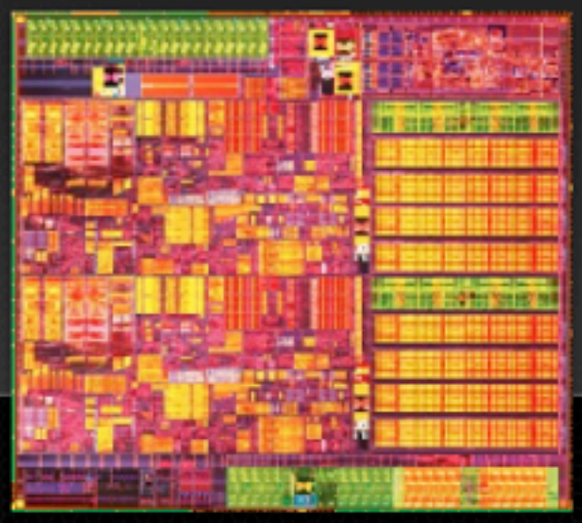
524

GPUs

CPU
1690 pJ/flop

Optimized for Latency

Caches

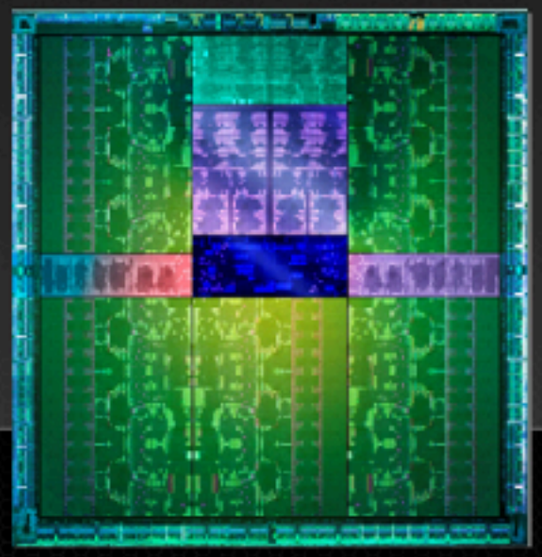


Westmere
32 nm

GPU
140 pJ/flop

Optimized for Throughput

Explicit Management
of On-chip Memory



Kepler
28 nm

NVIDIA

1ST ERA: Fixed Function

3D Geometry Transformation

$$V_{eye}\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = MVP \begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix} \bullet V_{obj} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

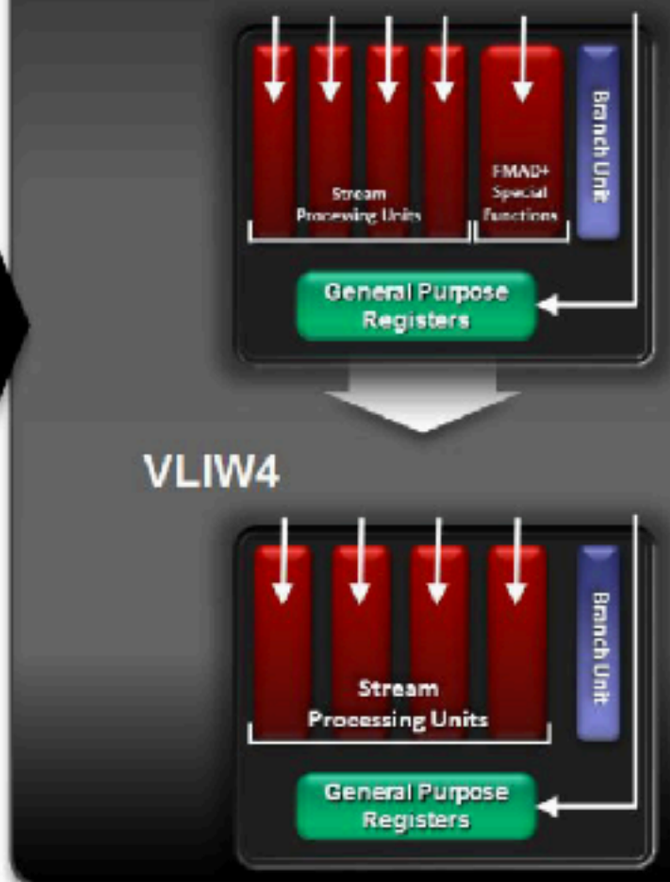
$$V_{tex}\begin{bmatrix} s \\ t \\ r \\ q \end{bmatrix} = M_{proj} \bullet V_{in}$$

Lighting

$$C_p = k_a L_a + \sum_{n=lights} Att_n (k_d (\hat{L}_n \bullet \hat{N}) + k_s (\hat{R}_n \bullet \hat{V})^{\alpha})$$

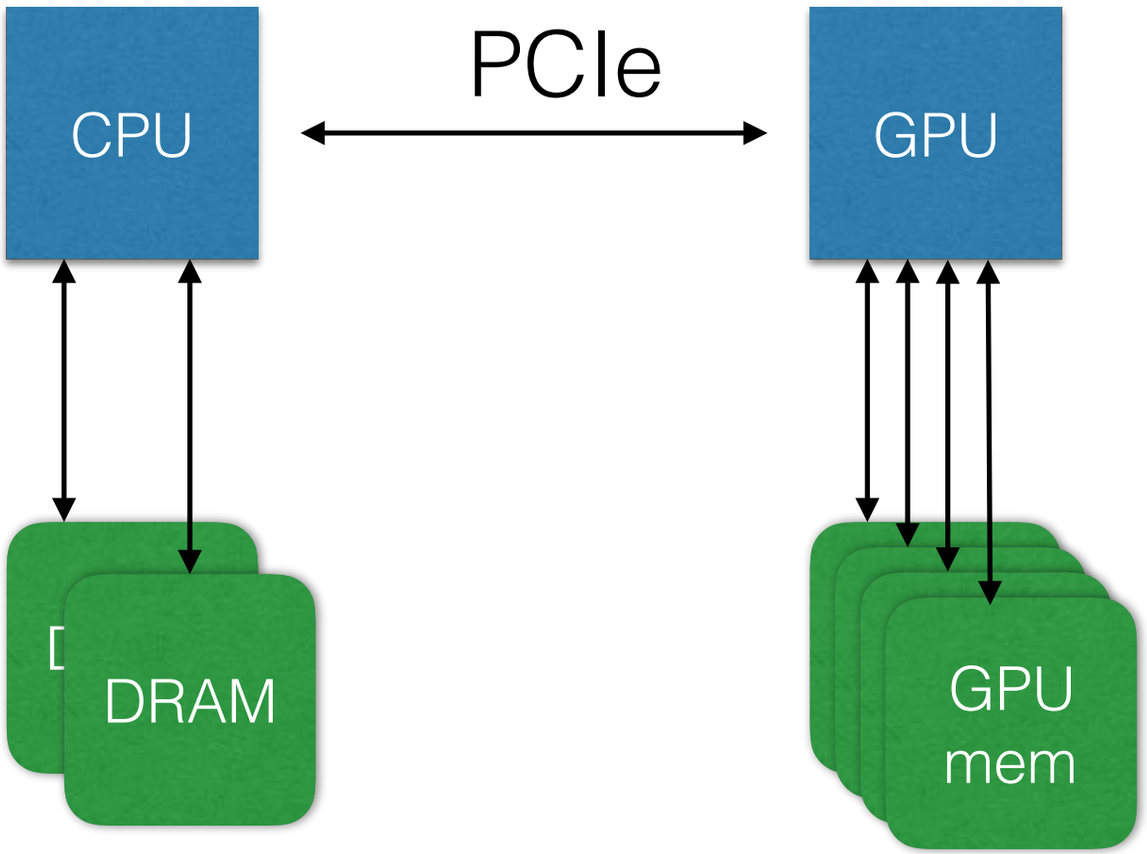
2ND ERA: Simple Shaders

3RD ERA: Graphics Parallel Core



GPUs have enormous power that is enormously difficult to use

- Nvidia GP100 - 5.3TFlops of double precision
    - This is equivalent to the fastest super computer in the world in 2001; put a single rack together of these and you would be in the top 500 (you could get there for under $1MM)
- Gianormous: 600mm^2
- 3,840 CUDA cores
- 720GB/s die-stacked DRAM (16GB total)
    - This is equivalent to about 11 DDR4 channels



- *partitioned* address space
  - (although this is changing!)
- CPU dispatches work to GPU
  GPU is not a first class compute device
  - ongoing research on this!

- **It's a process**
- ~ 2001 first hints of programmability
- ~ mid 2000's useful 32 bit math, sometime later 64 bit
- ~ 2013 unified virtual addressing
  - ~ 2015/6 and products that don't totally suck at it
- Active areas of research: GPU accesses to the filesystem network, etc.

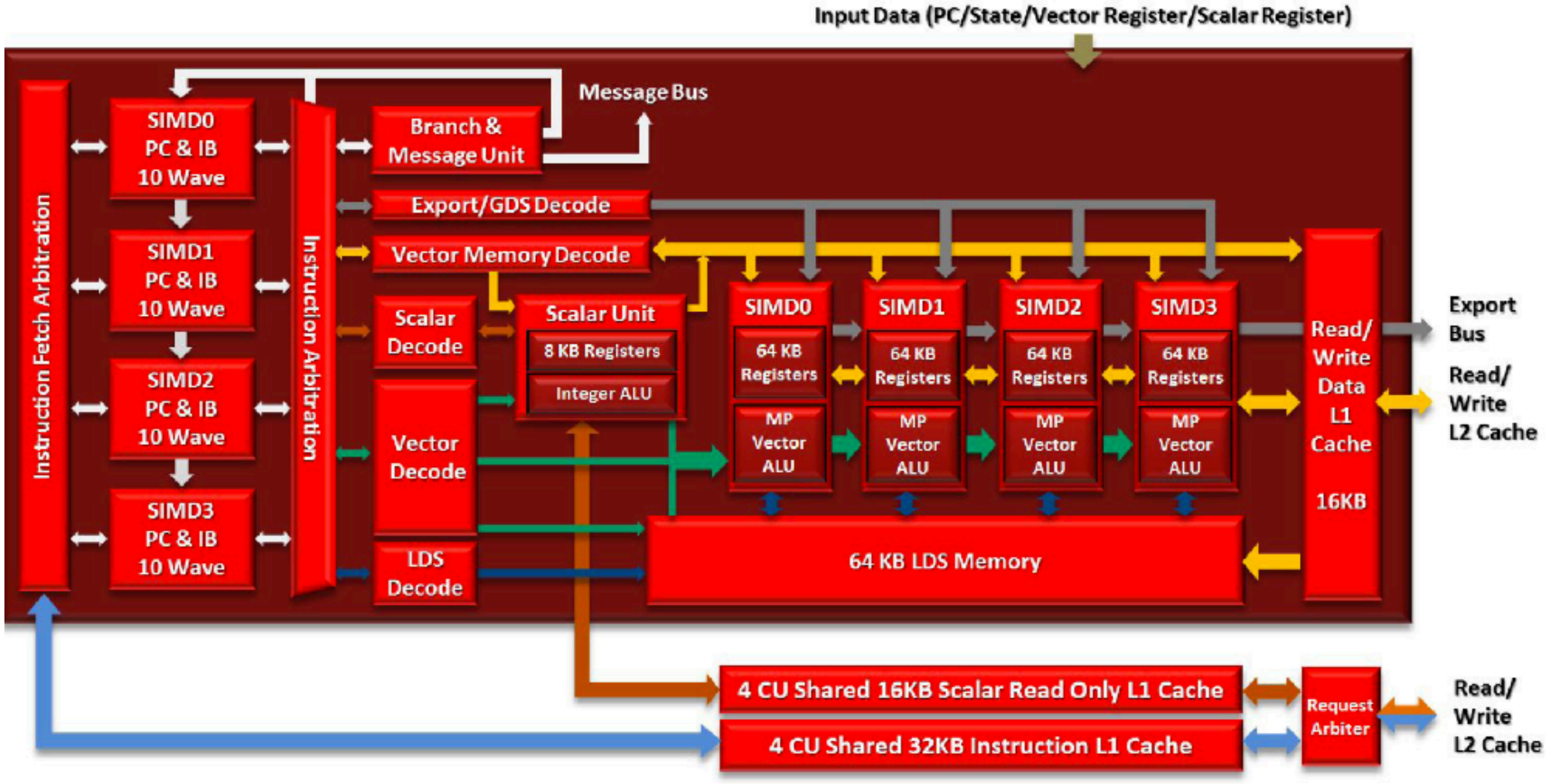
Some terminology…

- MIMD = Multiple Instruction, Multiple Data
    - Multicore
- SIMD = Single Instruction, Multiple Data
    - Vector
- FGMT = Fine-Grained Multithreading
- VLIW = Very Long Instruction Word
- Vector = what you think it means from Mathematics
- Bandwidth
- Injection Rate or Packet Rate or Message Rate
    - Peak Injection Rate = Peak Bandwidth / Smallest Packet Size
    - Peak Bandwidth = Largest Packet Size X Injection Rate (usually not peak)
- Channel or Bus
    - as in DDR3 channel, which for a typical rate has peak BW of ~18GB/s and Injection Rate of 280M Msg/S.

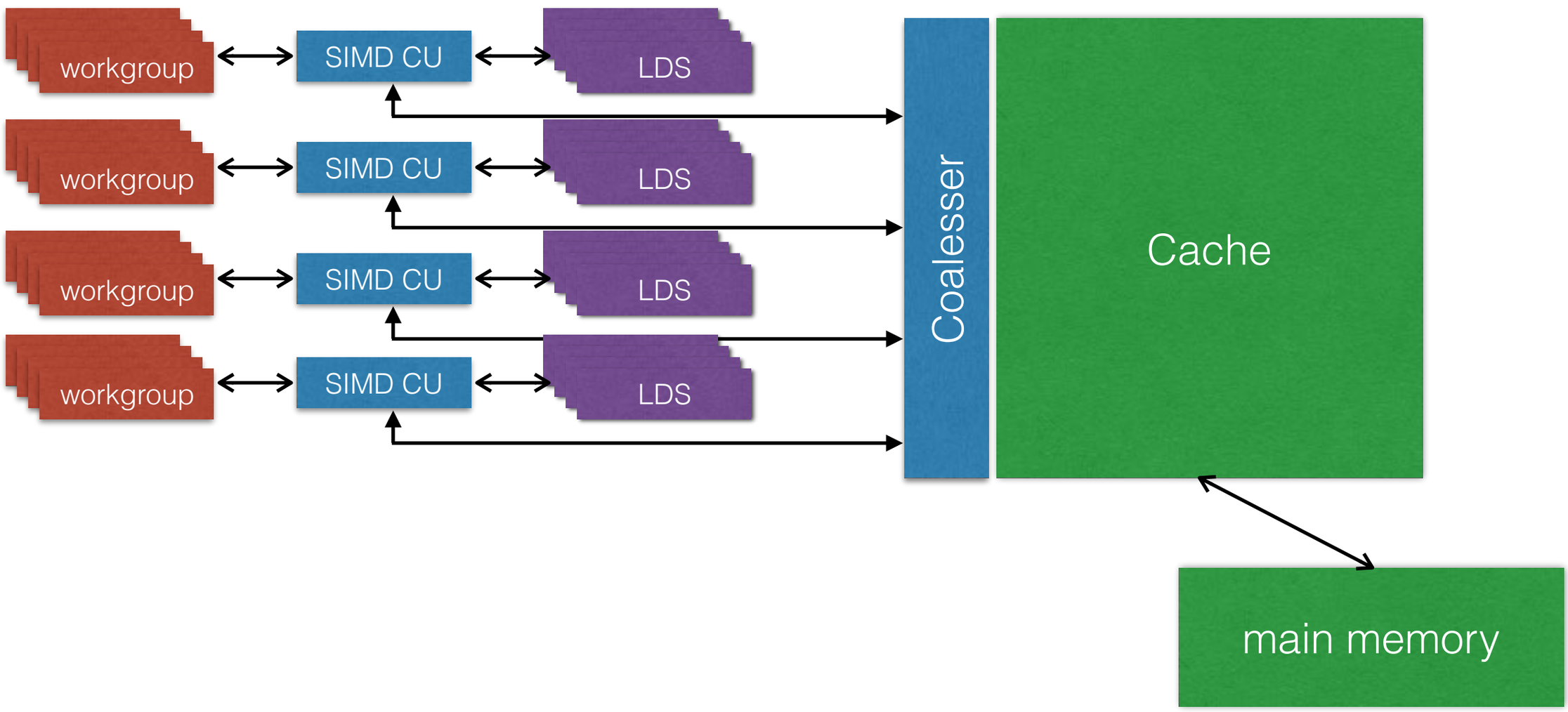
AMD / NVidia lingua franca

AMD	NVidia	Meaning
Workitem	Thread	a single task
Workgroup	Thread-block / CTA	a group of threads that can share data and synchronize locally
Wavefront	Warp	unit of hardware SIMT scheduling
VGPR (64x32 bit)	VGPR (32x32 bit)	Vector general purpose register
SGPR (32 bit)	N/A	Scalar general purpose register
Global memory	Global memory	Globally shared memory across all workitems/ threads
Local memory (LDS)	Shared memory	Software managed scratch-pad shared between a workgroup or thread-block. Per SM/CU
Private memory	Local memory	Workitem/thread private memory

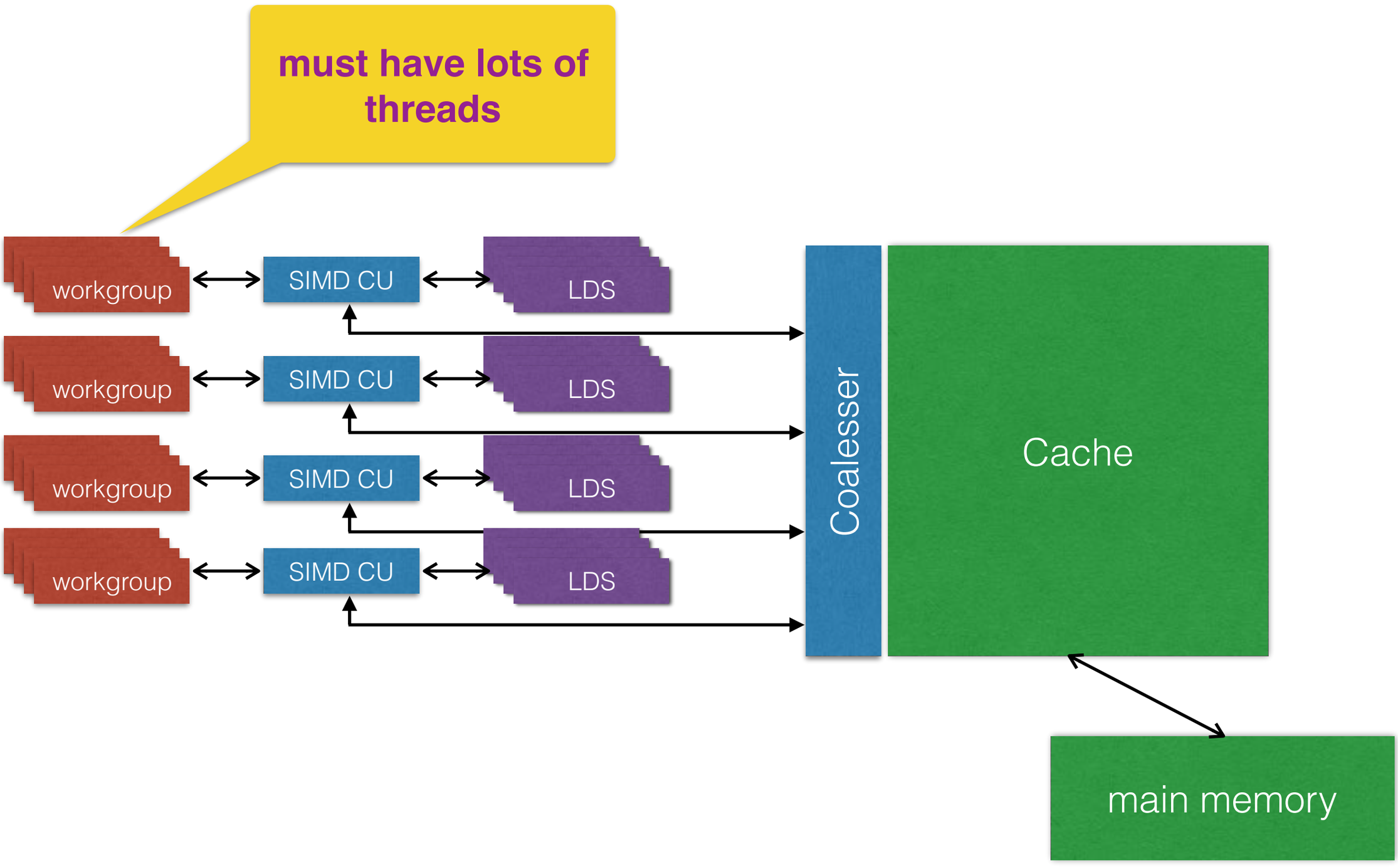
Mark Emily The person to ask…


Input Data (PC/State/Vector Register/Scalar Register)
Instruction Fetch Arbitration
SIMD0 PC & IB 10 Wave
SIMD1 PC & IB 10 Wave
SIMD2 PC & IB 10 Wave
SIMD3 PC & IB 10 Wave
Instruction Arbitration
Message Bus
Branch & Message Unit
Export/GDS Decode
Vector Memory Decode
Scalar Decode
Vector Decode
LDS Decode
Scalar Unit
8 KB Registers
Integer ALU
SIMD0
SIMD1
SIMD2
SIMD3
64 KB Registers
MP Vector ALU
64 KB LDS Memory
Read/ Write Data L1 Cache 16KB
Export Bus
Read/ Write L2 Cache
4 CU Shared 16KB Scalar Read Only L1 Cache
4 CU Shared 32KB Instruction L1 Cache
Request Arbiter
Read/ Write L2 Cache

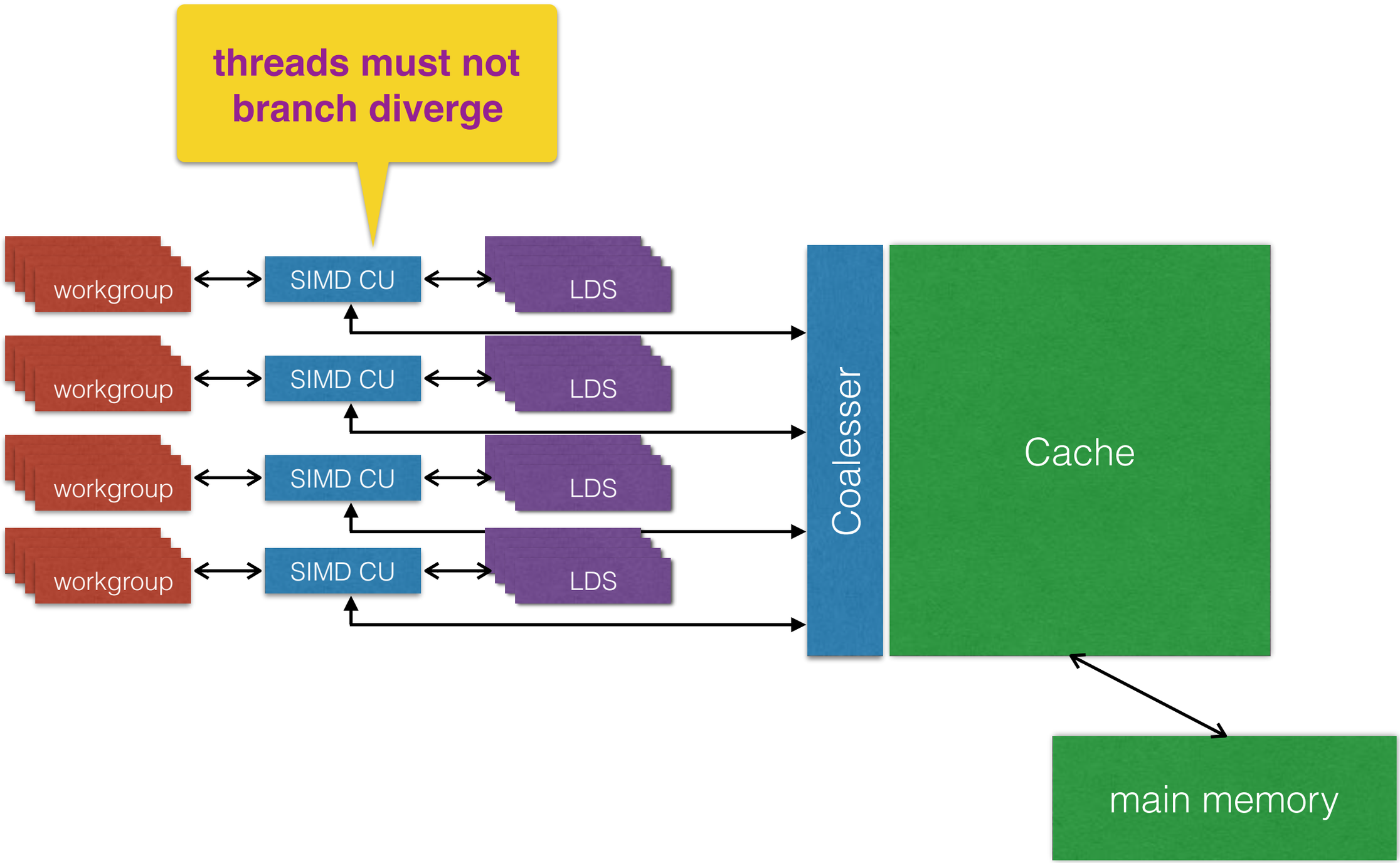
A simplified perspective on GPU architecture



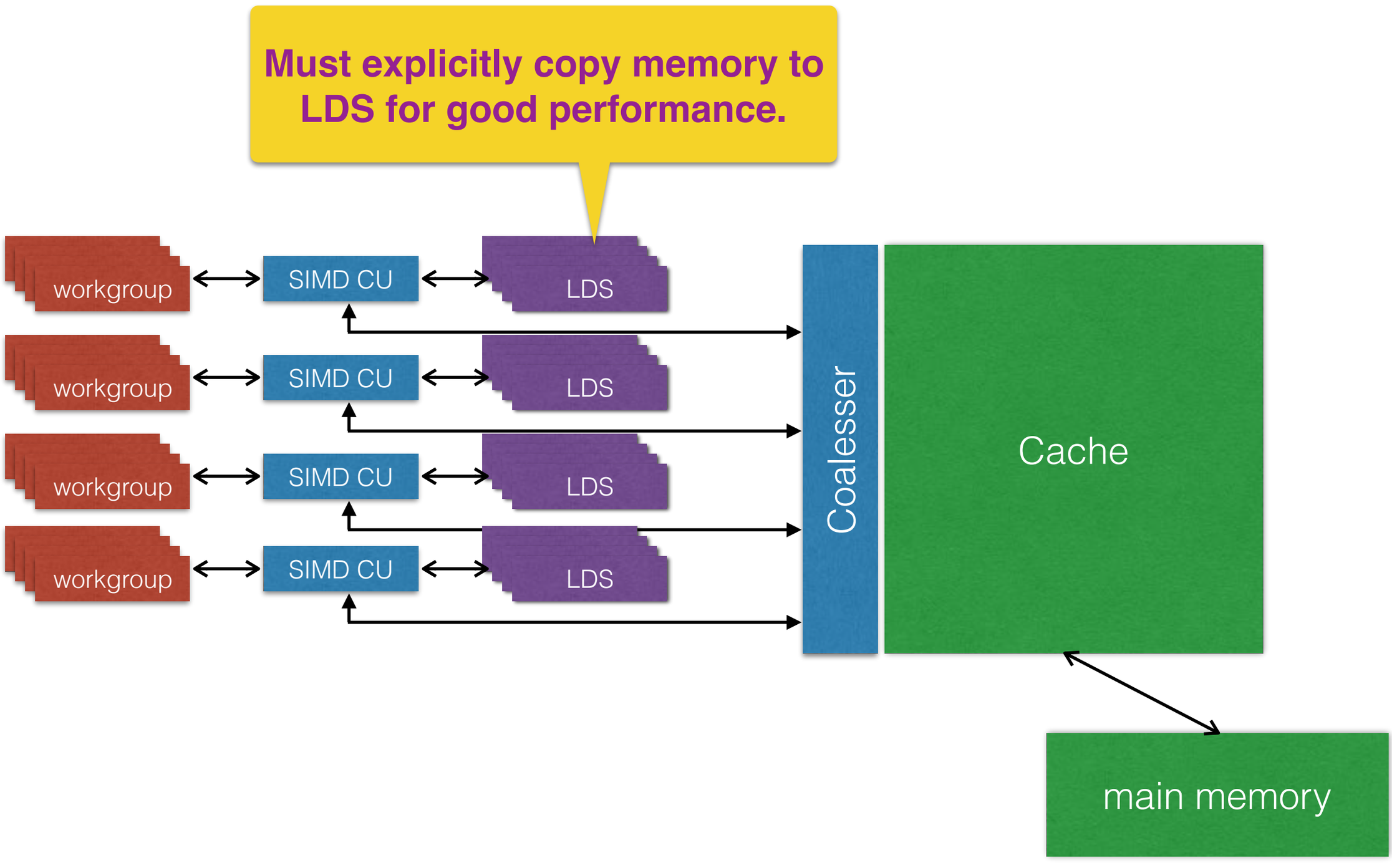
A simplified perspective on GPU architecture



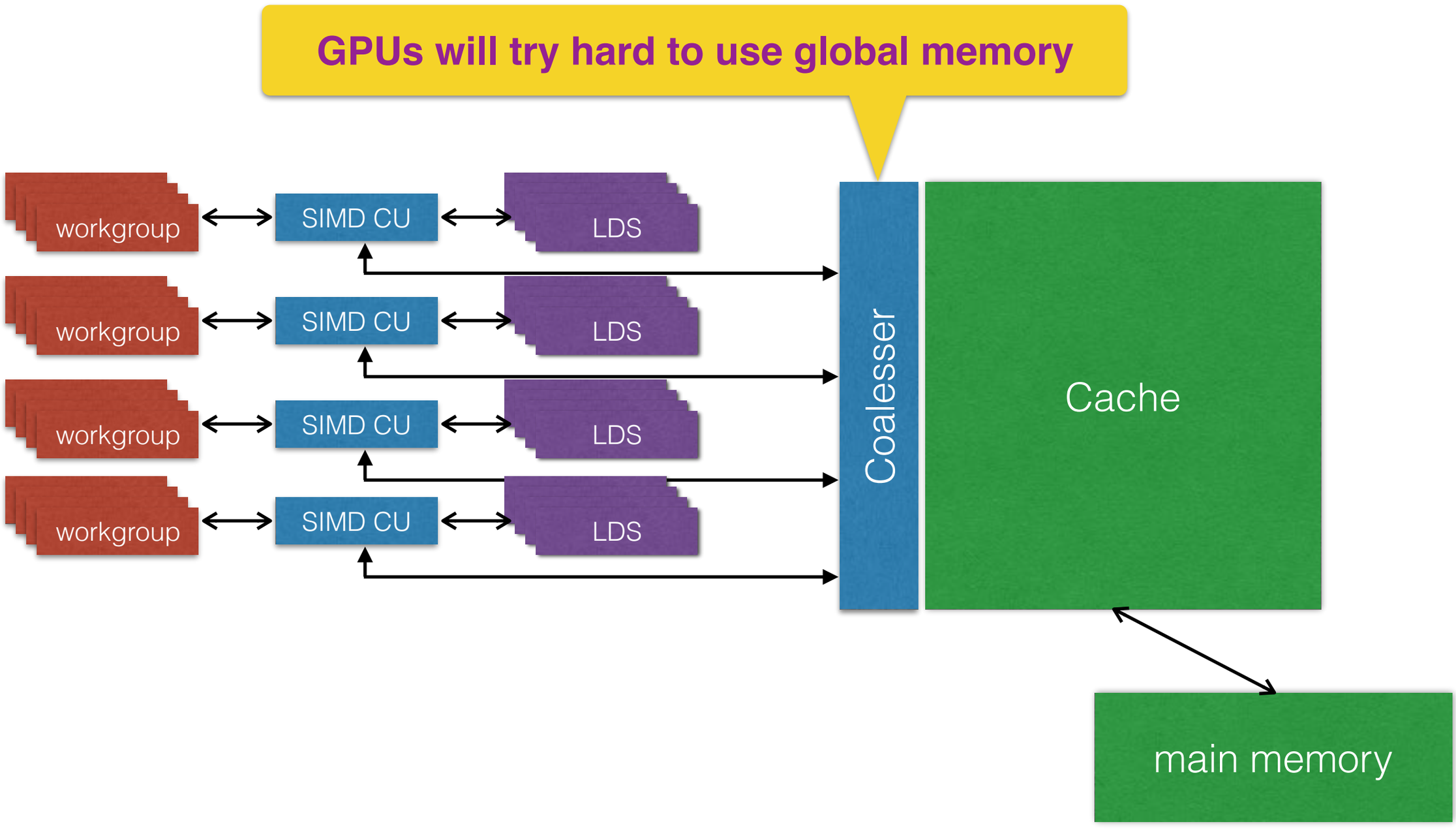
A simplified perspective on GPU architecture



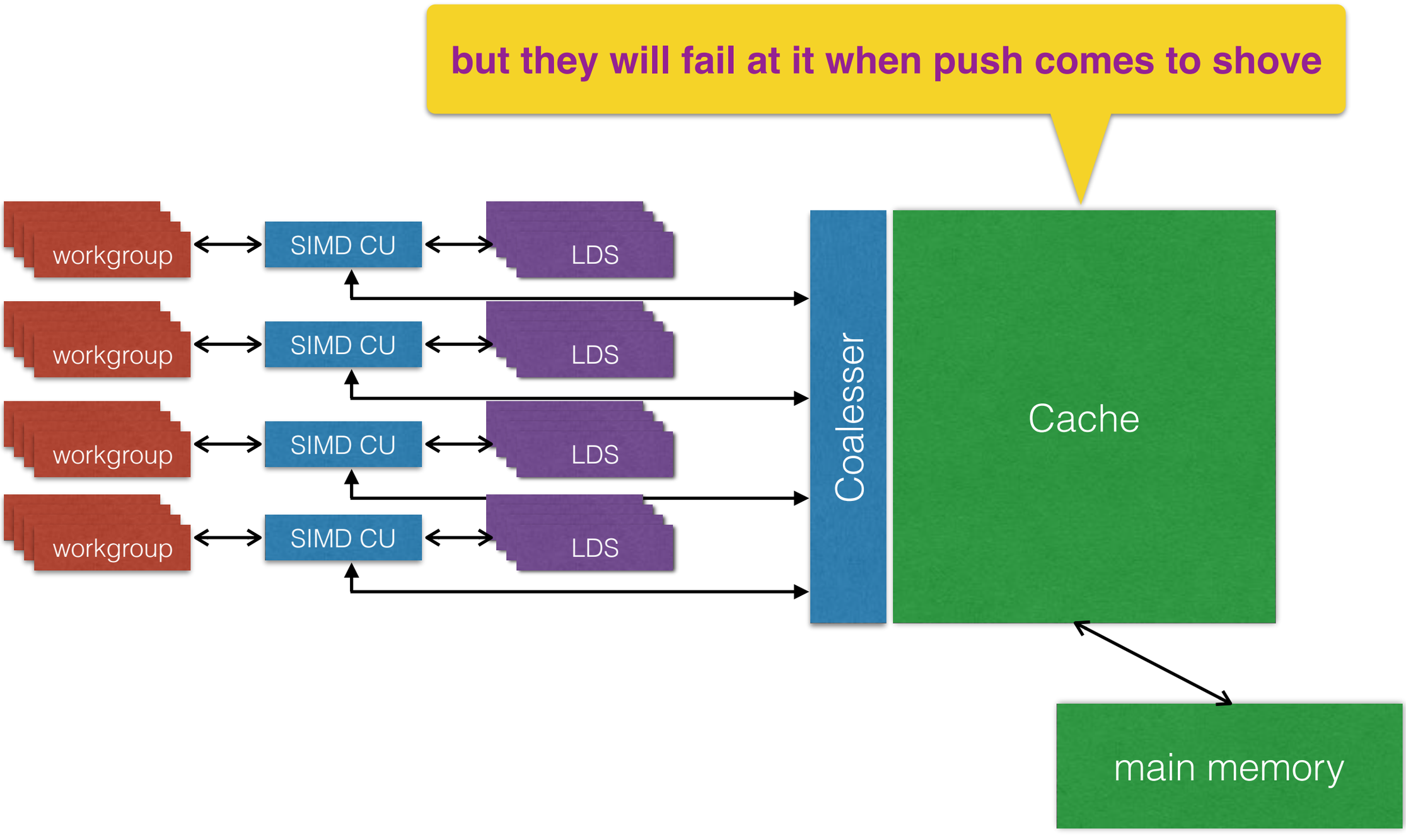
A simplified perspective on GPU architecture



A simplified perspective on GPU architecture



A simplified perspective on GPU architecture



Toolkit Zoo

- OpenCL
- C++AMP
  - hcc
- CUDA
- HIP
- GCD

After much consternation I decided to focus class lecture on OpenCL. This is because it is supported by AMD, NVidia and Intel and works on Mac OS X, Windows and Linux.

Personally, I like this. But it's a work in progress still and doesn't work with NVidia.

Like C++Amp but I don't have time/ infrastructure

No one cares

To first order, no one cares

Core concepts

- Memory, Memory, Memory
  - Memory hierarchy
- SIMD execution
- Threads for latency tolerance

An important mindset

Execution is free, data access is not

Energy Shopping List

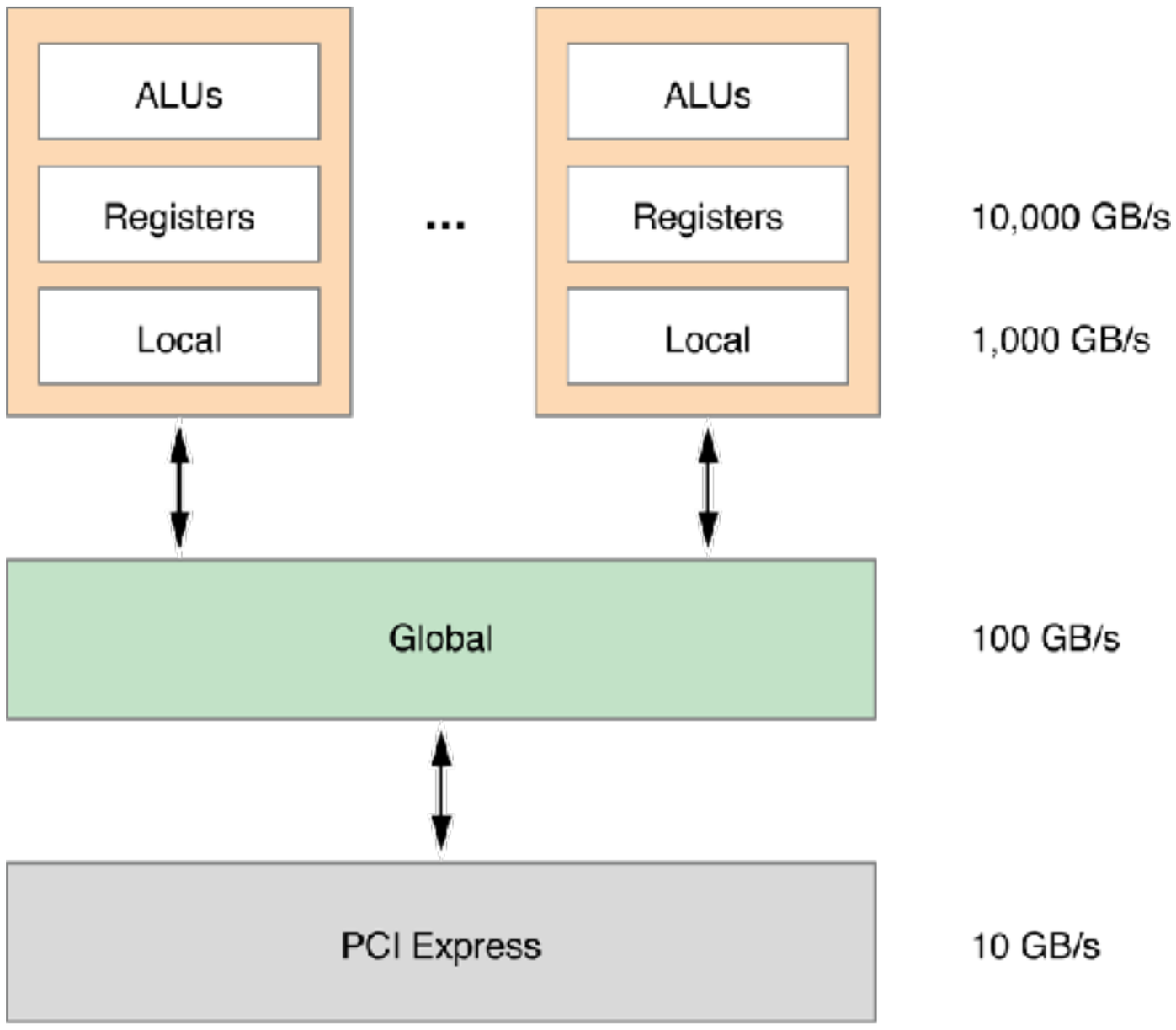
Processor Technology	40 nm	10nm
Vdd (nominal)	0.9 V	0.7 V
DFMA energy	50 pJ	7.6 pJ
64b 8 KB SRAM Rd	14 pJ	2.1 pJ
Wire energy (256 bits, 10mm)	310 pJ	174 pJ

Memory Technology	45 nm	16nm
DRAM interface pin bandwidth	4 Gbps	50 Gbps
DRAM interface energy	20-30 pJ/bit	2 pJ/bit
DRAM access energy	8-15 pJ/bit	2.5 pJ/bit

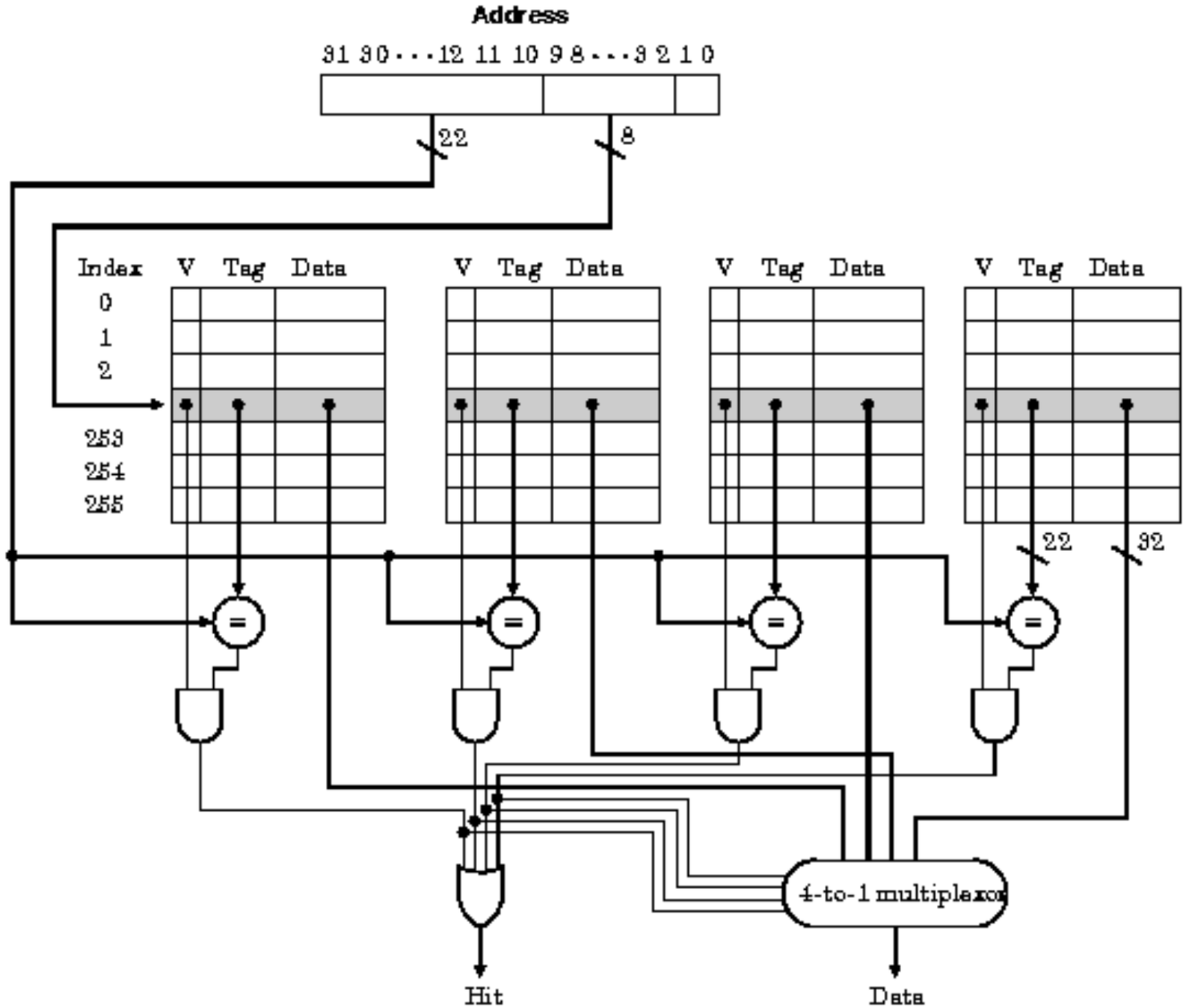
FP Op lower bound
=
4 pJ

Keckler [Micro 2011], Vogelsang [Micro 2010]

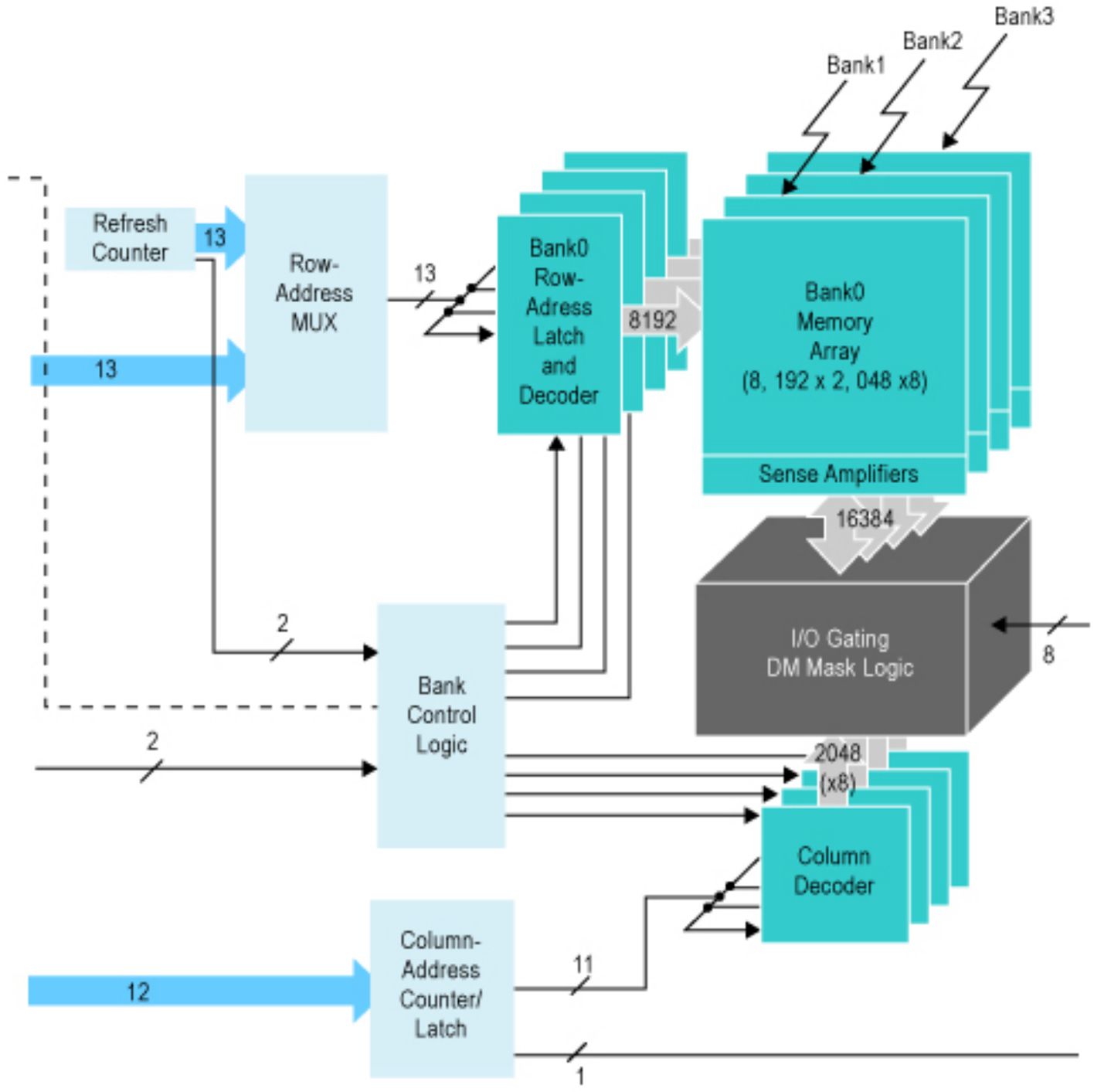
Memory Hierarchy



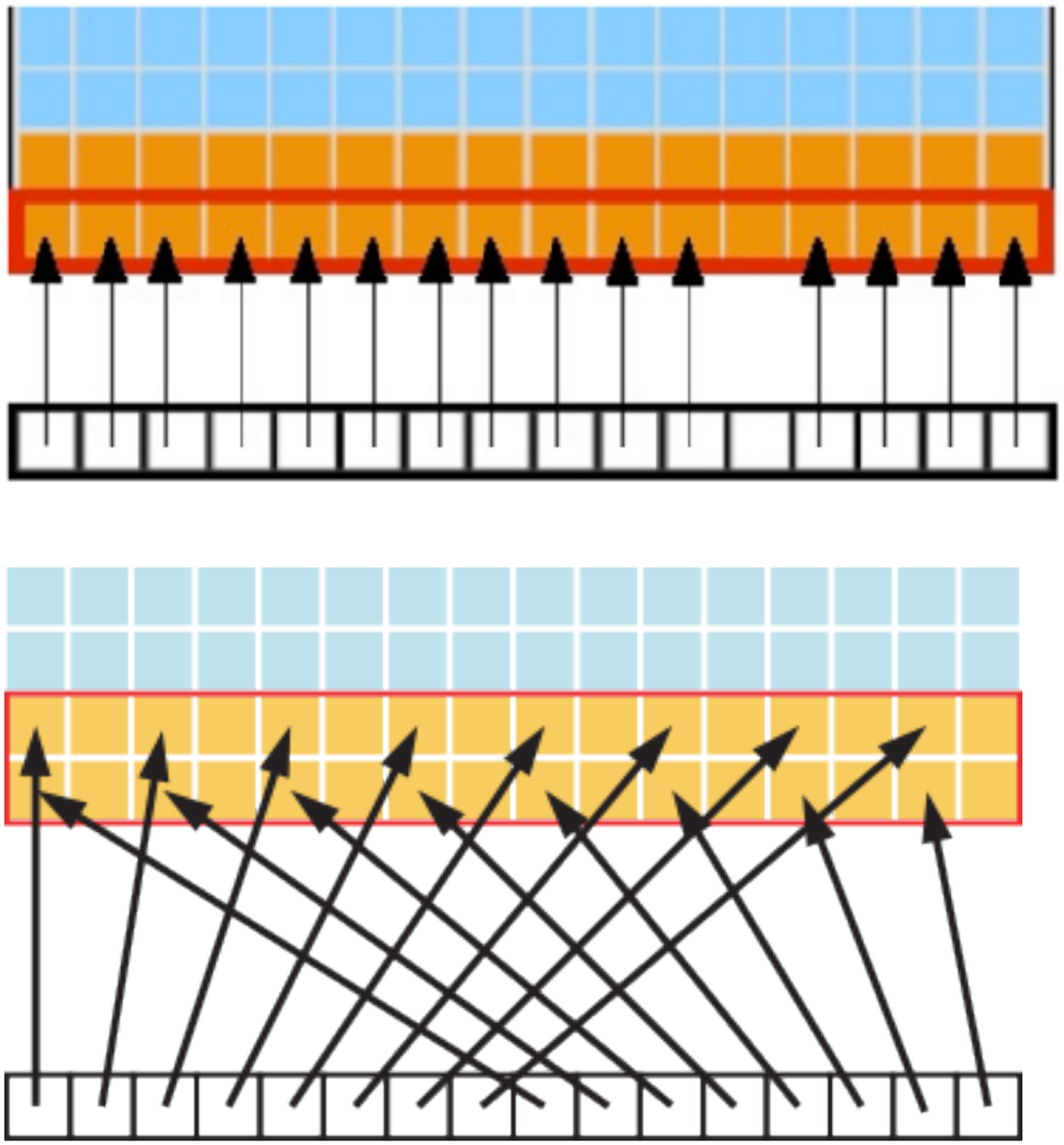
Let’s talk about caches….

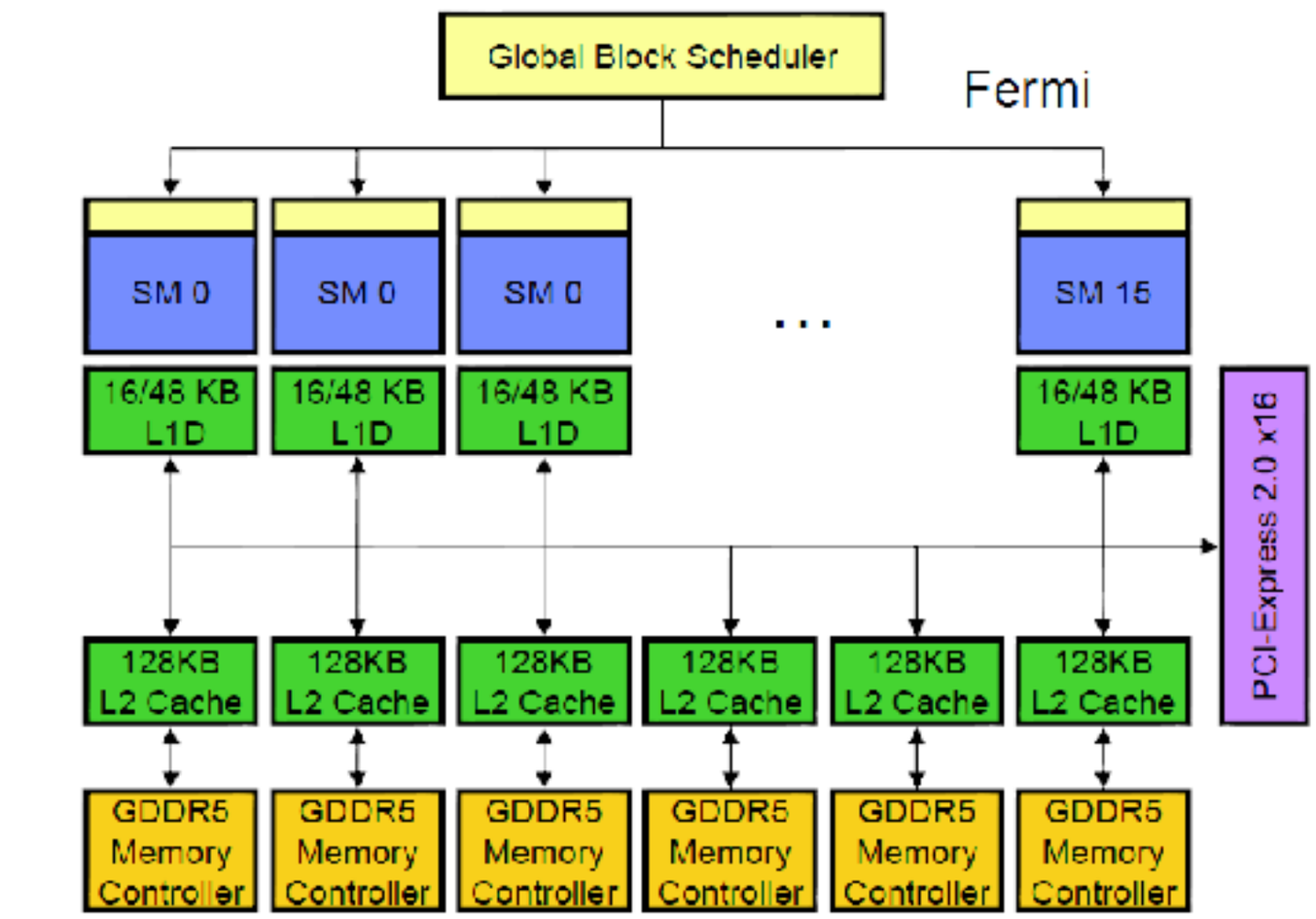


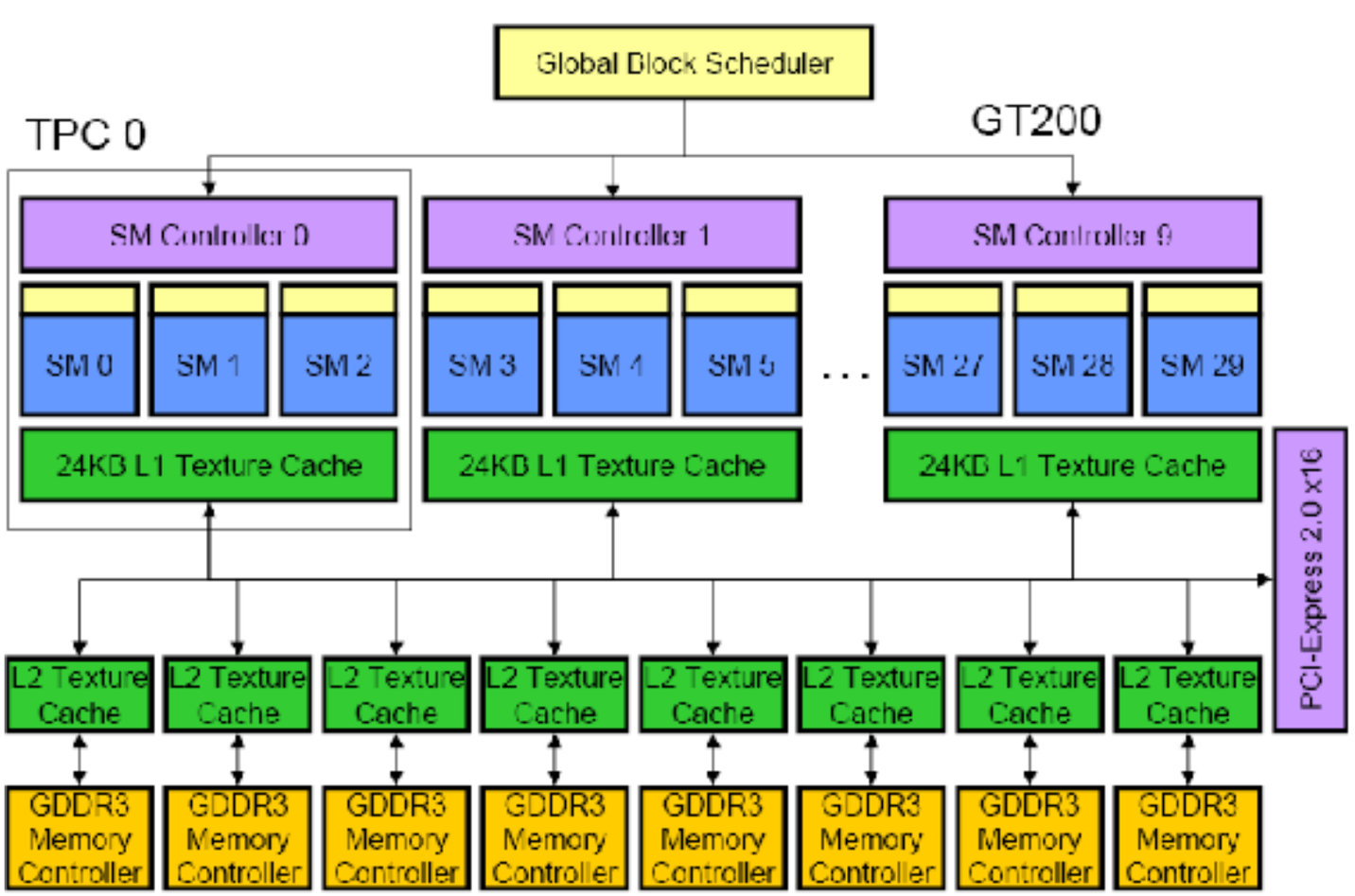
Let’s talk about DRAM

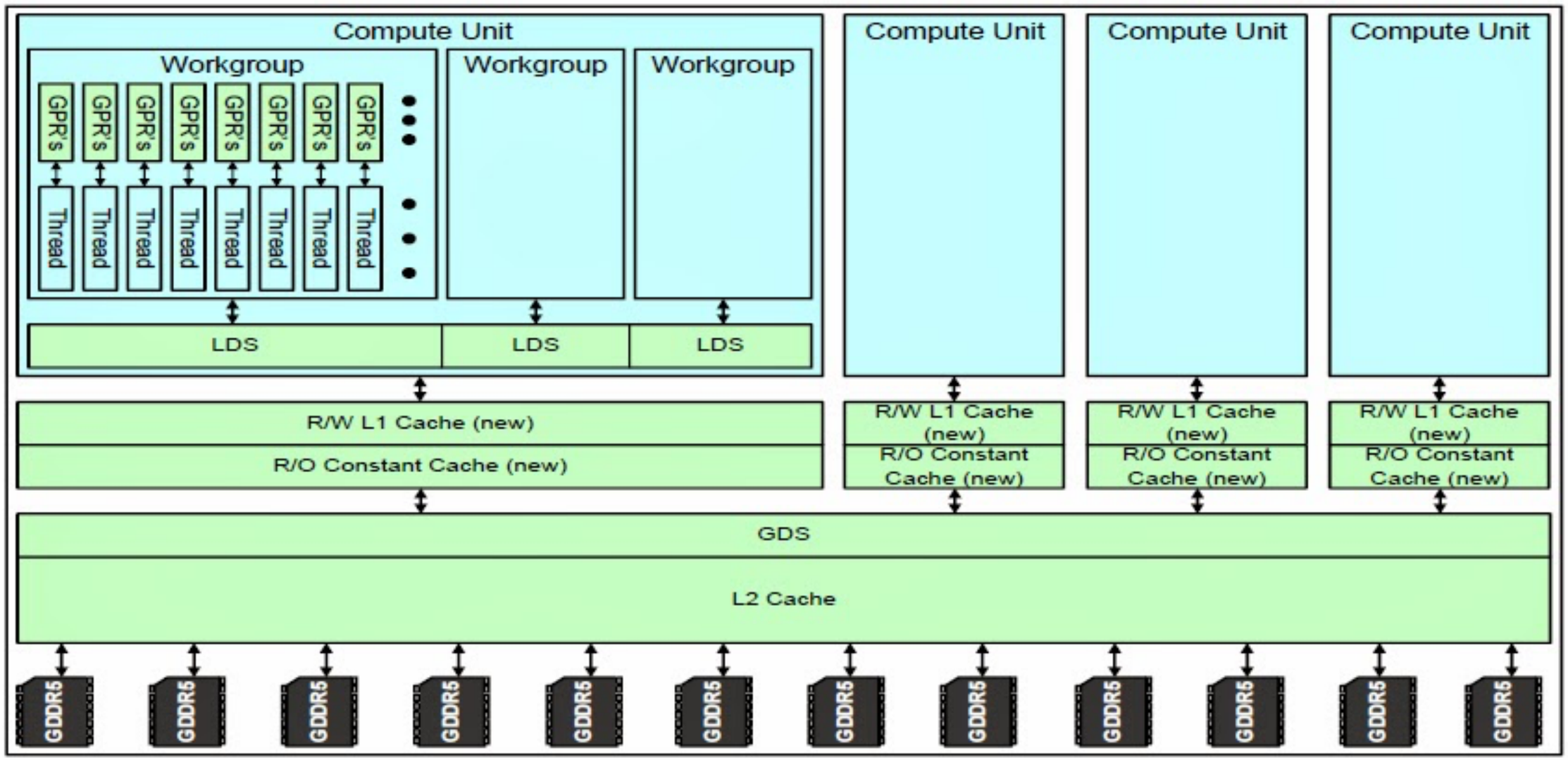


LDS



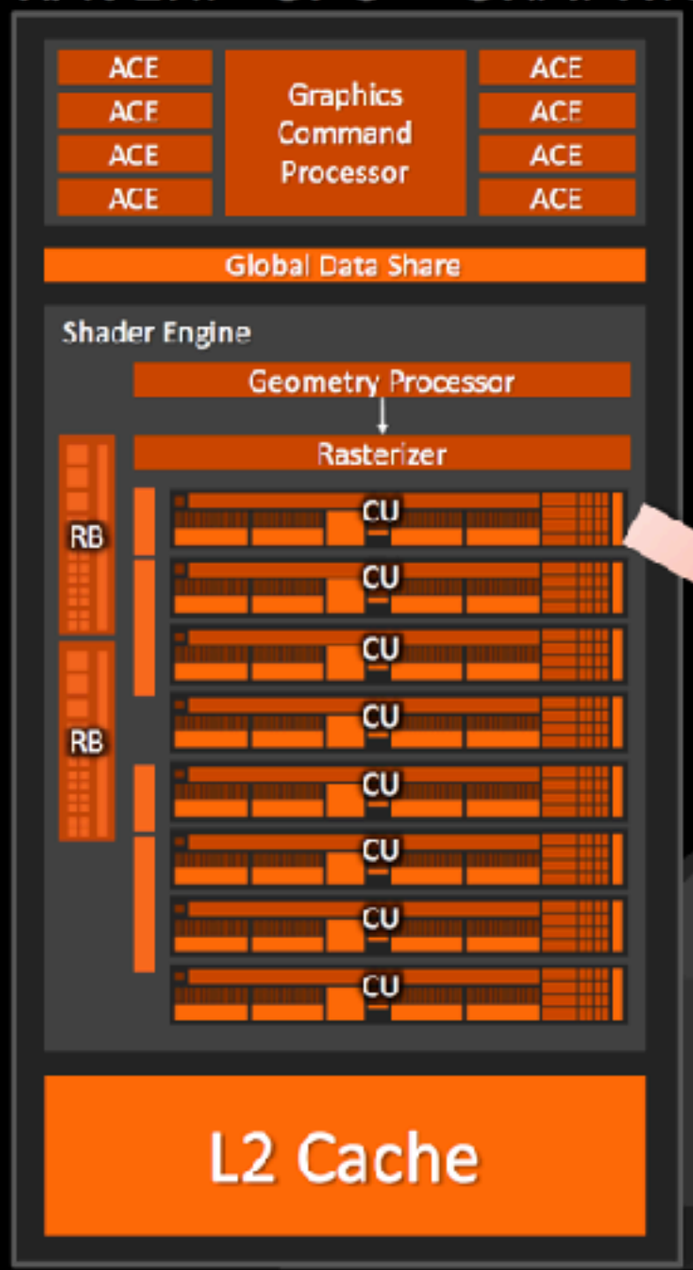

Global Block Scheduler
Fermi
SM 0
SM 0
SM 0
SM 15
16/48 KB L1D
16/48 KB L1D
16/48 KB L1D
16/48 KB L1D
128KB L2 Cache
128KB L2 Cache
128KB L2 Cache
128KB L2 Cache
128KB L2 Cache
128KB L2 Cache
GDDR5 Memory Controller
GDDR5 Memory Controller
GDDR5 Memory Controller
GDDR5 Memory Controller
GDDR5 Memory Controller
GDDR5 Memory Controller
PCI-Express 2.0 x16


Global Block Scheduler
TPC 0
GT200
SM Controller 0
SM Controller 1
SM Controller 9
SM 0
SM 1
SM 2
SM 3
SM 4
SM 5
SM 27
SM 28
SM 29
24KB L1 Texture Cache
24KB L1 Texture Cache
24KB L1 Texture Cache
L2 Texture Cache
L2 Texture Cache
L2 Texture Cache
L2 Texture Cache
L2 Texture Cache
L2 Texture Cache
L2 Texture Cache
L2 Texture Cache
GDDR3 Memory Controller
GDDR3 Memory Controller
GDDR3 Memory Controller
GDDR3 Memory Controller
GDDR3 Memory Controller
GDDR3 Memory Controller
GDDR3 Memory Controller
GDDR3 Memory Controller
PCI-Express 2.0 x16


Compute Unit
Workgroup
Workgroup
Workgroup
GPR's
Thread
LDS
LDS
LDS
R/W L1 Cache (new)
R/O Constant Cache (new)
Compute Unit
R/W L1 Cache (new)
R/O Constant Cache (new)
Compute Unit
R/W L1 Cache (new)
R/O Constant Cache (new)
Compute Unit
R/W L1 Cache (new)
R/O Constant Cache (new)
GDS
L2 Cache
GDDR5

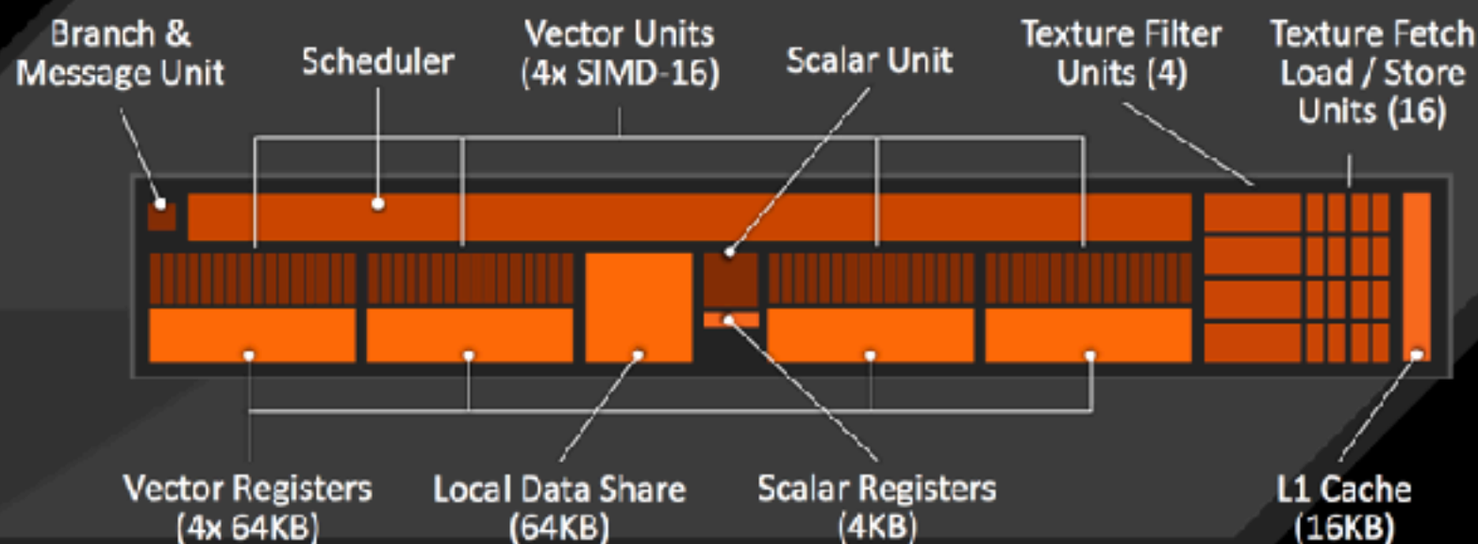
"KAVERI" GPU – GRAPHICS CORE NEXT ARCHITECTURE

AMD



47% of "Kaveri" is dedicated for GPU

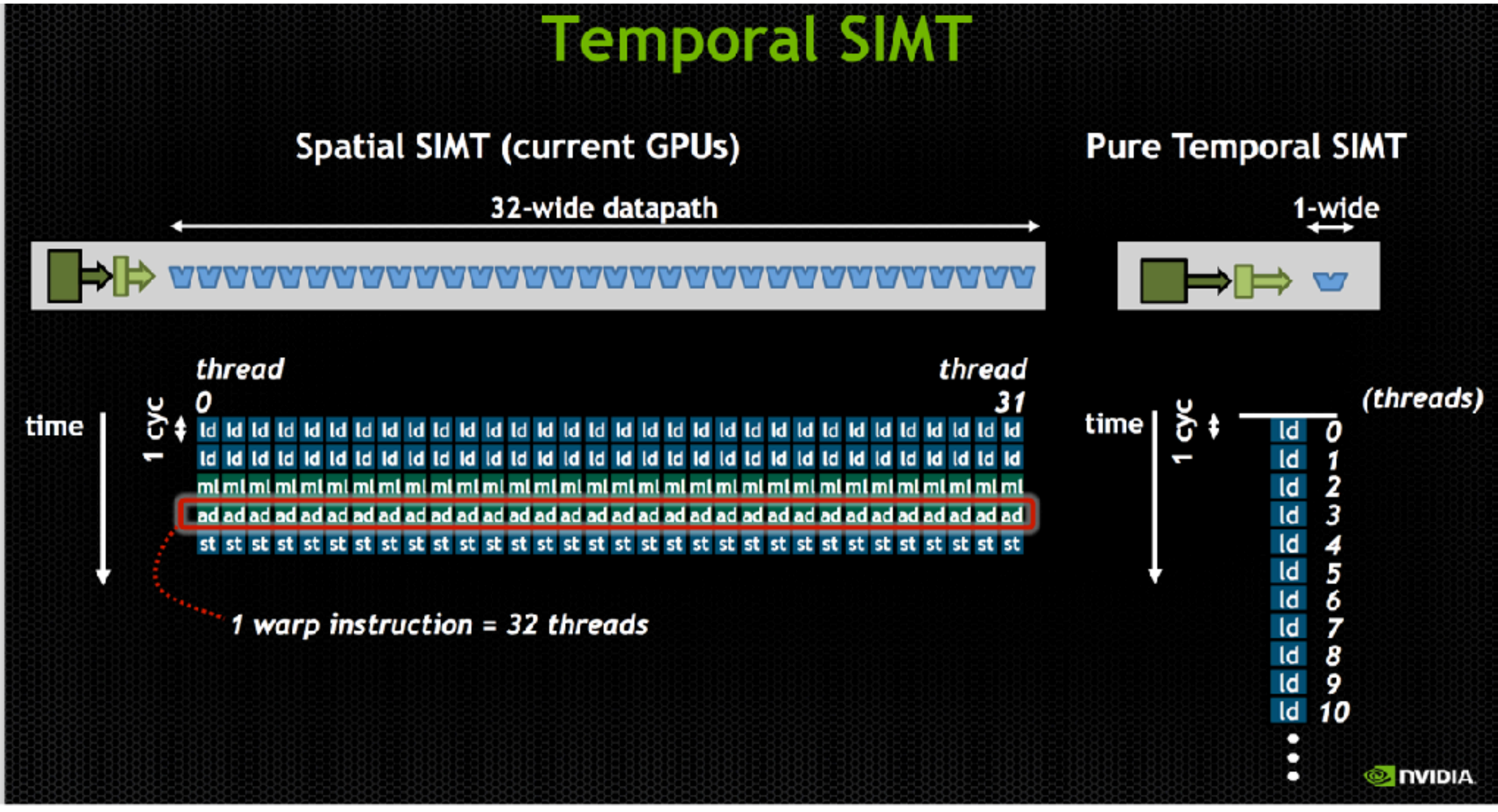
- 8 compute units (512 IEEE 2008-compliant shaders)
- Device flat (generic) addressing support
- Masked Quad Sum of Absolute Difference (MQSAD) with 32b accumulation and saturation
- Precision improvement for native LOG/EXP ops to 1ULP

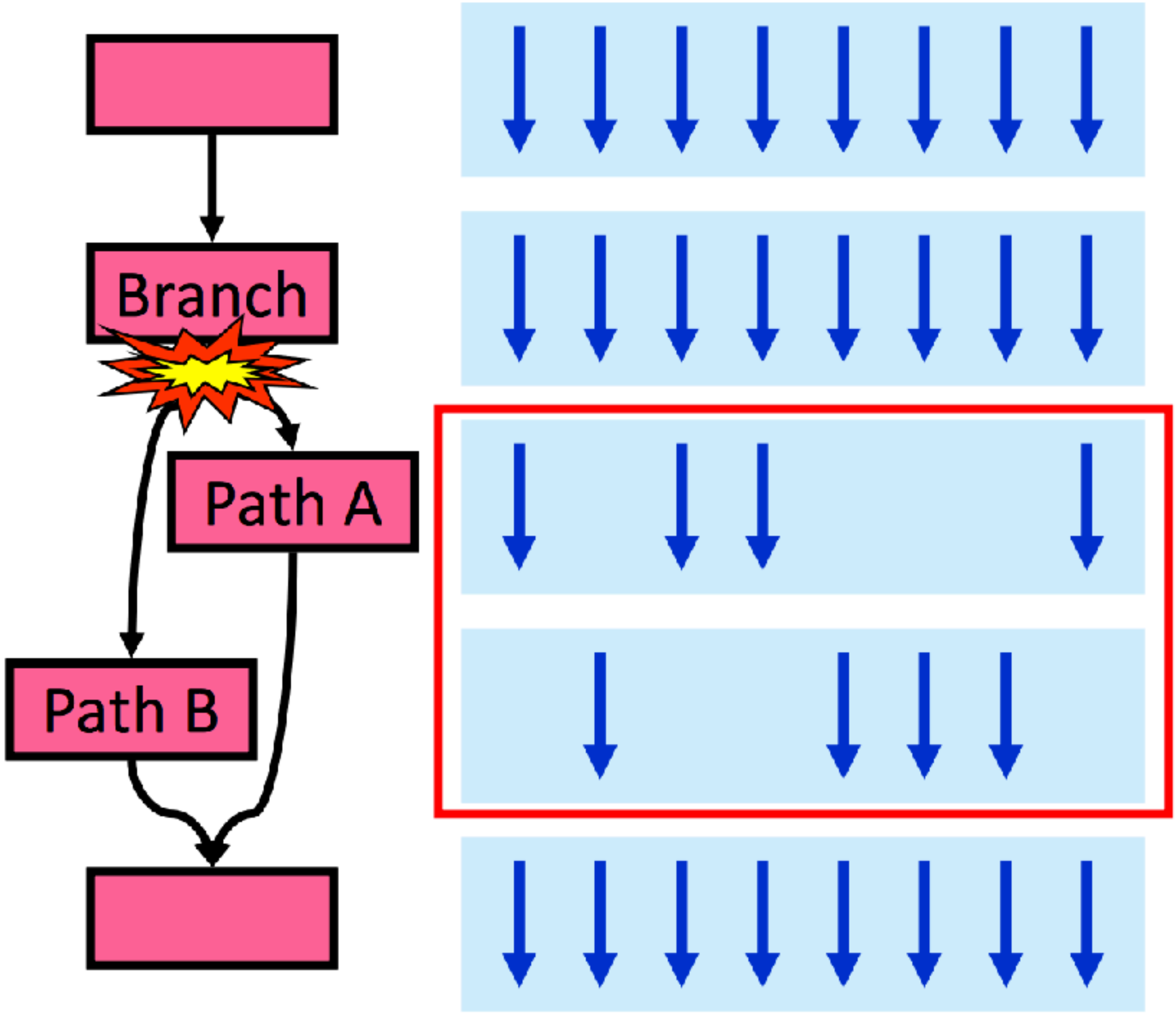


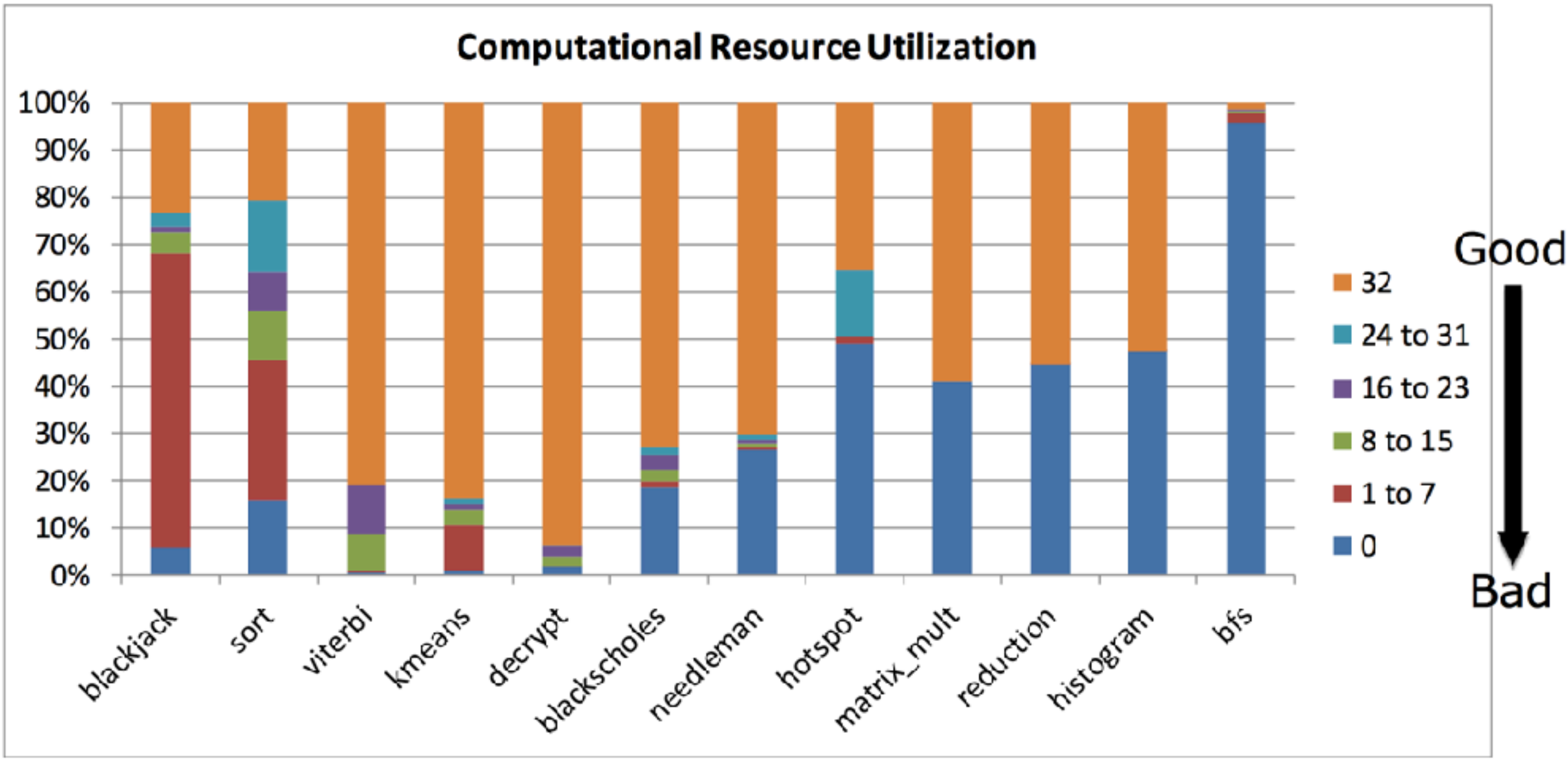
Some core thoughts to keep in mind

- every access to a cache or DRAM accesses a **block**.
- In front of the L2 on a GPU is a structure that coalesces accesses to the same block.
- For good L2 performance it is key that you use this
  - Easiest to use it by accessing the same block in *different* work items.
- The L1 is fine-grained interleaved, but the net/net conceptually is the same for you as a developer.

Temporal SIMT




Branch
Path A
Path B


Computational Resource Utilization
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
blackjack
sort
viterbi
kmeans
decrypt
blackscholes
needleman
hotspot
matrix_mult
reduction
histogram
bfs
32
24 to 31
16 to 23
8 to 15
1 to 7
0
Good
Bad

Thread scheduling

- There's nothing to say. It is implementation dependent
  - ***Do not write code that assumes anything being scheduled or completed across workgroups.***
- Within a workgroup:
  - barrier(…)

Example

A.2 High-Priority Recommendations

- To get the maximum benefit from OpenCL, focus first on finding ways to parallelize sequential code. (Section 1.1.3)
- Use the effective bandwidth of your computation as a metric when measuring performance and optimization benefits. (Section 2.2)
- Minimize data transfer between the host and the device, even if it means running some kernels on the device that do not show performance gains when compared with running them on the host CPU. (Section 3.1)
- Ensure global memory accesses are coalesced whenever possible. (Section 3.2.1)
- Minimize the use of global memory. Prefer shared memory access where possible. (Section 5.2)
- Avoid different execution paths within the same warp. (Section 6.1)
- Use the -cl-mad-enable build option. (Chapter 5)

A.3 Medium-Priority Recommendations

- Judiciously use "pinned" memory for host buffers (Section 3.1.1)
- Where feasible and for applications where it is effective, overlap host – device memory transfers with device computations and asynchronous host activities (Sections 3.1.2 and 3.1.3)
- For applications where the destination of computational results is the display, use OpenCL-OpenGL or OpenCL-D3D interop.
- Accesses to shared memory should be designed to avoid serializing requests due to bank conflicts. (Section 3.2.2.1)
- Use shared memory to avoid redundant transfers from global memory. (Section 3.2.2.2)
- To hide latency arising from register dependencies, maintain at least 25 percent occupancy on devices with compute capability 1.1 and lower, and 18.75 percent occupancy on later devices. (Section 4.3)
- The number of threads per block should be a multiple of 32 threads, because this provides optimal computing efficiency and facilitates coalescing. (Section 4.4)
- Use the native math library whenever speed trumps precision. (Section 5.1.4)

A.4 Low-Priority Recommendations

- For kernels with long argument lists, place some arguments into constant memory to save shared memory. (Section 3.2.2.4)
- Use shift operations to avoid expensive division and modulo calculations. (Section 5.1.1)
- Avoid automatic conversion of doubles to floats. (Section 5.1.3)
- Make it easy for the compiler to use branch predication in lieu of loops or control statements. (Section 6.2)

Some closing thoughts

- GPU's are in a sweet-enough spot between efficiency and pain
  - FPGAs = more pain, CPUS = less efficient