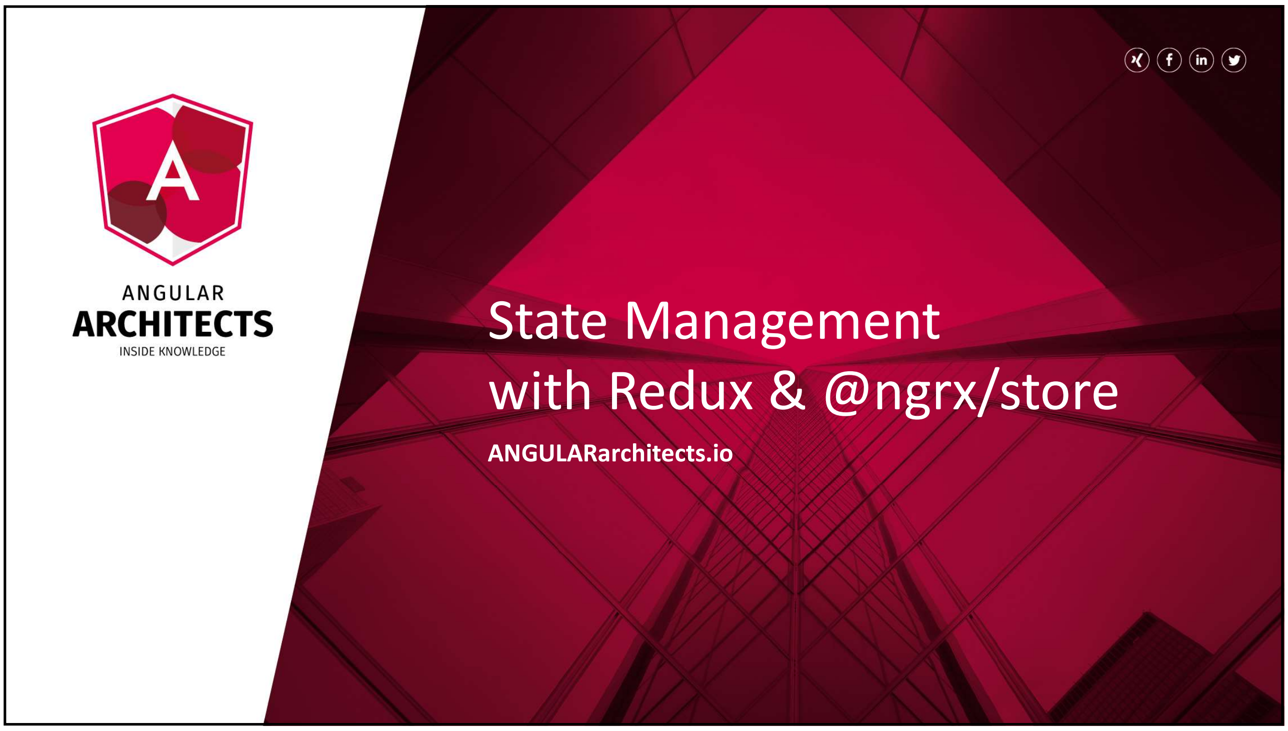


# State Management with Redux & @ngrx/store

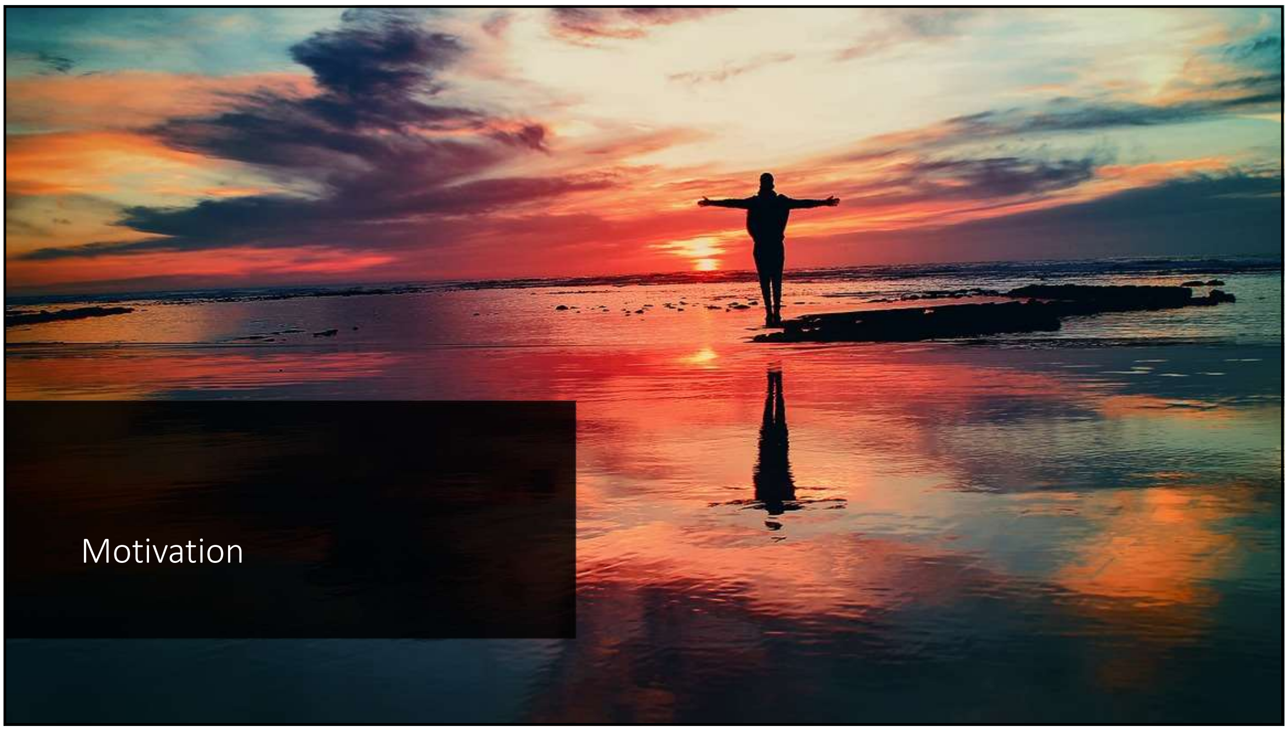
ANGULARarchitects.io

1

## Contents

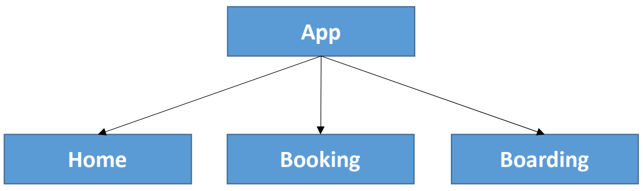
- Motivation
- State
- Actions
- Reducer
- Store
- Immutables
- Effects
- DEMO

2

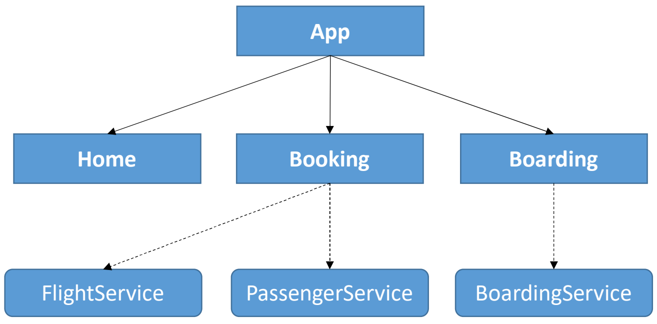


## Motivation

3

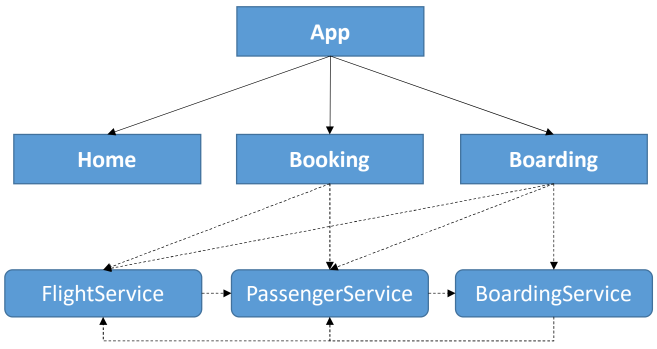


4



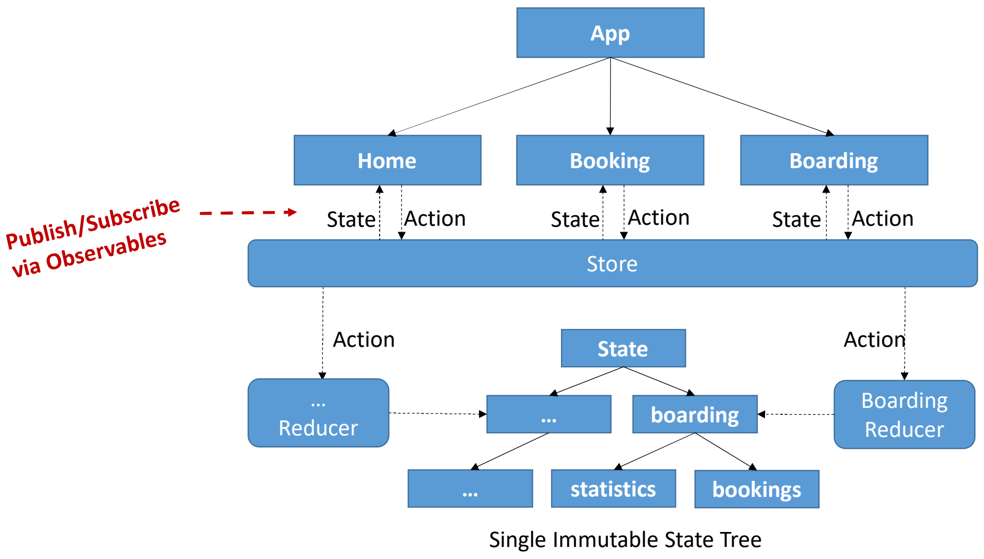
ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

5



ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

6



ANGULAR ARCHITECTS INSIDE KNOWLEDGE

7

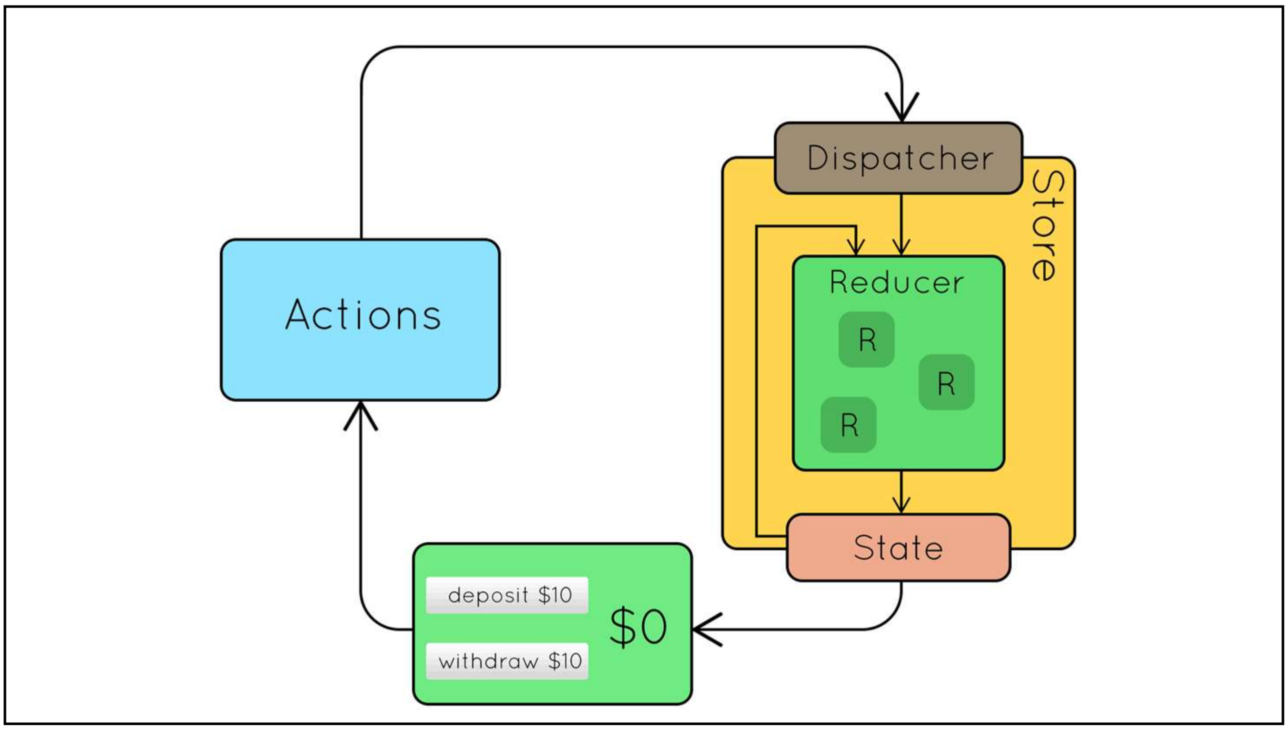
## Redux

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

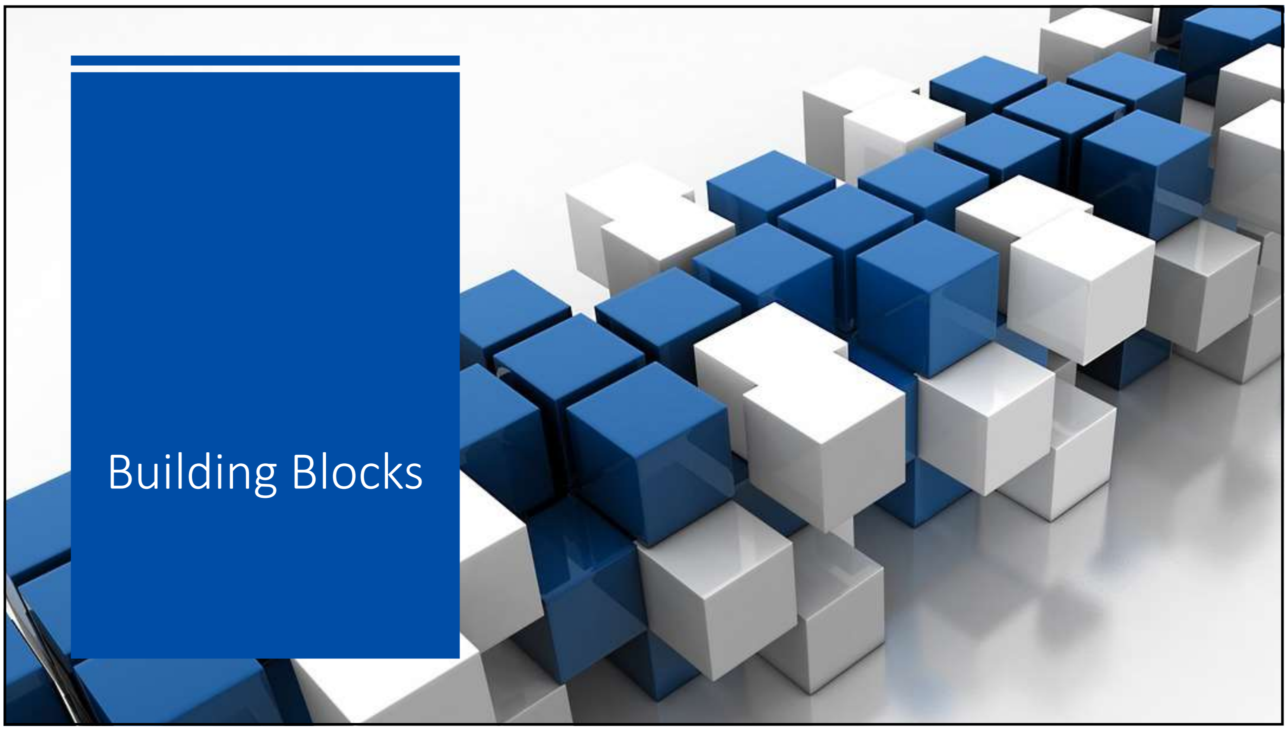
**npm install @ngrx/store --save**

ANGULAR ARCHITECTS INSIDE KNOWLEDGE

8



9



## Building Blocks

10

## State

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

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

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

11

## Action

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

12

## Reducer

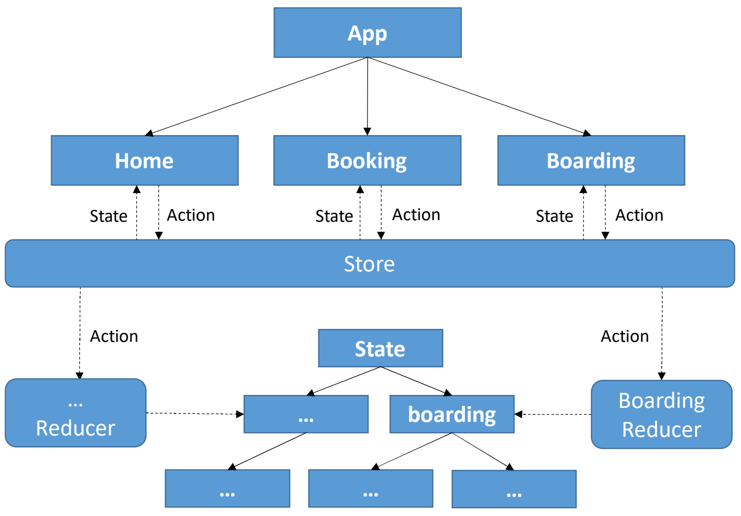
```
export const flightBookingReducer = createReducer(
    initialState,

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

13

## DEMO

14



Single Immutable State Tree

- One "source of truth"
- Prevents Cycles
- Easy to debug
- Structured
- Performance
  - Observables
  - Immutables

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

15

## Actions

20

## Parts of an Action

- Type
- Payload

21

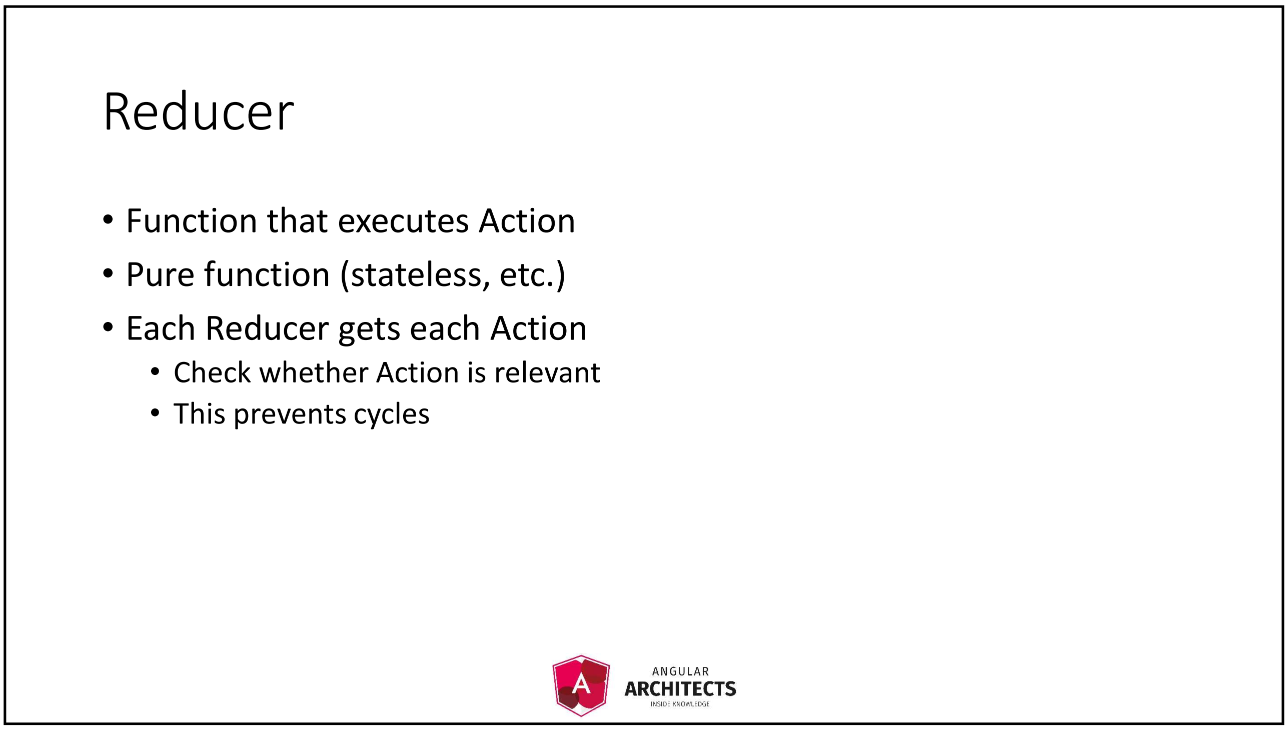
## Defining an Action

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

22

Reducer

23


## 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

24

## Reducer

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

25

## Reducer for FlightBookingState

```
export const flightBookingReducer = createReducer(
    initialState,

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

26

## Map Reducers to State Tree

```
const reducers = {
  "flightBooking": flightBookingReducer,
  "currentUser": authReducer
}
```

27

28

## Store

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

29

## Store

- pipe(
```
    select(tree => tree.flightBooking.flights): Observable<Flight[]>
  )
```
- dispatch(
```
    flightsLoaded({ flights })
  )
```

30

Registering @ngrx/store

31

## Registering @ngrx/Store

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

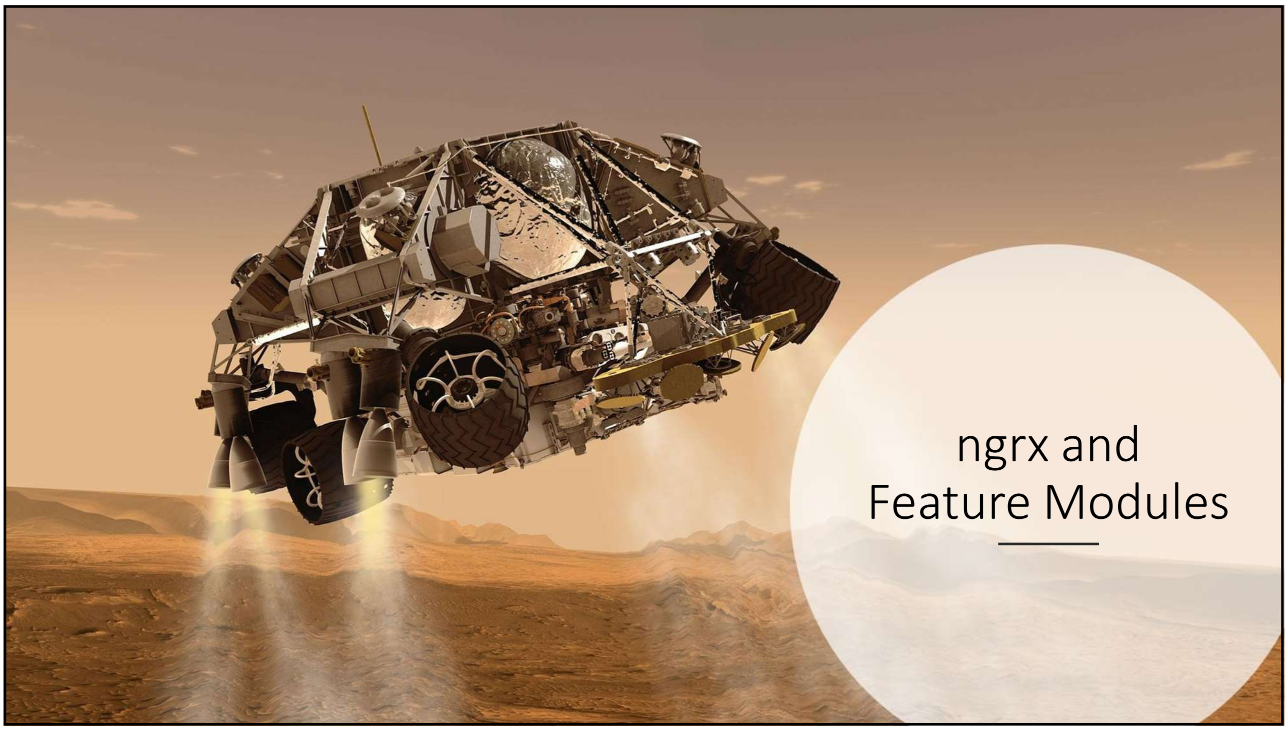
32

## Registering @ngrx/Store

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

**@ngrx/store-devtools**

33



## ngrx and Feature Modules

34

## Reducers for Shared State

```
const reducers = {
  flightBooking: flightBookingReducer,
  currentUser: authReducer
}
```

35

## Reducers for Shared State

```
const reducers = {
  ~~flightBooking: flightBookingReducer,~~
  currentUser: authReducer
}
```

36

## Registering @ngrx/Store

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

*State branch for feature*

ANGULAR ARCHITECTS INSIDE KNOWLEDGE

38

## DEMO

ANGULAR ARCHITECTS INSIDE KNOWLEDGE

39

## LABS

40

## Effects

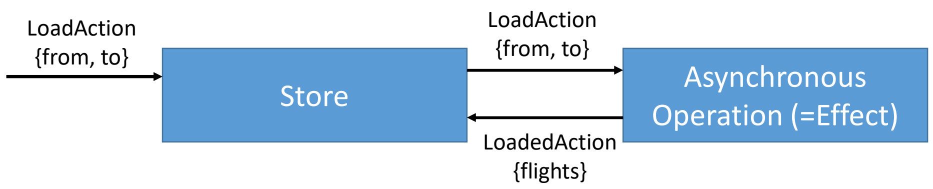
41

## Challenge

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

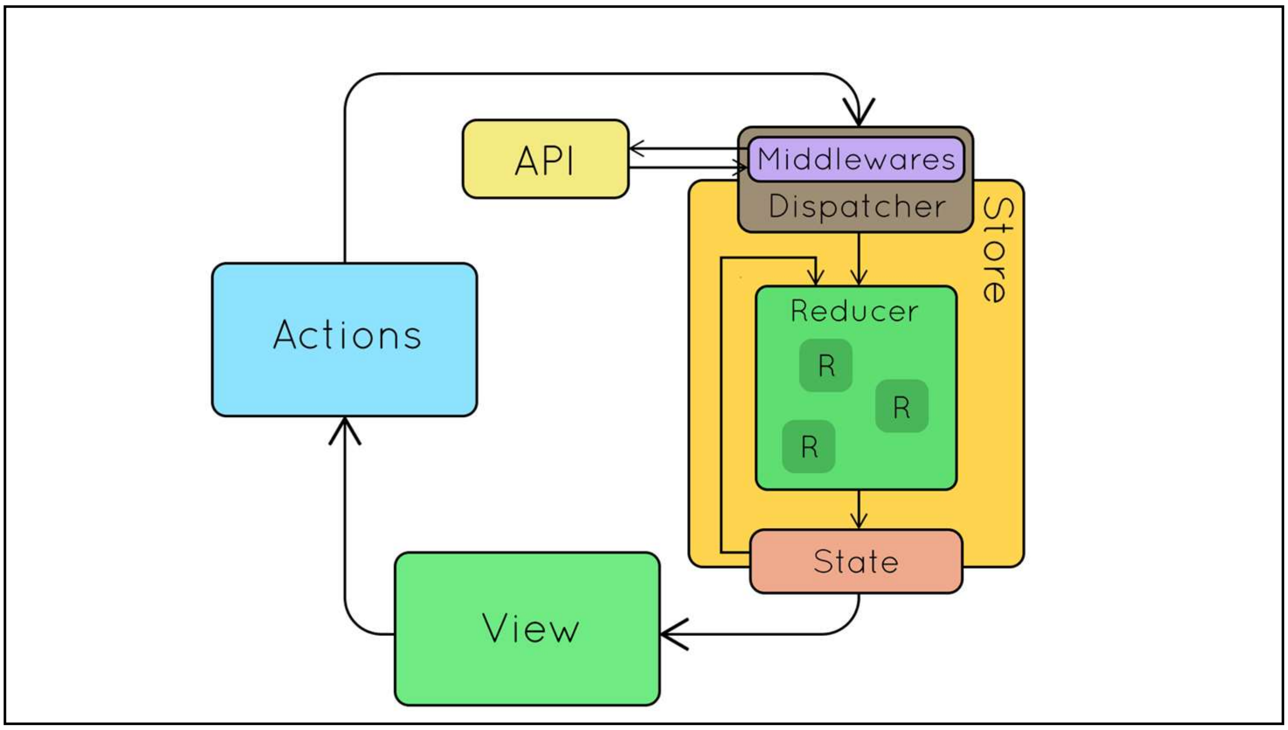
42

## Solution: Effects



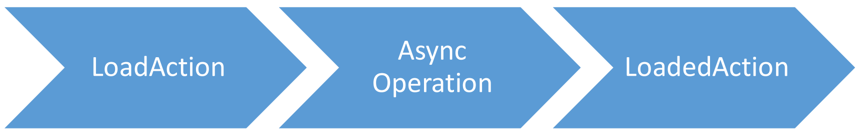
**ng add @ngrx/effects**

43



44

## Effects are Observables



ANGULAR ARCHITECTS INSIDE KNOWLEDGE

45

## Implementing Effects

```
@Injectable()
export class FlightBookingEffects {

    […]

}
```

46

## Implementing Effects

```
@Injectable()
export class FlightBookingEffects {

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

  […]

}
```

47

## Implementing Effects

```
@Injectable()
export class FlightBookingEffects {

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

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

48

## 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)));
}
```

49

## 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

50

## Implementing Effects

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

ANGULAR ARCHITECTS INSIDE KNOWLEDGE

51

## Implementing Effects

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

52

## DEMO

53

## LAB

54

## @ngrx/entity and @ngrx/schematics

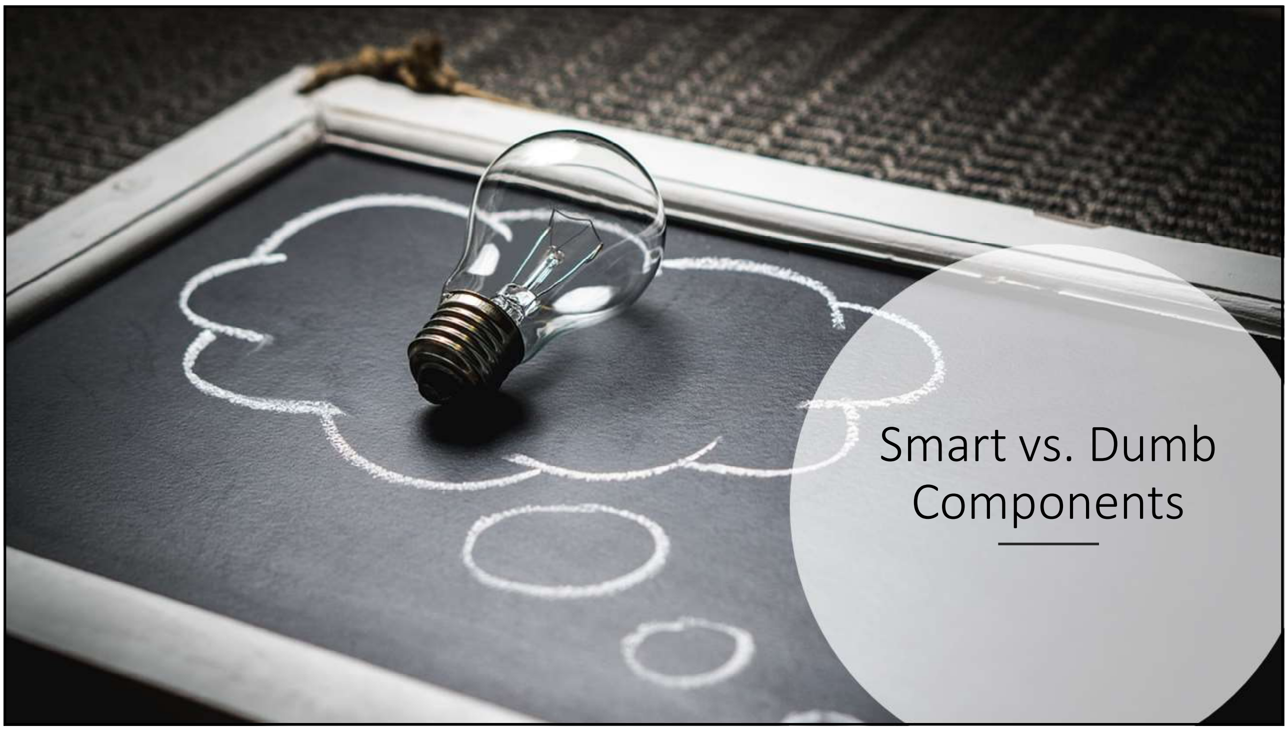
- ng add @ngrx/entity
- ng add @ngrx/schematics
- ng g module passengers
- ng g entity Passenger --module passengers.module.ts

55

## DEMO

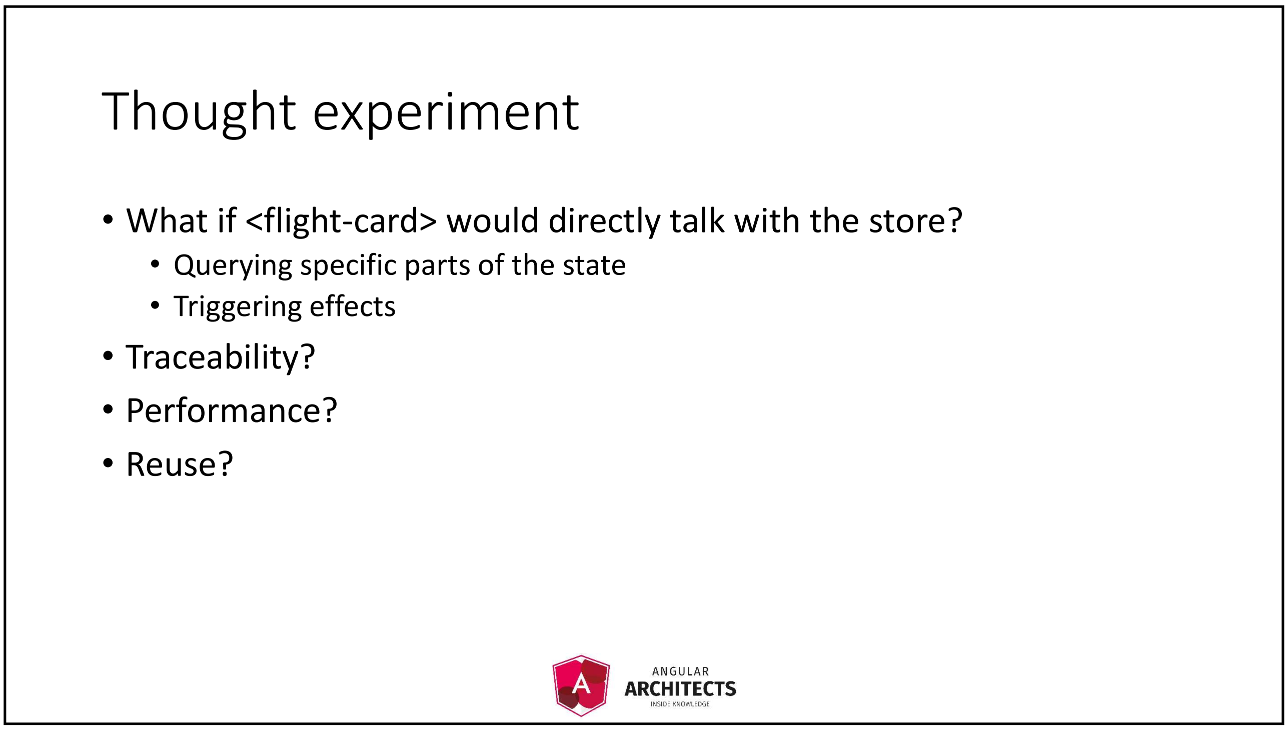
ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

56



## Smart vs. Dumb Components

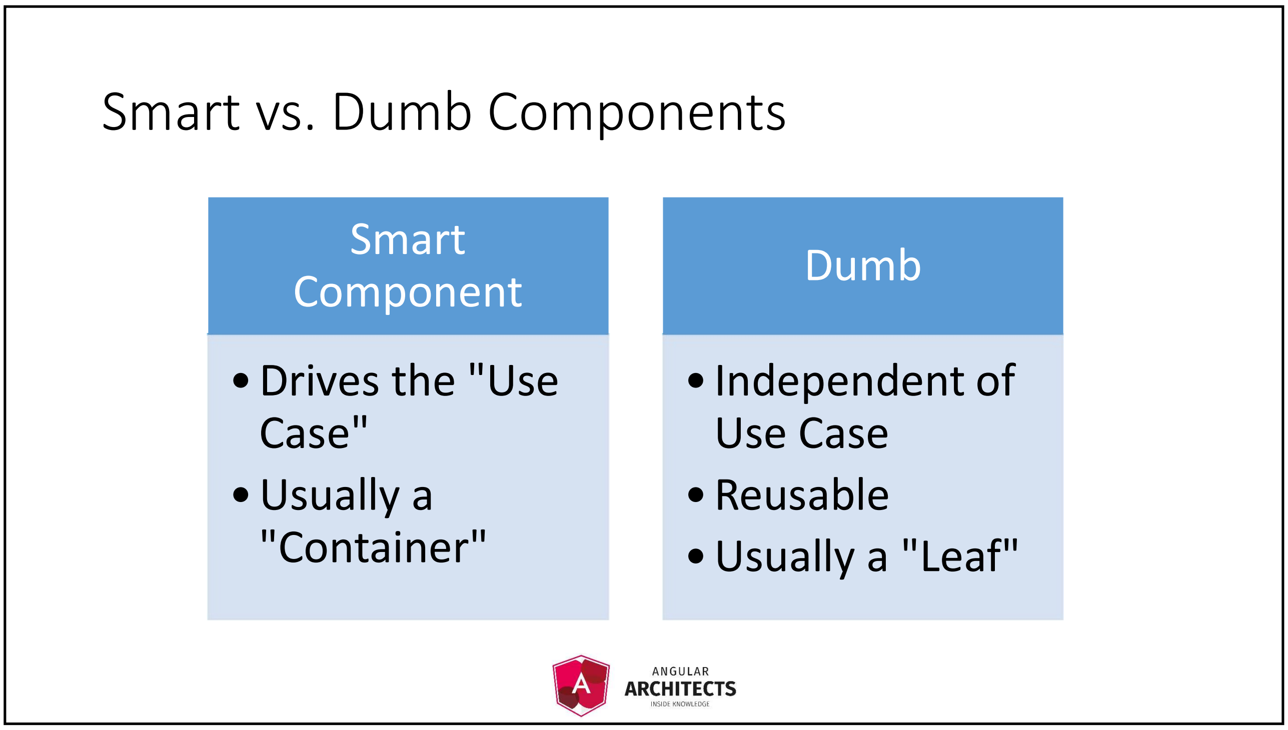
57



## Thought experiment

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

58



## 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" |

59