Asynchronous IO in .Net

Mikhail Raer

Michael.Raer@gmail.com

About myself

- Informational Technologies Mechanics and Optics, Saint-Petersburg
- Software Developer since 2003
- 4 years in Tallinn
- Senior Software Developer in Skype since 2012

What Async IO is not?

Async IO is not Multithreading

Thread 1

```
public async Task<int> GetData()
{
    var data = await someDataClient.GetData();

    return data;
}
```

Thread 1

What Async IO is not?

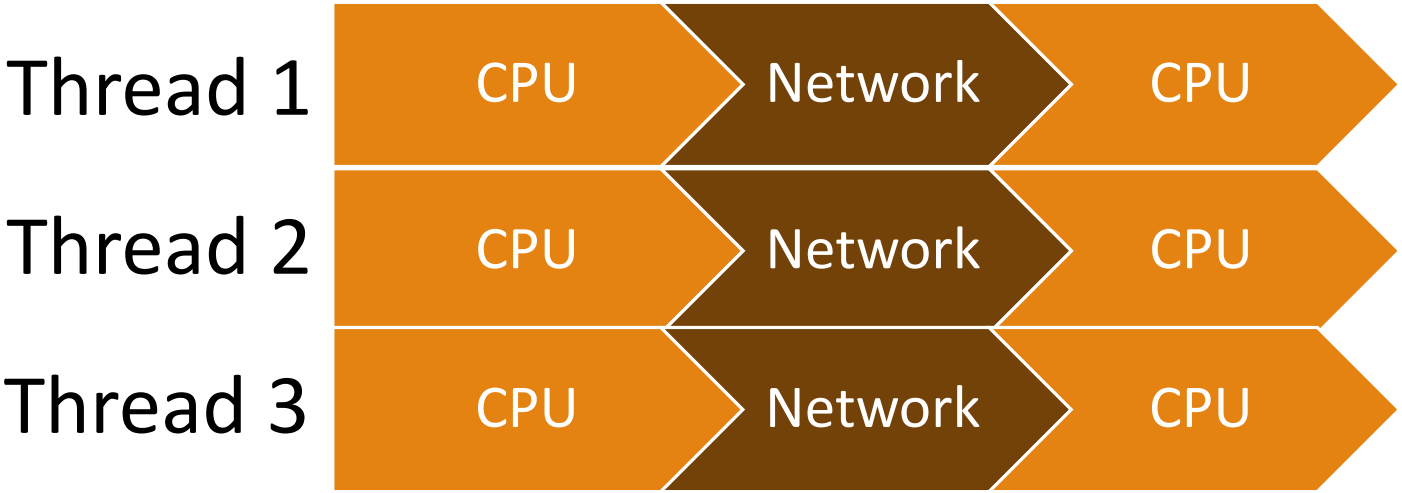
Async IO is not Parallelism

```
public async Task<int> GetSomething()
{
    var data1 = await client1.GetData();
    var data2 = await client2.GetData();

    return data1 + data2;
}
```

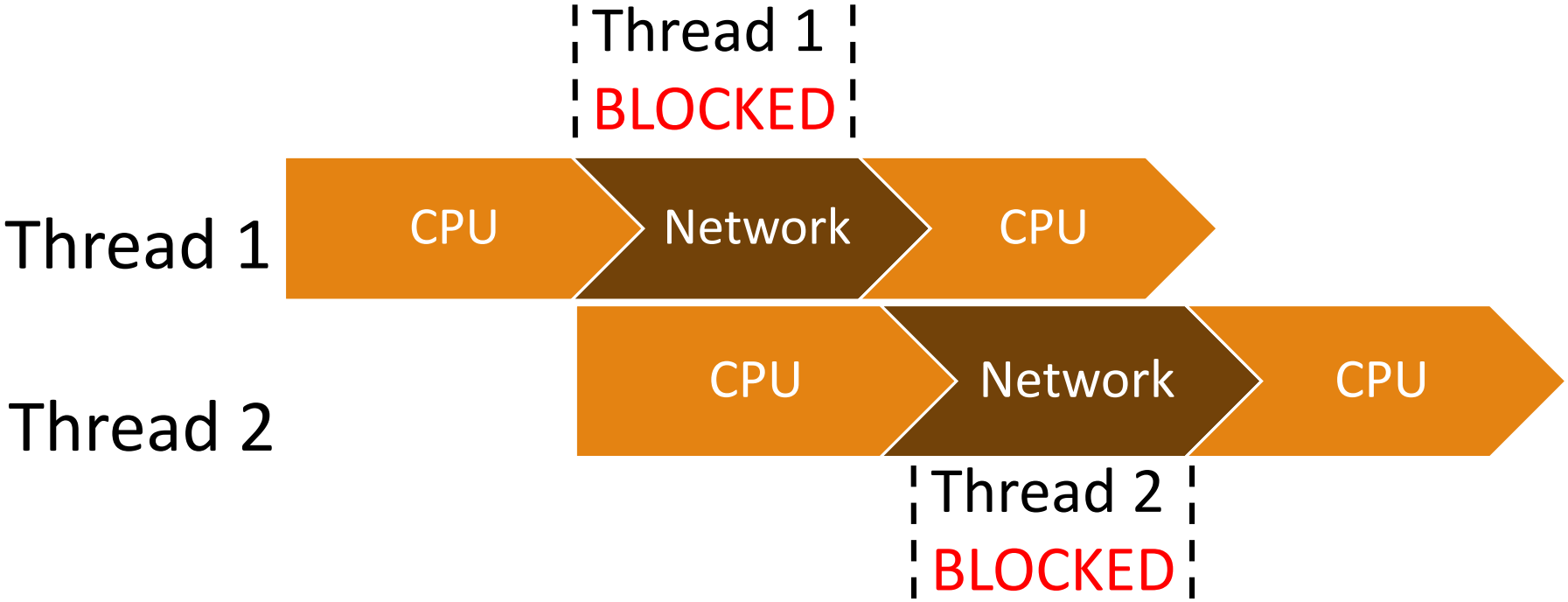
Typical flow for Request

- Prepare request to Storage
- *Waiting for result from Network*
- Process result
- Send response to user

Concurrent Requests



Threads


Thread 1
BLOCKED
Thread 1
CPU
Network
CPU
Thread 2
CPU
Network
CPU
Thread 2
BLOCKED

Threads

1000 requests? => 1000 threads?

Don’t block threads

Non blocking API

Callback

Blocked Thread

Happy Thread

Non blocking API (APM)

```
public IAsyncResult ReadFromFile()
{
  return fileStream.BeginRead(buffer, 0, 10, ReadCallback,
    fileStream);
}
```

Exited immediately

When operation finished

```
public void ReadCallback(IAsyncResult asyncResult)
{
  var fileStream = (FileStream) asyncResult.AsyncState;
  fileStream.EndRead(asyncResult);
}
```

Turns into a sausage on long processing

Sausage

```
private void ReadFromFile()
{
   fileStream.BeginRead(buffer, 0, 10, ReadCallback,
     fileStream);
}

private void ReadCallback(IAsyncResult asyncResult)
{
    var fs = (FileStream) asyncResult.AsyncState;
    fs.EndRead(asyncResult);
    request.BeginGetResponse(RequestCallback, request);
}

private void RequestCallback(IAsyncResult asyncResult)
{
    var wr = (HttpWebRequest)asyncResult.AsyncState;
    wr.EndGetResponse(asyncResult);
}
```

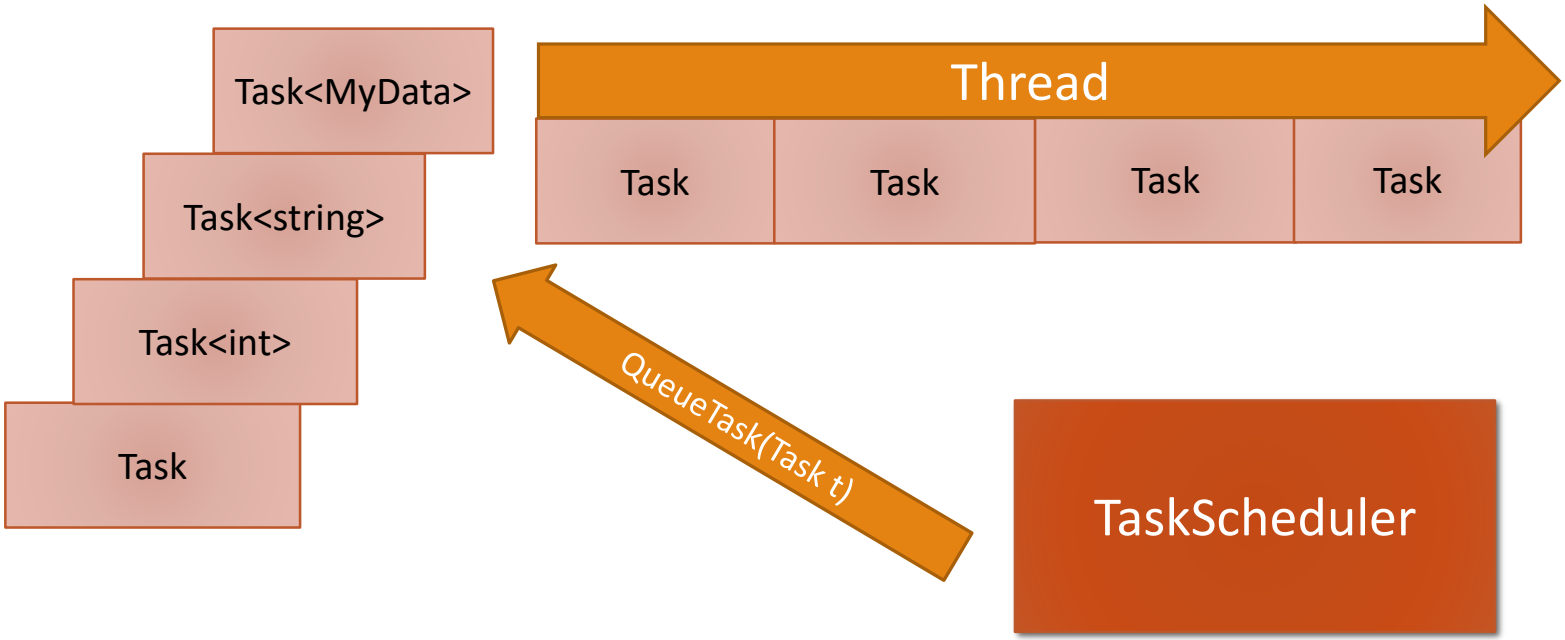
AsyncEnumerator

```
IEnumerable<int> WriteData(AsyncEnumerator ae, string filename)
{
 using (FileStream fs = new FileStream(…))
 {
    fs.BeginWrite(…, ae.End(), null);

    yield return 1;  Exited immediately

    Continues when write is done

    fs.EndWrite(ae.DequeueAsyncResult());
  }
}
```

Sausage collapses into single method

https://channel9.msdn.com/blogs/charles/jeffrey-richter-and-his-asyncenumerator

Tasks


Task<MyData>
Task<string>
Task<int>
Task
Thread
Task
Task
Task
Task
QueueTask(Task t)
TaskScheduler

async-await syntax

```
private async Task<int> ReadDataAsync()
{
    var response = await httpClient.GetAsync(url);
```

Exited immediately

```
    return response.Amount;
}
```

Continuation after callback

```
...

var result1 = await ReadDataAsync();
var resutl2 = ReadDataAsync().Result;
```

What you could await? Food!

```
public async Task<int> GetFood()
{
    var spaghetti = new Spaghetti();
    var result = await spaghetti;
    return result;
}

public class Spaghetti : INotifyCompletion
{
    public Spaghetti GetAwaiter()
    {
        return this;
    }

    public bool IsCompleted => false;
    public int GetResult() => 10;
    public void OnCompleted(Action continuation)
    {
       // Execute continuation when Completed
       continuation.Invoke();
    }
}
```

How async-await works?

```
public async Task<int> GetSomething()
{
    var data = await client.GetData();
    return data;
}


public Task<int> GetSomething()
{
    Program.<GetSomething>d__11 stateMachine =
        new Program.<GetSomething>d__11();
    stateMachine.<>4__this = this;
    stateMachine.<>t__builder =
        AsyncTaskMethodBuilder<int>.Create();
    stateMachine.<>1__state = -1;
    stateMachine.<>t__builder
      .Start<Program.<GetSomething>d__11>(ref stateMachine);

    return stateMachine.<>t__builder.Task;
}
```

```
private sealed class <GetSomething>d__11 : IAsyncStateMachine
{

    void IAsyncStateMachine.MoveNext()  Reentering here
    {
        int num1 = <>1__state;
        int result52;
        try
        {
             TaskAwaiter<int> awaiter;
             int num2;
             if (num1 != 0)
             {
               awaiter = this.<>4__this.client1.GetData().GetAwaiter();
               if (!awaiter.IsCompleted)
               {
                 this.<>1__state = num2 = 0;
                 this.<>u__1 = awaiter;
                 Program.<GetSomething>d__11 stateMachine = this;
                 this.<>t__builder.AwaitUnsafeOnCompleted<TaskAwaiter<int>,Program.<GetSomething>d__11>(
                     ref awaiter,
                     ref stateMachine
                 );
                 return;  Exited here
               }
             }
             int result = awaiter.GetResult();
             awaiter = new TaskAwaiter<int>();
             this.<>s__3 = result;
             this.<data>5__1 = this.<>s__3;
             this.<result>5__2 = this.<data>5__1 + 1;
             result52 = this.<result>5__2;
            }
            this.<>1__state = -2;
            this.<>t__builder.SetResult(result52);
        }
    }
```

Disadvantages

```
41
42
        1 reference
43      public async Task<int> GetData()
44      {
45          var bytesRead = await fileStream.ReadAsync(buffer, 0, 10);
46          return bytesRead;   ≤8ms elapsed
47      }
48
```

That’s how stack trace looks like

Call Stack

Name
ConsoleApplication1.exe!ConsoleApplication1.Program.GetData() Line 48
[Resuming Async Method]
mscorlib.dll!System.Runtime.CompilerServices.AsyncMethodBuilderCore.MoveNextRunner.InvokeMoveNext(object stateMachine) Line 1090
mscorlib.dll!System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext executionContext, System.Threading.ContextCallback callback, object state, bool preserveS
mscorlib.dll!System.Threading.ExecutionContext.Run(System.Threading.ExecutionContext executionContext, System.Threading.ContextCallback callback, object state, bool preserveSyncCtx)
mscorlib.dll!System.Runtime.CompilerServices.AsyncMethodBuilderCore.MoveNextRunner.Run() Line 1070
mscorlib.dll!System.Runtime.CompilerServices.AsyncMethodBuilderCore.OutputAsyncCausalityEvents.AnonymousMethod__0() Line 977
mscorlib.dll!System.Runtime.CompilerServices.AsyncMethodBuilderCore.ContinuationWrapper.Invoke() Line 1123
mscorlib.dll!System.Runtime.CompilerServices.TaskAwaiter.OutputWaitEtwEvents.AnonymousMethod__0() Line 288
mscorlib.dll!System.Runtime.CompilerServices.AsyncMethodBuilderCore.ContinuationWrapper.Invoke() Line 1123
mscorlib.dll!System.Threading.Tasks.AwaitTaskContinuation.RunOrScheduleAction(System.Action action, bool allov
mscorlib.dll!System.Threading.Tasks.Task.FinishContinuations() Line 3617
mscorlib.dll!System.Threading.Tasks.Task.FinishStageThree() Line 2363
mscorlib.dll!System.Threading.Tasks.Task<int>.TrySetResult(int result) Line 490
mscorlib.dll!System.Threading.Tasks.TaskFactory<int>.FromAsyncTrimPromise<System.IO.Stream>.Complete(Syste
mscorlib.dll!System.Threading.Tasks.TaskFactory<int>.FromAsyncTrimPromise<System.IO.Stream>.CompleteFrom
mscorlib.dll!System.IO.Stream.ReadWriteTask.InvokeAsyncCallback(object completedTask) Line 671
mscorlib.dll!System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext executionContext,
mscorlib.dll!System.Threading.ExecutionContext.Run(System.Threading.ExecutionContext executionContext, System
mscorlib.dll!System.IO.Stream.ReadWriteTask.System.Threading.Tasks.ITaskCompletionAction.Invoke(System.Threa
mscorlib.dll!System.Threading.Tasks.Task.FinishContinuations() Line 3633
mscorlib.dll!System.Threading.Tasks.Task.FinishStageThree() Line 2363
mscorlib.dll!System.Threading.Tasks.Task.FinishStageTwo() Line 2336
mscorlib.dll!System.Threading.Tasks.Task.Finish(bool bUserDelegateExecuted) Line 2253
mscorlib.dll!System.Threading.Tasks.Task.ExecuteWithThreadLocal(ref System.Threading.Tasks.Task currentTaskSlot) Line 2830
mscorlib.dll!System.Threading.Tasks.Task.ExecuteEntry(bool bPreventDoubleExecution) Line 2767
mscorlib.dll!System.Threading.Tasks.Task.System.Threading.IThreadPoolWorkItem.ExecuteWorkItem() Line 2704
mscorlib.dll!System.Threading.ThreadPoolWorkQueue.Dispatch() Line 820
mscorlib.dll!System.Threading._ThreadPoolWaitCallback.PerformWaitCallback() Line 1161

Call Stack | Breakpoints | Exception Settings | Command Window | Immediate Window | Output

ExecutionContext

State bag captured on one thread and restored on another, follows logical flow.

ExecutionContext contains SecurityContext, HostExecutionContext, CallContext, SynchronizationContext


```
// at Task.Run for example or before await
ExecutionContext ec = ExecutionContext.Capture();


// Invoked by awaiter in OnCompleted
ExecutionContext.Run(ec, delegate
// executed on another Thread where context is restored
{
    // ec state applied
}, null);
```

ActivityId

```
Trace.CorrelationManager.ActivityId = Guid.NewGuid();
```

```
CallContext.LogicalSetData(activityIdSlotName, value);
```

```
public static void LogicalSetData(String name, Object data)
{
   ExecutionContext ec = Thread.CurrentThread.GetMutableExecutionContext();
   ec.IllogicalCallContext.FreeNamedDataSlot(name);
   ec.LogicalCallContext.SetData(name, data);
}
```

SynchronizationContext

Abstraction of an environment where task should be continued (UI Thread, ASP.NET environment)

(WindowsFormSynchonizationContext, AspNetSynchronizationContext)

```
// Capture
var sc = SynchronizationContext.Current;

// Run delegate in captured
context environment
sc.Post(state => { }, null);
```

SynchronizationContext captured before await and continuation posted to it.

Opt-out: `await task.ConfigureAwait(false);`

SynchronizationContext

SynchronizationContext flows as part of ExecutionContext

```
private async void button1_Click(object sender, EventArgs e)
{
    button1.Text = await Task.Run(async delegate
    {
        string data = await DownloadAsync();
        return Compute(data);
    });
}
```

Current SC is UI SC –Continuation posted to it

Current SC is UI SC?

But finally Compute will be run on ThreadPool, not UI Thread

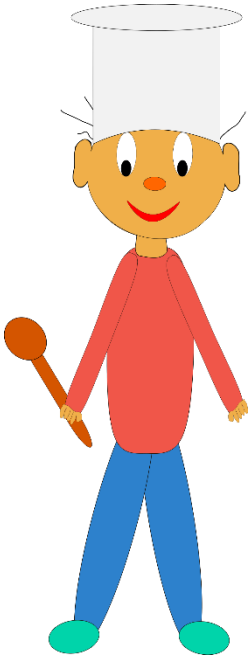
Configure await in libs

Use ConfigureAwait(false) in libraries

```
private async void button1_Click(object sender, EventArgs e)
{
    button1.Text = await DownloadAsync();
}
```

```
private async Task<string> DownloadAsync()
{
  var response = await httpClient.GetAsync(url).ConfigureAwait(false);
  return
     await response.Content.ReadAsStringAsync().ConfigureAwait(false);
}
```

Q&A



Email: Michael.Raer@gmail.com

Skype: Michael.Raer