YOU DON'T KNOW MOBX STATE TREE

HI 👋

I'M MAX GALLO

PRINCIPAL ENGINEER AT DAZN

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

THE PLAN

» MobX intro

» MobX State Tree overview

» Designing a Reactive Project

» Best Practises

IF MOBX IS THE ENGINE
MOBX STATE TREE IS THE CAR

MOBX

THE ENGINE OF THE CAR

» Decouples View from Business Logic

» Uses Reactive paradigms

» Unopinionated

MOBX PILLARS

Observable state

Mutable Application State

Computed Values

Automatically derived values

@Observer

Subscribing to observables

Reactions

Side effects like updating a React component

```
class CarPark {
    @observable cars = []
    @computed get howManyFerrari() {
        return this.cars.filter(this.isFerrari).length;
    }
    isFerrari = carName => carName.includes('Ferrari')
}

@observer
class CarParkView extends Component {
    renderCar(car){ return <li key={car}>{car}</li> }
    render() {
        const { carPark } = this.props
        return (<div>
            <ul>{ carPark.cars.map(this.renderCar) } </ul>
            How many Ferrari? { carPark.howManyFerrari }
        </div>);
    }
}

const carPark = new CarPark();

render(<CarParkView carPark={carPark} />, element('root'));

setTimeout( () => carPark.cars.push('Ferrari 458')  , 1000);
setTimeout( () => carPark.cars.push('Ferrari Enzo') , 2000);
```

MOBX

Observable state

Mutable Application State

Computed Values

Automatically derived values

@Observer

Subscribing to observables

Reactions

Side effects like updating a React component

MOBX STATE TREE

» Opinionated / Ready to use

» Powered by MobX

» Relies on the concept of Trees (Stores)

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

const CarParkStore = types
    .model('CarPark', {
        cars: types.array(CarStore),
    })
    .views(self => ({
        get howManyFerrari(){
            return self.cars.filter(
                car => car.isFerrari
            ).length;
        }
    }))
    .actions(self => ({
        addCar(car) { self.cars.push(car) }
    }));

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

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

WHAT'S A TREE ?

ALSO KNOWN AS STORE

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

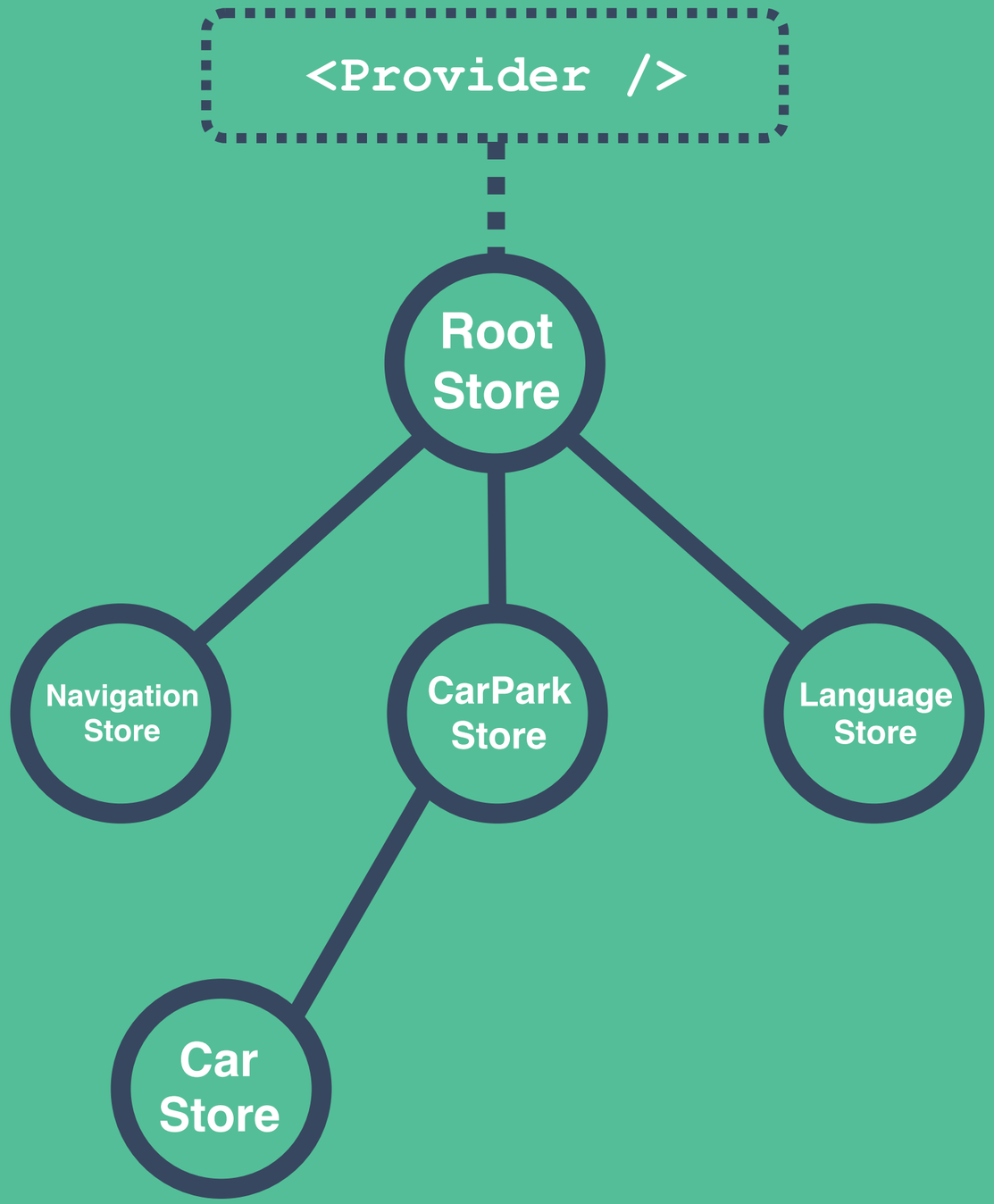
» Environment is shared per tree

» Useful for testing

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

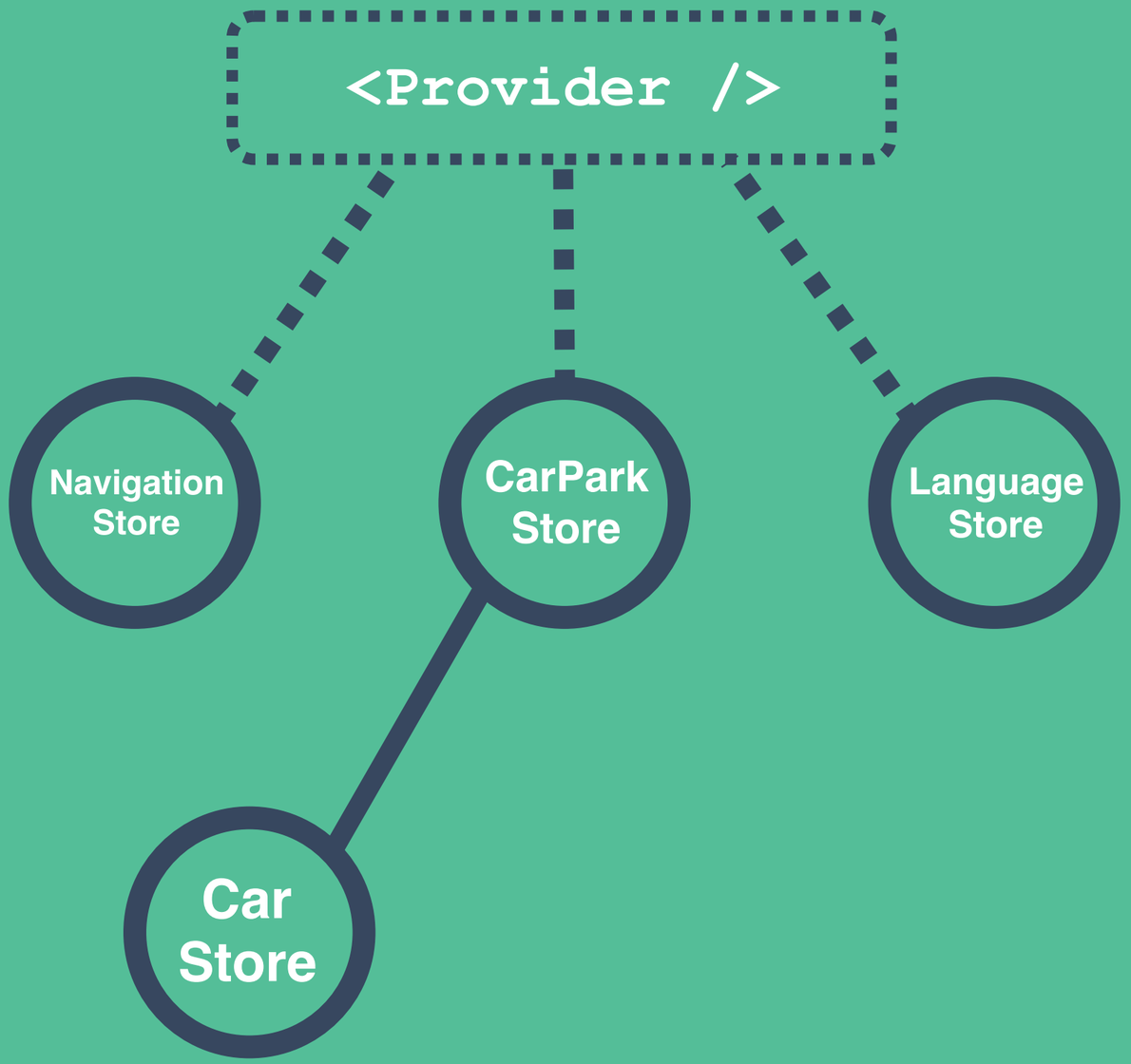
Pros

» 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

» 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 region() {
            getEnv(self).regionStore.region;
        },
        get urlPath() {
            return `${self.region}/${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

» Carefull about circular dependencies

ONE MORE THING ...

STORE COMPOSITION

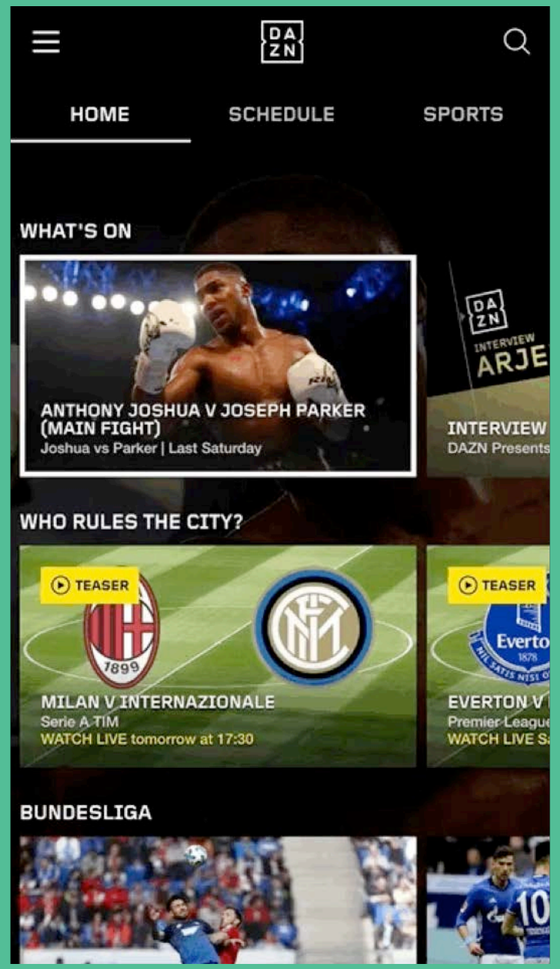
```
const BigStore = types
    .model('BigStore', {})
    .views(self => ({
        get updatedArray(){
            return self.itemArray.map(x => `big ${x}`);
        }
    }))

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']
```

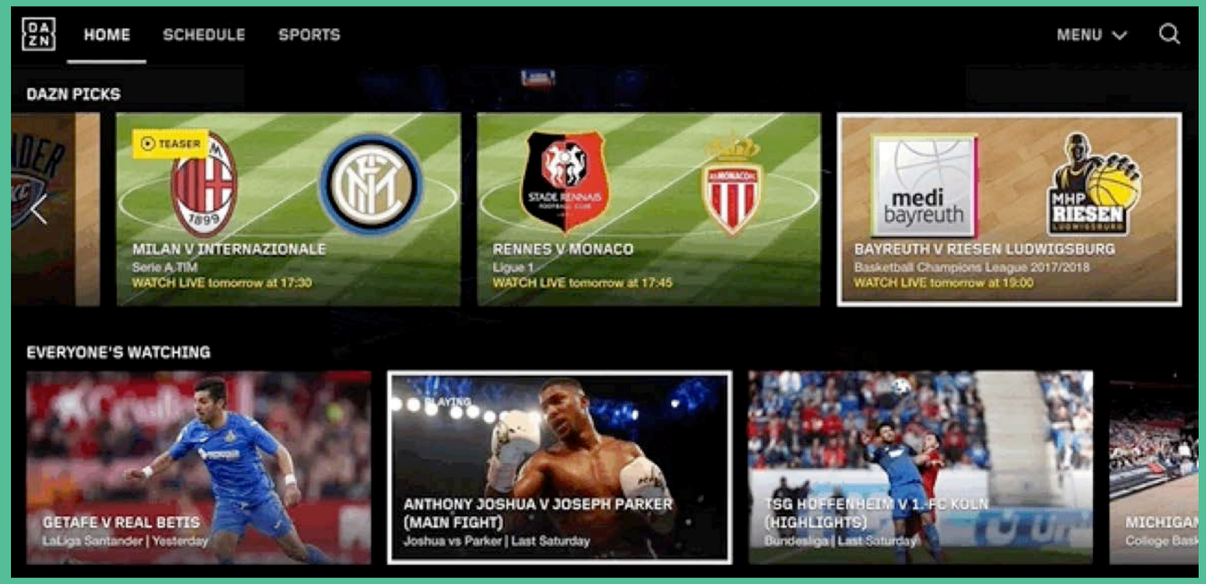
Two or more stores can be composed

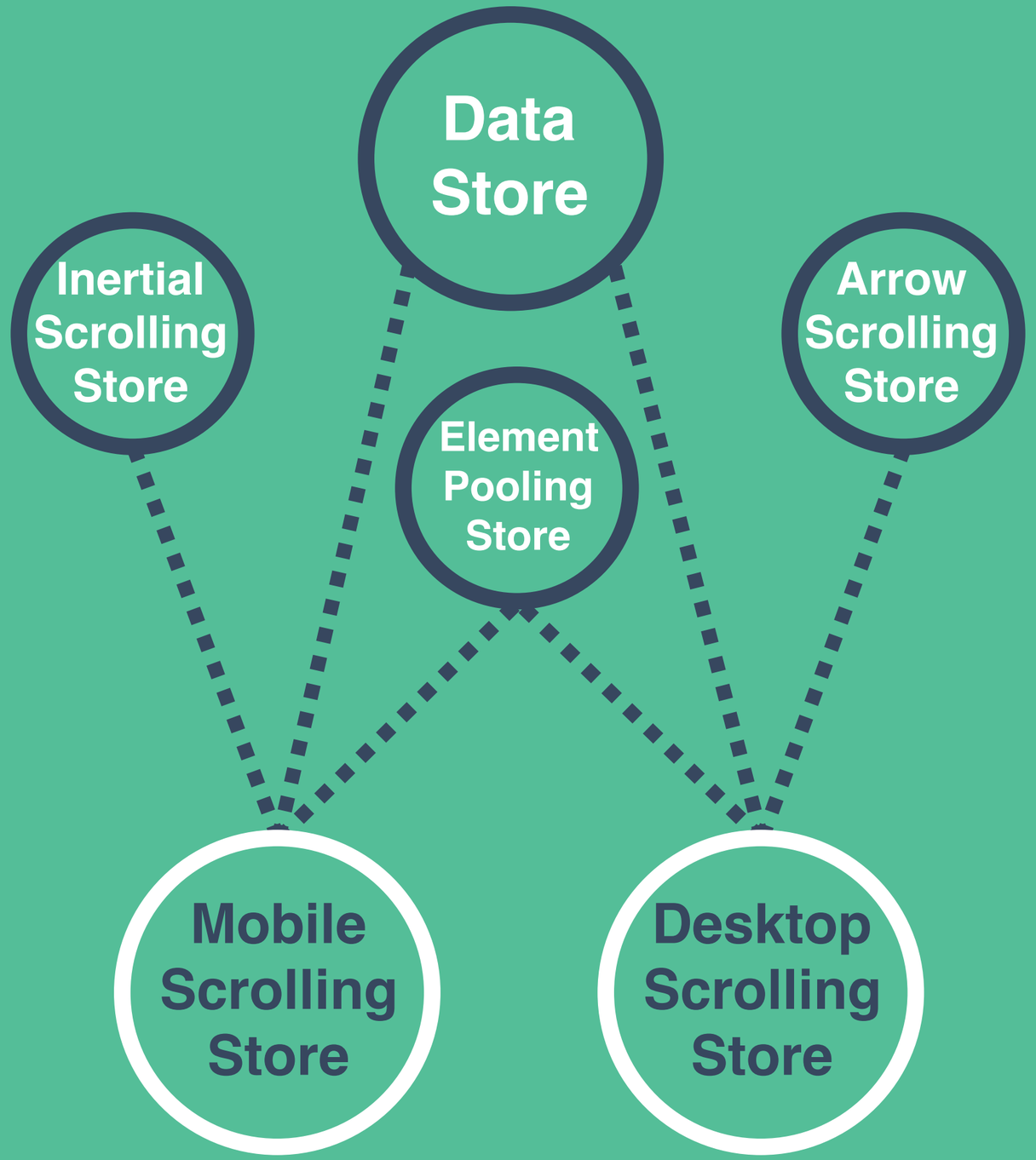
» Separation of Concerns

» 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

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