Vulkan®

# DEVELOPER DAY

## Memory management in Vulkan

Jordan Logan, AMD

KHRONOS™ GROUP

UBISOFT

MONTRÉAL
APRIL 2018

# Agenda

- Introduction
- Memory Types
- Tips & Tricks
- Libraries
- Conclusions


KHRONOS GROUP

# Introduction

# The challenge

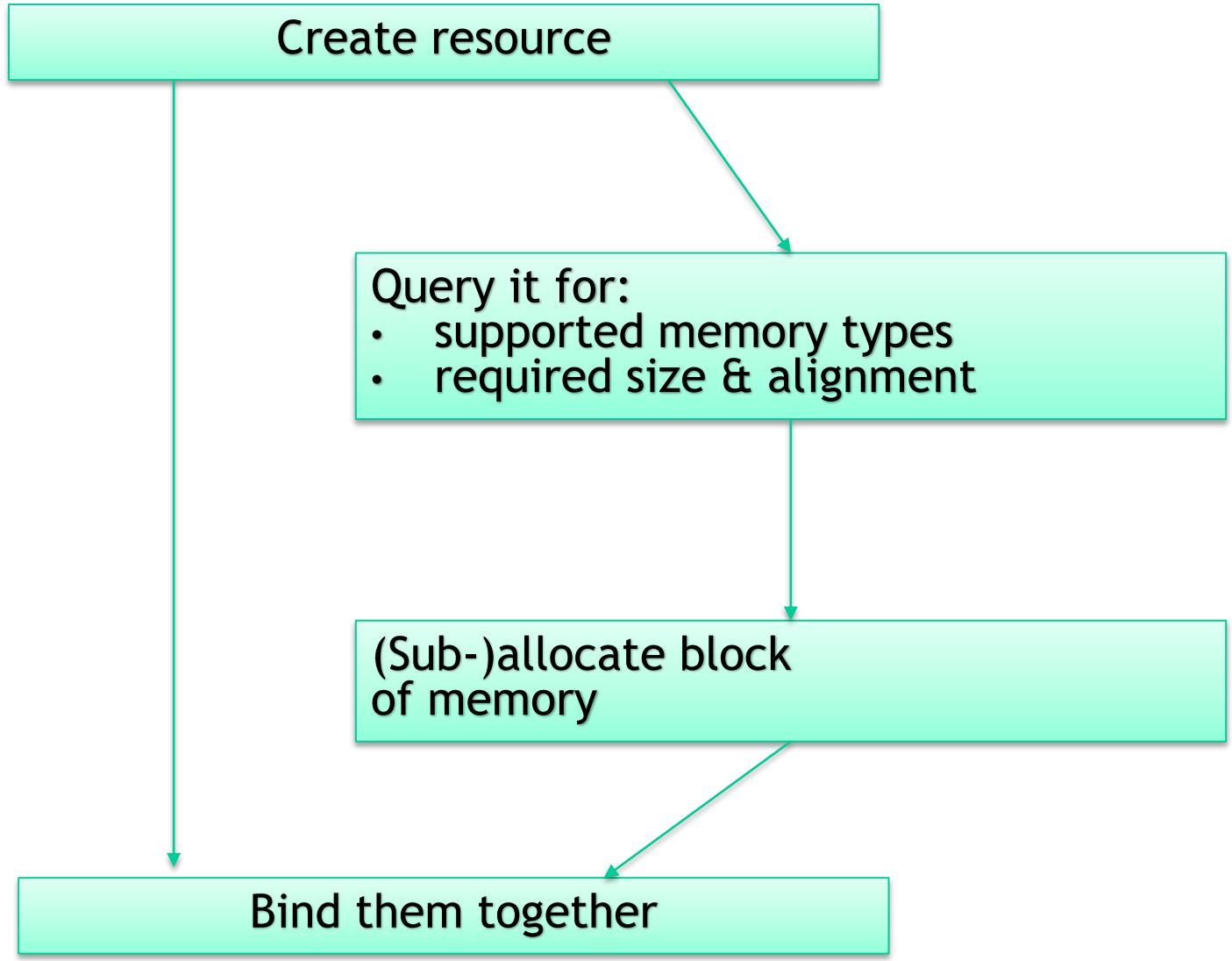
- **Previous generation APIs (OpenGL™, DirectX® 11) manage memory automatically.**
  - You create a resource (e.g. texture, constant buffer), backing memory is allocated automatically.

```
ID3D11Texture2D* pTexture;
pD3D11Device->CreateTexture2D(&desc, nullptr, &pTexture);
```

- **Vulkan™ is lower level, requires explicit memory management.**

# The challenge

- It is now your responsibility to:



# Advantages

Explicit memory management makes it possible to:

- Achieve a smaller memory footprint.
- Better optimize for specific platforms.
- Alias (overlap) transient resources.

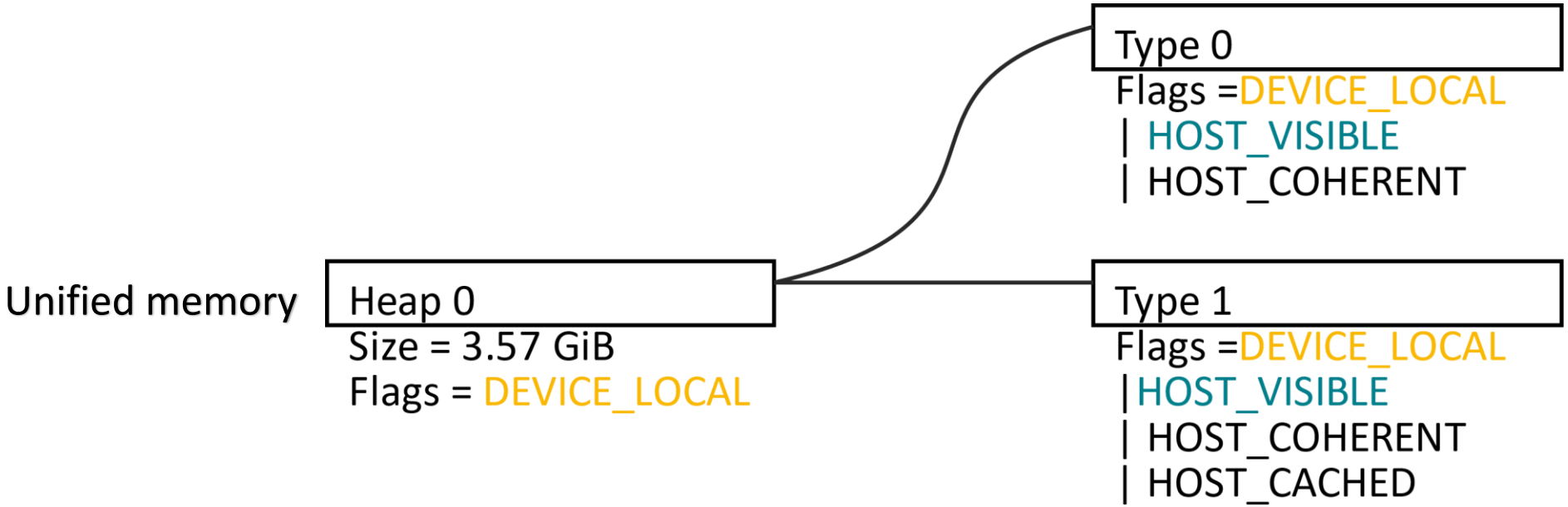
# Memory Types

# Memory Heaps and types

- Heaps: Physical memory (e.g. VRAM, DRAM).
- Types: Describes the properties of the memory and which heap to use.

# Memory types: Intel

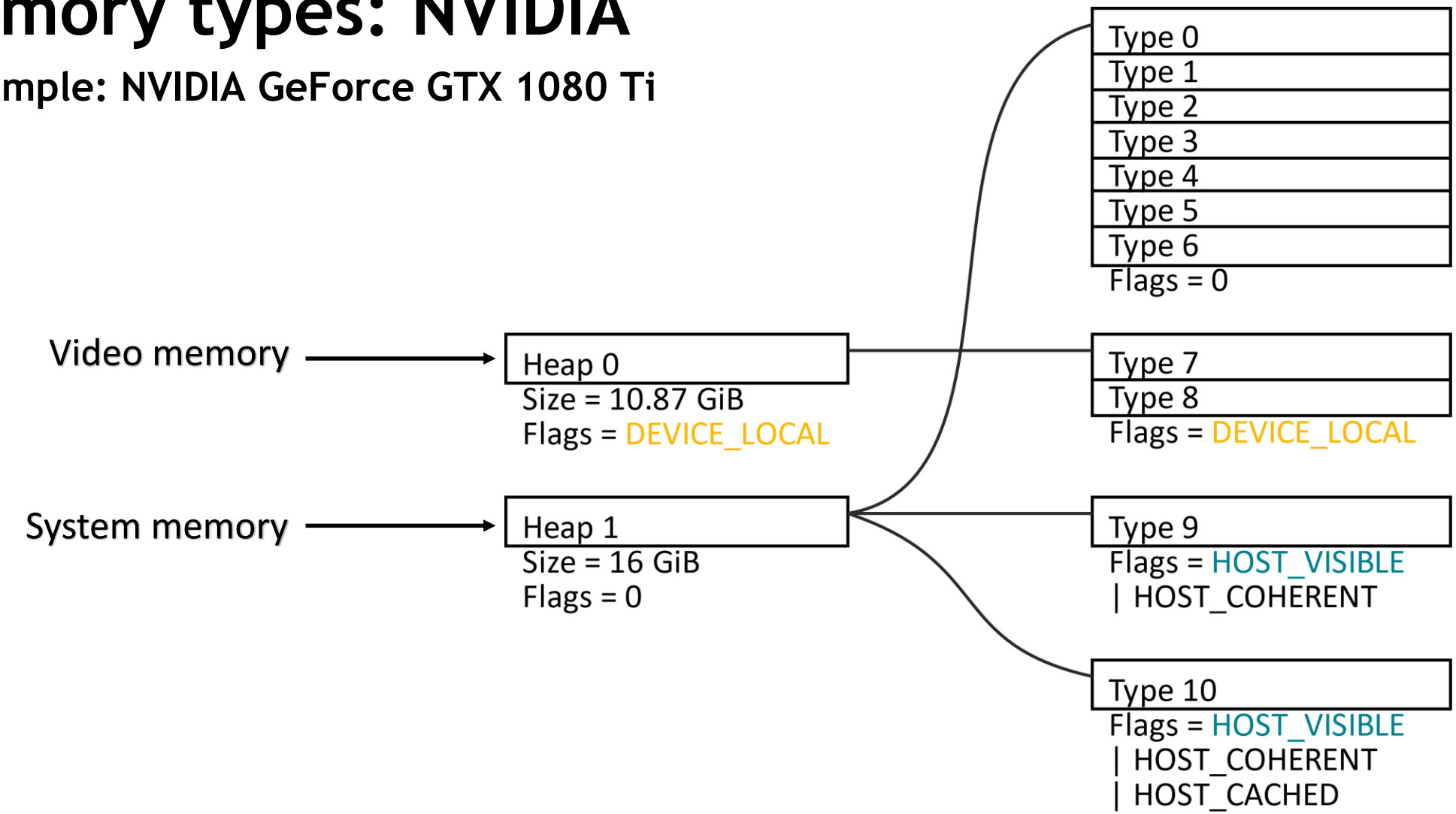
- Example: Intel Iris Plus Graphics 640



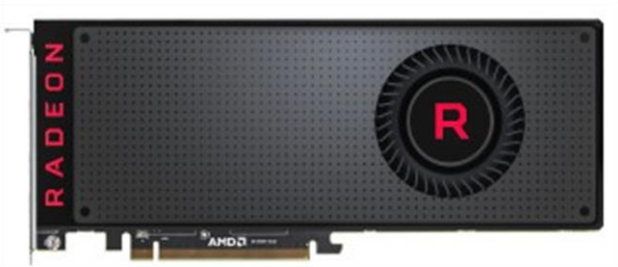
# Memory types: NVIDIA

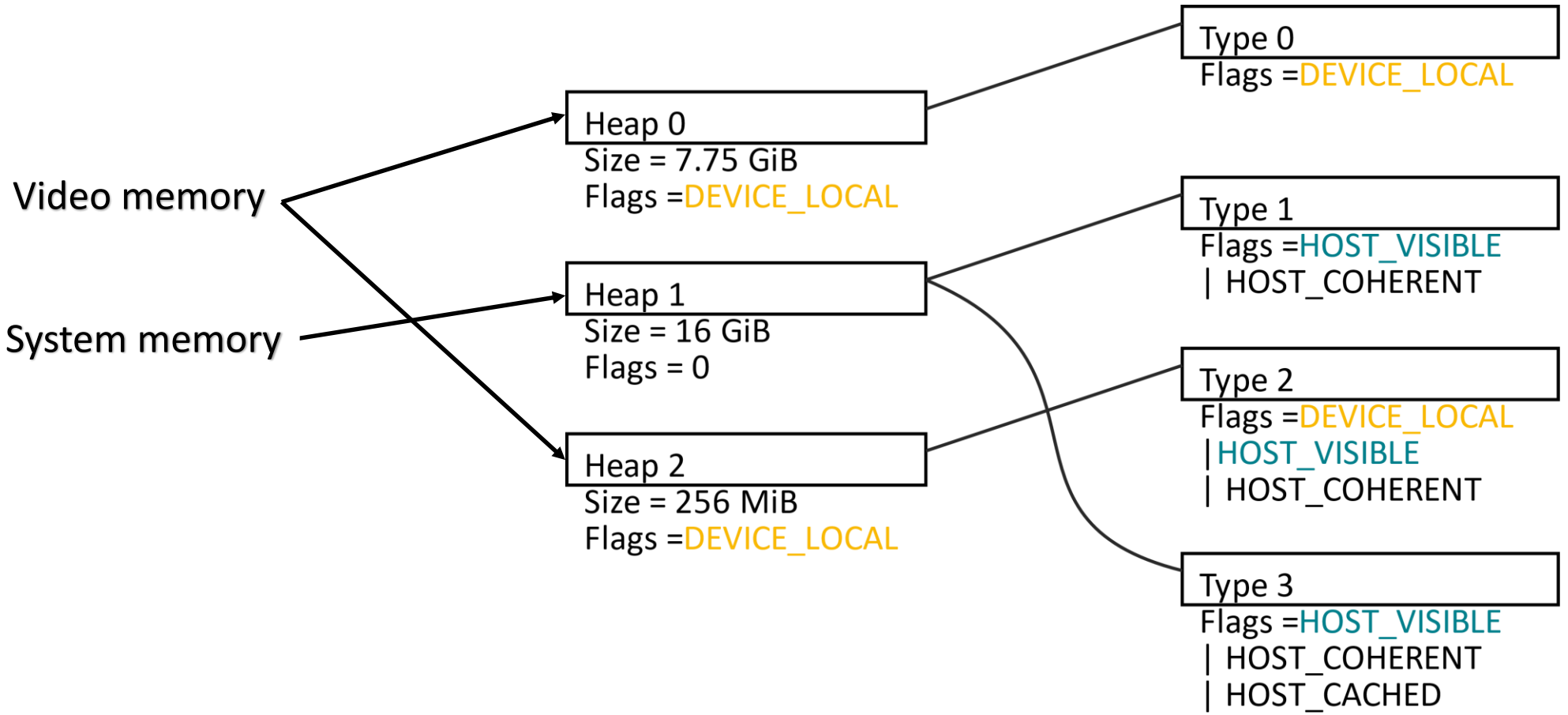
- Example: NVIDIA GeForce GTX 1080 Ti



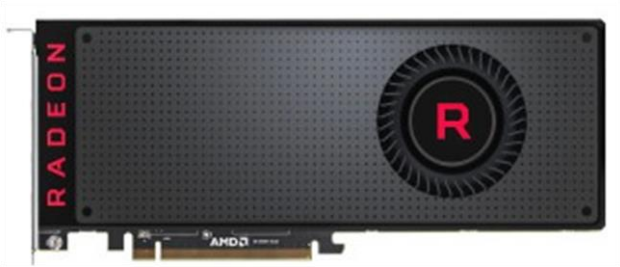
# Memory types: AMD

- Example: AMD Radeon RX Vega 64



# DEVICE_LOCAL

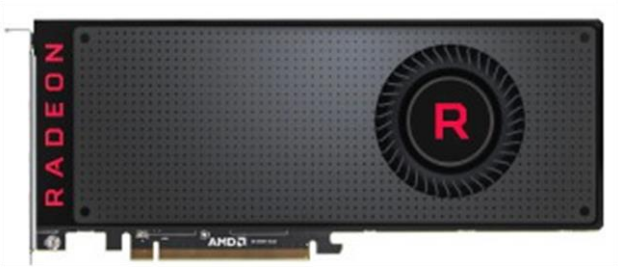



- Video memory. Fast access from GPU.
- No direct access from CPU – mapping not possible.
- Good for resources written and read frequently by GPU.
- Good for resources uploaded once (immutable) or infrequently by CPU, read frequently by GPU.

# HOST_VISIBLE


- System memory. Accessible to CPU – mapping possible.
- Access from GPU possible but slow.
- Across PCIe® bus, reads cached on GPU.
- Good for CPU-side (staging) copy of your resources – used as source of transfer.
- Data written by CPU, read once by GPU (e.g. constant buffer) may work.
  - (always measure!)
- Large data read by GPU – place here as last resort.
- Large data written and read by GPU – shouldn't ever be here.
- Hazard: Uncached. Writes may be write-combined.

# DEVICE_LOCAL + HOST_VISIBLE



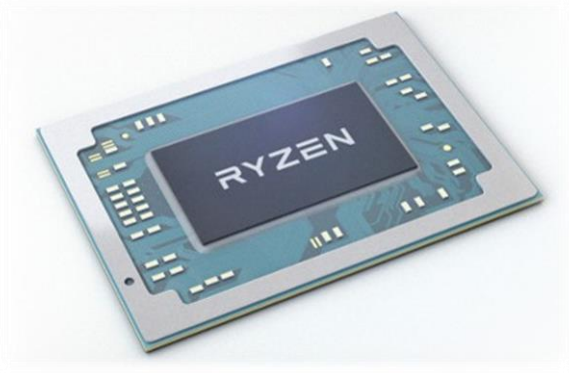

- **Special pool of video memory.**
- **Exposed on AMD only. 256 MiB.**
- **Fast access from GPU.**
- **Direct access by both CPU and GPU.**
  - You don't need to do explicit transfer.
  - Mapping possible.
- **Good for resources updated frequently by CPU (dynamic), read by GPU.**
- **Use as fallback if DEVICE_LOCAL is small and oversubscribed.**
- **Hazard: Driver will use this memory too.**
- **Hazard: Uncached. Writes may be write-combined.**

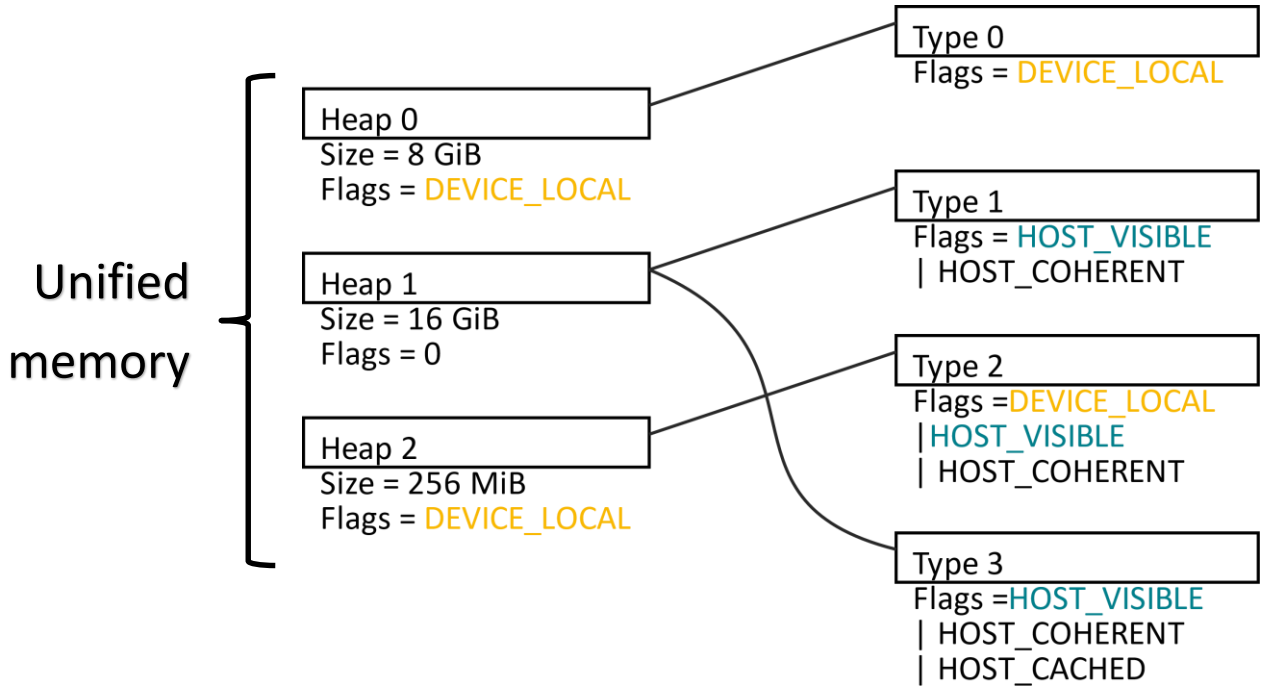
# HOST_VISIBLE + HOST_CACHED


- System memory
- CPU reads and writes cached (write-back).
- GPU access through PCIe.
- GPU reads snoop CPU cache.
- Good for resources written by GPU, read by CPU.
  - Results of computations.
- Direct access by both CPU and GPU.
  - No need to do explicit transfer.
- Use for any resources read or accessed randomly on CPU.

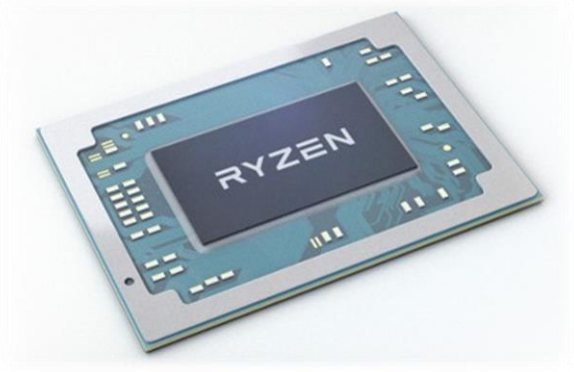
# Memory types: AMD APU

- AMD integrated graphics reports various memory types, like discrete AMD GPUs.
- Reported DEVICE_LOCAL heap can be any size,
  0 B … few GiB.



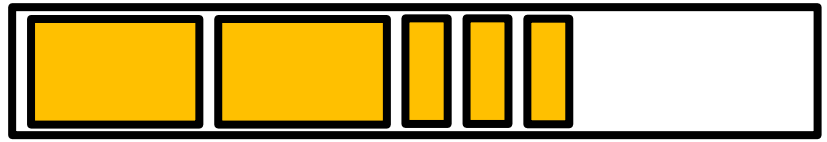
# Memory types: AMD APU




- **Memory is really unified.**
  - DEVICE_LOCAL is still faster.
  - Render and Depth Targets will see the most benefit.
- **If you detect integrated graphics:**
  **VK_PHYSICAL_DEVICE_TYPE_INTEGRATED_GPU:**
  - Count size of all memory heaps together.
  - Put your resources in DEVICE_LOCAL first. If that heap is full then try other heaps.

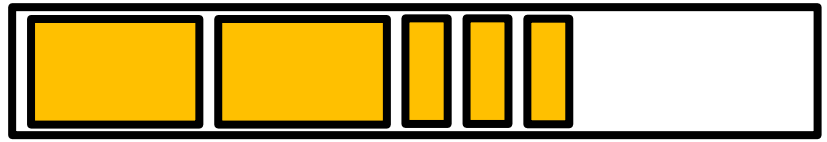
# Tips & Tricks

# Memory blocks



- **In Old API's the resource owns the memory.**
  - Memory is allocated when creating the resource.
  - No control of where that memory is placed.
- **In Vulkan the memory is allocated by itself.**
  - Memory is allocated in blocks.
  - A single block can have many resources.
- **Allows many new optimization techniques**
  - Pool similar resources for tighter packing.
  - Keep object resources together to simplify streaming of that objects required resources.

# Suballocation



- **Don’t allocate a separate memory block for each resource.**
  - Small limit on maximum number of allocations (e.g. 4096).
- **Allocate bigger blocks and sub-allocate ranges for your resources.**
  - 256 MiB is good default block size.
  - For heaps <= 1 GiB use smaller blocks.
    (e.g. heap size / 8).
- **Allocations are slow.**
  - Prefer not to allocate or free memory blocks during gameplay to avoid hitching.
  - If you need to, you can do it on background thread.

# Over-commitment

**What happens when you exceed the maximum amount of physical video memory?**

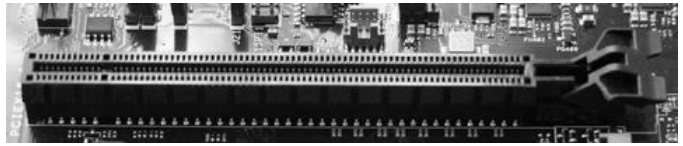
- **It depends on the driver.**
  - Allocation may fail (VK_ERROR_OUT_OF_DEVICE_MEMORY).
  - Allocation may succeed (VK_SUCCESS).
    - Some blocks are silently migrated to system memory.
- **Blocks may be migrated to system memory anyway.**
  - You are not alone – other applications can use video memory.
  - Using blocks migrated to system memory on GPU degrades performance.
- **Don’t use 100% of the heap**
  - Leave room for implicit resources
  - (e.g. 20% of DEVICE_LOCAL,
    33% of DEVICE_LOCAL + HOST_VISIBLE).

# Mapping

```
void* ptr;
```

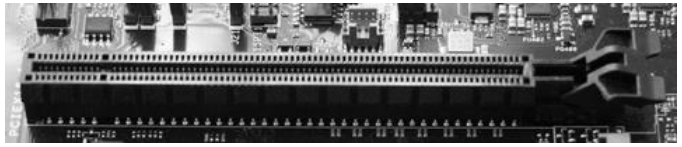
- **No more versioning of resources by the API.**
- **Having entire memory block persistently mapped is generally OK.**
  - You don’t need to unmap before using on GPU.
- **Exceptions:**
  - AMD, Windows® version < 10: Blocks of DEVICE_LOCAL + HOST_VISIBLE memory that stay mapped for the time of any call to Submit or Present are migrated to system memory.
  - Keeping many large memory blocks mapped may impact stability or performance of debugging tools.

# Transfer



- **Transfer queue is designed for efficient transfer via PCIe.**
  - Heavily underutilized queue.
  - Use it in parallel with 3D rendering, even asynchronously to rendering frames.
    - Good for texture streaming.
  - Fastest way to copy across the PCIe bus.
  - Does not use any compute or graphics hardware.
  - Do your transfers long before the data is needed on graphics queue.
- **Transfer queue can be used for defragmenting.**
  - Copy a resource to a new address asynchronously to rendering.
  - When done update next frames descriptor.

# Transfer



- **Some hardware supports more than one queue.**
  - e. g. AMD RX 580 has 2 transfer queues.
  - Can use one queue to defrag while the other one updates frame resources. (constant buffers, etc.)
- **GPU to (the same) GPU copies are much faster on graphics queue.**
  - Use it if graphics queue needs to wait for transfer result anyway.
- **Be careful with granularity.**
  - Always respect queue granularly for copies otherwise you may have corruption, crash, or even deadlock.
  - Don’t need to worry when doing full sub resource copies.

# Aliasing

- You can alias different resources – bind them to the same or overlapping range of memory.
- It saves memory.
- Good for transient resources (e.g. render targets) used only during part of the frame.
- After the memory was used by different resource, treat your resource as uninitialized.
  - …unless you really know what you're doing.

# Resource Creation

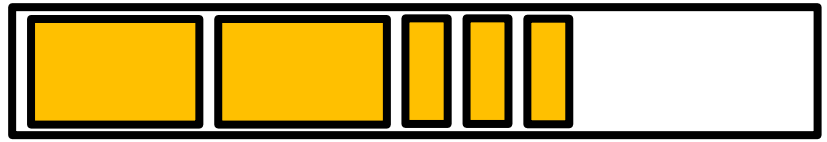
- **Don’t use images with TILING_LINEAR unless you have to.**
  - Use TILING_OPTIMAL.
  - Copy images from/to buffers.
- **Minimize number of USAGE_BITS used during resource creation.**
  - Only fill out what you need.
- **Avoid VK_SHARING_MODE_CONCURRENT on render and depth target textures.**
  - It disables optimizations.
  - Prefer VK_SHARING_MODE_EXCLUSIVE and do explicit queue family ownership transfer barriers.
- **Use VK_KHR_image_format_list for mutable formats.**
  - If you need different formats e.g. to interpret as linear/sRGB, use the VK_KHR_image_format_list extension with the VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT.

# Libraries

# Vulkan Memory Allocator

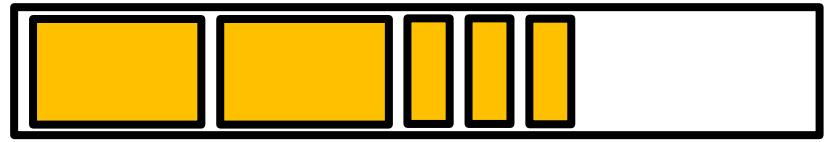
- https://github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator
- **Library from AMD.**
  - MIT license
  - easy to integrate – single header
  - interface in C (same style as Vulkan™), implementation in C++
  - well documented
- **Already used in some AAA titles.**
- **Great starting point even if you write your own.**

# Vulkan Memory Allocator



- **Functions 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.**
  - Library keeps track of allocated memory blocks, used and unused ranges inside them,
  - respects alignment and buffer/image granularity.

# Vulkan Memory Allocator



- **Functions that create image/buffer, (sub-)allocate memory for it and bind them together - all in one call.**

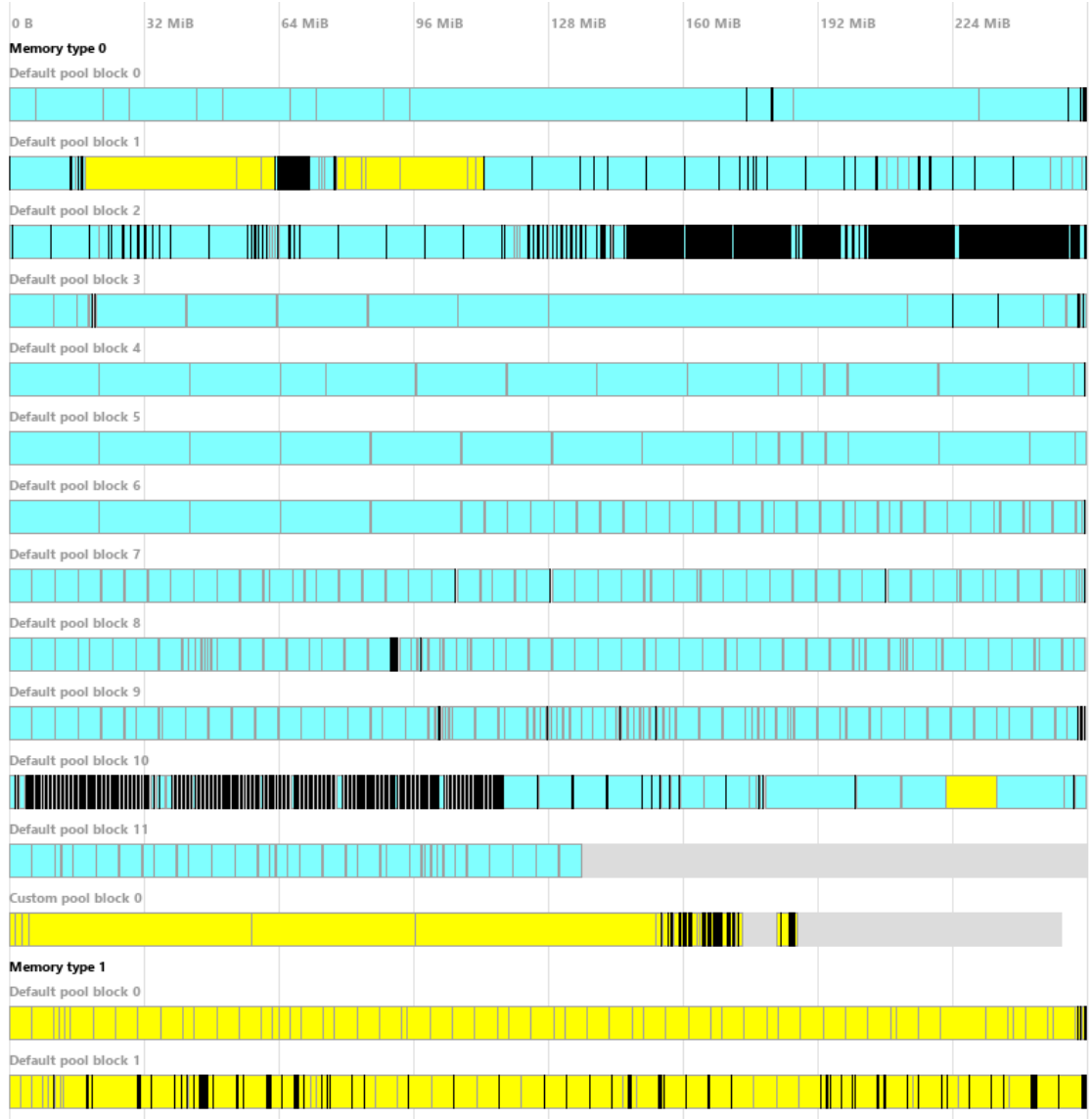
```
VkBufferCreateInfo bufferInfo = { VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO };
bufferInfo.size = 65536;
bufferInfo.usage =
VK_BUFFER_USAGE_VERTEX_BUFFER_BIT | VK_BUFFER_USAGE_TRANSFER_DST_BIT;

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

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

# VmaDumpVis.py

- Auxiliary tool that visualizes JSON dump from Vulkan Memory Allocator.
- Light gray - unused by any allocation.
- Yellow - buffer.
- Aqua - image with TILING_OPTIMAL.
- Green - image with TILING_LINEAR.
- Black - one or more allocations that are too small to visualize.



# Conclusions

- **Vulkan® is lower level, 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.**
- **There are open-source libraries that can help you with this task.**

# References

- Vulkan Memory Allocator (AMD)
  https://github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator
- Vulkan Device Memory, Timothy Lottes, GPUOpen
  https://gpuopen.com/vulkan-device-memory/
- Vulkan Memory Management, Chris Hebert, NVIDIA Developer
  https://developer.nvidia.com/vulkan-memory-management
- What's your Vulkan Memory Type?, Mathias Schott & Jeff Bolz, NVIDIA Developer
  https://developer.nvidia.com/what%E2%80%99s-your-vulkan-memory-type
- Vulkan Hardware Database, Sascha Willems
  http://vulkan.gpuinfo.org/

# Thank you

- Adam Sawicki(@Reg___)
- Mike Smith
- Dominik Baumeister
- Lou Kramer (@lou_auroyup)
- Matthäus G. Chajdas (@NIV_Anteru)
- Rys Sommefeldt (@ryszu)
- Timothy Lottes (@TimothyLottes)

# Disclaimer & Attribution

DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

ATTRIBUTION


AMD

# Backup

# Cache control

- **Any memory type that doesn't have HOST_COHERENT flag needs manual cache control:**
  - vkInvalidateMappedMemoryRanges before read on CPU
  - vkFlushMappedMemoryRanges after write on CPU
- **In practice, all PC GPU vendors (AMD, Intel, NVIDIA) support HOST_COHERENT on every memory type that is HOST_VISIBLE.**
  - No need to worry about it on current Windows PCs.

# Dedicated allocation

- **Some resources may benefit from having their own, dedicated memory block instead of region suballocated from a bigger block.**
  - Driver may use additional optimizations.
  - VK_KHR_dedicated_allocation
- **Use the get_memory_requirements2 extension to query the driver if it recommends or requires using a dedicated allocation.**
- **Use for:**
  - Render targets, depth-stencil, UAV.
  - Shared resources. (required)
  - Very large buffers and images. (dozens of MiB)
  - Large allocations that may need to be resized (freed and reallocated) at run-time.

# MISC

- **Avoid VK_IMAGE_LAYOUT_GENERAL. Always transition image to appropriate VK_IMAGE_LAYOUT_*_OPTIMAL.**
  - Exception: storage image requires VK_IMAGE_LAYOUT_GENERAL.
- **Memory requirements (e.g. size) can vary for different resources (e.g. images) even when created with same parameters (format, width, height, mip levels etc.)**
  - It really happens in the wild. Be prepared for that. Don’t cache result. Query each resource for requirements.