DAILY PERFORMANCE PITFALLS

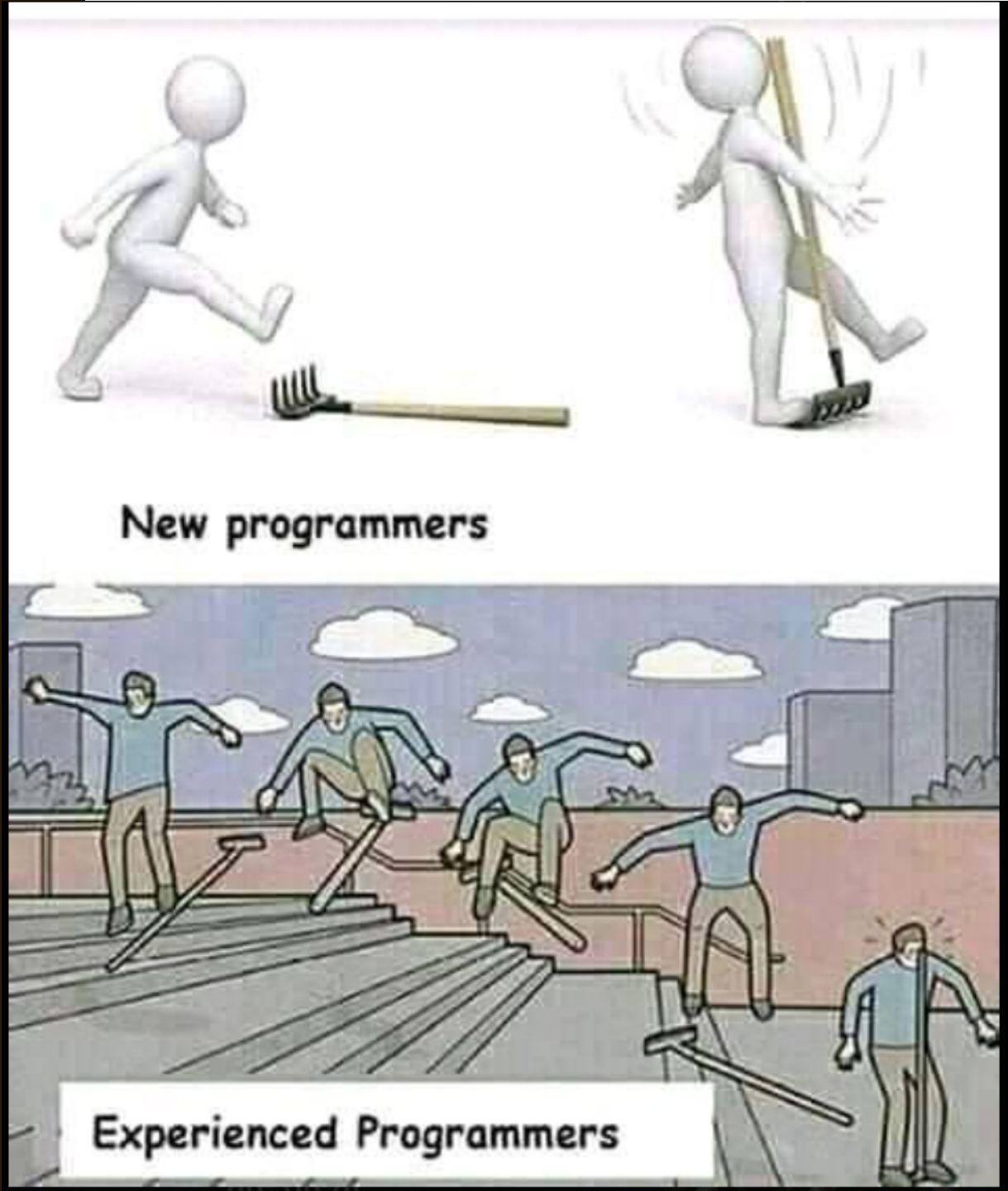
ŁUKASZ PYRZYK

ŁUKASZ PYRZYK

- Works remotely from Wroclaw
- Co-founder of Dotnetos
- Senior Full Stack Cloud Developer at Sonova/JCommerce
- Tweets as @lukaszpyrzyk

AGENDA

- Assuming performance improvement by looking at code which may look faster
- Decompiling our code with Sharplab.io to understand how optimizer can predict our ideas
- Story of the releasing product in debug mode
- Relying on the debug code behaviours and object state
- Surprising behaviours of tiered compilation
- Steaming network data with ASP.NET
- Optimizing code with ArrayPool and ETW events
- Simple, but valuable optimization of the CosmosDB which come from documentation
- Pitfalls of optimizing code without taking a wider perspective on the problem



I am Programmer,I have no life.
· 2 listopada ·

16 tys.

425 komentarzy
3 tys. udostępnień

Lubię to! Komentarz Udostępnij

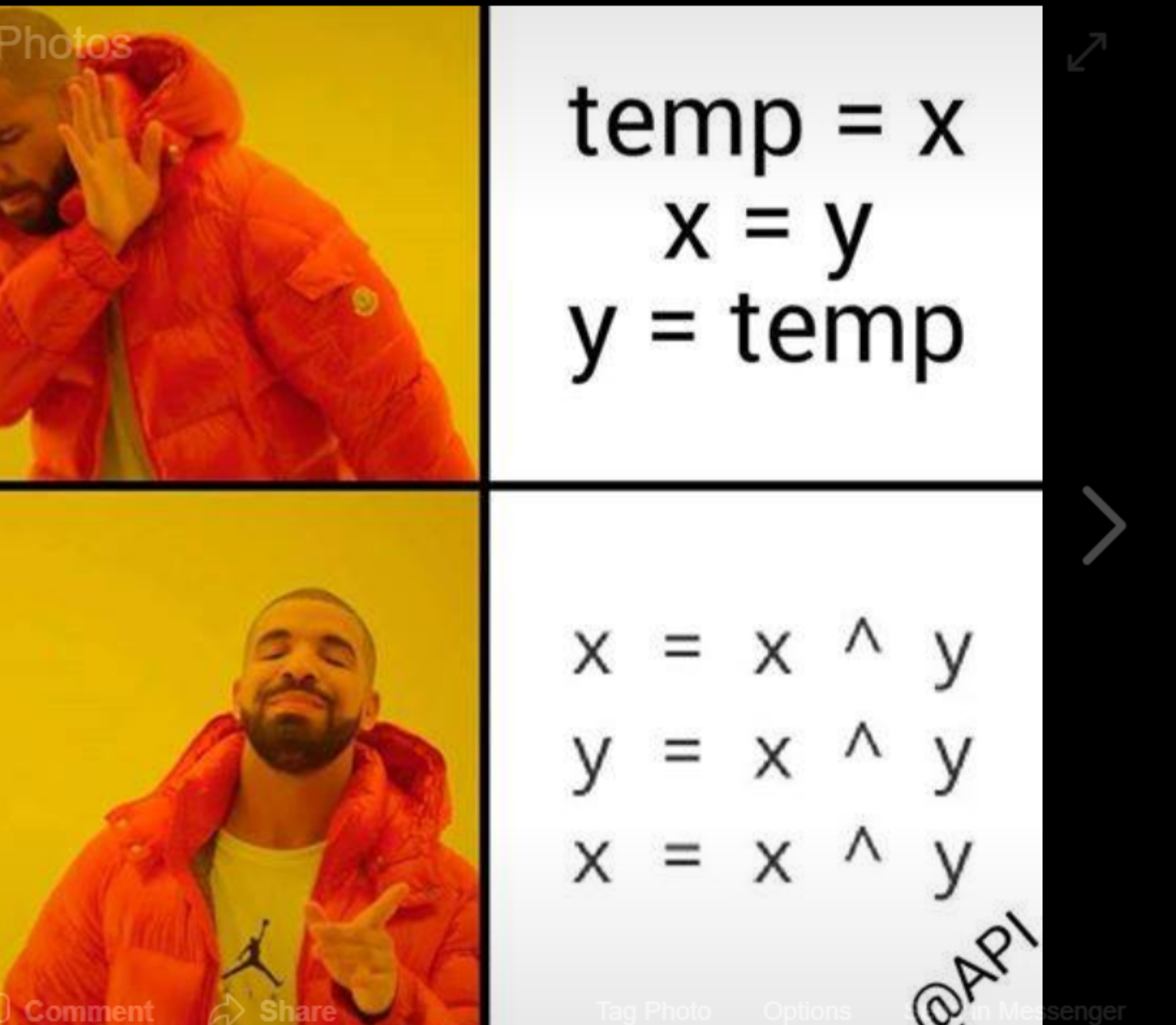
Najtrafniejsze

Lider wśród fanów
Younouss Porédaka Jalloh I have badge too.. 💪🤣🤣
Lubię to! · Odpowiedz · 4 d
8

Lider wśród fanów
Harsh Bangera Thanks 😊
Lubię to! · Odpowiedz · 4 d

Lider wśród fanów
Dino De Los Reyes Even seasoned

Napisz komentarz...



I am Programmer,I have no life.
Page Liked · October 14 ·

With Neel Beniwal, Ashira Dilshan, Ashira Dilsh
Rahul Kumar and Salmaan Bhati.

752 52 Comments 58

Like Comment Share

New

Neetesh Verma Pragya Jain Tushar Sadawarte ever did this? 😌😌

Like · Reply · 1w · Edited

30 Replies · October 17 at 3:20 PM

Бадрал Эрдэнэбулган swap(a,b);

Like · Reply · 1w

ISSUE #1

ASSUMING PERFORMANCE IMPROVEMENT

```
public void XOR(ref int x, ref int y)
{
        x = x ^ y;
        y = x ^ y;
        x = x ^ y;
}
```

- **No temp int variable**
- int is 4 bytes, so it **saves 4 bytes**
- **CPU** has XOR operation, so it's 1:1 mapping, so it's should be fast

00007ffc`d7459170 **XOR**(Int32 ByRef, Int32 ByRef)

IL_0000: ldarg.1, IL_0001: ldarg.1

IL_0002: ldind.i4, IL_0003: ldarg.2

IL_0004: ldind.i4,

IL_0005: **xor**

IL_0006: stind.i4

00007ffc`d7459170 418b00 **mov** eax,dword ptr [r8]

00007ffc`d7459173 3102 **xor** dword ptr [rdx],eax

IL_0007: ldarg.2, IL_0008: ldarg.1

IL_0009: ldind.i4, IL_000a: ldarg.2

IL_000b: ldind.i4

IL_000c: **xor**

IL_000d: stind.i4

00007ffc`d7459175 8b02 **mov** eax,dword ptr [rdx]

00007ffc`d7459177 413100 **xor** dword ptr [r8],eax

IL_000e: ldarg.1, IL_000f: ldarg.1

IL_0010: ldind.i4, IL_0011: ldarg.2

IL_0012: ldind.i4

IL_0013: **xor**

IL_0014: stind.i4

00007ffc`d745917a 418b00 **mov** eax,dword ptr [r8]

00007ffc`d745917d 3102 **xor** dword ptr [rdx],eax

IL_0015: ret

00007ffc`d745917f c3 ret

```
00007ffc`d7459170 TempVariable(Int32 ByRef, Int32 ByRef)
IL_0000: ldarg.1
IL_0001: ldind.i4
IL_0002: stloc.0
00007ffc`d7459170 8b02          mov     eax,dword ptr [rdx]
IL_0003: ldarg.1
IL_0004: ldarg.2
IL_0005: ldind.i4
IL_0006: stind.i4
00007ffc`d7459172 418b08        mov     ecx,dword ptr [r8]
00007ffc`d7459175 890a          mov     dword ptr [rdx],ecx
IL_0007: ldarg.2
IL_0008: ldloc.0
IL_0009: stind.i4
00007ffc`d7459177 418900        mov     dword ptr [r8],eax
IL_000a: ret
00007ffc`d745917a c3            ret
```

XOR

```
00007ff9`b7092140 418b00    mov     eax,dword ptr [r8]
00007ff9`b7092143 3102      xor     dword ptr [rdx],eax
00007ff9`b7092145 8b02      mov     eax,dword ptr [rdx]
00007ff9`b7092147 413100    xor     dword ptr [r8],eax
00007ff9`b709214a 418b00    mov     eax,dword ptr [r8]
00007ff9`b709214d 3102      xor     dword ptr [rdx],eax
00007ff9`b709214f c3        ret
```

TempVariable

```
00007ff9`b70b2140 8b02      mov     eax,dword ptr [rdx]
00007ff9`b70b2142 418b08    mov     ecx,dword ptr [r8]
00007ff9`b70b2145 890a      mov     dword ptr [rdx],ecx
00007ff9`b70b2147 418900    mov     dword ptr [r8],eax
00007ff9`b70b214a c3        ret
```

Method	Mean	Error	StdDev	**Median**
XOR	2.8275 ns	0.0615 ns	0.0575 ns	**2.8419 ns**

BenchmarkDotNet=v0.11.5, OS=Windows 10.0.16299.1146 (1709/FallCreatorsUpdate/Redstone3)
Intel Core i5-7200U CPU 2.50GHz (Kaby Lake), 1 CPU, 4 logical and 2 physical cores
Frequency=2648437 Hz, Resolution=377.5812 ns, Timer=TSC

Method	Mean	Error	StdDev	**Median**
XOR	2.8275 ns	0.0615 ns	0.0575 ns	**2.8419 ns**
TempVariable	0.0114 ns	0.0131 ns	0.0123 ns	**0.0000 ns**

The method duration is indistinguishable from the empty method duration

Requires Benchmarkdotnet v0.11.2+

```
public void AddAndSubtract(ref int x, ref int y)
{
        x = x + y;
        y = x - y;
        x = x - y;
}
```

Method	Mean	Error	StdDev	**Median**
XOR	2.8275 ns	0.0615 ns	0.0575 ns	**2.8419 ns**
TempVariable	0.0114 ns	0.0131 ns	0.0123 ns	**0.0000 ns**
AddAndSubtract	2.8454 ns	0.0691 ns	0.0646 ns	**2.8798 ns**

BenchmarkDotNet=v0.11.5, OS=Windows 10.0.16299.1146 (1709/FallCreatorsUpdate/Redstone3)
Intel Core i5-7200U CPU 2.50GHz (Kaby Lake), 1 CPU, 4 logical and 2 physical cores
Frequency=2648437 Hz, Resolution=377.5812 ns, Timer=TSC

Method	Runtime	Mean	Error	StdDev	Median
TempVariable	Clr	0.0022 ns	0.0067 ns	0.0062 ns	**0.0000 ns**
AddAndSubtract	Clr	3.0518 ns	0.0767 ns	0.0641 ns	**3.0383 ns**
XOR	Clr	2.9713 ns	0.0636 ns	0.0594 ns	**2.9579 ns**
TempVariable	Core	0.0199 ns	0.0264 ns	0.0418 ns	**0.0000 ns**
AddAndSubtract	Core	3.0355 ns	0.0953 ns	0.1135 ns	**3.0258 ns**
XOR	Core	2.8102 ns	0.0778 ns	0.0728 ns	**2.7733 ns**
TempVariable	Mono	0.4762 ns	0.0937 ns	0.2704 ns	**0.3619 ns**
AddAndSubtract	Mono	2.9212 ns	0.1042 ns	0.1115 ns	**2.8869 ns**
XOR	Mono	2.8912 ns	0.0686 ns	0.0573 ns	**2.8840 ns**


GOING
HOME

```
public void XOR()
{
        int x = 4096, y = 8192;
        x = x ^ y;
        y = x ^ y;
        x = x ^ y;
        Console.WriteLine(x);
        Console.WriteLine(y);
}
```

```
public void Temp()
{
        int x = 4096, y = 8192;
        int temp = x;
        x = y;
        y = temp;
        Console.WriteLine(x);
        Console.WriteLine(y);
}
```

BENCHMARKING....

IT IS THE SAME!

Code C# Default (2.9.0)

```
using System;
public class C {
    public void M() {
        int x = 4096;
        int y = 8192;
        x = x ^ y;
        y = x ^ y;
        x = x ^ y;
        Console.WriteLine(x);
        Console.WriteLine(y);
    }
}
```

Debug assembly 30 lines

Results JIT Asm Debug

```
; Desktop CLR v4.7.2671.00 (clr.dll) on x86.

C..ctor()
    L0000: push ebp
    L0001: mov ebp, esp
    L0003: push eax
    L0004: mov [ebp-0x4], ecx
    L0007: cmp dword [0x41131578], 0x0
    L000e: jz L0015
    L0010: call 0x73783a00
    L0015: mov ecx, [ebp-0x4]
    L0018: call System.Object..ctor()
    L001d: nop
    L001e: nop
    L001f: mov esp, ebp
    L0021: pop ebp
    L0022: ret

C.M()
    L0000: push ebp
    L0001: mov ebp, esp
    L0003: sub esp, 0xc
    L0006: mov [ebp-0x4], ecx
    L0009: cmp dword [0x41131578], 0x0
    L0010: jz L0017
    L0012: call 0x73783a00
    L0017: xor edx, edx
    L0019: mov [ebp-0x8], edx
    L001c: xor edx, edx
    L001e: mov [ebp-0xc], edx
    L0021: nop
    L0022: mov dword [ebp-0x8], 0x1000
    L0029: mov dword [ebp-0xc], 0x2000
    L0030: mov eax, [ebp-0xc]
    L0033: xor [ebp-0x8], eax
    L0036: mov eax, [ebp-0x8]
    L0039: xor [ebp-0xc], eax
    L003c: mov eax, [ebp-0xc]
    L003f: xor [ebp-0x8], eax
    L0042: mov ecx, [ebp-0x8]
    L0045: call System.Console.WriteLine(Int32)
    L004a: nop
    L004b: mov ecx, [ebp-0xc]
    L004e: call System.Console.WriteLine(Int32)
    L0053: nop
    L0054: nop
    L0055: mov esp, ebp
    L0057: pop ebp
    L0058: ret
```

Code C# Default (2.9.0)

```
using System;
public class C {
    public void M() {
        int x = 4096;
        int y = 8192;
        x = x ^ y;
        y = x ^ y;
        x = x ^ y;
        Console.WriteLine(x);
        Console.WriteLine(y);
    }
}
```

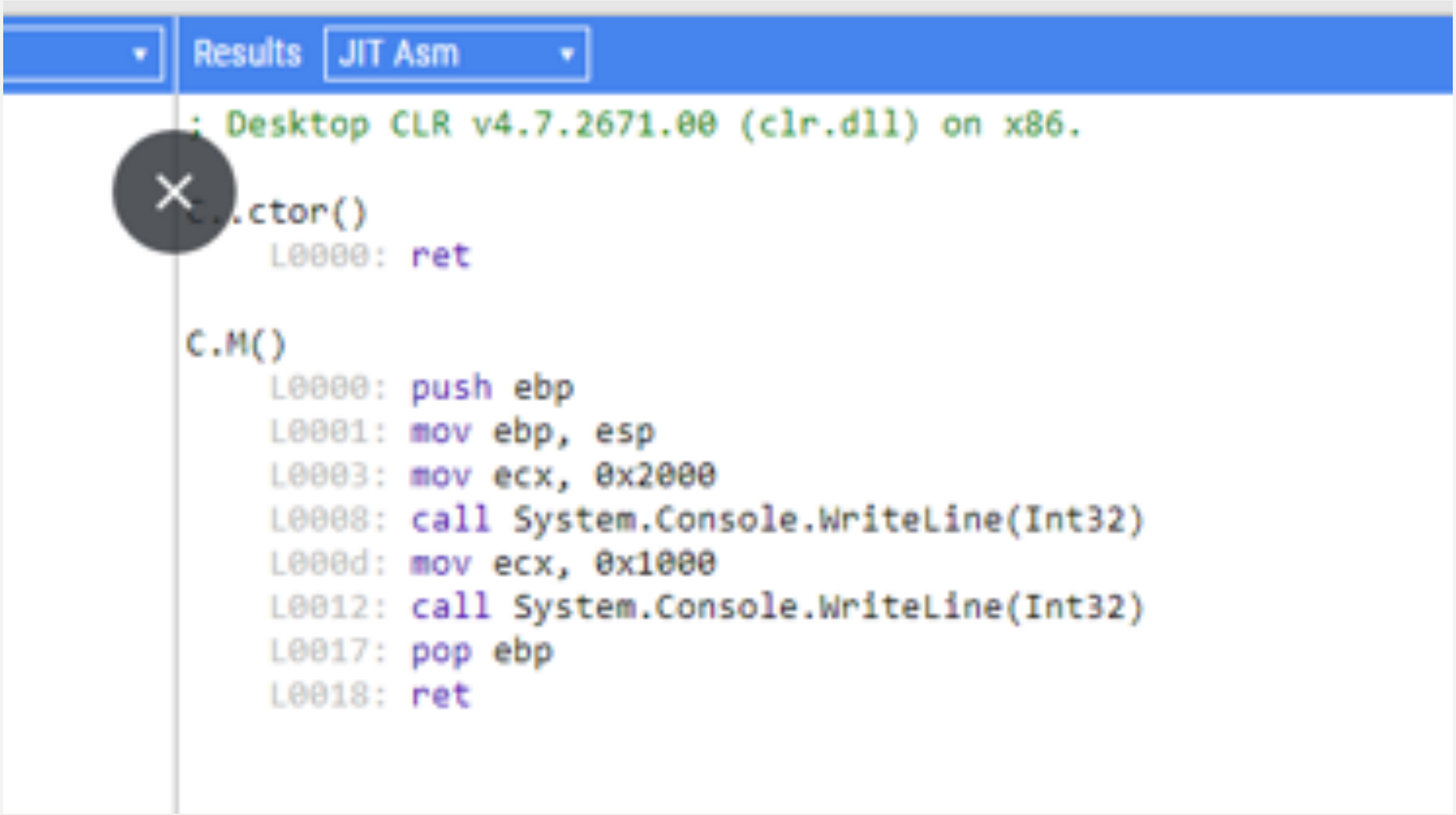
Release

assembly

8 lines

Results JIT Asm Release

```
; Desktop CLR v4.7.2671.00 (clr.dll) on x86.

C..ctor()
    L0000: ret

C.M()
    L0000: push ebp
    L0001: mov ebp, esp
    L0003: mov ecx, 0x2000
    L0008: call System.Console.WriteLine(Int32)
    L000d: mov ecx, 0x1000
    L0012: call System.Console.WriteLine(Int32)
    L0017: pop ebp
    L0018: ret
```


Results
JIT Asm
; Desktop CLR v4.7.2671.00 (clr.dll) on x86.
C..ctor()
L0000: ret
C.M()
L0000: push ebp
L0001: mov ebp, esp
L0003: mov ecx, 0x2000
L0008: call System.Console.WriteLine(Int32)
L000d: mov ecx, 0x1000
L0012: call System.Console.WriteLine(Int32)
L0017: pop ebp
L0018: ret

ISSUE #2

RELEASING PRODUCT IN DEBUG MODE


NO WAY. SORRY.
NOT GONNA HAPPEN!

YAML

YAMLDOTNET

PREVIOUS YAMLDOTNET BENCHMARKS

```
foreach(var test in tests)
{
   Console.Write("{0}\t{1}\t", adapterName, test.GetType().Name);
   var graph = test.Graph;
   RunTest(serializer, graph); // warmup
   if (!Stopwatch.IsHighResolution)
      Console.Error.WriteLine("Stopwatch is not high resolution!");
   var timer = Stopwatch.StartNew();
   for (var i = 0; i < iterations; ++i)
      RunTest(serializer, graph);
   var duration = timer.Elapsed;
   Console.WriteLine("{0}", duration.TotalMilliseconds / iterations);
}
```

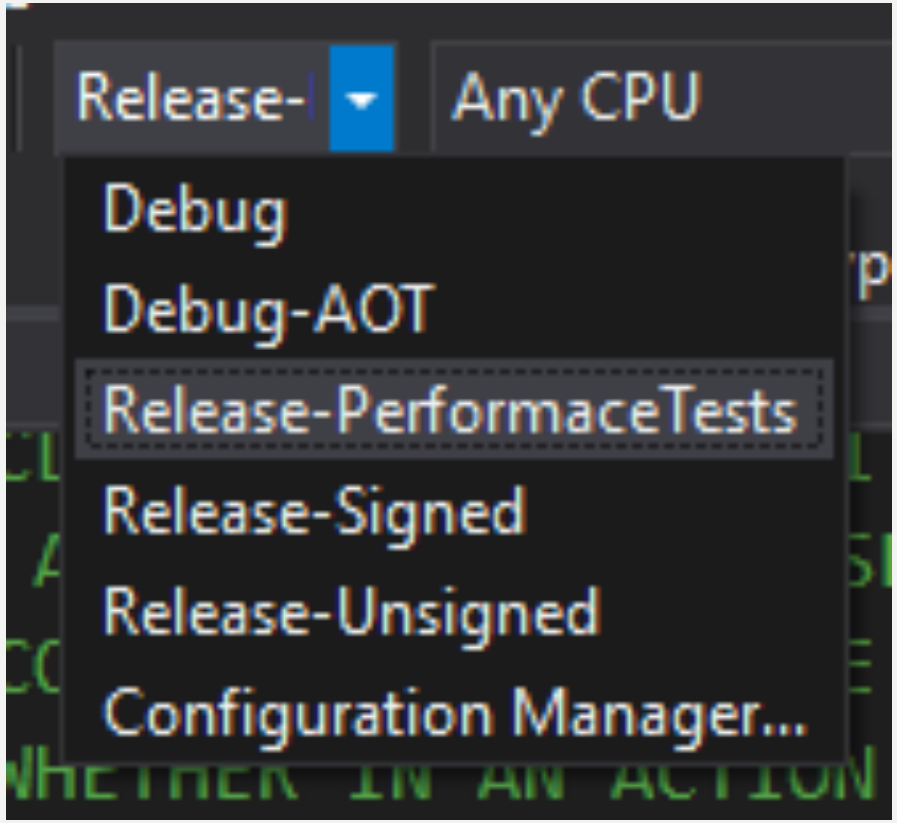
ATTEMP WITH BENCHMARKDOTNET

```
[MemoryDiagnoser]
public class ReceiptTest
{
    private readonly Receipt _receipt = new Receipt();
    private readonly StringWriter _buffer = new StringWriter();
    private readonly ISerializer _serializer = new SerializerBuilder()
        .WithNamingConvention(new CamelCaseNamingConvention())
        .Build();
    [Benchmark]
    public void Serialize()
    {
        _serializer.Serialize(_buffer, _receipt.Graph);
    }
}
```

dotnet run –c Release

```
// Validating benchmarks:
Assembly YamlDotNet.PerformanceTests.vlatest which defines benchmarks references non-optimized YamlDotNet
- If you own this dependency, please, build it in RELEASE.
- If you don't, you can create custom config with DontFailOnError to disable our custom policy and allow this benchmark to run.
```

PROBLEM



New csproj format doesn't have the same behaviour as the old one. Configurations like **Release-*** don't inherit from **Release** configuration.

FIX

```
<PropertyGroup Condition="'$(Configuration)' == 'Release-Signed' Or
'$(Configuration)' == 'Release-Unsigned' ">
    <DefineConstants>$(DefineConstants);RELEASE;TRACE</DefineConstants>
    <DebugSymbols>false</DebugSymbols>
    <DebugType>portable</DebugType>
    <Optimize>true</Optimize>
</PropertyGroup>
```

FIX

	Mean	Error	StdDev	Gen0	Gen1	Allocated
v1.2.1	128.7 us	1.285 us	1.202 us	5.8594	0.2441	23.66 KB
v2.2.0	240.6 us	3.467 us	3.243 us	18.0664	0.4883	60.03 KB
v2.3.0	307.9 us	6.112 us	10.21 us	20.0195	0.4883	67.32 KB
v3.8.0	292.2 us	4.225 us	3.952 us	21.4844	0.4883	70.82 KB
v4.0.0	283.2 us	3.075 us	2.876 us	22.9492	0.4883	74.26 KB
v5.2.1	**539.5 us**	5.710 us	5.062 us	8.7891	0.9766	30.82 KB
vlatest	**145.8 us**	1.671 us	1.563 us	8.3008	0.4883	30.7 KB

Wojciech Nagórski
@WojtekNagorski

Follow

Thanks to #BenchmarkDotNet I've improved performance of YamlDotNet by 370%.
There is an article about this story:
wojciechnagorski.com/2018/12/how-i- ...
Big thanks to @antoineaubry for the review.

5:37 AM - 20 Dec 2018

14 Retweets **45** Likes

ISSUE #3

BUFFERING RESPONSES

BUFFERING A FILE

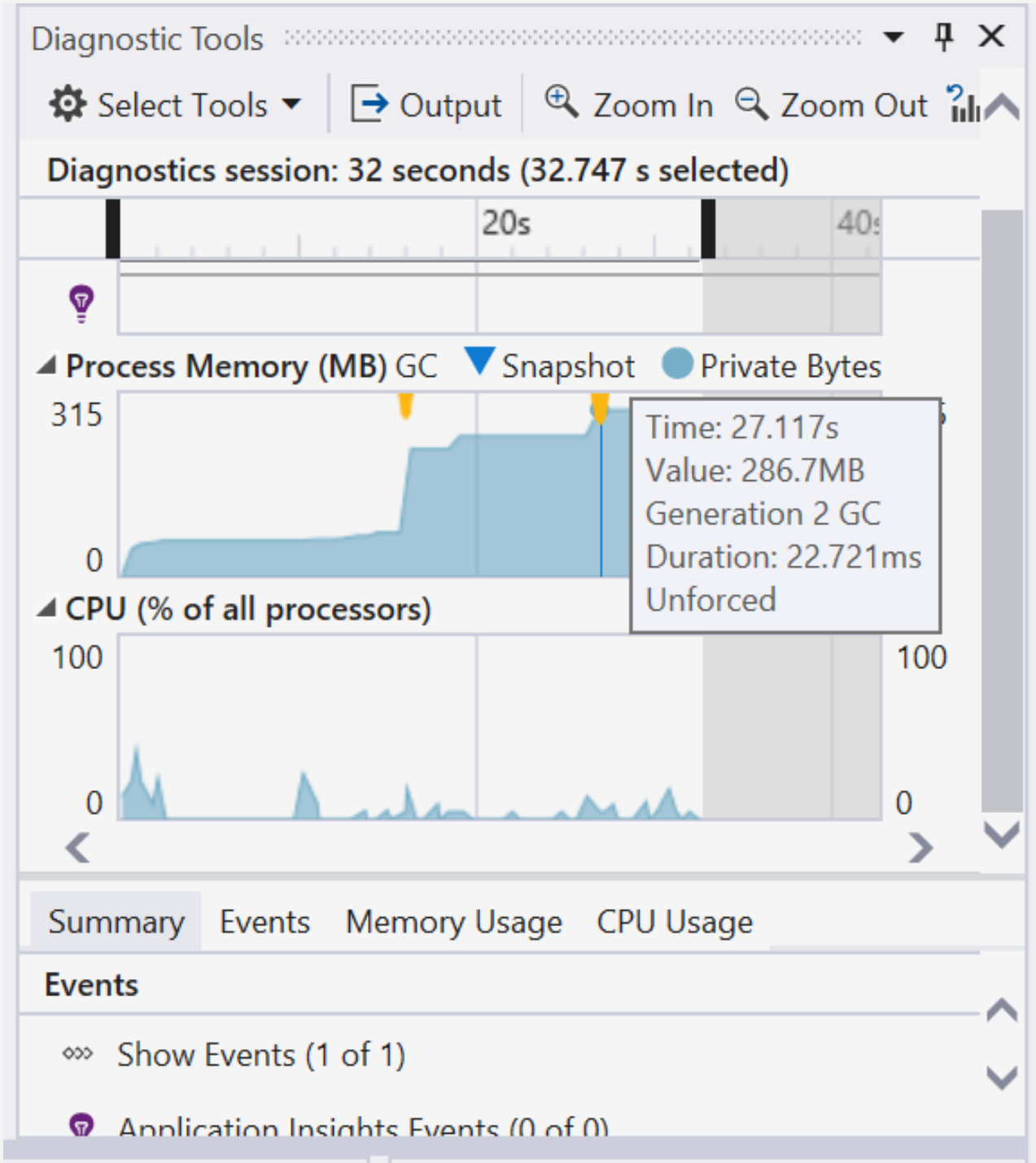
https://github.com/dotnet/corefx/archive/v2.2.0-preview3.zip

BUFFERING A FILE

http://dotnet.microsoft.com/download/

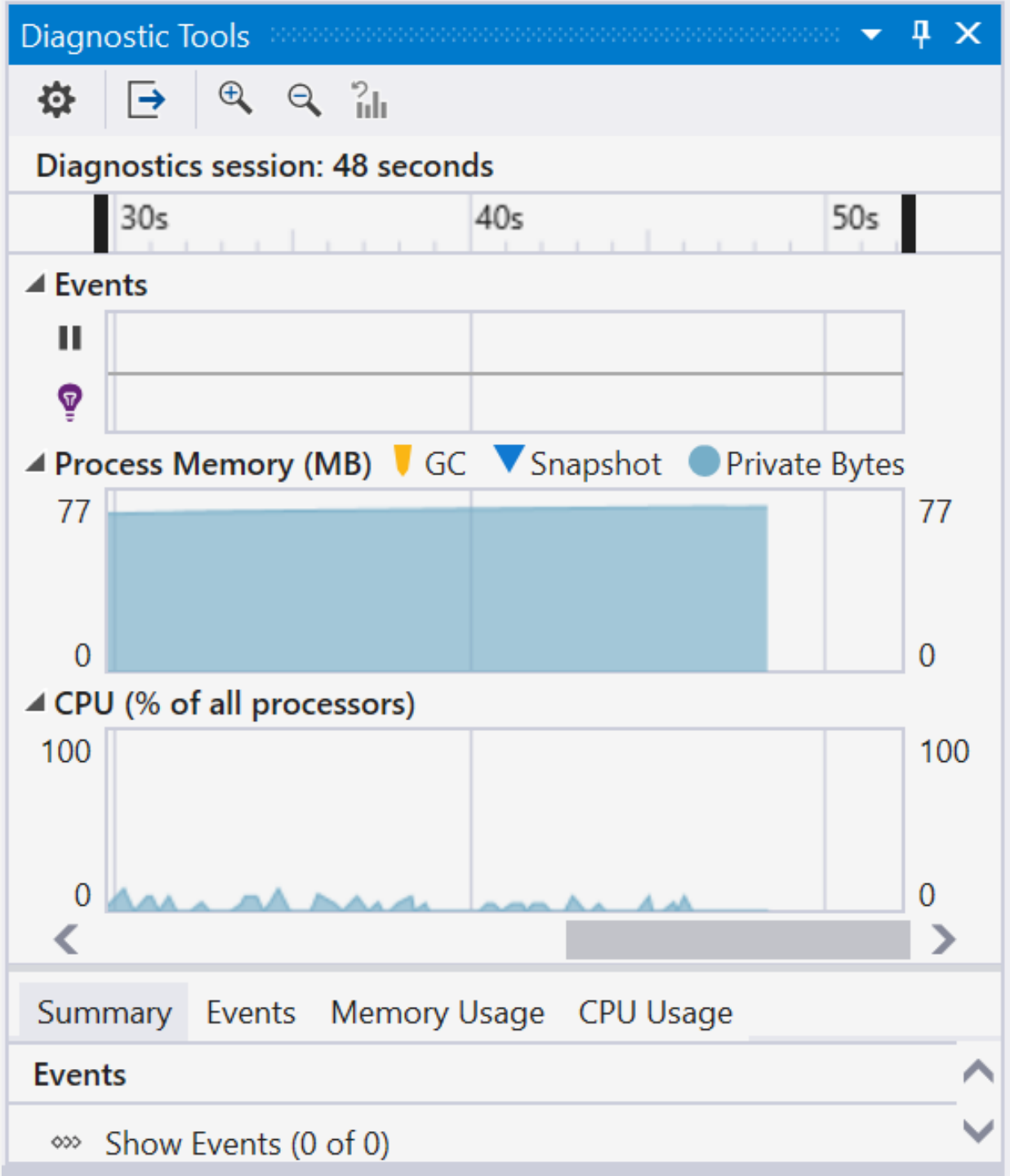
BUFFERING A FILE

```
public async Task<IActionResult> WithBuffering()
{
    var request = new HttpRequestMessage(HttpMethod.Get,
"dotnet/corefx/archive/v2.2.0-preview3.zip");

    var response = await _client.Client.SendAsync(request);
    var stream = await response.Content.ReadAsStreamAsync();
    var contentType = response.Content.Headers.ContentType.MediaType;
    return new FileStreamResult(stream, contentType);
}
```


Diagnostic Tools
Select Tools
Output
Zoom In
Zoom Out
Diagnostics session: 32 seconds (32.747 s selected)
20s
40s
Process Memory (MB) GC
Snapshot
Private Bytes
315
0
Time: 27.117s
Value: 286.7MB
Generation 2 GC
Duration: 22.721ms
Unforced
CPU (% of all processors)
100
0
100
0
Summary
Events
Memory Usage
CPU Usage
Events
Show Events (1 of 1)
Application Insights Events (0 of 0)

BUFFERING A FILE

```
public async Task<IActionResult> WithoutBuffering()
{
    var request = new HttpRequestMessage(HttpMethod.Get,
"dotnet/corefx/archive/v2.2.0-preview3.zip");

    var response = await _client.Client.SendAsync(request,
HttpCompletionOption.ResponseHeadersRead);
    var stream = await response.Content.ReadAsStreamAsync();
    var contentType = response.Content.Headers.ContentType.MediaType;
    return new FileStreamResult(stream, contentType);
}
```



BUFFERING A FILE

```
public async Task<Stream> Stream()
{
    var request = new HttpRequestMessage(HttpMethod.Get,
"dotnet/corefx/archive/v2.2.0-preview3.zip");

    var response = await _client.Client.GetStreamAsync(request.RequestUri);
    return response;
}
```

ISSUE #4

RELYING ON THE OBJECT STATE

CODE

```
static void Main(string[] args)
{
    var timer = new Timer(x => Console.WriteLine(DateTime.UtcNow), null, 0, 100);
    Console.WriteLine("Timer started?");
    GC.Collect();
    Console.Read();
}
```

DEBUG

```
dotnet run -c Debug
14/10/2019 18:43:22
Timer started?
14/10/2019 18:43:22
14/10/2019 18:43:22
14/10/2019 18:43:22
[…]
```

RELEASE

```
dotnet run -c Release
Timer started?
14/10/2019 18:48:56
```

EAGER ROOT COLLECTION

IT IS A JUST-IN-TIME COMPILER OPTIMIZATION WHICH MAKES LOCAL REFERENCES IRRELEVANT AFTER THEIR LAST USAGE

KEEP ALIVE

```
static void Main(string[] args)
{
    var timer = new Timer(x => Console.WriteLine(DateTime.UtcNow), null, 0, 100);
    Console.WriteLine("Timer started?");
    GC.Collect();
    Console.Read();
    GC.KeepAlive(timer);
}
```

CALLING DISPOSE

```
static void Main(string[] args)
{
    var timer = new Timer(x => Console.WriteLine(DateTime.UtcNow), null, 0, 100);
    Console.WriteLine("Timer started?");
    GC.Collect();
    Console.Read();
    timer.Dispose();
}
```

CALLING DISPOSE

```
static void Main(string[] args)
{
   using (var timer = new Timer(x => Console.WriteLine(DateTime.UtcNow), null, 0, 100))
   {
      Console.WriteLine("Timer started?");
      GC.Collect();
      Console.Read();
   }
}
```

CALLING DISPOSE

C# 8.0

CALLING DISPOSE

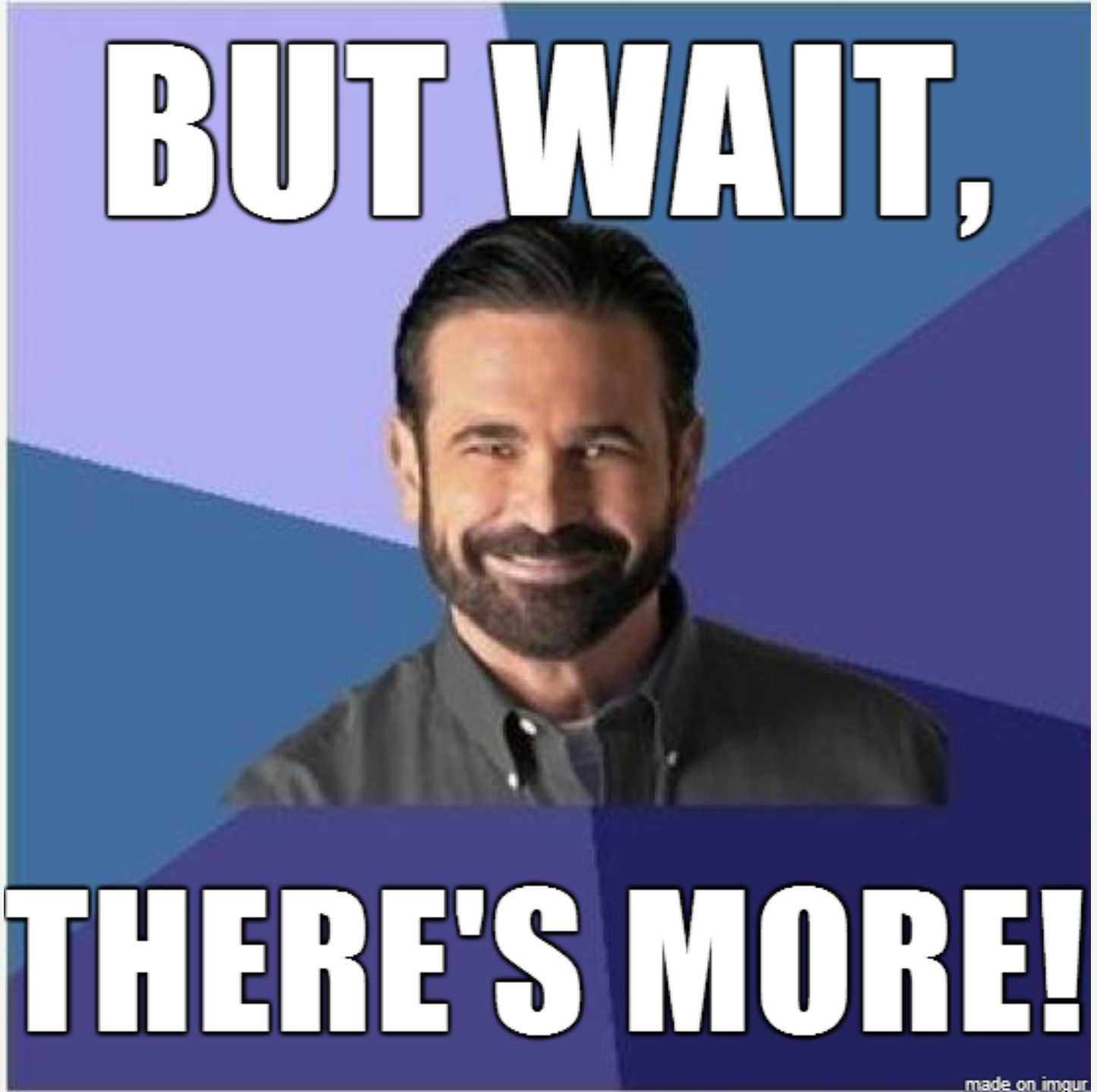
```
static void Main(string[] args)
{
   using var timer = new Timer(x => Console.WriteLine(DateTime.UtcNow), null, 0,
100);
   Console.WriteLine("Timer started?");
   GC.Collect();
   Console.Read();
}
```

CALLING DISPOSE

```
[NullableContext(1)]
[Nullable(0)]
public sealed class Timer : MarshalByRefObject, IAsyncDisposable, Idisposable
{
   /// <summary>Releases all resources used by the current instance of <see cref="T:System.Threading.Timer" />.</summary>
   /// <returns>A <see cref="T:System.Threading.Tasks.ValueTask" /> that completes when all work associated with the timer has ceased.</returns>
   public ValueTask DisposeAsync();
}
```

CALLING DISPOSE

```
static async Task Main(string[] args)
{
    await using var timer =
        new Timer(x => Console.WriteLine(DateTime.UtcNow), null, 0, 100);
    Console.WriteLine("Timer started?");
    GC.Collect();
    Console.Read();
}
```


BUT WAIT,
THERE'S MORE!
made on imgur

CODE

```
static void Main(string[] args)
{
    var timer = new Timer(x => Console.WriteLine(DateTime.UtcNow), null, 0, 100);
    Console.WriteLine("Timer started?");
    GC.Collect();
    Console.Read();
}
```

RELEASE

```
dotnet run -c Release
Timer started?
16/10/2019 16:36:51
16/10/2019 16:36:51
16/10/2019 16:36:51
16/10/2019 16:36:51
16/10/2019 16:36:51
16/10/2019 16:36:51
16/10/2019 16:36:51
[..]
```

WHAT?

ISSUE #4

FORGETTING ABOUT TIERED COMPILATION

.NET Core version	Tiered compilation
2.0	None
2.1	Optional
3.0	Default

RELEASE

```
<Project Sdk="Microsoft.NET.Sdk">
   <PropertyGroup>
      <OutputType>Exe</OutputType>
      <TargetFramework>netcoreapp2.1</TargetFramework>
      <TieredCompilation>true</TieredCompilation>
   </PropertyGroup>
</Project>
```

RELEASE

```
<Project Sdk="Microsoft.NET.Sdk">
   <PropertyGroup>
      <OutputType>Exe</OutputType>
      <TargetFramework>netcoreapp3.0</TargetFramework>
   </PropertyGroup>
</Project>
```


Pro .NET Memory Management
Pro .NET Memory Management
For Better Code, Performance, and Scalability
Konrad Kokosa
Apress
Pro .NET Memory Management
Apress

TIERED COMPILATION

CAN INFLUENCE THE PROFILING AND BENCHMARKING RESULTS.

IT CAN CHANGE RESULTS OF SAMPLES FROM BOOKS OR BLOG POSTS

If your intention is to optimize the method as quickly as possible, you can use attribute the method with MethodImpl(MethodImplOptions.AggressiveOptimization)], which skips tier 0.

However, in the future the attribute may not generate fully optimized code, but it may try to optimize method as quickly as possible.

EgorBo / Disasmo

Watch 19 | Unstar 220 | Fork 8

Code | Issues 5 | Pull requests 0 | Projects 0 | Wiki | Security | Insights

VS2019 Add-in. Click on any method or class to see what .NET Core's JIT generates for them (ASM).

asm disassembler netcore csharp visual-studio-extension

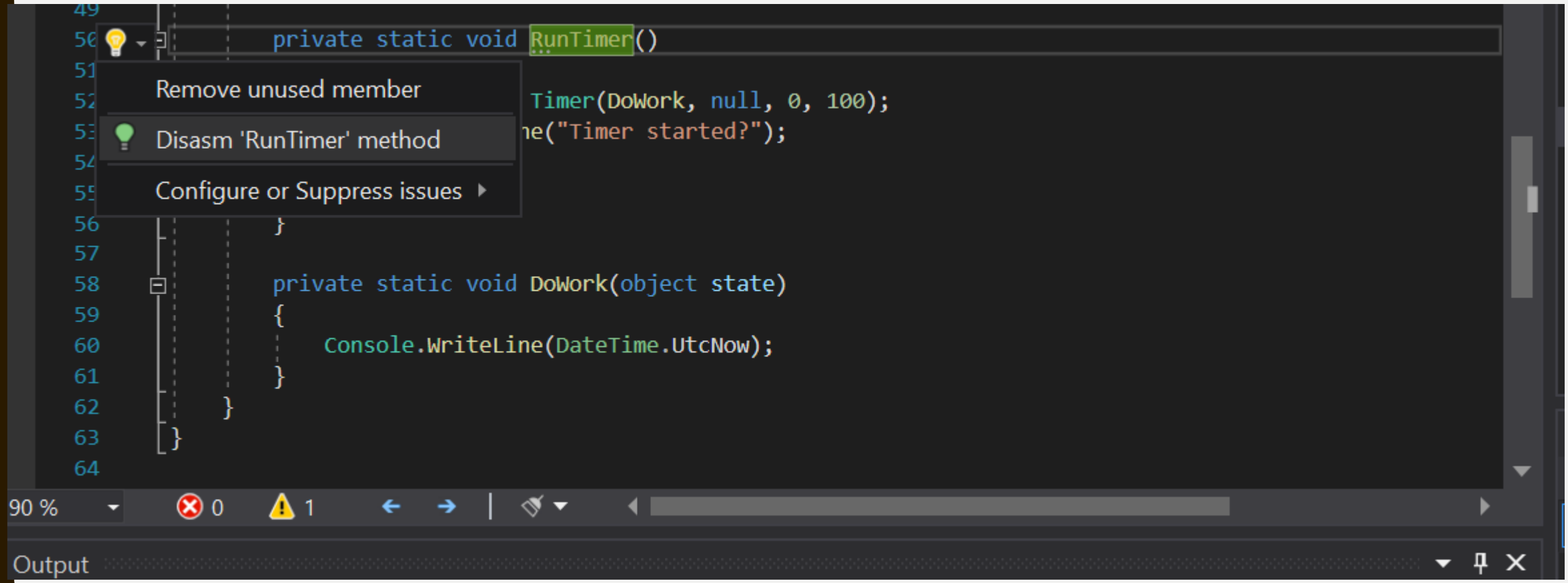
53 commits | 1 branch | 0 packages | 0 releases | 2 contributors | MIT

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

EgorBo Add RunOnLocalClrViewModel — Latest commit 41cc329 on 20 Jul

images	Update screenshot, get rid of `Microsoft.VisualStudio.MPF.15.0`	9 months ago
src	Add RunOnLocalClrViewModel	4 months ago
.gitignore	Initial commit	9 months ago
Disasmo.sln	Small refactoring	9 months ago
LICENSE	Initial commit	9 months ago
README.md	Update README.md	9 months ago

README.md


private static void RunTimer()
Remove unused member
Disasm 'RunTimer' method
Configure or Suppress issues
Timer(DoWork, null, 0, 100);
ne("Timer started?");
}
private static void DoWork(object state)
{
Console.WriteLine(DateTime.UtcNow);
}
}
}
90 %
0
1
Output

Disasmo

CoreCLR Path: C:\dev\coreclr\ ... | Refresh | Tiered JIT

Output | Previous output | Settings | S.R.Intrinsics | Tests

```
DPP.EarlyRootCollection.Program.RunTimer():
    mov     rcx,7FFEF5FDEE68h
    call    coreclr!coreclr_shutdown_2+0xe010
    mov     rdi,rax
    lea     rcx,[rdi+8]
    mov     rdx,rdi
    call    coreclr!coreclr_shutdown_2+0xd150
    mov     rcx,7FFEF5D9D070h
    mov     qword ptr [rdi+18h],rcx
    mov     rcx,7FFEF5ECF760h
    mov     qword ptr [rdi+20h],rcx
    mov     rcx,7FFEF5FDF0D0h
    call    coreclr!coreclr_shutdown_2+0xe010
    mov     rcx,rax
    mov     dword ptr [rsp+20h],64h
    mov     dword ptr [rsp+28h],1
    mov     rdx,rdi
    xor     r8d,r8d
    xor     r9d,r9d
    call    System.Threading.Timer.TimerSetup(System.Threading.TimerCallback, System.Object, UInt32, UInt32, Boolean)
    mov     rcx,1E29C3030D0h
    mov     rcx,qword ptr [rcx]
    call    System.Console.WriteLine(System.String)
    xor     r11d,r11d
    mov     ecx,0FFFFFFFFh
    mov     edx,2
    mov     rax,7FFEF5F60D00h
    mov     qword ptr [rbp-68h],rax
    lea     rax,[00007ffe`f5ee2f99]
    mov     qword ptr [rbp-50h],rax
    lea     rax,[rbp-78h]
    mov     qword ptr [rsi+10h],rax
    mov     byte ptr [rsi+0Ch],0
    call    qword ptr [00007ffe`f5f61218]
    mov     byte ptr [rsi+0Ch],1
    cmp     dword ptr [coreclr!g_CLREngineMetrics+0x8ff0
    je      M00_L00
    call    qword ptr [coreclr!GetCLRRuntimeHost+0x31bbd8
    M00_L00
    mov     rax,qword ptr [rbp-70h]
    mov     qword ptr [rsi+10h],rax
```

212 bytes of code

Disasmo

CoreCLR Path: C:\dev\coreclr\ ... | Refresh | Tiered JIT

Output | Previous output | Settings | S.R.Intrinsics | Tests

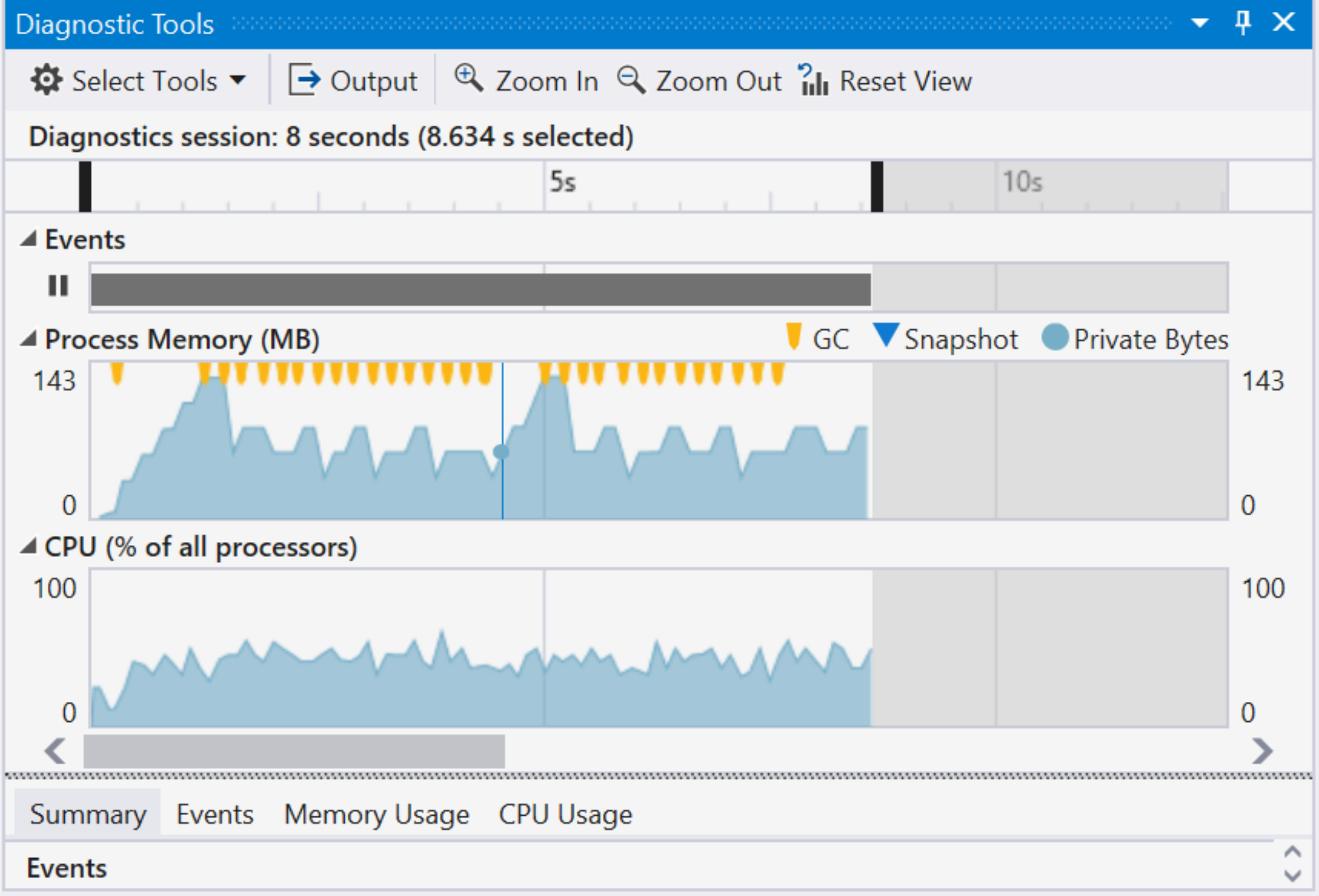
```
DPP.EarlyRootCollection.Program.RunTimer():
    mov     rcx,7FFEF5FFFE38h
    call    coreclr!coreclr_shutdown_2+0xe010
    mov     qword ptr [rbp-8],rax
    mov     r8,7FFEF5EE1750h
    mov     rcx,qword ptr [rbp-8]
    xor     edx,edx
    mov     r9,7FFEF5DAD070h
    call    System.MulticastDelegate.CtorOpened(System.Object, IntPtr, IntPtr)
    mov     rcx,7FFEF60101B0h
    call    coreclr!coreclr_shutdown_2+0xe010
    mov     qword ptr [rbp-10h],rax
    mov     dword ptr [rsp+20h],64h
    mov     rcx,qword ptr [rbp-10h]
    mov     rdx,qword ptr [rbp-8]
    xor     r8d,r8d
    xor     r9d,r9d
    call    System.Threading.Timer..ctor(System.Threading.TimerCallback, System.Object, Int32, Int32)
    mov     rcx,22BCC2730D0h
    mov     rcx,qword ptr [rcx]
    call    System.Console.WriteLine(System.String)
    call    System.GC.Collect()
    call    System.Console.Read()
    nop
; Total bytes of code 125


System.Threading.Timer..ctor(System.Threading.TimerCallback, System.Object, Int32, Int32):
; Total bytes of code 0


System.GC.Collect():
; Total bytes of code 0
```

125 bytes of code

ISSUE #5

NOT USING ETW EVENTS

```
private static async Task Default()
{
    var buffer = new byte[8 * 1024];
    for (int i = 0; i < Iterations; i++)
    {
        foreach (var file in Directory.EnumerateFiles("images"))
        {
            var fileInfo = new FileInfo(file);
            var array = new byte[fileInfo.Length];
            await using var stream = File.OpenRead(file);
            await ReadStreamToArray(fileInfo, stream, buffer, array);
        }
    }
}
```


Diagnostic Tools
Select Tools
Output
Zoom In
Zoom Out
Reset View
Diagnostics session: 8 seconds (8.634 s selected)
5s
10s
Events
Process Memory (MB)
GC
Snapshot
Private Bytes
143
0
143
0
CPU (% of all processors)
100
0
100
0
Summary
Events
Memory Usage
CPU Usage
Events

```
CommandLine:
"C:\dev\DPF\Src\DPF.ArrayPool\bin\Release\netcoreapp3.0\DPF.Arra
yPool.exe"
Runtime Version: V 4.0.30319.0 (built on 13/09/2019 01:02:04)
CLR Startup Flags: None
Total CPU Time: 11,368 msec
Total GC CPU Time: 19 msec
Total Allocs : 1,263.242 MB
GC CPU MSec/MB Alloc : 0.015 MSec/MB
Total GC Pause: 20.5 msec
% Time paused for Garbage Collection: 0.3%
% CPU Time spent Garbage Collecting: 0.2%
Max GC Heap Size: 145.235 MB
Peak Process Working Set: 170.570 MB
Peak Virtual Memory Usage: 2,199,714.988 MB
```

ARRAYPOOL<BYTE>.SHARED

ArrayPool<T> Class

Namespace: System.Buffers

Assemblies: System.Buffers.dll, netstandard.dll

Provides a resource pool that enables reusing instances of type T[].

C# Copy

```
public abstract class ArrayPool<T>
```

Type Parameters

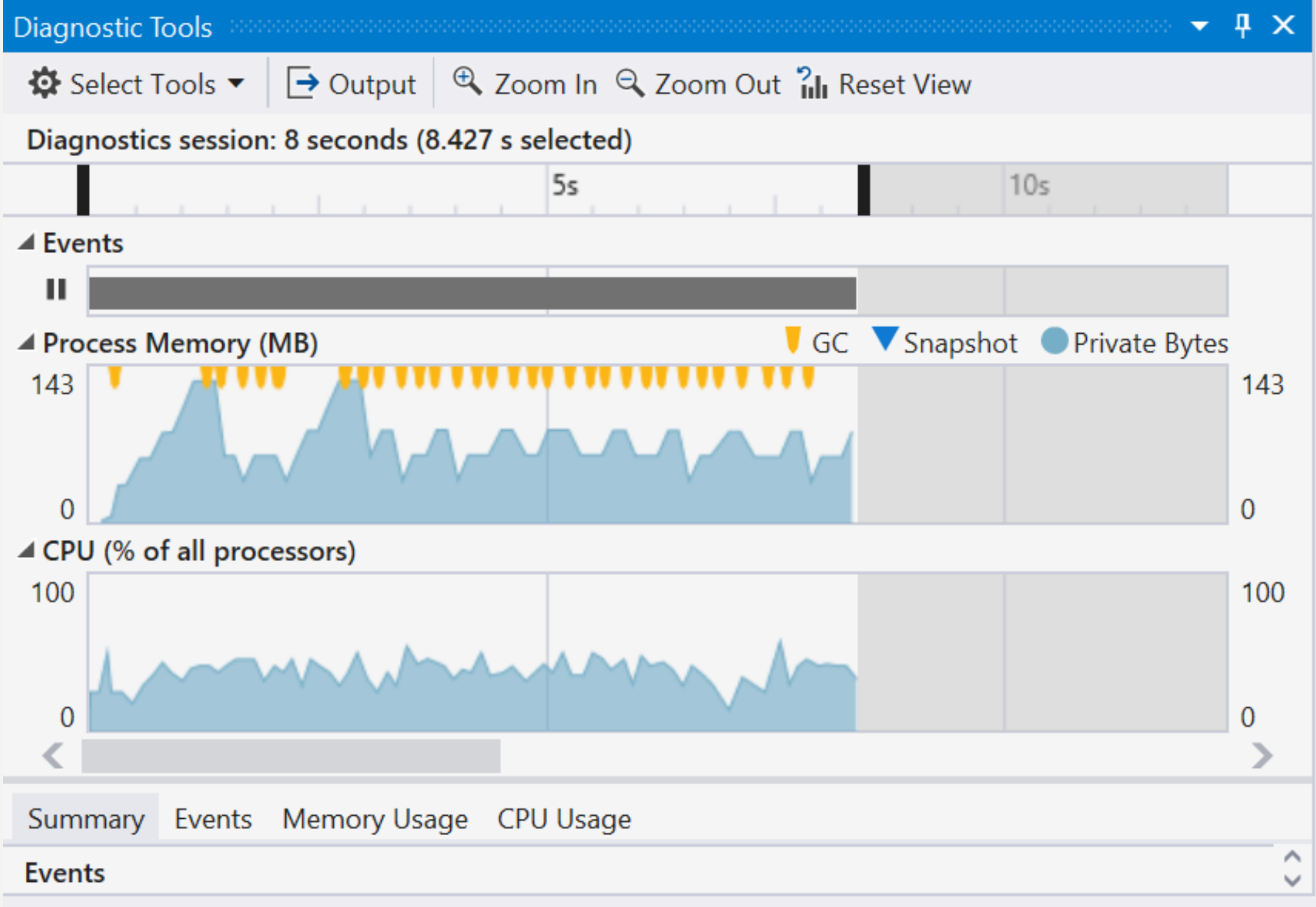
T

The type of the objects that are in the resource pool.

Inheritance Object → ArrayPool<T>

Remarks

```
var arrayPool = ArrayPool<byte>.Shared; var buffer = arrayPool.Rent(8 * 1024);
try
{
    for (int i = 0; i < Iterations; i++)
    {
        foreach (var file in Directory.EnumerateFiles("images"))
        {
            var fileInfo = new FileInfo(file);
            var array = arrayPool.Rent(Convert.ToInt32(fileInfo.Length));
            try
            {
                    await using var stream = File.OpenRead(file);
                    await ReadStreamToArray(fileInfo, stream, buffer, array);
            }
            finally
            {
                arrayPool.Return(array);
            }
        }
    }
}
finally
{
    arrayPool.Return(buffer);
}
```


Diagnostic Tools
Select Tools
Output
Zoom In
Zoom Out
Reset View
Diagnostics session: 8 seconds (8.427 s selected)
5s
10s
Events
Process Memory (MB)
GC
Snapshot
Private Bytes
143
0
143
0
CPU (% of all processors)
100
0
100
0
Summary
Events
Memory Usage
CPU Usage
Events

```
CommandLine:
"C:\dev\DPF\Src\DPF.ArrayPool\bin\Release\netcoreapp3.0\DPF.Arra
yPool.exe"
Runtime Version: V 4.0.30319.0 (built on 13/09/2019 01:02:04)
CLR Startup Flags: None
Total CPU Time: 12,527 msec
Total GC CPU Time: 12 msec
Total Allocs : 1,014.148 MB
GC CPU MSec/MB Alloc : 0.012 MSec/MB
Total GC Pause: 25.1 msec
% Time paused for Garbage Collection: 0.3%
% CPU Time spent Garbage Collecting: 0.1%
Max GC Heap Size: 146.111 MB
Peak Process Working Set: 158.175 MB
Peak Virtual Memory Usage: 2,199,715.676 MB
```

Remarks

Using the ArrayPool<T> class to rent and return buffers (using the Rent and Return methods) **can improve performance** in situations where arrays are created and destroyed frequently, resulting in significant memory pressure on the garbage collector.

Collecting ETW Data while running a command

This dialog give displays options for collecting ETW profile data. The only required field the 'Command' field and this is only necessary when using the 'Run' command.

If you wish to analyze on another machine use the Zip option when collecting data. See Collecting ETW Profile Data. for more.

Command: dotnet run -c Release

Data File: C:\dev\DPF\Src\DPF.ArrayPool\sharedArrayPool.etl ...

Current Dir: C:\dev\DPF\Src\DPF.ArrayPool

Zip: ☐ Circular MB: 0 Merge: ■ Thread Time: ☐ Mark Text: Mark 1 Mark Run Command Log Cancel

Status: Enter a command to run.

Advanced Options

Kernel Base: ☑	Cpu Samples: ☑	Page Faults: ☐	File I/O: ☐	Registry: ☐	VirtAlloc: ☐	MemInfo: ☐
Handle: ☐	RefSet: ☐	IIS: ☐	NetMon: ☐	Net Capture: ☐	Tasks (TPL): ☑	
.NET : ☑	.NET Stress: ☐	Background JIT: ☐	.NET Calls: ☐	JIT Inlining: ☐	NET Native CCW: ☐	
GC Collect Only: ☐	GC Only: ☐	.NET Alloc: ☐	.NET SampAlloc: ☐	ETW .NET Alloc: ☐	Dump Heap: ☐	

Additional Providers: *System.Buffers.ArrayPoolEventSource Provider Browser ?

CPU Sample Interval Msec 1 Cpu Ctrs Ctrs OS Heap Exe OS Heap Process

.NET Symbol Collection ☐ No V3.X NGEN Symbols ☑ Symbol TimeOut 120

Max Collect Sec Stop Trigger

Events sharedArrayPool.etl.zip in DPF.ArrayPool (C:\dev\DPF\Src\DPF.ArrayPool\sharedArrayPool.etl.zip)

File Help | Event View Help (F1) | Troubleshooting | Tips

Update | Start: 0.000 | End: 17,918.037 | MaxRet: 10000 | Find:

Process Filter: | Text Filter: | Columns To Display: Cols

Event Types | Filter: buf | Histogram: _____1___5_4___11441511111_1121221_21A1122111121121112121121_____

- System.Buffers.ArrayPoolEventSource/BufferAllocated
- System.Buffers.ArrayPoolEventSource/BufferRented
- System.Buffers.ArrayPoolEventSource/BufferReturned
- System.Buffers.ArrayPoolEventSource/BufferTrimPoll
- System.Buffers.ArrayPoolEventSource/ManifestData
- Windows Kernel/DiskIO/FlushBuffers

:"14,076" ProcessorNumber="2" bufferId="39,449,526" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"14,076" ProcessorNumber="1" bufferId="50,346,327" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"13,980" ProcessorNumber="3" bufferId="14,333,193" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"12,740" ProcessorNumber="0" bufferId="64,109,423" bufferSize="262,144" poolId="32,854,180" bucketId="-1" reason="PoolExhausted"
:"13,980" ProcessorNumber="3" bufferId="13,009,416" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"10,280" ProcessorNumber="2" bufferId="49,924,125" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"12,740" ProcessorNumber="1" bufferId="35,236,192" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"12,740" ProcessorNumber="1" bufferId="21,943,666" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"13,980" ProcessorNumber="0" bufferId="41,728,762" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"13,980" ProcessorNumber="3" bufferId="2,174,563" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"13,644" ProcessorNumber="3" bufferId="64,828,693" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"13,644" ProcessorNumber="1" bufferId="10,104,599" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"10,280" ProcessorNumber="0" bufferId="41,773,672" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"13,980" ProcessorNumber="0" bufferId="63,062,333" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"13,980" ProcessorNumber="3" bufferId="16,868,352" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"12,740" ProcessorNumber="0" bufferId="53,052,340" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"13,644" ProcessorNumber="3" bufferId="51,288,387" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"14,076" ProcessorNumber="0" bufferId="50,874,780" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"12,740" ProcessorNumber="0" bufferId="5,896,758" bufferSize="16,384" poolId="32,854,180" bucketId="-1" reason="PoolExhausted"
:"12,740" ProcessorNumber="1" bufferId="60,375,305" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"14,076" ProcessorNumber="1" bufferId="3,318,699" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"14,076" ProcessorNumber="1" bufferId="55,683,007" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"10,280" ProcessorNumber="0" bufferId="4,831,898" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"10,280" ProcessorNumber="0" bufferId="34,361,009" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"10,280" ProcessorNumber="0" bufferId="50,492,551" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"13,644" ProcessorNumber="1" bufferId="7,141,266" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"12,740" ProcessorNumber="0" bufferId="46,228,029" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
:"13,644" ProcessorNumber="2" bufferId="44,313,942" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"

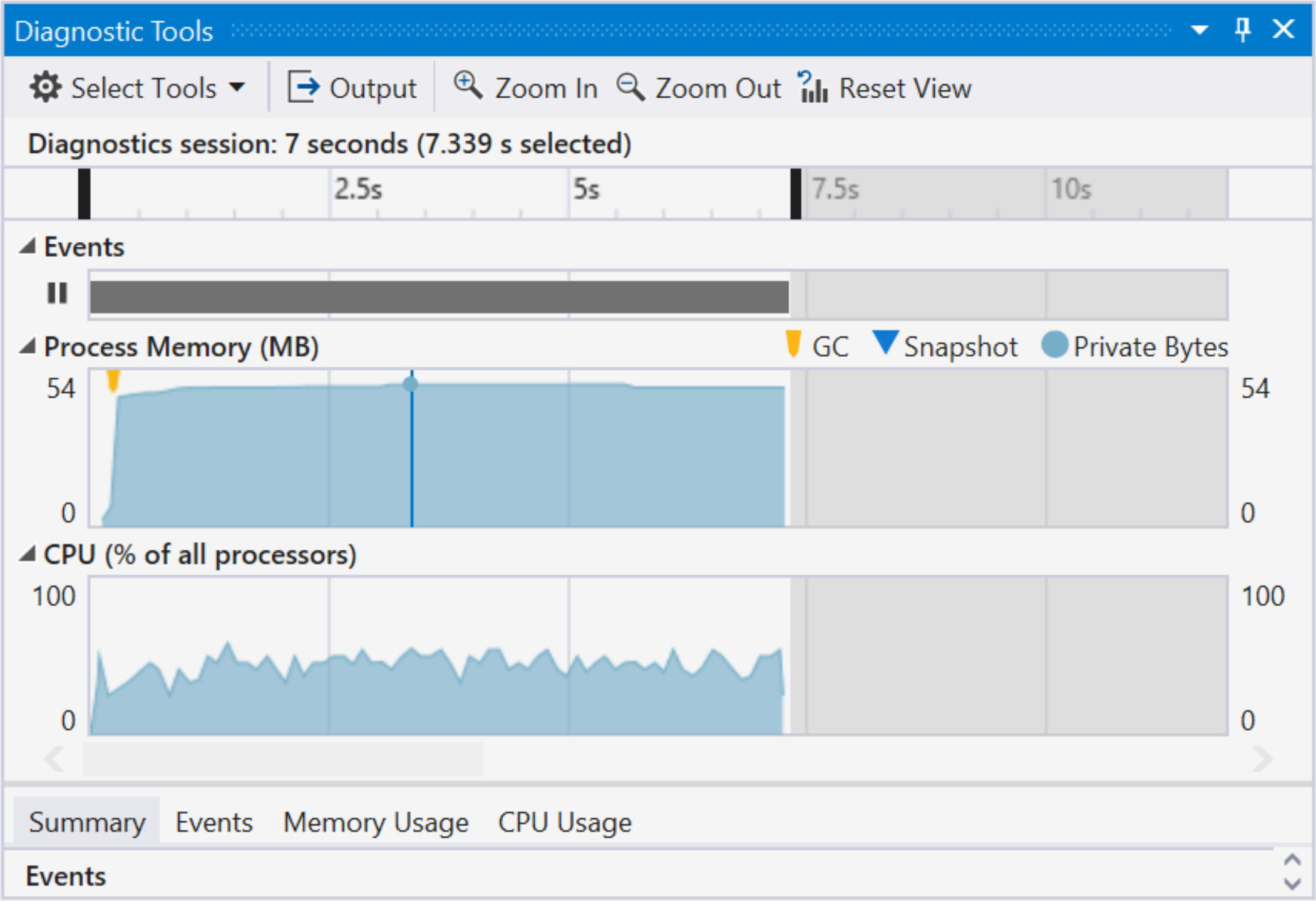
Found 75 Records. 75 total events.

Ready | Log | Cancel

0,280" ProcessorNumber="2" bufferId="49,924,125" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
2,740" ProcessorNumber="1" bufferId="35,236,192" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
2,740" ProcessorNumber="1" bufferId="21,943,666" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
3,980" ProcessorNumber="0" bufferId="41,728,762" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
3,980" ProcessorNumber="3" bufferId="2,174,563" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
3,644" ProcessorNumber="3" bufferId="64,828,693" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
3,644" ProcessorNumber="1" bufferId="10,104,599" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
0,280" ProcessorNumber="0" bufferId="41,773,672" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
3,980" ProcessorNumber="0" bufferId="63,062,333" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
3,980" ProcessorNumber="3" bufferId="16,868,352" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
2,740" ProcessorNumber="0" bufferId="53,052,340" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
3,644" ProcessorNumber="3" bufferId="51,288,387" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
4,076" ProcessorNumber="0" bufferId="50,874,780" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
2,740" ProcessorNumber="0" bufferId="5,896,758" bufferSize="16,384" poolId="32,854,180" bucketId="-1" reason="PoolExhausted"
2,740" ProcessorNumber="1" bufferId="60,375,305" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
4,076" ProcessorNumber="1" bufferId="3,318,699" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
4,076" ProcessorNumber="1" bufferId="55,683,007" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
0,280" ProcessorNumber="0" bufferId="4,831,898" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
0,280" ProcessorNumber="0" bufferId="34,361,009" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
0,280" ProcessorNumber="0" bufferId="50,492,551" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
3,644" ProcessorNumber="1" bufferId="7,141,266" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
2,740" ProcessorNumber="0" bufferId="46,228,029" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"
3,644" ProcessorNumber="2" bufferId="44,313,942" bufferSize="23,907,331" poolId="32,854,180" bucketId="-1" reason="OverMaximumSize"

ARRAYPOOL<BYTE>.CREATE

```
var biggestFile = Directory.GetFiles("images").Select(x => new FileInfo(x)).Select(x =>
x.Length).Max();
var arrayPool = ArrayPool<byte>.Create(maxArrayLength: Convert.ToInt32(biggestFile),
maxArraysPerBucket: 1);
var buffer = arrayPool.Rent(8 * 1024);
try
{
    for (int i = 0; i < Iterations; i++)
    {
        foreach (var file in Directory.EnumerateFiles("images"))
        {
            var fileInfo = new FileInfo(file);
            var array = arrayPool.Rent(Convert.ToInt32(fileInfo.Length));
            try
            {
                await using var stream = File.OpenRead(file);
                await ReadStreamToArray(fileInfo, stream, buffer, array);
            }
            finally
            {
                arrayPool.Return(array);
            }
        }
    }
}
finally
{
    arrayPool.Return(buffer);
}
```


Diagnostic Tools
Select Tools
Output
Zoom In
Zoom Out
Reset View
Diagnostics session: 7 seconds (7.339 s selected)
2.5s
5s
7.5s
10s
Events
Process Memory (MB)
GC
Snapshot
Private Bytes
54
0
54
0
CPU (% of all processors)
100
0
100
0
Summary
Events
Memory Usage
CPU Usage
Events

```
CommandLine:
"C:\dev\DPF\Src\DPF.ArrayPool\bin\Release\netcoreapp3.0\DPF.Arra
yPool.exe"
Runtime Version: V 4.0.30319.0 (built on 13/09/2019 01:02:04)
CLR Startup Flags: None
Total CPU Time: 11,358 msec
Total GC CPU Time: 8 msec
Total Allocs : 58.261 MB
GC CPU MSec/MB Alloc : 0.137 MSec/MB
Total GC Pause: 11.3 msec
% Time paused for Garbage Collection: 0.2%
% CPU Time spent Garbage Collecting: 0.1%
Max GC Heap Size: 37.899 MB
Peak Process Working Set: 51.962 MB
Peak Virtual Memory Usage: 2,199,574.643 MB
```

Histogram: 3 66 A 3 66

D="6,356" ProcessorNumber="0" bufferId="12,547,953" bufferSize="8,192" poolId="11,429,296" bucketId="-1" reason="PoolExhausted"
D="12,252" ProcessorNumber="2" bufferId="28,145,867" bufferSize="128" poolId="58,604,500" bucketId="-1" reason="PoolExhausted"
D="12,252" ProcessorNumber="2" bufferId="19,956,848" bufferSize="256" poolId="58,604,500" bucketId="-1" reason="PoolExhausted"
D="14,400" ProcessorNumber="1" bufferId="65,849,037" bufferSize="128" poolId="58,604,500" bucketId="-1" reason="PoolExhausted"
D="14,400" ProcessorNumber="1" bufferId="62,407,605" bufferSize="256" poolId="58,604,500" bucketId="-1" reason="PoolExhausted"
D="14,400" ProcessorNumber="3" bufferId="44,484,078" bufferSize="512" poolId="58,604,500" bucketId="-1" reason="PoolExhausted"
D="12,300" ProcessorNumber="2" bufferId="37,489,757" bufferSize="8,192" poolId="64,828,693" bucketId="-1" reason="PoolExhausted"
D="14,400" ProcessorNumber="3" bufferId="52,564,479" bufferSize="4,096" poolId="64,828,693" bucketId="-1" reason="PoolExhausted"
D="9,132" ProcessorNumber="2" bufferId="4,016,864" bufferSize="512" poolId="14,406,273" bucketId="-1" reason="PoolExhausted"
D="13,344" ProcessorNumber="3" bufferId="32,854,180" bufferSize="8,192" poolId="58,225,482" bucketId="27,252,167" reason="Pooled"
D="13,344" ProcessorNumber="3" bufferId="43,942,917" bufferSize="33,554,432" poolId="58,225,482" bucketId="59,941,933" reason="Pooled"
D="10,076" ProcessorNumber="0" bufferId="35,320,229" bufferSize="16,384" poolId="58,225,482" bucketId="17,653,682" reason="Pooled"
D="10,076" ProcessorNumber="0" bufferId="42,194,754" bufferSize="262,144" poolId="58,225,482" bucketId="15,688,314" reason="Pooled"

	Allocated bytes	Max GC Heap Size
New arrays	1,263.242 MB	145.235 MB
Shared ArrayPool	1,014.148 MB	146.111 MB
Custom ArrayPool<byte>	58.261 MB	37.899 MB

TIP

ARRAYPOOL.SHARED ALLOCATES MEMORY IF IT NEEDS TO RENT ARRAY BIGGER THAN 2^20 (1024 * 1024) ELEMENTS

ISSUE #6

RELYING ON THE LIBRARY AND CROSS PLATFORM FEATURES

```
public class TelemetryEntry
{
    public DateTimeOffset Timestamp { get; set; }
    public int UserId { get; set; }
}
```

```
[Route("[controller]")]
[ApiController]
public class TelemetryController : Controller
{
        [HttpPut]
        public async Task<IActionResult> Insert([FromServices]
DataStorageService db)
        {
                var entry = CreateEntry();
                await db.Insert(entry);
                return NoContent();
        }
}
```

```
public class SlowDataStorageService
{
        private readonly DataStorageServiceOptions _options;
        private readonly DocumentClient _client;

        public SlowDataStorageService(DataStorageServiceOptions options)
        {
                _options = options;
                _client = new DocumentClient(options.Endpoint, options.ApiKey);
                _client.CreateDatabaseIfNotExistsAsync(new Database { Id =
options.DatabaseName }).GetAwaiter().GetResult();
                _client.CreateDocumentCollectionIfNotExistsAsync(
                     UriFactory.CreateDatabaseUri(options.DatabaseName),
                     new DocumentCollection { Id = options.CollectionName })
                  .GetAwaiter().GetResult();
         }
}
```

```
public async Task Insert(TelemetryEntry telemetryEntry)
{
        var collectionUri =
                UriFactory.CreateDocumentCollectionUri(_options.DatabaseName,
_options.CollectionName);
        await _client.CreateDocumentAsync(collectionUri, telemetryEntry);
}
```

```
public void ConfigureServices(IServiceCollection services)
{
      var dbOptions = GetOptions();
      services.AddSingleton(dbOptions);
      services.AddTransient<DataStorageService>();
}
```

TESTING – NO DOCS

```
Running 30s test @ http://localhost:5000/telemetry/insert/slow
  1 threads and 1 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   876.89ms   45.82ms 993.87ms   76.47%
    Req/Sec     0.88      0.33     1.00     88.24%
  34 requests in 30.07s, 2.69KB read
Requests/sec:      1.13
Transfer/sec:      91.57B
```

QUITE SLOW

QUITE UGLY

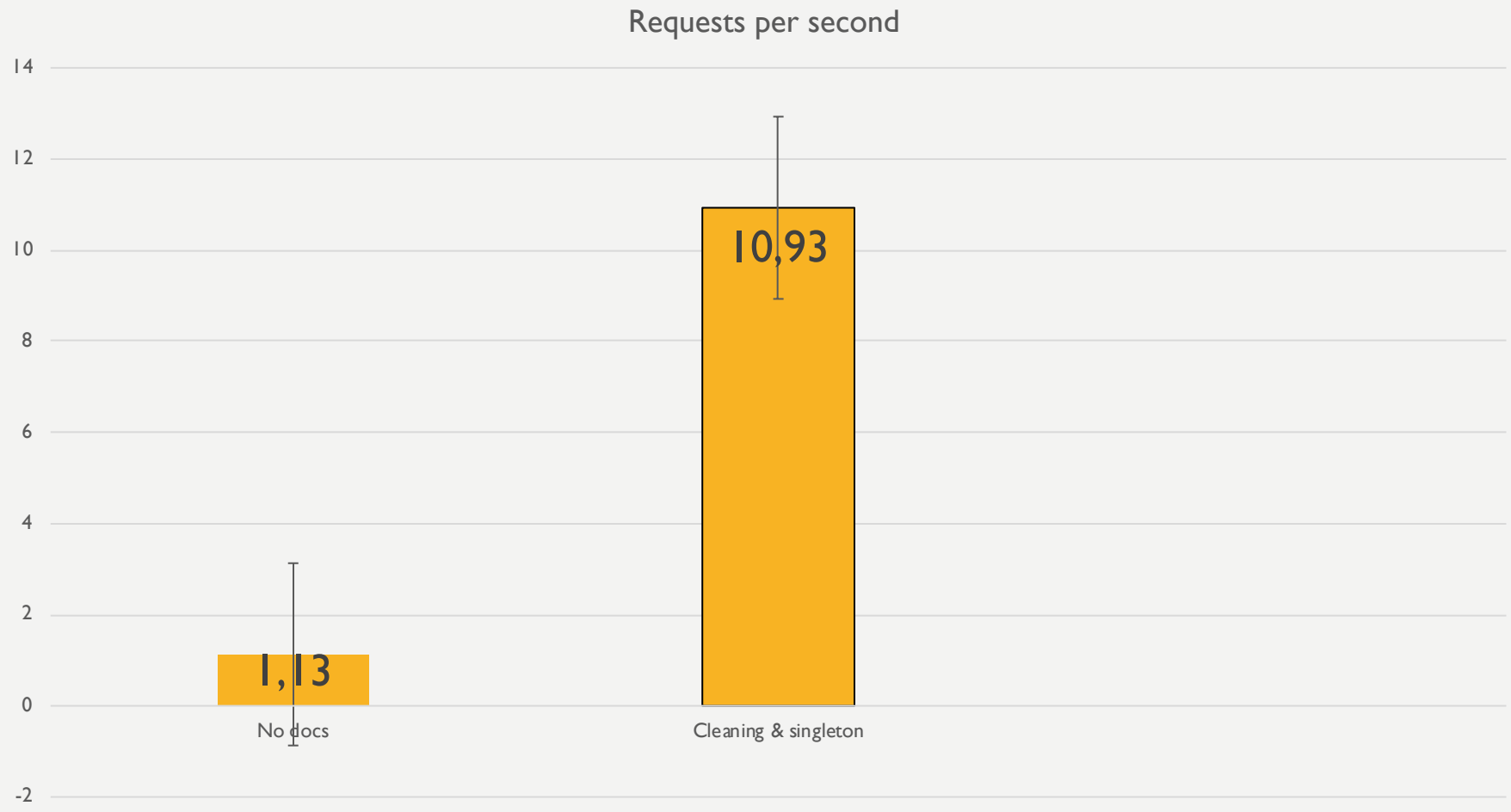
```
public class AsyncLazy<T> : Lazy<Task<T>>
{
      public AsyncLazy(Func<Task<T>> valueFactory) :
base(valueFactory)
      {
      }
}
```

```
public class DataStorageService
{
        private readonly AsyncLazy<DocumentClient> _clientFactory;
        private readonly Uri _collectionUri;
        public DataStorageService(DataStorageServiceOptions options)
        {
                _collectionUri = 
UriFactory.CreateDocumentCollectionUri(options.DatabaseName,
options.CollectionName);
                _clientFactory = new AsyncLazy<DocumentClient>(async () =>
                {
                    var client = new DocumentClient(options.Endpoint,
options.ApiKey);
                   await InitialDatabase(client, options);
                });
         }
}
```

```
public async Task Insert(TelemetryEntry telemetryEntry)
{
        var client = await _clientFactory.Value;
        await client.CreateDocumentAsync(_collectionUri, telemetryEntry);
}
```

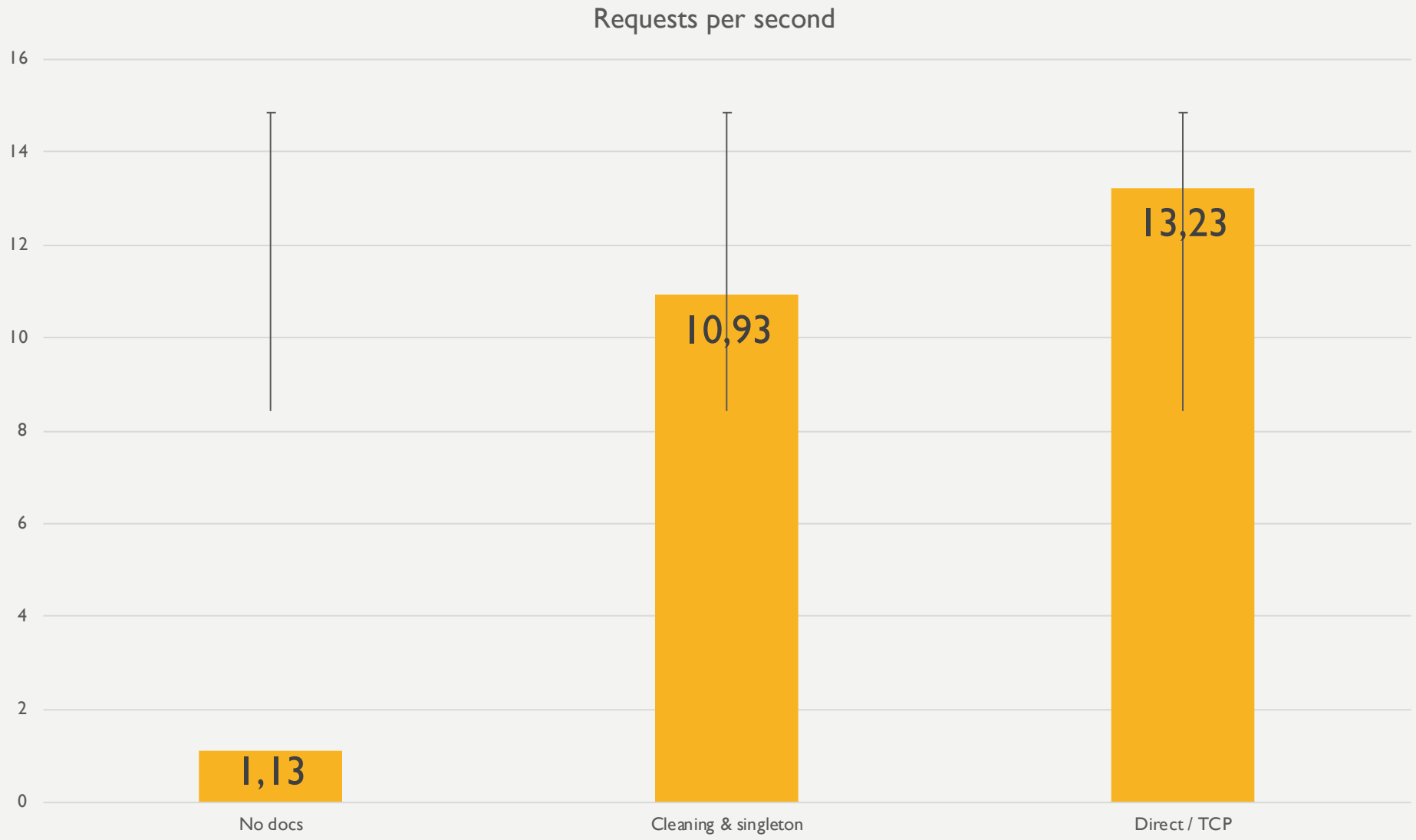
```
public void ConfigureServices(IServiceCollection services)
{
        var dbOptions = GetOptions();
        services.AddSingleton(dbOptions);
        services.AddSingleton<DataStorageService>();
}
```

TESTING - OPTIMIZATIONS



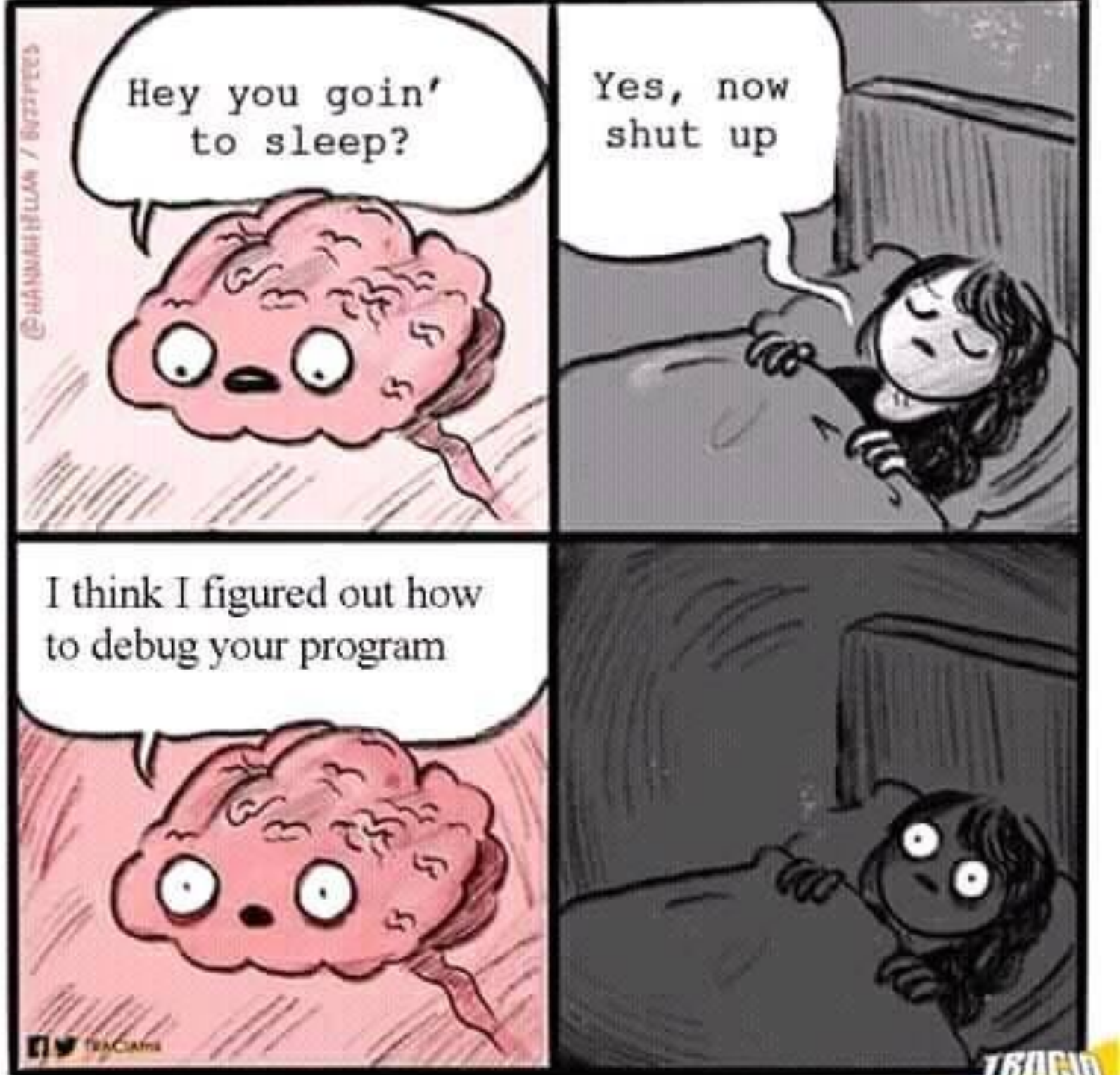
```
_clientFactory = new AsyncLazy<DocumentClient>(async () =>
{
      var client = new DocumentClient(options.Endpoint,
options.ApiKey, new ConnectionPolicy
      {
            ConnectionMode = ConnectionMode.Direct,
            ConnectionProtocol = Protocol.Tcp
      });
      await InitialDatabase(client, options);
});
```

TESTING – DIRECT / TCP



THE GONEEXCEPTION STORY

- Microservices
- CosmosDB
- Migration to the Direct/TCP connection policy, feature toggle
- Thousands of the GoneException saying "The requested resource is no longer available at the server"
- We have a bug!!!
- Race condition? Dispose? Network down? Firewall?



@lukaszpyrzyk

Microsoft.Azure.DocumentDB.Core

2.0.0-preview

This client library enables client applications targeting .NET Core to connect to Azure Cosmos DB via the DocumentDB (SQL) API. Azure Cosmos DB is a globally distributed, multi-model database service. For more information, refer to https://azure.microsoft.com/services/cosmos-db/.

ⓘ This is a prerelease version of Microsoft.Azure.DocumentDB.Core.

ⓘ There is a newer version of this package available.
See the version list below for details.

Package Manager | .NET CLI | Paket CLI

```
PM> Install-Package Microsoft.Azure.DocumentDB.Core -Version 2.0.0-preview
```

yrzyk

1.0.0	24,716	22/12/2016
0.1.0-preview	10,165	16/11/2016
0.0.6-preview	372	15/11/2016
0.0.5-preview	392	15/11/2016

THE GONEEXCEPTION STORY

GoneException("The requested resource is no longer available at the server")

might be

PlatformNotSupportedException();

filipw commented on 18 May • edited

I think the fact that Direct mode is not supported in the .NET Standard 2.0 client on non-Windows platform is a pretty large adoption roadblock, especially given how much push for Unix-based containerization we see in the ASP.NET Core world.

It would be very helpful if:

- this information was added to this article https://docs.microsoft.com/en-us/azure/cosmos-db/performance-tips. This is particularly confusing, as the article suggests to use direct mode as the default one, which can easily lead to this error
- this information was added as known limitation to the .NET Standard SDK release notes https://docs.microsoft.com/en-us/azure/cosmos-db/sql-api-sdk-dotnet-core. At the moment this page states that `The Azure Cosmos DB .NET Core SDK has feature parity with the latest version of the Azure Cosmos DB .NET SDK` which is simply false.
- the SDK should throw a `PlatformNotSupportedException` on macOS / Linux. At the moment the exception is a `GoneException` with a message that `The requested resource is no longer available at the server.` which is very difficult to troubleshoot and doesn't really reflect the real situation
- of course ultimately, the most helpful thing would be if this was fixed, or at least some more concrete plan of how and when this will be dealt with (rather than "first half of 2018", as we are almost there already) was shared here

thanks ⭐

 8

https://github.com/Azure/azure-cosmosdb-dotnet/issues/194

Microsoft.Azure.Cosmos 3.3.2

This client library enables client applications to connect to Azure Cosmos via the SQL API. Azure Cosmos is a globally distributed, multi-model database service. For more information, refer to https://azure.microsoft.com/services/cosmos-db/.

Package Manager | .NET CLI | **PackageReference** | Paket CLI

```
<PackageReference Include="Microsoft.Azure.Cosmos" Version="3.3.2" />
```

ⓘ For projects that support PackageReference, copy this XML node into the project file to reference the package.

ISSUE #7

LET'S OPTIMIZE IT

THE BUSINESS PROBLEM

- This is very important for our company
- It's PoC
- Please make it ASAP
- Performance doesn't matter
- Don't spend to much time on it
- Keep clean and easy to understand for another developers

```
public ulong SecretAlgorithm(ulong n)
{
        if (n == 1 || n == 2) return 1;
        return SecretAlgorithm(n - 2) + SecretAlgorithm(n - 1);
}
```

```
public ulong Fibonacci(ulong n)
{
        if (n == 1 || n == 2) return 1;
        return Fibonacci(n - 2) + Fibonacci(n - 1);
}
```

THE BUSINESS PROBLEM

- It works
- It is simple

THE BUSINESS PROBLEM

Make it faster!

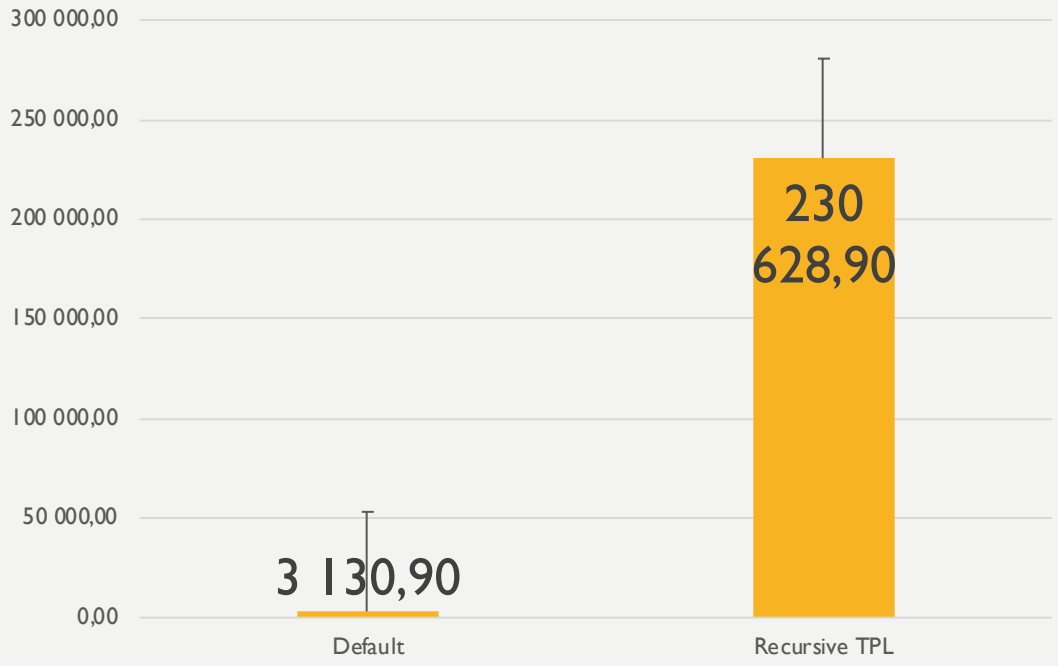
THE BUSINESS PROBLEM

What kind of optimizations techniques do I know?

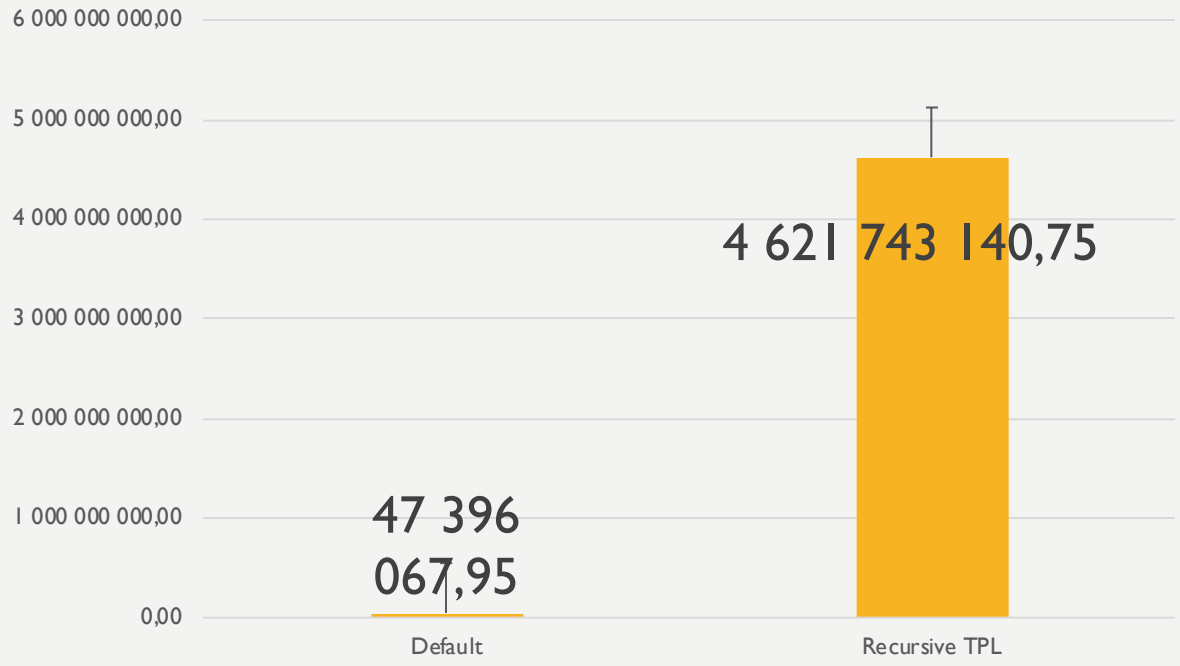
THREADS !!!

```
public ulong Fibonacci(ulong n)
{
        if (n == 1 || n == 2) return 1;
        var a = Task.Run(() => Fibonacci(n - 2));
        var b = Task.Run(() => Fibonacci(n - 1));
        Task.WaitAll(a, b);
        return a.Result + b.Result;
}
```

n = 15, ns



n = 35, ns

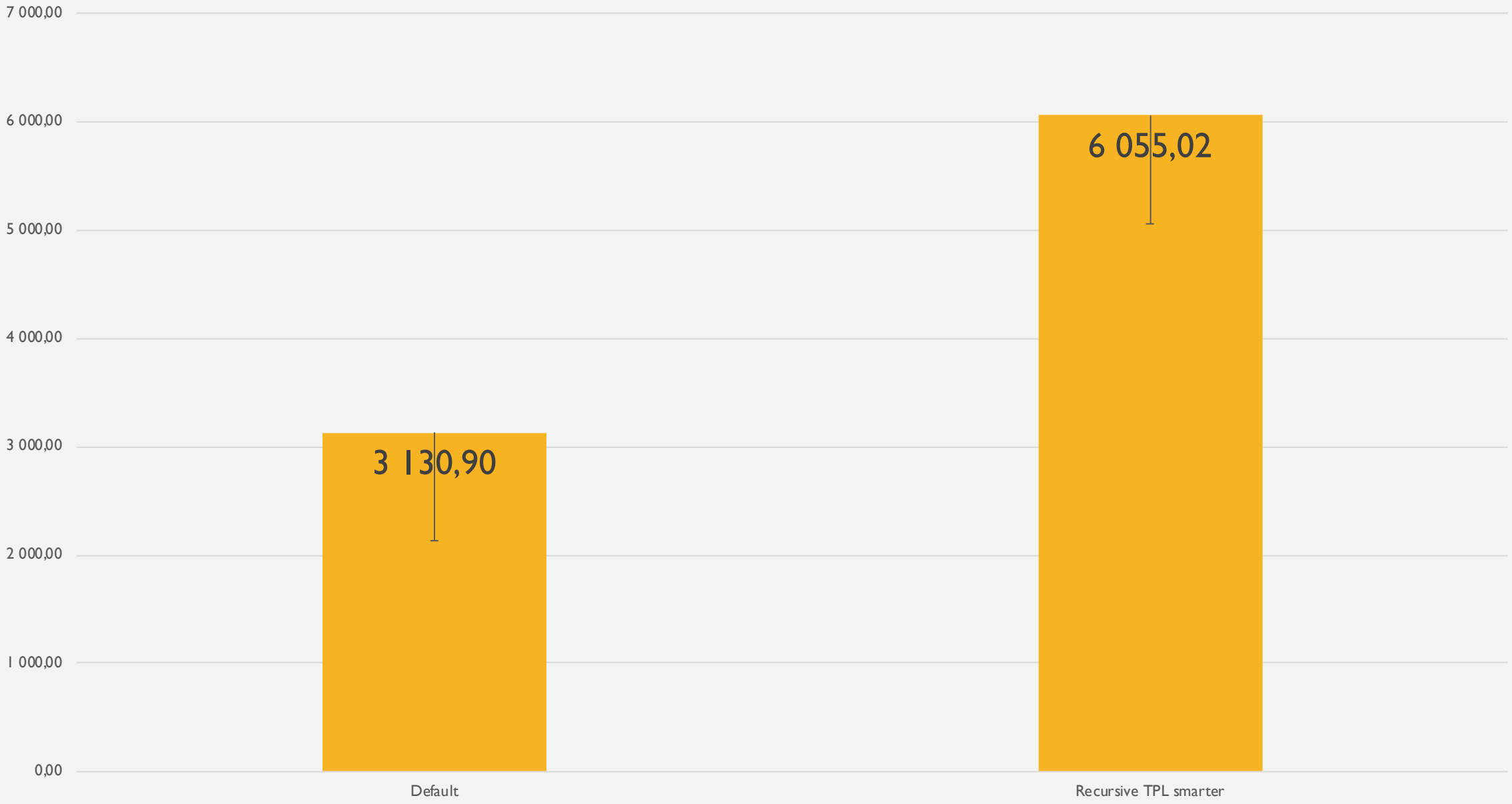


@lukaszpyrzyk

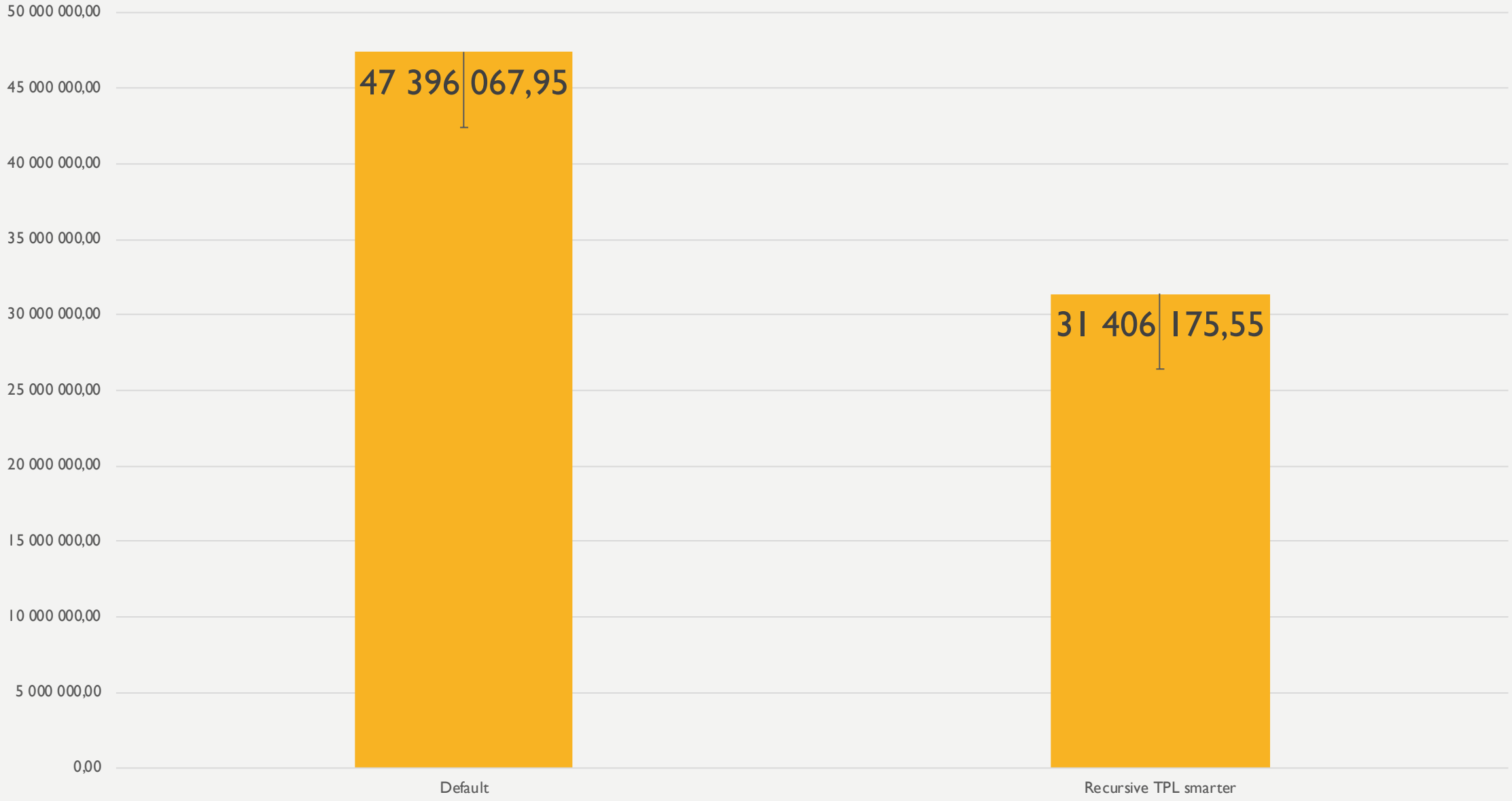
LESS THREADS

```
public ulong Fibonacci(ulong n)
{
        if (n == 1 || n == 2) return 1;
        var a = Task.Run(() => FibonacciImplementation(n - 2));
        var b = Task.Run(() => FibonacciImplementation(n - 1));
        Task.WaitAll(a, b);
        return a.Result + b.Result;
}
private ulong FibonacciImplementation(ulong n)
{
        if (n == 1 || n == 2) return 1;
        return Recursive(n - 2) + Recursive(n - 1);
}
```

N = 15, ns


7 000,00
6 000,00
5 000,00
4 000,00
3 000,00
2 000,00
1 000,00
0,00
3 130,90
6 055,02
Default
Recursive TPL smarter

N = 35, ns


50 000 000,00
45 000 000,00
40 000 000,00
35 000 000,00
30 000 000,00
25 000 000,00
20 000 000,00
15 000 000,00
10 000 000,00
5 000 000,00
0,00
47 396 067,95
31 406 175,55
Default
Recursive TPL smarter

IF STATEMENT

```
public ulong Fib(ulong n)
{
    if (n == 1 || n == 2) return 1;
    const int goldenNumber = 18;
    return n < goldenNumber ? Recursive(n) : RecursiveTPLStart(n);
}
```

Let's debug it

```
public static ulong Recursive(ulong n)
{
        Console.WriteLine($"Calculating Fibonacci for {n}");
        if (n == 1 || n == 2) return 1;
        return Recursive(n - 2) + Recursive(n - 1);
}
```

```
Fibonacci(5);
```

Calculating Fibonacci for 5

Calculating Fibonacci for 3

Calculating Fibonacci for 1

Calculating Fibonacci for 2

Calculating Fibonacci for 4

Calculating Fibonacci for 2

Calculating Fibonacci for 3

Calculating Fibonacci for 1

Calculating Fibonacci for 2

Algoritms and data structures

MEMOIZATION

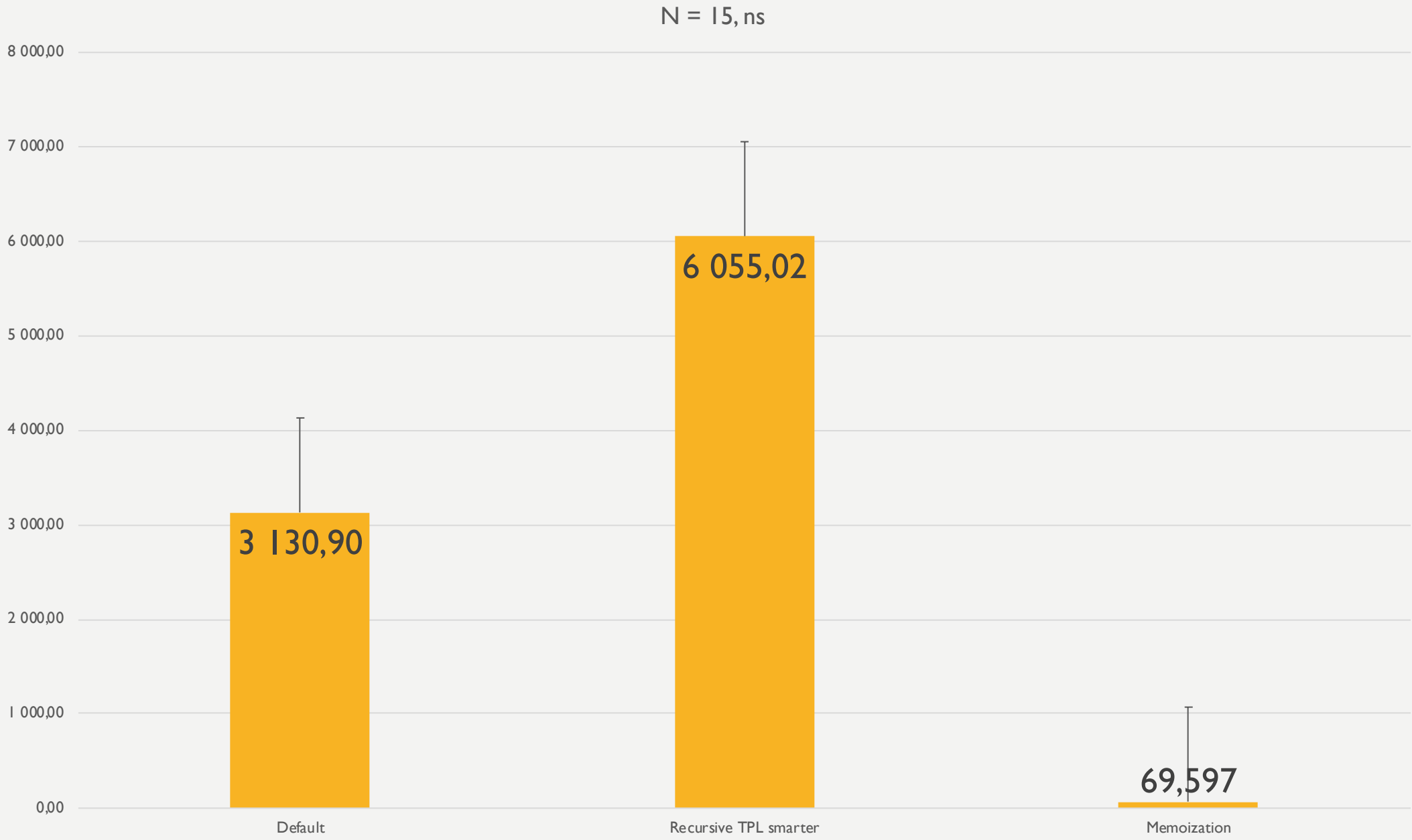
It is an optimization technique used primarily to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again

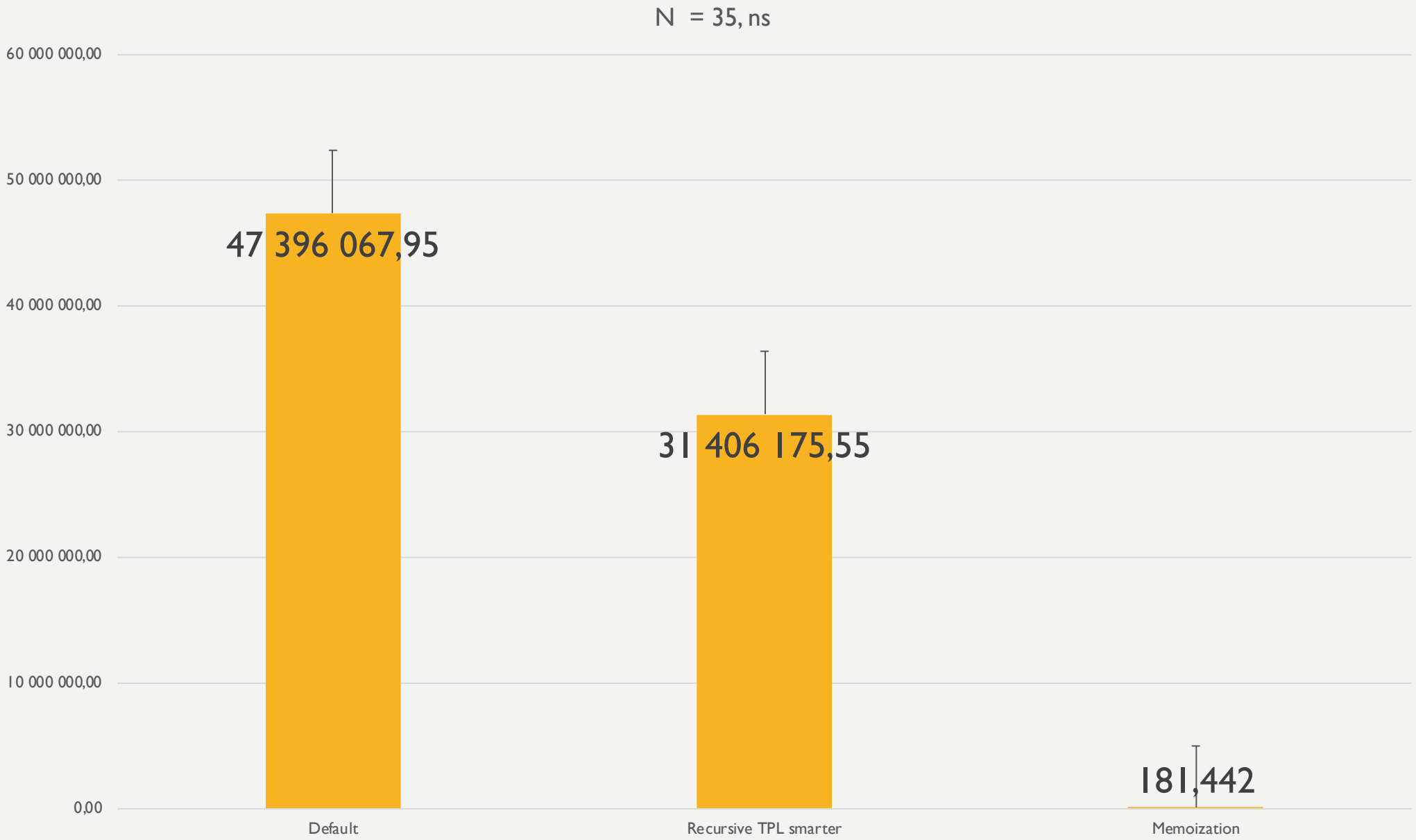
MEMOIZATION

```
public ulong Fibonacci(ulong n)
{
        if (n == 1 || n == 2) return 1;
        var results = new ulong[n];
        results[0] = 1;
        results[1] = 1;
        return FibWithMemoization(n, results);
}
```

MEMOIZATION

```
private static ulong FibWithMemoization(ulong n, ulong[] results)
{
        var current = n - 1;
        var previous = current - 1;
        var beforePrevious = previous - 1;
        if (results[beforePrevious] == 0)
                results[beforePrevious] = FibWithMemoization(previous, results);
        if (results[previous] == 0)
                results[previous] = FibWithMemoization(current, results);

        results[current] = results[beforePrevious] + results[previous];
        return results[n - 1];
}
```

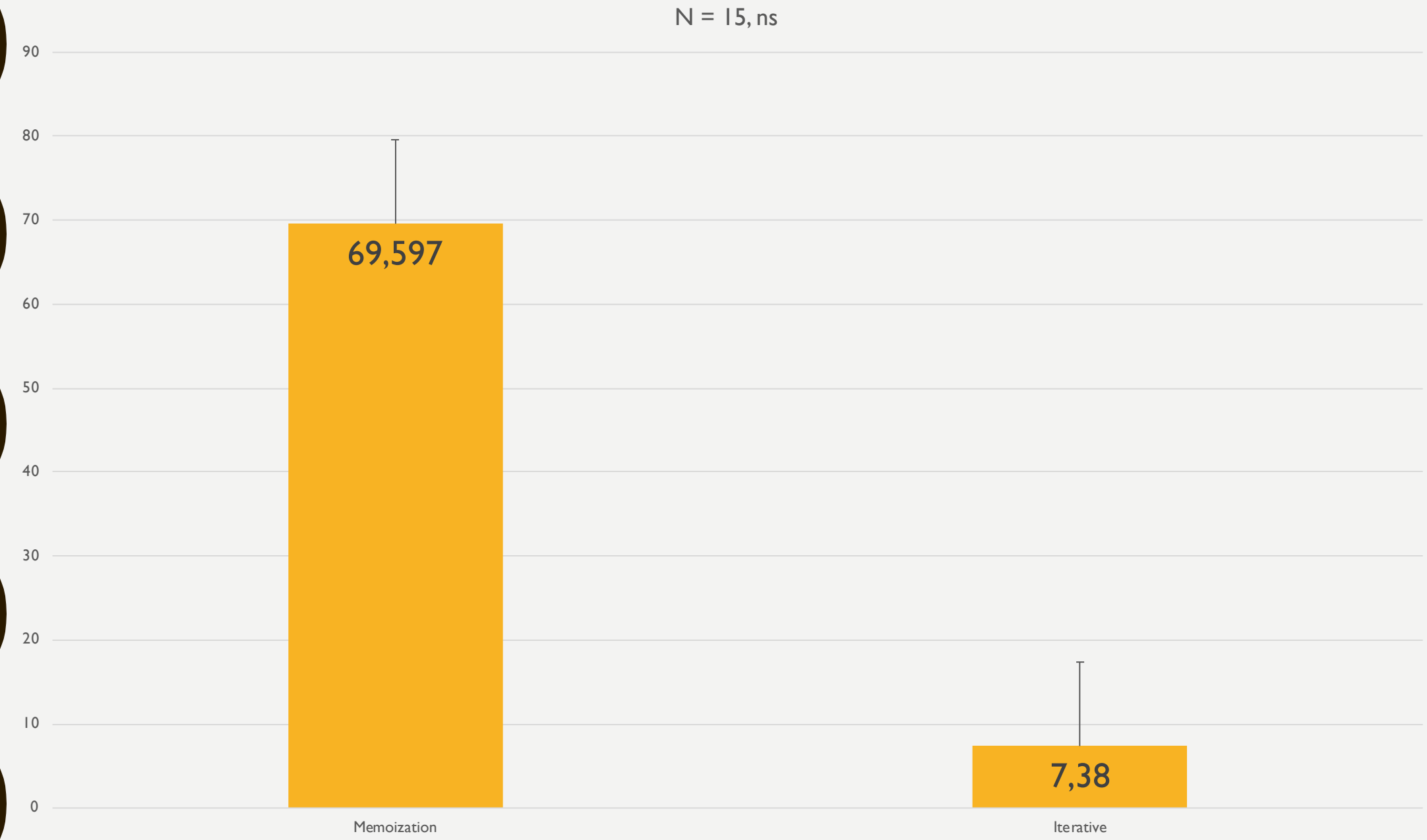

N = 15, ns
8 000,00
7 000,00
6 000,00
5 000,00
4 000,00
3 000,00
2 000,00
1 000,00
0,00
3 130,90
6 055,02
69,597
Default
Recursive TPL smarter
Memoization



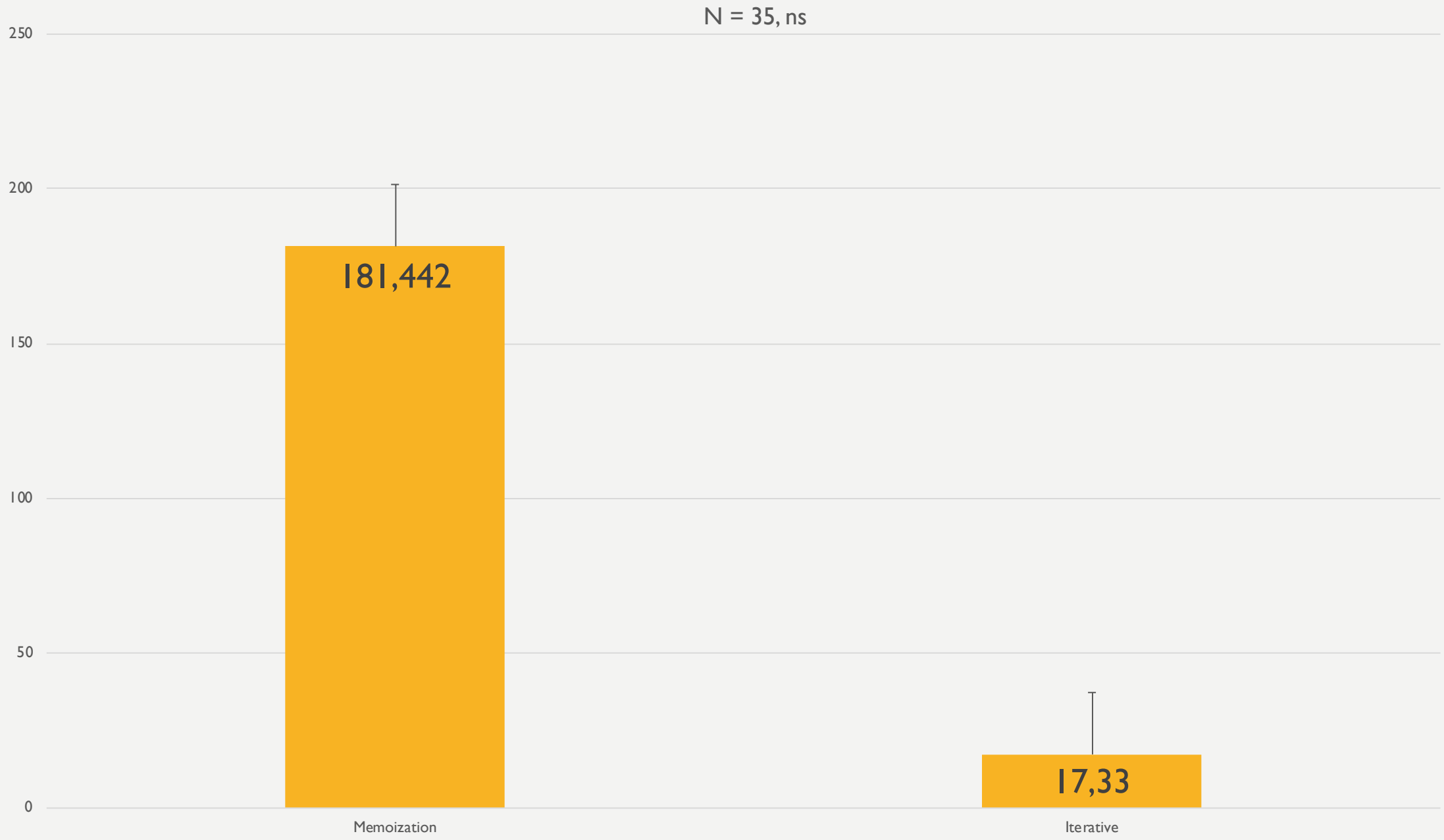
@lukaszpyrzyk

WHAT WE MISSED?

```
public ulong Fibonacci(ulong n)
{
        if (n == 1 || n == 2) return 1;
        ulong a = 1, b = 1;
        for (ulong i = 2; i < n; i++)
        {
                ulong temp = a + b;
                a = b;
                b = temp;
        }
        return b;
}
```



@lukaszpyrzyk



@lukaszpyrzyk

TIP

BEFORE DOING OPTIMIZATIONS MAKE SURE THAT CURRENT SOLUTION STILL FITS

SUMMARY

- We should always validate our performance improvements by measurement
- Releasing product in debug sounds funny, but there are real cases where by mistake debug code was shipped to the clients
- Tiered compilation is new in .NET world and we need get used to it. It is a good idea to disable tierec compilation for profiling and benchmarking or get code warm
- Reading documentation and API notes may give you easy performance boost, for example by making CosmosDB client singleton with Direct connection or using HttpCompletionOption
- ETW event are a great collection of knowledge about our application, for example about ArrayPool usage
- Before doing code optimizations we should understand how it is used

LINKS

- Konrad Kokosa - Pro .NET Memory Management, https://prodotnetmemory.com/
- Andrey Akinshin - Pro .NET Benchmarking, https://www.apress.com/gp/book/9781484249406
- https://docs.microsoft.com/en-us/azure/cosmos-db/performance-tips
- https://docs.microsoft.com/en-us/azure/cosmos-db/sql-api-sql-query-metricshttp://www.tugberkugurlu.com/archive/efficiently-streaming-large-http-responses-with-httpclient
- https://github.com/davidfowl/AspNetCoreDiagnosticScenarios
- https://wojciechnagorski.com/2018/12/how-i-improved-the-yamldotnet-performance-by-370/
- https://github.com/aaubry/YamlDotNet/pull/356

Premium .NET Conference with top class speakers

Warsaw, Poland
dotnetos.org

Thank you!

@lukaszpyrzyk

lukasz.pyrzyk@gmail.com

https://github.com/lukasz-pyrzyk/DailyPerformancePitfalls