INTEL® HARDWARE INTRINSICS IN .NET CORE

Han Lee, Intel Corporation

han.lee@intel.com

Notices and Disclaimers

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request. No product or component can be absolutely secure.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel, the Intel logo, and other Intel product and solution names in this presentation are trademarks of Intel

*Other names and brands may be claimed as the property of others

What Do These Have in Common?

Domain	Example
Image processing	Color extraction
High performance computing (HPC)	Matrix multiplication
Data processing	Hamming code
Text processing	UTF-8 conversion
Data structures	Bit array
Machine learning	Classification

For performance sensitive code, consider using Intel® hardware intrinsics

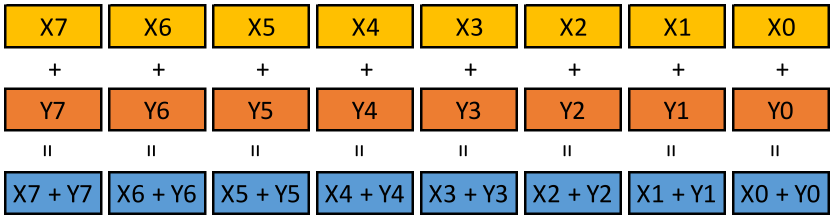
Objectives

- (Very brief) Intro to SIMD
- Design Motivation
- Hardware Intrinsics
- Call to Action

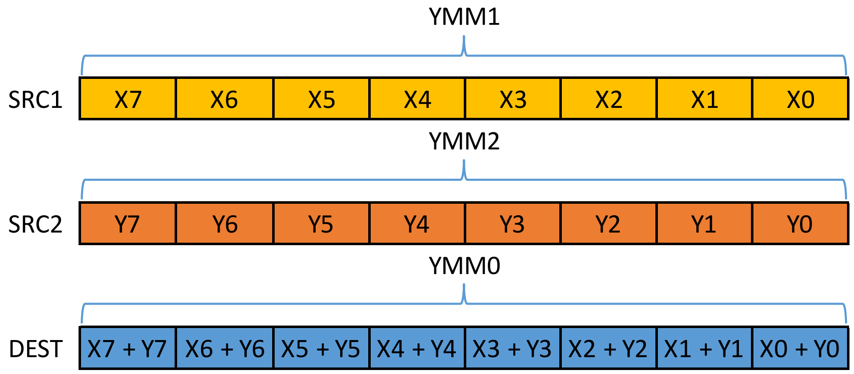
Single Instruction, Multiple Data (SIMD)

Perform same operation on multiple data using a single instruction

- Intel Advanced Vector Extensions 2 (Intel® AVX2) supports 256-bit SIMD instructions
  - Add eight 32-bit integers using one *vpaddd* instruction



8x ADD R32, R32



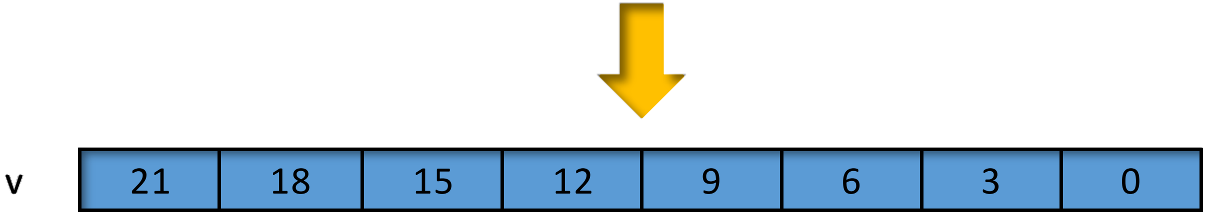
`VPADDD YMM0, YMM1, YMM2`

Single Instruction, Multiple Data (SIMD)

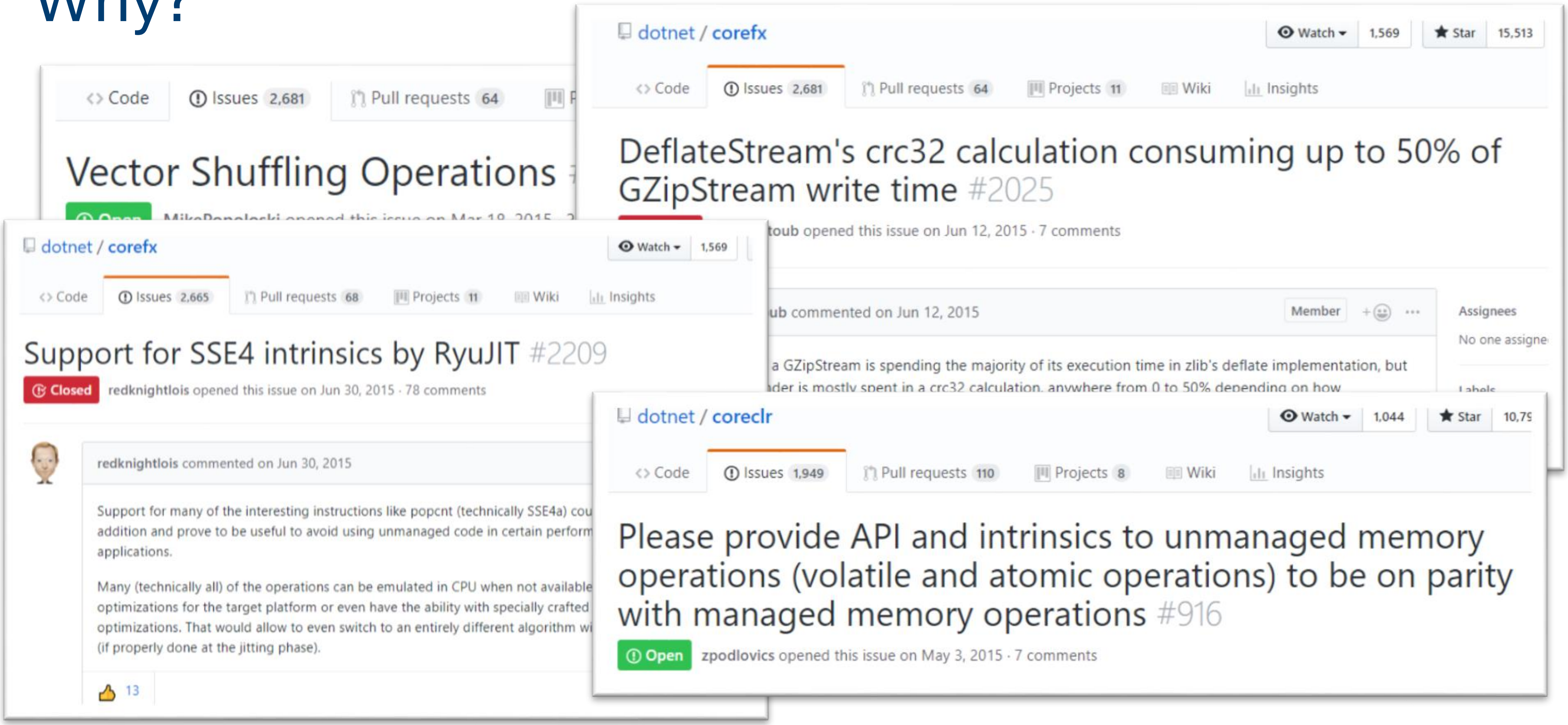
C# abstraction is System.Numerics.Vector*

- *System.Numerics.Vector<T>*
  - On Intel AVX2 system, *Vector<int>.Count* is 8
  - On Intel Streaming SIMD Extensions 2 (Intel SSE2) system, *Vector<int>.Count is 4*

```
Vector<int> v = New Vector<int>(0, 1, 2, 3, 4, 5, 6, 7)
              + New Vector<int>(0, 2, 4, 6, 8, 10, 12, 14)
```



Why?



*https://github.com/dotnet/corefx/issues/1168, https://github.com/dotnet/corefx/issues/2209, https://github.com/dotnet/corefx/issues/2025, https://github.com/dotnet/coreclr/issues/916

Hardware Intrinsics (a.k.a. Intrinsic functions / Platform Dependent Intrinsics)

Special functions that directly map to hardware instructions

- Useful when
  - Algorithms can better be mapped to underlying hardware
  - Maximum control over code generation is desired
- Mainstream C/C++ compilers have intrinsic functions
  - Field tested and proven to be useful
  - Provides design guidelines
- Both SIMD and non-SIMD

Intel Hardware Intrinsics

- APIs originally designed and proposed by Intel
- Major enhancements with the help of .NET Core community and Microsoft
- Implementation by Intel, Microsoft and .NET Core community
- Namespace *System.Runtime.Intrinsics* contains platform-agnostic types & functions
  - E.g., *Vector256<int>, Vector256<int>.GetLower()*
- New namespace System.Runtime.Intrinsics.X86 contains 15 top-level classes corresponding to different Instruction Set Architectures (ISAs)
  - *AES, AVX, AVX2, BMI1, BMI2, FMA, LZCNT, PCLMULQDQ, POPCNT, SSE, SSE2, SSE3, SSE4.1, SSE4.2, and SSSE3*
- Available as an experimental features since .NET Core 2.1
- Available in .NET Core 3.0 Preview

Intel Hardware Intrinsics Design

- ISA class contains *IsSupported* boolean property to check hardware support
- ISA class contains intrinsic methods corresponding to underlying instructions
- Methods closely mirror C/C++ intrinsic function

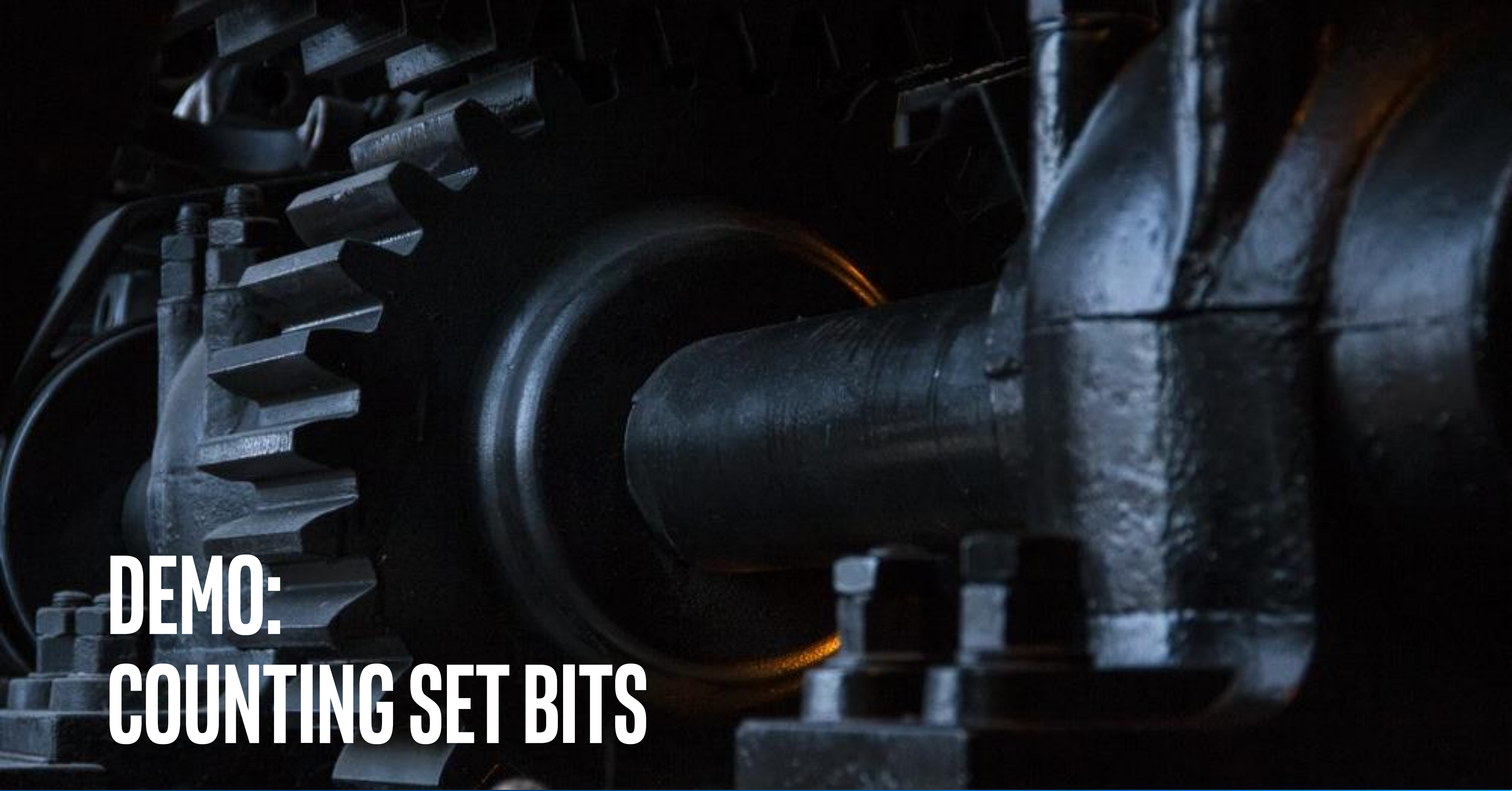
```
/// <summary>
/// int _mm_popcnt_u32 (unsigned int a)
///   POPCNT reg, reg/m32
/// </summary>
public static uint PopCount(uint value);
```

- Majority of Intel hardware intrinsics operate over *Vector128/256<T>*

```
public static Vector256<int> Add(Vector256<int> left, Vector256<int> right)
```

- Unsafe methods for operating over pointers

```
public static unsafe Vector256<sbyte> LoadVector256(sbyte* address)
```



DEMO: COUNTING SET BITS

Using Intel Hardware Intrinsics in .NET Core

```
using System.Runtime.Intrinsics.X86;
using System.Runtime.Intrinsics;

static int[] Func(int[] data)
{
    if (Avx2.IsSupported)
    {
        // AVX2 implementation
    }
    else if (Sse.IsSupported)
    {
        // SSE implementation
    }
    else
    {
        // Software scalar implementation
    }
}
```

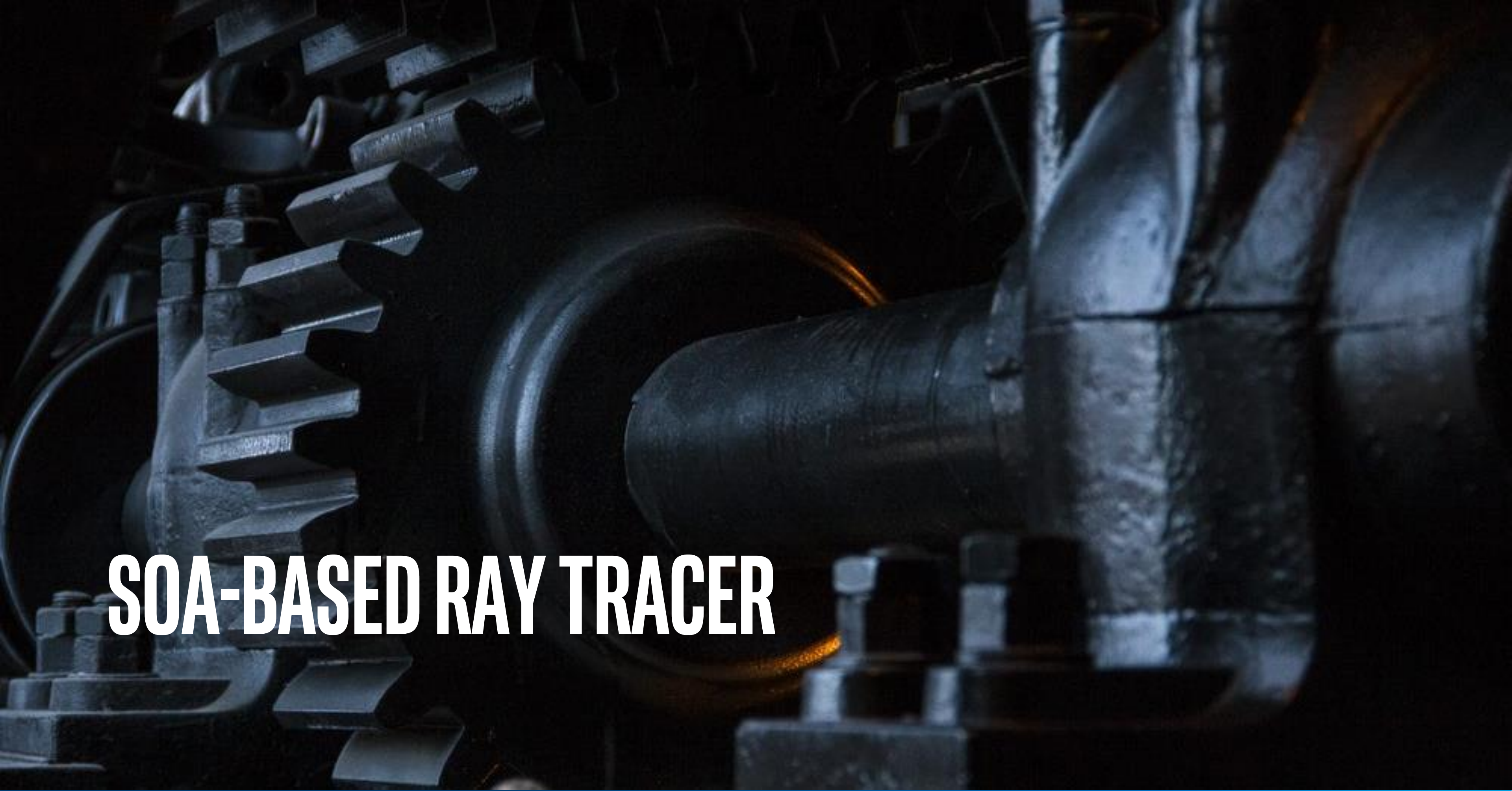
Import the namespace to use Intel HW intrinsic

Import the namespace to use Vector128/256<T> as needed

Check hardware ISA support before using any HW intrinsic

The checks will be optimized away by the Just-In-Time compiler

NOTE: Calling HW intrinsic on unsupported hardware will result in *System.PlatformNotSupportedException*



SOA-BASED RAY TRACER

How to Vectorize a Ray-tracer?

Ray-tracer based on 3D vector computation

- Most of the underlying structures can be abstracted to 3D vectors
  - Position/direction (x, y, z) in 3D space, color (R, G, B)

Vectorize the underlying 3D vector computation

- AoS (Array of Structure)
- SoA (Structure of Array)

AoS vectorization

Example: Add 3D vectors of float on AVX-capable machine

Scalar

x_1	y_1	z_1	+	x_2	y_2	z_2

$$x = x_1 + x_2$$
$$y = y_1 + y_2$$
$$z = z_1 + z_2$$

AoS

x_1	y_1	z_1	-	+	x_2	y_2	z_2	-

XMM1 XMM2

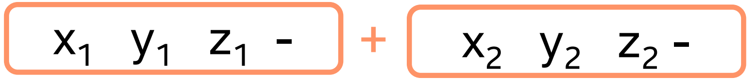
vaddps xmm, xmm1, xmm2

Cannot leverage wider SIMD architecture!

SoA vectorization

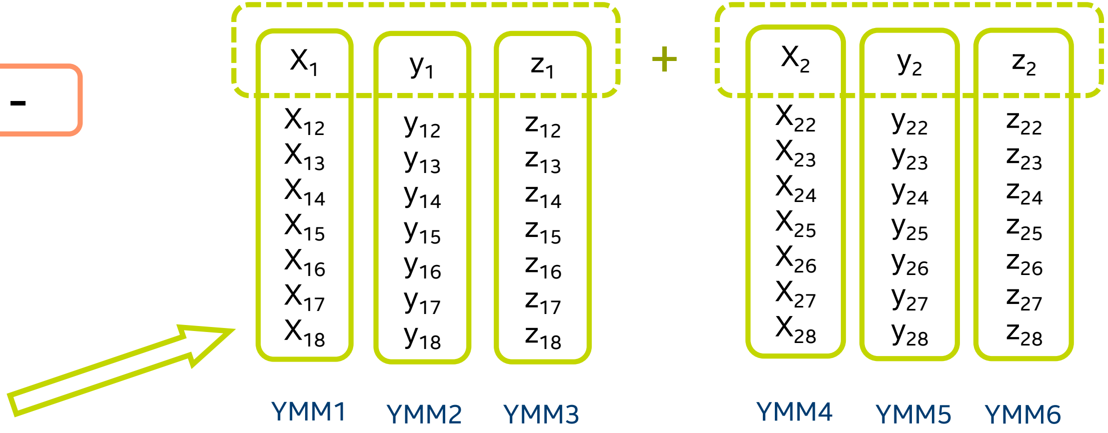
Example: Add 3D vectors of float on AVX-capable machine

AoS



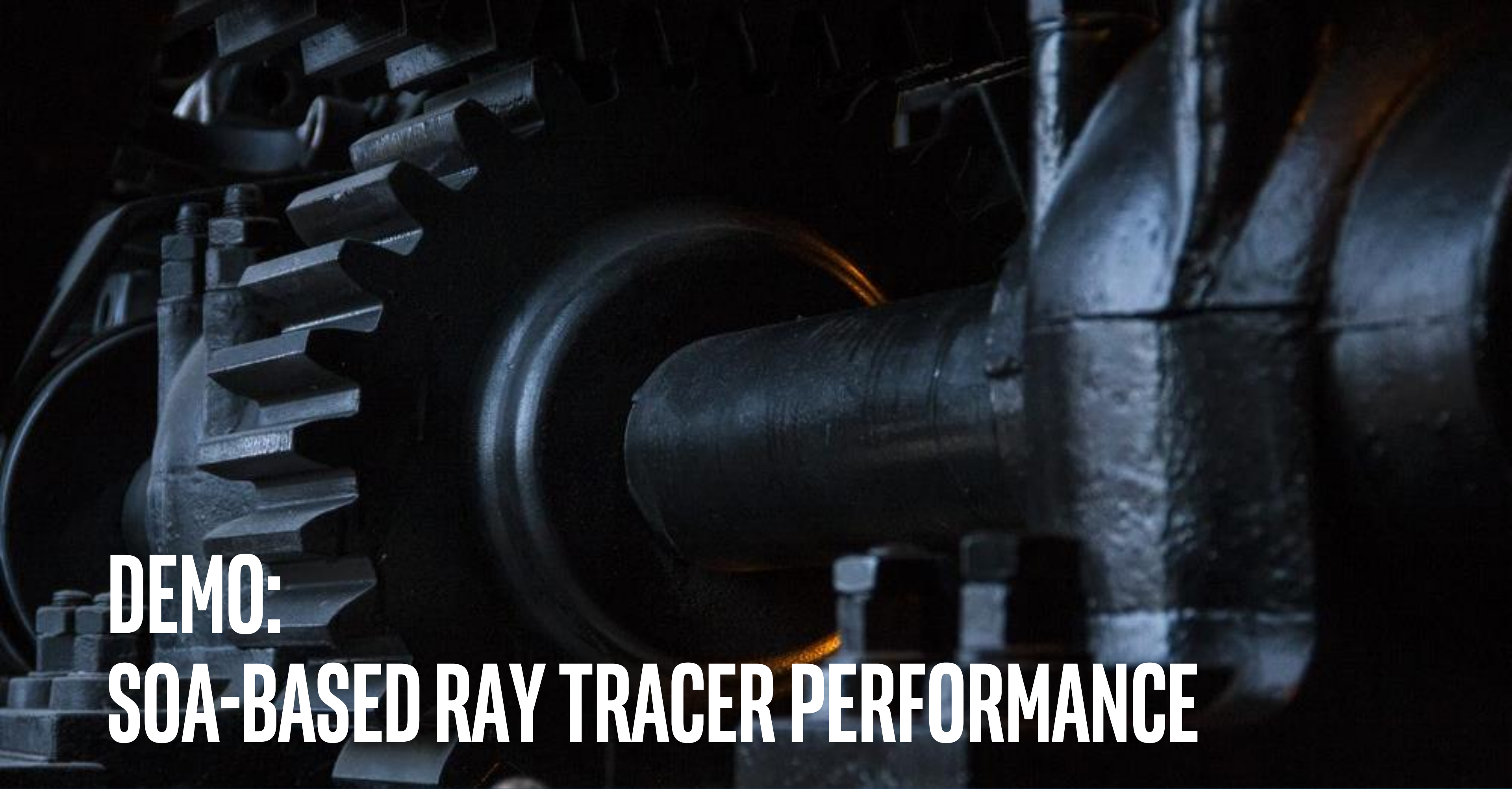
vaddps xmm, xmm1, xmm2

SoA




```
class VectorPacket256
{
    public Vector256<float> Xs;
    public Vector256<float> Ys;
    public Vector256<float> Zs;
}
```

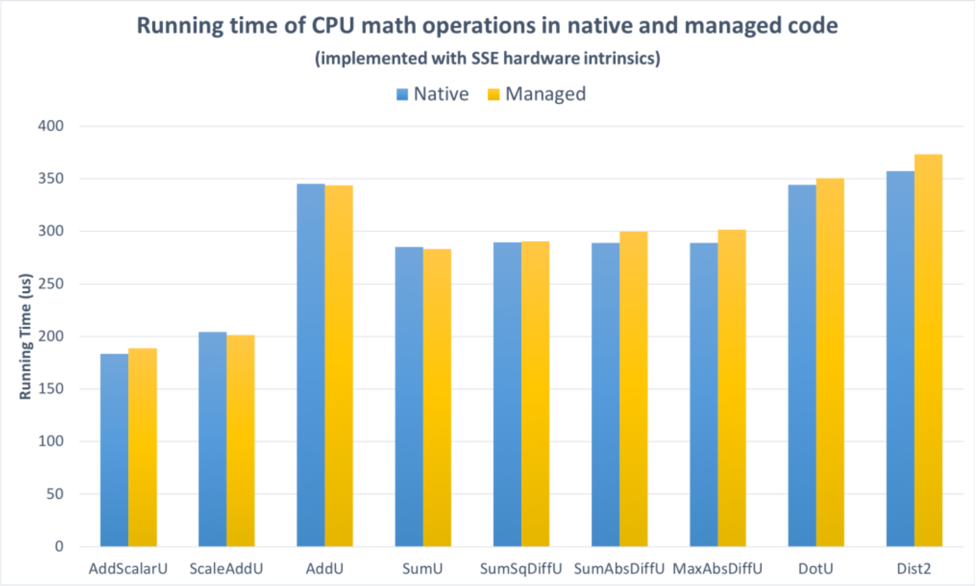
vaddps ymm, ymm1, ymm4
vaddps ymm, ymm2, ymm5
vaddps ymm, ymm3, ymm6



DEMO:
SOA-BASED RAY TRACER PERFORMANCE

Intel Hardware Intrinsics in Use

- CPU math operations in ML.NET
- SoA implementation of C# Raytracer
- Bit operations
- Matrix4x4 operations
- BLAKE2 hashing
- Your application!



*The figure shown above is from Microsoft blog on ML.NET

TURBOCHARGE YOUR APPLICATIONS WITH INTEL® HARDWARE INTRINSICS IN .NET CORE 3.0

Accelerate Your Applications

- Understand your bottlenecks
  - Intel VTune™ Amplifier is a great tool for profiling .NET Core application
- Use existing wealth of knowledge available on hardware intrinsics
  - Intel Intrinsics Guide: https://software.intel.com/sites/landingpage/IntrinsicsGuide/
  - How-to guides: https://software.intel.com/en-us/articles/how-to-use-intrinsics
- Explore existing solutions in C/C++
  - E.g., "Fast conversion from UTF-8 with C++, DFAs, and SSE intrinsics" (https://github.com/BobSteagall/CppCon2018)
- Optimize your application & re-measure
- Contribute to .NET Core and hardware intrinsics

BACKUP

Intrinsic Programming Tips

- Developers are responsible for checking hardware support
  - Different from higher-level SIMD APIs where software fallback is provided
- Test your application with different ISA levels
  - Use environment variables: e.g., COMPlus_EnableAVX
- Instruction encoding issues are handled automatically
  - No need to worry about AVX/SSE transitions
  - Best encoding by the JIT
- Use *using static* for concise syntax

intel®

experience what's inside™