Multiplatform szoftverfejlesztés

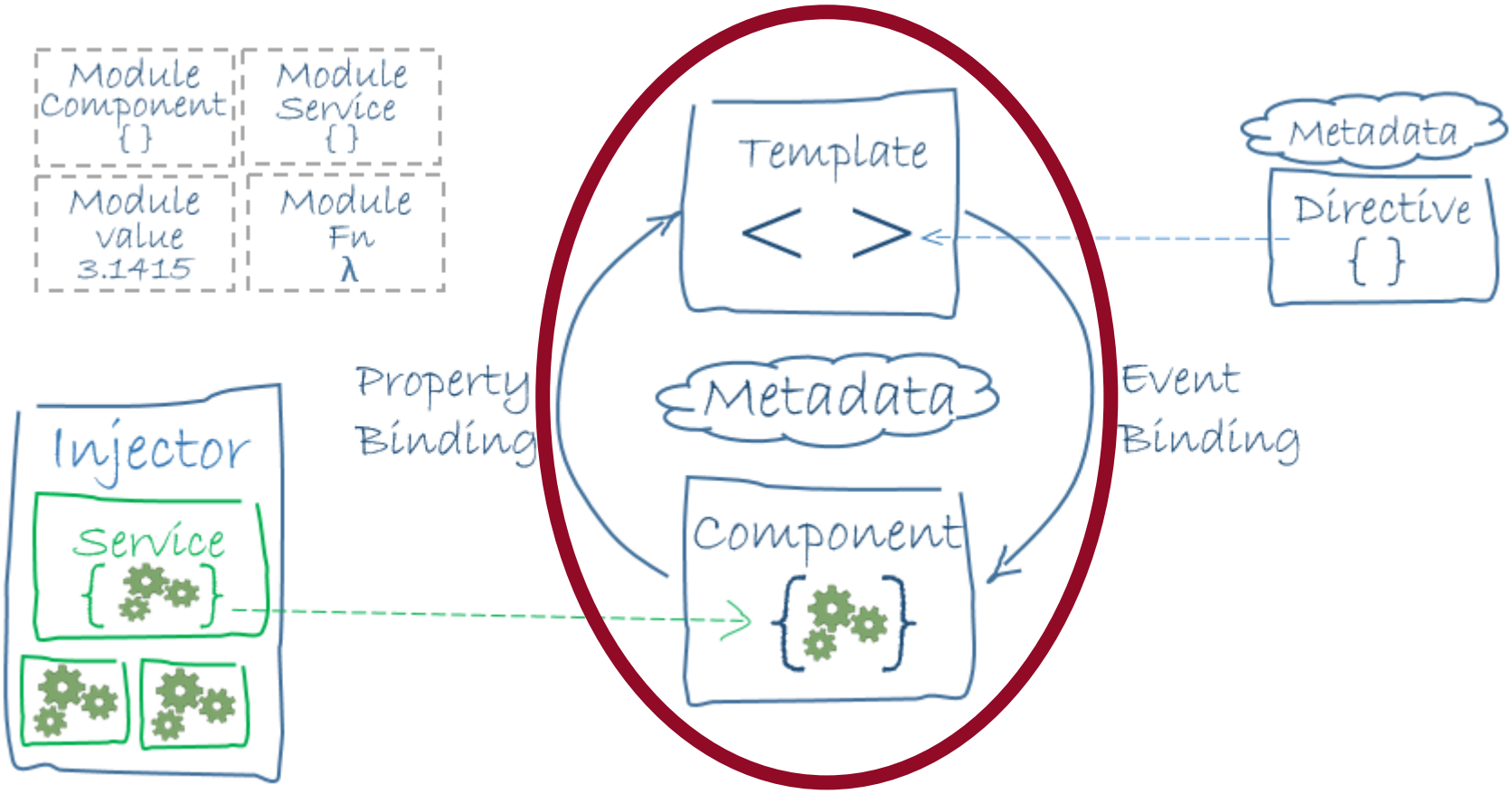
Angular

Emlékeztető

- Mit is akarunk megoldani?
  - Automatizált HTML frissítés
    - Listákkal, feltételekkel
  - Input kezelés
  - Adatkötés (kétirányú, ha lehetséges)
  - Kompozíció
    - Felület komponensenkénti kezelése
  - Tooling
    - Debug, test

Angular

Angular architektúra



Komponens

- X.component.ts
  - A komponens kódja (TypeScript)
- X.component.html
  - A megjelenítésért felelős HTML sablon
- X.component.css
  - A komponenshez (HTML sablonhoz) tartozó CSS
- X.component.spec.ts
  - Teszt kód, opcionális
  - A komponenssel együtt írhatjuk a unit tesztet

Hello World (component)

```
<p>Hello, {{name}}!</p>
```

```
import { Component } from '@angular/core';
@Component( {
  selector: 'welcome',
  templateUrl: './welcome.component.html',
  styleUrls: [ './welcome.component.css' ]
} )
export class WelcomeComponent
{
  name = "Leo";
}
```

@Component decorator

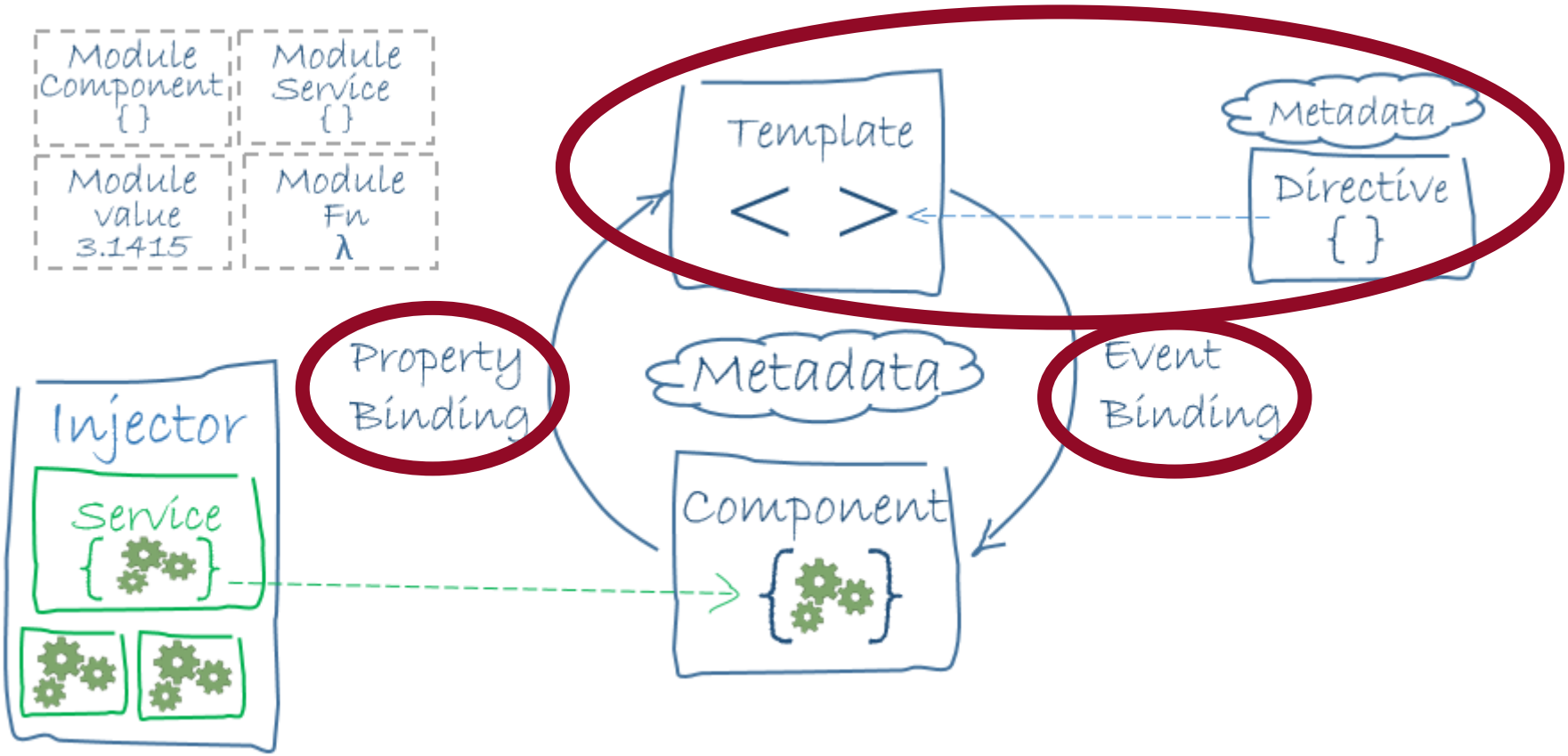
- Importálni kell
- Mit tud egy decorator?
  - Kb. mint attribútum C#-ban, itt mindig függvény
  - Írhatunk saját decoratort is
- Plusz információt ad – metaadat
  - selector: milyen HTML tag-re tegye magát
  - template, vagy templateUrl
    - Közvetlenül a HTML sablon, vagy az elérhetősége
  - styles, vagy styleUrls
    - Maga a CSS, vagy elérhetősége

export class

- A komponens kódja osztály formában
- Kívülről is látható (export kulcsszó miatt)
- Normális osztály, bármit beleírhatunk
  - Bizonyos függvények speciális funkcionalitást valósítanak meg (ld. életciklus)
- Adatkötés forrása
  - Az osztály minden tulajdonságához tudunk kötni

Sablon szintaktika

Angular architektúra



HTML sablon

- Mint Vue-nál: normál HTML + attribútumok
  - Emlékeztető: React-nél kód van
  - Szinte minden HTML tag megengedett
    - script, html, body, base nem
- Adatkötés
  - Egyirányú (komponens ⇨ sablon) : {{}}, [] = "..."
  - Egyirányú (sablon ⇨ komponens): ()
  - Kétirányú: [()]
- Strukturális direktívák
  - Az adott HTML elem megjelenjen-e vagy sem
  - *ngIf, *nfGor, *ngSwitch stb.
  - Sajátot is készíthetünk

0, vagy 1 elem: *ngIf

- Kifejezést fogad el

```
<p *ngIf="numbers.length>3">Sok szám</p>
```

- Teljesen kiveszi az elemet
  - Az elem tartalmát nem értékeli ki
  - Teljes részfát törli, komponenseket is
- Ha csak rejteni akarjuk
  - display-t kell none-ra állítani kötéssel
  - Nincs külön megoldás, mint pl. Vue-ban
- Van ngSwitch, mintha sok ngIf lenne
  - Kényelmesebb

0, vagy több elem: *ngFor

- "let x of c" szintaktika

```
<li *ngFor="let n of numbers">{{n}}</li>
```

  - x az elem
  - c a gyűjtemény
- Megszerezhetjük az indexet is

```
<li *ngFor="let n of numbers; let i=index">{{i+1}}: {{n}}</li>
```

- Azonosító megadása: trackBy (React: key)
  - Függvény, ami visszaadja a kulcsot

```
<li *ngFor="let n of numbers; trackBy: numbersKey">{{n}}</li>
```

Adatkötés {{}}

- Betehetjük önállóan, vagy szöveg mellé

```
<p>Hello, {{name}}!</p>
```

- Egyirányú adatkötés (komponens ⇨ sablon)
- Tetszőleges TS kifejezés lehet benne
  - Kivétel olyanok, amiknek mellékhatása van
    - Például értékadás
  - Cél, hogy ne legyen bonyolult
    - Deklaratív megoldásokat nehéz tesztelni
- Attribútumban is működik

Adatkötés [attrib]=“expr”

- Azonos hatása van, egyirányú
- De ez a DOM elem tulajdonságához köt

```
<p [id]="numbers[0]">ID</p>
```

  - Esetünkben azonos az attribútummal (id)
  - De a legtöbb esetben más (className vs. class)
  - Vagy nincs is olyan tulajdonság (aria-label)
- Ha az elem egy komponens (és nem DOM elem), akkor annak a tulajdonságához köt
- Alternatív szintaktika
  - [prop] helyett bind-prop

Adatkötés [class]

- A class attribútum egy lista, a kötése egyedi
- Egyesével ki-be kapcsolni osztályt

```
<p [class.centered]="isCentered">Közép</p>
```

- Egyszerre többet kezelni
  - ngClass direktíva

```
<p [ngClass]="{'centered': numbers.length==2}">Közép</p>
<p [ngClass]="n==3? 'centered' : 'normal'">Közép</p>
```

  - Van még pár megadási lehetőség

Adatkötés [style]

- A style attribútum egy objektum
- Egyesével

```
<p [style.display]="n==2 ? 'block' : 'none'">Hello</p>
```

  - Mértékegység megadással is lehet

```
<p [style.font-size.em]="numbers.length">Big</p>
```

- Egyszerre többet kezelni
  - ngStyle direktíva, hasonló az ngClass-hoz

Adatkötés (esemény)

- (esemény) szintaktika
  - Kódot futtat
  - Nem elég megadni a fv. nevét, meg is kell hívni

```
<button (click)="onSave()">Mentés</button>
```

- Ha szükségünk van az esemény objektumra
  - $event változóban van
- Alternatív szintaktika: on-esemény
- Egyirányú adatkötés (sablon ⇨ komponens)

Kétirányú adatkötés: [()]

- [()] szintaktika adja a kétirányú adatkötést

```
<comp [(prop)]="myprop">
<comp [prop]="myprop" (propChange)="myprop=$event">
```

  - []: egyirányú adatkötés a tulajdonságra
    - comp.prop kötve this.myprop-hoz
  - (): eseményre feliratkozás
    - comp.propChange-re
- Ez meg is oldaná a kétirányú adatkötést, ha a DOM-ban egységesen ez lenne az elnevezés
  - De nem ez, és nem egységes
- Komponenseinkben ezt érdemes használni

Kétirányú adatkötés: ngModel

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

- ngModel direktíva szükséges
  - Ismeri az összes beépített HTML elemet
    - Mindegyik eseménykezelőjére fel tud iratkozni
    - És tudja állítani az értékét
- Kell hozzá importálni a FormsModule-t
  - Form elemekre alkalmazható
- Kiterjeszthető, ha külső könyvtárhoz kéne igazítani

HTML elem azonosítása

- Megkereshetjük a fában, pl. querySelector
  - Ha átírjuk a sablont, akkor a kód hibás lesz
  - Ezt el kell kerülni
    - Szerepkörök szétválasztása fontos paradigma (separation of concerns)
- Jelöljük meg és hivatkozzunk rá

```
<p #subtitle>Felirat</p>
<p>{{subtitle.textContent}}</p>
```

- Kódban is elérhetjük: @ViewChild

Formátum konvertálók – Pipes

- Ha az adat formátuma nem megfelelő
- Használhatunk beépített konvertálót
  - date: dátum

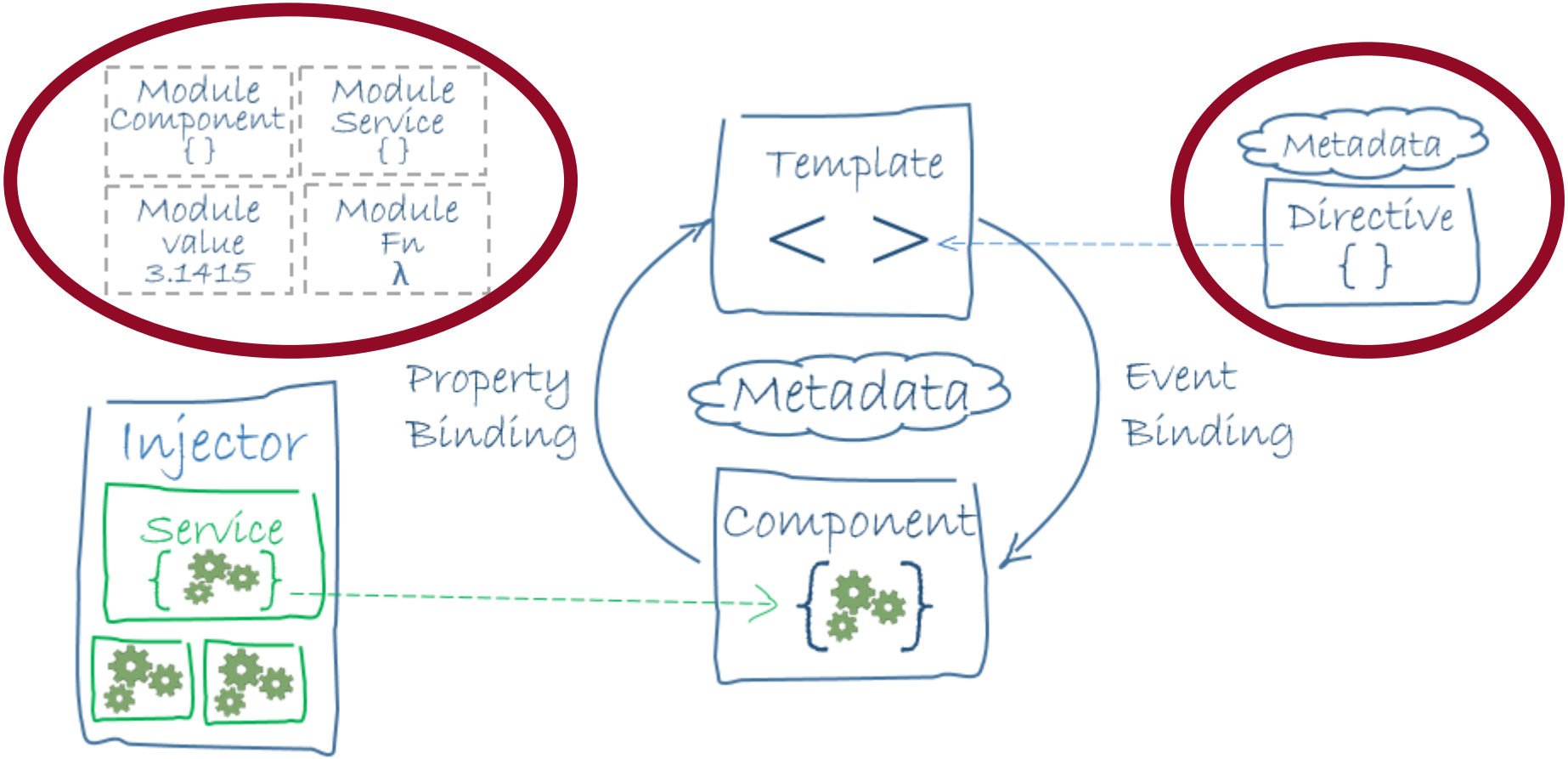
```
<p>Ma: {{ today | date }}</p>
```

  - uppercase, lowercase, titlecase
  - number: szám formázás
  - async: amint megjön az adat, frissít és kijelzi
  - json: JSON.stringify()
- Egymás után láncolhatjuk őket
- Írhatunk sajátot

Felépítés

Életciklus, modulok, direktívák

Angular architektúra



Életciklus kezelése

- Angular esetén ritkán van szükségünk rá
  - A legtöbb feladatra van kész megoldás
- ngOnInit: konstruktor helyett inicializáló logika ide
- ngOnDestroy: végső takarításra használjuk
- ngOnChanges: adatkötött tulajdonság változásra
- ngDoCheck: ha valamiért nem lehet egy változást észlelni, akkor itt beavatkozhatunk
- stb.

Modul

- A modul komponensek halmaza
  - Amik összetartoznak
  - Kifelé egységesen tudnak fellépni
  - Egy adott feladatot oldanak meg együtt
  - Komponensen kívüli kód is tartozhat hozzá
- Csak egyben lehet őket importálni/használni
- Nem JS modul, annál tipikusan kisebb
- Egy alkalmazás több modulból áll általában
- A keretrendszer is ilyen modulokból áll
  - És külső könyvtárak is

@NgModul

- Modul dekorátor

```
@NgModule({
  declarations: [AppComponent, WelcomeComponent],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

  - declarations: modul tartalma (pl. komponensek)
  - imports: függőségek
  - providers: szolgáltatások (ld. később)
  - bootstrap: belépési pont

Modul indítása

- Main.ts-ben indul a fő modulunk

```
platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

- A fő modul pedig indítja, amit a bootstrap-ben megadtunk

Klterjeszthetőség

- Angular funkcionalitása kiterjeszthető
  - Alapvető funkciókat is kiterjesztés old meg (ngIf)
  - A keretrendszer nagy része kiterjesztés
- Direktívák
  - A HTML generálásba és komponensek működésébe beavatkozás
- Szolgáltatások
  - Funkciókat valósítanak meg komponensen kívül
  - Például adatlekérés a szerverről
- Pipe, serviceWorker, webWorker, …

Direktívák

- Beavatkoznak az elemek működésébe
  - HTML elemeken és komponenseken is működnek
  - Akár megváltoztatják a HTML fa felépítését is
- Attribútum direktíva az adott elemen működik
  - Például ngClass, ami osztályokat tesz rá/vesz le
- Strukturális direktíva a fát változtatja meg
  - Például *ngFor, ami HTML fát generál
  - * és mikroszintaktika, hogy keveset kelljen írni
    - E nélkül <template>-et kéne definiáljunk
- Sajátot is írhatunk

Direktívák

- A direktíva egy osztály
  - Nem komponens, mert nincs felülete
  - Ezen kívül szinte ugyanazt tudja, mint egy komponens
- @Input és @Output tulajdonságokkal vesz részt adatkötésben
  - De ez nem kötelező
  - @Input és @Output = dekorátorok

Attribútum direktíva

- 2 fájlból áll: az osztály és a teszt
- Egy példa, ami bold-ra állít

```
import { Directive, ElementRef } from '@angular/core';
@Directive( { selector: '[appBold]' } )
export class BoldDirective
{
  constructor( el: ElementRef )
  {
    el.nativeElement.style.fontWeight = 'bold';
  }
}
```

Attribútum direktíva – HostListener

- Eseményekre tudunk figyelni

```
@HostListener( 'mouseenter' ) onMouseEnter()
{
  this.el.nativeElement.style.fontWeight = 'bold';
}
@HostListener( 'mouseleave' ) onMouseLeave()
{
  this.el.nativeElement.style.fontWeight = 'normal';
}
```

  - A HostListener megoldja a feliratkozást és a leiratkozást

Attribútum direktíva – adatkötés

- @Input lehetővé teszi az adatkötést
- Ha azonos a neve a direktívával, akkor default
  - Eltérő név esetén a komponensben meg kell adni a nevet

```
@Input( 'appBold' ) bold: boolean;
@HostListener( 'mouseenter' ) onMouseEnter()
{
  this.el.nativeElement.style.fontWeight = this.bold ? 600 : 300;
}
```

- Használata

```
<p [appBold]="numbers.length>3">Közép</p>
```

Attribútum direktíva – adatkötés

- Van @Output is a kétirányú adatkötéshez

```
@Output( 'appBoldChange' ) boldChange = new EventEmitter<void>();
```

- emit függvény tüzeli az eseményt

```
@HostListener( 'mouseenter' ) onMouseEnter()
{
  if ( this.bold )
  {
    this.el.nativeElement.style.fontWeight = 'bold';
    this.bold = false;
    this.boldChange.emit();
  }
}
```

Attribútum direktíva – adatkötés

- Felhasználása

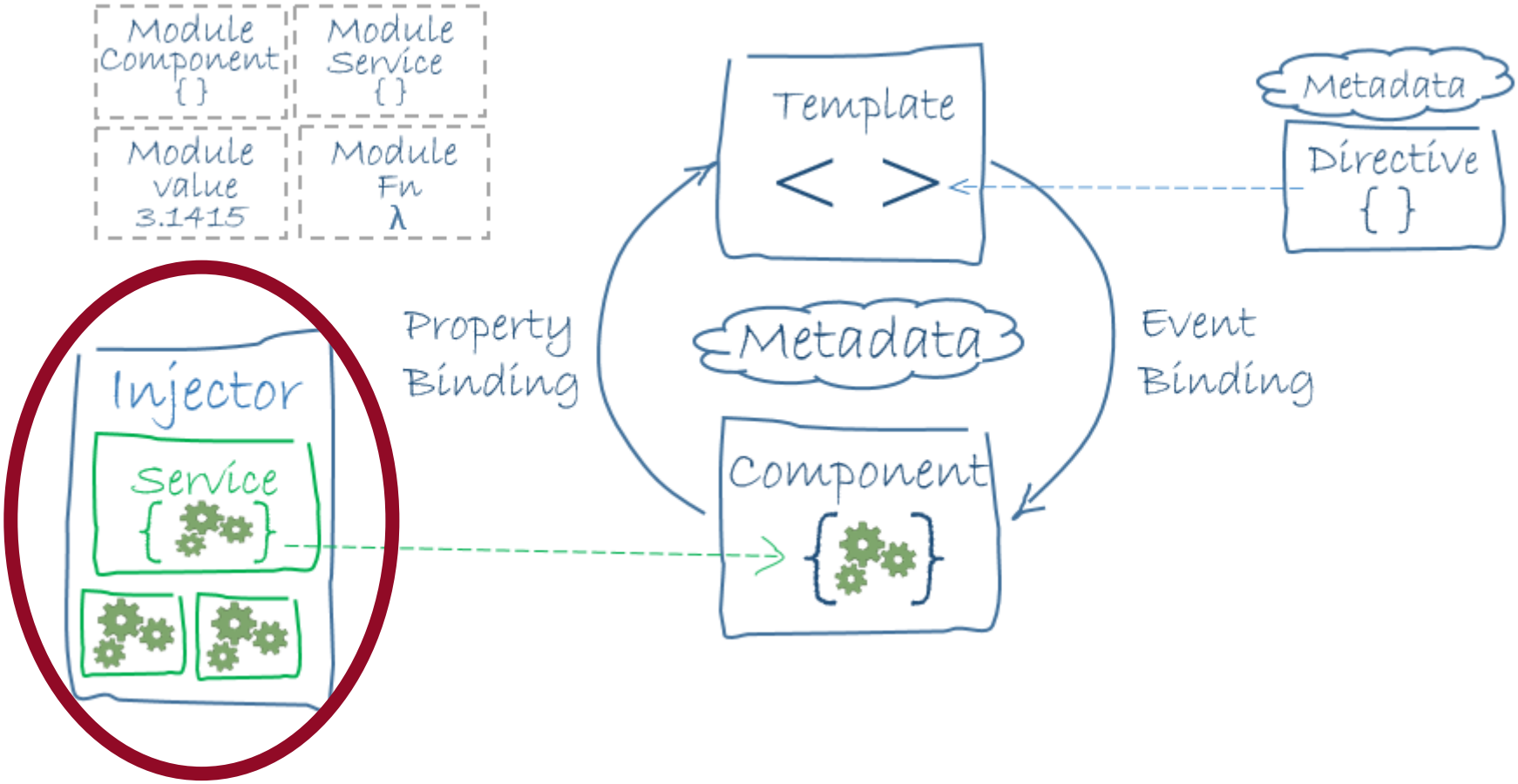
```
<p [appBold]="isBold" (appBoldChange)="isBold=$event">Bold</p>
```

- A kétirányú adatkötés szintaktikával

```
<p [(appBold)]="isBold">Bold</p>
```

  - Ez csak azért megy, mert van
    - appBold
    - appBoldChange
  - Más elnevezésnél nem működik
    - Külön ki kell írni az eseménykezelőt
- Törekedjünk erre az elnevezésre

Angular architektúra



Szolgáltatások (service)

- Tipikusan nem globálisan hozzuk létre a komponensen kívüli objektumokat
  - Szolgáltatások formájában
  - De nem kódoljuk bele egyik komponensbe sem
  - Felsoroljuk, hogy mik vannak
  - Mi hivatkozik mire
  - És melyik komponensnek mi kell
- A keretrendszer automatikusan adja
- Függőség injektálás (Dependency Injection)

Függőség injektálás (DI)

- Általánosan: Az objektum által használt függőséget nem az objektum kezeli, hanem csak kapja. Az injektor felelős a függőségek kezeléséért (életciklus, kiosztás).
- Esetünkben a komponens nem felelős a szolgáltatásokért, csak kapja őket használatra
- A komponensek jönnek-mennek, míg a szolgáltatások életciklusa teljesen eltérő
- A komponensnek nem kell tudnia, hogyan hozzon létre szolgáltatásokat

Függőség injektálás (DI)

- Szerepkörök szétválasztása miatt fontos
  - Tesztelhetőség nő
  - Dekompozíció erősödik
  - Újrafelhasználhatóság javul
  - Komponens kódja kisebb, olvashatóbb marad
- Nem csak Angular-ban van
  - Így ismerhetjük fel a DI-t: használunk egy szolgáltatást (külső funkcionalitást), ami nem globális, és nem is mi hoztuk létre

Példa szolgáltatás

- Attól szolgáltatás, hogy injektálható

```
import { Injectable } from '@angular/core';
@Injectable( { providedIn: 'root' } )
export class RandomDataService
{
  constructor() { }
  get() { return [ 1, 2, 5, 6, 7 ]; }
}
```

- Amúgy csak egy sima osztály
- providedIn: mely komponensek számára elérhető (root mindenkinek)

Szolgáltatás felhasználása

- Simán átvesszük a konstruktorban, és használjuk

```
constructor( rng: RandomDataService )
{
  this.numbers = rng.get();
}
```

- Az injektor gondoskodik arról, hogy
  - Létre legyen hozva
  - Átadja, amikor létrejön a komponens
  - Megszüntesse valamikor

Szolgáltatások

- Szolgáltatások
  - HttpClient: XHR+json kommunikáció
  - Location: address bar
  - FormBuilder: űrlap kezelő
  - Router: navigáció
  - …
- Tree shaking működik szolgáltatásokra (DI-re)
  - Csak azok kerülnek bele a végső kódba, amire hivatkozunk

Kérdések?