Angular 2 Essentials: Modules, Components, Templates, and Metadata

John Papa
PRINCIPAL ARCHITECT

@john_papa www.johnpapa.net

Overview

ES Modules

Angular Modules

Components

Templates

Metadata

ES Modules

ES Modules are often referred to simply as “Modules”

Modules

We assemble our application from modules.

A module exports an asset such as a Service, Component, or a shared value vehicle.service.ts

```
export interface Vehicle {
  id: number;
  name: string;
}

export class VehicleService {
  //...
}
```

Exporting modules

Assets can be exported using the export keyword vehicle.component.ts

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

import { Vehicle, VehicleService } from './vehicle.service';
```

Importing modules

Modules and their contents can be imported using the import keyword

We import the Vehicle and VehicleService using destructuring

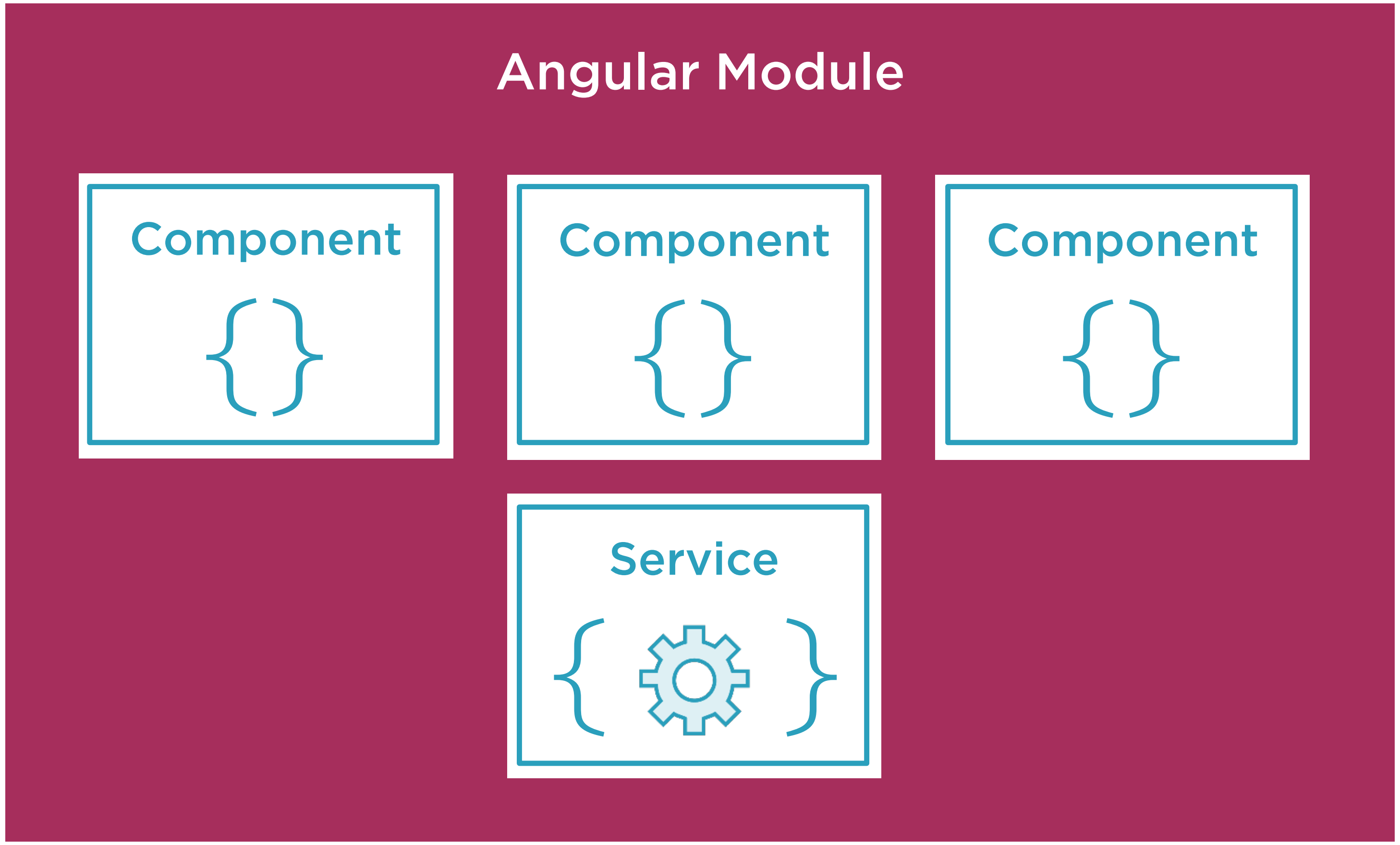
Angular Modules

aka NgModules

Angular Modules

We use NgModule to organize our application into cohesive blocks of related functionality

Angular Modules Organize Functionality



Separate Features into Angular Modules

An Angular Module is a class decorated by @NgModule

Roles of Angular Modules

Import other Angular Modules

Identify Components, Pipes, and Directives

Export it’s Features

Provide services to injectors

Can be eagerly or lazily loaded

The Root Angular Module

```
@NgModule({
  imports: [
    BrowserModule,
    FormsModule
  ],
  declarations: [
    VehiclesComponent
  ],
  providers: [
    VehicleService
  ],
  bootstrap: [VehiclesComponent],
})
export class AppModule { }
```

Import modules we depend on

Declare components, directives, pipes

Provide services to app root injector

Bootstrap a component

Class to define the NgModule

Every app begins with one Angular Module

Components

Angular 2 Components

A Component contains application logic that controls a region of the user interface that we call a view.

Anatomy of a Component

Imports (use other modules)

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

import { Vehicle } from './vehicle.service';
```

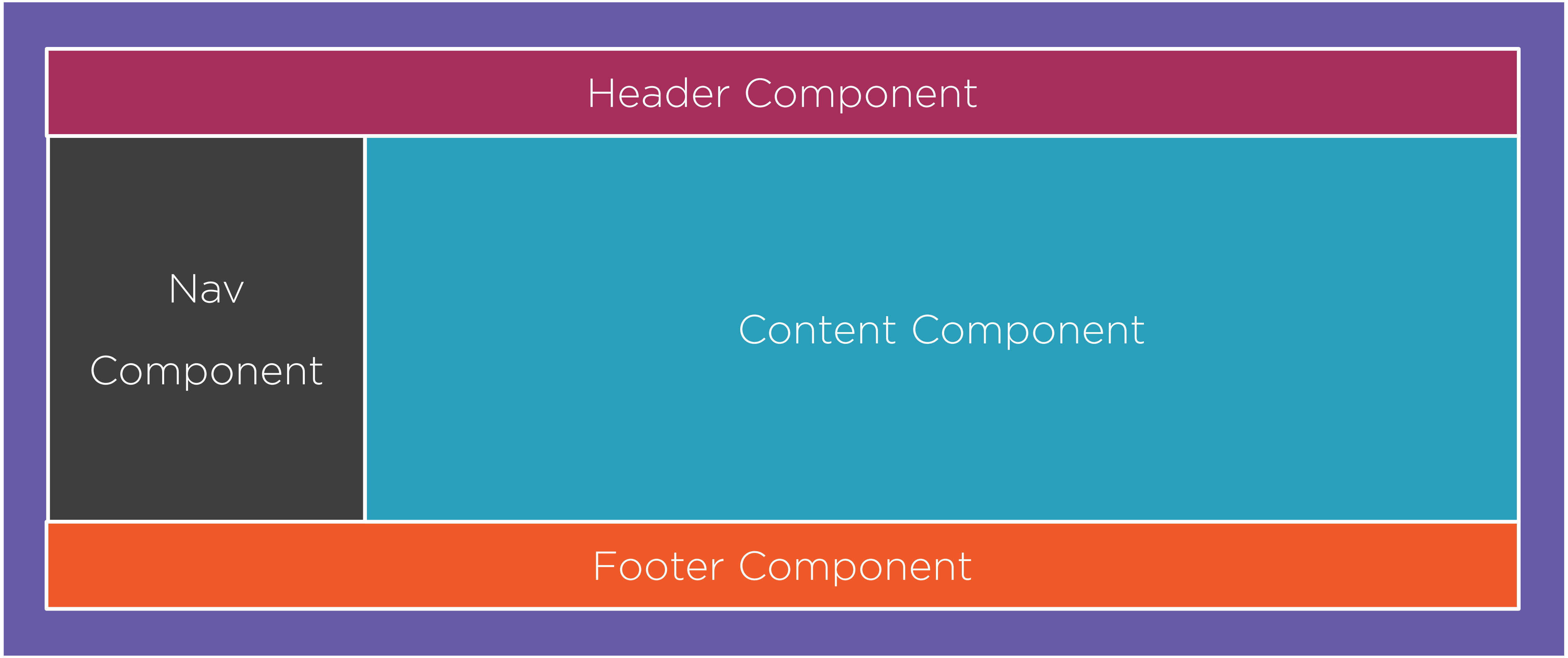
Metadata (describe the component)

```
@Component({
  moduleId: module.id,
  selector: 'story-vehicles',
  templateUrl: 'vehicles.component.html',
})
```

Class (define the component)

```
export class VehicleListComponent {
  vehicles: Vehicle[];
}
```

Assembling Our App from Components



main.ts

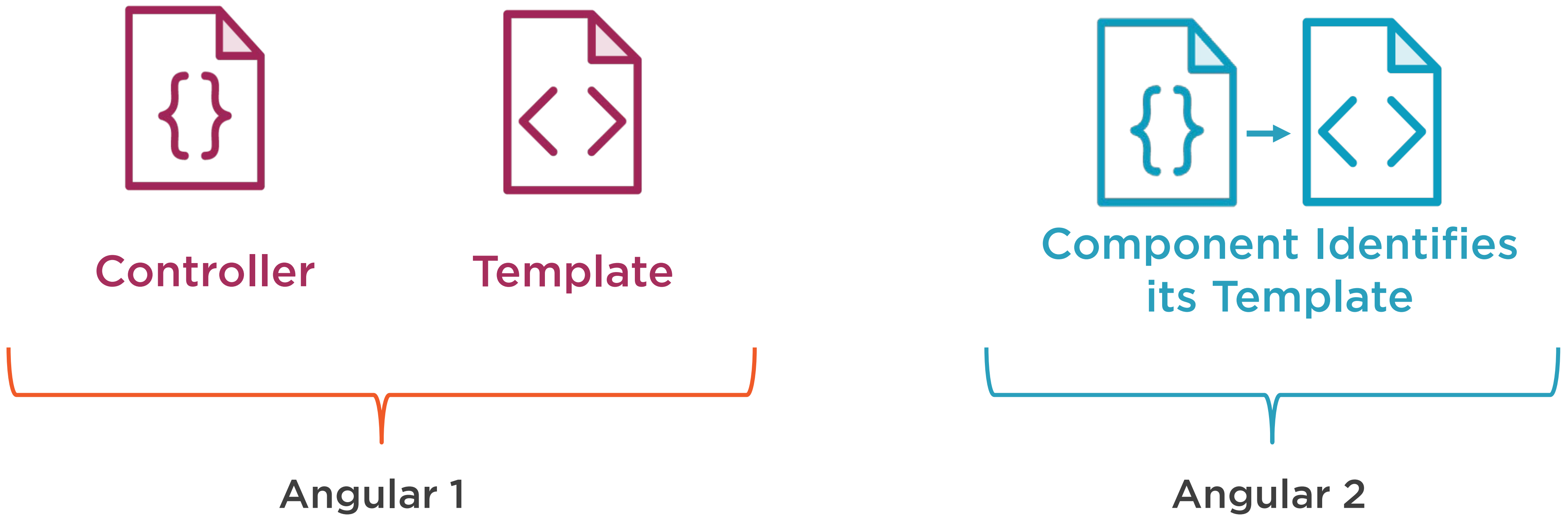
```typescript
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';

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

Bootstrapping a Component

Entry point for the app

This is where we start

Comparing Angular 1 to Angular 2



character.component.ts

```
@Component({
  moduleId: module.id,
  selector: 'story-character',
  templateUrl: 'character.component.html'
})
export class CharacterComponent {
  name = 'Han Solo';
}
```

Component has logic

character.component.html

```
<h3>My name is {{name}}</h3>
```

What is rendered

index.html

```
<story-character>Loading Demo ...</story-character>
```

Where the component is placed

character.component.ts

Component has logic

```
@Component({
  moduleId: module.id,
  selector: 'story-character',
  templateUrl: 'character.component.html'
})
export class CharacterComponent {
  name = 'Han Solo';
}
```

character.component.html

What is rendered

```
<h3>My name is {{name}}</h3>
```

index.html

Where the component is placed

```
<story-character>Loading Demo ...</story-character>
```

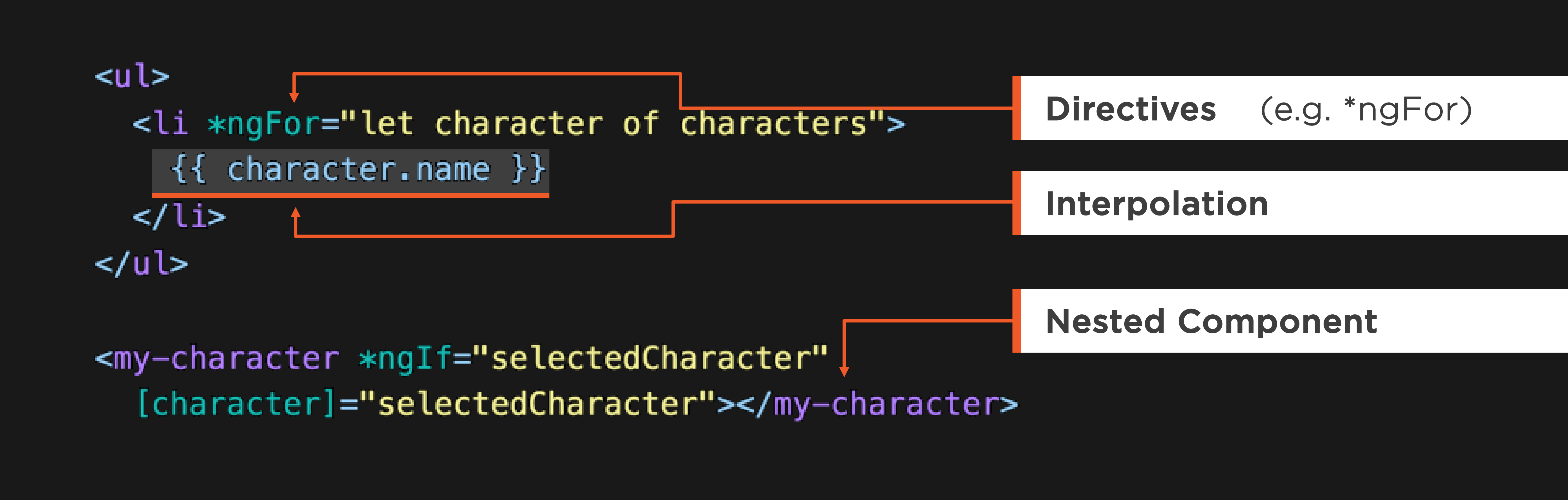
Components Demo

Components

Templates

Templates are the View

Templates are mostly HTML, with a little help from Angular. They tell Angular how to render the Component



```
<ul>
  <li *ngFor="let character of characters">
   {{ character.name }}
  </li>
</ul>

<my-character *ngIf="selectedCharacter"
  [character]="selectedCharacter"></my-character>
```

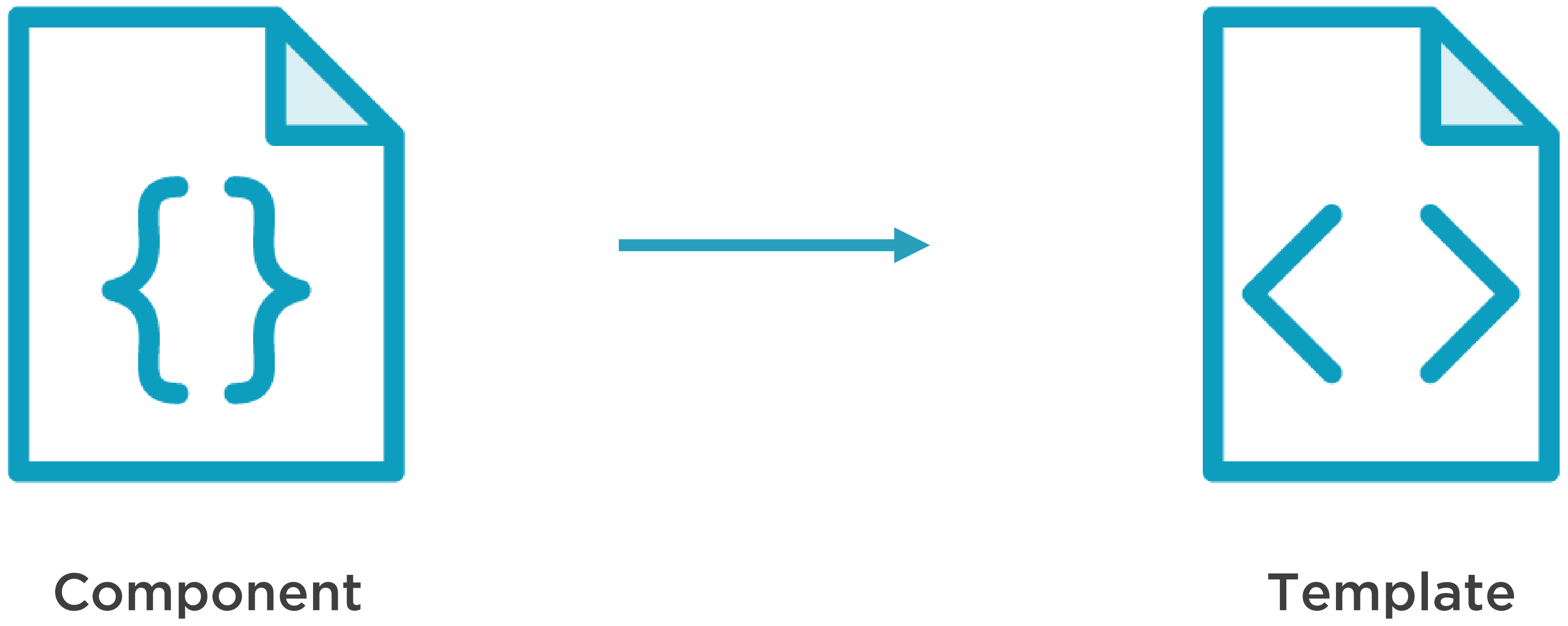
Templates

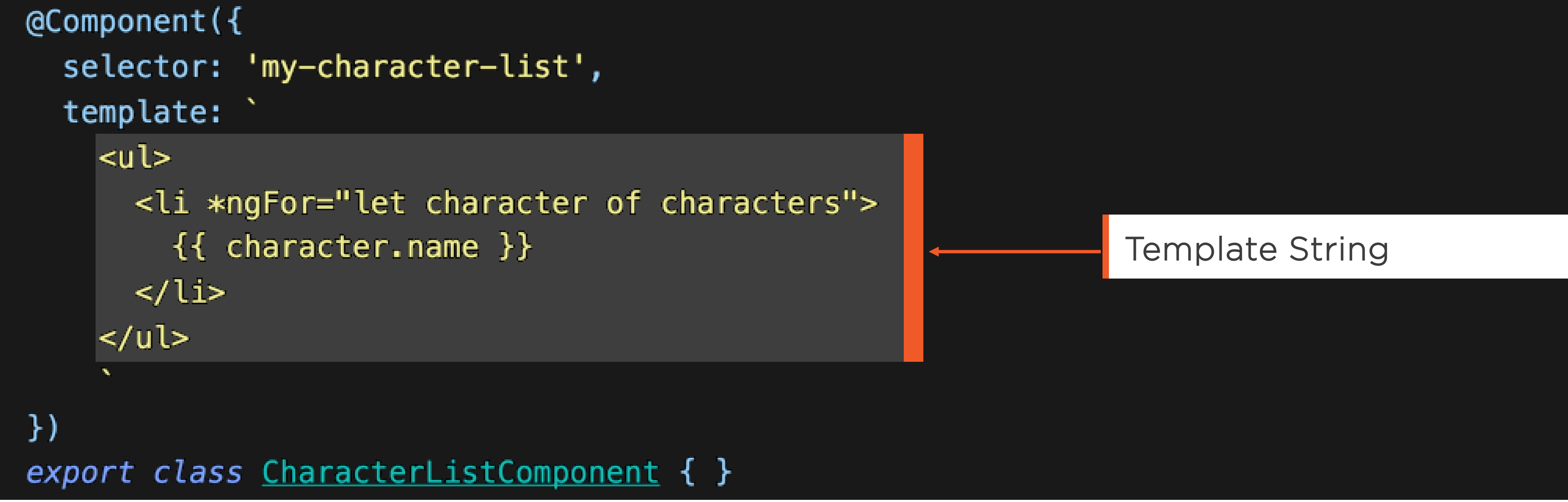
HTML

Directives, as needed

Template Binding Syntax

Connecting the Component to its Template



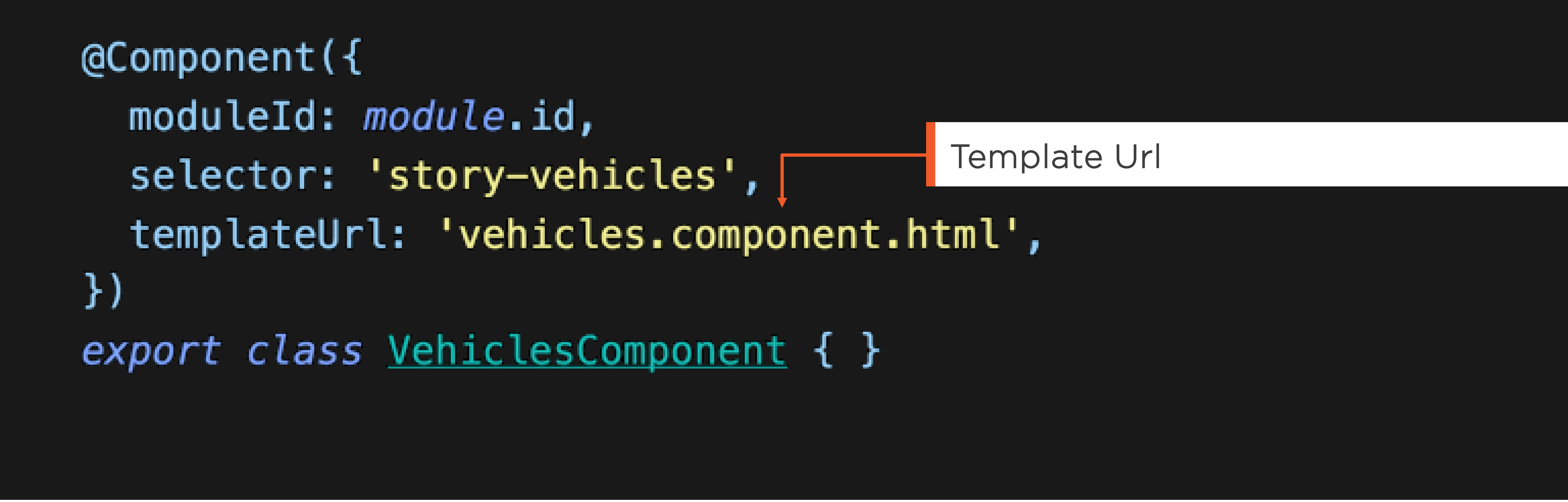


```
@Component({
  selector: 'my-character-list',
  template: `
    <ul>
      <li *ngFor="let character of characters">
        {{ character.name }}
      </li>
    </ul>
    `
})
export class CharacterListComponent { }
```

Inline Templates

template defines an embedded template string

Use back-ticks for multi-line strings



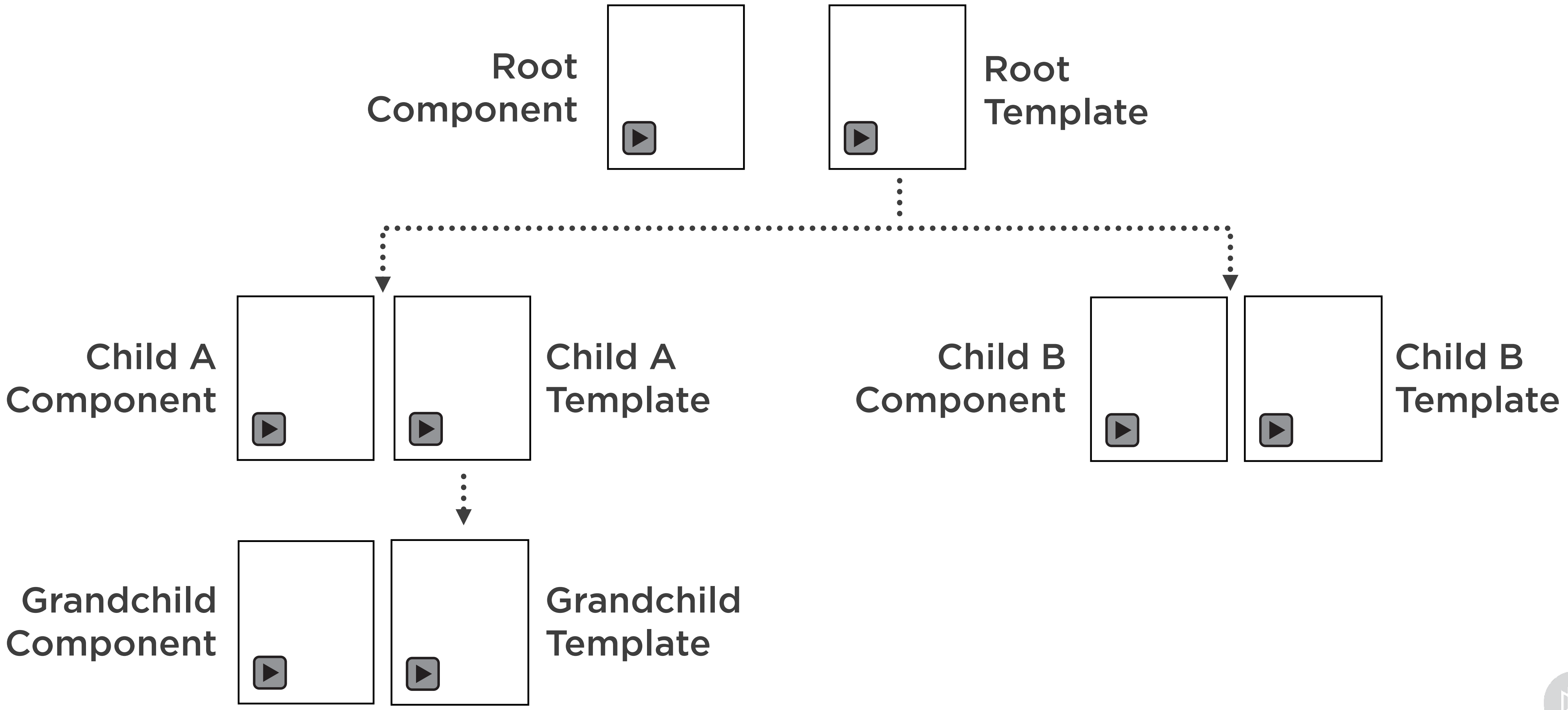
```
@Component({
  moduleId: module.id,
  selector: 'story-vehicles',
  templateUrl: 'vehicles.component.html',
})
export class VehiclesComponent { }
```

Linked Templates

templateUrl links the Component to its Template

Components have templates, which may use other components

Templates Contain Other Components



Demo

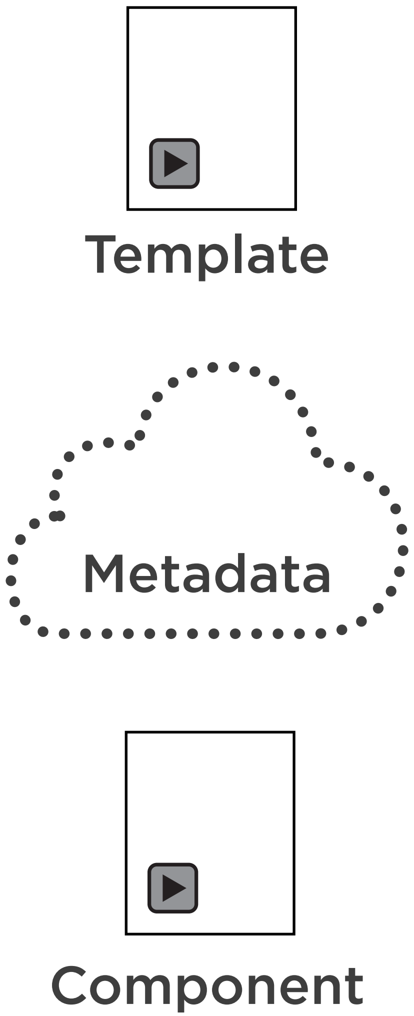
Templates

Metadata

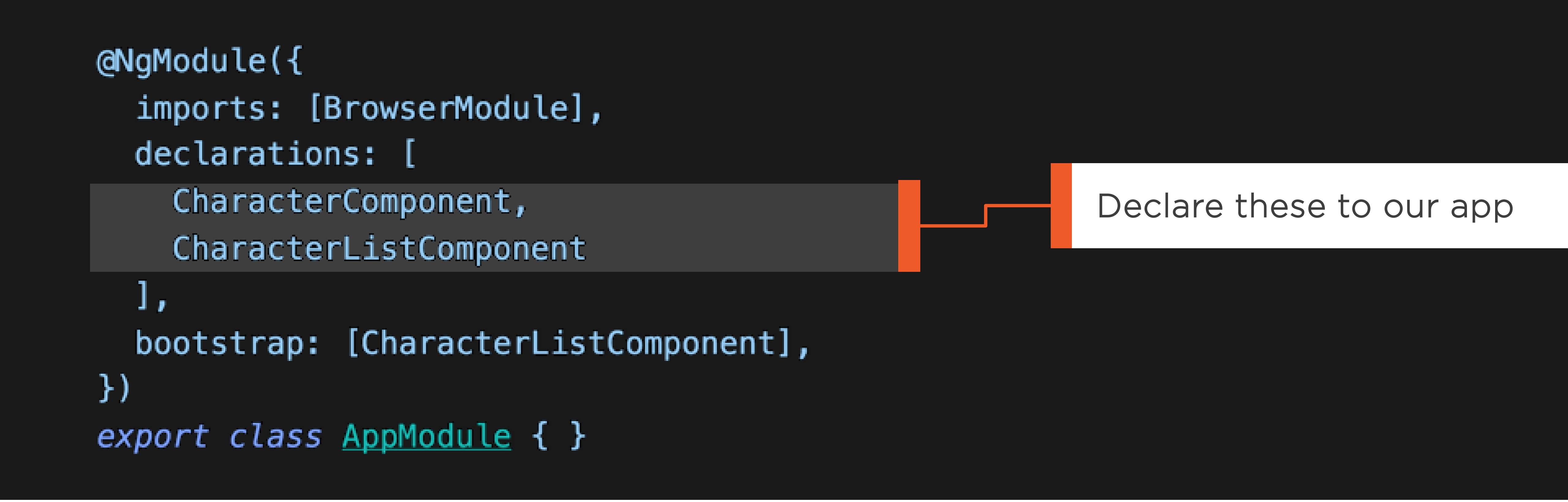
Metadata

We use Metadata to tell Angular about the objects we build.

Metadata Links the Template to the Component



We declare our components, directives and pipes in an Angular Module



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

Declaring Components

BrowserModule includes CommonModule

Built-in directives like *ngFor and ngClass are in CommonModule

We tell Angular what <my-character-list> and <my-character> are

Examining a Component and its Metadata

Decorators

The @ is a decorator that provides metadata describing the Component

Component

Component definition class. Controls a patch of screen real estate that we call a View

```
@Component({
  moduleId: module.id,
  selector: 'story-characters',
  templateUrl: 'characters.component.html',
  styleUrls: ['characters.component.css'],
  providers: [CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Character[];
  selectedCharacter: Character;

  constructor(private characterService: CharacterService) { }

  ngOnInit() {
    this.characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```

Template and Styles

Tells the Component where to find them.

```
@Component({
  moduleId: module.id,
  selector: 'story-characters',
  templateUrl: 'characters.component.html',
  styleUrls: ['characters.component.css'],
  providers: [CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Character[];
  selectedCharacter: Character;

  constructor(private characterService: CharacterService) { }

  ngOnInit() {
    this.characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```

Providers

These services will be registered with this component's injector. Only do this once.

Generally, prefer registering providers in angular modules to registering in components.

```
@Component({
  moduleId: module.id,
  selector: 'story-characters',
  templateUrl: 'characters.component.html',
  styleUrls: ['characters.component.css'],
  providers: [CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Character[];
  selectedCharacter: Character;

  constructor(private characterService: CharacterService) { }

  ngOnInit() {
    this.characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```

Injection

Inject a Service into another object.

```
@Component({
  moduleId: module.id,
  selector: 'story-characters',
  templateUrl: 'characters.component.html',
  styleUrls: ['characters.component.css'],
  providers: [CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Character[];
  selectedCharacter: Character;

  constructor(private characterService: CharacterService) { }

  ngOnInit() {
    this.characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```

Output

Component can communicate to anyone hosting it

Emit Events

Component emits events via output

```
@Component({
  moduleId: module.id,
  selector: 'story-characters',
  templateUrl: 'characters.component.html',
  styleUrls: ['characters.component.css'],
  providers: [CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Character[];
  selectedCharacter: Character;

  constructor(private characterService: CharacterService) { }

  ngOnInit() {
    this.characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```

Input

Pass values into the Component

```
@Component({
  moduleId: module.id,
  selector: 'story-characters',
  templateUrl: 'characters.component.html',
  styleUrls: ['characters.component.css'],
  providers: [CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Character[];
  selectedCharacter: Character;

  constructor(private characterService: CharacterService) { }

  ngOnInit() {
    this.characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```

Properties

Component exposes properties that can be bound to its Template

```
@Component({
  moduleId: module.id,
  selector: 'story-characters',
  templateUrl: 'characters.component.html',
  styleUrls: ['characters.component.css'],
  providers: [CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Character[];
  selectedCharacter: Character;

  constructor(private characterService: CharacterService) { }

  ngOnInit() {
    this.characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```

Actions

Functions can be exposed, bound and called by the Template to handle events

```
@Component({
  moduleId: module.id,
  selector: 'story-characters',
  templateUrl: 'characters.component.html',
  styleUrls: ['characters.component.css'],
  providers: [CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Character[];
  selectedCharacter: Character;

  constructor(private characterService: CharacterService) { }

  ngOnInit() {
    this.characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```

Input and Output

Components allow input properties to flow in, while output events allow a child Component to communicate with a parent Component.

Output

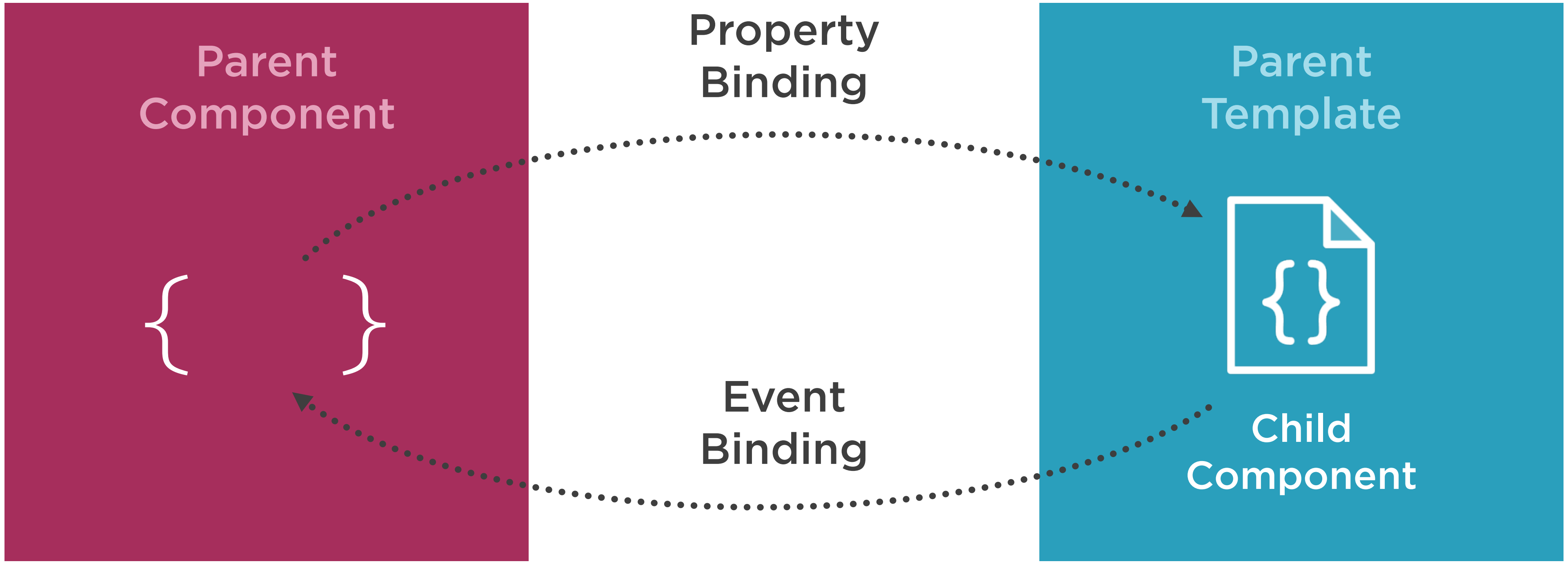
```
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Character[];
  selectedCharacter: Character;

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
```

Output property

Emit the output

```
<div>
  <h1>Storyline Tracker</h1>
  <h3>Component Demo</h3>
  <story-characters [storyId]="7"
    (changed)=changed($event)>
  </story-characters>
</div>
```

Bind to the event in the Parent Component


Parent Component
Property Binding
Parent Template
Event Binding
Child Component

Demo

Component Input and Output

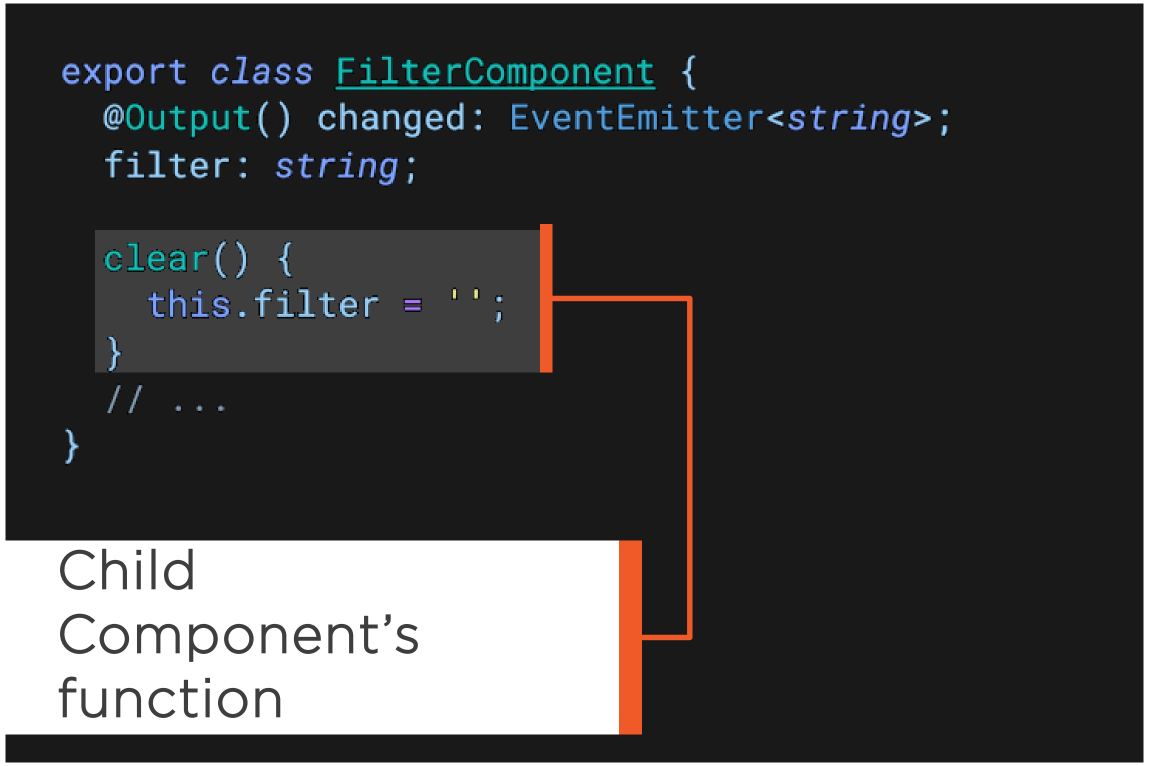
ViewChild

Use ViewChild when a parent Component needs to access a member of its child Component

ViewChild

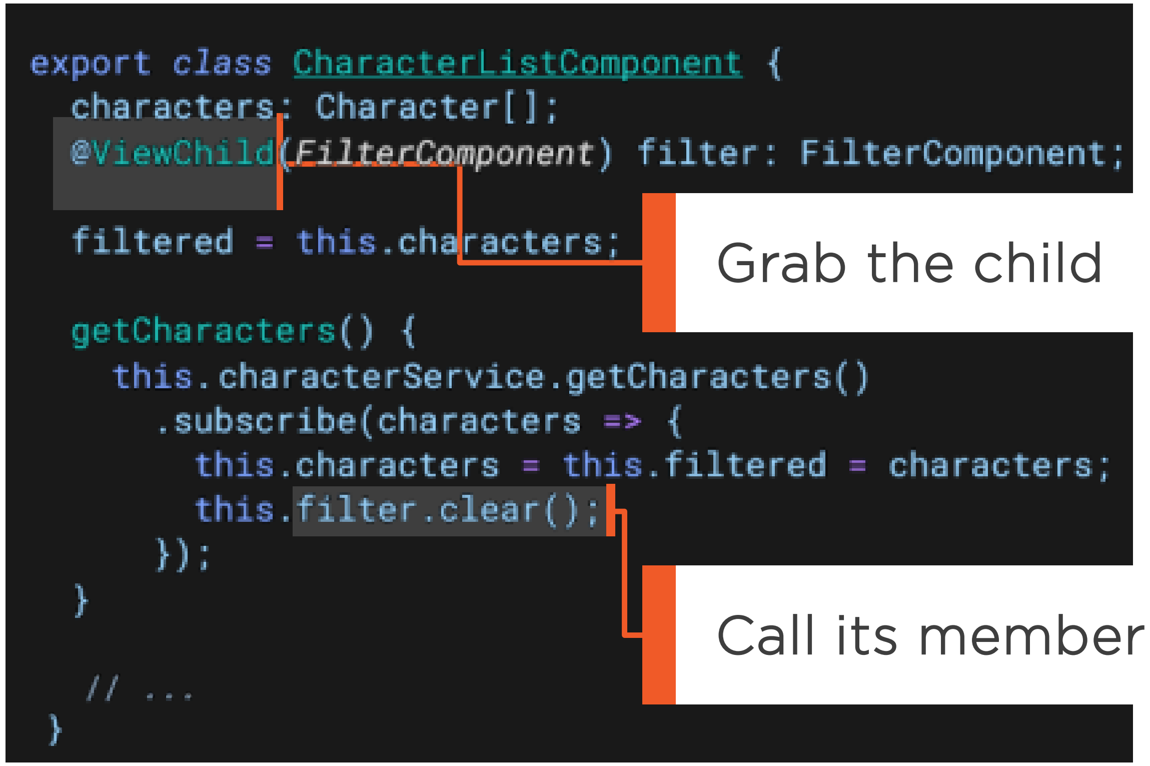
Components

Child



```
export class FilterComponent {
  @Output() changed: EventEmitter<string>;
  filter: string;

  clear() {
    this.filter = '';
  }
  // ...
}
```

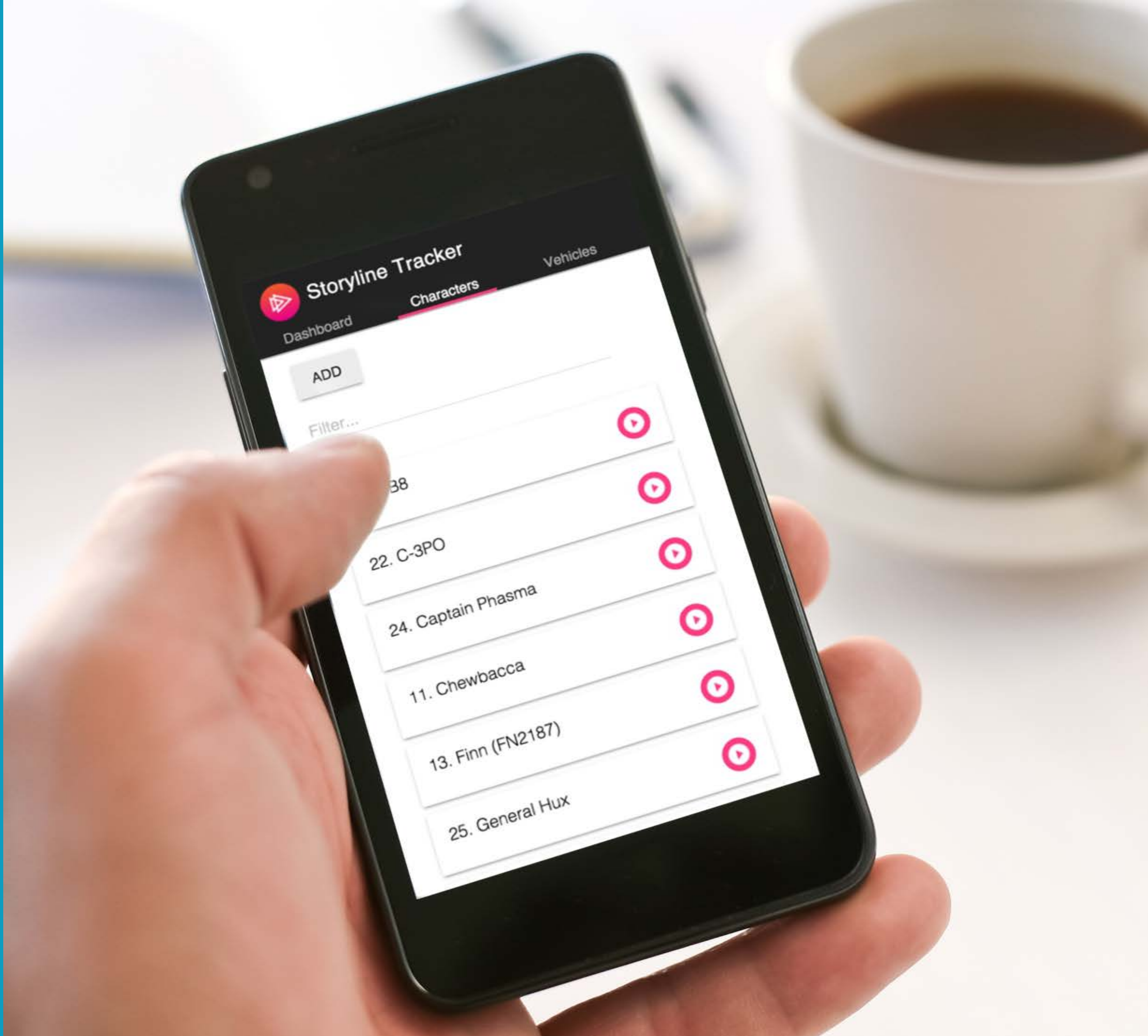
Parent



```
export class CharacterListComponent {
  characters: Character[];
  @ViewChild(FilterComponent) filter: FilterComponent;

  filtered = this.characters;

  getCharacters() {
    this.characterService.getCharacters()
      .subscribe(characters => {
        this.characters = this.filtered = characters;
        this.filter.clear();
      });
  }

  // ...
}
```

Demo

Putting It All Together



Summary

Angular Modules organize an app

Components control a region of the page

Templates tell Angular how to render

Metadata describes objects