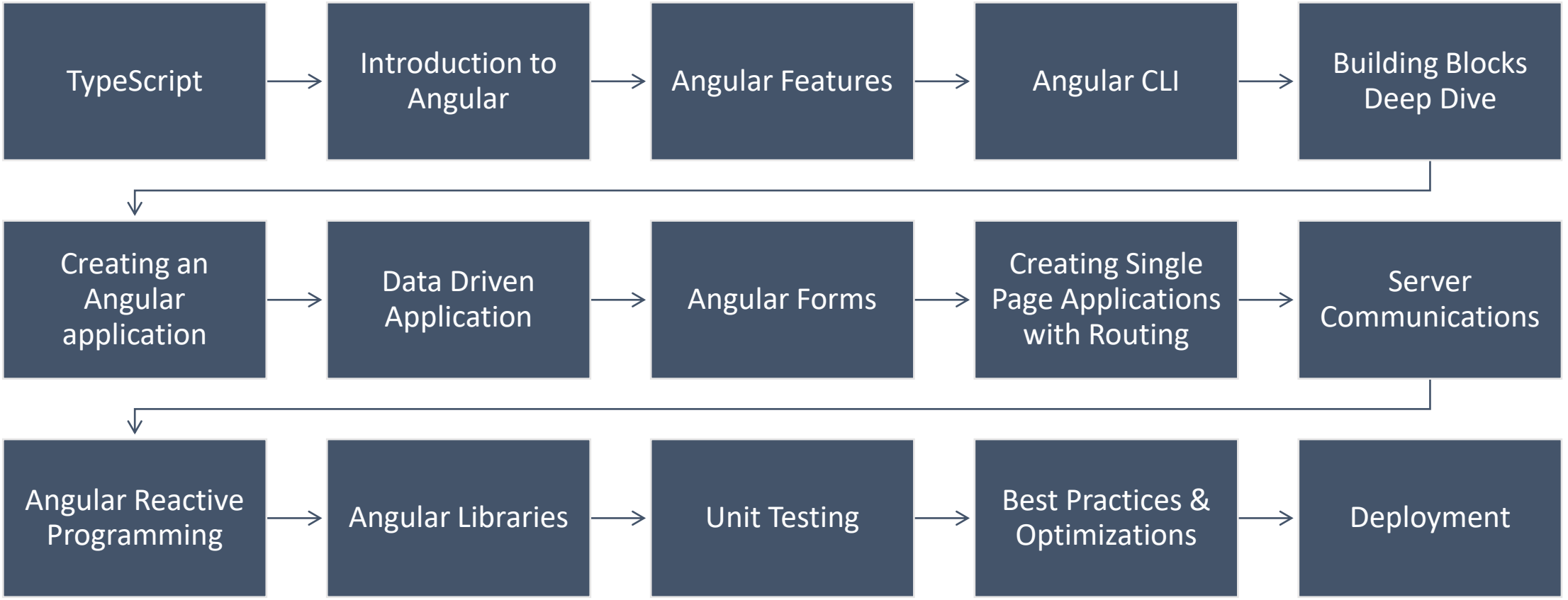
ANGULAR

ANIL JOSEPH

Timmins Training Consulting

Agenda


TypeScript
Introduction to Angular
Angular Features
Angular CLI
Building Blocks Deep Dive
Creating an Angular application
Data Driven Application
Angular Forms
Creating Single Page Applications with Routing
Server Communications
Angular Reactive Programming
Angular Libraries
Unit Testing
Best Practices & Optimizations
Deployment

Introduction

Anil Joseph

- Over 20 years of experience in Training and Development
- Technologies
  - C++
  - Java
  - .NET and .NET Core
  - Node, Node Express and other frameworks
  - UI Technologies: React, Angular, ExtJS, jQuery, Knockout, Redux etc
  - Mobile: Native Android, Xamarin and React Native
- Worked on numerous projects
- Conducted training for corporates(700+)

Software

NodeJs and NPM

- Version 16 or higher

Angular CLI

- npm install @angular/cli -g

Visual Studio Code

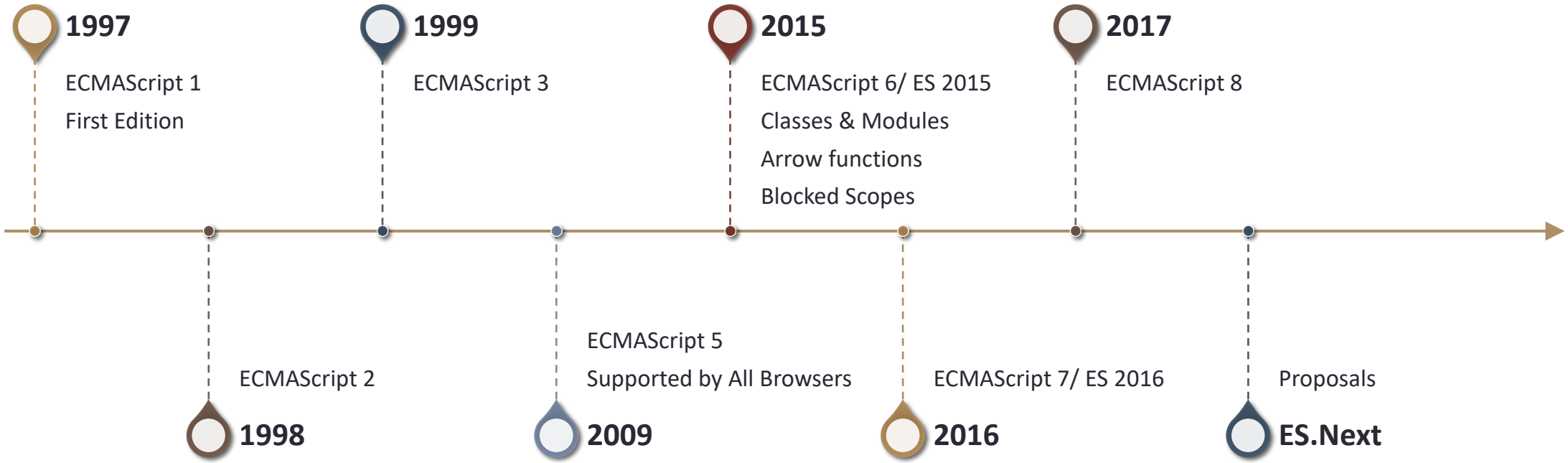
Browsers

- Chrome(Chromium) and any other browsers

JavaScript

- JavaScript (JS) is an interpreted programming language.
- It's a dynamic language.
- Supports object-oriented programming.
- All Web browsers have a JavaScript Engine. In the browser JavaScript is used
  - For executing Client-side scripts to interact with the user.
  - Alter the document content that is displayed.
  - Control the browser.
  - Communicate asynchronously.
- Server-side JavaScript is available with Nodejs
- JavaScript was developed by Brendan Eich at Netscape.
- Released in September 1995

ECMAScript Versions



TypeScript

TypeScript

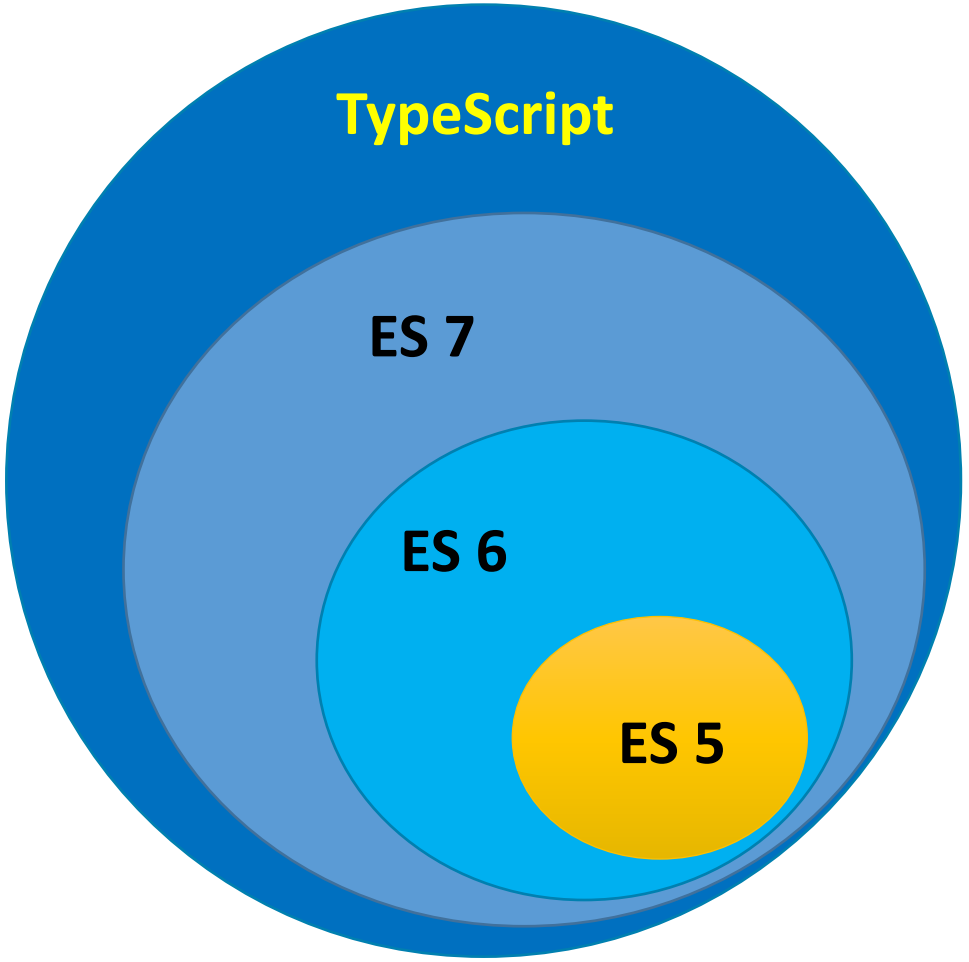
TypeScript is programming language developed and maintained by Microsoft.

TypeScript is a typed superset of JavaScript.

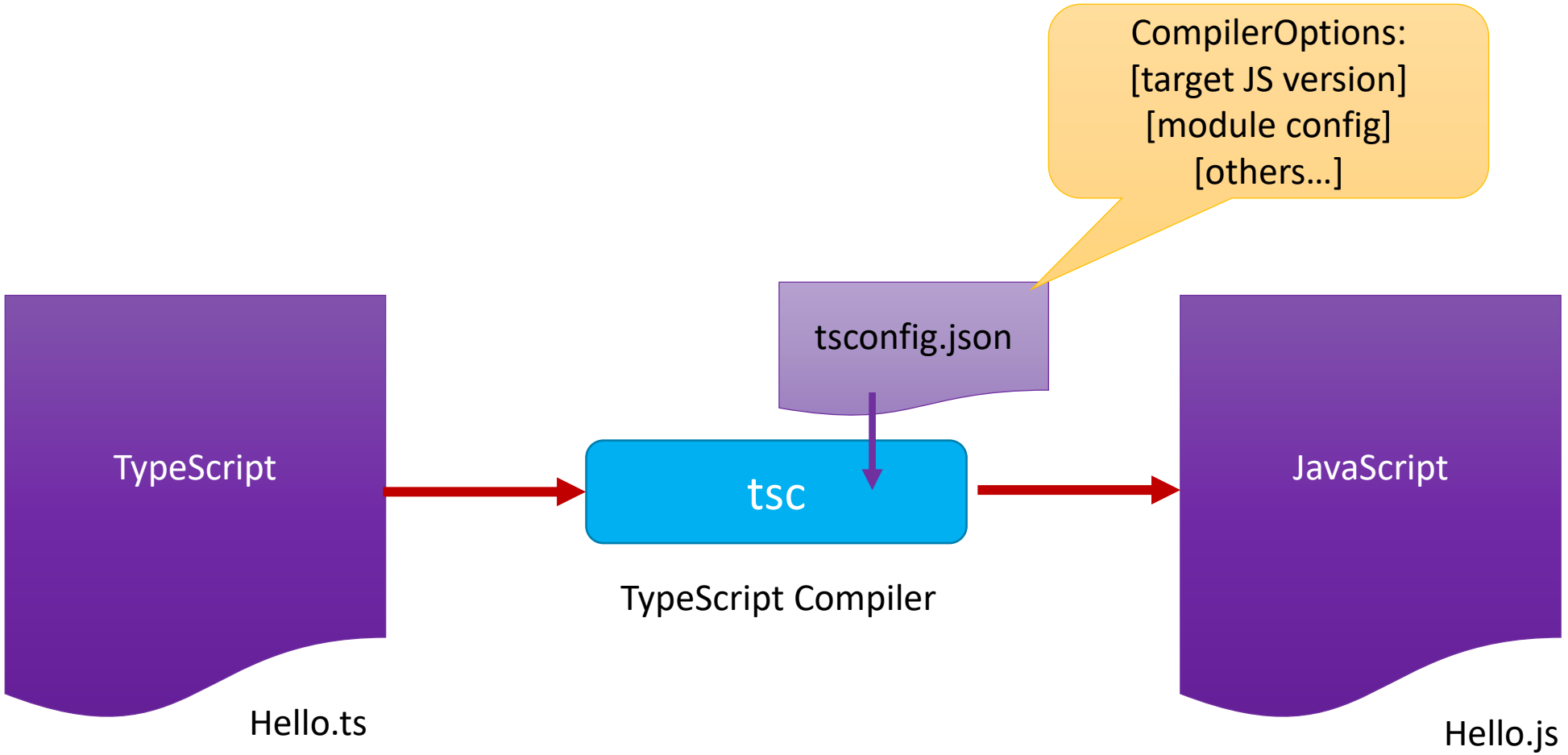
Transcompiles to JavaScript.

Designed for development of large applications.

Open Source.



TypeScript


CompilerOptions:
[target JS version]
[module config]
[others…]
tsconfig.json
TypeScript
tsc
JavaScript
TypeScript Compiler
Hello.ts
Hello.js

TypeScript Features

- Type Annotations
- Compile-Time Type Checking
- Type Inference
- Interfaces
- Classes and Inheritance
- Namespaces and Modules
- Generics
- Decorators
- Arrow Functions

TypeScript Installation

- Install NodeJs and NPM
- Run the command **npm install -g typescript**

TypeScript Types

Boolean

- **let** isAvailable:boolean = false

Number

- **let** age: number = 16;

String

- **let** name: string = “Anil";

Array

- **let** list: number[] = [1, 2, 3];
- **let** list: Array<number> = [1, 2, 3];

TypeScript Types

Enum

- **enum** Color {Red, Green, Blue}
- **let** c: Color = Color.Green;

Any

- **let** x: any = 4;
- x = “hello”

void

```
function foo(): void {
  console.log(“foo");
}
```

null and undefined

- var x: string = null;
- var y:string= undefined

Classes

- Traditionally JavaScript uses functions and prototype-based inheritance to build up reusable components.
- Starting with ECMAScript 2015(ES6), JavaScript introduced the object-oriented class-based approach.
- Typescript supports classes that compile down JavaScript.
  - Works across all major browsers and platforms, without having to wait for the next version of the browser.

Classes

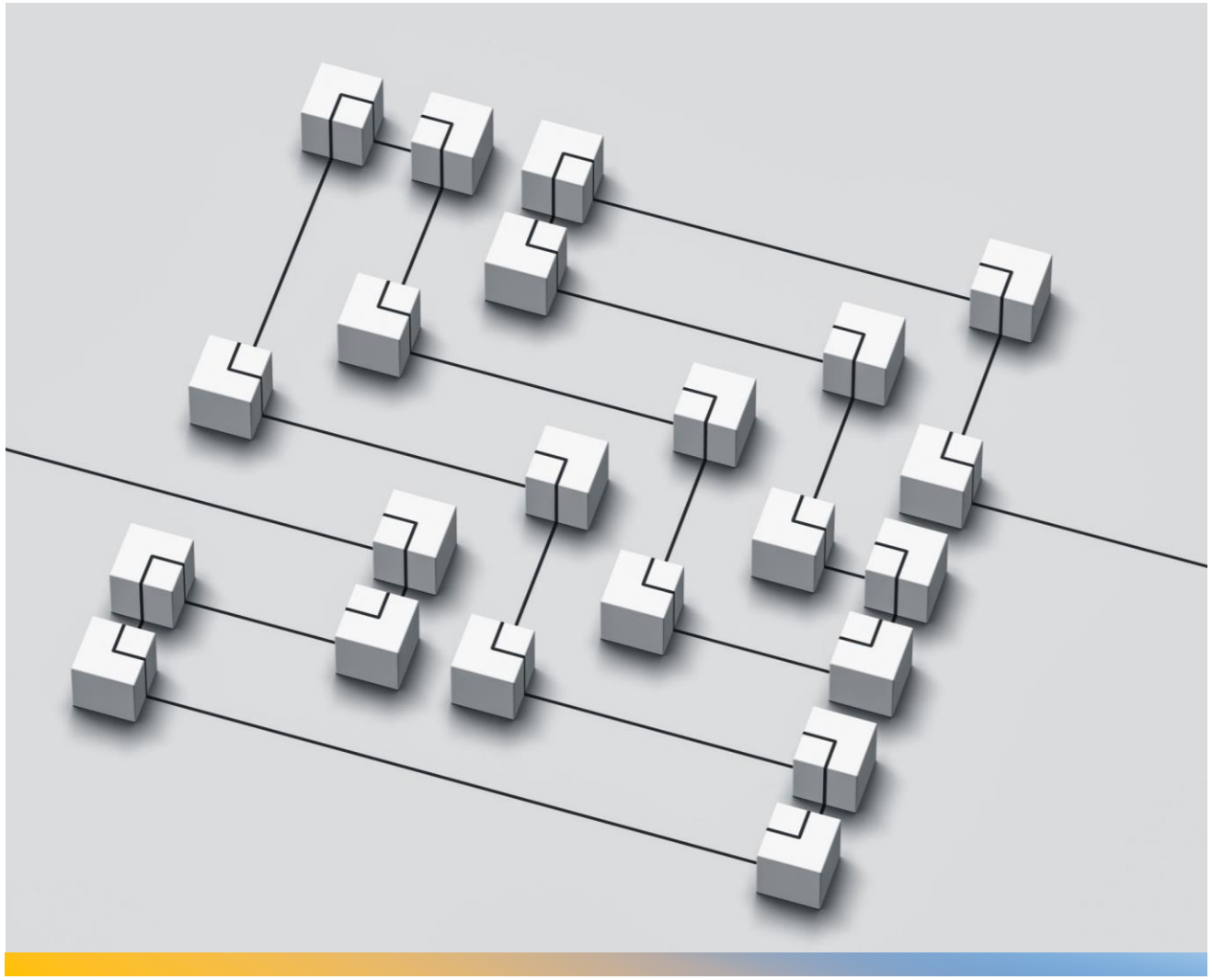
- Access Modifiers
  - public, private, protected
- Constructors
- Properties
- Static Members
- Inheritance
- Abstract classes and methods

Arrow Functions

- Represents a function expression.
- An arrow function expression has a shorter syntax than a function expression.
- They do not receive the implicit “this” and “arguments”.
- Used widely for asynchronous and functional programming

TypeScript Modules

- Starting with ECMAScript 2015(ES6), JavaScript has the concept of modules.
- Modules have a scope of their own
- In the module system every JS file is a module and all declarations in the file is scoped to that module.
- The same concept is shared in TypeScript



Modules

- Use the import and export statements.

```
let foo = function(){
   //some code
}


export default foo;
```

one.js

```
import foo from ‘./one’;

foo();
```

two.js

```
import bar from ‘./one’;

bar();
```

three.js

Modules

```
export let foo = function(){
   //some code
}

 export let bar = function(){
    //some code
 }
```

one.js

```
import {foo, bar} from ‘./one’;


 foo();
 bar();
```

two.js

Angular

A client-side JavaScript Framework for building Web Applications

A Google Product

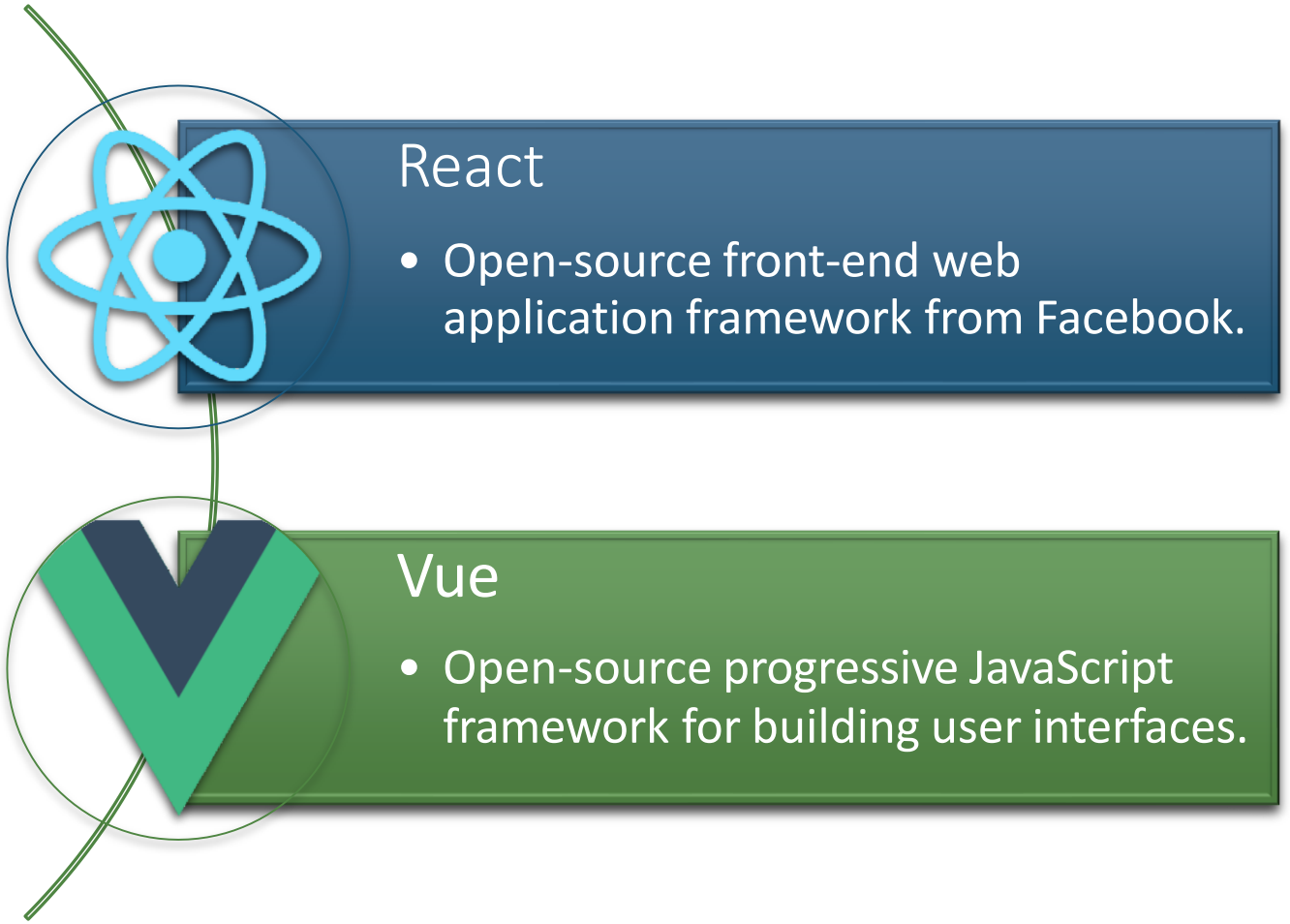
Developed in 2009 by Miško Hevery and Adam Abrons(Google Employees).

Open source under the MIT license.

Angular Core Features

- Client-side JavaScript Framework
- Declarative UI(Extends HTML)
- Data Binding
- Modular
- Loose coupling with Dependency Injection
- Designed for Single Page Application
- Develop using JavaScript, **Typescript** or Dart
- Tools available for development
- Create for Desktop and Mobile
- Supports server-side rendering

Alternatives

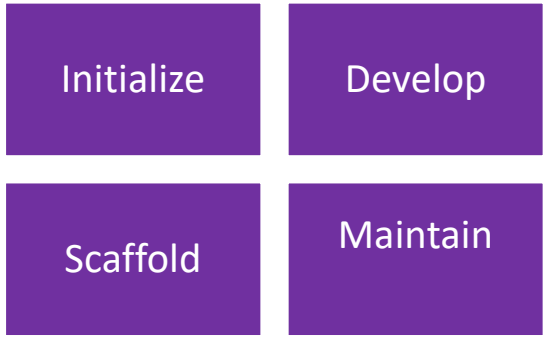


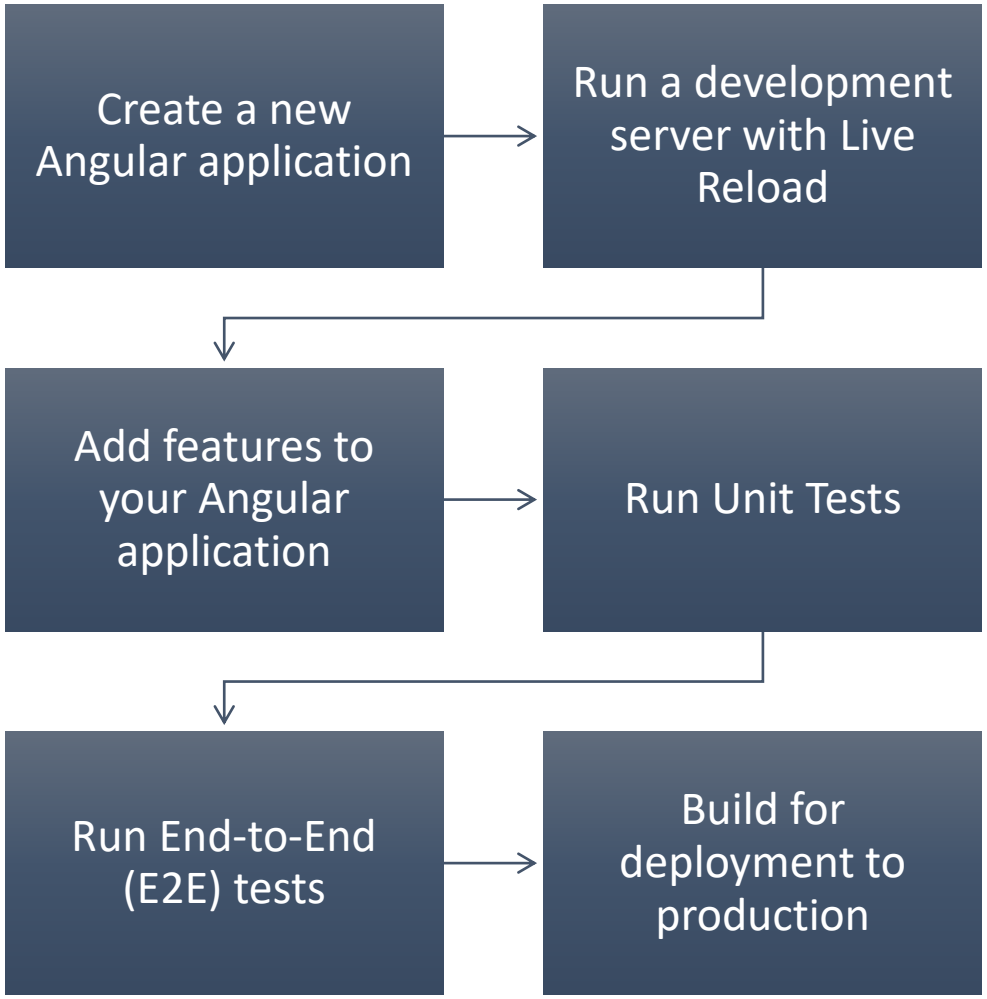
37

Create an Angular Project

Angular CLI

A command-line interface.





Getting Started: CLI

Install Node.js and npm

- node 14 or higher

Install the Angular CLI globally.

- npm install –g @angular/cli@14

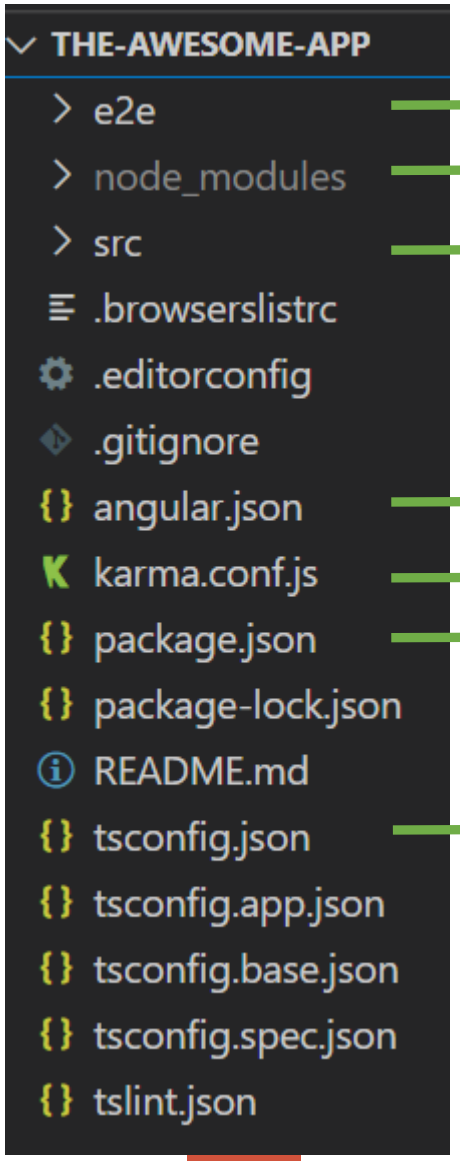
Create a new project

- ng new awesome-app

Serve the application

- cd awesome-app
- ng serve --open

Project Files



e2e configuration

Folder contains downloaded dependencies

Folder contains all the source code

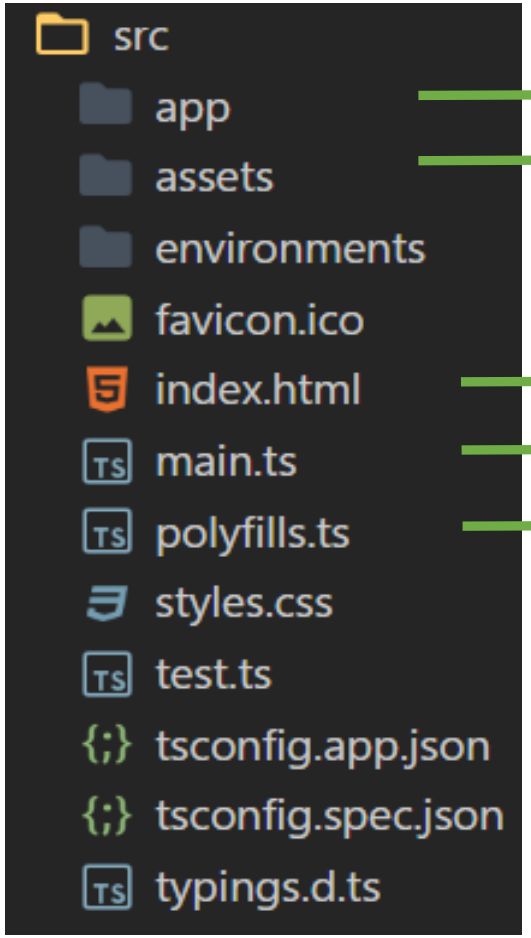
Angular CLI configuration file

Karma configuration file, the test-runner

NPM configuration file, contains the dependencies and scripts

Typescript configuration file, contains typescript compiler options

Project Files



Folder contains all the code

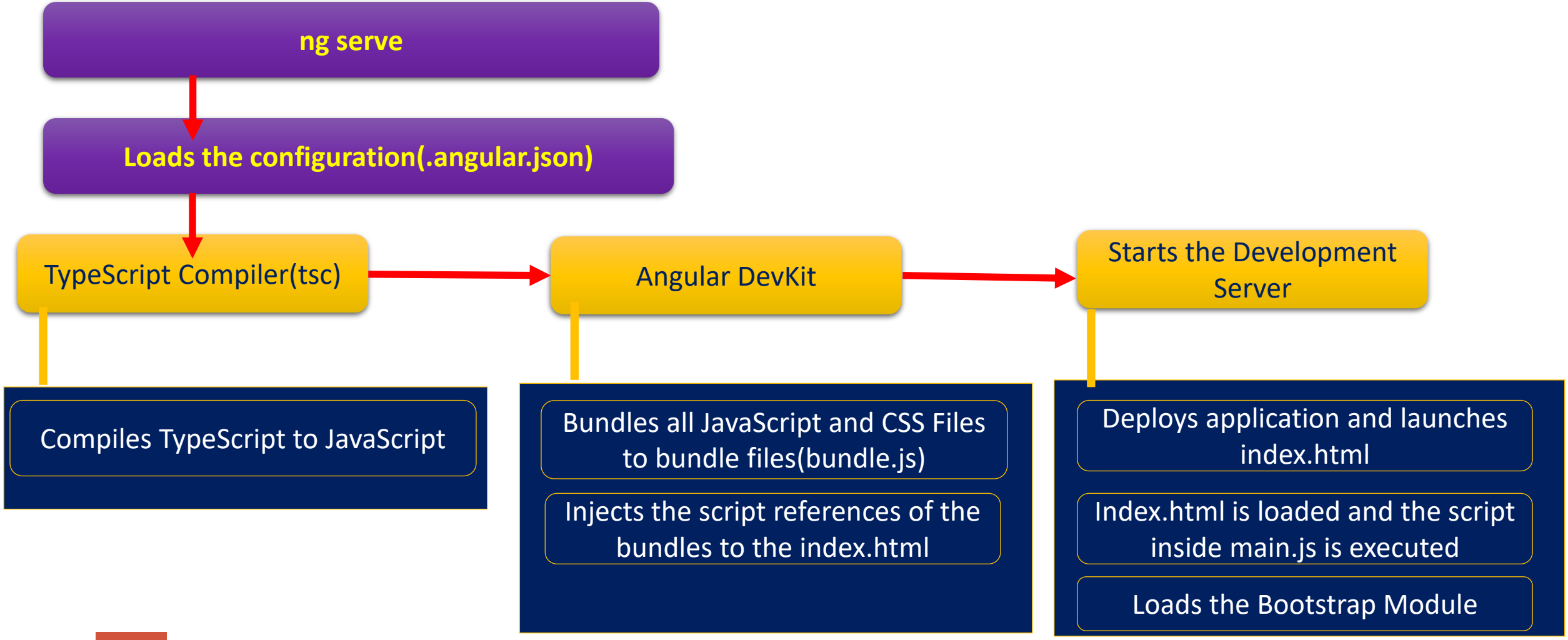
Folder contains all static contents

The page displayed on the browser(In SPA it's the only page)

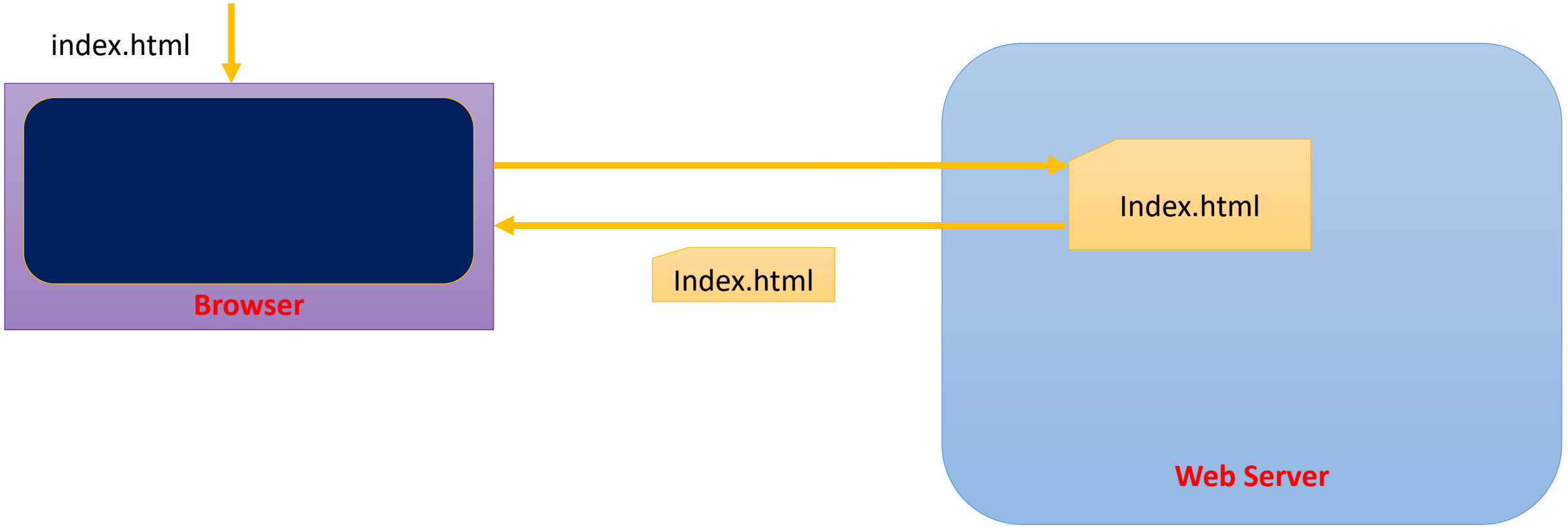
The bootstrapping JavaScript

Config file for polyfills(support older browsers and IE)

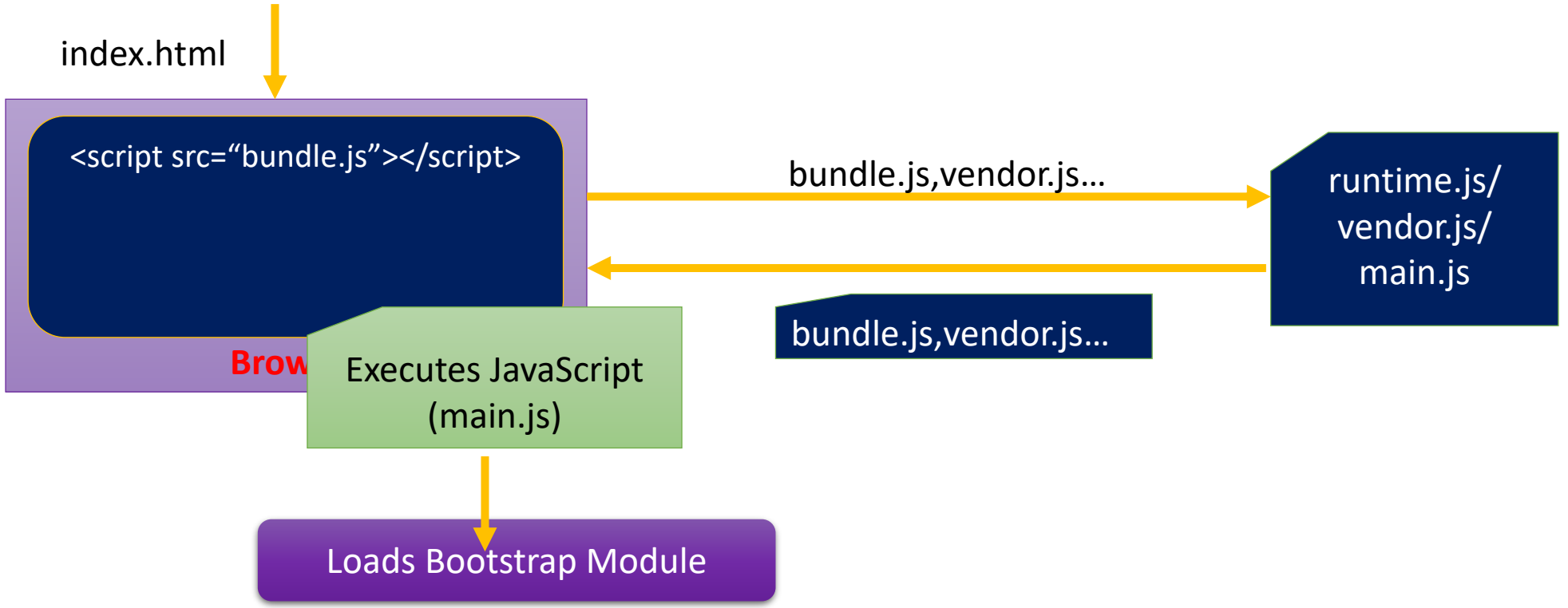
Project Execution



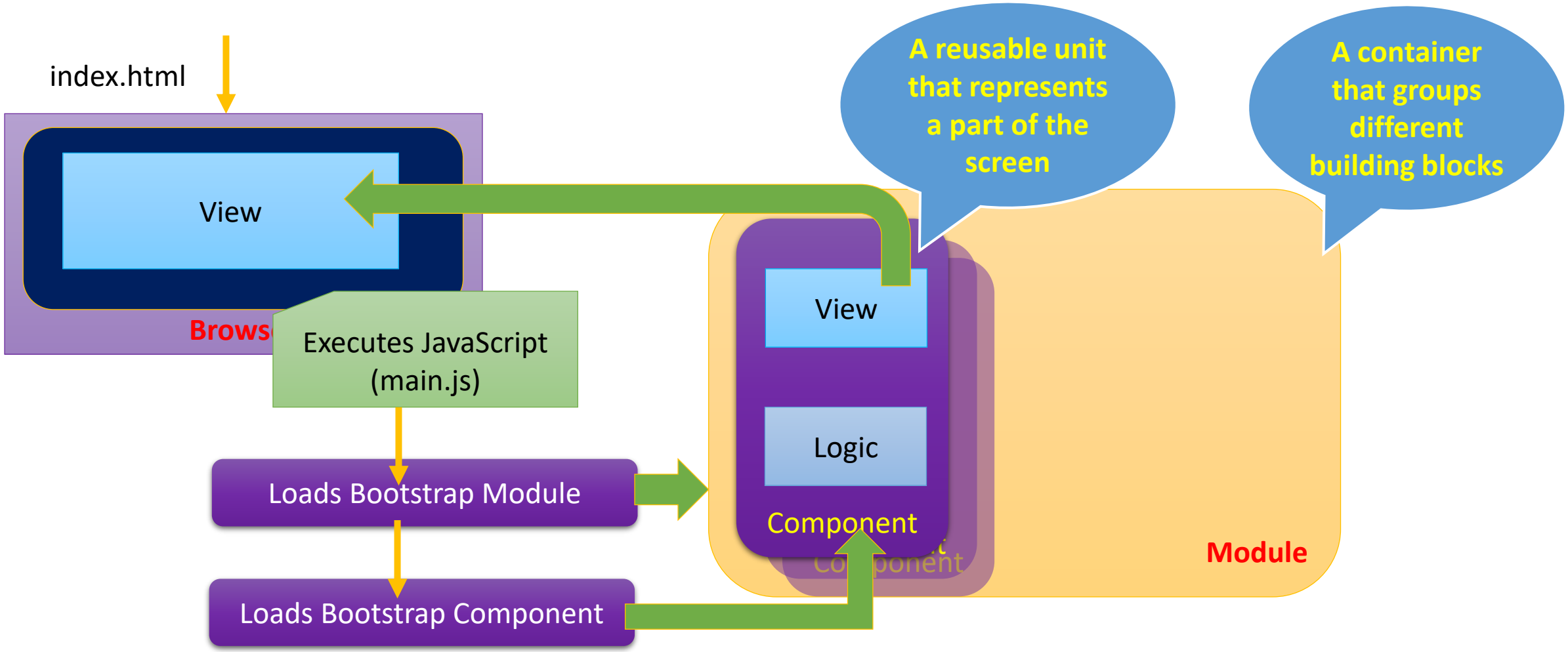
Application Startup



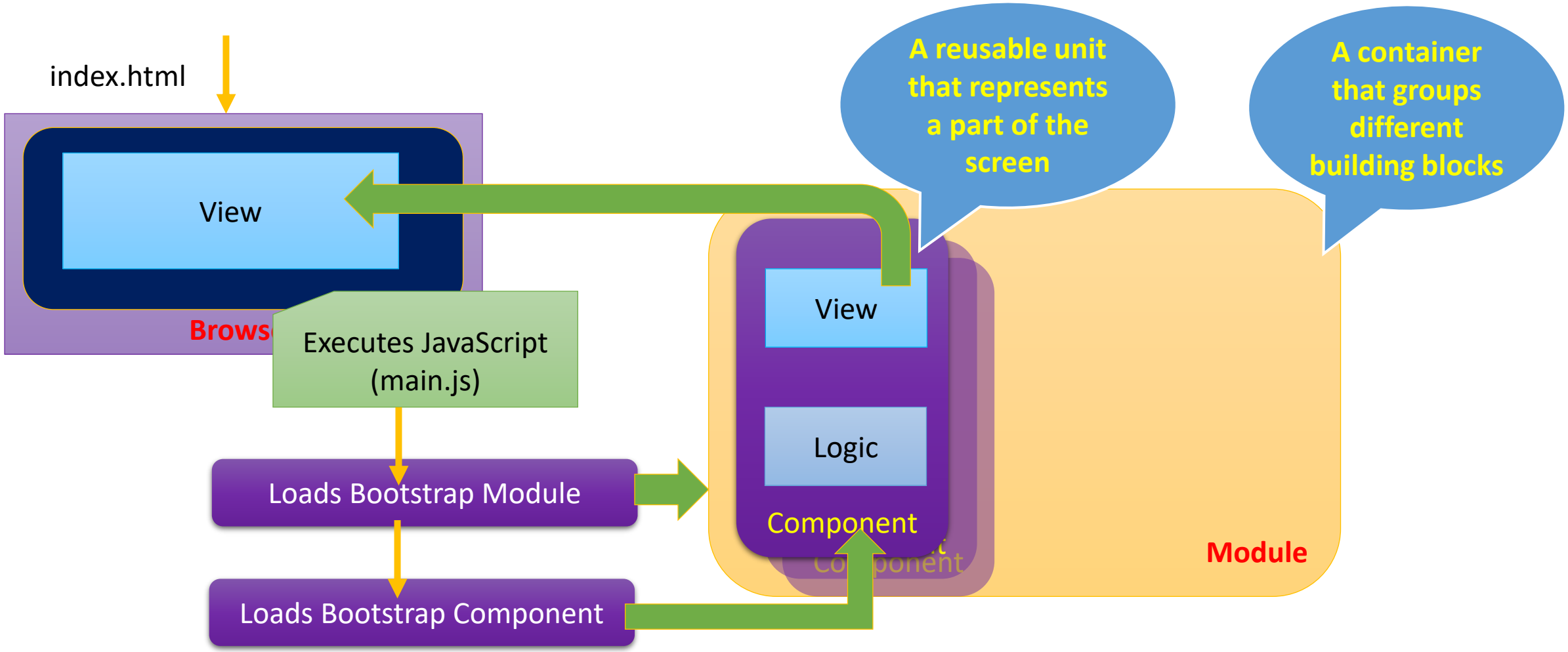
Application Startup



Application Startup

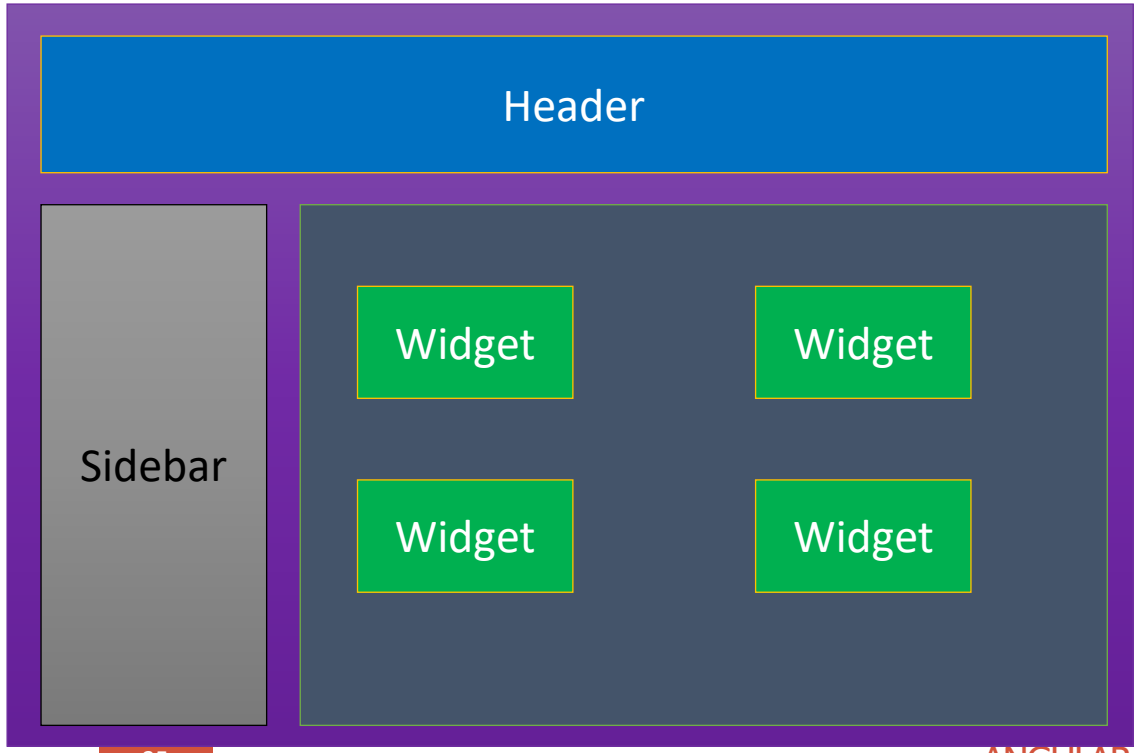


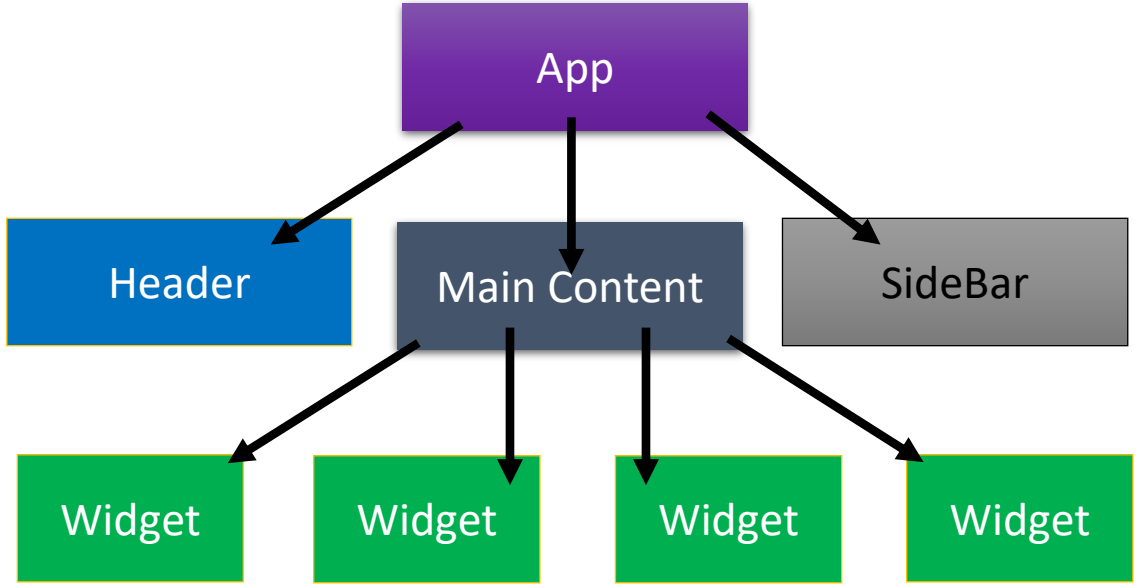
Application Startup



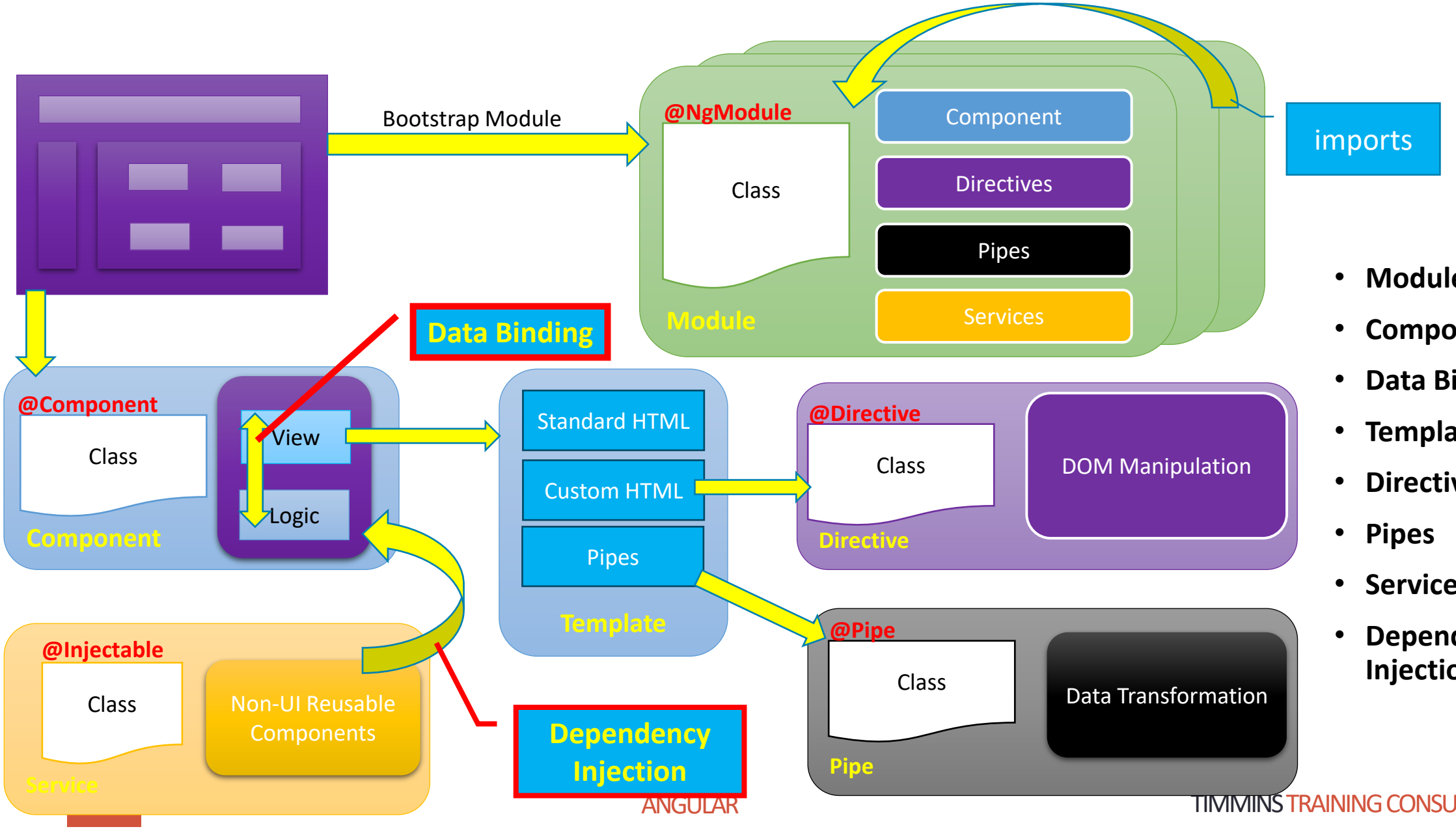
Components

- Components are the **core building blocks of Angular applications**
- An Angular app can be depicted as a **component tree**





Angular Building Blocks



- **Modules**
- **Components**
- **Data Binding**
- **Templates**
- **Directives**
- **Pipes**
- **Services**
- **Dependency Injection**

Angular Building Blocks

Modules

- A module is a set of similar utilities that perform a similar task.
- Contains components, services, directives etc.

Metadata

- Metadata tells Angular how to process a class.
- Also known as decorators(TypeScript)

Components

- Components are the **basic building blocks** of an Angular 2 application.
- It assembles a screen, ui-element or a route in the application.
- Comprises of the view and application logic

Angular Building Blocks

Templates

- We define a component's view with its companion template.
- A template looks like regular HTML

Data Binding

- A mechanism for coordinating parts of a template with parts of a component

Directives

- Angular renders templates(it transforms the DOM) according to the instructions given by directives.

Pipes

- Pipes transform displayed values within a template.

Angular Building Blocks

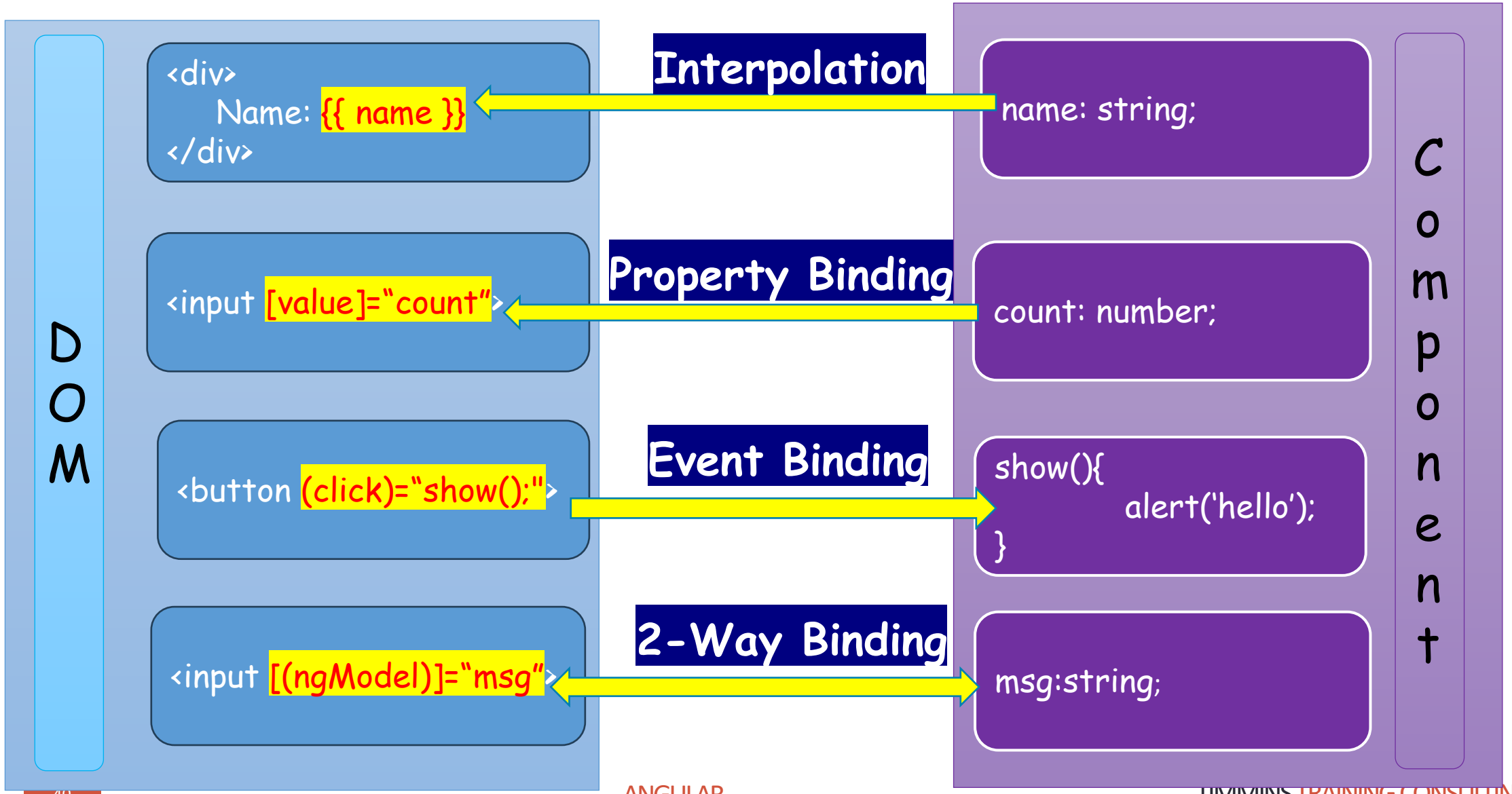
Services

- *Service* is a broad category encompassing any value, function, or feature that your application needs.
- Examples: Logging, DataServices

Dependency Injection

- *Dependency injection* is a way to supply a new instance of a class with the fully-formed dependencies it requires.

Data Binding



Modules

- Modules consolidate components, directives, services and pipes into cohesive blocks of functionality.
- Modules can be loaded eagerly when the application starts.
- They can also be *lazy loaded* asynchronously by the router.
- Every application will have a module called the root module. (This is bootstrapped from the main.js file)
- Decorate a class with @NgModule to define a module.

Directives

Directives allows to manipulate the HTML DOM.

The extend HTML with custom markup.

Angular 1.x shipped with over 60 directives

Angular 2 has a smaller list of directives.

The new binding syntax eliminates the need for many directives.

Directive Types

Components

- Angular 2 components are *actually* just directives under the hood
- Directives with a template.

Structural directives

- Change the DOM layout by adding and removing DOM elements.

Attribute directives

- Change the appearance or behavior of an element, component, or another directive.

Structural Directives

Structural directives are responsible for HTML layout.

Typically used for adding, removing, or manipulating elements.

An asterisk (*) precedes the directive attribute name.

- <div *ngIf="hero" >{{hero.name}}

Built-in structural directives

- NgIf
- NgFor
- NgSwitch

Use <ng-container> to group elements when there is no suitable host element for the directive.

Components

- A *component* controls a patch of screen called a *view*.
- Defined as class with the Component decorator
- @Component properties
  - **selector** - css selector that identifies this component in a template
  - **styleUrls** - list of urls to stylesheets to be applied to this component's view
  - **styles** - inline-defined styles to be applied to this component's view
  - **template** - inline-defined template for the view
  - **templateUrl** - url to an external file containing a template for the view

Attribute Directives

Attribute directives listen to and modify the behavior of other HTML elements, attributes, properties, and components.

They are usually applied to elements as if they were HTML attributes

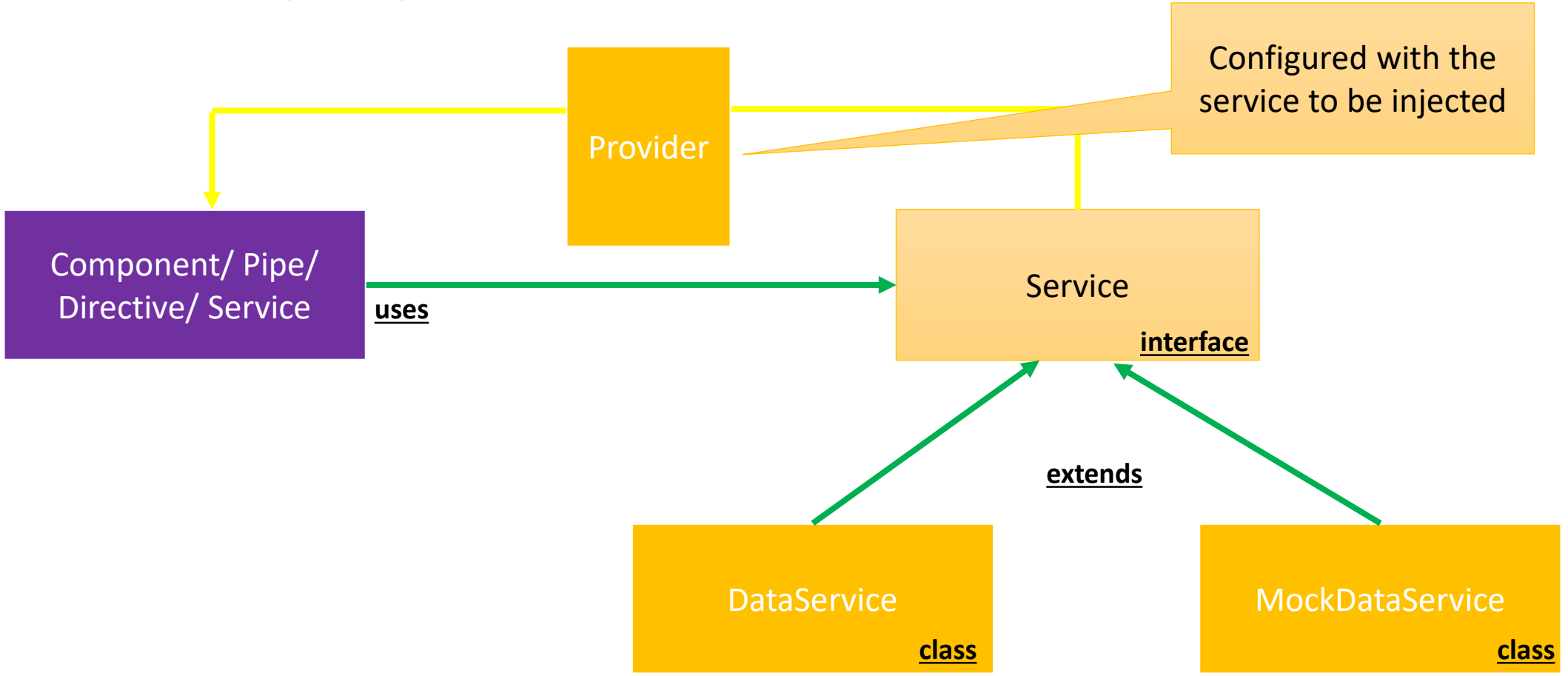
Built-in attribute directives

- NgClass
- NgStyle
- NgModel

Services

- Services are reusable functionalities.
- They can be injected into components, services, pipes and directives
- Implemented as Simple Classes in Angular
- Services can be configured as singletons
- They can be used to share data between components

Dependency Injection



Pipes

- Pipes transform displayed values within a template.
- Used inside the interpolation expression with the pipe operator (|)
  - {{ dateOfBirth | date}}
- Built-in Pipes
  - uppercase
  - lowercase
  - currency
  - percent
  - json
- Pipes can be chained togather
  - {{ dateOfBirth | date | uppercase }}
- Custom pipes.

Pipes

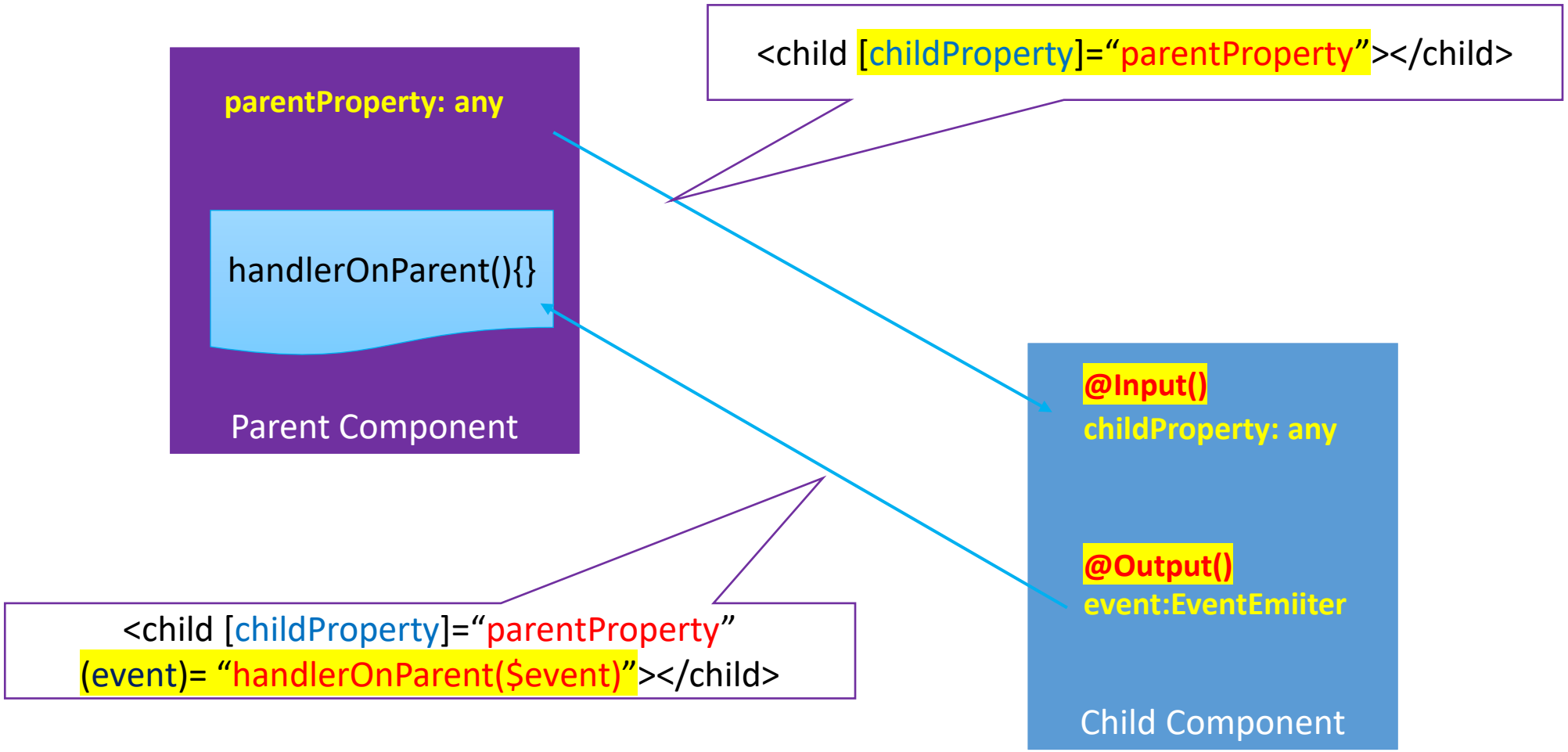
{{ message | uppercase }}

```
transform(input: String): String{
        let output;
        // some code

        return output;
}
```

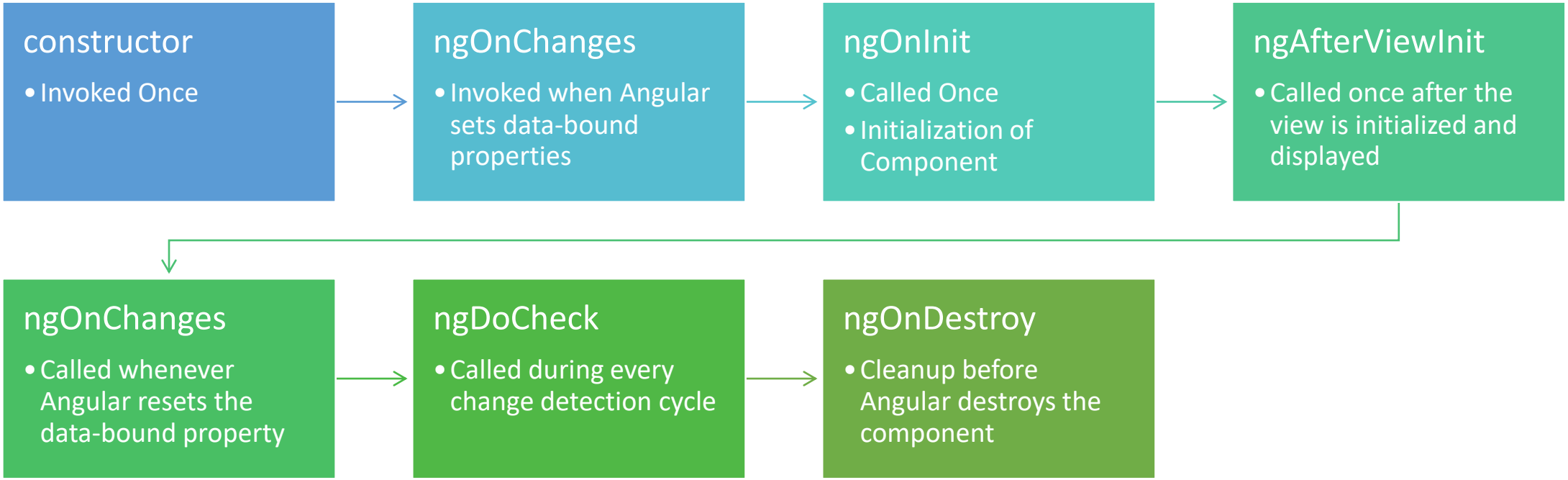
{{ dateOfBirth | date: 'mdy' }}

```
transform(input: Date, format: String): Date{
        let output;
        // some code

        return output;
}
```

Component Interaction(Parent-Child)



Component Lifecycle



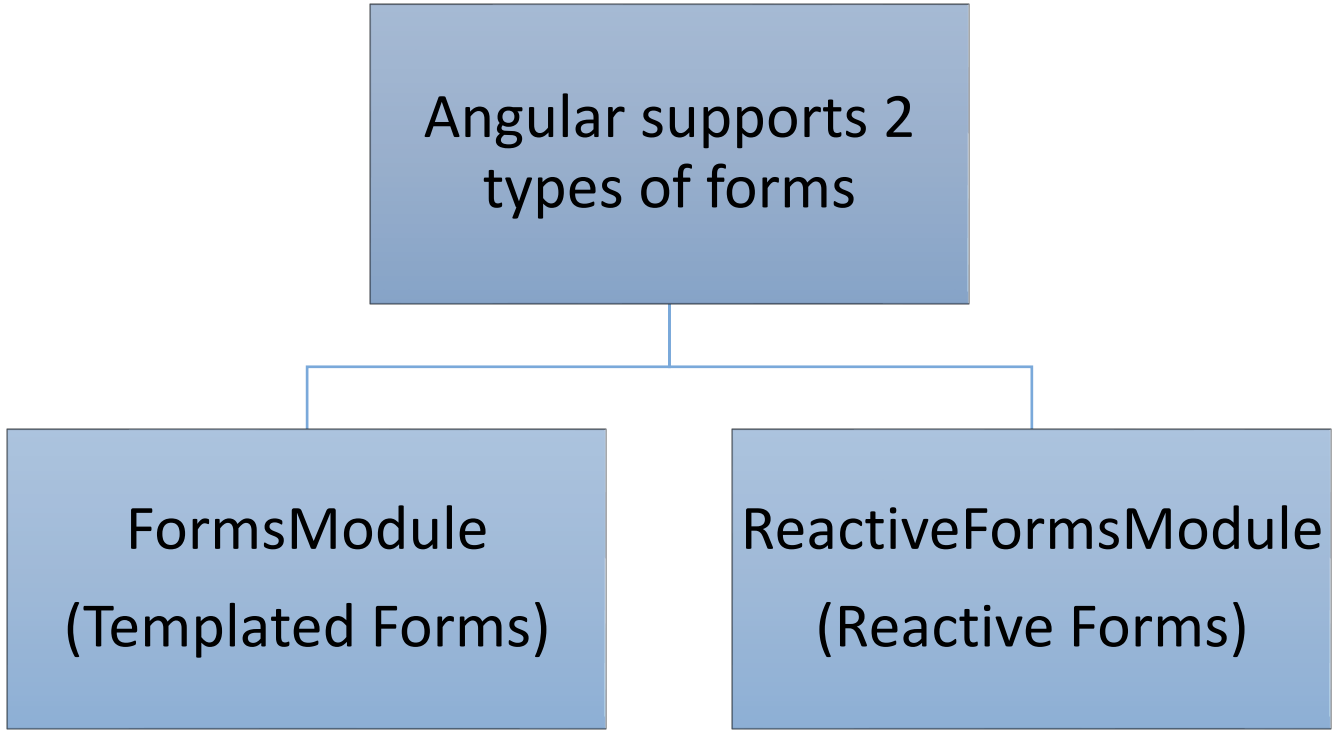
Change Detection

- Change Detection means updating the view (DOM) when the data has changed.
- Angular Provides 2 strategies
  - Default
  - Push
- Default: By default, Angular will check every time something may have changed, this is called dirty checking.
- onPush: Angular will only depend on the component's **inputs, events, markForCheck** method, or the use of the **async pipe** in the template, to perform a change detection mechanism and update the view.

View Encapsulation

- Defines how the CSS styles are scoped or encapsulates
- Emulated
  - No Shadow DOM
  - Provides Encapsulation
- ShadowDom/Native
  - Uses Shadow DOM
  - Provides Encapsulation
- None
  - No Shadow DOM
  - No Style Encapsulation

Forms

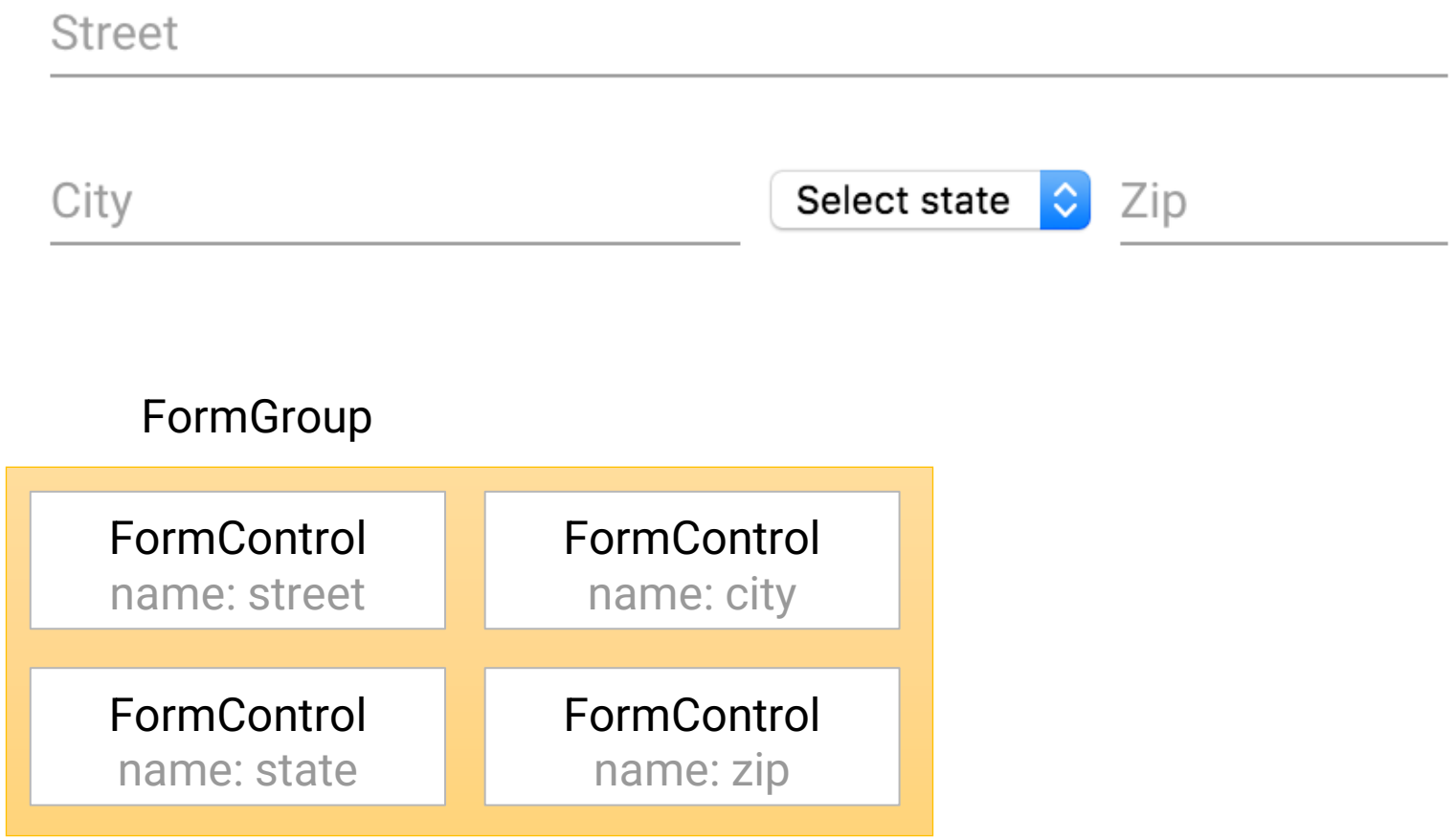

Angular supports 2 types of forms
FormsModule (Templated Forms)
ReactiveFormsModule (Reactive Forms)

Forms: Building Blocks

Forms: FormControl

Value
Validation status
User interactions
Events

Forms: FormGroup



Forms

Reactive Forms Module

- Explicit creation of FormControls
- Source of truth: component class
- Reactive Programming Support

Forms Module

- Implicit creation of FormControl by directives
- Source of truth: template

Reactive Programming

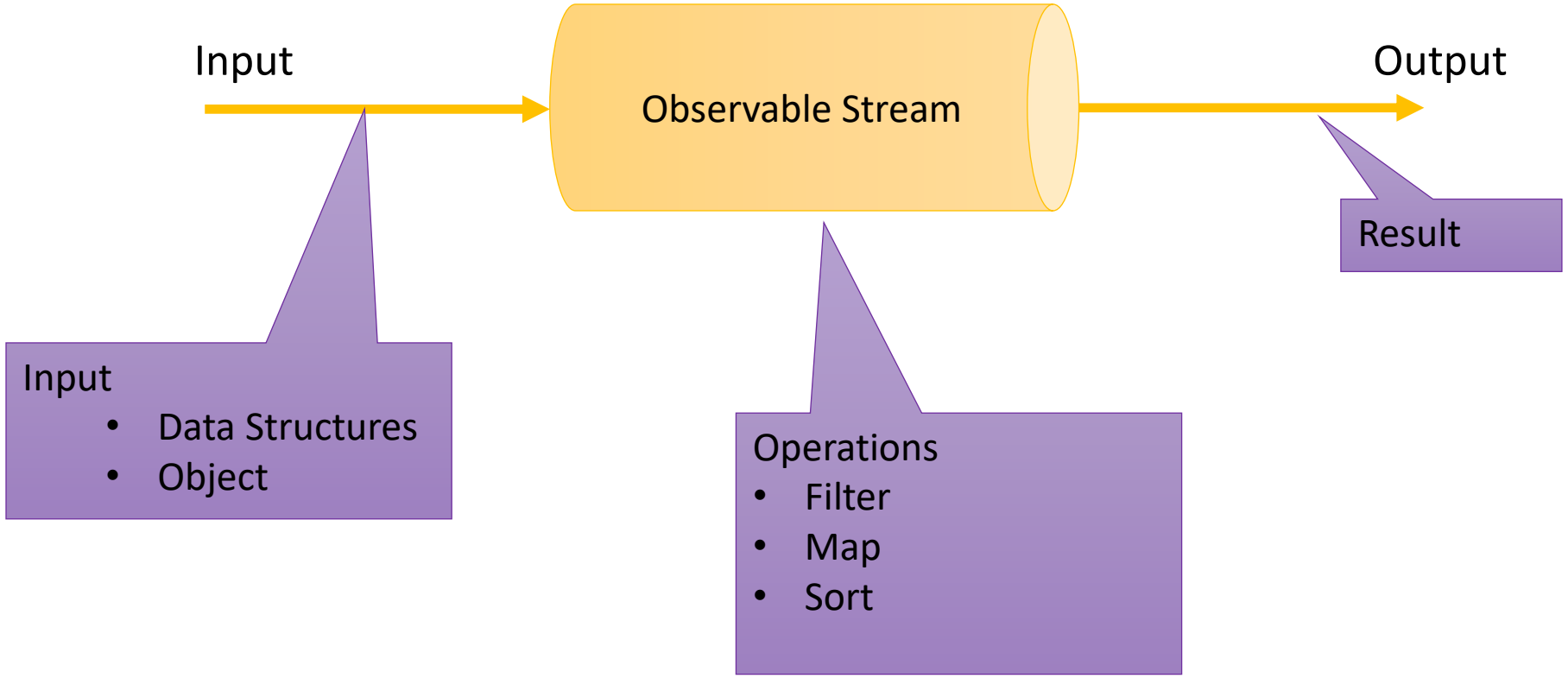
Reactive programming is a declarative programming paradigm concerned with data streams and the propagation of change

ReactiveX(http://reactivex.io/) is an API for asynchronous programming with observable streams

RxJS is a JavaScript library for reactive programming using Observables

In Angular, RxJS is used with Reactive Forms, Http, Routers

Reactive Programming(Streams)



Http Client

- Angular exposes a service to send AJAX request to the server.
- The service is a wrapper around the XMLHttpReqeust object.
- Angular 4
  - HttpModule
  - Http(Service)
- Angular 5
  - HttpClientModule
  - HttpClient

Http: Methods

request

Performs any type of http request.

A RequestOptions object allows to configure the call

get

Performs a request with http method.

A RequestOptions object allows to configure the call

post

Performs a request with http method.

A RequestOptions object allows to configure the call

put

Performs a request with http method.

A RequestOptions object allows to configure the call

delete

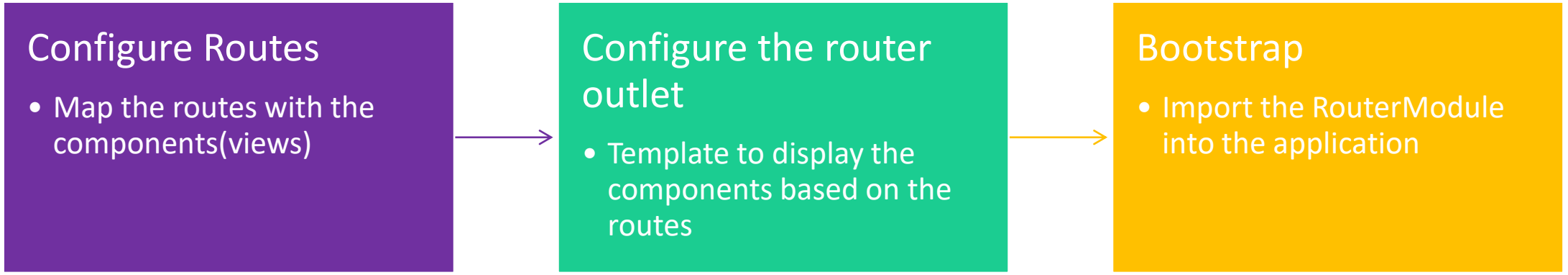
Performs a request with http method.

A RequestOptions object allows to configure the call

Router

- The Angular Router enables navigation from one view to the next as users perform application tasks.
- Location Stratergies
  - Hash Location Eg: #/home
  - Path Location
    - Requires <base href="../">
    - Eg: /home

Router: Setup



Router: Navigation

- Using regular links with hash
  - #/home
  - /home
- Using routerLink directive
  - <a [routerLink]="/home">Home
- Programmatically
  - router.navigate(['users', 10])
  - router.navigateByUrl('/users/10')

NgRx

- NgRx is a framework for building reactive applications in Angular.
- Inspired by Redux
- NgRx provides state management for
  - Creating maintainable explicit applications
  - Storing single state
  - Use of actions in order to express state changes.s

NgRx(When to use?)

- Shared
  - State that is accessed by many components and services.
- Hydrated
  - State that is persisted and rehydrated from external storage.
- Available
  - State that needs to be available when re-entering routes.
- Retrieved
  - State that must be retrieved with a side-effect.
- Impacted
  - State that is impacted by actions from other sources.

Ivy Compiler

Ivy is Angular's next-generation compilation and rendering pipeline.

Creates smaller bundles and faster compilation

Better debugging

Dynamic loading of modules

Presented By Anil Joseph(anil.jos@gmail.com)

Best Practices

- Change Detection : Use onPush for better performance
- Design for immutability
- Use Pure Pipes in templates over methods
- Use trackBy with ngFor
- Lazy load modules
- Server-side rendering
- Use Progressive Web Applications

Best Practices

Caching Server calls

Clean Up components

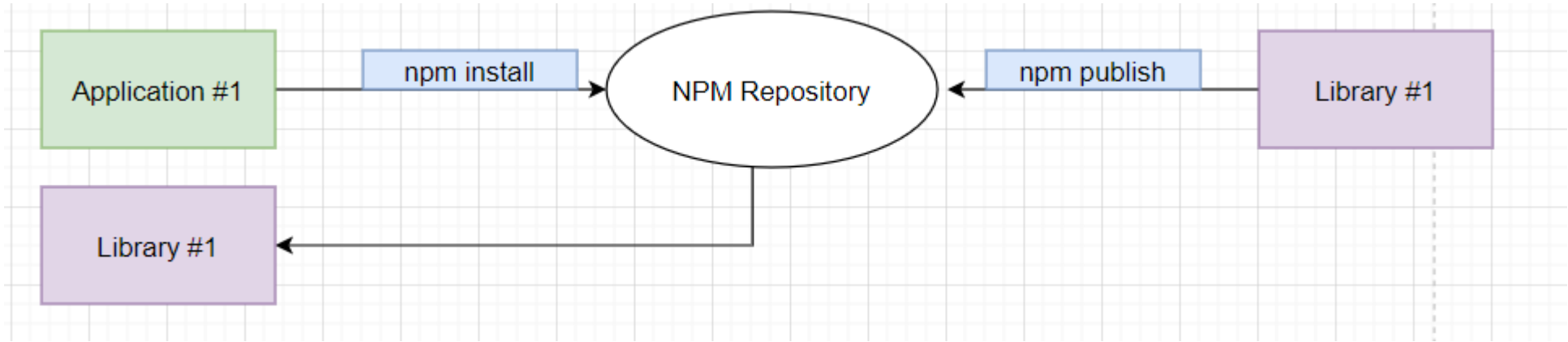
Modularize

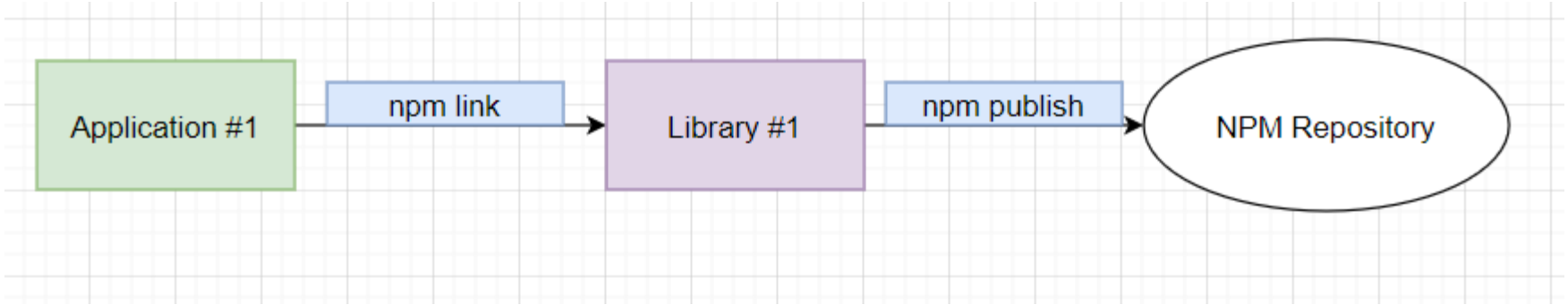
Design for Reusability

Smaller Components

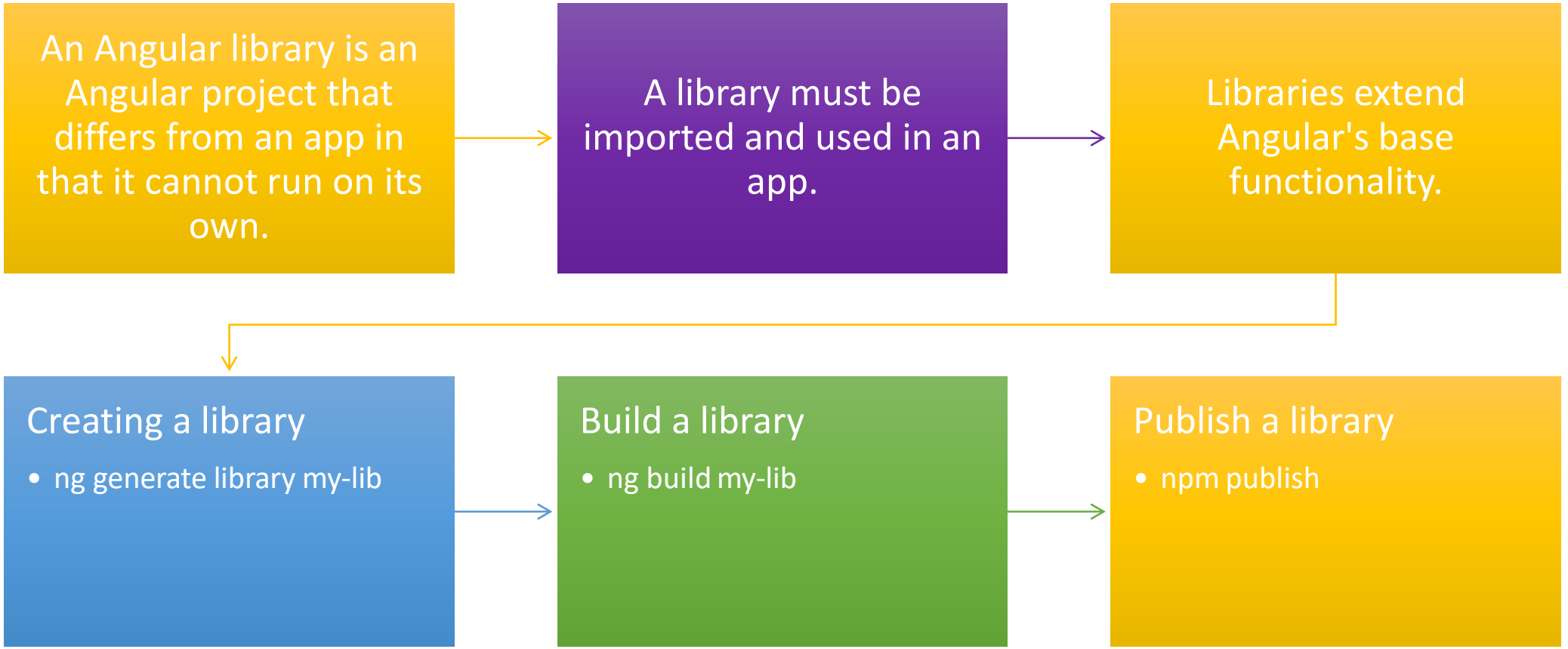
Use Libraries

Angular Library

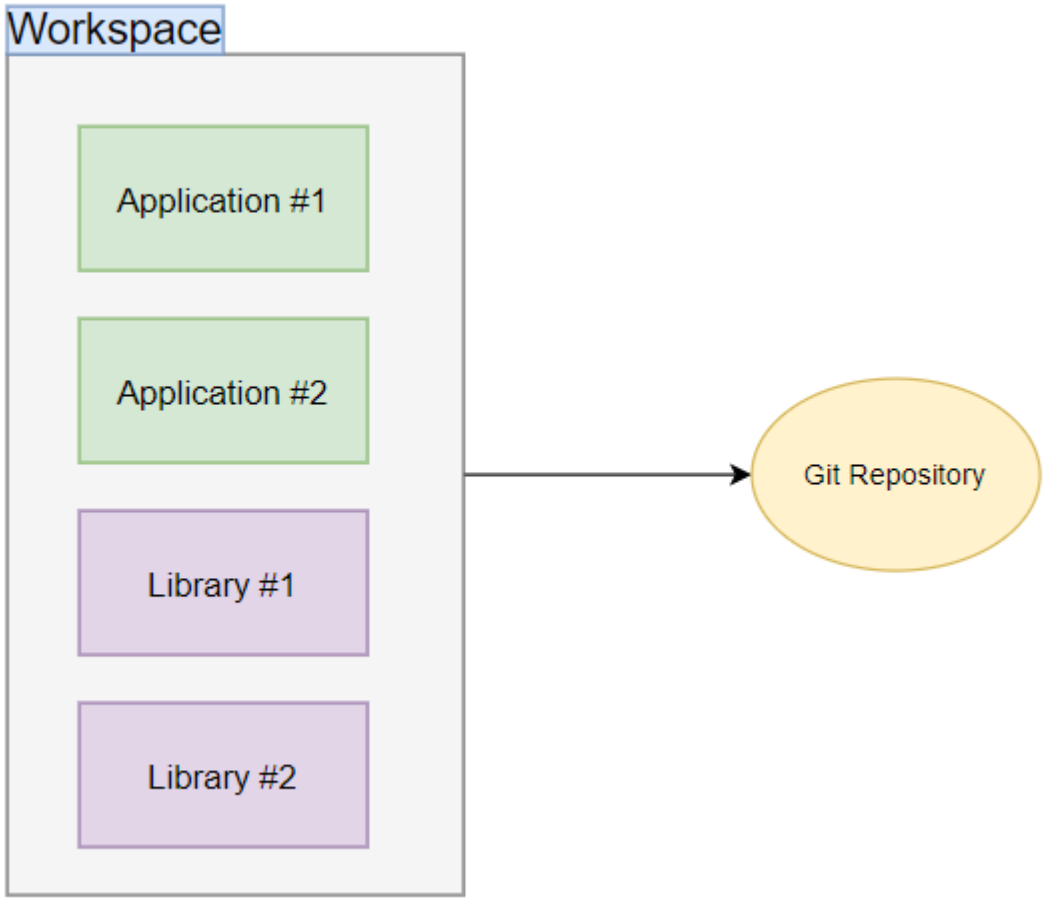




Angular Library



Angular Workspace(Mono Repo)



Create a workspace

```
ng new my-workspace –create-application=“false”
```

Create an application in a workspace

```
cd my-workspace
ng generate application my-first-app
```

Create a library in a workspace

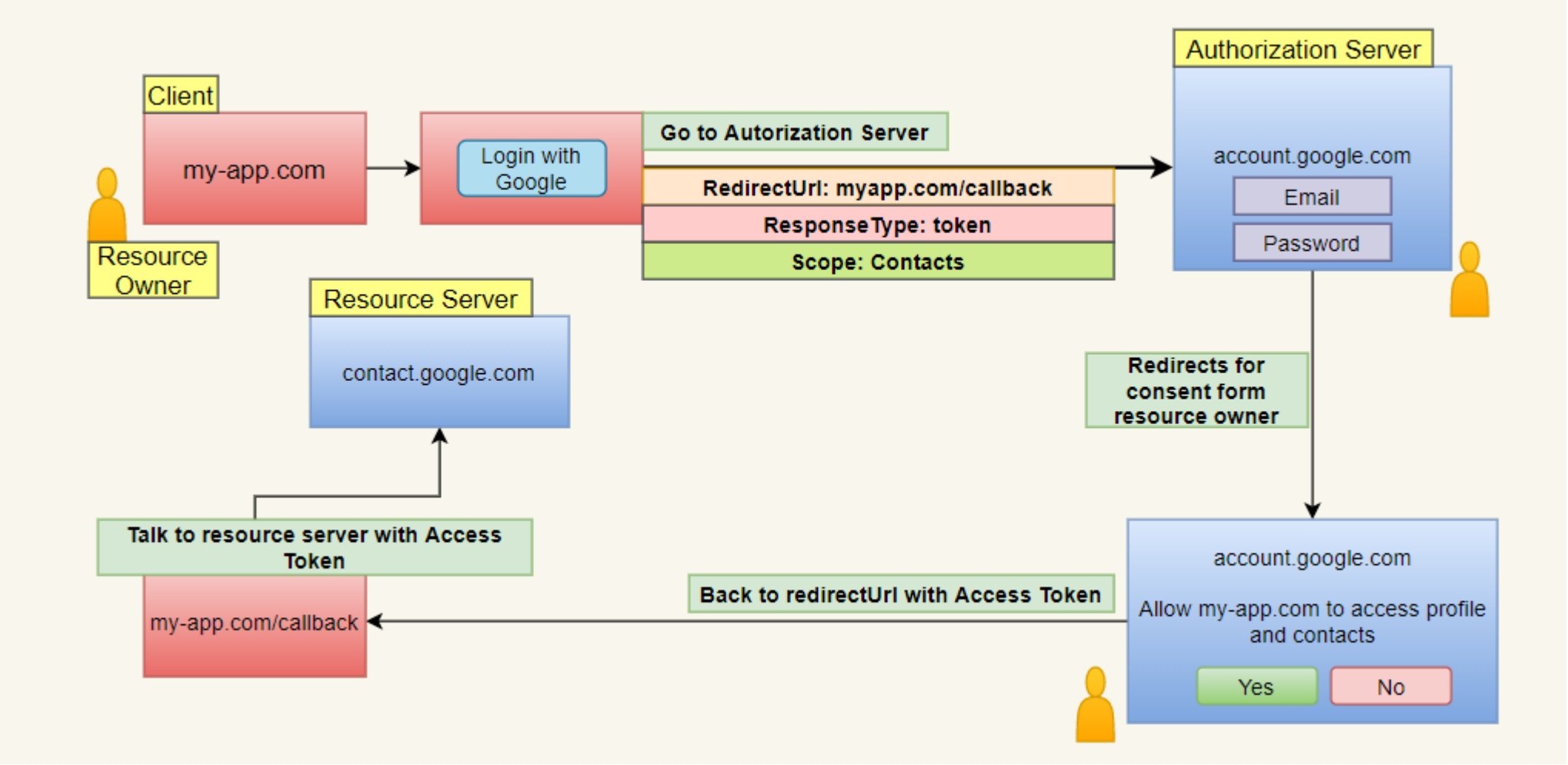
```
cd my-workspace
ng generate library my-lib
```

Angular Workspace(Mono Repo)

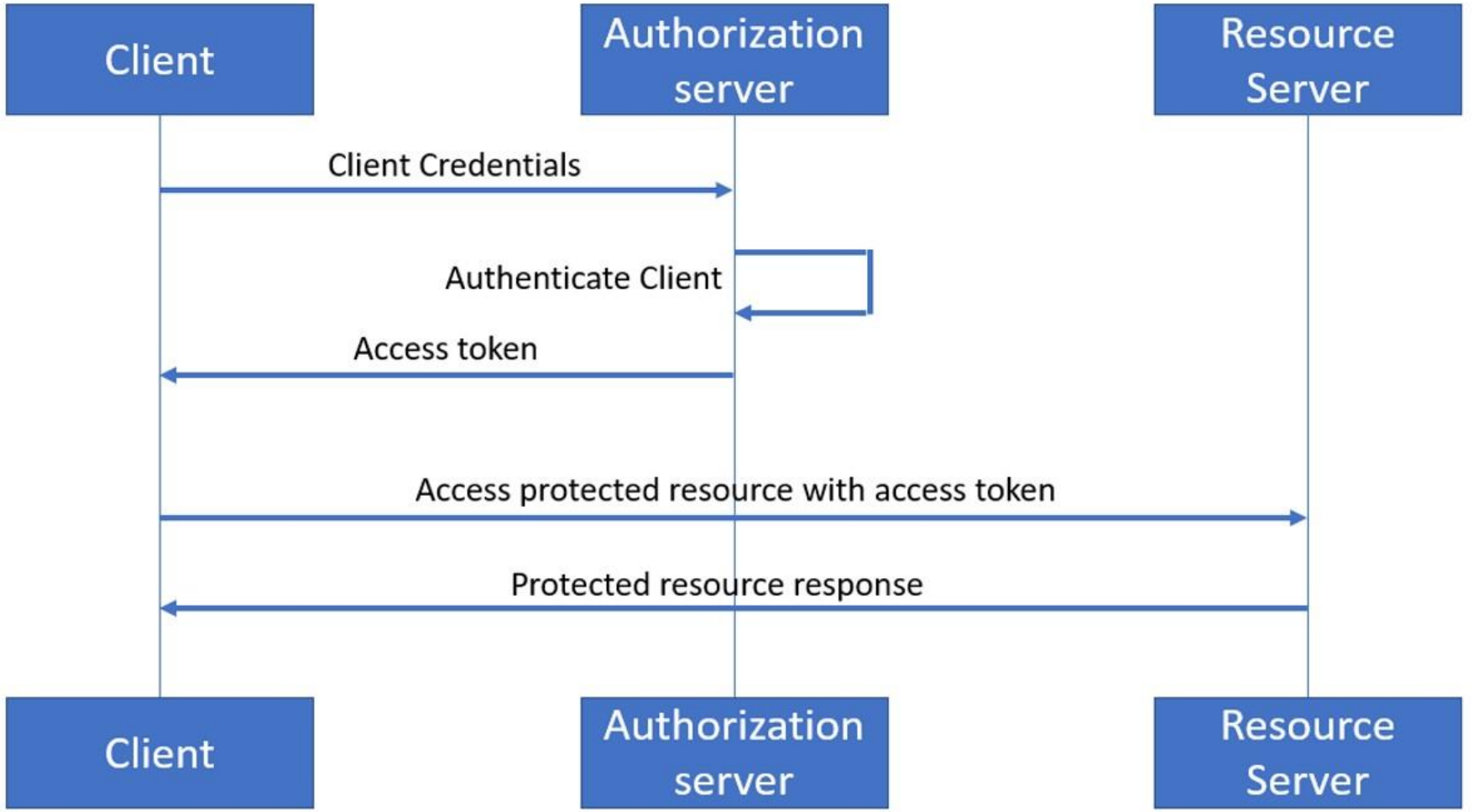
OAuth 2.0

- OAuth is an open standard for access delegation.
- Used as a way for Internet users to grant websites or applications access to their information on other websites but without giving them the passwords.
- Commonly used by companies such as Amazon, Google, Facebook, Microsoft and Twitter to permit the users to share information about their accounts with third party applications or websites.

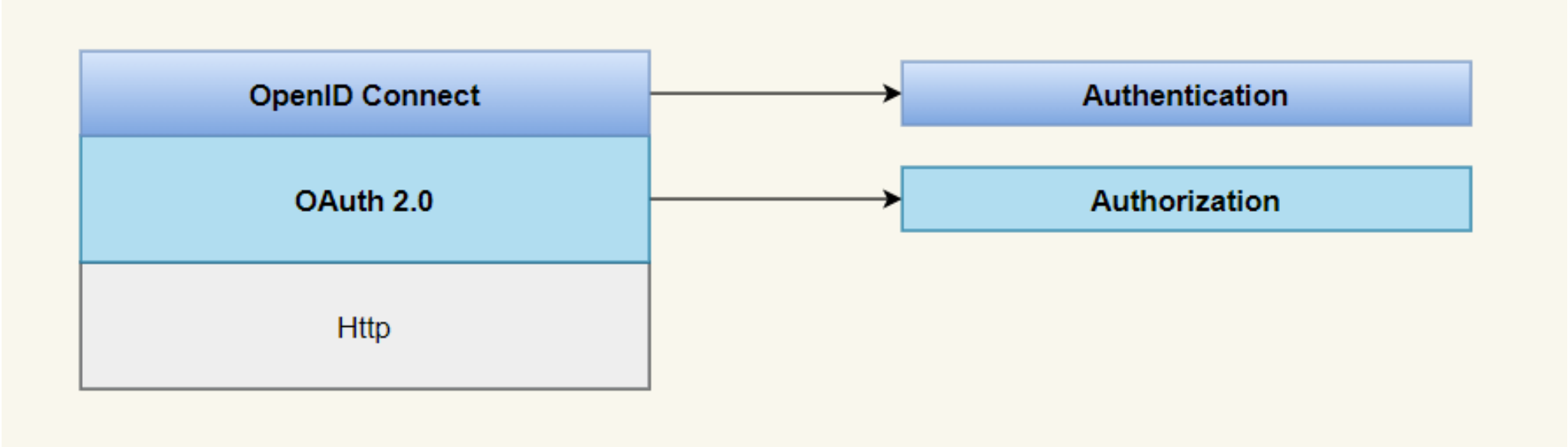
OAuth 2.0


Client
my-app.com
Resource Owner
Login with Google
Go to Autorization Server
RedirectUrl: myapp.com/callback
ResponseType: token
Scope: Contacts
Authorization Server
account.google.com
Email
Password
Resource Server
contact.google.com
Redirects for consent form resource owner
Talk to resource server with Access Token
my-app.com/callback
Back to redirectUrl with Access Token
account.google.com
Allow my-app.com to access profile and contacts
Yes
No

OAuth 2.0


Client
Authorization server
Resource Server
Client Credentials
Authenticate Client
Access token
Access protected resource with access token
Protected resource response
Client
Authorization server
Resource Server

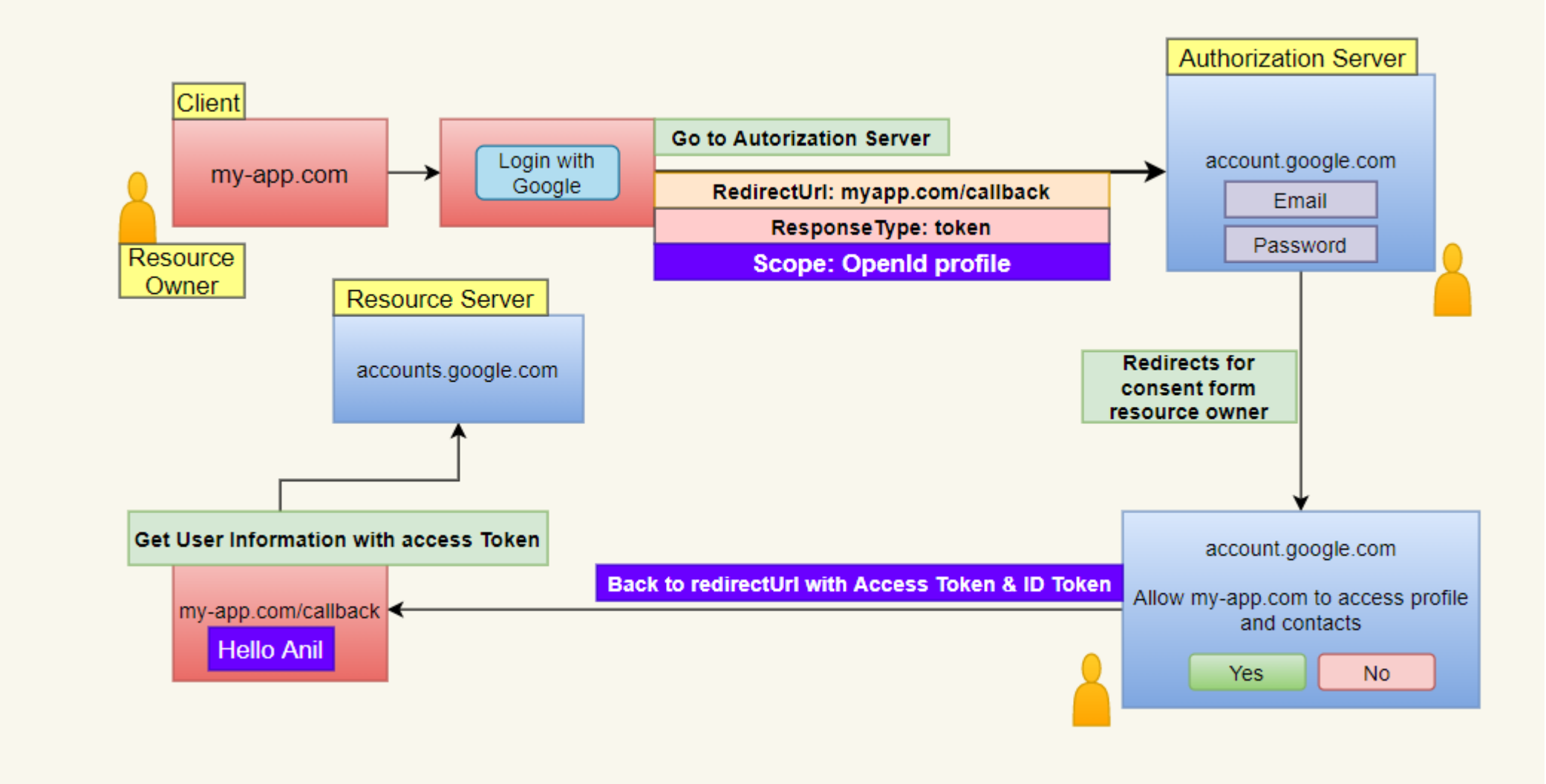
OpenID Connect



OpenID Connect

- OpenID Connect is a simple identity layer on top of the OAuth 2.0 protocol.
- Allows Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server.
- Can obtain basic profile information about the End-User in an interoperable and REST-like manner.

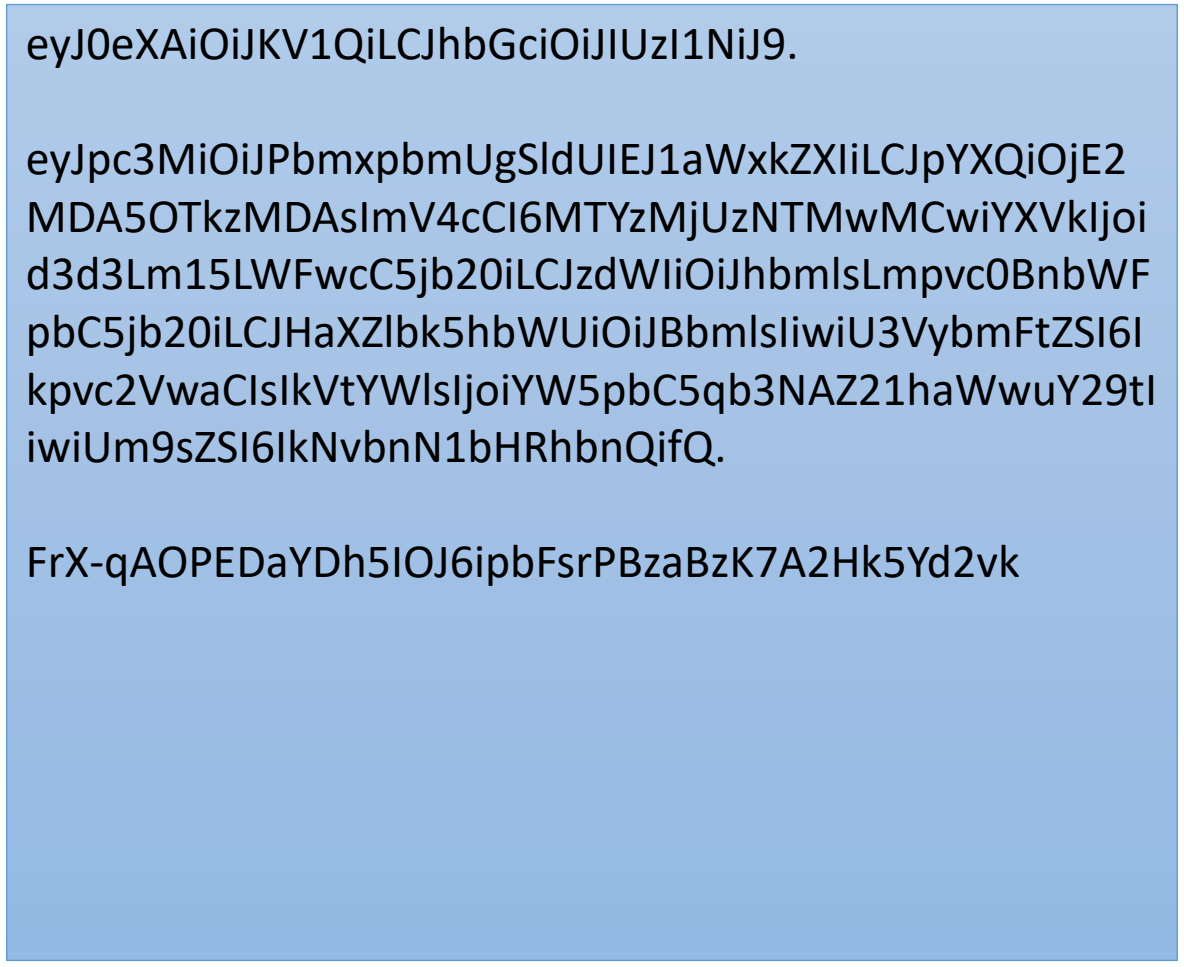
OpenID Connect



Json Web Token

- JSON Web Token (JWT) is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
- Digitally Signed
- User for Authorization and Information Exchange

JWT

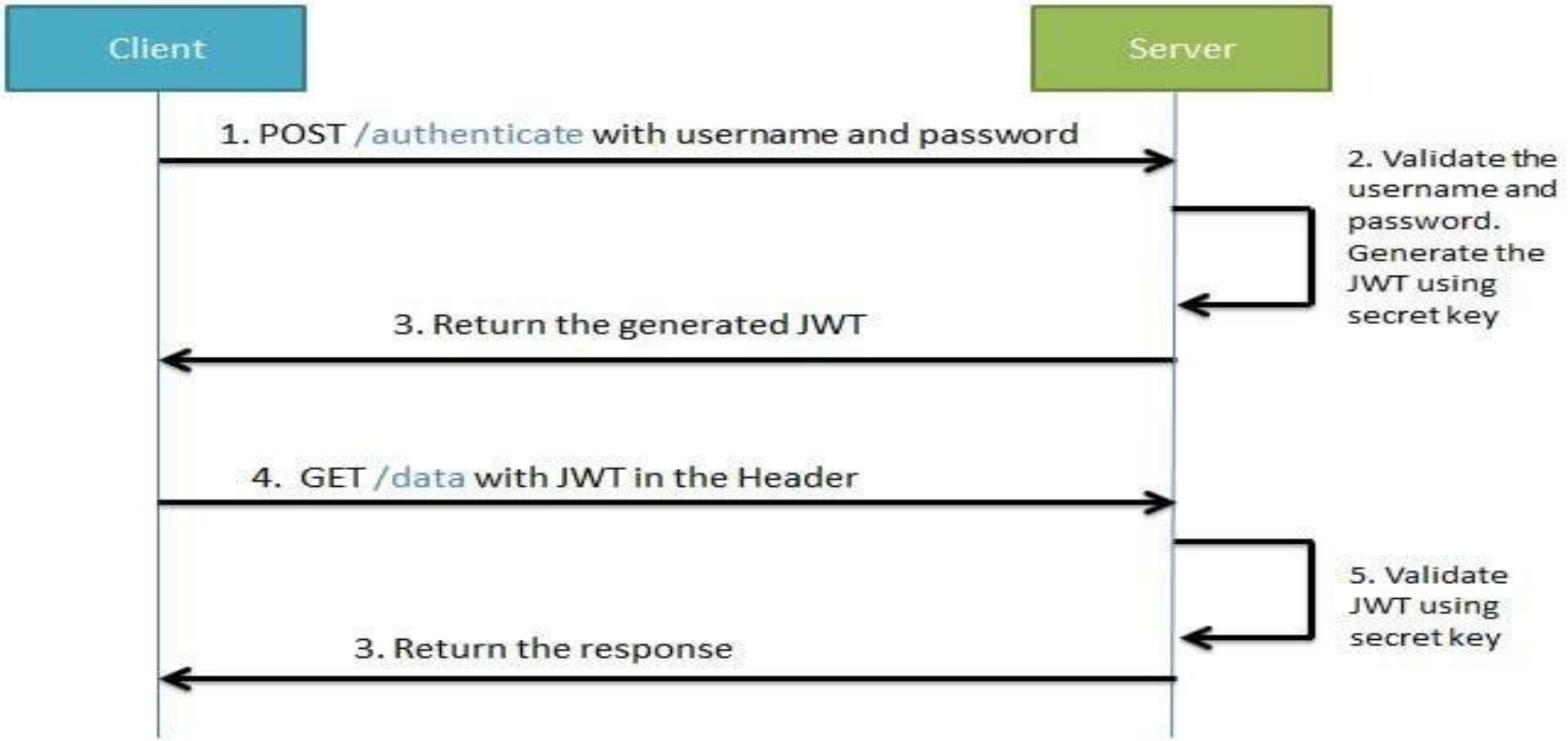



```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

```
{
  "iss": "Online JWT Builder",
  "iat": 1600999300,
  "exp": 1632535300,
  "aud": "www.my-app.com",
  "sub": "anil.jos@gmail.com",
  "GivenName": "Anil",
  "Surname": "Joseph",
  "Email": "anil.jos@gmail.com",
  "Role": "Consultant"
}
```


Signature

JWT with Simple Forms Login



Testing

Tools

Jasmine

- Library for Unit Tests

Karma

- TestRunner

Protractor

- Library for End to End Tests

Jasmine

- Jasmine is a behavior-driven development framework for testing JavaScript code.
- It does not depend on any other JavaScript frameworks.
- It does not require a DOM.

Jasmine

- Suites
  - Describe your tests
  - A test suite begins with a call to the global Jasmine function ***describe***
- Specs
  - The tests
  - Specs are defined by calling the global Jasmine function ***it***
  - A spec contains one or more expectations.
- Expectations
  - An expectation in Jasmine is an assertion that is either true or false.
  - Expectations are built with the function ***expect.***
- Matchers
  - A matcher implements a Boolean comparison between the actual value and the expected value.
  - Jasmine has a rich set of matchers included.

115

Jasmine

- beforeEach
  - called once before each spec in the describe in which it is called,
- afterEach
  - The afterEach function is called once after each spec.
- beforeAll
  - The beforeAll function is called only once before all the specs in describe are run
- afterAll
  - The afterAll function is called after all specs finish.

116

Angular Testing Utilities

- TestBed
  - TestBed is the first and most important of the Angular testing utilities.
  - It creates an Angular testing module
    - an @NgModule class
  - Use the configureTestingModule method to create the module.
  - The configureTestingModule method takes an @NgModule-like metadata object.
- ComponentFixture
  - A handle on the test environment surrounding the created component.
  - The fixture provides access to the component instance itself and to the DebugElement.
- DebugElement
  - A handle on the component's DOM element.

E2E Tools

Cypress

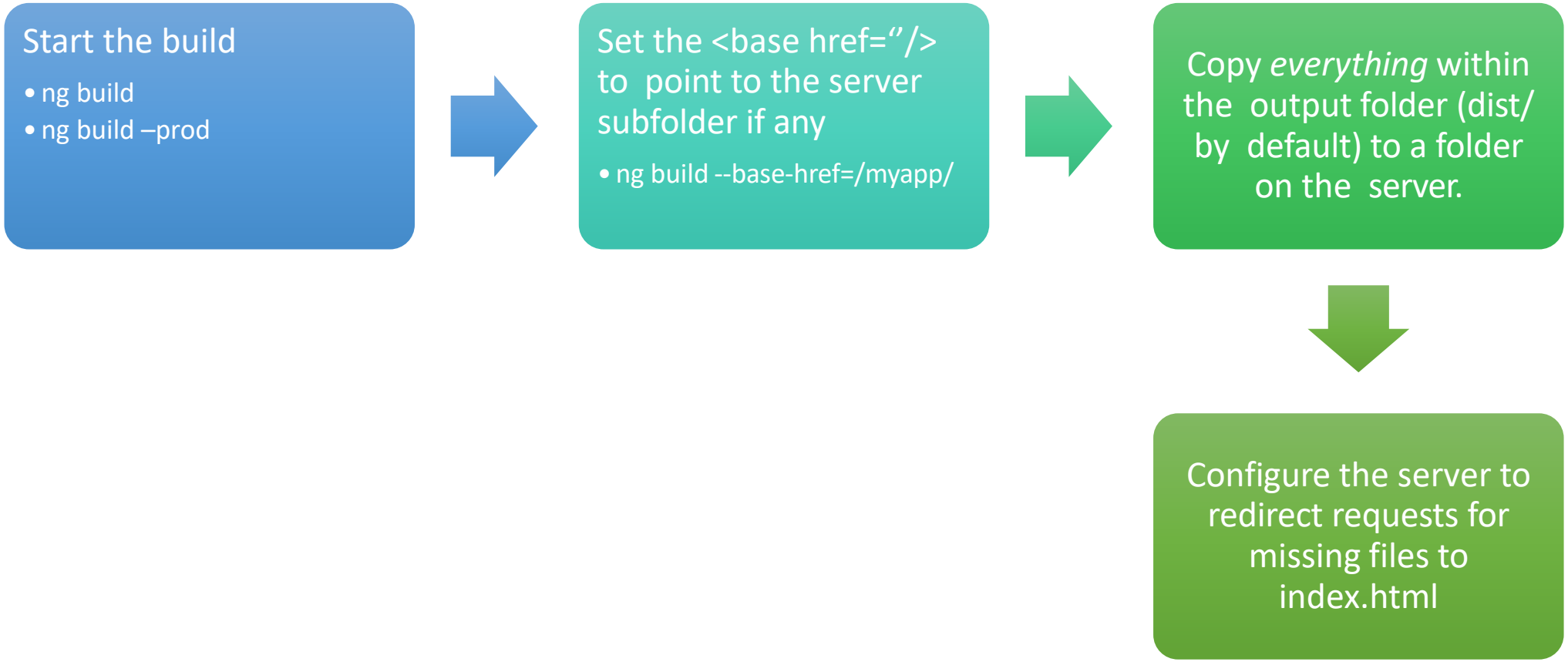
Pupeeter

Selenium Webdriver

Nightwatch.js

Presented By Anil Joseph(anil.jos@gmail.com)

Deployment


Start the build
• ng build
• ng build –prod
Set the <base href="/> to point to the server subfolder if any
• ng build --base-href=/myapp/
Copy *everything* within the output folder (dist/ by default) to a folder on the server.
Configure the server to redirect requests for missing files to index.html

Build for Production

ng build --prod

Ahead-of-Time (AOT) Compilation	• Pre-compiles Angular component templates.
Production mode	• Deploys the production environment which enables production mode.
Bundling	• Concatenates your many application and library files into a few bundles.
Minification	• Removes excess whitespace, comments, and optional tokens.
Uglification	• Rewrites code to use short, cryptic variable and function names.
Dead code elimination	• Removes unreferenced modules and much unused code.

Features

- Angular Elements,
- Service Workers,
- RxJS 6.0
- Drag & Drop,
- Virtual Scrolling
- Ivy Compiler,
- Differential Loading
- Web Workers
- Hot Module Replacement

Resources

- Angular
  - https://angular.io
- Books
  - https://angular.io/resources?category=education

Thank You
Email: anil.jos@gmail.com
WhatsApp: 9833169784

ANIL JOSEPH

THANK YOU

MALAYSIA

Timmins Training Consulting

C-11-01 Komplek Danau Kota,

Taman Zeta@Zetapark, 67, Taman

Ibu Kota, Setapak 53300 KL

www.timmins-consulting.com

raj@consult-timmins.com

INDONESIA

PT Timmins Konsultan Utama

Graha Mampang Lt. 3 Suite 305 Jl.

Mampang Prapatan Raya Kav. 100

Jakarta Selatan 12760

www.timmins-consulting.com

dhira@consult-timmins.com

CANADA

Timmins Training Consulting Inc.

Suite #1203,

1, Reidmount Avenue, Toronto,

Ontario, M1S 4V3

www.timmins-consulting.com

raj@consult-timmins.com