Frontend-Entwicklung mit Angular

Mail: jonas.bandi@ivorycode.com

Twitter: @jbandi

ABOUT ME

Jonas Bandi
jonas.bandi@ivorycode.com
Twitter: @jbandi

- Freelancer, in den letzten 4 Jahren vor allem in Projekten im Spannungsfeld zwischen modernen Webentwicklung und traditionellen Geschäftsanwendungen.
- Dozent an der Berner Fachhochschule seit 2007
- In-House Kurse: Web-Technologien im Enterprise
  UBS, Postfinance, Mobiliar, BIT, SBB ...

JavaScript / Angular / React
Schulungen & Coachings,
Project-Setup & Proof-of-Concept:

http://ivorycode.com/#schulung

AGENDA

DAY 1

- Intro
- Angular CLI
- Angular Components
- Angular Routing
- Basic Constructs
- ToDo App

DAY 2

- Angular Forms
- UI Constructs Part 2
- Backend Access
- Modularization & Routing Part 2
- State Management

SIDE-TRACK

- Modern JavaScript Development
- JavaScript: ES5 & ES2015+
- TypeScript

Material

Git Repository:
`https://github.com/jbandi/ng-sbb-2018-1`

Initial Checkout:
```
git clone https://github.com/jbandi/ng-sbb-2018-1
```

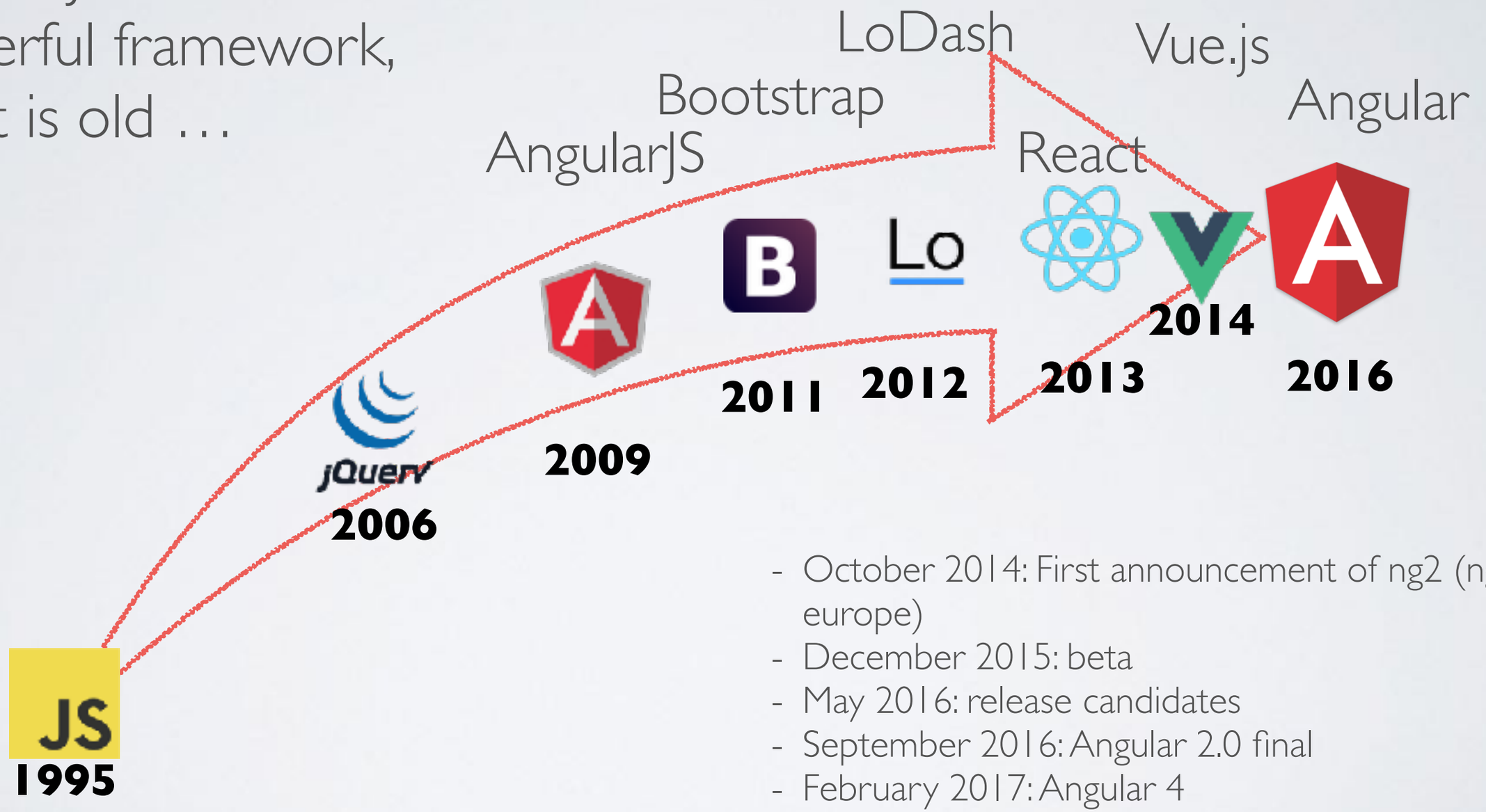
Update:
```
git pull
```

```
git reset --hard
git pull
```
(setzt alle lokalen Veränderungen zurück)

Slides & Exercises: `<checkout>/00-CourseMaterial`

The History of Angular

AngularJS is a
powerful framework,
but it is old …



- October 2014: First announcement of ng2 (ng-europe)
- December 2015: beta
- May 2016: release candidates
- September 2016: Angular 2.0 final
- February 2017: Angular 4
- November 2017: Angular 5
- March 2018: Angular 6

https://github.com/angular/angular/blob/master/docs/RELEASE_SCHEDULE.md


Angular is a new implementation of the concepts behind AngularJS ...
... for the modern web.
... but Angular is not an update to AngularJS.

Angular is **built upon** the modern web:

- shadow dom
- web workers

Angular is **built for** the modern web:

- mobile browsers
- modern browsers
- server-side rendering

Angular **improves** over AngularJS:

- faster
- easier to use & learn
- built on proven best practices (i.e. ui-components, unidirectional data flow …)

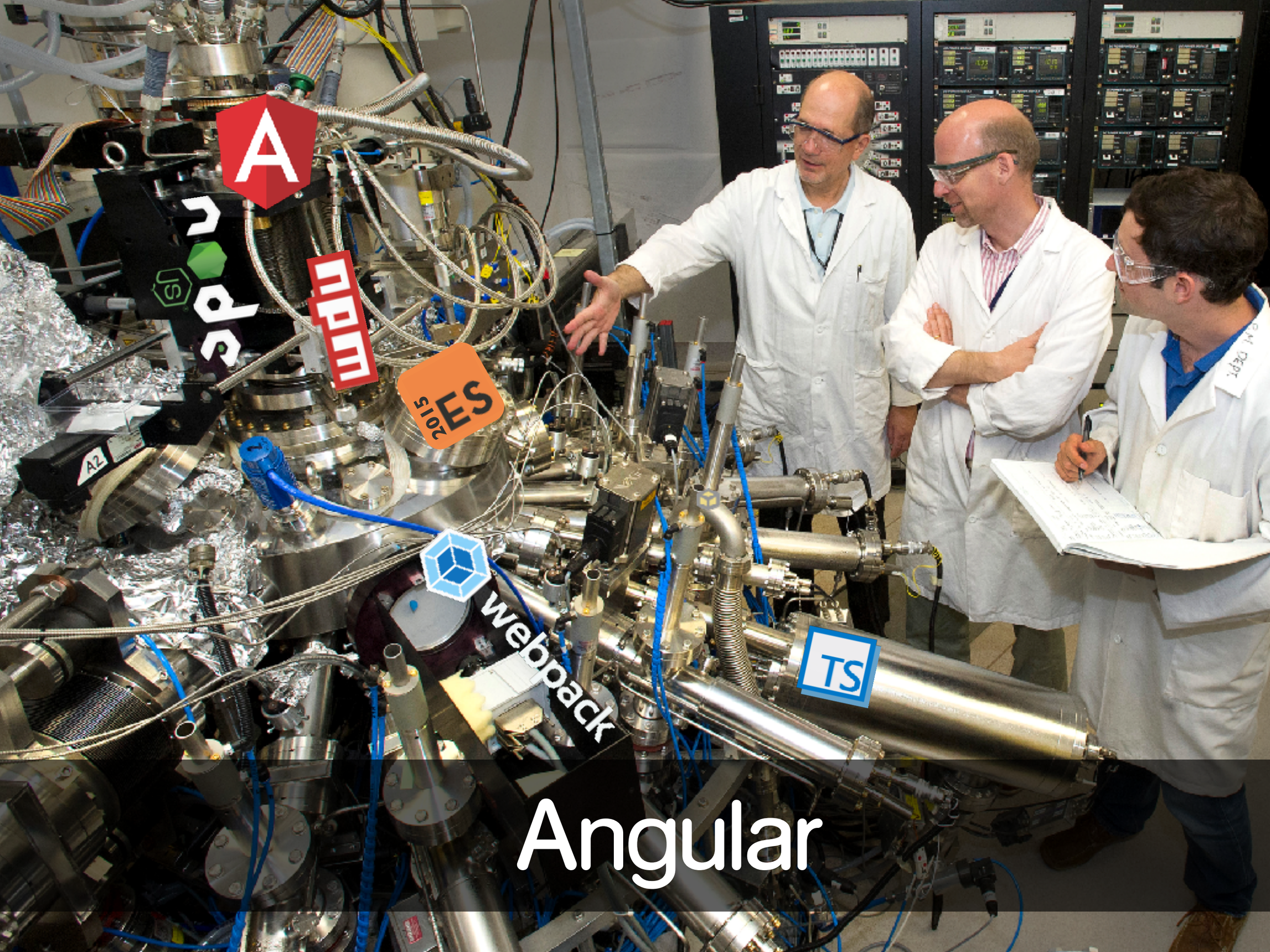
Angular JS

AngularJS

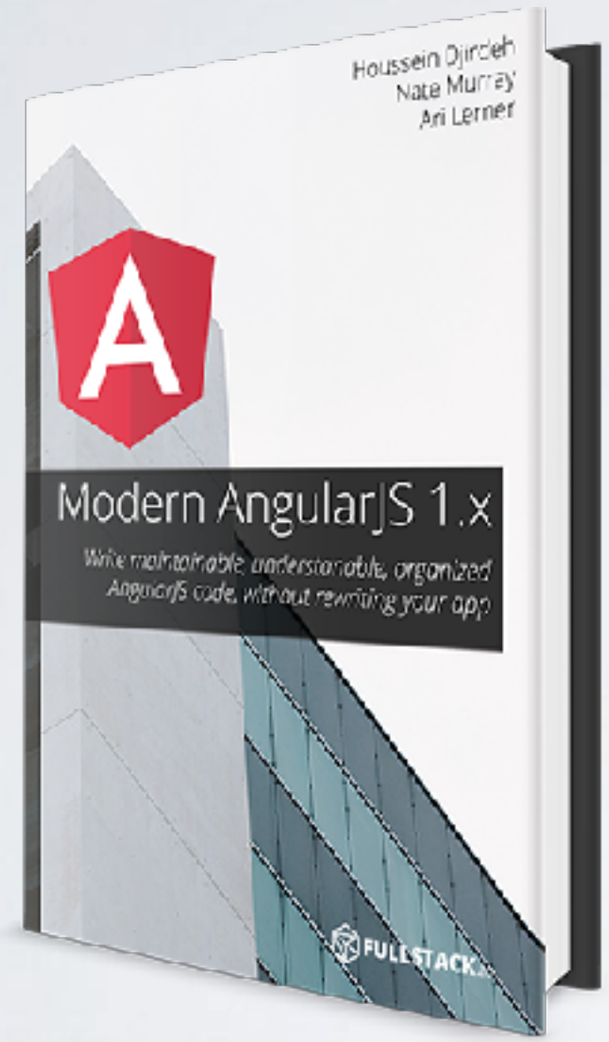
```
(function () {
    'use strict';

    var app = angular.module('todoApp');
    app.controller('todoController', ToDoController);

    ToDoController.$inject = ['toDoService'];
    function ToDoController(todoService) {
        var ctrl = this;

        ctrl.newToDo = new ToDoItem();
        ctrl.todos = todoService.getToDos();

        ctrl.addToDo = function () {
            ctrl.newToDo.created = new Date();
            todoService.addToDo(ctrl.newToDo);
            ctrl.newToDo = new ToDoItem();
        };

        ctrl.removeToDo = function (todo) {
            todoService.removeToDo(todo);
        };
    }
})();
```

Angular

```
import { Component, Output,
    EventEmitter } from '@angular/core';
import {ToDo} from "../../model/todo.model";

@Component({
    selector: 'td-new-todo',
    templateUrl: './new-todo.component.html'
})
export class NewTodo {

    constructor(){
        this.newToDo = new ToDo();
    }

    @Output() onAddToDo = new EventEmitter();

    addToDo() {
        this.onAddToDo.emit(this.newToDo);
        this.newToDo = new ToDo();
    }
}
```


Angular
webpack
TS
ES
2015

Sidenote: Modern AngularJS

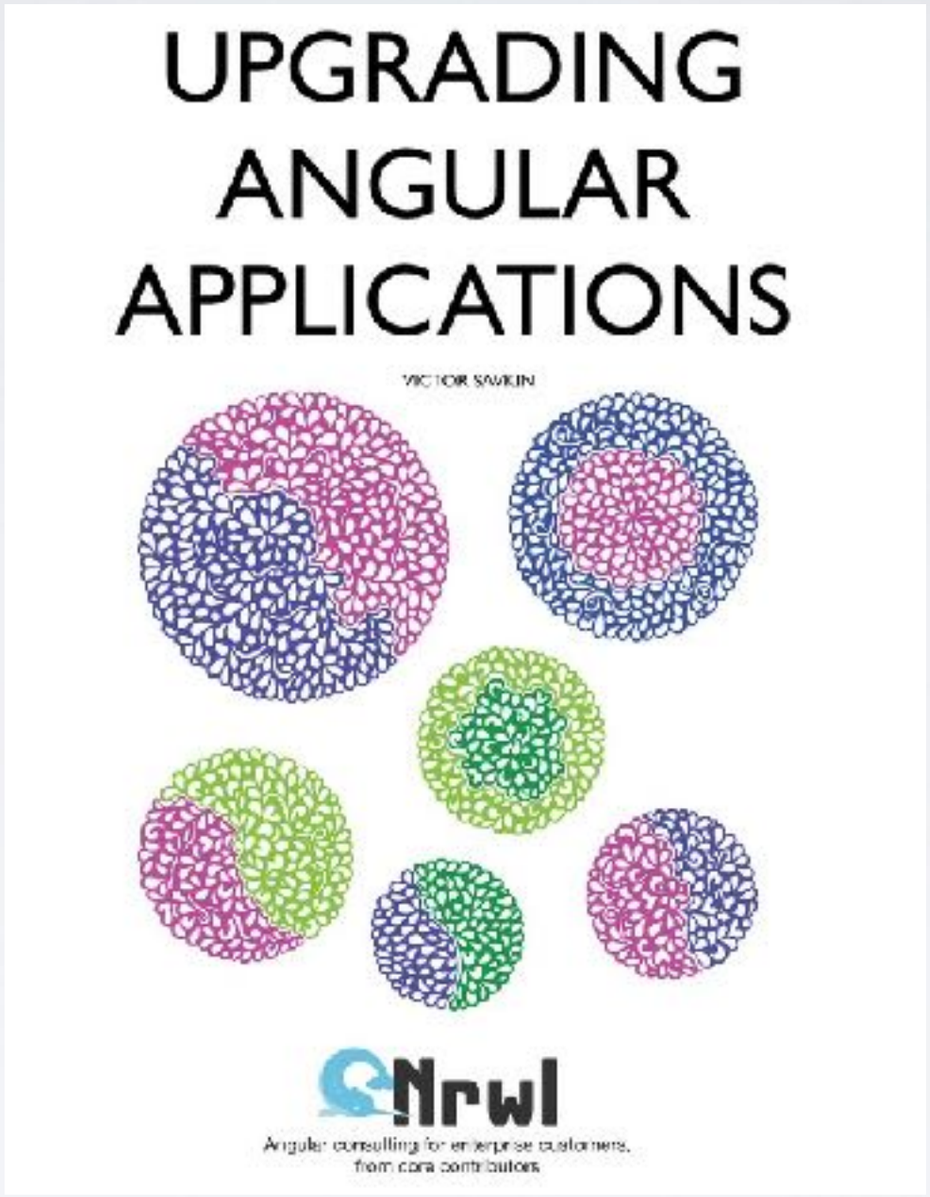


https://www.angular2patterns.com
https://github.com/simpulton/eggly-es6

https://www.ng-book.com/modern-ng1/

https://github.com/toddmotto/angular-styleguide

Sidenote: Migration



https://leanpub.com/ngupgrade

Angular aspires to be a platform

classic web-apps for desktops

WebPlatform.org

progressive web-apps for mobile (web workers, cache, push, offline)

installed mobile apps (hybrid)

installed mobile apps (native integrations)

server side rendering
https://universal.angular.io/

dev tooling
https://cli.angular.io/

ELECTRON

installed desktop apps

https://github.com/NathanWalker/angular2-seed-advanced

What is Angular?

- Angular is a framework for dynamic web applications (aka. Single Page Applications)
- A framework for dynamically manipulating the DOM and linking it with application functionality.
- Angular is a HTML processor
- Angular lets you extend HTML

Language Choices

no compilation

compilation

Dropped in Angular 5

weakly typed

(optional) strongly typed

Angular 2: Why TypeScript? - https://vsavkin.com/writing-angular-2-in-typescript-1fa77c78d8e8

Getting started quickly with Angular CLI

Execute the following commands in a terminal:

```
npm install -g @angular/cli

ng new awesome-app
cd awesome-app
npm start
```

https://github.com/angular/angular-cli

EXERCISES

Exercise 1 – Create an Angular App

TODO: Build

https://medium.com/the-node-js-collection/modern-javascript-explained-for-dinosaurs-f695e9747b70

Dissecting an Angular App

```
npm start
npm run build -- --prod -vc
npm test
npm e2e
```

Project Configuration:

- package.json
- tsconfig.json
- .angular-cli.json
- polyfills.ts

App Component

- /src/app

Bootstrapping

- main.ts
- index.html

Angular CLI Project Setup

ES2015+/TypeScript

Write Code

TypeScript

Compile

TypeScript is compiled to ES5

webpack
MODULE BUNDLER

Bundle

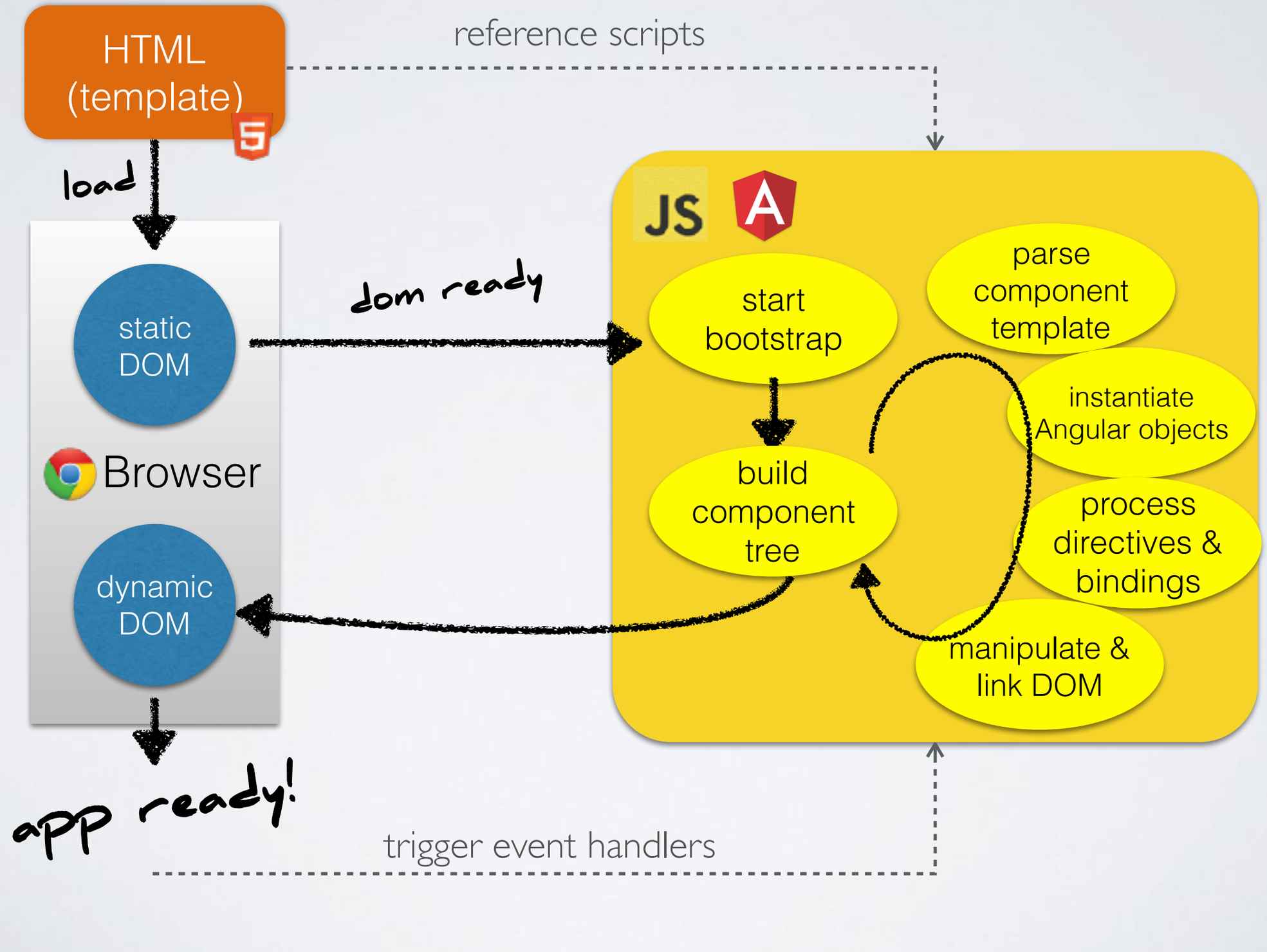
Create coarse-grained & optimised bundles from many files and 3rd-party libraries

ES5

Load pure JavaScript bundles.

Run in Browser

How Angular Works



How Angular Works

"Angular is a HTML processor"

template

```
<ul class="todo-list" >
    <li *ngFor="let todo of todos">
        {{todo.title}}
        <button class="remove-button"
            (click)="removeToDo(todo)">
            x
        </button>
    </li>
</ul>
```

parse

generate

components
services
...

interact

JS

manipulate

handle events

DOM

Bootstrapping

src/index.html

html shell

```
<body>

  <app-root>
    Loading...
  </app-root>
  <script src=…>
  </script>
```

src/main.ts

bootstrap-script

```
platformBrowserDynamic()
  .bootstrapModule(AppModule)
```

root module

```
@NgModule({
    imports: [BrowserModule],
    declarations: [AppComponent],
    bootstrap: [AppComponent]
})
export class AppModule {}
```

src/app/app.component.ts

root component

```
@Component({
    selector: 'app-root',
    template: `Hello world`
})
export class AppComponent {}
```

src/app/app.module.ts

My concerns with Angular CLI

I recommend using the Angular CLI, but ...

- Underlying tooling is "hidden", but it is a leaky abstraction (npm, webpack, typescript …)
- It's intransparent which steps/optimizations are performed and which tools are used.
- Documentation is sparse …

=> `ng eject` can be a solution

- It's not an official part of Angular & Google internally is not using the CLI, CLI might not be on the future path.
- The CLI is notoriously buggy:
  https://github.com/angular/angular-cli/issues/6583
  https://github.com/angular/angular-cli/issues/8359

https://medium.com/@Jakeherringbone/what-angular-is-doing-with-bazel-and-closure-21f526f64a34

Other Starting Points

Official Quickstart: https://github.com/angular/quickstart	using SystemJS, no production workflow
Angular Seed, Webpack: https://github.com/angular/angular2-seed	outdated
Webpack Starter: https://github.com/AngularClass/angular2-webpack-starter	very complex
Angular 2 seed: https://github.com/mgechev/angular2-seed	very complex
dotnet core templates https://github.com/aspnet/JavaScriptServices	good setup for .NET
Nrwl Extensions for Angular https://nrwl.io/nx	Based on Angular CLI

Tooling

Debugging in Chrome:

https://augury.angular.io/

Webstrom & IntelliJ:
Angular 2 TypeScript Live Templates
https://github.com/MrZaYaC/ng2-webstorm-snippets
https://plugins.jetbrains.com/plugin/8395-angular-2-typescript-live-templates

Visual Studio Code:
Angular Essentials
https://github.com/MrZaYaC/ng2-webstorm-snippets

Angular Components

Angular Components

Components are the main building-block of Angular.

Template + Class + Metadata

= Component

A Simple Component

```
import {Component} from 'angular2/core';

@Component({
    selector: 'my-app',
    template: '<h1>My First Angular 2 App</h1>'
})
export class AppComponent { }
```

New components can be generated with the Angular CLI:

```
ng generate component hello
```

Directives & Components

A component is a special kind of directive.
A directive is a construct, that is embedded into html and has a special meaning for the framework.

Databinding

Interpolation

{{value}}

Property Binding

[property]="value"

Event Binding

(event)="handler"

Two Way Binding

[(ngModel)]="value"

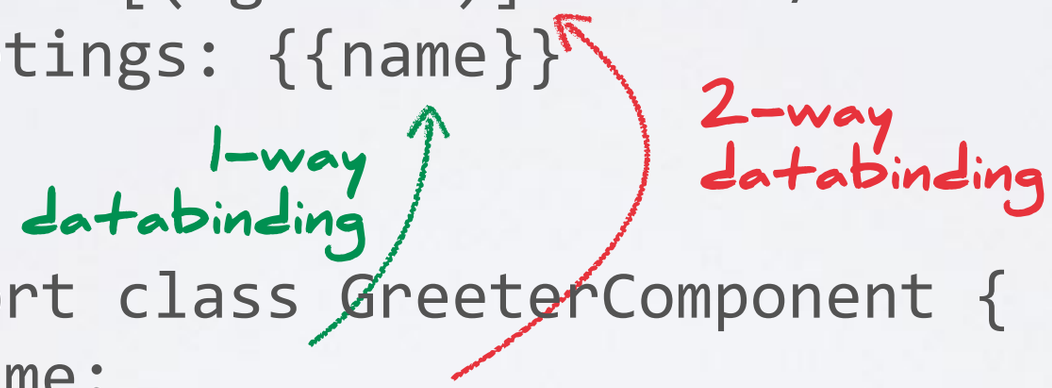
(ngModel is provided by the FormsModule)

DOM
(Template)

component

Write Your First Component

```
import {Component} from '@angular/core';

@Component({
    selector: 'greeter',
    template: `
<input [(ngModel)]="name"/>
Greetings: {{name}}
`
})
export class GreeterComponent {
  name;
  constructor() {
      this.name = 'John';
    }
}
```



EXERCISES

Exercise 2 – Write your first component

Special Element Bindings
(examples)

CSS class binding (DOM: classList)

```
<p [className]="myClass"></p>
```

```
<p [class.is-active]="isActive()"></p>
```

```
<div [ngClass]="['bold-text', 'green']">array of classes</div>
<div [ngClass]="'italic-text blue'">string of classes</div>
<div [ngClass]="{'small-text': true, 'red': true}”>
   object of classes
</div>
```

CSS style binding (DOM: style property)

```
<p [style.display]="!isActive() ? 'none' : null"></p>
```

```
<div [ngStyle]="{'color': color, 'font-size': size>}”
style using ngStyle
</div>
```

Attribute binding (DOM: setAttribute)

```
<p [attr.role]="role"></p>
```

Special Event Bindings

Angular provides "pseudo-events":

```
<input type="text" (keyup.enter)="handleEnter($event)">
```

```
<input type="text" (keyup.shift.enter)="handleShiftEnter($event)">
```

Unfortunately the supported pseudo events are not documented …?

https://angular.io/guide/user-input#key-event-filtering-with-keyenter

Testing Components

Tests are run via Karma: **`npm run test`**

For unit tests you want to keep the "code under test" small. Angular provides a **`TestBed`** to assemble a module which can only contain the minimal code amount.

Drawback: The **`TestBed`** must be configured for each test.

When working with external templates, the assembling of the **`TestBed`** must be asynchronous.

The **`TestBed`** can create component fixtures which provide a reference to the component object and the associated DOM element.

https://angular.io/guide/testing

Client Side Routing

Client Side Routing in a SPA

The traditional web is built on the concept of linked documents (URL, bookmarking, back-button …)

In a SPA the document is just the shell for an application. When you navigate away from the document, the application is "stopped".

As a consequence a SPA should "emulate" the traditional user-experience on the web:

- navigate via urls, links & back-button
- bookmarks and deep links

Client-Side Routing in a SPA

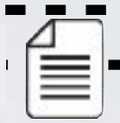
http://thedomain:80/app#/first

Browser

Server

initial request
http://thedomain:80/app

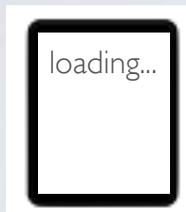
html-shell



assets

optional ajax request
http://thedomain:80/api/data/first

data



loading...

html-shell is loaded

"navigate" to "first" screen

load data &
render screen with JS

http://thedomain:80/app#/**first**

navigation

"navigate" to "second" screen

load data &
render screen with JS

http://thedomain:80/app#/**second**

optional ajax request
http://thedomain:80/api/data/second

data

bootstrapping app

Routing in Angular

Angular provides the **`@angular/router`** module, which helps to implement complicated routing scenarios.

With the Angular router module you can map paths to components.

https://angular.io/guide/router

Generating an app with routing with the Angular CLI

```
ng new fantastic-ng --routing
```

In `src/app/app-routing.module.ts` routes can be configured:

```
const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: 'first'  },
  { path: 'first', component: FirstComponent  },
  { path: 'second', component: SecondComponent  }
];
```

The router-module provides the `<router-outlet>` directive which is used in `app.component.html`

http://localhost:8080/app

<head>
</head>

<body>

App Component

...

<router-outlet>

...

</body>

SPA Shell

/app/path1

compontent 1

/app/path2

component 2

Router Directives

The router provides directives for placing a "routed" component into a template.

Directives: `router-outlet`, `routerLink`, `routerLinkActive`

```
<nav>
  <a routerLink="/databinding" routerLinkActive="active">Databinding</a>
  <a routerLink="/pipes" routerLinkActive="active">Pipes</a>
</nav>

<router-outlet></router-outlet>
```

Routing

Angular provides the **`@angular/router`** package for client-side routing.

Routing is configured statically when loading a module.
It is good practice to configure routing in a separate module, which is imported by the app module.

```
const routes: Routes = [
  { path: '', component: OverviewComponent },
  { path: 'done', component: DoneTodosComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

```
@NgModule({
  declarations: [ ... ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

EXERCISES

Exercise 3 - Create an app with routing

Structural Directives

Structural directives shape or reshape the DOM's structure.

The host element of the directive is used as a template that is instantiated by Angular.

The three common, built-in structural directives:
`*ngFor, *ngIf, *ngSwitchCase`

```
<ul>
  <li *ngFor="let character of characters">
    <span>{{character.firstName}}</span>
    <span>{{character.lastName}}</span>
    <span *ngIf="showDistrict">{{character.district}}</span>
    <span [ngSwitch]="character.emotion">
      <span *ngSwitchCase="Emotion.InLove">❤️</span>
      <span *ngSwitchCase="Emotion.Angry">🔪</span>
      <span *ngSwitchCase="Emotion.Sad">💧</span>
      <span *ngSwitchDefault>⁉️</span>
    </span>
  </li>
</ul>
```

https://angular.io/guide/structural-directives

Structural Directives

The `*` is part of the name of the structural directive.

Structural directives are a *mycrosyntax* that expands into **`<ng-template>`** elements.

```
<ul>
    <ng-template ngFor let-character [ngForOf]="characters">
        <li>
            <span>{{character.firstName}}</span>
            <span>{{character.lastName}}</span>
            <ng-template [ngIf]="showDistrict">
                <span>{{character.district}}</span>
            </ng-template>
        </li>
    </ng-template>
</ul>
```

`<ng-template>` and **`<ng-content>`** are Angular elements used to render html. They can also be used directly.

https://angular.io/guide/structural-directives

*ngIf & else

```
<span *ngIf="character.isInLove; else elseBlock">❤️</span>
<ng-template #elseBlock>⁉️</ng-template>
```

```
<span *ngIf="character.isInLove; then thenBlock; else elseBlock"></span>
<ng-template #thenBlock>❤️</ng-template>
<ng-template #elseBlock>⁉️</ng-template>
```

https://angular.io/api/common/NgIf

*ngFor & trackBy

***ngFor** re-creates the DOM elements when the bound objects change.

With a **trackBy** function Angular can reuse DOM elements.

```
<ul>
  <li *ngFor="let character of characters; trackBy:trackByFirstName">
    <span>{{character.firstName}}</span>
    <span>{{character.lastName}}</span>
    <span *ngIf="showDistrict">{{character.district}}</span>
  </li>
</ul>
```

```
@Component({ ... })
export class StructuralDirectivesComponent {

  ...

  trackByFirstName(index: number, character: ICharacter) {
    return character.firstName;
  }
}
```

https://angular.io/guide/template-syntax#ngfor-with-trackby

EXERCISES

Exercise 4.1 – ToDo App

Component Lifecycle Hooks

Angular calls specific methods on a component during its life-cycle:

`ngOnChanges`	before **`ngOnInit`** and when a data-bound input property value changes.
`ngOnInit`	after the first **`ngOnChanges`**.
`ngDoCheck`	during every Angular change detection cycle.
`ngAfterContentInit`	after projecting content into the component.
`ngAfterContentChecked`	after every check of projected component
`ngAfterViewInit`	after initializing the component's views and child views
`ngAfterViewChecked`	after every check of the component's views and
`ngOnDestroy`	just before Angular destroys the directive/

Angular offers matching interfaces: **`OnInit, AfterViewInit`** …
They are optional, just for tooling at build time.

https://angular.io/guide/lifecycle-hooks

Services

Services are "Injectables"

"Service" is a broad category encompassing any value, function or feature that our application needs.

A service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well.

It typically provides data and/or logic for other Angular constructs.

```
@Injectable()
export class DataService {
  constructor(private _otherService: OtherService)
  ...
}
```

```
export class OtherService {
  ...
}
```

`@Injectable()` is only needed if the server itself has dependencies.
However it is a good practice to add it to every service as default.

Dependency Injection

With dependency injection we can provide instances of classes to other Angular constructs.

Consumers declare dependencies.

Dependencies can have dependencies.

```
export class DataService {
  ...
}
```

```
@Component({
    selector: 'my-component',
    template: 'path/template.html'),
    providers: [DataService],
})
export class MyComponent {
  constructor(private dataService: DataService){}
  ...
}
```

Typically injection relies on typescript type metadata.
Configure `tsc` with `emitDecoratorMetadata:true`
Alternative: `constructor(@Inject(DataService) _dataService)`

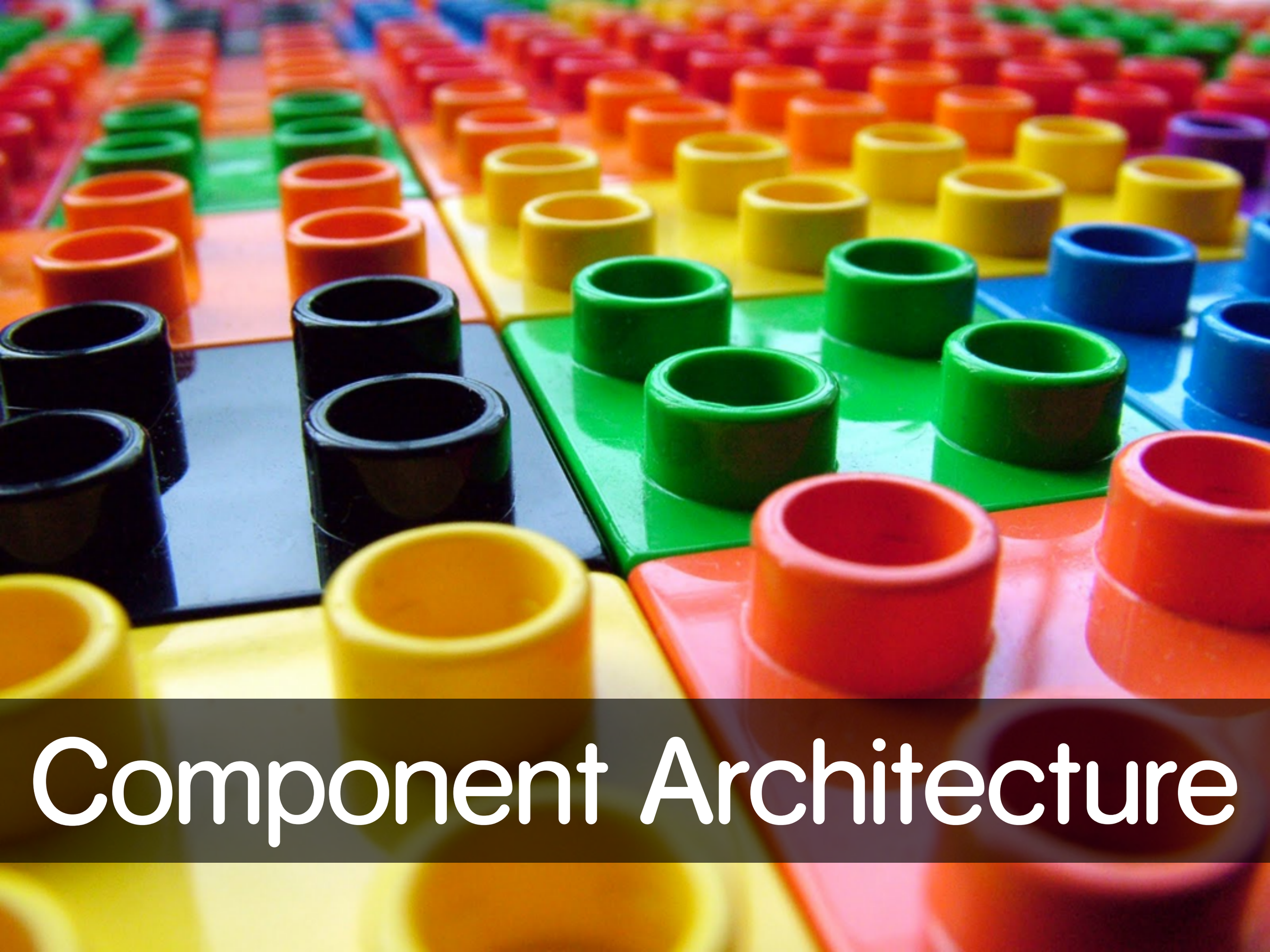
Dependency Injection

Dependencies can have dependencies.

```
export class FirstService {
  ...
}
```

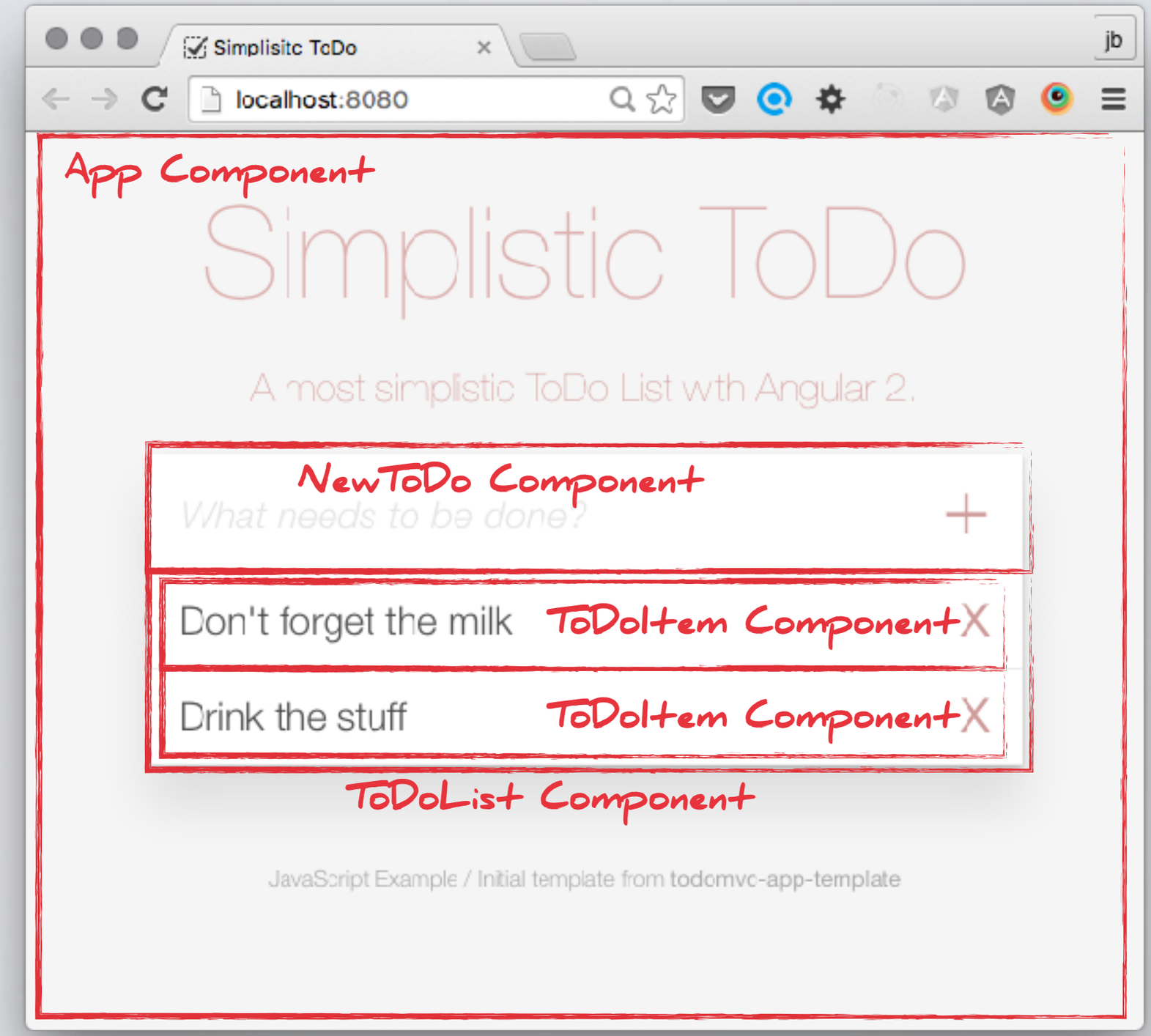
```
@Injectable()
export class SecondService {
  constructor(private firstService: FirstService){}
}
```

```
@Component({
    selector: 'my-component',
    template: 'path/template.html'),
    providers: [SecondService],
})
export class MyComponent {
  constructor(private secondService: SecondService){}
  ...
}
```



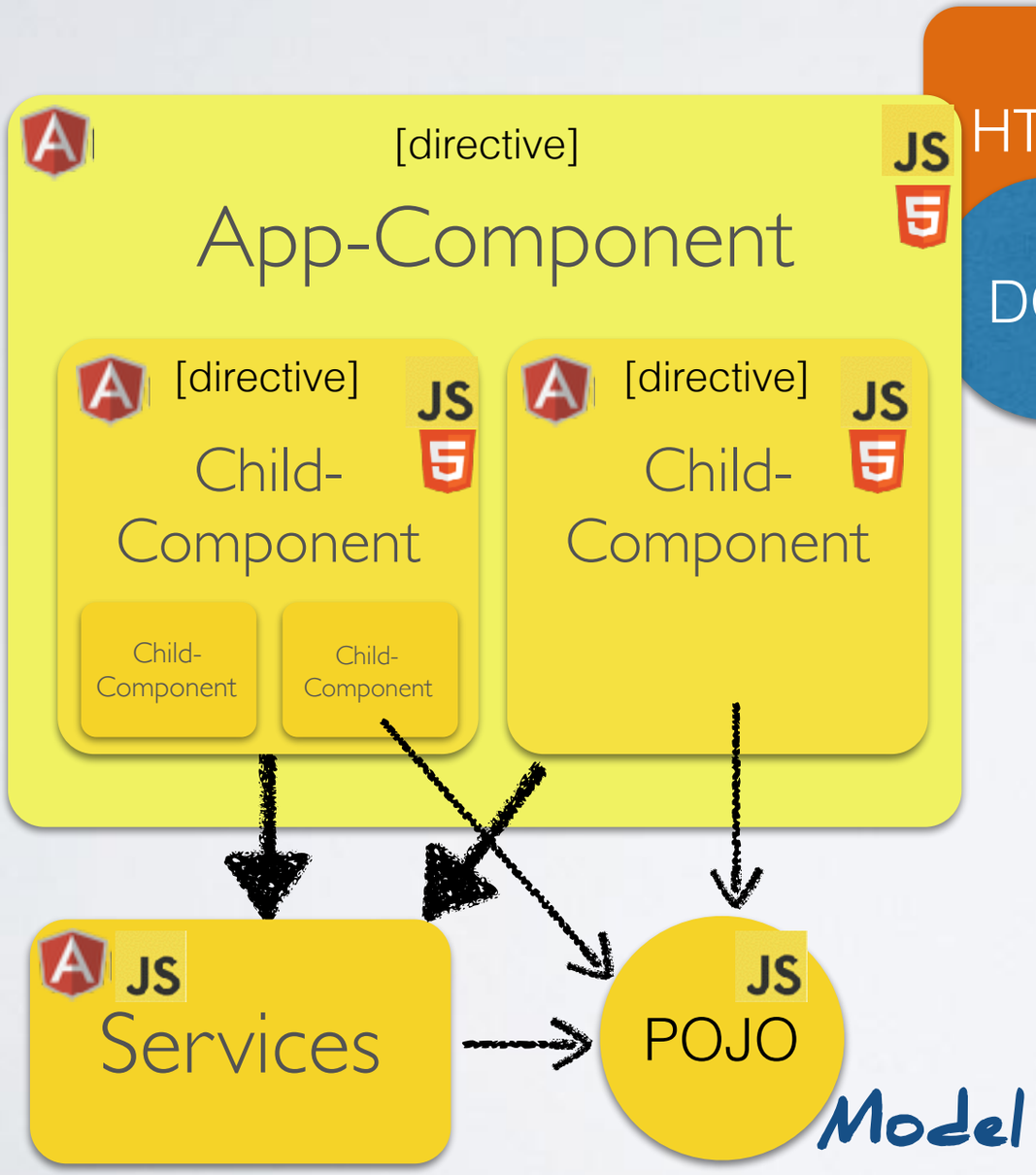
Component Architecture

Components: The Main Building Blocks



The Component Architecture

An app consists of a tree of components



Components are Angular directives, that are processed when compiling the html

Each component consists of a template and a class.

A component can be composed from other components.

similar approaches: React, VueJS, Web-Components/Polymer

Components vs. MVC

Components only control their own View and Data

Components have a well-defined public API => Inputs and Outputs

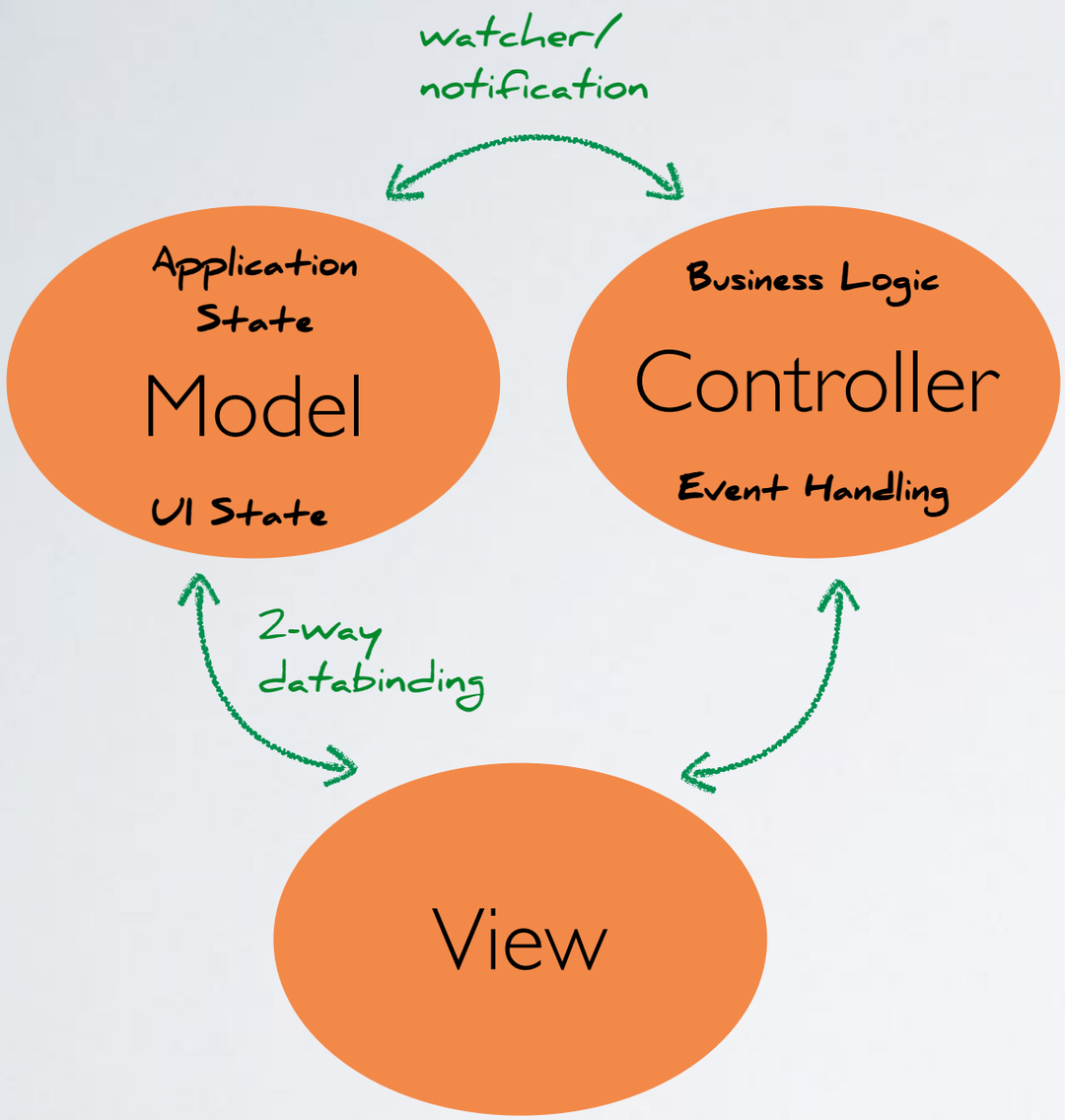
Components have a well-defined lifecycle

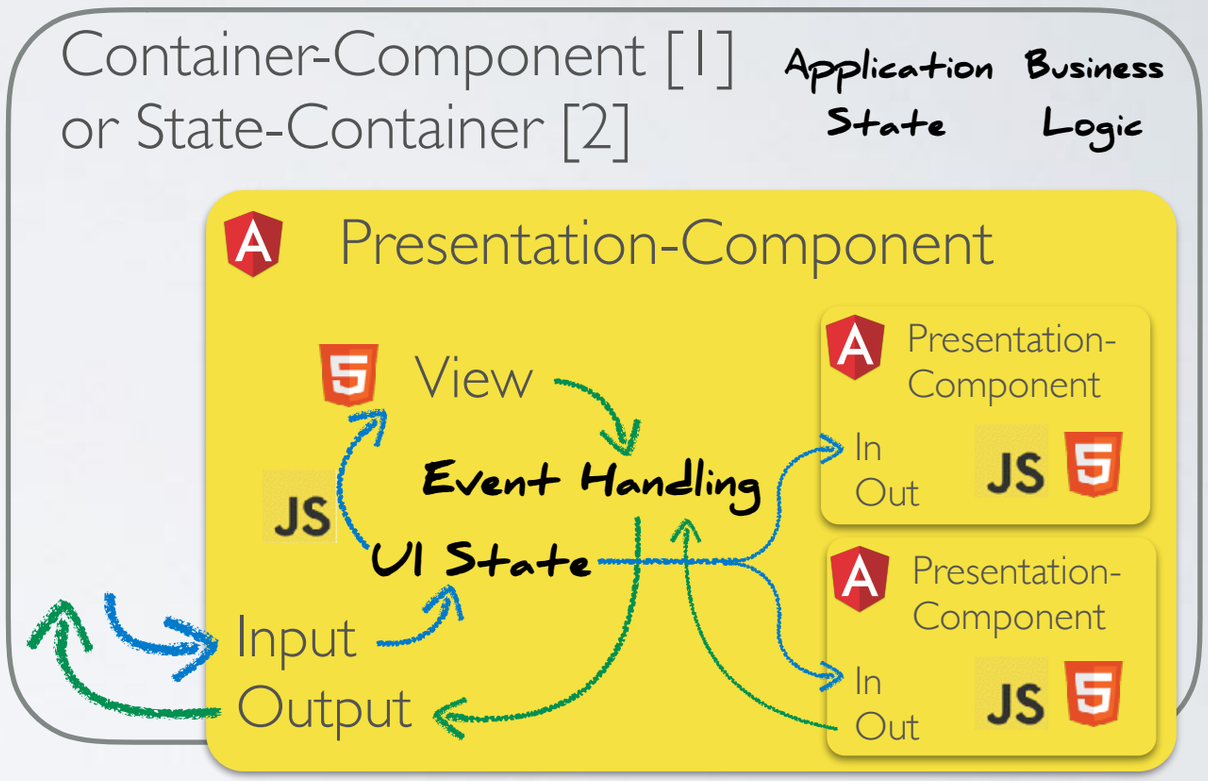
An application is a tree of components

https://docs.angularjs.org/guide/component

Sidenote:
A component architecture can be realised with AngularJS (1.5+):
https://www.gitbook.com/book/onehungrymind/angular2-patterns-in-angular1
https://github.com/toddmotto/angular-styleguide

MVC vs. Components





[1] aka: Smart-Component, Screen …

[2] Redux, ngrx…

https://medium.freecodecamp.com/is-mvc-dead-for-the-frontend-35b4d1fe39ec#.53qljfmaw
https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0#.dao1ozirh

Nested Components

A parent component can use child components in its template:

```
<div>
    <aw-child-list></aw-child-list>
</div>
```

To use a child component in a template, the child component must be declared or imported in the corresponding NgModule:

```
@NgModule({
  declarations: [AppComponent, ParentComponent, ChildListComponent],
  imports     : [BrowserModule, FormsModule, HttpModule],
  bootstrap   : [AppComponent]
})
export class AppModule {}
```

Using a Component as a Directive

index.html

```
<html>
<head> … </head>
<body>
  <my-app>Loading...</my-app>
</body>

</html>
```

parent component

```
@Component({
  selector: 'my-app',
  templateUrl: 'app.html',
})
export class AppComponent { }
```

```
<div>
  <h1>App</h1>
  <child-component></child-component>
</div>
```

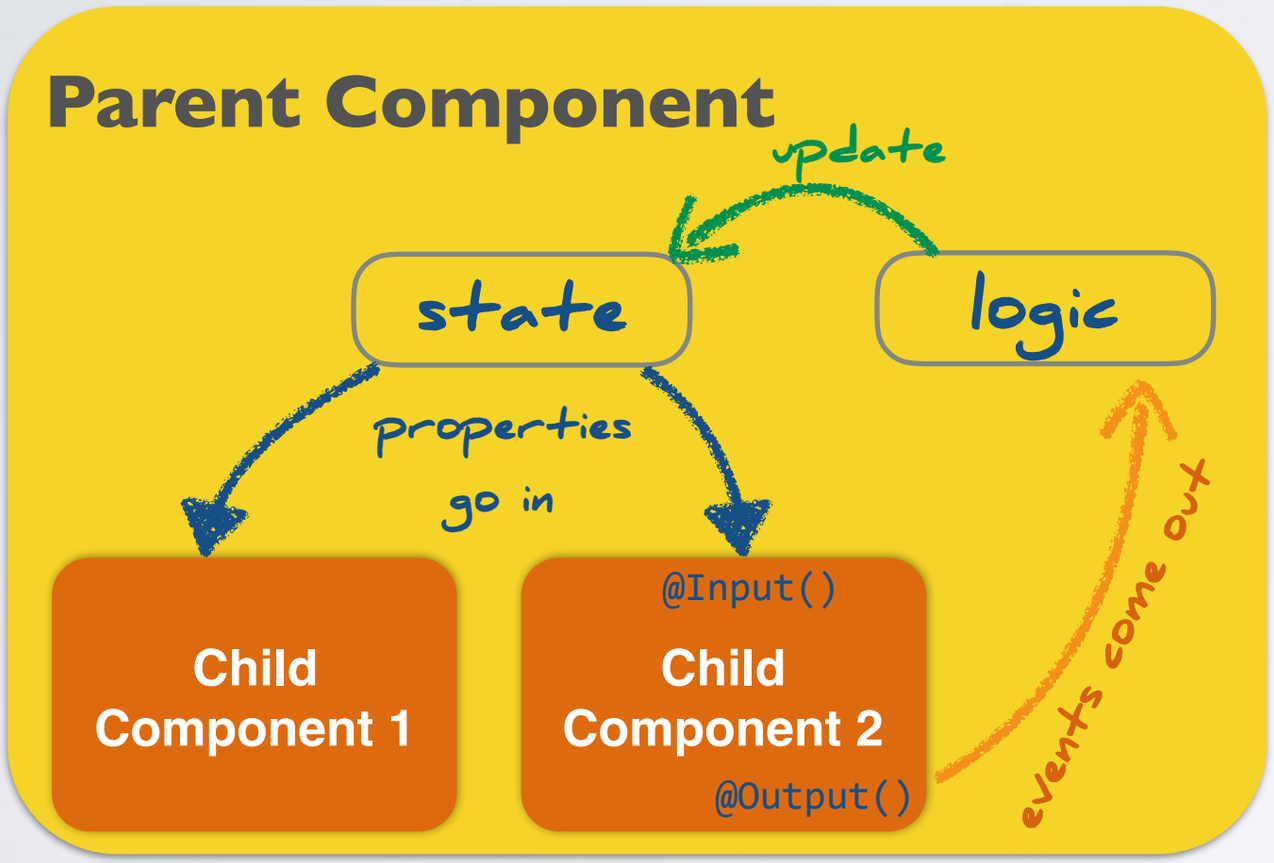
```
@NgModule({
  declarations: [AppComponent,
                 ChildComponent],
  imports     : [BrowserModule],
  bootstrap   : [AppComponent]
})
export class AppModule {}
```

```
@Component({
  selector: 'child-component',
  templateUrl: 'child.html',
})
export class ChildComponent { }
```

child component

Nested Components: Data-Flow

State should be explicitly owned by a component.



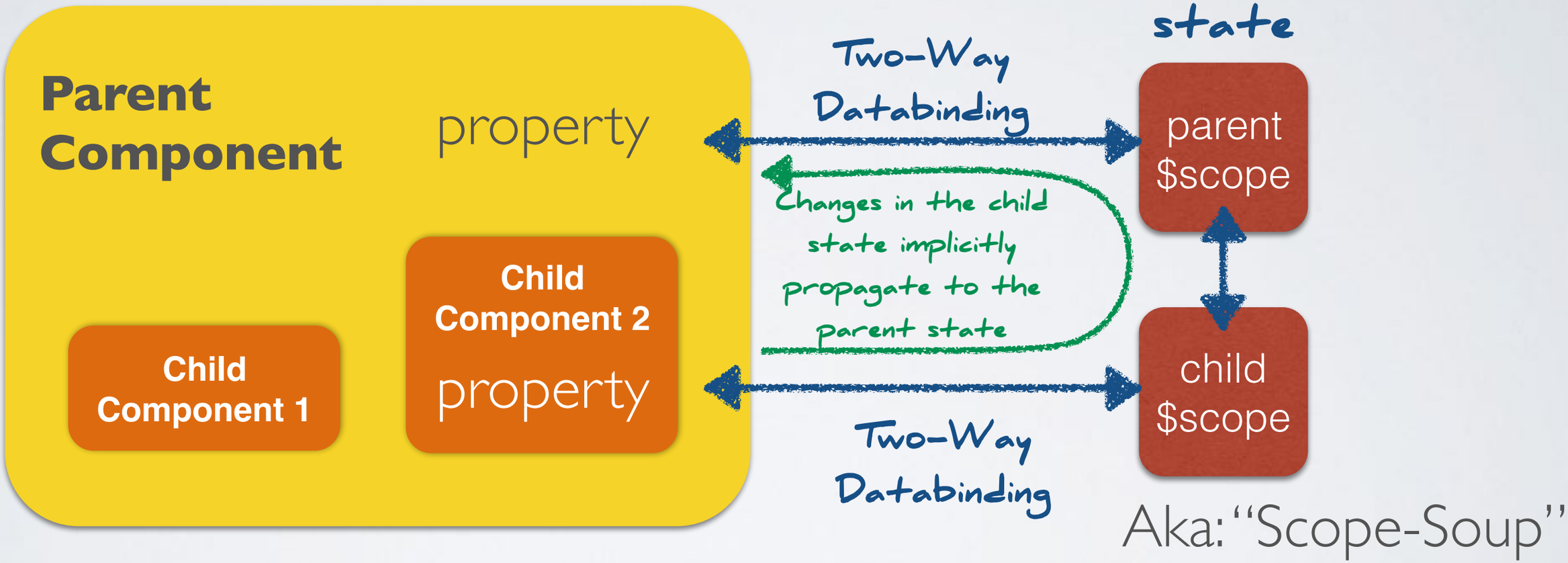
- A parent component passes state to children
- Children should not edit state of their parent
- Children "notify" parents (events, actions …)

Angular formalises **unidirectional data-flow** with `@Input()` and `@Output()` properties.

https://www.robinwieruch.de/learn-react-before-using-redux/

Compared to Two-Way Databinding

AngularJS **`$scope`** and two-way-databinding can be confusing in regard to state-management.



With two-way data-binding it's not clear how the data flows because it can flow in all directions (including from child components to parents) - this makes it hard to understand the app and understand of impact of model changes in one part of the app on another (seemingly unrelated) part of it.

Angular Weekly Meeting Notes, July 7 2014, http://bit.ly/2tbC2Jh

Input- & Output-Properties

Angular property- and event-binding to DOM elements:

template.html

```
<p [innerText]="name" [style.color]="color"></p>
<input type="text" (change)="onChange($event)">
```

Input- and Output properties apply the same concept to application-specific data:

parent-template.html

```
<my-child [children]="persons"
    (onAddPerson)="addPerson($event)">
</my-child>
```

parent-component.ts

```
@Component({ ... })
export class ParentComponent {
  private persons = [];
  addPerson(person){ ... }
}
```

child-template.html

```
<button (click)="addPerson()">Add</button>
<ul>
    <li *ngFor="#child of children">
        ...
    </li>
</ul>
```

child-component.ts

```
@Component({ ... })
export class ChildComponent {
  @Input() children;
  @Output() onAddPerson = new EventEmitter();
  addPerson(){onAddPerson.emit(...)}
}
```

EXERCISES

Exercise 4.2 – ToDo App