ANGULAR

Outline

- What is Angular and why should you care!
- Single Page Application (SPA)
- Angular Architecture
- Angular features:
  - Components
  - Directives

What is ANGULAR ?

- Angular is an open source front-end web application framework for **efficiently** creating a Single Page Application (SPA)
  - SPA is a Web app that load a single HTML page and dynamically update that page as the user interacts with the app.
  - **Component based framework**
    - UI is composed of small reusable parts
    - A components encapsulates related UI elements and the behavior associated with them
  - Has built-in **client-side Template engine** that generates HTML views from an html template containing place holders that will be replaced by dynamic content
- Popular framework built by Google and has a large community behind it
    - **Google is paying developers to actively develop Angular**

Angular Competitors

- React is a strong competitor!

https://reactjs.org/

- Vue.js

https://vuejs.org/

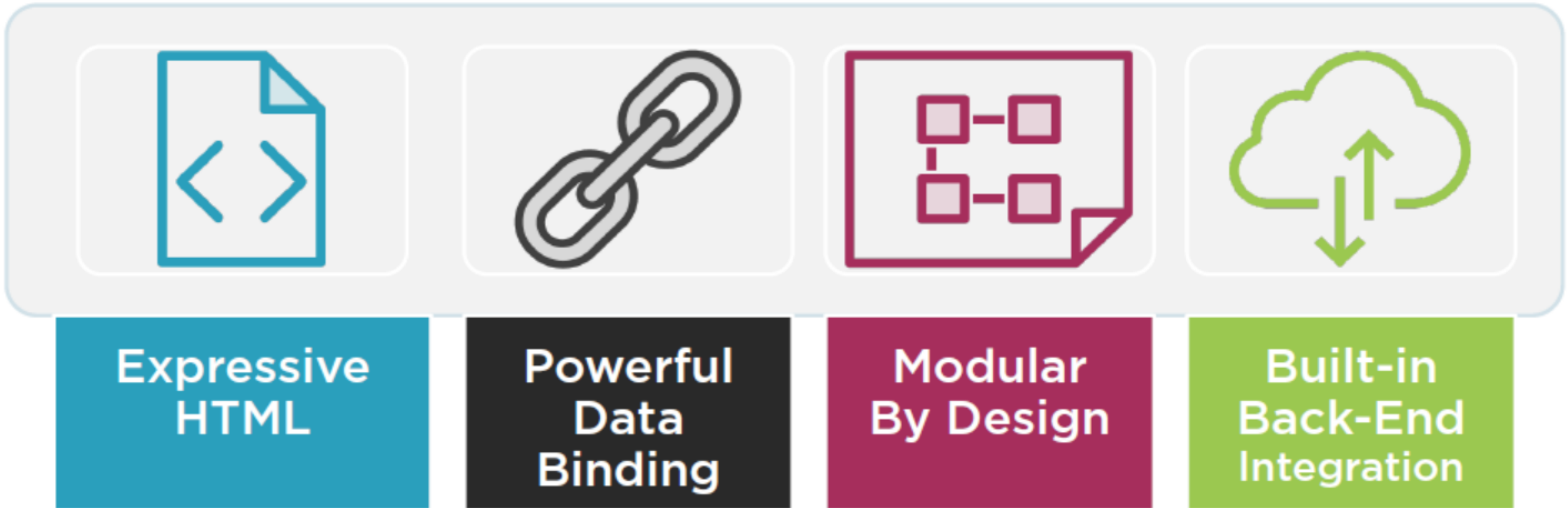
~~Direct DOM Manipulation~~

~~<form> </form> To Submit to server~~

→

Component based UI

Why Angular?

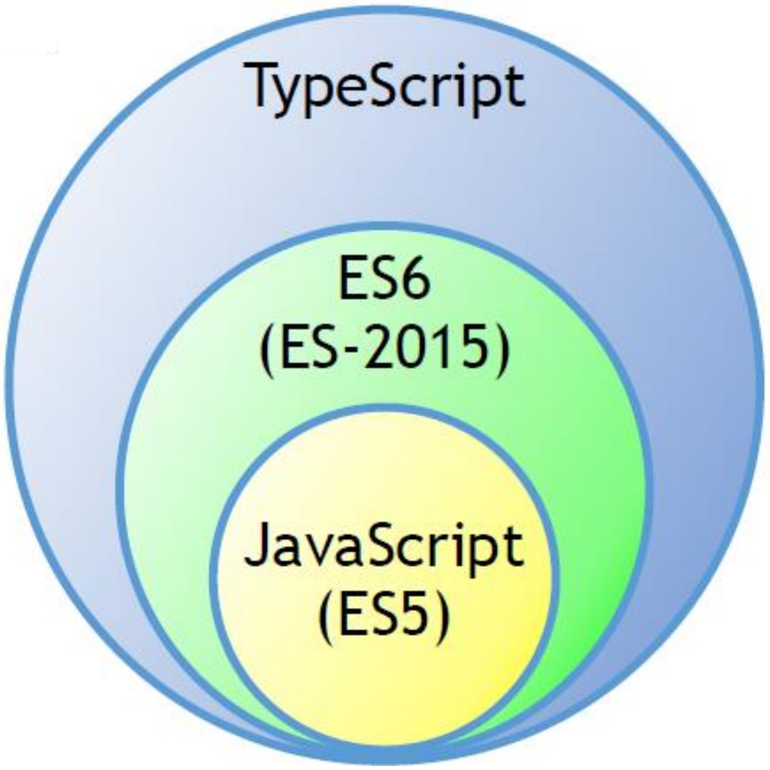

Expressive HTML
Powerful Data Binding
Modular By Design
Built-in Back-End Integration

TypeScript = JavaScript + Types

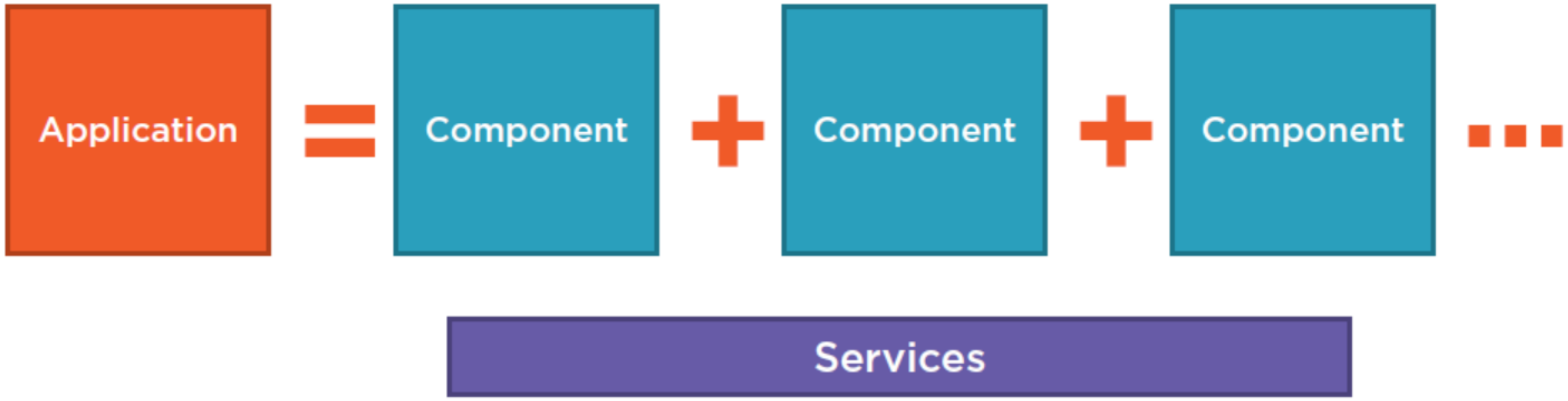
- type checking at dev time

string, number, boolean, any,
Array<T>, interfaces

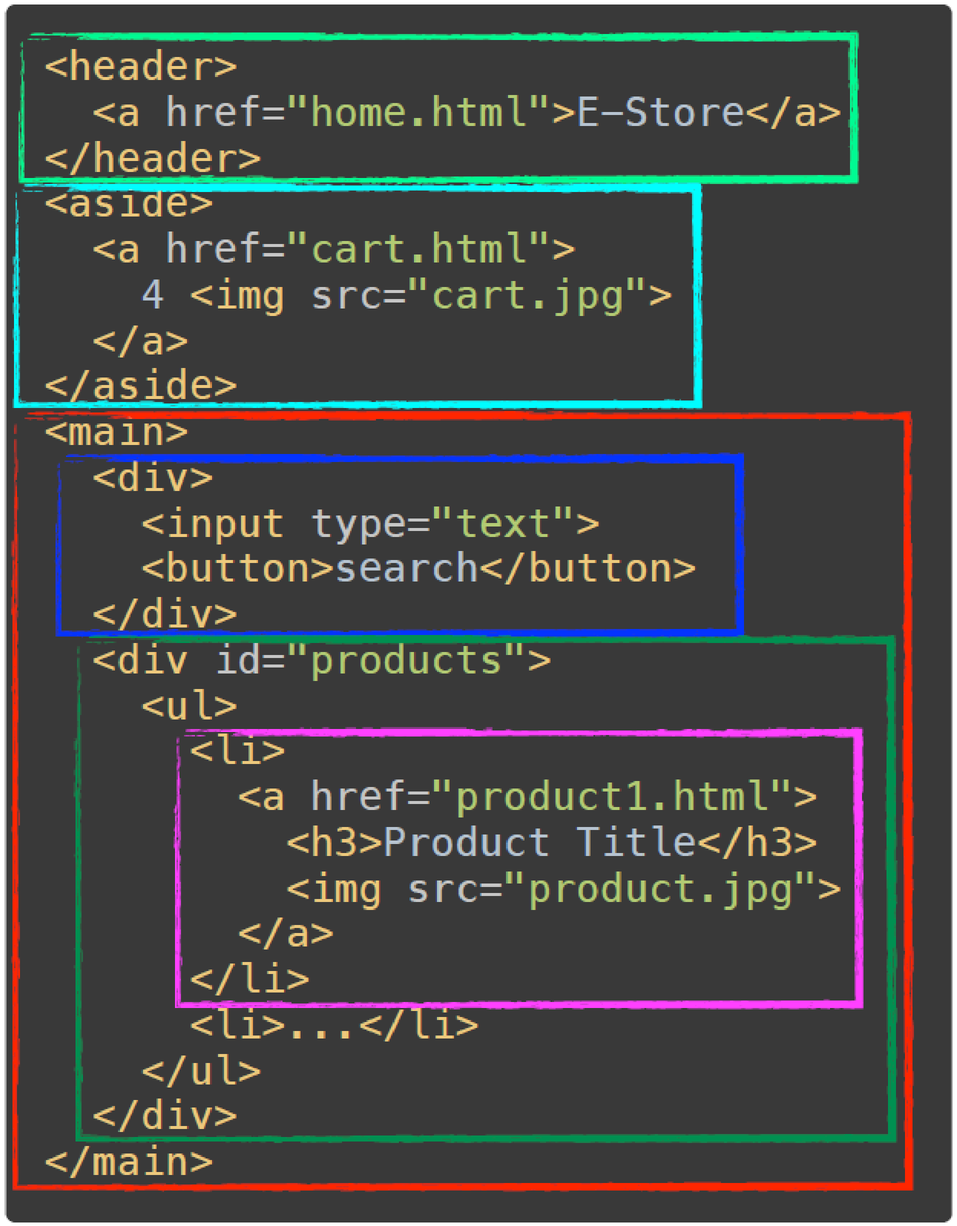
- code help - intellisense
- @decorators
- and more...



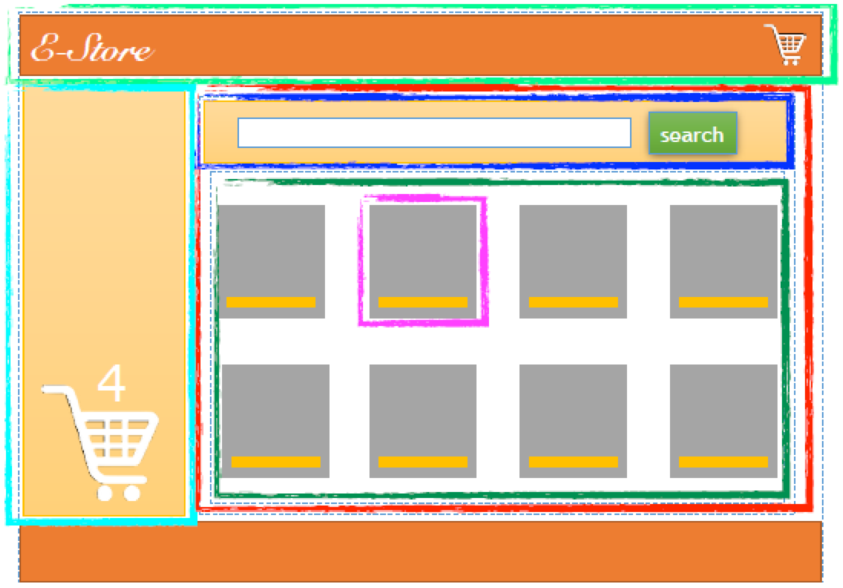
Anatomy of an Angular Application

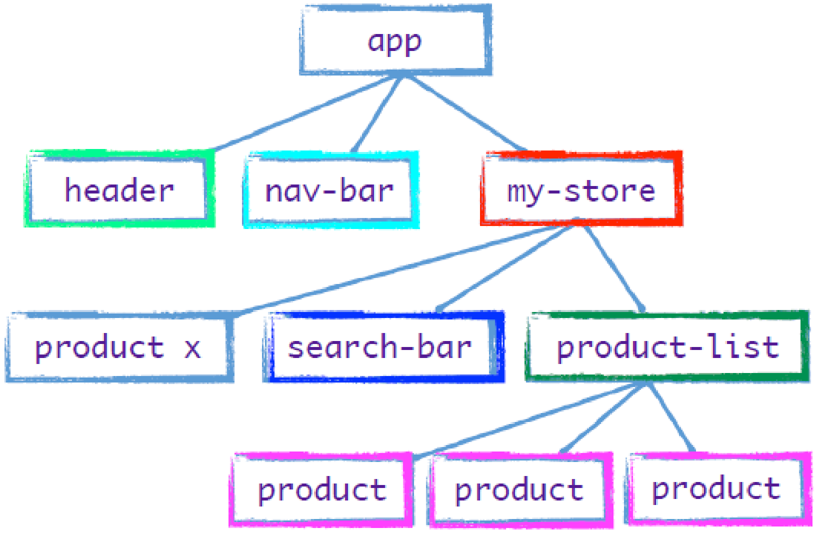


An app is a tree of components

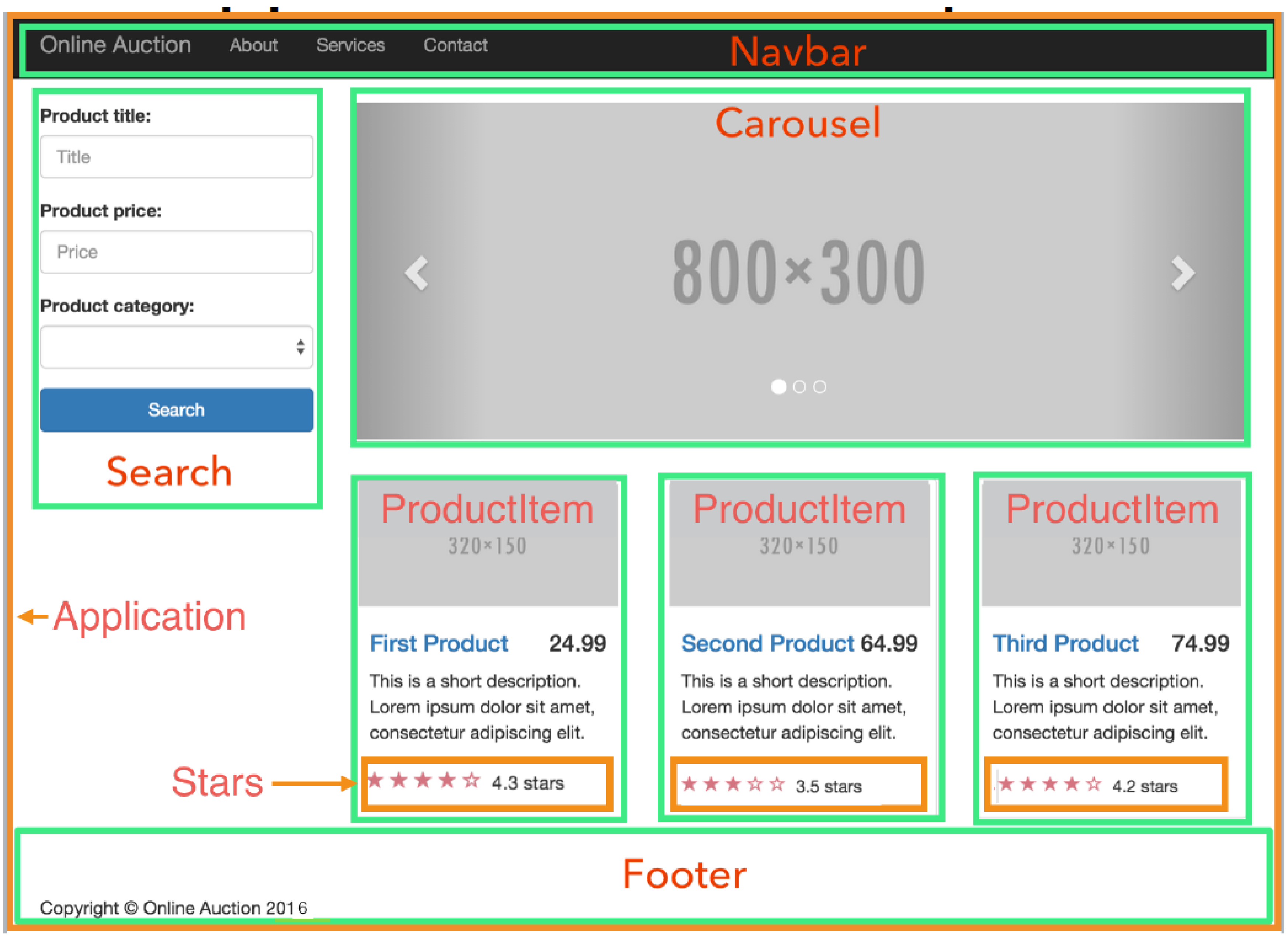


```
<header>
  <a href="home.html">E-Store</a>
</header>
<aside>
  <a href="cart.html">
    4 <img src="cart.jpg">
  </a>
</aside>
<main>
  <div>
    <input type="text">
    <button>search</button>
  </div>
  <div id="products">
    <ul>
      <li>
        <a href="product1.html">
          <h3>Product Title</h3>
          <img src="product.jpg">
        </a>
      </li>
      <li>...</li>
    </ul>
  </div>
</main>
```





An app is a tree of components

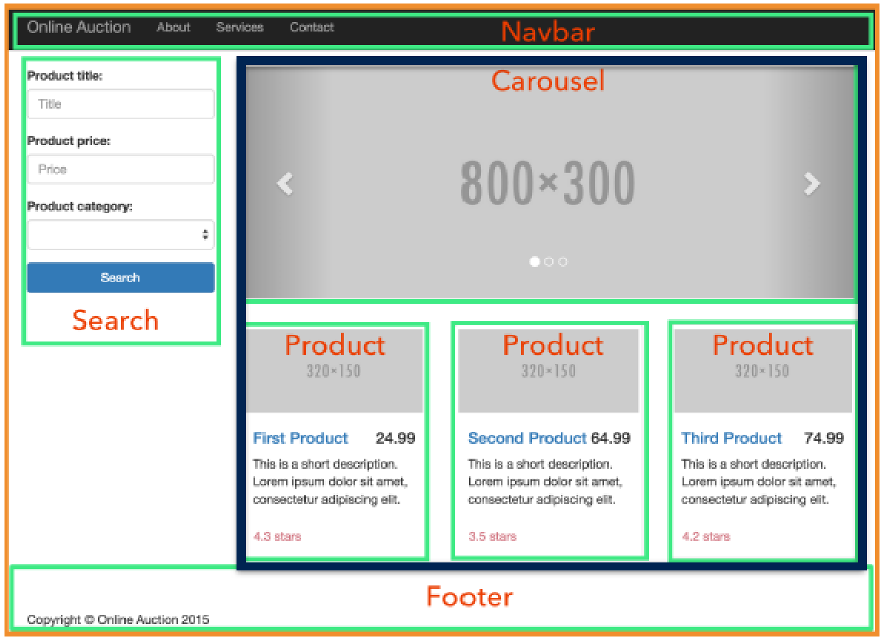


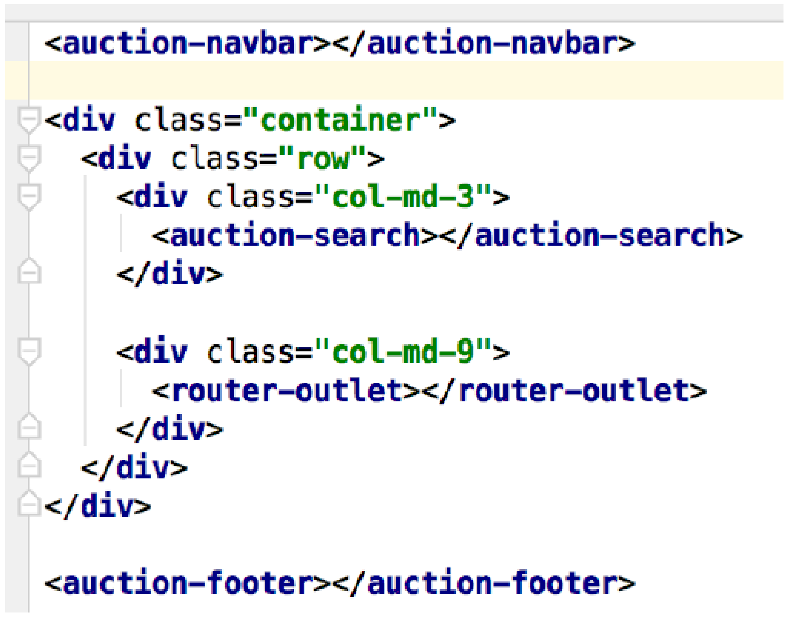
This is a short description. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

This is a short description. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

This is a short description. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

An app is a tree of components


Online Auction
About
Services
Contact
Navbar
Product title:
Title
Product price:
Price
Product category:
Search
Search
Carousel
800×300
Product
320×150
First Product 24.99
This is a short description. Lorem ipsum dolor sit amet, consectetur adipiscing elit.
4.3 stars
Product
320×150
Second Product 64.99
This is a short description. Lorem ipsum dolor sit amet, consectetur adipiscing elit.
3.5 stars
Product
320×150
Third Product 74.99
This is a short description. Lorem ipsum dolor sit amet, consectetur adipiscing elit.
4.2 stars
Footer
Copyright © Online Auction 2015



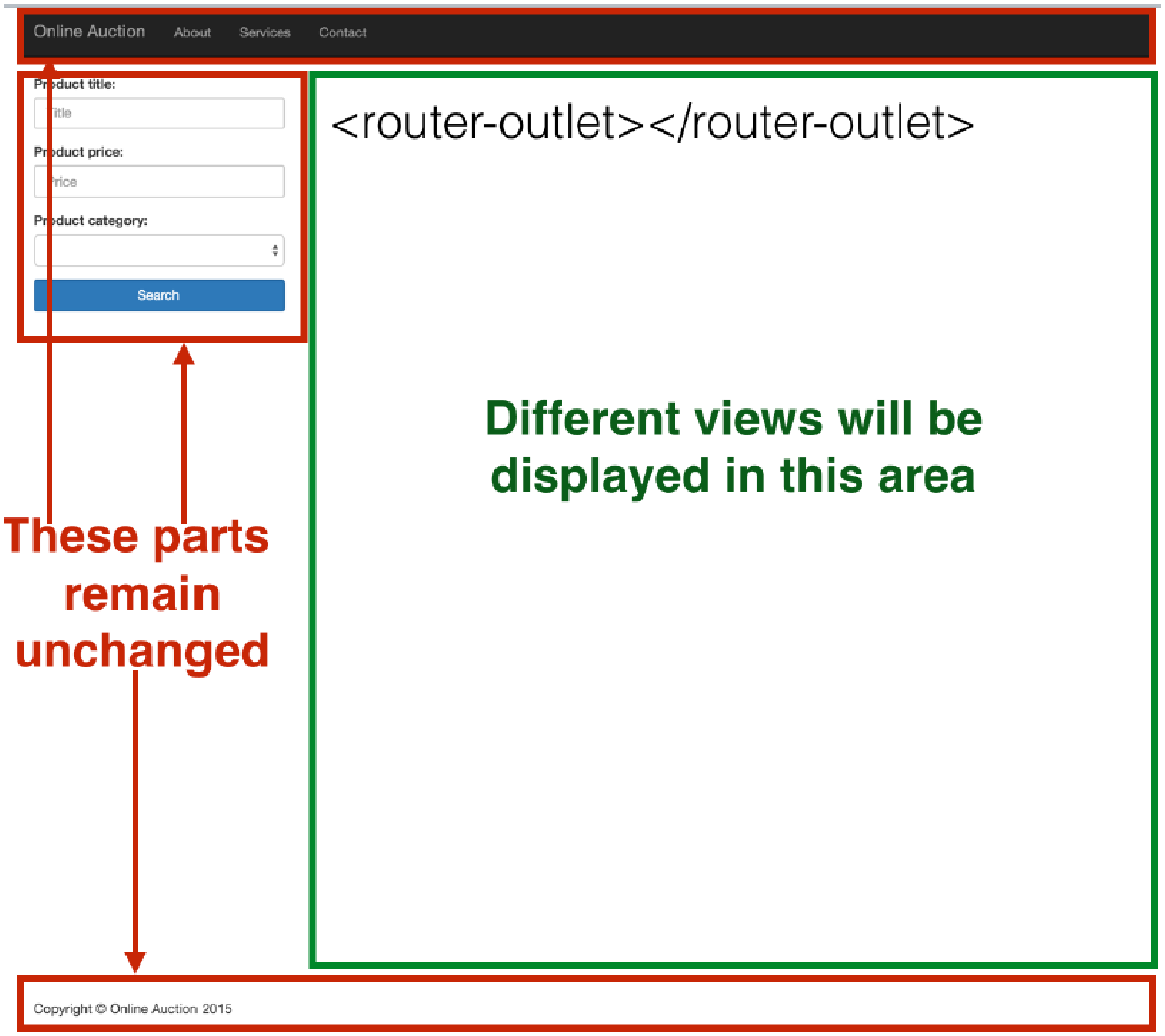
```
<auction-navbar></auction-navbar>

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

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

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

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

@Component({
  selector: 'app-root',
  templateUrl: 'application.html',
  styleUrls: ['application.css']
})
export class AppComponent {
  products: Array<Product> = [];

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

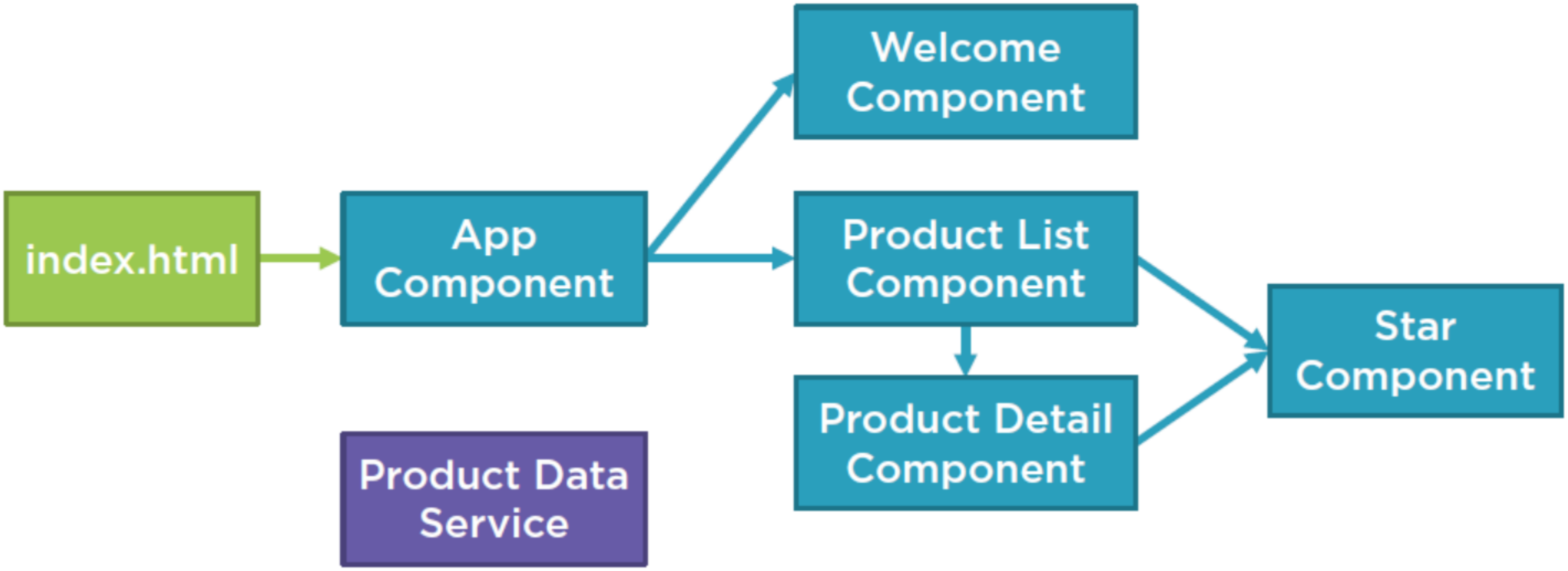
HTML, CSS

TypeScript

Single Page App


Online Auction
About
Services
Contact
Product title:
Title
Product price:
Price
Product category:
Search
<router-outlet></router-outlet>
Different views will be displayed in this area
These parts remain unchanged
Copyright © Online Auction 2015

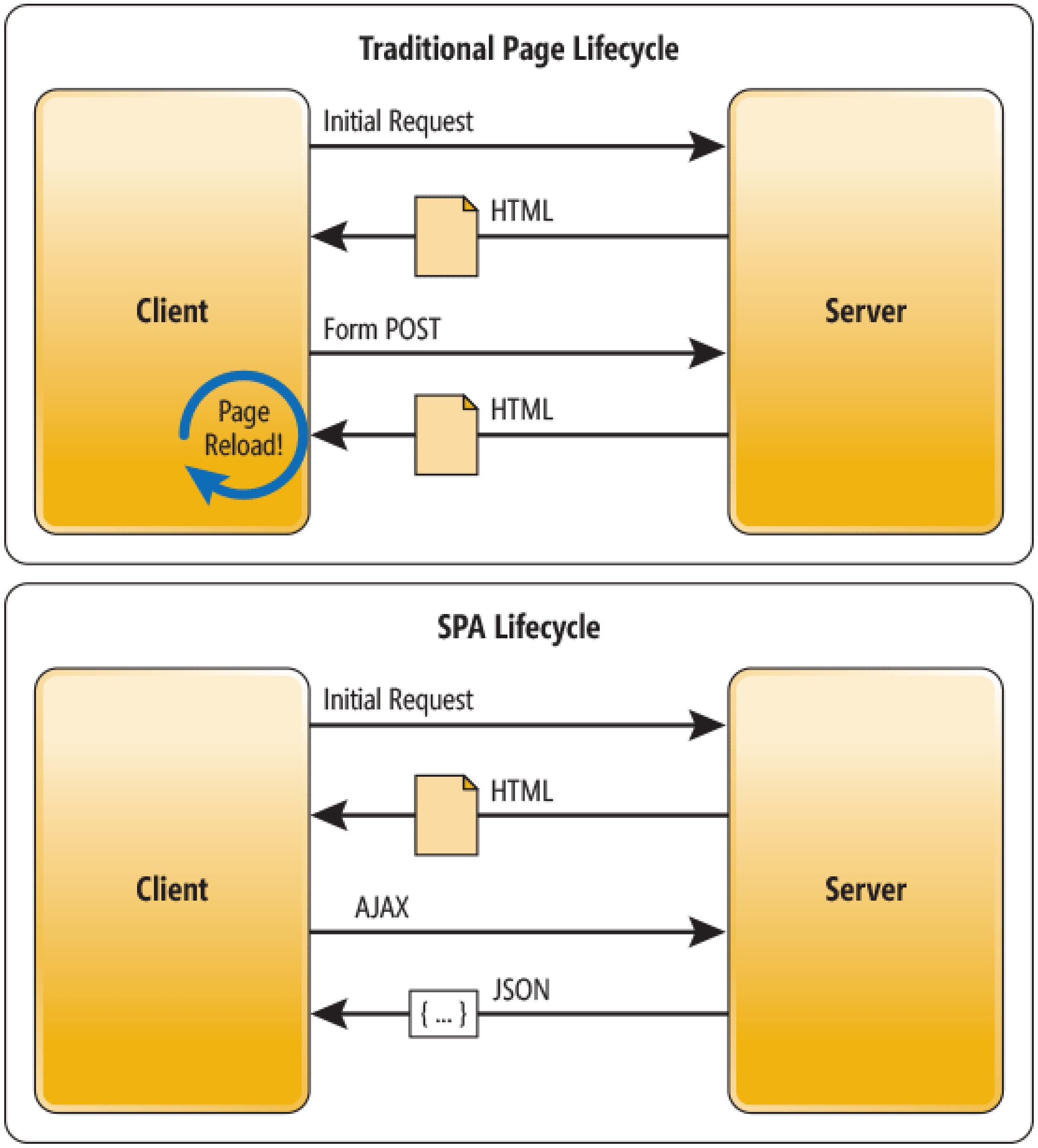
Sample Application Architecture



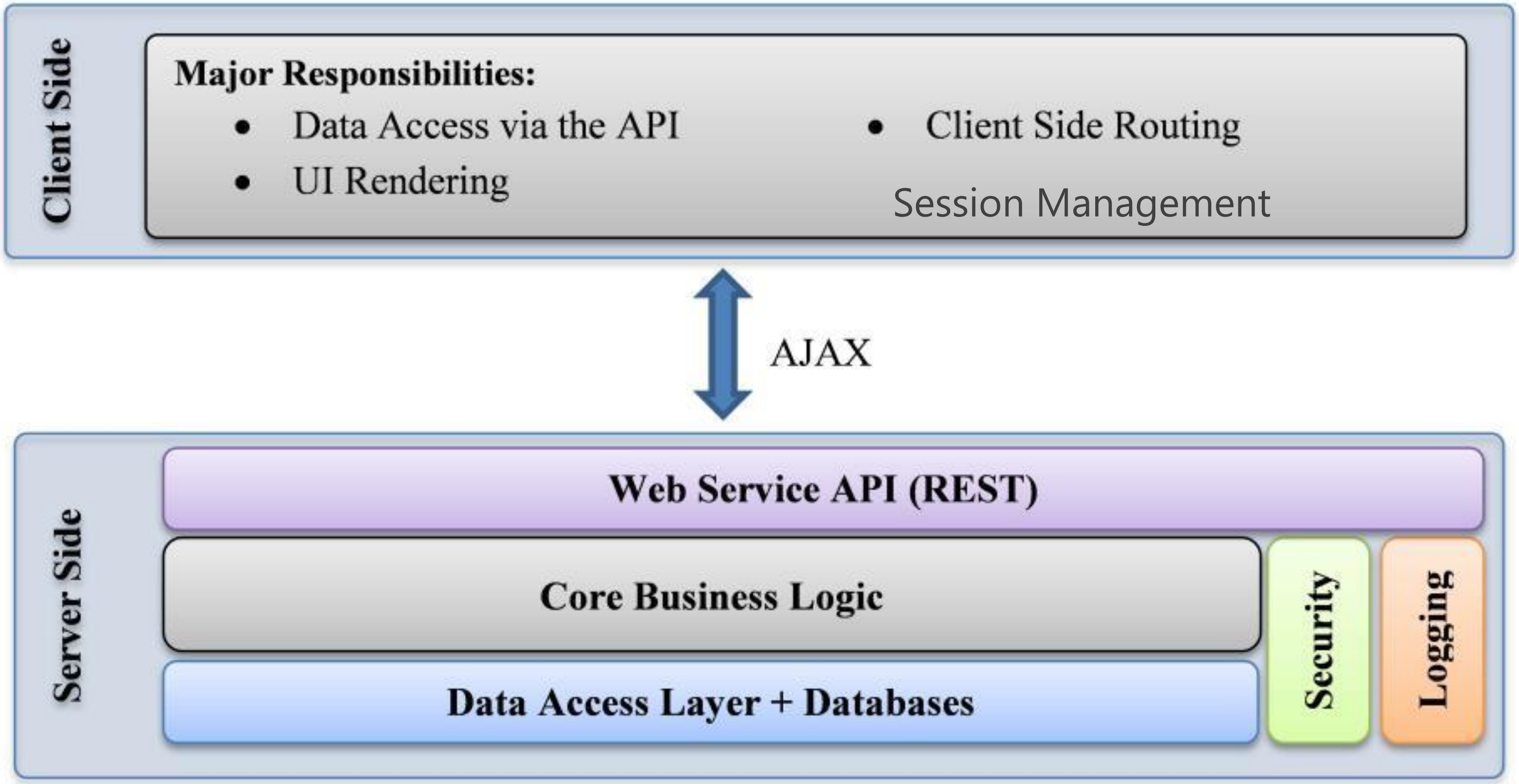
Single Page Application (SPA)

Back

Traditional vs. SPA Lifecycle

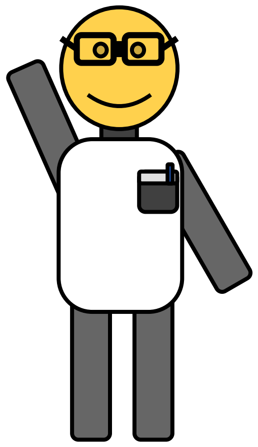


Role of Client and Server in SPA



Benefits of a Single Page App

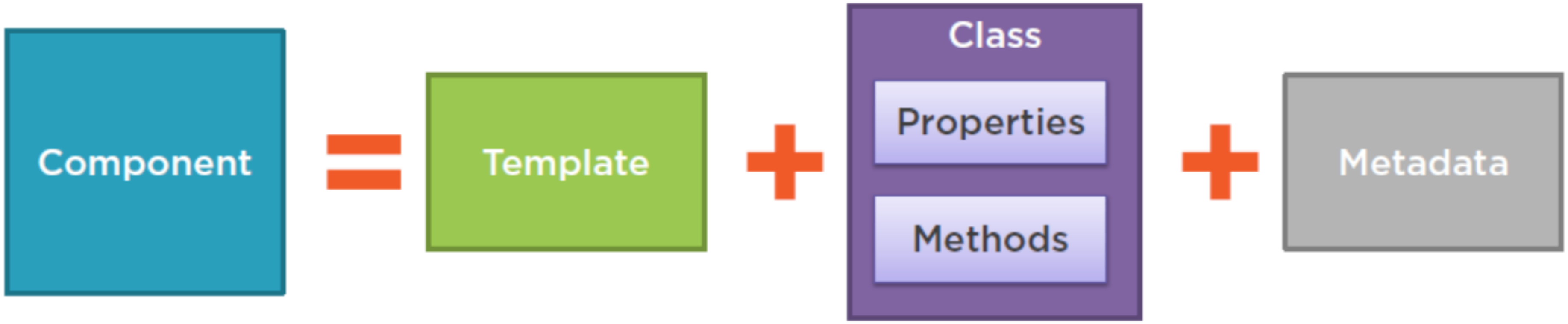
- **Better User experience**
- More interactive and responsive
- Less network activity and waiting
- Developer experience
  - Better (if you use a framework!)
  - No constant DOM refresh
- **State can be maintained on client + offline support**
  - Can use HTML5 JavaScript APIs to store state in the browser's localStorage



Angular App Architecture

Back

Component

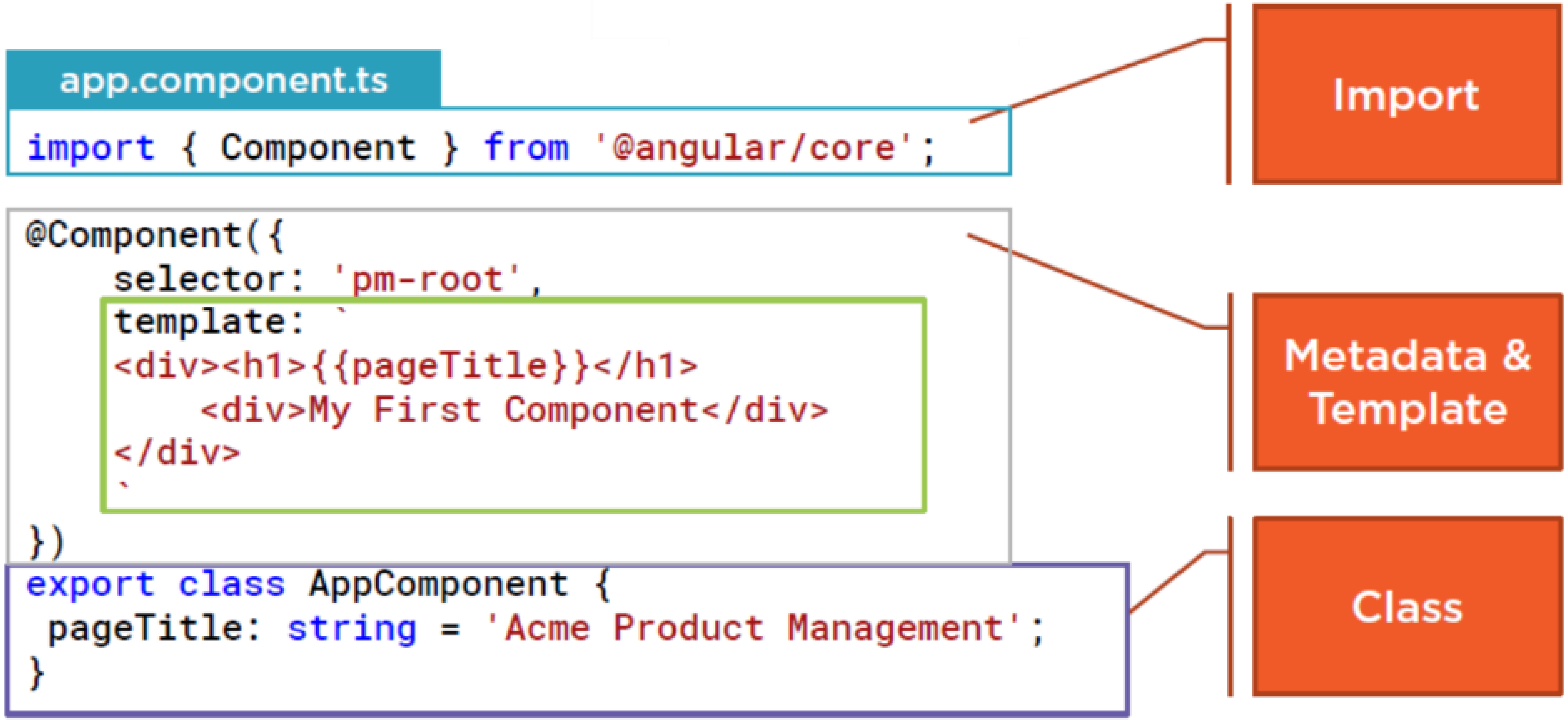


- View layout
- Created with HTML
- Includes binding and directives

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

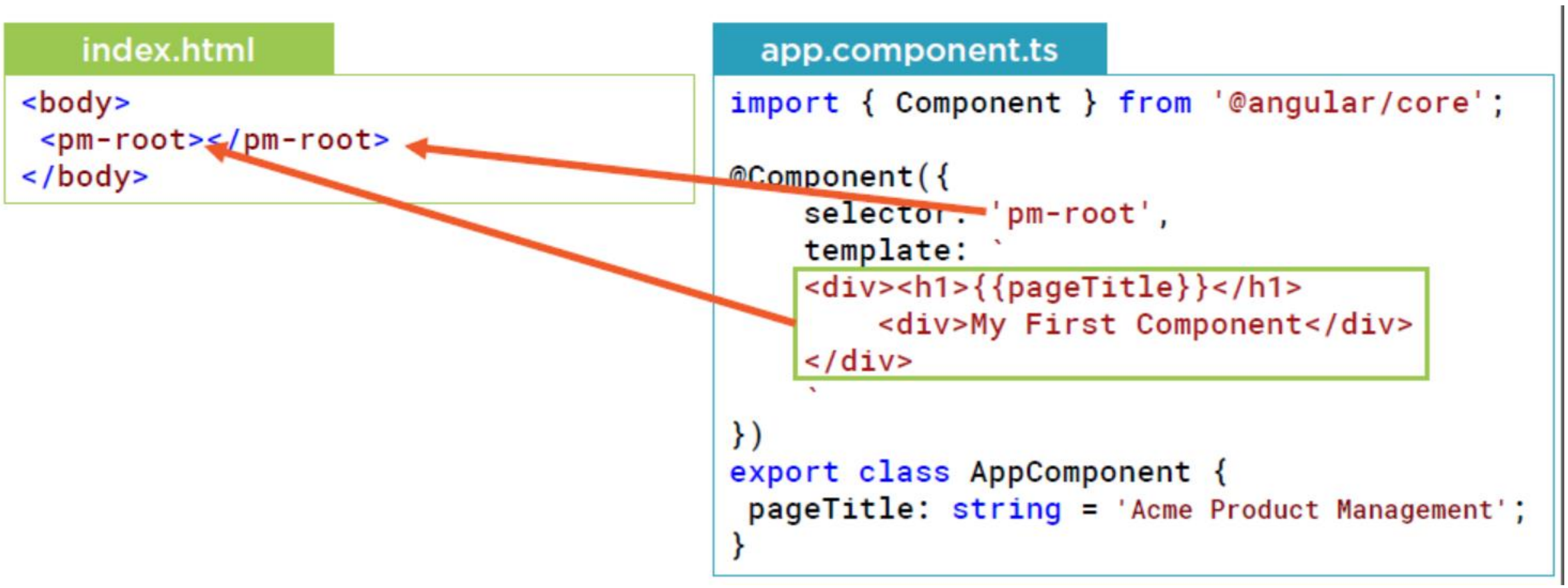
- Extra data for Angular
- Defined with a decorator

Component Example



```
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';
}
```

Loading a Component in the Shell Page

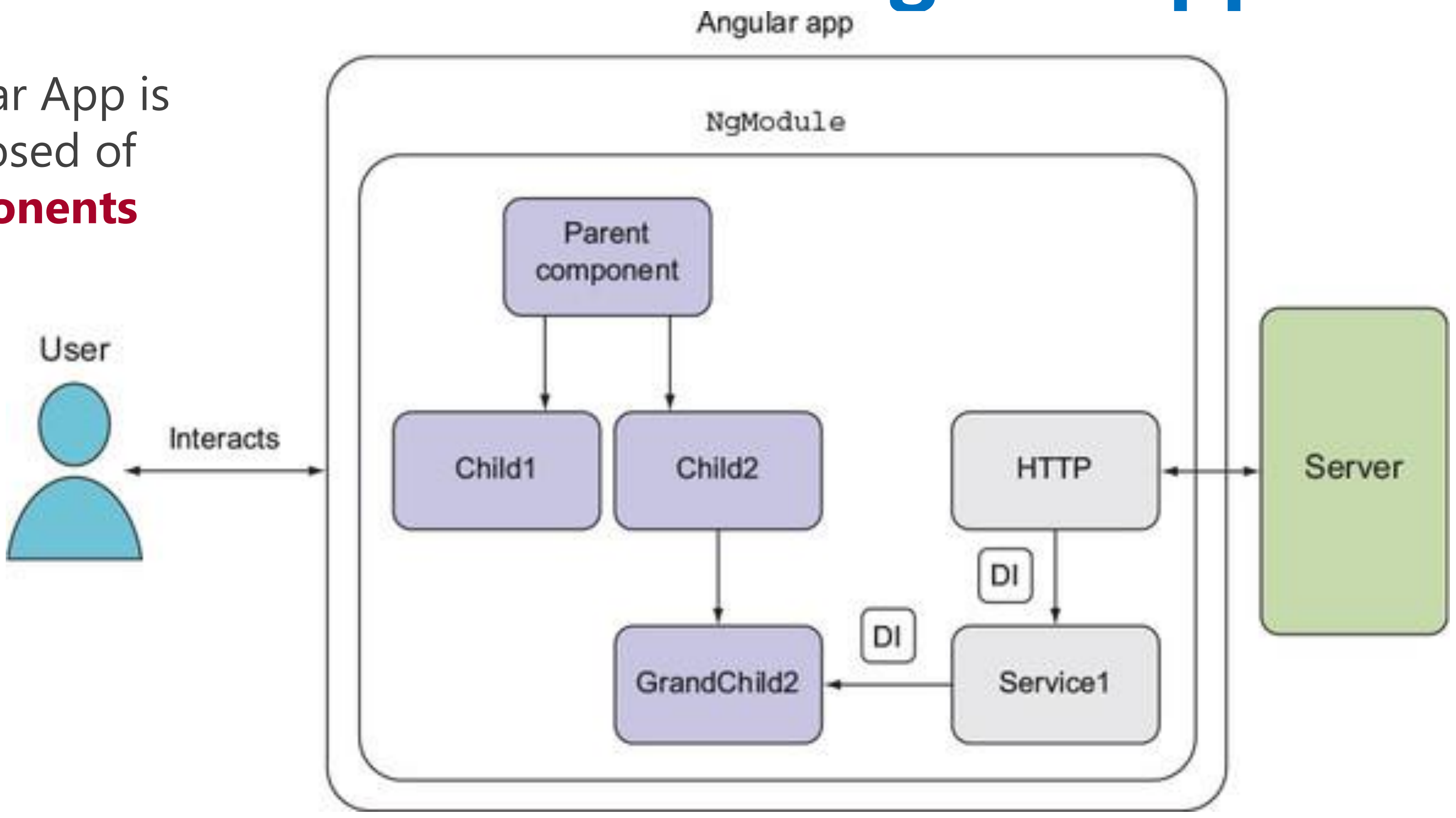


```
<body>
 <pm-root></pm-root>
</body>
```

```
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';
}
```

Architecture of an Angular app

Angular App is composed of **components**



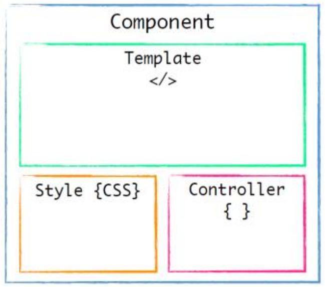
The figure shows a high-level diagram of a sample Angular application that consists of four components and two services; all of them are packaged inside a module. Angular's Dependency Injection (DI) module injects the Http service into Service1, which in turn is injected into the GrandChild2 component.

Source: '*Angular 2 Development with TypeScript*', Yakov Fain and Anton Moiseev, ISBN 9781617293122

Angular Architecture Highlights

- Angular App is composed of components.
  - A ***component*** has an HTML *template* and a class to ***provide data*** and ***handle events*** raised from the template.
  - Application logic in encapsulated in ***services*** that can be injected in components.
- A Component is a class (presentation logic) annotated with **@Component** annotation, it specifies:
  - a **selector** declaring the name of the custom tag to be used to load to component in HTML document
  - the **template** (=an HTML fragment with data binding expressions to render by the view) or **templateURL**

Component Example

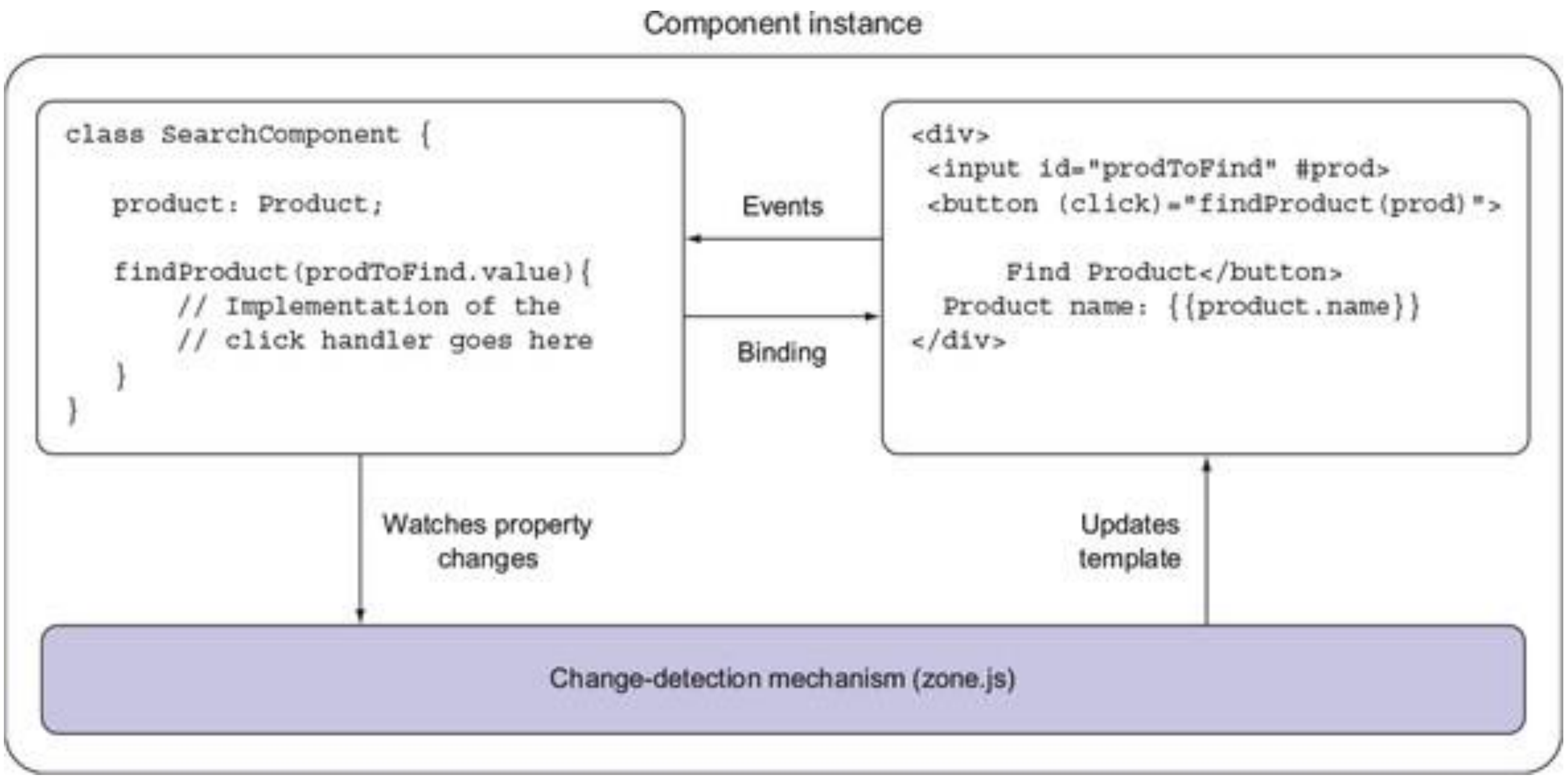


```
import { Component } from '@angular/core';

@Component({
  selector: 'app-hello',
  template: `
    <h1>{{ title }}</h1>`
})

export class HelloComponent {

 title = 'Hello World!';

}
```

Somewhere in your app

```
  <app-hello></app-hello>
```

Component internals



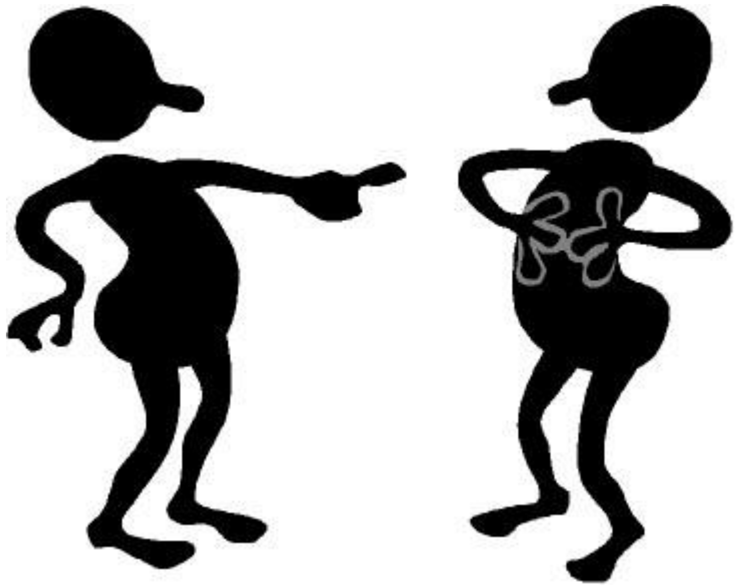
```
class SearchComponent {

   product: Product;

   findProduct(prodToFind.value){
       // Implementation of the
       // click handler goes here
   }
}
```

```
<div>
 <input id="prodToFind" #prod>
 <button (click)="findProduct(prod)">

      Find Product</button>
  Product name: {{product.name}}
</div>
```

Component is a unit encapsulating the presentation logic and the auto-generated change detector

Source: '*Angular 2 Development with TypeScript*', Yakov Fain and Anton Moiseev, ISBN 9781617293122

Directives



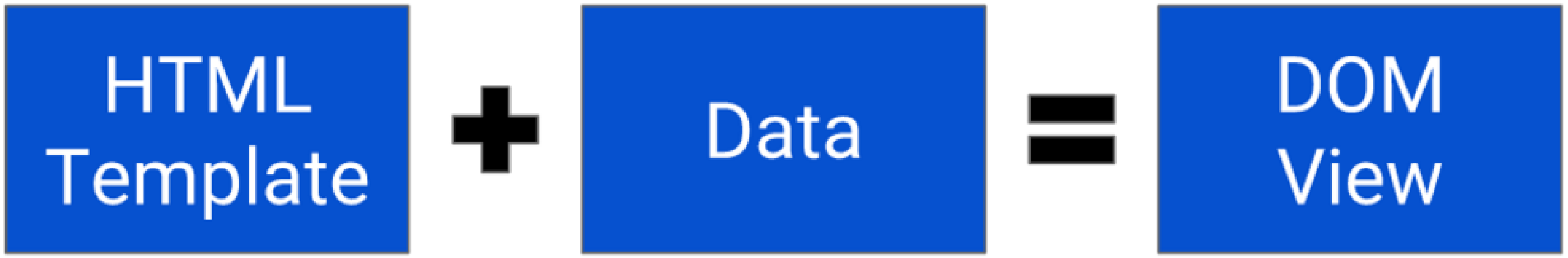
Back

Directives

- Directives are used to create **client-side HTML templates**
  - Adds additional markup to the view (e.g., dynamic content place holders)
  - A directive is just a function which executes when Angular ‘compiler’ encounters it in the DOM
  - Built-in directives start with ***ng** and they cover the core needs

HTML Template

- Template is:
  - Partial HTML file that contains only part of a web page
  - Contains HTML augmented with Angular Directives
  - Rendered in a "parent" view



Common Built-in Directives : ngFor

- **ngFor**: **repeater** directive. It marks <li> element (and its children) as the "repeater template"

```
<li *ngFor="#hero of heroes">
      {{ hero }}
</li>
```

- The **#hero** declares a local variable named hero

Common Built-in Directives : ngIf

- **ngIf**: conditional display of a portion of a view only if certain condition is true

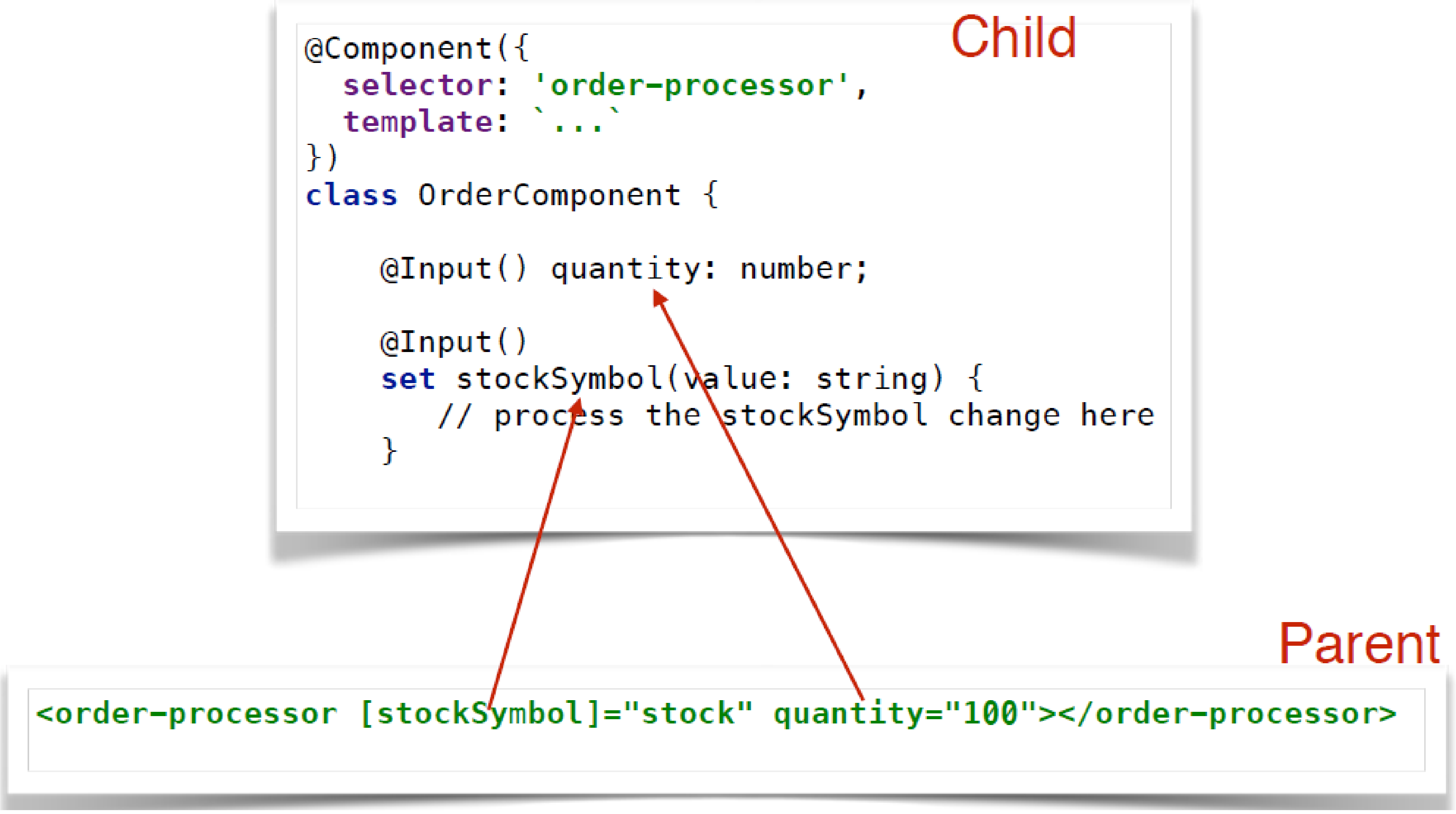
```
<p *ngIf="heroes.length > 3">There are many heroes!</p>
```

- This element will be displayed only if *heroes.length > 3*

Inter-component communications

Back

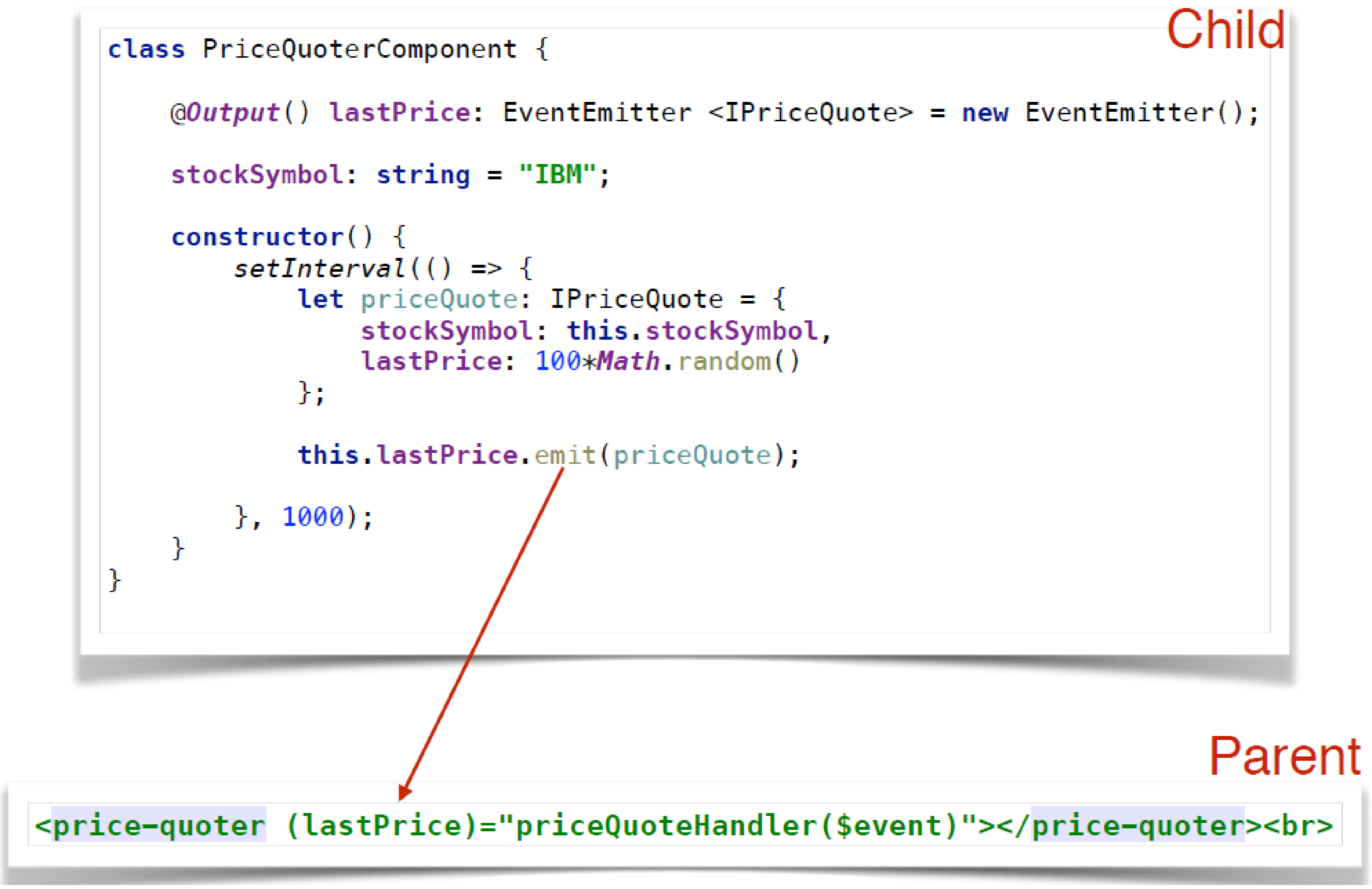
@Input properties



```
@Component({
  selector: 'order-processor',
  template: `...`
})
class OrderComponent {

    @Input() quantity: number;

    @Input()
    set stockSymbol(value: string) {
       // process the stockSymbol change here
    }
```

```
<order-processor [stockSymbol]="stock" quantity="100"></order-processor>
```

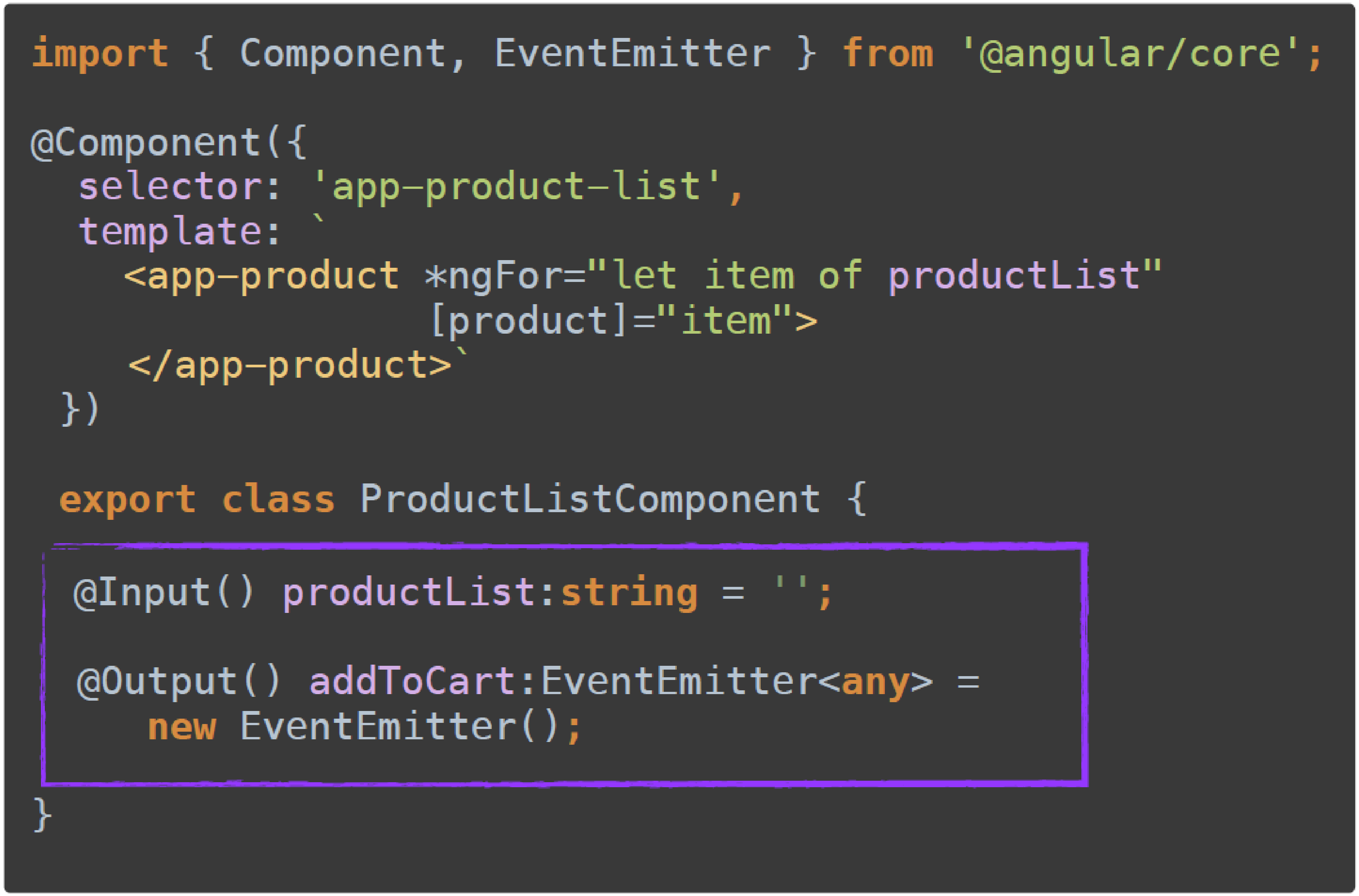
@Output properties



```
class PriceQuoterComponent {

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

    stockSymbol: string = "IBM";

    constructor() {
        setInterval(() => {
            let priceQuote: IPriceQuote = {
                stockSymbol: this.stockSymbol,
                lastPrice: 100*Math.random()
            };

            this.lastPrice.emit(priceQuote);

        }, 1000);
    }
}
```

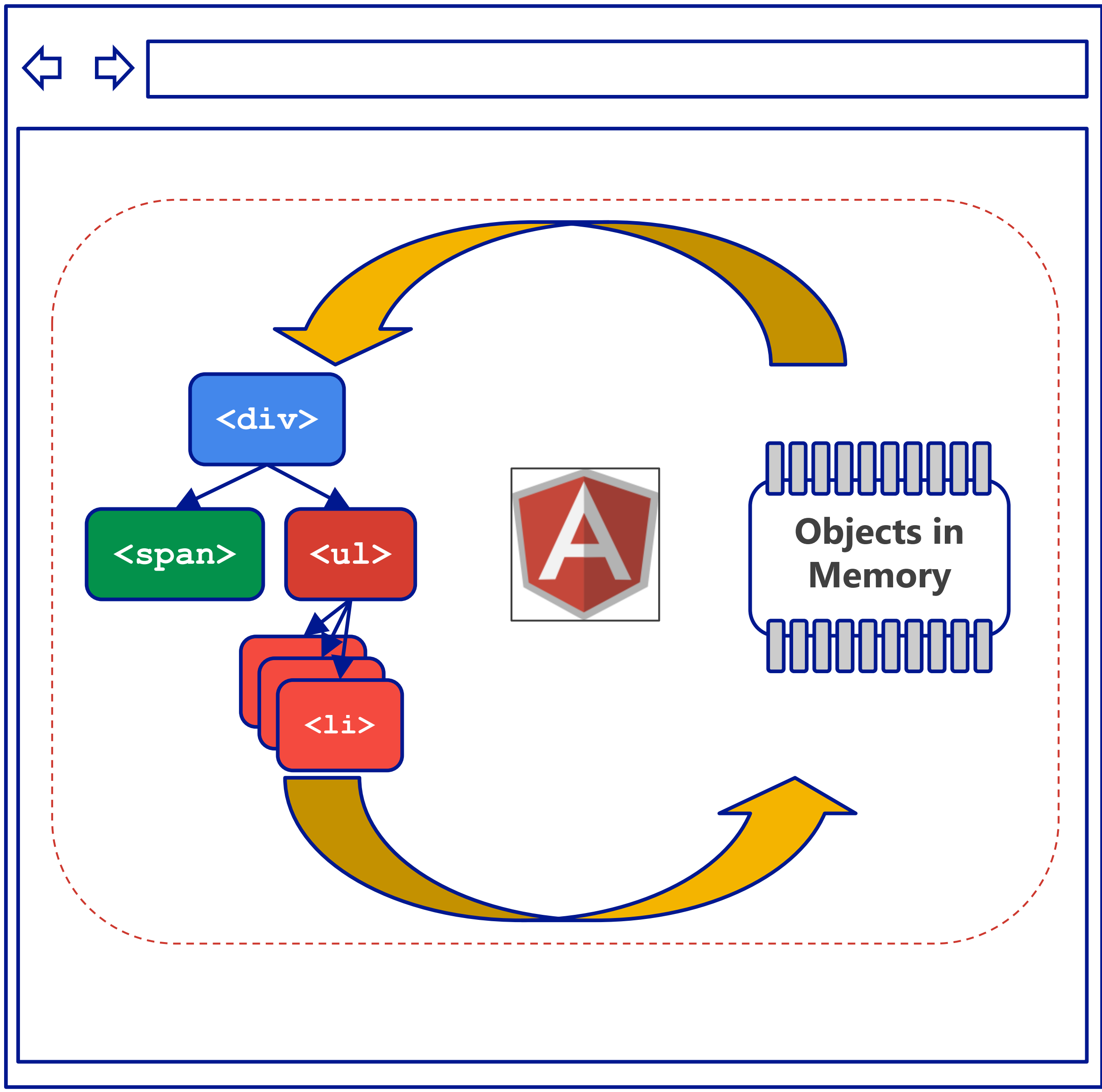
```
<price-quoter (lastPrice)="priceQuoteHandler($event)"></price-quoter><br>
```

Another Example



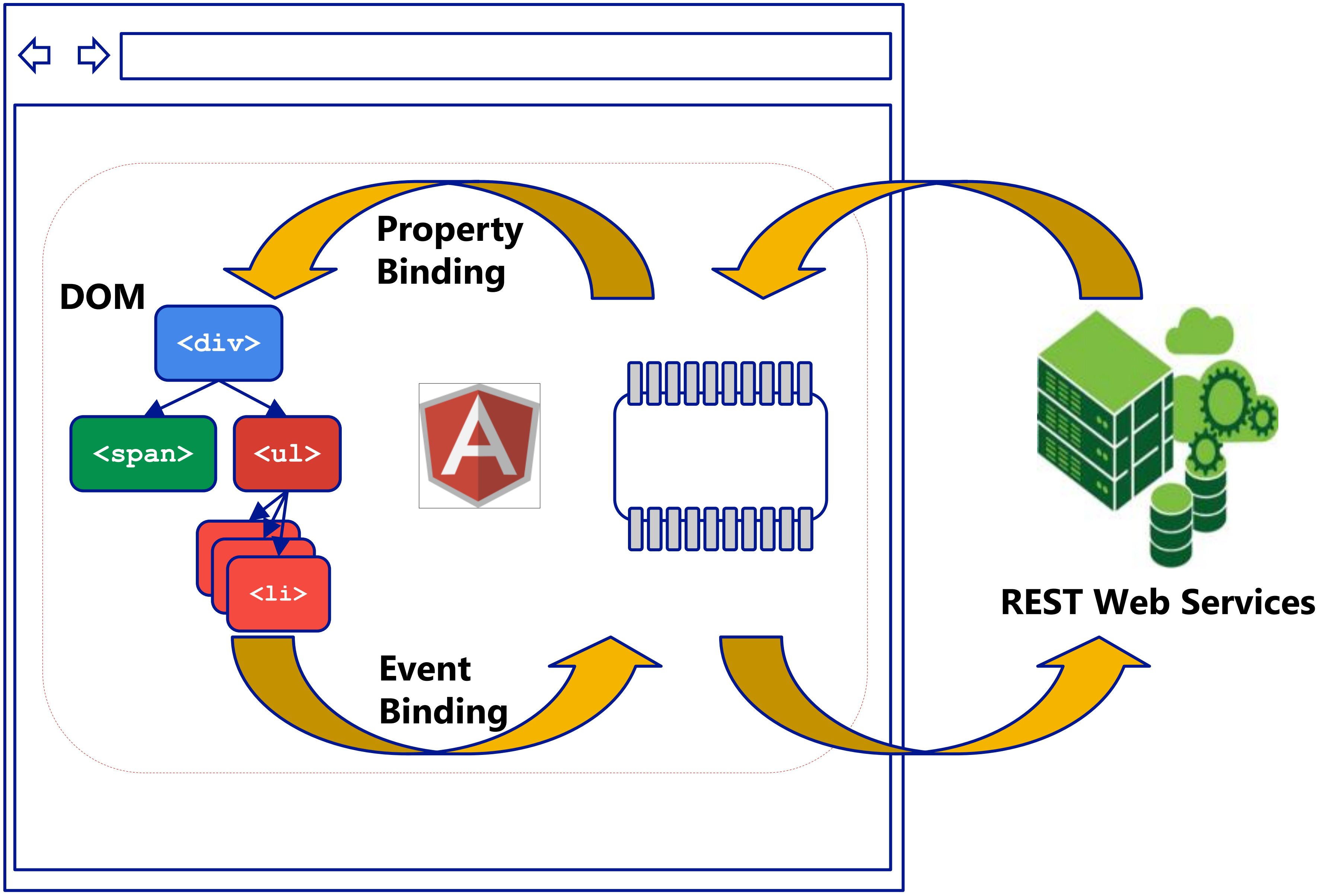
```
import { Component, EventEmitter } from '@angular/core';

@Component({
  selector: 'app-product-list',
  template: `
    <app-product *ngFor="let item of productList"
                 [product]="item">
    </app-product>`
 })

 export class ProductListComponent {

  @Input() productList:string = '';

  @Output() addToCart:EventEmitter<any> =
     new EventEmitter();

}
```

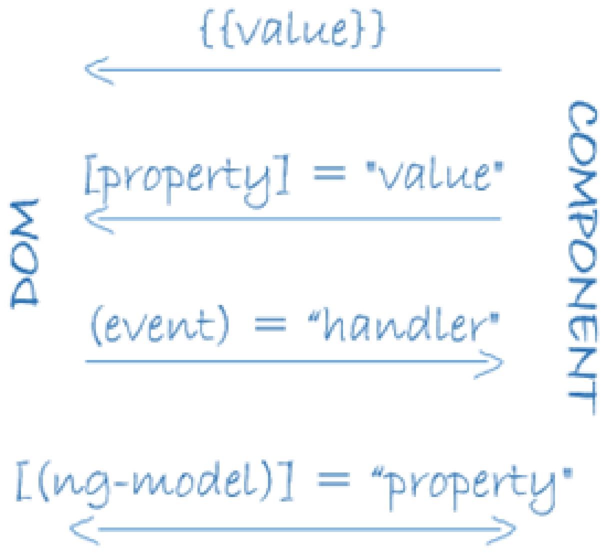
Binding



Back

Binding - big picture



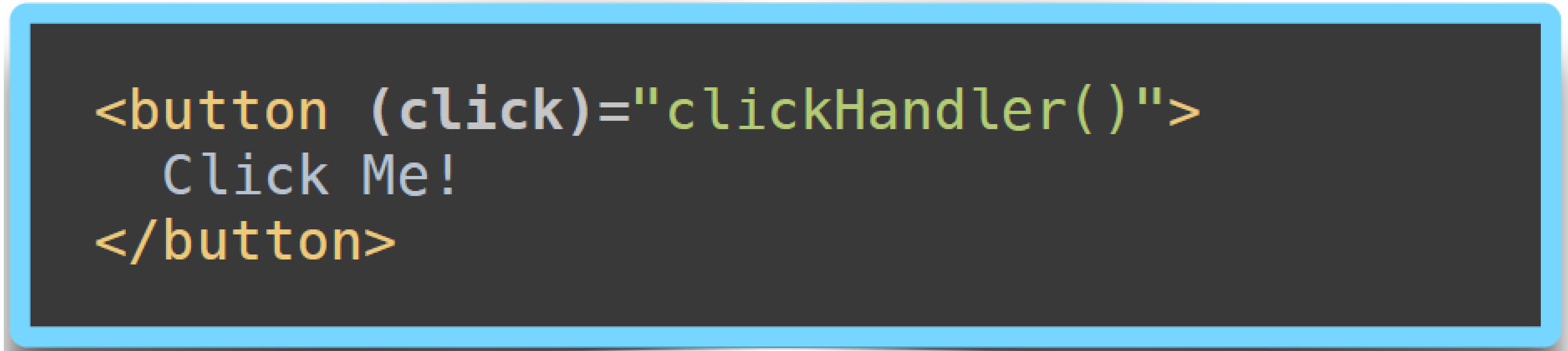


Things you can bind to

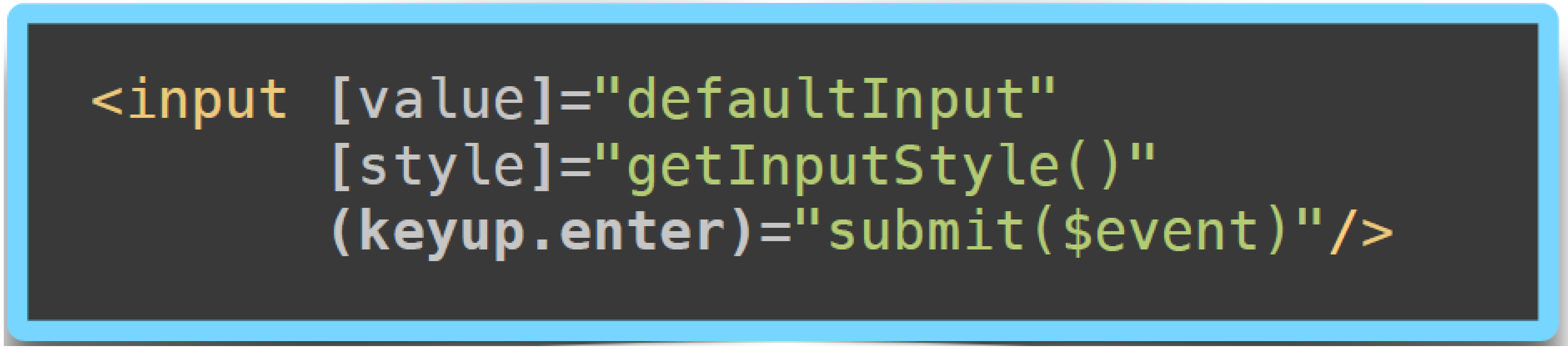
Binding	Example
Properties	<input **[value]**="firstName">
Events	<button **(click)**="buy($event)">
Two-way	<input **[(ngModel)]**="userName">

Data binding associates the Model with the View

Property & Event Binding

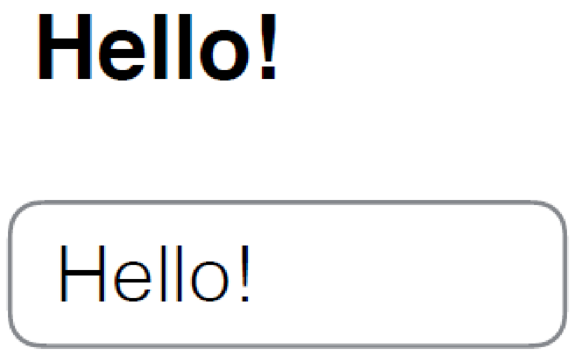


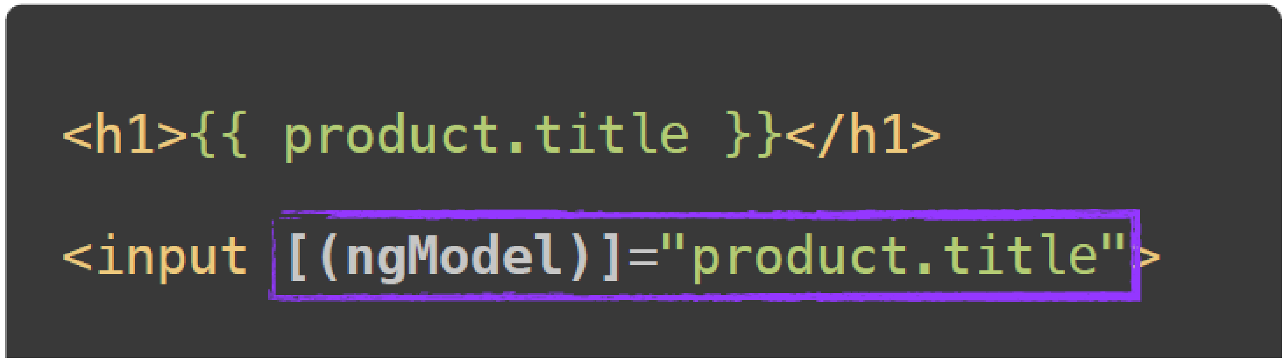
```
<button (click)="clickHandler()">
  Click Me!
</button>
```



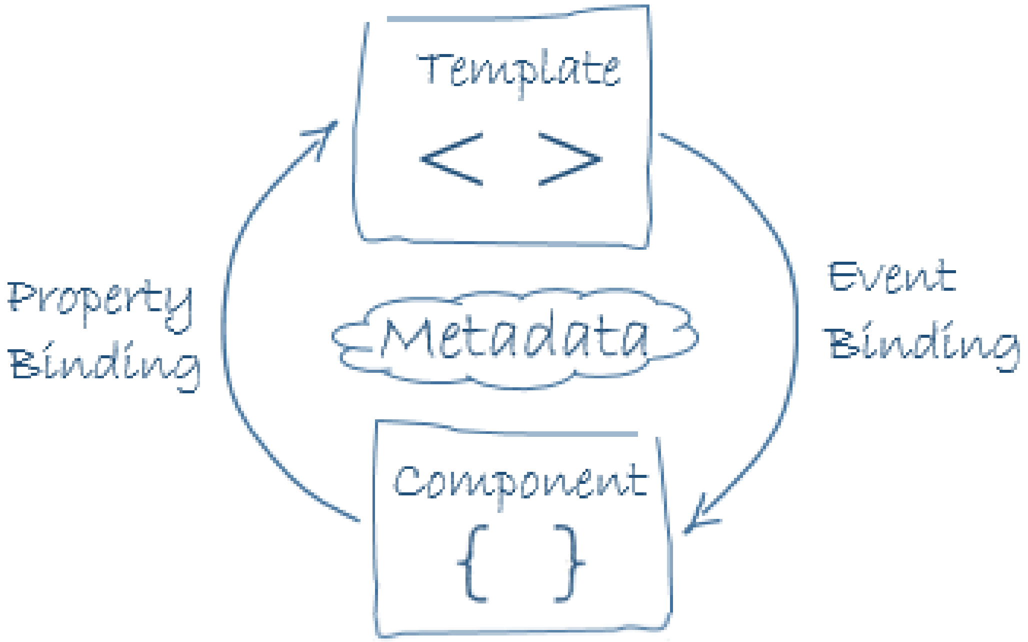
```
<input [value]="defaultInput"
       [style]="getInputStyle()"
       (keyup.enter)="submit($event)"/>
```

Inputs & Outputs - ngModel

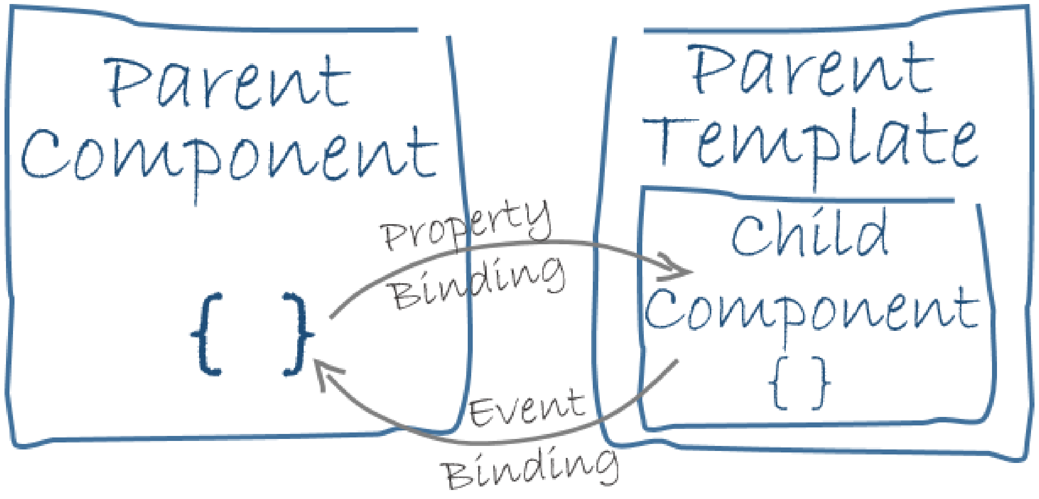




```
<h1>{{ product.title }}</h1>

<input [(ngModel)]="product.title">
```



Communication between a template and its component



Communication between parent and child components

Example

```
<button
        [disabled]="!inputIsValid"
        (click)="authenticate()">
    Login
</button>
```

Calls a function defined in the component class

```
<amazing-chart
        [series]="mySeries"
        (drag)="handleDrag()"/>
```

```
<div *ngFor="#guest of guestList">
  <guest-card [guest]="guest"></guest-card>
</div>
```

Angular Event Binding syntax

- **(eventName) = eventHandler**: respond to the click event by calling the component's onBtnClick method

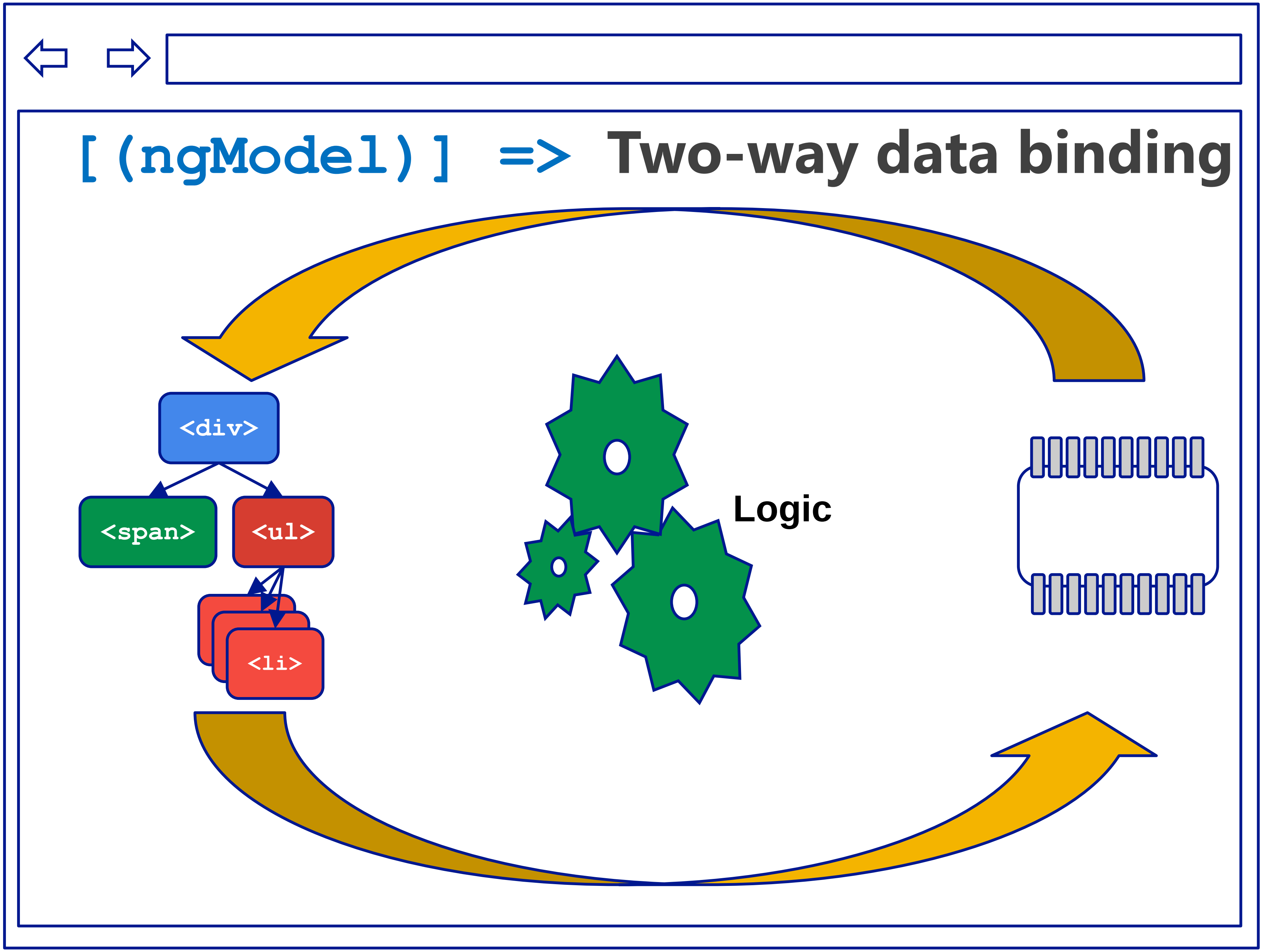
```
<button (click)="onBtnClick()">Click me!</button>
<input (keyup)="onKey($event)">
```

- $event is an optional standard DOM event object. It is value is determined by the source of the event.

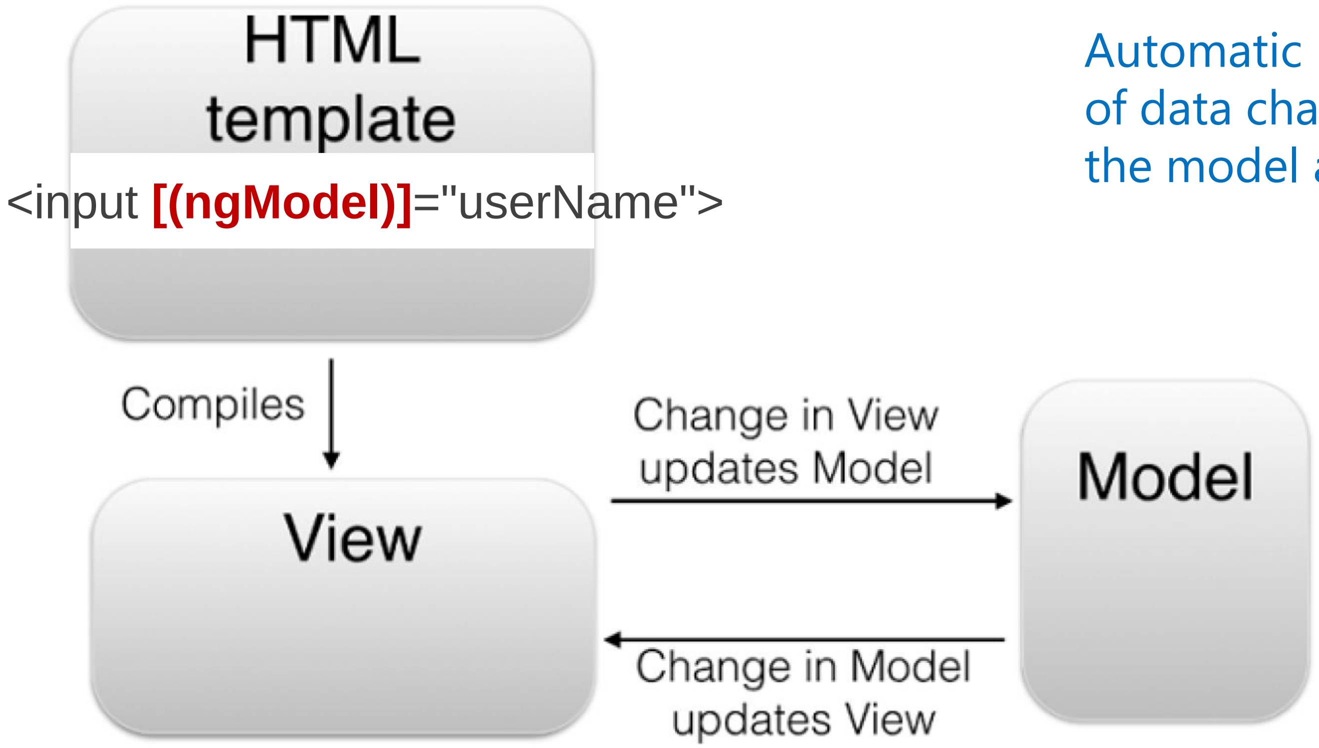
SearchComponent Example

```
@Component({
  selector: 'search-product',
  template:
     `<form>
        <div>
          <input id="prodToFind" #prod>
          <button (click)="findProduct(prod)">Find Product</button>
          Product name: {{product.name}}
        </div>
      </form>
    `
})
class SearchComponent {
   product: Product;

   findProduct(product){
    // Implementation of the click handler goes here
   }
}
```

[(ngModel)] => Two-way data binding



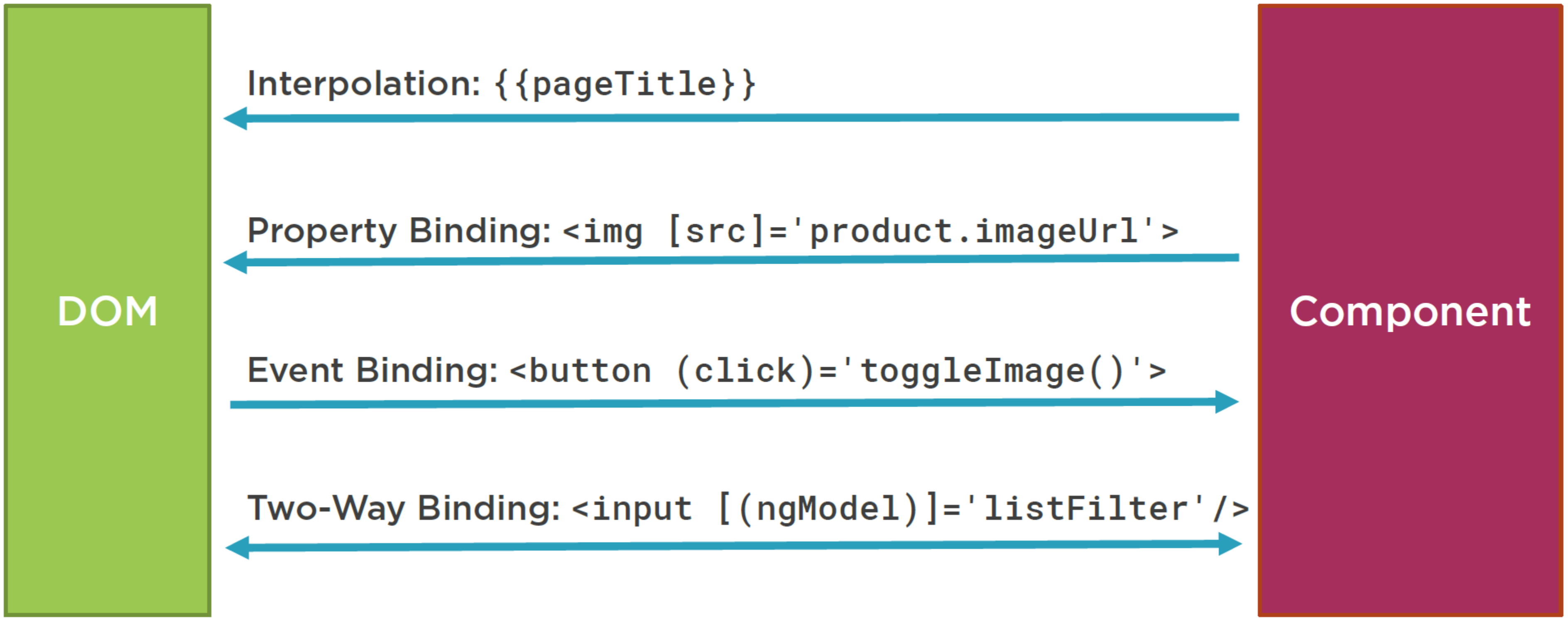
Two-way binding


HTML template
<input [(ngModel)]="userName">
Compiles
View
Change in View updates Model
Model
Change in Model updates View

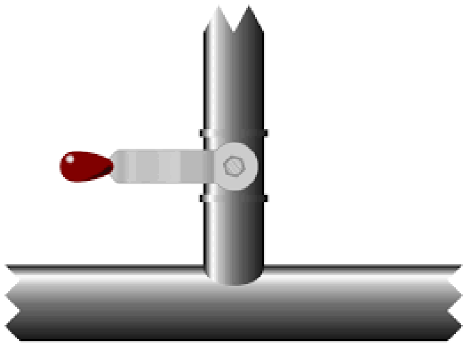
Automatic propagation of data changes between the model and the view

ngModel will display the userName in a view and it will automatically update it in case it changes in the model. If the user modifies the userName on the view then the changes are propagated to the model. Such a **two-directional** updates mechanism is called two-way data binding

Data Binding Summary

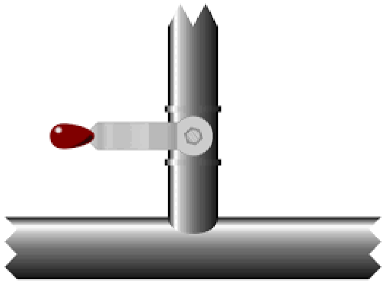

DOM
Component
Interpolation: {{pageTitle}}
Property Binding: <img [src]='product.imageUrl'>
Event Binding: <button (click)='toggleImage()'>
Two-Way Binding: <input [(ngModel)]='listFilter'/>

Pipes



Back

Pipes



- Pipes are declarative way to
  - Format / transform displayed data
  - Can create custom pipes to **filter** and **sort** data arrays
- Using pipes

```
{{ expression | pipe }}
```

- Built-in pipes
  - uppercase, lowercase
  - date
  - decimal
  - number, currency, percent
  - json , async

Example built-in pipe

```
<span>
  Today’s date is {{today | date}}
</span>
```

Today’s Date is May 1, 2017

```
<p>
   My birthday is {{ birthday | date:"dd/MM/yyyy" | uppercase }}
</p>
```

Custom pipe

```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({ name: 'double'  })
class DoublePipe implements PipeTransform {
  transform(value, args) { return value * 2; }
}

@Component({
  template: '{{ 10 | double}}'
})
class CustomComponent {}
```

Resources

- Cheat Sheet https://angular.io/cheatsheet
- Guide https://angular.io/docs/ts/latest/guide/
- Tour of Heroes tutorial

https://angular.io/docs/ts/latest/guide/learning-angular.html

- Angular 5 Education Resources

https://github.com/AngularClass/awesome-angular

- Angular 2 Development with TypeScript (free book via QU Library eResources)

https://www.safaribooksonline.com/library/view/angular-2-development/9781617293122/

https://www.ng-book.com/2/