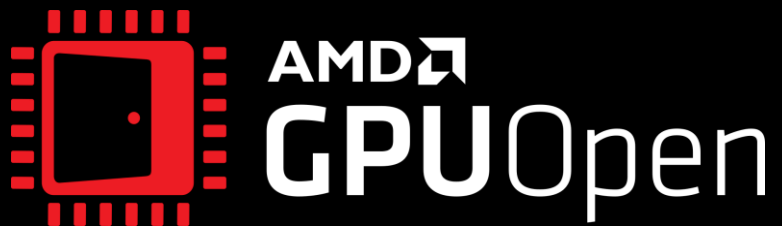




EFFICIENT USE OF GPU MEMORY IN MODERN GAMES

ADAM SAWICKI

DEVELOPER TECHNOLOGY ENGINEER, AMD



AGENDA

- Types of memory
- A case study
- Resizable BAR, Smart Access Memory
- Performance tips
- Summary

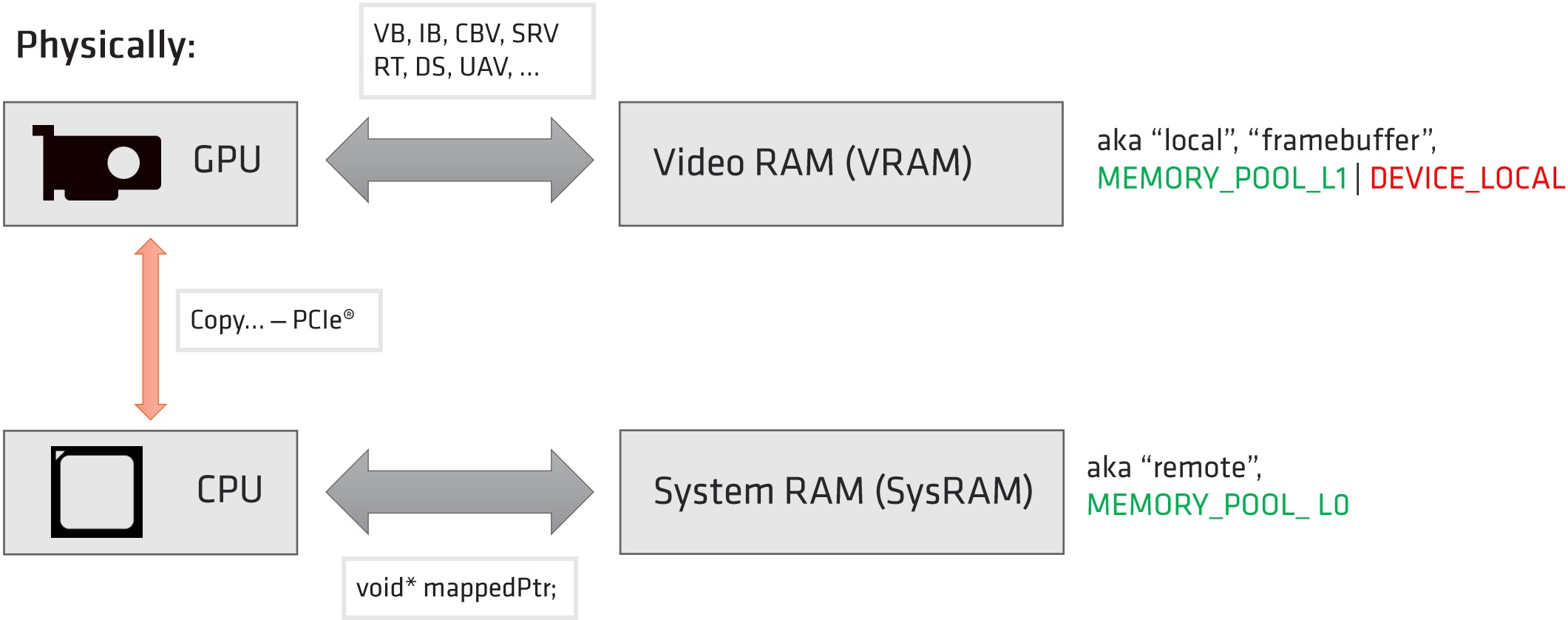
Talking about desktop PC only.

It will be low level...

TYPES OF MEMORY



Physically:



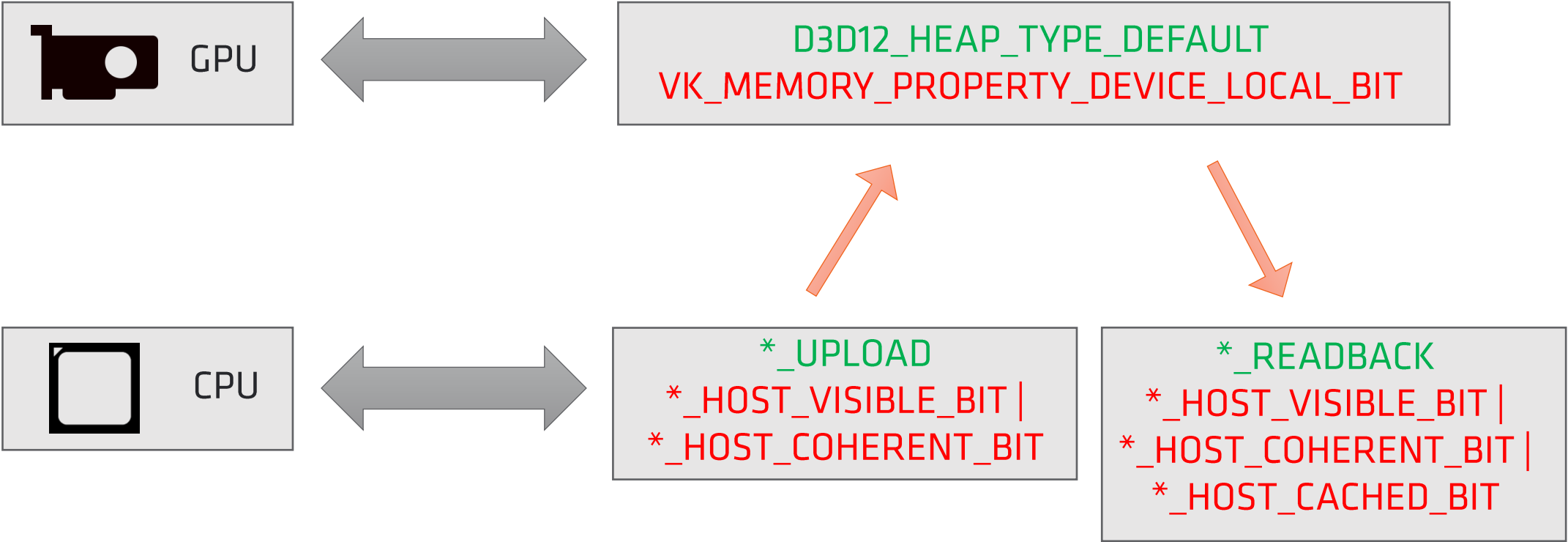
See also [1], [2]



TYPES OF MEMORY



Logically:

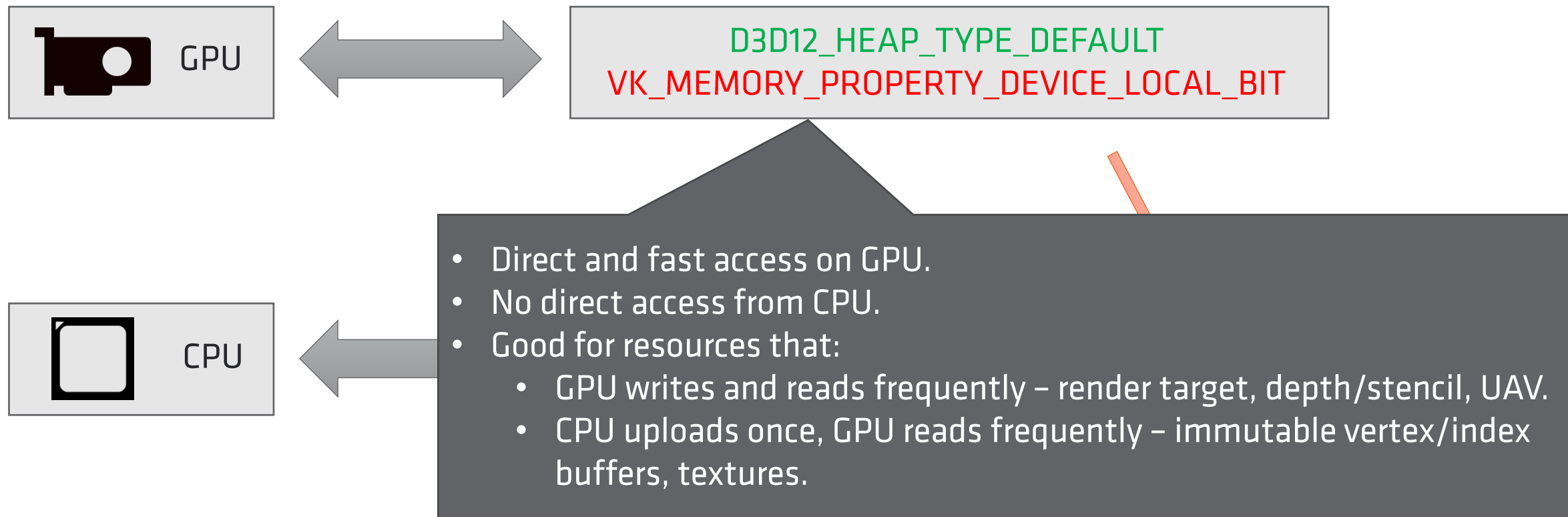


Direct3D® 12 | Vulkan®

TYPES OF MEMORY



Logically:

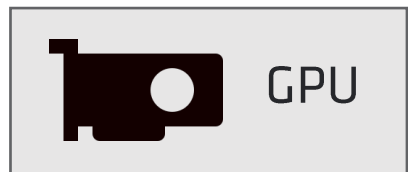


Direct3D® 12 | Vulkan®

TYPES OF MEMORY



Logically:



- Direct access on CPU (mapping), cached.
- Good for resources:
 - Copied from or written directly by GPU, read by CPU.

(A less common case, will not discuss here.)



*_UPLOAD
*_HOST_VISIBLE_BIT |
*_HOST_COHERENT_BIT

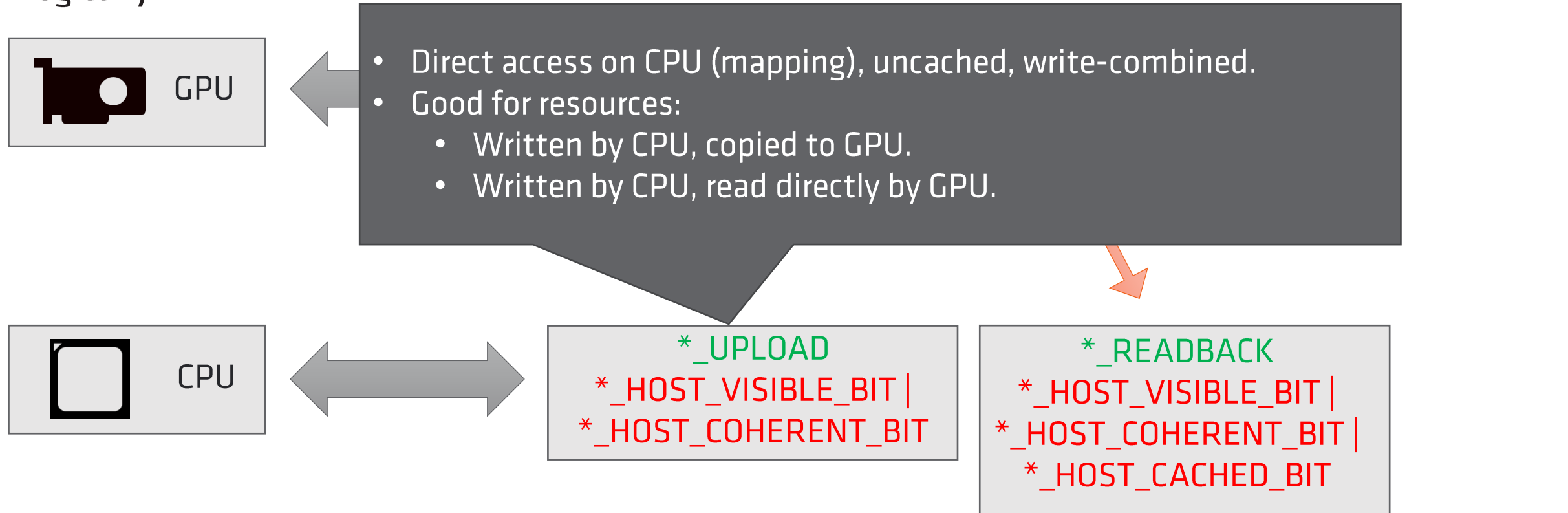
*_READBACK
*_HOST_VISIBLE_BIT |
*_HOST_COHERENT_BIT |
*_HOST_CACHED_BIT

Direct3D® 12 | Vulkan®

TYPES OF MEMORY



Logically:



Direct3D® 12 | Vulkan®

UPLOAD HEAP

Uncached & write-combined means:

Fast:



Sequential writes

- `mappedPtr[i] = srcData[i];`



Copy to it

- `memcpy(mappedPtr, srcData, ...);`

Slow:



Scattered writes

- `mappedPtr[indirect[i]] = x;`



Reads

- `y = mappedPtr[i];`



Incl. implicit reads

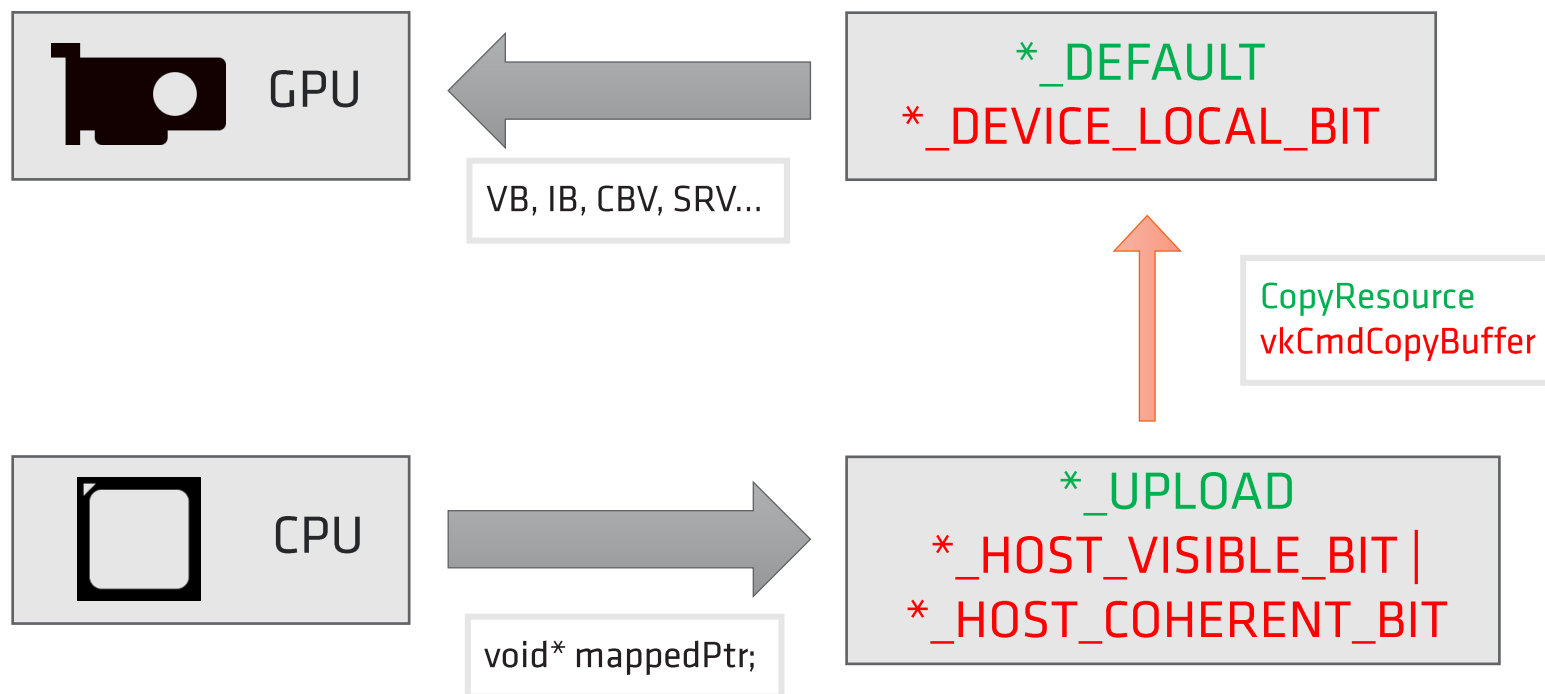
- `mappedPtr[i] += z;`

Pro tip: Align start of your data to 64 B.

WAYS TO UPLOAD DATA



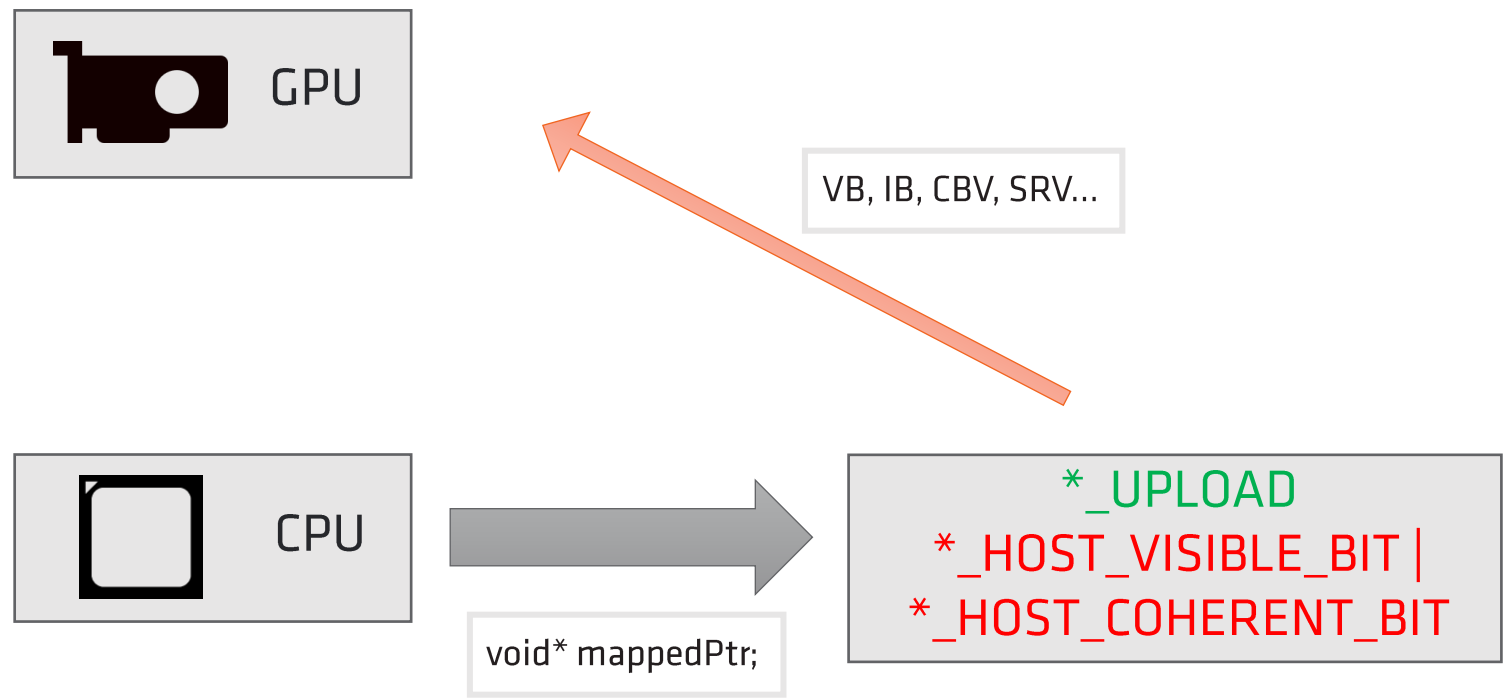
Method 1. CPU writes to UPLOAD → GPU executes copy command → GPU reads from DEFAULT



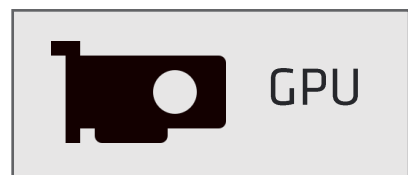
WAYS TO UPLOAD DATA



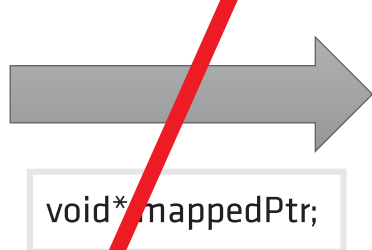
Method 2. CPU writes to UPLOAD → GPU reads from UPLOAD



WAYS TO UPLOAD DATA



VB, IB, CBV, SRV...



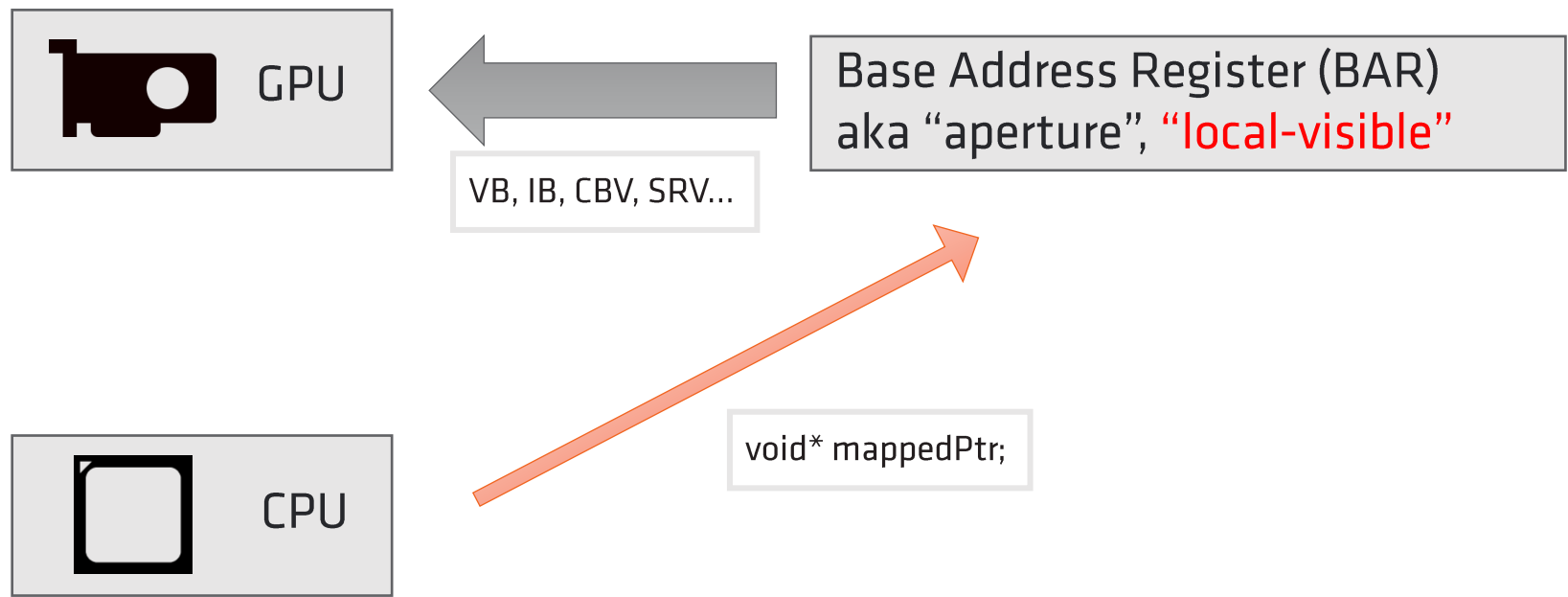
*_UPLOAD
*_HOST_VISIBLE_BIT |
*_HOST_COHERENT_BIT

- Good for buffers.
- Textures|images better be in opaque GPU-specific format to use optimized swizzling/compression:
D3D12_TEXTURE_LAYOUT_UNKNOWN |
VK_IMAGE_TILING_OPTIMAL → need to do buffer-image copy (see also [3]).

BAR



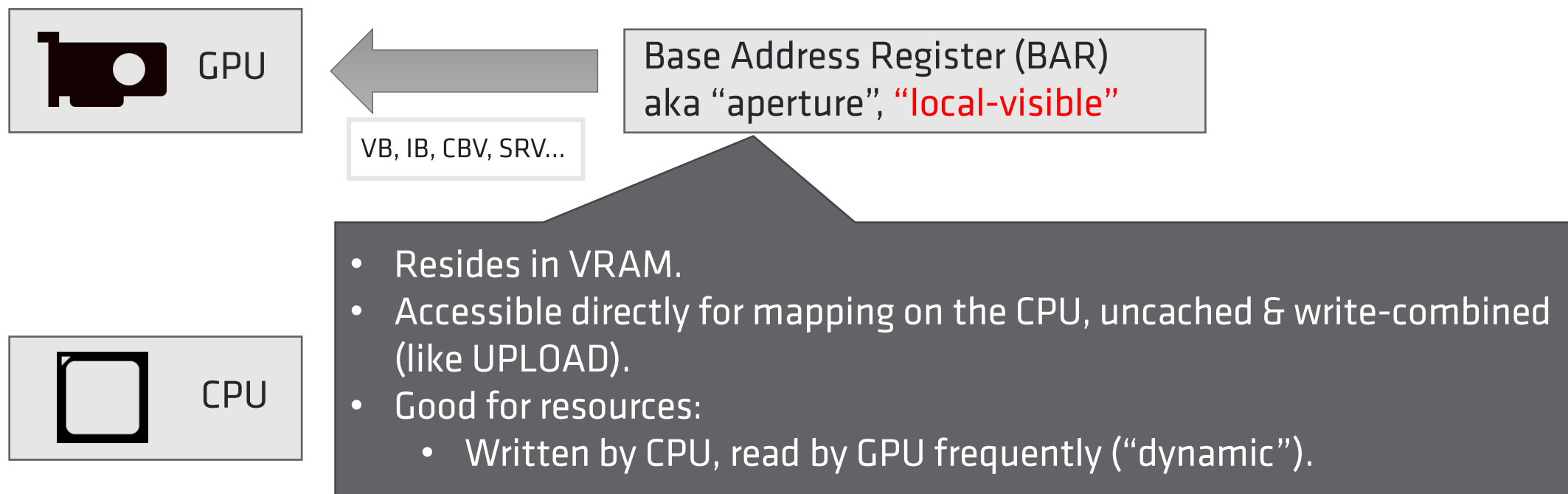
Method 3. There is a 4th type of memory:



BAR



Method 3. There is a 4th type of memory:



BAR

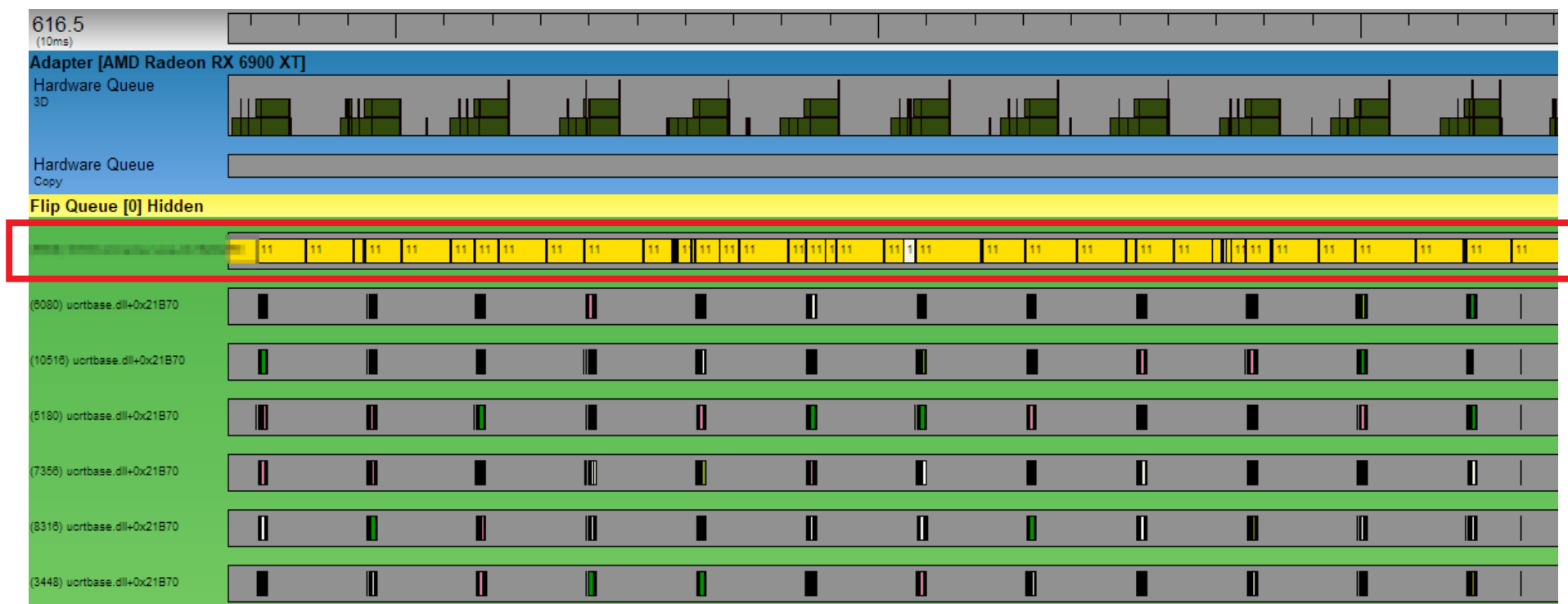


- `VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT | *_HOST_VISIBLE_BIT | *_HOST_COHERENT_BIT`
 - Not always available – availability depends on GPU and driver.
- D3D12: not exposed as of today.

A CASE STUDY



- A game using DX11.
- Running on Radeon RX 6900 XT, 1920x1080 at 38 FPS. 🐢 🐢 🐢
- GPUView showed the game was CPU-bound in the renderer DLL.



A CASE STUDY



- AMD μ Prof showed hotspots as 2 renderer functions that just read & write some pointer...

AMDuProf - [C:/Users/Adam Saw...-26-2021_14-33-06.db]

PROFILE SUMMARY ANALYZE SOURCES

CHWShader_D3D::GetSPIBufferData(union _m128 * const, struct SRenderItem *, int, int)

Filters

PID: [] TID: TID [16088] View timer-based profile

Address	Line	Assembly	CPU_TIME (s)
0x7ff929a19265		add rsp,000001c8h	0.00
0x7ff929a1926c		pop r13	0.01
0x7ff929a1926e		pop rdi	
0x7ff929a1926f		pop rsi	
0x7ff929a19270		pop rbp	
0x7ff929a19271		retq	
0x7ff929a19272		mov rcx,[0000000180403b10h] 0x180403b10	
0x7ff929a19279		mov [rsp+000001c0h],rbx	4.33
0x7ff929a19281		add rcx,0000e950h	0.01
0x7ff929a19288		cmp [000000018038c2b0h],rsi 0x18038c2b0	2.29
0x7ff929a1928f		mov [rsp+000001b8h],r12	0.01
0x7ff929a19297		mov [rsp+000001b0h],r14	0.00
0x7ff929a1929f		mov [rsp+000001a8h],r15	3.28
0x7ff929a192a7		jnz 00000001800e92cah 0x1800e92ca	
0x7ff929a192a9		xor r9d,r9d	

A CASE STUDY



- RenderDoc showed before every draw call they map a dynamic VB and CB...

1696	DrawIndexed(5922)	23.76
1704	DrawIndexed(5922)	28.32
1721	DrawIndexed(5922)	24.64
1728	DrawIndexed(5922)	11.16
1738	DrawIndexed(5922)	24.24
1745	DrawIndexed(5922)	10.80
1755	DrawIndexed(5922)	6.48
1762	DrawIndexed(5922)	22.68
1772	DrawIndexed(5922)	19.32
1785	DrawIndexedInstanced(5922, 3)	29.12
1807	DrawIndexedInstanced(2958, 12)	78.12
1825	DrawIndexedInstanced(2958, 33)	39.64
1847	DrawIndexedInstanced(1479, 3)	36.12
1865	DrawIndexedInstanced(1479, 25)	40.64
1880	DrawIndexed(1479)	18.92
1895	DrawIndexedInstanced(1479, 25)	26.40
1910	DrawIndexed(1479)	20.22

API Inspector	Event
> 1763	ID3D11DeviceContext::Map
> 1764	ID3D11DeviceContext::Map
> 1765	ID3D11DeviceContext::Unmap
> 1766	ID3D11DeviceContext::Unmap
> 1767	ID3D11DeviceContext::VSSetConstantBuffers
> 1768	ID3D11DeviceContext::VSSetConstantBuffers
> 1769	ID3D11DeviceContext::PSSetConstantBuffers
> 1770	ID3D11DeviceContext::PSSetConstantBuffers
> 1771	ID3D11DeviceContext::PSSetSamplers
> 1772	ID3D11DeviceContext::DrawIndexed


Resource Initialisation Parameters	
Parameter	Value
✓ ID3D11Device...	
✓ pDesc	D3D11_BUFFER_DESC()
ByteWidth	256
Usage	D3D11_USAGE_DYNAMIC
BindFlags	D3D11_BIND_CONSTANT_BUFFER
CPUAccess...	D3D11_CPU_ACCESS_WRITE
MiscFlags	0
StructureB...	0
pInitialData	NULL
pBuffer	Buffer 26709
InitialData	(256 bytes)
InitialDataLe...	256

A CASE STUDY



- Did they read from a mapped pointer?
- They did! By accident...

```
ctx->Map(buf, 0, D3D11_MAP_WRITE_DISCARD, 0, &mapped);  
Vector* pDst = (Vector*)mapped->pData;  
pDst[0] = ...  
pDst[1] = ...  
...  
if(...) {  
    pDst[0] += ...  
    pDst[1] += ...  
}  
ctx->Unmap(buf, 0);
```

A small black silhouette of a turtle, facing right, positioned to the right of the code block.

A CASE STUDY



Why so slow?

- This memory is uncached & write-combined → should be only written, never read.
- Microsoft® explicitly warns about it in the docs of ID3D11DeviceContext::Map.
- Our driver decided to put a dynamic DX11 buffer in BAR (VRAM) instead of SysRAM → making this bad access pattern even slower. 🐢 🐢

A CASE STUDY



The solution:

```
alignas(64) Vector src[N];  
src[0] = ...  
src[1] = ...  
if(...) {  
    src[0] += ...  
    src[1] += ...  
}  
ctx->Map(buf, 0, D3D11_MAP_WRITE_DISCARD, 0, &mapped);  
memcpy(mapped->pData, src, N * sizeof(Vector));  
ctx->Unmap(buf, 0);
```

Result: final game running at 4K at 80 FPS. 

REBAR & SAM

- Traditional BAR: fixed size 256 MB.
- Resizable BAR (ReBAR): makes entire VRAM CPU-visible.
- Smart Access Memory (SAM): AMD technology that utilizes ReBAR to boost performance in games [4].

SMART ACCESS MEMORY

How to ensure compatibility?

ReBAR is a low-level feature...

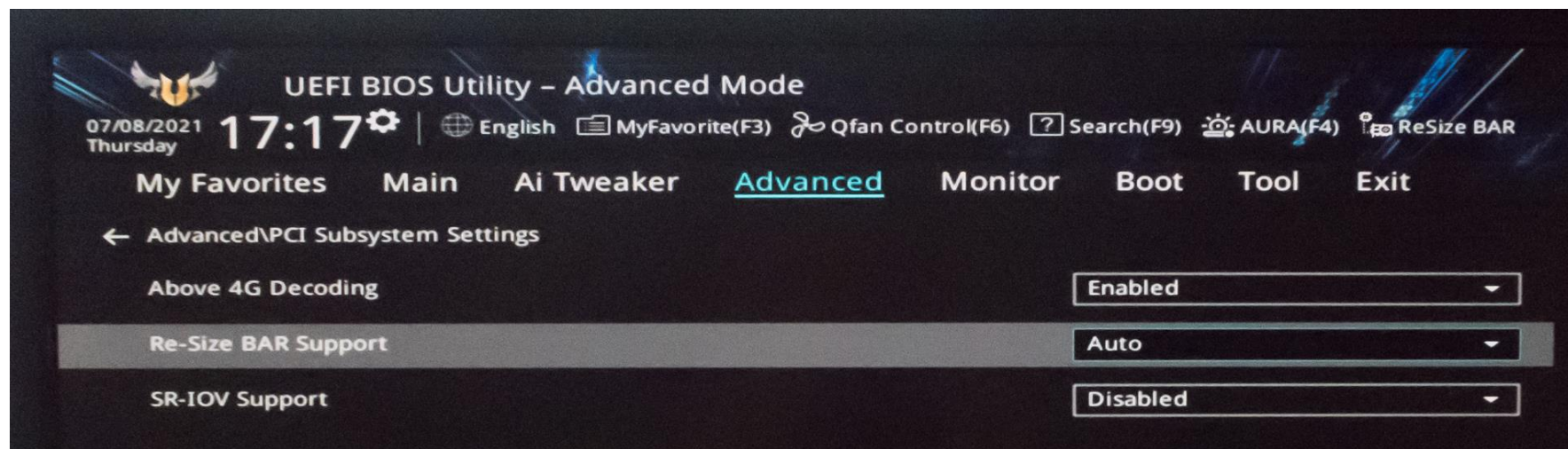
- Ensure compatible hardware: motherboard, CPU, GPU.
 - E.g., Ryzen 5000 series processor, Radeon 6000 graphics card.
- Update motherboard BIOS.
- Use Windows® 10 with latest updates.
- Update graphics driver.

SMART ACCESS MEMORY

How to enable?

Enter BIOS, enable: Advanced → PCI Subsystem Settings →

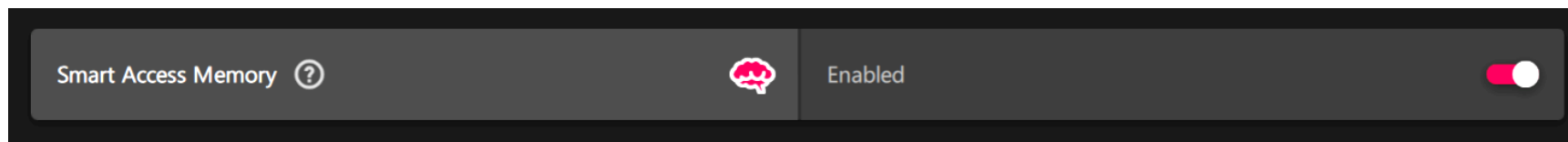
- Above 4K Decoding
- Re-Size BAR Support



SMART ACCESS MEMORY

How to check it is working?

- AMD Radeon Software → Performance → Tuning → Smart Access Memory



SMART ACCESS MEMORY

How to use it?

- Vulkan: Detect and use >256 MB of *_DEVICE_LOCAL_BIT | *_HOST_VISIBLE_BIT memory.
- D3D12: No direct access at the moment.
 - You can just prepare your game to work fast in any case...

REBAR IN VULKAN

Radeon RX 6900 XT, driver 21.3.2, ReBAR = Off

heapCount=3, typeCount=8

Heap 0: 16894656512 B (15.73 GB) DEVICE_LOCAL, MULTI_INSTANCE

heapBudget = 16072161280 B (14.97 GB)

heapUsage = 22675456 B (21.62 MB)

Type 0: DEVICE_LOCAL

Type 4: DEVICE_LOCAL, DEVICE_COHERENT (AMD), DEVICE_UNCACHED (AMD)

Heap 1: 25454182400 B (23.71 GB)

heapBudget = 24918255616 B (23.21 GB)

heapUsage = 4526080 B (4.32 MB)

Type 1: HOST_VISIBLE, HOST_COHERENT

Type 3: HOST_VISIBLE, HOST_COHERENT, HOST_CACHED

Type 5: HOST_VISIBLE, HOST_COHERENT, DEVICE_COHERENT (AMD), DEVICE_UNCACHED (AMD)

Type 7: HOST_VISIBLE, HOST_COHERENT, HOST_CACHED, DEVICE_COHERENT (AMD), DEVICE_UNCACHED (AMD)

Heap 2: 268435456 B (256.00 MB) DEVICE_LOCAL, MULTI_INSTANCE

heapBudget = 255367008 B (243.54 MB)

heapUsage = 0 B (0)

Type 2: DEVICE_LOCAL, HOST_VISIBLE, HOST_COHERENT

Type 6: DEVICE_LOCAL, HOST_VISIBLE, HOST_COHERENT, DEVICE_COHERENT (AMD), DEVICE_UNCACHED (AMD)

← VRAM

← SysRAM

← BAR

REBAR IN VULKAN

Radeon RX 6900 XT, driver 21.3.2, ReBAR = On

heapCount=2, typeCount=8

Heap 0: 25454182400 B (23.71 GB)

heapBudget = 24918255616 B (23.21 GB)

heapUsage = 4526080 B (4.32 MB)

Type 1: HOST_VISIBLE, HOST_COHERENT

Type 3: HOST_VISIBLE, HOST_COHERENT, HOST_CACHED

Type 5: HOST_VISIBLE, HOST_COHERENT, DEVICE_COHERENT (AMD), DEVICE_UNCACHED (AMD)

Type 7: HOST_VISIBLE, HOST_COHERENT, HOST_CACHED, DEVICE_COHERENT (AMD), DEVICE_UNCACHED (AMD)

Heap 1: 17163091968 B (15.98 GB) DEVICE_LOCAL, MULTI_INSTANCE

heapBudget = 16058044416 B (14.96 GB)

heapUsage = 0 B (0)

Type 0: DEVICE_LOCAL

Type 2: DEVICE_LOCAL, HOST_VISIBLE, HOST_COHERENT

Type 4: DEVICE_LOCAL, DEVICE_COHERENT (AMD), DEVICE_UNCACHED (AMD)

Type 6: DEVICE_LOCAL, HOST_VISIBLE, HOST_COHERENT, DEVICE_COHERENT (AMD), DEVICE_UNCACHED (AMD)

← SysRAM

← ReBAR

PERFORMANCE TIPS



CPU writes to BAR

- If your resource ends up in VRAM (BAR) not SysRAM (UPLOAD), bad CPU access patterns become many times slower! 🐢 🐢 🐢
 - Same recommendations apply: only write sequentially or use memcpy(). 🐇
- With PCIe 4.0, CPU writes to VRAM can be same order of magnitude as to SysRAM!
 - No need to be afraid of writing to BAR. 🐇

See more tips in [5], [6]

PERFORMANCE TIPS



Avoid overhead

- Overhead of each separate buffer/**texture**|**image** → use few large buffers instead of many small buffers, sub-allocate parts of them, use offsets to address your data.
 - Better to have ≥ 64 KB of meaningful data in a buffer.
- Overhead of each **Submit**|**Execute** and cross-queue synchronization using **semaphores**|**events** →
 - hide the latency by other concurrent work,
 - copy on the same queue where you use the data,
 - or avoid copy by reading data directly.
- Map/Unmap has some overhead → leaving buffer persistently mapped is correct and recommended.

PERFORMANCE TIPS



Allocation

Since UPLOAD resources can go to VRAM:

- Don't oversubscribe VRAM → don't allocate too much UPLOAD memory, only as much as necessary.
- "Preferred heap" is decided upon creation → allocate the most important resources first to increase chances they go to VRAM.
- "Residency" is managed for entire DeviceMemory block|heap → create big and important resources as separate allocations – committed|dedicated.

PERFORMANCE TIPS



Which queue to use for a copy: Graphics/Compute vs Copy|Transfer ?

- When copying from UPLOAD over PCIe, copy queue is a bit faster,
- but when copy source ends up in VRAM, copy queue is few times slower!



PERFORMANCE TIPS



- Use copy queue:
 - When copying in the background, asynchronously to render frames (e.g., texture streaming).
- Use graphics/compute queue:
 - When the results are needed immediately (e.g., dynamic data needed in this frame).
 - Maybe use async compute for background copies?
- Consider skipping the copy, CPU-write and GPU-read directly from UPLOAD or BAR
 - For small amounts of data (e.g., a **constant**|**uniform** buffer) or data to be read only once.

COMMERCIAL BREAK

- Vulkan Memory Allocator
 - <https://github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator>
- D3D12 Memory Allocator
 - <https://github.com/GPUOpen-LibrariesAndSDKs/D3D12MemoryAllocator>

See [7], [8]

- C++ libraries
 - Open source, MIT license
 - Work with any GPU and on any platform supporting Vulkan|DX12
1. Help to choose right memory type for a resource.
 2. Allocate large DeviceMemory blocks|heaps and sub-allocate parts of them for your resources.
 3. Hide boilerplate code inside convenient functions like CreateResource|vmaCreateBuffer.

SUMMARY

- Know available types of memory.
- Resizable BAR / Smart Access Memory.
- Be careful with CPU access to uncached mapped memory – only sequential writes or memcpy().
- 3 ways to upload data CPU → GPU.
- Plan carefully which queue to use for a copy (or none at all).

REFERENCES

1. S. Tovey, “Vulkan Memory Management” presented at Vulkanised, 2018.
2. A. Sawicki, “Memory management in Vulkan and DX12” presented at the Game Developers Conference, 2018.
3. C. Brennan, “Getting the Most Out of Delta Color Compression” GPUOpen.
<https://gpuopen.com/learn/dcc-overview/>
4. “AMD Smart Access Memory” AMD.
<https://www.amd.com/en/technologies/smart-access-memory>
5. O. Homburg, “How to get most out of Smart Access Memory (SAM)” GPUOpen.
<https://gpuopen.com/learn/get-the-most-out-of-smart-access-memory/>
6. “AMD RDNA 2 Performance Guide” GPUOpen.
<https://gpuopen.com/performance/>
7. “Vulkan Memory Allocator” GitHub.
<https://github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator>
8. “D3D12 Memory Allocator” GitHub.
<https://github.com/GPUOpen-LibrariesAndSDKs/D3D12MemoryAllocator>

THANKS!

- Bryan Turkelson
- Jonas Gustavsson
- Matthäus Chajdas
- Nicolas Thibieroz
- Oskar Homburg
- Paul Blinzer
- Steven Tovey

© 2021 Advanced Micro Devices, Inc. All rights reserved.

AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Microsoft is a registered trademark of Microsoft Corporation in the US and other jurisdictions. Windows is a registered trademark of Microsoft Corporation in the US and other jurisdictions. Vulkan and the Vulkan logo are registered trademarks of the Khronos Group Inc.

Disclaimer

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. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. 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.

THIS INFORMATION IS PROVIDED 'AS IS.' 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 NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, 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.

