D3D12 리소스 관리 전략

유영천

https://megayuchi.com

tw:@dgtman

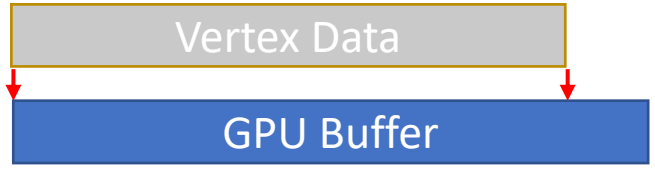
D3D11 Create – Update - Draw

- CreateBuffer()
- UpdateSubResource()
- Draw()

D3D11 Create – Update - Draw

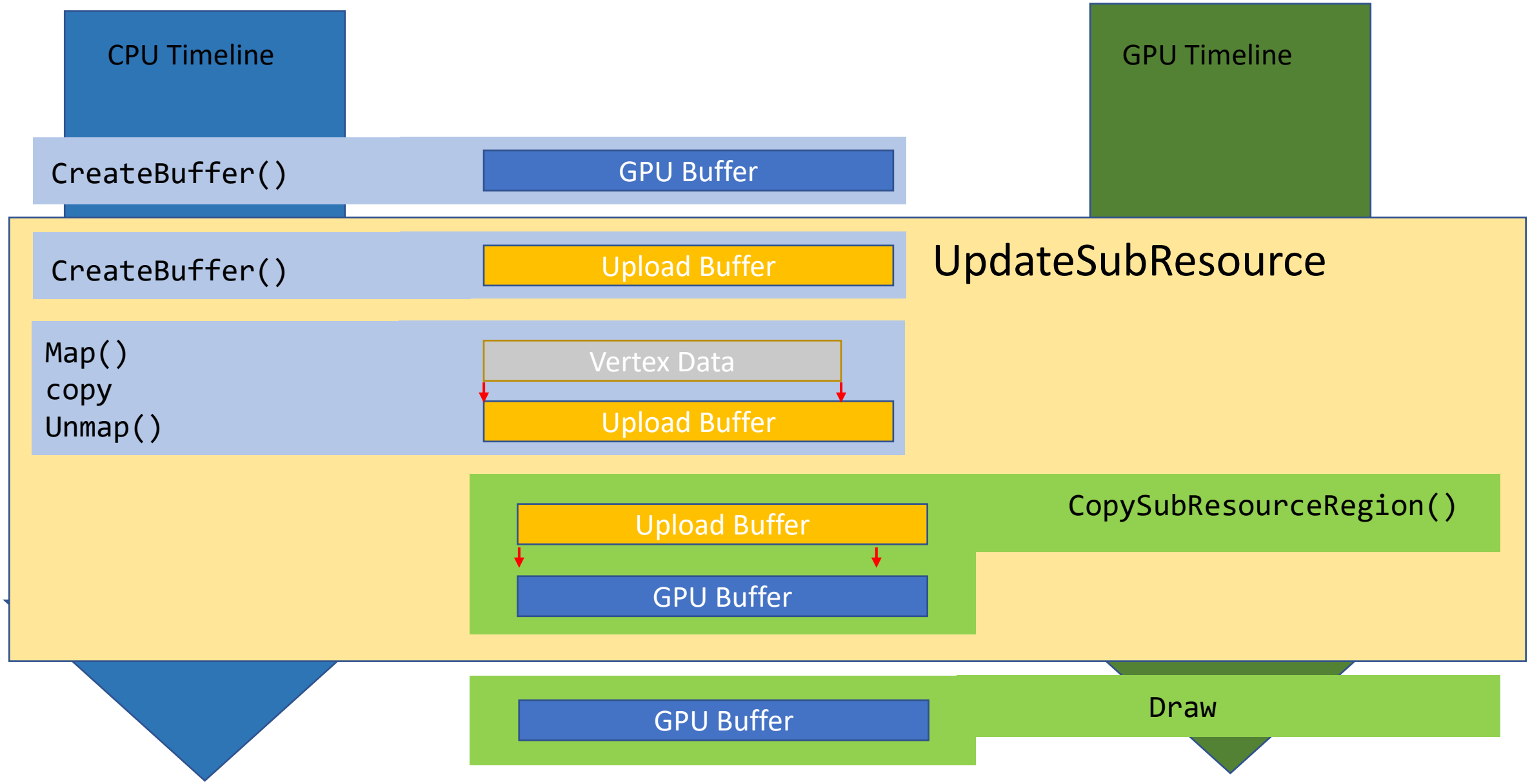
CreateBuffer()

UpdateSubResource



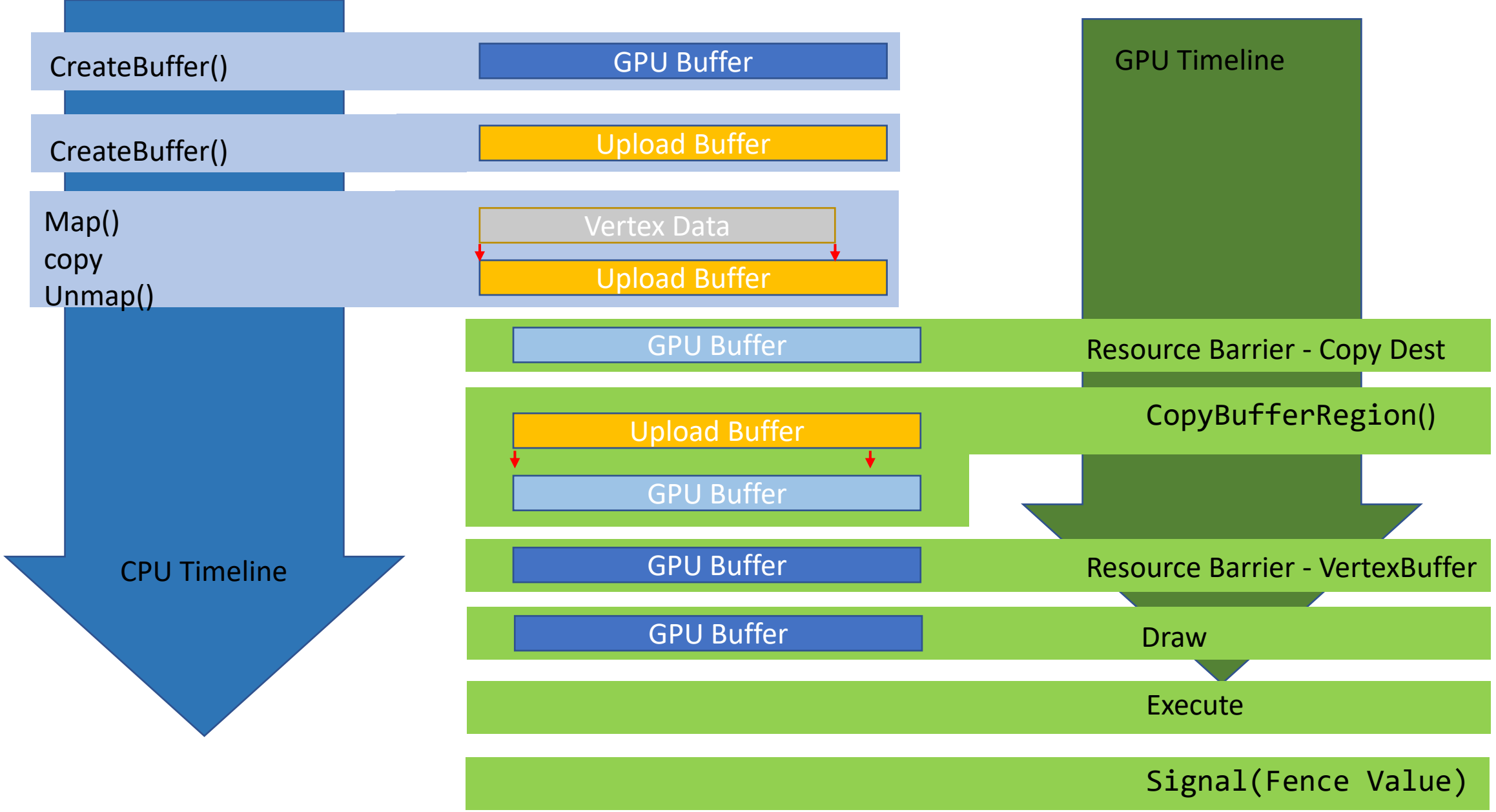
Draw

D3D11- Manually

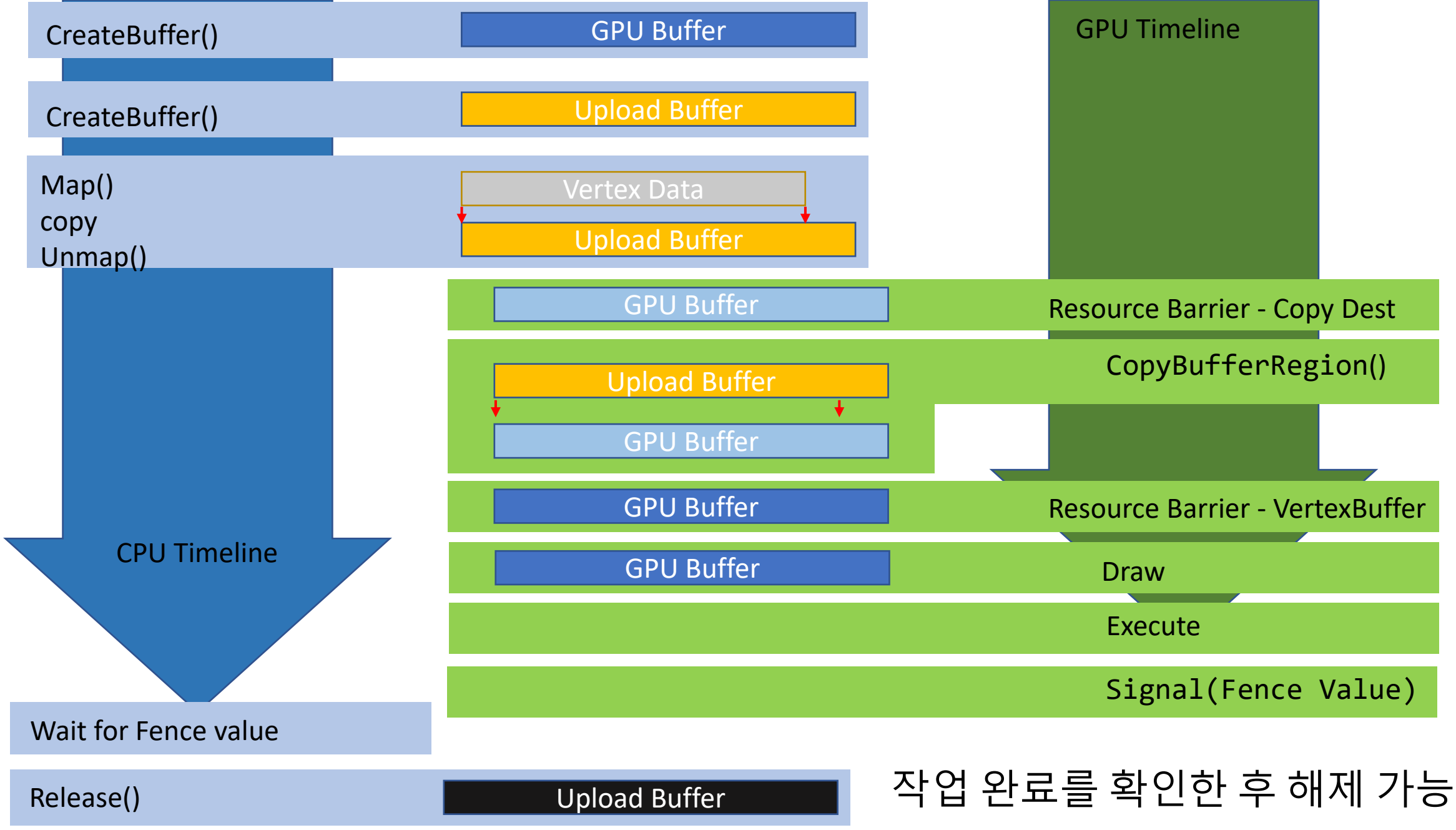


D3D12 Create – Update - Draw

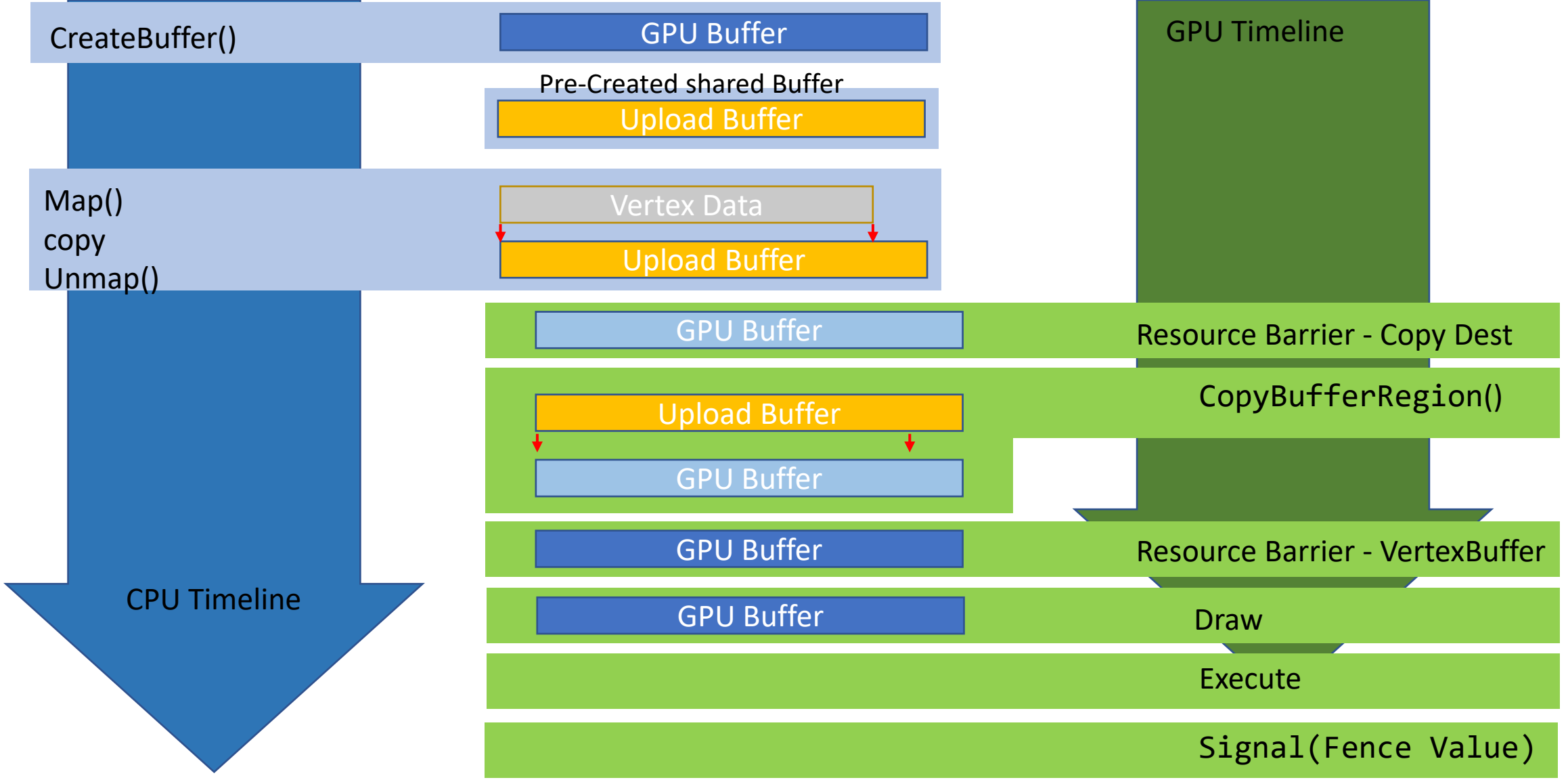
- CreateBuffer() – GPU Buffer
- CreateBuffer() – UploadBuffer
- ResourceBarrier() – Copy Dest
- CopyResource (UploadBuffer -> GPU Buffer)
- ResourceBarrier() – Vertex Buffer
- Draw()
- Execute()



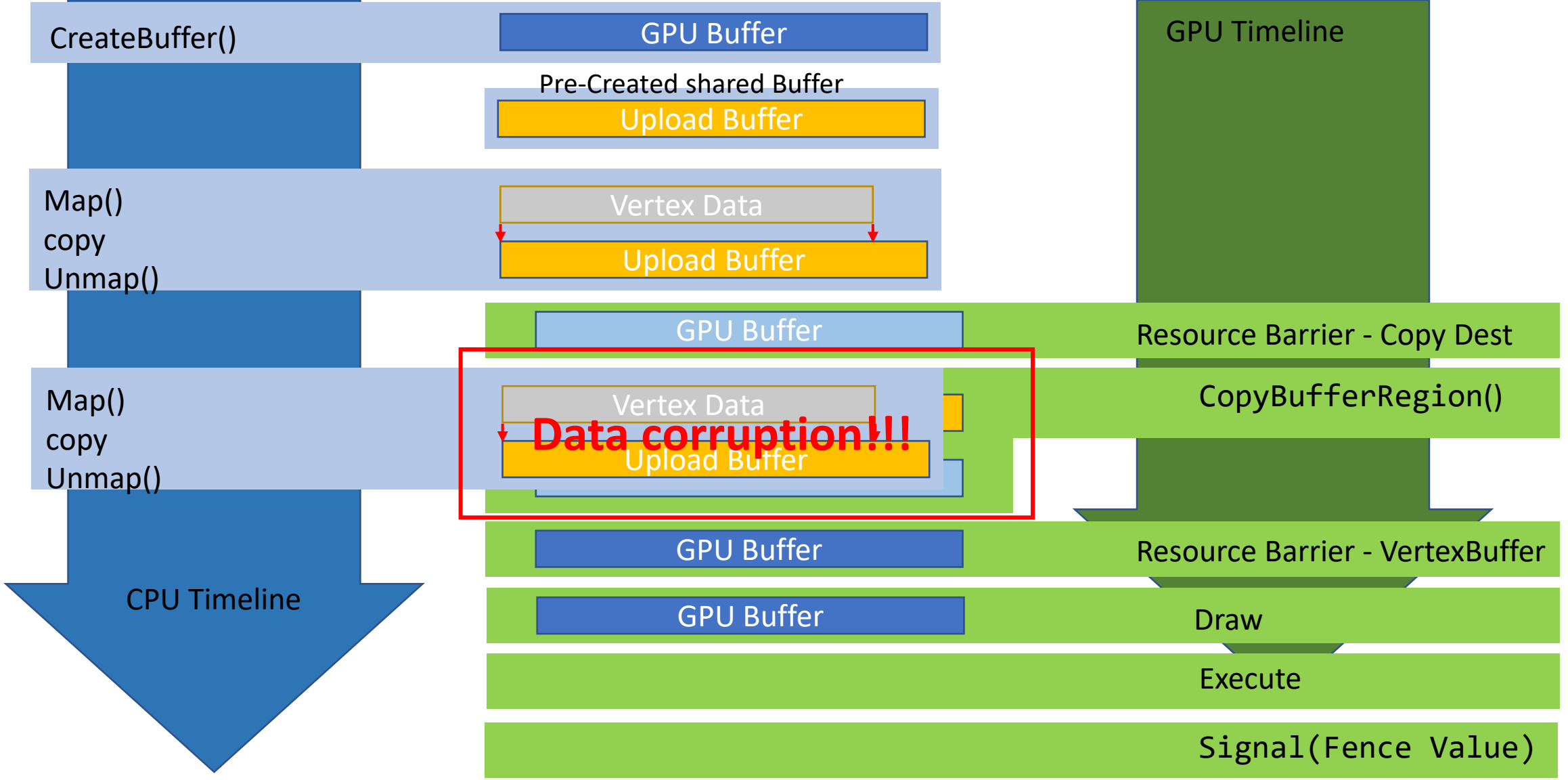
Upload Buffer는 언제 해제할까?



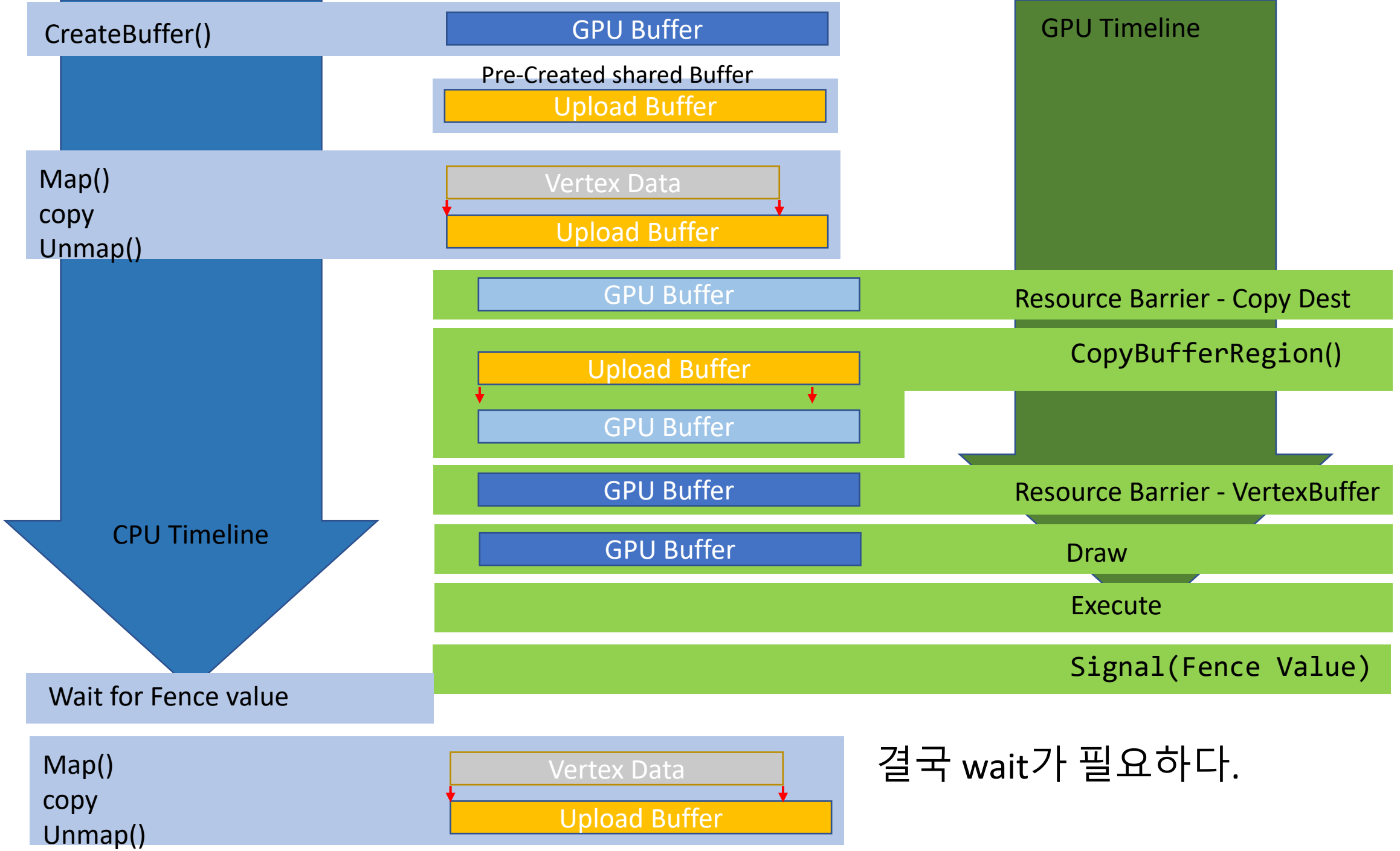
작업 완료를 확인한 후 해제 가능



미리 할당해둔 shared Upload Buffer를 사용하면 runtime에 할당/해제할 필요가 없잖아? Wait for fence도 필요없지 않을까?



GPU타임라인의 작업이 아직 완료되지 않았다면 데이터가 오염된다.



결국 wait가 필요하다.

D3D12 Create – Update - Draw

- CreateBuffer() – GPU Buffer
- CreateBuffer() – UploadBuffer
- ResourceBarrier() – Copy Dest
- CopyResource
- ResourceBarrier() – Vertex Buffer
- Draw()
- Execute()
- Signal(Fence Value)
- Wait for Fence value

Create-copy –execute-wait의 문제점

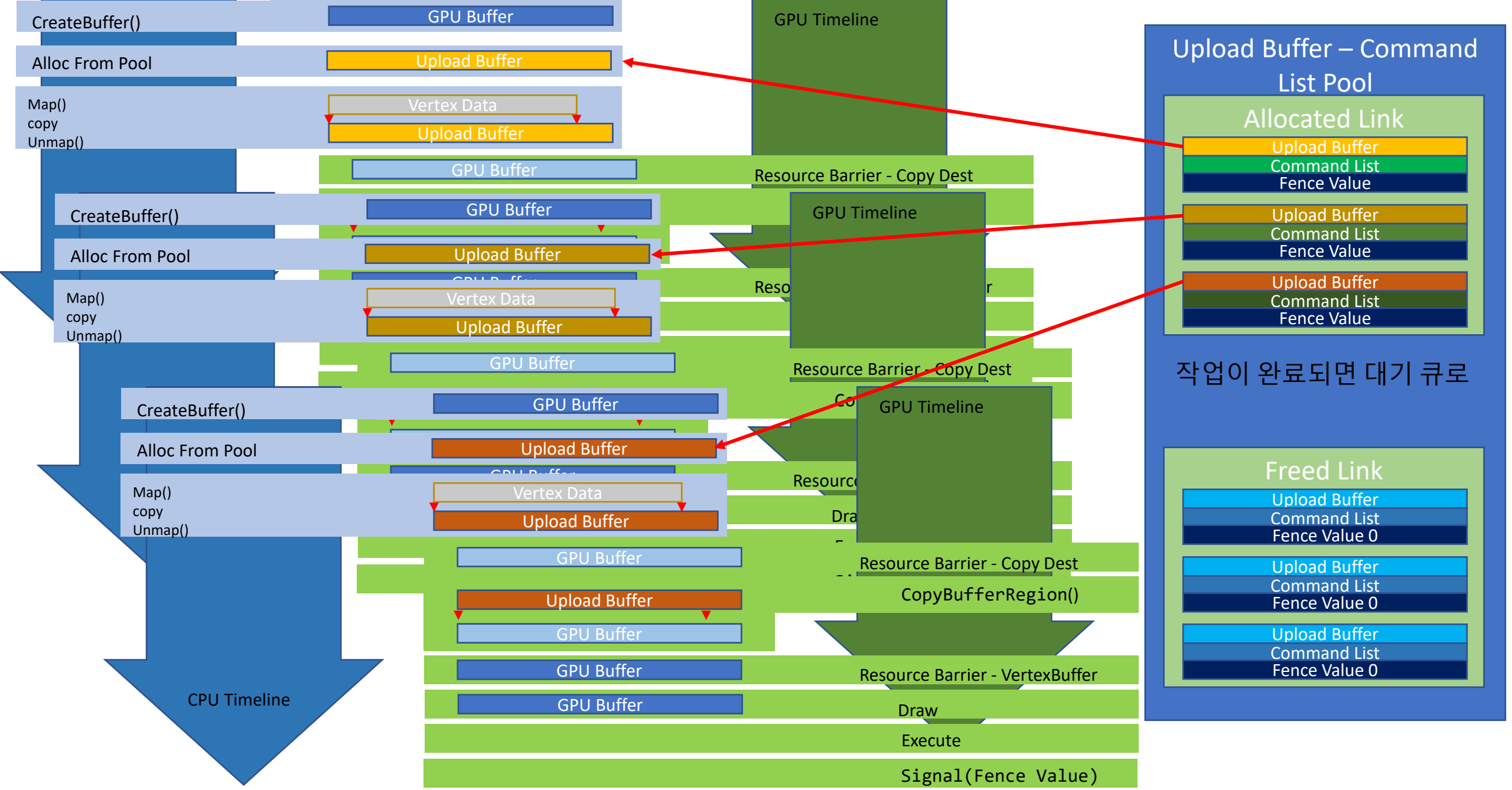
- GPU Buffer의 내용을 갱신하기 위해서는 Upload Buffer가 반드시 필요하다.
- Copy작업이 끝나기 전에 Upload Buffer가 갱신되거나 삭제되면 hazard가 발생한다.
- Execute하고나서 작업이 끝나기를 wait할때 크게 성능이 저하된다(GPU가 엄청나게 논다)
- Upload Buffer를 매번 할당/해제 할때마다 CPU자원을 낭비한다.

Copy 작업중 데이터 오염 방지 대책

- Copy 작업을 수행할때 Upload Buffer와 Command List를 할당한다.
- Command List와 Upload Buffer로 copy작업을 수행(execute)한 후 fence value를 설정한다. Command List, Upload Buffer, fence value를 어딘가에 저장해두고 주기적으로 작업이 완료됐는지를
- 검사한다.
- 작업이 완료됐으면 Command List와 Upload Buffer를 해제한다.

할당 해제 비용 줄이기

- Upload Buffer와 Command List를 묶어서 Pool로부터 할당받는다.
- Copy 작업을 수행할때 Upload Buffer와 Command List를 pool로부터 할당한다.
- Command List와 Upload Buffer로 copy작업을 수행(execute)한 후 fence value를 설정한다.
- Command List, Upload Buffer, fence value를 어딘가에 저장해두고 주기적으로 작업이 완료됐는지를 검사한다.
- 작업이 완료됐으면 Command List와 Upload Buffer를 pool에 반환한다.


CreateBuffer()
GPU Buffer
Alloc From Pool
Upload Buffer
Map()
copy
Unmap()
Vertex Data
Upload Buffer
GPU Timeline
GPU Buffer
Resource Barrier - Copy Dest
CreateBuffer()
GPU Buffer
Alloc From Pool
Upload Buffer
Map()
copy
Unmap()
Vertex Data
Upload Buffer
GPU Timeline
GPU Buffer
Resource Barrier - Copy Dest
CreateBuffer()
GPU Buffer
Alloc From Pool
Upload Buffer
Map()
copy
Unmap()
Vertex Data
Upload Buffer
GPU Timeline
GPU Buffer
Upload Buffer
GPU Buffer
Resource Barrier - Copy Dest
CopyBufferRegion()
GPU Buffer
GPU Buffer
Resource Barrier - VertexBuffer
Draw
Execute
Signal(Fence Value)
CPU Timeline
Upload Buffer – Command List Pool
Allocated Link
Upload Buffer
Command List
Fence Value
Upload Buffer
Command List
Fence Value
Upload Buffer
Command List
Fence Value
작업이 완료되면 대기 큐로
Freed Link
Upload Buffer
Command List
Fence Value 0
Upload Buffer
Command List
Fence Value 0
Upload Buffer
Command List
Fence Value 0

중첩 렌더링 중 GPU버퍼의 업데이트

- 하나의 Command Queue를 사용한다고 하면
- GPU버퍼의 갱신은 Resource Barrier로 동기화 가능
- 그림

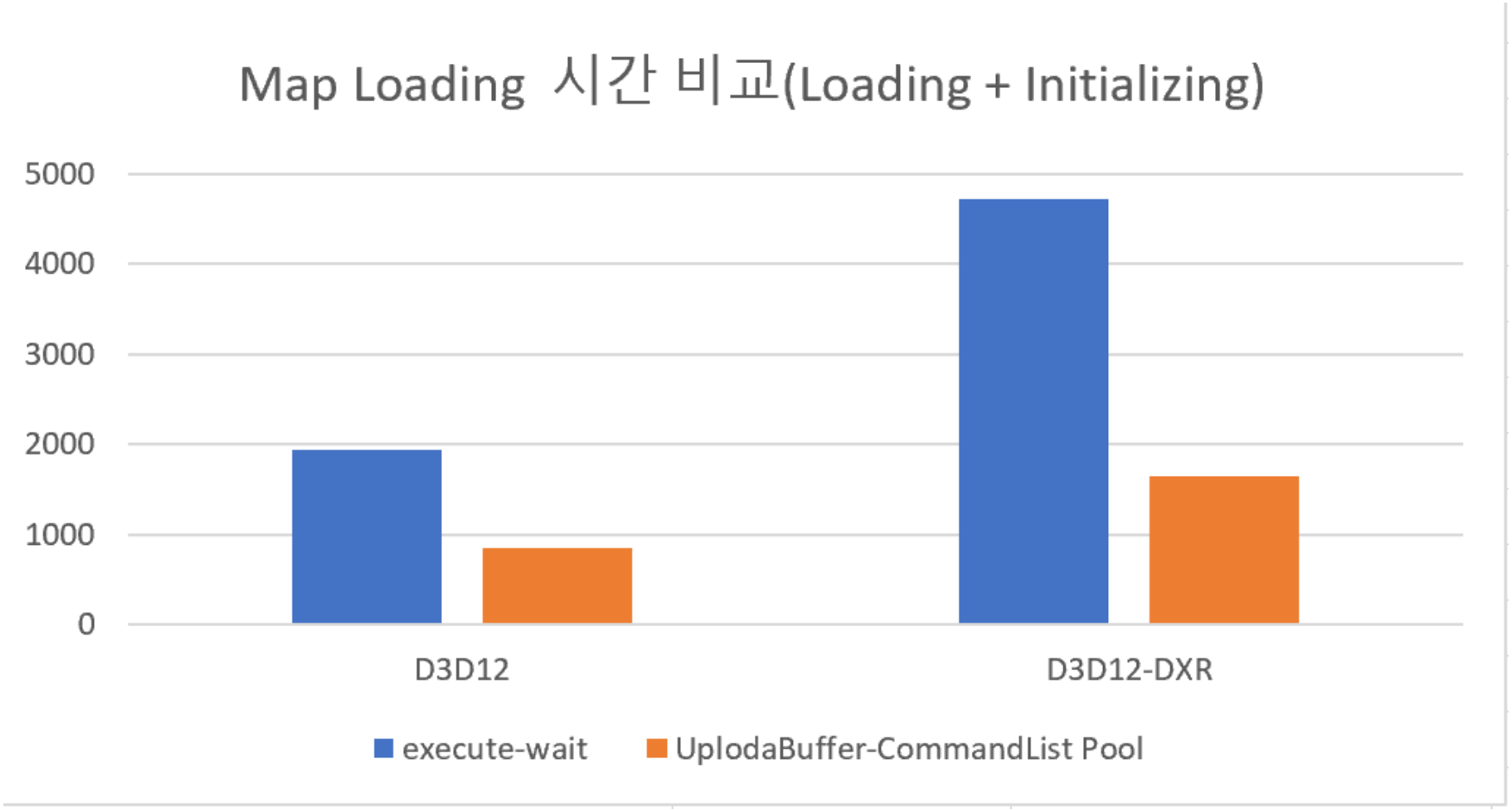
Resource Barrier

Command Queue

- ResourceBarrier는 각 작업간 수행순서를 보장한다.
- Dispatch#1작업이 완료되기 전에 Dispatch#2작업이 수행되지 않는다.
- Dispatch#2작업이 완료되기 전에 Copy작업이 수행되지 않는다.
- Copy작업이 완료되기 전에 Draw작업이 수행되지 않는다.

Map Loading(Loading + Initializing)성능 비교

	D3D12	D3D12-DXR
execute-wait	1938.77	4715.36
UplodaBuffer-CommandList Pool	849.68	1645.56



중첩 렌더링중 D3D Resource의 해제

- 신경 끈다.
  - 어플리케이션 또는 드라이버의 크래시.
- Command Queue의 모든 작업이 완료되었는지 확인 후 해제
  - 성능 저하
- Draw / compute 수행 전 임의의 자료구조에 리소스들을 등록, 일괄적으로 AddRef()호출 , Draw/Compute 작업 완료시마다 등록된 리소스들에 대해서 Release()호출.
  - 현실적으로 정상 작동하기 어려움.
  - D3DResource로부 파생되는 핸들들은 ref count와 무관함 - SRV,UAV등은 ref count와 무관함.

중첩 렌더링중 D3D Resource의 해제

- 중첩 렌더링 프레임 수만큼 지연시켜서 Release()호출
- IUnknown::Release()를 대신 호출해주는 매니저를 만든다. 이하 DefRelease()로 명명
- pTex->Release() 대신 g_pDefReleaseManager->DefRelease(pTex);로
- DefRelease()호출시 현재 프레임 카운트를 함께 저장.
- 중첩된 렌더링 프레임이 2프레임이라면 DefRelease매니저에서 주기적으로 등록된 IUnknown 객체들의 삭제 당시 프레임 카운트로부터 2프레임이 지난 시점에서 IUknown::Release()를 호출

결론

- 그래도 D3D11보다 한참 느리다-_-;
- 하지만 DXR때문에 12를 피해갈 수 없다.
- 최선을 다해서 최적화하자.....ㅠㅠ