YOU DON'T KNOW MOBX STATE TREE

HI 👋
I'M MAX GALLO

About me: 🍝 💻 🇬🇧 🎶 🏍 📷 ✈️ ✍️

Principal Engineer @ DAZN

twitter: @_maxgallo
more: maxgallo.io

AGENDA

» Part One: MobX

» Part Two: MobX State Tree

» Part Three: Designing a Reactive Project

PART ONE

MOBX

```
const navStoreViews = self => ({
    get currentPath() {
        return `/${self.path}`
    }
})

const NavStore = types
    .model('NavStore', navStoreModel)
    .views(navStoreViews);


test('NavStore has default value "home" for "path" ', t => {
    const navStore = NavStore.create({});

    t.is(navStore.path, 'home');
})
```

MOBX

» Simple introduction to Reactive Programming

» Flexible / Unopinionated

» Transparent Functional Reactive Programming

» Helps Decoupling View from Business Logic

MOBX OBSERVABLES & REACTIONS

```
import { observable, autorun} from 'mobx';

const album = observable({
    title: 'Californication',
    playCount: 0,
});

autorun(() => console.log(`New play count: ${album.playCount}`))
// New play count: 0

album.playCount = 1; // New play count: 1

album.playCount = 24; // New play count: 24
```

MOBX COMPUTED VALUES

```
import { observable, autorun, computed} from 'mobx';

const album = observable({
    title: 'Californication',
    playCount: 0,
});

const all = computed(() => album.title + album.playCount);

autorun(() => console.log(all))
// Californication0

album.playCount = 1;          // Californication1
album.title = 'OkComputer'; // OkComputer1
album.playCount = 24;         // OkComputer24
```

MOBX RECAP

Observable state

Mutable Application State

Reactions

Side effects like autorun or updating a React component

Computed Values

Automatically derived values, lazily evaluated

PART TWO

MOBX STATE TREE

```
const navStoreViews = self => ({
    get fullPath() {
        return `/${self.path}`
    }
})

const NavStore = types
    .model('NavStore', navStoreModel)
    .views(navStoreViews);


test('NavStore has default value "home" for "path" ', t => {
    const navStore = NavStore.create({});

    t.is(navStore.path, 'home');
})
```

MOBX STATE TREE

» Powered by MobX

» The State is strongly typed

» Opinionated / Ready to use

» Relies on the concept of Trees (Stores)

WHAT'S A TREE/STORE ?

```
import { types } from 'mobx-state-tree';

const CarStore = types
    .model('Car', {
        name: types.string          // mobx observable
    })
    .views(self => ({
        get isFerrari() {           // mobx computed
            return self.name.includes('Ferrari')
        }
    }))

const carStore = CarStore.create({ name: 'Ferrari Enzo'});
console.log(carStore.isFerrari); // true
```

MOBX STATE TREE STORES

```
const CarStore = types
    .model('Car', { // ------------------> Model
        name: types.string,
    })
    .views(self => ({ // -----------------> Views
        get isFerrari() {
            return self.name.includes('Ferrari')
        }
    }))

const CarParkStore = types
    .model('CarPark', { // --------------> Model
        cars: types.array(CarStore),
    })
    .actions(self => ({ // --------------> Action
        addCar(car) { self.cars.push(car) }
    }));

const carParkStore = CarParkStore.create(
    { cars: [ { name: 'Fiat 500'} ] }
);

carParkStore.addCar({ name: 'Ferrari 458 Italia' });
carParkStore.addCar({ name: 'Ferrari Enzo' });
```

Model

» Mutable observable state

» Contains type information

» 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 ?

```
/** ------------- App.js -------------------- **/

import { Provider }    from 'mobx-react'
import App             from './App.js'
import ShopStore       from './Shop.store.js'
import NavigationStore from './Navigation.store.js'

const shopStore = ShopStore.create()

ReactDOM.render(
    <Provider
        shop={shopStore}
        navigation={navigationStore}
    >
        <App />
    </Provider>,
    document.getElementById('root')
)
```

```
/** ---------- CheckoutView.js ---------- **/

import React                from 'react'
import { inject, observer } from 'mobx-react'

@inject('shop') @observer
class CheckoutView extends React.Component {
    render() {
        const { shop } = this.props;
        return (
            Amount:{ shop.checkountAmount }
        );
    }
}
```

MOBX STATE TREE STORES

DEEP DIVE

» Mutable and Immutable (Snapshots, Time Travelling)

» Composition

» Lifecycle Methods

» Dependency Injection

```
1  import { getEnv, types } from 'mobx-state-tree';
2
3  const CarParkStore = types
4      .model('CarPark', {
5          cars: types.array(CarStore),
6      })
7      .actions(self => ({
8          downloadCars() {
9              getEnv(self).getCarsFromBackend()
10                 .then(cars => self.cars = cars);
11         }
12     }));
13
14 const getCarsFromBackend = function() {
15     /** Fetching Cars from Backend **/
16 }
17
18 const carParkStore = CarParkStore.create(
19     { cars: ['Fiat 500'] },  // initial state
20     { getCarsFromBackend }   // environment
21 );
```

MOBX STATE TREE STORES

DEPENDENCY INJECTION

» Inject anything

» Environment is shared per tree

» Useful for testing

PART THREE

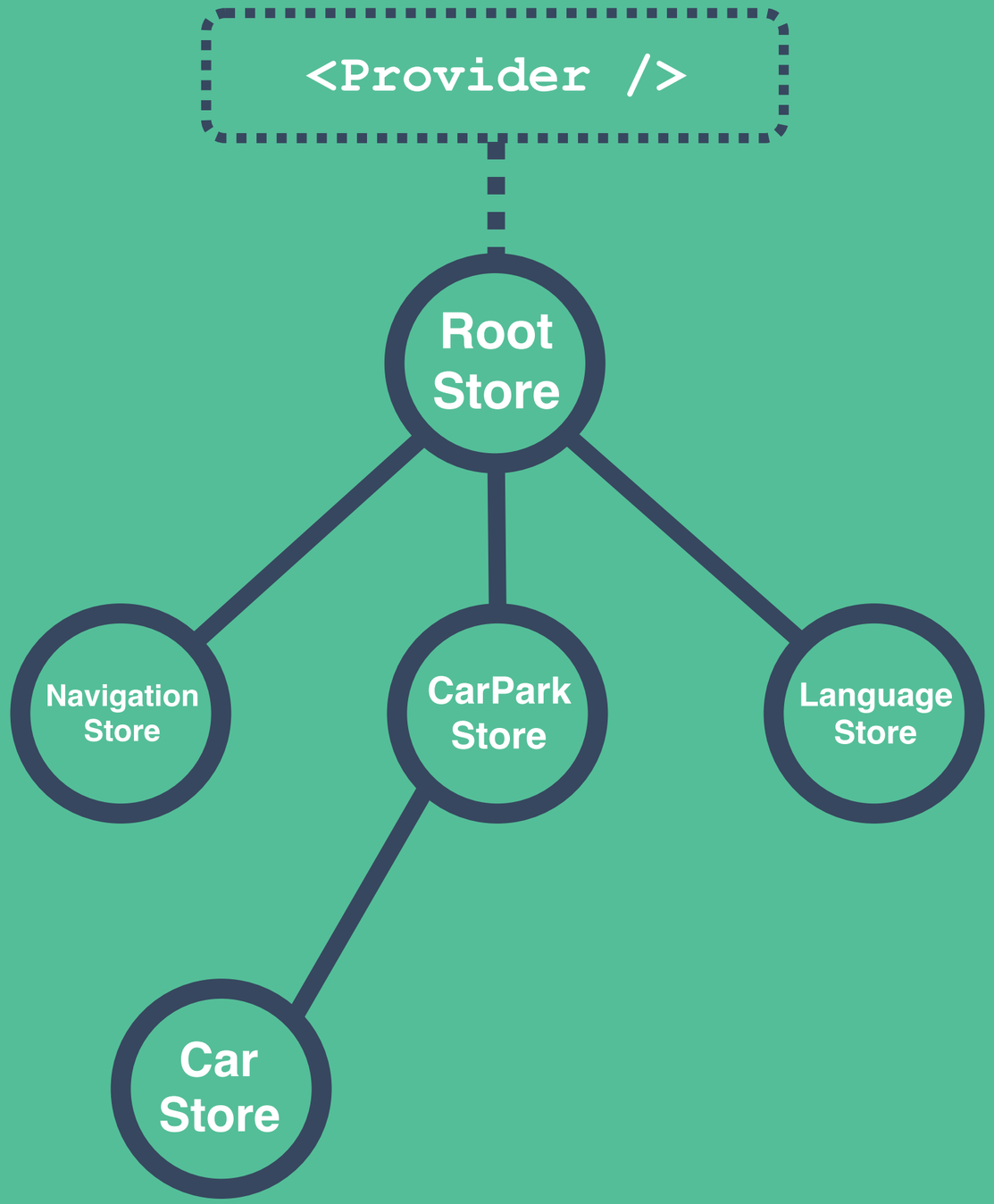
DESIGNING A REACTIVE PROJECT

```
const navStoreViews = self => ({

})

const NavStore = types
    .model('NavStore', navStoreModel)
    .views(navStoreViews);


test('NavStore has default value "home" for "path" ', t => {
    const navStore = NavStore.create({});

    t.is(navStore.path, 'home');
})
```

DESIGNING STORES

1. Shape your Trees
One Root Store vs Multiple Root Stores

2. Stores Communication
How Stores communicate between each other



SHAPE YOUR TREES
ONE ROOT STORE

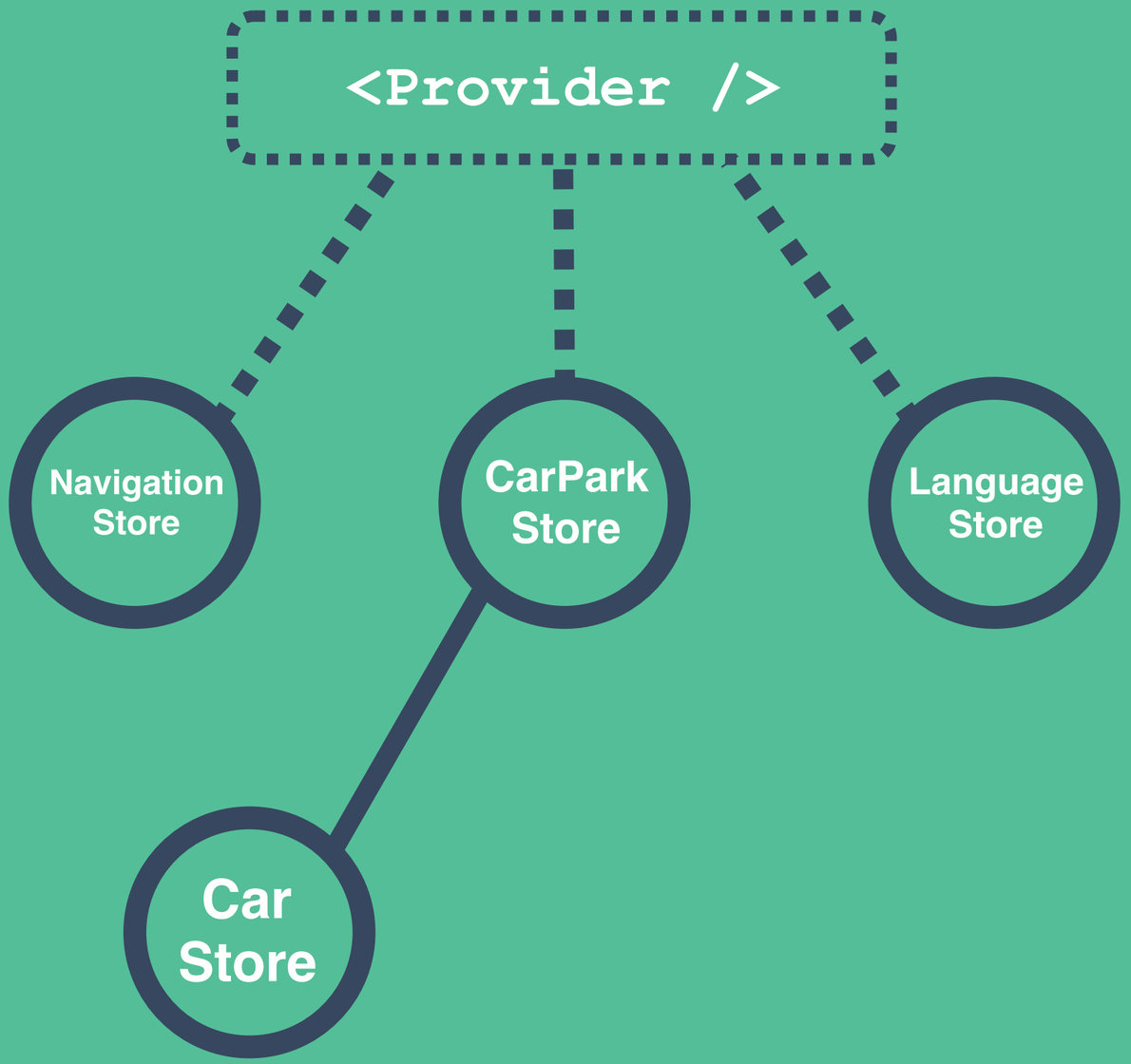
Pros

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

» Unique environment for dependency injection.

Cons

Very easy to create tightly coupled stores



SHAPE YOUR TREES

MULTIPLE ROOT STORES

Pros

Easier to reason by Domain

Cons

» Less immediate to perform actions on everything

» Not single environment for dependency injection

REAL WORLD
STORES COMMUNICATION 

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

STORES COMMUNICATION
DEFAULT APPROACH

```
/** ------------- Root.store.js ---------------- **/

import { types } from 'mobx-state-tree';

const RootStore = types
    .model('RootStore', {
        navStore  : types.maybe(NavStore),
        pageStore : types.maybe(PageStore)
    })

/** ------------- Page.store.js ---------------- **/

import { types, getParent } from 'mobx-state-tree';

const PageStore = types
    .model('PageStore', {
        currentView : types.option(types.string, '')
    })
    .actions(self => ({
        showLoginForm() {
            self.currentView = 'login';
            getParent(self).navStore.setPath('/login')
        },
    }));
```

Each Store access directly other Stores.

» Easier when using a Single Root Store

» Each Store could end up knowing the whole structure ⚠️

STORES COMMUNICATION
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

» Friendly interface

» Knows a lot about your App

```
/** ---------- Region.store.js ---------------- **/
const RegionStore = types
    .model('RegionStore', {
        region: types.optional(types.string, 'UK')
    })

/** ---------- Navigation.store.js ------------ **/
import { types, getEnv } from 'mobx-state-tree';

const NavigationStore = types
    .model('NavigationStore', { path: types.string })
    .view(self => ({
        get urlPath() {
            return getEnv(self).regionStore.region
                + '/' + self.path;
        }
    }));

/** ---------- index.js ----------------------- **/
const regionStore = RegionStore.create({});
const navigationStore = NavigationStore.create(
    { path: 'login'},
    { regionStore }
);

console.log(navigationStore.urlPath); // 'UK/login'
```

STORES COMMUNICATION
DEPENDENCY INJECTION

Injecting one or multiple stores into another one.

» You could use it for both Actions and Views

» Circular dependencies while loading could be non-trivial

ONE MORE THING ...

STORE COMPOSITION

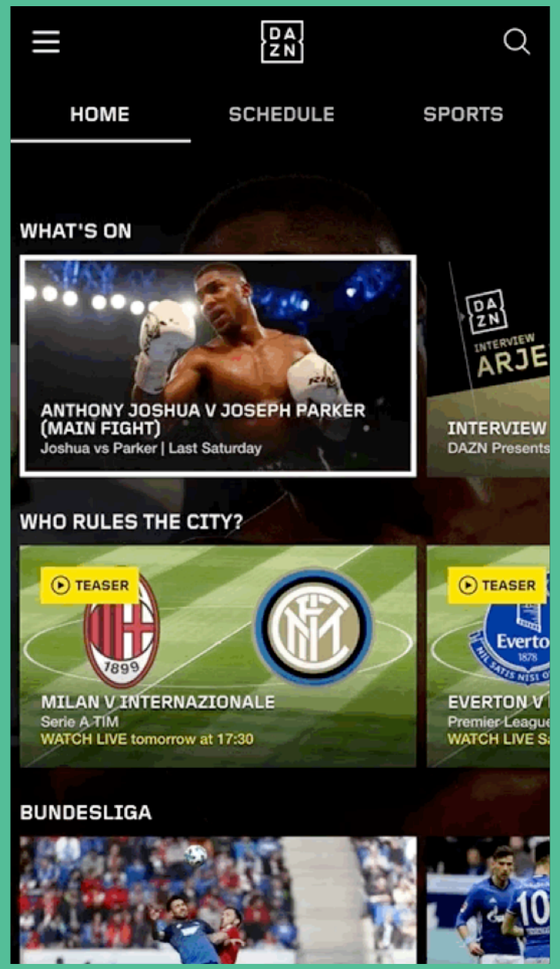
```
const BigStore = types
    .model('BigStore', {})
    .views(self => ({
        get updatedArray(){
            return self.itemArray.map(x => `big ${x}`);
        }
    }))

const DataStore = types
    .model('DataStore', {
        itemArray: types.array(types.string)
    })

const BigDataStore = types.compose(DataStore, BigStore);

const bigDataStore = BigDataStore.create({
    itemArray: ['pen', 'sword']
});

bigDataStore.itemArray    // ['pen', 'sword']
bigDataStore.updatedArray // ['big pen', 'big sword']
```

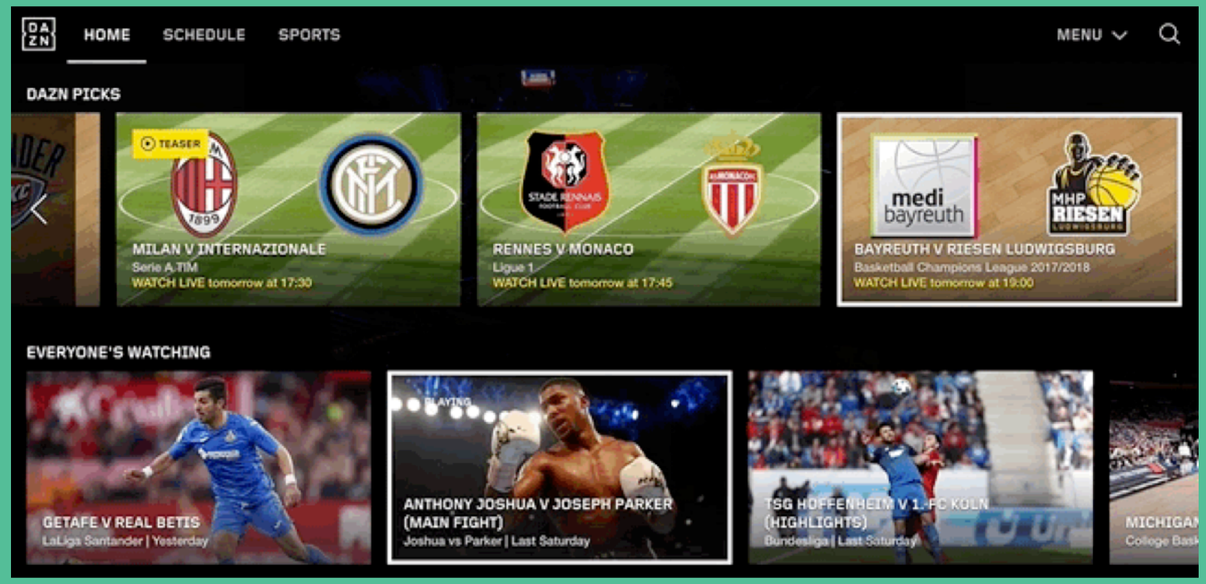
Two or more stores can be composed

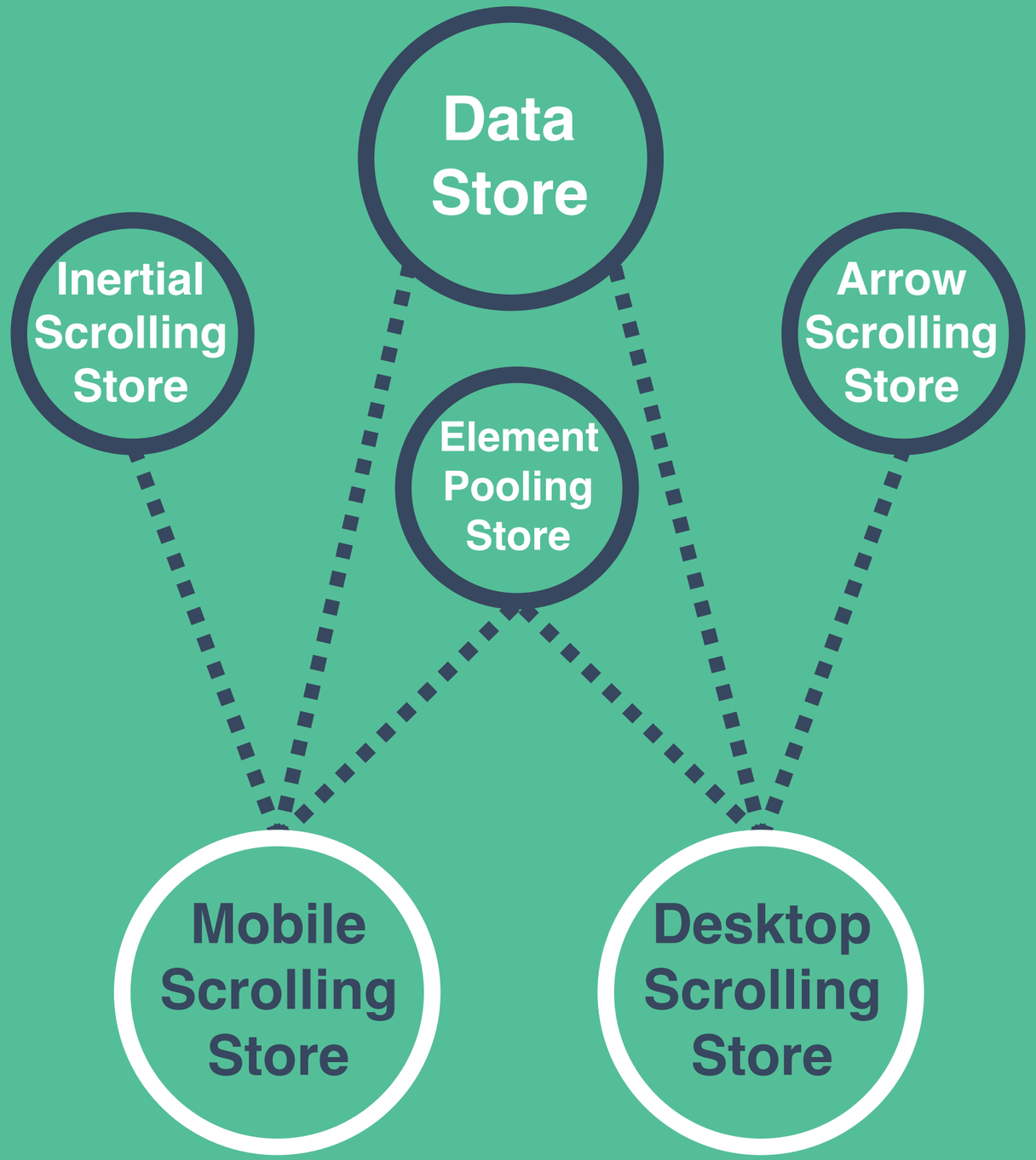
» Separation of Concerns

» Reusability

COMPOSITION
REAL WORLD EXAMPLE







COMPOSITION

REAL WORLD EXAMPLE

Data Store
Holds the data to render

Inertial/Arrow Scrolling
Manages scrolling

Element Pooling Store
Renders only in view

CONCLUSIONS

DERIVE EVERYTHING

```
const navStoreViews = self => ({
}
const NavStore = types
    .model('NavStore', navStoreModel)
    .views(navStoreViews);


test('NavStore has default value "home" for "path" ', t => {
    const navStore = NavStore.create({});

    t.is(navStore.path, 'home');
})
```

Next time you're adding properties to the Model, ask yourself first

CAN I DERIVE IT? 

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

TAKEAWAYS

» MobX opens the doors of Reactive Programming

» 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

web maxgallo.io