Demystifying the most significant C# language features from 8.0 to 9.0 and beyond

Dr. Milan Milanović
Lead Software Architect, Zühlke Group

https://milan.milanovic.org

C# history

C# language history

- Developed around 2000. by Microsoft.
- Anders Hejlsberg.
- Approved as an international standard by:
  - Ecma (ECMA-334) in 2002.
  - ISO (ISO/IEC 23270) in 2003.

C# language version history

C# 1.0

Managed

C# 2.0

Generics

C# 3.0

LINQ

C# 4.0

Dynamic programming

C# 5.0

Asynchronous programming

C# 6.0

Roslyn

C# 7.0

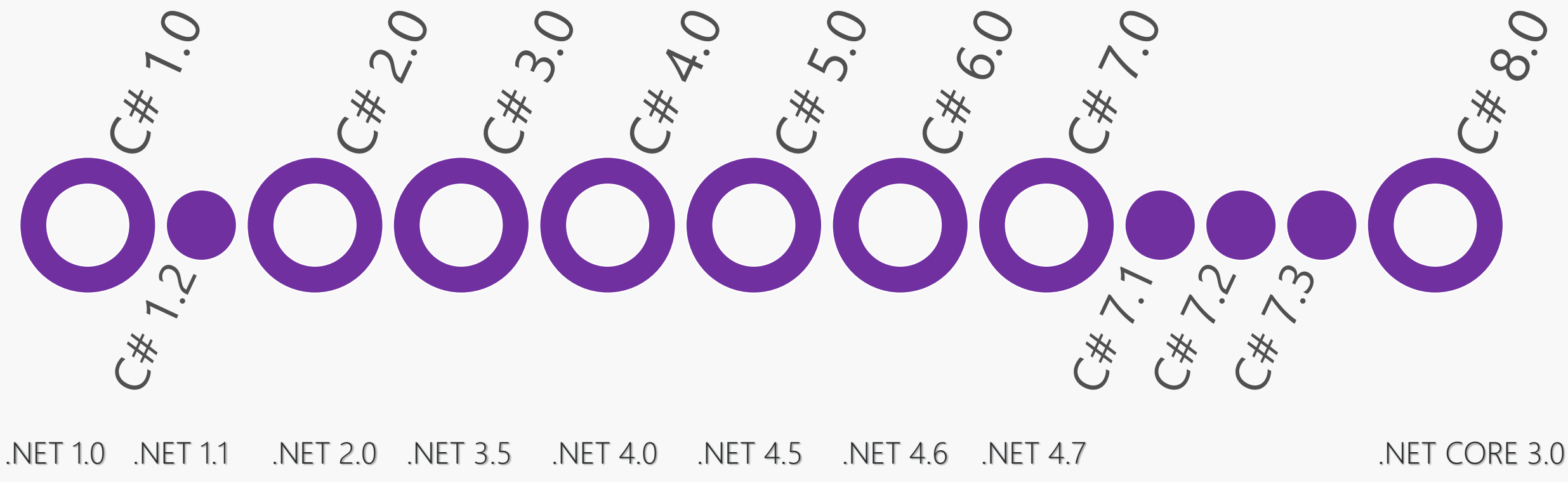
Tuples

C# 8.0

Nullable ref. types

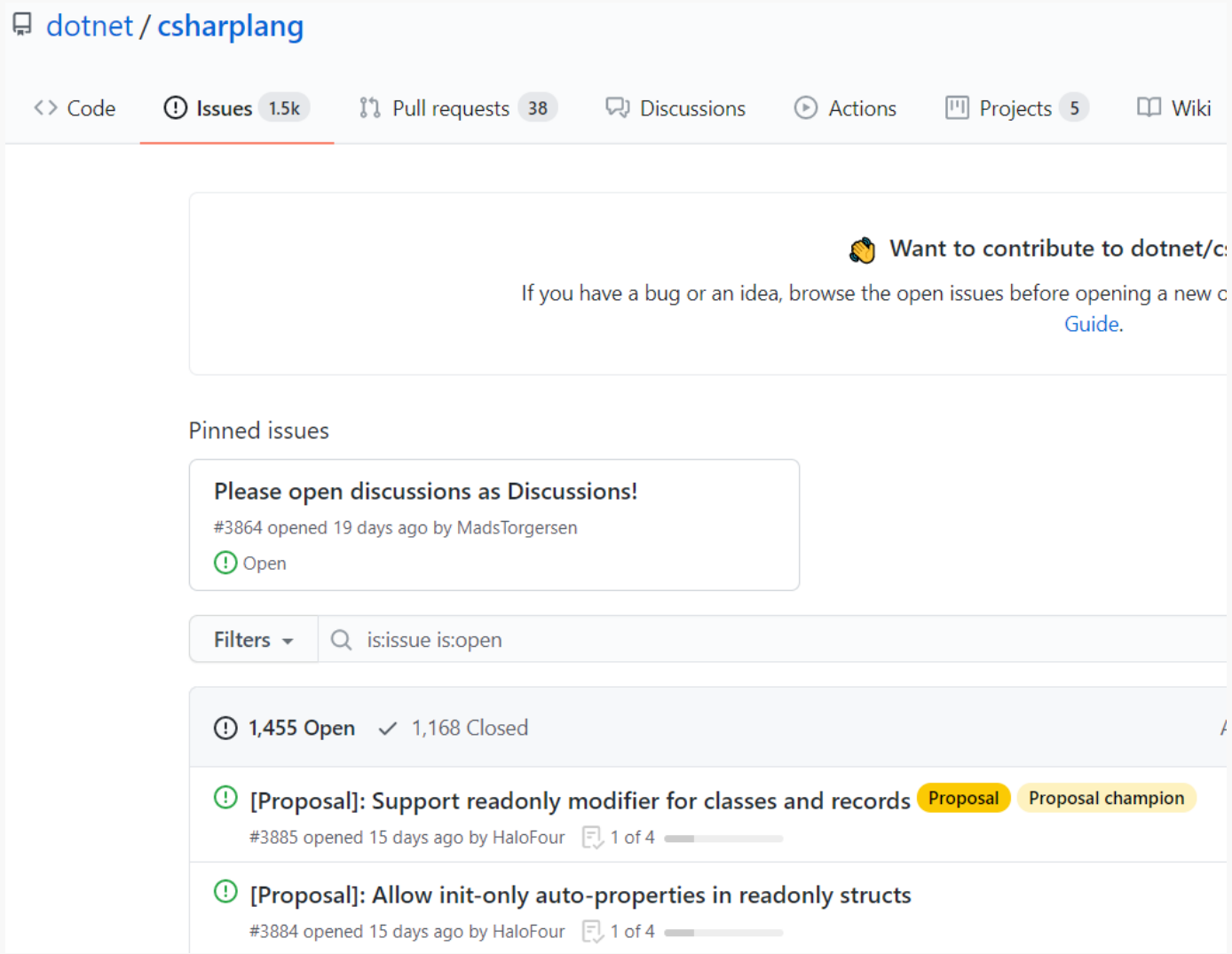
.NET runtimes

.NET runtimes



C# Language

C# Language Design



https://github.com/dotnet/csharplang

C# 8.0 features

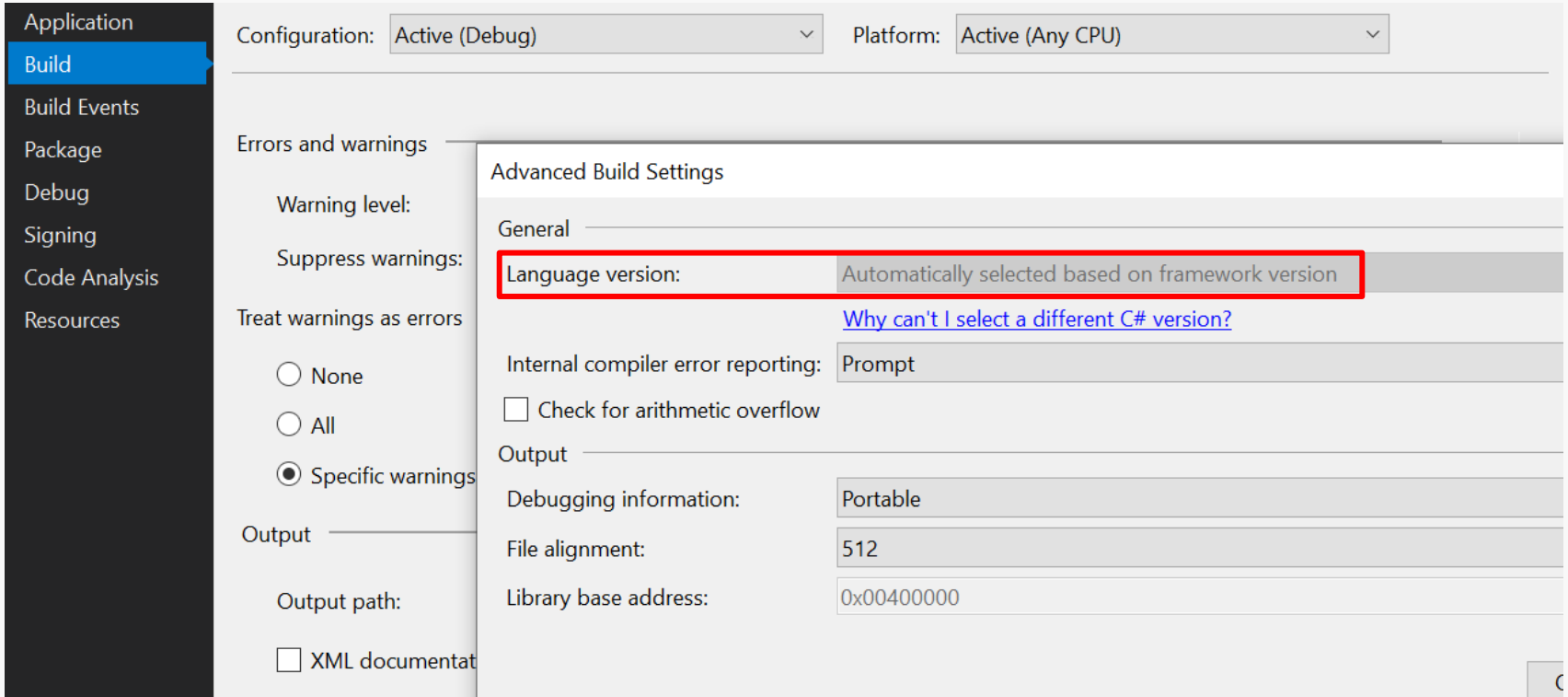
Types of Language Features

- Syntax only
- Requires Types
- Requires Runtime Support

```
1   error CS8701: Target runtime doesn't support default interface implementation.
```

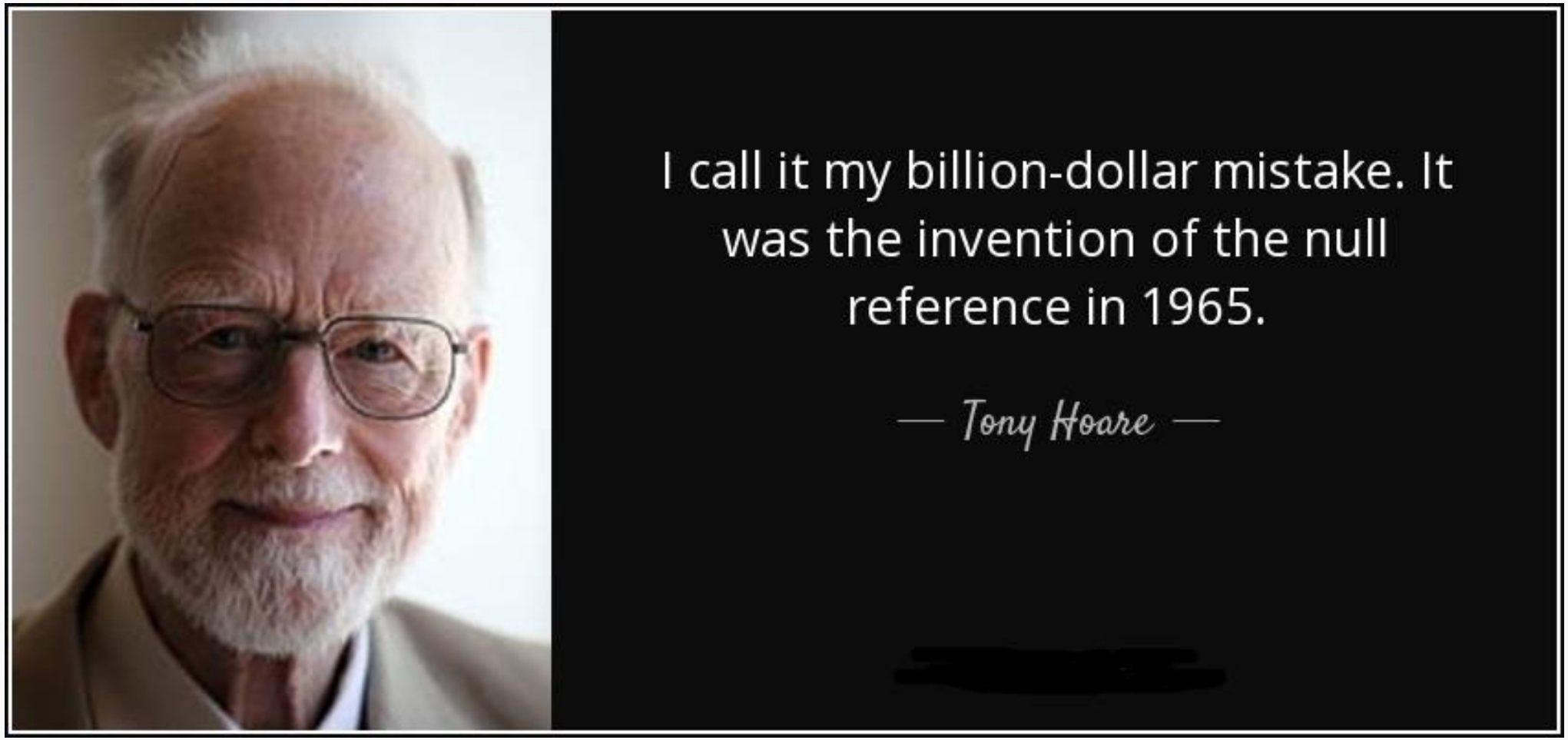
How to enable C# 8 in Visual Studio 2019

- C# compiler determines a default language version.
- You can change it by editing the *csproj* file.
- C# 8.0 is supported only from .NET Core 3.x.



NullReferenceException

- The billion-dollar mistake.



Nullable Reference types

- Explicitly tell the compiler to check for null values.
- Switching from **`null`** to **`not null`** for all types.
- In .csproj:

```
<LangVersion>8.0</LangVersion>
<Nullable>Enable</Nullable>
```

- Or in code:

```
#nullable enable
#nullable disable
#nullable restore
```

Nullable Reference types

- New operators: **?** And **!**

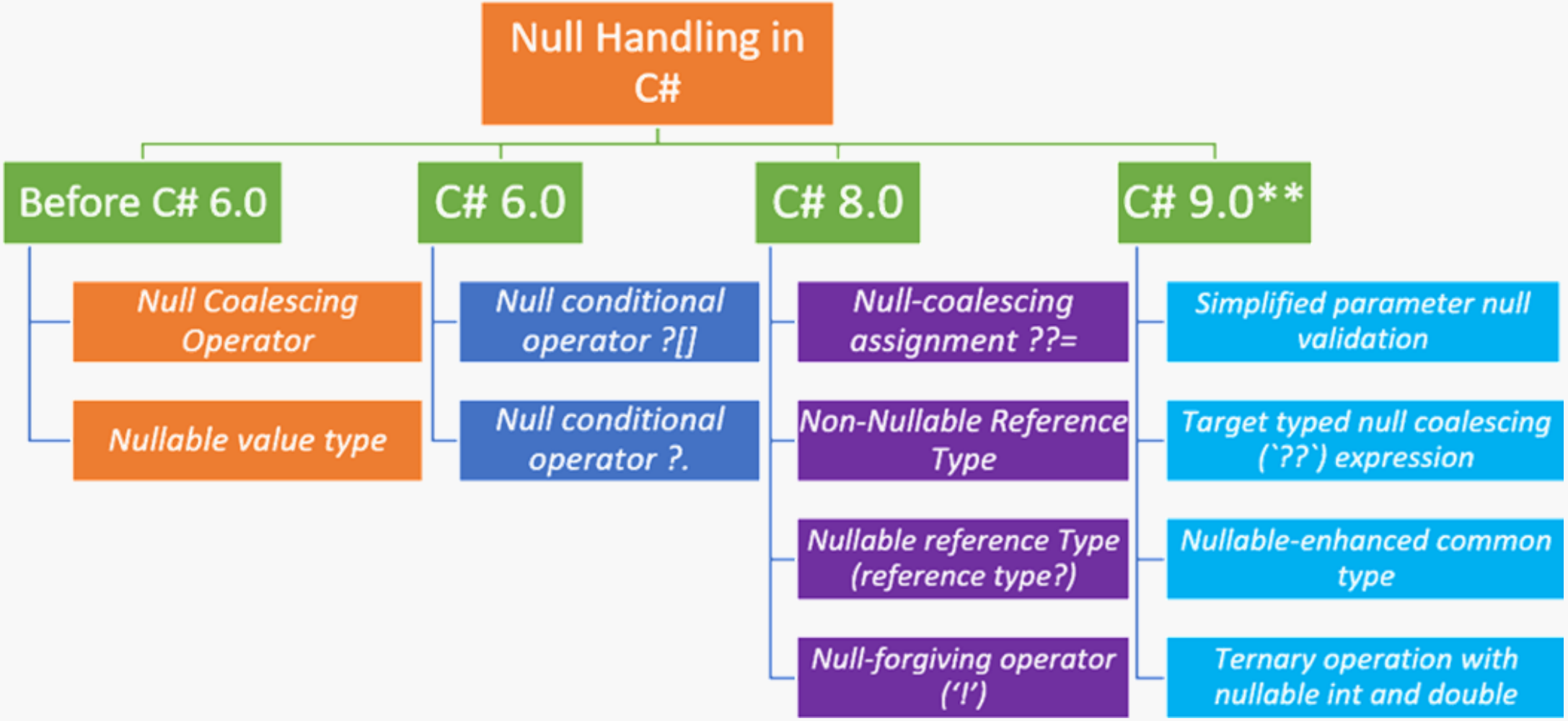
```
1 // this will make sure we can assign a null value to that variable
2 string? a = null;
3
4 // this will remove the warning and let us try to call `.Length`
5 var l = a!.Length;
```

Null coalescing operator

- **??=** is a helpful little operator that allows you to assign a default value in case of a null.

```
1 // Before
2 var possibleNullValue = possibleNullValue ?? "default value";
3
4 // Now this is also allowed
5 var possibleNullValue ??= "default value";
```

Null Value And Null Reference Handling



Switch Expression with pattern matching

- Matching on types:

```
1  public decimal CalculateToll(object vehicle)
2  {
3      return vehicle switch
4      {
5          Car c ⇒ 2.00m,
6          Taxi t ⇒ 3.50m,
7          Bus b ⇒ 5.00m,
8          DeliveryTruck t ⇒ 10.00m,
9          { } ⇒ throw new ArgumentException("Not a known vehicle type", nameof(vehicle)),
10         null ⇒ throw new ArgumentNullException(nameof(vehicle))
11     };
12 }
```

Switch Expression with pattern matching

- Matching on properties

```
public decimal CalculateToll(object vehicle)
{
    return vehicle switch
    {
        Car { Passengers: 0 } ⇒ 2.00m + 0.50m,
        Car { Passengers: 1 } ⇒ 2.0m,
        Car { Passengers: 2 } ⇒ 2.0m - 0.50m,
        Car _ ⇒ 2.00m - 1.0m,

        Taxi { Fares: 0 } ⇒ 3.50m + 1.00m,
        Taxi { Fares: 1 } ⇒ 3.50m,
        Taxi { Fares: 2 } ⇒ 3.50m - 0.50m,
        Taxi _ ⇒ 3.50m - 1.00m,

        Bus b ⇒ 5.00m,
        DeliveryTruck t ⇒ 10.00m,
        { } ⇒ throw new ArgumentException("Not a known vehicle type", nameof(vehicle)),
        null ⇒ throw new ArgumentNullException(nameof(vehicle))
    };
}
```

Switch Expression with pattern matching

- Conditional boolean expression

```
1  public decimal CalculateToll(object vehicle)
2  {
3      return vehicle switch
4      {
5          // car and taxi hidden here to make it easier to read
6          //  ...
7          Bus b when (double)b.Riders / (double)b.Capacity < 0.50 ⇒ 5.00m + 2.00m,
8          Bus b when (double)b.Riders / (double)b.Capacity > 0.90 ⇒ 5.00m - 1.00m,
9          Bus _ ⇒ 5.00m,
10
11         DeliveryTruck t ⇒ 10.00m,
12         { } ⇒ throw new ArgumentException("Not a known vehicle type", nameof(vehicle)),
13         null ⇒ throw new ArgumentNullException(nameof(vehicle))
14     };
15 }
```

Switch Expression with pattern matching

- Nesting

```
public decimal CalculateToll(object vehicle)
{
    return vehicle switch
    {
        Car c ⇒ c.Passengers switch
        {
            0 ⇒ 2.00m + 0.5m,
            1 ⇒ 2.0m,
            2 ⇒ 2.0m - 0.5m,
            _ ⇒ 2.00m - 1.0m
        },

        Taxi t ⇒ t.Fares switch
        {
            0 ⇒ 3.50m + 1.00m,
            1 ⇒ 3.50m,
            2 ⇒ 3.50m - 0.50m,
            _ ⇒ 3.50m - 1.00m
        },
        // ...
    };
}
```

Async Streams

- New way to iterate over a stream of data.
- To enable asynchronous streams:
  - Method must be `async`.
  - The return type of method must be `IAsyncEnumerable<T>`.
  - The method body must contain at least one `yield` return.

Async Streams

- An example: count all users names that contain **n** letter.

```
1  public static async IAsyncEnumerable<string> GetAllNames()
2  {
3      var pageIndex = 0;
4      const int pageSize = 100;
5      var hasMore = false;
6      do
7      {
8          await using var conn = new SqlConnection("ConnectionString here");
9          await using var cmd = new SqlCommand(
10             @$"
11             SELECT Name
12             FROM Users
13             ORDER BY Name
14             OFFSET {pageIndex * pageSize} ROWS
15             FETCH NEXT {pageSize} ROWS ONLY",
16             conn);
17         await using var reader = await cmd.ExecuteReaderAsync();
18         while (reader.Read())
19         {
20             yield return reader.GetString(0); // This is the "Name" column
21         }
22
23         pageIndex++;
24         hasMore = reader.HasRows;
25     } while (hasMore);
26 }
```

```
1  public async Task<int> CountNamesWithN()
2  {
3      var namesContainingN = 0;
4      await foreach (var name in GetAllNames())
5      {
6          if (name.Contains("n"))
7          {
8              namesContainingN++;
9          }
10     }
11
12     return namesContainingN;
13 }
```

Consumer

Indices and ranges

- Manipulating large set of data in multiple arrays.
- New operators: ^ and . .

```
private string[] words = new string[]
{
                // index from start    index from end
    "The",      // 0                   ^4
    "quick",    // 1                   ^3
    "brown",    // 2                   ^2
    "fox"
};
```

```
Index the = ^3;
words[the];

Range phrase = 1..4;
words[phrase];
```

```
var allWords = words[..]; // contains "The" through "fox".
var firstPhrase = words[..4]; // contains "The" through "fox".
var lastPhrase = words[2..]; // contains "brown" and "fox".
var lazyDog = words[^2..^0]; // contains "brown" and "fox".
```

Default interface methods

- Add new methods to an interface without breaking existing implementations.

```
1  interface IOutput
2  {
3      sealed void PrintException(Exception exception)
4          ⇒ PrintMessageCore($"Exception: {exception}");
5
6      protected void PrintMessageCore(string message);
7
8      protected static void PrintToConsole(string message)
9          ⇒ Console.WriteLine(message);
10 }
11
12 class ConsoleOutput : IOutput
13 {
14     void IOutput.PrintMessageCore(string message)
15     {
16         IOutput.PrintToConsole(message);
17     }
18 }
```

Improving productivity with C# 8.0

- Using variables

```
1  // Before
2  using (var stream = new FileStream("", FileMode.Open))
3  {
4      using (var sr = new StreamReader(stream))
5      {
6          ...
7      }
8  }
9
10 // Now this is also allowed
11 using var stream = new FileStream("", FileMode.Open);
12 using var sr = new StreamReader(stream);
```

Improving productivity with C# 8.0

- Verbatim string with interpolation **$@ @$**

```
1 // Before
2 $@"This → {nameof(this)}";
3
4 // Now this is also allowed
5 @$"This → {nameof(this)}";
```

Additional features

- Recursive patterns.
- Unmanaged generic structs.
- Static Local Functions.
- Disposable ref Structs.
- Readonly members.
- Many more.

Usage of C# 8.0 with .NET Framework

- What will work:
  - Static local functions.
  - Using declarations.
  - Null-coalescing assignment.
  - Readonly members.
  - Disposable ref structs.
  - Positional patterns.
  - Tuple patterns.
  - Switch expressions.
- What will not work: Default interface members.
- What could work: Async streams and indices and ranges.

C# 9.0 features

.NET

C# 9.0 (still in preview)

- Expected to be released in November of 2020 along with .Net 5.
- Change **`<LangVersion>`** to preview in .csproj.
- Update project **`targetFramewrok`** to net5.0.

```
1 <Project>
2  <PropertyGroup>
3    <LangVersion>preview</LangVersion>
4    <TargetFramework>net5.0</TargetFramework>
5  </PropertyGroup>
6 </Project>
```

C# 9.0 (still in preview)

- In order to use it, the right version of Visual Studio and .NET 5.0 is needed.
- Check installed versions: dotnet --info.
- Enable Preview Feature: *Use previews of .NET Core SDK*.
- The latest version: SDK 5.0.100-rc.1.
- Requires Visual Studio 2019 (v16.8, Preview 3).

Top-level statements

- Remove unnecessary ceremony from many apps.

```
1 using System;
2
3 namespace HelloWorld
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Console.WriteLine("Hello World!");
10        }
11    }
12 }
```

Before

```
1 System.Console.WriteLine("Hello World!");
```

Now

Top-level statements

```
using System;
using System.Runtime.CompilerServices;

// Token: 0x02000002 RID: 2
[CompilerGenerated]
internal static class <Program>$
{
    // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
    private static void <Main>$(string[] args)
    {
        Console.WriteLine("Hello World!!");
    }
}
```

Record types

- Bridge the gap between **`class`** and **`struct`** types.
- Records are intended to be **immutable**.

```
1 public record Order
2 {
3     public int Id { get; init set; }
4     public string Status { get; init set; }
5     public bool IsPaid { get; init set; }
6 }
```

Record types

- Updating records:

```
1 var updatedOrder = order with { Status = "Delivered" };
```

- Usage:
  - DTOs
  - Events.
  - ...

Init-only Properties

- Wow do we instantiate immutable types, like records?

```
1 public record Order
2 {
3     public int Id { get; init set; }
4     public string Status { get; init set; }
5     public bool IsPaid { get; init set; }
6 }
```

Declaration

```
1 var order = new Order
2 {
3     Id = 100,
4     Status = "Created",
5     IsPaid = false
6 };
```

Initialization

Improved Pattern Matching

```
1 static int CalculateTicketPrice(Person person)
2 {
3     return person switch
4     {
5         var p when p.Age ⩽ 12 ⇒ 5,
6         var p when p.Age > 12 && p.Age ⩽ 60 ⇒ 15,
7         var p when p.Age > 60 ⇒ 10
8     };
9 }
```

C# 8.0

```
1 static int CalculateTicketPrice(Person person)
2 {
3     return person when person.Age switch
4     {
5         ⩽ 12 ⇒ 5,
6         > 12 and ⩽ 60 ⇒ 15,
7         > 60 ⇒ 10
8     };
9 }
```

C# 9.0

```
1 foreach (var v in vectors)
2 {
3     if (v is { X: 1 })
4     {
5         Console.WriteLine(v);
6     }
7 }
```

Target Typing Improvements

- The concept that allows the compiler to infer the type, for instance, when assigning **`null`**:

```
1 string s = null;
```

- Target Typed **`new`** expressions:

```
1  Vector3 vec = new(1, 2, 3);
```

Target Typing Improvements

- Target Typing and Shared Types

```
1 public IEnumerable<int> GetNumbers(bool even)
2 {
3     return even ? new List<int> {2, 4, 6} : new []{1, 3, 5};
4 }
```

Additional features

- Covariant returns.
- Native sized integers.
- Function pointers.
- Static lambdas.
- Lambda discard parameters.
- Many more.

https://github.com/dotnet/csharplang

C# Next

C# Next

- Work on the next version already started (C# 10.0).
- More with records, initialization and immutability.
- Factories.
- Final initializers.
- More details:

https://github.com/dotnet/roslyn/blob/master/docs/Language%20Feature%20Status.md

Thanks!
Questions?

https://milan.milanovic.org