SYCL Academy

ND RANGE KERNELS

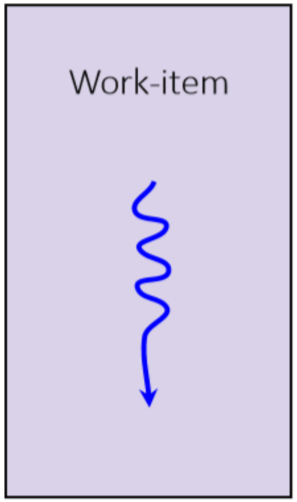
NHR SW

LEARNING OBJECTIVES

- Learn about the SYCL execution and memory model
- Learn how to enqueue an nd-range kernel function

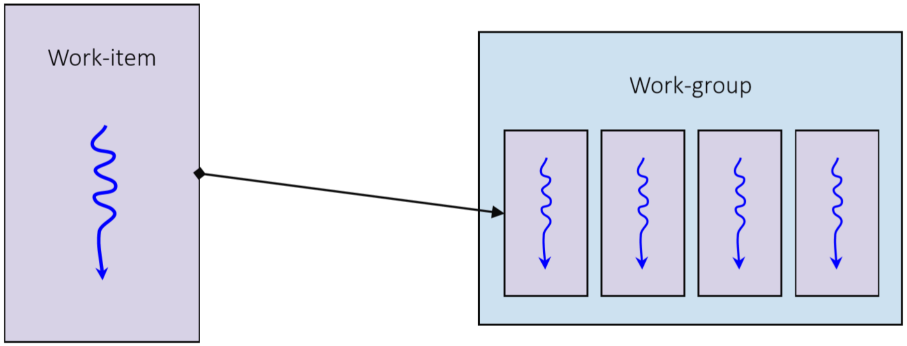
SYCL EXECUTION MODEL

- SYCL kernel functions are executed by **work-items**
- You can think of a work-item as a thread of execution
- Each work-item will execute a SYCL kernel function from start to end
- A work-item can run on CPU threads, SIMD lanes, GPU threads, or any other kind of processing element



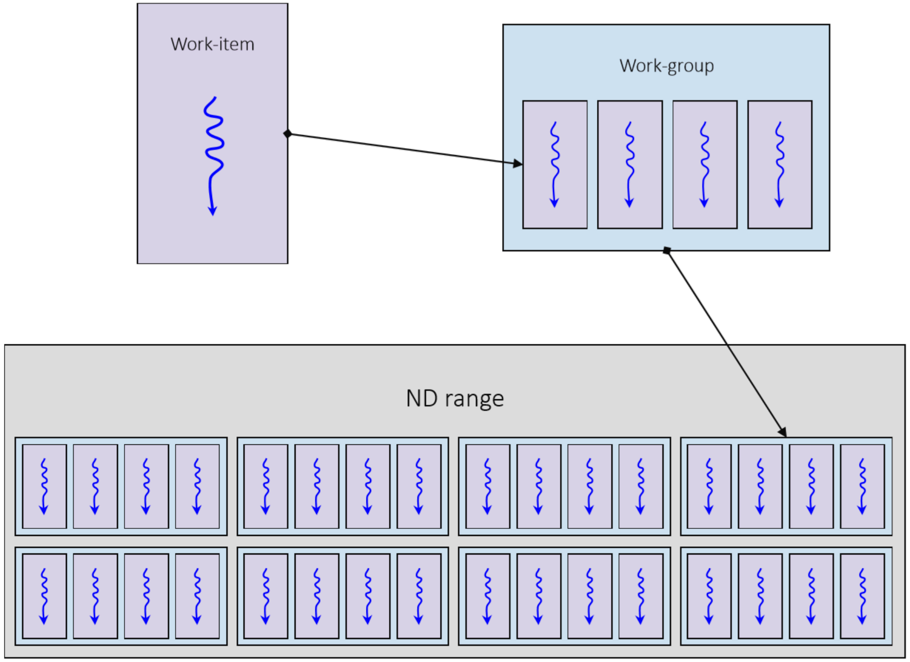
SYCL EXECUTION MODEL

- Work-items are collected together into **work-groups**
- The size of work-groups is generally relative to what is optimal on the device being targeted
- It can also be affected by the resources used by each work-item



SYCL EXECUTION MODEL

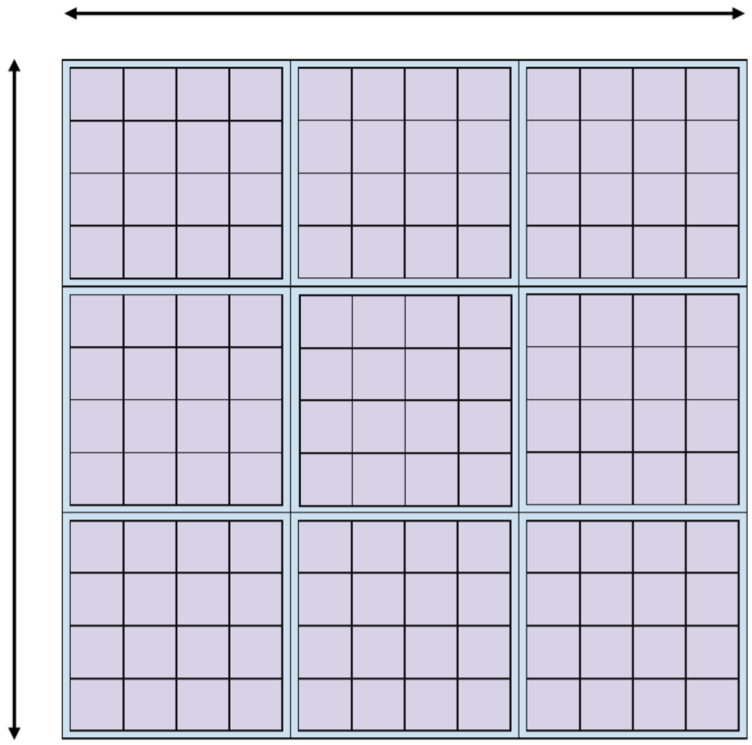
- SYCL kernel functions are invoked within an **nd-range**
- An nd-range has a number of work-groups and subsequently a number of work-items
- Work-groups always have the same number of work-items



SYCL EXECUTION MODEL

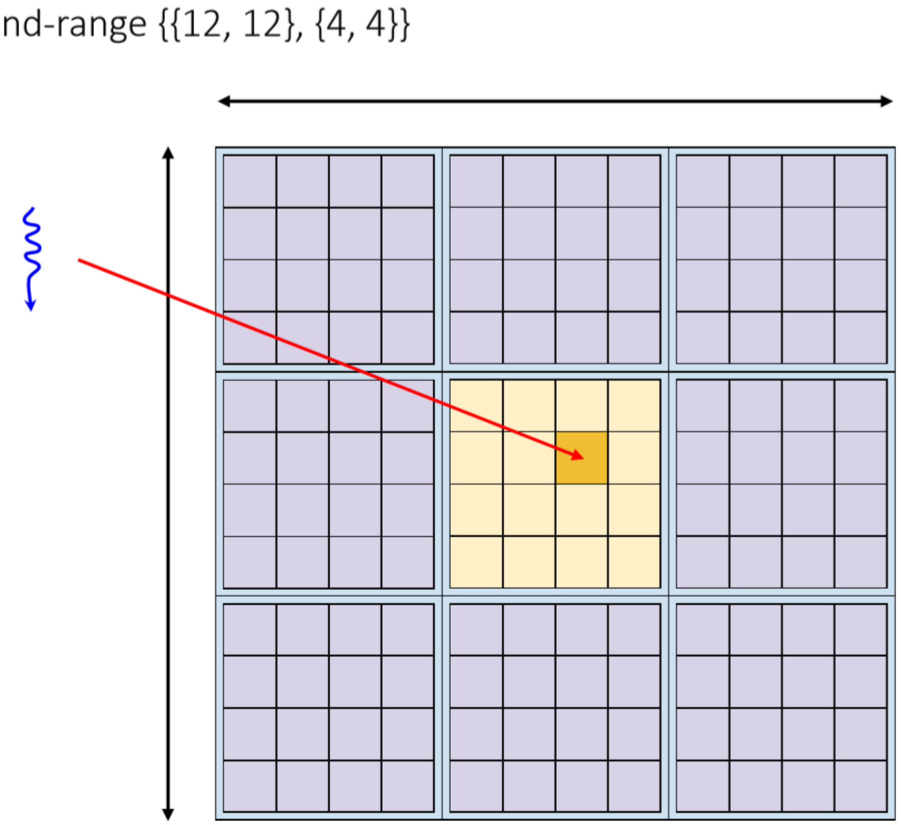
- The nd-range describes an **iteration space**: how it is composed in terms of work-groups and work-items
- An nd-range can be 1, 2 or 3 dimensions
- An nd-range has two components
  - The **global-range** describes the total number of work-items in each dimension
  - The **local-range** describes the number of work-items in a work-group in each dimension

nd-range {{12, 12}, {4, 4}}



SYCL EXECUTION MODEL

- Each invocation in the iteration space of an nd-range is a work-item
- Each invocation knows which work-item it is on and can query certain information about its position in the nd-range
- Each work-item has the following:
  - **Global range**: {12, 12}
  - **Global id**: {5, 6}
  - **Group range**: {3, 3}
  - **Group id**: {1, 1}
  - **Local range**: {4, 4}
  - **Local id**: {1, 2}

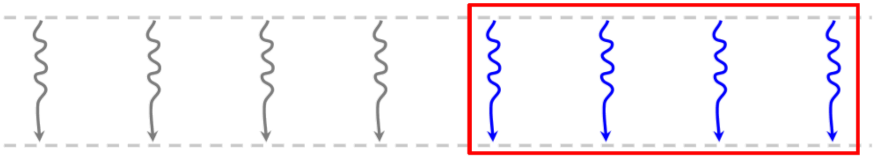


SYCL EXECUTION MODEL

Typically an nd-range invocation SYCL will execute the SYCL kernel function on a very large number of work-items, often in the thousands

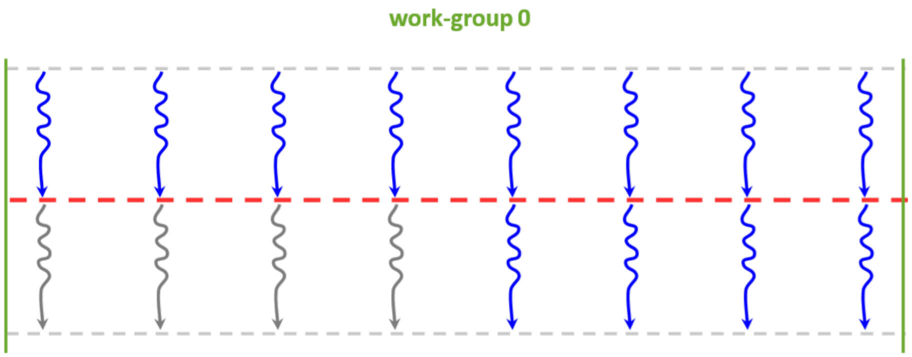
SYCL EXECUTION MODEL

- Multiple work-items will generally execute concurrently
- On vector hardware this is often done in lock-step, which means the same hardware instructions
- The number of work-items that will execute concurrently can vary from one device to another
- Work-items will be batched along with other work-items in the same work-group
- The order work-items and work-groups are executed in is implementation defined



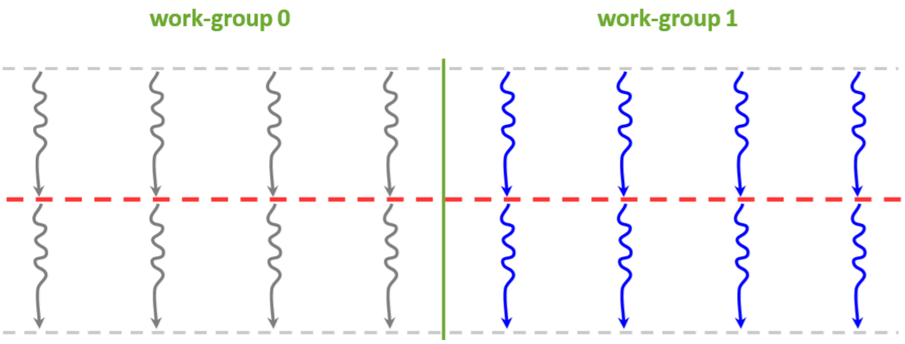
SYCL EXECUTION MODEL

- Work-items in a work-group can be synchronized using a work-group barrier
  - All work-items within a work-group must reach the barrier before any can continue on



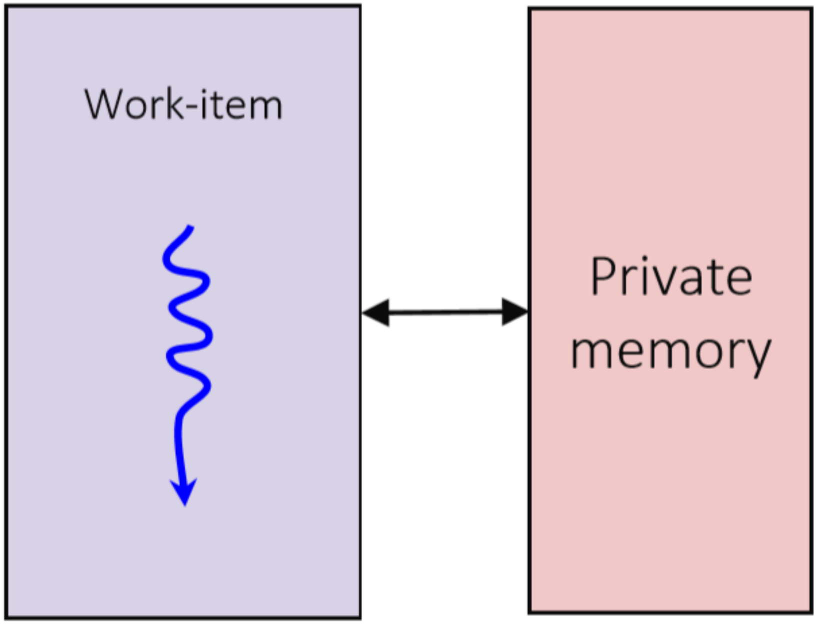
SYCL EXECUTION MODEL

- SYCL does not support synchronizing across all work-items in the nd-range
- The only way to do this is to split the computation into separate SYCL kernel functions

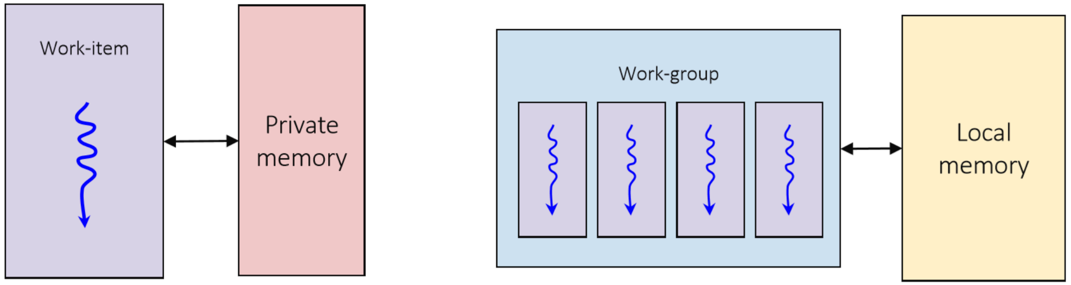


SYCL MEMORY MODEL

- Each work-item can access a dedicated region of **private memory**
- A work-item cannot access the private memory of another work-item

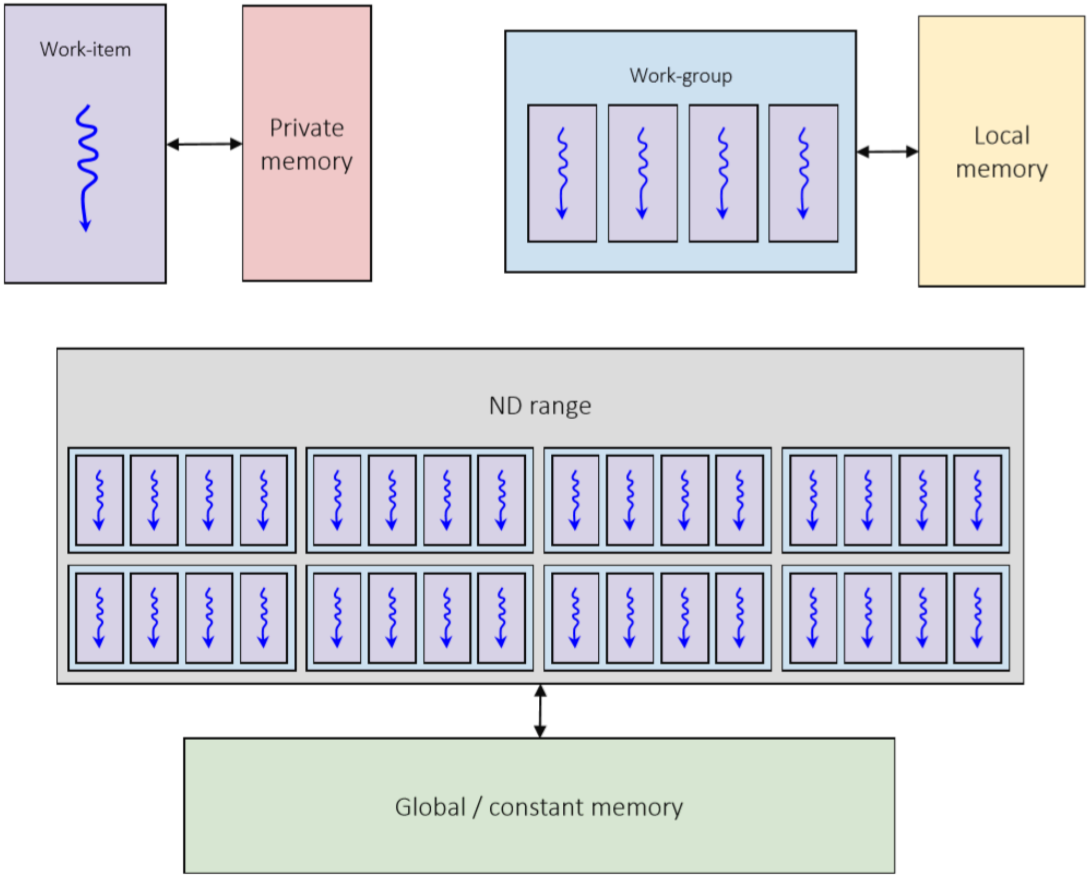


SYCL MEMORY MODEL



- Each work-item can access a dedicated region of **local memory** accessible to all work-items in a work-group
- A work-item cannot access the local memory of another work-group

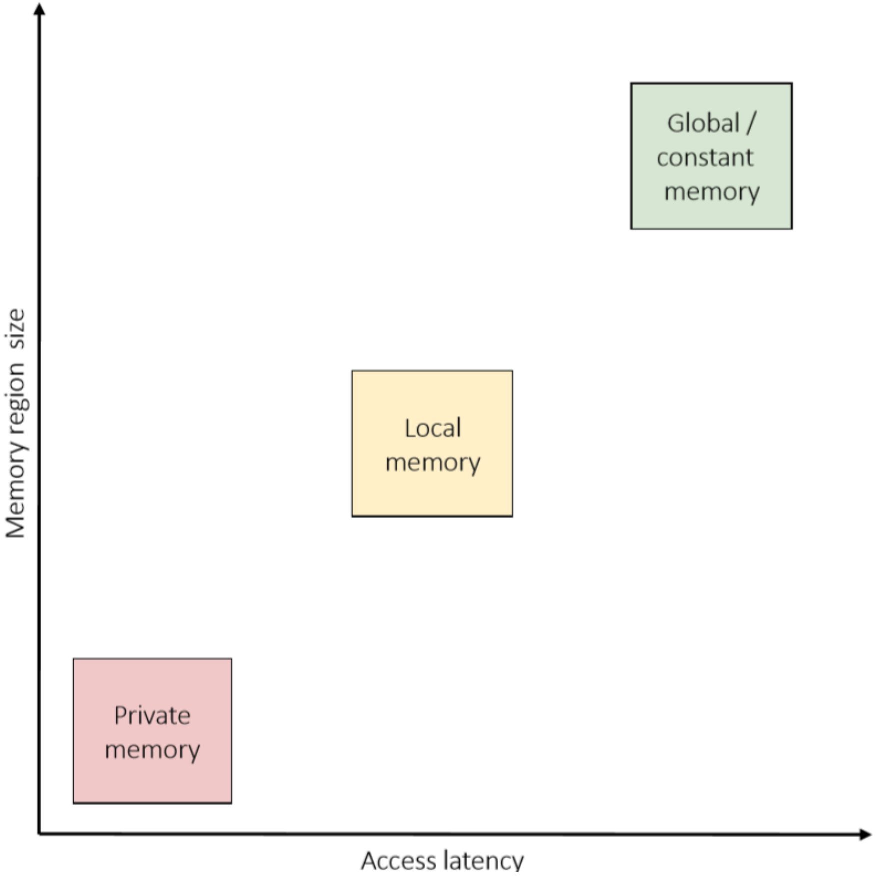
SYCL MEMORY MODEL



- Each work-item can access a single region of **global memory** that's accessible to all work-items in a ND-range
- Each work-item can also access a region of global memory reserved as **constant memory**, which is read-only

SYCL MEMORY MODEL

- Each memory region has a different size and access latency
- Global / constant memory is larger than local memory and local memory is larger than private memory
- Private memory is faster than local memory and local memory is faster than global / constant memory



EXPRESSING PARALLELISM

```
cgh.parallel_for(
  range<1>(1024),
  [=](id<1> idx){
    /* kernel function code */
});
```

- Embarrassingly parallel overload taking a **range** object for the global range, runtime decides local range
- An **id** parameter represents the index within the global range

```
cgh.parallel_for(
  range<1>(1024),
  [=](item<1> item){
    /* kernel function code */
});
```

- Embarrassingly parallel overload taking a **range** object for the global range, runtime decides local range
- An **item** parameter represents the global range & index in the global range

```
q.parallel_for(
  nd_range<1>(
        range<1>(1024),
range<1>(32)),
  [=](nd_item<1> ndItem){
         ernel function code */
});
```

- Overload taking an **nd_range** object specifies the global and local range
- An **nd_item** parameter represents the global and local range and index

QUESTIONS

EXERCISE

Code_Exercises/Section_6_ND_Range_Kernel/source

Implement a SYCL application that will perform a vector add using `parallel_for`, adding multiple elements in parallel.