IDIAP 2015-03-17

OpenCL

Crash Course

frederic.dubouchet@idiap.ch

Sources + slides

http://github.com/anirul/OpenCL_Crash_Course.git

- Known to work on Linux/OSX:
  - a C++ compiler (g++/clang++)
  - the OpenCL Header + libs
  - cmake / OpenCV / Boost / GL / GLU / glut

Plan

- General overview (GPGPU -> OpenCL)
- Code dive (various examples)
- Conclusion (Optimisation tips)

General Overview

- GPGPU - Technology overview
- OpenCL
  - Language and API
  - Device Model
  - Objects

GPGPU

General Purpose GPU

- Using Graphical Processing Unit to compute.
- Shader languages (GLSL / DirectX)
- CUDA (Nvidia proprietary)
- DirectCompute (Windows)
- OpenACC (no free compiler support yet)

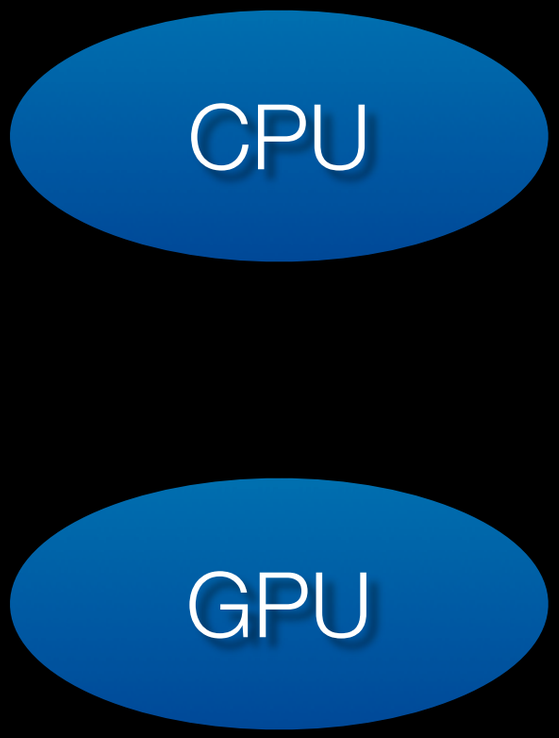
OpenCL

heterogeneous computing platforms

- Khronos (OpenGL, Vulkan, COLLADA, etc…)
  - Intel, QUALCOMM, AMD, Altera Corporation, Vivante Corporation, Xilinx, Inc., MediaTek Inc, ARM Limited, Imagination Technologies, Apple, Inc., STMicroelectronics International NV, ARM, IBM Corporation, Creative Labs, NVIDIA, Samsung Electronics.
- Work on CPU / GPU / DSP / FPGA …
- Open Standard

OpenCL

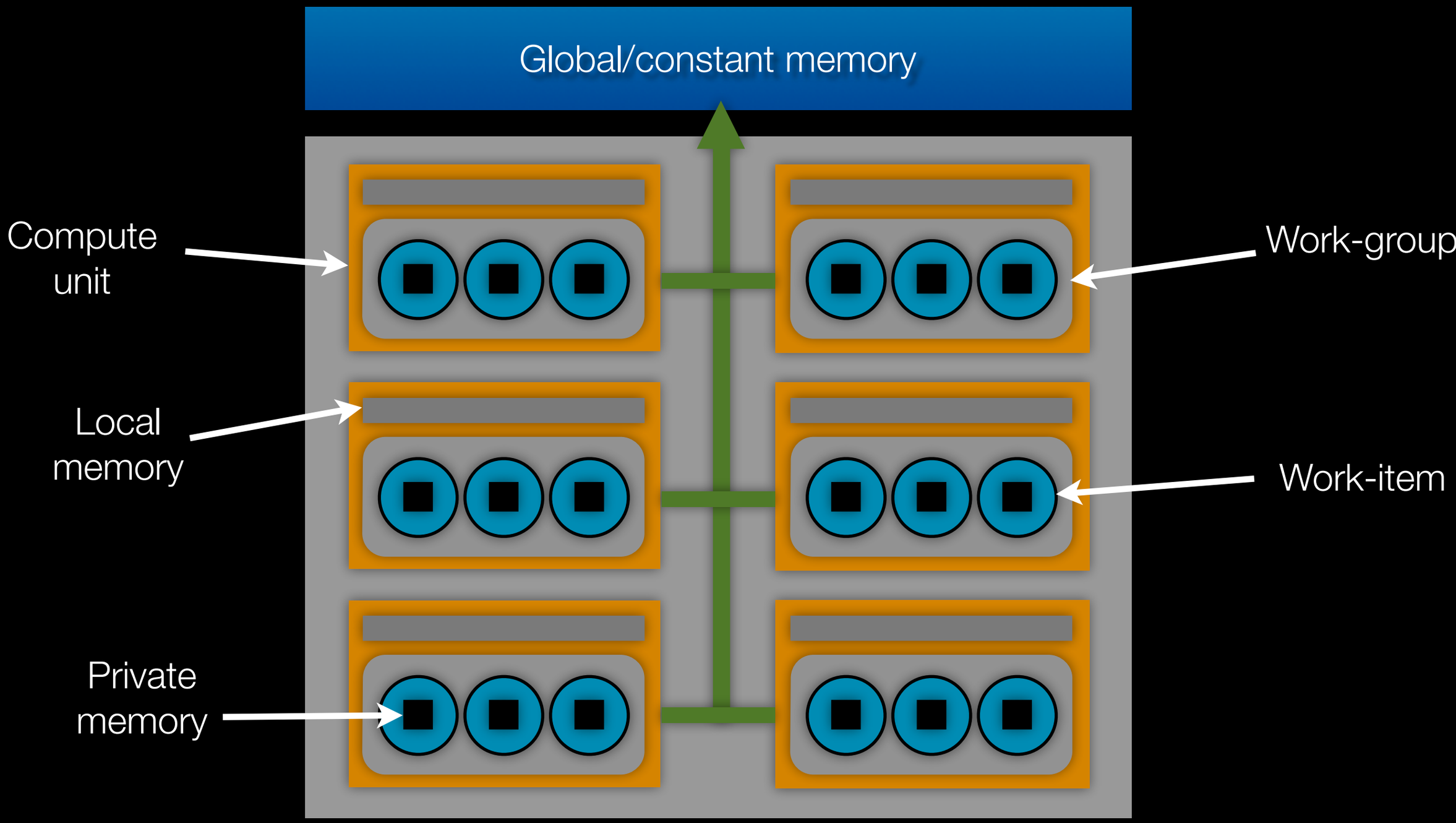
Language and API



- An API (run on the Host)
  - in C but with interface to many other languages
- A language (run on the Device)
  - vector oriented
  - C99 inspired (2.1 -> C++14)

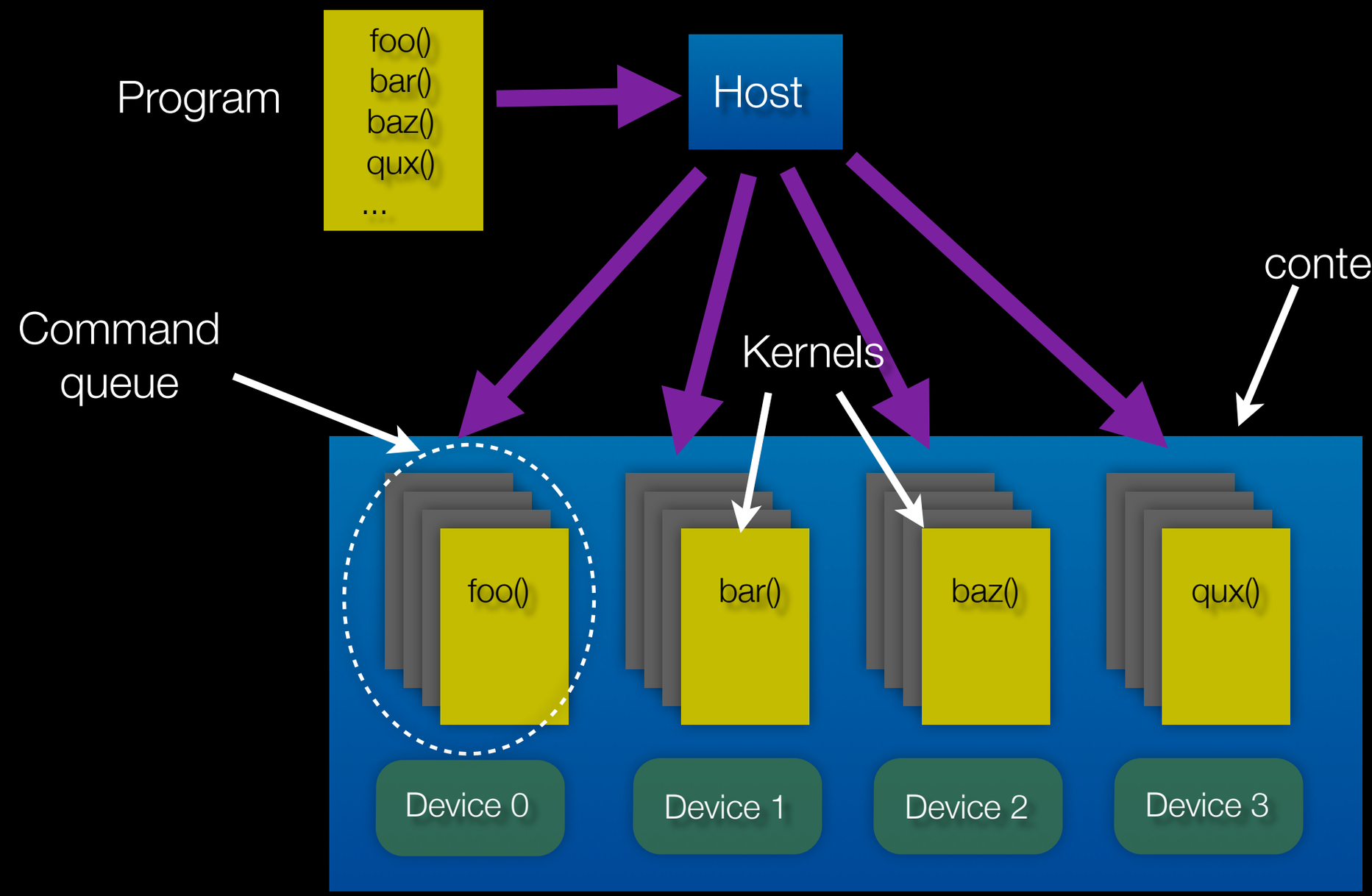
OpenCL

Device Architecture



OpenCL

Objects

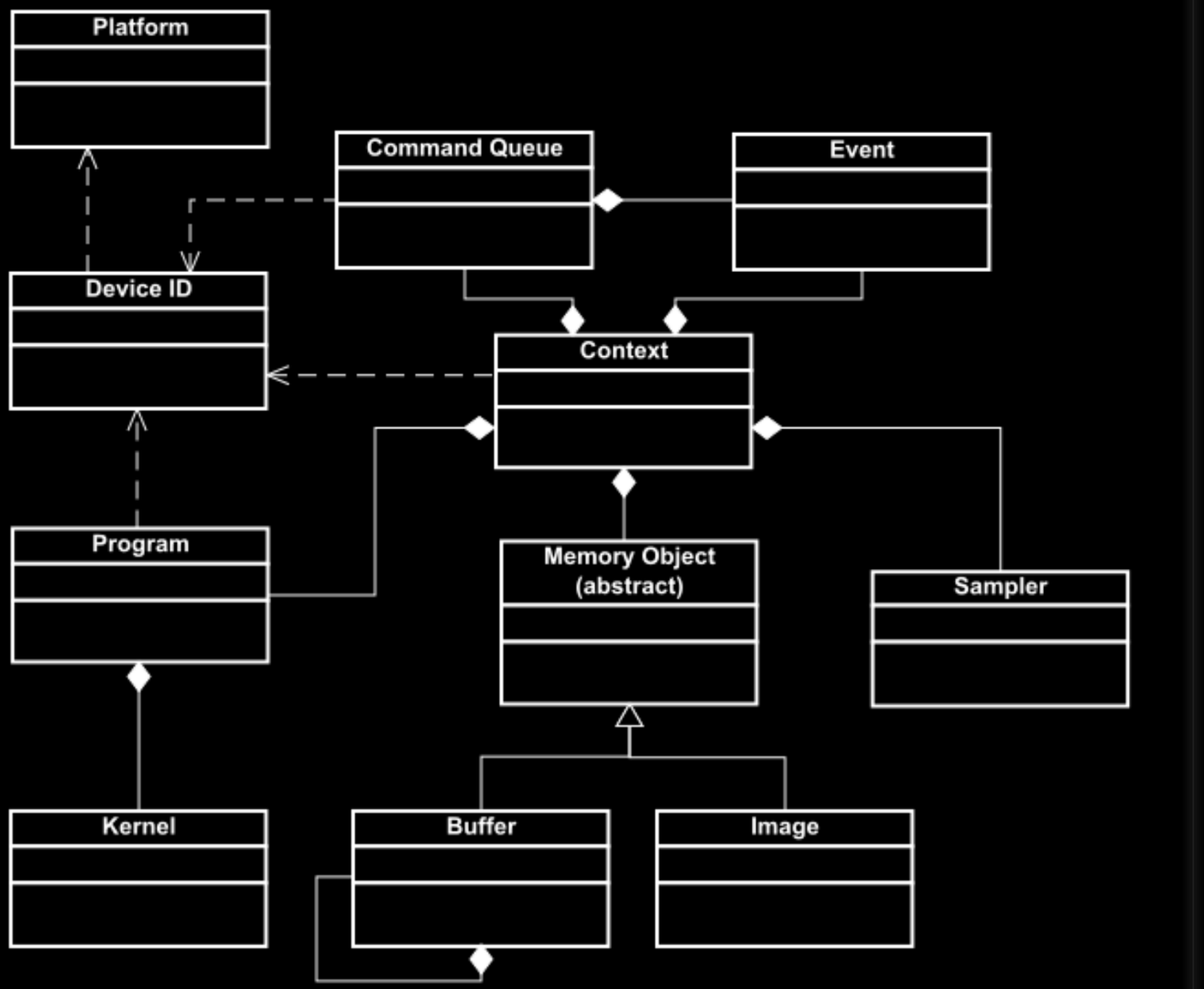


Code Dive

- Khronos C++ wrapper
- Simple - (as it can be!)
- Floyd-Warshall - (memory coalescing)
- Histogram (reduce, local memory)
- Video (the full stack)

Khronos C++ wrapper

- Officially published by Khronos
- Template based
- Header only
- 100% portable
- Object Oriented



Simple

- Very simple example using the C++ wrapper
- Compute the product of 2 vectors
- Single file only OpenCL dependent
  - simple.cpp
- Not a practical example!

Device code

```
// very simple kernel

kernel void simple(
     global read_only float* in1,
     global read_only float* in2,
     global write_only float* out)
{
   const uint pos = get_global_id(0);
   out[pos] = in1[pos] * in2[pos];
}
```

- Simple product of two vectors

Host code (1)

```
#define __CL_ENABLE_EXCEPTIONS
#include <CL/cl.hpp>
```

- Add exception support to OpenCL

```
std::vector<cl::Platform> platforms;
cl::Platform::get(&platforms);
std::vector<cl::Device> devices_;
platforms[platform_id].getDevices(
    CL_DEVICE_TYPE_ALL,
    &devices_);
```

- Get Platform and device

Host code (2)

```
cl_context_properties properties[] = {
    CL_CONTEXT_PLATFORM,
    (cl_context_properties)(platforms[platform_id])(),
    0
};
cl::Context context_ = cl::Context(CL_DEVICE_TYPE_ALL, properties);
cl::CommandQueue queue_(context_, devices_[device_id], 0 , nullptr);
```

- Generate a context

```
cl::Program::Sources source(
    1,
    std::make_pair(
        kernel_source.c_str(),
        kernel_source.size()));
cl::Program program_(context_, source);
program_.build(devices_);
cl::Kernel kernel_(program_, "simple");
```

- Get the source build and select a kernel

Host code (3)

```
cl::Buffer buf_in1_ = cl::Buffer(
    context_,
    CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
    sizeof(cl_float) * in1.size(),
    (void*)&in1[0]);
```

- Create a buffer from a STL vector

```
kernel_.setArg(0, buf_in1_);
kernel_.setArg(1, buf_in2_);
kernel_.setArg(2, buf_out_);
```

- Set the arguments of the kernel

Host code (4)

```
queue_.enqueueNDRangeKernel(
    kernel_,
    cl::NullRange,
    cl::NDRange(vector_size),
    cl::NullRange);
```

- Execute the kernel

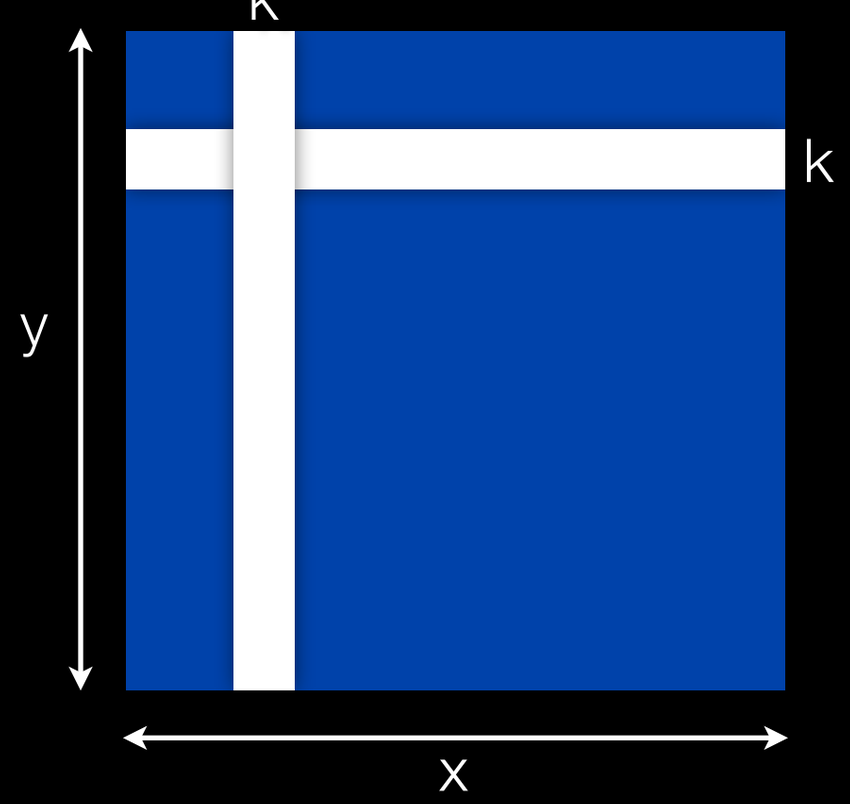
```
queue_.enqueueReadBuffer(
    buf_out_,
    CL_TRUE,
    0,
    vector_size * sizeof(float),
    &out[0]);
```

- Get the result

Floyd Warshall

For each values of k (in 0..N -1)

The Matrix

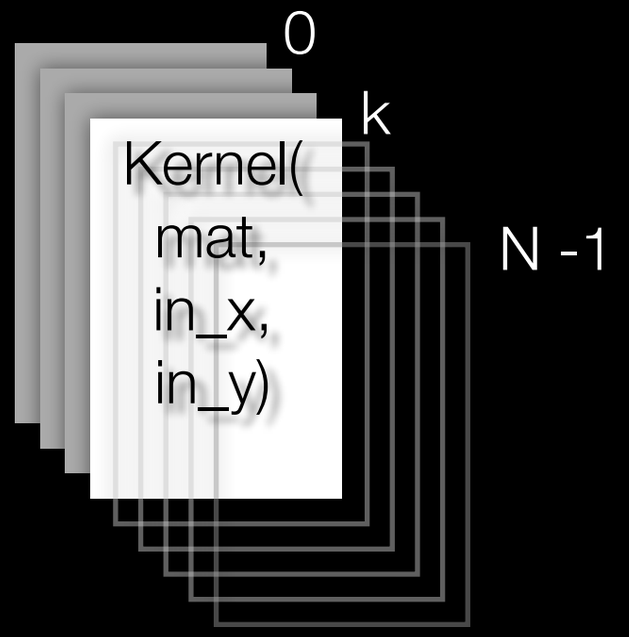


mat (whole)

in_x = y[k]

in_y = x[k]

The Kernel Stack



Conclusion

- Third part (conclusion)
- Optimisation tips
- Questions?