WASM

La mort de JavaScript ?

WASM

La mort de JavaScript ?

Denis Voituron

MVP Reconnect

Principal Software Engineer

Podcaster

Principal Contributor

https://github.com/microsoft/fluentui-blazor

Brendan Eich

wikipedia.org

JavaScript

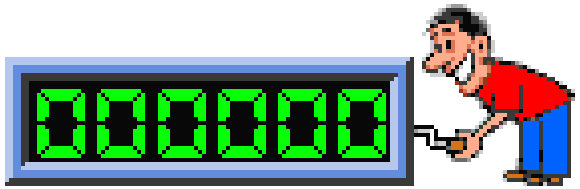
Langage de **script léger**, orienté objet, principalement connu comme le langage de script des pages web (...)

Le code JavaScript est **interprété** ou **compilé à la volée** (JIT)

C'est un langage à objets utilisant le concept de prototype, disposant d'un **typage faible** et **dynamique** (...)

Historique

Year	ECMA	Browser
1995		JavaScript was invented by **Brendan Eich**
1996		**Netscape 2** was released with JavaScript 1.0
1997		JavaScript became an **ECMA standard** (ECMA-262)
1997	ES1	**ECMAScript 1** was released
1997	ES1	**Internet Explorer 4** was the first to support ES1
1998	ES2	**ECMAScript 2** was released
1998		Netscape 42 was released with **JavaScript 1.3**



De 1995 ...

Work

Netflix

https://www.netflix.com/browse/genre/34399

NETFLIX

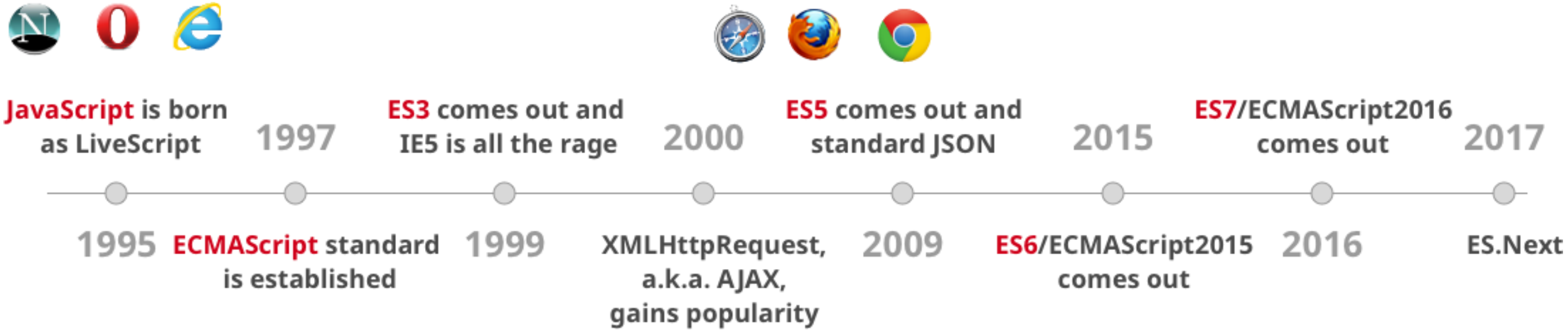
Accueil Séries Films Nouveautés les plus regardées Ma liste Explorer par langue

5

https://www.netflix.com/browse/genre/34399

… à nos jours

2008 : Moteur V8



WASM

D’abord comme Front-End

JS

Java

Front-End Developer

WASM

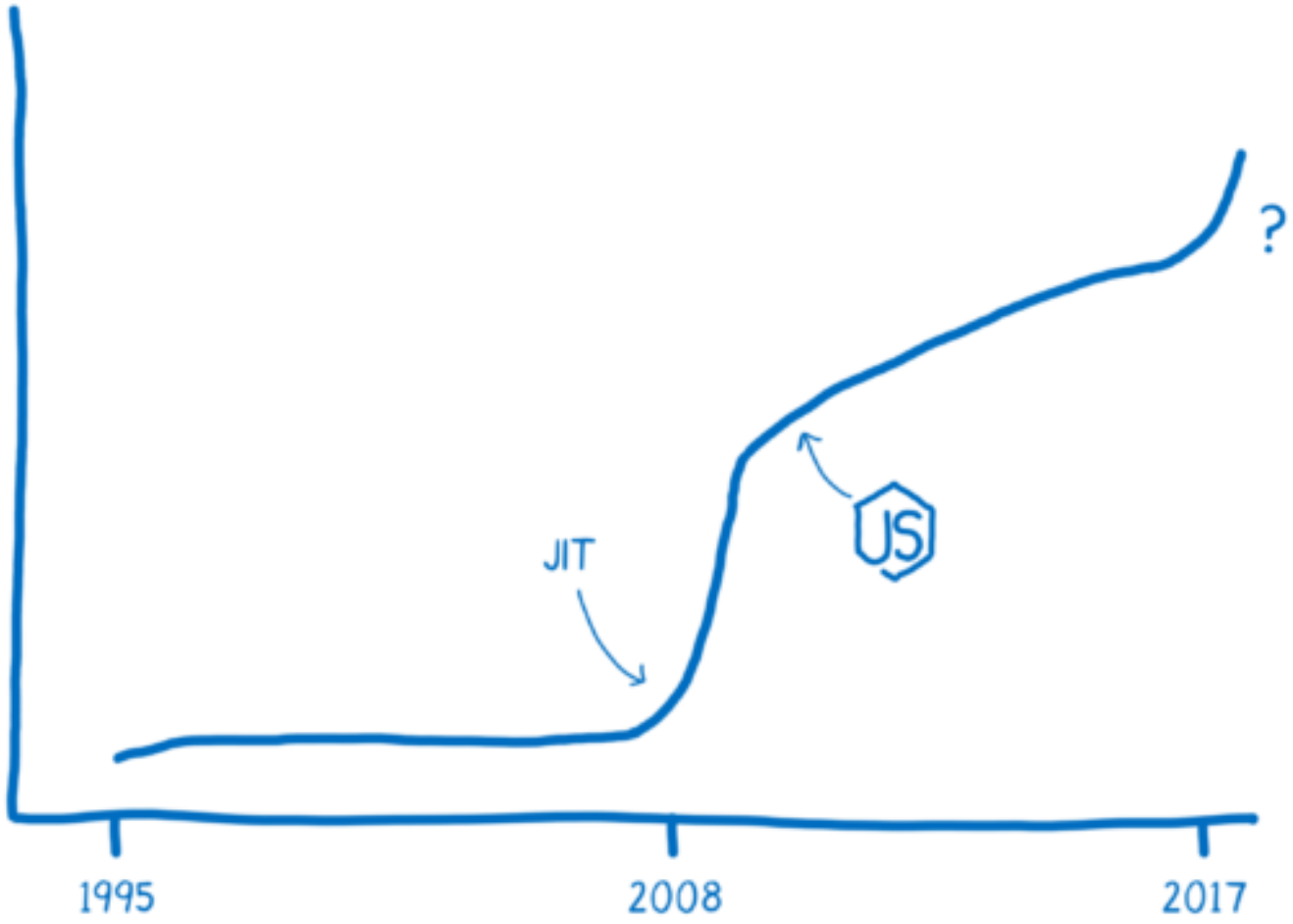
Ensuite comme Back-End

Front-End Developer

VS

Back-End Developer

Et maintenant ?



- Limité par sa conception (Interprété / JIT)
- Code existant (C, C++, C#, ...)
- Performances "moyennes"
- Sécurité côté client
- Inconsistances dues au navigateur

Tentatives de solution

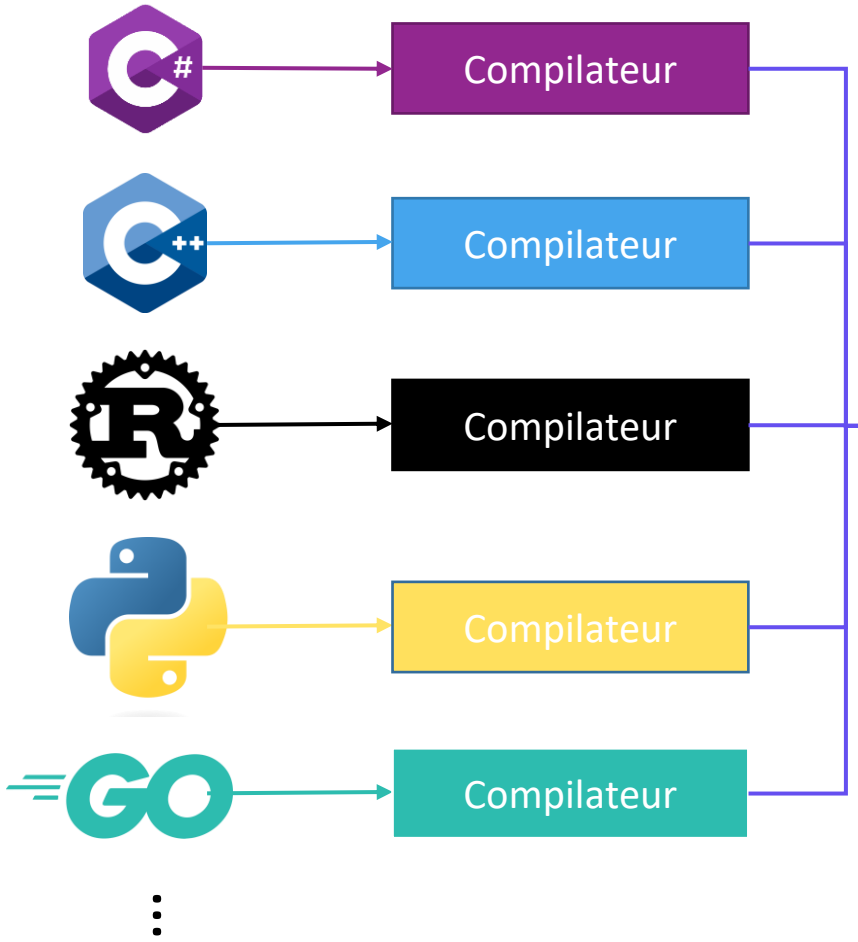
2015
Annonce de WASM

Mars 2017
MVP de l'API WASM

Novembre 2017
support de tous les navigateurs majeurs

Décembre 2019
WASM Core 1.0 (W3C)

Comment ça marche ?



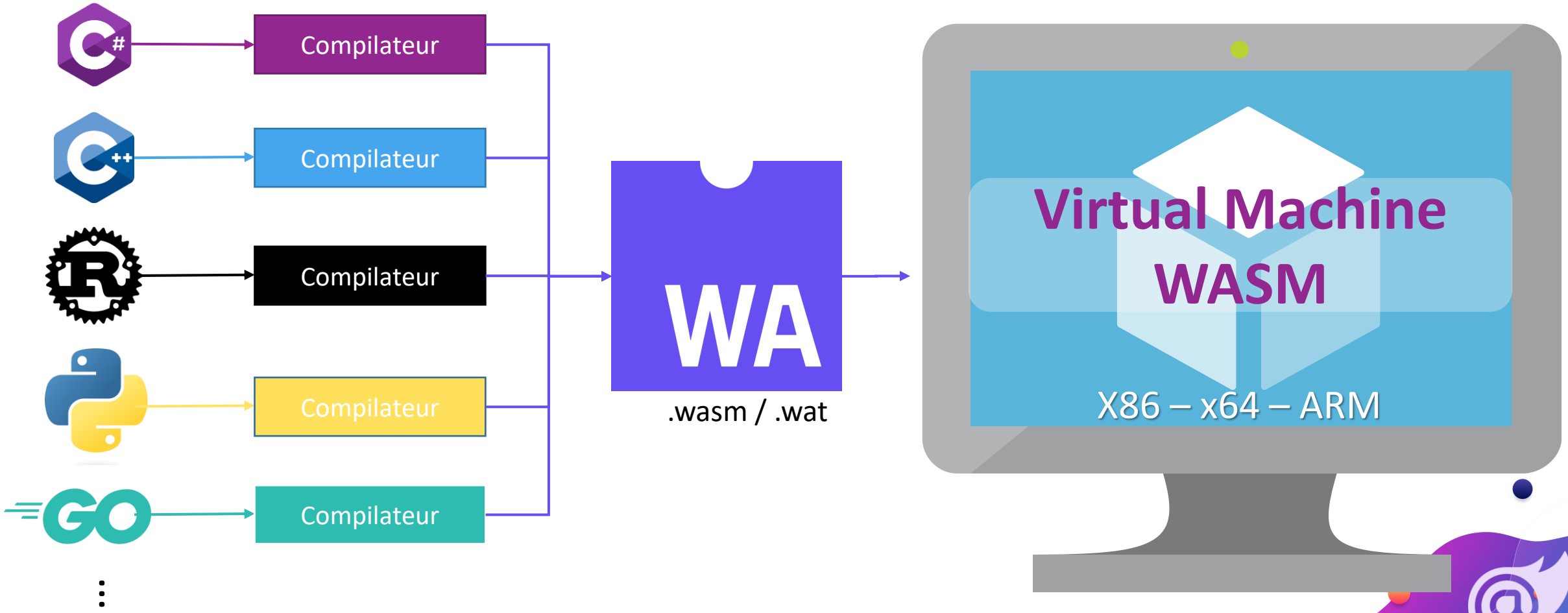
.wasm / .wat

```
(module
  (type $t0 (func (param i64) (result i
  (func $fac (export "fac") (type $t0)
    (if $I0 (result i64)
      (i64.lt_s
        (local.get $p0)
        (i64.const 1))
      (then
        (i64.const 1))
      (else
        (i64.mul
          (local.get $p0)
          (call $fac
            (i64.sub
              (local.get $p0)
              (i64.const 1))))))))

```

.wat

Comment ça marche ?




emscripten
WA
C++
Démo

*C:\VSO\Perso\emsdk\HelloWorld\hello.c - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

hello.c

```c
#include <stdio.h>

int main(){
    printf("Hello world!\n");
    return 0;
}
```

C source file | length : 84 lines : 7 | Ln : 7 Col : 1 Pos : 85 | Windows (CR LF) | UTF-8 | INS

Mieux qu’un “Hello world”

https://d07riv.github.io/diabloweb

https://d07riv.github.io/diabloweb

Adobe Photoshop in the browser thanks to WASM/Emscripten, Web Components, and Project Fugu

https://x.com/Adobe/status/1453034805004685313

JS et WASM - Complémentaires

- Flexibilité
- Simplicité
- Grande communauté

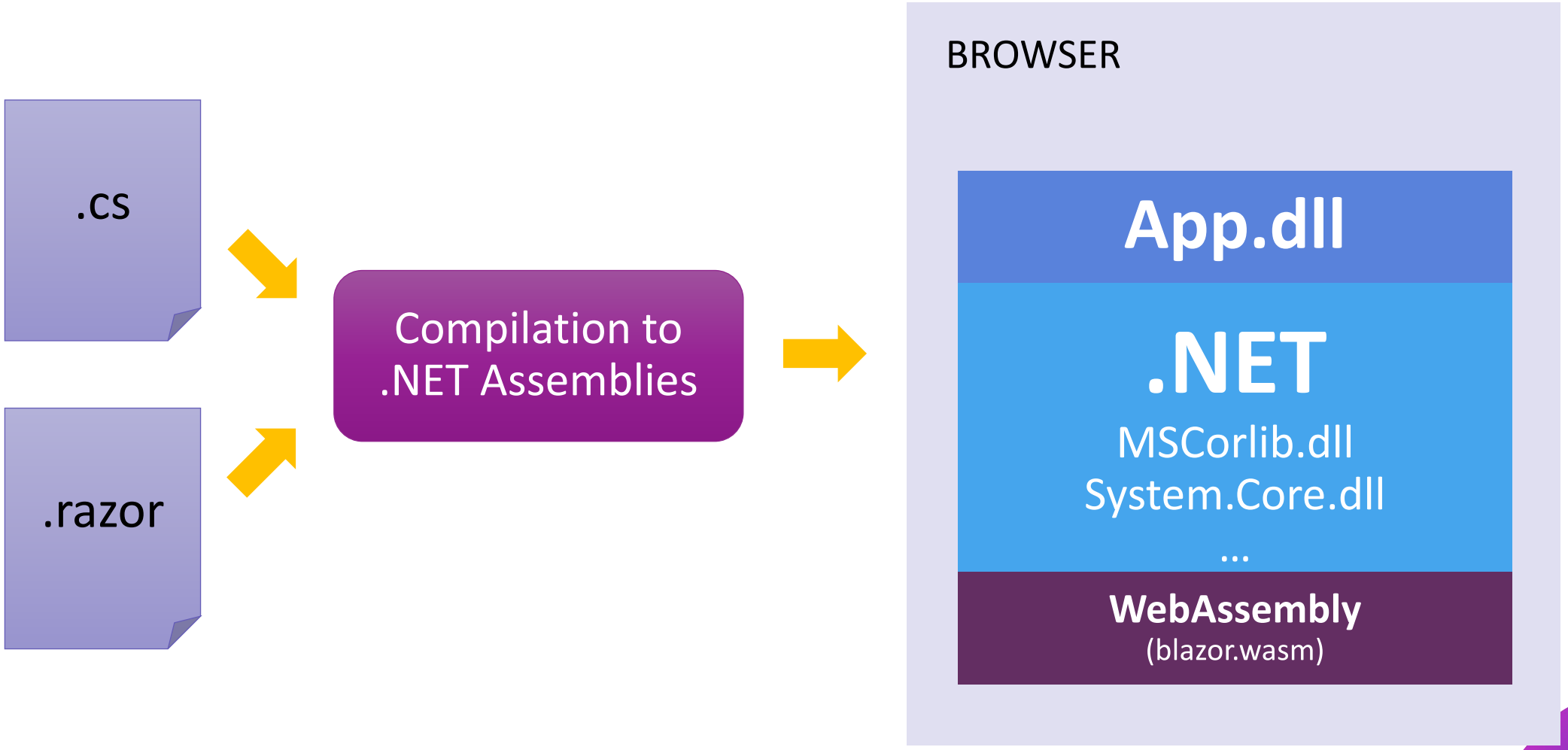
⇧ ⇩

- Performant
- Sécurisé (sandbox)
- Langages déjà connus (C#, C++, …)

Blazor

La solution de Microsoft

WASM

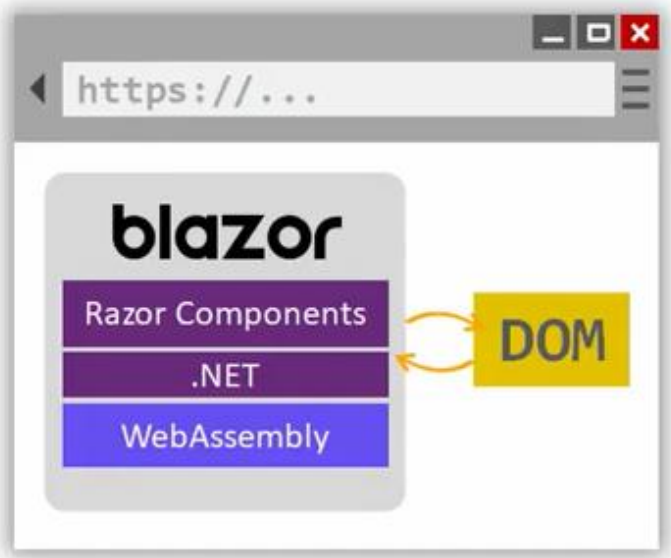

.cs
.razor
Compilation to
.NET Assemblies
BROWSER
App.dll
.NET
MSCorlib.dll
System.Core.dll
...
WebAssembly
(blazor.wasm)

Planning

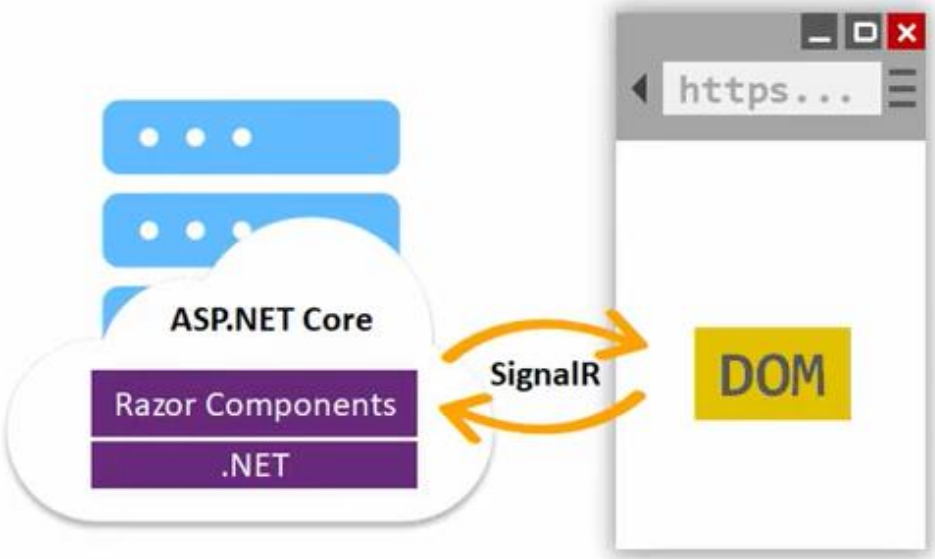

July 2019
.NET Core 3.0
RC
Sept 2019
.NET Core 3.0
GA
Nov 2019
.NET Core 3.1
LTS
Nov 2020
.NET 5.0
GA
Nov 2021
.NET 6.0
LTS
Nov 2022
.NET 7.0
GA
Nov 2023
.NET 8.0
LTS

WASM

Blazor… WebAssembly vs Serveur



VS



Single Page Applications

Applications hébergées
ASP.NET Core + SignalR

https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/blazor/server

Windows PowerShell

```
PS C:\_temp\Blazor-Demo>
```

Windows PowerShell

```
PS C:\_temp\Blazor-Demo>
```

WASM

Blazor Client ou Serveur ?

Blazor WebAssembly

- SPA, interactivité complète
- Utilisation des ressources clientes
- Support Offline, sites statiques…

- Taille à télécharger importante
- (Demande un navigateur compatible)

Blazor Server

- Peu de téléchargement, rapide à télécharger
- Architecture simplifiée
- Code ne quitte pas le serveur

- Latence
- Pas de support Offline
- Consommation de ressources sur le serveur

5,000 concurrent users: https://docs.microsoft.com/en-us/aspnet/core/blazor/host-and-deploy/server

WASM

Sept 2019

Blazor Server

Web App Server
Chaque interaction est interceptée sur le serveur

Nov 2020

Blazor WebAssembly

Web App avec interactions sur le client
Chargée depuis un serveur web

Blazor PWA

Affichée comme une application native
Fonctionne offline et online

Blazor Hybrid

Rendus natifs via Electron ou WebView
Affichée comme une application native
Fonctionne offline et online

Blazor Native

Même modèle de programmation,
Mais basé sur un rendu graphique non-HTML

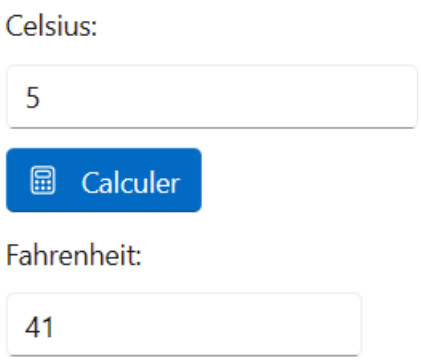
$$F = \frac{9}{5}C + 32$$

Coding

Agenda

- Créer un composant
- Gérer une route
- Ajouter des [Parameter]
- Data binding
- Utiliser FluentUI Blazor
- Ajouter un UnitTest

https://github.com/dvoituron/temperature-wasm



Composant

TemperaturePage.razor

```
@page "/temperature"
@page "/temperature/{Celsius:int}"
@using Microsoft.Fast.Components.FluentUI

<FluentStack Orientation="Orientation.Vertical">

    <FluentLabel>Celsius:</FluentLabel>
    <FluentNumberField @bind-Value="@Celsius" />

    <FluentButton Appearance="Appearance.Accent"
                  OnClick="btnCalculate_Click">
        <FluentIcon Icon="Icons.Regular.Size16.Calculator"
                    Color="Color.Lightweight"
                    Slot="start" />
        Calculer
    </FluentButton>

    <FluentLabel>Fahrenheit:</FluentLabel>
    <FluentNumberField @bind-Value="@Fahrenheit"
                       Readonly="true" />
</FluentStack>
```

TemperaturePage.razor.cs

```
namespace MyTemperature.Pages;
using Microsoft.AspNetCore.Components;

public partial class TemperaturePage
{
    [Parameter]
    public int Celsius { get; set; } = 0;

    public int Fahrenheit { get; set; } = 0;

    protected override void OnInitialized()
    {
        btnCalculate_Click();
    }

    public void btnCalculate_Click()
    {
        Fahrenheit = Convert.ToInt32(1.8 *
                         Celsius + 32);
    }
}
```

Composant

TemperatureTest

```
[Fact]
public void Temperature_20Celsius_68Fahrenheit()
{
    // Arrange
    using var ctx = new TestContext();

    // Act
    var page = ctx.RenderComponent<TemperaturePage>(parameters =>
    {
        parameters.Add(p => p.Celsius, 20);
    });

    // Assert
    Assert.Contains("value=\"68\"", page.Markup);

    var fahrenheit = page.FindComponents<FluentNumberField<int>>().Last();
    Assert.Equal(68, fahrenheit.Instance.Value);
}
```


dvoituron
Merci
https://github.com/dvoituron/temperature-wasm