Kalray OpenCL 1.2
Training

Scope

- Introduction to the Kalray OpenCL support for the Kalray MPPA architecture
- Practical exercises using the Kalray OpenCL API

OpenCL

- Open standard for parallel programming of heterogeneous systems : https://www.khronos.org/opencl/

Overview

- Kalray OpenCL for MPPA devices is a subset of the OpenCL 1.2 framework
- MPPA device is an accelerator device
- Both host runtime and device support
- Kalray OpenCL programs can be executed on both simulator and hardware

The Kalray OpenCL Model

- Platform Model :
  - 16 compute units of 16 processing elements
    - 1 MPPA board is 1 compute device
    - Work-groups executed on compute units
    - Work-items executed on processing elements

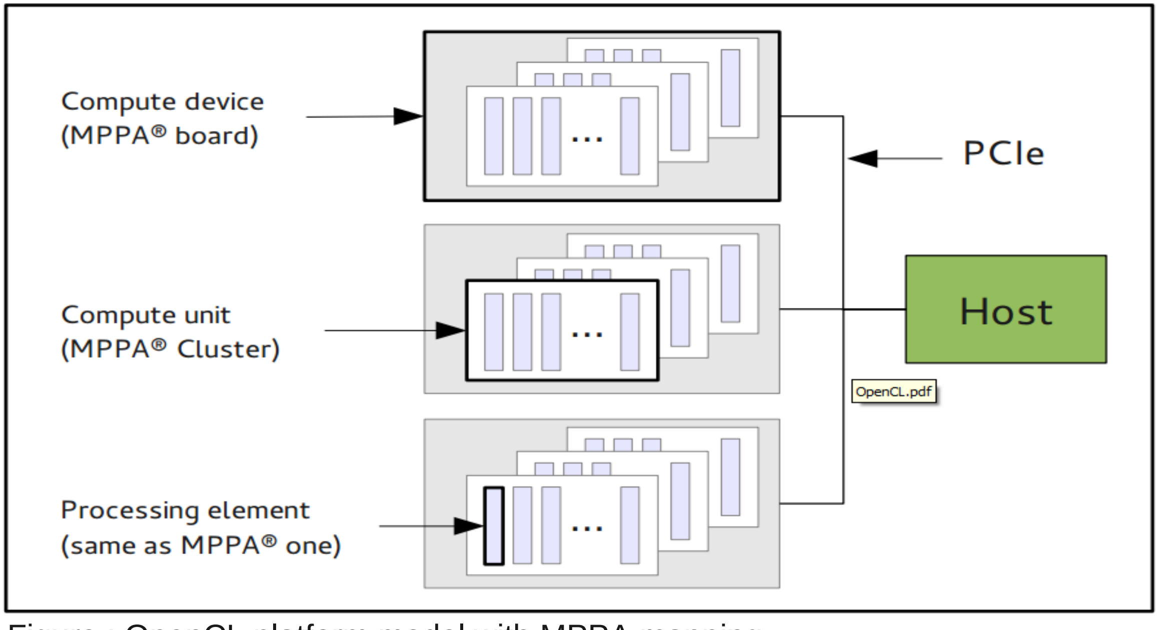


Figure : OpenCL platform model with MPPA mapping

The Kalray OpenCL Model

- Memory Model :
  - Private data (__private) on PE's stack
  - Local data (__local) in buffer shared between Pes
  - Global data (__global) in DDR memory and accessed using the DSM system

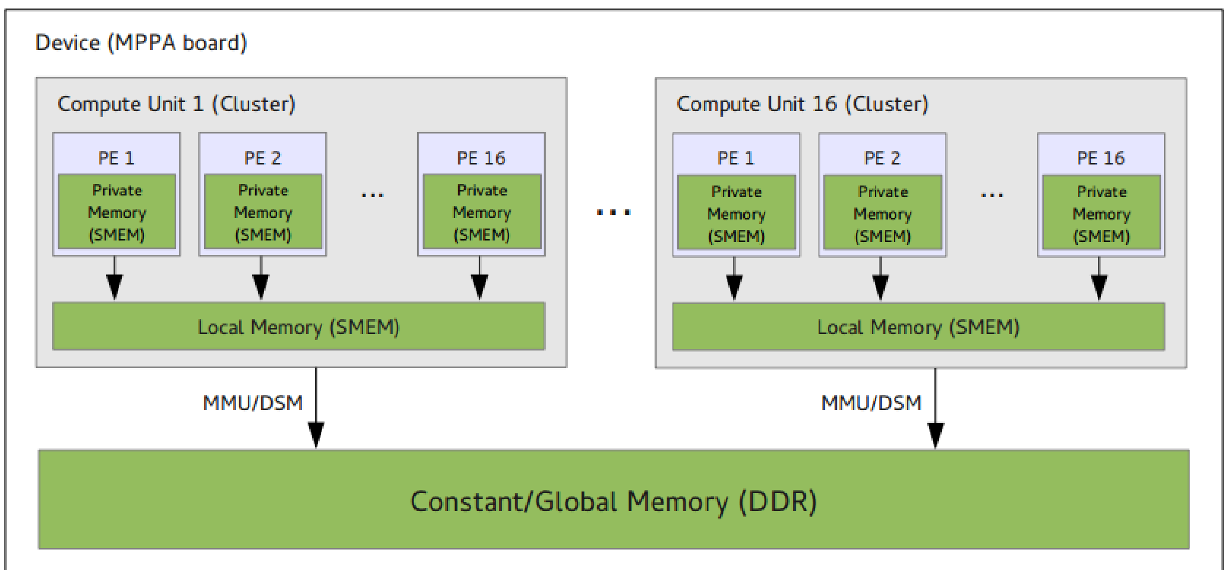


Figure : OpenCL memory model for the MPPA architecture

The Kalray OpenCL Implementation

- OpenCL Host support :
  - Based on the POCL project
  - Functionalities supported by the Kalray OpenCL API :
    - Buffers and sub-buffers
    - Out-of-Order execution for command queues
    - Events, wait lists
    - NDRange, Task and Native kernels
    - C++ OpenCL API Wrapper
  - Kernels have to be written in plain C (C11)

The Kalray OpenCL Implementation

- Memory space keywords :
  - __local, __global, and __constant are supported if placed in the kernel parameters declaration and ignored otherwise
  - __global buffers are allocated in the DDR. This memory is currently limited to 1024MB
  - __constant is identical to __global
  - __local buffers are allocated in the shared memory (SMEM) with a total maximum of 128kB
  - __private is not yet supported. However, any variable declared on the stack (ie in the body of a function) will have private visibility for the work-item. The maximum amount of private memory (ie stack size) is 4kB

Memory Space Keywords : An Example

```
__kernel void my_kernel(__global void *arg1, /* Allowed */
                           __local void *arg2) /* Allowed */
{
       __local int var1[10]; /* Ignored,
                              * visibility defaults to private */
       __private int var2;   /* Ignored,
                              * visibility defaults to private */
       char var3[4*1024];    /* Harmful, Stack overflow
                              * Stack size is only 4K */
}
```

The Kalray OpenCL Implementation

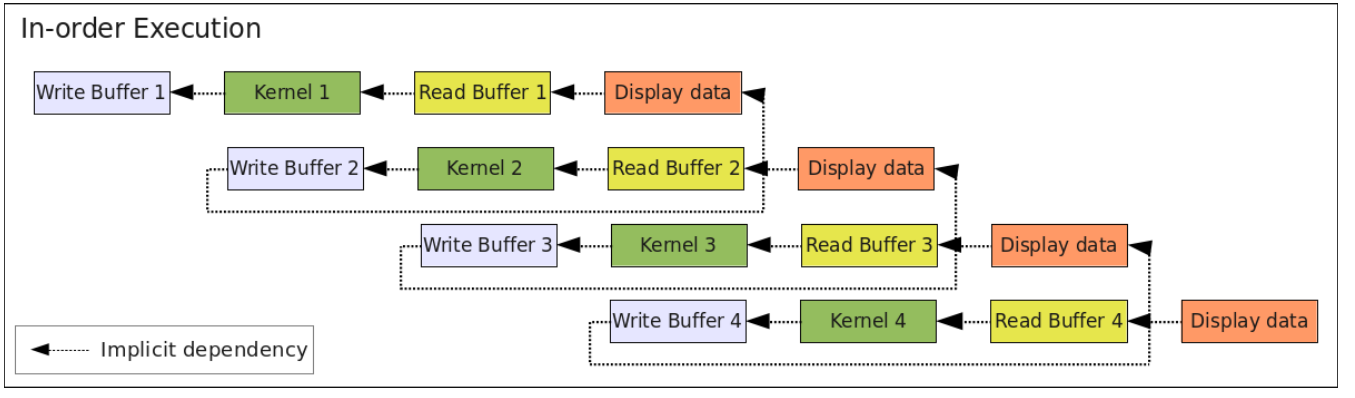
- Memory accesses :
  - Distributed Shared Memory (DSM) system is used
  - When a buffer is accessed, a page (8kB by default) of data is fetched from the DDR and stored in the SMEM
  - Each PE can store roughly 10 pages in the SMEM

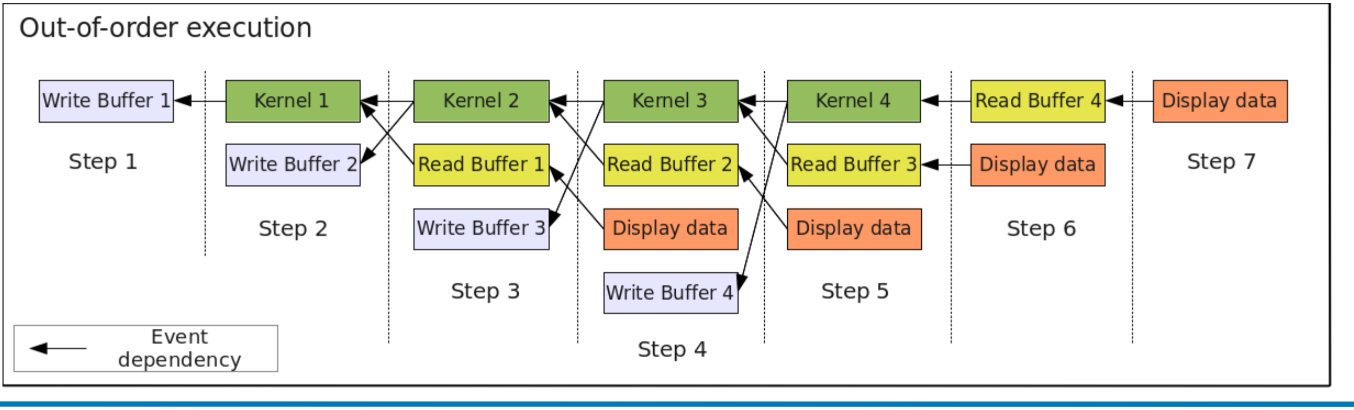
The Kalray OpenCL Implementation

- Tips to harness the full performance of the MPPA :
  - Better if each work-item accesses different pages
  - Use the OpenCL builtin `prefetch` to speedup the fetch of a page of data
  - Enable Out-of-Order (OoO) execution
  - Run multiple kernels concurrently by enabling OoO execution and by carefully sizing work-group dimensions

Out-of-Order Execution

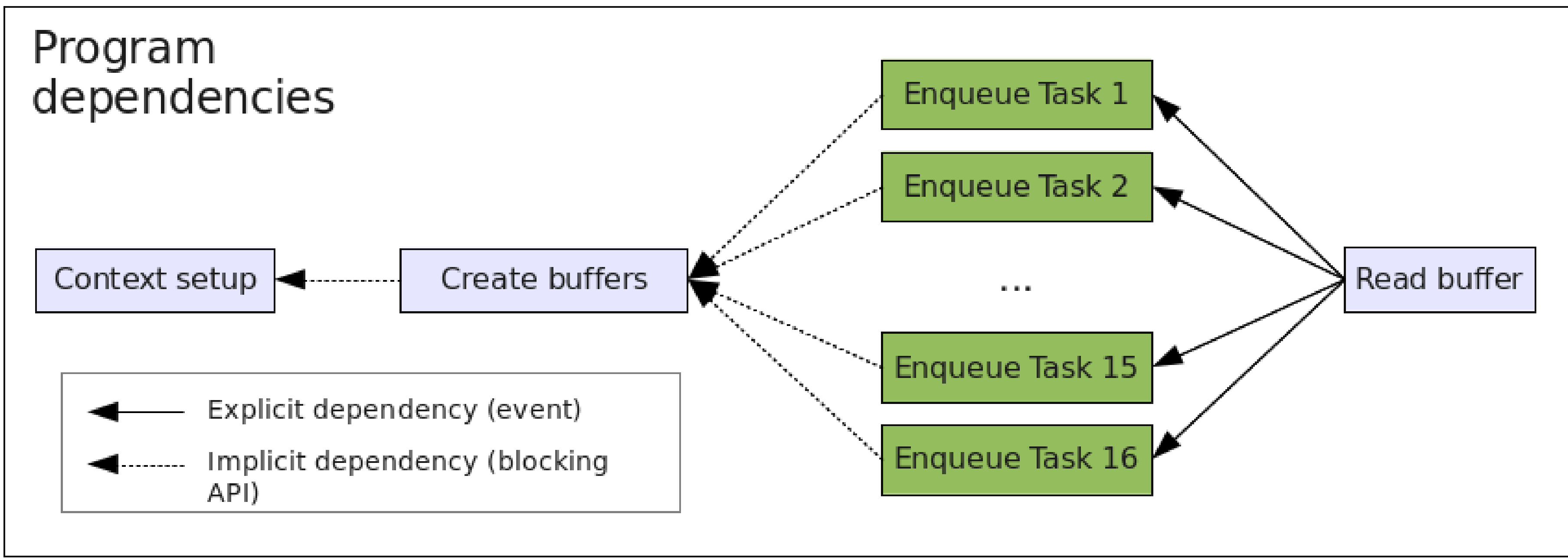
- Multiple commands can be executed simultaneously by using OpenCL events to describe dependencies





Task Kernel Support

- To achieve task parallelism with several kernels
- Task kernels are run on only 1 PE
- Run kernels concurrently by enabling out-of-order execution so that they run on each cluster



NDRange Kernel Support

- Up to 3 dimensions for NDRange kernels supported
- The number of work-groups is practically unlimited
  - If using more than 16 work-groups, the first dimension will be dispatched over the 16 compute units
  - Eg : if work-group dimension is (gx, gy, gz) the compute unit **n** will execute the range (gx * **n**/16, 0, 0) to ((gx/16) * (**n**+1) - 1, gy-1, gz-1)
  - The 1st dimension should be a multiple of 16, while the 2nd and 3rd dimensions have no constraints
- Work items are limited to 16 per work-group

The OpenCL-C Language

- OpenCL-C not yet supported
  - A subset is available
    - Some builtins
    - Partially supported OpenCL-C types
- Subset of ISO C99 :
  - No function pointers
  - No variable length arrays
  - No recursion
- Superset of ISO C99 :
  - Work-items and work-groups (get_global_id(), etc)
  - Vector support
  - Synchronisation
  - Address space qualifiers

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

```
__kernel void opencl_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
```

The OpenCL Kernel Program

- Written inlined within the host application code OR
- Separately in a .cl file and loaded at runtime from the host application
- The keyword __kernel is used to declare a kernel

Developing OpenCL Programs

- Set-up OpenCL on the host :
  - #include <CL/cl_kalray.h>
  - Create a device *context*
    - Specifies the OpenCL environment (GPU(s), MPPA(s), ...)
  - Create a *program*
    - Source compiled for specific *context* during execution of Host application
  - Create a *kernel*
    - Provides an access point from the Host to the OpenCL functions in the *program*
  - Create OpenCL *buffers*
    - Allocation of *global memory*
- Program the body of the application :
  - Copy data between Host and global memory
  - Execute kernels on work-items
- Exit sequence :
  - Release buffers, kernel, program and context

OpenCL Objects

- cl_platform_id
- cl_device_id
- cl_context
- cl_command_queue
- cl_program
- cl_kernel
- cl_mem
- cl_event

Synchronization

- Work items in a single work group
  - Barrier (encountered by all work-items in the work-group)
- Commands enqueued to command-queues in a single context
  - Command-queue barrier
  - Waiting on an event

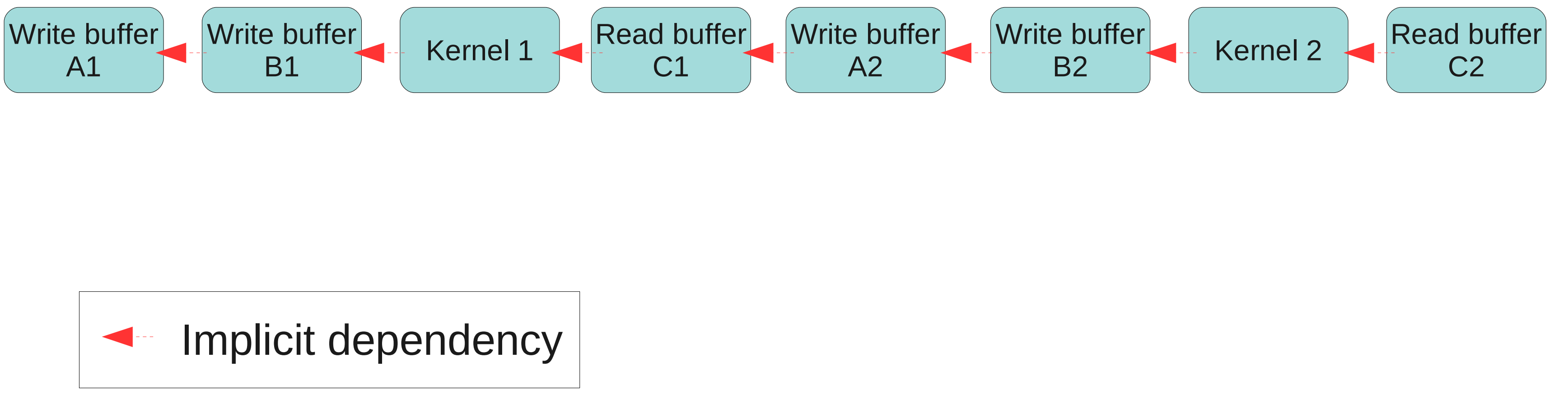
How To Compile and Run

- Use Kalray Makefile (`Makefile.opencl`)
  - `include $K1_TOOLCHAIN_DIR)/share/make/Makefile.opencl`
  - `host_ndrange-srcs`
- To run using simulator :
  - `k1-pciesim-runner <ocl_exec>`
- To run on the HW :
  - `<ocl_exec>`
- To trace :
  - `k1-opencl-trace -- <ocl_exec>`

Example 1 : vector_add

- Source code : vector_add
- Exercise :
  - Compile and run the code (make run_hw target)
  - Run the make run_hw_trace target to trace the application
  - change the parameters for the clEnqueueNDRangeKernel function to only use 1 compute-unit (1 cluster)

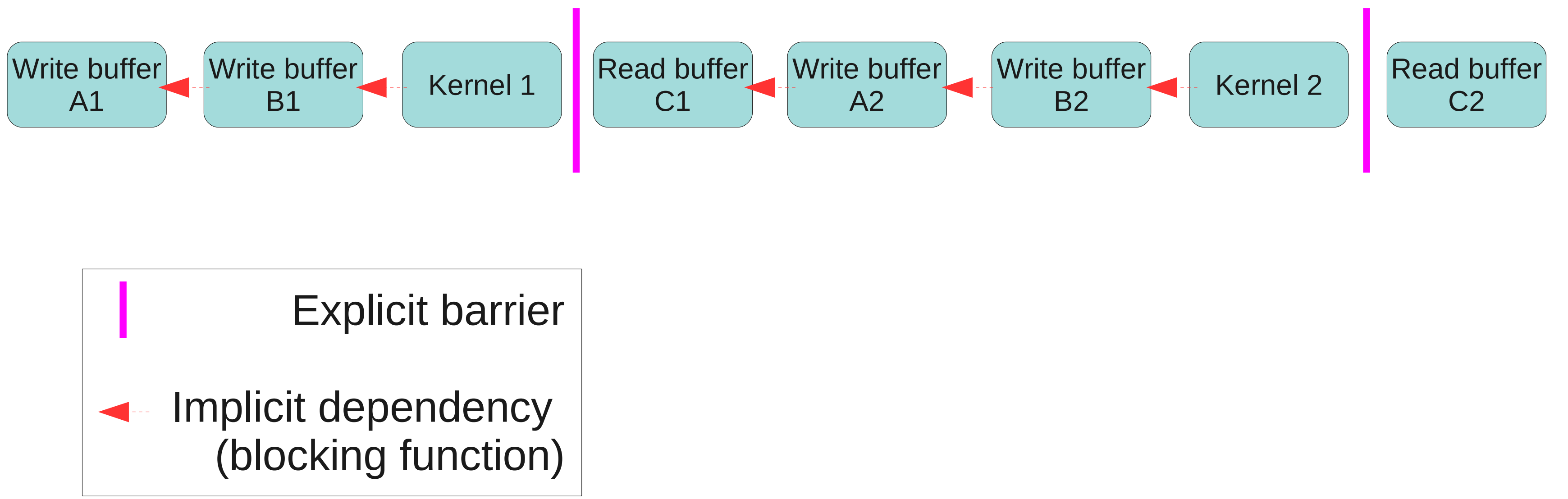
Two Kernels – In-order execution



Example 2 : 2 Kernels in-order

- Exercise :
  - Add code to call the kernel again. Make sure you set the input buffer to the output of the previous kernel
  - Set the command queue parameter to run the kernels in-order (i.e. Do not change the default)

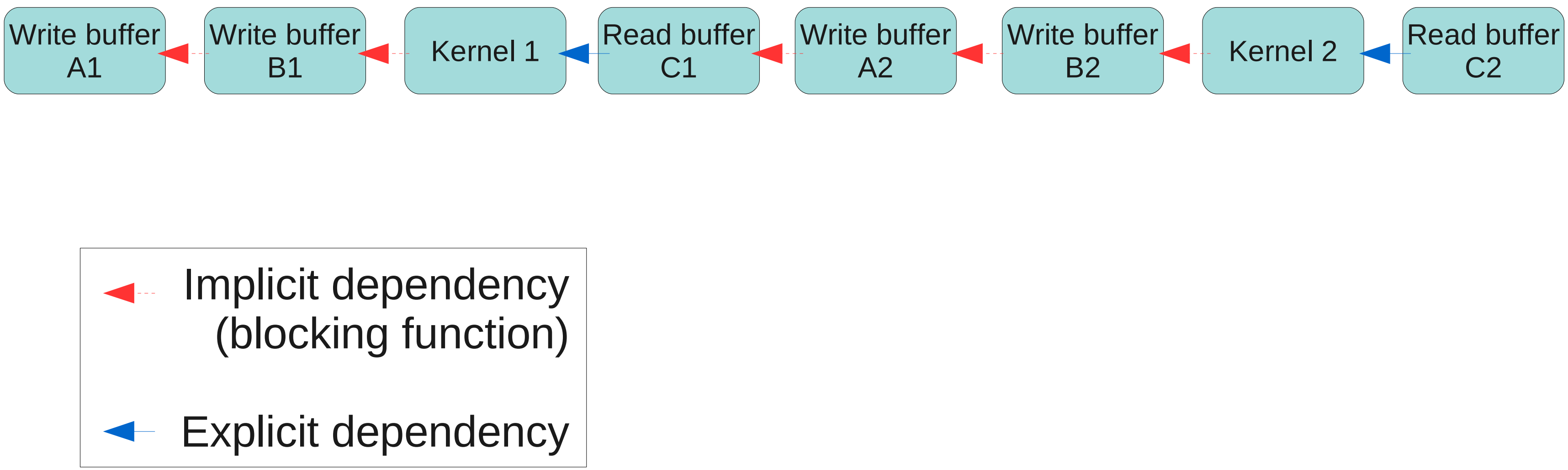
Two Kernels – Out-of-order execution using `clFinish()`



Example 3 : 2 Kernels out-of-order

- Exercise :
  - Set the command queue parameter to run the kernels out-of-order
  - Fix the bug that appears

Two Kernels – Out-of-order execution using events



Example 4 : 2 Kernels out-of-order

- Exercise :
  - Set the command queue parameter to run the kernels out-of-order using events this time