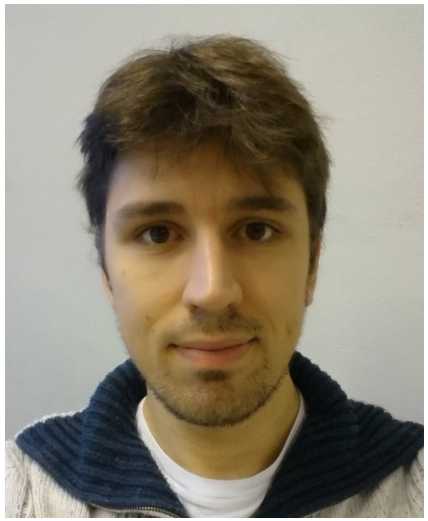
HETEROGENEOUS **CPU+GPU** COMPUTING

Ana Lucia Varbanescu – University of Amsterdam
a.l.varbanescu@uva.nl
Stijn Heldens – Twente University
mail@stijnh.nl

Significant contributions by: **Pieter Hijma** (UvA, NL), **Jie Shen** (TUDelft, NL), **Basilio Fraguela** (A Coruna University, ESP)

Welcome to our tutorial!

Ana Lucia Varbanescu
Assistant Professor
Informatics Institute
University of Amsterdam
a.l.varbanescu@uva.nl



Stijn Heldens
PhD student (specialist
in heterogeneous computing)
University of Twente.
mail@stijnh.nl

Jie Shen
Assistant Professor
School of Computer
NUDT, China
j.shen@nudt.cn.edu

Today's agenda

- Preliminaries
- Part I: Introduction to heterogeneous computing
  - Performance promise vs. challenges
- Part II: Programing models
- Part III: Workload partitioning models
  - Static vs. Dynamic partitioning
- Part IV: Tools for (programming) heterogeneous systems
  - Low-level to high-level
- Part IV: Hands-on
  - Let's try things out!
- Take home message: dialogue

Goal

- Discuss heterogeneous computing as a promising solution for efficient resource utilization
  - And performance!
- Introduce methods for efficient heterogeneous computing
  - Programming
  - Partitioning
- Provide comparisons & selection criteria
- Provide practical examples
  - And an entertaining hands-on part!

- Current challenges and open research questions.

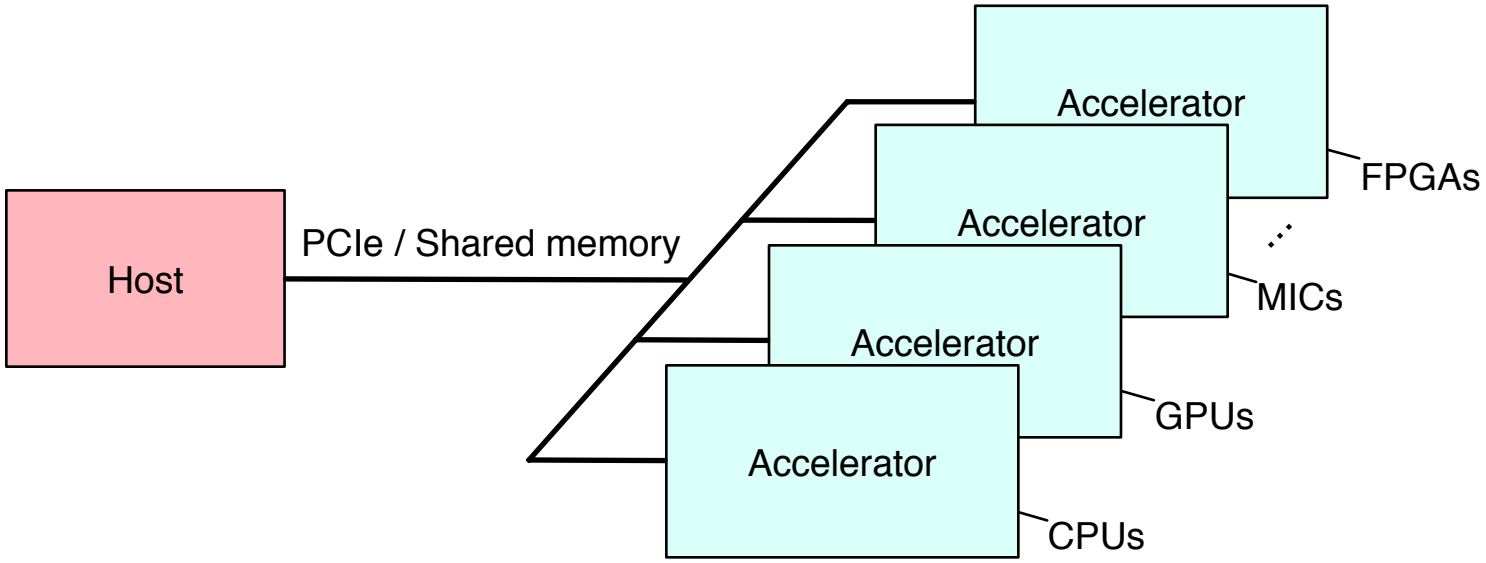
- Fair to others, but we advertise our research ☺

Heterogeneous platforms

- Systems combining main processors and accelerators
  - e.g., CPU + GPU, CPU + Intel MIC, AMD APU, ARM SoC
  - Everywhere from supercomputers to mobile devices

Heterogeneous platforms

- Host-accelerator hardware model



Heterogeneous platforms

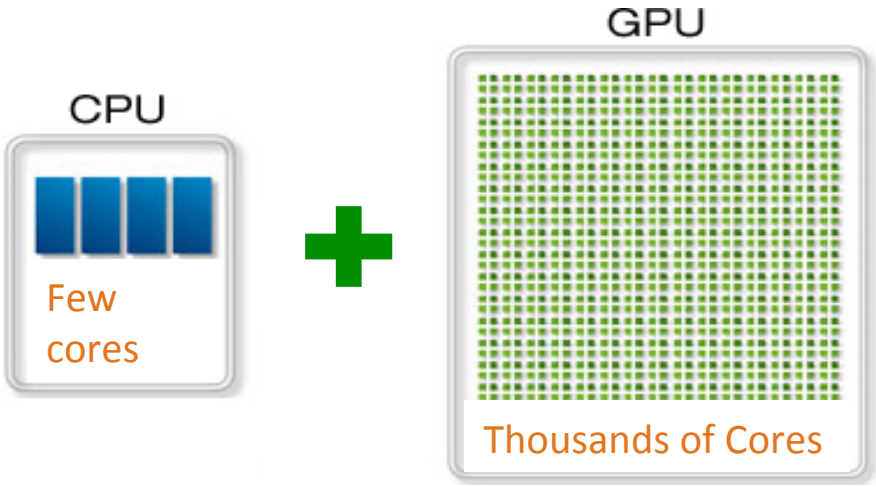
- Top 500 (June 2015)

RANK	SITE	SYSTEM	CORES	RMAX (TFLOP/S)	RPEAK (TFLOP/S)	POWER (KW)
1	National Super Computer Center in Guangzhou China 195 cores/node	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT Accelerated!	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc. Accelerated!	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	1,572,864	17,173.2	20,132.7	7,890
4	RI Co Ja					0
5	DOE/SC/Argonne National Laboratory United States	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945

All systems are based on multi-cores.
90 systems have accelerators (18%).
Of those, 50% are NVIDIA GPUs, 30% are Intel MICs (Xeon Phi).

Our focus today …

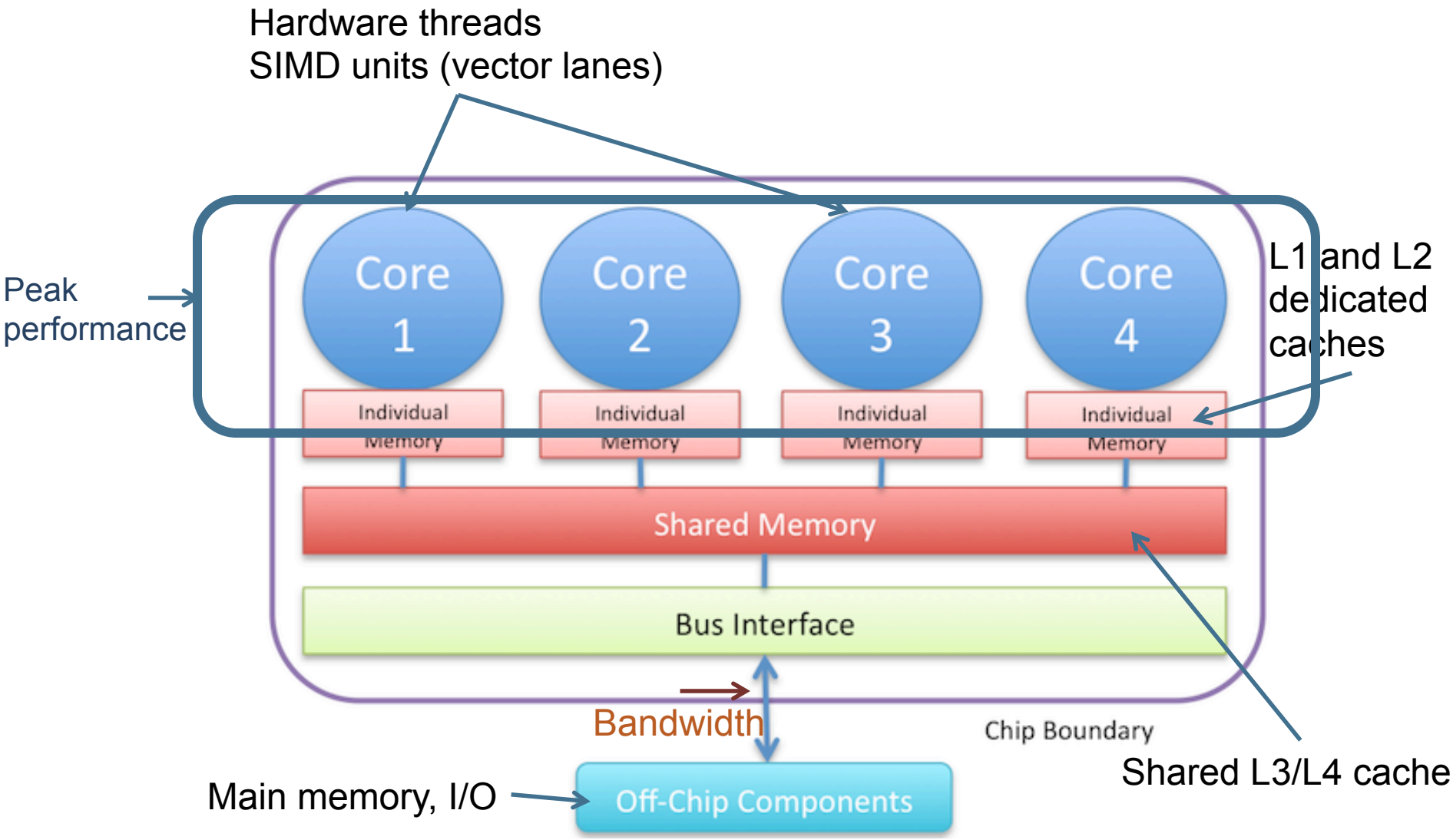
- A heterogeneous platform = **CPU** + **GPU**
  - Most solutions work for other/multiple accelerators
- An application workload = an application + its input dataset
- Workload partitioning = workload distribution among the processing units of a heterogeneous system



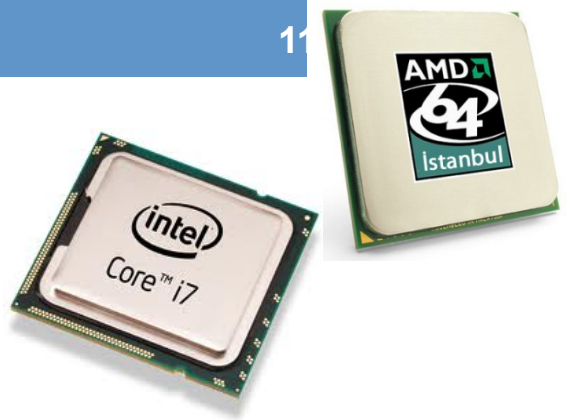
BEFORE WE START …

Basic knowledge about CPUs and GPUs

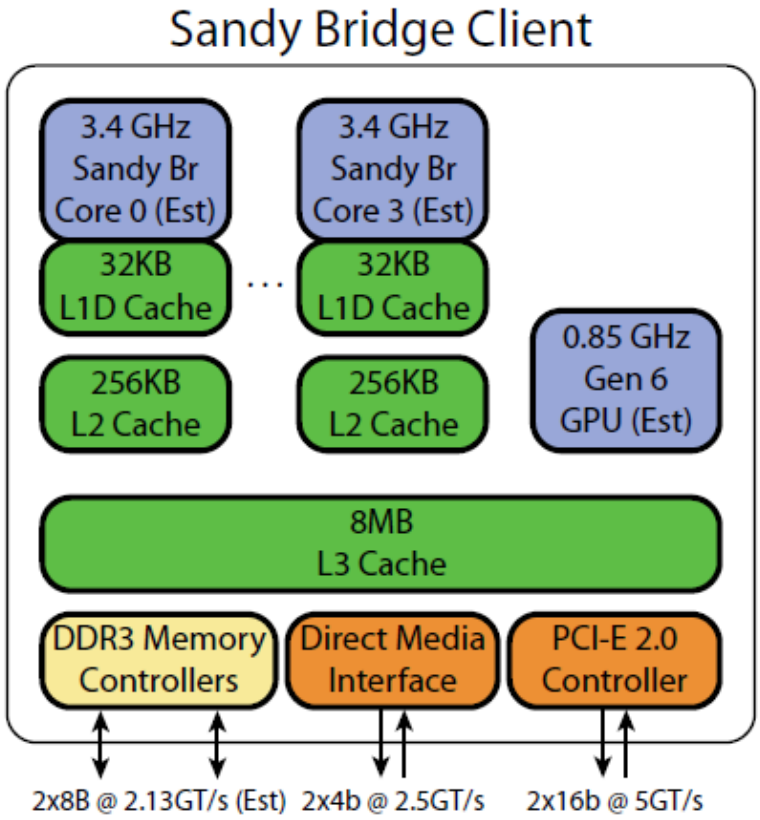
Generic multi-core CPU



Multi-core CPUs



- Architecture
  - Few large cores
  - (Integrated GPUs)
  - Vector units
    - Streaming SIMD Extensions (SSE)
    - Advanced Vector Extensions (AVX)
  - Stand-alone
- Memory
  - Shared, multi-layered
  - Per-core caches + shared caches
- Programming
  - Multi-threading
  - OS Scheduler



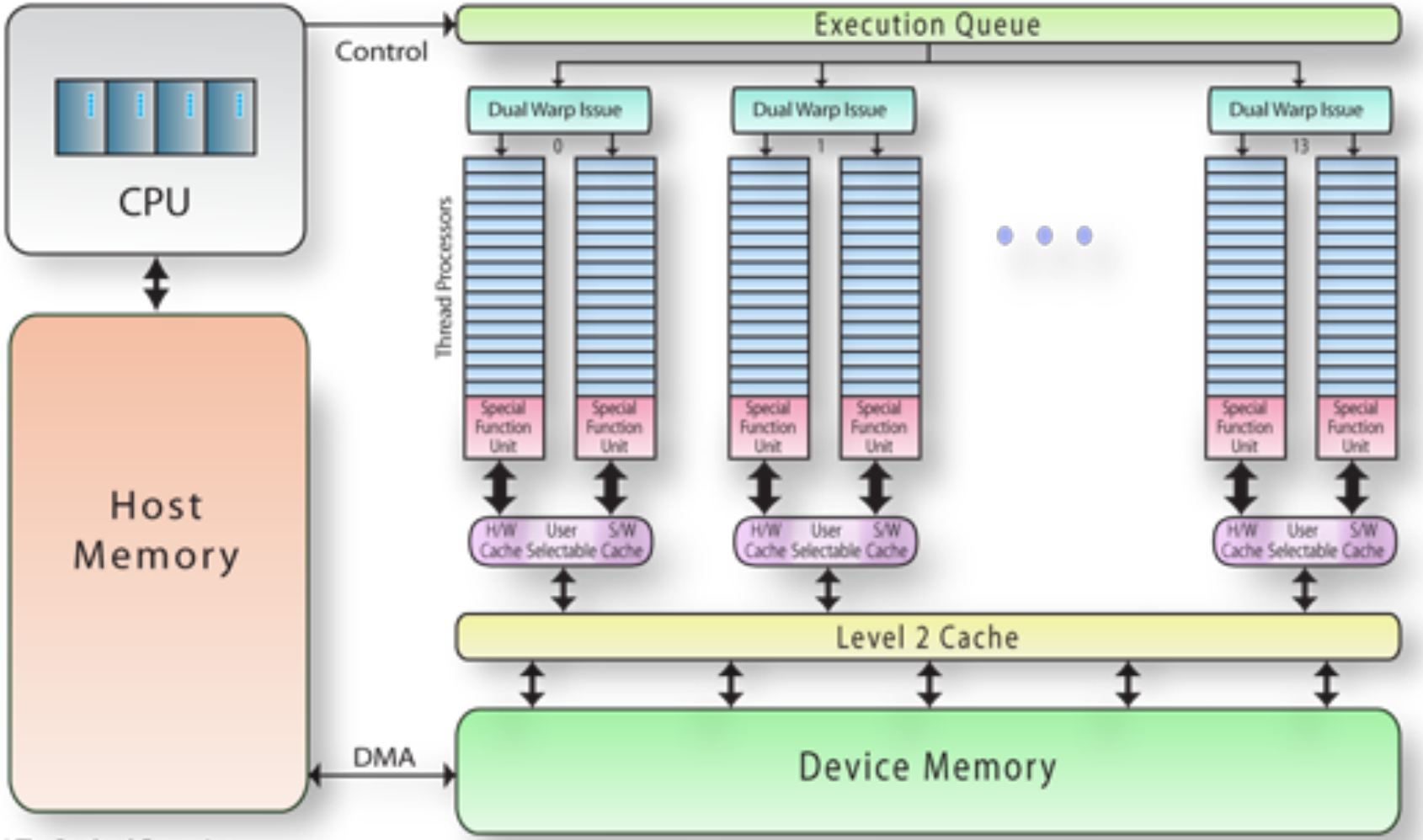
Parallelism

- Core-level parallelism ~ task/data parallelism (coarse)
  - 4-12 of powerful cores
    - Hardware hyperthreading (2x)
  - Local caches
  - Symmetrical or asymmetrical threading model
  - Implemented by programmer
- SIMD parallelism = data parallelism (fine)
  - 4-SP/2-DP floating point operations per second
    - 256-bit vectors
  - Run same instruction on different data
  - Sensitive to divergence
    - NOT the same instruction => performance loss
  - Implemented by programmer OR compiler

Programming models

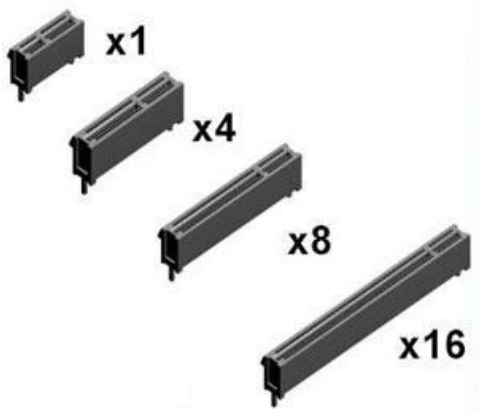
- Pthreads + intrinsics
- TBB – Thread building blocks
  - Threading library
- OpenCL
  - To be discussed …
- OpenMP
  - Traditional parallel library
  - High-level, pragma-based
- Cilk
  - Simple divide-and-conquer model

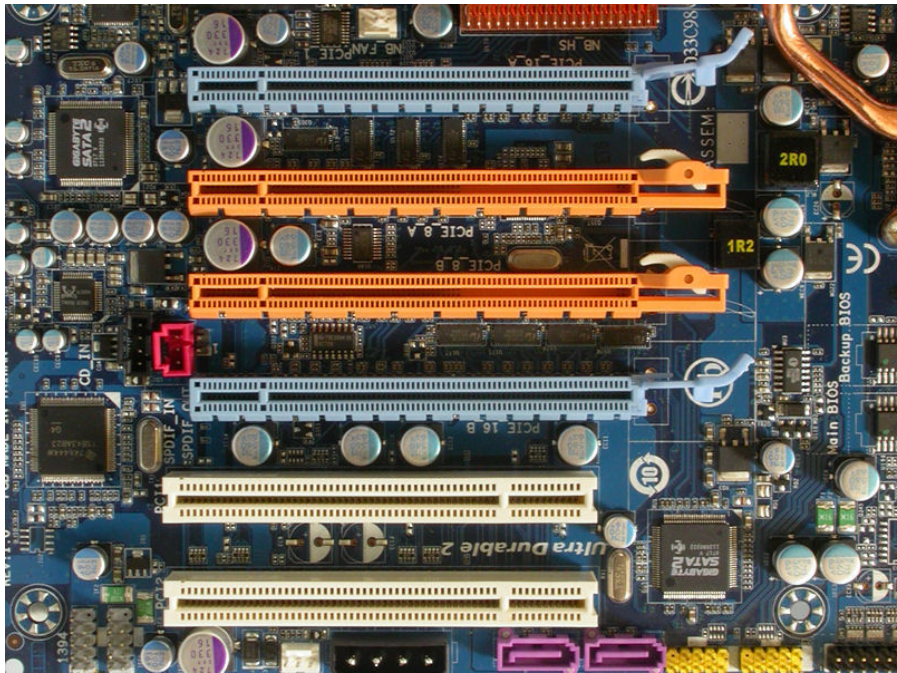
A GPU Architecture



Integration into host system

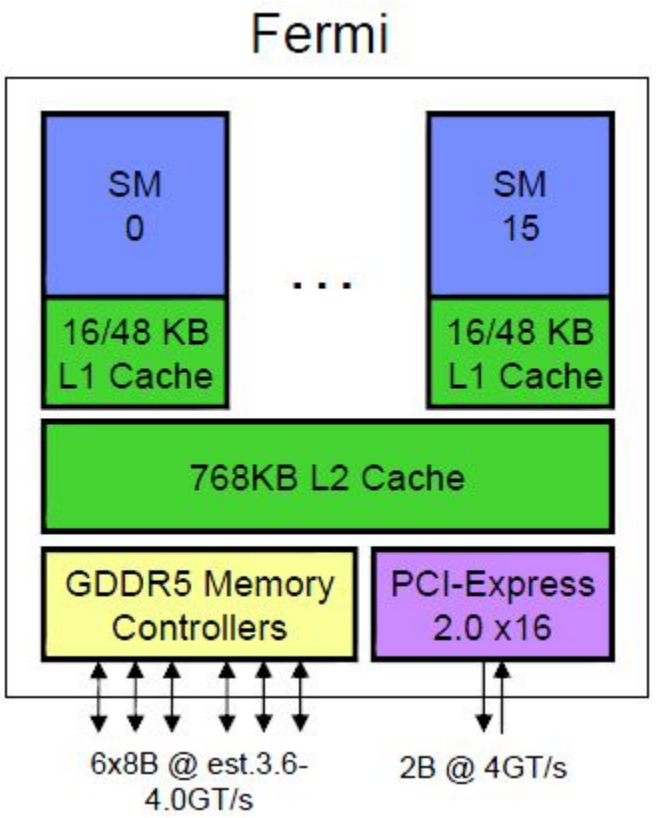
- Typically PCI Express 2.0
- Theoretical speed 8 GB/s
  - Effective ≤ 6 GB/s
  - In reality: 4 – 6 GB/s
- V3.0 recently available
  - Double bandwidth
  - Less protocol overhead





(NVIDIA) GPUs

- Architecture
  - Many (100s) slim cores
  - Sets of (32 or 192) cores grouped into “multiprocessors” with shared memory
    - SM(X) = stream multiprocessors
  - Work as accelerators
- Memory
  - Shared L2 cache
  - Per-core caches + shared caches
  - Off-chip global memory
- Programming
  - Symmetric multi-threading
  - Hardware scheduler



GPU Parallelism

- Data parallelism (fine-grain)
- **SIMT** (Single Instruction **Multiple Thread**) execution
  - Many threads execute concurrently
    - Same instruction
    - Different data elements
    - HW automatically handles divergence
  - Not same as SIMD because of multiple register sets, addresses, and flow paths*
- Hardware multithreading
  - HW resource allocation & thread scheduling
    - Excess of threads to hide latency
    - Context switching is (basically) free

*http://yosefk.com/blog/simd-simt-smt-parallelism-in-nvidia-gpus.html

Specific programming model: CUDA

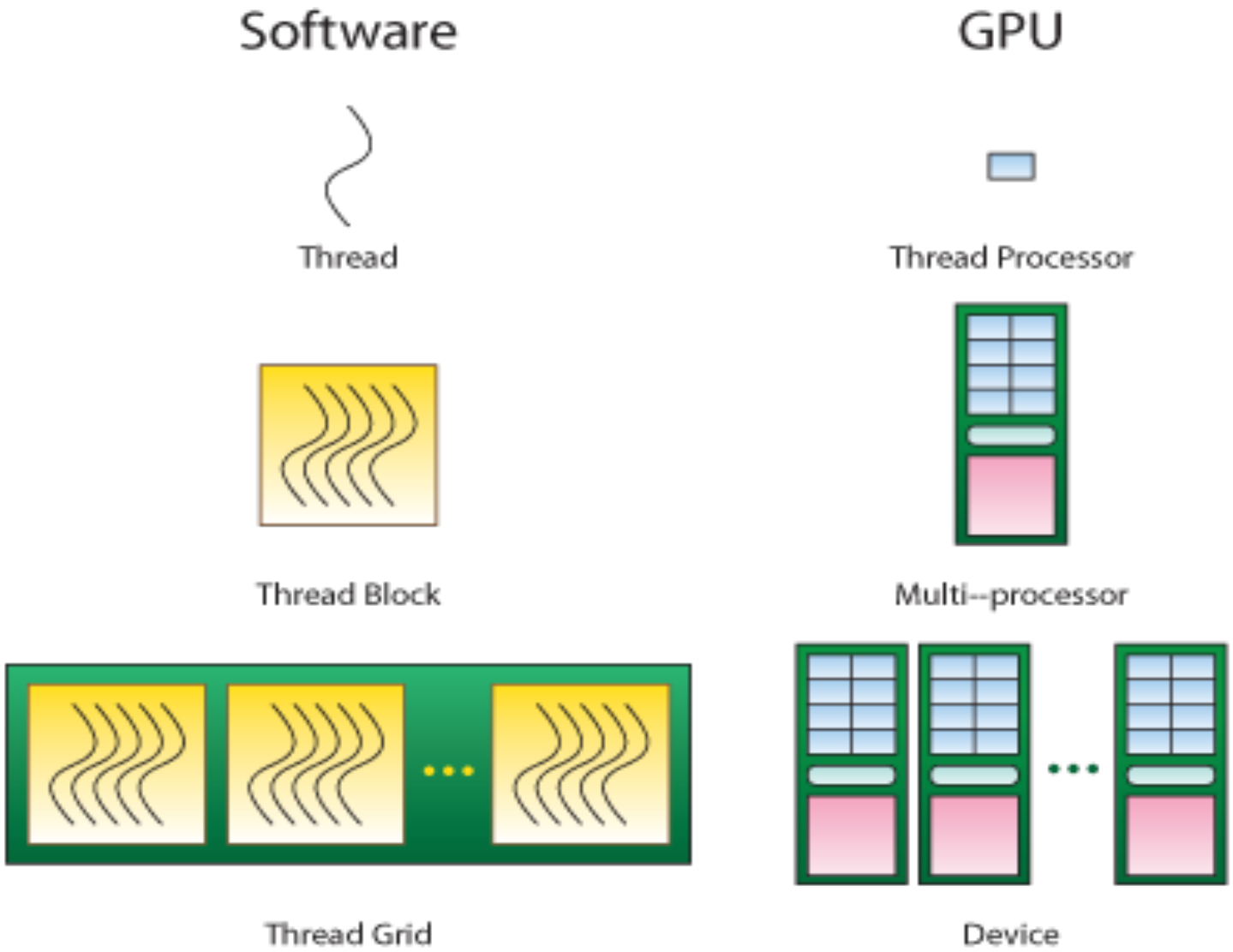
- CUDA: Compute Unified Device Architecture
  - C/C++ extensions
    - Other wrappers exist
- Straightforward mapping onto hardware
  - Hierarchy of threads (map to cores)
    - Configurable at logical level
  - Various memory spaces (map to physical mem. spaces)
    - Usable via variable scopes
- SIMT: single instruction multiple threads
  - Have 1000s threads running concurrently
  - Hardware multi-threading
    - GPU threads are lightweight

CUDA: Hierarchy of threads

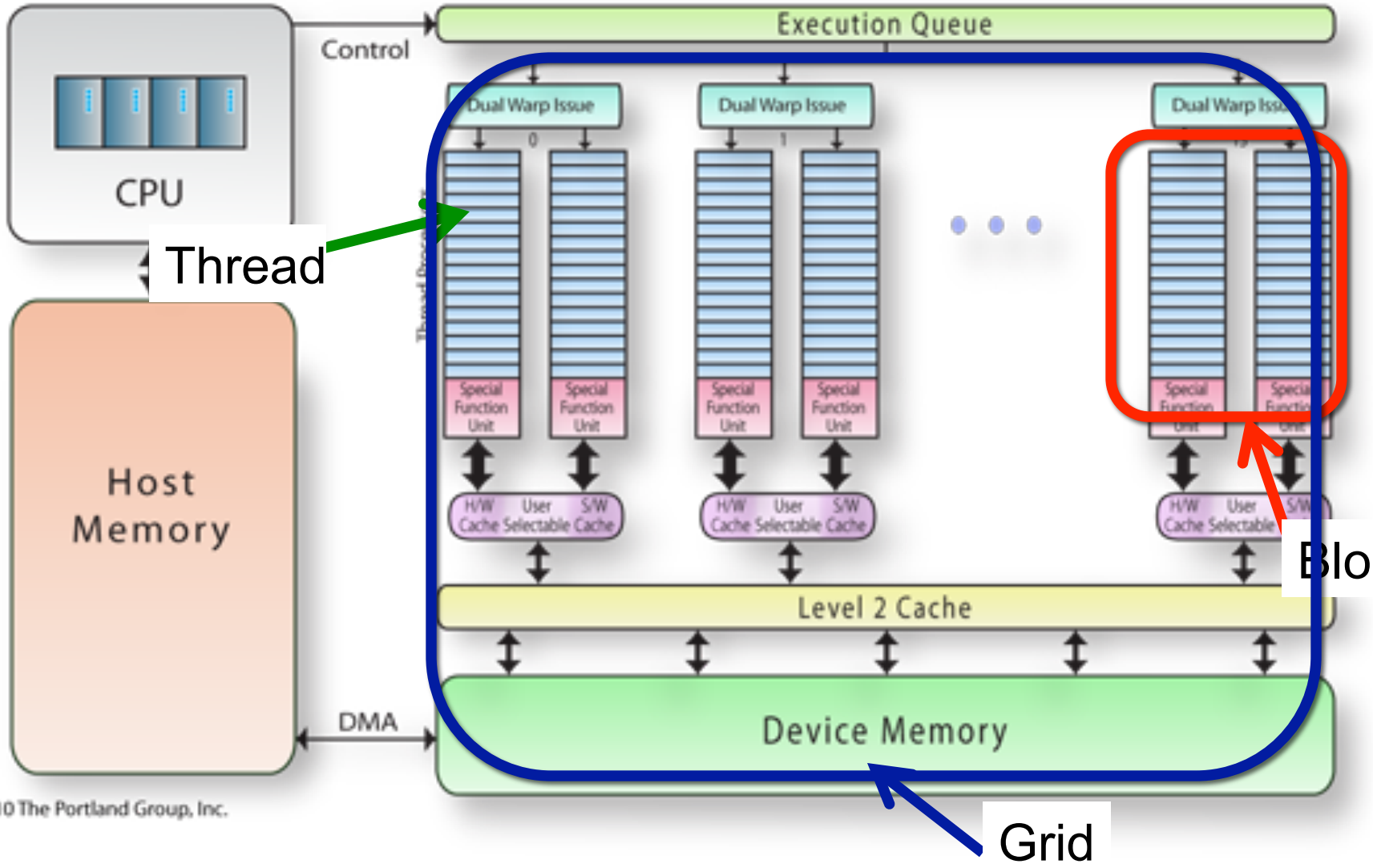
- Each thread executes the kernel code
  - One thread runs on one CUDA core
- Threads are logically grouped into thread blocks
  - Threads in the same block can cooperate
  - Threads in different blocks cannot cooperate
- All thread blocks are logically organized in a Grid
  - 1D or 2D or 3D
  - Threads and blocks have unique IDs

- A grid specifies in how many instances the kernel is being run

CUDA Model of Parallelism



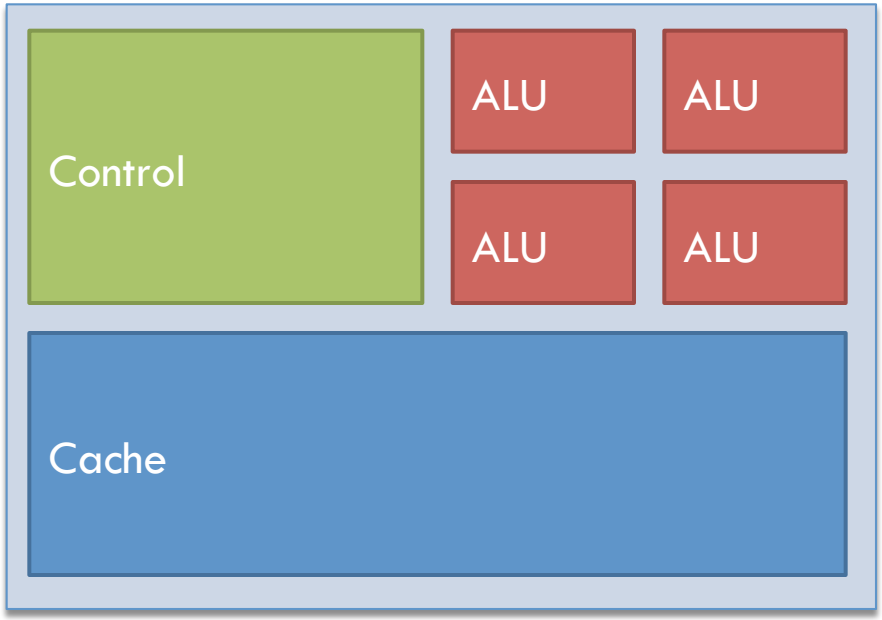
Hierarchy of threads



CUDA Model of Parallelism

- CUDA virtualizes the physical hardware
  - A block is a virtualized streaming multiprocessor
    - threads, shared memory
  - A thread is a virtualized scalar processor
    - registers, PC, state
- Execution model:
  - Threads execute in warps (32 threads per warp)
    - Called “wavefronts” by AMD (64 threads)
  - All threads in a warp execute the same code
    - On different data
  - Blocks = multiple warps
    - Scheduled independently on the same SM

CPU vs. Accelerator (GPU)

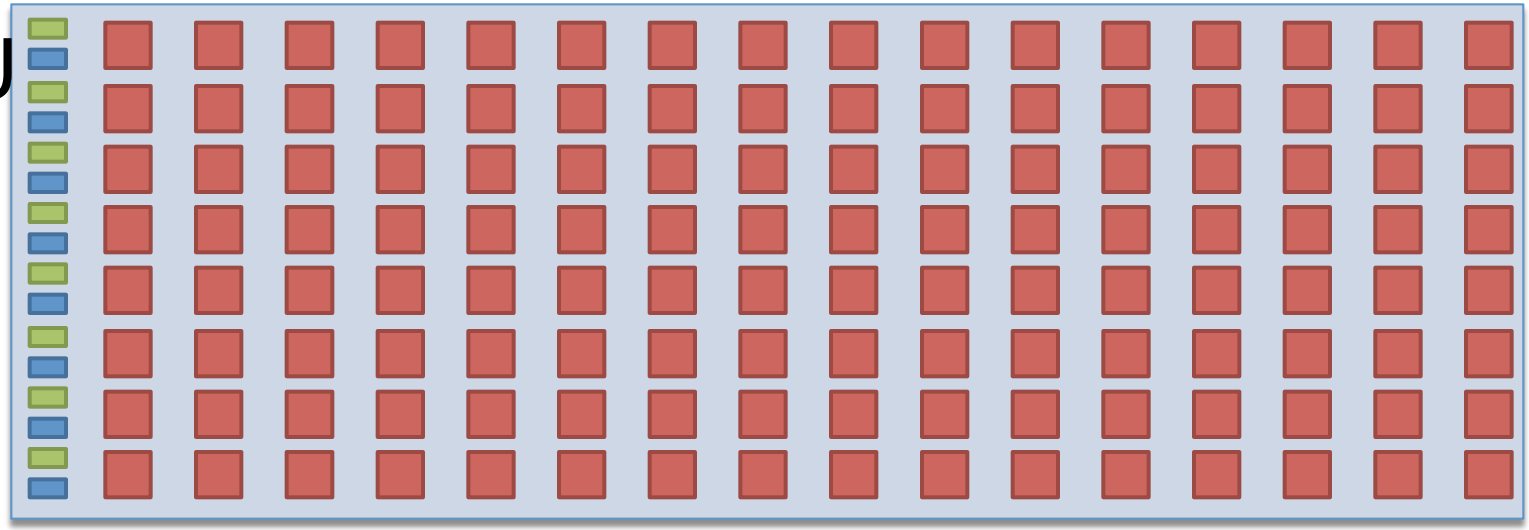


CPU

Low latency, high flexibility.
Excellent for irregular codes with limited parallelism.

GPU

High throughput.
Excellent for massively parallel workloads.

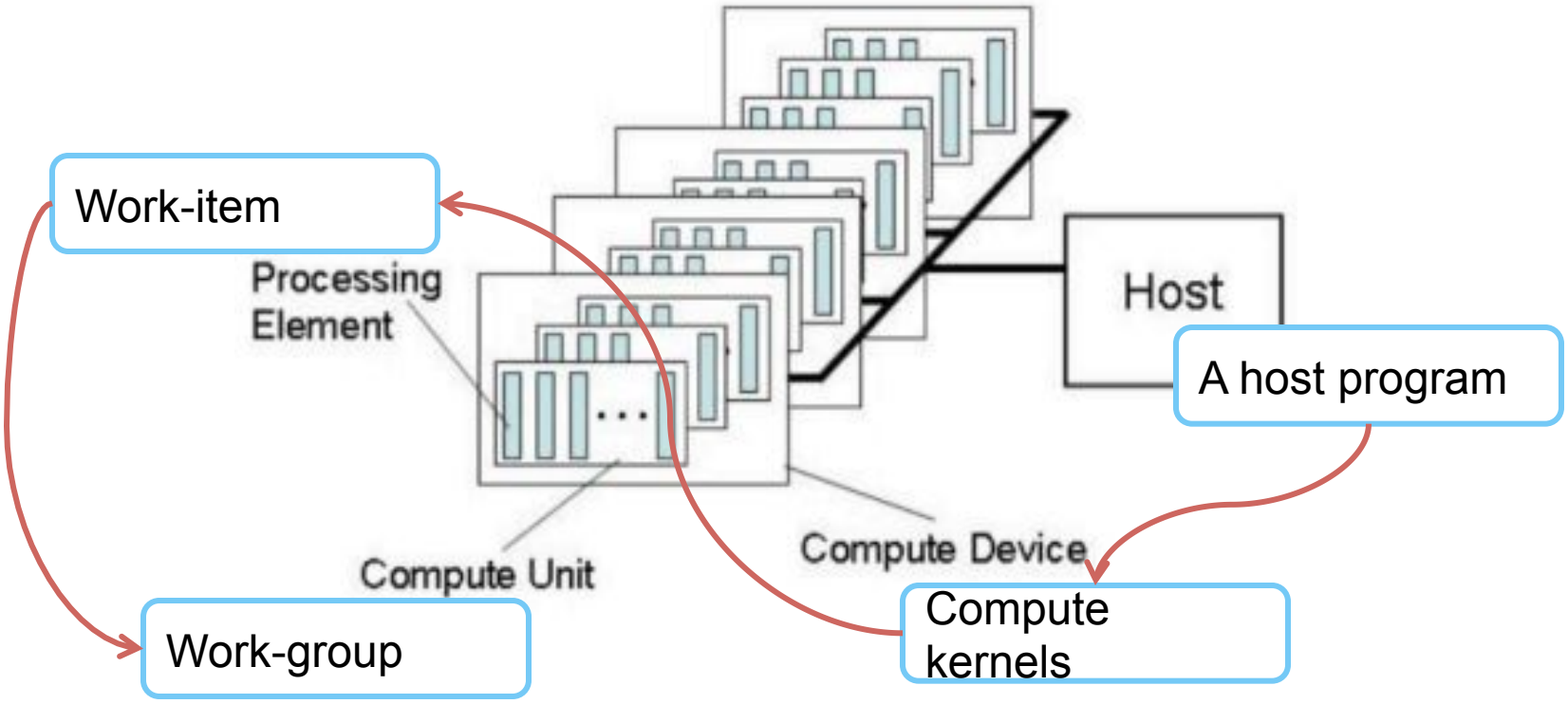


OpenCL in a nutshell

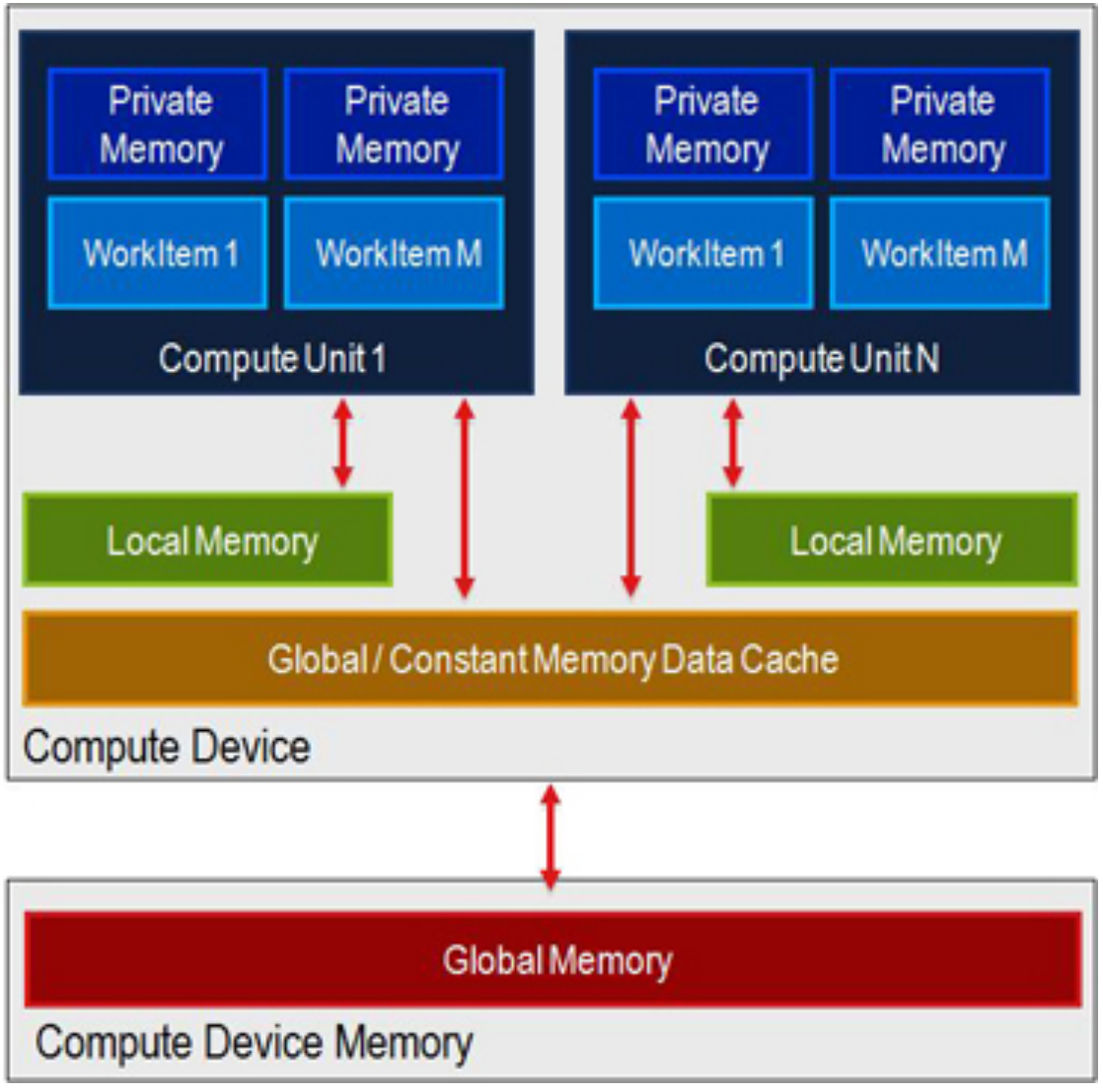
- CPU and GPU programming models are very different
  - Heterogeneous computing is hard
  - Performance comparison and portability are difficult to assess

- Open standard for portable multi-core programming
- Architecture independent
  - Explicit support for multi-/many-cores
- Low-level host API
    - High-level bindings (e.g., Java, Python)
- Separate kernel language
- Run-time compilation
- Supports (some) architecture-dependent optimizations
  - Explicit & implicit

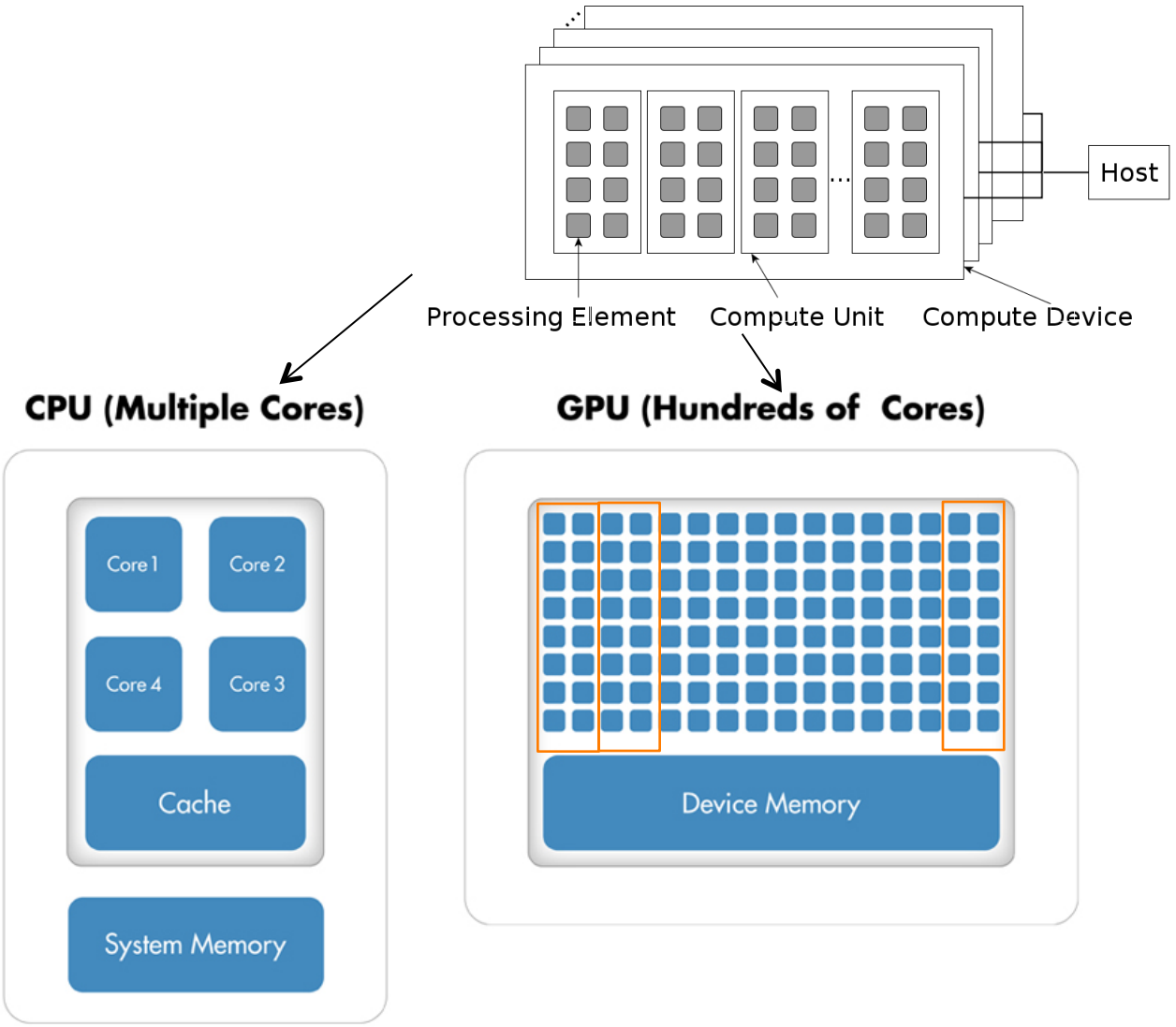
The OpenCL platform model



The OpenCL memory model



The OpenCL virtual platform



Programming in OpenCL

- Kernels are the main functional units in OpenCL
  - Kernels are executed by work-items
  - Work-items are mapped transparently on the hardware platform
- **Functional portability** is guaranteed
  - Programs run correctly on different families of hardware
  - Explicit platform-specific optimizations are dangerous

OpenCL for heterogeneous platforms

- Functional portability guaranteed by the standard
- Performance portability is NOT guaranteed
  - vs. CUDA:
    - Used to be comparable (2012)
    - Lagging behind due to lack of support from NVIDIA
  - Vs. OpenMP/other CPU models: 3 challenges
    - GPU-like programming styles
      - Memory access patterns, data transfer, (local memory) => MUST remove
    - Parallelism granularity

OpenCL is an efficient programming model for heterogeneous platforms iff we specialize the code to fit different processors.

PART I

Heterogeneous processing: pro’s and con’s