Play in C#

Mads Torgersen, Microsoft

Changing our tune…

Run on Windows	→	Run everywhere
Edit in Visual Studio	→	Use your favorite editor
.NET as system component	→	Deploy with app
Run on VM (CLR)	→	Compile to native
Black box compilers	→	Open compiler APIs
Proprietary	→	Open source

Project Roslyn

“Let’s rewrite the C# and VB compilers and IDE from scratch!”

- Evolve the languages and language-based tooling
- Enable other IDEs and language-based tools
- Get them all to agree on semantics
- Dogfood!

The compiler/IDE dichotomy

Compiler

- Batch
- Throughput
- Correctness
- Prevent badness
- Reactive to what you did

IDE

- Incremental
- Responsiveness
- Error tolerance
- Enable goodness
- Predictive of what you will do

Other language understanding scenarios

- Custom diagnostics
- Source code transformation
  - Refactorings, fixers, migraters, updaters…
- Scripting
- Intermixed coding and execution
  - REPL, edit-and-continue
- Documentation
- Static analysis
  - Code metrics, graphs, telemetry, code querying, …
- Metaprogramming
  - Source generators, injectors, weavers…

The Roslyn Compiler API

There should only need to be
one code base in the world
for understanding C#

Language evolution explained

Stagnation

Evolve...

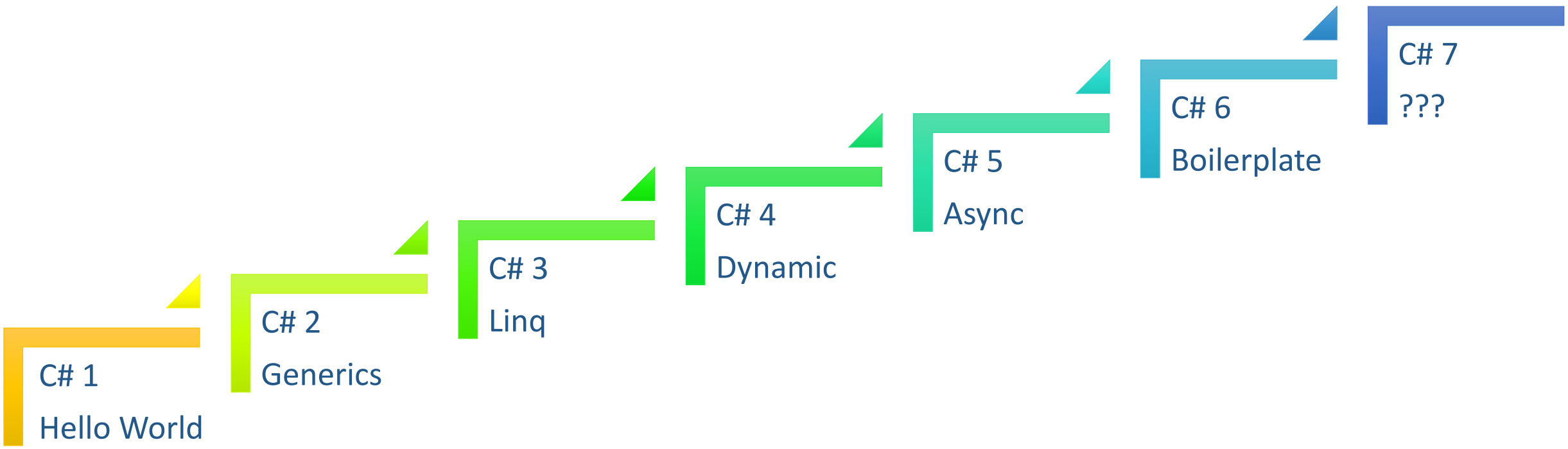
a little

a lot

C#

Explosion

Evolution of C#



Demo: REPL and C# 6

C# “7”

Tuples

```
public (int sum, int count) Tally(IEnumerable<int> values)
{
    var s = 0; var c = 0;
    foreach (var value in values) { s += value; c++; }
    return (s, c);
}

var t = Tally(myValues);
Console.WriteLine($"Sum: {t.sum}, count: {t.count}");

(var s, var c) = Tally(myValues);
Console.WriteLine($"Sum: {s}, count: {c}");
```

Pattern matching

```
if (o is Point p) { WriteLine($"({p.X}, {p.Y})"); }

if (o is Point p && p.X == 5) { WriteLine($"Y: {p.Y}"); }

if (o is Point(5, var y)) { WriteLine($"Y: {y}"); }
```

Patterns in switch statements

```
switch (o)
{
    case int i:
        WriteLine($"Number {i}");
        break;
    case Point(int x, int y):
        WriteLine($"({x},{y})");
        break;
    case string s when s.Length > 0:
        WriteLine(s);
        break;
    case null:
        WriteLine("<null>");
        break;
    default:
        WriteLine("<other>");
        break;
}
```

Records

```
class Person(string First, string Last);
```

```
class Person : IEquatable<Person>
{
    public string First { get; }
    public string Last { get; }

    public Person(string First, string Last) { this.First = First; this.Last = Last; }

    public (string First, string Last) Deconstruct() => (First, Last);

    public bool Equals(Person other) => First == other.First && Last == other.Last;

    public override bool Equals(object obj) => obj is Person other ? Equals(other) : false;
    public override int GetHashCode() => GreatHashFunction(First, Last);

    …
}
```

Creating immutable objects

```
var p1 = new Point { X = 3, Y = 7 };

var p2 = p1 with { X = -p1.X };
```

Nullable and non-nullable reference types

```
string? n; // Nullable reference type
string  s; // Non-nullable reference type

n = null;  // Sure; it's nullable
s = null;  // Warning! Shouldn’t be null!
s = n;     // Warning! Really!

WriteLine(s.Length); // Sure; it’s not null
WriteLine(n.Length); // Warning! Could be null!

if (n != null) { WriteLine(n.Length); } // Sure; you checked
WriteLine(n!.Length); // Ok, if you insist!
```

Custom analyzers

Going deep on Roslyn

Custom analyzers and fixes

- Framework for plugging in to diagnostic reporting and code fixing infrastructure
- Both batch and interactive
- “Code-aware libraries”
  - APIs can ship with analyzers and fixes to guide their users
- Toolboxes
  - Style enforcement, discovery of opportunities

Demo: Custom analyzer

?