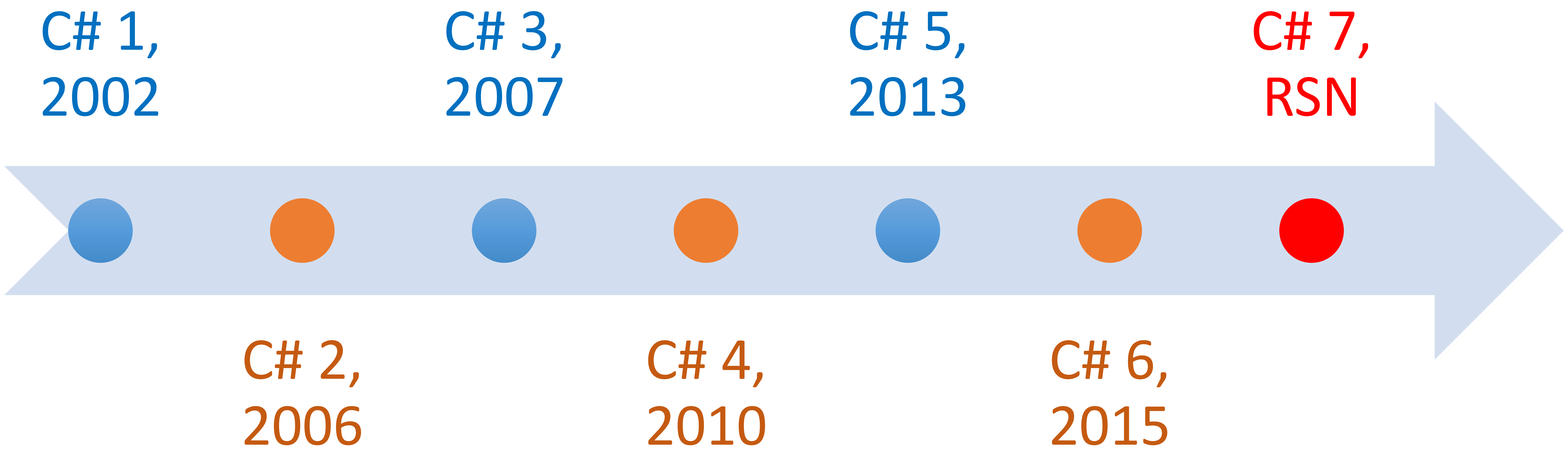
What’s New in C# 6 and C# 7?

Jesse Liberty
@jesseliberty

Microsoft Most Valuable Professional

C# - History



Key Features (C# 6)

Null Conditional

Auto Property

Getter Property

Expression Bodied Function Members

Static Using

String Interpolation

Key Features (C# 7)

Tuples

Pattern Matching

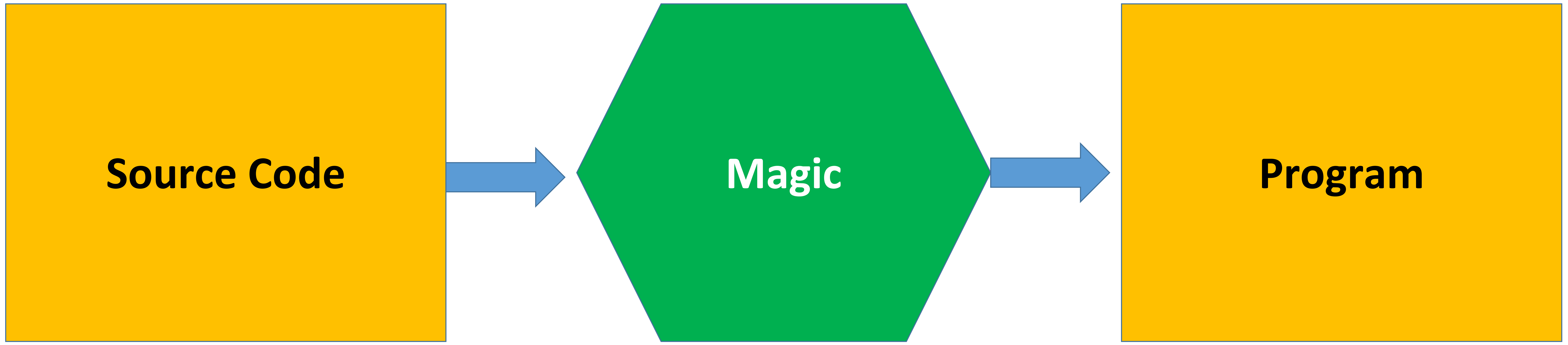
Ref. Returns, Async Returns, Exceptions

Deconstruction

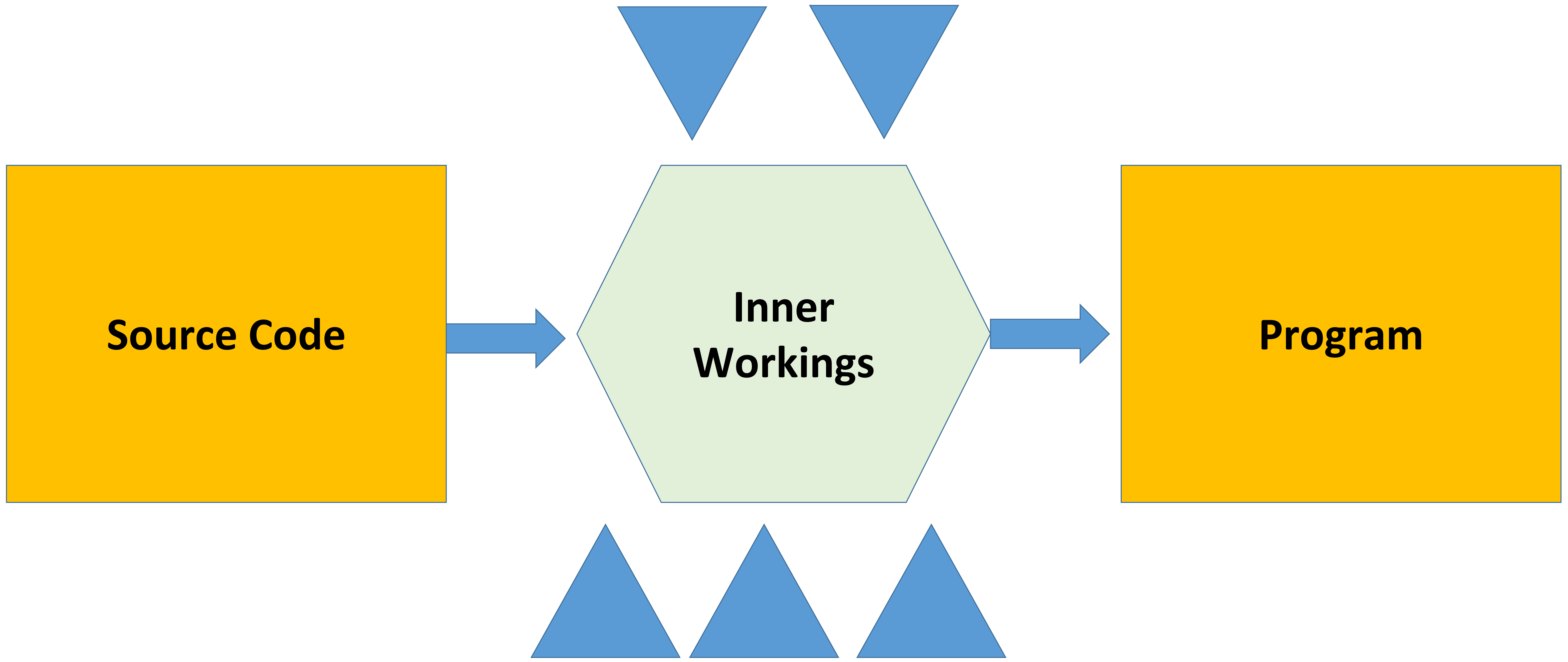
Local Functions

Out Variables Literals

Traditional Understanding…



Transparent Compiler (Roslyn)

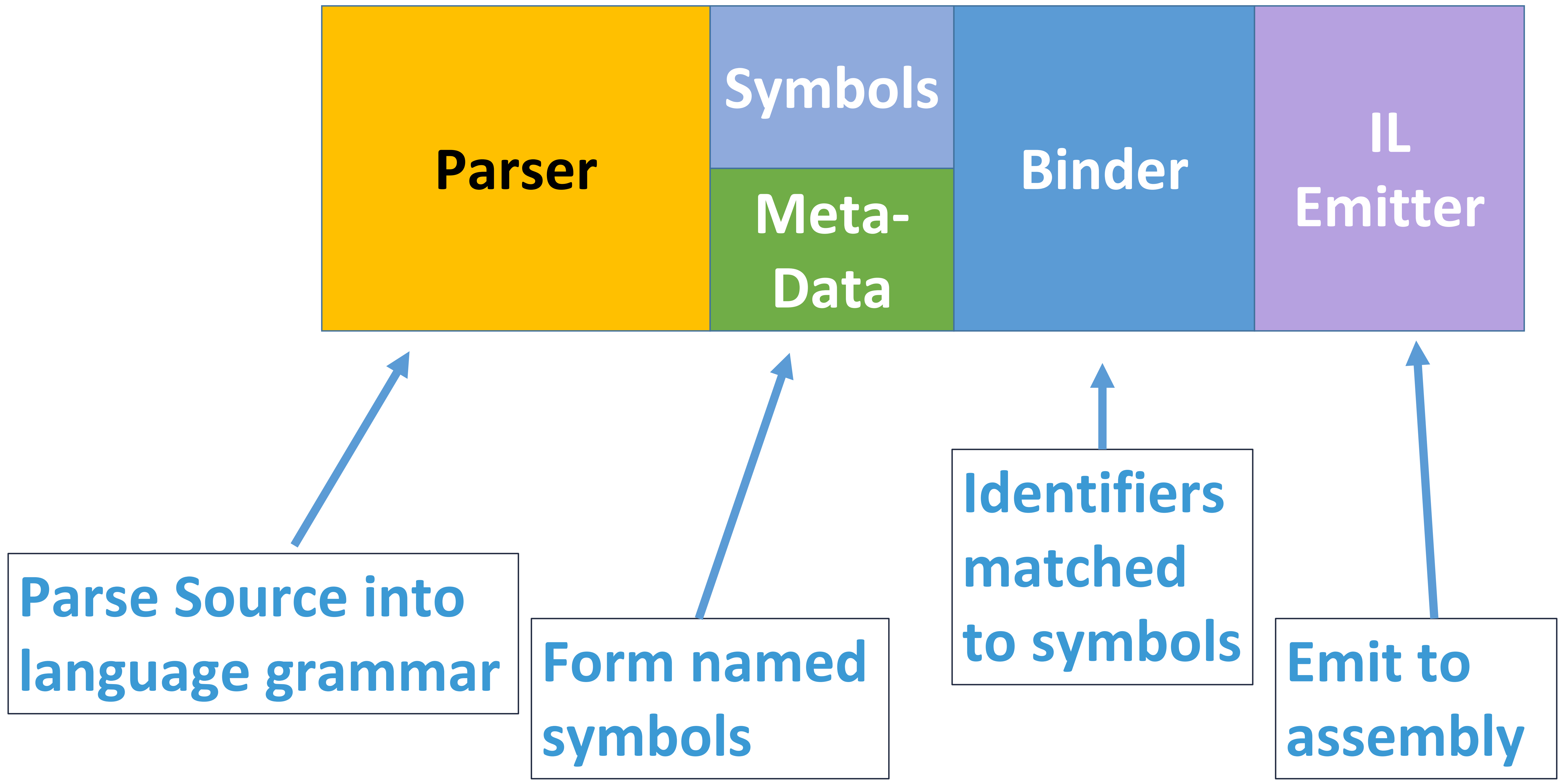


Intellisense, References, Definitions, Analysis, Profilers, Code Generation

Compilers as platforms

- Lower barriers to entry
- Create code-focused tools
- Meta-programming
- Code transformation and generation
- Interactive C#

Compiler Pipeline



Visual Studio Was Re-written in 2013

- Code outling and formatting use syntax tree
- Object browser and navigation use symbol table
- Refactorings and Go To Definition use semantic model

Features

Null Conditional Operator/ Null Coalescing

```
List<string> authors = null;
int? count = authors?.Count;          // count = null

int howMany = authors?.Count ?? 0;  // howMany = 0
```

Auto-Property Declaration & Read-Only

```
public class Person
{
    public string First { get; private set; } = "Jane";
    public string Last { get; private set; } = "Doe";

    public string FirstName{ get; } = "John";
    public string LastName { get; } = "Smith";
}
```

Expression Bodied Function Members

```
public int Add1 (int a, int b)
{
    return a + b;
}

public int Add2 (int a, int b) => a + b;
```

Static Using

```
using static System.Console;
using static System.Math;

class Program
{
    static void Main ()
    {
        WriteLine (Sqrt (3 * 3 + 4 * 4));
    }
}
```

String Interpolation

```
int result = Add (5, 7);

Console.WriteLine("result: {0}”, result);

Console.WriteLine ($"result: {result}");
```

Out Variables

What’s wrong with Out Parameters?

- Not very fluid
- Must declare out variable before callign method
- Cannot use var to declare them

- Solution: out variables

Out Parameters

```
public class Point
{
    int x = 20;
    int y = 50;
    public void GetCoordinates(out int a, out int b)
    {
        a = x;
        b = y;
    }
}

public class Runner
{
     public void PrintCoordinates(Point p)
    {
        int xx, int yy;
        p.GetCoordinates(out xx, out yy);

        Console.WriteLine($"({xx}, {yy})");   // 20, 50
    }
}
```

Out Variables

```
public class Point
{
    int x = 20;
    int y = 50;
    public void GetCoordinates(out int a, out int b)
    {
        a = x;
        b = y;
    }
}

public class Runner
{
     public void PrintCoordinates(Point p)
    {
        p.GetCoordinates(out int xx, out int yy);
        Console.WriteLine($"({xx}, {yy})");   // 20, 50
    }
}
```

Out Variables

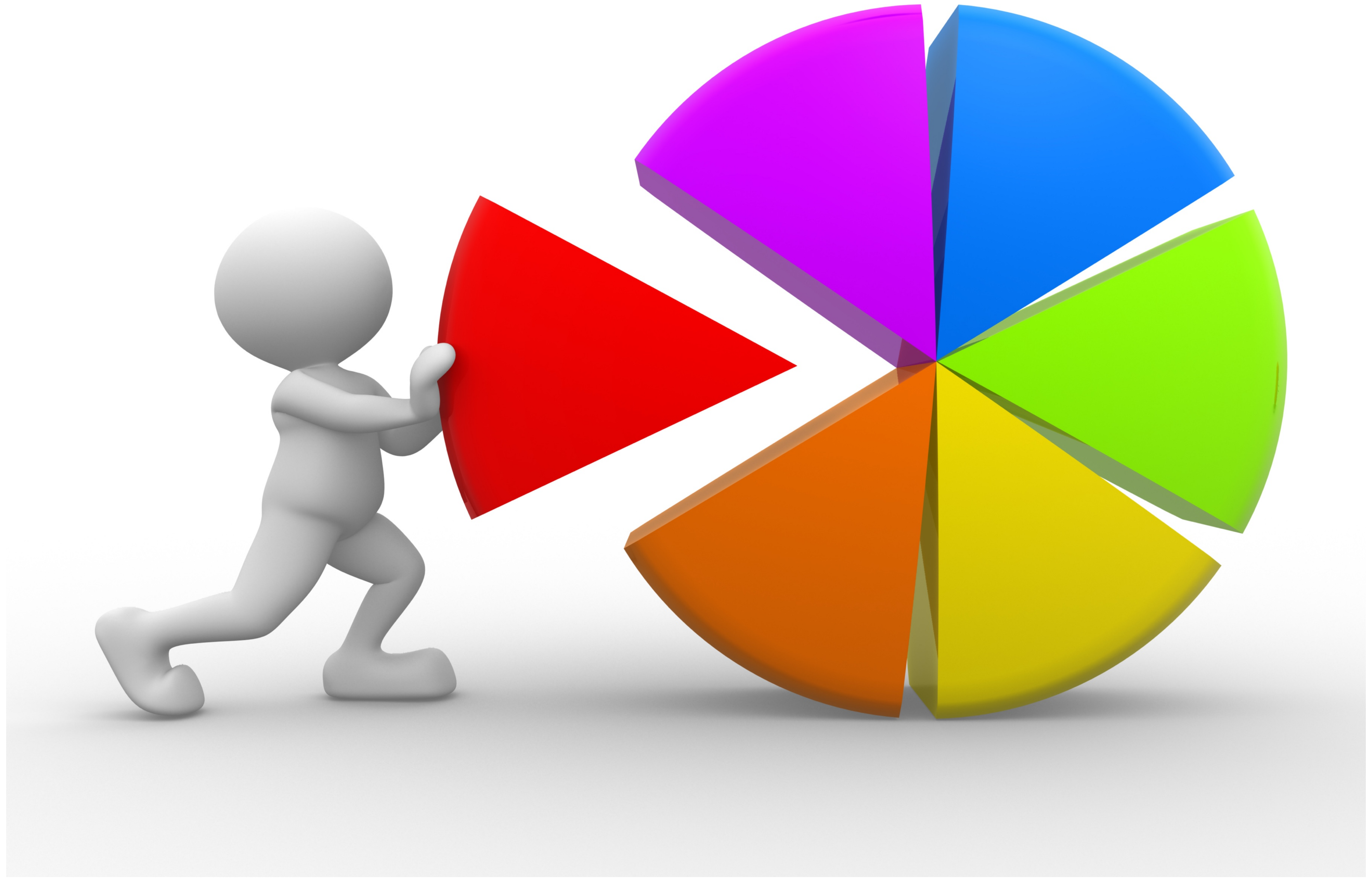
```
public class Point
   {
       int x = 20;
       int y = 50;
      public void GetCoordinates(out int a, out int b)
       {
           a = x;
           b = y;
       }
   }

  public class Runner
   {
       public void PrintCoordinates(Point p)
       {
           p.GetCoordinates(out var xx, out var yy);
           Console.WriteLine($"({xx}, {yy})");   // 20, 50
       }
   }
```

Out Variables

```
public void PrintStars(string s)
{
    if (int.TryParse(s, out var i))
      { Console.WriteLine(new string('*', i)); }
}
```

Pattern Matching



Patterns

- Syntactic elements that can test that a value has a certain “shape”
- Extract information from the value when it has the ”shape” expected
- Three types of patterns
    - Constant patterns of the form c which test that the input is equal to c
    - Type patterns of the form T x which test that the input has type T and extracts the value of x
    - Var patterns of the form var x which always match, and put the value of the input into a fresh variable x

Patterns

- Enhancing two existing constructs:
  - Is expressions can have a pattern on the right hand side, not just types
  - case clauses in switch statements can now match on patterns, not just constants

Patterns

```
public void IsExpressionWithPatterns(object o)
{
    if (o is null) return;

    if ( ! (o is int i)) return;

    Console.WriteLine(new string('*', i));
}
```

Patterns

```
public void UsingPatternsWithTryMethods(object o)
{
    if (o is int i ||
        (o is string s && int.TryParse(s, out i)))
    {
        Console.WriteLine(new String('*', i));
    }
}

UsingPatternsWithTryMethods(5);       // *****
UsingPatternsWithTryMethods("7");     // *******
UsingPatternsWithTryMethods("hello"); // fails
```

Switch Statements with Patterns

- You can switch on any type
- Patterns can be used in case clauses
- Case clauses can have additional conditions!

Switch Statements with Patterns

```
switch (shape)
{
    case Circle c:
    {
        Console.WriteLine($"radius of {c.Radius}");
    }
    break;

     case Square s when s.Side > 50:
      {
        Console.WriteLine("A big square");
      }
      break;
}
```

Tuples

What Problem Are We Trying to Solve?

- Getting more than one value returned from a method
- Out parameters don’t cut it
    - They are clunky
    - They cannot be used with async methods
- System.Tuple<T>
    - verbose and require allocation of tuple object
- Anonymous types returned through dynamic return type
    - High performance overhead
    - No static type checking

Tuple Types and Tuple Literals

- Tuples can be a return type
- Tuples can be a literal such as
  return (firstName, middleInitial, lastName);

Each element in a tuple can be accessed with dot notation
The tuple parts are automatically named Item1, Item2, etc.
You can name the return tuple parts
(string firstName, string middleInitial, string lastName)   GetNames(int id);

Tuple Types and Tuple Literals

- Tuples can be freely converted to other Tuple types
  - There are warnings or errors if you swap the names, etc.
- Tuples are value types
- Tuple elements are public, mutable fields
- Use case: multiple return types
- Use case: dictionary with multiple keys

Tuple Types and Tuple Literals

```
public (string, string, int) LookUpCustomer(int Id)
{
            var first = "Jesse";
            var last = "Liberty";
            var age = 21;

            return (first, last, age);
}

public void Test()
{
   var customer = LookUpCustomer(5);
   Console.WriteLine($"Customer is {customer.Item1}
          {customer.Item2}, who is {customer.Item3} years old");
}
```

Tuple Types and Tuple Literals

```
public (string first, string last, int age) LookUpCustomer(int Id)
{
            var first = "Jesse";
            var last = "Liberty";
            var age = 21;

            return (first, last, age);
}

public void Test()
{
   var customer = LookUpCustomer(5);
   Console.WriteLine($"Customer is {customer.first} {customer.last},
                      who is {customer.age} years old");
}
```

Deconstruction

Consume Tuples Through Deconstruction

- Splits a tuple into new variables
- You can use var for the deconstructing declaration

- **(var first, var middle, var last) = GetName(id);**

- You can even put the var outside the parentheses as shorthand

- **var(first, middle, last) = GetName(id);**

- You can deconstruct into existing variables
- You can use wildcards

```
public void Test()
{
    (string first, string last, int age) = LookUpCustomer(5);
    Console.WriteLine($"Customer name: {first} {last}");
}
public void Test()
{
    (var first, var last, var age) = LookUpCustomer(14);
    Console.WriteLine($"Customer name: {first} {last}");
}
public void Test()
{
    var (first, last, age) = LookUpCustomer(12);
    Console.WriteLine($"Customer name: {first} {last}");
}
```

Local Functions

Local Functions

```
public int Fibonacci(int x)
{
    if (x < 0) throw new ArgumentException();
    return Fib(x).current;

    (int current, int previous) Fib(int i)
    {
        if (i == 0) return (1, 0);
        var (p, pp) = Fib(i - 1);
        Console.WriteLine($"{p}");
        return (p + pp, p);
    }
}
```

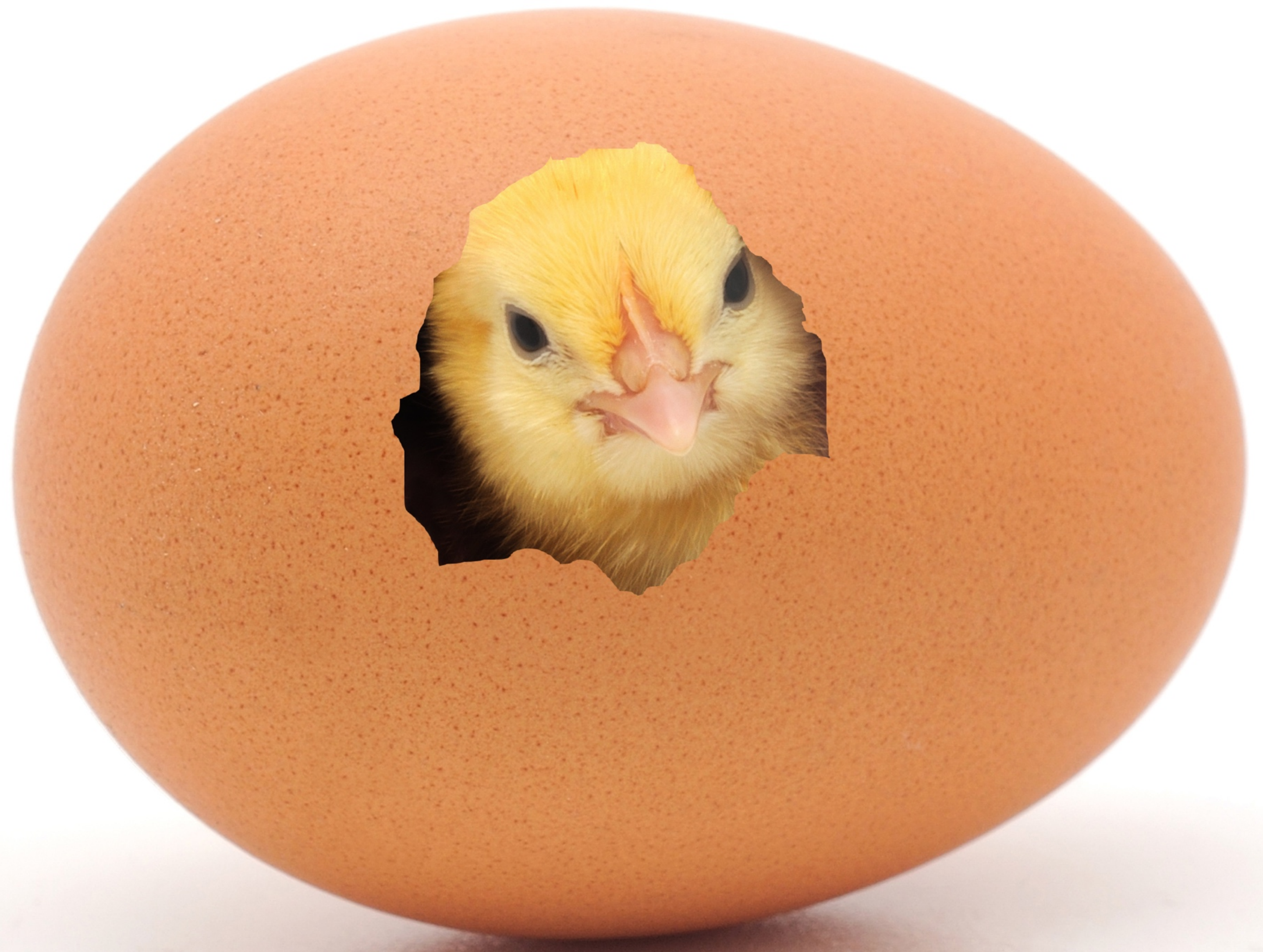
Improvements to Literals

Literals

- You may now use _ between digits (improves readabilty)
- **var bigValue = 1_476_392;**

- You can also specify bit patterns
- **var b = 0b1001_1101_1100_0011;**

Returning By Reference



```
public ref int Changer(int newNumber, int[] numbers)
{
    for (int i = 0; i< numbers.Length; i++)
    {
        if (numbers[i] == newNumber)
        {
            return ref numbers[i];
        }
    }
    throw new IndexOutOfRangeException($"{nameof(newNumber)}
                                                            not found!");
}

public void Test()
{
    int[] array = { 1, 3, 5, 7, 9, 11 };
    Console.WriteLine(array[3]);                  // prints 7
    ref int num = ref Changer(7, array);   // return it
    num = 24;                                       // modify it by reference
    Console.WriteLine(array[3]);                  // prints 24
}
```

Throwing Expressions

```
public class Runner
{
    public string Name { get; }
    public Person (string name) => Name == name
           ?? throw new ArgumentNullException();

    public string GetFirstName()
   {
       var parts = Name.Split(" ");
        return (parts.Length > 0)
           ? parts[0] : throw new InvalidOperationException();
   }

    public string GetLastName() =>
           throw new NotImplementedException();
}
```

Questions?

Merci
Thank You
Gracias
Danke
Grazie
Thank You
Gracias
Merci
Thank You
Danke
Thank You
Grazie
Thank You
Merci
Gracias
Thank You
Danke
Grazie
Merci
Thank You
Danke
Gracias
Thank You
Merci
Grazie