Angular

Introduction

Eng: Mostafa Saqly

Angular Is …

A JavaScript framework

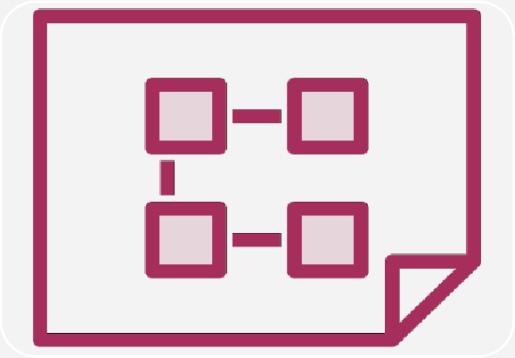
For building client-side applications

Using HTML, CSS and TypeScript

Why Angular?

Expressive HTML

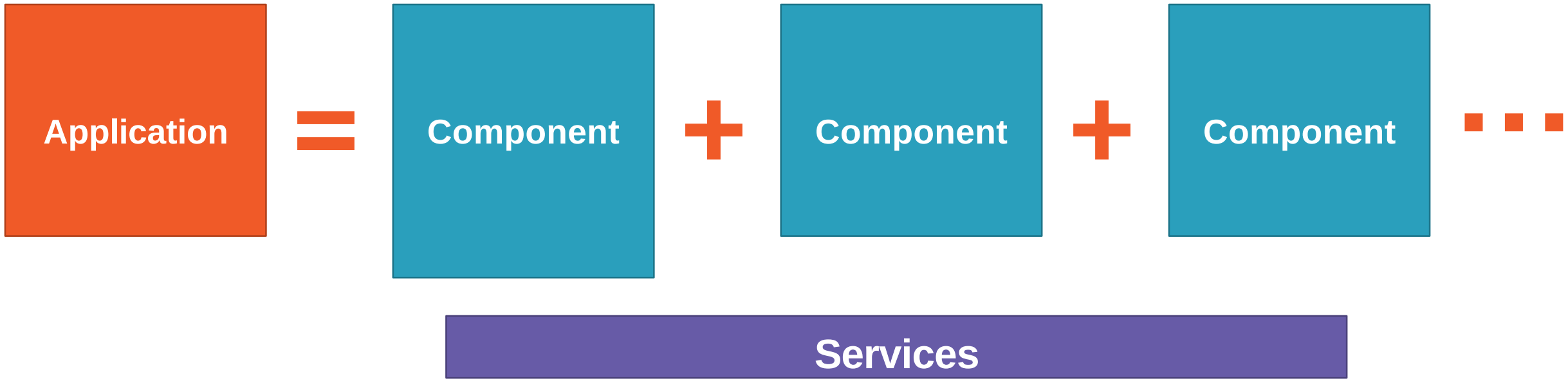
Powerful Data Binding



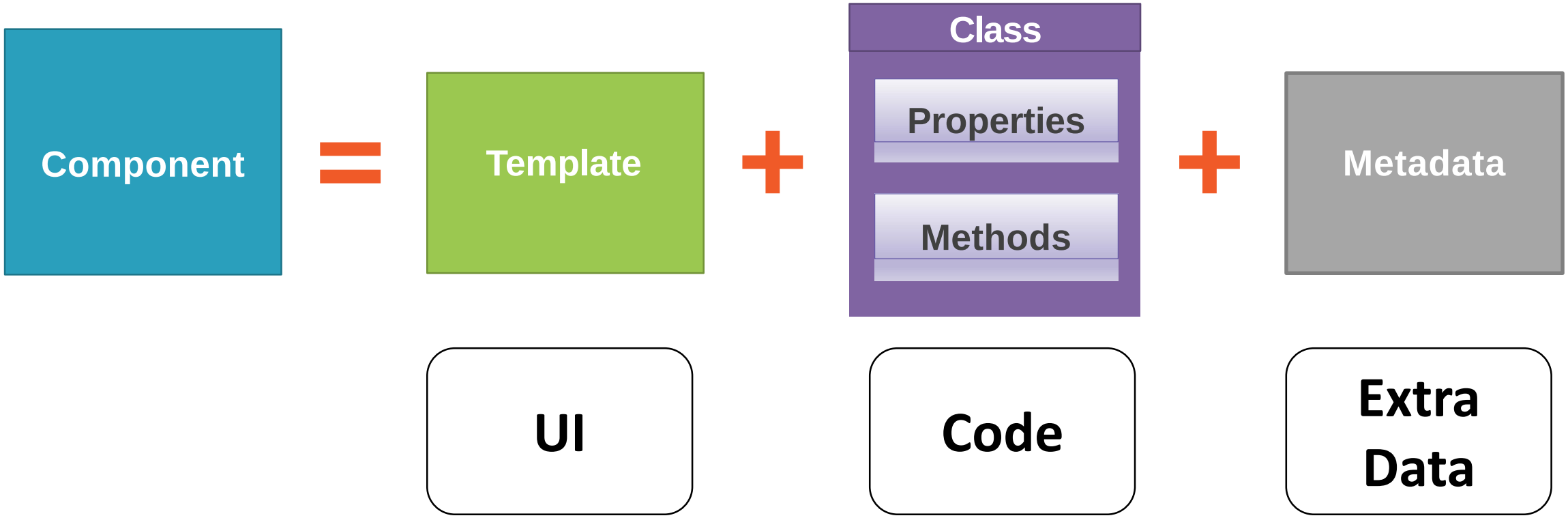
Modular By Design

Built-in Back-End Integration

Anatomy of an Angular Application



Component


Component
=
Template
+
Class
Properties
Methods
+
Metadata
UI
Code
Extra Data

TypeScript
is the programming language
we use
when building
Angular applications

TypeScript

- Open-source language by Microsoft
- Superset of JavaScript
- Transpilers to plain JavaScript
- Strongly typed
- Class-based object-orientation

package.json

dependencies

- Packages required for development and deployment

devDependencies

- Packages only required for development

```
12  "dependencies": {
13    "@angular/animations": "^14.0.0",
14    "@angular/common": "^14.0.0",
15    "@angular/compiler": "^14.0.0",
16    "@angular/core": "^14.0.0",
17    "@angular/forms": "^14.0.0",
18    "@angular/platform-browser": "^14.0.0",
19    "@angular/platform-browser-dynamic": "^14.0.0",
20    "@angular/router": "^14.0.0",
21    "rxjs": "~7.5.0",
22    "tslib": "^2.3.0",
23    "zone.js": "~0.11.4"
24  },
25  "devDependencies": {
26    "@angular-devkit/build-angular": "^14.0.3",
27    "@angular/cli": "~14.0.3",
28    "@angular/compiler-cli": "^14.0.0",
29    "@types/jasmine": "~4.0.0",
30    "jasmine-core": "~4.1.0",
31    "karma": "~6.3.0",
32    "karma-chrome-launcher": "~3.1.0",
33    "karma-coverage": "~2.2.0",
34    "karma-jasmine": "~5.0.0",
35    "karma-jasmine-html-reporter": "~1.7.0",
36    "typescript": "~4.7.2"
37  }
```

When Setting Up Existing Angular Code

#1

Navigate down to the project folder

The project folder contains the package.json **file**

#2

Run: npm install

To install the packages defined in the package.json file

#3

Run: npm start

To start the installed Angular application

Demo

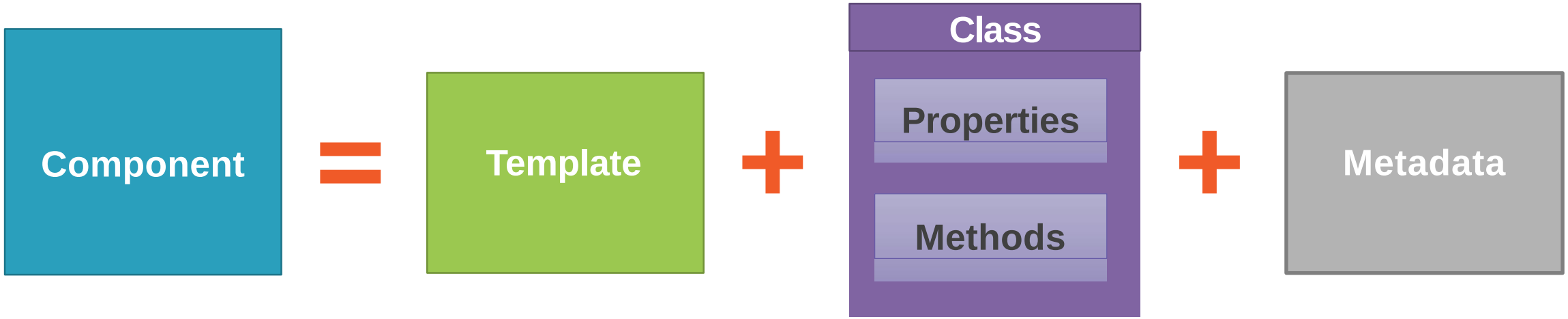
Create an Angular app using the Angular CLI

```
npm install –g @angular/cli
ng new project-name
npm install
ng serve -o
```

Module Overview

- What is a component?
- Creating the component class
- Defining the metadata with a decorator
- Importing what we need
- Bootstrapping our app component
- Something’s wrong!

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

Import

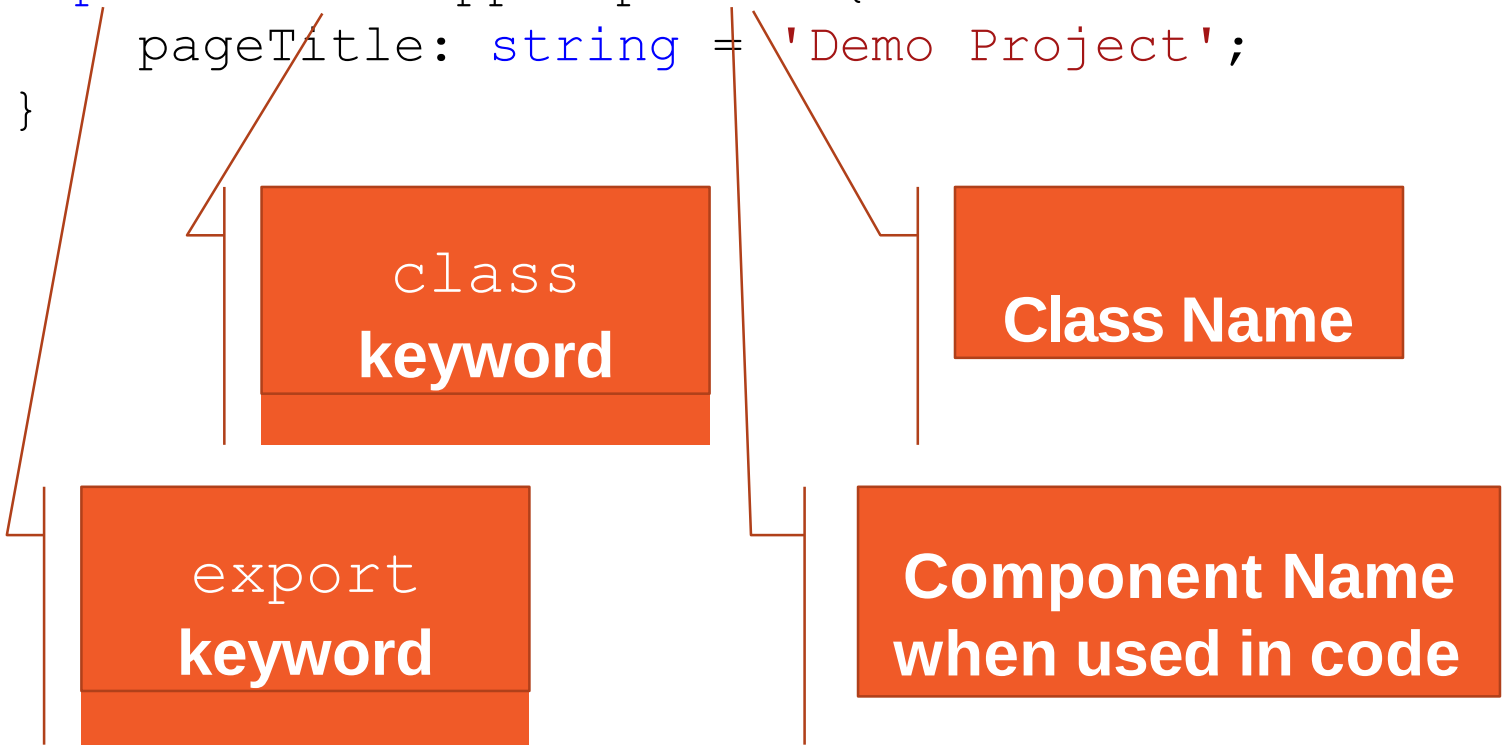
Metadata & Template

Class

Creating the Component Class

app.component.ts

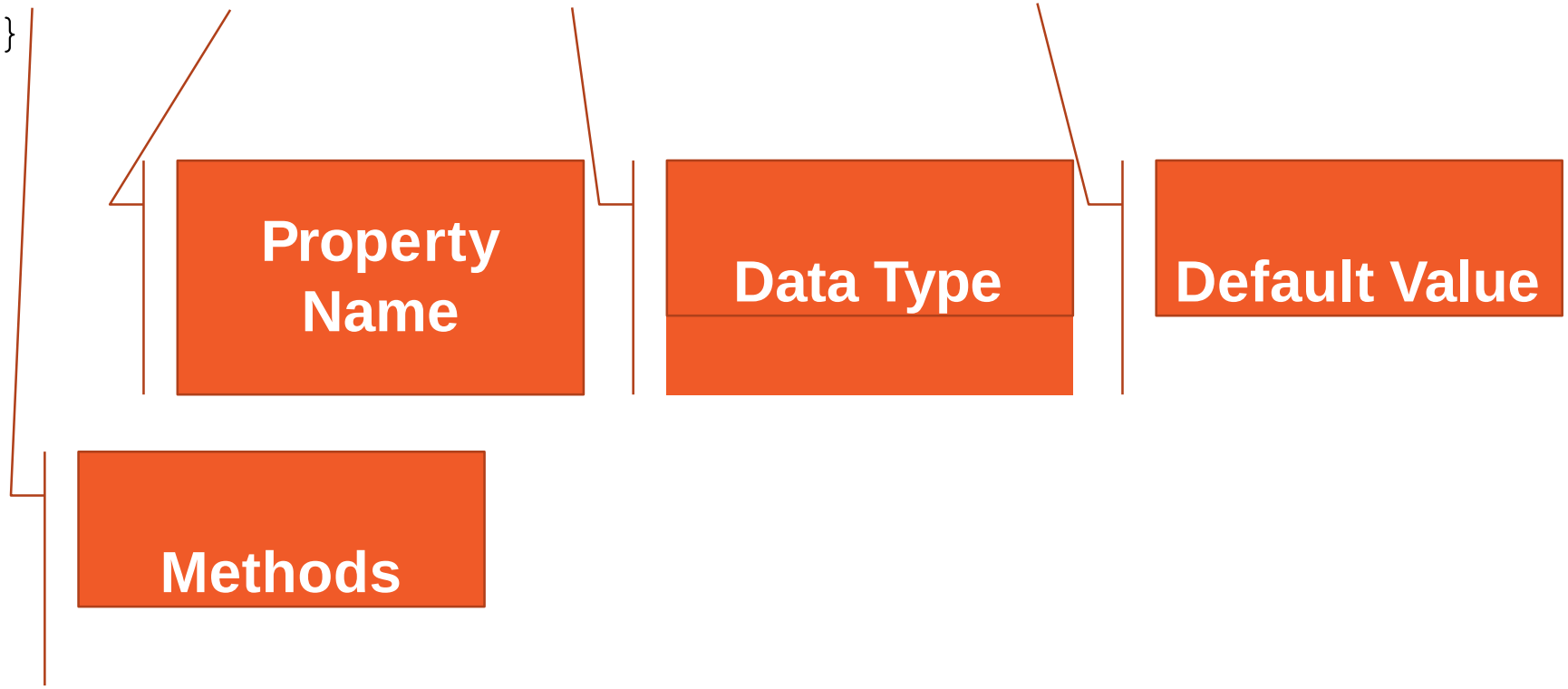
```
export class AppComponent {
    pageTitle: string = 'Demo Project';
}
```



Creating the Component Class

app.component.ts

```
export class AppComponent {
    pageTitle: string = ' Demo Project';
}
```



Defining the Metadata

app.component.ts

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

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-root',
    template: `
    <div><h1>{{pageTitle}}</h1>
        <div>My First Component</div>
    </div>
    `
})
export class AppComponent {
 pageTitle: string = ' Demo Project ';
}
```

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

Allows us to use exported members from:

- **Other files in our application**
- **Angular framework**
- **External JavaScript libraries**

Importing What We Need

app.component.ts

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

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

`import` **keyword**

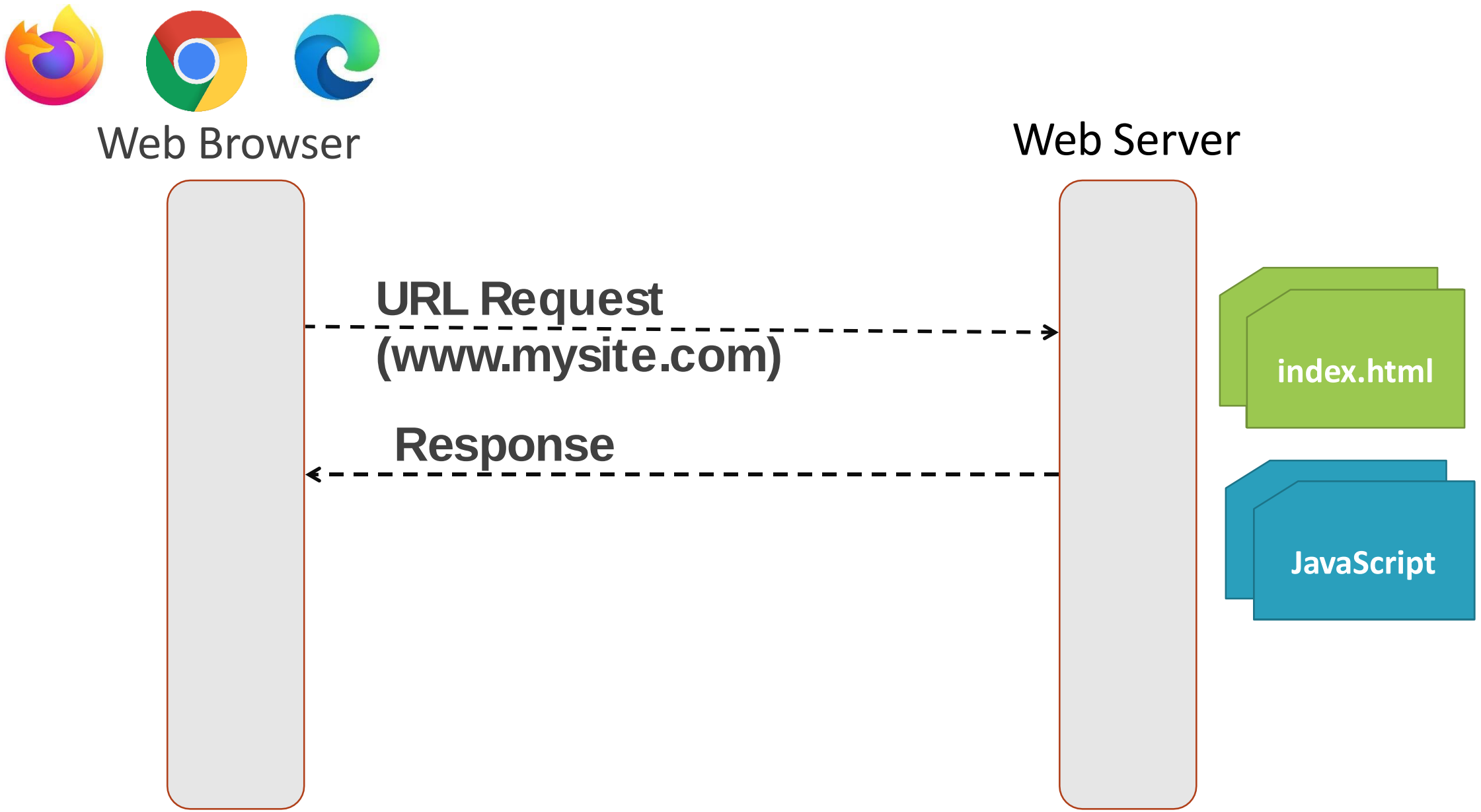
Angular library name

Member name

Completed Component

app.component.ts

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

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


Web Browser
Web Server
URL Request
(www.mysite.com)
Response
index.html
JavaScript

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)

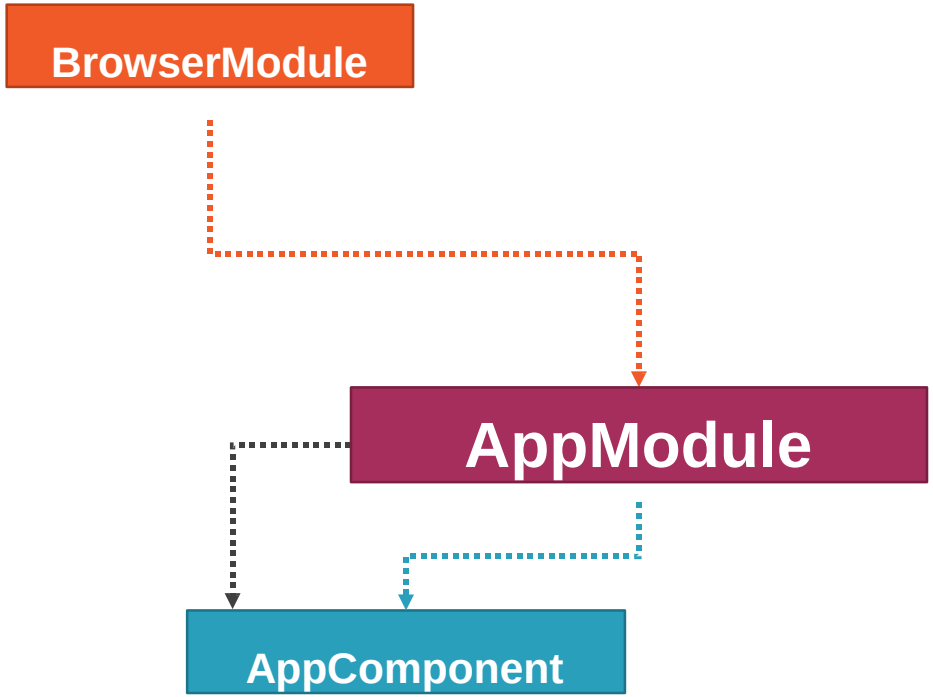
Hosting the Application

index.html

```
<body>
 <pm-root></pm-root>
</body>
```

app.component.ts

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

@Component({
    selector: 'pm-root',
    template: `
    <div><h1>{{pageTitle}}</h1>
        <div>My First Component</div>
    </div>
    `

})
export class AppComponent {
 pageTitle: string = 'Demo Project';
}
```



Organization

Boundaries

Template resolution environment

Imports

Exports

Declarations

Bootstrap

Defining the Angular Module

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 { }
```

Angular compiles our
HTML templates and
TypeScript components
to JavaScript

Start with Your Code Editor - Errors

Check for squiggly lines

```
TS app.component.ts ×
1  @Component({ })
2  e
3                                  ement';
4  }
5
   any
   Cannot find name 'Component'. ts(2304)
   Peek Problem (Alt+F8)   Quick Fix... (Ctrl+.)
```

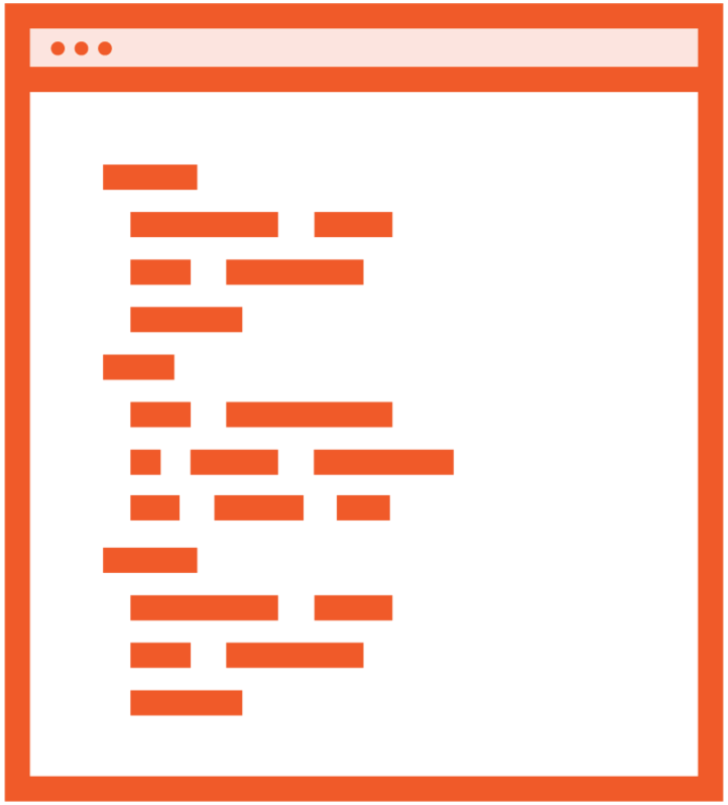
Open the terminal

```
PROBLEMS 1   TERMINAL   ···                1: node

Error: src/app/app.component.ts:1:2 - error TS2304: Cannot find name 'Component'.

1 @Component({ })
  ~~~~~~~~~
```

Stop (Ctrl+C) and Restart

```
PROBLEMS 1   OUTPUT   TERMINAL   ···        1: powershell

Terminate batch job (Y/N)? y
PS C:\Users\Deborah\Pluralsight\Angular Getting Started\APM> npm start
```

Recheck Your Code

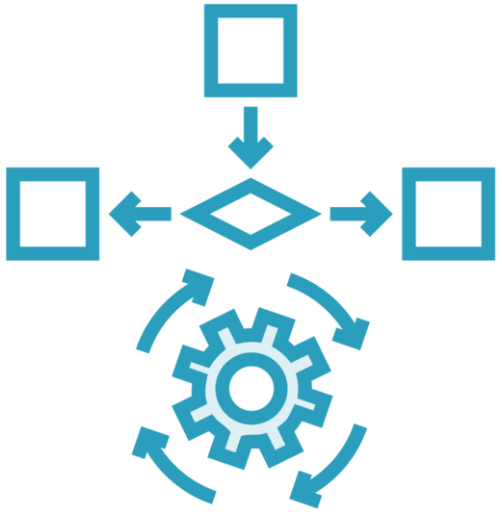


- **HTML**
  - **Close tags**
  - **Angular directives are case sensitive**
- **TypeScript**
  - **Close braces**
  - **TypeScript is case sensitive**

Templates, Interpolation, and Directives

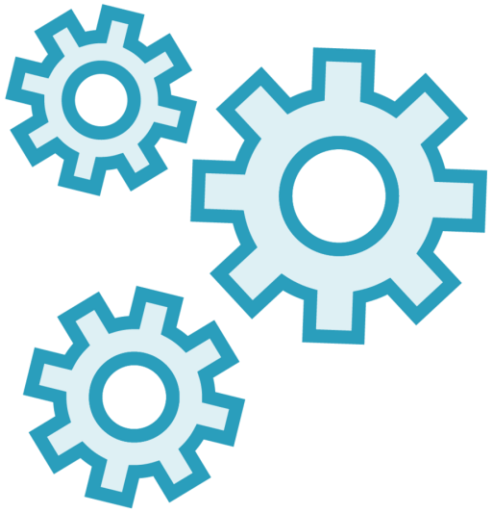
Power up HTML

Data binding



Angular directives

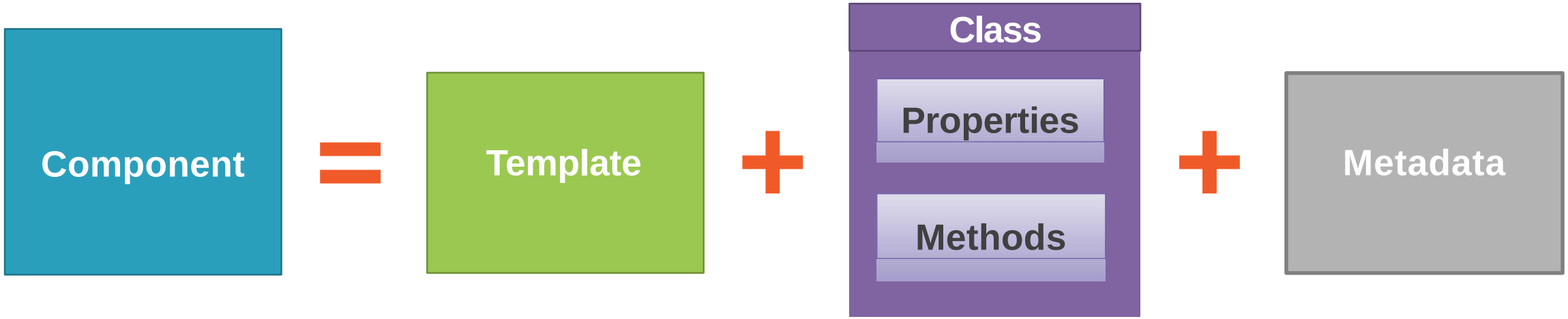
(Custom HTML syntax)



Angular components

(Custom Directives)

Component



Component

app.component.ts

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

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

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'
```

Demo

Install Bootstrap and font-awesome

```
npm install bootstrap font-awesome
```

Building New Component

Binding

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

Directive

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

- **Custom**
- **Built-In**

Angular Built-in Directives

Structural Directives

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

Data Binding & Pipes

Property Binding

Element Property

Template Expression

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

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

<input type='text' [disabled]='isDisabled'/>

<img src='http://myImages.org/{{product.imageUrl}}'>
```

Event Binding

Template

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

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

<button (click)='toggleImage()'>
```

Event

Method

Component

```
export class ListComponent {
 pageTitle: string = 'Product
 List'; products: any[] = […];

}
```

Two-way Binding

Template

```
<div class='col-md-2'>Filter by:</div>
<div class='col-md-4'>
   <input type='text'
          [(ngModel)]='listFilter'/>
</div>
```

[()]
Banana in a Box

Component

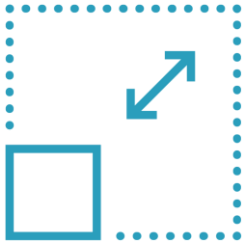
```
export class ListComponent {
 listFilter: string = 'cart';
}
```

Visual studio

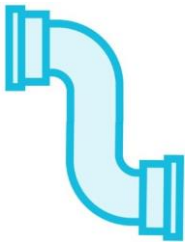
Ngmodel => formsModule

Aap.module.ts

Transforming Data with Pipes



Transform bound properties before display



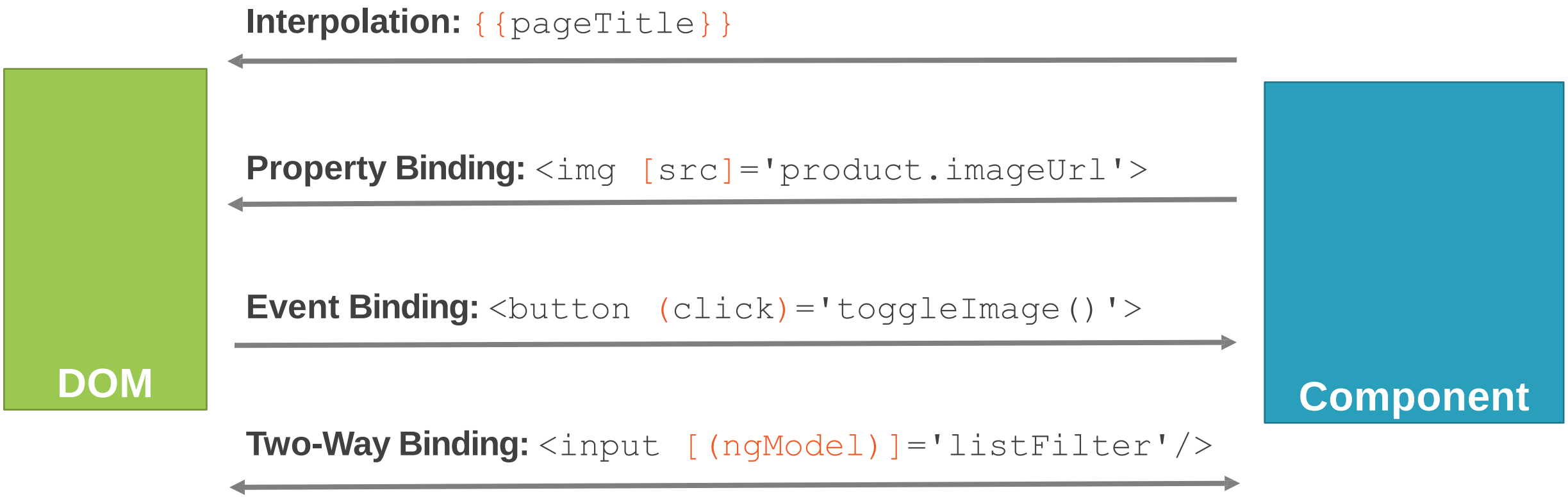
Built-in pipes: date, number, decimal, percent, currency, json, etc.

Custom pipes

Pipe Examples

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

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

{{ product.price | currency | lowercase }}

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

Data Binding



More on Components

Introduction

Improving Our Components

Strong typing & interfaces

Encapsulating styles

Lifecycle hooks

Custom pipes

Nested components

An interface is a specification identifying a related set of properties and methods.

Handling Unique Component Styles

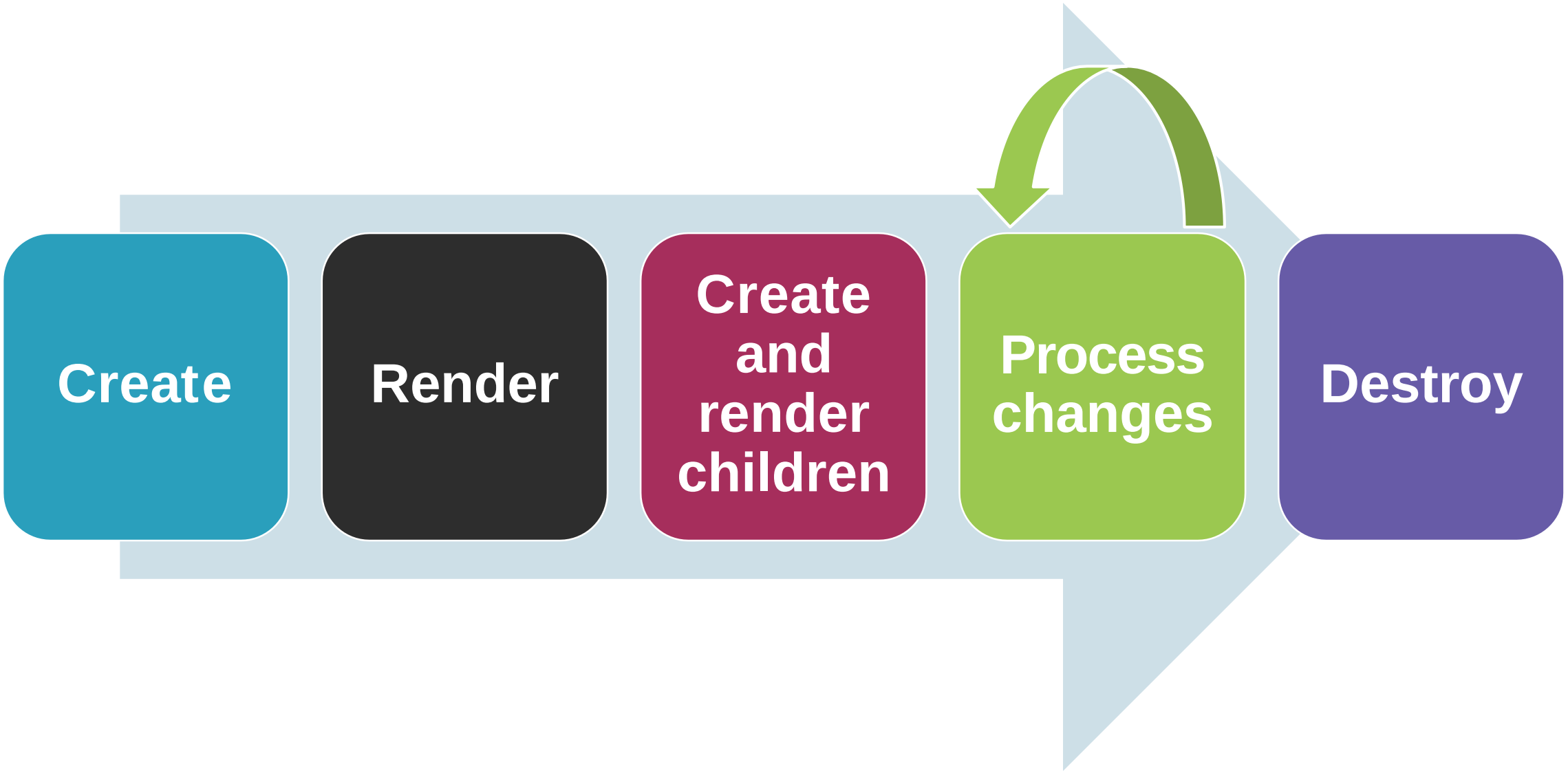
Templates sometimes require unique styles

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!

Component Lifecycle



A lifecycle hook is an interface we implement to write code when a component lifecycle event occurs.

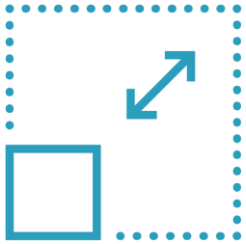
Component Lifecycle Hooks

`OnInit`: **Perform component initialization, retrieve data**

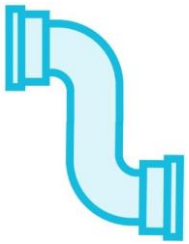
`OnChanges`: **Perform action after change to input properties**

`OnDestroy`: **Perform cleanup**

Transforming Data with Pipes



Transform bound properties before display



Built-in pipes: date, number, decimal, percent, currency, json, etc.

Custom pipes

Getters and Setters

```
private _amount: number = 0;
```

```
get amount(): number {
    // process the amount
    // return amount from private storage
    return this._amount;
}
set amount(value: number) {
    // process the amount
    // retain amount in private storage
    this._amount = value;
}
```

```
this.amount = 200;
```

```
console.log(this.amount);
```

An arrow function is compact syntax for defining a function.