

High performance Coding with .NET Core and C#

Gergely Kalapos
Elastic

Web: www.kalapos.net
Twitter: [@gregkalapos](https://twitter.com/gregkalapos)



Gergely Kalapos

Web: <http://kalapos.net>
Twitter: @gregkalapos

- Works on observability and performance monitoring tools at Elastic
- Loves OpenSource, works in a distributed Team
- Born in Hungary, lives in Austria

GitHub repo with slides and sample code

tiny.cc/gwm46y

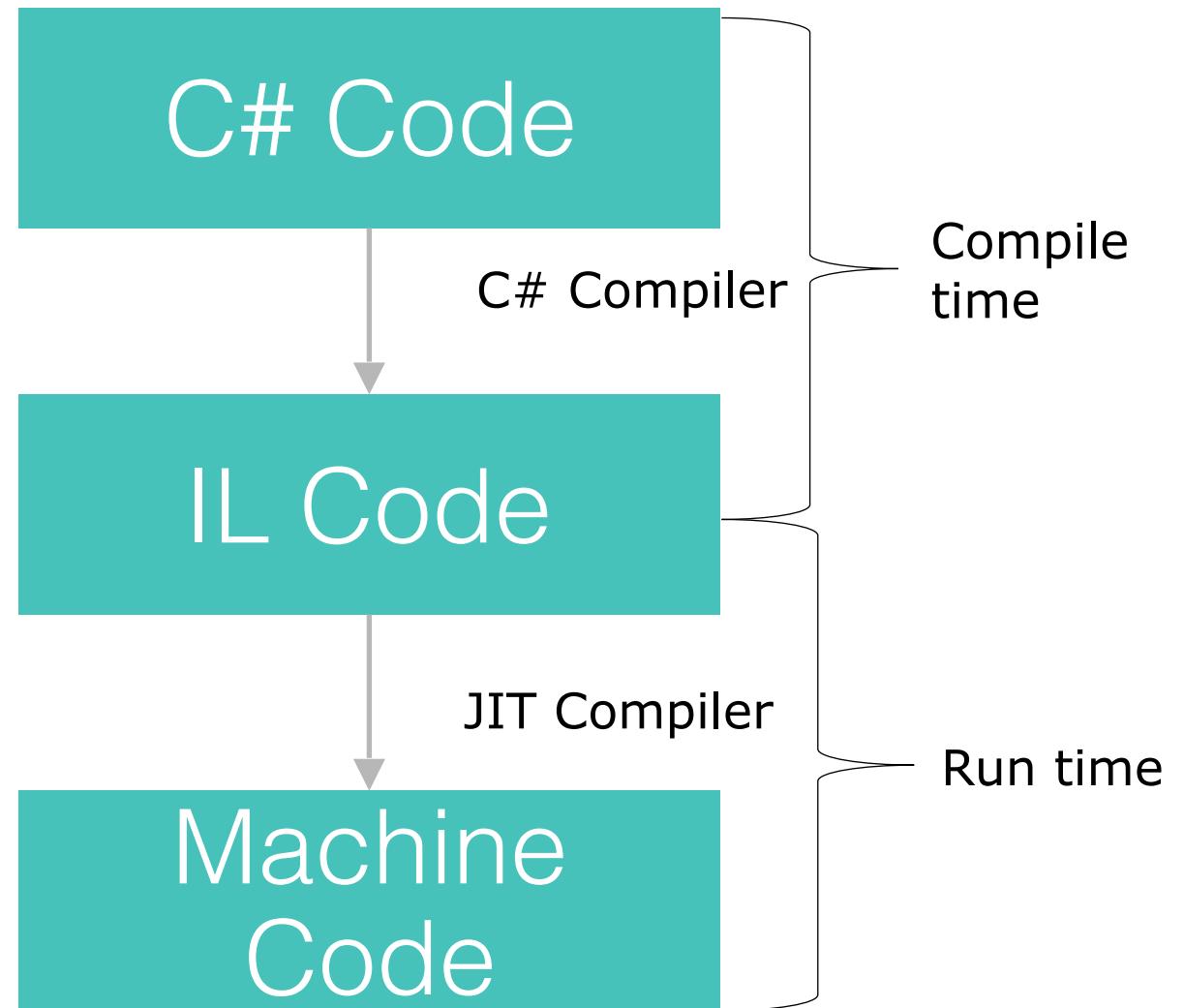
Agenda

- LINQ Sample - .NET Core vs. Full Framework
- BenchmarkDotNet
- Why is Performance important?
- ArrayPool
- C# 7 ref return
- Span<T>
- SIMD and Vectorization

Gergely Kalapos
 @gregkalapos
 www.kalapos.net

Terminology

- Full Framework
- Desktop CLR
- .NET Core
- CoreCLR
- IL Code



Full Framework vs. .NET Core – LINQ – Moving average calculation



Full Framework vs .NET Core - LINQ

- Benchmarking - summary:
 - MSFT Stock
 - 14-days Simple Moving Average
 - Von 1986
 - ~80.000 data points
- https://github.com/gregkalapos/ADC19Samples/tree/master/Sample1_Linq_FullFramework_Vs_DotNetCore

Full Framework vs .NET Core - LINQ

Method	Job	Runtime	Mean	Error	StdDev	Gen 0	Gen 1	Allocated
CalculateWithLinq	Clr	Clr	541.506 ms	2.5407 ms	2.2523 ms	875.0000	312.5000	3.47 MB
CalculateWithLinq	Core	Core	9.015 ms	0.1246 ms	0.1104 ms	281.2500	93.7500	1.14 MB


```
// * Hints *
Outliers
Program.CalculateWithLinq: Clr  -> 1 outlier was removed
Program.CalculateWithLinq: Core -> 1 outlier was removed

// * Legends *
Mean      : Arithmetic mean of all measurements
Error     : Half of 99.9% confidence interval
StdDev    : Standard deviation of all measurements
Gen 0     : GC Generation 0 collects per 1k Operations
Gen 1     : GC Generation 1 collects per 1k Operations
Allocated : Allocated memory per single operation (managed only, inclusive, 1KB = 1024B)
1 ms      : 1 Millisecond (0.001 sec)
```

~60x Performance improvement thanks to Community Contribution (Jon Hanna)

<https://github.com/dotnet/corefx/commit/>

a087c2d97d818762c1b4dd2c6305a2d9e836631c#diff-5c3753c526c75a2790d71c348c0c364aR246

<https://github.com/dotnet/corefx/blob/master/src/System.Linq/src/System/Linq/Skip.cs#L35>

.NET Blog - Performance Improvements in .NET Core

- Posts with perf. improvements with benchmarks

<https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-core-3-0/>

<https://blogs.msdn.microsoft.com/dotnet/2017/06/07/performance-improvements-in-net-core/>



.NET Blog

A first-hand look from the .NET engineering teams

Performance Improvements in .NET Core

June 7, 2017 by Stephen Toub - MSFT // 68 Comments



Update (2017/06/12): Added BenchmarkDotNet blog post link.

There are many exciting aspects to .NET Core (open source, cross platform, x-copy deployable, etc.) that have been covered in posts on this blog before. To me, though, one of the most exciting aspects of .NET Core is performance. There's been a lot of discussion about the significant advancements that have been made in ASP.NET Core performance, its status as a top contender on various TechEmpower benchmarks, and the continual advancements being made in pushing it further. However, there's been much less discussion about some equally exciting improvements throughout the runtime and the base class libraries.

There are way too many improvements to mention. After all, as an open source project that's [very accepting of contributions](#), Microsoft and community developers from around the world have found places where performance is important to them and submitted pull requests to improve things. I'd like to thank all the community developers for their .NET Core contributions, some of which are specifically called out in this post. We expect that many of these improvements will be brought to the .NET Framework over

BenchmarkDotNet

- 15:45-16:51: "Einstieg in Performanceoptimierung mit BenchmarkDotNet"
- Add NuGet package: “BenchmarkDotNet”

BenchmarkDotNet

```
namespace BenchmarkSample
{
    [MemoryDiagnoser]
    public class Program
    {
        static void Main(string[] args)
        {
            var summary = BenchmarkRunner.Run<Program>();
        }
    [Benchmark]
    public static void BenchmarkMethod()
    {
        //work to measure
    }
}
```


Why is Performance important? – Cloud

- Azure App Service costs

Basic:

INSTANZ	KERNE	RAM	STORAGE	PREISE
B1	In case an application runs on a: • single B1 instance: 54,75\$/Monat	1,75 GB	10 GB	~\$54,75/Monat
B2	• Single P3v2 instance: 584\$/Monat	3,50 GB	10 GB	~\$109,50/Monat
B3	• 50 P3v2 instances : 29200\$/Monat	7 GB	10 GB	~\$219/Monat

Premium:

- 50 P3v2 instances : 29200\$/Monat

INSTANZ	KERNE	RAM	STORAGE	PREISE
P1v2	1	3,50 GB	250 GB	~\$146/Monat
P2v2	2	7 GB	250 GB	~\$292/Monat
P3v2	4	14 GB	250 GB	~\$584/Monat

Why is Performance important? – Mobile

The screenshot shows a section of the Apple Developer website titled "Energy Efficiency and the User Experience". The page includes a search bar, navigation links for "Energy Essentials" and "Energy Efficiency and the User Experience", and a "On This Page" dropdown menu.

Your Obligation as a Developer

Even small inefficiencies in apps add up, significantly affecting battery life, performance, and responsiveness. As an app developer, you have an obligation to make sure your app runs as efficiently as possible. Use recommended APIs so the system can make smart decisions about how best to manage your app and the resources it uses. Whenever possible, batch and reduce network operations, and avoid unnecessary updates to the user interface. Power-intensive operations should be under the user's control. If a user is playing a graphics-heavy game, for example, the user should not be surprised if the activity consumes power. Strive to make your app absolutely idle when it is not responding to user input.

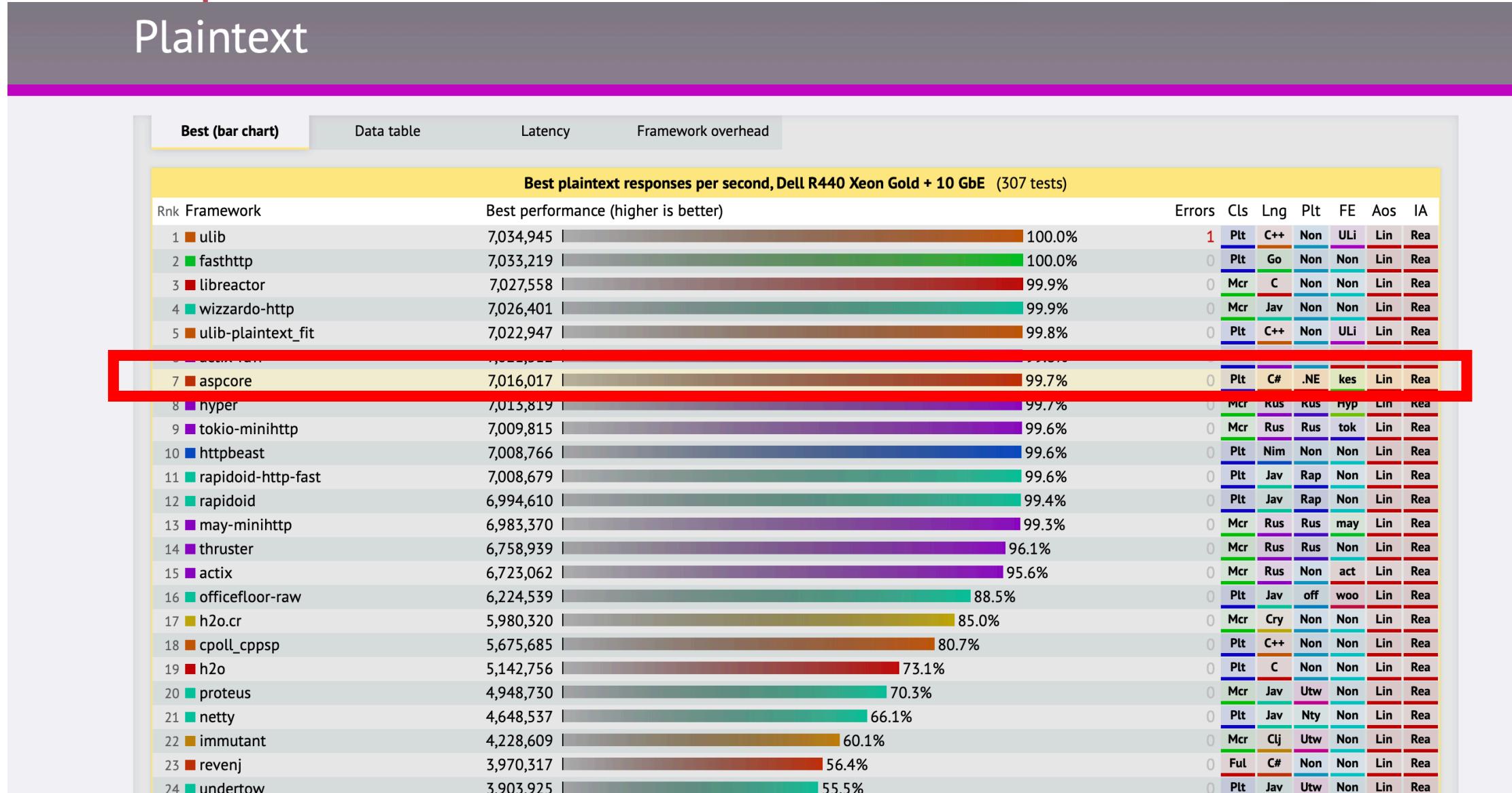
By adhering to recommended guidelines, you can make big contributions to the overall energy efficiency of the platform and the satisfaction of your users.

Web applications

- SEO
- Conversion-rate
- Slow Website -> Unhappy users
- Docker

TechEmpower Benchmark

Plaintext



TechEmpower Benchmark



TechEmpower Benchmark

275	spock	12,840	10.2%	0	Mcr	Hkl	Wai	Wrp
276	valum	10,879	10.2%	249	Mcr	val	gli	Lin
277	chicagoboss	10,711	10.2%	0	Ful	Erl	Cow	Rea
278	vsgi	10,710	10.2%	970	Plt	val	gli	Lin
279	nancy	8,775	10.1%	0	Mcr	C#	.NE	Rea
280	cakephp	7,706	10.1%	0	Ful	PHP	Non	ngx
281	laravel	7,624	10.1%	0	Ful	PHP	Non	ngx
282	bottle-nginx-uwsgi	7,258	10.1%	0	Mcr	Py	uWS	ngx
283	flask-nginx-uwsgi	7,189	10.1%	0	Mcr	Py	Non	ngx
284	slim-hhvm	6,959	10.1%	7,087	Mcr	PHP	Non	ngx
285	aspnet-mono-nginx	5,666	10.1%	0	Plt	C#	.NE	ngx
286	servicestack	5,450	10.1%	0	Ful	PHP	Non	ngx
287	zend	5,182	10.1%	0	Ful	PHP	Non	ngx
288	symfony	5,101	10.1%	0	Ful	PHP	Non	ngx
289	officefloor-rapidoid	2,725	10.0%	0	Ful	Jav	off	woo
290	klein	2,152	10.0%	12	Mcr	Py	Non	Twi
291	webware	1,693	10.0%	0	Mcr	Py	Non	Non
292	swoole	1,092	10.0%	205	Plt	PHP	Non	swo
293	cherrypy-py3	923	10.0%	373	Mcr	Py	Non	Non
294	sanic	728	10.0%	21,858	Mcr	Py	Non	Non
295	cherrypy	595	10.0%	2,025	Mcr	Py	Non	Non
296	smart-socket	105	10.0%	0	Plt	Jav	sma	Non
297	rocket	14	10.0%	158	Ful	Rus	Rus	Hyp
298	falcone	—	Did not complete	—	Mcr	Go	Non	Non
299	goji	—	Did not complete	—	Mcr	Go	Non	Non
300	hexagon-undertow-mongodb	—	Did not complete	—	Mcr	Kot	Svt	Non
301	hexagon-undertow-postgresql	—	Did not complete	—	Mcr	Kot	Svt	Non
302	http4k-sunhttp	—	Did not complete	—	Mcr	Kot	sun	Non
303	kami	—	Did not complete	—	Mcr	Go	Non	Non

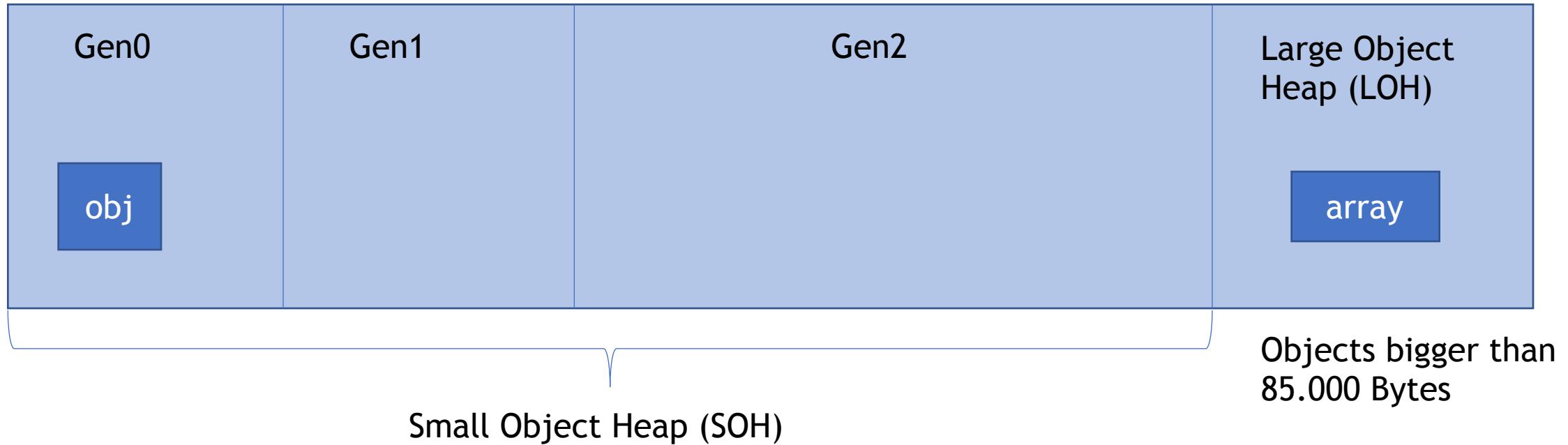
ArrayPool

Gergely Kalapos
 @gregkalapos
 www.kalapos.net

Managed Heap - GC Segments

```
var obj = new Object();
```

```
var array = new int[100_000];
```



Large Object Heap Collection = Full GC (Gen0+Gen1+Gen2+LOH)

Arrays and GC

- Method with array allocation

```
public void MethodCalledOften()
{
    var myArray = new int[15];
    //use myArray...
}
```

- myArray is allocation on the heap -> more work for the GC
- In each call memory is allocation-> GC cleans up -> and in the next round memory is allocated again

Arrays and GC

- Method with a bigger array

```
public void MethodCalledOften()
{
    var myArray = new int[256*1024];
    //use myArray...
}
```

- myArray is allocated on the Large Object Heap -> even more work for the GC

Solution: ArrayPool<T>

- Holds a number of objects ready to use
- .NET Standard 2.1
- NuGet package: System.Buffers - can be used already with .NET Core 1.0, Included in .NET Core 2.1 SDK - Microsoft.NETCore.App
- **ArrayPool<T>.Shared.Rent(int length)**: gives you an array
- **ArrayPool<T>.Shared.Return(T[] array)**: you give the array back to the pool
- <https://github.com/dotnet/corefx/blob/master/src/Common/src/CoreLib/System/Buffers/ArrayPool.cs>

ArrayPool

- Instead of allocation an array each time, use ArrayPool:

```
public void MethodCalledOften()
{
    var arr = ArrayPool<int>.Shared.Rent(256 * 1024);
    Console.WriteLine(arr[0].ToString());
    //use arr.
    ArrayPool<int>.Shared.Return(arr);
}
```

Benchmark Ergebnis

1000 iterations: ArrayPool.Shared.Rent() vs. new Array[]

<https://github.com/greenshadow/ArrayPoolSample>

```
C:\Windows\System32\cmd.exe
Total time: 00:00:52 (52.94)

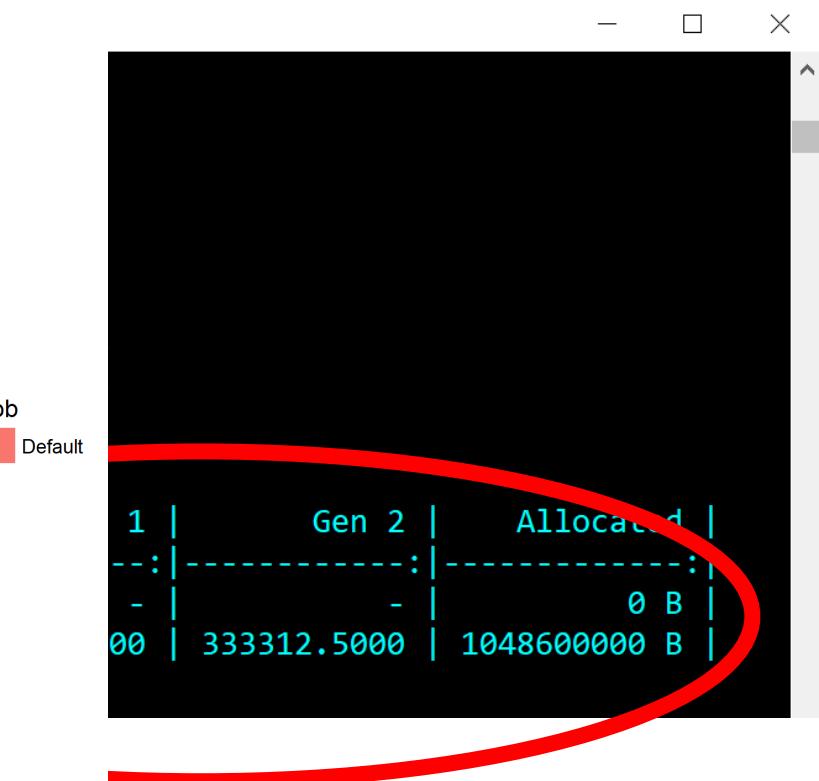
// * Summary *

BenchmarkDotNet=v0.10.14, OS=Windows 10 Pro 10.0.19041
Intel Core i5-5200U CPU 2.20GHz
.NET Core SDK=2.1.300
[Host]     : .NET Core 2.1.300
DefaultJob : .NET Core 2.1.300

    Method | Time | Allocated
ArrayPoolBenchmark | 56 | 0 B
NoPoolBenchmark | 50,912 | 1048600000 B
```



<https://github.com/greenshadow/ArrayPoolSample>



ArrayPool – A few more things

- `ArrayPool<T>.Shared` is the „Shared Pool“
- Max array size in Shared Pool: 2^{20} (1024×1024)
- Calling `Shared.Rent(n)` with $n > 2^{20}$: new array will be allocated -> no pooling
- `ArrayPool<T>.Create(maxArrayLength, maxArraysPerBucket)` creates a new pool. With that you can create pools with arrays bigger than 2^{20}

ArrayPool – Creating a custom pool

- Pooling Arrays with size > 2^{20} ($1024 * 1024$)

```
public ArrayPool<int> MyPool =
    ArrayPool<int>.Create(1024*1024*3, 10);

public void MethodWithCustomPool()
{
    var arr = MyPool.Rent(1024*1024*2);
    //use arr...
    MyPool.Return(arr);
}
```

ArrayPool – Summary

- Very simple API -> goal: avoid GC (especially Gen2 GC)
- Shared Property returns there Shared-Pool (max. 2^{20})
- The Create method created a new Pool
- Where is ArrayPool used? E.g.: Kestrel, ASP.NET Core

Ref return, ref local

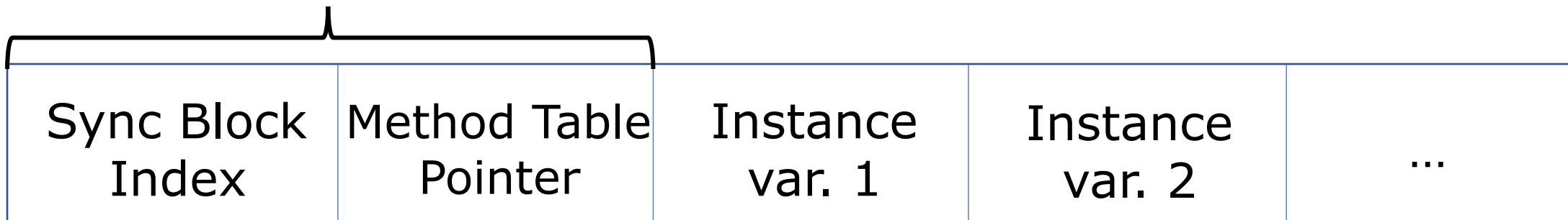
Gergely Kalapos
 @gregkalapos
 www.kalapos.net

Reference types – object header

- Advantage: Polymorphism, Inheritance, lock(){}{}
- Disadvantage: higher memory usage, can be only allocated on the heap

```
class Point
{
    int x, y;
}
```

Per Object Overhead



Value Types on .NET (and .NET Core)

- Contains the value directly
- E.g.: Numerical Typen, Bool, Char, Date, Structs, Enums
- With “=” is the contained value copied
- With “==” are the contained valued compared

Struct Point:

Int: X

Int: Y

```
Point p = new Point();
```

Benchmark: class vs. struct

```
public class  
Point_Class  
{  
    public int x;  
    public int y;  
}
```

VS.

```
public struct  
Point_Struct  
{  
    public int x;  
    public int y;  
}
```

Benchmark: class vs. struct

```
static List<Point_Class> ClassList  
= new List<Point_Class>();  
  
[Benchmark]  
public static void ManyClasses()  
{  
    for (int i = 0; i < 1_000_000; i++)  
    {  
        if(ClassList[i].x == -1)  
        {  
            Debug.WriteLine("-1");  
        }  
    }  
}
```

```
static List<Point_Struct> StructList  
= new List<Point_Struct>();  
  
[Benchmark]  
public static void ManyStructs()  
{  
    for (int i = 0; i < 1_000_000; i++)  
    {  
        if (StructList[i].x == -1)  
        {  
            Debug.WriteLine("-1");  
        }  
    }  
}
```

Benchmark: class vs. struct

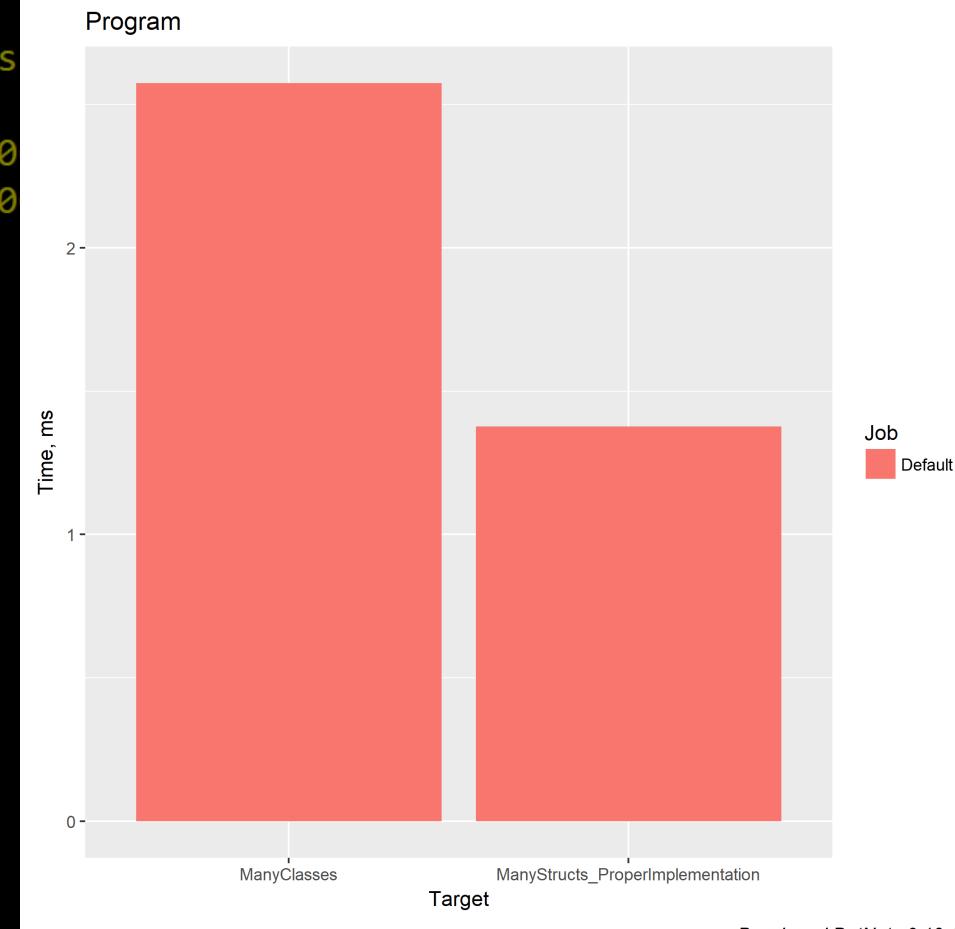
https://github.com/gregkalapos/DWX2018_SampleCode/tree/master/RefAndSpan/StructVsClass

```
BenchmarkDotNet=v0.10.14, OS=Windows 10.0.17134
Intel Core i5-5200U CPU 2.20GHz (Broadwell), 1 CPU, 4 logical and 2 phys
.NET Core SDK=2.1.300
[Host]      : .NET Core 2.1.0 (CoreCLR 4.6.26515.07, CoreFX 4.6.26515.0
DefaultJob : .NET Core 2.1.0 (CoreCLR 4.6.26515.07, CoreFX 4.6.26515.0
```

Method	Mean	Error	StdDev
<hr/>			
ManyClasses	2.574 ms	0.0094 ms	0.0088 ms
ManyStructs_ProperImplementation	1.376 ms	0.0044 ms	0.0041 ms

```
// * Legends *
Mean   : Arithmetic mean of all measurements
Error   : Half of 99.9% confidence interval
StdDev  : Standard deviation of all measurements
1 ms    : 1 Millisecond (0.001 sec)
```

```
// ***** BenchmarkRunner: End *****
// * Artifacts cleanup *
```



Value Types - Nachteile

- No OOP, no *lock(valuetype){}*
- Value Semantic: By default value types are copied (the value inside it is copied)

```
public struct Matrix_Struct
{
    public int i00, i01, i02;
    public int i10, i11, i12;
    public int i20, i21, i22;
}

public static void DoSomethingWithMatrix(Matrix_Struct
matrix_Struct)
{
    //...
}
```

Value Types - Benchmark

```
public class Matrix_Class
{
    public int i00, i01, i02;
    public int i10, i11, i12;
    public int i20, i21, i22;
}

[Benchmark]
public static void BenchmarkMatrixClass()
{
    PrintFirstItem(MatrixClass);
}

public static void
PrintFirstItem(Matrix_Class matrix_Class)
=> Debug.WriteLine(matrix_Class.i00);
```

```
public struct Matrix_Struct
{
    public int i00, i01, i02;
    public int i10, i11, i12;
    public int i20, i21, i22;
}

[Benchmark]
public static void
BenchmarkMatrixStruct()
{
    PrintFirstItem(MatrixStruct);
}

public static void
PrintFirstItem(Matrix_Struct
matrix_Struct)
=> Debug.WriteLine(matrix_Struct.i00);
```

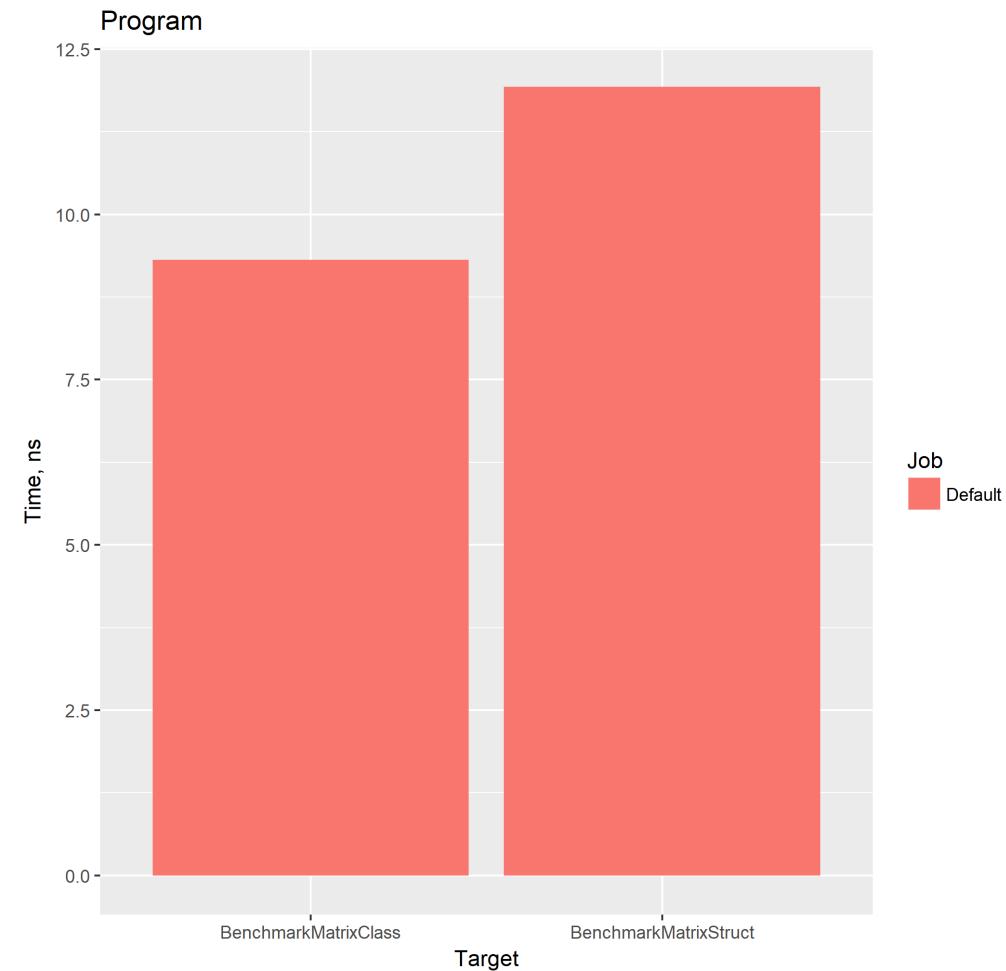
Value Types - Benchmark

- Class vs. Struct als Method Parameter

```
C:\Windows\System32\cmd.exe
.NET Core SDK=2.1.300
[Host]      : .NET Core 2.1.0 (CoreCLR 4.6.26515.07, CoreFX
DefaultJob : .NET Core 2.1.0 (CoreCLR 4.6.26515.07, CoreFX

          Method |      Mean |     Error |    StdDev |
-----:|-----:|-----:|-----:|
BenchmarkMatrixClass | 9.281 ns | 0.0512 ns | 0.0479 ns |
BenchmarkMatrixStruct | 12.248 ns | 0.0131 ns | 0.0116 ns |
```

https://github.com/gregkalapos/ADC19Samples/tree/master/RefAndSpan/StructVsClass_MethodParameter



Value Types

- Solution: pass by reference

```
public struct Matrix_Struct
{
    public int i00, i01, i02;
    public int i10, i11, i12;
    public int i20, i21, i22;
}
```

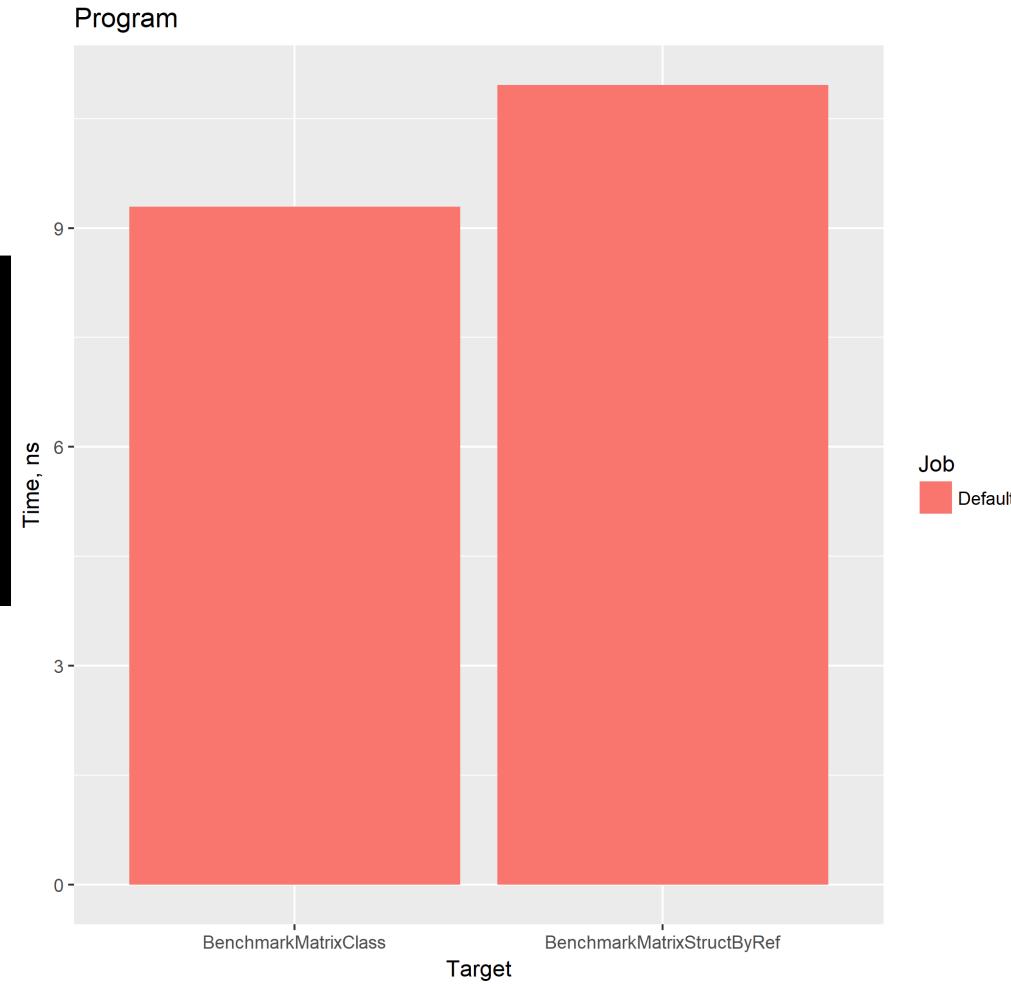
```
public static void DoSomethingWithMatrix(ref Matrix_Struct matrix_Struct)
{
    //...
}
```

Value Types

- Solution: pass by reference

```
C:\Windows\System32\cmd.exe
Method | Mean | Error | StdDev |
-----+-----+-----+-----+
BenchmarkMatrixClass | 9.358 ns | 0.0789 ns | 0.0659 ns |
BenchmarkMatrixStructByRef | 10.912 ns | 0.0637 ns | 0.0564 ns |
...
```

https://github.com/gregkalapos/ADC19Samples/tree/master/RefAndSpan/StructVsClass_MethodParameter_PassByReference



Ref keyword in C# = Managed pointer in the CLR

C#:

```
public static void PrintFirstItem_Struct(Matrix_Struct matrix_Struct)
```

IL:

```
.method public hidebysig static  
    void PrintFirstItem_Struct (  
        valuetype SampleProject.Matrix_Struct matrix_Struct  
    ) cil managed
```

C#:

```
public static void PrintFirstItem_Struct_Ref(ref Matrix_Struct matrix_Struct)
```

IL:

```
.method public hidebysig static  
    void PrintFirstItem_Struct_Ref (  
        valuetype SampleProject.Matrix_Struct& matrix_Struct  
    ) cil managed
```

Ref return

- Since C# 7 we can also use the “ref” keyword with the type

```
public static ref Matrix DoSomethingWithMatrix(ref Matrix  
matrix)  
{  
    //  
    return ref matrix;  
}
```

```
.method public hidebysig static  
valuetype SampleApp.Matrix& DoSomethingWithMatrix (  
    valuetype SampleApp.Matrix& matrix  
) cil managed
```

Ref return – Sample 1

```
public static ref Matrix GetBiggerByReference(ref Matrix matrix1, ref Matrix matrix2)
{
    var sum1 = matrix1.i00 + matrix1.i01 + matrix1.i02 +
        matrix1.i10 + matrix1.i11 + matrix1.i12 +
        matrix1.i20 + matrix1.i21 + matrix1.i22;

    var sum2 = matrix2.i00 + matrix2.i01 + matrix2.i02 +
        matrix2.i10 + matrix2.i11 + matrix2.i12 +
        matrix2.i20 + matrix2.i21 + matrix2.i22;

    if (sum2 > sum1)
        return ref matrix2;
    else
        return ref matrix1;
}
```

Ref return – Benchmark

```
public static ref Matrix GetBiggerByReference(ref Matrix matrix1, ref Matrix matrix2)  
public static Matrix GetBiggerByValue(Matrix matrix1, Matrix matrix2)
```

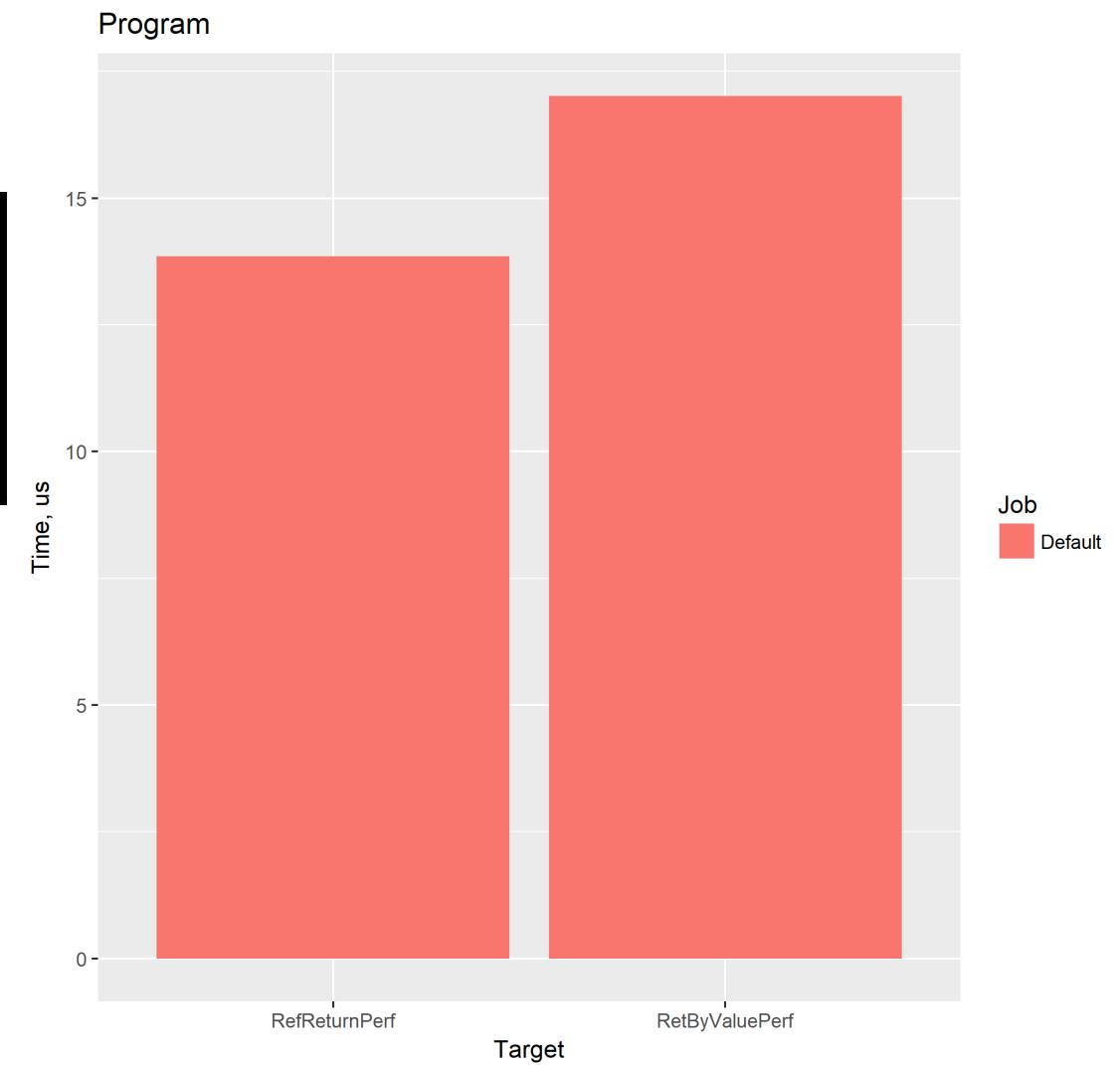
VS.

```
[Benchmark]  
public static void RefReturnPerf()  
{  
    for (int i = 0; i < 1000; i++)  
    {  
        var item1 = matrixList[i].Item1;  
        var item2 = matrixList[i].Item2;  
  
        ref Matrix biggerMatrix = ref GetBiggerByReference(ref item1, ref item2);  
        Debug.WriteLine(biggerMatrix.i00);  
    }  
}
```

Ref return - Benchmark

Method	Mean	Error	StdDev
RefReturnPerf	13.85 us	0.0724 us	0.0677 us
RetByValuePerf	17.01 us	0.0852 us	0.0797 us

<https://github.com/gregkalapos/ADC19Samples/tree/master/RefAndSpan/RefReturnMatrix>



Ref return – Sample 2

```
class MyList
{
    private int[] _items = { 3, 4, 7, 1, 4, 6 };

    public ref int GetRefToSmallestItem()
    {
        var smallest = _items[0];
        int index = 0;
        for (int i = 1; i < _items.Length; i++)
        {
            if (_items[i] < smallest)
            {
                smallest = _items[i];
                index = i;
            }
        }
        return ref _items[index];
    }
}
```

Ref return – Sample 2

```
static void Main(string[] args)
{
    MyList l = new MyList();
    ref int i = ref l.GetRefToSmallestItem();
    i = 42;
}
```

```
.method private hidebysig static
void Main (
    string[] args
) cil managed
{
    .maxstack 2
    .entrypoint
    .locals init (
        [0] class CollectionWithRef.MyList
    ,
        [1] int32&
    )
}
```

```
/* 0x000002E6 0A */ IL_0006: stloc.0
/* 0x000002E7 06 */ IL_0007: ldloc.0
/* 0x000002E8 6F01000006 */ IL_0008:
callvirt instance int32& CollectionWithRef.MyList::GetRefToSmallestItem()
/* 0x000002ED 0B */ IL_000D: stloc.1
/* 0x000002EE 07 */ IL_000E: ldloc.1
/* 0x000002EF 1F2A */ IL_000F: ldc.i4.s 42
```

Ref in C# 7 – what can we return with “ref”?

- Ref Parameter

```
public ref int MethodReturningIntByRef(ref int myInt)
{
    //...do something with 'myInt'
    return ref myInt;
}
```

- Out Parameter

```
public ref int MethodReturningIntByRef(out int myInt)
{
    myInt = 42;
    //...do something with 'myInt'
    return ref myInt;
}
```

Ref in C# 7 – what can we return with “ref”?

- In Parameter (C# 7.2)

```
public ref readonly int MethodReturningIntByRef(in int  
myInt)  
{  
    //...do something with 'myInt'  
    return ref myInt;  
}
```

- Instance variables of a type

```
class MyClass  
{  
    private int myInt;  
  
    public ref int GetRefToMyInt() => ref myInt;  
}
```

Ref in C# 7 – what CAN'T we return with “ref”

- Lokale variables

```
public ref int MyMethod()  
{  
    int i = 42;  
    return ref i;  
}
```

- ,null'

```
public ref object MyMethod()  
{  
    return ref null;  
}
```

- ,this'

```
public ref object MyMethod()  
{  
    return ref this;  
}
```

Ref in C# 7 – what CAN'T we return with “ref”

- async methods

```
public async ref Task<int> MyMethod()  
{  
    //let's do something async:  
    await Task.Delay(500);  
    return ref 42;  
}
```

- consts

```
const int I = 42;  
public ref int MyMethod()  
{  
    return ref I;  
}
```

- ,enum'

```
public enum Day { Monday, Friday }  
public ref Day MyMethod()  
{  
    var day = Day.Monday;  
    return ref day;  
}
```

Ref return – Where is it used?

- ASP.NET Core
- Span<T>

Span<T>

Gergely Kalapos
 @gregkalapos
 www.kalapos.net

Allocation in .NET

- Managed memory

```
byte[ ] myArray = new byte[100];
```

- Unmanaged memory

```
IntPtr myArray =
Marshal.AllocHGlobal(100);
//...use myArray
Marshal.FreeHGlobal(myArray);
```

- Stack memory

```
byte* myArray = stackalloc byte[100];
```

Accepting handles to different types of memory in an API

- „High Performance“ libraries usually use managed and unmanaged (e.g. stack) memory as well (E.g.: ASP.NET Core/Kestrel)
- But it's very hard to deal with this

```
public void ProcessByteArray(byte[] array)
```

```
public void ProcessByteArray(void* array)
```

```
public void ProcessByteArray(byte[] array, int offset)
```

```
public void ProcessByteArray(void* array, int offset)
```

Span<T>

- Represents a contiguous region of memory
- An abstraction over every type of memory that we can allocate in .NET
- .NET Standard 1.0

```
public void ProcessByteArray(Span<byte> array)
```

Span<T> - wrapping memory from the managed heap

```
byte[ ] myArray = new byte[100];
Span<byte> myArraySpan = new Span<byte>(myArray);
ProcessByteArray(myArraySpan);
```

Span<T> - wrapping unmanaged memory

```
IntPtr myArray = Marshal.AllocHGlobal(100);
Span<byte> nativeSpan;
unsafe
{
    nativeSpan = new Span<byte>(myArray.ToPointer(), 100);
}
ProcessByteArray(nativeSpan);

Marshal.FreeHGlobal(myArray);
```

Span<T> - wrapping stack memory

Since C# 7.2:

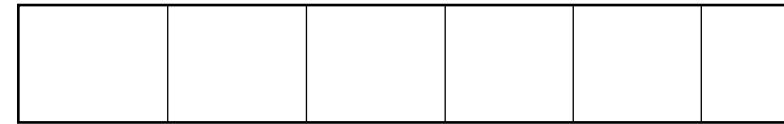
```
Span<byte> stackSpan;  
unsafe  
{  
    Span<byte> stackSpan = stackalloc byte[100];  
    byte* myArray = stackalloc byte[100];  
    stackSpan = new Span<byte>(myArray, 100);  
}  
ProcessByteArray(stackSpan);
```

Span<T>

Span<T>

Start
Offset
Size

Arbitrary memory (can be managed,
unmanaged, stack allocated)



Span<T> API

- Span erstellen:

```
public Span(T[] array);  
  
public Span(void* pointer, int length);  
  
public Span(T[] array, int start, int length)
```

- Indexing:

```
public ref T this[int index] { get; }
```

- Slicing:

```
public Span<T> Slice(int start);  
  
public Span<T> Slice(int start, int length);
```

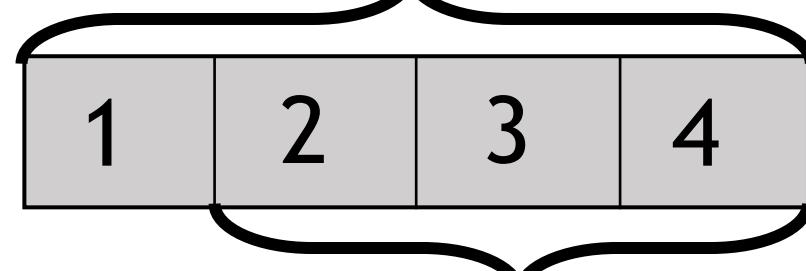
(*Not a complete list of all methods!)

Span<T>.Slice(start, length)

```
byte[ ] array = new byte[ ] { 1, 2, 3, 4 };
```

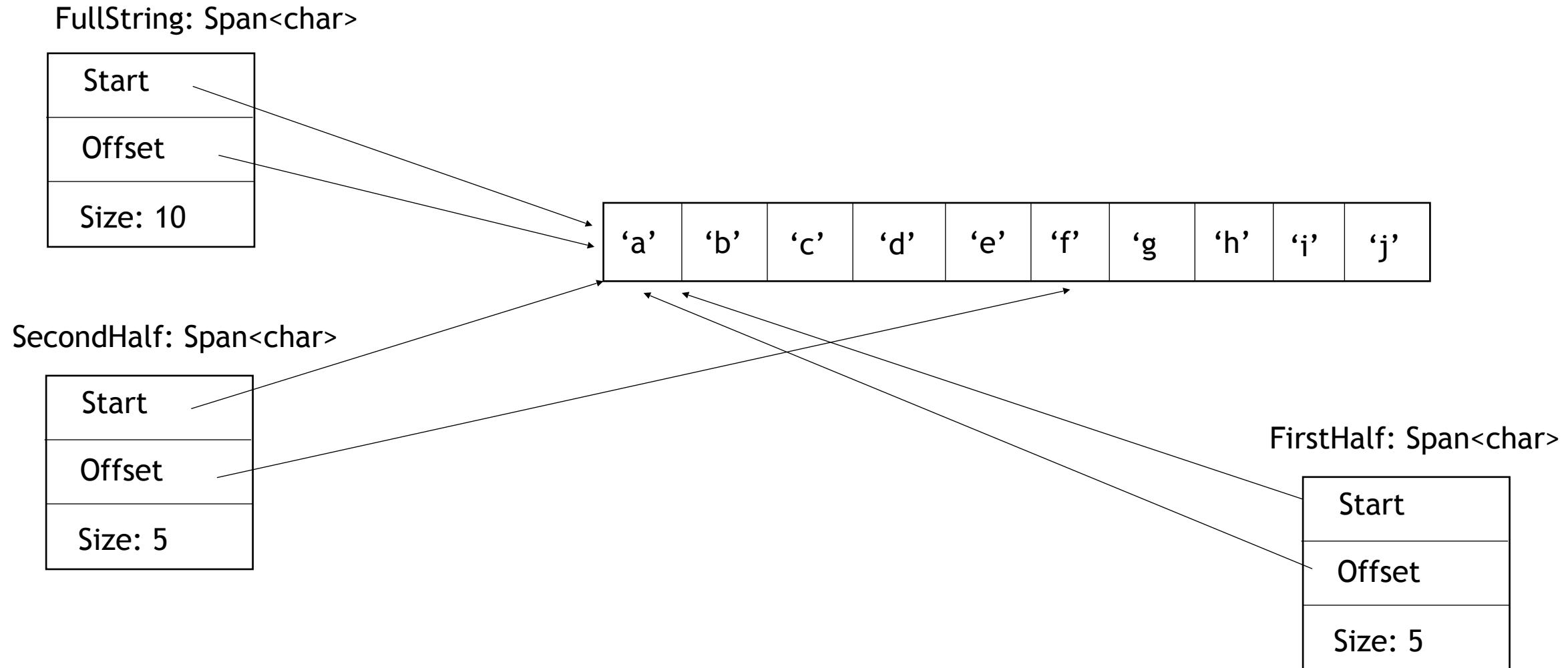
```
Span<byte> span = new Span<byte>(array);
```

```
Span<byte> newSpan = span.Slice(1, 3);
```



This does not allocate at all

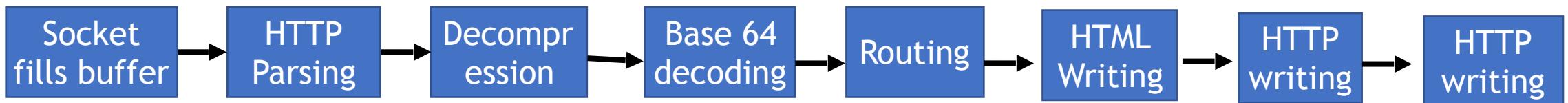
Span<T> - Performance Advantage



Span use cases

- String parsing
- Encoding
- Formatting
- Compression/Decompression
- Data Pipelines
- JSON Serialization/Deserialization (System.Text.Json-in .NET Core 3.0)

HTTP server:



Span<T> - String parsing

```
public static void WorkWithSpan()
{
    ReadOnlySpan<char> str = @"Request URL: http://kalapos.net/
        Request Method:GET
        Status Code:200 OK
        Remote Address:81.10.250.159:80
        Referrer Policy:no-referrer-when-downgrade
        Accept:text/html,application/xhtml+xml,application/xml;q=0.9,
        Accept-Encoding:gzip, deflate
        Accept-Language:de-DE,de;q=0.8,en-US;q=0.6,en;q=0.4,hu;q=0.2
        Connection:keep-alive
        Cookie:_cfduid=d37ec605870fbb75a1b5b9616bf161ed01507890791;
            _ga=GA1.2.460922426.1476689834; _gid=GA1.2.1163677089.1508317942
        Host:kalapos.net
        Upgrade-Insecure-Requests:1
        User-Agent:Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36".AsSpan();
    ParseStringSpan(str);
}
```

Span<T> - String parsing

```
public static void  
ParseStringSpan(ReadOnlySpan<char> str)  
{  
    var url = str.Slice(13, 19);  
    var method = str.Slice(58, 3);  
    Int32.TryParse(str.Slice(84, 3), out int  
statusCode);  
    var remoteAddr = str.Slice(116, 14);  
    var referrerPolicy = str.Slice(157, 26);  
    var accept = str.Slice(201, 85);  
    var acceptEncoding = str.Slice(313, 13);  
    var acceptLanguage = str.Slice(353, 44);  
    var connection = str.Slice(419, 10);  
    var cookie = str.Slice(447, 118);  
    var host = str.Slice(581, 11);  
    Int32.TryParse(str.Slice(629, 1), out  
int upgradeInsecureRequest);  
    var userAgent = str.Slice(652, 115);  
}
```

```
public static void ParseStringClassic(String  
str)  
{  
    var url = str.Substring(13, 19);  
    var method = str.Substring(58, 3);  
    Int32.TryParse(str.Substring(84, 3), out int  
statusCode);  
    var remoteAddr = str.Substring(116, 14);  
    var referrerPolicy = str.Substring(157, 26);  
    var accept = str.Substring(201, 85);  
    var acceptEncoding = str.Substring(313, 13);  
    var acceptLanguage = str.Substring(353, 44);  
    var connection = str.Substring(419, 10);  
    var cookie = str.Substring(447, 118);  
    var host = str.Substring(581, 11);  
    Int32.TryParse(str.Substring(629, 1), out int  
upgradeInsecureRequest);  
    var userAgent = str.Substring(652, 115);  
}
```

Span<T> - Benchmark

```
C:\Windows\System32\cmd.exe
```

```
// * Summary *
```

```
BenchmarkDotNet=v0.10.14, OS=Windows 10.0.17134
```

```
Intel Core i5-5200U CPU 2.20GHz (Broadwell), 1 CPU, 4 logical and 2 physical cores
```

```
.NET Core SDK=2.1.300
```

```
[Host]     : .NET Core 2.1.0 (CoreCLR 4.6.26515.07, CoreFX 4.6.26515.06)
```

```
DefaultJob : .NET Core 2.1.0 (CoreCLR 4.6.26515.07, CoreFX 4.6.26515.06)
```

Method	Mean	Error	StdDev	Gen 0	Allocated
WorkWithString	509.3 ns	1.8835 ns	1.7619 ns	0.8183	1288 B
WorkWithSpan	222.5 ns	0.6024 ns	0.5340 ns	-	0 B

```
// * Legends *
```

```
Mean      : Arithmetic mean of all measurements
```

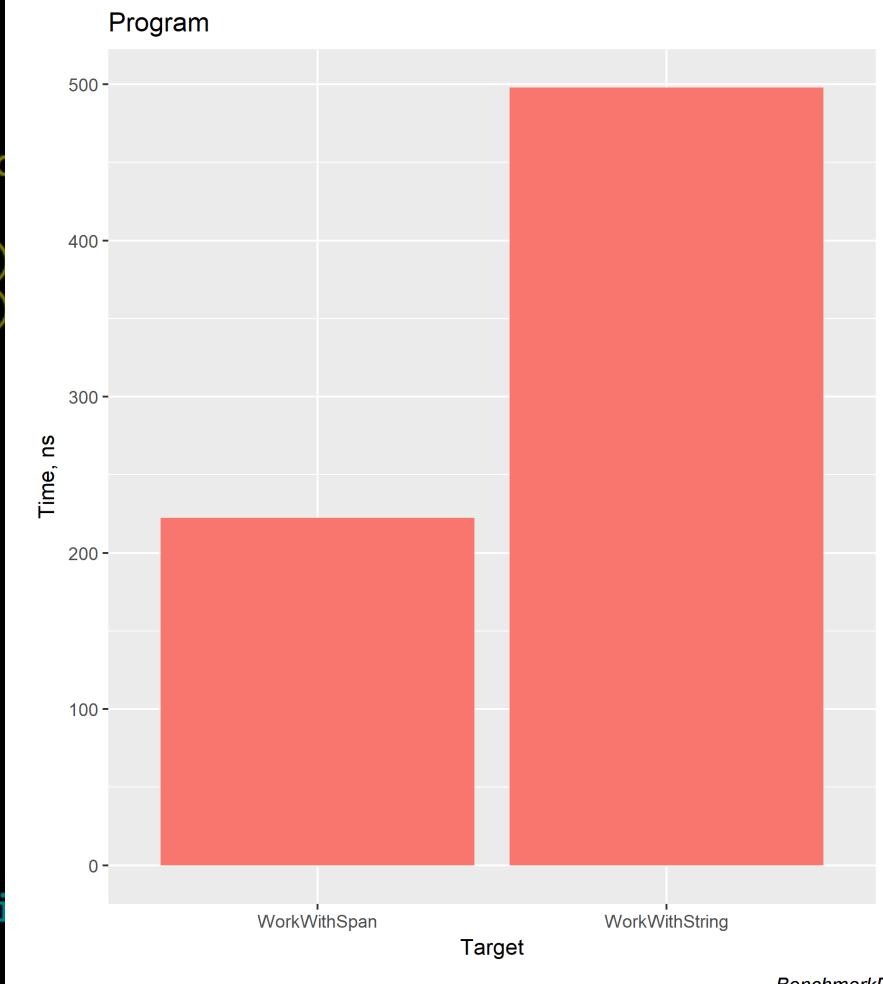
```
Error     : Half of 99.9% confidence interval
```

```
StdDev    : Standard deviation of all measurements
```

```
Gen 0     : GC Generation 0 collects per 1k Operations
```

```
Allocated : Allocated memory per single operation (managed only, includes
```

```
1 ns      : 1 Nanosecond (0.000000001 sec)
```



<https://github.com/gregkalapos/ADC19Samples/tree/master/RefAndSpan/StringParsingWithSpan>

.NET Standard 2.1

- ~3K new API
- Lots of them are Span<T> or ReadonlySpan<T> overloads

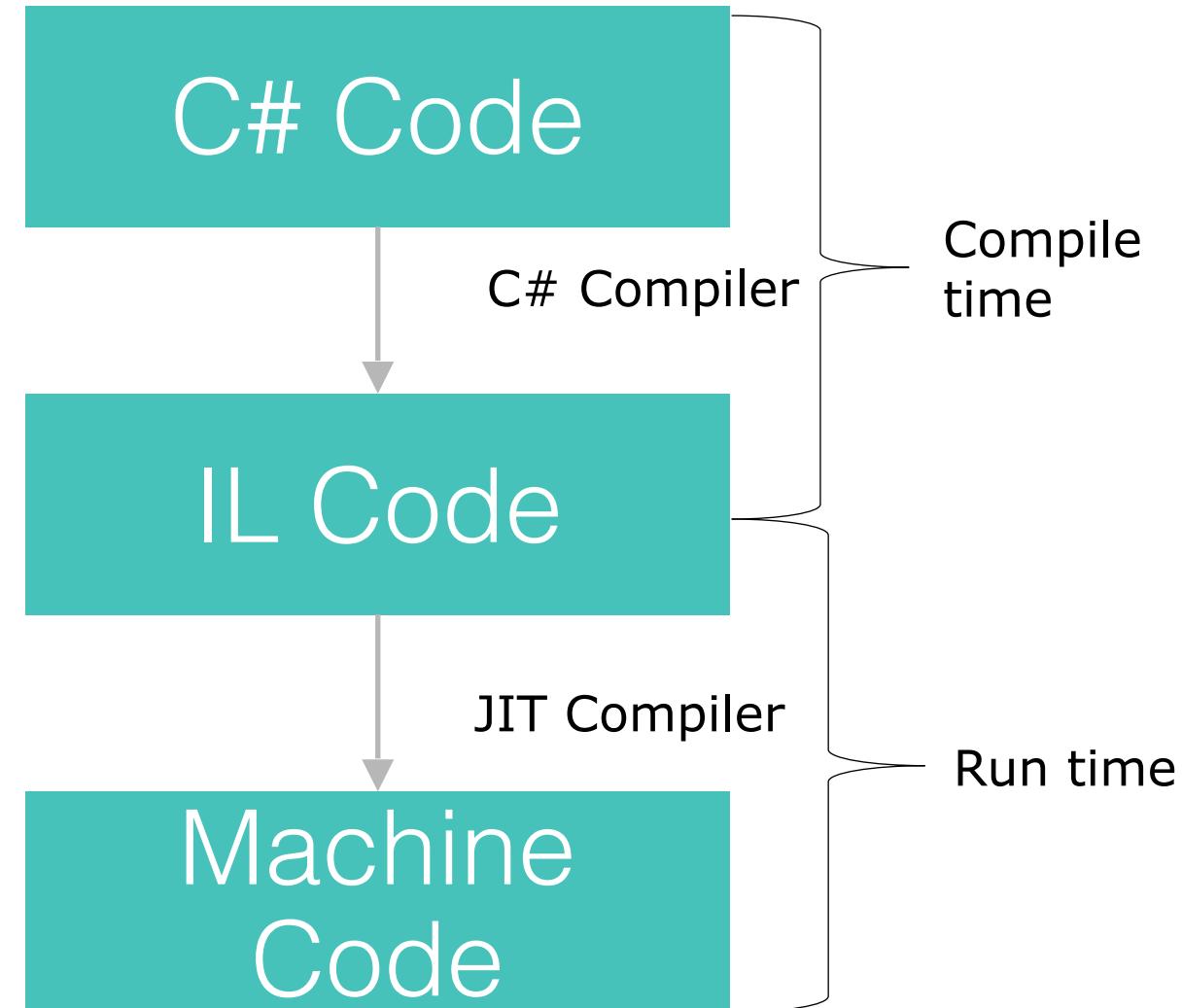
```
public readonly struct Int32 : IComparable, IComparable<int>, IConvertible, IEquatable<int>, IFormattable {  
    public const int MaxValue = 2147483647;  
    public const int MinValue = -2147483648;  
    public Int32 CompareTo(Int32 value);  
    public Int32 CompareTo(object value);  
    public bool Equals(Int32 obj);  
    public override bool Equals(object obj);  
    public override Int32 GetHashCode();  
    public TypeCode GetTypeCode();  
    + public static Int32 Parse(ReadOnlySpan<char> s, NumberStyles style = NumberStyles.Integer, IFormatProvider provider = null);  
    public static Int32 Parse(string s);  
    public static Int32 Parse(string s, NumberStyles style);  
    public static Int32 Parse(string s, NumberStyles style, IFormatProvider provider);  
    public static Int32 Parse(string s, IFormatProvider provider);  
    bool System.IConvertible.ToBoolean(IFormatProvider provider);  
    byte System.IConvertible.ToByte(IFormatProvider provider);  
    char System.IConvertible.ToChar(IFormatProvider provider);  
    DateTime System.IConvertible.ToDateTime(IFormatProvider provider);  
    decimal System.IConvertible.ToDecimal(IFormatProvider provider);  
    double System.IConvertible.ToDouble(IFormatProvider provider);  
    short System.IConvertible.ToInt16(IFormatProvider provider);  
    Int32 System.IConvertible.ToInt32(IFormatProvider provider);  
    long System.IConvertible.ToInt64(IFormatProvider provider);  
    sbyte System.IConvertible.ToSByte(IFormatProvider provider);  
    float System.IConvertible.ToSingle(IFormatProvider provider);  
    object System.IConvertible.ToType(Type type, IFormatProvider provider);  
    ushort System.IConvertible.ToUInt16(IFormatProvider provider);  
    uint System.IConvertible.ToUInt32(IFormatProvider provider);  
    ulong System.IConvertible.ToUInt64(IFormatProvider provider);  
    public override string ToString();  
    public string ToString(IFormatProvider provider);  
    public string ToString(string format);  
    public string ToString(string format, IFormatProvider provider);  
    + public bool TryFormat(Span<char> destination, out Int32 charsWritten, ReadOnlySpan<char> format = default(ReadOnlySpan<char>), IFormatProvider provider = null);  
    + public static bool TryParse(ReadOnlySpan<char> s, NumberStyles style, IFormatProvider provider, out Int32 result);  
    + public static bool TryParse(ReadOnlySpan<char> s, out Int32 result);  
    public static bool TryParse(string s, NumberStyles style, IFormatProvider provider, out Int32 result);  
    public static bool TryParse(string s, out Int32 result);  
}
```

SIMD

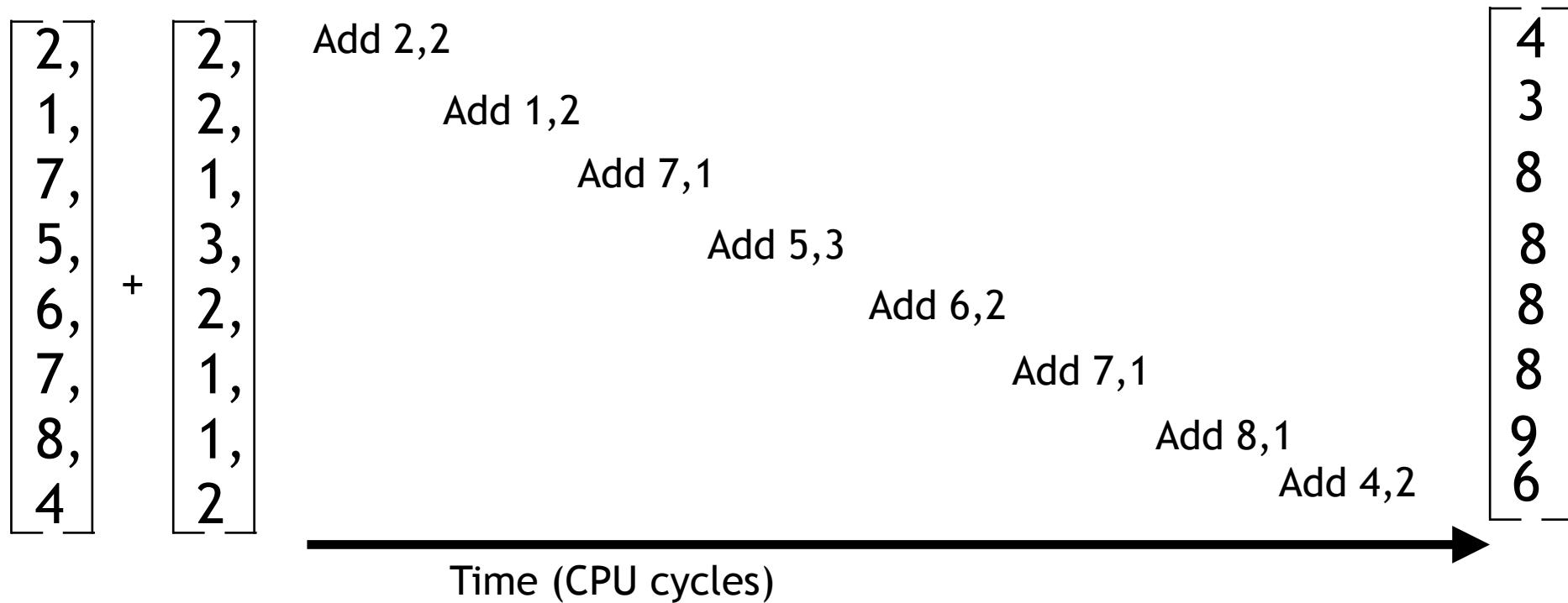
- Part of .NET Standard 2.1
- Single instruction, multiple data
- Parallelism
- Perform the same operation on multiple data points simultaneously

SIMD - Classic Assembly (no SIMD)

- add (adds 2 integers, result: 1 integer)
`add eax, ebx`
- mov:
`mov ebx, count`
- Take 1 or 2 values and produces single result



SIMD - Classic Assembly (no SIMD)



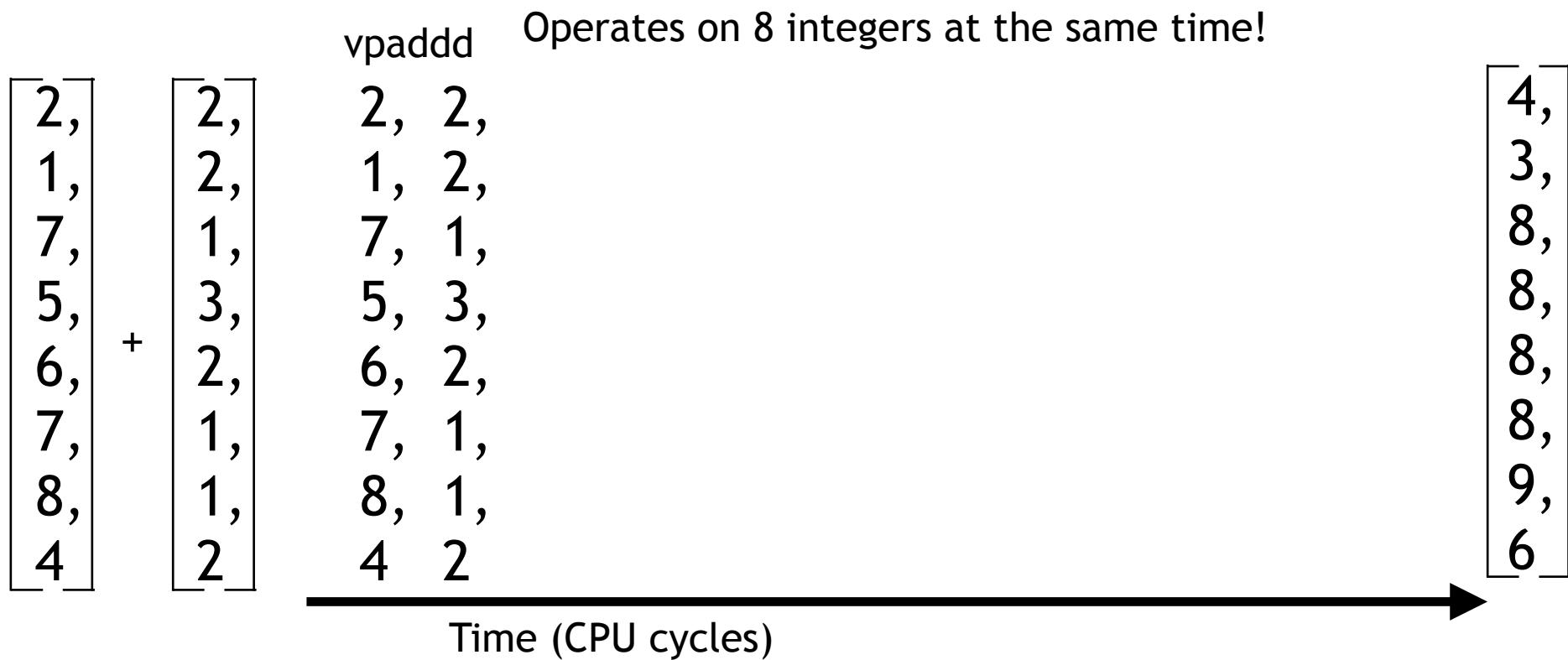
SIMD

- Single instruction, multiple data
- Those instructions can have multiple operands
- vpaddd: add packed doubleword integers

SIMD in .NET

- If the CPU supports SIMD, the JIT compiler emits SIMD instructions
- System.Numerics namespace

SIMD - add 2 arrays with `vpadd`



Add 2 array, no SIMD

```
public static double[] AddArrays_Simple(double[] v1, double[] v2)
{
    double[] retVal = new double[v1.Length];

    for (int i = 0; i < v1.Length; i++)
    {
        retVal[i] = v1[i] + v2[i];
    }

    return retVal;
}
```

Add 2 array, no SIMD

```
setne    cl
movzx   ecx,cl
and     ecx,1
test    ecx,1
je      M01_L01
cmp     dword ptr [rsi+8],ebx
setge   cl
movzx   ecx,cl
cmp     dword ptr [rax+8],ebx
setge   r8b
movzx   r8d,r8b
test    r8d,ecx
je      M01_L01
                                         retVal[i] = v1[i] + v2[i];
                                         ^^^^^^^^^^^^^^^^^^^^^^^^^^
IL_000d: ldloc.0
IL_000e: ldloc.1
IL_000f: ldarg.0
IL_0010: ldloc.1
IL_0011: ldelem.i4
IL_0012: ldarg.1
IL_0013: ldloc.1
IL_0014: ldelem.i4
IL_0015: add
IL_0016: stelem.i4
M01_L00
        movsxrd rcx,edx
        mov     r8d,dword ptr [rdi+rcx*4+10h]
        add     r8d,dword ptr [rsi+rcx*4+10h]
        mov     dword ptr [rax+rcx*4+10h],rou
                                         for (int i = 0; i < v1.Length; i++)
                                         ^
IL_0017: ldloc.1
IL_0018: ldc.i4.1
IL_0019: add
IL_001a: stloc.1
```

Add 2 array, with SIMD

```
public static double[] AddArrays_Vector(double[] v1, double[] v2)
{
    double[] retVal = new double[v1.Length];
    var vectSize = Vector<double>.Count; 
    int i = 0;
    for (i = 0; i <= v1.Length - vectSize; i += vectSize)
    {
        var va = new Vector<double>(v1, i);
        var vb = new Vector<double>(v2, i);
        var vc = va + vb;
        vc.CopyTo(retVal, i);
    }
    if (i != v1.Length)
    {
        for (int j = i; j < v1.Length; j++)
        {
            retVal[j] = v1[j] + v2[j];
        }
    }
    return retVal;
}
```

Add 2 array, with SIMD

```
--      -----
    lea    r10d,[rdx+7]
    cmp    r10d,ebx
    jae    00007ffc`d9351179
    vmovupd ymm0,ymmword ptr [rdi+rdx*4+10h]
    var vb = new Vector(v2, i);
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

    IL_001e: ldloca.s V_4
    IL_0020: ldarg.1
    IL_0021: ldloc.2
    IL_0022: call System.Void System.Numerics.Vector`1::ctor(!0[],System.Int32)
    cmp    edx,r8d
    jae    00007ffc`d9351179
    cmp    r10d,r8d
    jae    00007ffc`d9351179
    vmovupd ymm1,ymmword ptr [rsi+rdx*4+10h]
    var vc = va + vb;
    ^^^^^^^^^^^^^^

    IL_0027: ldloc.3
    IL_0028: ldloc.s V_4
    IL_002a: call System.Numerics.Vector`1 System.Numerics.Vector`1::op_Addition(System.Numerics.Vector`1,System.Numerics.Vector`1)
    IL_002c: ldloc.4
    vpadddd ymm0,ymm0,ymm1
    vc.CopyTo(retval, i);
    ^^^^^^^^^^^^^^

    IL_0031: ldloca.s V_5
    IL_0033: ldloc.0
    IL_0034: ldloc.2
    IL_0035: call System.Void System.Numerics.Vector`1::CopyTo(!0[],System.Int32)
    cmp    edx,r9d
    jae    00007ffc`d935117e
    cmp    r10d,r9d
    jae    00007ffc`d935117e
    vmovupd ymmword ptr [rax+rdx*4+10h],ymm0
    for (i = 0; i < v1.Length - vecSize; i += vecSize)
    ^^^^^^^^^^

    IL_003a: ldloc.2
    IL_003b: ldloc.1
```

SIMD vs. Simple loop

```
BenchmarkDotNet=v0.10.13, OS=Windows 10 Redstone 3 [1709, Fall Creators Update] (10.0.16299.192)
Intel Core i5-5200U CPU 2.20GHz (Broadwell), 1 CPU, 4 logical cores and 2 physical cores
Frequency=2143475 Hz, Resolution=466.5321 ns, Timer=TSC
.NET Core SDK=2.1.4
[Host]      : .NET Core 2.0.5 (CoreCLR 4.6.26020.03, CoreFX 4.6.26018.01), 64bit RyuJIT
DefaultJob : .NET Core 2.0.5 (CoreCLR 4.6.26020.03, CoreFX 4.6.26018.01), 64bit RyuJIT
```

Method	Mean	Error	StdDev	Median
AddArrays_Vector_Benchmark	89.25 ns	0.6531 ns	0.5790 ns	89.42 ns
AddArrays_Simple_Benchmark	168.01 ns	8.7787 ns	25.3286 ns	153.28 ns

```
// * Hints *
Outliers
Program.AddArrays_Vector_Benchmark: Default -> 1 outlier was removed
Program.AddArrays_Simple_Benchmark: Default -> 4 outliers were removed
```

```
// * Legends *
Mean   : Arithmetic mean of all measurements
Error   : Half of 99.9% confidence interval
StdDev  : Standard deviation of all measurements
Median  : Value separating the higher half of all measurements (50th percentile)
1 ns    : 1 Nanosecond (0.000000001 sec)
```

Resources

- State of the .NET Performance - Adam Sitnik (<http://adamsitnik.com>) <https://www.youtube.com/watch?v=CSPSvBeqJ9c>
- Vladimir Sadov - <http://mustoverride.com>
- BenchmarkDotNet: <https://benchmarkdotnet.org/>
- Ref return and ref locals (Microsoft): <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/ref-returns>

Sample code + Slides

- Slides and Sample code: <https://github.com/gregkalapos/ADC19Samples>
- Twitter: @gregkalapos
(<https://twitter.com/gregkalapos>)