C++/WinRT and the future of C++ on Windows

SCOTT JONES & KENNY KERR

MICROSOFT

What’s this all about

Replacing proprietary extensions with standard C++ code

Making Windows a great place for C++ developers

Benefiting from advanced C++17 and TS features

Providing tooling C++ developers love to use

What’s coming up next

Looking into the future

What is WinRT & C++/WinRT

Windows Runtime

- Metadata (.winmd files)
- Language projections (C++, C#, JavaScript)
- Windows ABI

C++/WinRT

- Standard and modern C++
- Header-only library
- Classy type system

Who’s using C++/WinRT

Microsoft Windows

Microsoft Office

Adobe Photoshop

Spotify

Many more!

From proprietary… to standard

C++/CX	C++/WinRT
__declspec(uuid) & __uuidof(T)	C++ constexpr guid_of<T>
Precompiled headers	C++ modules
IDL and MIDL	C++ reflection and metaclasses
Dude, whatever compiles ☺	/permissive- and Clang

1. Get pictures folder.

```
IAsyncAction Sample()
{
  auto library = co_await StorageLibrary::GetLibraryAsync(KnownLibraryId::Pictures);
  auto folder = library.SaveFolder();
```

2. Create file.

```
  auto file = co_await folder.CreateFileAsync(L"cppcon.jpg",
                                              CreationCollisionOption::ReplaceExisting);
```

3. Prepare webcam.

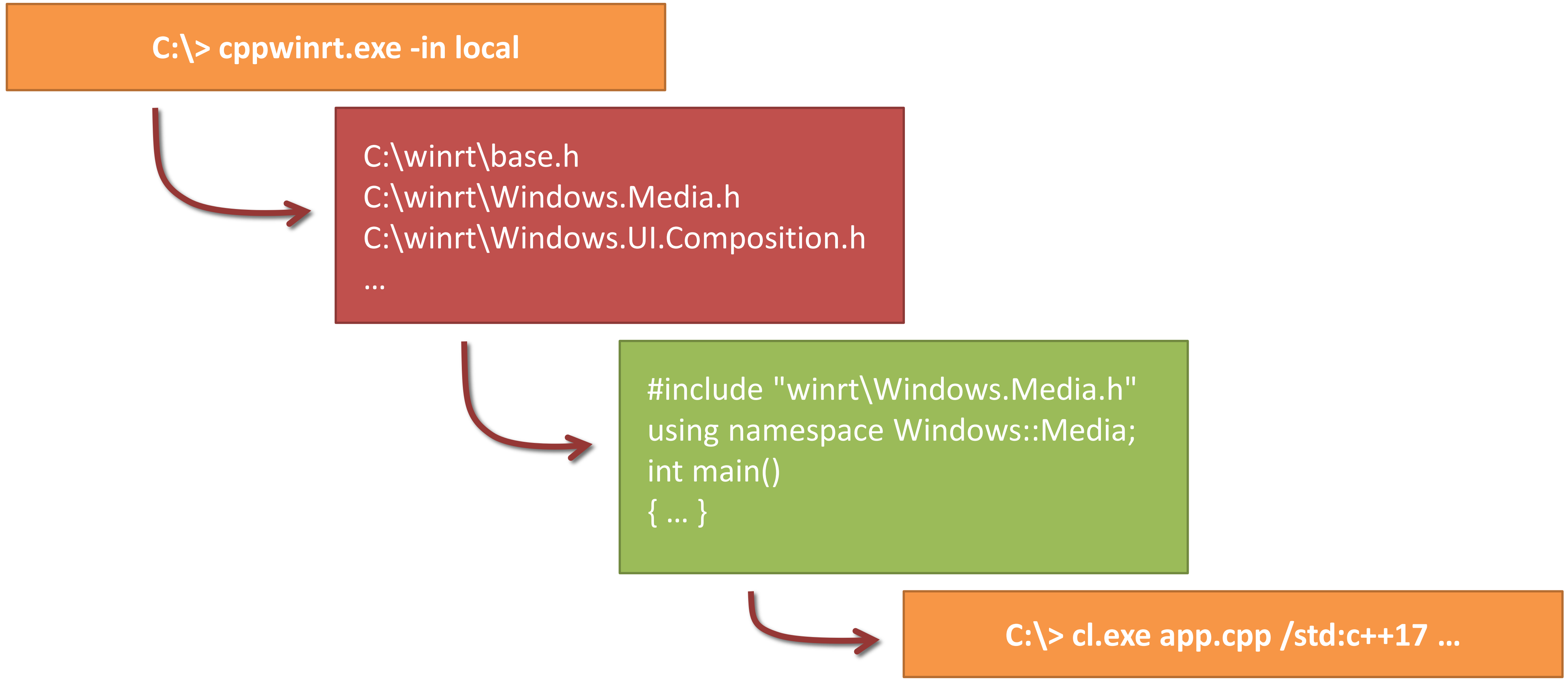
```
  MediaCapture capture;
  co_await capture.InitializeAsync();
```

4. Capture photo!

```
  auto props = ImageEncodingProperties::CreateJpeg();
  co_await capture.CapturePhotoToStorageFileAsync(props, file);
}

int main()
{
  init_apartment();
  Sample().get();
}
```

“Hello webcam”

From cppwinrt.exe to cl.exe



```
C:\> cppwinrt.exe -in local
```

```
C:\winrt\base.h
C:\winrt\Windows.Media.h
C:\winrt\Windows.UI.Composition.h
…
```

```
#include "winrt\Windows.Media.h"
using namespace Windows::Media;
int main()
{ … }
```

```
C:\> cl.exe app.cpp /std:c++17 …
```

Demo time!

“FROM WINDOWS WITH LOVE”

Benefiting from C++17 and TS features

```
namespace winrt
{
    namespace Windows
    {
        namespace UI
        {
            namespace Composition
            {
                struct AmbientLight
                {
                    // ...
                };
            }
        }
    }
}
```

Nested namespace definitions

```cpp
namespace winrt::Windows::UI::Composition
{
    struct AmbientLight
    {
        // ...
    };
}
```

Nested namespace definitions

```
template <typename T>
struct com_array : array_view<T>
{
    void clear() noexcept
    {
        if (this->m_data == nullptr) { return; }

        for (value_type& v : *this)
        {
            v.~value_type();
        }

        CoTaskMemFree(this->m_data);
        this->m_data = nullptr;
        this->m_size = 0;
    }
```

Almost destroying stuff with C++03

```
template <typename T>
struct com_array : array_view<T>
{
    void clear() noexcept
    {
        if (this->m_data == nullptr) { return; }
        destroy(std::is_trivially_destructible<value_type>());
        CoTaskMemFree(this->m_data);
        this->m_data = nullptr;
        this->m_size = 0;
    }
private:
    void destroy(std::true_type) noexcept
    { /* do nothing */ }
    void destroy(std::false_type) noexcept
    {
        for (value_type& v : *this)
        {
            v.~value_type();
        }
    }
```

Destroying stuff with C++11

```
template <typename T>
struct com_array : array_view<T>
{
    void clear() noexcept
    {
        if (this->m_data == nullptr) { return; }

        std::destroy(this->begin(), this->end());

        CoTaskMemFree(this->m_data);
        this->m_data = nullptr;
        this->m_size = 0;
    }
```

Even simpler!

Destroying stuff with C++17

```
struct Sample : implements<Sample, IStringable, IClosable>
{
    hstring ToString()
    {
        return L"Hello world";
    }

    void Close()
    {
    }
};
```

IUnknown
IInspectable
IAgileObject
IMarshal
IWeakReferenceSource

Logical operator traits

```
struct Sample : implements<Sample, IStringable, IClosable>
{
    hstring ToString()
    {
        return L"Hello world";
    }

    void Close()
    {
    }
};
```

Am I agile? Yes!

Logical operator traits

```cpp
struct Sample : implements<Sample, IStringable, IClosable, non_agile>
{
    hstring ToString()
    {
        return L"Hello world";
    }

    void Close()
    {
    }
};
```

Am I agile? No.

Logical operator traits

```
struct Sample : implements<Sample, IStringable, non_agile, IClosable>
{
    hstring ToString()
    {
        return L"Hello world";
    }

    void Close()
    {
    }
};
```

Am I agile? Still no.

Logical operator traits

```
struct Sample : implements<Sample, non_agile, IStringable, IClosable>
{
    hstring ToString()
    {
        return L"Hello world";
    }

    void Close()
    {
    }
};
```

Am I agile? Stop it.

Logical operator traits

```cpp
using is_agile = std::negation<std::disjunction<std::is_same<non_agile, I> ...>>;
```

Is I non_agile?

$!(\,..I_1> \text{OR}\ ..I_2> \text{OR}\ ..I_N>)$

Logical operator traits

```
using is_factory = std::disjunction<std::is_same<IActivationFactory, I> ...>;

using is_inspectable = std::disjunction<std::is_base_of<IInspectable, I> ...>;

using is_weak_ref_source =
    std::conjunction<is_inspectable,
                     std::negation<is_factory>,
                     std::negation<std::disjunction<std::is_same<no_weak_ref, I> ...>>>;
```

is_inspectable && !is_factory && !(no_weak_ref …)

Logical operator traits

Is ‘First’ an interface?

```
template <typename First, typename ... Rest>
void* find_interface(GUID const& id,
                     std::enable_if_t<!is_marker_v<First> &&
                                      !is_implements_v<First>>* = nullptr) const noexcept
{
    if (id == guid_of<First>())
    {
        return to_abi<First>(this);
    }

    return find_interface<Rest ...>(id);
}
```

Then let’s compare its GUID…

and possibly return vptr.

Otherwise we’ll keep looking.

if constexpr

If ‘First’ is *not* an interface…

```
template <typename First, typename ... Rest>
void* find_interface(GUID const& id,
                     std::enable_if_t<is_marker_v<First> ||
                                      is_implements_v<First>>* = nullptr) const noexcept
{
    return find_interface<Rest ...>(id);
}
```

then we’ll keep looking.

if constexpr

??

```cpp
template <int = 0>
void* find_interface(GUID const& id) const noexcept
{
    return base_type::find_interface_override(id);
}
```

Possibly look elsewhere.

if constexpr

```
template <typename First, typename ... Rest>
void* find_interface(GUID const& id) const noexcept
{
    if constexpr (!is_marker_v<First> && !is_implements_v<First>)
    {
        if (id == guid_of<First>())
        {
            return to_abi<First>(this);
        }
    }

    if constexpr (sizeof...(Rest) > 0)
    {
        return find_interface<Rest ...>(id);
    }
    else
    {
        return base_type::find_interface_override(id);
    }
}
```

Is ‘First’ an interface?

If it has a matching GUID then return the vptr.

If there are more type params then keep looking.

Otherwise look elsewhere.

if constexpr

```
struct __declspec(uuid("96369f54-8eb6-48f0-abce-c1b211e627c3"))
    IStringable : IInspectable
{
    virtual HRESULT __stdcall ToString(HSTRING* value) = 0;
};


int main()
{
    GUID guid = __uuidof(IStringable);
}
```

constexpr functions

```
struct Stringable : IStringable
{
    HRESULT __stdcall QueryInterface(GUID const& id, void** object) noexcept override
    {
        if (id == __uuidof(IStringable))
        {
            *object = static_cast<IStringable*>(this);
.
.
.

int main()
{
    IUnknown* object = ...
    IStringable* stringable;
    object->QueryInterface(__uuidof(IStringable), reinterpret_cast<void**>(stringable));
.
.
.
```

constexpr functions

```
struct __declspec(uuid("96369f54-8eb6-48f0-abce-c1b211e627c3"))
    IStringable : IInspectable
{
    virtual HRESULT __stdcall ToString(HSTRING* value) = 0;
};

template <typename T>
struct __declspec(uuid("..."))
    IVector : IInspectable
{
    virtual HRESULT __stdcall Clear() = 0;
.
.
.
```

??

constexpr functions

```cpp
template <typename T>
struct not_specialized_type
{
    static constexpr bool value = false;
};

template <typename T>
struct not_specialized
{
    static_assert(not_specialized_type<T>::value,
        "This generic interface has not been specialized. "
        "Each distinct instantiation of this generic interface requires a GUID. "
        "This GUID must be provided by a template specialization. "
        "Good luck trying to figure out what the value should be! :)");
};

template <typename T> struct IVector : not_specialized<IVector<T>>
{
};
```

constexpr functions

```
template <typename T>
struct not_specialized_type
{
    static constexpr bool value = false;
};

template <typename T>
struct not_specialized
{
    static_assert(not_specialized_type<T>::value,
        "This generic interface has not been specialized. "
        "Each distinct instantiation of this generic interface requires a GUID. "
        "This GUID must be provided by a template specialization. "
        "Good luck trying to figure out what the value should be! :)");
};

template <typename T> struct IVector : not_specialized<IVector<T>>
{
};
```

constexpr functions

```
GUID const& a = guid_of<IStringable>();
```

{96369f54-8eb6-48f0-abce-c1b211e627c3}

```
using mad_container = IMap<IStringable,
                           IMapView<hstring,
                                    IVector<IAsyncOperation<bool>>>>;

GUID const& c = guid_of<mad_container>();
```

{02705479-42fb-514e-87b8-6d4d679cb5e4}

constexpr functions

```
template <>
struct guid<Windows::Foundation::IStringable>
{
    static constexpr GUID value
    { 0x96369F54,0x8EB6,0x48F0,{ 0xAB,0xCE,0xC1,0xB2,0x11,0xE6,0x27,0xC3 } };
};

template <typename K, typename V>
struct guid<Windows::Foundation::Collections::IMap<K, V>>
{
    static constexpr GUID value
    {
        // According to RFC 4122...
        // Generate a string representing the given specialization.
        // Create buffer with big endian GUID and UTF-8 form of string above.
        // Compute SHA1 hash then take first 16 bytes as GUID.
        // Convert GUID to little endian.
        // A bit more bit twiddling just for fun!
    };
};
```

constexpr functions

std::optional

std::variant

std::string_view

[[deprecated]]

__has_include

Coroutines

Fold expressions

Template argument deduction

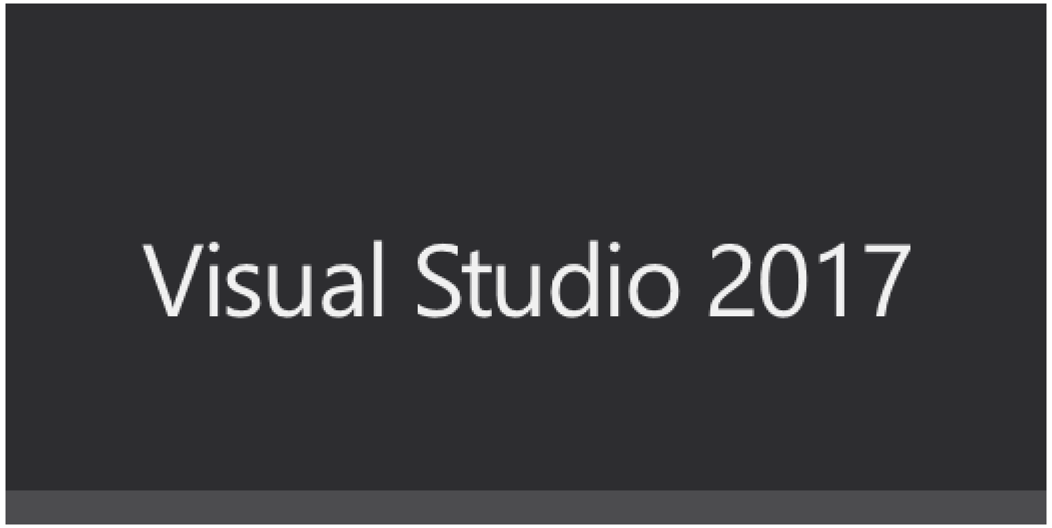
Hopefully soon!

And many more

Developing components

```
C:\> cppwinrt.exe -component -in Sample.winmd -ref local
```

```
C:\Sample\Sample.vcxproj
C:\Sample\module.def
C:\Sample\module.cpp
C:\Sample\module.h
C:\Sample\Chicken.cpp + .h
C:\Sample\Duck.cpp + .h
…
```


Visual Studio 2017

Developing components

Demo time!

“SHARING IS CARING” OR “DLL ALL THE THINGS”

What’s coming soon

Windows build

Windows SDK

XAML compiler

Visual Studio

Looking into the future

Reducing library size and complexity

Reducing binary size

Reducing instruction count

Inline all the things

Talking semantics with the compiler

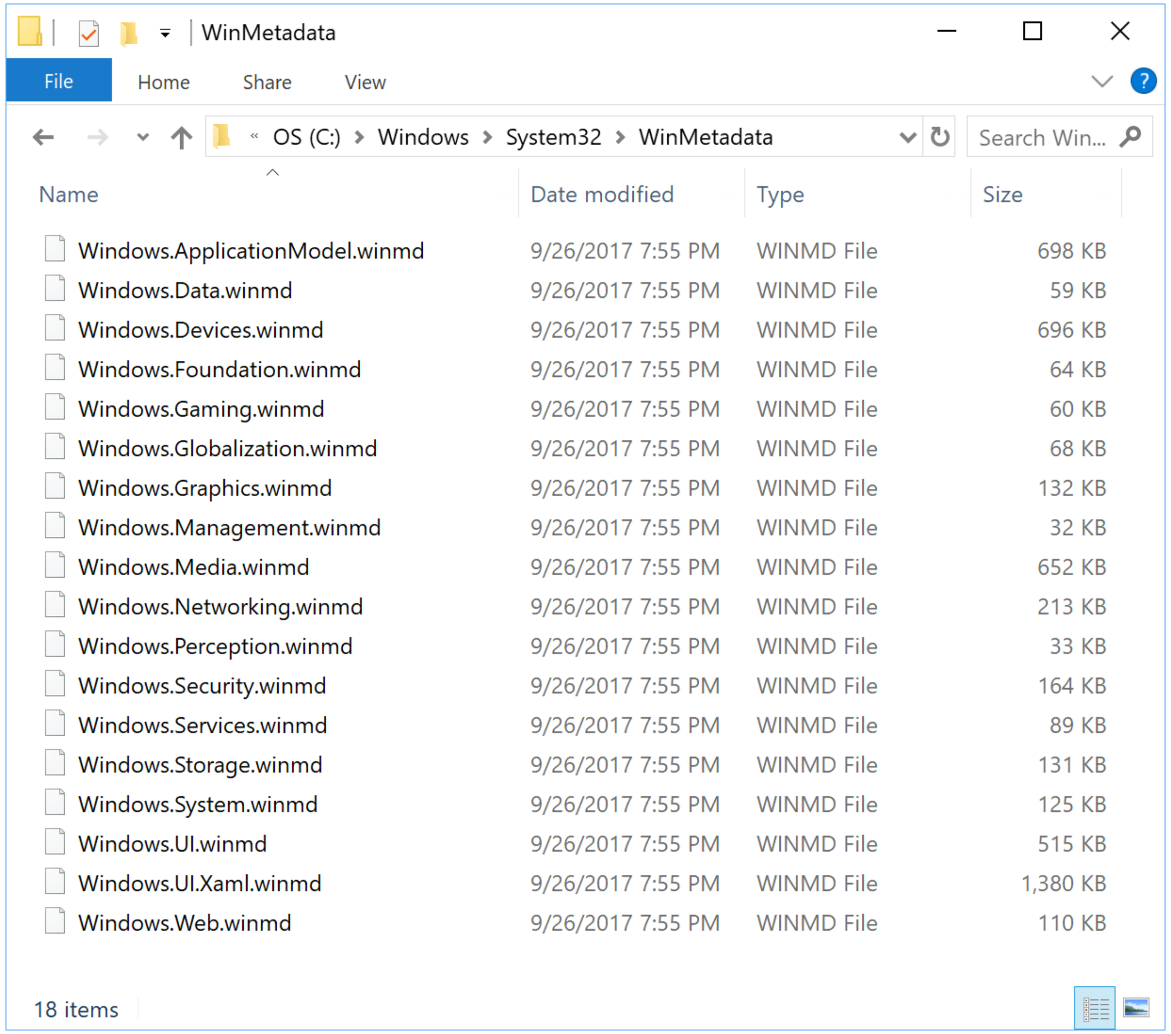
constexpr all the things

Exceptions, exceptions, exceptions

Optimizing for modern C++

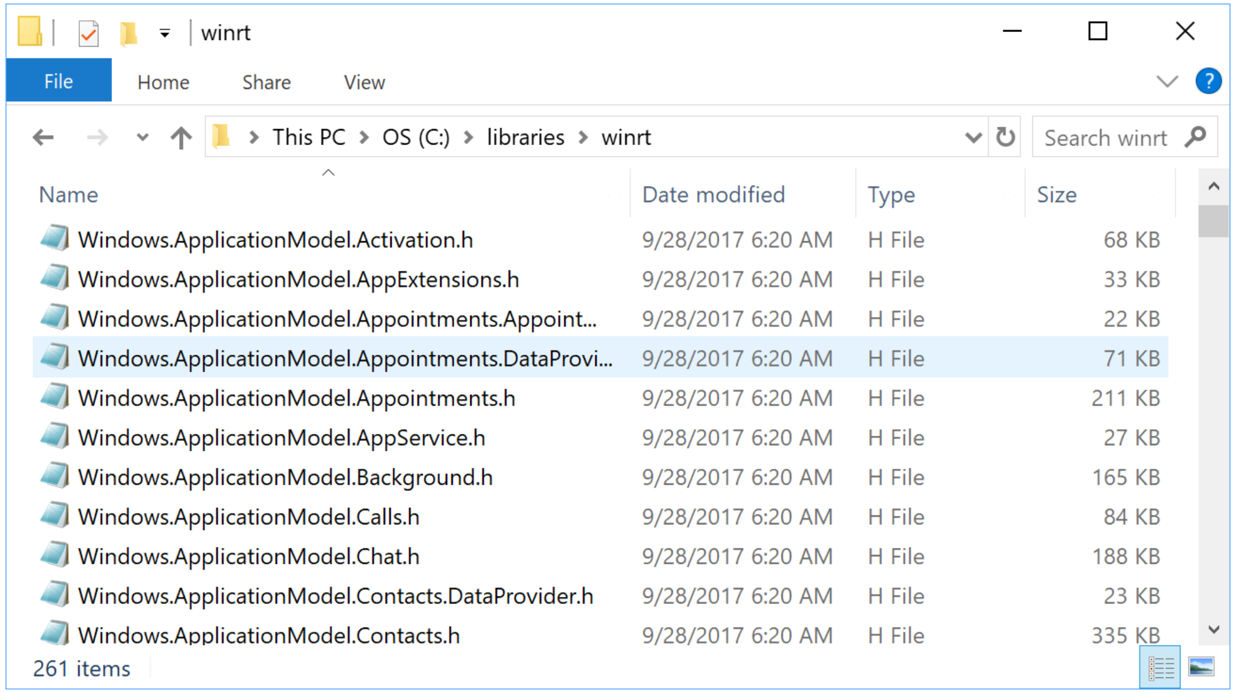
C++ modules

```
A:\> cppwinrt.exe -in local -out C:\libraries
```



Name	Date modified	Type	Size
Windows.ApplicationModel.winmd	9/26/2017 7:55 PM	WINMD File	698 KB
Windows.Data.winmd	9/26/2017 7:55 PM	WINMD File	59 KB
Windows.Devices.winmd	9/26/2017 7:55 PM	WINMD File	696 KB
Windows.Foundation.winmd	9/26/2017 7:55 PM	WINMD File	64 KB
Windows.Gaming.winmd	9/26/2017 7:55 PM	WINMD File	60 KB
Windows.Globalization.winmd	9/26/2017 7:55 PM	WINMD File	68 KB
Windows.Graphics.winmd	9/26/2017 7:55 PM	WINMD File	132 KB
Windows.Management.winmd	9/26/2017 7:55 PM	WINMD File	32 KB
Windows.Media.winmd	9/26/2017 7:55 PM	WINMD File	652 KB
Windows.Networking.winmd	9/26/2017 7:55 PM	WINMD File	213 KB
Windows.Perception.winmd	9/26/2017 7:55 PM	WINMD File	33 KB
Windows.Security.winmd	9/26/2017 7:55 PM	WINMD File	164 KB
Windows.Services.winmd	9/26/2017 7:55 PM	WINMD File	89 KB
Windows.Storage.winmd	9/26/2017 7:55 PM	WINMD File	131 KB
Windows.System.winmd	9/26/2017 7:55 PM	WINMD File	125 KB
Windows.UI.winmd	9/26/2017 7:55 PM	WINMD File	515 KB
Windows.UI.Xaml.winmd	9/26/2017 7:55 PM	WINMD File	1,380 KB
Windows.Web.winmd	9/26/2017 7:55 PM	WINMD File	110 KB

18 items



Name	Date modified	Type	Size
Windows.ApplicationModel.Activation.h	9/28/2017 6:20 AM	H File	68 KB
Windows.ApplicationModel.AppExtensions.h	9/28/2017 6:20 AM	H File	33 KB
Windows.ApplicationModel.Appointments.Appoint...	9/28/2017 6:20 AM	H File	22 KB
Windows.ApplicationModel.Appointments.DataProvi...	9/28/2017 6:20 AM	H File	71 KB
Windows.ApplicationModel.Appointments.h	9/28/2017 6:20 AM	H File	211 KB
Windows.ApplicationModel.AppService.h	9/28/2017 6:20 AM	H File	27 KB
Windows.ApplicationModel.Background.h	9/28/2017 6:20 AM	H File	165 KB
Windows.ApplicationModel.Calls.h	9/28/2017 6:20 AM	H File	84 KB
Windows.ApplicationModel.Chat.h	9/28/2017 6:20 AM	H File	188 KB
Windows.ApplicationModel.Contacts.DataProvider.h	9/28/2017 6:20 AM	H File	23 KB
Windows.ApplicationModel.Contacts.h	9/28/2017 6:20 AM	H File	335 KB

261 items

C++ modules

What’s wrong with headers?

- There are lots of them
- Roughly 1032 files
- About 40MB worth of headers
- They take time to compile
- Precompiled headers to the rescue!

Not really…

- They’re huge
- Not reusable
- Don’t offer isolation
- Often requires /bigobj and x64/x86 cross compiler

C++ modules

```cpp
#include <winrt/Windows.ApplicationModel.Activation.h>
#include <winrt/Windows.Foundation.h>
#include <winrt/Windows.UI.Xaml.Controls.h>
#include <winrt/Windows.UI.Xaml.Media.h>
#include <winrt/Windows.Storage.Streams.h>
#include <winrt/Windows.Graphics.Imaging.h>
#include <winrt/Windows.Media.Ocr.h>
#include <winrt/Windows.Storage.Pickers.h>

using namespace winrt;
using namespace Windows::ApplicationModel::Activation;
using namespace Windows::Foundation;
using namespace Windows::UI;
using namespace Windows::UI::Xaml;
using namespace Windows::UI::Xaml::Controls;
using namespace Windows::UI::Xaml::Media;
using namespace Windows::Storage;
using namespace Windows::Storage::Streams;
using namespace Windows::Graphics::Imaging;
using namespace Windows::Media::Ocr;
using namespace Windows::Storage::Pickers;
```

1GB-3GB .pch

Error prone

Redundant

C++ modules

```cpp
import winrt;
using namespace winrt;

using namespace Windows::ApplicationModel::Activation;
using namespace Windows::Foundation;
using namespace Windows::UI;
using namespace Windows::UI::Xaml;
using namespace Windows::UI::Xaml::Controls;
using namespace Windows::UI::Xaml::Media;
using namespace Windows::Storage;
using namespace Windows::Storage::Streams;
using namespace Windows::Graphics::Imaging;
using namespace Windows::Media::Ocr;
using namespace Windows::Storage::Pickers;
```

No more .pch!

Eliminates linker errors

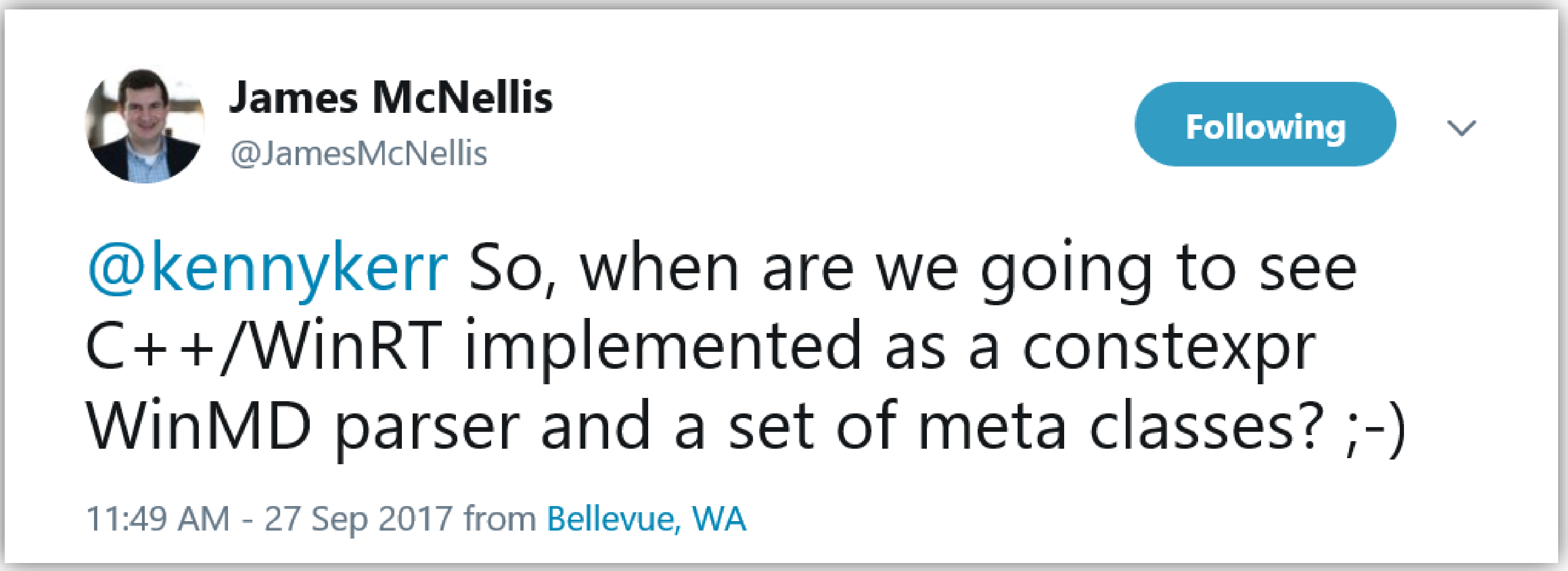
No redundancy

Macro isolation

Improved IntelliSense and much faster builds

C++ modules

Reflection, code injection, meta classes



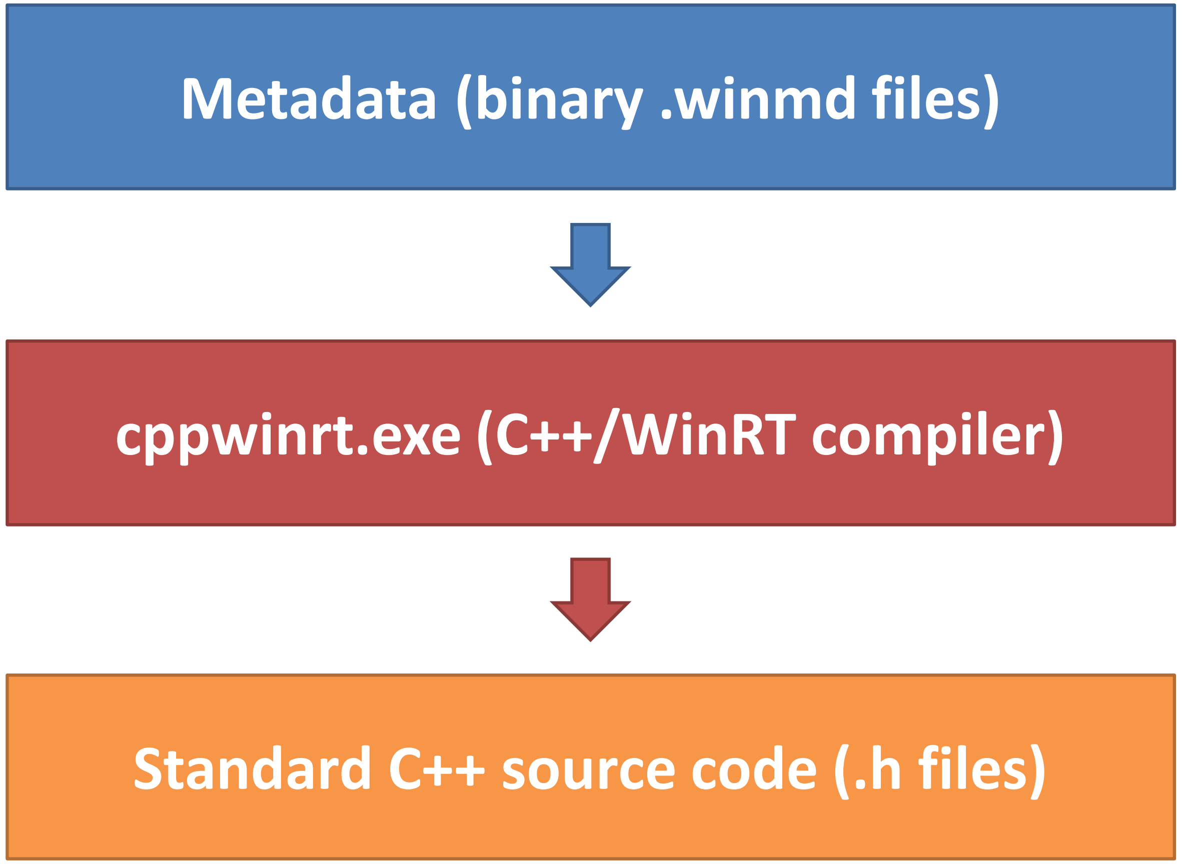
James McNellis
@JamesMcNellis

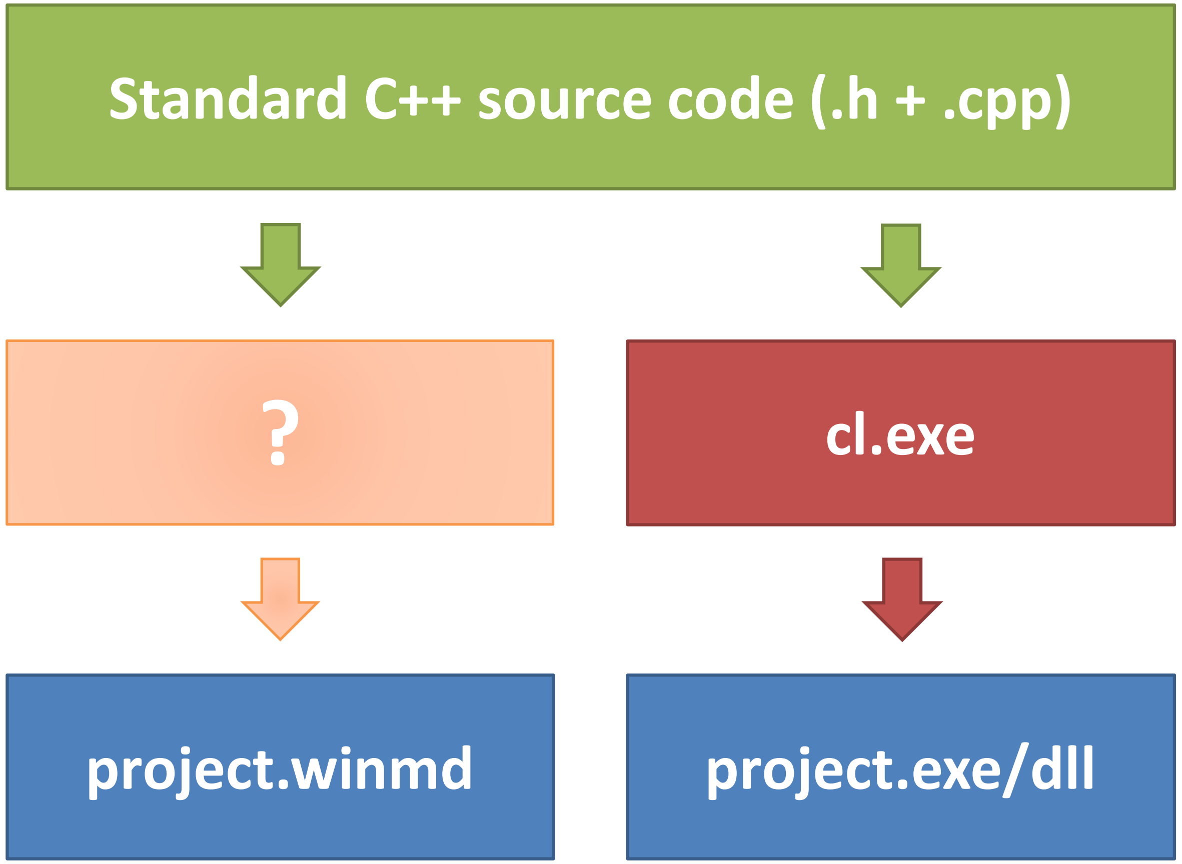
@kennykerr So, when are we going to see C++/WinRT implemented as a constexpr WinMD parser and a set of meta classes? ;-)

11:49 AM - 27 Sep 2017 from Bellevue, WA

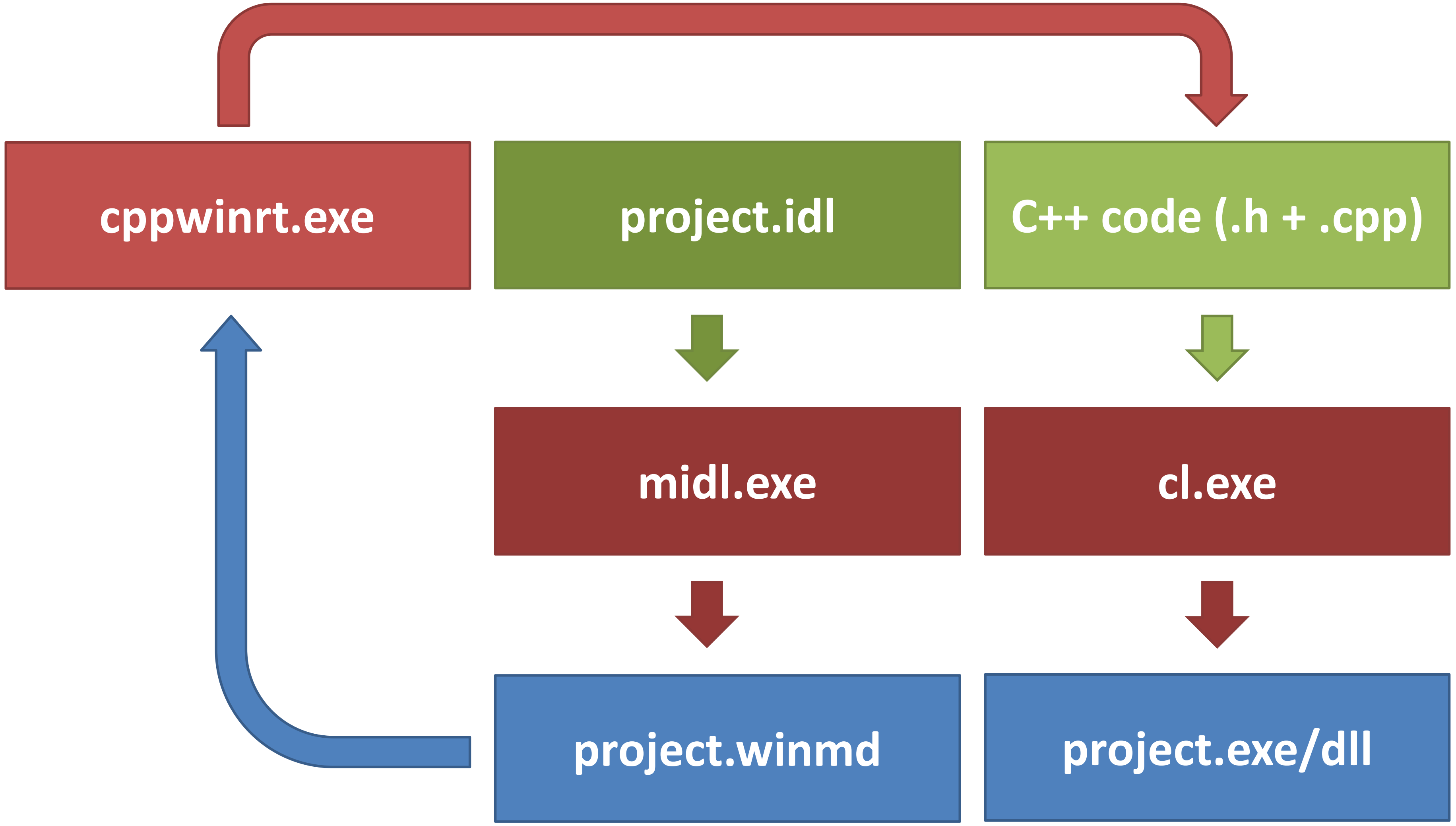
During Herb’s talk on meta classes…

Consuming metadata

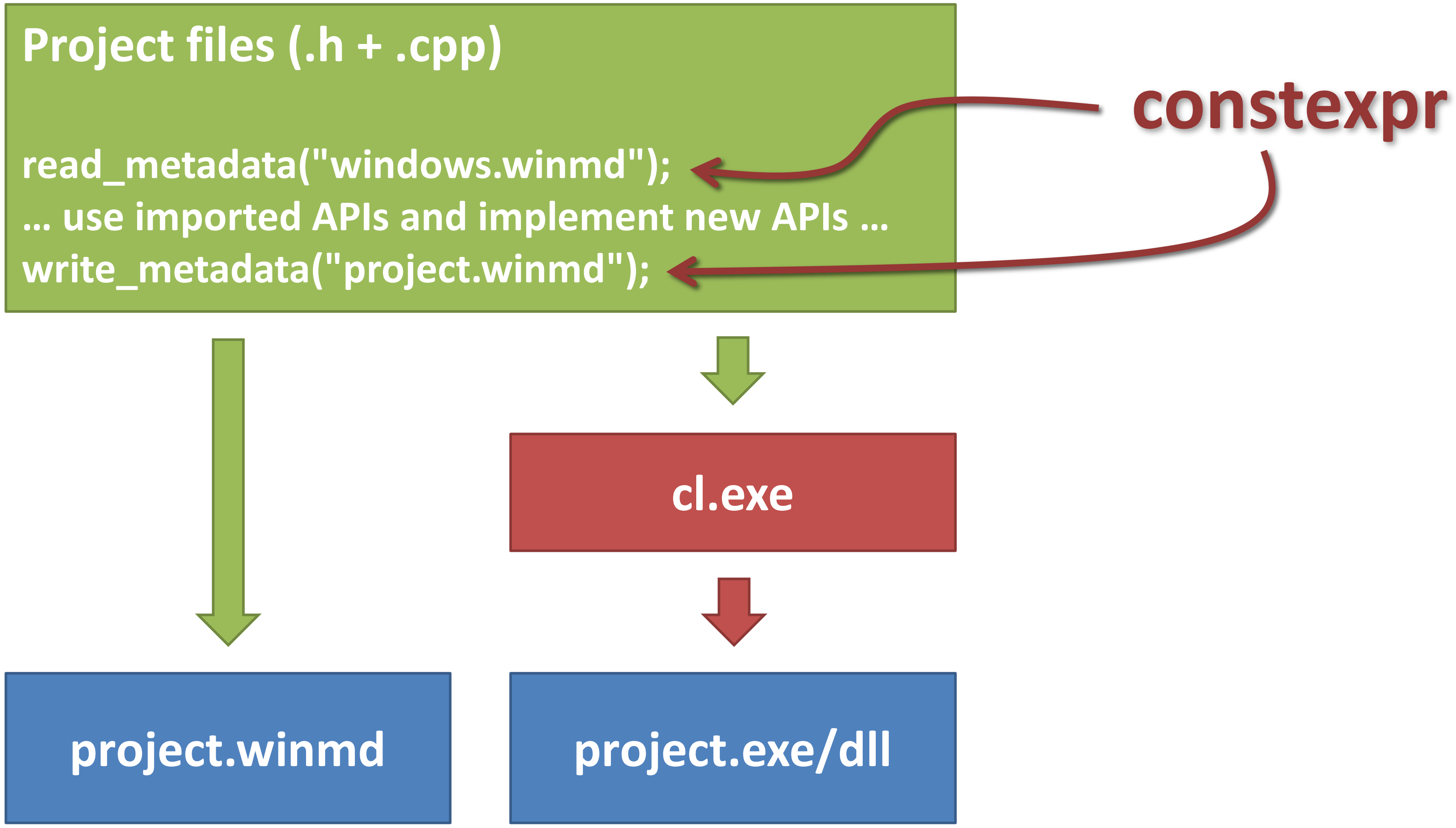

Metadata (binary .winmd files)
cppwinrt.exe (C++/WinRT compiler)
Standard C++ source code (.h files)



Producing metadata



Why so complicated?


Project files (.h + .cpp)
read_metadata("windows.winmd");
… use imported APIs and implement new APIs …
write_metadata("project.winmd");
constexpr
cl.exe
project.winmd
project.exe/dll

End-to-end in C++

```
namespace winrt
{
    $class runtime_class { ... };

    $class interface { ... };

    $class value { ... };

    template <typename T>
    $class property { ... };
}
```

WinRT metaclasses

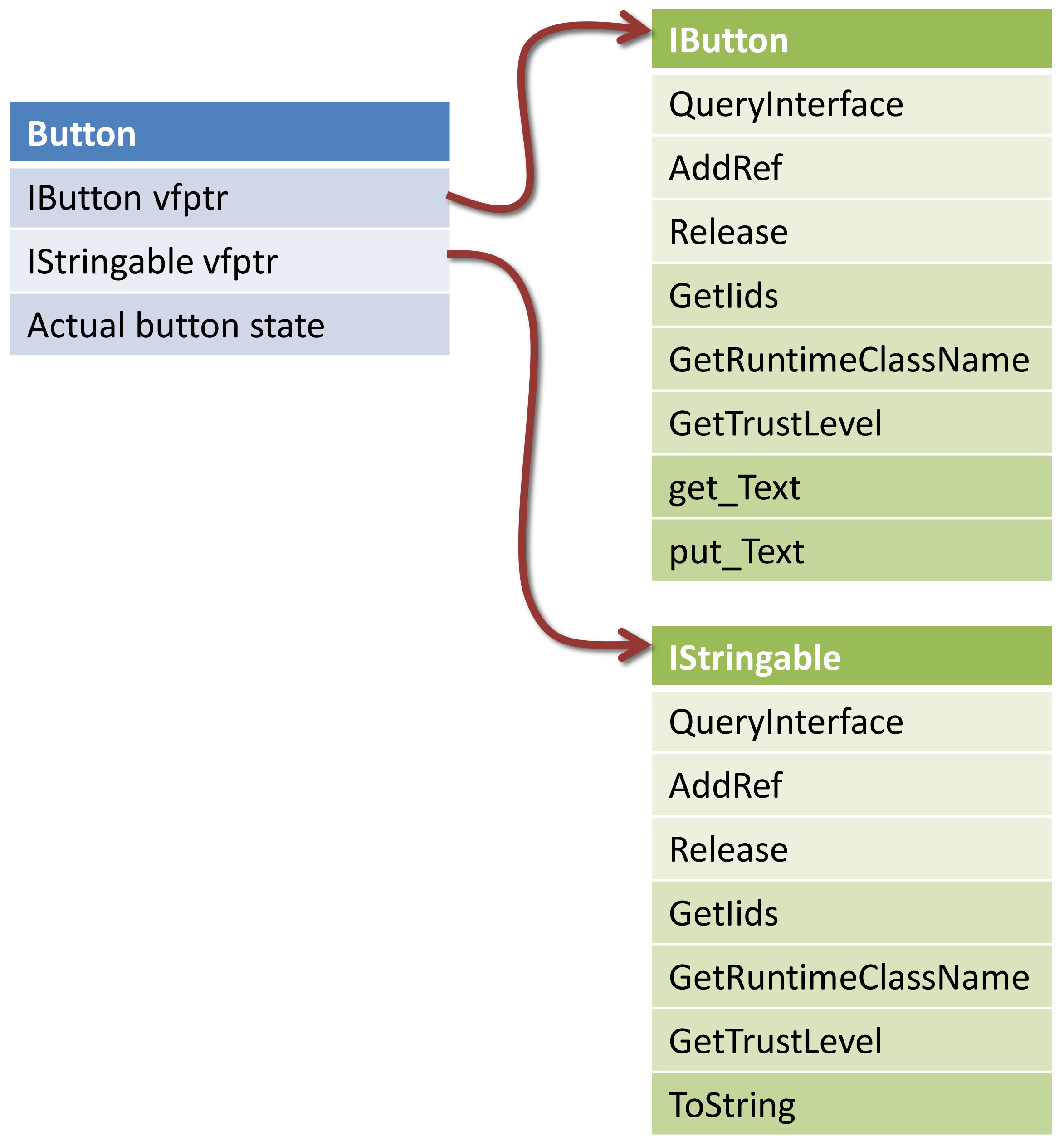
```
interface IRectangle
{
    property<int> X {};
    property<int> Y {};
    property<int> Width {};
    property<int> Height {};

    property<int const> Area {};

    void Offset(int x, int y);
};
```

Authoring WinRT types

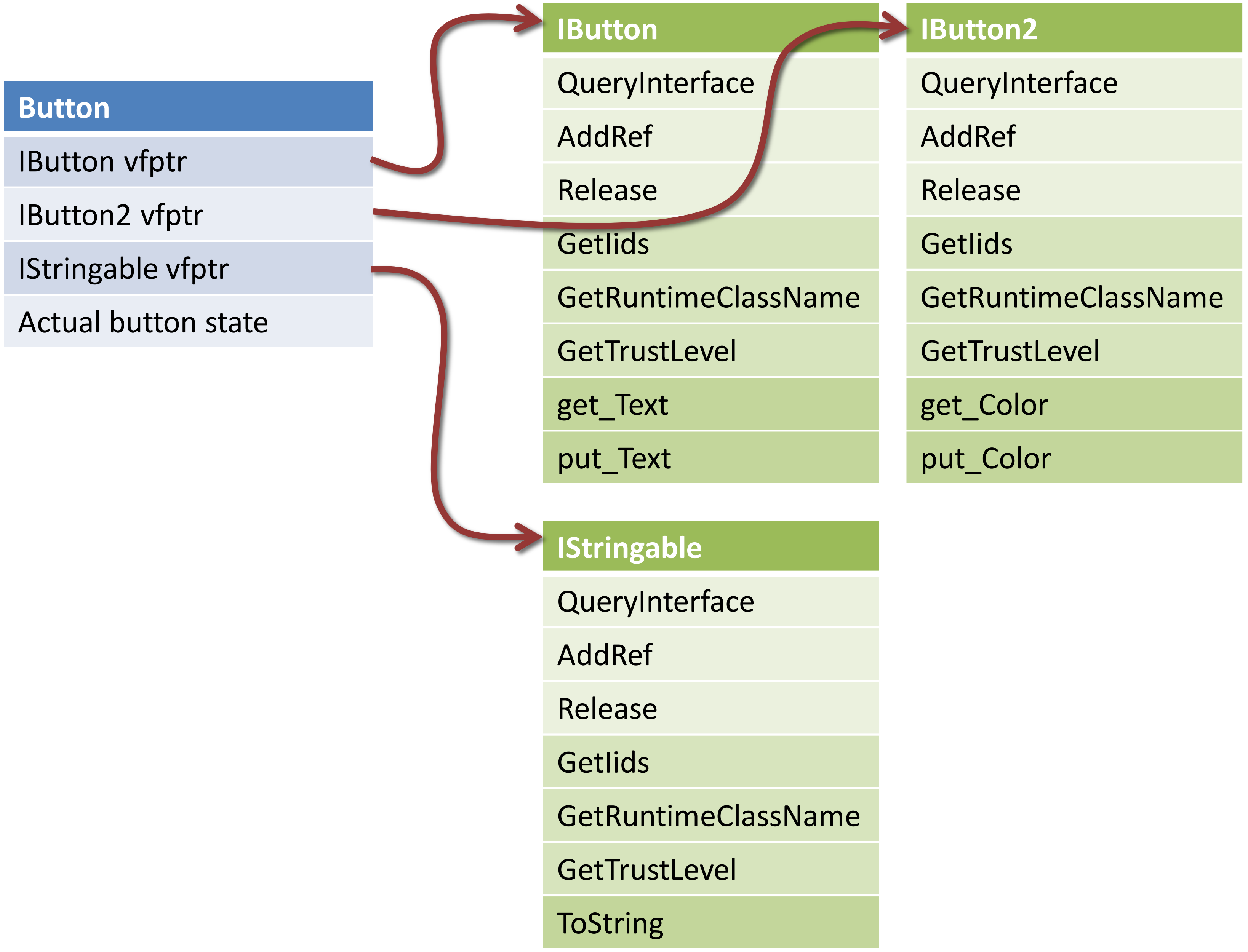
```
runtime_class Rectangle
{
    property<int> X {};
    property<int> Y {};
    property<int> Width {};
    property<int> Height {};

    property<int const> Area
    {
        int get()
        {
            return Width * Height;
        }
    };

    void Offset(int x, int y)
    {
        X += x;
        Y += y;
    }
};
```

Authoring Windows Runtime types

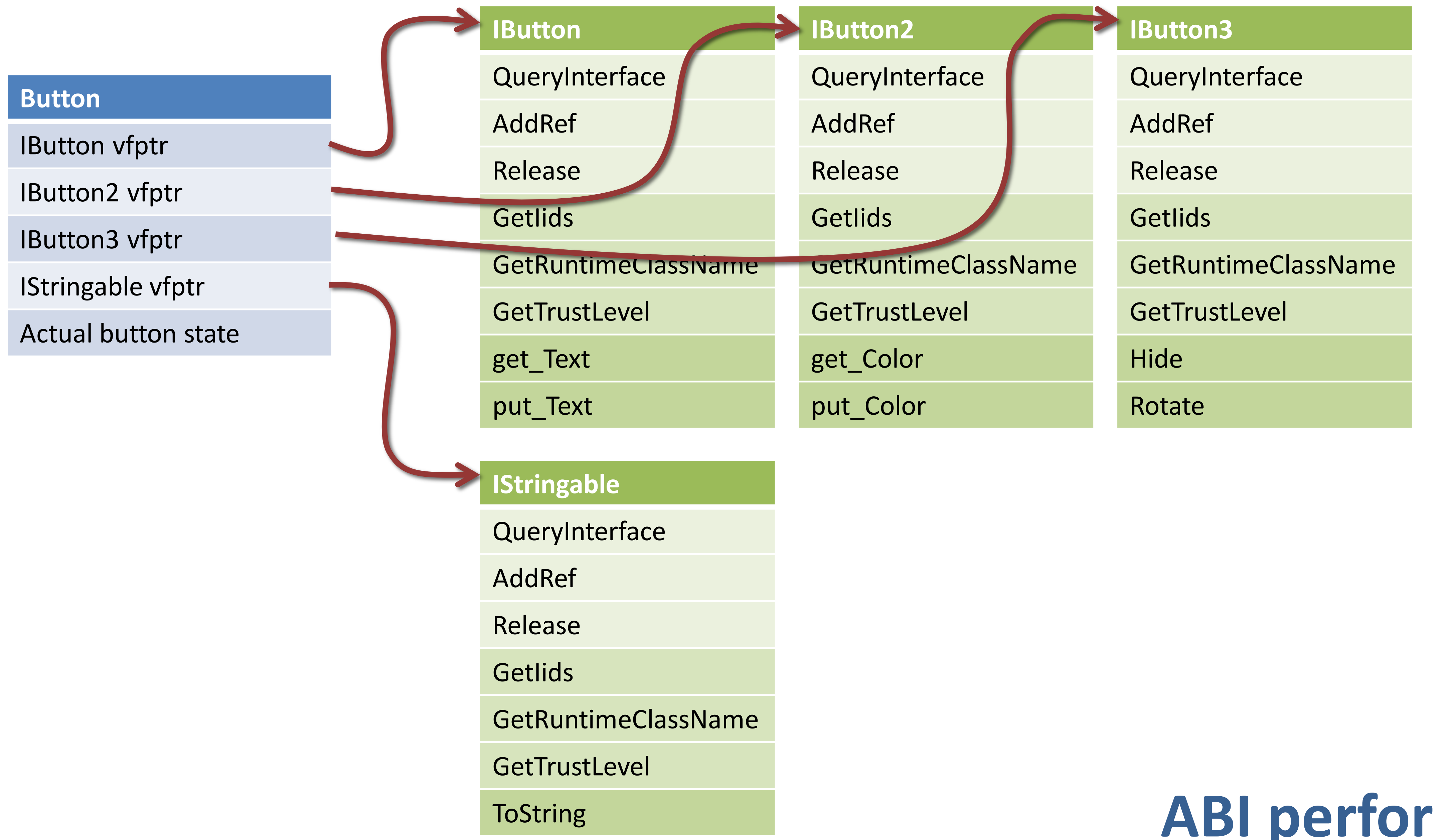
Building a better platform



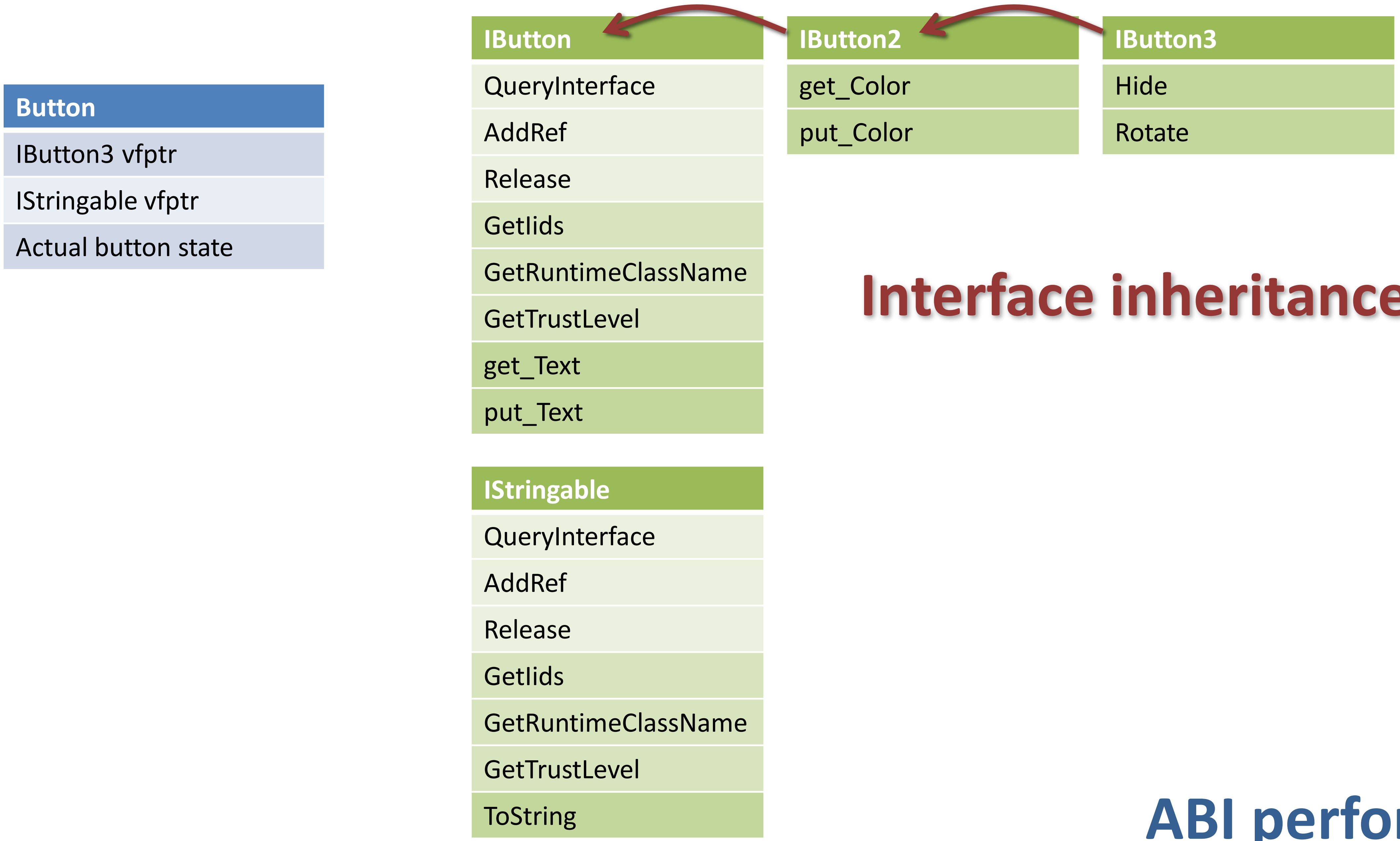
ABI performance



ABI performance



ABI performance



Button
IButton3 vfptr
IStringable vfptr
Actual button state

IButton
QueryInterface
AddRef
Release
GetIids
GetRuntimeClassName
GetTrustLevel
get_Text
put_Text

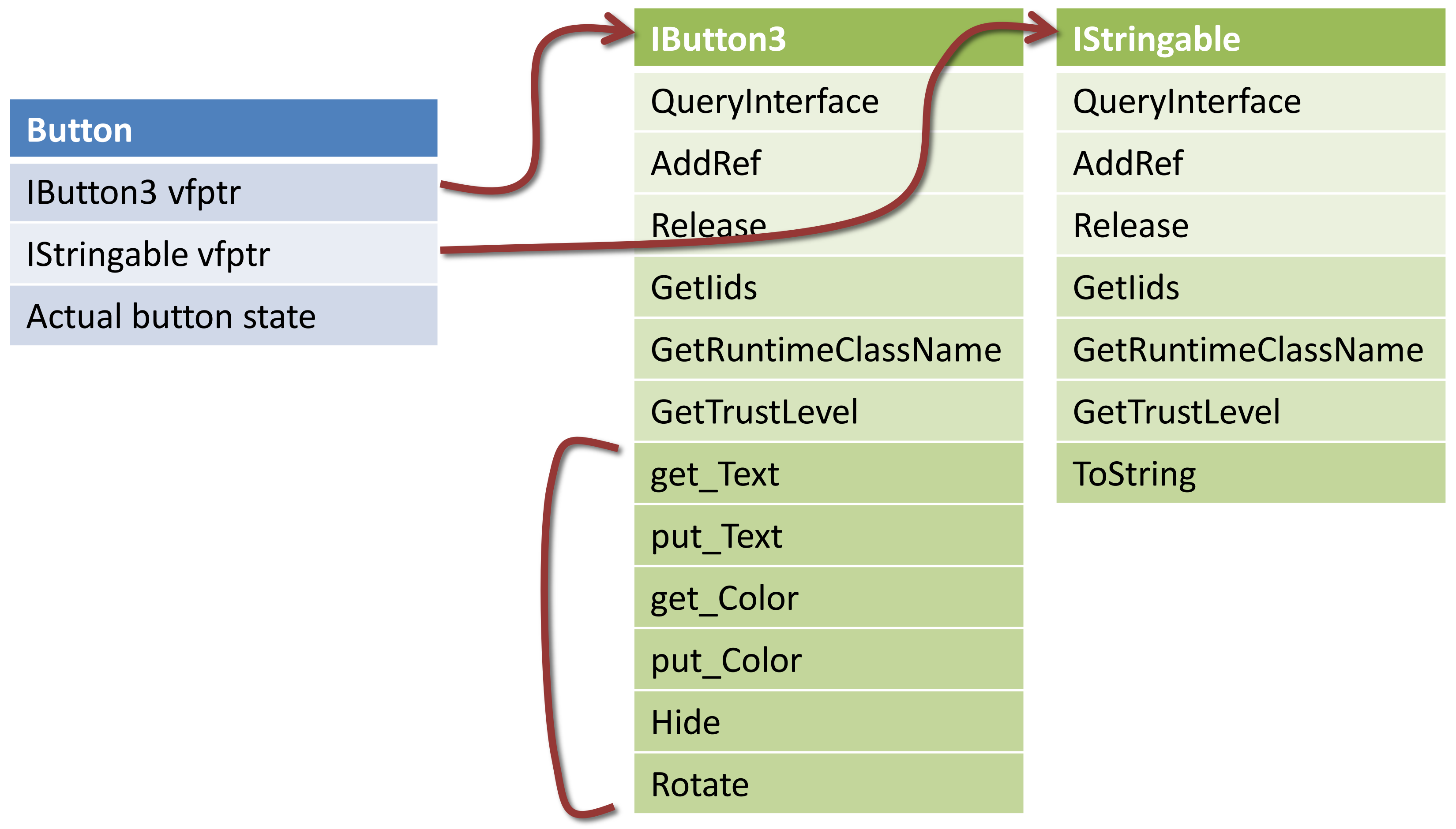
IButton2
get_Color
put_Color

IButton3
Hide
Rotate

IStringable
QueryInterface
AddRef
Release
GetIids
GetRuntimeClassName
GetTrustLevel
ToString

Interface inheritance!

ABI performance



ABI performance

```
Button button;
```

Holds IButton*

ptr->put_Text()

```
button.Text(L"Complete Survey!");
```

QueryInterface(IButton2) + ptr2->put_Color()

```
button.Color(Colors::HotPink());
```

QueryInterface(IButton3) + ptr3->Rotate()

```
button.Rotate(45.0f);
```

QueryInterface(IStringable) + ptr4->ToString()

```
hstring value = button.ToString();
```

Release() x 4

ABI performance

```
Button button;

button.Text(L"Complete Survey!");

button.Color(Colors::HotPink());

button.Rotate(45.0f);

hstring value = button.ToString();
```

Holds IButton3*

ptr->put_Text()

ptr->put_Color()

ptr->Rotate()

QueryInterface(IStringable) + ptr2->ToString()

Release() x 2

ABI performance

C++ ❤ Windows

https://**github.com/microsoft/cppwinrt** ← **Download now**

https://**moderncpp.com** ← **Learn more**

@kennykerr ← **Follow ;-)**

Many thanks to the amazing developers at Microsoft who have helped to make C++/WinRT a reality including Scott Jones, Ryan Shepherd, Brent Rector, Kevin Welton, Ben Kuhn, Herb Sutter, Cody Miller, James McNellis, Ken Sykes, Harry Pierson, Andrew Pardoe, Jonathan Caves, Gabriel Dos Reis, Larry Osterman, Jevan Saks, Stephan Lavavej, Gor Nishanov, Larry Olson, Neeraj Singh, Ulzii Luvsanbat, Victor Tong, Xiang Fan, and more...