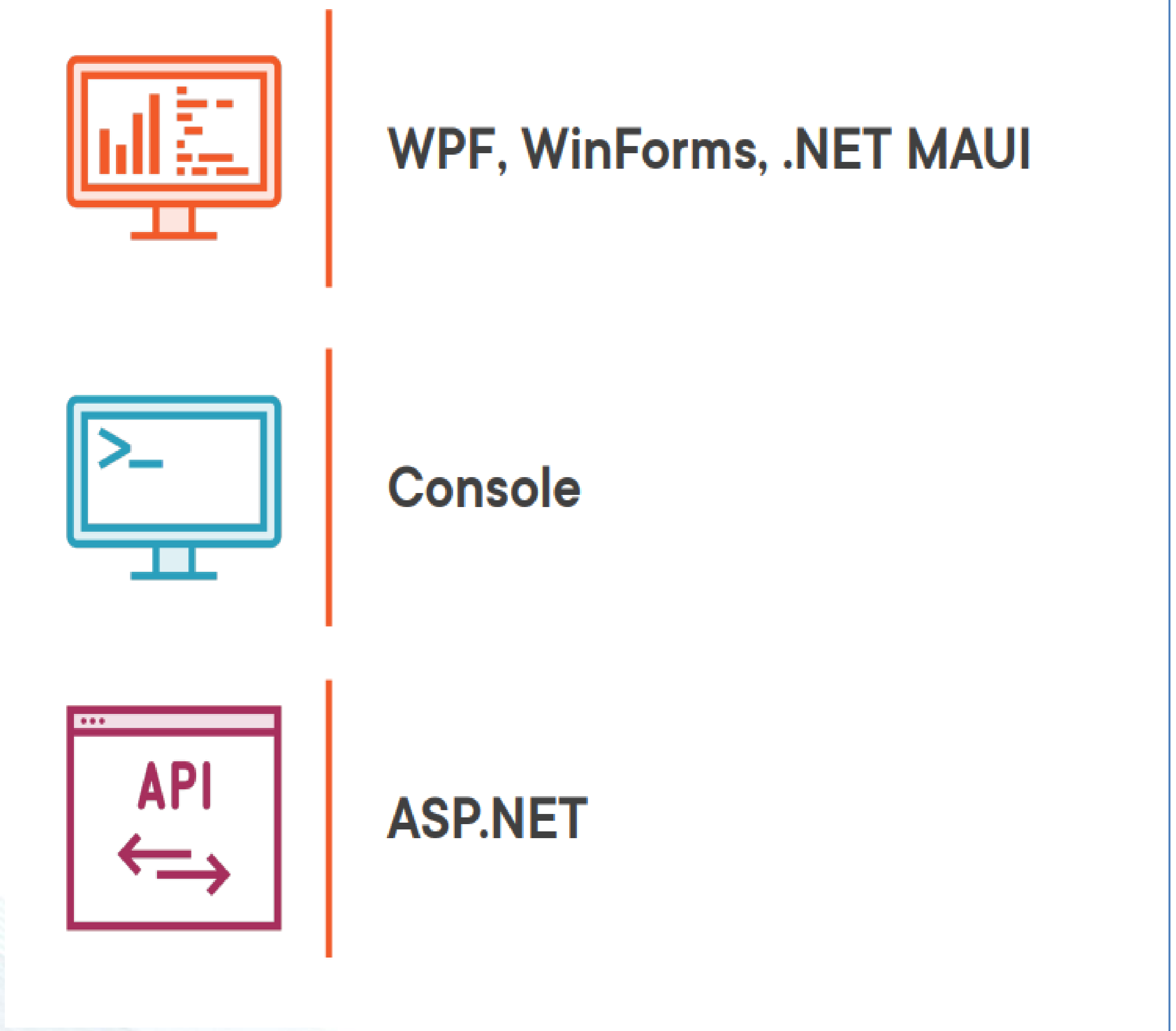
Best Practices for Asynchronous – Multithreaded Programming

Shiva Singh - Solution Architect

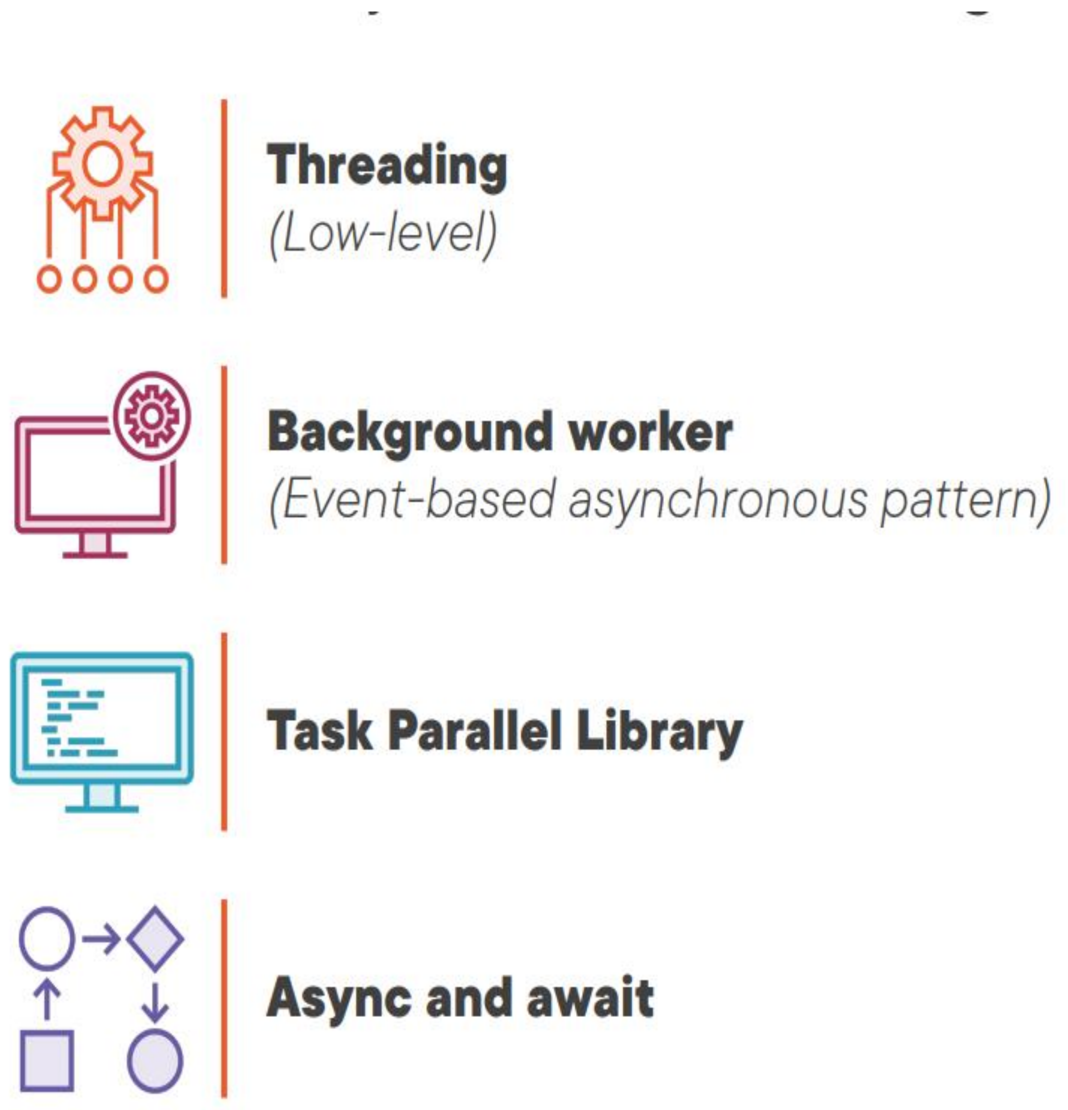
Asynchronous Programming for Dot Net Application



WPF, WinForms, .NET MAUI

Console

ASP.NET



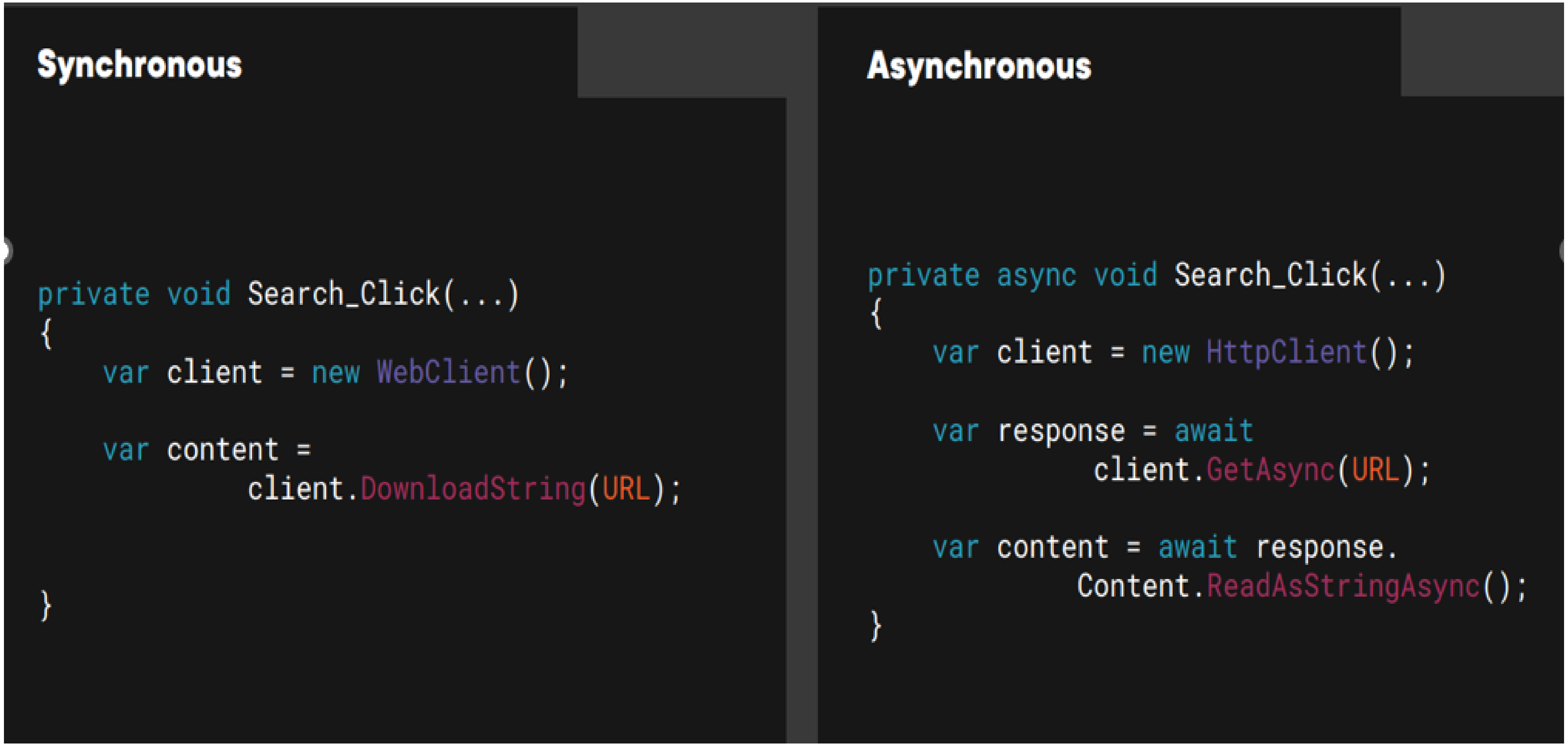
Threading
(Low-level)

Background worker
(Event-based asynchronous pattern)

Task Parallel Library

Async and await

Synchronous vs Asynchronous



Synchronous

```
private void Search_Click(...)
{
    var client = new WebClient();

    var content =
            client.DownloadString(URL);


}
```

Asynchronous

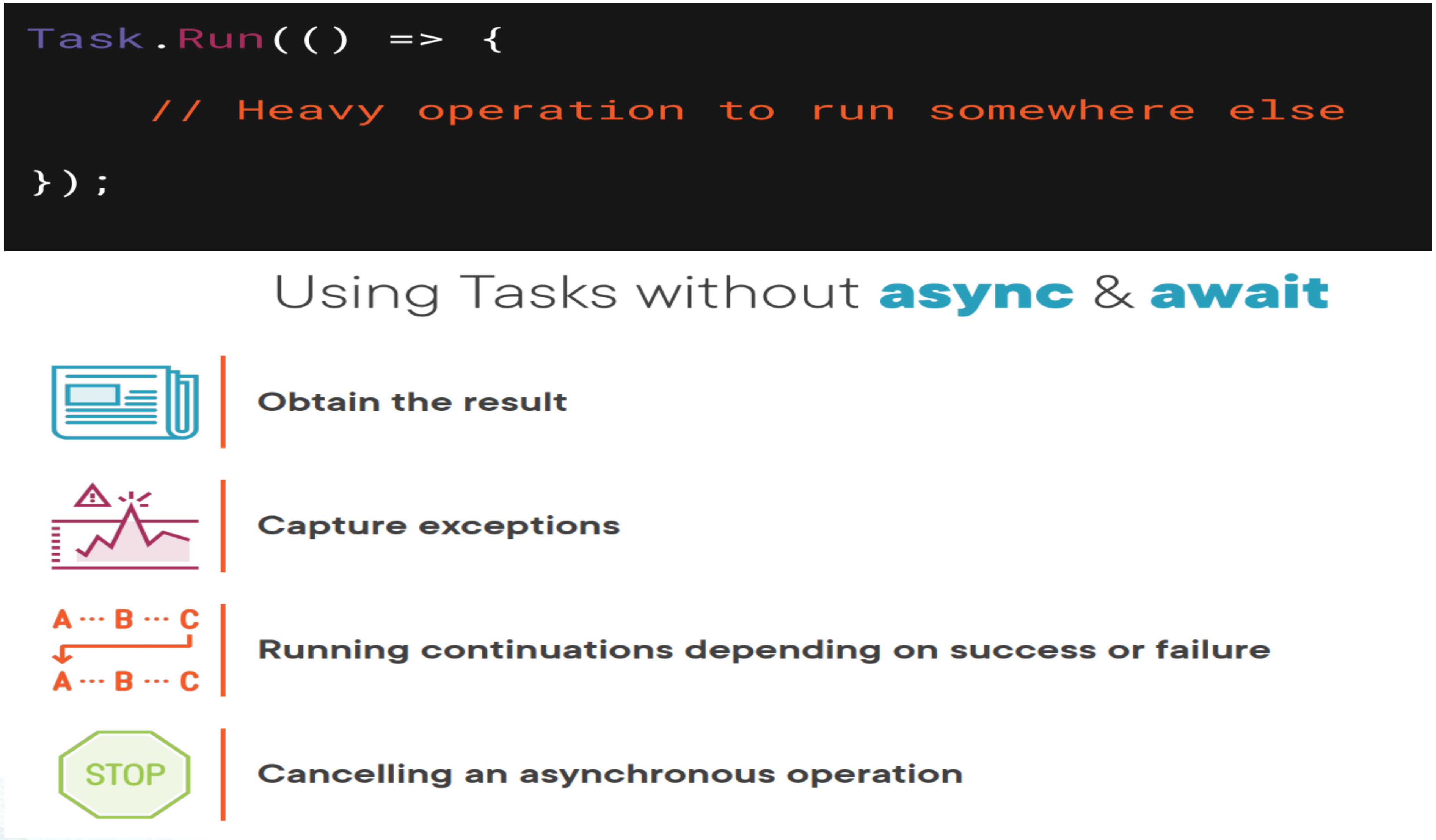
```
private async void Search_Click(...)
{
    var client = new HttpClient();

    var response = await
             client.GetAsync(URL);

    var content = await response.
            Content.ReadAsStringAsync();
}
```

An asynchronous operation occurs in parallel and relieves the calling thread of the work

Task Parallel Library

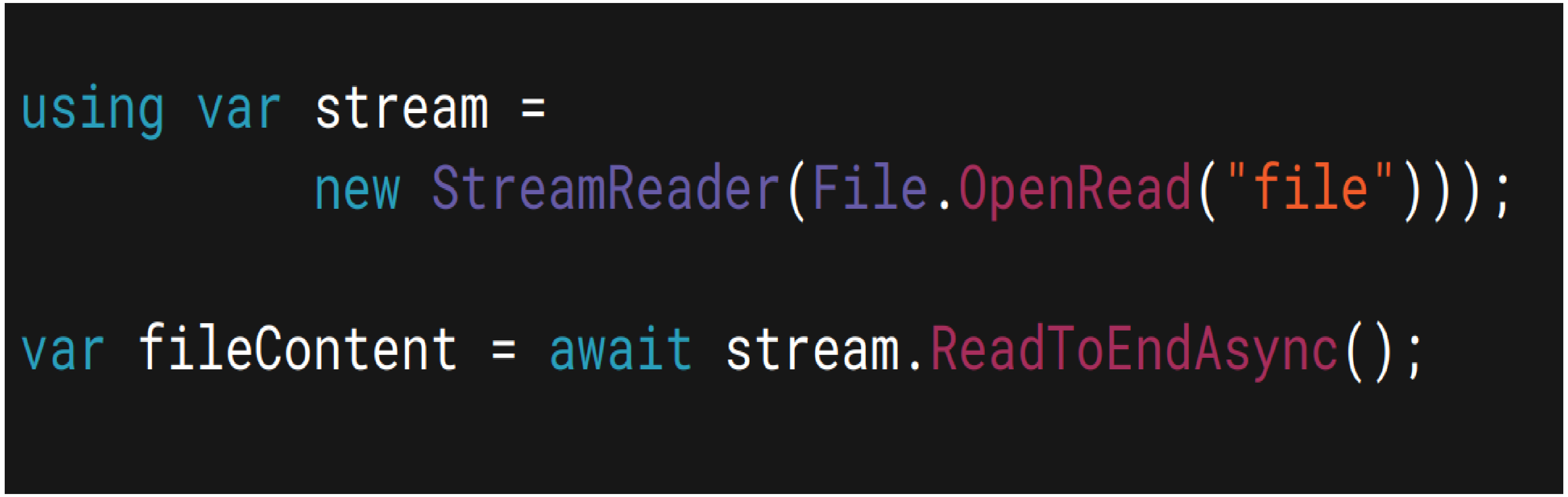


```
Task.Run(() => {
      // Heavy operation to run somewhere else
});
```

Using Tasks without **async** & **await**

- **Obtain the result**
- **Capture exceptions**
- **Running continuations depending on success or failure**
- **Cancelling an asynchronous operation**

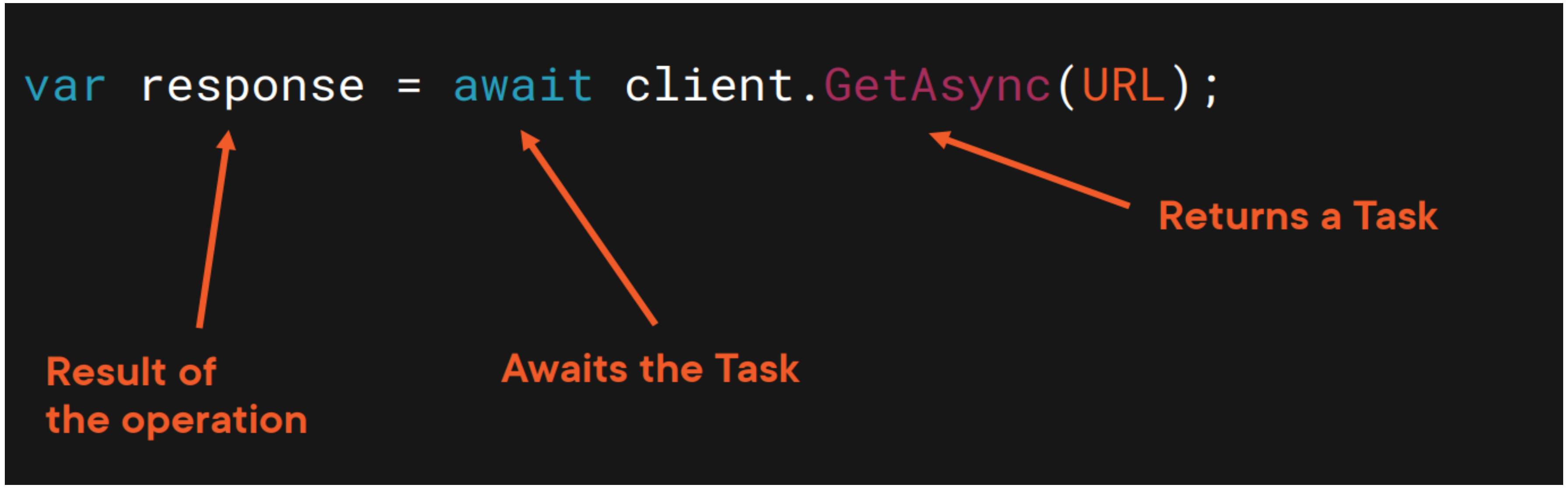
Read File Content Ashynchornoulsy



```
using var stream =
         new StreamReader(File.OpenRead("file")));

var fileContent = await stream.ReadToEndAsync();
```

Single Task



```
var response = await client.GetAsync(URL);
```

Task from the Task Parallel Library

Represents a single asynchronous operation

Functionality provided by Single Task

- Execute work on a different thread
- Get the result from the asynchronous operation
- Subscribe to when the operation is done by introducing a continuation
- It can tell you if there was an exception

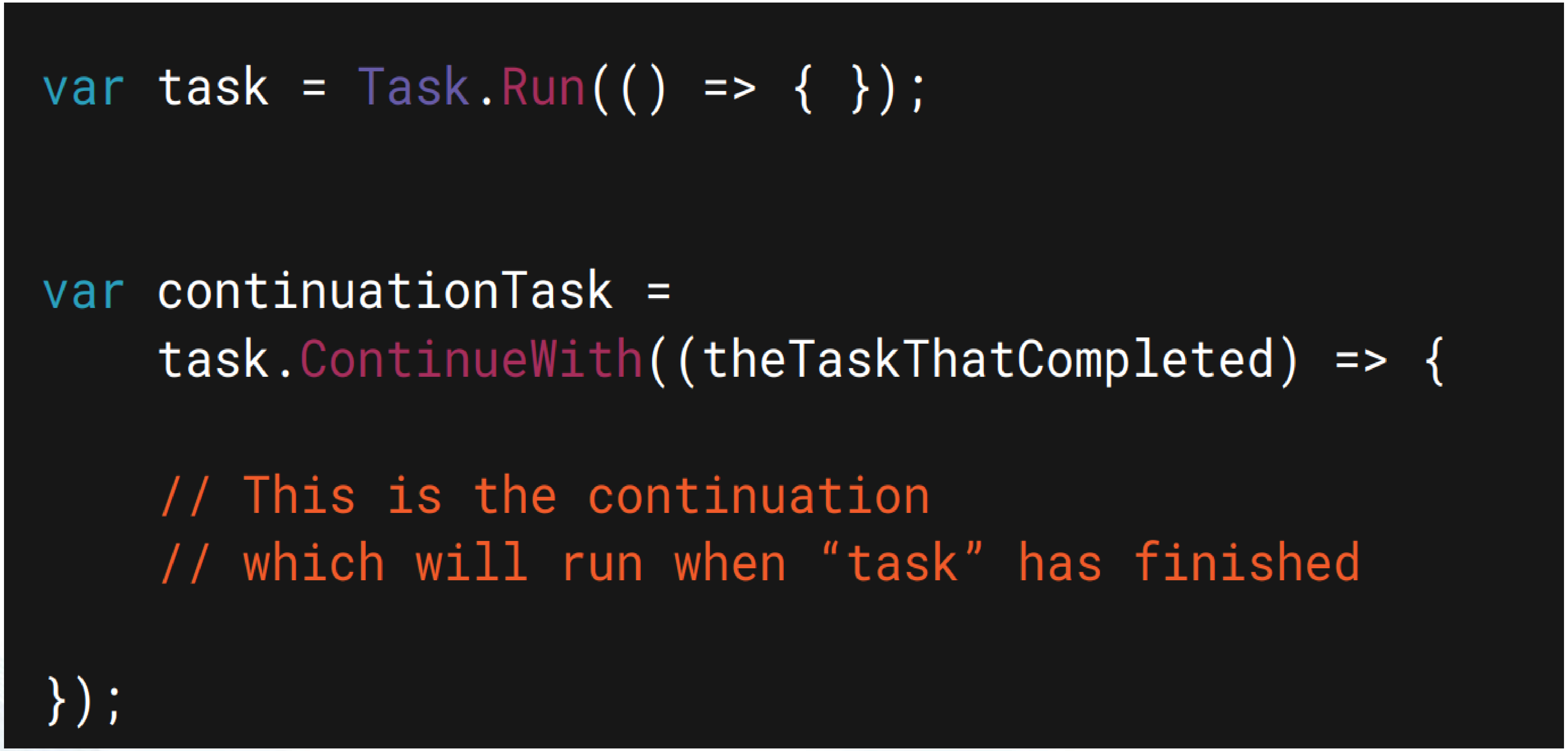
Project Overview

Thread Blocking : Task.Delay().Wait Vs Await Task.Delay()

Exception Handling: Shall we use Try-catch block around- Await or Task.IsFaulted

Task Cancellation : CancellationToken

Continuation



```
var task = Task.Run(() => { });


var continuationTask =
    task.ContinueWith((theTaskThatCompleted) => {

    // This is the continuation
    // which will run when “task” has finished

});
```

Thank You