Webalkalmazás architektúrák

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék

Webes programozás

- Elosztott programozás

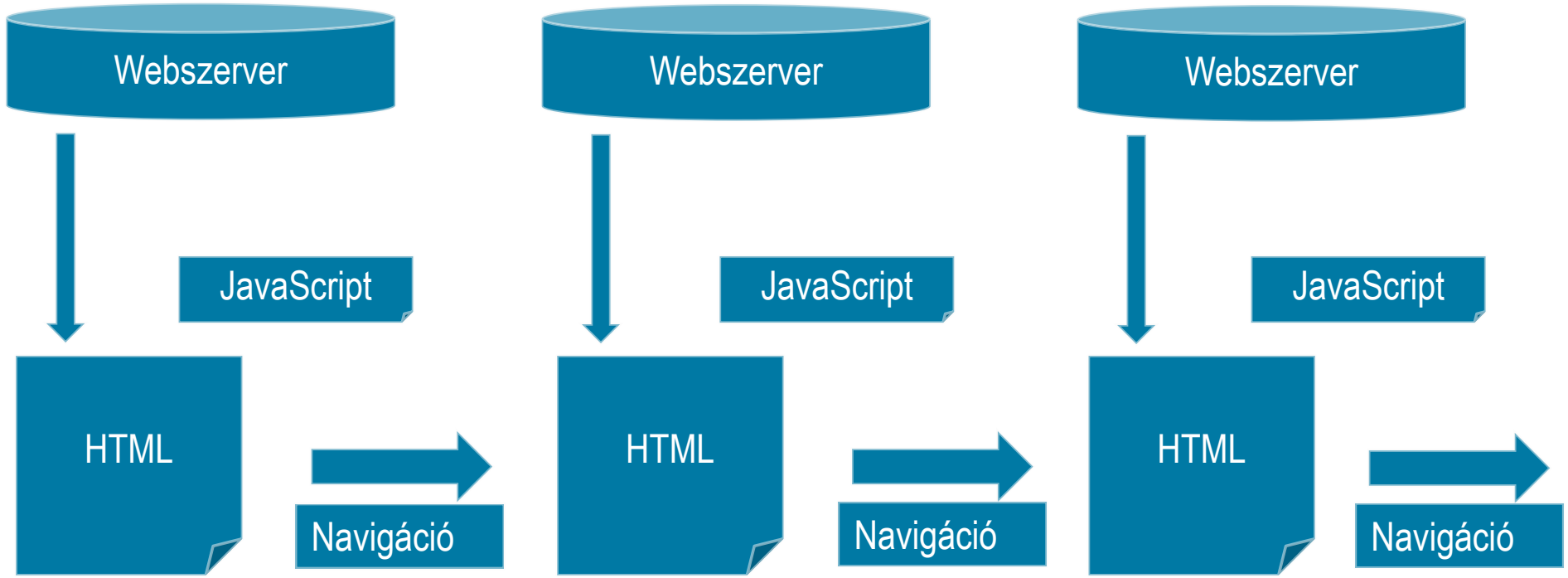
Kliens oldal

Szerver oldal

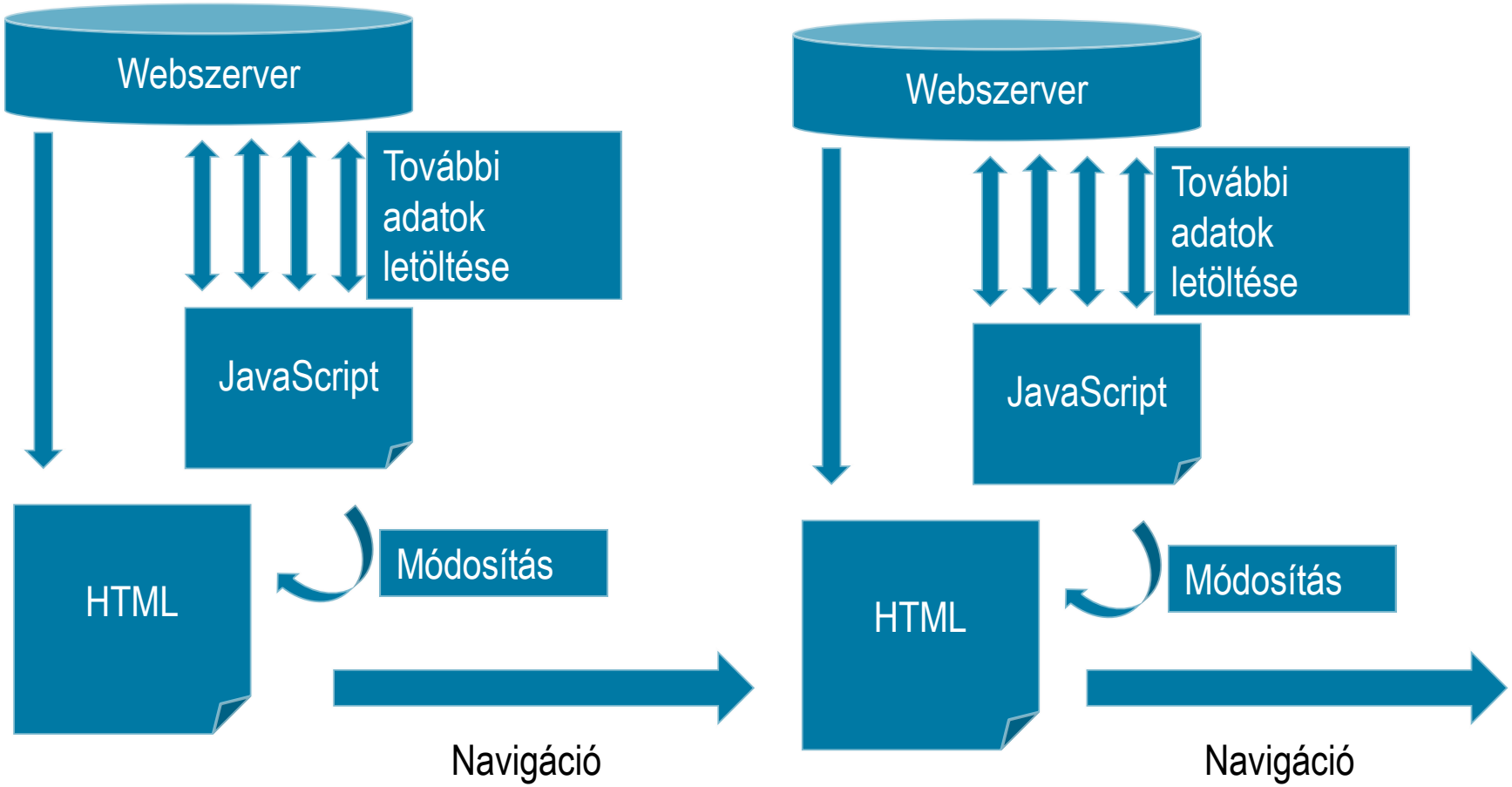
Webes programozás

- Separation of concerns

Régen

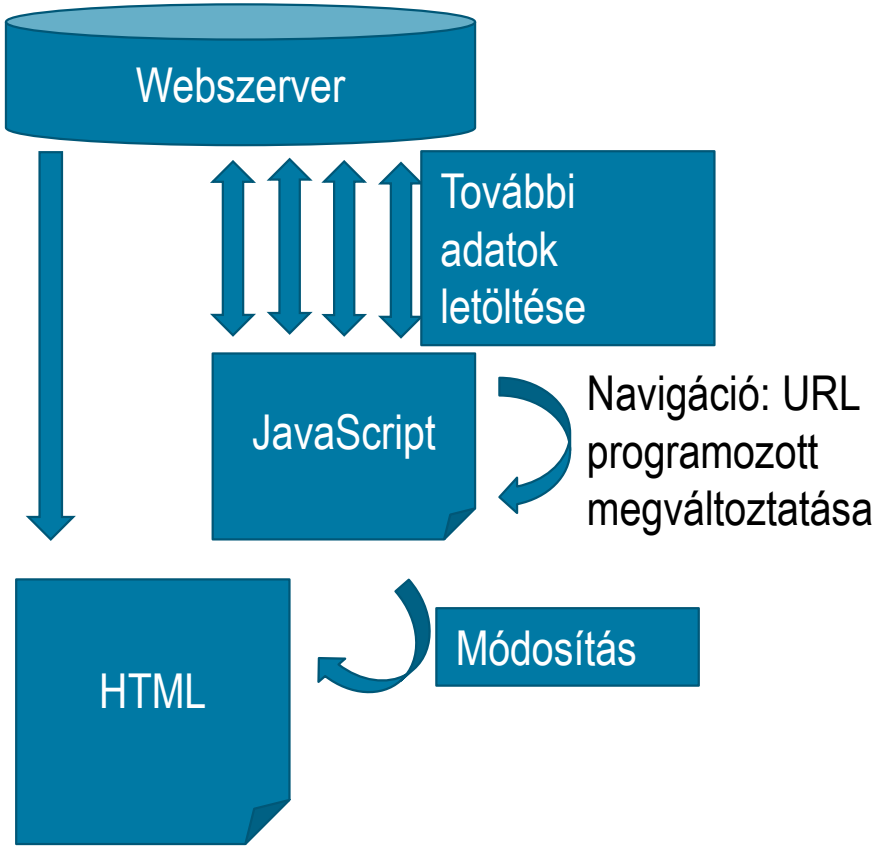

Webszerver
JavaScript
HTML
Navigáció
Webszerver
JavaScript
HTML
Navigáció
Webszerver
JavaScript
HTML
Navigáció

Modern web alkalmazás



Single Page Application (SPA)

- Navigálás: új erőforrások megcímzése – megjegyezhetjük az állapotokat



A jelen

- SPA, SPA komponensek
- Fejlettebb JavaScript
  - > Összetett keretrendszerek
  - > Fejlődő szabvány (EcmaScript 6, TypeScript, CoffeeScript)
  - > Fejlett, gyors böngészők

- Bonyolultabb JS kód
  - > Tervezés / architektúra
  - > Tervezési / architektúrális minták

Feladatok

- HTML nézet előállítása
- Aszinkron kommunikáció a szerveroldallal
- Eseménykezelés
- Modulkezelés
- Állapotkezelés
- Validáció
- Tesztelés

Miről lesz szó?

- HTML templating, JSX
- MV*
- Egy vs. két irányú adatkötés
- Hatékony DOM kezelés: React, Angular
- Event-driven architecture / Aszinkronitás kezelése

Webes architektúra fogalma

Framework

Library

Utility

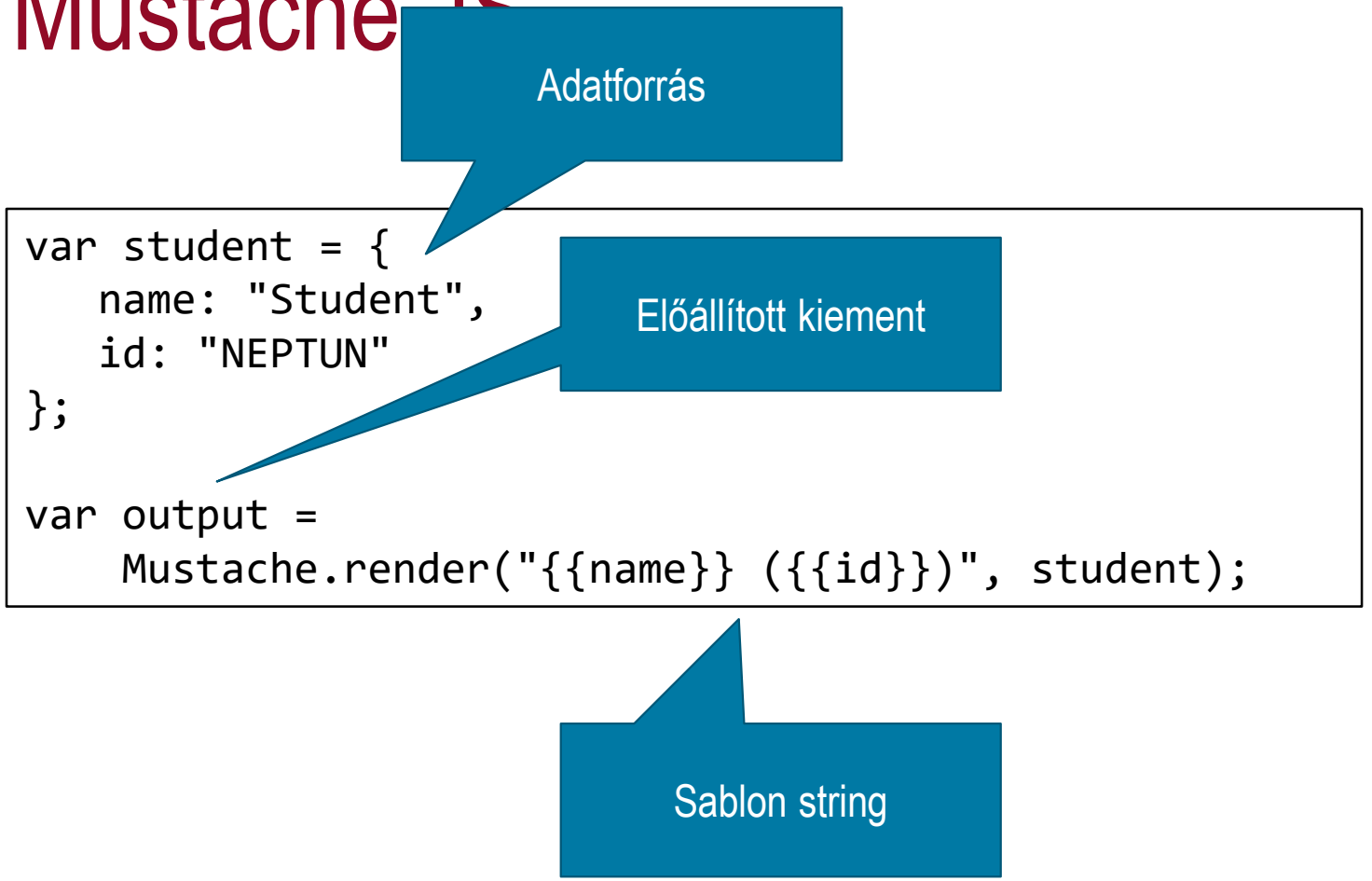
Architektúra

All in one tool

HTML templating

HTML templating - Mustache.JS

- Szöveges sablonok
- Használata:
  - > A könyvtár betöltése
  - > Adatforrás objektum
  - > Sablon string
    - – Speciális tag-ek {{…}}
  - > Renderelés



Mustache JS

- Tag típusok
  - > Változó: {{*változónév*}}
  - > Szekció + iteráció: {{#*változnév*}} … {{/*változónév*}}
  - > Komment: {{! *comment*}}
  - > Függvények
  - > Parciális nézetek

```
<script id="template" type="text/x-tmpl-mustache" >
   <ul>
     {{#students}}
       <li>{{name}} ({{id}}</li>
     {{/students}}
   </ul>
</script>
```

Sablon

```
<div id=" placeaHolder "></div>
```

Ide illesztjük be a kimenetet

Adatforrás

```
var database = {
   students: [
     { name: "Student 1", id: "NEPTUN1" },
     { name: "Student 2", id: "NEPTUN2" },
     { name: "Student 3", id: "NEPTUN3" },
   ],
   classes : { }
};

var template = $("#template").html();
var output = Mustache.render(template, database);
$("#placeaHolder").html(output);
```

Sablon string beolvasása

HTML DOM frissítése

Kiement előállítása

Mustache JS – DEMO

MV*

MVC, MVP, PAC, MVVM

MVVM – Model-View-ViewModel

- Architektúrális minta
- Más mintákkal szoros kapcsolat (Model-View-Controller, Model-View-Presenter, Document-View, Presentation Model)
- Eredetileg WPF-hez definiálták
- Célja: eseményvezérelt programozási környezetben a nézet és a logika szétválasztása

MVVM

UI
Deklaratív adatkötés (kétirányú)
Deklaratív command kötés

Megjelenítéshez szükséges
logika +
Változásértesítés +
Származtatott adatok

Adatmodell +
Üzleti logika

Person

First Name: X
Last Name: Y
Full name: X, Y
Address: My address
Update address

PersonVM:
- FirstName
- LastName
- Address
- FullName // FirstName, LastName

- UpdateAddress**Command**
  // Model.UpdateAddress

Person:
- FirstName
- LastName
- ID
- Address

- UpdateAddress(…)

MVVM

- Speciális programozási környezet
  - > Deklaratív adatkötés – *kétirányú*
  - > Változásértesítés
  - > Deklaratív command kötés – *kétirányú*
- View és ViewModel elszapárálása → Tesztelhetőség

Knockout.js

- JavaScript MVVM keretrendszer
- Deklaratív adatkötés
  - Kétirányú: automatikus UI és model frissítés
- Származtatott értékek esetében a függőségek automatikus felderítése
- Sablonok használata

Knockout.js

A ViewModel egy tagváltozójához kötjük a belül található szöveget

```
<!-- VIEW -->
<p>First name: <strong data-bind="text: firstName"></strong></p>
<p>Last name: <strong data-bind="text: lastName"></strong></p>
```

```
// ViewModel + „model”
function AppViewModel() {
    this.firstName = "Bert";
    this.lastName = "Bertington";
}
```

ViewModel létrehozása

„Modell”: itt statikus adatok

```
// Activates knockout.js
ko.applyBindings(new AppViewModel());
```

MVVM rendszer indítása

First name: **Bert**
Last name: **Bertington**

Knockout.js – minden kód együtt

```
<html>
<head>
    <script type='text/javascript'
                src='http://ajax.aspnetcdn.com/ajax/knockout/knockout-3.0.0.js'></script>
</head>
<body>
    <h1>Knockout test</h1>
    <p>First name: <strong data-bind="text: firstName"></strong></p>
    <p>Last name: <strong data-bind="text: lastName"></strong></p>

    <script type='text/javascript'>
        function AppViewModel() {
            this.firstName = "Bert";
            this.lastName = "Bertington";
        }
        ko.applyBindings(new AppViewModel());
    </script>
</body>
</html>
```

Knockout.js

```
<p>First name: <strong data-bind="text: firstName"></strong></p>
<p>Last name: <strong data-bind="text: lastName"></strong></p>
```

```
<p>First name: <input data-bind="value: firstName" /></p>
<p>Last name: <input data-bind="value: lastName" /></p>
```

Kétirányú adatkötés

```
function AppViewModel() {
    this.firstName = ko.observable("Bert");
    this.lastName = ko.observable("Bertington");
}
ko.applyBindings(new AppViewModel());
```

First name: **Bert**

Last name: **Bertington**

First name: Bert

Last name: Bertington

Knockout.js

```
function AppViewModel() {
    this.firstName = ko.observable("Bert");
    this.lastName = ko.observable("Bertington");

    this.fullName = ko.computed(function() {
        return this.firstName() + " " + this.lastName();
    }, this);

    this.capitalizeLastName = function() {
        var currentVal = this.lastName();
        this.lastName(currentVal.toUpperCase());
    };
}

ko.applyBindings(new AppViewModel());
```

Változásértesítés az adat megváltozásáról

Származtatott adat (automatikus függőség követés)

Command: ahol elérhető a scope, vagyis az aktuális ViewModel

this.lastName = ko.observable(…) erdménye: függvényként érhető el a tárolt adat

Knockout.js

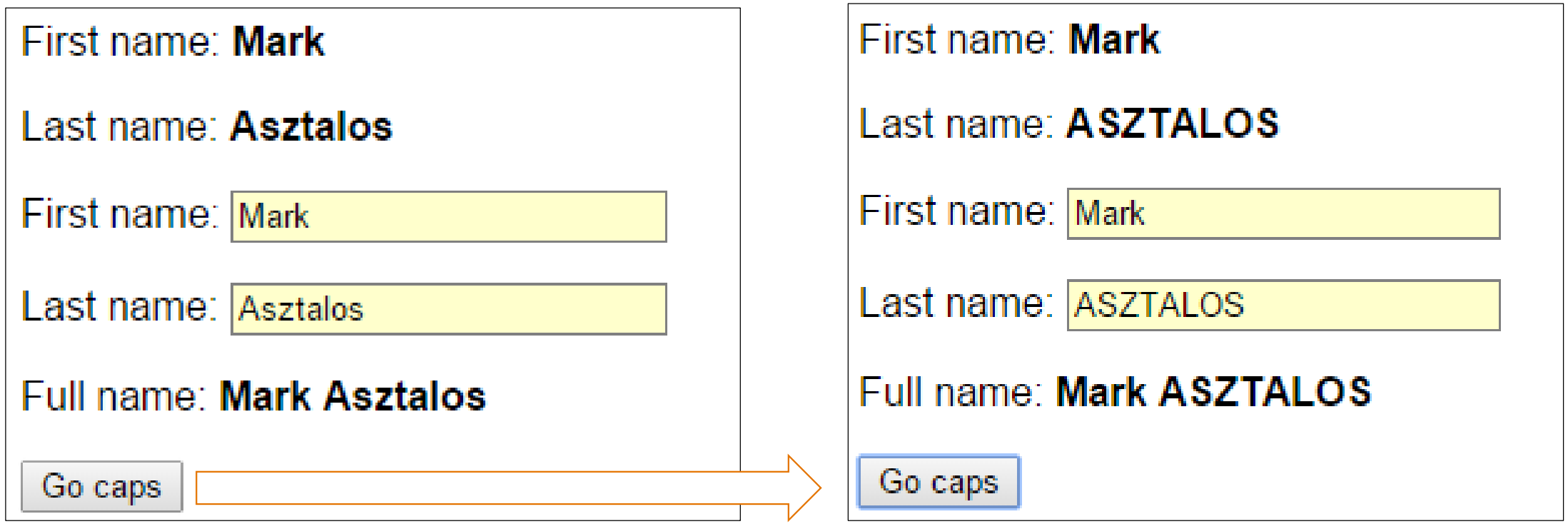
```
<p>First name: <strong data-bind="text: firstName"></strong></p>
<p>Last name: <strong data-bind="text: lastName"></strong></p>

<p>First name: <input data-bind="value: firstName" /></p>
<p>Last name: <input data-bind="value: lastName" /></p>

<p>Full name: <strong data-bind="text: fullName"></strong></p>

<button data-bind="click: capitalizeLastName">Go caps</button>
```

Knockout.js


First name: Mark
Last name: Asztalos
First name: Mark
Last name: Asztalos
Full name: Mark Asztalos
Go caps
First name: Mark
Last name: ASZTALOS
First name: Mark
Last name: ASZTALOS
Full name: Mark ASZTALOS
Go caps

Knockout.js – DEMO

MVVM – tesztelés

```
<script type="text/javascript" src="code.js"></script>
<body>
        <div id="qunit"></div>
        <div id="qunit-fixture"></div>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/qunit/1.18.0/qunit.js">
        </script>
        <script type="text/javascript" src="tests.js">
         </script>
</body>
```

```
var viewModel = new SoftArchViewModel();
QUnit.test( "hello test", function( assert ) {
  viewModel.selectedSemester(viewModel.semesters()[0]);
  viewModel.newStudentId("8");
  viewModel.newStudentName("XY");
  assert.equal(viewModel.canAddNewStudent(), true, "Student can be added");
  viewModel.addNewStudent();
  assert.equal(viewModel.canAddNewStudent(), false, "Student ID should be unique");
});
```

MVVM – Értékelés

- Előnyök
  - Tesztelhetőség
  - Karbantarthatóság
  - Kód újrafelhasználás
  - Fejlesztők és UI designerek együttműködésének segítése

- Hátrányok
  - Egyszerű UI → overkill
  - Bonyolult UI → memória

JSX

React JS

Mi a baj az eddigi HTML használattal?

„HTML should be the projection of the app state not the source of truth”

Mi a baj az eddigi HTML használattal?

- Separation of concerns


```
<!-- MUSTACHE -->
<script id="template" type="text/x-tmpl-mustache" >
   <ul>
     {{#students}}
       <li>{{name}} ({{id}}</li>
     {{/students}}
   </ul>
</script>
```

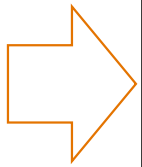
```
<!- HTML + JS -->
<div onclick="alert('x');">
```

```
<!-- KNOCKOUT -->
<p>First name: <strong data-bind="text: firstName"></strong></p>
<p>Last name: <strong data-bind="text: lastName"></strong></p>
```

JSX + React

- JavaScript szintaxisának kiterjesztése

```
const element = <h1>Helló Világ</h1>;
```



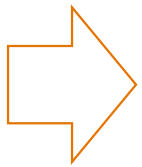
```
React.createElement(
      'h1',
      {},
      'Helló Világ');
```

Virtuális DOM elem a memóriában

JSX – paraméterezés

HTML template paraméter

```
const greet = "Helló";
const element = <h1 id="header">{greet}</h1>;
```



```
React.createElement(
      'h1',
      { id : 'header'},
      greet);
```

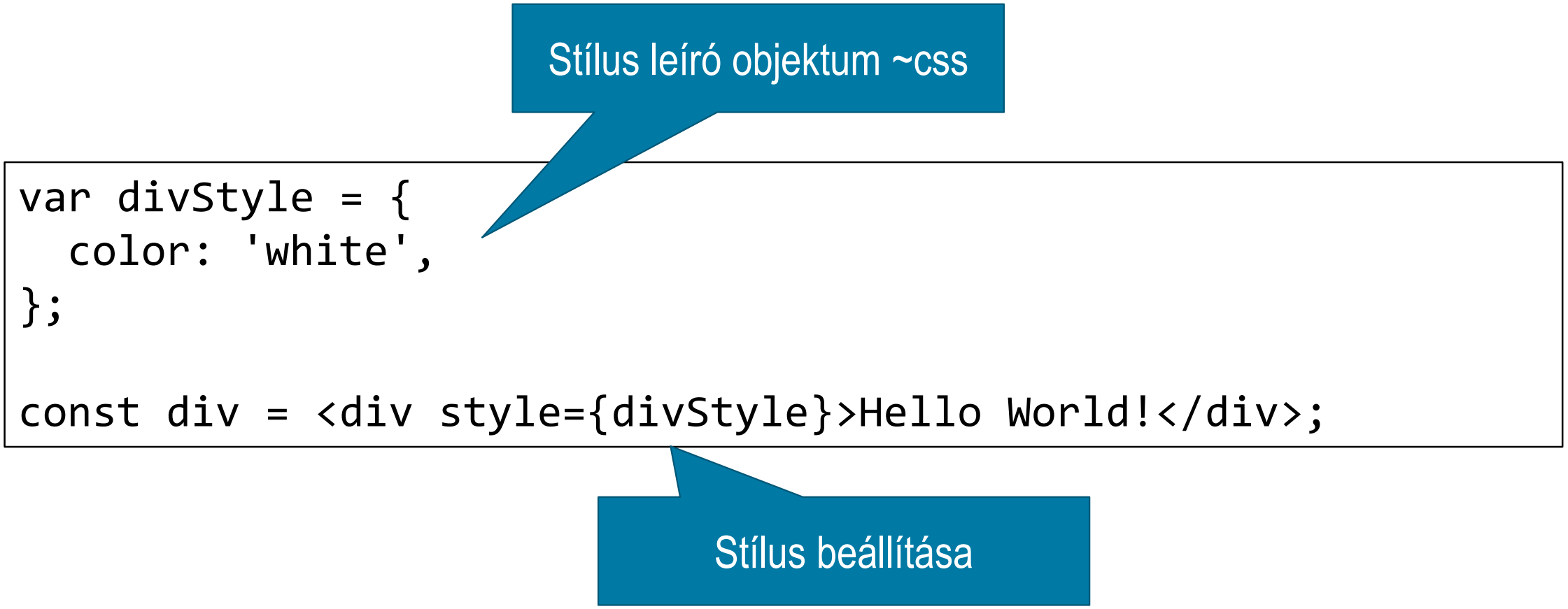
JSX – eseménykezelés

eseménykezelő

```
function handleClick(e) {
      e.preventDefault();
      console.log('The link was clicked.');
}
const a = <a href="#" onClick={handleClick}> Click me </a> );
```

Speciális nevű attribútumok az eseményfeliratkozáshoz

JSX – stílusok, attribútumok



JSX + React

- Virtuális DOM objektum felhasználása

Eredeti HTML tartalom

```
<div id="root"></div>
```

```
const element = <h1>Hello, world</h1>;
ReactDOM.render(
      element,
      document.getElementById('root'));
```

Virtuális DOM renderelése

React - komponens

```
class Hello extends React.Component {
    render() {
        return <h1>Hello {this.props.message}!</h1>;
    }
}
```

```
ReactDOM.render(
   <div>
      <Hello message="X" />
      <Hello message="Y" />
   </div>,
   document.getElementById("root")
);
```

Minden komponensben van *props*, ebből olvashatók ki a megkapott paraméterek

Újrafelhasználható, paraméterezhető komponensek

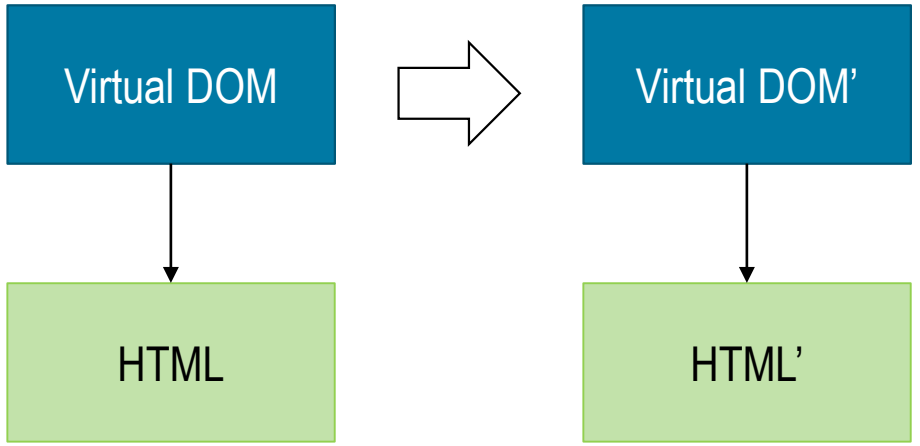
React

- JSX
  - Paraméterezhető
  - Függvényfeliratkozások
  - Stílusbeállítások
- Komponens alapú
  - Újraflehasználható
  - Input paraméterek (props)
  - Saját állapot (state)
  - Separation of components a separation of concerns elv helyett

Virtual DOM (React JS)

- **Virtual DOM** ← saját HTML DOM reprezentáció
  - > Dinamikus renderelés
  - > Csak a változások frissítése – teljesítmény
  - > Böngészőfüggetlen, szabványos eseménykezelés
  - > Az adatokat nem a HTML-ben tároljuk
  - > Isomoprhic rendering (renderelés szerveroldalon)
  - > Sokkal gyorsabb, mint közvetlenül az igazi DOM-ot módosítani

Virtual DOM (React JS)



Tudjuk a különbséget az előző DOMhoz képest

Csak a tényleges változtatást (törlés, létrehozás, módosítás) kell emittálni.

React DEMO

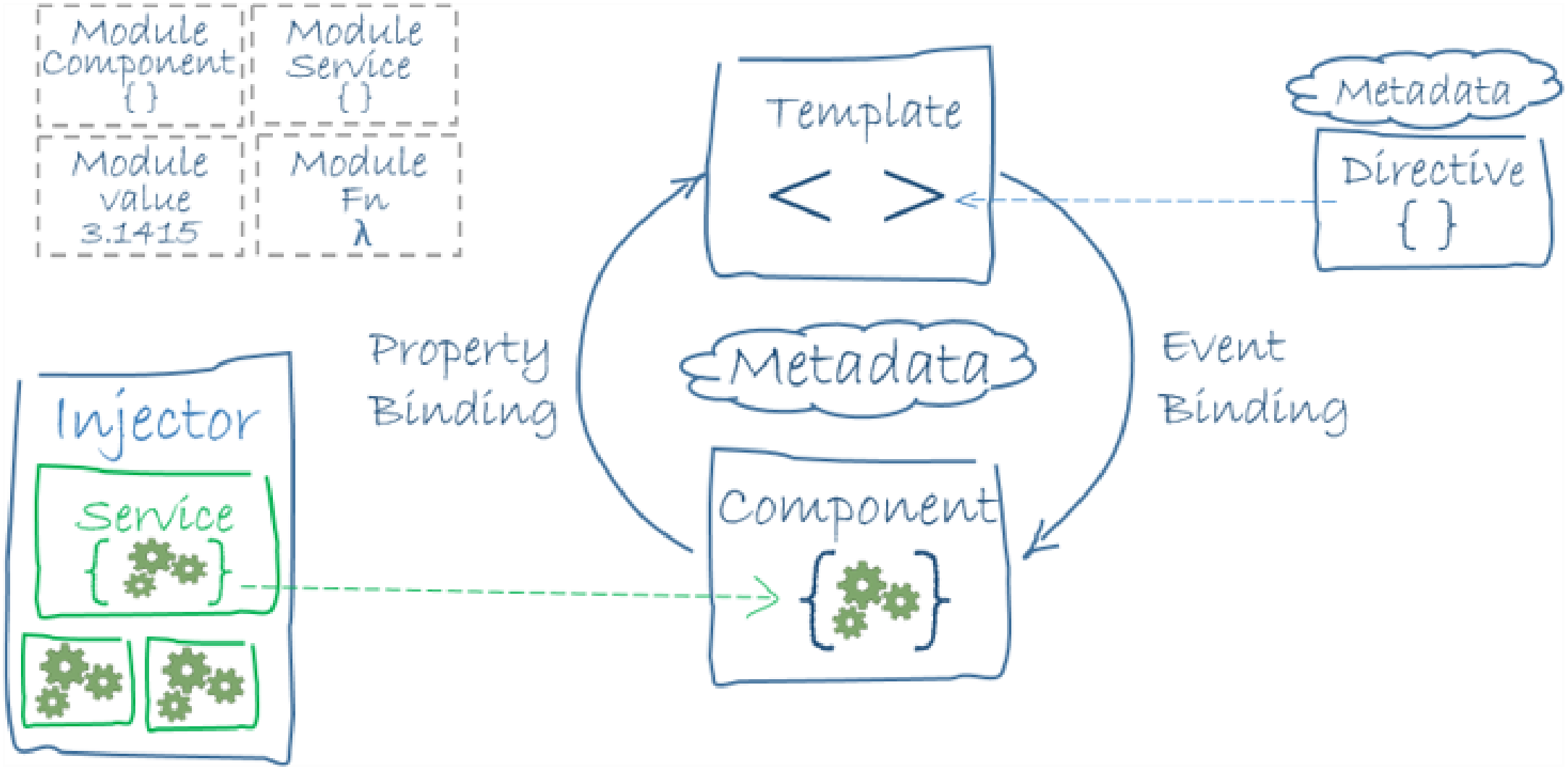
Angular

Angular

- Angular.io (≠ AngularJS)
- TypeScript
- Arhitektúra (osztályok, dekorátorokkal)
  - Modulok: egységbe fognak kódrészleteket
  - Komponensek:
    - nézetek (viewmodel szerű kód + HTML template)
    - Hook metódusok az életciklushoz kapcsolódóan
  - Szolgáltatások (service) (egyszerű objektumok, amik dependency injectionnel elérhetők)
- Dependency Injection

Angular DEMO

Angular


Module
Component
{}
Module
Service
{}
Module
value
3.1415
Module
Fn
λ
Template
< >
Metadata
Directive
{}
Property
Binding
Metadata
Event
Binding
Injector
Service
{}
Component
{}

Hogyan működik?

- Változásfigyelés (change detection
  - > Események figyelése (zone.js)
    - – Események: timeout, aszinkron hívások, DOM események
  - > Minden esemény után megvizsgáljuk a komponens régi és új állapotát, ha különbözik újrarendereljük
  - > Nem kellenek speciális property-k az osztályokba, működik a változásértesítés