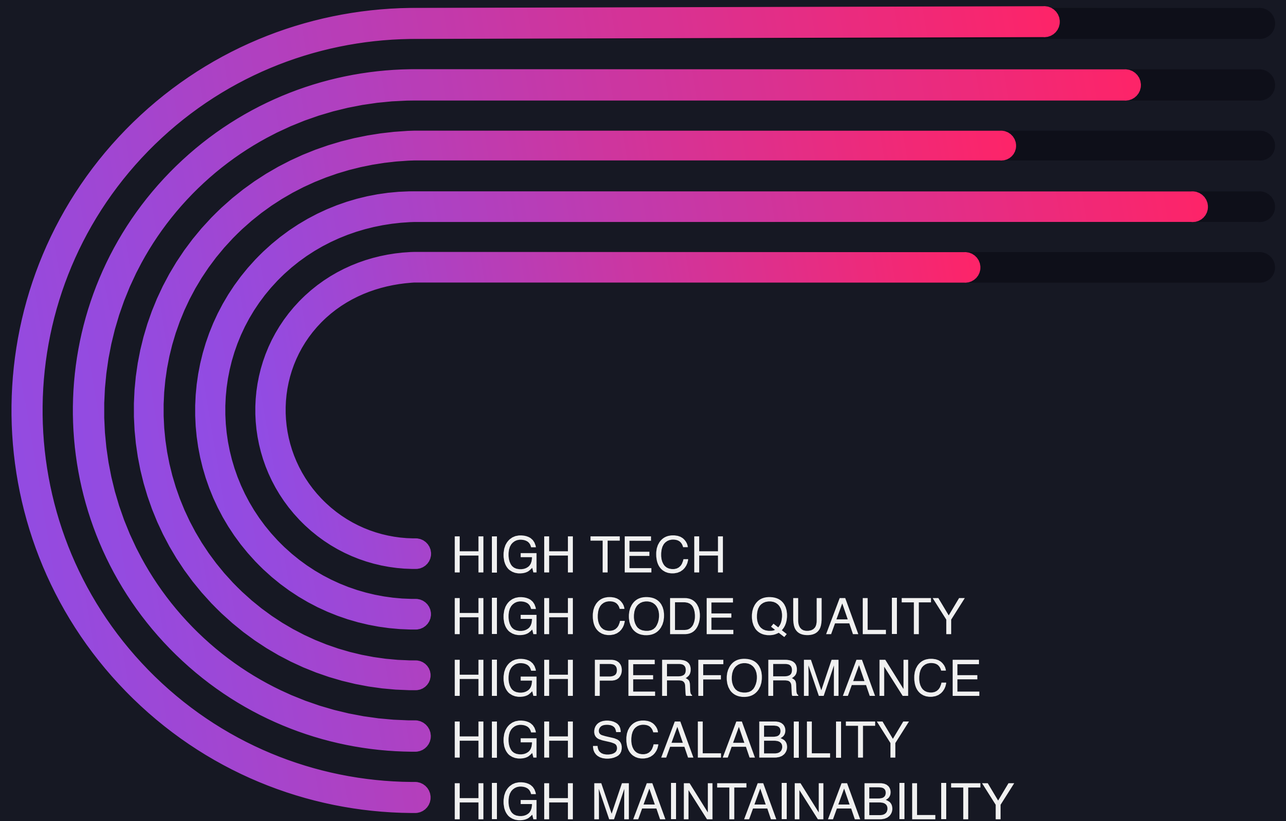

VIETNAM
WEB
SUMMIT



HIGH TECH
HIGH CODE QUALITY
HIGH PERFORMANCE
HIGH SCALABILITY
HIGH MAINTAINABILITY

Single Page App

SEXY ANGULAR STACK

CURIOUS CASES

by @cuongtruong

Hello!

I am Cuong Truong

FE Architect/Scrum Master

I am here because I love to share knowledge

You can contact me via
cuongtruong.info@gmail.com

Agenda

Angular
State Container
ES6+
TypeScript
Linter
UI kit
Unit Test
Webpack
Code Structure
Take away + Demo

ANGULAR

#Framework for building SPA

Uni-directional data flow

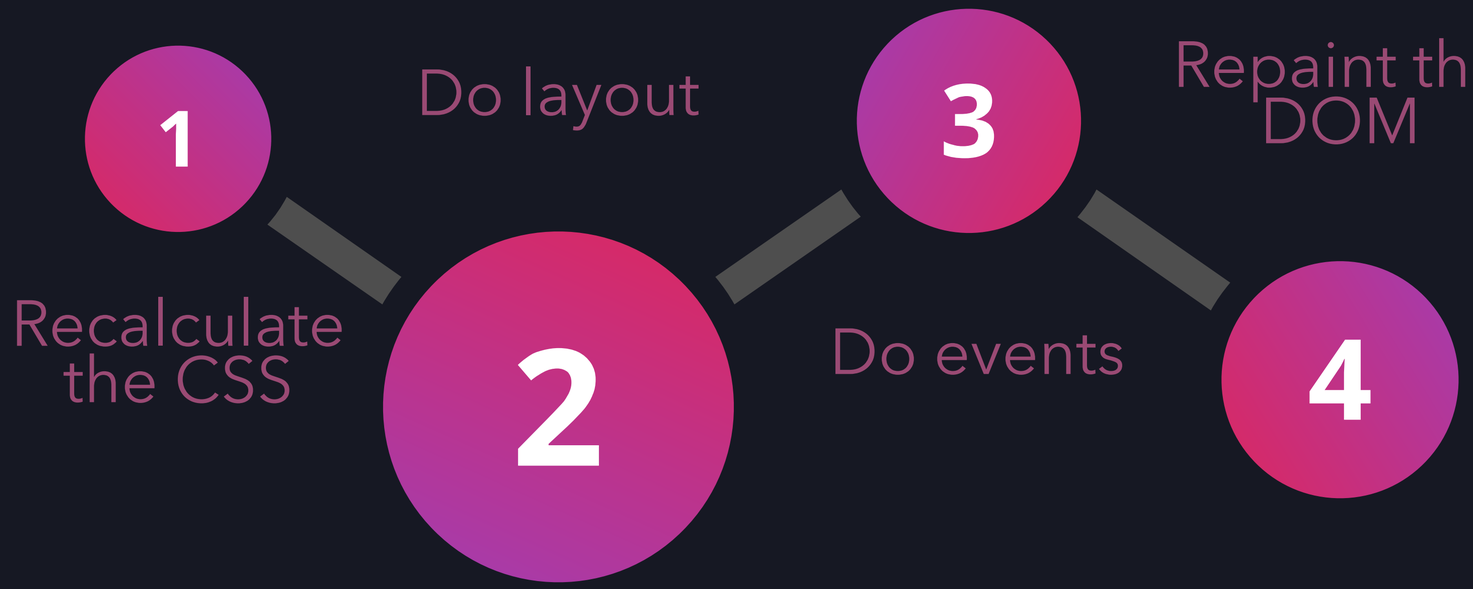
Component based

Across all platforms

Server-side rendering

#Cheat Sheet

Every time DOM changes, browser needs to



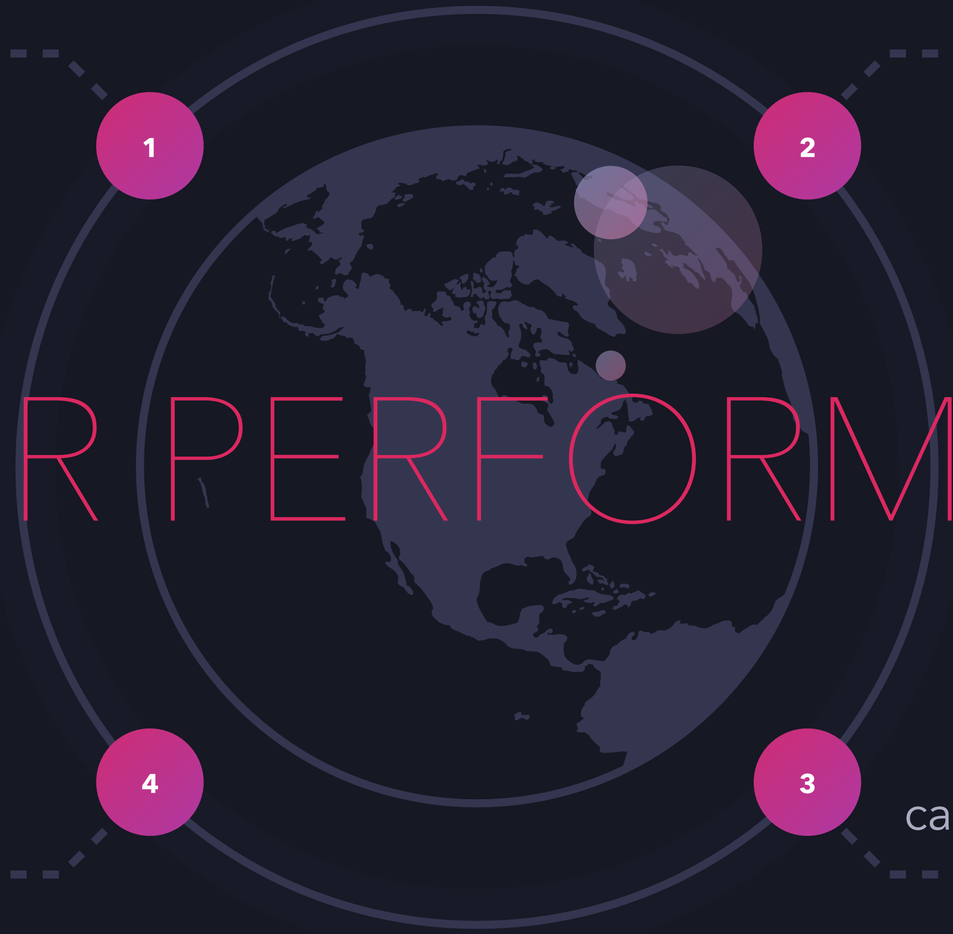
TAKES TIME

TRADITIONAL WAY

Modifies DOM when MODEL changed

USE "1-WAY"
Unidirectional Data Flow

1



2

RUN
change detection on the Model

BETTER PERFORMANCE

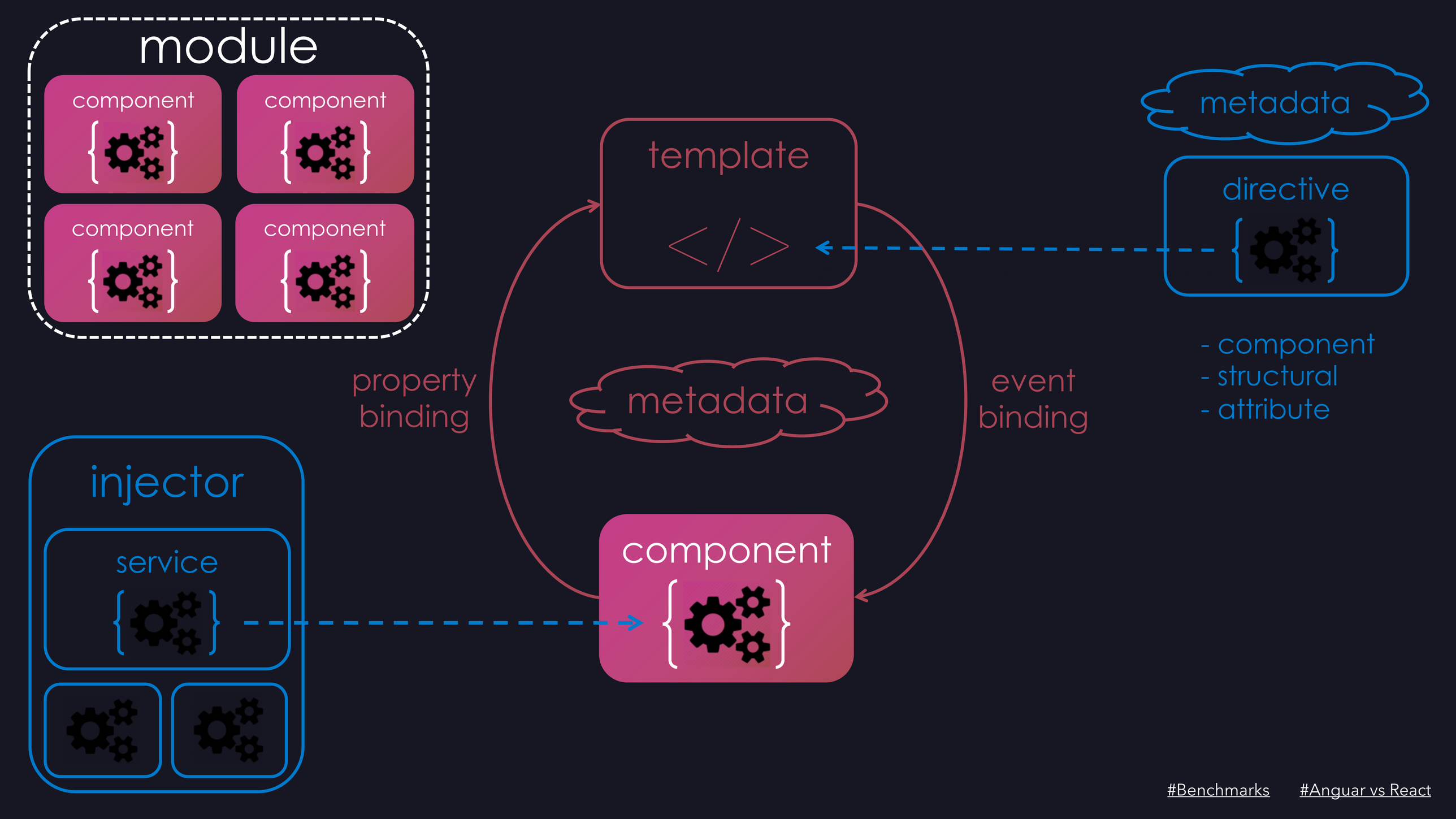
BATCH
Update bindings, execute all changes

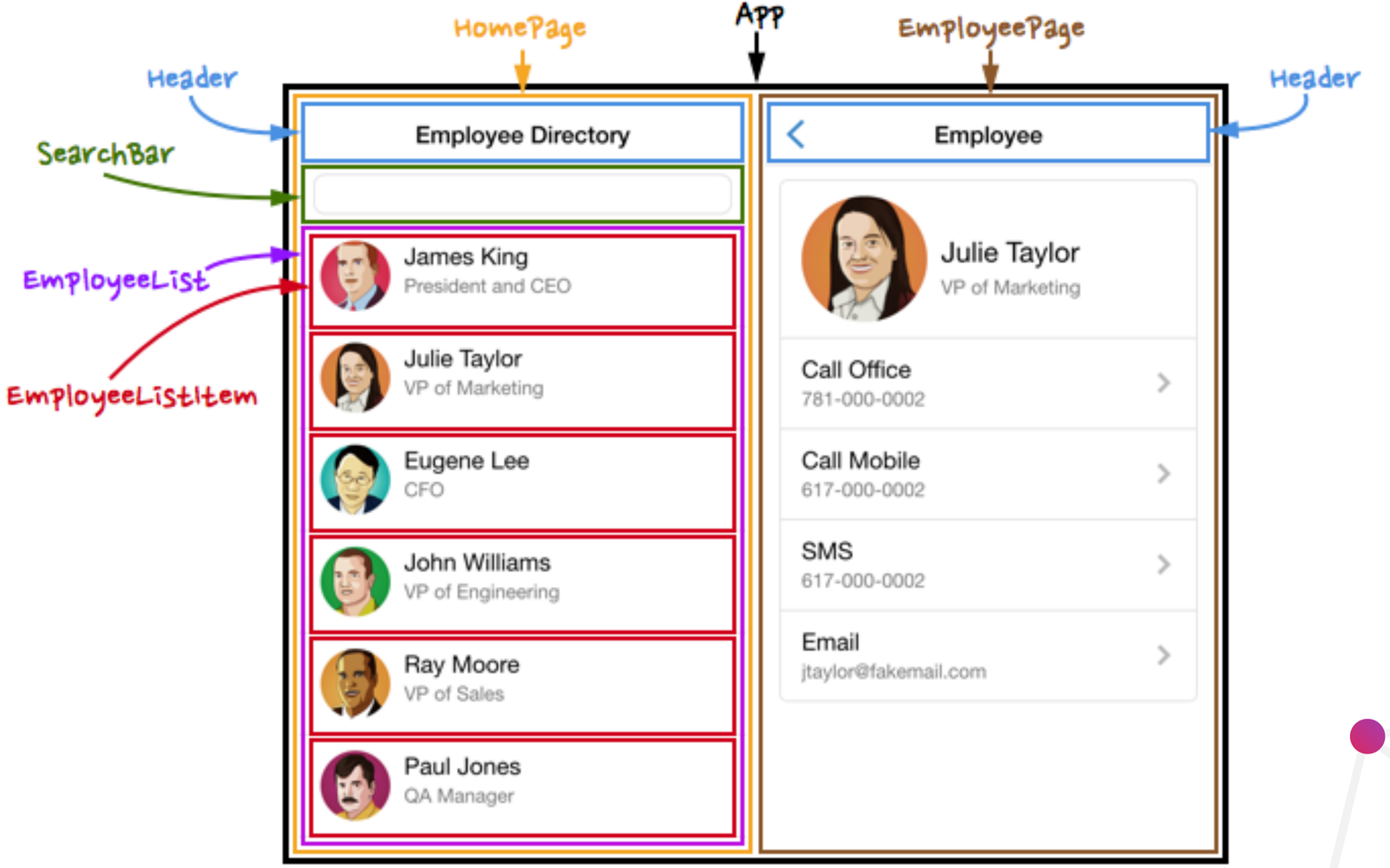
4

3

COMPUTE
minimal set of changes cause DOM to be updated

ANGULAR WAY


module
component
component
component
component
template
metadata
directive
- component
- structural
- attribute
property
binding
metadata
event
binding
injector
service
component
#Benchmarks
#Anguar vs React

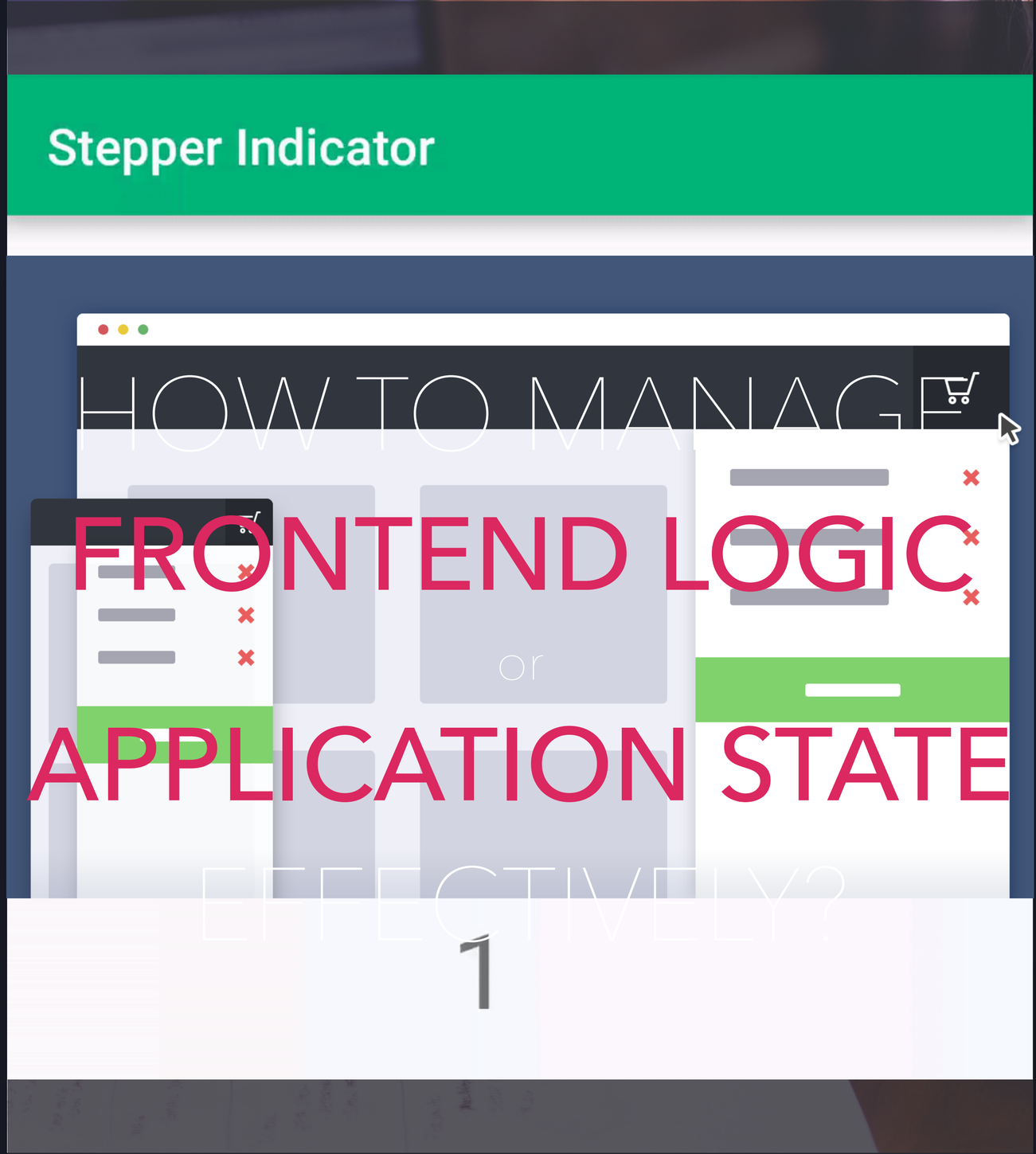


THINKING IN ANGULAR

- App
  - HomePage
    - Header
    - SearchBar
    - EmployeeList
      - EmployeeListItem
  - EmployeePage
    - Header
      - EmployeeDetails

If you need

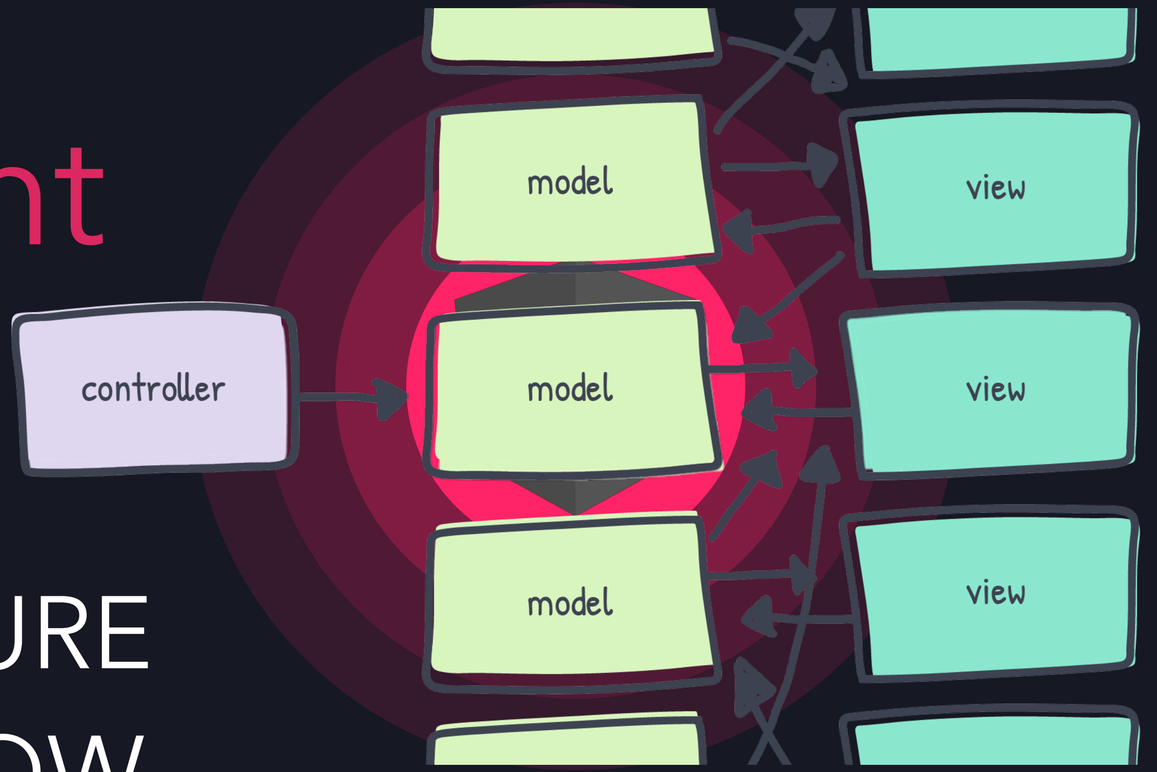
- A solution for concurrent data modification by multiple actors
- A client container for temporary UI state. e.g. wizard, shopping cart, …
- A client cache for avoiding excessive HTTP requests
- A solution for Undo/Redo



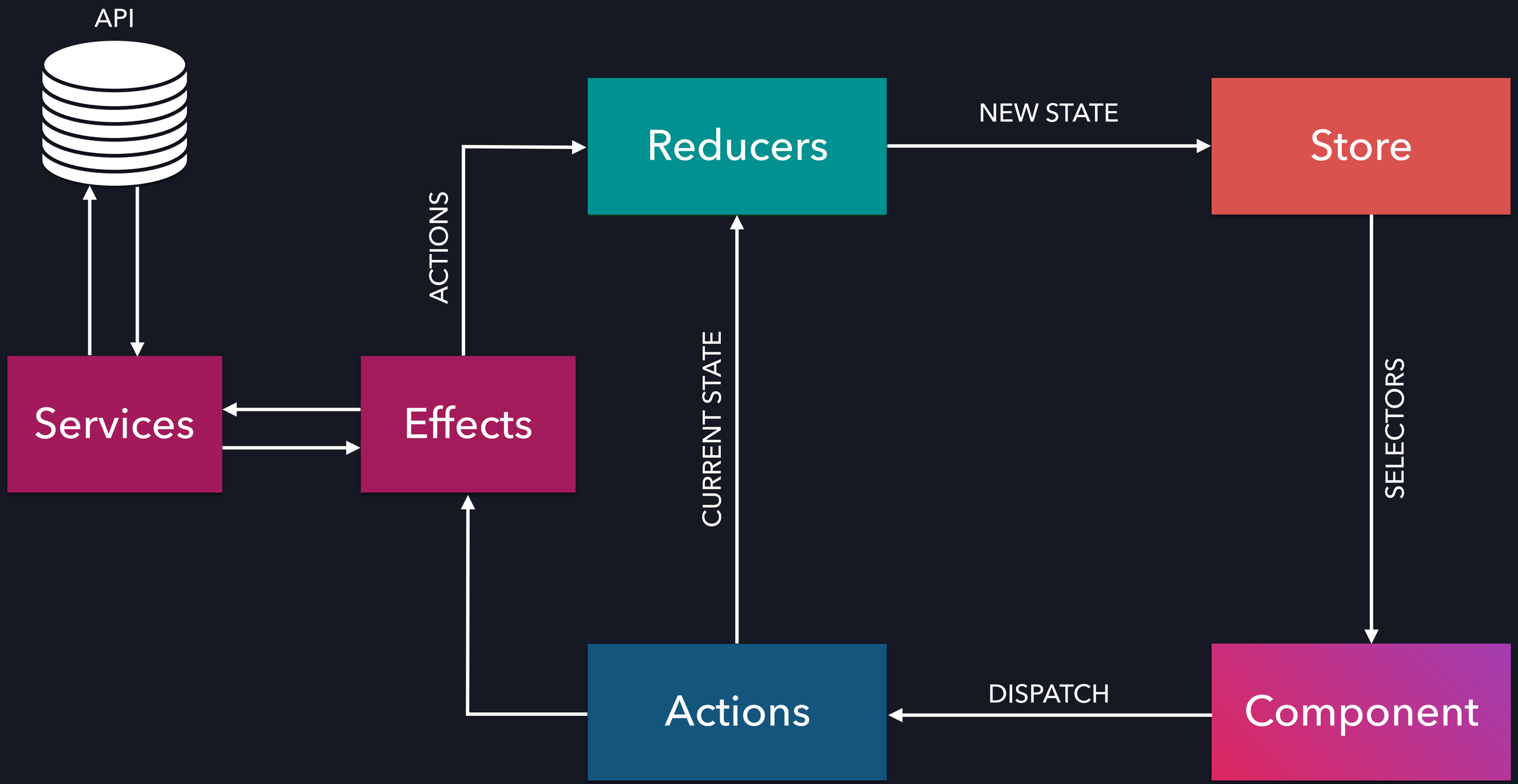
NgRx Store

state management

for Angular applications



IMMUTABLE DATA STRUCTURE
UNIDIRECTIONAL DATA FLOW


API
Reducers
NEW STATE
Store
ACTIONS
CURRENT STATE
SELECTORS
Services
Effects
Actions
DISPATCH
Component

@ngrx/store

vs

services

#ngrx/store vs services? #DoWeReallyNeedStateManagement? #Akita

ECMAScript 6+

Let + Const

```
const myObject = {propA, propB}
```

Arrow function, Default

```
const myFunction = (arg = ‘value’) => {}
```

Class

```
export class MyClass {constructor(){}}
```

Module

```
import {MyClass} from ‘my/path’
```

Template String

```
const tps = `Template string with ${…}`
```

Destructuring

```
const {propA, propB} = myObject
```

Rest, Spread ...

Promise

```
myPromise.then().then().catch()
```

Async/Await

```
const myFunction = async () => {await…}
```

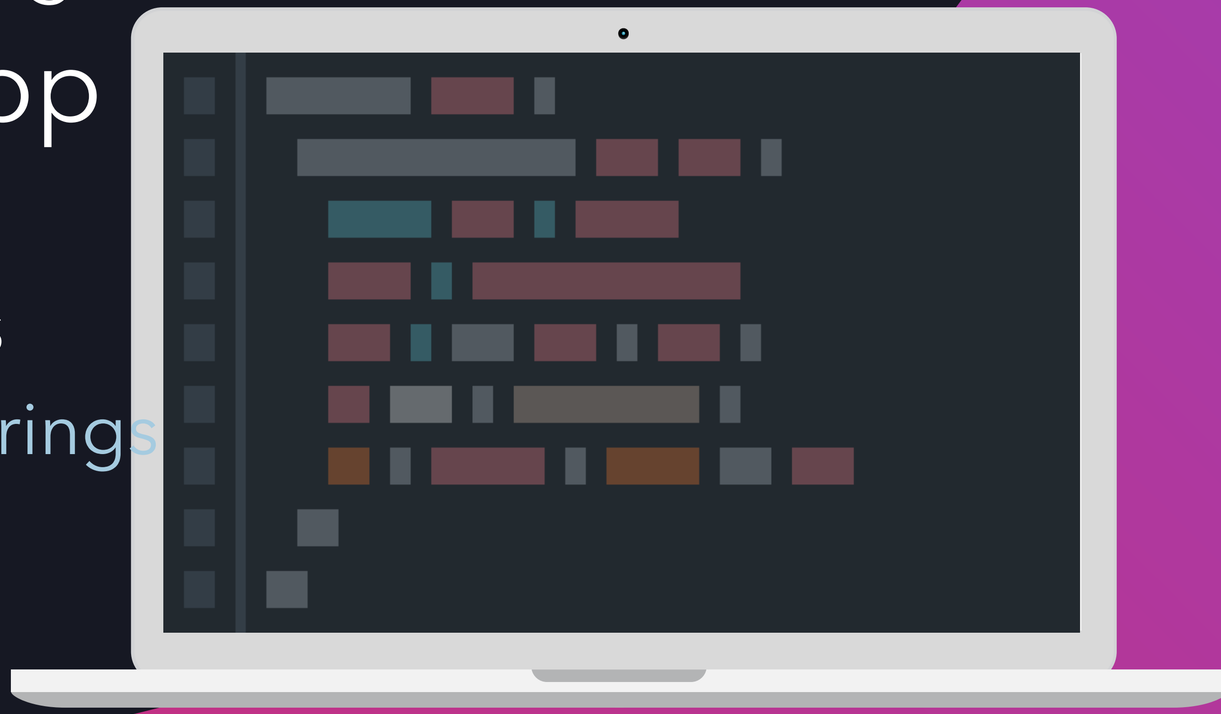
Decorator

```
@myDecorator
class MyClass {…}
```

#ES6 #Mozilla #ES-Next

TypeScript

- ✓ Static type checker for JS
  - strong/statically typed advantages
- ✓ Strong tool for LARGE app
  - covers 100% of code
  - catches incorrect assumptions
    - e.g. numbers represented as strings
- ✓ Provides advanced
  - autocompletion / intellisense
  - navigation
  - safe refactoring



#WhyTypeScript

- ✗ project does not live for long
- ✗ project is really simple
- ✓ people enter or leave your team frequently
- ✓ there is a chance you will need to refactor the thing
- ✓ system is very important or even crucial for the success of company

My Recommendation

TSLint/ESLint

discovers problems with TypeScript code without executing it

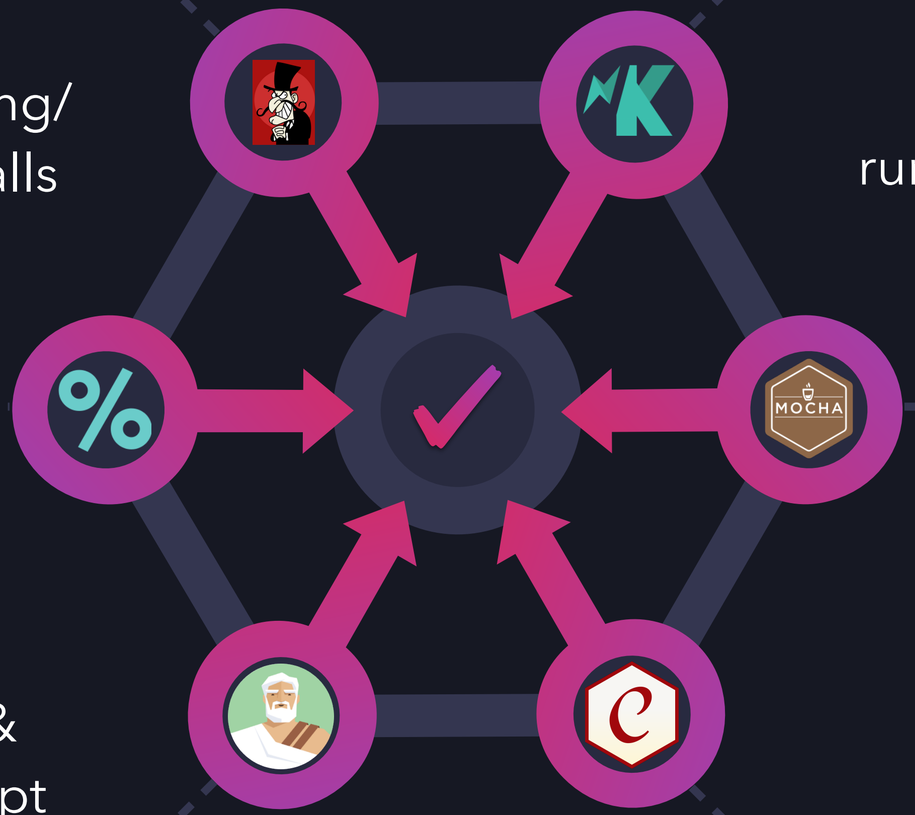
Coding conventions
Readability
Maintainability
Functionality errors

#TSLint #ESLint

Front-end Unit Test

FAKER
Some other mocking/
#faking data/api calls

KARMA
Spectacular #test
runner for JavaScript



ISTANBUL
JS #code coverage tool

MOCHA
Feature-rich JavaScript
#test framework

SINON
Test #spies, stubs &
mocks for JavaScript

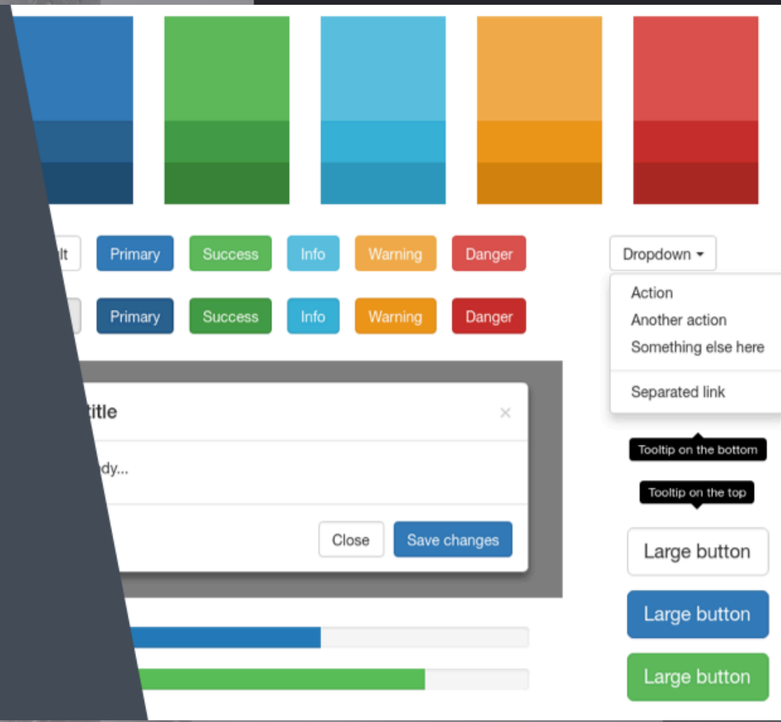
CHAI
BBD/TDD
#assertion library

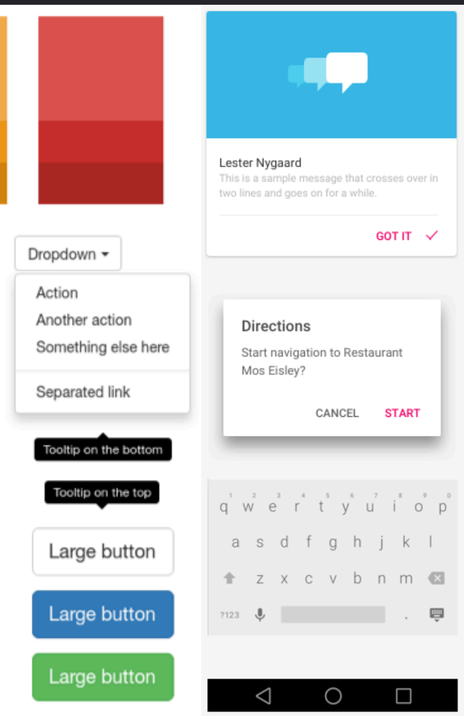
#Protractor

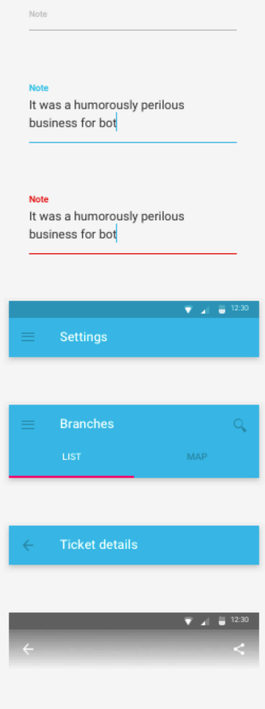
Build yourself ?!

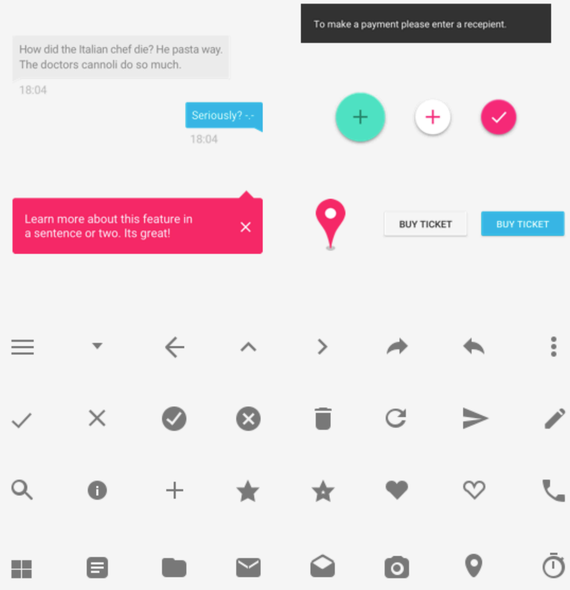
Bootstrap

UI Kit









Ng Bootstrap

Material UI

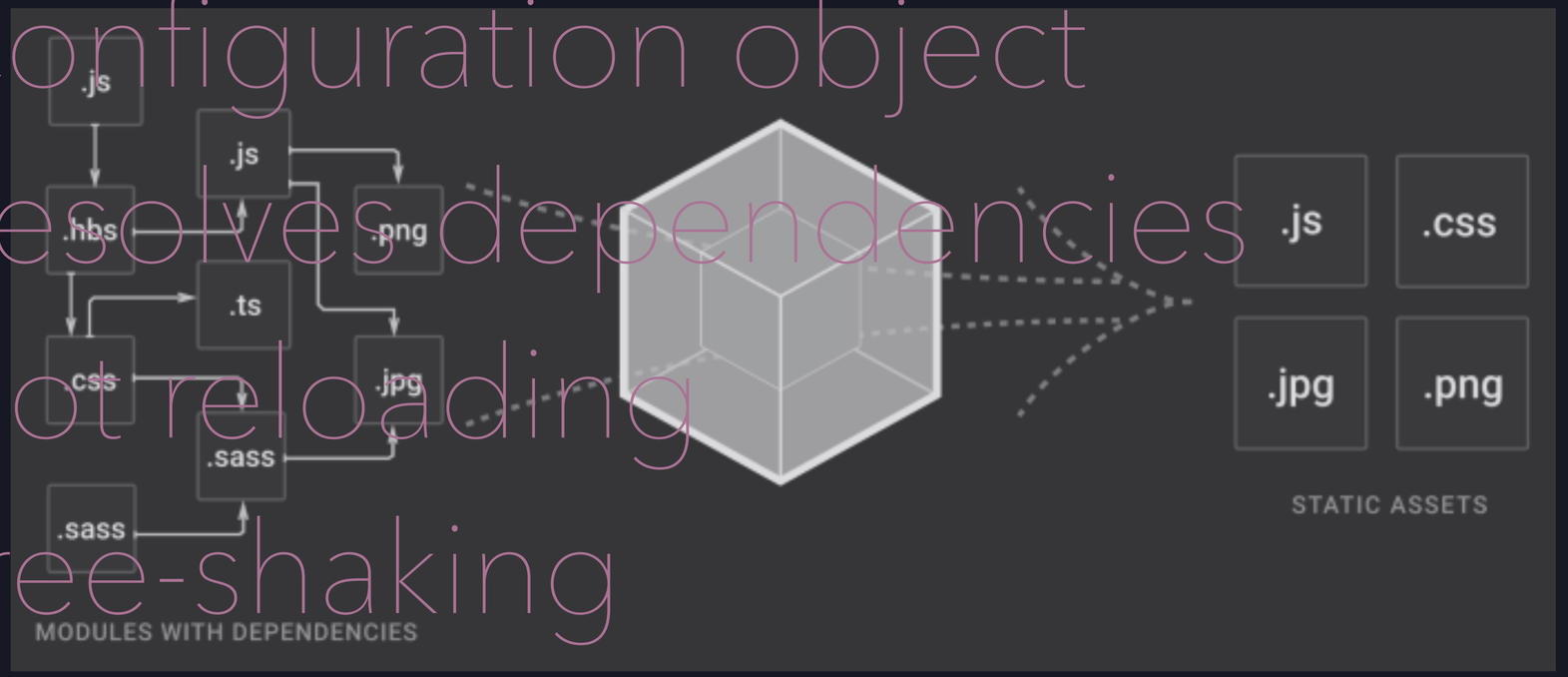
#BEM #SMACSS #MVCSS

WEBPACK

Bundles your

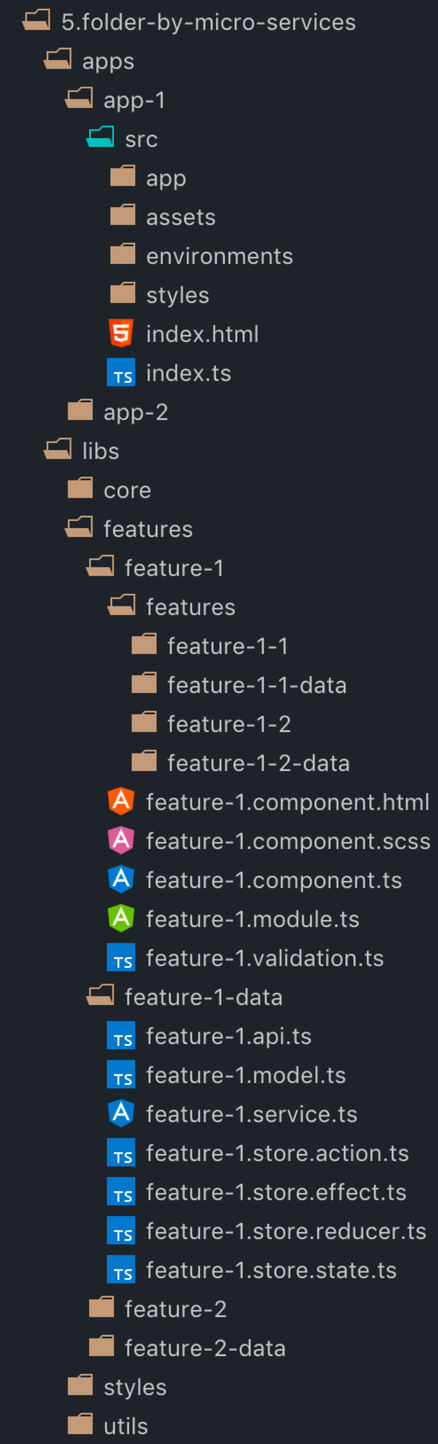
Scripts
Styles
Assets
Images

Configuration object
Resolves dependencies
Hot reloading
Tree-shaking
Cache busting/hashing system



Code structure?

CODE STRUCTURE



take away

Pros

- ✓ One framework to rule them all
- ✓ Full control/easy to manage coding flow
- ✓ Building reusable code/components
  - Testable
  - Readable
  - Maintainable
- ✓ Performance
- ✓ Developer experience/community
- ✓ Server-side rendering => SEO
- ✓ Size is smaller after each release

Cons

- ✔ Learning curve
- ✔ Need to keep conventions/boilerplate
- ✔ Need to care about performance
- ✔ All in One framework for small app

My Recommendation

- ✓ People are a part of solution
- ✓ Keep conventions/commitments
- ✓ Strategies for managing dependencies
- ✓ Take care about architecture and configurations
- ✓ Write small & pure functions as much as possible
- ✓ Write meaningful unit test as much as possible
- ✓ Don’t USE if you don’t NEED or UNDERSTAND it

References

#JSCodingStandards
#AngularStyleguide

#Ngrx #NgrxDevTools
#Entities #BestPractices
#Akita
#Nx - Micro-architecture
#xplat

#TypeScript #Cheatsheet
#SASS #BEM & SMACSS
#MVCSS #Material #Bootstrap

#Babel #ESLint #TSLint
#Karma #Mocha #Chai #Sinon
#Jest #Protractor
#Webpack

Q&A

THANK YOU

@CuongTruong

cuongtruong.official@gmail.com