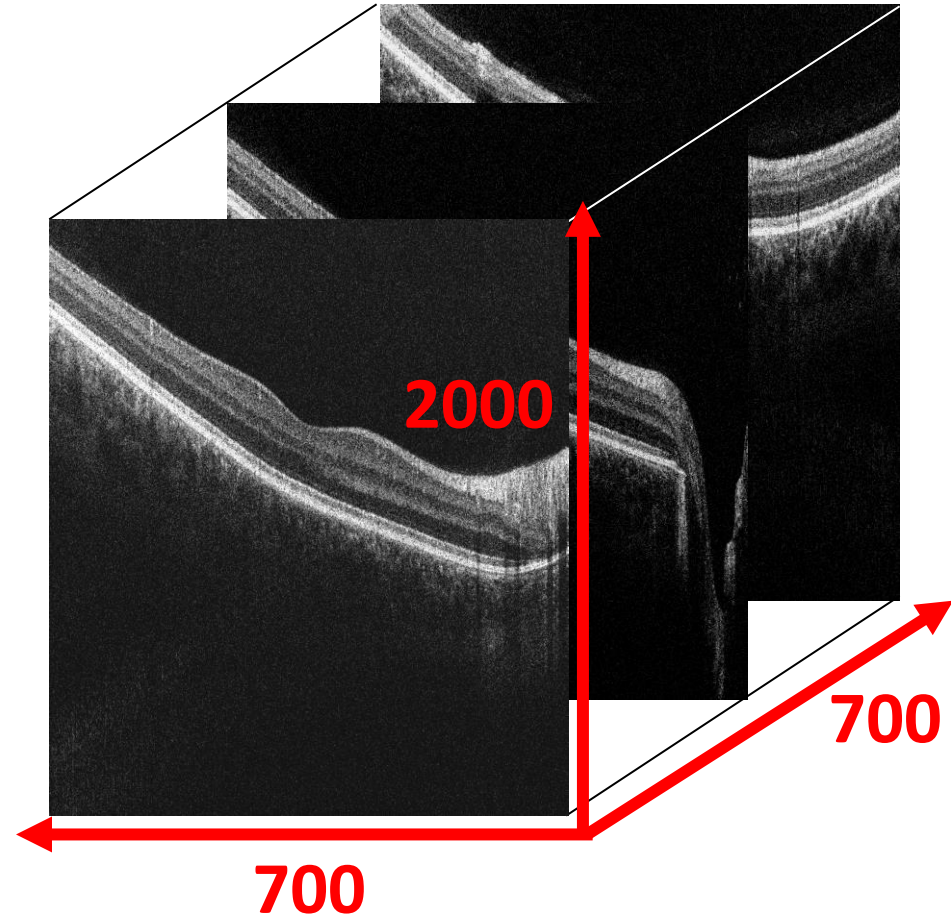
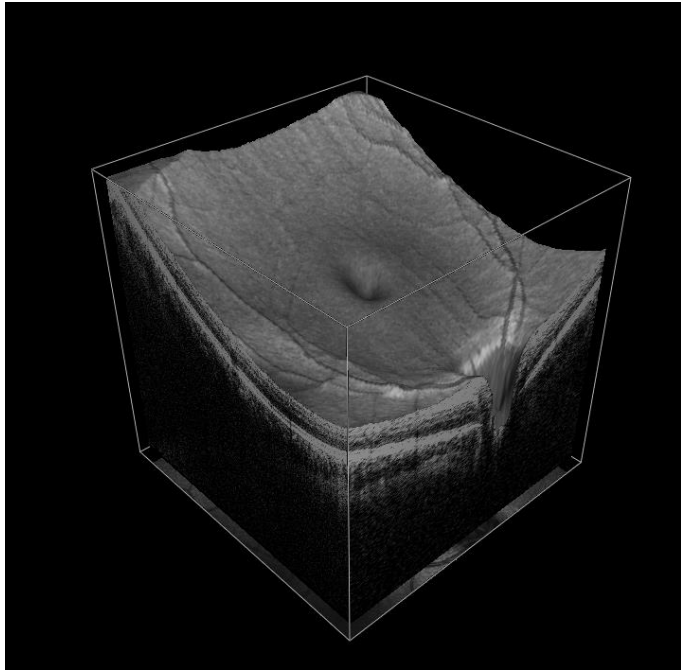


Dwa oblicza szybkości: czyli jak i dlaczego łączyć C++ z C#

Zygfried Wieszok

Optyczna tomografia oka



$700 * 2000 * 700 \approx 1GB \text{ pikseli}$

Obrazy pochodzą z *Ophthalmic Software Platform for HS-100*, **Canon**

BenchmarkDotNet

```
public class AdditionBenchmark{
    private const int N = 1024 * 1024 * 1024;
    private readonly byte[] argument1;
    private readonly byte[] argument2;
    private readonly byte[] result;

    public AdditionBenchmark(){
        argument1 = new byte[N];
        argument2 = new byte[N];
        result = new byte[N];

        var random = new Random(43);
        random.NextBytes(argument1);
        random.NextBytes(argument2);
    }

    [Benchmark]
    public void ManagedAdd(){
        // Todo: Add benchmarked code
    }
}
```

```

class Program{
    static void Main(string[] args){
        BenchmarkRunner.Run<AdditionBenchmark>();
    }
}

```

```

C:\WINDOWS\system32\cmd.exe
ConfidenceInterval = [5.3008 s; 5.4112 s] (CI 99.9%), Margin = 0.0552 s (1.03% of Mean)
Skewness = -0.43, Kurtosis = 2.23, MValue = 2
----- Histogram -----
[5.234 s ; 5.461 s] | @@@@@@@@@@@@@@@@
-----

// * Summary *

BenchmarkDotNet=v0.11.1, OS=Windows 10.0.17134.285 (1803/April2018Update/Redstone4)
Intel Core i7-7700HQ CPU 2.80GHz (Max: 2.81GHz) (Kaby Lake), 1 CPU, 8 logical and 4 physical cores
Frequency=2742184 Hz, Resolution=364.6728 ns, Timer=TSC
[Host]      : .NET Framework 4.7.2 (CLR 4.0.30319.42000), 64bit RyuJIT-v4.7.3163.0
DefaultJob  : .NET Framework 4.7.2 (CLR 4.0.30319.42000), 64bit RyuJIT-v4.7.3163.0

  Method | Mean | Error | StdDev |
-----:|-----:|-----:|-----:|
ManagedAdd | 5.356 s | 0.0552 s | 0.0516 s |

// * Legends *
Mean   : Arithmetic mean of all measurements
Error  : Half of 99.9% confidence interval
StdDev : Standard deviation of all measurements
1 s    : 1 Second (1 sec)

// ***** BenchmarkRunner: End *****
Run time: 00:02:28 (148.96 sec), executed benchmarks: 1

// * Artifacts cleanup *
Press any key to continue . . .

```

C#

C++ / CLI

C++

Prosta implementacja C#

```
[Benchmark]
public void ManagedAdd()
{
    for (int i = 0; i < N; i++)
    {
        var value = argument1[i] + argument2[i];
        result[i] = value <= byte.MaxValue ?
                        (byte) value : byte.MaxValue;
    }
}
```

C#

Method	Mean	Error	StdDev	Median
-----	-----:	-----:	-----:	-----:
ManagedAdd	4,897.7 ms	10.2526 ms	9.5903 ms	4,897.3 ms

C# wielowątkowo

[Benchmark]

C#

```
public void ManagedParallelAdd()
{
    var partitioner = Partitioner.Create(0, N);
    Parallel.ForEach(partitioner, range =>
    {
        for (int i = range.Item1; i < range.Item2; i++)
        {
            var value = argument1[i] + argument2[i];
            result[i] = value <= byte.MaxValue ?
                                (byte)value : byte.MaxValue;
        }
    });
}
```


Method	Mean	Error	StdDev	Median
ManagedAdd	4,897.7 ms	10.2526 ms	9.5903 ms	4,897.3 ms
ManagedParallelAdd	1,040.8 ms	18.8413 ms	17.6242 ms	1,045.0 ms

Implementacja C++

```
void NativeAdd(uint8_t * arg1, uint8_t * arg2,  
               uint8_t * dst,  int length) C++  
{  
    const auto charMax = std::numeric_limits<uint8_t>::max();  
    for (int i = 0; i < length; i++)  
    {  
        int value = arg1[i] + arg2[i];  
        dst[i] = value < charMax ? value : charMax;  
    }  
}
```

```
[Benchmark] C#  
public void NativeAdd()  
{  
    NativeLib.NativeAdd(argument1, argument2, result, N);  
}
```

Method	Mean	Error	StdDev	Median
-----	-----:	-----:	-----:	-----:
ManagedAdd	4,897.7 ms	10.2526 ms	9.5903 ms	4,897.3 ms
ManagedParallelAdd	1,040.8 ms	18.8413 ms	17.6242 ms	1,045.0 ms
NativeAdd	737.8 ms	14.3365 ms	19.1388 ms	731.4 ms

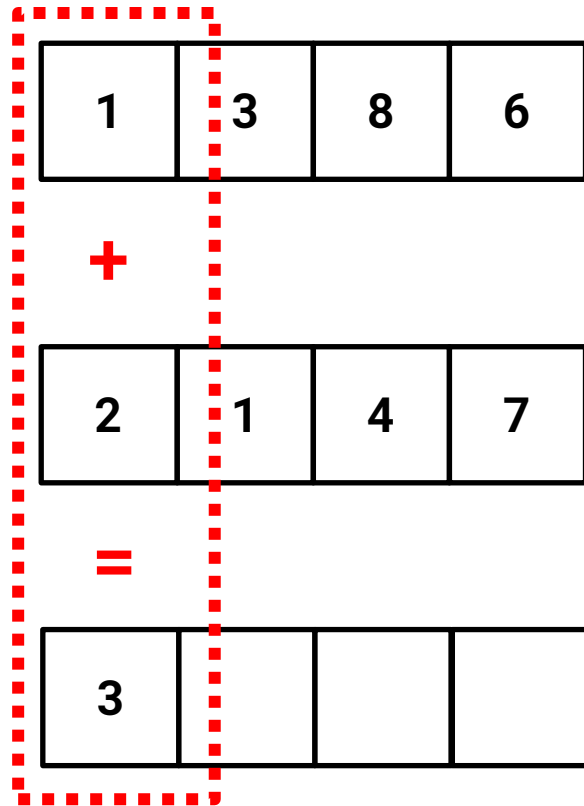
C++ wielowątkowo

```
void NativeParallelAdd(uint8_t * arg1, uint8_t * arg2,      C++
                      uint8_t * dst, int length)
{
    const auto charMax = std::numeric_limits<uint8_t>::max();

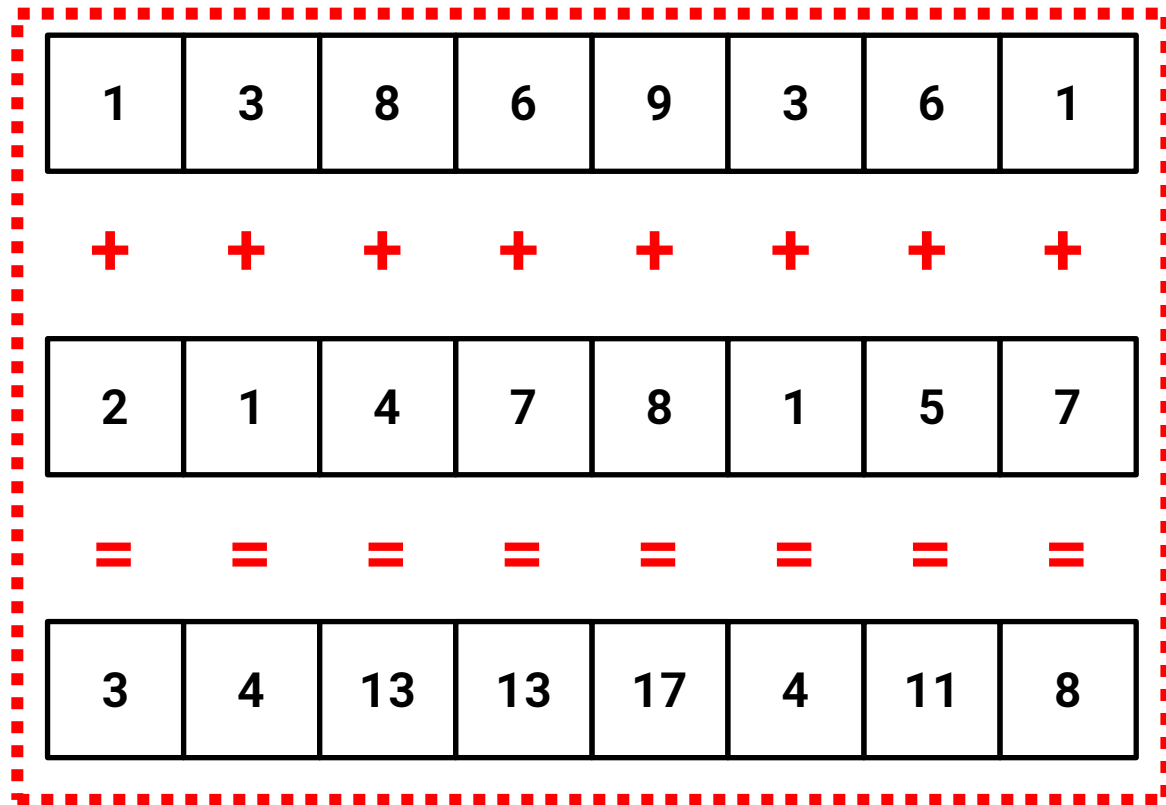
    #pragma omp parallel for
    for (int i = 0; i < length; i++)
    {
        int value = arg1[i] + arg2[i];
        dst[i] = value < charMax ? value : charMax;
    }
}
```

Method	Mean	Error	StdDev	Median
-----	-----:	-----:	-----:	-----:
ManagedAdd	4,897.7 ms	10.2526 ms	9.5903 ms	4,897.3 ms
ManagedParallelAdd	1,040.8 ms	18.8413 ms	17.6242 ms	1,045.0 ms
NativeAdd	737.8 ms	14.3365 ms	19.1388 ms	731.4 ms
NativeParallelAdd	233.8 ms	6.6497 ms	19.6068 ms	245.1 ms

Scalar mode



SIMD mode [MMX 64b]



```
void SimdAdd(uint8_t * arg1, uint8_t * arg2,  
            uint8_t * dst, int length)  
{  
    __m128i* arg1Ptr = reinterpret_cast<__m128i*>(arg1);  
    __m128i* arg2Ptr = reinterpret_cast<__m128i*>(arg2);  
    __m128i* dstPtr  = reinterpret_cast<__m128i*>(dst);  
  
    int i = 0;  
    for (; i < length; i += 128 / 8) {  
        __m128i a = _mm_loadu_si128(arg1Ptr);  
        __m128i b = _mm_loadu_si128(arg2Ptr);  
        __m128i c = _mm_adds_epu8(a, b);  
        _mm_storeu_si128(dstPtr, c);  
        arg1Ptr++;  
        arg2Ptr++;  
        dstPtr++;  
    }  
  
    // Add the remaining part of array indivisible by SIMD size  
    auto charMax = std::numeric_limits<uint8_t>::max();  
    for (; i < length; i++) {  
        int value = arg1[i] + arg2[i];  
        dst[i] = value < charMax ? value : charMax;  
    }  
}
```

```
int i = 0;
for (; i < length; i += 128 / 8) {
    __m128i a = _mm_loadu_si128(arg1Ptr);
    __m128i b = _mm_loadu_si128(arg2Ptr);
    __m128i c = _mm_adds_epu8(a, b);
    _mm_storeu_si128(dstPtr, c);
    arg1Ptr++;
    arg2Ptr++;
    dstPtr++;
}
```



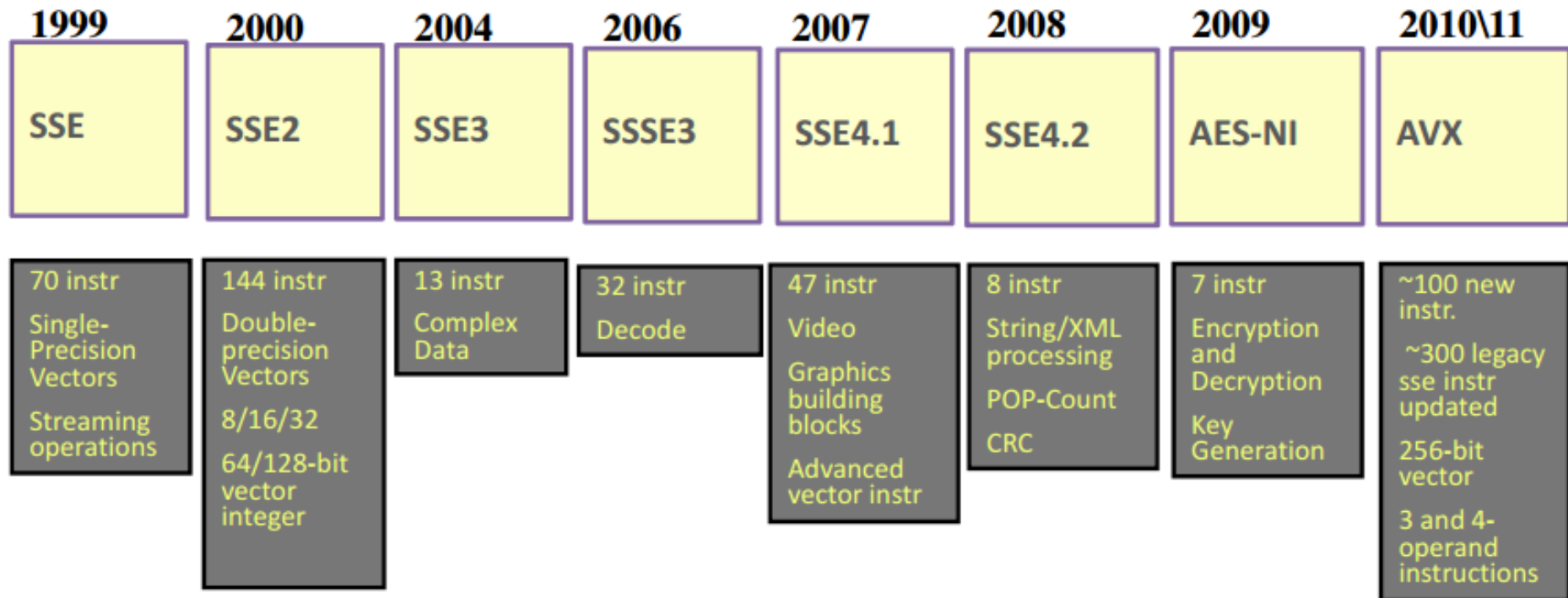
```
void SimdAdd(uint8_t * arg1, uint8_t * arg2,
             uint8_t * dst, int length)
{
    __m128i* arg1Ptr = reinterpret_cast<__m128i*>(arg1);
    __m128i* arg2Ptr = reinterpret_cast<__m128i*>(arg2);
    __m128i* dstPtr  = reinterpret_cast<__m128i*>(dst);

    int i = 0;
    for (; i < length; i += 128 / 8) {
        __m128i a = _mm_loadu_si128(arg1Ptr);
        __m128i b = _mm_loadu_si128(arg2Ptr);
        __m128i c = _mm_adds_epu8(a, b);
        _mm_storeu_si128(dstPtr, c);
        arg1Ptr++;
        arg2Ptr++;
        dstPtr++;
    }

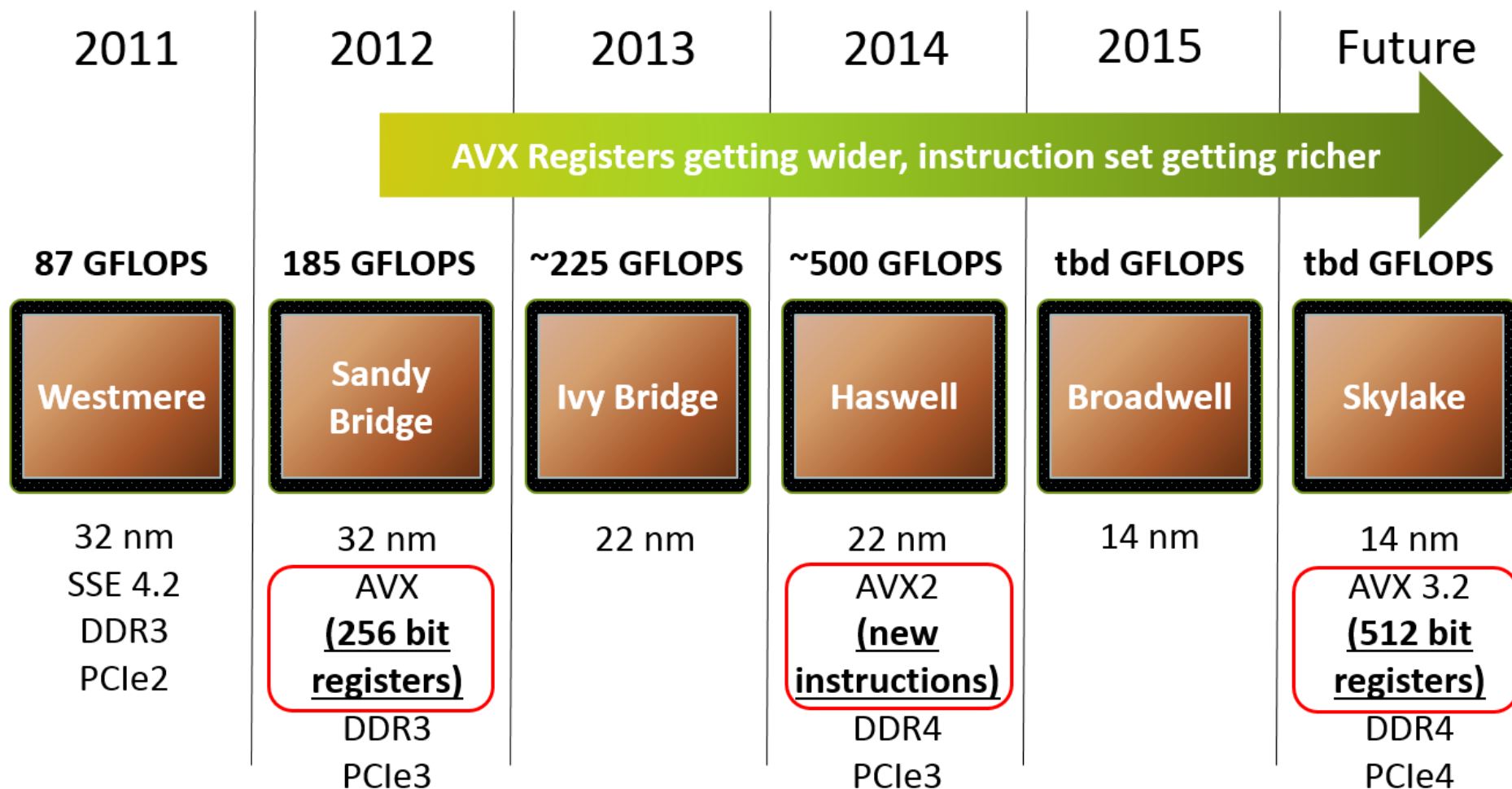
    // Add the remaining part of array indivisible by SIMD size
    auto charMax = std::numeric_limits<uint8_t>::max();
    for (; i < length; i++) {
        int value = arg1[i] + arg2[i];
        dst[i] = value < charMax ? value : charMax;
    }
}
```

Method	Mean	Error	StdDev	Median
-----	-----:	-----:	-----:	-----:
ManagedAdd	4,897.7 ms	10.2526 ms	9.5903 ms	4,897.3 ms
ManagedParallelAdd	1,040.8 ms	18.8413 ms	17.6242 ms	1,045.0 ms
NativeAdd	737.8 ms	14.3365 ms	19.1388 ms	731.4 ms
NativeParallelAdd	233.8 ms	6.6497 ms	19.6068 ms	245.1 ms
SimdAdd	178.4 ms	1.6278 ms	1.5227 ms	177.8 ms

SIMD: Continuous Evolution



Intel Advanced Vector eXtensions



Intel IPP

C++

```
void IntelIppAdd(uint8_t * arg1, uint8_t * arg2,  
                uint8_t * dst, int length)  
{  
    ippsAdd_8u_Sfs(arg1, arg2, dst, length, 0);  
}
```

Method	Mean	Error	StdDev	Median
-----	-----:	-----:	-----:	-----:
ManagedAdd	4,897.7 ms	10.2526 ms	9.5903 ms	4,897.3 ms
ManagedParallelAdd	1,040.8 ms	18.8413 ms	17.6242 ms	1,045.0 ms
NativeAdd	737.8 ms	14.3365 ms	19.1388 ms	731.4 ms
NativeParallelAdd	233.8 ms	6.6497 ms	19.6068 ms	245.1 ms
SimdAdd	178.4 ms	1.6278 ms	1.5227 ms	177.8 ms
IntelIppAdd	171.7 ms	1.3140 ms	1.1648 ms	171.4 ms

IPP wielowątkowo

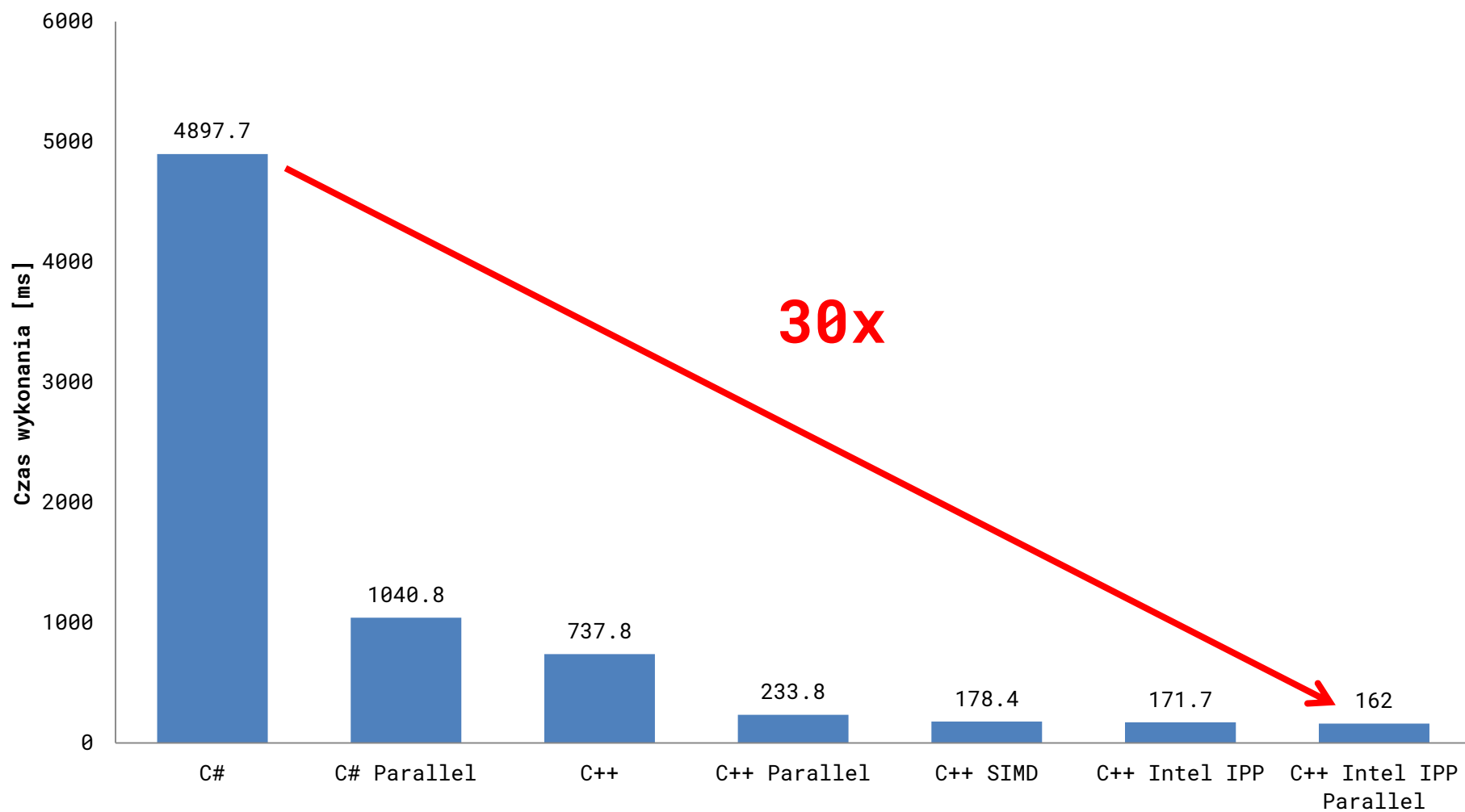
```
void IntelIppParallelAdd(uint8_t * arg1, uint8_t * arg2,          C++
                        uint8_t * dst, int length)
{
    int blockSize = 4096;
    int blockCount = length / blockSize;

    #pragma omp parallel for
    for (int n = 0; n < blockCount; ++n)
        ippsAdd_8u_Sfs(arg1 + n * blockSize, arg2 + n * blockSize,
                        dst + n * blockSize, blockSize, 0);

    // Add the remaining part of array indivisible by block size
    auto remainOffset = blockCount * blockSize;
    int remainLength = length - remainOffset;
    ippsAdd_8u_Sfs(arg1 + remainOffset, arg2 + remainOffset,
                    dst + remainOffset, remainLength, 0);
}
```

Method	Mean	Error	StdDev	Median
-----	-----:	-----:	-----:	-----:
ManagedAdd	4,897.7 ms	10.2526 ms	9.5903 ms	4,897.3 ms
ManagedParallelAdd	1,040.8 ms	18.8413 ms	17.6242 ms	1,045.0 ms
NativeAdd	737.8 ms	14.3365 ms	19.1388 ms	731.4 ms
NativeParallelAdd	233.8 ms	6.6497 ms	19.6068 ms	245.1 ms
SimdAdd	178.4 ms	1.6278 ms	1.5227 ms	177.8 ms
IntelIppAdd	171.7 ms	1.3140 ms	1.1648 ms	171.4 ms
IntelIppParallelAdd	162.0 ms	0.2794 ms	0.2614 ms	162.0 ms

Benchmark



Wywołanie C++ z C#

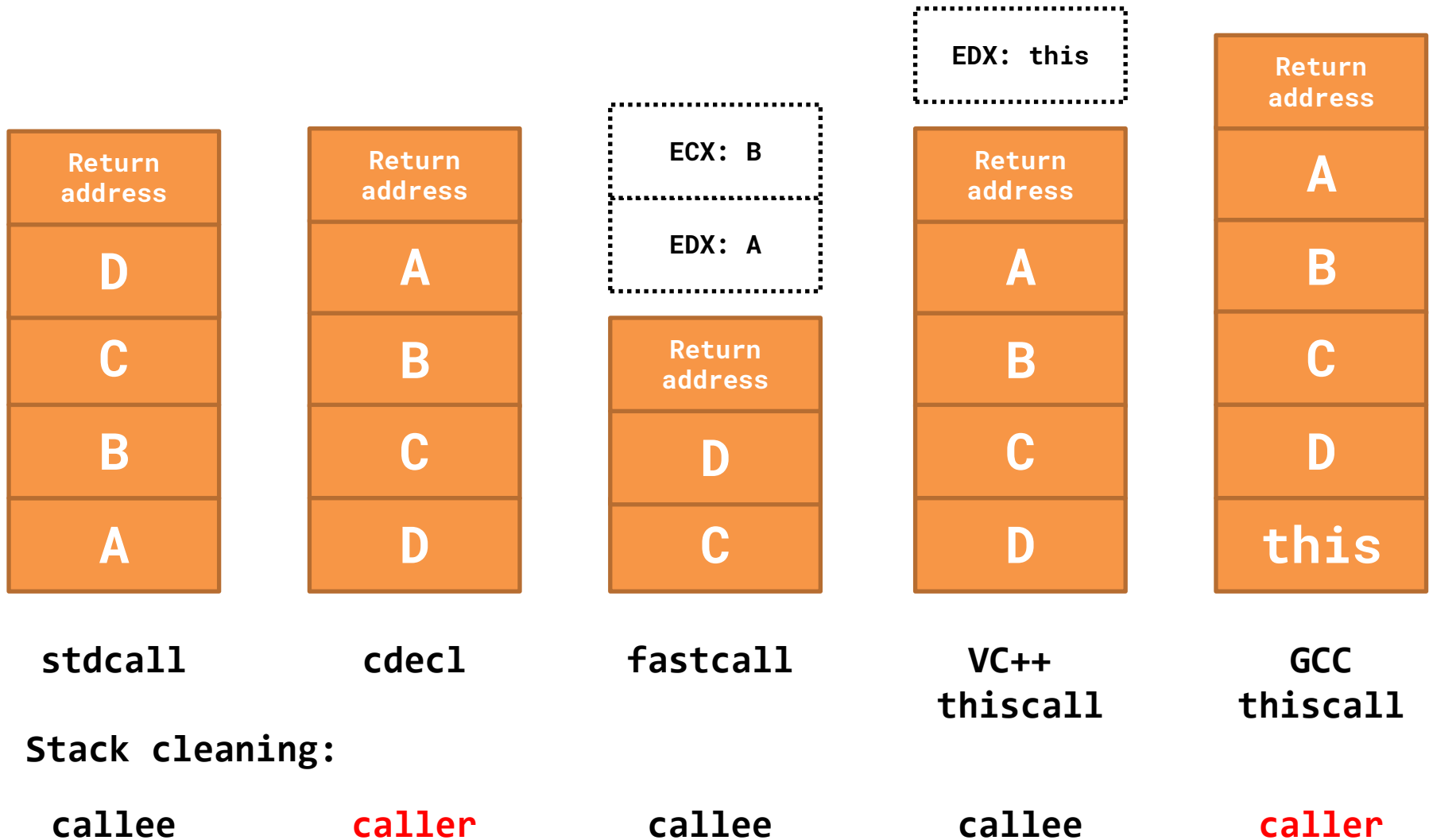
```
#ifdef NATIVELIB_EXPORTS
#define NATIVELIB_API __declspec(dllexport)
#else
#define NATIVELIB_API __declspec(dllimport)
#endif

void NATIVELIB_API NativeAdd(uint8_t * arg1, uint8_t * arg2,
                               uint8_t * dst, int length);
```

```
using System.Runtime.InteropServices;
public static class NativeLib
{
    [DllImport("NativeLib.dll",
               CallingConvention = CallingConvention.Cdecl)]
    public static extern void NativeAdd(byte[] arg1, byte[] arg2,
                                       byte[] arg3, int length);
}
```

C#

Fun(A, B, C, D)



Wywołanie C++ z C#

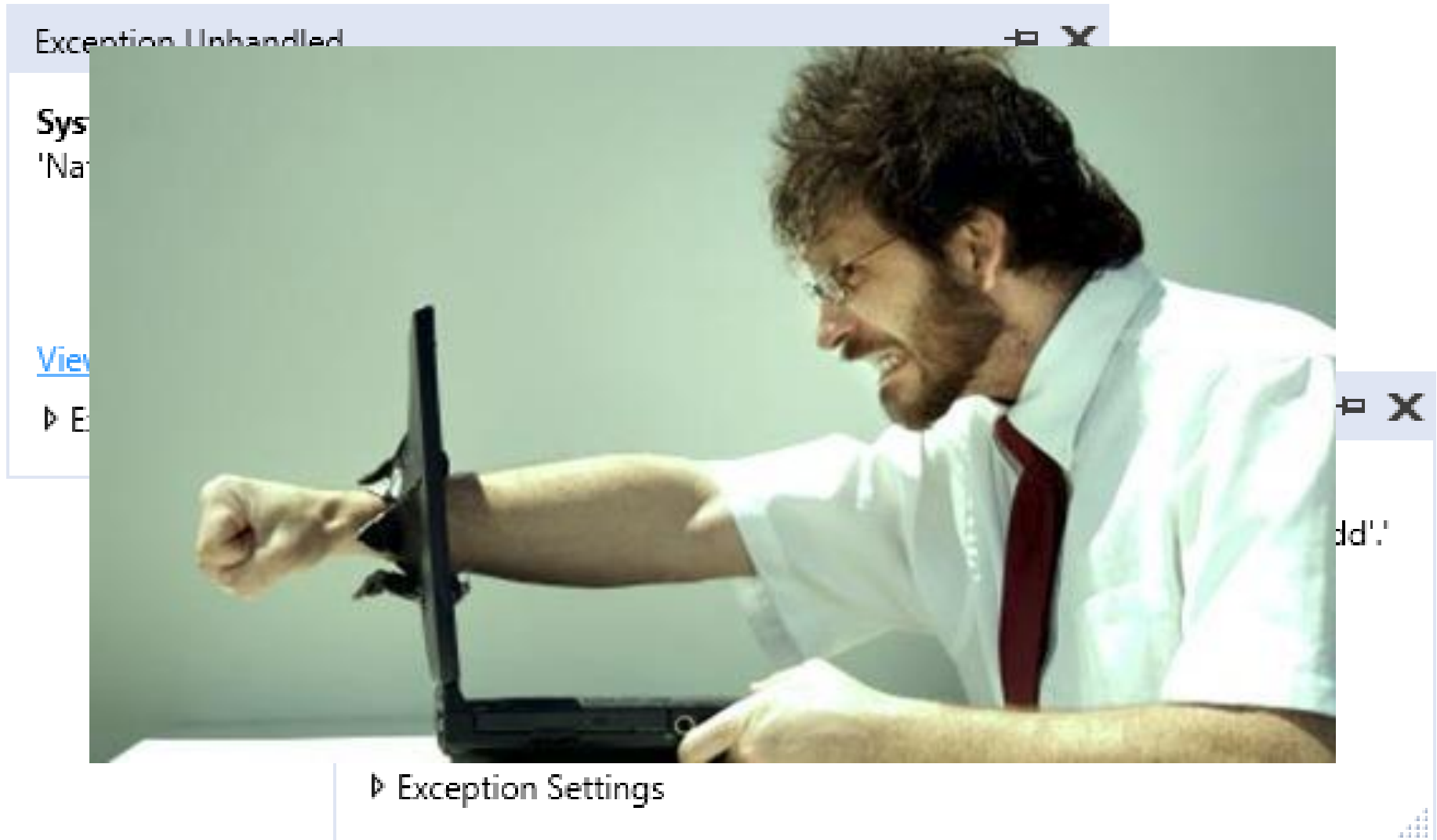
```
#ifdef NATIVELIB_EXPORTS
#define NATIVELIB_API __declspec(dllexport)
#else
#define NATIVELIB_API __declspec(dllimport)
#endif

void NATIVELIB_API NativeAdd(uint8_t * arg1, uint8_t * arg2,
                               uint8_t * dst, int length);
```

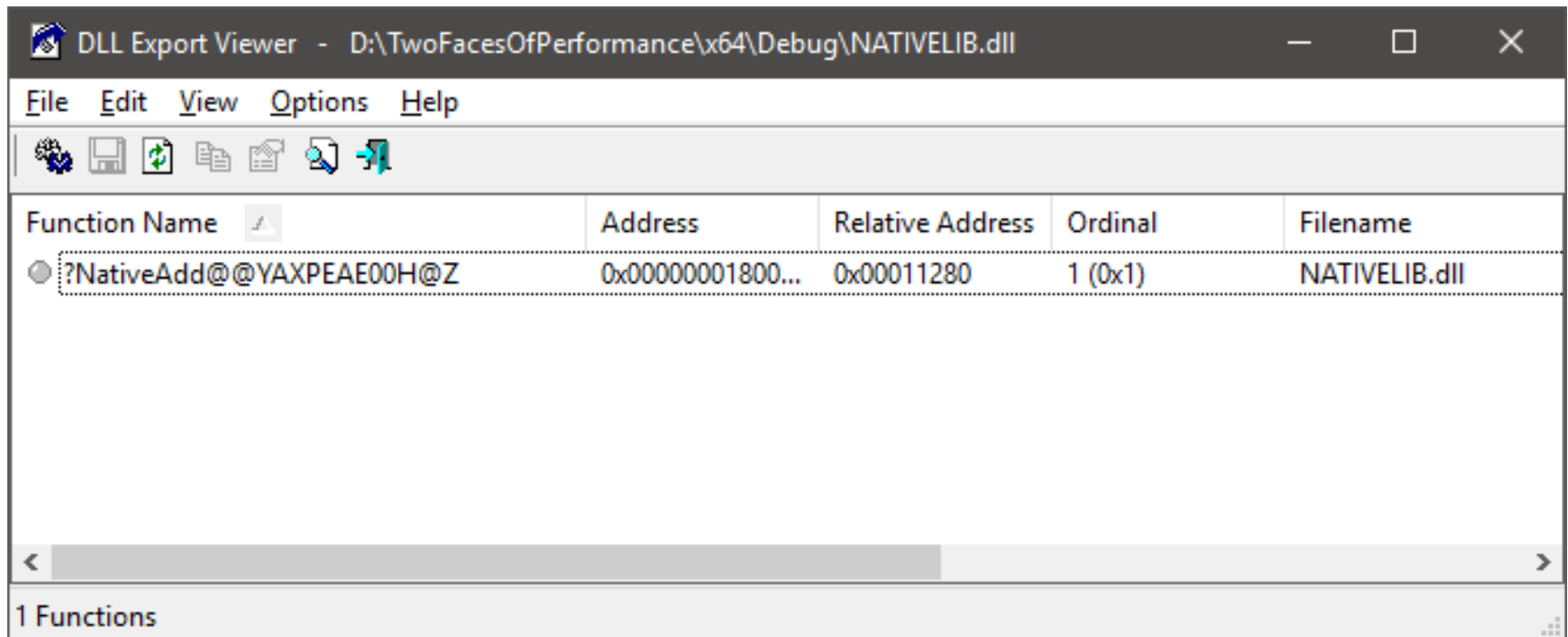
```
using System.Runtime.InteropServices;
public static class NativeLib
{
    [DllImport("NativeLib.dll",
               CallingConvention = CallingConvention.Cdecl)]
    public static extern void NativeAdd(byte[] arg1, byte[] arg2,
                                         byte[] arg3, int length);
}
```

C#

Compile and RUN!



Dll Export Viewer



NativeAdd -> ?NativeAdd@@YAXPEAE00H@Z

```
using System.Runtime.InteropServices;
public static class NativeLib
{
    [DllImport("NativeLib.dll",
        CallingConvention = CallingConvention.Cdecl
        EntryPoint = "?Name@@YAXPEAM00H@Z")]
    public static extern void NativeAdd(byte[] arg1, byte[] arg2,
        byte[] arg3, int length);
}
```

C#

LUB

```
#ifdef NATIVELIB_EXPORTS
#define EXPORT_API __declspec(dllexport)
#else
#define EXPORT_API __declspec(dllimport)
#endif

extern "C" {
    void EXPORT_API NativeAdd(unsigned char * arg1, unsigned char * arg2,
        unsigned char * dst, int length);
}
```

C++

A co z klasami?

C++

```
class NATIVELIB_API NativeCalculator
{
public:
    void Add(
        unsigned char* arg1,
        unsigned char* arg2,
        unsigned char* dst,
        int length);

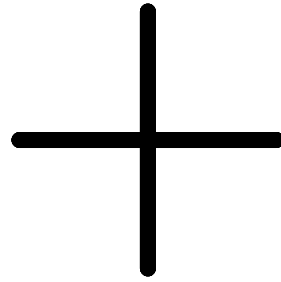
    void Sum(
        const std::vector<unsigned char*>& arguments,
        unsigned char* dst,
        int length);
};
```

C++

```
extern „C”{  
    void* NATIVELIB_API CreateCalculator()  
    {  
        return new NativeCalculator();  
    }  
  
    void AddCalculator(void* object, uint8_t* arg1,  
        uint8_t* arg2, uint8_t* dst, int length)  
    {  
        auto calc = (NativeCalculator*)object;  
        calc->Add(arg1, arg2, dst, length);  
    }  
  
    [...]  
  
    void NATIVELIB_API ReleaseCalculator(void* object)  
    {  
        delete object;  
    }  
}
```

C++ / CLI

C++



C#



C++/CLI



THE LIFE OF A SOFTWARE
ENGINEER.

CLEAN SLATE. SOLID
FOUNDATIONS. THIS TIME
I WILL BUILD THINGS THE
RIGHT WAY.



MUCH LATER...

OH MY. I'VE
DONE IT AGAIN,
HAVEN'T I ?



C#

Application



C++/CLI

Wrapper



C++

Native code

C++/CLI

Native code

```
NativeClass * aClass = new NativeClass()  
  
aClass->Function()  
  
delete aClass;
```

Managed code

```
ManagedClass ^ bClass = gcnew ManagedClass()  
  
bClass->Function()
```

```
public ref class CalculatorWrapper : public System::IDisposable
{
    NativeCalculator *nativeCalculator;
public:
    CalculatorWrapper(){
        nativeCalculator = new NativeCalculator();
    }

    void Add(cli::array<unsigned char>^ arg1,
            cli::array<unsigned char>^ arg2,
            cli::array<unsigned char>^ dst){
        // TODO: Implementation goes here
    }

    ~CalculatorWrapper(){
        this->!CalculatorWrapper();
    }

    !CalculatorWrapper(){
        delete nativeCalculator;
    }
}
```


C++

```
Class NativeCalculator
{
public:
    void Add(
        unsigned char* arg1,
        unsigned char* arg2,
        unsigned char* dst,
        int length);

    void Sum(
        const std::vector<unsigned char*>& arguments,
        unsigned char* dst,
        int length);
};
```

C++/CLI

```
void Add(cli::array<unsigned char>^ arg1,  
        cli::array<unsigned char>^ arg2,  
        cli::array<unsigned char>^ dst)  
{  
    pin_ptr<unsigned char> arg1Pin = &arg1[0];  
    pin_ptr<unsigned char> arg2Pin = &arg2[0];  
    pin_ptr<unsigned char> dstPin  = &dst[0];  
  
    try  
    {  
        nativeCalculator->Add(arg1Pin, arg2Pin,  
                               dstPin,  arg1->Length);  
    }  
    catch (const std::exception& ex)  
    {  
        throw gcnew System::Exception(gcnew String(ex.what()));  
    }  
}
```

C++

```
Class NativeCalculator
{
public:
    void Add(
        unsigned char* arg1,
        unsigned char* arg2,
        unsigned char* dst,
        int length);

    void Sum(
        const std::vector<unsigned char*>& arguments,
        unsigned char* dst,
        int length);
};
```

```
cli::array<unsigned char>^ Sum(cli::array<cli::array<unsigned char>^>^ arguments)
{
    cli::array<GCHandle>^ memoryHandles = gcnew cli::array<GCHandle>(arguments->Length);

    try
    {
        std::vector<unsigned char*> nativePtrs(arguments->Length);
        for (int i = 0; i < arguments->Length; i++)
        {
            memoryHandles[i] = GCHandle::Alloc(arguments[i], GCHandleType::Pinned);
            nativePtrs[i] = (unsigned char*) memoryHandles[i].AddrOfPinnedObject().ToPointer();
        }

        pin_ptr<unsigned char> outputPin = &output[0];

        nativeCalculator->Sum(nativePtrs, outputPin, arguments[0]->Length);
    } catch (const std::exception& ex)
    {
        throw gcnew System::Exception(gcnew String(ex.what()));
    } finally
    {
        for (int i = 0; i < arguments->Length; i++)
        {
            memoryHandles[i].Free();
        }
    }
}
```

```
cli::array<GCHandle>^ memoryHandles =  
    gcnew cli::array<GCHandle>(arguments->Length);  
  
try  
{  
    std::vector<unsigned char*> nativePtrs(arguments->Length);  
    for (int i = 0; i < arguments->Length; i++){  
        memoryHandles[i] = GCHandle::Alloc(arguments[i],  
                                             GCHandleType::Pinned);  
  
        nativePtrs[i] = (unsigned char*)  
            memoryHandles[i].AddrOfPinnedObject().ToPointer();  
    }  
[...]  
} finally  
{  
    for (int i = 0; i < arguments->Length; i++)  
    {  
        memoryHandles[i].Free();  
    }  
}
```

```
cli::array<unsigned char>^ Sum(cli::array<cli::array<unsigned char>^>^ arguments)
{
    cli::array<GCHandle>^ memoryHandles = gcnew cli::array<GCHandle>(arguments->Length);

    try
    {
        std::vector<unsigned char*> nativePtrs(arguments->Length);
        for (int i = 0; i < arguments->Length; i++)
        {
            memoryHandles[i] = GCHandle::Alloc(arguments[i], GCHandleType::Pinned);
            nativePtrs[i] = (unsigned char*) memoryHandles[i].AddrOfPinnedObject().ToPointer();
        }

        pin_ptr<unsigned char> outputPin = &output[0];

        nativeCalculator->Sum(nativePtrs, outputPin, arguments[0]->Length);
    } catch (const std::exception& ex)
    {
        throw gcnew System::Exception(gcnew String(ex.what()));
    } finally
    {
        for (int i = 0; i < arguments->Length; i++)
        {
            memoryHandles[i].Free();
        }
    }
}
```

```
public ref class CalculatorWrapper : public System::IDisposable
{
    NativeCalculator *nativeCalculator;
public:
    CalculatorWrapper(){
        nativeCalculator = new NativeCalculator();
    }

    ~CalculatorWrapper(){

        // Dispose managed

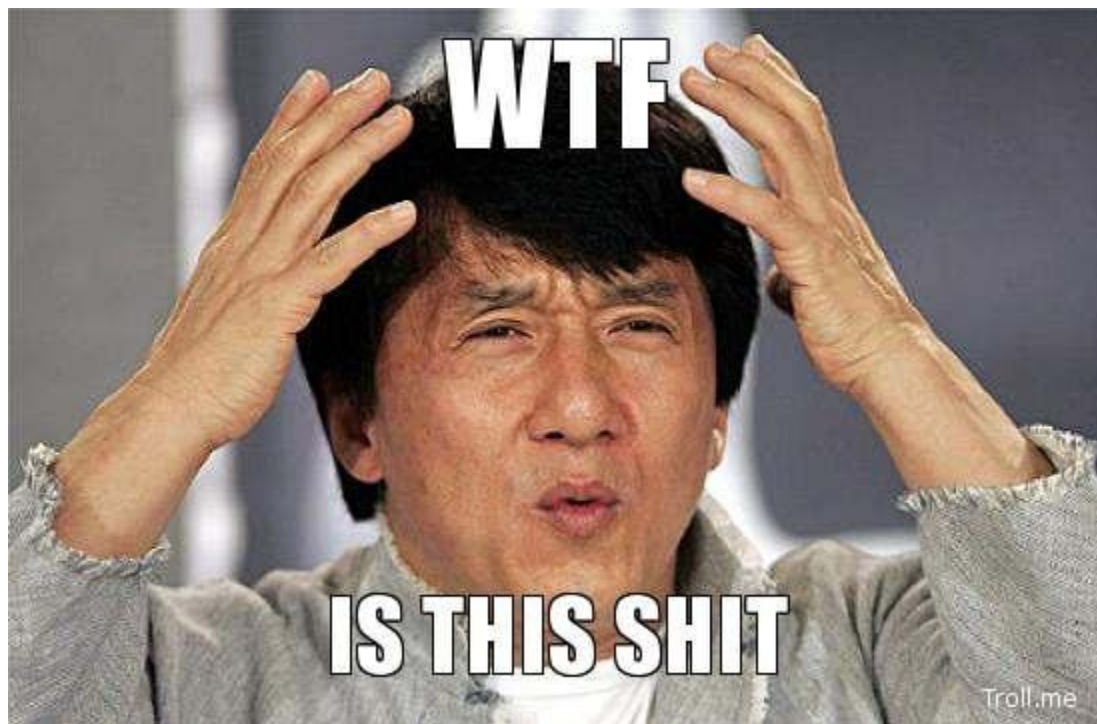
        //MANDATORY: Call finalizer
        this->!CalculatorWrapper();
    }

    !CalculatorWrapper(){

        //Dispose unmanaged
        delete nativeCalculator;
    }
}
```

Zwalnianie Zasobów

Typ	C#	C++ / CLI	C++
Managed	Dispose()	~ClassName()	
Unmanaged	~ClassName()	!ClassName()	~ClassName()



C#

GUI [WPF]



C++ / CLI

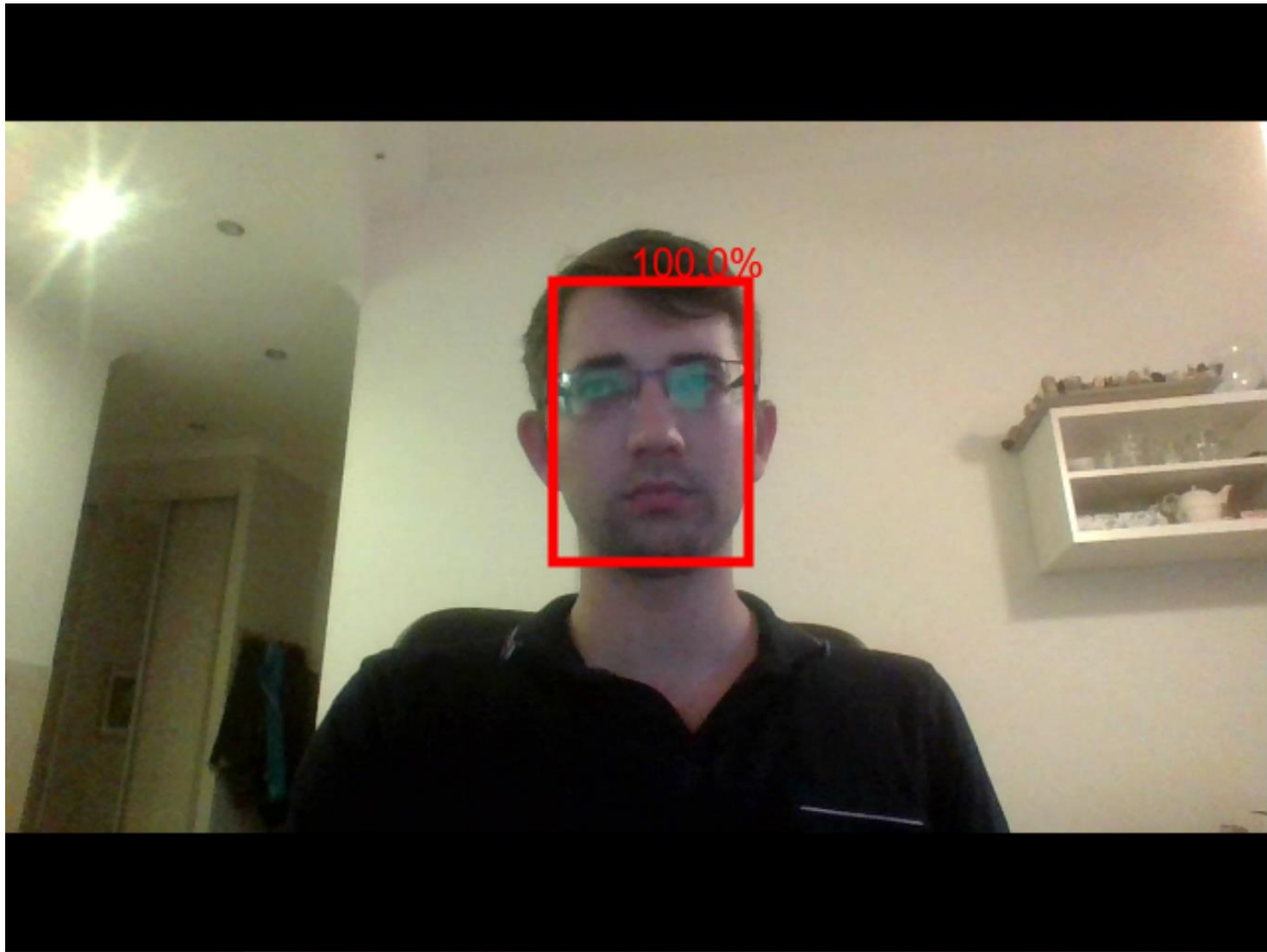
Wrapper



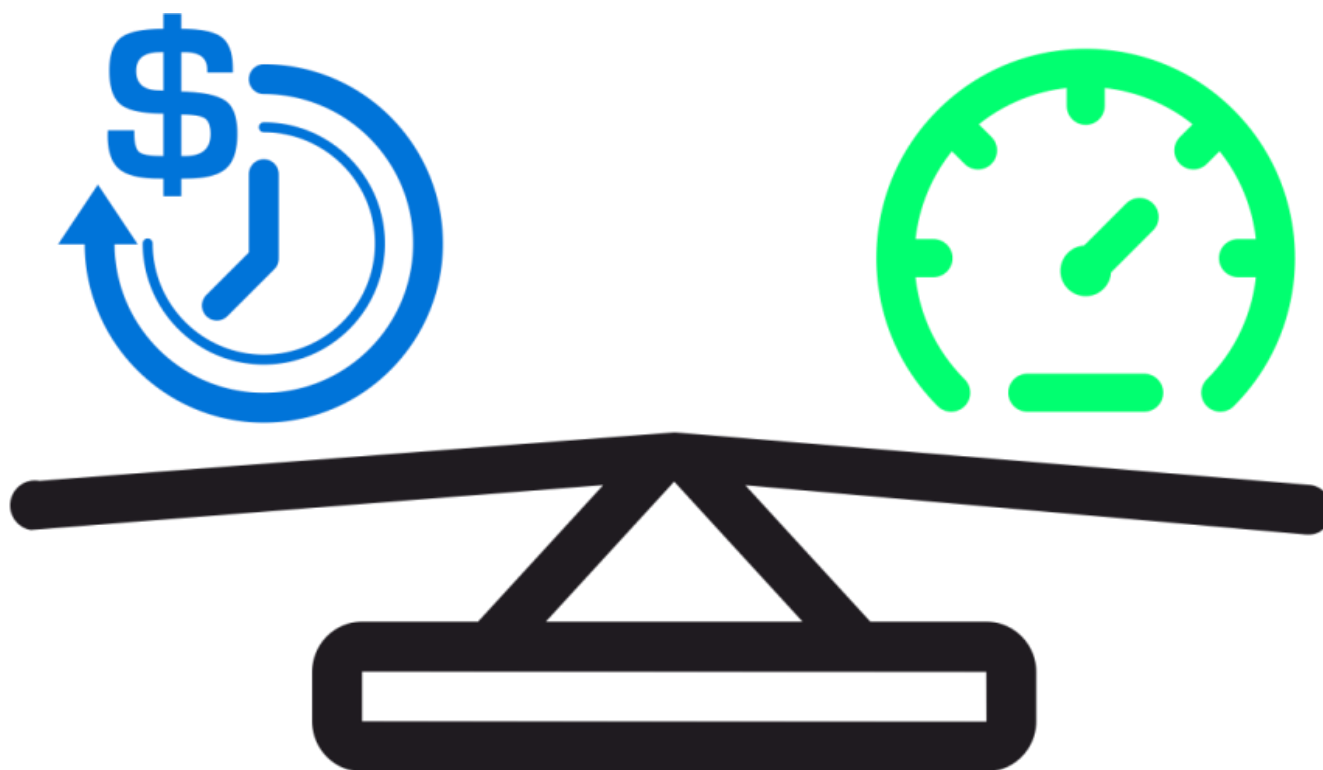
C++

OpenCV

WPF+DNN



Podsumowanie



Zygfried Wieszok

**Dwa oblicza szybkości:
czyli jak i dlaczego łączyć C++ z C#**

<https://github.com/zygfrydw/TwoFacesOfPerformance>

Zygfried.wieszok@gmail.com

Dispose Pattern

C#

```
using System;
class BaseClass : IDisposable{
    bool disposed = false;

    public void Dispose(){
        Dispose(true);
        GC.SuppressFinalize(this);
    }

    protected virtual void Dispose(bool disposing){
        if (disposed)
            return;
        if (disposing) {
            // Free any other managed objects here.
        }
        // Free any unmanaged objects here.
        disposed = true;
    }

    ~BaseClass(){
        Dispose(false);
    }
}
```

C++/CLI

```
~CalculatorWrapper(){  
    this->!CalculatorWrapper();  
}  
  
!CalculatorWrapper(){  
    delete nativeCalculator;  
}
```

```
private void \u007ECalculatorWrapper() {  
    this.\u0021CalculatorWrapper();  
  
private unsafe void \u0021CalculatorWrapper() {  
    \u003CModule\u003E.delete(  
        (void*) this.nativeCalculator, 1UL);  
}  
  
[HandleProcessCorruptedStateExceptions]  
protected virtual void  
Dispose([MarshalAs(UnmanagedType.U1)] bool A_0) {  
    if (A_0) {  
        this.\u007ECalculatorWrapper();  
    } else {  
        try {  
            this.\u0021CalculatorWrapper();  
        } finally {  
            // ISSUE: explicit finalizer call  
            base.Finalize();  
        }  
    }  
}  
  
public virtual void Dispose()  
{  
    this.Dispose(true);  
    GC.SuppressFinalize((object) this);  
}  
  
~CalculatorWrapper()  
{  
    this.Dispose(false);  
}
```

C++/CLI

```
~CalculatorWrapper(){  
}  
  
!CalculatorWrapper(){  
    delete nativeCalculator;  
}
```

Wywołanie metody `Dispose` spowoduje wywołanie **`SuppressFinalizer`**. W związku z czym `Finalizer` nigdy nie będzie wywołany == wyciek pamięci!!!

```
private void \u007ECalculatorWrapper(){  
}  
  
private unsafe void \u0021CalculatorWrapper(){  
    \u003CModule\u003E.delete(  
        (void*) this.nativeCalculator, 1UL);  
}  
  
[HandleProcessCorruptedStateExceptions]  
protected virtual void  
Dispose([MarshalAs(UnmanagedType.U1)] bool A_0){  
    if (A_0)  
        return;  
    try{  
        this.\u0021CalculatorWrapper();  
    } finally {  
        // ISSUE: explicit finalizer call  
        base.Finalize();  
    }  
}  
  
public virtual void Dispose() {  
    this.Dispose(true);  
    GC.SuppressFinalize((object) this);  
}  
  
~CalculatorWrapper() {  
    this.Dispose(false);  
}
```