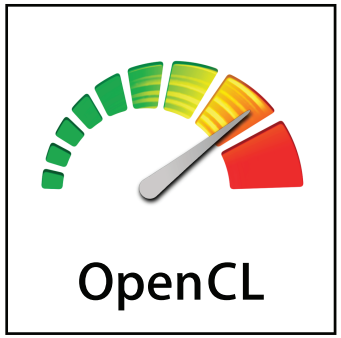
Portable Programs for Heterogeneous Computing: A Hands-on Introduction

Tim Mattson
Intel Corp.

Alice Koniges
Berkeley Lab/NERSC

Simon McIntosh-Smith
University of Bristol



Agenda

Lectures	Exercises
OpenCL Overview --The OpenCL Python Environment	Exercises to Explore the Spec --Logging on and Accounts at NERSC and other Systems --Vector addition --Matix Multiplication
OpenMP Overview --The OpenMP target directive	Exercises to Introduce OpenMP Accelerator Directives --Matrix Multiplication: sending loops to an attached device --The Pi program reductions and the target directive
Student Exploration	A variety of examples depending on student expertise. We offer a choice of beginning/intermediate/advanced programs on a several architectures

OpenCL, OpenMP, Python, and Editors: Reference Cards

To aid the students, we provide reference cards to help with the language standards as well as editors to perform the exercises either remotely on NERSC or on their own laptops. Here is a sample.

OpenCL C 1.2 Reference Card
OpenCL C++ 1.2 Reference Card

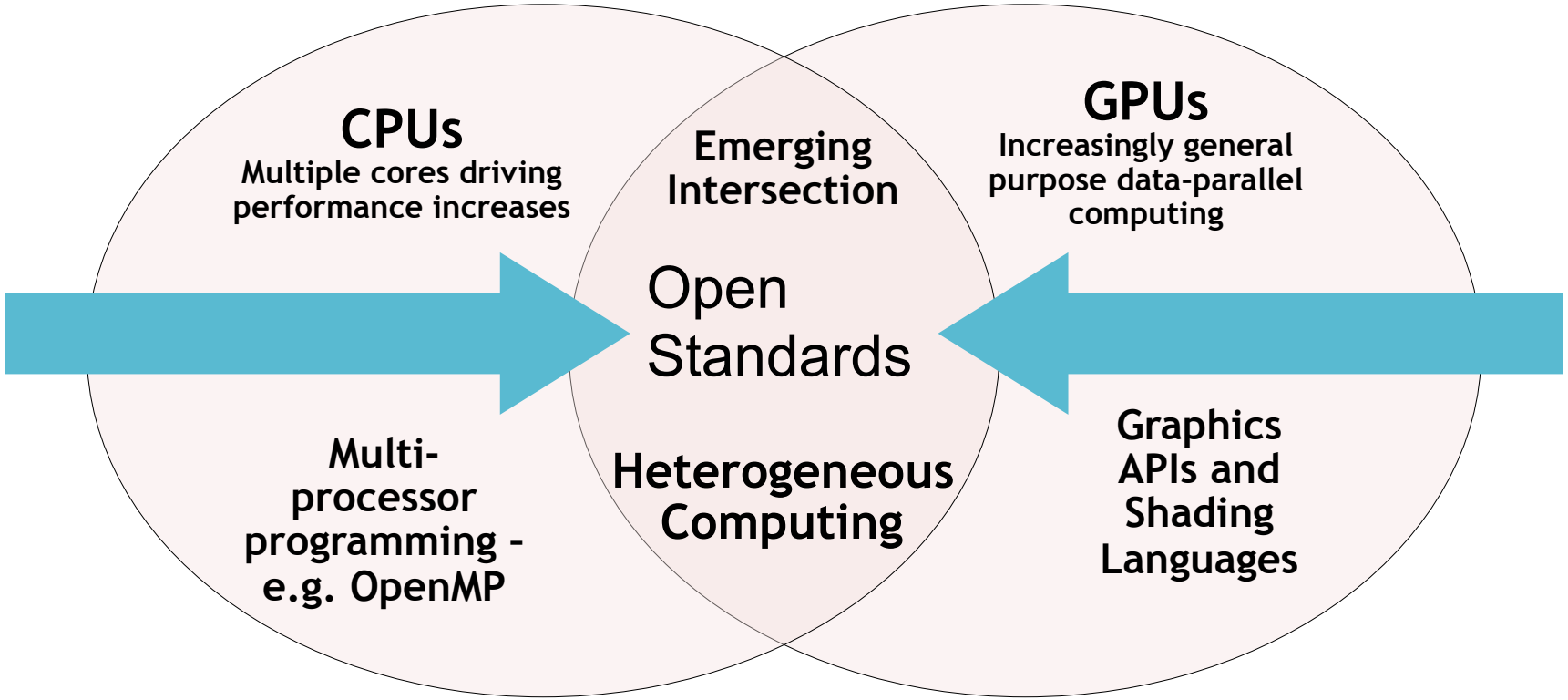
These cards will help you keep track of the API as you do the exercises:

https://www.khronos.org/files/opencl-1-2-quick-reference-card.pdf

The v1.2 spec is also very readable and recommended to have on-hand:

https://www.khronos.org/registry/cl/specs/opencl-1.2.pdf

Industry Standards for Programming Heterogeneous Platforms



Open, royalty-free standard for portable, parallel programming of heterogeneous parallel computing CPUs, GPUs, and other processors

Outline/Schedule (part 1)

OpenCL overview (45 minutes)

- OpenCL history and motivation
- General models in OpenCL
- Hands-on: Accessing the servers we'll be using in the course

- The OpenCL python environment (45 minutes)
  - The OpenCL Host API
  - The python interface to the Host API
  - Hands-on: running a canned program (to test the environment)
- Running a basic OpenCL program: part 1 (30 minutes)
  - Hands-on: Vector addition: the basic platform layer and writing your own host code.
- Running a basic OpenCL program: part 2 (30 minutes)
  - Hands-on: Matrix multiplication: writing simple kernels
- Optimizing Kernel code (60 minutes)
  - The OpenCL memory model
  - Hands-on: using local and private memory, the pi program
  - Kernel performance pitfalls
  - Hands-on: optimizing matrix multiplication

Outline/Schedule (part 2)

OpenMP overview (30 min)

- OpenCL history and motivation
- General models in OpenCL
- Hands-on: Accessing the servers we'll be using in the course

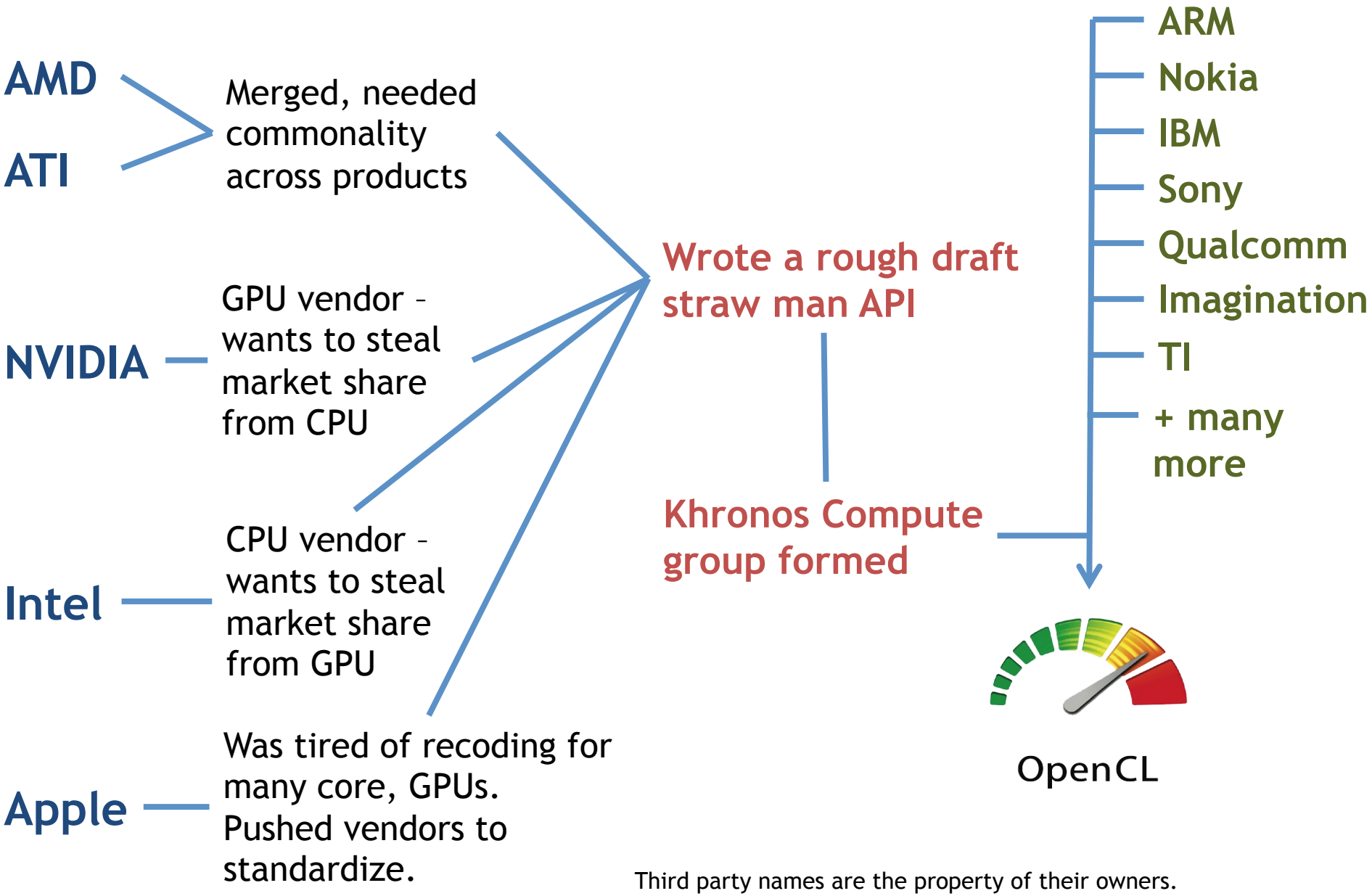
- The OpenMP target directive (60 minutes)
  - Matrix multiplication: sending loops to an attached device
  - The Pi program: reductions and the target directive
- Portable programming goals and basic concepts (60 minutes)
  - Hands-on: Student exploration: GPUs, CPUs, and the Intel® Xeon Phi™ processor
- Key design patterns and basic lessons of portable parallel programming (30 Minutes)

Outline/Schedule (part 3)

- Continued Exploration on the provided “zoo” of architectures
  - TBD, depending on hardware available when course starts

Here we give examples of the quality of slides.
based on our previous tutorials. Since the material is continuously evolving, we have not prepared the entire set yet for this particular tutorial. However quality is of foremost importance to us. Please note that the slides are clear, easy to read, and with minimal background distractions. The slides are designed with programmers in mind, so they help one to understand and create good code.

The origins of OpenCL

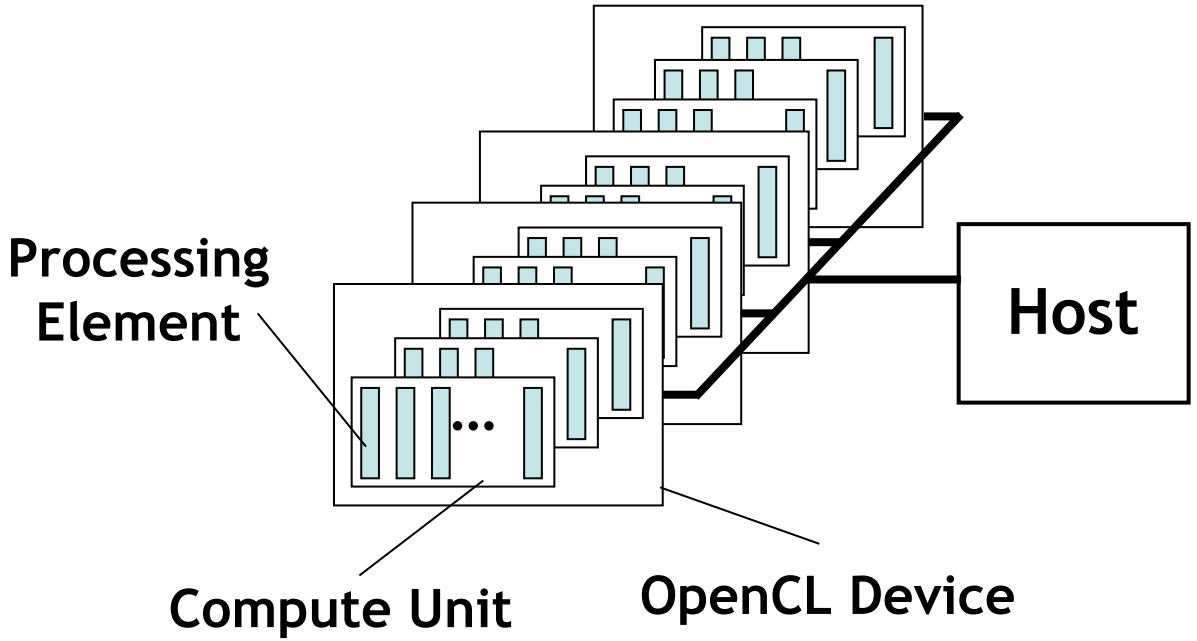


OpenCL: From cell phone to supercomputer

- OpenCL Embedded profile for mobile and embedded silicon
  - Relaxes some data type and precision requirements
  - Avoids the need for a separate “ES” specification
- Khronos APIs provide computing support for imaging & graphics
  - Enabling advanced applications in, e.g., Augmented Reality
- OpenCL will enable parallel computing in new markets
  - Mobile phones, cars, avionics

A camera phone with GPS processes images to recognize buildings and landmarks and provides relevant data from internet

OpenCL Platform Model



- One ***Host*** and one or more ***OpenCL Devices***
  - Each OpenCL Device is composed of one or more ***Compute Units***
    - Each Compute Unit is divided into one or more *Processing Elements*
- Memory divided into ***host memory*** and ***device memory***

The BIG idea behind OpenCL

- Replace loops with functions (a kernel) executing at each point in a problem domain
  - E.g., process a 1024x1024 image with one kernel invocation per pixel or 1024x1024=1,048,576 kernel executions

Traditional loops

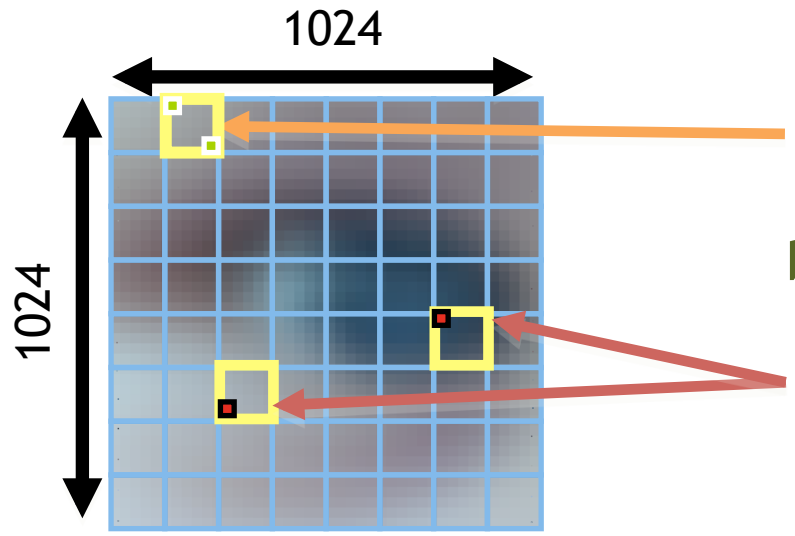
```
void
mul(const  int n,
    const  float *a,
    const  float *b,
           float *c)
{
  int i;
  for (i = 0; i < n; i++)
    c[i] = a[i] * b[i];
}
```

OpenCL

```
__kernel void
mul(__global const float *a,
    __global const float *b,
    __global       float *c)
{
  int id = get_global_id(0);
  c[id]  = a[id] * b[id];
}
// execute over n work-items
```

An N-dimensional domain of work-items

- **Global** Dimensions:
  - 1024x1024 (whole problem space)
- **Local** Dimensions:
  - 128x128 (**work-group**, executes together)



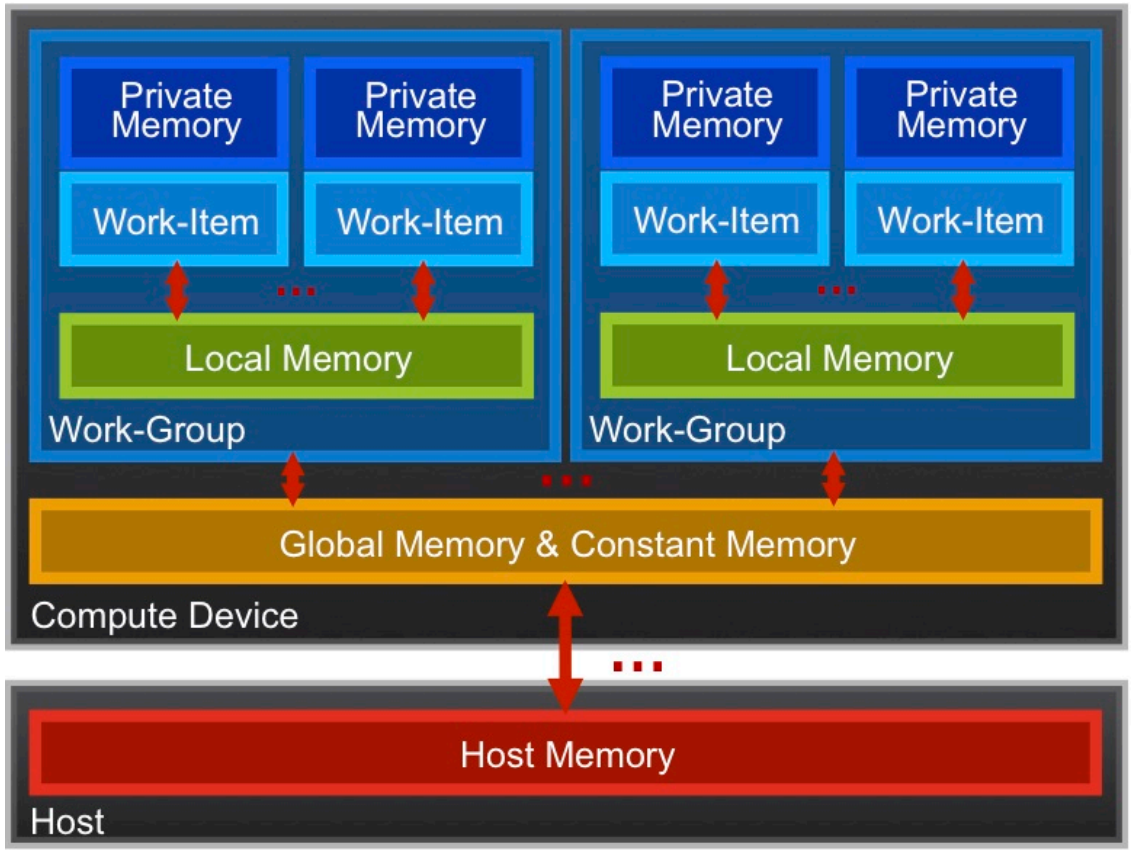
Synchronization between work-items possible only within work-groups: barriers and memory fences

Cannot synchronize between work-groups within a kernel

- Choose the dimensions (1, 2, or 3) that are "best" for your algorithm

OpenCL Memory model

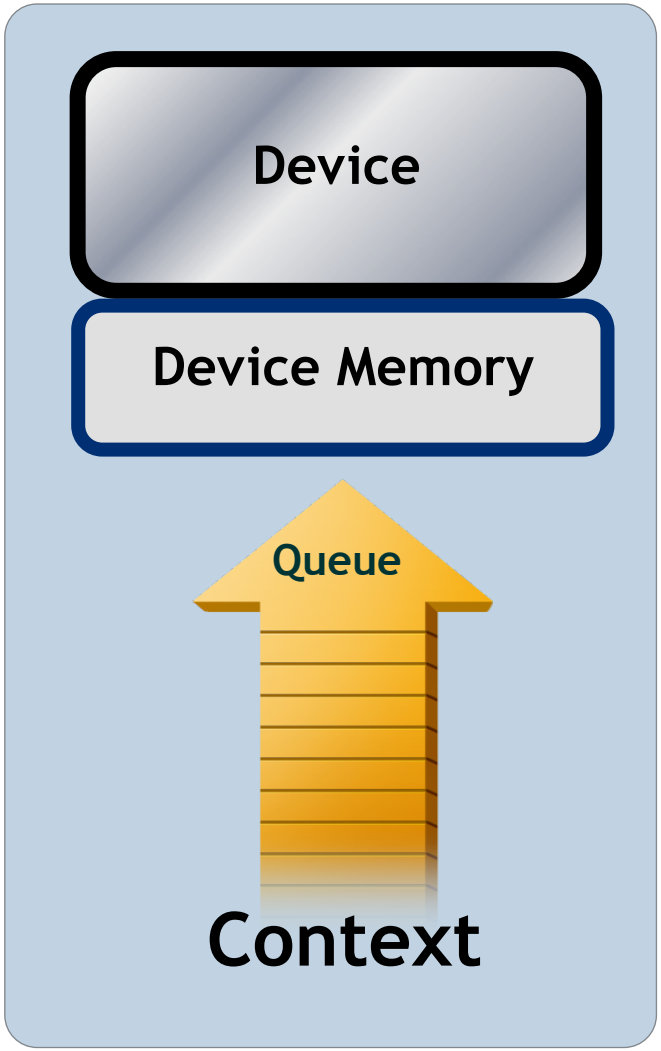
- ***Private Memory***
  - Per work-item
- ***Local Memory***
  - Shared within a work-group
- ***Global Memory***
  Constant Memory
  - Visible to all work-groups
- ***Host memory***
  - On the CPU



Memory management is explicit:
You are responsible for moving data from
host → global → local *and* back

Context and Command-Queues

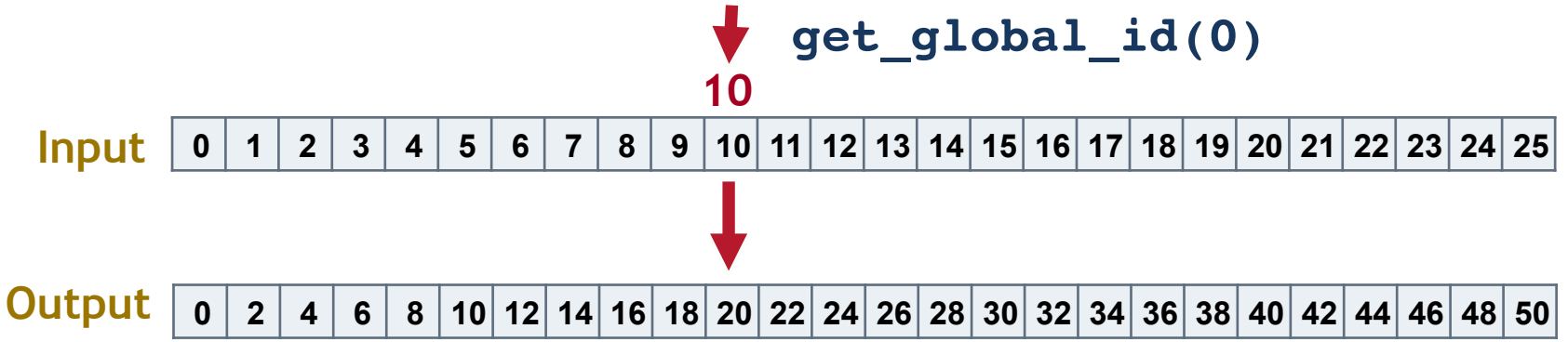
- ***Context***:
  - The environment within which kernels execute and in which synchronization and memory management is defined.
- The ***context*** includes:
  - One or more devices
  - Device memory
  - One or more command-queues
- All ***commands*** for a device (kernel execution, synchronization, and memory operations) are submitted through a ***command-queue***.
- Each ***command-queue*** points to a single device within a context.



Execution model (kernels)

- OpenCL execution model … define a problem domain and execute an instance of a **kernel** for each point in the domain

```
__kernel void times_two(
    __global float* input,
    __global float* output)
{
   int i = get_global_id(0);
   output[i] = 2.0f * input[i];
}
```



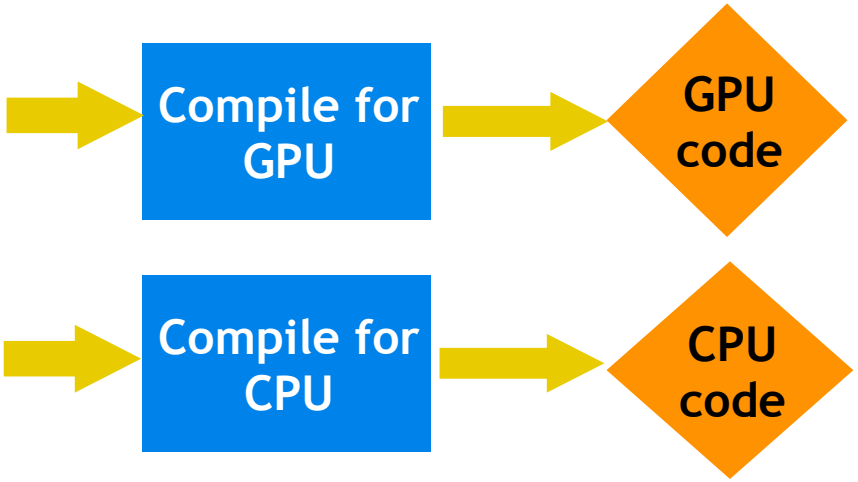
Building Program Objects

- The program object encapsulates:
  - A context
  - The program source or binary, and
  - List of target devices and build options
- The build process to create a program object:

OpenCL uses **runtime compilation** … because in general you don't know the details of the target device when you ship the program

```
cl::Program program(context, KernelSource, true);
```

```
__kernel void
horizontal_reflect(read_only image2d_t src,
                   write_only image2d_t dst)
{
  int x = get_global_id(0);  // x-coord
  int y = get_global_id(1);  // y-coord
  int width = get_image_width(src);
  float4 src_val = read_imagef(src, sampler,
                               (int2)(width-1-x, y));
  write_imagef(dst, (int2)(x, y), src_val);
}
```



Example: vector addition

- The “hello world” program of data parallel programming is a program to add two vectors

```
C[i] = A[i] + B[i] for i=0 to N-1
```

- For the OpenCL solution, there are two parts
  - Kernel code
  - Host code

Vector Addition - Kernel

```
__kernel void vadd(
            __global const float *a,
            __global const float *b,
            __global       float *c)
{
    int gid = get_global_id(0);
    c[gid]  = a[gid] + b[gid];
}
```

Exercise 1: Running the Vector Add kernel

- Goal:
  - To inspect and verify that you can run an OpenCL kernel
- Procedure:
  - Take the Vadd program we provide you. It will run a simple kernel to add two vectors together.
  - Look at the host code and identify the API calls in the host code. Compare them against the API descriptions on the OpenCL C++ reference card.
- Expected output:
  - A message verifying that the program completed successfully

1. ssh -X train#@carver.nersc.gov (and enter supplied password)
2. ssh -X dirac# (and enter supplied password)
3. cp -r /projects/projectdirs/training/SC14/OpenCL_exercises/ .
4. module unload pgi openmpi cuda
5. module load gcc-sl6
6. module load openmpi-gcc-sl6
7. module load cuda
8. cd OpenCL_exercises
9. cp Make_def_files/dirac_linux_general.def make.def
8. cd /Exercises/Exercise01
9. make; ./vadd (etc)
More: https://www.nersc.gov/users/computational-systems/testbeds/dirac/opencl-tutorial-on-dirac/