

What's new in .NET 8 & C# 12

Filip Ekberg

Hello,

I'm Filip Ekberg

fili@ekberg.dev

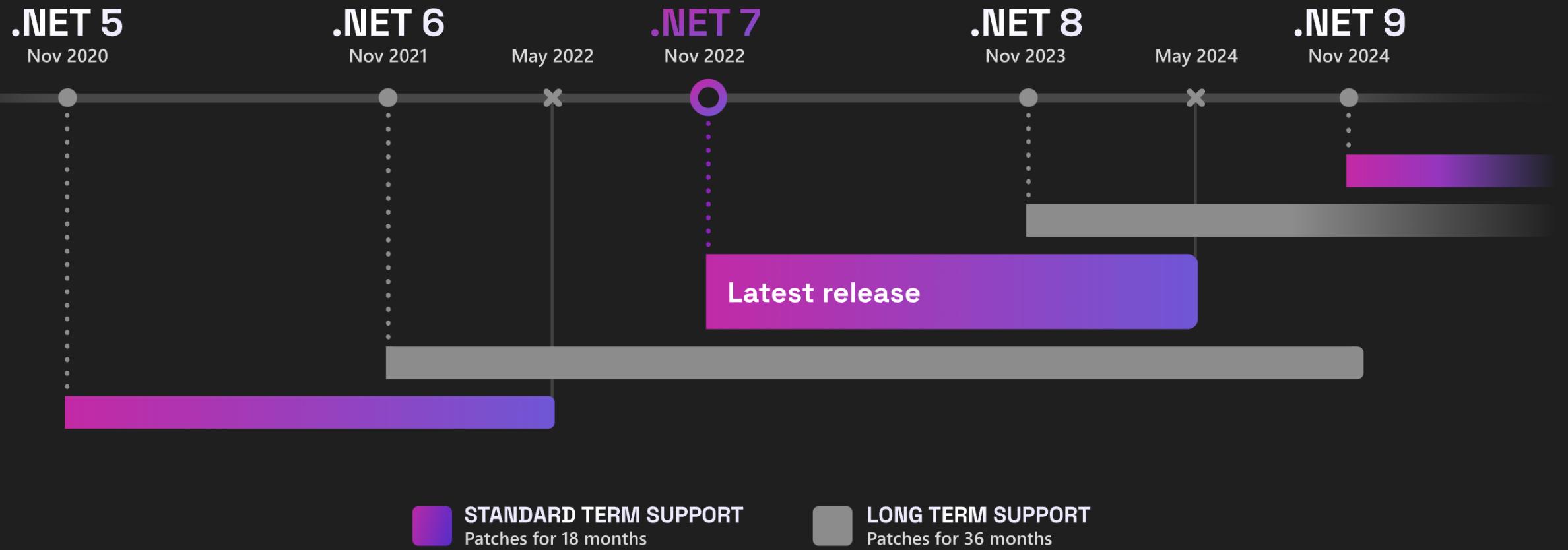
 **C# smorgasbord** free @ filipekberg.se



Using Statements for Static Members
Implicitly typed local variables
Extension methods
Getter & Setter separate accessibility
Null-conditional operators
Exception Filters
Anonymous methods
Named and optional parameters
Lambdas
Static classes
Auto-properties
String interpolation
Stackalloc initializers
Caller info attributes
Nullable types
Stackalloc with Span
Dynamic binding
Digit Separators
private protected
Default Expressions
digit separators
Tuples and Deconstruction
Ref Assemblies
Expression trees
Local Functions
Generics
non-trailing named arguments
out improvements
Dictionary Initializers
Infer Tuple Names
Query expressions
Partial types
Iterators
ref returns
Improved generic constraints
Await inside Catch & Finally blocks
nameof operator
Generic co- and contravariance
Asynchronous programming
Async & Await
Infer Tuple Names
Partial types
Pattern Matching
Object and collection initializers
conditional ref operator
Binary Literals
Expression-bodied members
Auto-Property Initializers
Anonymous types
ref readonly & in parameter
Main
Tuple equality

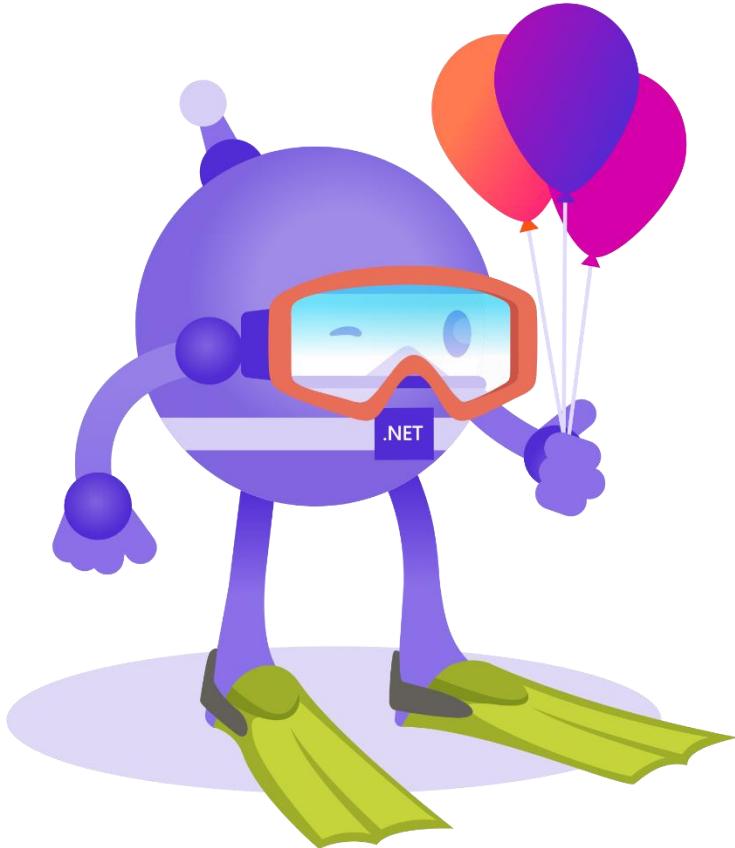
.NET 8



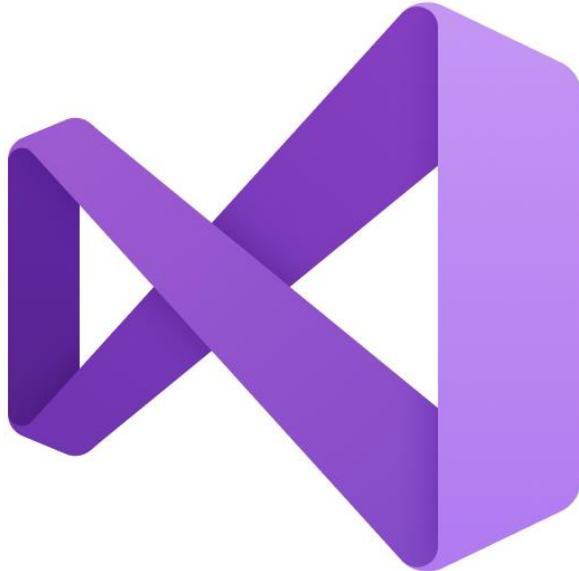


“Build anything”

.NET 8



- Performance, performance, performance!
- Time abstraction
- System.Text.Json
- ASP.NET Core
- .NET MAUI
- Blazor



.NET + C#

- .NET Core 3.0 for C# 8
- .NET 5.0 for C# 9
- .NET 6.0 for C# 10
- .NET 7.0 for C# 11
- .NET 8.0 for C# 12



BEFORE



AFTER

Spans + Ranges = Performance

Range

“Ranges and indices provide a succinct syntax for accessing single elements or ranges in a sequence.”

[2..] [..^1] [^2..]

Zero Allocations



```
byte[] payload = [0x1, 0xf1, 0xaa, 0xf2];
```

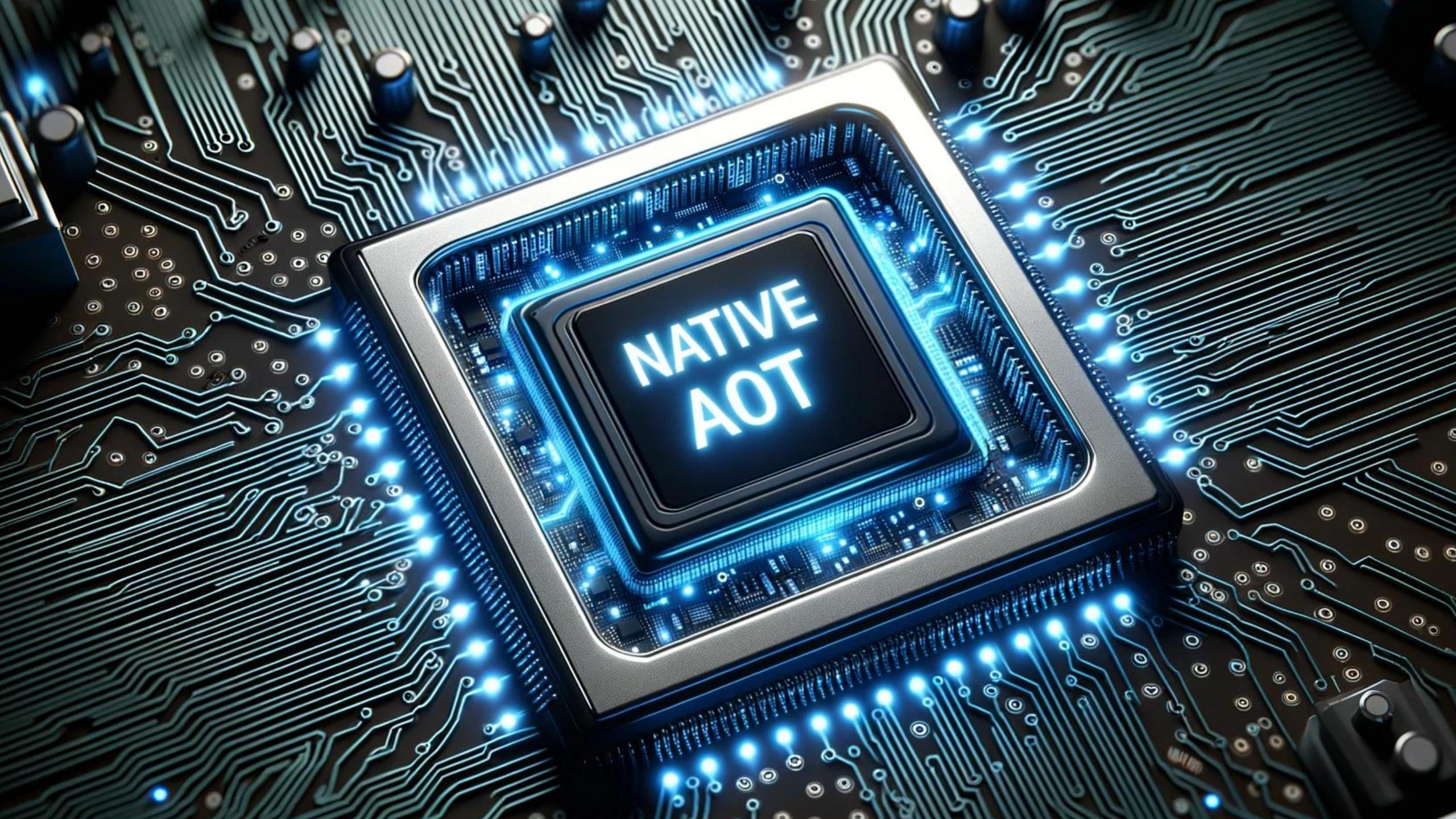
```
Span<byte> payloadSpan = payload;
```

```
Span<byte> subset = payloadSpan[^2..];
```

Zero Allocations



```
ReadOnlySpan<byte> payload = "Filip Ekberg"u8;  
  
int space = payload.IndexOf(" ")u8);  
  
ReadOnlySpan<byte> first = payload[..  
space];  
  
ReadOnlySpan<byte> second = payload[  
++space..  
];
```



NATIVE
AOT

Performance-focused types



```
System.Collections.Frozen.  
FrozenDictionary.ToFrozenDictionary(source);
```

Serialization

Time abstraction

How'd you test this?



```
class OrderService
{
    public bool HasPaymentExpired(Order order)
    {
        return (DateTimeOffset.UtcNow - order.PaymentReservedOn).TotalDays > 30;
    }
}
```

Randomness



```
var randomItems = Random.Shared.GetItems([1, 2, 3, 4, 5, 6, 7], 3);
```

ASP.NET Core

Short circuit routes



```
app.MapGet("/short-circuit", () => "I'm executed immediately!")
    .ShortCircuit();
```

Keyed Services



```
builder.Services.AddKeyedTransient<ICache, DistributedCache>("distributed");
builder.Services.AddKeyedTransient<ICache, InMemoryCache>("memory");

[FromKeyedServices("memory")] ICache cache
```

Complex Binding in Minimal APIs



```
app.MapPost("/upload", async (
    IFormFile data,
    HttpContext context,
    IAntiforgery antiforgery) =>
{
    await antiforgery.ValidateRequestAsync(context);
});
```

Blazor

“With the release of .NET 8, Blazor is a full-stack web UI framework for developing apps that render content at either the component or page”

.NET MAUI

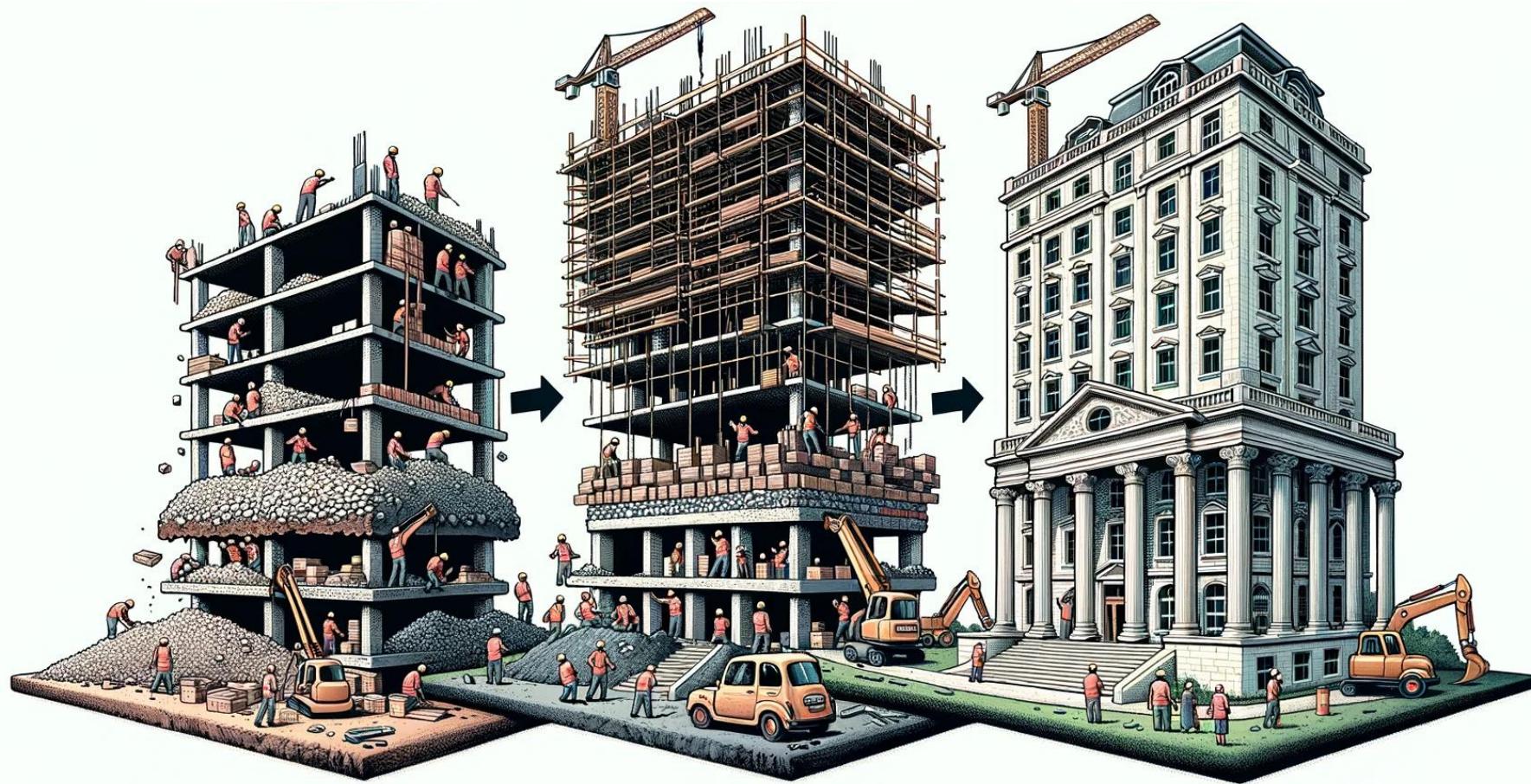


.NET MAUI

.NET 6

.NET 7

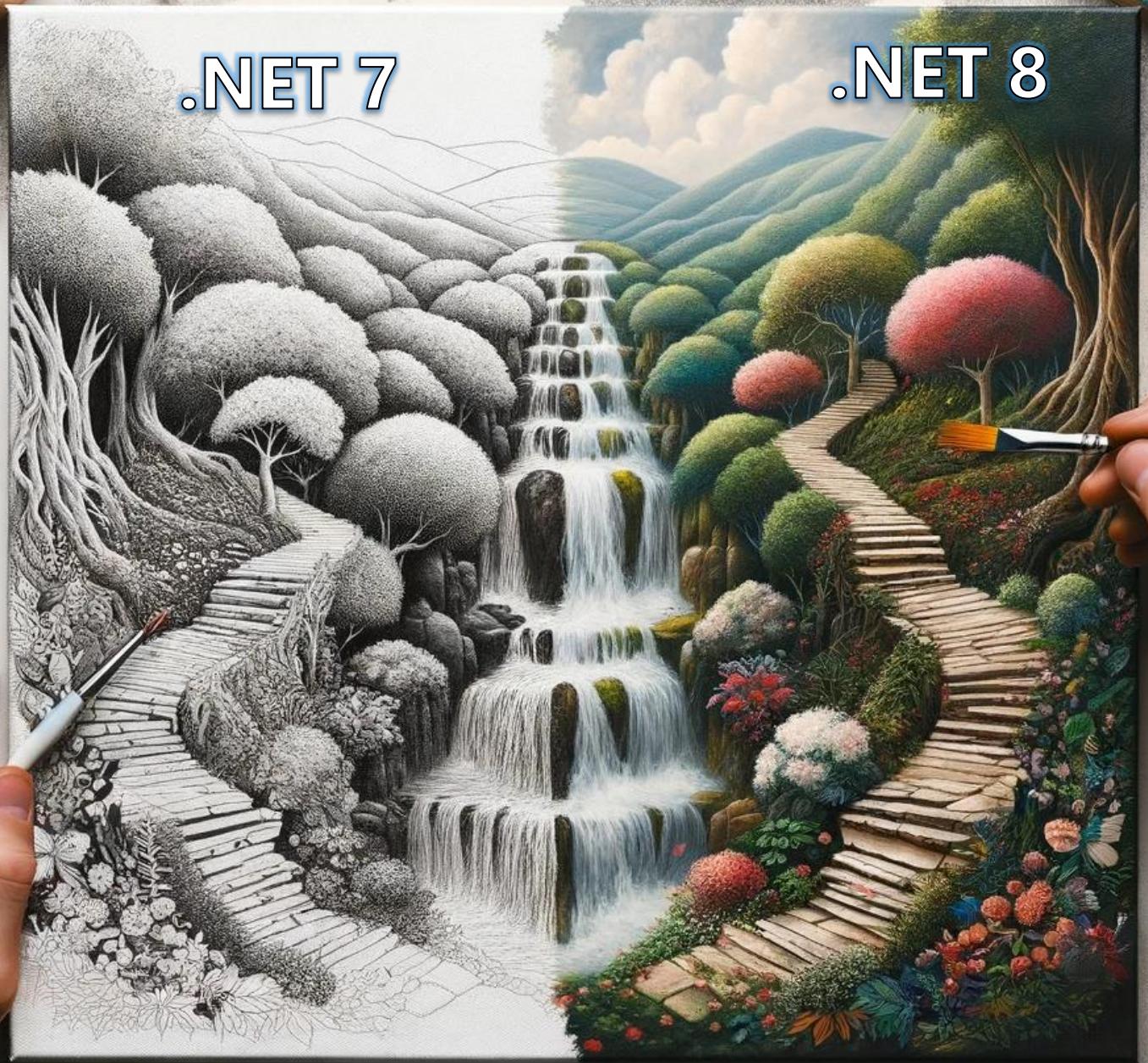
.NET 8

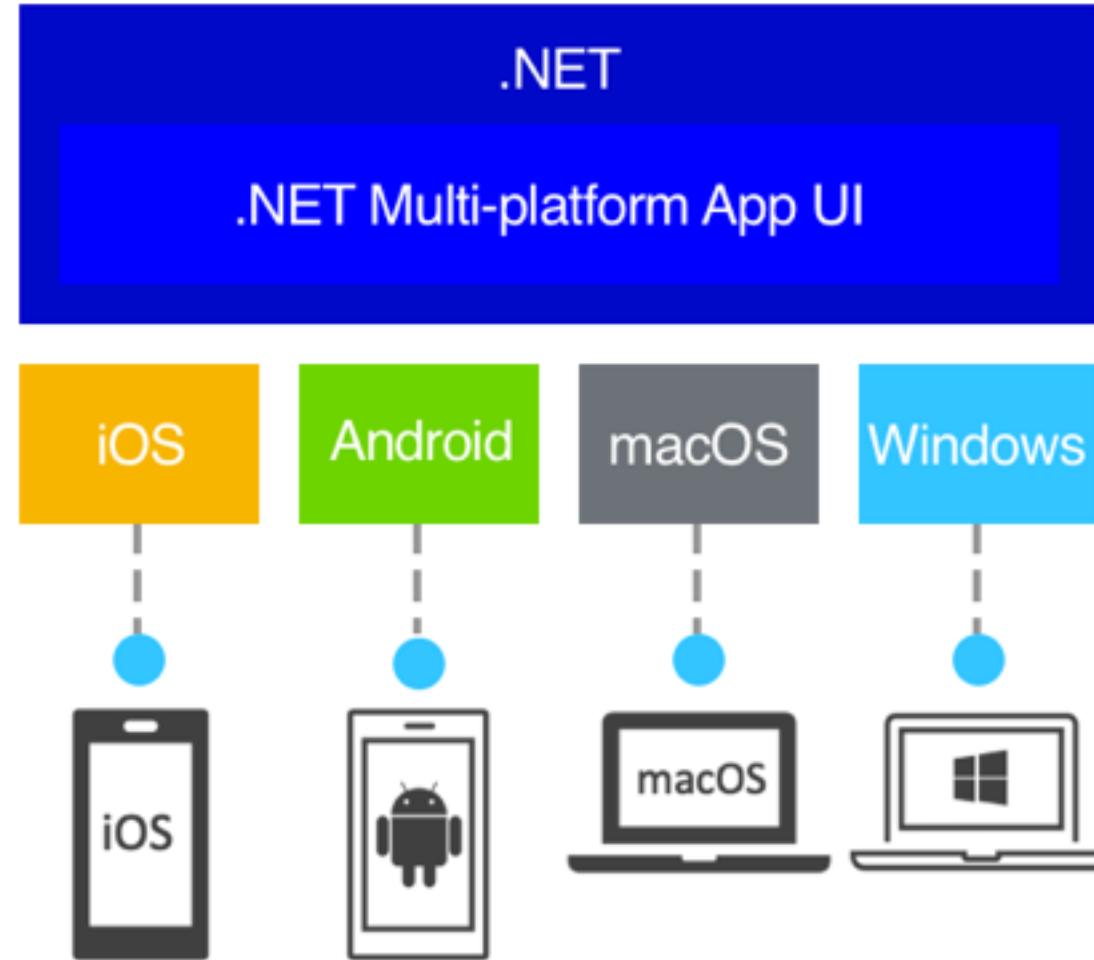


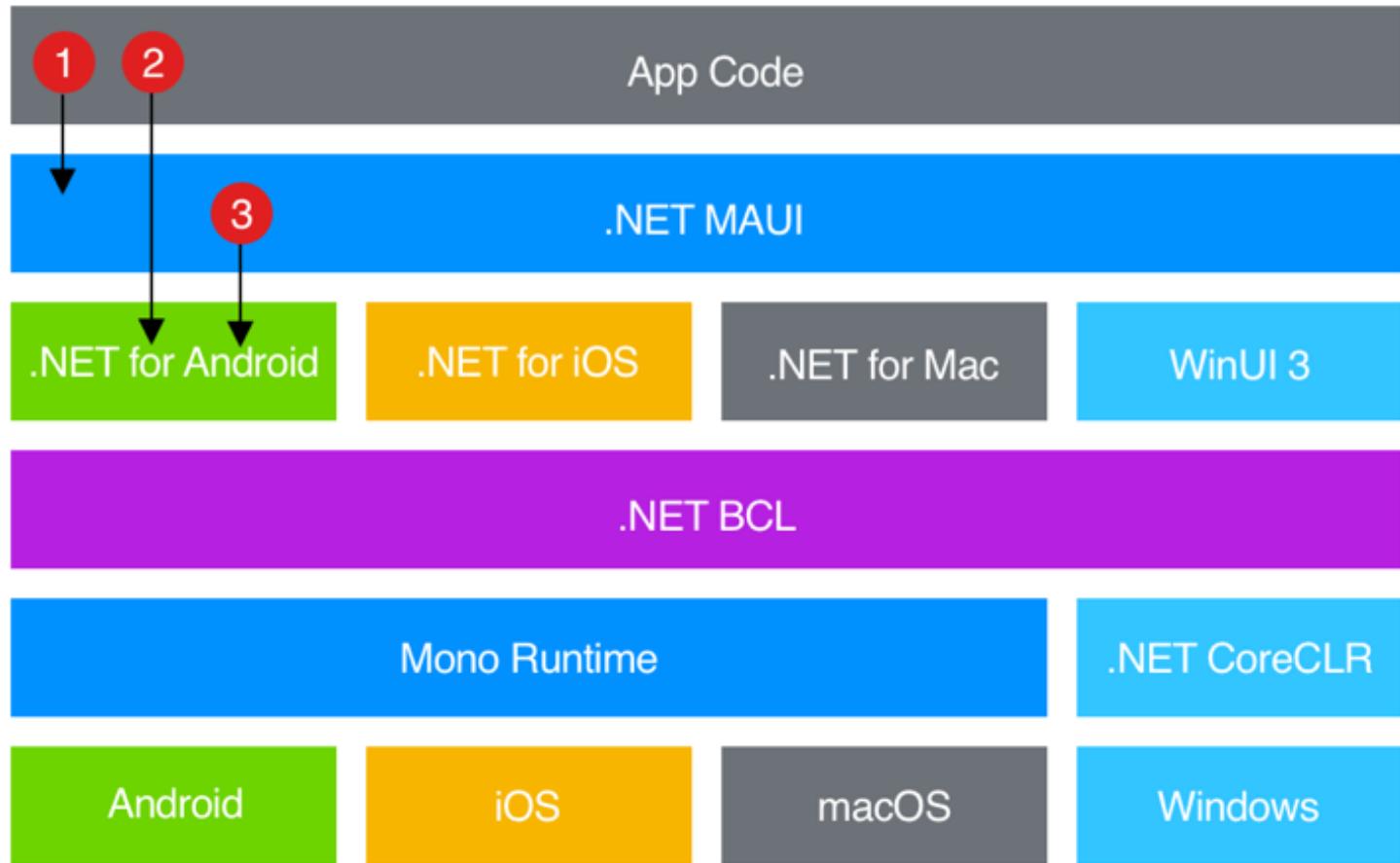
.NET MAUI

.NET 7

.NET 8







Using Statements for Static Members
Implicitly typed local variables
Extension methods
Getter & Setter separate accessibility
Null-conditional operators
Exception Filters
Anonymous methods
Named and optional parameters
Lambdas
Static classes
Auto-properties
String interpolation
stackalloc initializers
Caller info attributes
Nullable types
private protected
stackalloc with Span
Dynamic binding
Digit Separators
Default Expressions
digit separators
non-trailing named arguments
out improvements
Dictionary Initializers
Tuples and Deconstruction
Generics
Default Expressions
Expression trees
Local Functions
Ref Assemblies
Asynchronous programming
Infer Tuple Names
Query expressions
Partial types
Iterators
ref returns
Improved generic constraints
nameof operator
Generic co- and contravariance
Await inside Catch & Finally blocks
Pattern Matching
Object and collection initializers
conditional ref operator
Async Main
Tuple equality
Binary Literals
Expression-bodied members
Auto-Property Initializers
Anonymous types
ref readonly & in parameter

C# 12

C# 12

Primary Constructors

Collection expressions
& spread operator

Interceptors

Optional parameters
in lambdas

Alias any type

Inline arrays

ref readonly parameters

Experimental attribute

Collection expressions



```
Span<byte> payload = [0x1, 0xf1, 0xaa, 0xf2];
```

```
Span<byte> checksum = [0xff, 0xab];
```

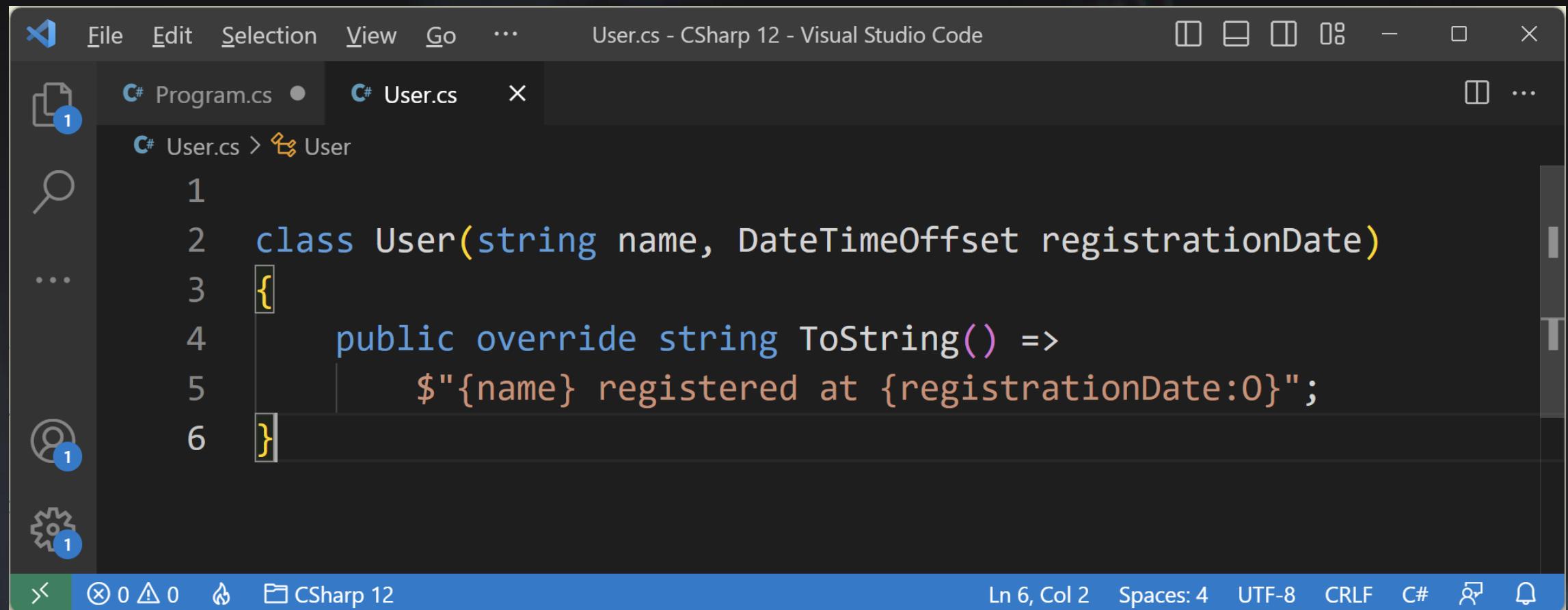
```
Span<byte> result = [..payload, ..checksum];
```

Primary constructors



```
class OrderController(IUserRepository repository)
{
    public (Guid, decimal) GetTotalFor(Guid orderId)
    {
        return (orderId, repository.Sum(orderId));
    }
}
```

Primary constructors

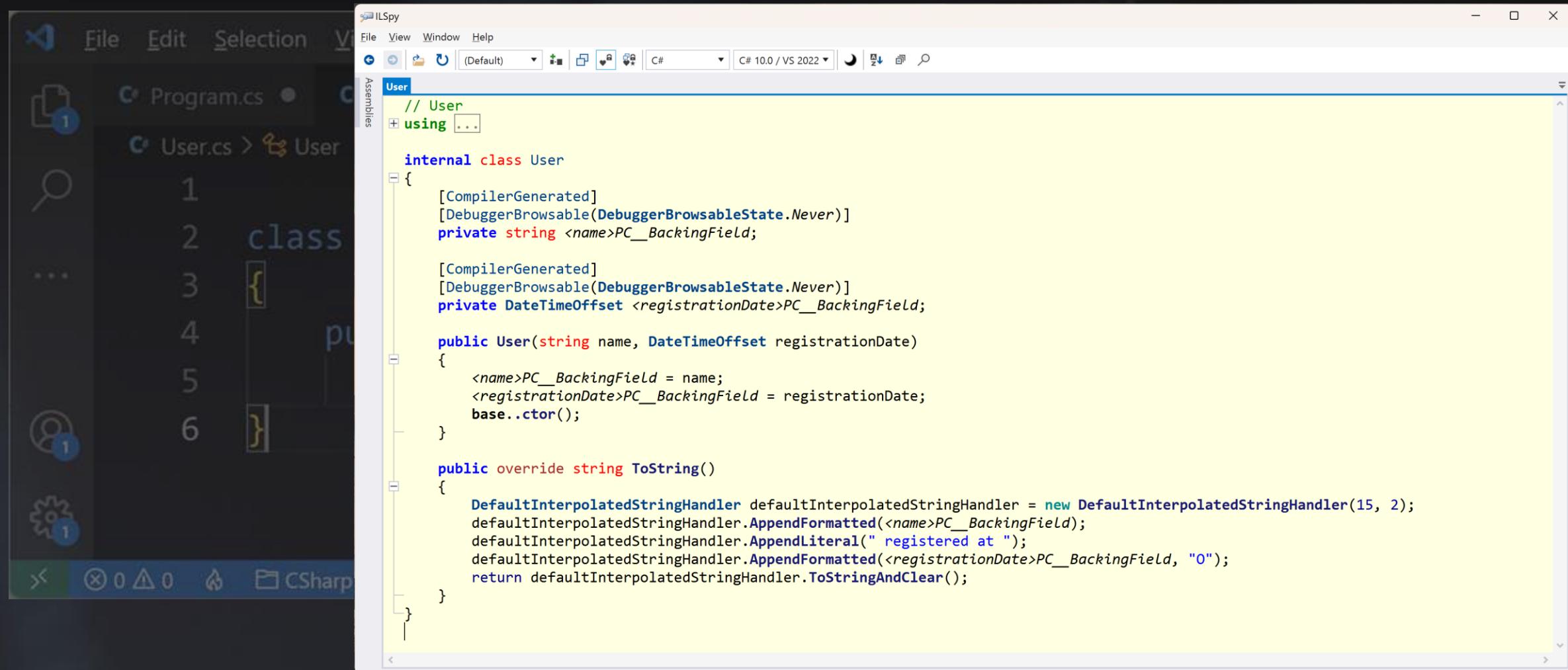


A screenshot of the Visual Studio Code interface showing a C# file named `User.cs`. The code defines a `User` class with a primary constructor taking `name` and `registrationDate` parameters. It also includes a `ToString` method.

```
1
2 class User(string name, DateTimeOffset registrationDate)
3 {
4     public override string ToString() =>
5         $"{name} registered at {registrationDate:0}";
6 }
```

The code editor shows syntax highlighting for C# keywords and types. The status bar at the bottom indicates the file is saved (0 changes), the code page is UTF-8, and the encoding is CRLF. The bottom right corner shows a small icon with a number 1, likely indicating a pending update or notification.

Primary constructors



The screenshot shows the ILSpy decompiler interface. The assembly tree on the left shows a single assembly containing a class named `User`. The decompiled code for the `User` class is displayed in the main window:

```
// User
using ...;

internal class User
{
    [CompilerGenerated]
    [DebuggerBrowsable(DebuggerBrowsableState.Never)]
    private string <name>PC__BackingField;

    [CompilerGenerated]
    [DebuggerBrowsable(DebuggerBrowsableState.Never)]
    private DateTimeOffset <registrationDate>PC__BackingField;

    public User(string name, DateTimeOffset registrationDate)
    {
        <name>PC__BackingField = name;
        <registrationDate>PC__BackingField = registrationDate;
        base..ctor();
    }

    public override string ToString()
    {
        DefaultInterpolatedStringHandler defaultInterpolatedStringHandler = new DefaultInterpolatedStringHandler(15, 2);
        defaultInterpolatedStringHandler.AppendFormatted(<name>PC__BackingField);
        defaultInterpolatedStringHandler.AppendLiteral(" registered at ");
        defaultInterpolatedStringHandler.AppendFormatted(<registrationDate>PC__BackingField, "0");
        return defaultInterpolatedStringHandler.ToStringAndClear();
    }
}
```

The code illustrates the use of primary constructors in C#. The constructor takes two parameters: `name` and `registrationDate`. It initializes the corresponding backing fields (`<name>PC__BackingField` and `<registrationDate>PC__BackingField`) and calls the base class's constructor (`base..ctor()`). The `ToString` method uses a `DefaultInterpolatedStringHandler` to format the object's state.

Optional parameters in lambda expressions

A.K.A. Default lambda parameters

```
var process = (string name = "Default") => {  
    Console.WriteLine($"Processing: {name}");  
};
```

Optional parameters in lambda expressions

A.K.A. Default lambda parameters

```
var process = (string name = "Default") => {  
    Console.WriteLine($"Processing: {name}");  
};
```

```
process("Filip Ekberg");
```

```
process();
```

Optional parameters in lambda expressions

A.K.A. Default lambda parameters

```
var process = (string name = "Default") => {  
    Console.WriteLine($"Processing: {name}");  
};
```

```
process("Filip Ekberg");
```

```
process();
```



```
Processing: Filip Ekberg  
Processing: Default
```

Alias any type

```
using Point = (int x, int y);

void Refresh(Point p)
{
    Console.WriteLine(p.x);
    Console.WriteLine(p.y);
}
```

Interceptors

```
1  public class Logger
2  {
3      public void Log(string message)
4      {
5          LogInternal(message);
6      }
7  }
8
9  public static class Interceptor
10 {
11     [InterceptsLocation("Logger.cs",
12         5,
13         9)]
14     public static void DebugLogger(this Logger logger, string message)
15     => Console.WriteLine(message);
```

Interceptors in ASP.NET Core

Improved performance using Interceptors

The Request Delegate Generator uses the new [C# 12 interceptors compiler feature](#) to support intercepting calls to minimal API's `Map` action methods with statically generated variants at runtime. The use of interceptors results in increased startup performance for apps compiled with `PublishAot`.

learn.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-8.0

What's next?

github.com/dotnet/roslyn/blob/main/docs/Language%20Feature%20Status.md



Filip Ekberg

Pluralsight Author

Following

2807 Followers

Filip is an enthusiastic developer that strives to learn something new every day. With over a decade of experience in .NET, Filip actively spreads his knowledge and ideas around the globe, be it speaking at conferences or online. Filip has worked in a range of different technologies such as WPF,...

Show more...

[fekberg.com](#)

[Twitter](#)

[Facebook](#)

[LinkedIn](#)

CONTENT AUTHORED

24

All time

TOPICS AUTHORED



C#

TOTAL RATINGS

4,531

AVG CONTENT RATING

4.5

Content authored

Building a Real-world C# 10 Application

NEW!

Course · Intermediate · 3 hr 39m · Jun 28, 2023 · ★★★★★ (16)

Globalization and Internationalization in .NET 6

NEW!

Course · Intermediate · 2 hr 36m · Apr 21, 2023 · ★★★★★ (10)

Date and Time in .NET 6

Course · Intermediate · 2 hr 4m · Feb 24, 2023 · ★★★★★ (20)

Asynchronous Programming in C# 10

Course · Intermediate · 5 hr 1m · Oct 13, 2022 · ★★★★★ (86)

Data Access in C# 10 Fundamentals

Course · Intermediate · 4 hr 18m · Aug 15, 2022 · ★★★★★ (46)

C# 10 Advanced Language Features

Course · Intermediate · 6 hr 50m · Mar 31, 2022 · ★★★★★ (132)

Thanks!

I'm Filip Ekberg

@fekberg

fili@ekberg.dev

 **C# smorgasbord** free @ filipekberg.se

