SYCL Academy

FURTHER OPTIMIZATIONS

LEARNING OBJECTIVES

- Learn about compute and memory bound algorithms
- Learn about optimizing for occupancy
- Learn about optimizing for throughput

COMPUTE-BOUND VS MEMORY-BOUND

- When a particular algorithm is applied to a processor such as a GPU it will generally be either compute-bound or memory-bound.
- If an algorithm is compute-bound then the limiting factor is occupancy, utilizing the available hardware resources.
- If an algorithm is memory-bound then the limiting factor is throughput, reducing memory latency.

OPTIMIZING FOR OCCUPANCY

- Occupancy is a metric used to measure how much of the computing power of a processor such as a GPU is being used.
- Optimizing for occupancy means getting the most out of the resources of the GPU.
- For memory-bound algorithms you don't necessarily need 100% occupancy, you simply need enough to hide the latency of global memory access.
- For compute-bound algorithms occupancy becomes more crucial, even then it's not always possible to achieve 100% occupancy.

LIMITING FACTORS

There are a number of factors which can limit occupancy for a given kernel function and the data you pass to it.

- Number of work-items required.
- Amount of private memory (registers) required by each work-item.
- Number of work-items per work-group.
- Amount of local memory required by each work-group.

CHOOSING AN OPTIMAL WORK-GROUP SIZE

An important optimization for occupancy is to choose an optimal work-group size.

LOCAL MEMORY IMAGE CONVOLUTION PERFORMANCE

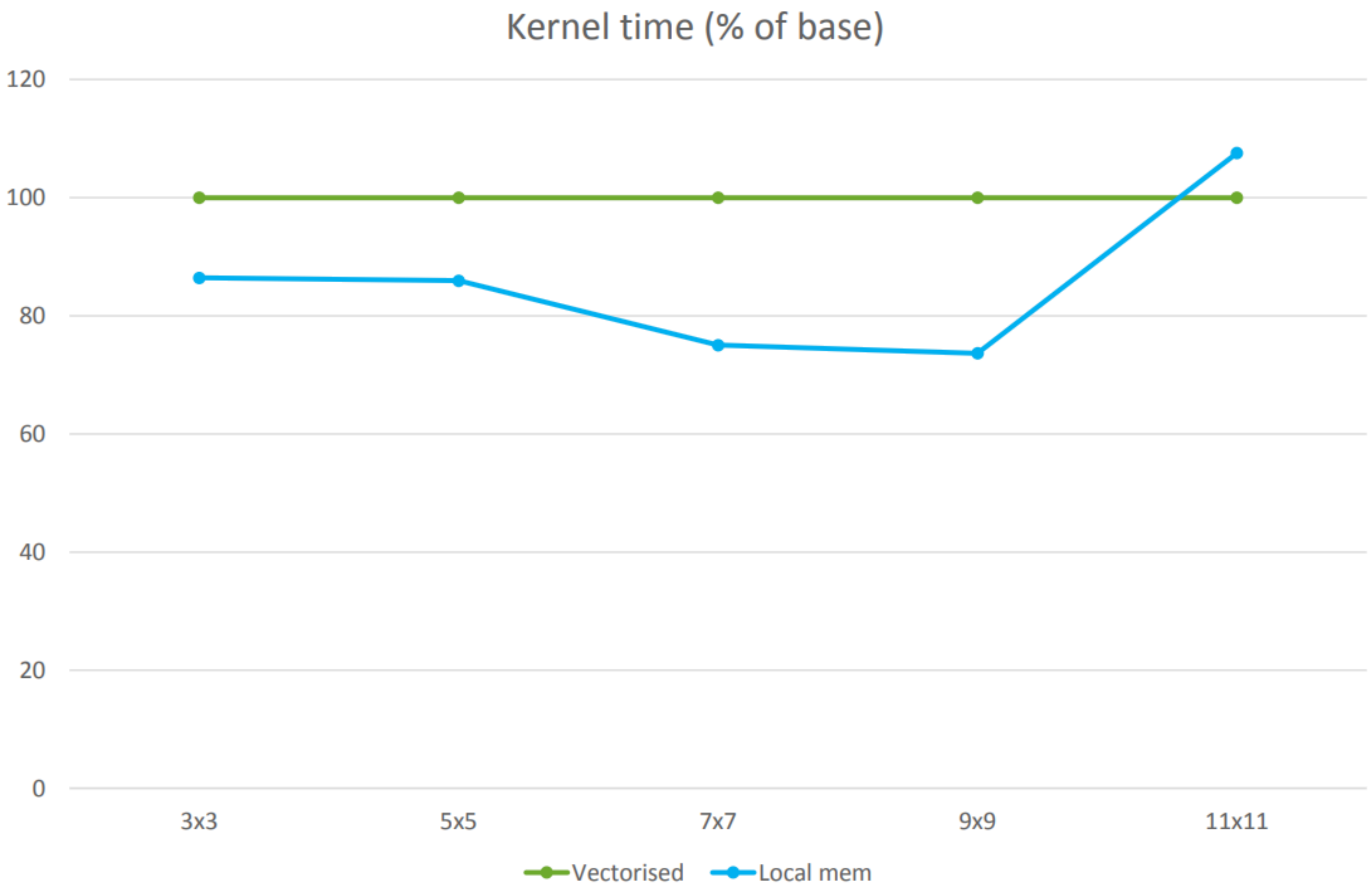


Image conv

512x512 source image

Intel HD Graphics 530

CHOOSING AN OPTIMAL WORK-GROUP SIZE

- It must be smaller than the maximum work-group size for the device you are targeting.
- It must be large enough that you are utilizing the hardware concurrency (warp or wavefront)
- It should be a power of 2 to allow multiple work-groups to fit into a compute unit.
- It's best to experiment with different sizes and benchmark the performance of each.

VARYING WORK-GROUP SIZE IMAGE CONVOLUTION PERFORMANCE

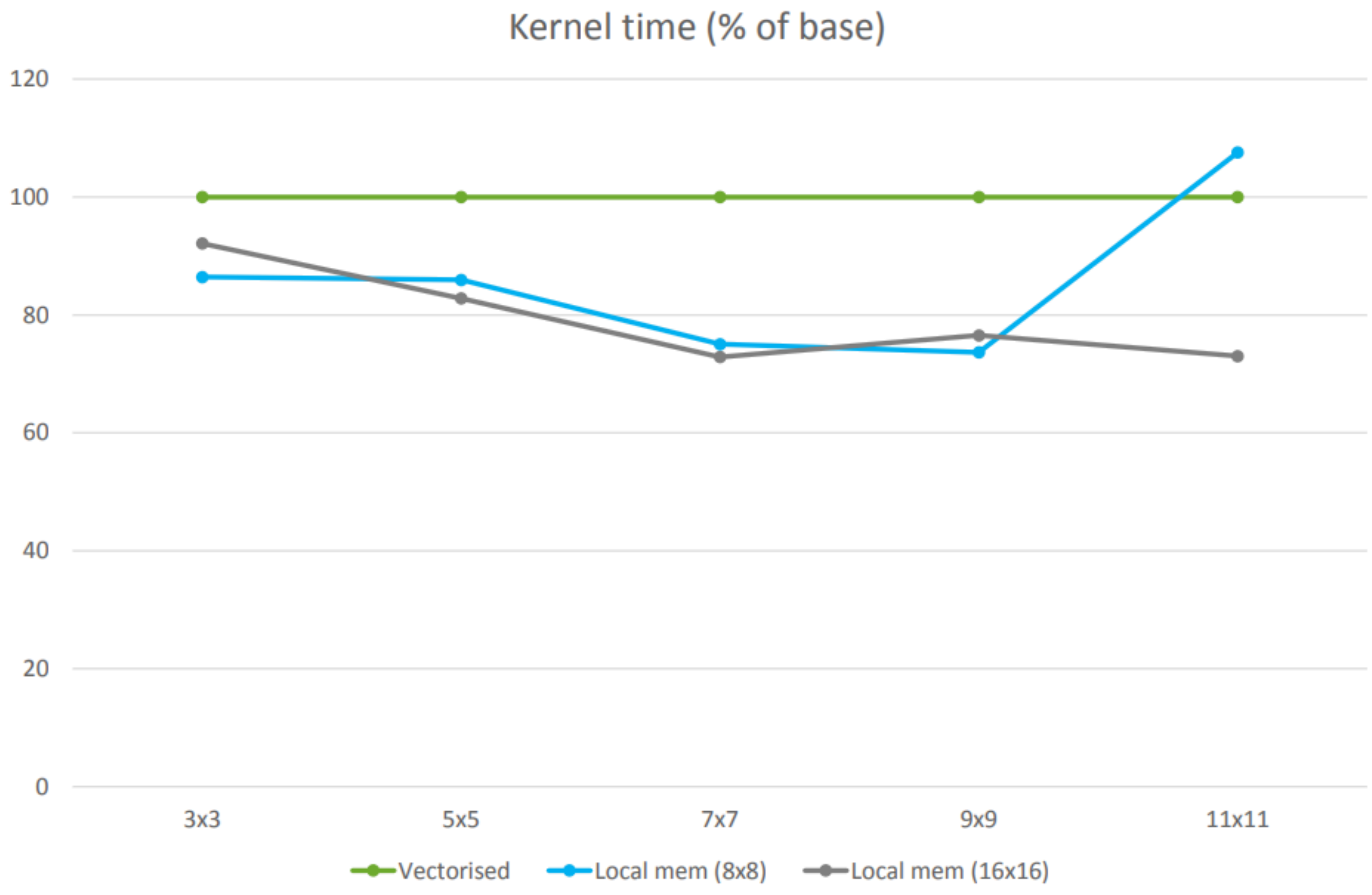


Image conv

512x512 source image

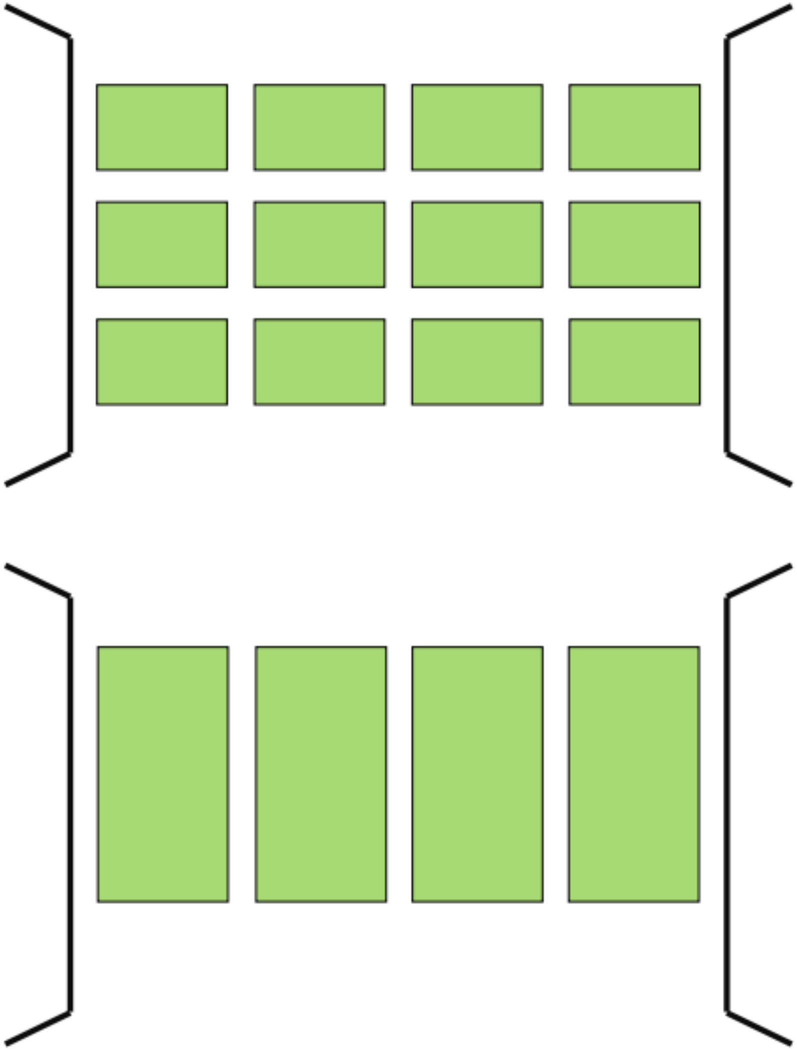
Intel HD Graphics 530

REUSING DATA

Another important optimization for occupancy is to re-use data as much as possible.

- Larger work-group sizes often mean that more work-items can share partial results and overall less memory operations are required.
- Though you can hit limitations in the number of work-items required or the amount of local memory required.
- If you hit this limitation then you can perform work in batches.

WORK BATCHING



- If you hit occupancy limitations then each compute unit must process multiple rounds of work-groups and work-items.
- This often means reading and writing often the same data again.
- Batching work for each work-item that share neighboring data allows you to further share local memory and registers.

OPTIMIZING FOR THROUGHPUT

- Throughput is a metric used to measure how much data is being passed through the GPU.
- Optimizing for throughput means getting the most out of the memory bandwidth at the various levels of the memory hierarchy.
- The key is to hide the latency of the memory transfers and avoid GPU resources being idle.
- This is most important for memory-bound algorithms.

OPTIMIZING FOR THROUGHPUT

We have seen a number of the techniques to optimize throughput already.

- Coalescing global memory access.
- Using local memory.
- Explicit vectorization.

OPTIMIZING FOR THROUGHPUT

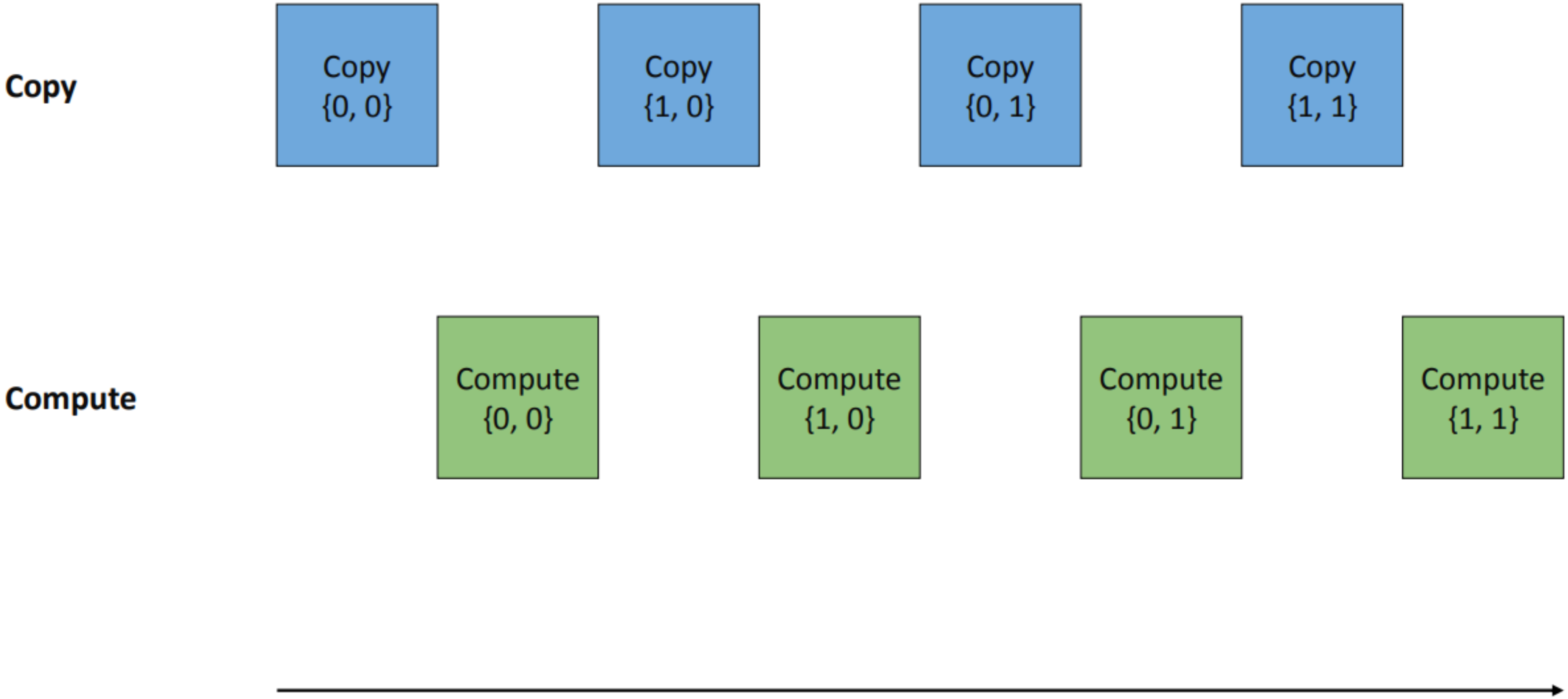
There are a number of other optimization to consider, some depend on the application or the target device.

- Double buffering.
- Using constant memory.
- Using texture memory.

DOUBLE BUFFERING

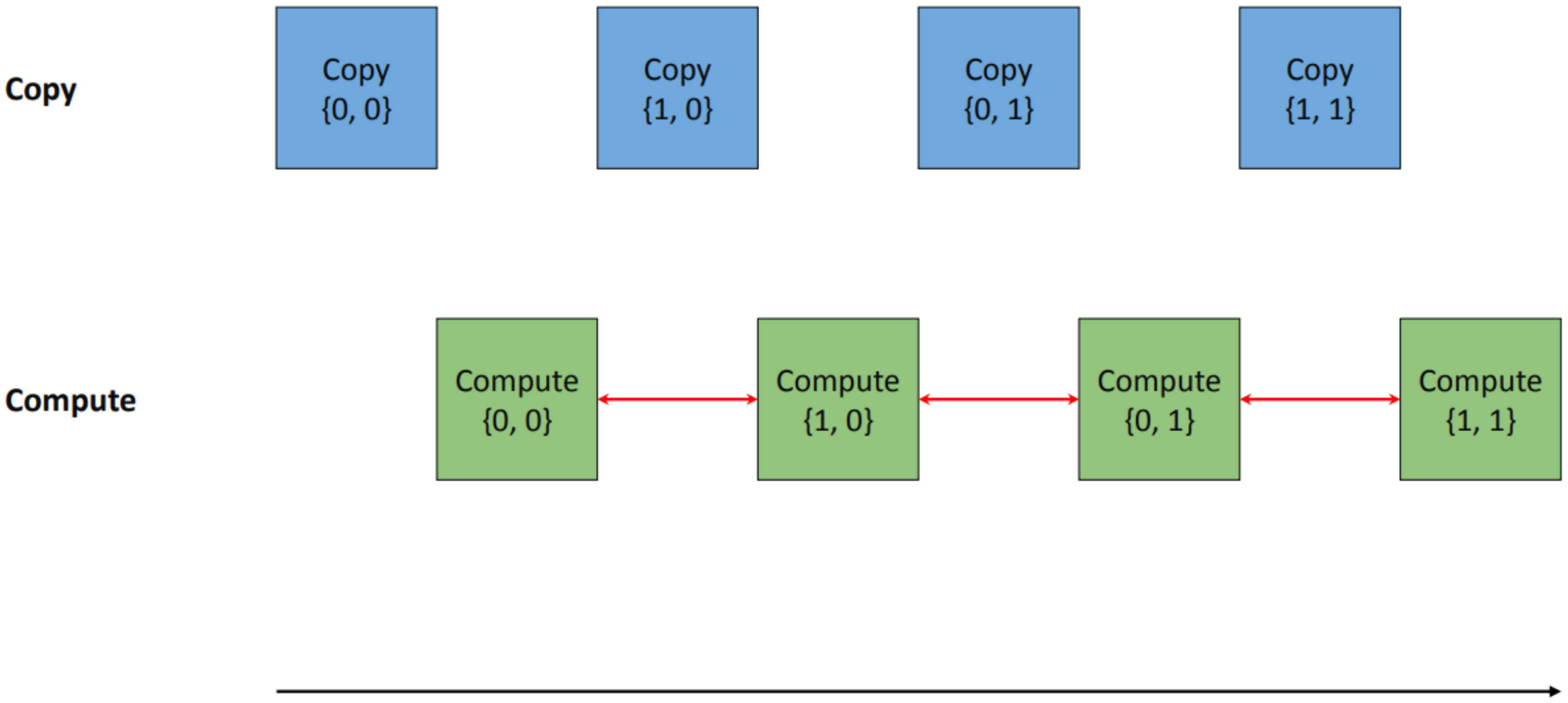
- With particularly large data you can hit global memory limitations.
- When this happens you need to pipeline the work, execute the kernel function itself multiple times, each time operating on a tile of the input data.
- But copying to global memory can be very expensive.
- To maintain performance you need to hide the latency of moving the data to the device.

DOUBLE BUFFERING



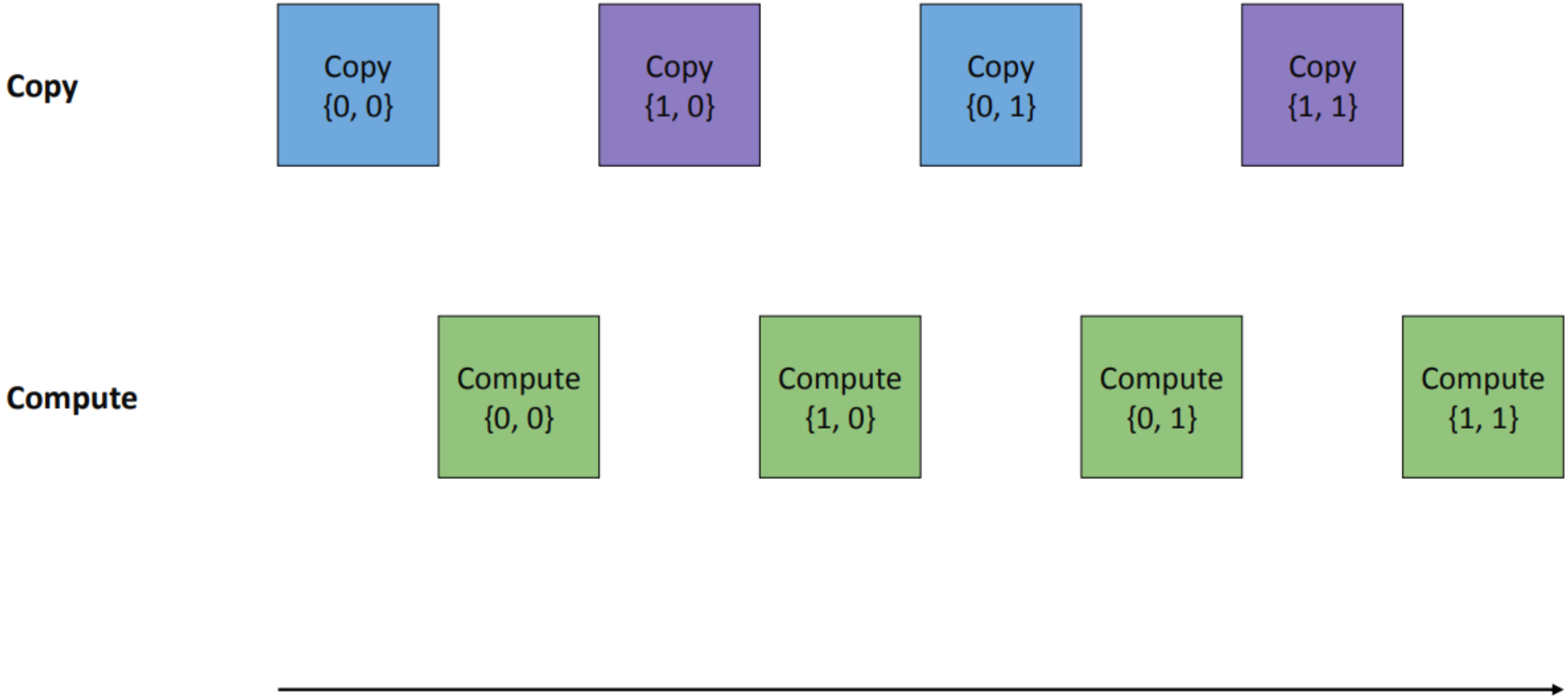
- So in this case you have multiple invocation of the kernel function.
- And between each invocation the data of the next tile must be moved to the GPU

DOUBLE BUFFERING



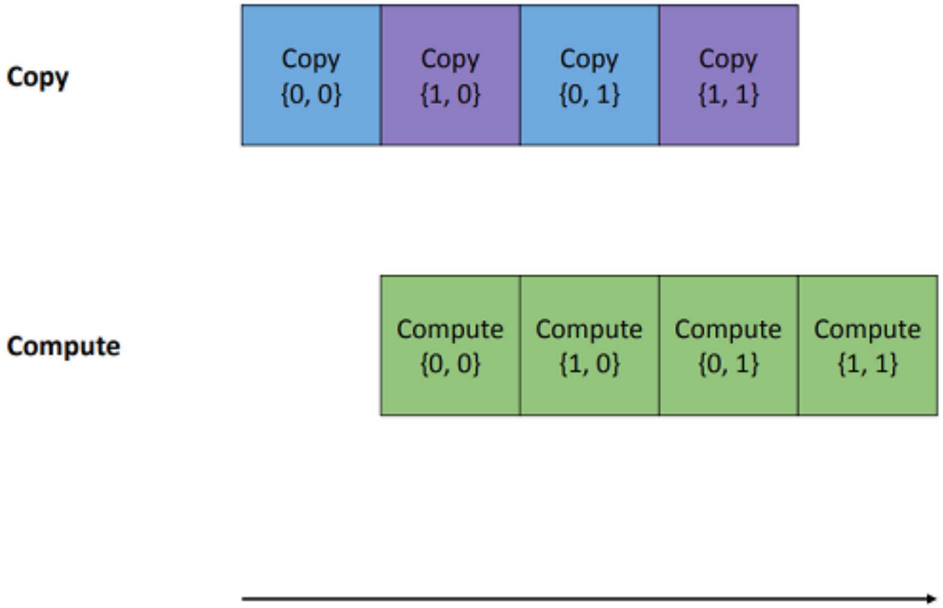
- With the cost of moving data to the GPU being very high so this causes significant gaps where the GPU is idle.

DOUBLE BUFFERING



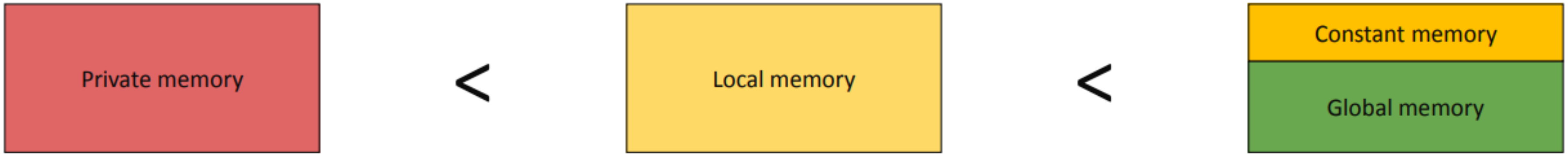
- An option for optimizing this is to double buffer the data being moved to and accessed on the GPU.

DOUBLE BUFFERING



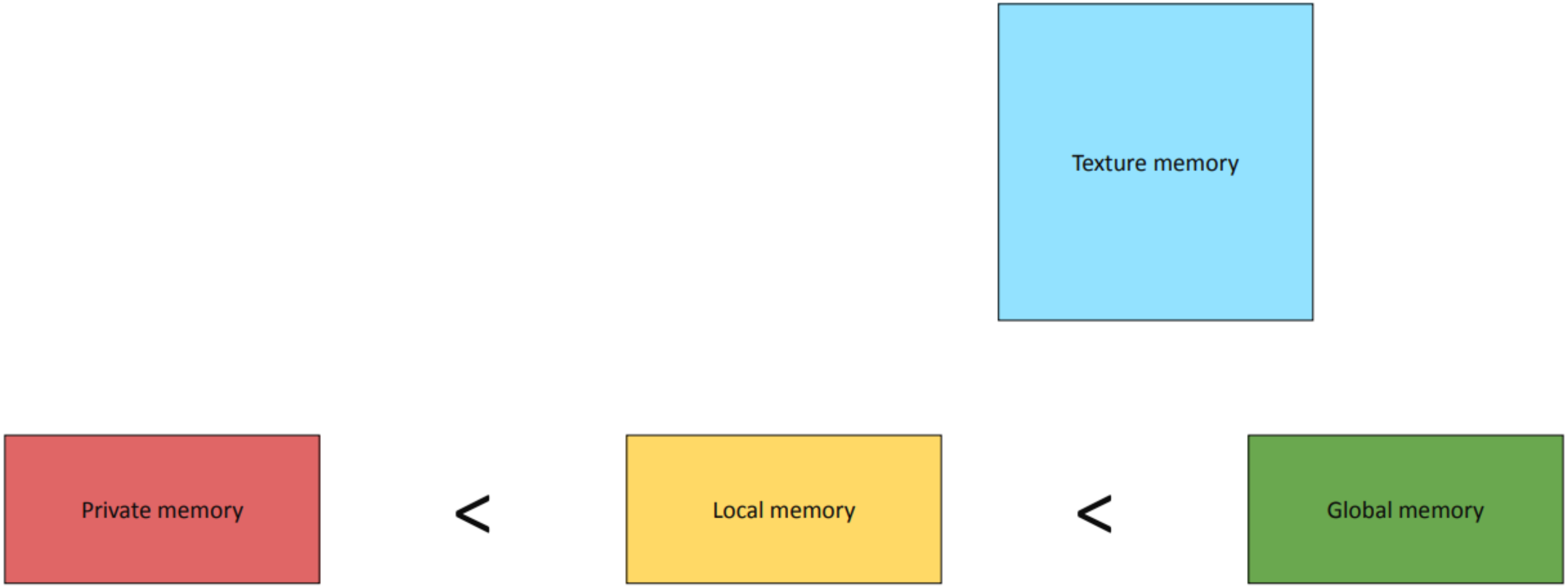
- This allows overlapping of compute and data movement.
- This means that while one kernel invocation computes a tile of data the data for the next tile is being moved.

USE CONSTANT MEMORY



- Some SYCL devices provide a benefit from using constant memory.
- This a dedicated region of global memory that is read-only, and therefore can be faster to access.
- Not all devices will benefit from using constant memory.
- There is generally a much lower limit to what you can allocate in constant memory.
- To use constant memory simply create an `accessor` with the `access::target::constant_buffer` access target.

USE TEXTURE MEMORY



- Some SYCL devices provide a benefit from using constant memory.
- This the texture memory used by the render pipeline.
- This can be more efficient when accessing data in a pixel format.
- To use texture memory use the `image` class.

FURTHER TIPS

- Profile your kernel functions.
- Follow vendor optimization guides.
- Tune your algorithms.

QUESTIONS

EXERCISE

Code_Exercises/Exercise_19_Work_Group_Sizes/source

Try out different work-group sizes and measure the performance.

Try out some of the other optimization techniques we've looked at here.