I have an async joke.
But it only works in the future.

PROGRAMMING

Javascript Developer Finally Understands async-await Halfway Through an Orgy

Paul E. Morfism

Feb 15, 2024 — 1 min read

From Task.Run to Task.WhenAll

The Good, The Bad, And The async

Steven Giesel // DWX ‘25

```
{
  "name": "Steven Giesel",
  "websites": [
    "https://steven-giesel.com",
    "mailto:contact@steven-giesel.com",
    "https://bitspire.ch",
    "mailto:steven.giesel@bitspire.ch"
  ],
  "isMvp": true,
  "github": "linkdotnet",
  "linkedIn": "Steven Giesel",
  "blueSky": "@steven-giesel.com",
  "x": null
}
```

What is the runtime of the snippets?

Snippet A

```
var a = Task.Delay(1_000);
var b = Task.Delay(1_000);
var c = Task.Delay(1_000);
var d = Task.Delay(1_000);
var e = Task.Delay(1_000);

await a;
await b;
await c;
await d;
await e;
```

Snippet B

```
await Task.Delay(1_000);
await Task.Delay(1_000);
await Task.Delay(1_000);
await Task.Delay(1_000);
await Task.Delay(1_000);
```

A. Snippet A and B run 5 seconds

B. Snippet A and B run 1 second

C. Snippet A runs 5 seconds and Snippet B runs 1 second

D. Snippet A runs 1 second and Snippet B runs 5 seconds

What is the runtime of the snippets?

Snippet A

```
var a = Task.Delay(1_000);
var b = Task.Delay(1_000);
var c = Task.Delay(1_000);
var d = Task.Delay(1_000);
var e = Task.Delay(1_000);

await a;
await b;
await c;
await d;
await e;
```

Snippet B

```
await Task.Delay(1_000);
await Task.Delay(1_000);
await Task.Delay(1_000);
await Task.Delay(1_000);
await Task.Delay(1_000);
```

A. Snippet A and B run 5 seconds

B. Snippet A and B run 1 second

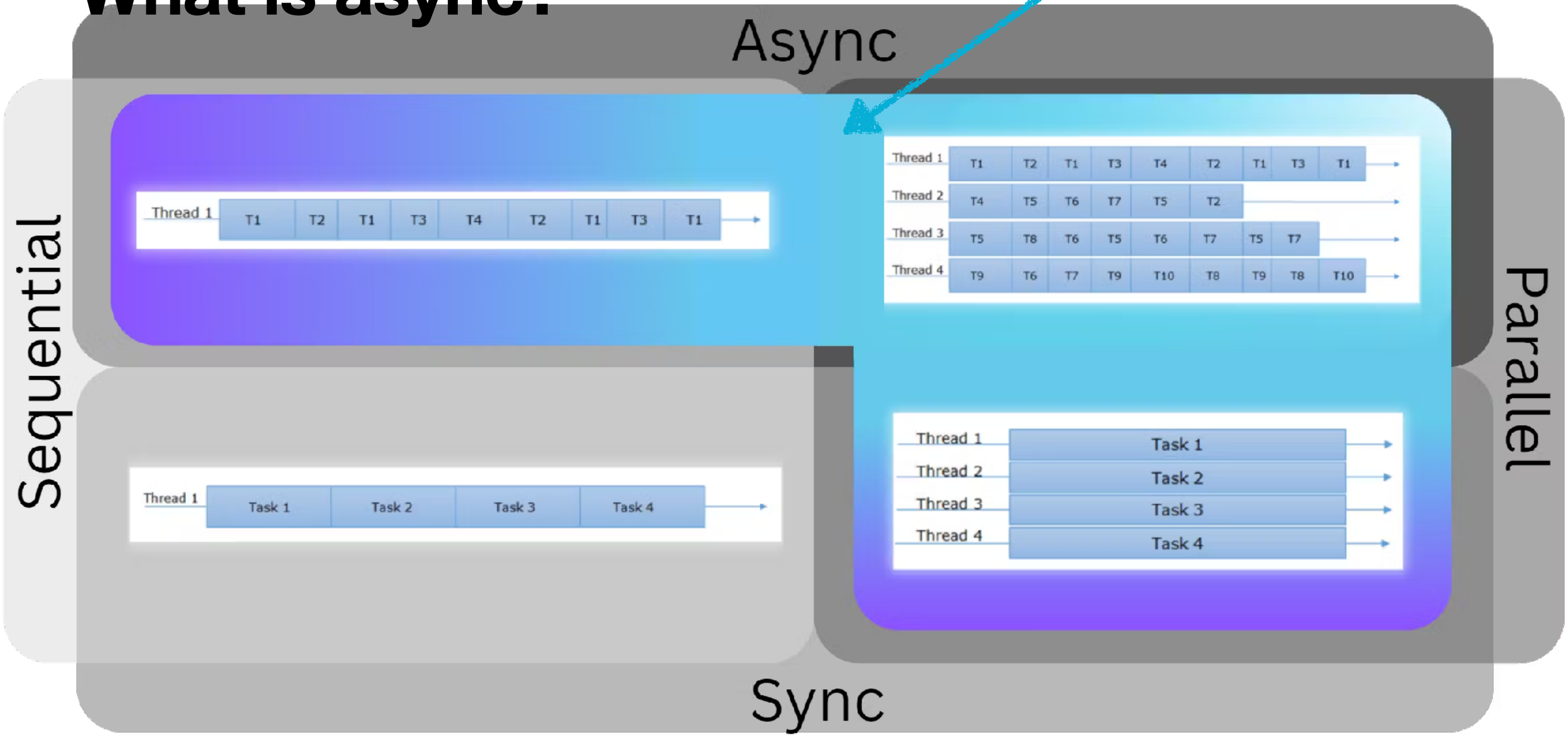
C. Snippet A runs 5 seconds and Snippet B runs 1 second

D. Snippet A runs 1 second and Snippet B runs 5 seconds

What is async?

What is async?

Concurrency

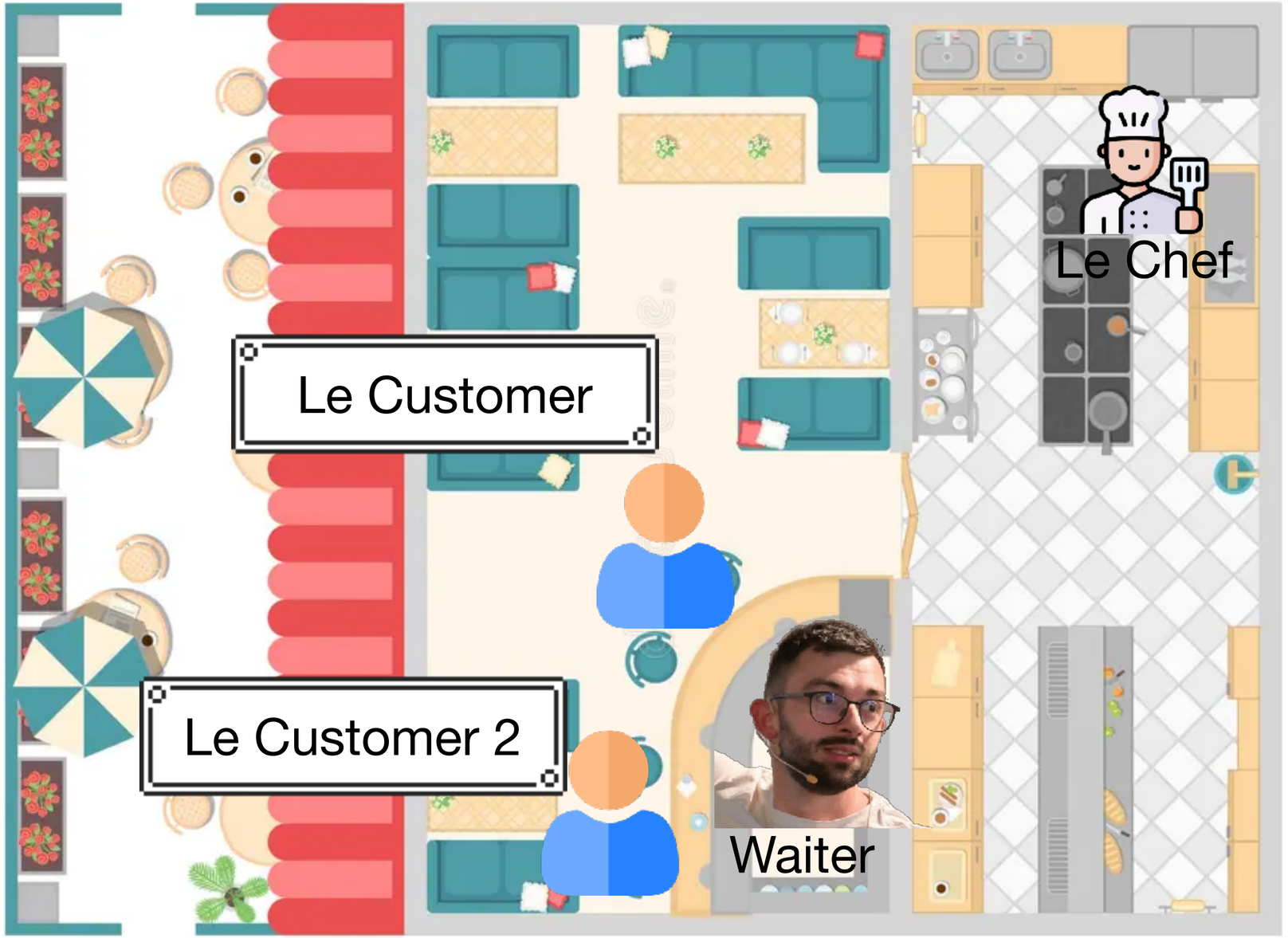


What is async/await?*

* I/O Bound operations

What is async/await?

SYNC version

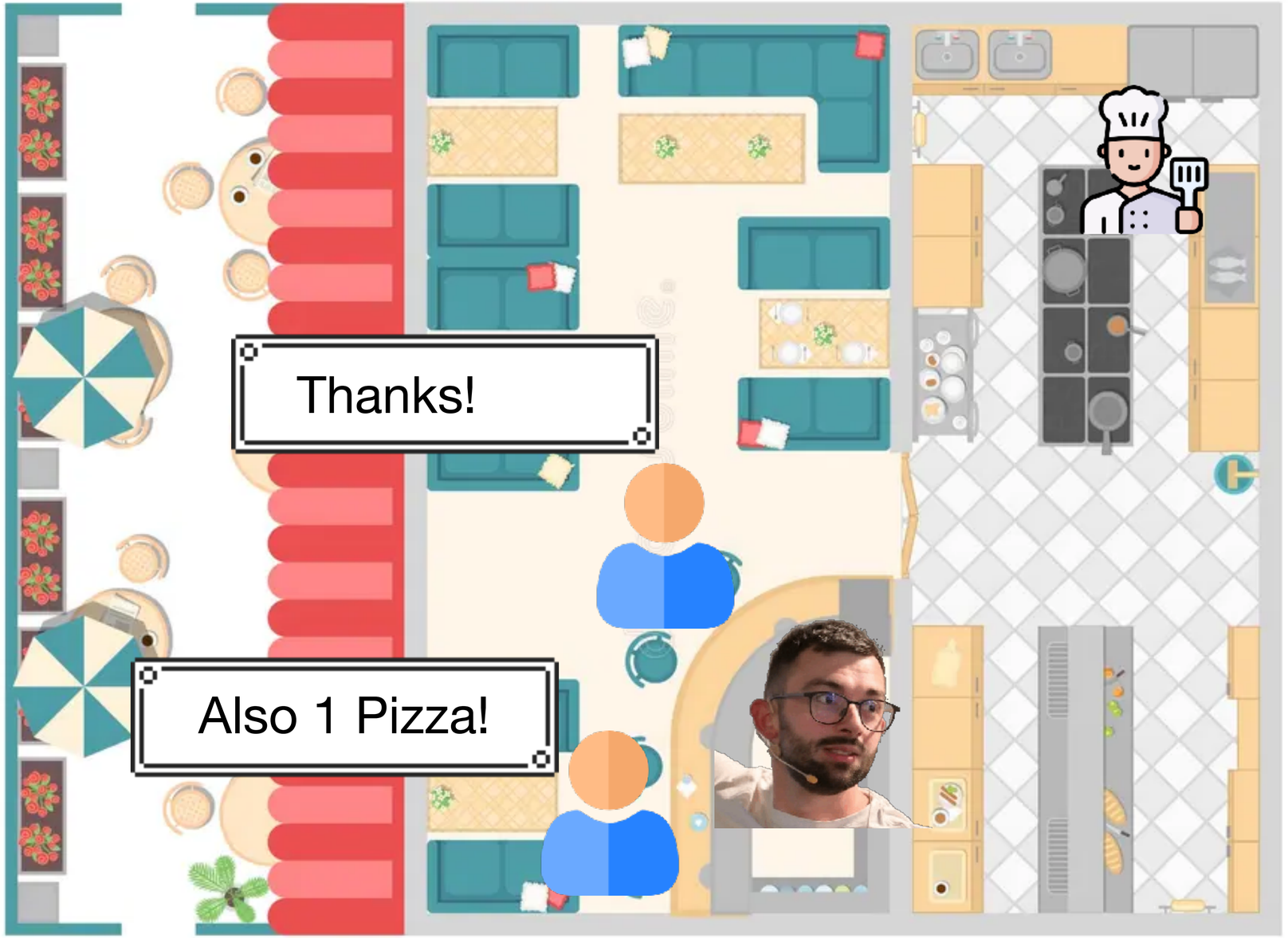


What is async/await?

SYNC version

What is async/await?

SYNC version

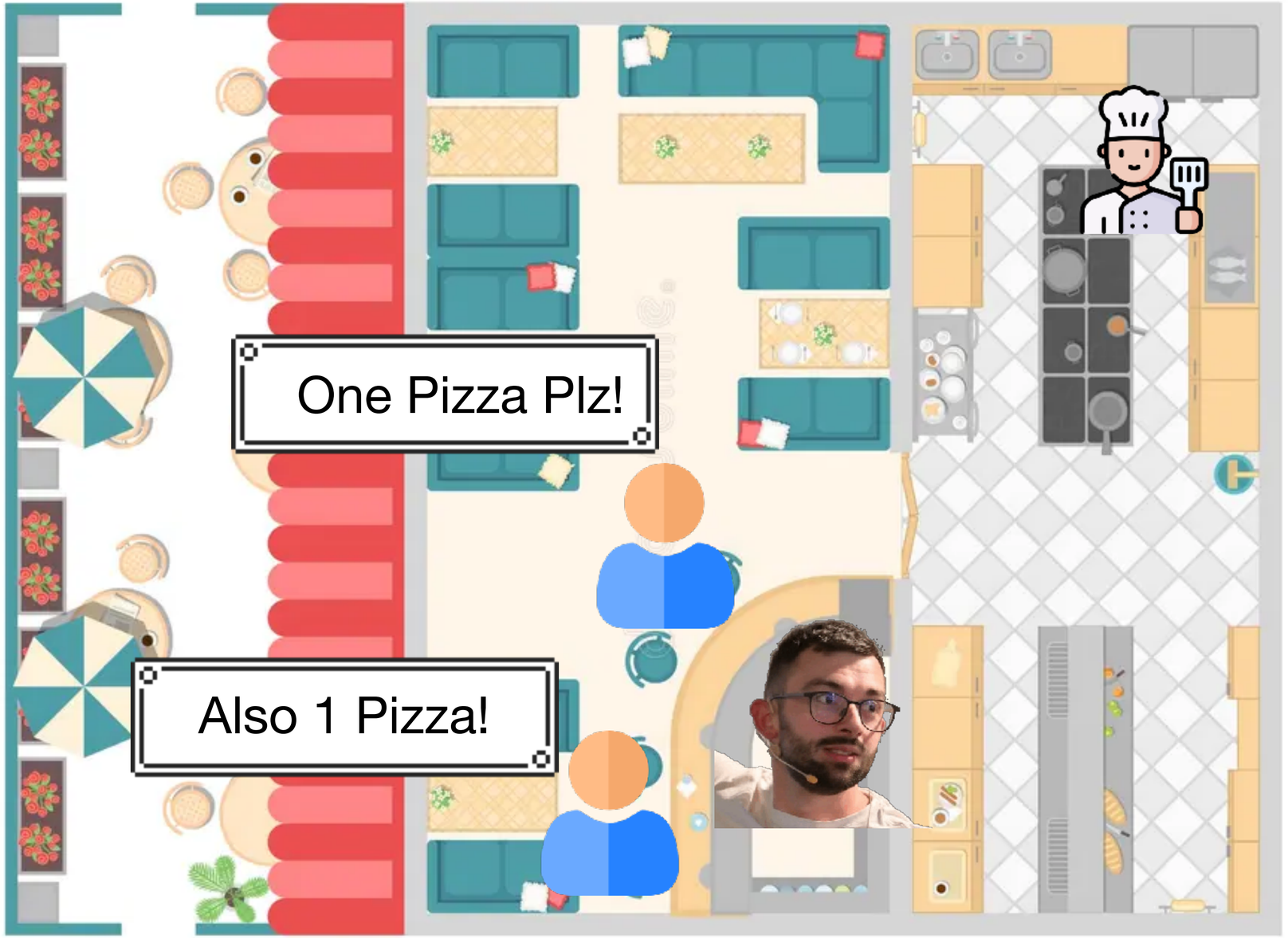


What is async/await?

ASYNC version

What is async/await?

ASYNC version



What is async/await?

ASYNC version

A (HTTP) Request

Workerthread (Kestrel / IIS) aka your ASP.NET Backend

Database Call / HTTP Call

*

What is a SynchronizationContext?

*

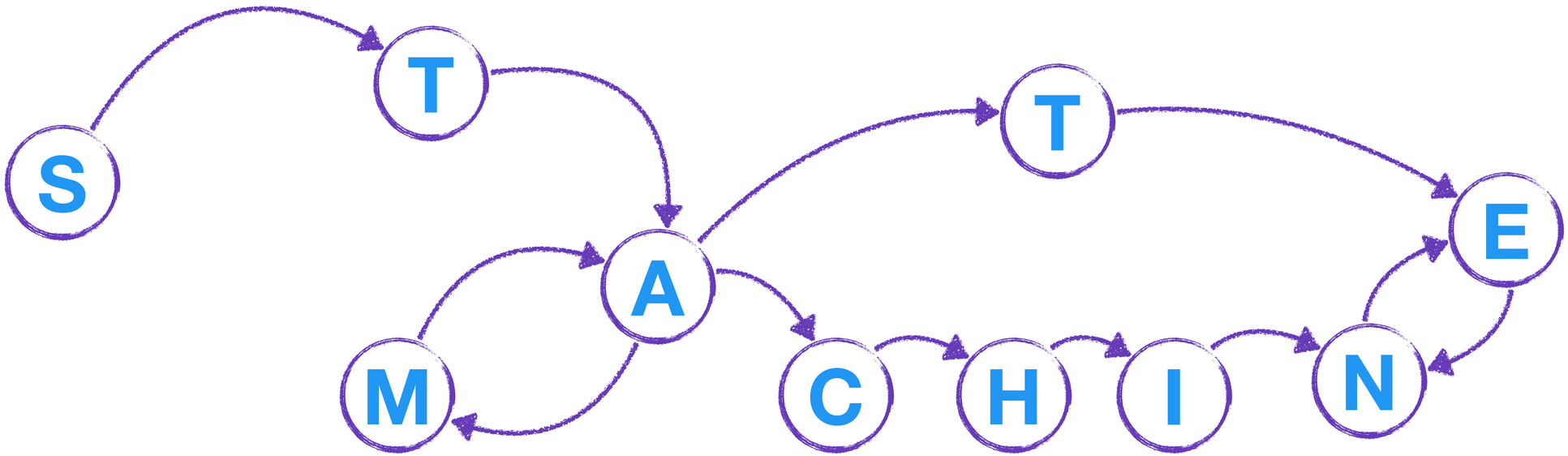
What is a SynchronizationContext?

*

What is a SynchronizationContext?

*

What is async/await?



What is async/await?

```
public async Task<IReadOnlyCollection<PokemonDto>> GetAllAsync()
{
    using var client = new HttpClient();
    client.BaseAddress = new Uri("https://pokeapi.co/api/v2/pokemon/");
    var response = await client.GetAsync("");
    response.EnsureSuccessStatusCode();
    var content = await response.Content.ReadAsStringAsync();
    var dto = JsonSerializer.Deserialize<ListPokemonDto>(content);
    return dto.Results;
}
```

1. Cut function at await boarder

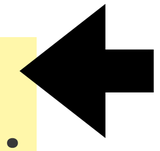
What is async/await?

```
public async Task<IReadOnlyCollection<PokemonDto>> GetAllAsync()
{
    using var client = new HttpClient();
    client.BaseAddress = new Uri("https://pokeapi.co/api/v2/pokemon/");
    var response = await client.GetAsync("");
    response.EnsureSuccessStatusCode();
    var content = await response.Content.ReadAsStringAsync();
    var dto = JsonSerializer.Deserialize<ListPokemonDto>(content);
    return dto.Results;
}
```

1. Cut function at await boarder

What is async/await?

```
using var client = new HttpClient();
client.BaseAddress = new Uri("https://pokeapi.co/api/v2/pokemon/");
var response = await client.GetAsync("");
```



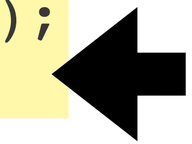
```
response.EnsureSuccessStatusCode();
var content = await response.Content.ReadAsStringAsync();
```

```
var dto = JsonSerializer.Deserialize<ListPokemonDto>(content);
return dto.Results;
```

2 Execute synchronous part until await

What is async/await?

```
using var client = new HttpClient();
client.BaseAddress = new Uri("https://pokeapi.co/api/v2/pokemon/");
var response = await client.GetAsync("");
```



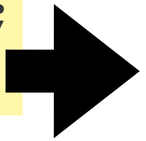
```
response.EnsureSuccessStatusCode();
var content = await response.Content.ReadAsStringAsync();
```

```
var dto = JsonSerializer.Deserialize<ListPokemonDto>(content);
return dto.Results;
```

3 On await return to caller

What is async/await?

```
using var client = new HttpClient();
client.BaseAddress = new Uri("https://pokeapi.co/api/v2/pokemon/");
var response = await client.GetAsync("");
```



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

```
var dto = JsonSerializer.Deserialize<ListPokemonDto>(content);
return dto.Results;
```

4 Continue when “inner” await is done. Then Rinse&Repeat

Where is the state stored?

Where is the state stored?

```
public async Task<IReadOnlyCollection<PokemonDto>> GetAllAsync()
{
    using var client = new HttpClient();
    client.BaseAddress = new Uri("https://pokeapi.co/api/v2/pokemon/");
    var response = await client.GetAsync("");
    response.EnsureSuccessStatusCode();
    var content = await response.Content.ReadAsStringAsync();
    var dto = JsonSerializer.Deserialize<ListPokemonDto>(content);
    return dto.Results;
}
```

Where is the state stored?

```
public struct GetAllAsyncStateMachine : IAsyncStateMachine
{
    public int state;  // Tracks the current state of the machine
    private AsyncTaskMethodBuilder<IReadOnlyCollection<PokemonDto>> methodBuilder;  // To build and complete the Task
    private HttpClient client;  // HttpClient instance
    private HttpResponseMessage response;  // Stores the response from GetAsync
    private TaskAwaiter<HttpResponseMessage> getAsyncAwaiter;  // Awaiter for client.GetAsync("")
    private TaskAwaiter<string> readAsStringAsyncAwaiter;  // Awaiter for response.Content.ReadAsStringAsync()
    private string content;  // Stores the content from ReadAsStringAsync
    private IReadOnlyCollection<PokemonDto> result;  // Stores the final result

    public void MoveNext()
    {
        try
        {
            switch (state)
            {
                case 0:  // Initial state
                    client = new HttpClient();
                    client.BaseAddress = new Uri("https://pokeapi.co/api/v2/pokemon/");
                    getAsyncAwaiter = client.GetAsync("").GetAwaiter();

                    if (!getAsyncAwaiter.IsCompleted)
                    {
                        state = 1;  // Update state before awaiting
                        methodBuilder.AwaitOnCompleted(ref getAsyncAwaiter, ref this);
                        return;
                    }
                    goto case 1;

                case 1:  // Resumed after client.GetAsync("")
                    response = getAsyncAwaiter.GetResult();
                    response.EnsureSuccessStatusCode();  // Ensure the response was successful

                    readAsStringAsyncAwaiter = response.Content.ReadAsStringAsync().GetAwaiter();
                    if (!readAsStringAsyncAwaiter.IsCompleted)
                    {
                        state = 2;  // Update state before awaiting
                        methodBuilder.AwaitOnCompleted(ref readAsStringAsyncAwaiter, ref this);
                        return;
                    }
                    goto case 2;

                case 2:  // Resumed after response.Content.ReadAsStringAsync()
                    content = readAsStringAsyncAwaiter.GetResult();
                    var dto = JsonSerializer.Deserialize<ListPokemonDto>(content);
                    result = dto.Results;

                    // Task is completed successfully with the result
                    methodBuilder.SetResult(result);
                    break;
            }
        }
        catch (Exception ex)
        {
            // Set the Task as faulted with the caught exception
            methodBuilder.SetException(ex);
        }
        finally
        {
            client?.Dispose();  // Ensure HttpClient is disposed
        }
    }
}
```

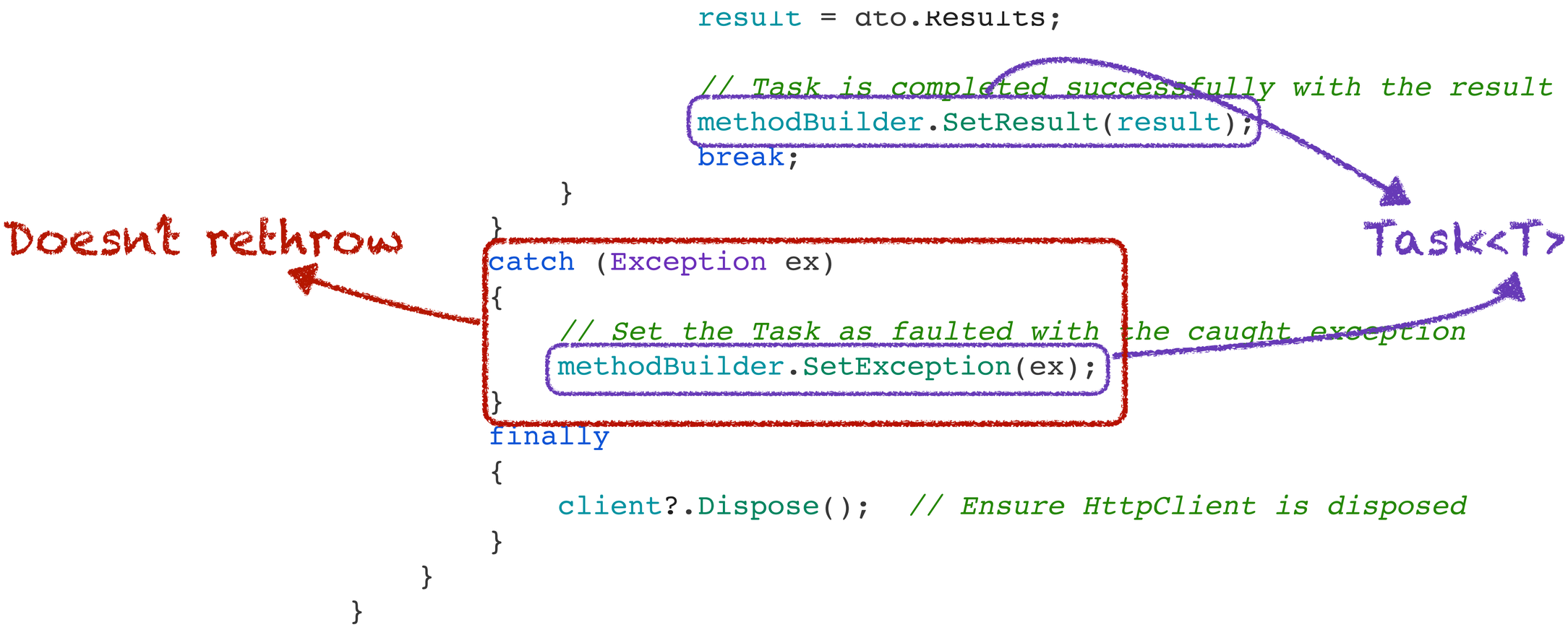
Where is the state stored?

```
public struct GetAllAsyncStateMachine : IAsyncStateMachine
{
    public int state;  // Tracks the current state of the machine
    private AsyncTaskMethodBuilder<IReadOnlyCollection<PokemonDto>> methodBuilder;  // To build and complete the Task
    private HttpClient client;  // HttpClient instance
    private HttpResponseMessage response;  // Stores the response from GetAsync
    private TaskAwaiter<HttpResponseMessage> getAsyncAwaiter;  // Awaiter for client.GetAsync("")
    private TaskAwaiter<string> readAsStringAsyncAwaiter;  // Awaiter for response.Content.ReadAsStringAsync()
    private string content;  // Stores the content from ReadAsStringAsync
    private IReadOnlyCollection<PokemonDto> result;  // Stores the final result

    public void MoveNext()
    {
        try
        {
            switch (state)
            {
                case 0:  // Initial state
                    client = new HttpClient();
                    client.BaseAddress = new Uri("https://pokeapi.co/api/v2/pokemon/");
                    getAsyncAwaiter = client.GetAsync("").GetAwaiter();

                    if (!getAsyncAwaiter.IsCompleted)
                    {
                        state = 1;  // Update state before awaiting
                        methodBuilder.AwaitOnCompleted(ref getAsyncAwaiter, ref this);
                        return;
                    }
                    goto case 1;

                case 1:  // Resumed after client.GetAsync("")
                    response = getAsyncAwaiter.GetResult();
                    response.EnsureSuccessStatusCode();  // Ensure the response was successful

                    readAsStringAsyncAwaiter = response.Content.ReadAsStringAsync().GetAwaiter();
                    if (!readAsStringAsyncAwaiter.IsCompleted)
                    {
```

Where is the state stored?

```
switch (state)
{
    case 0:  // Initial state
        client = new HttpClient();
        client.BaseAddress = new Uri("https://pokeapi.co/api/v2/pokemon/");
        getAsyncAwaiter = client.GetAsync("").GetAwaiter();

        if (!getAsyncAwaiter.IsCompleted)
        {
            state = 1;  // Update state before awaiting
            methodBuilder.AwaitOnCompleted(ref getAsyncAwaiter, ref this);
            return;
        }
        goto case 1;

    case 1:  // Resumed after client.GetAsync("")
        response = getAsyncAwaiter.GetResult();
        response.EnsureSuccessStatusCode();  // Ensure the response was successful

        readAsStringAsyncAwaiter = response.Content.ReadAsStringAsync().GetAwaiter();
        if (!readAsStringAsyncAwaiter.IsCompleted)
        {
            state = 2;  // Update state before awaiting
            methodBuilder.AwaitOnCompleted(ref readAsStringAsyncAwaiter, ref this);
            return;
        }
        goto case 2;

    case 2:  // Resumed after response.Content.ReadAsStringAsync()
        content = readAsStringAsyncAwaiter.GetResult();
        var dto = JsonSerializer.Deserialize<ListPokemonDto>(content);
        result = dto.Results;

        // Task is completed successfully with the result
        methodBuilder.SetResult(result);
        break;
}
```

Where is the state stored?




WON'T SOMEBODY PLEASE
THINK OF THE
Performance

What is async/await?

Q: How much bytes does one async allocate?

A. ~10 bytes

B. ~100 bytes

C. ~1000 bytes

D.~10000 bytes

Performance

```
[MemoryDiagnoser]
public class TaskBenchmarks
{
    private static readonly Task<string> IamAString = Task.FromResult(".DWX 2025”);

    [Benchmark]
    public async Task<string> GetStringWithoutAsync()
        => await GetStringWithoutAsyncCore();

    [Benchmark]
    public async Task<string> GetStringWithAsync()
        => await GetStringWithAsyncCore();

    private Task<string> GetStringWithoutAsyncCore()
        => IamAString;

    private async Task<string> GetStringWithAsyncCore()
        => await IamAString;
}
```

What is async/await?

Q: How much bytes does one async allocate?

A. ~10 bytes

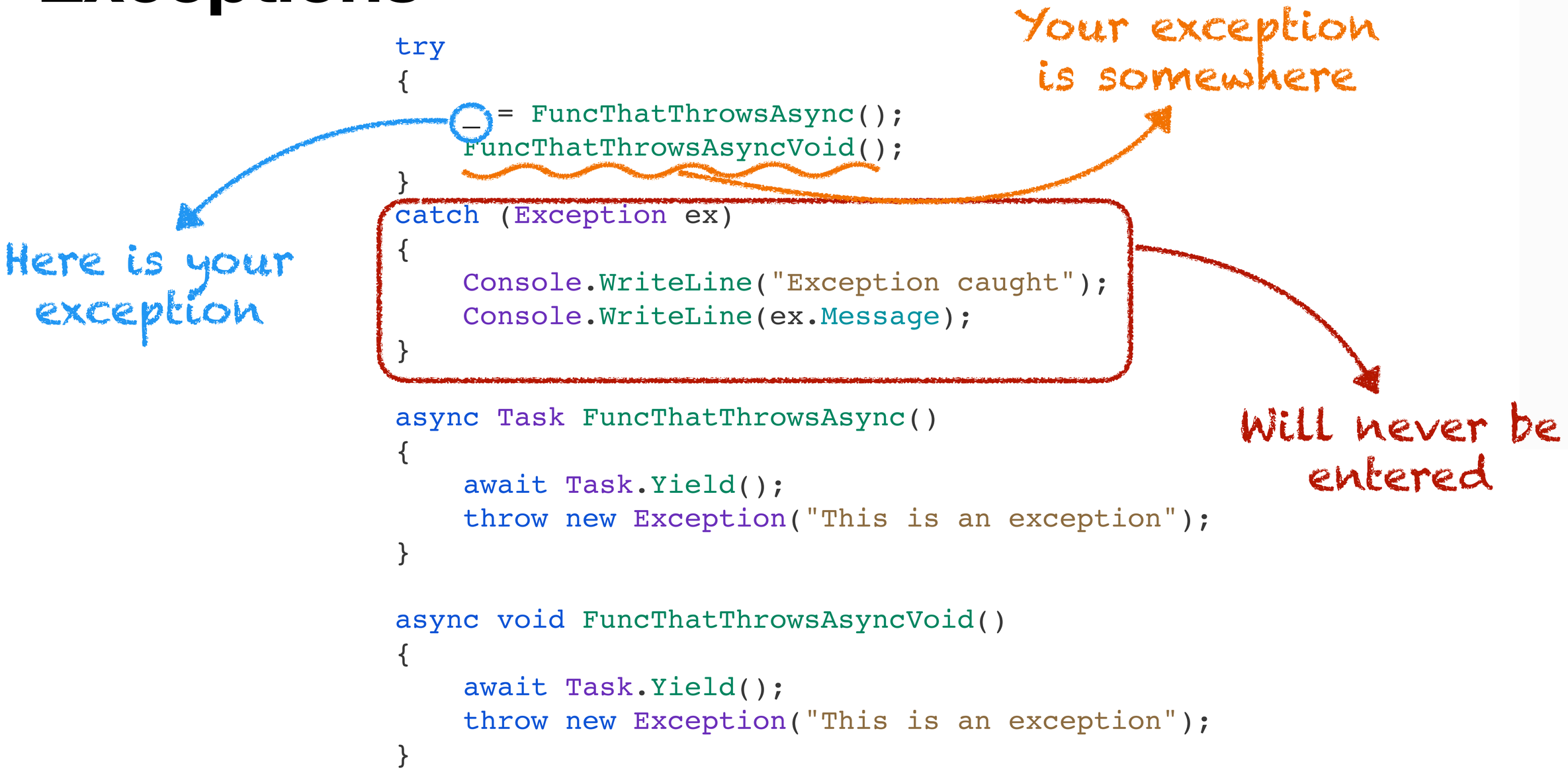
B. ~100 bytes

C. ~1000 bytes

D.~10000 bytes

Exceptions

Exceptions




```
try
{
    _ = FuncThatThrowsAsync();
    FuncThatThrowsAsyncVoid();
}
catch (Exception ex)
{
    Console.WriteLine("Exception caught");
    Console.WriteLine(ex.Message);
}

async Task FuncThatThrowsAsync()
{
    await Task.Yield();
    throw new Exception("This is an exception");
}

async void FuncThatThrowsAsyncVoid()
{
    await Task.Yield();
    throw new Exception("This is an exception");
}
```

Exceptions

```
try
{
    await DoWorkWithoutAwaitAsync();
}
catch (Exception e)
{
    Console.WriteLine(e);
}

static Task DoWorkWithoutAwaitAsync()
    => ThrowExceptionAsync();

static async Task ThrowExceptionAsync()
{
    await Task.Yield();
    throw new Exception("Hey");
}
```

```
System.Exception: Hey
   at Program.<<Main>$>g__ThrowExceptionAsync|0_1() in
/Users/stgi/repos/Benchmark/Program.cs:line 19
   at Program.<Main>$(String[] args) in
/Users/stgi/repos/Benchmark/Program.cs:line 6
```

Exceptions

```
Task<string> GetContentFromUrlAsync(string url)
{
    // Don't do this! Creating new HttpClients
    // is expensive and has other caveats
    // This is for the sake of demonstration
    using var client = new HttpClient();
    return client.GetStringAsync(url);
}
```

Exceptions

```
async Task<string> GetContentFromUrlAsync(string url)
{
    // Don't do this! Creating new HttpClients
    // is expensive and has other caveats
    // This is for the sake of demonstration
    using var client = new HttpClient();
    return await client.GetStringAsync(url);
}
```

Other Pitfalls

Other Pitfalls

async void the 2nd

```
var ids = new List<int>();
// ...
ids.ForEach(id => _myRepo.UpdateAsync(id));
```

Other Pitfalls

async void the 2nd

```
var ids = new List<int>();
// ...
ids.ForEach(async id => await _myRepo.UpdateAsync(id));
```

Other Pitfalls

Task vs await Task

```
await MyLongRunningTaskAsync();
```

Other Pitfalls

Task vs await Task

```
await MyLongRunningTaskAsync();
```


```
Task task = MyLongRunningTaskAsync();
await task;
```

Other Pitfalls

Task vs await Task

```
await MyLongRunningTaskAsync();
```

```
Task task = MyLongRunningTaskAsync();
await task;
```

```
MyLongRunningTask();
// Also:
// MyLongRunningTaskAsync().GetAwaiter().GetResult();
```

General Tips

General Tips

- Async != Parallel
- Task != Thread -> **THERE IS NO THREAD*** (for I/O Bound async/await)
  - Technologies like **Direct Memory Access** and **Interrupt Service Routine** are the magic ingredient
- As a rule of thumb: Try not to elide the async keyword

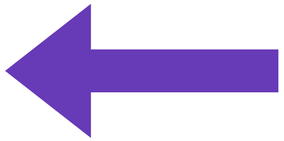
*https://blog.stephencleary.com/2013/11/there-is-no-thread.html

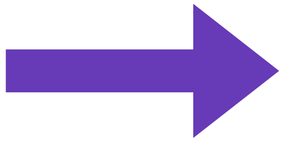
General Tips

- Task.Run is meant for for CPU bound operations
  - Be cautious when using this in ASP.NET (Core) as you decrease your scalability. For trivial things: Task.FromResult
  - ASP.NET (Core): Each request is one thread, but with Task.Run you will remove one thread from the ThreadPool
  - Your thread pool size is finite ;) and depends on various factors like virtual address space
  - Just because you can have hundreds of threads from the thread pool doesn't mean they can be executed "in parallel"

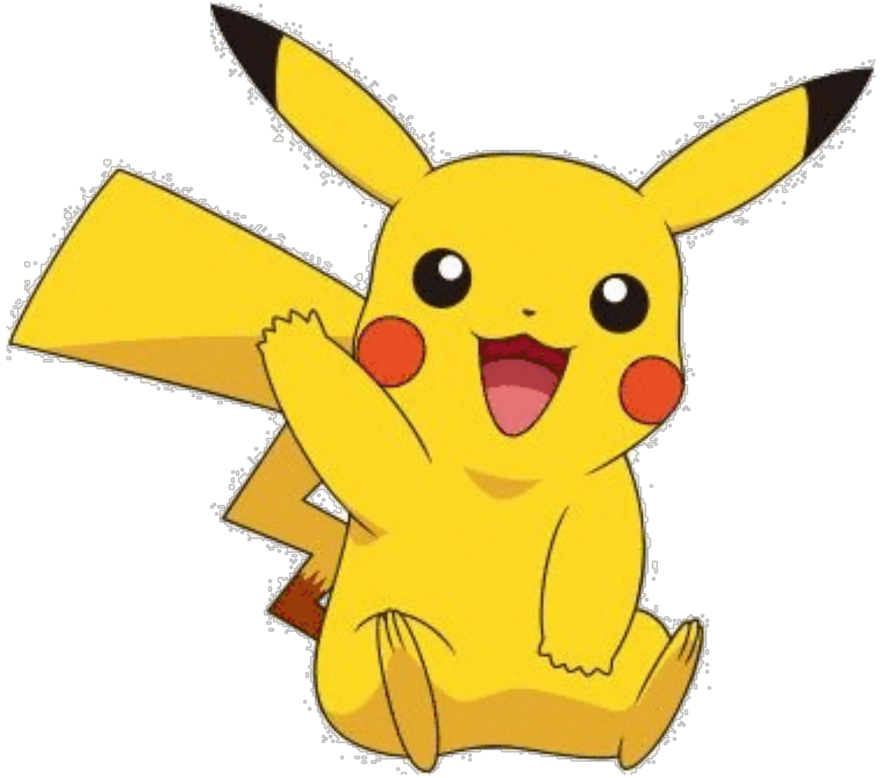
General Tips

- Invest some time into **CancellationToken**
- Do we have time for **ValueTask?**





Thanks & Goodbye!



ValueTask

```
public readonly struct ValueTask<T>
{
    public readonly T? SyncValue { get; }
    public readonly Task<T>? AsyncValue { get; }
}
```

ValueTask

```
public readonly struct ValueTask<T>
{
    public readonly T? SyncValue { get; }
    public readonly Task<T>? AsyncValue { get; }
}
```

```
public class WeatherService
{
    private readonly Dictionary<string, WeatherData> _weatherCache;
    private readonly IWeatherRepository weatherRepository;

    // ...
    public async ValueTask<WeatherData> GetWeatherForCityAsync(string city)
    {
        if (_weatherCache.TryGetValue(city, out var weatherData))
        {
            return weatherData;
        }

        return await weatherRepository.GetForCityAsync(city);
    }
}
```

Thanks & Goodbye!