A⁴ ANGULAR

by Google

Outline

- What is Angular and why should you care!

- Single Page Application (SPA)

- Angular Architecture

- Angular features:

  - Components

  - Directives

  - Forms

  - Routing

  - Angular Material 2

What is ![Angular by Google] ?

- 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 #1?

- Angular appears to be winning the JavaScript framework battle
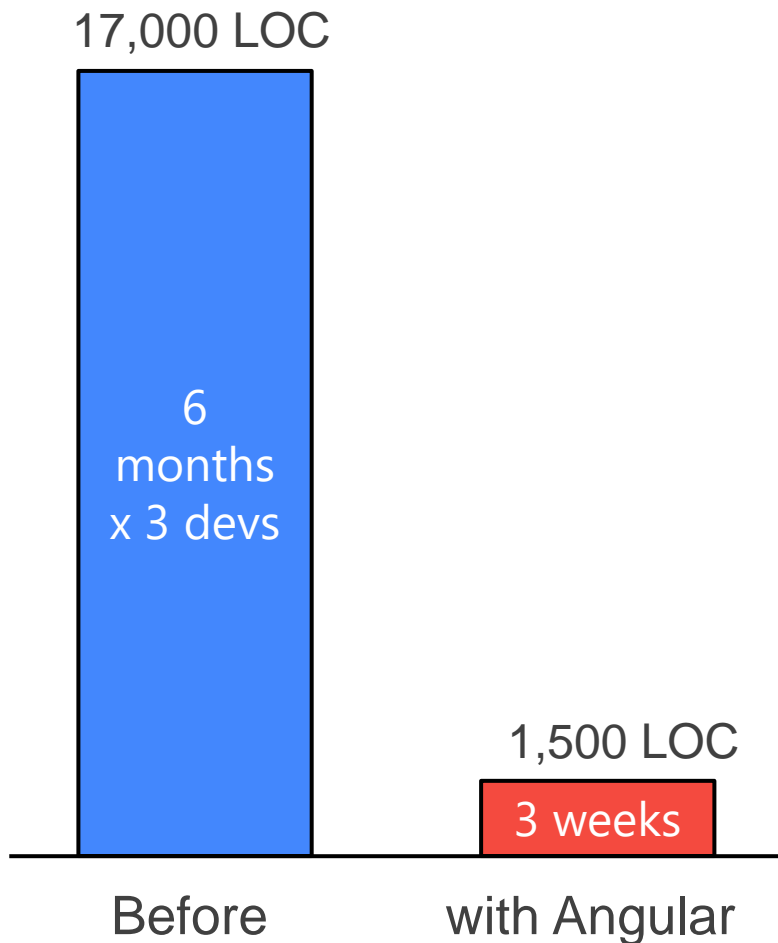
- React is a strong competitor!
https://facebook.github.io/react/

Direct DOM Manipulation

<form> </form> To Submit to server

Component based UI

Google feedback

17,000 LOC

6
months
x 3 devs

1,500 LOC

3 weeks

Before

with Angular

f_x | =sum(A2+B2)

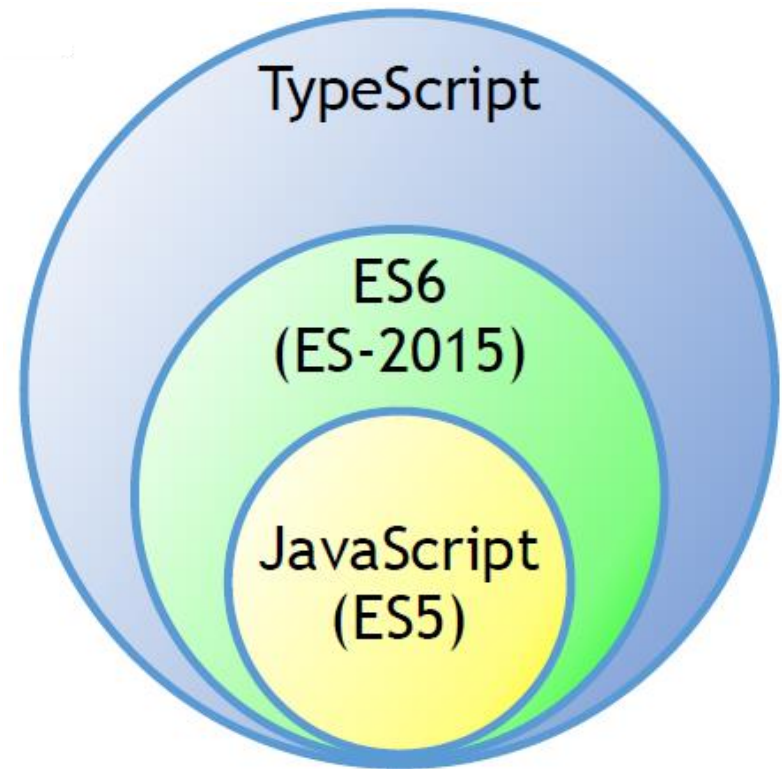|	A	B	C	D
1	5	2	7	
2	4	6	=sum(A2+B2)	
3				
4				
5				
6				
7				

Angular Features

- Component-based (not MVC)

- Dependency Injection

- Router

- Can write apps in **TypeScript**

- UI components: Angular Material 2

- The rendering engine

TypeScript = JavaScript + Types
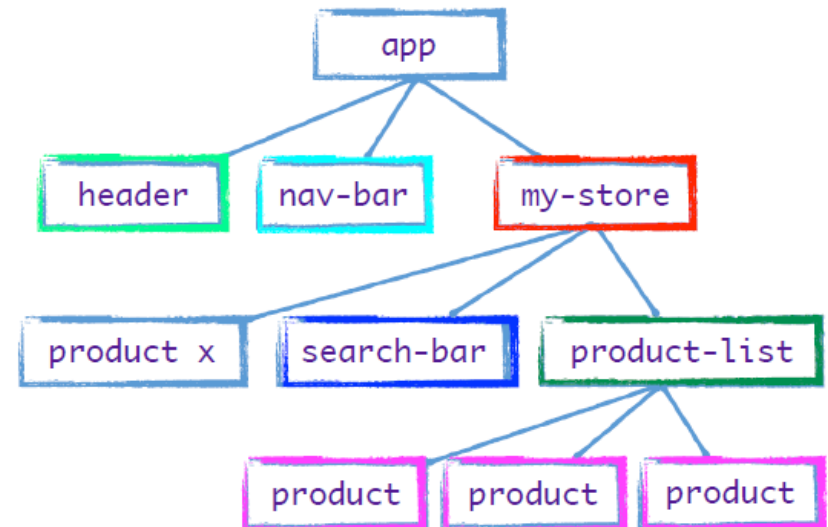
• type checking at dev time

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

• code help - intellisense

• @decorators

• and more…

An app is a tree of components

```html
<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>
```
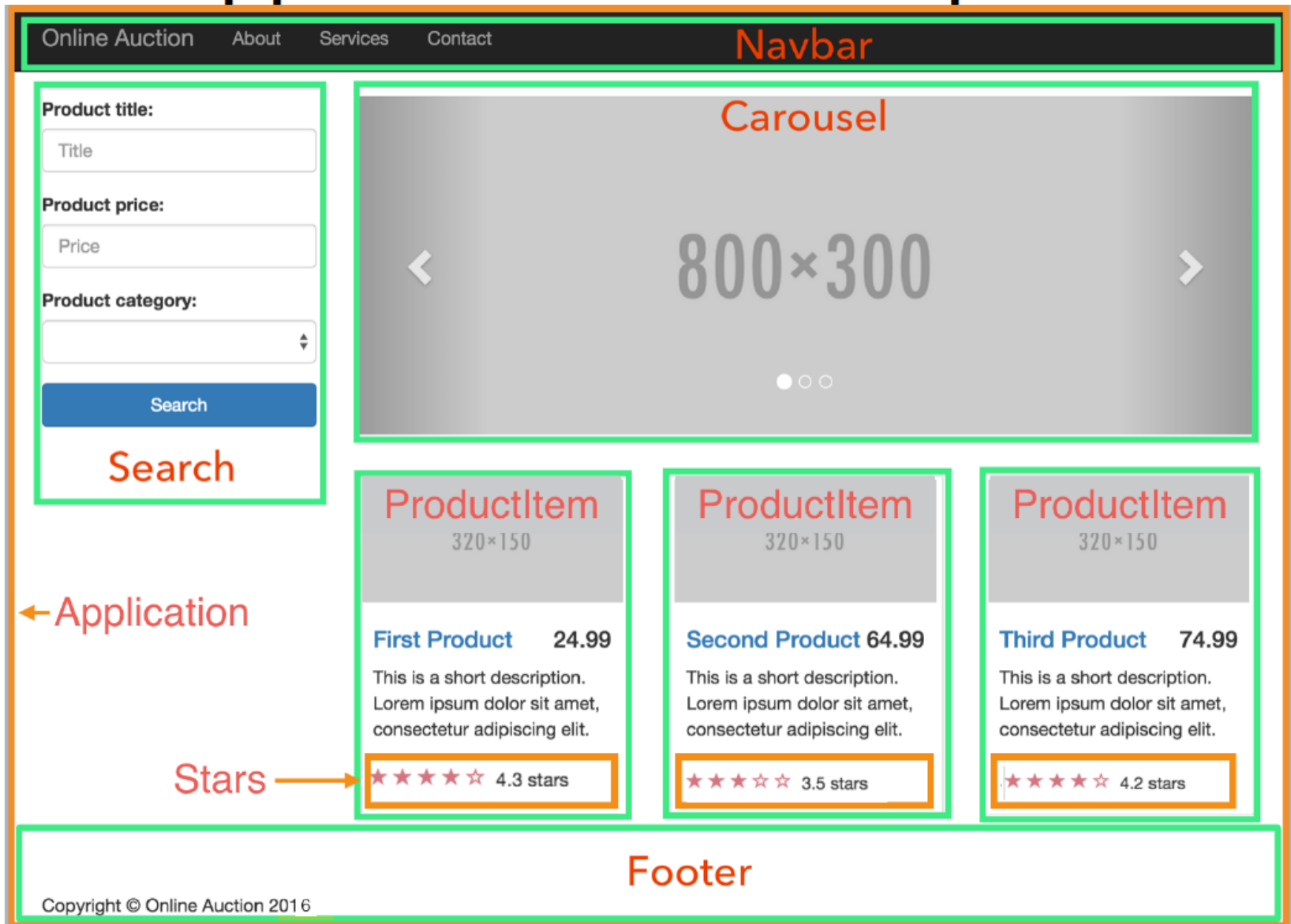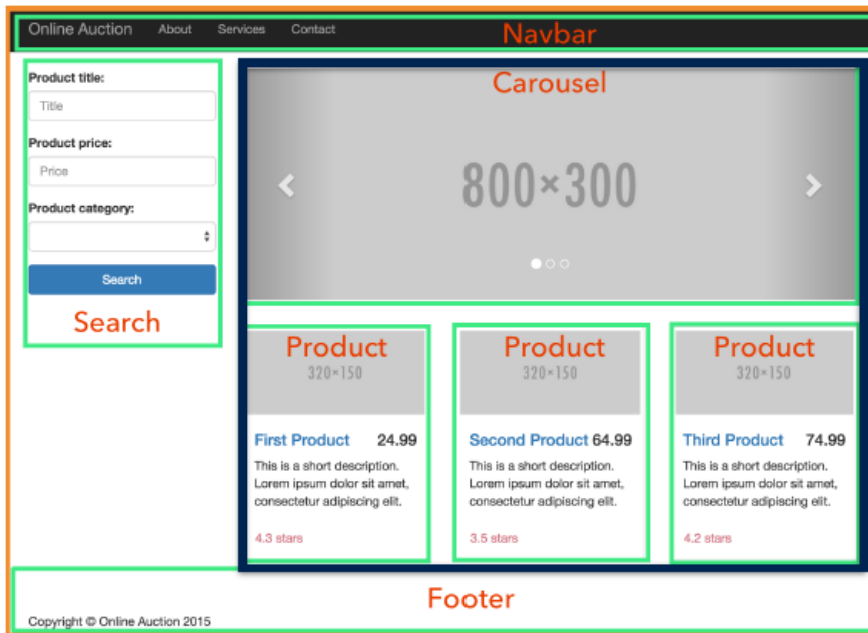
An app is a tree of components

An app is a tree of components


```html
<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>
```

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

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

  constructor(private productService: ProductService) {   ← TypeScript
    this.products = this.productService.getProducts();
  }
}
```
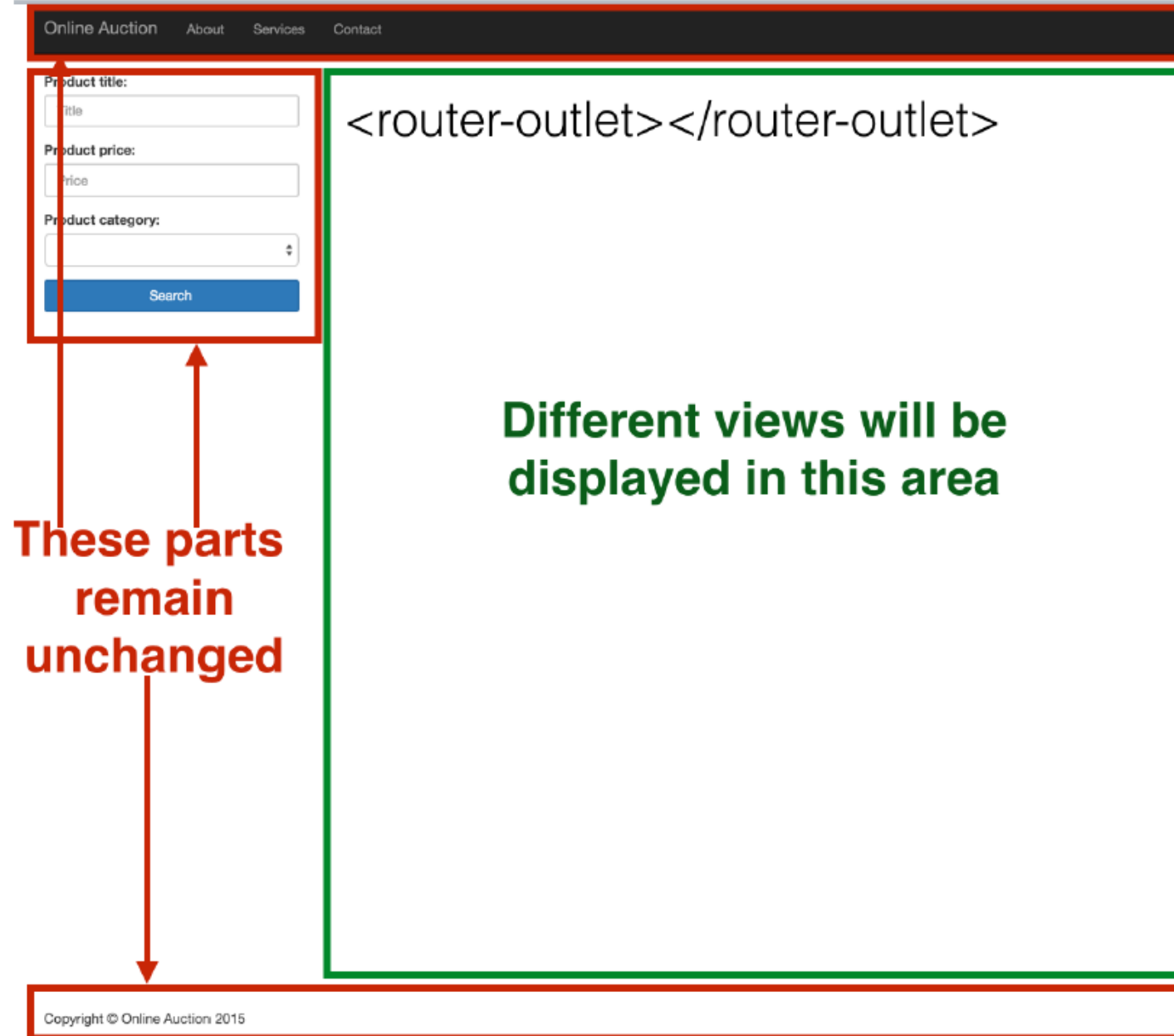
Single Page App

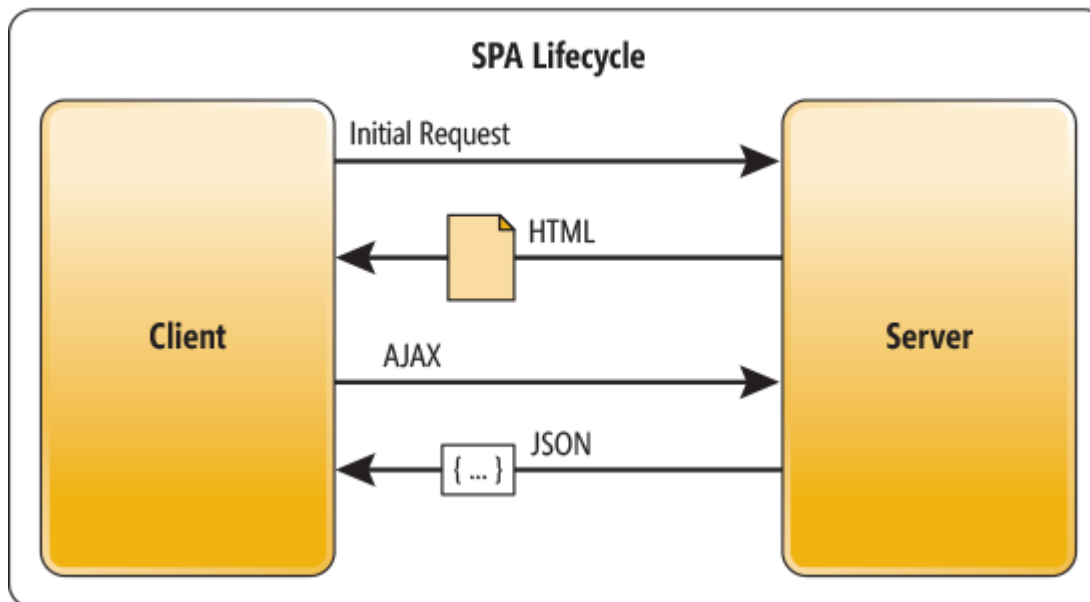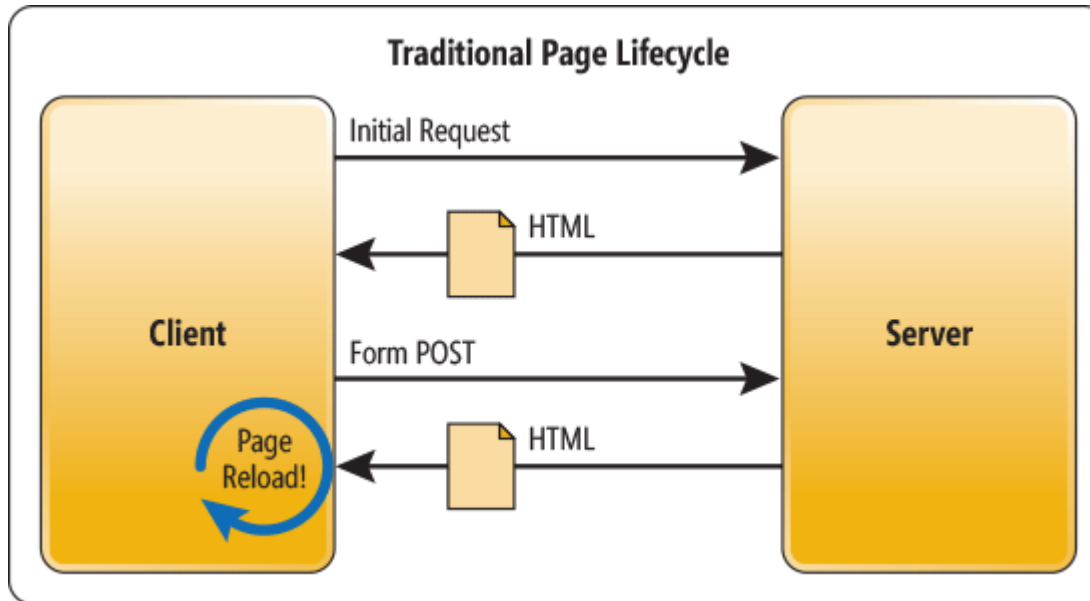Different views will be displayed in this area

These parts remain unchanged

Single Page Application (SPA)

Traditional vs. SPA Lifecycle

Role of Client and Server in SPA

Client Side

Major Responsibilities:
- Data Access via the API
- UI Rendering
- Client Side Routing

Session Management

AJAX

Server Side

Web Service API (REST)

Core Business Logic

Data Access Layer + Databases

Security

Logging
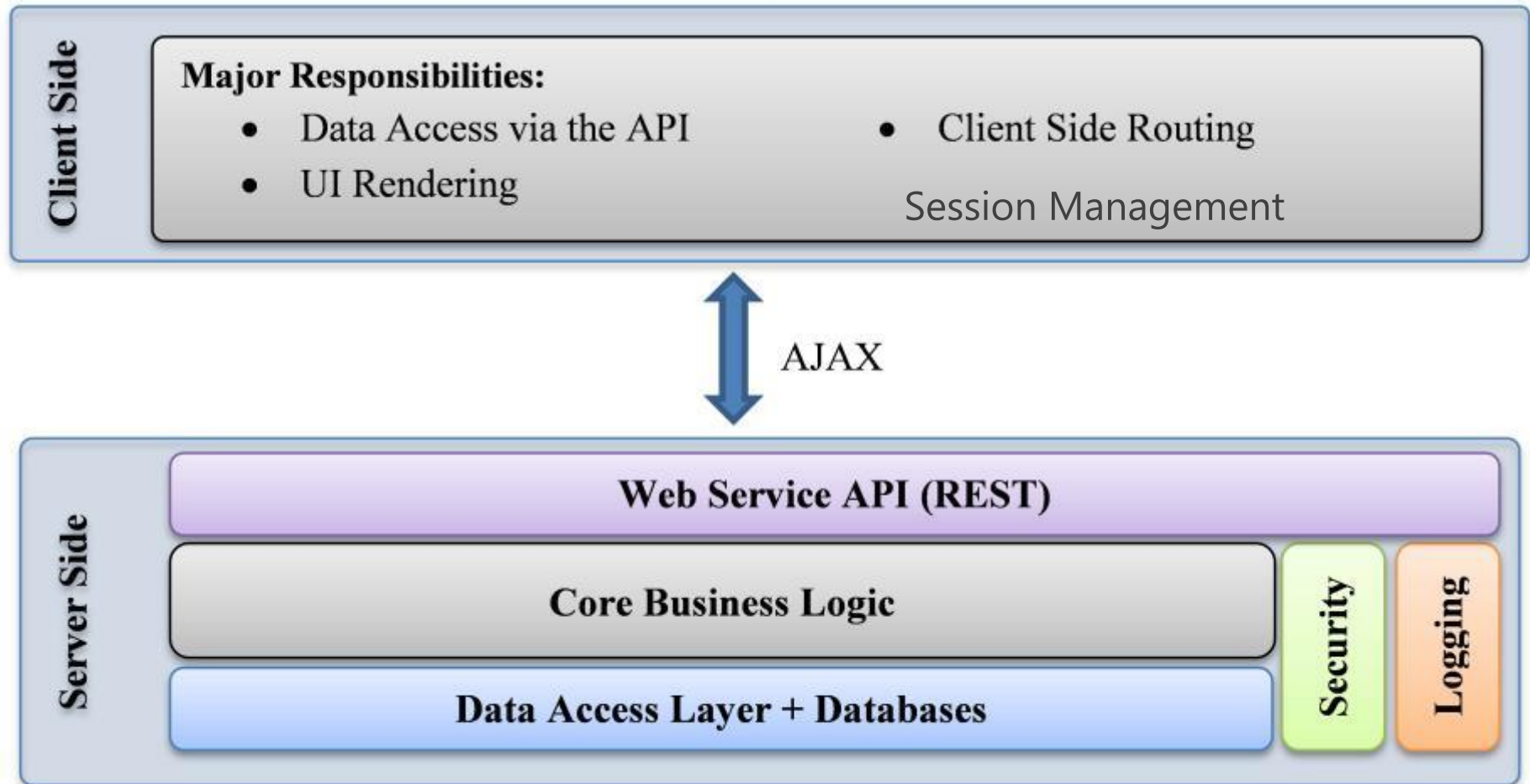
Benefits of a Single Page App
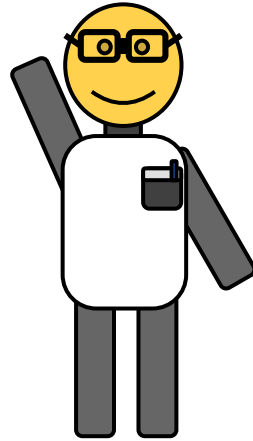
- **Better User experience**

- 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

Architecture of an Angular app

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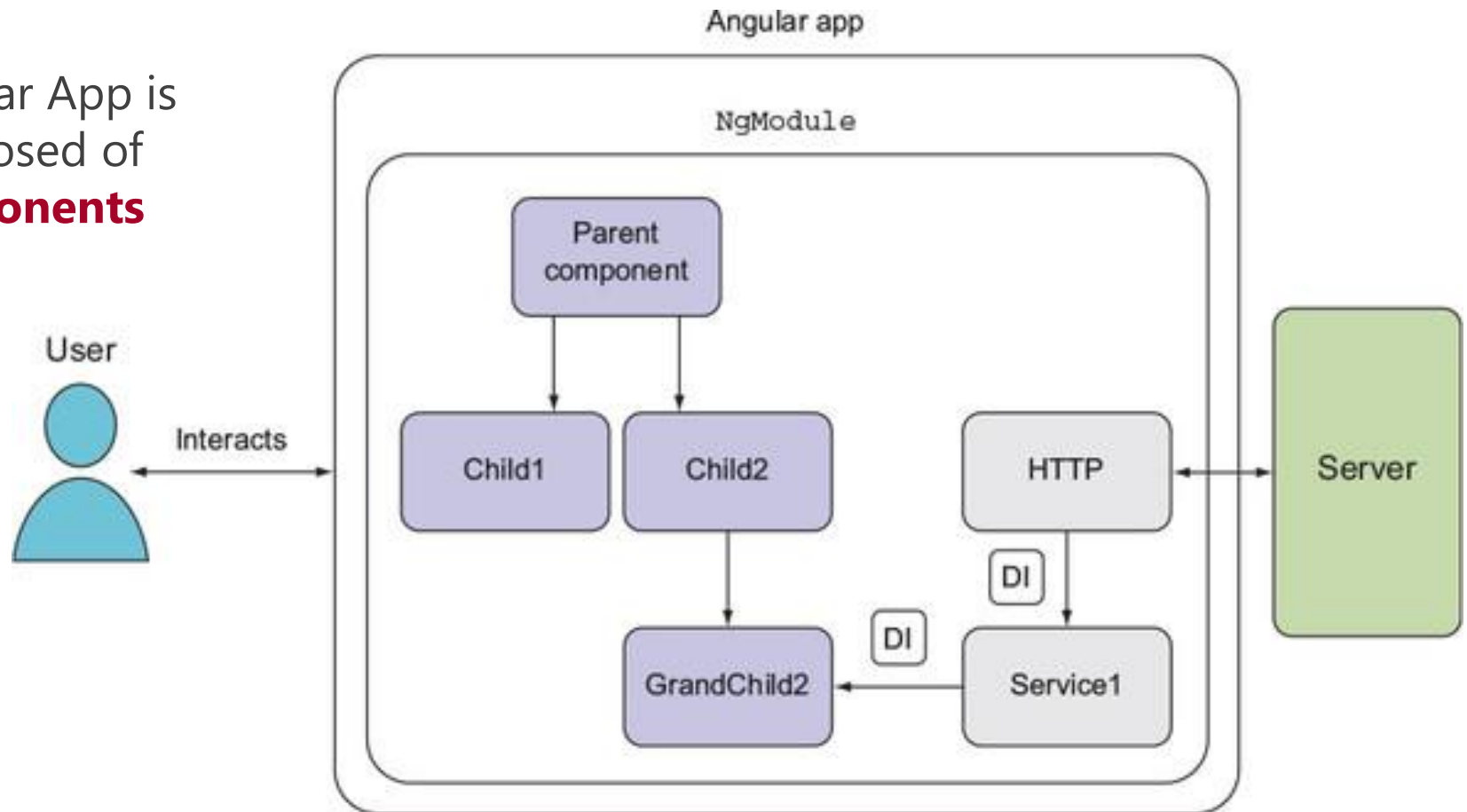
The figure shows a high-level diagram of a sample Angular application that consists of four components and two services; all of them are packaged inside a module. Angular's Dependency Injection (DI) module injects the Http service into Service1, which in turn is injected into the GrandChild2 component.

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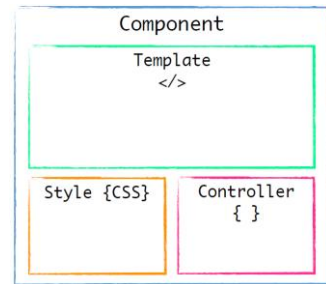
```typescript
import { Component } from '@angular/core';

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

export class HelloComponent {

  title = 'Hello World!';

}
```
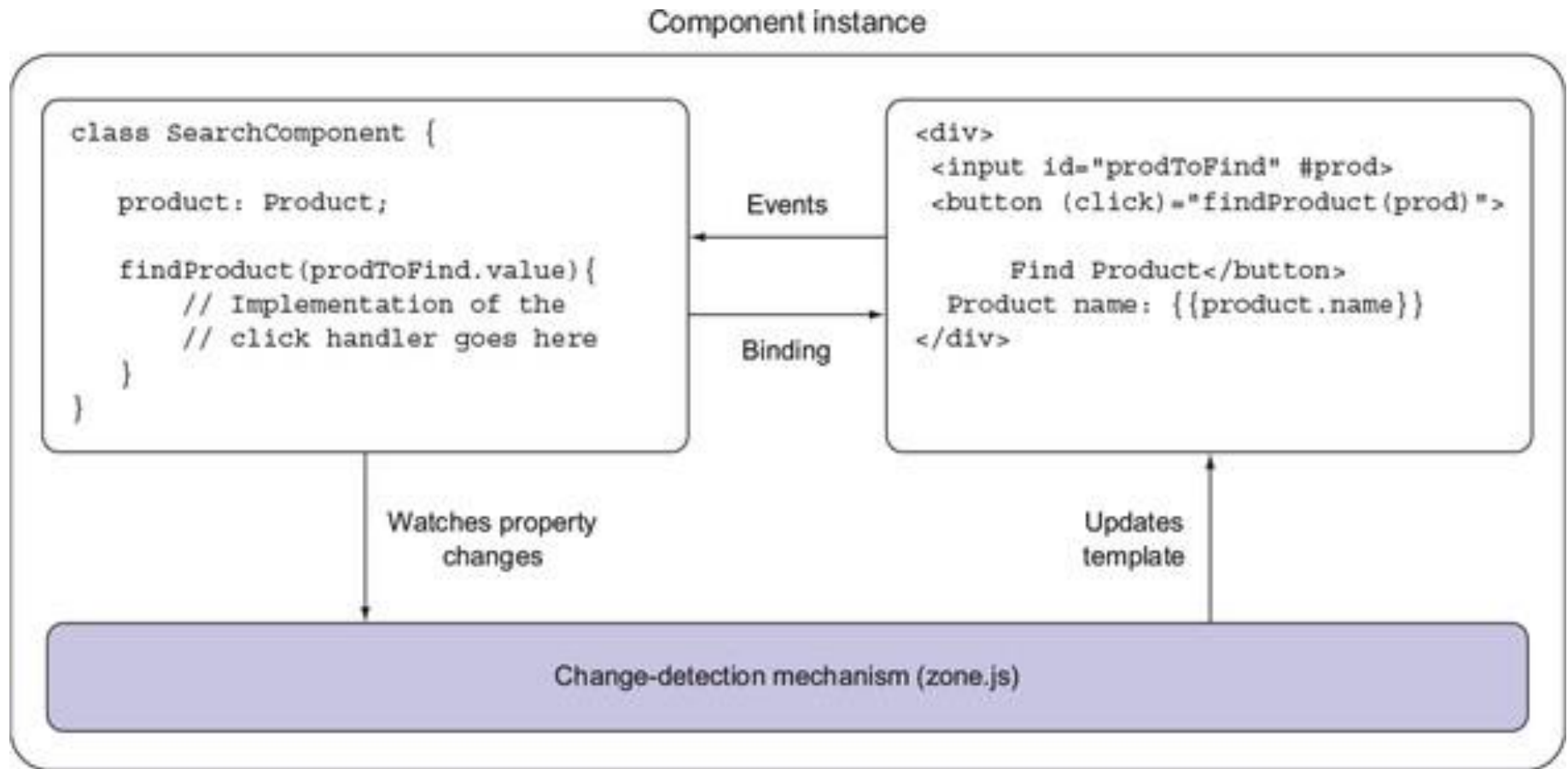
Somewhere in your app

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

Component internals

Component instance

```
class SearchComponent {

    product: Product;

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

Events

Binding

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

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

Watches property changes

Updates template

Change-detection mechanism (zone.js)
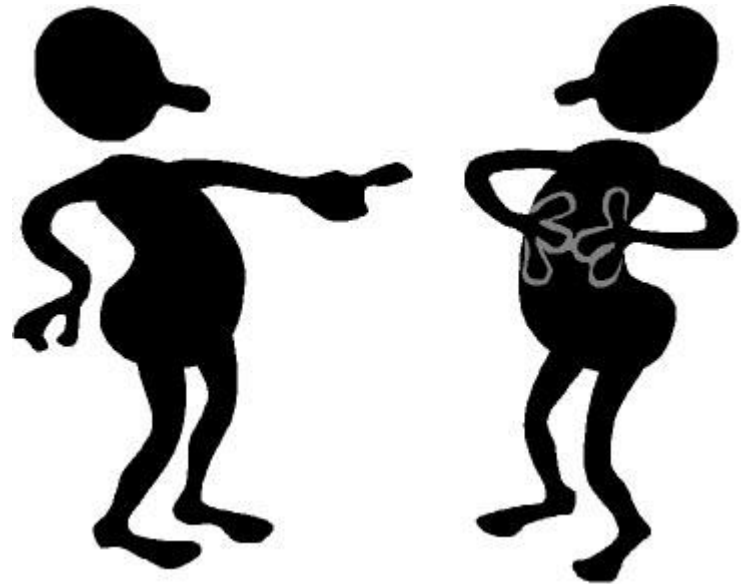
Component is a unit encapsulating the presentation logic and the auto-generated change detector

Directives

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

HTML Template **+** Data **=** DOM 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

@Input properties


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

    @Input() quantity: number;

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

Child

Parent

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

@Output properties

Child

```
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);
    }
}
```

Parent

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

Another Example

```typescript
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();

}
```
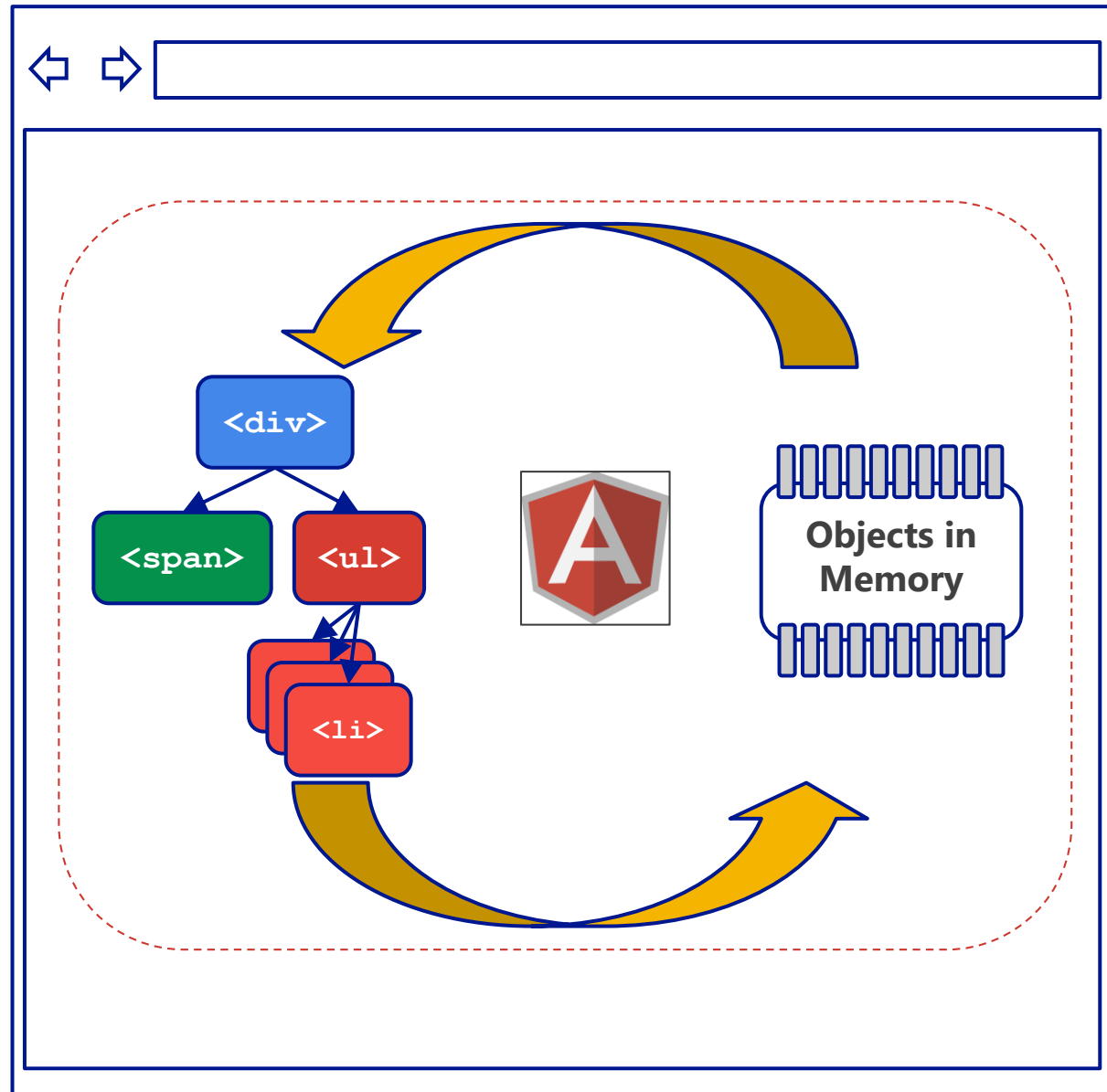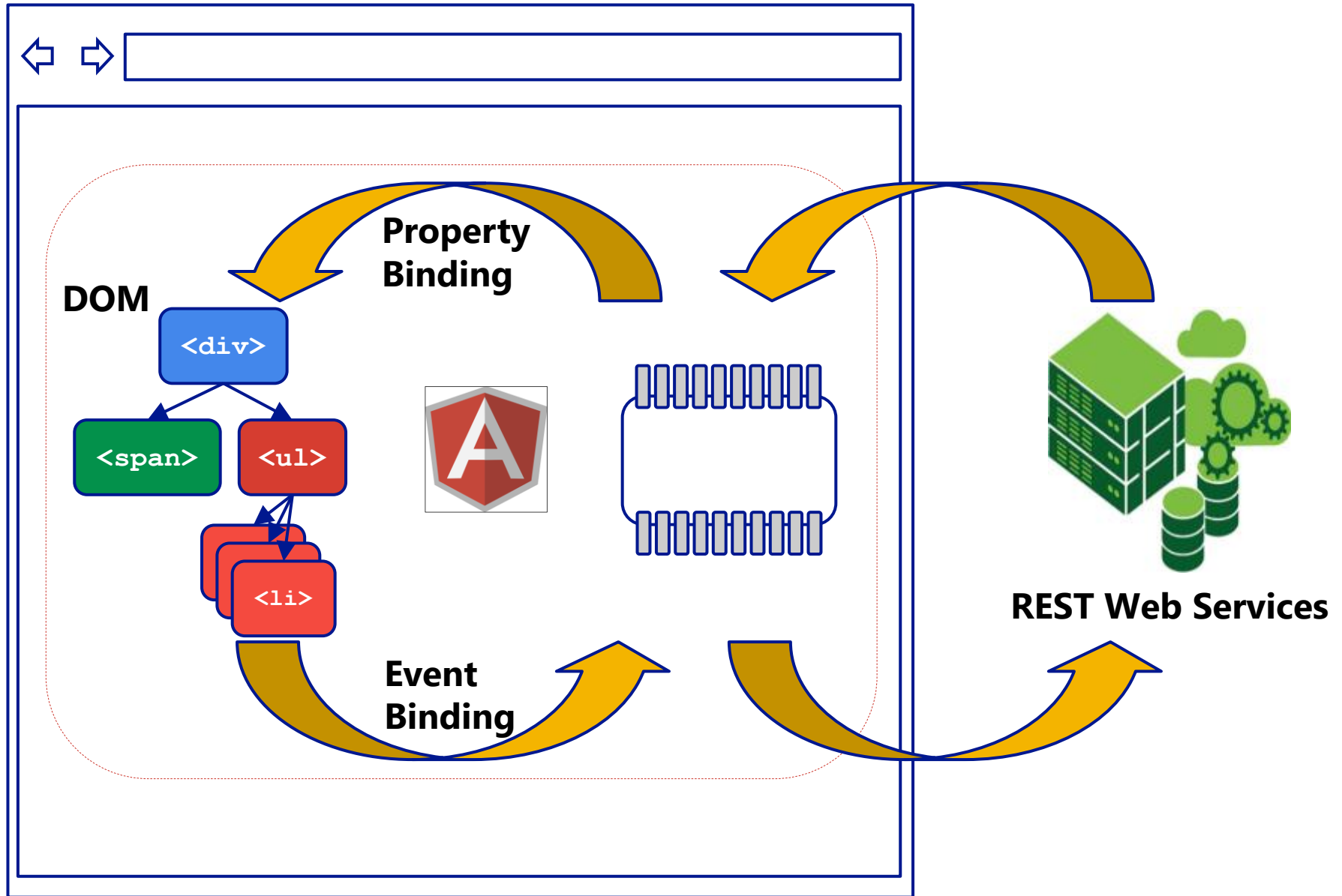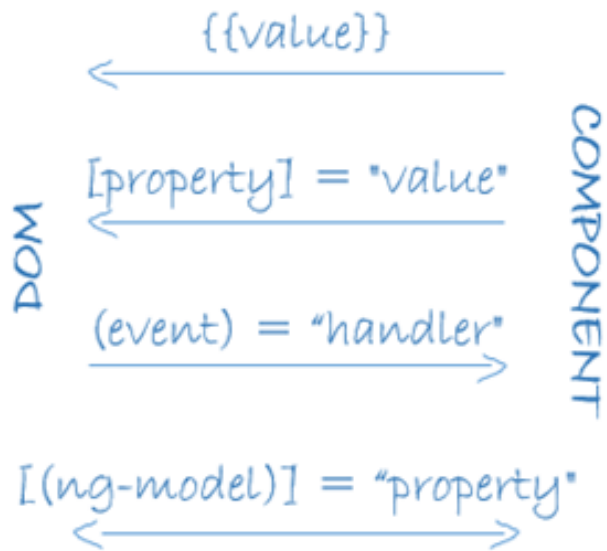
Binding

Binding - big picture

DOM

`<div>`

`<span>` `<ul>`

`<li>`

Property Binding

Event Binding

REST Web Services

Things you can bind to

{{value}}

[property] = "value"

(event) = "handler"

[(ng-model)] = "property"

DOM

COMPONENT

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)"/>
```
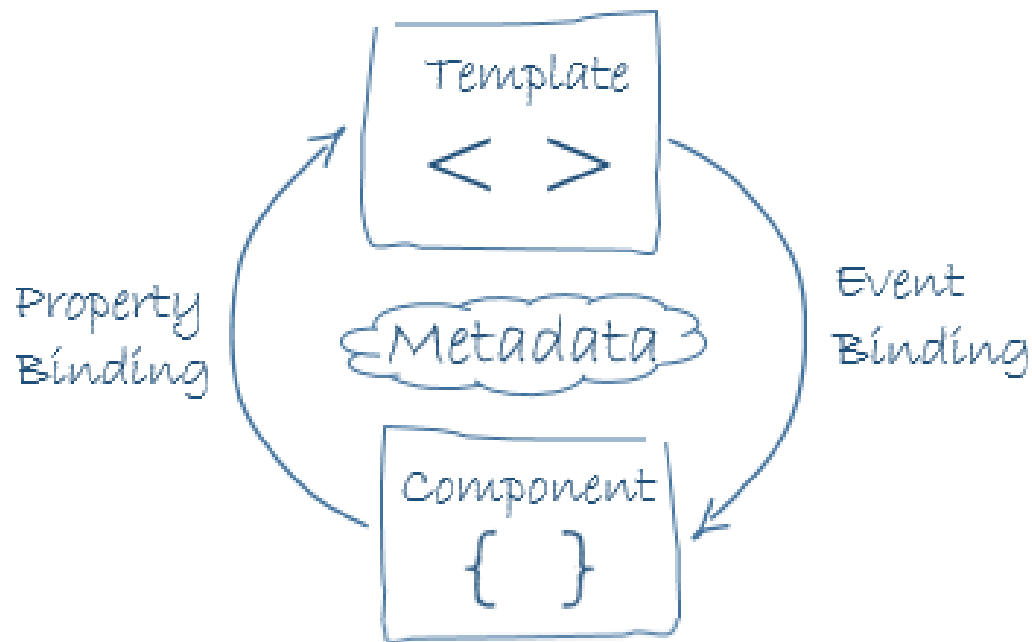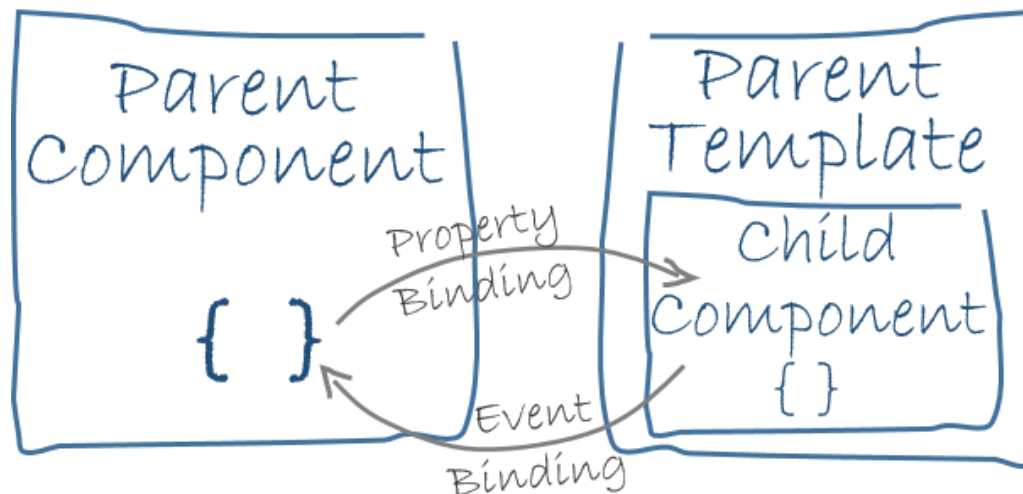
Inputs & Outputs - ngModel

Hello!

Hello!

```html
<h1>{{ product.title }}</h1>
<input [(ngModel)]="product.title">
```

Communication between a template and its component

Communication between parent and child components

Example

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

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

> Calls a function defined in the component class

```html
<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
    }
}
```
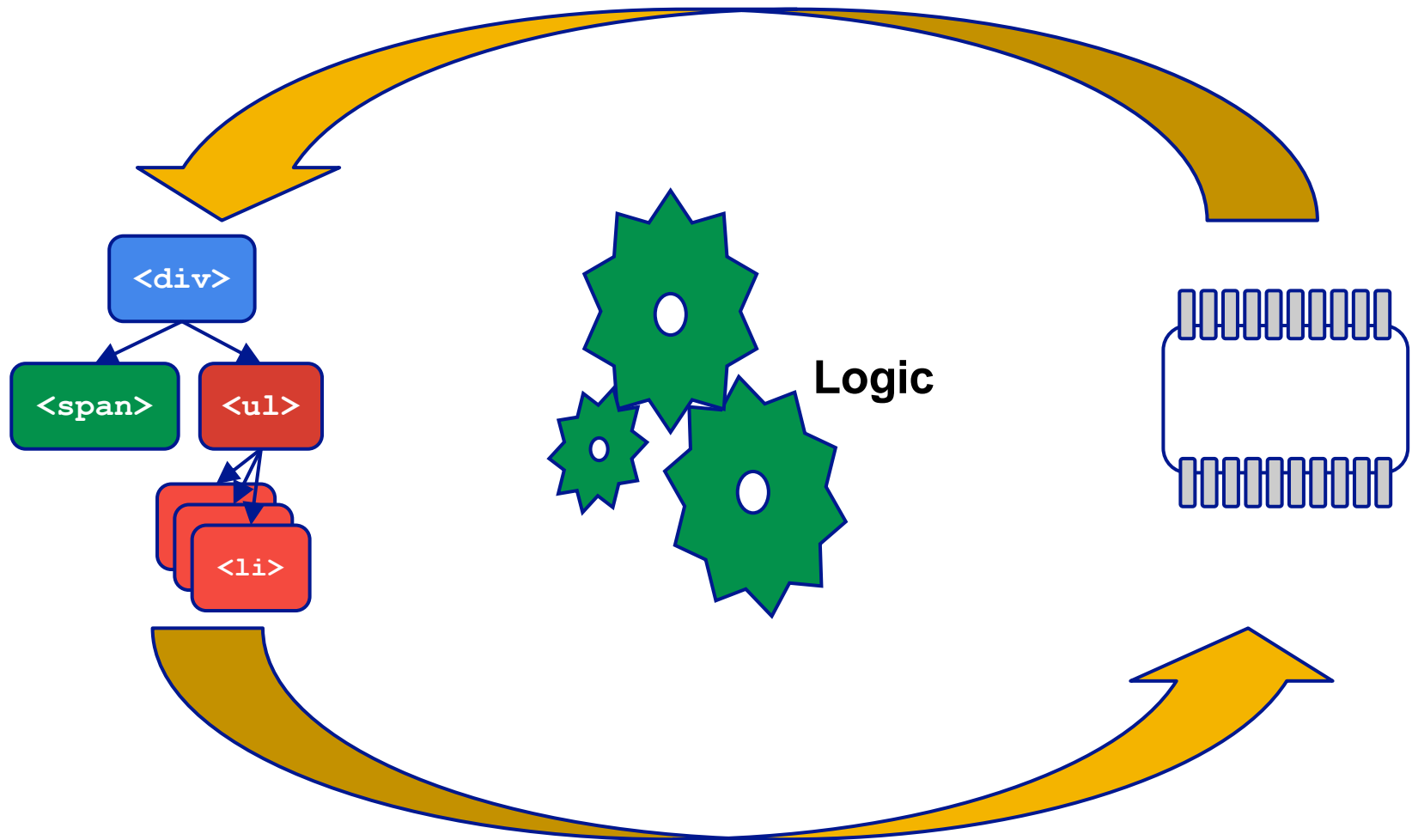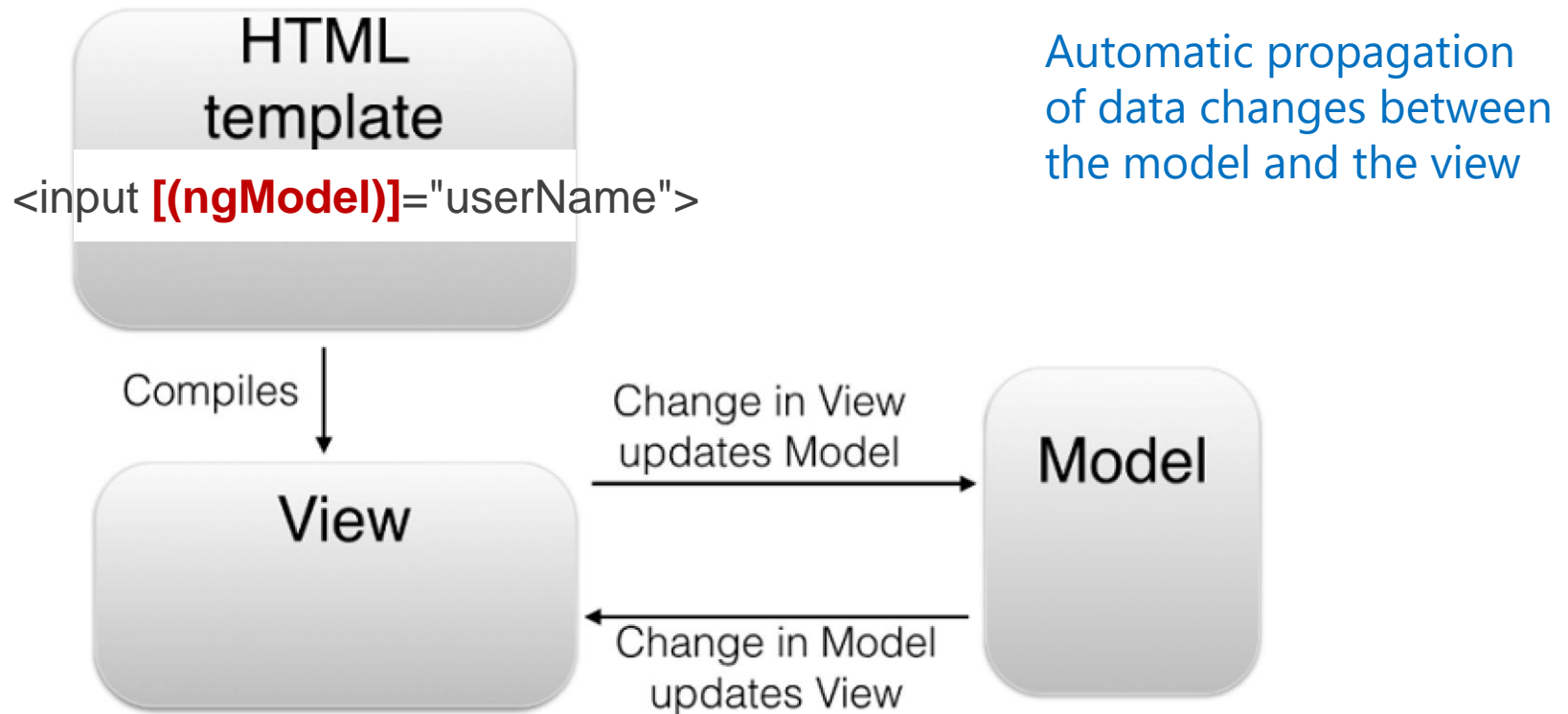
**[(ngModel)] => Two-way data binding**

Two-way binding


```
HTML
template

<input [(ngModel)]="userName">
```

Automatic propagation of data changes between the model and the view

Compiles → View

Change in View updates Model →
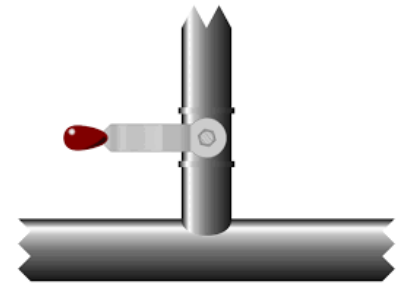
Model

Change in Model updates 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

Pipes
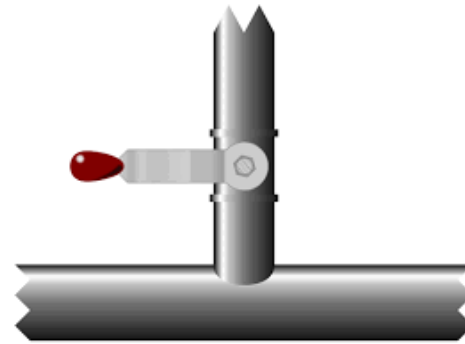
Pipes

- 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 {}
```

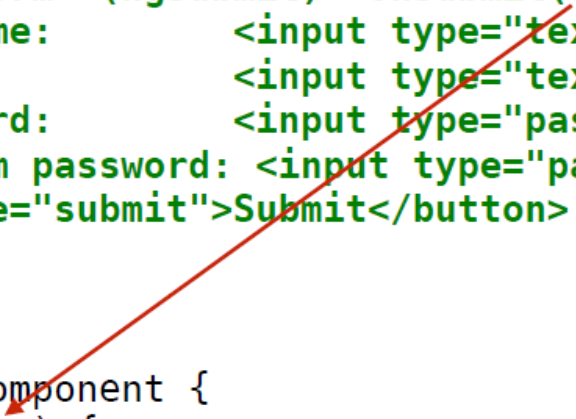# Forms

- Template-driven forms

- Reactive forms

- Form validation

A template-driven form

```
@Component({
  selector: 'app-root',
  template: `
    <form #f="ngForm" (ngSubmit)="onSubmit(f.value)">
      <div>Username:        <input type="text"     name="username" ngModel></div>
      <div>SSN:             <input type="text"     name="ssn"      ngModel></div>
      <div>Password:        <input type="password" name="password" ngModel></div>
      <div>Confirm password: <input type="password" name="pconfirm" ngModel></div>
      <button type="submit">Submit</button>
    </form>
  `
})
export class AppComponent {
  onSubmit(formData) {
    console.log(formData);
  }
}
```

Form Validation

- Default validation

  - Required – makes property required

  - ngPattern – regex pattern

  - Form Properties – valid / invalid
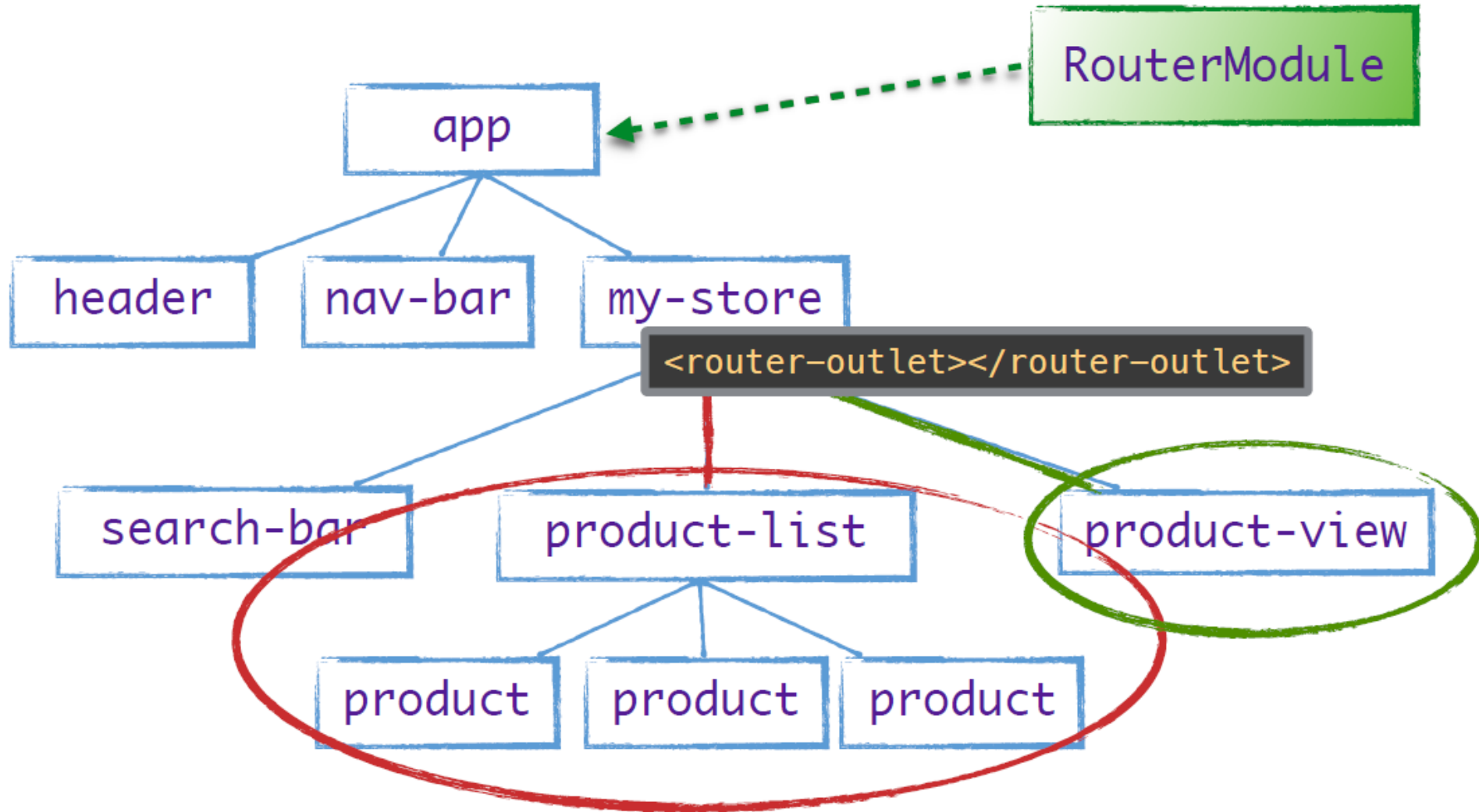
  - CSS Classes – classes can be styled

Routing and views

Routing

Routes

- Implement client-side navigation for SPA:

  - Configure routes, map them to the corresponding components in a declarative way.

- Defines the app navigation in *@RouteConfig* *annotation*

  - On URL change => load a particular component

Angular Router

- **RouterOutlet** – a directive that serves as a placeholder within your Web page where the router should render the component

- **RouteModule (part of @NgModule)** – to map URLs to components to be rendered inside the *<router-outlet></router-outlet>* area

- **RouteParams** – a service for passing parameter to a component rendered by the router

- **RouterLink** – a directive to declare a link to a view and may contain optional parameters to be passed to the component

Router Programming Steps (1 of 2)

1. Configure the router on the root component level to map the URL fragments to the corresponding named components

- If some of the components expect to receive input values, you can use route **params**

- Needs

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

```
RouterModule.forRoot([
  { path: '/', component: AppComponent },
  { path: '/hero/:id', component: HeroFormComponent }
])
```

Router Programming Steps (2 of 2)

2. Add **<router-outlet></router-outlet>** to the view to specify where the router will render the component

3. Add the HTML anchor tags with **[routerLink]** attribute, so when the user clicks on the link the router will render the corresponding component.

- Think of **[routerLink]** as href attribute of anchor tag

```
<a [routerLink]="['/']">Home</a>
<a [routerLink]="['/hero', {id: 1234}]">
Hero</a>
<router-outlet></router-outlet>
```

Route Parameters

- Route parameters start with "**:**"

```
RouterModule.forRoot([
    { path: '/product/:id', component: ProductDetailComponent  }
])
```

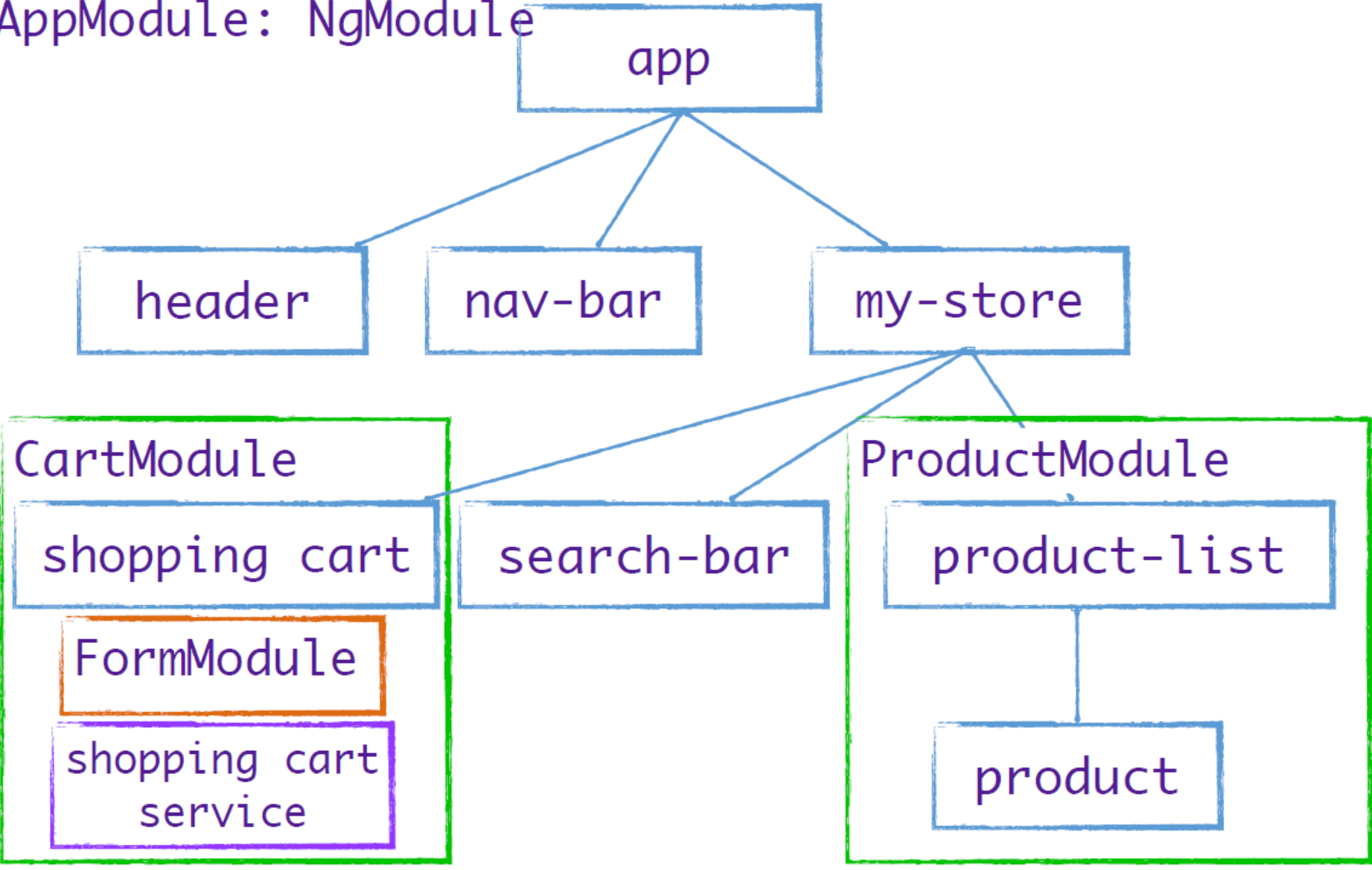- You can use them later in ViewModel constructor

```
export class ProductDetailComponent {
    productID:string;

    constructor(params : RouteParams) {
        this.productID = params.get('id');
    }
}
```

Angular Modules

Help organize an application into cohesive blocks of functionality

Component Tree

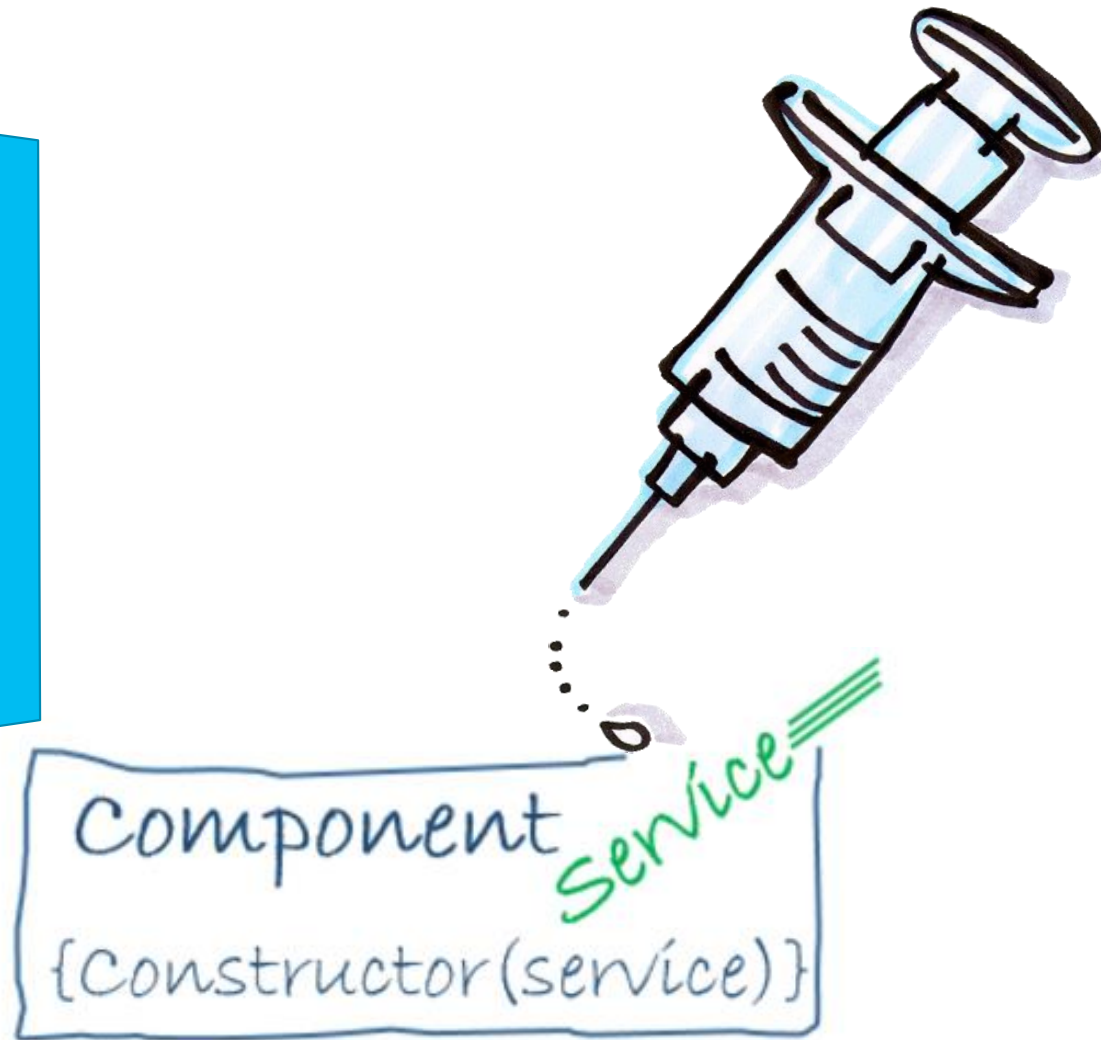AppModule: NgModule

app

header   nav-bar   my-store

CartModule

shopping cart   search-bar

FormModule

shopping cart service

ProductModule

product-list

product

Angular Module

Dependency Injection
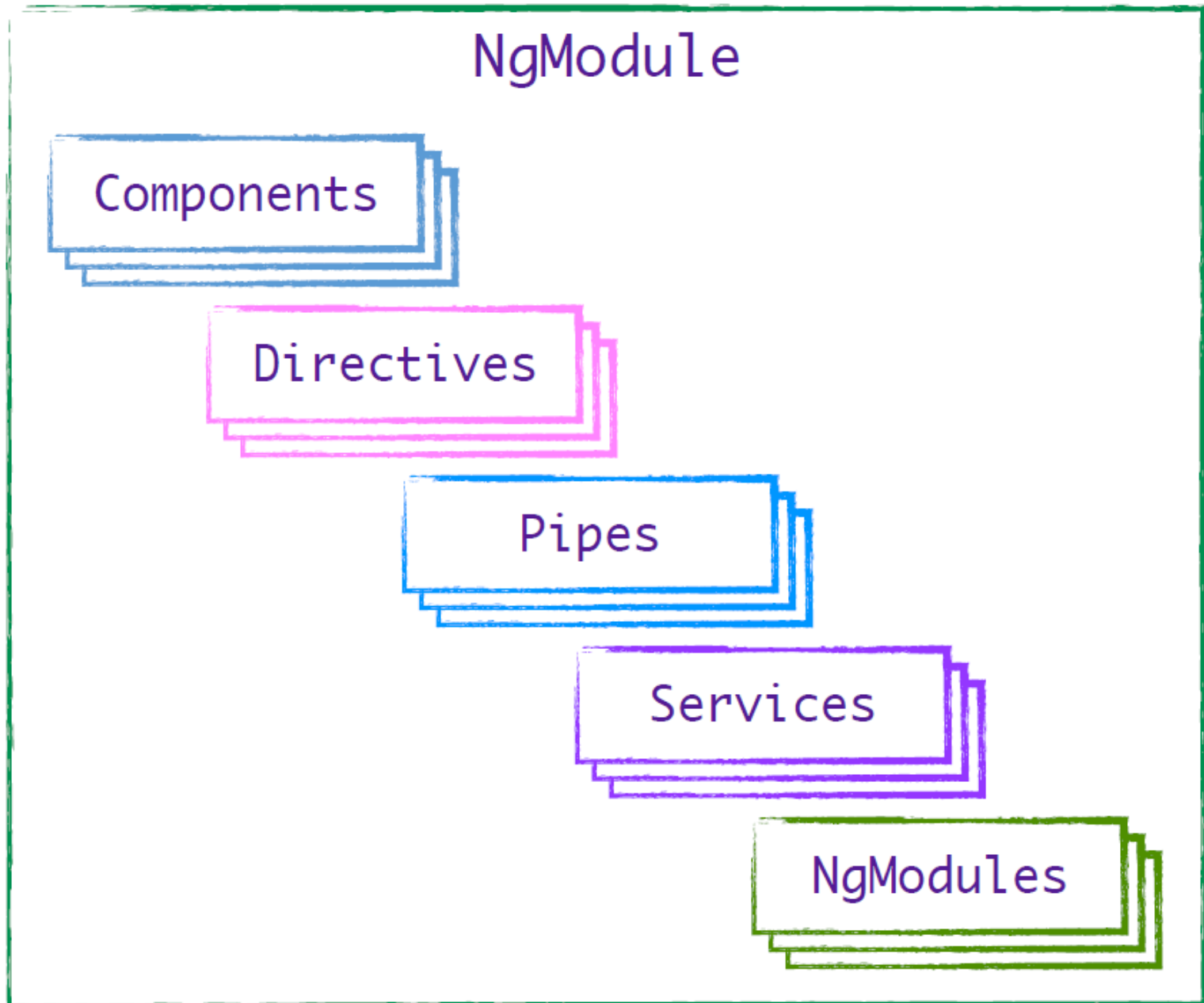
component service
{Constructor(service)}

Dependency Injection

- An Angular Service is a JavaScript class than can be injected and made available to the entire application or to a particular component

- Angular injects **services** into components via constructors

```
@Injectable()
export class ProductService{

  getProduct(): Product {

    // An HTTP request can go here

    return new Product( 0, "iPhone 7", 249.99, "The latest iPhone, 7-inch screen");
  }
}
```

Injecting the ProductService

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

@Component({
  selector: 'di-product-page',
  template: `<div>
              <h1>Product Details</h1>
              <h2>Title: {{product.title}}</h2>
              <h2>Description: {{product.description}}</h2>
              <h2>Price: \${{product.price}}</h2>
            </div>`,
  providers:[ProductService]
})

export class ProductComponent {
  product: Product;

  constructor( productService: ProductService ) {
    this.product = productService.getProduct();
  }
}
```

Injection

Dependency Injection

- Dependency Injection is a design pattern that inverts the way of creating objects your code depends on

- Dependency Injection is a way to supply a instance of a class to a component

- Instead of explicitly creating object instances (e.g. with new) the framework will create and inject them into your code (i.e.,)

- Angular comes with a dependency injection module

- You can inject dependencies into the component only via its constructor

Dependency Injection

```
@Component({
  selector: 'search-product',
  viewProvider: [ProductService],
  template:[<div>...<div>]
})
class SearchComponent {
  products: Array<Product> = [];

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

- Inject the **ProductService** object into the **SearchComponent** by declaring it as a constructor argument
- Angular will instantiate the ProductService and provide its reference to the SearchComponent.

Summary

Angular introduces many innovations:

- Component Router

- Dependency Injection (DI)

- Client-side templating

- Solid tooling thanks to Typescript + excellent testing support

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 4 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/2n/