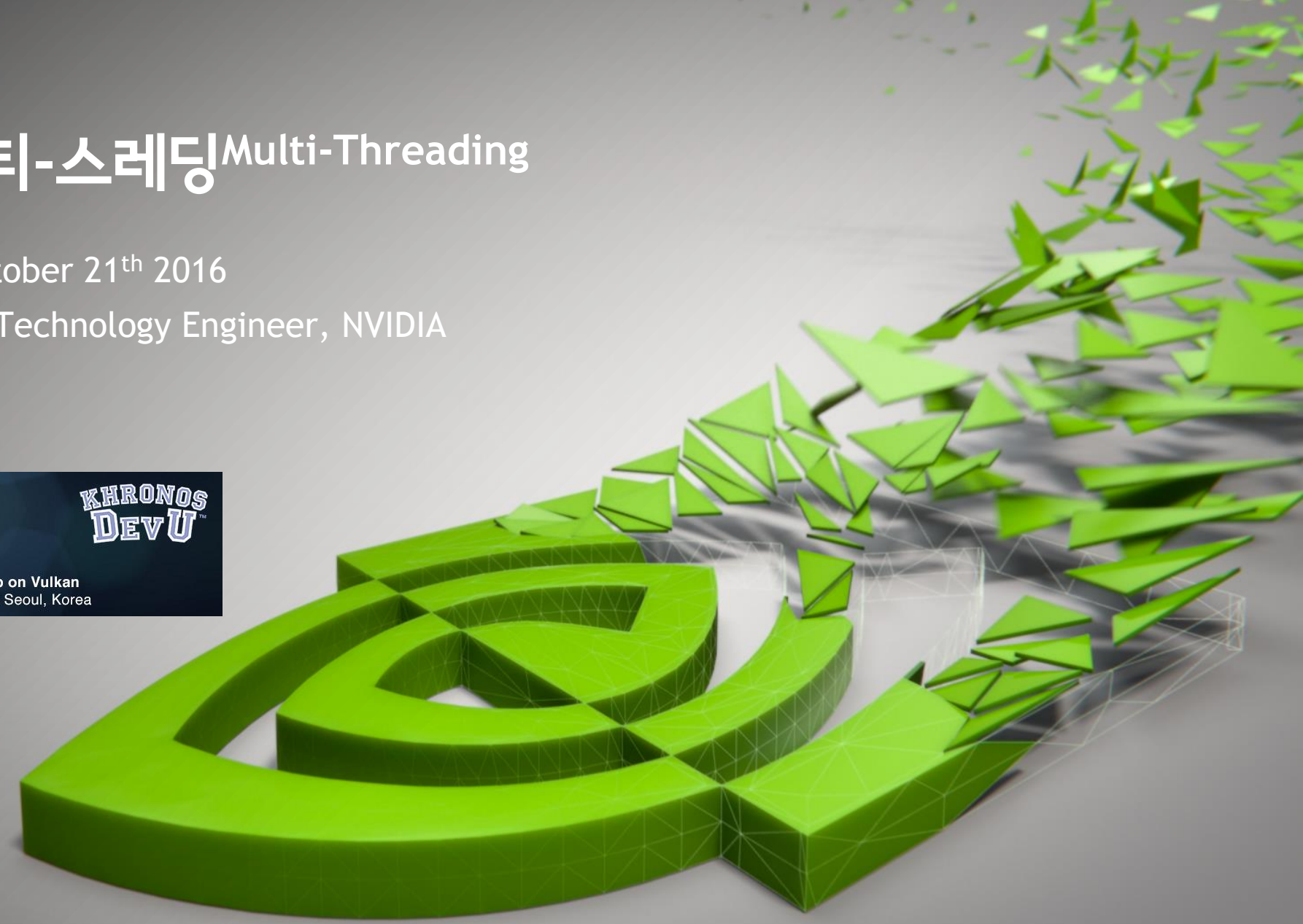
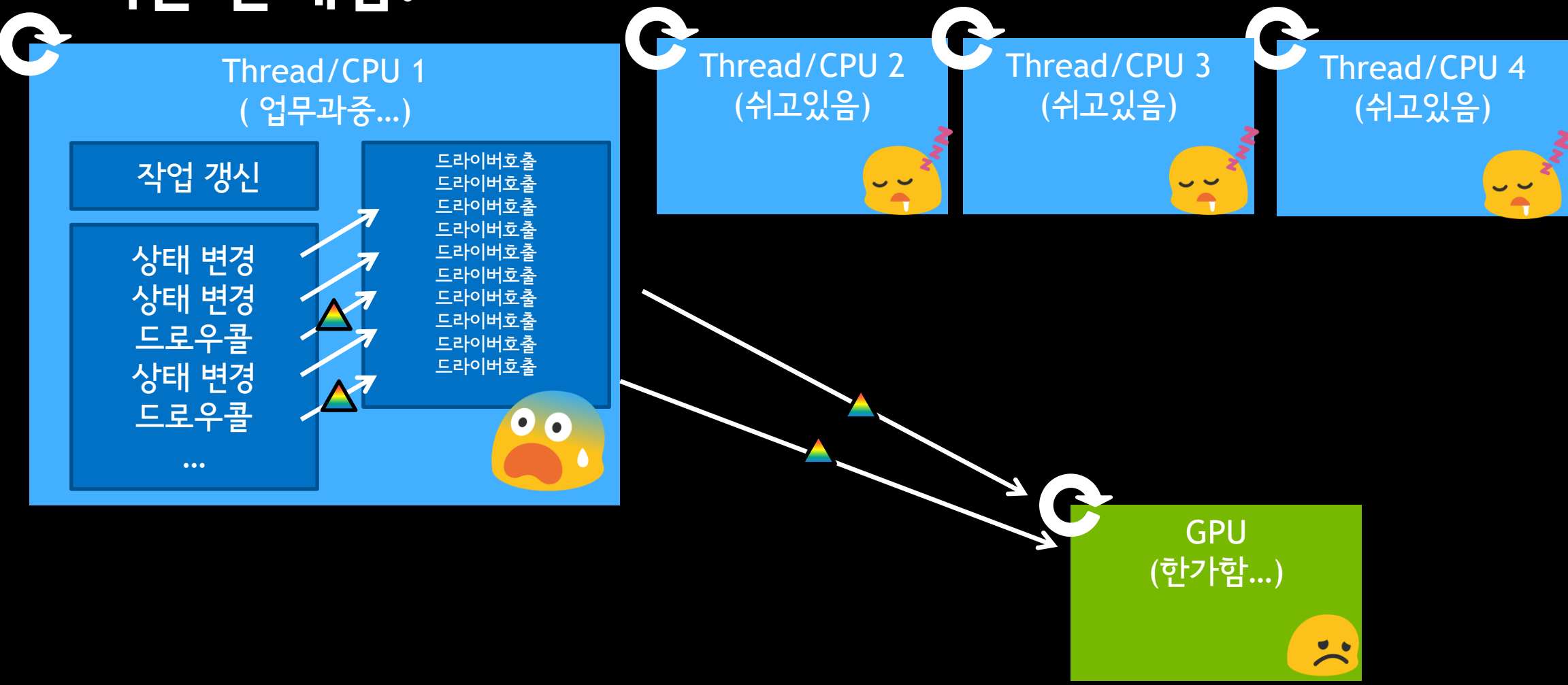
Vulkan 멀티-스레딩Multi-Threading

Khronos DevU, October 21th 2016

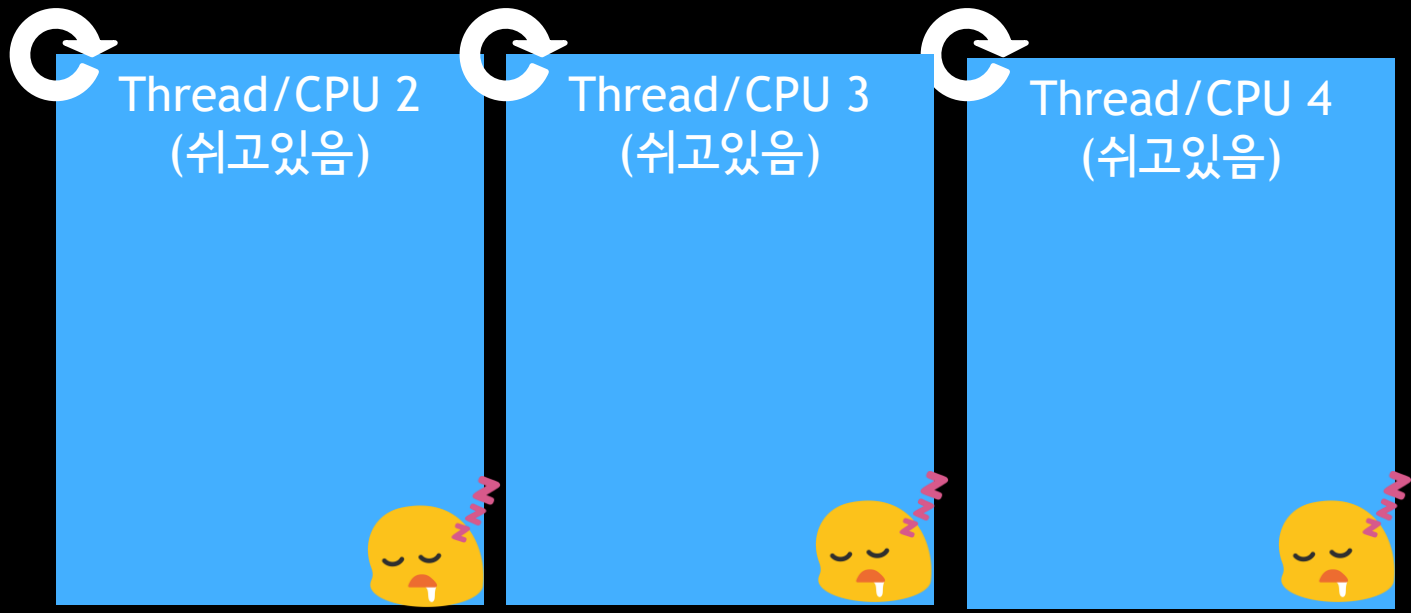
최지호, Developer Technology Engineer, NVIDIA



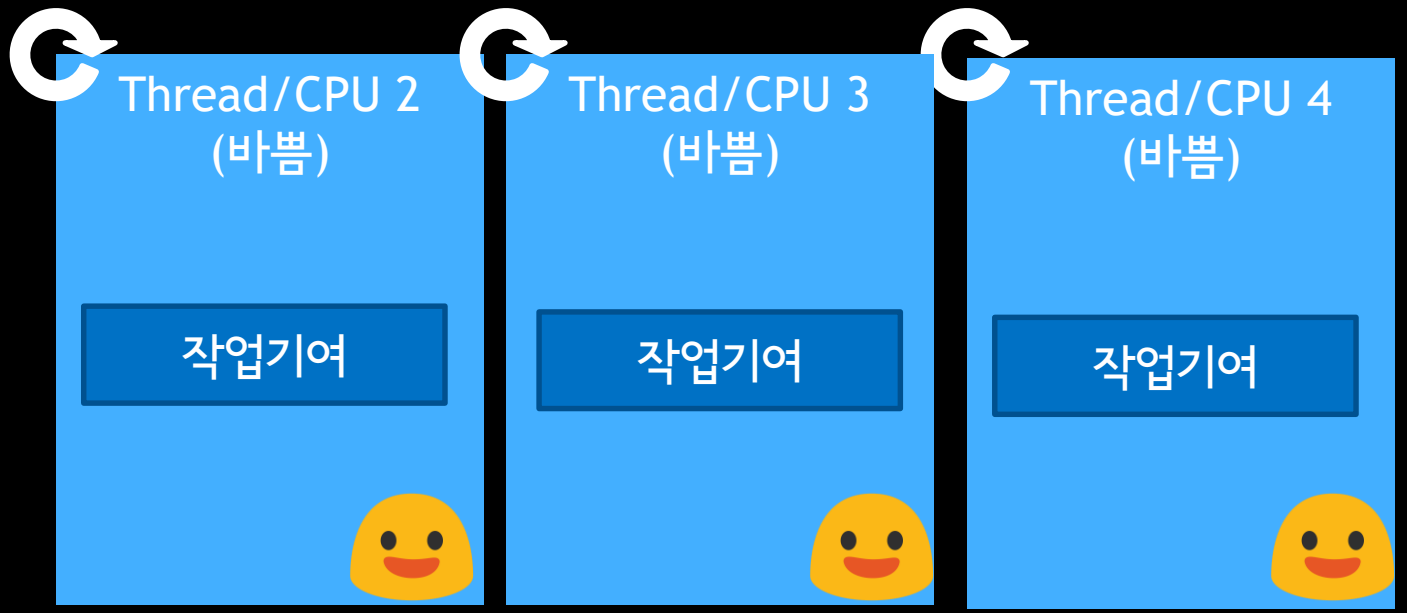
기존 문제점?


Thread/CPU 1
(업무과중...)
작업 갱신
상태 변경
상태 변경
드로우콜
상태 변경
드로우콜
...
드라이버호출
드라이버호출
드라이버호출
드라이버호출
드라이버호출
드라이버호출
드라이버호출
드라이버호출
드라이버호출
드라이버호출
Thread/CPU 2
(쉬고있음)
Thread/CPU 3
(쉬고있음)
Thread/CPU 4
(쉬고있음)
GPU
(한가함...)

개발자들은 스레드 친화적인 API를 원한다!



개발자들은 스레드 친화적인 API를 원한다!



Vulkan 철학: 명시적 스레드 지원

- Vulkan 은 처음부터 스레드 친화적으로 설계
  - 스펙에 스레드 안정성과 함수 호출 결과에 대해 자세하게 명시
  - 하지만 책임은 모두 앱에 있음 – 결과적으로는 장점
- 앱 레벨의 스레딩이 인기를 얻어가는 추세
  - 앱은 멀티 스레드 상에서 렌더링 작업을 생성하기를 원함
  - 유효성 확인 비용 및 서밋 비용에 대한 부담을 다중 스레드로 분할
  - 드라이버보다 상위 레벨에서 오브젝트/엑세스 동기화를 앱이 직접 처리

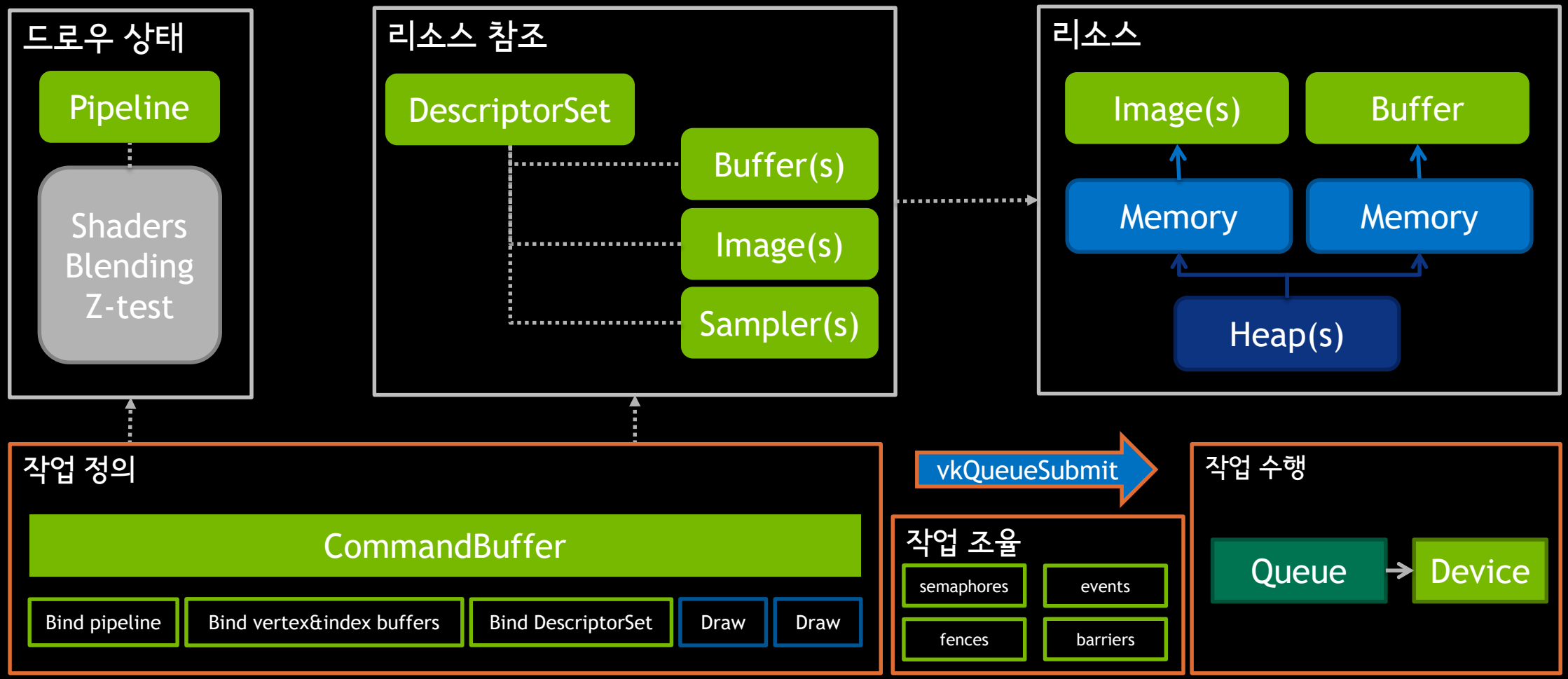
Vulkan에서 권장하는 스레딩 사례

- 리소스 (버퍼) 를 다중 스레드 상에서 갱신
  - CPU 버텍스 데이터나 인스턴스 데이터 애니메이션(예, 모핑)
  - CPU 유니폼 버퍼 데이터 갱신(예, 변환 행렬 갱신)

- 파이프라인 상태 병렬 생성
  - 쉐이더 컴파일 및 상태 확인

- 스레드 렌더링 / 드로우콜
  - 다중 스레드 상에서 커맨드 버퍼를 생성

작업 정의와 서밋을 분리!


드로우 상태
Pipeline
Shaders
Blending
Z-test
리소스 참조
DescriptorSet
Buffer(s)
Image(s)
Sampler(s)
리소스
Image(s)
Buffer
Memory
Memory
Heap(s)
작업 정의
CommandBuffer
Bind pipeline
Bind vertex&index buffers
Bind DescriptorSet
Draw
Draw
vkQueueSubmit
작업 조율
semaphores
events
fences
barriers
작업 수행
Queue
Device

작업 정의: 커맨드 버퍼

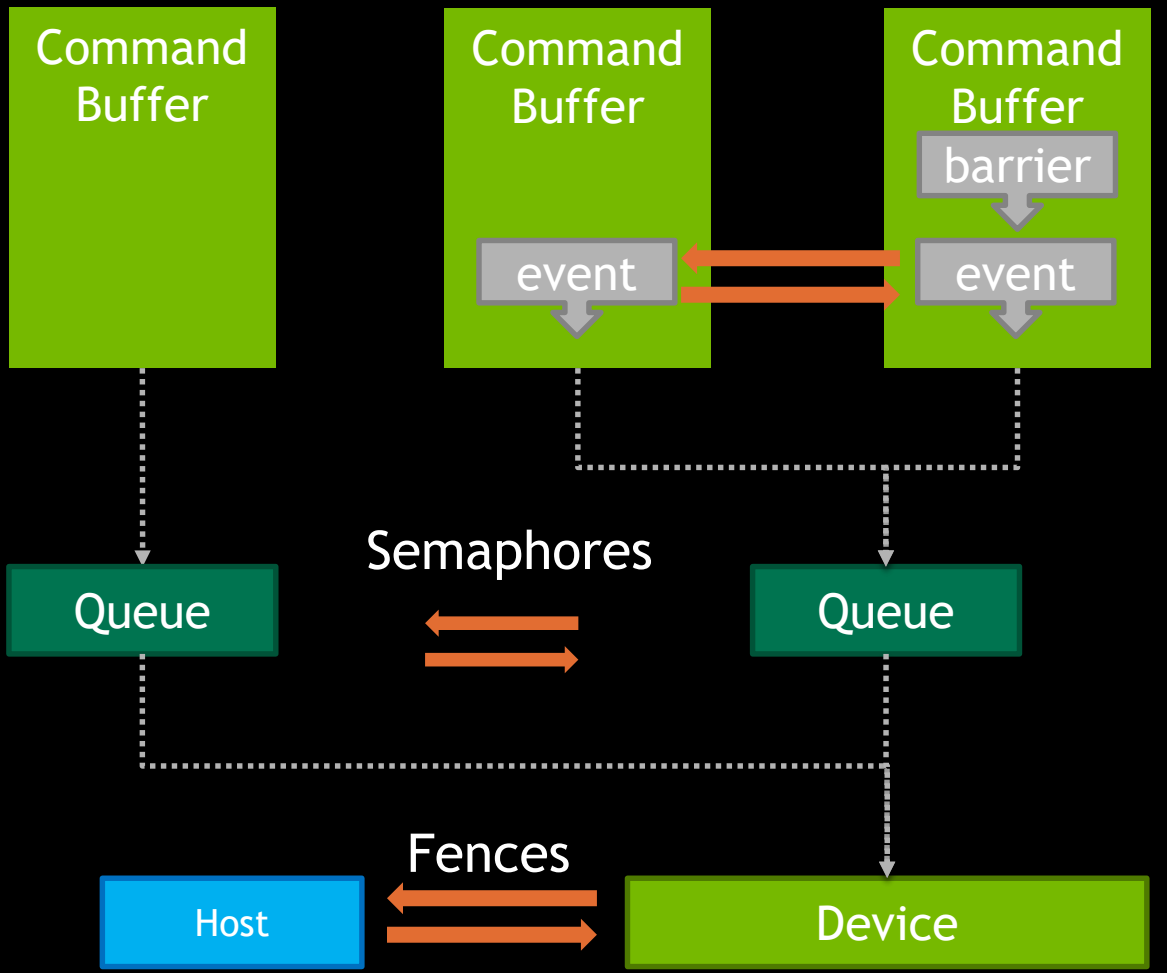
- 모든 Vulkan렌더링은 커맨드 버퍼를 통해서 이루어짐
- 일회용이거나, 반복 서밋도 가능
  - 그에 따라 드라이버가 버퍼를 최적화할 수 있음
- 프라이머리 & 세컨더리 커맨드 버퍼 Primary & Secondary Command buffers
  - 정적인 작업은 재사용할 수 있도록
- *중요: 커맨드 버퍼간에 상태는 전혀 유지되지 않는다!*

작업 수행 : 큐Queue

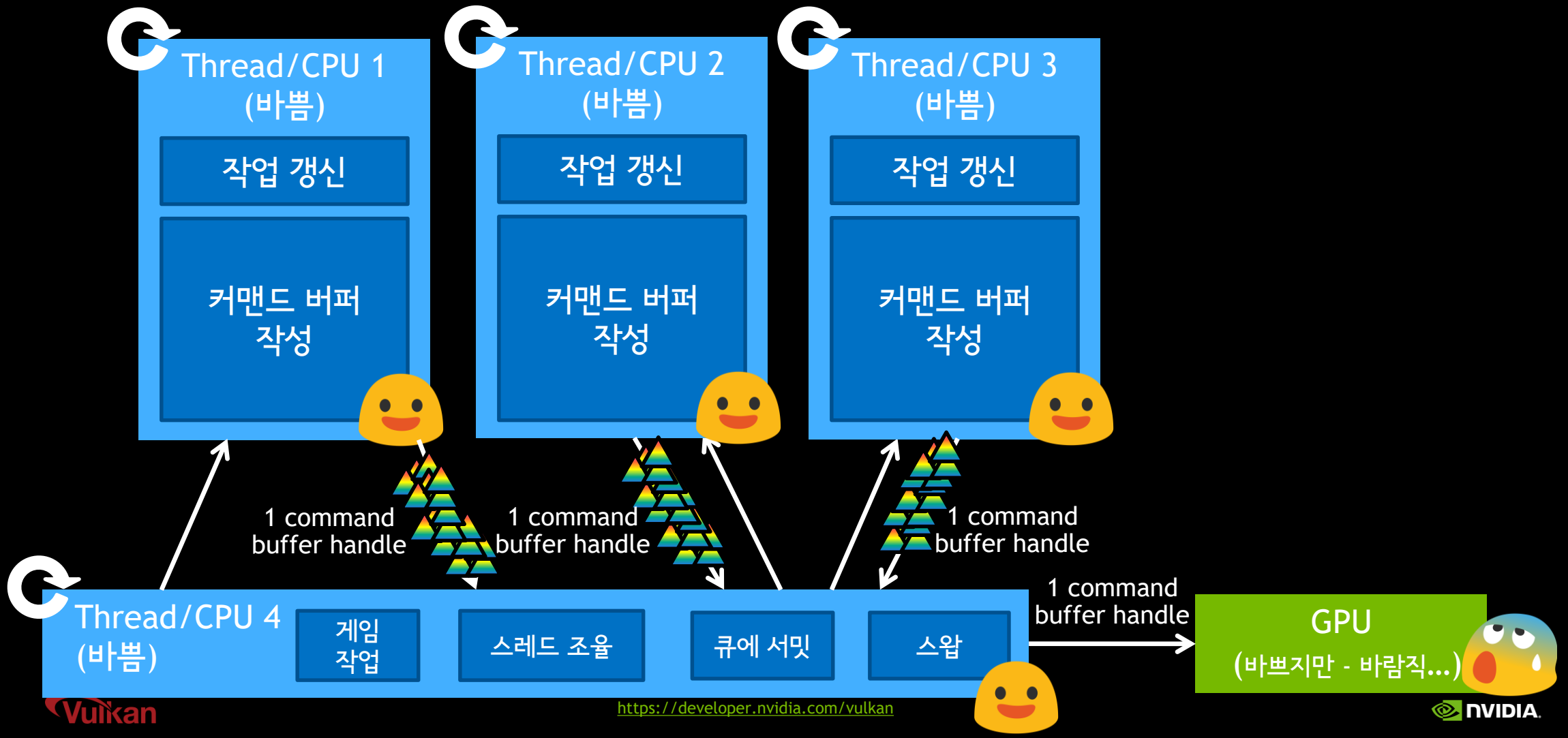
- 명시적인 커맨드 큐; GL에서는 암묵적으로 컨텍스트내 존재
  - 작업을 서밋할 때 " 컨텍스트 바인딩 " 할 필요가 없음
  - 여러 스레드들이 서로 다른 큐에 서밋할 수 있음
- 커맨드 버퍼를 서밋하면 큐가 GPU작업을 받음
  - 큐가 하는 작업은 매우 단순: 기본적으로, “작업을 서밋” 하거나 “대기wait for idle”
- 큐 작업 서밋에 포함되는 기본 동기화 작업:
  - *Wait* : 서밋된 작업을 수행하기 전에 대기
  - *Signal* : 서밋된 작업이 완료되면 시그널
- 큐 “패밀리 " 로 다른 타입의 작업들을 구분하여 접수
    - 큐마다 작업 형태 지정 (예. DMA/메모리 전송 전용 큐)

작업 조율: 동기화

- **세마포** semaphores
  - 큐간 작업 동기화, 또는 단일 큐 내 작지 않은 서밋 동기화를 위해 사용
- **이벤트** events and **베리어** barriers
  - 커맨드 버퍼 내 작업 동기화, 또는 같은 큐에 서밋된 커맨드 버퍼들간 동기화를 위해 사용
- **펜스** fences
  - 디바이스와 호스트간 작업 동기화를 위해 사용

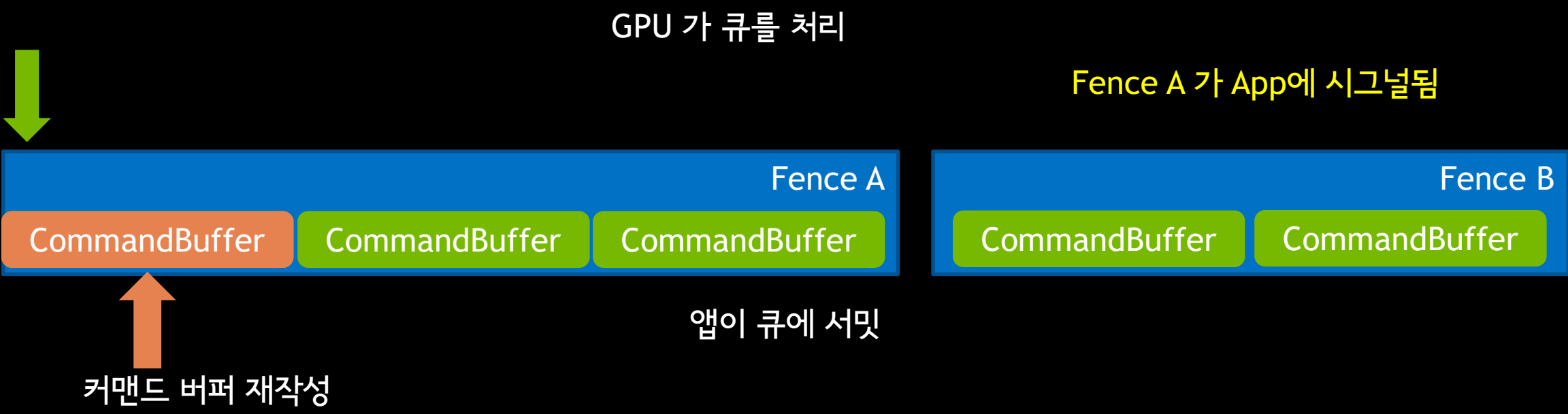


스레드 별 커맨드 버퍼 생성



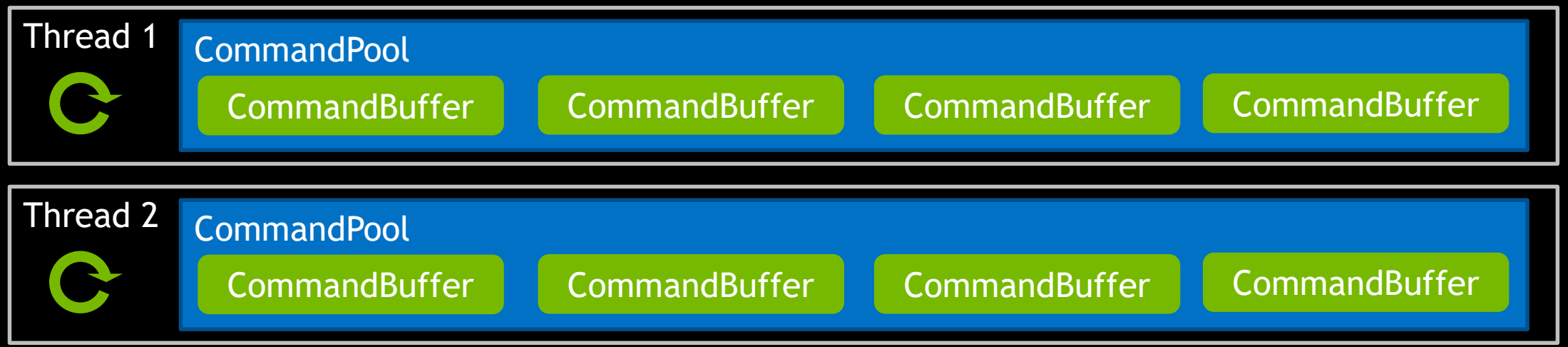
커맨드 버퍼 스레드 안정성

- 커맨드 버퍼가 수행중in flight일 때 재사용하면 안됨
- 그렇다고, 매 프레임 큐를 플러쉬할 건 아님
- 큐 서밋시 지정한 VkFences 를 통해 커맨드 버퍼의 재사용가능 여부를 판별할 수 있음



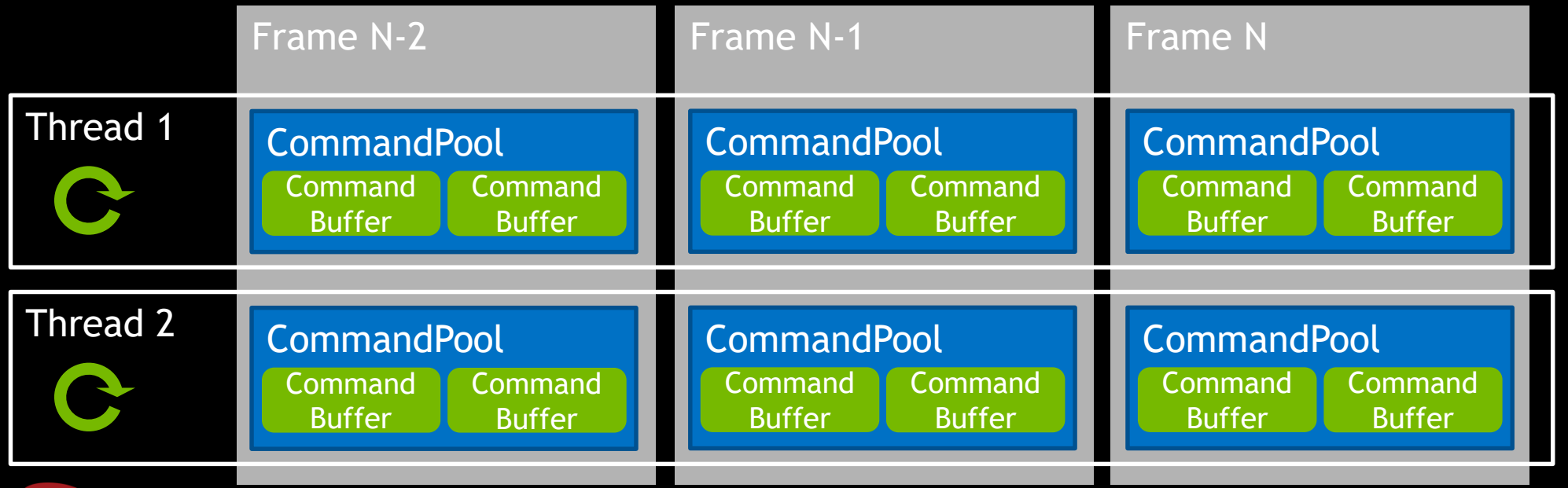
Vulkan 스레드: 커맨드 풀Command Pools

- VkCommandPool 오브젝트가 다중 스레드 커맨드 생성의 핵심
- VkCommandBuffers 는 “부모인 " VkCommandPool에 의해 할당
- 서로 다른 스레드에서 채워지는 VkCommandBuffers 는 서로 다른 풀에서 만들자
  - 그렇지 않으면, 버퍼와 풀 모두 명시적으로 동기화되어야 함, 비효율적



Vulkan 스레드: 커맨드 풀Command Pools

- 한 스레드에 여러 커맨드 버퍼를 사용할 수 있음
  - 사용중이 아닐 때에만 커맨드 버퍼를 재사용할 수 있음
- 스레드들은 여러 개의, 프레임 당 독립적인 버퍼들을 가질 수 있음
- 스레드/프레임이 사용중이 아니라면, 풀을 리셋하는게 더 빠름



Vulkan 스레드: 디스크립터 풀Descriptor Pools

- 스레드별 오브젝트 상태 생성 시 VkDescriptorPool 필요
  - 예) 스레드별로 동적 생성되는 렌더링 오브젝트들의 경우
- 풀은 여러 타입의 VkDescriptorSet를 담을 수 있음
  - 예) 샘플러, 유니폼 버퍼 등
  - 각 타입의 최대 개수는 풀 생성시 지정
- VkDescriptorSets 는 부모인 VkDescriptorPool 로부터 할당
  - 다른 스레드에서 할당된 디스크립터들은 다른 풀을 사용해야 함
- 하지만, 동일한 풀로부터 만든 VkDescriptorSets 은 다른 스레드에서도 쓰기가 가능

Vulkan 멀티-스레딩

감사합니다!

Reference

- https://developer.nvidia.com/sites/default/files/akamai/gameworks/blog/munich/mschott_vulkan_multi_threading.pdf