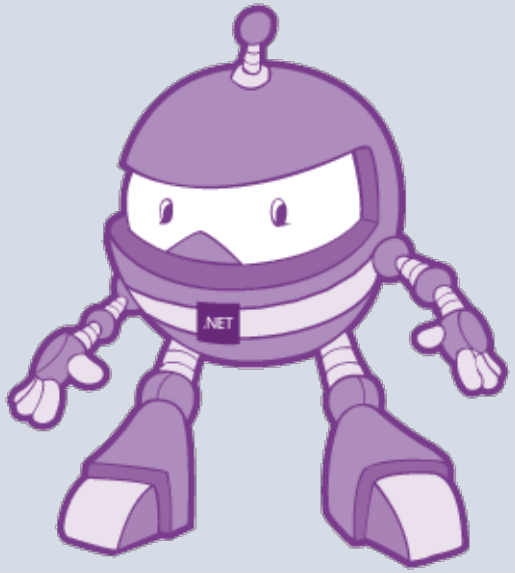
# C# Everywhere with Blazor



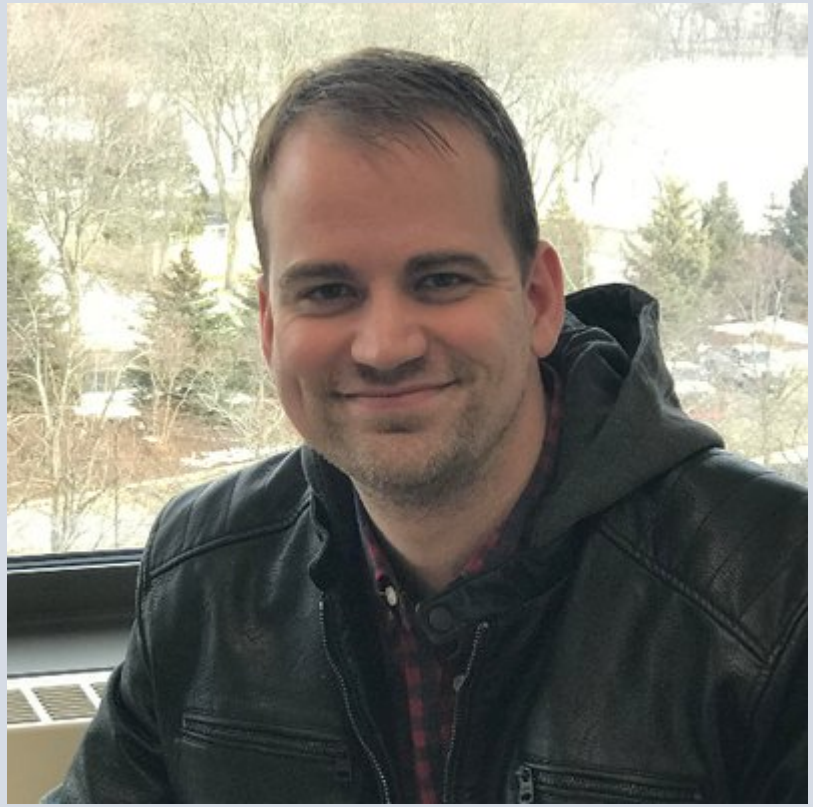

Dave Brock

Central Wisconsin IT Conference

October 6, 2018

# Hello, I'm Dave

- Dave Brock, officially
- "Full stack" software engineer in Madison
- Focus on .NET stack
    - C#
    - .NET Core
    - SQL Server
    - JavaScript/TypeScript
- Advocate of <*blink*> tag from 1995-1999 (#sorrynotsorry)



# The web is amazing

- We have evolved and can write code against so many devices:
  - Mobile!
  - Tablets!
  - Gaming and consoles!
  - Embedded devices (like cars)!

# What about the browser?

# The browser *is* JavaScript







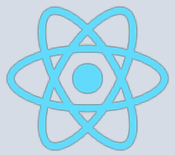












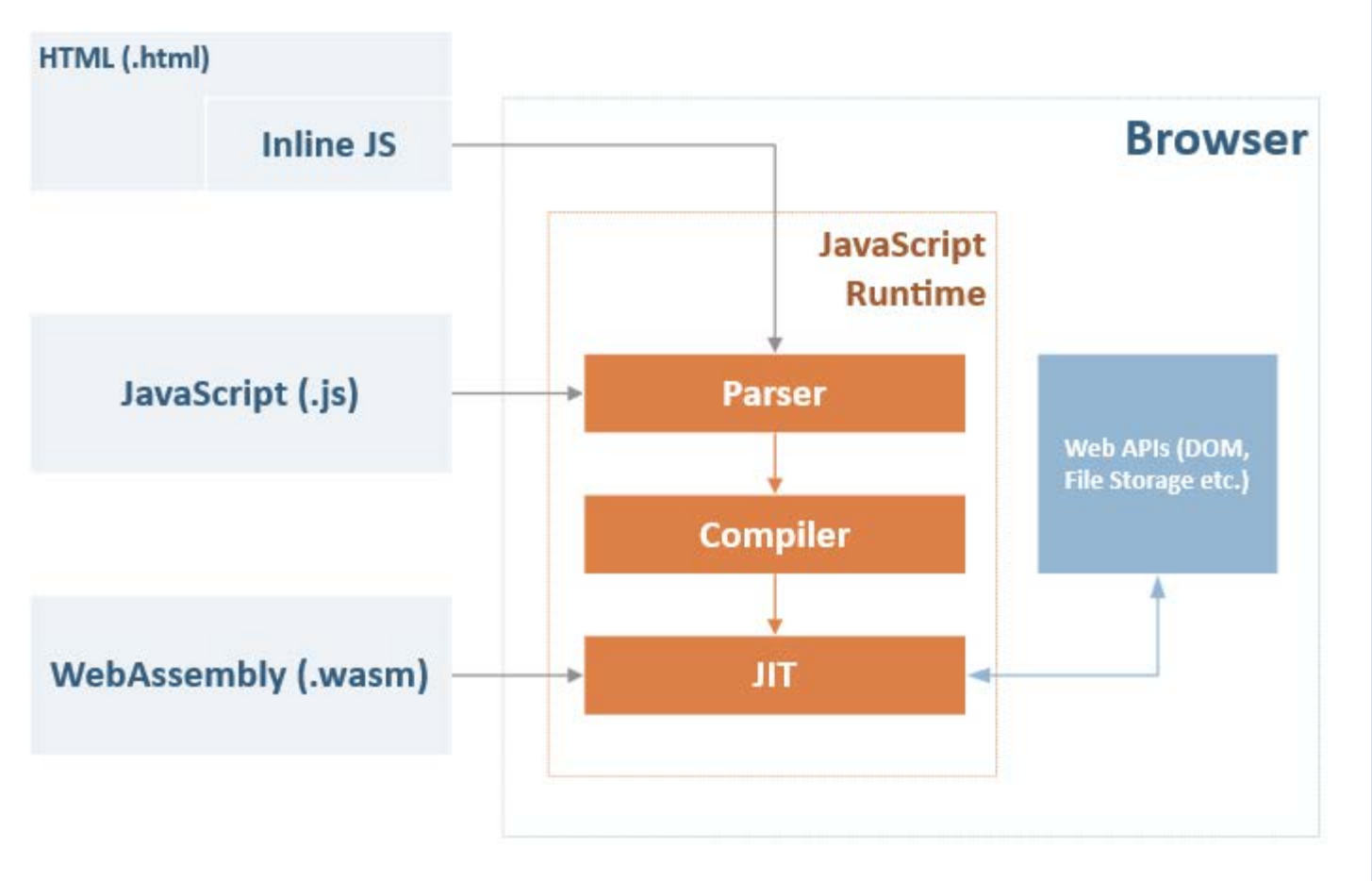

# The choice is yours

# Some wisdom about writing for the browser

Any developer can write code in the browser for any language he/she wants…
**so long as it's JavaScript.**

Henry Ford, 1909
(paraphrasing)

# How WebAssembly Works (Diagram)




*~~Stolen~~ Borrowed from Dave Glick's article at https://daveaglick.com/posts/blazor-razor-webassembly-and-mono*

# Is WebAssembly trying to replace JavaScript?


# NO!

From webassembly.org:

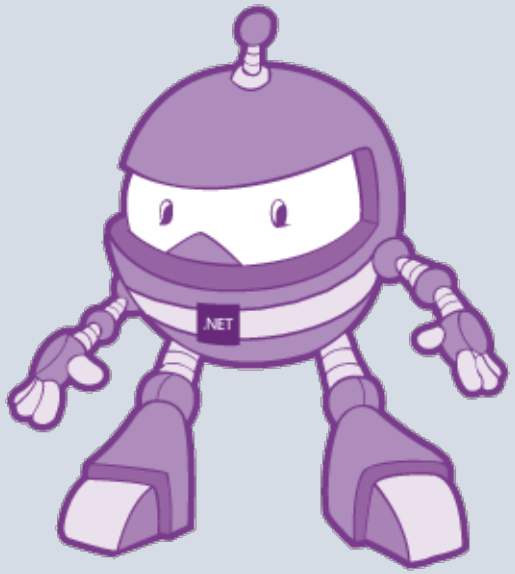
**“WebAssembly is designed to be a complement to … JavaScript. While WA will over time allow many languages to be compiled to the Web, JavaScript … will remain the single, privileged dynamic language of the web.”**

On a practical level, the architecture of WebAssembly does not allow access to web functions like manipulating the DOM or calling browser APIs (like local storage)

# Why use .NET in the browser, anyway?

Using .NET in the browser offers:

- **Wide browser support**: runs on open web standards with no plugins and no code transpilation
- **Leverage existing skills**: use your existing .NET skillset to write client-side code
- **Tooling**: Visual Studio and Visual Studio Code offers a first-rate development experience
- **Modern languages**: .NET languages are constantly improving




*https://blazor.net/docs/introduction/faq.html*

# What is Blazor?


**An *experimental* single-page web app framework built on .NET that runs in the browser with WebAssembly.**

*https://blazor.net/docs/introduction/faq.html*


# Experimental.

# Blazor goals?


Component model

JavaScript interoperability

Rich tooling

Layouts

Live (“hot”) reloading in the browser

Full .NET debugging

Forms and validation

Backward compatibility with *asm.js*

Publishing and app size trimming

Routing

Server-side rendering

Dependency injection

*https://blazor.net/docs/introduction/faq.html*

# Time for a talk

# Why Blazor is different

Unlike plug-ins like Silverlight (or Flash), Blazor is built on open web standards and leverages the advantages of WebAssembly

Because it is built on Web Assembly, it offers:

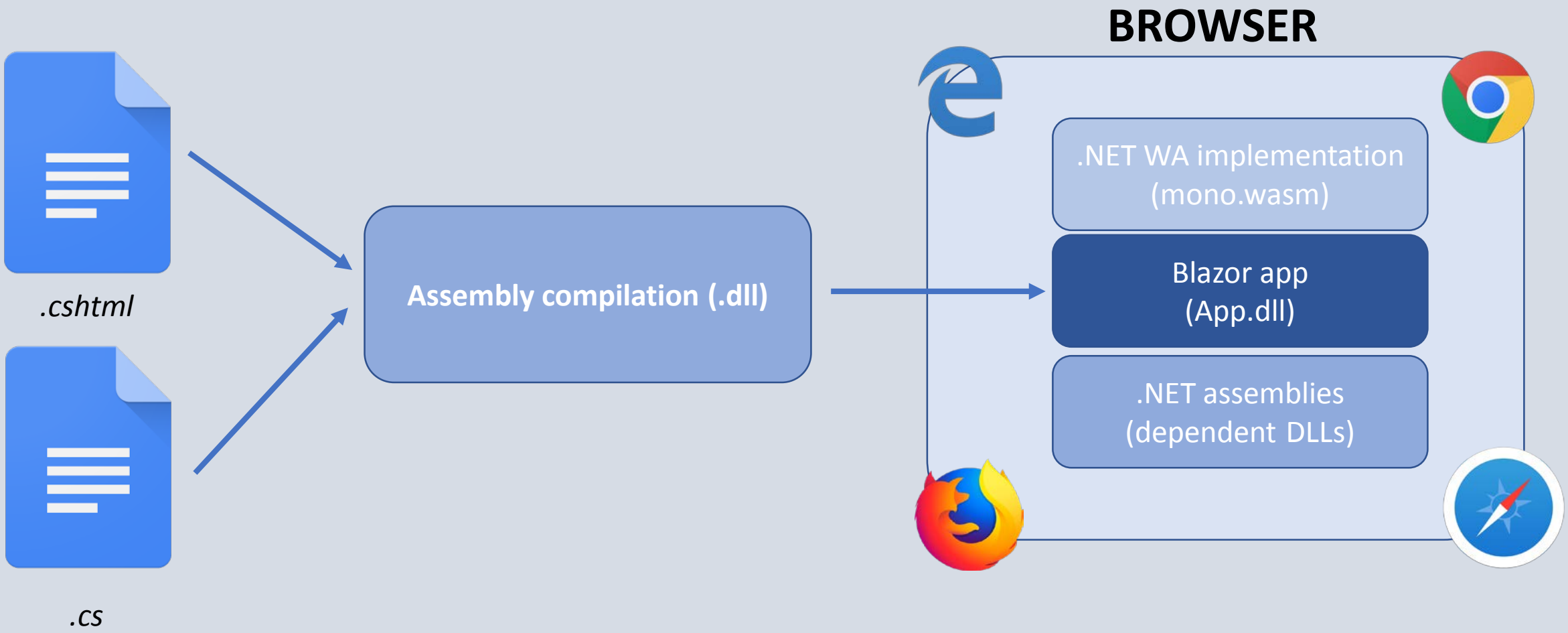
- Near native performance
- Safety (it runs in the same sandbox as JS to prevent malicious behavior on the client machine)
- Advantages over native JS like high-memory processing and complex multi-threading

# How does Blazor work with WebAssembly?

We need a runtime compiled for WA that is aware of .NET.

- We use Mono, an open-source and highly-portable .NET runtime. For Blazor, Mono is the only runtime that needs to be compiled to WebAssembly
- Mono is used by Xamarin for native client apps on Android, iOS, and macOS, and Unity for game development. Since Blazor is a client-side UI framework, Mono is the natural choice.
- Mono is preferred over .NET Core, which is primarily for server applications

# How Does Blazor Work with Web Assembly?



# Getting started with Blazor

Blazor is now on version 0.6.0, **released 10/2/2018**

Prerequisites:

- .NET Core 2.1 SDK (2.1.402 or later)
- Visual Studio 2017 (15.8 or later) with the **ASP.NET and web development workload** selected
- The Blazor Language Services extension from the Visual Studio Marketplace
- Installing the Blazor templates from the command line:

```
dotnet new –i Microsoft.AspNetCore.Blazor.Templates
```

Details at ***https://blazor.net/docs/get-started.html***

# Blazor: Key Concepts – Razor

Razor is a templating engine for creating dynamic web pages in .NET. You can use a combination of Razor syntax and C# code in *.cshtml* files.

```
<div>
    <h1>@Title</h1>
    <button onclick=@DoSomething>Do Something!</button>
</div>

@functions {
  public string Title { get; set; }
  public Action DoSomething { get; set; }
}
```

# Blazor: Key Concepts – Components

Blazor applications are built with components, which can be viewed of a piece of section of the UI.

Components can then be tested and reused within the application and even between projects!

To Blazor, a component is a class – usually a C# (*.cs*) or Razor (*.cshtml*) file.

# Blazor: Key Concepts – JavaScript Interop

You will still need JS for **browser APIs, DOM manipulation, and access to JS libraries**.

Blazor can use any library or API that JavaScript uses.

C# can call into JavaScript code, and JavaScript code can call into C# code.

# Blazor: Key Concepts – JavaScript Interop

```
window.exampleFunction = {
    showPrompt: function (message {
        return prompt(message, ‘Anything’);
    }
};


public class Interop
{
    public static Task<string> Prompt(string message)
    {
        return JSRuntime.Current.InvokeAsync<string>(
           “exampleFunction.showPrompt”, message);
     }
}
```

# Demo Time


https://blazorimageofday.azurewebsites.net/

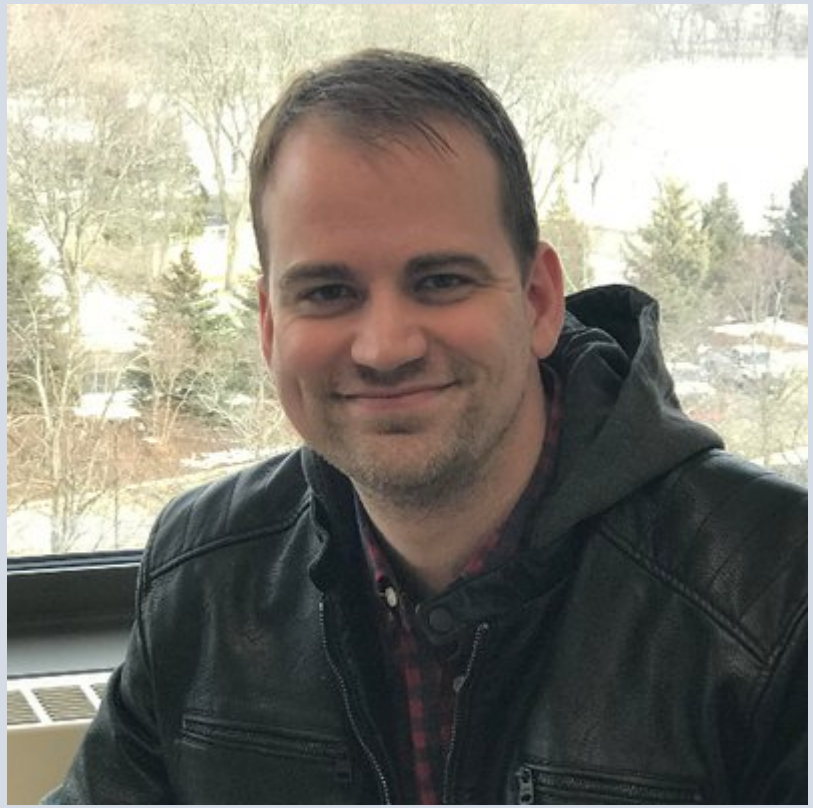
# The different forms of Blazor and what's coming

Blazor comes in three distinct project types (you will see this when you create a new project in Visual Studio)

- **Client-Side –** what we discussed today. Web Assembly and client rendering. Will be experimental for awhile
- **Full Stack -** .NET Core hosting, a Web API, and shared logic capabilities
- **Server-Side (Razor Components)** – uses a SignalR client with a .NET Core backend – no WebAssembly needed! ***Coming to .NET Core 3.0!***

# Thank you!

Reach out anytime (info in header below)



## Resources

- Web Assembly
- Blazor documentation
- Razor documentation
- C# Guide (Microsoft Docs)
- Blazor on GitHub
- Blazor on Gitter

(slides and code)