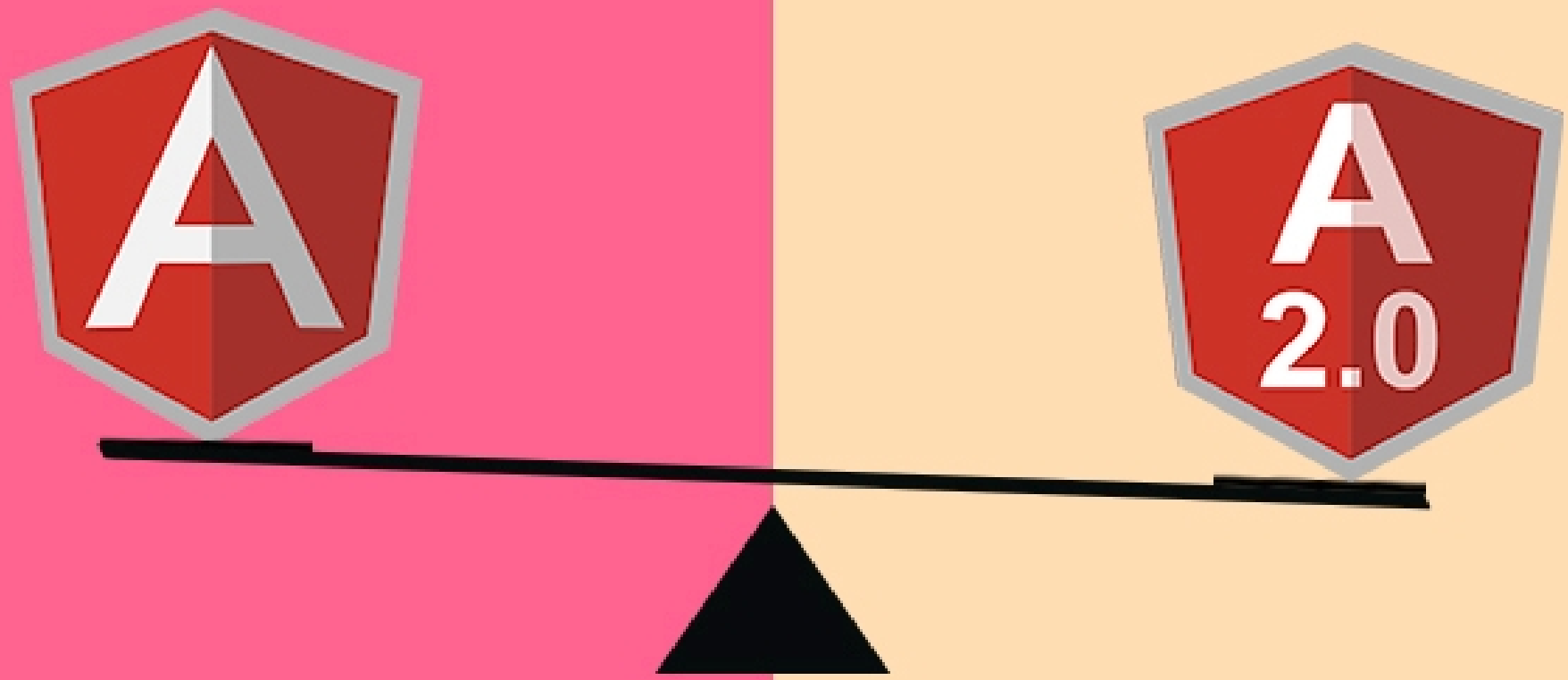
# Angular VS Angular 2




TECHMEET 360

# Чим є Angular?

- Frontend/клієнтський фреймворк JavaScript
- Створено та підтримується Google
- Використовується для створення потужних односторінкових додатків (SPA)
- Частина дуже потужного стека MEAN

# Чим angular не є

- Серверним фреймворком/технологією
- Javascript бібліотекою (jQuery, React, etc)
- Шаблоном оформлення
- Платформою або мовою (.NET, Java)
- Плагіном або розширенням

# Історія версій Angular

**AngularJS/Angular 1**

**Angular 2 :** Заново переписаний Angular JS

**Angular 3 :** Пропущений

**Angular 4 :** Зворотно сумісний з Angular 2

**Angular 5, Angular 6, Angular 7**

**Angular 8:** запланований на березень-квітень 2019 року

# Будівельні блоки Angular

Головними будівельними блоками Angular є:

- Модулі
- Компоненти
- Шаблони
- Метадані
- Прив’язки даних
- Директиви
- Сервіси
- Ін’єкції залежностей

# Модулі

Angular програми є модульними і для підтримки модульності ми маємо модулі Angular або можна сказати NgModules. Кожне додаток Angular містить принаймні один модуль Angular ( кореневий ). Як правило, він називається AppModule. Кореневий модуль може бути єдиним у невеликому додатку. Хоча більшість програм мають кілька модулів. Можна сказати, що модуль — це згуртований блок коду з відповідним набором можливостей, які мають певний домен програми або робочий процес. Будь-який Angular модуль - це клас з декоратором @NgModule.

Декоратори - це функції, які змінюють класи JavaScript.

Декоратори в основному використовуються для приєднання метаданих до класів, так що він знає конфігурацію цих класів і те, як вони повинні працювати.

NgModule — це декоратор, який приймає об'єкт метаданих, властивості якого описують модуль.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

@NgModule({
imports:[ BrowserModule ],
providers: [ BookList ],
declarations: [ AppComponent ],
exports: [],
bootstrap: [ AppComponent ]
})

export class AppModule { }
```

# Компоненти

Компонент керує однією або кількома секціями на екрані, який називається переглядом. Наприклад, якщо ви створюєте програму списку фільмів, ви можете мати такі компоненти, як App Component, компонент Movielist, компонент опису фільму тощо.

Усередині компонента ви визначаєте логіку програми компонента, тобто як він підтримує перегляд - всередині класу. Клас взаємодіє з видом через API властивостей і методів.

Кожний додаток має основний компонент, який завантажується в основний модуль, тобто AppComponent.

```
import { Component } from '@angular/core';
@Component({
selector:'app-root',
templateUrl:'./app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
title = 'app works!';
}
```

# Шаблони

Кожен компонент пов’язується з своїм шаблоном.

Шаблон - це не що інше, як форма HTML-тегів, яка говорить Angular про те, як рендерувати компонент. Шаблон виглядає як звичайний HTML, за винятком декількох відмінностей.

```
<app-navbar></app-navbar>


<div class ="container">
<flash-messages></flash-messages>
<router-outlet></router-outlet>
</div>
```

## Метадані

Метадані повідомляють Angular, як обробляти клас. Щоб сказати Angular, що компонент MovieList є компонентом, метадані прикріплюються до класу. У TypeScript ви додаєте метадані за допомогою декоратора.

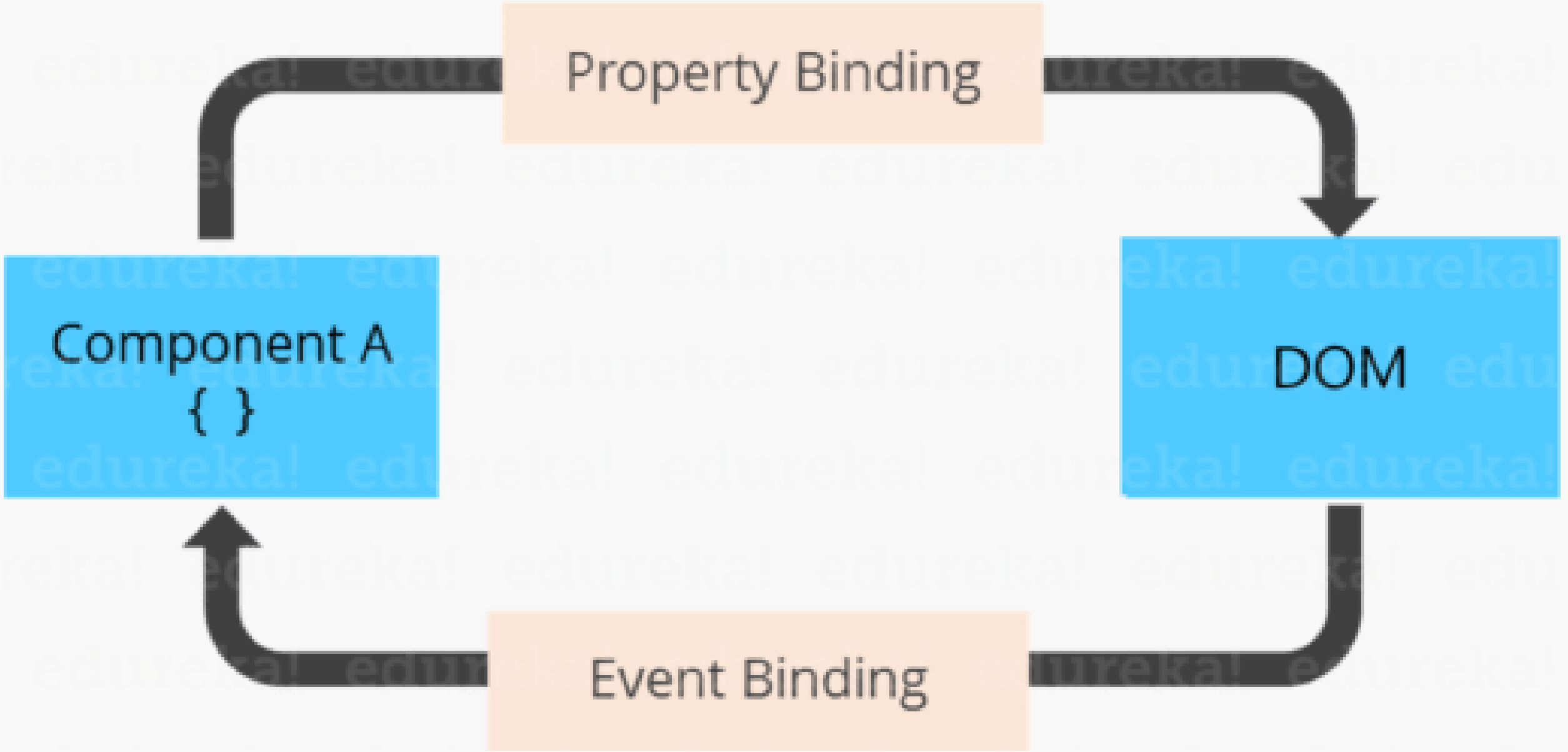
```
import { Component, OnInit } from '@angular/core';

@Component({
selector: 'app-movies',
templateUrl: './movies.component.html',
styleUrls: ['./movies.component.css']
})
```

# Прив'язка даних

Якщо  не використовувати фреймворк, необхідно відправляти значення даних до елементів керування HTML і перетворювати відповіді користувача на деякі дії та оновлення значень. Написання такої логіки push / pull  втомливе, схильне до помилок і є кошмаром для читання. Angular підтримує прив'язку даних - це механізм для узгодження частин шаблону з частинами компонента. Потрібно додати розмітку прив'язки до шаблону HTML, щоб сказати Angular як підключити обидві сторони.

edureka!



```
<li> {{movie.name}}</li>
<movie-detail [movie]="selectedMovie"></movie-detail>
<li (click)="selectMovie(Movie)"></li>
```

Інтерполяція {{movie.name}} відображає значення властивості імені компонента в елементі <li>.

Прив'язка властивості [movie] передає значення selectedMovie від батьківського MovieListComponent до властивості фільму дочірнього MovieDetailComponent.

Прив'язка події (клік) викликає метод selectMovie компонента, коли користувач натискає назву фільму.

# Директиви

Директива - це клас з декоратором @Directive. Компонент є директивою з шаблоном; декоратор @Component фактично є декоратором @Directive, розширеним за допомогою шаблонів-орієнтованих функцій.

Незважаючи на те, що компонент є технічно директивою, вони настільки важливі, що їх можна відокремити.

Існують ще два види директив: структурні й атрибутивні.

Директиви мають тенденцію з'являтися в тезі елементів, як це роблять атрибути, іноді по імені, але частіше як задання призначення або прив'язки.

Структурні директиви змінюють макет шляхом додавання, видалення та заміни елементів у DOM. Цей прикладний шаблон використовує дві вбудовані структурні директиви:

```
<li *ngFor="let movie of movies"></li>
<movie-detail *ngIf="selectedMovie"></movie-detail>
```

* ngFor розповідає Angular, щоб отримати по одному <li> на фільм у фільмах
* ngIf включає компонент MovieDetail, лише якщо існує вибраний фільм.

Директиви атрибутів змінюють зовнішній вигляд або поведінку існуючого елемента. У шаблонах вони виглядають як звичайні атрибути HTML. Як приклад, ngModel яка реалізує двосторонню прив'язку даних. ngModel змінює поведінку існуючого елемента, встановлюючи його властивість відображуваних значень і відповідаючи на зміни подій.

Angular має ще кілька директив, які або змінюють структуру компонування (наприклад, ngSwitch), або змінюють аспекти елементів і компонентів DOM (наприклад, ngStyle і ngClass).

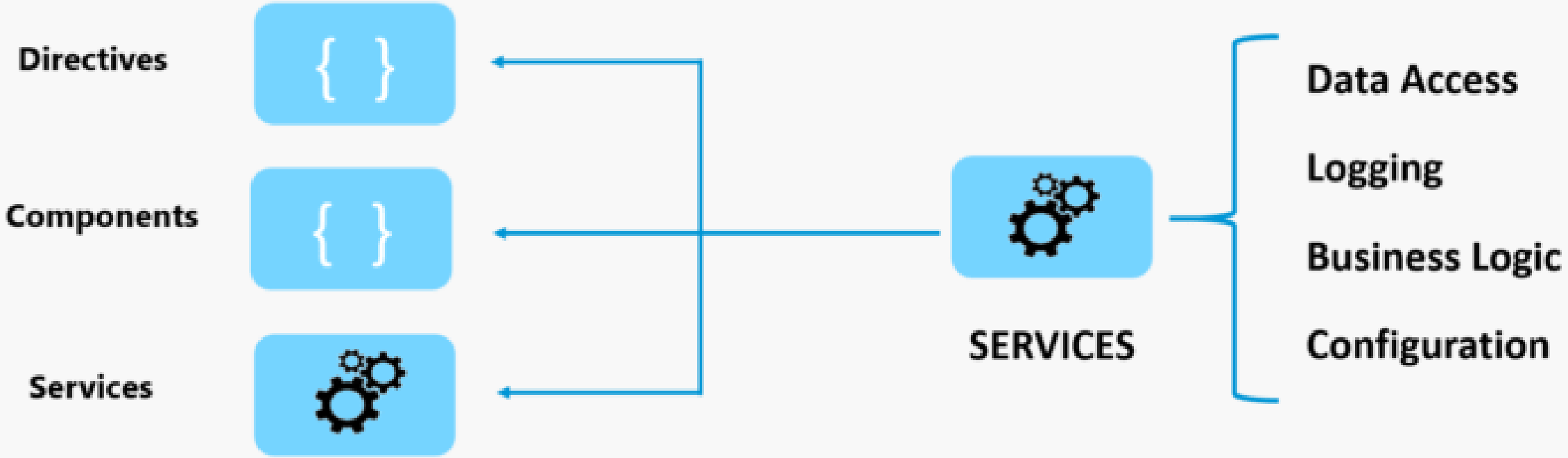
Ви також можете написати власні директиви.

# Сервіси

Сервіси - це широка категорія, яка охоплює будь-яку цінність або функцію, яку потребує ваша програма. Сервіс зазвичай є класом з чітко визначеною метою. Все може бути сервісом
Приклади:

- служби реєстрації
- послуги передачі даних
- шина повідомлень
- податковий калькулятор
- конфігурації програми

edureka!

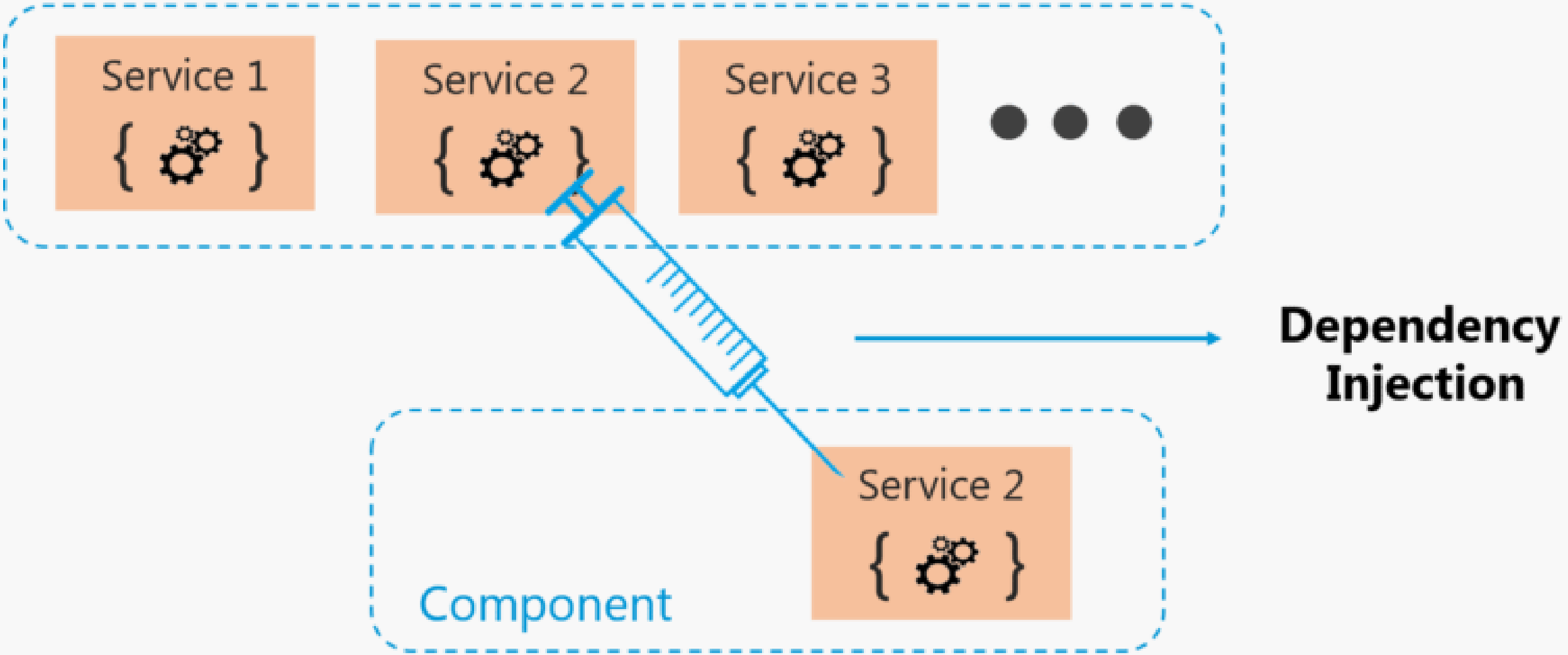

Directives
Components
Services
SERVICES
Data Access
Logging
Business Logic
Configuration

# Ін'єкція залежностей

Ін'єкція залежностей - це спосіб подачі нового екземпляра класу з повністю сформованими залежностями, які він вимагає. Більшість залежностей є сервісами. Angular використовує ін'єкції залежностей для забезпечення нових компонентів необхідними послугами. Angular може визначити, які послуги потребує компонент, розглядаючи типи параметрів конструктора.

Коли Angular створює компонент, він спочатку запитує інжектор для служб, які вимагає компонент.

edureka!




**Dependency Injection**

Дякую за увагу.