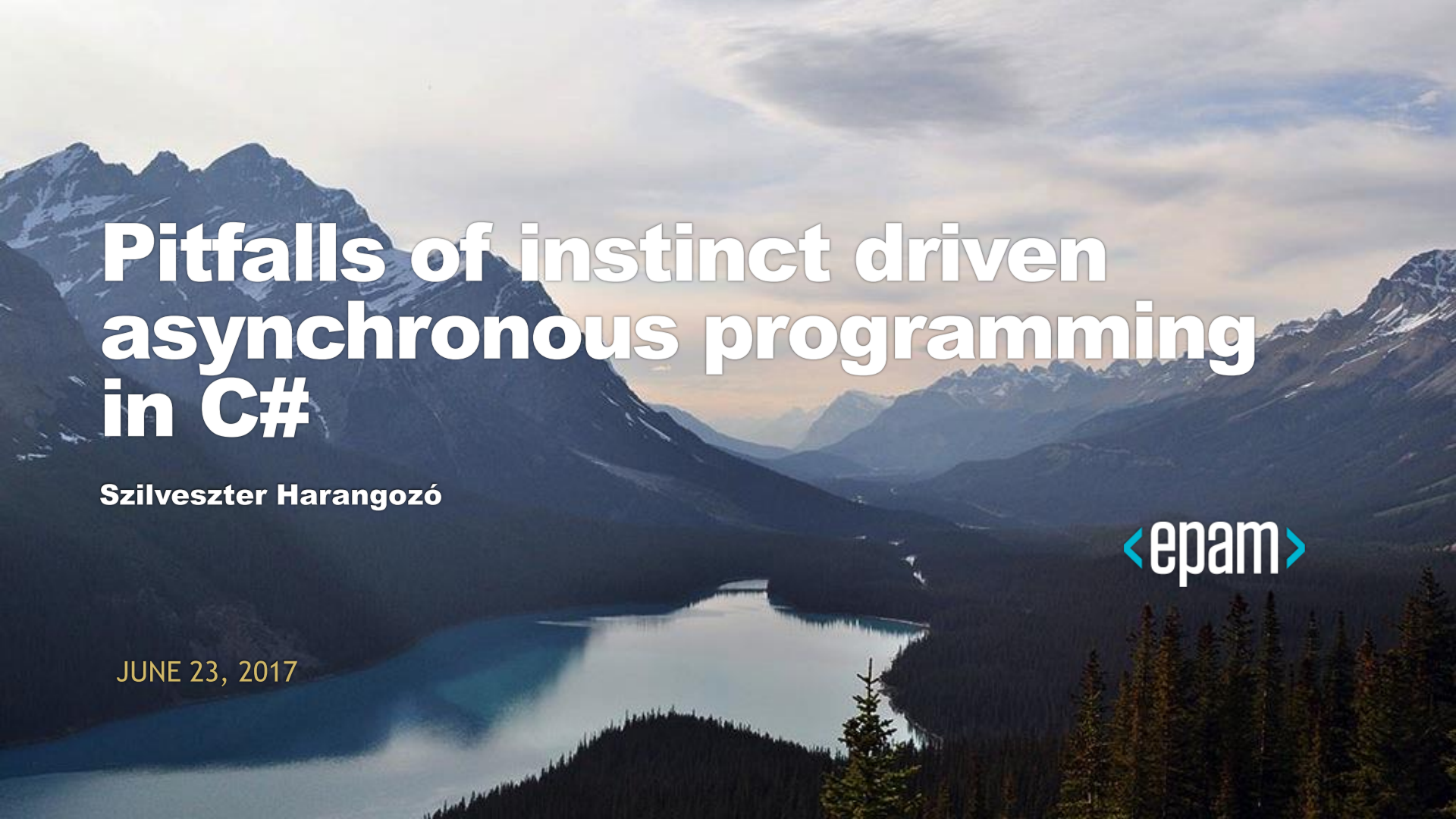

Pitfalls of instinct driven asynchronous programming in C#
Szilveszter Harangozó
epam
JUNE 23, 2017

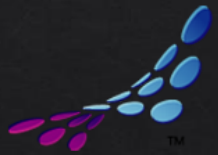
AGENDA

TOPICS

- Task-Based Asynchronous Pattern
- Async/Await

POINT OF FAILURES

- Task completion
- Method signature
- Synchronization context
- Special cases



msdn

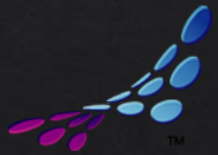
TASK-BASED ASYNCHRONOUS PATTERN

THREADS

- costly
- non-trivial amount of memory
- hard to handle

TASKS

- Not bound to a physical resource
- Thread Pool
- Task Scheduler

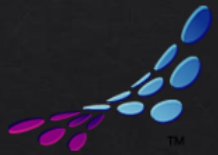


msdn

TASK-BASED ASYNCHRONOUS PATTERN

KEY FEATURES

- Return types: Task or Task<TResult>
- Exception handling
- Status report
- Cancellation (optional)
- Progress report (optional)



msdn

TASK-BASED ASYNCHRONOUS PATTERN

TASK CREATION

- Cost of Synchronization payed at *Start*
- Trigger methods: *Wait*,*Result*
- Suggestion: use *Task.Factory*

```
var task = new Task(Func<TResult> func);
task.Start();
//...
task.Wait();

var task = Task.Run(Func<TResult> func);
//...
task.Wait();

var task = Task.Factory.StartNew(Func<TResult> func);
//...
task.Wait();
```

TASK COMPLETION

TASK COMPLETION SOURCE

- Source explicitly controlled by the methods
- *Task* can be handed out to consumers
- Exceptions should be passed

Psychic Debugging of Async Methods

```
public Task<int> SomeLibraryMethodAsync()
{
    var tcs = new
    TaskCompletionSource<int>();
    Task.Factory.StartNew(() =>
    {
        try
        {
            int result = SomeLibraryMethod();
            tcs.SetResult(result);
        }
        catch (Exception e)
        {   // Bug!
            tcs.SetException(e);
        }
    });
    return tcs.Task;
}
```

ASYNC/AWAIT

ASYNC/AWAIT

ASYNC

- Decorate methods, lambda expressions
- Run without caller thread blocking
- Allows usage of await

AWAIT

- Releases the thread
- Informs the caller when result is ready

```
private static async Task GetHttpResponseAsync()
{
    using (var httpClient = new HttpClient())
    {
        var responseTask = httpClient
            .GetAsync("https://msdn.microsoft.com");

        // Do independent work...

        var response = await responseTask;
        Console.WriteLine(response.Headers);
    }
}
```

ASYNC/AWAIT

PROS

- Framework responsible for Threads
- No cost for thread creation
- Readable code
- No necessity for method delegates
- **Fast development**

```
private static async Task GetHttpResponseAsync()
{
    using (var httpClient = new HttpClient())
    {
        var responseTask = httpClient
            .GetAsync("https://msdn.microsoft.com");

        // Do independent work...

        var response = await responseTask;
        Console.WriteLine(response.Headers);
    }
}
```

ASYNC/AWAIT

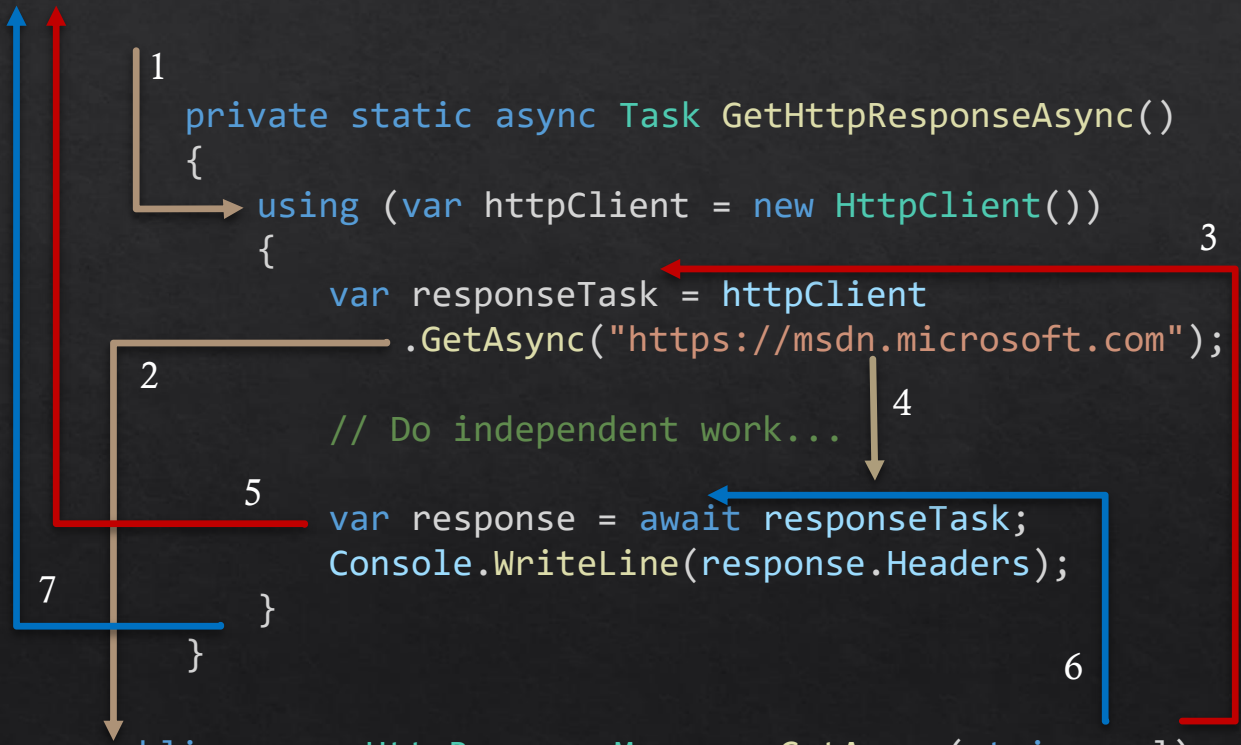
CONS

- Computational overhead
- No info on threads
- Difficult debug
- **Can do, but should not**

Normal processing

Yielding control to caller

Resuming suspended process



Async and Await (msdn)

ASYNC/AWAIT UNDER THE HOOD

```
private static async Task GetHttpResponseAsync()
{
    using (var httpClient = new HttpClient())
    {
        var responseTask = httpClient
            .GetAsync("https://msdn.microsoft.com");

        // Do independent work...

        var response = await responseTask;
        Console.WriteLine(response.Headers);
    }
}
```

ASYNC/AWAIT UNDER THE HOOD

```
[AsyncStateMachine(typeof(<GetHttpResponseAsync>d__1)), DebuggerStepThrough]
private static Task GetHttpResponseAsync()
{
    <GetHttpResponseAsync>d__1 stateMachine = new <GetHttpResponseAsync>d__1 {
        <>t__builder = AsyncTaskMethodBuilder.Create(),
        <>1__state = -1
    };
    stateMachine.<>t__builder.Start<<GetHttpResponseAsync>d__1>(ref stateMachine);
    return stateMachine.<>t__builder.Task;
}

[CompilerGenerated]
private sealed class <GetHttpResponseAsync>d__1 : IAsyncStateMachine
```

Behind the .NET 4.5 Async Scene

SIGNATURE

ASYNC VOID

- Will this code print „Failed”?
- Useful for event handlers
- Fire and forget
- The recommendation is...

```
private async void ThrowExceptionAsync()
{
    throw new InvalidOperationException();
}
public void CallThrowExceptionAsync()
{
    try
    {
        ThrowExceptionAsync();
    }
    catch (Exception)
    {
        Console.WriteLine("Failed");
    }
}
```

For goodness' sake stop using async void

SIGNATURE

ASYNC LAMBDA

- What will be the second snippet's result?
- *Async lambda* mapped to *async void*
- Caller signature!

```
Seconds: 1.000361
Seconds: 0.001521
Seconds: 1.006651
```

```
var secs = Time(() =>
{
    Thread.Sleep(1000);
});
Console.WriteLine($"Seconds: {secs:F6}");

var secs2 = Time(async () =>
{
    await Task.Delay(1000);
});
Console.WriteLine($"Seconds: {secs2:F6}");


public static double Time(Action action)
public static double Time(Func<Task> func)
```

SIGNATURE

ASYNC LAMBDA

- What will be the type of **u**?
- *StartNew* in: *Func<TResult>* out: *Task<TResult>*
- Task nesting

```
var t = Task.Factory.StartNew(() =>
{
    Thread.Sleep(1000);
    return 42;
});

var u = Task.Factory.StartNew(async () =>
{
    await Task.Delay(1000);
    return 42;
}).Unwrap();
```

Passing async lambdas

TASK SYNCHRONIZATION

- What will be the message print order?
- Problem: heavy computation before *await*

```
work          started
started       work
completed
```

```
private async Task WorkThenWait()
{
    await Task.Yield();
    Thread.Sleep(1000);
    Console.WriteLine("work");
    await Task.Delay(1000);

}
public void Demo()
{
    var child = WorkThenWait();
    Console.WriteLine("started");
    child.Wait();
    Console.WriteLine("completed");
}
```

Asynchronous gotchas in C#

TASK SYNCHRONIZATION

- What is the current SynchronizationContext?
- WebAPI: ASP.NET request context
- Deadlock!
- *ConfigureAwait(false)* **HACK**

```
public class MyController : ApiController
{
    private static async Task<JObject> GetJsonAsync(Uri uri)
    {
        using (var client = new HttpClient())
        {
            var jsonString = await client.GetStringAsync(uri);
            return JObject.Parse(jsonString);
        }
    }
    public async Task<string> GetAsync()
    {
        var json =
            await GetJsonAsync(new Uri("http://bing.com"));
        return json.ToString();
    }
}
```

Don't Block on Async Code

AWAIT AND COMPOUND ASSIGNMENT

- 2 calls, stack *m_sum*
- Field read to stack once
- Local variable: OK

```
4
6
```

Don't mix await and compound assignment

```
class Accumulator
{
    private int m_sum = 0;
    public int Sum => m_sum;

    public async Task Add(Task<int> value)
    {
        var temp = await value;
        m_sum += temp;
    }
}

var task1 = acc.Add(tcs1.Task);
var task2 = acc.Add(tcs2.Task);
tcs1.SetResult(2); tcs2.SetResult(4);
await task1; await task2;
Console.WriteLine(acc.Sum);
```

MESSAGE

- Don’t forget to complete your Tasks!
- Watch out for Signatures!
- Be aware of your Synchronization context!
- Think double, less trouble...


epam
THANK YOU
Superdry