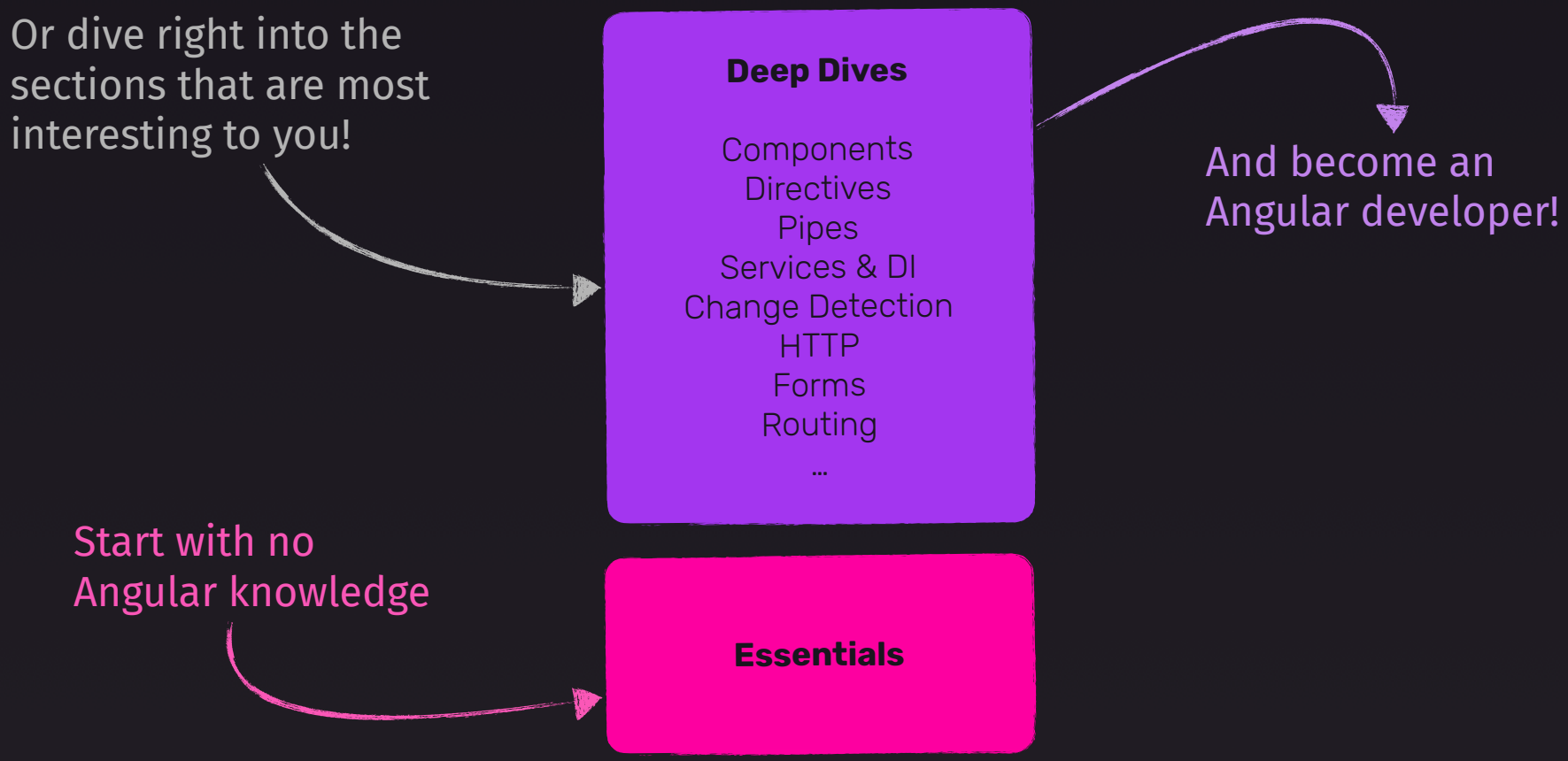
Angular — The Complete Guide

A Web Framework

For Building Highly Interactive Web Applications


Or dive right into the sections that are most interesting to you!
Deep Dives
Components
Directives
Pipes
Services & DI
Change Detection
HTTP
Forms
Routing
...
And become an Angular developer!
Start with no Angular knowledge
Essentials

Maximilian Schwarzmüller

Professional Web Developer & Course Instructor

@maxedapps

maximilian-schwarzmueller.com

academind.com

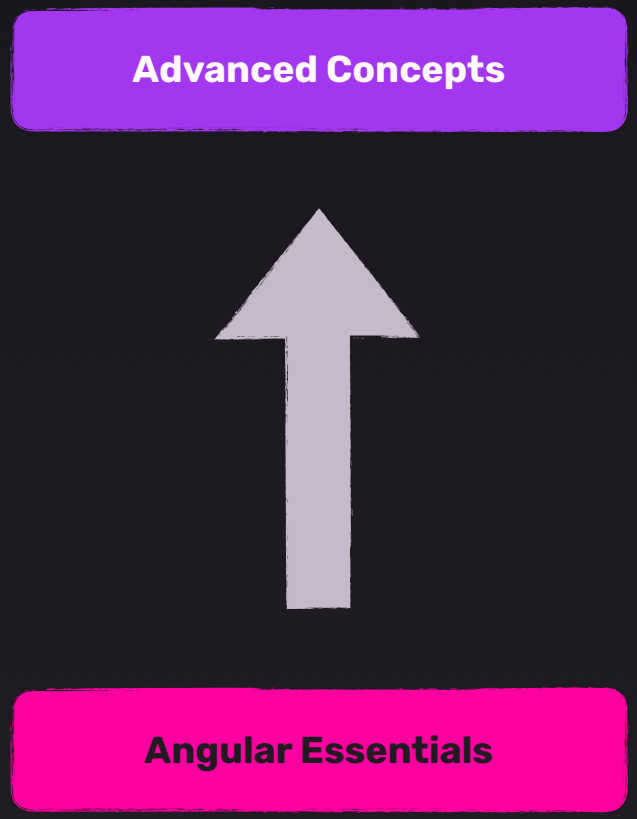
Welcome To The Course

Maximilian Schwarzmüller

@maxedapps

maximilian-schwarzmueller.com

academind.com


Advanced Concepts
Angular Essentials

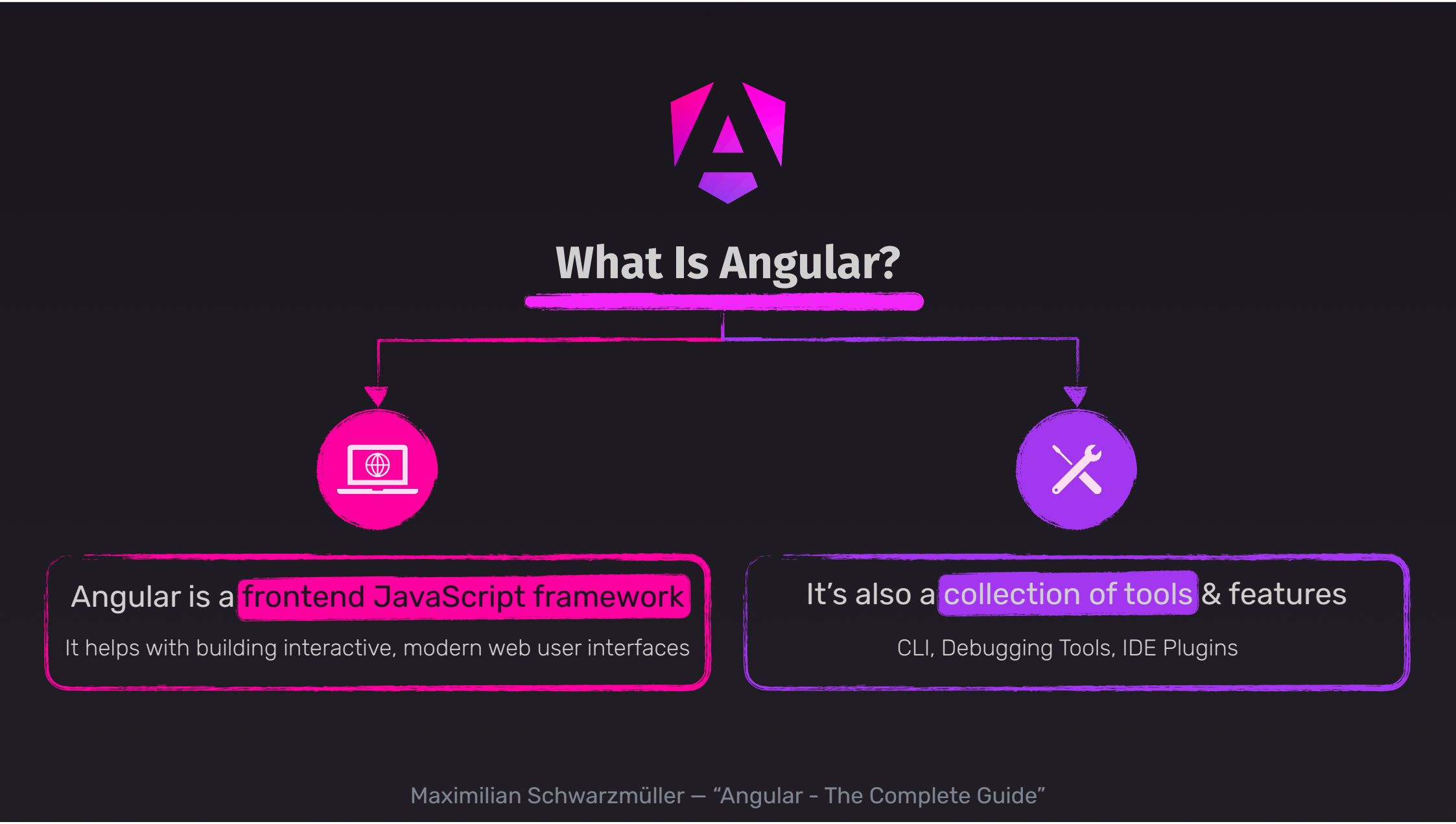
No prior Angular

experience required!

Basic web development & JavaScript
experience is all you need

What Is Angular?

And why would you use it?



Maximilian Schwarzmüller — "Angular - The Complete Guide"

Why would you use Angular?

Why a framework?

You don’t need it!

For trivial websites and web apps!

But that changes as your applications get more complex!

Why a framework?

Simplifies the process of building complex, interactive web user interfaces

Write **declarative code**

Separation of concerns via **components**

Object-oriented programming concepts & principles

Use TypeScript

Maximilian Schwarzmüller — “Angular - The Complete Guide”

With “Just JavaScript”

You write **imperative** code!

Step-by-step instructions (in code) that tell the browser what to do

Example

1) Find DOM element & store in variable
2) Add event listener to element
3) In triggered function: Find another element
4) Set CSS visibility of that element to 0
5) Create a new <p> DOM element
6) Set text content of <p> to “Hello World!”
7) Find element into which the <p> should be inserted
8) Insert <p> element

And so on …

With Angular

You write **declarative** code!

Define the target UI states & how they change and let Angular do the rest

Example

1) Manage “activeTab” state property
2) Depending on “activeTab”, show different content
3) Change “activeTab” upon click on tab element

Why Components?

Modular Applications

Break up complex applications into simple building blocks

Split up responsibilities & concerns

Build a component once and re-use it as often as needed

Better Development Process

Assign different team members to different components

Share components & logic across the team

Reduce dependencies

Using TypeScript

TypeScript is a JavaScript superset — an extension of JavaScript

It extends the JavaScript syntax to support strong & strict typing

Unlike “vanilla JavaScript”, TypeScript enforces types and prevents unexpected value type changes

TypeScript is not an Angular feature but can be used independently — Angular embraces it though, hence all Angular projects use TypeScript

Maximilian Schwarzmüller — “Angular - The Complete Guide”

Easier Development With TypeScript

With TypeScript, you can often catch errors early on during development

For this course, you don't need to know TypeScript!

You'll **learn it along the way** and there also is an **optional "TypeScript Crash Course" section** at the end of the course — you can always go through that section if you're struggling to follow along with TypeScript

Why Angular?

Highly scalable, well maintained, innovative & batteries included

Developed & used by **Google**

Improvements, fixes & features are added continuously

The framework keeps **evolving**

Backward-compatibility & stability is a number 1 priority

It's a all-in-one package — lots of **core features are built-in**

Maximilian Schwarzmüller — "Angular - The Complete Guide"

Angular: A Stable Yet Evolving Framework

Angular has backward-compatibility as a number 1 priority

At the same time, the Angular keeps innovating & aims to improve and advance the framework over time

Angular: A Stable Yet Evolving Framework

2016 Angular 2 is released

2017 Angular 4 is released (v3 is skipped)

2017 Angular 5 is released

... Angular 6 - 13 are released

2022 Angular 14 & 15 are released → Angular 14 introduces “Standalone Components”

2023 Angular 16 is released → Angular 16 introduces “Signals”

The Angular team has a versioning & release policy where a **new major version** is released **every ~6 months**

BUT: Despite labeled as major versions, these releases do NOT break your code & “change the framework”

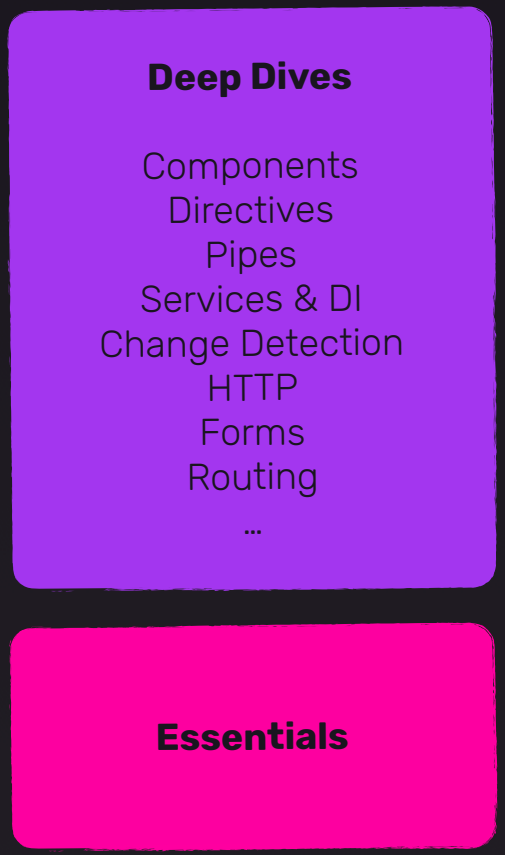
Angular keeps innovating & changing - with backward compatibility as a top priority

Maximilian Schwarzmüller — “Angular - The Complete Guide”

Different Projects, Different Versions

Creating an Angular Project

About This Course



This Course Kind Of Includes Two Courses

This course was re-recorded & I'm currently keeping the old course online

Modern Angular

Covers key features like standalone components & signals right from the start

Still also teaches you "older" Angular features since you might need them for some projects

Simply continue with the next lectures & sections to take this course!

Legacy Angular

Does NOT use standalone components & signals

Instead uses "older" Angular features & syntax that's still valid but not the most modern way of writing Angular code

Use the attached link to jump ahead to the "old course" if you need to learn about this older version

You can ignore the course you're not taking — simply mark lectures as completed manually or watch at double speed to quickly pace through it (to get 100% course completion)

Angular Essentials

Understanding Angular Project Structure

Working with Components

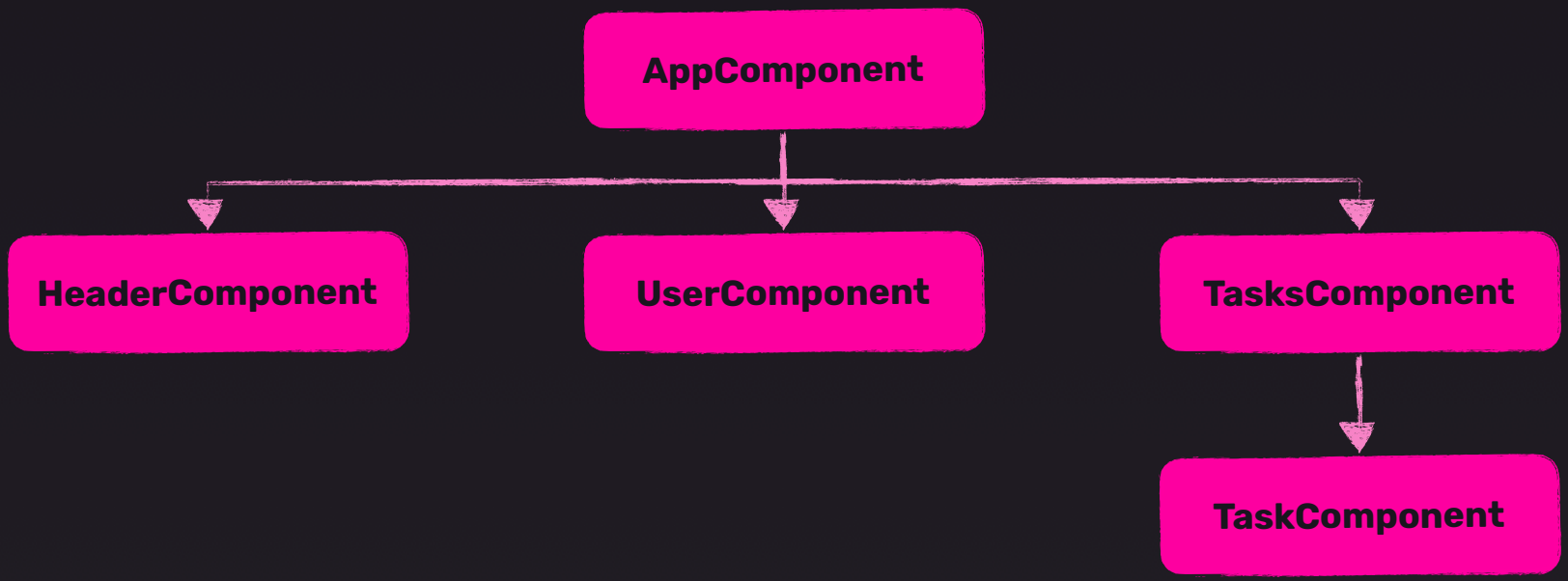
Handling User Events

Rendering & Updating Dynamic UI Content

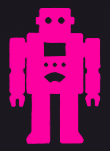
Maximilian Schwarzmüller — “Angular - The Complete Guide”

You Build A Component Tree

One Angular Application = One Component Tree



There Are Two Approaches For Updating State



Option 1

Relying on Zone.js & Angular's change detection mechanism

Works automatically, no special instructions required

Supported since Angular 2

Option 2

Using Signals to notify Angular about value changes & required UI updates

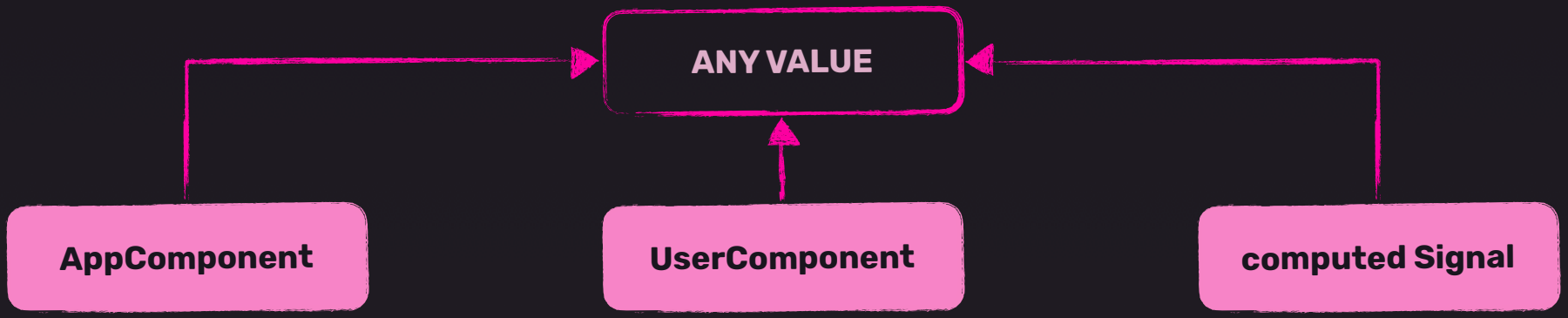
Requires usage of special "signal" instructions & code

Supported since Angular 16

Signals Are Trackable Data Containers

Signal

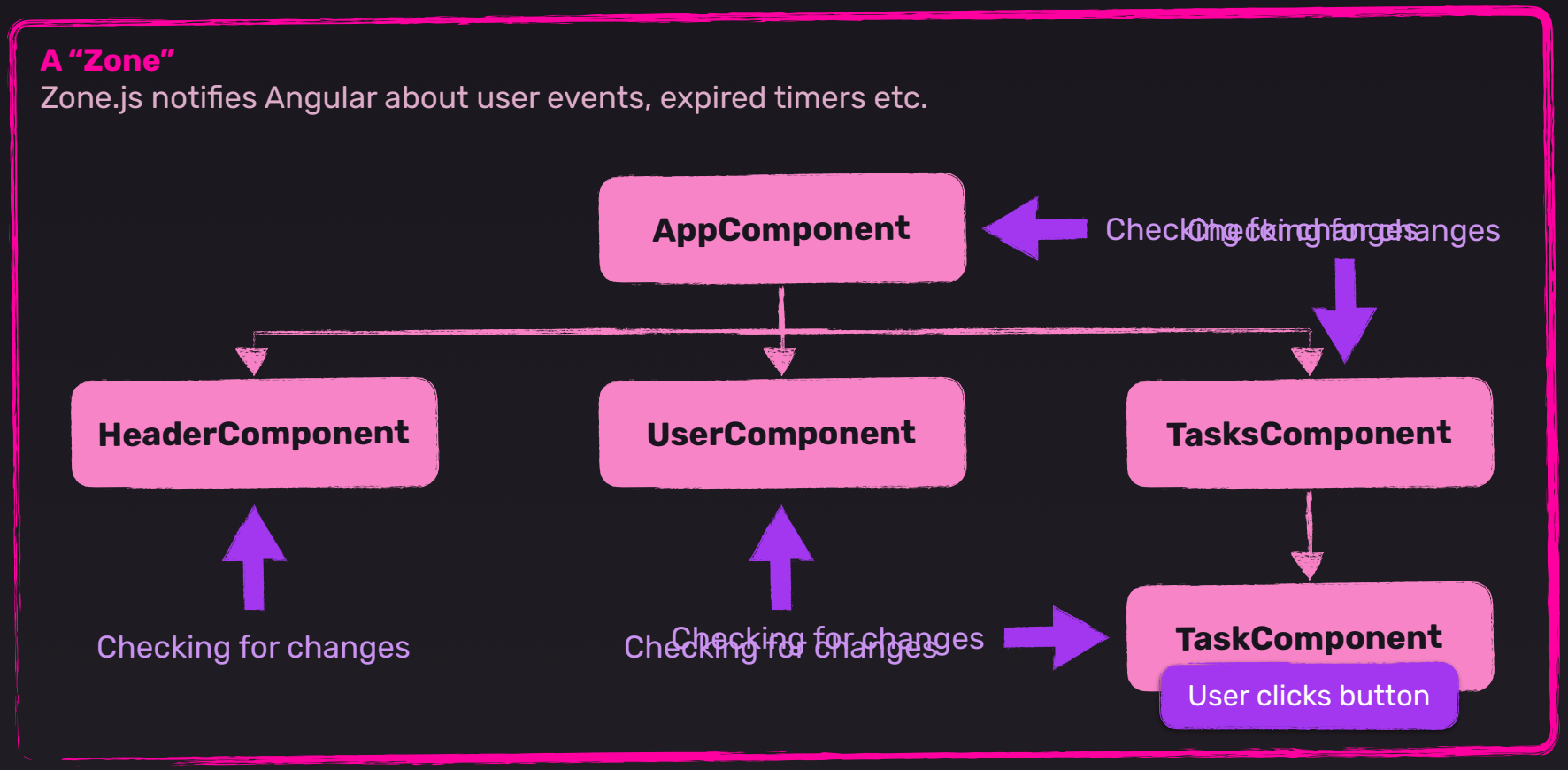
A signal is an object that stores a value (any type of value, including nested objects)



Angular manages subscriptions to the signal to get notified about value changes

When a change occurs, Angular is then able to update the part of the UI that needs updating

Angular's Change Detection Mechanism

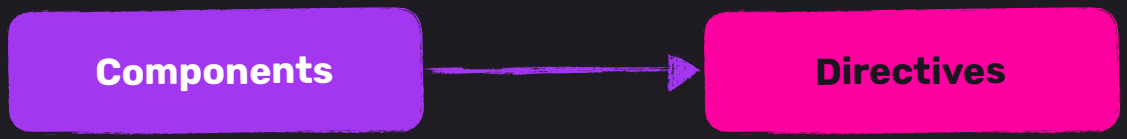


Directives

With Angular, you can "enhance" elements by adding so-called Directives to them

An "element enhancement" that helps with extracting (or changing) user input values

Directives, unlike components, don't have a template!



Components are directives! Directives with templates

Exercise

Create and use a “Tasks” component (`app-tasks`)

Receive & output the name of the selected user

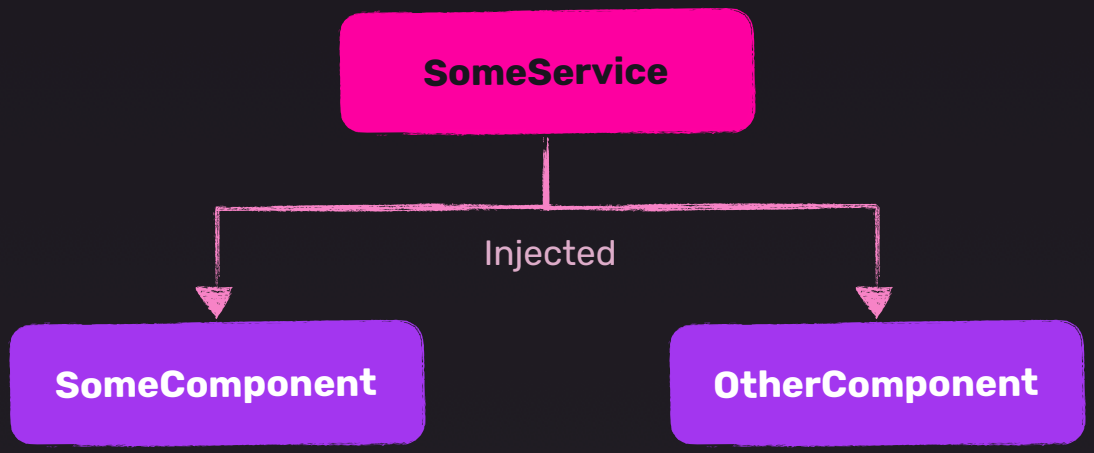
HINT

You can find a user with a specific id in the `DUMMY_USERS` array like this:

```
DUMMY_USERS.find(user => user.id === selectedUserId)!
```

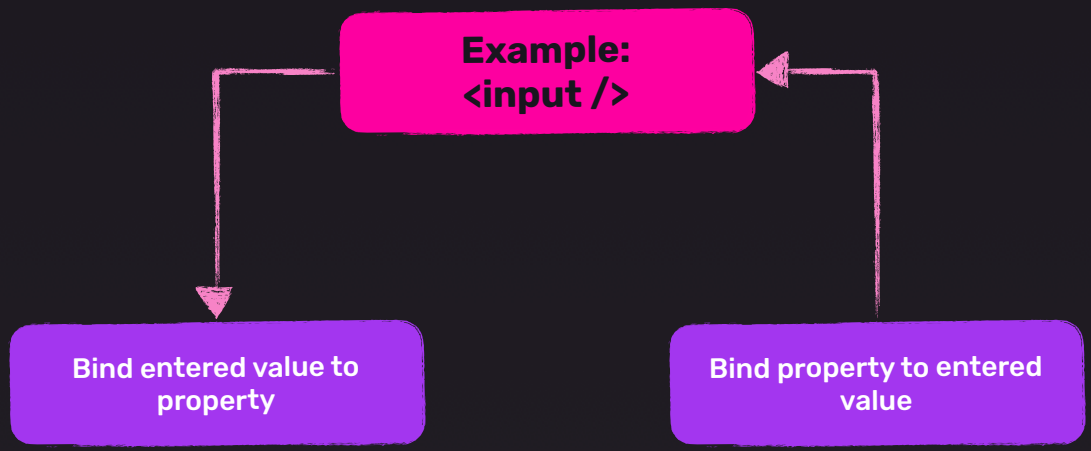
Services

For centralized & sharable data- and logic-management



Two-Way-Binding

Change data AND listen to data changes at the same time



Angular Modules

Understanding Angular Modules (NgModule)

Using Modules Instead Of Standalone Components

Declaring & Exporting Components

Shared Modules

Angular Modules Exist For Historic Reasons

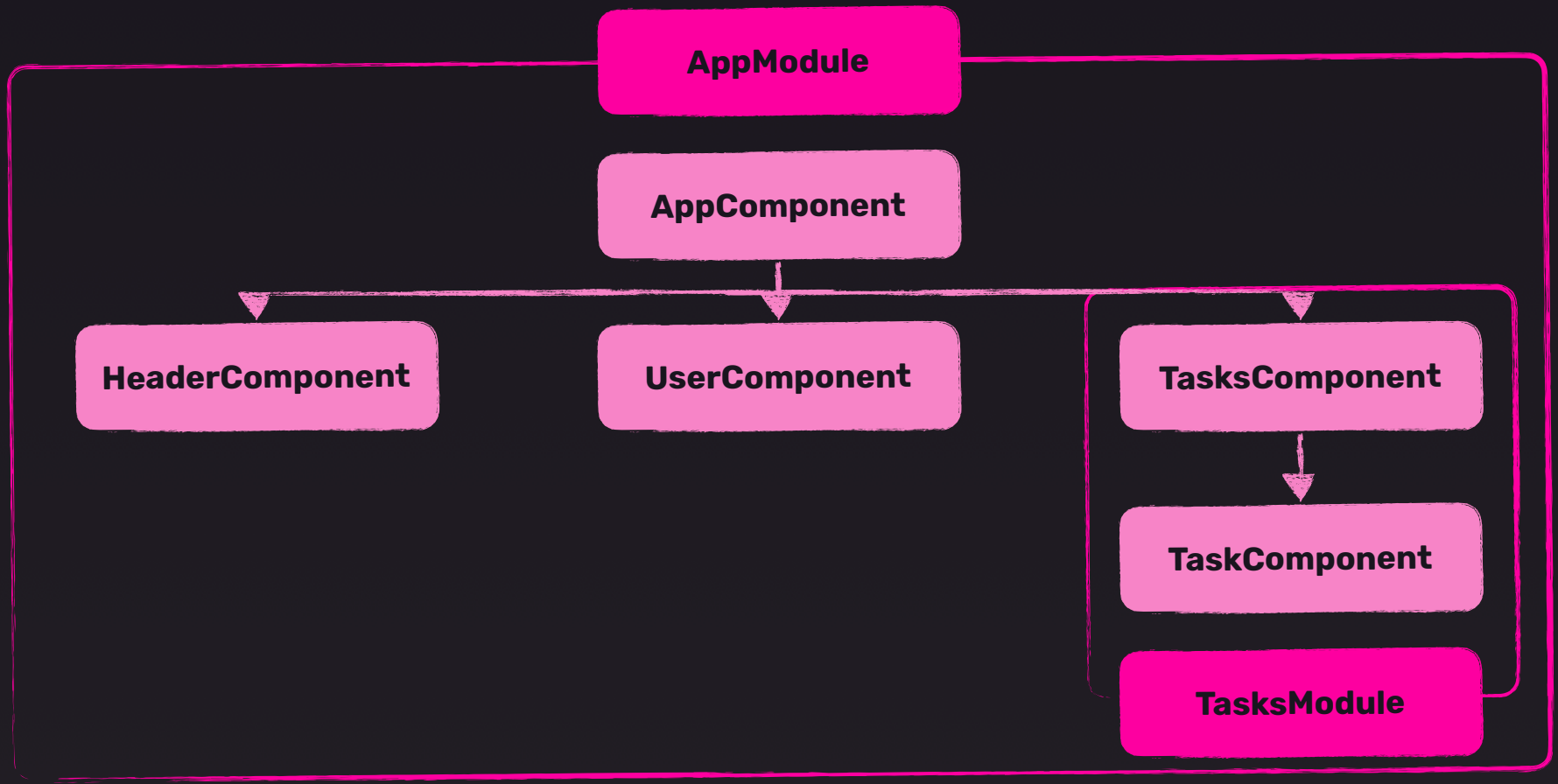
When Angular 2 was released in 2016, there were no “Standalone Components”

Today, Standalone Components are the recommended way of building components

But you can also still use “Module-based Components” with Angular Modules

Understanding Angular Modules

Angular Modules “make components (& more) available to each other”



Angular Debugging

Understanding & Fixing Error Messages

Debugging Logical Errors

Using the Angular Developer Tools

Angular Essentials - Practice Project

Apply Your Knowledge & Practice What You Learned

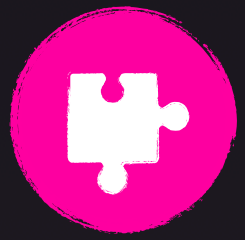
Maximilian Schwarzmüller — “Angular - The Complete Guide”

Exercise

Build an “Investment Calculator” Application

1. Add a “Header” component with a title & image
2. Add a “User Input” component that collects user input with two-way-binding
3. Add an “InvestmentResults” component that outputs results in a table

You can use signals or not
You can use standalone components or module-based components



Components Deep Dive

Advanced Components Features & Concepts

Working with the Host Element

Inputs, Outputs & Two-Way-Binding

Interacting with Component Views & Content

Component Lifecycle

Maximilian Schwarzmüller — “Angular - The Complete Guide”

When should you split a component?

It’s up to you!

Separation of Concerns

Every component should only do “one thing”

It’s a trade-off

Separation of Concerns

vs

Simplicity & Code Colocation

Enums

TypeScript Feature

Enums are “groups of allowed values”

```
enum ViewEncapsulation {
  Emulated = 0,
  None = 2,
  ShadowDom = 3
}
```

Internally, the pre-defined allowed values (e.g., “Emulated”) map to integers which can be used by Angular as identifiers

The Shadow DOM

A browser feature that allows you to attach hidden DOM structures to DOM elements

Example

The built-in `<video>` element hides a more complex DOM tree that's used internally

For CSS styling, the Shadow DOM can be used to scope CSS styles to that hidden tree — instead of applying styles globally to the entire page.

Angular can emulate this Shadow DOM browser feature for its own components.

Component Host Elements

Every Angular component has a Host Element

Example

A component with a selector of "`app-header`" targets an `<app-header>` element which is rendered into the real DOM

Important: The elements targeted by your component selectors **do NOT** act as placeholders and **are NOT** replaced when the page is rendered!

Instead, the selected elements are **preserved** and simply **"enhanced" / taken over** by your component logic & markup!

Directives Deep Dive

Directives vs Components

Attribute Directives

Structural Directives

Built-in Directives

Building Custom Directives

Maximilian Schwarzmüller — “Angular - The Complete Guide”

What are Directives?

What are Directives?

Directives are **“enhancements”** for elements (built-in ones or components)

Example

```
<input name=“title” ngModel />
```

They can change the configuration (properties, attributes), styling or behavior of elements.

Unlike Components, **Directives have no template!**

In other words: Components are Directives with a template.

Two Types Of Directives

Attribute Directives

Do **not** directly change the DOM structure

They may change the attributes, properties or behavior of an element

Example

```
<p [ngClass]="{active: isActive}">…</p>
```

Structural Directives

Do directly change the DOM structure

They are applied on `<ng-template>` elements or via the `*` syntactic sugar

Example

```
<p *ngIf="someCondition">…</p>
```

Pipes Deep Dive

What are Pipes?

Built-in Pipes

Building Custom Pipes

Pure vs Impure Pipes

What are Pipes?

Pipes transform the way data is displayed

Services & Dependency Injection Deep Dive

Revisiting Services

Revisiting Dependency Injection (DI)

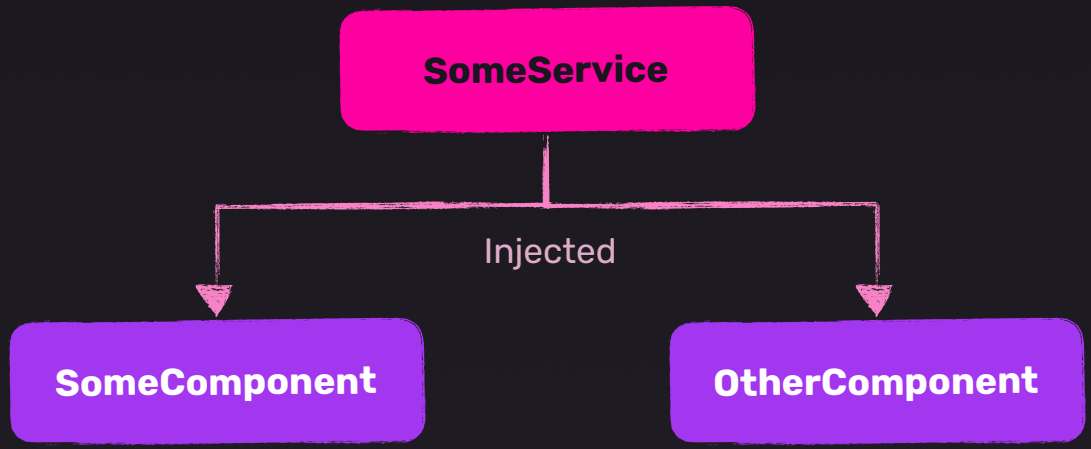
Hierarchical Injectors & DI Resolution Process

Injection Tokens & Values

Maximilian Schwarzmüller — “Angular - The Complete Guide”

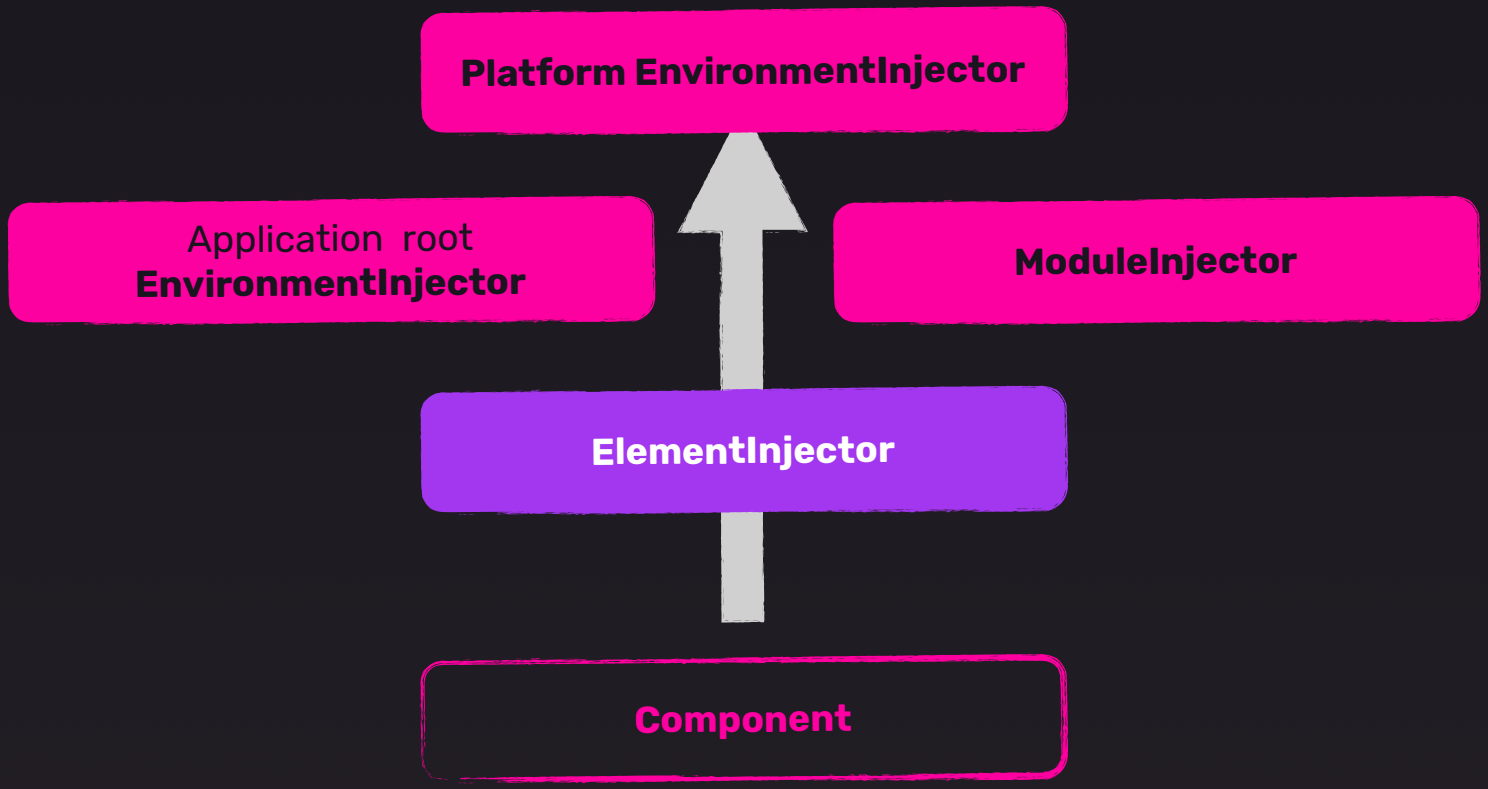
Understanding Services

Services allow you to **share logic and data** across the application



Understanding Dependency Injection

You don't create service instances yourself — instead, you **request them from Angular**



Maximilian Schwarzmüller — "Angular - The Complete Guide"

Understanding Change Detection

What is Change Detection?

Understanding Angular's Change Detection Mechanism

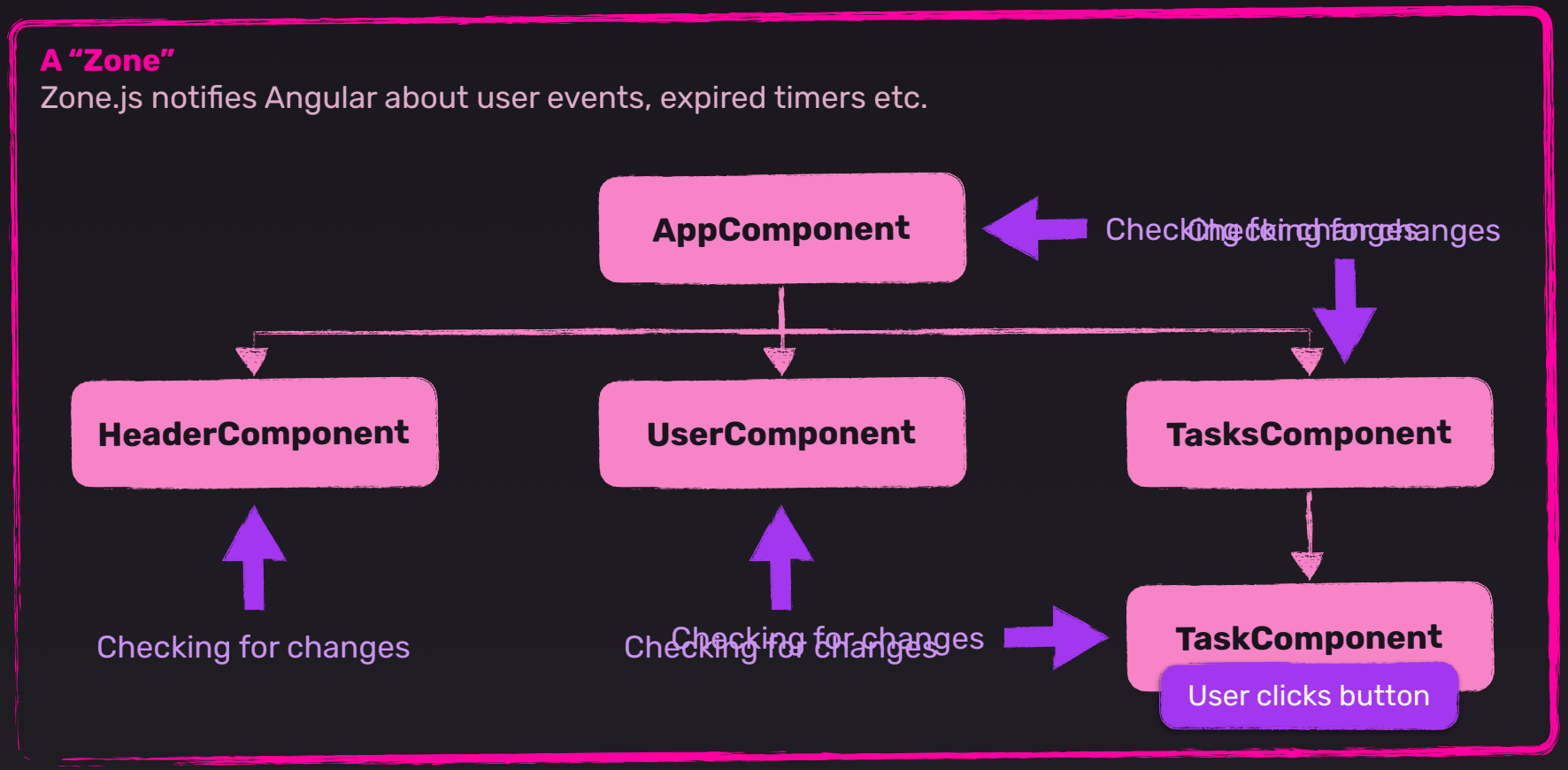
Using the OnPush Strategy

Change Detection & Signals

Going Zoneless

Maximilian Schwarzmüller — "Angular - The Complete Guide"

Angular's Change Detection Mechanism



The “OnPush” Strategy

RxJS & Observables

What are Observables?

Creating & Using Observables

Observable Operators

Observables vs Signals

What are Observables?

It’s a concept provided by RxJS

NOT by Angular!

A Stream Of Data

Observables

Values over time

Signals

Values in a container

Observables

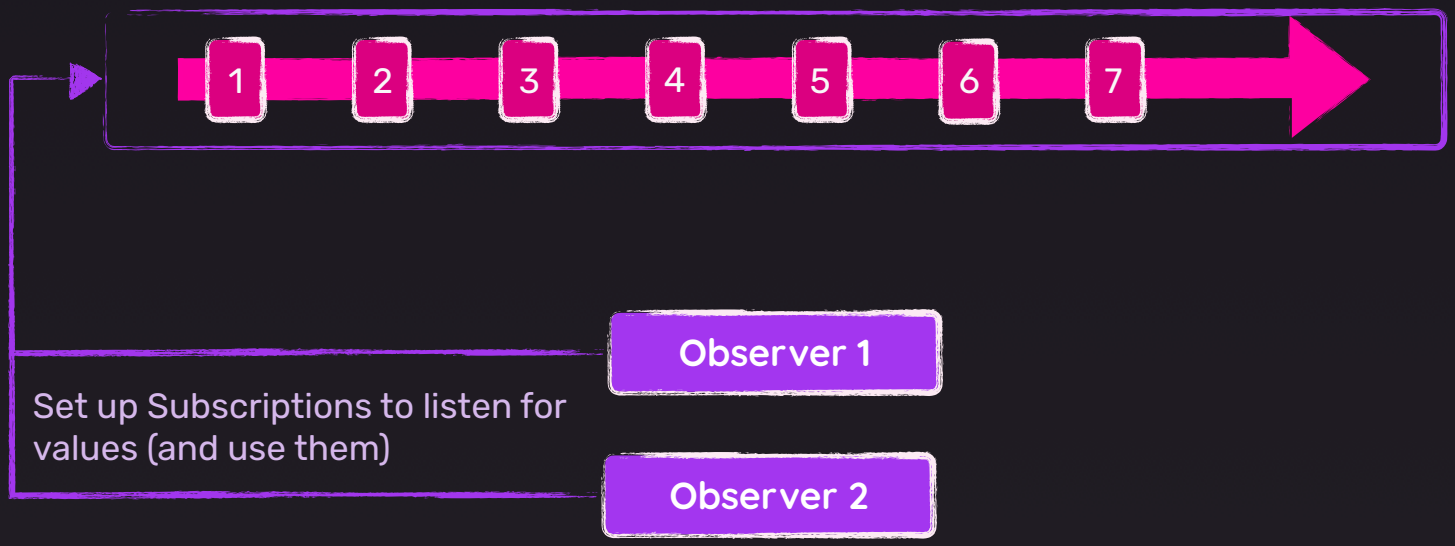
Great for managing events & streamed data

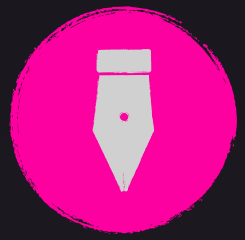
Signals

Great for managing application state

A Stream Of Data

RxJS Observables emit values over time — you can set up subscriptions to handle them





Managing Forms

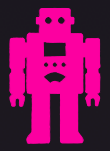
2 Ways of Handling Forms in Angular

Template-driven Forms

Reactive Forms

Managing Inputs, Values & Validation

Two Main Ways Of Handling Forms



Template-driven

Setting up forms via component templates

- Easy to get started
- Implementing more complex logic & forms can be tricky

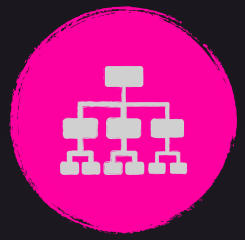
Reactive

Setting up forms via TS code

- Setup requires more verbose code
- Handling more complex forms can be easier

Template-driven Forms

Reactive Forms



Routing

What & Why?

Route Configuration

Nested Routes

Resolving Data & Controlling Access

Angular Apps are Single Page Applications

Yet they often consist of multiple pages

Angular Has Built-in Routing Support

Client-side Routing

Angular watches & manipulates the URL and renders different components for different URLs

Important: It's all happening in the browser! There's no server-side routing involved.

Example

/users → UsersComponent

/shop → ShopComponent

Lazy Loading

What & Why?

Deferrable Views

Lazily Loaded Routes

What Is Lazy Loading?

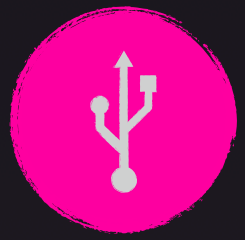
Only load & run code when it's needed!

Advantage

Smaller initial bundle size,
application is up & running quicker

Route-based Lazy Loading

Deferrable Views



Deployment

Build Options: SPA, SSR, SSG

What, Why & When?

Deployment Examples

Deployment?

Option 1

Single Page Applications

Build a client-side only web application

All code executes in the browser

No dynamic web server needed — a static host suffices

Potential disadvantages: Initially missing content, bad SEO

Option 2

Server Side Rendered App

Angular app routes are rendered on-demand on a dynamic web server

Browser receives finished, rendered page

Web app is hydrated (“activated”) and becomes a SPA after initial rendering

Dynamic web server is required

Potential disadvantages: Long-taking tasks may cause empty pages, complexity

Option 3

Static Site Generation

Angular app routes are pre-rendered at build time

Browser receives finished, rendered pages

Web app is hydrated (“activated”) and becomes a SPA

Dynamic web server is required — static host suffices if ALL pages are pre-rendered

Potential disadvantages: No dynamic server-side data fetching