Khronos Meetup 2018 - 05

DR. MATTHÄUS G. CHAJDAS, AMD

SHOUT OUT TO … UNTERNEHMERTUM!

AGENDA

- Matthäus – VK 1.1 and more
- Abil – OpenVX, NNEF, and machine learning.
- Fabian – SPEAR
- ~19:30 – food!
- ~22:00 – closing shop

VULKAN 1.1 AND BEYOND

- GDC 2018 – Vulkan 1.1 was released!
- Many quality-of-life improvements
  - Subgroup operations
  - Improved HLSL support
  - mGPU through device groups
  - More interop
  - More 16-bit
  - VR support (multiview)
  - A lot of maintenance
  - … and some more stuff

VULKAN 1.1 AND BEYOND

- Development also happening outside new core revisions
- Debugging & tools: `EXT_debug_utils`
- Portability & features: Bindless resources access
- Ecosystem: DXC

MAINTENANCE

- VK_KHR_maintenance_1 – 3 got merged
- The viewport origin can be set to top-left (through negative height) – this is now core 1.1 functionality!
- Images with per-view usage flags
  - Enables some common use cases with mixed formats
  - Example: SRGB view on a storage-use UNORM image resource
- Depth-stencil can be transitioned independently
- Uncompressed views of compressed data
  - Run-time compression is super common in games!
  - Mostly used to blend terrain and lookup later, in games from Battlefield 3 to World of Tanks

HLSL COMPATIBILITY

- Members in structures have different alignment requirements => Makes it sometimes impossible to use the same CPU structure
- This has been (mostly) resolved*

*arrrgh `float3` arrrgh!

16-BIT STORAGE, SHARING, AND MORE

- 16-bit storage
  - Load and store 16-bit data directly (i.e. 16-bit aligned)
  - Orthogonal to conversion to 32-bit in the shader core
- Multiview
  - Submit once, render to multiple views
  - VR, cube maps, etc.
- Faster descriptor updates
- External sharing
  - Interop with many other APIs
  - Shares both resources and sync primitives!

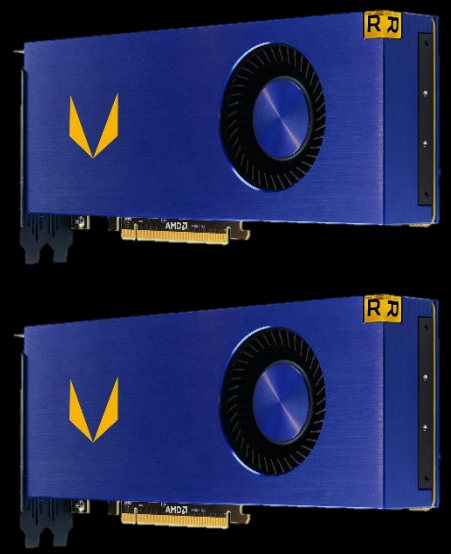
EXT_DEBUG_UTILS

- Previously, debug marker names never showed up in debug callbacks
  - The names were handled by `EXT_debug_marker`
  - The reporting was provided by `EXT_debug_report`
- `EXT_debug_utils` is the one extension to rule them all
- You should port your code **now***

* It’s hot off the presses so tools are catching up, but get yourself ready at least

DEVICE GROUPS

- Homogenous mGPU – 2 or more identical GPUs
- All the things you'd expect
  - Allocate on multiple devices
  - Bind across devices
  - Present across devices
- Don't forget this also works for compute
  - I.e. process data on N GPUs
  - Simulate on one GPU, render on the other

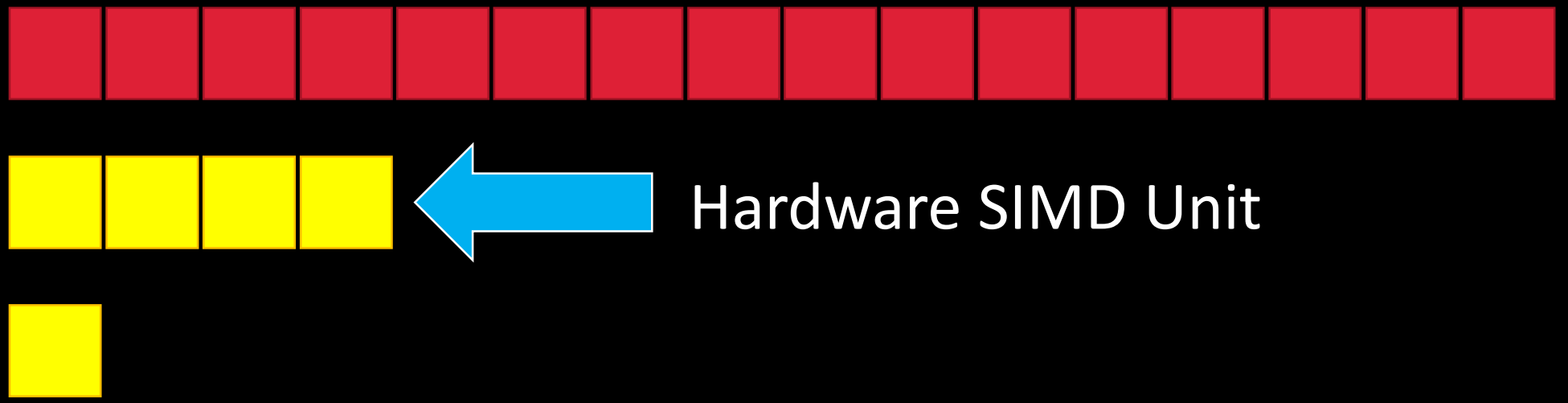


SUBGROUPS

- So far, Vulkan pretends that the only synchronization possible is between work group items
- In practice, the hardware has one execution level between a single work item and the whole work group
  - The hardware SIMD unit width (aka wavefront, warp, etc.)
  - Subgroup in OpenCL and Vulkan
- Exploiting subgroups allows for various optimizations
  - Scalarization (i.e. marking wave-wide uniform data)
  - Reduction of local memory traffic

WORK GROUPS, SUBGROUPS, THREADS

- Let’s assume we have a machine with a 4-wide vector unit
- A possible execution of a 16 element work group could be …



SUBGROUPS

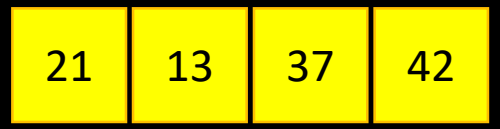
- Behave as if they get executed together
  - SIMD unit run in lockstep
  - “as-if” model, HW could be implemented differently
- The subgroup doesn’t necessarily have to fill the whole hardware unit (for instance, draw call doesn’t produce enough pixels, etc.)
- Subgroup size varies by vendor (though everyone agreed on <= 64)
  - You shouldn’t hardcode it
  - Workgroup size is ideally an integer multiple of the subgroup size
  - Can be queried outside of shader but rarely useful there

SUBGROUP OPERATIONS

- Instructions which allow one lane to see data from other lanes
  - Broadcast
  - Ballots
  - Reads
- Instructions which perform cross-lane computation
  - Either returning the same value per lane or varying data
  - Prefix-sums, reductions, etc.

SUBGROUP OPERATIONS

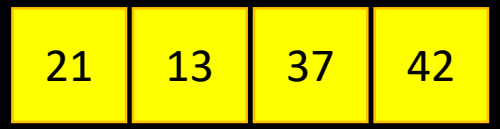
- Ballot broadcasts 1 bit per lane across all lanes



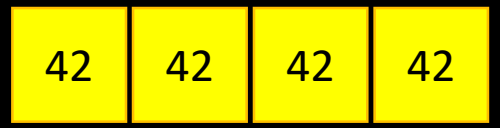
`subgroupBallot(v <= 21)` => 1100 (i.e. every lane sees the input of all other lanes)

SUBGROUP OPERATIONS

- Data broadcast – read one lane across the subgroup




```
subgroupBroadcast(v, 3)
```

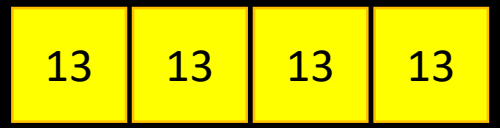


SUBGROUP OPERATIONS

- Broadcast – read one lane across the subgroup

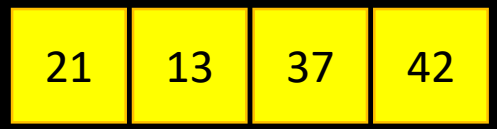
(Gray lanes are inactive)

```
subgroupBroadcastFirst (v)
```



SUBGROUP OPERATIONS

- On top of those two, we can build higher-level functions (v = current lane value)
- `subgroupAllEqual (v)` => false
- `subgroupAny(v == 21)` => true for the whole subgroup!
- `subgroupAll(v == 21)` => false for the whole subgroup!



SUBGROUP OPERATIONS

Reductions: `subgroup<op> (value)`

Given an operation $\oplus$ and invocations $a_0 \dots a_n$

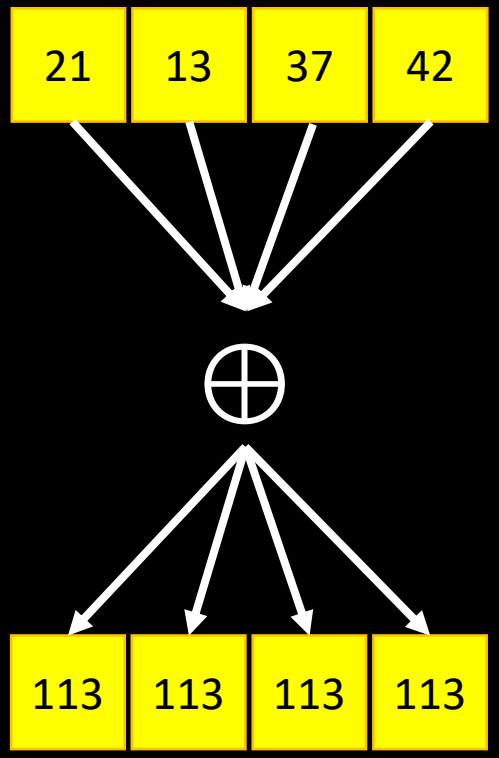
`subgroup`$\oplus$, for all invocations:	$a_0 \oplus \dots \oplus a_n$
`subgroupInclusive`$\oplus$ for invocation a_i:	$a_0 \oplus \dots \oplus a_i$
`subgroupExclusive`$\oplus$ for invocation a_i:	$id \oplus a_0 \oplus \dots \oplus a_{i-1}$

Id is the identity element for the operation:

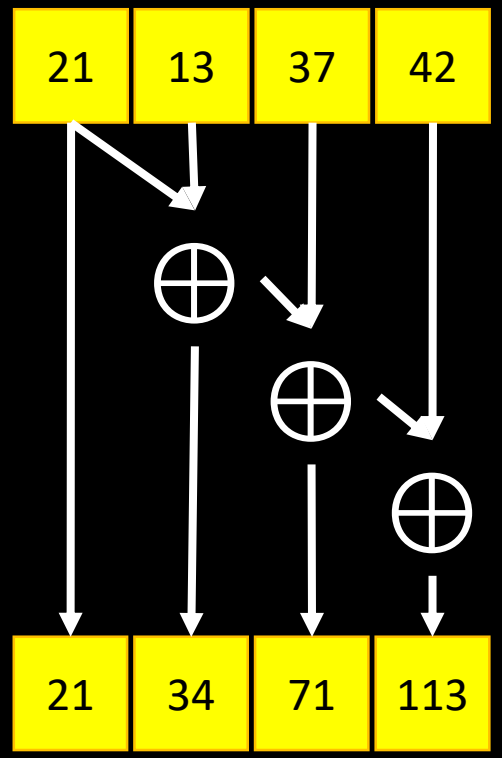
0 for additions, 1 for multiplications, etc.

SUBGROUP OPERATIONS

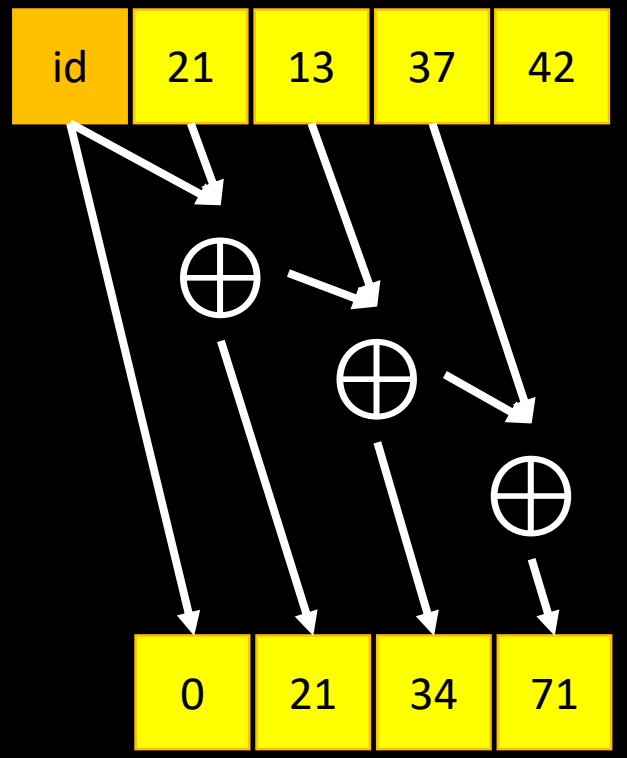
subgroupAdd



subgroupInclusiveAdd



subgroupExclusiveAdd

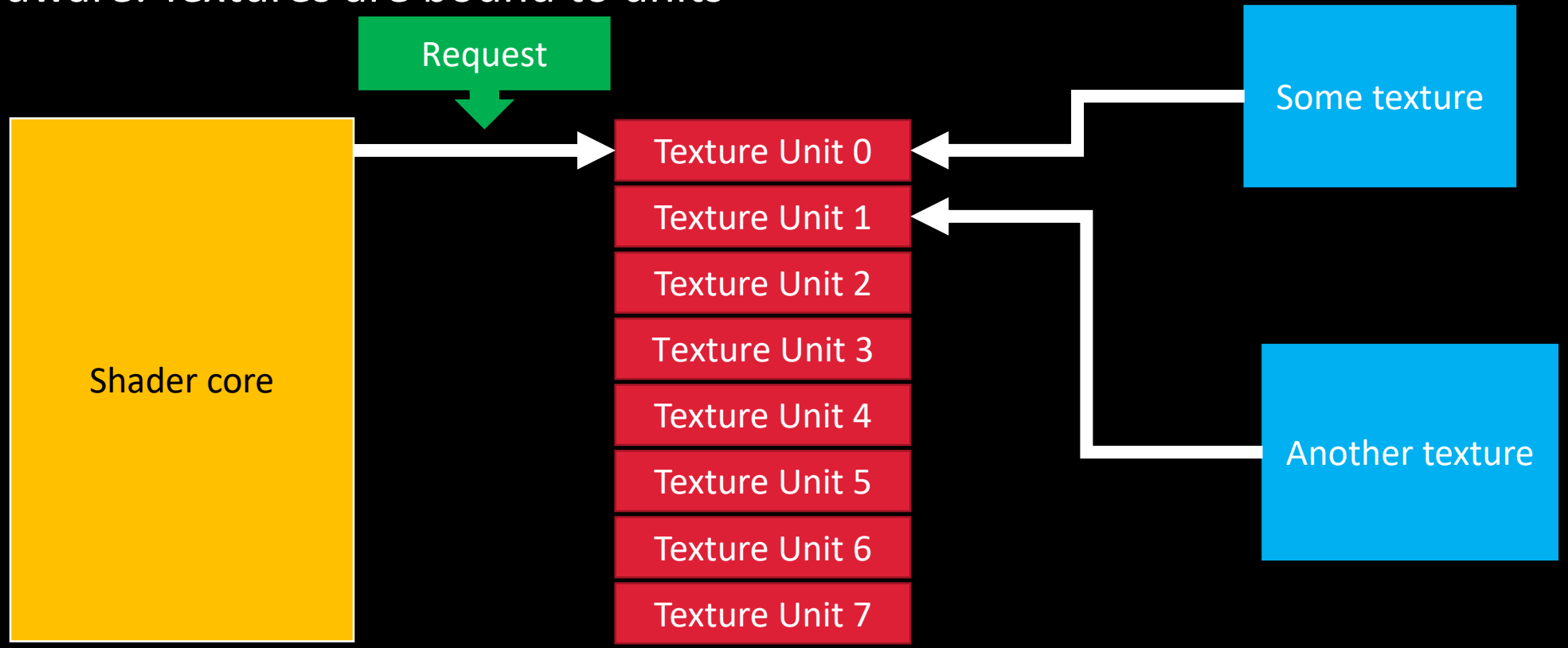


SUBGROUPS

- … and more
- Why do you want this?
  - Direct visibility into divergence: If more than half of the threads take the expensive path, might as well do it for all
  - Optimize wide reductions: subgroup first, then through shared memory, then through global memory
- Subgroup operation support varies per vendor, **query** which operation are present
- Don’t **assume a certain width** to ensure portability!
- Subgroups **can** be available in more than just compute shaders!

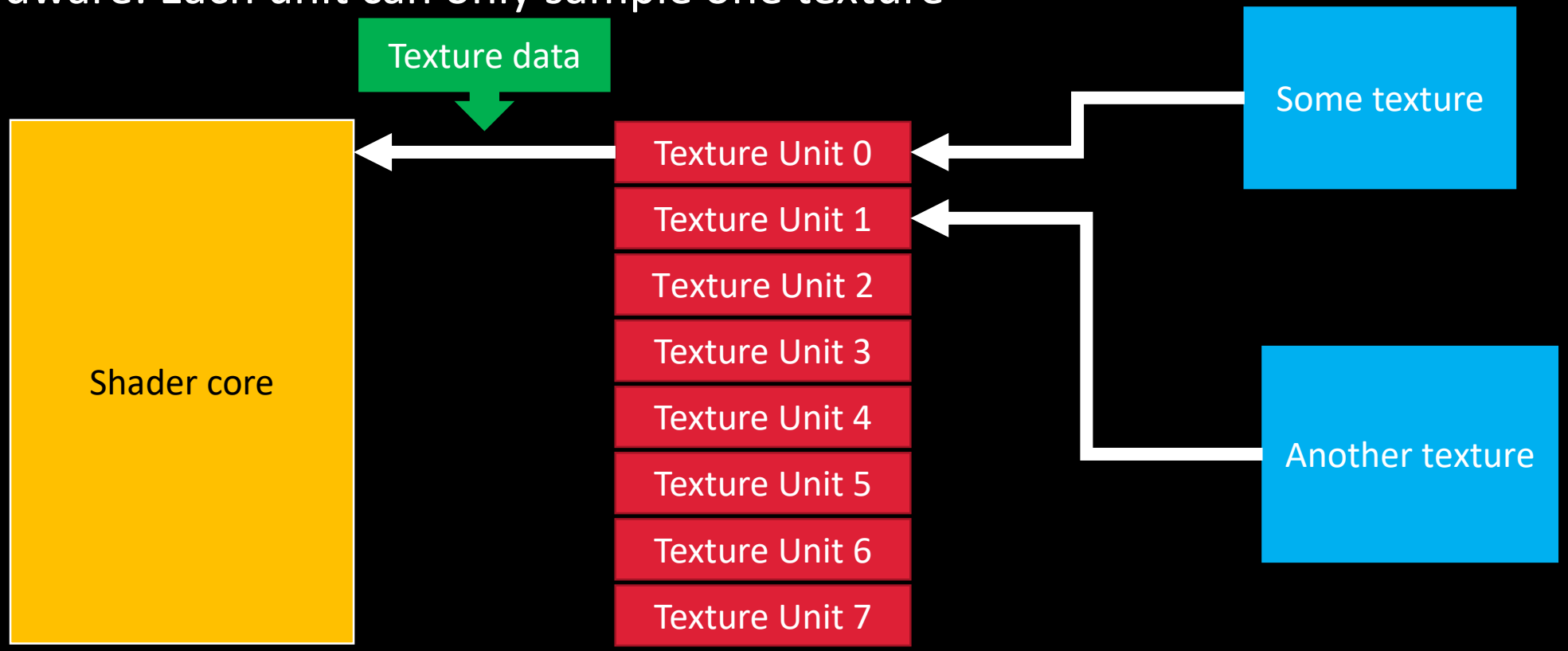
DESCRIPTORS

- Old hardware: Textures are bound to units



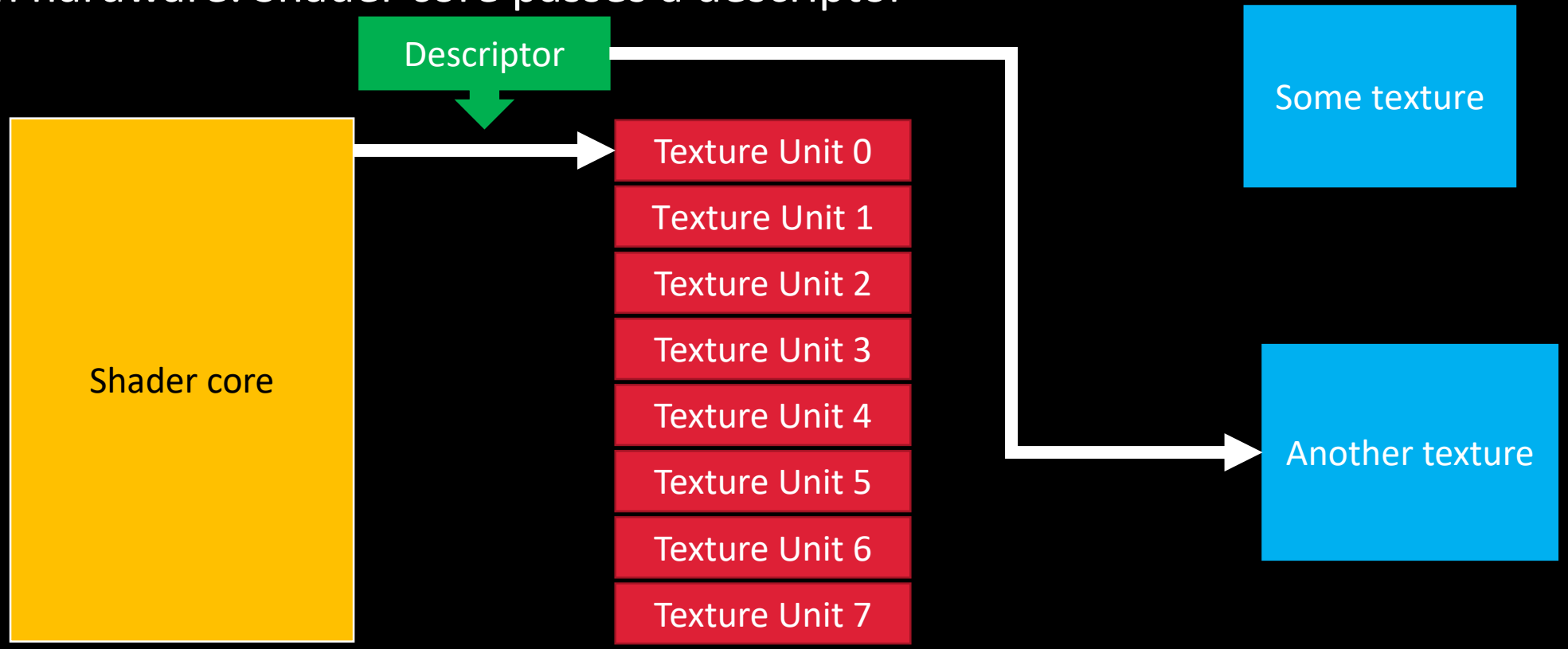
DESCRIPTORS

- Old hardware: Each unit can only sample one texture



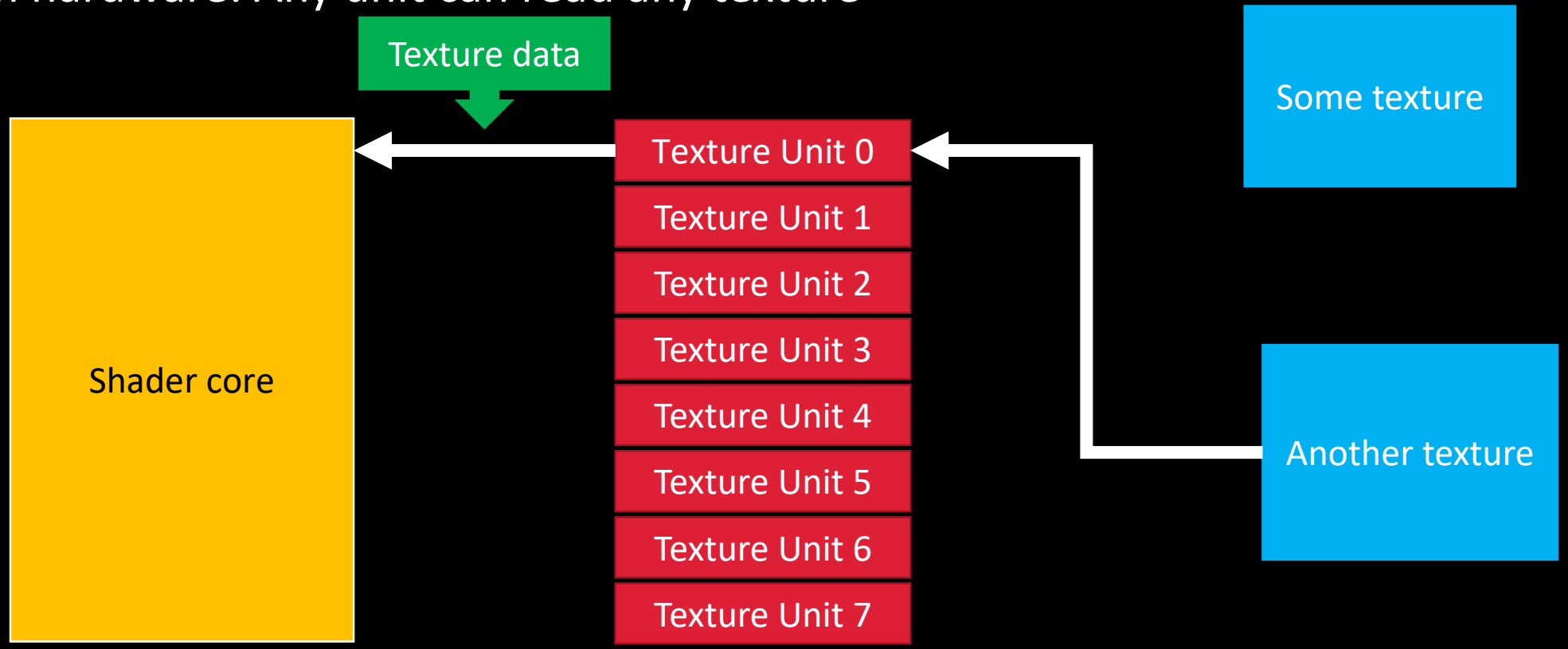
DESCRIPTORS

- Modern hardware: Shader core passes a descriptor



DESCRIPTORS

- Modern hardware: Any unit can read any texture



BINDLESS AKA DESCRIPTOR INDEXING

- So far, we're still binding descriptors per draw
- Bindless is the idea to have all resources bound at all times for all draw calls
  - Shader gets to decide
  - Shader sees all descriptors
- This is EXT for now, so not core

BINDLESS … COULDN'T WE DO THIS BEFORE IN VULKAN?

- Yes and no …
- You could bind many descriptors
- … but **all** had to be **valid** (no gaps)
- … divergent access had to be resolved **manually**
- … no **update** after binding
- … limits were not very impressive on some HW

BINDLESS … NOW

- Yes!
- You can bind them all now!
- … and they can have **gaps** (needs an extra flag to enable)
- … you use extra markup, and the **compiler/driver solves divergent access**
- … you can **update** after binding as long as the descriptor is not currently used
- … limits are **no longer a concern** (500k descriptors should be enough for anyone)

BINDLESS IN PRACTICE

- Bind an unbounded array and access it

```
RWTexture2D<float> Result : register(u0);
Texture2D LotsOfTextures[] : register (t1);

[numthreads(64,1,1)]
void main (uint3 tid : SV_DispatchThreadid)
{
    uint index = tid.x % 5324;
    Texture2D tex = LotsOfTextures [NonUniformResourceIndex(index)];
    float4 sampleValue = tex[tid.xy];
    Result[tid.xy] = sampleValue;
}
```

BINDLESS IN PRACTICE

Did that look like HLSL to you? More on that later ☺

Why do you want bindless?

- GPU driven pipelines – no longer need to bind resources per draw
- Simpler management – just have all resources available in one large set instead of juggling sets
- Easier update – update descriptors at any time

DXC/GLSLANG

- “Improved HLSL support” – but that still needs a compiler
- We have now two compilers accepting HLSL and emitting SPIR-V
  - GLSLang is the go-to compiler for now and has good support up to SM5
  - DXC is the “new” one
- DXC is the future
  - It’s the same compiler that is used for HLSL going forward
  - Fully supports SM6
- With DXC, you can use large existing HLSL codebases

LARGE ECOSYSTEM AROUND SHADERS

spirv-opt is integrated in DXC and GLSLang for HLSL code

spirv-reflect provides similar functionality to D3D reflection

spirv-cross generates source (in other shading languages!) from SPIR-V

SUMMARY

- VK 1.1 is just the beginning
- Lots of innovation happening around shaders with the community
  - More people targeting SPIR-V – see Fabian's talk later on!
  - HLSL is a **first-class citizen** for Vulkan
- Continuous feedback from the community is **critically important**!
  - Participate on GitHub, the forums, Twitter
  - Tells us what went wrong!

Thanks! Questions?

@NIV_ANTERU | MATTHAEUS.CHAJDAS@AMD.COM

We're hiring!