.NET 5

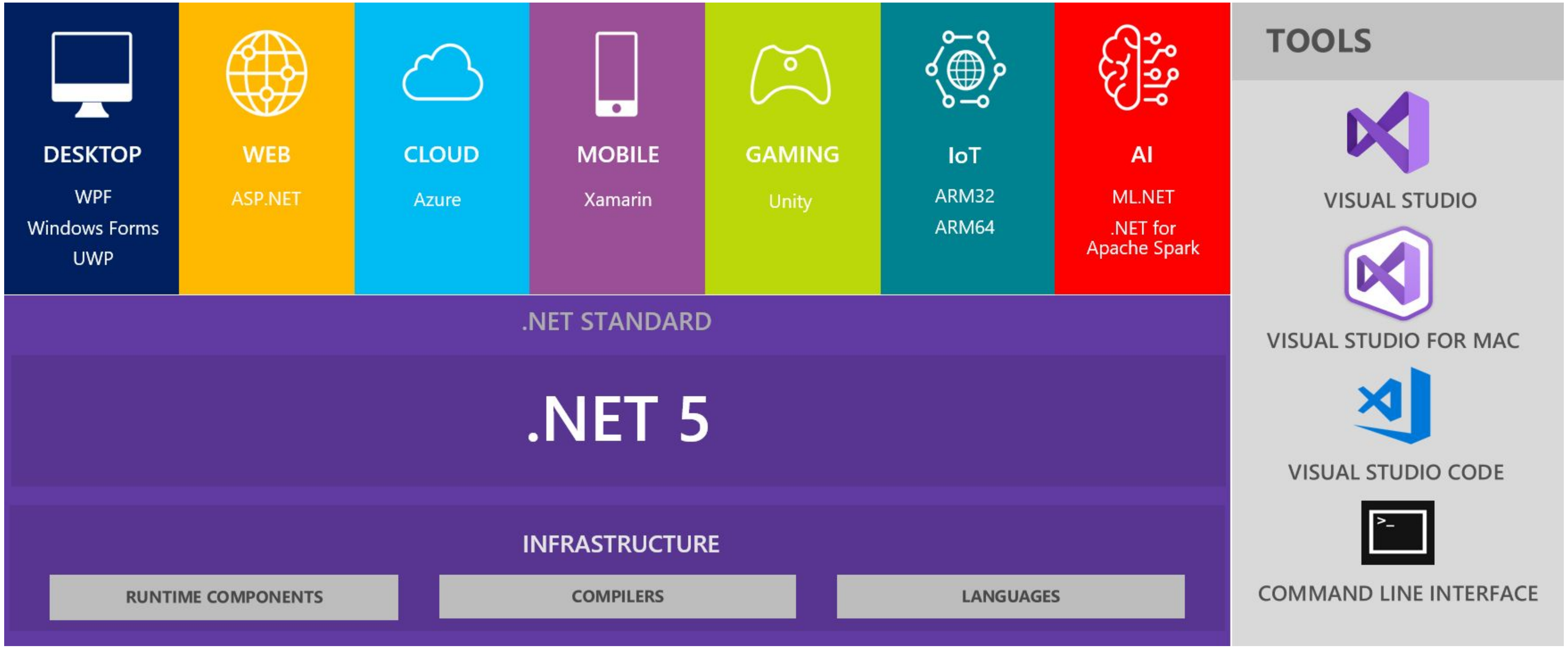
What's new

VISMA | DataLøn

The goals

- Produce a single .NET runtime and framework that can be used everywhere and that has uniform runtime behaviors and developer experiences.
- Expand the capabilities of .NET by taking the best of .NET Core, .NET Framework, Xamarin and Mono.
- Build that product out of a single code-base that developers (Microsoft and the community) can work on and expand together and that improves all scenarios.

.NET - A unified platform



The Naming

- **.NET 5.0** is the next major release of **.NET Core** following **3.1**
- **ASP.NET Core 5.0** is based on **.NET 5.0** but retains the name **"Core"** to avoid confusing it with **ASP.NET MVC 5**
- **Entity Framework Core 5.0** retains the name **"Core"** to avoid confusing it with **Entity Framework 5** and **6**

Where is .NET Core 4.0?

- Skipped version numbers **4.x** to avoid confusion with **.NET Framework 4.x**.
- Dropped **"Core"** from the name to emphasize that this is the main implementation of **.NET** going forward.
- **.NET 5.0** supports more types of apps and more platforms than **.NET Core** or **.NET Framework**.

.NET 5.0 doesn't replace .NET Framework

- **.NET 5.0** is the main implementation of **.NET** going forward.
- **.NET Framework 4.x** is still supported.
- The following technologies are dropped:
  - **Web Forms**
  - **Windows Workflow**
- Recommended alternatives:
  - **ASP.NET Core Blazor** or **Razor Pages**
  - Open-source **CoreWF**

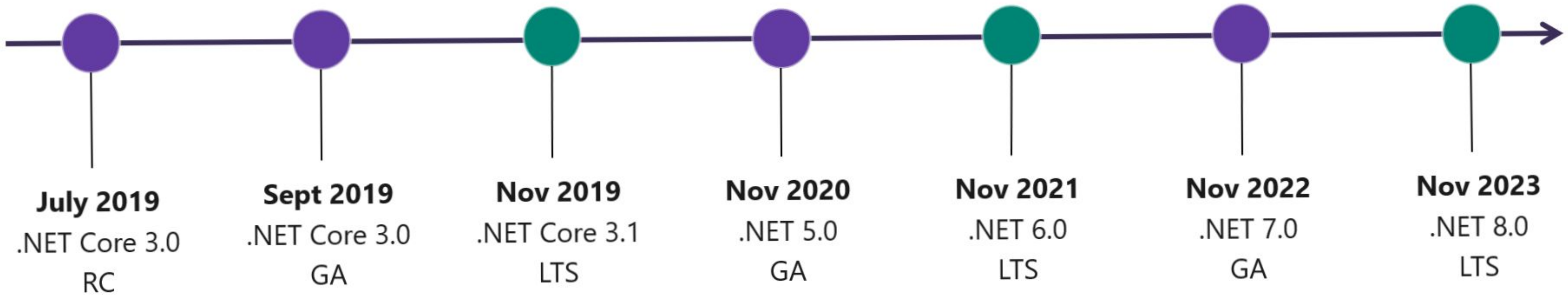
.NET 5.0 doesn't replace .NET Standard

- You can specify the `net5.0` **target framework moniker (TFM)** for all project types.
- Sharing code between **.NET 5** workloads is simplified in that all you need is the `net5.0` **TFM**, which combines and replaces the `netcoreapp` and `netstandard` **TFM**s.
- To share code between **.NET Framework**, **.NET Core**, and **.NET 5** workloads, you can do so by specifying `netstandard2.0` as your **TFM**.

VISMA | DataLøn

.NET Schedule

- Major releases every year (November)
- LTS for even numbered releases
- Predictable schedule, minor releases if needed



Highlights

- **C# 9** and **F# 5**
- C# **Source Generators**
- Improved **performance**
- Enhanced performance of **Json serialization**, **regular expressions**, and **HTTP**
- Dropped **P95 latency**
- Better **application deployment** options
- Platform scope expanded with **Windows Arm64** and **WebAssembly**
- Already in production - **Bing.com**, **dot.net**

Application deployment

- Improved single file applications.
- Reduced container size for docker multi-stage builds.
- Better support for deploying ClickOnce applications.

C# 9 - What’s new

- Top-level statements
- Init-only properties
- Records
- Pattern matching enhancements
- Performance and interop
- Fit and finish features
- Support for code generators

C# 9 - Top-level statements

Before:

```
using System;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

After:

```
System.Console.WriteLine("Hello, world");
```

C# 9 - Init-only properties

- Introduces an `init` accessor that is a variant of the `set` accessor which can only be called during object initialization
- Makes individual properties immutable
- Any subsequent assignment to the `FirstName` and `LastName` properties is an error.

```
public class Person
{
    public string FirstName { get; init; }
    public string LastName { get; init; }
}
```

C# 9 - Records

- **Record types** are a reference types that provides synthesized methods to provide value semantics for equality.
- Makes the whole object immutable.
- Object behaves like a value.
- Records are meant to be seen more as “values” – data! – and less as objects.

```
public record Person
{
    public string FirstName { get; init; }
    public string LastName { get; init; }
}
```

C# 9 - Pattern matching enhancements

- ***Type patterns*** - match a variable is a type
- ***Parenthesized patterns*** - enforce or emphasize the precedence of pattern combinations
- ***Conjunctive*** `and` ***patterns*** - require both patterns to match
- ***Disjunctive*** `or` ***patterns*** - require either pattern to match
- ***Negated*** `not` ***patterns*** - require that a pattern doesn’t match
- ***Relational patterns*** - require the input be less than, greater than, less than or equal, or greater than or equal to a given constant.

```
public static bool IsLetter(this char c) =>
    c is >= 'a' and <= 'z' or >= 'A' and <= 'Z';
public static bool IsLetterOrSeparator(this char c) =>
    c is (>= 'a' and <= 'z') or (>= 'A' and <= 'Z') or '.' or ',';

if (e is not null) {  // ... }
if (e is not string) {  // ... }
```

C# 9 - Performance and interop

- Native sized integers
- Function pointers
- Skip `localsinit` flag

C# 9 - Native sized integers

- Integer types `nint` and `nuint`
- Underlying types `System.IntPtr` and `System.UIntPtr`

```
nint x = IntPtr.Zero;
nuint y = 234;
```

C# 9 - Function pointers

- Provide an easy syntax to access the IL opcodes `ldftn` and `calli`.
- You can declare function pointers using new `delegate*` syntax.
- A `delegate*` type is a pointer type.
- Invoking the `delegate*` type uses `calli`, in contrast to a `delegate` that uses `callvirt` on the `Invoke()` method.

```
delegate*<string, int> functionPointer = &GetLength;
int length = functionPointer("Hello, world");
static int GetLength(string s) => s.Length;
```

C# 9 - Skip localsinit flag

- You can add the `SkipLocalsInitAttribute` to instruct the compiler not to emit the `localsinit` flag.
- This flag instructs the CLR to zero-initialize all local variables (since C# 1.0).
- You may add it to a single method or property, or to a class, struct, interface, or even a module.

```
var x = stackalloc Foo[1000];   // No initialization overhead

[System.Runtime.CompilerServices.SkipLocalsInitAttribute]
struct Foo
{
    public int X, Y, Z;
}
```

C# 9 - Fit and finish features

- Target-typed `new`
- Static lambdas
- Covariant return types
- `GetEnumerator` extension method in `foreach` loops
- Lambda discard parameters
- Attributes on local functions

C# 9 - Target-typed new

The type in a `new` expression can be omitted when the created object's type is already known.

```
private List<WeatherObservation> _observations = new();

...
public WeatherForecast ForecastFor(DateTime forecastDate,
WeatherForecastOptions options)
...
var forecast = station.ForecastFor(DateTime.Now.AddDays(2), new());

WeatherStation station = new() { Location = "Seattle, WA" };
```

C# 9 - Static lambdas

- Analogous to the `static` local functions.
- Can't capture local variables or instance state.
- The `static` modifier prevents accidentally capturing other variables.

```
"This is a test".Select(static c => char.ToUpper(c));
```

C# 9 - Covariant return types

An override method can return a type derived from the return type of the overridden base method.

```
class A
{
   public virtual A Clone() => new A();
}

class B : A
{
   public override B Clone() => new B();
}

var b = new B();
var clone = b.Clone();
```

C# 9 - GetEnumerator extension method

- The `foreach` loop will recognize and use an extension method `GetEnumerator` that otherwise satisfies the `foreach` pattern.
- You can add `foreach` support to any type.

```
static class Extensions
{
   public static IEnumerator<int> GetEnumerator (this int x) =>
      Enumerable.Range(0, x).GetEnumerator();
}

foreach (int i in 10)
{
    // ...
}
```

C# 9 - Lambda discard parameters

You can use discards `_` as parameters to lambda expressions.

```
"Password".Select(_ => "*");
```

C# 9 - Attributes on local functions

You can now apply attributes to local functions.

```
class MyAttribute : Attribute { }

[MyAttribute]
void Foo() => "Foo";

class AnotherDemo
{
   void Foo()
   {
      [MyAttribute]
      void Local() { }
   }
}
```

C# 9 - Support for code generators

- A component you can write that is similar to a roslyn analyzer or code fix.
- Analyze code and write new source code files as part of the compilation process.
- Can only add code, they aren't allowed to modify any existing code in the compilation.
- Added extensions to **partial method syntax**, and **module initializers**.

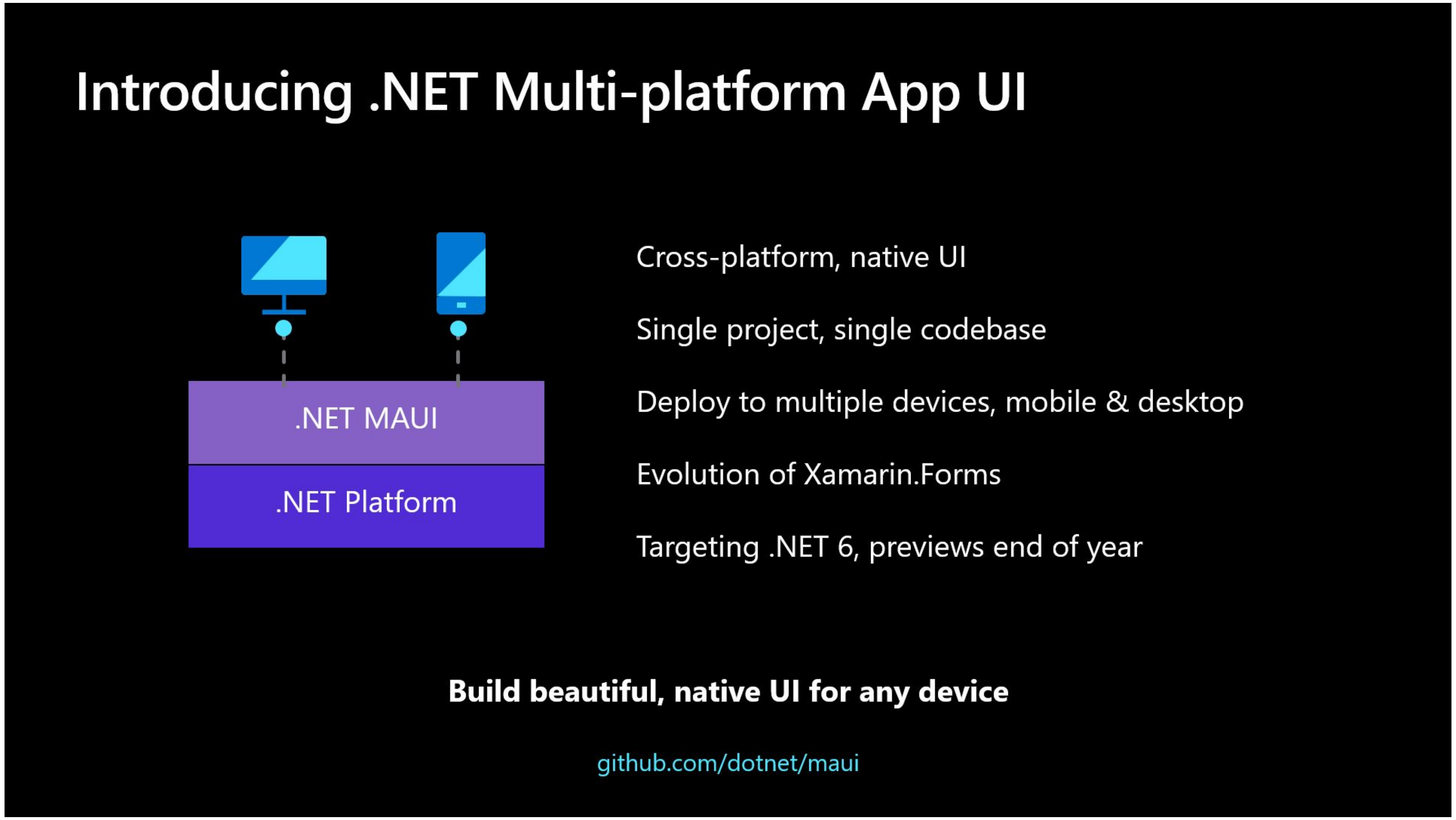
```
partial class A
{
    public partial string Test(out string s);
}
partial class A
{
    public partial string Test(out string s) => s = "Hello, world";
}

[System.Runtime.CompilerServices.ModuleInitializer]
internal static void Init()
{
    var init = $"Module initializer for {Assembly.GetExecutingAssembly().GetName()}";
}
```

Demo

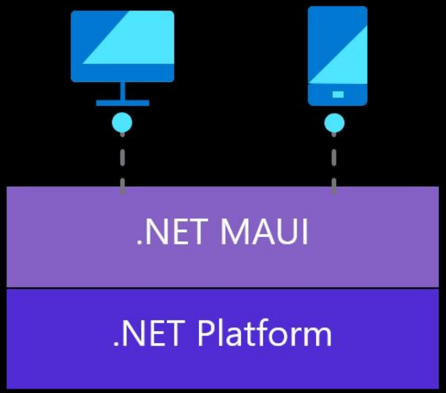
Features postponed to .NET 6

- Blazor AOT (ahead-of-time compilation)
- Windows Presentation Foundation (WPF)
- Windows Forms
- Xamarin mobile

.NET 6



Introducing .NET Multi-platform App UI



Cross-platform, native UI

Single project, single codebase

Deploy to multiple devices, mobile & desktop

Evolution of Xamarin.Forms

Targeting .NET 6, previews end of year

Build beautiful, native UI for any device

github.com/dotnet/maui

Q&A

References

- What's new in .NET 5
- Announcing .NET 5.0
- .NET 5 Arrives
- What's new in C# 9.0

Thank you

You are great ;)