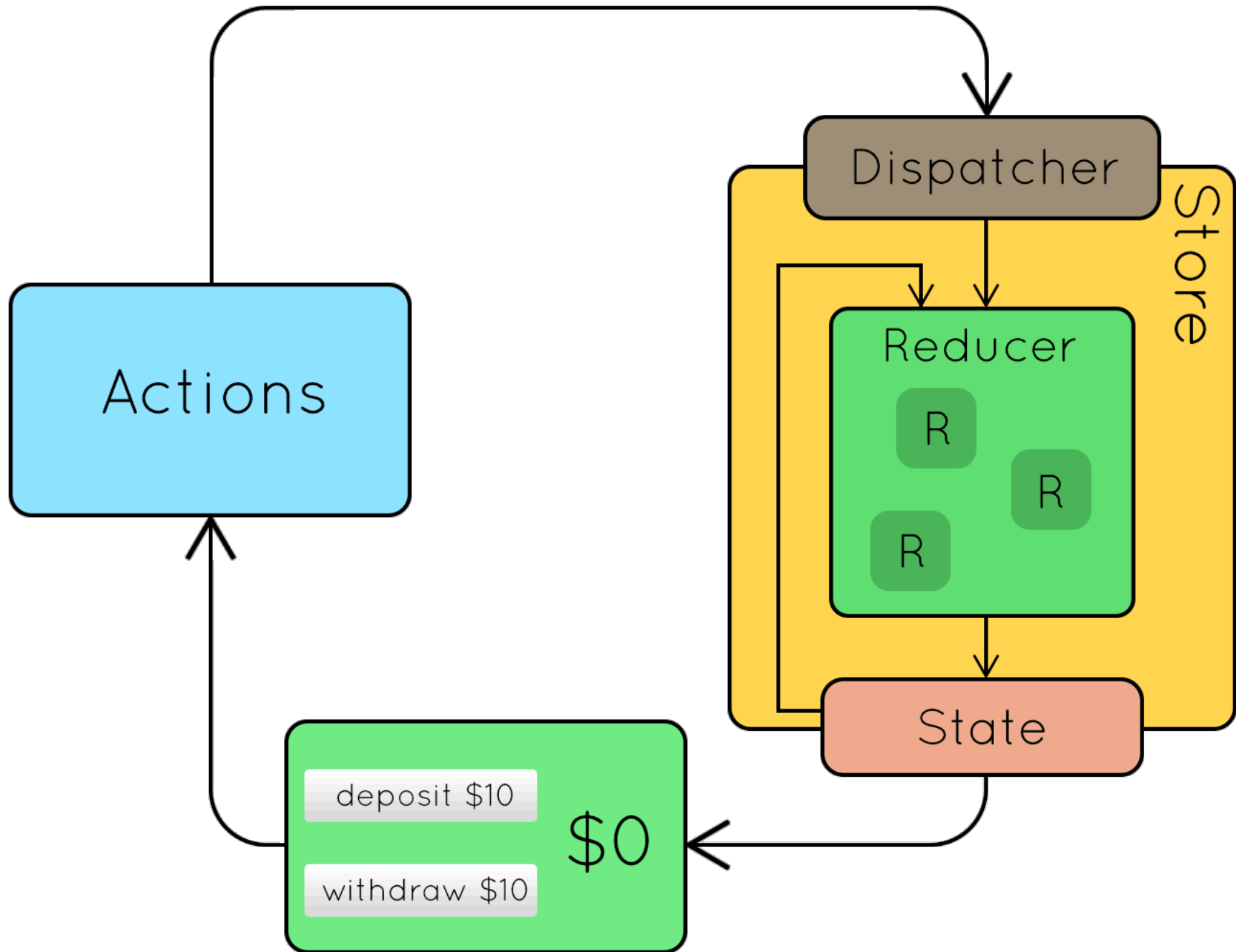
Single Immutable State Tree



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT



State

State

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



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

State

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

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



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

AppState

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



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT



Actions

Parts of an Action

- Type
- Payload



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Actions

- Actions express *events* that happen throughout your application
- `dispatch(flightsLoaded({ flights })))`



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Defining an Action

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



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Reducer



Reducers

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



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Reducer

(currentState, action) => newState



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Reducer

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



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Reducer for FlightBookingState

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



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT



Store

Store

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



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT



Registering @ngrx/store

Registering @ngrx/Store

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



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Registering @ngrx/Store

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

@ngrx/store-devtools



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

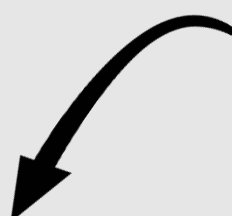


ngrx and Feature Modules

Registering @ngrx/Store

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

State branch for feature



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

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[]>`



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

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?

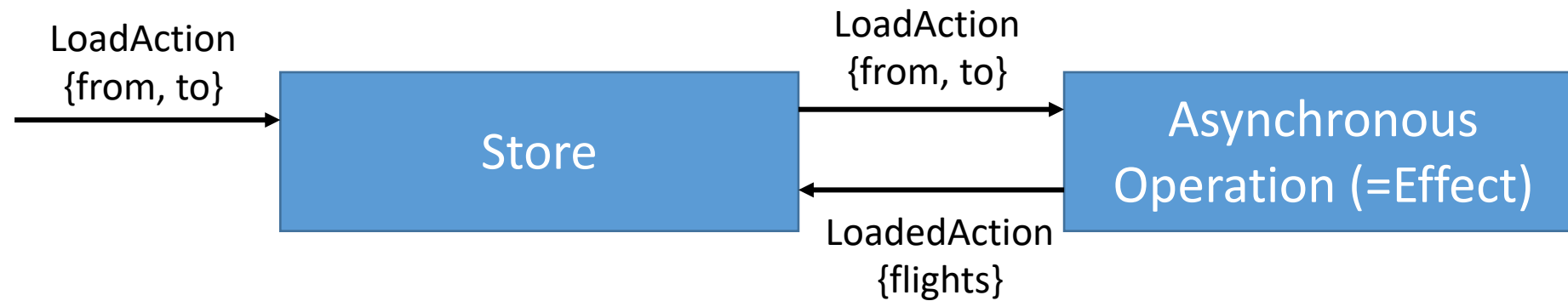


ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Solution: Effects



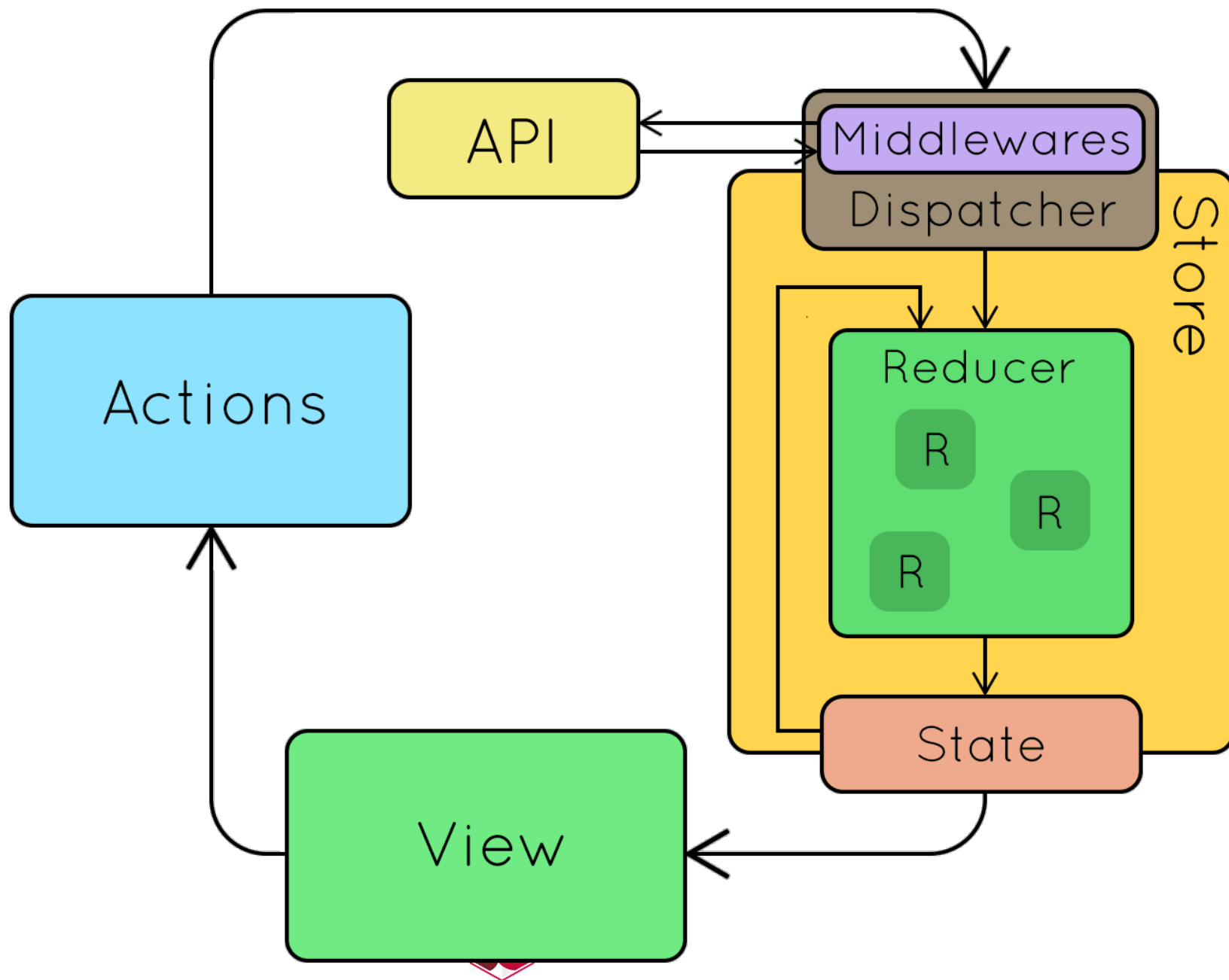
ng add @ngrx/effects



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT



Effects are Observables



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Implementing Effects

```
@Injectable()
export class FlightBookingEffects {

  [...]

}
```



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Implementing Effects

```
@Injectable()
export class FlightBookingEffects {

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

  [...]

}
```



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Implementing Effects

```
@Injectable()
export class FlightBookingEffects {

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

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



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

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)))));
}
```



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

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})))));
}
```



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Implementing Effects

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



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

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



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

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



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

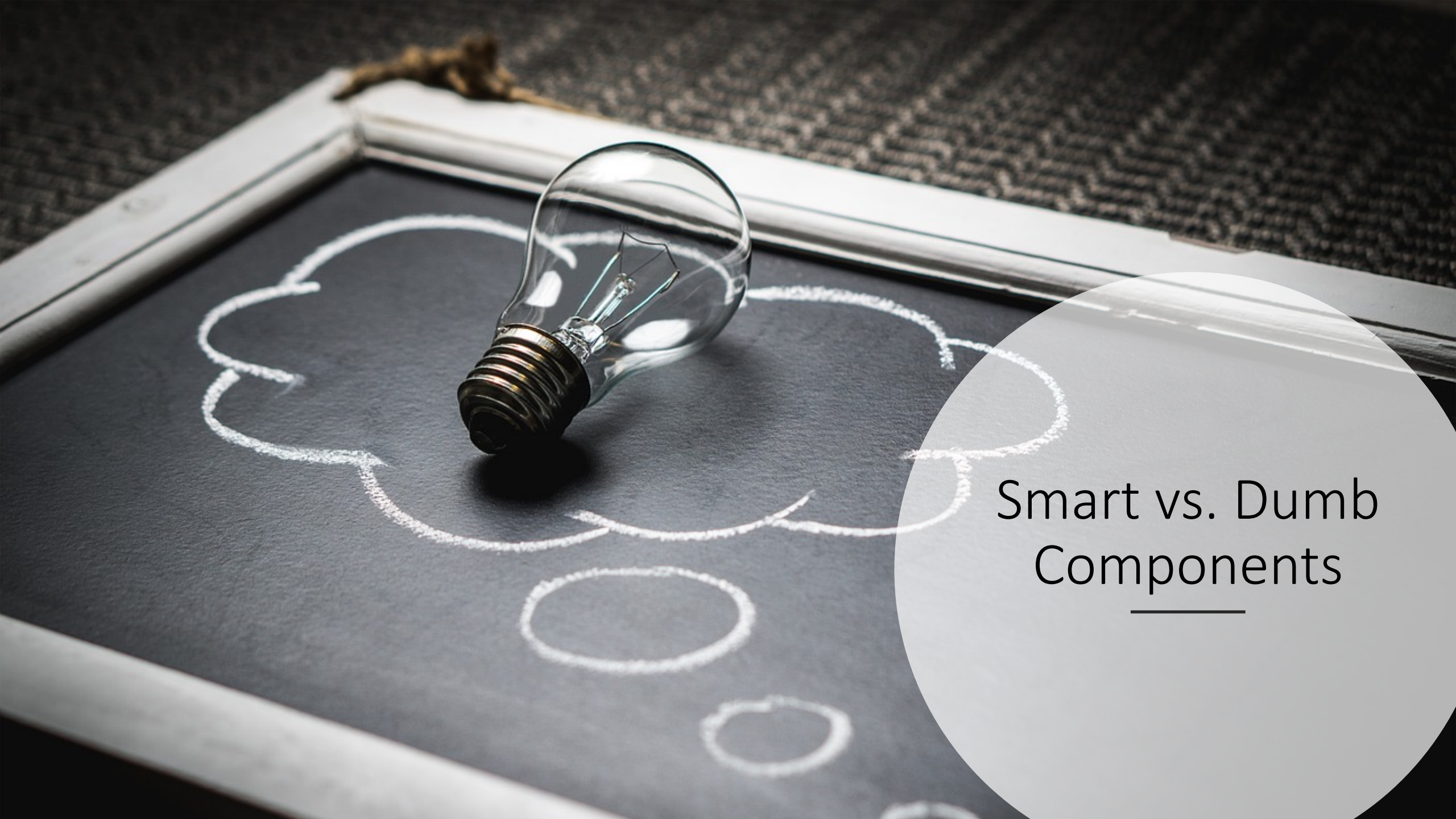
DEMO



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT



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?



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Smart vs. Dumb Components

Smart Component

- Drives the "Use Case"
- Usually a "Container"

Dumb

- Independent of Use Case
- Reusable
- Usually a "Leaf"



Like this topic?

Check out the NgRx Guide

<https://ngrx.io/guide/>