Memory Management in Vulkan™ and DX12

Adam Sawicki
Developer Technology Engineer, AMD

Agenda

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

GDC

Introduction

The challenge

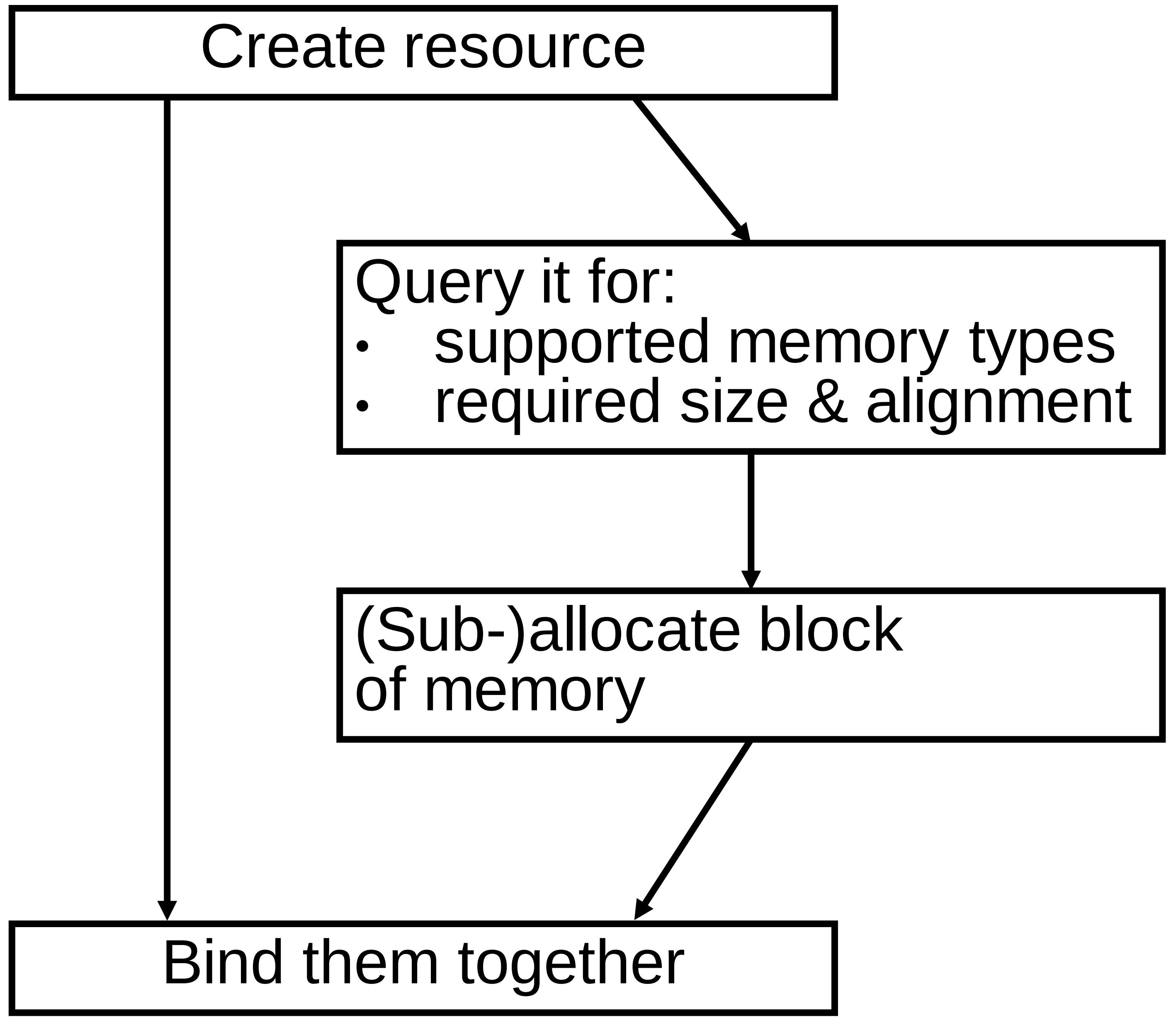
- Previous generation APIs (OpenGL™, DirectX® 11) manage memory automatically.
  - You create a resource (e.g. texture, constant buffer), backing memory is allocated automatically.

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

- New APIs (Vulkan™, DirectX® 12) are lower level, require explicit memory management.

The challenge

It is now your responsibility to:



Advantages

Explicit memory management makes it possible to:

- better manage memory
- better optimize for specific platforms
- alias (overlap) transient resources

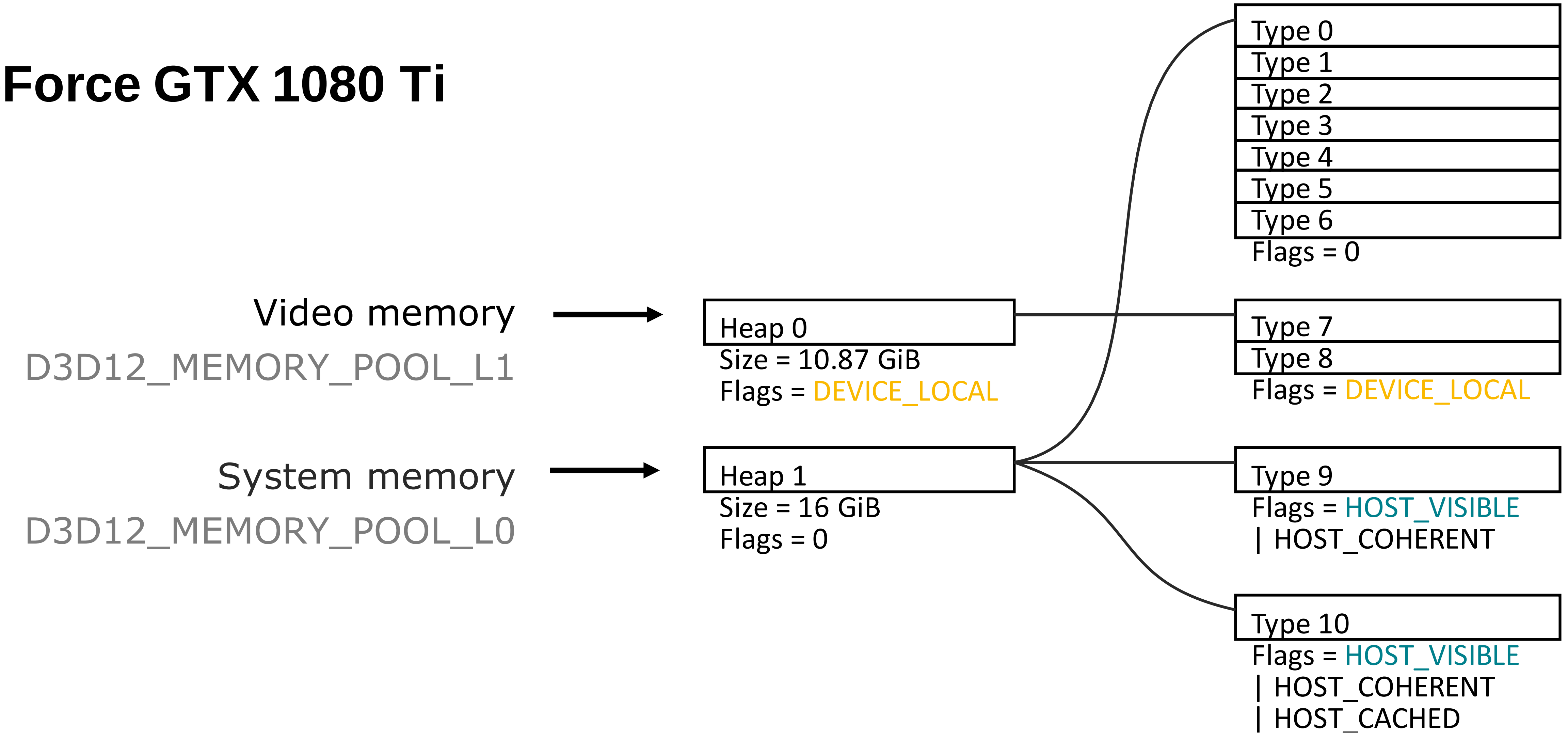
GDC®

Memory Types

UBM

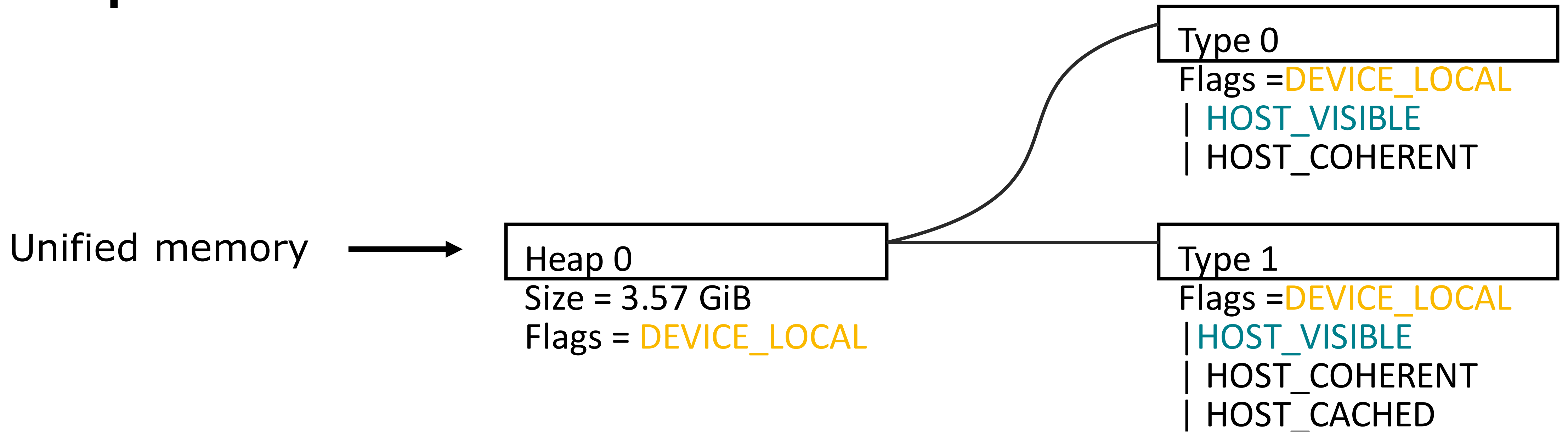
Memory types: NVIDIA

Example: **NVIDIA GeForce GTX 1080 Ti**

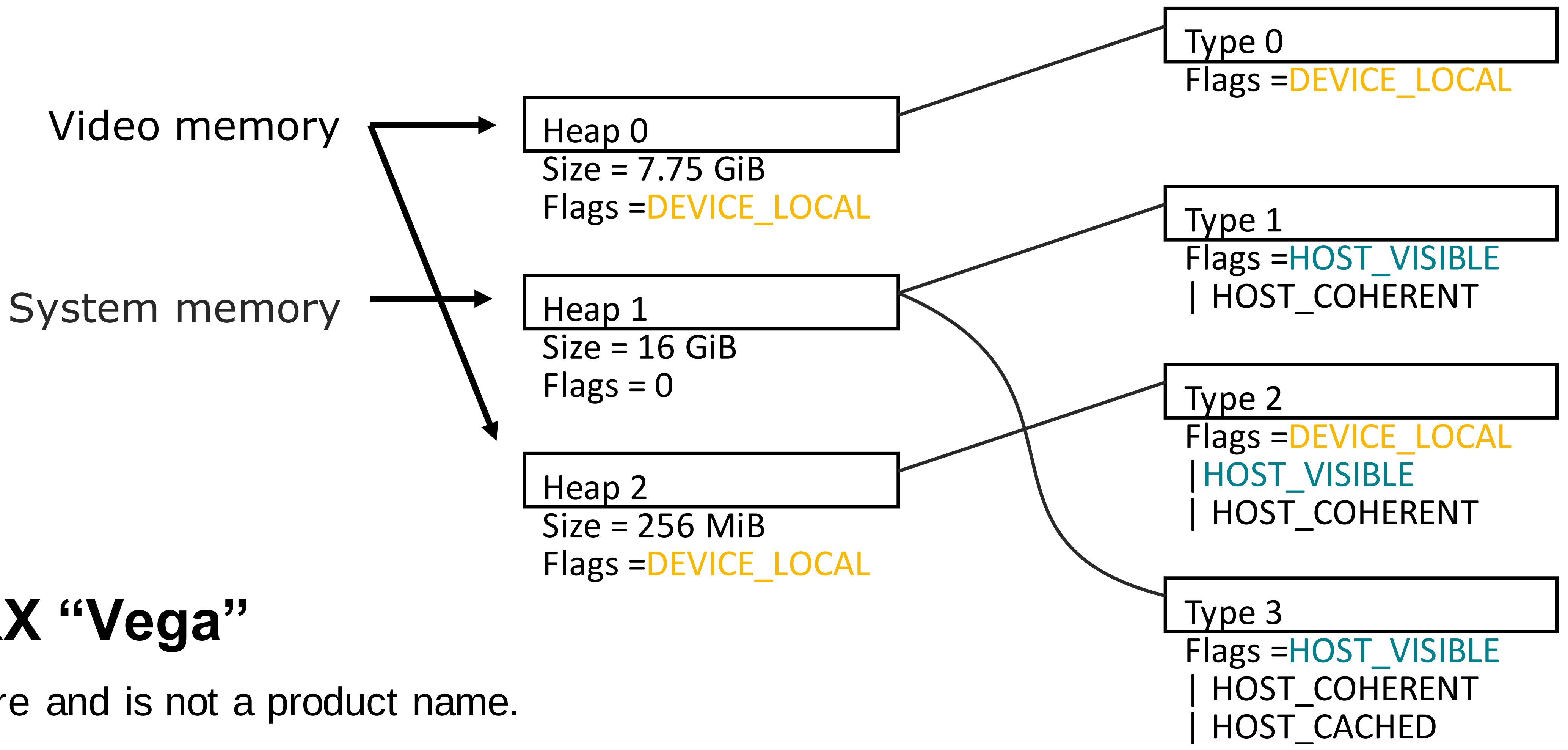


Memory types: Intel

Example: **Intel Iris Plus Graphics 640**



Memory types: AMD



Example: **AMD Radeon™ RX “Vega”**

Vega is a codename for AMD architecture and is not a product name.

DEVICE_LOCAL

D3D12_HEAP_TYPE_DEFAULT

- **Video memory.** Fast access from GPU.
- No direct access from CPU – mapping not possible.

DEVICE_LOCAL

- 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

D3D12_HEAP_TYPE_UPLOAD

- **System memory.** Accessible to CPU – mapping possible.
- Uncached. Writes may be write-combined.
- Access from GPU possible but slow
  Across PCIe® bus, reads cached on GPU.

HOST_VISIBLE

- 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 fine (always measure!)
  Cache on GPU may help.
- Large data read by GPU – place here as last resort.
- Large data written and read by GPU – shouldn't ever be here.

DEVICE_LOCAL + HOST_VISIBLE

- **Special pool of video memory.**
- Exposed on AMD only. 256 MiB.
- Fast access from GPU.
- Accessible to CPU – mapping possible.
  - Written directly to video memory.
  - Writes may be write-combined.
  - Uncached. Don’t read from it.

DEVICE_LOCAL + HOST_VISIBLE

- Good for resources updated frequently by CPU (dynamic), read by GPU.
- Direct access by both CPU and GPU – you don't need to do explicit transfer.
- Use as fallback if DEVICE_LOCAL is small and oversubscribed.

HOST_VISIBLE + HOST_CACHED

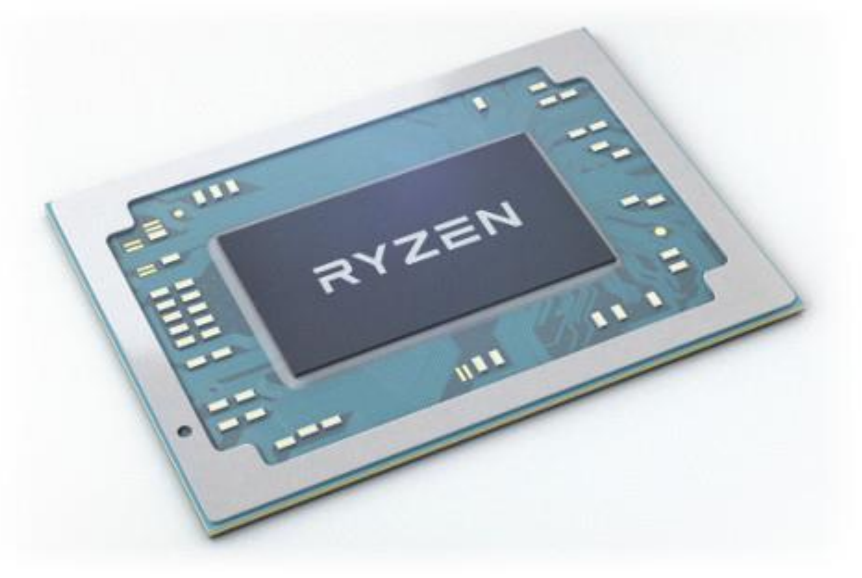
D3D12_HEAP_TYPE_READBACK

- **System memory**
- CPU reads and writes cached (write-back).
- GPU access through PCIe.
  GPU reads snoop CPU cache.

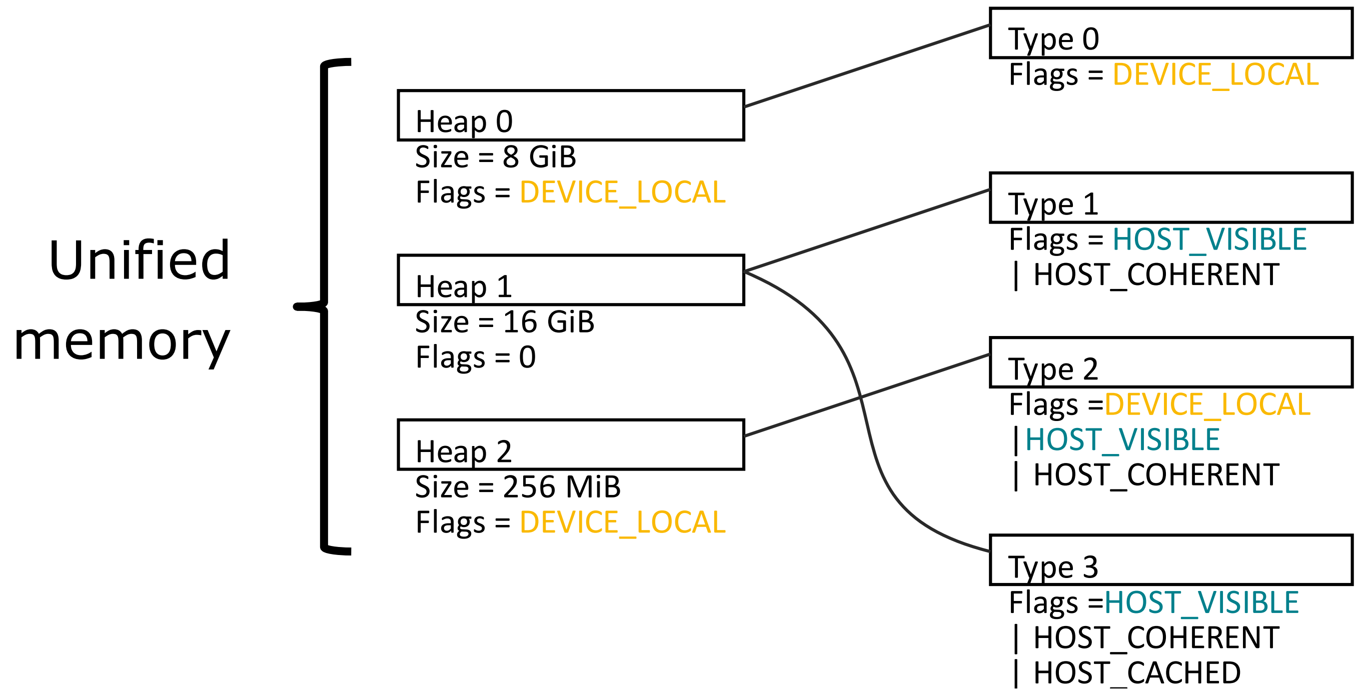
HOST_VISIBLE + HOST_CACHED

- Good for resources written by GPU, read by CPU – results of computations.
- Direct access by both CPU and GPU – you don't 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 GPU.
- Reported DEVICE_LOCAL heap can be any size, 0 B … few GiB.

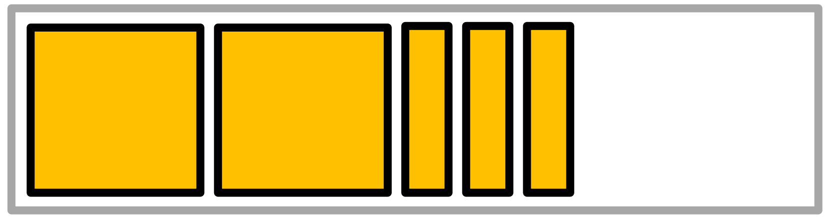


Memory types: AMD APU

- Memory is really unified – all heaps are equally fast.
- If you detect integrated graphics: VK_PHYSICAL_DEVICE_TYPE_INTEGRATED_GPU:
  - Count size of all memory heaps together.
  - Put your resources in whatever memory type meets your requirements.

Tips & Tricks

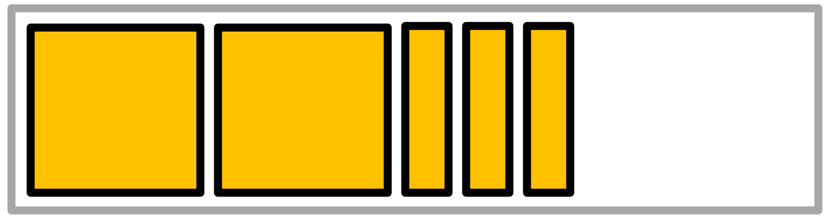
Suballocation



- Don’t allocate separate memory block for each resource (DX12: CreateCommittedResource).
  - small limit on maximum number of allocations (e.g. 4096)
  - allocation is 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.

Suballocation



Allocate bigger blocks and sub-allocate ranges for your resources (DX12: CreatePlacedResource).

- 256 MiB is good default block size.
- For heaps <= 1 GiB use smaller blocks (e.g. heap size / 8).

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.

Over-commitment – Vulkan™

- Size of memory heap: VkMemoryHeap::size.
- No known way to manually control residency of memory blocks.
- No known way to query amount of used or available free memory.

  You need to estimate.

Over-commitment – Vulkan™

- Sum up the size of your VkDeviceMemory blocks.
- Remember about implicit resources that also occupy memory.
- Leave some margin free (e.g.
  20% of DEVICE_LOCAL,
  33% of DEVICE_LOCAL + HOST_VISIBLE).

Over-commitment – DX12

- Size of memory types:
  DXGI_ADAPTER_DESC
- Current usage and available budget for your program:
  DXGI_QUERY_VIDEO_MEMORY_INFO
- You can register for notifications:
  IDXGIAdapter3::RegisterVideoMemoryBudgetChangeNotificationEvent

Over-commitment – DX12

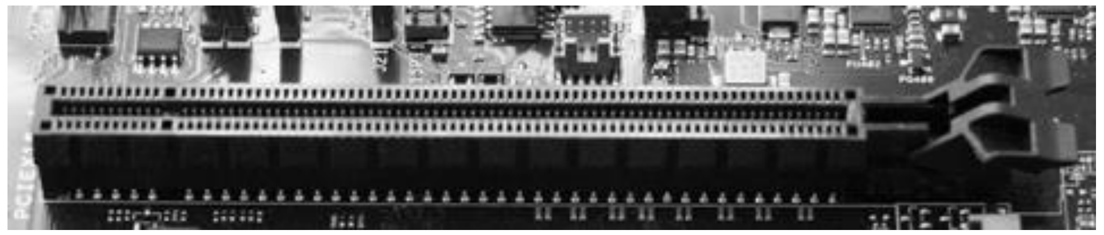
- You can page allocated blocks (heaps) in and out of video memory:
  ID3D12Device::Evict, MakeResident,
  ID3D12Device3::EnqueueMakeResident
- You can set residency priorities to resources:
  ID3D12Device1::SetResidencyPriority
- You can inform DX12 about minimum required memory:
  IDXGIAdapter3::SetVideoMemoryReservation

Mapping

```
void* ptr;
```

- Having entire memory block persistently mapped is generally OK.
  You don't need to unmap before using on GPU.
- Exceptions:
  - **Vulkan™, 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



- Copy queue is designed for efficient transfer via **PCIe**
  - Use it in parallel with 3D rendering, even asynchronously to rendering frames. Good for texture streaming.
  - Use it also for defragmentation of GPU memory in the background.
  - Do your transfers long before the data is needed on graphics queue.
- **GPU to** (the same) **GPU** copies are much faster on graphics queue.
  - Use it if graphics queue needs to wait for transfer result anyway.

GDC®

Libraries

Direct3D Residency Starter Library

- https://github.com/Microsoft/DirectX-Graphics-Samples/tree/master/Libraries/D3DX12Residency
- Library from Microsoft®
  - MIT license
  - easy to integrate – single C++ header
- manages residency of DX12 heaps / committed resources
- implements essentially the same memory management behavior that a DX11 app would get

Direct3D Residency Starter Library

Instead of just calling ExecuteCommandLists:

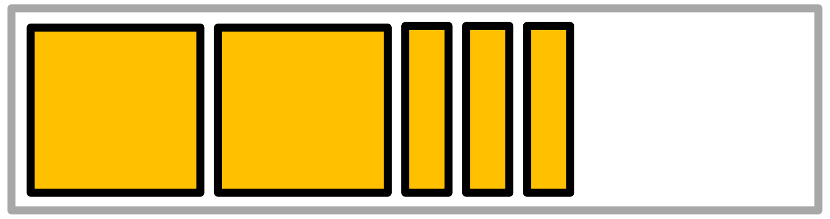
- you pass command lists to be executed together with a list of resources they use
- the library:
  - queries GXGI for memory budget
  - calls Evict for least recently used resources
  - calls MakeResident for resources that are going to be used
  - calls ExecuteCommandLists

Vulkan Memory Allocator

- https://gpuopen.com/gaming-product/vulkan-memory-allocator/
- 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.
- Releasing final version 2.0.0 now!

See also Dustin Land et al. talk "Getting Explicit: How Hard is Vulkan Really?"

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);
```

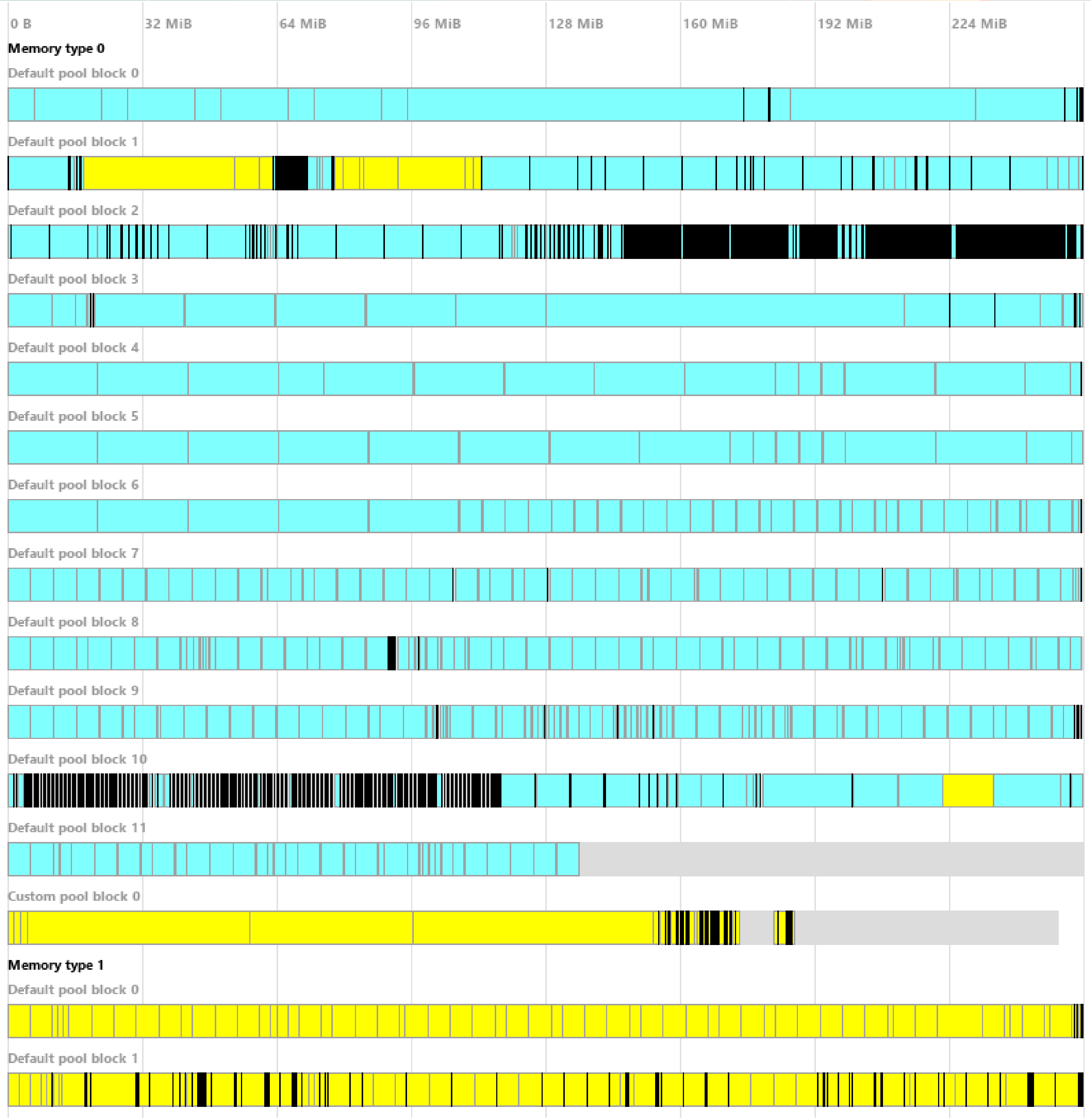
VmaDumpVis.py

Auxiliary tool that visualizes JSON dump from Vulkan Memory Allocator.

```
python VmaDumpVis.py -o Image.png VmaDump.json
```

Released just now!

🟦 Image
🟨 Buffer



GDC

Conclusions

Conclusions

- New graphics APIs (Vulkan™, DirectX 12) require more 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://gpuopen.com/gaming-product/vulkan-memory-allocator/
- Direct3D Residency Starter Library (Microsoft)
  https://github.com/Microsoft/DirectX-Graphics-Samples/tree/master/Libraries/D3DX12Residency
- 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
- DX12 Do's And Don'ts, NVIDIA Developer
  https://developer.nvidia.com/dx12-dos-and-donts
- Vulkan Hardware Database, Sascha Willems
  http://vulkan.gpuinfo.org/

Thank you

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

Disclaimer & Attribution

ATTRIBUTION

AMD

GDC

Backup

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.
- Use for:
  - Render targets, depth-stencil, UAV
  - Very large buffers and images (dozens of MiB)
  - Large allocations that may need to be resized (freed and reallocated) at run-time.

Dedicated allocation

- DX12:
  ID3D12Device::CreateCommittedResource function
- Vulkan™:
  VK_KHR_dedicated_allocation extension

Cache control

- Vulkan™ : 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.

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.

Miscellaneous

- Vulkan™: 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.
- Don't use images with TILING_LINEAR (DX12: LAYOUT_ROW_MAJOR) unless you have to.
  - Use TILING_OPTIMAL (DX12: LAYOUT_UKNOWN).
  - Copy images from/to buffers.

Miscellaneous

- Avoid VK_IMAGE_LAYOUT_GENERAL (D3D12: D3D12_RESOURCE_STATE_GENERIC_READ). Always transition image to appropriate VK_IMAGE_LAYOUT_*_OPTIMAL.
- Avoid VK_SHARING_MODE_CONCURRENT on render target textures. It disables DCC compression.
  - Prefer VK_SHARING_MODE_EXCLUSIVE and do explicit queue family ownership transfer barriers.
- Avoid VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT.
  - If you really need different formats e.g. to interpret as linear/sRGB, use it together with VK_KHR_image_format_list extension.