

AMD GPU Architecture

OpenCL™ Tutorial, PPAM 2009

Dominik Behr | September 13th, 2009

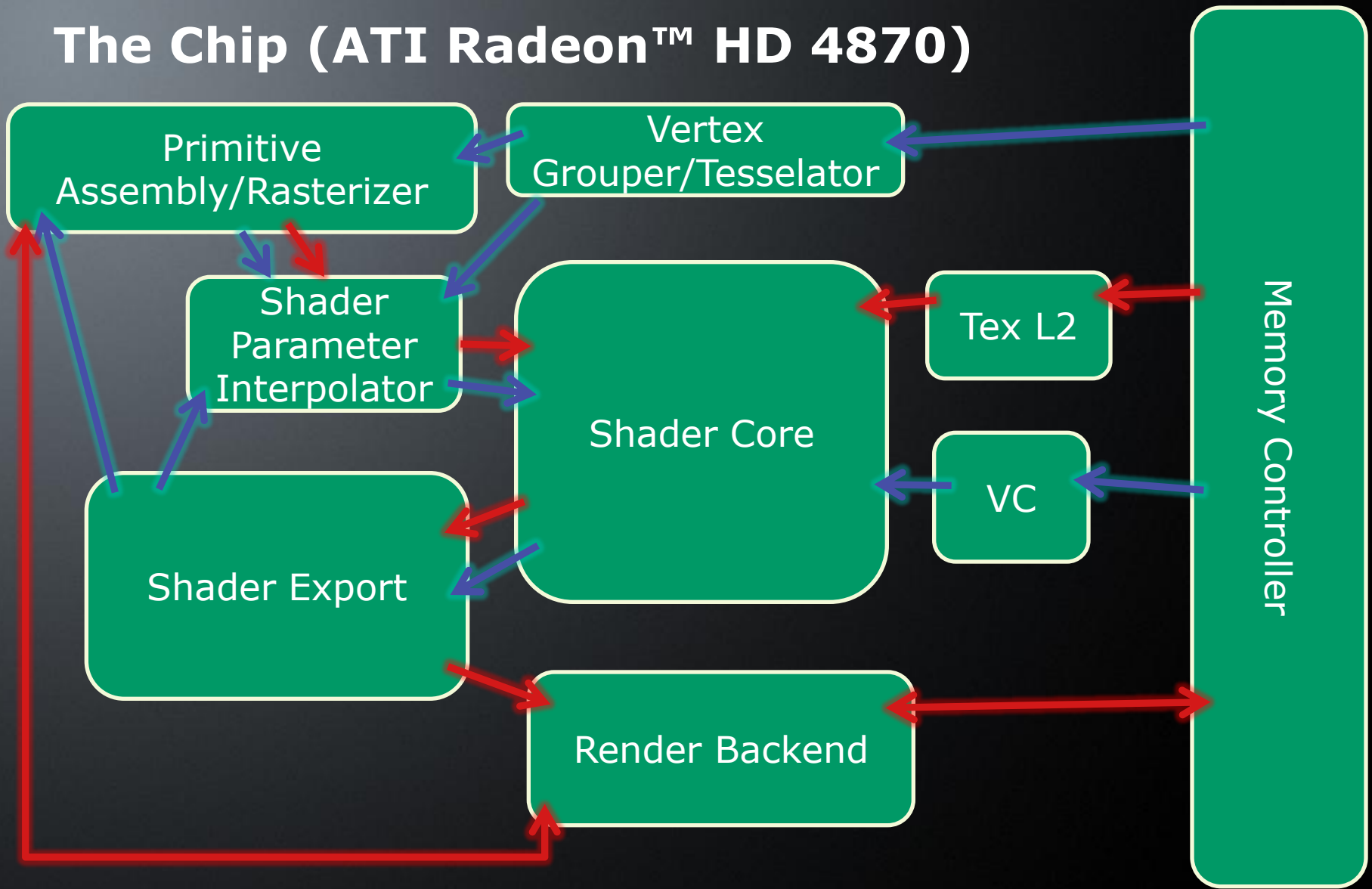


Overview

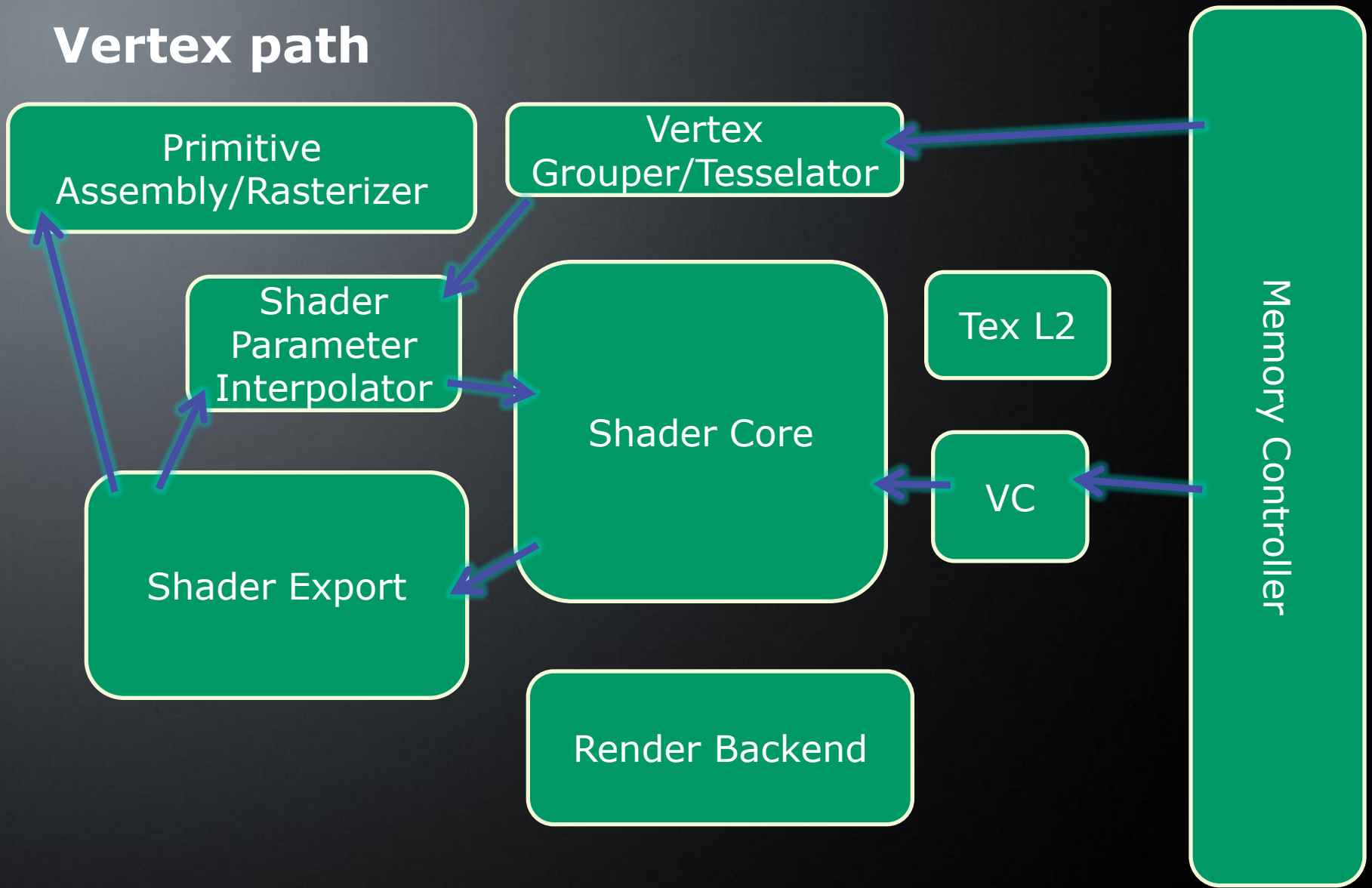
- AMD GPU architecture
- How OpenCL maps on GPU and CPU
- How to optimize for AMD GPUs and CPUs in OpenCL



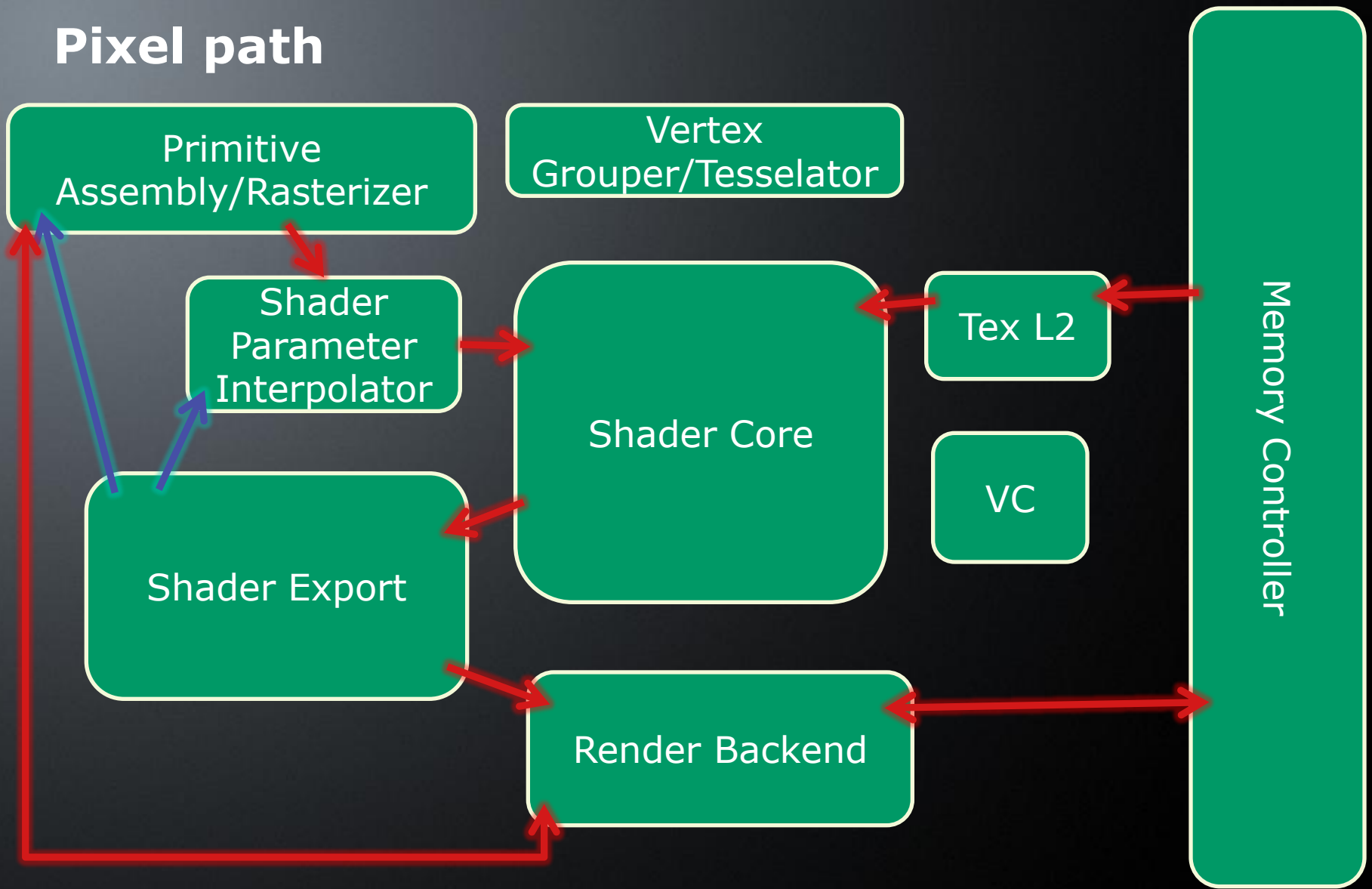
The Chip (ATI Radeon™ HD 4870)



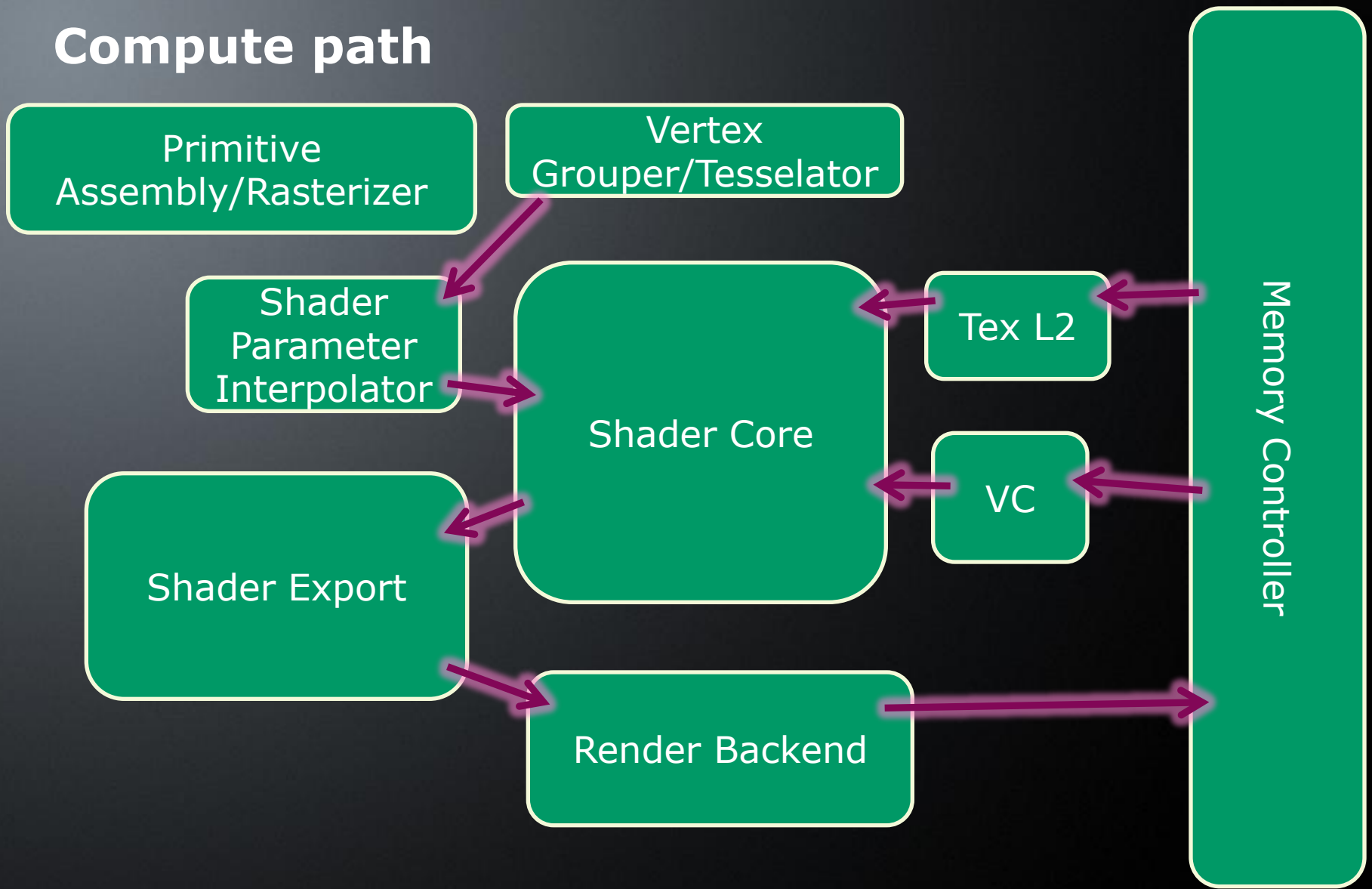
Vertex path



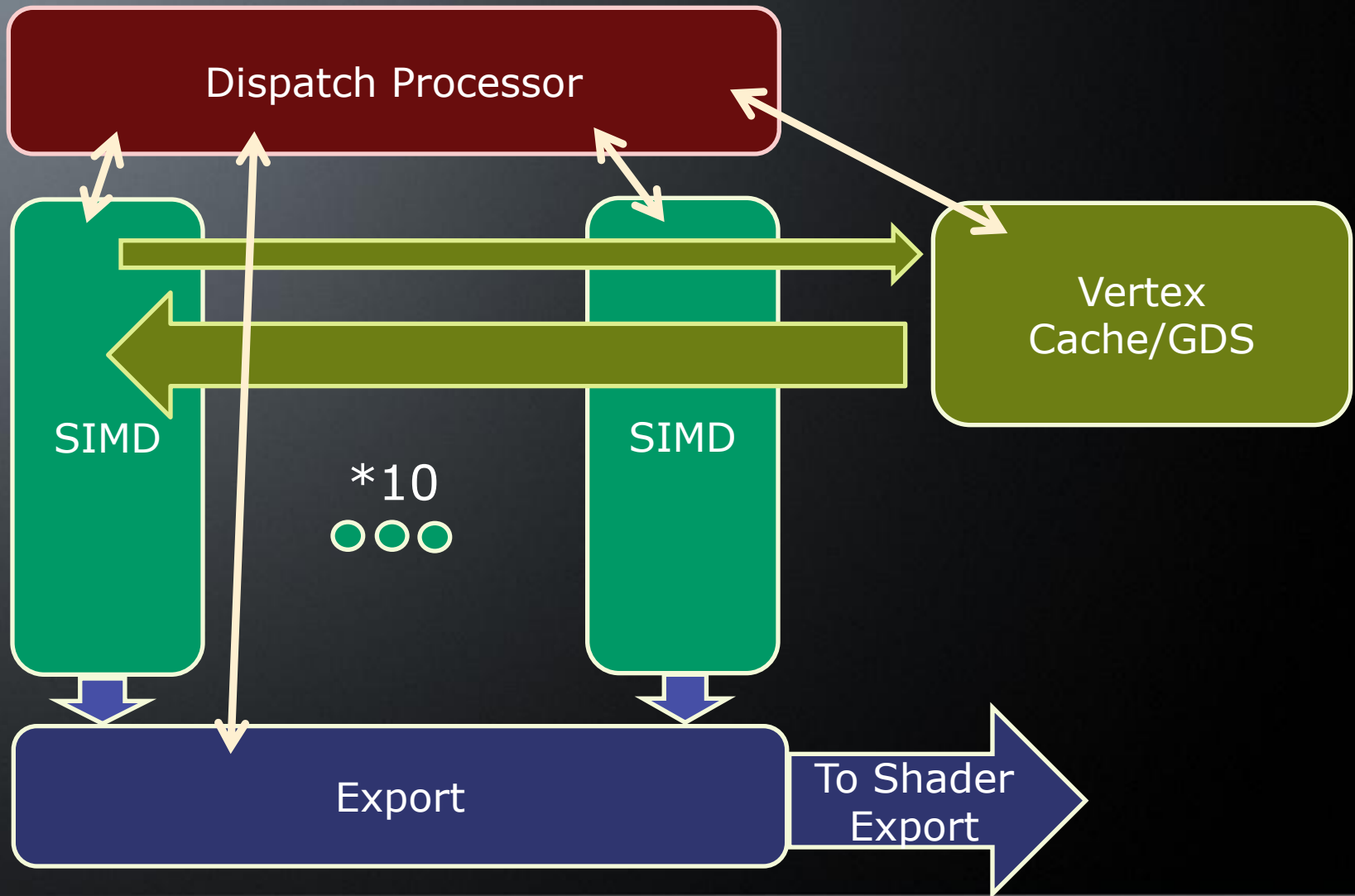
Pixel path



Compute path

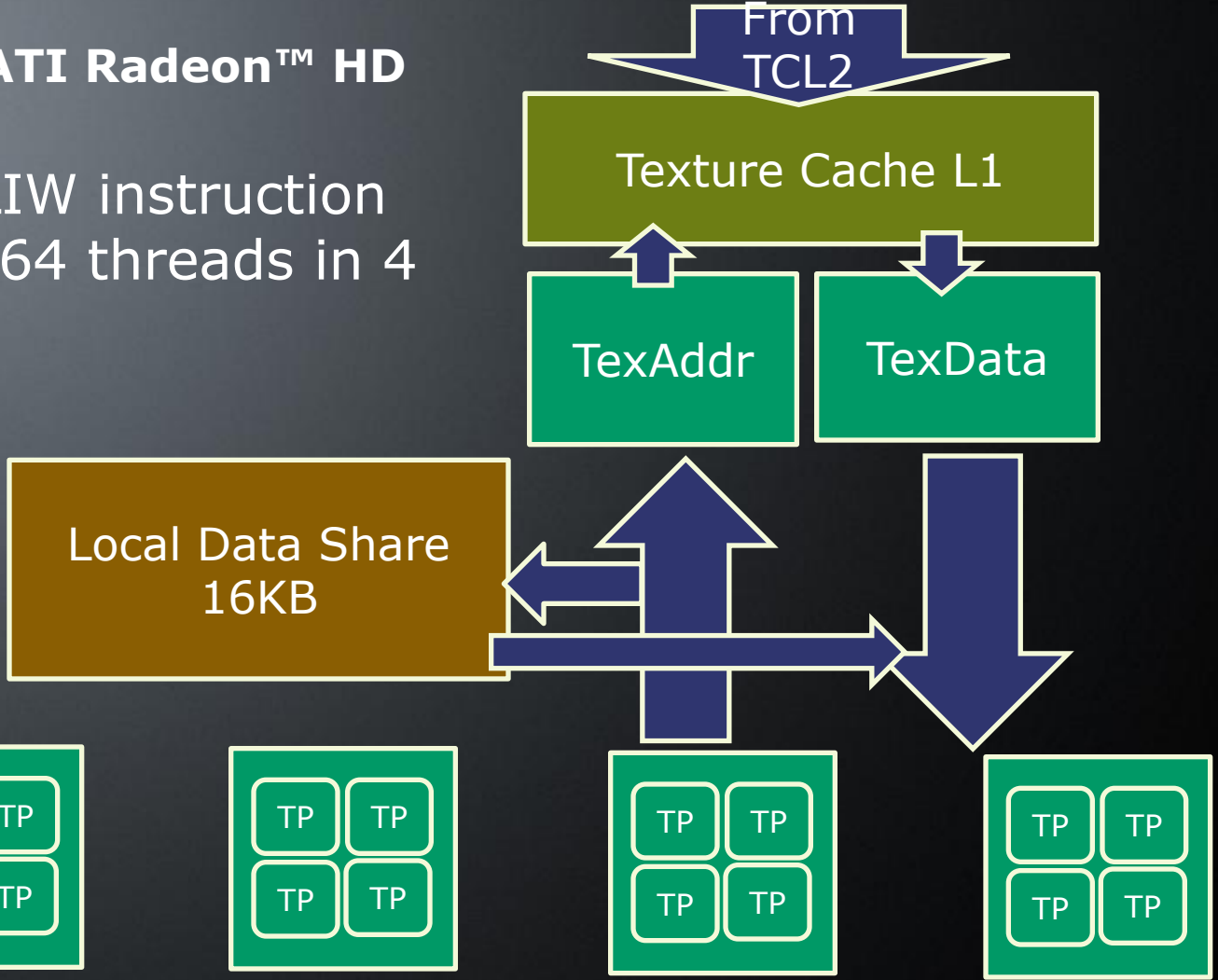


Shader Core (ATI Radeon™ HD 4870)

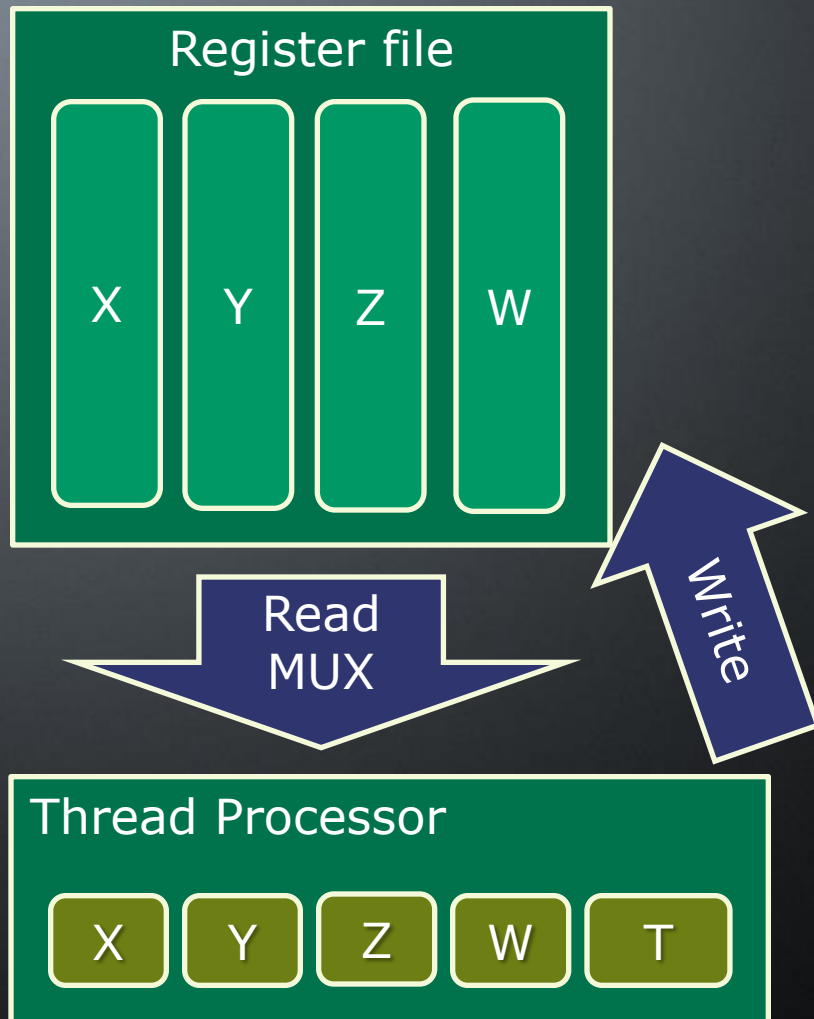


SIMD (ATI Radeon™ HD 4870)

One VLIW instruction across 64 threads in 4 cycles



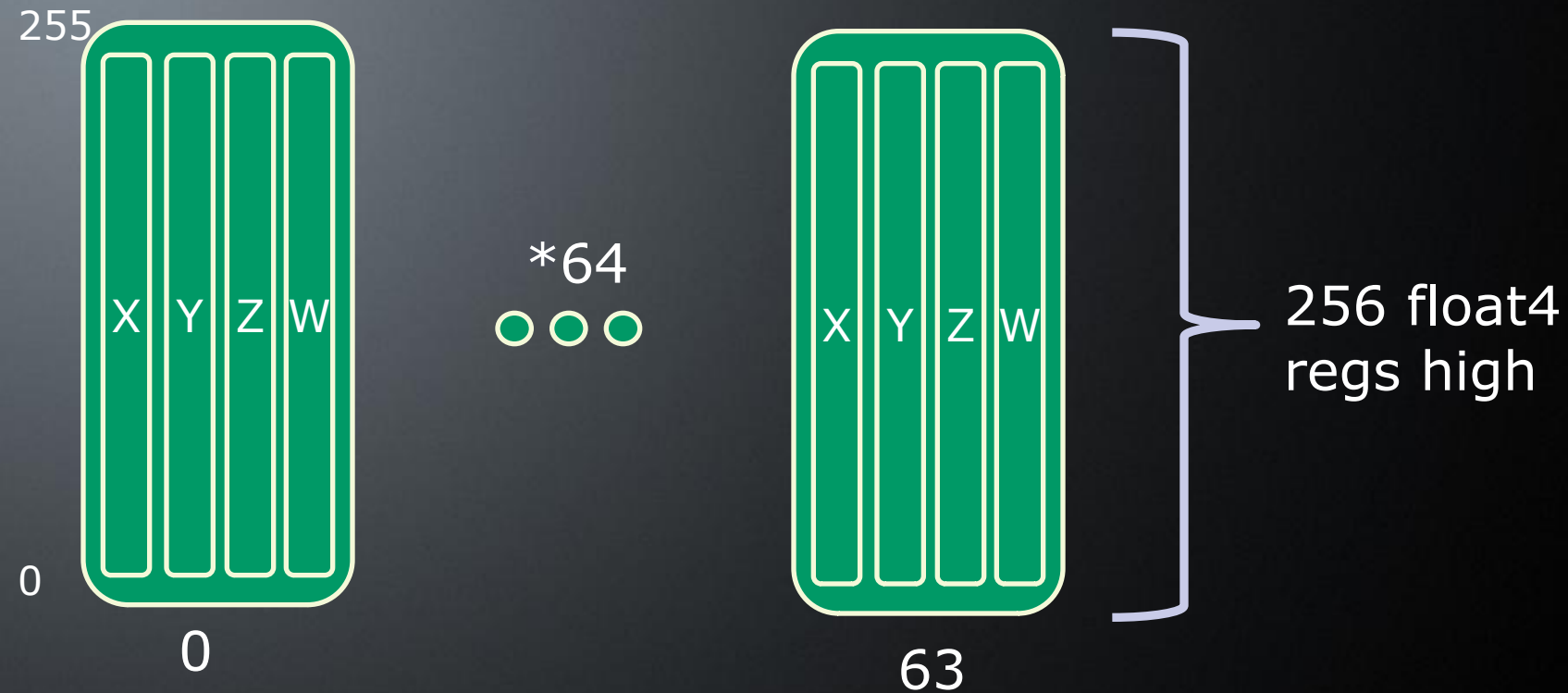
Thread Processor



Cycle	Read	Write
0	SrcA	Dst Vec
1	SrcB	Dst Scalar
2	SrcC/Exp	Interp
3	TexAddr/Exp	TexData



Register file (one SIMD)



float4 * 64 vectors * 256 registers *
10 SIMD = **2.5MB** of **registers** on
ATI Radeon™ HD 4870



Register file (one SIMD)

0 Pixel shader mode in CAL 255



0 OpenCL™ and Compute mode in CAL 255



Instruction Set

3 instruction classes:

- Control flow
- ALU
- Texture/Vertex fetch

http://developer.amd.com/gpu_assets/R600_Instruction_Set_Architecture.pdf

http://developer.amd.com/gpu_assets/R700-Family_Instruction_Set_Architecture.pdf

http://developer.amd.com/gpu_assets/Intermediate_Language_Specification--Stream_Processor.pdf



Instruction Set: Control Flow

Instruction classes

- Start ALU clause and lock constant cache lines
- Start Fetch clause (Texture, Vertex, Vertex_TC, global mem read)
- Control flow (PUSH,POP, ELSE, JUMP, LOOP, BREAK, CONTINUE)
- Export (position, vertex attributes, pixels, z, scratch memory, compute shader global mem export)



Instruction Set: ALU

One 5-way (XYZWT) VLIW instruction (bundle) per clock

Instruction classes:

- Floating point (ADD, MUL, MULADD etc)
- Double FP (XY,ZW slots) (mul_64, muladd_64)
- Integer (add_int, lshr_int, or_int etc)
- Transcendental (only in T slot) (sin, recip, sqrt, flt_to_int)
- Predicates, kill



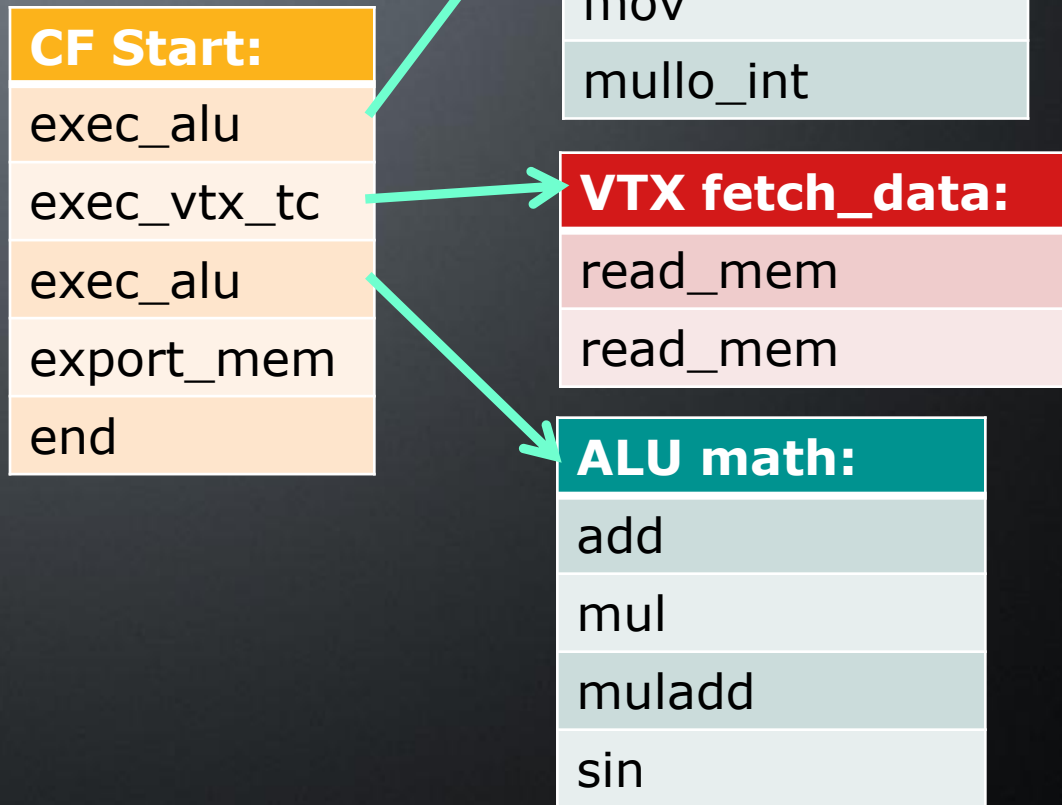
Instruction Set: Fetch

Instruction classes:

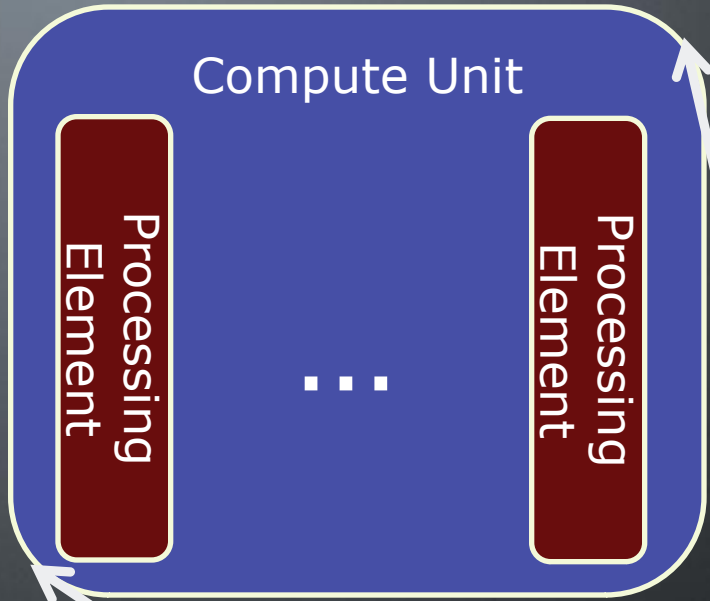
- sample (texture load using fp coords and filtering)
- ld (texture load using int coords)
- get/set gradients
- vfetch/semfetch (vertex fetches)
- read mem, scratch, reduction buffer (uncached)
- read/write global and local data share



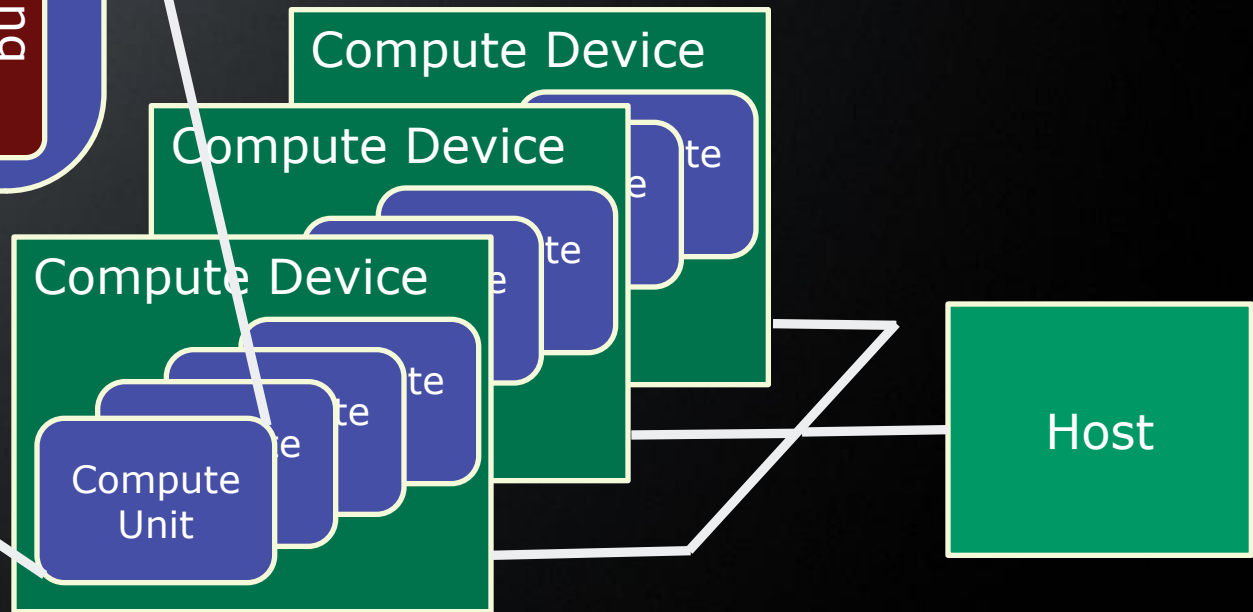
Instruction Set: Simple Compute Shader



OpenCL™ view



OpenCL (abstraction)	GPU (reality)
compute device	GPU (ATI Radeon™ HD 4870)
compute unit	SIMD
processing element	Thread Processor



How OpenCL™ maps onto GPU

OpenCL (abstract)	ATI Radeon™ HD 4870 (reality)
work-item	CS instance, pixel, vertex
work-group	group of wavefronts (waves)
local memory	Local Data Share *
private memory	scratch memory (x[] in CAL/IL)
global memory	global buffer (g[] in CAL/IL)
scalar variable	single or double component of a 128-bit register
vector variable	one or more 128-bit registers
image	texture ** (not yet)

*Local Data Share supports only owner writes in ATI Radeon™ HD 4870. **Image capabilities not currently implemented in ATI Stream 2.0 beta.



How OpenCL™ maps on CPU

OpenCL (abstract)	CPU (reality)
compute device	PC system
compute unit	processor core
processing element	processor core
work-item	software thread
work-group	hardware thread
local memory	CPU data L1 (per work-group allocated RAM)
global memory	system RAM
private memory	per work-item allocated system RAM
scalar variable	CPU register or stack space
vector variable	SSE register or stack space
image	(not yet) array in system RAM



Optimizations: Vectors

- We prefer float4 and int4 vectors. They map very well to 5-way VLIW and 128-bit registers on GPU and to SSE on CPUs.
- SSE 128-bit loads/stores.
- GPU 128-bit loads/stores.



Optimizations: Control Flow

- Divergent control flow within a wavefront is bad.
- GPU uses predication or thread masking, parts of the machine become unused.
- Kernel with many control flow statements with little math or fetches can become control flow bound.



Optimizations: Memory Access

Coalescing!

- Use 128-bit accesses. They are the most efficient and fully utilize the buses.
- Reads: Make wavefront read whole rows from memory. The HW will fetch whole cache lines so make other threads in wave reuse the data.
- Writes: Write whole rows. Be careful not to overload single backend or memory channel. Writes flow from Shader Core to Shader Export (coalescing) -> 4 Rendering Backends -> 8 memory channels (two per backend, bits 8-10 address the channel)



Optimizations: Registers

Stack variables and private variables (at discretion of the compiler) map to GPU registers.

Number of registers used affects number of wavefronts that can fit into SIMD.

A work-group must fit on a SIMD.

More waves help cover memory access latencies.

Big ALU clauses (more math in the kernel) also help cover memory latencies.

You need minimum of 3 waves (two in ALU, one in fetch), preferably 5 or more.

Sometimes better to recompute instead of fetch from memory.



Optimizations: Balance

Fetch:ALU:Export ratio

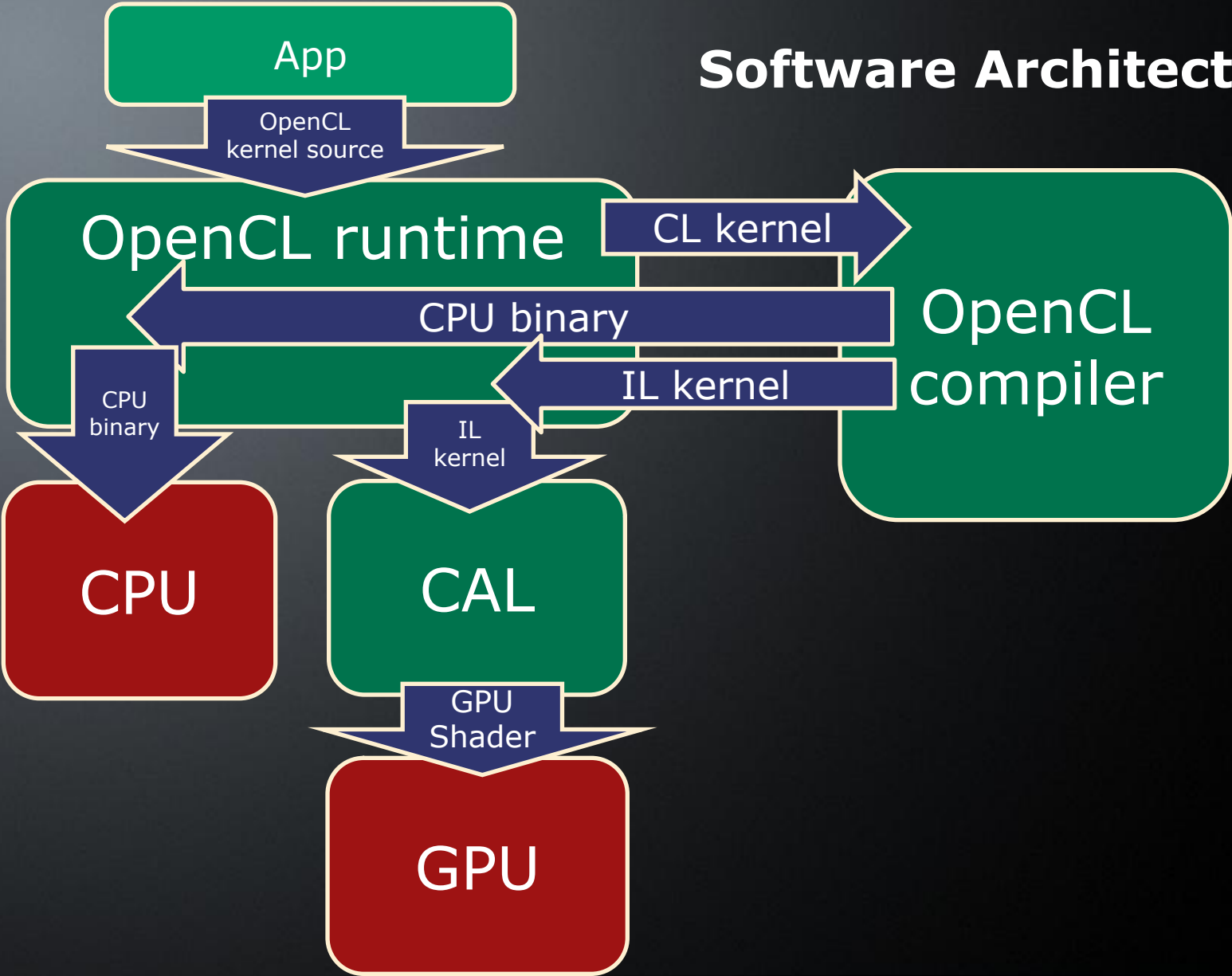
- 40 128-bit fetches
- 160 5-way VLIW ALU instructions
- 16 128-bit exports

Registers:Waves:Latency compensation

- Use registers to store fetched or intermediate data, there are a lot of them
- But make sure there is enough waves to hide latencies
- And/or enough math



Software Architecture



Optimizations: API

- Building programs and creating kernels is expensive. For interactive applications do it once and early while loading the app.
- Running with different work-group sizes may also trigger less expensive but significant compile.
- Moving data from host to (GPU) device and back can be also very expensive. Keep it on the device, chain kernels, read back only what you really need on host. Use Direct3D® and OpenGL bindings to share the data with rendering APIs.



Stream Kernel Analyzer

<http://developer.amd.com/gpu/ska/Pages/default.aspx>



Trademark Attribution

AMD, the AMD Arrow logo, ATI, the ATI logo, Radeon, and combinations thereof are trademarks of Advanced Micro Devices, Inc. OpenCL is trademark of Apple Inc. used under license to the Khronos Group Inc. Microsoft and Direct3D are registered trademarks of Microsoft Corporation in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.

©2009 Advanced Micro Devices, Inc. All rights reserved.

