BEYOND SIMPLE BENCHMARKS

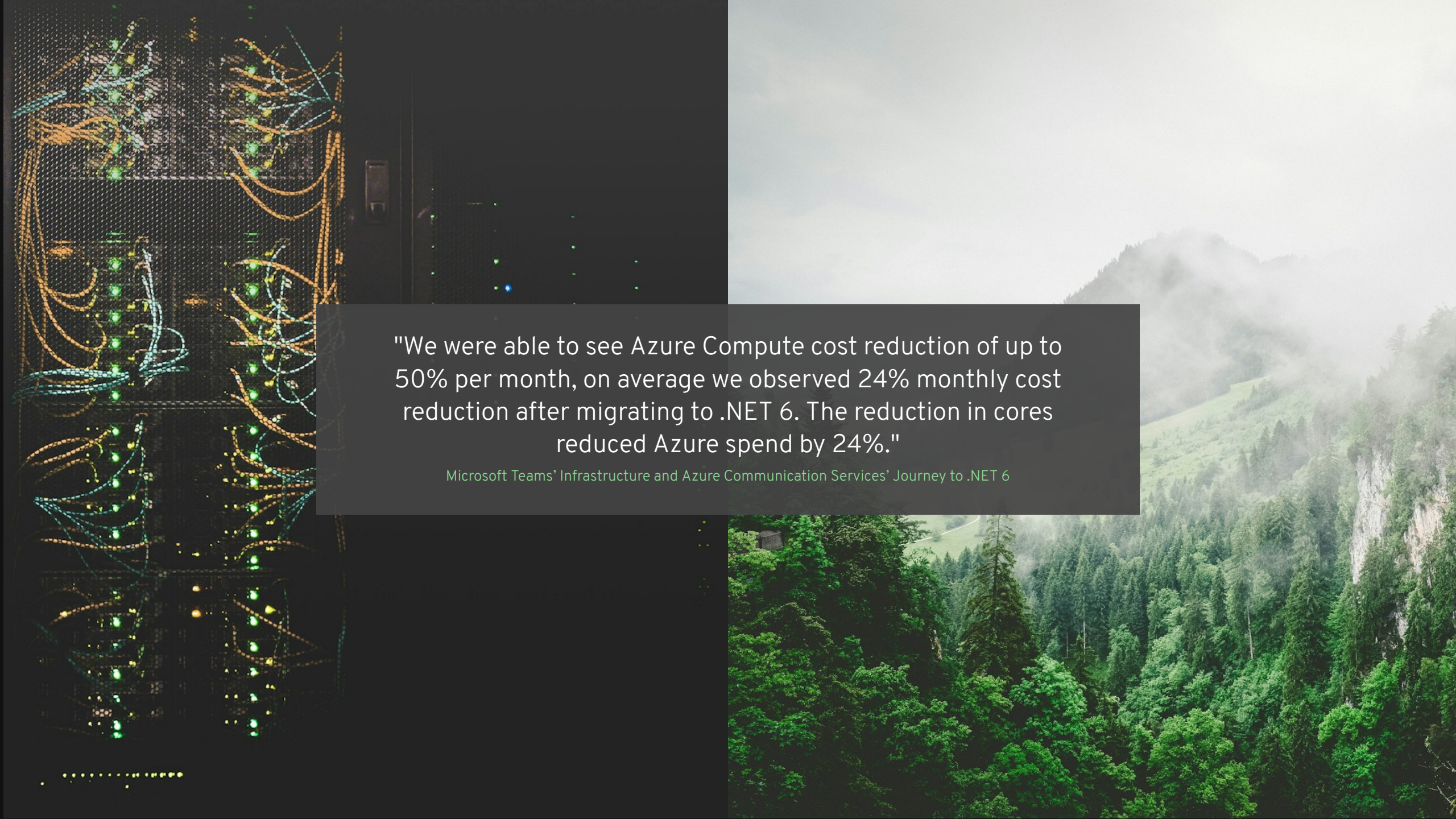
A PRACTICAL GUIDE TO OPTIMIZING CODE

danielmarbach | daniel.marbach@particular.net | Daniel Marbach

```
[SimpleJob]
[MemoryDiagnoser]
public class StringJoinBenchmarks {

  [Benchmark]
  public string StringJoin() {
    return string.Join(", ", Enumerable.Range(0, 10).Select(i => i.ToString()));
  }

  [Benchmark]
  public string StringBuilder() {
    var sb = new StringBuilder();
    for (int i = 0; i < 10; i++)
    {
        sb.Append(i);
        sb.Append(", ");
    }

    return sb.ToString(0, sb.Length - 2);
  }

  [Benchmark]
  public string ValueStringBuilder() {
    var seperator = new ReadOnlySpan<char>(new char[] { ',', ' '});
    using var sb = new ValueStringBuilder(stackalloc char[30]);
    for (int i = 0; i < 10; i++)
    {
        sb.Append(i);
        sb.Append(seperator);
    }

    return sb.AsSpan(0, sb.Length - 2).ToString();
  }
}
```

"SIMPLE"


Wrong!
2 + 2 = 5



"We were able to see Azure Compute cost reduction of up to 50% per month, on average we observed 24% monthly cost reduction after migrating to .NET 6. The reduction in cores reduced Azure spend by 24%."

Microsoft Teams' Infrastructure and Azure Communication Services' Journey to .NET 6

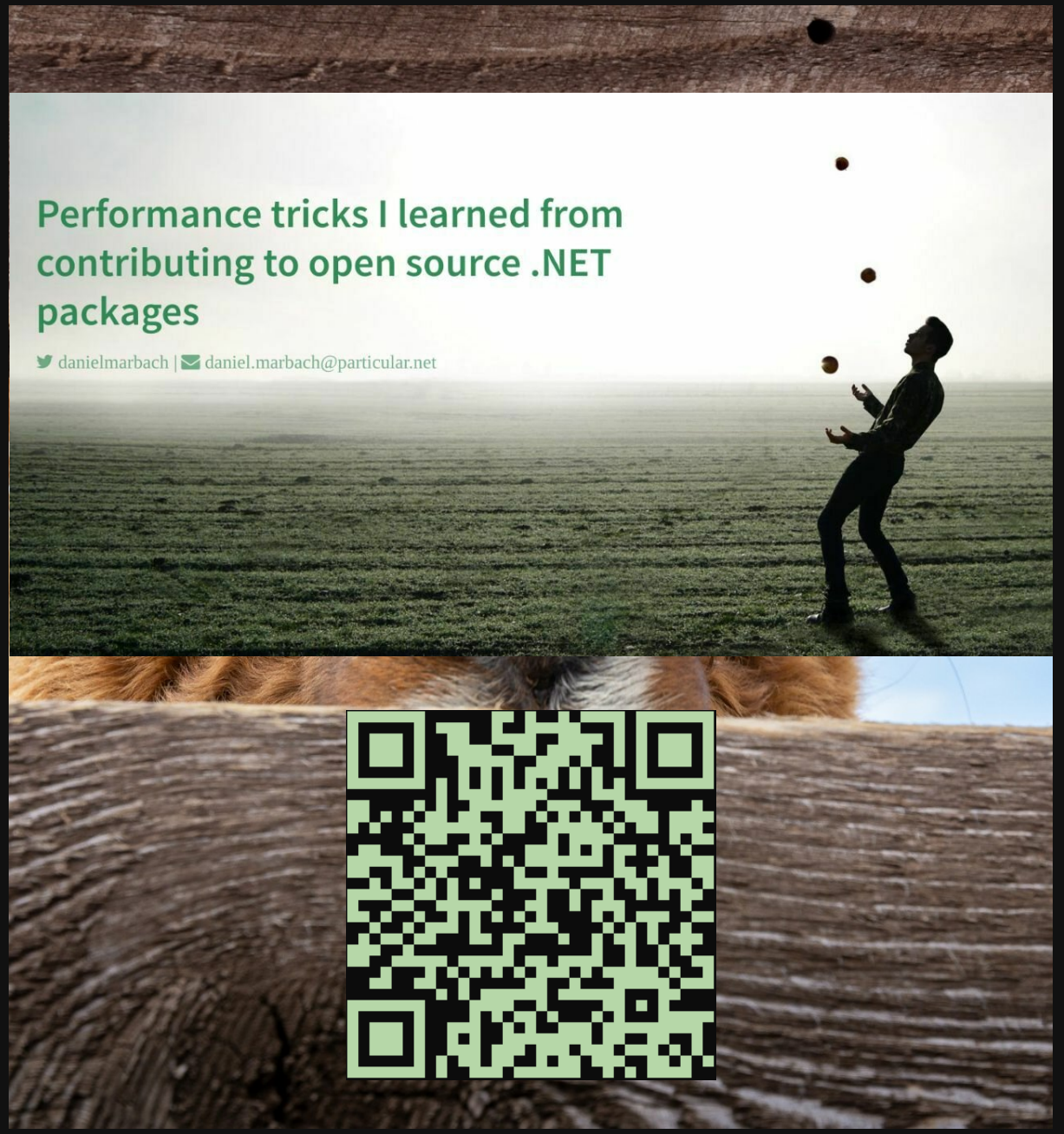
PERFORMANCE AWARE

BEAR

AWARE

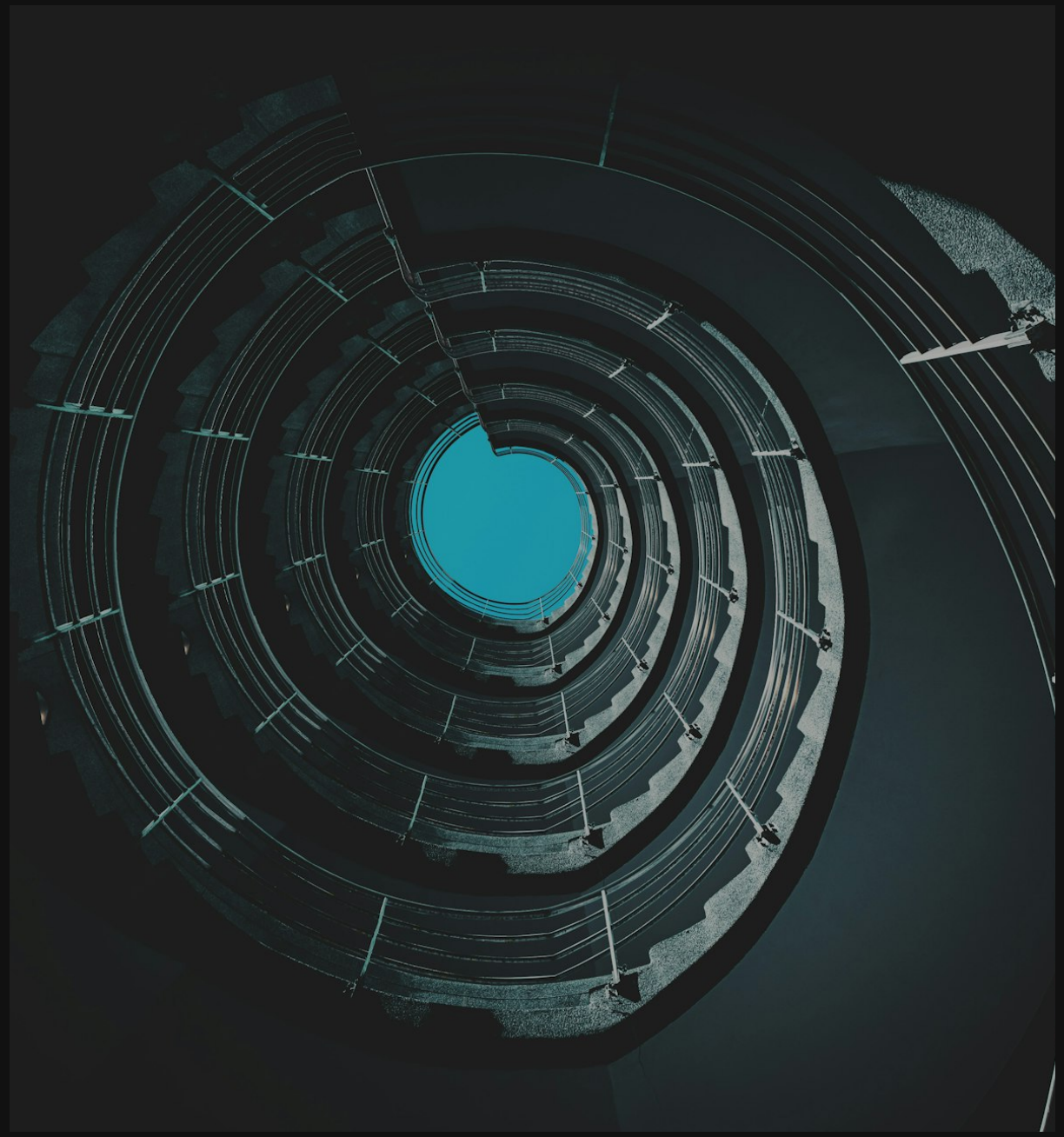
BE CURIOUS....
UNDERSTAND THE CONTEXT

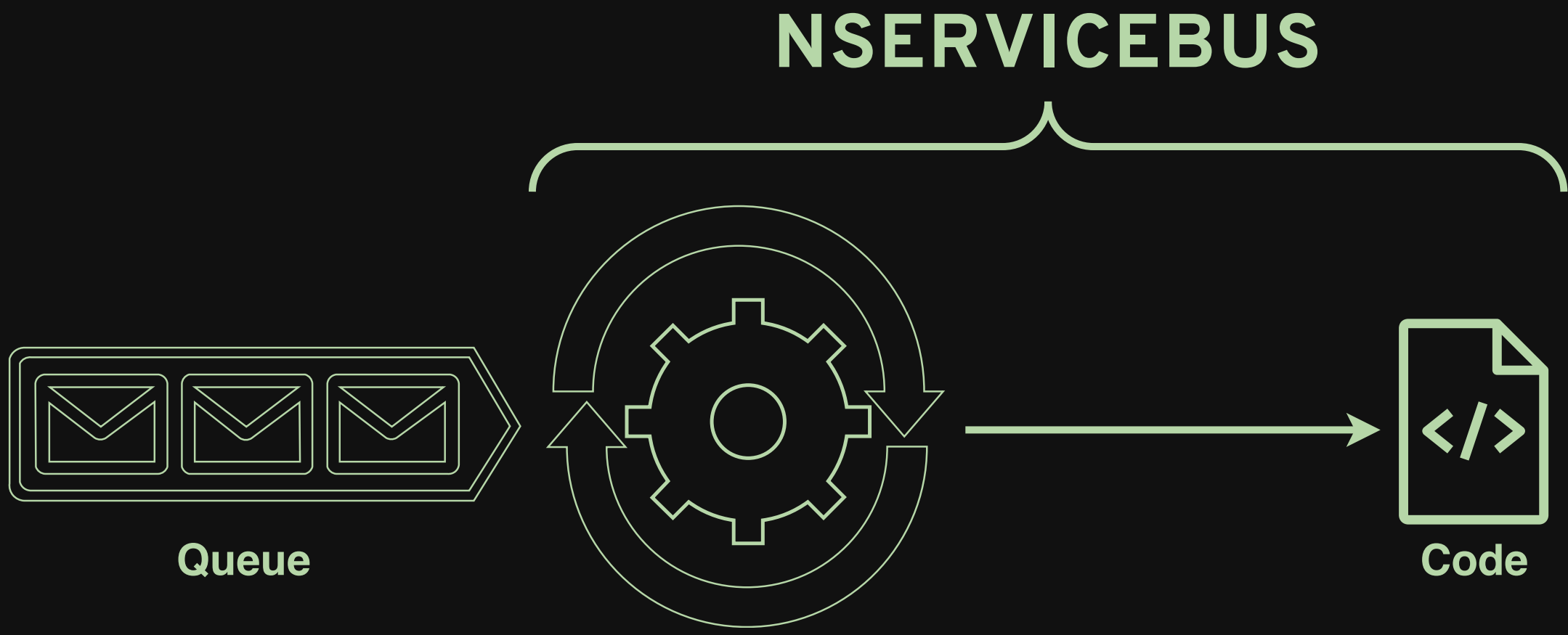
- How is this code going to be executed at scale, and what would the memory characteristics be (gut feeling)
- Are there simple low-hanging fruits I can apply to accelerate this code?
- Are there things I can move away from the hot path by simply restructuring a bit my code?
- What part is under my control and what isn't really?
- What optimizations can I apply, and when should I stop?



THE PERFORMANCE LOOP

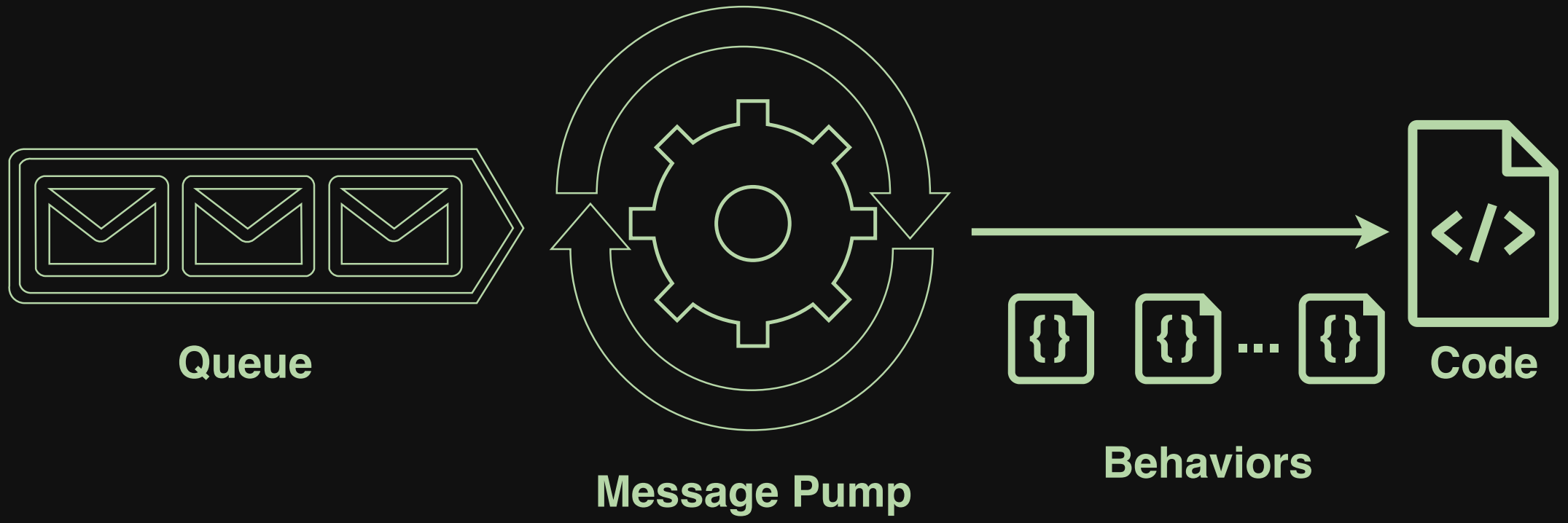
- Profile at least CPU and memory using a profiling harness
- Improve parts of the hot path
- Benchmark and compare
- Profile improvements again with the harness and make adjustments where necessary
- Ship and focus your attention to other parts





go.particular.net/webinar-2023-quickstart

NSERVICEBUS PIPELINE



ASP.NET CORE MIDDLEWARE

```
1  public class RequestCultureMiddleware {
2      private readonly RequestDelegate _next;
3
4      public RequestCultureMiddleware(RequestDelegate next) {
5          _next = next;
6      }
7
8      public async Task InvokeAsync(HttpContext context) {
9          // Do work that does something before
10         await _next(context);
11         // Do work that does something after
12     }
13 }
```

BEHAVIORS

```
1 public class Behavior : Behavior<IIncomingLogicalMessageContext> {
2     public override Task
3         Invoke(IIncomingLogicalMessageContext context, Func<Task> next) {
4         // Do work that does something before
5         await next();
6         // Do work that does something after
7     }
8 }
```

PROFILING THE PIPELINE

THE HARNESS

```
1  var endpointConfiguration = new EndpointConfiguration("PublishSample");
2  endpointConfiguration.UseSerialization<JsonSerializer>();
3  var transport = endpointConfiguration.UseTransport<MsmqTransport>();
4  transport.Routing().RegisterPublisher(typeof(MyEvent), "PublishSample");
5  endpointConfiguration.UsePersistence<InMemoryPersistence>();
6  endpointConfiguration.EnableInstallers();
7  endpointConfiguration.SendFailedMessagesTo("error");
8
9  var endpointInstance = await Endpoint.Start(endpointConfiguration);
10
11 Console.WriteLine("Attach the profiler and hit <enter>.");
12 Console.ReadLine();
13
14 var tasks = new List<Task>(1000);
15 for (int i = 0; i < 1000; i++)
16 {
17     tasks.Add(endpointInstance.Publish(new MyEvent()));
18 }
19 await Task.WhenAll(tasks);
20
21 Console.WriteLine("Publish 1000 done. Get a snapshot");
22 Console.ReadLine();
```

THE HARNESS

```
public class MyEventHandler : IHandleMessages<MyEvent> {
    public Task Handle(MyEvent message, IMessageHandlerContext context)
    {
        Console.WriteLine("Event received");
        return Task.CompletedTask;
    }
}
```

THE HARNESS

- Compiled and executed in Release mode
- Runs a few seconds and keeps overhead minimal
- Disabled Tiered JIT

  `<TieredCompilation>false</TieredCompilation>`
- Emits full symbols

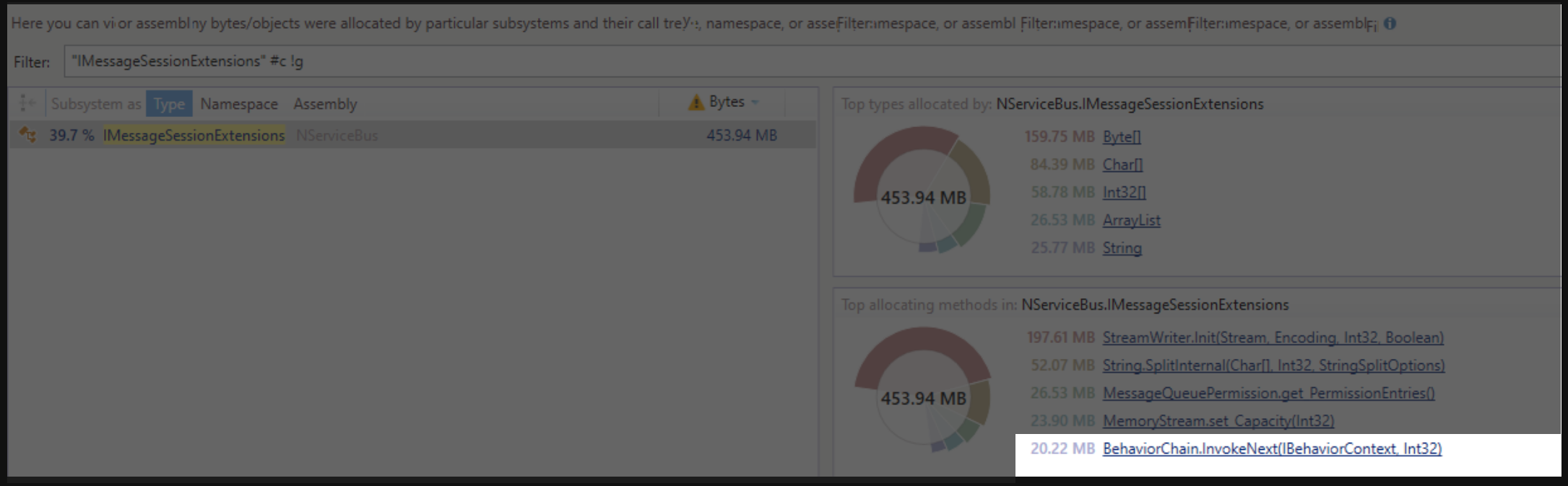
  `<DebugType>pdbonly</DebugType>`

  `<DebugSymbols>true</DebugSymbols>`

```
var endpointConfiguration = new EndpointConfiguration("PublishSample");
endpointConfiguration.UseSerialization<JsonSerializer>();
var transport = endpointConfiguration.UseTransport<MsmqTransport>();
transport.Routing().RegisterPublisher(typeof(MyEvent), "PublishSample");
endpointConfiguration.UsePersistence<InMemoryPersistence>();
endpointConfiguration.EnableInstallers();
endpointConfiguration.SendFailedMessagesTo("error");

var endpointInstance = await Endpoint.Start(endpointConfiguration);

Console.WriteLine("Attach the profiler and hit <enter>.");
Console.ReadLine();

var tasks = new List<Task>(1000);
for (int i = 0; i < 1000; i++)
{
    tasks.Add(endpointInstance.Publish(new MyEvent()));
}
await Task.WhenAll(tasks);

Console.WriteLine("Publish 1000 done. Get a snapshot");
Console.ReadLine();
```

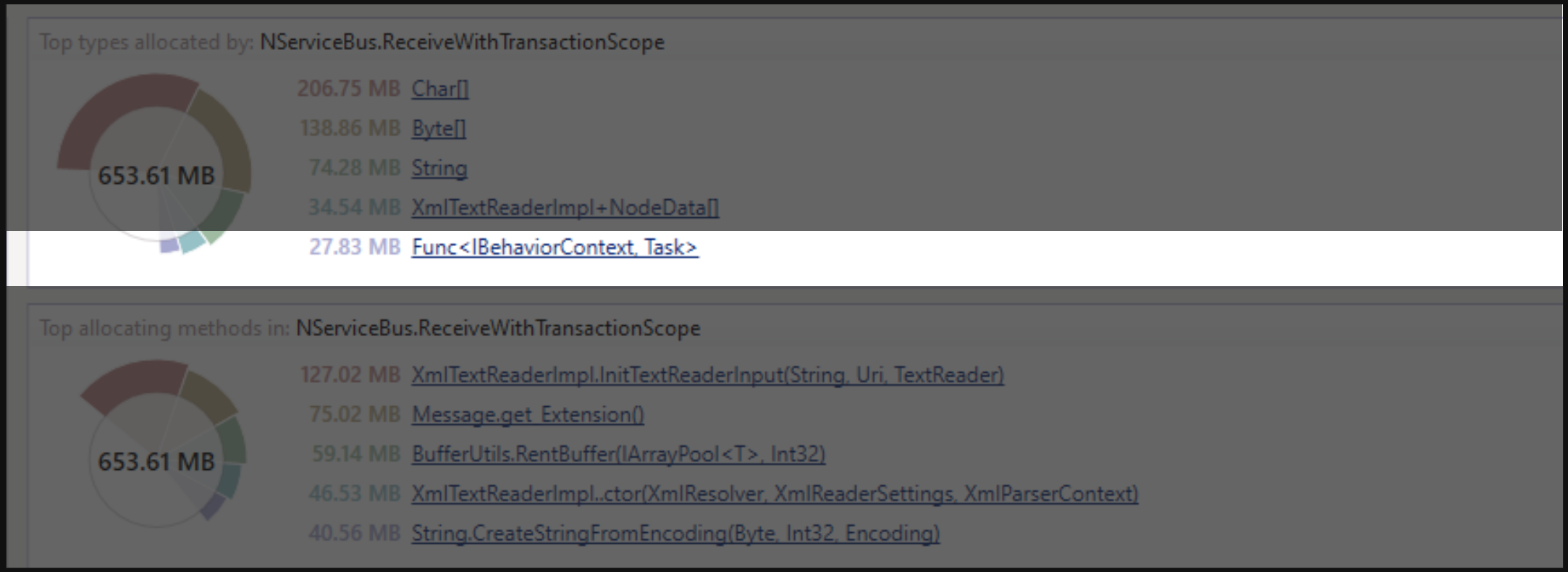
```
public class MyEventHandler : IHandleMessages<MyEvent> {
    public Task Handle(MyEvent message, IMessageHandlerContext context)
    {
        Console.WriteLine("Event received");
        return Task.CompletedTask;
    }
}
```

MEMORY CHARACTERISTICS



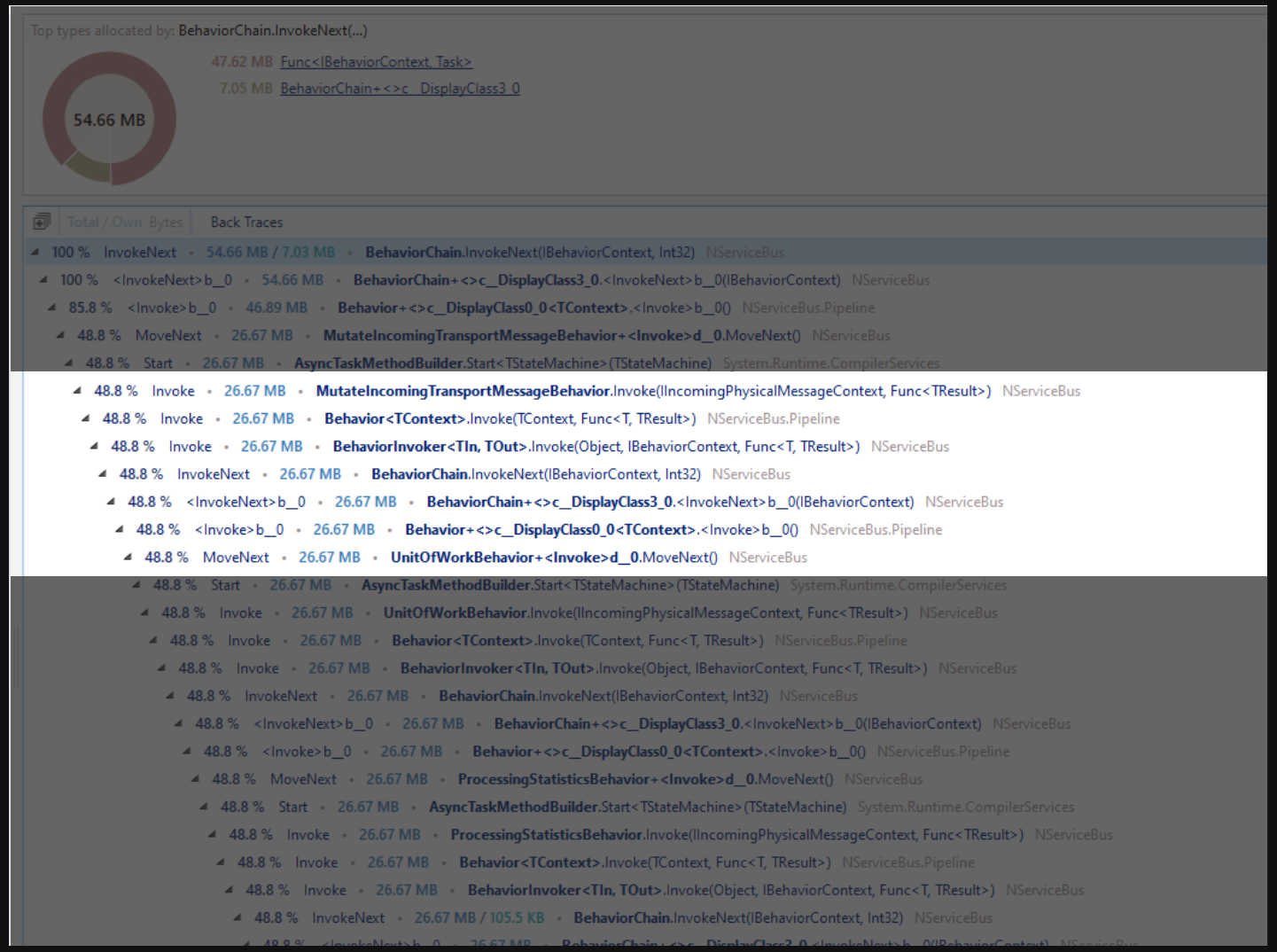
PUBLISH

MEMORY CHARACTERISTICS



RECEIVE

MEMORY CHARACTERISTICS



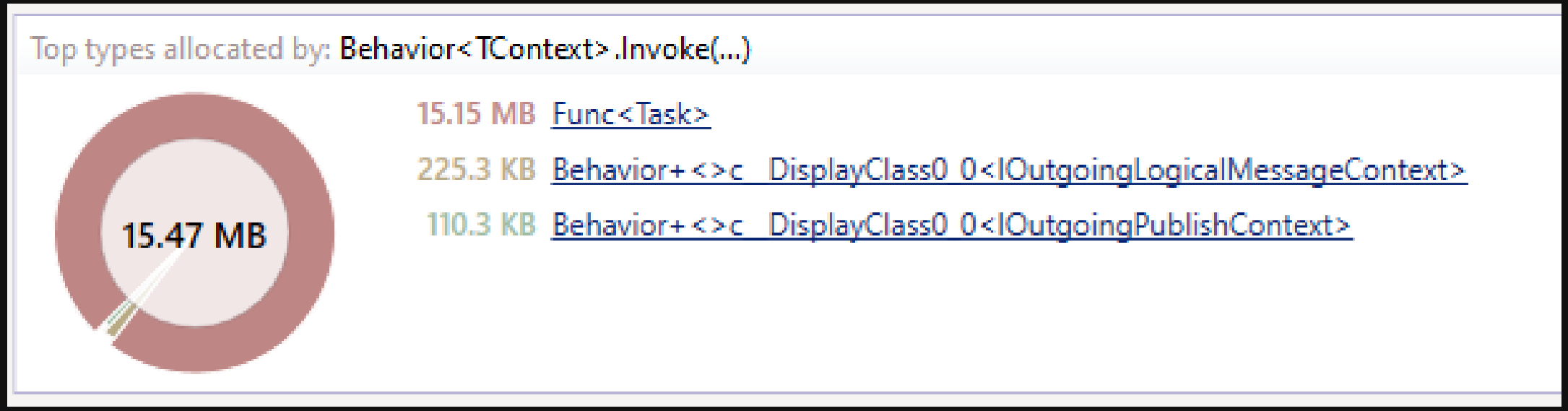
BEHAVIORCHAIN

🔁 Profiling the pipeline

MEMORY CHARACTERISTICS

CONTEXT MATTERS

MEMORY CHARACTERISTICS

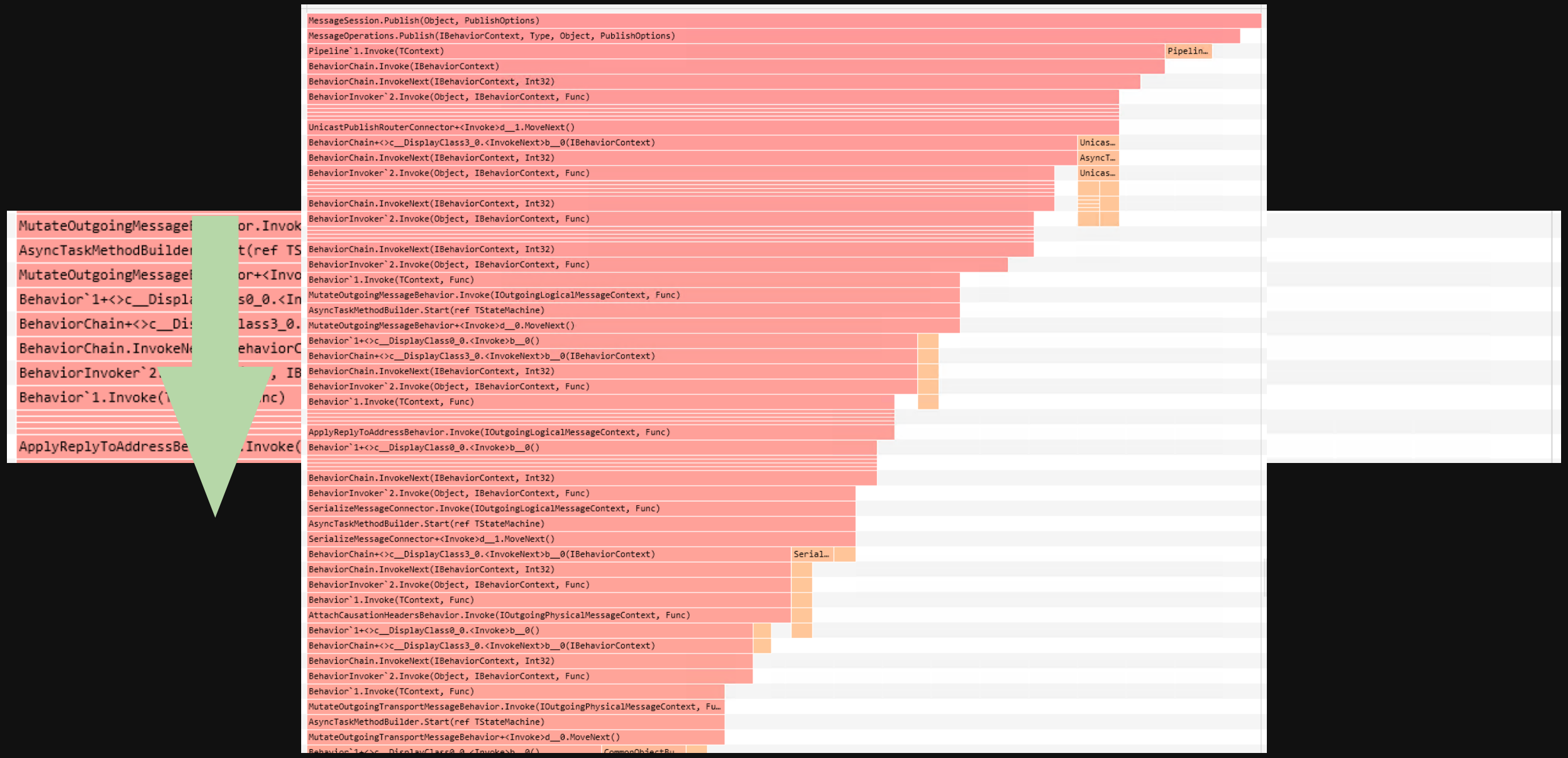


MEMORY CHARACTERISTICS

Type	Bytes
StageForkConnector+<>c__DisplayClass0_0<ITransportReceiveContext, IIncomingPhysicalM	15.74 MB
Behavior+<>c__DisplayClass0_0<IOutgoingLogicalMessageContext> NServiceBus.Pipeline	5.92 MB
Behavior+<>c__DisplayClass0_0<IRoutingContext> NServiceBus.Pipeline	315.7 KB
Behavior+<>c__DisplayClass0_0<IOutgoingPhysicalMessageContext> NServiceBus.Pipelin	205.9 KB

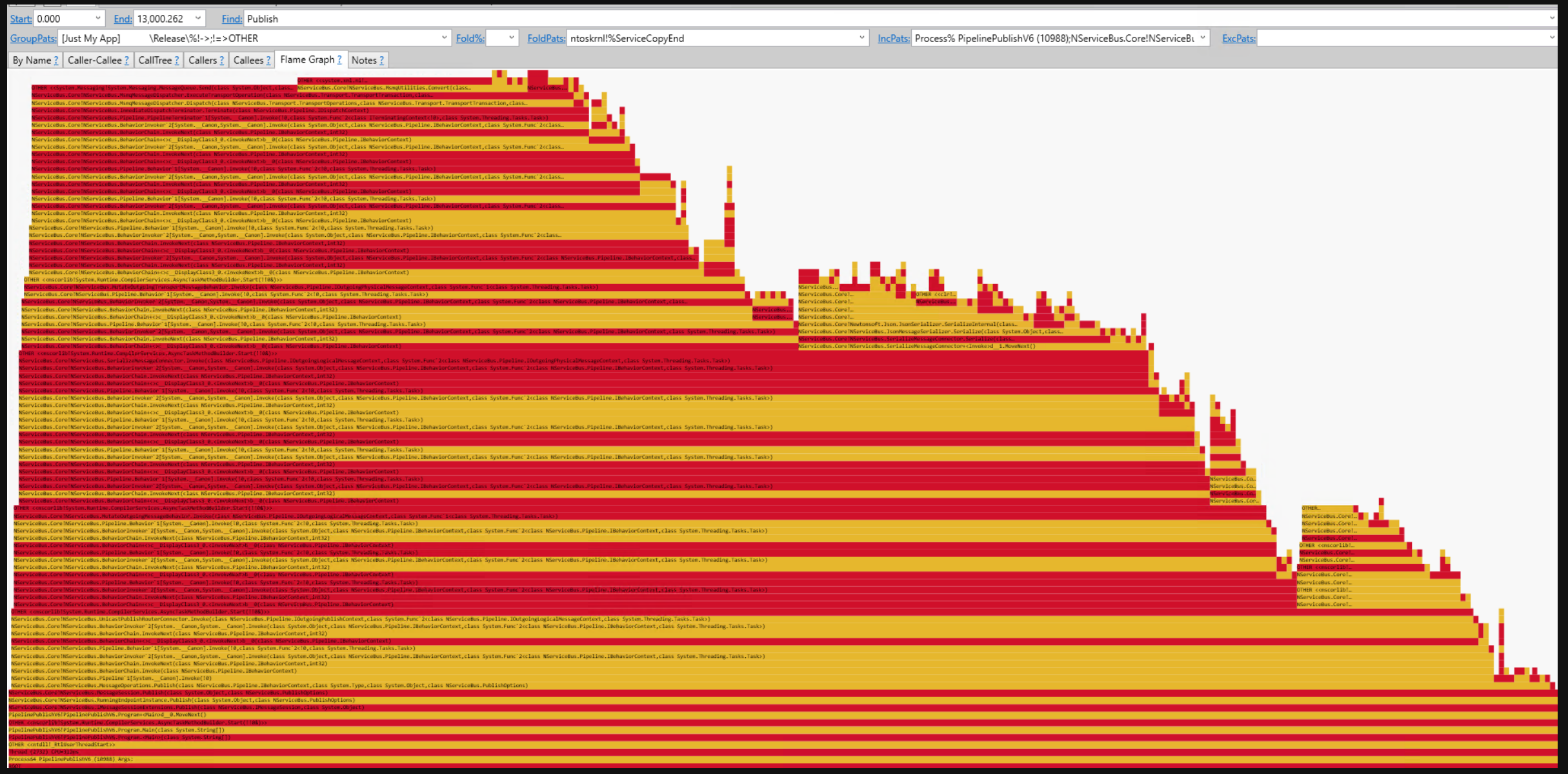
CPU CHARACTERISTICS

CPU CHARACTERISTICS



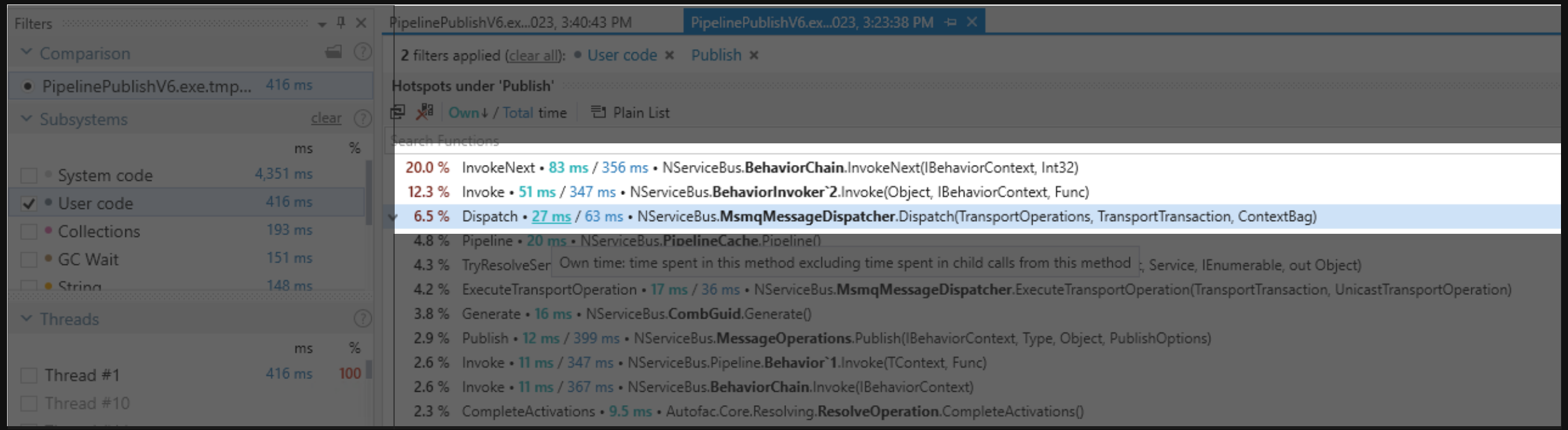
Profiling the pipeline

CPU CHARACTERISTICS



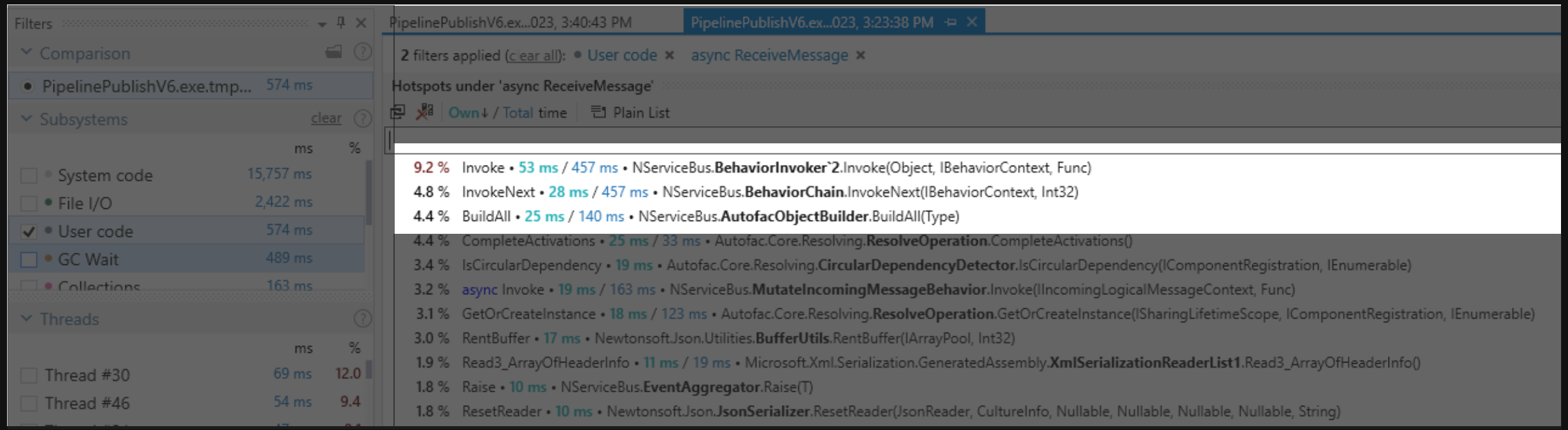
Profiling the pipeline

CPU CHARACTERISTICS



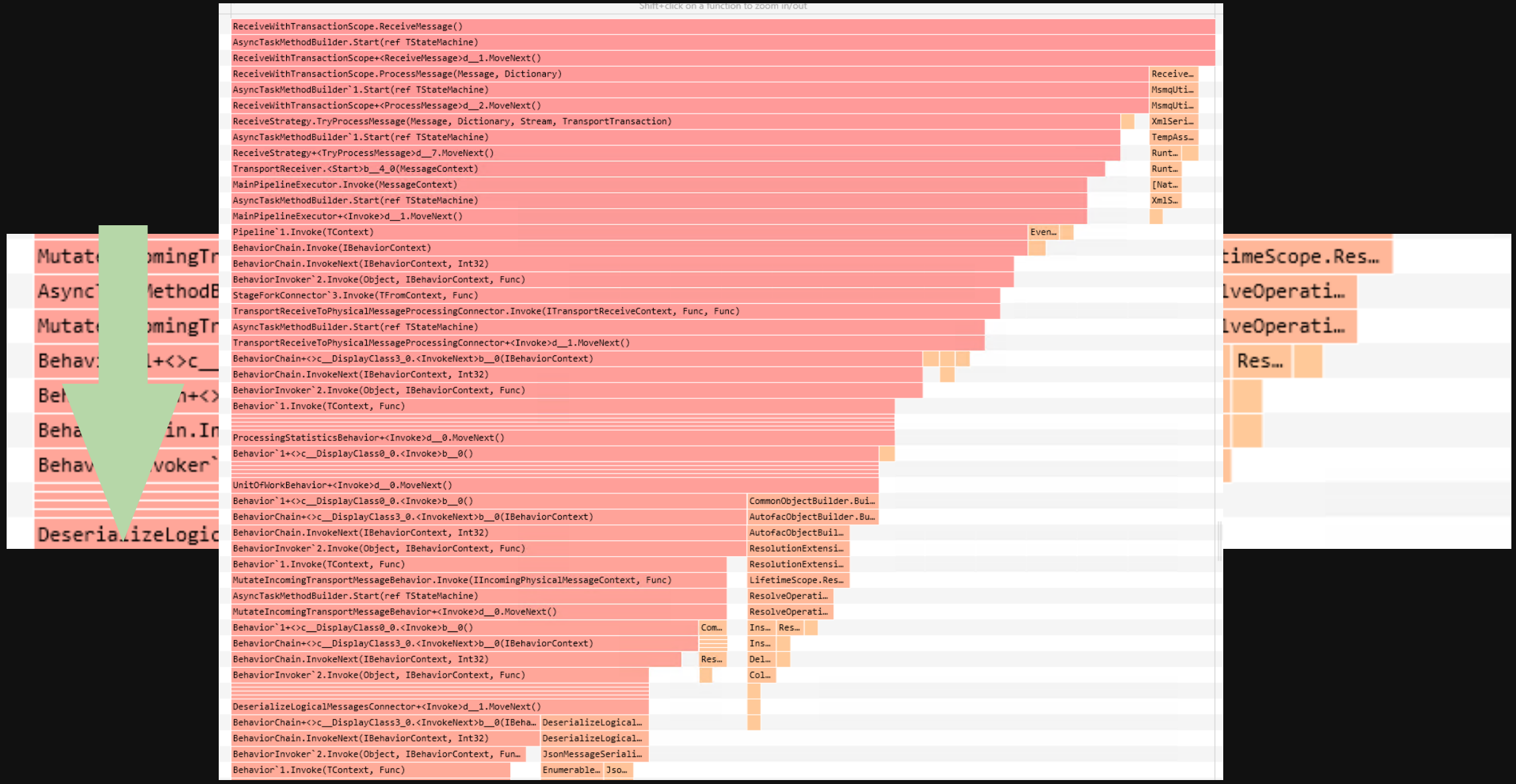
PUBLISH

CPU CHARACTERISTICS



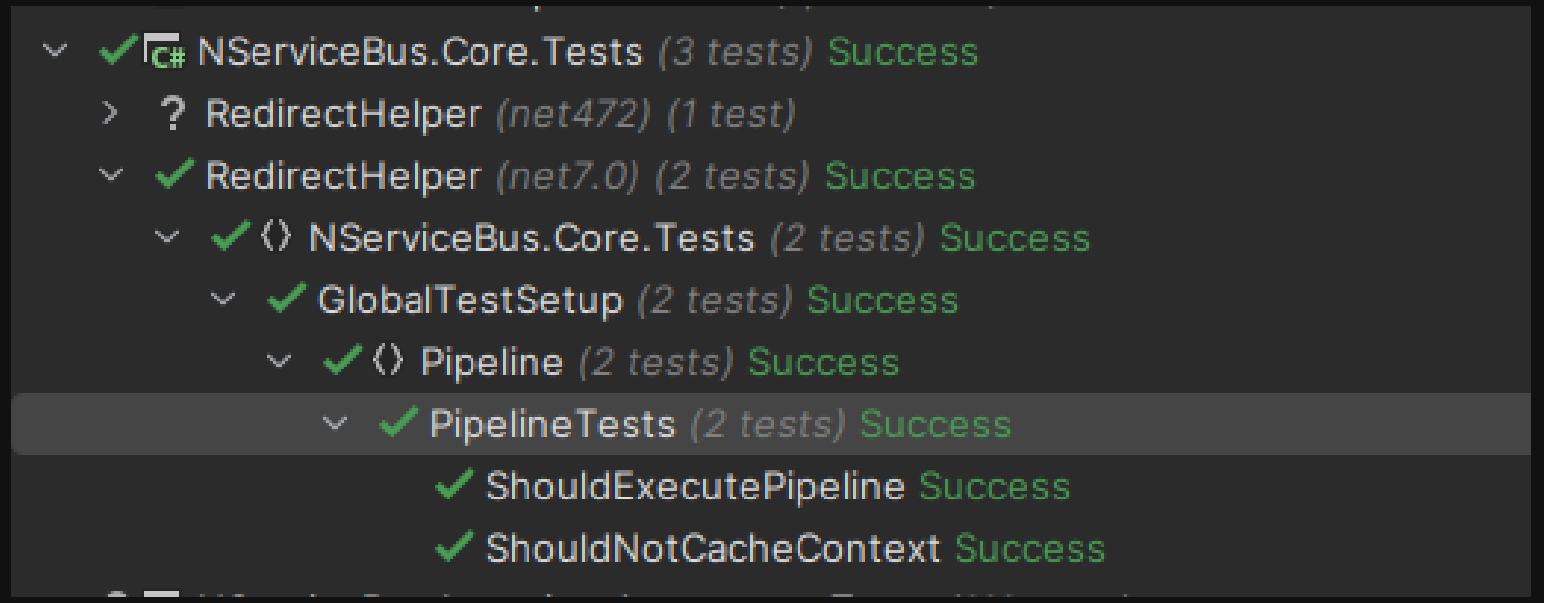
RECEIVE

CPU CHARACTERISTICS



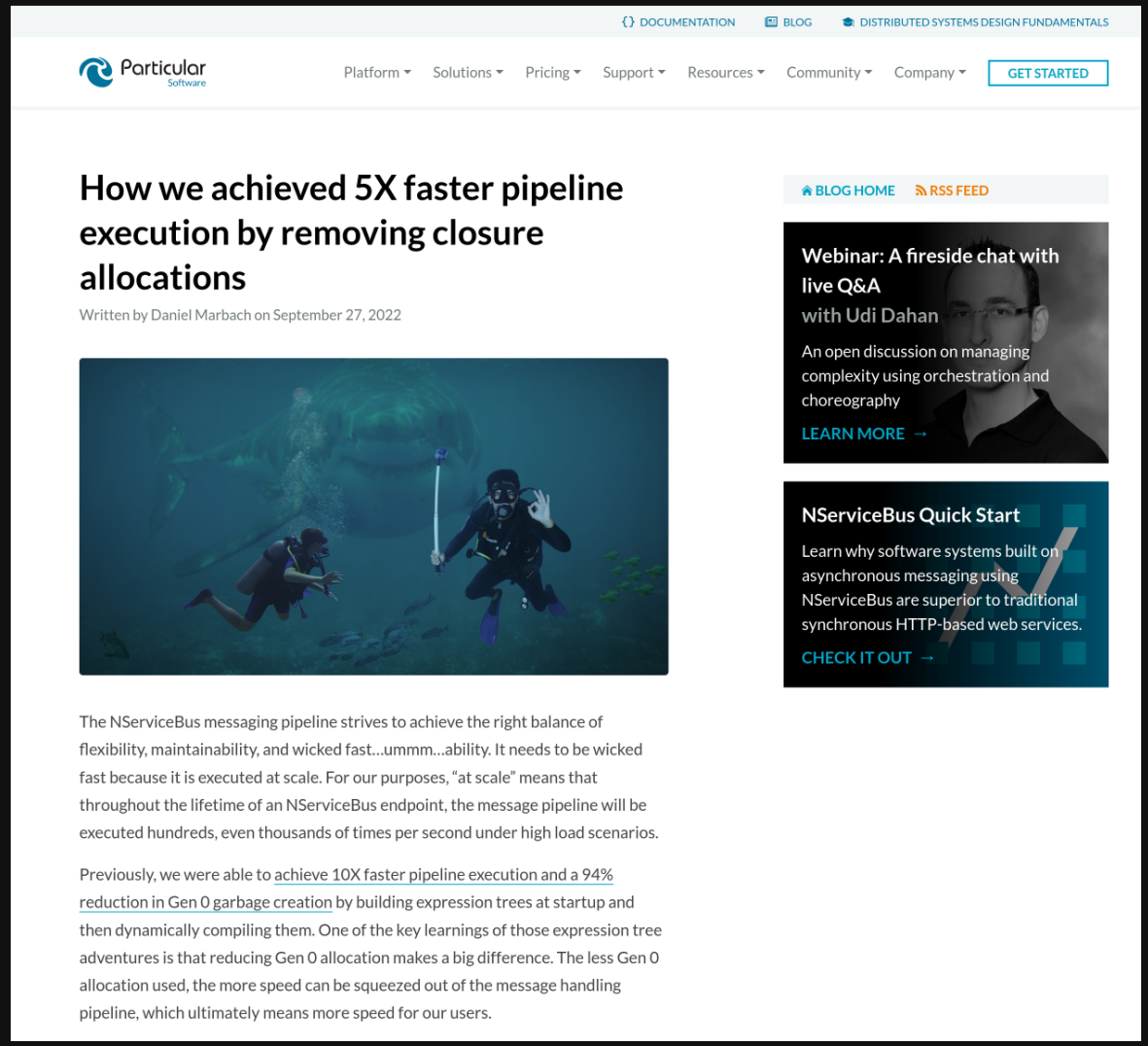
Profiling the pipeline

TESTING


NServiceBus.Core.Tests (3 tests) Success
RedirectHelper (net472) (1 test)
RedirectHelper (net7.0) (2 tests) Success
NServiceBus.Core.Tests (2 tests) Success
GlobalTestSetup (2 tests) Success
Pipeline (2 tests) Success
PipelineTests (2 tests) Success
ShouldExecutePipeline Success
ShouldNotCacheContext Success

IMPROVING

- 10X faster execution with compiled expression trees
- How we achieved 5X faster pipeline execution by removing closure allocations

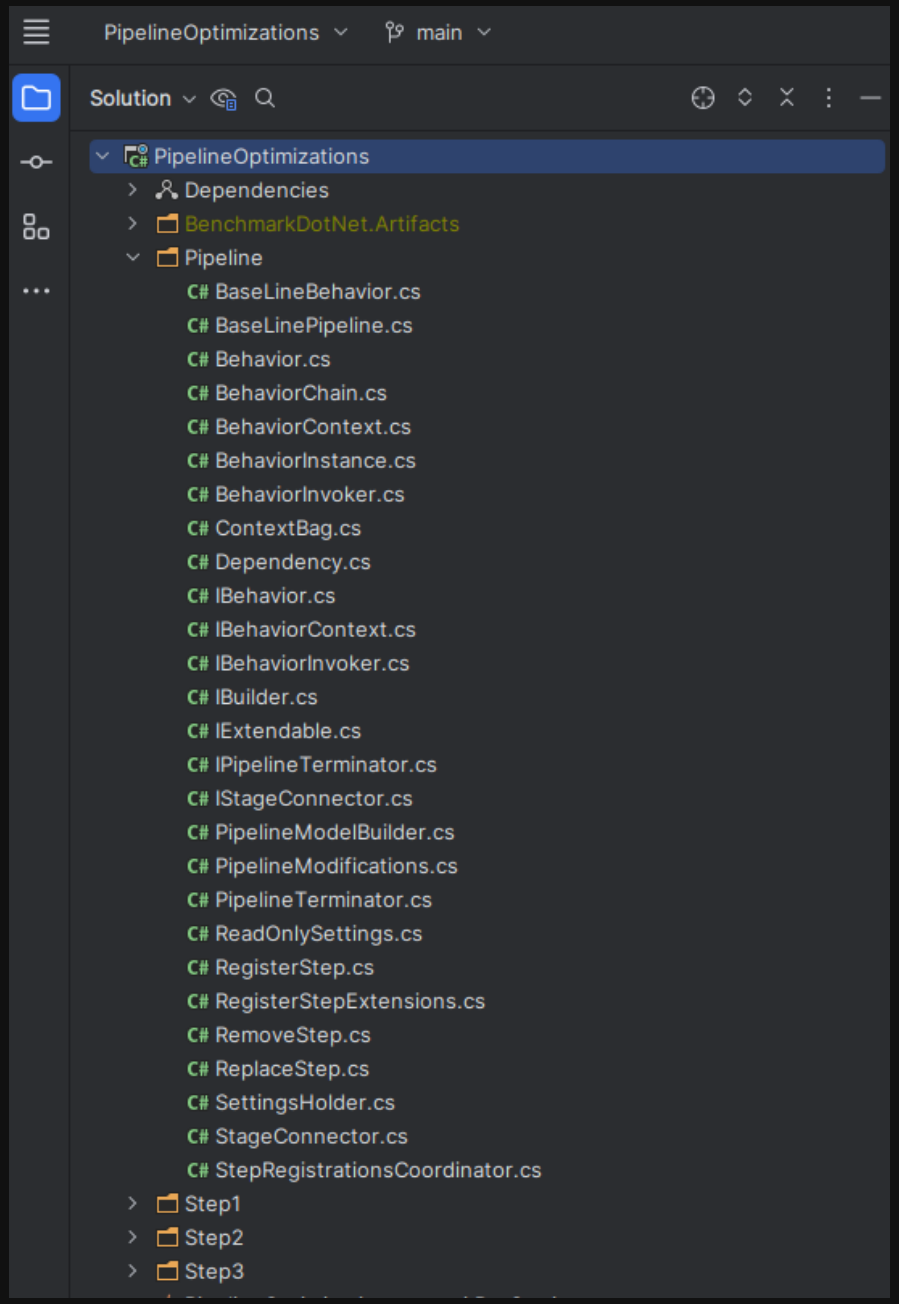


go.particular.net/webinar-2023-pipeline

BENCHMARKING THE PIPELINE

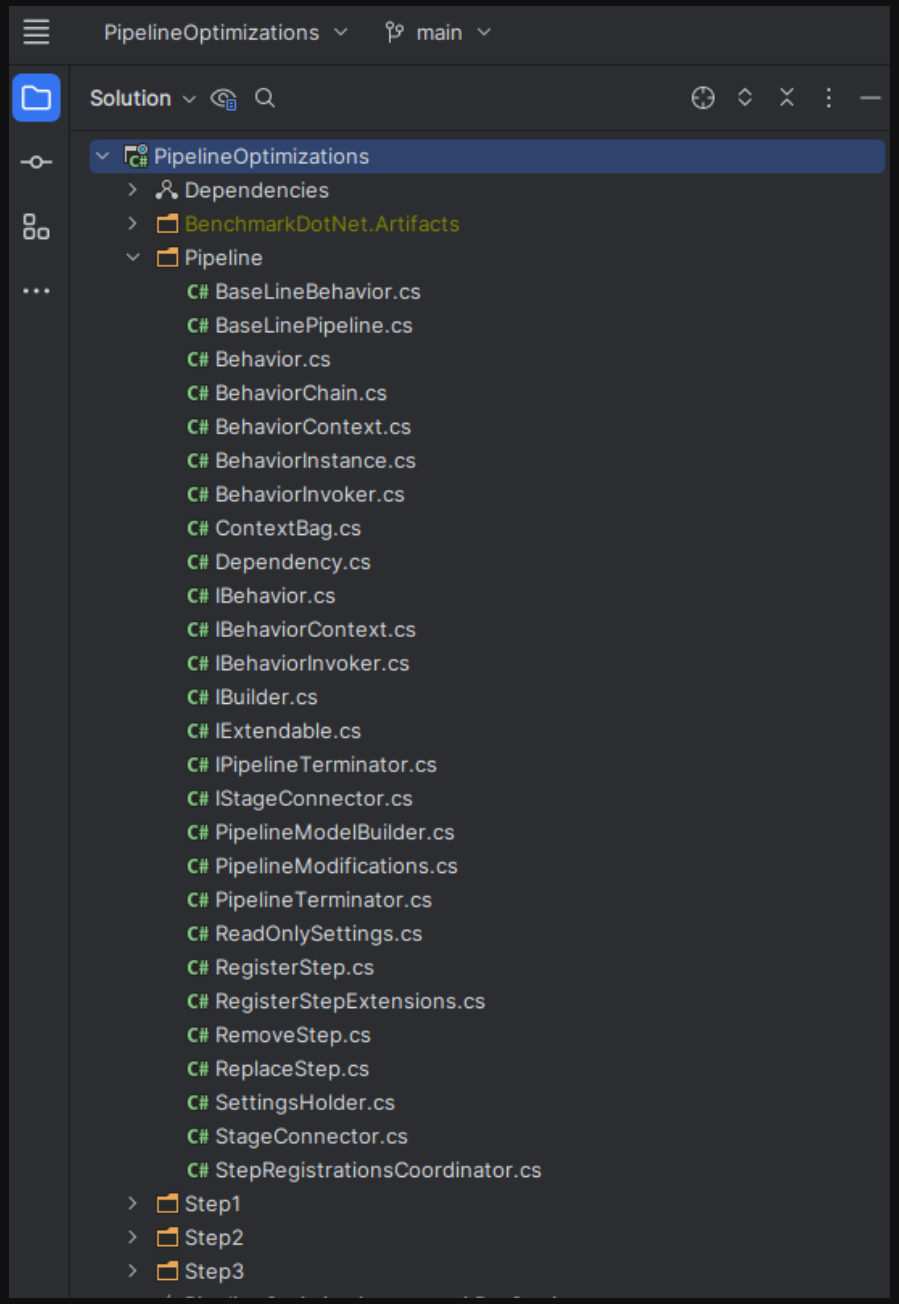
EXTRACT CODE

- Copy and paste relevant code
- Adjust it to the bare essentials to create a controllable environment



EXTRACT CODE

- Trim down to relevant behaviors
- Replaced dependency injection container with creating relevant classes
- Replaced IO-operations with completed tasks



PERFORMANCE CULTURE

- Get started with small steps
- Culture change takes time
- Make changes gradually

```
[ShortRunJob]
[MemoryDiagnoser]
public class PipelineExecution {

    [Params(10, 20, 40)]
    public int PipelineDepth { get; set; }


    [GlobalSetup]
    public void SetUp()  {
        behaviorContext = new BehaviorContext();

        pipelineModificationsBeforeOptimizations = new PipelineModifications();
        for (int i = 0; i < PipelineDepth; i++)
        {
            pipelineModificationsBeforeOptimizations.Additions.Add(RegisterStep.Create(i.ToString(),
                typeof(BaseLineBehavior), i.ToString(), b => new BaseLineBehavior()));
        }

        pipelineModificationsAfterOptimizations = new PipelineModifications();
        for (int i = 0; i < PipelineDepth; i++)
        {
            pipelineModificationsAfterOptimizations.Additions.Add(RegisterStep.Create(i.ToString(),
                typeof(BehaviorOptimization), i.ToString(), b => new BehaviorOptimization()));
        }

        pipelineBeforeOptimizations = new BaseLinePipeline<IBehaviorContext>(null, new SettingsHolder(),
            pipelineModificationsBeforeOptimizations);
        pipelineAfterOptimizations = new PipelineOptimization<IBehaviorContext>(null, new SettingsHolder(),
            pipelineModificationsAfterOptimizations);
    }

    [Benchmark(Baseline = true)]
    public async Task Before() {
        await pipelineBeforeOptimizations.Invoke(behaviorContext);
    }

    [Benchmark]
    public async Task After() {
        await pipelineAfterOptimizations.Invoke(behaviorContext);
    }
}
```

Benchmarking the pipeline

```
[ShortRunJob]
[MemoryDiagnoser]
public class PipelineExecution {

    [Params(10, 20, 40)]
    public int PipelineDepth { get; set; }


    [GlobalSetup]
    public void SetUp()  {
        behaviorContext = new BehaviorContext();

        pipelineModificationsBeforeOptimizations = new PipelineModifications();
        for (int i = 0; i < PipelineDepth; i++)
        {
            pipelineModificationsBeforeOptimizations.Additions.Add(RegisterStep.Create(i.ToString(),
                typeof(BaseLineBehavior), i.ToString(), b => new BaseLineBehavior()));
        }

        pipelineModificationsAfterOptimizations = new PipelineModifications();
        for (int i = 0; i < PipelineDepth; i++)
        {
            pipelineModificationsAfterOptimizations.Additions.Add(RegisterStep.Create(i.ToString(),
                typeof(BehaviorOptimization), i.ToString(), b => new BehaviorOptimization()));
        }

        pipelineBeforeOptimizations = new BaseLinePipeline<IBehaviorContext>(null, new SettingsHolder(),
            pipelineModificationsBeforeOptimizations);
        pipelineAfterOptimizations = new PipelineOptimization<IBehaviorContext>(null, new SettingsHolder(),
            pipelineModificationsAfterOptimizations);
    }

    [Benchmark(Baseline = true)]
    public async Task Before() {
        await pipelineBeforeOptimizations.Invoke(behaviorContext);
    }

    [Benchmark]
    public async Task After() {
        await pipelineAfterOptimizations.Invoke(behaviorContext);
    }
}
```

Benchmarking the pipeline

```
[ShortRunJob]
[MemoryDiagnoser]
public class PipelineExecution {

    [Params(10, 20, 40)]
    public int PipelineDepth { get; set; }


    [GlobalSetup]
    public void SetUp()  {
        behaviorContext = new BehaviorContext();

        pipelineModificationsBeforeOptimizations = new PipelineModifications();
        for (int i = 0; i < PipelineDepth; i++)
        {
            pipelineModificationsBeforeOptimizations.Additions.Add(RegisterStep.Create(i.ToString(),
                typeof(BaseLineBehavior), i.ToString(), b => new BaseLineBehavior()));
        }

        pipelineModificationsAfterOptimizations = new PipelineModifications();
        for (int i = 0; i < PipelineDepth; i++)
        {
            pipelineModificationsAfterOptimizations.Additions.Add(RegisterStep.Create(i.ToString(),
                typeof(BehaviorOptimization), i.ToString(), b => new BehaviorOptimization()));
        }

        pipelineBeforeOptimizations = new BaseLinePipeline<IBehaviorContext>(null, new SettingsHolder(),
            pipelineModificationsBeforeOptimizations);
        pipelineAfterOptimizations = new PipelineOptimization<IBehaviorContext>(null, new SettingsHolder(),
            pipelineModificationsAfterOptimizations);
    }

    [Benchmark(Baseline = true)]
    public async Task Before() {
        await pipelineBeforeOptimizations.Invoke(behaviorContext);
    }

    [Benchmark]
    public async Task After() {
        await pipelineAfterOptimizations.Invoke(behaviorContext);
    }
}
```

Benchmarking the pipeline

PRACTICES

- Single Responsibility Principle
- No side effects
- Prevents dead code elimination
- Delegates heavy lifting to the framework
- Is explicit
  - No implicit casting
  - No var
- Avoid running any other resource-heavy processes while benchmarking

BenchmarkDotNet

Powerful .NET library for benchmarking

Benchmarking is really hard

BenchmarkDotNet will protect you from the common pitfalls
because it does all the dirty work for you

```
[ShortRunJob]
[MemoryDiagnoser]
public class Step1_PipelineWarmup {
    // rest almost the same

    [Benchmark(Baseline = true)]
    public BaseLinePipeline<IBehaviorContext> Before() {
        var pipelineBeforeOptimizations = new
BaseLinePipeline<IBehaviorContext>(null, new SettingsHolder(),
            pipelineModificationsBeforeOptimizations);
        return pipelineBeforeOptimizations;
    }

    [Benchmark]
    public PipelineOptimization<IBehaviorContext> After() {
        var pipelineAfterOptimizations = new
PipelineOptimization<IBehaviorContext>(null, new SettingsHolder(),
            pipelineModificationsAfterOptimizations);
        return pipelineAfterOptimizations;
    }
}
```

Benchmarking the pipeline

```
[ShortRunJob]
[MemoryDiagnoser]
public class Step2_PipelineException {
    [GlobalSetup]
    public void SetUp() {
        ...
        var stepdId = PipelineDepth + 1;
        pipelineModificationsBeforeOptimizations.Additions.Add(RegisterStep.Create(stepdId.ToString(), typeof(Throwing), "1", b
=> new Throwing()));

        ...
        pipelineModificationsAfterOptimizations.Additions.Add(RegisterStep.Create(stepdId.ToString(), typeof(Throwing), "1", b
=> new Throwing()));

        pipelineBeforeOptimizations = new Step1.PipelineOptimization<IBehaviorContext>(null, new SettingsHolder(),
            pipelineModificationsBeforeOptimizations);
        pipelineAfterOptimizations = new PipelineOptimization<IBehaviorContext>(null, new SettingsHolder(),
            pipelineModificationsAfterOptimizations);
    }

    [Benchmark(Baseline = true)]
    public async Task Before() {
        try
        {
            await pipelineBeforeOptimizations.Invoke(behaviorContext).ConfigureAwait(false);
        }
        catch (InvalidOperationException)
        {
        }
    }

    [Benchmark]
    public async Task After() {
        try
        {
            await pipelineAfterOptimizations.Invoke(behaviorContext).ConfigureAwait(false);
        }
        catch (InvalidOperationException)
        {
        }
    }

    class Throwing : Behavior<IBehaviorContext> {
        public override Task Invoke(IBehaviorContext context, Func<Task> next)
        {
            throw new InvalidOperationException();
        }
    }
}
```

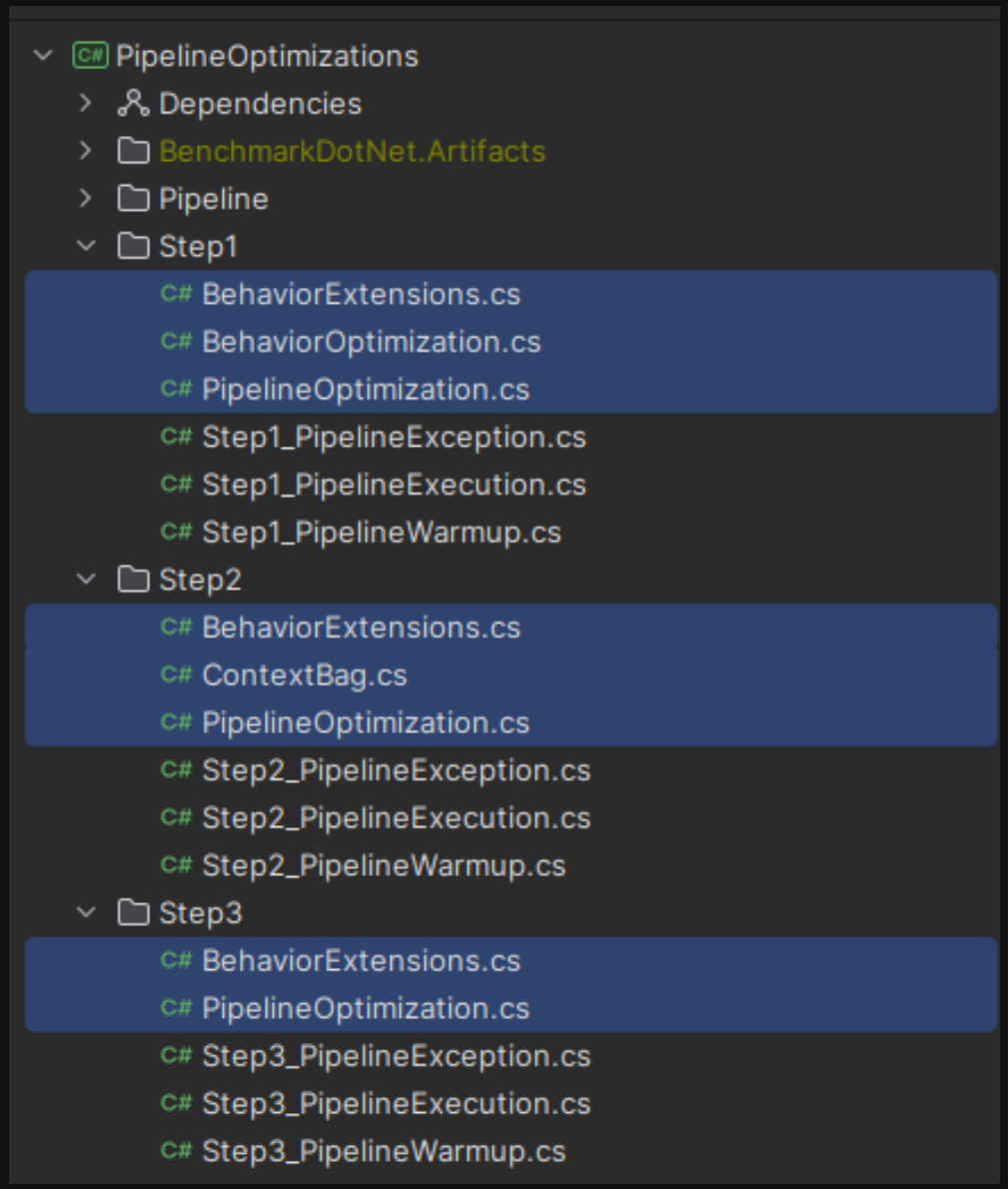
Benchmarking the pipeline

```
1  [ShortRunJob]
2  [MemoryDiagnoser]
3  public class Step2_PipelineException {
4      [GlobalSetup]
5      public void SetUp() {
6          ...
7          var stepdId = PipelineDepth + 1;
8
   pipelineModificationsBeforeOptimizations.Additions.Add(RegisterStep.Create(stepd
   Id.ToString(), typeof(Throwing), "1", b => new Throwing()));
9
10         ...
11
   pipelineModificationsAfterOptimizations.Additions.Add(RegisterStep.Create(stepdI
   d.ToString(), typeof(Throwing), "1", b => new Throwing()));
12
13         pipelineBeforeOptimizations = new
   Step1.PipelineOptimization<IBehaviorContext>(null, new SettingsHolder(),
14             pipelineModificationsBeforeOptimizations);
15         pipelineAfterOptimizations = new PipelineOptimization<IBehaviorContext>
   (null, new SettingsHolder(),
16             pipelineModificationsAfterOptimizations);
17     }
18 }
```

🔁 Benchmarking the pipeline

```
[ShortRunJob]
[MemoryDiagnoser]
public class Step2_PipelineException {
    [GlobalSetup]
    public void SetUp() {
      ...
    }

    [Benchmark(Baseline = true)]
    public async Task Before() {
        try
        {
            await pipelineBeforeOptimizations.Invoke(behaviorContext);
        }
        catch (InvalidOperationException)
        {
        }
    }

    [Benchmark]
    public async Task After() {
        try
        {
            await pipelineAfterOptimizations.Invoke(behaviorContext);
        }
        catch (InvalidOperationException)
        {
        }
    }
    ...
}
```

🔁 Benchmarking the pipeline

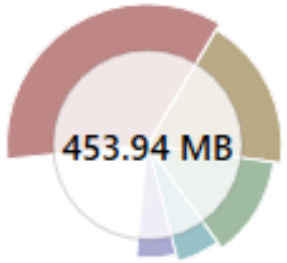


🔁 Benchmarking the pipeline

PROFILING THE PIPELINE (AGAIN)

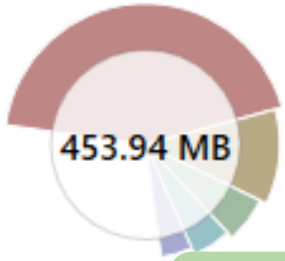
MEMORY CHARACTERISTICS

Top types allocated by: NServiceBus.IMessageSessionExtensions



159.75 MB Byte[]
84.39 MB Char[]
58.78 MB Int32[]
26.53 MB ArrayList
25.77 MB String

Top allocating methods in: NServiceBus.IMessageSessionExtensions

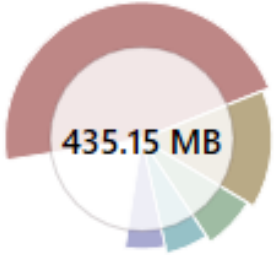


197.61 MB StreamWriter.Init(Stream, Encoding, Int32, Boolean)
52.07 MB String.SplitInternal(Char[], Int32, StringSplitOptions)
26.53 MB MessageQueuePermission.get_PermissionEntries()
23.90 MB MemoryStream.set_Capacity(Int32)
20.22 MB BehaviorChain.InvokeNext(IBehaviorContext, Int32)

BEFORE

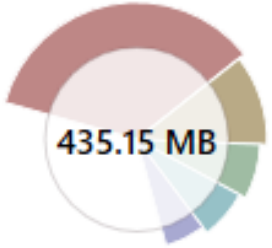
PUBLISH

Top types allocated by: NServiceBus.IMessageSessionExtensions



201.96 MB Byte[]
63.77 MB Char[]
30.59 MB Int32[]
25.92 MB Lookup+Grouping<String, Subscriber>[]
25.48 MB String

Top allocating methods in: NServiceBus.IMessageSessionExtensions

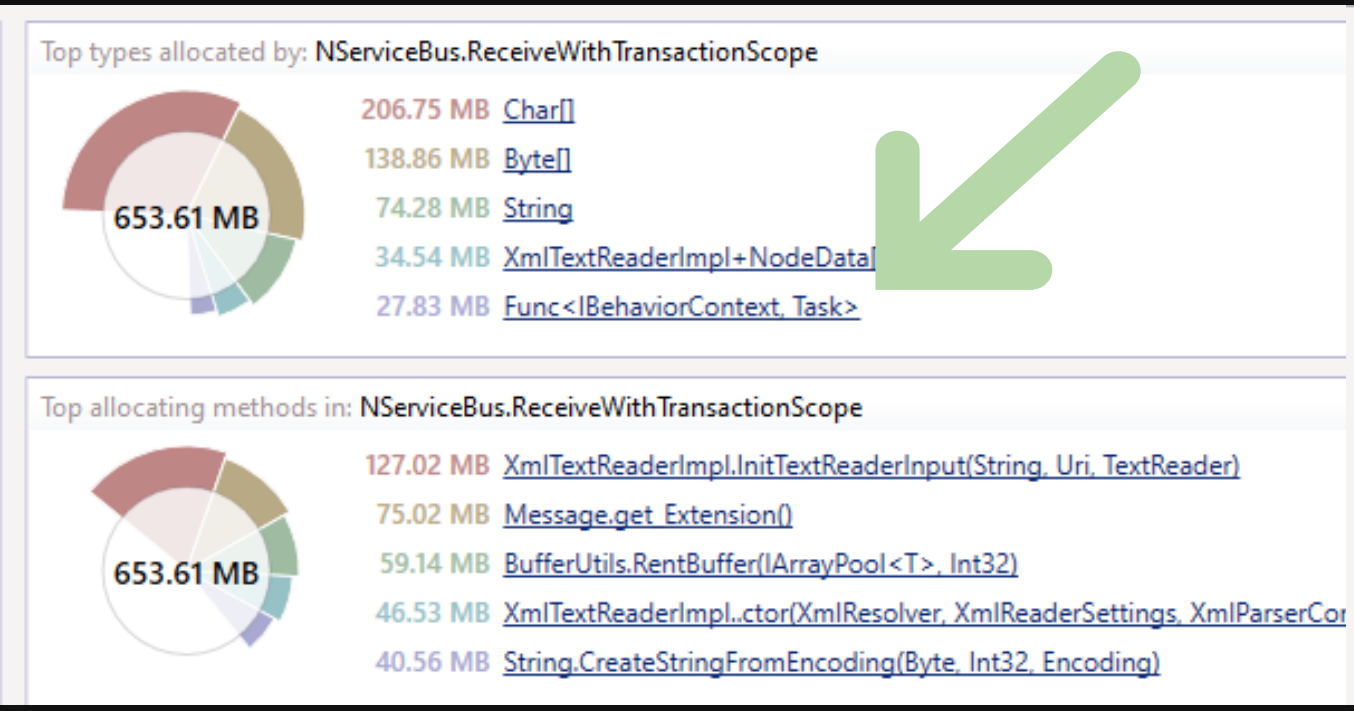


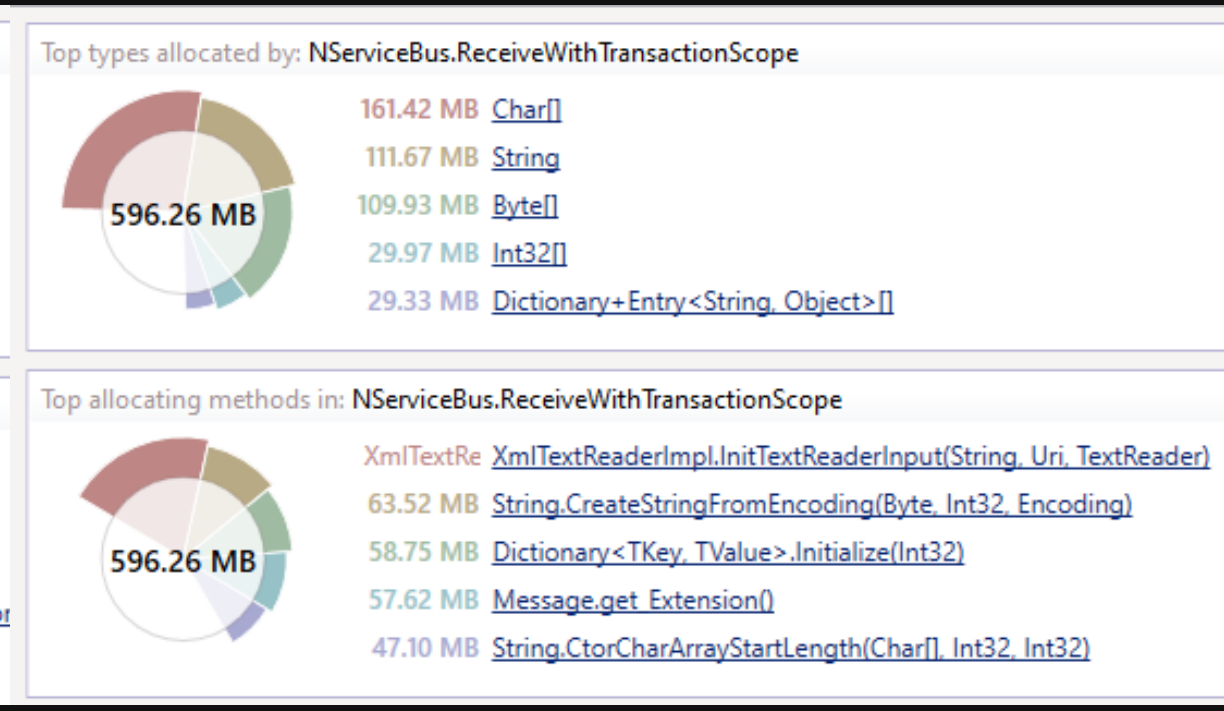
152.55 MB StreamWriter.Init(Stream, Encoding, Int32, Boolean)
49.01 MB MemoryStream.set_Capacity(Int32)
31.30 MB MemoryStream.ToArray()
30.02 MB Message.IdFromByteArray(Byte[])
25.92 MB Lookup<TKey, TElement>..ctor(IEqualityComparer<T>)

AFTER

🔁 Profiling the pipeline (again)

MEMORY CHARACTERISTICS

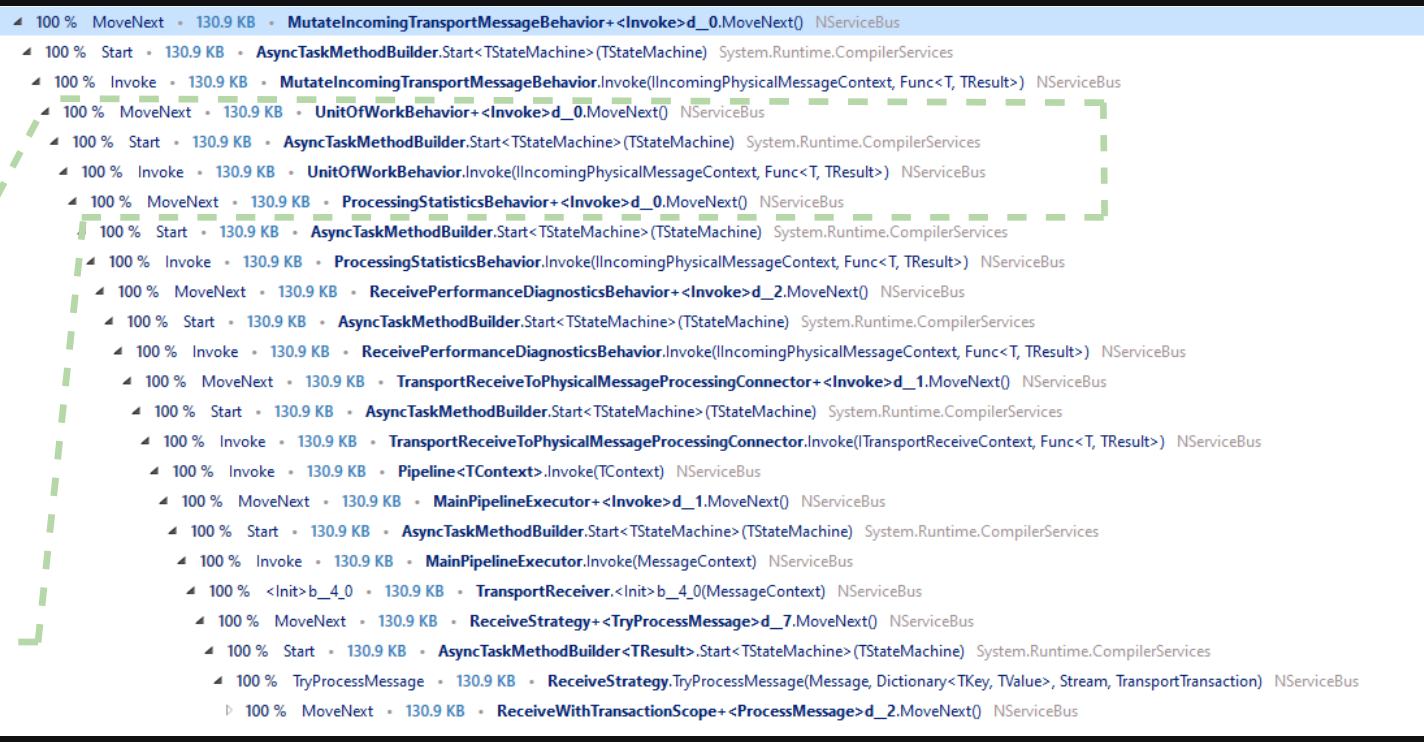




BEFORE RECEIVE AFTER

🔁 Profiling the pipeline (again)

MEMORY CHARACTERISTICS



BEFORE RECEIVE AFTER

🔁 Profiling the pipeline (again)

MEMORY CHARACTERISTICS

Type	Bytes
StageForkConnector+<>c__DisplayClass0_0<ITransportReceiveContext, IIncomingPhysicalM	15.74 MB
Behavior+<>c__DisplayClass0_0<IOutgoingLogicalMessageContext> NServiceBus.Pipeline	5.92 MB
Behavior+<>c__DisplayClass0_0<IRoutingContext> NServiceBus.Pipeline	315.7 KB
Behavior+<>c__DisplayClass0_0<IOutgoingPhysicalMessageContext> NServiceBus.Pipelin	205.9 KB

BEFORE

Clear:ageSessionExtensionsıy objects of a certain type were allocated during the selected time intervClear:ageSessionExtensi Bytes:ageS

Filter: NServiceBus.Pipeline

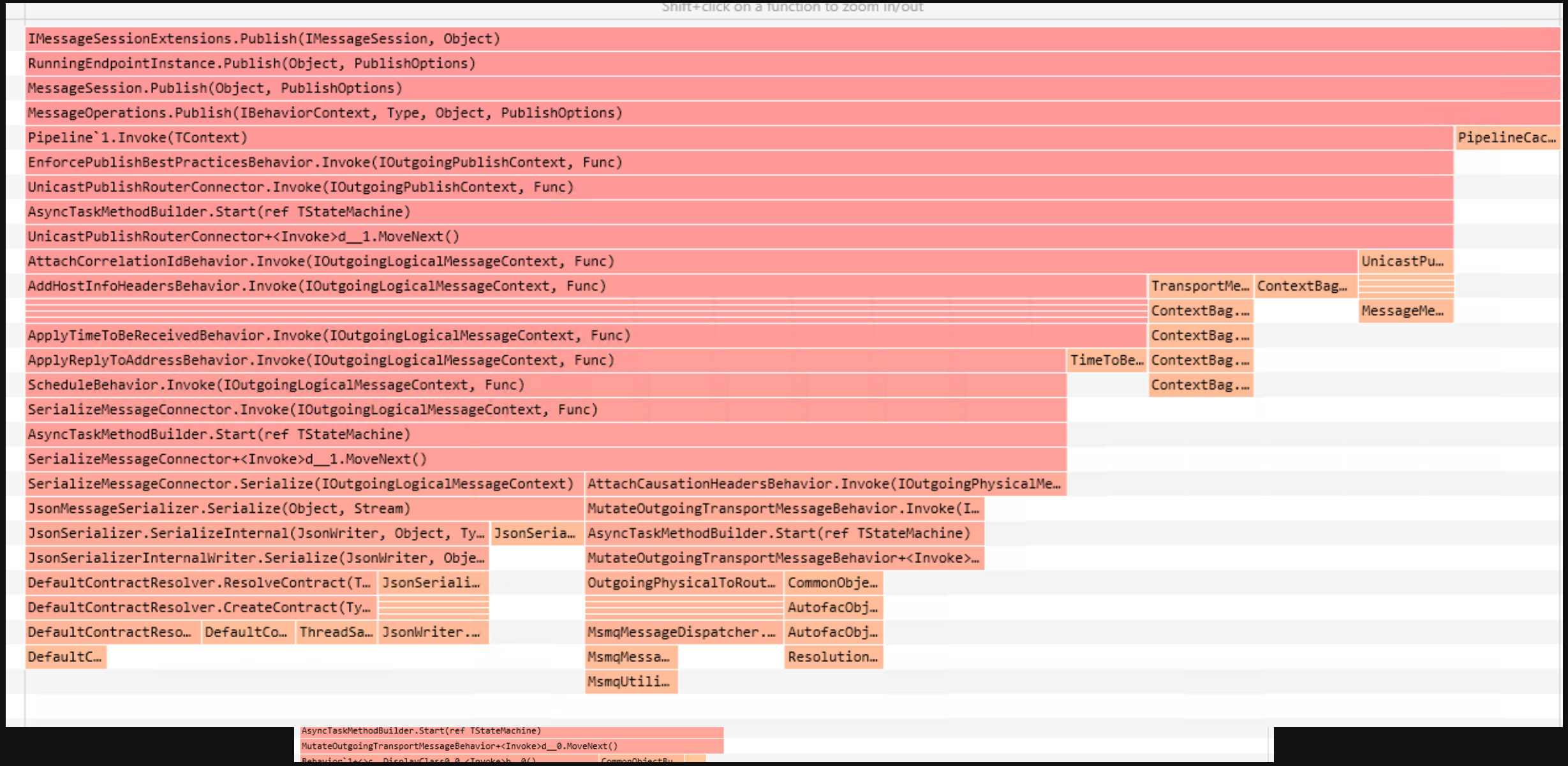
Type	Bytes
LogicalMessage[] NServiceBus.Pipeline	130.9 KB

AFTER

oh look, there is nothing 😌

🔁 Profiling the pipeline (again)

CPU CHARACTERISTICS

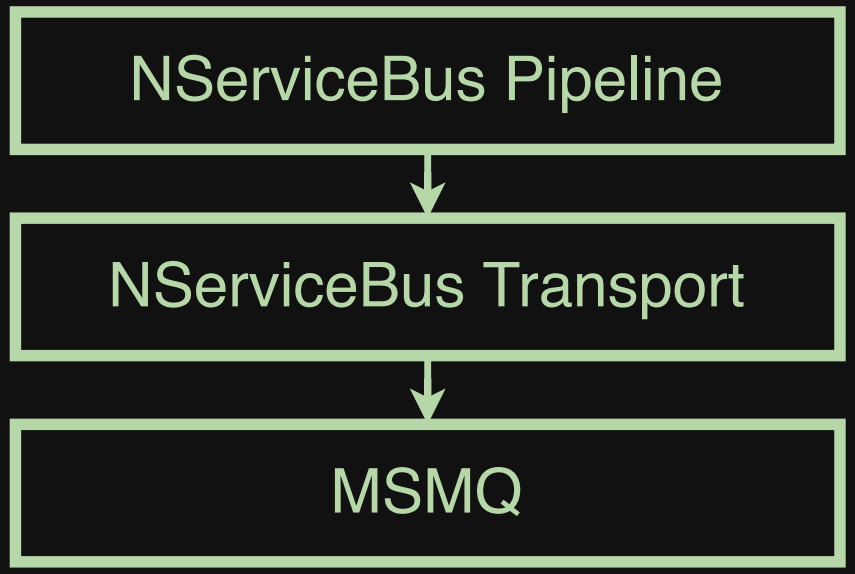


🔁 Profiling the pipeline (again)

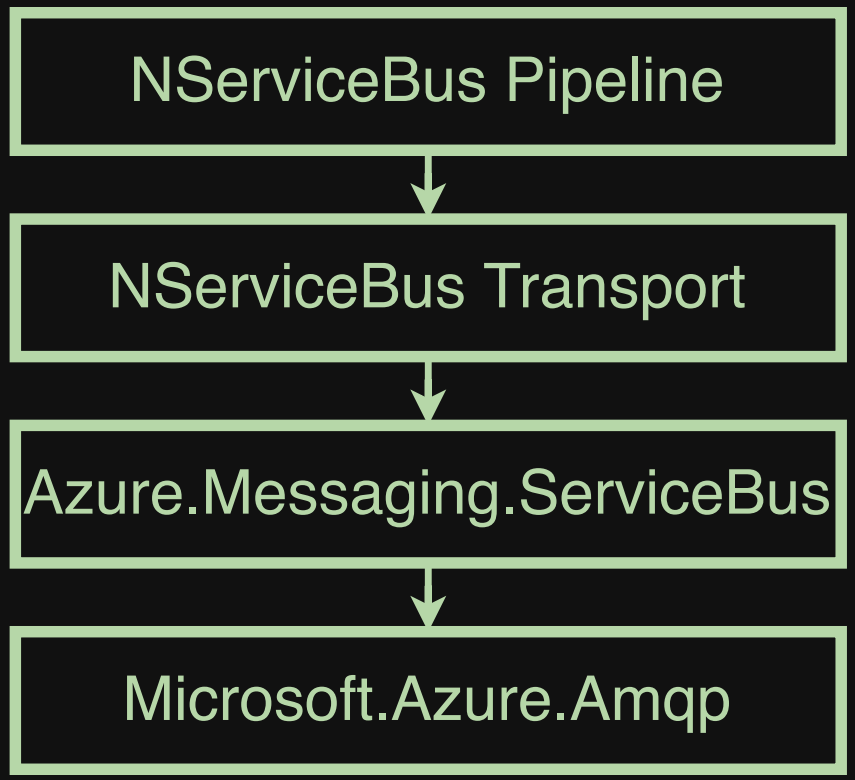
CPU CHARACTERISTICS

🔁 Profiling the pipeline (again)

GETTING LOWER ON THE STACK



GETTING LOWER ON THE STACK



THE HARNESS

```
await using var serviceBusClient = new ServiceBusClient(connectionString);

await using var sender = serviceBusClient.CreateSender(destination);
var messages = new List<ServiceBusMessage>(1000);
for (int i = 0; i < 1000; i++) {
    messages.Add(new ServiceBusMessage(UTF8.GetBytes($"Deep Dive {i} Deep Dive {i} Deep Dive {i} Deep Dive {i} Deep Dive {i} Deep Dive {i}")));

    if (i % 100 == 0) {
        await sender.SendMessagesAsync(messages);
        messages.Clear();
    }
}

await sender.SendMessagesAsync(messages);

WriteLine("Messages sent");
Console.WriteLine("Take snapshot");
Console.ReadLine();

var countDownEvent = new CountdownEvent(1000);

var processorOptions = new ServiceBusProcessorOptions {
    AutoCompleteMessages = true,
    MaxConcurrentCalls = 100,
    MaxAutoLockRenewalDuration = TimeSpan.FromMinutes(10),
    ReceiveMode = ServiceBusReceiveMode.PeekLock,
};

await using var receiver = serviceBusClient.CreateProcessor(destination, processorOptions);
receiver.ProcessMessageAsync += async messageEventArgs => {
    var message = messageEventArgs.Message;
    await Out.WriteLineAsync(
        $"Received message with '{message.MessageId}' and content '{UTF8.GetString(message.Body)}' / binary {message.Body}");
    countDownEvent.Signal();
};
// rest omitted
await receiver.StartProcessingAsync();

countDownEvent.Wait();

Console.WriteLine("Take snapshot");
Console.ReadLine();

await receiver.StopProcessingAsync();
```

Getting lower on the stack

THE HARNESS

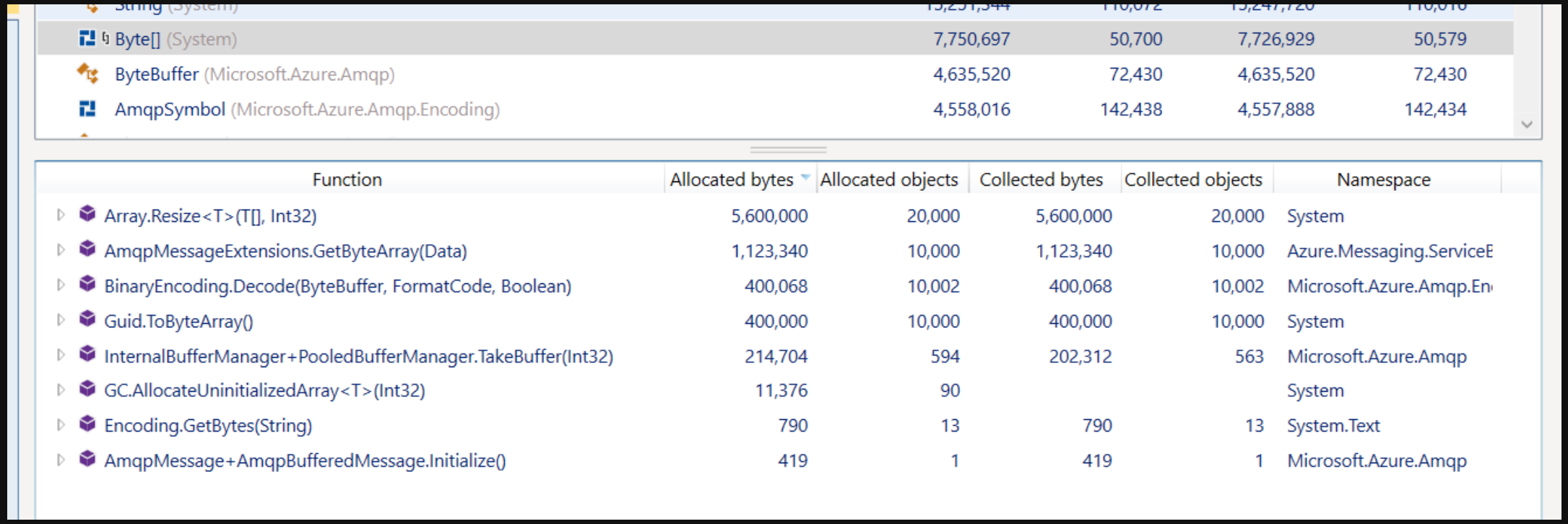
```
await using var serviceBusClient = new ServiceBusClient(connectionString);

await using var sender = serviceBusClient.CreateSender(destination);
var messages = new List<ServiceBusMessage>(1000);
for (int i = 0; i < 1000; i++) {
    messages.Add(new ServiceBusMessage(UTF8.GetBytes($"Deep Dive {i} Deep Dive {i} Deep Dive {i} Deep Dive {i} Deep
Dive {i} Deep Dive {i}")));

    if (i % 100 == 0) {
        await sender.SendMessagesAsync(messages);
        messages.Clear();
    }
}

await sender.SendMessagesAsync(messages);

WriteLine("Messages sent");
Console.WriteLine("Take snapshot");
Console.ReadLine();
```

THE HARNESS

```
var countDownEvent = new CountdownEvent(1000);

var processorOptions = new ServiceBusProcessorOptions
{
    AutoCompleteMessages = true,
    MaxConcurrentCalls = 100,
    MaxAutoLockRenewalDuration = TimeSpan.FromMinutes(10),
    ReceiveMode = ServiceBusReceiveMode.PeekLock,
};

await using var receiver = serviceBusClient.CreateProcessor(destination, processorOptions);
receiver.ProcessMessageAsync += async messageEventArgs => {
    var message = messageEventArgs.Message;
    await Out.WriteLineAsync(
        $"Received message with '{message.MessageId}' and content '{UTF8.GetString(message.Body)}' / binary
{message.Body}");
    countDownEvent.Signal();
};
// rest omitted
await receiver.StartProcessingAsync();

countDownEvent.Wait();

Console.WriteLine("Take snapshot");
Console.ReadLine();

await receiver.StopProcessingAsync();
```

Getting lower on the stack

MEMORY CHARACTERISTICS

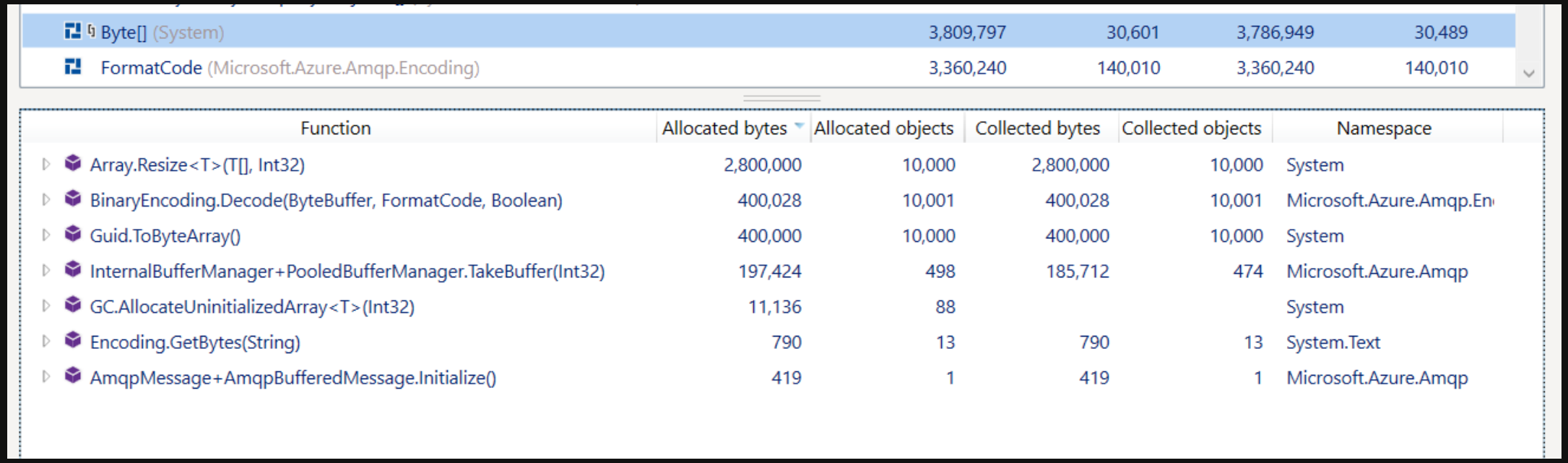


Byte[] (System)	7,750,697	50,700	7,726,929	50,579
ByteBuffer (Microsoft.Azure.Amqp)	4,635,520	72,430	4,635,520	72,430
AmqpSymbol (Microsoft.Azure.Amqp.Encoding)	4,558,016	142,438	4,557,888	142,434

Function	Allocated bytes	Allocated objects	Collected bytes	Collected objects	Namespace
Array.Resize<T>(T[], Int32)	5,600,000	20,000	5,600,000	20,000	System
AmqpMessageExtensions.GetByteArray(Data)	1,123,340	10,000	1,123,340	10,000	Azure.Messaging.ServiceE
BinaryEncoding.Decode(ByteBuffer, FormatCode, Boolean)	400,068	10,002	400,068	10,002	Microsoft.Azure.Amqp.En
Guid.ToByteArray()	400,000	10,000	400,000	10,000	System
InternalBufferManager+PooledBufferManager.TakeBuffer(Int32)	214,704	594	202,312	563	Microsoft.Azure.Amqp
GC.AllocateUninitializedArray<T>(Int32)	11,376	90			System
Encoding.GetBytes(String)	790	13	790	13	System.Text
AmqpMessage+AmqpBufferedMessage.Initialize()	419	1	419	1	Microsoft.Azure.Amqp

Getting lower on the stack

MEMORY CHARACTERISTICS



Byte[] (System)	3,809,797	30,601	3,786,949	30,489
FormatCode (Microsoft.Azure.Amqp.Encoding)	3,360,240	140,010	3,360,240	140,010

Function	Allocated bytes	Allocated objects	Collected bytes	Collected objects	Namespace
Array.Resize<T>(T[], Int32)	2,800,000	10,000	2,800,000	10,000	System
BinaryEncoding.Decode(ByteBuffer, FormatCode, Boolean)	400,028	10,001	400,028	10,001	Microsoft.Azure.Amqp.En
Guid.ToByteArray()	400,000	10,000	400,000	10,000	System
InternalBufferManager+PooledBufferManager.TakeBuffer(Int32)	197,424	498	185,712	474	Microsoft.Azure.Amqp
GC.AllocateUninitializedArray<T>(Int32)	11,136	88			System
Encoding.GetBytes(String)	790	13	790	13	System.Text
AmqpMessage+AmqpBufferedMessage.Initialize()	419	1	419	1	Microsoft.Azure.Amqp

Getting lower on the stack

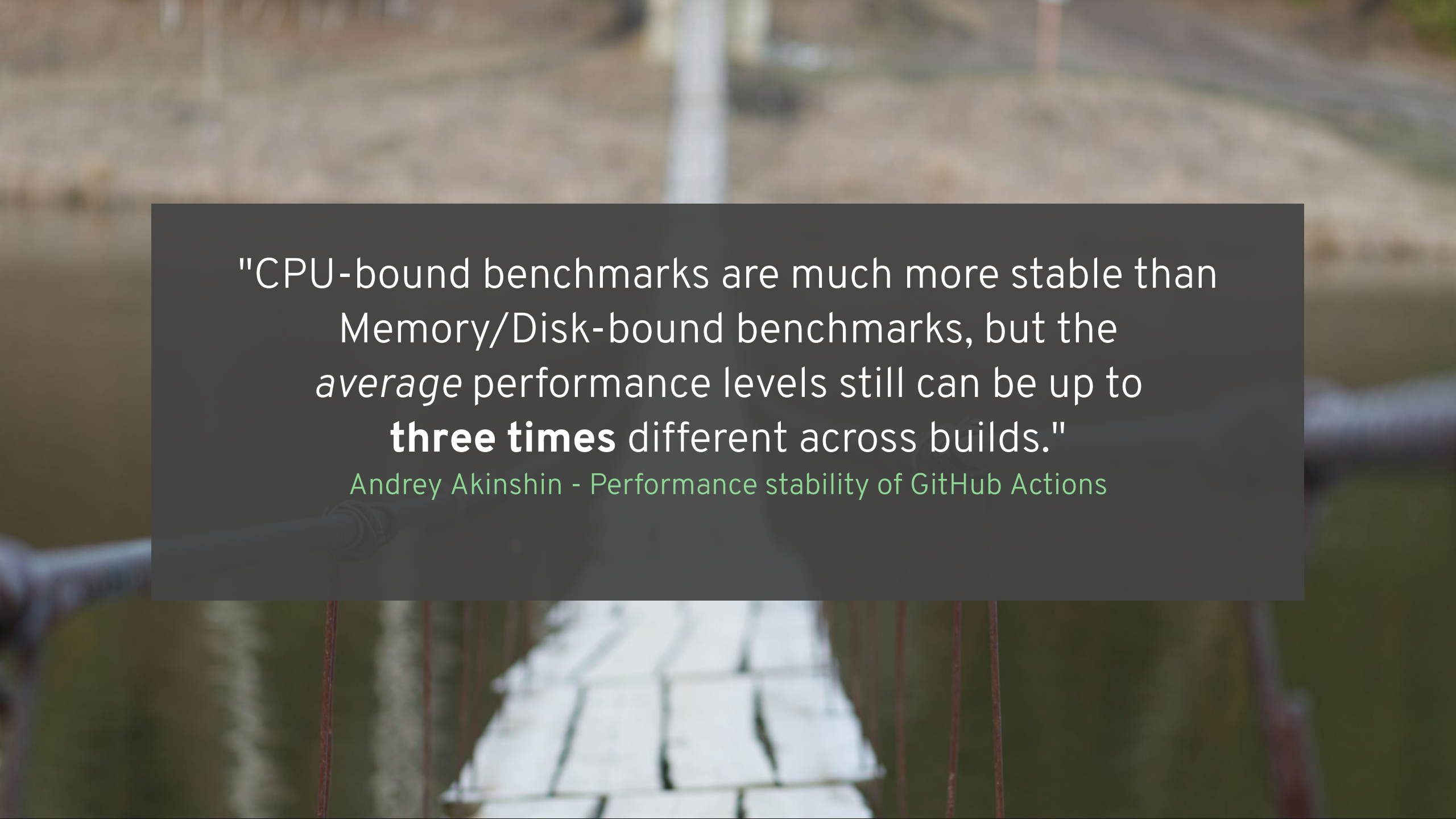
PREVENTING REGRESSIONS

- Guidance Preventing Regressions
- ResultComparer Tool

```
C:\Projects\performance\src\benchmarks\micro>
dotnet run -c Release -f net8.0 \
    --artifacts "C:\results\before"
```

```
C:\Projects\performance\src\benchmarks\micro>
dotnet run -c Release -f net8.0 \
    --artifacts "C:\results\after"
```

```
C:\Projects\performance\src\tools\ResultsComparer>
dotnet run --base "C:\results\before"
--diff "C:\results\after" --threshold 2%
```



"CPU-bound benchmarks are much more stable than Memory/Disk-bound benchmarks, but the *average* performance levels still can be up to **three times** different across builds."

Andrey Akinshin - Performance stability of GitHub Actions

BEYOND SIMPLE BENCHMARKS

A PRACTICAL GUIDE TO OPTIMIZING CODE

danielmarbach | daniel.marbach@particular.net | Daniel Marbach

- Use the performance loop to improve your code where it matters
- Combine it with profiling to observe how the small changes add up
- Optimize until you hit a diminishing point of return
- You'll learn a ton about potential improvements for a new design

github.com/danielmarbach/BeyondSimpleBenchmarks