D3D12 GPU Upload Heaps 소개 및 적용

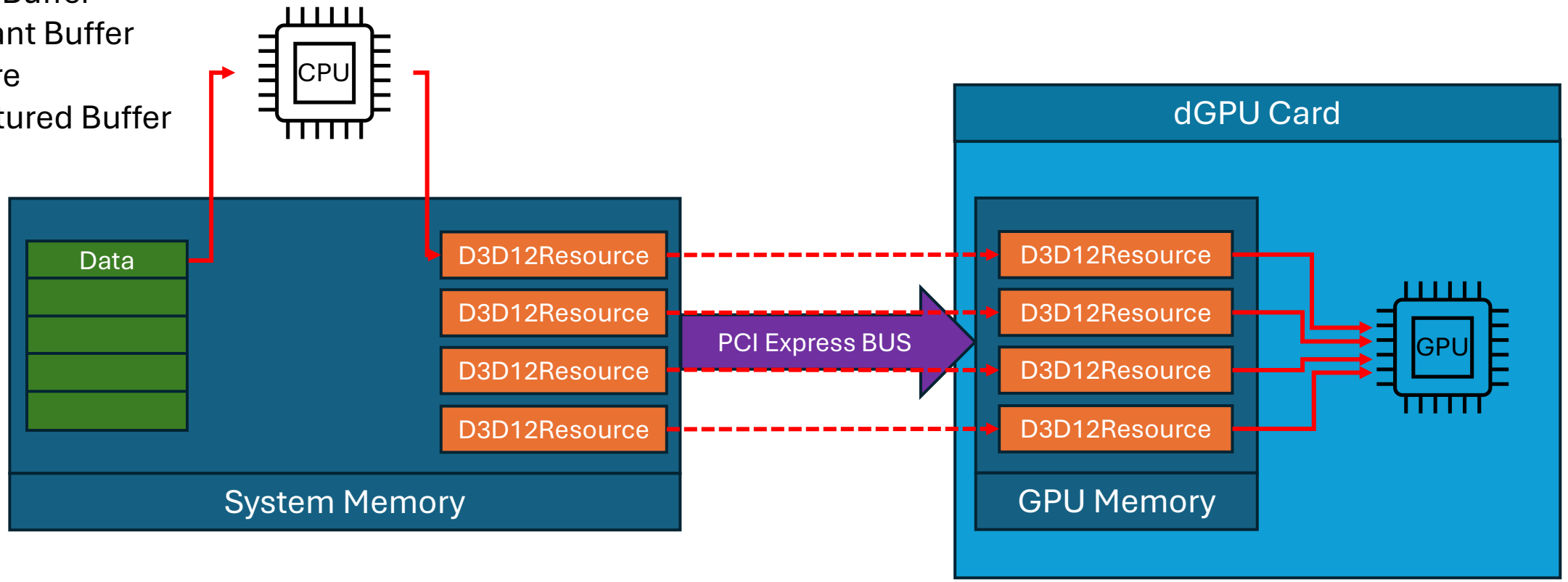
유영천

https://megayuchi.com

GPU에서 리소스를 억세스하는 방법

Asynchronous PCIe DMA Transfer

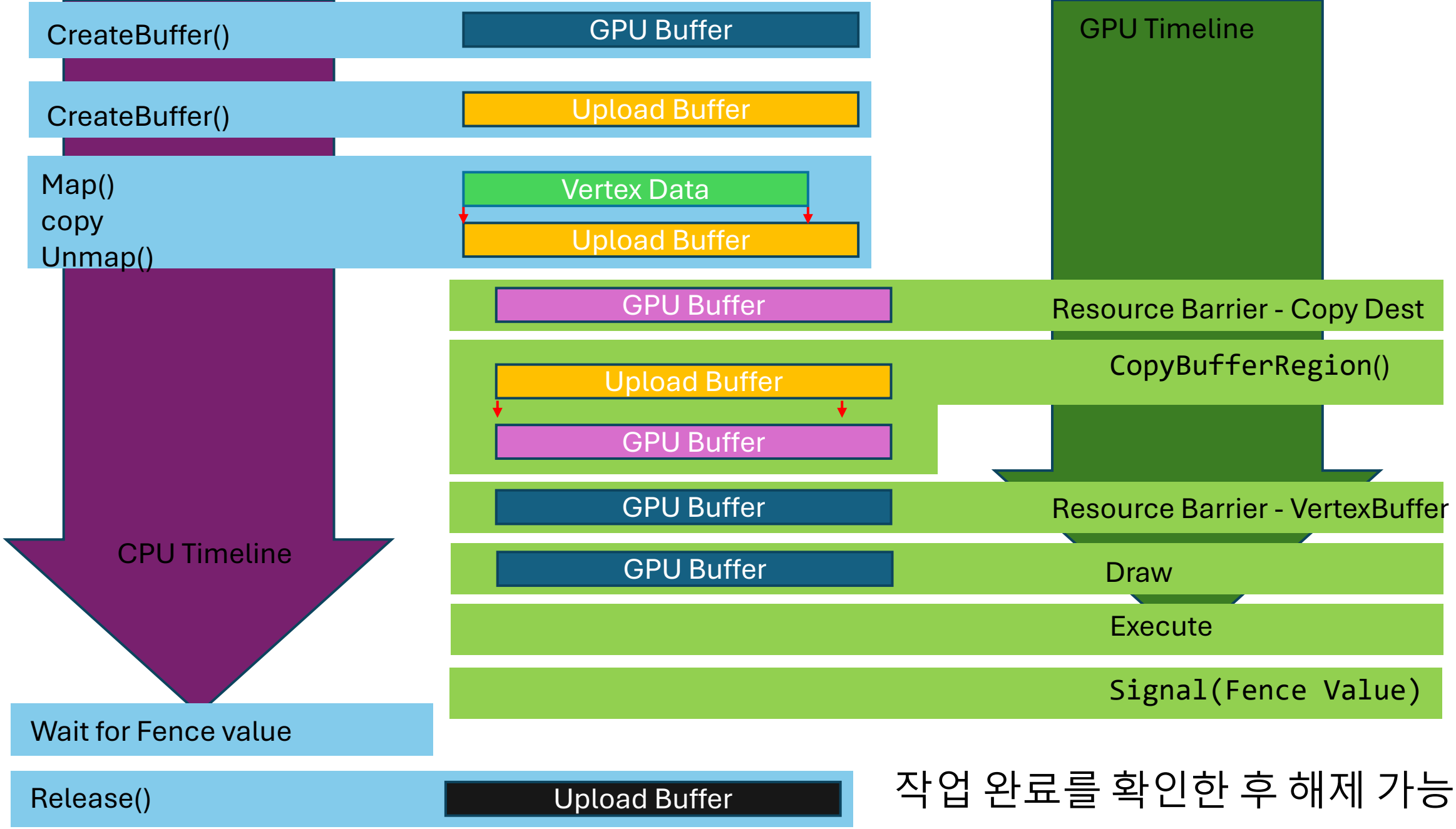
Vertex Buffer
Index Buffer
Contant Buffer
Texture
Structured Buffer



Copy data
System Memory → GPU Memory

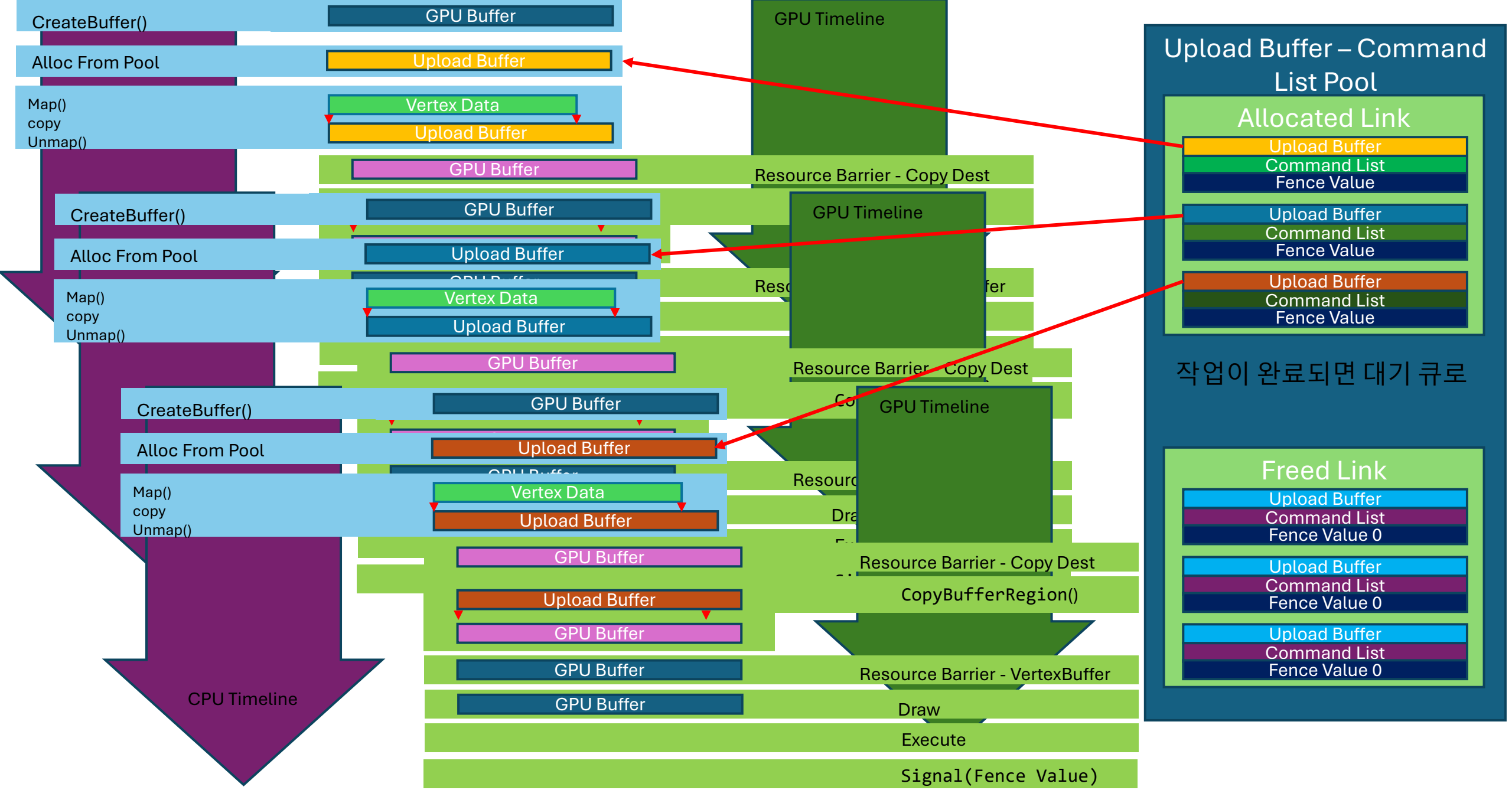
Asynchronous PCIe DMA Transfer

- 리소스가 GPU메모리에 있음.
- 리소스를 GPU에 올려놓기 위해 CPU가 시스템 메모리에 데이터를 써넣고 이를 DMA를 통해 GPU메모리로 전송.
- GPU리소스에 데이터를 업데이트하는 기본적인 방법.
- GPU에서 리소스를 소비할 때의 성능 가장 좋음.
- 모든 리소스 타입에 대해 사용 가능.
- 추가 코드 잔뜩 붙음.
  - 단순 구현 – 전송 요청 -> fence -> wait : 엄청나게 느리다.
  - 실제 사용을 위해서는 복잡한 비동기 업데이트 매니저 코드가 필요.



작업 완료를 확인한 후 해제 가능

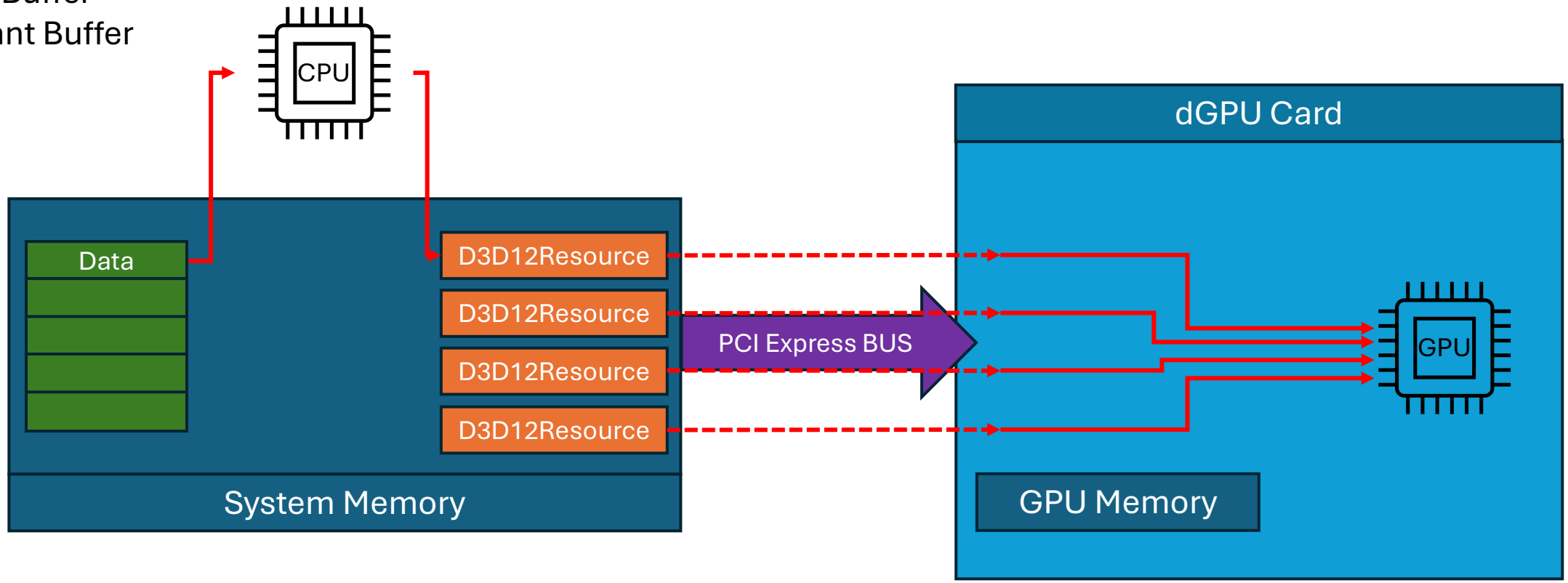
단순 구현 – 전송 요청 -> fence -> wait



비동기 업데이트 매니저 구현

GPUMMU or GART (Graphics Address Remapping Table)

Vertex Buffer
Index Buffer
Contant Buffer



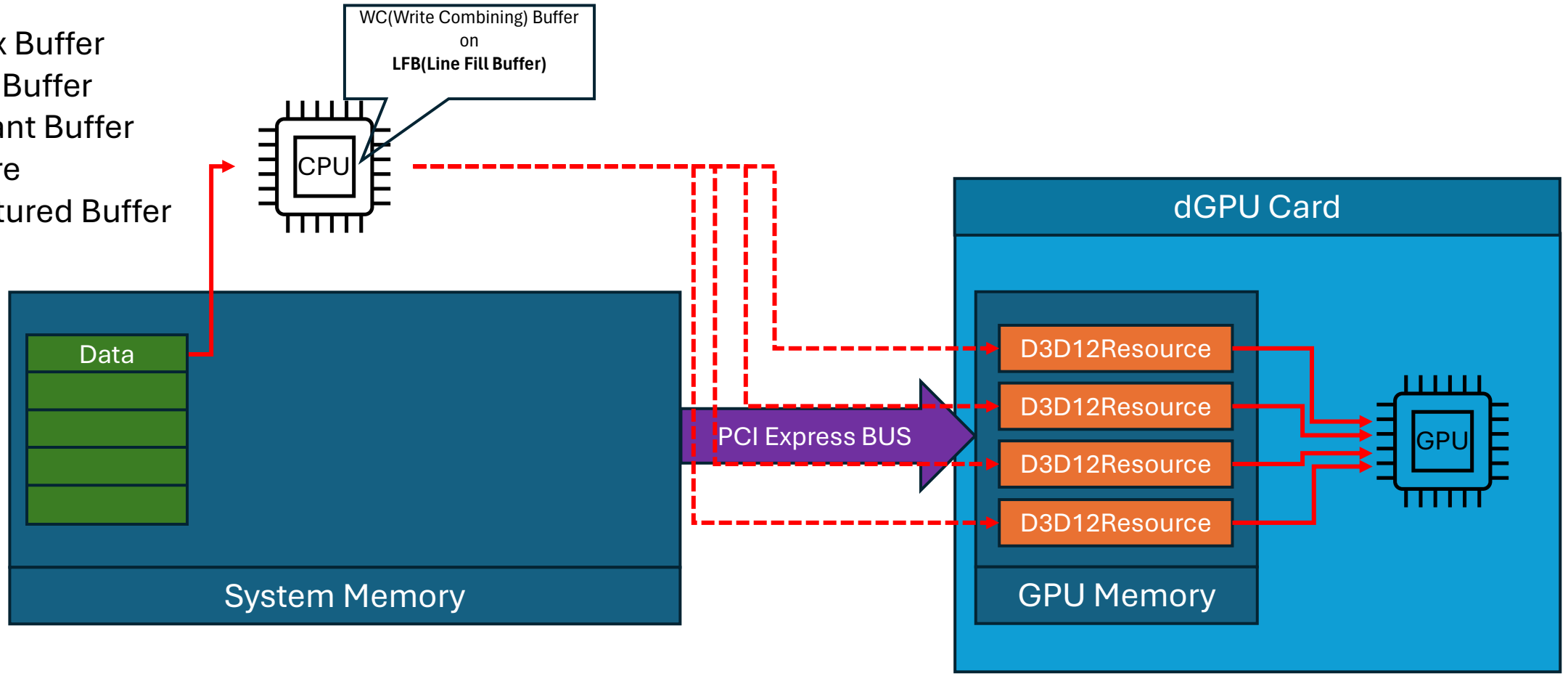
Zero copy

GPUMMU or GART (Graphics Address Remapping Table)

- 시스템 메모리에 리소스를 올려두고 GPU가 이를 직접 사용.
- 추가 코드 없이 CPU가 직접 시스템 메모리에 데이터를 써넣으므로 업데이트 빠름.
- GPU에서 리소스를 소비할 때의 성능은 떨어짐.
- 간결한 코드.
- 성능 문제와 구조적 문제로 Texture, Structured Buffer에 대해 사용 불가.

BAR (Base Address Register) Mapping - MMIO

Vertex Buffer
Index Buffer
Contant Buffer
Texture
Structured Buffer



Zero copy

BAR (Base Address Register) Mapping - MMIO

- MMIO를 이용해서 CPU가 GPU메모리에 직접 데이터를 써넣음.
- MMIO의 구조적 한계로 인한 쓰기 성능 저하 가능성 있음.
- 쓰기 성능 저하는 Write Combining기법으로 보완.
- 256BM제약으로 한동안 그래픽 API에서 사용할 수 없었음.

Resizable BAR

- 기존 BAR방식으로는 256MB맵핑만 가능
  - 한 프로세스가 256MB공간을 점유하면 다른 프로세스는 사용불가.
- BAR의 256MB 제한을 넘어서 GPU 메모리 전체를 MMIO 주소공간으로 맵핑가능.
  - BIOS에서 Resizable BAR 활성화 → OS 부팅 시 PCIe 장치(GPU)의 BAR 크기를 조정.
  - GPU Memory 전체를 MMIO 주소 공간에 직접 매핑.
  - CPU가 해당 MMIO 영역을 직접 접근 가능.
- PCI-E 3.0이상부터 지원.
- WDDM 2.0부터 Resizable BAR사용 가능.
- nvidia 기준 RTX3000이상부터 지원.
  - 지원 안되는 하드웨어가 아직은 많다....

GPU Upload Heaps

- Resizable BAR를 이용하여 GPU Memory전체에 대해 MMIO주소 공간으로 맵핑 가능.
- MMIO로 맵핑된 GPU Memory를 D3D12 Resource로 사용.
- CPU가 직접 읽고 쓰기 가능(읽기는 가능은 하지만 성능상의 이유로 권장되지 않음).
- 내부적으로 PCI-E버스를 통한 전송이므로 system memory쓰기보다 느리지만 Write combining Buffer를 사용할 수 있으므로 쓰기에 있어서는 성능저하가 적다.
- 비동기 전송을 통한 방식에 비해 copy작업이 없고 전송준비/전송확인 코드가 필요없어 성능향상을 기대할 수 있다.
- 코드가 간결해짐.

Vertex Buffer / Index Buffer / Constant Buffer / Structured Buffer

```
ID3D12Resource* pVertexBuffer = nullptr;
UINT VertexBufferSize = SizePerVertex * dwVertexNum;

if (FAILED(pDevice->CreateCommittedResource(
    &CD3DX12_HEAP_PROPERTIES(D3D12_HEAP_TYPE_GPU_UPLOAD),
    D3D12_HEAP_FLAG_NONE,
    &CD3DX12_RESOURCE_DESC::Buffer(VertexBufferSize),
    D3D12_RESOURCE_STATE_COMMON,
    nullptr,
    IID_PPV_ARGS(&pVertexBuffer)))
{
    __debugbreak();
}

CD3DX12_RANGE readRange(0, 0);
char* pDest = nullptr;
pVertexBuffer->Map(0, &readRange, (void**)&pDest);
memcpy(pDest, pSrcData, VertexBufferSize);
pVertexBuffer->Unmap(0, nullptr);
```

Texture

```
// Create Texture for Render
if (FAILED(pD3DDevice->CreateCommittedResource(
    &CD3DX12_HEAP_PROPERTIES(D3D12_HEAP_TYPE_GPU_UPLOAD),
    D3D12_HEAP_FLAG_NONE,
    &textureDesc,
    D3D12_RESOURCE_STATE_ALL_SHADER_RESOURCE,
    nullptr,
    IID_PPV_ARGS(&pTexResource))))
{
    __debugbreak();
}

D3D12_SUBRESOURCE_DATA textureData = {};
textureData.pData = temp.get();
textureData.RowPitch = rowPitch;
textureData.SlicePitch = textureData.RowPitch * theight;

pTexResource->WriteToSubresource(0, nullptr, textureData.pData, textureData.RowPitch,
textureData.SlicePitch);
```

Texture데이터는 D3D Resource상에서 선형적으로 배치되지 않으므로 선형적으로 배열된 픽셀 데이터를 D3D Resource의 메모리에 직접 copy해서는 안된다.

Update방법 비교

1. [fence/wait] CPU->UploadBuffer(System)->DMA Copy->GPU-Resource -> fence/wait
   - 예제에서나 사용하는 정도. 실전에서 사용하기엔 성능이 너무 떨어짐.
2. [no wait] CPU->UploadBuffer(System)->DMA Copy(async update manager)->GPU-Resource
   - 1에 비해서는 압도적인 성능. GPU Upload Heaps feature가 지원되지 않는 장비에선 선택의 여지가 없다. 무조건 구현해야 함.
3. [no wait] CPU->UploadBuffer(GPU)
   - 2에 비해서 5%-30% 성능 향상이 있음. 코드가 간결해져서 이쪽으로 통일하면 좋겠지만 지원되지 않는 장비가 여전히 많음. 옵션으로 지원.

아키텍처에 따른 D3D12 Resource의 메모리 속성

- UMA지원 : GPU/CPU 통합메모리 시스템
- UMA장치에서 CacheCoherentUMA 지원 :
  - CPU와 GPU 간 메모리 억세스 시 캐시 동기화가 자동으로 유지됨.
- MemoryPoolPreference
  - L0 : System Memory
  - L1 : GPU Memory
- CPUPageProperty
  - WRITE_BACK
    - 일반적인 캐시 정책. 캐시에 먼저쓰고 나중(필요한 때)에 메모리에 써넣기.
    - 읽기/쓰기/랜덤억세스에서 적당히 좋은 성능.
    - Cache coherency보장됨
  - WRITE_COMBINE
    - WC Buffer에 데이터를 모아서(일반적으로 64 bytes) 일괄적으로 써넣기.
    - 연속된 데이터일 때 쓰기 매우 빠름/읽기 많이 느림.
    - Cache coherency보장 안됨

아키텍처에 따른 D3D12 Resource의 메모리 속성

D3D12_FEATURE_DATA_ARCHITECTURE::UMA == FALSE

Heap Type	**How the returned D3D12_HEAP_PROPERTIES members convert**
D3D12_HEAP_TYPE_UPLOAD	CPUPageProperty = WRITE_COMBINE, MemoryPoolPreference = L0.
D3D12_HEAP_TYPE_DEFAULT	CPUPageProperty = NOT_AVAILABLE, MemoryPoolPreference = L1.
D3D12_HEAP_TYPE_READBACK	CPUPageProperty = WRITE_BACK, MemoryPoolPreference = L0.
D3D12_HEAP_TYPE_GPU_UPLOAD	CPUPageProperty = WRITE_COMBINE, MemoryPoolPreference = L1.

아키텍처에 따른 D3D12 Resource의 메모리 속성

D3D12_FEATURE_DATA_ARCHITECTURE::UMA == TRUE
D3D12_FEATURE_DATA_ARCHITECTURE::CacheCoherentUMA == FALSE

Heap Type	**How the returned D3D12_HEAP_PROPERTIES members convert**
D3D12_HEAP_TYPE_UPLOAD	CPUPageProperty = WRITE_COMBINE, MemoryPoolPreference = L0.
D3D12_HEAP_TYPE_DEFAULT	CPUPageProperty = NOT_AVAILABLE, MemoryPoolPreference = L0.
D3D12_HEAP_TYPE_READBACK	CPUPageProperty = WRITE_BACK, MemoryPoolPreference = L0.
D3D12_HEAP_TYPE_GPU_UPLOAD	CPUPageProperty = WRITE_COMBINE, MemoryPoolPreference = L0.

아키텍처에 따른 D3D12 Resource의 메모리 속성

D3D12_FEATURE_DATA_ARCHITECTURE::UMA == TRUE
D3D12_FEATURE_DATA_ARCHITECTURE::CacheCoherentUMA == TRUE

Heap Type	**How the returned D3D12_HEAP_PROPERTIES members convert**
D3D12_HEAP_TYPE_UPLOAD	CPUPageProperty = WRITE_BACK, MemoryPoolPreference = L0.
D3D12_HEAP_TYPE_DEFAULT	CPUPageProperty = NOT_AVAILABLE, MemoryPoolPreference = L0.
D3D12_HEAP_TYPE_READBACK	CPUPageProperty = WRITE_BACK, MemoryPoolPreference = L0.
D3D12_HEAP_TYPE_GPU_UPLOAD	CPUPageProperty = WRITE_BACK, MemoryPoolPreference = L0.

예제분석

- https://github.com/megayuchi/D3D12Lecture
  - 23번 예제

성능비교 – 맵 로딩 시간 비교

	Name	CPU	Working set (memory)	Memory (active private working set)	Paged pool	Threads	Platform	GPU	GPU engine	Dedicated GPU memory	Shared GPU memory	Loading Time(ms)
[D3D12]												
Map-V101												
GPU Upload Heaps(release)	Client_x64_release.exe	3	1,512,508 K	1,434,972 K	2,116 K	211	64 bit	34	GPU 1 - 3D	826,240 K	477,400 K (-11%)	725.1 (-10%)
UpdateResource(release)	Client_x64_release.exe	3	1,586,492 K	1,509,016 K	2,179 K	212	64 bit	35	GPU 1 - 3D	827,656 K	537,176 K	800.75
Map-V111												
GPU Upload Heaps(release)	Client_x64_release.exe	14	2,273,780 K	2,196,240 K	2,863 K	212	64 bit	46	GPU 1 - 3D	999,320 K	655,680 K (-6%)	2938.53 (-30%)
UpdateResource(release)	Client_x64_release.exe	14	2,322,428 K	2,244,944 K	2,930 K	212	64 bit	47	GPU 1 - 3D	1,000,992 K	697,856 K	4151.67
[D3D12-Raytracing]												
Map-V101												
GPU Upload Heaps(release)	Client_x64_release.exe	5	1,731,996 K	1,585,204 K	2,267 K	213	64 bit	99	GPU 1 - 3D	1,200,832 K	572,224 K (-8%)	440.8 (-10.5%)
UpdateResource(release)	Client_x64_release.exe	5	1,781,976 K	1,635,596 K	2,328 K	212	64 bit	99	GPU 1 - 3D	1,198,980 K	618,320 K	493.04
Map-V111												
GPU Upload Heaps(release)	Client_x64_release.exe	11	2,194,496 K	2,048,108 K	2,842 K	213	64 bit	97	GPU 1 - 3D	2,443,120 K	745,908 K (-5%)	2382.58 (-22%)
UpdateResource(release)	Client_x64_release.exe	10	2,243,956 K	2,097,136 K	2,885 K	212	64 bit	98	GPU 1 - 3D	2,445,312 K	784,356 K	3057.07

참고자료

- GPU Upload Heaps
  - https://microsoft.github.io/DirectX-Specs/d3d/D3D12GPUUploadHeaps.html

- GPUMMU
  - https://learn.microsoft.com/en-us/windows-hardware/drivers/display/gpu-virtual-memory-in-wddm-2-0

- IOMMU
  - https://learn.microsoft.com/ko-kr/windows-hardware/drivers/display/iommu-model