HETEROGENEOUS PARALLEL COMPUTING WITH OPENCL

Sebastian Klose
kloses@in.tum.de
27.05.2013

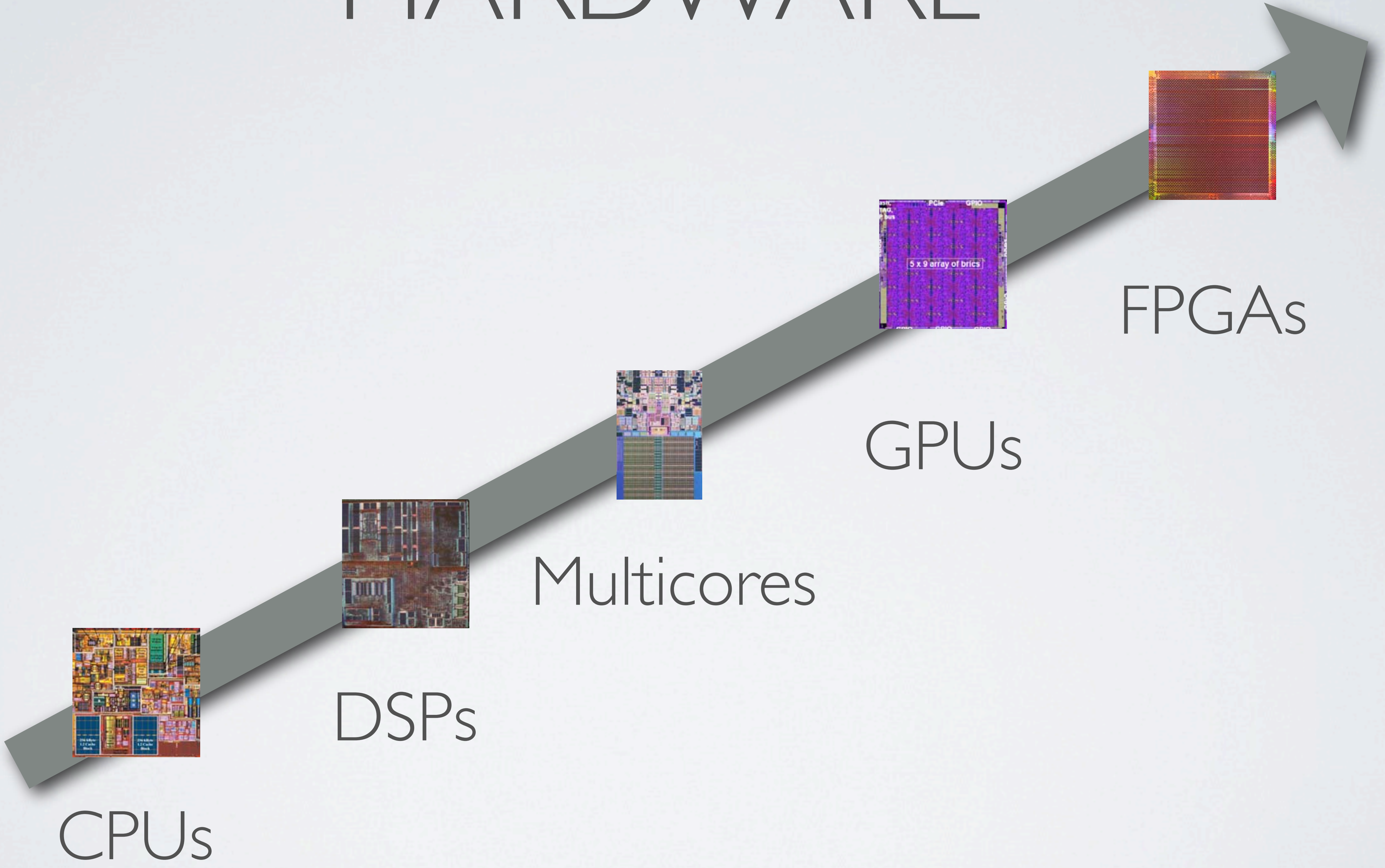
Robotics and
Embedded Systems

AGENDA

- Motivation
- OpenCL Overview
- Hardware Examples
- OpenCL on Altera FPGAs

MOTIVATION

TREND OF PROGRAMMABLE HARDWARE



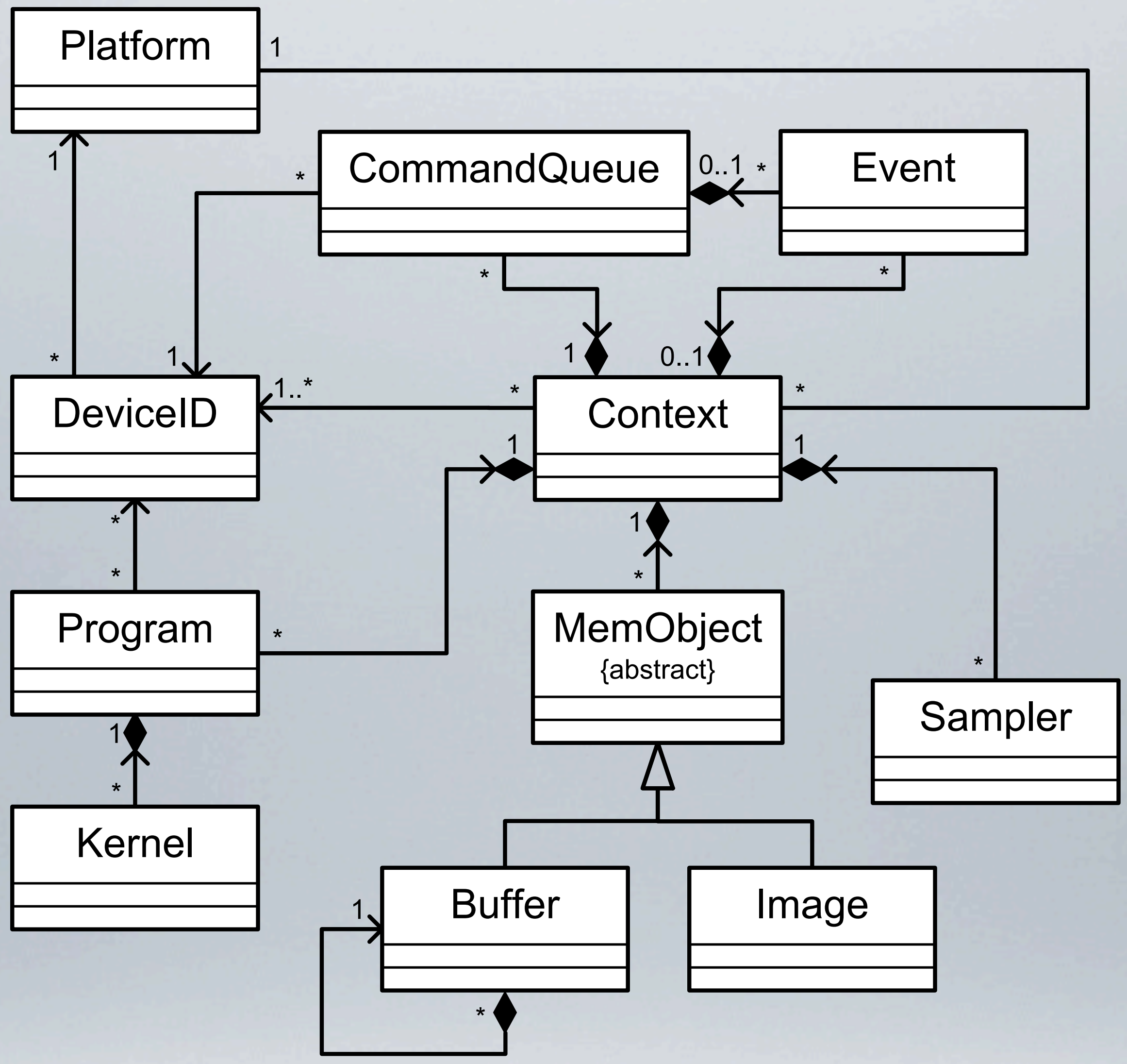
Robotics and
Embedded Systems

OPENCL GOALS

- Khronos standard for unified programming of heterogeneous hardware (GPU, CPU, DSP, Embedded Systems, FPGAs, ...)
- Initiated by Apple (2008) - Dec. 2008 OpenCL 1.0 Release
- explicit declaration of parallelism
- portability/code reuse: same code for different hardware
- ease programmability of parallel hardware

AVAILABILITY

SDK	Hardware	Version
Intel OpenCL SDK	Core i3/i5/i7 CPU & Integrated Intel HD GPU	1.2
AMP APP SDK	AMD GPUs X86 CPUs	1.2
Apple	CPUs & GPUs	1.2
Altera	selected boards	1.0
Qualcomm	Qualcomm CPU & GPU	1.1
ARM	Mali T604 GPU	1.1
	...	

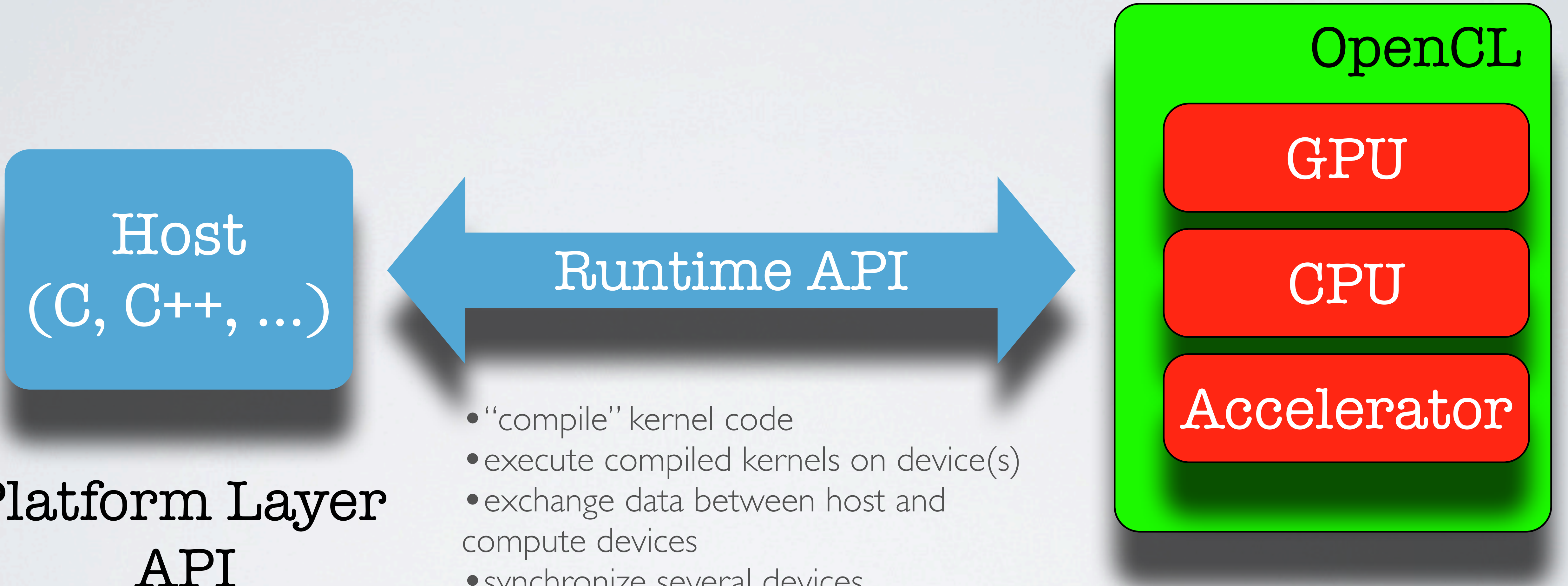


OPENCL OVERVIEW

Robotics and
Embedded Systems

Montag, 27. Mai 13

OPEN COMPUTING LANGUAGE



Platform Layer API

- query/select devices of host
- initialize compute devices

- "compile" kernel code
- execute compiled kernels on device(s)
- exchange data between host and compute devices
- synchronize several devices

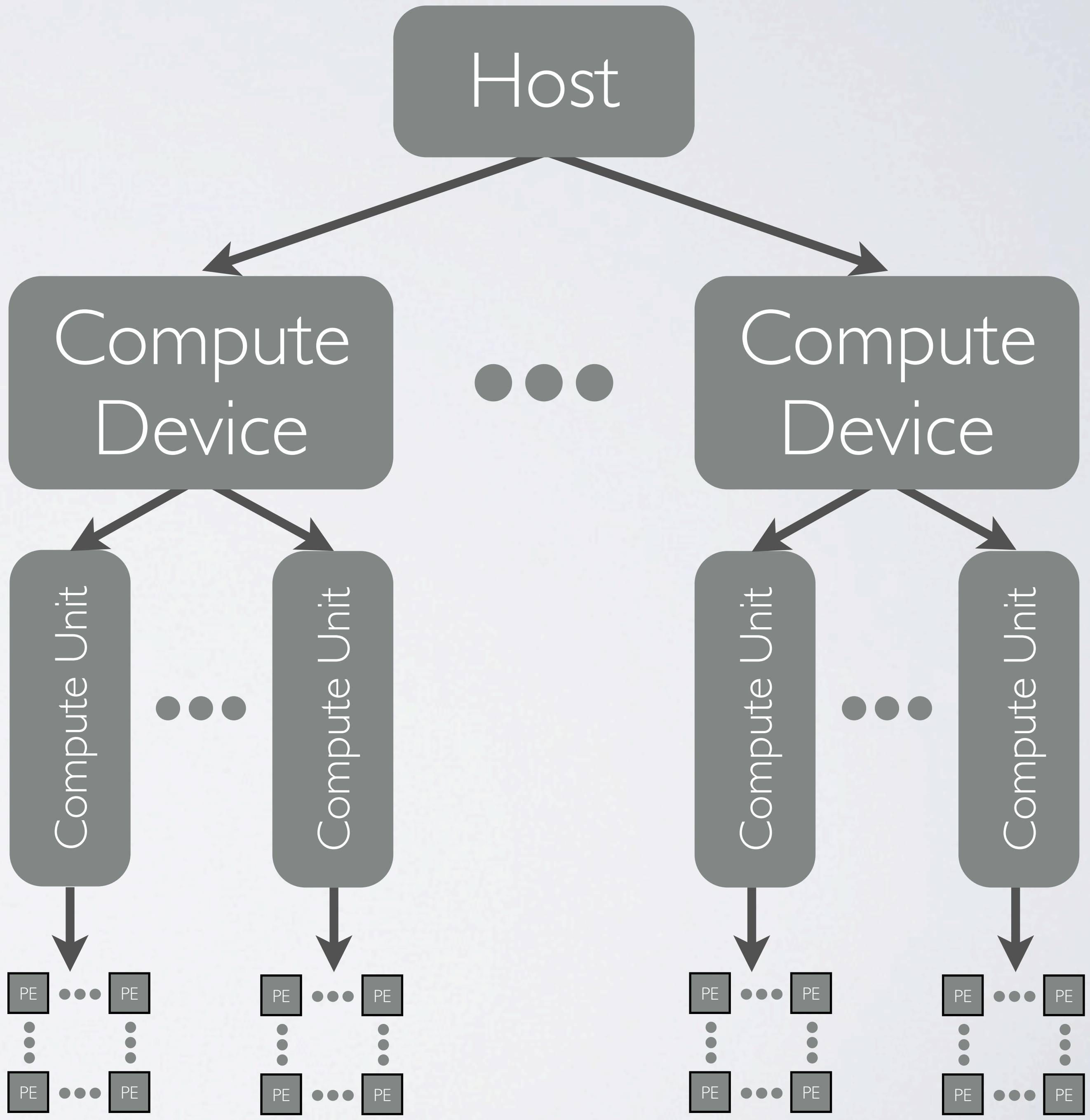
OpenCL C Language (Kernel)

- C99 subset + extensions
- built-in functions
- (sin, cos, cross, dot, ...)

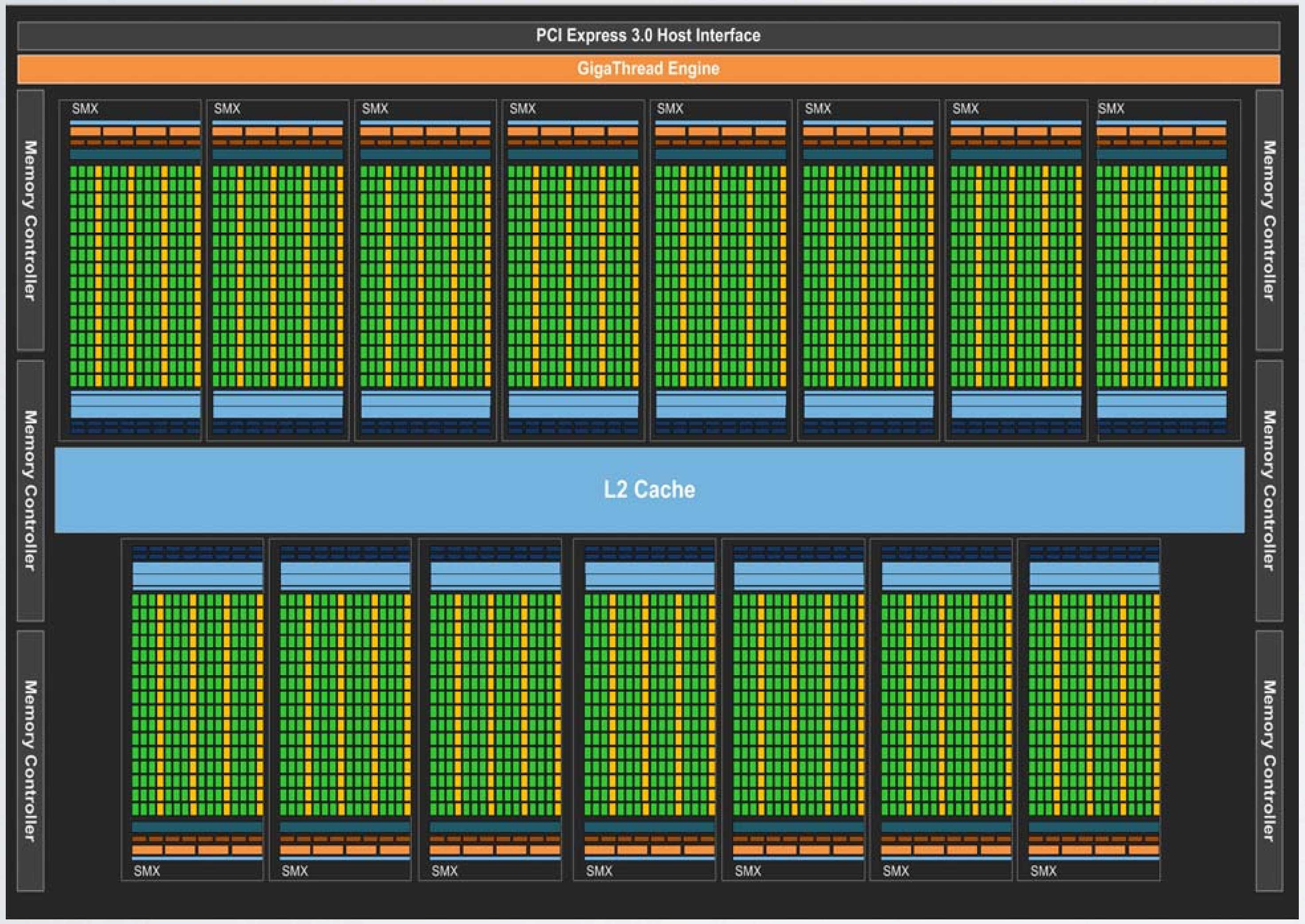
Robotics and Embedded Systems

PLATFORM MODEL

- one host
- one or more compute devices
- compute devices are composed of one or more compute units
- compute units are divided into one or more processing elements



EXAMPLE: NVIDIA KEPLER GK110



- 15 SMX = Compute Units
- 1SMX = 192 Processing Elements

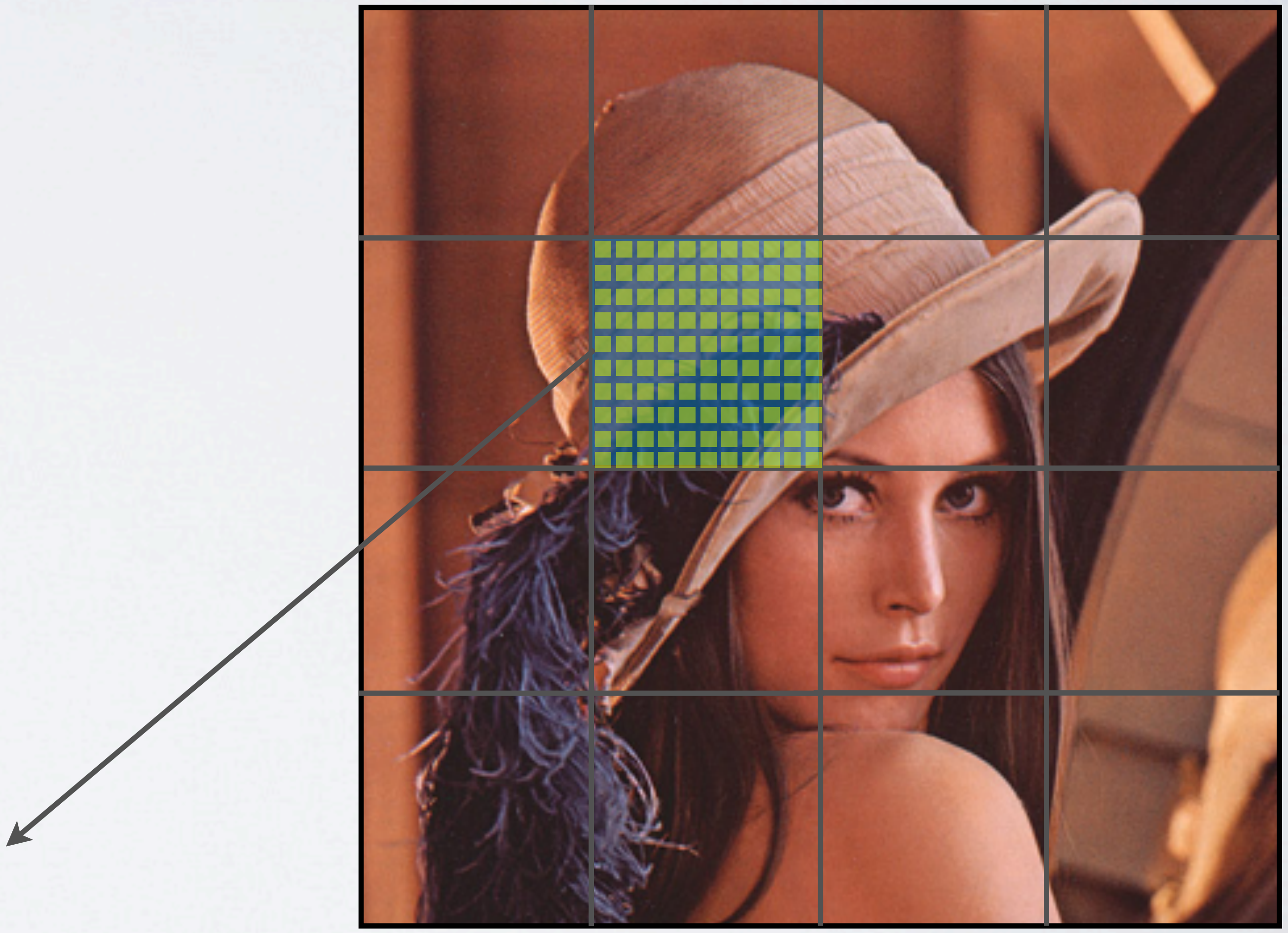
Source: http://www.nvidia.de/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf

EXECUTION MODEL

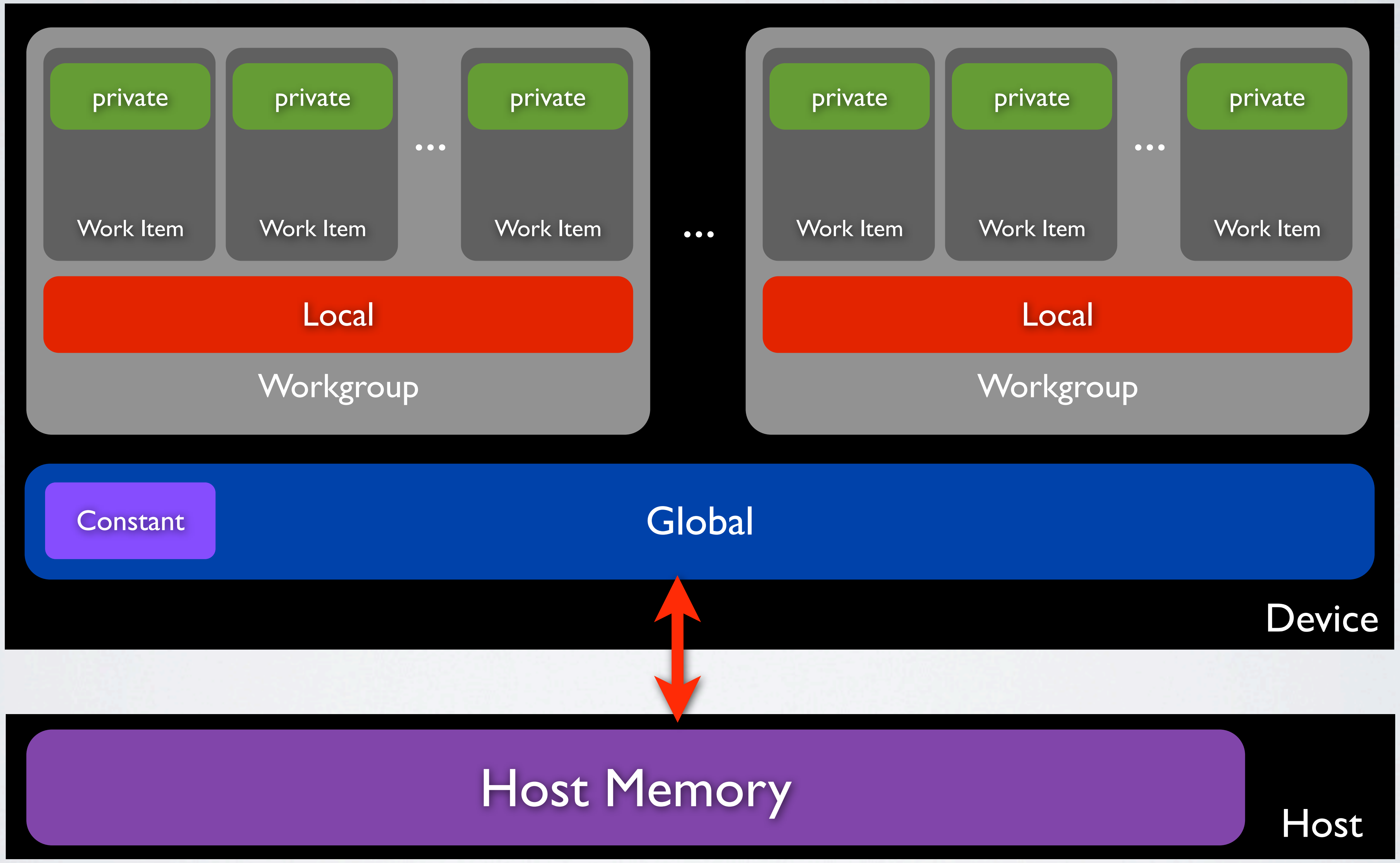
- **Kernel**
  unit of executable code - data parallel or task parallel
- **Program**
  collection of kernels and functions (think of dynamic library)
- **Command Queue**
  host application queues kernels & data transfers
  in order or out of order
- **Work-Item**
  execution of a kernel by a single processing element (think of thread)
- **Work-Group**
  collection of work-items that execute on a single compute unit (think of cpu core)

SPECIFY PARALLELISM

- say our "global" problem consists of 1024x1024 tasks, which can be executed in parallel
- therefore we have 1024x1024 work-items
- group several work-items into local workgroups



MEMORY MODEL

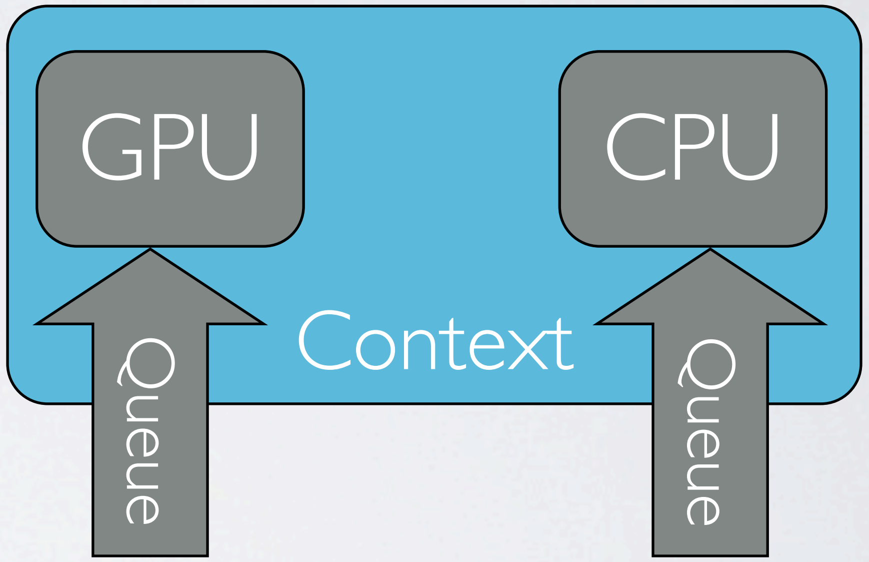


Robotics and
Embedded Systems

EXECUTION MODEL

- host application submits work to the compute devices via command queues

- **context:** environment within which work-items executes
  includes: devices, memories and command queues



SYNCRONIZATION

- **Events:**
  synchronize kernel executions between different queues in the same context
- **Barriers**
  synchronize kernels within a queue

SIMPLE KERNEL EXAMPLE

```
void inc( float* a, float b, int N )
{
   for( int i = 0; i < N; ++i )
   {
      a[ i ] = a[ i ] + b;
   }
}

void main( void )
{
   ...
   inc( a, b, N );
}
```

```
kernel
void inc( global float* a, float b )
{
   int i = get_global_id( 0 );
   a[ i ] = a[ i ] + b;
}

// host code
void main( void )
{
   ...
   clEnqueueNDRangeKernel( ..., &N, ... );
}
```

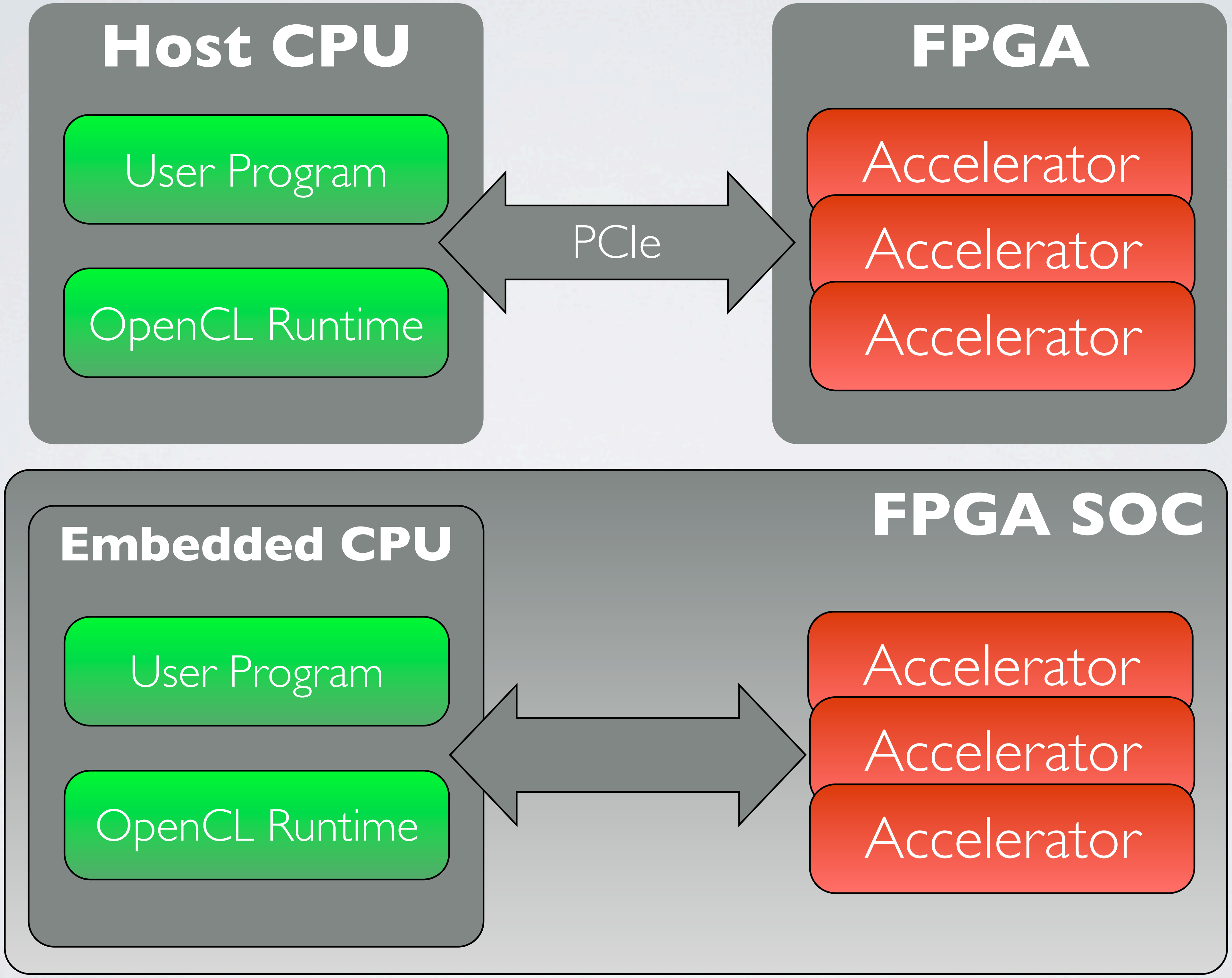
- inside the **kernel** you (e.g.) program the execution of the inner part of a loop
- **built-in** kernels can be used to exploit special hardware units

+

OPENCL ON ALTERA FPGAS

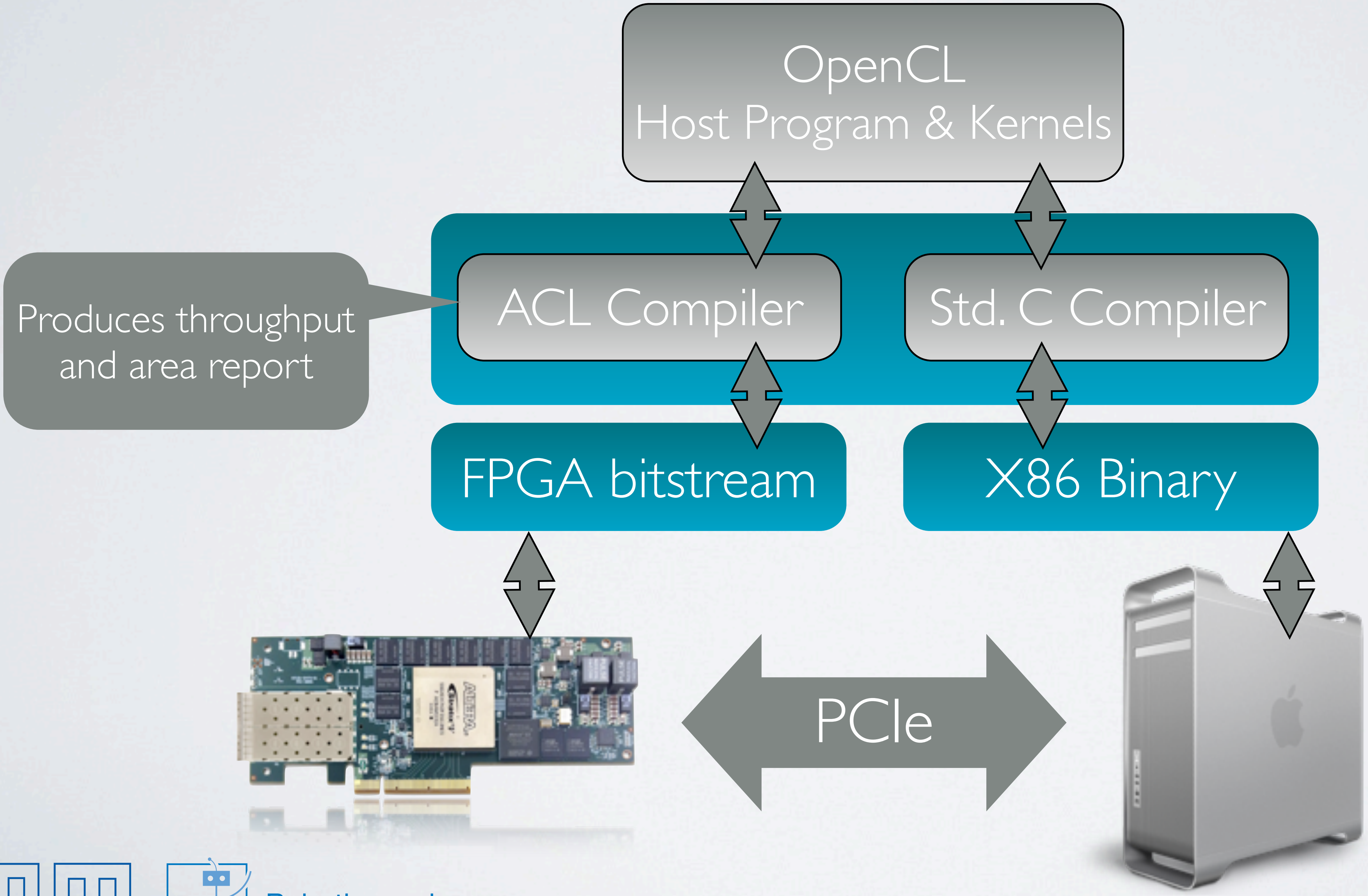
Robotics and
Embedded Systems

ALTERAS EXECUTION MODEL



Robotics and
Embedded Systems

ALTERA OPENCL COMPILATION



Robotics and Embedded Systems

DIFFERENCES TO OTHER OPENCL IMPLEMENTATIONS

- in contrast to CPU/GPU, specialized datapaths are generated
- for each kernel, custom hardware is created
- logic is organized in functional units, based on operation and linked together to form the dedicated datapath required to implement the special kernel
- execution of multiple workgroups in parallel in a custom fashion
- pipeline parallelism

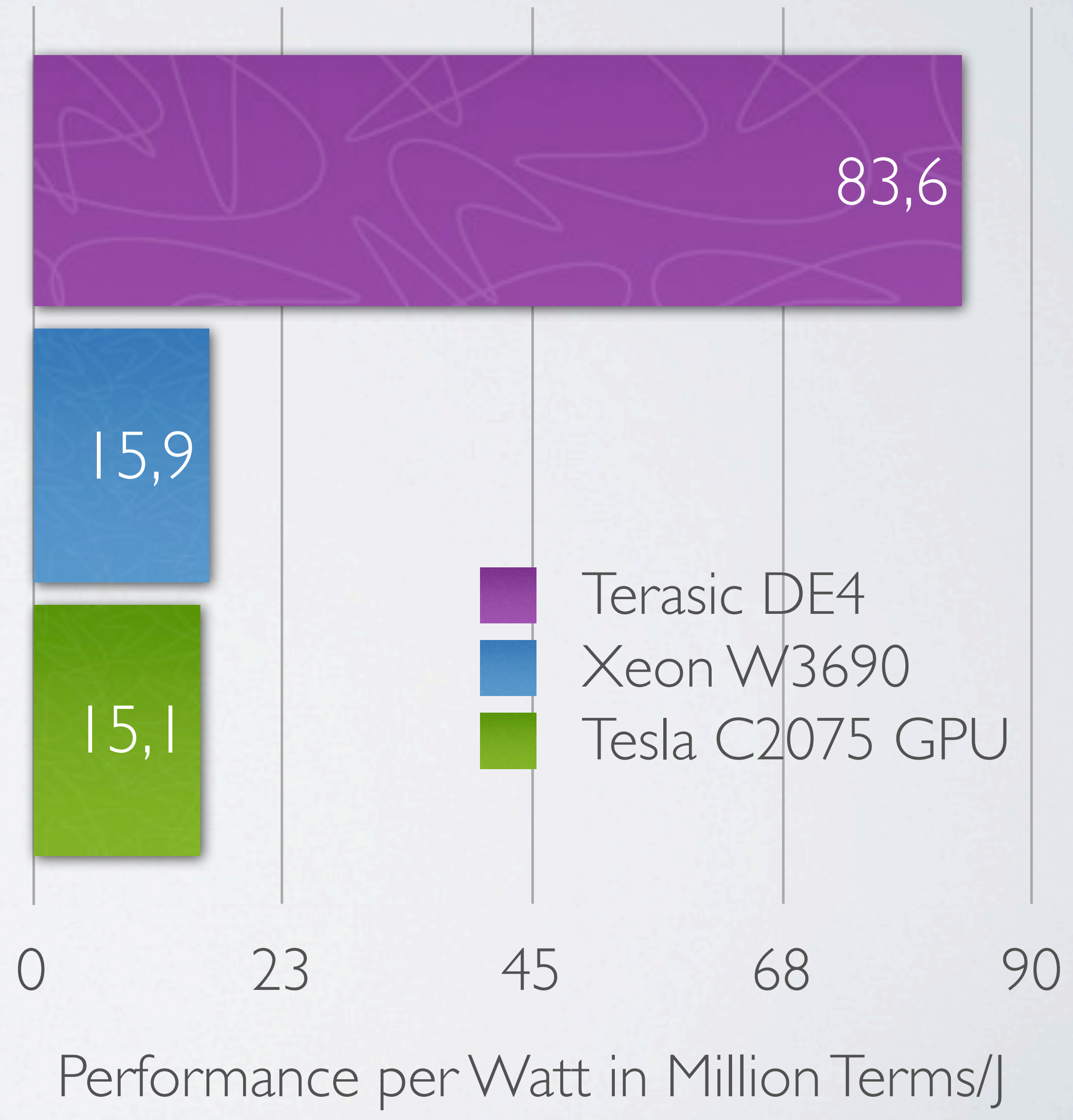
Robotics and
Embedded Systems

ALTERA LIMITATIONS

- **OpenCL Version 1.0** (nearly completely)
#contexts: 2	#cmd_queues/ctxt = 3;
#program_objects/ctxt = 3	#outstanding events/ctxt around 100
#concurrently running kernels=3	#kernels / device = 16
#arguments / kernel = 128	#work_items / work_group = 256
- max 256MB of host-side memory (CL_MEM_ALLOC_HOST_PTR)
- max 2GB DDR as device global memory
- No support for out-of order execution of command queues
- *no Image read and write function (atm), no explicit memory fences, no floating point exceptions*
- Memory:
  - **global** memory visible to all work-items (e.g. DDR Memory on FPGA)
  - **local** memory visible to a work-group (e.g. on-chip memory)
  - **private** memory visible to a work-item (e.g. registers)

BENEFITS

- shorter design cycles & time to market
- performance
- performance per watt
- explicit parallelism
- "portable"



Robotics and Embedded Systems