Angular 4

Angular 4 Is …

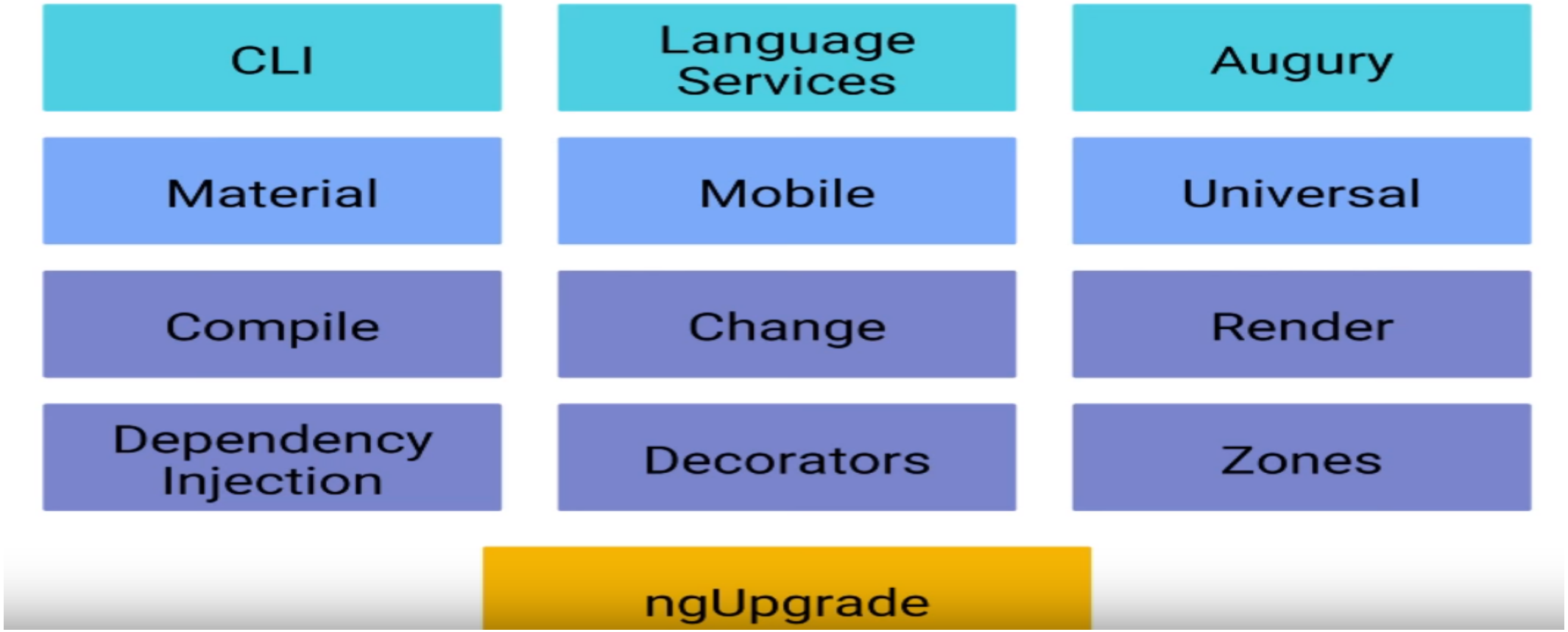
- Angular is a TypeScript-based open-source front-end web application platform for building complex web applications.
- Angular 4 is a complete rewrite from the same team that built AngularJS 1.x.

Angular 4 Features

Features

- Better Performance
- Easier to Use
- Future Proof with ES2015
- Using Web Standards
- Lazy Loading
- Better Dependency Injection

Framework to Platform



Developer Tools

```
> npm install -g angular-cli
> ng new my-dream-app
> cd my-dream-app
> ng serve
```

Angular CLI

Protractor

Angular Augury

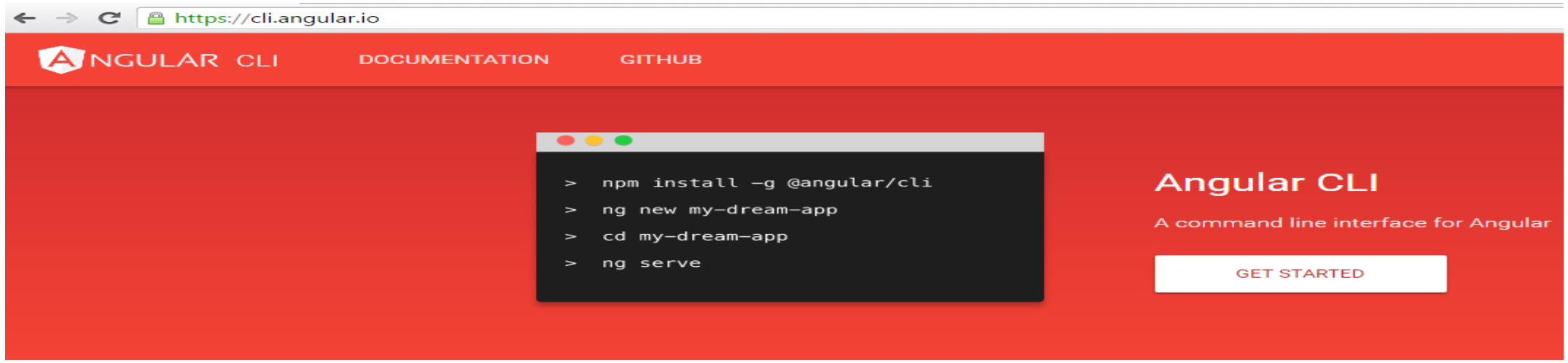
by RANGLE.IO

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

@Component({
    template: '<h1 [style.font]='font'></h1>
    <div>{{person.address.str
    <span (click)="updatePers
})
```

Language Services

Angular CLI

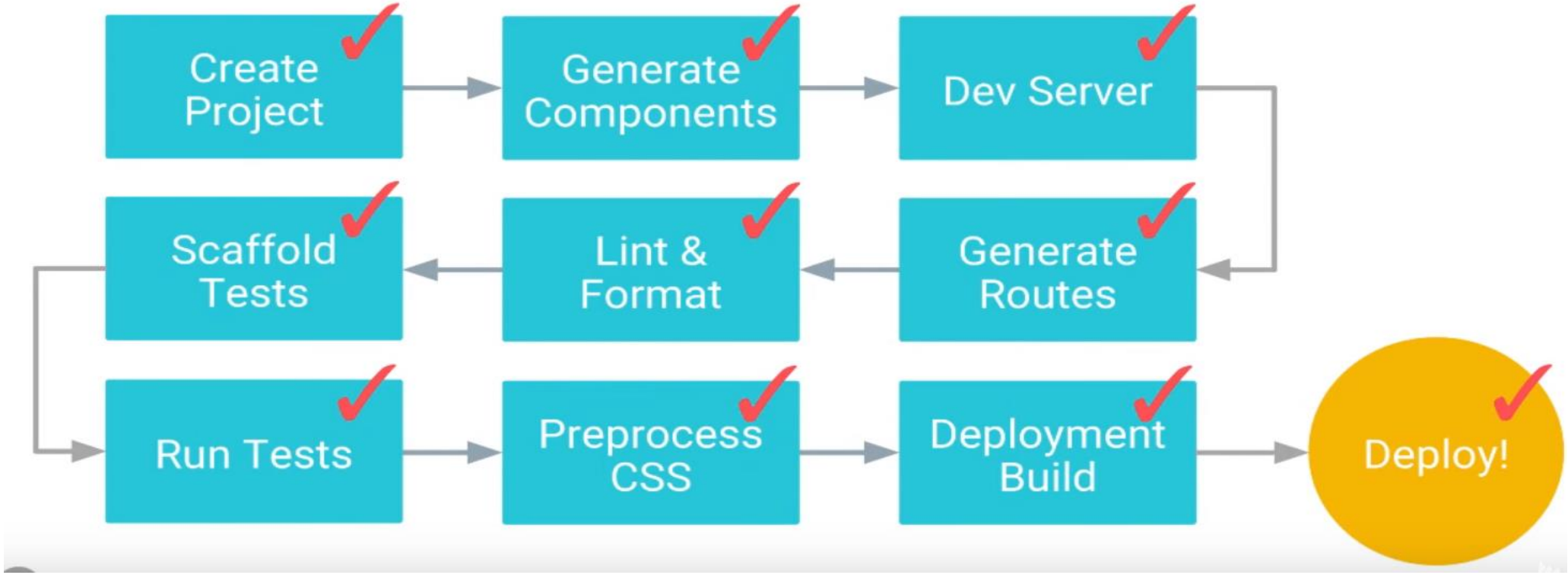


ng new

The Angular CLI makes it easy to create an application that already works, right out of the box. It already follows our best practices!

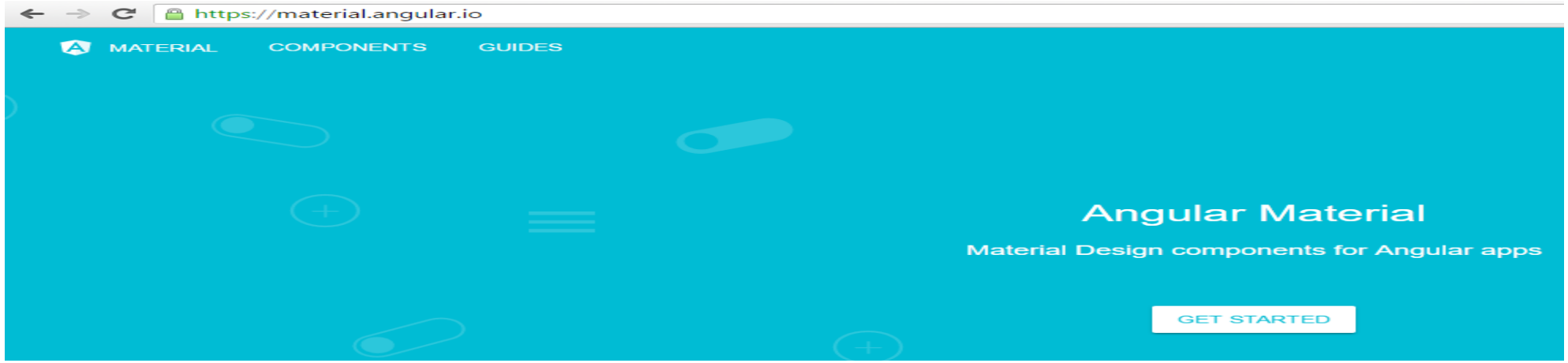
Workflow Automation: CLI

- Everything can be done through commands

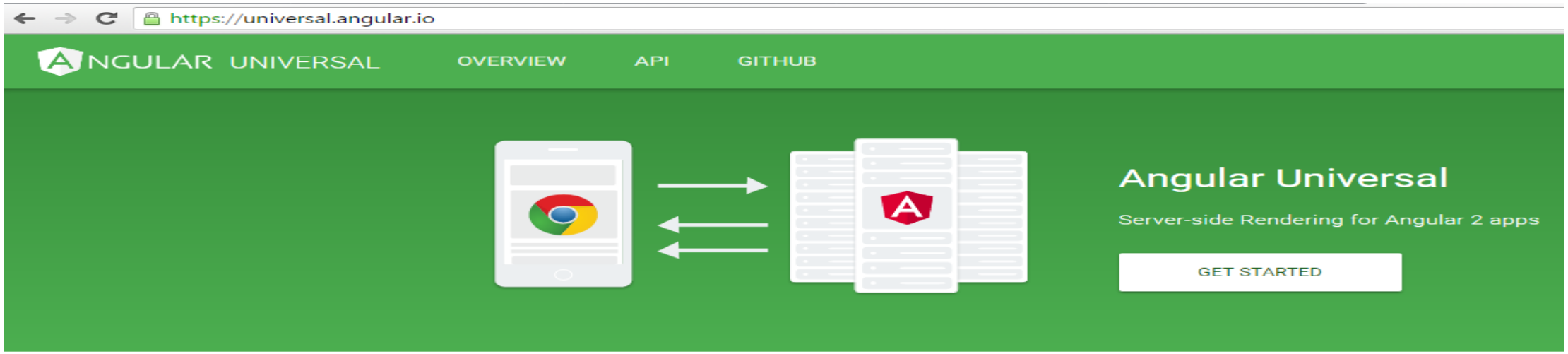


Angular Material

- Angular Material provides a set of reusable, well-tested, and accessible UI components based on Material Design.
- Angular Material v2 development is in-progress at the angular/material2 GitHub repository.



Angular Universal

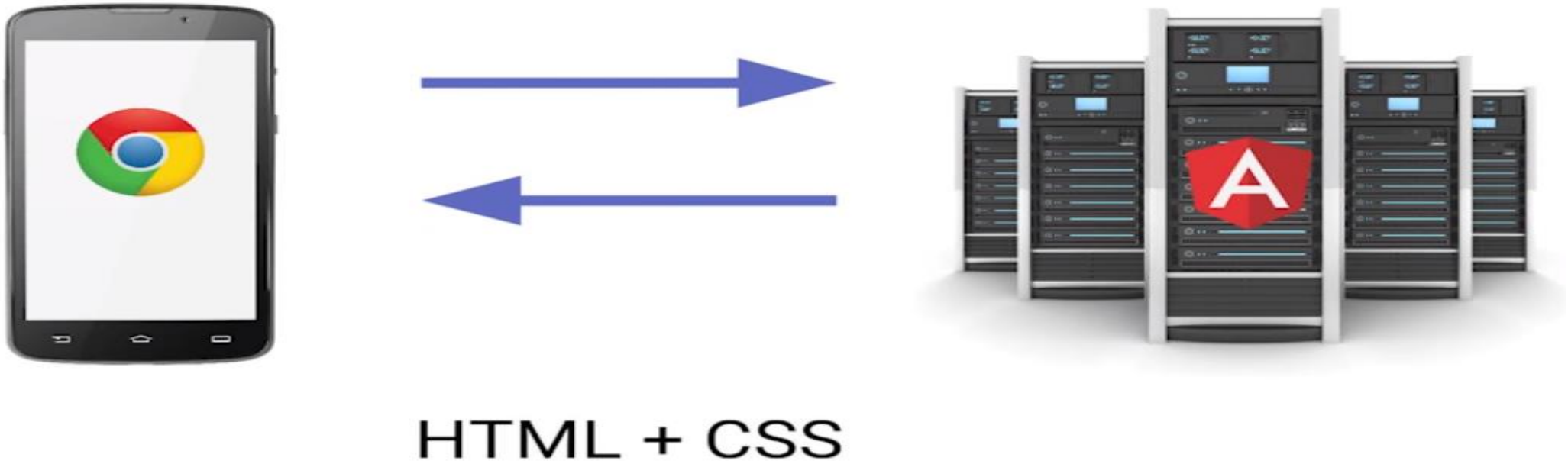


Better Perceived Performance

First time users of your application will instantly see a server rendered view which greatly improves perceived performance and the overall user experience. According to research at Google, the difference of just 200 milliseconds in page load performance has an impact on user behavior.

Initial Rendering: Universal

Good in 2G and slow networks



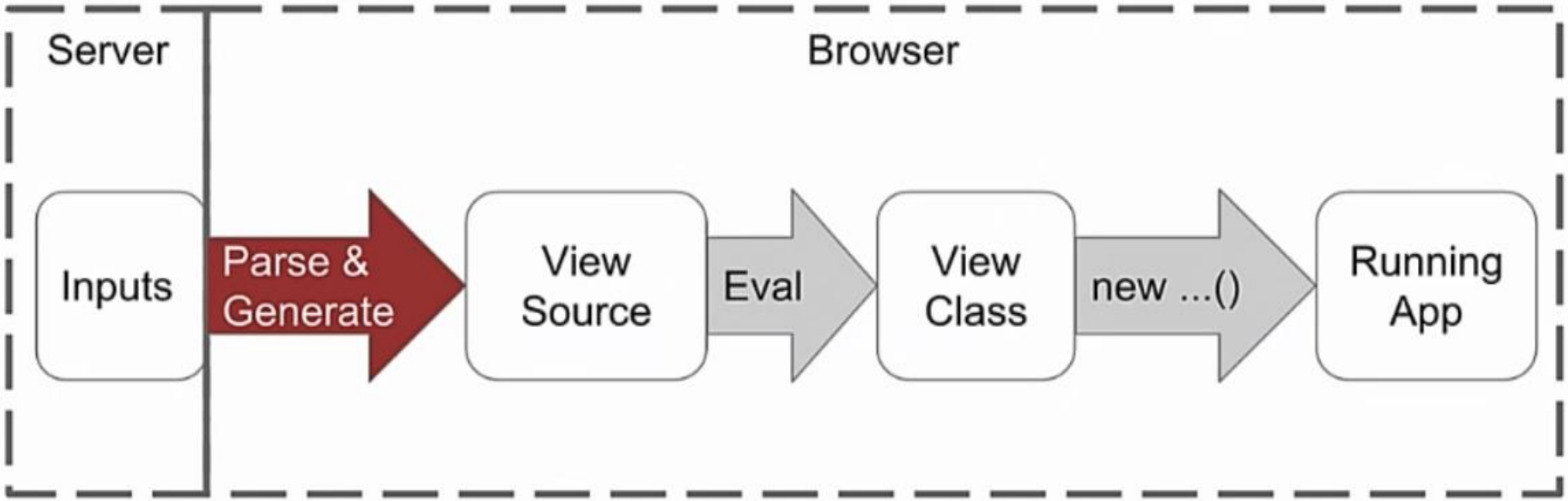
Initial Rendering: Universal

Node.js

ASP.NET

Others...

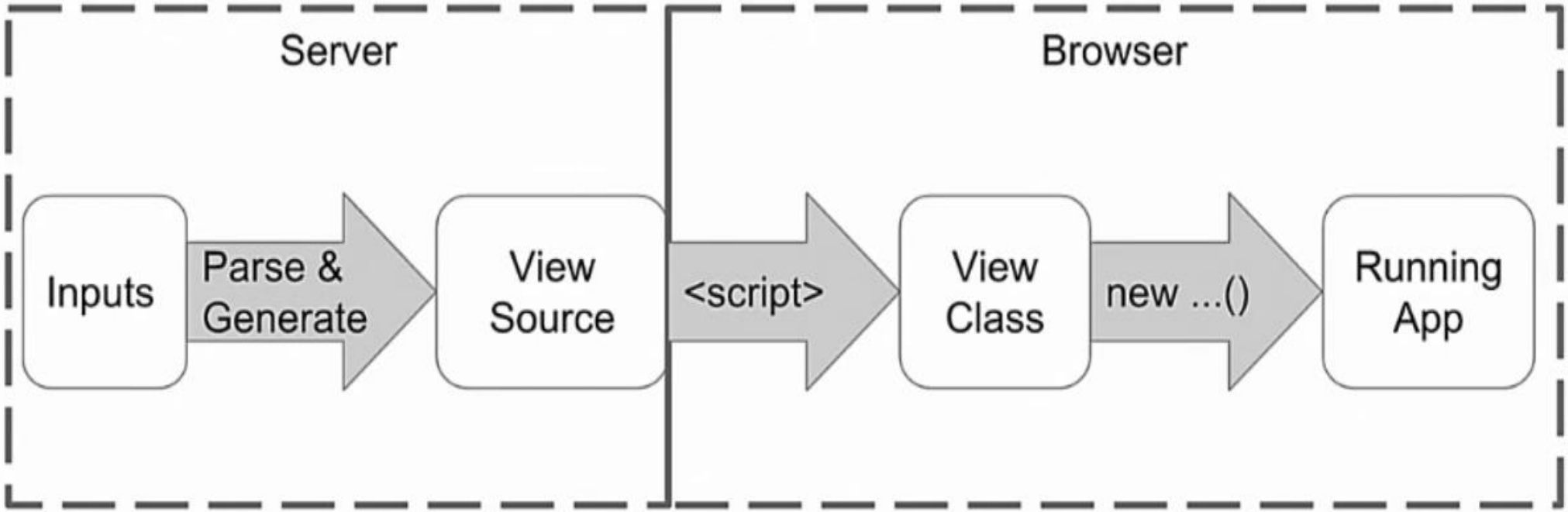
JIT Compilation



JIT Compilation Drawbacks

- We need to parse & generate in browser, it takes time depending on how many components we have
- It should go character by character
- The angular compiler needs to be in browser
- Using Eval application may be hijacked
- Some servers don’t allow browser to run Eval so they may stop angular

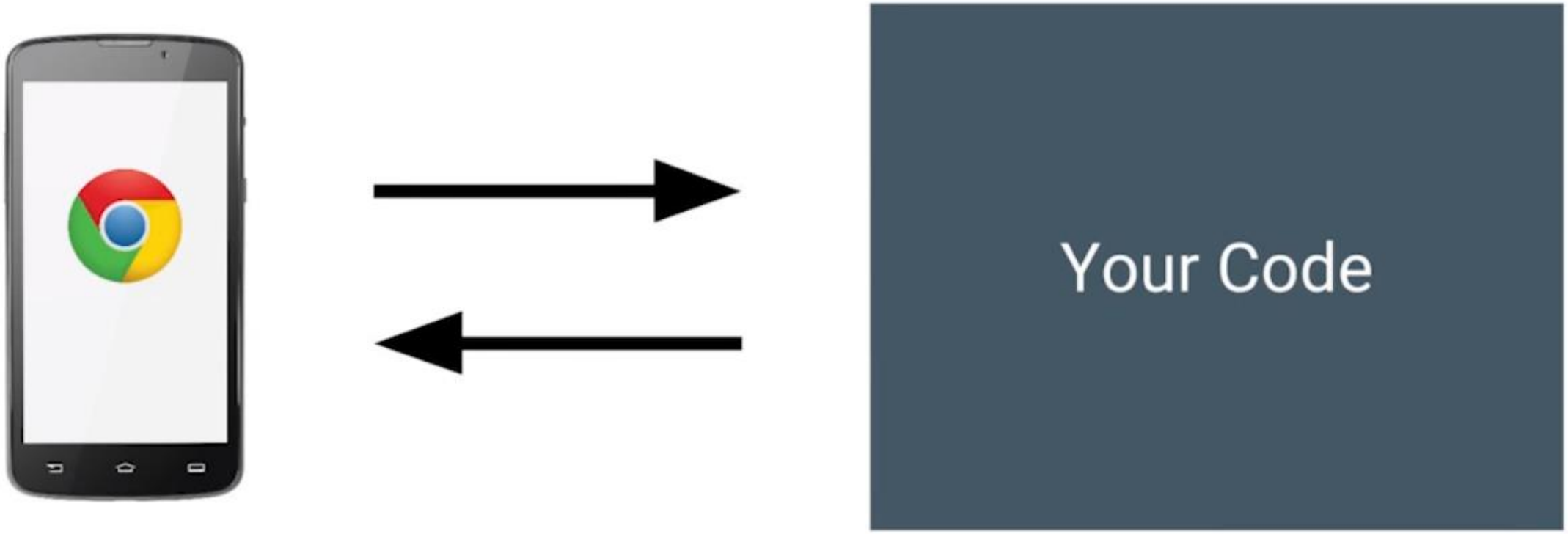
Ahead Of Time Compilation



Benefits Of AOT Compilation

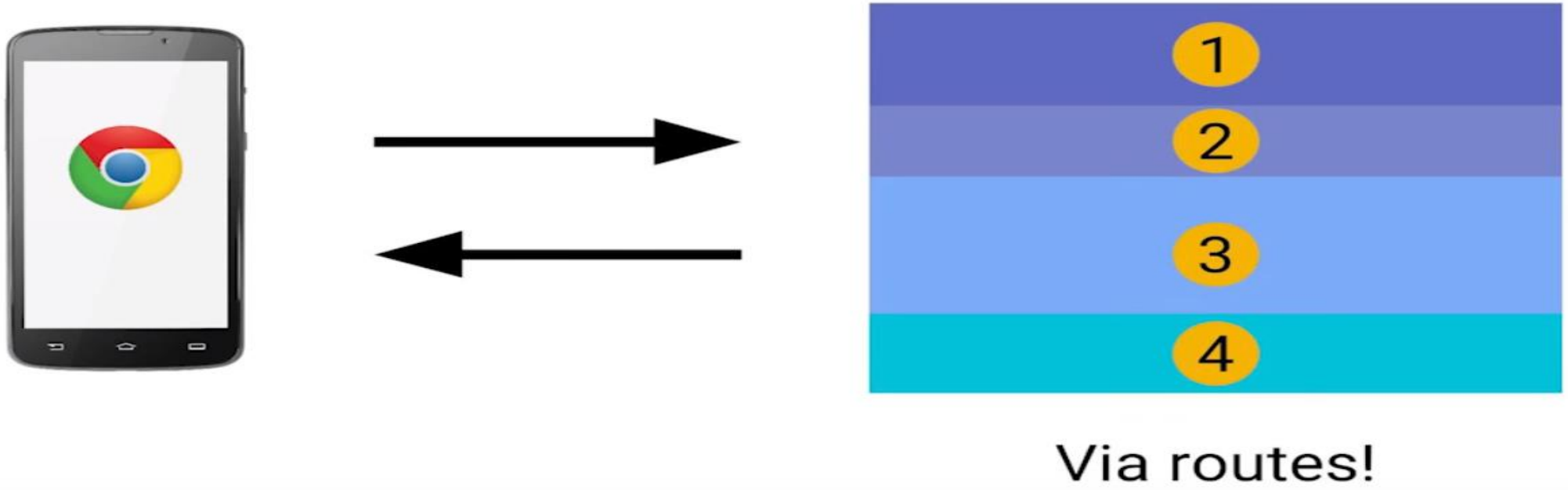
- Parse & Generate on the Server
- Generate View Classes on the Server
- Browser picks them up loads them as script tags
- Enhanced minification

Automatic Lazy Loading



Automatic Lazy Loading

- Let the router manage lazy loading
- When user selects route 1 load bits of route 1 and route 2 load bits of route 2 and so on…



Simplicity 1 of 2

Angular 1	Angular 2
43 directives	[ ] and ( )

Simplicity 2 of 2

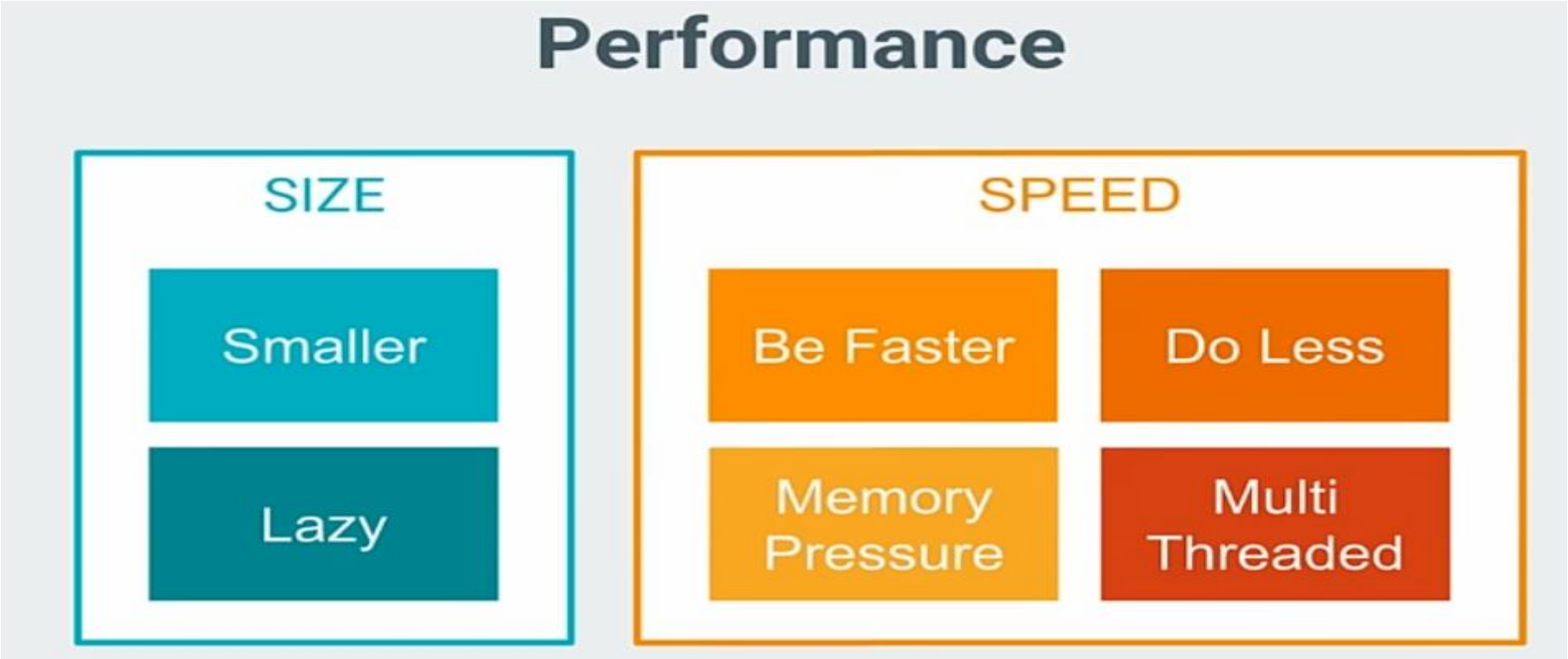
- AngularJS 1.x
  -
  - <button ng-click="do()"></button>
  - <input ng-model="firstName"/>

- Angular 2 and above
  -
  - <button (click)="do()"></button>
  - <input [(ngModel)]="firstName" />

Angular 2, directly uses the valid HTML DOM element properties and events.

Performance

Performance



Dependency Injection 1 of 2

Improved Dependency Injection

```
static $inject = ['app.dataServices.DocumentDataService', 'app.businessServices.SelectedBusinessService',
    'app.blocks.URLBuilderService', '$location', 'app.config.appConfig', '$stateParams', '$scope',
    'app.blocks.KendoGridHelperService', 'app.blocks.LoadingSpinnerHelper', '$http', 'app.blocks.AdalSecurityInterceptor',
    'app.blocks.AlertsService', 'app.blocks.superIpModal', 'app.businessServices.DownloadService',
    'app.dataServices.TagDataService'];
```

Dependency Injection 2 of 2

AngularJS 1.x

```
var myApp = angular
.module("myModule", [])
  .controller("productController",
function($scope, $http) {
      //code goes here
   });
```

Angular 2 and above

```
import { Injectable } from
'angular2/core';
@Injectable()
export class ProductService {
   constructor(private _http: Http) {
}
];
   }
}
```

Modules

- Angular 2 uses ES2015 Modules
- Lazy loading of modules will boost performance

Choice of languages

- Angular 4 provides more choice for languages. You can use any of the languages from **ES5, ES6**, **TypeScript** or **Dart** to write Angular 2 code.
- Using of TypeScript is a great step as TypeScript is an awesome way to write JavaScript.

Angular 4 implements web standards

- Angular 4 implements **web standards** like **components,** and it provides better performance than Angular 1.

Choice of languages

- Angular 2 provides more choice for languages. You can use any of the languages from **ES5, ES6**, **TypeScript** or **Dart** to write Angular 2 code.
- Using of TypeScript is a great step as TypeScript is an awesome way to write JavaScript.

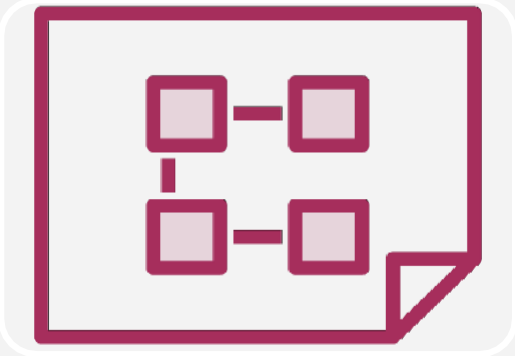
Angular 4 implements web standards

- Angular 4 implements **web standards** like **components,** and it provides better performance than Angular 1.

Why Angular?

Expressive HTML

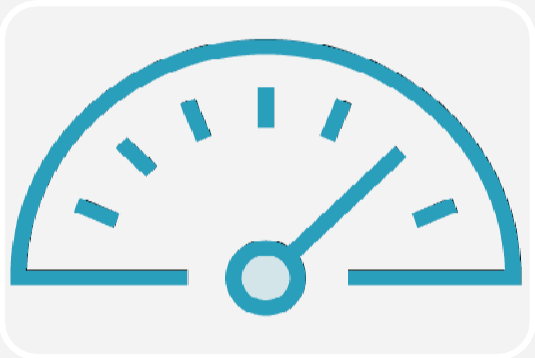
Powerful Data Binding



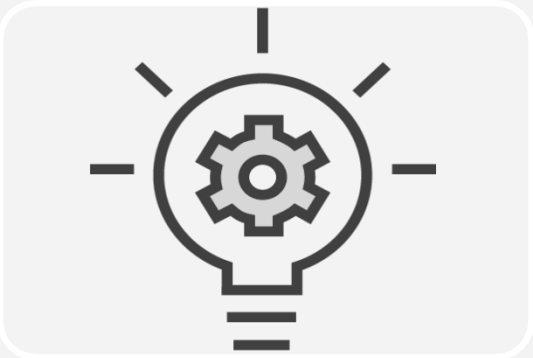
Modular By Design

Built-in Back-End Integration

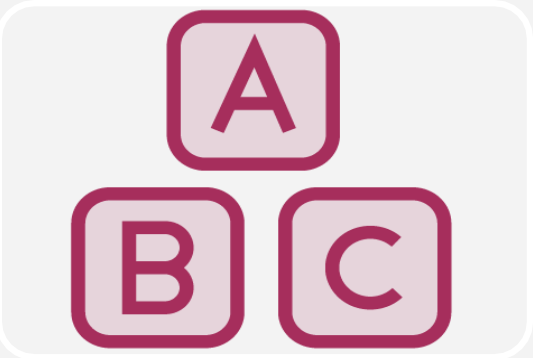
Why Angular 4?



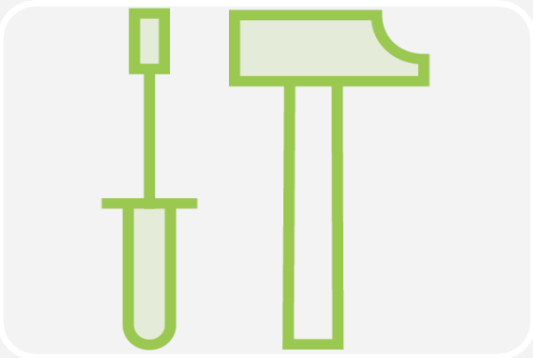
Built for Speed



Modern

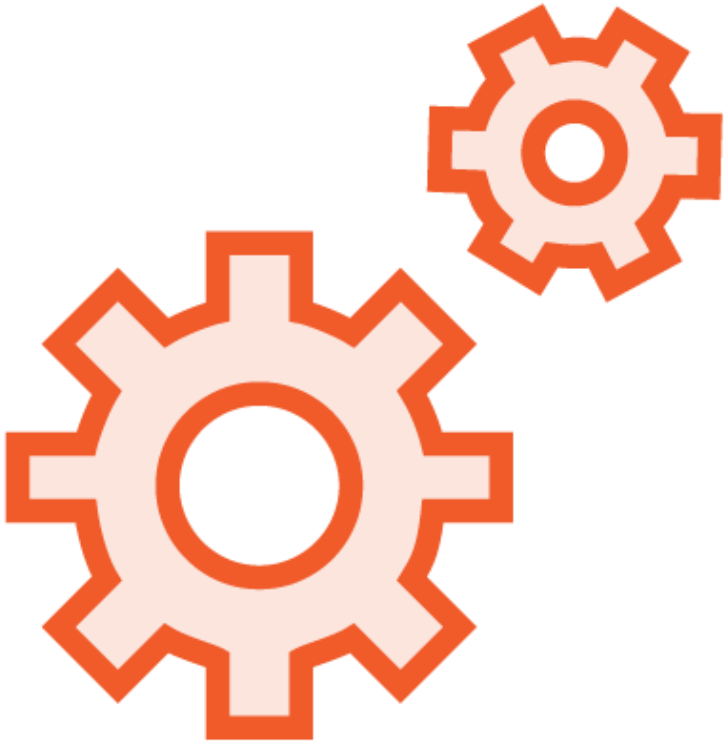


Simplified API



Enhances Productivity

Setting up Our Environment

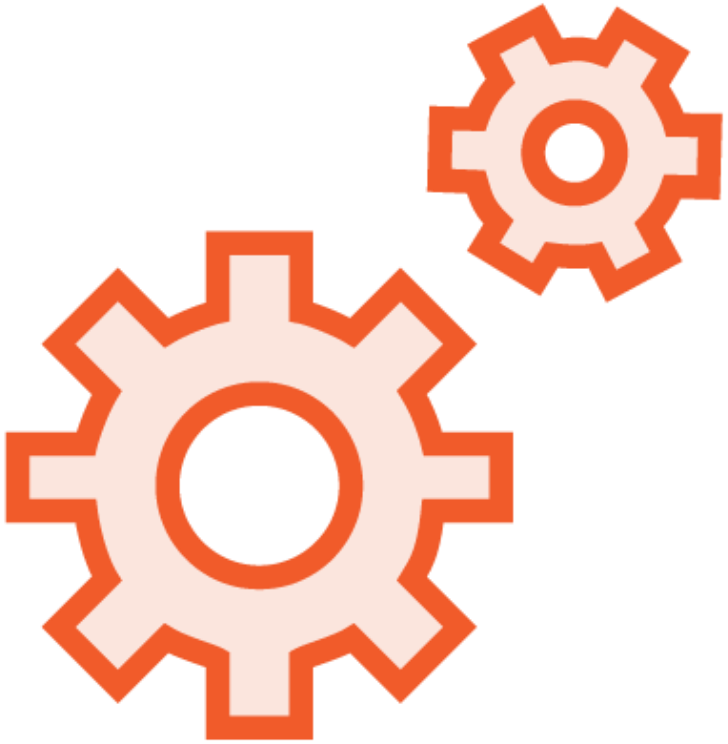


NodeJS

npm

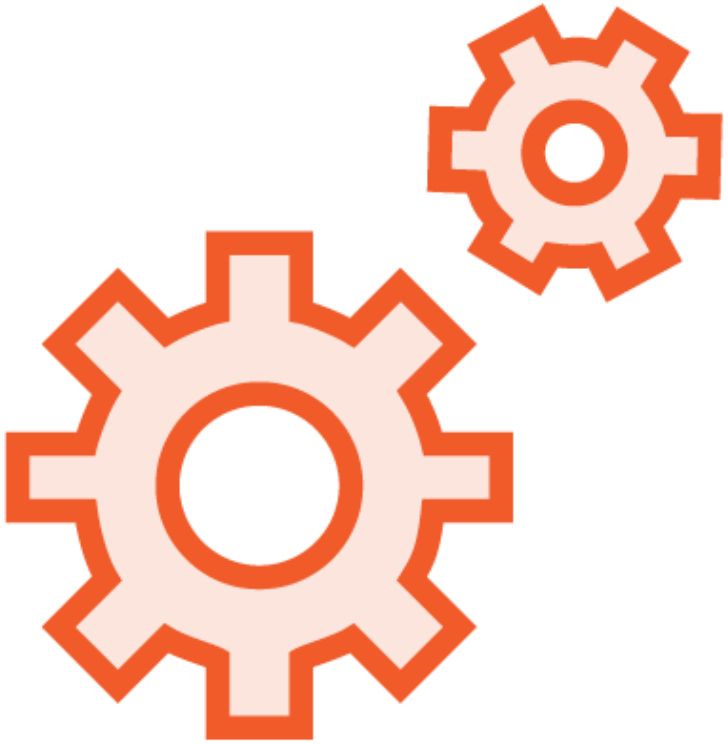
Set up the Angular 2 application

Setting up an Angular 4 Application



1. Create an application folder
2. Add package definition and configuration files
3. Install the packages
4. Create the app's Angular Module
5. Create the main.ts file
6. Create the host Web page (index.html)

Setting up an Angular 4 Application



Manually perform each step

www.angular.io Quick Start

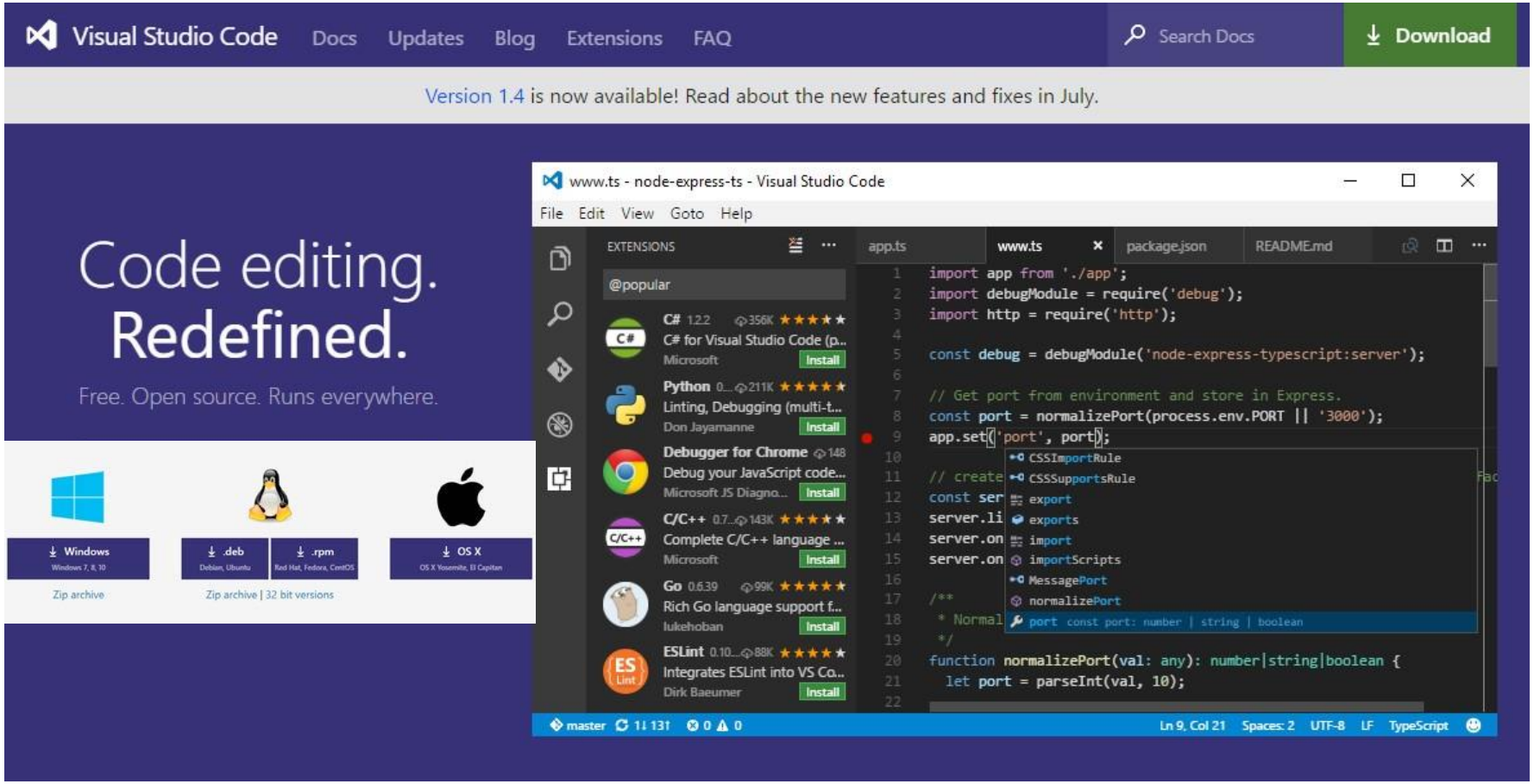
Download the results of these steps

https://github.com/angular/quickstart

Editors

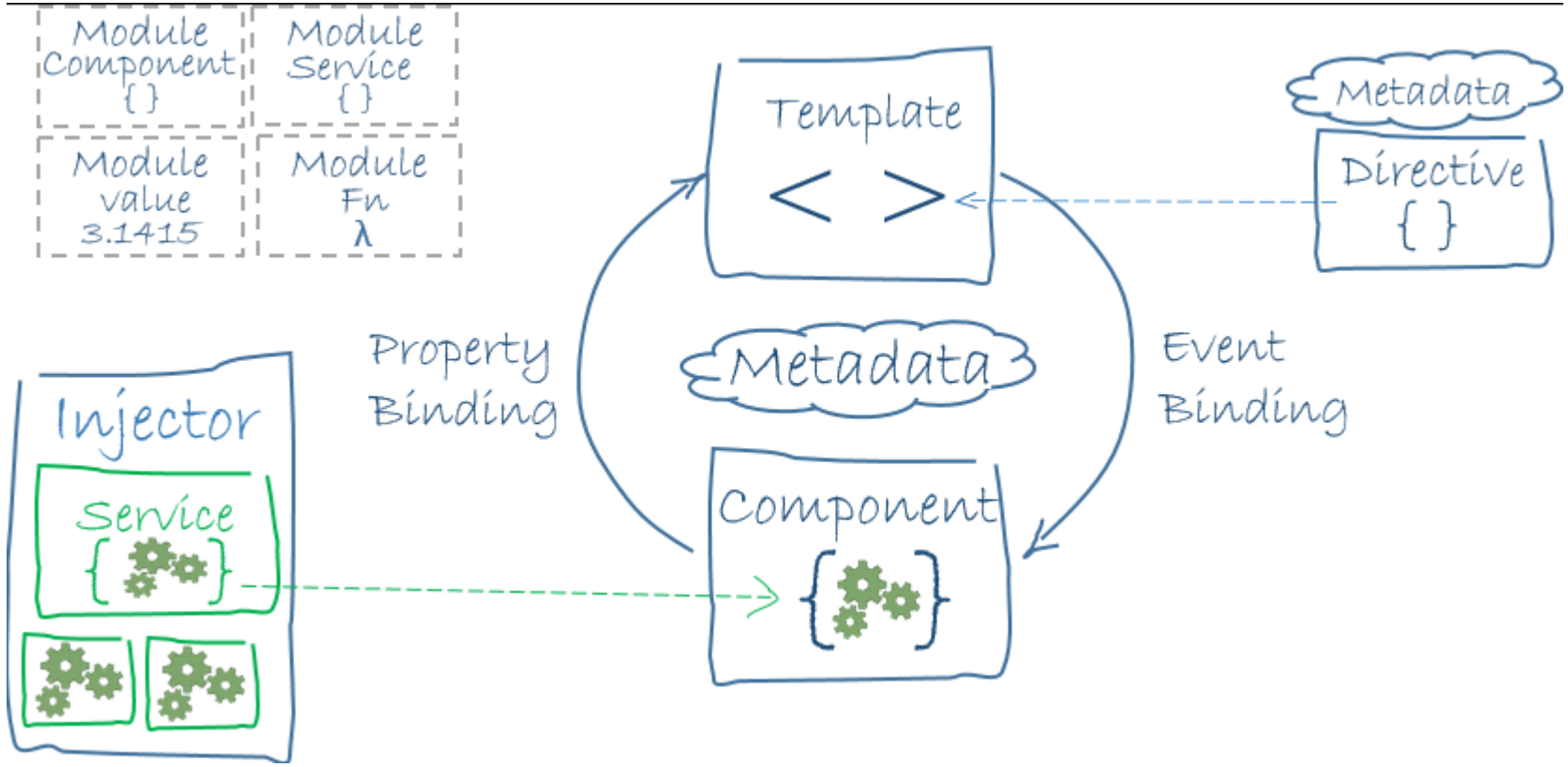
- Visual Studio
- Visual Studio Code
- WebStorm
- Atom
- Eclipse
- Others

Visual Studio Code



https://code.visualstudio.com/

Angular Architecture



What Is an Angular Module?

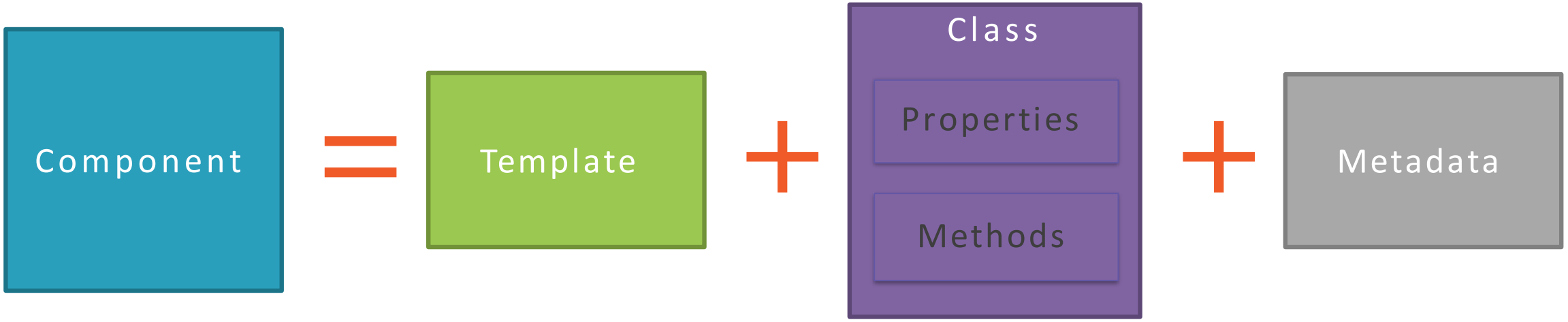
Module

A class with an NgModule decorator

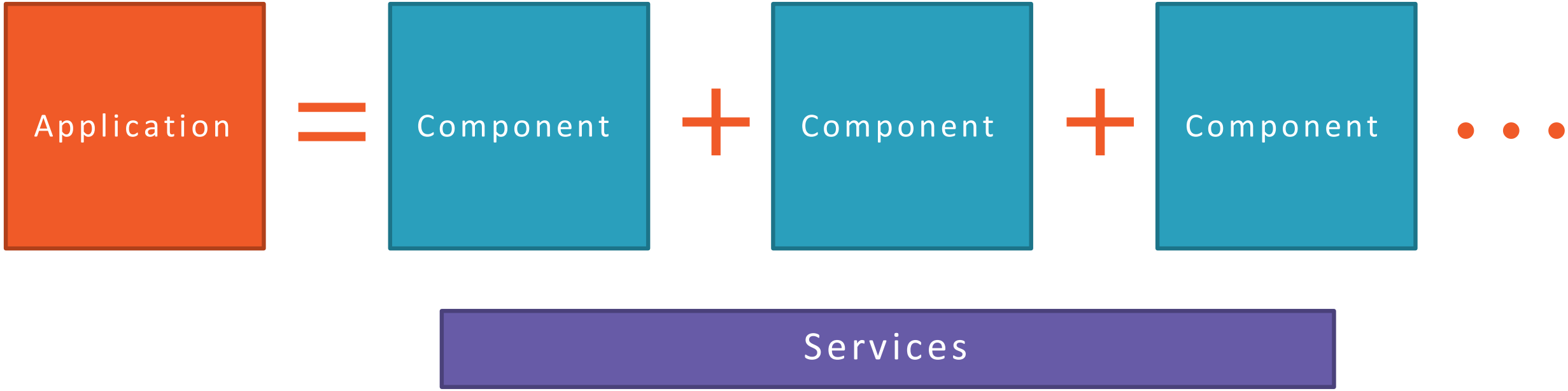
Its purpose:

- Organize the pieces of our application
- Arrange them into blocks
- Extend our application with capabilities from external libraries
- Provide a template resolution environment
- Aggregate and re-export

Component



Anatomy of an Angular 4 Application



Components

What Is a Component?

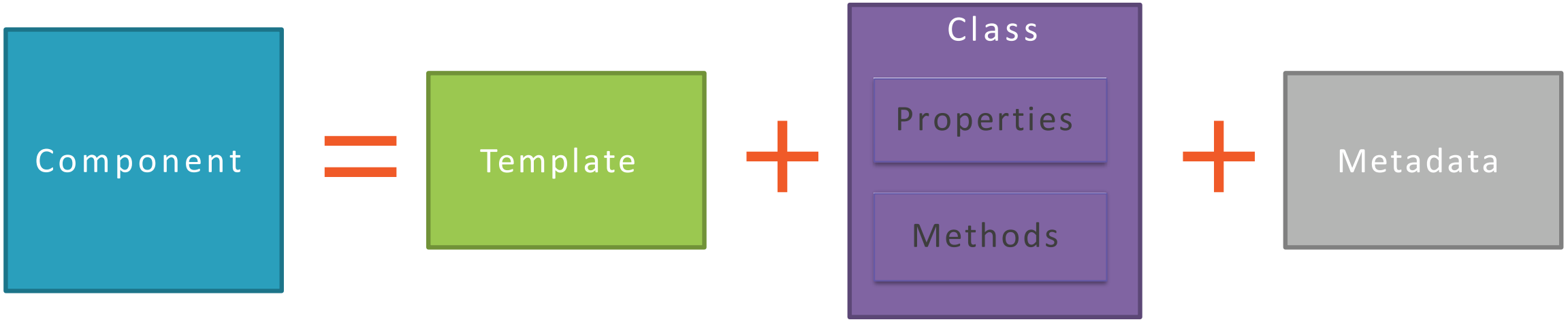
Creating the Component Class

Defining the Metadata with a Decorator

Importing What We Need

Bootstrapping Our App Component

What Is a Component?



- View layout
- Created with HTML
- Includes binding and directives

- Code supporting the view
- Created with TypeScript
- Properties: data
- Methods: logic

- Extra data for Angular
- Defined with a decorator

Component

app.component.ts

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

@Component({
    selector: 'pm-app',
    template: `
      <div><h1>{{pageTitle}}</h1>
        <div>My First Component</div>
      </div>
    `
})
export class AppComponent {
 pageTitle: string = 'Acme Product Management';
}
```

Import

Metadata & Template

Class

Creating the Component Class

app.component.ts

```
export class AppComponent {
    pageTitle: string = 'Acme Product Management';
}
```

class keyword

Class Name

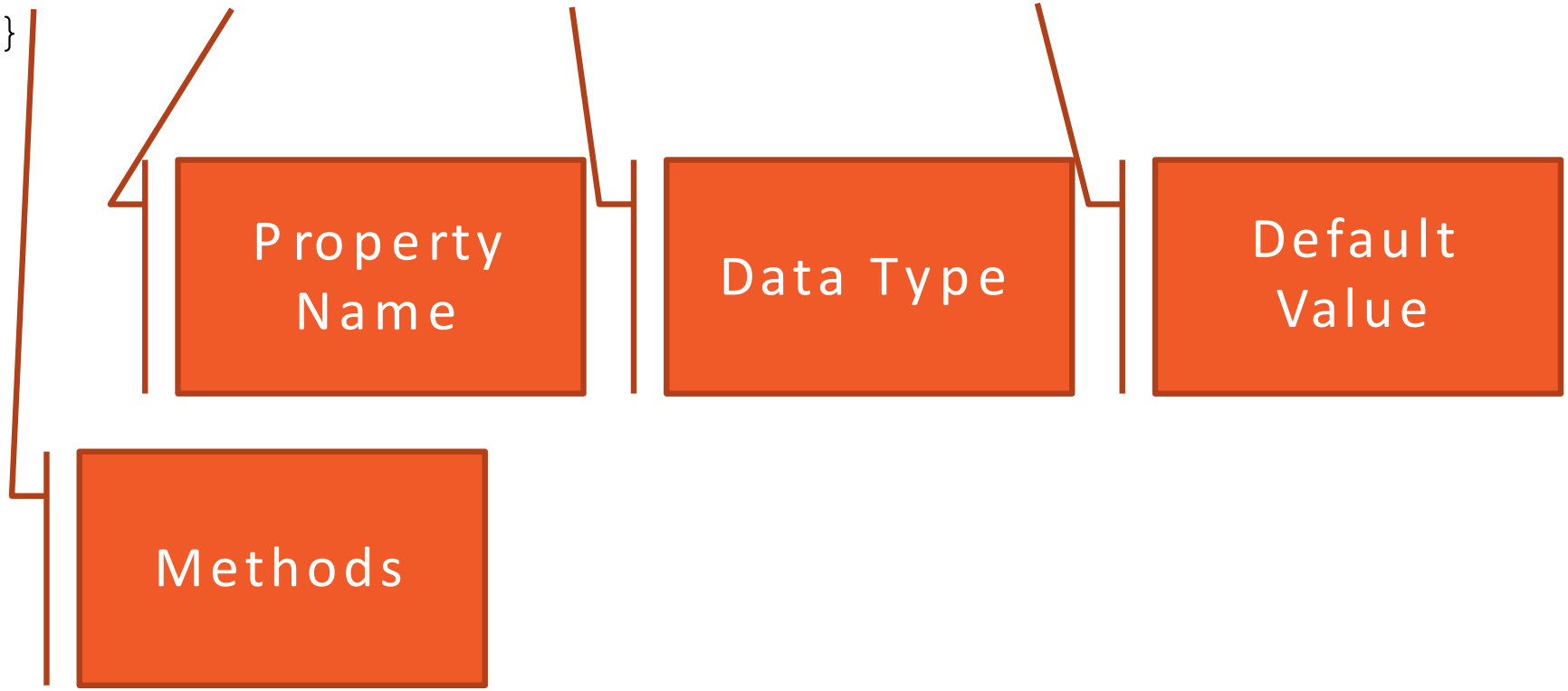
export keyword

Component Name when used in code

Creating the Component Class

app.component.ts

```
export class AppComponent {
    pageTitle: string = 'Acme Product Management';
}
```



Defining the Metadata

app.component.ts

```
@Component({
    selector: 'pm-app',
    template: `
    <div><h1>{{pageTitle}}</h1>
        <div>My First Component</div>
    </div>
    `
})
export class AppComponent {
 pageTitle: string = 'Acme Product Management';
}
```

Decorator

A function that adds metadata to a class, its members, or its method arguments.

Prefixed with an @.

Angular provides built-in decorators.

```
@Component()
```

Defining the Metadata

app.component.ts

```
@Component({
    selector: 'pm-app',
    template: `
    <div><h1>{{pageTitle}}</h1>
        <div>My First Component</div>
    </div>
    `
})
export class AppComponent {
 pageTitle: string = 'Acme Product Management';
}
```

Component decorator

Directive Name used in HTML

View Layout

Binding

Importing What We Need

Before we use an external function or class, we define where to find it

`import` statement

`import` allows us to use exported members from external ES modules

Import from a third-party library, our own ES modules, or from Angular

Angular IsModular

@angular/core

@angular/animate

@angular/http

@angular/router

https://www.npmjs.com/~angular

Importing What We Need

app.component.ts

```
@Component({
    selector: 'pm-app',
    template: `
    <div><h1>{{pageTitle}}</h1>
        <div>My First Component</div>
    </div>
    `
})
export class AppComponent {
 pageTitle: string = 'Acme Product Management';
}
```

Importing What We Need

app.component.ts

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

@Component({
    selector: 'pm-app',
    template: `
    <div><h1>{{pageTitle}}</h1>
        <div>My First Component</div>
    </div>
    `
})
export class AppComponent {
 pageTitle: string = 'Acme Product Management';
}
```

import keyword

Angular library module name

Member name

Completed Component

app.component.ts

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

@Component({
    selector: 'pm-app',
    template: `
    <div><h1>{{pageTitle}}</h1>
        <div>My First Component</div>
    </div>
    `
})
export class AppComponent {
 pageTitle: string = 'Acme Product Management';
}
```

Bootstrapping App Component

Load the root component

(bootstrapping)

Host the application

Files and purpose

File	Purpose
app/app.component.ts	It is the **root** component of what will become a tree of nested components as the application evolves.
app/app.module.ts	Defines AppModule, the root module that tells Angular how to assemble the application
main.ts	Compiles the application with the JIT compiler and bootstraps the application's main module (AppModule) to run in the browser.

Bootstrapping Angular App

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule }              from './app/app.module';
platformBrowserDynamic().bootstrapModule(AppModule);
```

- This code creates a browser platform for dynamic (JIT) compilation and bootstraps the `AppModule` described above.

- The bootstrapping process sets up the execution environment, digs the root AppComponent out of the module's bootstrap array, creates an instance of the component and inserts it within the element tag identified by the component's selector.

Single Page Application (SPA)

- `index.html` contains the main page for the application
- This is often the only Web page of the application
- Hence an Angular application is often called a Single Page Application (SPA)

Hosting the Application

index.html

```
<body>
  <pm-app>Loading App ...</pm-app>
</body>
```

app.component.ts

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

@Component({
    selector: 'pm-app',
    template: `
    <div><h1>{{pageTitle}}</h1>
        <div>My First Component</div>
    </div>
    `
})
export class AppComponent {
 pageTitle: string = 'Acme Product Management';
}
```

Angular Application Startup

index.html

```
System.import('app')...;
```

```
<body>
  <pm-app>Loading App ...
  </pm-app>
</body>
```

Systemjs.config.js

```
packages: {
 app: {
  main: './main.js',
  defaultExtension: 'js'
 },
...
```

```
import { platformBrowserDynamic }
 from '@angular/platform-browser-dynamic';
import { AppModule }
 from './app.module';

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

app.component.ts

```
...
@Component({
 selector: 'pm-app',
 template: `
 <div>{{pageTitle}}</div>
 `
})
export class AppComponent {
...
}
```

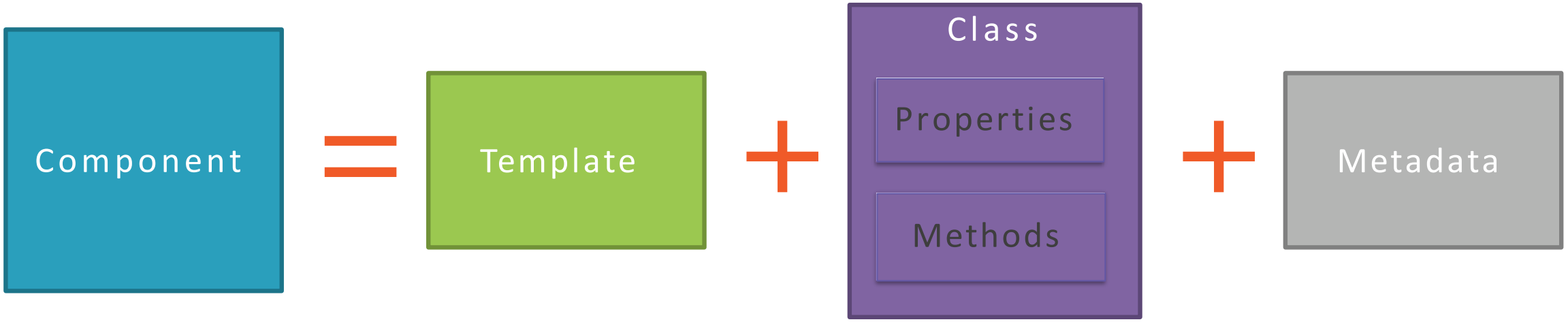
app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent }  from './app.component';

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

Templates, Interpolation, and Directives

Component



Module Overview

Building a Template

Using a Component as a Directive

Binding with Interpolation

Adding Logic with Directives

Component

app.component.ts

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

@Component({
    selector: 'pm-app',
    template: `
    <div><h1>{{pageTitle}}</h1>
        <div>My First Component</div>
    </div>
    `
})
export class AppComponent {
 pageTitle: string = 'Acme Product Management';
}
```

Defining a Template in a Component

Inline Template

```
template:
"<h1>{{pageTitle}}</h1>"
```

Inline Template

```
template: `
<div>
 <h1>{{pageTitle}}</h1>
 <div>
     My First Component
 </div>
</div>
`
```

ES 2015
Back Ticks

Linked Template

```
templateUrl:
'product-list.component.html'
```

Building the Component

product-list.component.ts

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

@Component({
    selector: 'pm-products',
    templateUrl: 'app/products/product-list.component.html'
})
export class ProductListComponent {
 pageTitle: string = 'Product List';
}
```

Binding

Coordinates communication between the component's class and its template and often involves passing data.

Interpolation

Template

```
<h1>{{pageTitle}}</h1>

{{'Title: ' + pageTitle}}

{{2*20+1}}

{{'Title: ' + getTitle()}}

<h1 innerText={{pageTitle}}></h1>
```

Class

```
export class AppComponent {
 pageTitle: string =
     'Acme Product Management';
}getTitle(): string {...};
}
```

Directive

Custom HTML element or attribute used to power up and extend our HTML.

- Custom
- Built-In

Custom Directives

app.component.ts

```
@Component({
  selector: 'pm-app',
  template: `
    <div><h1>{{pageTitle}}</h1>
      <pm-products></pm-products>
    </div>
    `
})
export class AppComponent { }
```

product-list.component.ts

```
@Component({
  selector: 'pm-products',
  templateURL:
    'app/products/product-list.component.html'
})
export class ProductListComponent { }
```

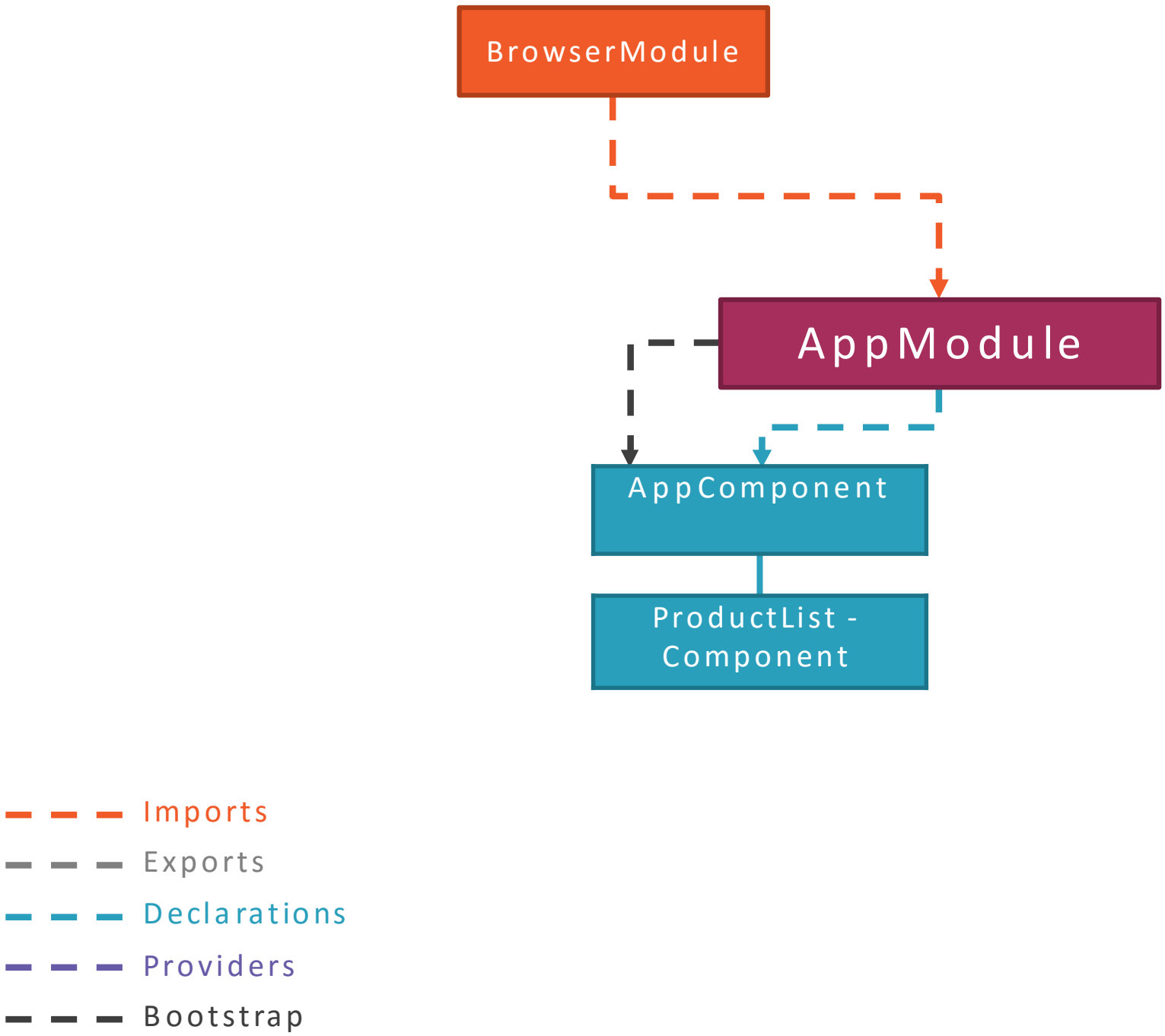
Angular Built-in Directives

Structural Directives

- `*ngIf`: If logic
- `*ngFor`: For loops

*ngIf Built-In Directive

```
<div class='table-responsive'>
 <table class='table' *ngIf='products && products.length'>
  <thead> ...
  </thead>
  <tbody> ...
  </tbody>
 </table>
</div>
```


BrowserModule
AppModule
AppComponent
ProductList - Component
Imports
Exports
Declarations
Providers
Bootstrap

*ngFor Built-In Directive

Template input variable

```
• <tr *ngFor='let product of products'>
    • <td></td>
    • <td>{{ product.productName }}</td>
    • <td>{{ product.productCode }}</td>
    • <td>{{ product.releaseDate }}</td>
    • <td>{{ product.price }}</td>
    • <td>{{ product.starRating }}</td>
• </tr>
```

for…of vs for…in

- Iterates over iterable objects, such as an array.
- Result: di, boo, punkeye

```
let nicknames= ['di', 'boo', 'punkeye'];

for (let nickname of nicknames) {
  console.log(nickname);
}
```

- for…in
- Iterates over the properties of an object.
- Result: 0, 1, 2

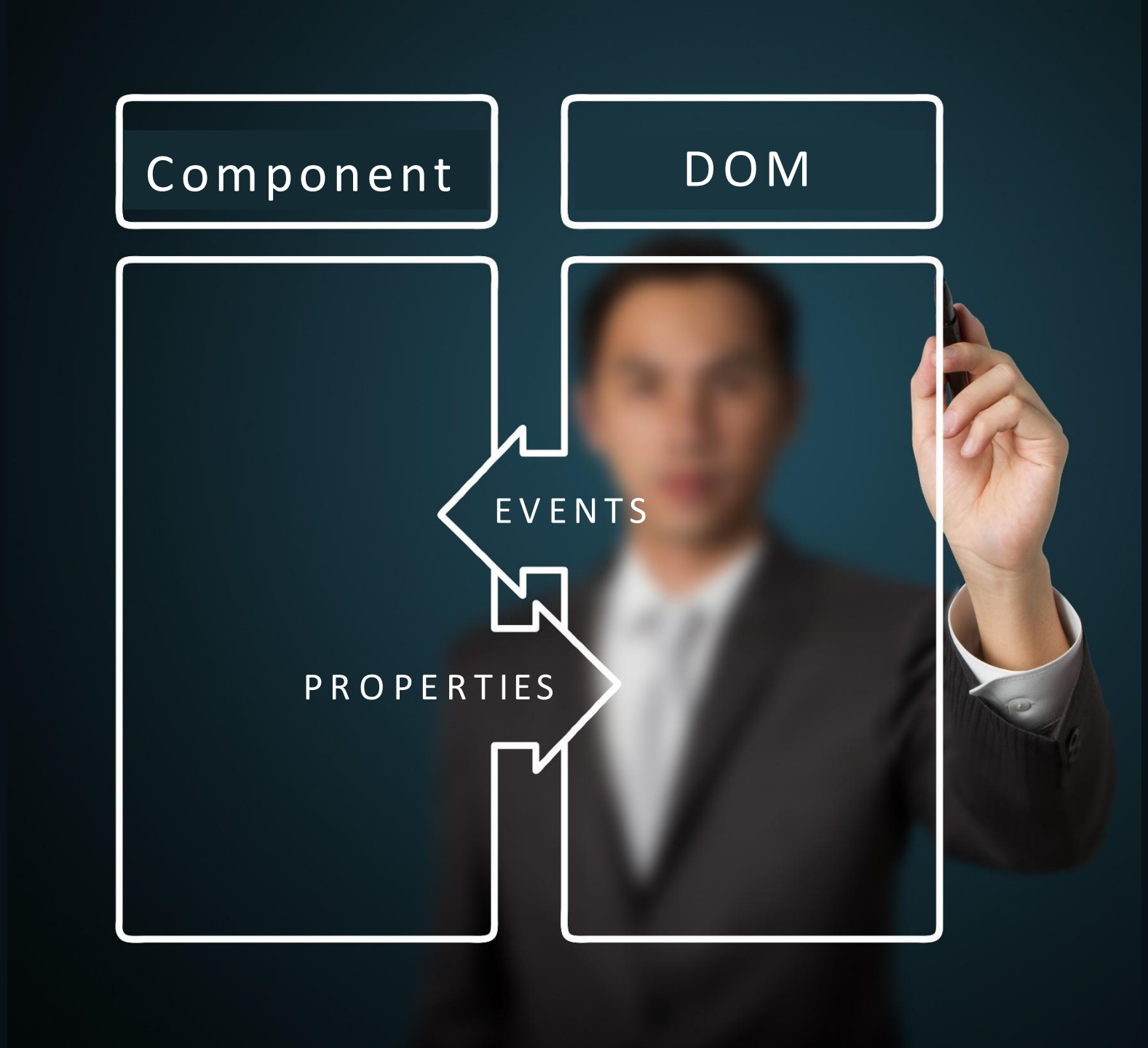
```
let nicknames= ['di', 'boo', 'punkeye'];

for (let nickname in nicknames) {
  console.log(nickname);
}
```

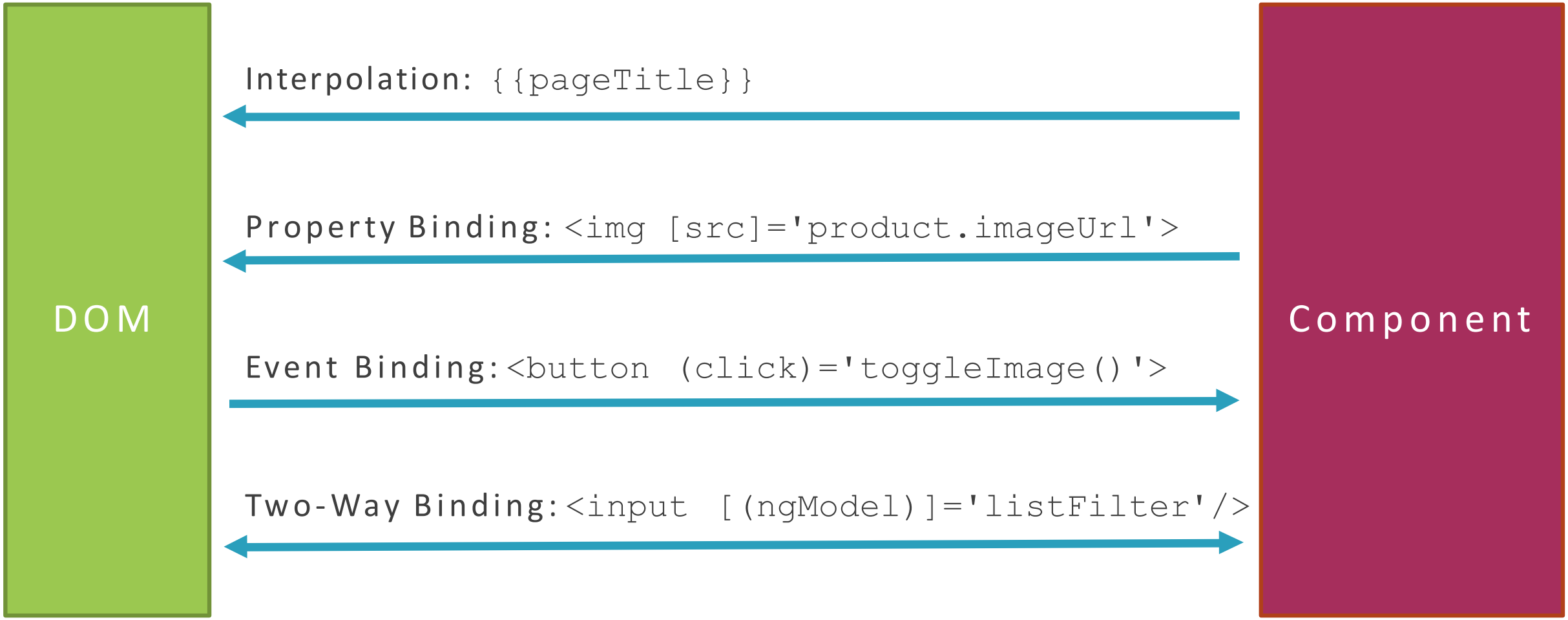
*ngFor Built-In Directive

- `<tr *ngFor='let product of products'>`
  - `<td></td>`
  - `<td>{{ product.productName }}</td>`
  - `<td>{{ product.productCode }}</td>`
  - `<td>{{ product.releaseDate }}</td>`
  - `<td>{{ product.price }}</td>`
  - `<td>{{ product.starRating }}</td>`
- `</tr>`

Data Binding & Pipes


Component
DOM
EVENTS
PROPERTIES

Data Binding



Transforming Data with Pipes

Transform bound properties before display

Built-in pipes

- date
- number, decimal, percent, currency
- json, slice
- etc

Custom pipes

Pipe Examples

```
{{ product.name | uppercase }}

<img [src]='product.imageUrl'
     [title]='product.productName | uppercase'>

{{ product.price | currency | lowercase }}
```

Parameterizing a pipe

- <td> {{ mfg | date:“medium" }} </td>

Chaining pipes

- {{ mfg | date | uppercase}}

Decimal Pipes

- number_expression | number[:digitInfo]

- digitInfo is a string which has a following format:
  {minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}
  minIntegerDigits Defaults to 1.
  minFractionDigits Defaults to 0.
  maxFractionDigits Defaults to 3

- **Example: <td>{{product.rating | number:’1.1-2’}}</td>**

No FilterPipe or OrderByPipe

- Angular doesn't provide pipes for filtering or sorting lists. Developers familiar with AngularJS know these as filter and orderBy.
- There are no equivalents in Angular.
- Angular doesn't offer such pipes because they perform poorly and prevent aggressive minification.
- Both filter and orderBy require parameters that reference object properties.
- These pipes must be impure and that Angular calls impure pipes in almost every change-detection cycle.

Custom Pipes

- A pipe is a class decorated with pipe metadata.
- The pipe class implements the PipeTransform interface's transform method that accepts an input value followed by optional parameters and returns the transformed value.

The *PipeTransform* interface

The transform method is essential to a pipe. The PipeTransform interface defines that method and guides both tooling and the compiler.

Summary

Property Binding

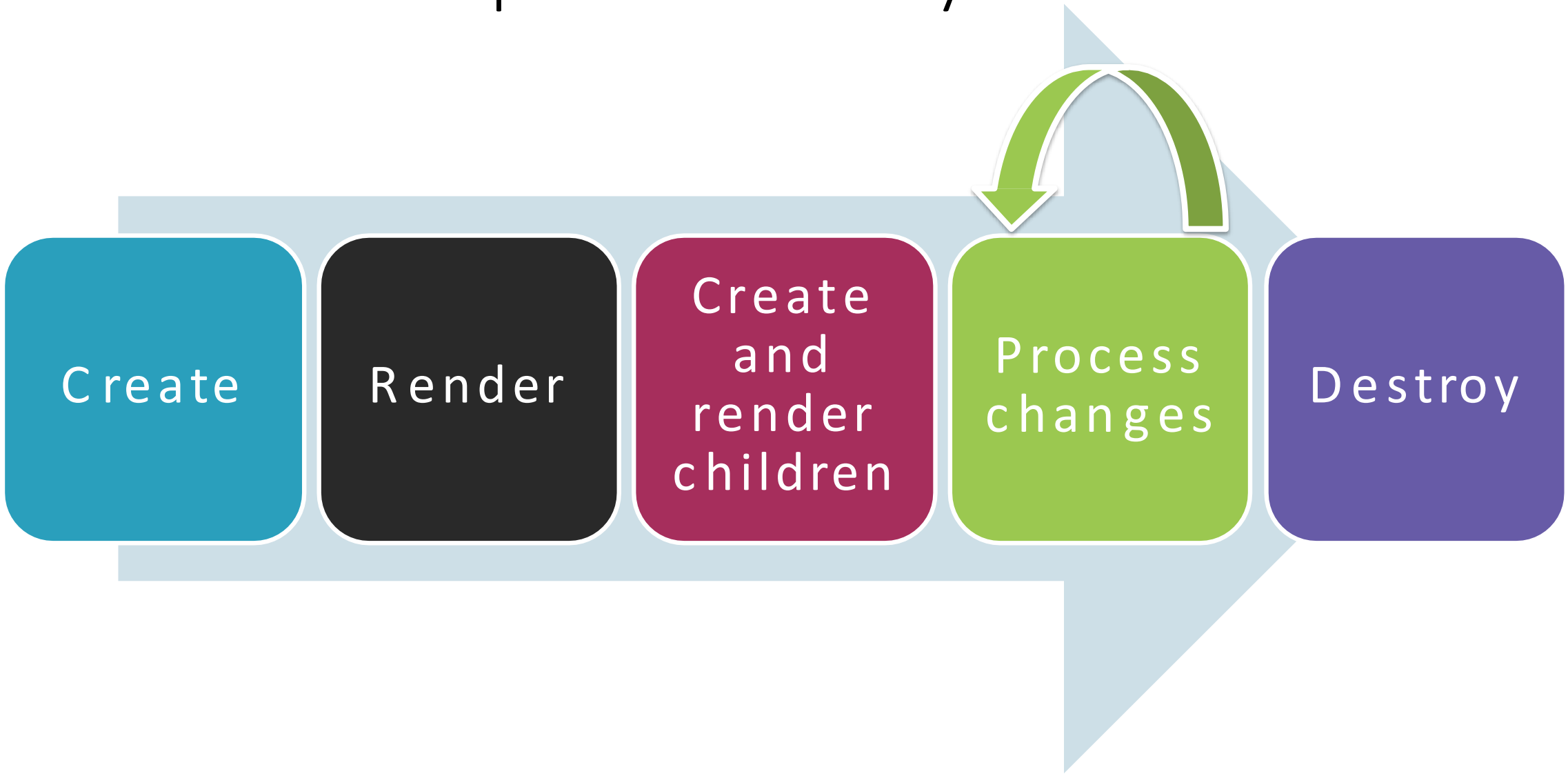
Handling Events with Event Binding

Handling Input with Two-way Binding

Transforming Data with Pipes

More on Components

Component Lifecycle



Component Lifecycle Hooks

OnInit: Perform component initialization, retrieve data

OnChanges: Perform action after change to input properties

OnDestroy: Perform cleanup

Using a Lifecycle Hook

2

1

```
export class ProductListComponent
                    implements OnInit {
  pageTitle: string = 'Product List';
  showImage: boolean = false;
  listFilter: string = 'cart';
  products: IProduct[] = […];
```

3

```
}
```

Services and Dependency Injection

Service

A class with a focused purpose.

Used for features that:

- Are independent from any particular component
- Provide shared data or logic across components
- Encapsulate external interactions

How Does It Work?

Building a Service

Registering the Service

Injecting the Service

How Does It Work?

Service

```
export class myService {}
```

Component

```
let svc = new myService();
```

svc

Dependency Injection

A coding pattern in which a class receives the instances of objects it needs (called dependencies) from an external source rather than creating them itself.

Steps in creating and using services

- Build a service
- Register a service
- Inject a service

Building a Service

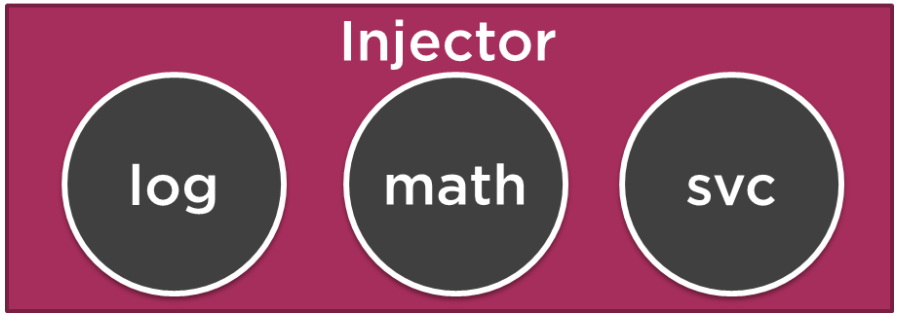
product.service.ts

```
import { Injectable } from '@angular/core'

@Injectable()
export class ProductService {
  getProducts(): IProduct[] {
  return [{
   id:1,
   name: "pencil",
     }];
  }

}
```

Registering a Service



Register a provider

- Code that can create or return a service
- Typically the service class itself

Define in component OR Angular module metadata

Registered in component:

- Injectable to component AND its children

Registered in Angular module:

- Injectable everywhere in the application

Registering a Provider

app.component.ts

```
...
import { ProductService } from './products/product.service';

@Component({
  selector: 'pm-app',
  template: `
    <div><h1>{{pageTitle}}</h1>
      <pm-products></pm-products>
    </div>
    `,
  providers: [ProductService]
})
export class AppComponent { }
```

Injecting the Service

product-list.component.ts

```
...


@Component({
  selector: 'pm-products',
  templateUrl: 'product-list.component.html'
})
export class ProductListComponent {

 constructor() {
 }

}
```

Injecting the Service

product-list.component.ts

```
...
import { ProductService } from './products/product.service';

@Component({
  selector: 'pm-products',
  templateUrl: 'product-list.component.html'
})
export class ProductListComponent {
 private _productService;
 constructor(productService: ProductService) {
    _productService = productService;
 }

}
```

Injecting the Service

product-list.component.ts

```
...
import { ProductService } from './products/product.service';

@Component({
  selector: 'pm-products',
  templateUrl: 'product-list.component.html'
})
export class ProductListComponent {

 constructor(private _productService: ProductService) {
 }

}
```

Retrieving Data Using HTTP

Module Overview

Observables and Reactive Extensions

Sending an Http Request

Exception Handling

Subscribing to an Observable

Observables and Reactive Extensions

Help manage asynchronous data

Treat events as a collection

- An array whose items arrive asynchronously over time

Are a proposed feature for ES 2016

Use Reactive Extensions (RxJS)

Are used within Angular

Observable Operators

Methods on observables that compose new observables

Transform the source observable in some way

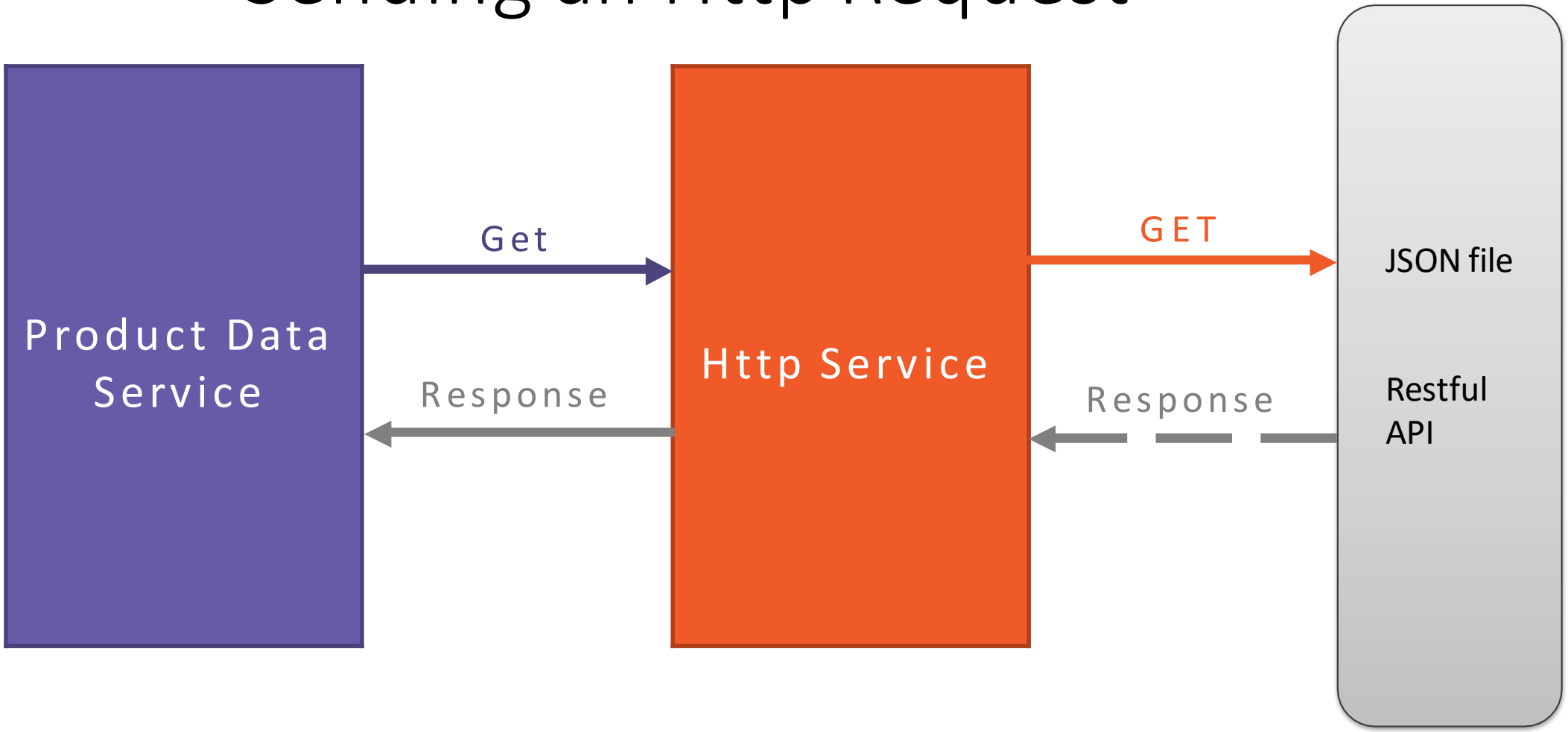
Process each value as it is emitted

Examples: map, filter, take, merge, …

Promise vs Observable

Promise	Observable
Provides a single future value	Emits multiple values over time
Not lazy	Lazy
Not cancellable	Cancellable
	Supports map, filter, reduce and other operators

Sending an Http Request



Sending an Http Request

product.service.ts

```
...
import { Http } from '@angular/http';



@Injectable()
export class ProductService {
  private _productUrl = 'api/products/products.json';

  constructor(private _http: Http) { }

  getProducts() {
   return this._http.get(this._productUrl);

  }
}
```

Registering the Http Service Provider

app.module.ts

```
...
import { HttpModule } from '@angular/http';

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

Sending an Http Request

product.service.ts

```
...
import { Http } from '@angular/http';



@Injectable()
export class ProductService {
  private _productUrl = 'api/products/products.json';

  constructor(private _http: Http) { }

  getProducts() {
   return this._http.get(this._productUrl);

  }
}
```

Sending an Http Request

product.service.ts

```
...
import { Http, Response} from '@angular/http';
import { Observable } from 'rxjs/Observable';


@Injectable()
export class ProductService {
  private _productUrl = 'api/products/products.json';

  constructor(private _http: Http) { }

  getProducts(): Observable<Response> {
   return this._http.get(this._productUrl);

  }
}
```

Sending an Http Request

product.service.ts

```
...
import { Http, Response} from '@angular/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';

@Injectable()
export class ProductService {
  private _productUrl = 'api/products/products.json';

  constructor(private _http: Http) { }

  getProducts(): Observable<IProduct[]> {
   return this._http.get(this._productUrl)
              .map((response: Response) => <IProduct[]>response.json());
  }
}
```

Exception Handling

- product.service.ts

```
• ...
•import 'rxjs/add/operator/do';  import
'rxjs/add/operator/catch';
• ...
            .map((response: Response) => <IProduct[]>response.json())
            .do(data => console.log('All: ' + JSON.stringify(data)))
            .catch(this.handleError);
} •getProducts(): Observable<IProduct[]> {  return
 this.http.get(this.productUrl)
private handleError(error: Response) {
}
```

Subscribing to an Observable

```
x.then(valueFn, errorFn)                          //Promise
x.subscribe(valueFn, errorFn)                     //Observable
x.subscribe(valueFn, errorFn, completeFn)         //Observable
let sub = x.subscribe(valueFn, errorFn, completeFn)
```

product-list.component.ts

```
ngOnInit(): void {
    this._productService.getProducts()
            .subscribe(products => this.products = products,
                       error => this.errorMessage = <any>error);
}
```

Summary

- Observables and Reactive Extensions
- Sending an Http Request
- Exception Handling
- Subscribing to an Observable

Routing Basics

Module Overview

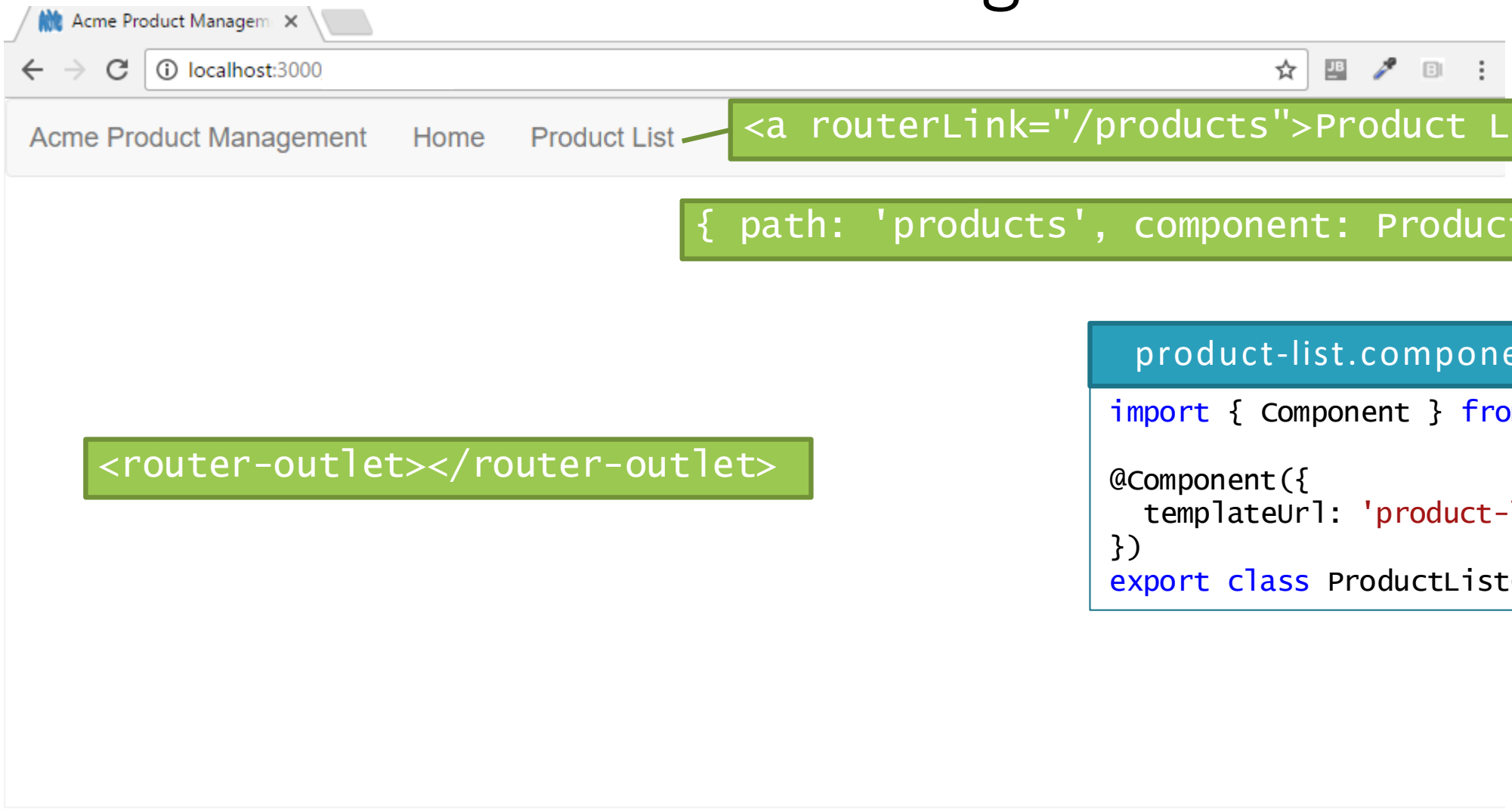
How Does Routing Work?

Configuring Routes

Tying Routes to Actions

Placing the Views

How Routing Works




```
<a routerLink="/products">Product List</a>
```

```
{ path: 'products', component: ProductListComponent }
```

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

product-list.component.ts

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

@Component({
  templateUrl: 'product-list.component.html'
})
export class ProductListComponent { }
```

Configuring Routes

app.module.ts

```
@NgModule({
  imports: [
    ...,
    RouterModule.forRoot([
      { path: 'products', component: ProductListComponent },

      { path: 'welcome', component: WelcomeComponent },
      { path: '', redirectTo: 'welcome', pathMatch: 'full' },
      { path: '**', redirectTo: 'welcome', pathMatch: 'full' }
    ])
  ],
  declarations: [...],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Tying Routes to Actions

app.component.ts

- ...
- @Component({
  - selector: 'pm-app', template: `
    - <ul class='nav navbar-nav'>

```
  <li><a [routerLink]="['/welcome']">Home</a></li>
  <li><a [routerLink]="['/products']">Product List</a></li>
</ul>
`
})
```

Placing the Views

app.component.ts

- ...
- @Component({
  - selector: 'pm-app', template: `
    - <ul class='nav navbar-nav'>

```
  <li><a [routerLink]="['/welcome']">Home</a></li>
  <li><a [routerLink]="['/products']">Product List</a></li>
</ul>
<router-outlet></router-outlet>
`
})
```

Summary

How Does Routing Work?

Configuring Routes

Tying Routes to Actions

Placing the Views

Routing Advance

Module Overvie w

Passing Parameters to a Route

Activating a Route with Code

Protecting Routes with Guards

Passing Parameters to a Route

http://localhost:3000/product/5

Acme Product Management

Acme Product Management Home Product List

Product Detail: 5: Hammer

Name:	Hammer
Code:	TBX-0048
Description:	Curved claw steel hammer
Availability:	May 21, 2016
Price:	$8.9
5 Star Rating:	★★★★★

Back

Passing Parameters to a Route

app.module.ts

```
@NgModule({
  imports: [
    ...,
    RouterModule.forRoot([
      { path: 'products', component: ProductListComponent },
      { path: 'product/:id', component: ProductDetailComponent },
      { path: 'welcome', component: WelcomeComponent },
      { path: '', redirectTo: 'welcome', pathMatch: 'full' },
      { path: '**', redirectTo: 'welcome', pathMatch: 'full' }
    ])
  ],
  declarations: [...],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Passing Parameters to a Route

product-list.component.html

```
<td>
  <a [routerLink]="['/product', product.productId]">
     {{product.productName}}
  </a>
</td>
```

app.module.ts

```
 { path: 'product/:id', component: ProductDetailComponent }
```

Reading Parameters from a Route

product-detail.component.ts

```
import { ActivatedRoute } from '@angular/router';

    constructor(private _route: ActivatedRoute) {
       console.log(this._route.snapshot.params['id']);
    }
```

app.module.ts

```
 { path: 'product/:id', component: ProductDetailComponent }
```

Activating a Route with Code

http://localhost:3000/product/5

Acme Product Management

Acme Product Management Home Product List

Product Detail: 5: Hammer

Name:	Hammer
Code:	TBX-0048
Description:	Curved claw steel hammer
Availability:	May 21, 2016
Price:	$8.9
5 Star Rating:	★★★★★

Back

Activating a Route with Code

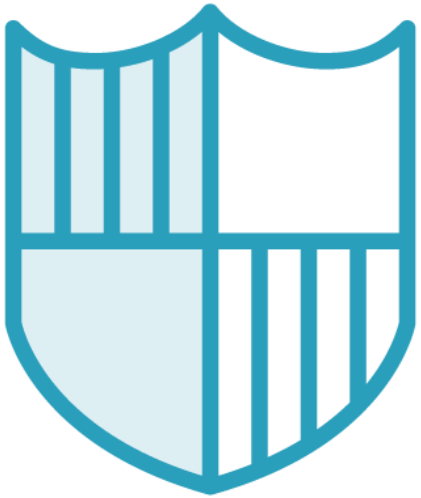
product-detail.component.ts

```
import { Router } from '@angular/router';
...
    constructor(private _router: Router) { }

    onBack(): void {
      this._router.navigate(['/products']);
    }
```

Route Guards

Using Route Guards



Limit access to a route

Warn before leaving a route

Retrieve data before accessing a route

Route Guards

- May be User is not authorized to navigate to the target component.
- May be the user must login (*authenticate*) first.
- May be you should fetch some data before you display the target component.
- You might want to save pending changes before leaving a component.
- You might ask the user if it's OK to discard pending changes rather than save them.

Route Guards

A guard's return value controls the router's behavior:

- If it returns `true`, the navigation process continues.
- If it returns `false`, the navigation process stops and the user stays put.

Route Guards

- The guard *might* return its boolean answer synchronously.
- But in many cases, the guard can't produce an answer synchronously.
- The guard could ask the user a question, save changes to the server, or fetch fresh data. These are all asynchronous operations.
- Accordingly, a routing guard can return an Observable<boolean> or a Promise<boolean> and the router will wait for the observable to resolve to true or false.

Kinds Of Guards

1. CanActivate to mediate navigation *to* a route.
2. CanActivateChild() to mediate navigation *to* a child route.
3. CanDeactivate to mediate navigation *away* from the current route.
4. Resolve to perform route data retrieval *before* route activation.
5. CanLoad to mediate navigation *to* a feature module loaded *asynchronously*.

Building a Guard

product-guard.service.ts

```
import { Injectable } from '@angular/core';
import { CanActivate } from '@angular/router';

@Injectable()
export class ProductDetailGuard implements CanActivate {

    canActivate(): boolean {
     ...
    }
}
```

Registering a Guard

- app.module.ts

```
• ...
• import { ProductDetailGuard } from './products/product-guard.service';

• @NgModule({  imports:
providers: [ ProductDetailGuard ],
[ ... ],
bootstrap: [ AppComponent ]
})• declarations: [...],
export class AppModule { }
```

Using a Guard

app.module.ts

```
@NgModule({
  imports: [
    ...,
    RouterModule.forRoot([
      { path: 'products', component: ProductListComponent },
      { path: 'product/:id',
        canActivate: [ ProductDetailGuard ],
        component: ProductDetailComponent },
      ...])
  ],
  declarations: [...],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Summary

Passing Parameters to a Route

Activating a Route with Code

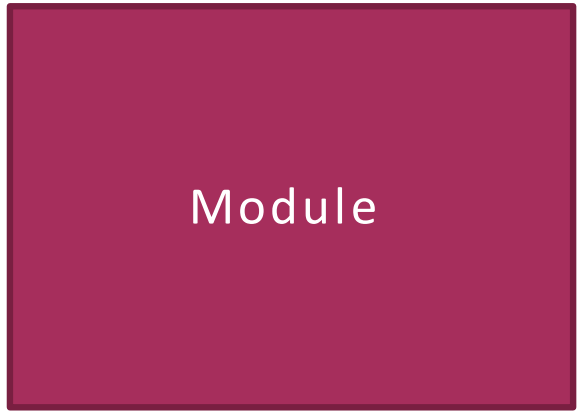
Protecting Routes with Guards

Modules

Module Overview

- What Is an Angular Module?
- Angular Module Metadata
- Creating a Feature Module
- Defining a Shared Module
- Revisiting AppModule

What Is an Angular Module?

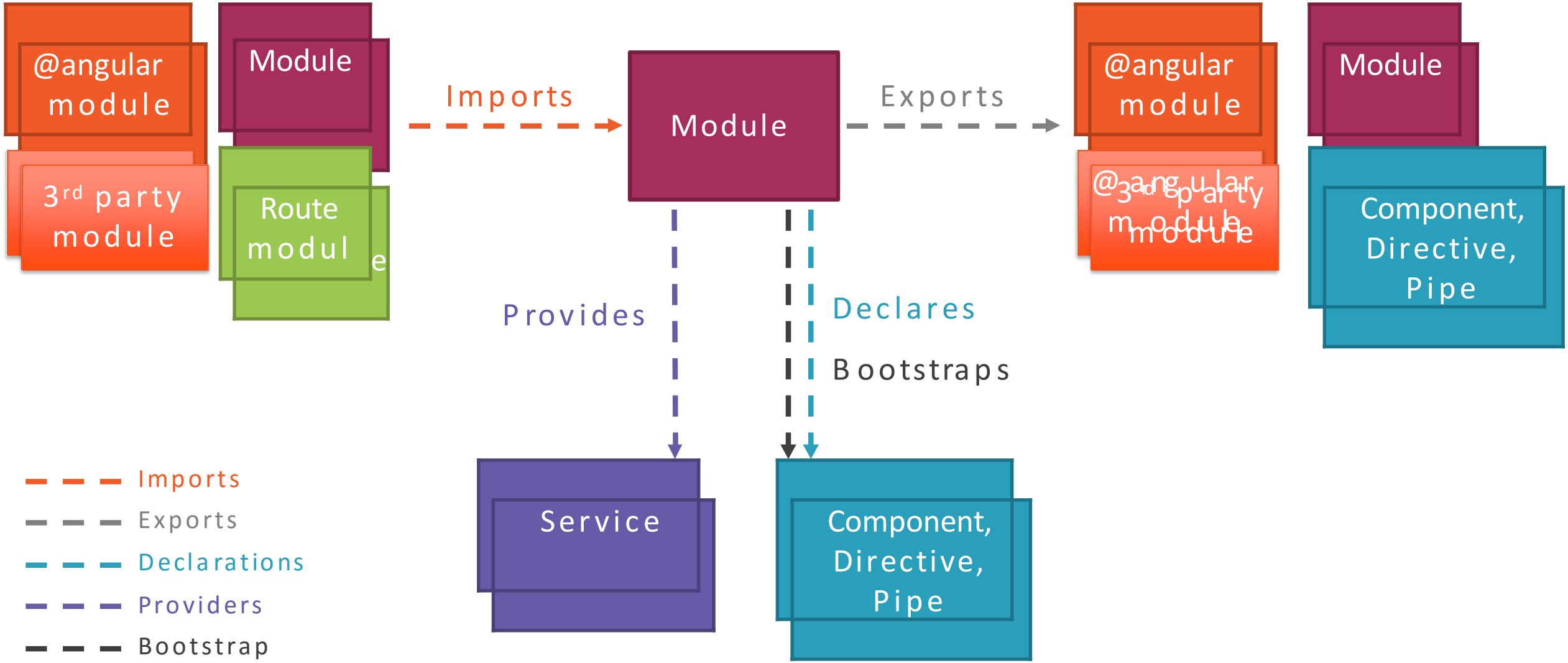


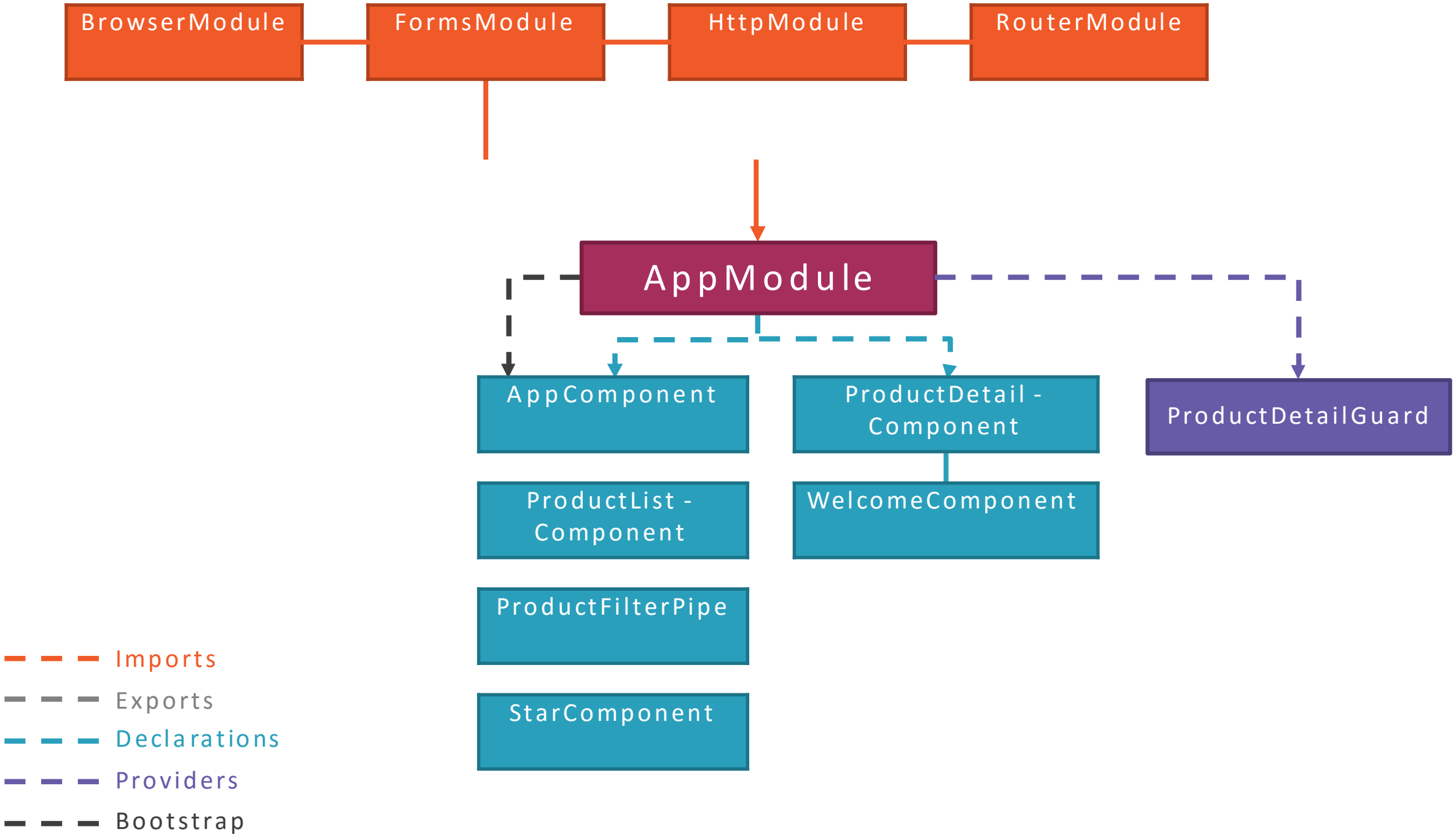
A class with an NgModule decorator

Its purpose:

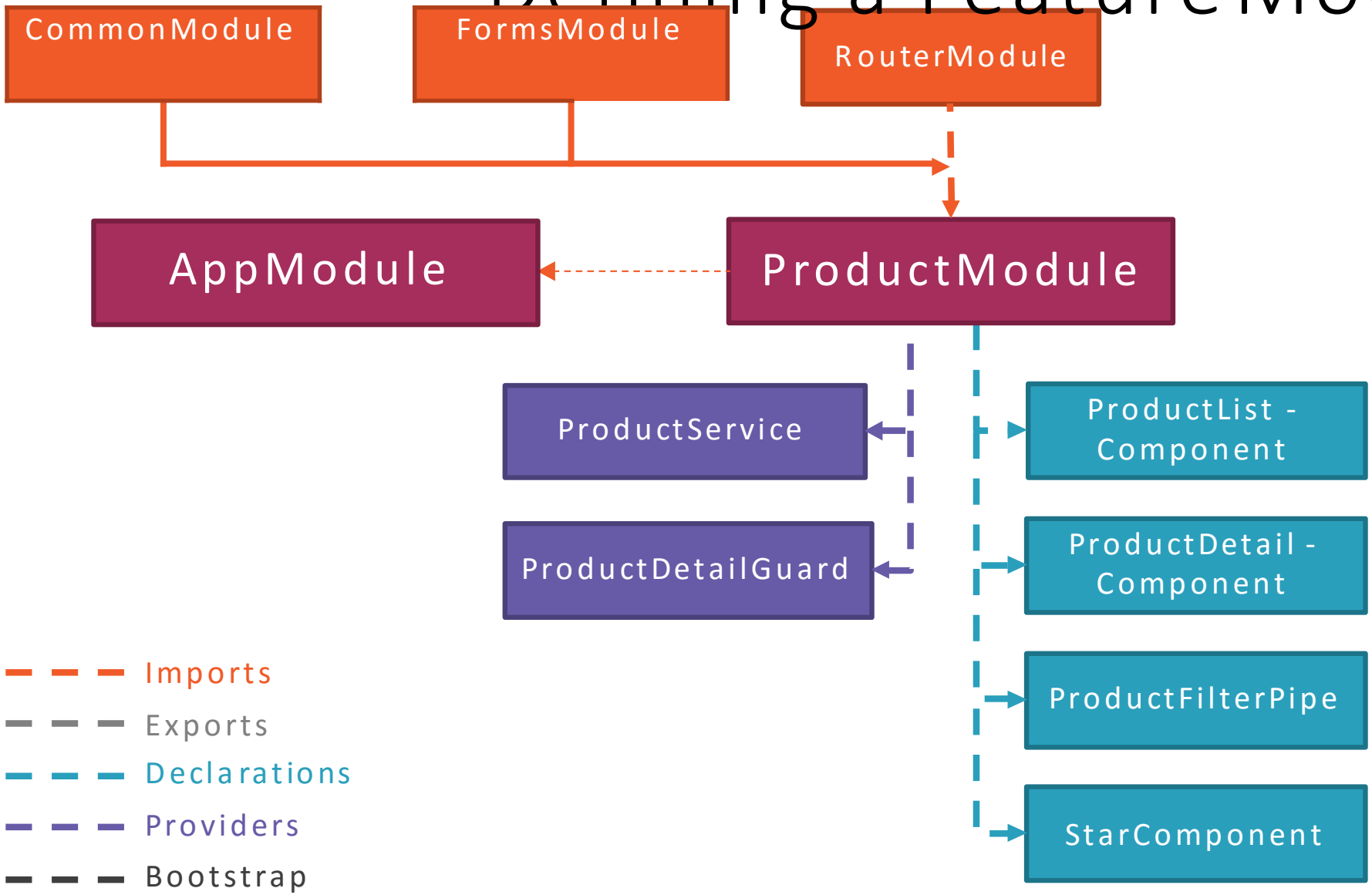
- Organize the pieces of our application
- Arrange them into blocks
- Extend our application with capabilities from external libraries
- Provide a template resolution environment
- Aggregate and re-export

Angular Module

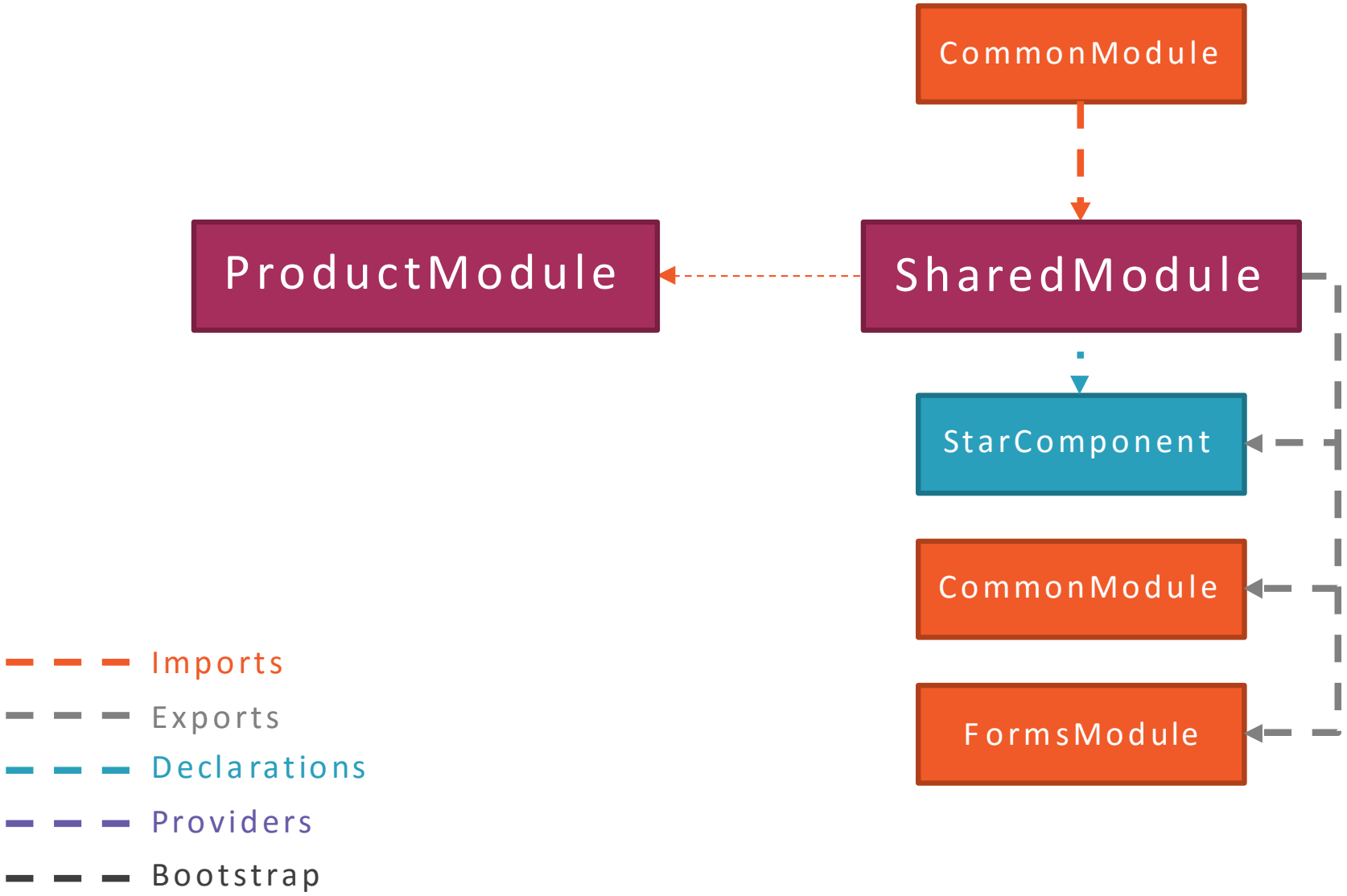



BrowserModule
FormsModule
HttpModule
RouterModule
AppModule
AppComponent
ProductDetail - Component
ProductDetailGuard
ProductList - Component
WelcomeComponent
ProductFilterPipe
StarComponent
Imports
Exports
Declarations
Providers
Bootstrap

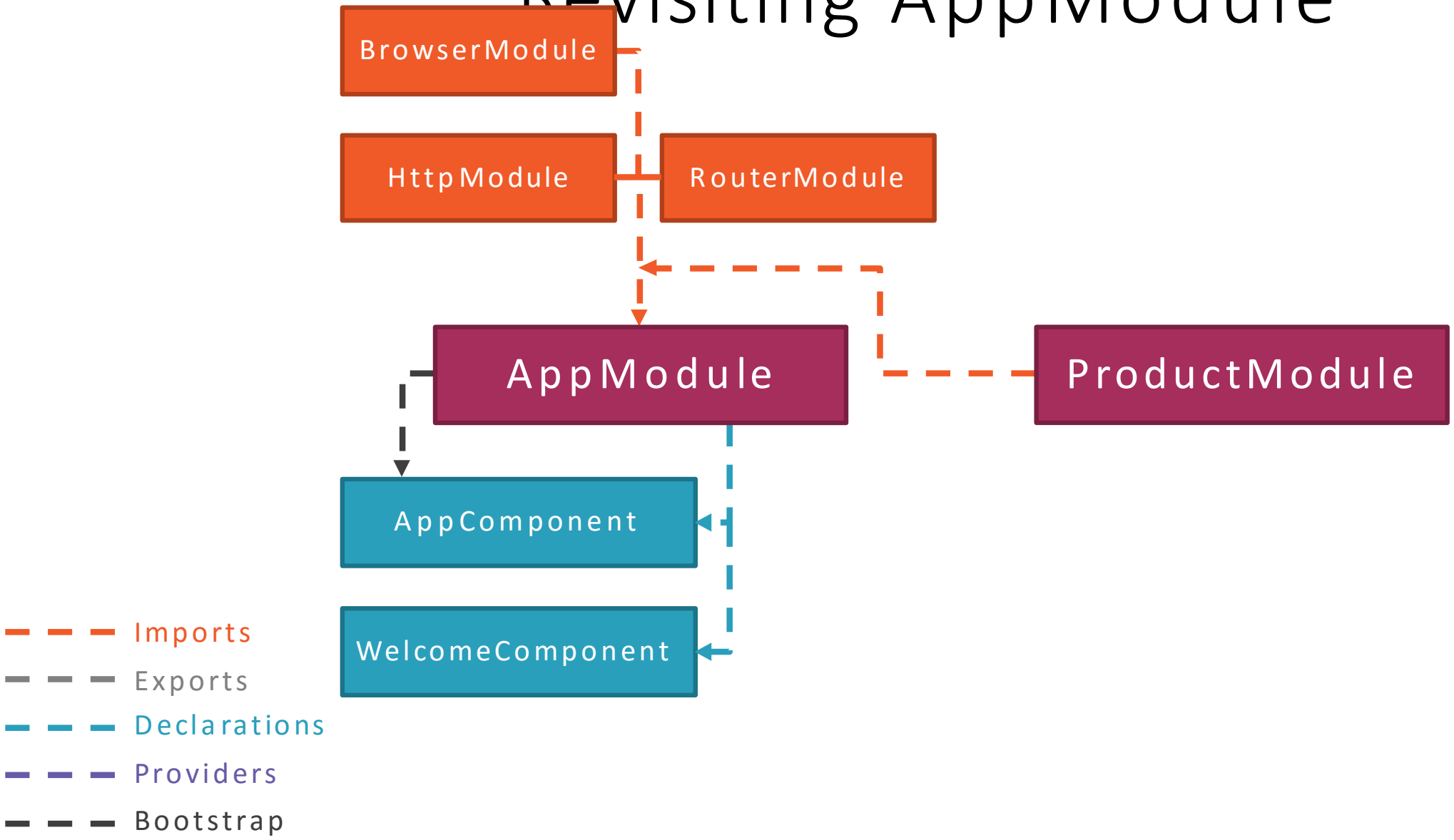
Defining a Feature Module

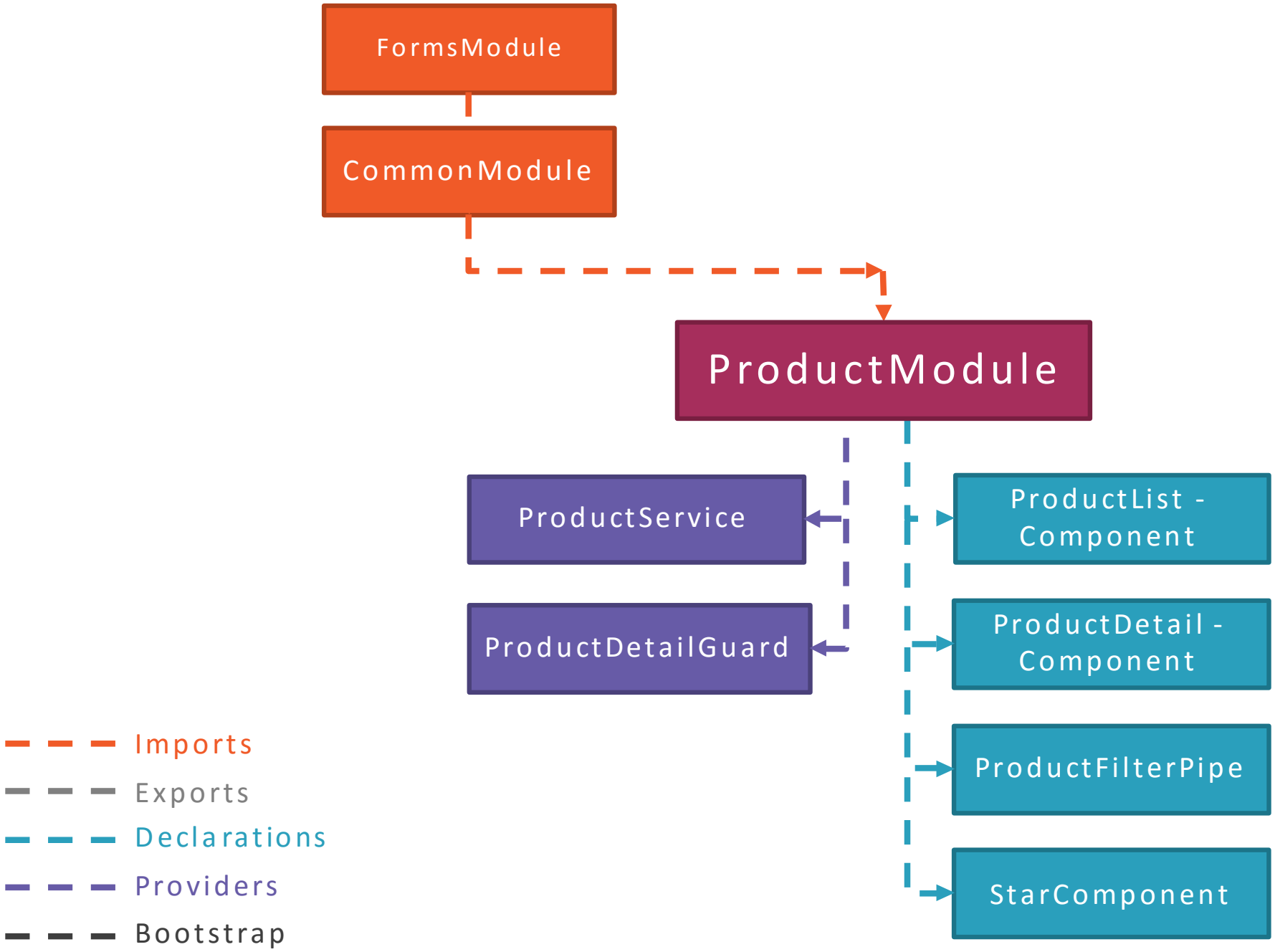


Defining a SharedModule



Revisiting AppModule




FormsModule
CommonModule
ProductModule
ProductService
ProductDetailGuard
ProductList - Component
ProductDetail - Component
ProductFilterPipe
StarComponent
Imports
Exports
Declarations
Providers
Bootstrap

Application Routing Module

app-routing.module.ts

```
import { NgModule }  from '@angular/core';
import { RouterModule } from '@angular/router';

import { WelcomeComponent } from './home/welcome.component';


@NgModule({
  imports: [
    RouterModule.forRoot([
      { path: 'welcome', component: WelcomeComponent },
      { path: '', redirectTo: 'welcome', pathMatch: 'full'},
      { path: '**', redirectTo: 'welcome', pathMatch: 'full' }
    ])
  ],
  exports: [ RouterModule ]
})
export class AppRoutingModule { };
```

Using the Routing Module

app.module.ts

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

Feature Routing Module

product-routing.module.ts

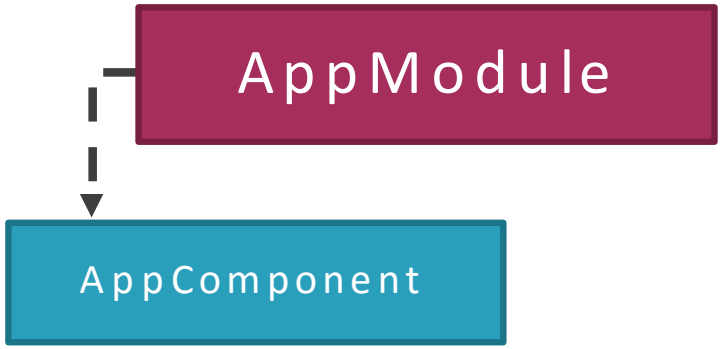
```
import { NgModule }  from '@angular/core';
import { RouterModule } from '@angular/router';
import { ProductListComponent } from './product-list.component';
import { ProductDetailComponent } from './product-detail.component';
import { ProductDetailGuard } from './product-guard.service';
@NgModule({
 imports: [
    RouterModule.forChild([
      { path: 'products', component: ProductListComponent },
      { path: 'product/:id', canActivate: [ ProductDetailGuard],
        component: ProductDetailComponent }
    ])
  ],
  exports: [ RouterModule ]
})
export class AppRoutingModule { };
```

Using the Routing Module

product.module.ts

```
@NgModule({
  imports: [
    SharedModule,
    ProductRoutingModule
  ],
  declarations: [
    ProductListComponent,
    ProductDetailComponent,
    ProductFilterPipe
  ],
  providers: [
    ProductService,
    ProductDetailGuard
  ]
})
export class ProductModule {}
```

Bootstrap Array



app.module.ts

```
...
bootstrap: [ AppComponent ]
...
```

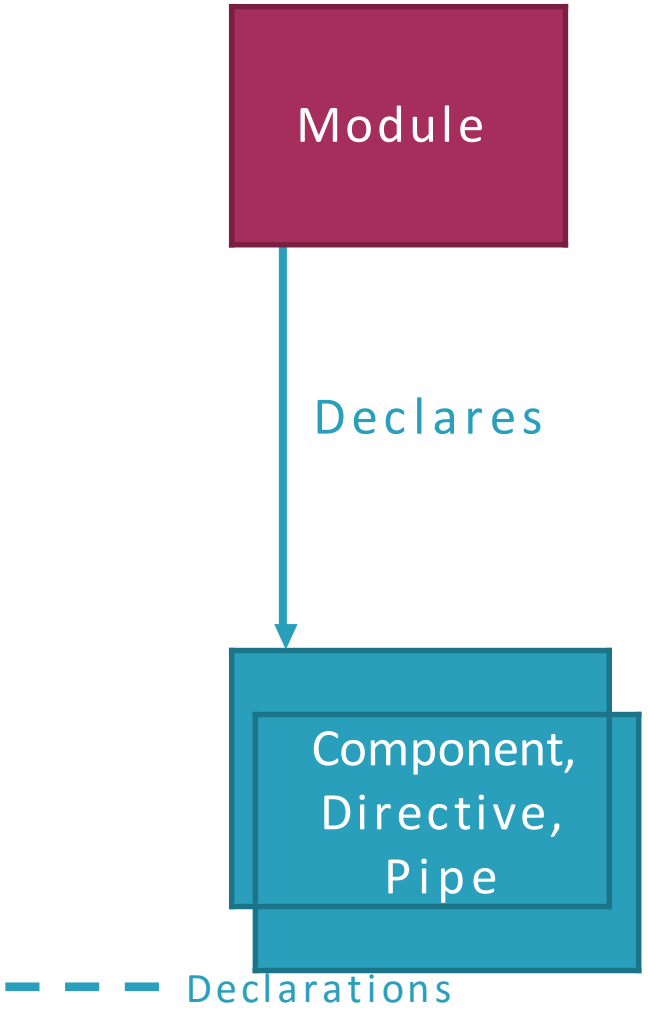
Bootstrap

Bootstrap Array Truths

- Every application must bootstrap at least one component, the root application component.

- The bootstrap array should only be used in the root application module, AppModule.

Declarations Array



app.module.ts

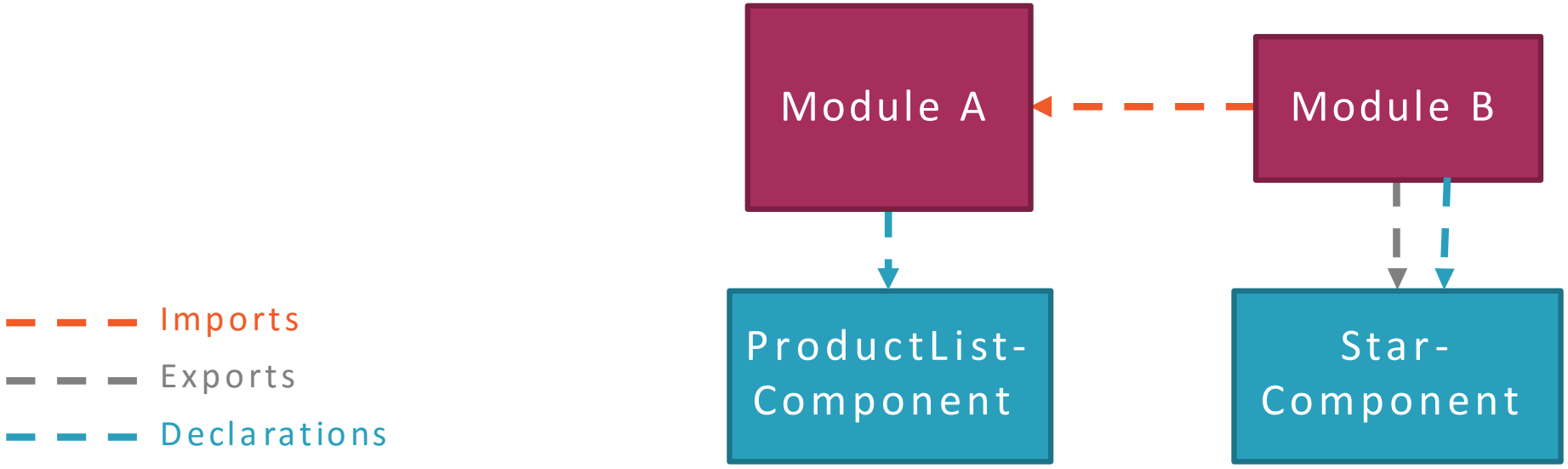
```
...
declarations: [
    AppComponent,
    WelcomeComponent,
    ProductListComponent,
    ProductDetailComponent,
    ProductFilterPipe,
    StarComponent
]
...
```

Declarations Array Truths

- Every component and pipe we create must belong to one and only one Angular module.
- Only declare components, directives and pipes.

Declarations Array Truth #3

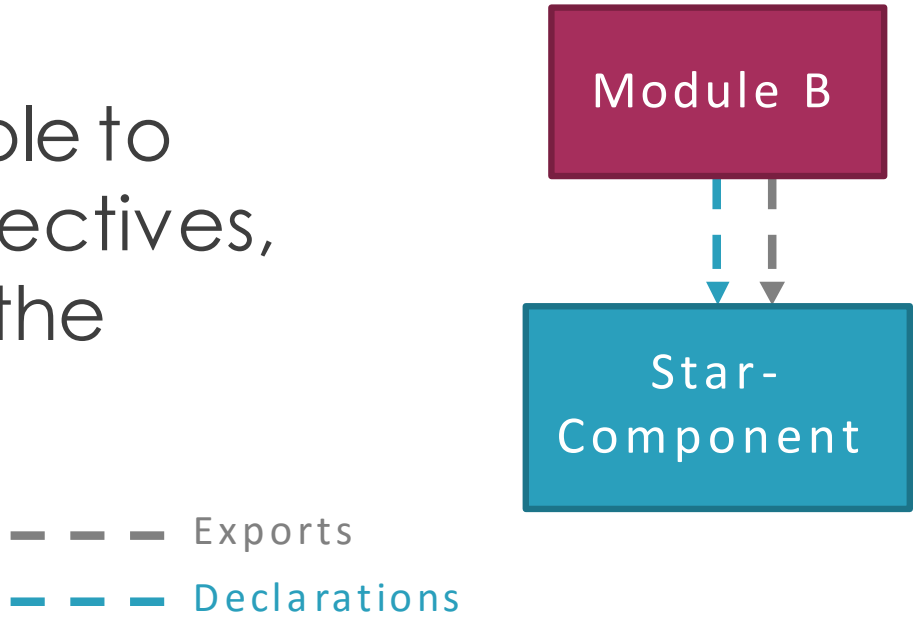
Never re-declare components, directives, or pipes that belong to another module



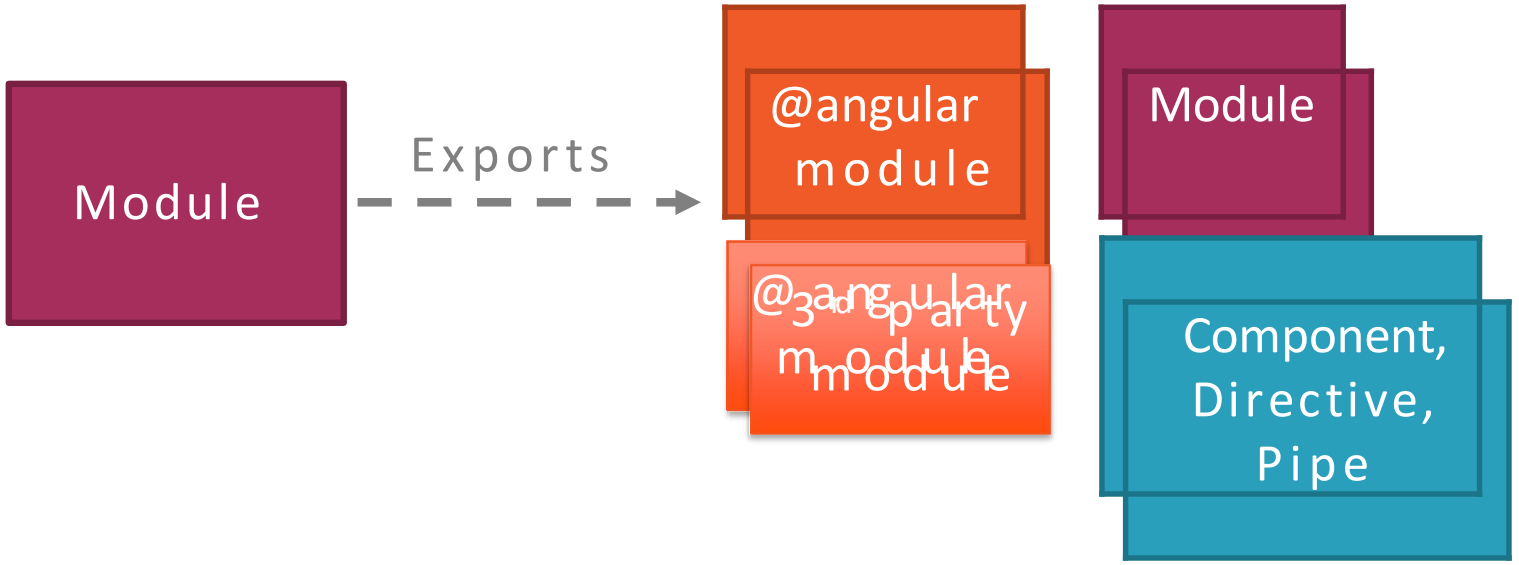
Declarations Array Truth #4

All declared components, directives, and pipes are private by default.

They are only accessible to other components, directives, and pipes declared in the same module.



Exports Array

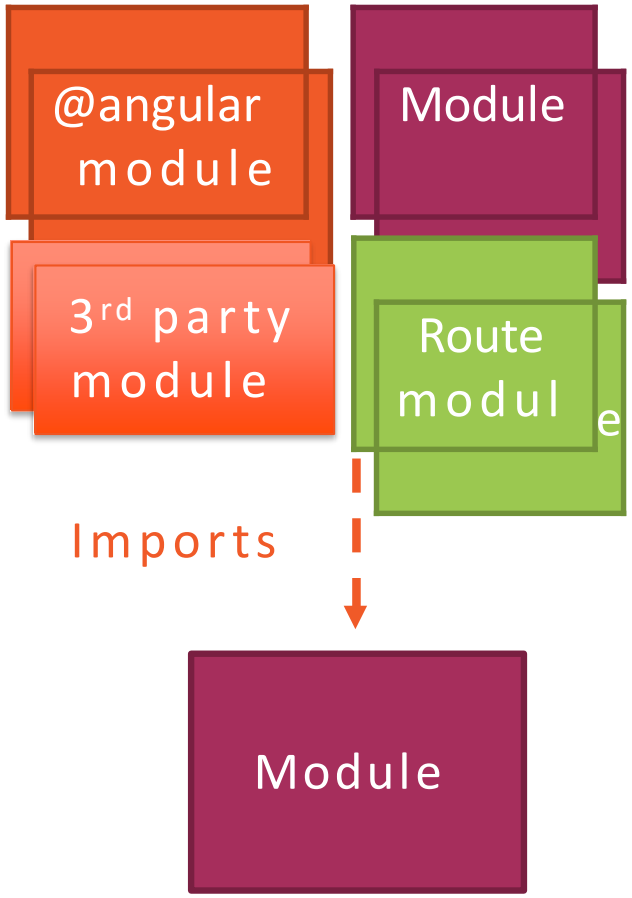


- Imports
- Exports
- Declarations
- Providers
- Bootstrap

Exports Array Truths

- Export any component or pipe if another components need it.
- Re-export modules to re-export their components, and pipes.

Imports Array

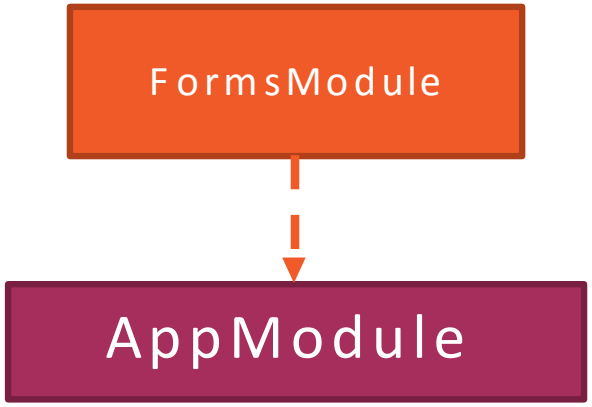


app.module.ts

```
...
imports: [
 BrowserModule,
 FormsModule,
 HttpModule,
 RouterModule.forRoot([...])
]
...
```

Imports Array Truth #1

Importing amodule makes available any exported components, directives, and pipes from that module.



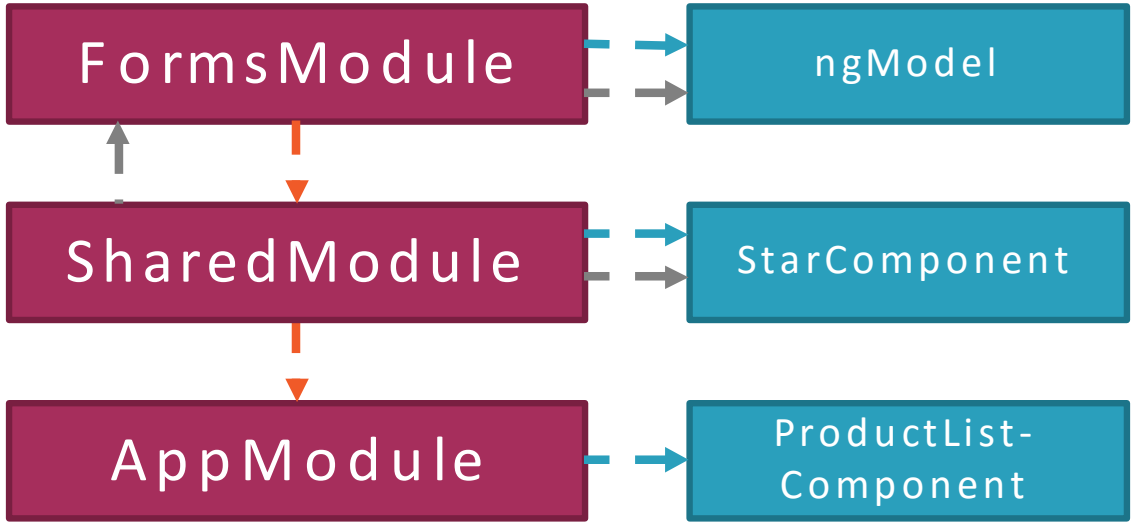
Imports
Exports
Declarations
Providers
Bootstrap

Imports Array Truth #2

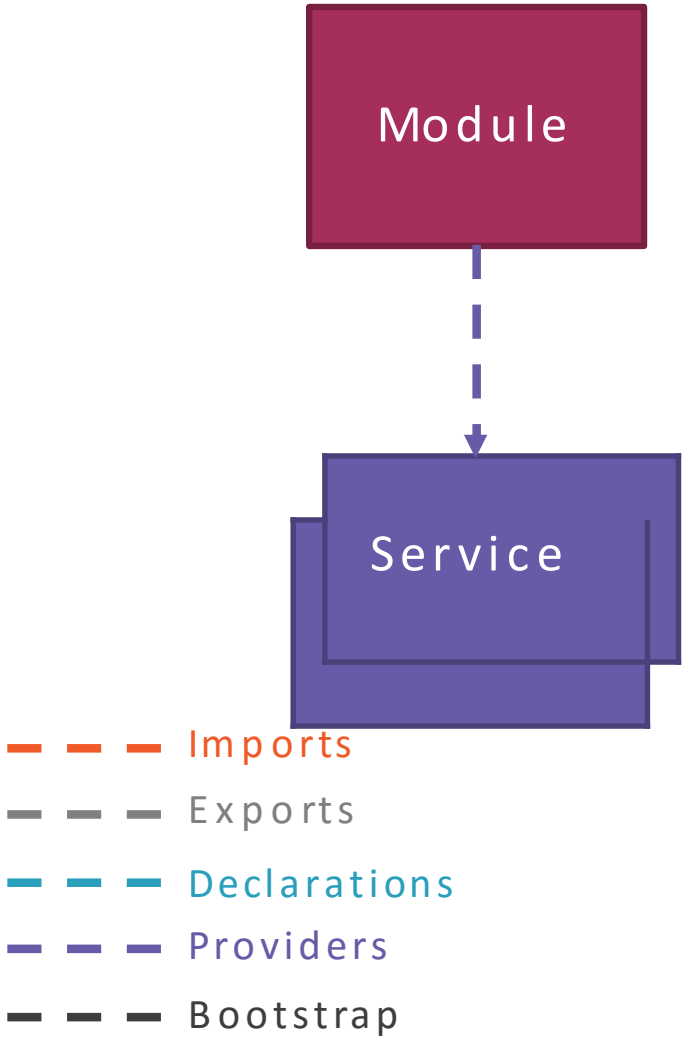
Only import what this module needs.

Imports Array Truth #3

Importing amodule does NOT provide access to its imported modules



Providers Array

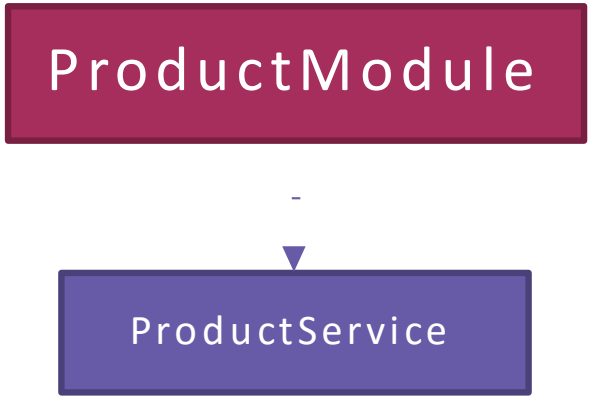


app.module.ts

```
...
providers: [ ProductDetailGuard ]
...
```

Providers Array Truth #1

Any service provider added to the providers array is registered at the root of the application.



- Imports
- Exports
- Declarations
- Providers
- Bootstrap

Providers Array Truth #2

Don't add services to the providers array of a shared module.

Consider building a CoreModule for services and importing it once in the AppModule.

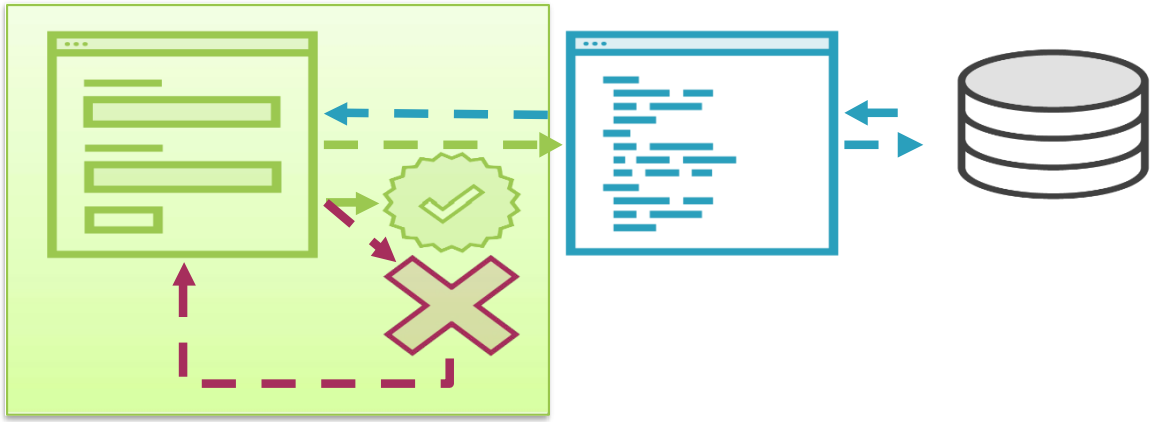
Providers Array Truth #3

Routing guards must be added to the providers array of an Angular module.

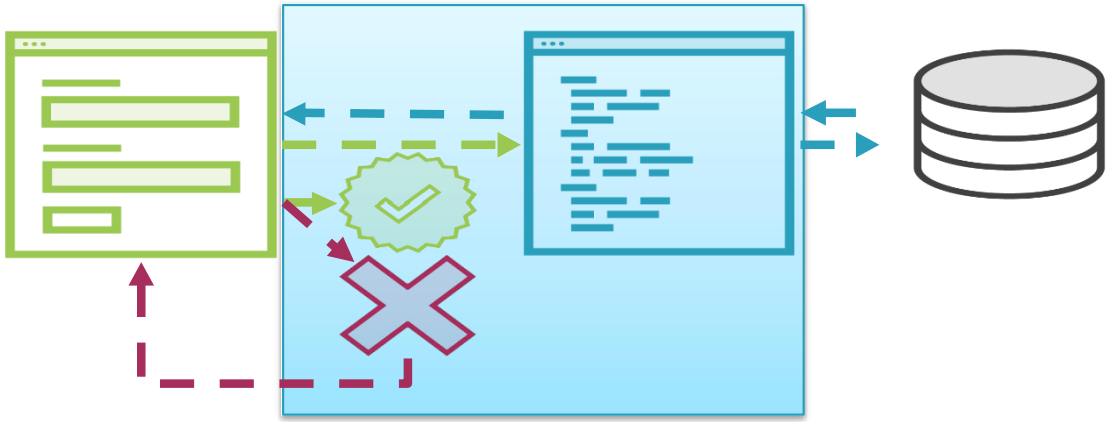
Forms

Angular Forms

Template-driven



Reactive
(Model-driven)



Angular Forms

Template-driven

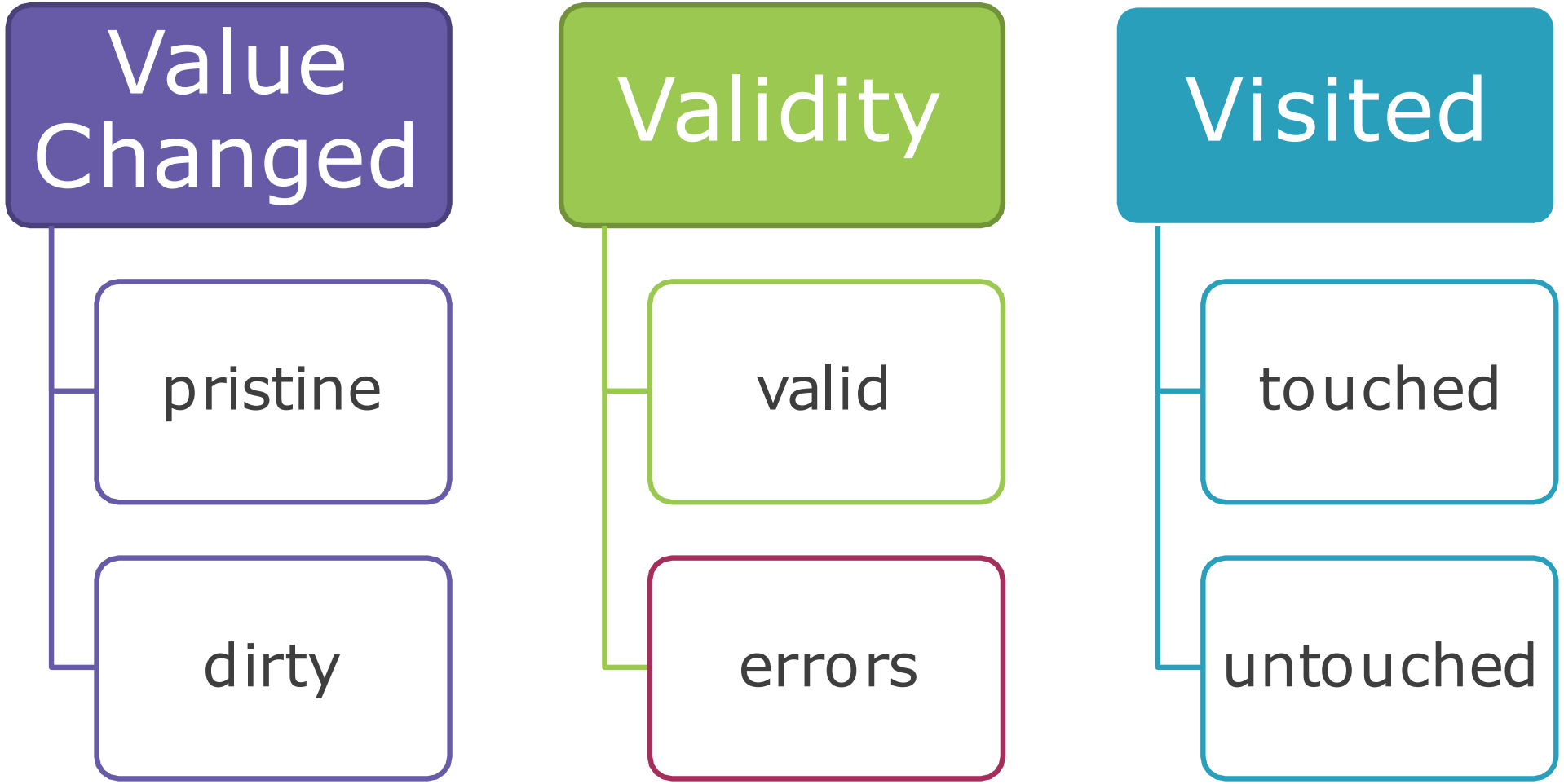
Easy to use

Similar to Angular 1

- **Reactive**
- More flexible ->
- more complex scenarios
- Easily add input elements

dynamically Easier unit testing

State


Value Changed
pristine
dirty
Validity
valid
errors
Visited
touched
untouched

Template-driven Form

customer.component.html

- <form (ngSubmit)="save()">
- <fieldset>
- <div [ngClass]="{'has-error': firstNameVar.touched &&

```
                                                    !firstNameVar.valid }">
```

- <label for="firstNameId">First Name</label>
- <input id="firstNameId" type="text" placeholder="First Name (required)" required
- minlength="3" [(ngModel)]="customer.firstName"
- name="firstName"
- #firstNameVar="ngModel" />

```
  <span *ngIf="firstNameVar.touched && firstNameVar.errors">
    Please enter your first name.
  </span>
</div>
...
<button type="submit">Save</button>
</fieldset>
</form>
```

Reactive Form

customer.component.html

```
<form (ngSubmit)="save()" [formGroup]="signupForm">
  <fieldset>
    <div [ngClass]="{'has-error': formError.firstName }">
      <label for="firstNameId">First Name</label>
      <input id="firstNameId" type="text"
             placeholder="First Name (required)"
             formControlName="firstName" />
      <span *ngIf="formError.firstName">
        {{formError.firstName}}
      </span>
    </div>
    ...
    <button type="submit">Save</button>
  </fieldset>
</form>
```

Built-in Validation Rules

```
this.customerForm = this.fb.group({
      firstName: ['', Validators.required],
      sendCatalog: true
});
```

```
this.customerForm = this.fb.group({
      firstName: ['',
         [Validators.required, Validators.minLength(3)]],
      sendCatalog: true
});
```

Custom Validator

```
function myCustomValidator(c: AbstractControl):
                    {[key: string]: boolean} | null {
    if (somethingIsWrong) {
        return { 'myvalidator': true };
    }
    return null;
}
```

Custom Validator with Parameters

```
function myCustomValidator(param: any): ValidatorFn {
    return (c: AbstractControl):
                    {[key: string]: boolean} | null => {
    if (somethingIsWrong) {
        return { 'myvalidator': true };
    }
    return null;
}
```

References

- https://angular.io/