C# and .NET

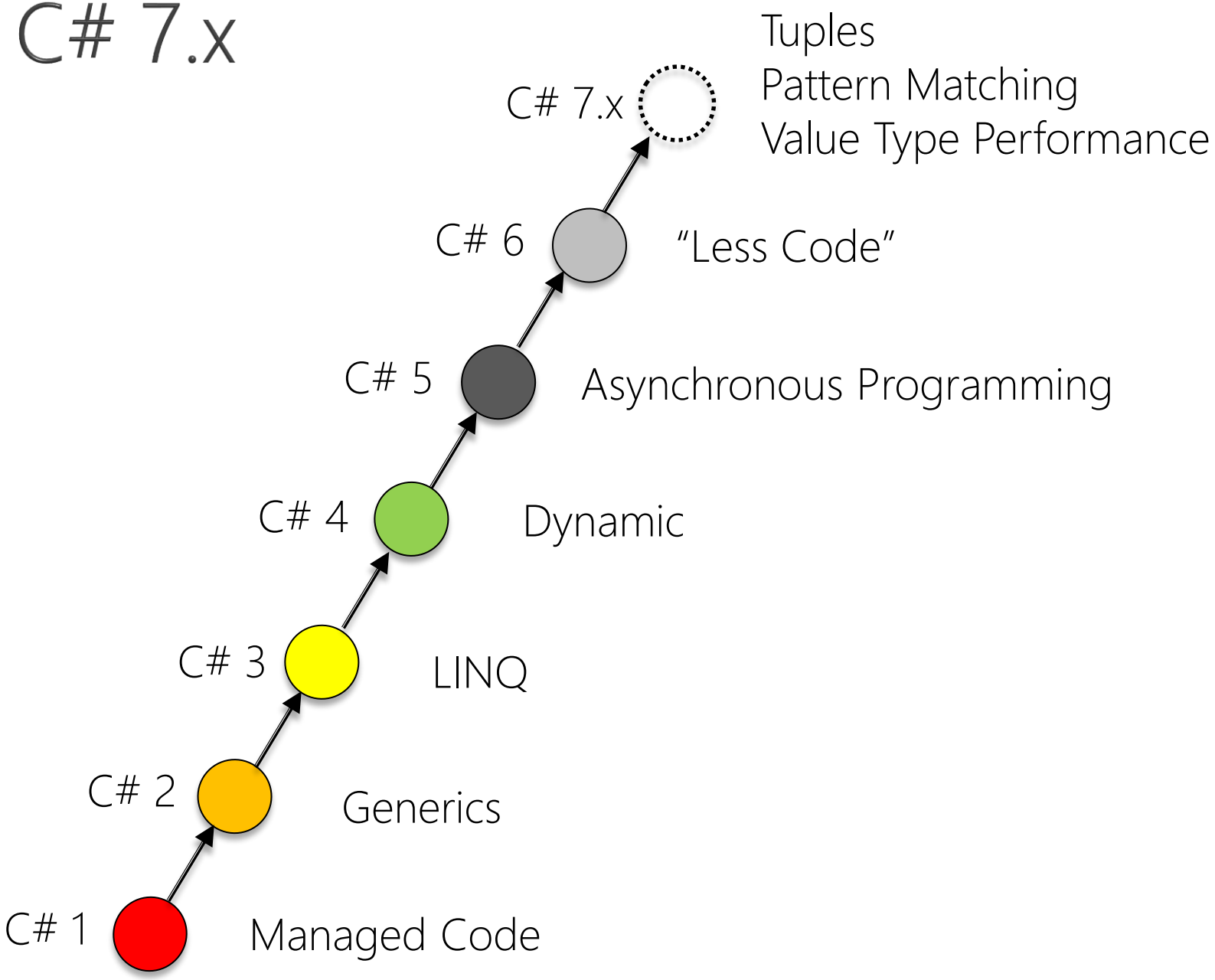
From 4.8 Framework to .NET 6

Jesper Gulmann Henriksen

PFA Seminar 2023-01-10

TEKNOLOGISK
INSTITUT

Evolution of C# 7.x



Agenda

TEKNOLOGISK INSTITUT

- Part 1: New Features of C# 8 and 9
  - Tuples and Deconstructors
  - Pattern Matchings and Switches
  - Nullable Reference Types
  - Indexes and Ranges
  - Default Interface Implementation
  - Record Classes
- Part 2: Removing Overhead and Adding Completion in C# 9 and 10
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
- Part 4: Framework Improvements

Introducing Tuples

- Not the **Tuple<T1,T2>** type already in .NET 4.0
  - Instead it is a value type with dedicated syntax

```
(int, int) FindVowels( string s )
{
    int v = 0;
    int c = 0;
    foreach (char letter in s)
    {
        ...
    }
    return (v, c);
}
```

```
string input = ReadLine();

var t = FindVowels(input);
WriteLine($"There are {t.Item1} vowels and
 {t.Item2} consonants in \"{input}\"");
```

Custom Deconstruction

- Can be easily deconstructed to individual parts

```
(int vowels, int cons) = FindVowels(input);
```

- Custom types can also be supplied with a *deconstructor* with out parameters

```
public class Employee
{
    ...
    public void Deconstruct( out string firstName, out string lastName )
    {
        firstName = FirstName;
        lastName = LastName;
    }
}
```

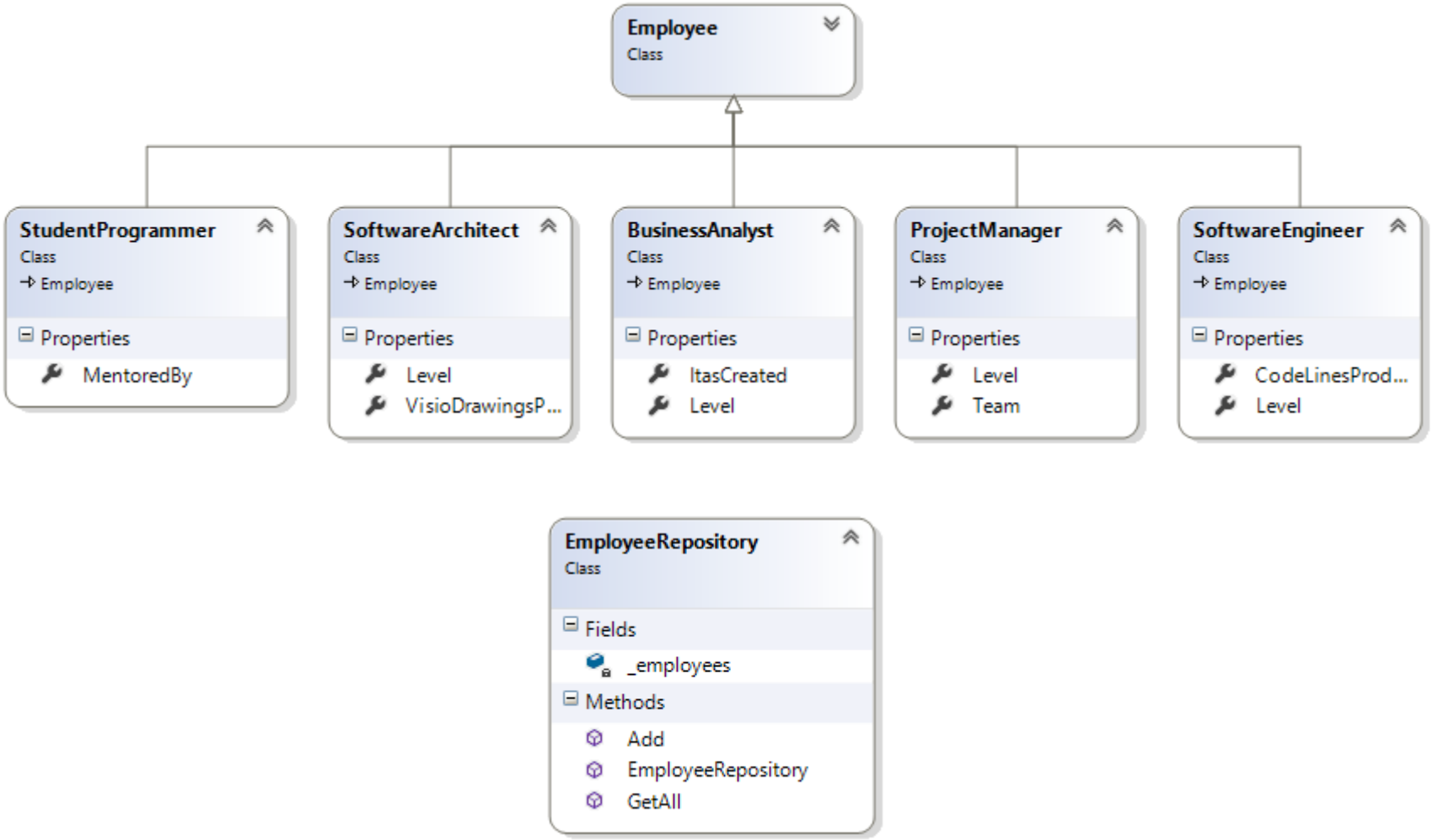
```
Employee elJefe = new Employee { ... };
var (first, last) = elJefe;
WriteLine(first);
```

- Works for two or more deconstruction parts
  - Deconstructors can be overloaded

Agenda

- Part 1: New Features of C# 8 and 9
  - Tuples and Deconstructors
  - Pattern Matchings and Switches
  - Nullable Reference Types
  - Indexes and Ranges
  - Default Interface Implementation
  - Record Classes
- Part 2: Removing Overhead and Adding Completion in C# 9 and 10
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
- Part 4: Framework Improvements

Example: Employee



Pattern Matching with `is`

- Three types of patterns for matching in C# 7
  - Constant patterns *c* e.g. `null`
  - Type patterns *T x* e.g. `int x`
  - Var patterns `var` *x*
- Matches and/or captures to identifiers to nearest surrounding scope
- More patterns are introduced in later C# versions

```
foreach (Employee e in all)
{
    if (e is SoftwareEngineer se)
    {
        WriteLine($"{se.FullName} has produced {se.CodeLinesProduced} lines of C#");
    }
}
```

- The `is` keyword is now compatible with patterns

Type Switch with Pattern Matching

- Can switch on any type
  - Case clauses can make use of patterns and new **when** conditions

```
Employee e = ...;
switch (e)
{
    case SoftwareArchitect sa:
        WriteLine($"{sa.FullName} plays with Visio");
        break;
    case SoftwareEngineer se when se.SoftwareEngineerLevel == SoftwareEngineerLevel.Lead:
        WriteLine($"{se.FullName} is a lead software engineer");
        break;
    case null:
    default:
        break;
}
```

- Cases are no longer disjoint – evaluated sequentially!

Switch Expressions

- A new functionally-inspired **switch** expression

```
string Choose( Employee employee ) =>
      employee switch
      {
            SoftwareArchitect sa => $"Hello, Mr. Architect {sa.LastName}",
            SoftwareEngineer se => "Please code!",
            StudentProgrammer sp => $"Please get coffee, {sp.FirstName}",
            _ => "Have a nice day... :-)"
      }
}
```

- Produces a value, so
  - no fallthrough!
  - **case** and **:** elements are replaced with **=>**
  - **default** case is replaced with a **_**
  - bodies can only be expressions (not statements!)

C# 8 Pattern Matching Enhancements

- C# 7 introduced three patterns for matching
  - Constant patterns *c* e.g. `null`
  - Type patterns *T x* e.g. `int x`
  - Var patterns `var` *x*

- C# 8 introduces three additional patterns for matching
  - Property patterns *Type{ p1: v1, ... , pn: vn }* e.g. `{IsValid: false}`
  - Tuple patterns *(x1, ... , xn)* e.g. `(42, 87)`
  - Positional patterns *Type(x1, ... , xn)* e.g. `Album(s, age)`

- Moreover, in C# 8 patterns are now be "**compositional**"!

TEKNOLOGISK INSTITUT

Property Patterns

- Property patterns match member properties to values

```
string Evaluate( SoftwareEngineer se ) =>
    se switch
    {
        { Level: SoftwareEngineerLevel.Lead } => $"{se.FullName} does great work",
        { Level: SoftwareEngineerLevel.Chief } => $"You da boss, {se.FullName}",
        null => "You're not even a software engineer, dude!",
        _ => $"Well done coding SOLID, {se.Level}... :-)"
     }
}
```

- Also works for multiple, simultaneous name-value pairs

TEKNOLOGISK INSTITUT

Property Patterns Variations

- Can in fact simultaneously match the type as well…

```
string Evaluate( Employee employee ) =>
    employee switch
    {
        SoftwareEngineer { Level: SoftwareEngineerLevel.Lead } => $"...",
        SoftwareArchitect { Level: SoftwareArchitectLevel.Chief } => $"...",
        _ => $"Well done making the company thrive... :-)"
     }
}
```

- Not tied to **`switch`** expressions: Also works for **`is`** etc.

Tuple Patterns

- Tuple patterns use two or more values for matching

```
Hand left = GetRandomMember<Hand>();
Hand right = GetRandomMember<Hand>();

Outcome winner = (left, right) switch
{
    (Hand.Paper, Hand.Rock) => Outcome.Left,
    (Hand.Paper, Hand.Scissors) => Outcome.Right,
    (Hand.Rock, Hand.Paper) => Outcome.Right,
    (Hand.Rock, Hand.Scissors) => Outcome.Left,
    (Hand.Scissors, Hand.Paper) => Outcome.Left,
    (Hand.Scissors, Hand.Rock) => Outcome.Right,
    (_,_) => Outcome.Tie
};
```

Positional Patterns

- Positional patterns use deconstructors for matching

```
Album album = new Album(
    "Depeche Mode",
    "Violator",
    new DateTime(1990, 3, 19)
);

string description = album switch
{
    Album(_, string s, int age) when age >= 25 => $"{s} is vintage <3",
    Album(_, string s, int age) when age >= 10 => $"{s} is seasoned",
    Album(_, string s, _) => $"{s} is for youngsters only! ;-)"
};
```

- Can be simplified using **`var`**

C# 9 Pattern Matching Enhancements

- C# 7 and 8 introduced a total of 6 patterns

- C# 9 introduces 6 additional patterns or enhancements:

• Type patterns	*Type*	e.g.	`int`
• Negation patterns	not *P1*	e.g.	`not null`
• Parenthesized patterns	(*P*)	e.g.	`(string)`
• Conjunctive patterns	*P1 and P2*	e.g.	`A and (not B)`
• Disjunctive patterns	*P1 or P2*	e.g.	`int or string`
• Relational patterns	*P1* < *P2*	e.g.	`< 87`
	P1 <= *P2*	e.g.	`<= 87`
	P1 > *P2*	e.g.	`> 87`
	P1 >= *P2*	e.g.	`>= 87`

Type Patterns

- This is more or less only a compiler-theoretic enhancement
  - But now it "mixes better" with the new or compound patterns

```
object o1 = 87;
object o2 = "Yeah!";

var t = (o1, o2);

if (t is (int, string))
{
    Console.WriteLine("o1 is an int and o2 is a string");
}
```

TEKNOLOGISK INSTITUT

Negation Patterns

- At last(!) we are allowed negative pattern assertions

```
public void DoStuff(object o)
{
    if( o is not null )
    {
        Console.WriteLine(o);
    }
}
```

Parenthesized Patterns

- This is simply a means to disambiguate parsing
  - Carries not semantic meaning in itself

```
public string WhatIsIt(object o) =>
    o switch
    {
        (((string))) => "string",
        (((int))) => "int",
        _ => "Something else :-)",
     };
```

- But has tremendous significance for the other patterns following shortly…

Conjunctive Patterns

- Conjunctive patterns specify an **and** between patterns

```
string evalution = employee switch
{
    (not ProjectManager) and (not StudentProgrammer) =>
        "Codes a little",
    _ => "Probably codes a bit more..."
};

Console.WriteLine($"{employee.FullName}: {evalution}");
```

TEKNOLOGISK INSTITUT

Disjunctive Patterns

- Disjunctive patterns specify an **`or`** between patterns

```
IEnumerable<object> elements = new List<object>
{
    42, "Yay", 87.0, "Nay", 12.7m
};

foreach (var o in elements)
{
    Console.WriteLine(o switch {
        int or double or decimal => $"{o} is a number",
        _ => "Not a number..."
    });
}
```

Relational Patterns

- The relational patterns are all the "usual" comparisons
  - **<** , **<=** , **>** , **>=**

```
int temperature = int.Parse(Console.ReadLine());
string forecast = temperature switch
{
    <= 0 => "Freezing...",
    < 12 => "Autumn-like",
    <= 19 => "Spring-ish",
    <= 40 => "Summer!",
    _ => "Death Valley?"
};
```

- Note that there is no **=>**

C# 10 Pattern Matching Enhancements

- C# 7, 8, and 9 introduced a total of 12 patterns and enhancements
- C# 10 introduces just one: Extended Property Pattern
  - *Type{ p1.p2: v }*

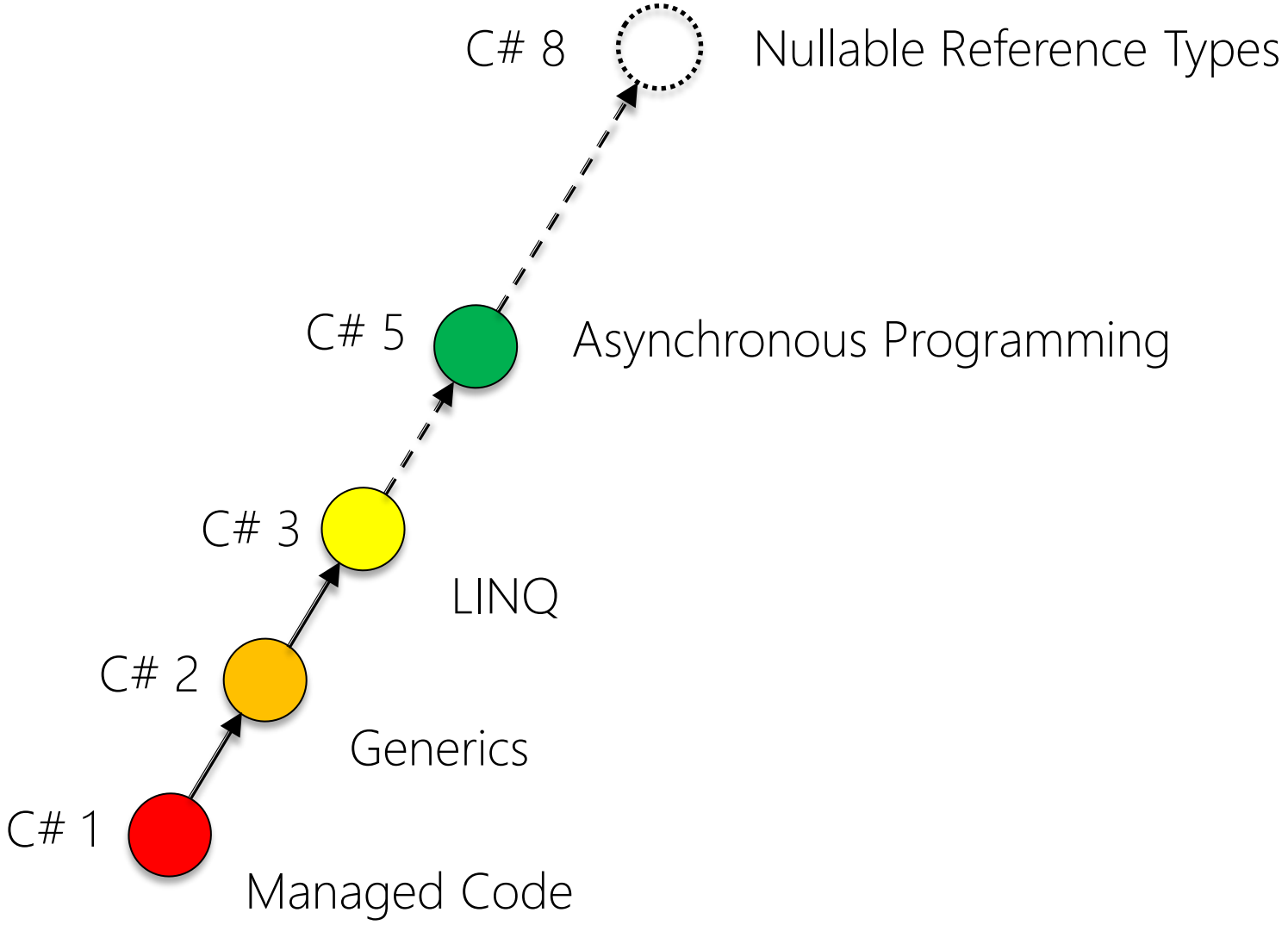
```
record class Company(string Name, Company? OwnedBy = default);
```

```
var query = companies
    .Where(c => c is { OwnedBy.OwnedBy.Name: "Sharp10" })
    ;
```

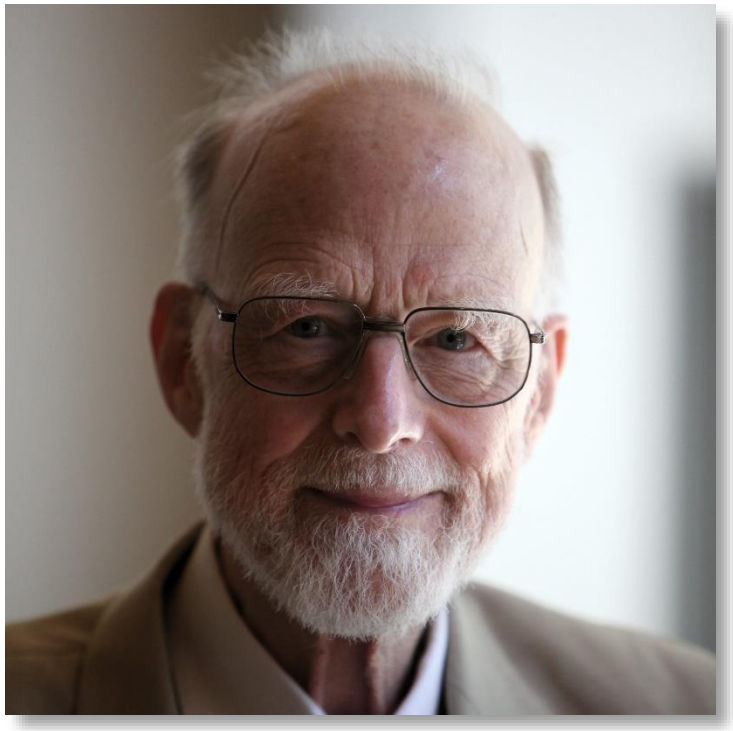
Agenda

- Part 1: New Features of C# 8 and 9
  - Tuples and Deconstructors
  - Pattern Matchings and Switches
  - Nullable Reference Types
  - Indexes and Ranges
  - Default Interface Implementation
  - Record Classes
- Part 2: Removing Overhead and Adding Completion in C# 9 and 10
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
- Part 4: Framework Improvements

Evolution of C# 8



Null References: "The Billion-dollar Mistake"



– Tony Hoare 2009

"I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years."

Introducing Nullable Reference Types

- C# 8 allows declaring intent of reference types
  - *Nonnullable Reference Types*
    - A reference is not supposed to be null
  - *Nullable Reference Types*
    - A reference is allowed to be null

```
class Person
{
    public string  FirstName { get; }  // Non-nullable string
    public string? MiddleName { get; } // Nullable string
    public string  LastName { get; }   // Non-nullable string

    ...
}
```

- Traditionally, C# reference types do not make this distinction!

TEKNOLOGISK INSTITUT

Static Analysis

- Produces compile-time static analysis warning when
  - Setting a **nonnullable** to null
  - Dereferencing a **nullable** reference

```
class Person
{
    public string FirstName { get; }
    public string? MiddleName { get; }
    public string LastName { get; }

    public Person( string firstName ) => FirstName = firstName;

    int GetLengthOfMiddleName( Person p ) => p.MiddleName.Length;
}
```

Null-forgiving Operator

- You can assert to the compiler that a reference is not null using the *Null-forgiving Operator* **!**

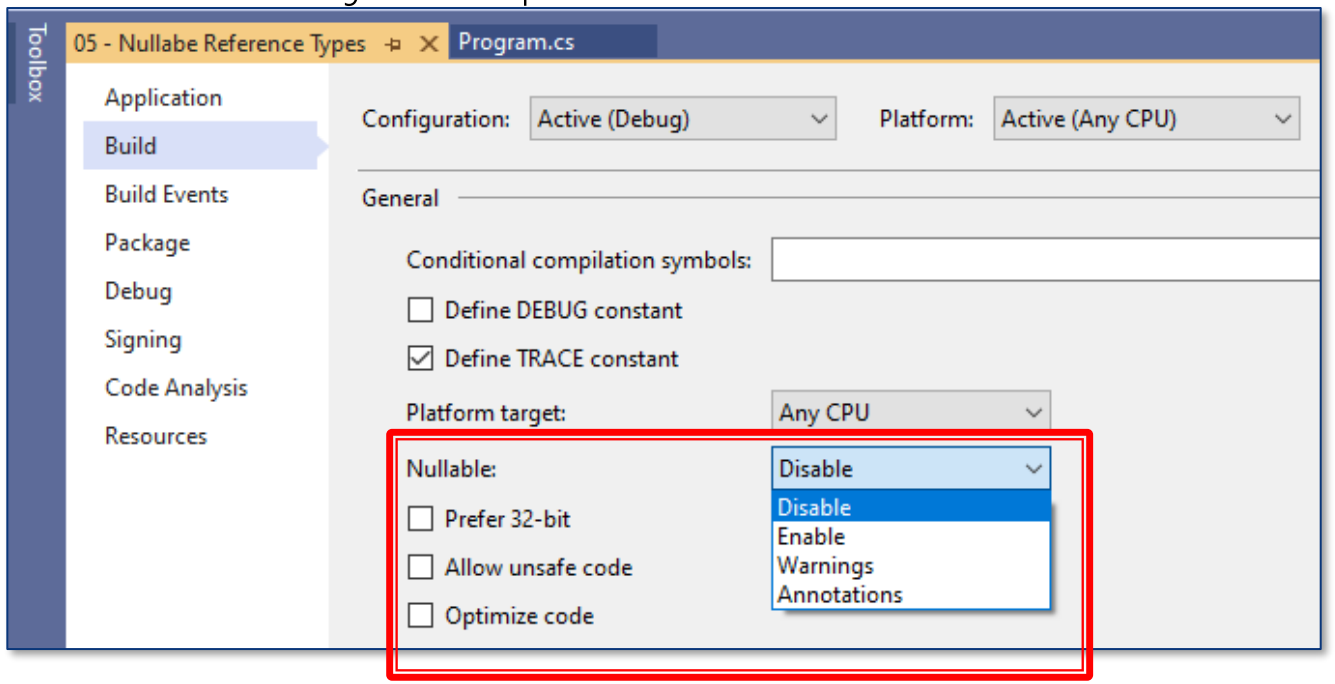
```
class Person
{
    public string FirstName { get; }
    public string? MiddleName { get; }
    public string LastName { get; }

    public Person( string firstName ) => FirstName = firstName;

    int GetLengthOfMiddleName( Person p ) => p.MiddleName!.Length;
}
```

Wait a Minute...!?

- Not Backwards Compatible with C# 7.x!
- Behavior can be controlled in Project Properties



- Nullable Contexts
  - Annotations
  - Warnings

Annotations + Warning Contexts

- Can also be enabled/disabled locally by means of compiler directive `#nullable`
  - `enable` / `disable` / `restore`
  - `warnings` / `annotations`

```
class Person
{
    public string FirstName { get; }
    public string? MiddleName { get; }
    public string LastName { get; }

#nullable disable
    public Person( string firstName ) => FirstName = firstName;
#nullable restore
}
```

Agenda

- Part 1: New Features of C# 8 and 9
  - Tuples and Deconstructors
  - Pattern Matchings and Switches
  - Nullable Reference Types
  - Indexes and Ranges
  - Default Interface Implementation
  - Record Classes
- Part 2: Removing Overhead and Adding Completion in C# 9 and 10
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
- Part 4: Framework Improvements

Indices

- The `^` operator describes the end of the sequence

```
string[] elements = new string[]
{
    "Hello", "World", "Booyah!", "Foobar"
};

Console.WriteLine(elements[^1]);
Console.WriteLine(elements[^0]); // ^0 == elements.length
Index i = ^2;
Console.WriteLine(elements[i]);
```

- Indices are captured by a new **`System.Index`** type
  - Can be manipulated using variables etc. as any other type

Ranges

- The .. operator specifies (sub)ranges using indices *i* and *j*
  - **`i..j`** Full sequence (start is inclusive, end is exclusive)
  - **`i..`** Half-open sequence (start is inclusive)
  - **`..i`** Half-open sequence (end is exclusive)
  - **`..`** Entire sequence (equivalent to **`0..^0`**)

```
foreach (var s in elements[0..^2])
{
    Console.WriteLine( s );
}

Range range = 1..;
```

- Ranges are captured by a new **`System.Range`** type
  - Can be manipulated using variables etc. as any other type

Supported Types

`string`	Indices	Ranges
`Array`	Indices	Ranges
`List<T>`	Indices	
`Span<T>`	Indices	Ranges
`ReadOnlySpan<T>`	Indices	Ranges

- Any type that provides an indexer with a `System.Index` or `System.Range` parameter (respectively) explicitly supports indices or ranges

- Compiler will implement some implicit support for indices and ranges

Agenda

TEKNOLOGISK INSTITUT

- Part 1: New Features of C# 8 and 9
  - Tuples and Deconstructors
  - Pattern Matchings and Switches
  - Nullable Reference Types
  - Indexes and Ranges
  - Default Interface Implementation
  - Record Classes
- Part 2: Removing Overhead and Adding Completion in C# 9 and 10
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
- Part 4: Framework Improvements

Default Interface Members

- Allow better backwards compatibility in interfaces

```
interface ILogger
{
    void Log(LogLevel level, string message);
    void Log(Exception ex) => Log(LogLevel.Error, ex.ToString());
}
```

```
class FileLogger : ILogger
{
    public void Log(LogLevel level, string message) { ... }
}
```

```
class ConsoleLogger : ILogger
{
    public void Log(LogLevel level, string message) { ... }
    public void Log(Exception ex) { ... }
}
```

Static Members in Interfaces

```
interface ILogger
{
    static string ProduceExceptionLog(Exception exception) =>
                $"Exception occurred: {exception.Message}. " +
                $"Call stack size: {exception.StackTrace?.Length ?? 0}";

    void Log(LogLevel level, string message);
    void Log(Exception ex) =>
        Log(LogLevel.Error, ProduceExceptionLog(ex));
}
```

```
class ConsoleLogger : ILogger
{
    public void Log(Exception ex)  =>
        ... ILogger.ProduceExceptionLog(exception) ...
    ...
}
```

C# 8 Interfaces vs. Classes

- Default interface members cannot be invoked on concrete classes – only through the interface!
  - Bears resemblance to explicit interface implementation
- But…

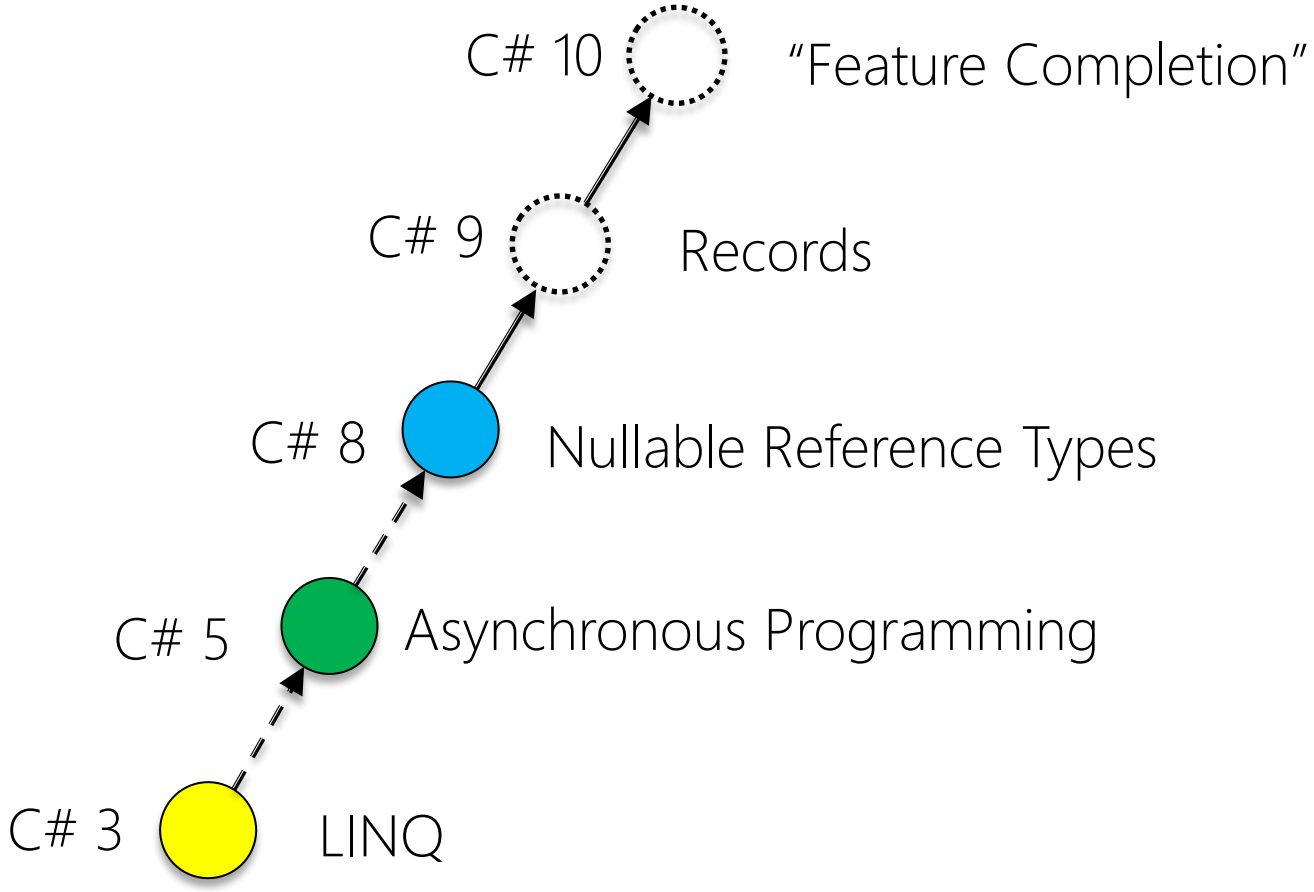
- Static members can have access modifiers in interfaces..!
  - Default access modifier on interface members: `public`
  - Default access modifier on class members: `private`

Agenda

- Part 1: New Features of C# 8 and 9
  - Tuples and Deconstructors
  - Pattern Matchings and Switches
  - Nullable Reference Types
  - Indexes and Ranges
  - Default Interface Implementation
  - Record Classes
- Part 2: Removing Overhead and Adding Completion in C# 9 and 10
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
- Part 4: Framework Improvements

TEKNOLOGISK INSTITUT

Evolution of C# 9 and C# 10



TEKNOLOGISK INSTITUT

Init-only Setters

- Restricted setter only allowing initialization:

```
public sealed class Album
{
    public string Artist { get; init; }
    public string AlbumName { get; init; }
    public DateTime ReleaseDate { get; init; }

    ...
}
```

- Can have usual visibility on init-only setter
- Can not have both **`init`** and **`set`**...!

Records

- Records are simpler, immutable classes

```
record Person(string FirstName, string LastName);
```

- Defines init-only properties with "Primary Constructors"

- Can have additional properties + methods, of course

```
record Album(string Artist, string AlbumName, DateTime ReleaseDate)
{
    public int Age
    {
        get { ... }
    }
}
```

Built-in Features of Records

- Overrides
  - `ToString()`
  - `Equals()` (Implements `IEquatable<T>`)
  - `GetHashCode()`
  - `==` and `!=`
- What about `ReferenceEquals`?
- Supplies built-in deconstructors

Mutation-free Copying

- Additional keyword: Create copies using **with**

```
Album album = new Album("Prince",
                        "Purple Rain",
                        new DateTime(1984, 11, 02));

Album renamed = album with
{
    Artist = "The Artist Formerly Known as..."
};
```

- Does not mutate source record
  - Copies and replaces

Records and Inheritance

- Almost all OO aspects are identical to classes
  - Visibility, parameters, etc.
  - But Records and Classes cannot mix inheritance!

- Can override and change built-in method overrides, if needed

Agenda

- Part 1: New Features of C# 8 and 9
- **Part 2: Removing Overhead and Adding Completion in C# 9 and 10**
  - **Record Structs**
  - Top-level Statements
  - Target-typed New
  - Covariant Return Types
  - Improved Namespaces and Usings
  - New LINQ methods in .NET 6
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
- Part 4: Framework Improvements

TEKNOLOGISK INSTITUT

C# 9 Object-oriented Topology

- Value Types:
- `struct`

- Reference Types:
- `class`
  - `record`
- Anonymous Types

TEKNOLOGISK INSTITUT

C# 10 Object-oriented Topology

- Value Types:
- `struct`
  - `record struct`

- Reference Types:
- `class`
  - `record class`
- Anonymous Types

Record Structs and Record Classes

- Use **`record struct`** for "value-type records"

```
Money m1 = new(87, 25);
Money m2 = new(87, 25);

Console.WriteLine(m1 == m2);

record struct Money( int Euro, int Cents)
{
    public int TotalCents => Euro * 100 + Cents;
}
```

- Use **`record`** or **`record class`** for "reference-type records"

Comments on Record Structs

- **`record class`**
  - Immutable for primary constructor parameters
  - Mutable for other properties
- **`record struct`**
  - Mutable for primary constructor parameters
  - Mutable for other properties

- However, thinking back to C# 7.x:

- **`readonly record struct`**
  - Immutable for primary constructor parameters
  - Other properties are not allowed to be mutable!

C# 10 Additions to Structs

- Default constructors and initializers are allowed in C# 10

```
struct Money
{
    public int Euro { get; set; } = 99;
    public int Cents { get; set; } = 99;

    public Money()
    {
        Euro = 1;
        Cents = 0;
    }
}
```

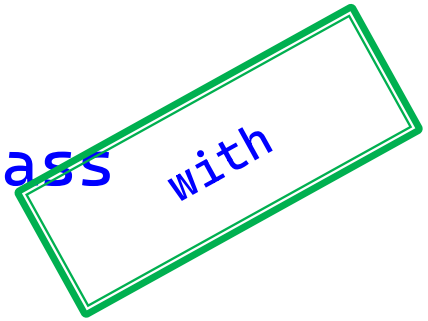
- Note: Beware of relation to **`default`** keyword and/or non-default constructors

TEKNOLOGISK INSTITUT

C# 9 Non-destructive Mutation

- Value Types:
- `struct`

- Reference Types:
- `class`
- `record class`

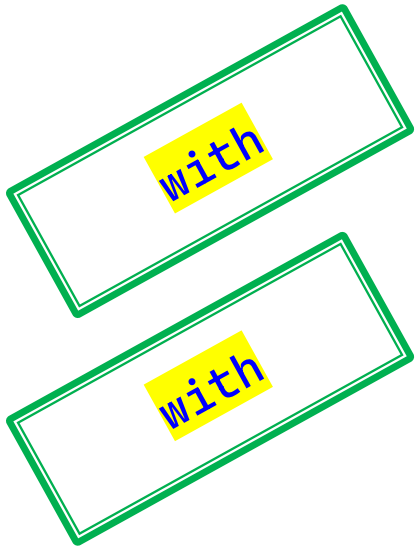


- Anonymous Types

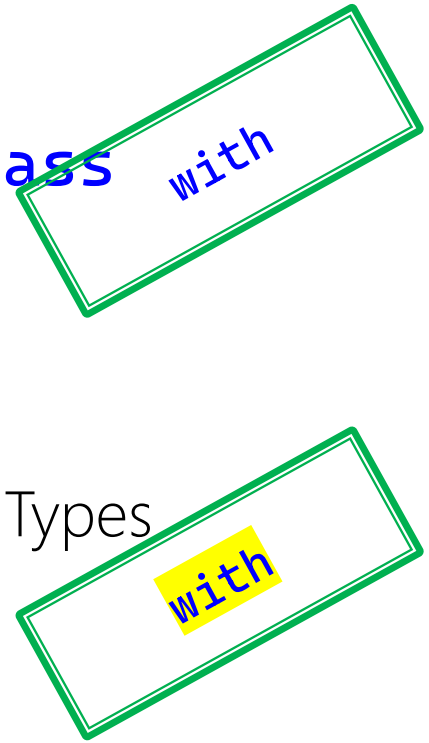
TEKNOLOGISK INSTITUT

C# 10 Non-destructive Mutation

- Value Types:
  - `struct`
  - `record struct`



- Reference Types:
  - `class`
  - `record class`
  - Anonymous Types



TEKNOLOGISK INSTITUT

Non-destructive Mutation Extended

- C# 9 **with** expressions allowed for all structs in C# 10

```
struct Money
{
    public int Euro { get; set; };
    public int Cents { get; set; };
}
```

```
Money m1 = new(87, 25);
Money m2 = m1 with { Cents = 87 };
```

- C# 9 **with** expressions also allowed for anonymous types

```
var p1 = new { FirstName = "Bruce", LastName = "Wayne" };
var p2 = p1 with { LastName = "Campbell" };
```

Agenda

- Part 1: New Features of C# 8 and 9
- **Part 2: Removing Overhead and Adding Completion in C# 9 and 10**
  - Record Structs
  - **Top-level Statements**
  - Target-typed New
  - Covariant Return Types
  - Improved Namespaces and Usings
  - New LINQ methods in .NET 6
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
- Part 4: Framework Improvements

TEKNOLOGISK INSTITUT

Top-level Statements

- A fundamental rule of C# has now been relaxed:

```
using System;
namespace Wincubate.CS9.B
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

- But what about the arguments then?

Agenda

- Part 1: New Features of C# 8 and 9
- Part 2: Removing Overhead and Adding Completion in C# 9 and 10
  - Record Structs
  - Top-level Statements
  - Target-typed New
  - Covariant Return Types
  - Improved Namespaces and Usings
  - New LINQ methods in .NET 6
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
- Part 4: Framework Improvements

Target-typed New

- Target-typed new expressions are essentially the "counterpart" of **`var`**

```
Dictionary<string, List<int>> field = new()
{
   { "item1", new() { 1, 2, 3 } }
};
```

- There are a number of disallowed scenarios:
  - Interfaces
  - Enums
  - Dynamic types
  - Tuples
  - …

Agenda

- Part 1: New Features of C# 8 and 9
- **Part 2: Removing Overhead and Adding Completion in C# 9 and 10**
  - Record Structs
  - Top-level Statements
  - Target-typed New
  - **Covariant Return Types**
  - Improved Namespaces and Usings
  - New LINQ methods in .NET 6
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
- Part 4: Framework Improvements

Covariant Return Types

- Return types for methods are now relaxed to covariance

```
public class ConfigProvider
{
   public virtual Config GetConfig() { ... }
}

public class AppleConfigProvider : ConfigProvider
{
    public override AppleConfig GetConfig() { ... }
}
```

- Must exist an implicit conversion

- What about interface, strings, int, doubles, … ?

Agenda

- Part 1: New Features of C# 8 and 9
- **Part 2: Removing Overhead and Adding Completion in C# 9 and 10**
  - Record Structs
  - Top-level Statements
  - Target-typed New
  - Covariant Return Types
  - **Improved Namespaces and Usings**
  - New LINQ methods in .NET 6
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
- Part 4: Framework Improvements

Using Declarations

- Instruct compiler to dispose at the end of the scope

```
using FileStream inStream = File.OpenRead(sourceFilePath);
using FileStream outStream = File.Create(destinationFilePath);
using DeflateStream compress = new DeflateStream(
    outStream, CompressionMode.Compress );

for (int i = 0; i < inStream.Length; i++)
{
    compress.WriteByte((byte)inStream.ReadByte());
}

// inStream, outStream, compress are disposed here at the end of the scope!
```

- Also works for the new async disposables **`await using`**!

File-scoped Namespace Declarations

- In spirit of the **`using`** directive, the **`namespace`** declarations have been "horizontally optimized" similarly

```
namespace Wincubate.CS10.Shapes;

interface IShape
{
    double Area { get; }
}

class Rectangle : IShape
{
    ...
}
```

- How often do you have multiple namespaces in same file?

TEKNOLOGISK INSTITUT

Global Using Directives

- Project-wide **`using`** directives are now supported

```
global using System.Text.Json;

namespace Wincubate.CS10.Shapes;

record Rectangle(double Width, double Height) : IShape
{
    public double Area => Width * Height;

    public string Serialize() => JsonSerializer.Serialize(this);
}
```

- Also works for **`using static`**

TEKNOLOGISK INSTITUT

Implicit Usings

- Implicit **`using`**s are enabled in project file for **new** projects

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
    <RootNamespace>Wincubate.CS10.A</RootNamespace>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>

</Project>
```

- .NET libraries supply default implicit **`using`**s

TEKNOLOGISK INSTITUT

Custom Implicit Usings

- You can configure your custom implicit **`using`**s

```
<Project Sdk="Microsoft.NET.Sdk">
  ...
 <ItemGroup Condition="'$(ImplicitUsings)' == 'enable'">
    <Using Remove="System.Threading" />
    <Using Include="System.Text.Json" />
    <Using Include="Wincubate.CS10.Shapes" />
    <Using Static="true" Include="System.Console"/>
  </ItemGroup>

</Project>
```

- An alternative to **`GlobalUsings.cs`** or similar

Agenda

- Part 1: New Features of C# 8 and 9
- Part 2: Removing Overhead and Adding Completion in C# 9 and 10
  - Record Structs
  - Top-level Statements
  - Target-typed New
  - Covariant Return Types
  - Improved Namespaces and Usings
  - New LINQ methods in .NET 6
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
- Part 4: Framework Improvements

LINQ Additions in .NET 6 Overview

- `ElementAt<T>` and `ElementAtOrDefault<T>`
  - New support for `Index`
- `Take<T>`
  - New support for `Range`

- `XxxOrDefault<T>`
  - New support for supplying default
- `Zip<T>`
  - New support for three enumerables

- New `Chunk<T>` method
- New `DistinctBy<T>`, `MinBy<T>` and `MaxBy<T>` methods
- New `UnionBy<T>`, `IntersectBy<T>`, and `ExceptBy<T>`
- New `TryGetNonEnumeratedCount<T>`

TEKNOLOGISK INSTITUT

LINQ Additions in .NET 6 for Movie Lovers ☺

```
IEnumerable<Movie> movies = new List<Movie>
{
    new("Total Recall", 2012, 6.2f),
    new("Evil Dead", 1981, 7.5f),
    new("The Matrix", 1999, 8.7f),
    new("Cannonball Run", 1981, 6.3f),
    new("Star Wars: Episode IV - A New Hope", 1977, 8.6f),
    new("Don't Look Up", 2021, 7.3f),
    new("Evil Dead", 2013, 6.5f),
    new("Who Am I", 2014, 7.5f),
    new("Total Recall", 1990, 7.5f),
    new("The Interview", 2014, 6.5f)
};
```

Agenda

- Part 1: New Features of C# 8 and 9
- Part 2: Removing Overhead and Adding Completion in C# 9 and 10
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
  - Overview and Installation of Lab Files
  - Lab Session
  - Recapping the Labs
- Part 4: Framework Improvements

Install Course Files

- Clone or Download Zip from
  - https://github.com/wincubate/pfa-net6

- Presentations
  - PDFs
  - Examples code

- Labs
  - Lab Manual
  - Files for individual labs
    - Begin
    - Complete

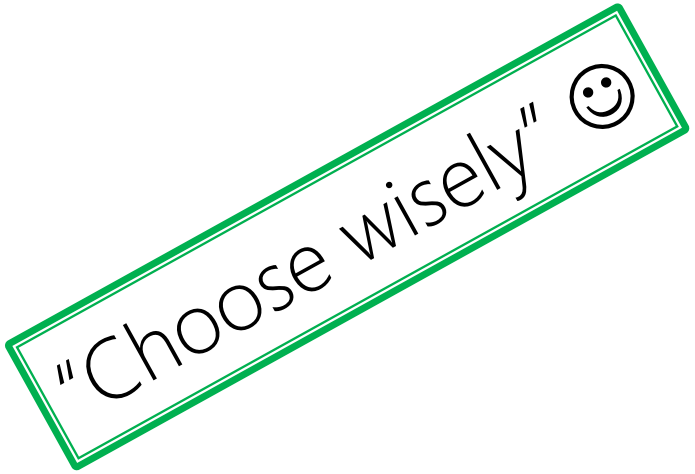
TEKNOLOGISK INSTITUT

Lab Session

Lab Recap at 14.15

- Choose **one** (or more ☺) of the following labs:

- **Lab 1.1: Adding Nullability to Reference Types**
- **Lab 1.2: Playing with Pattern Matchings**
- Lab 1.3: Convert Classes to Records
- **Lab 2.1: LINQ Additions in .NET 6**
- Lab 2.2: Refactor and Modernize to C# 10



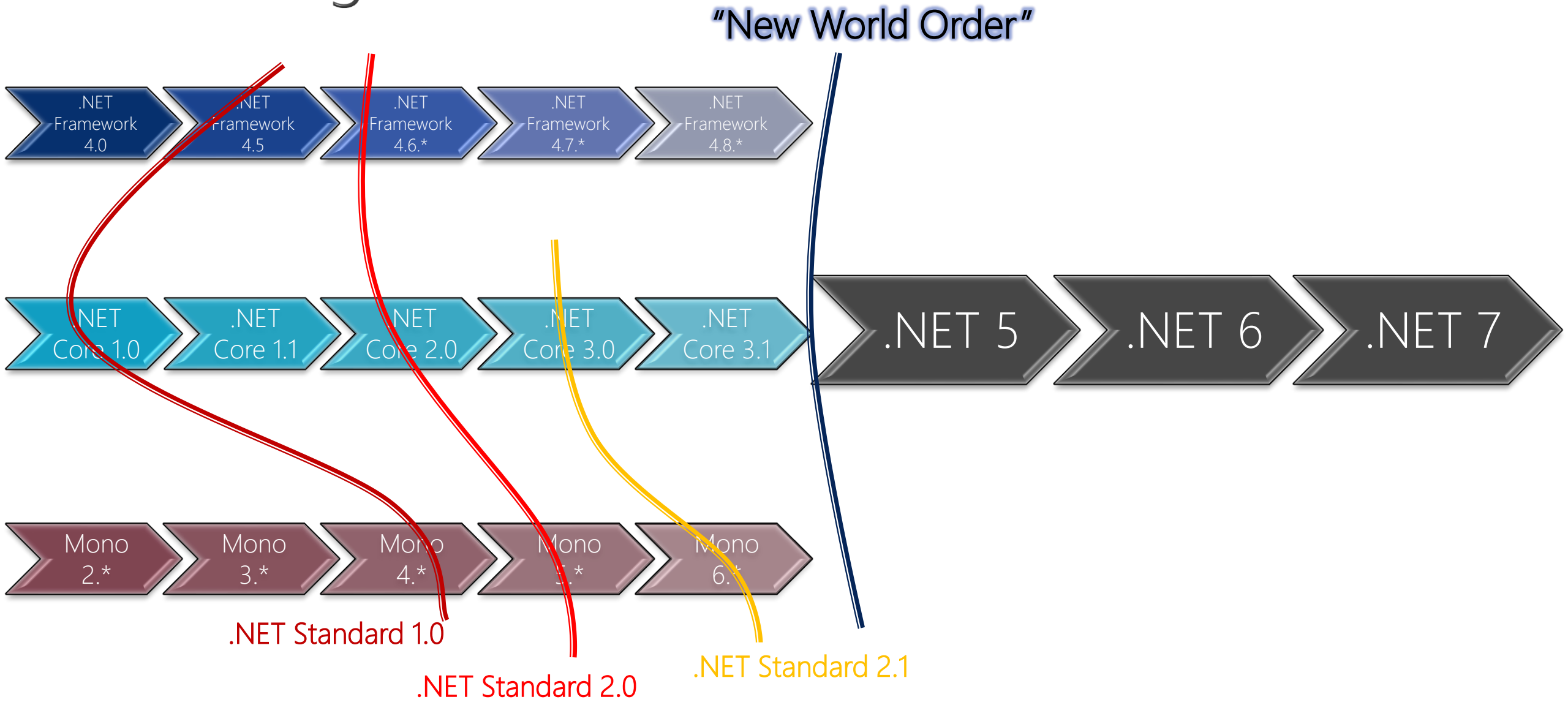
- (Lab X.1: Maximum Subsum Problem)

Agenda

- Part 1: New Features of C# 8 and 9
- Part 2: Removing Overhead and Adding Completion in C# 9 and 10
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
- **Part 4: Framework Improvements**
  - **One .NET to Rule Them All**
  - Changes in the VS and Framework
  - Introduction to the CLI and Projects
  - What Does the Future Hold…?

TEKNOLOGISK INSTITUT

.NET Jungle



.NET – A unified development platform

CLOUD

Azure

WEB

ASP.NET

Blazor

DESKTOP

.NET MAUI

WPF

WinForms

MOBILE

.NET MAUI

Xamarin

GAMING

Unity

IoT

ARM32

ARM64

AI

ML.NET

.NET for Apache Spark

ECOSYSTEM

NuGet

GitHub

Components, tools, library vendors

TOOLS

Visual Studio

Visual Studio for Mac

Visual Studio Code

CLI

.NET 6

COMMON BASE LIBARIES/APIs

INFRASTRUCTURE

RUNTIME COMPONENTS

COMPILERS

LANGUAGES

TEKNOLOGISK INSTITUT

The .NET 6 Platform

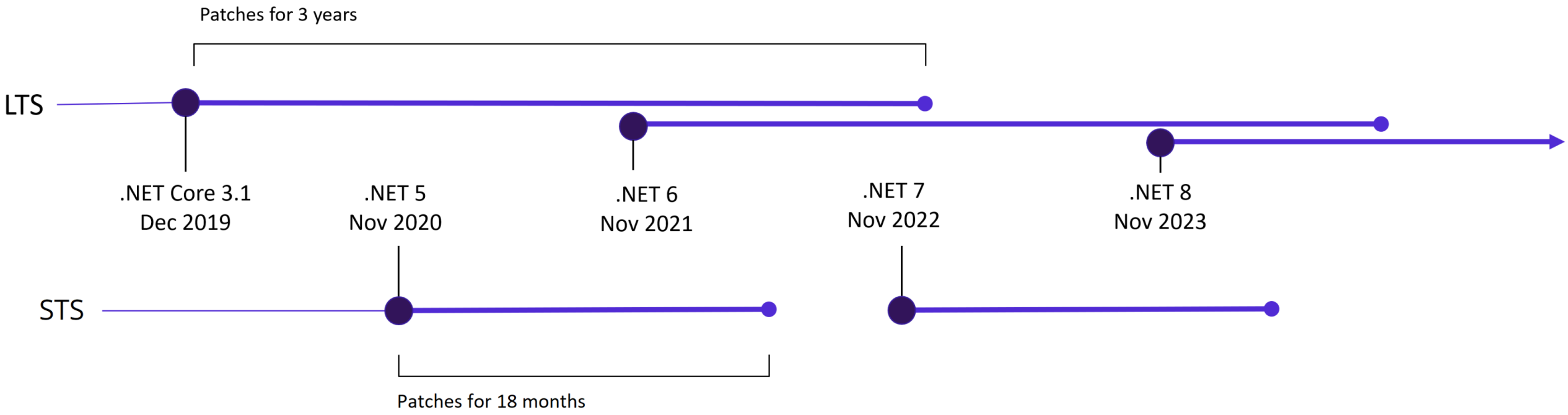
- .NET 6 represents a shared code base for
  - .NET Core
  - Mono
  - Xamarin
- Instead of targeting **`netstandard`** we now target **`net6.0`**

- For platform-specific project we now target
  - **`net6.0-windows`**
  - **`net6.0-android`**
  - **`net6.0-ios`**
  - **`net6.0-macos`**

- In this way **`net6.0`** is the next .NET Standard…!

TEKNOLOGISK INSTITUT

.NET Release Cadence



- 3-day **free** release conference at https://www.dotnetconf.net/

Agenda

- Part 1: New Features of C# 8 and 9
- Part 2: Removing Overhead and Adding Completion in C# 9 and 10
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
- **Part 4: Framework Improvements**
  - One .NET to Rule Them All
  - **Changes in the VS and Framework**
  - Introduction to the CLI and Projects
  - What Does the Future Hold…?

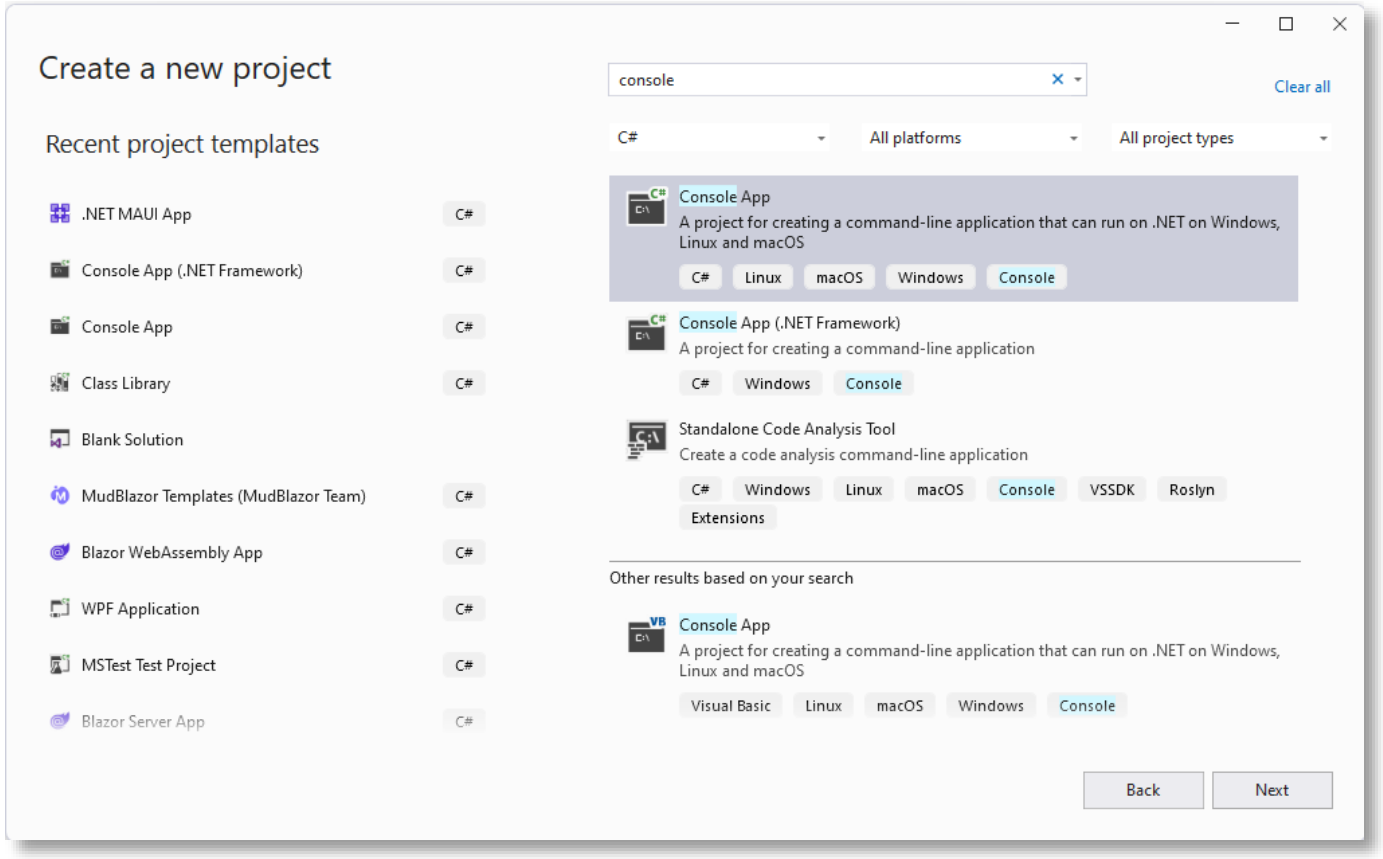
TEKNOLOGISK INSTITUT

Most Important New Features

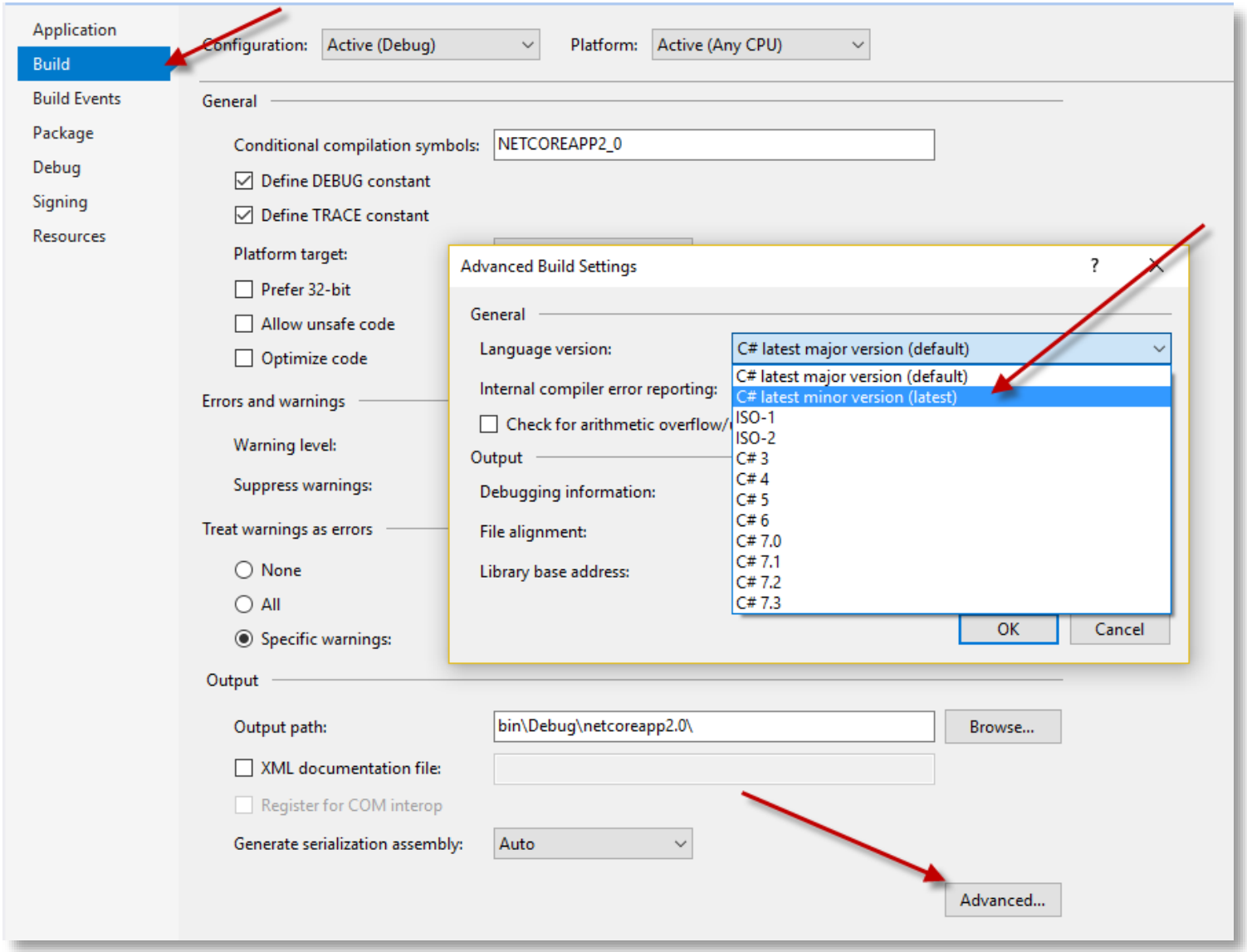
- **Visual Studio 2022 + Tooling**
  - New Language Infrastructure
  - Better Templates
  - More Integration
  - Hot Reload
  - Performance
  - CLI
  - Upgrade Assistant
- **.NET 6**
  - Blazor Server + Blazor WASM
  - .NET MAUI
  - Minimal APIs
  - EF Core 6
  - HTTP/3 + gRPC
  - System.Text.Json
  - Performance
- Release notes at https://devblogs.microsoft.com/dotnet/announcing-net-6/

Same Visual Studio, But…

- Different features depending upon .NET version
  - Separate project templates for e.g. Console App
  - Distinct property pages
  - …



Enabling C# 7.x in Visual Studio 2017



TEKNOLOGISK INSTITUT

Visual Studio 2022 Default C# Versions

Target framework	version	C# language version default
.NET	7.x	C# 11
.NET	6.x	C# 10
.NET	5.x	C# 9
.NET Core	3.x	C# 8
.NET Core	2.x	C# 7.3
.NET Standard	2.1	C# 8
.NET Standard	2.0	C# 7.3
.NET Standard	1.x	C# 7.3
.NET Framework	all	C# 7.3

- Visual Studio 2017 introduced **`LangVersion`** in project file
- Visual Studio 2019 + 2022 use defaults

Agenda

- Part 1: New Features of C# 8 and 9
- Part 2: Removing Overhead and Adding Completion in C# 9 and 10
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
- **Part 4: Framework Improvements**
  - One .NET to Rule Them All
  - Changes in the VS and Framework
  - **Introduction to the CLI and Projects**
  - What Does the Future Hold…?

.NET CLI

- Cross-platform command-line interface for projects and solutions

- **`dotnet`**
  - **`new`**
  - **`build`**
  - **`run`**
  - …

- More details at
  - [`https://learn.microsoft.com/en-us/dotnet/core/tools/`](https://learn.microsoft.com/en-us/dotnet/core/tools/)

TEKNOLOGISK INSTITUT

Migrating from .NET Framework 4.8 to .NET 6

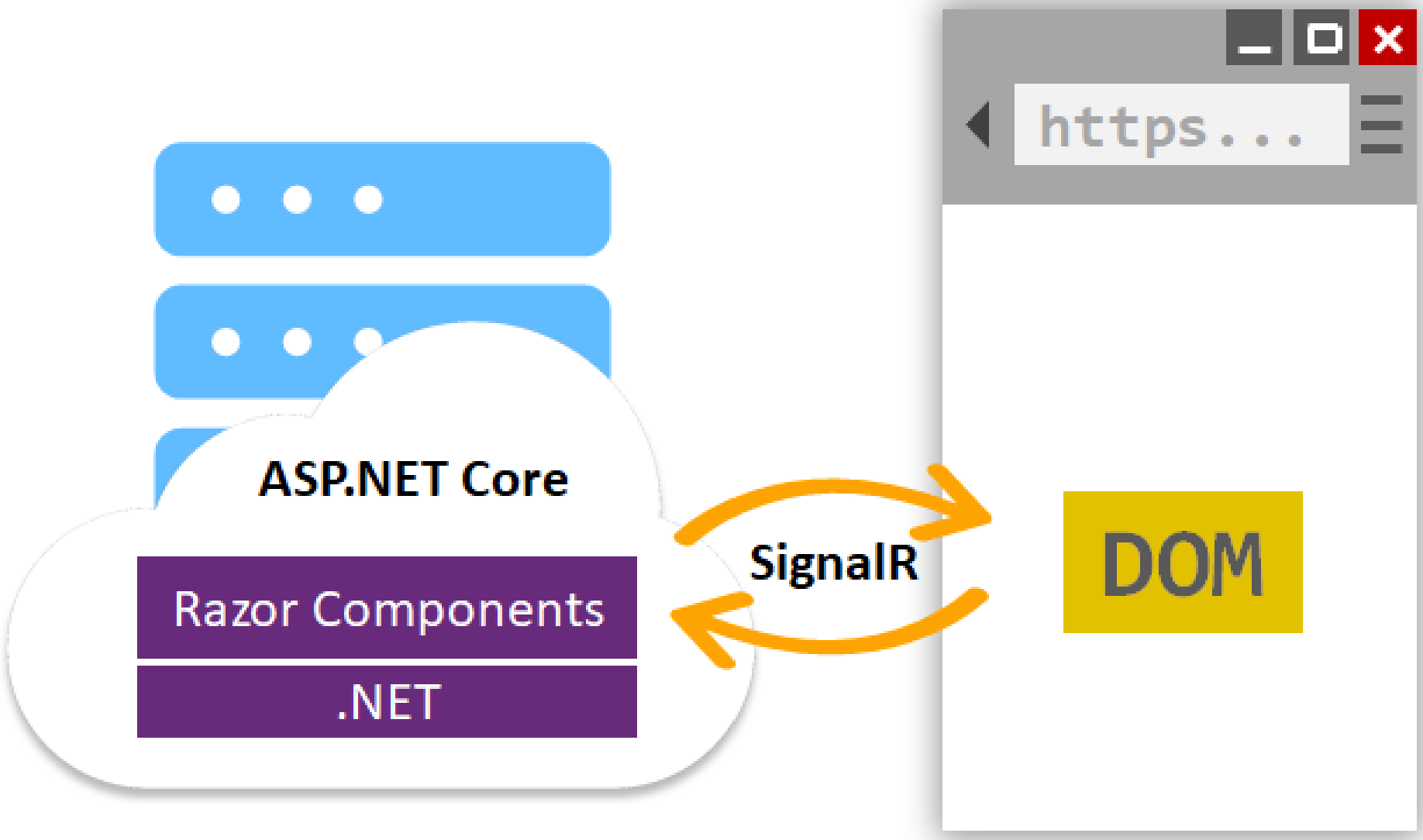
- .NET Upgrade Assistant
  - `dotnet tool install -g upgrade-assistant`
  - See https://learn.microsoft.com/en-us/dotnet/core/porting/upgrade-assistant-overview for details
- Probably best to create new .NET 6 solution/projects and move
- Tutorials
  - https://learn.microsoft.com/en-us/dotnet/core/porting/upgrade-assistant-aspnetmvc
  - …

Agenda

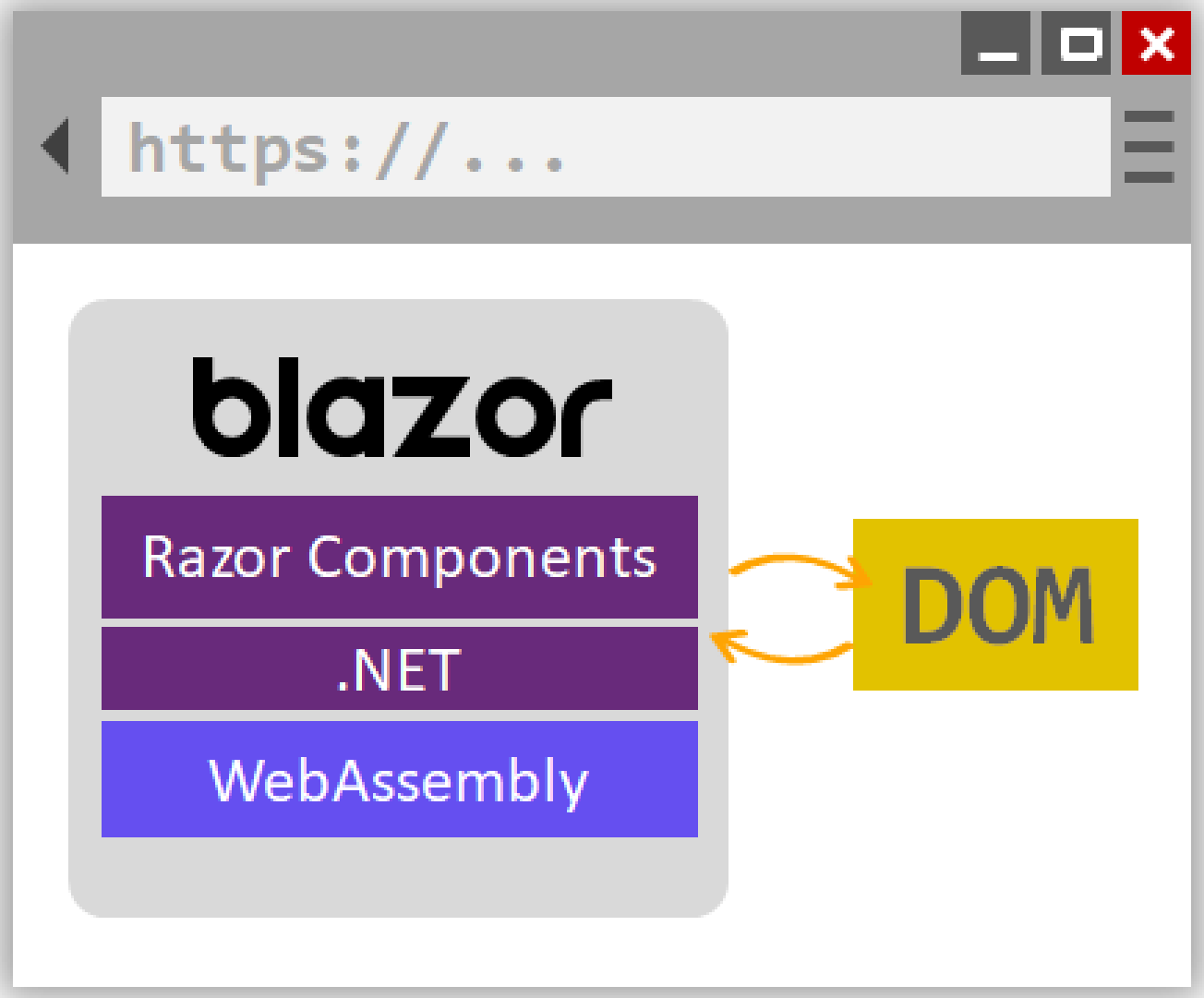
- Part 1: New Features of C# 8 and 9
- Part 2: Removing Overhead and Adding Completion in C# 9 and 10
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
- Part 4: Framework Improvements
  - One .NET to Rule Them All
  - Changes in the VS and Framework
  - Introduction to the CLI and Projects
  - What Does the Future Hold…?

TEKNOLOGISK INSTITUT

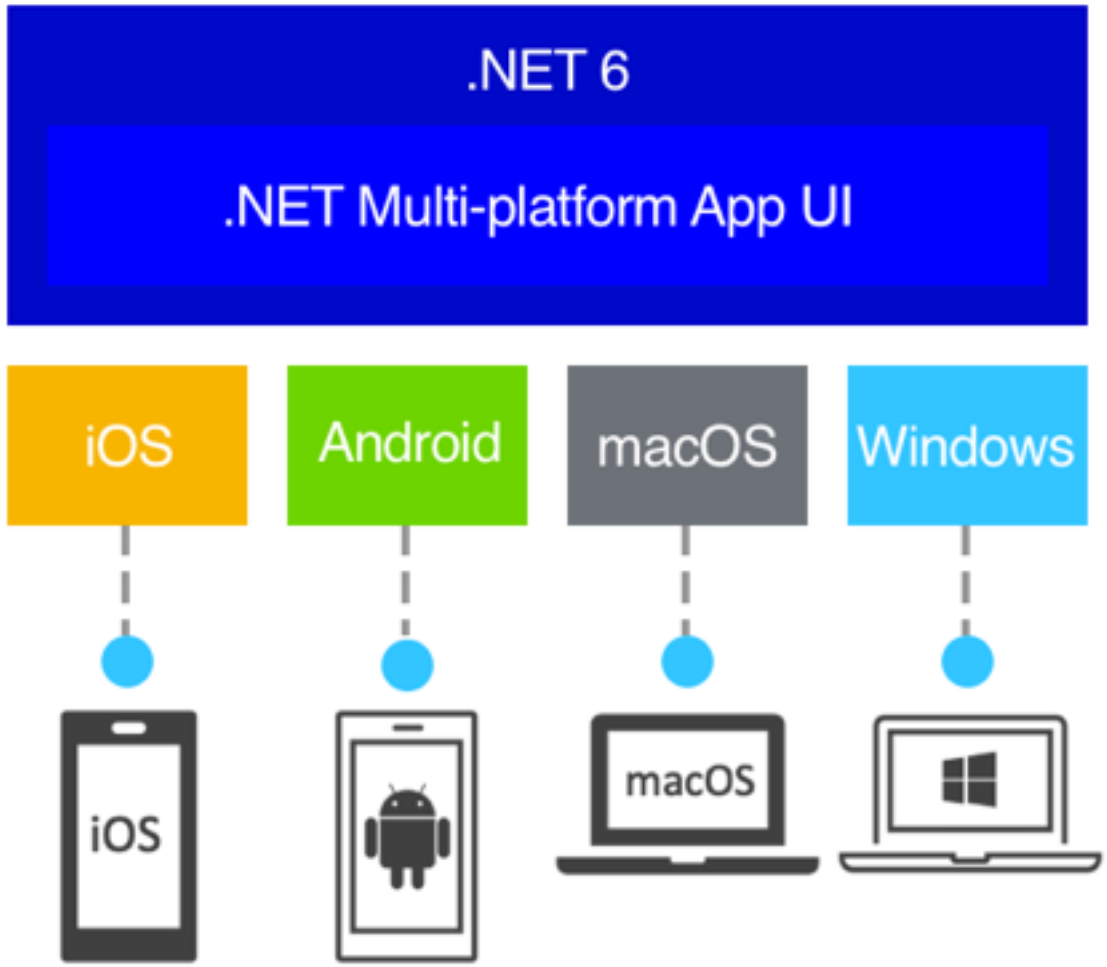
Blazor Server



Blazor WebAssembly



.NET MAUI


.NET 6
.NET Multi-platform App UI
iOS
Android
macOS
Windows
iOS
macOS

Summary

- Part 1: New Features of C# 8 and 9
- Part 2: Removing Overhead and Adding Completion in C# 9 and 10
- *Part 3: Hands-on Labs: C# 10 and .NET 6*
- Part 4: Framework Improvements

linkedin.com/in/jespergulmann/

youtube.dotnet.coach

TEKNOLOGISK
INSTITUT