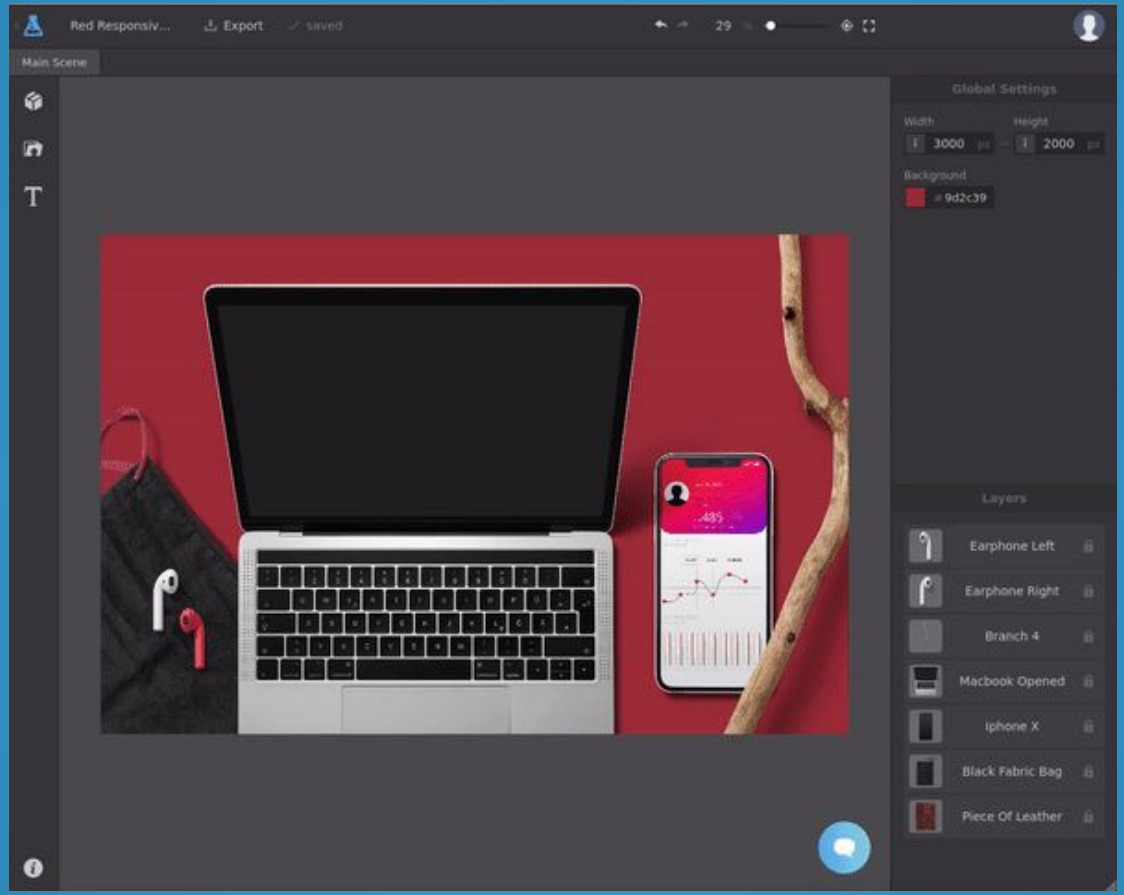
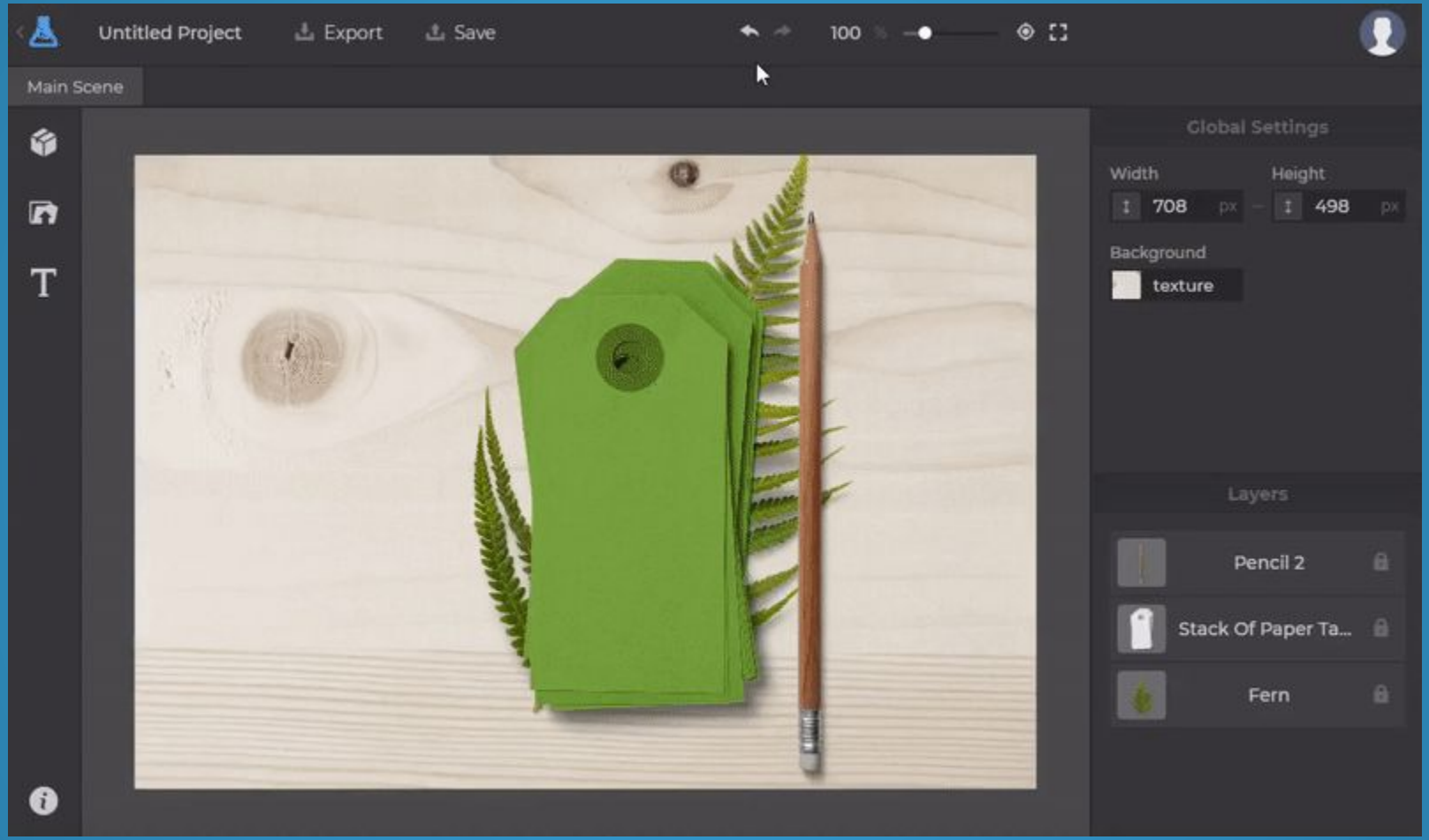
UNDO-REDO WITH ANGULAR & NGRX


Red Responsiv...
Export
saved
29
Main Scene
Global Settings
Width
3000
Height
2000
Background
9d2c39
Layers
Earphone Left
Earphone Right
Branch 4
Macbook Opened
Iphone X
Black Fabric Bag
Piece Of Leather

Minimize opportunity for error, but accept that mistakes will happen


Untitled Project
Export
Save
100
Main Scene
Global Settings
Width
708
px
Height
498
px
Background
texture
Layers
Pencil 2
Stack Of Paper Ta...
Fern

HEY

NILS MEHLHORN

freelance software engineer

founder of scenelab.io

www nils-mehlhorn.de

@n_mehlhorn

NGRX BOOK

Pay what you want for the complete learning resource

gum.co/angular-ngrx-book

ERROR TOLERANCE = USER-FRIENDLY DESIGN

- users have different backgrounds
- ease onboarding

→ confidence & creativity

⚠ browser supports only some interactions

KEYBOARD SHORTCUTS: CONSIDERATIONS

- common combinations: Ctrl + Z / Ctrl + Shift + Z
- provide legend and/or tooltips
- consider existing browser shortcuts
- consider internationalization

KEYBOARD SHORTCUTS: IMPLEMENTATION

- Key Event Bindings
- EventManager
- NgRx Effect

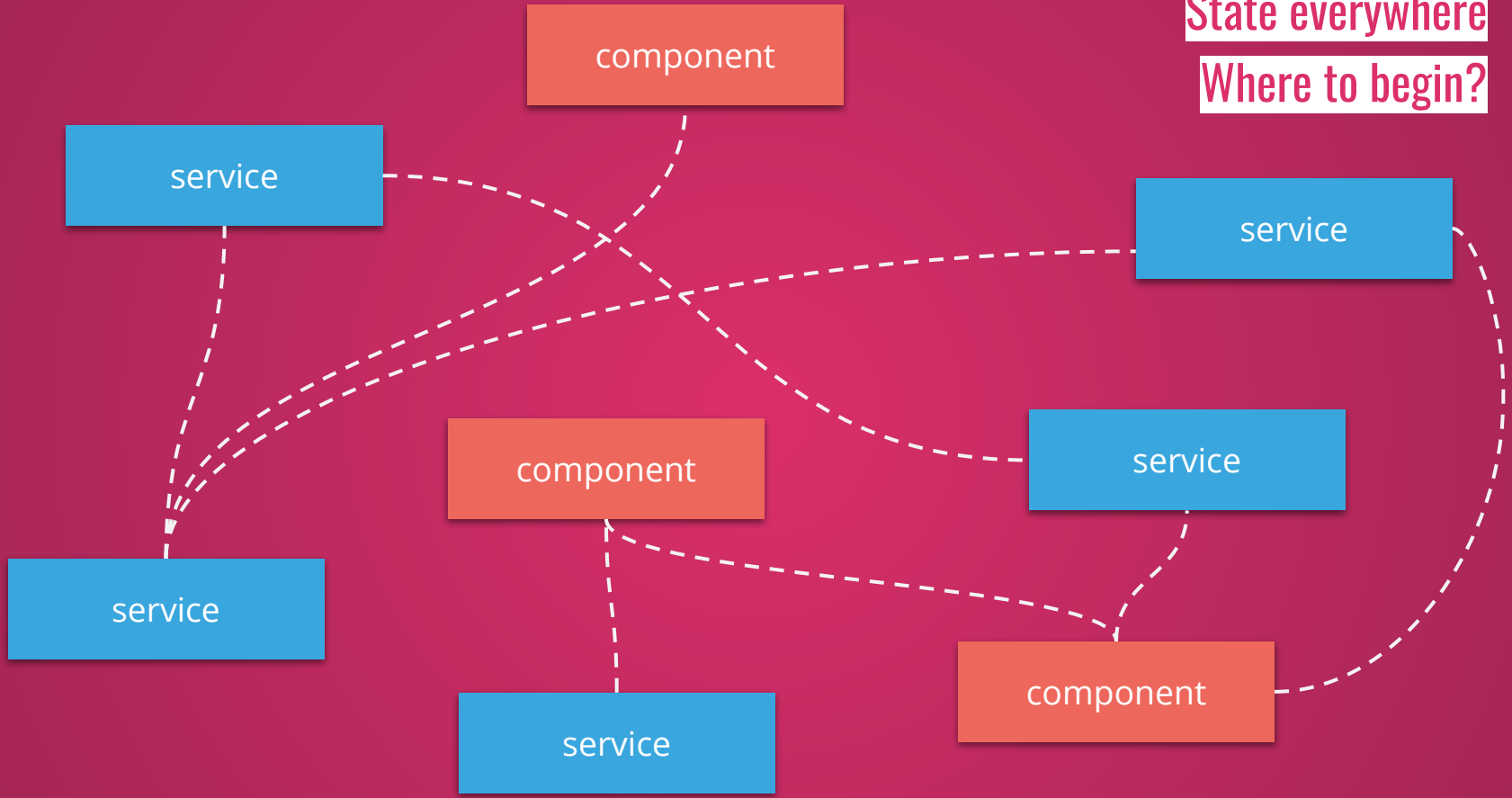
```
<div (keydown.control.z)="undo()"
     (keydown.control.shift.z)="redo()”>
</div>
```

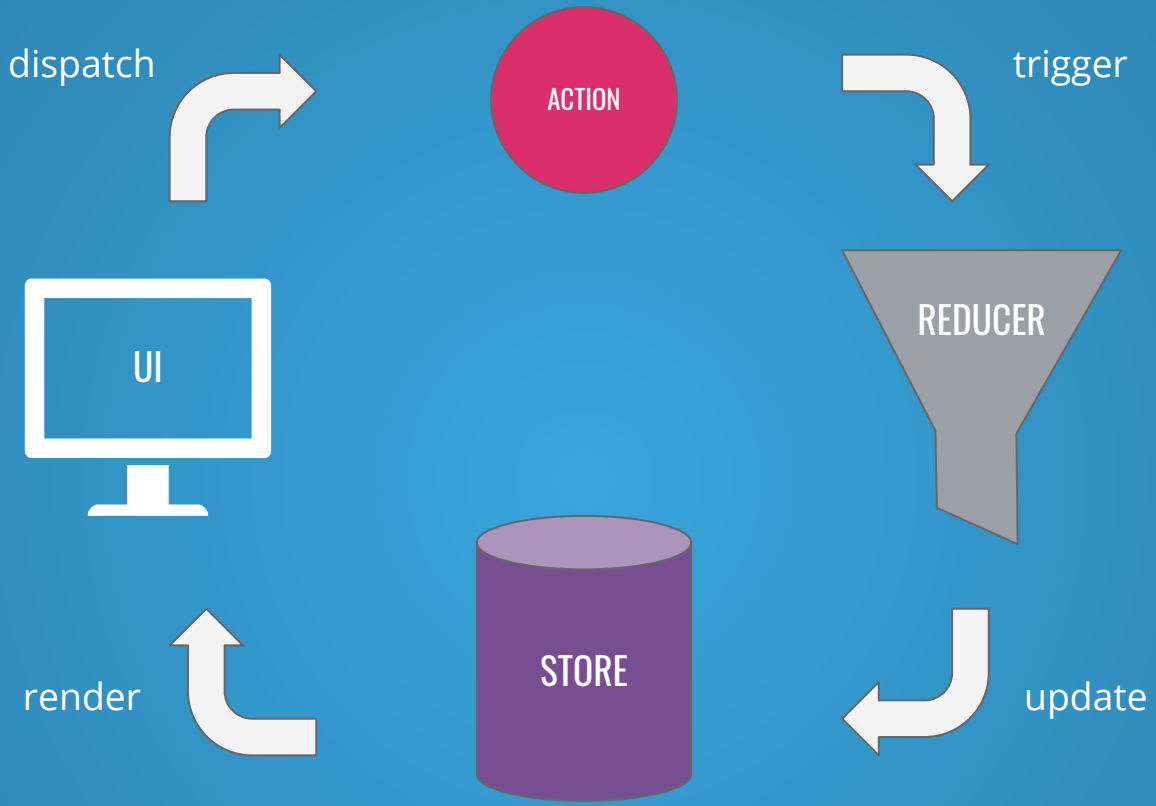
Recommended Read
Keyboard Shortcuts in Angular -- Netanel Basal

```
shortcut$ = createEffect(() =>
    fromEvent(document, ‘keydown’)
        .pipe(...)
)
```

State everywhere

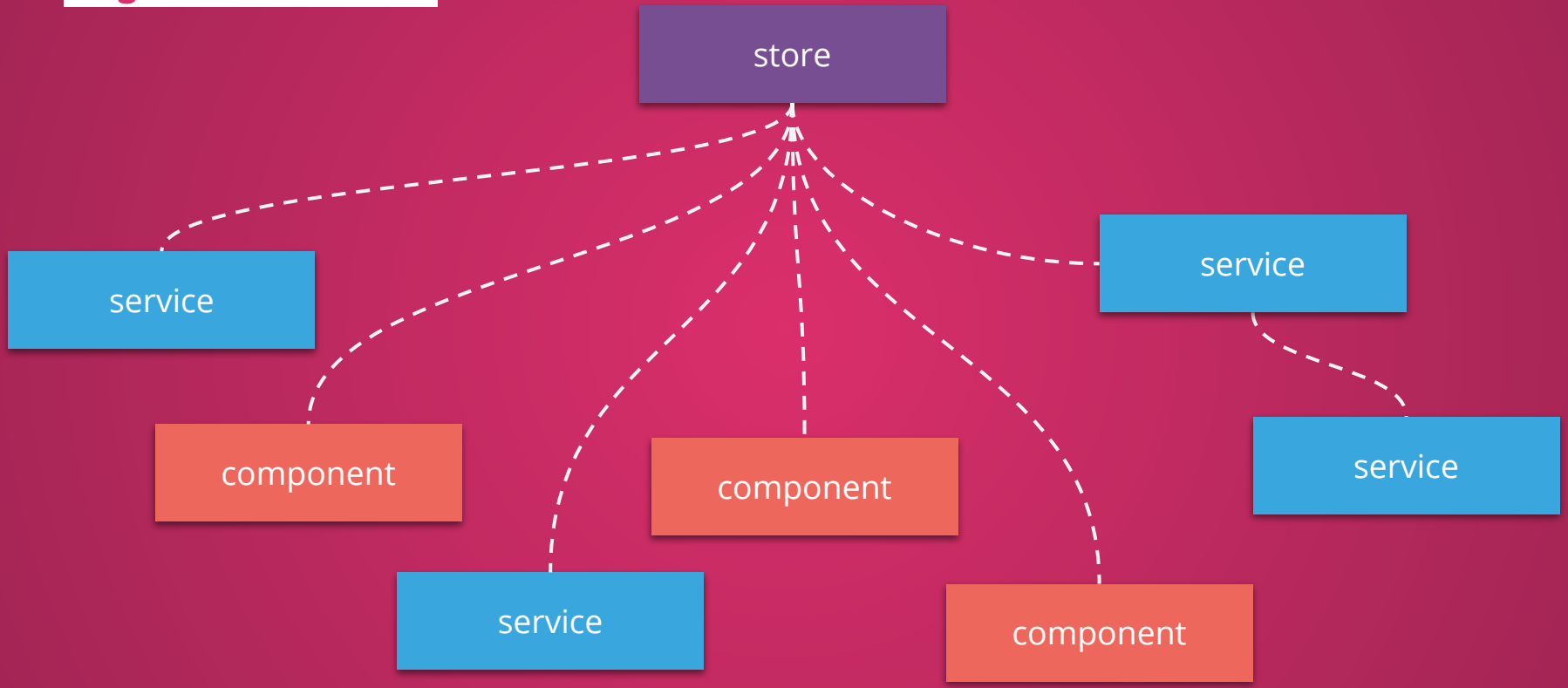
Where to begin?



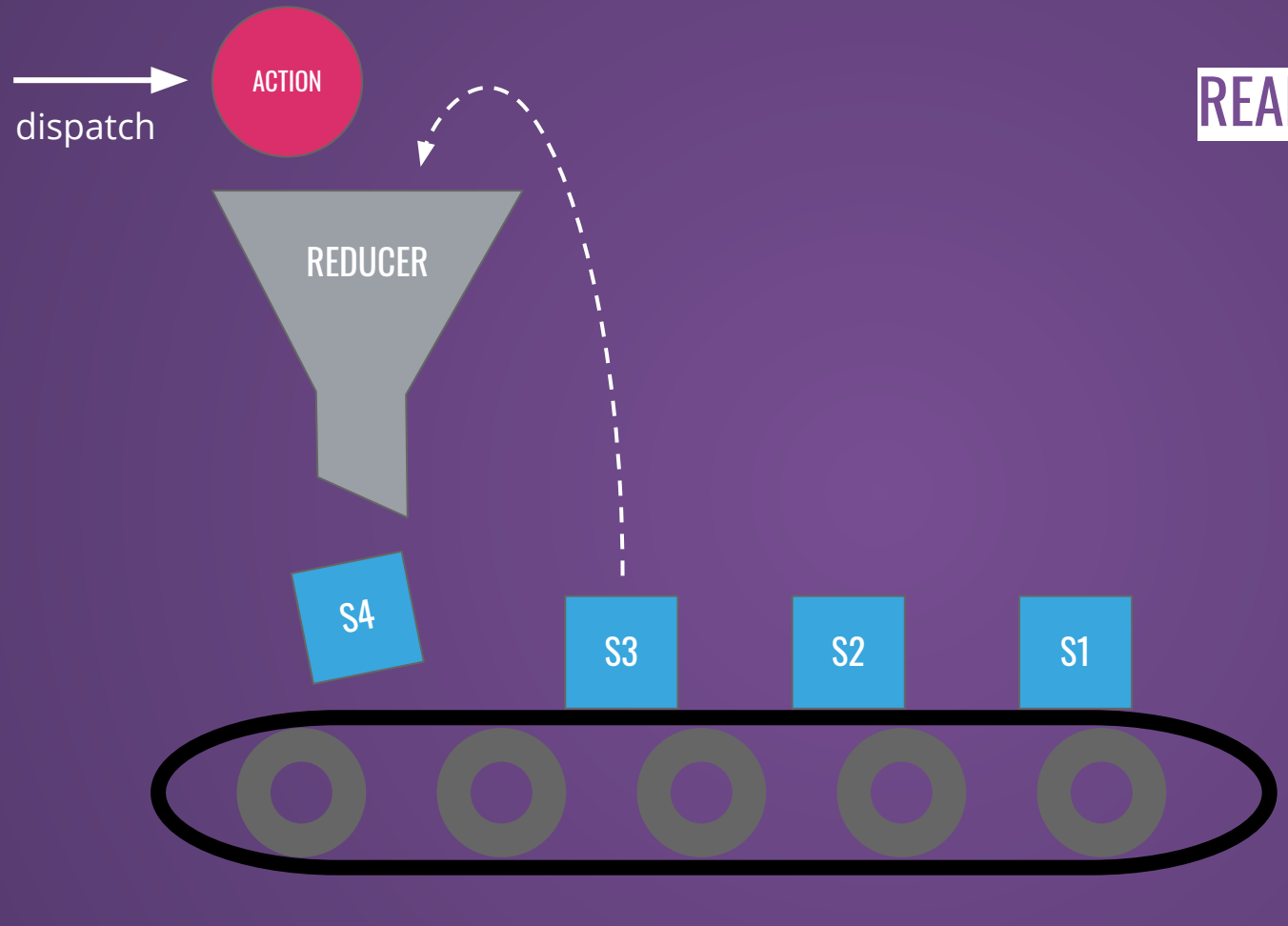


REDUX / NGRX ARCHITECTURE

Single Source of Truth



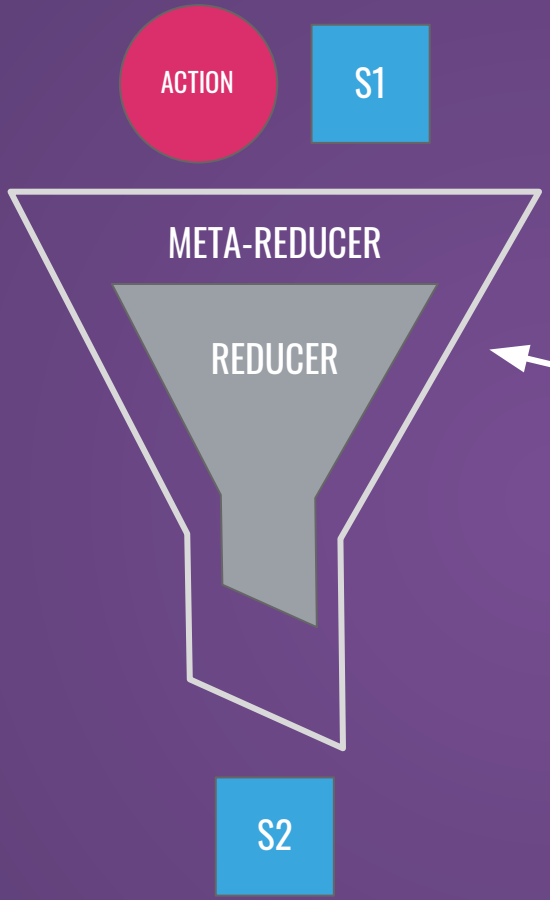
READ-ONLY STATES



Services
Components

Undo-Redo
History?

PURE FUNCTIONS



Put Undo-Redo
Logic here

```
interface History {
  past: Array<State>
  present: State
  future: Array<State>
}
```

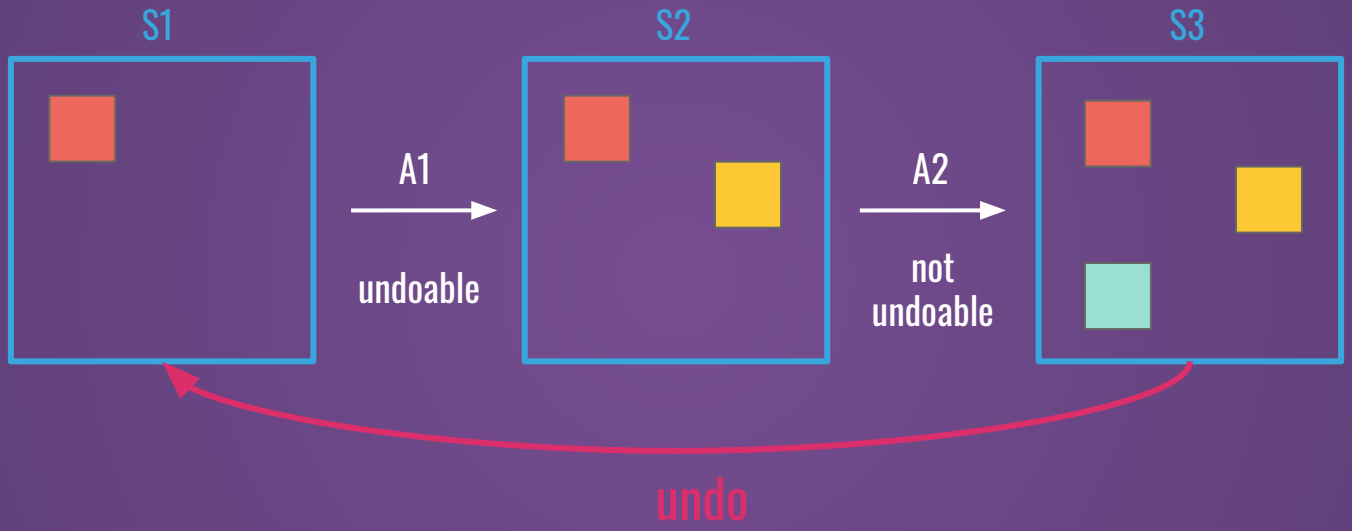
```
default:
  const newPresent = reducer(state, action)
  history = {
    past: [history.present, ...history.past],
    present: newPresent,
    future: [] // clear future
  }
  return newPresent
```

```
case 'UNDO':
  const previous = history.past[0]
  const newPast = history.past.slice(1)
  history = {
    past: newPast,
    present: previous,
    future: [history.present, ...history.future]
  }
  return previous
```

HISTORY OF STATES

HISTORY OF STATES

- 👍 intuitive implementation
- 👍 most libraries do this
- 👎 it can get big
- 👎 it’s all or nothing



ALL OR NOTHING: GOING BACK MEANS LOSING THE GREEN SQUARE

HISTORY OF STATES

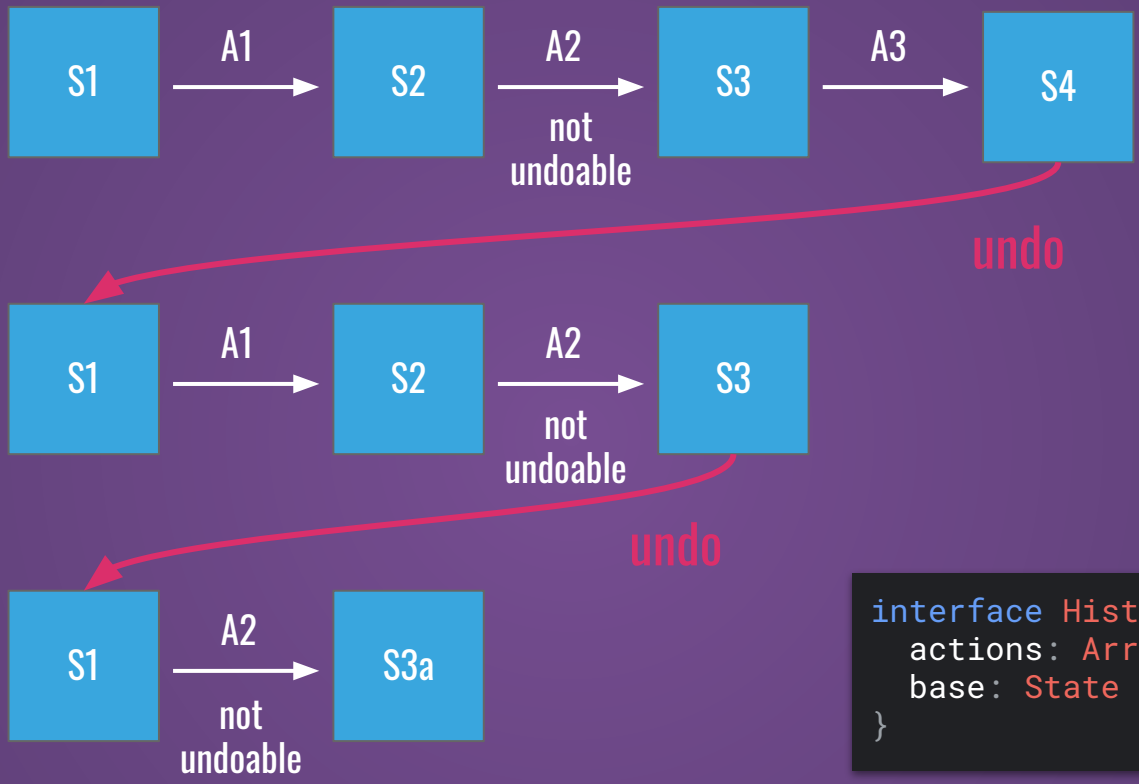
👍 intuitive implementation

👍 most libraries do this

👎 it can get big

👎 it’s all or nothing

⚠️ careful reducer composition required




```
interface History {
  actions: Array<Action>
  base: State
}
```

HISTORY OF ACTIONS

```
interface History {
  actions: Array<Action>
  base: State
}
```

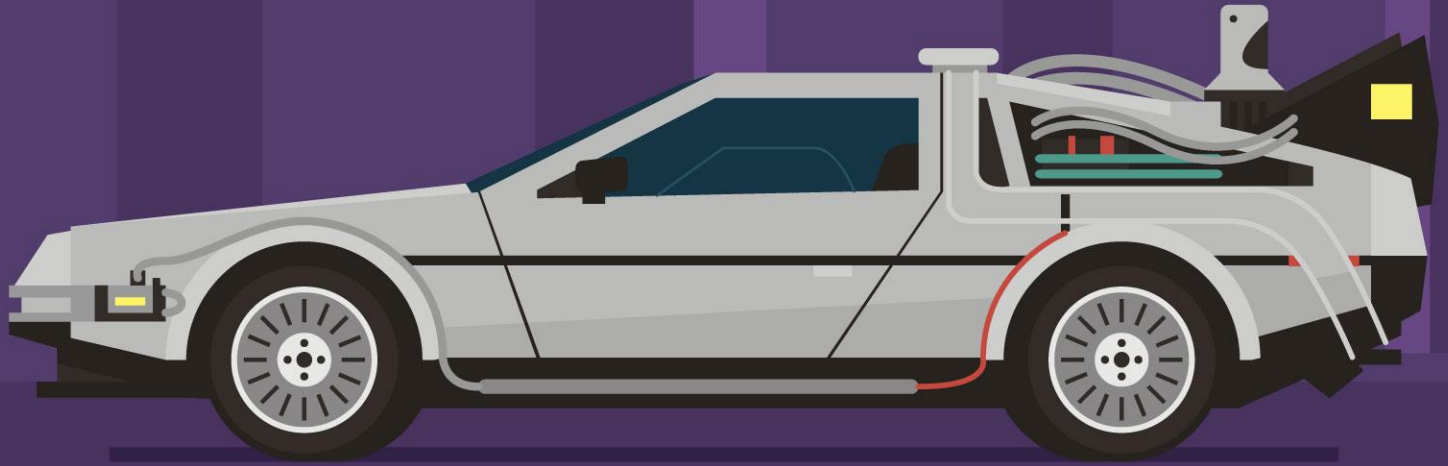
```
const lastState = history.actions
    .slice(0, -1) // every action except the last one
    .reduce(
        (state, action) => reducer(state, action),
        history.base
    )
```

HISTORY OF ACTIONS

HISTORY OF ACTIONS

- 👍 actions < states
- 👍 ignore some actions
- 👎 tricky implementation
- 👎 expensive recalculation

There’s another way back to the future



Delorean Vectors by Vecteezy

```
// initial state
const state = { "firstname": "John" }

// JSON Patch representing what reducer did to change the state
const patch = [
  { "op": "add", "path": "/lastname", "value": "Doe" }
]

// result state when applying patch to S1
const next = { "firstname": "John", "lastname": "Doe" }
```

JSON Patch

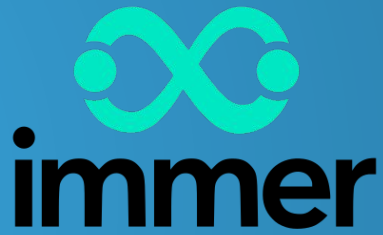
```
// result state from before
const next = { "firstname": "John", "lastname": "Doe" }

// JSON Patch representing the reverse
// of what reducer did to change the state
const inversePatch = [
  { "op": "remove", "path": "/lastname" }
]

// resulting initial state when applying inversePatch to S2
const state = { "firstname": "John" }
```

Inverse JSON Patch

immer

COPY-ON-WRITE

```
import produce, {applyPatches} from "immer"

const state = { "firstname": "John" }

let undoPatches

const next = produce(
        state,
        draft => {
          draft.lastname = "Doe"
        },
        (patches, inversePatches) => {
          undoPatches = inversePatches
        }
)

const patched = applyPatches(next, undoPatches)

expect(patched).toEqual(state)
```



PATCHES WITH IMMER

HISTORY OF PATCHES

- 👍 lightweight
- 👍 ignore some actions
- 👍 feasible implementation
- 👍 no recalculation

- 👉 requires Immer

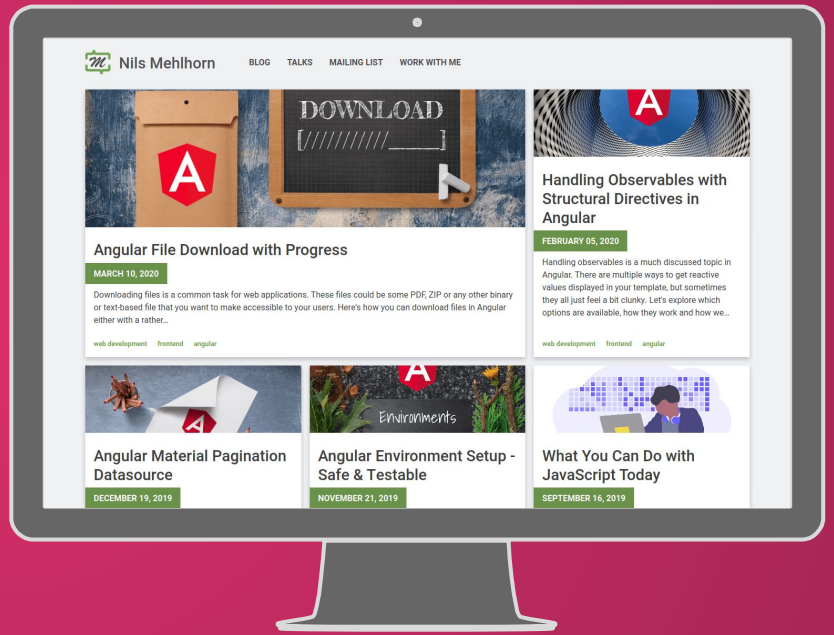
NGRX-WIEDER

- patch-based undo-redo
- ignore actions
- merge actions
- segmentation

DEMO

THANKS

1. Visit Blog
2. Join Mailing List
3. Follow On Twitter
4. Work With Me



www nils-mehlhorn.de

@n_mehlhorn