Windows Display Driver Model (WDDM)

Guangming Zhu

AGENDA

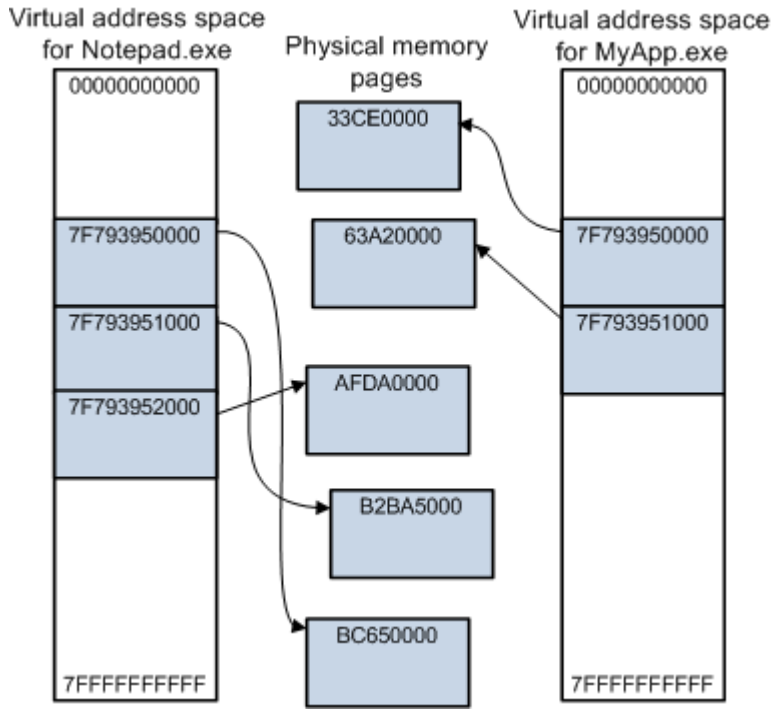
- Concepts for All Drivers Developers
- Windows Display Driver Model (WDDM)
- Knowledge of Video Memory Management
- Tasks in the WDDM

Concepts for All Drivers Developers

Virtual Address Spaces

Accessing memory through a virtual address has these advantages:

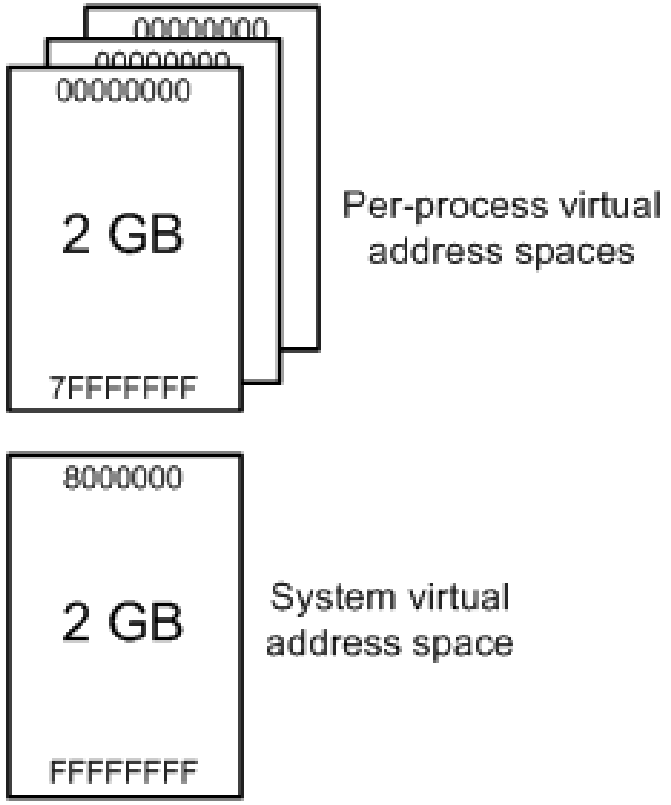
- Contiguous range of virtual addresses
- Larger memory buffer
- Isolated virtual addresses



some of the key features of virtual address spaces

User Space and System Space

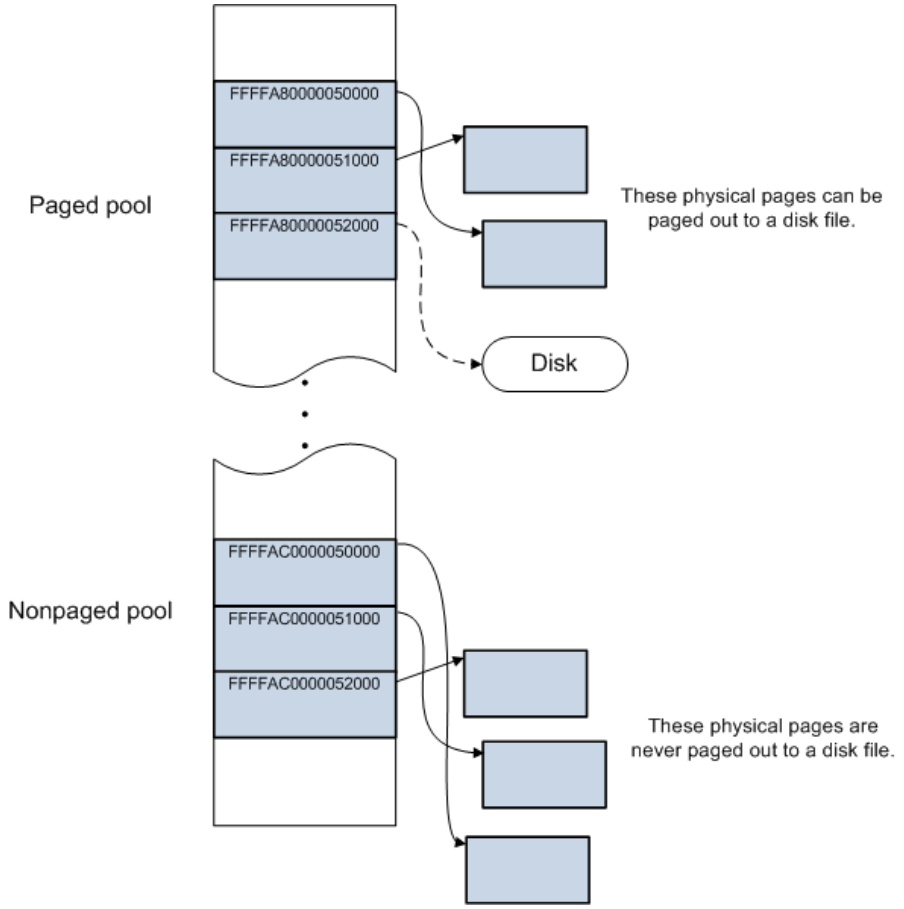
Each user-mode process has its own private virtual address space, but all code that runs in kernel mode shares a single virtual address space called system space. The virtual address space for a user-mode process is called user space.



User space and system space

Paged Pool and Unpaged Pool

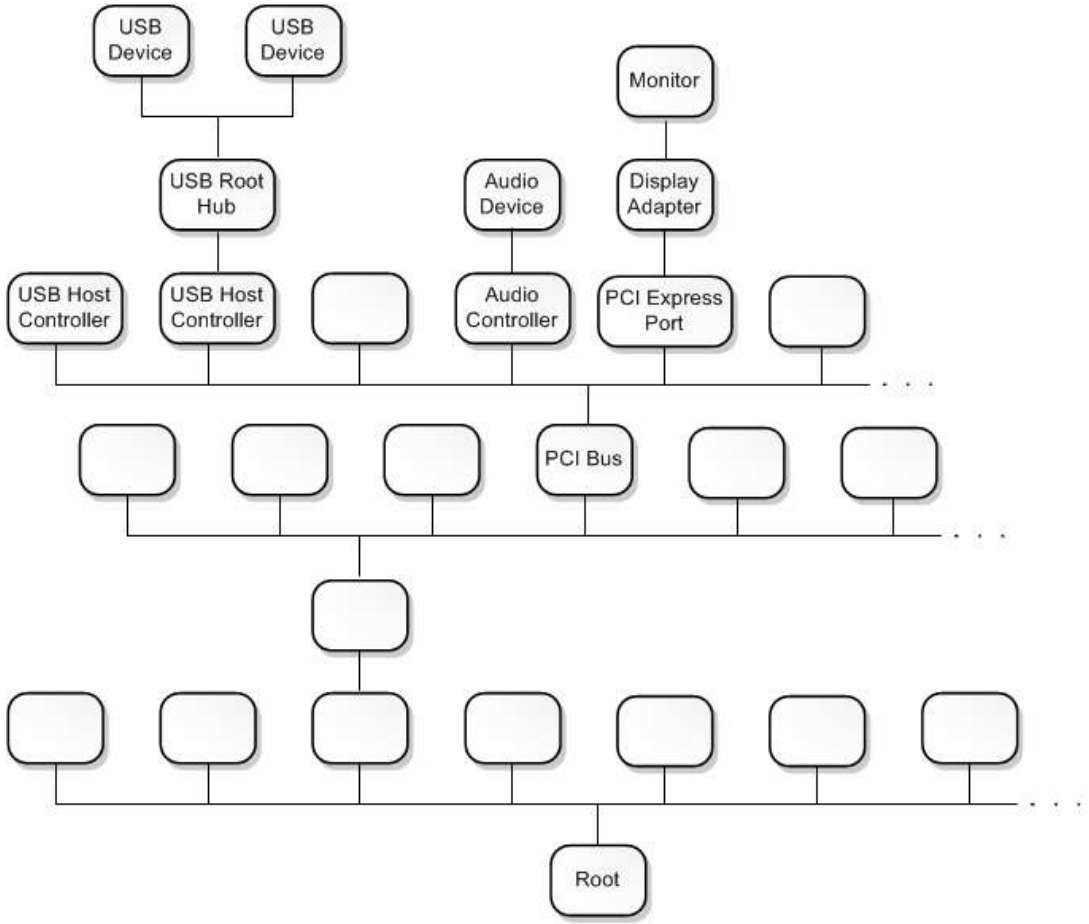
- Paged pool
- Unpaged pool



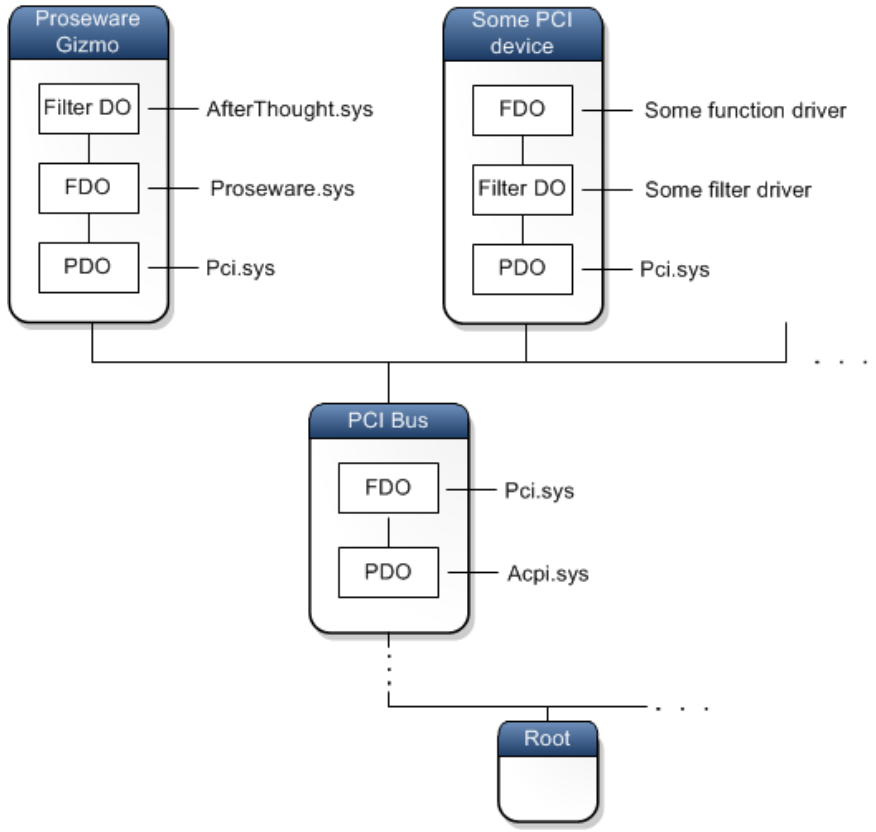
Paged pool and unpaged pool

Device Nodes and Device Stacks

In Windows, devices are represented by device nodes in the Plug and Play (PnP) device tree.



device nodes and device stacks



device tree

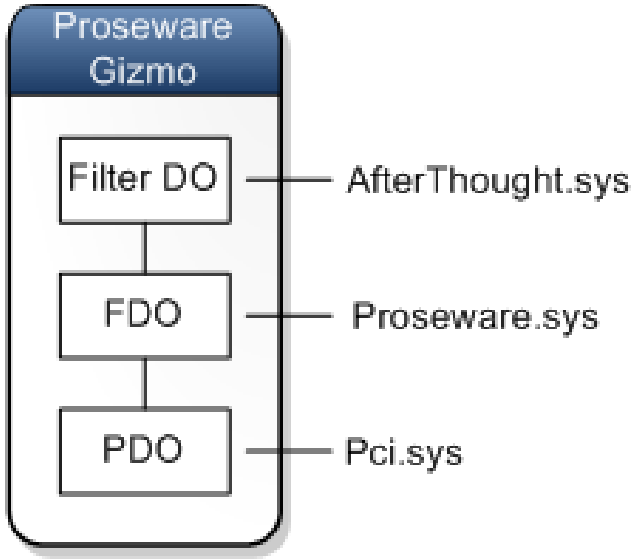
I/O Request Packets(IRP)

Most of the requests that are sent to device drivers are packaged in I/O request packets (IRPs). An operating system component or a driver sends an IRP to a driver by calling IoCallDriver.

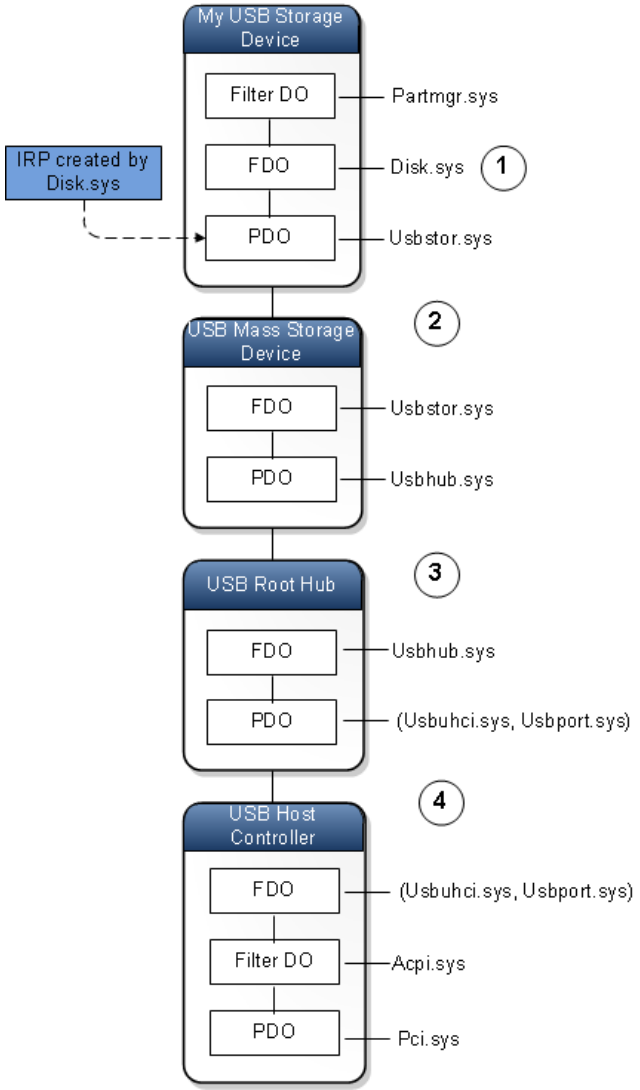
PDO: Physical Device Object

FDO: Functional Device Object

Filter DO: Filter Device Object



device node



four device stacks are involved in processing a single IRP

The Windows Driver kit(WDK)

The Windows Driver Kit (WDK) contains all the header files (.h files) that you need to build kernel-mode and user-mode drivers.

Constant	操作系统版本
NTDDI_WIN10	Windows 10
NTDDI_WINBLUE	Windows 8.1
NTDDI_WIN8	Windows 8
NTDDI_WIN7	Windows 7
NTDDI_WS08SP4	Windows Server 2008 SP4
NTDDI_WS08SP3	Windows Server 2008 SP3
NTDDI_WS08SP2	Windows Server 2008 SP2
NTDDI_WS08	Windows 2008 Server

predefined constant values that represent versions of the Microsoft Windows operating system

```
#if (NTDDI_VERSION >= NTDDI_WIN7)
_Must_inspect_result_
NTKERNELAPI
NTSTATUS
KeSetTargetProcessorDpcEx (
    _Inout_ PKDPC Dpc,
    _In_ PPROCESSOR_NUMBER ProcNumber
    );
#endif
```

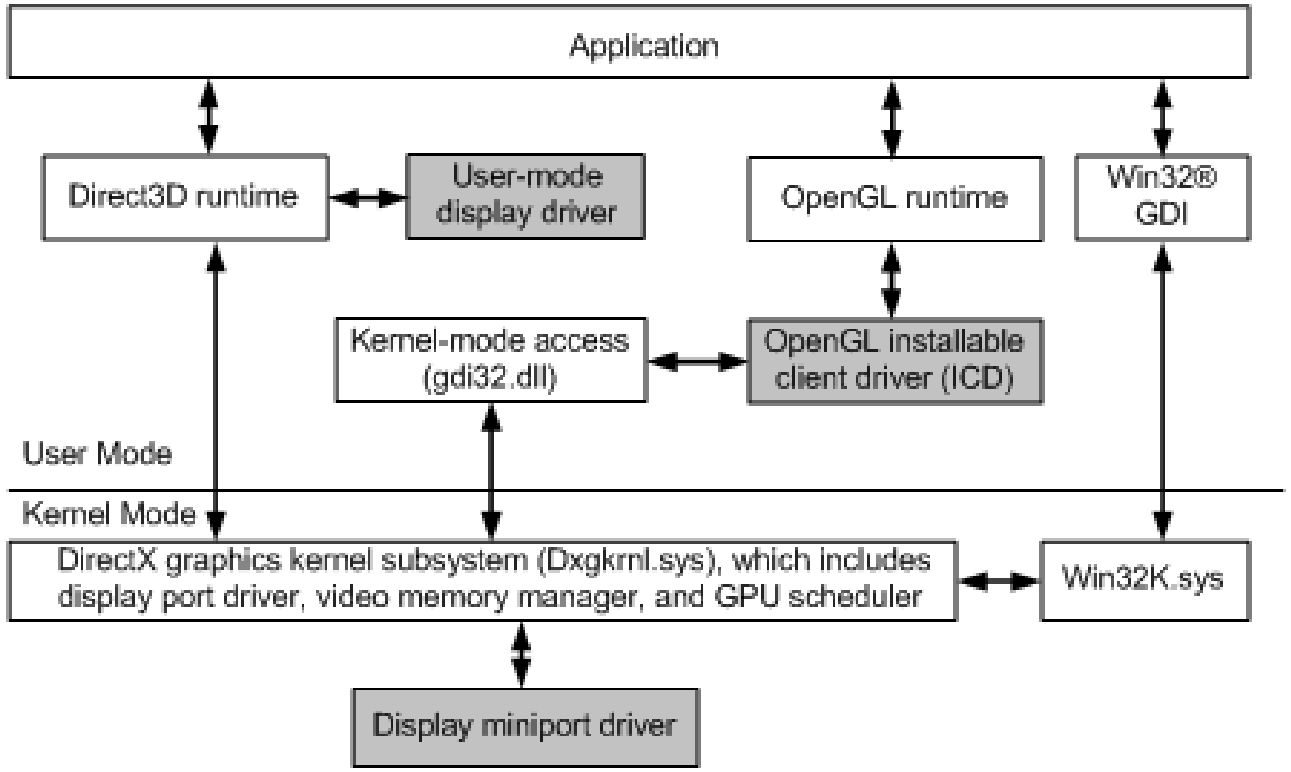
compare the value of NTDDI_VERSION with a set of predefined constant values

Windows Display Driver Model(WDDM)

WDDM Architecture

The display driver model architecture for the Windows Display Driver Model (WDDM).

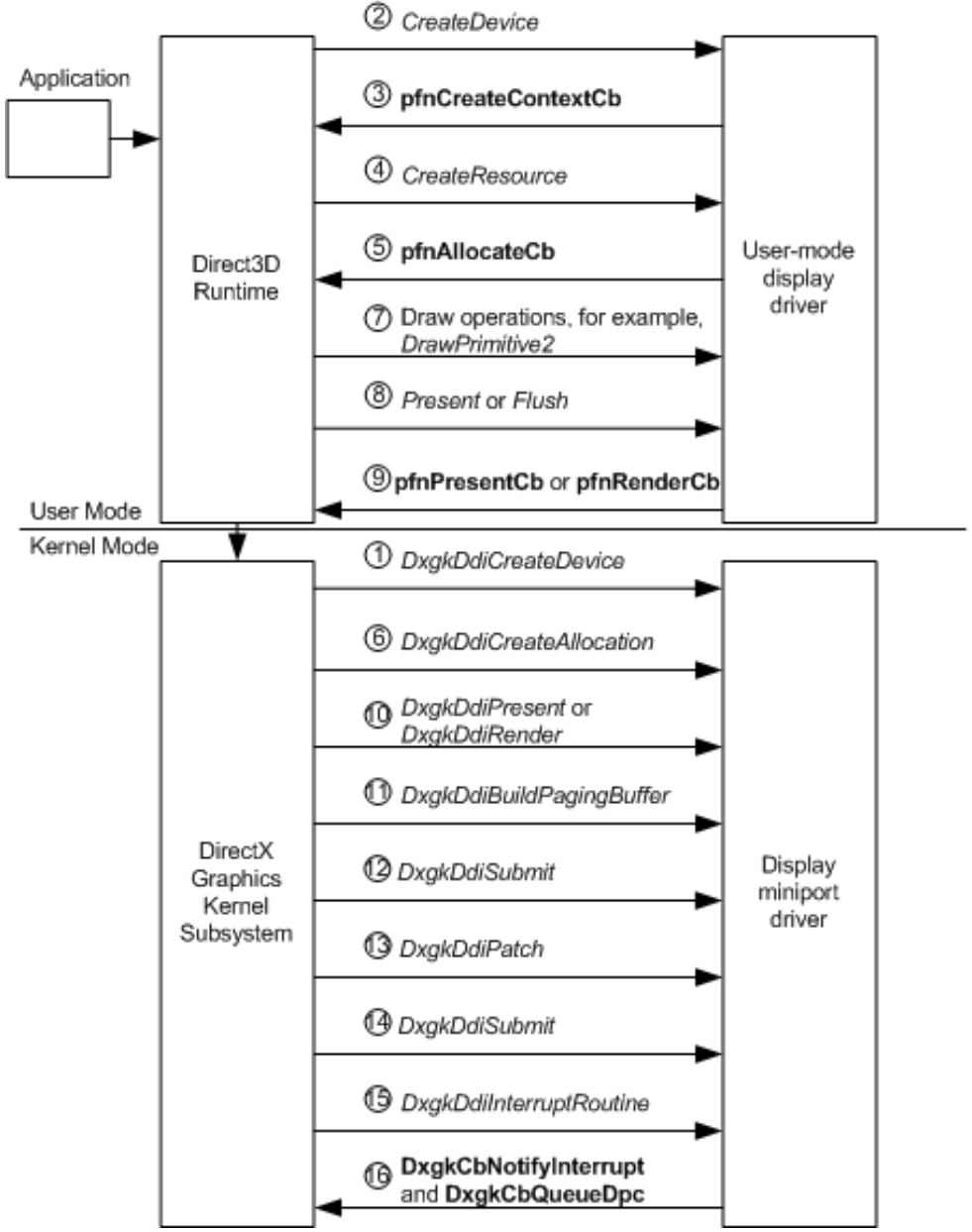
- user-mode
- kernel-mode



WDDM architecture

WDDM Architecture

- Creating a Rendering Device
- Creating Surfaces for a Device
- Submitting the Command Buffer to Kernel Mode
- Submitting the DMA Buffer to Hardware



the flow of Windows Display Driver Model (WDDM) operations

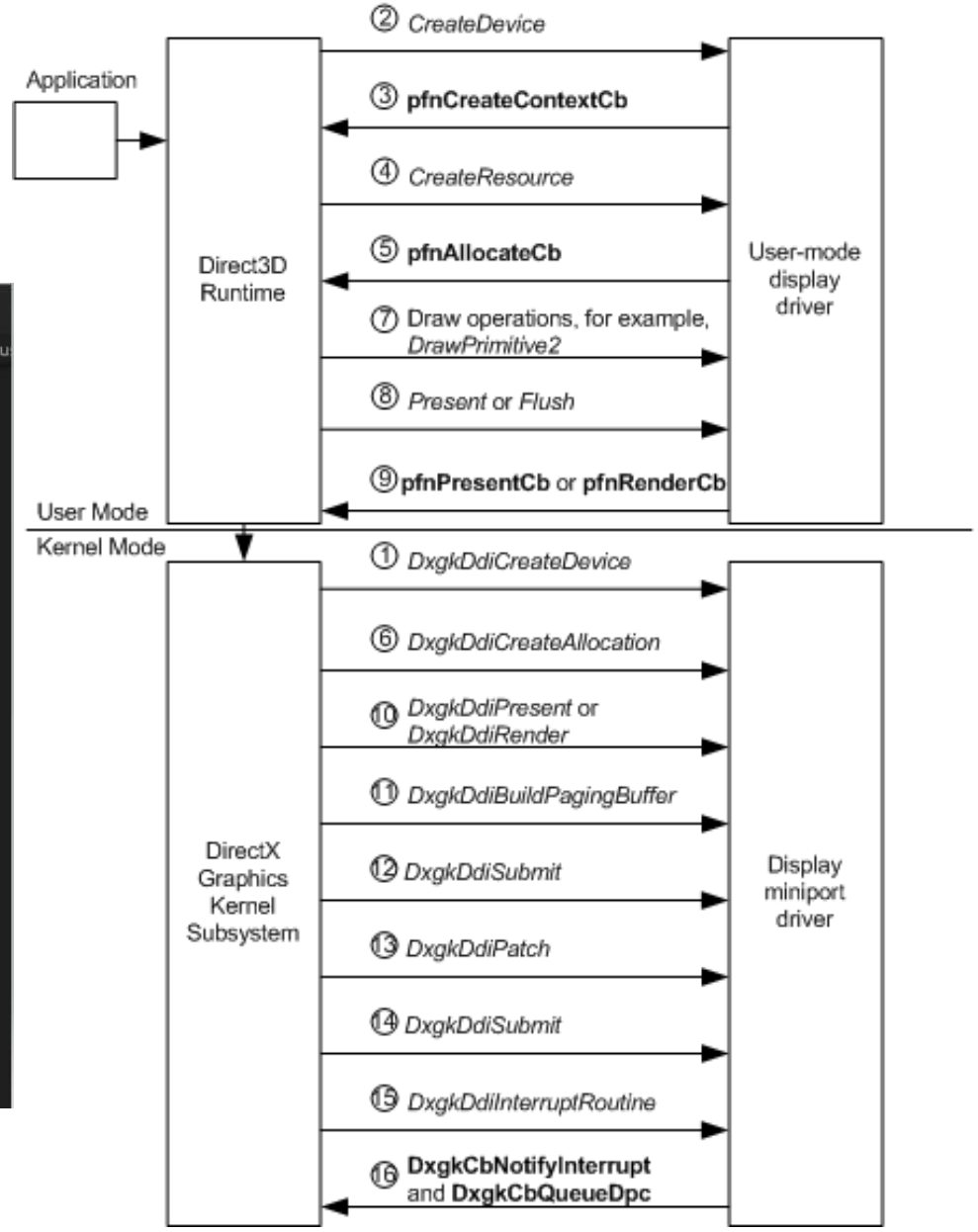
Associated with Code

KMDENTRY#261.CPP

C: > Users > guangzhu > AppData > Local > Temp > P4V > SHA-L-GUANGZHU_SHSWP4P1_1666 > DEPOT > MAIN > DRIVERS > KMD > SRC > KMDENTRY#261.CPP > AtiQueryChildStatu

```
293
294     // Adapter functions
295     DriverInitializationData.DxgkDdiQueryAdapterInfo            = Dispatch_QueryAdapterInfo;
296     DriverInitializationData.DxgkDdiCreateDevice                = Dispatch_CreateDevice;
297     DriverInitializationData.DxgkDdiCreateAllocation            = Dispatch_CreateAllocation;
298     DriverInitializationData.DxgkDdiDestroyAllocation           = Dispatch_DestroyAllocation;
299     DriverInitializationData.DxgkDdiDescribeAllocation          = Dispatch_DescribeAllocation;
300     DriverInitializationData.DxgkDdiGetStandardAllocationDriverData = Dispatch_GetStandardAllocationDriverData;
301     DriverInitializationData.DxgkDdiAcquireSwizzlingRange       = Dispatch_AcquireSwizzlingRange;
302     DriverInitializationData.DxgkDdiReleaseSwizzlingRange       = Dispatch_ReleaseSwizzlingRange;
303     DriverInitializationData.DxgkDdiPatch                       = Dispatch_Patch;
304     DriverInitializationData.DxgkDdiSubmitCommand               = Dispatch_SubmitCommand;
305     DriverInitializationData.DxgkDdiBuildPagingBuffer           = Dispatch_BuildPagingBuffer;
306     DriverInitializationData.DxgkDdiSetPalette                  = Dispatch_SetPalette;
307     DriverInitializationData.DxgkDdiSetPointerShape             = Dispatch_SetPointerShape;
308     DriverInitializationData.DxgkDdiSetPointerPosition          = Dispatch_SetPointerPosition;
309     DriverInitializationData.DxgkDdiPreemptCommand              = Dispatch_PreemptCommand;
310     DriverInitializationData.DxgkDdiResetFromTimeout            = Dispatch_ResetFromTimeout;
311     DriverInitializationData.DxgkDdiRestartFromTimeout          = Dispatch_RestartFromTimeout;
312     DriverInitializationData.DxgkDdiEscape                      = Dispatch_Escape;
313     DriverInitializationData.DxgkDdiCollectDbgInfo              = Dispatch_CollectDbgInfo;
314     DriverInitializationData.DxgkDdiQueryCurrentFence           = Dispatch_QueryCurrentFence;
315     DriverInitializationData.DxgkDdiStopCapture                 = Dispatch_StopCapture;
316     // MMDMM support:
317     DriverInitializationData.DxgkDdiIsSupportedVidPn            = Dispatch_IsSupportedVidPn;
318     DriverInitializationData.DxgkDdiRecommendFunctionalVidPn    = Dispatch_RecommendFunctionalVidPn;
319     DriverInitializationData.DxgkDdiEnumVidPnCofuncModality     = Dispatch_EnumVidPnCofuncModality;
320     DriverInitializationData.DxgkDdiSetVidPnSourceVisibility    = Dispatch_SetVidPnSourceVisibility;
```

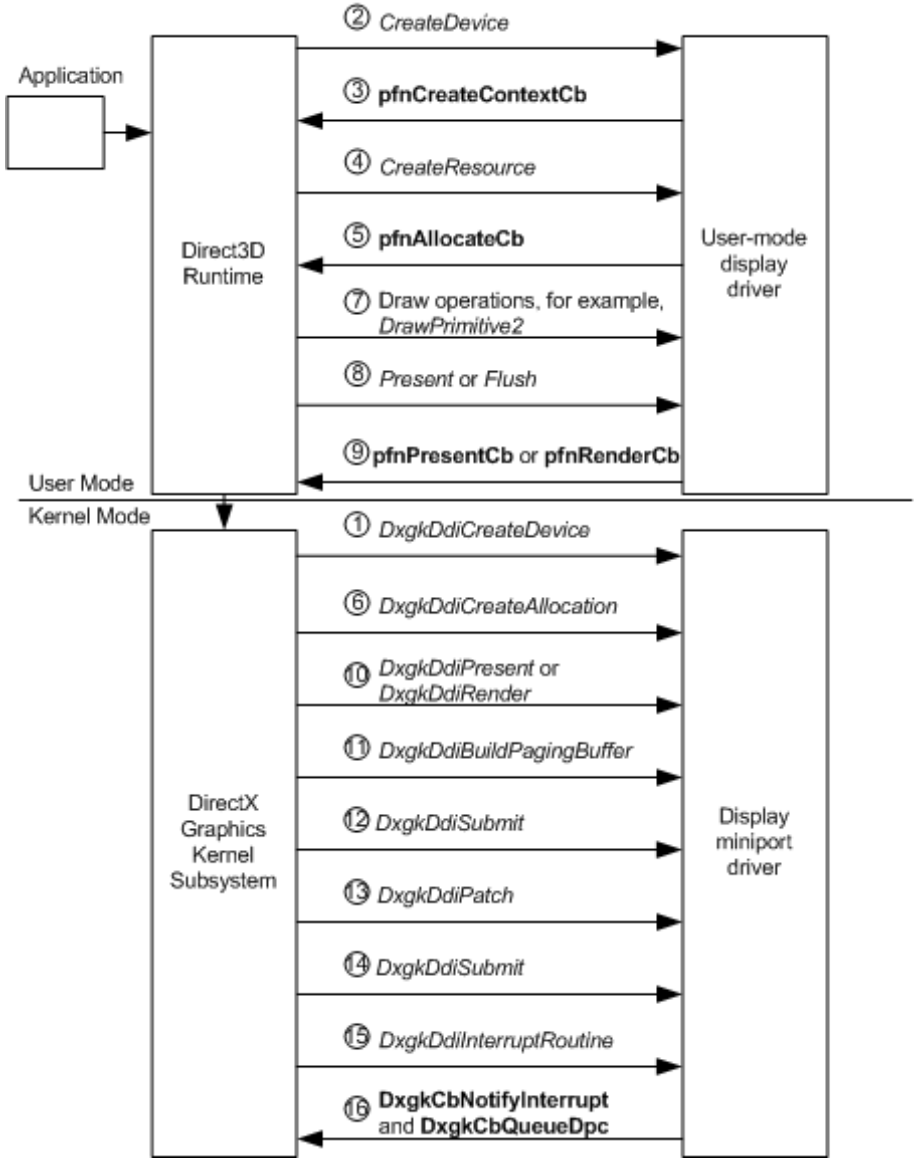
KMDENTRY.cpp code



the flow of Windows Display Driver Model (WDDM) operations

Creating a Rendering Device

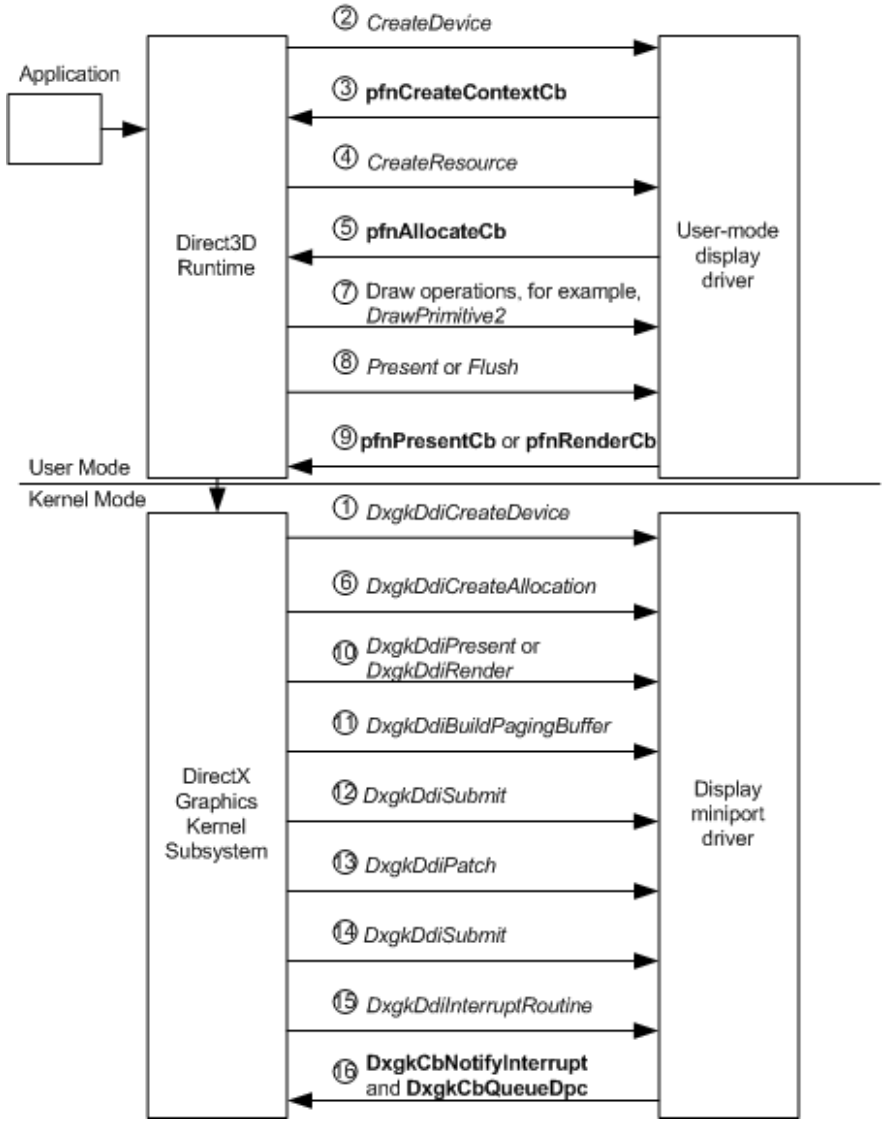
- **DxgkDdiCreateDevice**
- CreateDevice
- pfnCreateContextCb



the flow of Windows Display Driver Model (WDDM) operations

Creating Surfaces for a Device

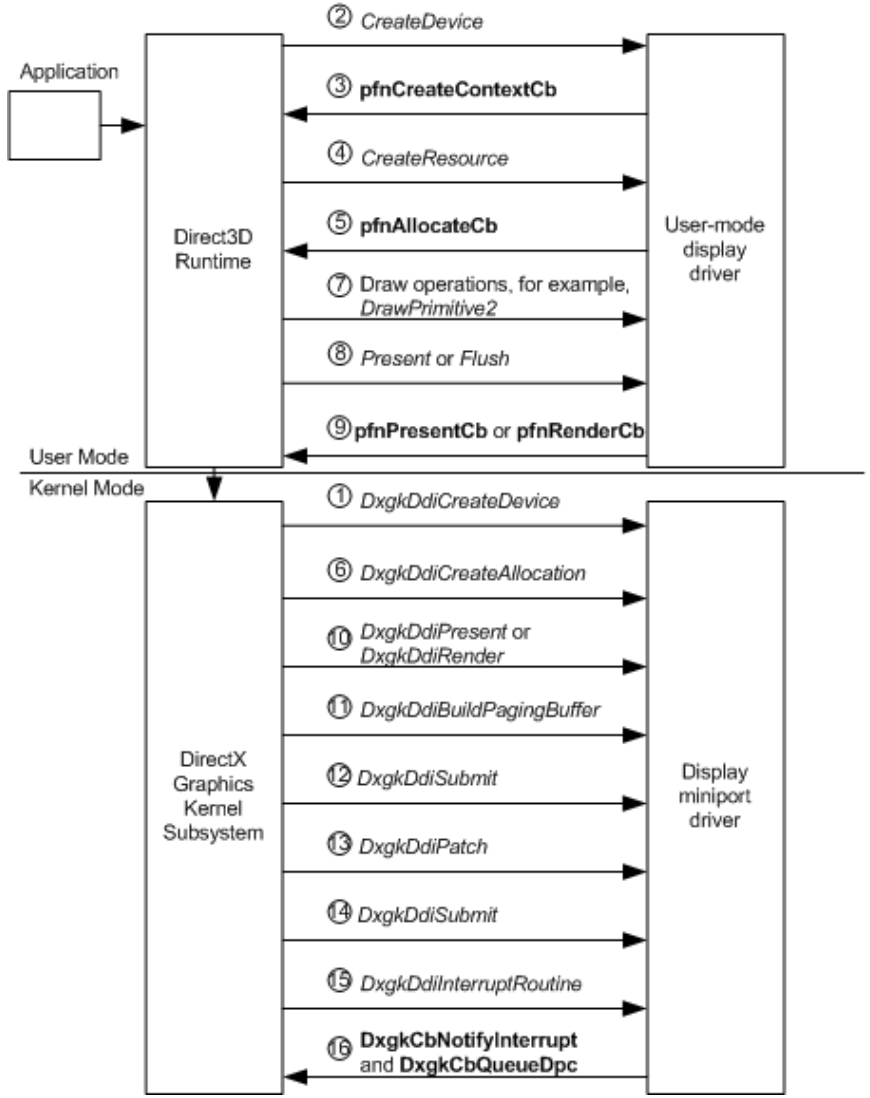
- CreateResource
- pfnAllocateCb
- DxgkDdiCreateAllocation



the flow of Windows Display Driver Model (WDDM) operations

Submitting the Command Buffer to Kernel Mode

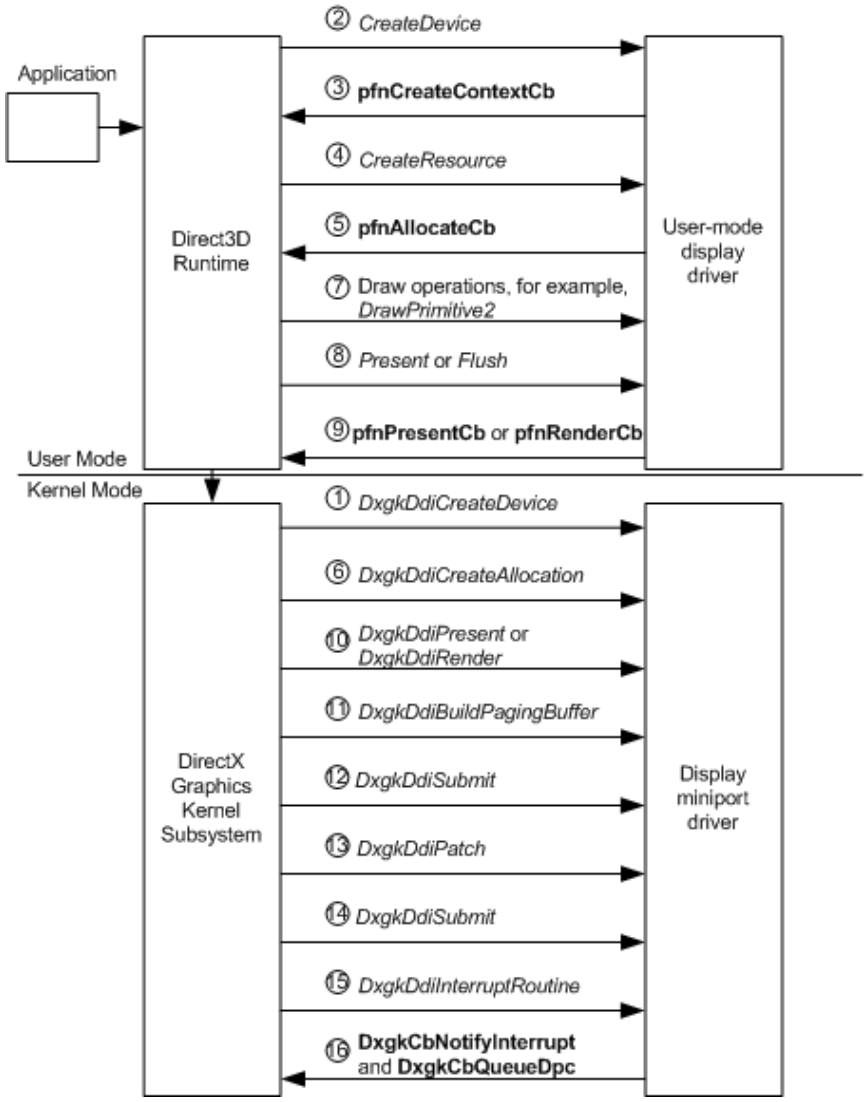
- Draw operations, for example, DrawPrimitive2
- Present or Flush
- Present ---> pfnPresentCb ---> DxgkDdiPresent
- Flush ---> pfnRenderCb ---> DxgkDdiRender



the flow of Windows Display Driver Model (WDDM) operations

Submitting the DMA Buffer to Hardware

- DxgkDdiBuildPagingBuffer
- DxgkDdiSubmitCommand
- DxgkDdiPatch
- DxgkDdiSubmitCommand
- DxgkDdiInterruptRoutine
- DxgkCbNotifyInterrupt
- DxgkCbQueueDpc



the flow of Windows Display Driver Model (WDDM) operations

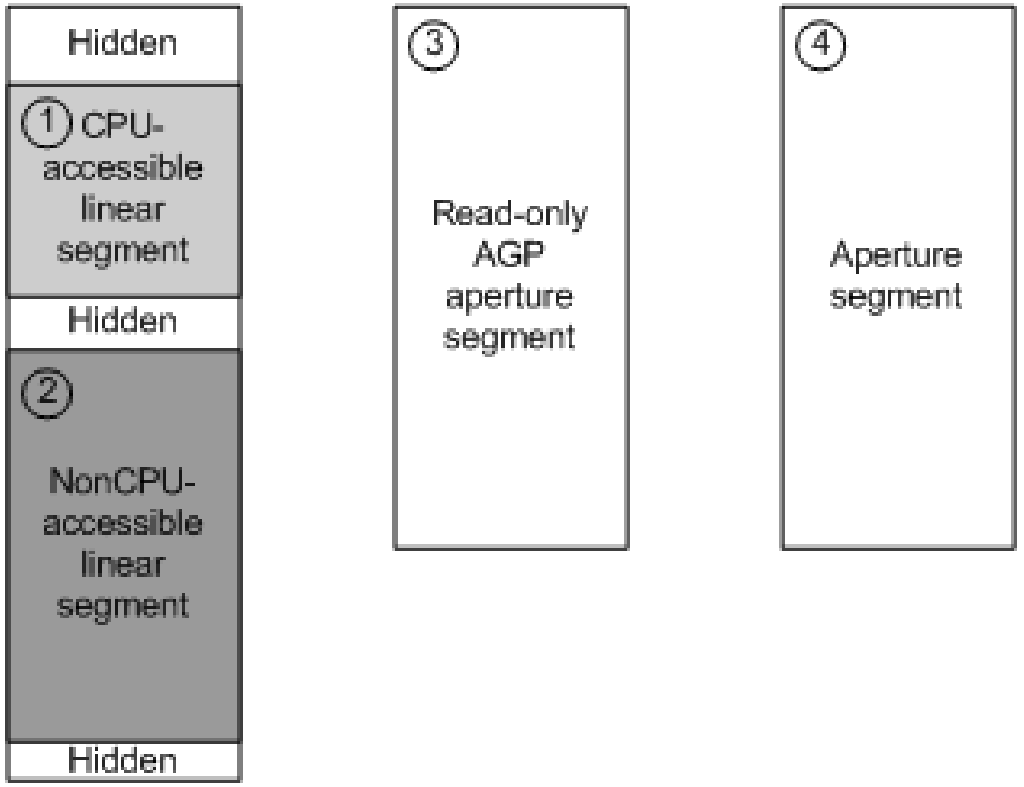
Knowledge of Video Memory Management

Memory Segment Types

The video memory manager and display hardware only support certain types of memory segments, so the display miniport driver can only configure segments of those types.

The display miniport driver can configure the following types of memory segments:

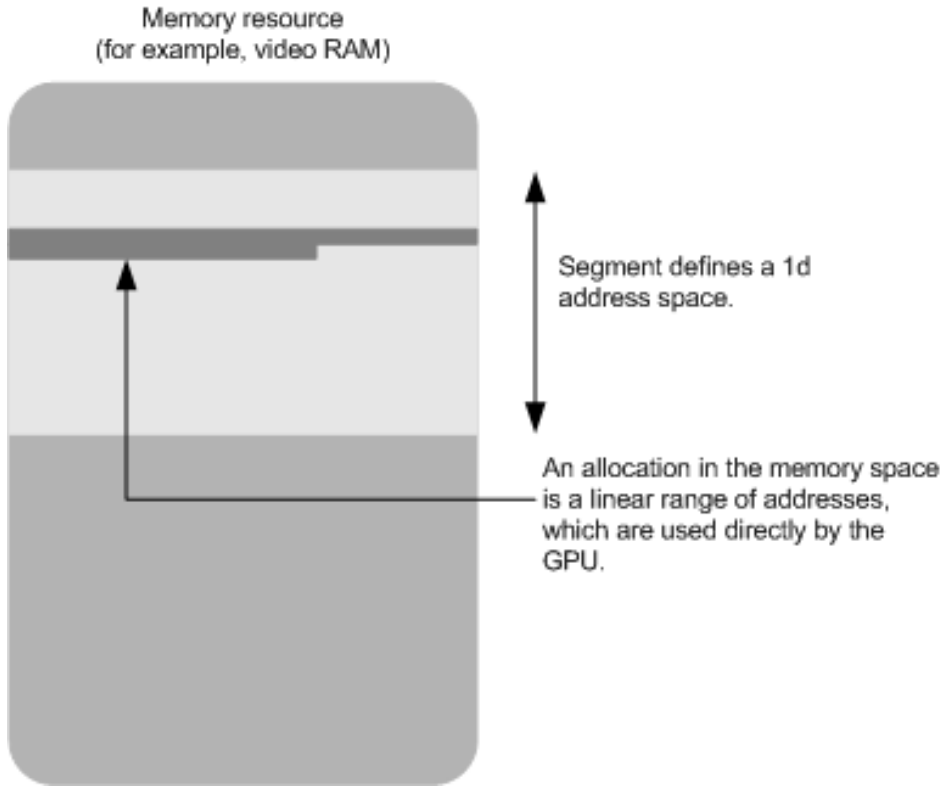
- Linear Memory-Space Segments
- Linear Aperture-Space Segments
- AGP-Type Aperture-Space Segments



Driver configure memory segments from the GPU address space

Linear Memory-Space Segments

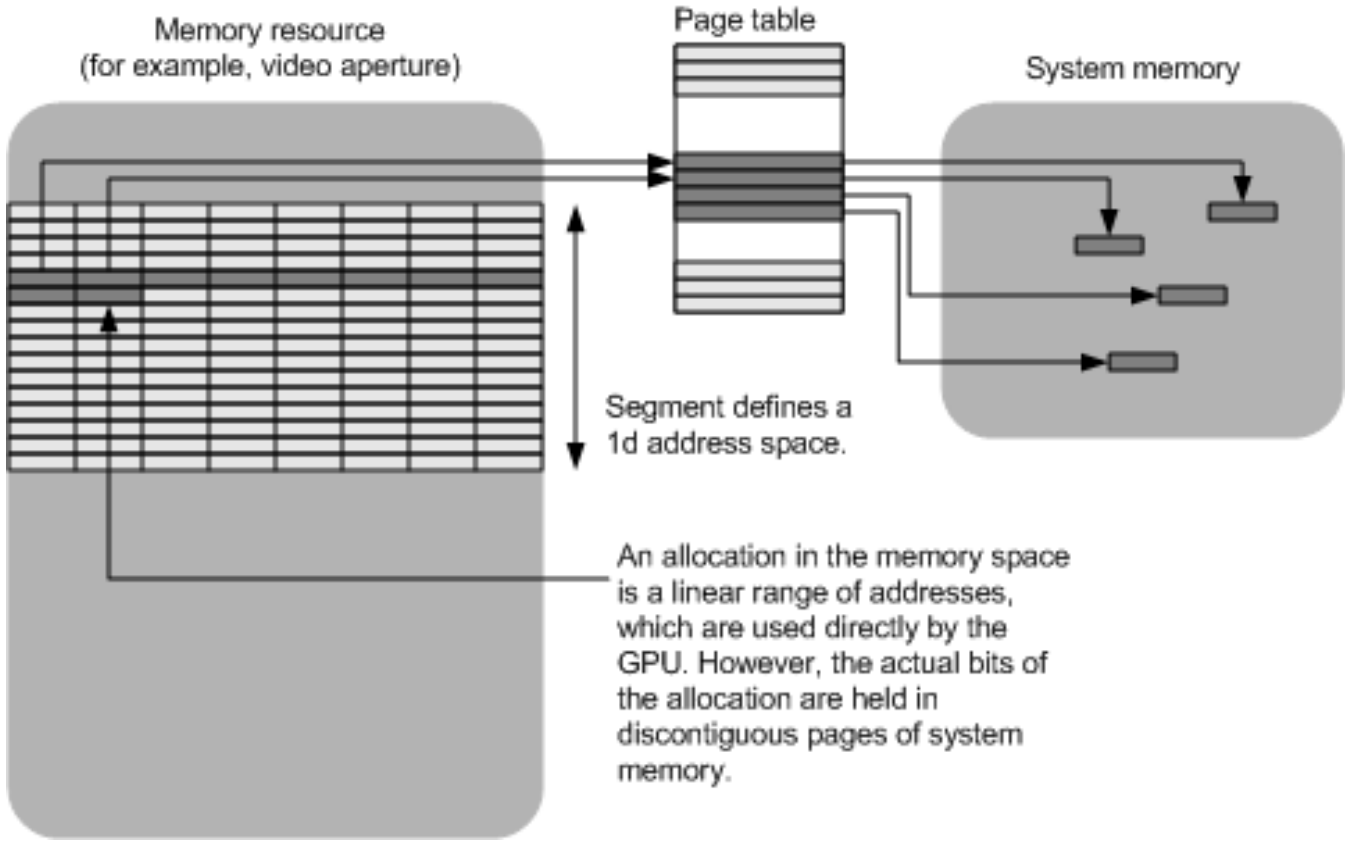
- Virtualizes video memory located on the graphics adapter.
- Is accessed directly by the GPU.
- Is managed linearly in a one-dimensional address space.



a visual representation of a linear memory-space segment

Linear Aperture-Space Segments

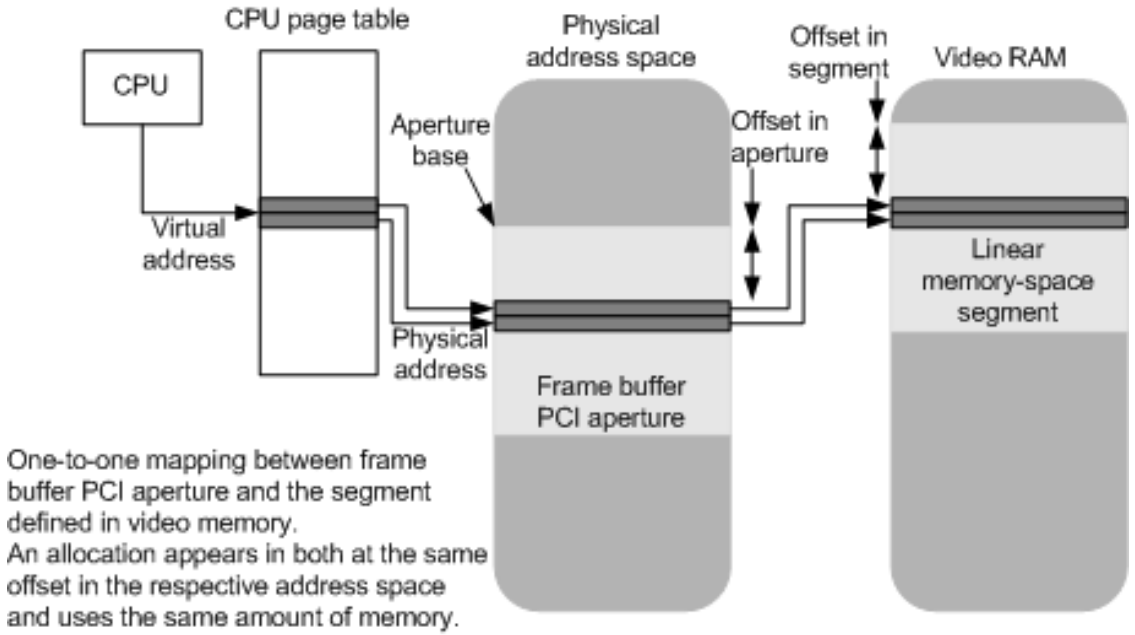
A linear aperture-space segment is similar to a linear memory-space segment; however, the aperture-space segment is only an address space and cannot hold bits.



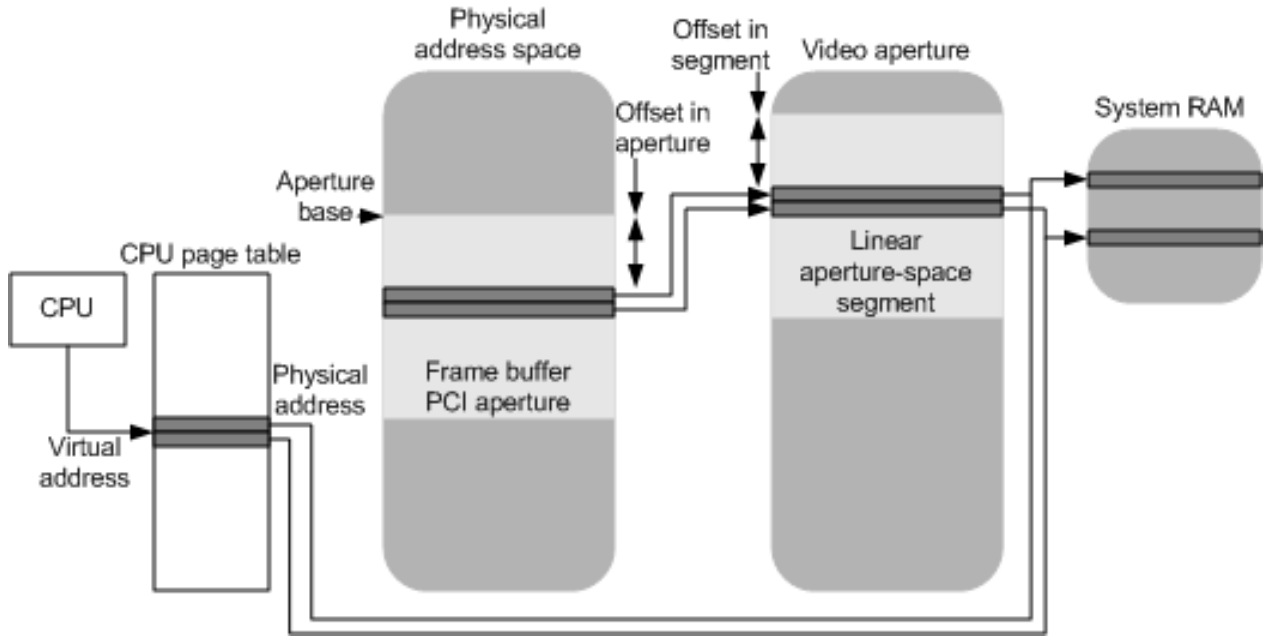
a visual representation of a linear aperture-space segment

Mapping Virtual Addresses to a Memory Segment

The display miniport driver can specify, for each memory-space or aperture-space segment that it defines, whether CPU virtual addresses can map directly to an allocation located in the segment by setting the CpuVisible bit-field flag in the Flags member of the DXGK_SEGMENTDESCRIPTOR structure for the segment.



virtual addresses are mapped to a linear memory-space segment



virtual addresses are mapped to the underlying pages of a linear aperture-space segment

Introduction to Command and DMA Buffers

Command and DMA buffers closely resemble each other. However, a command buffer is used by the user-mode display driver, and a DMA buffer is used by the display miniport driver.

A command buffer has the following characteristics:

- It is never directly accessed by the GPU.
- The hardware vendor controls the format.
- It is allocated for the user-mode display driver from regular pageable memory in the private address space of the tendering application.

4 GB

A DMA buffer has the following characteristics:

- It is based on the validated content of a command buffer.
- It is allocated by the display miniport driver from kernel pageable memory.
- Before the GPU can read from a DMA buffer, the display miniport driver must page-lock the DMA buffer and map the DMA buffer through an aperture.

Tasks in the WDDM

Tasks in the WDDM

- Requesting and using surface memory
- Specifying memory type for a resource
- Locking memory
- Locking swizzled allocations
- Manipulating 3-D virtual textures directly from hardware
- Registering hardware information

Requesting and Using Surface Memory

- User-mode display driver receive CreateResource function when creating a list of surfaces.
- The user-mode display driver calls the pfnAllocate function to allocate memory for the surfaces.
- When calling the pfnRenderCb function to submit a command buffer to the display miniport driver, the user-mode display driver uses the allocation handles that correspond to the surfaces.
- The display miniport driver can call the DxgkCbGetHandleData function to determine to which surface allocations the user-mode display driver refers.

Specifying Memory Type for a Resource

The memory type is specified as either system or video memory through the Pool member of the D3DDDIARG_CREATERESOURCE structure.

- D3DDDIPOOL_LOCALVIDMEM------the driver use local video memory.
- D3DDDIPOOL_NONLOCALVIDMEM------the driver use nonlocal video memory (for example, AGP memory).

```
typedef struct _D3DDDIARG_CREATERESOURCE {
  [in]     D3DDDIFORMAT                    Format;
  [in]     D3DDDI_POOL                     Pool;
  [in]     D3DDDIMULTISAMPLE_TYPE          MultisampleType;
  [in]     UINT                            MultisampleQuality;
  [in]     const D3DDDI_SURFACEINFO        *pSurfList;
  [in]     UINT                            SurfCount;
  [in]     UINT                            MipLevels;
  [in]     UINT                            Fvf;
  [in]     D3DDDI_VIDEO_PRESENT_SOURCE_ID  VidPnSourceId;
  [in]     D3DDDI_RATIONAL                 RefreshRate;
  [in/out] HANDLE                          hResource;
  [in]     D3DDDI_RESOURCEFLAGS            Flags;
  [in]     D3DDDI_ROTATION                 Rotation;
} D3DDDIARG_CREATERESOURCE;
```

the D3DDDIARG_CREATERESOURCE structure

Next Work

Weeks 5-6: KMD training to learn about KMD2 and KMD3 Architecture.

Q & A

Thanks