CDNA2 Memory Hierarchy

Alessandro Fanfarillo

A Team Effort

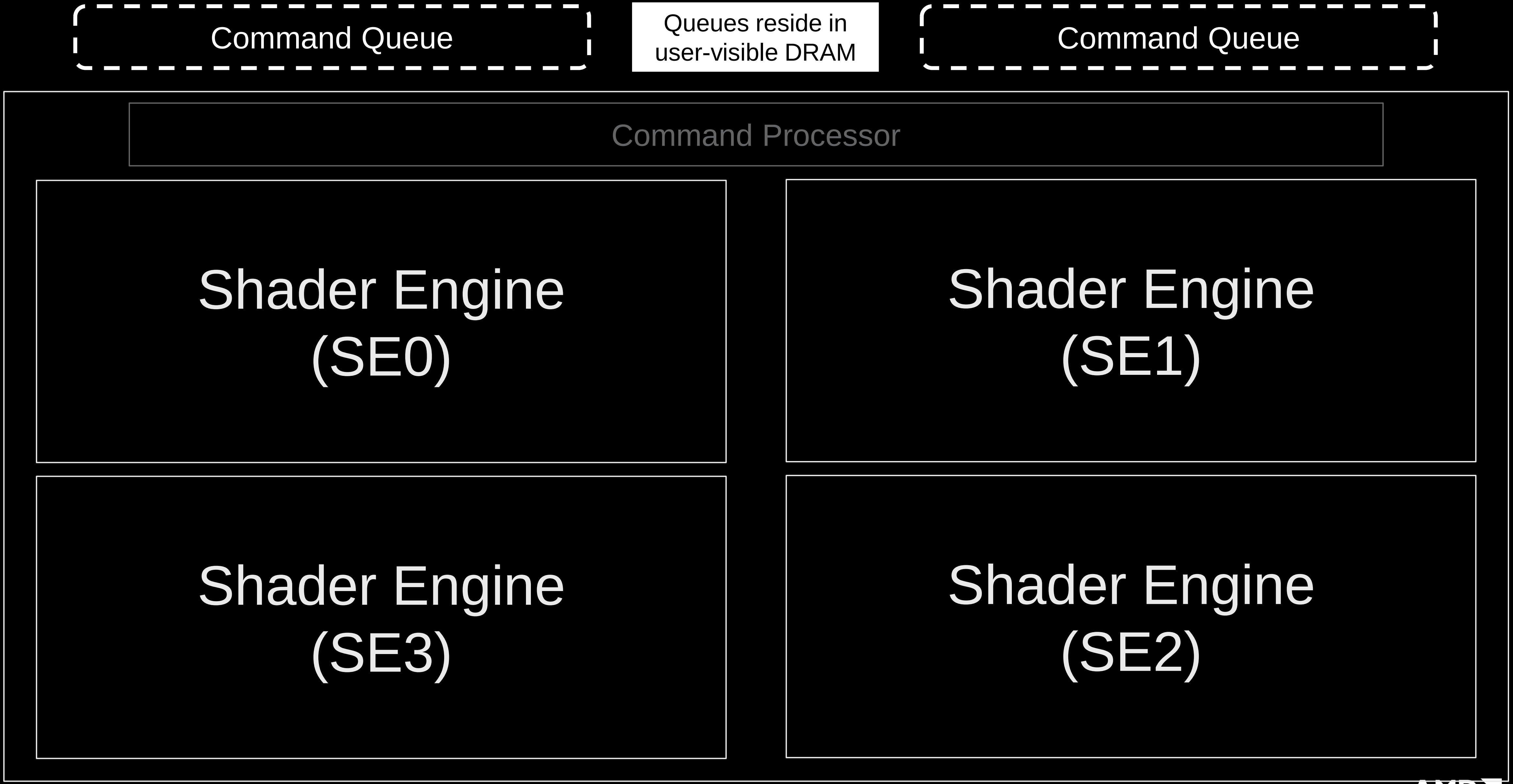
Thanks to all former contributors to this presentation:

Paul Bauman
Noel Chalmers
Nicholas Curtis
Chip Freitag
Joseph Greathouse
Nicholas Malaya
Damon McDougall
Scott Moe
René van Oostrum
Noah Wolfe

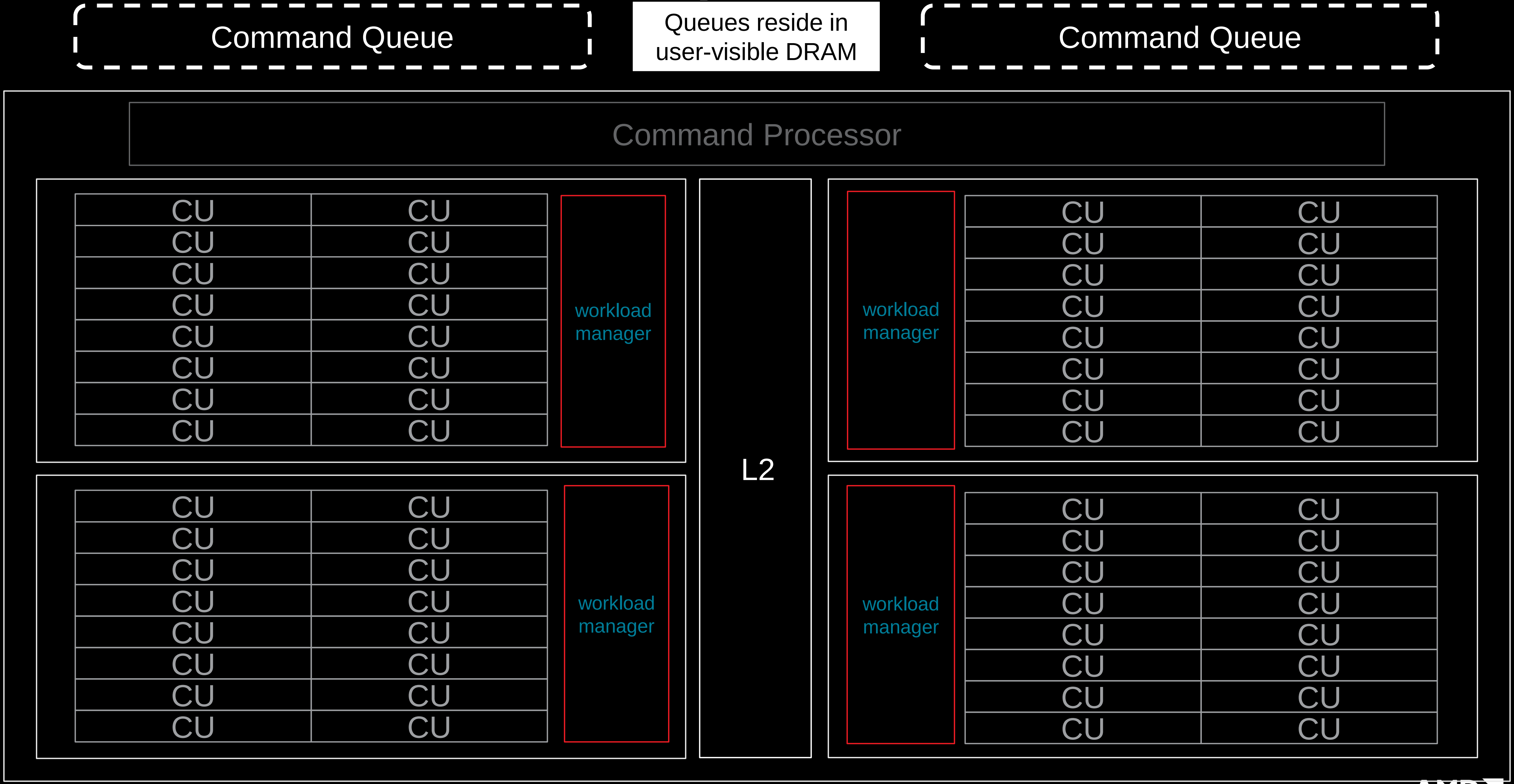
Agenda

- Introduction to CDNA2 Compute Unit architecture
- Memory hierarchy in CDNA2 Compute Units
- Occupancy considerations with examples

AMD CDNA2 GPU Hardware Layout

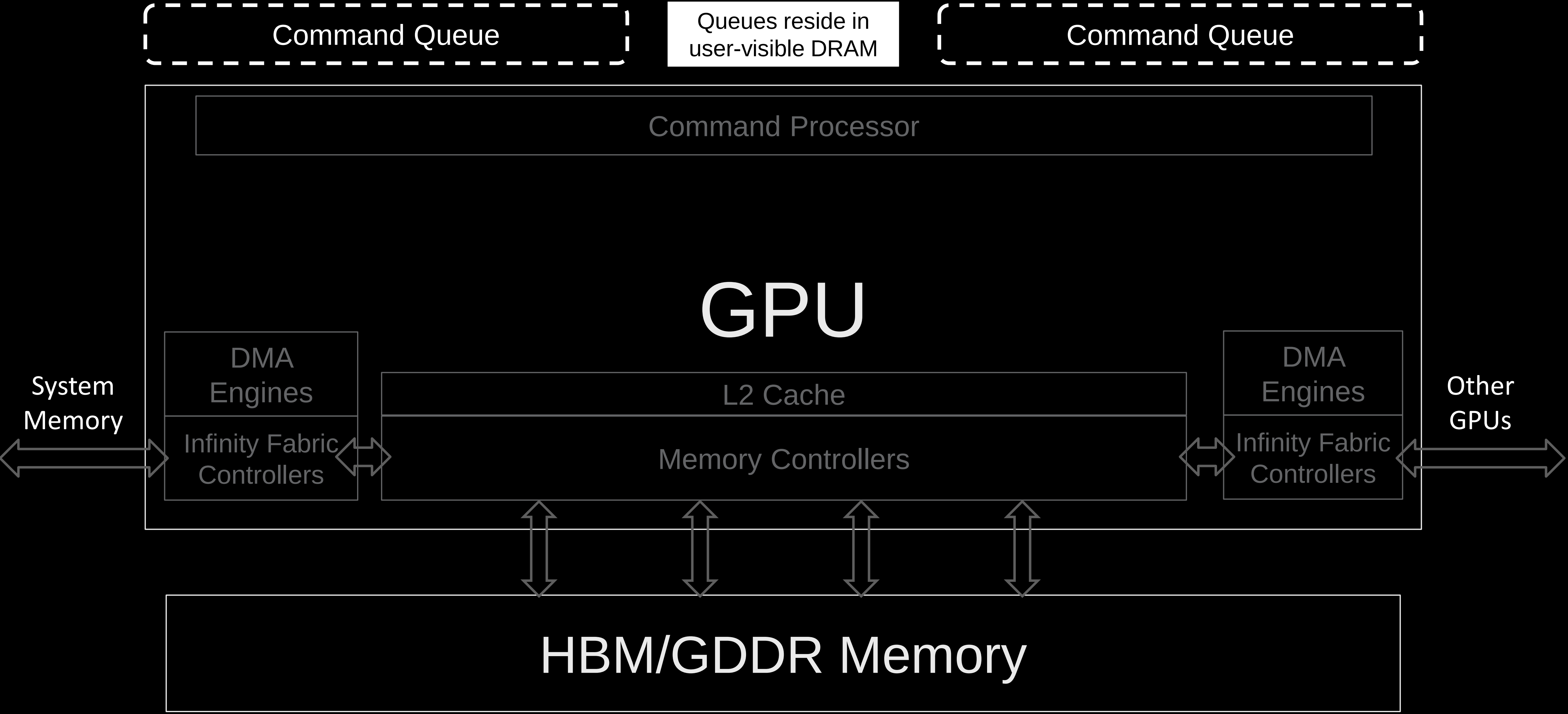


AMD CDNA2 GPU Hardware Layout



AMD together we advance_

GPU Memory, I/O, and Connectivity



The CDNA2 Compute Unit (CU)

Compute Unit (CU)

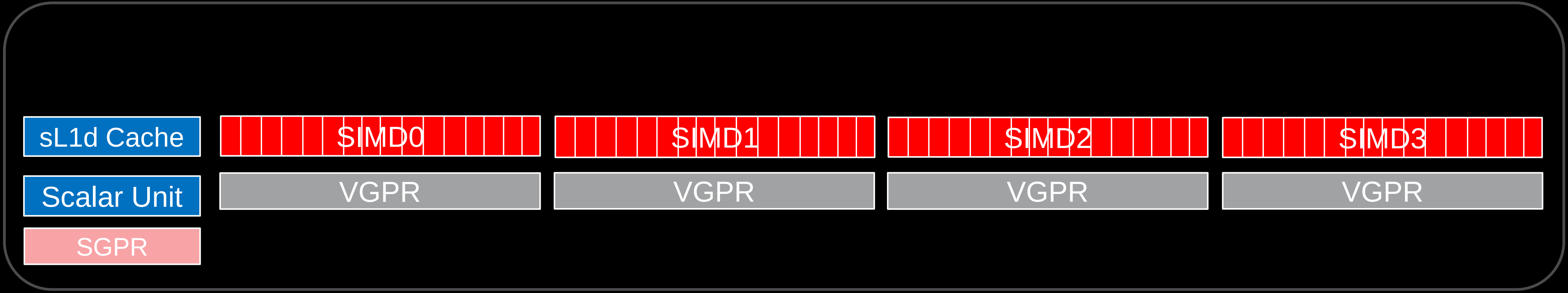
- The command processor sends work packages (i.e. workgroups of work-items in HIP) to the Compute Units (CUs)
  - Workgroups are executed in wavefronts (groups of 64 work-items on a SIMD)
  - All wavefronts in a workgroup reside on the same CU
  - The CU’s scheduler can hold wavefronts from many workgroups
  - At most 32 wavefronts total per CU (8 per SIMD)

The CDNA2 Compute Unit (CU) – Scalar Unit

sL1d Cache
Scalar Unit
SGPR

- Scalar Unit (SU)
  - Shared by all work-items in each wavefront, accessed on a per-wavefront level
  - Work-items in a wavefront performing the exact same operation can offload this instruction to the SU
  - Used for control flow, pointer arithmetic, dispatch a common constant value, etc. Only INT32 capability, no FP
  - SU connected to read/write sL1d cache of 16 KiB (not really into the CU but directly attached)
  - Has its own pool of Scalar General-Purpose Register (SGPR) file, 12.5KiB per CU, 800 per SIMD
    - Maximum of 102 SGPRs / wavefront allocated in groups of 16

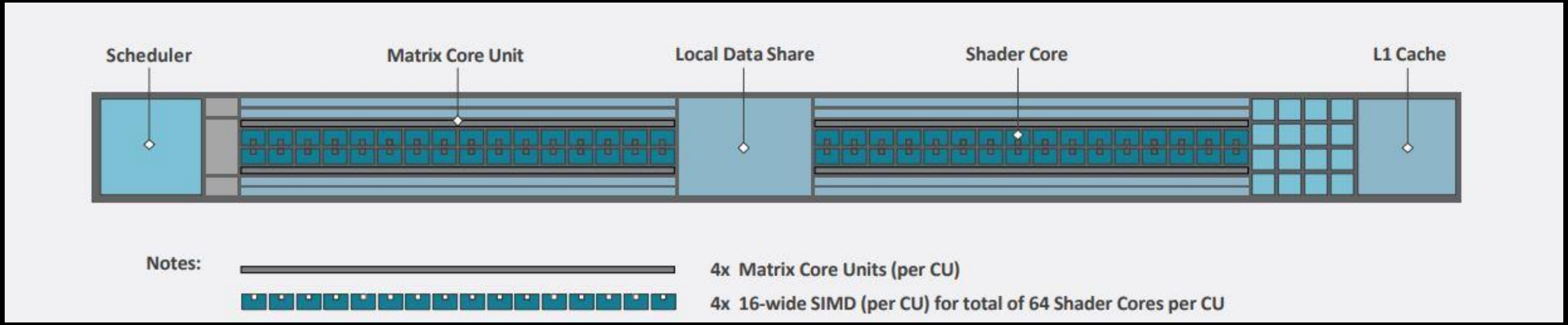
The CDNA2 Compute Unit (CU) – Vector ALU



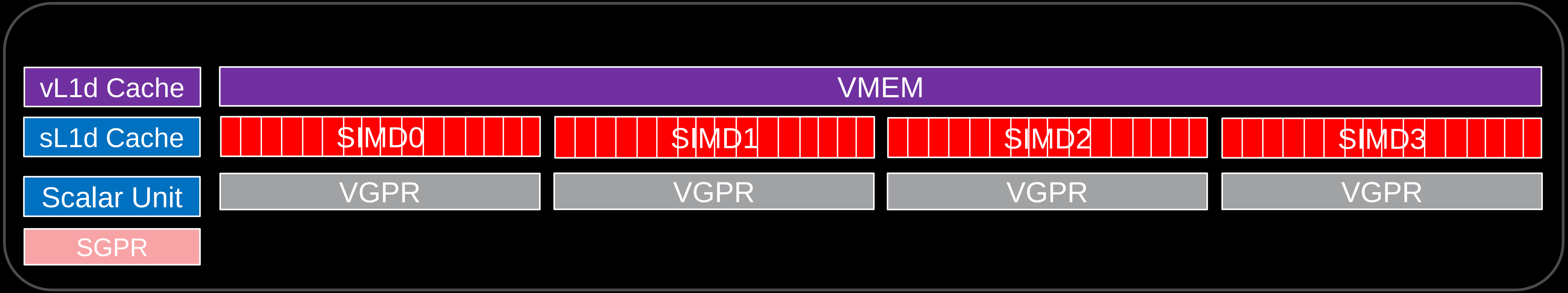
- SIMD Units / Execution Units (EU) / VALU
  - 4x SIMD vector units (each 16 lanes wide)
  - Each SIMD performs vector logical, integer, FP16, FP32, FP64 operations. FMAs for FP16, FP32, FP64. MFMAs for FP16, BF16, FP32, FP64. Packed FP16 and FP32.
  - Two pools of Vector General-Purpose Registers (VGPRs): regular VGPRs and Accumulation VGPRs (AccVGPRs)
  - Maximum of 512 registers per SIMD – each register is 64x 4-byte entries. For 64 bits operations 2 contiguous registers need to be used.
  - A wavefront can use up to 256 VGPRs (and 256 AccVGPRs)
  - Instruction buffer for 8 wavefronts on each SIMD unit. Each wavefront is local to a single SIMD unit, not spread among the four

The CDNA2 Compute Unit (CU) – Matrix Cores

- Matrix Fused Multiply Add (MFMA) instructions operate on a per-wavefront basis rather than on a per-thread basis
- For more info about MFMA instructions and register usage check out the AMD Matrix Instruction Calculator: https://github.com/RadeonOpenCompute/amd_matrix_instruction_calculator
- Matrix Cores leveraged is several ways:
  - Libraries: rocBLAS, rocWMMA
  - Use compiler intrinsics
  - HIP kernels with inline assembly
  - Write kernels completely in assembly…
- More details on how to use MFMA instructions: https://gpuopen.com/learn/amd-lab-notes/amd-lab-notes-matrix-cores-readme

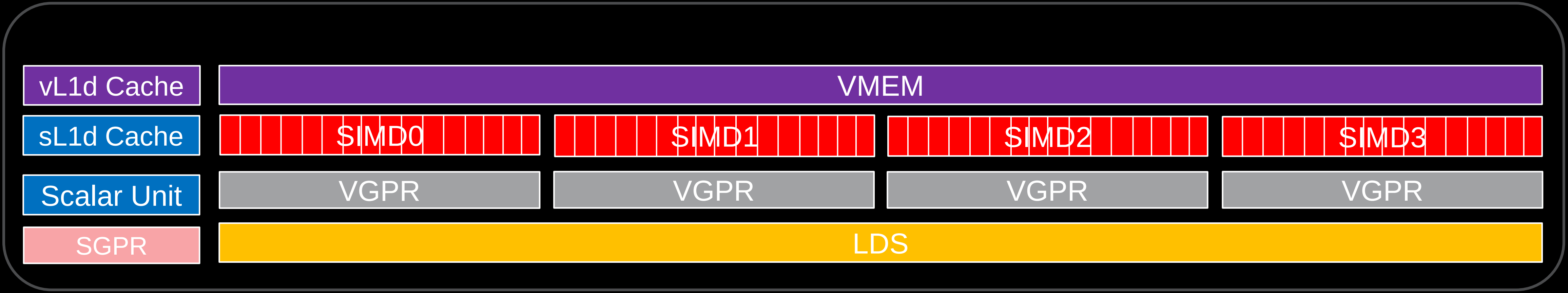

Scheduler
Matrix Core Unit
Local Data Share
Shader Core
L1 Cache
Notes:
4x Matrix Core Units (per CU)
4x 16-wide SIMD (per CU) for total of 64 Shader Cores per CU

The CDNA2 Compute Unit (CU) – Vector Memory Unit



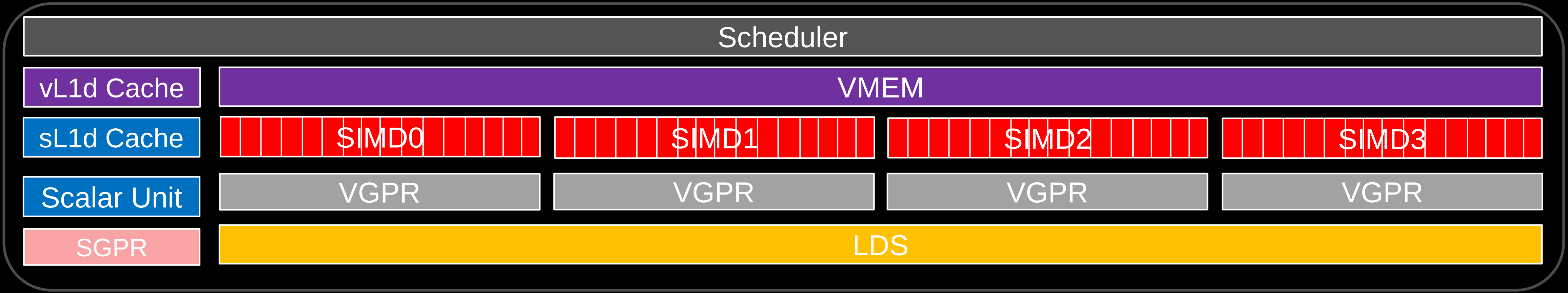
- Vector Memory Unit
  - Vector memory operations from all 4 SIMD units are routed to the Vector Memory Unit (VMEM)
  - Can handle uncoalesced memory addresses
  - Connects to a 16 KiB L1 Data Cache (vL1d). Cache lines of 64 bytes (L2 cache line is 128 bytes).
  - Write-through
  - vL1d cache not really into the CU but directly connected to it

The CDNA2 Compute Unit (CU) - LDS



- 64 KB Local Data Share (LDS, or shared memory)
  - 32 banks with conflict resolution
  - Can share data between all work-items in a workgroup
  - It supports various HW atomic operations for integer, logical, and floating-point data types.

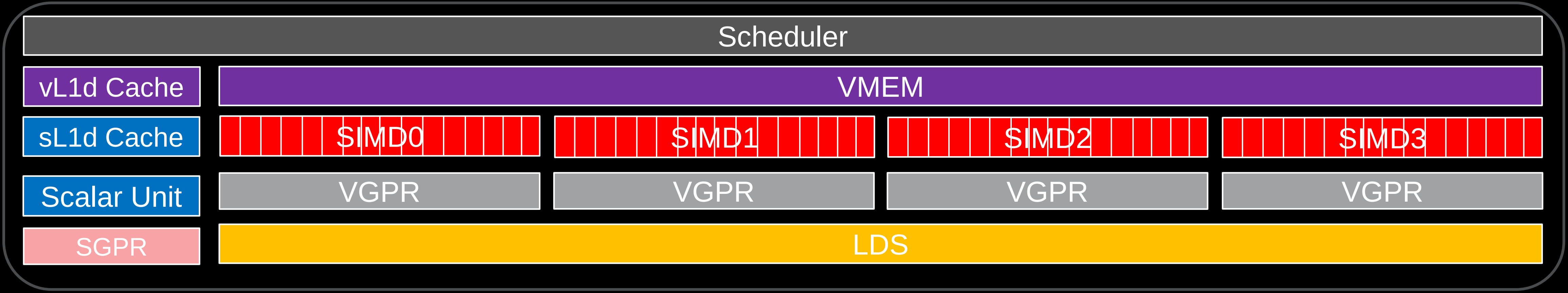
The CDNA2 Compute Unit (CU) - Scheduler



- Scheduler
  - Buffer for up to 32 wavefronts
  - Separate decode/issue for
    - VALU, VGPR load/store
    - SALU, SGPR load/store
    - LDS load/store
    - Global mem load/store
    - Special instructions (NoOps, barriers, branch instructions)

[Public]

The CDNA2 Compute Unit (CU) - Scheduler



- Scheduler
  - At each clock, waves on **1 SIMD unit** are considered for execution (Round Robin scheduling among SIMDs)
  - At most **1 instruction per wavefront** may be issued
  - At most **1 instruction from each category** may be issued (VALU, VMEM, SALU/SMEM, LDS, branch)
  - **Maximum of 5** instructions issued to wavefronts on a single SIMD, per cycle per CU
  - VALU instructions take a multiple of four cycles to retire
    - e.g. FP32 FMA: cycle 0 – lanes 0-15 | cycle 1 – lanes 16-31 | cycle 2 – lanes 32-47 | cycle 3 – lanes 48-63
    - Programmer can still ‘pretend’ CU operates in 64-wide SIMD: 64 FP32 FMA ops / cycle / CU

GPU Occupancy on CDNA2

AMD

What is Occupancy?

Occupancy: the ratio of active wavefronts executing on the GPU to the maximum number of possible wavefronts supported by the hardware.

- Occupancy is controlled by the utilization of resources on a CU
- Can indicate over/under utilization of resources, limiting performance

Different “flavors” of occupancy available:

→ **Achieved** occupancy is measured on the hardware and is a time-dependent metric (as the number of active wavefronts is not constant)

→ **Theoretical** occupancy is a calculated metric, derived from the resources requested by the kernel. Compiler can provide this information

→ In addition, occupancy may be reported per-SIMD/EU, per-CU, or per-GPU

To see why occupancy is important, we will consider a batch matrix-vector multiply kernel.

Occupancy: Limiting Factors

- Number of wavefronts: max 8 per SIMD, 32 per CU
- Register usage is a big limiting factor to occupancy. Both SGPRs and VGPRs play a role
- LDS usage is another limiting factor
- Number of wavefronts per workgroup (AKA thread block): max 16 (i.e., max 1024 threads per workgroup).
    - Note that all wavefronts of a workgroup are required to be scheduled on the same CU, but not necessarily on the same SIMD of the CU.

Occupancy: Limiting Factors - VGPRs

- Vector registers:
  - Total of 64x 512 registers available per SIMD (256 VGPRs + 256 AccVGPRs)
  - Each wavefront can use up to 256 VGPRs, if more are needed “spilling” to global memory (cacheable)

Num VGPRs	Occupancy per EU	Occupancy per CU
<= 64	8 waves	32 waves
<= 72	7 waves	28 waves
<= 80	6 waves	24 waves
<= 96	5 waves	20 waves
<= 128	4 waves	16 waves
<= 168	3 waves	12 waves
<= 256	2 waves	8 waves
> 256 (+ spilling to AVGPRs/scratch)	1 waves	4 waves

Occupancy: Limiting Factors - SGPRs

- Scalar registers:
- Total scalar register file size: 12.5 KB (3,200 registers, 800 per SIMD)
- A single wavefront can allocate up to 112 scalar registers in batches of 16
  - The last 6 of these are used for special purposes (such as VCC), and these cannot be used as general purpose scalar registers by user code
  - The 112 case is special; here, 4 additional registers cannot be used, leaving 102 for GPR purposes
  - For each wavefront, 16 additional registers are allocated for a trap handler
- Assuming no register spilling from SGPRs to VGPRs is performed by the compiler and that the number of VGPRs is low enough to allow max occupancy, occupancy will be 8 per SIMD up to 100 SGPRs
- When SGPRs usage > 100 occupancy will drop down to 7 wavefronts per SIMD

Occupancy: Register Spilling

- SGPRs
  - Not observed to be a common source of spilling
  - Spilled to vector registers (VGPRs)
- VGPRs
  - Spilled to AGPRs, then L2 (L1 is write-through), and finally to HBM
  - A wavefront can use directly up to 256 VGPRs. It can spill to up to 256 AVGPRs (assuming no MFMA instructions are used)
- `__launch_bounds__(MAX_THREADS_PER_BLOCK, MIN_WARPS_PER_EU)`
  - A function attribute that must be attached to a global device function
  - Provides hints for compiler to manage/reduce register usage per kernel
  - `MAX_THREADS_PER_BLOCK`: guarantees launch size to compiler
  - `MIN_WARPS_PER_EU`: asks compiler to minimize register usage to allow at least x-many warps to be active per SIMD unit/EU

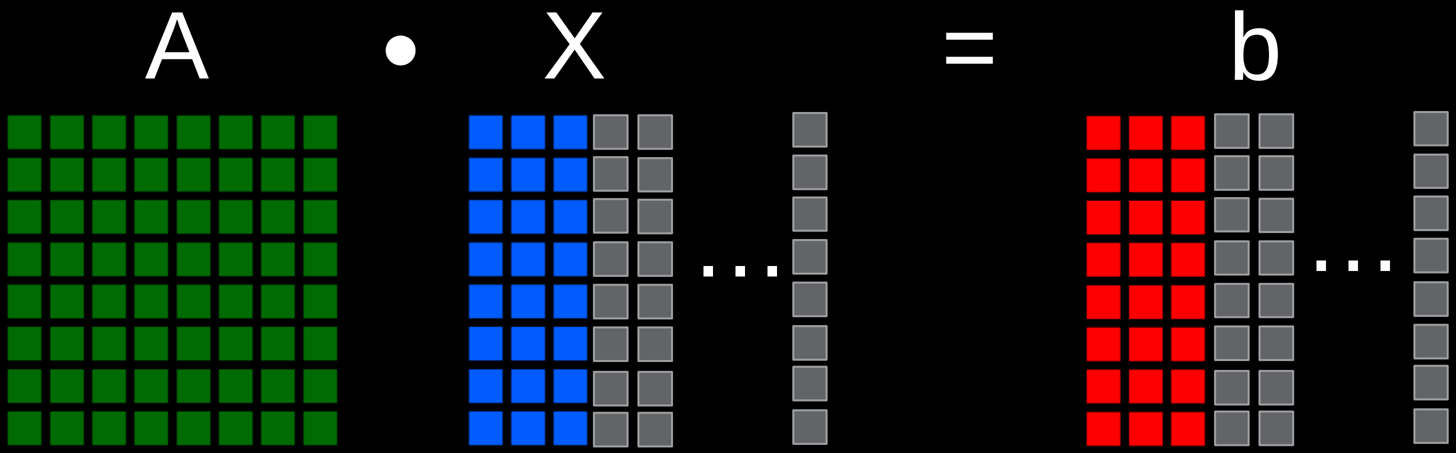
Occupancy: Limiting Factors - LDS

- Local Data Share:
  - Note that for occupancy calculations, we need to look at the usage per workgroup, not per wavefront
- 64 KB per Compute Unit

Example: batched matrix-vector multiply

As a test-bed for our occupancy calculations, we will use a batched matrix-vector multiplication kernel:

- $\bar{A}$ is a ($N_m x N_m$) matrix
- $\vec{x}$ and $\vec{b}$ are N_v vectors each of size (N_mx1)

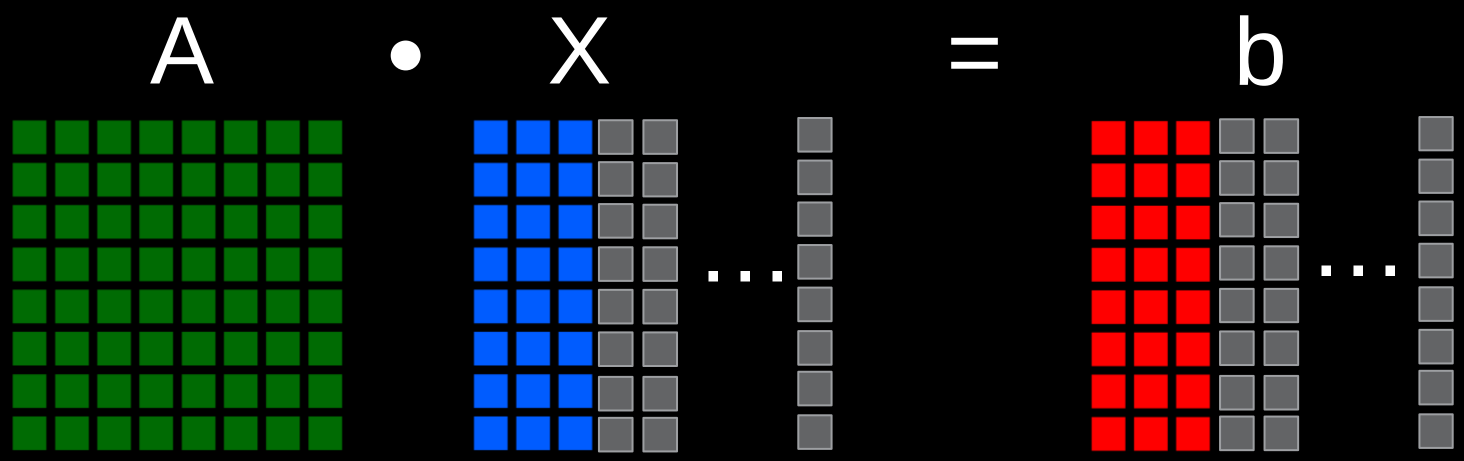


AMD
together we advance_

Example: batched matrix-vector multiply

Main implementation ideas:

- Every work-item multiplies $\bar{A}$ with multiple vectors from $\vec{x}$.
- The data of a vector from $\vec{x}$ is reused N_m times.
- Instead of loading a vector from $\vec{x}$ from HBM for every use, we preload a batch of WG-size * N_b of them in (faster) LDS, and use them repeatedly from there.



AMD together we advance_

Example occupancy calculation

Parameter	Value
WG-size	128
N_m	4
N_b	32
N_v	3.35E+08

Kernel configuration V0

Resulting performance ~55 GFLOP/s, very poor! Why?

One reason: using too much LDS per work-group!

```
mxv.cpp:44:1: remark:    SGPRs: 22 [-Rpass-analysis=kernel-resource-usage]
mxv.cpp:44:1: remark:    VGPRs: 74 [-Rpass-analysis=kernel-resource-usage]
mxv.cpp:44:1: remark:    AGPRs: 0 [-Rpass-analysis=kernel-resource-usage]
mxv.cpp:44:1: remark:    ScratchSize [bytes/lane]: 0 [-Rpass-analysis=kernel-
resource-usage]
mxv.cpp:44:1: remark:    Occupancy [waves/SIMD]: 1 [-Rpass-analysis=kernel-
resource-usage]
mxv.cpp:44:1: remark:    SGPRs Spill: 0 [-Rpass-analysis=kernel-resource-usage]
mxv.cpp:44:1: remark:    VGPRs Spill: 0 [-Rpass-analysis=kernel-resource-usage]
mxv.cpp:44:1: remark:    LDS Size [bytes/block]: 65536 [-Rpass-analysis=kernel-
resource-usage]
```

$$
\begin{aligned}
\mathrm{LDS} &= \mathrm{WG}_{\mathrm{size}} \times \mathrm{N_b} \times \mathrm{N_m} \times \mathrm{sizeof(float)} \\
&= 128 \times 32 \times 4 \times 4 \text{ bytes} \\
&= 64\ \mathrm{KB/WG}
\end{aligned}
$$

Example occupancy calculation

Parameter	Value
WG-size	256
N_m	4
N_b	16
N_v	3.35E+08

Kernel configuration V1

Resulting performance ~**93** GFLOP/s Why?

```
mxv.cpp:44:1: remark:    SGPRs: 22 [-Rpass-analysis=kernel-resource-usage]
mxv.cpp:44:1: remark:    VGPRs: 42 [-Rpass-analysis=kernel-resource-usage]
mxv.cpp:44:1: remark:    AGPRs: 0 [-Rpass-analysis=kernel-resource-usage]
mxv.cpp:44:1: remark:    ScratchSize [bytes/lane]: 0 [-Rpass-analysis=kernel-
resource-usage]
mxv.cpp:44:1: remark:    Occupancy [waves/SIMD]: 1 [-Rpass-analysis=kernel-
resource-usage]
mxv.cpp:44:1: remark:    SGPRs Spill: 0 [-Rpass-analysis=kernel-resource-usage]
mxv.cpp:44:1: remark:    VGPRs Spill: 0 [-Rpass-analysis=kernel-resource-usage]
mxv.cpp:44:1: remark:    LDS Size [bytes/block]: 65536 [-Rpass-analysis=kernel-
resource-usage]
```

$$\begin{aligned}\mathrm{LDS} &= \mathrm{WG}_{\mathrm{size}} \text{ x } \mathrm{N}_{\mathrm{b}} \text{ x } \mathrm{N}_{\mathrm{m}} \text{x sizeof(float)} \\ &= 128 \text{ x } 32 \text{ x } 4 \text{ x } 4 \text{ bytes} \\ &= 64 \text{ KB/WG}\end{aligned}$$

AMD
together we advance_

Example occupancy calculation

Recall: 64KB of LDS available per CU

→ Limited to a single WG of 128 work-items per CU in kernel V0

→ Limited to a single WG of 256 work-items per CU in kernel V1

Recall: 32 Wavefronts possible per CU:

→ $\mathrm{Occupancy} = \frac{2}{32} = 0.0625$ for kernel V0

→ $\mathrm{Occupancy} = \frac{4}{32} = 0.125$ for kernel V1

Solution: lower LDS usage per WG

- In this example, we can either decrease the workgroup size, or **decrease the batch size N_b**

Example occupancy calculation

Parameter	Value
WG-size	128
N_m	4
N_b	1
N_v	3.35E+08

Kernel configuration V2

Resulting performance ~**1031** GFLOP/s

```
mxv.cpp:44:1: remark:    Occupancy [waves/SIMD]: 8 [-Rpass-analysis=kernel-resource-usage]
mxv.cpp:44:1: remark:    LDS Size [bytes/block]: 2048 [-Rpass-analysis=kernel-resource-usage]
```

$$\begin{aligned} \mathrm{LDS} &= \mathrm{WG_{size}} \text{ x } \mathrm{N_b} \text{ x } \mathrm{N_m} \text{x sizeof(float)} \\ &= 128 \text{ x } 1 \text{ x } 4 \text{ x } 4 \text{ bytes} \\ &= 2\ \mathrm{KB/WG} \end{aligned}$$

Example occupancy calculation

Parameter	Value
WG-size	256
N_m	4
N_b	1
N_v	3.35E+08

Kernel configuration V3

Resulting performance ~**1039** GFLOP/s

```
mxv.cpp:44:1: remark:    Occupancy [waves/SIMD]: 8 [-Rpass-analysis=kernel-resource-usage]
mxv.cpp:44:1: remark:    LDS Size [bytes/block]: 4096 [-Rpass-analysis=kernel-resource-usage]
```

$$\begin{aligned} \mathrm{LDS} &= \mathrm{WG}_{\mathrm{size}} \text{ x } \mathrm{N_b} \text{ x } \mathrm{N_m} \text{x } \mathrm{sizeof(float)} \\ &= 256 \text{ x } 1 \text{ x } 4 \text{ x } 4 \text{ bytes} \\ &= 4 \text{ KB/WG} \end{aligned}$$

Wrap Up

- An entire workgroup is assigned to a single CU (round-robin across all the various SEs)
- An entire wavefront is assigned to a single SIMD unit / execution unit (EU)
- It takes 4 cycles to execute an entire wavefront. EUs are 16-wide
- 256 VGPRs + 256 AccVGPRs (512 total) usable by an EU
- 256 VGPRs (+256 AccVGPRs for spilling) usable by a wavefront
- 112 SGPRs usable by a wavefront (only 102 used by kernel)
- vL1 cache is 16 KB shared by all EUs (entire CU)
- sL1 cache is 16 KB shared by all EUs (entire CU)
- LDS is 64 KB per CU
- Occupancy limited by:
  1. Register pressure – Wavefront level
  2. LDS usage – Workgroup level
  3. Number of wavefronts per CU (HW limit is 32; 8 wavefronts per EU)
  4. Number of wavefronts per workgroup (16 wavefronts max per workgroup)

DISCLAIMERS AND ATTRIBUTIONS

AMD
together we advance_

AMD