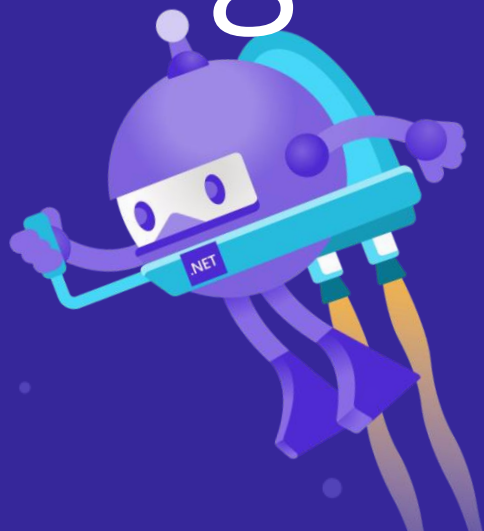
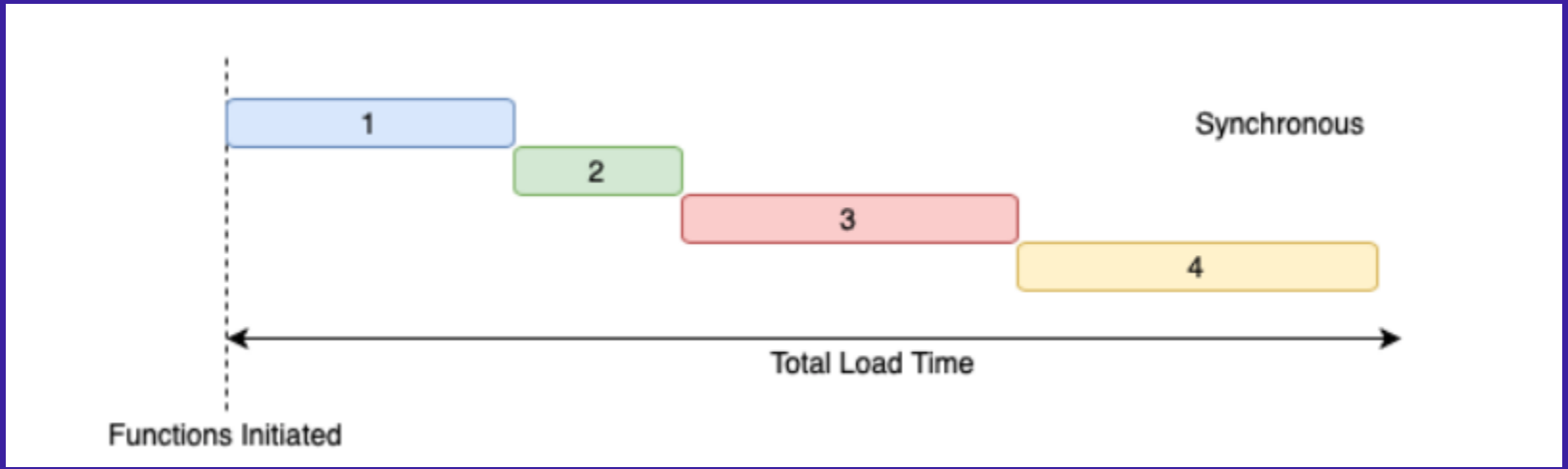
Day 20: Async Programming



Synchronous Programming

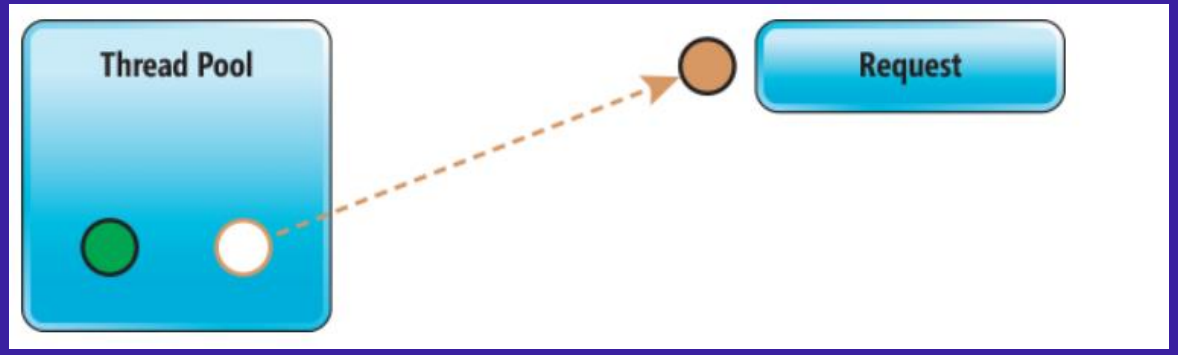
- In synchronous world, everything runs sequentially



- Typically, these applications will have only one thread

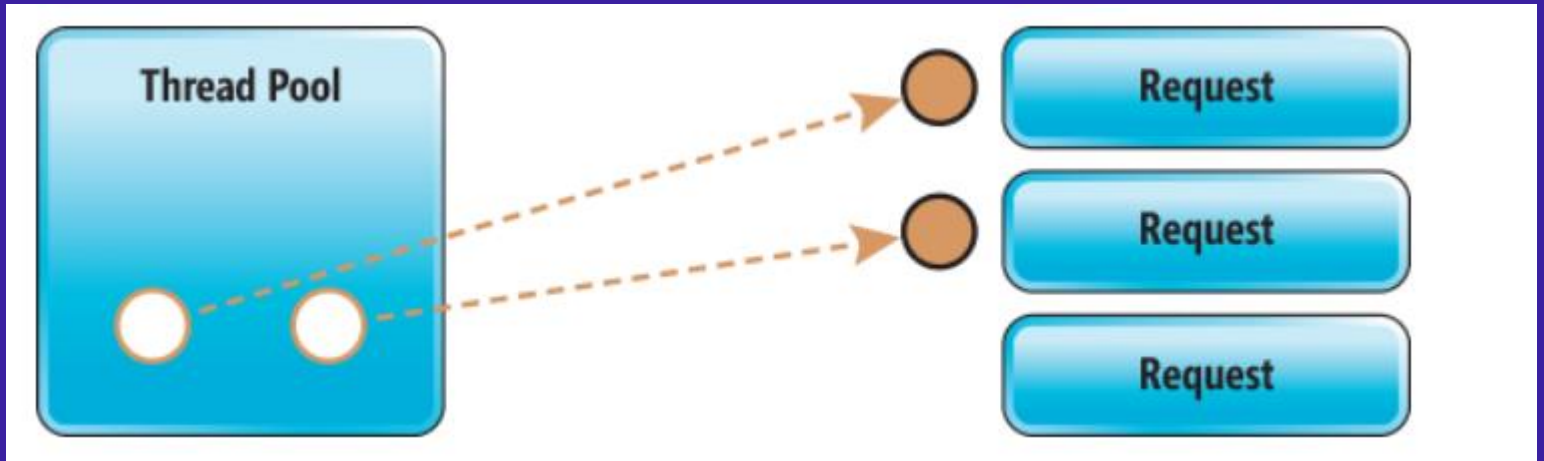
Synchronous Request Handling

1. When a request is initiated, runtime assigns a thread from the thread pool and assigns that to the request
2. Since its synchronous, whatever logic written inside that is called synchronously including calls to database or other external resources
3. This will block the thread until all the calls are executed



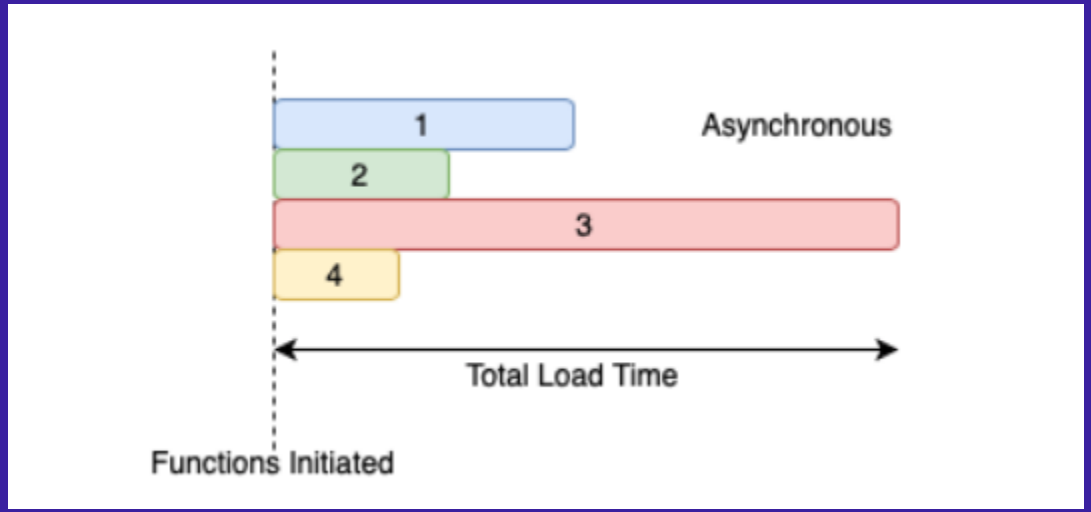
Synchronous Request Handling

1. This will work until our requests outrun the no of threads available in the pool
2. If there are no more threads available, then the requests will need to wait until one frees up



Asynchronous Programming

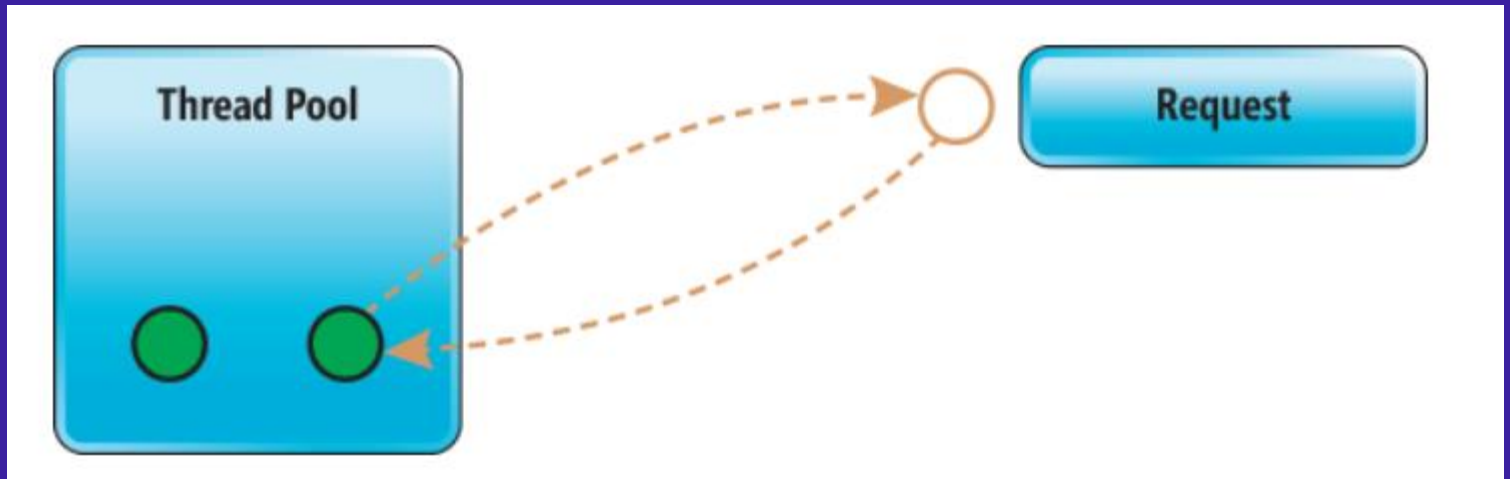
- In this type, the tasks are not run in a sequence



- Typically, it fires all the tasks and waits for their completion

Asynchronous Request Handling

1. When a request comes, one thread from the pool is assigned and the external request is called asynchronously
2. This causes the request thread to be returned to the pool until the call to the external resource is finished



Asynchronous Programming in C#

- Asynchronous is not multi threading
- Async Works on I/O Bound, Not CPU-Bound, Tasks
- Return Types
  - Task
    – represents work being done that will eventually return control to the caller
  - Task<T>
    - same as above, but returns an object of type T to the caller
  - Void
    – makes the method a true fire-and-forgot-one

Asynchronous model - Overview

- Core of Async programming is the *Task* and *Task<T>* objects
- Supported by *async* and *await* keywords
- For I/O bound operations
  - you await an operation that returns a Task or Task<T> inside of an async method.
- For CPU-bound operations
  - you await an operation that is started on a background thread with the Task.Run method.

async

- Async
  - Use the async modifier to specify that a method, lambda expression, or anonymous method is asynchronous
  - If you use this modifier on a method or expression, it's referred to as an async method.

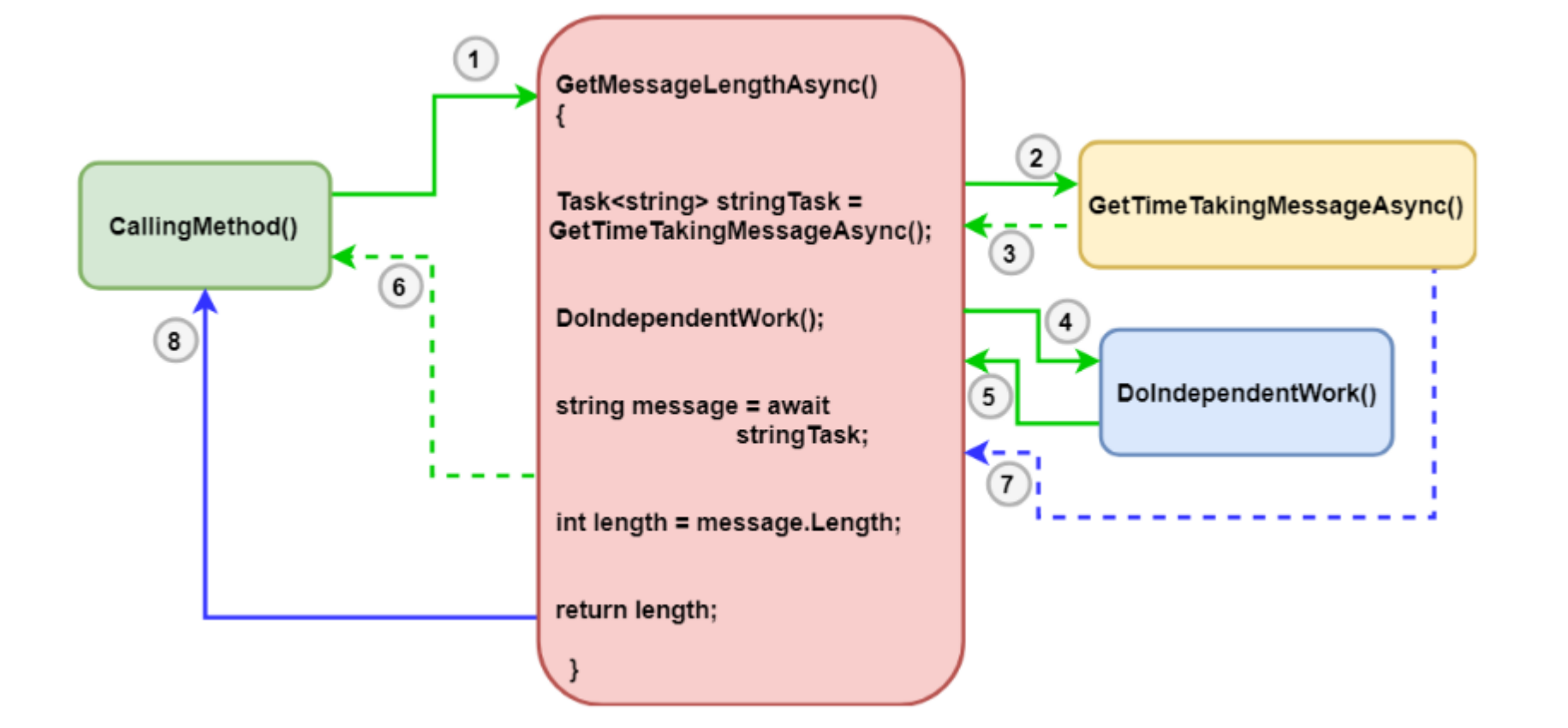
```
public async Task<int> ExampleMethodAsync()
{
    var httpClient = new HttpClient();
    int exampleInt = (await httpClient.GetStringAsync("http://msdn.microsoft.com")).Length;
    ResultsTextBox.Text += "Preparing to finish ExampleMethodAsync.\n";
    // After the following return statement, any method that's awaiting
    // ExampleMethodAsync (in this case, StartButton_Click) can get the
    // integer result.
    return exampleInt;
}
```

await

- The await operator suspends evaluation of the enclosing async method until the asynchronous operation represented by its operand completes
- When the asynchronous operation completes, the await operator returns the result of the operation, if any.

```
public static async Task Main()
{
    Task<int> downloading = DownloadDocsMainPageAsync();
    Console.WriteLine($"{nameof(Main)}: Launched downloading.");

    int bytesLoaded = await downloading;
    Console.WriteLine($"{nameof(Main)}: Downloaded {bytesLoaded} bytes.");
}

private static async Task<int> DownloadDocsMainPageAsync()
{
    Console.WriteLine($"{nameof(DownloadDocsMainPageAsync)}: About to start downloading.");

    var client = new HttpClient();
    byte[] content = await client.GetByteArrayAsync("https://docs.microsoft.com/en-us/");

    Console.WriteLine($"{nameof(DownloadDocsMainPageAsync)}: Finished downloading.");
    return content.Length;
}
```

What happens in an async method



References

- https://dotnet.microsoft.com/
- Learn .NET | Free tutorials, videos, courses, and more (microsoft.com)
- .NET Application Architecture Guides (microsoft.com)
- .NET Platform · GitHub
- GitHub - dotnet/aspnetcore:.
- .NET on Microsoft Learn | Microsoft Docs
- .NET Videos | Free videos from the .NET team and community (microsoft.com)

Thanks for joining!