Particular
Software

Welcome

Async / Await

The die is cast

Solution Architect
Enthusiastic Software Engineer
Microsoft Azure MVP

@danielmarbach
particular.net/blog
planetgeek.ch

Goals
target

CPU-bound vs IO-bound

Threads and Tasks

Async best-practices

Why async is the future

Terminology Why WrapUp

Terminology   Why   WrapUp

async

event-driven

Task

uniform

Task

IO-bound

Task

CPU-bound

concurrent
concurrent
concurrent concurrent
concurrent
interleaved

parallel

parallel

simultaneous

Continuation

function

Terminology | Why | WrapUp

async/await

simplicity

```
function1(function(err, res) {
  function2(function(err, res) {
    function3(function(err, res) {
      function4(function(err, res) {
        function5(function(err, res) {
          // do something useful
        })
      })
    })
  })
})
```

```
cleanLaundry.ContinueWith(t => {
  dryLaundry;
})
```

```
await cleanLaundry;
dryLaundry;
```

await Demo

javascript
ES2015

```javascript
function chainAnimationsPromise(elem, animations) {
    let ret = null;
    let p = currentPromise;
    for(const anim of animations) {
        p = p.then(function(val) {
            ret = val;
            return anim(elem);
        })
    }
    return p.catch(function(e) {
        /* ignore and keep going */
    }).then(function() {
        return ret;
    });
}
```

aluate   Presets: ☑ **es2015**   ☐ **es2015-loose**   ☑ **react**   ☐ **stage-0**   ☐ **stage-1**   ☐ **stage-2**   ☑ **stage-3**

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

```
 1  "use strict";
 2
 3  var chainAnimationsAsync = function () {
 4      var ref = _asyncToGenerator(regeneratorRuntime.mark(function _call
 5          var ret, _iteratorNormalCompletion, _didIteratorError, _iterat
 6
 7          return regeneratorRuntime.wrap(function _callee$(_context) {
 8              while (1) {
 9                  switch (_context.prev = _context.next) {
10                      case 0:
11                          ret = null;
12                          _context.prev = 1;
13                          _iteratorNormalCompletion = true;
14                          _didIteratorError = false;
15                          _iteratorError = undefined;
16                          _context.prev = 5;
17                          _iterator = animations[Symbol.iterator]();
18
19                      case 7:
20                          if (_iteratorNormalCompletion = (_step = _iter
21                              _context.next = 15;
22                              break;
23                          }
24
25                          anim = _step.value;
26                          _context.next = 11;
27                          return anim(elem);
28
29                      case 11:
30                          ret = _context.sent;
31
```

Try it out
babeljs.io

$ npm install babel-plugin-syntax-async-functions
$ npm install babel-plugin-transform-async-to-generator

dart

release 1.9

```
runUsingFuture() {
  //...
  findEntrypoint().then((entrypoint) {
    return runExecutable(entrypoint, args);
  }).then(flushThenExit);
}
```

dart

release 1.9

```
runUsingAsyncAwait() async {
  //...
  var entrypoint = await findEntrypoint();
  var exitCode = await
      runExecutable(entrypoint, args);
  await flushThenExit(exitCode);
}
```

python

release 3.5

```python
import asyncio

async def http_get(domain):
    reader, writer =
        await asyncio.open_connection(domain, 80)

    async for line in reader:
        print('>>>', line)
```

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

f... off

I don't care about your stupid async stuff!

cumbersome

```csharp
class VanillaHandler : IHandleMessages<AcquireVanilla>
{

    void Handle(AcquireVanilla message)
    {

        AcquireVanillaFromGovernmentAsync().Result;
        DownloadRecipeFromBlobStorageAsync().Wait();
        InsertVanillaUsageInDocumentDBAsync().Result;
        StoreTelemetryDataInEventHubAsync().Wait();

    }

}
```

wasteful

```csharp
class VanillaHandler : IHandleMessages<AcquireVanilla>
{

    void Handle(AcquireVanilla message)
    {

        AcquireVanillaFromGovernmentAsync().Result;
        DownloadRecipeFromBlobStorageAsync().Wait();
        InsertVanillaUsageInDocumentDBAsync().Result;
        StoreTelemetryDataInEventHubAsync().Wait();
    }


}
```

await **Demo**

dangerous

```csharp
class VanillaHandler : IHandleMessages<AcquireVanilla>
{

    void Handle(AcquireVanilla message)
    {
        await AcquireVanillaFromGovernmentAsync();
        await DownloadRecipeFromBlobStorageAsync();
        await InsertVanillaUsageInDocumentDBAsync();
        await StoreTelemetryDataInEventHubAsync() ;
    }

}
```

dangerous

Error   CS4033   The 'await' operator can only be used within an async method. Consider marking this method with the 'async' modifier and changing its return type to 'Task'.

dangerous

```csharp
class VanillaHandler : IHandleMessages<AcquireVanilla>
{

    async void Handle(AcquireVanilla message)
    {

        await AcquireVanillaFromGovernmentAsync();
        await DownloadRecipeFromBlobStorageAsync();
        await InsertVanillaUsageInDocumentDBAsync();
        await StoreTelemetryDataInEventHubAsync() ;
    }

}
```

await Demo

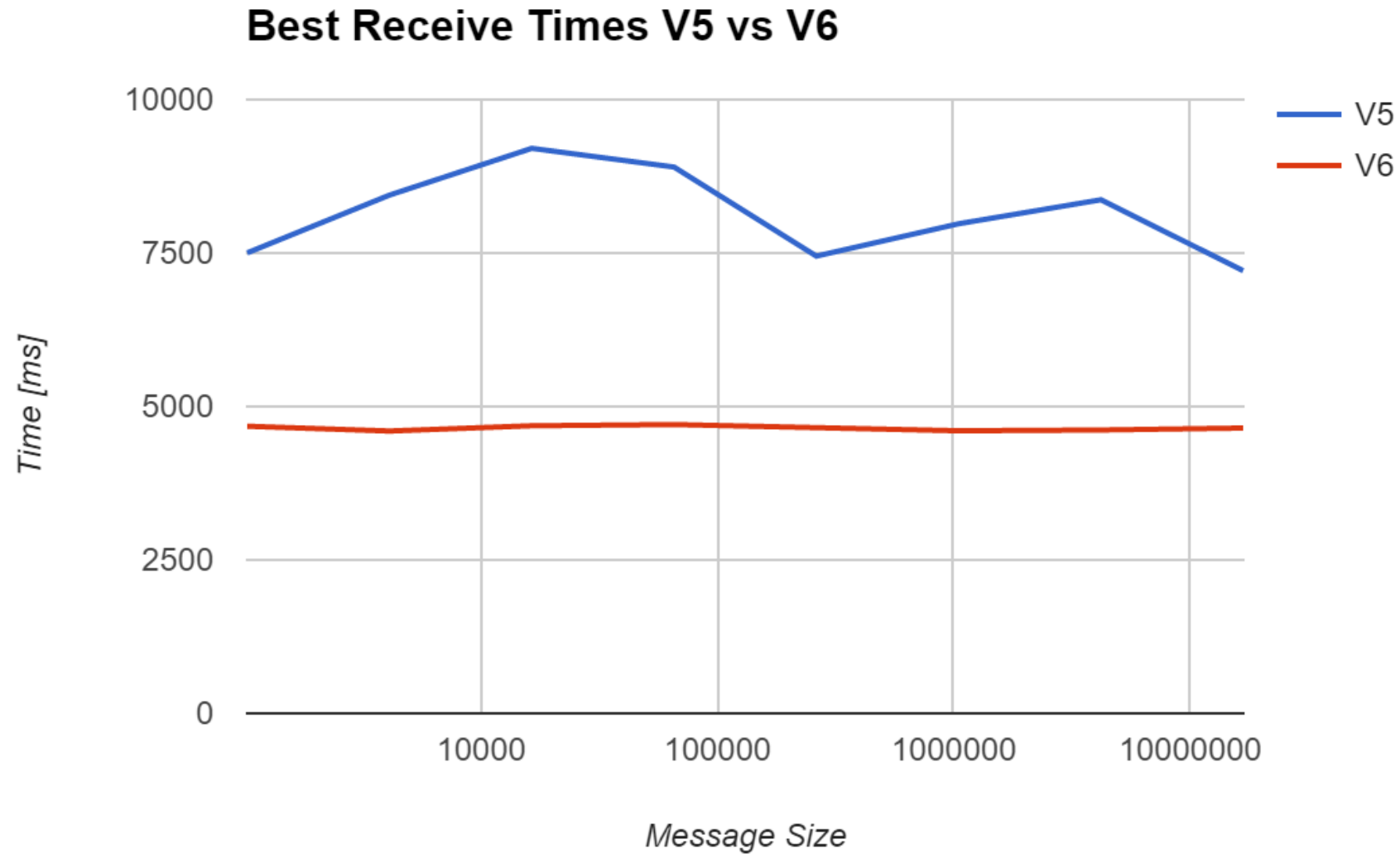# NServiceBus

Azure Service Bus   26 times

Azure Storage Queues   6 times

MSMQ   3 times

more message throughput

NServiceBus.SqlServer

Best Receive Times V5 vs V6

Terminology Why **WrapUp**

Forget thread!

think **Task**

The die **is** cast

NSB v6

Will be Async all the way

Applies the shown best-practies like ConfigureAwait(false) consequently, checked with Roslyn analyzer

particular.net/blog/async-await-its-time

Recap
reminder

Use **Task.Run**, **Factory.StartNew** for CPU-bound work

Use **Task** directly for IO-bound work

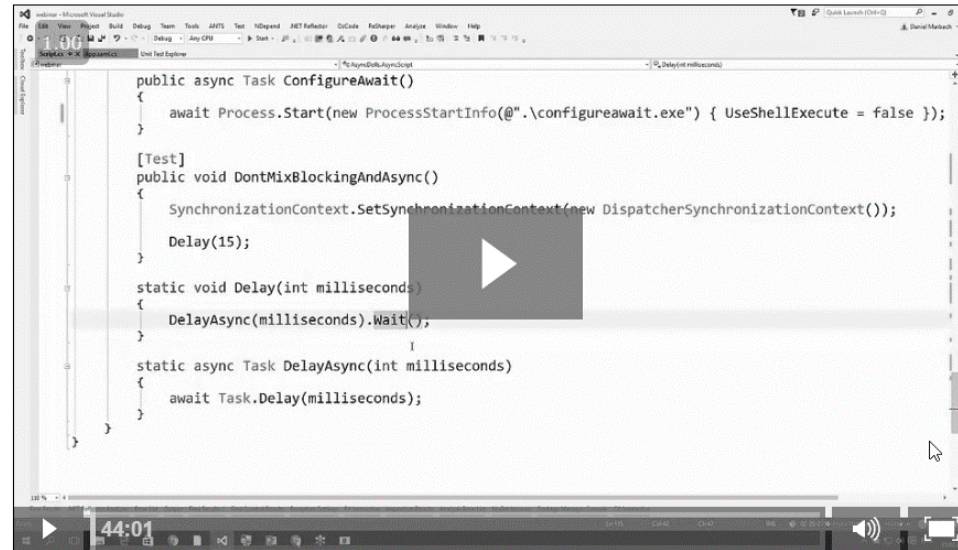Use **async Task** instead of **async void**

Recap
reminder

Libraries and frameworks should use ConfigureAwait(false)

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

Async/Await Webinar Series: Best Practices

See how to avoid common pitfalls in asynchronous code bases

go.particular.net/DNC016

▸ TPL & Message Pumps

▸ NServiceBus v6 API Update

f G+ in Share   Samples   Slides   Comments (0)→

Summary

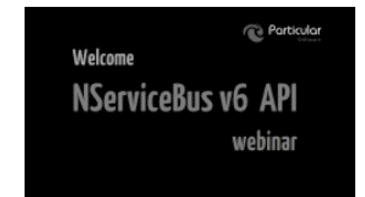Daniel Marbach shows how to avoid common pitfalls in asynchronous code bases.

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

Slides, Links...

github.com/danielmarbach/Async.DielsCast

await **Q & A**

Quiz

```
Task.Run(async () =>
{
    while(!cancelled) {
        Func<Task> receive = () => {

            var connection = new SqlConnection();
            await connection.OpenAsync()
                .ConfigureAwait(false);
        }

        receive().Ignore();
        await Task.Delay(100).ConfigureAwait(false);
    }
})
```

```
Task.Run(async () =>
{
    while(!cancelled) {
        Func<Task> receive = () => {
            await Task.Yield();
            var connection = new SqlConnection();
            await connection.OpenAsync()
                .ConfigureAwait(false);
        }

        receive().Ignore();
        await Task.Delay(100).ConfigureAwait(false);
    }
})
```