Integrating C# and C++

Merging native and managed worlds together

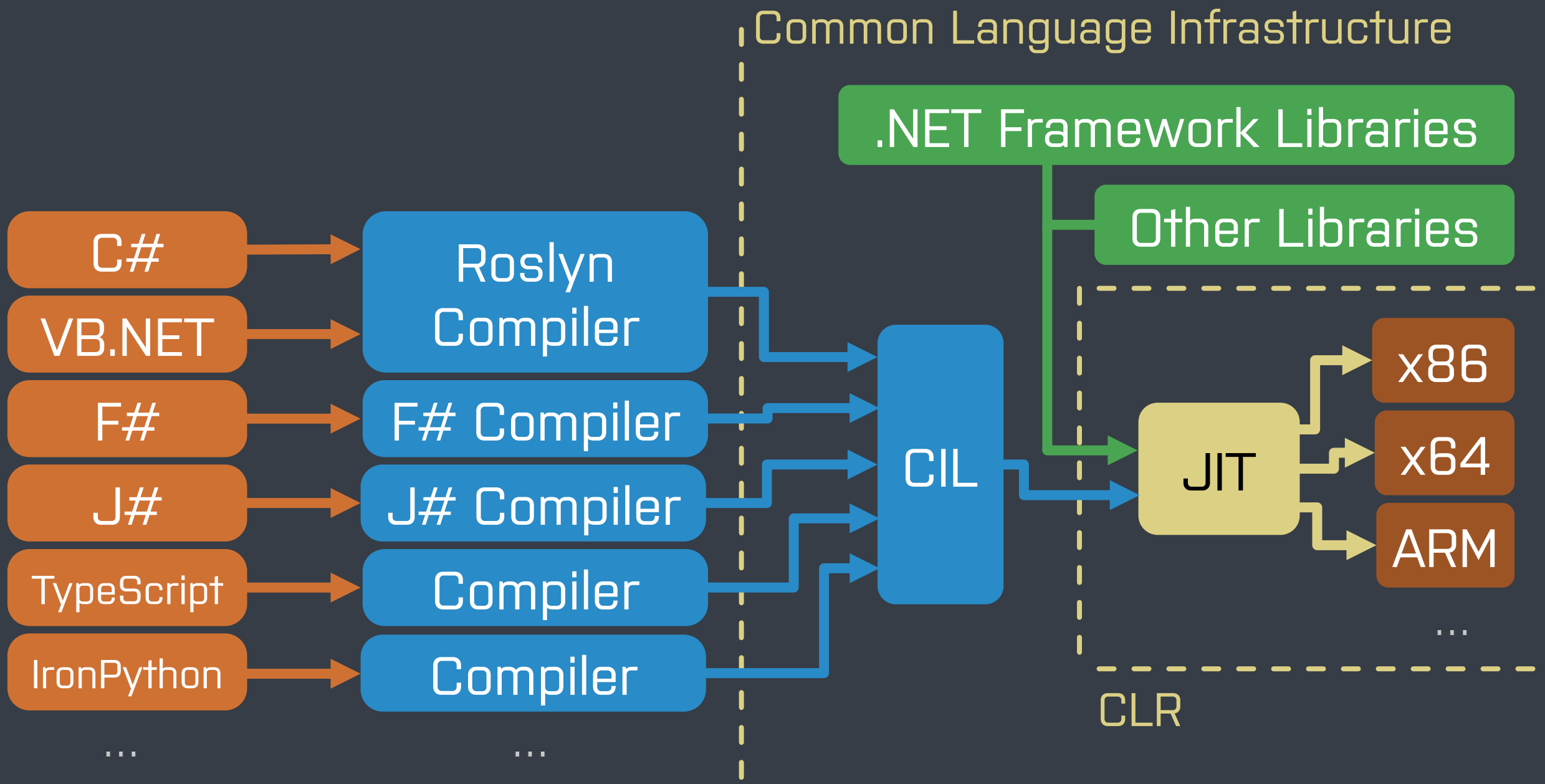
About Myself

- Began studying CS at the KIT since fall 2015
- Working part-time at the Fraunhofer's IOSB since spring 2018
- Used .NET since 2009 and C# since 2011
- Contributed to:
  - Microsoft's Roslyn C#/VB.NET Compiler project
  - C#-based OS "Cosmos"

QUICK INTRODUCTION: .NET AND C#

Quick Intro: .NET

- Developed by Microsoft in the 90s
- Alternative to Java
- OOP
- Safe development (CG, memory management)
- Robust API
- Cross-platform, cross-architecture:
  - .NET-Framework on Windows, MONO on Linux
  - .NET is open-source and runs on any platform


Common Language Infrastructure
.NET Framework Libraries
Other Libraries
C#
VB.NET
F#
J#
TypeScript
IronPython
...
Roslyn Compiler
F# Compiler
J# Compiler
Compiler
Compiler
...
CIL
JIT
x86
x64
ARM
...
CLR

C# Features

- C-syntax
- `class` (managed)
  `struct` (unmanaged)
  `interface` can be implemented by both
- Pointers! (with restrictions)
- LINQ
- Reflection
- λ-Functions
- Tuples
- String interpolation

```
static int Main(string[] args)
{
    var sorted = from a in args
                 where a.Length % 2 != 0
                 orderby a.Length descending
                 select (a.Length, a[0]);

    foreach ((int l, char c) in sorted)
        Console.WriteLine($"length: {l}, first char: {c}");

    return 0;
}
```

```
[StructLayout(LayoutKind.Sequential)]
struct Point2D
{
    public int X;
    public int Y;
}

static void Main(string[] args)
{
    Point2D point = default; // or new Point2D();
    Point2D* ptr = &point;

    // point has the value (X=0, Y=0)
    ptr->X = 42;
    point.Y = 315;
    // point has the value (X=42, Y=315)
}
```

C# Features

- Properties
- Event subscriptions
- Operator overloading
- Generics (= templates, but more run-time flexible)
- Local Functions
- Anonymous Types
- Window APIs (WinForms, WPF, …)

…

```
class List<T>
{
    Node<T> Head { get; }

    List() => Head = new Node<T>();

    class Node<T>
    {
        T Value { get; set; }
        Node<T> Next { set; get; }
    }
}
```

C#: Classes and Structures

`class`

- Heap-allocated
- Passed by reference
- CLR tracks instances
- Inheritance *(not multiple)*
- **`string`**s are reference types
  specially handled
- **`object`** is the base of all types

C#: Classes and Structures

`class`

- Heap-allocated
- Passed by reference
- CLR tracks instances
- Inheritance *(not multiple)*
- **`string`**s are reference types specially handled
- **`object`** is the base of all types

`struct`

- Stack-allocated
- Fixed-sized
- Can be used as pointer types
- Primitives, **`enum`**s, **`union`**s are **`struct`**s

*This does not apply to managed **`struct`**s*

Interop: C# and C++

- *ActiveX (deprecated)*
- *COM/OLE (deprecated)*
- P/Invoke services
- *Managed C++ (deprecated)*
- C++/CLI
- C++/CX

P/INVOKE
[PLATFORM INVOCATION SERVICES]

P/Invoke

C++

```cpp
typedef int(*intfunc)(void);


extern "C" __declspec(dllexport) double Test(intfunc func)
{
    return func() * 2.5;
}
```

C#

```csharp
public delegate int intfunc();

[DllImport("native.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern double Test(intfunc func);


static void Main()
{
    double result = Test(() => int.Parse(Console.ReadLine()));

    Console.WriteLine(result);
}
```

P/Invoke – Type Conversions

C#	Visual C++	Size
`sbyte`, `byte`, `bool`	`unsigned char`	1 B
`char` (UTF-16), `short`, `ushort`	(`unsigned`) `short`	2 B
`int`, `uint`, `float`	(`unsigned`) `int`, `float`	4 B
`long`, `ulong`, `double`	(`unsigned`) `long long`, `double`	8 B
`decimal`	`float128` (e.g. boost library)	16 B
`IntPtr`, `UIntPtr` (safe) `ref xxx` (safe), `xxx*` (unsafe)	`xxx*`	4 or 8 B
<delegate>	<function pointer>	4 or 8 B
`string`, `StringBuilder`	(`w`)`char*`	

P/Invoke – Type Conversions

```
struct data
{
    int i;
    float f;
};
```

Sequential structures

```
[StructLayout(LayoutKind.Sequential)]
public struct Data
{
    public int I;
    public float F;
}
```

```
union data
{
    int i;
    float f;
};
```

Unified structures

```
[StructLayout(LayoutKind.Explicit)]
public struct Data
{
    [FieldOffset(0)]
    public int I;
    [FieldOffset(0)]
    public float F;
}
```

P/Invoke – Type Conversions

```
struct data
{
public:
    int count;
    float raw[1024];
};
```

```
[StructLayout(LayoutKind.Sequential)]
public struct Data
{
    public int Count;
    public fixed float Raw[1024];
}
```

```
typedef int(*MyCallback)(void*, int);
```

```
delegate int MyCallback(void* data, int size);
```

```
typedef wchar_t TCHAR;

extern "C" __declspec(dllexport) void Print(TCHAR* str)
{
}
```

```
[DllImport("native.dll")]
private static extern void Print(
    [MarshalAs(UnmanagedType.LPTStr)] string str
);
```

P/Invoke – F# example

```fsharp
open System.Runtime.InteropServices


[<DllImport "native.dll">]
extern void generate_fibonacci(int[] fib, int len)

[<EntryPoint>]
let main argv =
    let count = 100
    let arr = Array.zeroCreate count

    generate_fibonacci(arr, count)

    printf "Generated %d fibonacci numbers:\n%A" count arr
    0
```

```cpp
extern "C" __declspec(dllexport) void generate_fibonacci(int* target, int count)
{
    if (target)
        for (int i = 0; i < count; ++i)
            target[i] = i < 2 ? i : target[i - 1] + target[i - 2];
}
```

Reverse P/Invoke – Calling managed code

1) Create runtime host
2) Create application domain (= context)
3) Fetch entry point
4) Create shared memory region
5) Pass data
6) Call entry point
7) Fetch back result(s)

→ Slow, massive overhead, dirty

→ **NOT ADVISED!!**

P/Invoke – Pros

+ Supports all dynamic libraries (.dll, .so, …)
+ Functions, interfaces and structures can be passed
+ Source-language independent (C#, F#, VB.NET, …)
+ Target-language independent (C, Pascal, Swift, …)

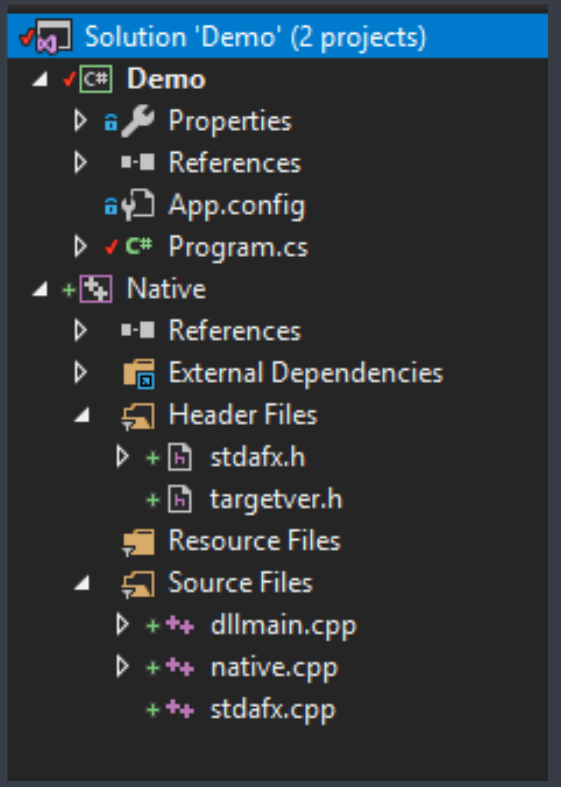
```
[DllImport("libc.so.6")]
static extern int getpid();


public static void Main() => Console.WriteLine($"My PID:  0x{getpid():x8}");
```

P/Invoke – Contras

- Project divided into two (e.g. C++ and C#)
- Must update signatures on both sides
- Must keep track of library file names

PI	Ordinal ^	Hint	Function	Entry Point
C	N/A	23 (0x0017)	CsrCaptureMessageString	Not Bound
C	N/A	25 (0x0019)	CsrClientCallServer	Not Bound
C	N/A	27 (0x001B)	CsrFreeCaptureBuffer	Not Bound
C	N/A	31 (0x001F)	CsrVerifyRegion	Not Bound
C	N/A	33 (0x0021)	DbgPrint	Not Bound
C	N/A	34 (0x0022)	DbgPrintEx	Not Bound

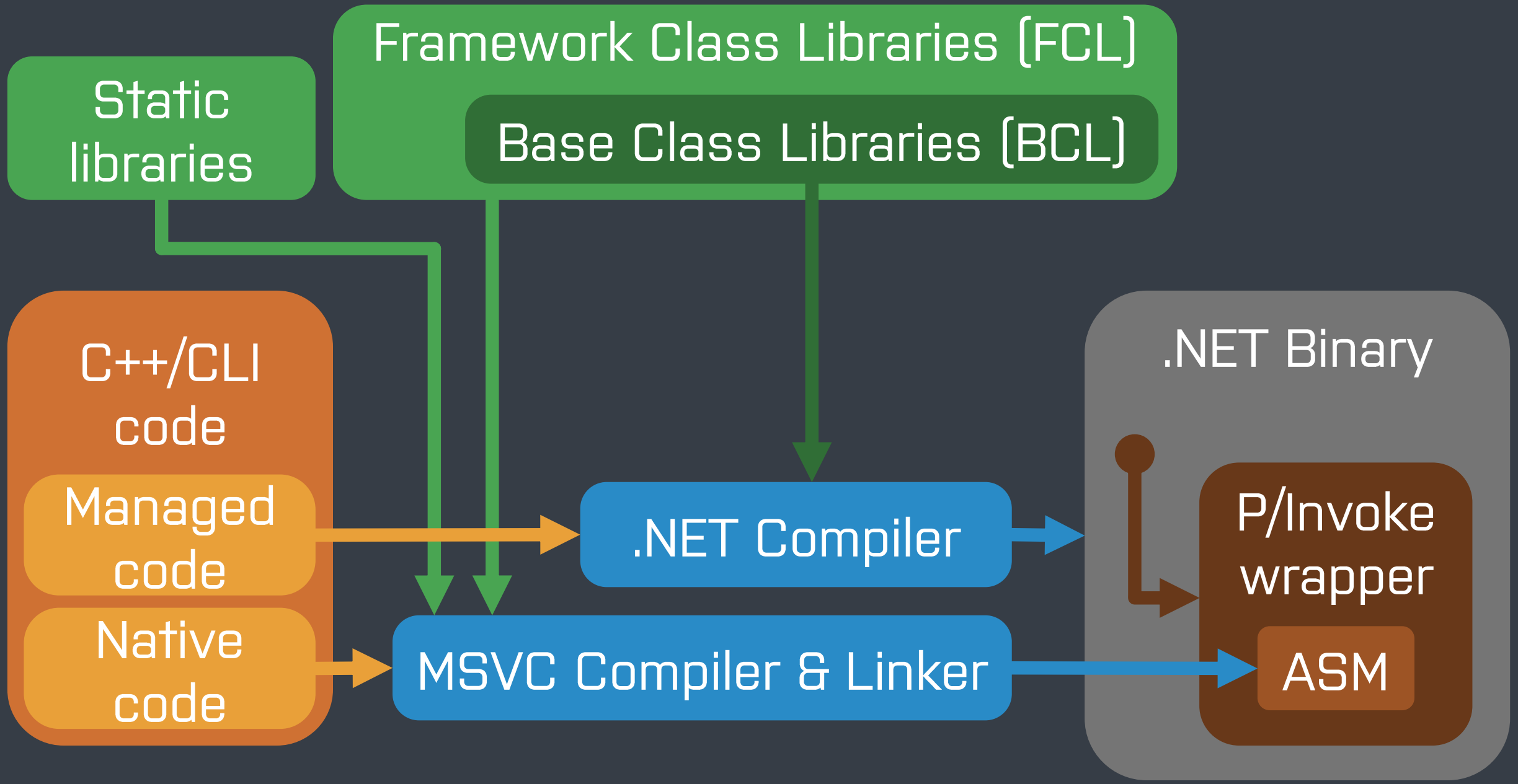
E	Ordinal ^	Hint	Function	Entry Point
O#	8 (0x0008)	N/A	N/A	0x000849E0
C	9 (0x0009)	0 (0x0000)	A_SHAFinal	0x000559B0
C	10 (0x000A)	1 (0x0001)	A_SHAInit	0x000567E0
C	11 (0x000B)	2 (0x0002)	A_SHAUpdate	0x00056820
C	12 (0x000C)	3 (0x0003)	AlpcAdjustCompletionListConcurrencyCount	0x000DFF40
C	13 (0x000D)	4 (0x0004)	AlpcFreeCompletionListMessage	0x000DFF70
C	14 (0x000E)	5 (0x0005)	AlpcGetCompletionListLastMessageInformation	0x000E0080
C	15 (0x000F)	6 (0x0006)	AlpcGetCompletionListMessageAttributes	0x000E00A0
C	16 (0x0010)	7 (0x0007)	AlpcGetHeaderSize	0x00075F70



C++/CLI

C++/CLI

- Replaces Managed C++
- Combines C++ with .NET technologies (BCL, GC, etc.)
- C++-flavoured syntax
- Compiles to mixed assembly (e.g. CIL and x86-ASM)
- CLR-types
- Has managed pointers/handles and native pointers


Framework Class Libraries (FCL)
Base Class Libraries (BCL)
Static libraries
C++/CLI code
Managed code
Native code
.NET Compiler
MSVC Compiler & Linker
.NET Binary
P/Invoke wrapper
ASM

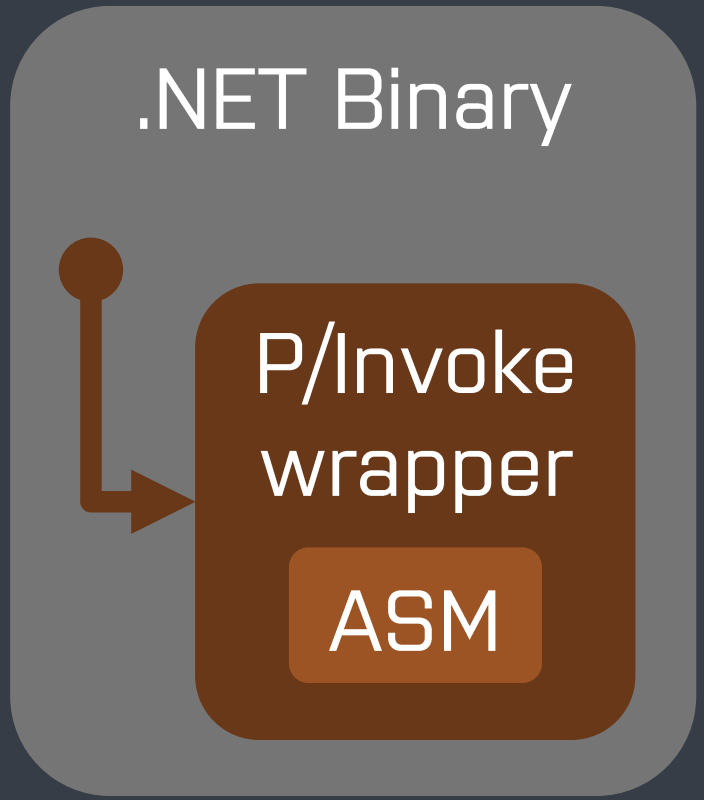
C++/CLI – Mixed Assemblies

- Generated when passing the option `/clr` to the compiler, e.g.:

```
cl /clr main.cpp
```

- Native code sections are architecture-dependent
- Internal P/Invoke-calls are implemented using:

```
[MethodImpl(MethodImplOptions.InternalCall)]
private static extern int Add(int a, int b);
```



C++/CLI

IDE:

- Windows, MacOS: Visual Studio
- Linux: Visual Studio Code

Development on Linux and MacOS are not advised (buggy),
C++/CLI applications can be run using Mono

C++/CLI – A QUICK INTRODUCTION

```
#using <System.Drawing.dll>

using namespace System::Drawing;

int main()
{
    Bitmap^ b = gcnew Bitmap(1920, 1080);
    Graphics^ g = Graphics::FromImage(b);
    Rectangle rect(50, 50, 200, 100);

    g->DrawRectangle(Pens::Red, rect);
    b->Save(L"my_bitmap.png");

    delete g; // optional
    delete b; // optional
}
```

Include .NET dependencies with **#using**

C++/CLI – A QUICK INTRODUCTION

```
#using <System.Drawing.dll>

using namespace System::Drawing;

int main()
{
    Bitmap^ b = gcnew Bitmap(1920, 1080);
    Graphics^ g = Graphics::FromImage(b);
    Rectangle rect(50, 50, 200, 100);

    g->DrawRectangle(Pens::Red, rect);
    b->Save(L"my_bitmap.png");

    delete g; // optional
    delete b; // optional
}
```

.NET namespaces

C++/CLI – A QUICK INTRODUCTION

```
#using <System.Drawing.dll>

using namespace System::Drawing;

int main()
{
    Bitmap^ b = gcnew Bitmap(1920, 1080);
    Graphics^ g = Graphics::FromImage(b);
    Rectangle rect(50, 50, 200, 100);

    g->DrawRectangle(Pens::Red, rect);
    b->Save(L"my_bitmap.png");

    delete g; // optional
    delete b; // optional
}
```

Managed class with ^ and **gcnew(...)**

Alternatively: ***ref new(...)***

C++/CLI – A QUICK INTRODUCTION

```
#using <System.Drawing.dll>

using namespace System::Drawing;

int main()
{
    Bitmap^ b = gcnew Bitmap(1920, 1080);
    Graphics^ g = Graphics::FromImage(b);
    Rectangle rect(50, 50, 200, 100);

    g->DrawRectangle(Pens::Red, rect);
    b->Save(L"my_bitmap.png");

    delete g; // optional
    delete b; // optional
}
```

.NET structures created as "usual"

C++/CLI – A QUICK INTRODUCTION

```
#using <System.Drawing.dll>

using namespace System::Drawing;

int main()
{
    Bitmap^ b = gcnew Bitmap(1920, 1080);
    Graphics^ g = Graphics::FromImage(b);
    Rectangle rect(50, 50, 200, 100);

    g->DrawRectangle(Pens::Red, rect);
    b->Save(L"my_bitmap.png");

    delete g; // optional
    delete b; // optional
}
```

Managed member access with `->`

C++/CLI – A QUICK INTRODUCTION

```
#using <System.Drawing.dll>

using namespace System::Drawing;

int main()
{
    Bitmap^ b = gcnew Bitmap(1920, 1080);
    Graphics^ g = Graphics::FromImage(b);
    Rectangle rect(50, 50, 200, 100);

    g->DrawRectangle(Pens::Red, rect);
    b->Save(L"my_bitmap.png");

    delete g; // optional
    delete b; // optional
}
```

.NET UTF-16 string with L prefix

C++/CLI – A QUICK INTRODUCTION

```
#using <System.Drawing.dll>

using namespace System::Drawing;

int main()
{
    Bitmap^ b = gcnew Bitmap(1920, 1080);
    Graphics^ g = Graphics::FromImage(b);
    Rectangle rect(50, 50, 200, 100);

    g->DrawRectangle(Pens::Red, rect);
    b->Save(L"my_bitmap.png");

    delete g; // optional
    delete b; // optional
}
```

Generally not needed with .NET objects

*Only when **`IDisposable`** is implemented*

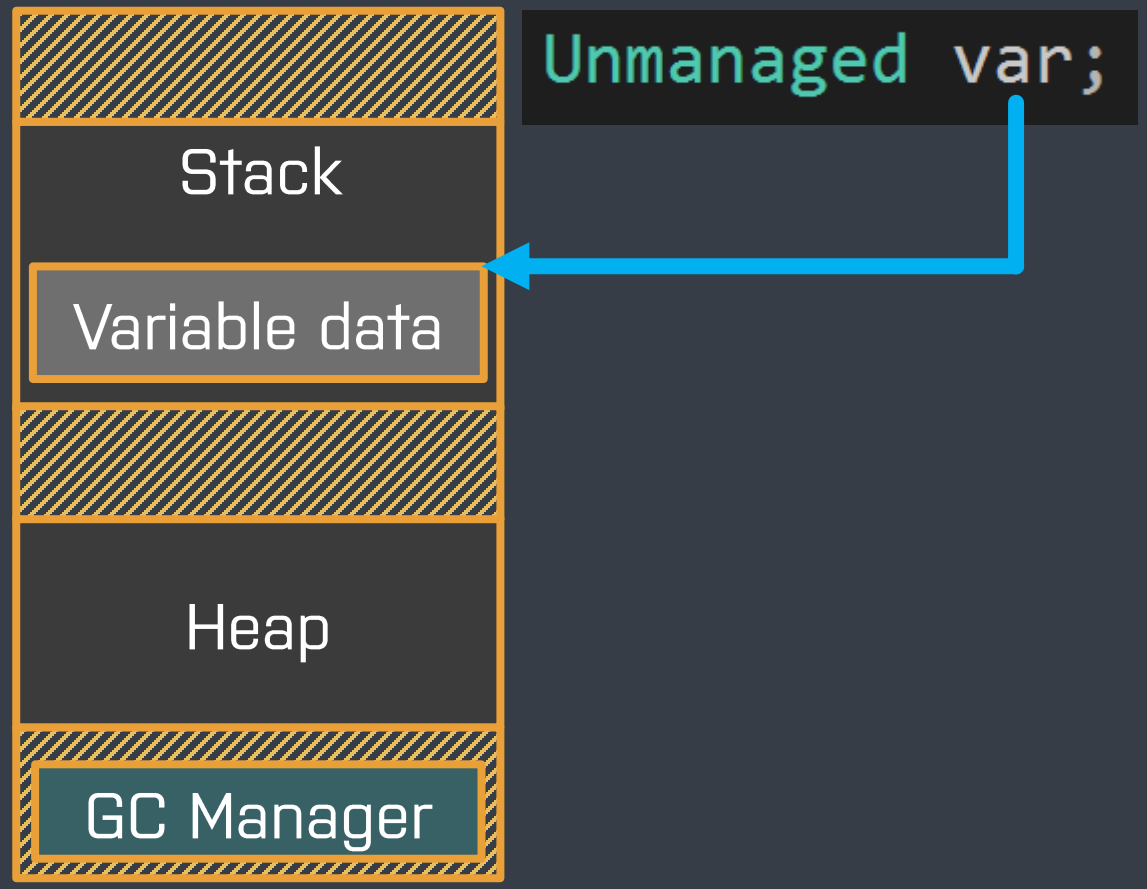
C++/CLI – A QUICK INTRODUCTION

```
#using <System.Drawing.dll>

using namespace System::Drawing;

int main()
{
    Bitmap^ b = gcnew Bitmap(1920, 1080);
    Graphics^ g = Graphics::FromImage(b);
    Rectangle rect(50, 50, 200, 100);

    g->DrawRectangle(Pens::Red, rect);
    b->Save(L"my_bitmap.png");

    delete g; // optional
    delete b; // optional
}
```

Variables are automatically destroyed after leaving their scope

C++/CLI – Variables

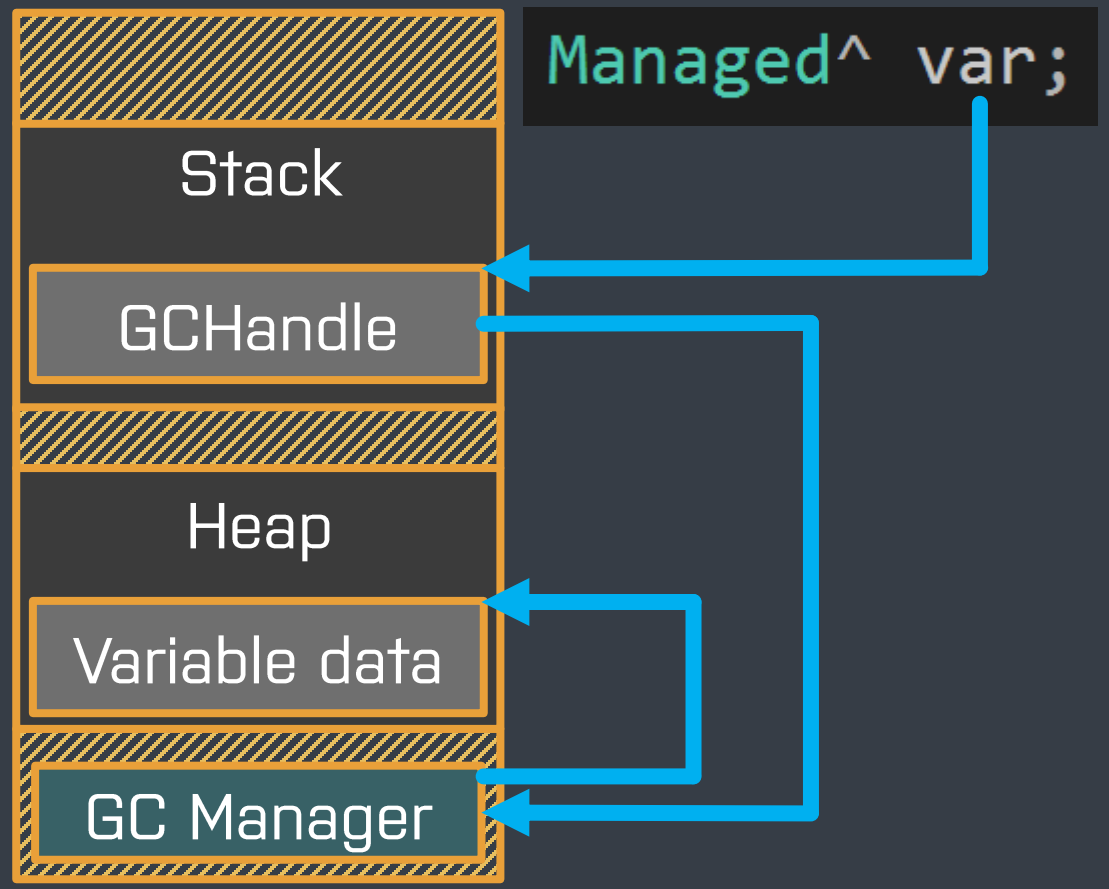
Unmanaged Variable




```
struct Unmanaged
{
public:
    int X;
};
```

C++/CLI – Variables

Managed Variable




```
ref class Managed
{
public:
    int X;
};
```

C++/CLI – Variables

- **GCHandle**: passed by value
- Data semantically passed by reference
- Use **%** for tracking reference
  = **&**-parameter in C++
  = **out**-/**ref**-parameter in C#

```
ref class Managed
{
public:
    int X;

    Managed(int x) { this->X = x; }
};

void iif(bool cond, Managed^% out, Managed^ a, Managed^ b)
{
    out = cond ? a : b;
}

void main(void)
{
    Managed^ a = gcnew Managed(42);
    Managed^ b = gcnew Managed(315);
    Managed^ result;

    iif(false, result, a, b);
}
```

C++/CLI – Variables

```
ref class Managed
{
public:
    int X;

    Managed(int x) { this->X = x; }
};

void iif(bool cond, Managed^% out, Managed^ a, Managed^ b)
{
    out = cond ? a : b;
}

void main(void)
{
    Managed^ a = gcnew Managed(42);
    Managed^ b = gcnew Managed(315);
    Managed^ result;

    iif(false, result, a, b);
}
```

```
class Native
{
public:
    int X;

    Native(int x) { this->X = x; }
};

void iif(bool cond, Native& out, Native a, Native b)
{
    out = cond ? a : b;
}

void main(void)
{
    Native a = Native(42);
    Native b = Native(315);
    Native result = 0;

    iif(false, result, a, b);
}
```

C++/CLI vs C# – A quick example

C++/CLI

```
void onclicked(Object^ sender, EventArgs^ e)
{
    MessageBox::Show(L"You clicked on the Window!");
}

int main(array<String^>^ args)
{
    Form^ f = gcnew Form();

    f->Text = L"Test Window";
    f->Size = Size(100, 100);
    f->Click += gcnew EventHandler(&onclicked);
    f->ShowDialog();

    return 0;
}
```

C#

```
static void onclicked(object sender, EventArgs e)
{
    MessageBox.Show("You clicked on the Window!");
}

static int Main(string[] args)
{
    Form f = new Form();

    f.Text = "Test Window";
    f.Size = new Size(100, 100);
    f.Click += onclicked;
    f.ShowDialog();

    return 0;
}
```

C++/CLI – Managed classes

```
public ref class GraphicsFetcher
{
private:
    Graphics^ gr;
    Form^ form;
public:
    property Graphics^ WindowGraphics { Graphics^ get(); };
    property void* WindowHandle { void* get(); };

    GraphicsFetcher(Form^);
    ~GraphicsFetcher();

    void DrawRectangle(Point, Size, Color);
};
```

C++/CLI – Managed classes

```
GraphicsFetcher::GraphicsFetcher(Form^ f)
{
    form = f;
    gr = Graphics::FromHwnd(f->Handle);
}

GraphicsFetcher::~GraphicsFetcher()
{
    gr->Dispose();
}

Graphics^ GraphicsFetcher::WindowGraphics::get() { return gr; }

void* GraphicsFetcher::WindowHandle::get() { return (void*)form->Handle; }

void GraphicsFetcher::DrawRectangle(Point p, Size s, Color c)
{
    if (gr != nullptr)
        gr->FillRectangle(gcnew SolidBrush(c), Rectangle(p, s));
    else
        throw gcnew InvalidOperationException("Underlying graphics are disposed.");
}
```

Deterministic
.NET deconstructor

C++/CLI – Managed classes

```
GraphicsFetcher::GraphicsFetcher(Form^ f)
{
    form = f;
    gr = Graphics::FromHwnd(f->Handle);
}

GraphicsFetcher::~GraphicsFetcher()
{
    gr->Dispose();
}

Graphics^ GraphicsFetcher::WindowGraphics::get() { return gr; }

void* GraphicsFetcher::WindowHandle::get() { return (void*)form->Handle; }

void GraphicsFetcher::DrawRectangle(Point p, Size s, Color c)
{
    if (gr != nullptr)
        gr->FillRectangle(gcnew SolidBrush(c), Rectangle(p, s));
    else
        throw gcnew InvalidOperationException("Underlying graphics are disposed.");
}
```

Read-only properties

C++/CLI – Managed classes

```
GraphicsFetcher::GraphicsFetcher(Form^ f)
{
    form = f;
    gr = Graphics::FromHwnd(f->Handle);
}

GraphicsFetcher::~GraphicsFetcher()
{
    gr->Dispose();
}

Graphics^ GraphicsFetcher::WindowGraphics::get() { return gr; }

void* GraphicsFetcher::WindowHandle::get() { return (void*)form->Handle; }

void GraphicsFetcher::DrawRectangle(Point p, Size s, Color c)
{
    if (gr != nullptr)
        gr->FillRectangle(gcnew SolidBrush(c), Rectangle(p, s));
    else
        throw gcnew InvalidOperationException("Underlying graphics are disposed.");
}
```

Equivalent to C#'s `null`

Translating C# to C++/CLI

```
readonly string MaybeString = null;
readonly double? MaybeDouble = null;
readonly int? MaybeInt = 315;
```

```
initonly String^ MaybeString = nullptr;
initonly Nullable<double> MaybeDouble();
initonly Nullable<int> MaybeInt = Nullable<int>(315);
```

Translating C# to C++/CLI

```
readonly string MaybeString = null;
readonly double? MaybeDouble = null;
readonly int? MaybeInt = 315;
```

```
initonly String^ MaybeString = nullptr;
initonly Nullable<double> MaybeDouble();
initonly Nullable<int> MaybeInt = Nullable<int>(315);
```

```
const int I = 42;
const string S = "/foo/bar";
```

```
literal int I = 42;
literal String^ S = L"/foo/bar";
```

Translating C# to C++/CLI

```
readonly string MaybeString = null;
readonly double? MaybeDouble = null;
readonly int? MaybeInt = 315;
```

```
initonly String^ MaybeString = nullptr;
initonly Nullable<double> MaybeDouble();
initonly Nullable<int> MaybeInt = Nullable<int>(315);
```

```
const int I = 42;
const string S = "/foo/bar";
```

```
literal int I = 42;
literal String^ S = L"/foo/bar";
```

```
object mutex = new object();

lock (mutex)
{
    // ...
}
```

```
Object^ mutex = gcnew Object();

Monitor::Enter(mutex);
// ...
Monitor::Exit(mutex);
```

C++/CLI – Casting Objects

```
class Parent
{
};

class Child : Parent
{
};
```

```
Parent x = new Child();

if (x is Child)
{
    Child c = (Child)x;

    Console.WriteLine("X is of the type 'Child'");
}

Child maybe = x as Child;

if (maybe == null)
    Console.WriteLine("X is not of the type 'Child'");
```

```
Parent^ x = gcnew Child();

if (dynamic_cast<Child^>(x) != nullptr)
{
    Child^ c = (Child^)x;
    // alternative:
    c = safe_cast<Child^>(x);

    Console::WriteLine("X is of the type 'Child'");
}

Child^ maybe = dynamic_cast<Child^>(x);

if (maybe == nullptr)
    Console::WriteLine("X is not of the type 'Child'");
```

C++/CLI – Casting Objects

```
static_cast<T^>(x);
dynamic_cast<T^>(x);
safe_cast<T^>(x);
(T^)x;
```

Very unsafe cast (but fast)

```
z = x is T;
z = x as T;
z = (T)x;
```

```
dynamic_cast<T^>(x) != nullptr;
```

Checks whether **x** is of type **T**

Used for inheritance

C++/CLI – Casting Objects

```
static_cast<T^>(x);
dynamic_cast<T^>(x);
safe_cast<T^>(x);
(T^)x;
```

If **x** has the type **T**,
then it returns **x** cast as **T**
Otherwise
null

```
z = x is T;
z = x as T;
z = (T)x;
```

Used for inheritance

C++/CLI – Casting Objects

```
static_cast<T^>(x);
dynamic_cast<T^>(x);
safe_cast<T^>(x);
(T^)x;
```

If **x** has the type **T**,
then it returns **x** cast as **T**
Otherwise
throws an **InvalidCastException**

```
z = x is T;
z = x as T;
z = (T)x;
```

Used for inheritance and user-defined casts

C++/CLI – Casting Objects

```
value class Point
{
private:
    initonly int X, Y;

public:
    Point(int x, int y)
    {
        this->X = x;
        this->Y = y;
    }

    static property Point Zero { Point get() { return Point(0, 0); } };

    static operator long(Point p)
    {
        return ((long)p.X << 32) | p.Y;
    }

    static explicit operator Point(long l)
    {
        return Point((int)(l >> 32), (int)(l & 0xffffffffL));
    }
};
```

```
Point p1 = new Point(315, -42);

long l = p1;
Point p2 = (Point)l;
Point p3 = l;
```

C++/CLI – Casting Objects

```
value class Point
{
private:
    initonly int X, Y;

public:
    Point(int x, int y)
    {
        this->X = x;
        this->Y = y;
    }

    static property Point Zero { Point get() { return Point(0, 0); } };

    static operator long(Point p)
    {
        return ((long)p.X << 32) | p.Y;
    }

    static explicit operator Point(long l)
    {
        return Point((int)(l >> 32), (int)(l & 0xffffffffL));
    }
};
```

```
Point p1 = new Point(315, -42);

long l = p1;
Point p2 = (Point)l;
Point p3 = l;
```

C++/CLI – Casting Objects

```
value class Point
{
private:
    initonly int X, Y;

public:
    Point(int x, int y)
    {
        this->X = x;
        this->Y = y;
    }

    static property Point Zero { Point get() { return Point(0, 0); } };

    static operator long(Point p)
    {
        return ((long)p.X << 32) | p.Y;
    }

    static explicit operator Point(long l)
    {
        return Point((int)(l >> 32), (int)(l & 0xffffffffL));
    }
};
```

```
Point p1 = new Point(315, -42);

long l = p1;
Point p2 = (Point)l;
Point p3 = l;
```

"Cannot convert **long** to **Point**. An explicit cast exists. (Are you missing a cast?)"

C++/CLI – Operator Overloading

Overloadable operators:

- Unary `!`, `+`, `-`, `++`, `--`, `~`, `true`, `false`
- Binary `!=`, `==`, `<`, `<=`, `>`, `>=`, `|`, `&`, `^`, `||`, `&&`, `<<`, `>>`, `+`, `-`, `*`, `/`, `%`

Assignments and functors cannot be overloaded

C++/CLI – Operator Overloading

```
public ref struct Complex
{
private:
    double _r, _i;

public:
    Complex(double re, double im) {
        _r = re;
        _i = im;
    }
}
```

```
static Complex^ operator+(Complex^ c, double real) {
    return gcnew Complex(c->_r + real, c->_i);
}

static Complex^ operator++(Complex^ c) {
    return c + 1.0;
}

Complex^ operator*(double f) {
    return gcnew Complex(this->_r * f, this->_i * f);
}

Complex^ operator--() {
    return this + (-1.0);
}
```

C++/CLI – Properties

- Exterior semantic of fields
- Implemented through (internal) methods
- Can be auto-implemented

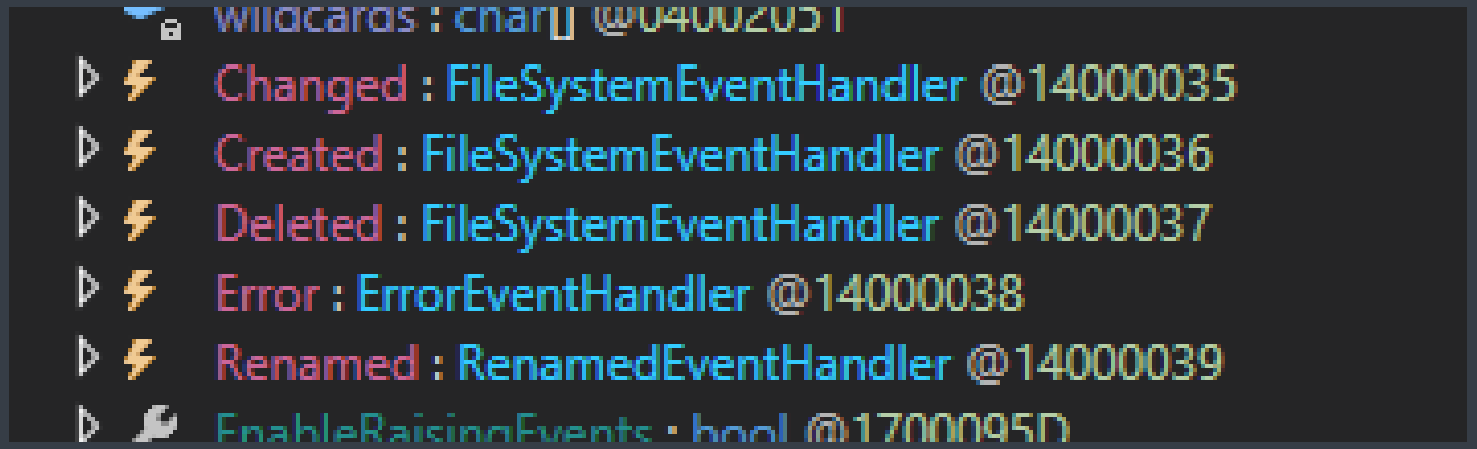
```
ref class SomeData
{
private:
    int _value;
    String^ _name;

public:
    property double SomeFloat;
    property int Value { int get() { return _value; } }
    property String^ Name { String^ get(); void set(String^); }
};
```

```
String^ SomeData::Name::get()
{
    return this->_name;
}

void SomeData::Name::set(String^ s)
{
    if (s != nullptr)
        this->_name;
}
```

C++/CLI – Events and Delegates

.NET Events have a subscriber-notifier-system:

- Methods can be subscribed to an event
- Raising an event notifies each subscriber
- Unsubscription removes Methods from notification list



C++/CLI – Events and Delegates

```
delegate void FileDelegate(FileInfo^);

ref class FileSysWatcher
{
public:
    event FileDelegate^ OnCreate;
    event FileDelegate^ OnDelete;

private:
    void WatchLoop()
    {
        for each (FileInfo^ file in directory)
        {
            // if (file->has_been_deleted) {
                OnDelete(file);
            // }
        }
    }
};
```

Function delegate definition

C++/CLI – Events and Delegates

```
delegate void FileDelegate(FileInfo^);

ref class FileSysWatcher
{
public:
    event FileDelegate^ OnCreate;
    event FileDelegate^ OnDelete;

private:
    void WatchLoop()
    {
        for each (FileInfo^ file in directory)
        {
            // if (file->has_been_deleted) {
                OnDelete(file);
            // }
        }
    }
};
```

Event definition

C++/CLI – Events and Delegates

```
delegate void FileDelegate(FileInfo^);

ref class FileSysWatcher
{
public:
    event FileDelegate^ OnCreate;
    event FileDelegate^ OnDelete;

private:
    void WatchLoop()
    {
        for each (FileInfo^ file in directory)
        {
            // if (file->has_been_deleted) {
                OnDelete(file);
            // }
        }
    }
};
```

Event calling notifies all event subscribers (in order of subscription)

C++/CLI – Events and Delegates

```
FileSysWatcher^ watcher = gcnew FileSysWatcher(L"C:/my_folder");

watcher->OnCreate += gcnew FileDelegate(&oncreate);
watcher->StartWatching();
```

Event subscription

```
void oncreate(FileInfo^ file)
{
    Console::WriteLine("The file " + file + "has been created!");
}
```

C++/CLI – Pinned Arrays

- CLR manages arrays on the heap (with GC)
- .NET arrays must be pinned for pointer operations:

```
array<int>^ values = gcnew array<int> { 0, 1, 2, 3, 4, 5 };
pin_ptr<int> pin = &values[0];
int* ptr = pin;

// [optional] frees the pin
pin = nullptr;
```

- Destroying the pin is not necessary

C++/CLI – Usage cases

C++/CLI should be used when one of the following are required:

- .NET support for often-changing APIs
- Complex .NET and native integration
- (Ultra-)high-performance in .NET applications
- Low-level functions in .NET applications

C++/CLI – Pros and Contras

+ Perfect and smooth integration between managed and native
+ Syntax-familiarity for C++-developers
+ Cross-platform support (Mono on Linux/MacOS)
+ Full IntelliSense support (as opposed to P/Invoke)
- No good non-Windows compiler
- Microsoft announced recently discontinuation
- Bulky syntax compared to C#
- No WPF support

Useful Resources

Try C# and F# online:	https://sharplab.io
.NET platform:	https://dot.net
C++/CLI Documentation:	https://docs.microsoft.com/en-us/cpp/dotnet/dotnet-programming-with-cpp-cli-visual-cpp
Visual Studio:	https://visualstudio.com/
P/Invoke Database:	https://pinvoke.net/

Useful Resources

My GitHub:	https://github.com/Unknown6656
My Email Address:	david.funke@iosb.fraunhofer.de
	david.funke@student.kit.edu
Roslyn compiler repo:	https://github.com/dotnet/roslyn
C# language design:	https://github.com/dotnet/csharplang
F# repository:	https://github.com/fsharp/fsharp
Cosmos OS:	https://www.gocosmos.org/