Angular 2 Development for Java developers

Yakov Fain, Farata Systems

@yfain

Angular 2

- Complete re-write of AngularJS ⟵ Google
- Component-based
- Better performance: http://www.roblog.io/js-repaint-perfs
- UI rendering is implemented in a separate layer
- Reactive extensions **RxJS**
- Can write apps in **TypeScript**,
  Dart, or JavaScript (ES5 or ES6)

https://github.com/microsoft/typescript/issues/6508

Getting to know TypeScript

http://www.typescriptlang.org

What's TypeScript?

- Designed by Anders Hejlsberg, the creator of C#, Delphi, and Turbo Pascal
- A superset of JavaScript
- A lot more toolable than JavaScript
- Easy to learn for Java developers
- Supports types, classes, interfaces, generics, annotations
- Great IDE support

Transpiling TypeScript Interactively

http://www.typescriptlang.org/play/

TypeScript → JavaScript (ES5)

Select...

TypeScript | Share | Run | JavaScript

```
1 class Bar{}
2
3 let bar = new Bar();
4
5 bar.doSomething();
6
7
```

```
1 var Bar = (function () {
2     function Bar() {
3     }
4     return Bar;
5 }());
6 var bar = new Bar();
7 bar.doSomething();
8
```

Compile-time error

Run-time error

Classes

TypeScript → JavaScript (ES5)

```typescript
class Person {
    firstName: string;
    lastName: string;
    age: number;
    ssn: string;
}

var p = new Person();

p.firstName = "John";
p.lastName = "Smith";
p.age = 29;
p.ssn = "123-90-4567";
```

```javascript
var Person = (function () {
    function Person() {
    }
    return Person;
})();
var p = new Person();
p.firstName = "John";
p.lastName = "Smith";
p.age = 29;
p.ssn = "123-90-4567";

```

A Class With Constructor

TypeScript → JavaScript (ES5)

TypeScript Select... **Share**

```typescript
class Person {

    constructor(public firstName: string,
        public lastName: string, public age: number,
         private _ssn: string) {}
}

var p = new Person("John", "Smith", 29, "123-90-4567");
console.log("Last name: " + p.lastName +
                " SSN: " + p._ssn);
```

Run **JavaScript**

```javascript
var Person = (function () {
    function Person(firstName, lastName, age, _ssn) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
        this._ssn = _ssn;
    }
    return Person;
}());
var p = new Person("John", "Smith", 29, "123-90-4567");
console.log("Last name: " + p.lastName +
    " SSN: " + p._ssn);

```

Inheritance

Classical syntax → Prototypal

TypeScript | Select... | Share

```typescript
class Person {

    constructor(public firstName: string,
        public lastName: string, public age: number,
        private _ssn: string) {
    }
}

class Employee extends Person{

}
```

Run | JavaScript

```javascript
var __extends = this.__extends || function (d, b) {
    for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p];
    function __() { this.constructor = d; }
    __.prototype = b.prototype;
    d.prototype = new __();
};
var Person = (function () {
    function Person(firstName, lastName, age, _ssn) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
        this._ssn = _ssn;
    }
    return Person;
})();
var Employee = (function (_super) {
    __extends(Employee, _super);
    function Employee() {
        _super.apply(this, arguments);
    }
    return Employee;
})(Person);
```

Generics

TypeScript | Select... | Share

```typescript
class Person {
    name: string;
}

class Employee extends Person{
    department: number;
}

class Animal {
    breed: string;
}

var workers: Array<Person> = [];

workers[0] = new Person();
workers[1] = new Employee();
workers[2] = new Animal();
```

Error

Run | JavaScript

```javascript
var __extends = (this && this.__extends) || function (d, b) {
    for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p];
    function __() { this.constructor = d; }
    d.prototype = b === null ? Object.create(b) : (__.prototy
};
var Person = (function () {
    function Person() {
    }
    return Person;
})();
var Employee = (function (_super) {
    __extends(Employee, _super);
    function Employee() {
        _super.apply(this, arguments);
    }
    return Employee;
})(Person);
var Animal = (function () {
    function Animal() {
    }
    return Animal;
})();
var workers = [];
workers[0] = new Person();
workers[1] = new Employee();
workers[2] = new Animal();

```

No Errors

Arrow Function Expressions (lambdas)

```
var getName = () => 'John Smith';

console.log(`The name is ` + getName());
```

TypeScript | Select... | Share

```
1 var getName = () => 'John Smith';
2 console.log(`The name is ` + getName());
```

Run | JavaScript

```
1 var getName = function () { return 'John Smith'; };
2 console.log("The name is " + getName());
3
```

Interfaces as Custom Types

TypeScript | Select... | Share | Run | JavaScript

```typescript
interface IPerson {

    firstName: string;
    lastName: string;
    age: number;
    ssn?: string;
}

class Person  {

    constructor(public config: IPerson) {
    }
}

var aPerson: IPerson = {
    firstName: "John",
    lastName: "Smith",
    age: 29
}

var p = new Person(aPerson);
console.log("Last name: " + p.config.lastName );
```

```javascript
var Person = (function () {
    function Person(config) {
        this.config = config;
    }
    return Person;
})();
var aPerson = {
    firstName: "John",
    lastName: "Smith",
    age: 29
};
var p = new Person(aPerson);
console.log("Last name: " + p.config.lastName);

```

Interfaces and `implements`

```typescript
interface IPayable{

  increasePay(percent: number): void
}

class Employee implements IPayable{

    increasePay(percent: number): void {
        // increase salary
    }
}

class Contractor implements IPayable{

    increasePay(percent: number): void {
        // increase hourly rate
    }
}

var workers: Array<IPayable> = [];
workers[0] = new Employee();
workers[1] = new Contractor();

workers.forEach(worker => worker.increasePay(30));
```

```javascript
var Employee = (function () {
    function Employee() {
    }
    Employee.prototype.increasePay = function (percent) {
        // increase salary
    };
    return Employee;
})();
var Contractor = (function () {
    function Contractor() {
    }
    Contractor.prototype.increasePay = function (percent) {
        // increase hourly rate
    };
    return Contractor;
})();
var workers = [];
workers[0] = new Employee();
workers[1] = new Contractor();
workers.forEach(function (worker) { return worker.increasePay(30); });

```

Back to Angular 2

What's a Component?

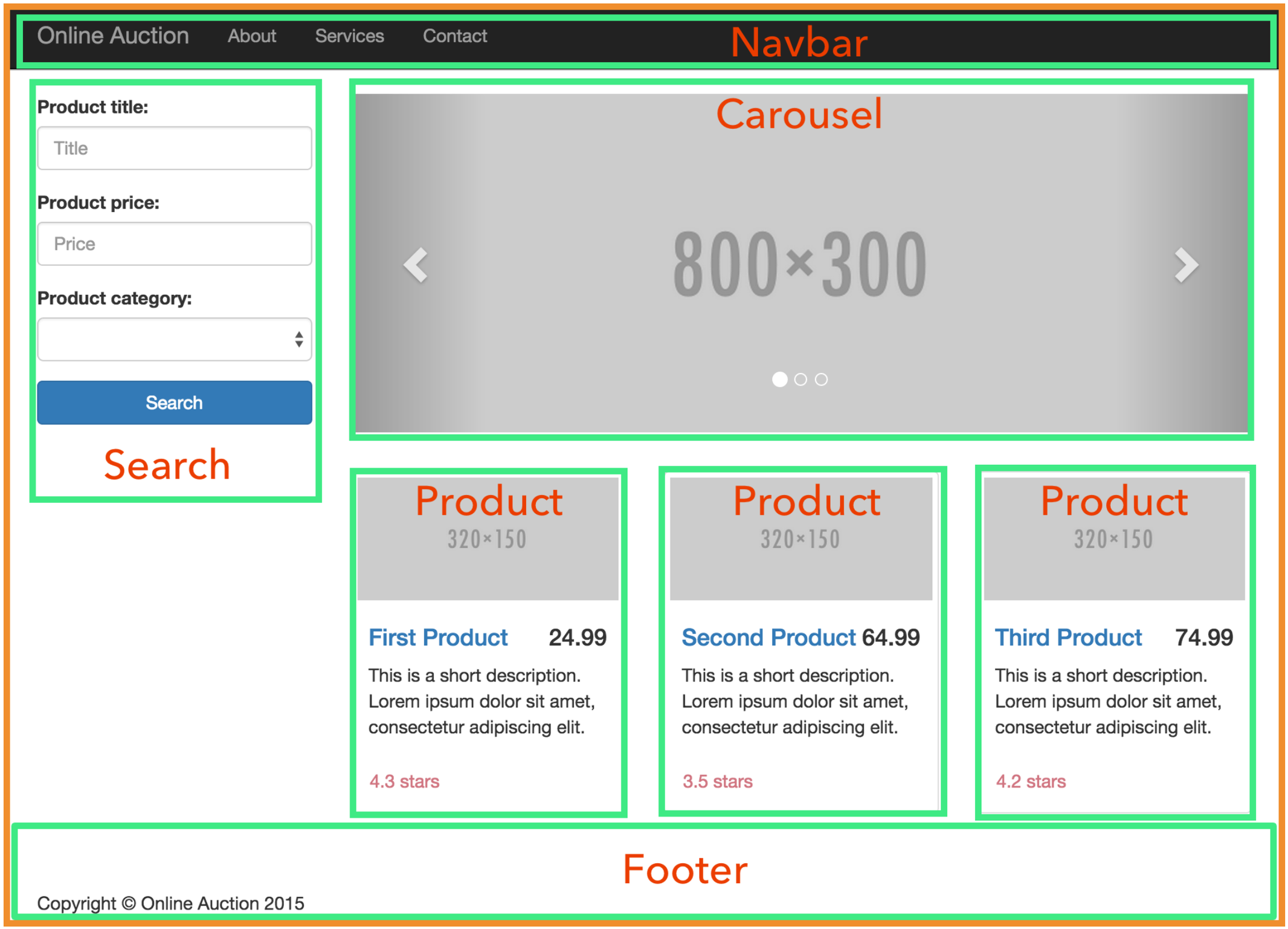
```
@Component({
  selector: 'hello-world',
  template: '<h1>Hello {{ name }}!</h1>'
})
class HelloWorldComponent {
  name: string;

  constructor() {
    this.name = 'Angular 2';
  }
}
```

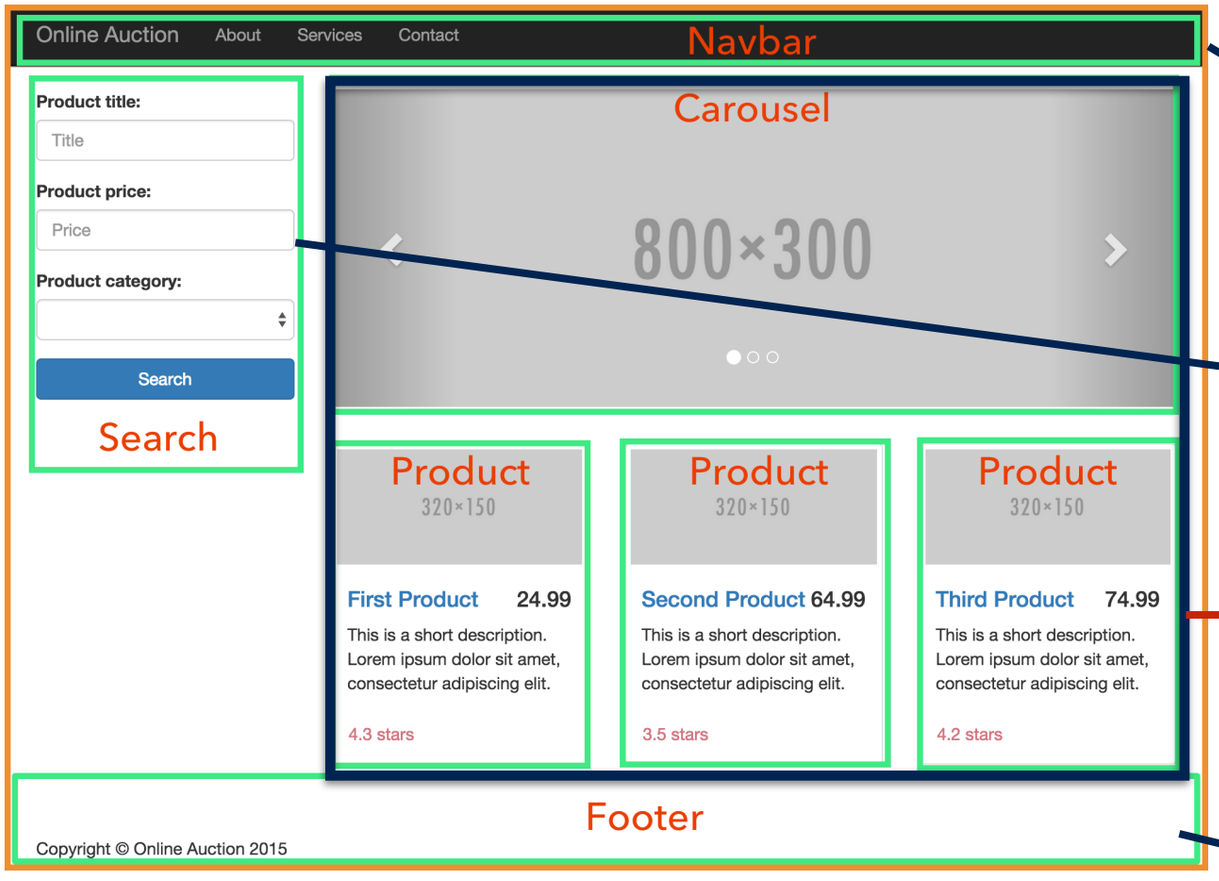
In HTML:

<hello-world></hello-world>

- A class annotated with `@Component`
- Can have inputs and outputs
- Has lifecycle hooks

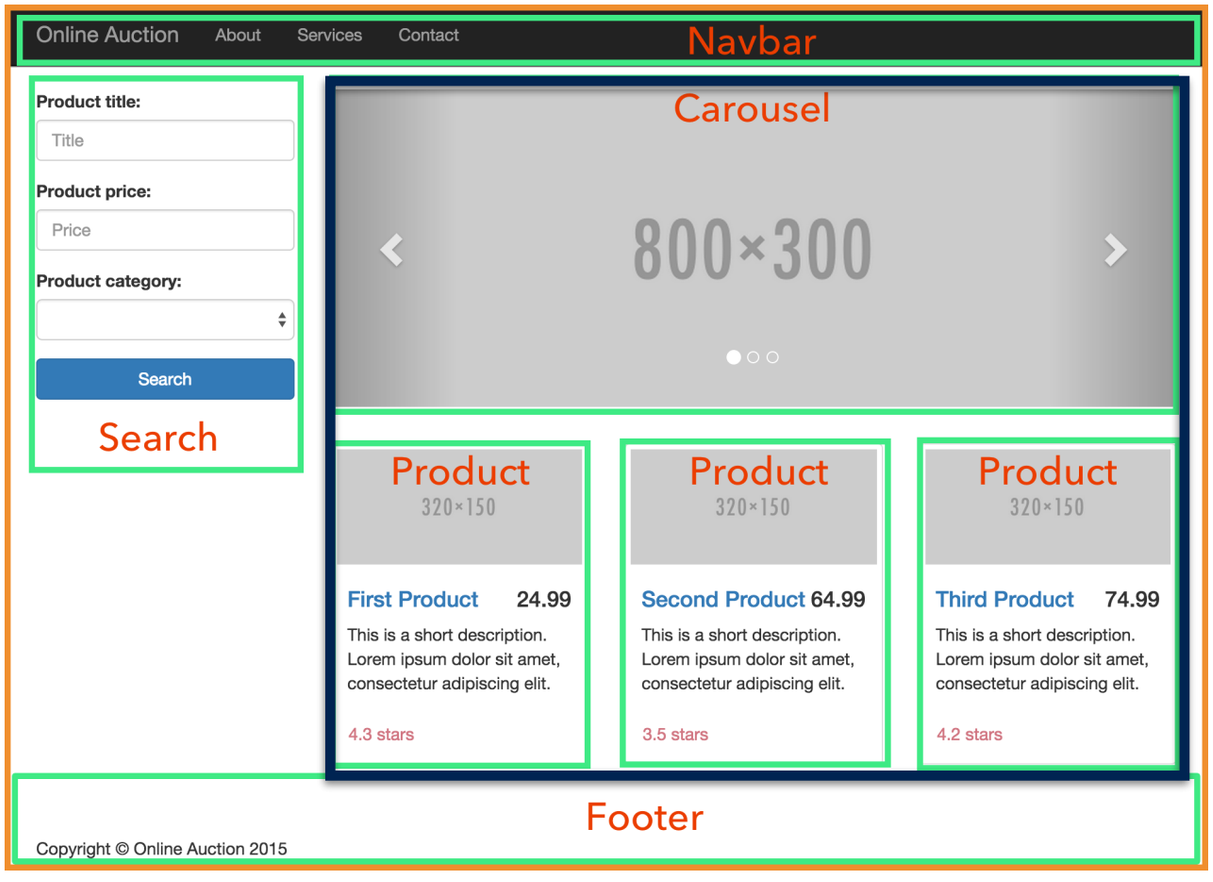
An app is a tree of components






```
<auction-navbar></auction-navbar>

<div class="container">
  <div class="row">
    <div class="col-md-3">
      <auction-search></auction-search>
    </div>

    <div class="col-md-9">
      <router-outlet></router-outlet>
    </div>
  </div>
</div>

<auction-footer></auction-footer>
```

HTML




```
<auction-navbar></auction-navbar>

<div class="container">
  <div class="row">
    <div class="col-md-3">
      <auction-search></auction-search>
    </div>

    <div class="col-md-9">
      <router-outlet></router-outlet>
    </div>
  </div>
</div>

<auction-footer></auction-footer>
```

HTML

```
// import …

@Component({
  selector: 'auction-application',
  templateUrl: 'app/components/application/application.html',
  directives: [
    NavbarComponent,
    FooterComponent,
    SearchComponent,
    HomeComponent
  ]
})
@Routes([
  {path: '/', component: HomeComponent},
  {path: '/products/:prodTitle', component: ProductDetailComponent}
])
export default class ApplicationComponent {}
```

Router Configuration

Component Router: main elements

- **@Routes** - map URLs to components to render inside the <router-outlet>

```
@Routes([
    {path: '/', component: HomeComponent},
    {path: '/product/:id', component: ProductDetailComponentParam}
]
```

- **RouterLink** ([routerLink]) - a link to a named route

```
template: `<a [routerLink]="['/Home']">Home</a>
           <a [routerLink]="['/ProductDetail', {id:1234}]">Product Details</a>`
```

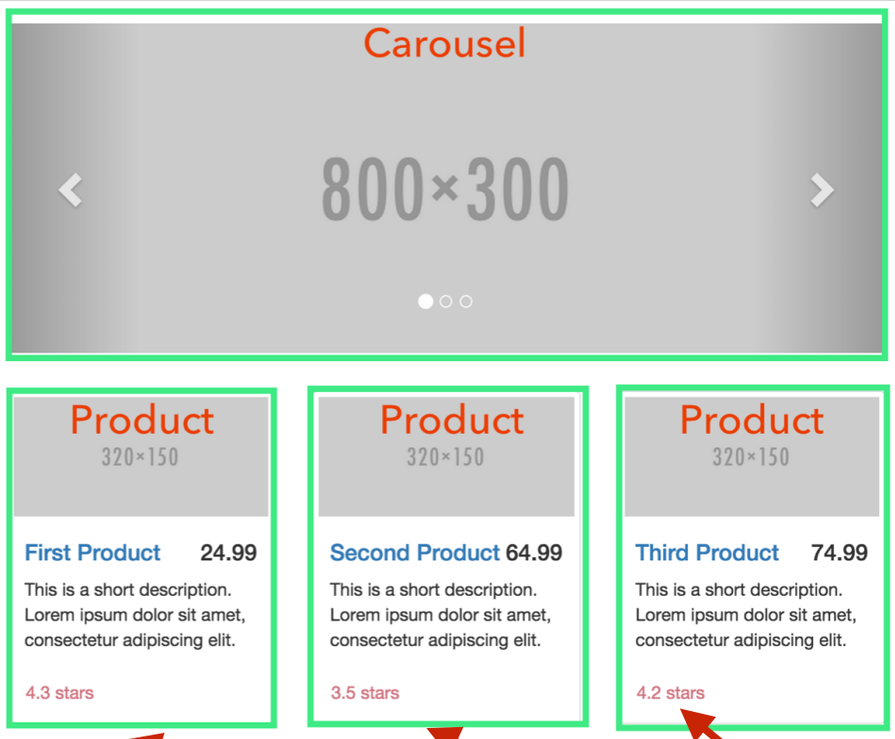
- **RouterOutlet** (<router-outlet>) - where the router should render the component

```
template: `…
           <router-outlet></router-outlet>
           `
```

- **RouteSegment** - a map of key-value pairs to pass parameters to the route

```
constructor(params: RouteSegment){
    this.productID = params.getParam('id');
}
```

HomeComponent




```
import …

@Component({
  selector: 'auction-home-page',
  directives: [
    CarouselComponent,
    ProductItemComponent
  ],
  styleUrls: ['app/components/home/home.css'],
  template: `
    <div class="row carousel-holder">
      <div>
        <auction-carousel></auction-carousel>
      </div>
    </div>
    <div>
      <div *ngFor=“let product of products">
        <auction-product-item [product]="product">
        </auction-product-item>
      </div>
    </div>
  `
})
export default class HomeComponent {
  products: Product[] = [];

  constructor(private productService: ProductService) {
    this.products = productService.getProducts();
  }
}
```

DI

Unidirectional Binding

From code to template:

Properties

<h1>Hello {{ **name** }}!</h1>

<span **[hidden]**="isZipcodeValid">Zip code is not valid

From template to code:

Events

<button **(click)**="placeBid()">Place Bid</button>

<input placeholder= "Product name" **(input)**="onInputEvent()">

Two-way Binding

Synchronizing Model and View:

```
<input [(ngModel)] = "myComponentProperty">
```

The `value` property of the **<input>** and `myComponentProperty` will be always in sync

Dependency Injection

- Angular injects objects into components **only via constructors**
- Each component has **its own injector**
- An injector looks at the **provider** to see how to inject
- Specify a provider either on the component or in one of its anscestors

ProductService

```
export class ProductService {
  getProduct(): Product {
    return new Product( 0, "iPhone 7", 249.99,
                "The latest iPhone, 7-inch screen");
  }
}
```

```
export class Product {

  constructor(
    public id: number,
    public title: string,
    public price: number,
    public description: string) {
  }
}
```

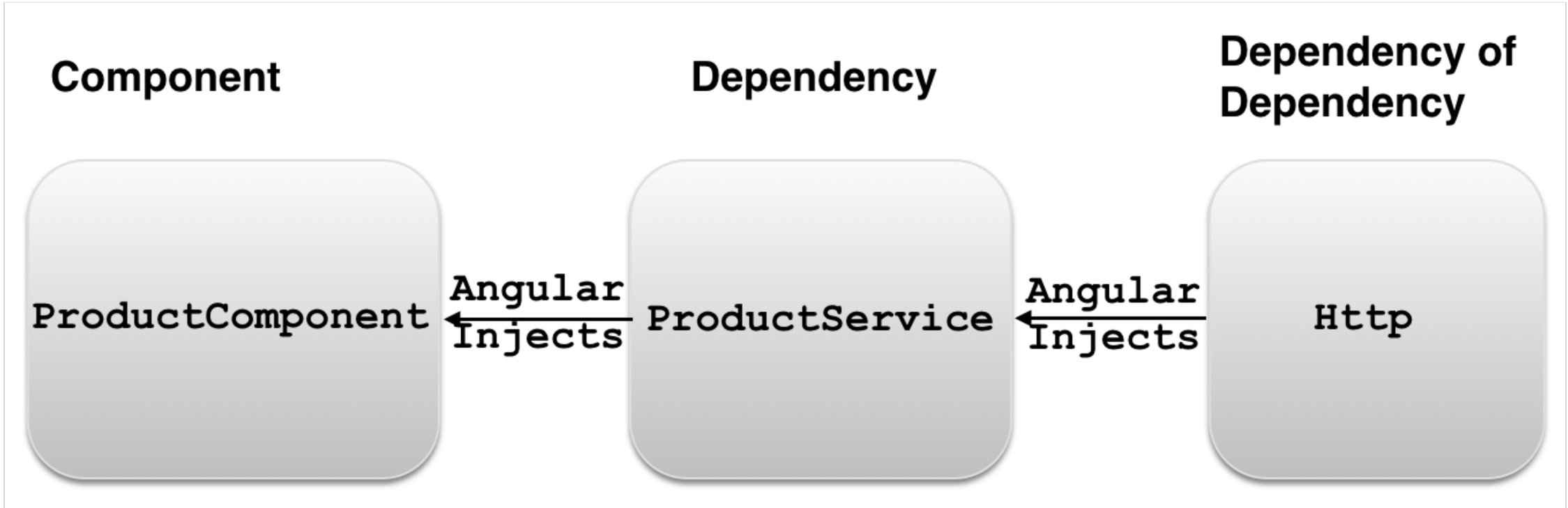
Injecting ProductService

```
import {Component, bind} from '@angular/core';
import {ProductService, Product} from "../services/product-service";

@Component({
  selector: 'di-product-page',
  template: `<div>
             <h1>Product Details</h1>
             <h2>Title: {{product.title}}</h2>
             <h2>Description: {{product.description}}</h2>
             </div>`,
  providers:[ProductService]
})

export default class ProductComponent {
  product: Product;

  constructor( productService: ProductService) {

    this.product = productService.getProduct();
  }
}
```

A provider can be a class, factory, or a value

An injection point

Injecting Dependencies of Dependencies




```
@Injectable
export class ProductService {

  constructor(private http:Http){
    let products = http.get('products.json');
  }

   …
}
```

Input and Output Properties

- A component is a black box with outlets (properties)
- Properties marked as **`@Input()`** are for **geting the data into a component** from the parent
- Properties marked as **`@Output()`** are for **emitting events from a component**

Binding to child's inputs

```
@Component({
    selector: 'app',
    template: `
    <input type="text" placeholder="Enter stock (e.g. AAPL)"
                       (change)="onInputEvent($event)">
    <br/>


    <order-processor [stockSymbol]="stock"></order-processor>
    `,
    directives: [OrderComponent]
})
class AppComponent {
    stock:string;

    onInputEvent({target}):void{
        this.stock=target.value;
    }
}
```

Binding to the child's prop

Using Destructuring

```
@Component({
    selector: 'app',
    template: `
    <input type="text" placeholder="Enter stock (e.g. AAPL)"
                       (change)="onInputEvent($event)">
    <br/>


    <order-processor [stockSymbol]="stock"></order-processor>
  `,
    directives: [OrderComponent]
})
class AppComponent {
    stock:string;

    onInputEvent({target}):void{
        this.stock=target.value;
    }
}
```

Destructuring

Property
bubbles
cancelable
currentTarget
defaultPrevented
eventPhase
isTrusted
target
timeStamp
type
view

Child receieves data via input properties

```
import {bootstrap} from '@angular/platform-browser-dynamic';
import {Component, Input} from '@angular/core';

@Component({
    selector: 'order-processor',
    template: `
    Buying {{quantity}} shares of {{stockSymbol}}
  `})
class OrderComponent {

    @Input() quantity: number;

    private _stockSymbol: string;

    @Input()
    set stockSymbol(value: string) {
        this._stockSymbol = value;

        console.log(`Sending a Buy order to NASDAQ: ${this.stockSymbol} ${this.quantity}`);
    }

    get stockSymbol(): string {
        return this._stockSymbol;
    }
}
```

Child sends events via an output property

```
@Component({
    selector: 'price-quoter',
    template: `<strong>Inside PriceQuoterComponent:
               {{stockSymbol}} {{price | currency:'USD':true:'1.2-2'}}</strong>`,

    styles:[`:host {background: pink;}`]
})
class PriceQuoterComponent {

    @Output() lastPrice: EventEmitter <IPriceQuote> = new EventEmitter();

    stockSymbol: string = "IBM";
    price:number;

    constructor() {
        setInterval(() => {
            let priceQuote: IPriceQuote = {
                stockSymbol: this.stockSymbol,

                lastPrice: 100*Math.random()
            };

            this.price = priceQuote.lastPrice;

            this.lastPrice.emit(priceQuote);

        }, 1000);
    }
}
```

A child emits events via output properties

Parent listens to an event

```
@Component({
    selector: 'app',
    template: `
      <price-quoter (lastPrice)="priceQuoteHandler($event)"></price-quoter><br>

      AppComponent received: {{stockSymbol}} {{price | currency:'USD':true:'1.2-2'}}
    `,

    directives: [PriceQuoterComponent]
})
class AppComponent {

    stockSymbol: string;
    price:number;

    priceQuoteHandler(event:IPriceQuote) {
        this.stockSymbol = event.stockSymbol;
        this.price = event.lastPrice;
    }
}
```

pipe

```
interface IPriceQuote {
    stockSymbol: string;
    lastPrice: number;
}
```

Projection

- A component can have only one template
- But a parent can pass HTML to the child's template during the runtime
- Add the **<ng-content>** to the child's template

Parent

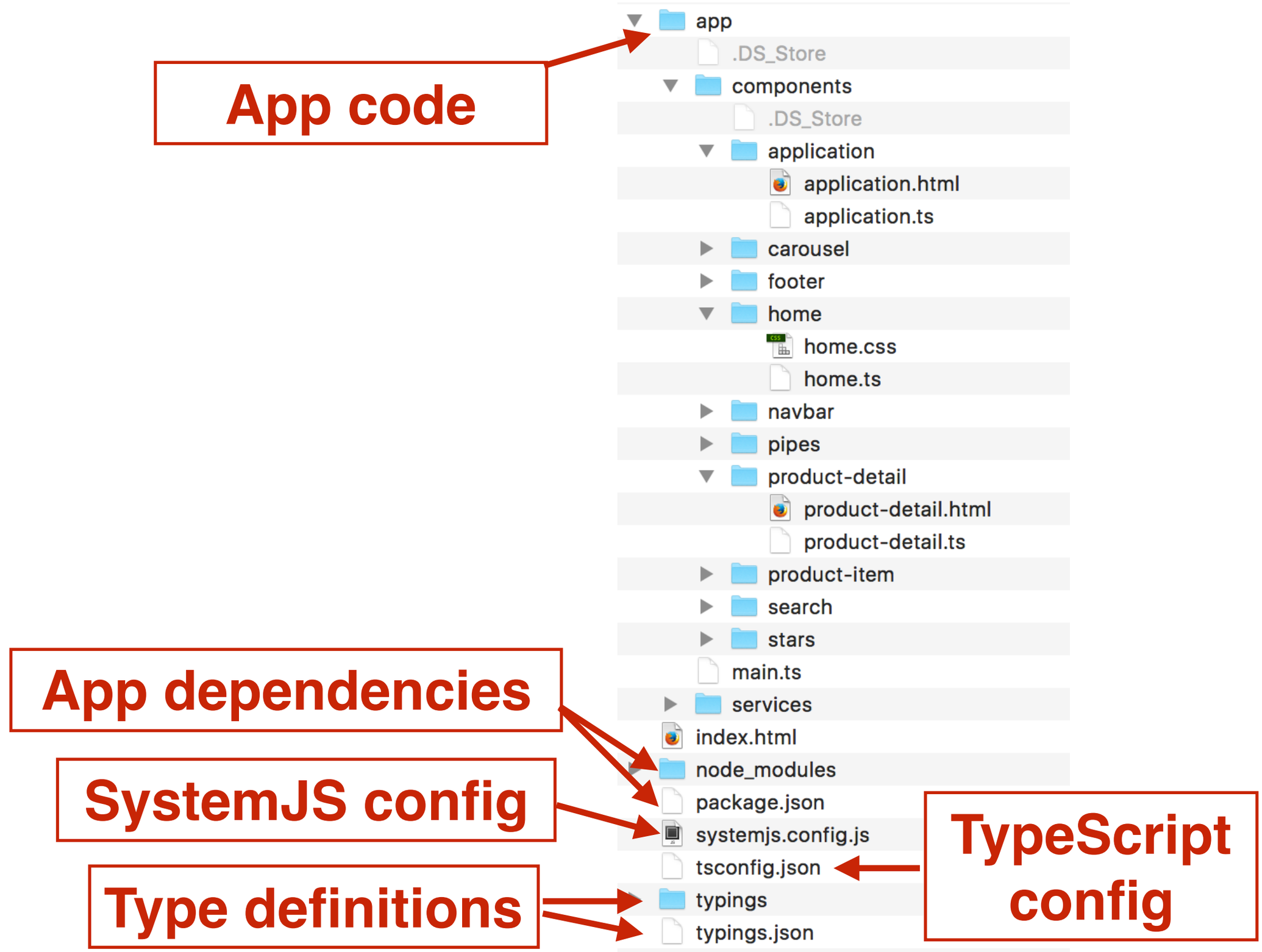
```
template: `
   <div class="wrapper">
    <h2>Parent</h2>
     <div>This div is defined in the Parent's template</div>
     <child>
       <div>Parent projects this div onto the child </div>
     </child>
   </div>`
```

Child

```
template: `
  <div class="wrapper">
   <h2>Child</h2>
    <div>This is child's content</div>

    <ng-content></ng-content>

  </div>`
```

◀---- insertion point

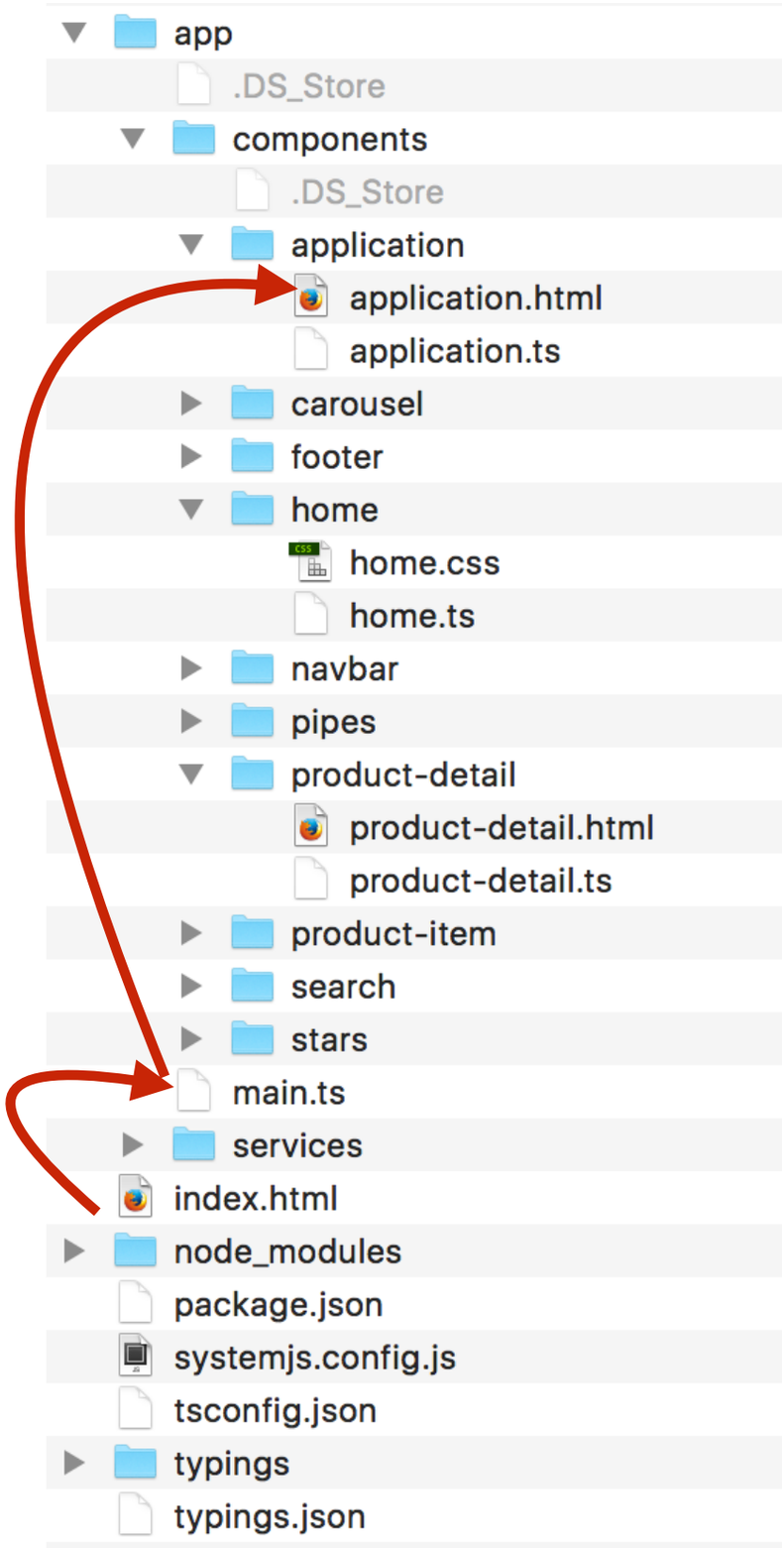
Sample Project Structure



Sample Project Structure

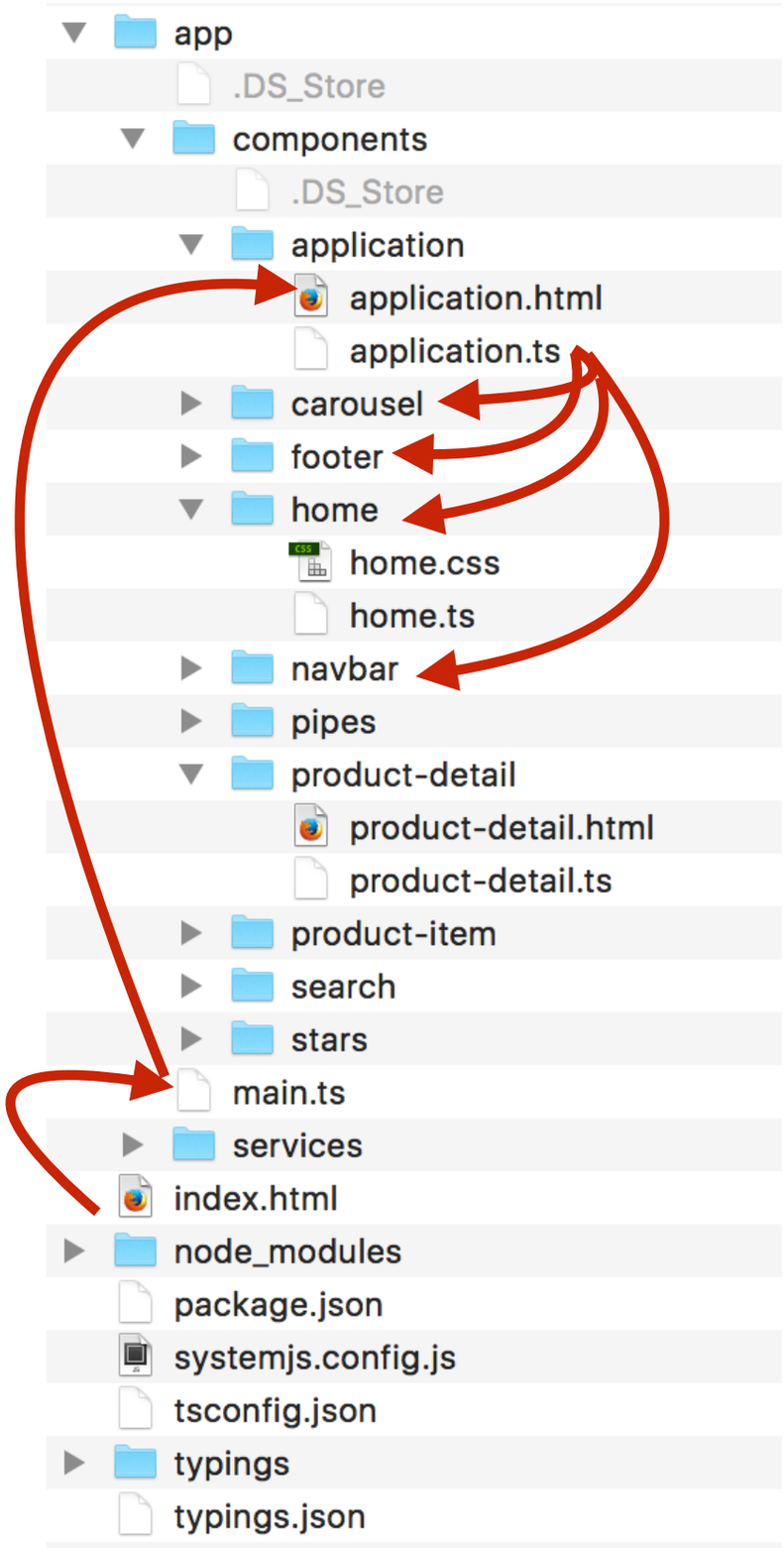
SystemJS transpiles TS and loads JS modules

```
bootstrap(ApplicationComponent)
```



Sample Project Structure

```
bootstrap(ApplicationComponent)
```

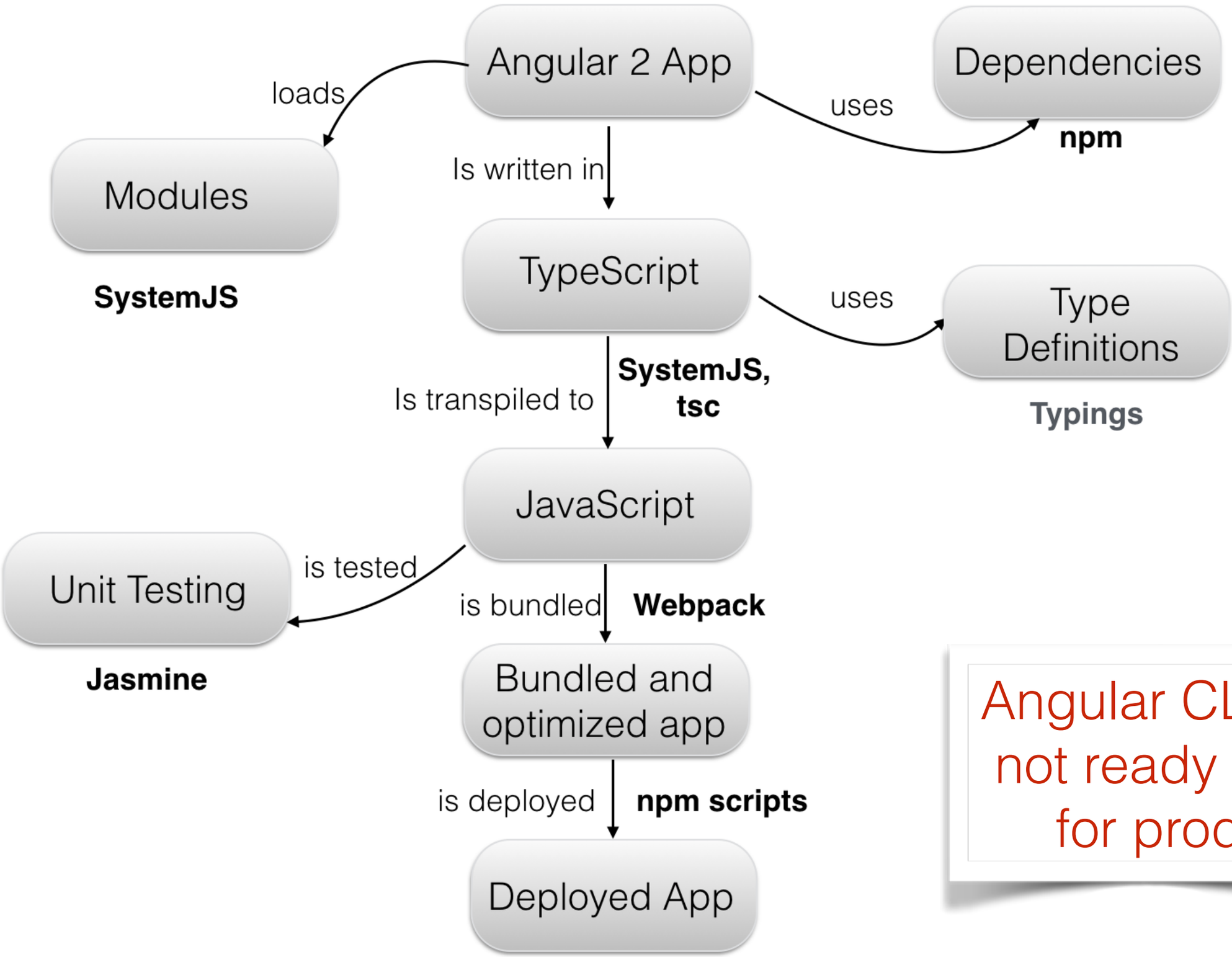


Demo

CH5: auction
npm start

https://github.com/Farata/angular2typescript

The tools that can be used today



Angular CLI is not ready yet for prod

TypeScript Compiler: tsc

- Install the typescript compiler with npm (comes with Node.js):

  npm install -g typescript

- Compile **main.ts** into **main.js** specifying target language syntax:

  tsc —t ES5 main.ts

- Usually the compiler options are set in tsconfig.json file
- The watch mode allows to auto-compile as file changes:

  tsc -w *.ts

Starting a new project with npm

1. Generate *package.json* for your project:
   npm init -y

2. Add Angular dependencies to *package.json*

3. Download dependencies into the dir *node_modules*:
   npm install

4. Install live-server
   npm install live-server -g

HelloWorldComponent in Angular: **main.ts**

```
import {bootstrap} from 'angular2/platform/browser';
import {Component} from 'angular2/core';

@Component({
  selector: 'hello-world',
  template: '<h1>Hello {{ name }}!</h1>'
})
class HelloWorldComponent {
  name: string;

  constructor() {
    this.name = ‘World!';
  }
}

bootstrap(HelloWorldComponent);
```

In HTML:

<hello-world></hello-world>

package.json

To run from cmd window

App dependencies

Dev dependencies

npm docs:
https://docs.npmjs.com

```
{
  "name": "test_samples",
  "description": "A sample Weather app",
  "private": true,
  "scripts": {
    "start": "live-server",
    "test": "karma start karma.conf.js"
  },
  "dependencies": {
    "@angular/common": "2.0.0-rc.1",
    "@angular/compiler": "2.0.0-rc.1",
    "@angular/core": "2.0.0-rc.1",
    "@angular/http": "2.0.0-rc.1",
    "@angular/platform-browser": "2.0.0-rc.1",
    "@angular/platform-browser-dynamic": "2.0.0-rc.1",
    "@angular/router": "2.0.0-rc.1",
    "reflect-metadata": "^0.1.3",
    "rxjs": "5.0.0-beta.6",
    "systemjs": "^0.19.27",
    "zone.js": "^0.6.12"
  },
  "devDependencies": {
    "jasmine-core": "^2.4.1",
    "karma": "^0.13.22",
    "karma-chrome-launcher": "^0.2.3",
    "karma-firefox-launcher": "^0.1.7",
    "karma-jasmine": "^0.3.8",
    "live-server": "0.8.2",
    "typescript": "^1.8.10"
  }
}
```

SystemJS: a Universal Module Loader

- ES6 defines modules, but not the loader
- ES7 will include the `System` object for loading modules
- SystemJS is a polyfill that loads modules

index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Basic Routing Samples</title>
  <script src="node_modules/zone.js/dist/zone.js"></script>
  <script src="node_modules/typescript/lib/typescript.js"></script>
  <script src="node_modules/reflect-metadata/Reflect.js"></script>
  <script src="node_modules/systemjs/dist/system.src.js"></script>

  <script src="systemjs.config.js"></script>

  <script>
    System.import('app')
          .catch(function (err) {console.error(err);});
  </script>
</head>
<body>
  <my-app></my-app>
</body>
</html>
```

systemjs.config.js

```
System.config({
    transpiler: 'typescript',
    typescriptOptions: {emitDecoratorMetadata: true},
    map: {
        'app' : 'app',
        'rxjs': 'node_modules/rxjs',
        '@angular'                          : ‘node_modules/@angular’,

     packages: {
        'app'                                   : {main: 'main.ts', defaultExtension: 'ts'},
        'rxjs'                                  : {main: 'index.js'},
        '@angular/core'                         : {main: 'index.js'},
        '@angular/common'                       : {main: 'index.js'},
        '@angular/compiler'                     : {main: 'index.js'},
        '@angular/router'                       : {main: 'index.js'},
        '@angular/platform-browser'             : {main: 'index.js'},
        '@angular/platform-browser-dynamic': {main: 'index.js'},
        '@angular/http'                         : {main: 'index.js'}

      }
});
```

Demo

CH9: test_weather

npm start

Reactive Extensions

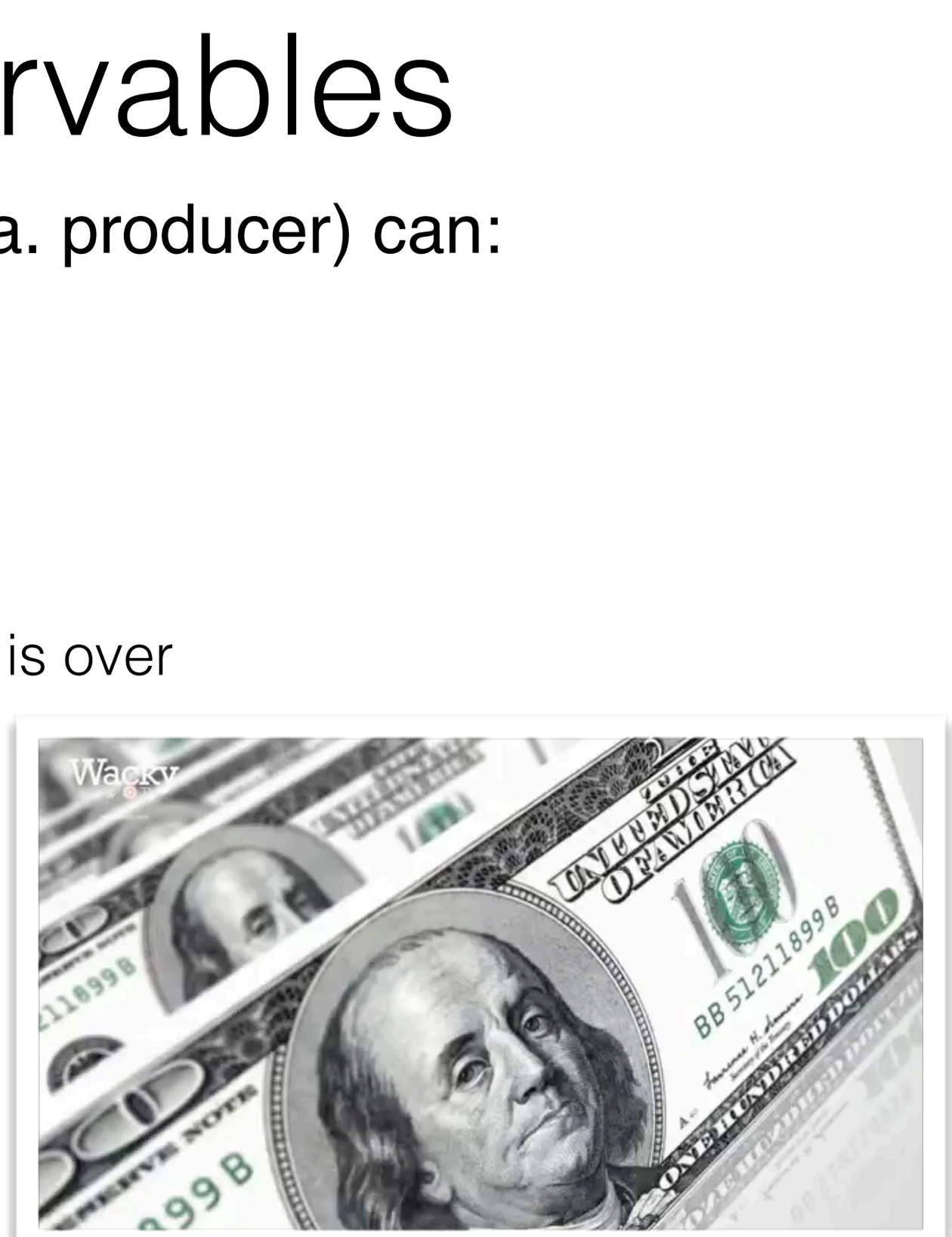
- Angular comes with RxJS library of reactive extensions
- A **observable function** emits the data over time to a subscriber.
- Supports multiple operators to transform the stream's data
- Stream samples:
  - UI events
  - HTTP responses
  - Data arriving over websockets

Observables

An Observable object (a.k.a. producer) can:

- Stream elements
- Throw an error
- Send a signal that the streaming is over

The streaming starts when your code invokes subscribe() on the observable



Observers

An Observer object (a.k.a. consumer) provides:

- A function to handle the next object from the stream
- A function for error handling
- A function for end-of-stream handling

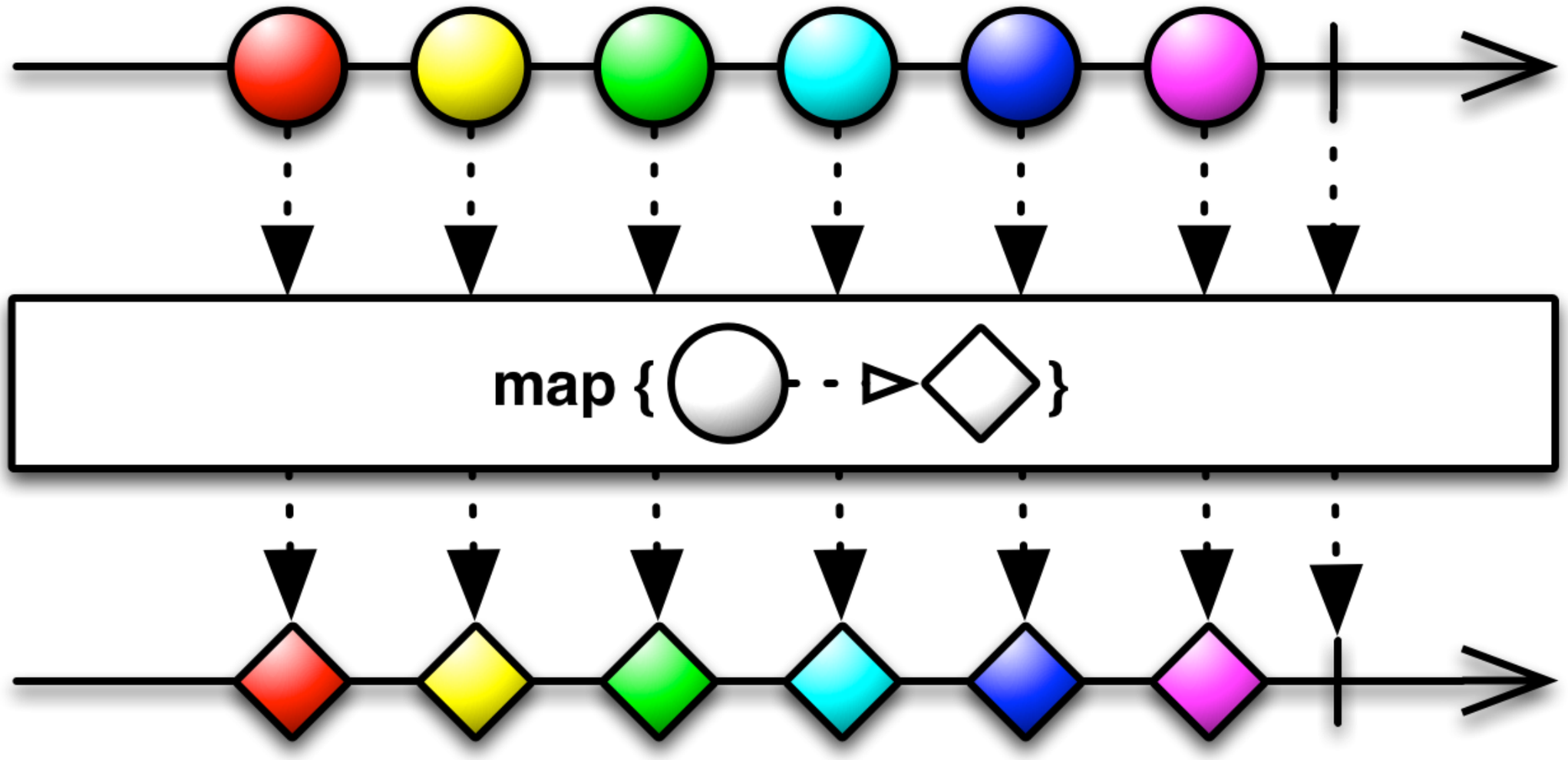
An Operator

transforms an observable into another observable

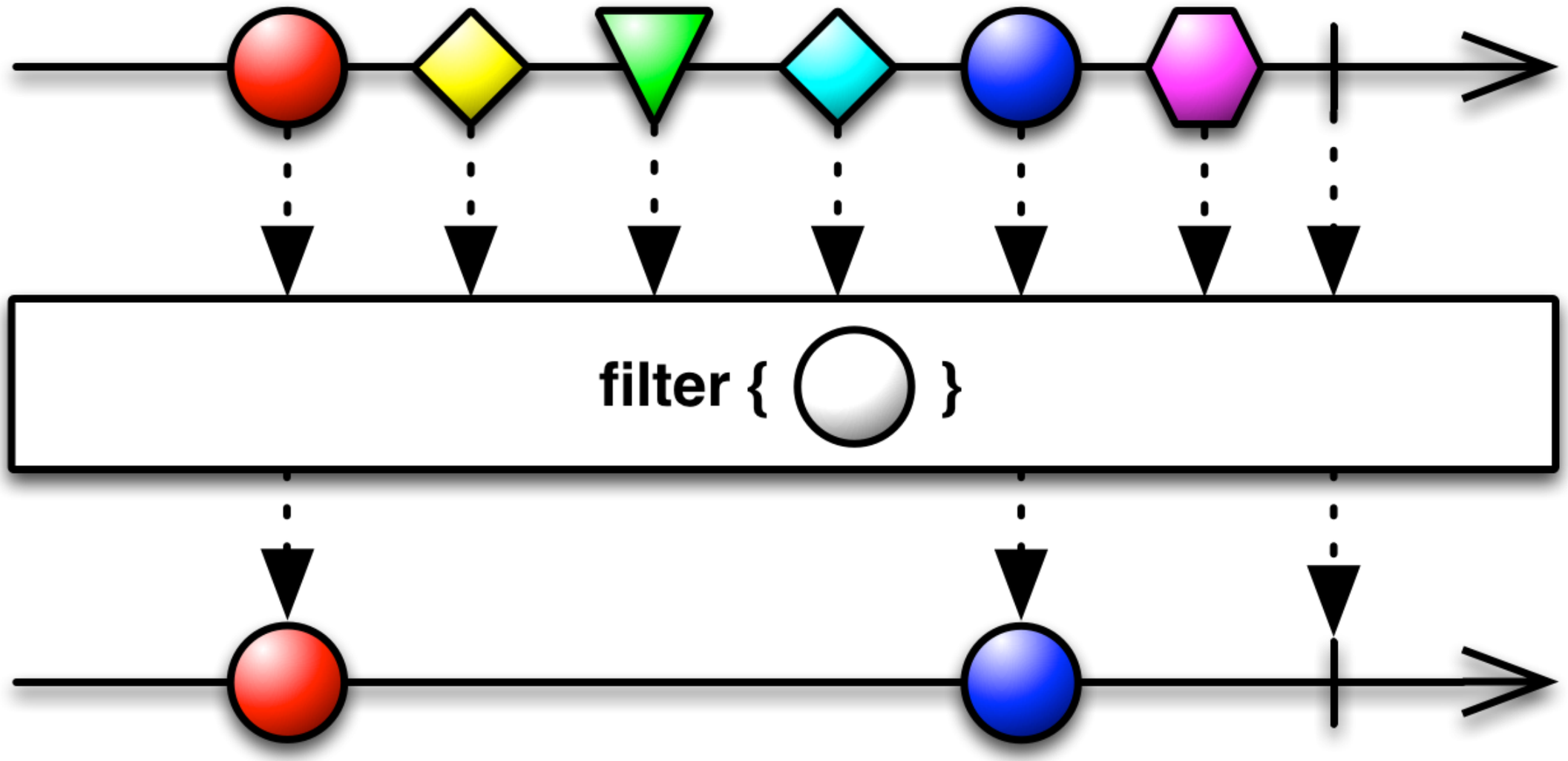
Observable →  → Observable

A transforming function

A map() operator

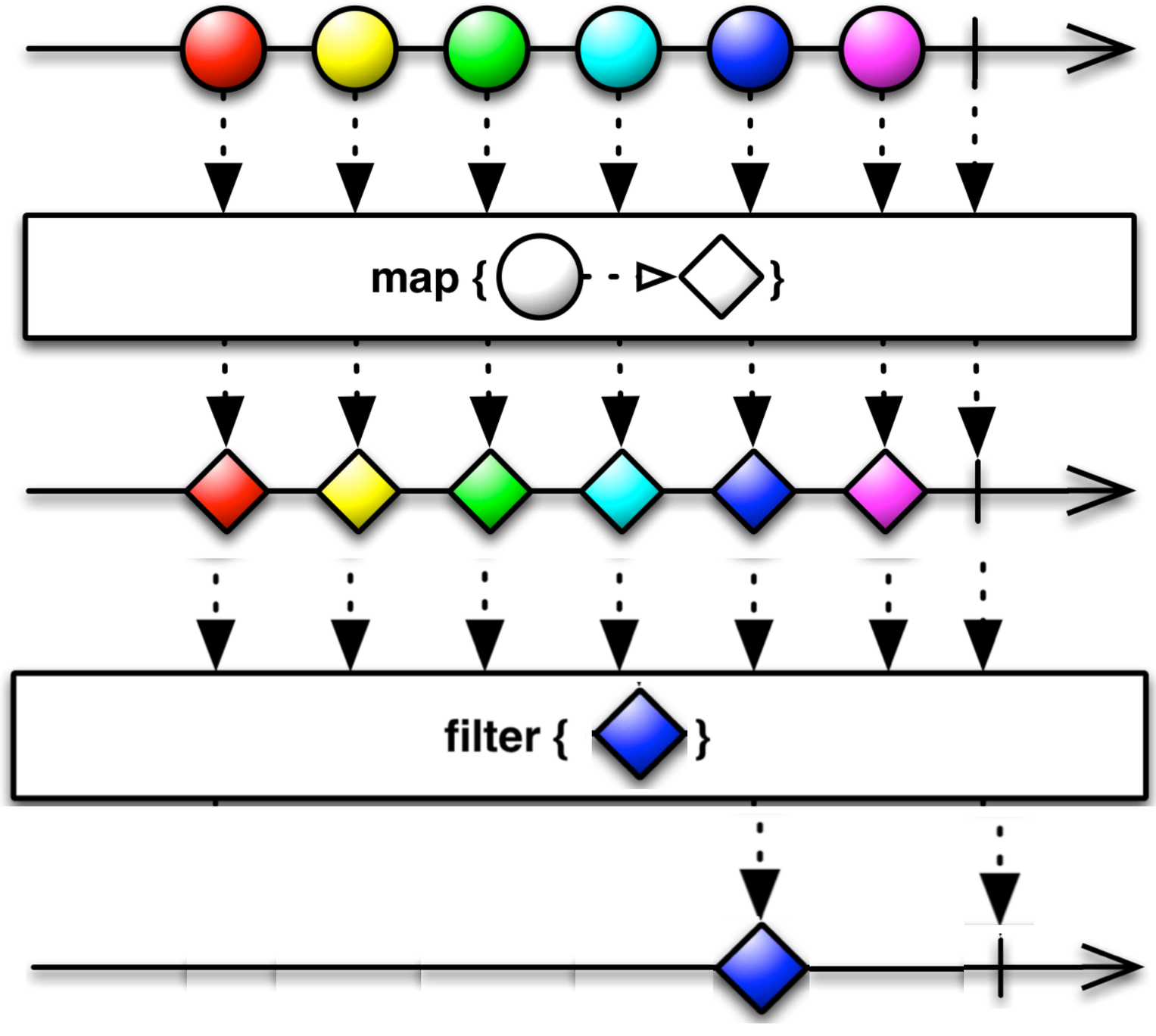


A filter() operator



http://rxmarbles.com

Operator chaining: map and filter



Observables in the UI

```
@Component({
    selector: "app",
    template: `
      <h2>Observable events demo</h2>
      <input type="text" placeholder="Enter stock" [ngFormControl]="searchInput">
    `
})
class AppComponent {

    searchInput: Control;

    constructor(){
        this.searchInput = new Control('');

        this.searchInput.valueChanges
            .debounceTime(500)
            .subscribe(stock => this.getStockQuoteFromServer(stock));
    }

    getStockQuoteFromServer(stock) {

        console.log(`The price of ${stock} is ${100*Math.random().toFixed(4)}`);
    }
}
```

Observable

Http and Observables

```
class AppComponent {

  products: Array<string> = [];

  constructor(private http: Http) {

    this.http.get('http://localhost:8080/products')
        .map(res => res.json())
        .subscribe(
          data => {
                this.products=data;
          },
          err =>
            console.log("Can't get products. Error code: %s, URL: %s ",
                                                    err.status, err.url),
          () => console.log('Product(s) are retrieved')
        );
  }
}
```

DI

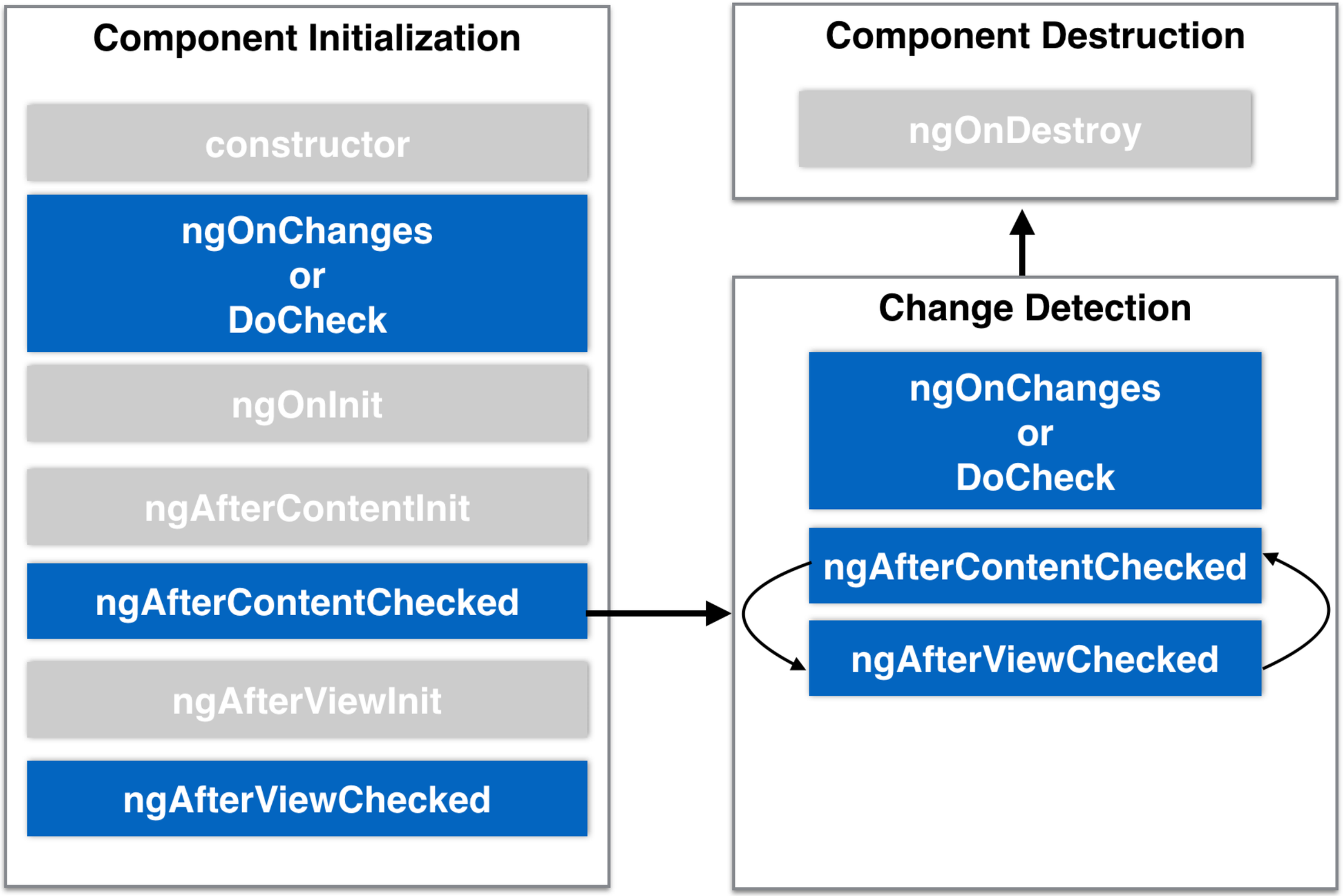
Observer

Change Detection

- **Zone.js** monitors all async events
- Every component has its own change detector
- CD is unidirectional and makes a single pass
- The changes can come only from a component
- Change detection runs from top to bottom of the component tree
- `OnPush` - don't run change detection unless bindings to input props change

```
changeDetection: ChangeDetectionStrategy.OnPush
```

Component Lifecycle



Building and Deploying Apps with Webpack

Objectives

- A SystemJS-based project makes too many requests and downloads megabytes

We want:

- Minimize the number of requests
- Reduce the download size
- Automate the build in dev and prod

Webpack bundler

1. Gained popularity after it was adopted by Instagram

2. It’s a powerful and production-ready tool

3. The config file is compact

4. Has its own loader (doesn’t use SystemJS)

Webpack Hello World

main.js

```
document.write('Hello World!');
```

Create a bundle:

```
webpack main.js bundle.js
```

webpack.config.js

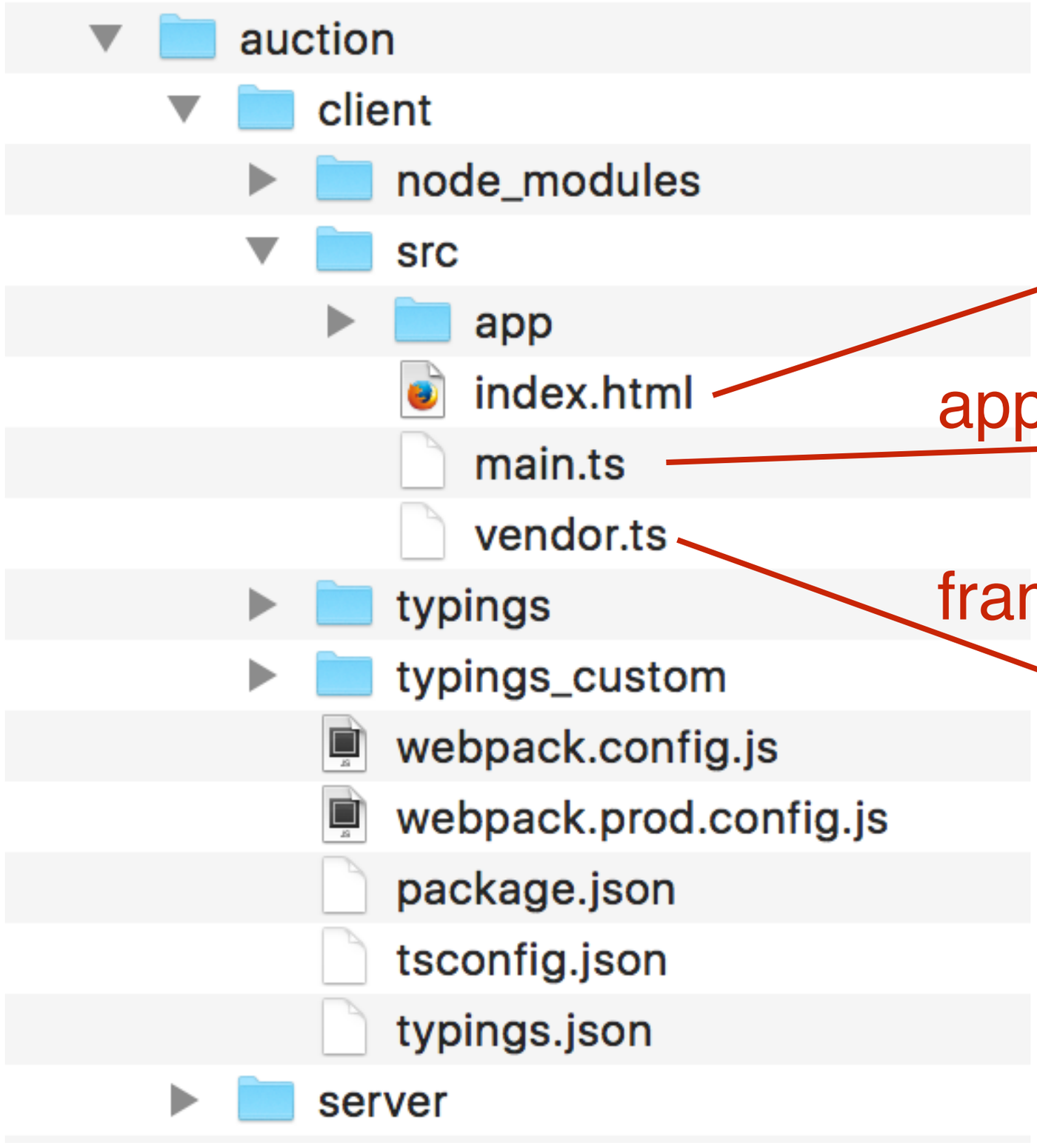
```
module.exports = {
  entry: './main',
  output: {
    path: './dist',
    filename: 'bundle.js'
  },
  watch: true,
  devServer: {
    contentBase: '.'
  }
};
```

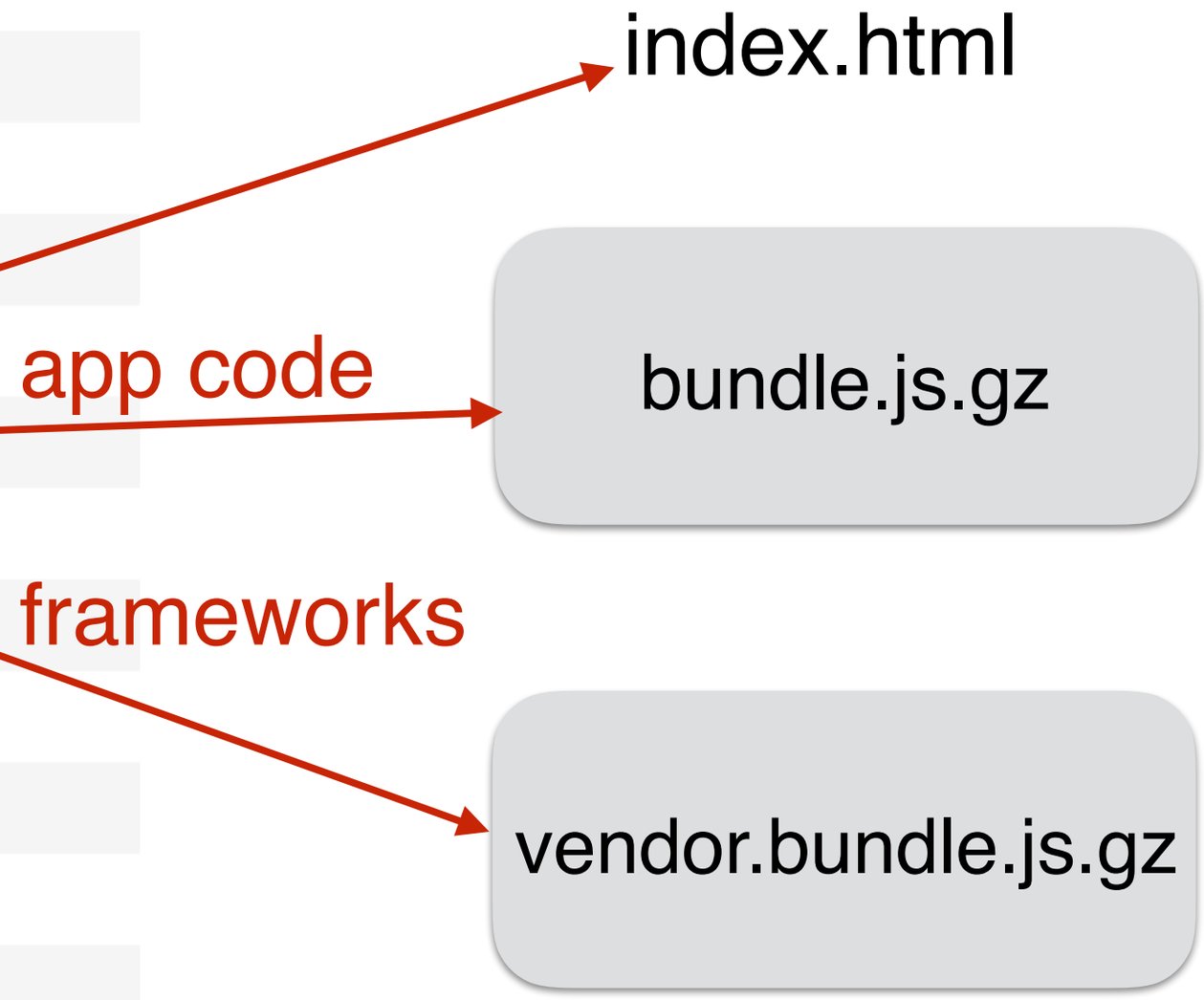
index.html

```
<!DOCTYPE html>
<html>
<head></head>
<body>
  <script src="/dist/bundle.js"></script>
</body>
</html>
```

Source Code

Deployed Code





Webpack Dev Server

Webpack Dev Server

In-memory

index.html

bundle.js

vendor.bundle.js

Prod Server

On file system

index.html

bundle.js.gz

vendor.bundle.js.gz

webpack.prod.config

```
const path = require('path');

const CommonsChunkPlugin   = require('webpack/lib/optimize/CommonsChunkPlugin');
const CompressionPlugin    = require('compression-webpack-plugin');
const CopyWebpackPlugin    = require('copy-webpack-plugin');
const DedupePlugin         = require('webpack/lib/optimize/DedupePlugin');
const DefinePlugin          = require('webpack/lib/DefinePlugin');
const OccurenceOrderPlugin = require('webpack/lib/optimize/OccurenceOrderPlugin');
const UglifyJsPlugin       = require('webpack/lib/optimize/UglifyJsPlugin');

const ENV = process.env.NODE_ENV = 'production';
const metadata = {
  env: ENV
};

module.exports = {
  debug: false,
  devtool: 'source-map',
  entry: {
    'main'  : './src/main.ts',
    'vendor': './src/vendor.ts'
  },
  metadata: metadata,
  module: {
    loaders: [
      {test: /\.css$/,  loader: 'to-string!css', exclude: /node_modules/},
      {test: /\.css$/,  loader: 'style!css', exclude: /src/},
      {test: /\.html$/, loader: 'raw'},
      {test: /\.ts$/,   loader: 'ts', query: {compilerOptions: {noEmit: false}}}
    ]
  },
  output: {
    path    : './dist',
    filename: 'bundle.js'
  },
  plugins: [
    new CommonsChunkPlugin({name: 'vendor', filename: 'vendor.bundle.js', minChunks: Infinity}),
    new CompressionPlugin({regExp: /\.css$|\.html$|\.js$|\.map$/}),
    new CopyWebpackPlugin([{from: './src/index.html', to: 'index.html'}]),
    new DedupePlugin(),
    new DefinePlugin({'webpack': {'ENV': JSON.stringify(metadata.env)}}),
    new OccurenceOrderPlugin(true),
    new UglifyJsPlugin({
      compress: {screw_ie8 : true},
      mangle:  {screw_ie8 : true}
    })
  ],
  resolve: {extensions: ['', '.ts', '.js']}
};
```

index.html after Webpack build

```
<!DOCTYPE html>
<html>
<head>
  <meta charset=UTF-8>
  <title>Angular Webpack Starter</title>
  <base href="/">
</head>
<body>
  <my-app>Loading...</my-app>
  <script src="vendor.bundle.js"></script>
  <script src="bundle.js"></script>
</body>
</html>
```

Demo

Chapter 10, angular2-webpack-starter

https://github.com/Farata/angular2typescript/tree/master/chapter10/angular2-webpack-starter

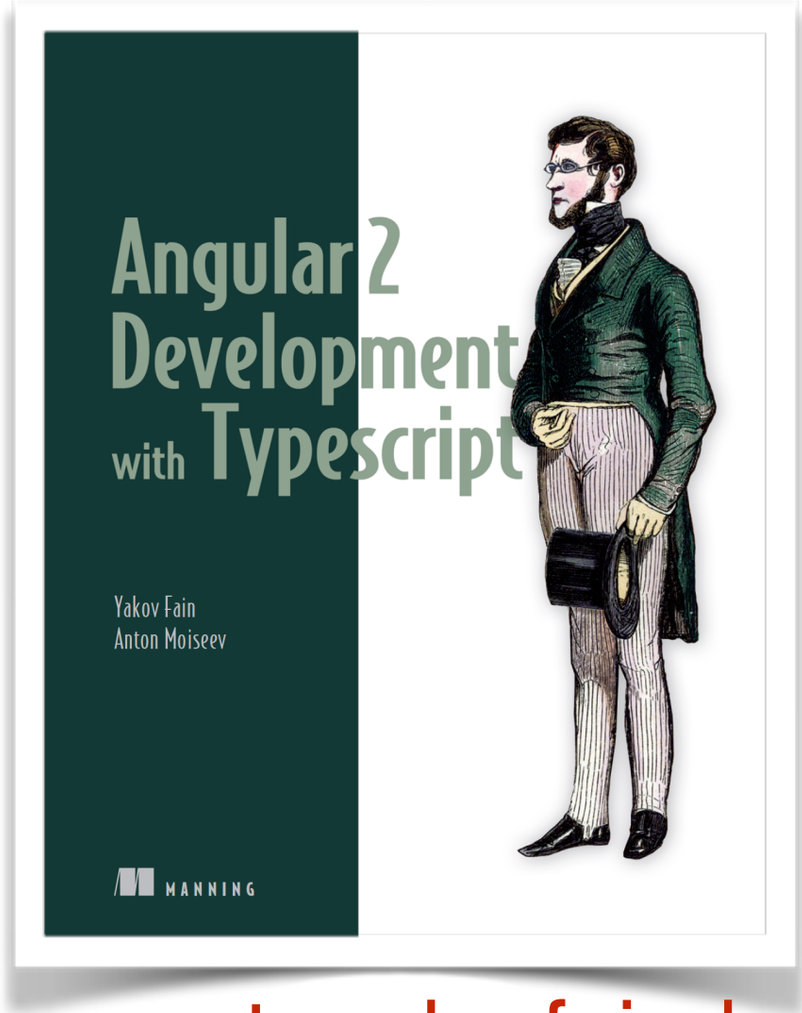
Links

- Angular 2 online workshop, starts on June 12:
  http://bit.ly/1pZICMP
  discount code: jeeconf

- Code samples: https://github.com/farata

- Our company: faratasystems.com

- Blog: yakovfain.com



discount code: faindz