



Introduction to Blazor

C# in the browser

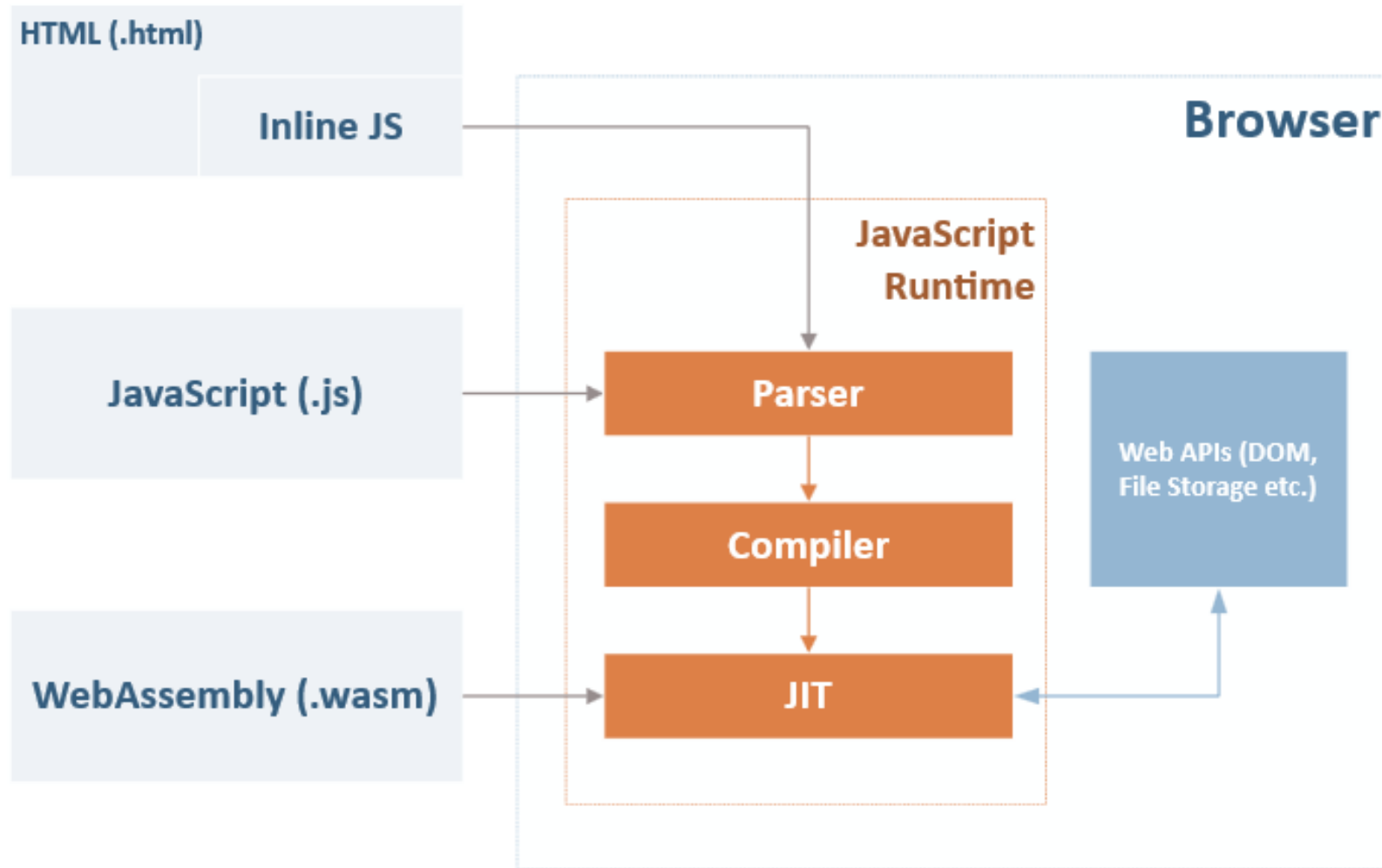


WebAssembly

WebAssembly (abbreviated *Wasm*) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications.



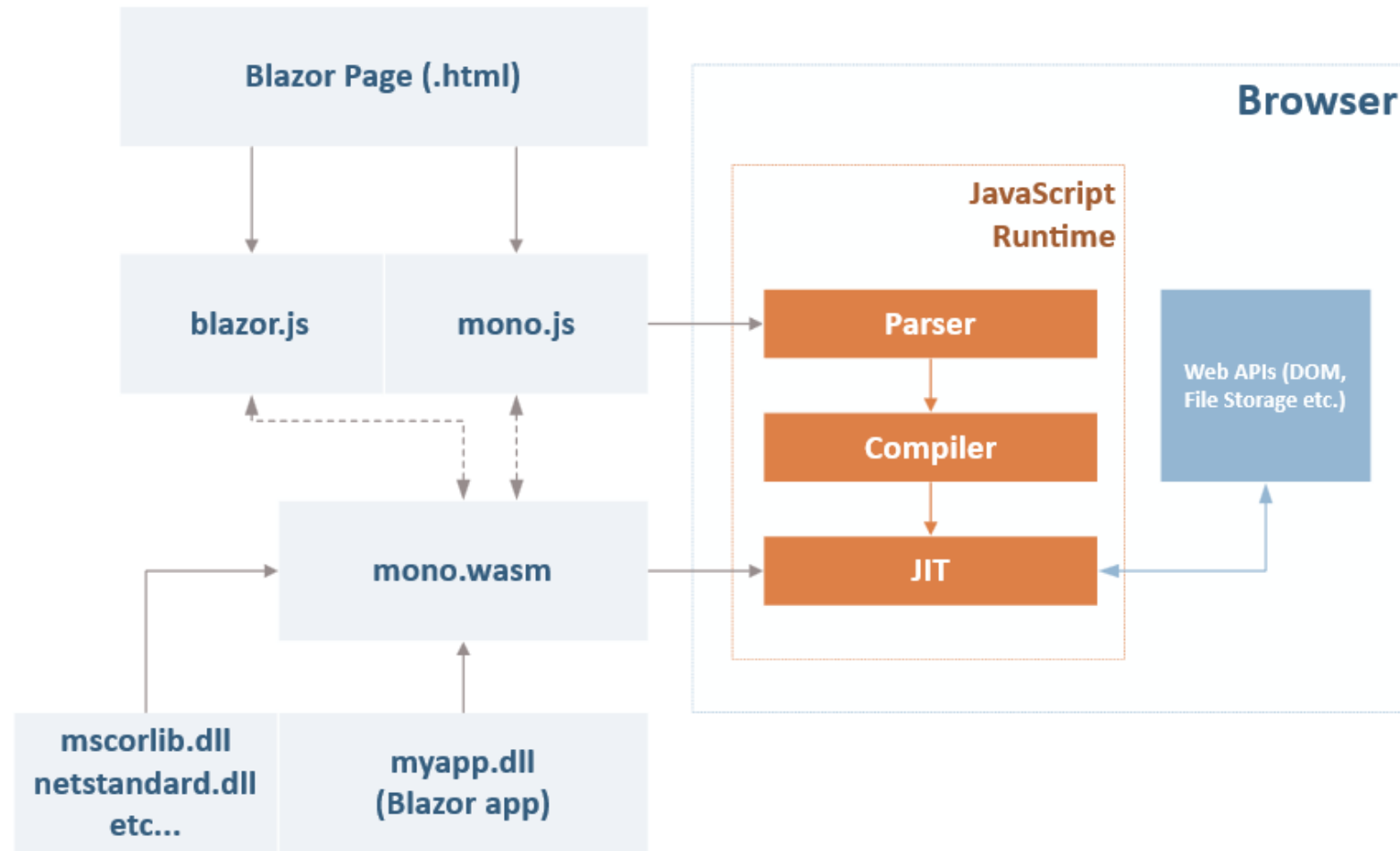
WebAssembly





















**“Mono C runtime into web assembly,
and then uses Mono’s IL interpreter to
run managed code.”**

Miguel de Icaza – Aug 9, 2017

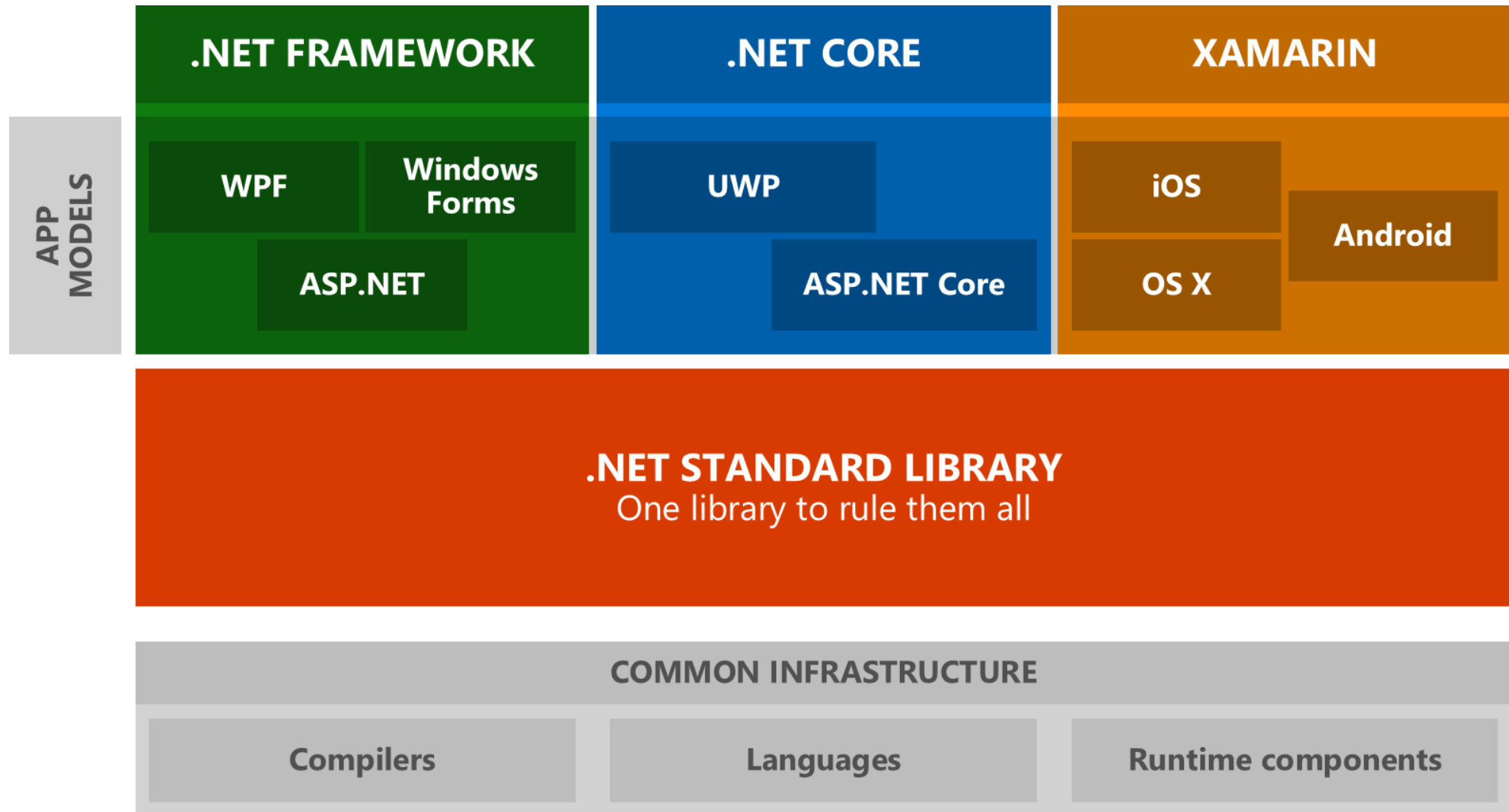
WASM and Mono



Example – network from web browser

| Name | Status | Type | Size |
|--|--------|------------|---------|
|  localhost | 200 | document | 698 B |
|  bootstrap.min.css | 200 | stylesheet | 24.5 KB |
|  site.css | 200 | stylesheet | 1.2 KB |
|  bootstrap-native.min.js | 200 | script | 8.4 KB |
|  blazor.js | 200 | script | 59.3 KB |
|  mono.js | 200 | script | 50.3 KB |
|  mono.wasm | 200 | fetch | 691 KB |
|  favicon.ico | 200 | text/html | 698 B |
|  BlazorDemo.dll | 200 | xhr | 6.7 KB |
|  Microsoft.AspNetCore.Blazor.Browser.dll | 200 | xhr | 16.4 KB |
|  Microsoft.AspNetCore.Blazor.dll | 200 | xhr | 46.8 KB |
|  Microsoft.Extensions.DependencyInjection.Abstractions.dll | 200 | xhr | 18.0 KB |
|  Microsoft.Extensions.DependencyInjection.dll | 200 | xhr | 23.0 KB |
|  mscorlib.dll | 200 | xhr | 660 KB |
|  netstandard.dll | 200 | xhr | 9.3 KB |
|  System.Core.dll | 200 | xhr | 140 KB |
|  System.dll | 200 | xhr | 39.8 KB |
|  System.Net.Http.dll | 200 | xhr | 30.9 KB |
| glyphicons-halflings-regular.woff2 | 200 | font | 17.8 KB |

.NET Standard library



Blazor Elements

- Component Model
- Routing
- Layouts
- Forms and Validation
- Dependency Injection
- Javascript Interop

Blazor Lifecycle

OnInit

Ready to start after receiving parameters from the parent in the render tree.

OnParametersSet

Invoked when component receives params from the parent in the render tree.

ShouldRender

Suppress refreshing the UI.

One-way data binding

```
@page "/counter"
```

```
<h1>Counter</h1>
<p>Current count: @currentCount</p>
<input type="number" bind="incrementAmount" />
<button onclick="@IncrementCount">Click me</button>
```

```
@functions {
    public int currentCount { get; set; } = 0;
    public int incrementAmount { get; set; } = 1;

    void IncrementCount()
    {
        currentCount += incrementAmount;
    }
}
```

Two-way data binding

```
@page "/counter"
```

```
<h1>Counter</h1>
<p>Current count: @currentCount</p>
<input type="number" bind="incrementAmount" />
<button onclick="@IncrementCount">Click me</button>
```

```
@functions {
    public int currentCount { get; set; } = 0;
    public int incrementAmount { get; set; } = 1;

    void IncrementCount()
    {
        currentCount += incrementAmount;
    }
}
```

Event binding

```
@page "/counter"
```

```
<h1>Counter</h1>
<p>Current count: @currentCount</p>
<input type="number" bind="incrementAmount" />
<button onclick="@IncrementCount">Click me</button>
```

```
@functions {
    public int currentCount { get; set; } = 0;
    public int incrementAmount { get; set; } = 1;
```

```
void IncrementCount()
{
    currentCount += incrementAmount;
}
```

```
}
```

Client side routing

```
@page "/counter"
```

```
<h1>Counter</h1>
<p>Current count: @currentCount</p>
<input type="number" bind="incrementAmount" />
<button onclick="@IncrementCount">Click me</button>
```

```
@functions {
    public int currentCount { get; set; } = 0;
    public int incrementAmount { get; set; } = 1;

    void IncrementCount()
    {
        currentCount += incrementAmount;
    }
}
```

Componet properties

```
@page "/counter"
```

```
<h1>Counter</h1>
<p>Current count: @currentCount</p>
<input type="number" bind="incrementAmount" />
<button onclick="@IncrementCount">Click me</button>
```

```
@functions {
```

```
    public int currentCount { get; set; } = 0;
    public int incrementAmount { get; set; } = 1;
```

```
    void IncrementCount()
    {
        currentCount += incrementAmount;
    }
}
```

Component usage

```
<Counter currentCount="40" incrementAmount="20"/>
```

JavaScript interop – C# calling JavaScript

```
<button onclick="@CallDoSomething">Do something</button>
```

```
<script>  
    function doSomething(message) {  
        console.log(message);  
        return true;  
    }
```

```
    Blazor.registerFunction('doSomething', doSomething);  
</script>
```

```
@functions {  
    public void CallDoSomething()  
    {  
        RegisteredFunction.Invoke<bool>("doSomething", "Hello World");  
    }  
}
```


Dependency Injection

```
class Program
{
    static void Main(string[] args)
    {
        var serviceProvider = new BrowserServiceProvider(services =>
        {
            // Add any custom services here
        });

        new BrowserRenderer(serviceProvider).AddComponent<App>("app");
    }
}
```

Dependency Injection

```
@page "/fetchdata"  
@inject HttpClient Http
```

```
<h1>Weather forecast</h1>  
└
```

Thanks for your attention!

