ESC 2023

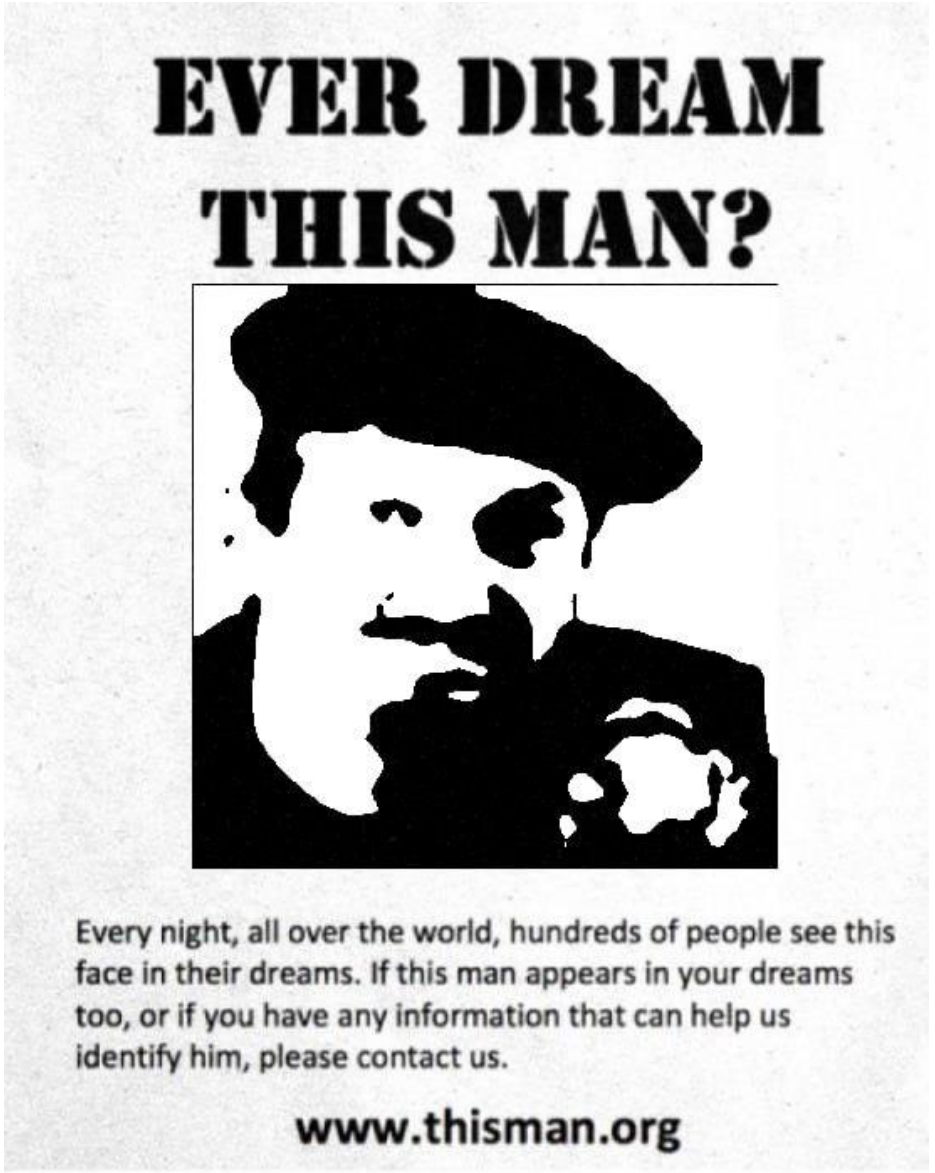
Discovering DirectX: user-land internals of the Windows Vista graphic stacks

Christian Rendina

whoami

Christian Rendina

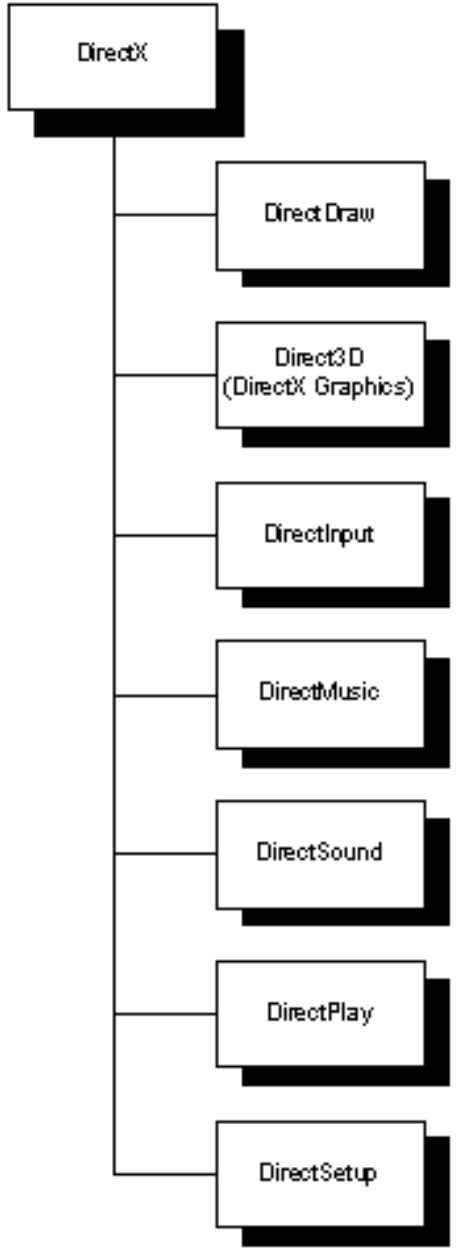
- Student
- Self-taught developer and improvised reverse engineer
- Low level, computer graphics and game dev enthusiast



(Inside joke with some friends)

What is DirectX

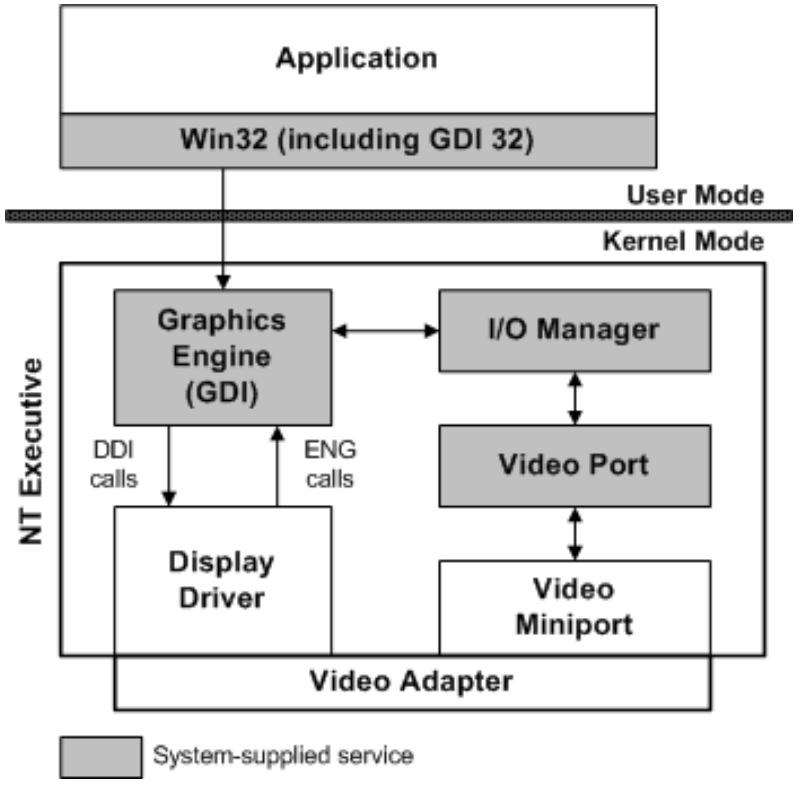
- A set of components to develop videogames
- Direct3D is the component that talks to the GPU to perform 3D functionalities
- The linux counterpart is either OpenGL or Vulkan



Source: informit.com

Windows XP Display architecture (XDDM)

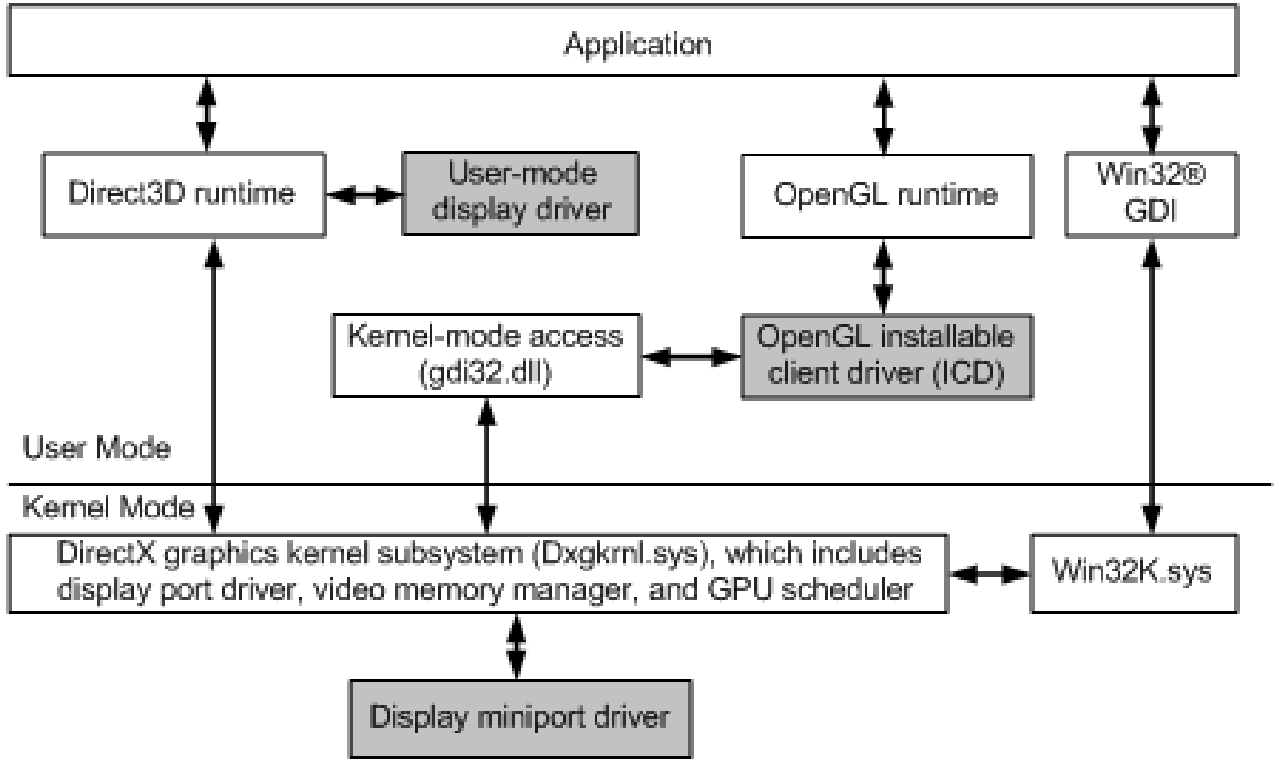
- Kernel-mode miniport driver
- Kernel-mode display driver
- Everything is connected via the Window subsystem (win32k.sys)
- Accesed in user-mode via Win32k syscalls (GDI32.dll)
- Display driver talks directly to the miniport via IOCTL



Source: MSDN

Windows Vista Display architecture (WDDM)

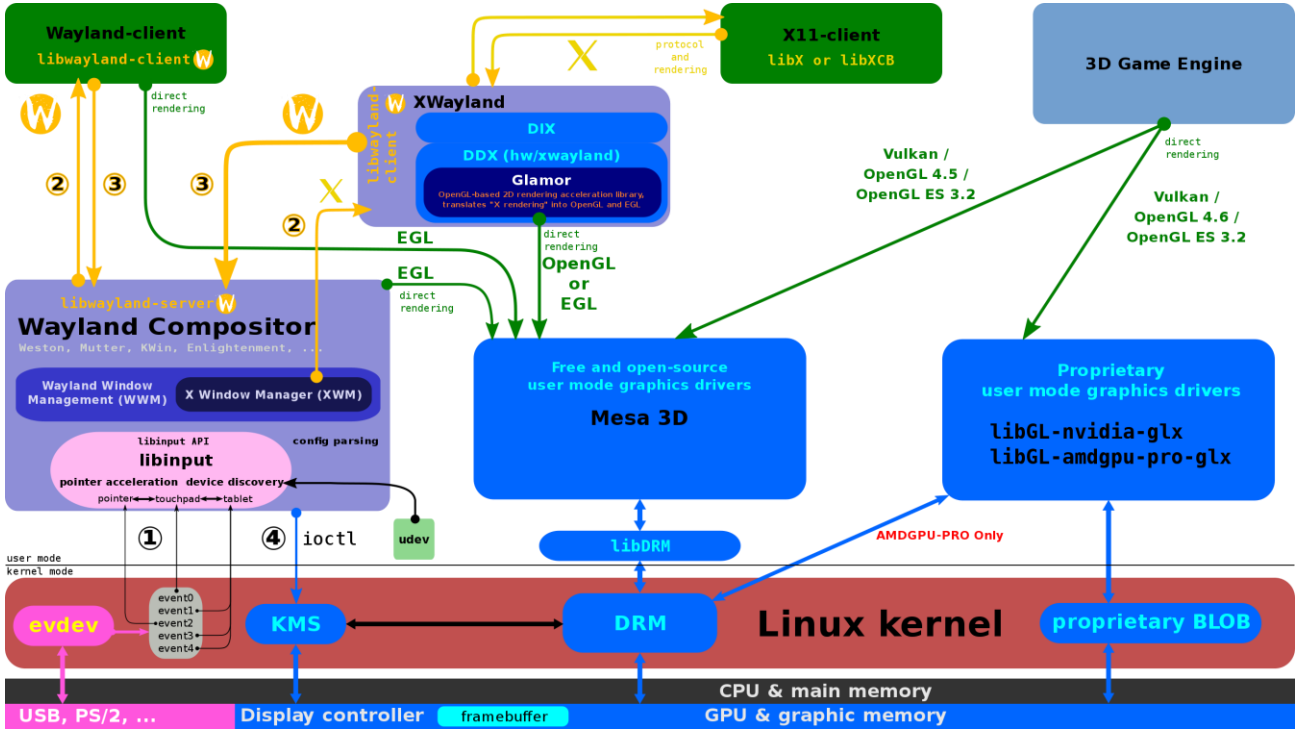
- Kernel-mode miniport driver
- User-mode display driver (UMD)
- DirectX now runs inside the kernel (dxgkrnl.sys)
- GDI is disconnected from the graphics subsystem
- No more IOCTL or specific communication to the GPU



Source: MSDN

Linux Display architecture (for comparison)

- The division between OpenGL runtime and the driver is internally done inside Mesa
- Mesa runs in user-mode like the UMD with the d3d runtime
- Different components of dxkgrnl are implemented inside DRM (KMS is part of it)
- No graphic subsystem running in the kernel



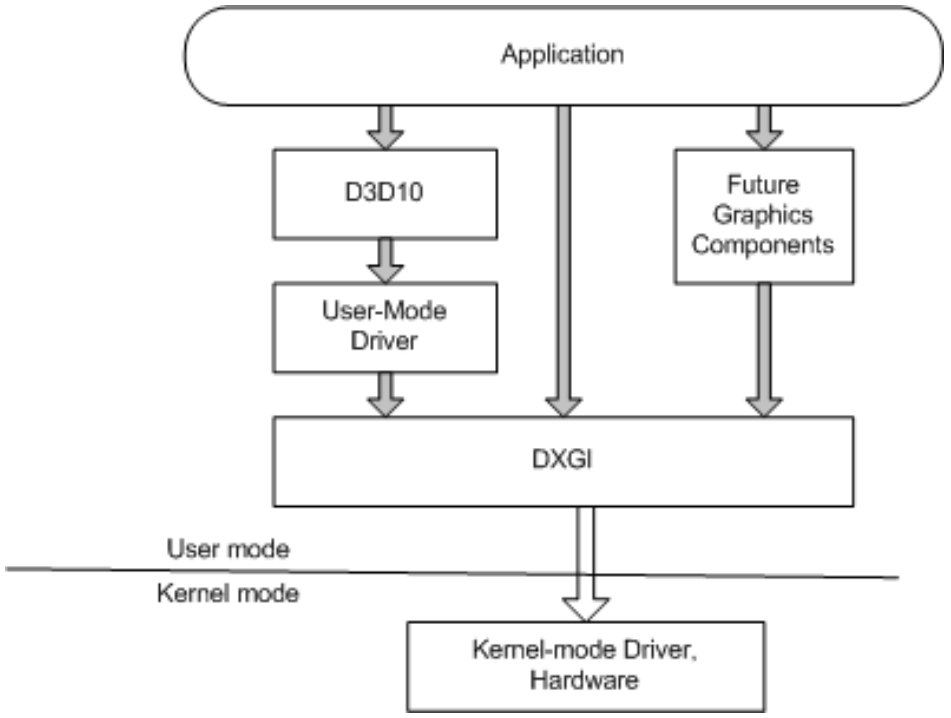
Source: Wikimedia Commons

User-land components

Two main components exposed for the userland:

- Direct3D Runtime
- DirectX Graphics Infrastructure (DXGI)

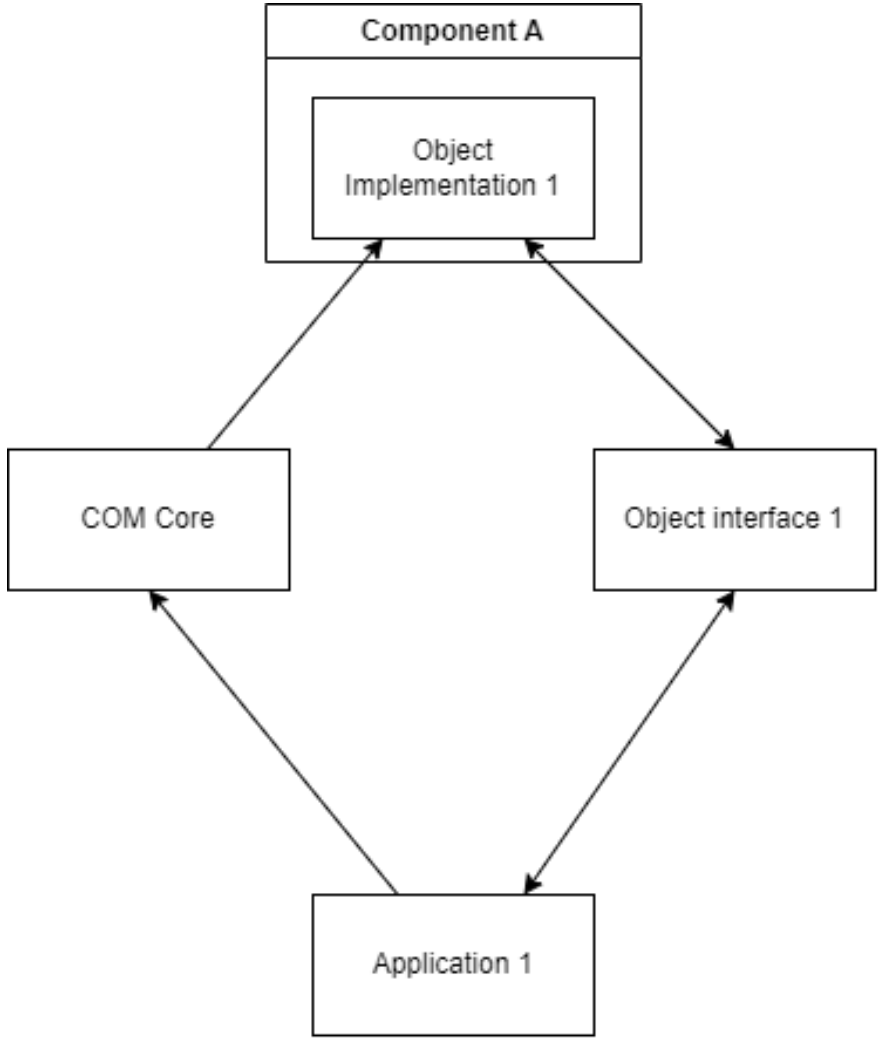
Looks quite simple, right?



Source: MSDN

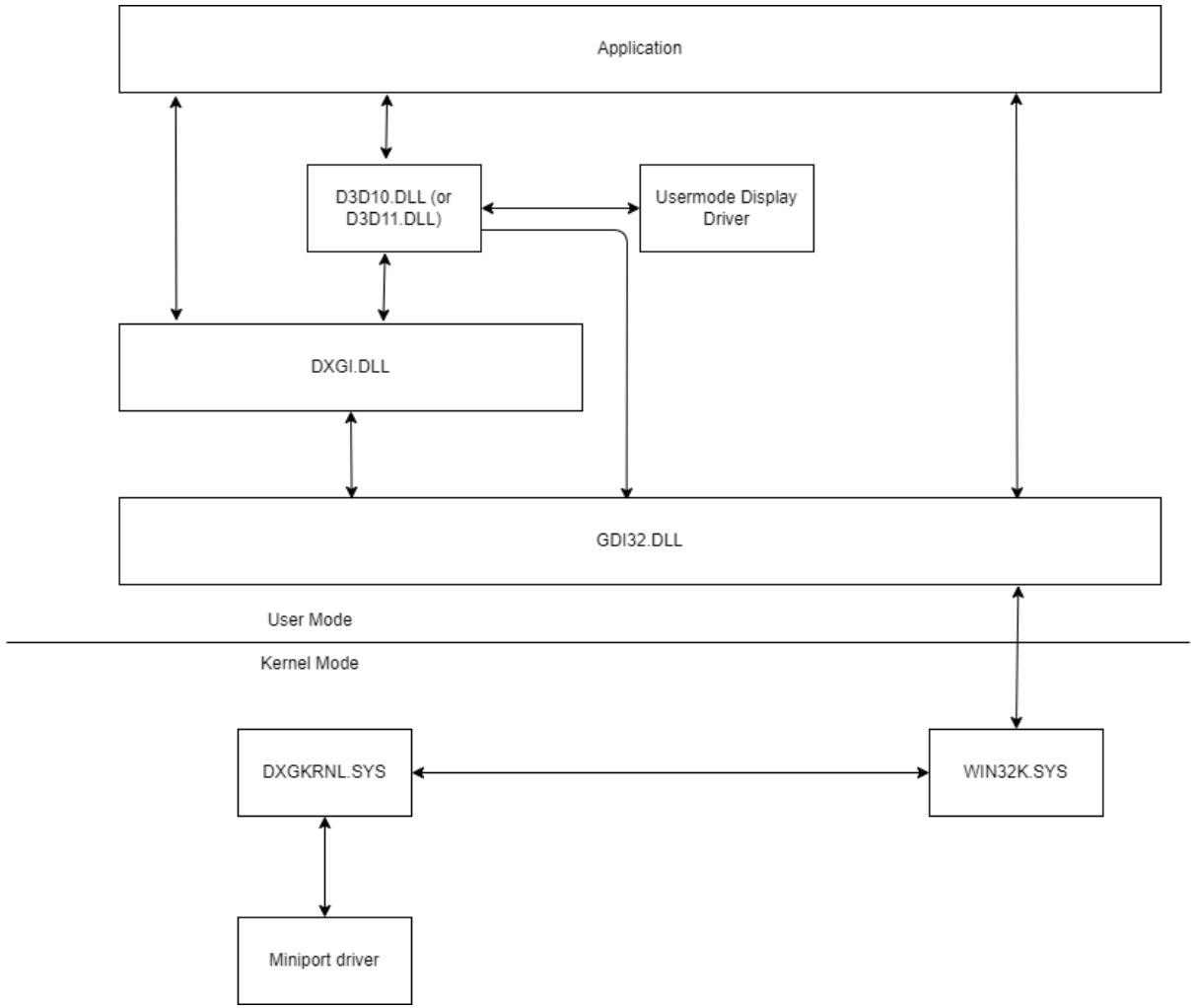
Cirno’s Perfect COM Class

- Enables inter-process communication of different components
- Each components doesn’t know their implementations but only an abstract interface
- Each component implements reference counting
- Every components has it’s own UUID which allows it to be easily identificable and buildable



How DXGI and D3D **really** works

- User-land diagram was mostly correct
- D3D10 and DXGI talks through undocumented COM interfaces
- Everything actually escapes back to GDI32 and WIN32K
- WIN32K is glued with DXGKRNL for GDI acceleration



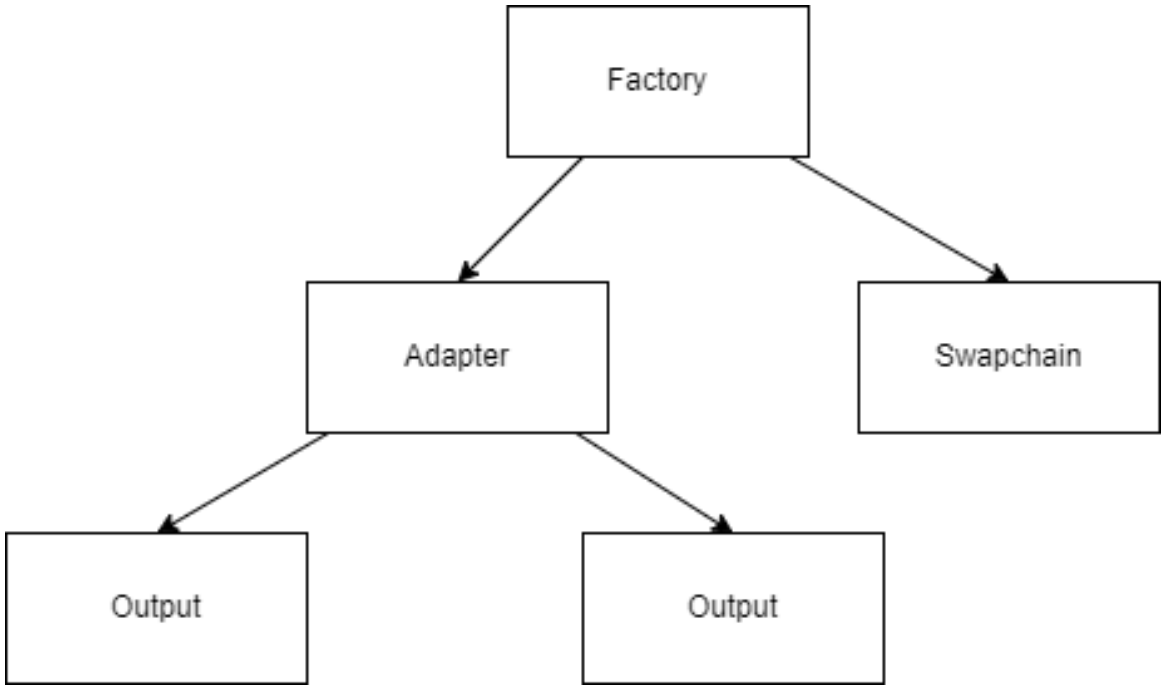
It’s all glued back to GDI

1. GDI32 exposes all the user-land functions
2. Each D3DKMT* (DXGKRNL exposed api) is implemented trough a syscall to WIN32K
3. WIN32K then forwards the function to DXGKRNL where the real operation happens

Source: Wikipedia

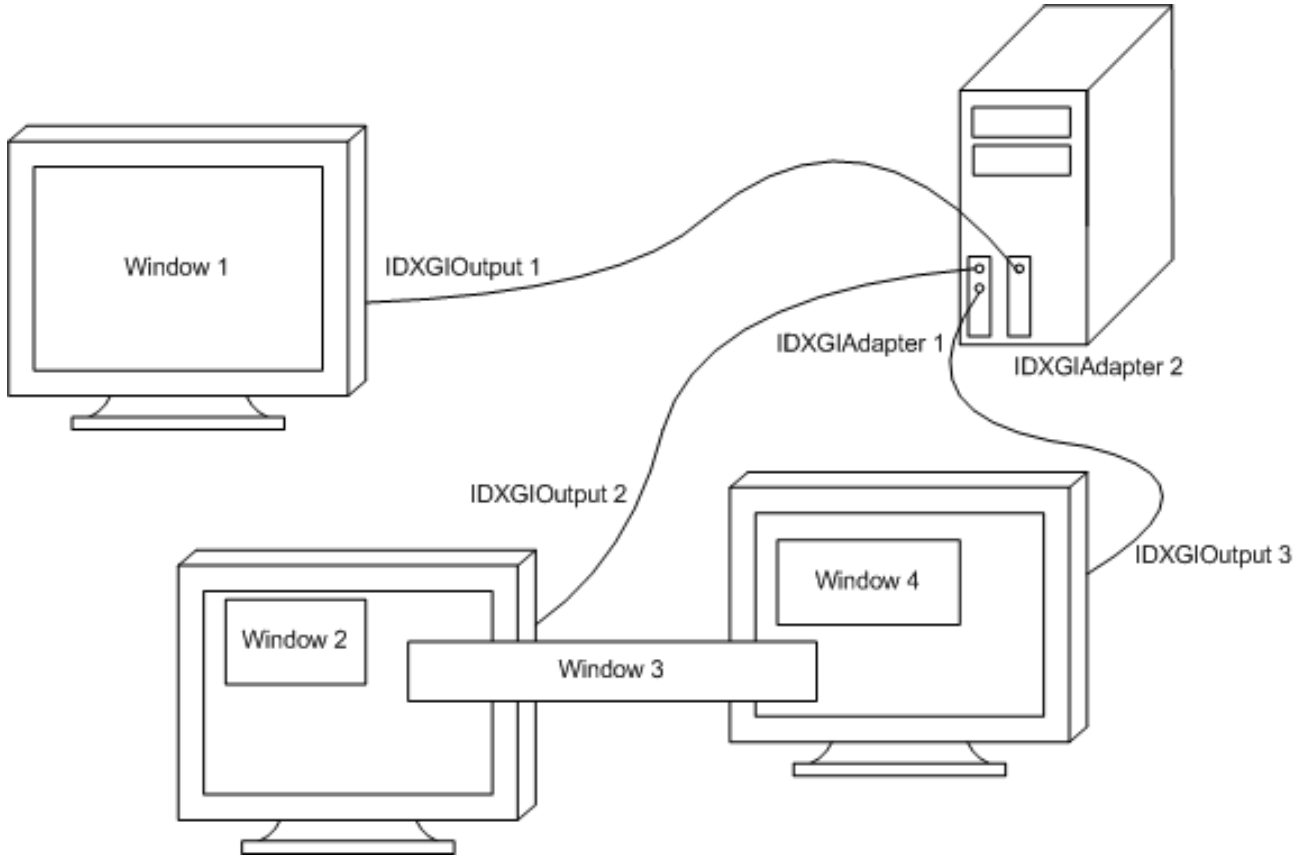
What is implemented inside DXGI?

- A factory to create all the other classes
- Rapresentations of GPU Adapters
- Rapresentations of Monitors
- A swapchain
- Base objects for everything DirectX related
- Internal structures



GPUs and Monitors

- A system can have one or more GPUs (rapresented as adapters)
- A GPU can have attached none, one or more monitors
- Remote GPUs are also supported



Source: MSDN

Example enumeration to adapters

```
for (; ids < MAX_ENUM_ADAPTERS; ids++) // attempt to enumate ALL adapters
{
        DISPLAY_DEVICEW dd = { 0 };
        dd.cb = sizeof(dd);

        if (!EnumDisplayDevicesW(nullptr, ids, &dd, 0))
                break; // found the last device

        if (!(dd.StateFlags & DISPLAY_DEVICE_ACTIVE))
                continue; // skip devices that are NOT active, tested via dxgi behavour

        D3DKMT_OPENADAPTERFROMGDIDISPLAYNAME gdi = { 0 };

        memcpy(gdi.DeviceName, dd.DeviceName, sizeof(dd.DeviceName));

        err = ApiCallback.D3DKMTOpenAdapterFromGdiDisplayName(&gdi);

        if (FAILED(err))
                break; // might not have an adapter on this device, exit

        bool skipThis = false;
```

Did Microsoft lie to us?

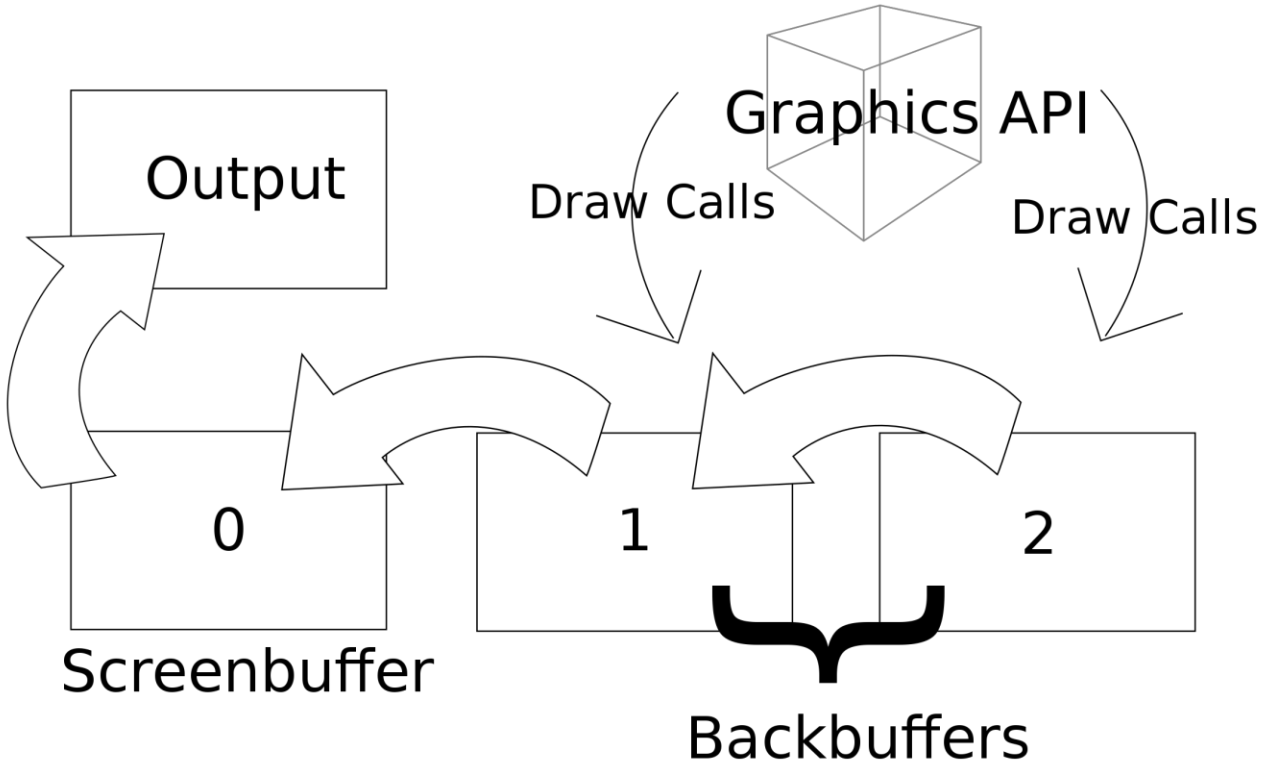
```
for (; ids < MAX_ENUM_ADAPTERS; ids++) // attempt to enumate ALL adapters
{
        DISPLAY_DEVICEW dd = { 0 };
        dd.cb = sizeof(dd);

        if (!EnumDisplayDevicesW(nullptr, ids, &dd, 0))
                break; // found the last device

        if (!(dd.StateFlags & DISPLAY_DEVICE_ACTIVE))
                continue; // skip devices that are NOT active, tested via dxgi behavour

        D3DKMT_OPENADAPTERFROMGDIDISPLAYNAME gdi = { 0 };

        memcpy(gdi.DeviceName, dd.DeviceName, sizeof(dd.DeviceName));

        err = ApiCallback.D3DKMTOpenAdapterFromGdiDisplayName(&gdi);

        if (FAILED(err))
                break; // might not have an adapter on this device, exit

        bool skipThis = false;
```

GDI32 calls

Adapter access from GDI name

What’s a swapchain

- A component that holds multiple «screens» into memory
- It’s main responsability is to swap them to the screen
- Avoids tearing and similar issue



Source: MSDN

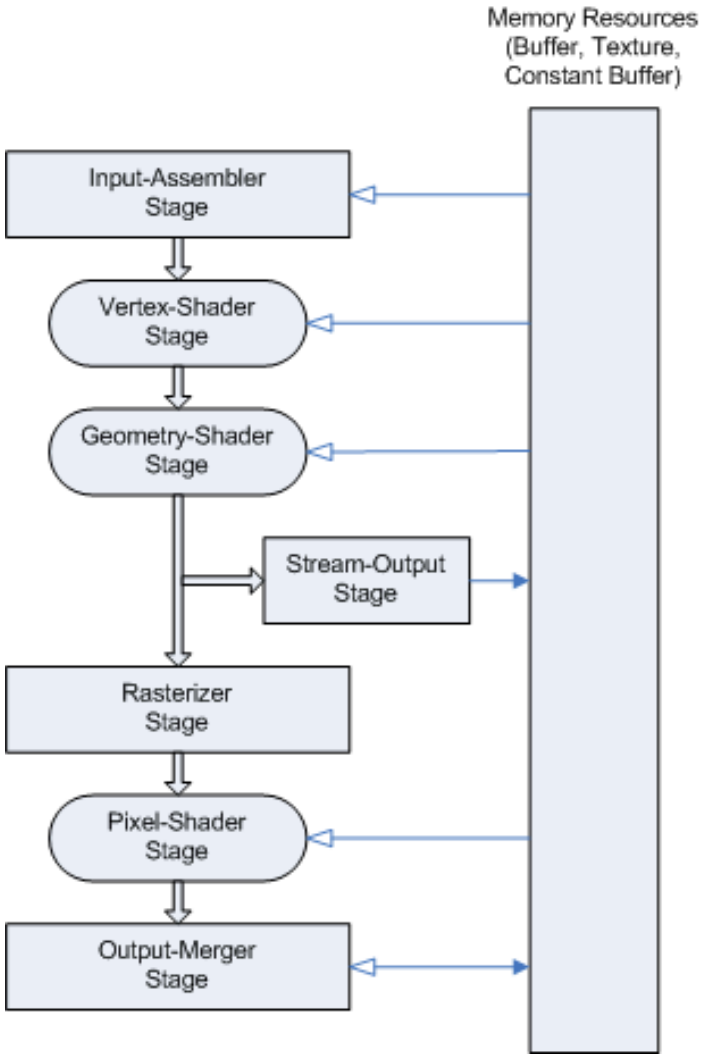
Types of swapchains

- Fullscreen Swapchain (DWM)
- DDA Swapchain
- Windowed Swapchain
- DirectComposition (Partner) Swapchain
- UWP (XAML) Swapchain
- Probably more undiscovered?

```
enum DXGI_SWAP_CHAIN_TYPE
{
  DXGI_SWAP_CHAIN_DWM = 0x0,
  DXGI_SWAP_CHAIN_DDA = 0x1,
  DXGI_SWAP_CHAIN_HWND = 0x2,
  DXGI_SWAP_CHAIN_COMPOSITION = 0x3,
  DXGI_SWAP_CHAIN_UWP = 0x4,
};
```

What's implemented in D3D10

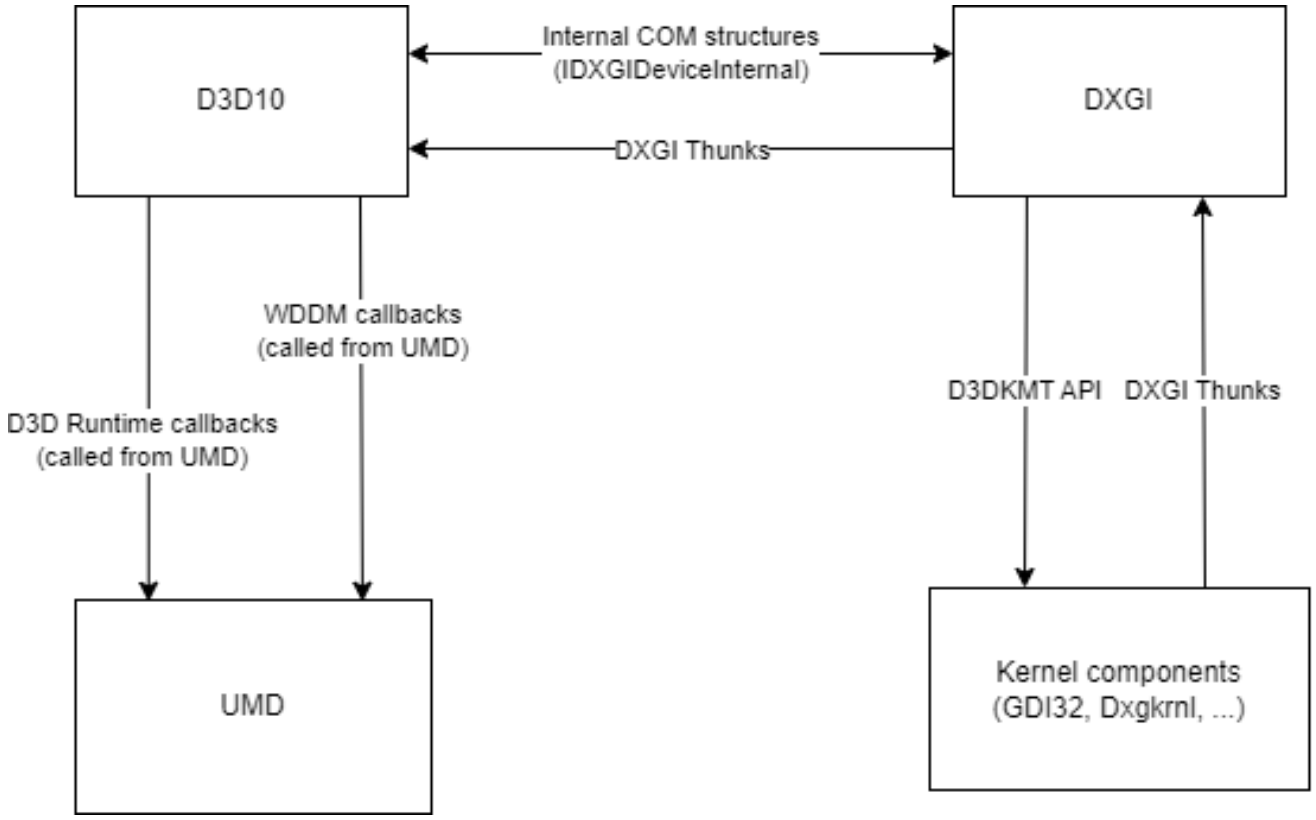
- A device to render resources to the GPU
- Connections to the UMD
- Every resources used in game (such as texture or vertex buffers)
- Presenting changes to the screen
- Shader compilation and submit to the GPU



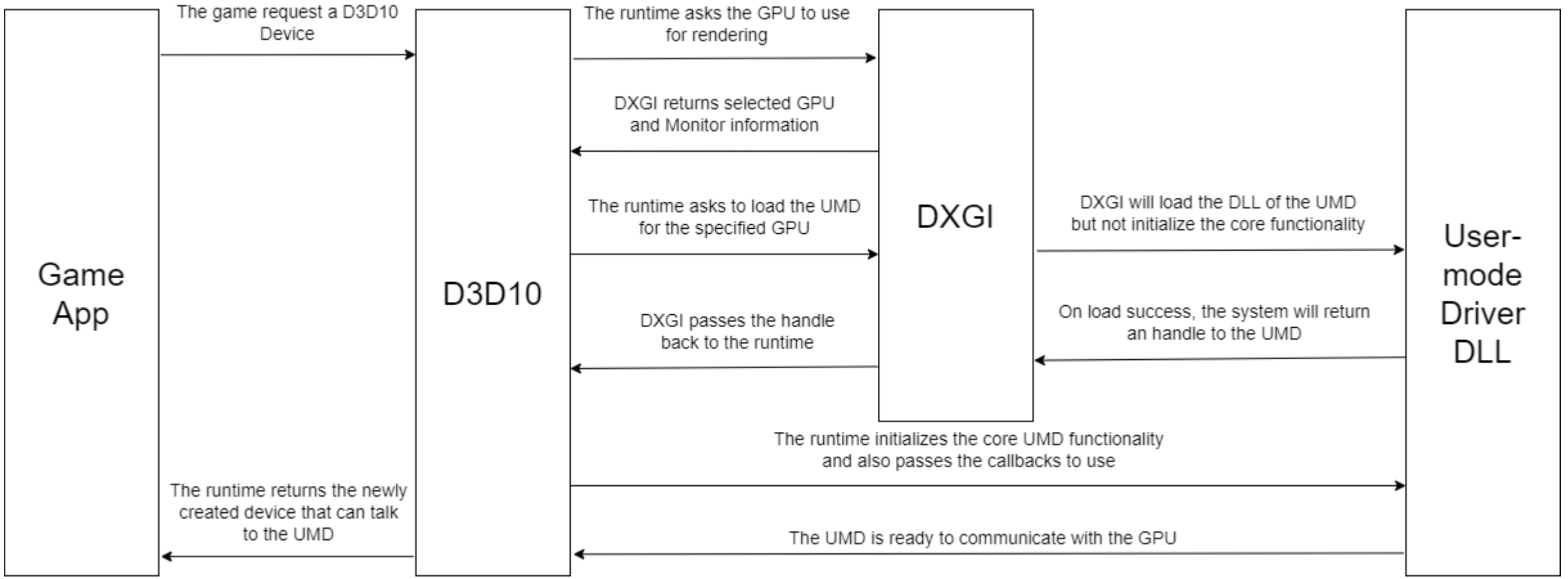
Source: MSDN

How D3D10 talks to DXGI and UMD

- DXGI Thunks are exchanged from kernel to the runtime to bypass DXGI
- Internal COM structures to talk to DXGI
- The exchange between the UMD is just a bunch of callbacks

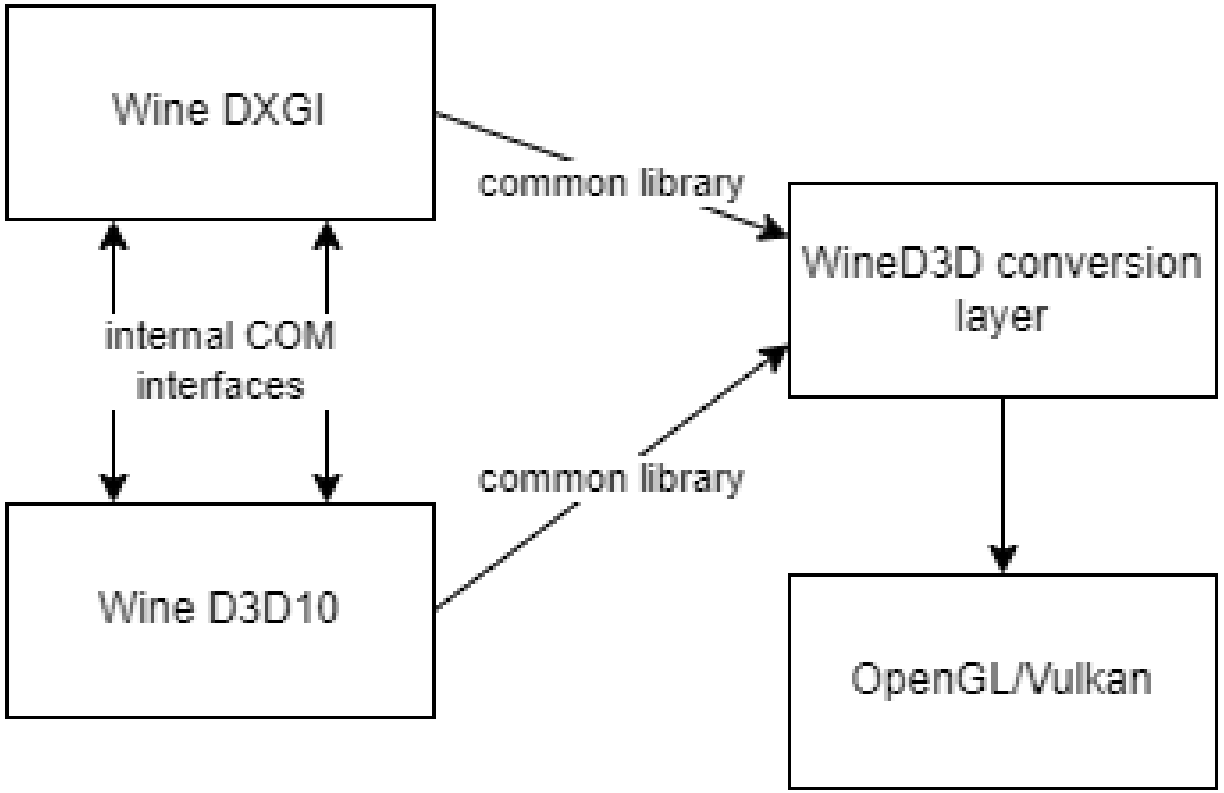


Who is responsable for the UMD?



Want some Wine?

- As expected, Wine doesn’t talk to any kernel
- Everything is translated back to OpenGL or Vulkan
- Internally D3D10 and DXGI communicates between each other via COM



Interoperability unlocked?

Live demo

Proof of concept of a custom made DXGI

Running under Windows 7 SP1 DirectX runtime (D3D11 with Windows 8 updates)

Is it really everything?

- More than 20 undocumented APIs on DXGI alone
- Undocumented APIs for DWM, XAML and DirectComposition
- Something more in the future?

```
[
    object,
    local,
    uuid(f74ee86f-7270-48e8-9d63-38af75f22d57)
]
/*
* Internal device for Windows7+
*/
interface IDXGIDeviceInternal3 : IUnknown
{
#if 1 // WINDOWS 7
    HRESULT Present(
        IDXGIDebugProducer* pDebugProducer,
        IDXGIResource*,
        void* pPresent, // D3DKMT_PRESENT
        UINT,
        UINT
    );

    HRESULT RotateResourceIdentities(
        IDXGIResource*,
        const IDXGIResource**,
        UINT
    );

    HRESULT GetContextResolver(
        void**
    );

    HRESULT CreateSurfaceInternal(
        [in] IUseCounted2*,
        [in] /*[optional]*/ IUseCounted2* l
```

```
[
    object,
    local,
    uuid(f69f223b-45d3-4aa0-98c8-c40c2b231029)
]
interface IDXGISwapChainDWM : IDXGIDeviceSubObject
{
    HRESULT Present(
        [in] UINT SyncInterval,
        [in] UINT Flags
    );

    HRESULT GetBuffer(
        [in] UINT Buffer,
        [in] REFIID riid,
        [out] void** ppSurface
    );

    HRESULT GetDesc(
        [out] DXGI_SWAP_CHAIN_DESC* pDesc
    );

    HRESULT ResizeBuffers(
        [in] UINT BufferCount,
        [in] UINT Width,
        [in] UINT Height,
        [in] DXGI_FORMAT NewFormat,
        [in] UINT SwapChainFlags
    );
```

Thank you for your attention!

- Special thanks to ESC for inviting me to do this talk
- Special thanks to the people at ReactOS Longhorn for help and support on the kernel part

https://github.com/lakor64/pff/