



My Compiler Did What?!?

Mike Harris

Agenda

- Hello World
- Record Type
- Enumerable
- Async / Await
- MoveNext()

Agenda

- **Hello World**
- Record Type
- Enumerable
- Async / Await
- MoveNext()

top level
statement

Hello World

```
using System;

var conference = "That";
Console.WriteLine($"Hello {conference} Conference!");

Action<string> sorry =
    conference => Console.WriteLine(
        $"Sorry, {conference} this is a bit ridiculous.");

sorry(conference);
Closing("fun");

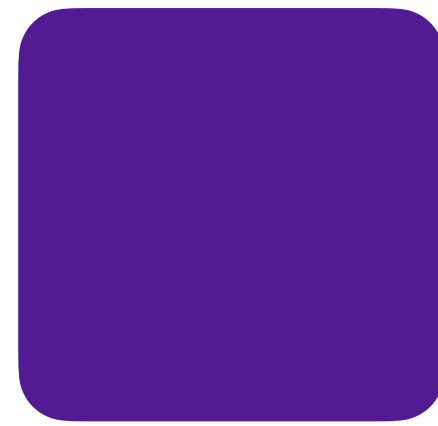
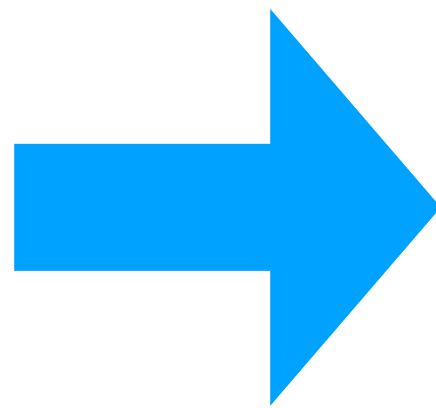
static void Closing(string state)
    => Console.WriteLine($"Hope you find it {state}!");
```

dotnet build

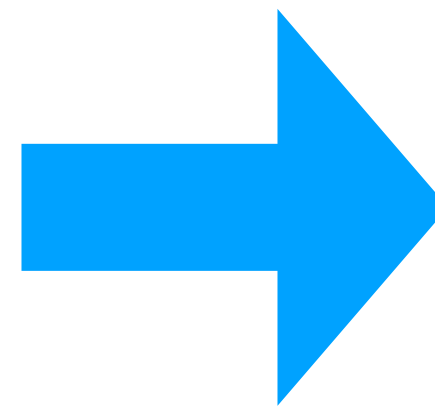
dotnet run



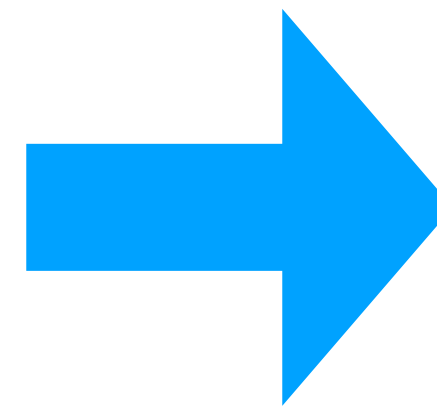
source code



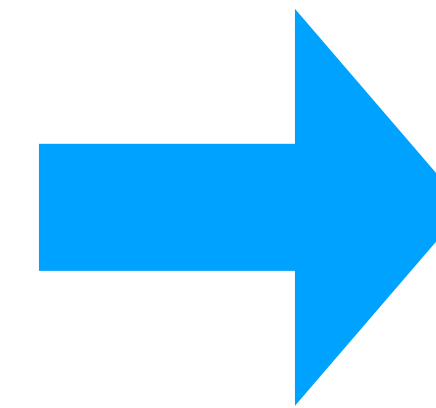
compiler



exe / dll



JIT



machine code

```
using System;

var conference = "That";
Console.WriteLine($"Hello {conference} Conference!");

Action<string> sorry =
    conference => Console.WriteLine(
        $"Sorry, {conference} this is a bit ridiculous.");

sorry(conference);
Closing("fun");

static void Closing(string state)
    => Console.WriteLine($"Hope you find it {state}!");
```

Hello That Conference!
Sorry, That this is a bit ridiculous.
Hope you find it fun!

Hello World

Original

Compiled

```
using System;

var conference = "That";
Console.WriteLine($"Hello {conference} Conference!");

Action<string> sorry =
    conference => Console.WriteLine(
        $"Sorry, {conference} this is a bit ridiculous.");

sorry(conference);
Closing("fun");

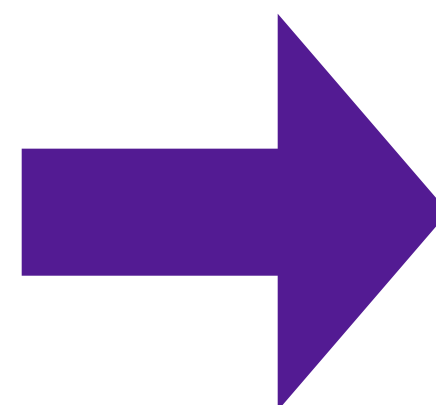
static void Closing(string state)
    => Console.WriteLine($"Hope you find it {state}!");
```

```
using System;

internal class Program
{
    private static void Main (string[] args)
    {
        string text = "That";
        Console.WriteLine ("Hello " + text + " Conference!");
        ((Action<string>)delegate (string conference) {
            Console.WriteLine ("Sorry, " + conference + " this i
        }) (text);
        Closing ("fun");
        static void Closing (string state)
        {
            Console.WriteLine ("Hope you find it " + state + "!"
        }
    }
}
```



source code



exe / dll

C#

Hello World

Original

```
using System;
```

top level
statement

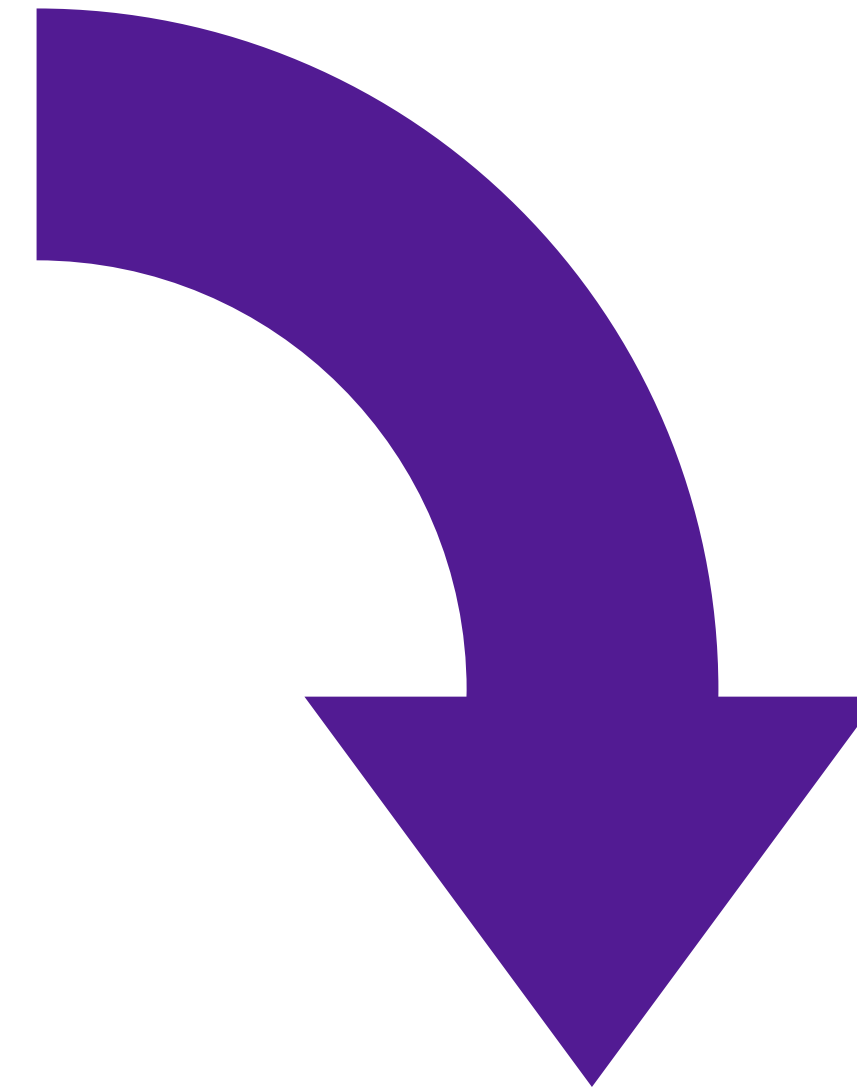
Compiled

```
using System;  
  
internal class Program  
{  
    private static void Main (string[] args)  
    {  
  
    }  
}
```

Hello World

Original

```
var conference = "That";  
  
Action<string> sorry =  
    conference => Console.WriteLine(  
        $"Sorry, {conference} this is a bit ridiculous.");
```



Compiled

```
string text = "That";  
  
((Action<string>)delegate (string conference) {  
    Console.WriteLine ("Sorry, " + conference + " this is a bit ridiculous.");  
}) (text);
```


Hello World

Original

```
Console.WriteLine($"Hello {conference} Conference!");
```

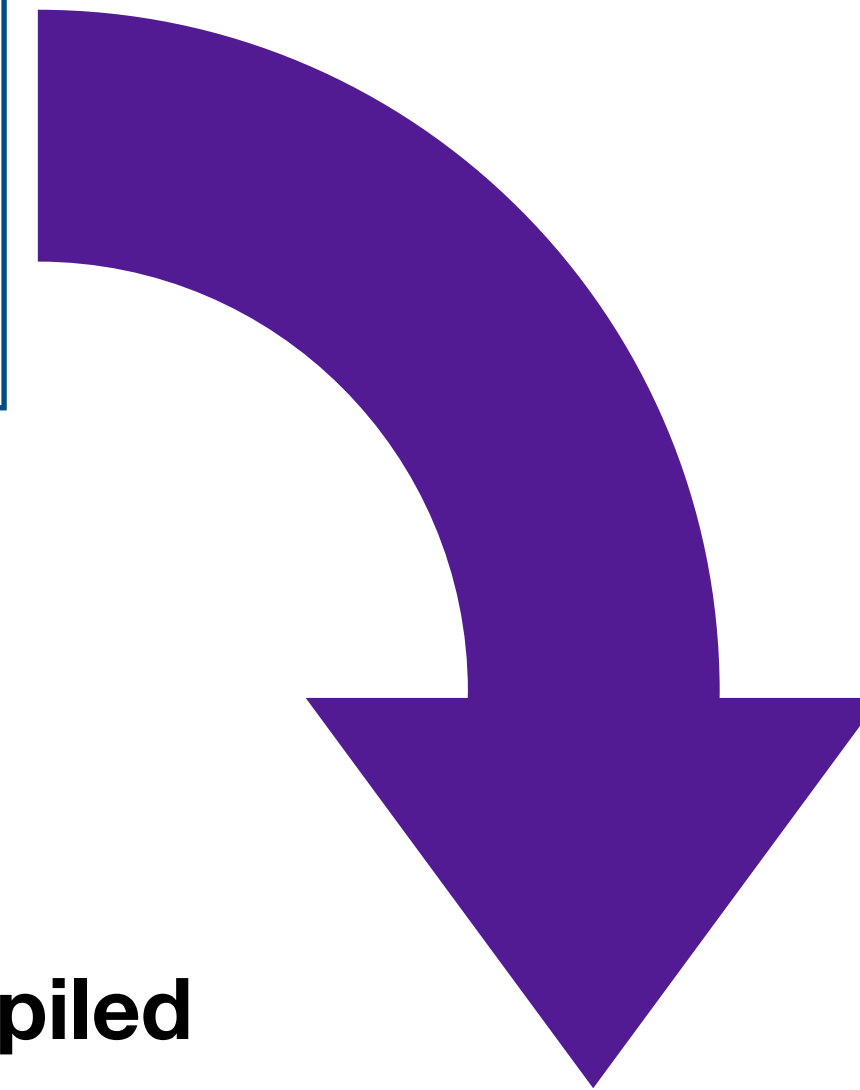
Compiled

```
Console.WriteLine ("Hello " + text + " Conference!");
```

Hello World

Original

```
Closing( "fun" );  
  
static void Closing(string state)  
    => Console.WriteLine($"Hope you find it {state}!");
```



Compiled

```
Closing ( "fun" );  
static void Closing (string state)  
{  
    Console.WriteLine ( "Hope you find it " + state + "!" );  
}
```

Agenda

- Hello World
- **Record Type**
- Enumerable
- Async / Await
- MoveNext()

Record Type

```
using System;

namespace App
{
    public record PersonRecord(string FirstName, string LastName);

    public class PersonClass
    {
        public PersonClass(string first, string last)
            => (FirstName, LastName) = (first, last);

        public string FirstName { get; init; }
        public string LastName { get; init; }
    }
}
```

Record Type

```
Console.WriteLine("Class record.");

var mikeRecord = new PersonRecord("Mike", "Harris");
var otherMikeRecord = new PersonRecord("Mike", "Harris");

Console.WriteLine($"\\tmikeRecord={mikeRecord}");
Console.WriteLine($"\\tHello {mikeRecord.FirstName}!");

if (mikeRecord == otherMikeRecord)
{
    Console.WriteLine($"\\tSame old {otherMikeRecord.FirstName}.");
}
else
{
    Console.WriteLine($"\\tYou have changed {mikeRecord.FirstName}.");
}
```

Class

```
Console.WriteLine("Class example.");

var mikeClass = new PersonClass("Mike", "Harris");
var otherMikeClass = new PersonClass("Mike", "Harris");

Console.WriteLine($"\\tmikeClass={mikeClass}");
Console.WriteLine($"\\tHello {mikeClass.FirstName}!");

if (mikeClass == otherMikeClass)
{
    Console.WriteLine($"\\tSame old {otherMikeClass.FirstName}.");
}
else
{
    Console.WriteLine($"\\tYou have changed {mikeClass.FirstName}.");
}
```

Record Type / Class

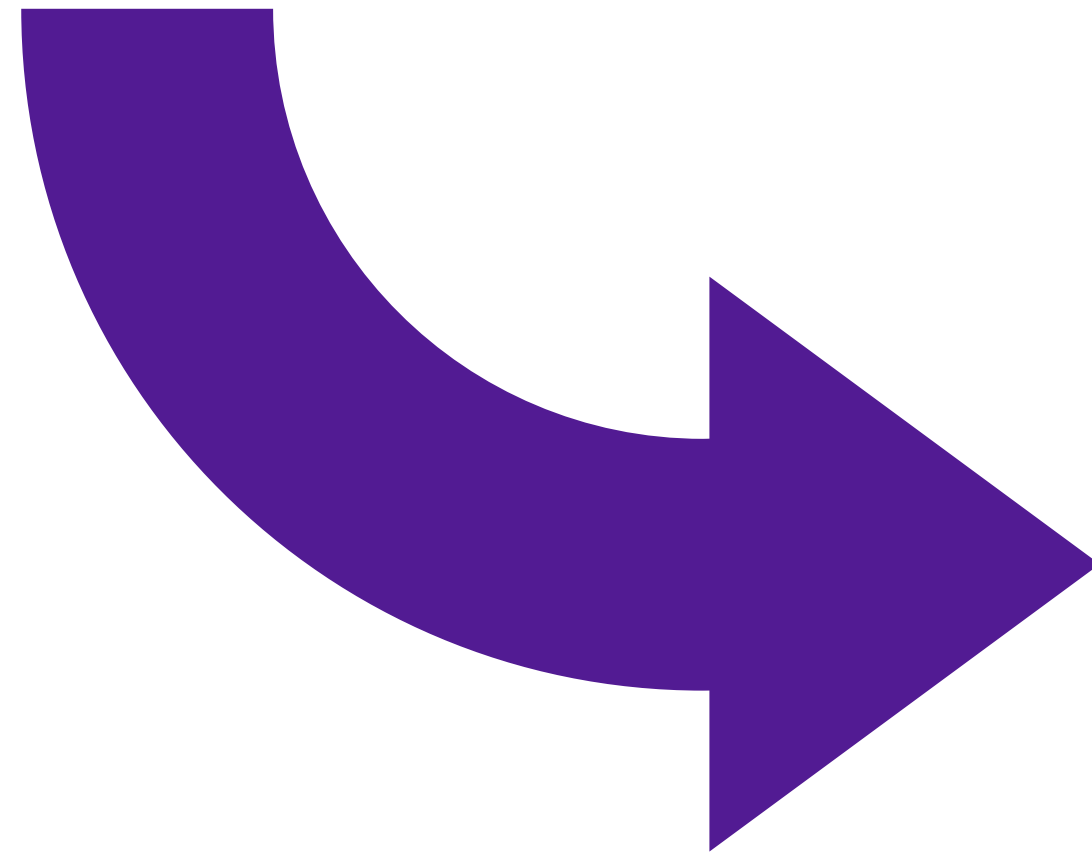
output

```
Class record.  
mikeRecord=PersonRecord { FirstName = Mike, LastName = Harris }  
Hello Mike!  
Same old Mike.  
Class example.  
mikeClass=App.PersonClass  
Hello Mike!  
You have changed Mike.
```

Class

Original

```
public class PersonClass
{
    public PersonClass(string first, string last)
        => (FirstName, LastName) = (first, last);
    public string FirstName { get; init; }
    public string LastName { get; init; }
}
```



Compiled

```
public class PersonClass
{
    public string FirstName {
        get;
        set;
    }

    public string LastName {
        get;
        set;
    }

    public PersonClass (string first, string last)
    {
        string text2 = FirstName = first;
        text2 = (LastName = last);
    }
}
```




















Record Type

Original

```
public record PersonRecord(string FirstName, string LastName);
```

Record Type

Compiled

```
▼  PersonRecord
  ►  Base Types
     .ctor(PersonRecord)
     .ctor(String, String)
     <Clone>$() : PersonRecord
     <FirstName>k__BackingField : String
     <LastName>k__BackingField : String
     Deconstruct(String&, String&) : Void
     EqualityContract : Type
     Equals(PersonRecord) : Boolean
     Equals(Object) : Boolean
     FirstName : String
     GetHashCode() : Int32
     LastName : String
     op_Equality(PersonRecord, PersonRecord) : Boolean
     op_Inequality(PersonRecord, PersonRecord) : Boolean
     PrintMembers(StringBuilder) : Boolean
     ToString() : String
```

Record Type

Compiled

```
public class PersonRecord : IEquatable<PersonRecord>
{
    protected virtual Type EqualityContract {
        [System.Runtime.CompilerServices.NullableContext (1)]
        [CompilerGenerated]
        get {
            return typeof(PersonRecord);
        }
    }
}
```

Record Type

Compiled

```
public PersonRecord (string FirstName, string LastName)
{
    this.FirstName = FirstName;
    this.LastName = LastName;
    base._002Ector ();
}

protected PersonRecord (PersonRecord original)
{
    FirstName = original.FirstName;
    LastName = original.LastName;
}
```

used by Clone

Record Type

Compiled

```
public void Deconstruct (out string FirstName, out string LastName)
{
    FirstName = this.FirstName;
    LastName = this.LastName;
}
```

Record Type

Compiled

```
public override string ToString ()
{
    StringBuilder stringBuilder = new StringBuilder ();
    stringBuilder.Append ("PersonRecord");
    stringBuilder.Append (" { ");
    if (PrintMembers (stringBuilder)) {
        stringBuilder.Append (" ");
    }
    stringBuilder.Append ("}");
    return stringBuilder.ToString ();
}
```

Record Type

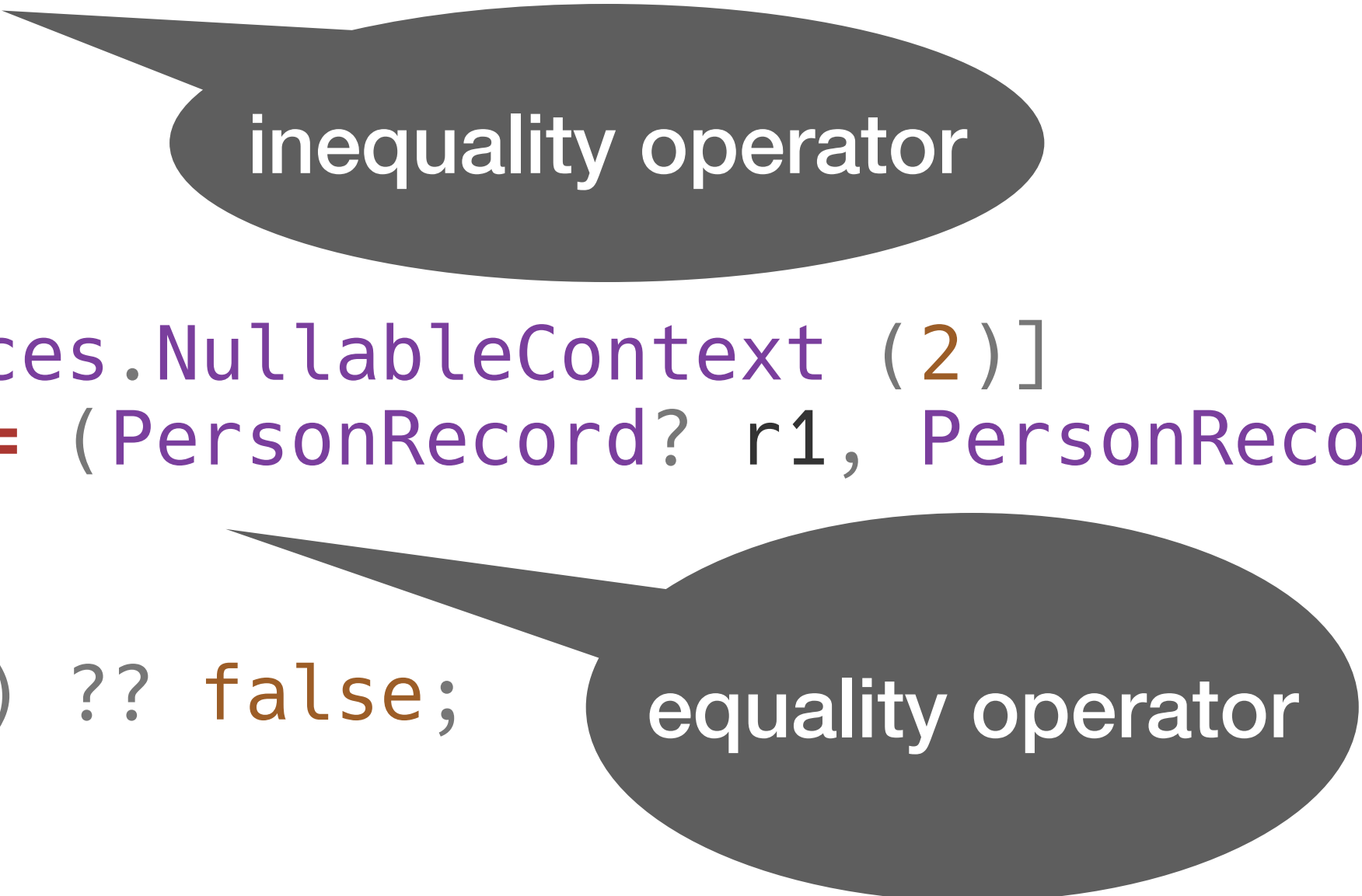
Compiled

```
protected virtual bool PrintMembers (StringBuilder builder)
{
    builder.Append ( "FirstName" );
    builder.Append ( " = " );
    builder.Append ( (object?)FirstName );
    builder.Append ( ", " );
    builder.Append ( "LastName" );
    builder.Append ( " = " );
    builder.Append ( (object?)LastName );
    return true;
}
```

Record Type

Compiled

```
[System.Runtime.CompilerServices.NullableContext (2)]  
public static bool operator != (PersonRecord? r1, PersonRecord? r2)  
{  
    return !(r1 == r2);  
}  
  
[System.Runtime.CompilerServices.NullableContext (2)]  
public static bool operator == (PersonRecord? r1, PersonRecord? r2)  
{  
    if ((object)r1 != r2) {  
        return r1?.Equals (r2) ?? false;  
    }  
    return true;  
}
```




Record Type

Compiled

```
public override bool Equals (object? obj)
{
    return Equals (obj as PersonRecord);
}

public virtual bool Equals (PersonRecord? other)
{
    if ((object)other != null && EqualityContract == other!.EqualityContract &&
    EqualityComparer<string>.Default.Equals (FirstName, other!.FirstName))
    {
        return EqualityComparer<string>.Default.Equals (LastName, other!.LastName);
    }
    return false;
}
```



value equality

Record Type

Compiled

```
public override int GetHashCode ()  
{  
    return (EqualityComparer<Type>.Default.GetHashCode (EqualityContract) * -1521134295 +  
    EqualityComparer<string>.Default.GetHashCode (FirstName)) * -1521134295 +  
    EqualityComparer<string>.Default.GetHashCode (LastName);  
}
```

Record Type

Compiled

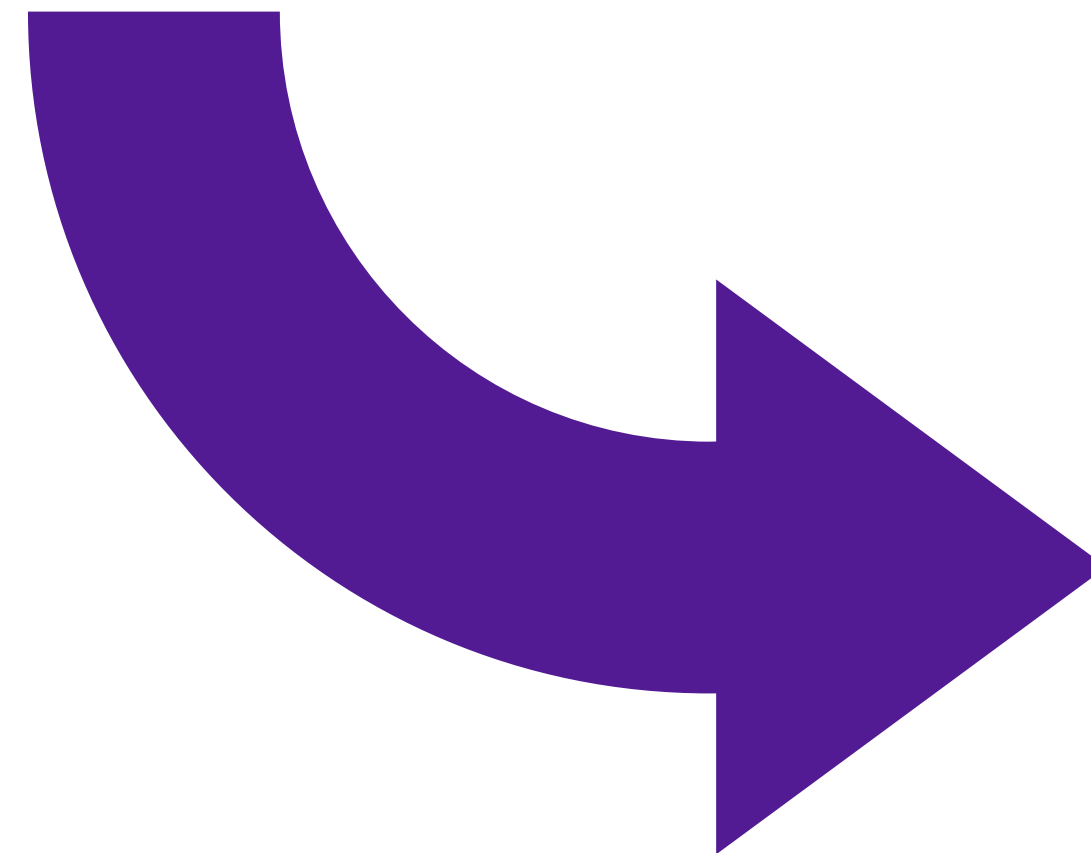
```
public virtual PersonRecord _003CClone_003E_0024 ( )  
{  
    return new PersonRecord (this);  
}
```

deep Clone

Record Type

Original

```
public record PersonRecord(  
    string FirstName, string LastName);
```



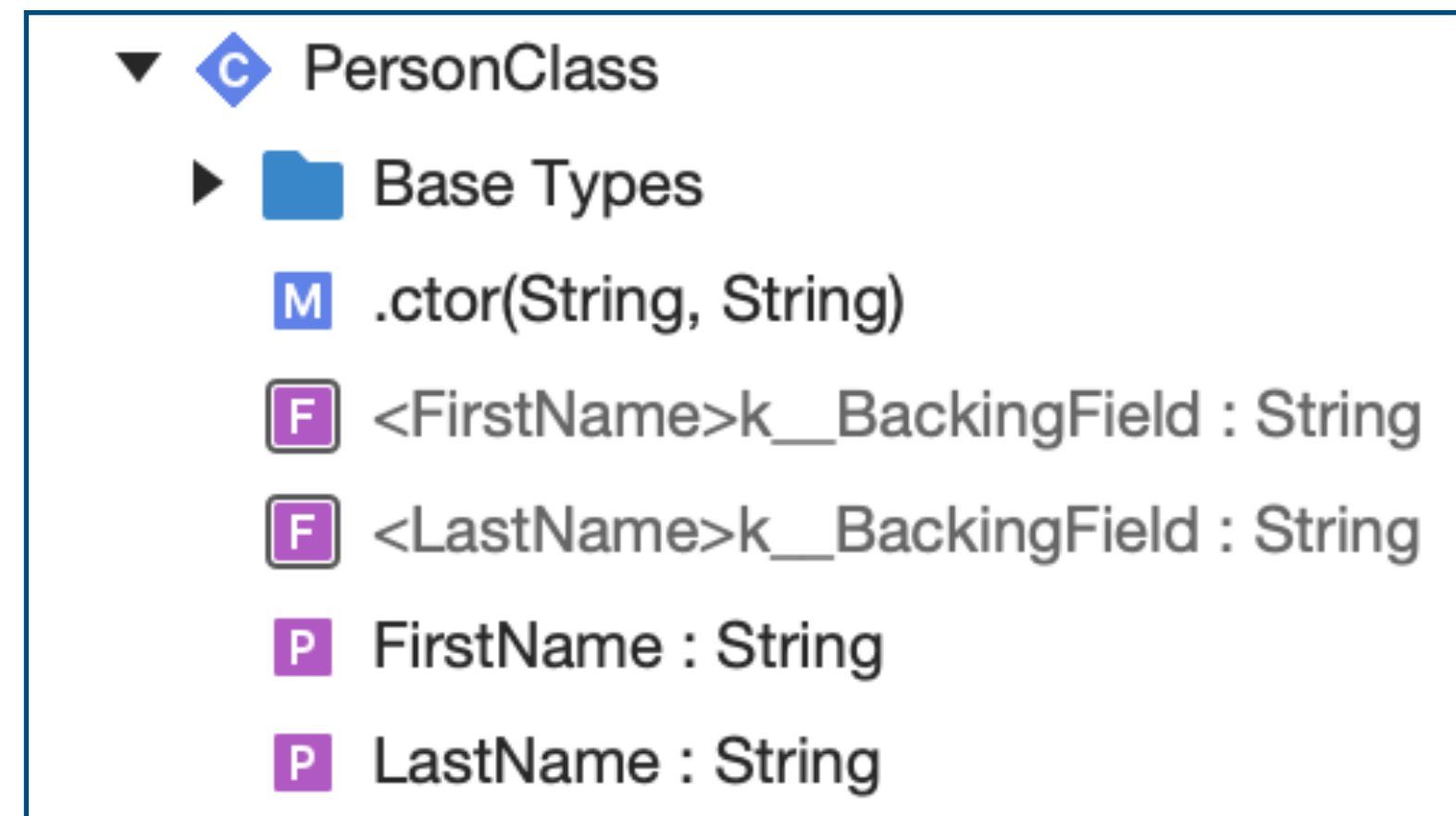
Compiled

```
▼ C PersonRecord  
  ► Base Types  
    M .ctor(PersonRecord)  
    M .ctor(String, String)  
    M <Clone>$() : PersonRecord  
    F <FirstName>k__BackingField : String  
    F <LastName>k__BackingField : String  
    M Deconstruct(String&, String&) : Void  
    P EqualityContract : Type  
    M Equals(PersonRecord) : Boolean  
    M Equals(Object) : Boolean  
    P FirstName : String  
    M GetHashCode() : Int32  
    P LastName : String  
    M op_Equality(PersonRecord, PersonRecord) : Boolean  
    M op_Inequality(PersonRecord, PersonRecord) : Boolean  
    M PrintMembers(StringBuilder) : Boolean  
    M ToString() : String
```

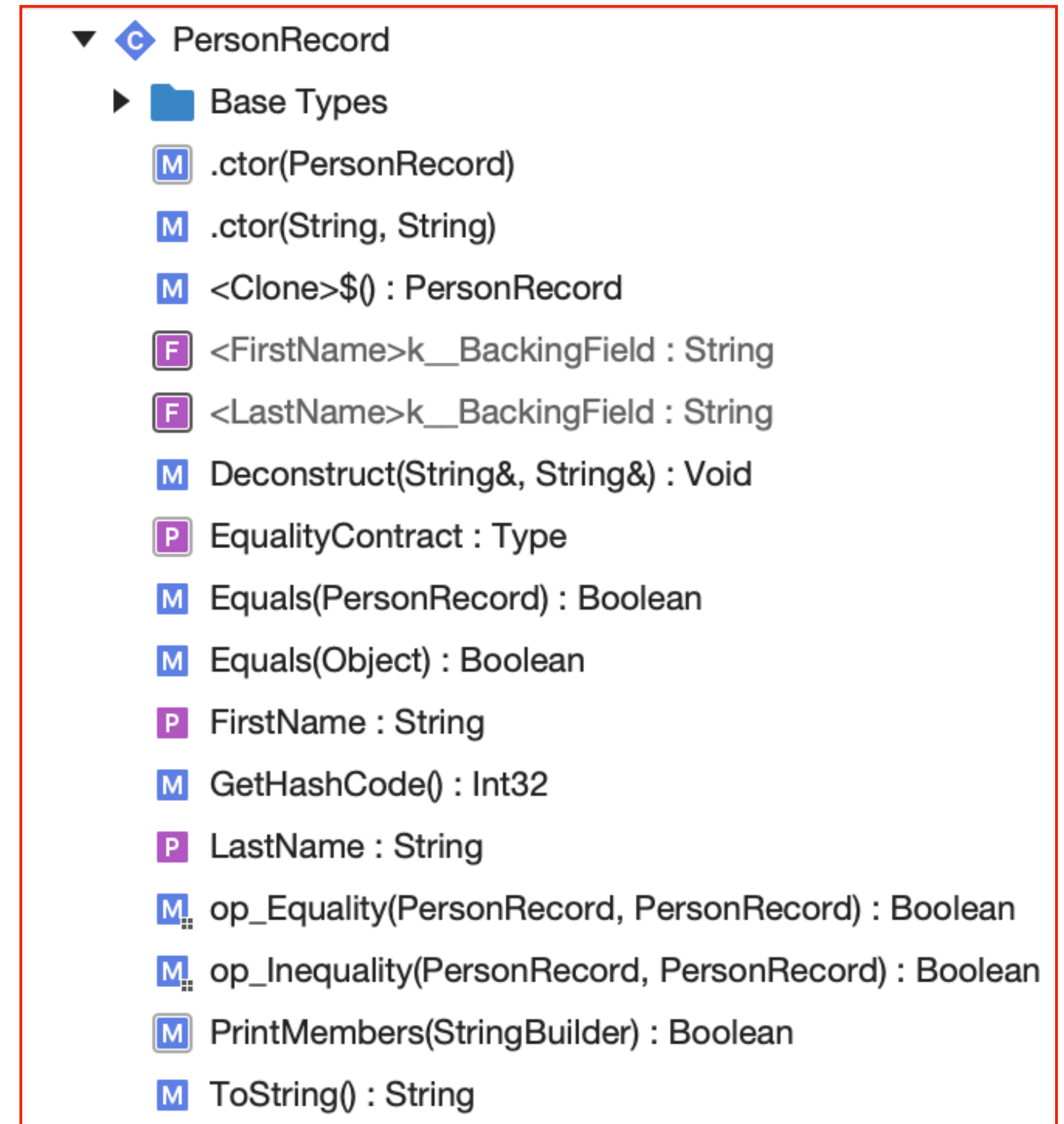
Record Type

Record Type

Class



Compare



Agenda

- Hello World
- Record Type
- **Enumerable**
- Async / Await
- MoveNext()

Enumerable

```
using System;
using System.Collections.Generic;
using System.Linq;

foreach(var n in Fibonacci().Take(10))
{
    Console.Write($"{n}, ");
}
Console.WriteLine($"{Fibonacci().ElementAt(10)}");
```

Enumerable

```
static IEnumerable<int> Fibonacci( )  
{  
    yield return 0;  
  
    int value = 1;  
    int next = 1;  
    while (true)  
    {  
        yield return value;  
  
        int t = value;  
        value = next;  
        next += t;  
    }  
}
```

output

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55

Enumerable

Original

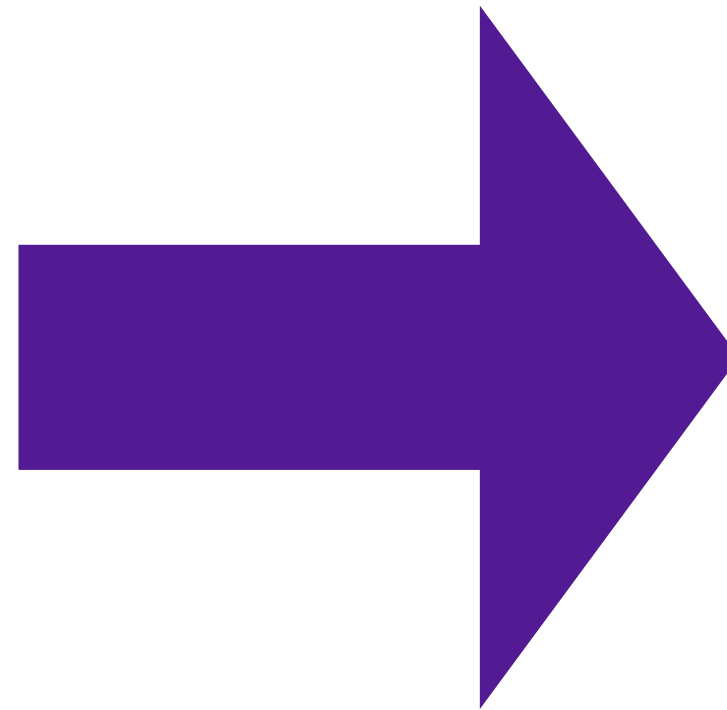
```
using System;
using System.Collections.Generic;
using System.Linq;

foreach(var n in Fibonacci().Take(10))
{
    Console.Write($"{n}, ");
}
Console.WriteLine($"{Fibonacci().ElementAt(10)}");

static IEnumerable<int> Fibonacci()
{
    yield return 0;

    int value = 1;
    int next = 1;
    while (true)
    {
        yield return value;

        int t = value;
        value = next;
        next += t;
    }
}
```




Compiled















```
using System;
using System.Collections.Generic;
using System.Linq;

private static void _003CMain_003E_0024 (string[] args)
{
    foreach (int item in Fibonacci ().Take (10)) {
        Console.Write ("{" + item + "}, ");
    }
    Console.WriteLine ("{" + Fibonacci ().ElementAt (10) + "}");
    static IEnumerable<int> Fibonacci ()
    {
        yield return 0;
        int value = 1;
        int next = 1;
        while (true) {
            yield return value;
            int num = value;
            value = next;
            next += num;
        }
    }
}
```

Enumerable

Compiled

▼  <Program>\$.<<<Main>\$>g__Fibonacci|0_0>d

- ▶  Base Types
 -  .ctor(Int32)
 -  <>1__state : Int32
 -  <>2__current : Int32
 -  <>l__initialThreadId : Int32
 -  <next>5__3 : Int32
 -  <value>5__2 : Int32
 -  MoveNext() : Boolean
 -  System.Collections.Generic.IEnumerable<System.Int32>.GetEnumerator() : IEnumerator<Int32>
 -  System.Collections.Generic.IEnumerator<System.Int32>.Current : Int32
 -  System.Collections.IEnumerable.GetEnumerator() : IEnumerator
 -  System.Collections.IEnumerator.Current : Object
 -  System.Collections.IEnumerator.Reset() : Void
 -  System.IDisposable.Dispose() : Void

Enumerable

Compiled

start state

```
var sequence = new EnumerableFibonacci(-2);  
foreach (var n in sequence.Take(10))  
{  
    Console.Write($"{n}, ");  
}  
Console.WriteLine($"{sequence.ElementAt(10)}");
```

Enumerable

hidden class

Compiled

```
public class EnumerableFibonacci : IEnumerable<int>, IEnumerable, IEnumerator<int>,
IEnumerator, IDisposable
{
    private int _state;
    private int _current;
    private int _initialThreadId;

    private int value;
    private int next;
```

enumerable
variables

local variables

Enumerable

Compiled

```
public EnumerableFibonacci(int state)
{
    this._state = state;
    _initialThreadId = Environment.CurrentManagedThreadId;
}
```

Enumerable

Compiled

```
IEnumerator<int> IEnumerable<int>.GetEnumerator( )  
{  
    if ( _state == -2 && _initialThreadId == Environment.CurrentManagedThreadId )  
    {  
        _state = 0;  
        return this;  
    }  
    return new EnumerableFibonacci(0);  
}
```

Enumerable

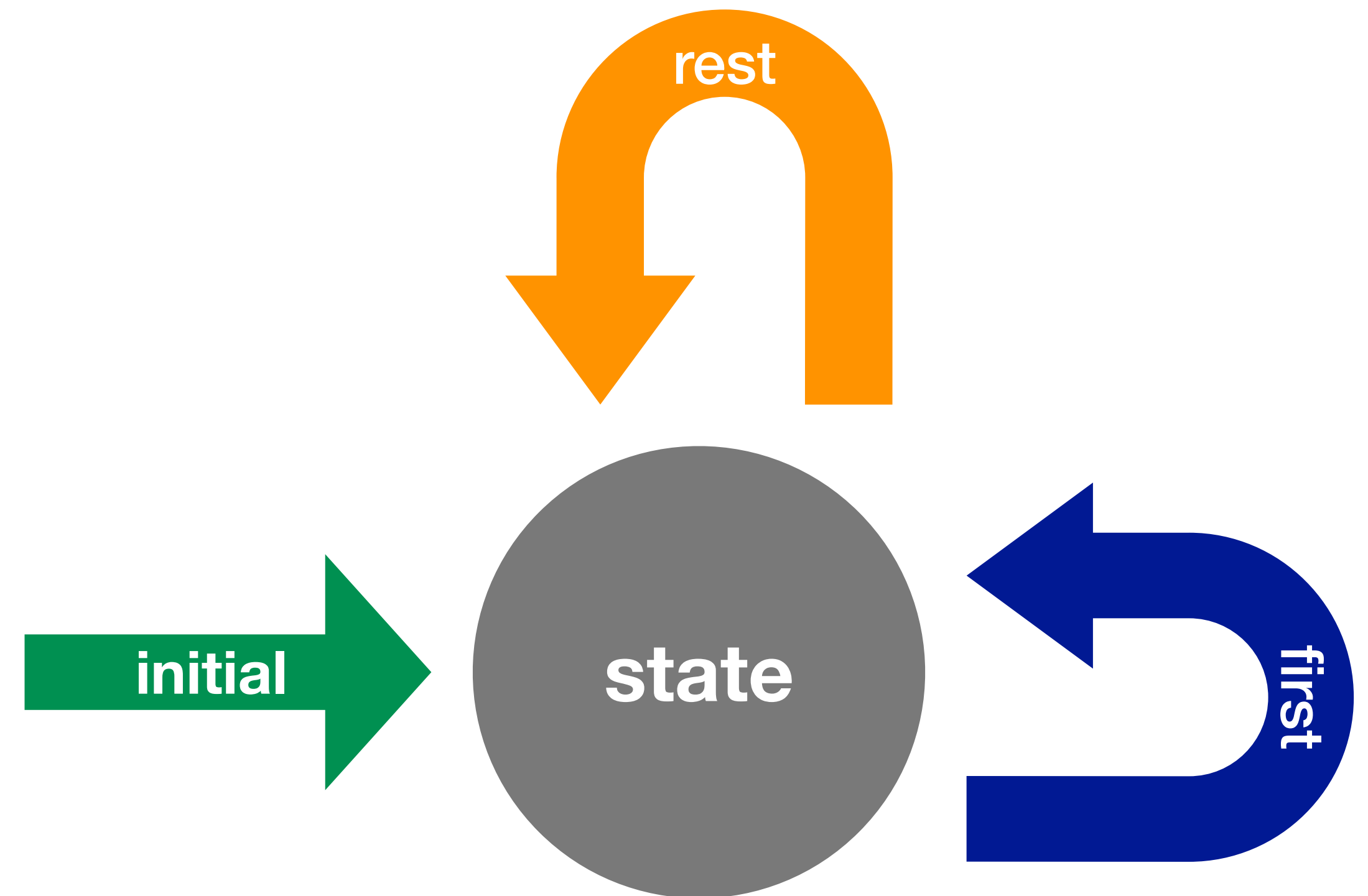
Compiled

```
int IEnumerator<int>.Current  
{  
    get  
    {  
        return _current;  
    }  
}
```

Enumerable

Compiled

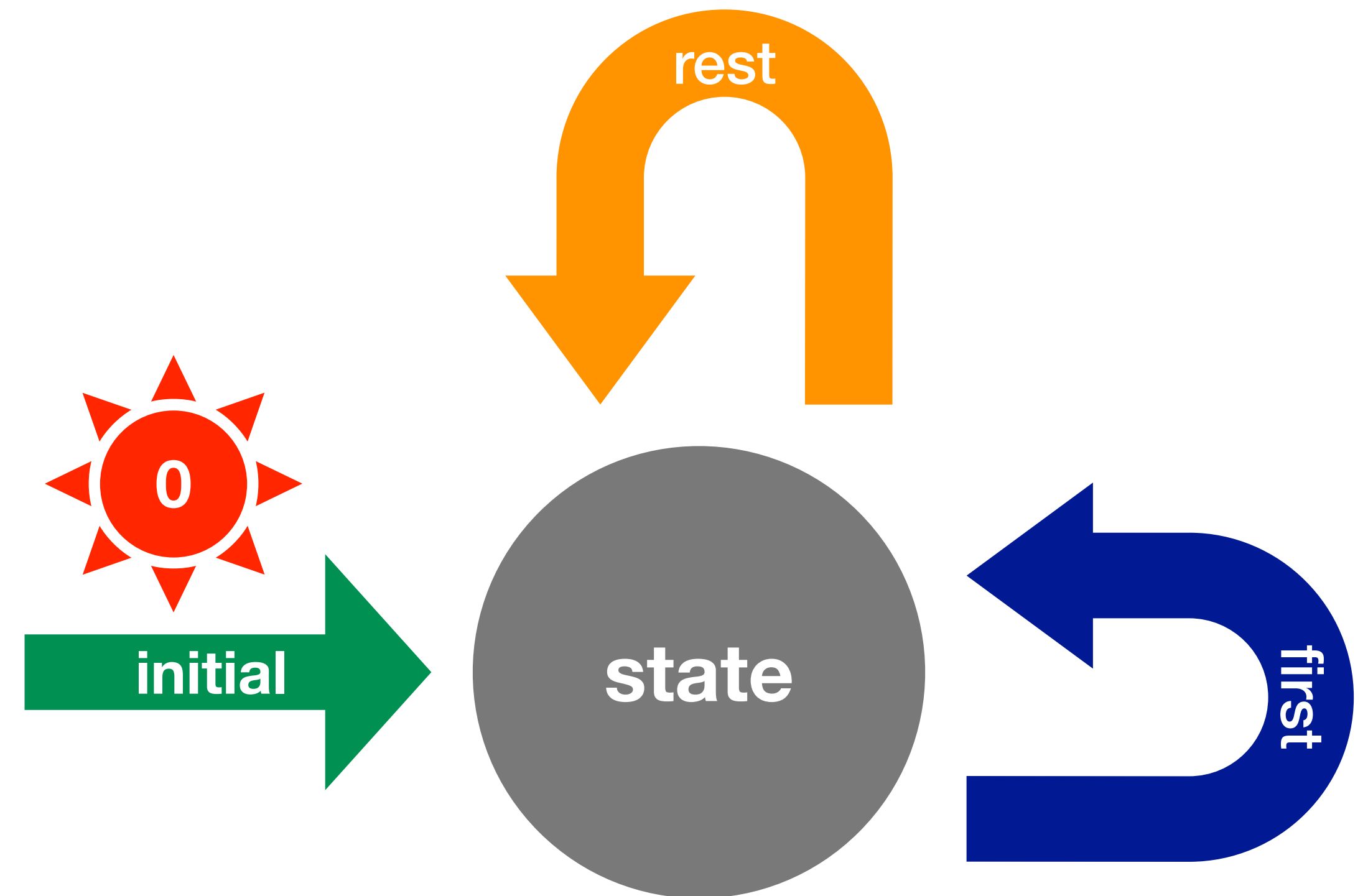
```
private bool MoveNext()  
{  
    const int ERROR = -1;  
    switch (_state)  
    {  
        default:  
            return false;  
        case 0:  
            // initial  
            _state = ERROR;  
            _current = 0; // Fibonacci(0)  
            _state = 1;  
            return true;  
        case 1:  
            // 1st  
            _state = ERROR;  
            value = 1; // Fibonacci(1)  
            next = 1; // Fibonacci(2)  
            break;  
        case 2:  
            // rest  
            _state = ERROR;  
            int temp = value;  
            value = next;  
            next += temp;  
            break;  
    }  
    _current = value;  
    _state = 2;  
    return true;  
}
```



Enumerable

Compiled

```
private bool MoveNext()
{
    const int ERROR = -1;
    switch (_state)
    {
        default:
            return false;
        case 0:
            // initial
            _state = ERROR;
            _current = 0; // Fibonacci(0)
            _state = 1;
            return true;
        case 1:
            // 1st
            _state = ERROR;
            value = 1; // Fibonacci(1)
            next = 1; // Fibonacci(2)
            break;
        case 2:
            // rest
            _state = ERROR;
            int temp = value;
            value = next;
            next += temp;
            break;
    }
    _current = value;
    _state = 2;
    return true;
}
```

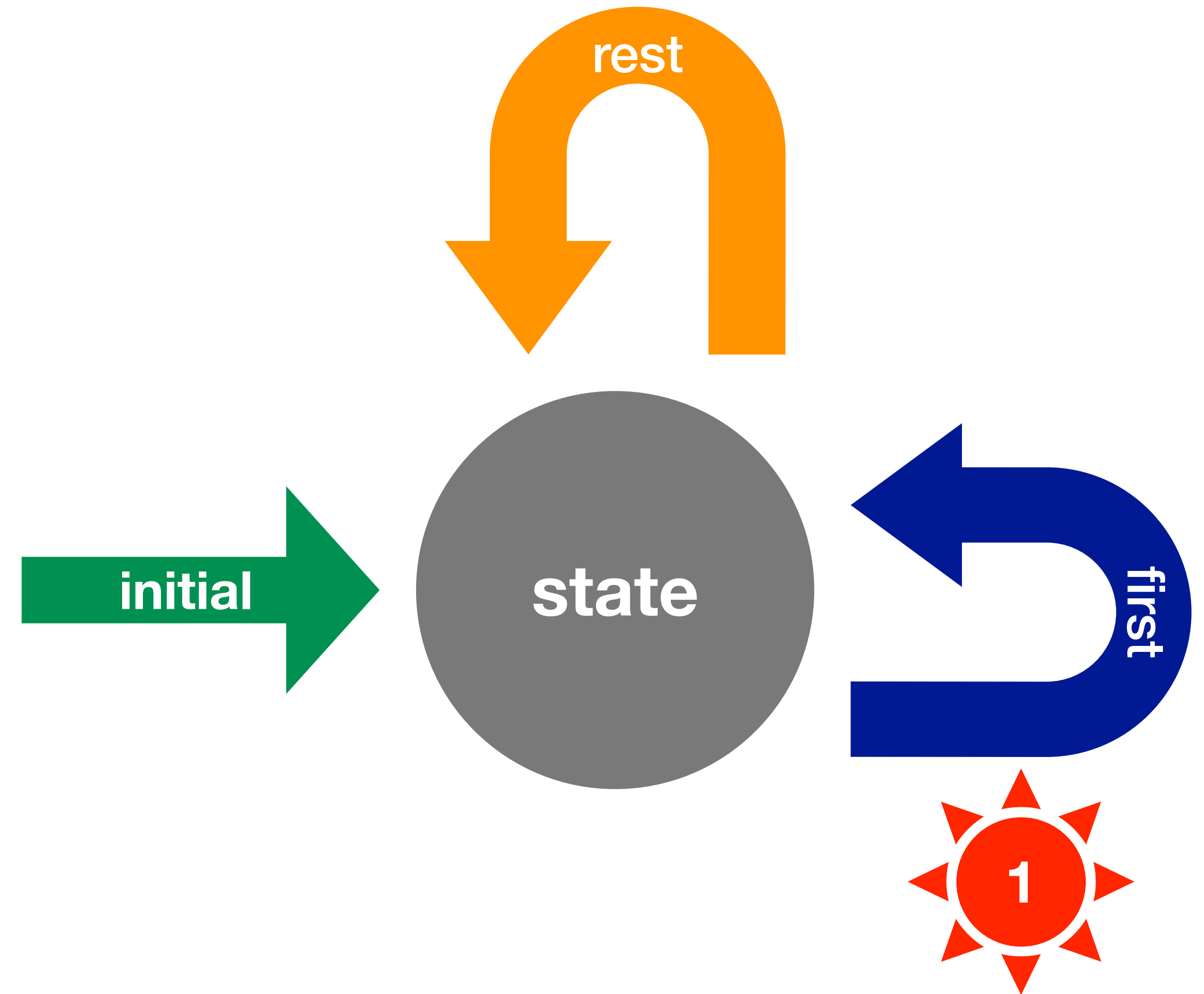


simplified C#

Compiled

Enumerable

```
private bool MoveNext()  
{  
    const int ERROR = -1;  
    switch (_state)  
    {  
        default:  
            return false;  
        case 0:  
            // initial  
            _state = ERROR;  
            _current = 0; // Fibonacci(0)  
            _state = 1;  
            return true;  
        case 1:  
            // 1st  
            _state = ERROR;  
            value = 1; // Fibonacci(1)  
            next = 1; // Fibonacci(2)  
            break;  
        case 2:  
            // rest  
            _state = ERROR;  
            int temp = value;  
            value = next;  
            next += temp;  
            break;  
    }  
    _current = value;  
    _state = 2;  
    return true;  
}
```

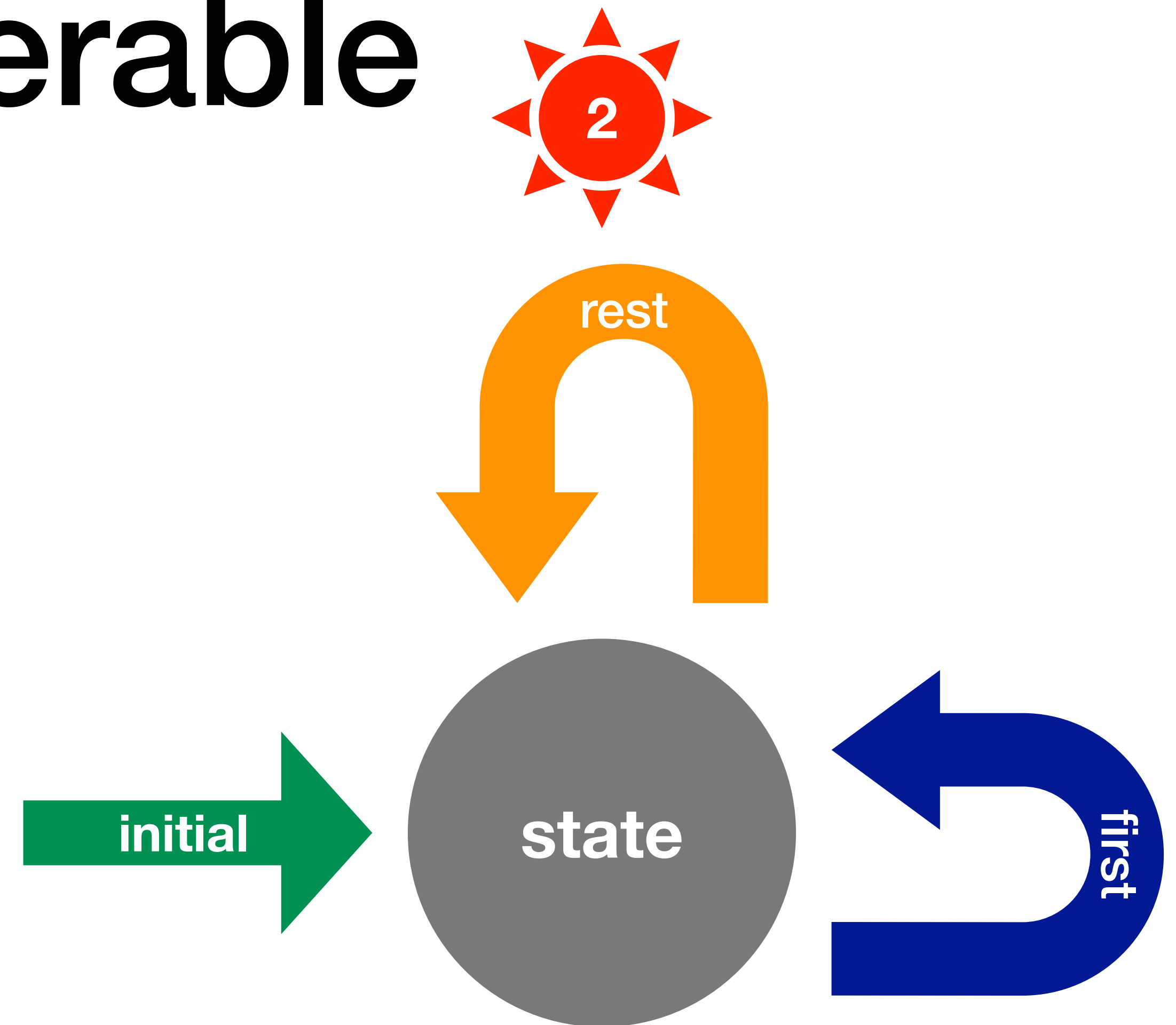


simplified C#

Compiled

```
private bool MoveNext()  
{  
    const int ERROR = -1;  
    switch (_state)  
    {  
        default:  
            return false;  
        case 0:  
            // initial  
            _state = ERROR;  
            _current = 0; // Fibonacci(0)  
            _state = 1;  
            return true;  
        case 1:  
            // 1st  
            _state = ERROR;  
            value = 1; // Fibonacci(1)  
            next = 1; // Fibonacci(2)  
            break;  
        case 2:  
            // rest  
            _state = ERROR;  
            int temp = value;  
            value = next;  
            next += temp;  
            break;  
    }  
    _current = value;  
    _state = 2;  
    return true;  
}
```

Enumerable



Enumerable

Main

foreach

Enumerable

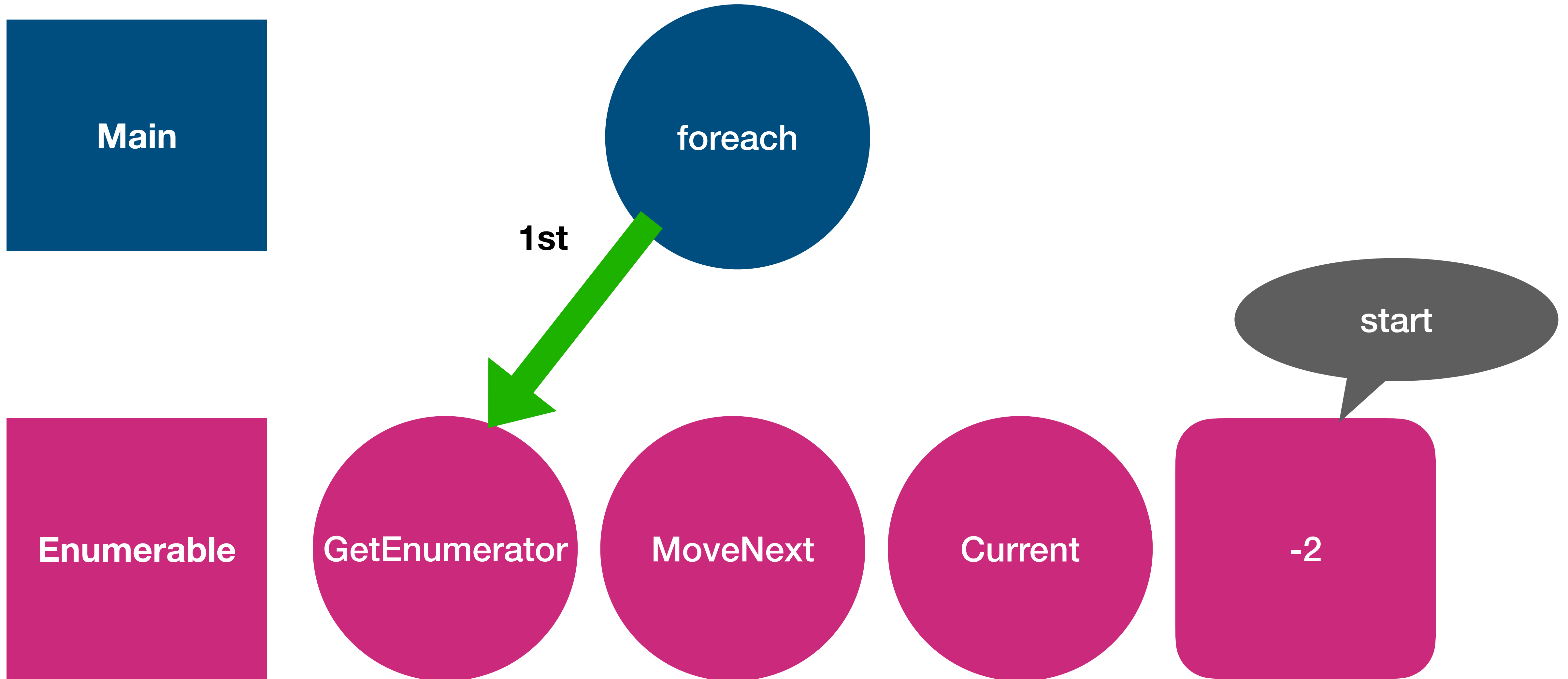
GetEnumerator

MoveNext

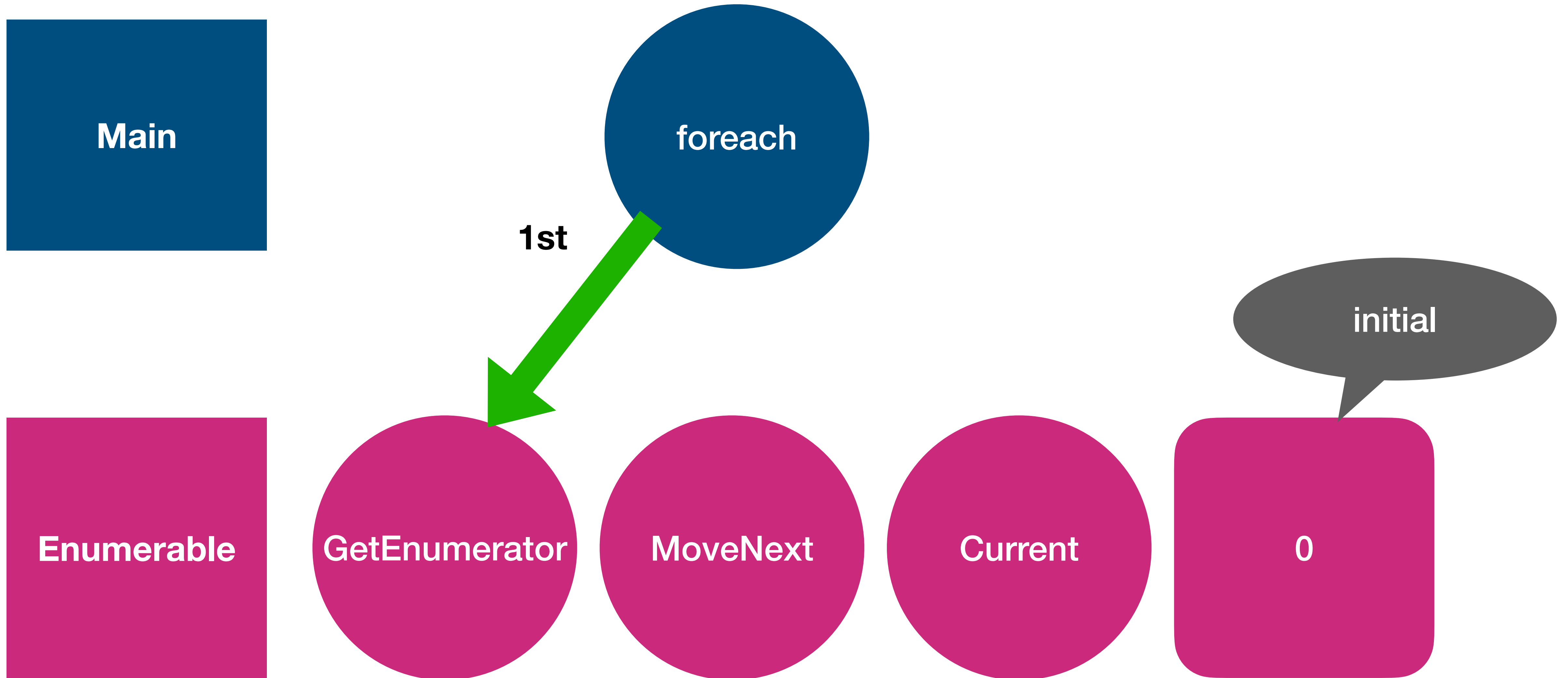
Current

_state

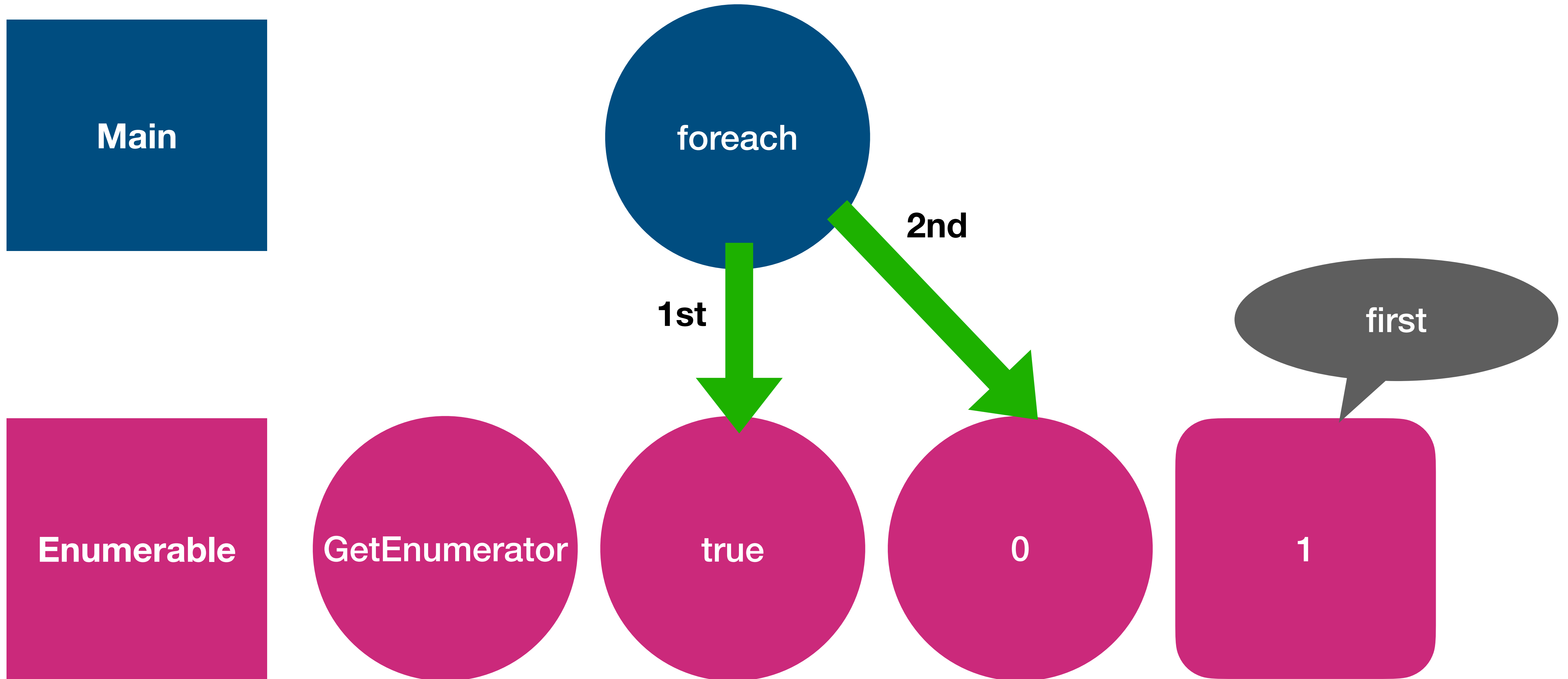
Enumerable



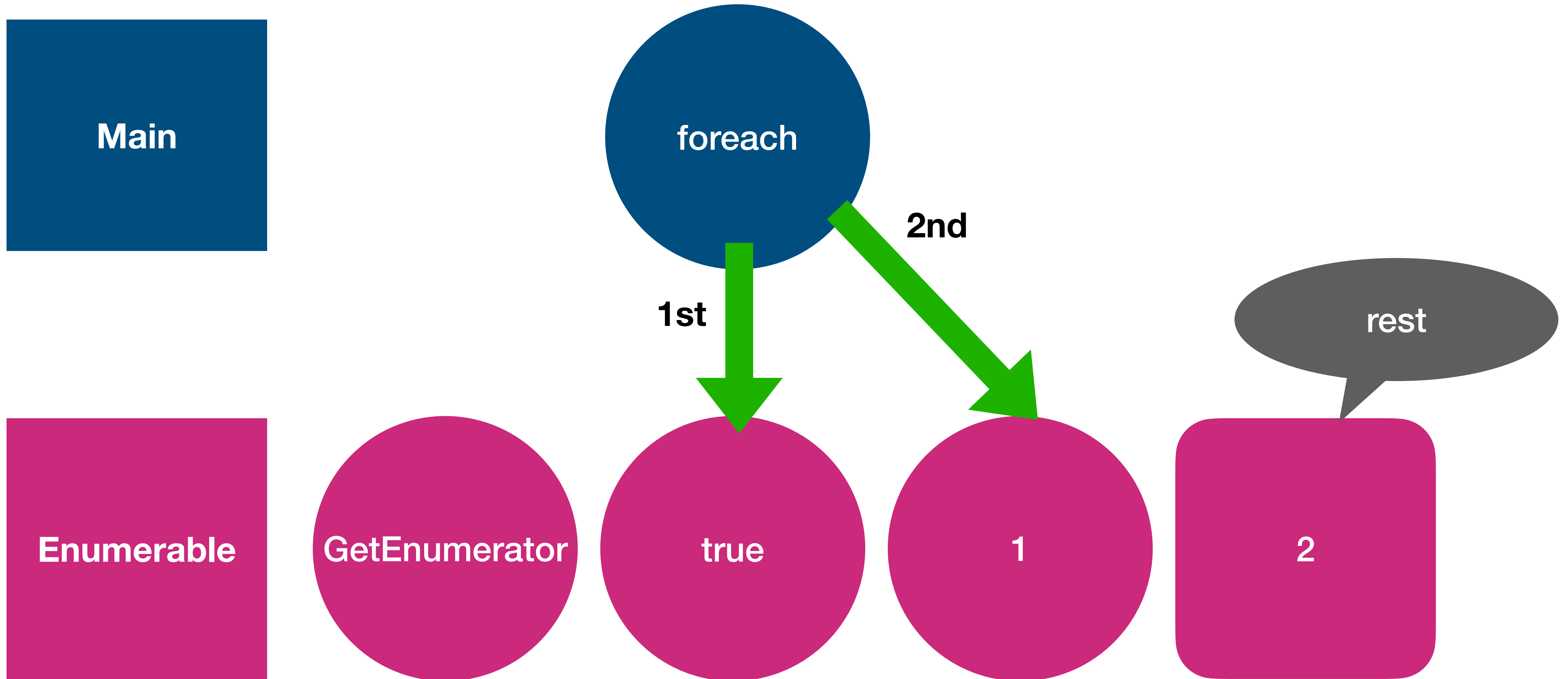
Enumerable



Enumerable



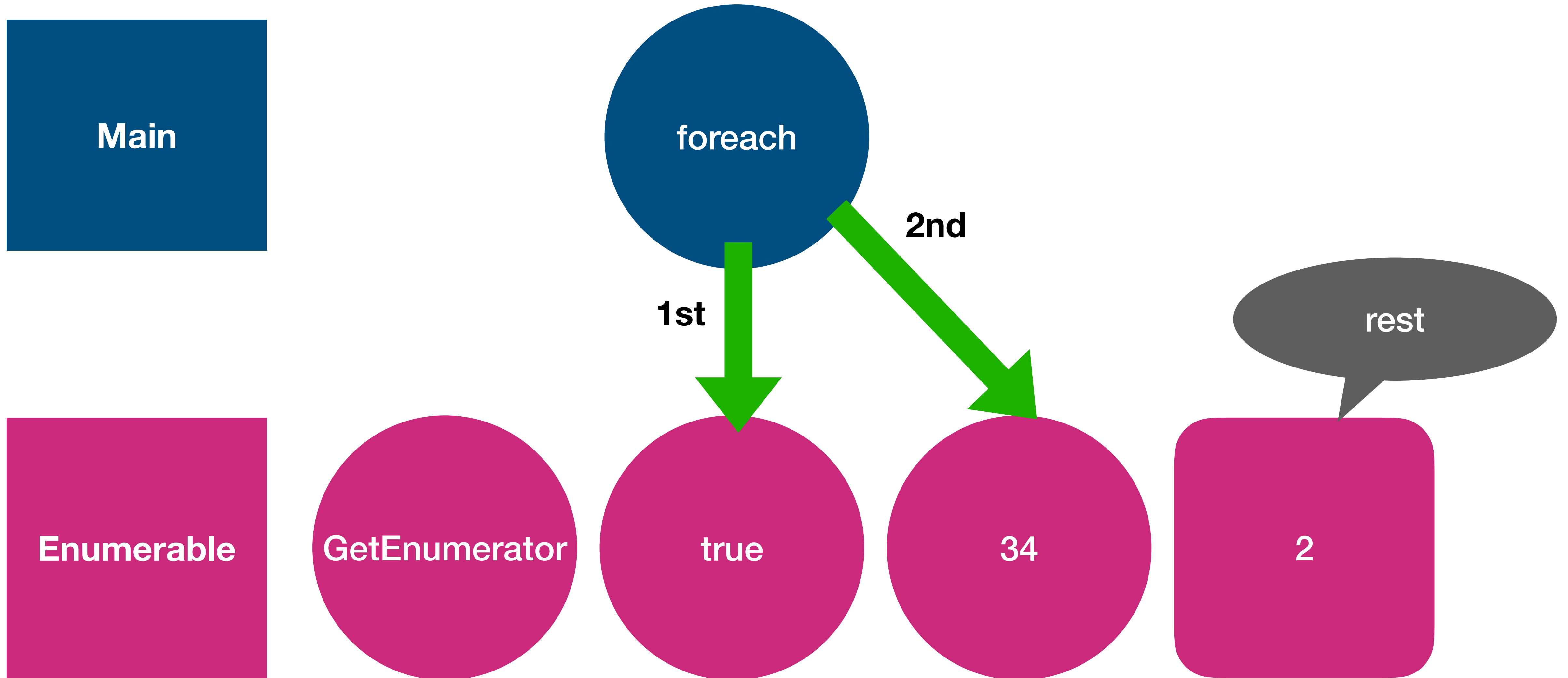
Enumerable



Enumerable

skip to the end...

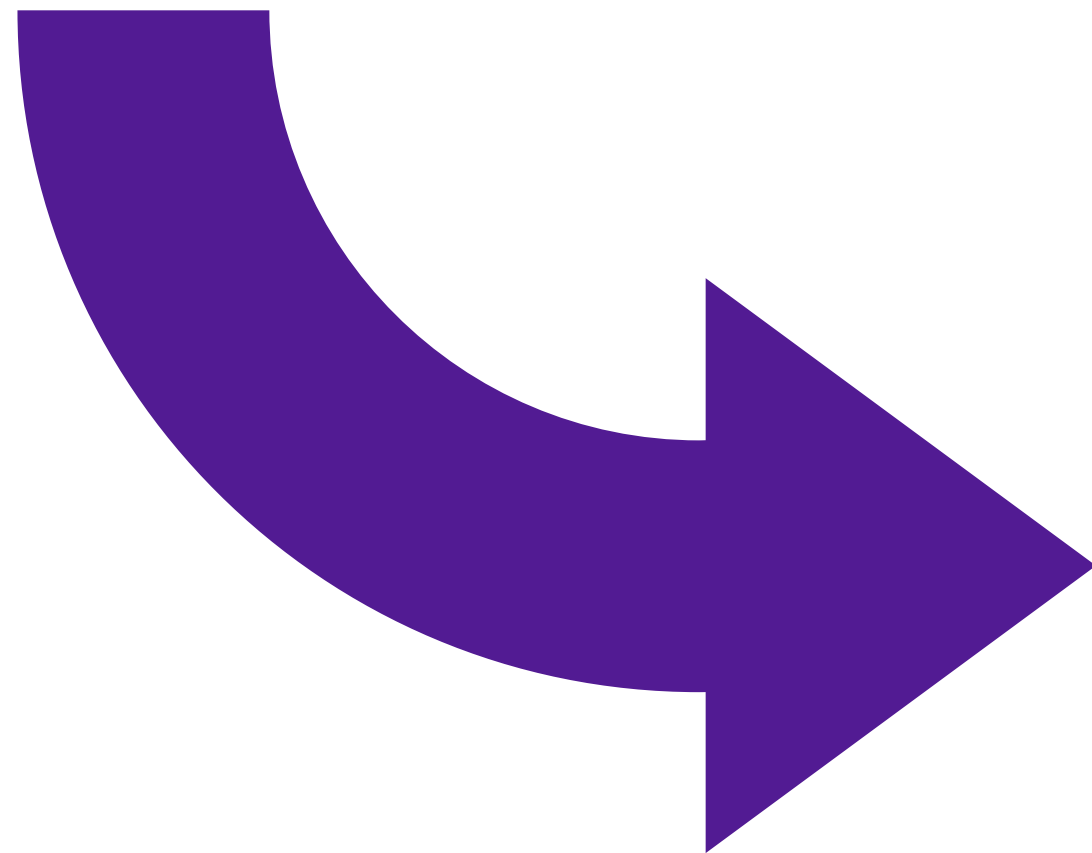
Enumerable



Enumerable

Original

```
foreach(  
    var n in Fibonacci( ).Take( 10 ) )  
{  
    Console.Write( $"{n}, " );  
}
```



```
▼ <Program>$.<<<Main>$>g__Fibonacci|0_0>d  
  ► Base Types  
    M .ctor(Int32)  
    F <>1__state : Int32  
    F <>2__current : Int32  
    F <>I__initialThreadId : Int32  
    F <next>5__3 : Int32  
    F <value>5__2 : Int32  
    M MoveNext() : Boolean  
    M System.Collections.Generic.IEnumerable<System.Int32>.GetEnumerator() : IEnumerator<Int32>  
    P System.Collections.Generic.IEnumerator<System.Int32>.Current : Int32  
    M System.Collections.IEnumerable.GetEnumerator() : IEnumerator  
    P System.Collections.IEnumerator.Current : Object  
    M System.Collections.IEnumerator.Reset() : Void  
    M System.IDisposable.Dispose() : Void
```

Agenda

- Hello World
- Record Type
- Enumerable
- **Async / Await**
- MoveNext()

Async / Await

```
using System;
using System.Threading.Tasks;

var x = Identity(6);
var y = await IdentityAsync(7);

Console.WriteLine($"{x:X4} + {y:X4} = {x * y:X4}");

static T Identity<T>(T x) => x;
static async Task<T> IdentityAsync<T>(T x) => x;
```

output

0006 + 0007 = 002A

only difference
is async

Async / Await

Original

```
using System;
using System.Threading.Tasks;

var x = Identity(6);
var y = await IdentityAsync(7);

Console.WriteLine($"{x:X4} + {y:X4} = {x * y:X4}");

static T Identity<T>(T x) => x;
static async Task<T> IdentityAsync<T>(T x) => x;
```

Compiled

```
using System;
using System.Runtime.CompilerServices;
using System.Threading.Tasks;

[CompilerGenerated]
internal static class _003CProgram_003E_0024
{
    private static async Task _003CMain_003E_0024 (string[] args)
    {
        int x2 = Identity<int> (6);
        int num = await IdentityAsync<int> (7);
        Console.WriteLine ($"{x2:X4} + {num:X4} = {x2 * num:X4}");
        static T Identity<T> (T x)
        {
            return x;
        }
        [AsyncStateMachine (typeof(_003C_003C_003CMain_003E_0024_003Eg__IdentityAsync_007C0_1_003Ed<>))]
        static Task<T> IdentityAsync<T> (T x)
        {
            _003C_003C_003CMain_003E_0024_003Eg__IdentityAsync_007C0_1_003Ed<T> stateMachine =
default(_003C_003C_003CMain_003E_0024_003Eg__IdentityAsync_007C0_1_003Ed<T>);
            stateMachine._003C_003Et__builder = AsyncTaskMethodBuilder<T>.Create ();
            stateMachine.x = x;
            stateMachine._003C_003E1__state = -1;
            stateMachine._003C_003Et__builder.Start (ref stateMachine);
            return stateMachine._003C_003Et__builder.Task;
        }
    }
}
```

Async / Await

Compiled

hidden class

```
[AsyncStateMachine(typeof(IdentityAsync<>))]
static Task<T> IdentityAsyncWraper<T>(T x)
{
    IdentityAsync<T> stateMachine = default(IdentityAsync<T>);
    stateMachine._builder = AsyncTaskMethodBuilder<T>.Create();
    stateMachine.x = x;
    stateMachine._state = -1;
    stateMachine._builder.Start(ref stateMachine);
    return stateMachine._builder.Task;
}
```

1. create State Machine
 - set Async Helper
 - set Parameters
 - set Initial State
2. call Async MoveNext
3. return Async Task

Async / Await

Compiled

```
[StructLayout(LayoutKind.Auto)]
struct IdentityAsync<T> : IAsyncStateMachine
{
    public int _state;
    public AsyncTaskMethodBuilder<T> _builder;

    public T x;

    private void SetStateMachine(IAsyncStateMachine stateMachine)
    {
        _builder.SetStateMachine(stateMachine);
    }

    void IAsyncStateMachine.SetStateMachine(IAsyncStateMachine stateMachine)
    {
        this.SetStateMachine(stateMachine);
    }
}
```

parameter

place State Machine on Heap

simplified C#

Async / Await

Compiled

```
private void MoveNext( )
{
    T result;
    try
    {
        result = x;
    }
    catch (Exception exception)
    {
        _state = -2;
        _builder.SetException(exception);
        return;
    }
    _state = -2;
    _builder.SetResult(result);
}
```

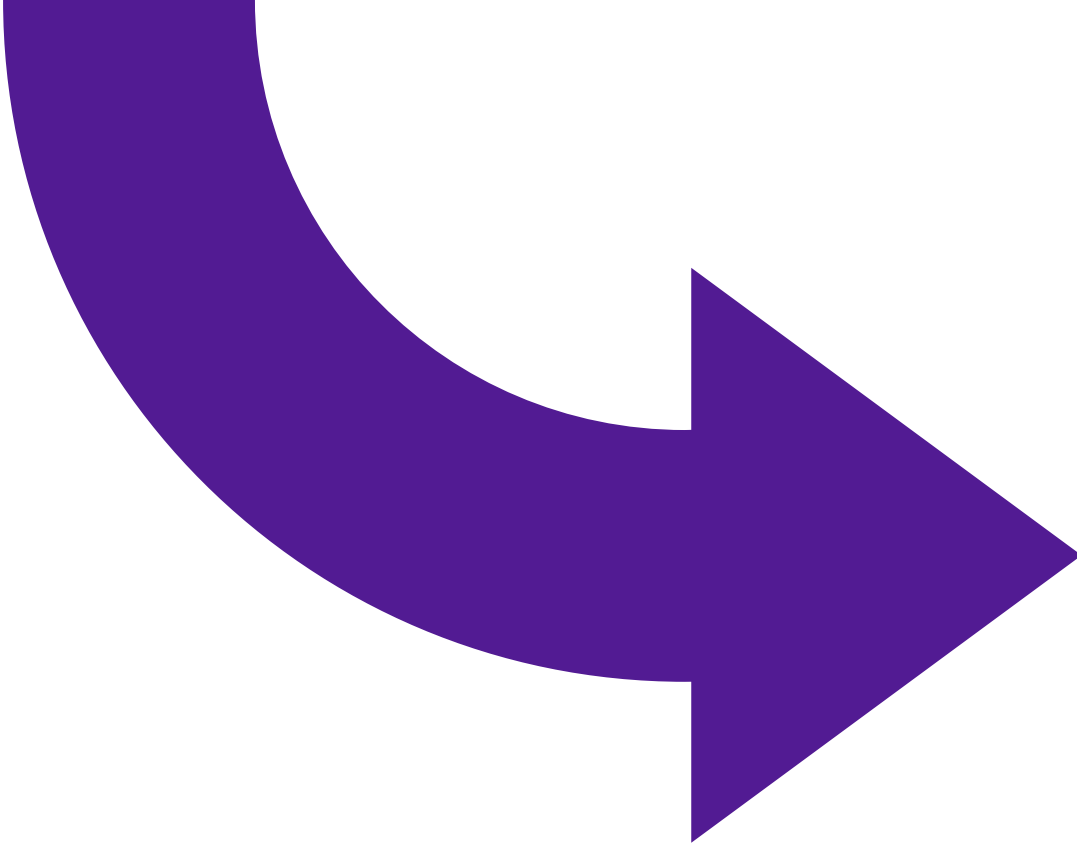
-2 state = complete

Async / Await

Original

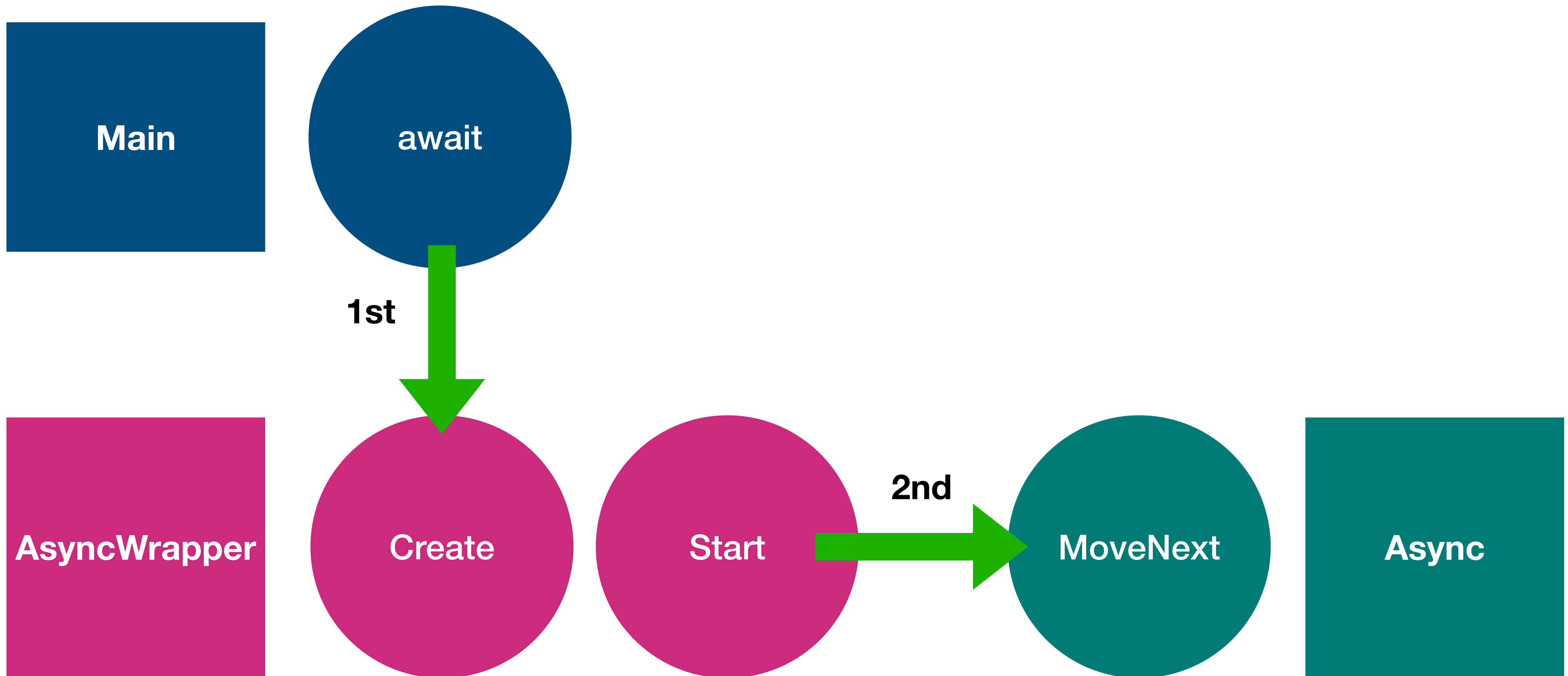
```
static async Task<T> IdentityAsync<T>(T x) => x;
```

Compiled



```
▼ <Program>$.<<<Main>$>g__IdentityAsync|0_1>d<T>  
  ► Base Types  
    M .ctor()  
    F <>1__state : Int32  
    F <>t__builder : AsyncTaskMethodBuilder<T>  
    M MoveNext() : Void  
    M SetStateMachine(IAsyncStateMachine) : Void  
    F x : T
```

Async / Await



Async / Await

something more interesting...

Async / Await

```
using System;
using System.Threading.Tasks;

await PrintAndWait(TimeSpan.FromMilliseconds(10));

static async Task PrintAndWait(TimeSpan delay)
{
    Console.WriteLine("before delays");
    await Task.Delay(delay);
    Console.WriteLine("between delays");
    await Task.Delay(delay);
    Console.WriteLine("after delays");
}
```

output

before delays
between delays
after delays

Async / Await

Original

```
using System;
using System.Threading.Tasks;

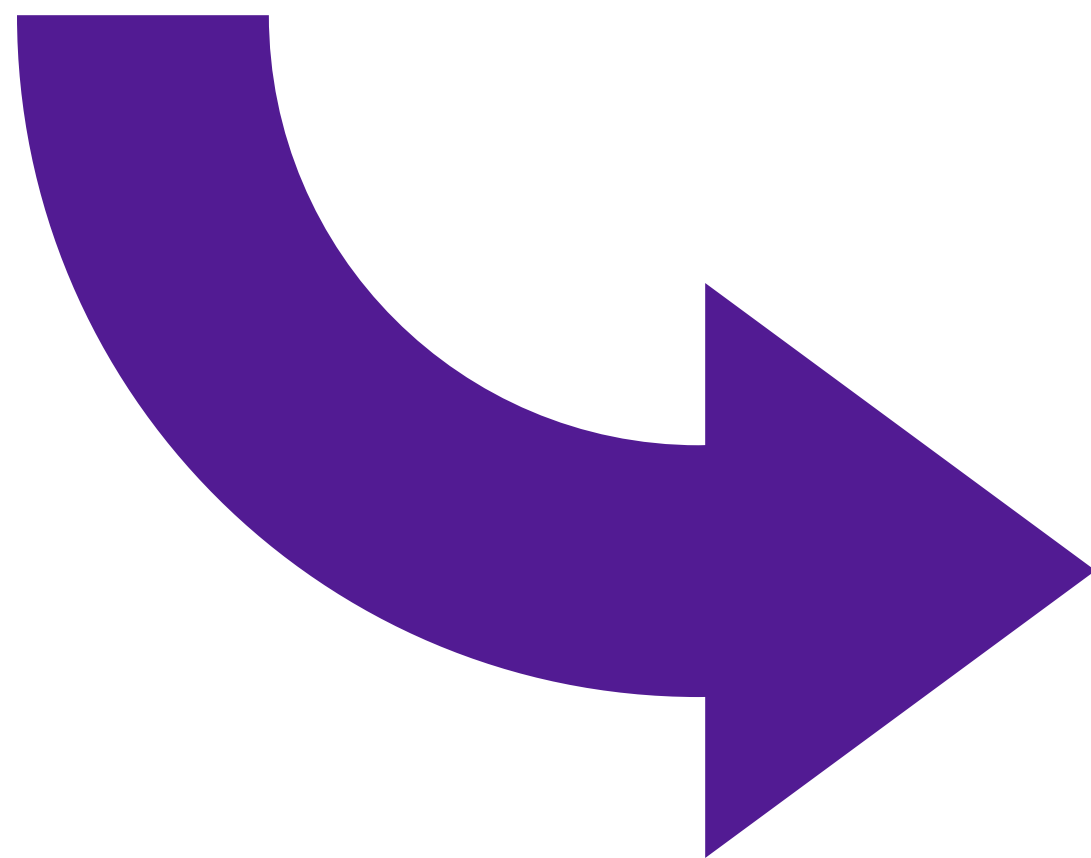
await PrintAndWait(TimeSpan.FromMilliseconds(10));

static async Task PrintAndWait(TimeSpan delay)
{
    Console.WriteLine("before delays");
    await Task.Delay(delay);
    Console.WriteLine("between delays");
    await Task.Delay(delay);
    Console.WriteLine("after delays");
}
```

Compiled

```
using System;
using System.Runtime.CompilerServices;
using System.Threading.Tasks;

[CompilerGenerated]
internal static class _003CProgram_003E_0024
{
    private static async Task _003CMain_003E_0024 (string[] args)
    {
        await PrintAndWait (TimeSpan.FromMilliseconds (10.0));
        [AsyncStateMachine (typeof(_003C_003C_003CMain_003E_0024_003Eg__PrintAndWait_007C0_0_003Ed))]
        static Task PrintAndWait (TimeSpan delay)
        {
            _003C_003C_003CMain_003E_0024_003Eg__PrintAndWait_007C0_0_003Ed stateMachine4 =
default(_003C_003C_003CMain_003E_0024_003Eg__PrintAndWait_007C0_0_003Ed);
            stateMachine4._003C_003Et__builder = AsyncTaskMethodBuilder.Create ();
            stateMachine4.delay = delay;
            stateMachine4._003C_003E1__state = -1;
            stateMachine4._003C_003Et__builder.Start (ref stateMachine4);
            return stateMachine4._003C_003Et__builder.Task;
        }
    }
}
```



Compiled

Async / Await

```
[StructLayout(LayoutKind.Auto)]
struct PrintAndWaitAsync : IAsyncStateMachine
{
    public int _state;
    public AsyncTaskMethodBuilder _builder;

    public TimeSpan delay;

    private TaskAwaiter _awaiter;

    private void SetStateMachine(IAsyncStateMachine stateMachine)
    {
        _builder.SetStateMachine(stateMachine);
    }

    void IAsyncStateMachine.SetStateMachine(IAsyncStateMachine stateMachine)
    {
        this.SetStateMachine(stateMachine);
    }
}
```

hidden class

parameter

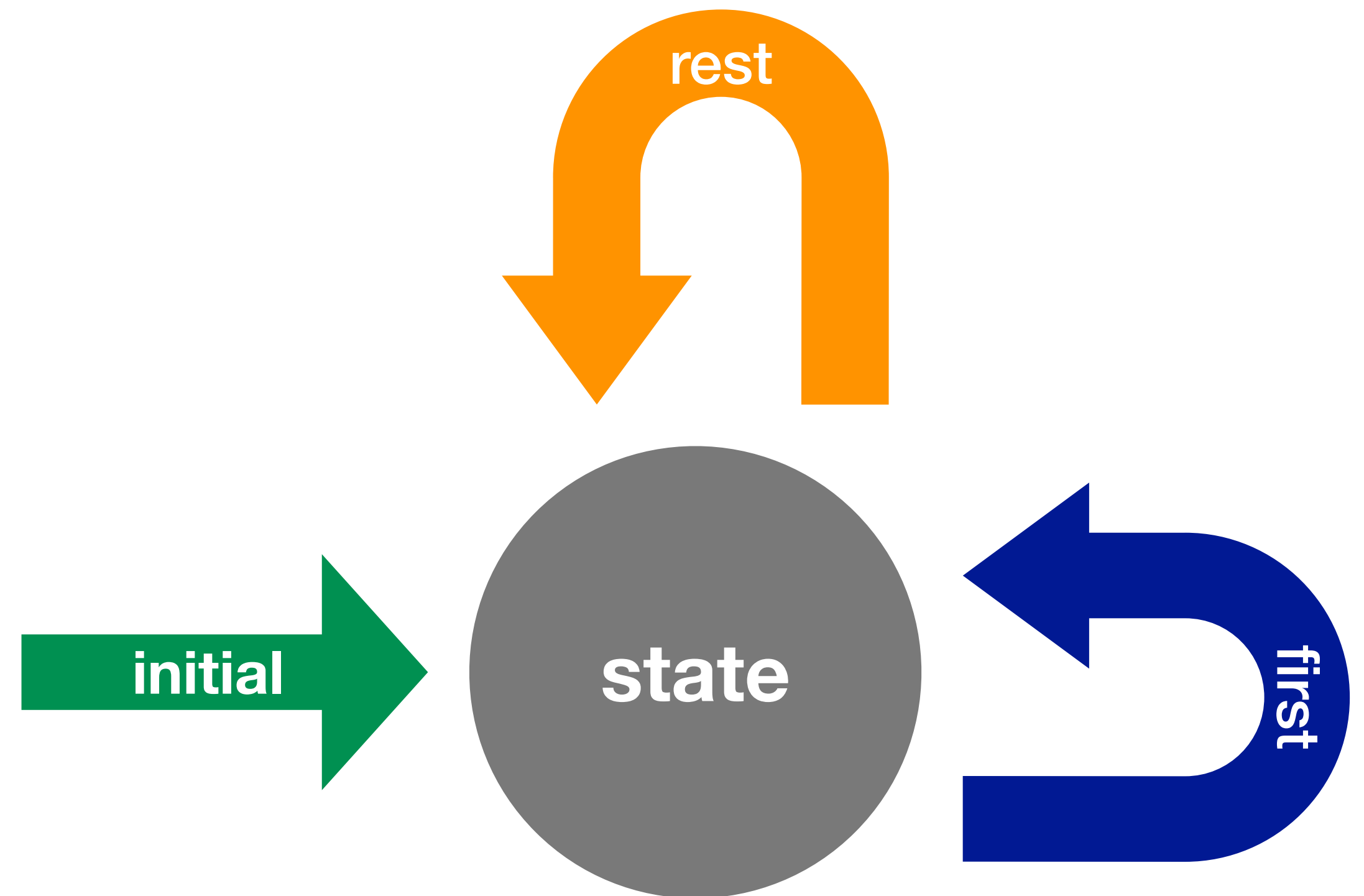
place State Machine on Heap

simplified C#

Compiled

```
private void MoveNext()
{
    int num = _state;
    try
    {
        TaskAwaiter awaiter;
        if (num != 0)
        {
            if (num == 1)
            {
                awaiter = _awaiter;
                _awaiter = default(TaskAwaiter);
                num = (_state = -1);
                goto done;
            }
            Console.WriteLine("before delays");
            awaiter = Task.Delay(delay).GetAwaiter();
            if (!awaiter.IsCompleted)
            {
                num = (_state = 0);
                _awaiter = awaiter;
                _builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);
                return;
            }
        }
        else
        {
            awaiter = _awaiter;
            _awaiter = default(TaskAwaiter);
            num = (_state = -1);
        }
        awaiter.GetResult();
        Console.WriteLine("between delays");
        awaiter = Task.Delay(delay).GetAwaiter();
        if (!awaiter.IsCompleted)
        {
            num = (_state = 1);
            _awaiter = awaiter;
            _builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);
            return;
        }
        goto done;
    }
    done:
        awaiter.GetResult();
        Console.WriteLine("after delays");
}
catch (Exception exception)
{
    _state = -2;
    _builder.SetException(exception);
    return;
}
_state = -2;
_builder.SetResult();
}
```

Async / Await



simplified C#

Async / Await

Original

```
Console.WriteLine("before delays");  
await Task.Delay(delay);
```

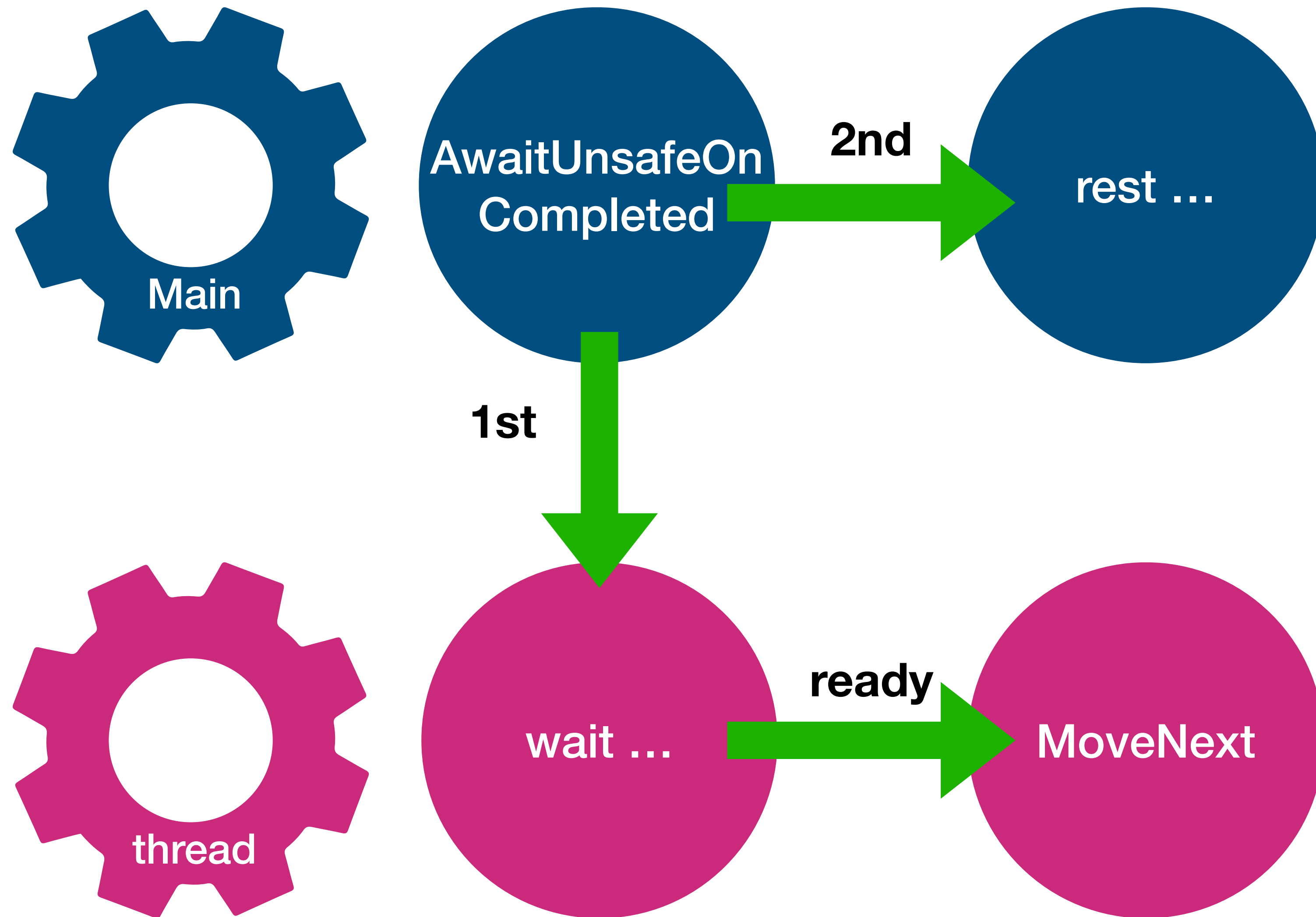
Compiled



```
Console.WriteLine("before delays");  
awaiter = Task.Delay(delay).GetAwaiter();  
if (!awaiter.IsCompleted)  
{  
    num = (_state = 0);  
    _awaiter = awaiter;  
    _builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);  
    return;  
}
```

creates thread with awaiter

Async / Await



Async / Await

Original

```
Console.WriteLine("between delays");  
await Task.Delay(delay);
```

Compiled

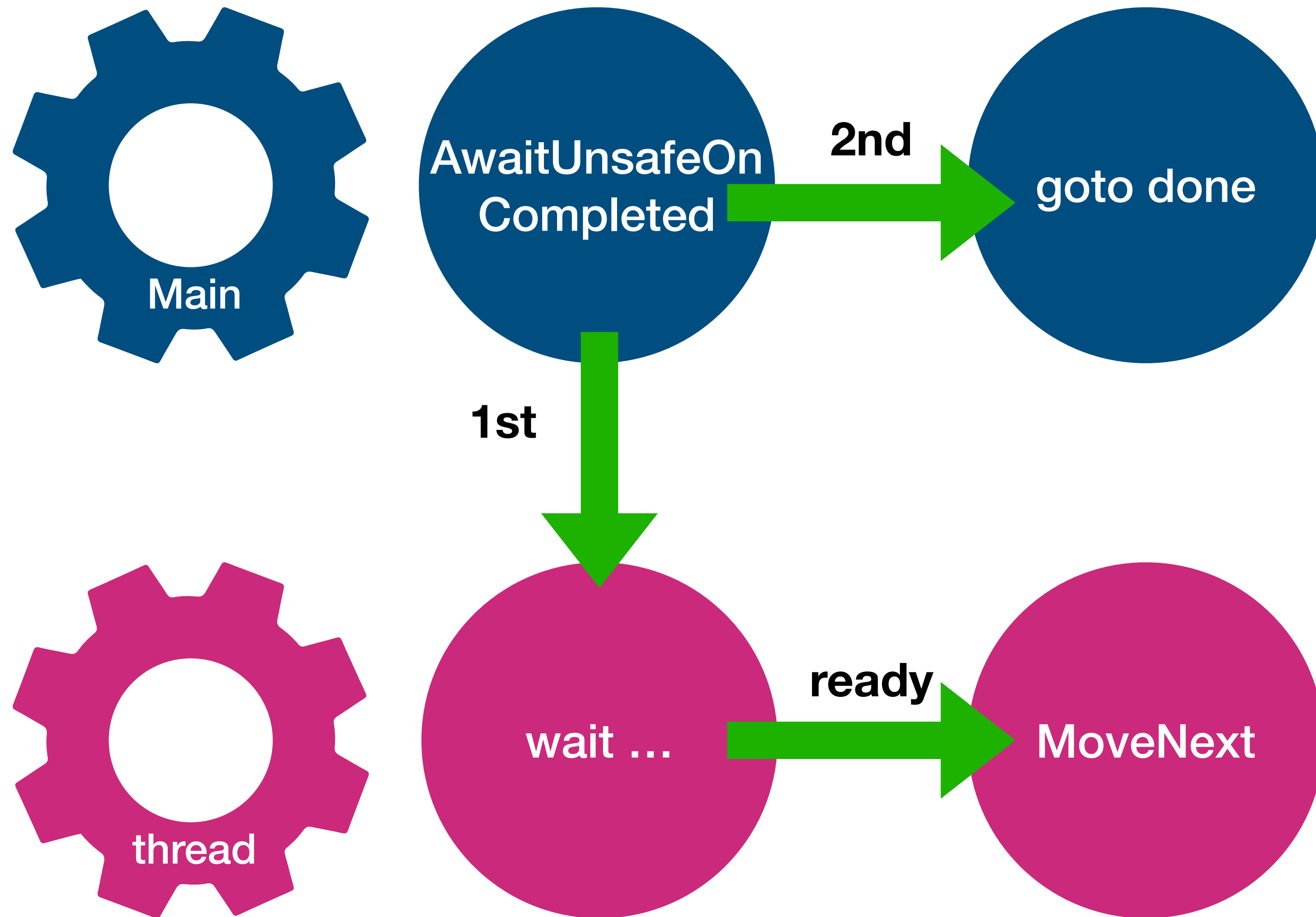


```
awaiter.GetResult();  
Console.WriteLine("between delays");  
awaiter = Task.Delay(delay).GetAwaiter();  
if (!awaiter.IsCompleted)  
{  
    num = (_state = 1);  
    _awaiter = awaiter;  
    _builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);  
    return;  
}  
goto done;
```

done with async

creates thread with awaiter

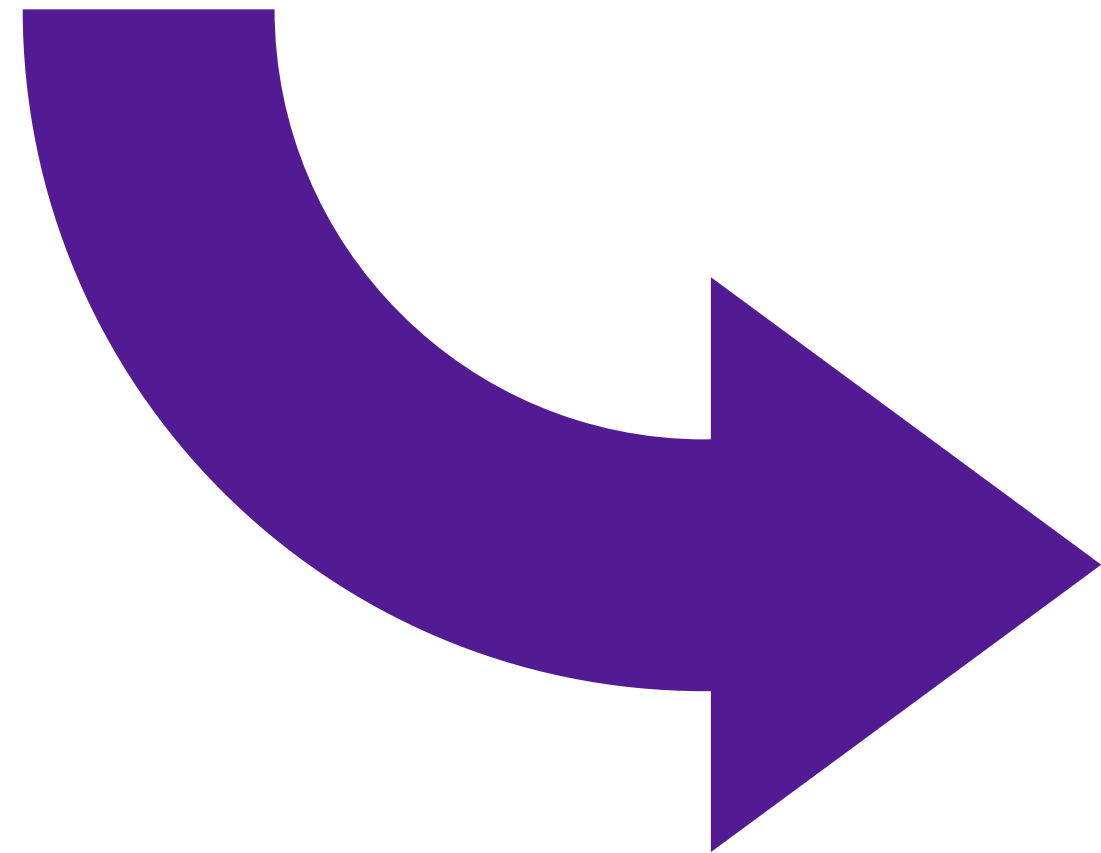
Async / Await



Async / Await

Original

```
Console.WriteLine("after delays");
```



Compiled

```
done:  
    awaiter.GetResult( );  
    Console.WriteLine("after delays");  
  
state = -2;  
_builder.SetResult( );
```

result of Task


Original Async / Await









```
using System;
using System.Threading.Tasks;

await PrintAndWait(TimeSpan.FromMilliseconds(10));

static async Task PrintAndWait(TimeSpan delay)
{
    Console.WriteLine("before delays");
    await Task.Delay(delay);
    Console.WriteLine("between delays");
    await Task.Delay(delay);
    Console.WriteLine("after delays");
}
```

Compiled

▼  <Program>\$.<<<Main>\$>g__PrintAndWait|0_0>d

- ▶  Base Types
 -  .ctor()
 -  <>1__state : Int32
 -  <>t__builder : AsyncTaskMethodBuilder
 -  <>u__1 : TaskAwaiter
 -  delay : TimeSpan
 -  MoveNext() : Void
 -  SetStateMachine(IAsyncStateMachine) : Void

Agenda

- Hello World
- Record Type
- Enumerable
- Async / Await
- **MoveNext()**

Next Steps

- **C# in Depth**, Fourth Edition by **Jon Skeet**
chapter 6
- **Working with C# Records** by **Roland Guijt** on Pluralsight
<https://app.pluralsight.com/library/courses/working-c-sharp-records/table-of-contents>
- **Extending the async methods in C#** by **Sergey Tepliakov**
<https://devblogs.microsoft.com/premier-developer/extending-the-async-methods-in-c/>
- **ILSpy**
<https://github.com/icsharpcode/ILSpy>



Thank you

Mike Harris

My Compiler Did What?!?

@MikeMKH

<https://github.com/MikeMKH/talk-my-compiler-did-what>

