SYCL Academy

LOCAL MEMORY

NHR SW

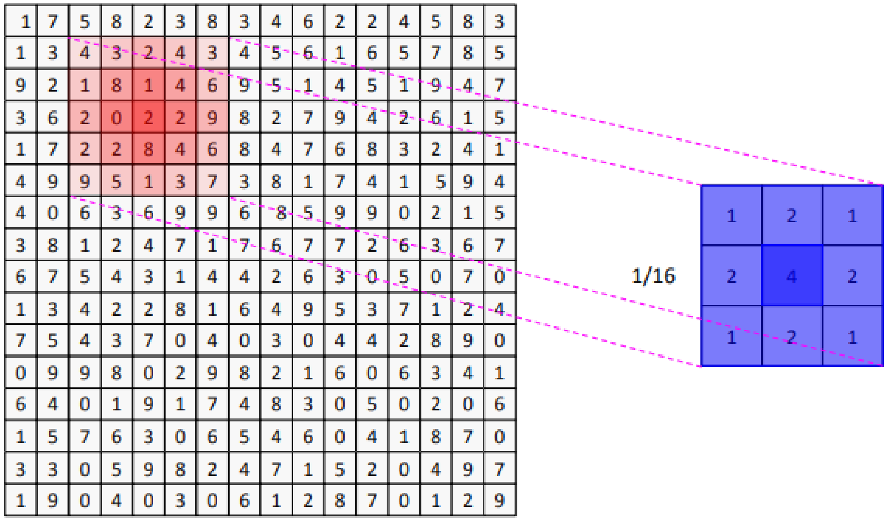
LEARNING OBJECTIVES

- Learn about tiling using local memory
- Learn about how to synchronize work-groups

COST OF ACCESSING GLOBAL MEMORY

- As we covered earlier global memory is very expensive to access.
- Even with coalesced global memory access if you are accessing the same elements multiple times that can be expensive.
- Instead you want to cache those values in a lower latency memory.

WHY ARE IMAGE CONVOLUTIONS GOOD ON A GPU?



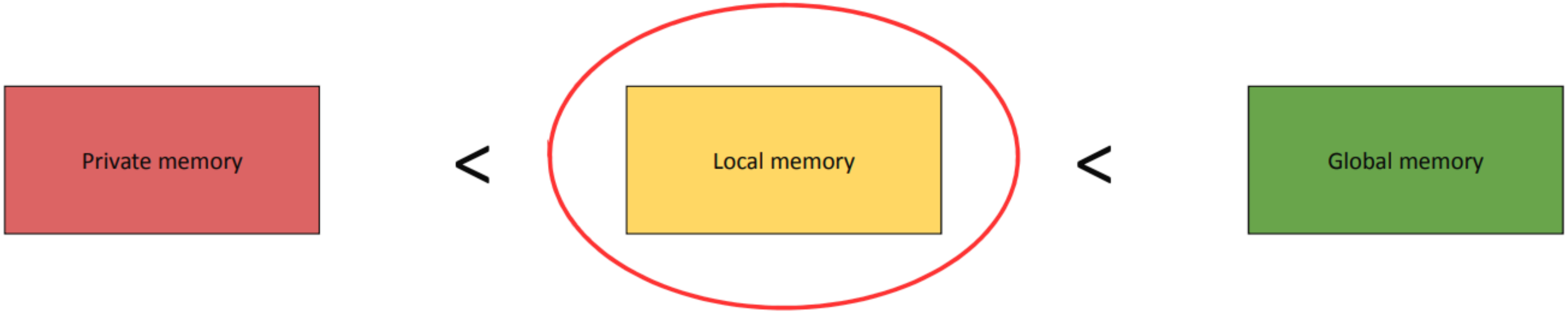
1	7	5	8	2	3	8	3	4	6	2	2	4	5	8	3
1	3	4	3	2	4	3	4	5	6	1	6	5	7	8	5
9	2	1	8	1	4	6	9	5	1	4	5	1	9	4	7
3	6	2	0	2	2	9	8	2	7	9	4	2	6	1	5
1	7	2	2	8	4	6	8	4	7	6	8	3	2	4	1
4	9	9	5	1	3	7	3	8	1	7	4	1	5	9	4
4	0	6	3	6	9	9	6	8	5	9	9	0	2	1	5
3	8	1	2	4	7	1	7	6	7	7	2	6	3	6	7
6	7	5	4	3	1	4	4	2	6	3	0	5	0	7	0
1	3	4	2	2	8	1	6	4	9	5	3	7	1	2	4
7	5	4	3	7	0	4	0	3	0	4	4	2	8	9	0
0	9	9	8	0	2	9	8	2	1	6	0	6	3	4	1
6	4	0	1	9	1	7	4	8	3	0	5	0	2	0	6
1	5	7	6	3	0	6	5	4	6	0	4	1	8	7	0
3	3	0	5	9	8	2	4	7	1	5	2	0	4	9	7
1	9	0	4	0	3	0	6	1	2	8	7	0	1	2	9

1/16

1	2	1
2	4	2
1	2	1

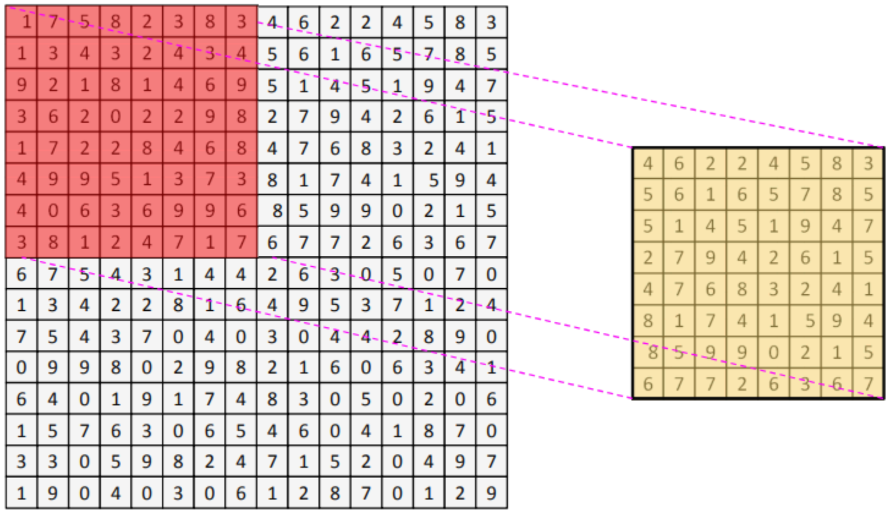
- Looking at the image convolution example.
- For each output pixel we are reading up to NxM pixels from the input image, where N and M are the dimensions of the filter.
- This means each input pixel is being read up to NxM times:
- 3x3 filter: up to 9 ops.
- 5x5 filter: up to 25 ops.
- 7x7 filter: up to 49 ops.
- If each of these operations is a separate load from global memory this becomes very expensive.

USING LOCAL MEMORY



- The solution is local memory.
- Local memory is generally on-chip and doesn't have a cache as it's managed manually so is much lower latency.
- Local memory is a smaller dedicated region of memory per work-group.
- Local memory can be used to cache, allowing us to read from global memory just once and then read from local memory instead, often referred to as a scratchpad.

TILING



1	7	5	8	2	3	8	3	4	6	2	2	4	5	8	3
1	3	4	3	2	4	3	4	5	6	1	6	5	7	8	5
9	2	1	8	1	4	6	9	5	1	4	5	1	9	4	7
3	6	2	0	2	2	9	8	2	7	9	4	2	6	1	5
1	7	2	2	8	4	6	8	4	7	6	8	3	2	4	1
4	9	9	5	1	3	7	3	8	1	7	4	1	5	9	4
4	0	6	3	6	9	9	6	8	5	9	9	0	2	1	5
3	8	1	2	4	7	1	7	6	7	7	2	6	3	6	7
6	7	5	4	3	1	4	4	2	6	3	0	5	0	7	0
1	3	4	2	2	8	1	6	4	9	5	3	7	1	2	4
7	5	4	3	7	0	4	0	3	0	4	4	2	8	9	0
0	9	9	8	0	2	9	8	2	1	6	0	6	3	4	1
6	4	0	1	9	1	7	4	8	3	0	5	0	2	0	6
1	5	7	6	3	0	6	5	4	6	0	4	1	8	7	0
3	3	0	5	9	8	2	4	7	1	5	2	0	4	9	7
1	9	0	4	0	3	0	6	1	2	8	7	0	1	2	9

4	6	2	2	4	5	8	3
5	6	1	6	5	7	8	5
5	1	4	5	1	9	4	7
2	7	9	4	2	6	1	5
4	7	6	8	3	2	4	1
8	1	7	4	1	5	9	4
8	5	9	9	0	2	1	5
6	7	7	2	6	3	6	7

- The iteration space of the kernel function is mapped across multiple work-groups.
- Each work-group has it's own region of local memory.
- You want to split the input image data into tiles, one for each work-group.

LOCAL ACCESSORS

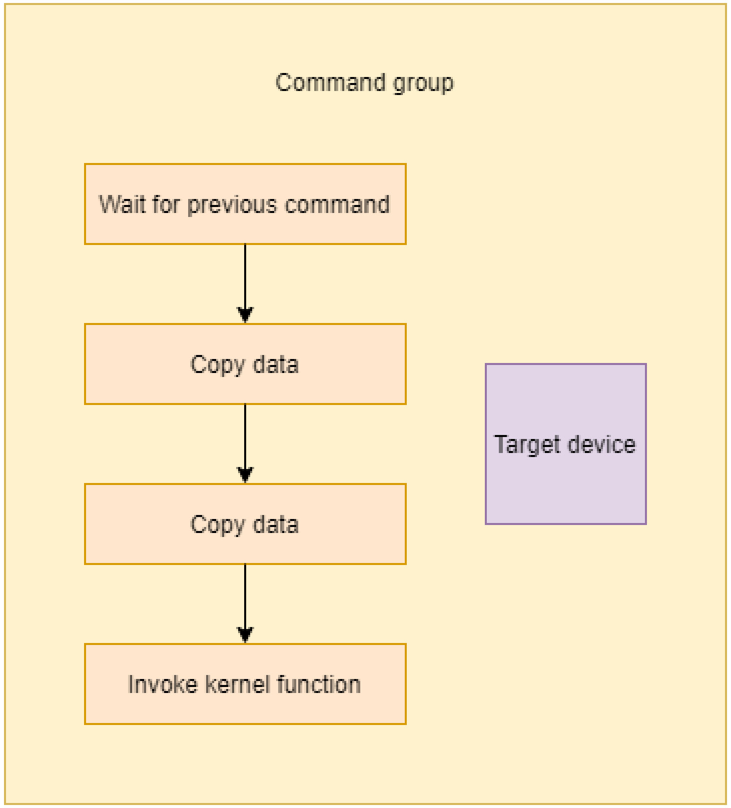
```
auto scratchpad = sycl::local_accessor<int, 1>
(sycl::range{workGroupSize}, cgh);
```

- Local memory is allocated via an `local_accessor`.
- They allocate memory per work-group for the duration of the kernel function.
- The `range` provided is the number of elements of the specified type to allocate per work-group.

LOCAL ACCESSORS - COMMAND GROUPS BACKGROUND

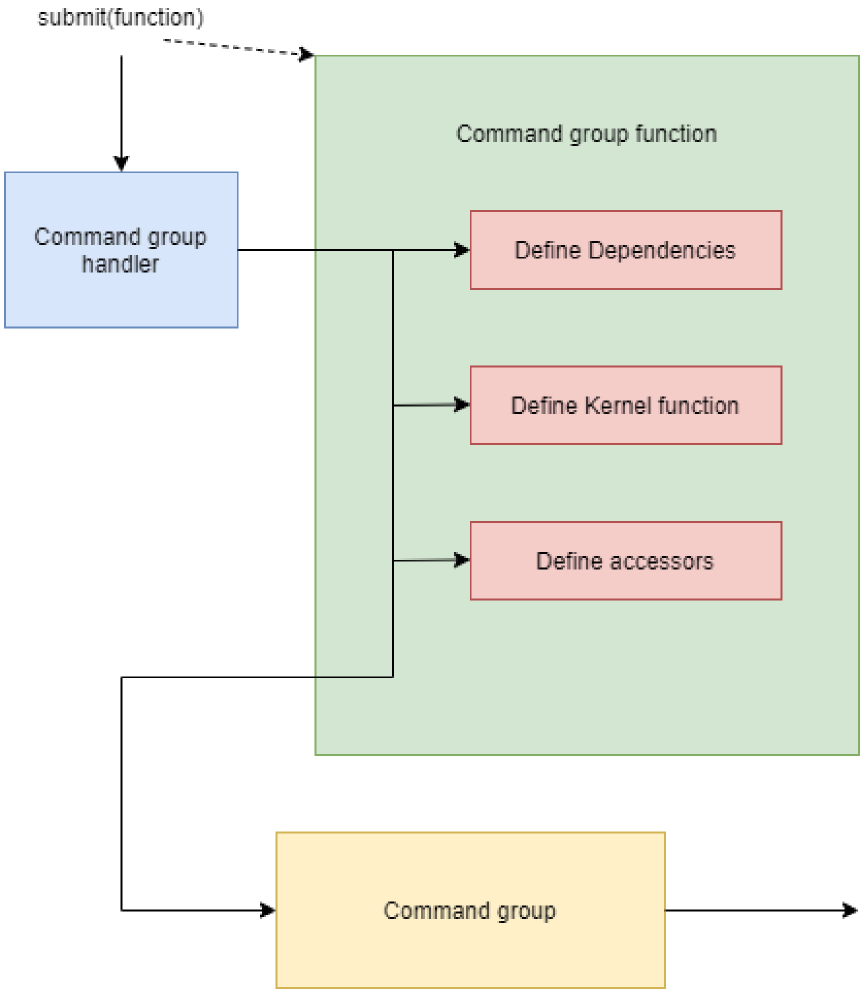
```
myQueue.submit([&](sycl::handler& cgh) {
  auto scratchpad = sycl::local_accessor<int, 1>(sycl::range{workGroupSize}, cgh);
  cgh.parallel_for(sycl::nd_range{globalRange, {workGroupSize}},
    [=](nd_item<1> item) {...});
});
```

COMMAND GROUPS



- In the buffer/accessor model or, when local memory is required, commands must be enqueued via command groups.
- A command group represents a series of commands to be executed by a device.
- These commands include:
  - Invoking kernel functions on a device.
  - Copying data to and from a device.
  - Waiting on other commands to complete.
  - Allocating local memory

COMPOSING COMMAND GROUPS



- Command groups are composed by calling the `submit` member function on a `queue`.
- The `submit` function takes a command group function which acts as a factory for composing the command group.
- The `submit` function creates a `handler` and passes it into the command group function.
- The `handler` then composes the command group.

LOCAL ACCESSORS - IN CONTEXT

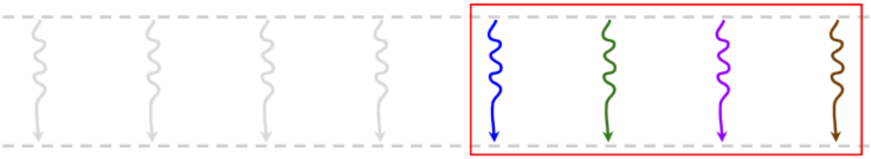
```
myQueue.submit([&](sycl::handler& cgh) {
  auto scratchpad = sycl::local_accessor<int, 1>(sycl::range{workGroupSize}, cgh);
  cgh.parallel_for(sycl::nd_range{globalRange, {workGroupSize}},
    [=](nd_item<1> item) {...});
});
```

- The call to `queue::submit` gives access to the command group handler `cgh`
- The `cgh` is needed to allocate the local memory
- In this scenario `parallel_for` is added as the last action of this command group.
    - Caution: only one kernel can be submitted per command group
- The kernel submission mechanisms, we got to know so far, are just convenient wrappers for this.

SYNCHRONIZATION

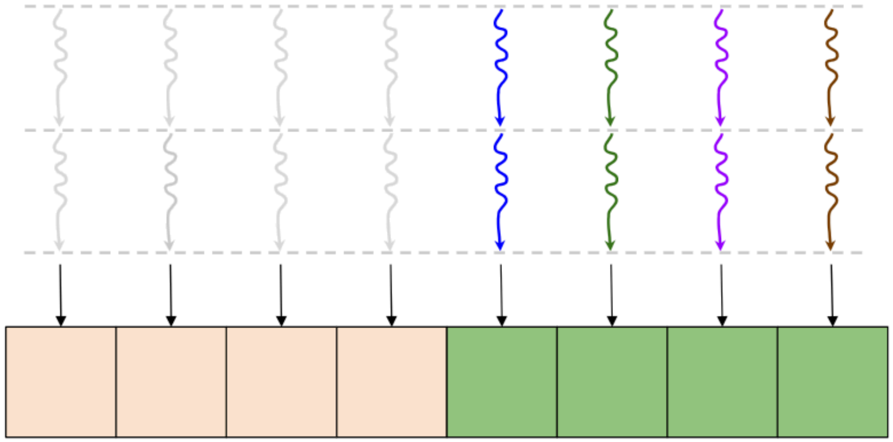
- Local memory can be used to share partial results between work-items.
- When doing so it's important to synchronize between writes and reads to memory to ensure all work-items have reached the same point in the program.

SYNCHRONIZATION



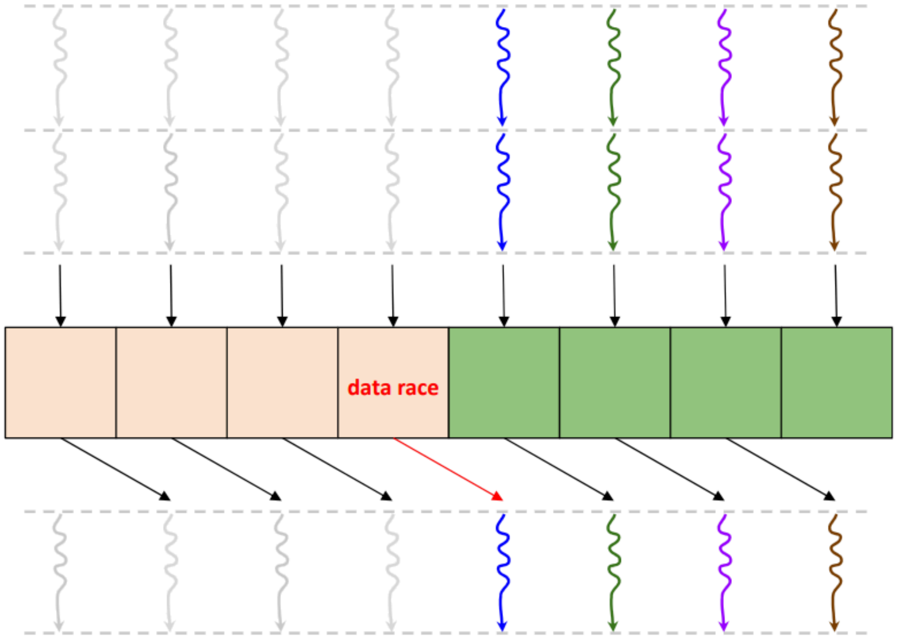
- Remember that work-items are not guaranteed to all execute at the same time (in parallel).

SYNCHRONIZATION



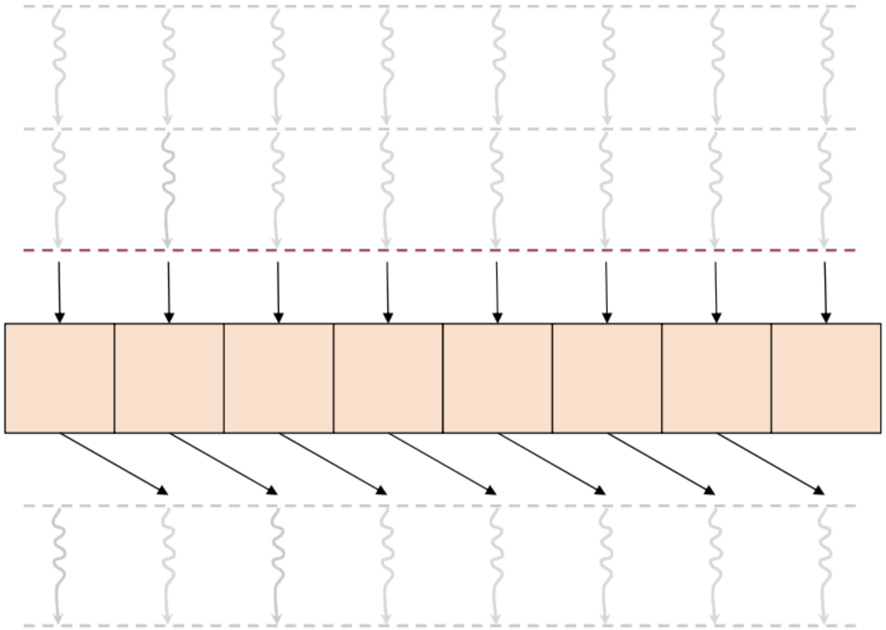
- A work-item can share results with other work-items via local (or global) memory.

SYNCHRONIZATION



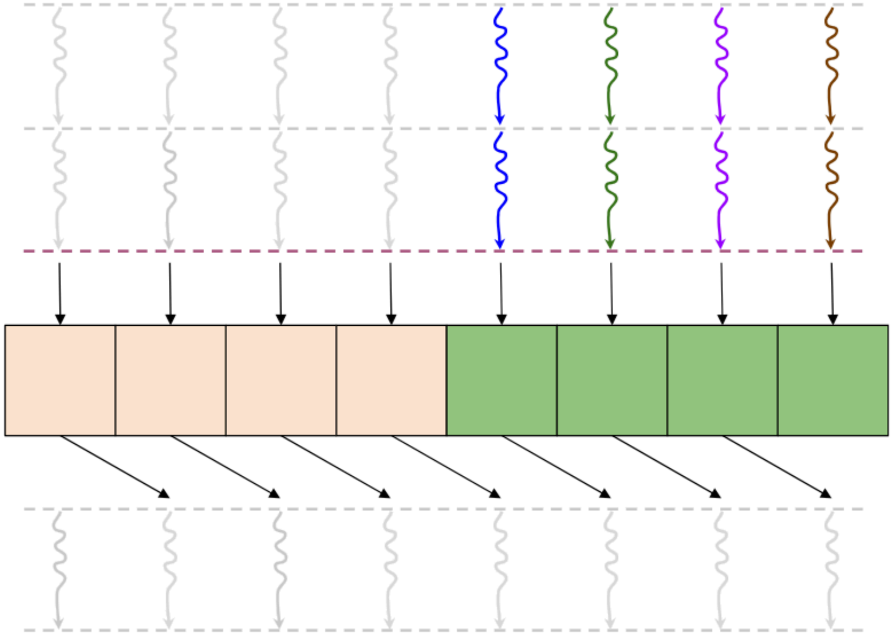
- This means it's possible for a work-item to read a result that hasn't been written to yet.
- This creates a data race.

SYNCHRONIZATION



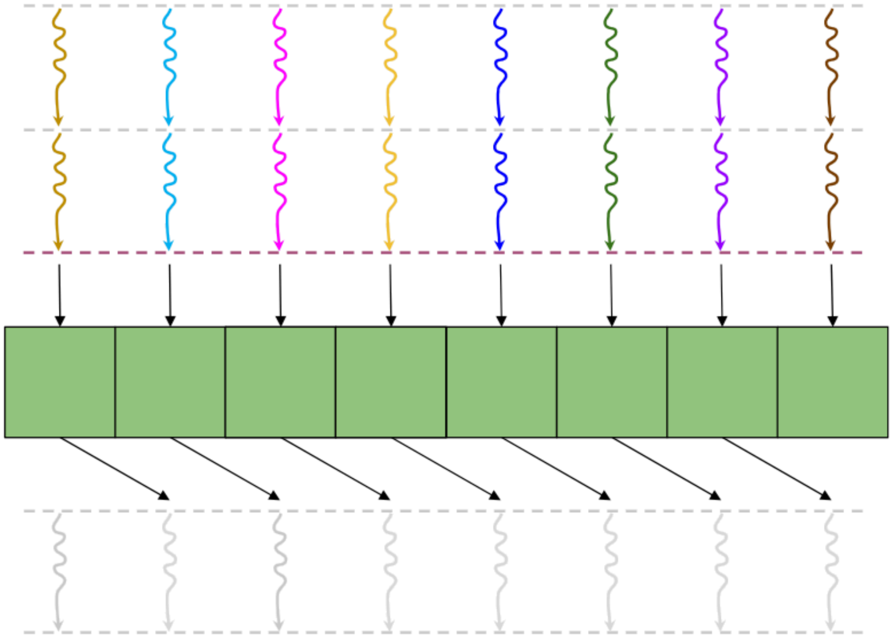
- This problem can be solved with a synchronization primitive called a work-group barrier.

SYNCHRONIZATION



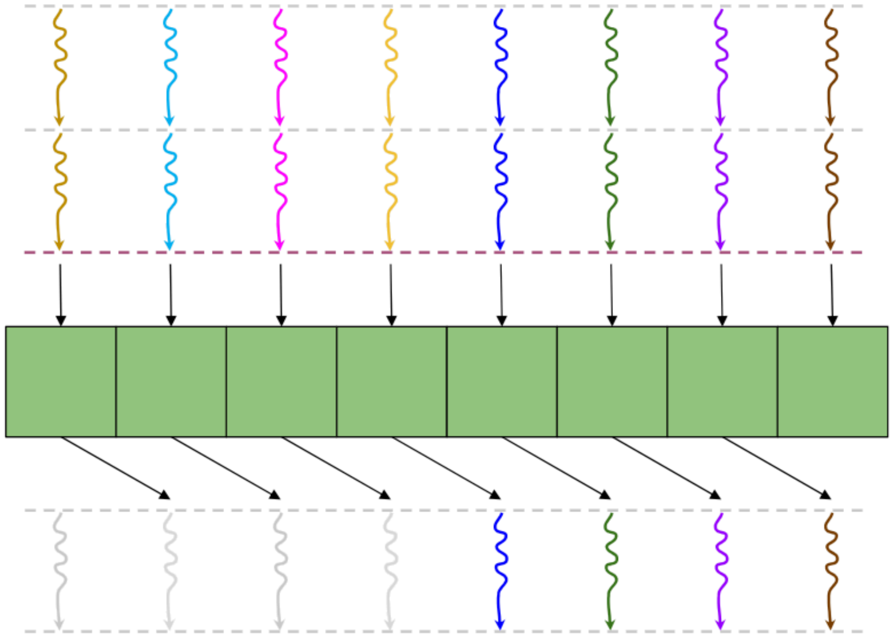
- When a work-group barrier is inserted work-items will wait until all work-items in the work-group have reached that point.

SYNCHRONIZATION



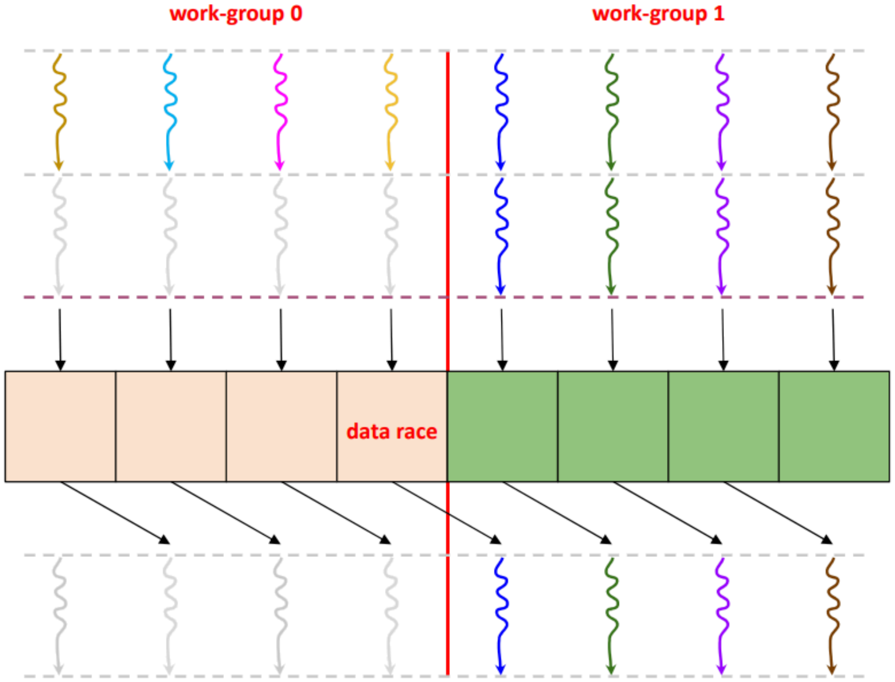
- Only then can any work-items in the work-group continue execution.

SYNCHRONIZATION



- So now you can be sure that all of the results that you want to read have been written to.

SYNCHRONIZATION



- However note that this does not apply across work-group boundaries.
- So if you write in a work-item of one work-group and then read it in a work-item of another work-group you again have a data race.
- Furthermore, remember that work-items can only access their own local memory and not that of any other work-groups.

GROUP_BARRIER

```
sycl::group_barrier(item.get_group());
```

- Work-group barriers can be invoked by calling `group_barrier` and passing a `group` object.
- You can retrieve a `group` object representing the current work-group by calling `get_group` on an `nd_item`.
- Note this requires the `nd_range` variant of `parallel_for`.

LOCAL MEMORY IMAGE CONVOLUTION PERFORMANCE

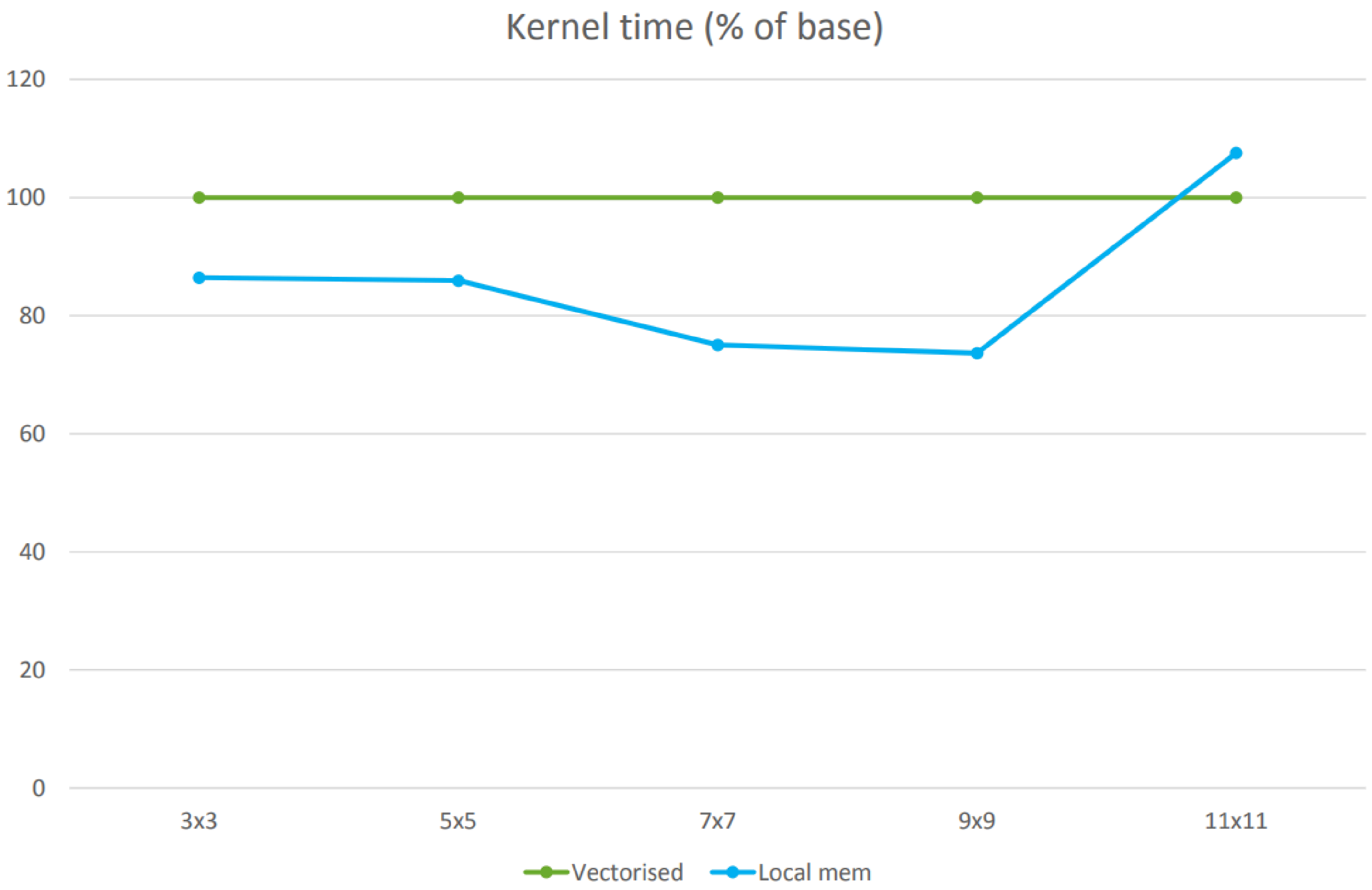


Image conv

512x512 source image

Intel HD Graphics 530

QUESTIONS

EXERCISE

Code_Exercises/Section_10_Local_Memory_Tiling/source

Use local memory to cache a tile of the input image data per work-group.