CUDA Programming

Week 6. Fermi and OpenCL

Outline

- Fermi
- ATI dragon
- OpenCL
- Homework/Project

FERMI

History perspective

- G80 unifies graphics and computing parallel processor
- GT200 extends the performance and functionality of G80.
- Fermi is based on the feedbacks from all the applications that were written on G80 and GT200.

Comparison

GPU	G80	GT200	Fermi
Transistors	681 million	1.4 billion	3.0 billion
CUDA Cores	128	240	512
Double Precision Floating Point Capability	None	30 FMA ops / clock	256 FMA ops /clock
Single Precision Floating Point Capability	128 MAD ops/clock	240 MAD ops / clock	512 FMA ops /clock
Special Function Units (SFUs) / SM	2	2	4
Warp schedulers (per SM)	1	1	2
Shared Memory (per SM)	16 KB	16 KB	Configurable 48 KB or 16 KB
L1 Cache (per SM)	None	None	Configurable 16 KB or 48 KB
L2 Cache	None	None	768 KB
ECC Memory Support	No	No	Yes
Concurrent Kernels	No	No	Up to 16
Load/Store Address Width	32-bit	32-bit	64-bit

Important features

- Improve double precision performance
- ECC support
- True Cache Hierarchy
- More Shared Memory
- Faster Context Switching
- Faster Atomic Operations

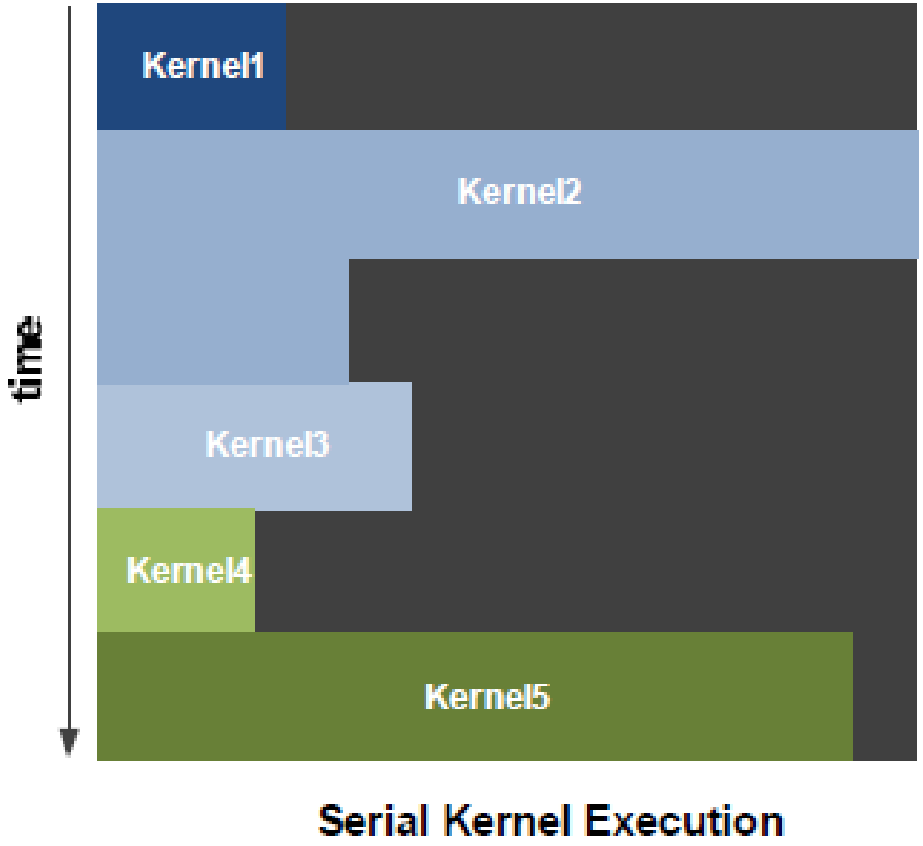
Streaming multiprocessors

- 32 CUDA cores per SM, 4x over GT200
- 8x the peak double precision floating point performance over GT200
- Dual warp scheduler simultaneously schedules and dispatches instructions from two independent warps
- 64 KB of RAM with a configurable partitioning of shared memory and L1 cache

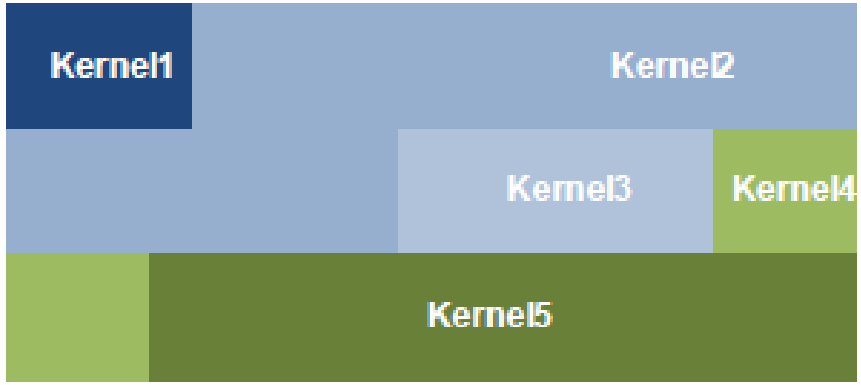
Thread execution

- Optimized for OpenCL and Direct Compute
- Full IEEE 754-2008 32-bit and 64-bit precision
- Full 32-bit integer path with 64-bit extensions
- Improved Performance through predication
- 10x faster application context switching
- Concurrent kernel execution
- Out of Order thread block execution
- Dual overlapped memory transfer engines

Concurrent Kernel execution



Serial Kernel Execution



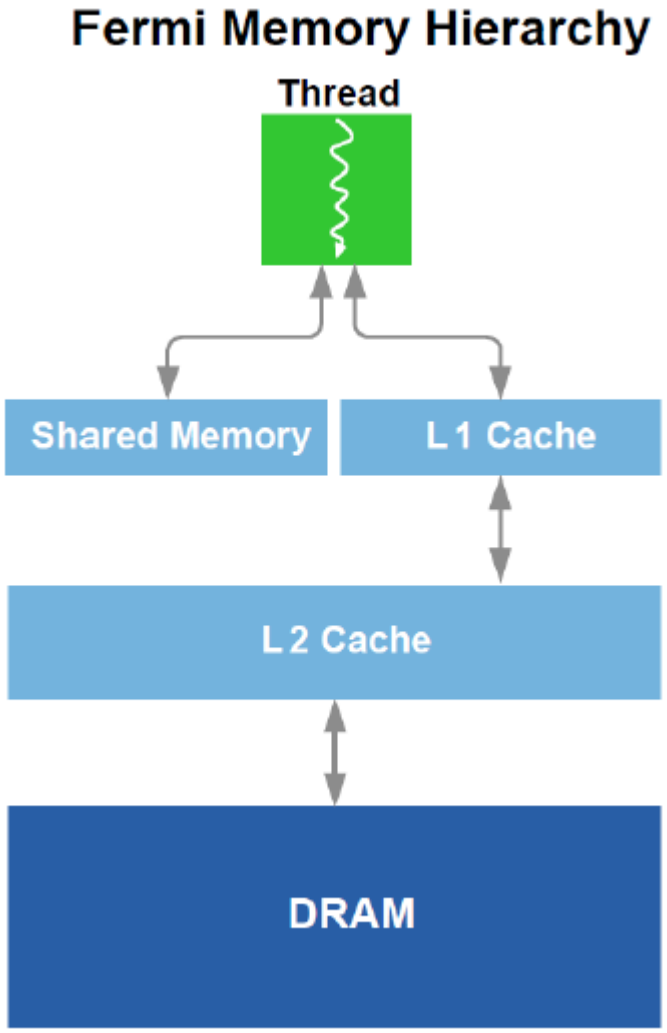
Concurrent Kernel Execution

Memory system

- Unified address space with full C++ support
- Memory access instructions to support transition to 64-bit addressing
- Configurable L1 and Unified L2 Caches
- First GPU with ECC memory support
- Greatly improved atomic memory operation performance

Configurable cache

- Shared memory is not a cache.
- 64K memory space is split for shared memory and L1 cache
  - Either 48K SM + 16K L1
  - Or 16K L1 + 48K SM



Nexus: beside the hardware

- The first development environment designed specifically to support massively parallel CUDA C, OpenCL, and DirectCompute applications.
  - Parallel-aware hardware source code debugging
  - Performance analysis
- Integrated in Visual Studio

Adopted from “An Introduction to OpenCL” by John Stone

OPENCL

What is OpenCL?

- Open Computational Language
- Cross-platform parallel computing API
- C-like language for heterogeneous devices
- Code is portable across devices:
  - Correctness is guaranteed
  - Performance is **not guaranteed**
- OpenCL supports for AMD and NVIDIA GPUs, x86 CPUs, IBM Cell and various hardware

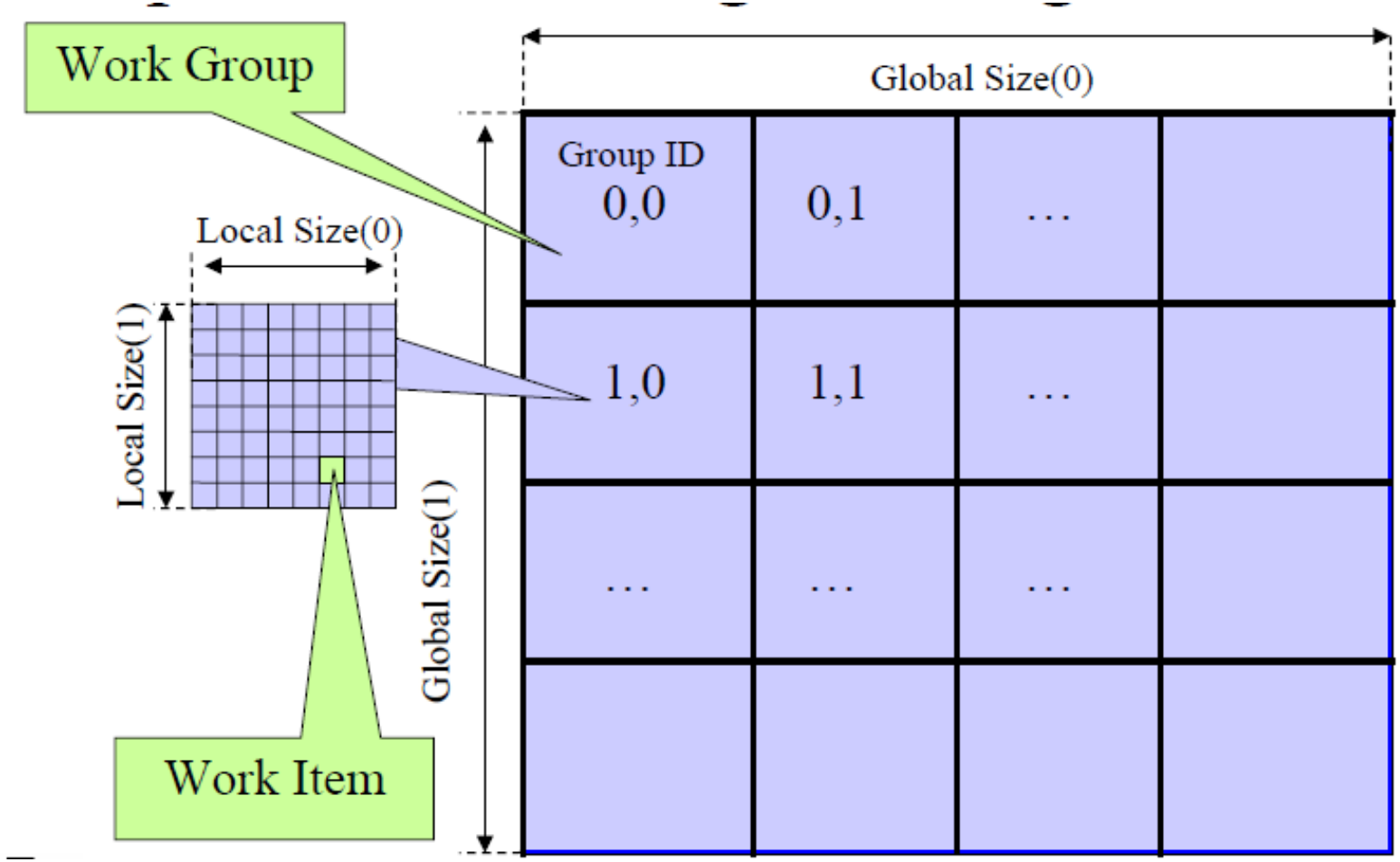
Weaknesses of OpenCL 1.0

- OpenCL is a low-level API
  - Developers are responsible for a lot of plumbing, lots of objects/handles to keep track of
- Developers are responsible for thread-safety
  - Some types of multi-accelerator codes are much more difficult to write than in the CUDA
- Great need for OpenCL middleware and libraries

Data Parallel Model

- Work is submitted to devices by kernels
- Kernels run over global dimension index ranges (NDRange), broken up into “work groups” (block), and “work items” (thread?)
- Work items executing within the same work group can synchronize with each other with barriers or memory fences
- Work items in different work groups can’t sync with each other, except by launching a new kernel

OpenCL NDRange



Hardware abstraction

- OpenCL exposes CPUs, GPUs, and other Accelerators as “devices”
- Each “device” contains one or more “compute units”, i.e. cores, SMs, etc...
- Each “compute unit” contains one or more SIMD “processing elements”

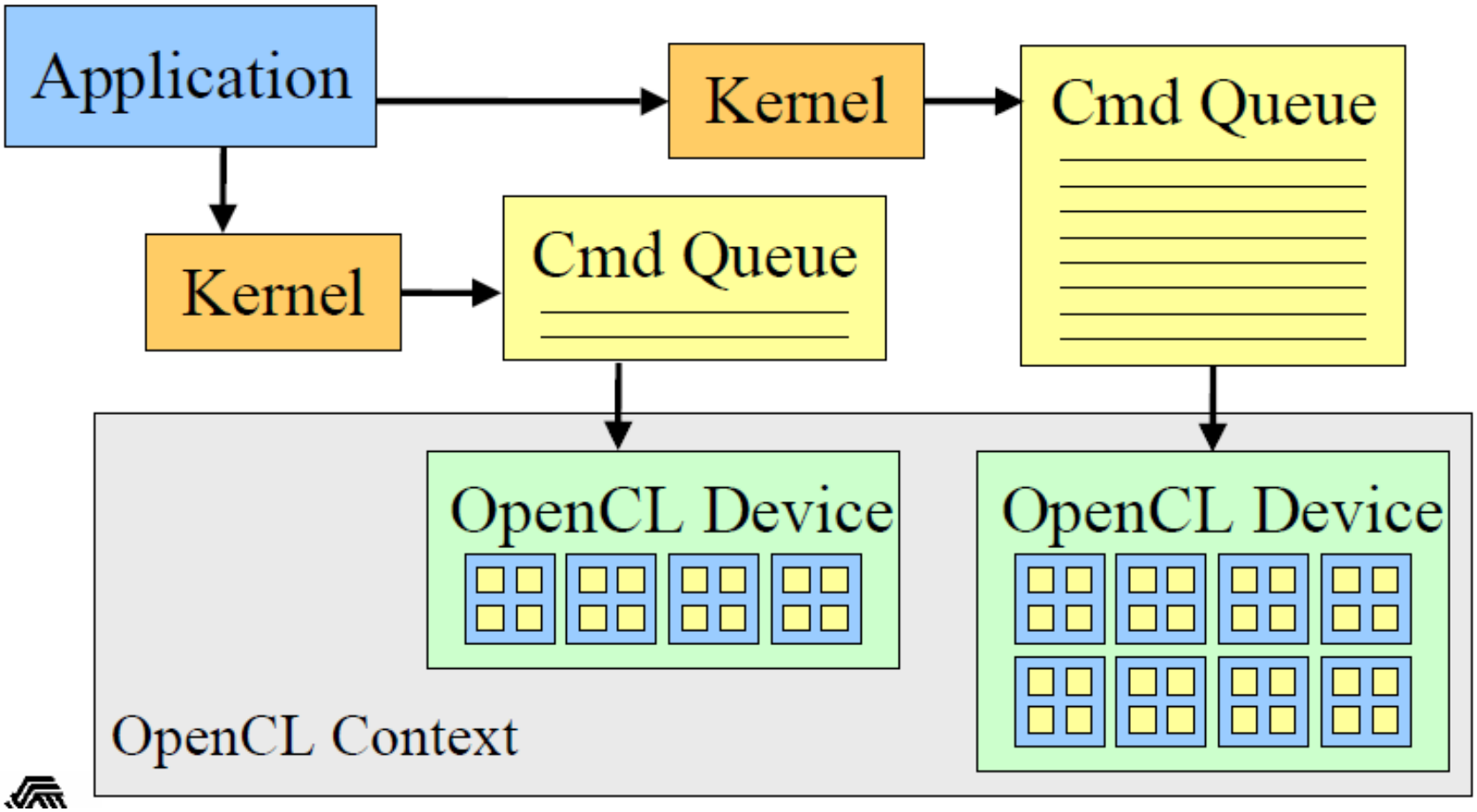
Memory system

- __global–large, high latency
- __private–on-chip device registers
- __local–memory accessible from multiple PEs or work items, SRAM or DRAM, must query…
- __constant–read-only constant cache
- Device memory is managed explicitly by the programmer, as with CUDA
- Pinned memory buffer allocations are created using the CL_MEM_USE_HOST_PTR flag

OpenCL Context

- Contains one or more devices
- OpenCL memory objects are associated with a context, not a specific device
- clCreateBuffer() emits error if an allocation is too large for any device in the context
- Each device needs its own work queue(s)
- Memory transfers are associated with a command queue (thus a specific device)

Multiple devices



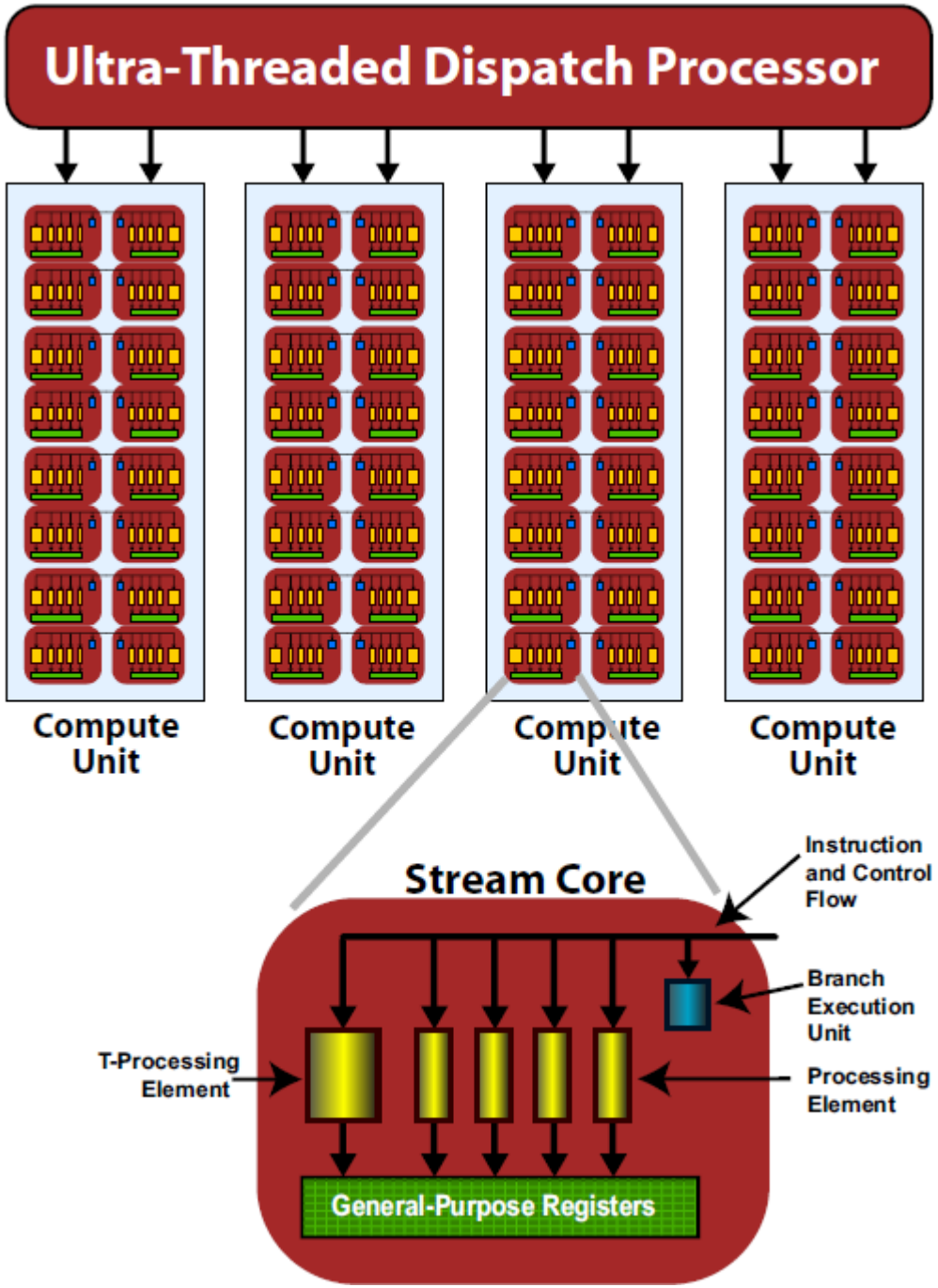
CUDA and OpenCL teminology

CUDA terminology	OpenCL terminology
Thread	Work-item
Thread block	Work-group
Global memory	Global memory
Constant memory	Constant memory
Shared memory	Local memory
Local memory	Private memory

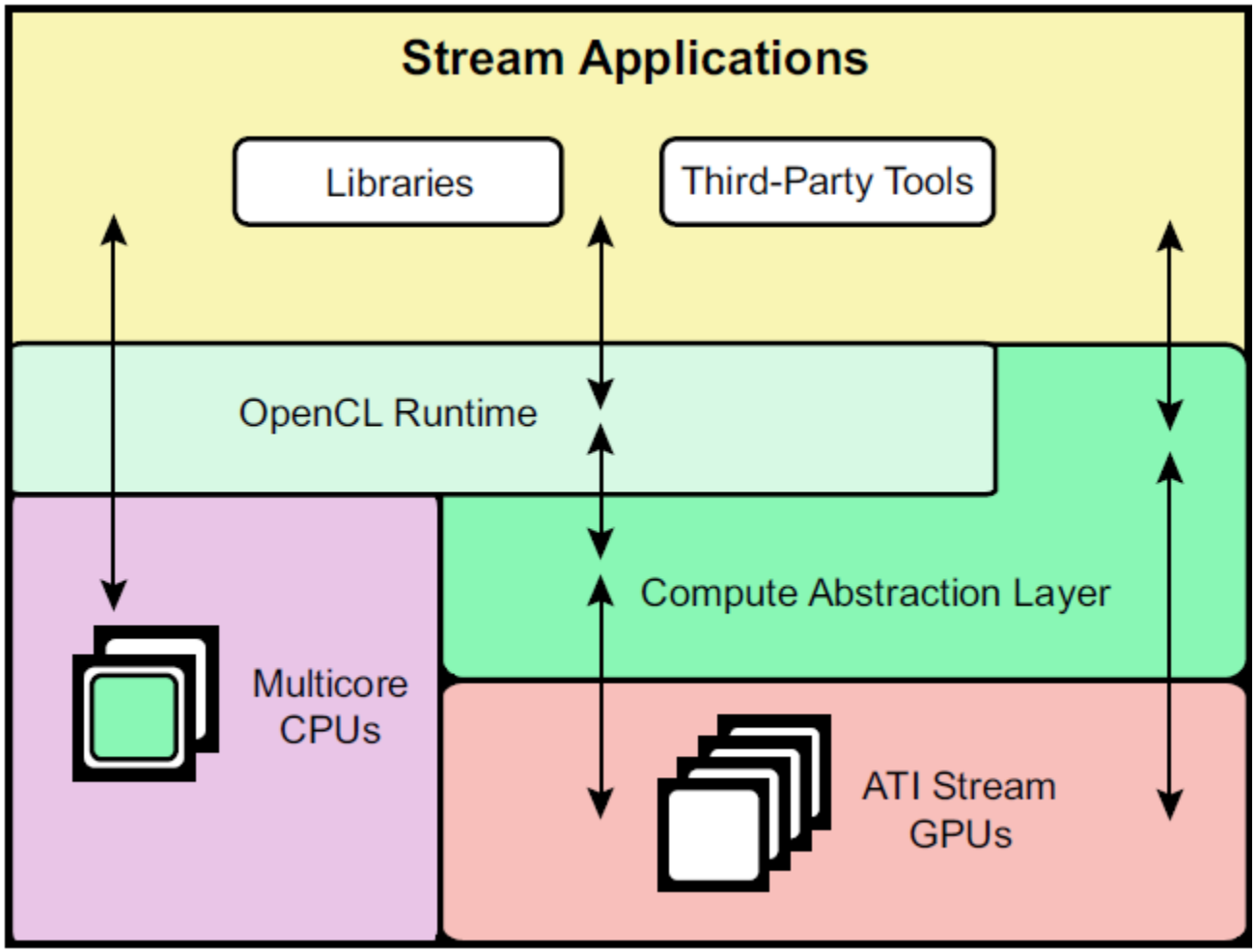
ATI RADEON

Hardware Overview

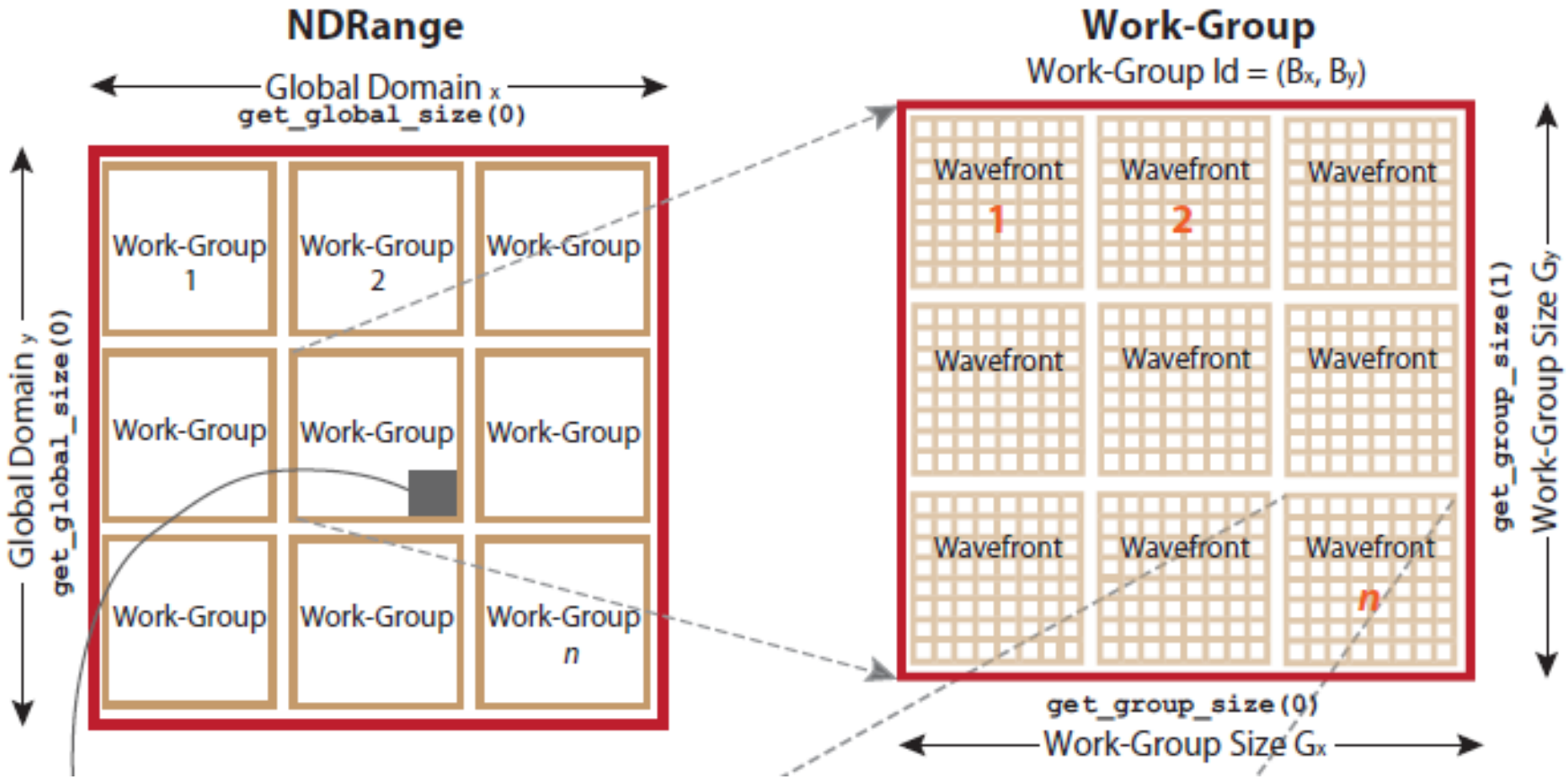
- A stream core is arranged as a five-way very long instruction word (VLIW) processor.
  - Up to five scalar operations can be issued in a VLIW instruction.



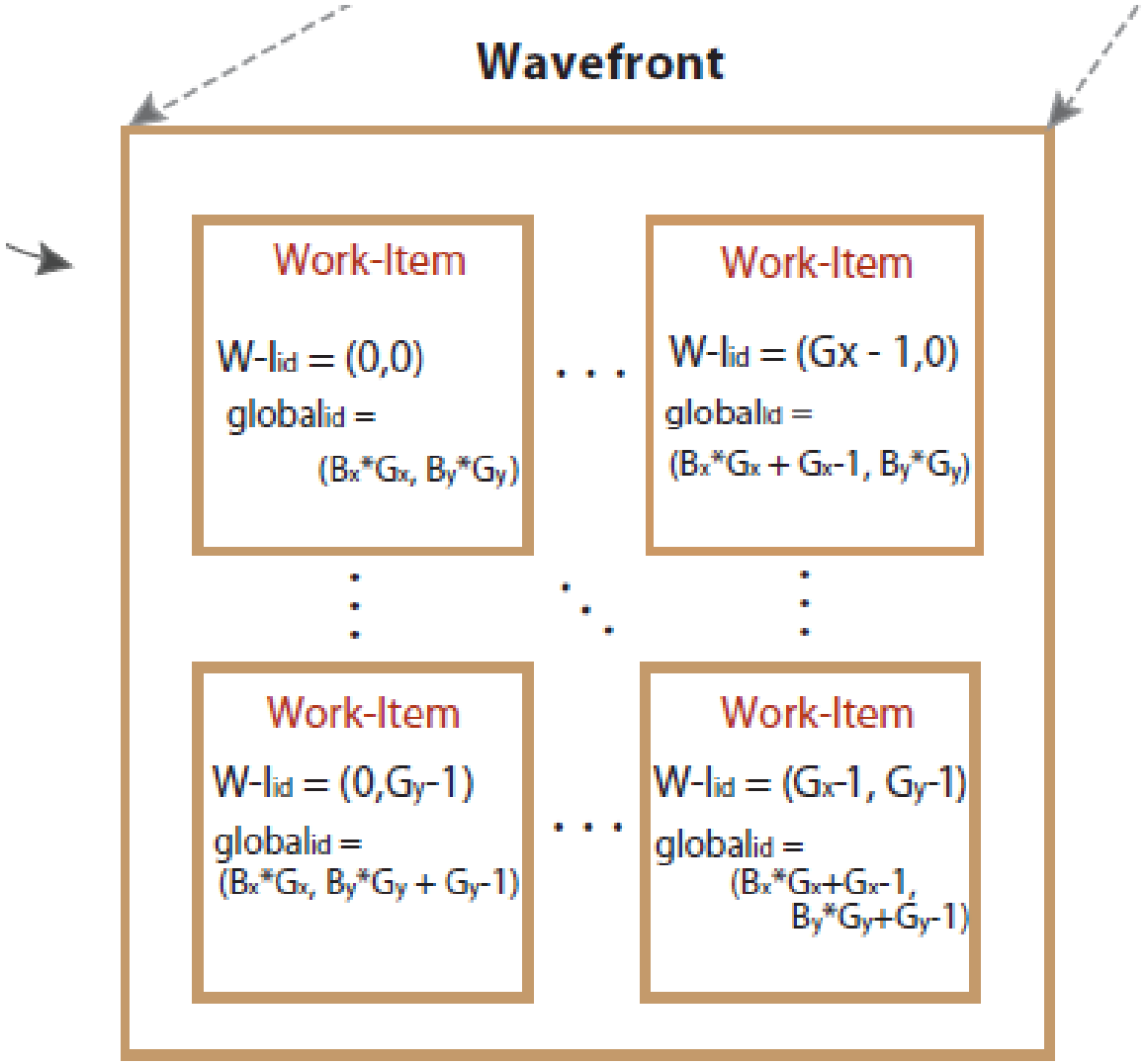
ATI Stream Computing of OpenCL



Work group

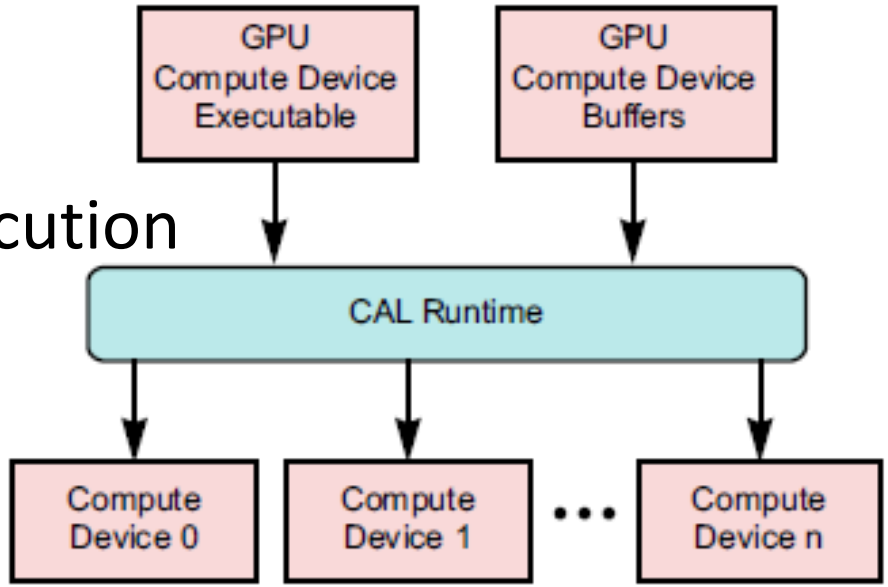

NDRange
Global Domain x
get_global_size(0)
Global Domain y
get_global_size(0)
Work-Group 1
Work-Group 2
Work-Group
Work-Group
Work-Group
Work-Group
Work-Group
Work-Group
Work-Group n
Work-Group
Work-Group Id = (Bx, By)
Wavefront 1
Wavefront 2
Wavefront
Wavefront
Wavefront
Wavefront
Wavefront
Wavefront
Wavefront n
get_group_size(1)
Work-Group Size Gy
get_group_size(0)
Work-Group Size Gx

Wavefront and Work Item

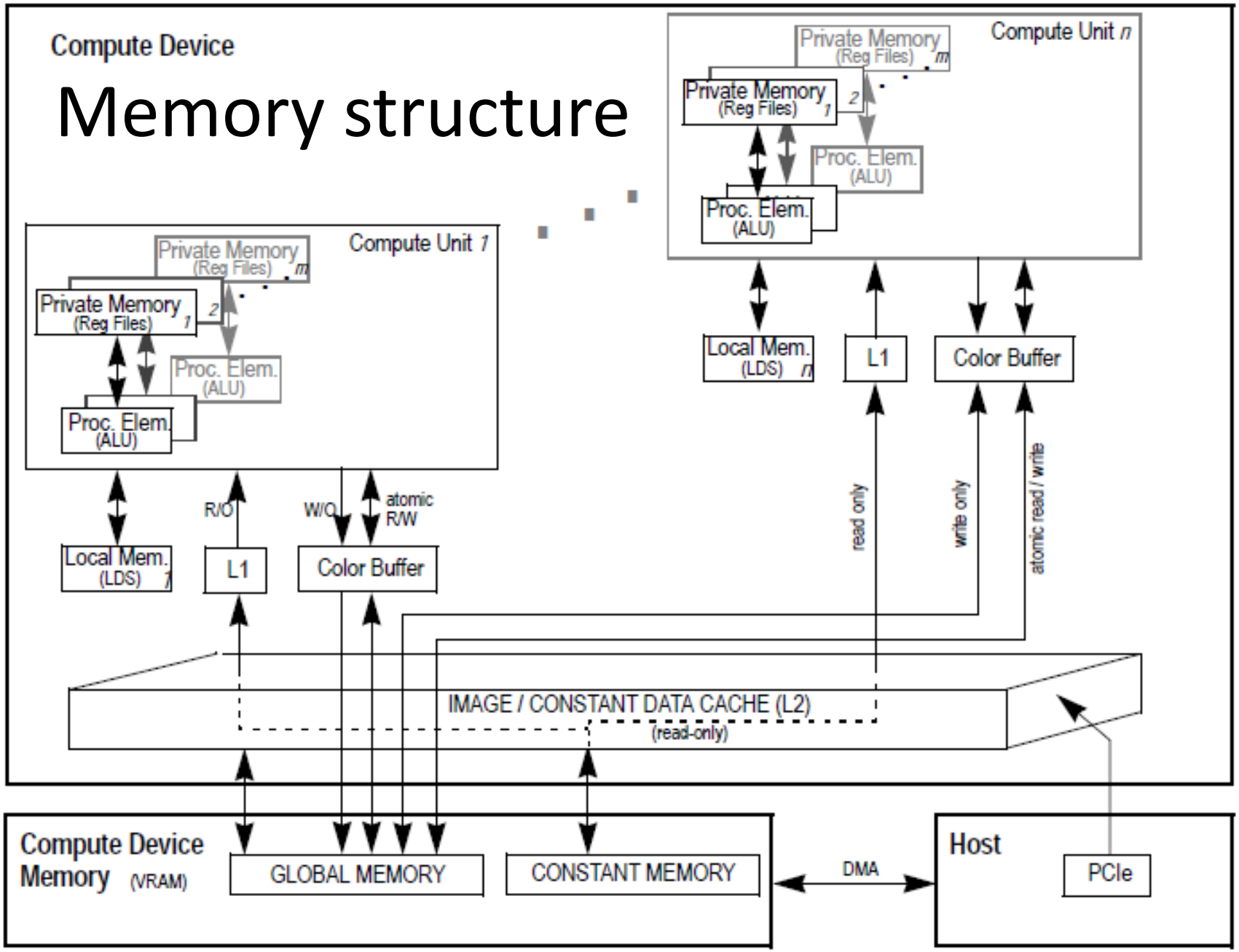


ATI Compute Abstraction Layer (CAL)

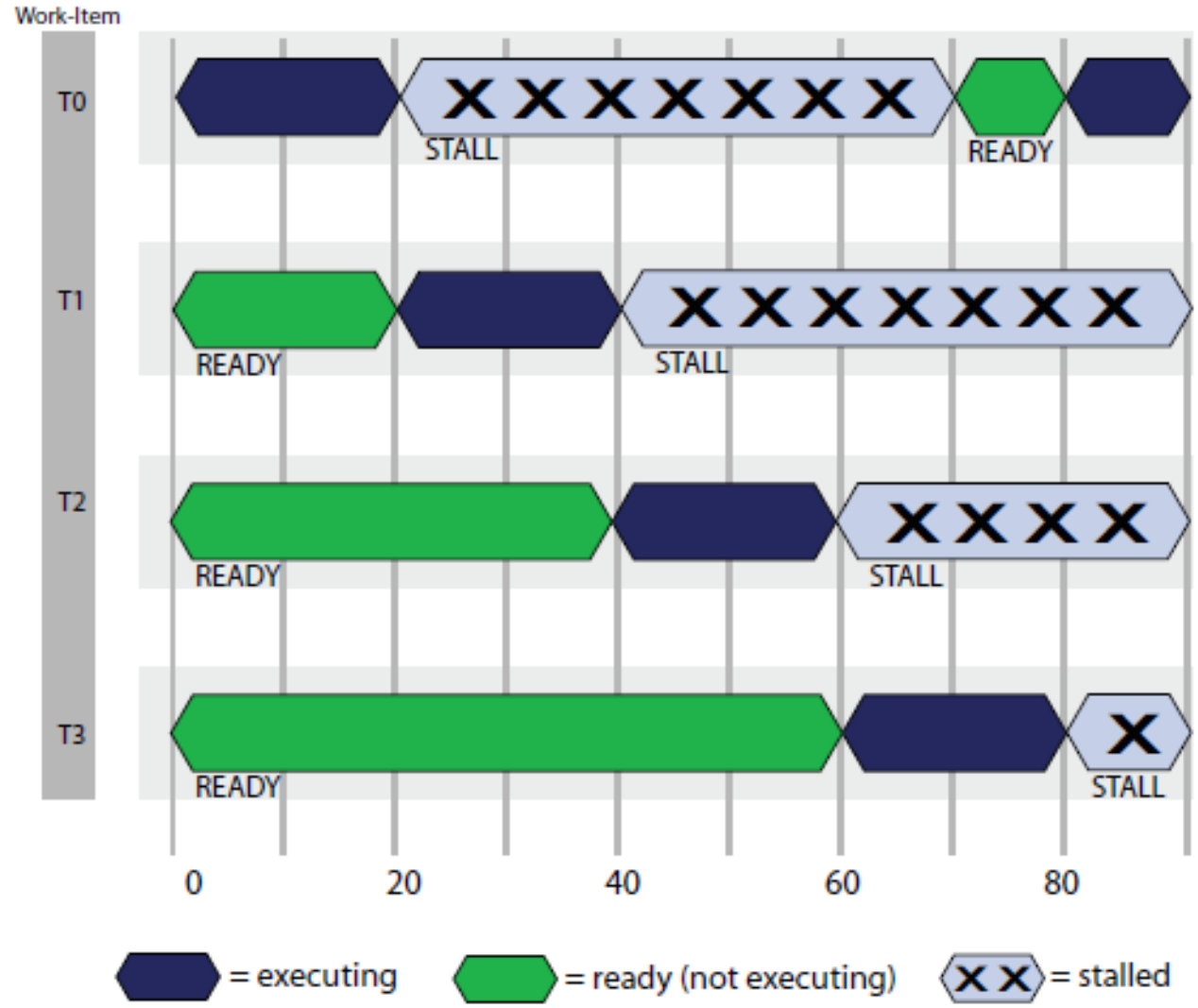
- A device driver library that provides a forward-compatible interface to ATI GPU
  - Device-specific code generation
  - Interoperability with 3D graphics APIs
  - Resource management
  - Device management
  - Kernel loading and execution
  - Multi-device support



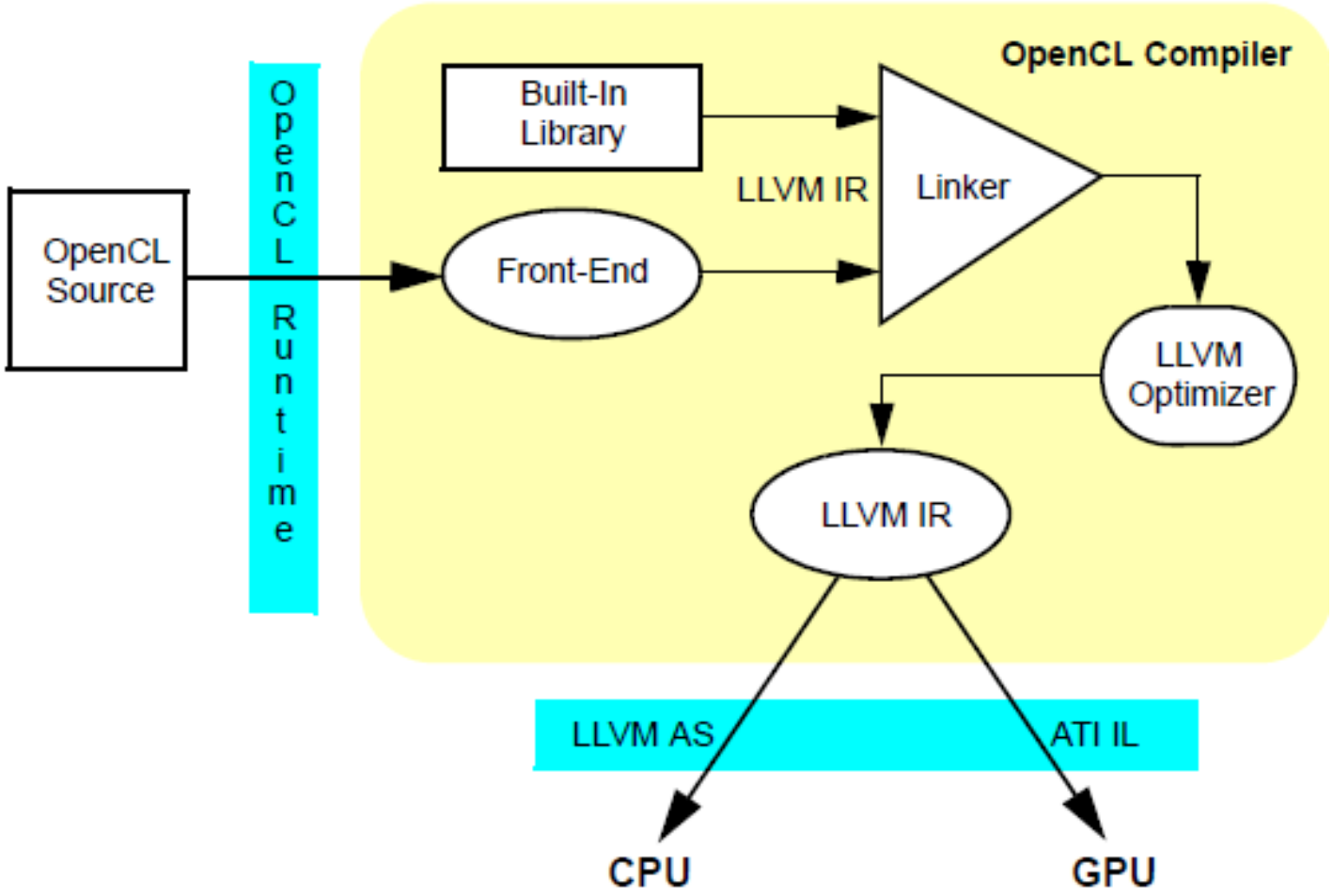
Memory structure



Device Scheduling



OpenCL compilation flow



Useful links

- OpenCL Tutorial: http://developer.amd.com/gpu/ATIStreamSDK/pages/TutorialOpenCL.aspx
- ATI Stream Beta 2.0: http://developer.amd.com/streambeta
- OpenCL Specification: http://www.khronos.org/registry/cl/

HOMEWORK/PROJECT

Homework

- Try OpenCL on nVidea GTX and ATI Radeon
  - I purchased a new machine equipped with ATI Radeon
- OpenCL Implementations, Tutorials and Sample codes
  - http://www.khronos.org/developers/resources/opencl/#texamples

Project

- Read and implement some papers about GPU
  - Designing efficient sorting algorithms for GPUs.
    - http://mgarland.org/files/papers/gpusort-ipdps09.pdf
  - Warp sort
    - http://www.ipdps.org/ipdps2010/ipdps2010-slides/session-05/Ye_ipdps10_slide.pdf
  - Sample sorting
    - http://arxiv.org/abs/0909.5649
  - Sparse matrix-vector multiplication
    - http://graphics.cs.uiuc.edu/~wnbell/publications/2009-08-SC-SpMV/sc09-spmv-throughput.pdf