

**YOU DON'T KNOW
MOBX STATE TREE**

HI


I'M MAX GALLO

PRINCIPAL ENGINEER AT DAZN



twitter: @_maxgallo (yes with an underscore)
!twitter: maxgallo.io

THE PLAN

- » MobX intro
- » MobX State Tree overview
- » Design a Reactive Project
- » Best Practises

**IF MOBX IS THE ENGINE
MOBX STATE TREE IS THE CAR**

MOUPDBY THE ENGINE OF THE CAR

- » Uses Reactive paradigms
- » Decouples view from business logic

MOBX

```
● ● ●  
1 class CarPark {  
2     @observable cars = []  
3     @computed get howManyFerrari() {  
4         return this.cars.filter(this.isFerrari).length;  
5     }  
6     isFerrari = carName => carName.includes('Ferrari')  
7 }  
8  
9 @observer  
10 class CarParkView extends Component {  
11     renderCar(car){ return <li key={car}>{car}</li> }  
12     render() {  
13         const { carPark } = this.props  
14         return (<div>  
15             <ul>{ carPark.cars.map(this.renderCar) } </ul>  
16             How many Ferrari? { carPark.howManyFerrari }  
17         </div>);  
18     }  
19 }  
20  
21 const carPark = new CarPark();  
22  
23 render(<CarParkView carPark={carPark} />, element('root'));  
24  
25 setTimeout( () => carPark.cars.push('Ferrari 458') , 1000);  
26 setTimeout( () => carPark.cars.push('Ferrari Enzo') , 2000);
```

Observable state

Mutable Application State

Computed Values

Automatically derived values

@Observer

Subscribe to every observable
in the render function

Reactions

Side effects like updating a
React component

MOBX STATE TREE

- » Powered by MobX
- » Opinionated
- » Relies on the concept of Trees

WHAT'S A TREE ? ALSO KNOWN AS STORE

```
1 const CarStore = types
2   .model('Car', {
3     name: types.string,
4   })
5   .views(self => {
6     get isFerrari() {
7       return self.name.includes('Ferrari')
8     }
9   })
10
11 const CarParkStore = types
12   .model('CarPark', {
13     cars: types.optional(types.array(CarStore), []),
14   })
15   .views(self => {
16     get howManyFerrari(){
17       return self.cars.filter(
18         car => car.isFerrari
19       ).length;
20     }
21   })
22   .actions(self => {
23     addCar(car) { self.cars.push(car) }
24   });
25
26 const carParkStore = CarParkStore.create();
27
28 carParkStore.addCar({ name: 'Ferrari 458 Italia' });
29 carParkStore.addCar({ name: 'Ferrari Enzo' });
```

Model

Strongly typed mutable observable state that could contain other trees.

Views

MobX computed values

Actions

The only way to update the model

MOBX STATE TREE HOW TO CONNECT STORES WITH REACT COMPONENTS?



```
1 /** ----- App.js ----- */
2
3 import { Provider }      from "mobx-react"
4 import App               from "./App.js"
5 import ShopStore         from './Shop.store.js'
6 import NavigationStore   from './Navigation.store.js'
7
8 const shopStore = ShopStore.create()
9
10 ReactDOM.render(
11     <Provider
12         shop={shopStore}
13         navigation={navigationStore}
14     >
15         <App />
16     </Provider>,
17     document.getElementById("root")
18 )
```



```
1 /** ----- CheckoutView.js ----- */
2
3 import React             from 'react'
4 import { inject, observer } from 'mobx-react'
5
6 @inject('shop') @observer
7 class CheckoutView extends React.Component {
8     render() {
9         const { shop } = this.props;
10        return (
11            { shop.checkoutAmount }
12        );
13    }
14 }
```

MOBX STATE TREE DEEP DIVE



- » Mutable and Immutable (Snapshots, Time Travelling)
- » Store Composition
- » Dependency Injection
- » Store Lifecycle Methods

DESIGNING STORES

You need to think about:

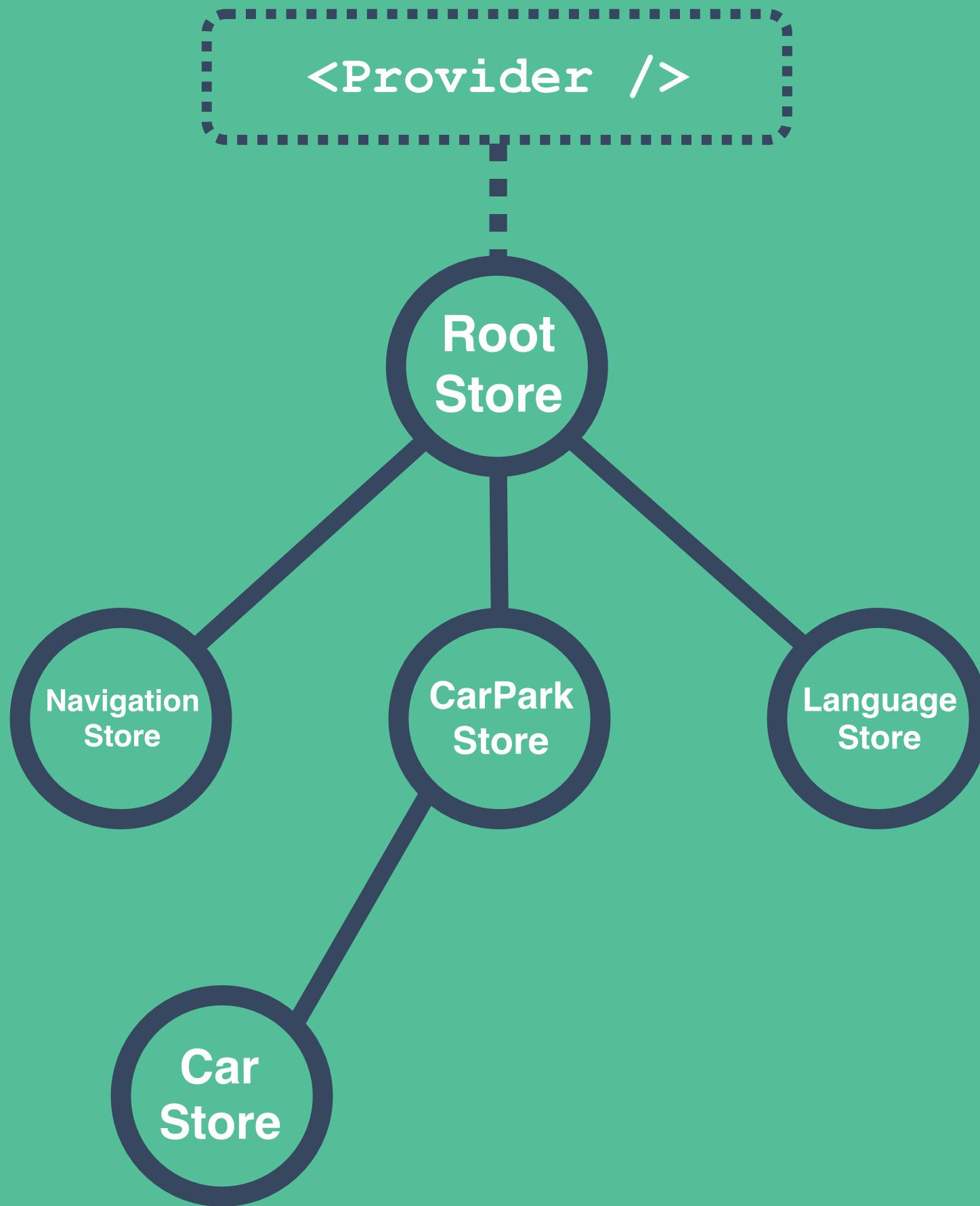
1. SHAPE YOUR TREES

One Root Store vs Multiple Root Stores

2. STORES COMMUNICATION

How Stores communicate between each other

MOBX STATE TREE ONE ROOT STORE



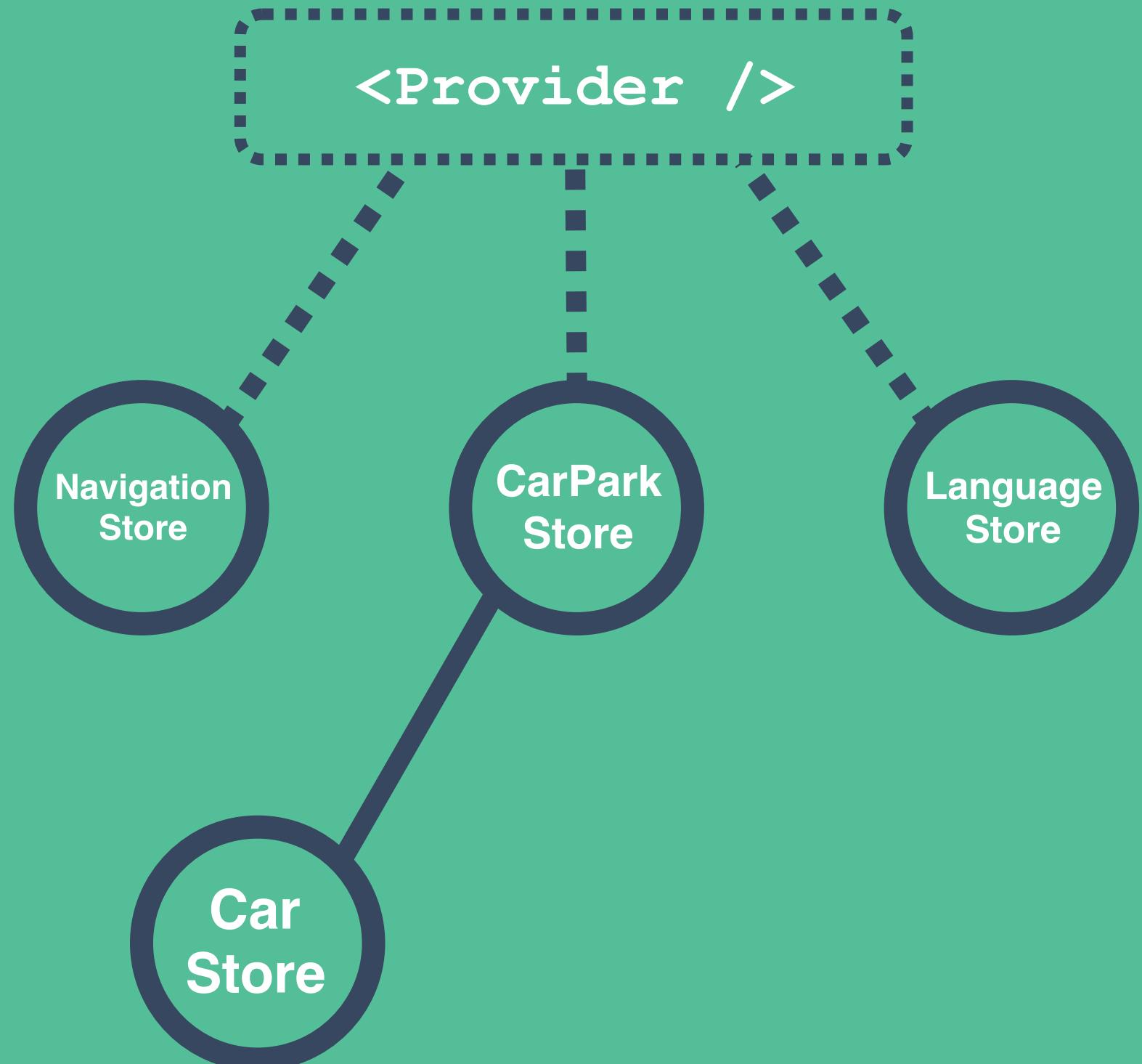
Pros

Easier to perform actions on everything at once (snapshot, creation, destroy)

Cons

Very easy to create tightly coupled stores

MOBX STATE TREE MULTIPLE ROOT STORES



Pros

Easier to reason by Domain

Cons

Less immediate to perform actions on everything

REAL WORLD STORES COMMUNICATION

1. Default Approach
2. Actions Wrapper
3. Dependency Injection



STORES COMMUNICATION 1

DEFAULT APPROACH

```
1 /** ----- Root.store.js ----- */
2
3 import { types } from 'mobx-state-tree';
4
5 const RootStore = types
6   .model('RootStore', {
7     navStore : types.maybe(NavStore),
8     pageStore : types.maybe(PageStore)
9   })
10
11 /** ----- Page.store.js ----- */
12
13 import { types, getParent } from 'mobx-state-tree';
14
15 const PageStore = types
16   .model('PageStore', {
17     currentView : types.option(types.string, '')
18   })
19   .actions(self => {
20     showLoginForm() {
21       self.currentView = 'login';
22       getParent(self).navStore.setPath('/login')
23     },
24   });

```

Each Store access directly other Stores.

- » Easier when using a Single Root Store

- » Each Store could end up knowing the whole structure



STORES COMMUNICATION 2

ACTIONS WRAPPER



```
1 import { types, getParent } from 'mobx-state-tree'
2
3 const ActionsWrapperStore = types
4   .model('ActionsWrapperStore', {})
5   .actions(self => ({
6     login() {
7       authStore.login()
8       pageStore.login()
9       navigationStore.login()
10    },
11    goHome() {
12      pageStore.showDefault();
13      navigationStore.login()
14    }
15  }));

```

One Store,
to rule them all 

- » Calls directly other Stores
- » Knows a lot about your App



STORES COMMUNICATION 3

DEPENDENCY INJECTION

```
1 /** ----- Region.store.js ----- */
2 const RegionStore = types
3   .model('RegionStore', {
4     region: types.optional(types.string, 'UK')
5   })
6
7 /** ----- Navigation.store.js ----- */
8 import { types, getEnv } from 'mobx-state-tree';
9
10 const NavigationStore = types
11   .model('NavigationStore', {
12     path: types.string,
13   })
14   .view(self => {
15     get region() {
16       getEnv(self).navigationStore.region;
17     },
18     get urlPath() {
19       return `${self.region}/${self.path}`;
20     }
21   });
22
23 /** ----- index.js ----- */
24 const regionStore = RegionStore.create({});
25 const navigationStore = NavigationStore.create(
26   { path: 'login' },
27   { regionStore }
28 );
29
30 console.log(navigationStore.urlPath); // 'UK/login'
```

Injecting one or multiple stores into another one.

- » You could use it for both Actions and Views
- » You need to be carefull about circular dependencies

ONE MORE THING . . .





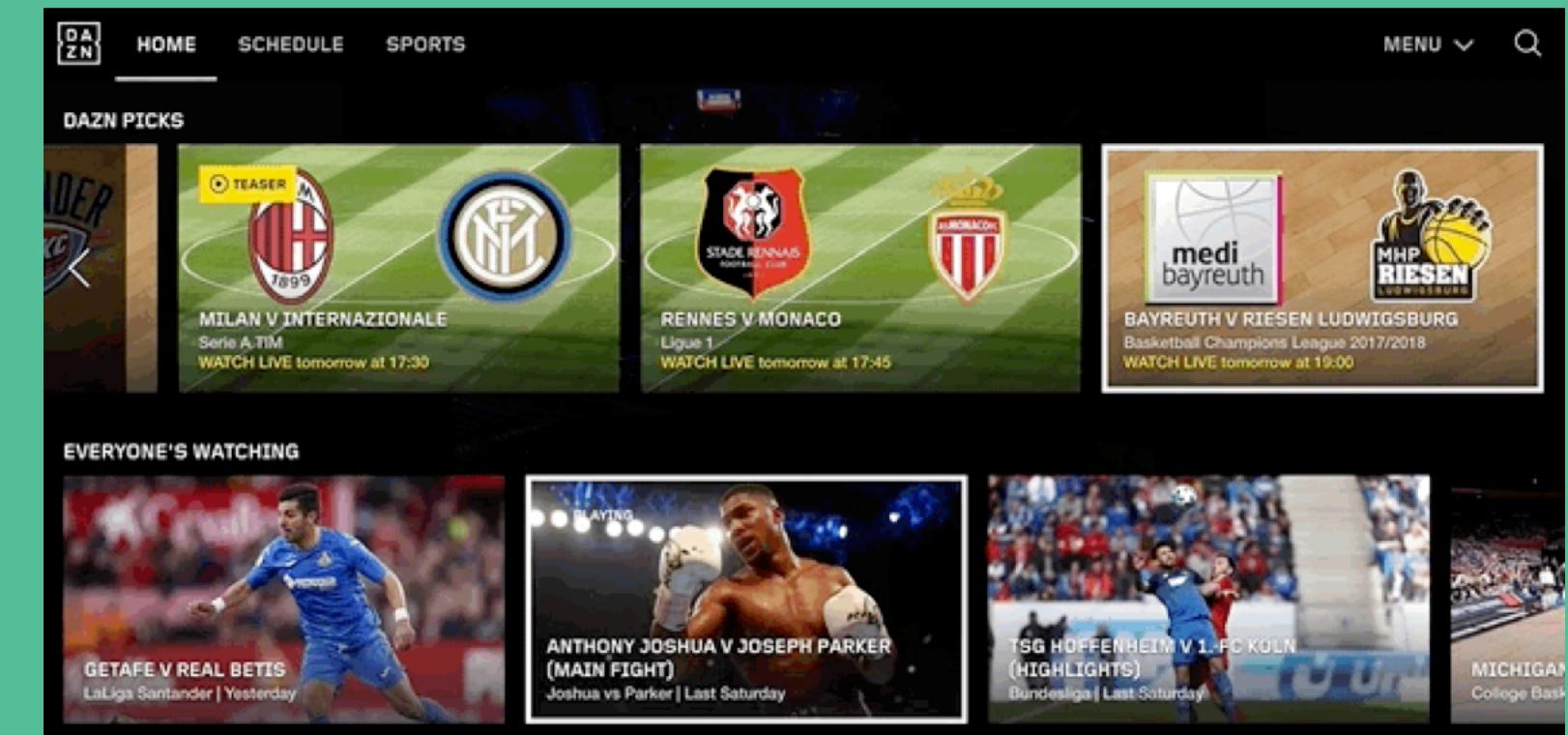
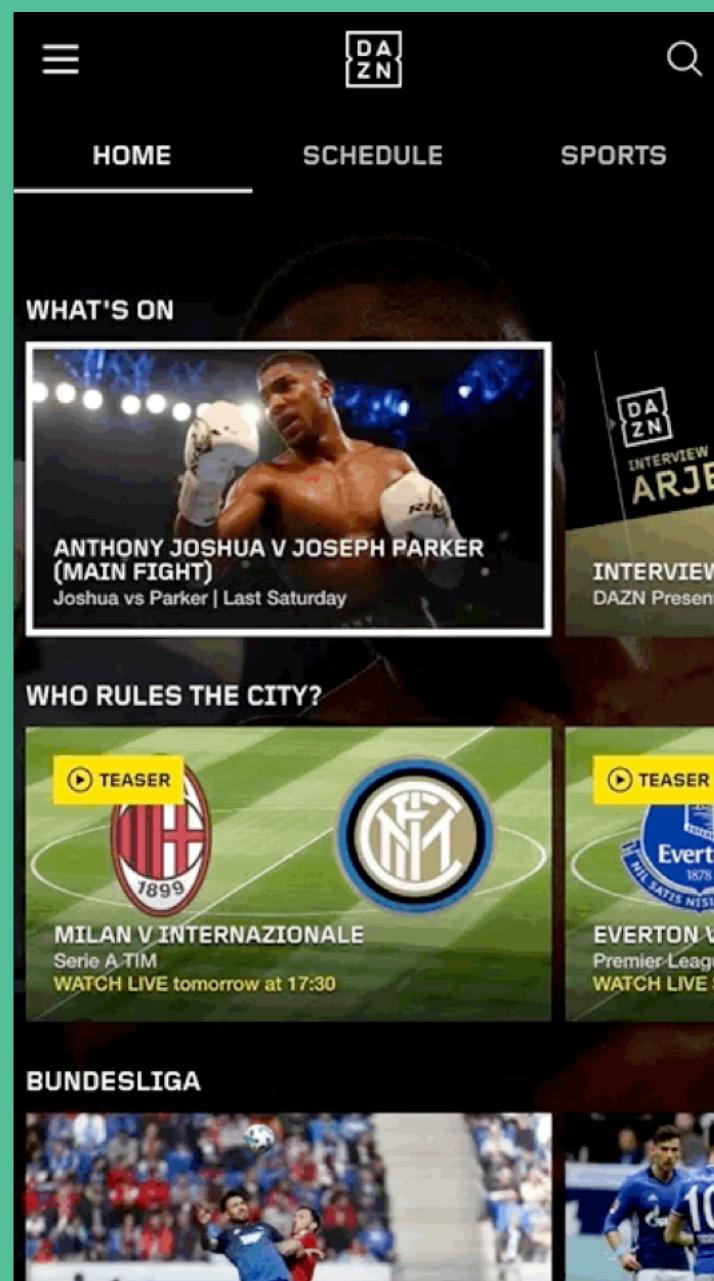
STORE COMPOSITION

Two or more stores can be composed

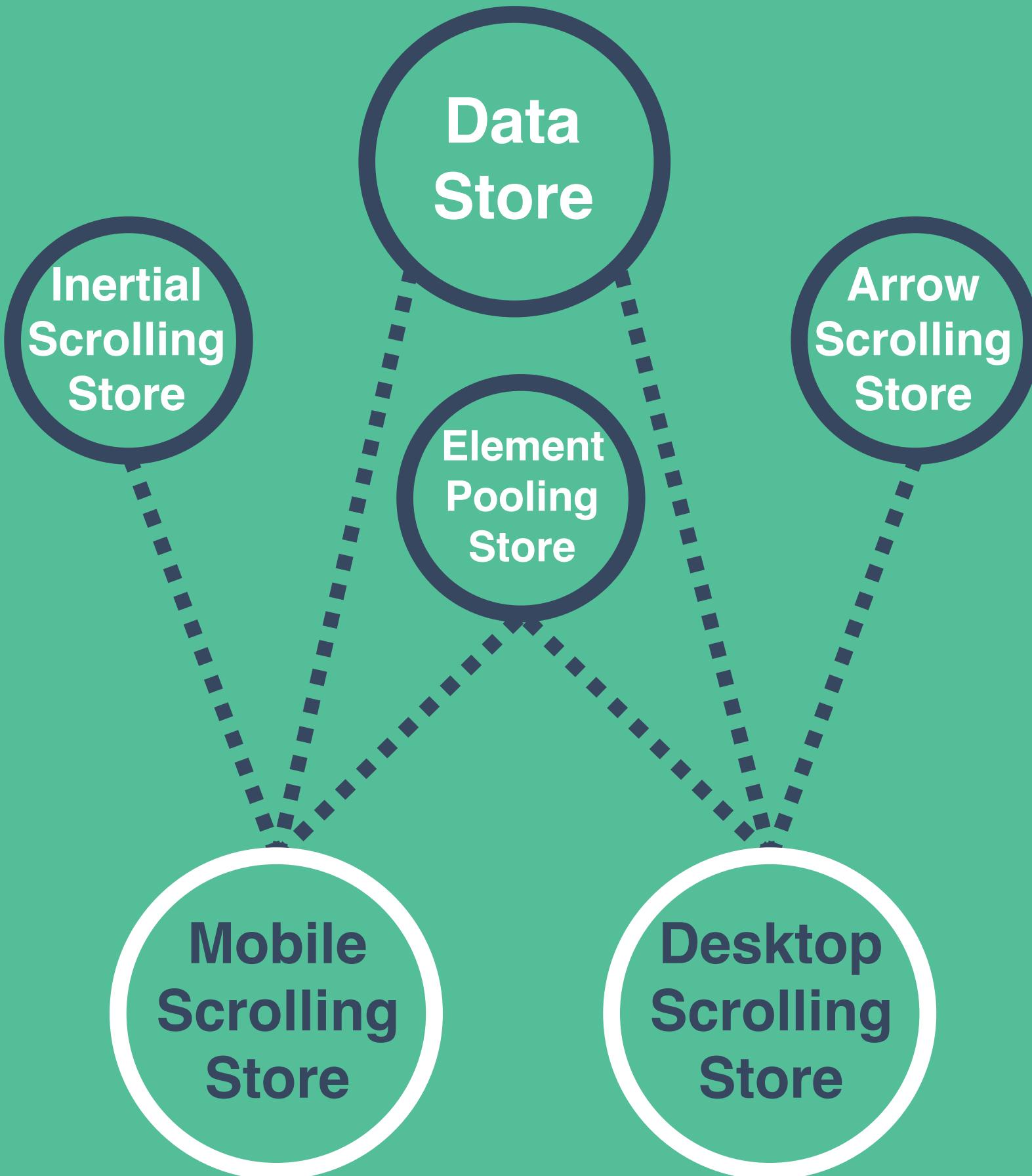
- » Good separation of concerns
- » Reusability of the Stores

```
1 const BigStore = types
2   .model('BigStore', {})
3   .views(self => {
4     get updatedArray(){
5       return self.itemArray.map(x => `big ${x}`);
6     }
7   })
8
9 const DataStore = types
10  .model('DataStore', {
11    itemArray: types.array(types.string)
12  })
13
14 const BigDataStore = types.compose(DataStore, BigStore);
15
16 const bigDataStore = BigDataStore.create({
17   itemArray: ['pen', 'sword']
18 });
19
20 bigDataStore.itemArray // ['pen', 'sword']
21 bigDataStore.updatedArray // ['big pen', 'big sword']
```

COMPOSITION REAL WORLD EXAMPLE



COMPOSITION REAL WORLD EXAMPLE



Data Store

Holds the data to render

Inertial/Arrow Scrolling

Manages scrolling

Element Pooling Store

Render only in view

MINDSET



DERIVE EVERYTHING

When you add a new property in the Model ask yourself: can I derive it somehow ?

“Anything that can be derived from the application state, should be derived. Automatically”

TAKEAWAYS

- » MobX helps you decoupling your code
- » MobX State Tree provides a structure
- » Shape your tree & setup the communication
- » Embrace Composition!
- » Embrace Reactivity!

THANKS

🤓 github.com/maxgallo/you-dont-know-mobx-state-tree
✉️ hello@maxgallo.io
twitter @maxgallo