Angular Is …

A JavaScript framework

For building client-side applications

Using HTML, CSS and JavaScript

Why Angular?

Expressive HTML

Powerful Data Binding

Modular By Design

Built-in Back-End Integration

Why Angular 5?

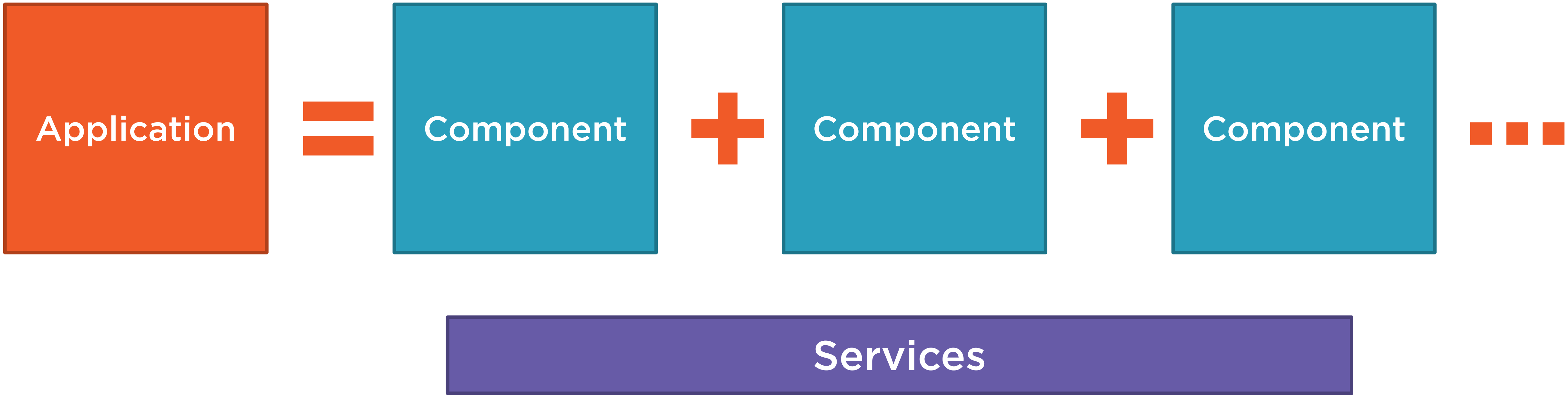
- Built for Speed
- Modern
- Simplified API
- Enhances Productivity

Anatomy of an Angular Application

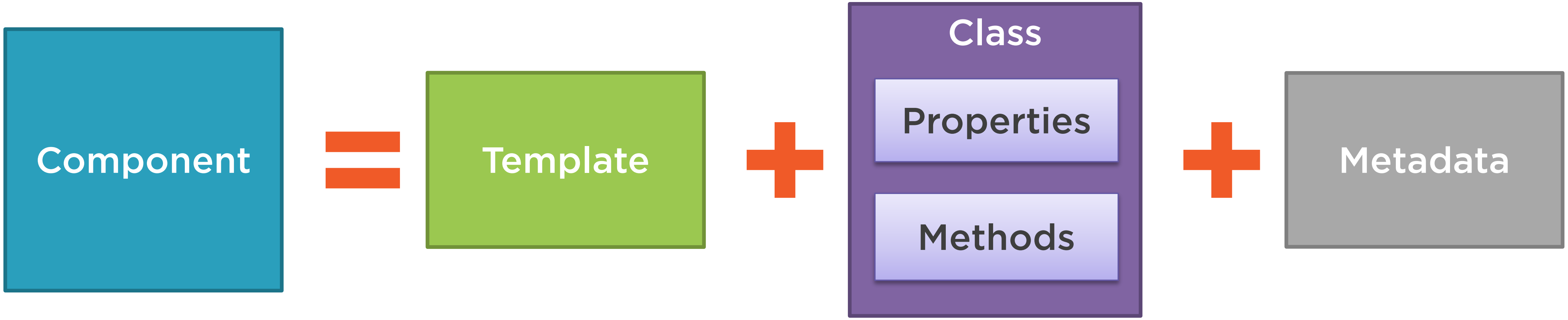
Sample Application

Course Outline

Anatomy of an Angular Application



Component



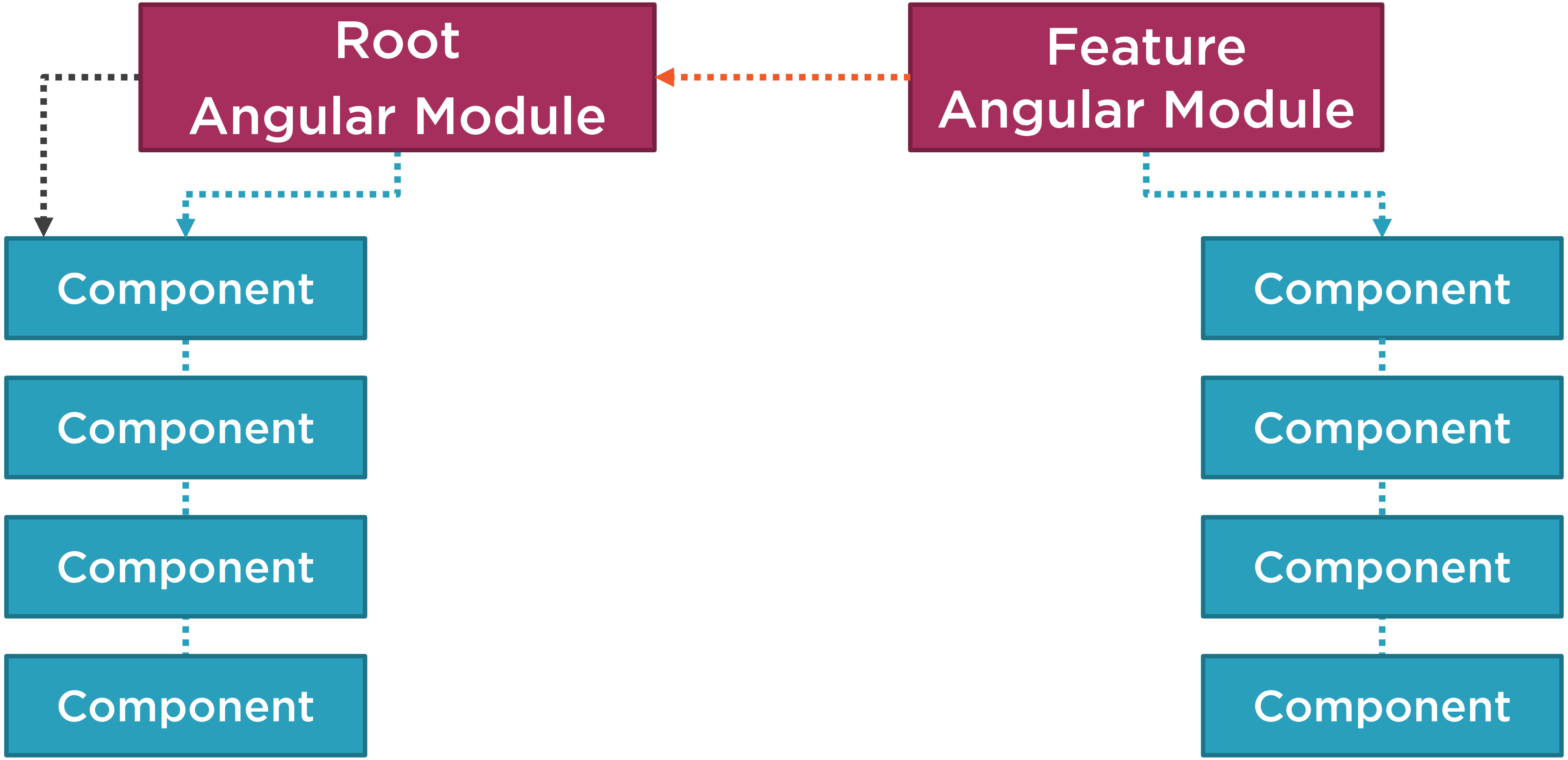
Component = Template + Class + Metadata

Class

Properties

Methods

Angular Modules



Root Angular Module

Feature Angular Module

Component

Component

Component

Component

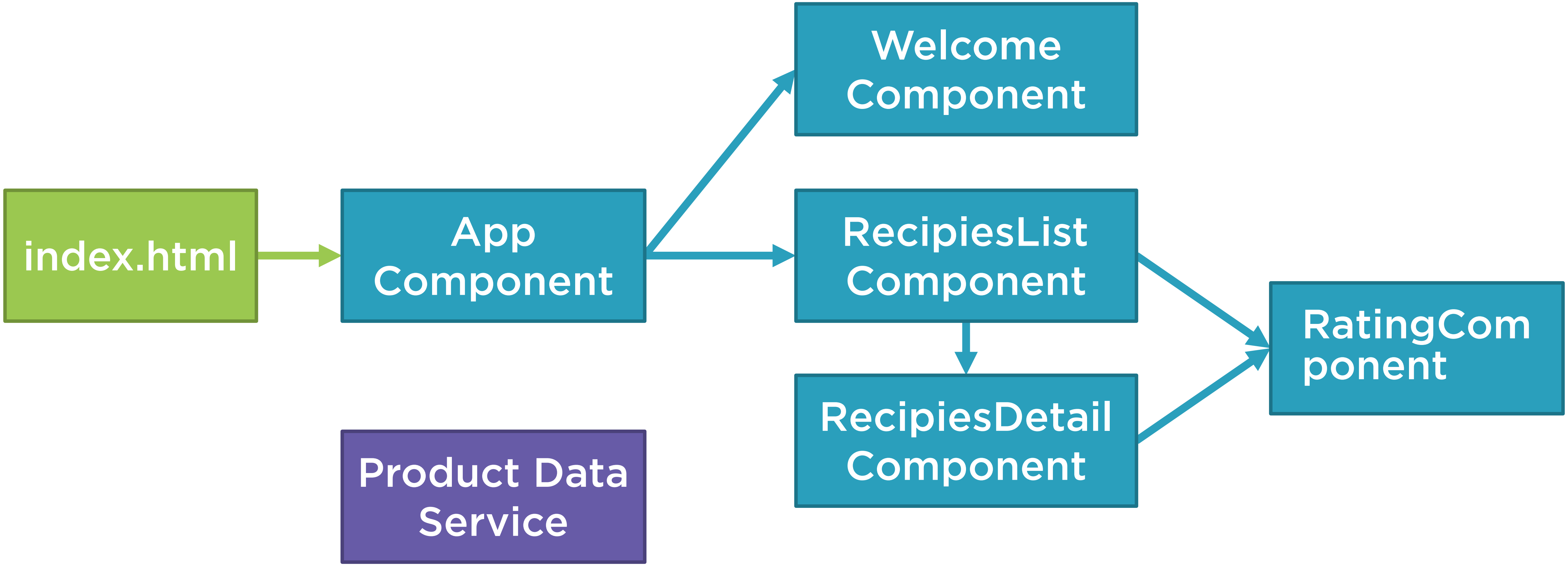
Component

Component

Component

Component

Sample Application Architecture



index.html

App Component

Welcome Component

RecipiesList Component

RecipiesDetail Component

RatingComponent

Product Data Service

Setting up an Angular Application

About Modules

JavaScript Language Specification

ECMAScript (ES)

ES 3

ES 5

ES 2015 (formerly known as ES 6)

- Must be transpiled

Selecting a Language

ES 5	ES 2015	TypeScript	Dart
• Runs in the browser • No compile required	• Lots of new features (classes, let, arrow, etc.)	• Superset of JavaScript • Strong typing • Great IDE tooling	• No JavaScript

What Is TypeScript?

TypeScript

Open source language

Superset of JavaScript

Transpiles to plain JavaScript

Strongly typed

- TypeScript type definition files (*.d.ts)

Class-based object-orientation

Setting up Our Environment

npm

Set up the Angular application

npm

Node Package Manager

Command line utility

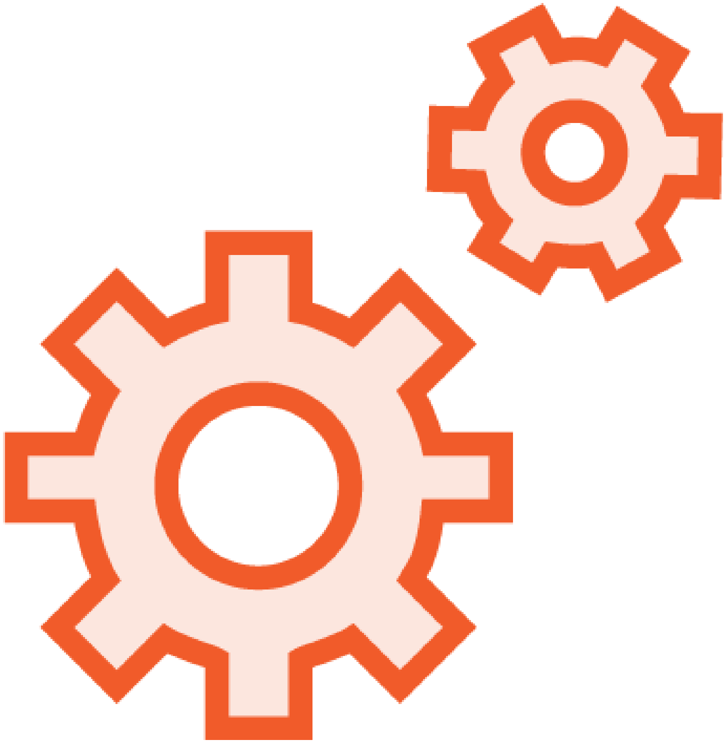
Installs libraries, packages, and applications

https://www.npmjs.com/

Setting up an Angular 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 2 Application



Manually perform each step

www.angular.io Quick Start

Download the results of these steps

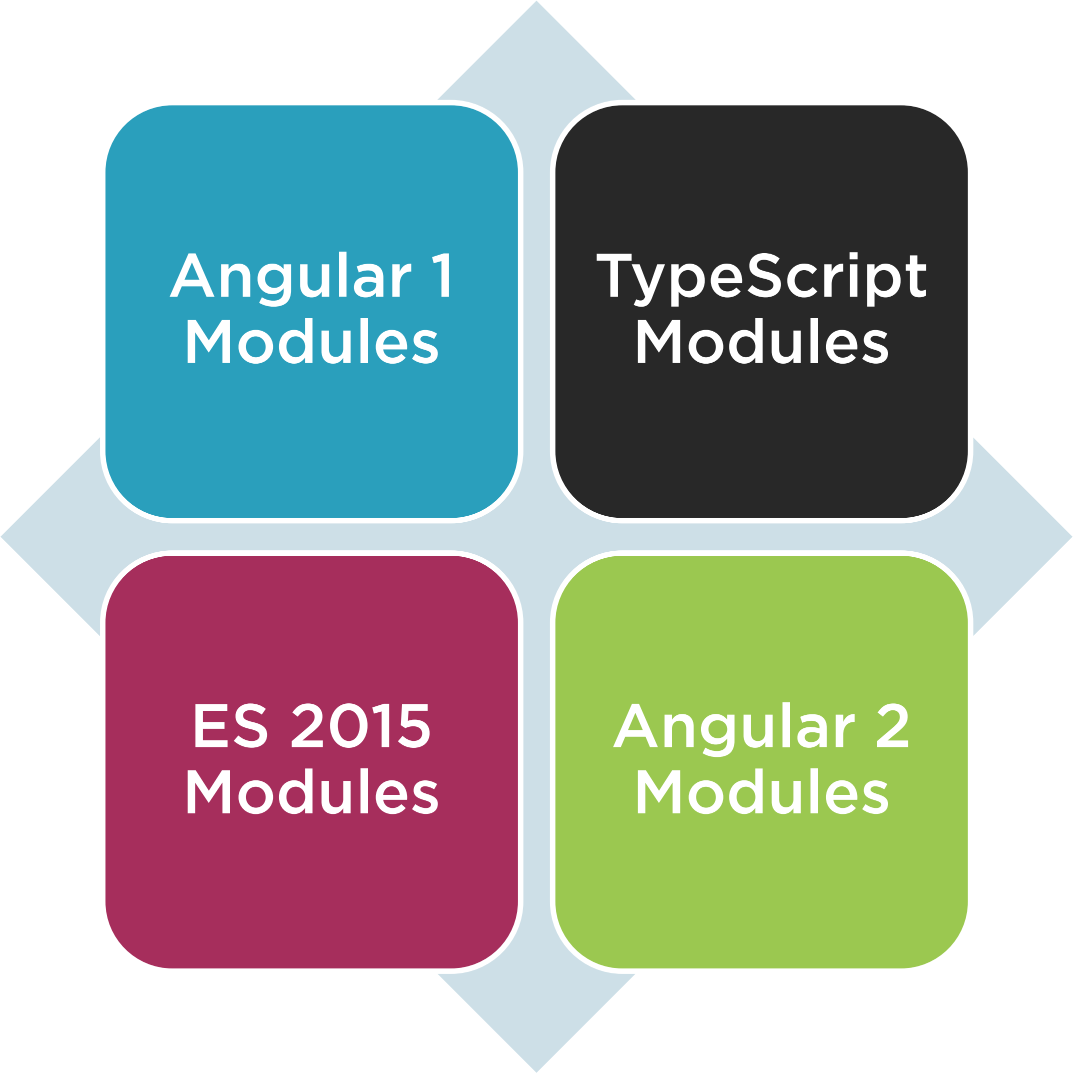
AngularCli

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

Modules

Namespaces

Code Organization



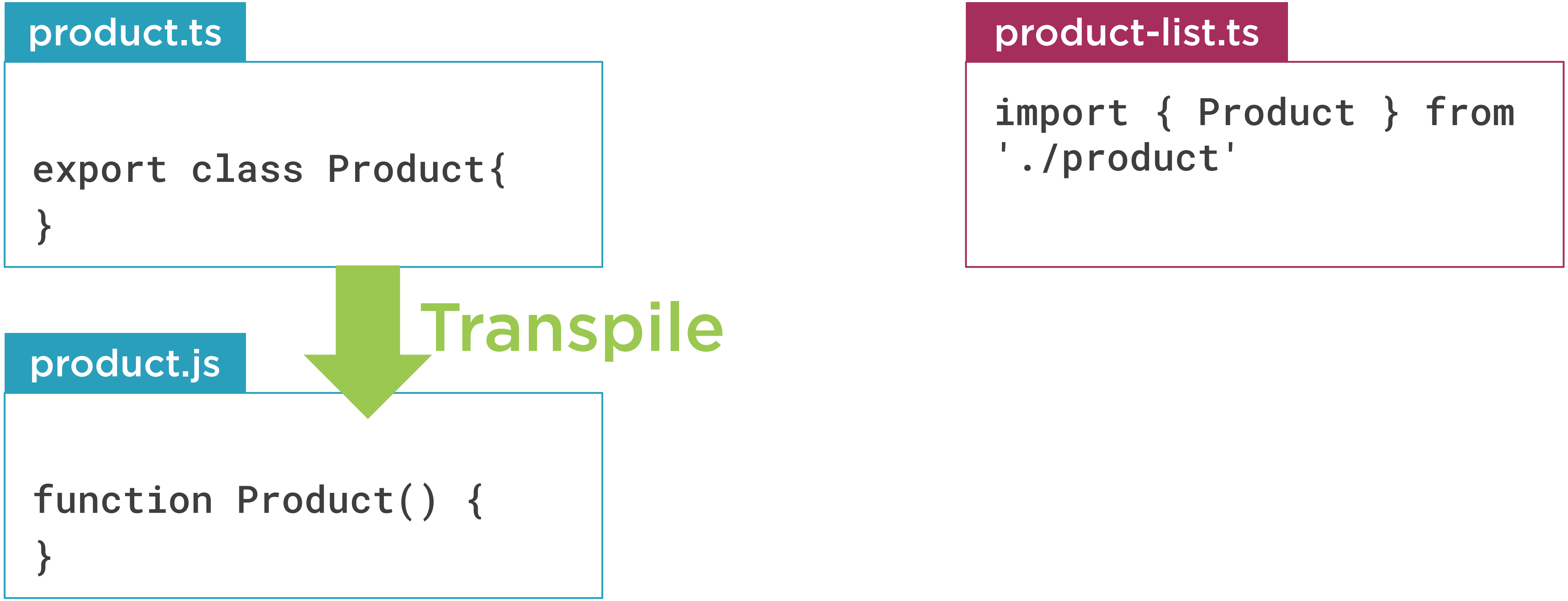
Angular 1 Modules

TypeScript Modules

ES 2015 Modules

Angular 2 Modules

ES 2015 Modules



Export

product.ts

```
export class Product{
}
```

Transpile

product.js

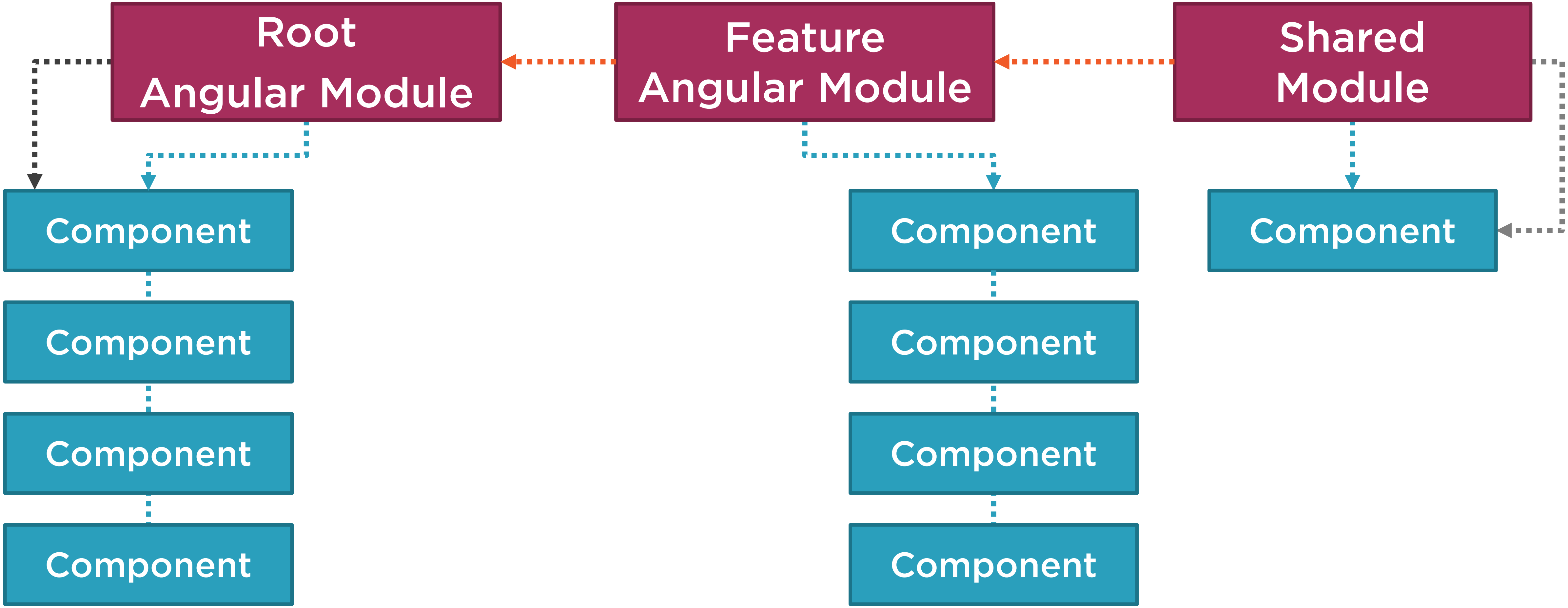
```
function Product() {
}
```

Import

product-list.ts

```
import { Product } from
'./product'
```

Angular Modules



Root Angular Module

Feature Angular Module

Shared Module

Component

Component

Component

Component

Component

Component

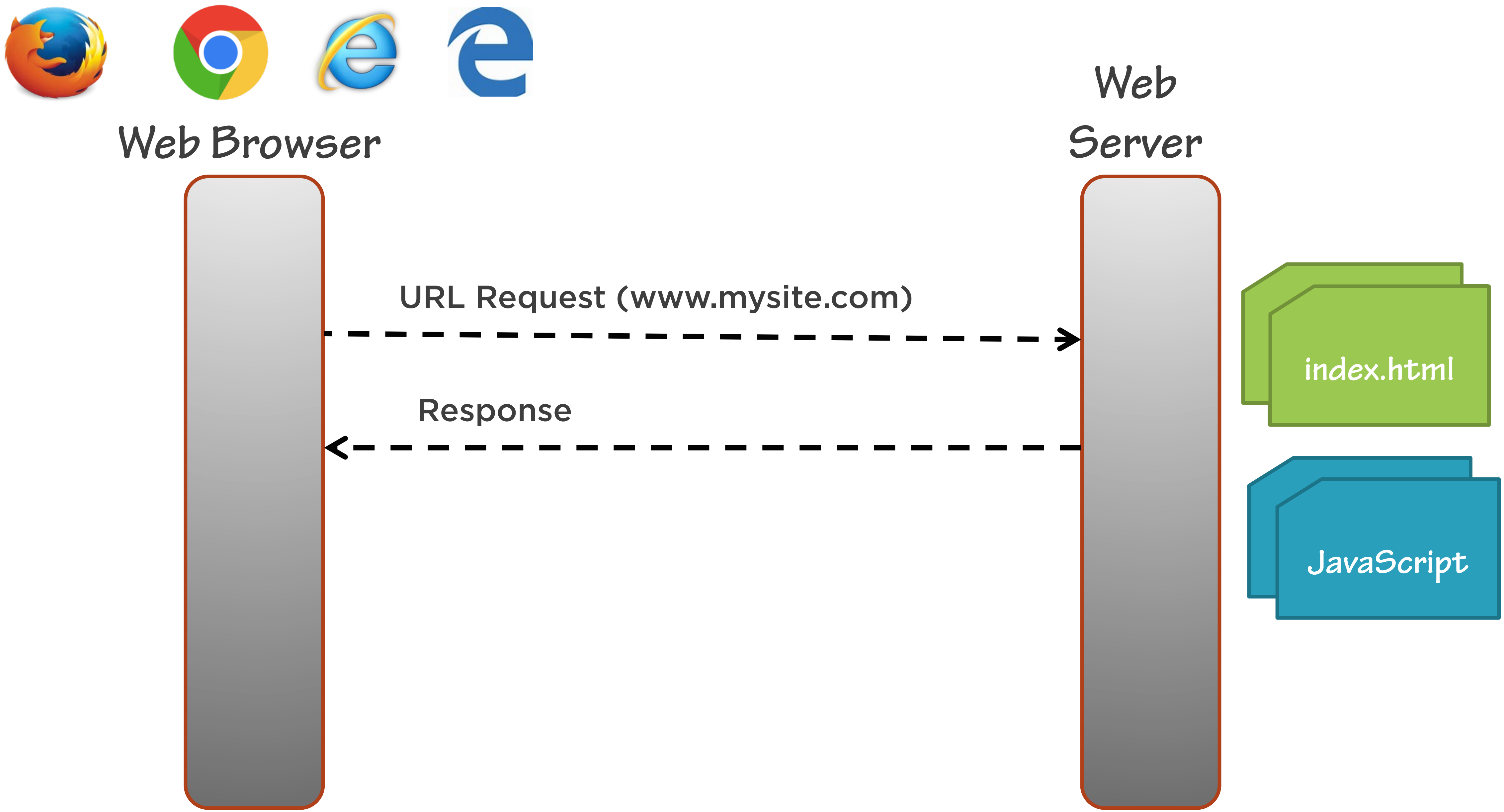
Component

Component

Component

Modules

ES Modules	Angular Modules
Code files that import or export something	Code files that organize the application into cohesive blocks of functionality
Organize our code files	Organize our application
Modularize our code	Modularize our application
Promote code reuse	Promote application boundaries



Web Browser

Web Server

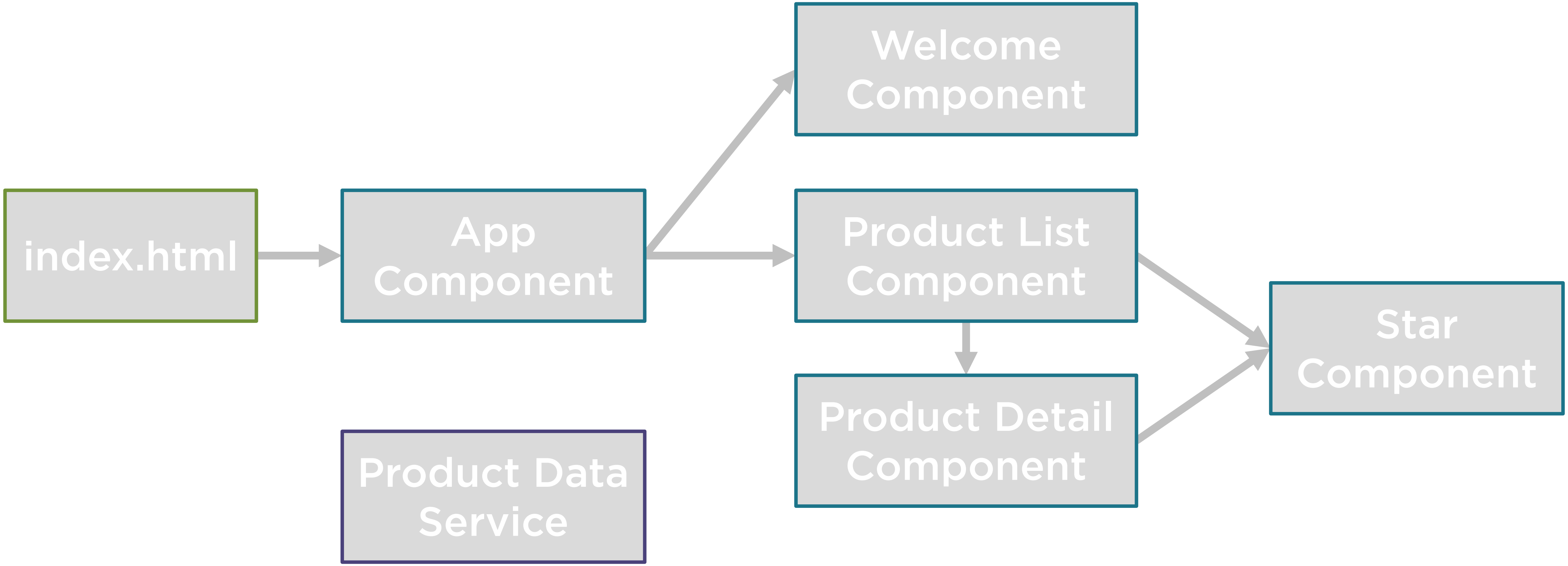
URL Request (www.mysite.com)

Response

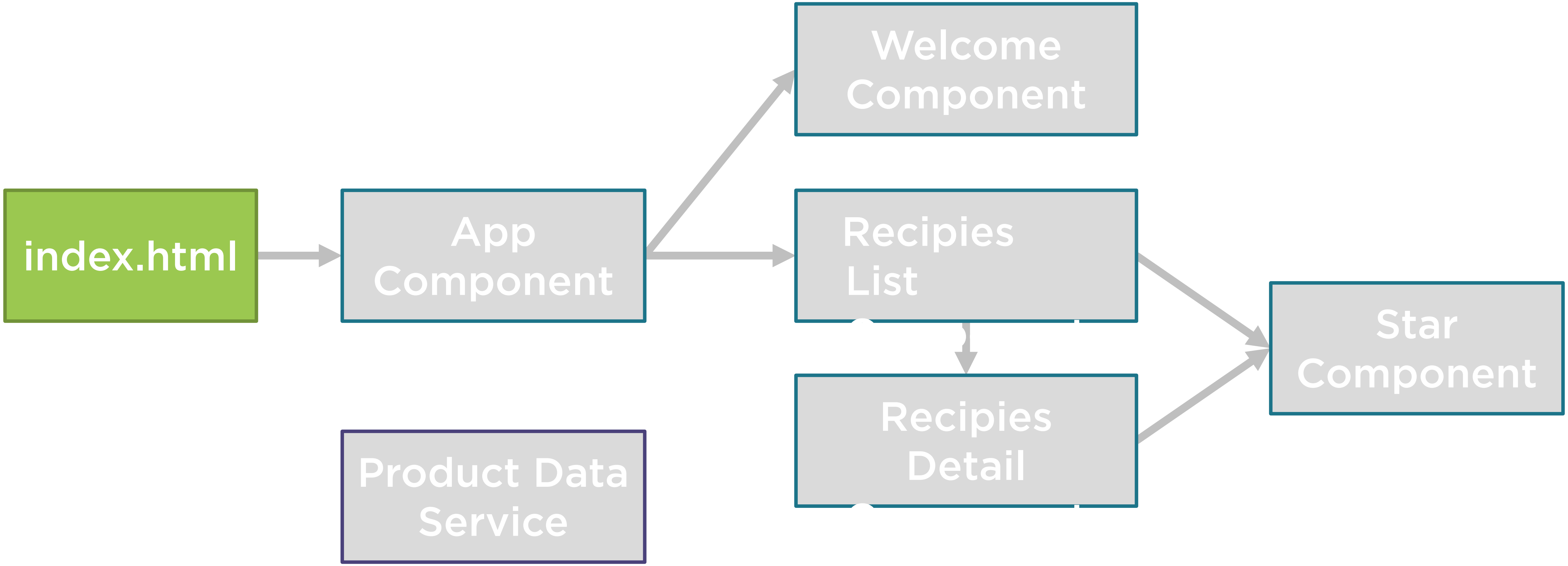
index.html

JavaScript

Application Architecture



Application Architecture



index.html

App Component

Welcome Component

Recipies List

Recipies Detail

Star Component

Product Data Service

Introduction to Components

What Is a Component?

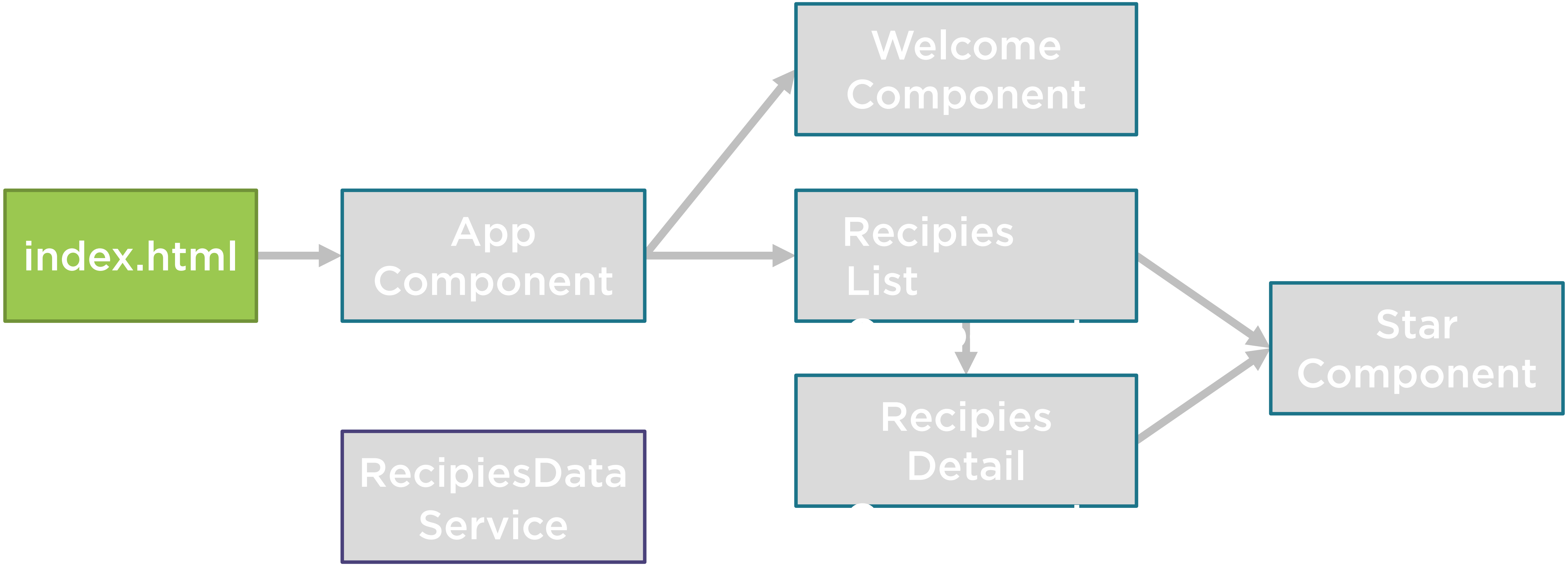
Creating the Component Class

Defining the Metadata with a Decorator

Importing What We Need

Bootstrapping Our App Component

Application Architecture



index.html

App Component

Welcome Component

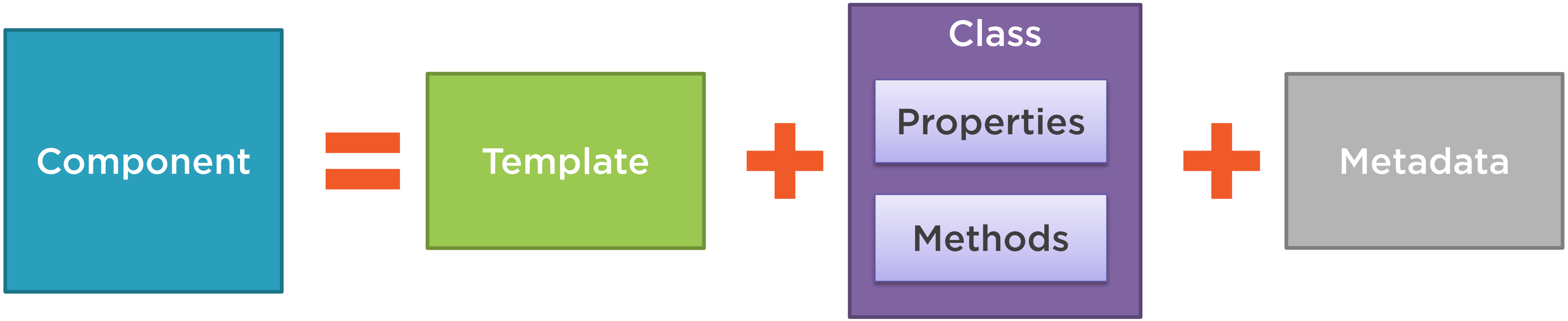
Recipies List

Recipies Detail

Star Component

RecipiesData Service

What Is a Component?



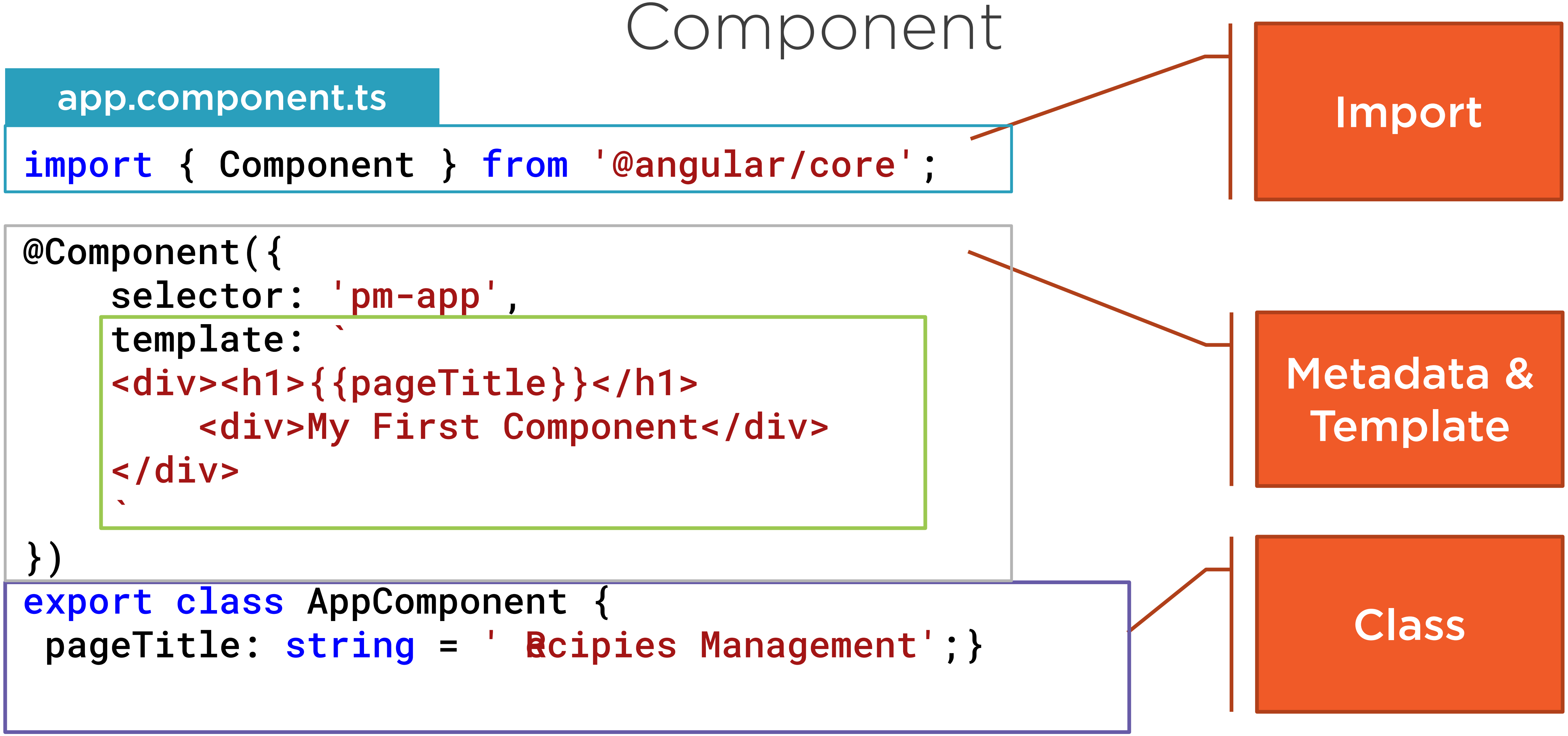
Component = Template + Class (Properties, Methods) + Metadata

- 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



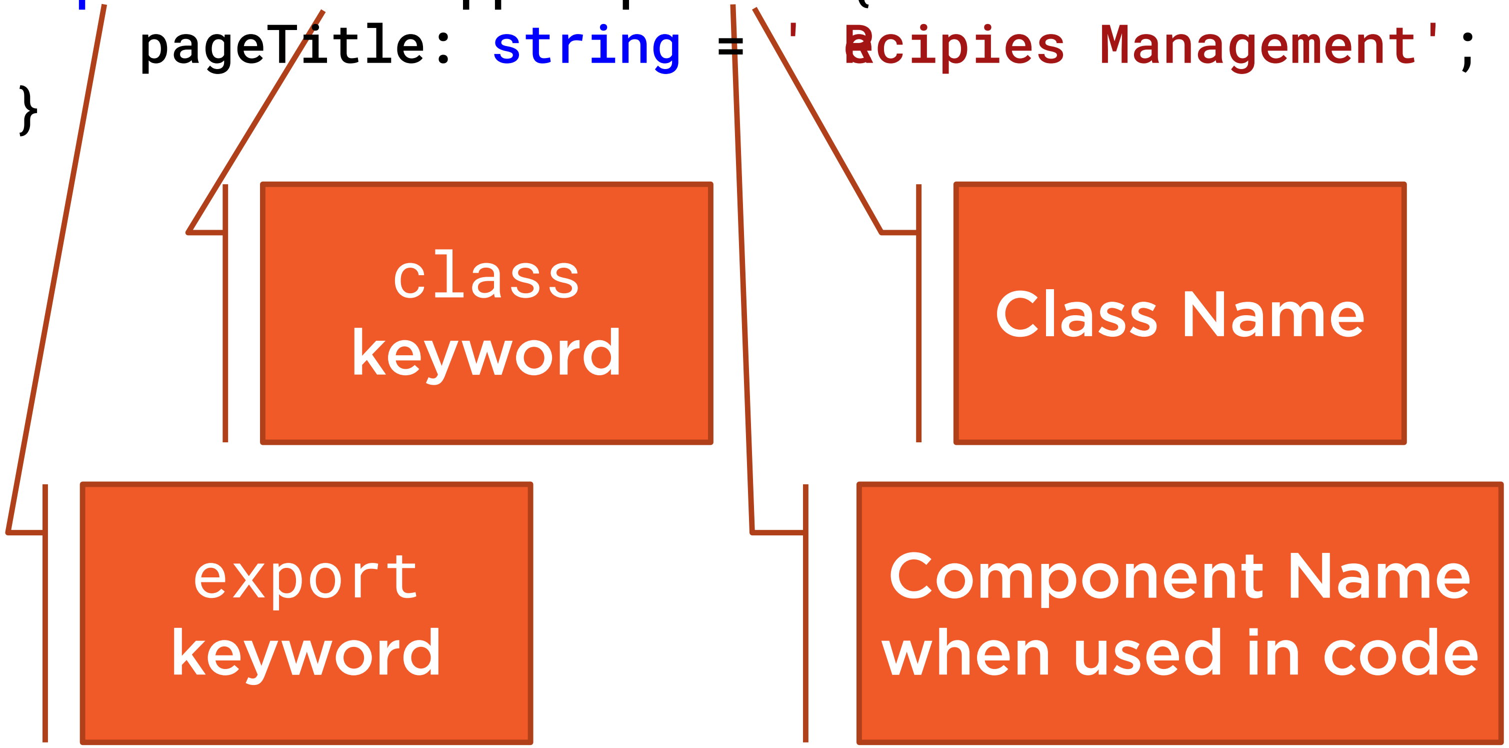
app.component.ts

```
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 = ' Rcipies Management';}
```

Creating the Component Class

app.component.ts

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



class keyword

Class Name

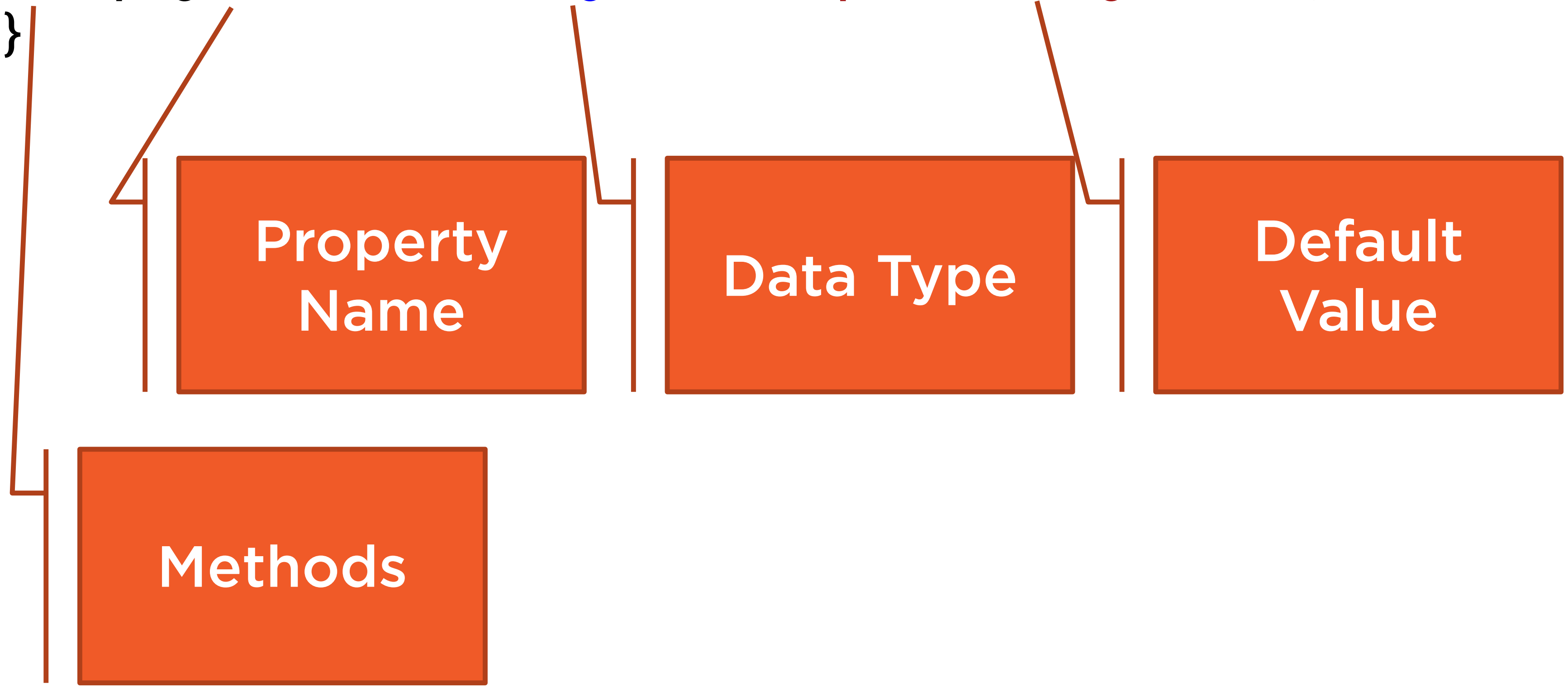
export keyword

Component Name when used in code

Creating the Component Class

app.component.ts

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



Property Name

Data Type

Default Value

Methods

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 = ' Recipies Management';}
```

Decorator

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

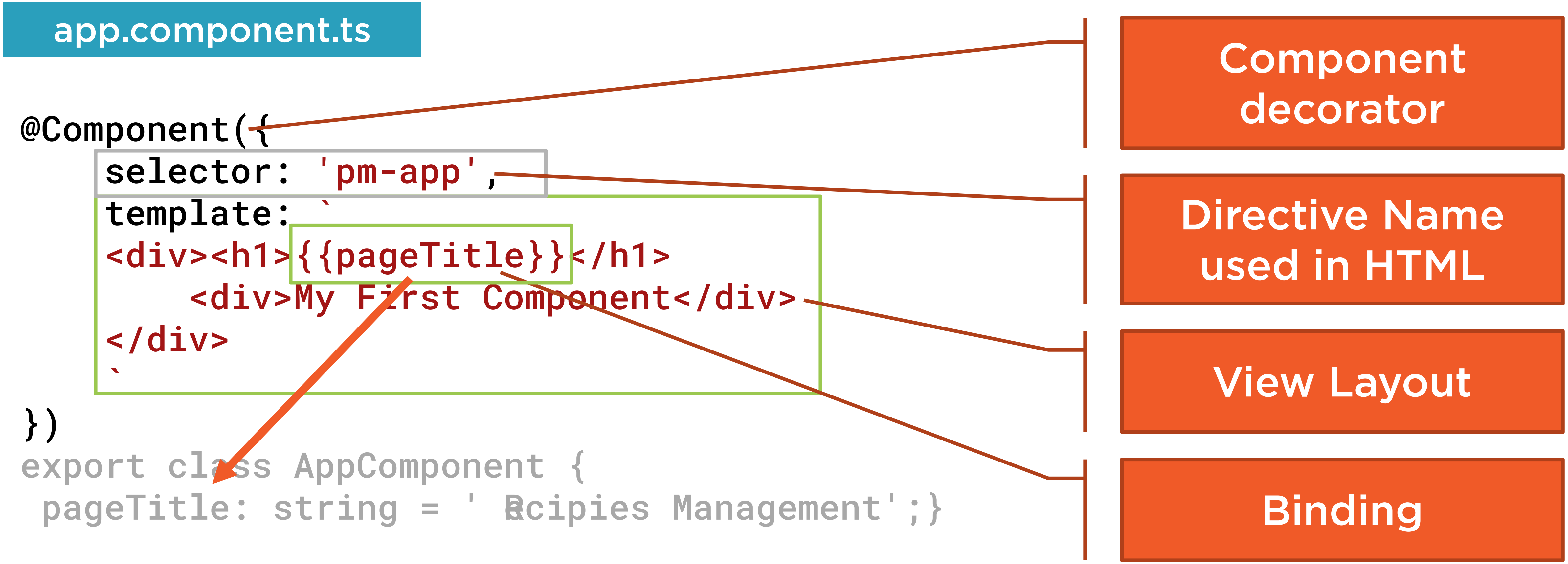
Prefixed with an @.

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 = ' Rcipies Management';}
```

Component decorator

Directive Name used in HTML

View Layout

Binding

Importing What We Need

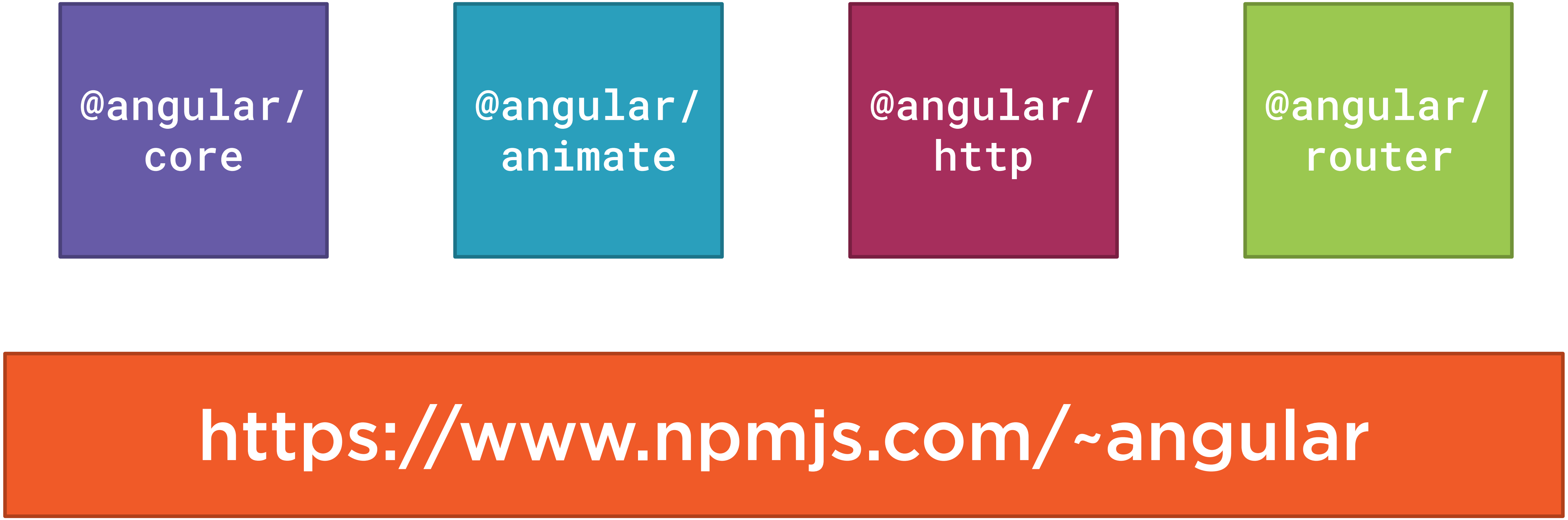
Before we use an external function or class, we define where to find it

`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 Is Modular



@angular/core

@angular/animate

@angular/http

@angular/router

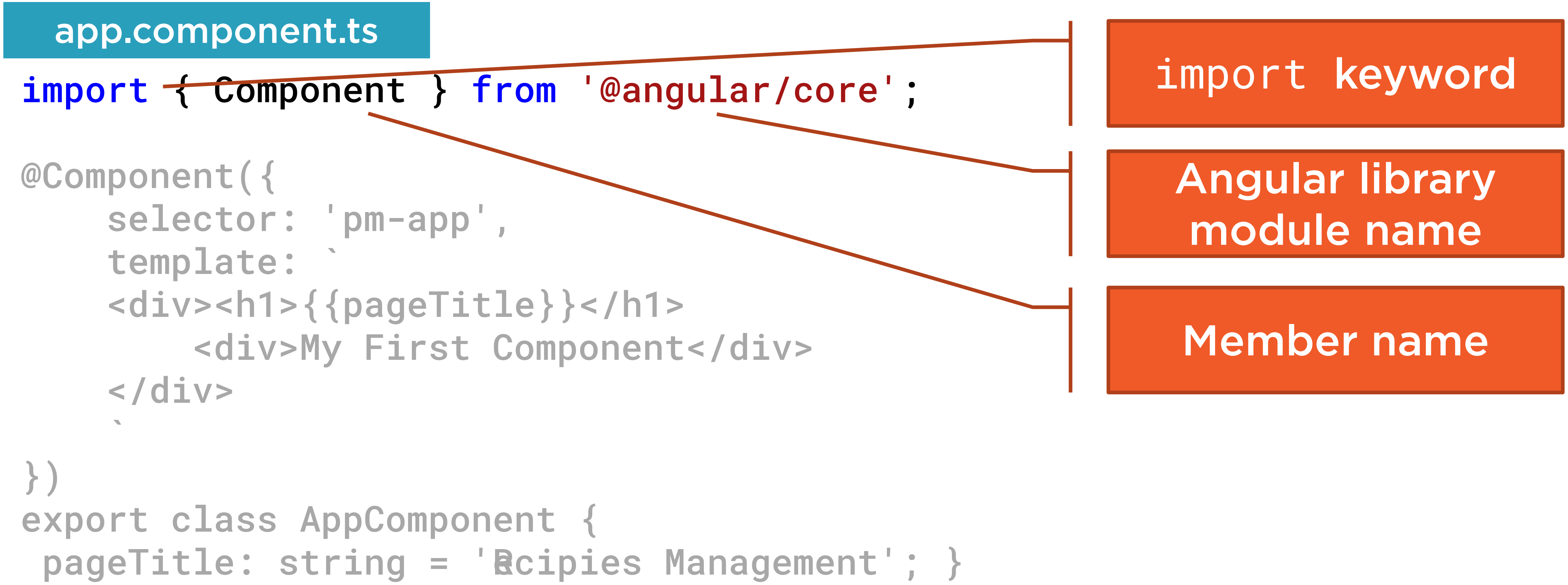
https://www.npmjs.com/~angular

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 = ' Recipies 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 = 'Recipies 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 = ' Recipies Management';}
```

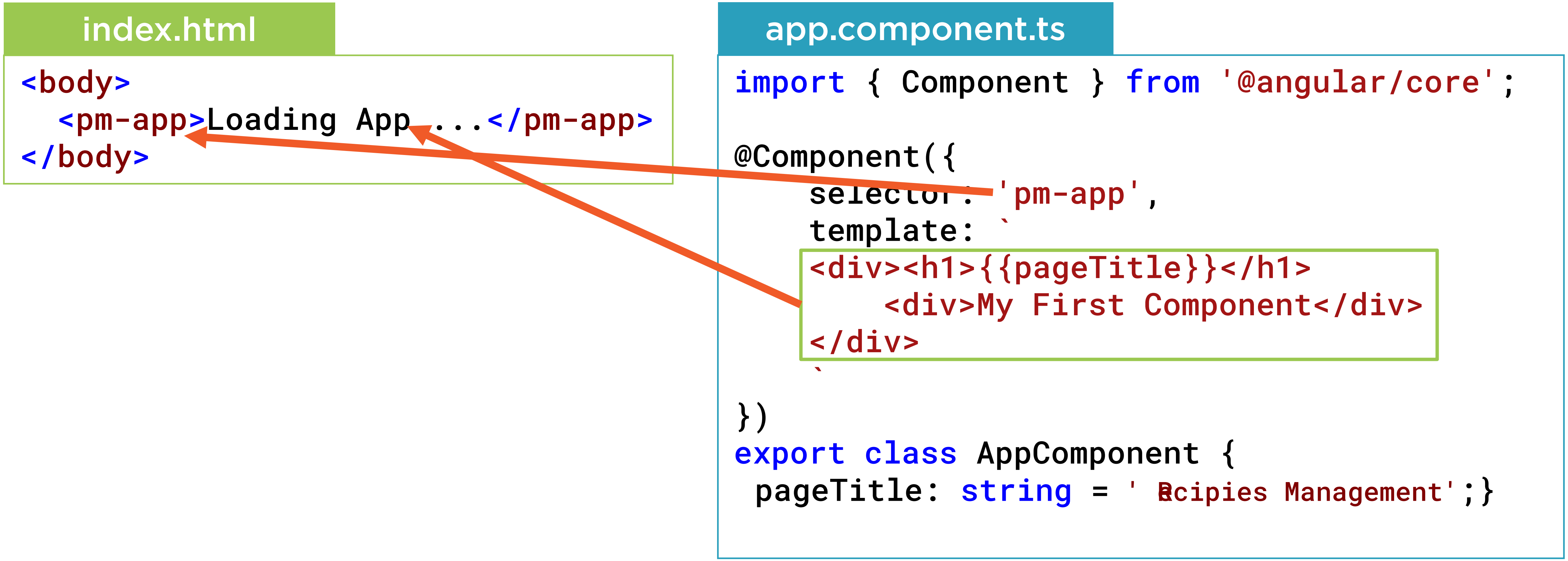
Single Page Application (SPA)

`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

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

app.component.ts

```typescript
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 = ' Rcipies 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'
 },
...
```

main.ts

```
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 { }
```

Component Checklist

Class -> Code

Decorator -> Metadata

Import what we need

Component Checklist: Class

Clear name

- Use CamelCasing
- Append "Component" to the name

`export` **keyword**

Data in properties

- Appropriate data type
- Appropriate default value
- camelCase with first letter lowercase

Logic in methods

- camelCase with first letter lowercase

Component Checklist: Metadata

Component decorator

- Prefix with @; Suffix with ()

`selector`: Component name in HTML

- Prefix for clarity

`template`: View's HTML

- Correct HTML syntax

Component Checklist: Import

Defines where to find the members that this component needs

`import` **keyword**

Member name

- Correct spelling/casing

Module path

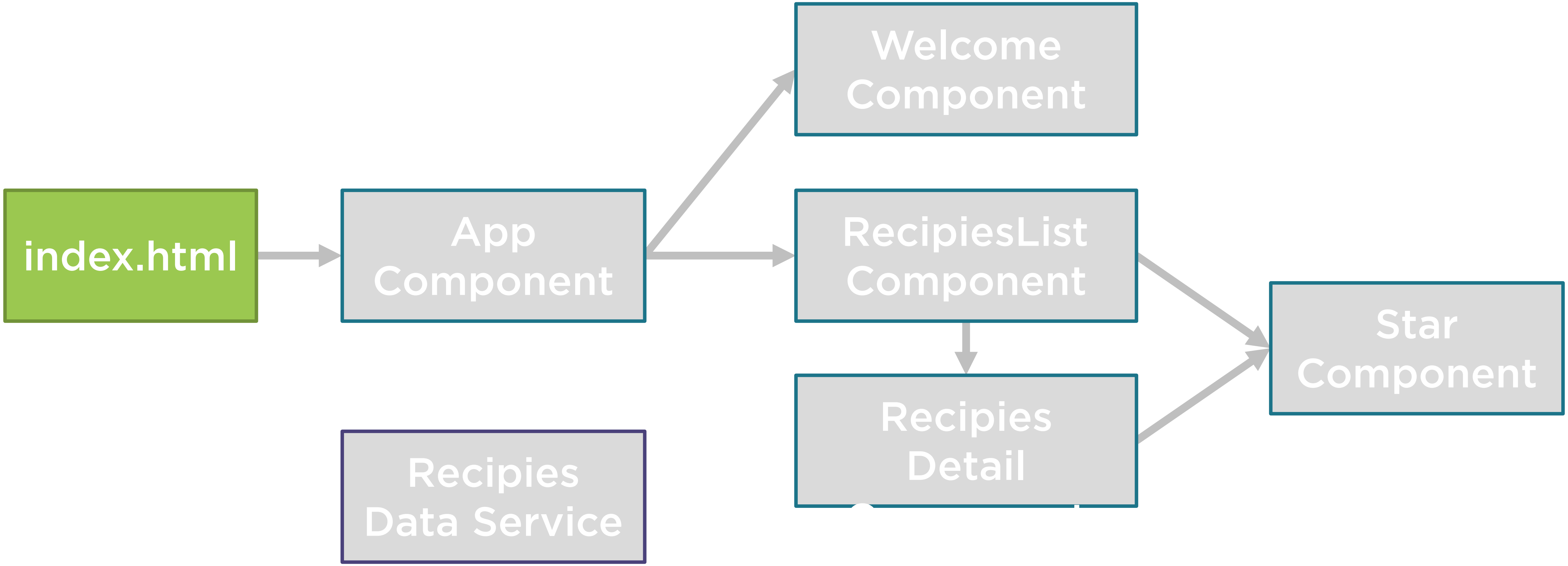
- Enclose in quotes
- Correct spelling/casing

Something's Wrong! Checklist

Recheck your code

- HTML
  - Close tags
  - Angular directives are case sensitive
- TypeScript
  - Close braces
  - TypeScript is case sensitive

Application Architecture



index.html

App Component

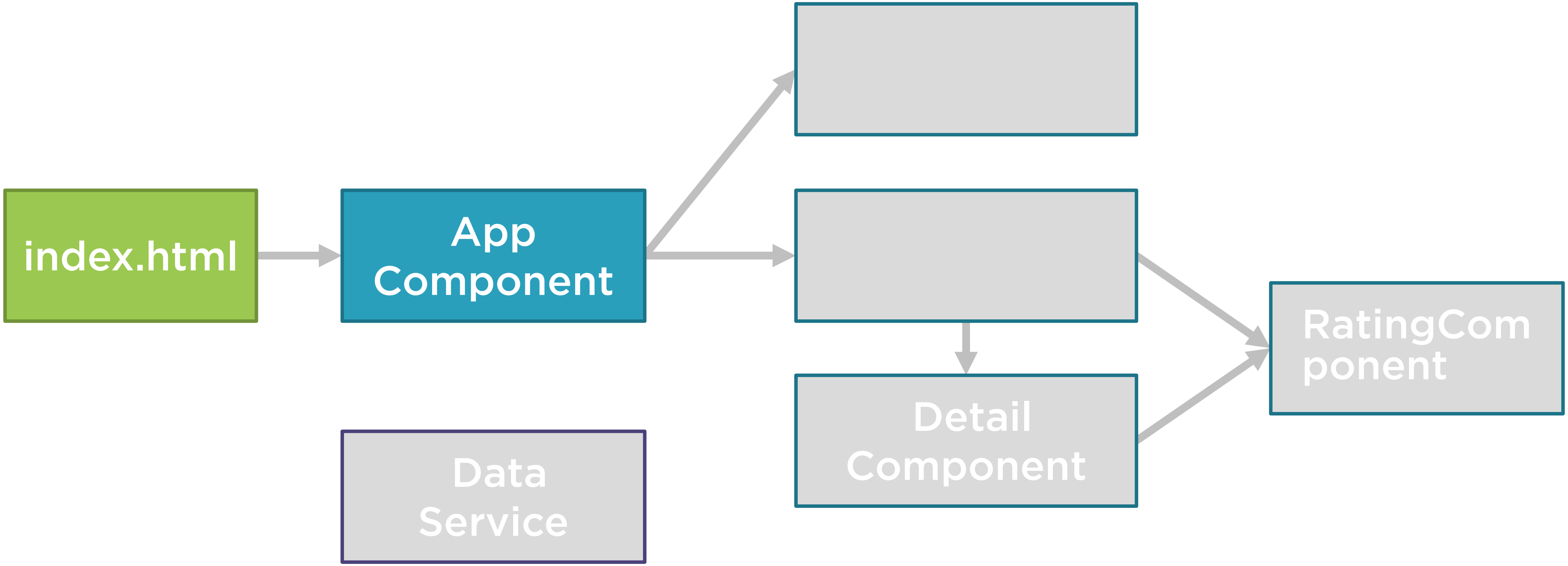
Welcome Component

RecipiesList Component

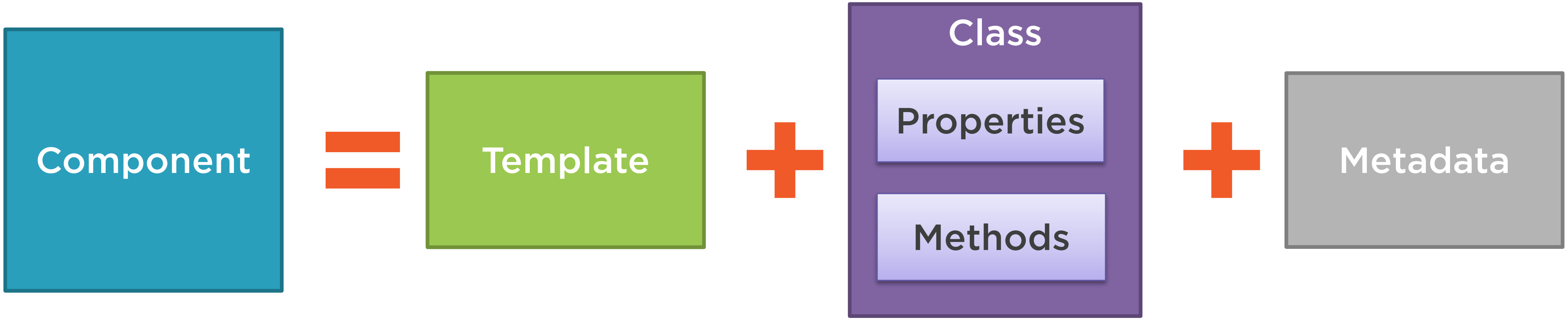
Recipies Detail

Star Component

Recipies Data Service


index.html
App Component
Detail Component
RatingComponent
Data Service

Component



Component = Template + Class (Properties, Methods) + Metadata

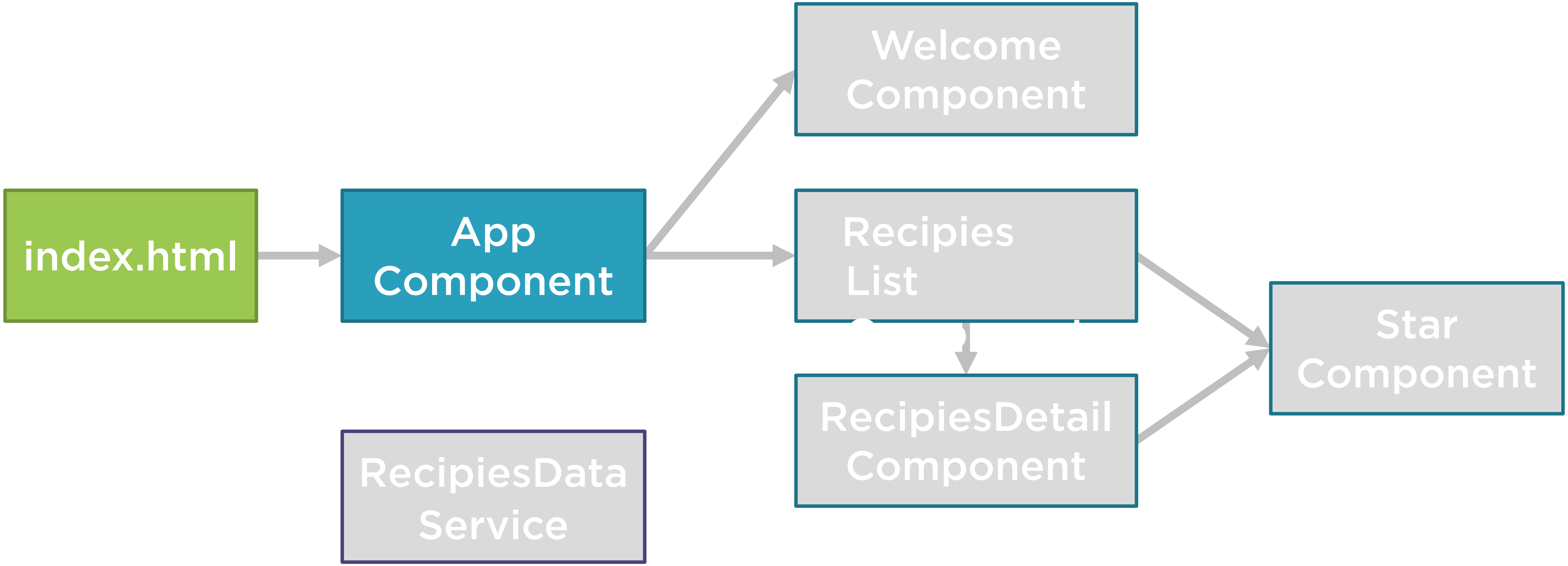
Building a Template

Using a Component as a Directive

Binding with Interpolation

Adding Logic with Directives

Application Architecture



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 = ' Recipies Management';}
```

Defining a Template in a Component

Inline Template

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

Inline Template

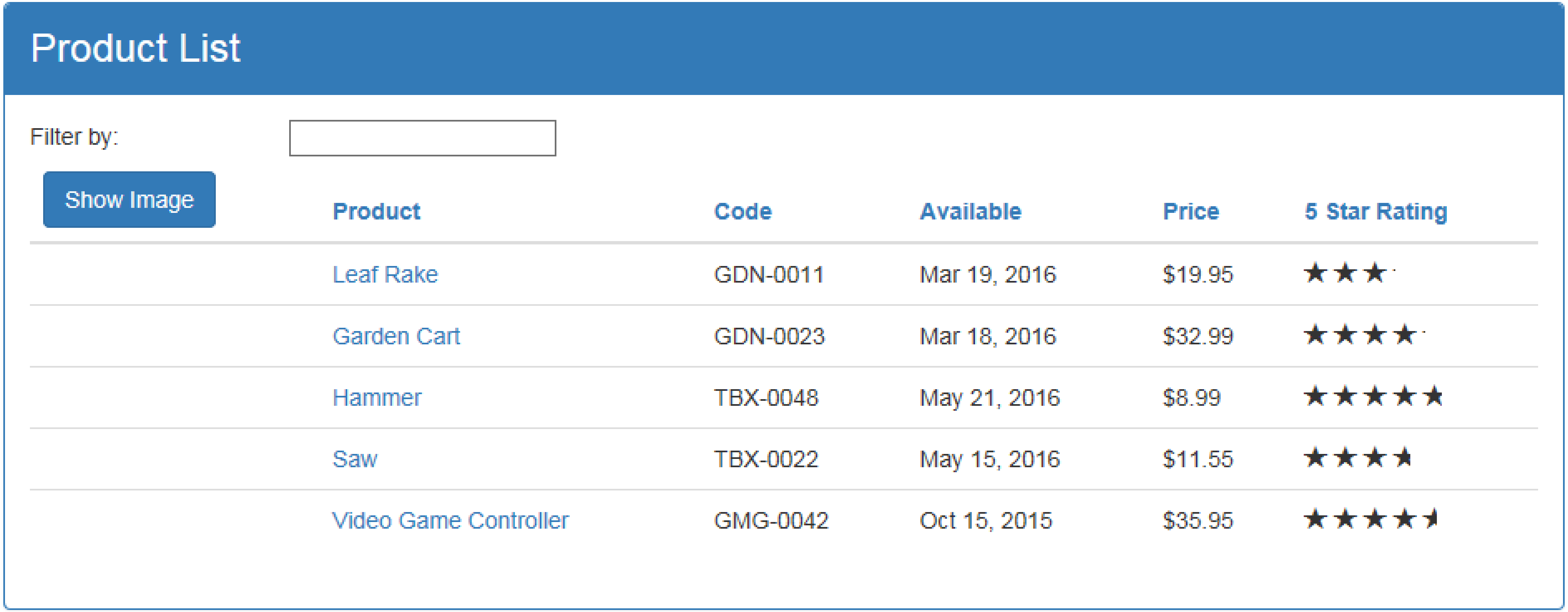
```
template: `
<div>
 <h1>{{pageTitle}}</h1>
 <div>
     My First Component
 </div>
</div>
`
```

ES 2015
Back Ticks

Linked Template

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

Recipies List View



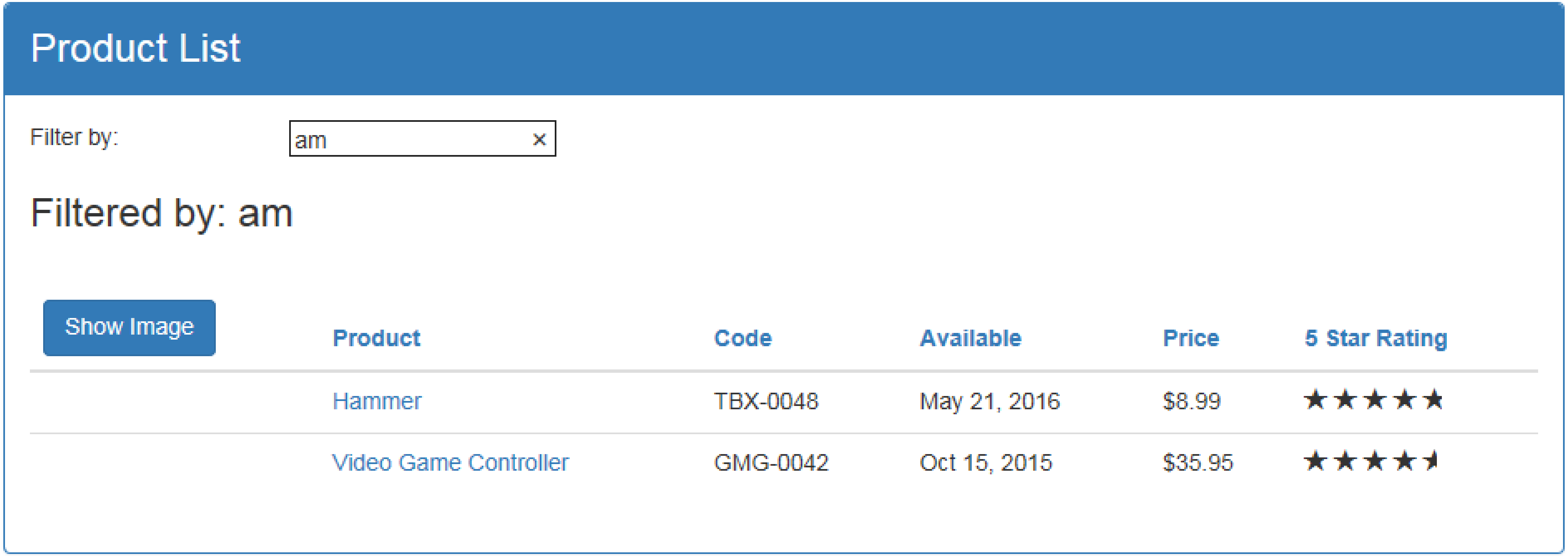
Product List

Filter by:

Show Image

Product	Code	Available	Price	5 Star Rating
Leaf Rake	GDN-0011	Mar 19, 2016	$19.95	★★★
Garden Cart	GDN-0023	Mar 18, 2016	$32.99	★★★★
Hammer	TBX-0048	May 21, 2016	$8.99	★★★★★
Saw	TBX-0022	May 15, 2016	$11.55	★★★★
Video Game Controller	GMG-0042	Oct 15, 2015	$35.95	★★★★★

Recipies List View



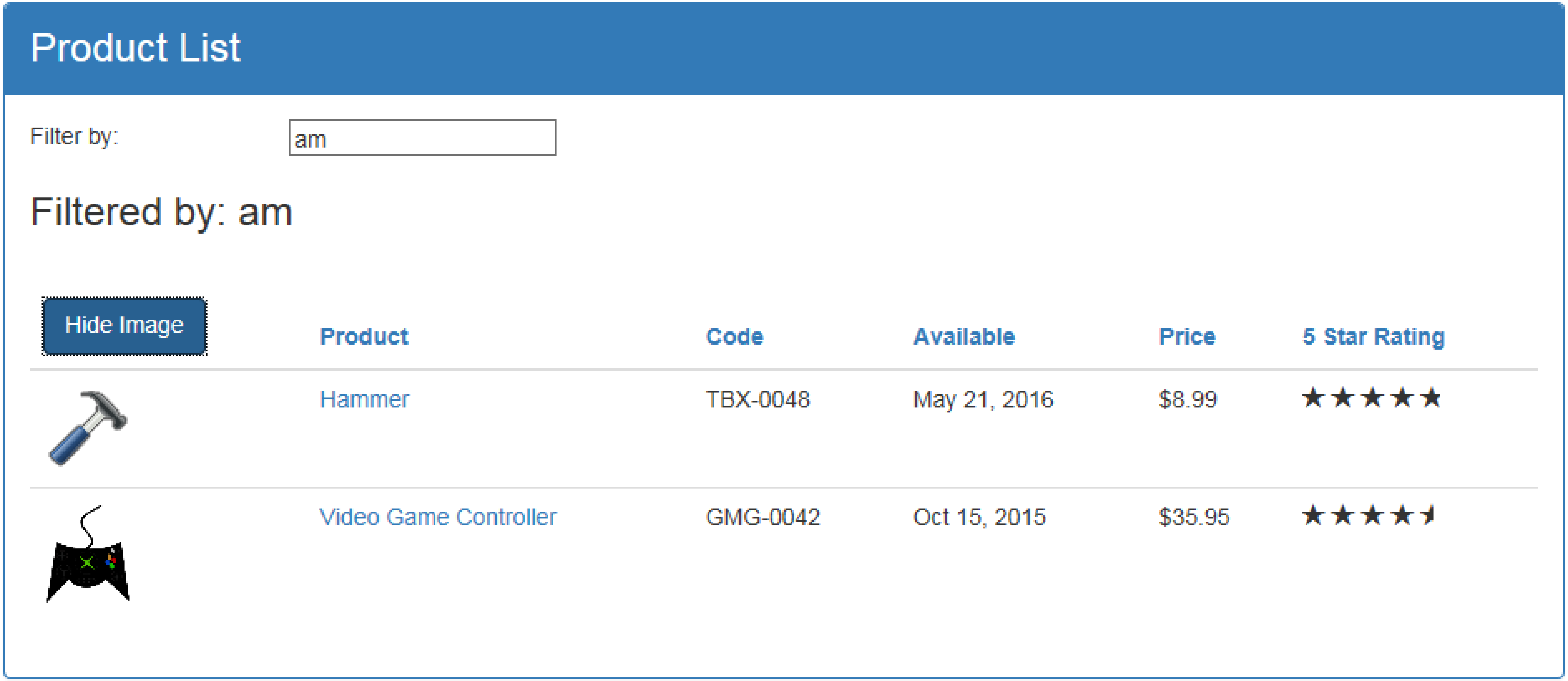
Product List

Filter by: am ×

Filtered by: am

Show Image	Product	Code	Available	Price	5 Star Rating
	Hammer	TBX-0048	May 21, 2016	$8.99	★★★★★
	Video Game Controller	GMG-0042	Oct 15, 2015	$35.95	★★★★⯨

Recipies List View



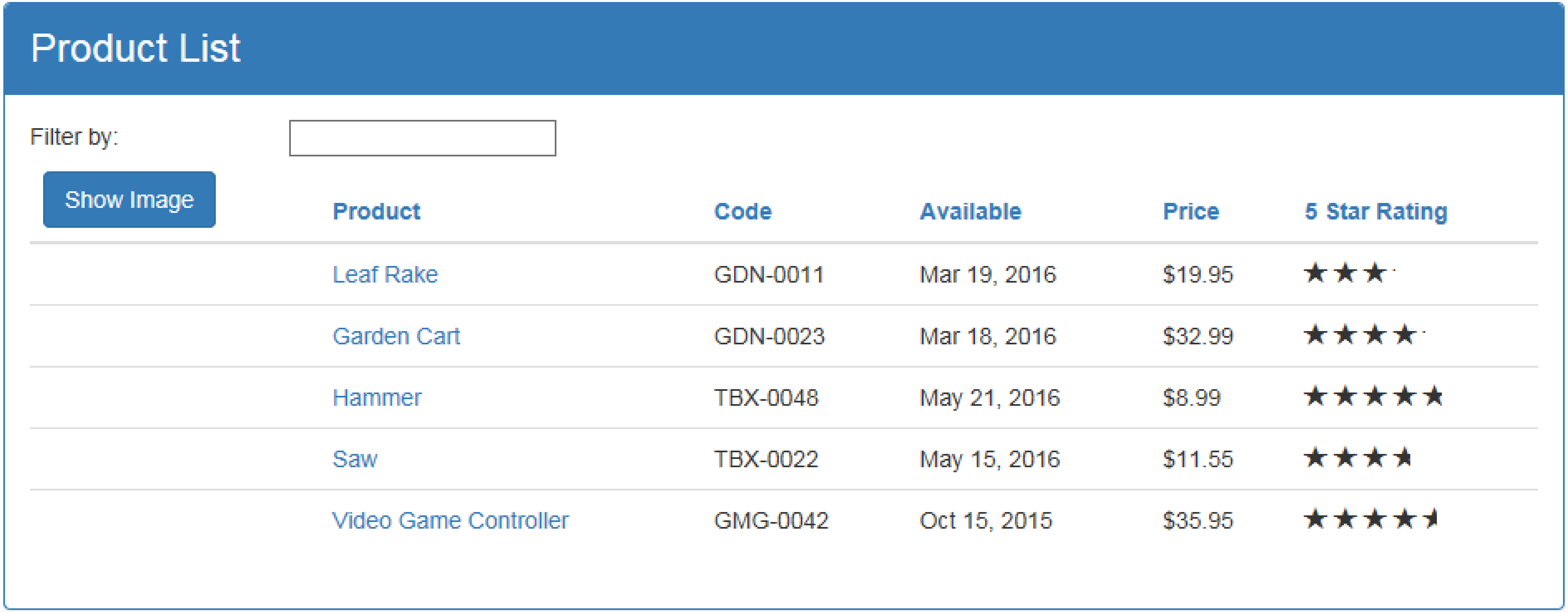
Product List

Filter by: am

Filtered by: am

Hide Image	Product	Code	Available	Price	5 Star Rating
	Hammer	TBX-0048	May 21, 2016	$8.99	★★★★★
	Video Game Controller	GMG-0042	Oct 15, 2015	$35.95	★★★★⯨

RecipiesList View



Product List

Filter by:

Product	Code	Available	Price	5 Star Rating
Leaf Rake	GDN-0011	Mar 19, 2016	$19.95	★★★
Garden Cart	GDN-0023	Mar 18, 2016	$32.99	★★★★
Hammer	TBX-0048	May 21, 2016	$8.99	★★★★★
Saw	TBX-0022	May 15, 2016	$11.55	★★★★
Video Game Controller	GMG-0042	Oct 15, 2015	$35.95	★★★★★

http://getbootstrap.com/

Building the Component

product-list.component.ts

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

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

Using a Component as a Directive

app.component.ts

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

product-list.component.ts

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

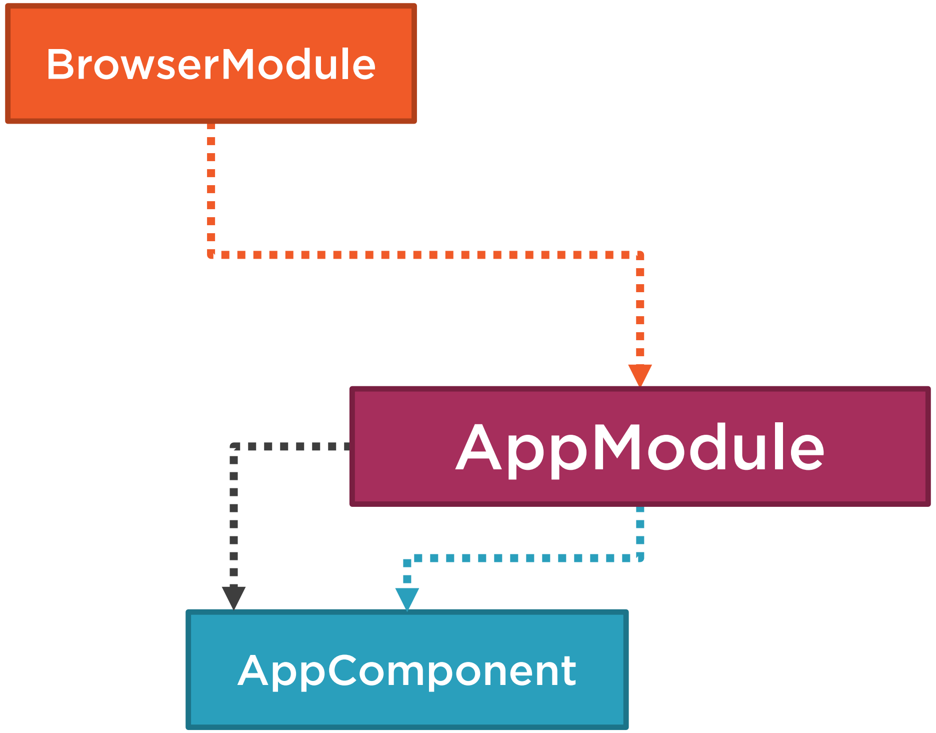
Using a Component as a Directive

app.component.ts

```
@Component({
 selector: 'pm-app',
 template: `
  <div><h1>{{pageTitle}}</h1>
1   <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 { }
```



BrowserModule

AppModule

AppComponent

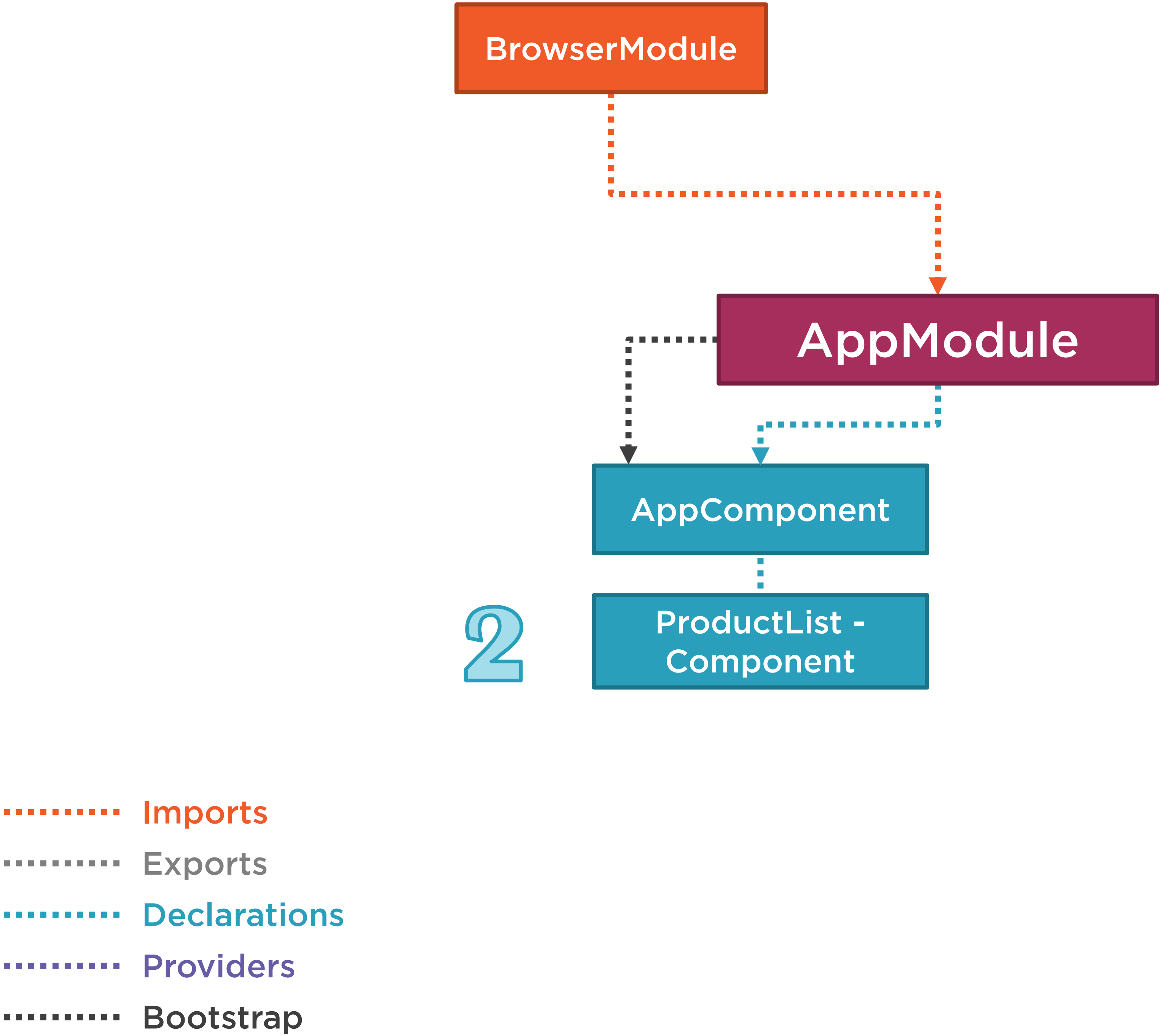
········· Imports

········· Exports

········· Declarations

········· Providers

········· Bootstrap


BrowserModule
AppModule
AppComponent
2
ProductList - Component
Imports
Exports
Declarations
Providers
Bootstrap

Binding

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

Template
Class
Properties
Methods

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 { }
```

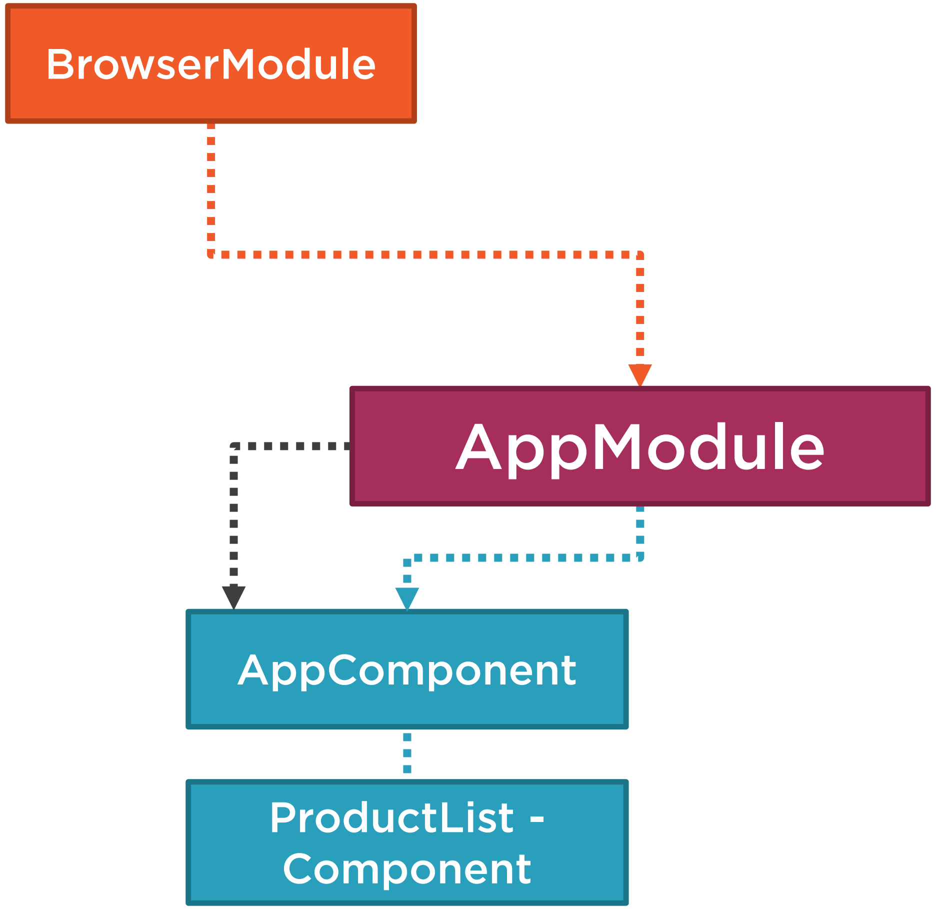
Angular Built-in Directives

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>
```



┅┅┅ Imports

┅┅┅ Exports

┅┅┅ Declarations

┅┅┅ Providers

┅┅┅ Bootstrap

*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>
```

Template input variable

for...of vs for...in

for...of

- 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>
```

Checklist: Template

Inline template

- For short templates
- Specify the **`template`** property
- Use the ES 2015 back ticks for multiple lines
- Watch syntax

Linked template

- For longer templates
- Specify the **`templateUrl`** property
- Define the path to the HTML file

Checklist: Component as a Directive

product-list.component.ts

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

app.component.ts

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

app.module.ts

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

Checklist: Interpolation

One way binding

- From component class property to an element property.

Defined with double curly braces

- Contains a template expression
- No quote needed

Checklist: Structural Directives

`*ngIf` **and** `*ngFor`

- Prefix with an asterisk
- Assign to a quoted string expression

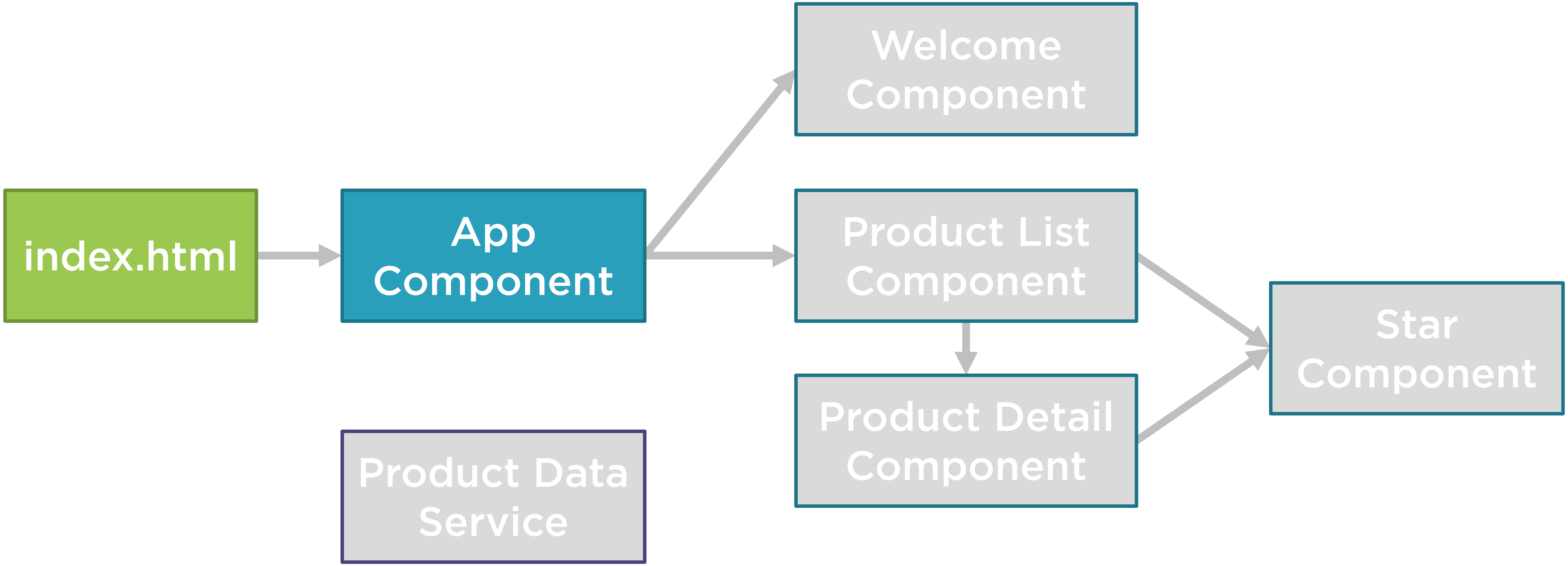
`*ngIf`

- Expression is evaluated as a true or false value

`*ngFor`

- Define the local variable with `let`
- Specify 'of': **`'let product of products'`**

Application Architecture



index.html

App Component

Welcome Component

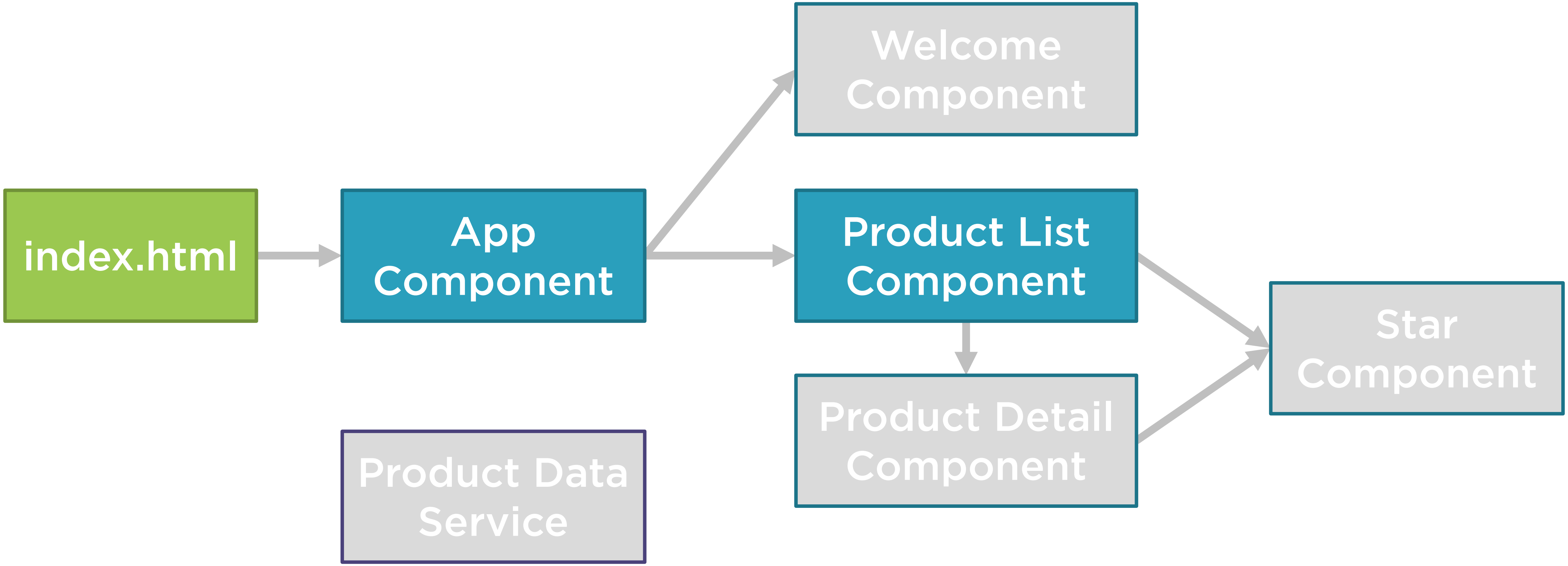
Product List Component

Product Detail Component

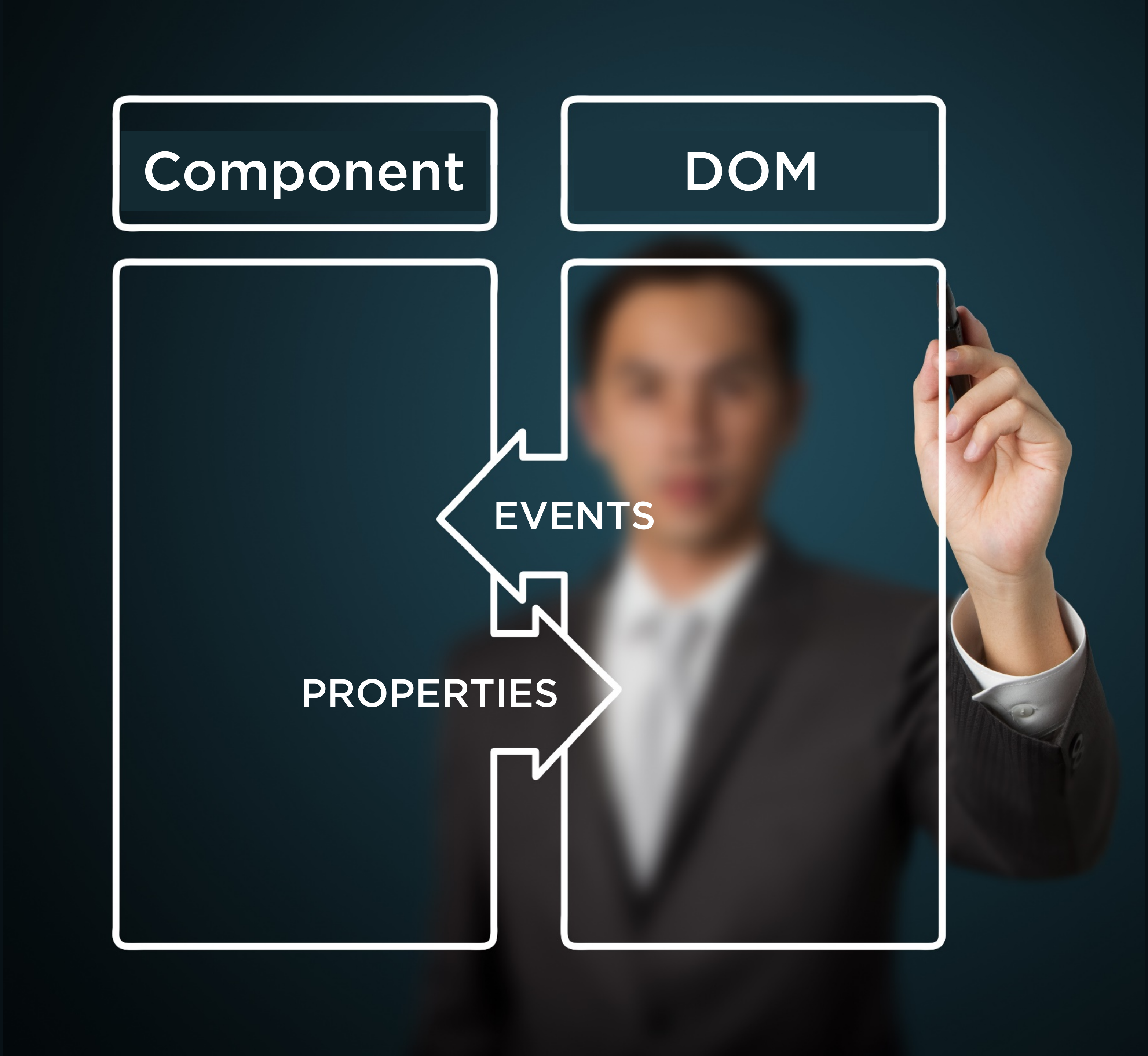
Star Component

Product Data Service

Application Architecture



Data Binding & Pipes



Component

DOM

EVENTS

PROPERTIES

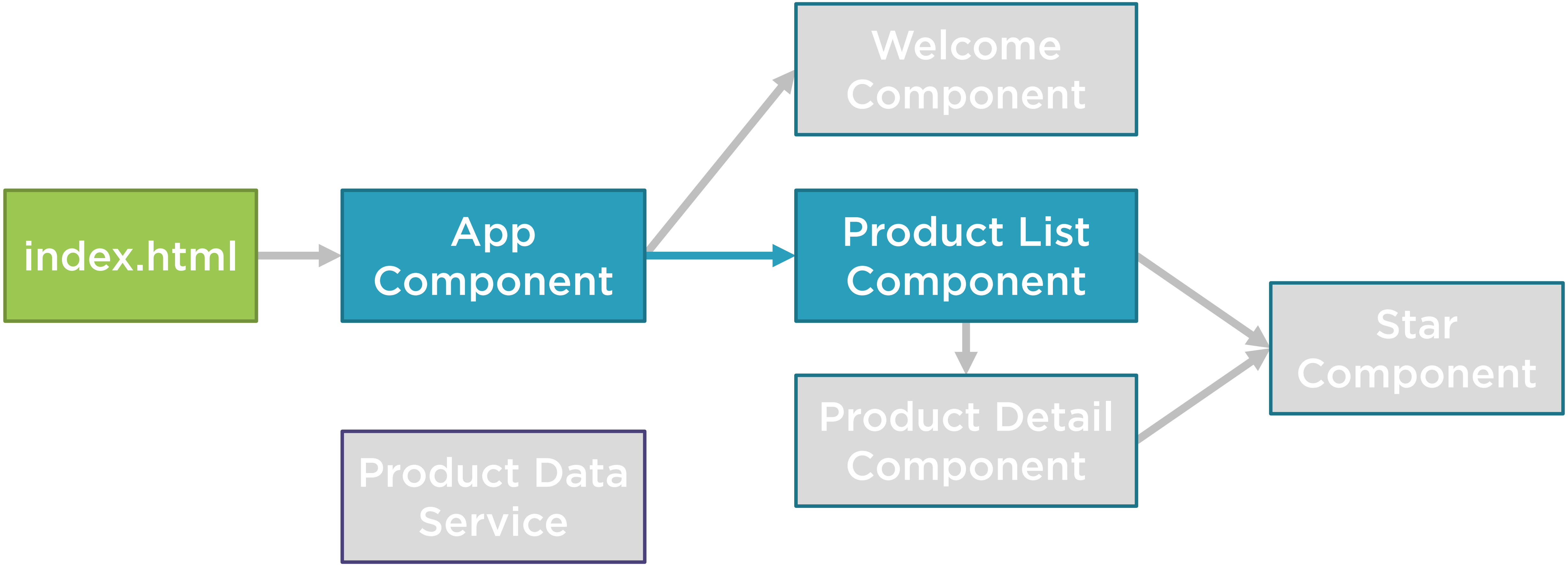
Property Binding

Handling Events with Event Binding

Handling Input with Two-way Binding

Transforming Data with Pipes

Application Architecture



Property Binding

```
<img [src]='product.imageUrl'>

<img src={{product.imageUrl}}>

penc                  t.imageUrl}}'>
```

Element Property

Template Expression

Event Binding

Template

Class

```
<h1>{{pageTitle}}</h1>
<img [src]='product.imageUrl'>
<button (click)='toggleImage()'>
```

```
export class ListComponent {
```

https://developer.mozilla.org/en-US/docs/Web/Events

( )
Target Event

' '
Template Statement

Two-way Binding

Template

Class

```
<input [(ngModel)]='listFilter'>
```

```
export class ListComponent {
 listFilter: string = 'cart';
}
```

Two-way Binding

Template

Class

```
<input [(ngModel)]='listFilter'>
```

```
export class ListComponent {
 listFilter: string = 'cart';
}
```

Two-way Binding

Template

Class

```
<input [(ngModel)]='listFilter'>
```

```
export class ListComponent {
 listFilter: string = 'cart';
}
```

Two-way Binding

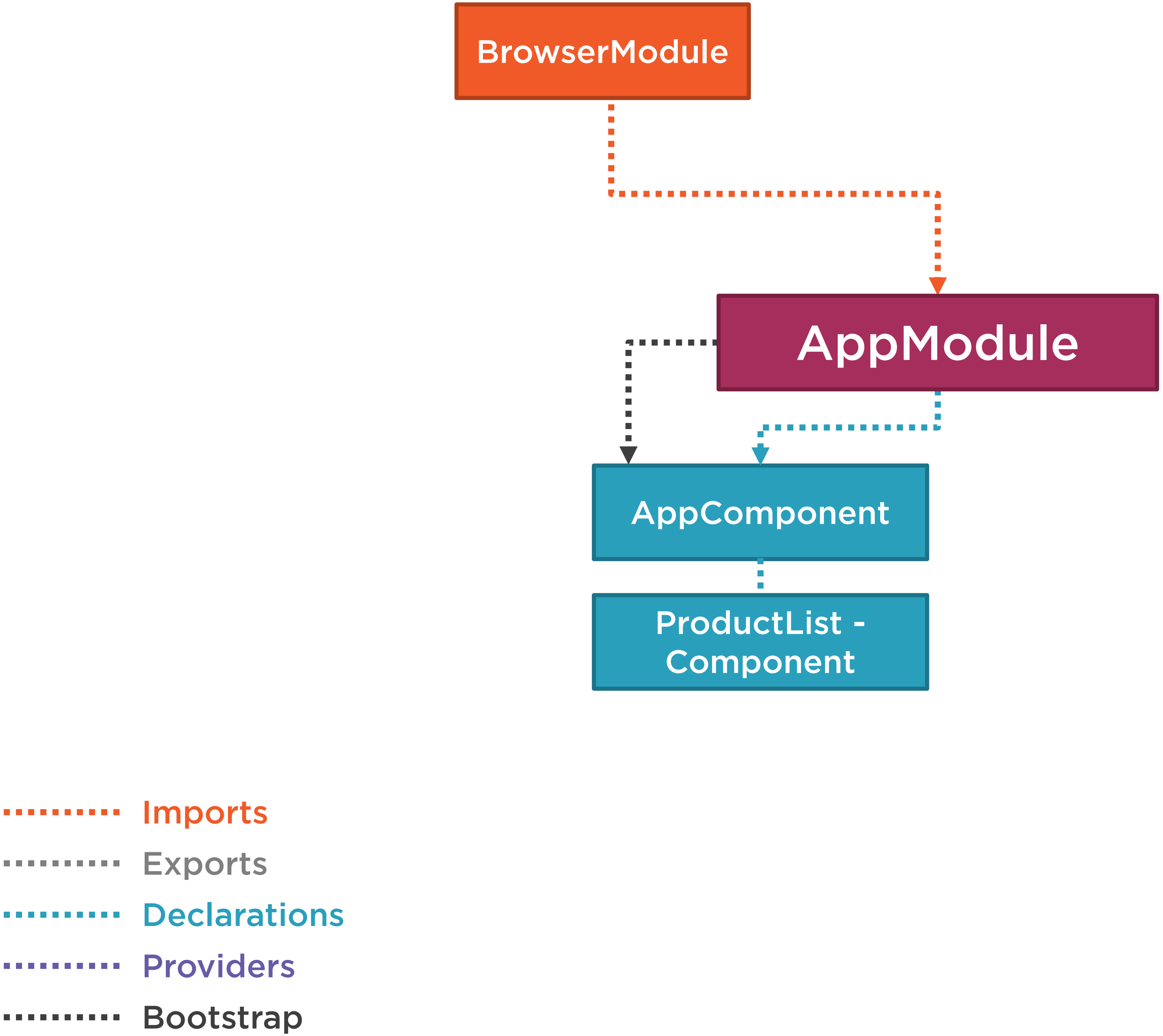
Template

Class

```
<input [(ngModel)]='listFilter'>
```

```
export class ListComponent {
 listFilter: string = 'cart';
}
```

[()]
Banana in a Box



BrowserModule

AppModule

AppComponent

ProductList - Component

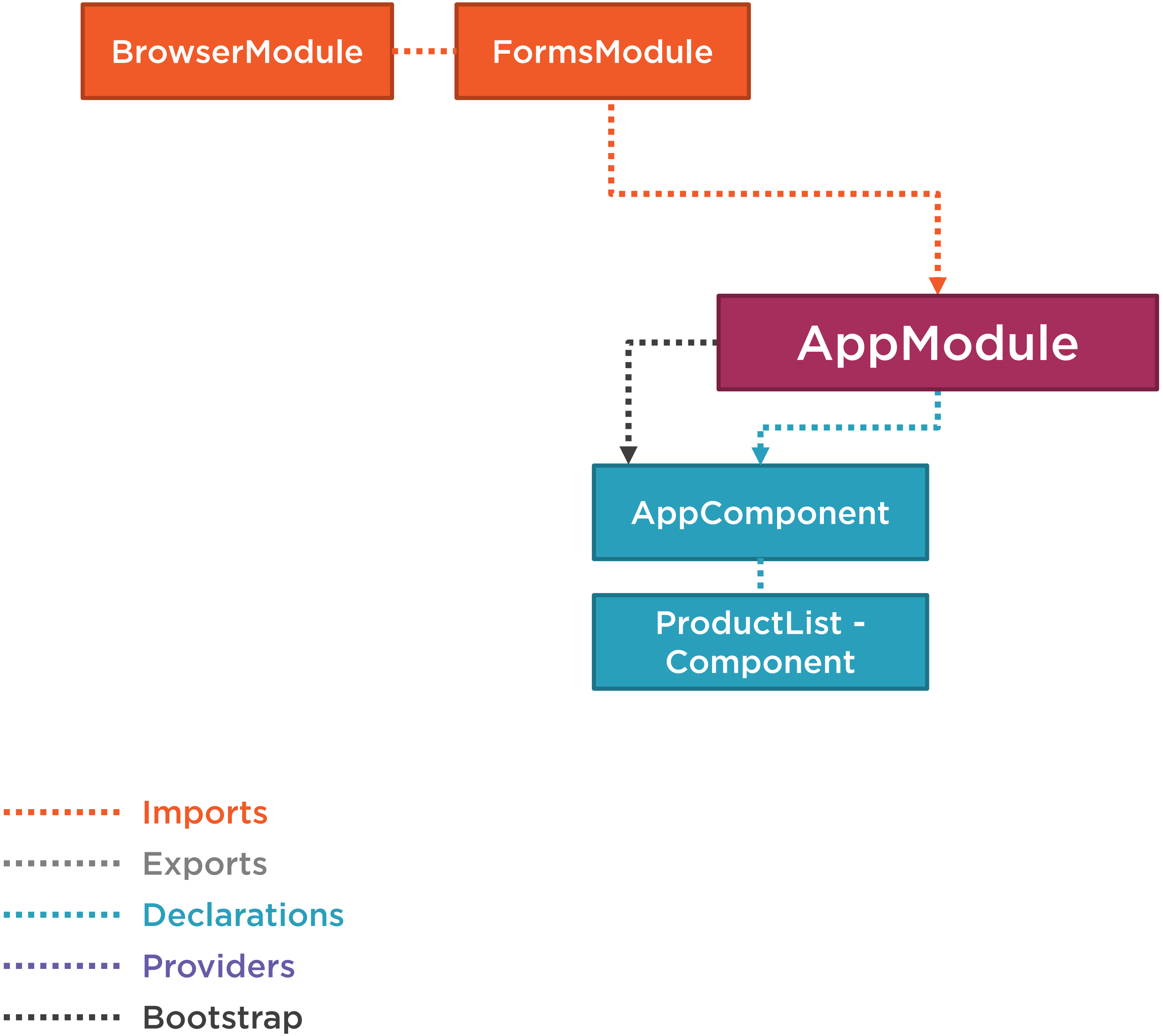
Imports

Exports

Declarations

Providers

Bootstrap



BrowserModule

FormsModule

AppModule

AppComponent

ProductList - Component

Imports

Exports

Declarations

Providers

Bootstrap

Transforming Data with Pipes

Transform bound properties before display

Built-in pipes

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

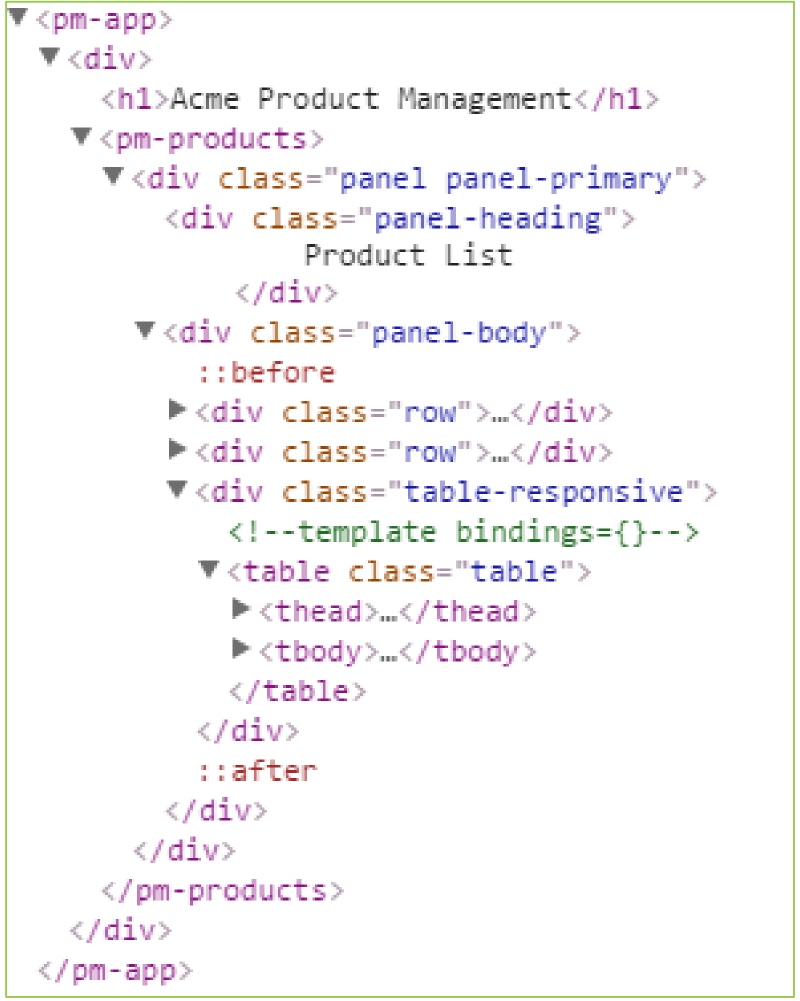
Custom pipes

Pipe Examples

```
{{ product.productCode | lowercase }}

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

{{ product.price | currency | lowercase }}

{{ product.price | currency:'USD':true:'1.2-2' }}
```

Data Binding

DOM

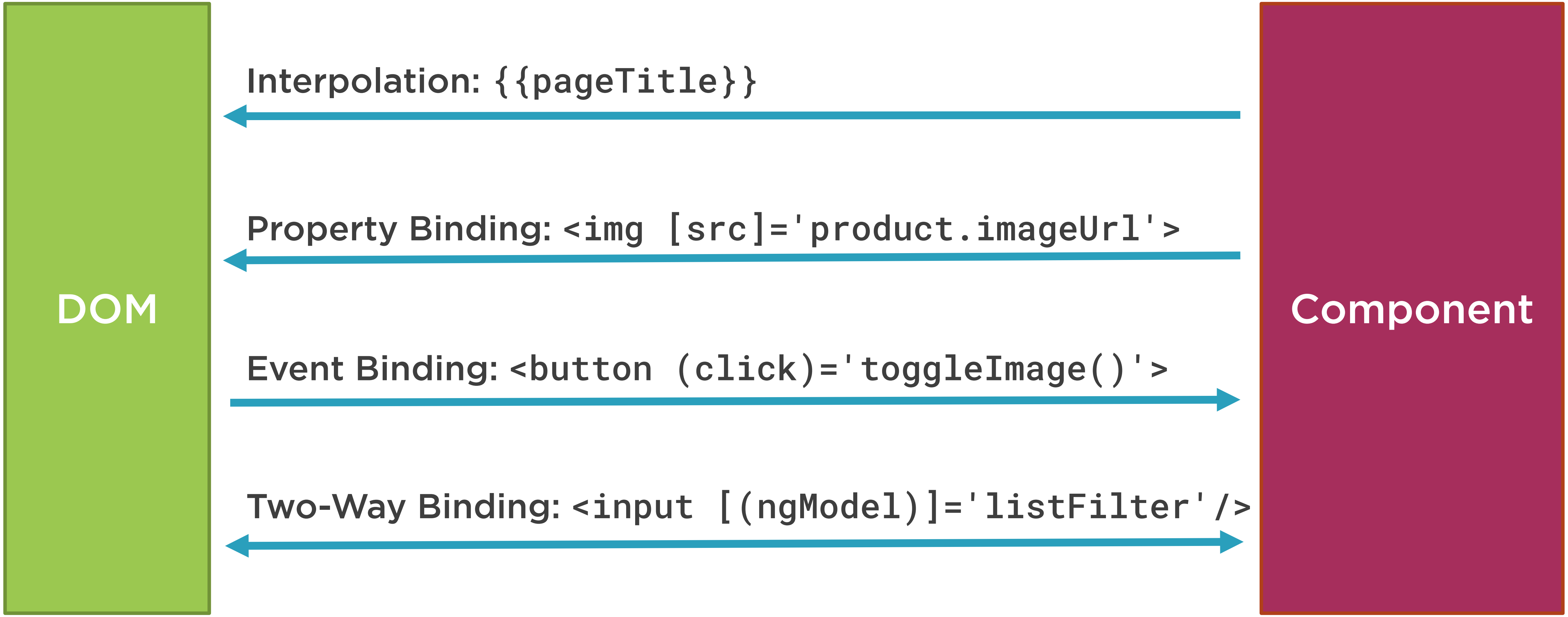


```
▼<pm-app>
  ▼<div>
      <h1>Acme Product Management</h1>
    ▼<pm-products>
      ▼<div class="panel panel-primary">
          <div class="panel-heading">
                Product List
              </div>
        ▼<div class="panel-body">
            ::before
          ▶<div class="row">…</div>
          ▶<div class="row">…</div>
          ▼<div class="table-responsive">
              <!--template bindings={}-->
            ▼<table class="table">
              ▶<thead>…</thead>
              ▶<tbody>…</tbody>
              </table>
            </div>
            ::after
          </div>
        </div>
      </pm-products>
    </div>
  </pm-app>
```

product-list.component.ts

```
@Component({
  selector: 'pm-products',
  templateURL: 'product-list.component.html'
})
export class ProductListComponent {
  pageTitle: string = 'Product List';
  listFilter: string = 'cart';
  products: any[] = […];
  toggleImage(): void {…}
}
```

Data Binding



DOM

Interpolation: {{pageTitle}}

Property Binding: <img [src]='product.imageUrl'>

Event Binding: <button (click)='toggleImage()'>

Two-Way Binding: <input [(ngModel)]='listFilter'/>

Component

Checklist: ngModel

product-list.component.html

```html
<div class='col-md-4'>
  <input type='text'
      [(ngModel)]='listFilter' />
</div>
```

app.module.ts

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

Checklist: Pipes

Pipe character |

Pipe name

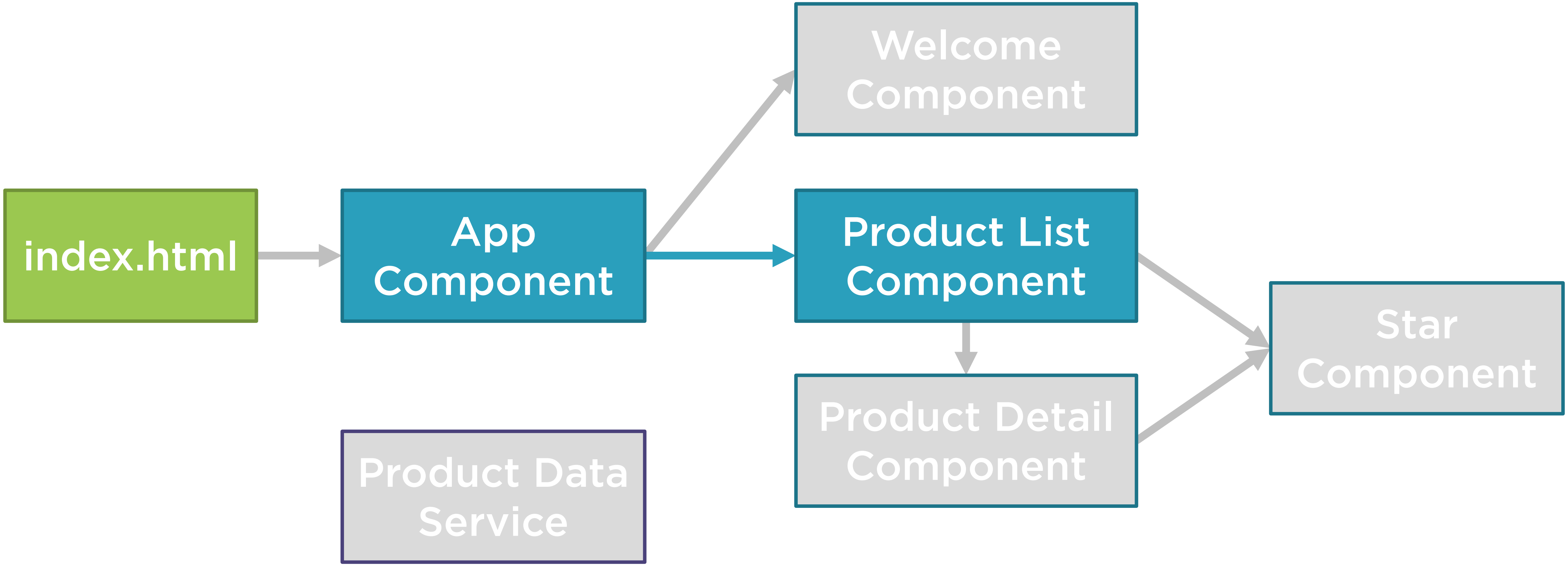
Pipe parameters

- Separated with colons

Example

-
  ```
  {{ product.price |
      currency:'USD':true:'1.2-2' }}
  ```

Application Architecture



index.html

App Component

Welcome Component

Product List Component

Product Detail Component

Star Component

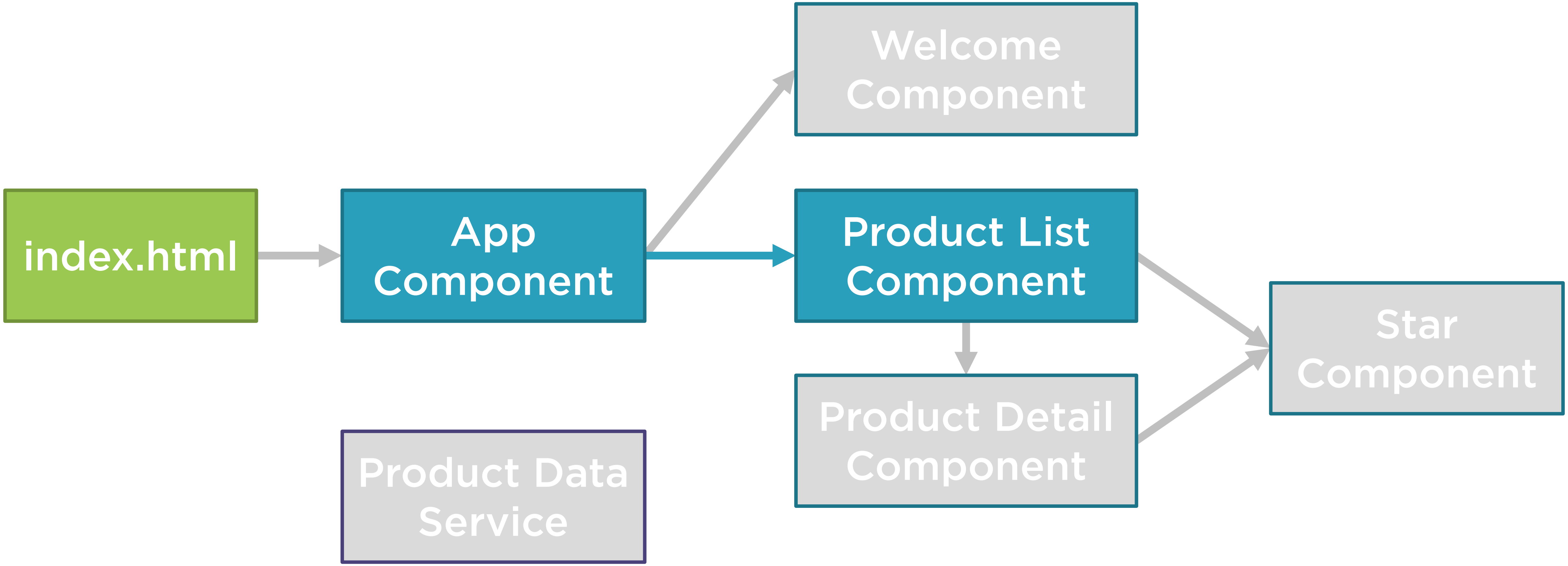
Product Data Service

More on Components

Improving Our Components

- Strong typing & interfaces
- Encapsulating styles
- Lifecycle hooks
- Custom pipes
- Relative Paths with Module Id

Application Architecture



Strong Typing

```
export class ProductListComponent {
   pageTitle: string = 'Product List';
   showImage: boolean = false;
   listFilter: string = 'cart';
   message: string;

   products: any[] = […];

   toggleImage(): void {
      this.showImage = !this.showImage;
   }

   onRatingClicked(message: string): void {
        this.message = message;
   }
```

Interface

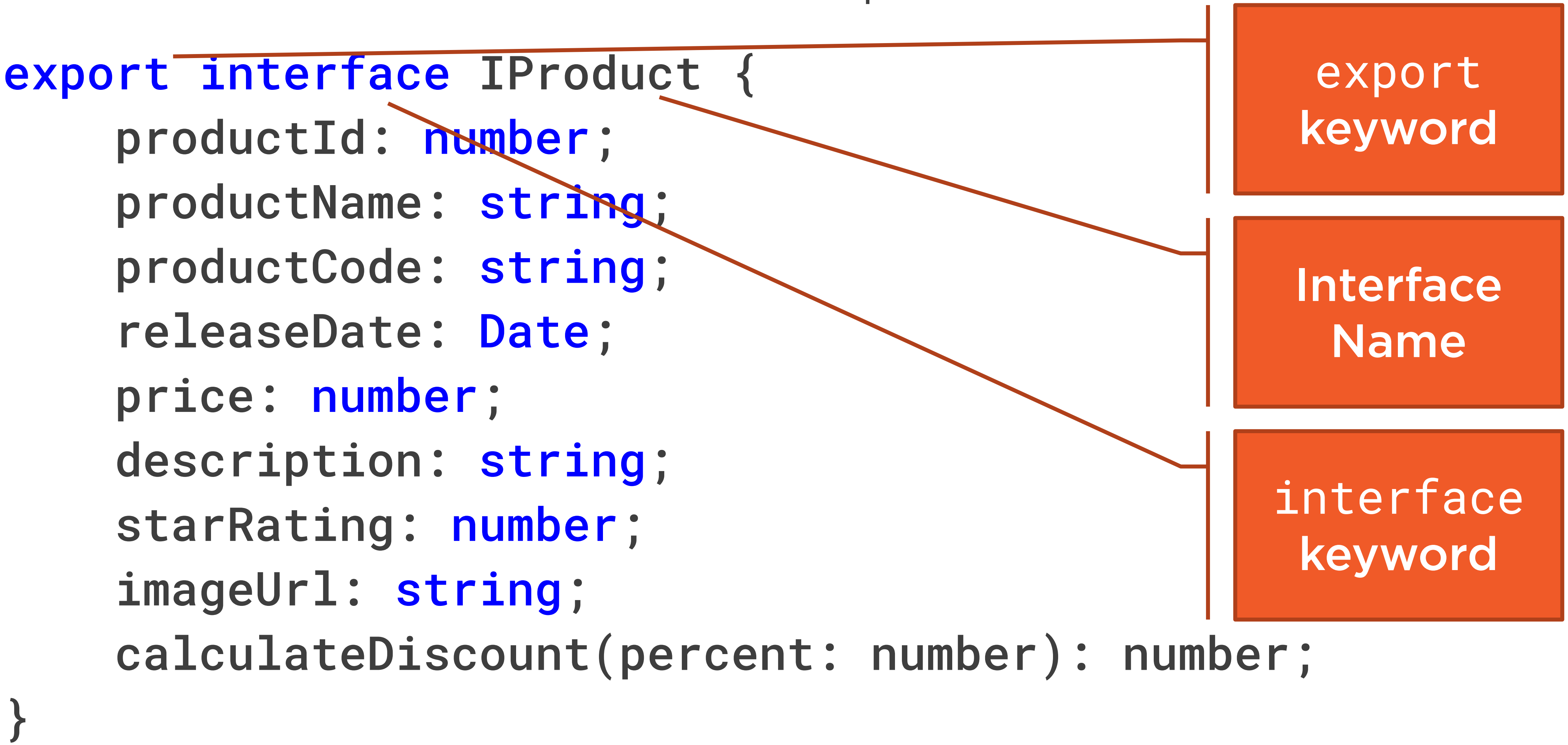
A specification identifying a related set of properties and methods.

A class commits to supporting the specification by implementing the interface.

Use the interface as a data type.

Development time only!

Interface Is a Specification



```
export interface IProduct {
    productId: number;
    productName: string;
    productCode: string;
    releaseDate: Date;
    price: number;
    description: string;
    starRating: number;
    imageUrl: string;
    calculateDiscount(percent: number): number;
}
```

export keyword

Interface Name

interface keyword

Using an Interface as a Data Type

```
import { IProduct } from './product';

export class ProductListComponent {
  pageTitle: string = 'Product List';
  showImage: boolean = false;
  listFilter: string = 'cart';

  products: IProduct[] = […];

  toggleImage(): void {
      this.showImage = !this.showImage;
   }
}
```

Handling Unique Component Styles

Templates sometimes require unique styles

We can inline the styles directly into the HTML

We can build an external stylesheet and link it in index.html

There is a better way!

Encapsulating Component Styles

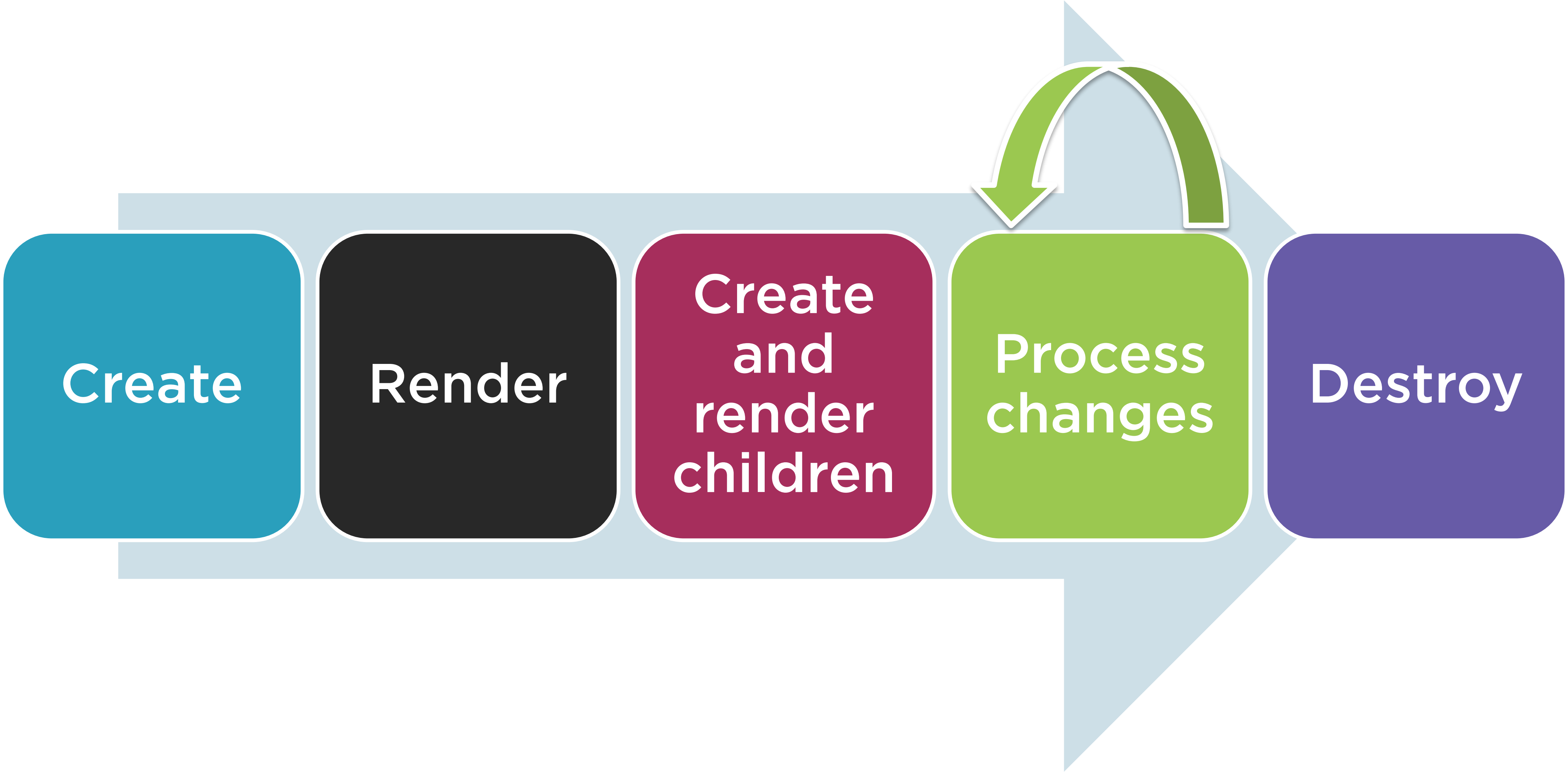
styles

```
@Component({
    selector: 'pm-products',
    templateUrl: 'app/products/product-list.component.html',
    styles: ['thead {color: #337AB7;}']})
```

styleUrls

```
@Component({
    selector: 'pm-products',
    templateUrl: 'app/products/product-list.component.html',
    styleUrls: ['app/products/product-list.component.css']})
```

Component Lifecycle



Create

Render

Create and render children

Process changes

Destroy

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

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

3



  }
```

Transforming Data with Pipes

Transform bound properties before display

Built-in pipes

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

Custom pipes

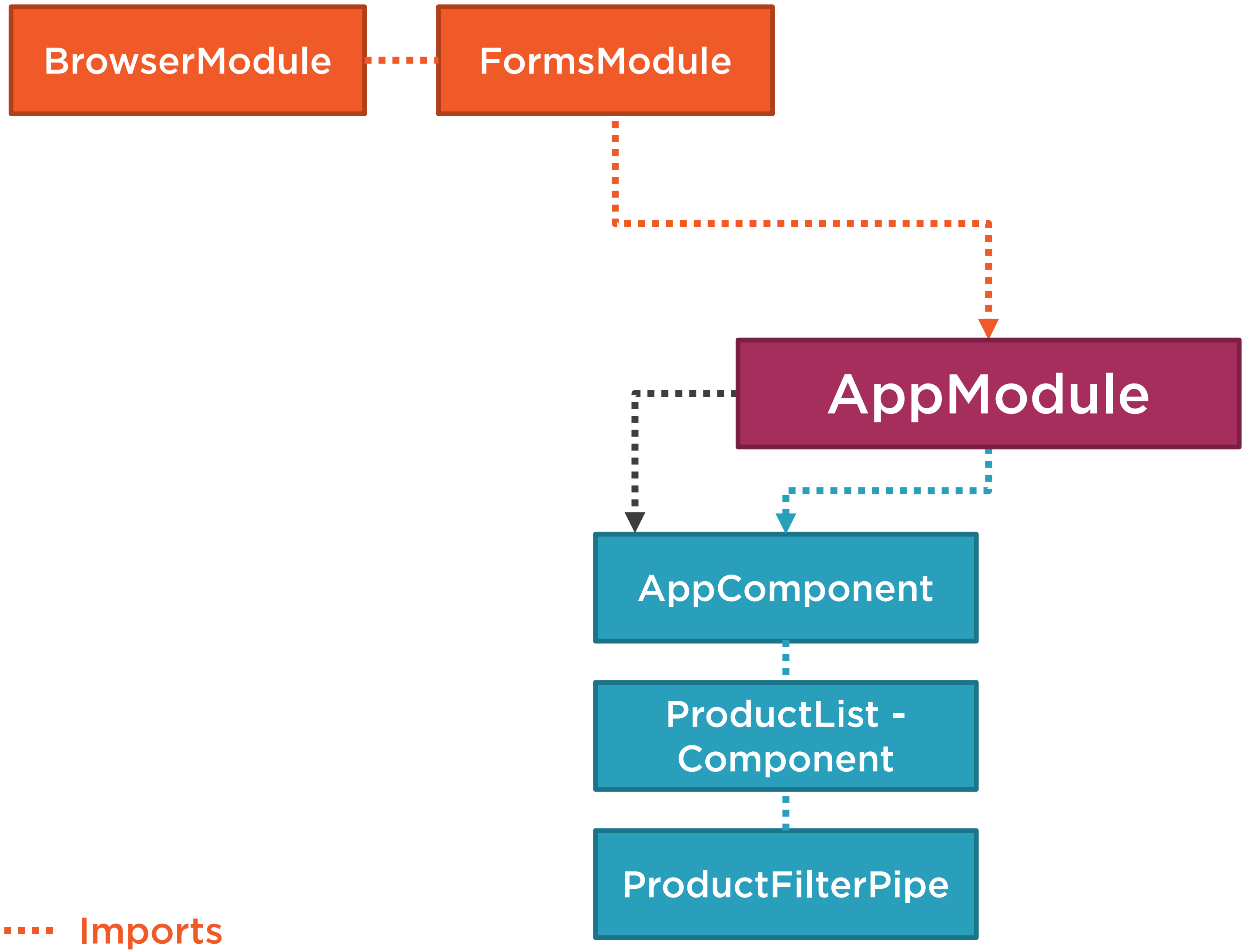
Building a Custom Pipe

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
    name:'productFilter'
})
export class ProductFilterPipe
                 implements PipeTransform {

  transform(value: IProduct[],
            filterBy: string): IProduct[]{
  }
}
```

Using a Custom Pipe

Template

```
<tr *ngFor ='let product of products | productFilter: listFilter'>
```



BrowserModule

FormsModule

AppModule

AppComponent

ProductList - Component

ProductFilterPipe

Imports

Exports

Declarations

Providers

Bootstrap

Using a Custom Pipe

Template

```
<tr *ngFor ='let product of products | productFilter: listFilter'>
```

Module

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

Relative Paths and Module Id

product-list.component.ts

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

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

Relative Paths and Module Id

product-list.component.ts

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

@Component({
    selector: 'pm-products',
    moduleId: module.id,
    templateUrl: 'product-list.component.html',
    styleUrls: ['product-list.component.css']
})
export class ProductListComponent {
 pageTitle: string = 'Product List';
 ...
}
```

`module.id`

Variable

- Available when using the CommonJS module format

Contains

- The absolute URL of the component class module file

Requires

- Writing modules in CommonJS format
- Using a module loader, such as SystemJS

Checklist: Interfaces

Defines custom types

Creating interfaces:

- `interface` keyword
- export it

Implementing interfaces:

- `implements` keyword & interface name
- Write code for each property & method

Checklist: Encapsulating Styles

`styles` **property**

- Specify an array of style strings

`styleUrls` **property**

- Specify an array of stylesheet paths

Checklist: Using Lifecycle Hooks

Import the lifecycle hook interface

Implement the lifecycle hook interface

Write code for the hook method

Checklist: Building a Custom Pipe

Import Pipe and PipeTransform

Create a class that implements PipeTransform

- `export` the class

Write code for the Transform method

Decorate the class with the Pipe decorator

Checklist: Using a Custom Pipe

Import the custom pipe

Add the pipe to the declarations array of an Angular module

Any template associated with a component that is also declared in that Angular module can use that pipe

Use the Pipe in the template

- Pipe character
- Pipe name
- Pipe arguments (separated with colons)

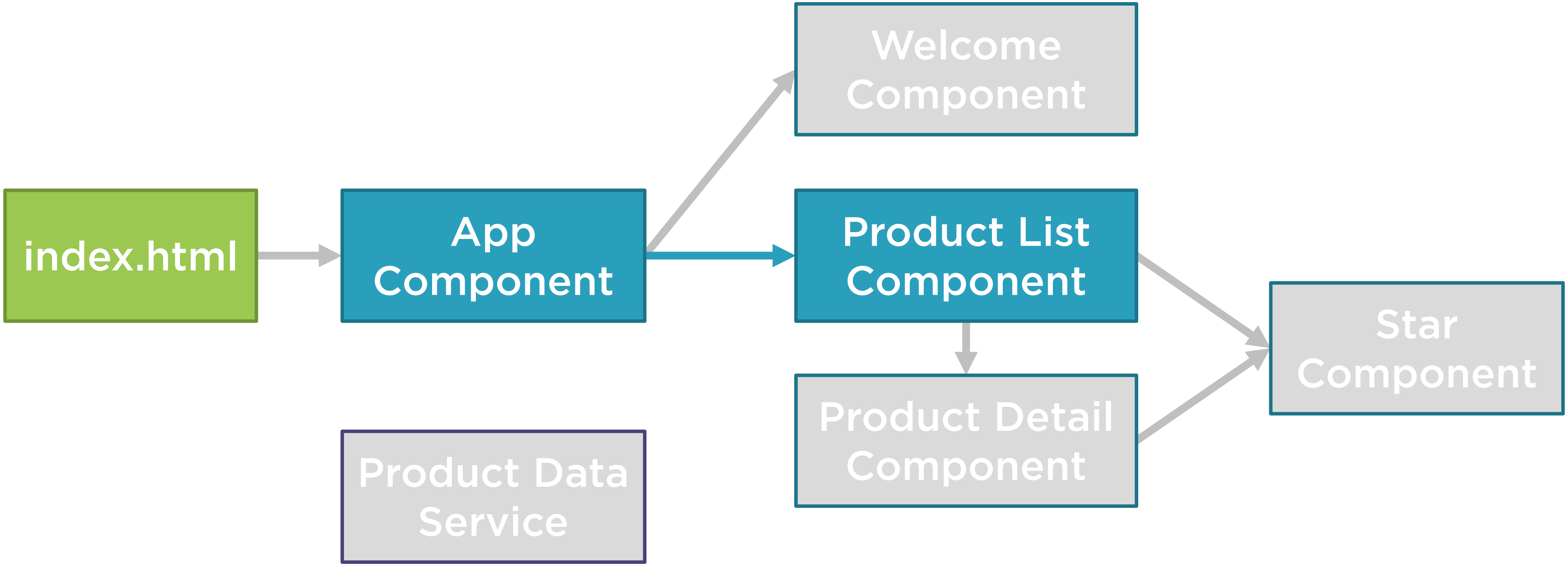
Checklist: Relative Paths with Module Id

Set the moduleId property of the component decorator to module.id

Change the Url to a component-relative path:

- templateUrl
- styleUrls

Application Architecture



index.html

App Component

Welcome Component

Product List Component

Product Detail Component

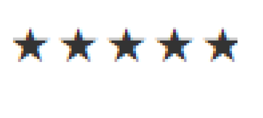
Star Component

Product Data Service

Building Nested Components

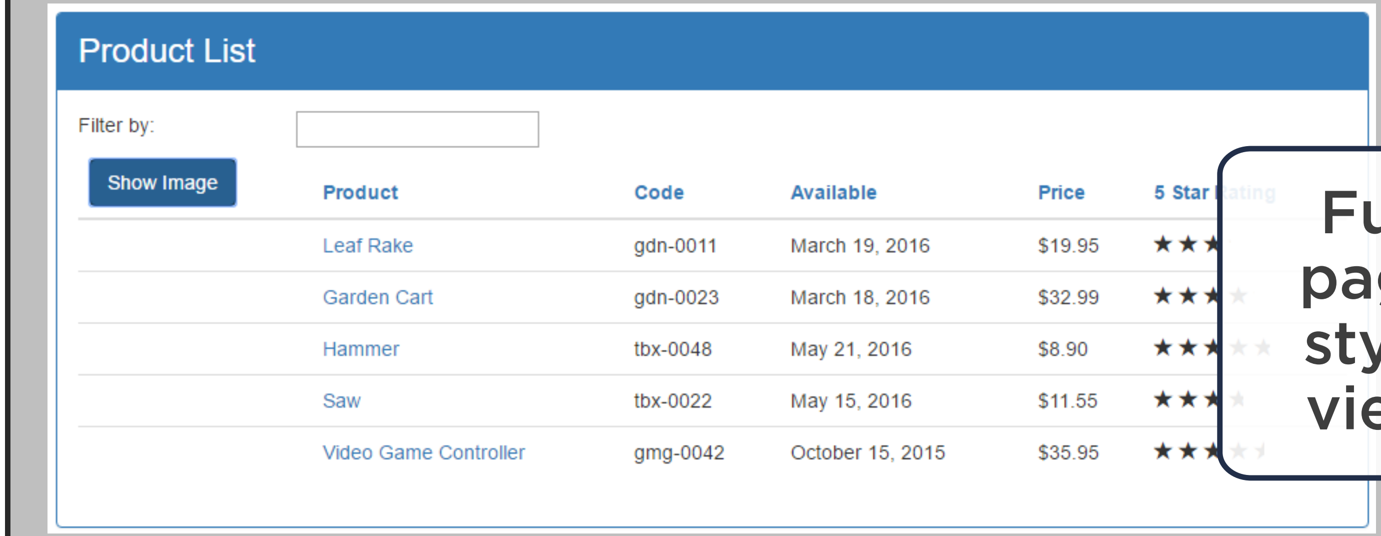
Using a Component

As a Directive



App Component OR Nested Component

As a Routing target



Product List

Filter by:

Show Image

Product	Code	Available	Price	5 Star Rating
Leaf Rake	gdn-0011	March 19, 2016	$19.95	★★★
Garden Cart	gdn-0023	March 18, 2016	$32.99	★★★
Hammer	tbx-0048	May 21, 2016	$8.90	★★★
Saw	tbx-0022	May 15, 2016	$11.55	★★★
Video Game Controller	gmg-0042	October 15, 2015	$35.95	★★★

Full page style view

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

What Makes a Component Nest-able?

Its template only manages a fragment of a larger view

It has a selector

It optionally communicates with its container

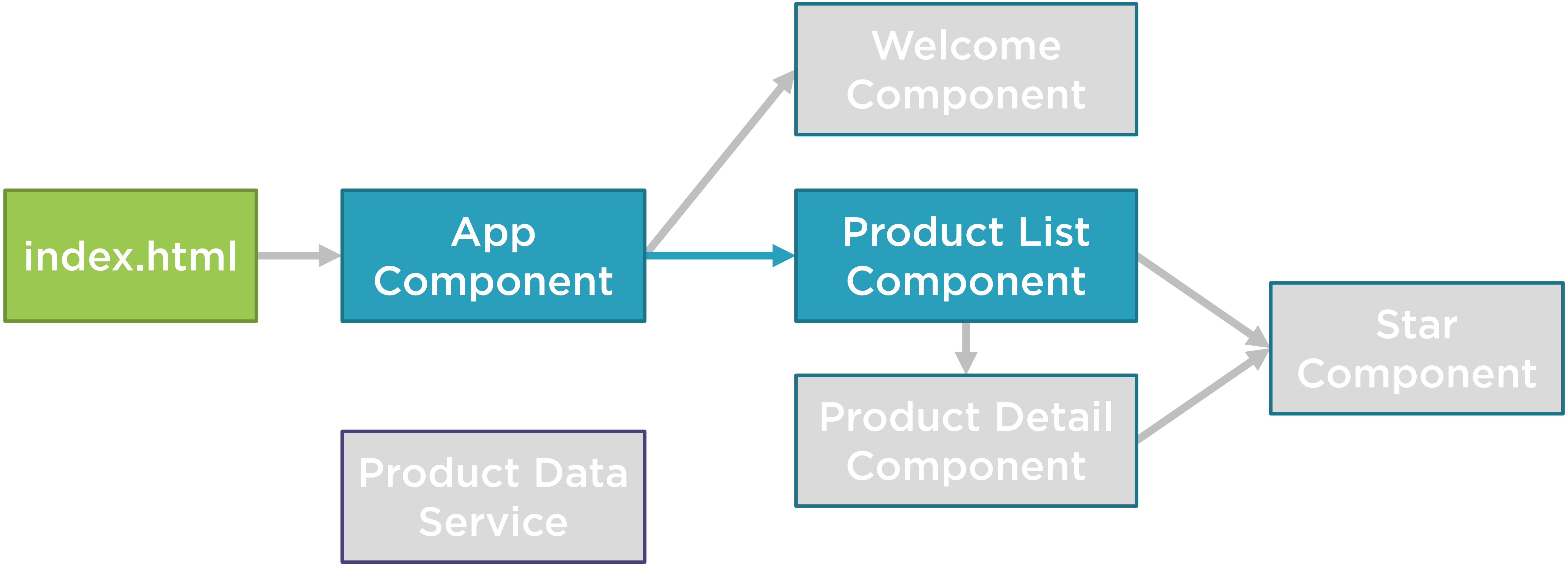
Building a Nested Component

Using a Nested Component

Passing Data to a Nested Component Using @Input

Raising an Event from a Nested Component Using @Output

Application Architecture



index.html

App Component

Welcome Component

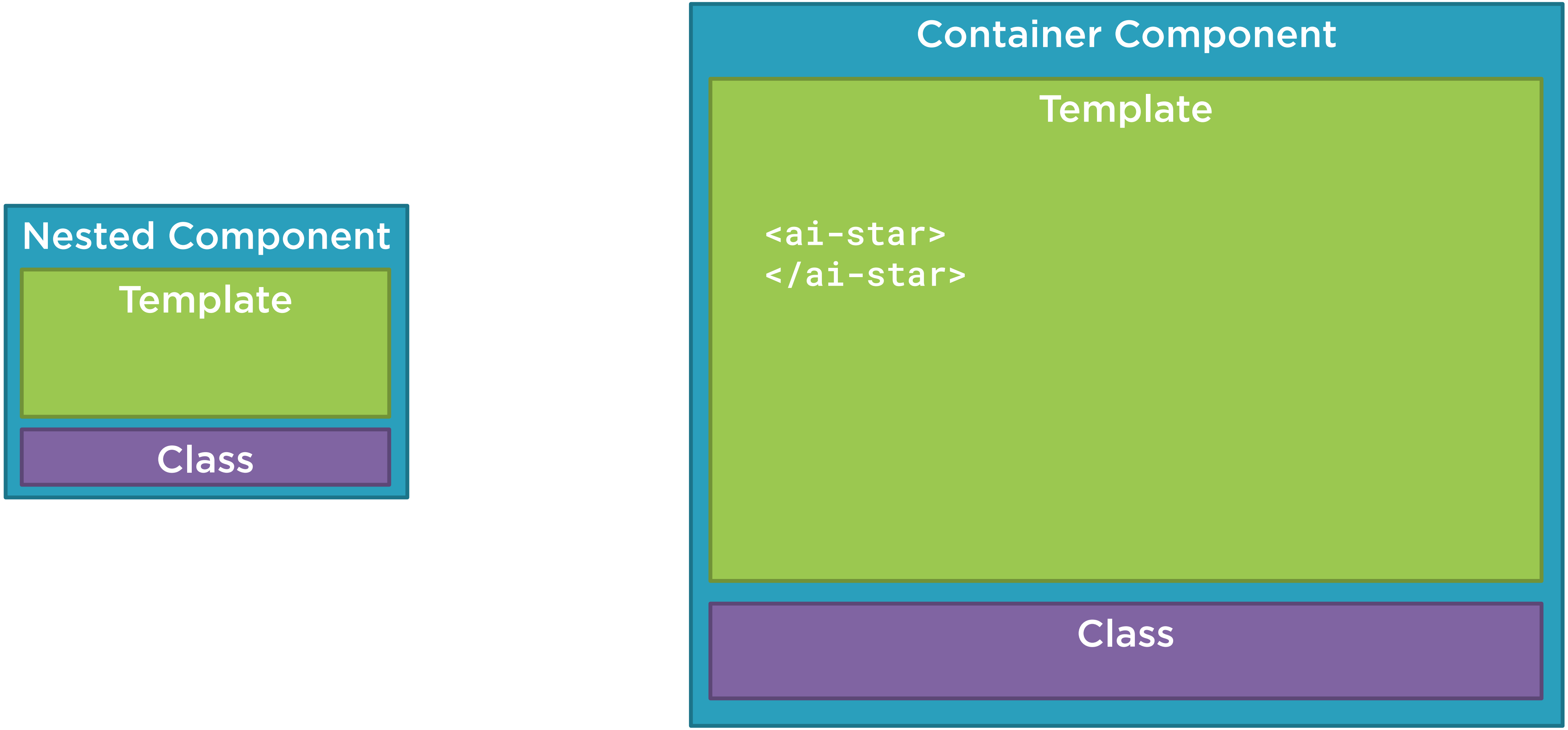
Product List Component

Product Detail Component

Star Component

Product Data Service

Building a Nested Component



Nested Component

Template

Class

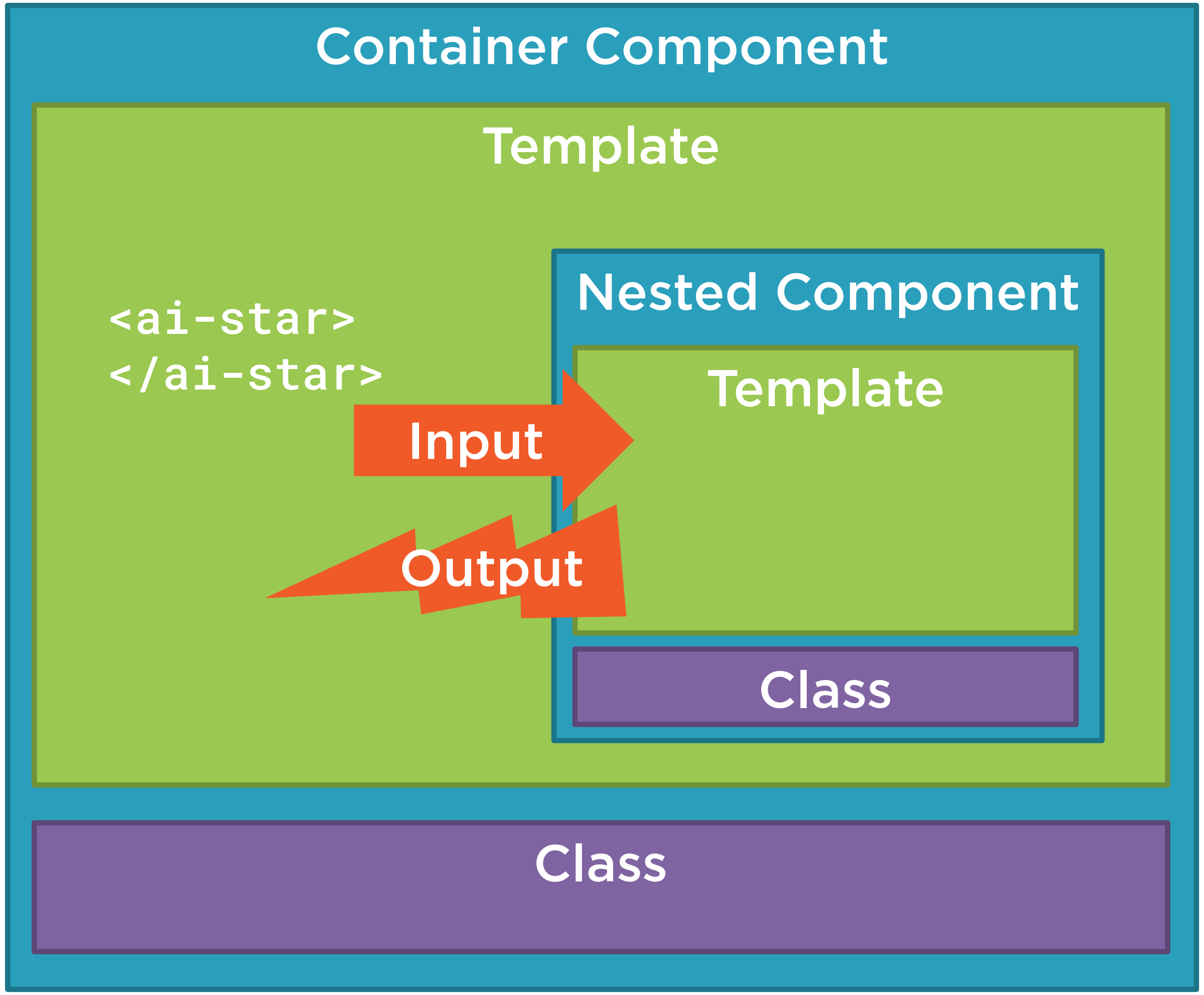
Container Component

Template

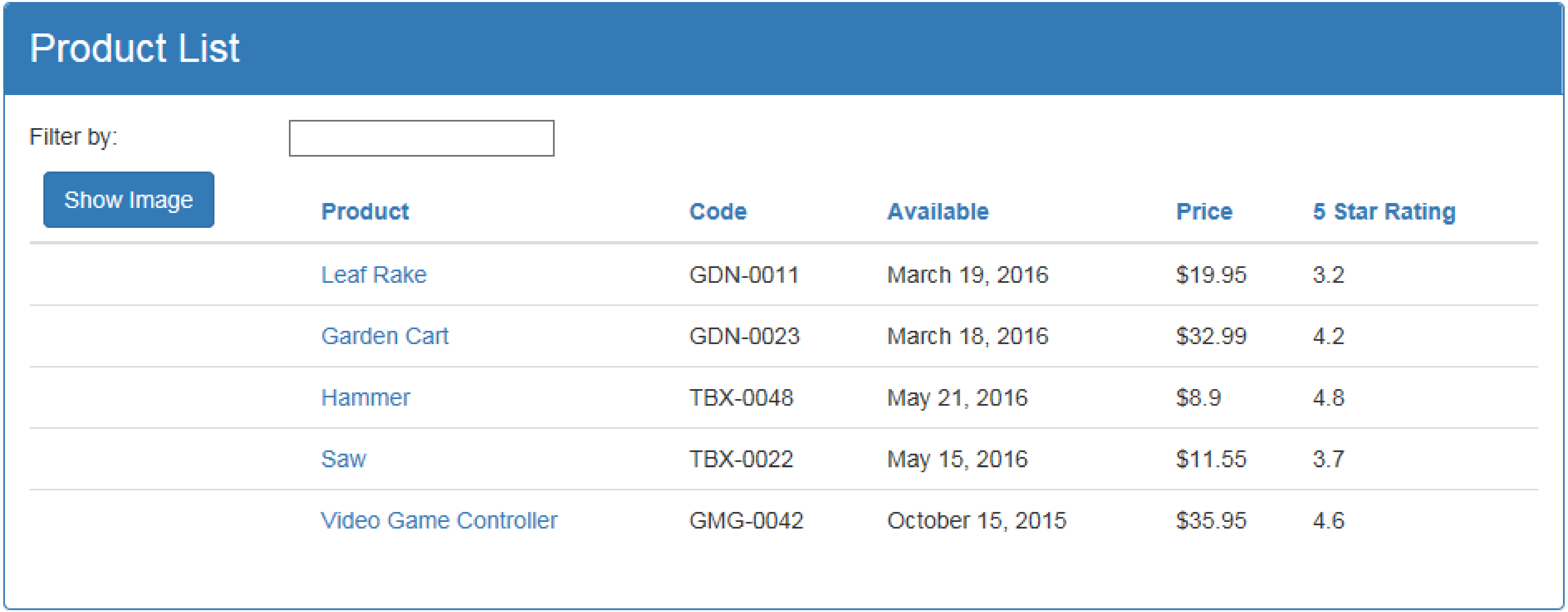
```
<ai-star>
</ai-star>
```

Class

Building a Nested Component



Product List View

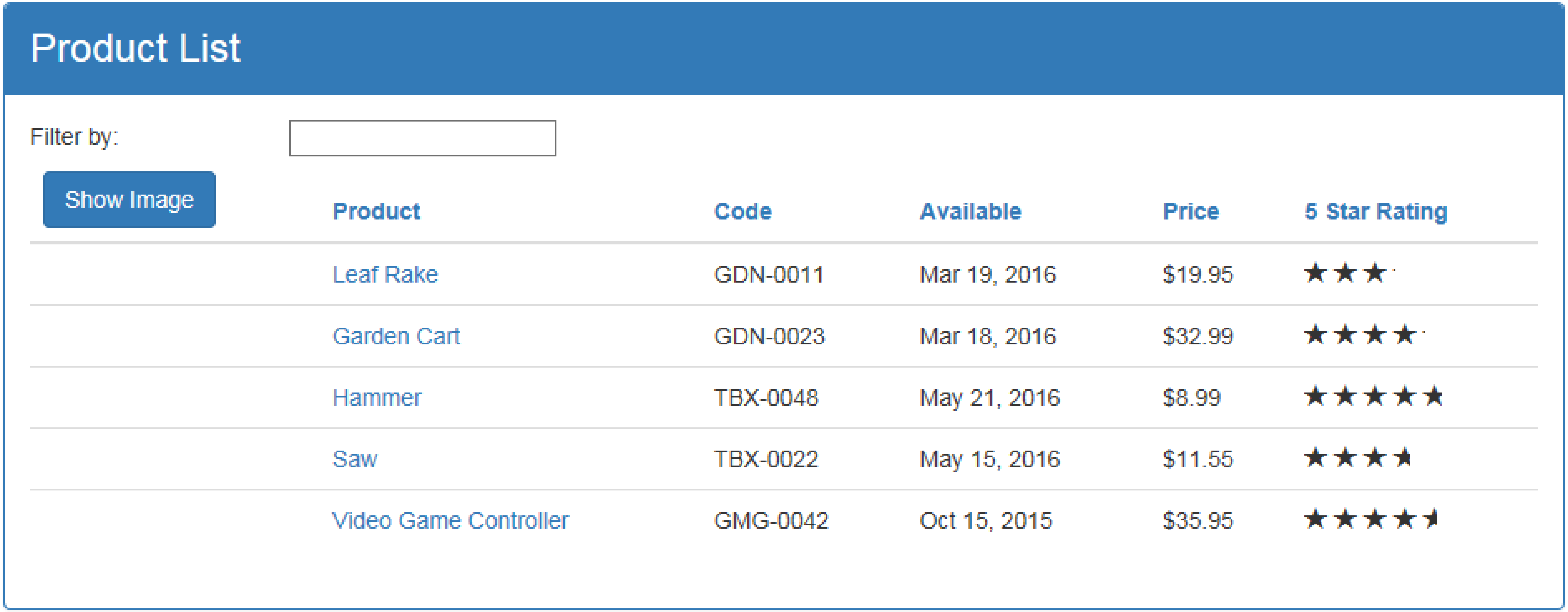


Product List

Filter by:

Product	Code	Available	Price	5 Star Rating
Leaf Rake	GDN-0011	March 19, 2016	$19.95	3.2
Garden Cart	GDN-0023	March 18, 2016	$32.99	4.2
Hammer	TBX-0048	May 21, 2016	$8.9	4.8
Saw	TBX-0022	May 15, 2016	$11.55	3.7
Video Game Controller	GMG-0042	October 15, 2015	$35.95	4.6

Product List View



Product List

Filter by:

Product	Code	Available	Price	5 Star Rating
Leaf Rake	GDN-0011	Mar 19, 2016	$19.95	★★★
Garden Cart	GDN-0023	Mar 18, 2016	$32.99	★★★★
Hammer	TBX-0048	May 21, 2016	$8.99	★★★★★
Saw	TBX-0022	May 15, 2016	$11.55	★★★★
Video Game Controller	GMG-0042	Oct 15, 2015	$35.95	★★★★★

Using a Nested Component as a Directive

product-list.component.ts

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

product-list.component.html

```
<td>
   {{ product.starRating | number }}
</td>
```

star.component.ts

```
@Component({
  selector: 'ai-star',
  templateURL: 'star.component.html'
})
export class StarComponent {
  rating: number;
  starWidth: number;
}
```

Using a Nested Component as a Directive

product-list.component.ts

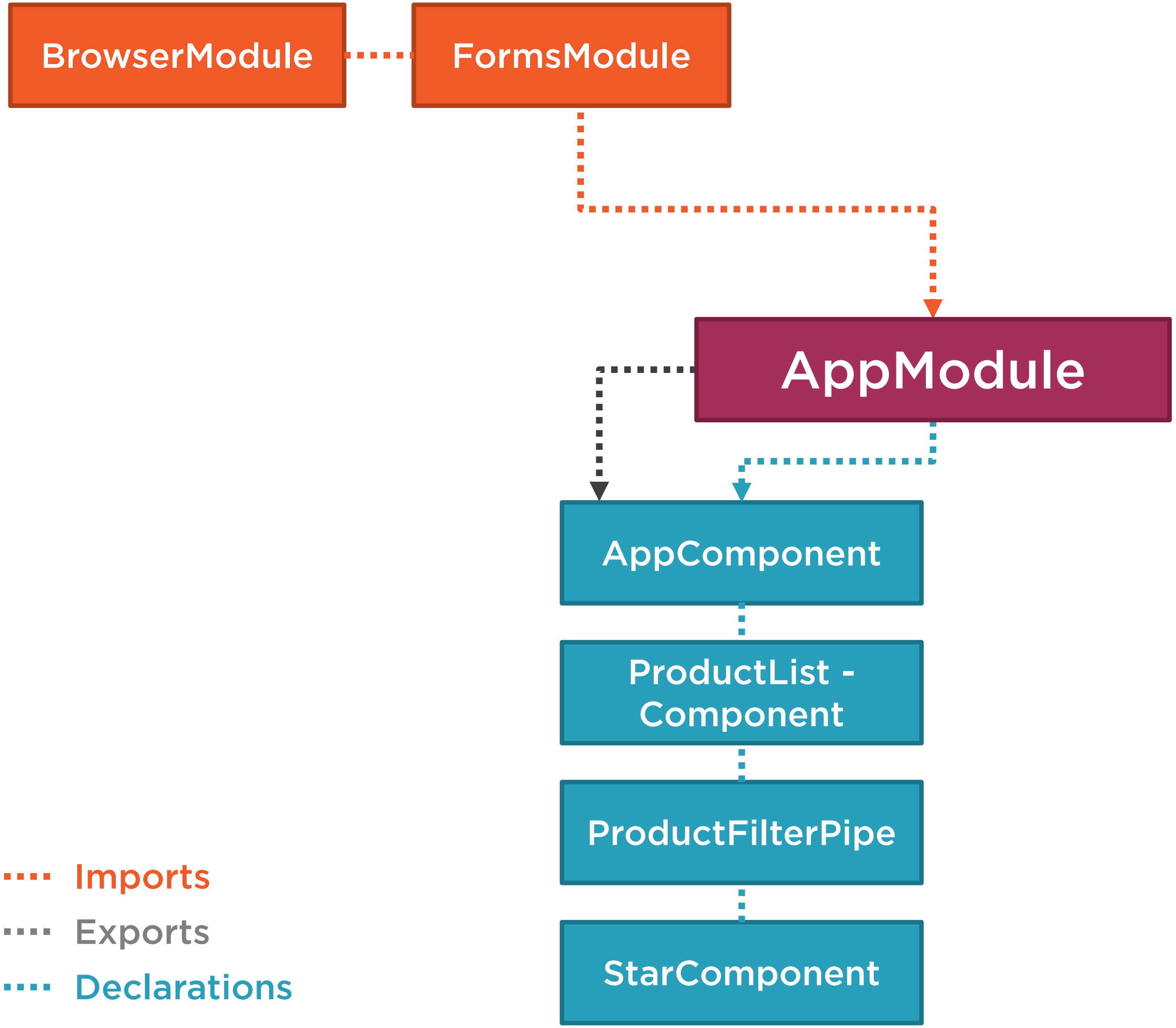
```
@Component({
  selector: 'pm-products',
  templateURL: 'product-list.component.html'
})
export class ProductListComponent { }
```

product-list.component.html

```
<td>
   <ai-star></ai-star>
</td>
```

star.component.ts

```
@Component({
  selector: 'ai-star',
  templateURL: 'star.component.html'
})
export class StarComponent {
  rating: number;
  starWidth: number;
}
```



BrowserModule

FormsModule

AppModule

AppComponent

ProductList - Component

ProductFilterPipe

StarComponent

·········· Imports

·········· Exports

·········· Declarations

·········· Providers

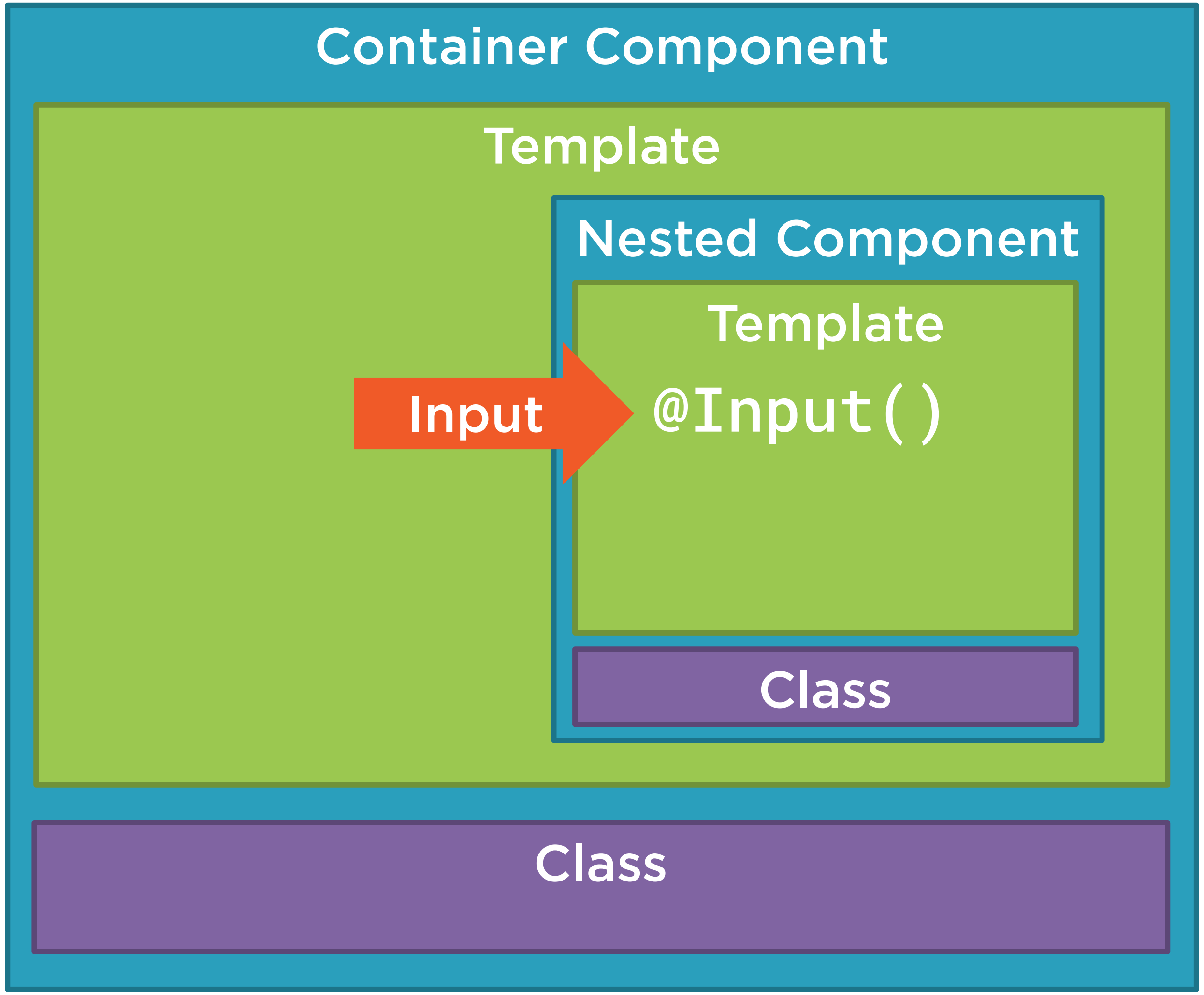
·········· Bootstrap

Telling Angular About Our Component

app.module.ts

```
...
import { StarComponent } from './shared/star.component';

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

Passing Data to a Nested Component (@Input)



Passing Data to a Nested Component (@Input)

product-list.component.ts

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

product-list.component.html

```
<td>
   <ai-star></ai-star>
</td>
```

star.component.ts

```
@Component({
  selector: 'ai-star',
  templateURL: 'star.component.html'
})
export class StarComponent {
  @Input() rating: number;
  starWidth: number;
}
```

Passing Data to a Nested Component (@Input)

product-list.component.ts

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

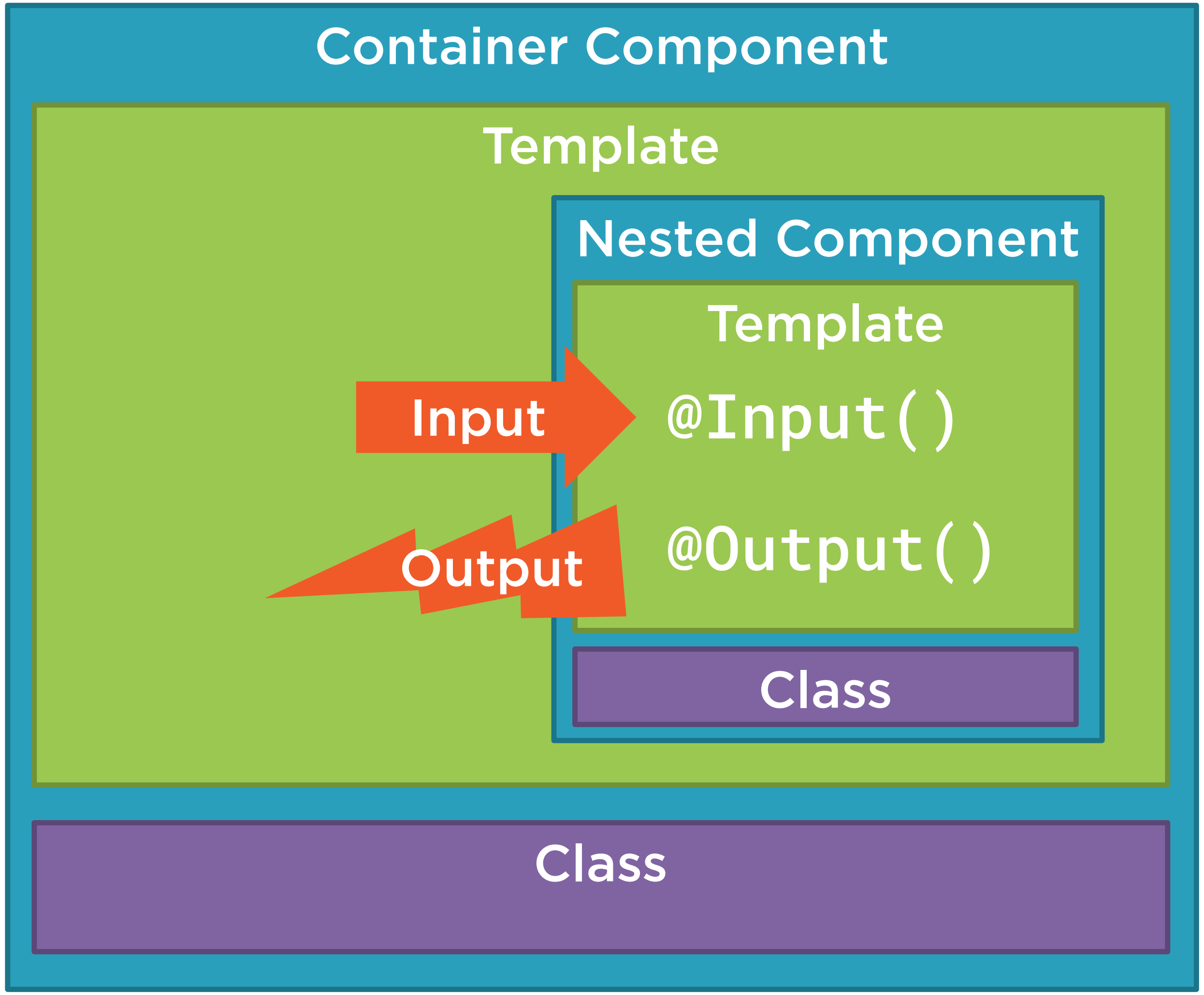
product-list.component.html

```
<td>
  <ai-star [rating]='product.starRating'>
  </ai-star>
</td>
```

star.component.ts

```
@Component({
  selector: 'ai-star',
  templateURL: 'star.component.html'
})
export class StarComponent {
  @Input() rating: number;
  starWidth: number;
}
```

Raising an Event (@Output)



Raising an Event (@Output)

product-list.component.ts

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

star.component.ts

```
@Component({
  selector: 'ai-star',
  templateURL: 'star.component.html'
})
export class StarComponent {
 @Input() rating: number;
 starWidth: number;
 @Output() notify: EventEmitter<string> =
              new EventEmitter<string>();
}
```

product-list.component.html

```
<td>
  <ai-star [rating]='product.starRating'>
  </ai-star>
</td>
```

Raising an Event (@Output)

product-list.component.ts

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

product-list.component.html

```
<td>
  <ai-star [rating]='product.starRating'>
  </ai-star>
</td>
```

star.component.ts

```
@Component({
  selector: 'ai-star',
  templateURL: 'star.component.html'
})
export class StarComponent {
 @Input() rating: number;
 starWidth: number;
 @Output() notify: EventEmitter<string> =
             new EventEmitter<string>();

 onClick() {
   this.notify.emit('clicked!');
 }
}
```

star.component.html

```
<div (click)='onClick()'>
  ... stars ...
</div>
```

Raising an Event (@Output)

product-list.component.ts

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

product-list.component.html

```
<td>
  <ai-star [rating]='product.starRating'
           (notify)='onNotify($event)'>
  </ai-star>
</td>
```

star.component.ts

```
@Component({
  selector: 'ai-star',
  templateURL: 'star.component.html'
})
export class StarComponent {
 @Input() rating: number;
 starWidth: number;
 @Output() notify: EventEmitter<string> =
             new EventEmitter<string>();

 onClick() {
   this.notify.emit('clicked!');
 }
}
```

star.component.html

```
<div (click)='onClick()'>
  ... stars ...
</div>
```

Raising an Event (@Output)

product-list.component.ts

```
@Component({
  selector: 'pm-products',
  templateURL: 'product-list.component.html'
})
export class ProductListComponent {
  onNotify(message: string): void { }
}
```

product-list.component.html

```
<td>
  <ai-star [rating]='product.starRating'
           (notify)='onNotify($event)'>
  </ai-star>
</td>
```

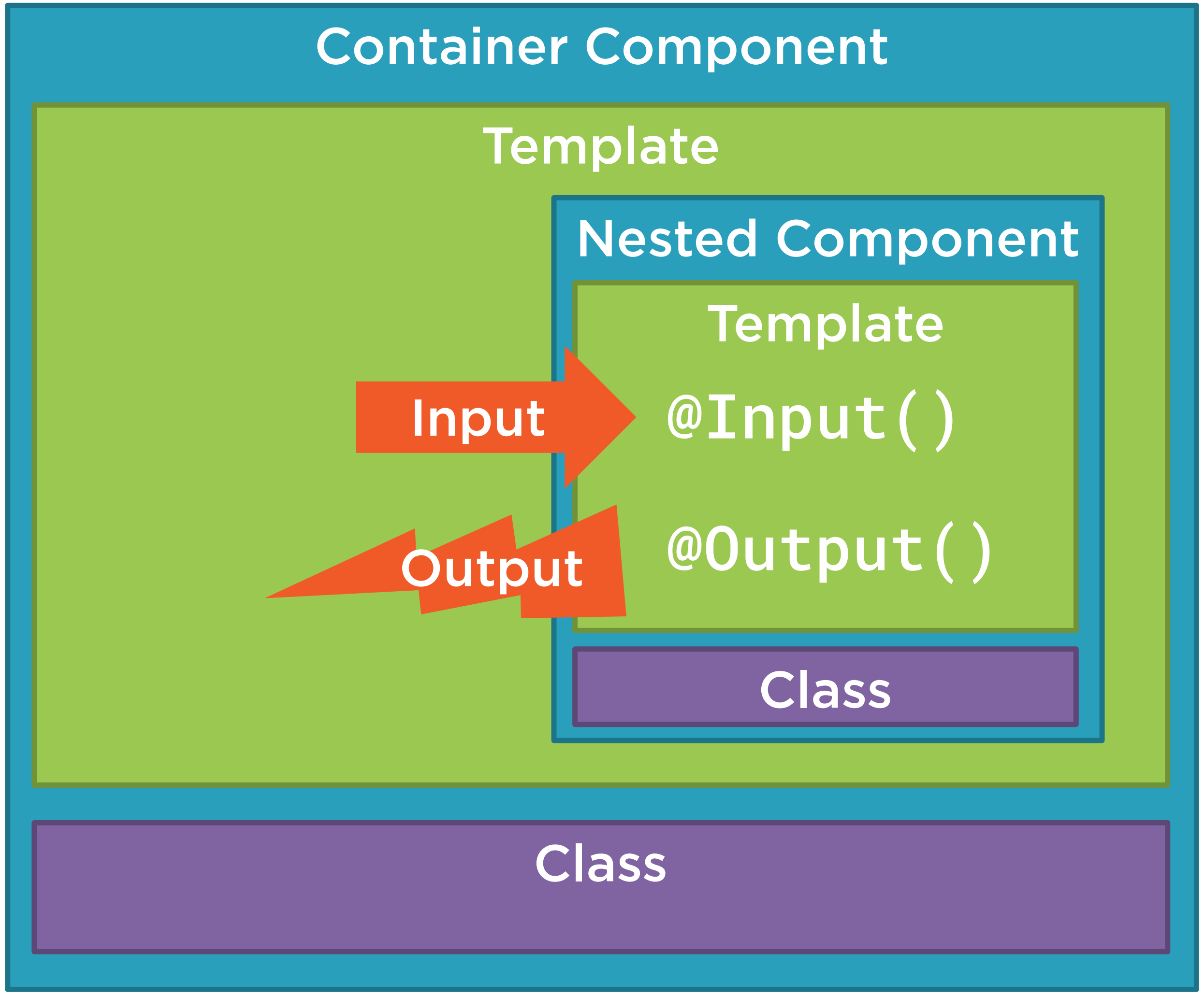
star.component.ts

```
@Component({
  selector: 'ai-star',
  templateURL: 'star.component.html'
})
export class StarComponent {
 @Input() rating: number;
 starWidth: number;
 @Output() notify: EventEmitter<string> =
             new EventEmitter<string>();

 onClick() {
   this.notify.emit('clicked!');
 }
}
```

star.component.html

```
<div (click)='onClick()'>
  ... stars ...
</div>
```

Nest-able Component's Public API



Checklist: Nested Component

Input decorator

- Attached to a property of any type
- Prefix with @; Suffix with ()

Output decorator

- Attached to a property declared as an EventEmitter
- Use the generic argument to define the event payload type
- Use the new keyword to create an instance of the EventEmitter
- Prefix with @; Suffix with ()

Checklist: Container Component

Use the directive

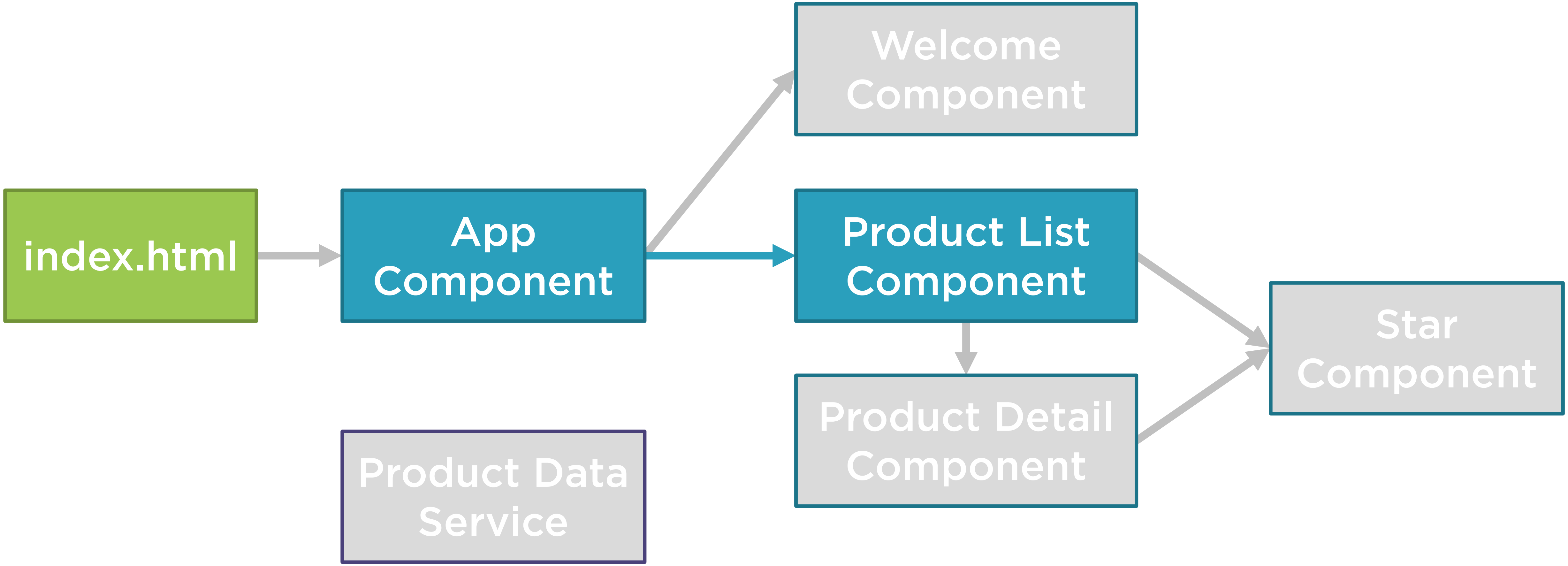
- Directive name -> nested component's selector

Use property binding to pass data to the nested component

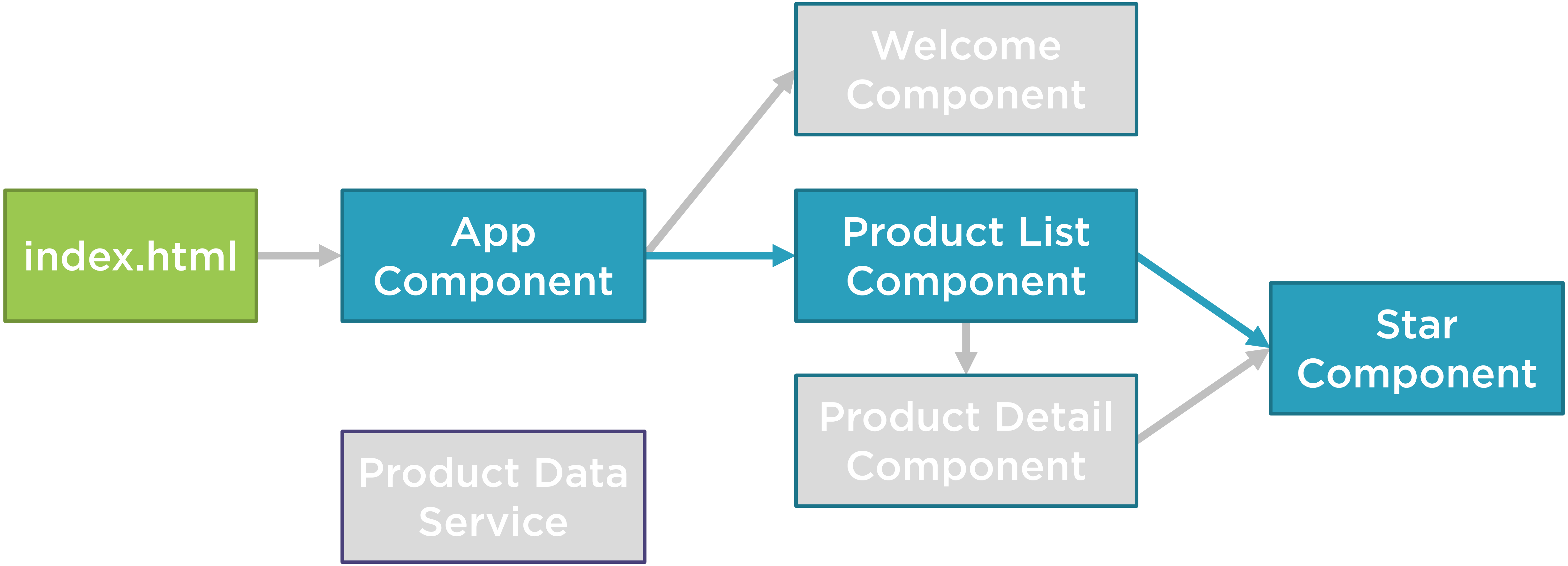
Use event binding to respond to events from the nested component

- Use $event to access the event payload passed from the nested component

Application Architecture



Application Architecture



Services and Dependency Injection

Products

Logging

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

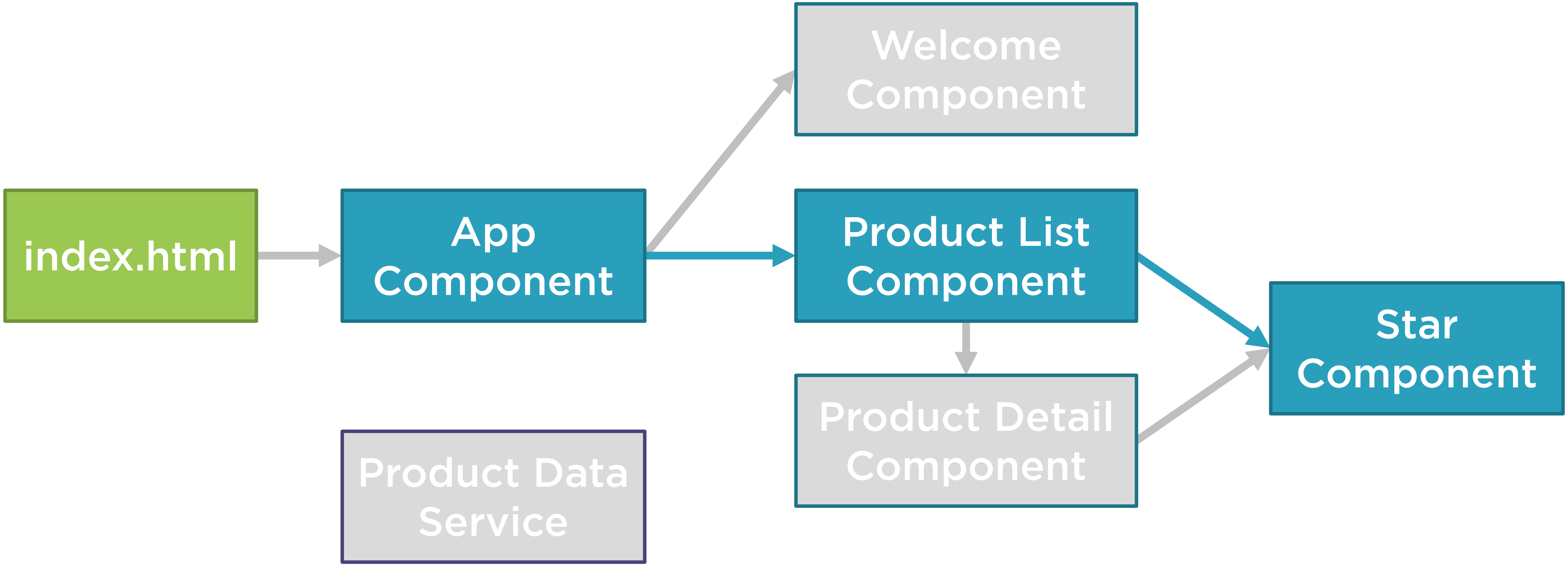
How Does It Work?

Building a Service

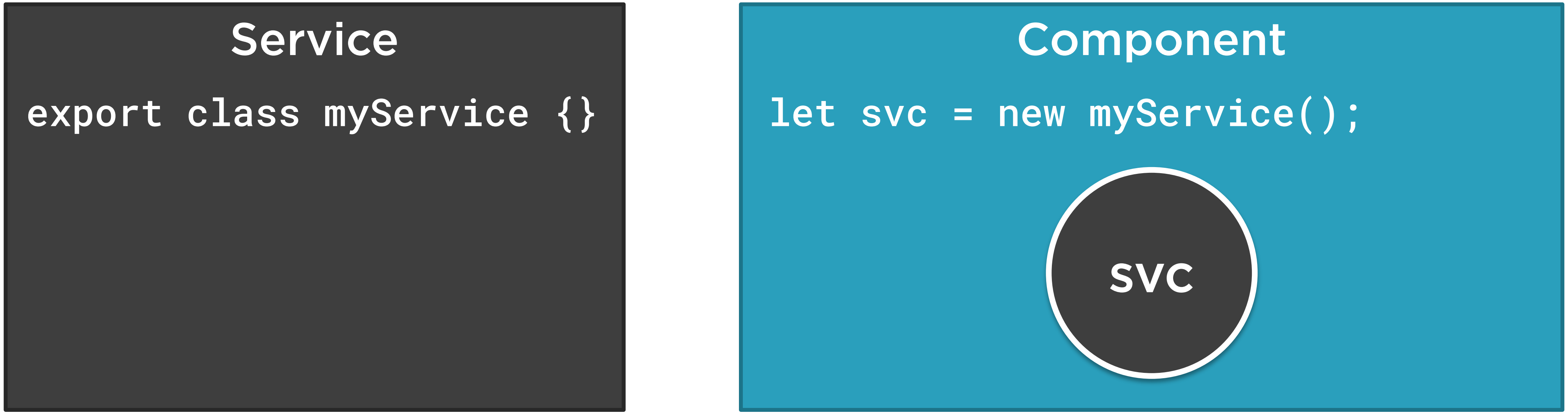
Registering the Service

Injecting the Service

Application Architecture



How Does It Work?



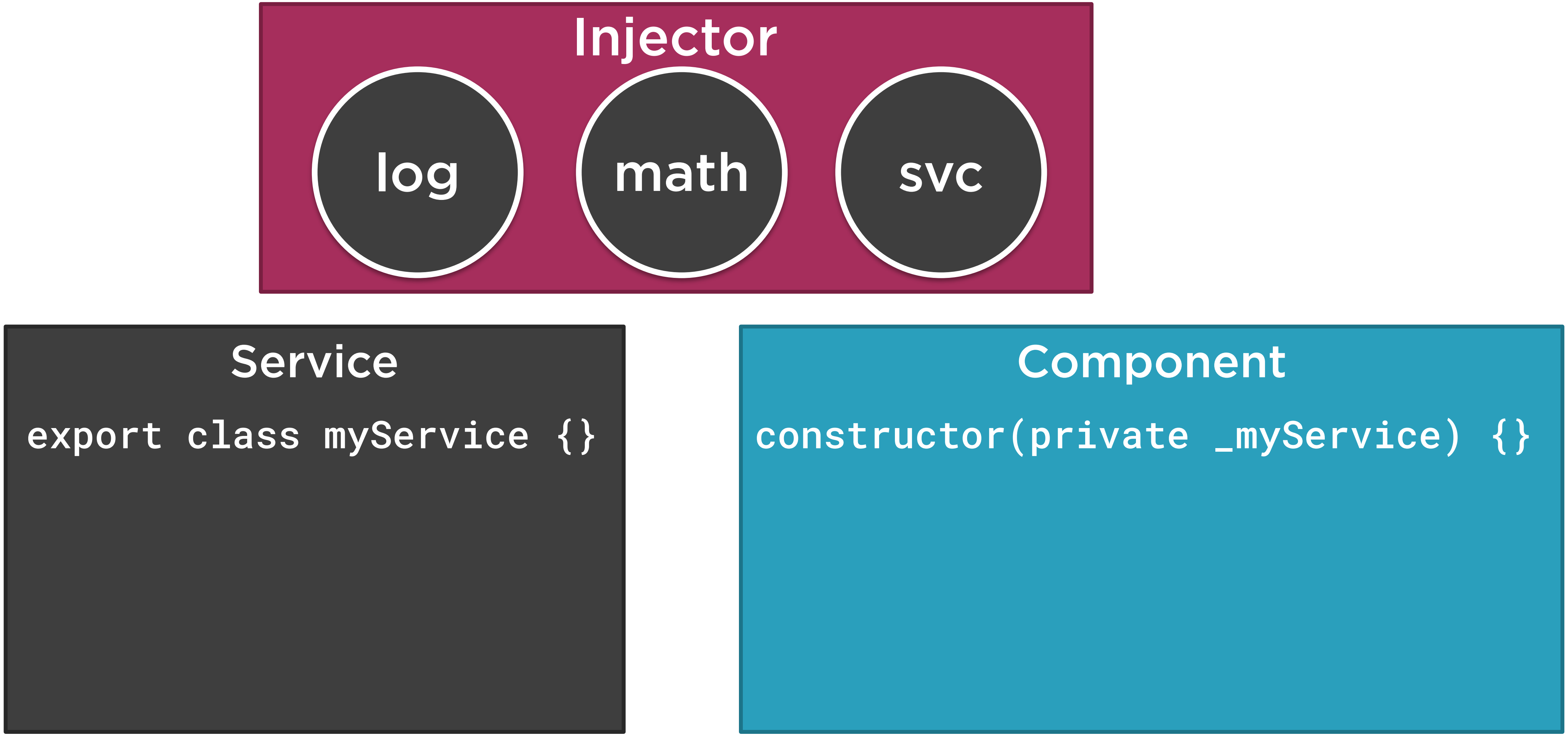
Service

```
export class myService {}
```

Component

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

How Does It Work?



Injector

log

math

svc

Service

```
export class myService {}
```

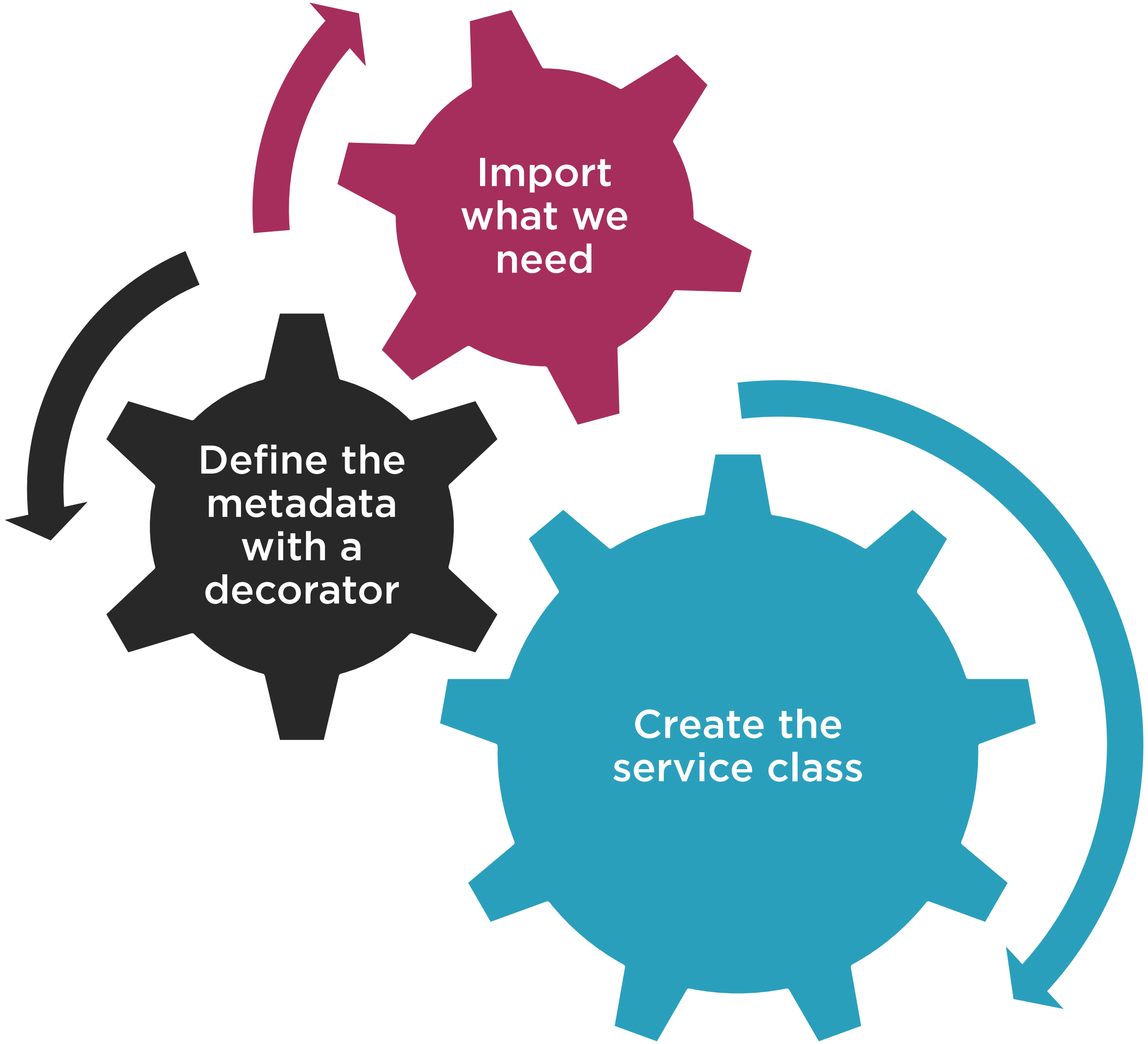
Component

```
constructor(private _myService) {}
```

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.

Building a Service

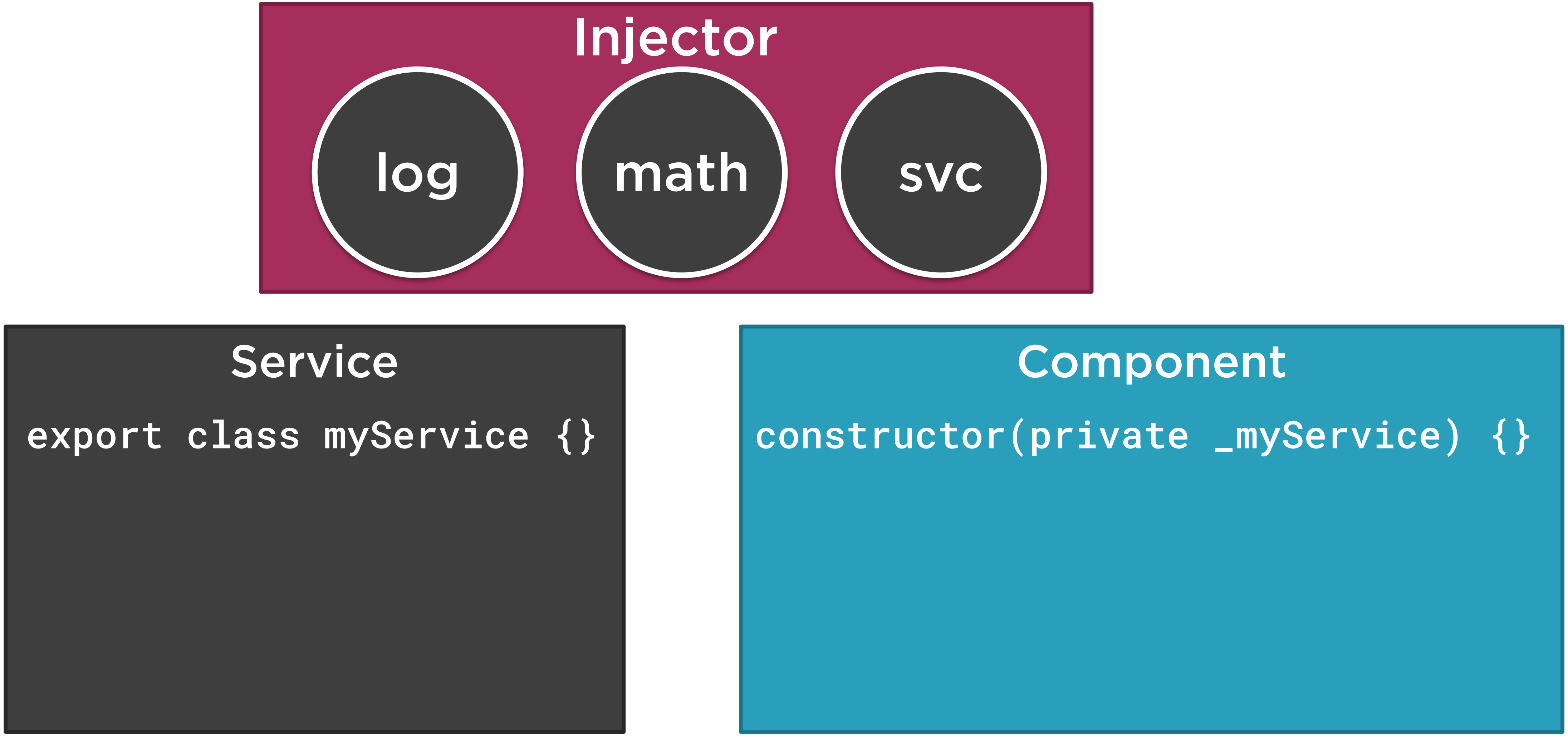


Building a Service

product.service.ts

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

@Injectable()
export class ProductService {

  getProducts(): IProduct[] {
  }

}
```

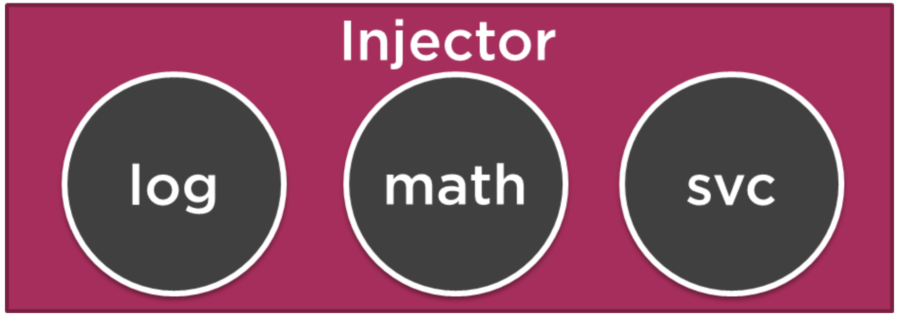
Registering the Service



```
export class myService {}
```

```
constructor(private _myService) {}
```

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

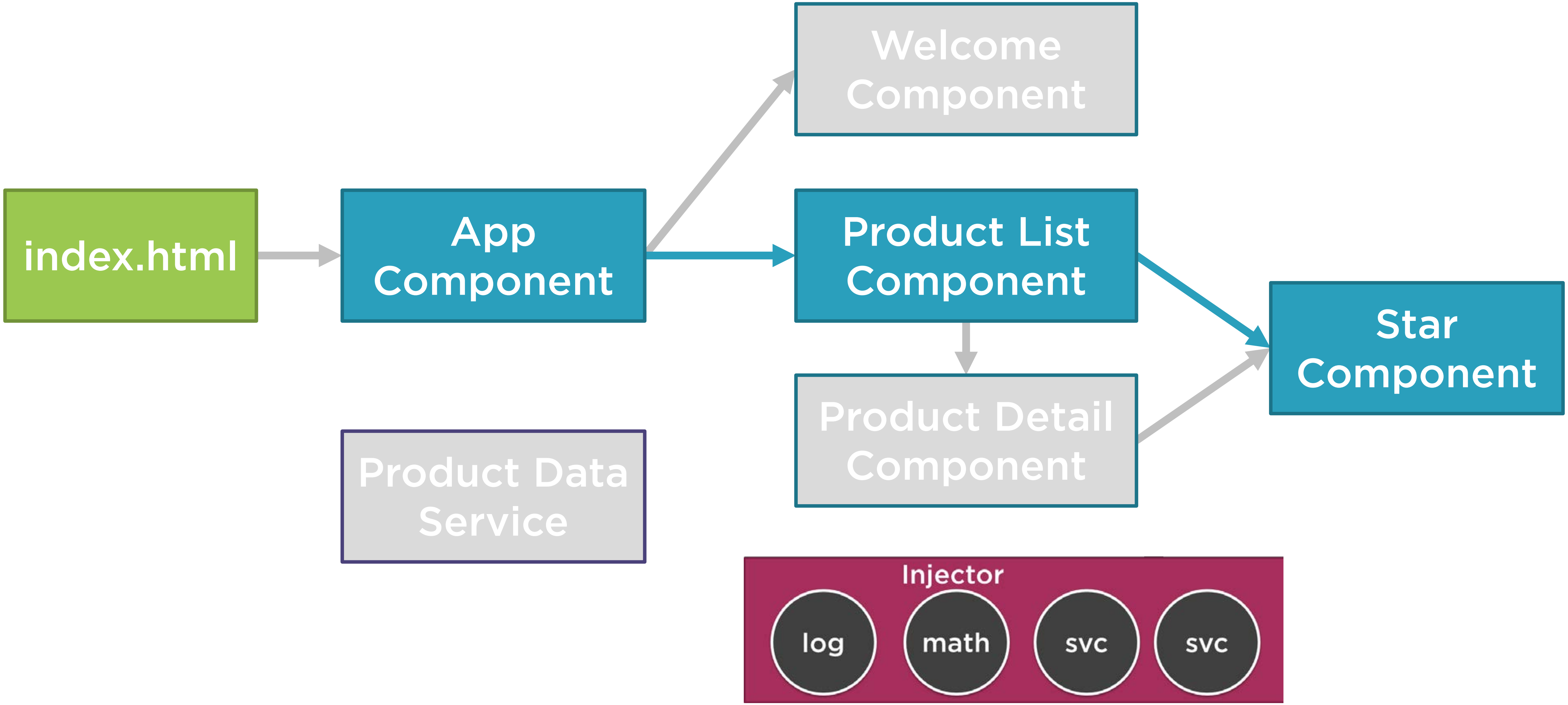
Registered in component:

- Injectable to component AND its children

Registered in Angular module:

- Injectable everywhere in the application

Application Architecture

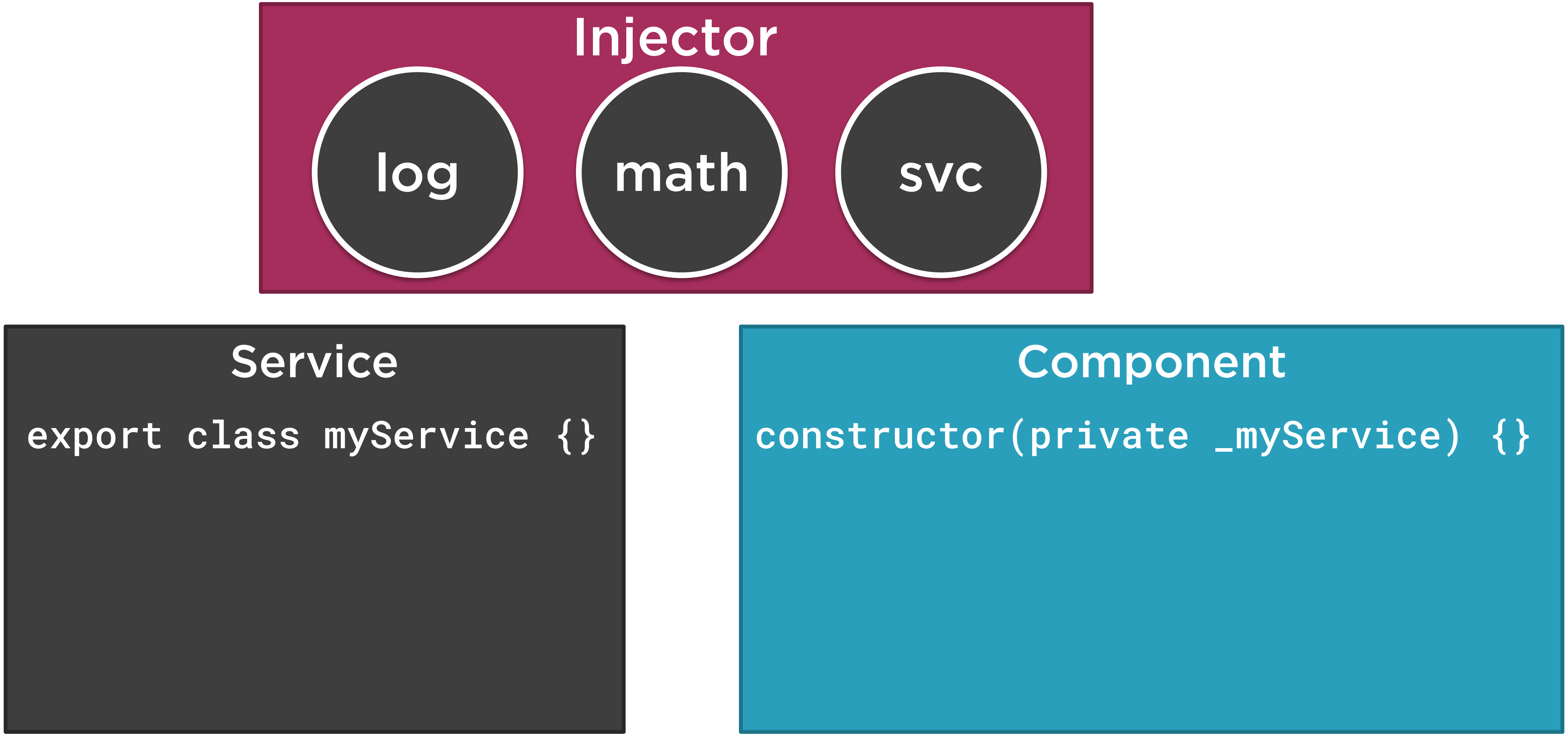


Registering a Provider

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



```
export class myService {}
```

```
constructor(private _myService) {}
```

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) {
 }

}
```

Checklist: Creating a Service

Service class

- Clear name
- Use PascalCasing
- Append "Service" to the name
- `export` keyword

Service decorator

- Use Injectable
- Prefix with @; Suffix with ()

Import what we need

Checklist: Registering a Service in a Component

Select the appropriate level in the hierarchy

- Root component if service is used throughout the application
- Specific component if only that component uses the service
- Otherwise, common ancestor

Component metadata

- Set the providers property
- Pass in an array

Import what we need

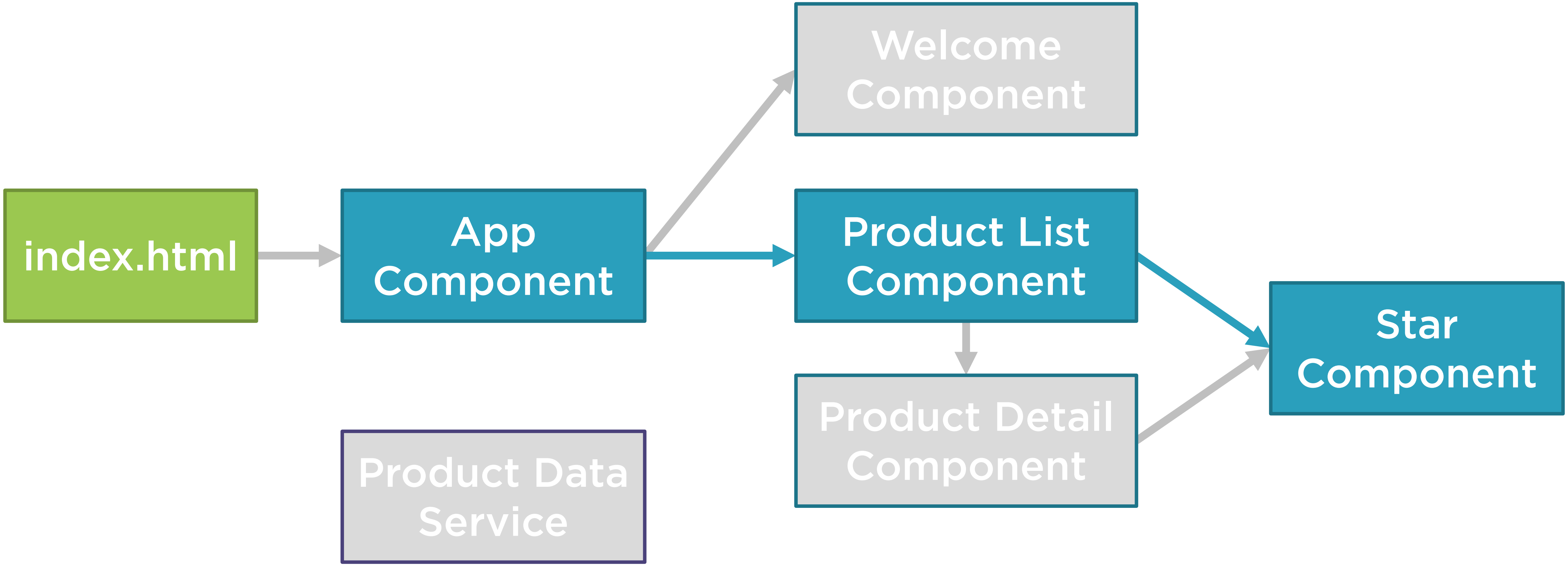
Checklist: Dependency Injection

Specify the service as a dependency

Use a constructor parameter

Service is injected when component is instantiated

Application Architecture



index.html

App Component

Welcome Component

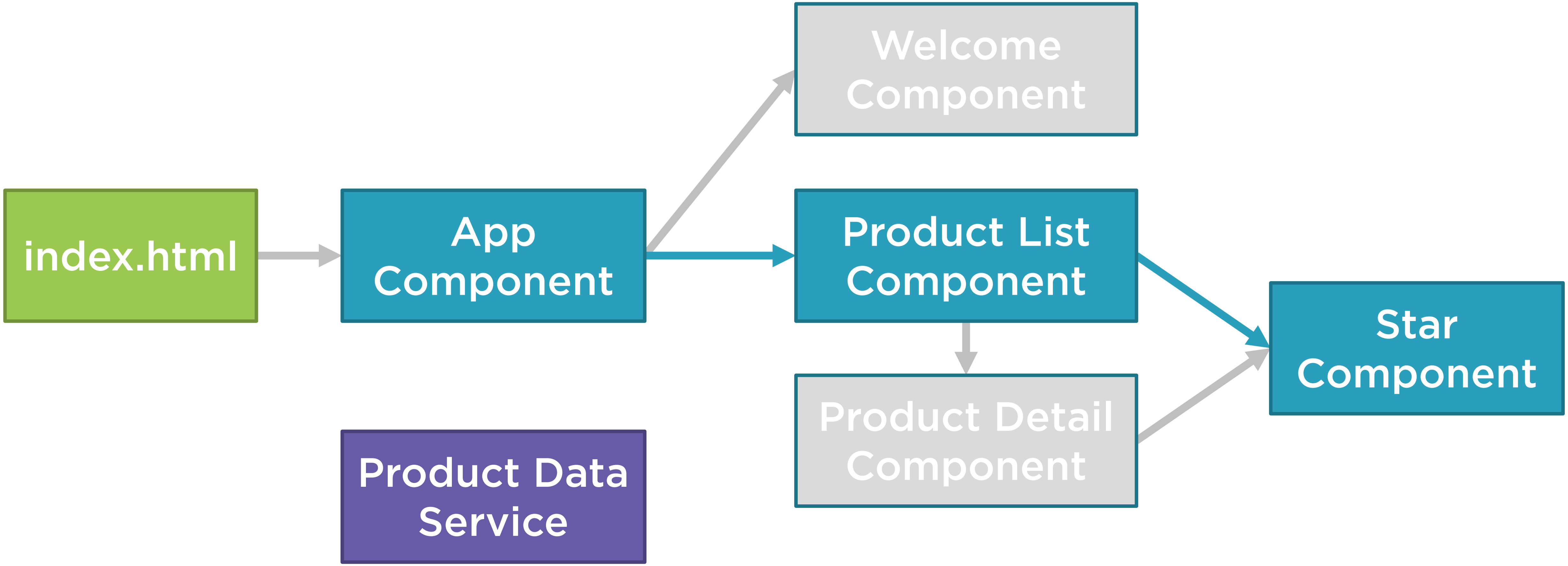
Product List Component

Product Detail Component

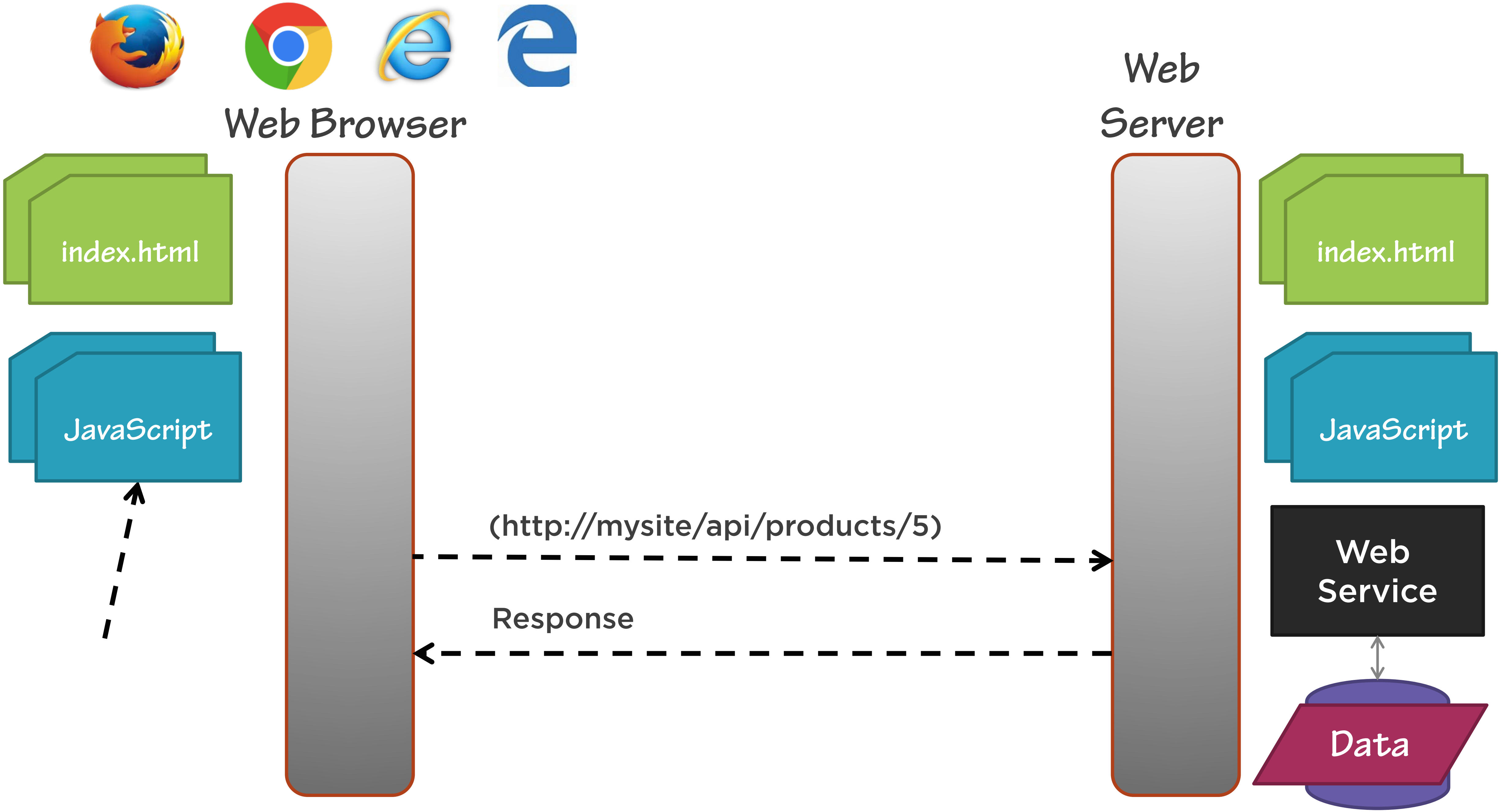
Star Component

Product Data Service

Application Architecture



Retrieving Data Using HTTP


Web Browser
Web
Server
index.html
JavaScript
index.html
JavaScript
(http://mysite/api/products/5)
Response
Web
Service
Data

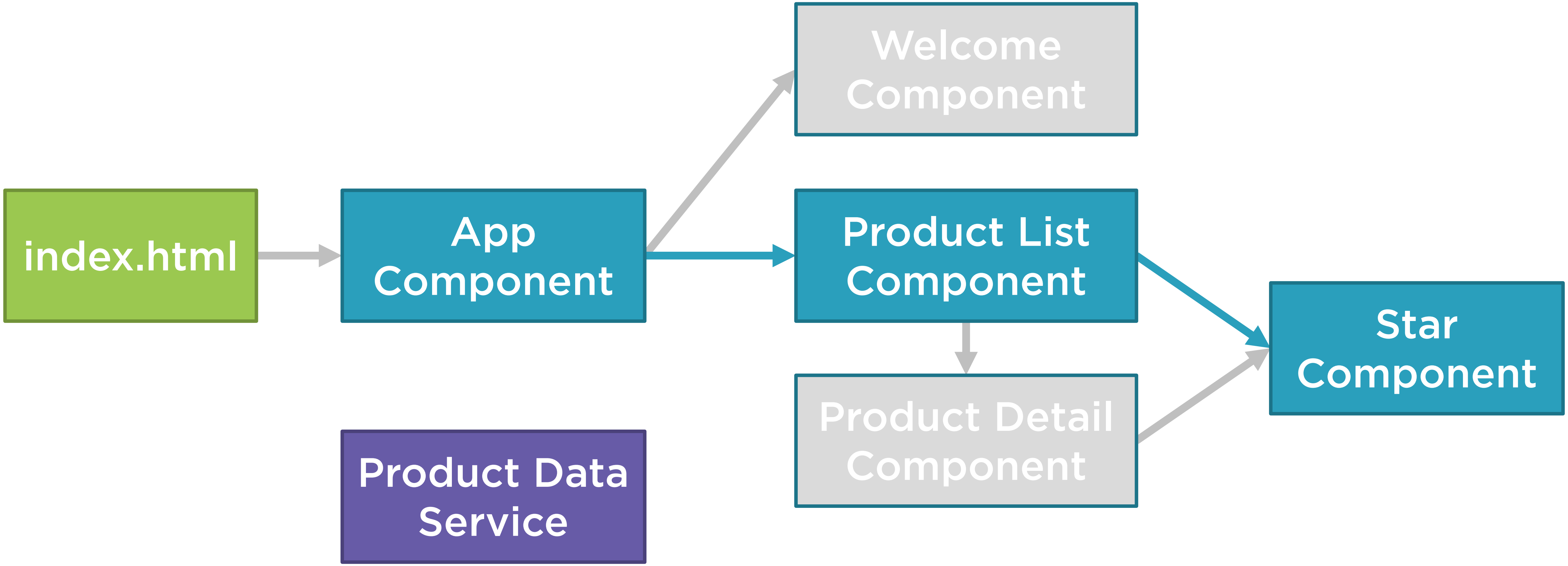
Observables and Reactive Extensions

Sending an Http Request

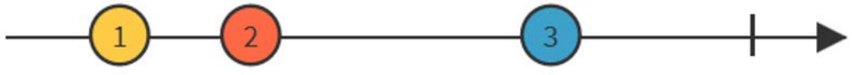
Exception Handling

Subscribing to an Observable

Application Architecture



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

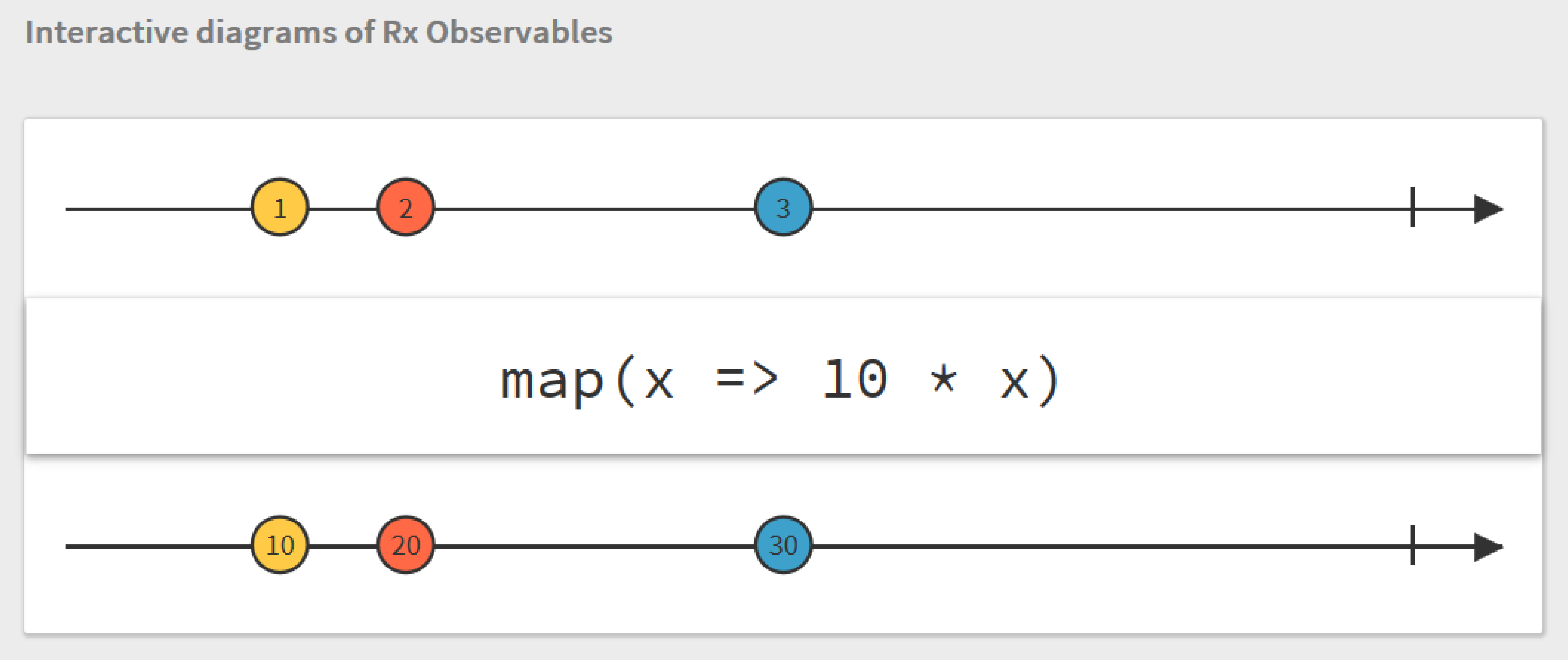
Methods on observables that compose new observables

Transform the source observable in some way

Process each value as it is emitted

Examples: map, filter, take, merge, …

Observables



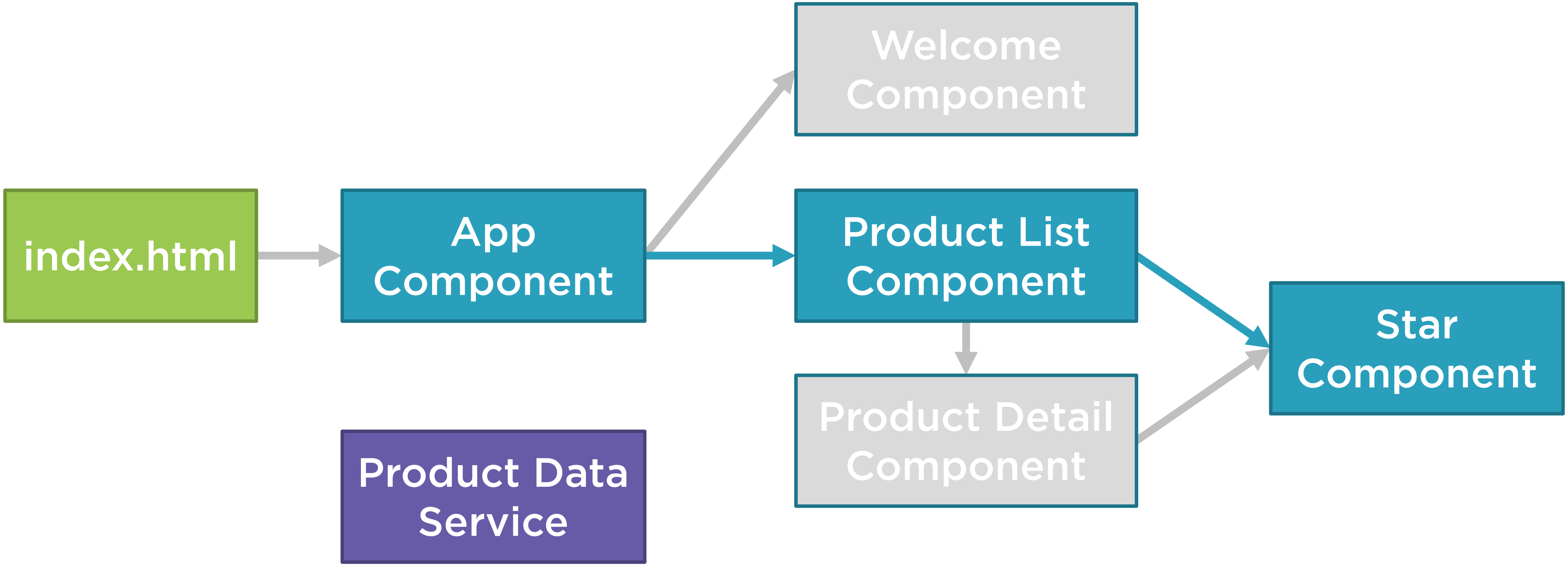
Interactive diagrams of Rx Observables

```
map(x => 10 * x)
```

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 similar operators

Application Architecture



index.html

App Component

Welcome Component

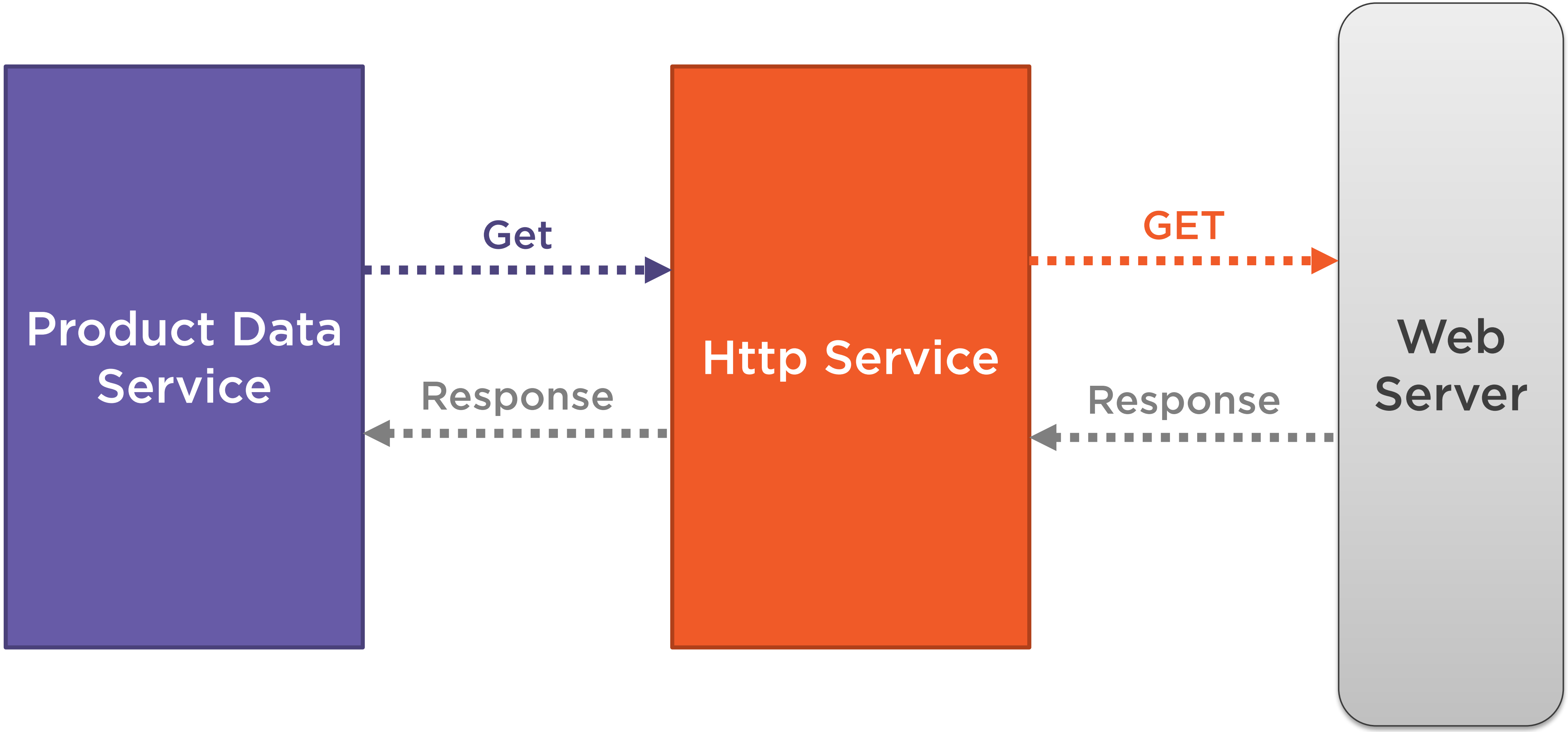
Product List Component

Product Detail Component

Star Component

Product Data Service

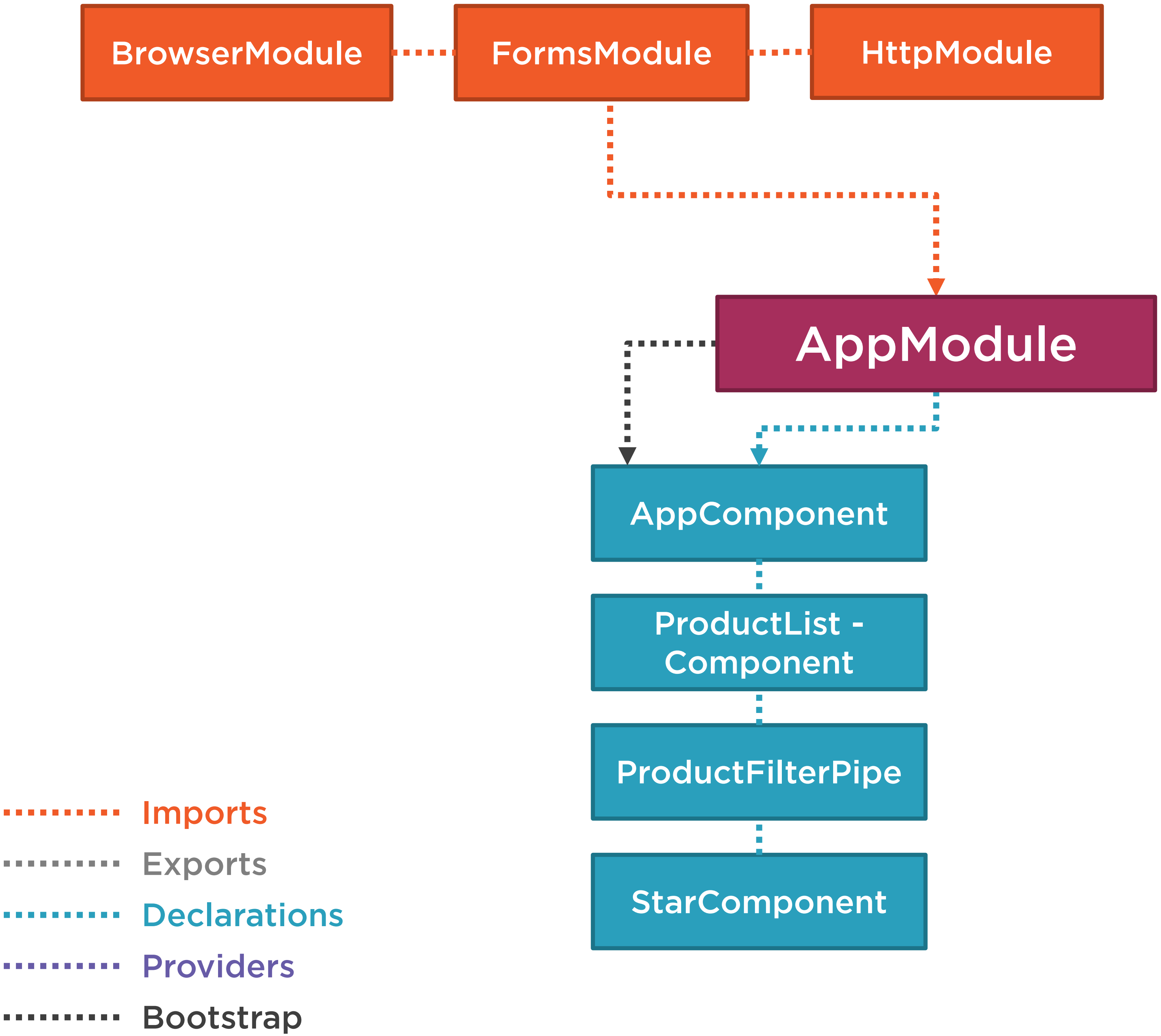
Sending an Http Request



Sending an Http Request

product.service.ts

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



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

  constructor(private _http: Http) { }

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

  }
}
```



BrowserModule

FormsModule

HttpModule

AppModule

AppComponent

ProductList - Component

ProductFilterPipe

StarComponent

Imports

Exports

Declarations

Providers

Bootstrap

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 = 'www.myWebService.com/api/products';

  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 = 'www.myWebService.com/api/products';

  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 = 'www.myWebService.com/api/products';

  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';
...

  getProducts(): Observable<IProduct[]> {
   return this._http.get(this._productUrl)
              .map((response: Response) => <IProduct[]>response.json())
              .do(data => console.log('All: ' + JSON.stringify(data)))
              .catch(this.handleError);
  }

  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);
}
```

Http Checklist: Setup

Add HttpModule to the imports array of one of the application's Angular Modules

Http Checklist: Service

Import what we need

Define a dependency for the http client service

- Use a constructor parameter

Create a method for each http request

Call the desired http method, such as get

- Pass in the Url

Map the Http response to a JSON object

Add error handling

Http Checklist: Subscribing

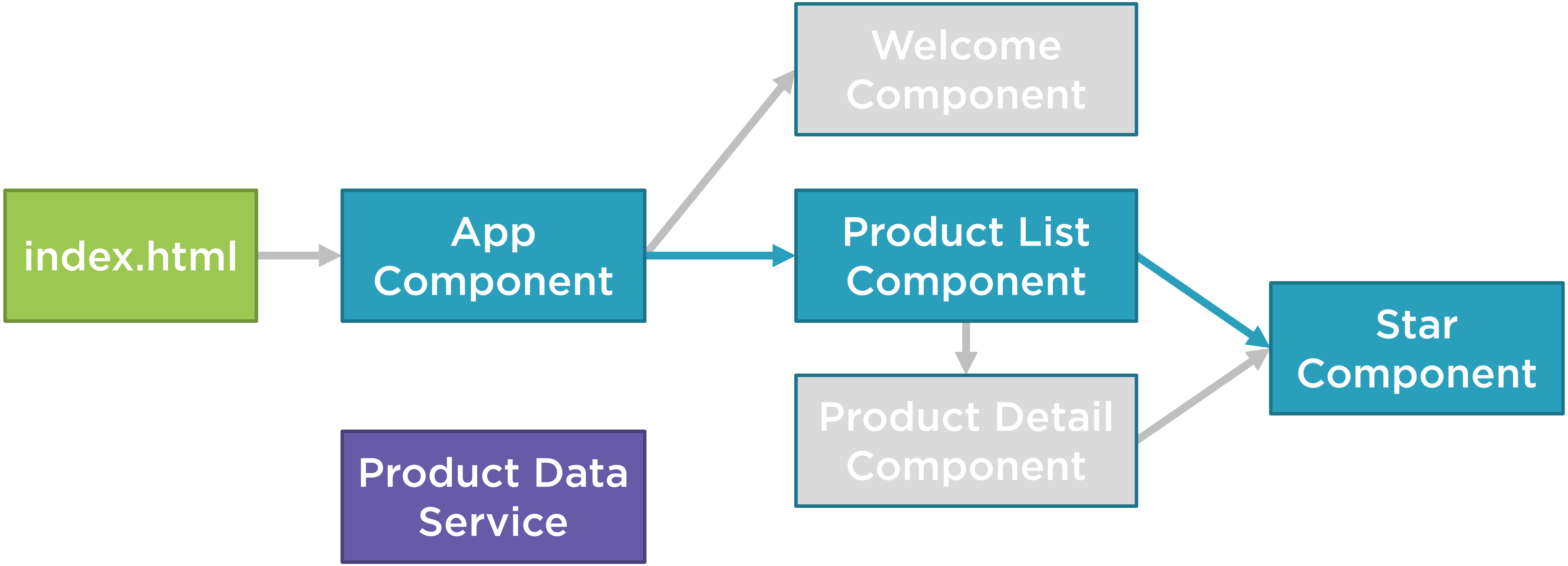
Call the subscribe method of the returned observable

Provide a function to handle an emitted item

- Normally assigns a property to the returned JSON object

Provide an error function to handle any returned errors

Application Architecture



index.html

App Component

Welcome Component

Product List Component

Product Detail Component

Star Component

Product Data Service

Navigation and Routing Basics

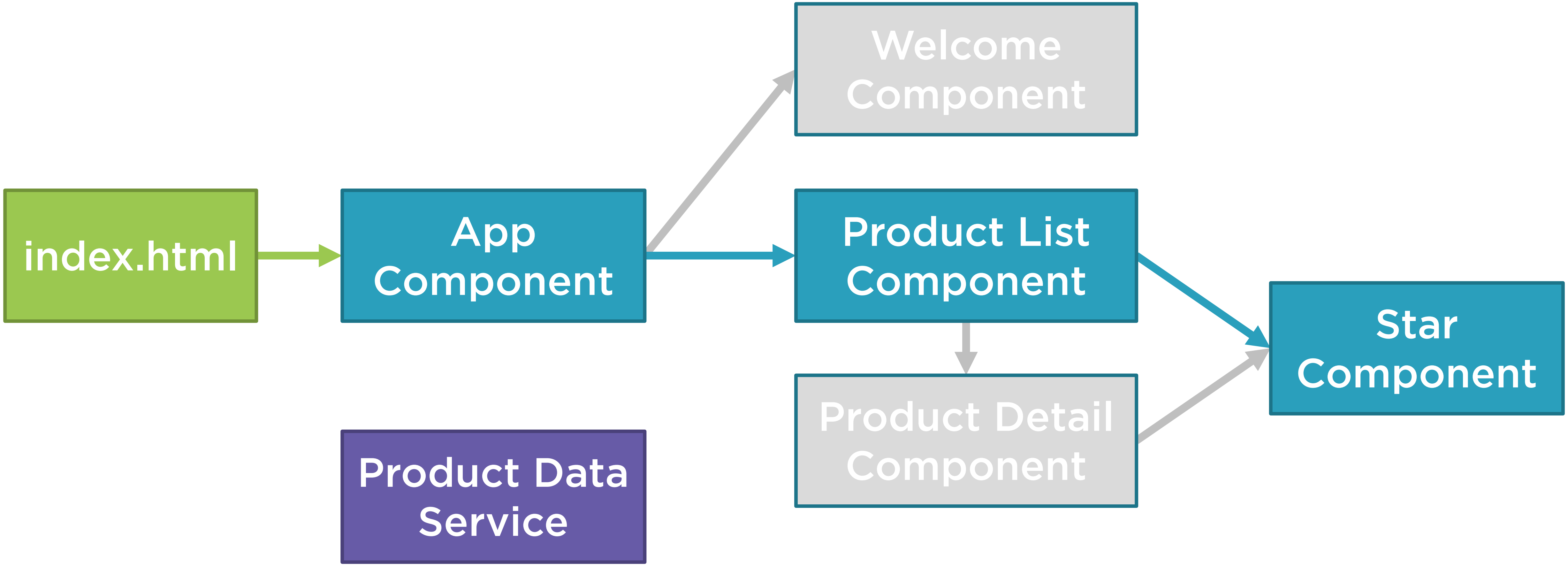
How Does Routing Work?

Configuring Routes

Tying Routes to Actions

Placing the Views

Application Architecture



index.html

App Component

Welcome Component

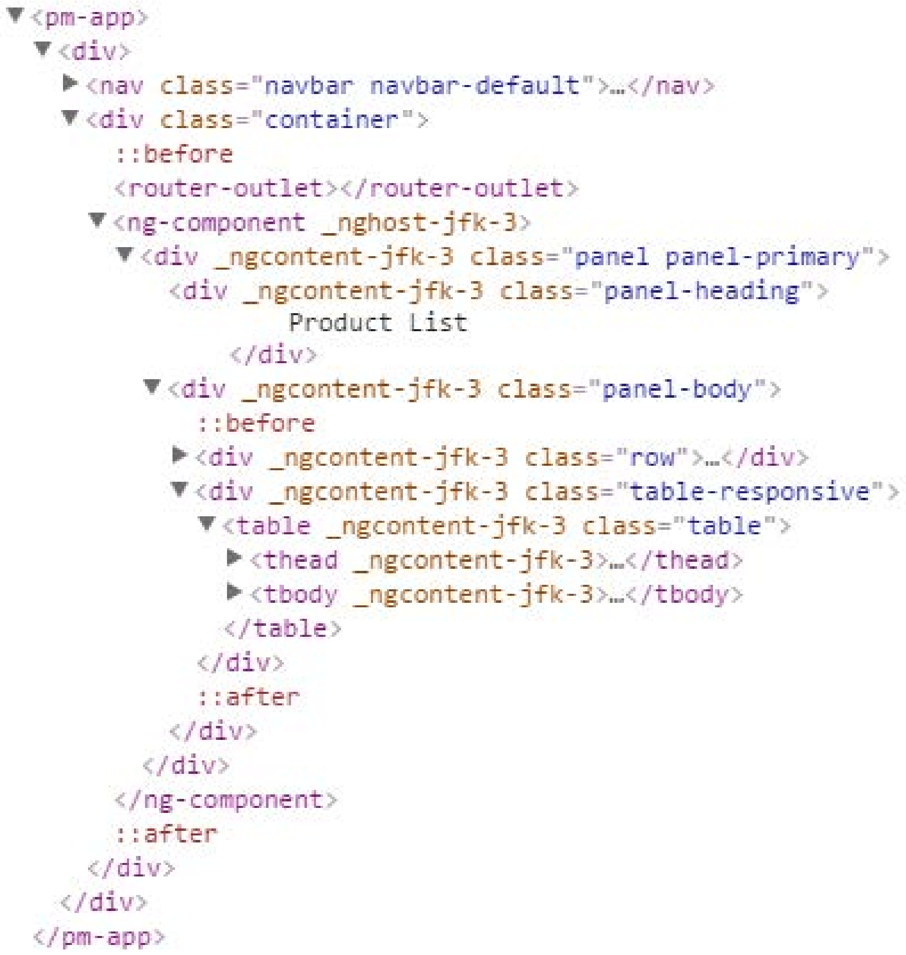
Product List Component

Product Detail Component

Star Component

Product Data Service

How Routing Works



```
▼<pm-app>
  ▼<div>
    ▶<nav class="navbar navbar-default">…</nav>
    ▼<div class="container">
        ::before
        <router-outlet></router-outlet>
      ▼<ng-component _nghost-jfk-3>
        ▼<div _ngcontent-jfk-3 class="panel panel-primary">
            <div _ngcontent-jfk-3 class="panel-heading">
                    Product List
                </div>
          ▼<div _ngcontent-jfk-3 class="panel-body">
              ::before
            ▶<div _ngcontent-jfk-3 class="row">…</div>
            ▼<div _ngcontent-jfk-3 class="table-responsive">
              ▼<table _ngcontent-jfk-3 class="table">
                ▶<thead _ngcontent-jfk-3>…</thead>
                ▶<tbody _ngcontent-jfk-3>…</tbody>
                </table>
              </div>
              ::after
            </div>
          </div>
        </ng-component>
        ::after
      </div>
    </div>
  </pm-app>
```

Configure a route for each component

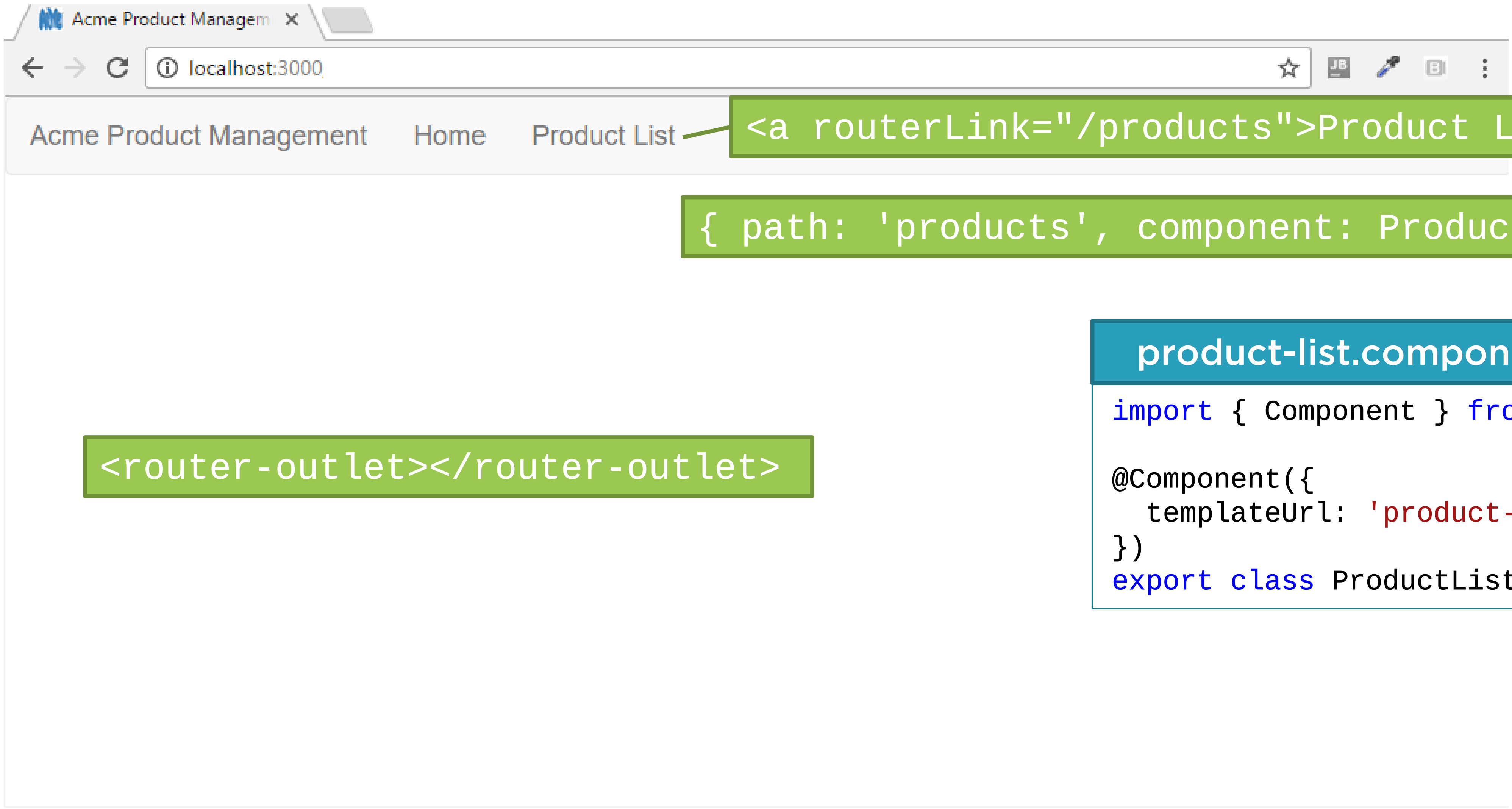
Define options/actions

Tie a route to each option/action

Activate the route based on user action

Activating a route displays the component's view

How Routing Works



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

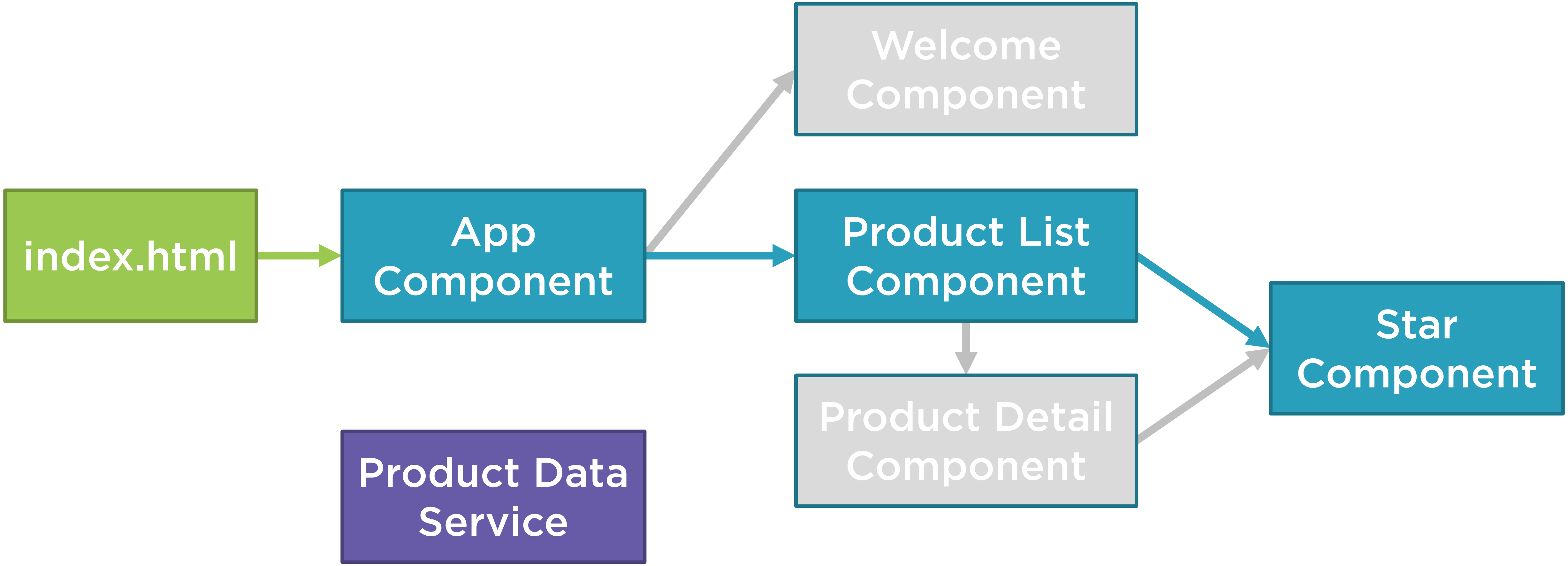
```
{ path: 'products', component: ProductListComponent }
```

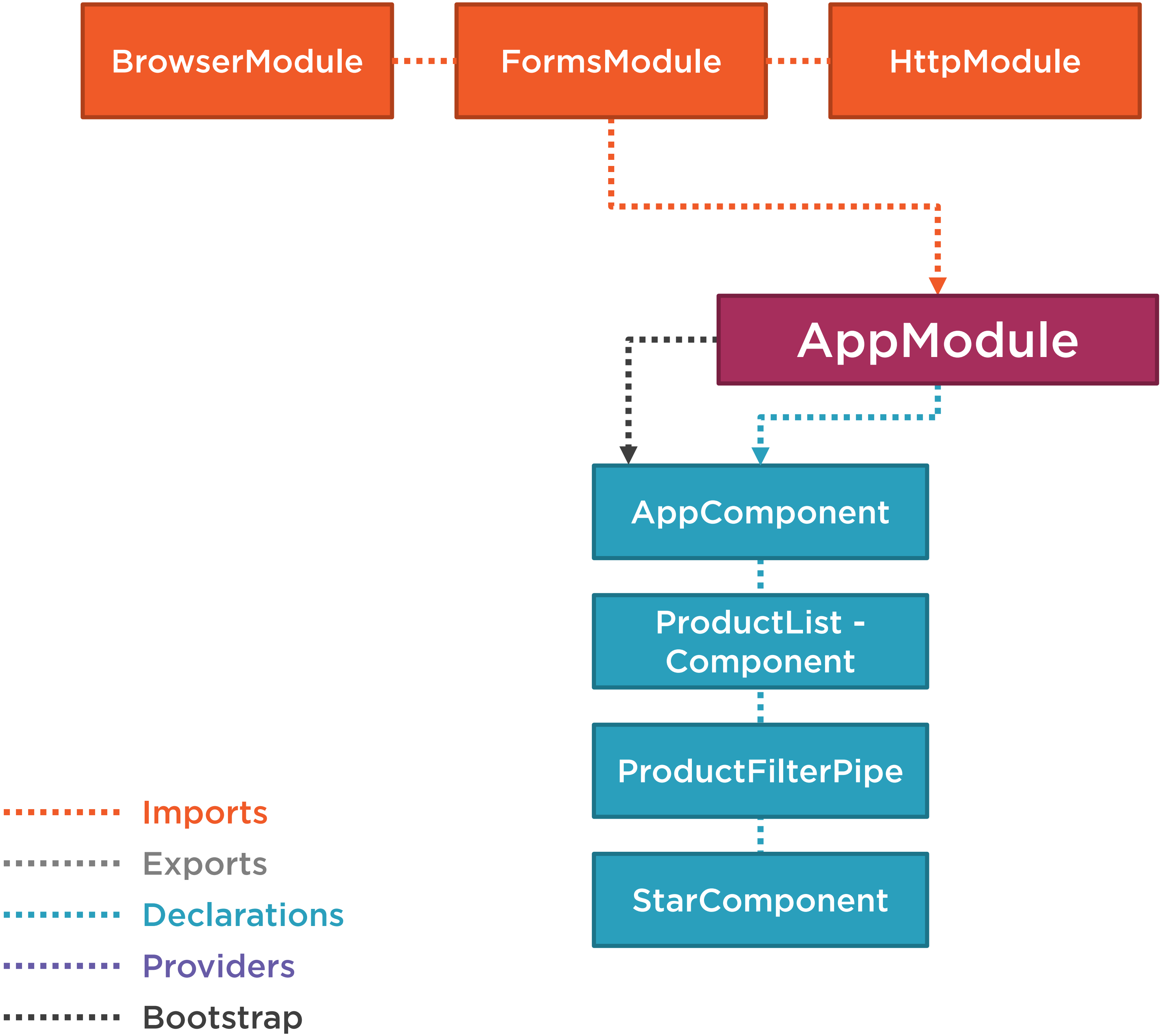
product-list.component.ts

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

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

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

Application Architecture





BrowserModule

FormsModule

HttpModule

AppModule

AppComponent

ProductList - Component

ProductFilterPipe

StarComponent

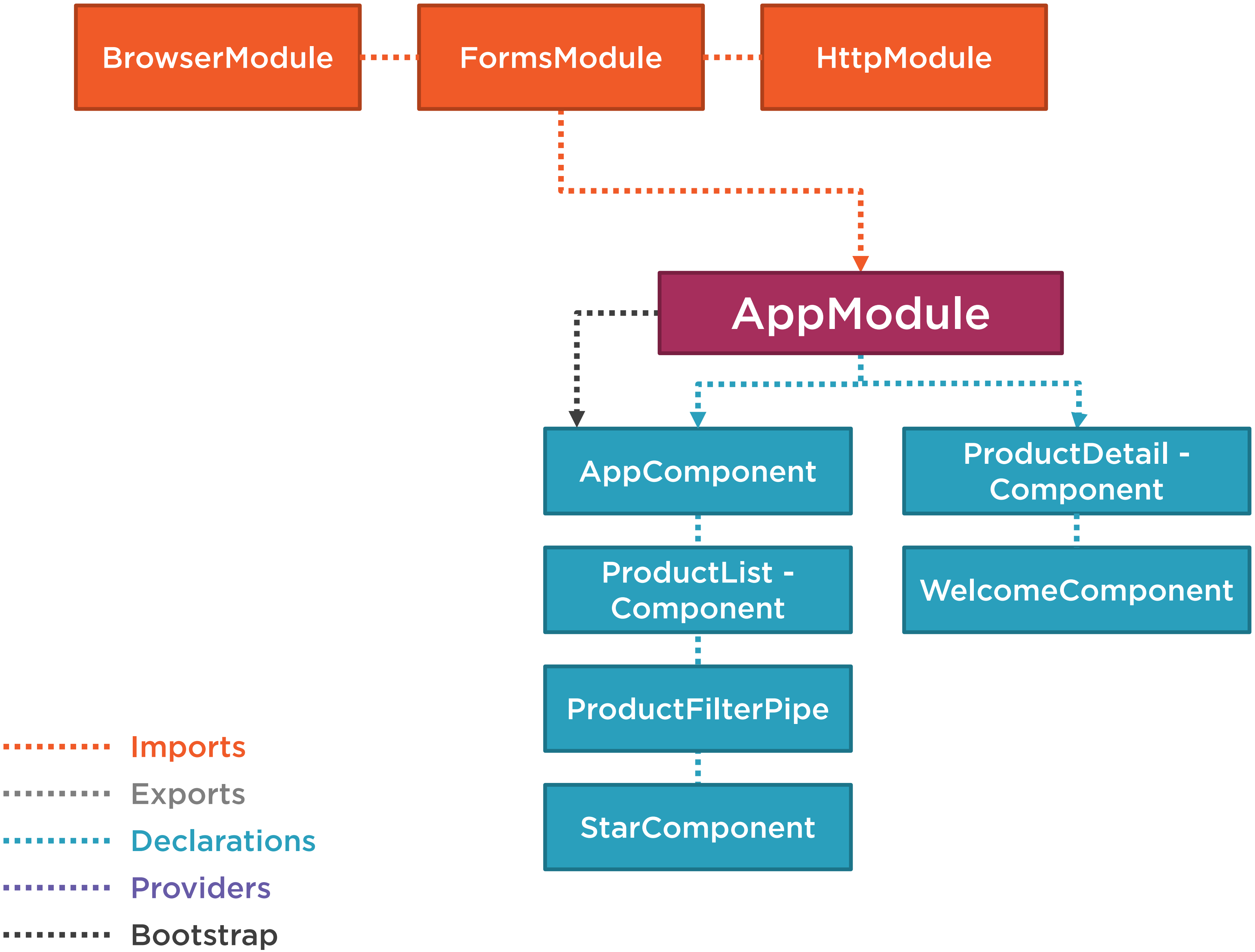
Imports

Exports

Declarations

Providers

Bootstrap



BrowserModule

FormsModule

HttpModule

AppModule

AppComponent

ProductDetail - Component

ProductList - Component

WelcomeComponent

ProductFilterPipe

StarComponent

Imports

Exports

Declarations

Providers

Bootstrap

'products', ProductListComponent

'product/:id', ProductDetailComponent

'welcome', WelcomeComponent

Configuring Routes

app.module.ts

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

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

Configuring Routes

app.module.ts

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

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

Configuring Routes

app.module.ts

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

@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    RouterModule.forRoot([], { useHash: true })
  ],
  declarations: [
    ...
  ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Configuring Routes

```
[
 { path: 'products', component: ProductListComponent },
 { path: 'product/:id', component: ProductDetailComponent },
 { path: 'welcome', component: WelcomeComponent },
 { path: '', redirectTo: 'welcome', pathMatch: 'full' },
 { path: '**', component: PageNotFoundComponent }
]
```

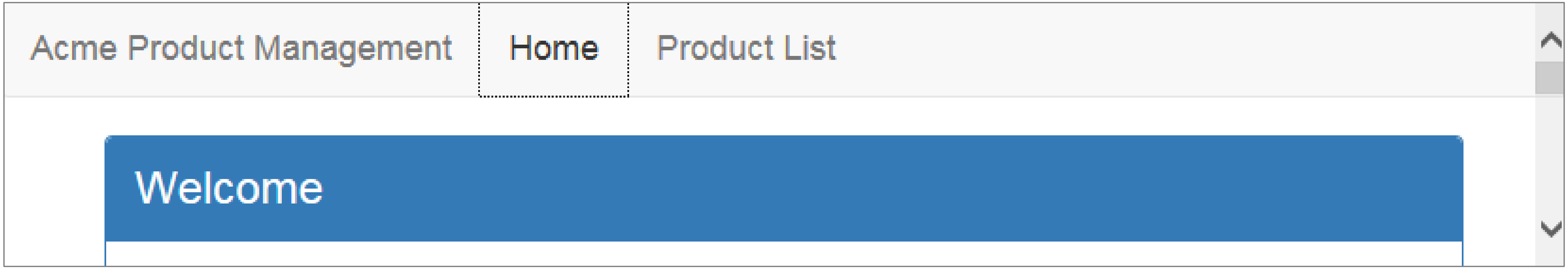
Navigating the Application Routes

Menu option, link, image or button that activates a route

Typing the Url in the address bar / bookmark

The browser's forward or back buttons

Tying Routes to Actions



Tying Routes to Actions

app.component.ts

```
...

@Component({
    selector: 'pm-app',
    template: `
     <ul class='nav navbar-nav'>
       <li><a>Home</a></li>
       <li><a>Product List</a></li>
     </ul>
    `
})
```

Tying Routes to Actions

app.component.ts

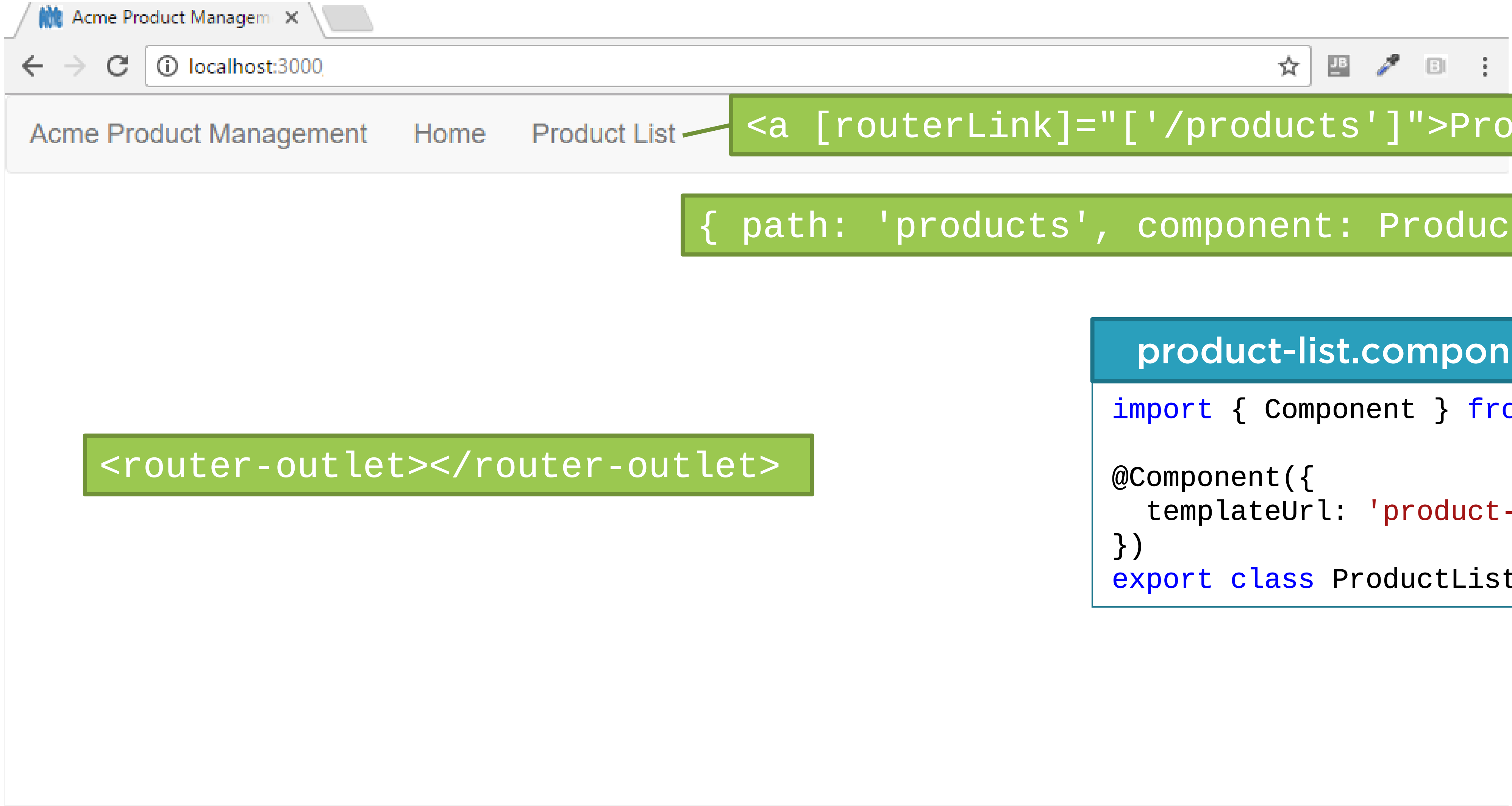
```
...

@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>
    `
})
```

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 { }
```

Checklist: Displaying Components

Nest-able components

- Define a selector
- Nest in another component
- No route

Routed components

- No selector
- Configure routes
- Tie routes to actions

Checklist: Doing Routing

Configure routes

Tie routes to actions

Place the view

Routing Checklist: Configuring Routes

Define the base element

Add RouterModule

- Add each route (RouterModule.forRoot)
- Order matters

path: Url segment for the route

- No leading slash
- '' for default route
- '**' for wildcard route

component

- Not string name; not enclosed in quotes

Routing Checklist: Tying Routes to Actions

Add the RouterLink directive as an attribute

- Clickable element
- Enclose in square brackets

Bind to a link parameters array

- First element is the path
- All other elements are route parameters

Routing Checklist: Placing the View

Add the RouterOutlet directive

- Identifies where to display the routed component's view
- Specified in the host component's template

Summary

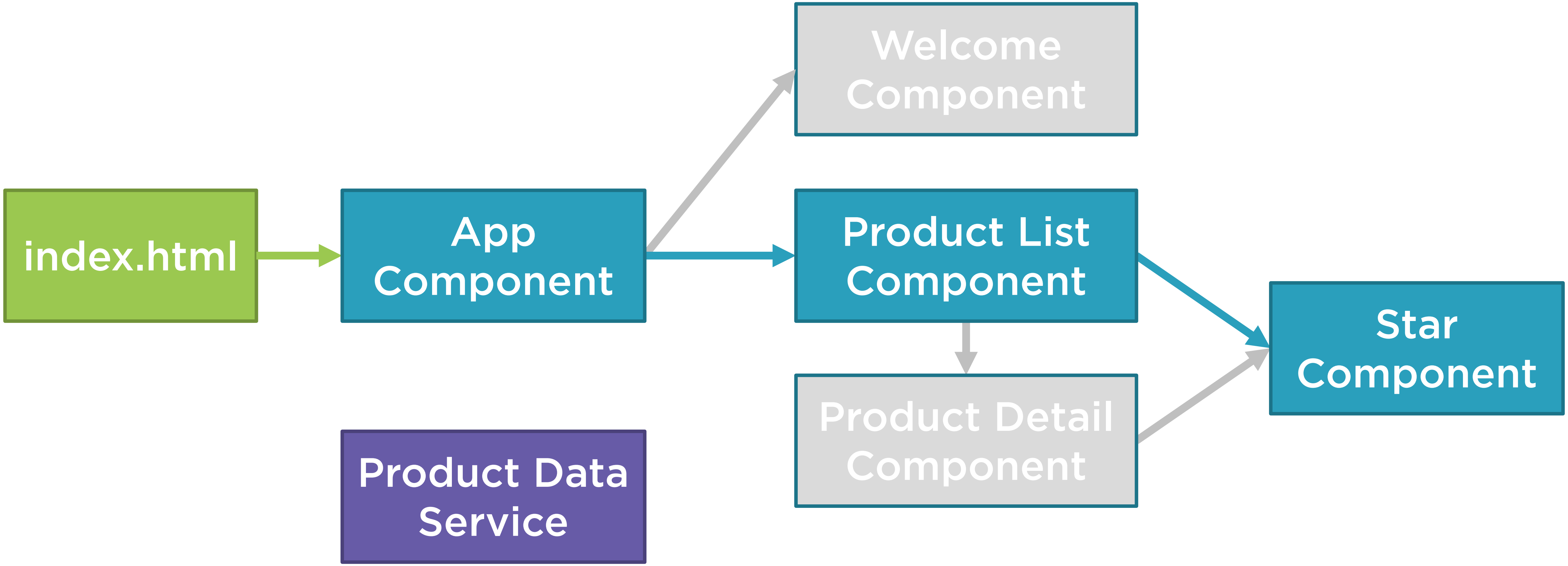
How Does Routing Work?

Configuring Routes

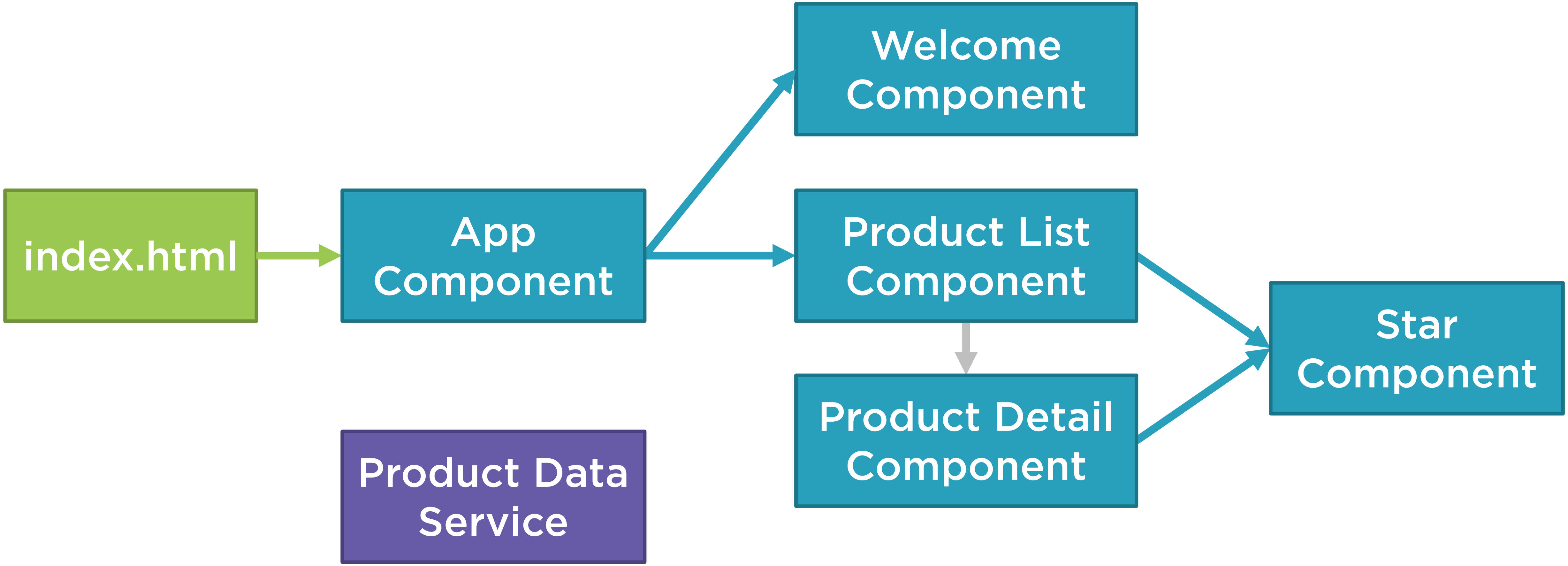
Tying Routes to Actions

Placing the Views

Application Architecture



Application Architecture



index.html

App Component

Welcome Component

Product List Component

Product Detail Component

Star Component

Product Data Service

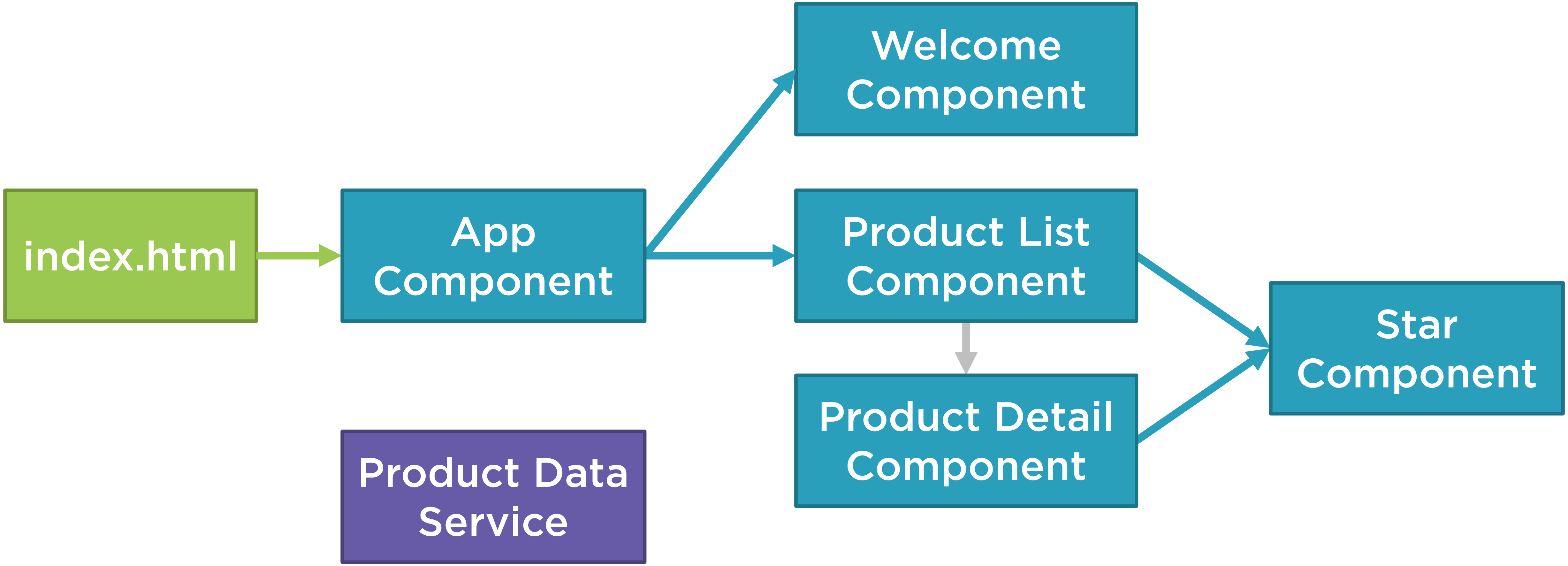
Navigation and Routing Additional Techniques

Passing Parameters to a Route

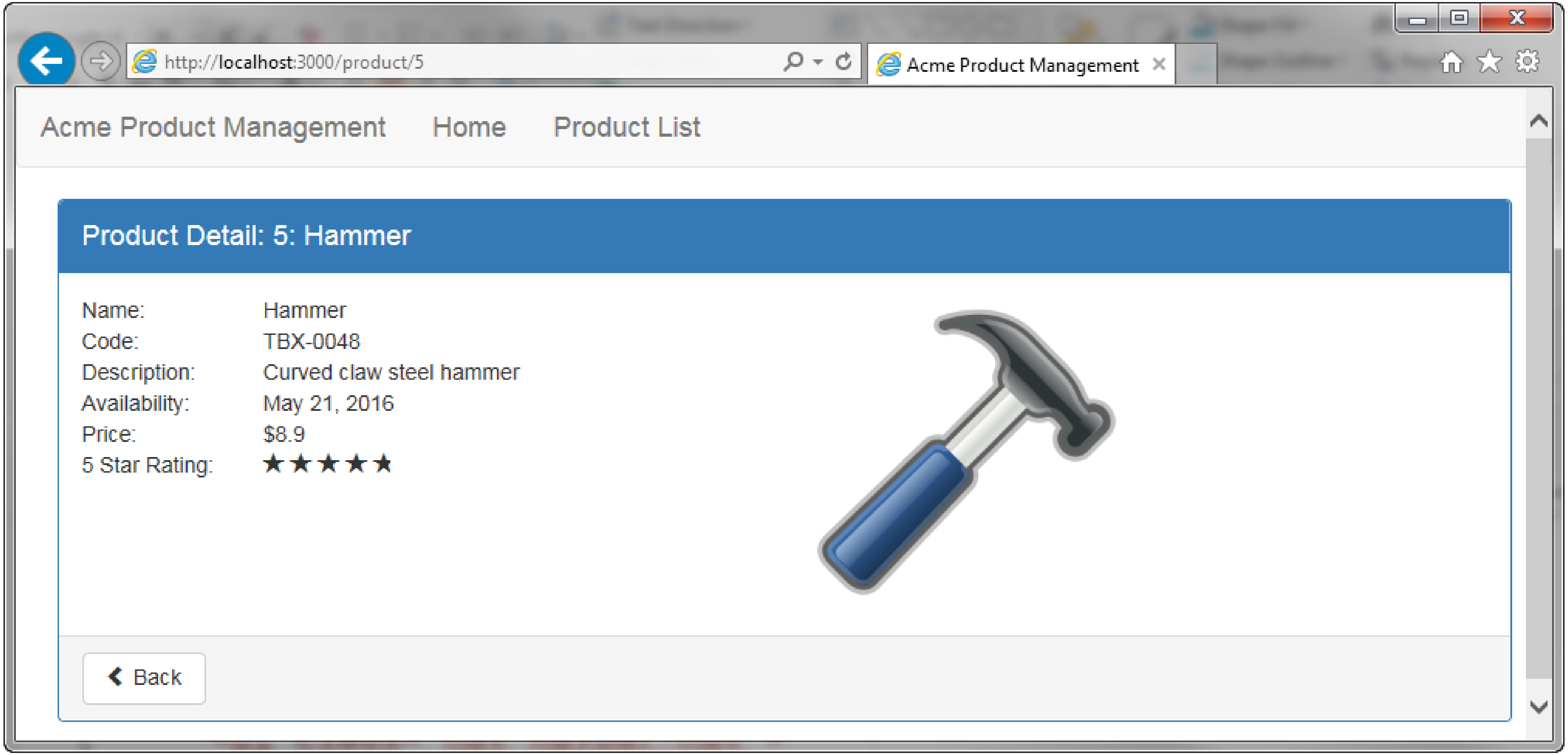
Activating a Route with Code

Protecting Routes with Guards

Application Architecture



Passing Parameters to a Route



http://localhost:3000/product/5

Acme Product Management

Product Detail: 5: Hammer

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

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

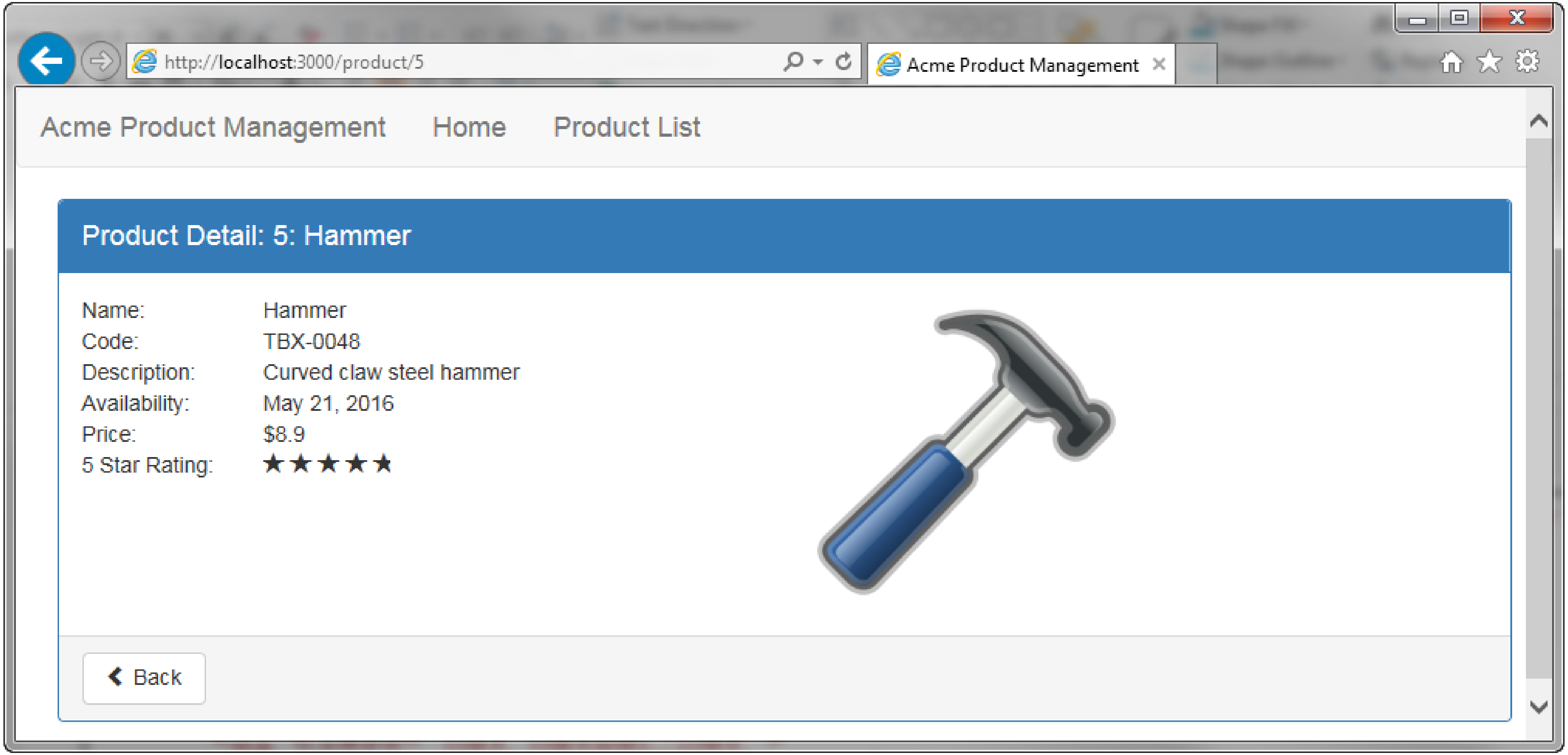
product-detail.component.ts

```
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



Product Detail: 5: Hammer

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

Activating a Route with Code

product-detail.component.ts

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

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

Protecting Routes with Guards

CanActivate

- Guard navigation to a route

CanDeactivate

- Guard navigation from a route

Resolve

- Pre-fetch data before activating a route

CanLoad

- Prevent asynchronous routing

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: [...],
  declarations: [...],
  providers: [ ProductDetailGuard ],
  bootstrap: [ AppComponent ]
})
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 { }
```

Routing Checklist: Passing Parameters

app.module.ts

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

product-list.component.html

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

product-detail.component.ts

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

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

Routing Checklist: Activate a Route with Code

Use the Router service

- Import the service
- Define it as a dependency

Create a method that calls the navigate method of the Router service

- Pass in the link parameters array

Add a user interface element

- Use event binding to bind to the created method

Routing Checklist: Protecting Routes with Guards

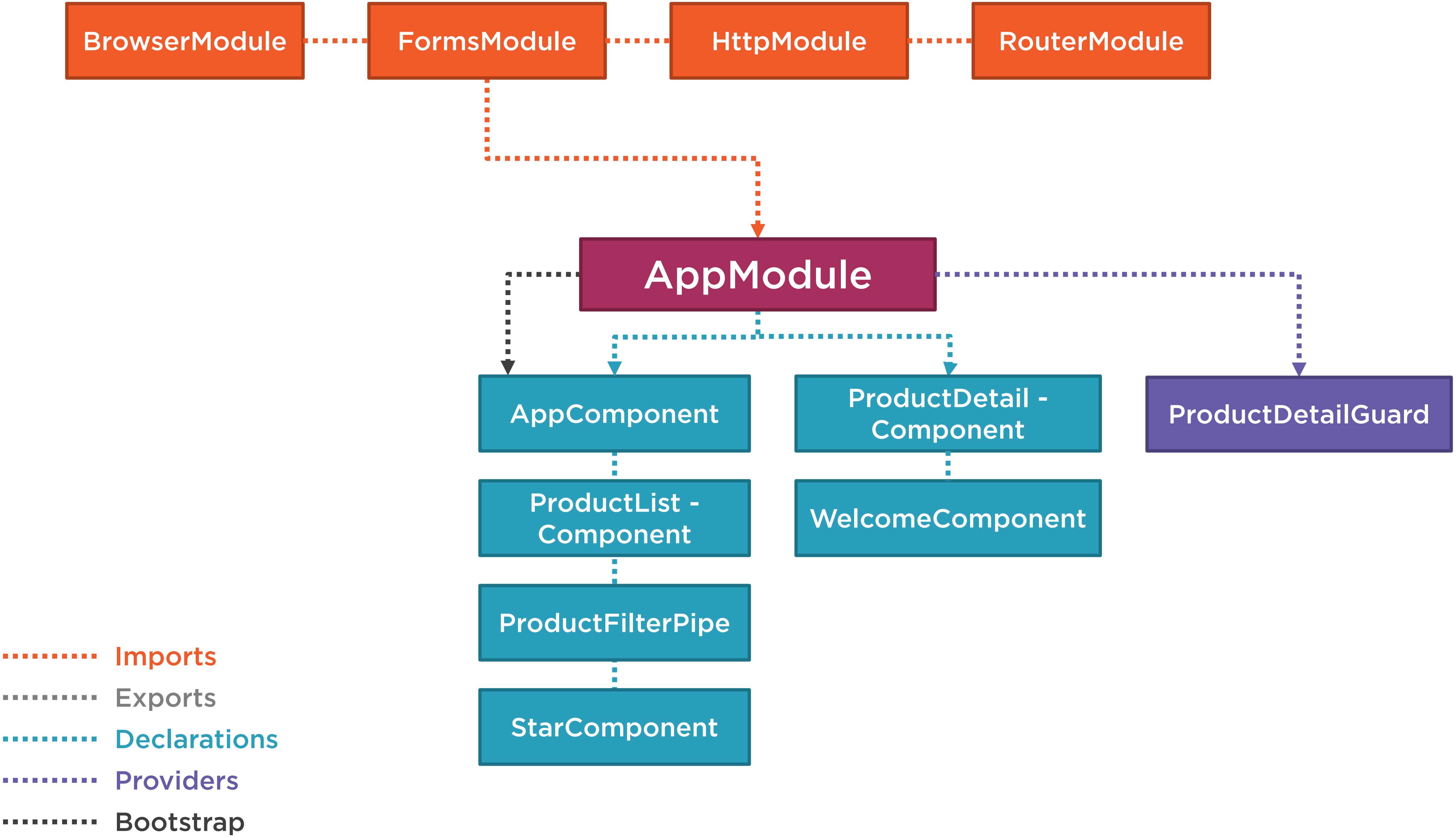
Build a guard service

- Implement the guard type (CanActivate)
- Create the method (canActivate())

Register the guard service provider

- Must be in an Angular module

Add the guard to the desired route



BrowserModule

FormsModule

HttpModule

RouterModule

AppModule

AppComponent

ProductList - Component

ProductFilterPipe

StarComponent

ProductDetail - Component

WelcomeComponent

ProductDetailGuard

Imports

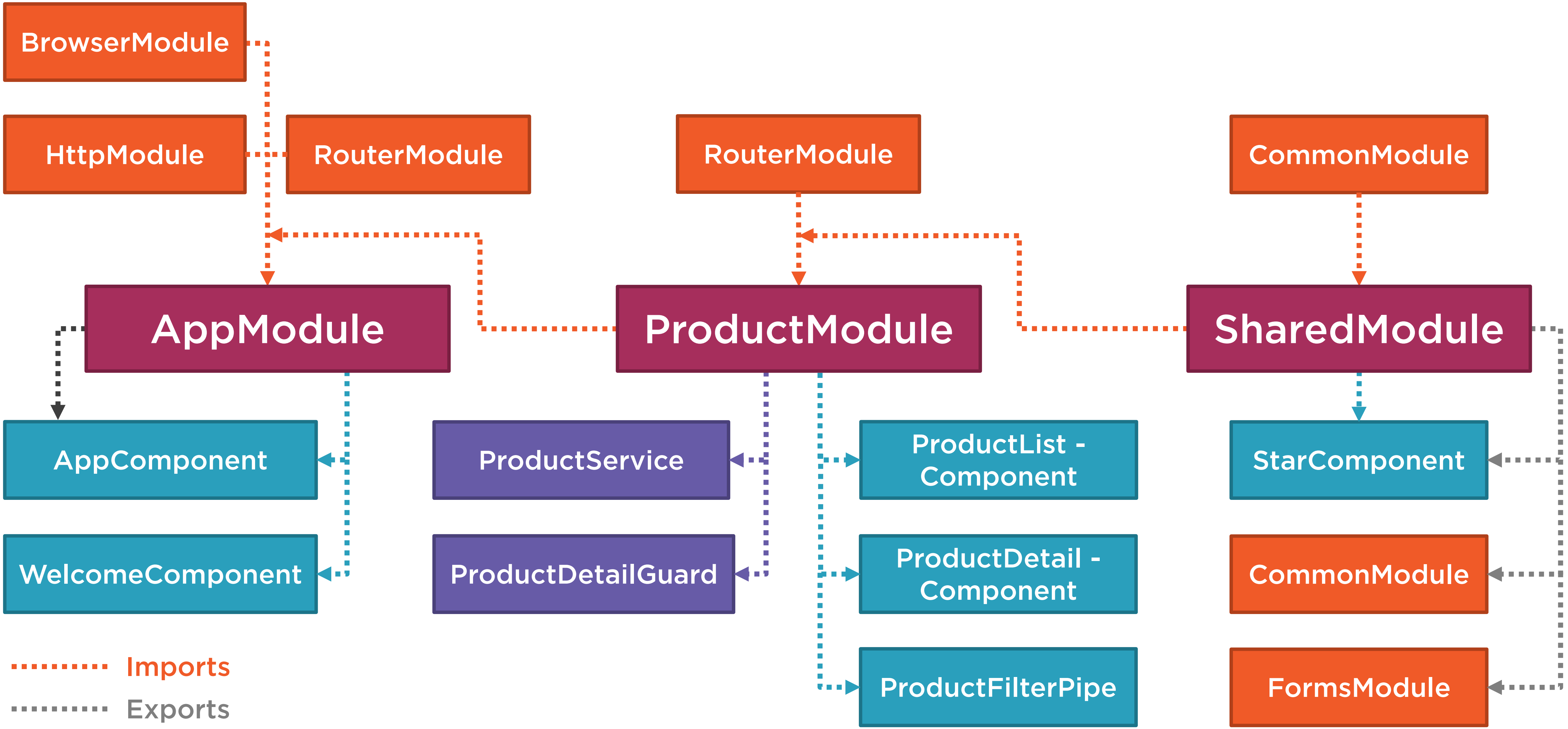
Exports

Declarations

Providers

Bootstrap

Angular Modules



- Imports
- Exports
- Declarations
- Providers
- Bootstrap

What Is an Angular Module?

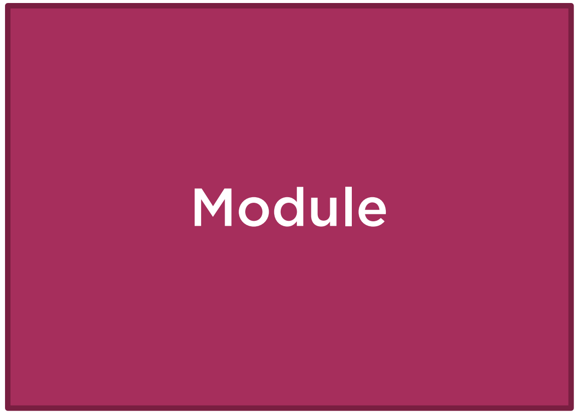
Angular Module Metadata

Creating a Feature Module

Defining a Shared Module

Revisiting AppModule

What Is an Angular Module?



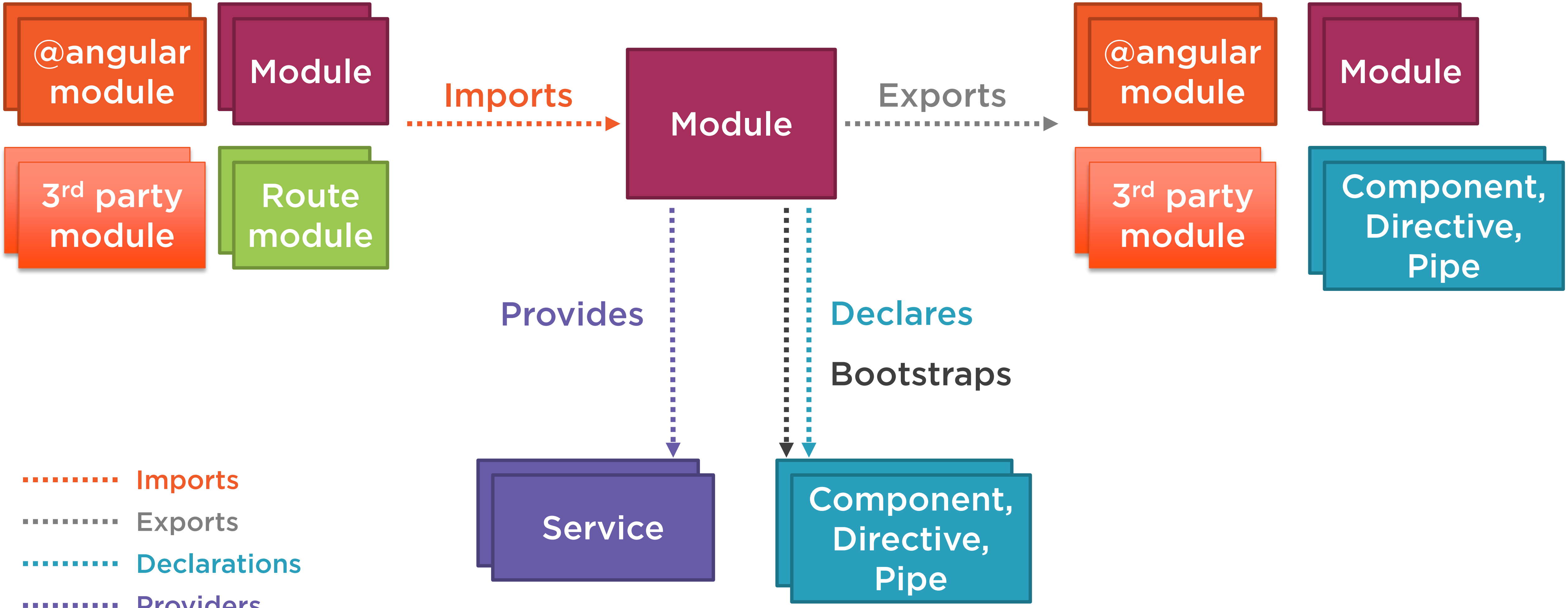
Module

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

Angular Module



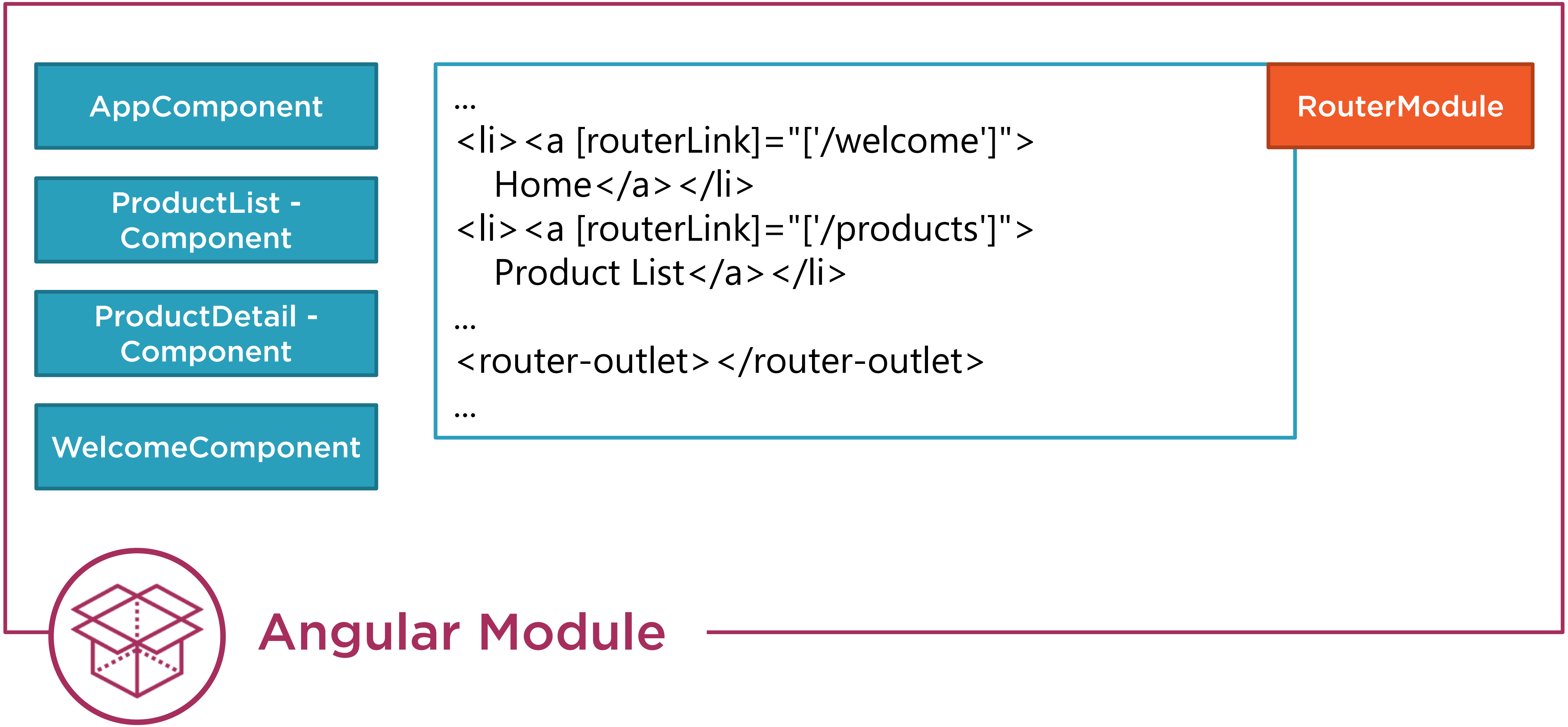
Imports

Exports

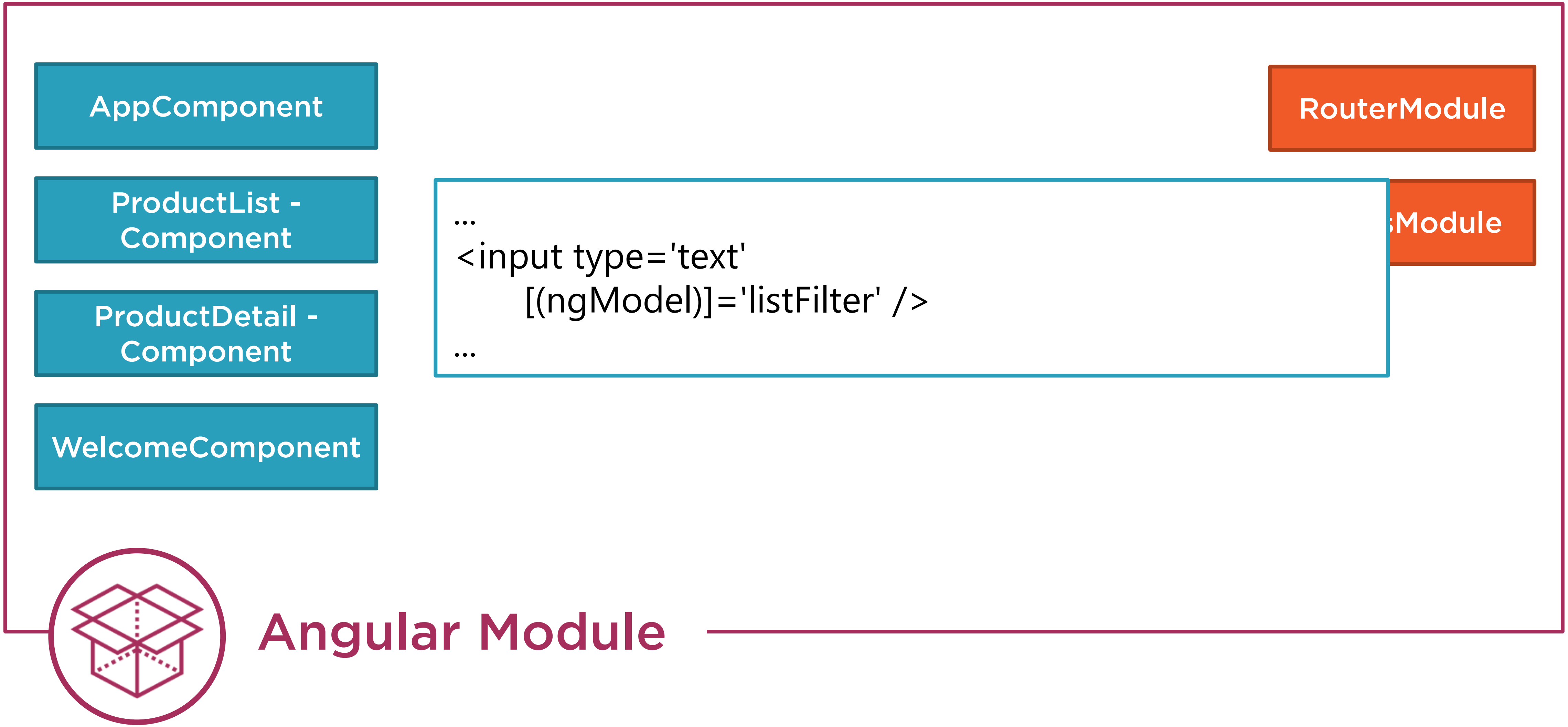
Declarations

Providers

Bootstrap



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



AppComponent

ProductList - Component

ProductDetail - Component

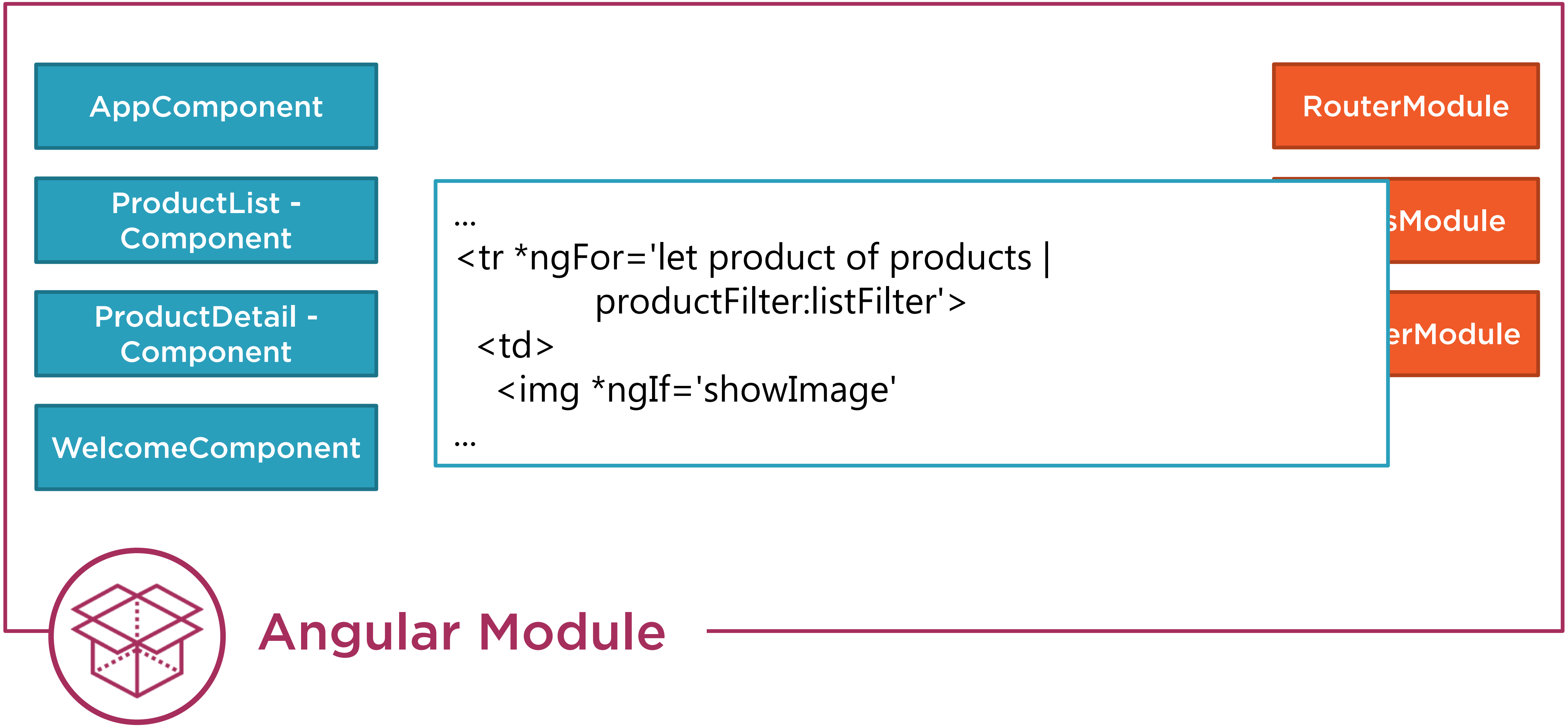
WelcomeComponent

RouterModule

sModule

```
...
<input type='text'
       [(ngModel)]='listFilter' />
...
```

Angular Module



AppComponent

ProductList - Component

ProductDetail - Component

WelcomeComponent

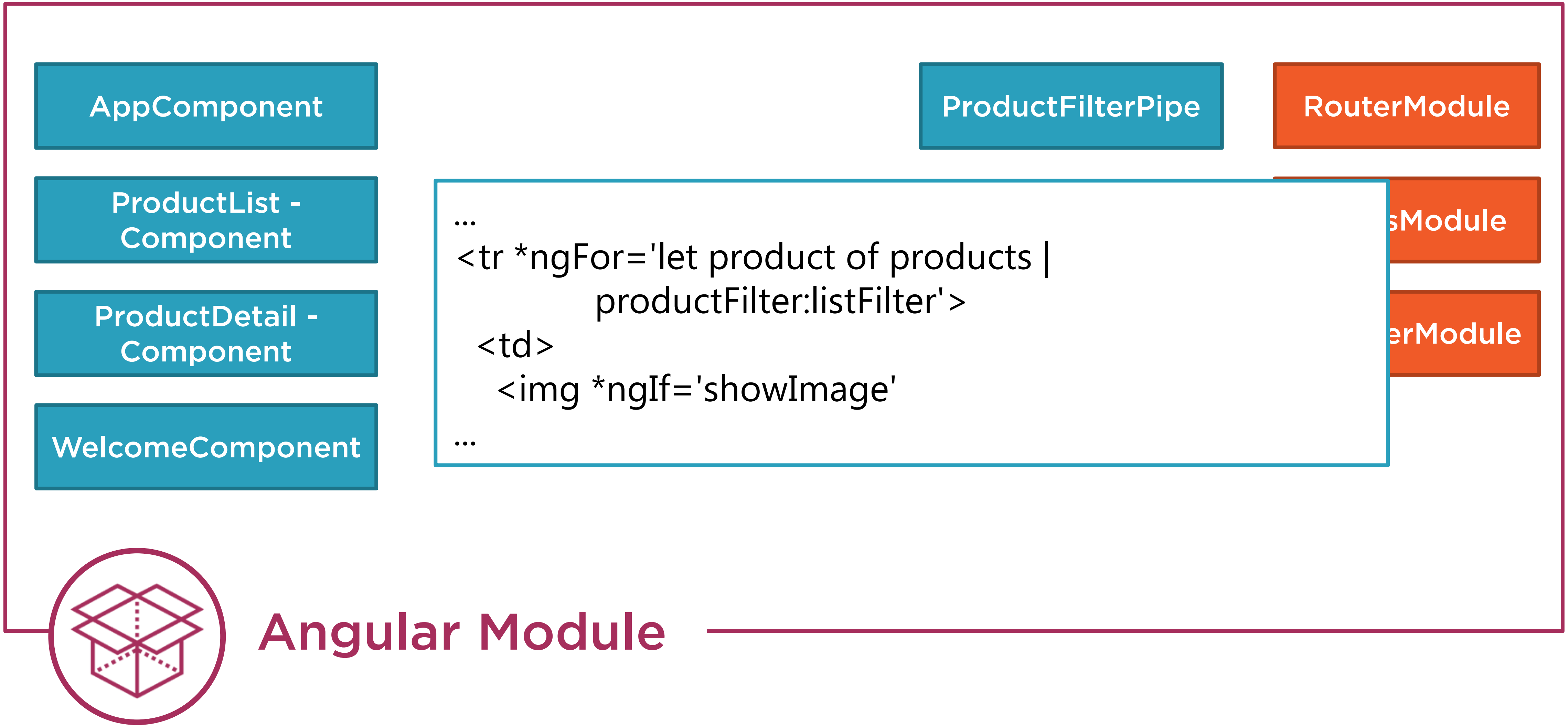
RouterModule

sModule

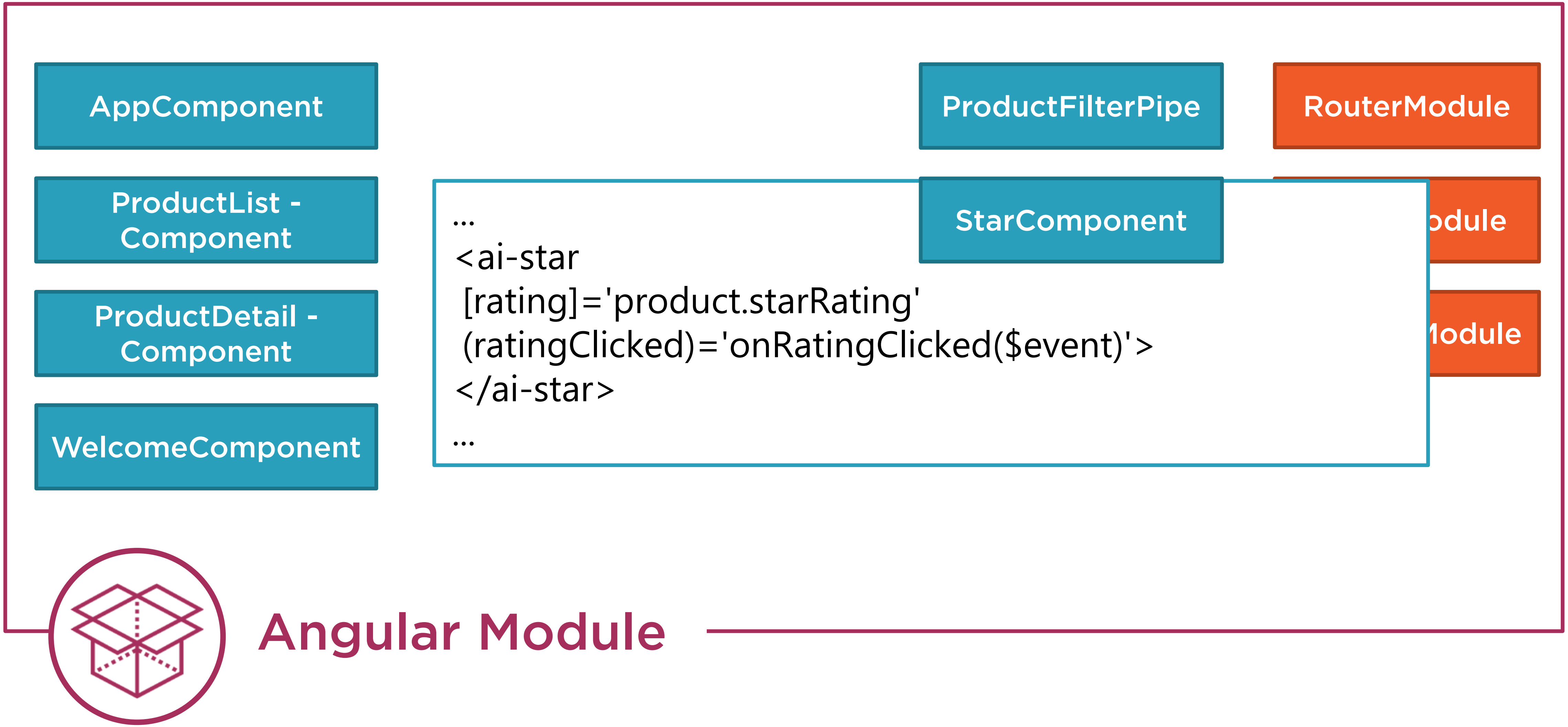
erModule

```
...
<tr *ngFor='let product of products |
              productFilter:listFilter'>
  <td>
    <img *ngIf='showImage'
...
```

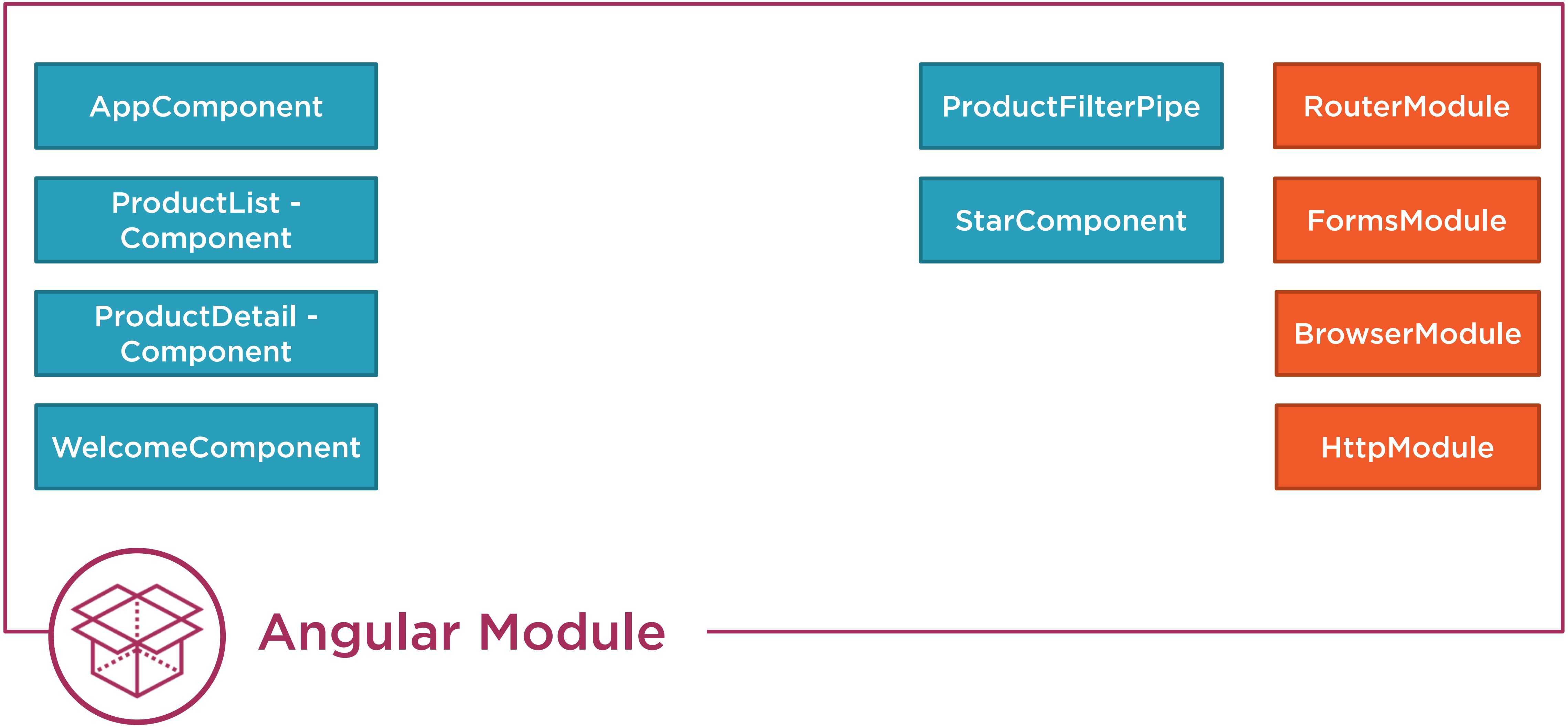
Angular Module



```
...
<tr *ngFor='let product of products |
            productFilter:listFilter'>
  <td>
    <img *ngIf='showImage'
...
```



```
...
<ai-star
 [rating]='product.starRating'
 (ratingClicked)='onRatingClicked($event)'>
</ai-star>
...
```



AppComponent

ProductList - Component

ProductDetail - Component

WelcomeComponent

ProductFilterPipe

StarComponent

RouterModule

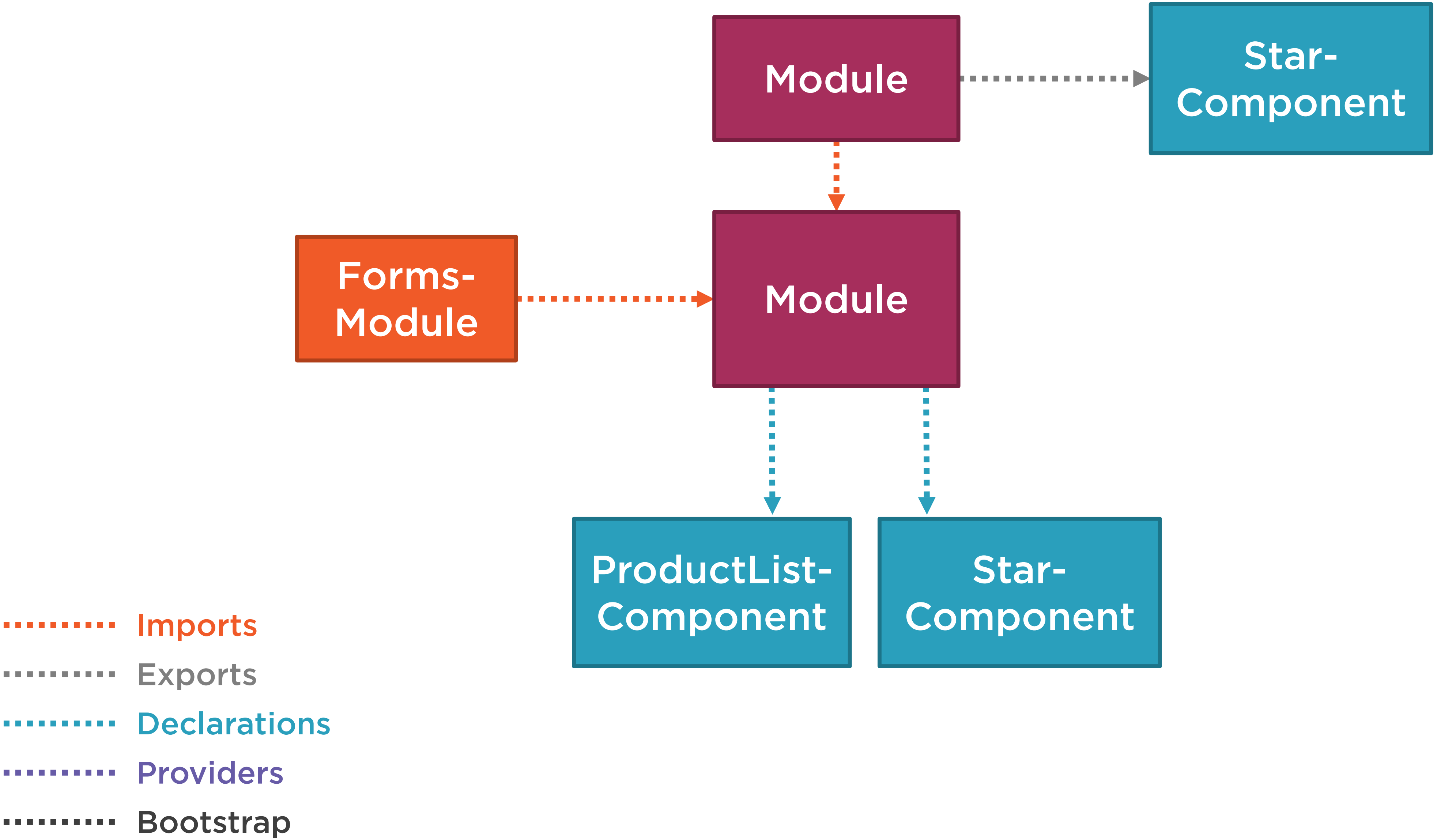
FormsModule

BrowserModule

HttpModule

Angular Module

Template Resolution Environment



Module

Star-Component

Forms-Module

Module

ProductList-Component

Star-Component

Imports

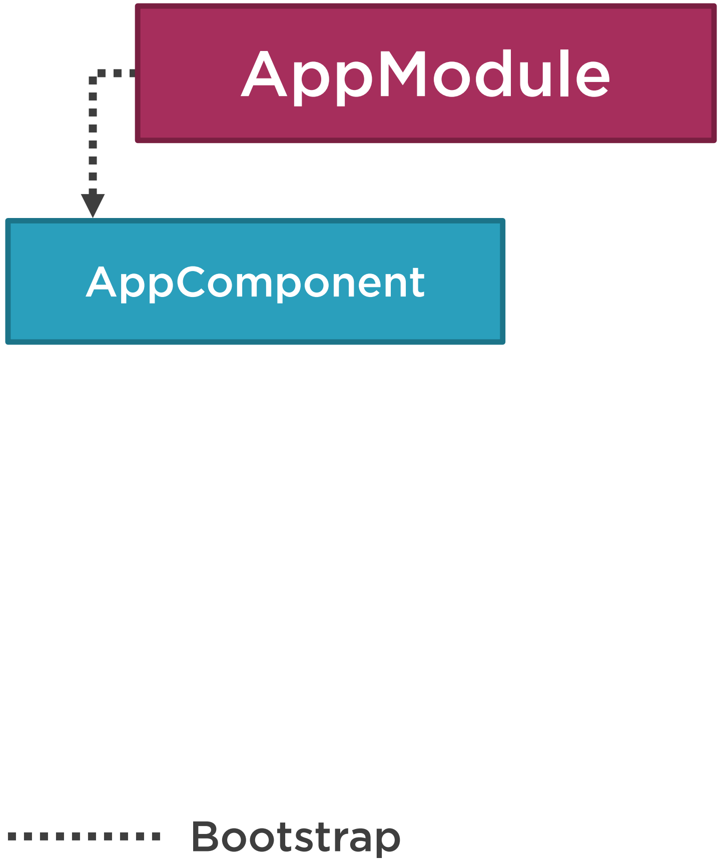
Exports

Declarations

Providers

Bootstrap

Bootstrap Array



app.module.ts

```
…
bootstrap: [ AppComponent ]
…
```

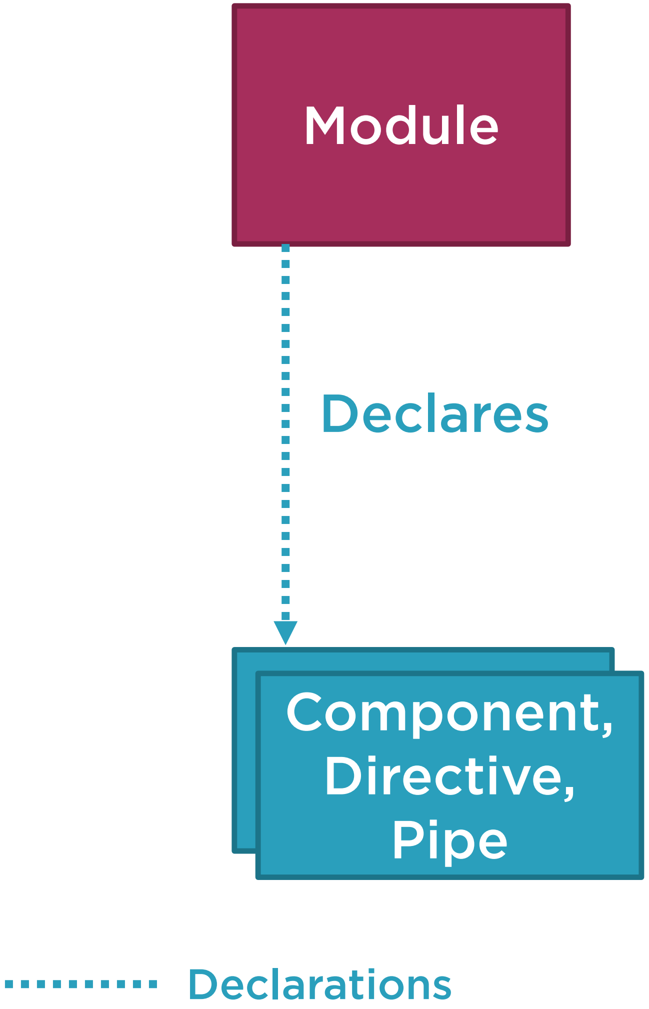
Bootstrap Array Truth #1

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

Bootstrap Array Truth #2

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

Declarations Array



Module

Declares

Component,
Directive,
Pipe

Declarations

app.module.ts

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

Declarations Array Truth #1

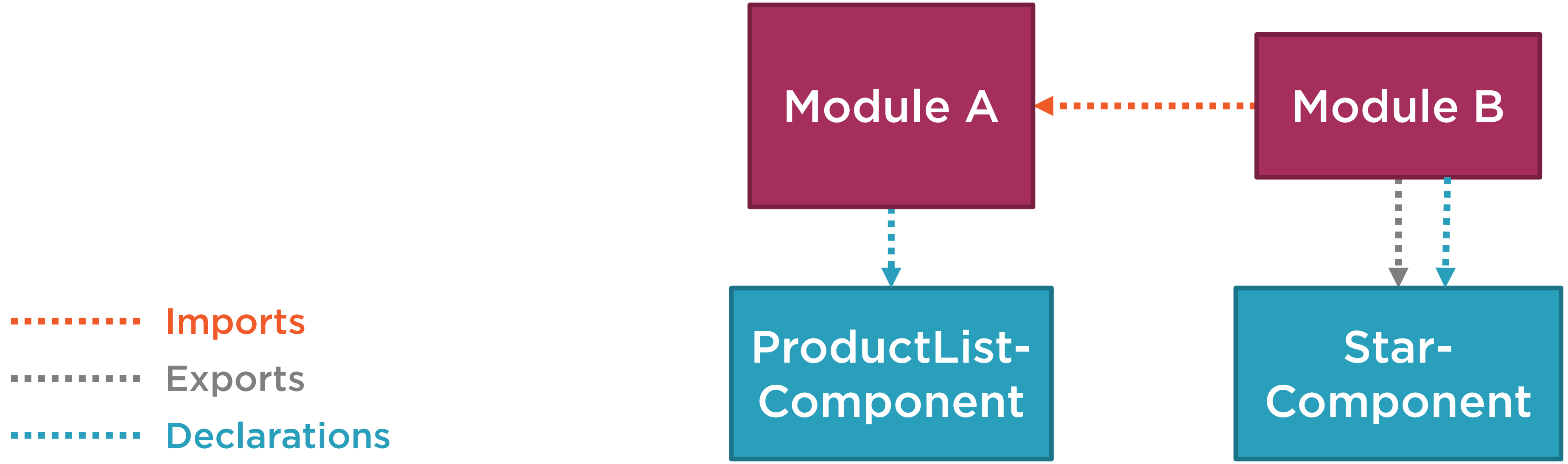
Every component, directive, and pipe we create must belong to one and only one Angular module.

Declarations Array Truth #2

Only declare components, directives and pipes.

Declarations Array Truth #3

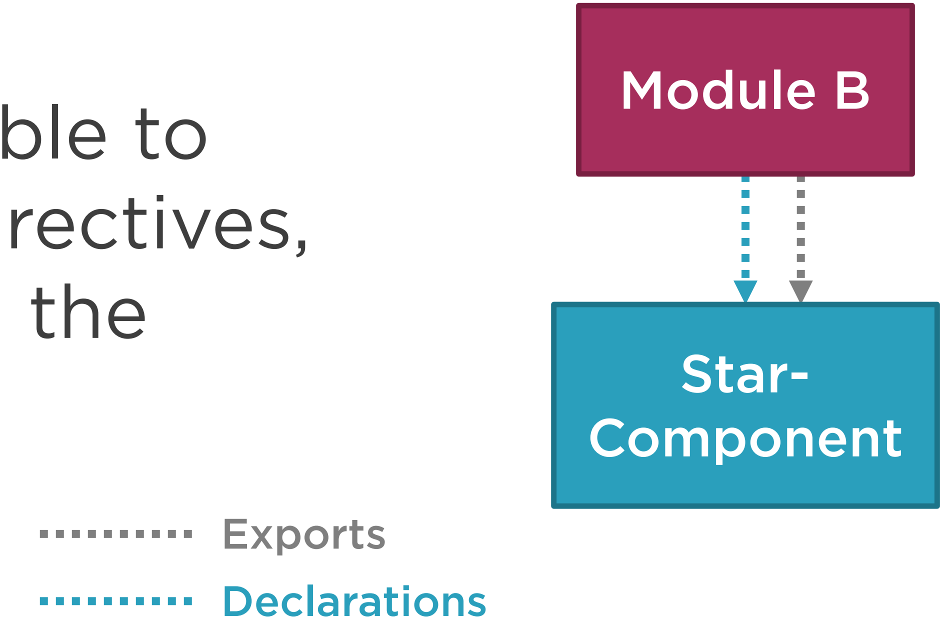
Never re-declare components, directives, or pipes that belong to another module



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.



Declarations Array Truth #5

The Angular module provides the template resolution environment for its component templates.

product-list.component.html

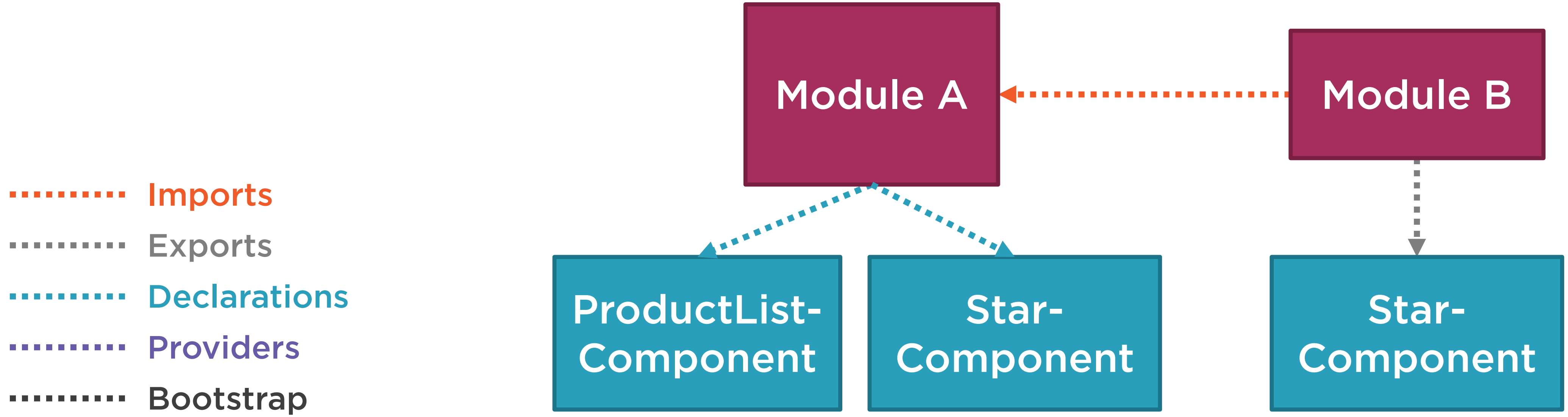
```
<ai-star …>
</ai-star>
```

star.component.ts

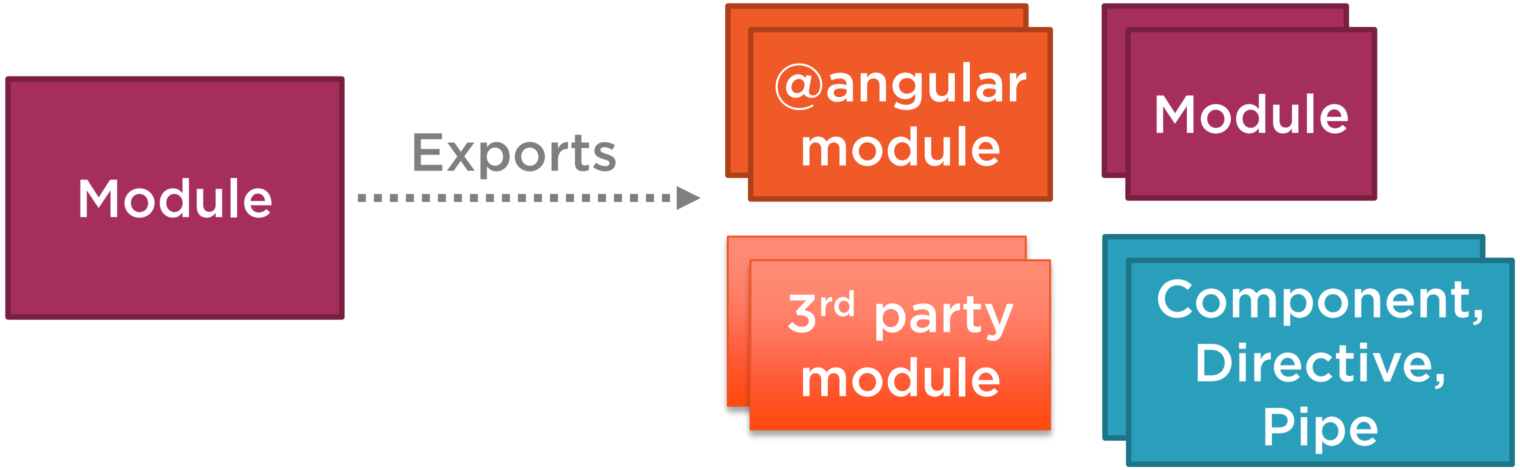
```
…
@Component({
   selector: 'ai-star',
   template: …
})
…
```

Declarations Array Truth #5

The Angular module provides the template resolution environment for its component templates.



Exports Array



Module

Exports

@angular module

Module

3rd party module

Component, Directive, Pipe

┈┈┈ Imports

┈┈┈ Exports

┈┈┈ Declarations

┈┈┈ Providers

┈┈┈ Bootstrap

Exports Array Truth #1

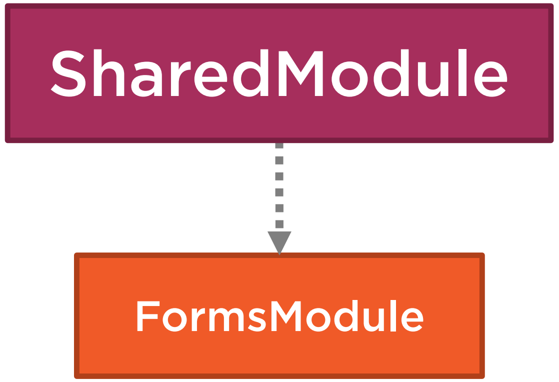
Export any component, directive, or pipe if another components need it.

Exports Array Truth #2

Re-export modules to re-export their components, directives, and pipes.

Exports Array Truth #3

We can re-export something without importing it first.



Imports

Exports

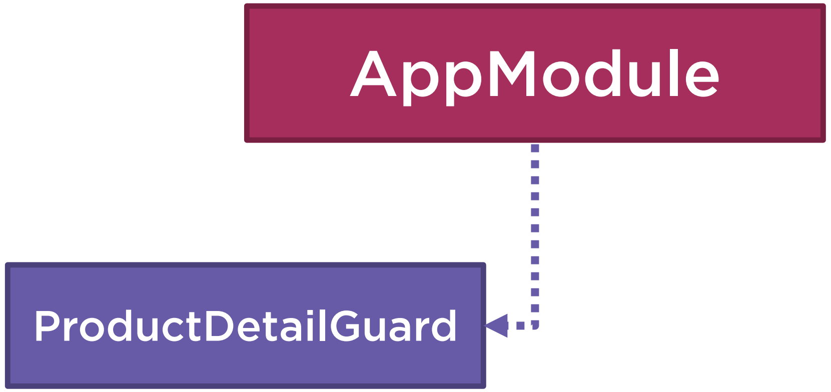
Declarations

Providers

Bootstrap

Exports Array Truth #4

Never export a service.



········· Imports

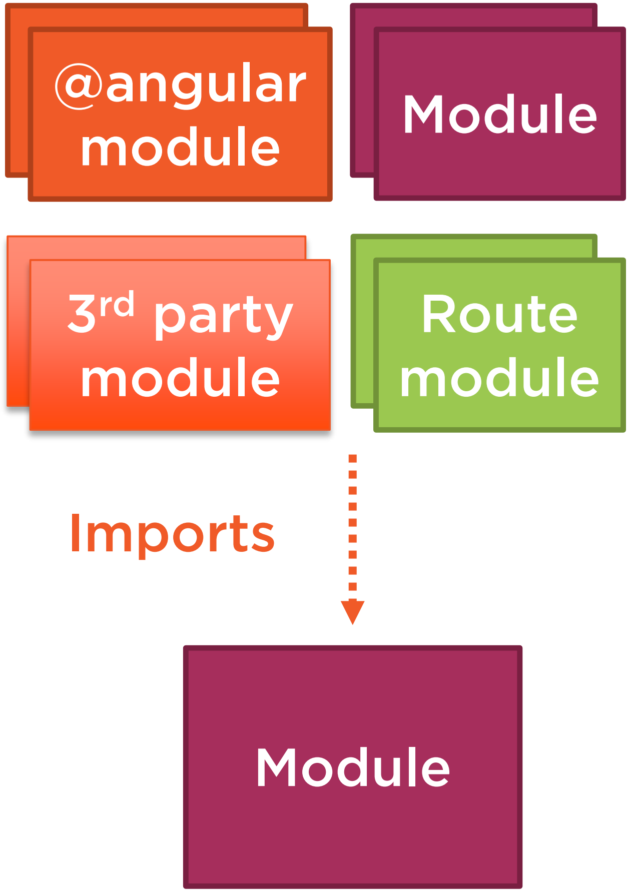
········· Exports

········· Declarations

········· Providers

········· Bootstrap

Imports Array



@angular module

Module

3rd party module

Route module

Imports

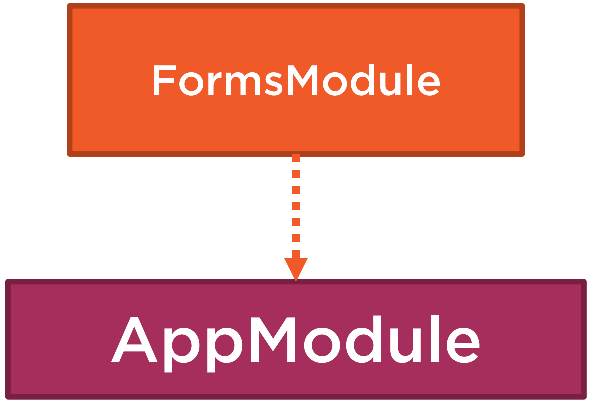
Module

app.module.ts

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

Imports Array Truth #1

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



FormsModule

AppModule

┈┈┈ Imports

┈┈┈ Exports

┈┈┈ Declarations

┈┈┈ Providers

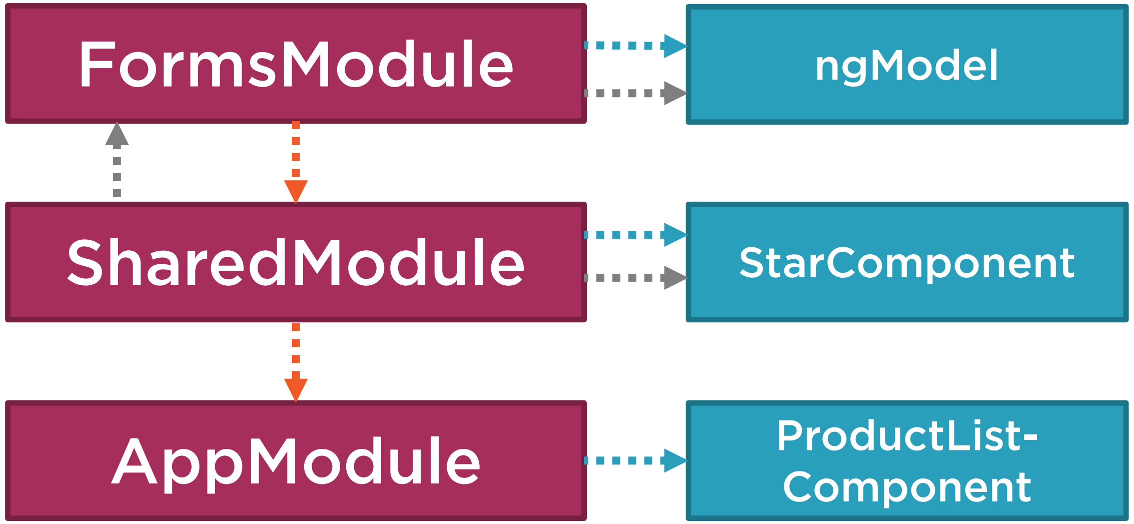
┈┈┈ Bootstrap

Imports Array Truth #2

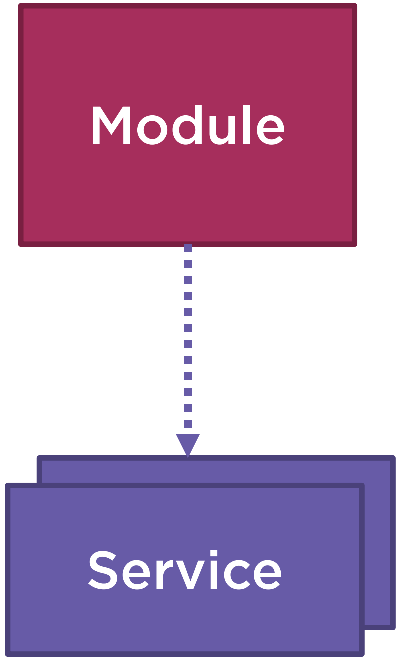
Only import what this module needs.

Imports Array Truth #3

Importing a module does NOT provide access to its imported modules



Providers Array



Module

Service

app.module.ts

```
…
providers: [ ProductDetailGuard ]
…
```

········· Imports

········· Exports

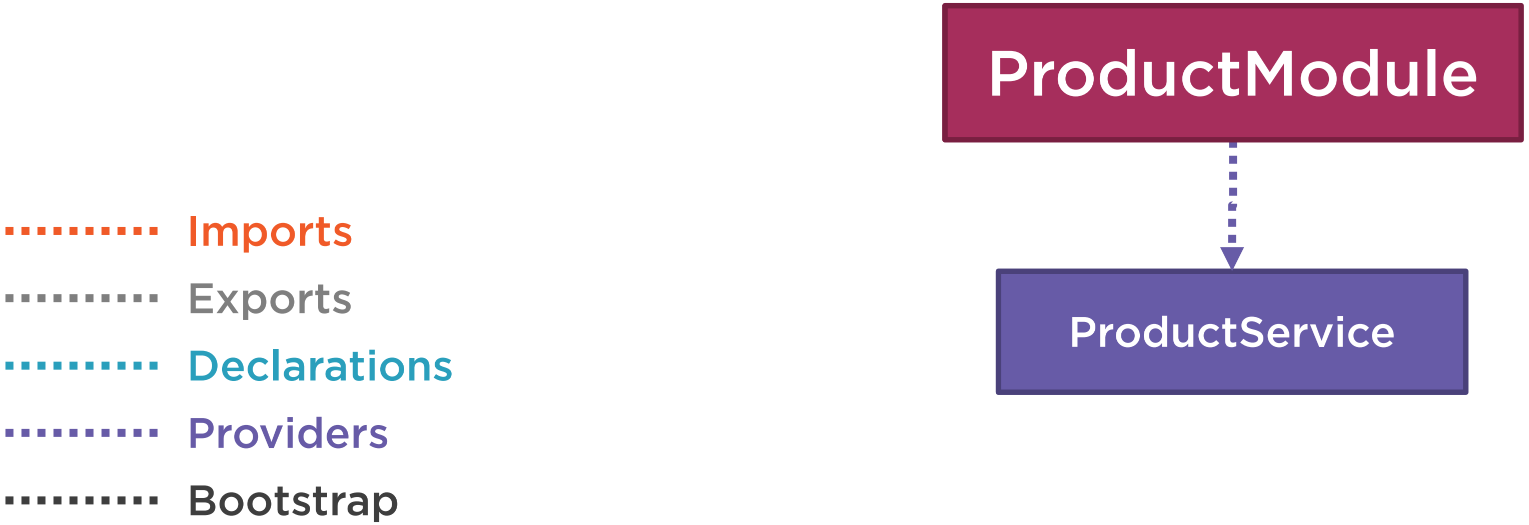
········· Declarations

········· Providers

········· Bootstrap

Providers Array Truth #1

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



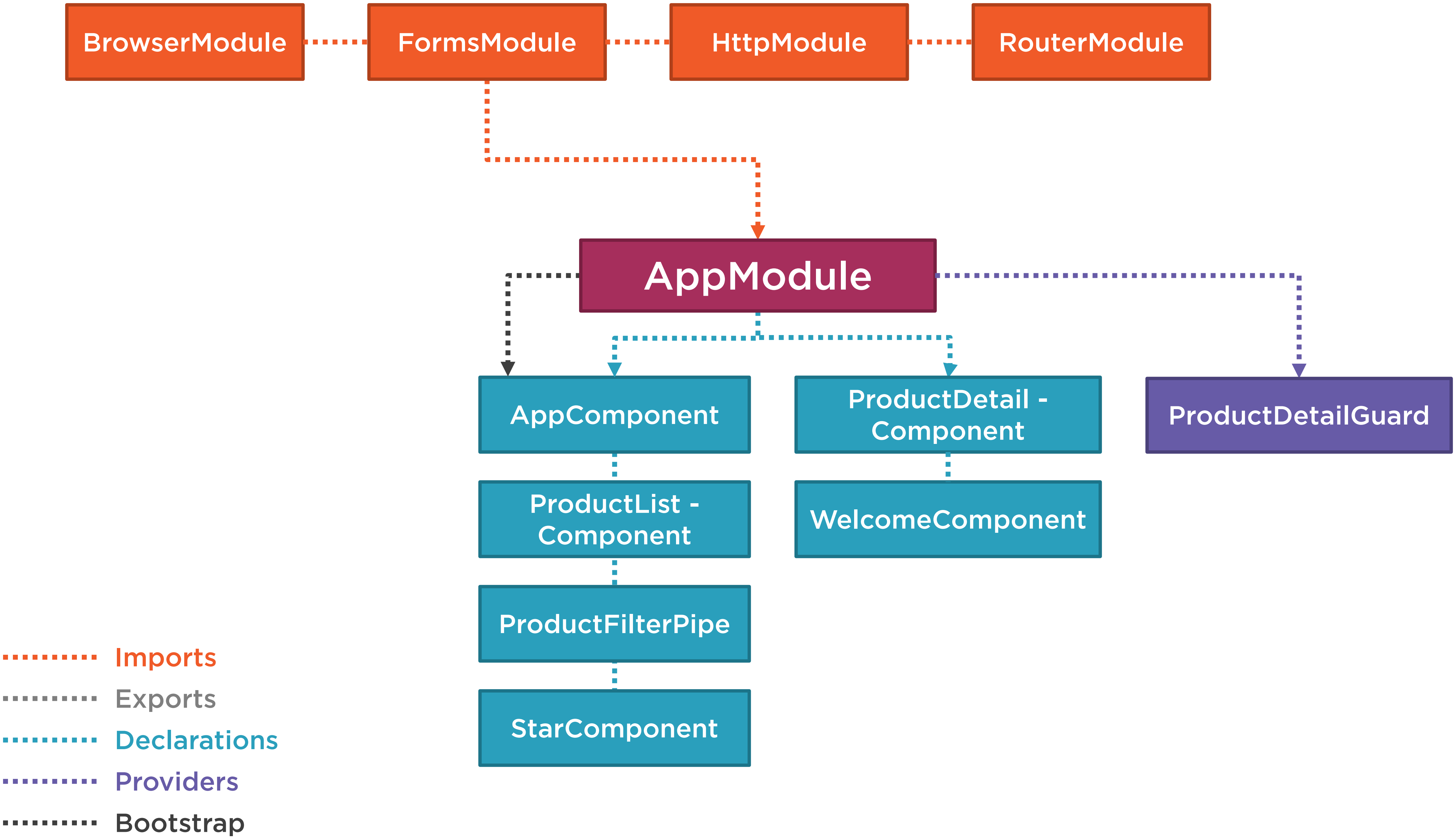
Providers Array Truth #2

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.



BrowserModule

FormsModule

HttpModule

RouterModule

AppModule

AppComponent

ProductList - Component

ProductFilterPipe

StarComponent

ProductDetail - Component

WelcomeComponent

ProductDetailGuard

┈┈┈ Imports

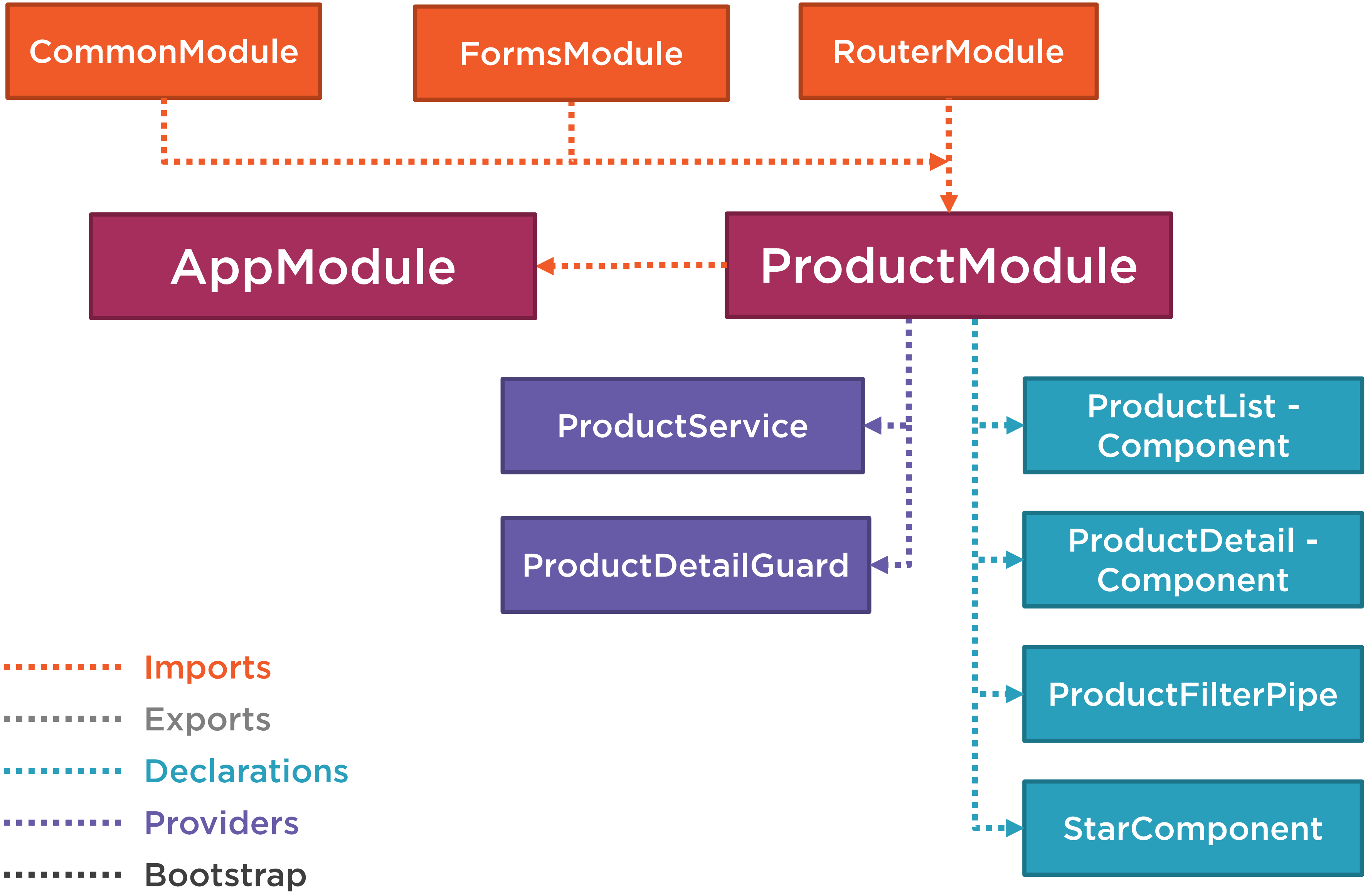
┈┈┈ Exports

┈┈┈ Declarations

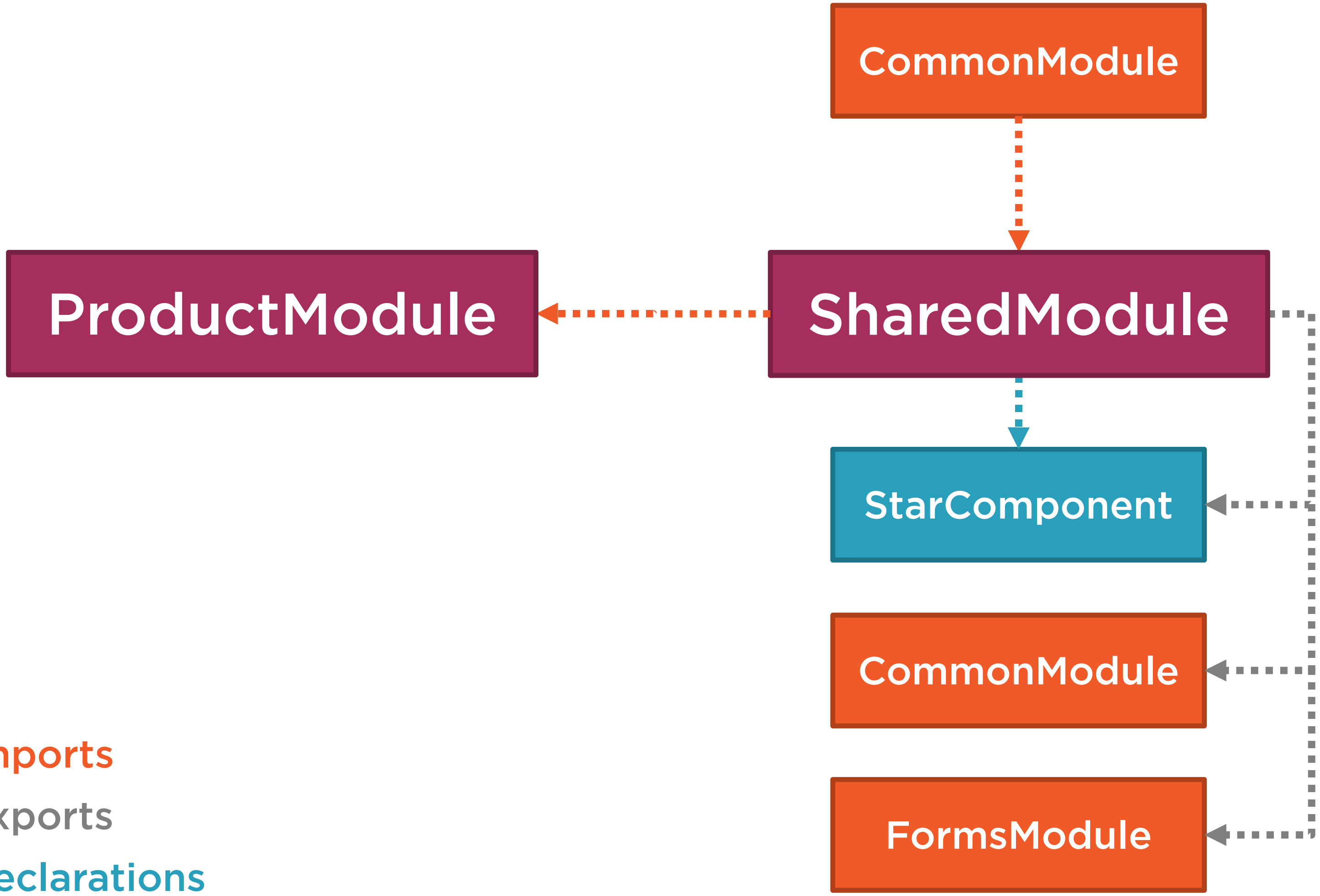
┈┈┈ Providers

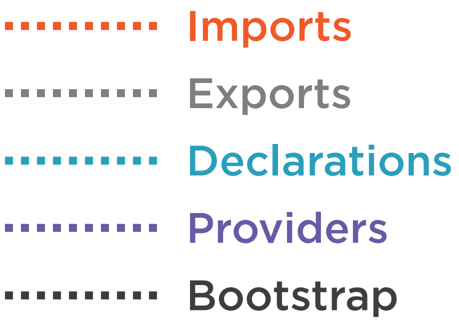
┈┈┈ Bootstrap

Defining a Feature Module

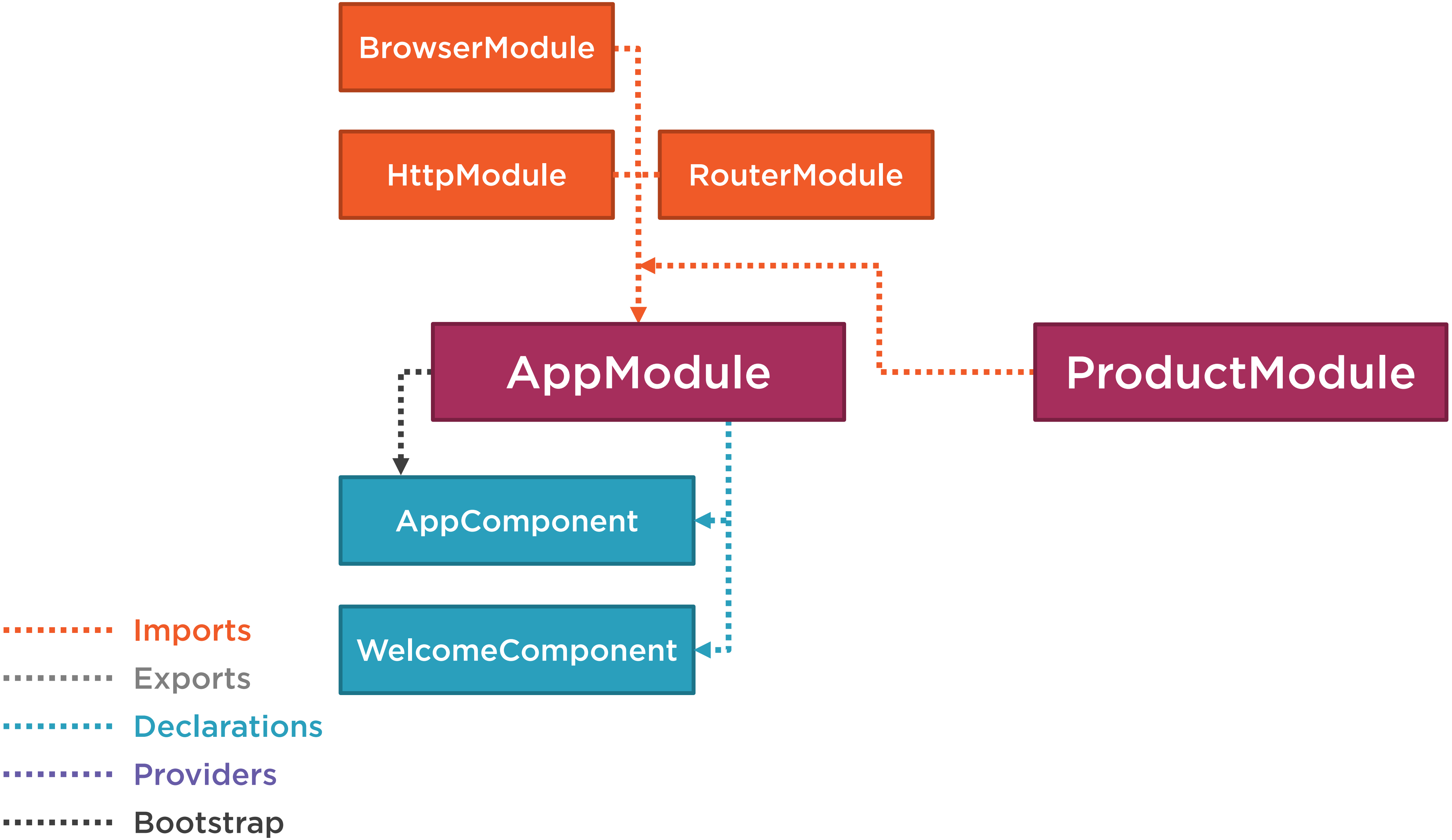


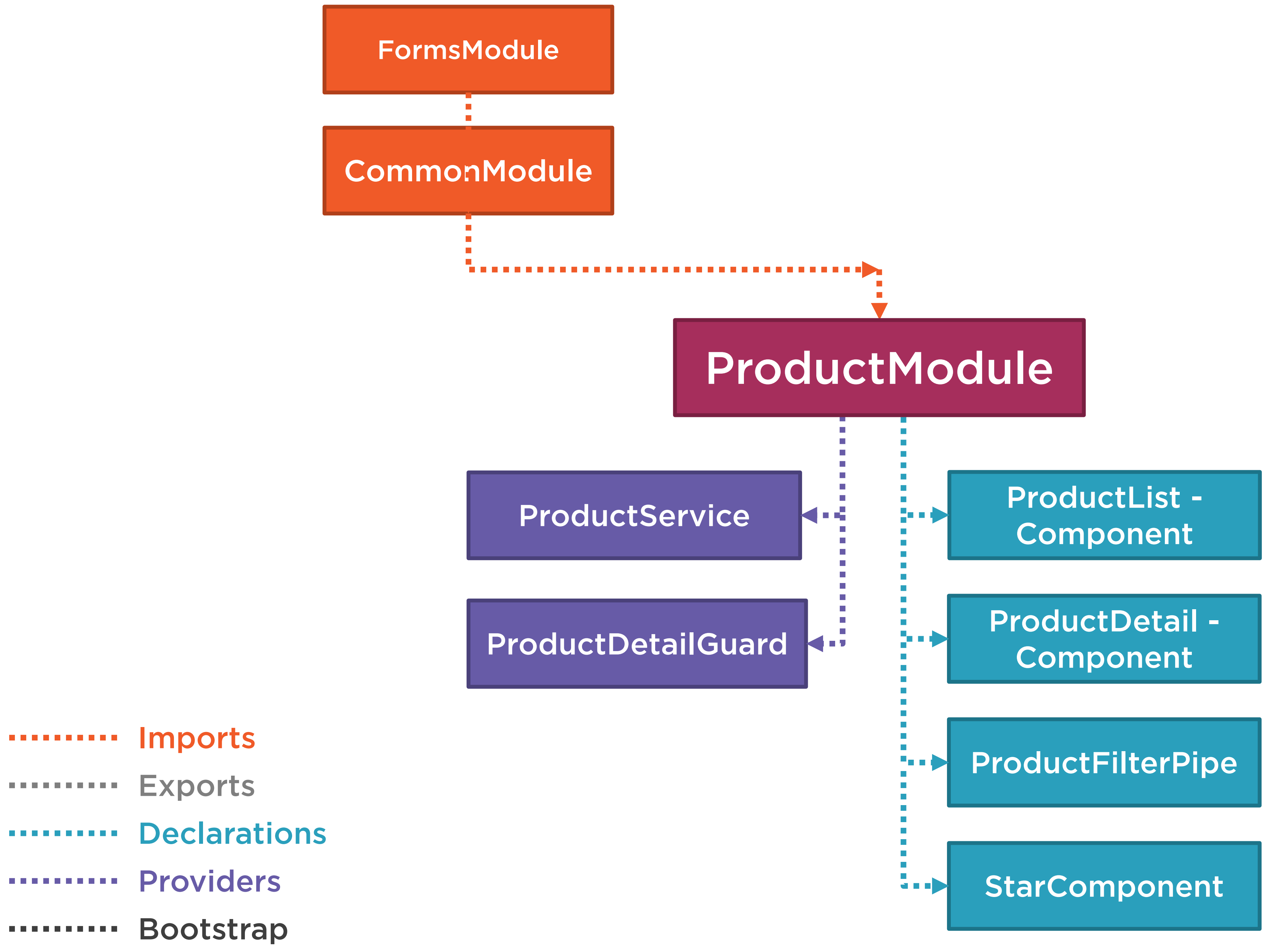
Defining a Shared Module





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 ProductRoutingModule { };
```

Using the Routing Module

product.module.ts

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

Angular Module Checklist: Module Structure

Root application module (AppModule)

Feature modules

Shared module (SharedModule)

Core module (CoreModule)

Routing modules

Angular Module Checklist: NgModule Metadata

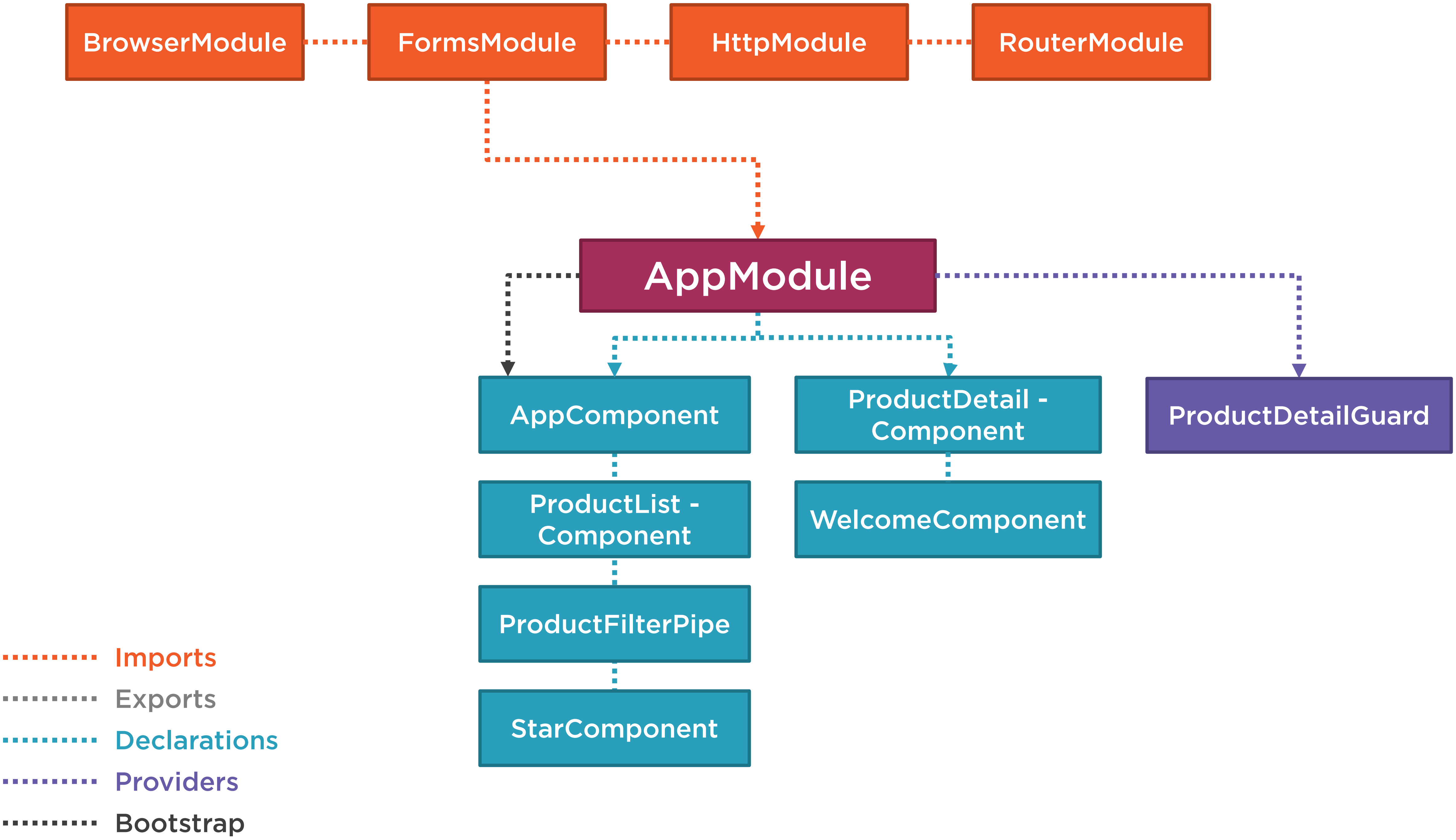
Bootstrap: Startup component(s)

Declarations: What belongs to this module

Exports: What an importing module can use

Imports: Supporting modules

Providers: Service providers



BrowserModule

FormsModule

HttpModule

RouterModule

AppModule

AppComponent

ProductList - Component

ProductFilterPipe

StarComponent

ProductDetail - Component

WelcomeComponent

ProductDetailGuard

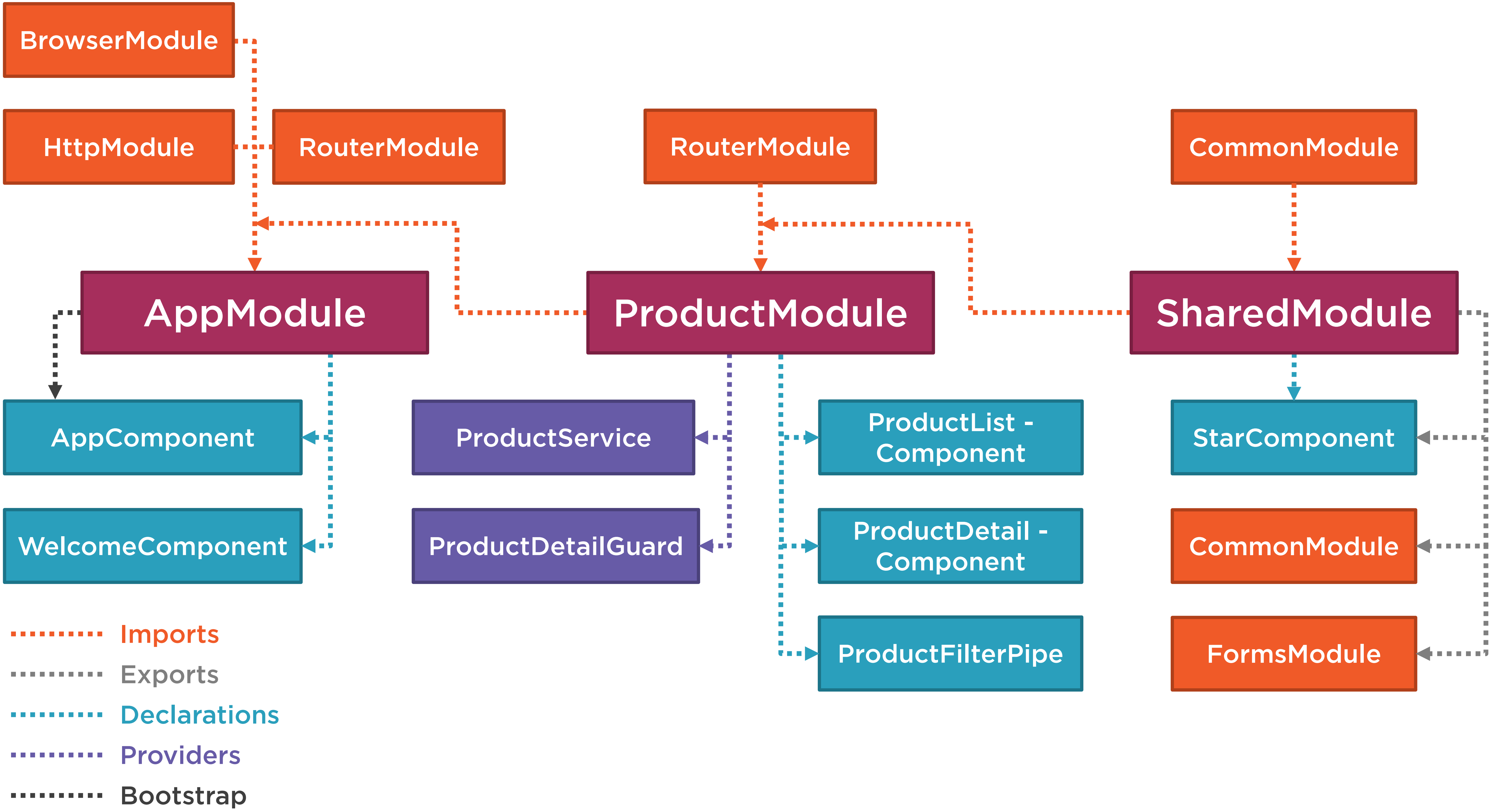
Imports

Exports

Declarations

Providers

Bootstrap


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