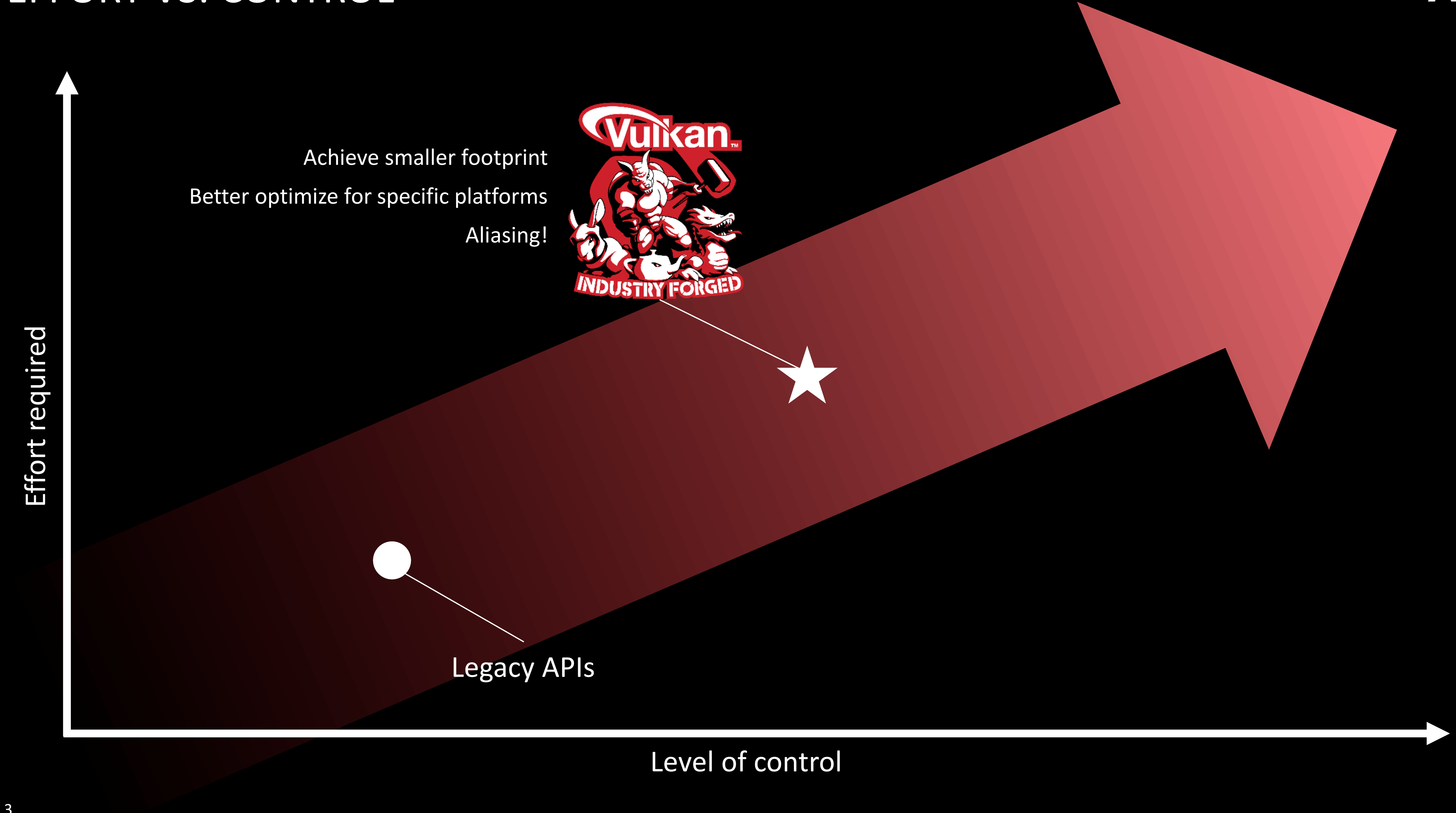
VULKAN MEMORY MANAGEMENT

STEVEN TOVEY
DEVELOPER TECHNOLOGY GROUP

AGENDA

- Heaps & Types
- Tips & Tricks
- The VMA Library
- Conclusion

EFFORT VS. CONTROL



Achieve smaller footprint

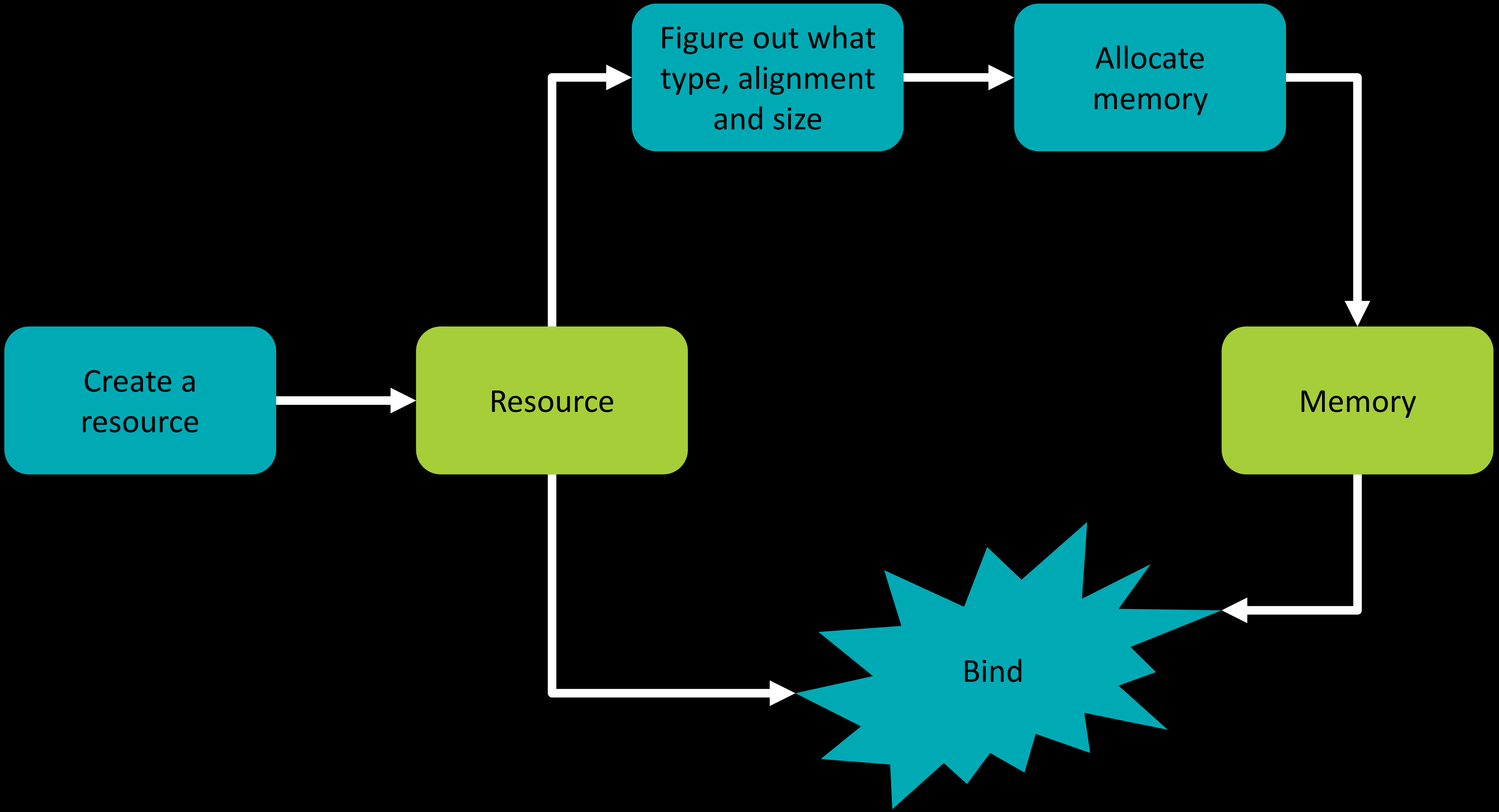
Better optimize for specific platforms

Aliasing!

Legacy APIs

Heaps & Types

YOUR MISSION, JIM…



ALLOCATING SOME MEMORY

```
VkResult vkAllocateMemory(
    VkDevice                        device,
    const VkMemoryAllocateInfo*     pAllocateInfo,
    const VkAllocationCallbacks*    pAllocator,
    VkDeviceMemory*                 pMemory);
```

```
typedef struct VkMemoryAllocateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceSize       allocationSize;
    uint32_t           memoryTypeIndex;
} VkMemoryAllocateInfo;
```

ALLOCATING SOME MEMORY

```
VkResult vkAllocateMemory(
    VkDevice                      device,
    const VkMemoryAllocateInfo*   pAllocateInfo,
    const VkAllocationCallbacks*  pAllocator,
    VkDeviceMemory*               pMemory);
```

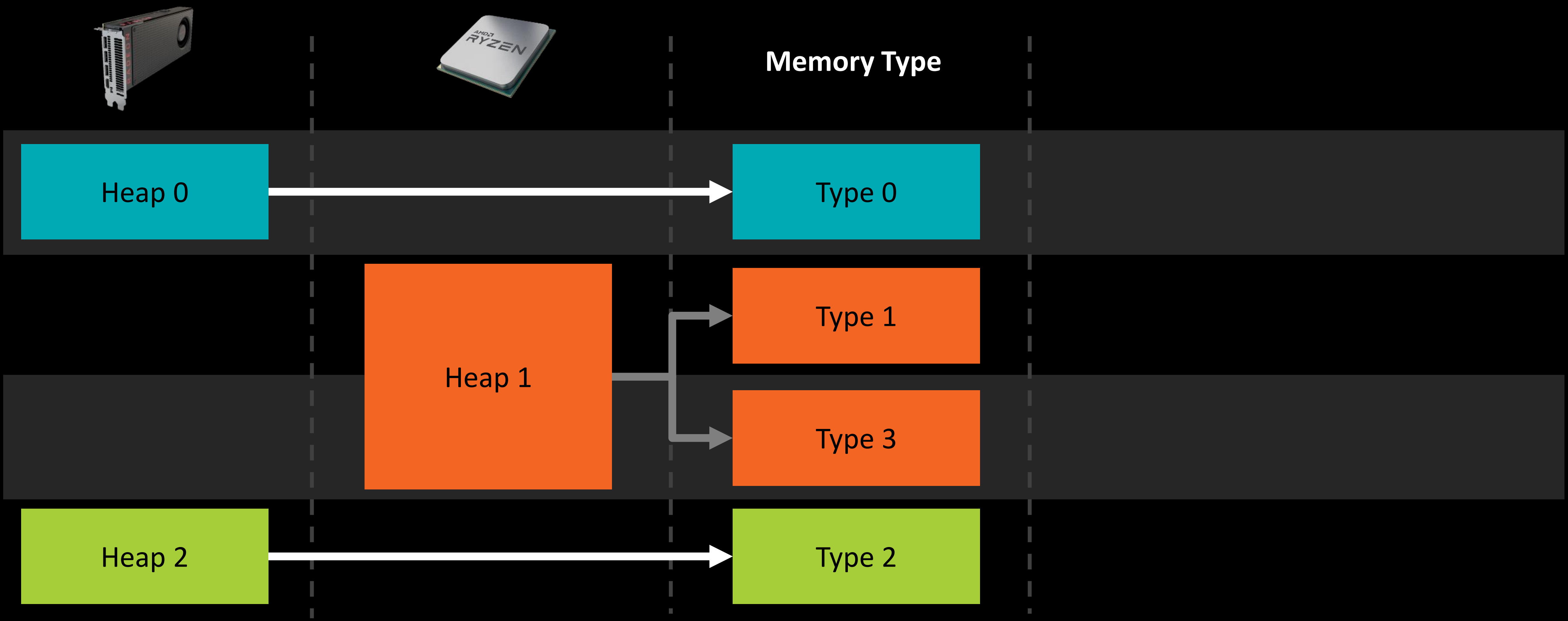
```
typedef struct VkMemoryAllocateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceSize       allocationSize;
    uint32_t           memoryTypeIndex;
} VkMemoryAllocateInfo;
```

```
vkGetPhysicalDeviceMemoryProperties(
    VkPhysicalDevice                    physicalDevice,
    VkPhysicalDeviceMemoryProperties*   pMemoryProperties);
```

```
typedef struct VkPhysicalDeviceMemoryProperties {
    uint32_t        memoryTypeCount;
    VkMemoryType    memoryTypes[VK_MAX_MEMORY_TYPES];
    uint32_t        memoryHeapCount;
    VkMemoryHeap    memoryHeaps[VK_MAX_MEMORY_HEAPS];
} VkPhysicalDeviceMemoryProperties;
```

MEMORY TYPES VS. HEAPS

(AMD RX VEGA 64)



Memory Type

Heap 0

Type 0

Heap 1

Type 1

Type 3

Heap 2

Type 2

MEMORY TYPES CHEAT SHEET

(AMD RX VEGA 64)

Memory Type	🛢	👁	$	🛢	👁	$	R	W	R	W	Size
0	✓	✓	✓	✗	✗	✗	🐇	🐇	✗	✗	Most of VRAM
1	✗	✓	✓	✓	✓	✗	🐢	🐢	🐢	🐇	
2	✓	✓	✓	✗	✓	✗	🐇	🐇	🐢	🐇	Fixed 256MiB
3	✗	✓	✓	✓	✓	✓	🐢	🐢	🐇	🐇	

🛢 - Storage 👁 - Visible $ - Cached

🐇 - Fast 🐢 - Slow

MEMORY TYPES CHEAT SHEET

(AMD RX VEGA 64)

Memory Type	🛢	👁	$	🛢	👁	$	R	W	R	W	Size
0	✓	[illegible]	✓	✗	✗	✗	🐇	🐇	✗	✗	Most of VRAM
1	✗	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	🐢	🐢	🐇	
2	✓	✓	✓	✗	✓	✗	🐇	🐇	🐢	🐇	Fixed 256MiB
3	✗	✓	✓	✓	✓	✓	🐢	🐢	🐇	🐇	

Maps to
VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT in
VkMemoryPropertyFlagBits.

🛢 - Storage 👁 - Visible $ - Cached

🐇 - Fast 🐢 - Slow

MEMORY TYPES CHEAT SHEET

(AMD RX VEGA 64)

Maps to
VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT and
VK_MEMORY_PROPERTY_CACHED_BIT respectively.

Memory Type	🛢	👁	$	🛢	👁	$	R	W	R	W	Size
0	✓	✓	✓	✗	✗	✗	🐇	🐇	✗	✗	Most of VRAM
1	✗	✓	✓	✓	✓	✗	🐢	🐢	🐢	🐇	
2	✓	✓	✓	✗	✓	✗	🐇	🐇	🐢	🐇	Fixed 256MiB
3	✗	✓	✓	✓	✓	✓	🐢	🐢	🐇	🐇	

🛢 - Storage 👁 - Visible $ - Cached

🐇 - Fast 🐢 - Slow

MEMORY TYPES CHEAT SHEET

(AMD RX VEGA 64)

Memory Type	🛢	👁	$	🛢	👁	$	R	W	R	W	Size
0	✓	✓	✓	✗	✗	✗	🐇	🐇	✗	✗	Most of VRAM
1	✗	✓	✓	✓	✓	✗	🐢	🐢	🐢	🐇	
2	✓	✓	✓	✗	✓	✗	🐇	🐇	🐢	🐇	Fixed 256MiB
3	✗	✓	✓	✓	✓	✓	🐢	🐢	🐇	🐇	

🛢 - Storage 👁 - Visible $ - Cached

🐇 - Fast 🐢 - Slow

MEMORY TYPES CHEAT SHEET

(AMD RX VEGA 64)

Memory Type	🛢	👁	$	🛢	👁	$	R	W	R	W	Size
0	✓	✓	✓	✗	✗	✗	🐇	🐇	✗	✗	Most of VRAM
1	✗	✓	✓	✓	✓	✗	🐢	🐢	🐢	🐇	
						✗	🐇	🐇	🐢	🐇	Fixed 256MiB
	✗	✓	✓	✓	✓	✓	🐢	🐢	🐇	🐇	

Okay, not *that* bad since we benefit from GPU caches, but certainly worse than just reading from `DEVICE_LOCAL`.

🛢 - Storage 👁 - Visible $ - Cached

🐇 - Fast 🐢 - Slow

MEMORY TYPES CHEAT SHEET

(AMD RX VEGA 64)

Memory Type	🛢	👁	$	🛢	👁	$				
0	✓	✓	✓	✗	✗	✗				
1	✗	✓	✓	✓	✓	✗	🐢	🐢	🐢	🐇
2	✓	✓	✓	✗	✓	✗	🐇	🐇	🐢	🐇
3	✗	✓	✓	✓	✓	✓				

Fixed 256MiB

On current GPUs & drivers, PC Windows® everything that is HOST_VISIBLE is also marked COHERENT.

On other architectures you may need:
vkInvalidateMappedMemoryRanges before reads
and
vkFlushMappedMemoryRanges after writes.

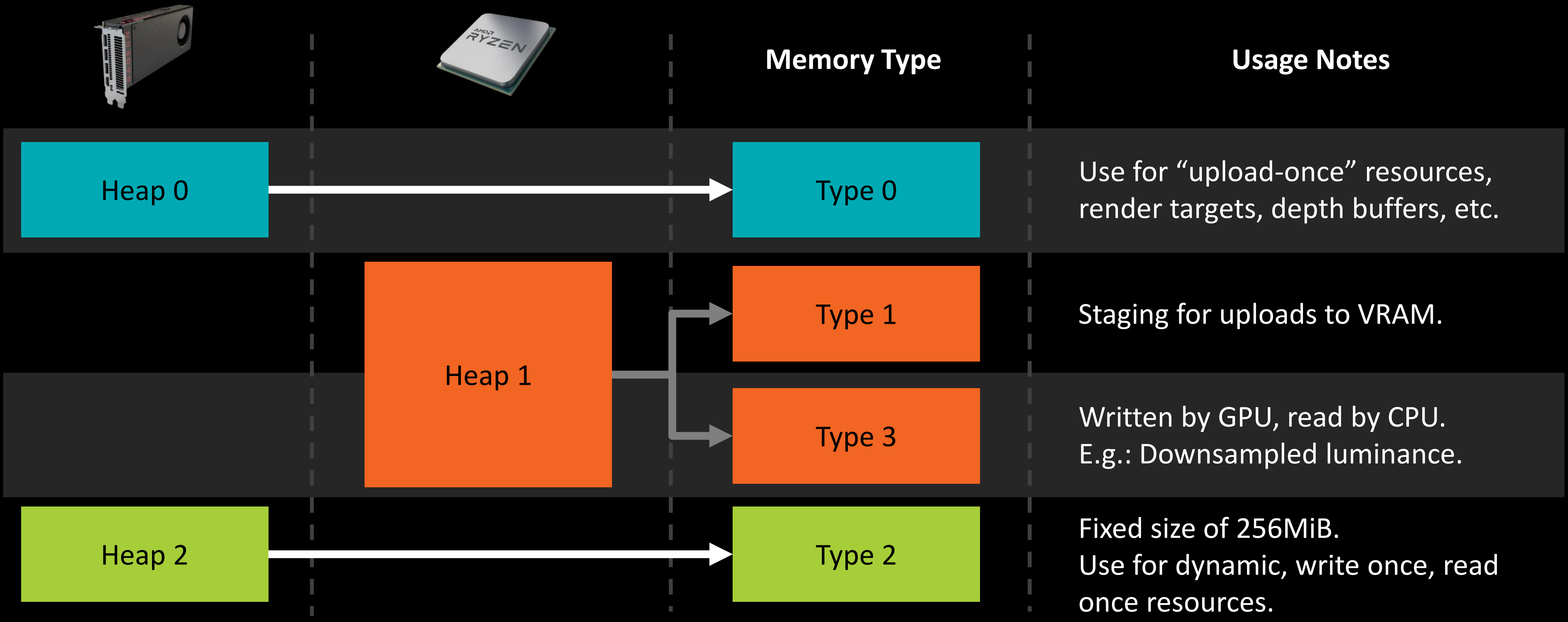
BEWARE: Unmapping **won't** do this for you!

🛢 - Storage 👁 - Visible $ - Cached

🐇 - Fast 🐢 - Slow

MEMORY TYPES VS. HEAPS

(AMD RX VEGA 64)



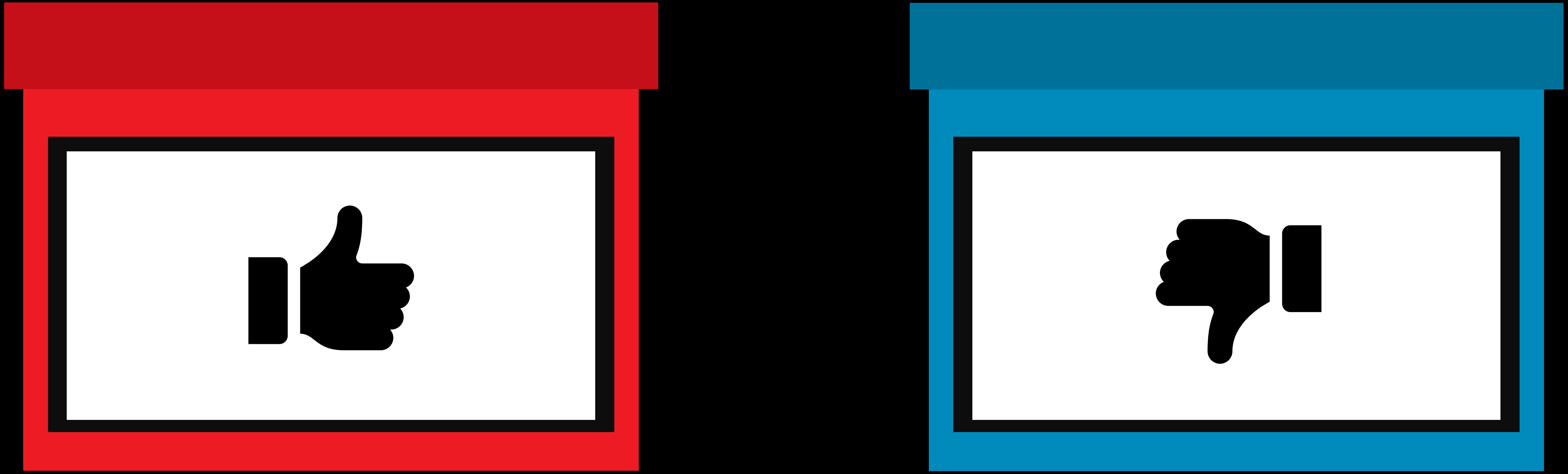
		Memory Type	Usage Notes
Heap 0		Type 0	Use for “upload-once” resources, render targets, depth buffers, etc.
	Heap 1	Type 1	Staging for uploads to VRAM.
		Type 3	Written by GPU, read by CPU. E.g.: Downsampled luminance.
Heap 2		Type 2	Fixed size of 256MiB. Use for dynamic, write once, read once resources.

WARNING!

- Not a good idea to hardcode the memory type indices.
  - Driver may change in future
  - May be different on other/newer hardware
  - Magic numbers
- Query the info using `vkGetPhysicalDeviceMemoryProperties`.
- Map to engine-specific enums.

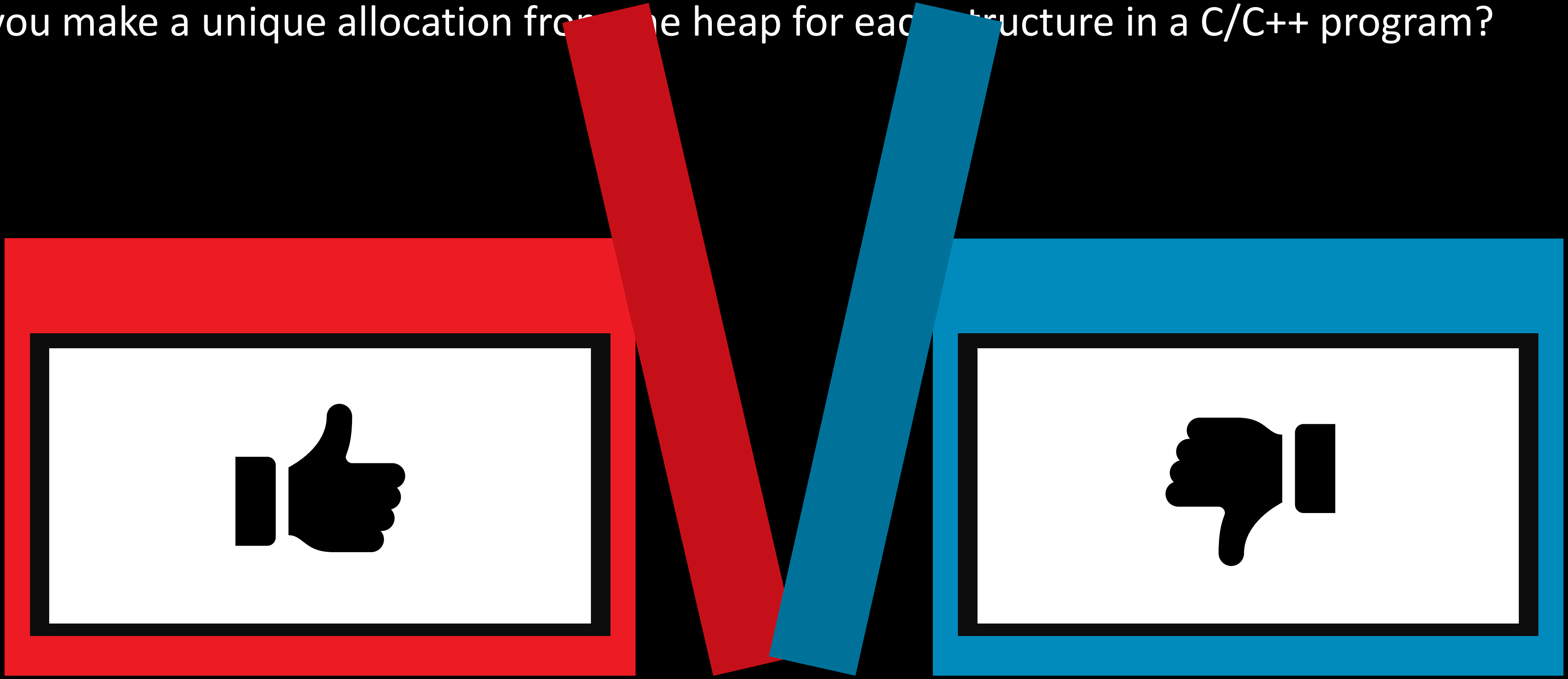
ALLOCATION STRATEGY

- Would you make a unique allocation from the heap for each structure in a C/C++ program?



ALLOCATION STRATEGY

- Would you make a unique allocation fro e heap for eac tructure in a C/C++ program?

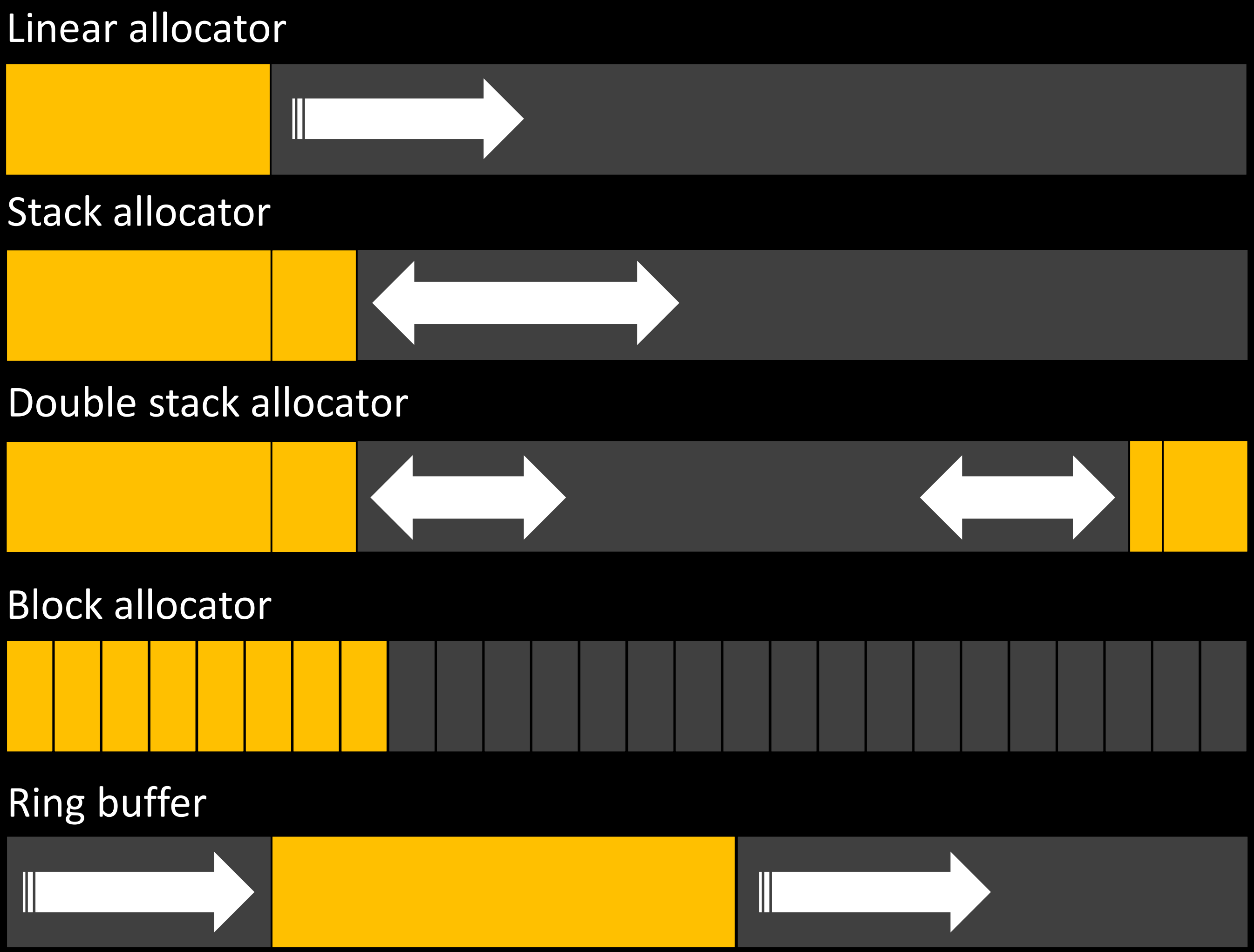


ALLOCATION STRATEGY

- Same idea on GPU for similar reasons:
  - Fragmentation
  - Performance
  - Data locality
  - Personal sanity

- Allocate reasonably large chunks of memory (256MiB).
  - Just 16 allocations fills 4GiB of VRAM.
  - Good balance between flexibility and performance.
    - On Windows®7 Vulkan memory allocations have larger overhead.
- Sub-allocate the memory for resources from these blocks.

ALLOCATION STRATEGY



Linear allocator

Stack allocator

Double stack allocator

Block allocator

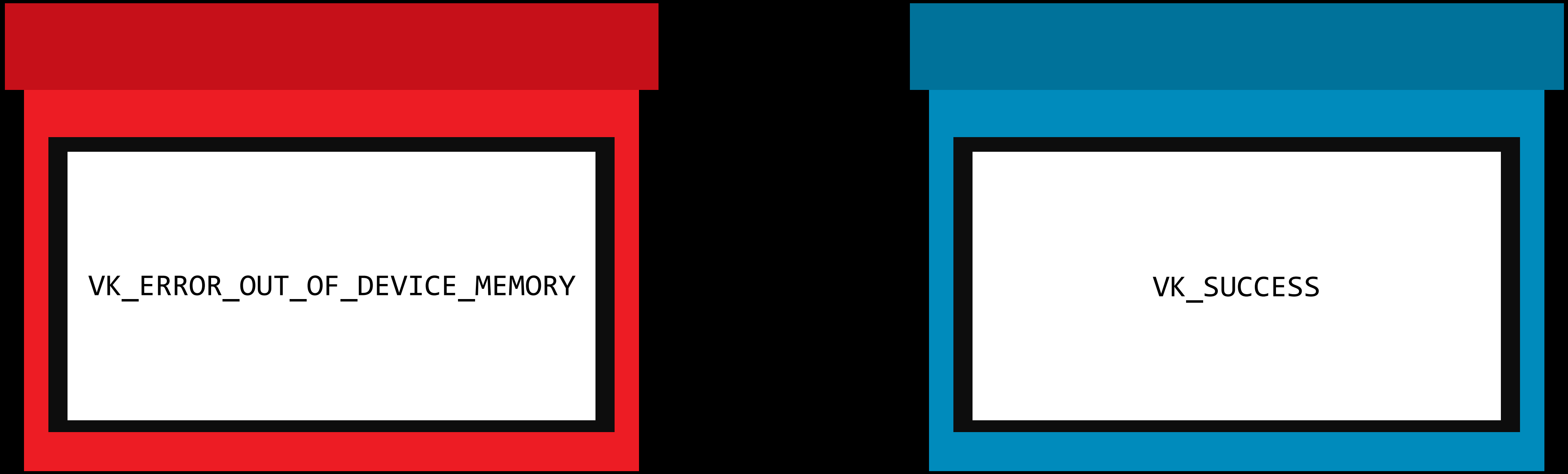
Ring buffer

🟨 - Used memory ⬛ - Free memory

Tips & Tricks

OVER-SUBSCRIPTION

- What happens when you run out of DEVICE_LOCAL memory (VRAM) and try vkAllocateMemory?

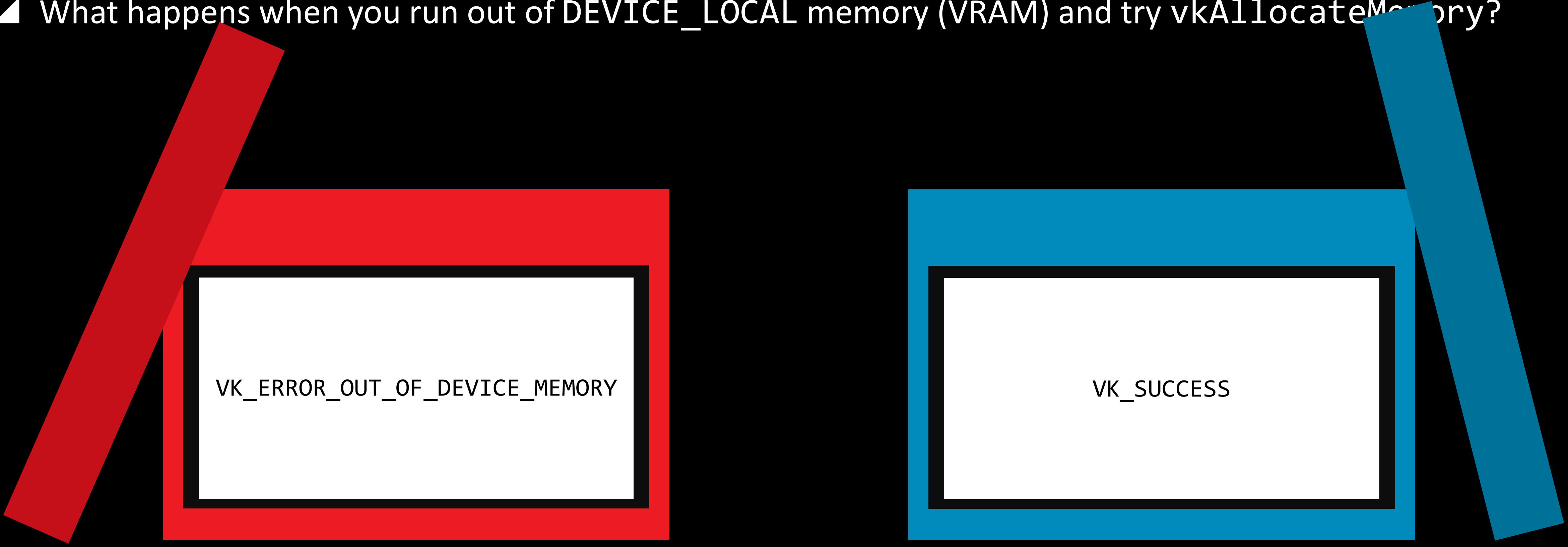


VK_ERROR_OUT_OF_DEVICE_MEMORY

VK_SUCCESS

OVER-SUBSCRIPTION

- What happens when you run out of DEVICE_LOCAL memory (VRAM) and try vkAllocateMemory?



VK_ERROR_OUT_OF_DEVICE_MEMORY

VK_SUCCESS

OVER-SUBSCRIPTION

VK_ERROR_OUT_OF_DEVICE_MEMORY

- Allocation fails.
- Application must handle out-of-memory conditions.
- Out-of-memory potentially changes per driver/hardware.

OVER-SUBSCRIPTION

VK_SUCCESS

- Allocation succeeds.
- Some blocks are silently migrated to system memory.

- Why would you want this?
  - Useful for development purposes - Artists don’t always stick to budgets.
  - Some of your blocks might get paged anyway (you’re not alone on the machine).
  - Application doesn’t have to handle out-of-memory.

- Accessing blocks migrated to system memory can degrade GPU performance.

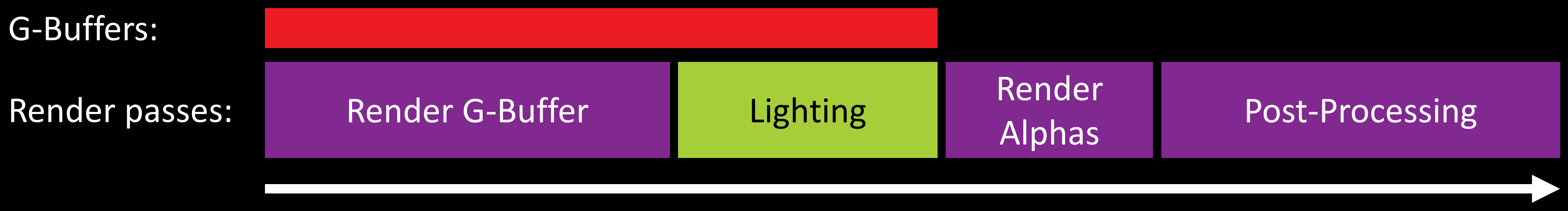
OVER-SUBSCRIPTION

- No way is exposed to control residency manually.
- No way is exposed to query the used/free memory.
- To make things worse, there are other implicit resources which need memory too:
  - Swap chains
  - Command buffers
  - Descriptors
  - Shaders / PSOs
  - Query results
- Use `VkMemoryHeap::size` then apply some “informed adjustments”:

Flags	Hack
DEVICE_LOCAL	VkMemoryHeap::size * 0.8f
DEVICE_LOCAL | HOST_VISIBLE	VkMemoryHeap::size * 0.66f

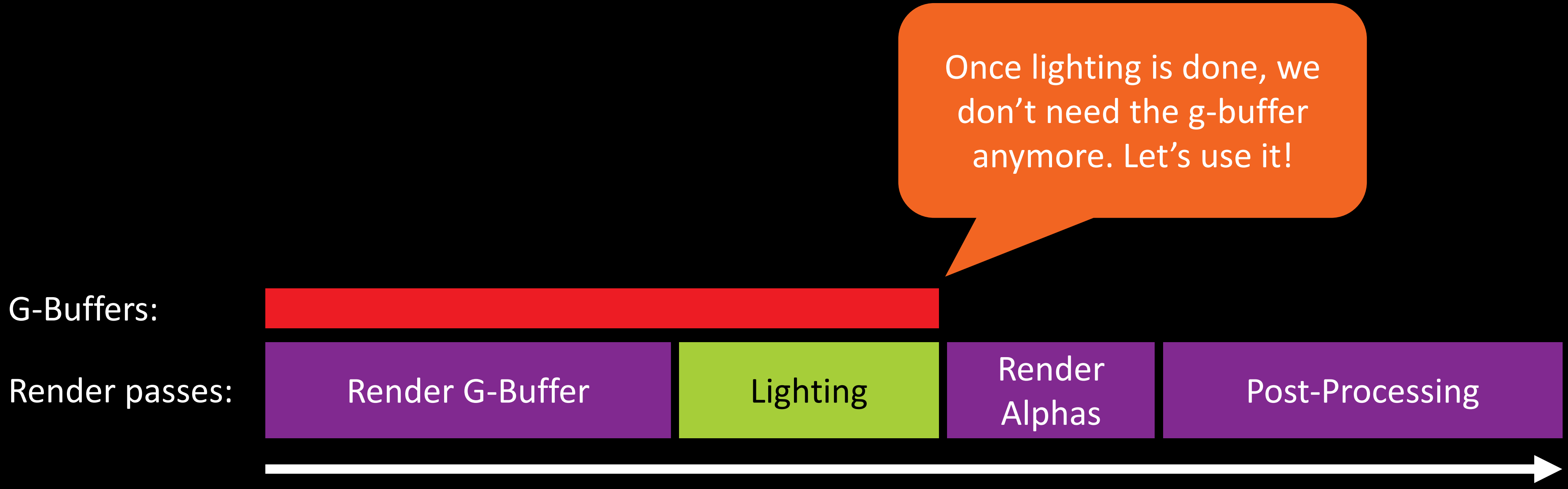
ALIASING

- As resolutions get larger, render targets follow suit.
- As many resources are transient, aliasing can be a solution to keep render target/UAV memory in check.



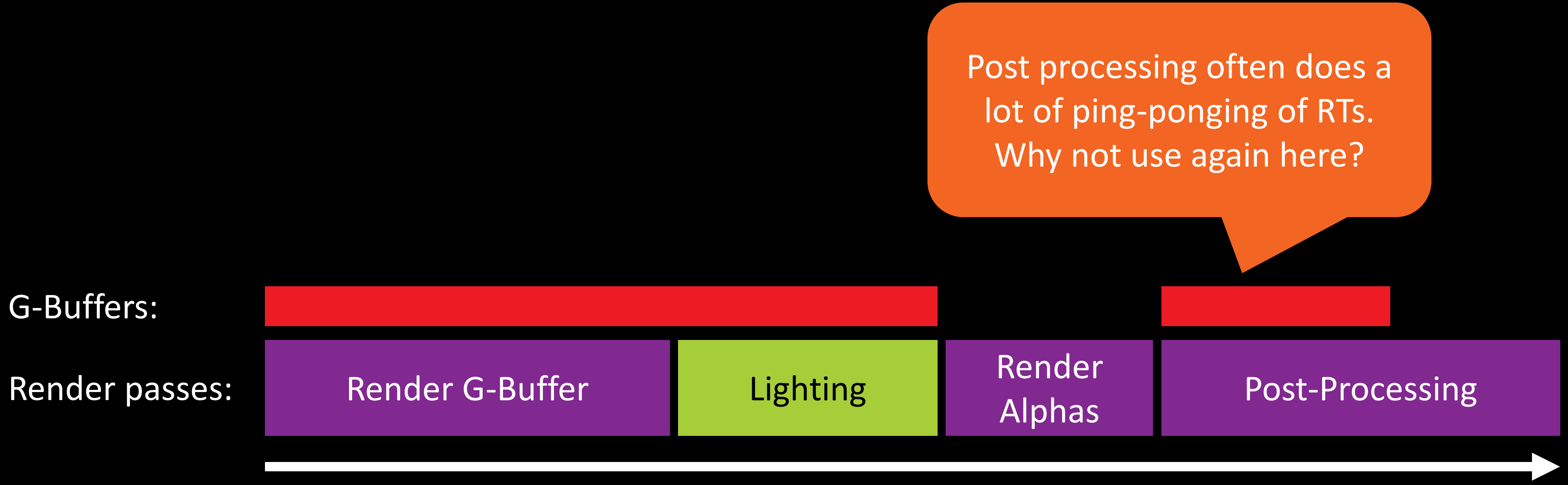
ALIASING

- As resolutions get larger, render targets follow suit.
- As many resources are transient, aliasing can be a solution to keep render target/UAV memory in check.



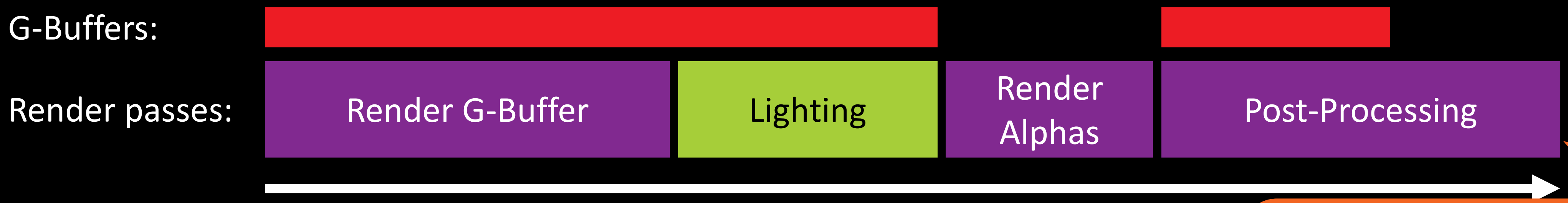
ALIASING

- As resolutions get larger, render targets follow suit.
- As many resources are transient, aliasing can be a solution to keep render target/UAV memory in check.


Post processing often does a lot of ping-ponging of RTs. Why not use again here?
G-Buffers:
Render passes:
Render G-Buffer
Lighting
Render Alphas
Post-Processing

ALIASING

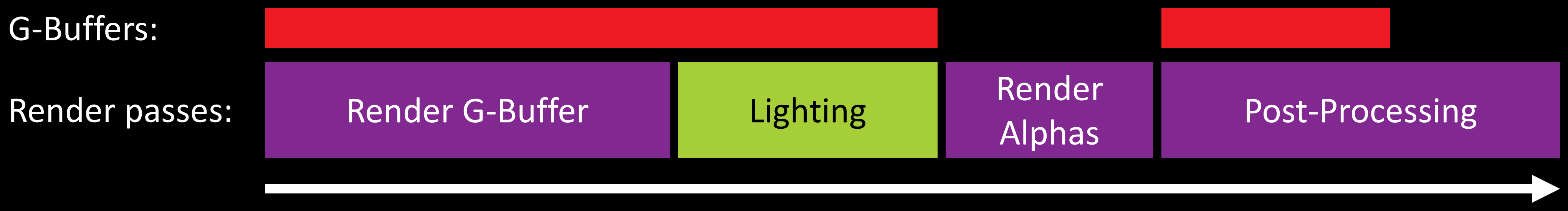
- As resolutions get larger, render targets follow suit.
- As many resources are transient, aliasing can be a solution to keep render target/UAV memory in check.



Maybe some compute shader in here needs a nice big UAV for something. No need to allocate, alias with a Render target.

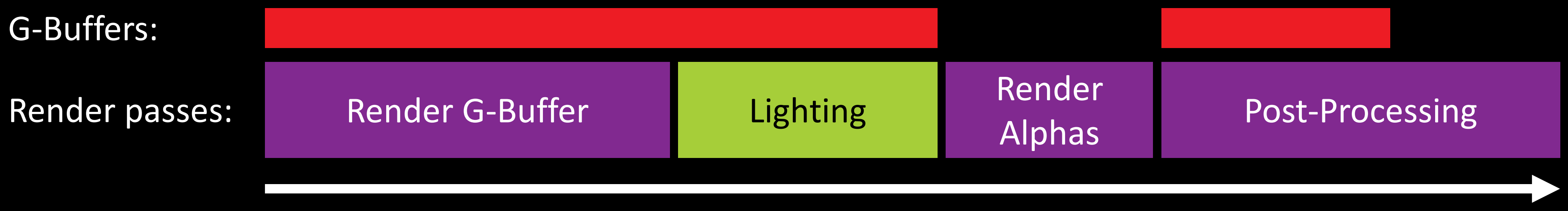
ALIASING

- As resolutions get larger, render targets follow suit.
- As many resources are transient, aliasing can be a solution to keep render target/UAV memory in check.
- For second, third, etc. use of aliased resource best to assume it contains garbage.



ALIASING

- As resolutions get larger, render targets follow suit.
- As many resources are transient, aliasing can be a solution to keep render target/UAV memory in check.
- For second, third, etc. use of aliased resource best to assume it contains garbage.
- >50% memory saved in some titles [ODonnell17].



TRANSFERS

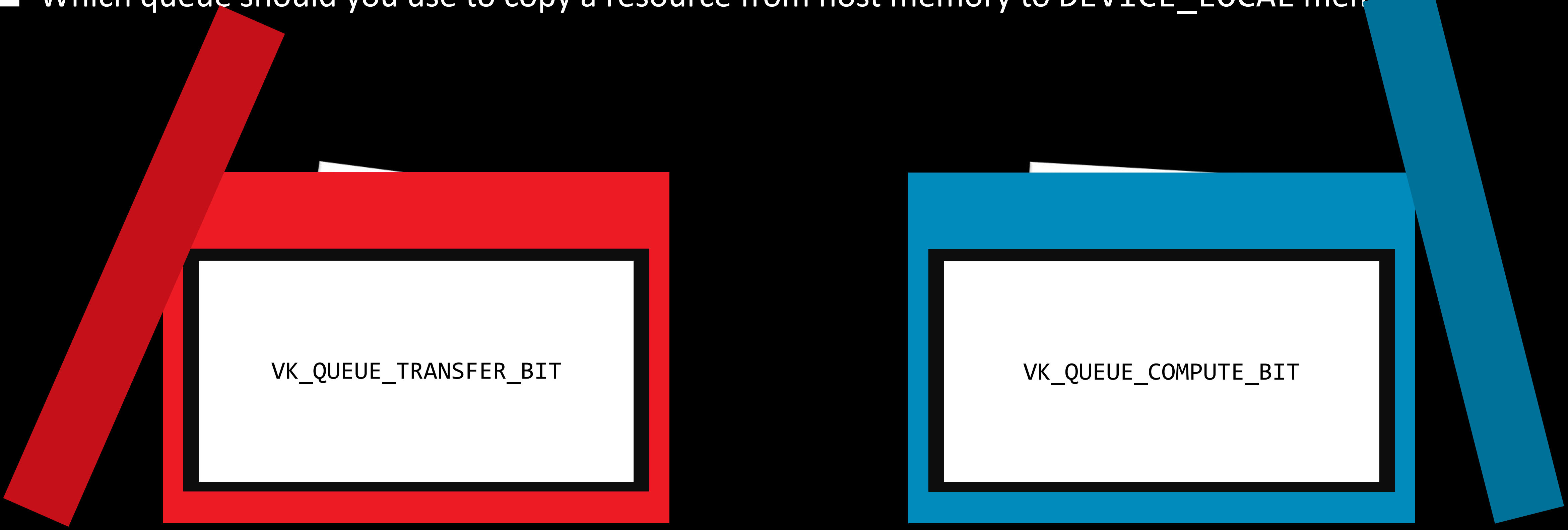
- Which queue should you use to copy a resource from host memory to DEVICE_LOCAL memory?

VK_QUEUE_TRANSFER_BIT

VK_QUEUE_COMPUTE_BIT

TRANSFERS

- Which queue should you use to copy a resource from host memory to DEVICE_LOCAL memory?



VK_QUEUE_TRANSFER_BIT

VK_QUEUE_COMPUTE_BIT

TRANSFERS

- The transfer queue is great.
  - DMA hardware that you drive asynchronously – doesn’t touch other queues.
  - The fastest way to copy across PCIe bus.
- Kick DMAs off as early as you can, waaaaaay before you need them on graphics/compute queue.
- Some hardware even has 2 transfer queues.
  - E.g. RX 580

TRANSFERS

- What about `DEVICE_LOCAL` to `DEVICE_LOCAL` copies?
  - Choice not as clear cut.
  - Peak transfer rates of Graphics/Compute are probably faster, but it “clogs up” GPU.
  - If you can pipeline the copies, transfer queue can still be a win.

- Use the queue to defrag your allocations.
  - Copy to a new address.
  - Next frame, update descriptor.

- General rules of thumb:
  - Need it now? Graphics/Compute queue.
  - It can wait? Transfer queue.
  - Respect granularity of the queue. Full sub-resource is fine.
  - Measure, measure, measure. Queue semaphores can cost you.

MAPPING

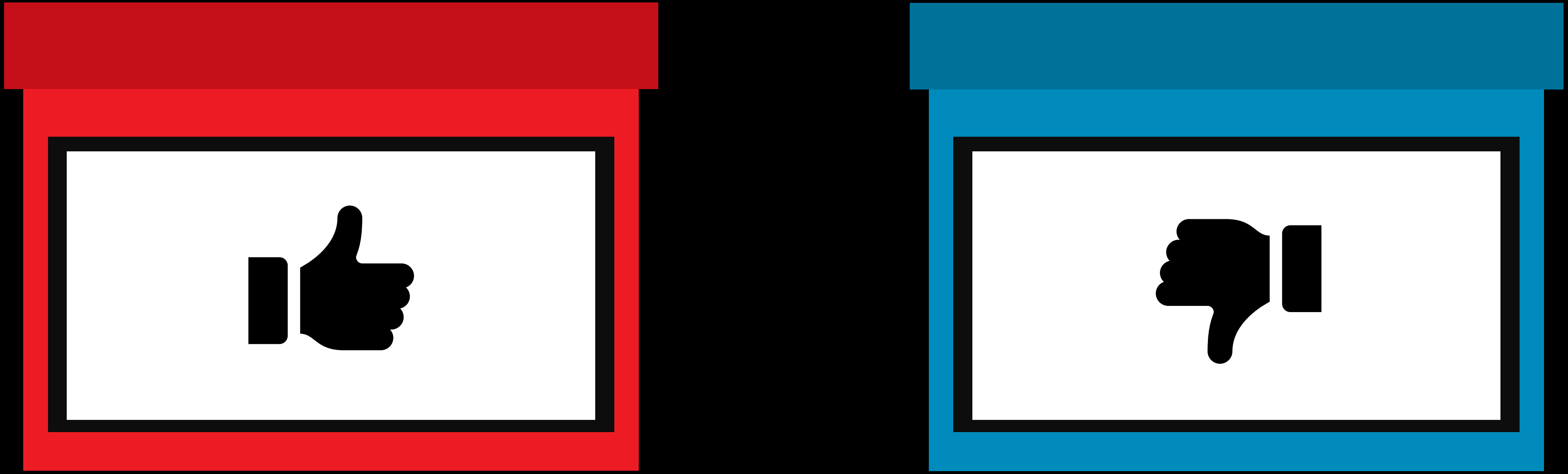
- Having your entire memory block persistently mapped is generally OK.
  - No longer any need to unmap before using stuff on GPU!
- Exceptions:
  - AMD, Windows® version < 10. Blocks of `DEVICE_LOCAL` that are also `HOST_VISIBLE` (type 2) that are still mapped at the time of `Submit` or `Present` will be migrated to system memory.
  - Keeping many large memory blocks mapped may impact stability/performance of debugging tools.

RESOURCE CREATION

- Avoid the following Vulkan ‘lazy-mode’ features:
  - `VK_IMAGE_LAYOUT_GENERAL`
    - Prefer to transition to appropriate `VK_IMAGE_LAYOUT_*_OPTIMAL` state.
  - `VK_SHARING_MODE_CONCURRENT` on render targets or depth buffers
    - It nobbles DCC compression.
    - Go for `VK_SHARING_MODE_EXCLUSIVE` and do explicit queue family ownership barriers.
  - `VK_IMAGE_TILING_LINEAR`
    - `VK_IMAGE_TILING_OPTIMAL` is more… well… optimal.
    - You can always copy to a buffer to de-tile things.
  - Setting too many usage bits on your stuff.
    - Great way to confuse drivers into flushing more caches, and draining the GPU.

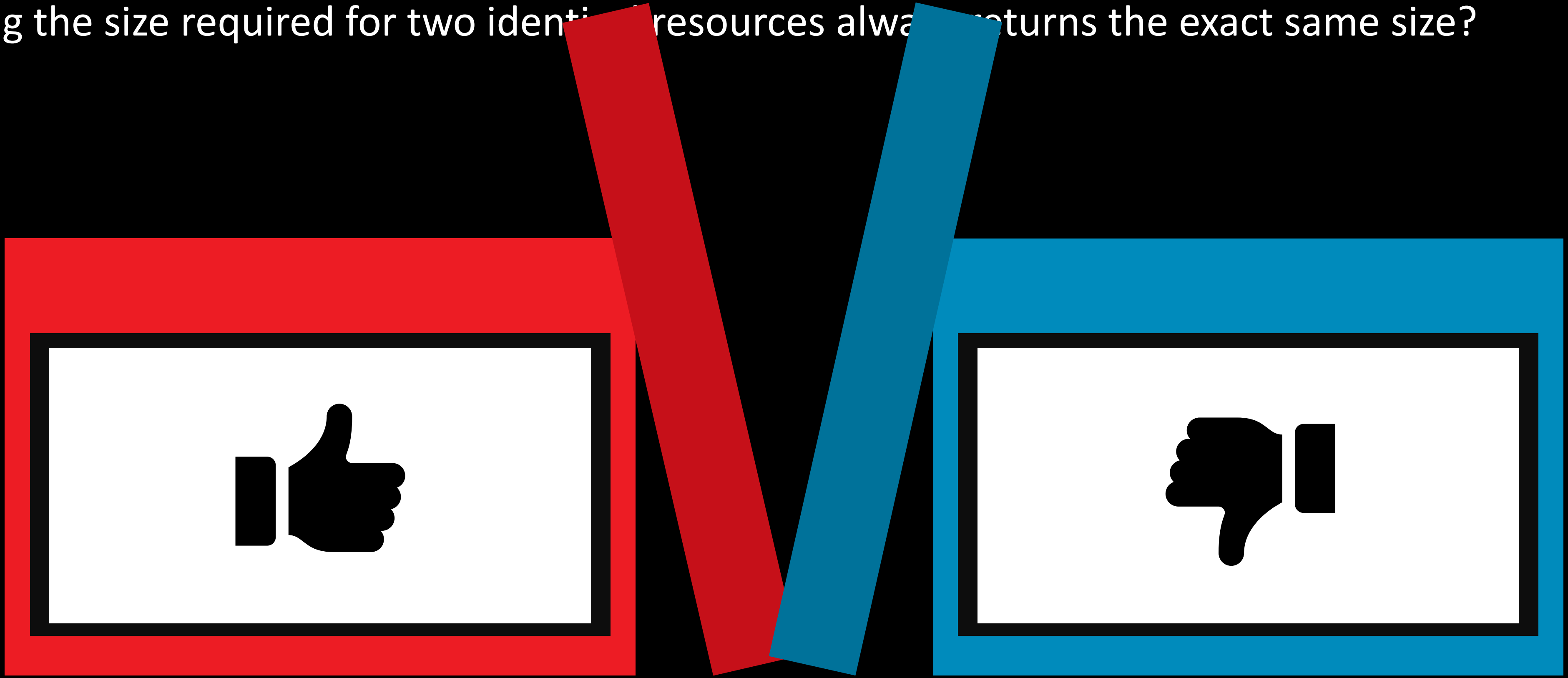
RESOURCE SIZES

- Querying the size required for two identical resources always returns the exact same size?



RESOURCE SIZES

- Querying the size required for two identi resources alwa turns the exact same size?



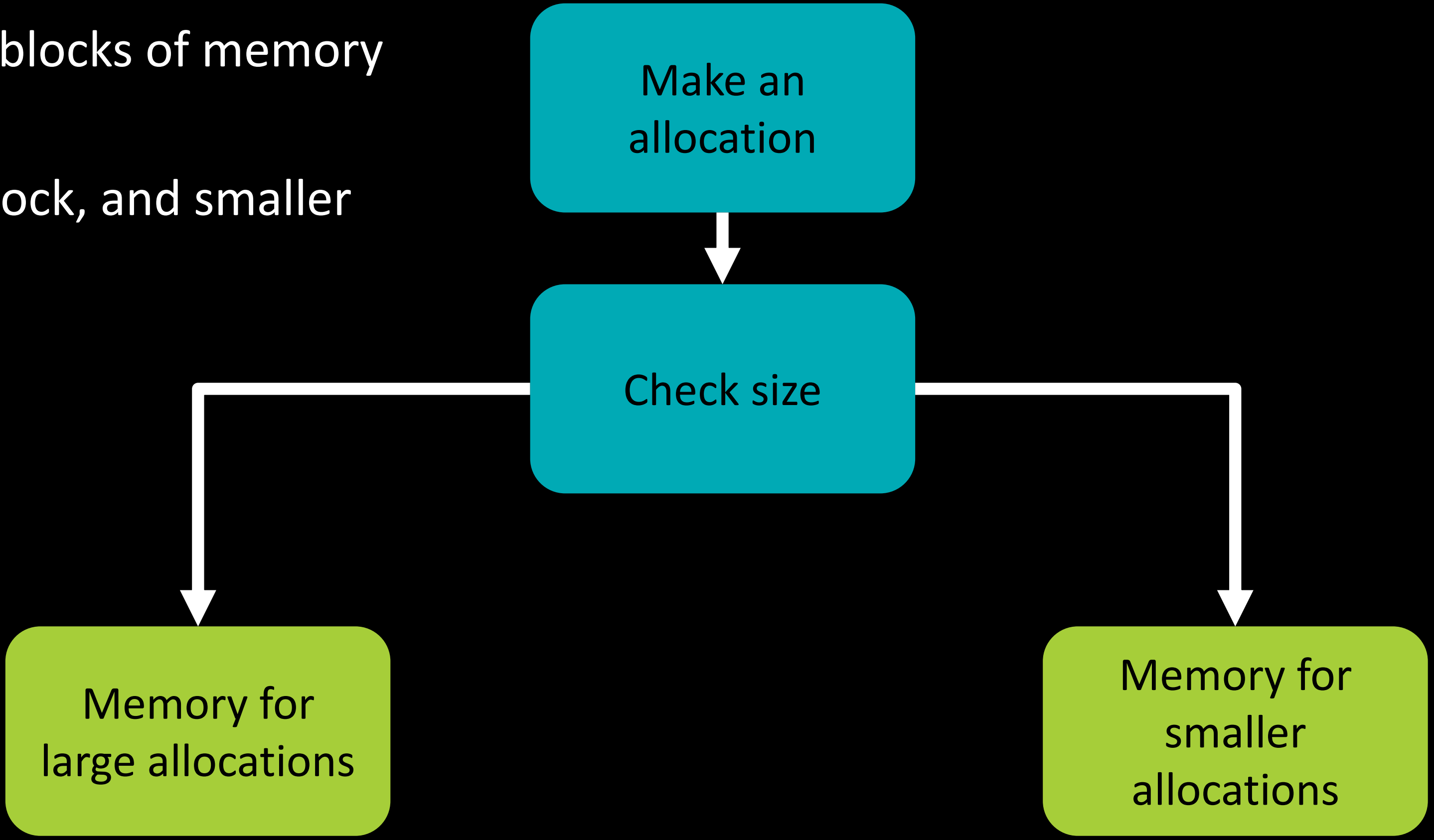
RESOURCE SIZES

- Memory requirements - e.g. size - can vary for different, similar resources.
  - Same format, width, height and mip-levels.
- Not just the preserve of ‘spec wonks’. This *really* happens - don’t cache the results when querying sizes.
- Make sure you query *each* resource for it’s specific requirements.

SMALL BLOCK ALLOCATIONS

DON'T GET FRAGGED!

- Mixing large and small allocations carelessly can be painful.
- Consider routing allocations to different blocks of memory based on their sizes.
- Pool larger allocations together in one block, and smaller allocations together in another one.



The Vulkan Memory Allocator

VULKAN MEMORY ALLOCATOR (VMA)

- Free. Open source. MIT license. Single header.
  - https://github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator
- Simple, C99 interface. Same style as Vulkan™.
- Battle tested, already getting some love in the community.

VULKAN MEMORY ALLOCATOR (VMA)

- Function that help to choose the correct and optimal memory type based on intended usage.
- Functions that allocate memory blocks, reserve and return parts of them to the user.
- Allocation tracker, look at used/unused, and fragmentation.
- Respects alignment and buffer/image granularity.

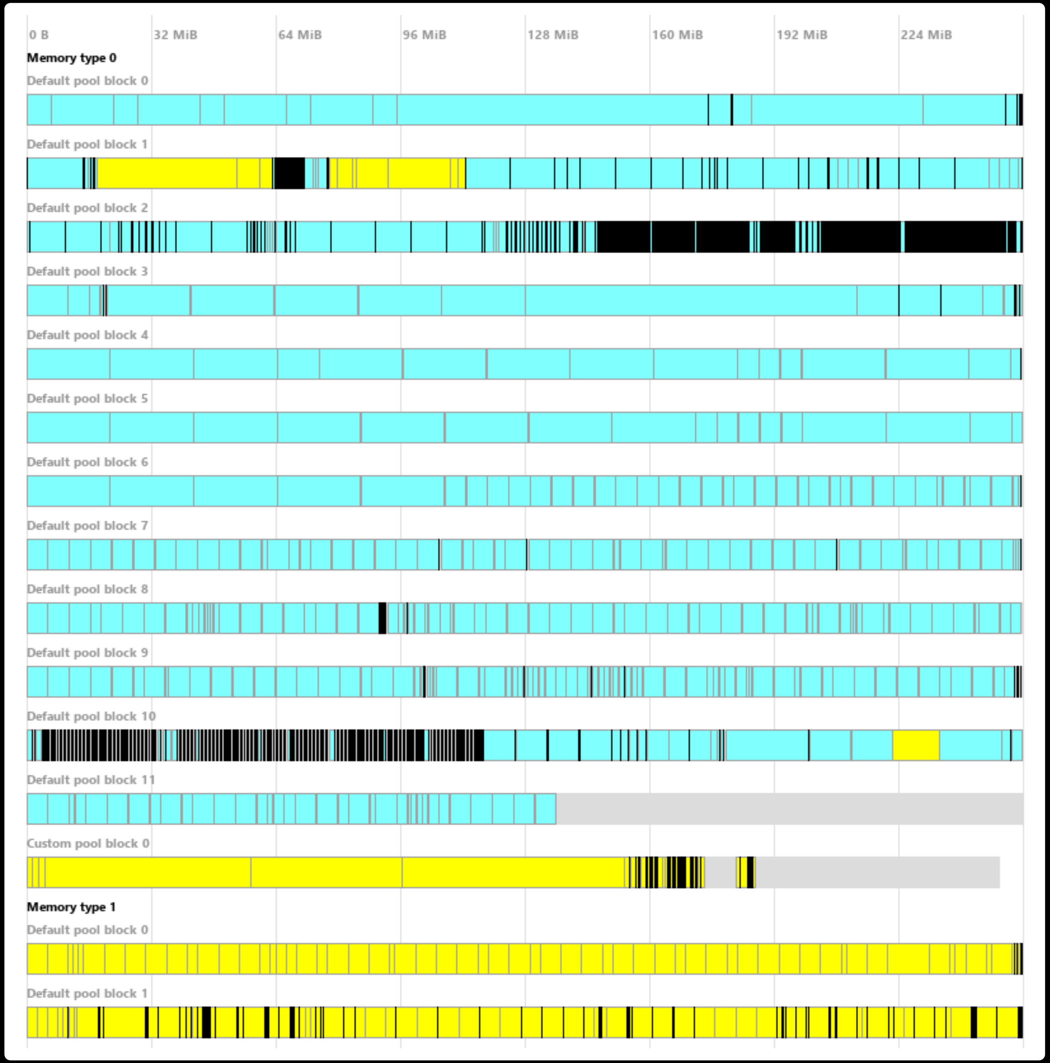
```
VkBufferCreateInfo bufferInfo = { VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO };
bufferInfo.size = 65536;
bufferInfo.usage = VK_BUFFER_USAGE_VERTEX_BUFFER_BIT | VK_BUFFER_USAGE_TRANSFER_DST_BIT;

VmaAllocationCreateInfo allocInfo = {};
allocInfo.usage = VMA_MEMORY_USAGE_GPU_ONLY;

VkBuffer buffer;
VmaAllocation allocation;
vmaCreateBuffer(allocator, &bufferInfo, &allocInfo, &buffer, &allocation, NULL);
```

VULKAN MEMORY ALLOCATOR (VMA)

- Even has some tooling!
- VMA can dump allocator state to JSON.
- Python script generates PNG file which shows the allocator contents.



CONCLUSION

- Vulkan is lower-level and requires explicit memory management.
  - Creating resources is a multi-stage process.
  - Former driver magic is now under your control.
- You need to deal with differences between GPUs.
- By following good practices you can achieve optimal performance on any GPU.
- Vulkan Memory Allocator (VMA) is battle-tested and can really help a lot.

SPECIAL THANKS

- Adam Sawicki
- Dominik Baumeister
- Lou Kramer
- Matthäus G. Chajdas
- Rys Sommefeldt
- Timothy Lottes
- Nicolas Thibieroz
- Alon Or-Bach

REFERENCES

- [ODonnell17]
  Yuriy O'Donnell – FrameGraph: Extensible Rendering Architecture in Frostbite
  https://www.gdcvault.com/play/1024612/FrameGraph-Extensible-Rendering-Architecture-in

DISCLAIMER & ATTRIBUTION

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

AMD