

Welcome

Rearchitect your code
towards `async/await`



Solution Architect
Enthusiastic Software Engineer
Microsoft MVP for systems integration

@danielmarbach
particular.net/blog
planetgeek.ch

Goals

target

Why async is **the future**

How to **gradually move** your
code towards async / await

The toolbelt for an **async ninja**

Premise



Intro

Phases

WrapUp



Intro

Phases

WrapUp

The die
is

cast

javascript

ES2015

```
async function chainAnimationsPromise(elem, animations)
{
    let ret = null;
    try {
        for(const anim of animations) {
            ret = await anim(elem);
        }
    } catch(e) { /* ignore and keep going */ }
    return ret;
}
```

```
$ npm install babel-plugin-syntax-async-functions
```

```
$ npm install babel-plugin-transform-async-to-generator
```


httpClient

```
using (var client = new HttpClient()) {  
    var response = await  
        client.GetAsync("api/products/1");  
    if (response.IsSuccessStatusCode)  
    {  
        var product = await  
            response.Content.ReadAsAsync<Product>();  
    }  
}
```

Azure SDK

```
var queryable =  
client.CreateDocumentQuery<Entity>(...)  
    .AsDocumentQuery();
```

```
while (queryable.HasMoreResults)  
{  
    foreach (var e in await  
queryable.ExecuteNextAsync<Entity>())  
    {  
        // Iterate through entities  
    }  
}
```

async
event-driven



Task

uniform



Task

IO-bound



Task

CPU-bound



Recap

best-practices

Use `async Task` instead of `async void`

Async all the way, don't mix blocking and asynchronous code

Async / await ●
is viral

but

It kicks your
servers

butt

Task.Run
Task.Factory.StartNew
Parallel.For
Parallel.ForEach

Worker
ThreadPool

IO
ThreadPool

await iobound
iobound.FireForget()



Task.Run
Task.Factory.StartNew
Parallel.For
Parallel.ForEach

Worker
ThreadPool

IO
ThreadPool

await iobound
iobound.FireForget()



NServiceBus

Azure Service Bus

26 times

Azure Storage Queues

6 times

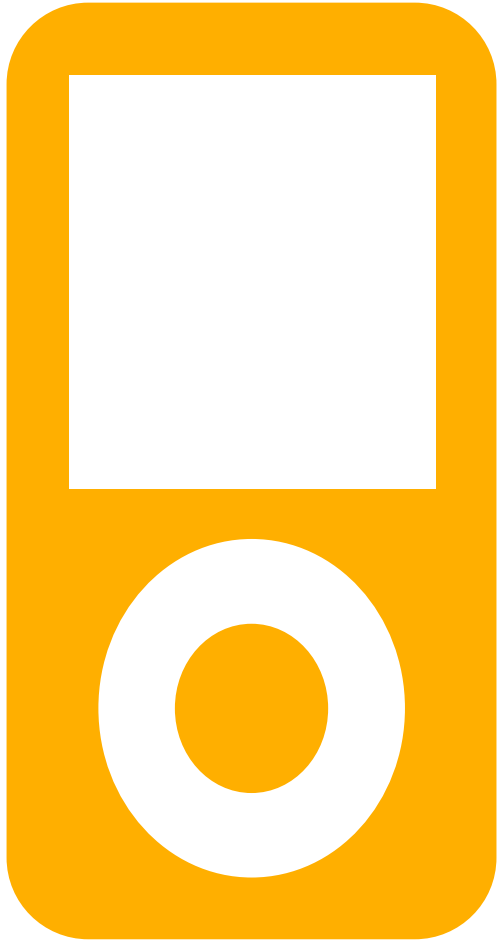
MSMQ

3 times

more message throughput

ASYNC





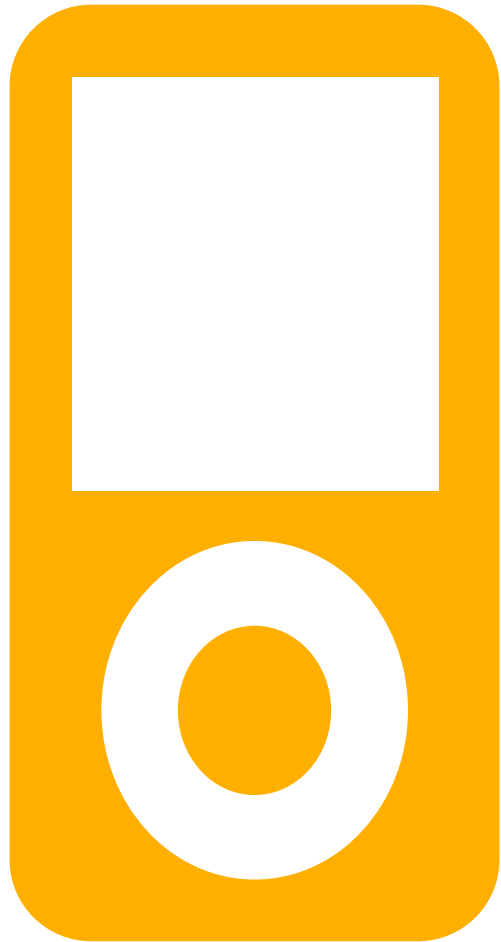
i dentify
ExP lore
O vercome
b ring together



Intro

Phases

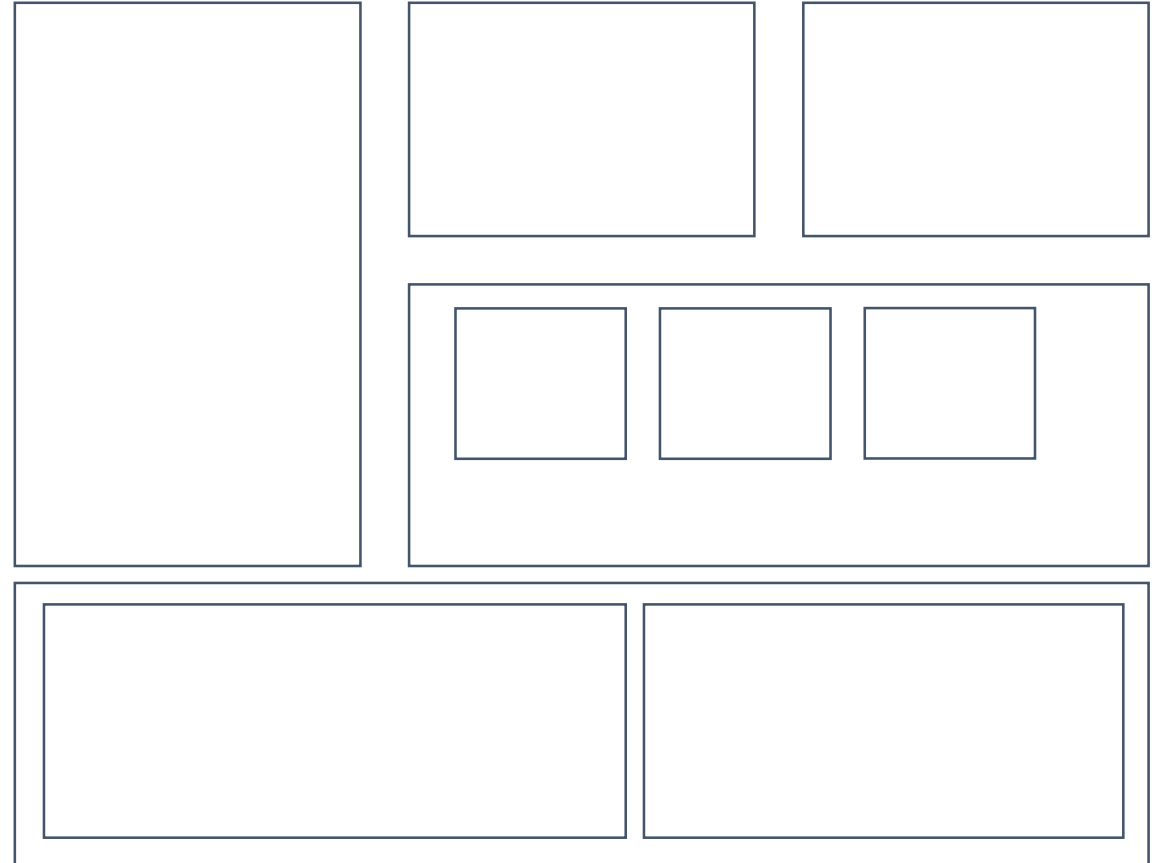
WrapUp



identify
Ex**P**lore
Overcome
bring together

Identify

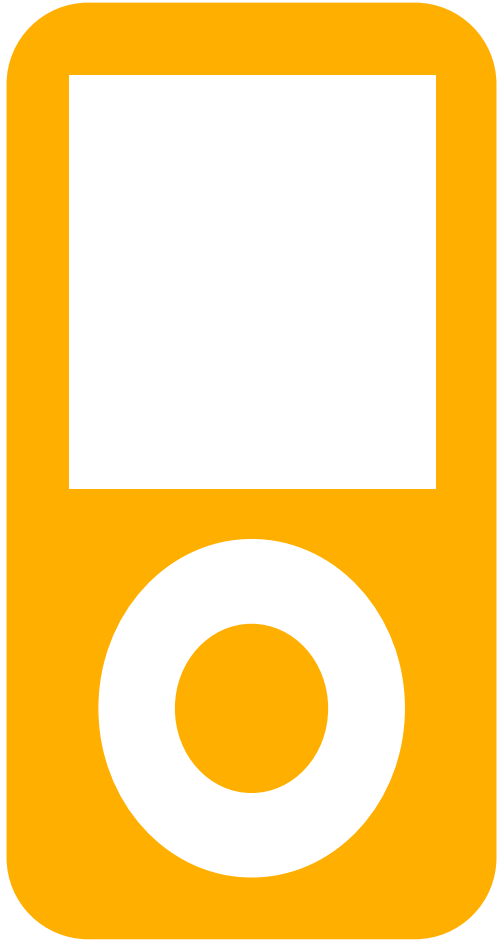
IO-bound



NServiceBus

IO-bound



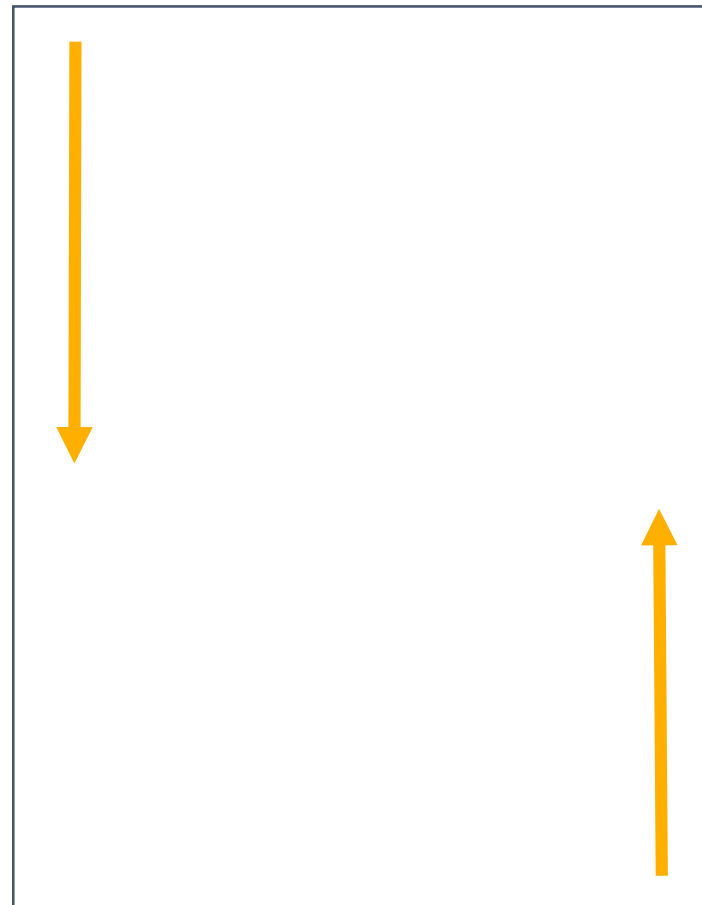


iIdentify
ExPlore
OOvercome
bbring together

Explore

IO-bound

High-level Spike

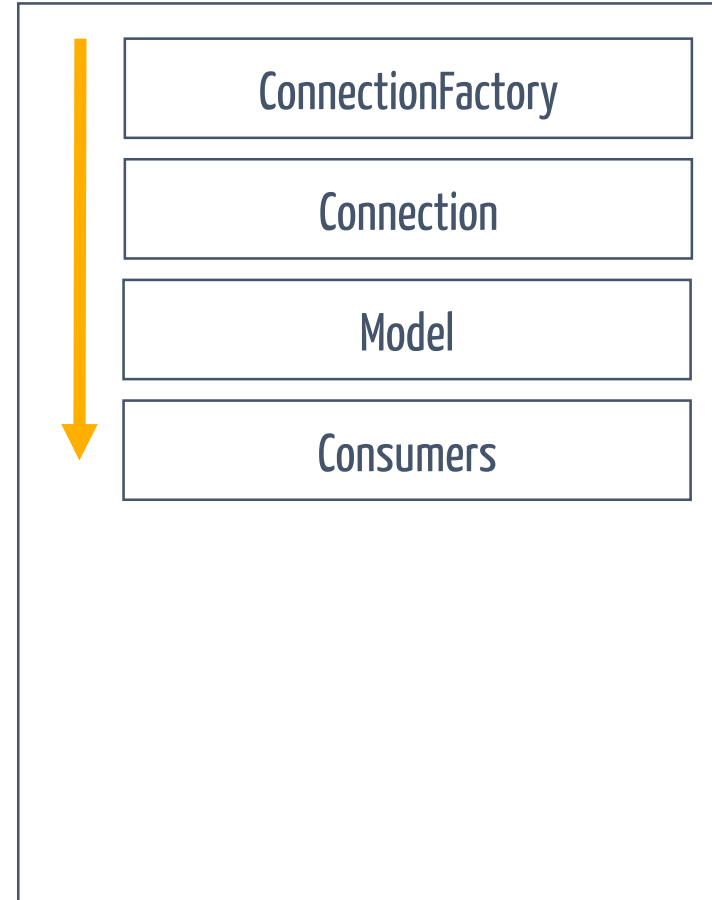


Low-level Spike

RabbitMQ Client

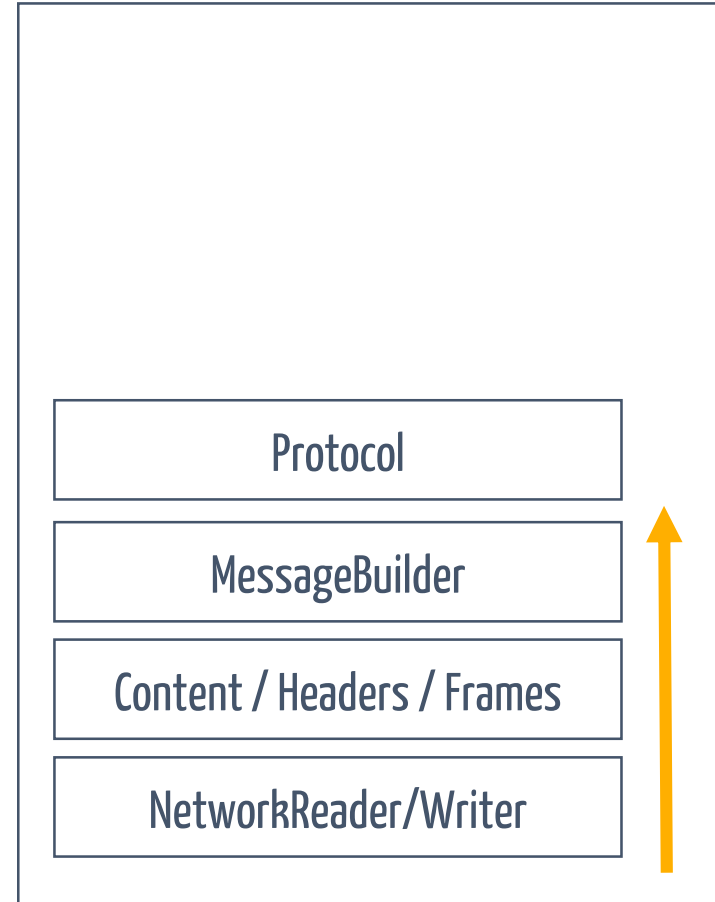
IO-bound

High-level Spike



RabbitMQ Client

IO-bound



Low-level Spike



Event handlers
Locks
Monitor
Semaphore / Mutex
Auto / ManualResetEvent
Ref/Out parameters
Thread
Ambient state
IO-bound calls in 3rd Party libs
Remote Procedure Calls

Event handler

Event handlers

```
public delegate void EventHandler(object sender, EventArgs e);
```

```
public delegate void EventHandler<TEventArgs>(object sender, TEventArgs e);
```



```
async void MyEventHandler(object sender, EventArgs e)
{
    await Task.Yield();
    throw new InvalidOperationException();
}
```

Event passed

Inside MyEventHandler

About to throw inside MyEventHandler

ManualResetEvent

ManualResetEvent



```
var syncEvent = new ManualResetEvent(false);
```

```
var t1 = Task.Run(() => {  
    syncEvent.WaitOne();  
});
```

```
var t2 = Task.Run(() => {  
    Thread.Sleep(2000);  
    syncEvent.Set();  
});
```

```
await Task.WhenAll(t1, t2);
```



void stinks wait smells

Remember

Async all the way means avoid blocking code

Locks

locks



```
var locker = new object();  
lock (locker)  
{  
    await Task.Yield();  
}
```

Error CS1996
Cannot await in the body of a lock statement

<http://stackoverflow.com/questions/7612602/why-cant-i-use-the-await-operator-within-the-body-of-a-lock-statement>

Ref / Out parameters

Ref/Out



```
static async Task Out(string content, out string parameter)
{
    var randomFileName = Path.GetTempFileName();
    using (var writer = new StreamWriter(randomFileName))
    {
        await writer.WriteLineAsync(content);
    }
    parameter = randomFileName;
}
```

Error CS1988

Async methods cannot have ref or out parameters

Ambient state

Ambient state



```
class ClassWithAmbientState
{
    static ThreadLocal<int> ambientState =
        new ThreadLocal<int>(() => 1);

    public void Do()
    {
        ambientState.Value++;
    }
}
```

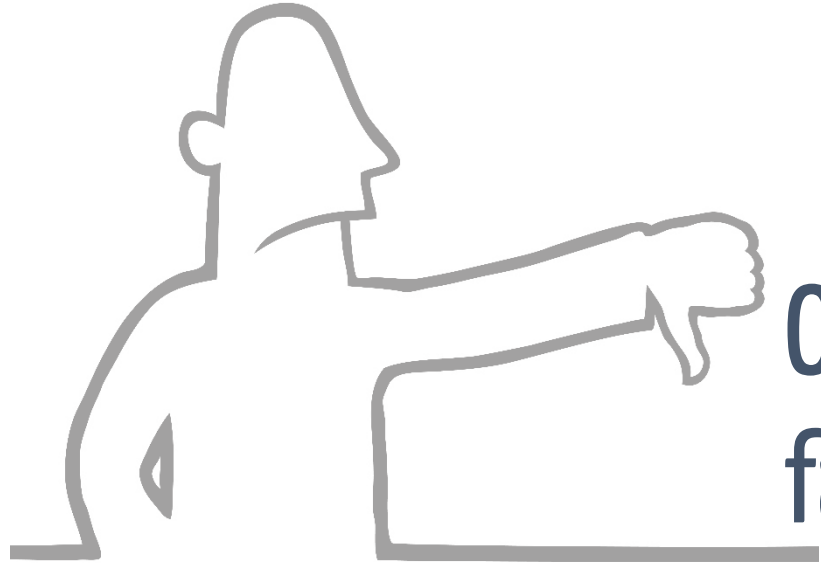
Ambient state



```
var instance = new ClassWithAmbientState();  
var tasks = new Task[3];  
for (int i = 0; i < 3; i++) {  
    tasks[i] = Task.Run(() => {  
        instance.Do();  
        Thread.Sleep(200);  
        instance.Do();  
    });  
}  
  
await Task.WhenAll(tasks);
```

AmbientState passed

```
05:50:09:187: Thread: 4, Value: 2  
05:50:09:187: Thread: 8, Value: 2  
05:50:09:187: Thread: 9, Value: 2  
05:50:09:390: Thread: 4, Value: 3  
05:50:09:391: Thread: 9, Value: 3  
05:50:09:391: Thread: 8, Value: 3
```

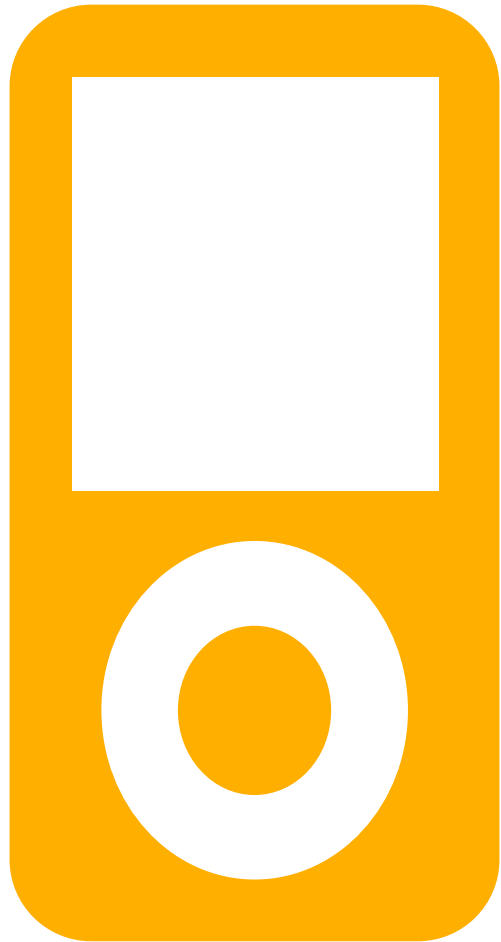


Older constructs **bound to threads**
fall apart in the async/await world

Remember

Forget thread!

think Task



iIdentify
ExPlore
OOvercome
bBring together

Event handler


```
public delegate void EventHandler(object sender, EventArgs e);
```

```
public delegate void EventHandler<TEventArgs>(object sender, TEventArgs e);
```

```
async void MyEventHandler(object sender, EventArgs e)
{
    await Task.Yield();
    throw new InvalidOperationException();
}
```

Event handlers

```
public delegate Task AsyncEventHandler(object sender, EventArgs e);
```

```
async Task MyAsyncEventHandler(object sender, EventArgs e) { }
```



```
async Task MyEventHandler(object sender, EventArgs e)
{
    await Task.Yield();
    throw new InvalidOperationException();
}
```

Event handlers



```
protected virtual Task OnMyAsyncEvent() {  
    var invocations = handler.GetInvocationList();  
    var handlerTasks = new Task[invocationList.Length];  
  
    for (int i = 0; i < invocations.Length; i++) {  
        handlerTasks[i] = ((AsyncEventHandler)invocations[i])(...);  
    }  
    return Task.WhenAll(handlerTasks);  
}
```

AsyncEvent passed

Inside MyAsyncEventHandler

About to throw inside MyAsyncEventHandler

Caught: Operation is not valid due to the current state of the object.

ManualResetEvent

```
var syncEvent = new ManualResetEvent(false);
```

```
var t1 = Task.Run(() => {  
    syncEvent.WaitOne();  
});
```

```
var t2 = Task.Run(() => {  
    Thread.Sleep(2000);  
    syncEvent.Set();  
});
```

```
await Task.WhenAll(t1, t2);
```

ManualResetEvent



```
var tcs = new TaskCompletionSource<object>();
```

```
var t1 = ((Func<Task>)(async () => {  
    await tcs.Task;  
}));
```

```
var t2 = ((Func<Task>)(async () => {  
    await Task.Delay(2000);  
    tcs.TrySetResult(null);  
}));
```

```
await Task.WhenAll(t1, t2);
```



TaskCompletionSource belongs
into your toolbox

Remember

ManualResetEvent



Works for **set once events** only.
For async reset events, an
approach is available on github

<https://github.com/danielmarbach/RearchitectTowardsAsyncAwait/blob/master/presentation/AsyncManualResetEvent.cs>

Locks

locks



Can we change the code so that
we don't have to await inside
the lock?

```
var locker = new object();  
lock (locker)  
{  
    await Task.Yield();  
}
```

Error CS1996

Cannot await in the body of a lock statement

locks



```
int sharedRessource = 0;  
var semaphore = new SemaphoreSlim(1);
```

```
var tasks = new Task[3];  
for (int i = 0; i < 3; i++) {  
    tasks[i] = ((Func<Task>) (async () => {  
        await semaphore.WaitAsync();  
        sharedRessource++;  
        semaphore.Release();  
    })))();  
}  
await Task.WhenAll(tasks);
```



SemaphoreSlim belongs
into your toolbelt

Remember

locks

```
using (await semaphore.LockAsync())  
{  
    sharedRessource++;  
}
```



<https://github.com/danielmarbach/RearchitectTowardsAsyncAwait/blob/master/presentation/AsyncLock.cs>

Ref / Out parameters

```
static async Task Out(string content, out string parameter)
{
    var randomFileName = Path.GetTempFileName();
    using (var writer = new StreamWriter(randomFileName))
    {
        await writer.WriteLineAsync(content);
    }
    parameter = randomFileName;
}
```

Error CS1988

Async methods cannot have ref or out parameters

Ref/Out



```
static async Task<string> Out(string content)
{
    var randomFileName = Path.GetTempFileName();
    using (var writer = new StreamWriter(randomFileName))
    {
        await writer.WriteLineAsync(content);
    }
    return randomFileName;
}
```

Ambient state

```
class ClassWithAmbientState
{
    static ThreadLocal<int> ambientState =
        new ThreadLocal<int>(() => 1);

    public void Do()
    {
        ambientState.Value++;
    }
}
```

Ambient state



```
class ClassWithAmbientState {  
    static AsyncLocal<int> ambientState =  
        new AsyncLocal<int>();
```

```
    static ClassWithAmbientState() {  
        ambientState.Value = 1;  
    }
```

```
    public void Do() {  
        ambientState.Value++;  
    }  
}
```

Ambient state

```
var instance = new ClassWithAmbientState();
var tasks = new Task[3];
for (int i = 0; i < 3; i++) {
    tasks[i] = ((Func<Task>)(async () => {
        instance.Do();
        await Task.Delay(200).ConfigureAwait(false);
        instance.Do();
    }));
}
```

```
await Task.WhenAll(tasks);
```



AmbientState passed

```
06:00:54:979: Thread: 5, Value: 2
06:00:54:985: Thread: 5, Value: 2
06:00:54:985: Thread: 5, Value: 2
06:00:55:185: Thread: 4, Value: 3
06:00:55:199: Thread: 4, Value: 3
06:00:55:199: Thread: 9, Value: 3
```

Ambient state



Even better:
Can we change the code so that
we float state into methods
that need it?

Ambient state

```
class ClassWithAmbientState {
```

```
    public int Do(int current) {
```

```
        current++;
```

```
        return current;
```

```
    }
```

```
}
```



Ambient state

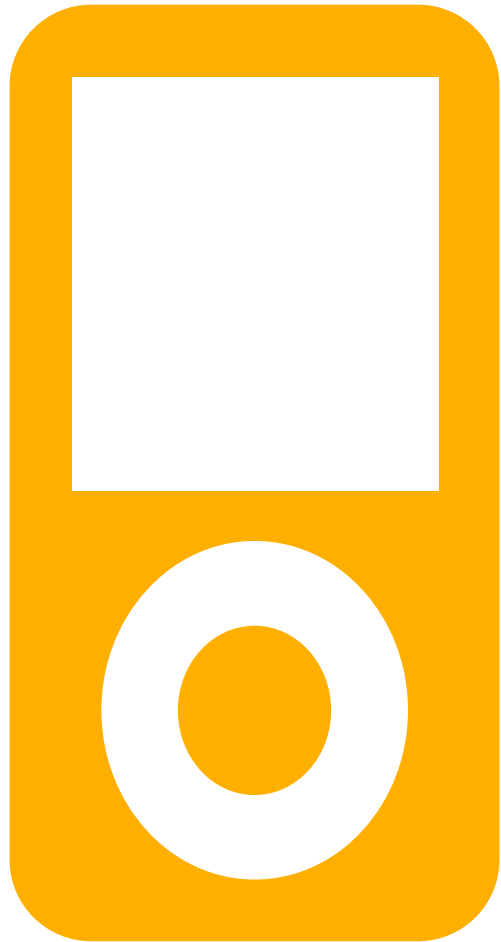
```
var instance = new ClassWithFloatingState();
```

```
var tasks = new Task[3];  
for (int i = 0; i < 3; i++) {  
    tasks[i] = ((Func<Task>)(async () => {  
        int current = 1;  
        current = instance.Do(current);  
        await Task.Delay(200).ConfigureAwait(false);  
        instance.Do(current);  
    })))();  
}  
await Task.WhenAll(tasks);
```



AmbientFloatingStateReturned passed

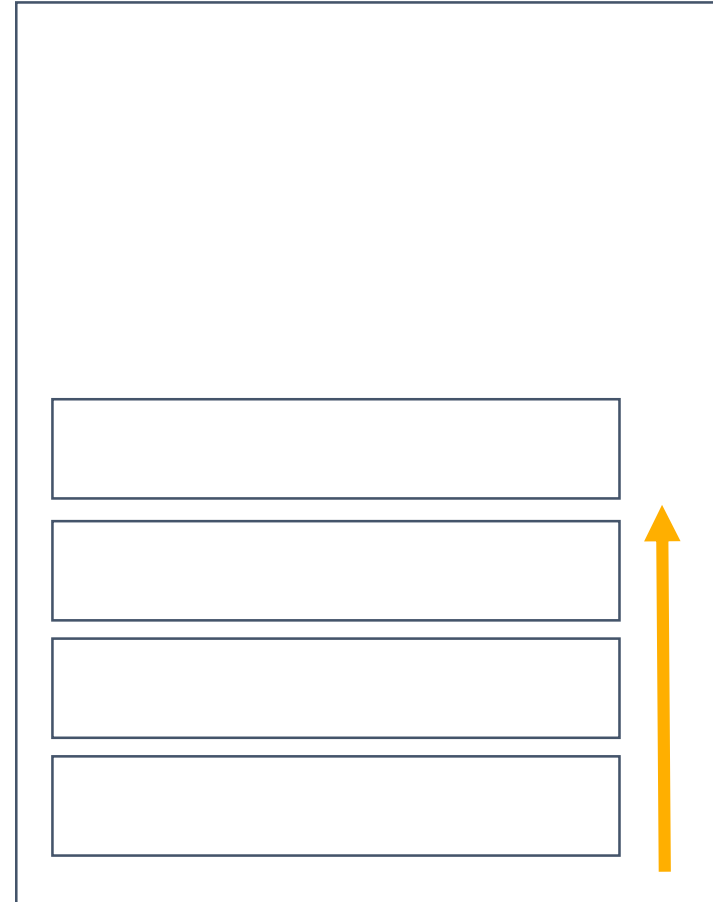
```
06:03:28:728: Thread: 5, Value: 2  
06:03:28:734: Thread: 5, Value: 2  
06:03:28:735: Thread: 5, Value: 2  
06:03:28:933: Thread: 4, Value: 3  
06:03:28:950: Thread: 4, Value: 3  
06:03:28:950: Thread: 9, Value: 3
```

Identify
ExPlore
Overcome
bring together

Bring it
together

High-level



Low-level

Bring it
together

```
void HighLevel() {  
    try {  
        MidLevel();  
    } catch(InvalidOperationException) { }  
}
```

```
void MidLevel() {  
    ...  
    LowLevel();  
    ...  
}
```

```
void LowLevel() {  
}
```

Bring it together

```
void HighLevel() {  
    try {  
        MidLevel();  
    } catch(InvalidOperationException) { }  
}
```

```
void MidLevel() {  
    ...  
    LowLevel().GetAwaiter().GetResult();  
    ...  
}
```

```
async Task LowLevel() {  
}
```

Commit. Push.

Bring it together

```
void HighLevel() {  
    try {  
        MidLevel().GetAwaiter().GetResult();  
    } catch(InvalidOperationException) { }  
}
```

```
async Task MidLevel() {  
    ...  
    await LowLevel().ConfigureAwait(false);  
    ...  
}
```

```
async Task LowLevel() {  
}
```

Commit. Push.

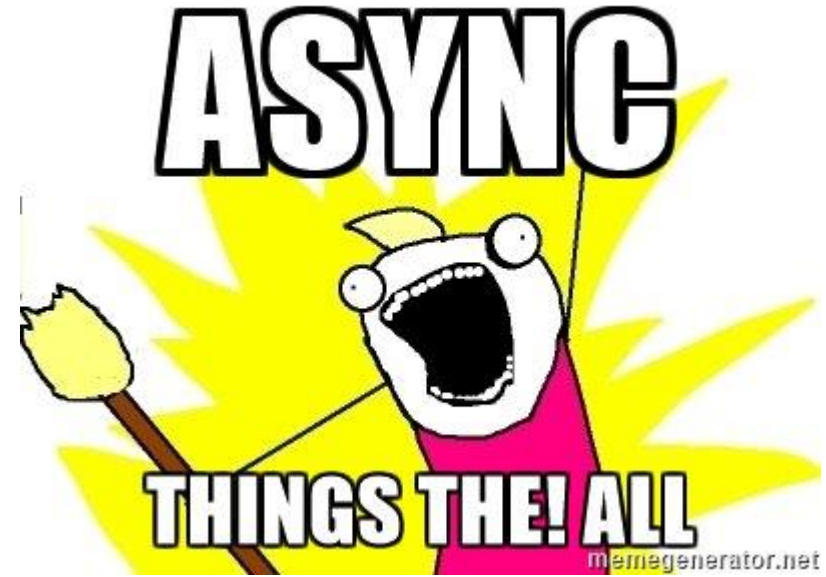
Bring it together

```
async Task HighLevel() {  
    try {  
        await MidLevel ().ConfigureAwait(false);  
    } catch(InvalidOperationException) { }  
}
```

```
async Task MidLevel() {  
    ...  
    await LowLevel().ConfigureAwait(false);  
    ...  
}
```

```
async Task LowLevel() {  
}
```


Yehaa!



Async all the way



Intro

Phases

WrapUp

Recap

reminder

Use **iPob** to move your code
step by step towards async / await

IO-bound paths benefit from async

Uniform API of Task allows to await
CPU-bound as well as IO-bound tasks

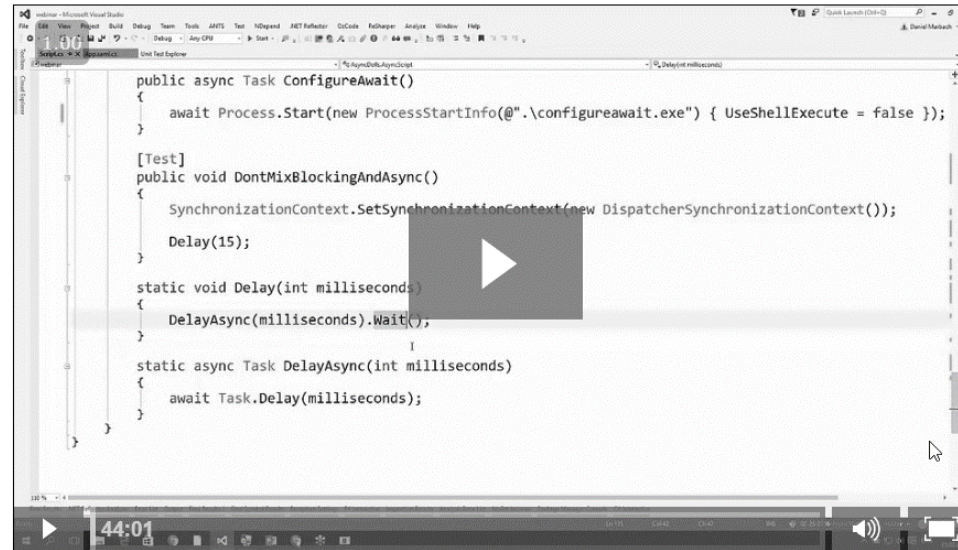
Slides, Links...

github.com/danielmarbach/RearchitectTowardsAsyncAwait

Async/Await Webinar Series: Best Practices

See how to avoid common pitfalls in asynchronous code bases

go.particular.net/ndc16.async



[f](#) [G+](#) [Twitter](#) [in](#) [Share](#) [Samples](#) [Slides](#) [Comments \(0\) →](#)

Summary

Daniel Marbach shows how to avoid common pitfalls in asynchronous code bases.

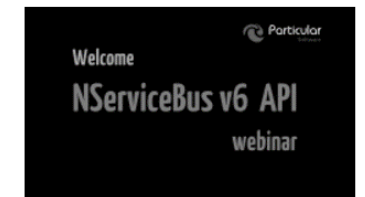
Learn how to:

- Differentiate between IO-bound vs CPU-bound work and how this relates to Threads and Tasks
- Avoid serious production bugs as a result of asynchronous methods returning void
- Opt-out from context capturing when necessary
- Deal with synchronous code in the context of asynchronous code

OTHER VIDEOS IN THE SERIES



► TPL & Message Pumps



► NServiceBus v6 API Update

await Q & A

Thanks