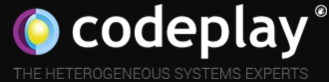


How to Deploy AI Software to Self Driving Cars

Rod Burns, Gordon Brown, Meenakshi Ravindran and Nicolas Miller

IWOCL`19 - May 2019

Codeplay - Connecting AI to Silicon

Products

C++ platform via the SYCL™ open standard, enabling vision & machine learning e
TensorFlow™

The heart of Codeplay's compute technology
enabling OpenCL™, SPIR™, HSA™ and Vulkan™

Addressable Markets

Automotive (ISO 26262)
IoT, Smartphones & Tablets
High Performance Compute (HPC)
Medical & Industrial

Technologies: Vision Processing
Machine Learning
Artificial Intelligence
Big Data Compute

Company

High-performance software solutions for custom heterogeneous systems

Enabling the toughest processor systems with tools and middleware based on open standards

Established 2002 in Scotland

~70 employees

Customers

© 2019 Codeplay Software Ltd.

Agenda

Emergent hardware for AI in automotive

Overview of OpenCL/SYCL programming model

Mapping typical hardware to the OpenCL/SYCL programming model

The Renesas R-Car architecture

Extending OpenCL & SYCL for R-Car

Optimising machine learning algorithms using R-Car

© 2019 Codeplay Software Ltd.

Autonomous driving is one of the biggest challenges in technology

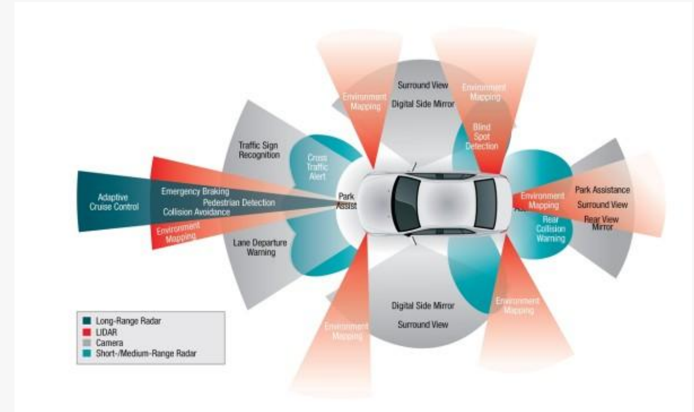
The automotive industry needs to deliver the latest AI technologies with safety, high performance and low power consumption

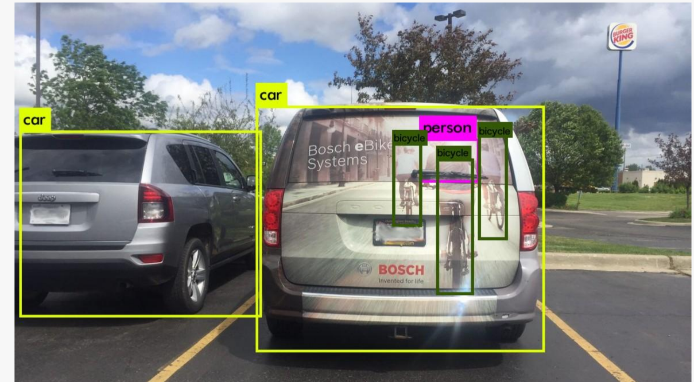
© 2019 Codeplay Software Ltd.

Delivering an autonomous vehicle is a huge software and hardware challenge

It requires scaling up software development to very high levels of complexity, performance and risk

Whilst maintaining low power consumption





© 2019 Codeplay Software Ltd.

Renesas R-Car architecture

- Embedded automotive architecture
- Optimized for computer vision processing and machine learning
- Designed for low latency, low power consumption and low cost

© 2019 Codeplay Software Ltd.

SYCL-BLAS, SYCL-DNN

SYCL

OpenCL

© 2019 Codeplay Software Ltd.

Agenda

Emergent hardware for AI in automotive

Overview of OpenCL/SYCL programming model

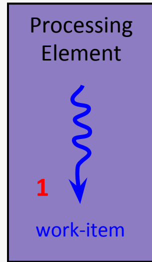
Mapping typical hardware to the OpenCL/SYCL programming model

The Renesas R-Car architecture

Extending OpenCL & SYCL for R-Car

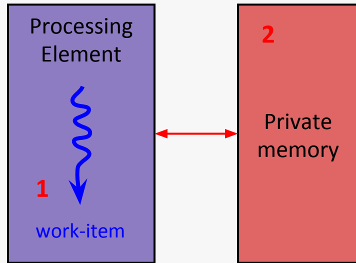
Optimising machine learning algorithms using R-Car

© 2019 Codeplay Software Ltd.



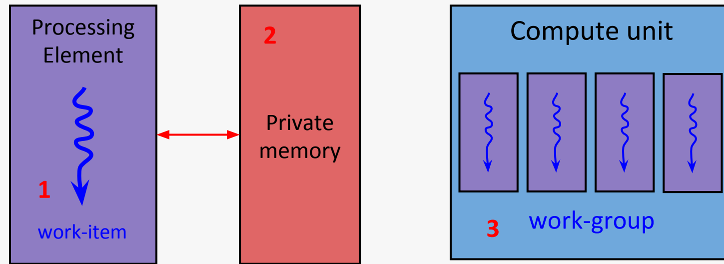
1. A processing element executes a single work-item

© 2019 Codeplay Software Ltd.



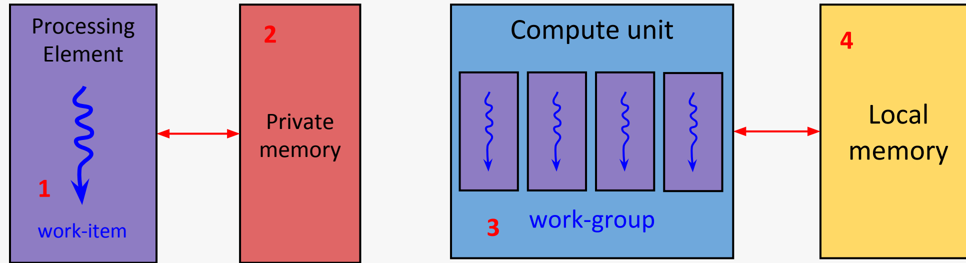
1. A processing element executes a single work-item
2. Each work-item can access private memory, a dedicated memory region for each processing element

© 2019 Codeplay Software Ltd.



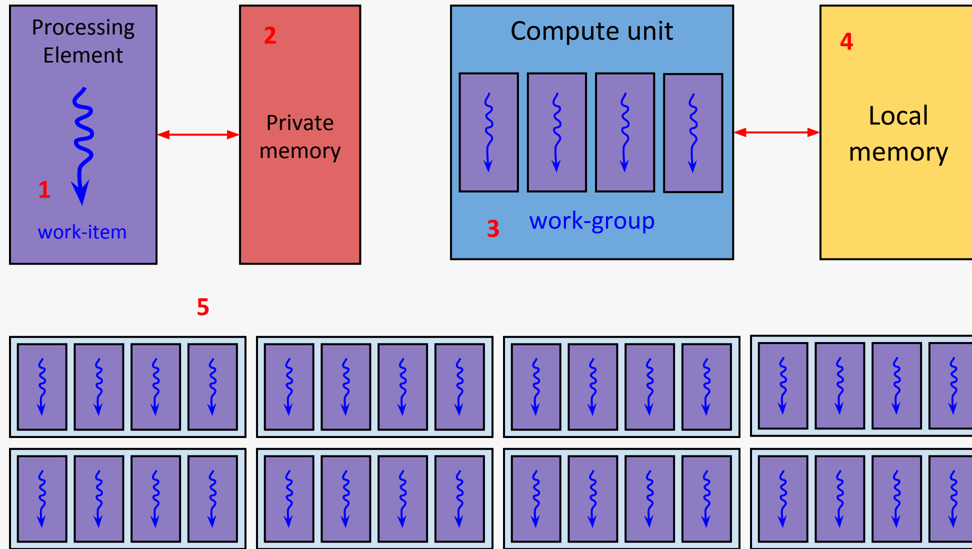
1. A processing element executes a single work-item
2. Each work-item can access private memory, a dedicated memory region for each processing element
3. A compute unit executes a work-group, composed of multiple work-items, one for each processing element in the compute unit

© 2019 Codeplay Software Ltd.



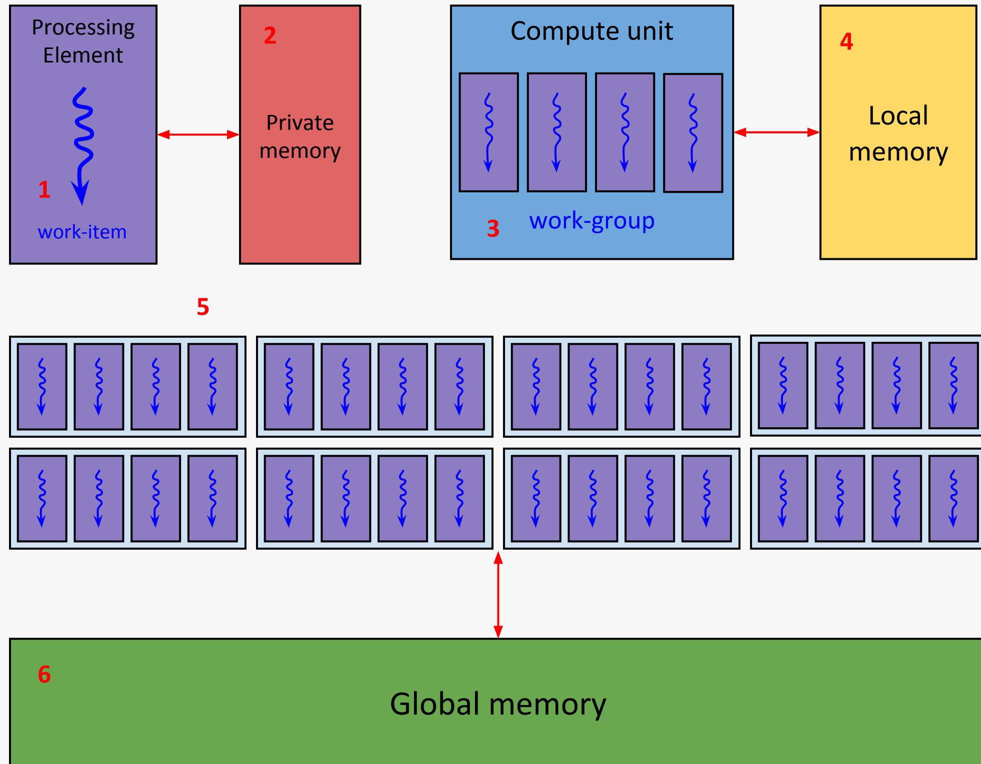
1. A processing element executes a single work-item
2. Each work-item can access private memory, a dedicated memory region for each processing element
3. A compute unit executes a work-group, composed of multiple work-items, one for each processing element in the compute unit
4. Each work-item can access local memory, a dedicated memory region for each compute unit

© 2019 Codeplay Software Ltd.



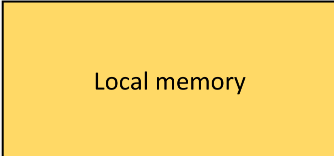
1. A processing element executes a single work-item
2. Each work-item can access private memory, a dedicated memory region for each processing element
3. A compute unit executes a work-group, composed of multiple work-items, one for each processing element in the compute unit
4. Each work-item can access local memory, a dedicated memory region for each compute unit
5. A device can execute multiple work-groups

© 2019 Codeplay Software Ltd.



1. A processing element executes a single work-item
2. Each work-item can access private memory, a dedicated memory region for each processing element
3. A compute unit executes a work-group, composed of multiple work-items, one for each processing element in the compute unit
4. Each work-item can access local memory, a dedicated memory region for each compute unit
5. A device can execute multiple work-groups
6. Each work-item can access global memory, a single memory region available to all processing elements

© 2019 Codeplay Software Ltd.

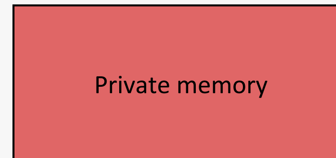


© 2019 Codeplay Software Ltd.

Work-item

© 2019 Codeplay Software Ltd.

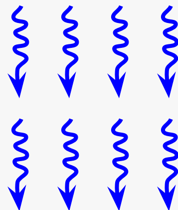
Work-item

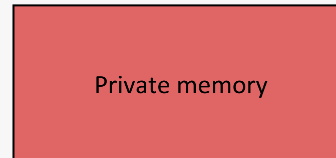


© 2019 Codeplay Software Ltd.

Work-item

Work-group

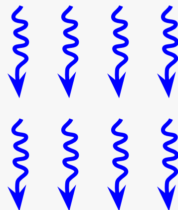




© 2019 Codeplay Software Ltd.

Work-item

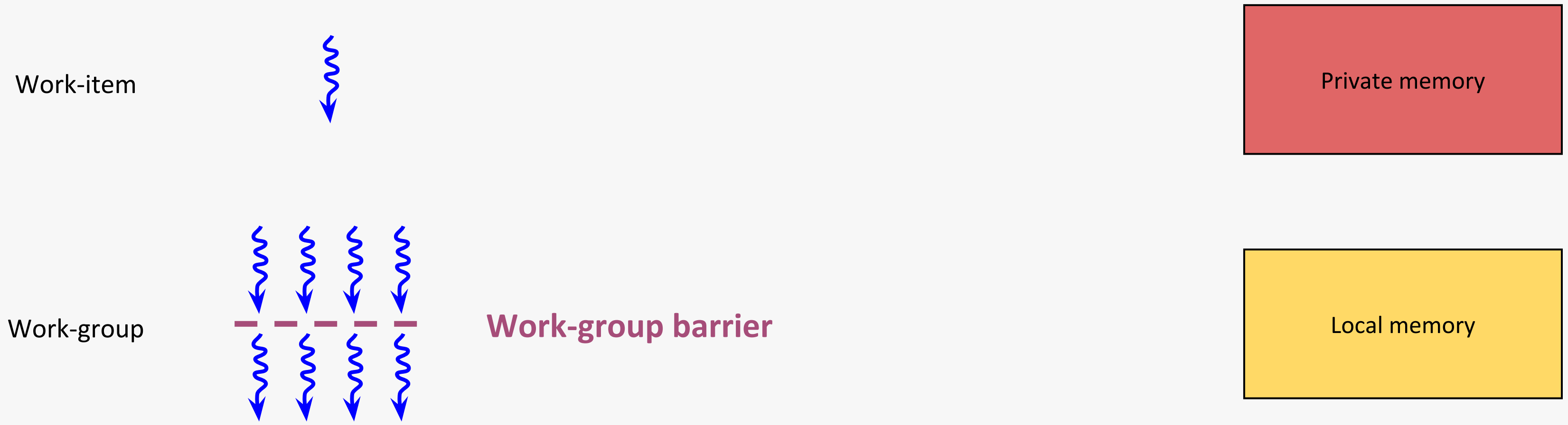
Work-group



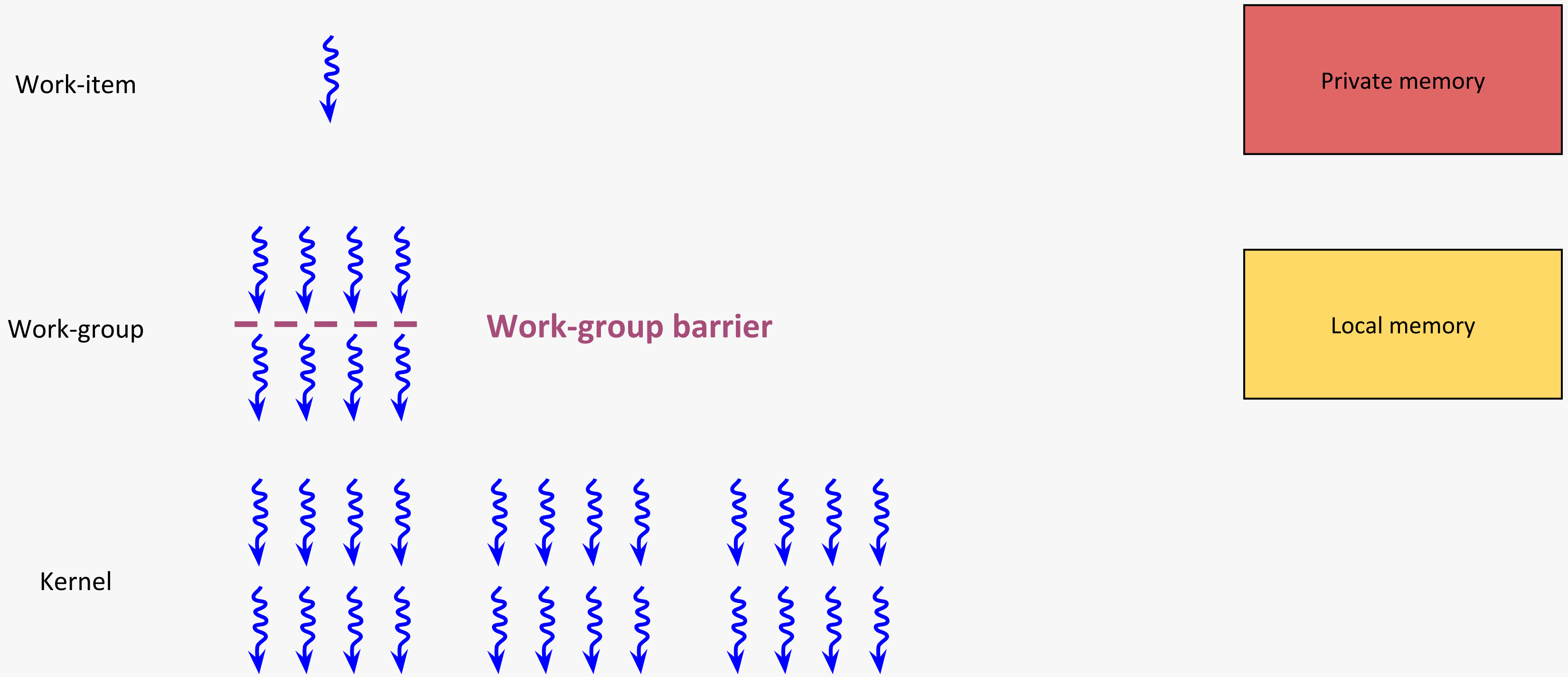
Private memory

Local memory

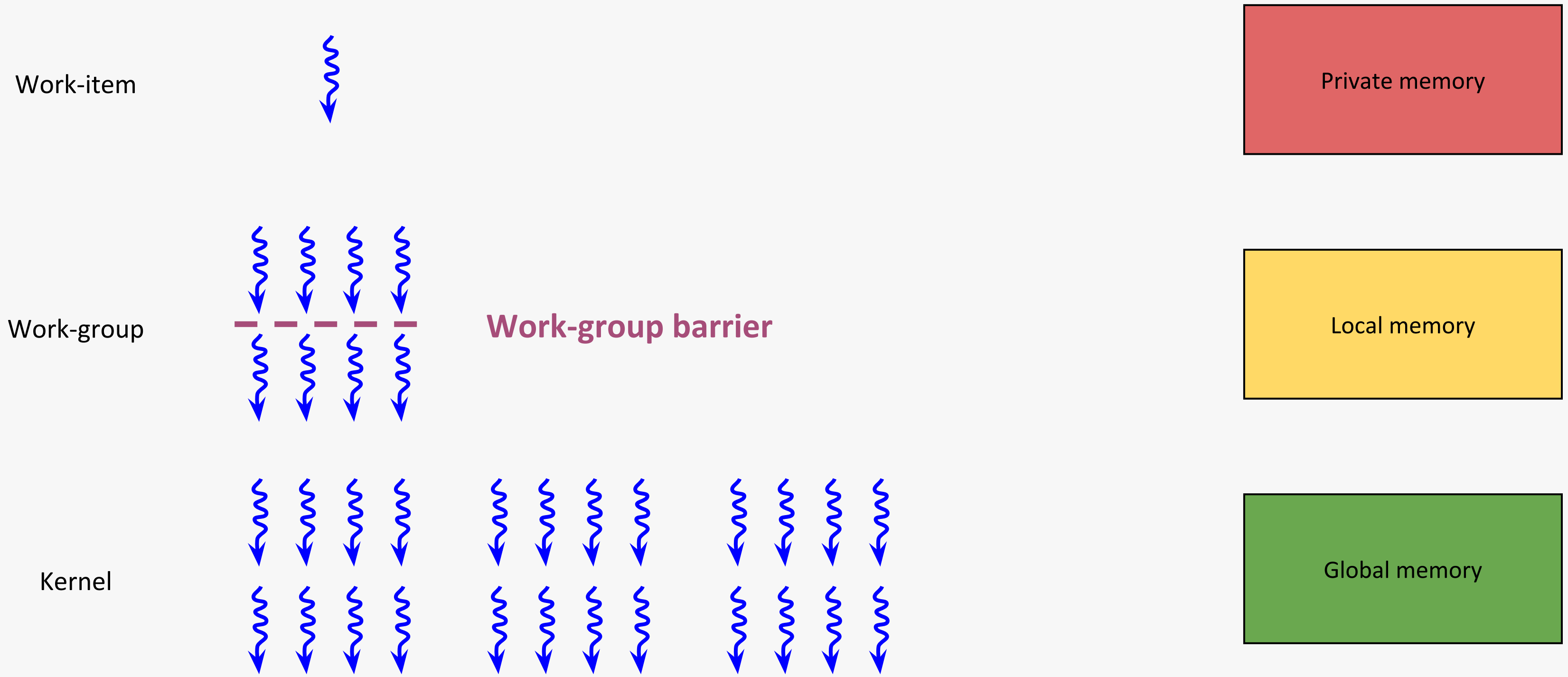
© 2019 Codeplay Software Ltd.



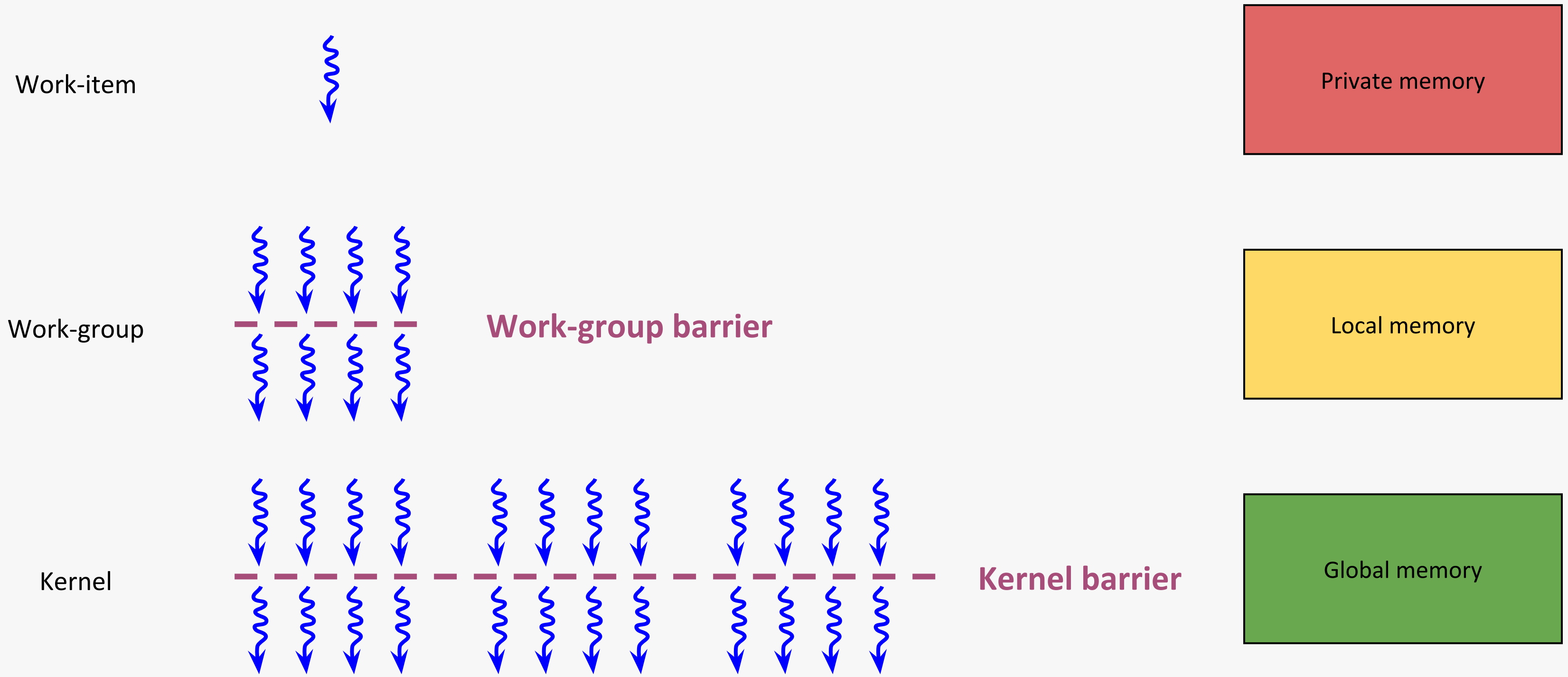
© 2019 Codeplay Software Ltd.



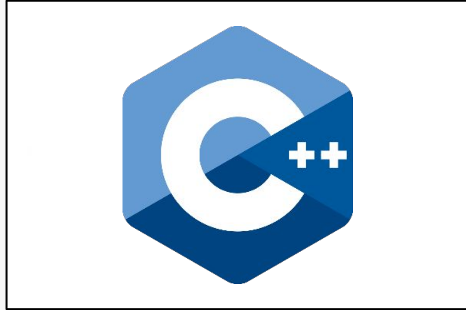
© 2019 Codeplay Software Ltd.

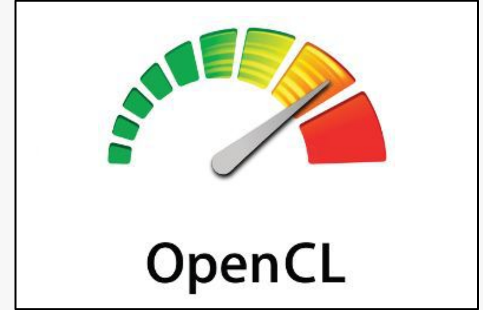


© 2019 Codeplay Software Ltd.



© 2019 Codeplay Software Ltd.





Cross-platform, single-source, high-level, C++ programming layer

Built on top of OpenCL and based on standard C++11

Delivering a heterogeneous programming solution for C++

© 2019 Codeplay Software Ltd.

```
__global__ vec_add(float *a, float *b, float *c) {
  return c[i] = a[i] + b[i];
}

float *a, *b, *c;
vec_add<<<range>>>(a
```

```
vector<float> a, b, c;

#pragma parallel_for
for(int i = 0; i < a.size(); i++) {
```

```
array_view<float> a, b, c;
extent<2> e(64, 64);

parallel_for_each(e, [=](index<2> idx) restrict(amp) {
  c[idx] = a[idx] + b[idx];
});
```

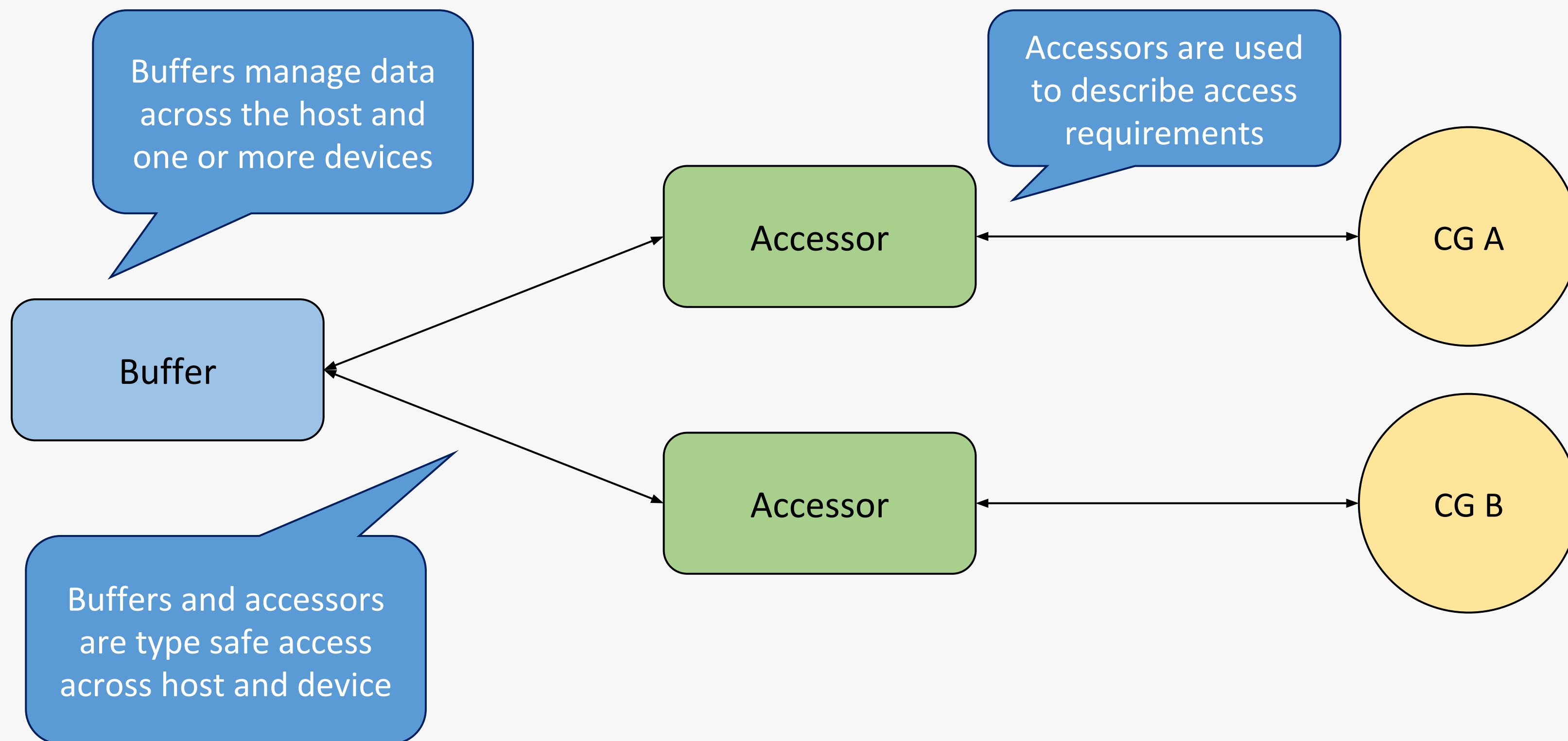
```
cgh.parallel_for<vec_add>(range, [=](cl::sycl::id<2> idx) {
  c[idx] = a[idx] + c[idx];
}));
```


© 2019 Codeplay Software Ltd.

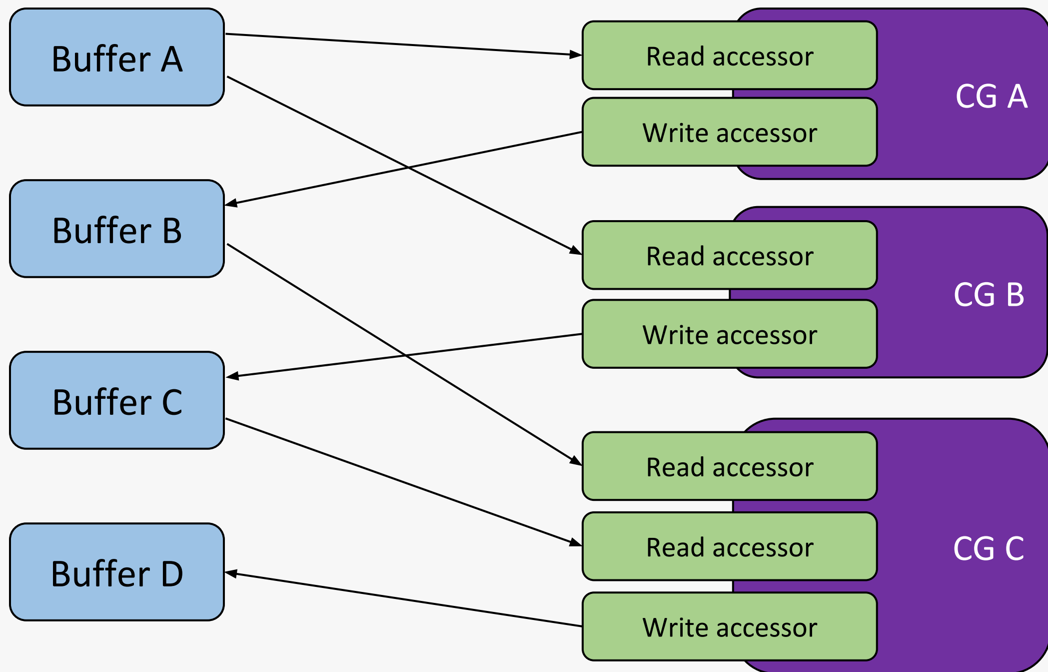
SYCL separates the storage and access of data through the use of buffers and accessors

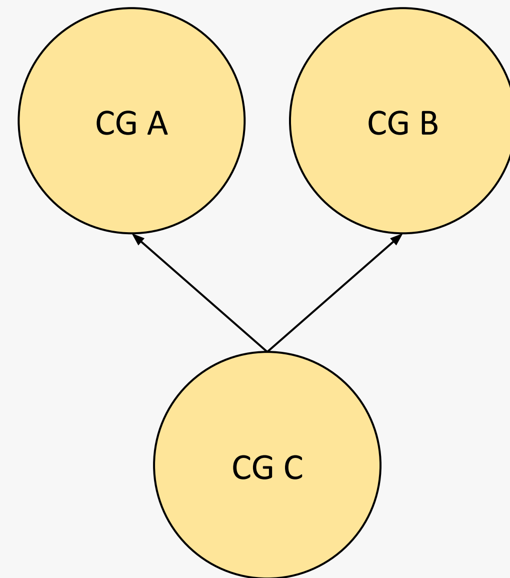
SYCL provides data dependency tracking based on accessors to optimise the scheduling of tasks

© 2019 Codeplay Software Ltd.

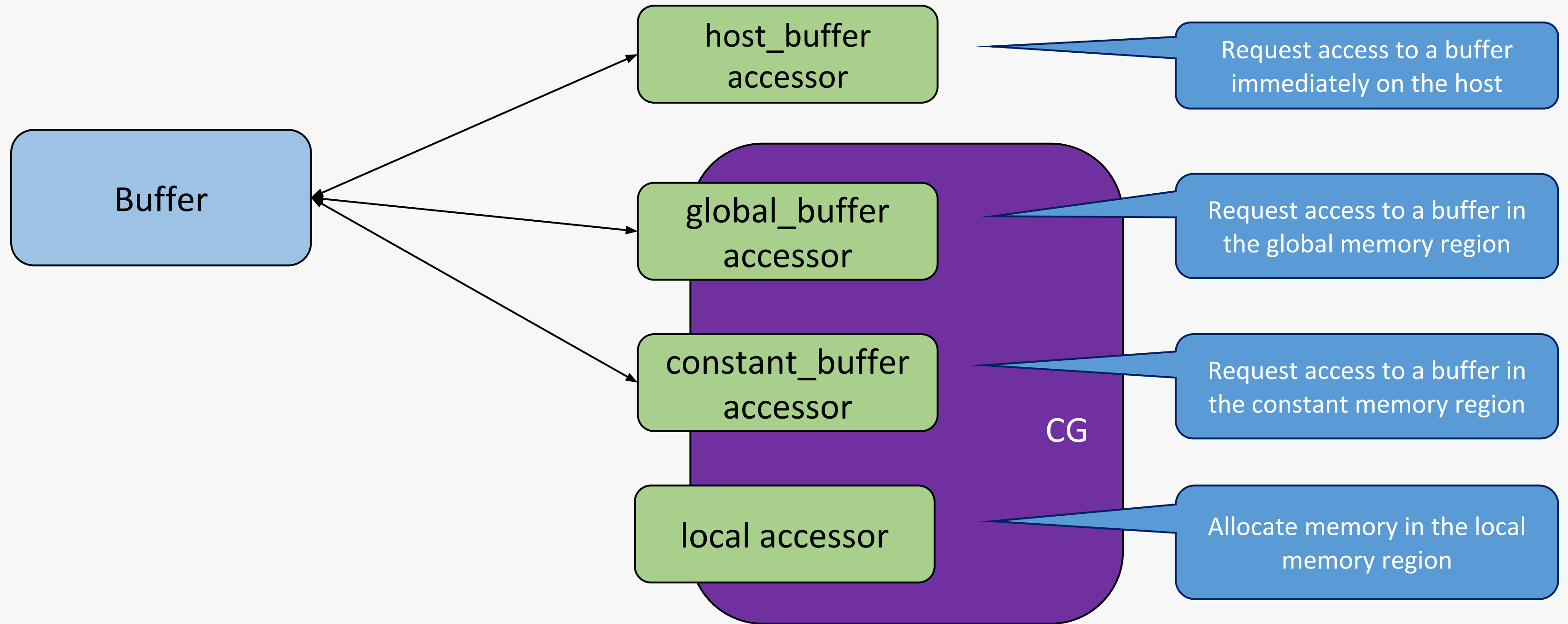


© 2019 Codeplay Software Ltd.





© 2019 Codeplay Software Ltd.



© 2019 Codeplay Software Ltd.

Benefits of data dependency task graphs

- Allows you to describe your tasks in terms of relationships
  - Removes the need to en-queue explicit copies
  - Removes the need for complex event handling

- Allows the runtime to make data movement optimizations
  - Preemptively copy data to a device before kernels are executed
  - Avoid unnecessarily copying data back to the host after execution on a device

© 2019 Codeplay Software Ltd.

Agenda

Emergent hardware for AI in automotive

Overview of OpenCL/SYCL programming model

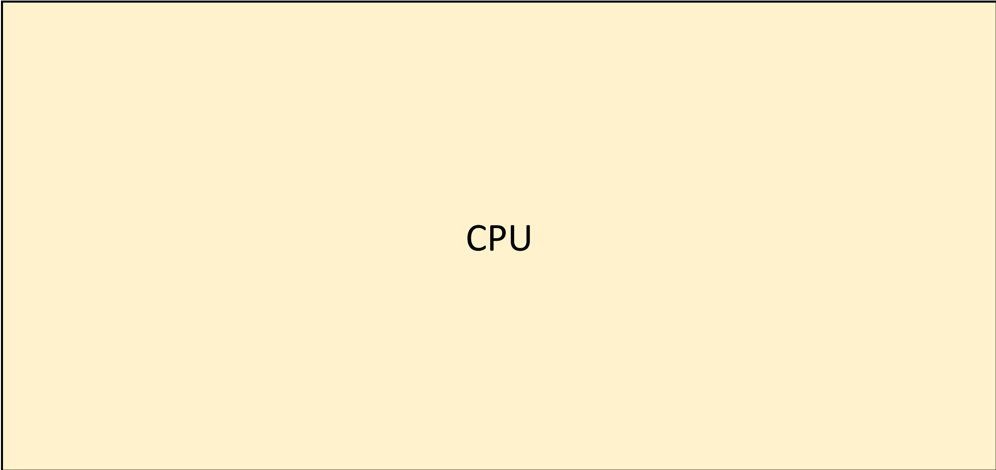
Mapping typical hardware to the OpenCL/SYCL programming model

The Renesas R-Car architecture

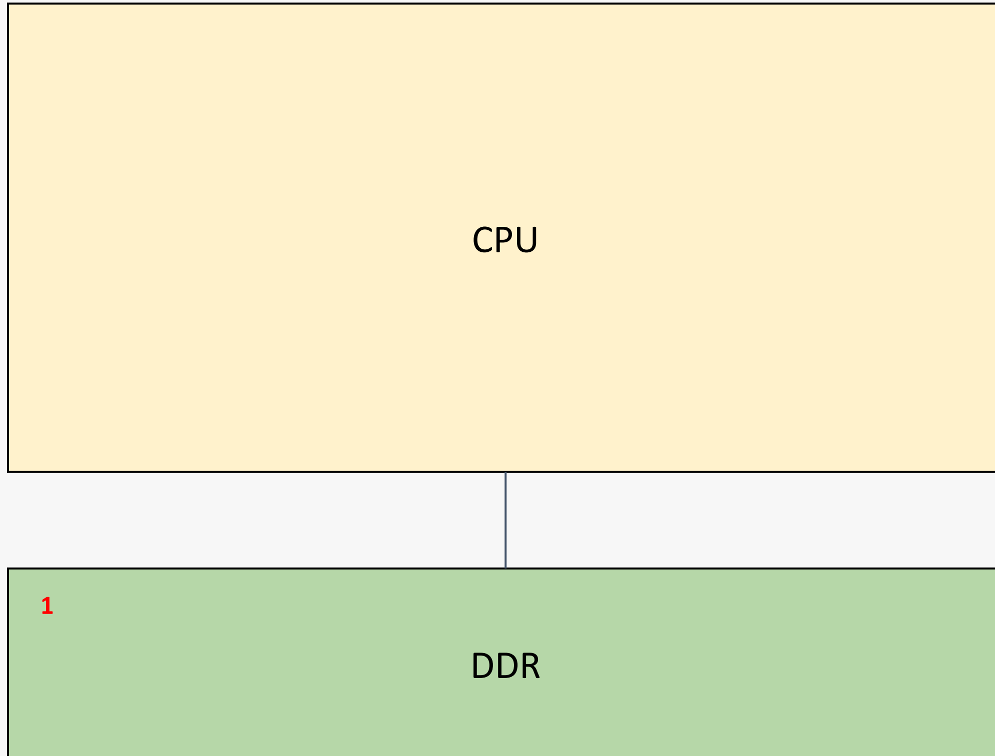
Extending OpenCL & SYCL for R-Car

Optimising machine learning algorithms using R-Car

© 2019 Codeplay Software Ltd.

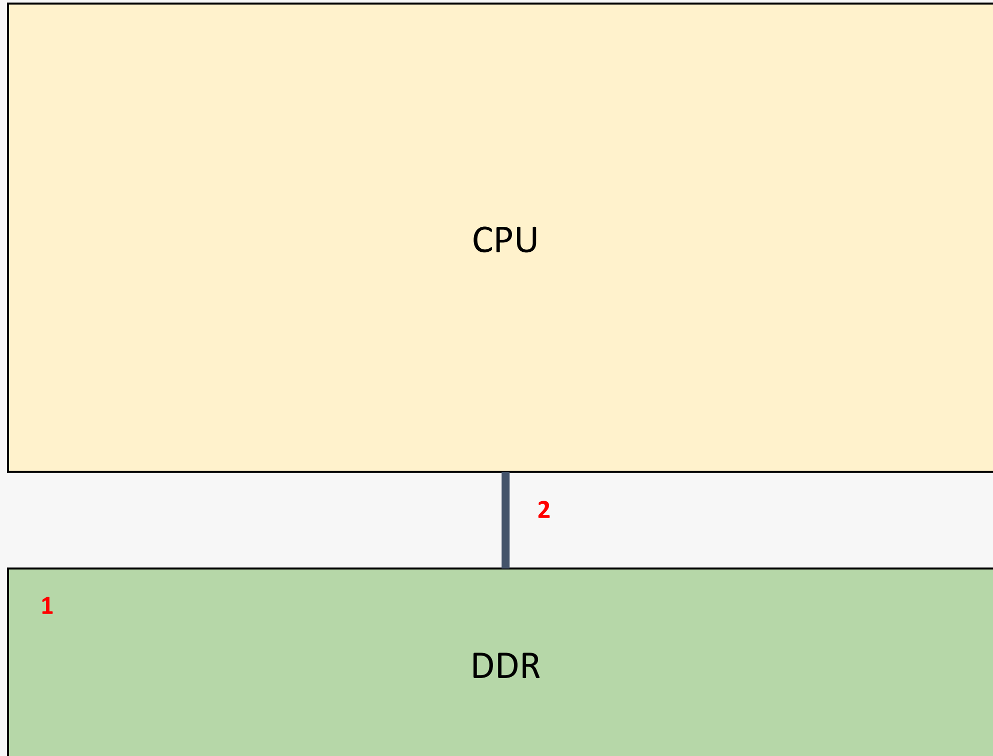


© 2019 Codeplay Software Ltd.



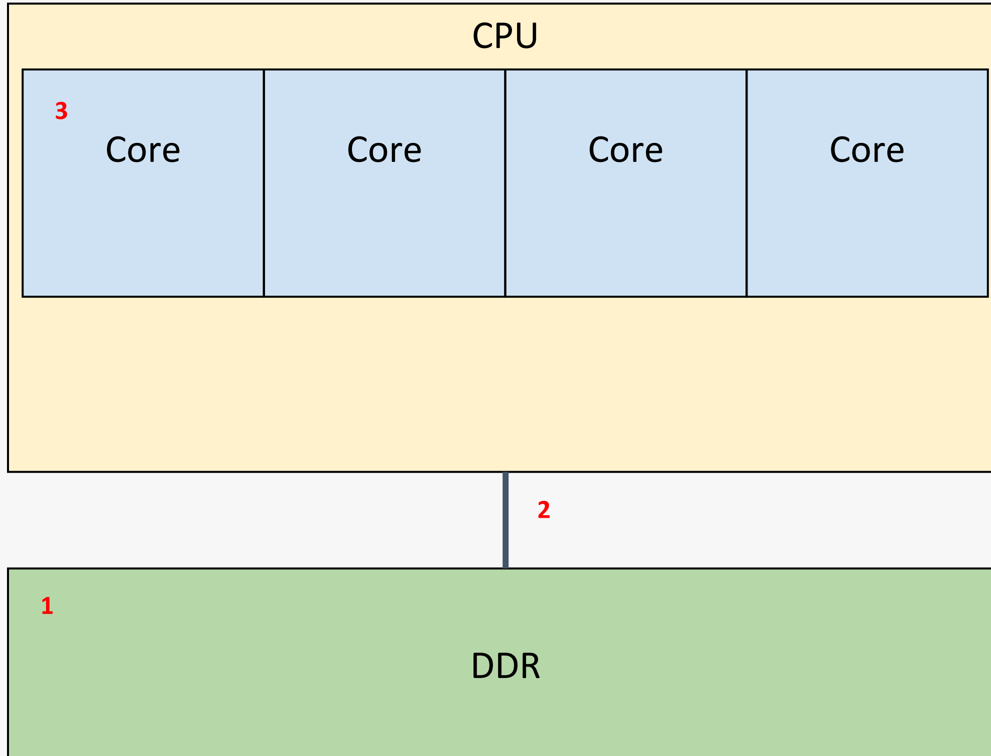
1. A CPU has a region of dedicated memory

© 2019 Codeplay Software Ltd.



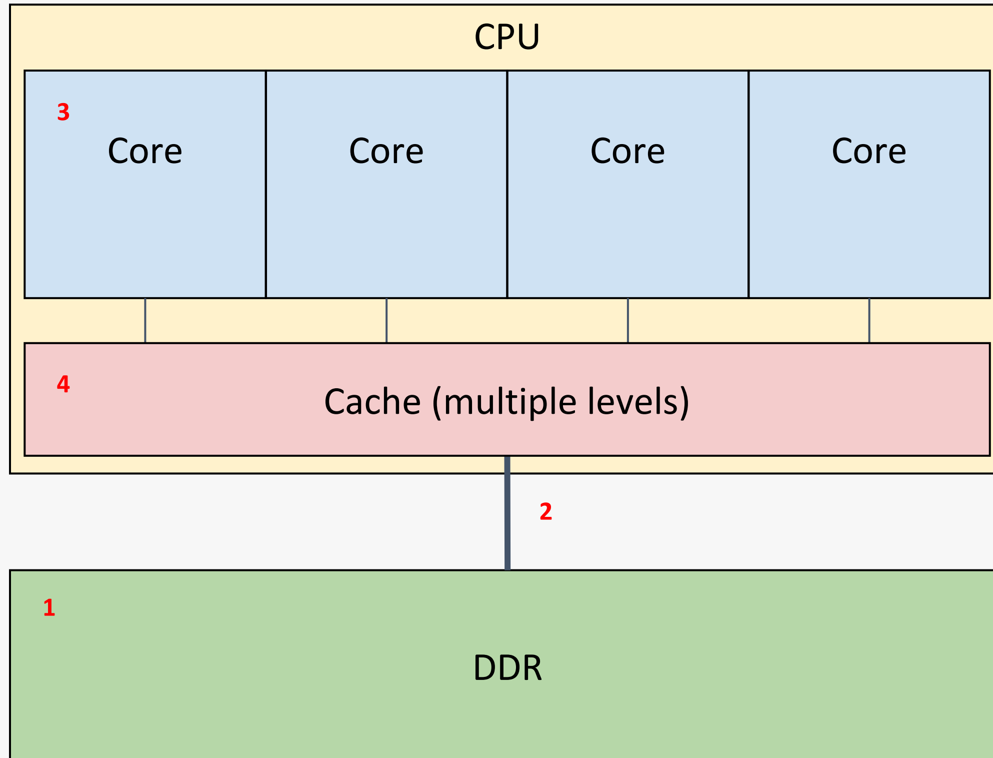
1. A CPU has a region of dedicated memory
2. CPU memory is connected to the CPU via a bus

© 2019 Codeplay Software Ltd.



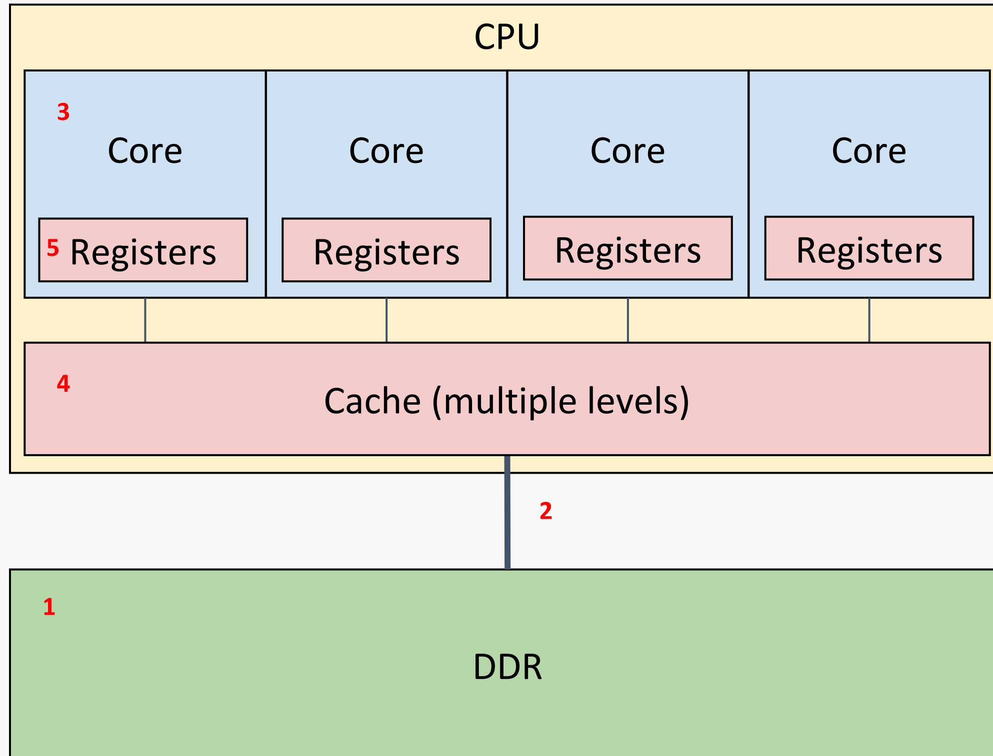
1. A CPU has a region of dedicated memory
2. The CPU memory is connected to the CPU via a bus
3. A CPU has a number of cores

© 2019 Codeplay Software Ltd.



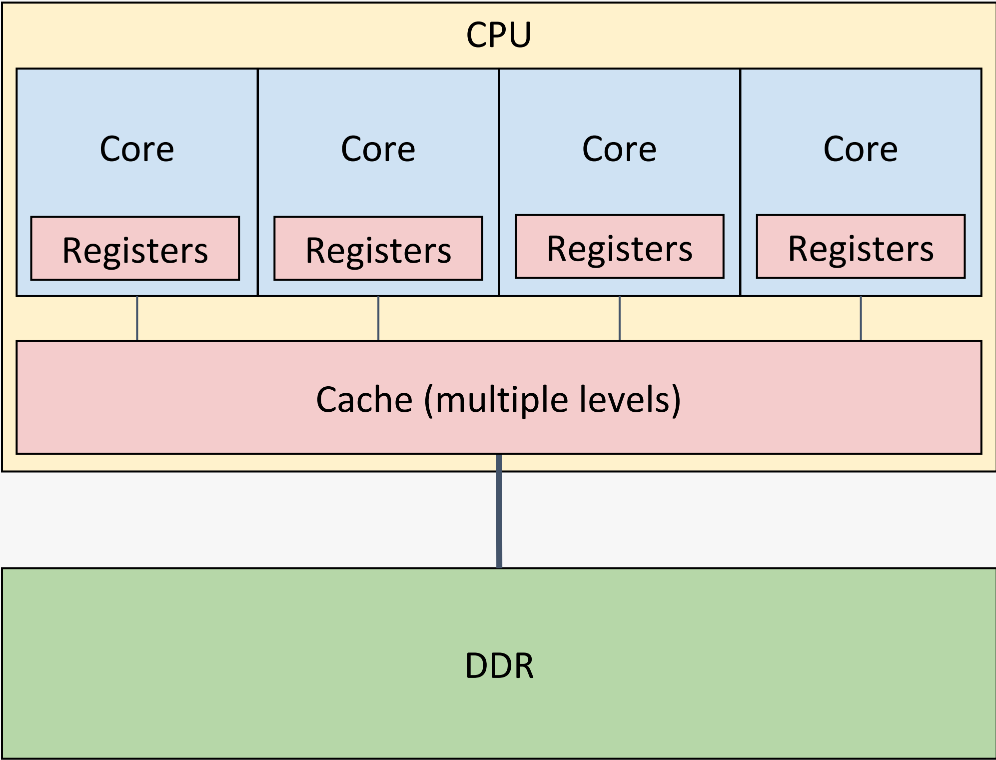
1. A CPU has a region of dedicated memory
2. The CPU memory is connected to the CPU via a bus
3. A CPU has a number of cores
4. A CPU has a number of caches of different levels

© 2019 Codeplay Software Ltd.

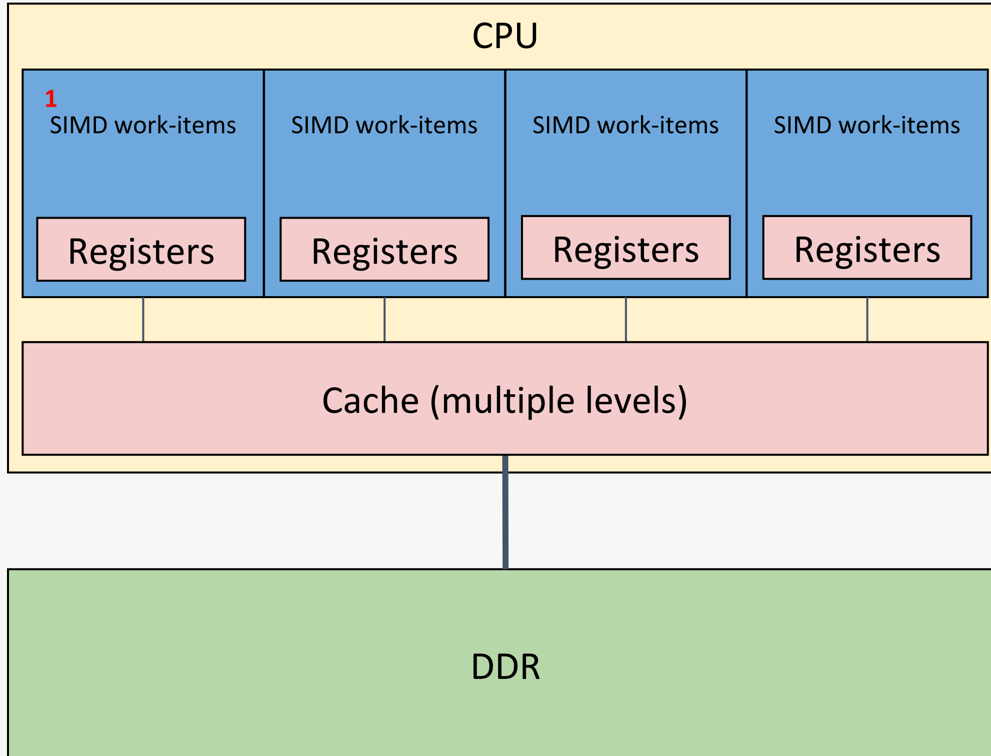


1. A CPU has a region of dedicated memory
2. The CPU memory is connected to the CPU via a bus
3. A CPU has a number of cores
4. A CPU has a number of caches of different levels
5. Each CPU core has dedicated registers

© 2019 Codeplay Software Ltd.

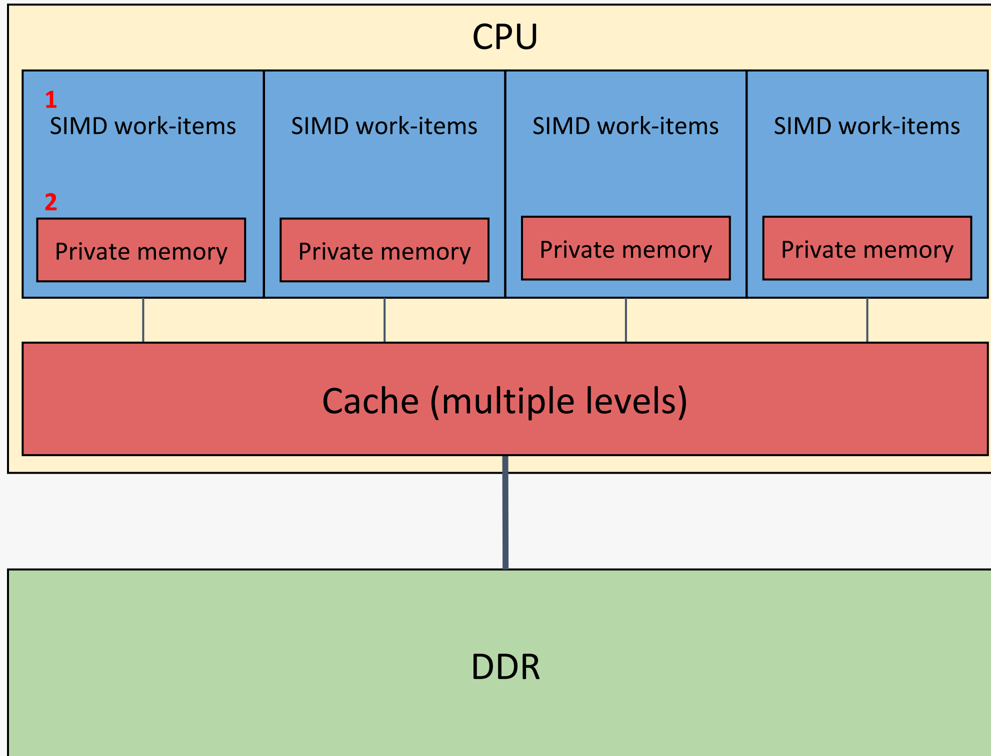


© 2019 Codeplay Software Ltd.



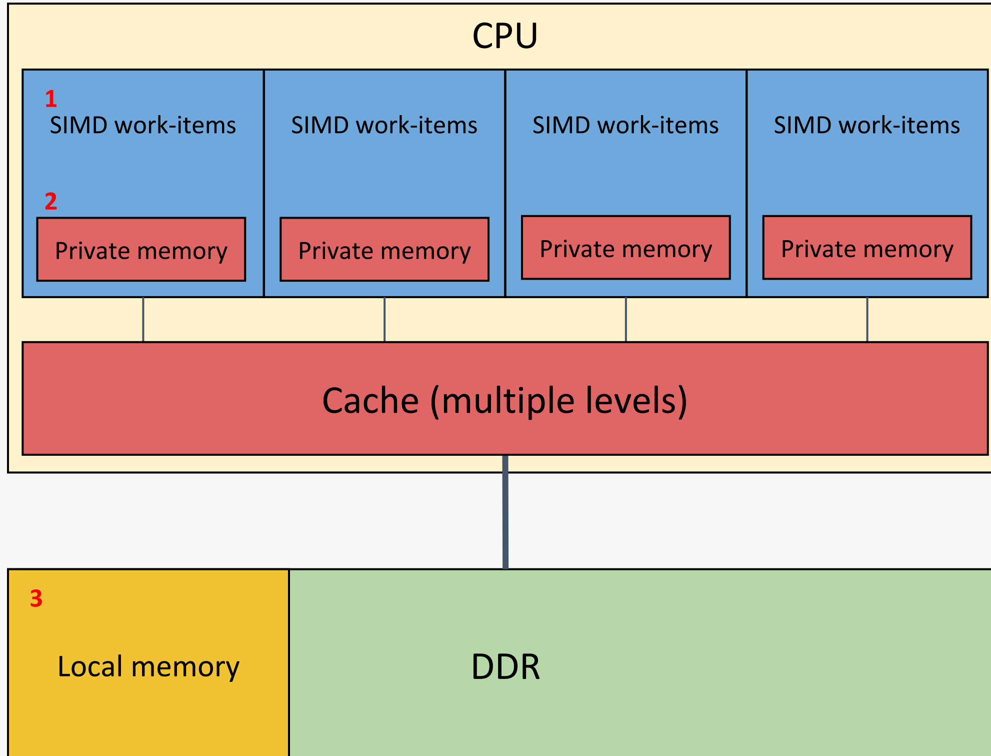
1. Lanes of the CPU core SIMD instructions are mapped to work-items

© 2019 Codeplay Software Ltd.



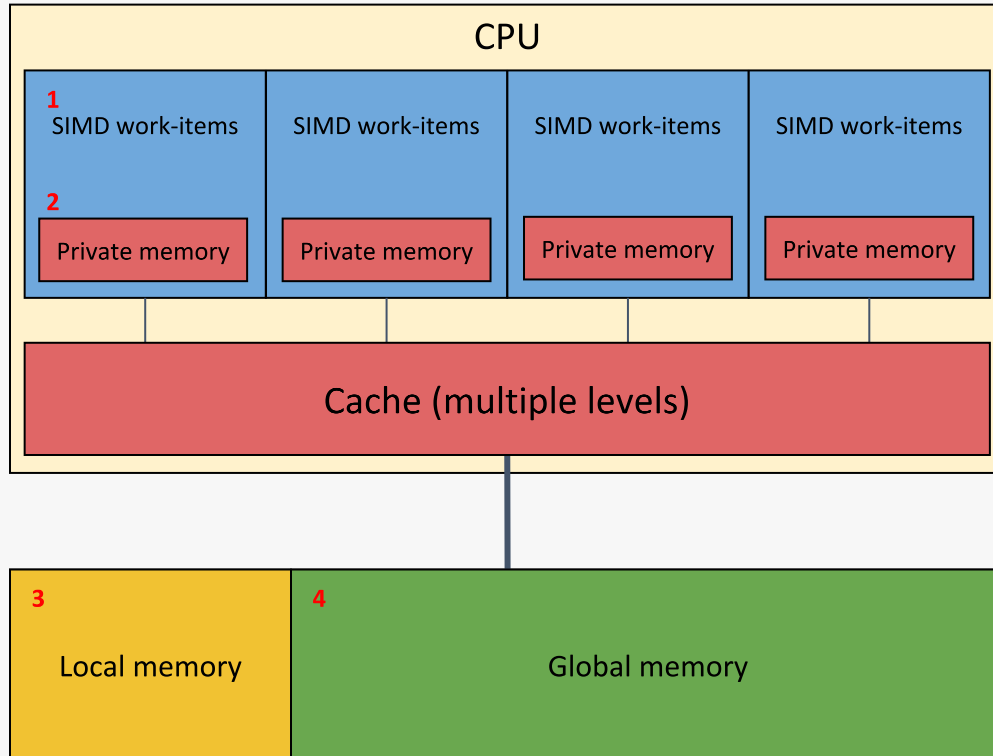
1. Lanes of the CPU core SIMD instructions are mapped to work-items
2. CPU registers and their associated caches are mapped to private memory

© 2019 Codeplay Software Ltd.



1. Lanes of the CPU core SIMD instructions are mapped to work-items
2. CPU registers and their associated caches are mapped to private memory
3. A section of DDR is mapped to local memory

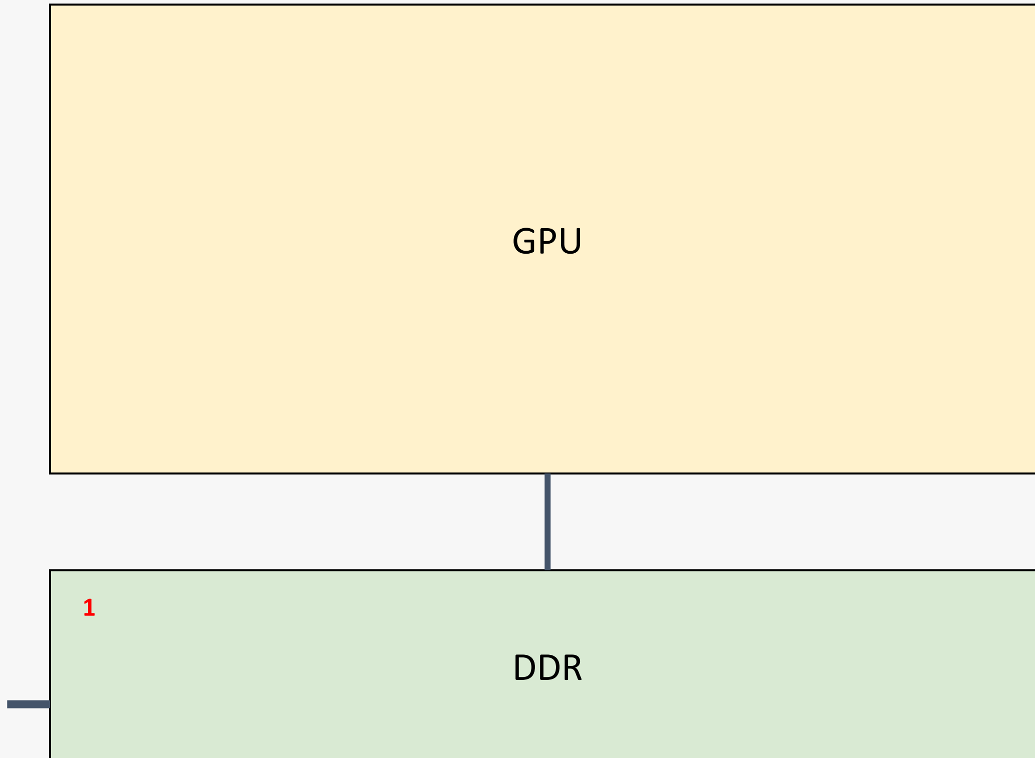
© 2019 Codeplay Software Ltd.



1. Lanes of the CPU core SIMD instructions are mapped to work-items
2. CPU registers and their associated caches are mapped to private memory
3. A section of DDR is mapped to local memory
4. The rest of DDR is mapped to global memory

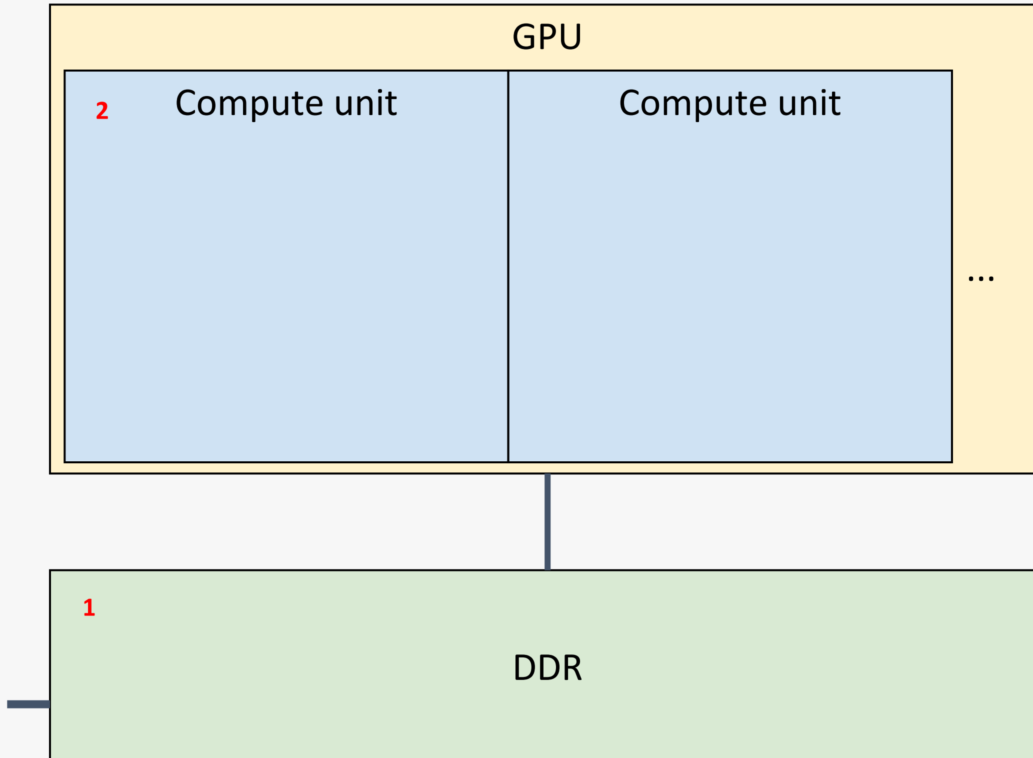
© 2019 Codeplay Software Ltd.

© 2019 Codeplay Software Ltd.



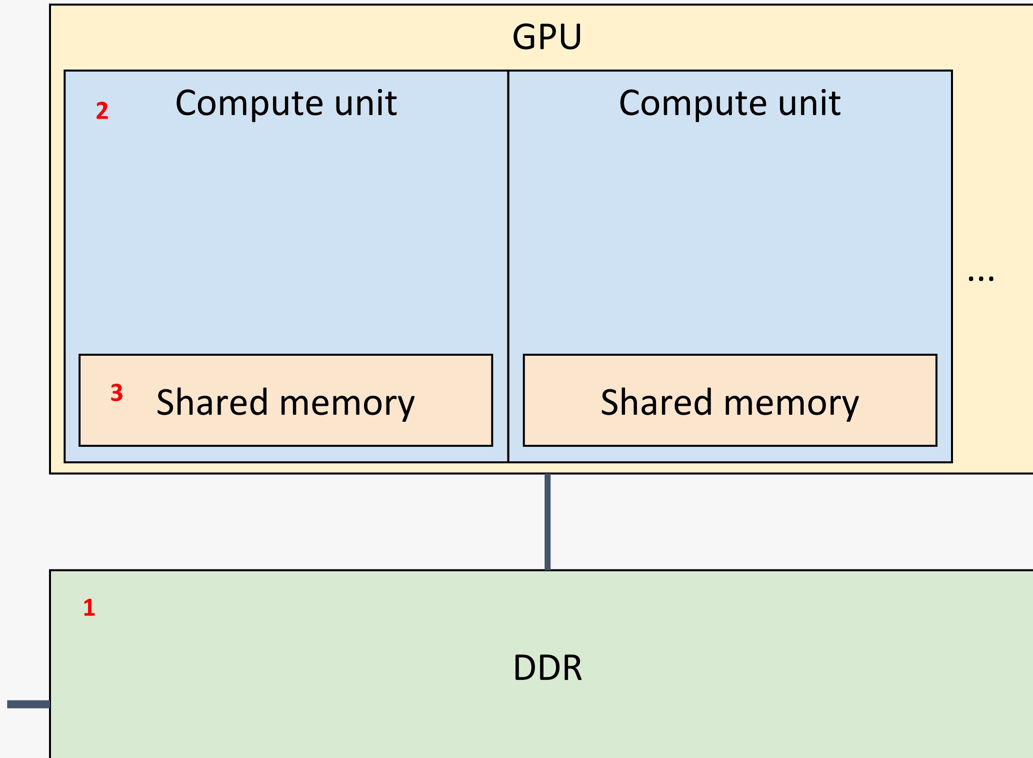
1. A GPU has a region of dedicated DDR memory which is connected to the CPU

© 2019 Codeplay Software Ltd.



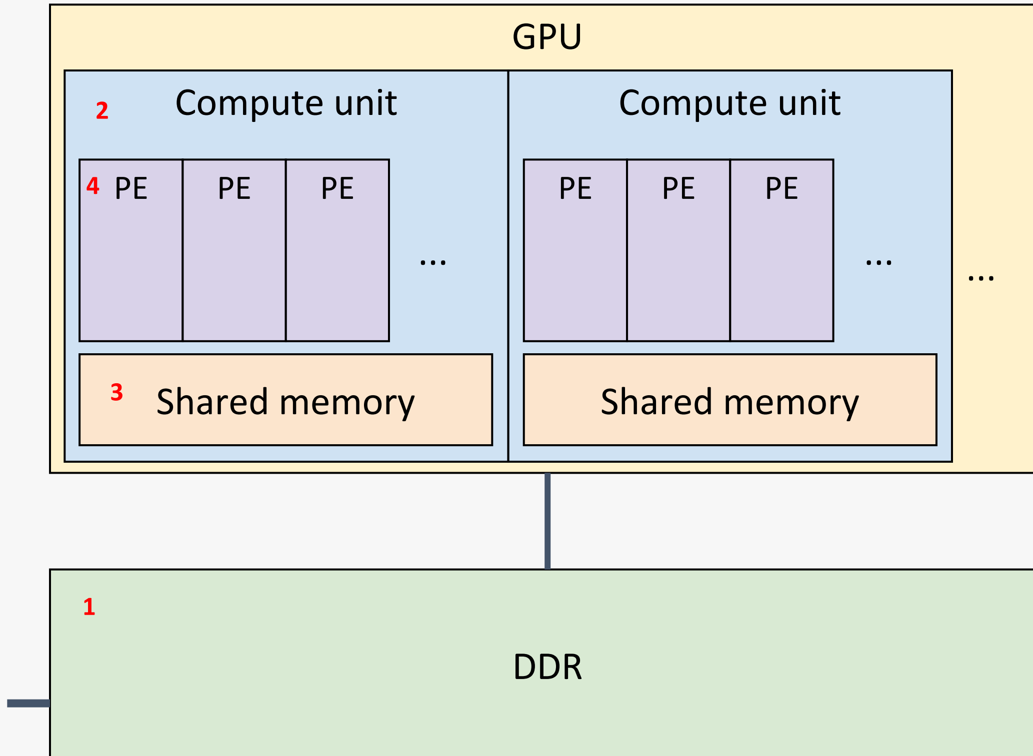
1. A GPU has a region of dedicated DDR memory which is connected to the CPU
2. A GPU is divided into a number of compute units

© 2019 Codeplay Software Ltd.



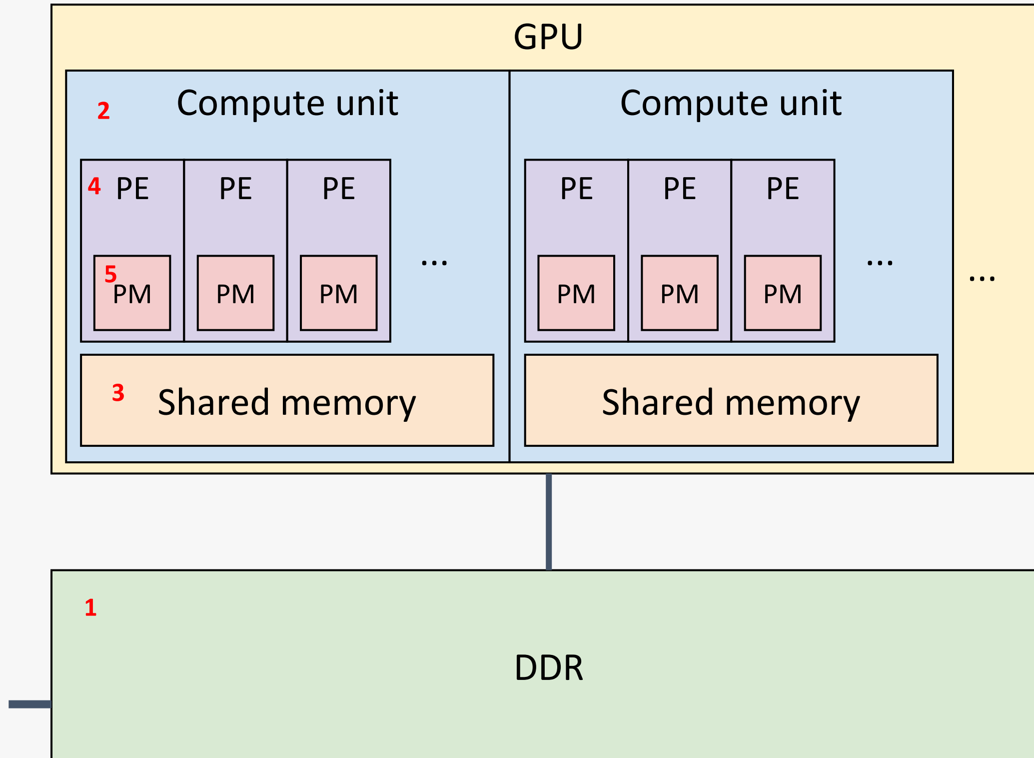
1. A GPU has a region of dedicated DDR memory which is connected to the CPU
2. A GPU is divided into a number of compute units
3. Each compute unit has dedicated shared memory

© 2019 Codeplay Software Ltd.



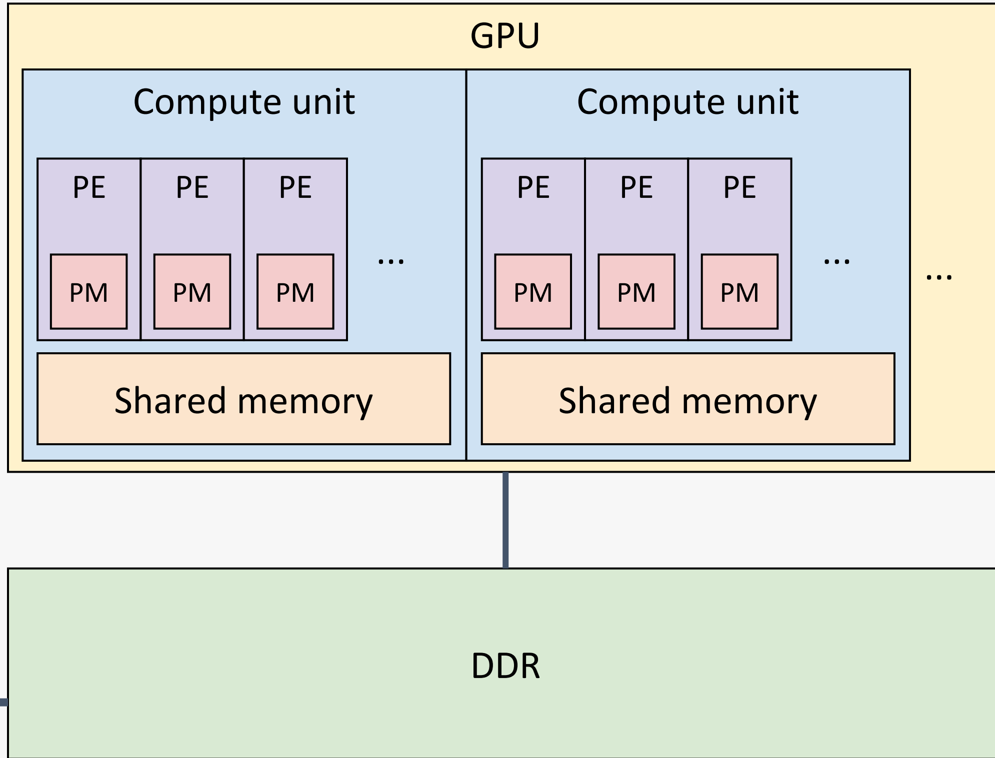
1. A GPU has a region of dedicated DDR memory which is connected to the CPU
2. A GPU is divided into a number of compute units
3. Each compute unit has dedicated shared memory
4. Each compute unit has a number of processing elements

© 2019 Codeplay Software Ltd.

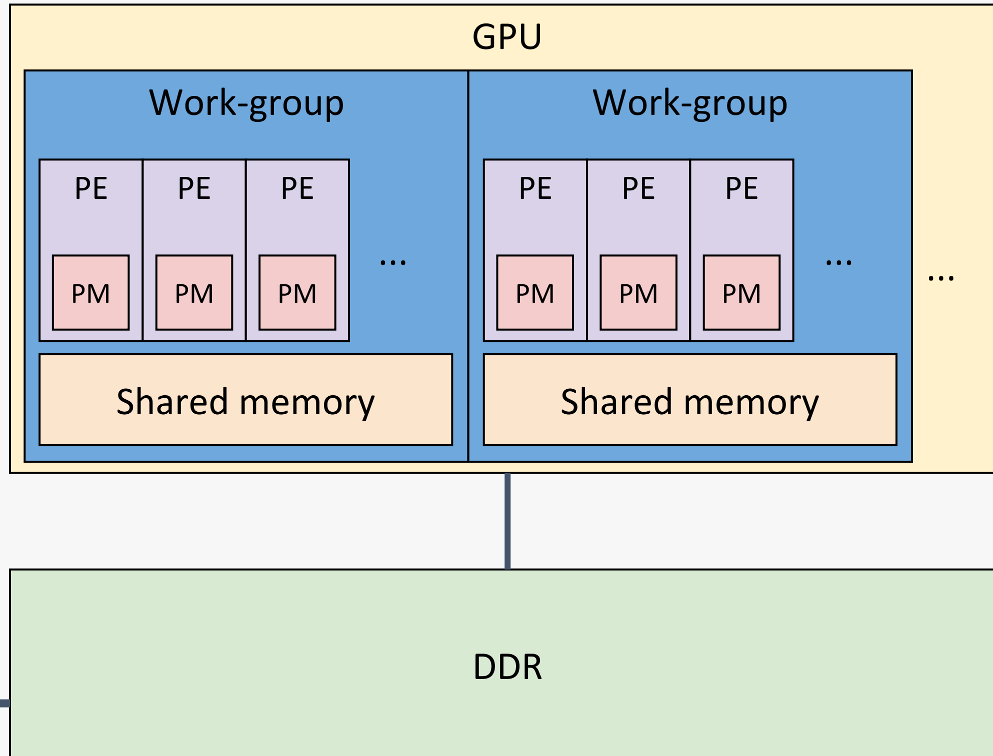


1. A GPU has a region of dedicated DDR memory which is connected to the CPU
2. A GPU is divided into a number of compute units
3. Each compute unit has dedicated shared memory
4. Each compute unit has a number of processing elements
5. Each processing element has dedicated processing element local memory

© 2019 Codeplay Software Ltd.

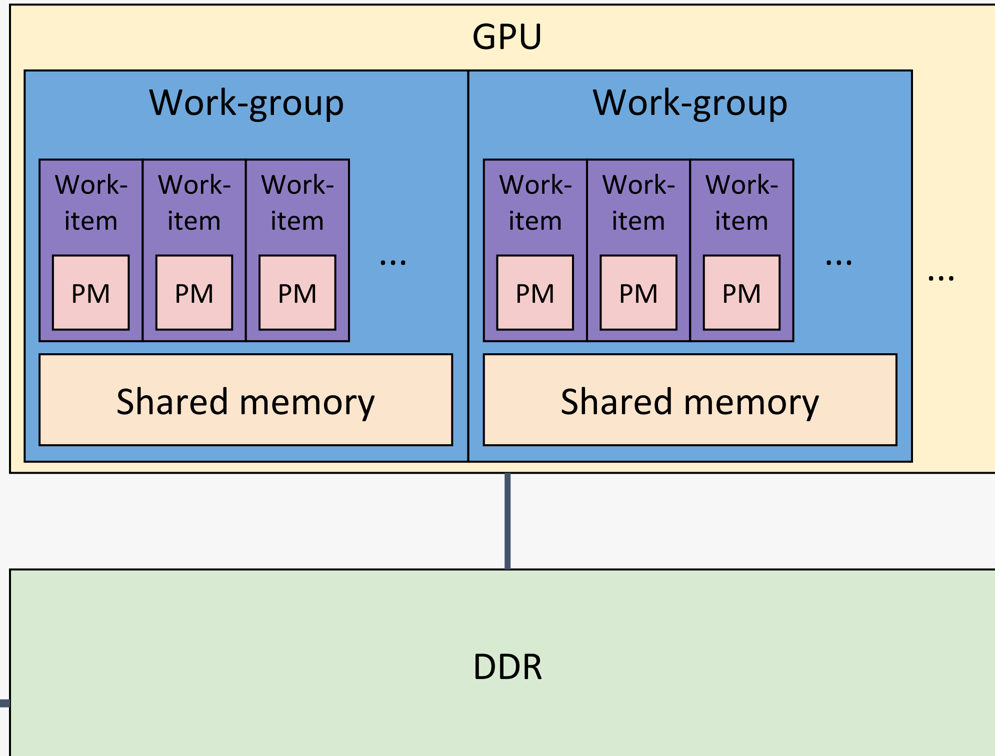


© 2019 Codeplay Software Ltd.



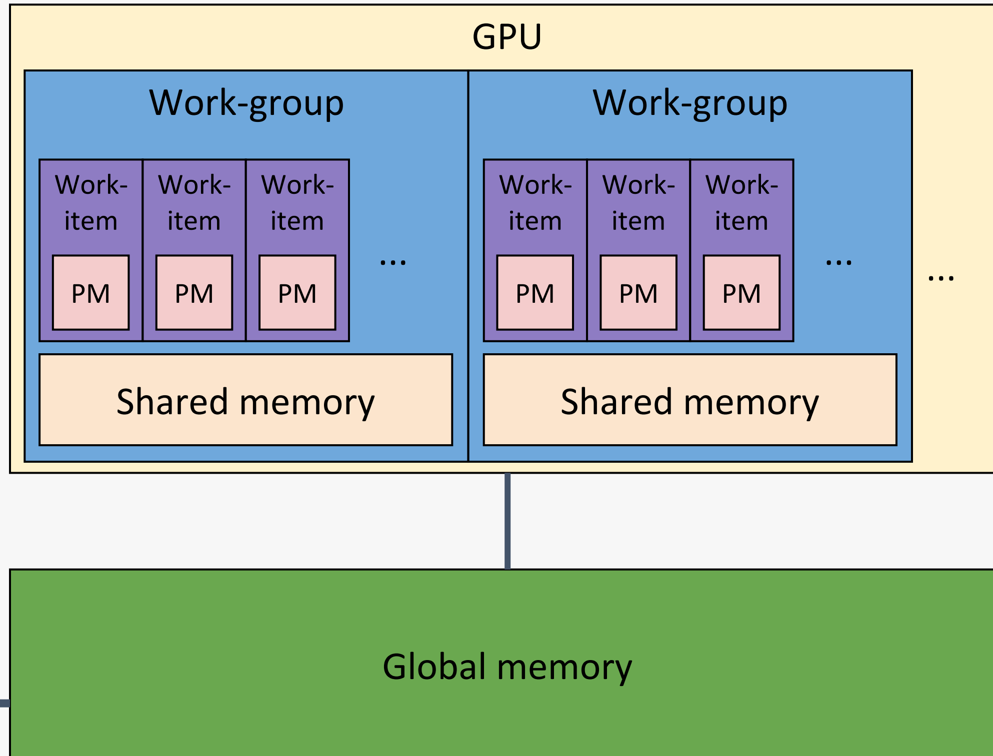
Compute units on are mapped to the optimal work-group size

© 2019 Codeplay Software Ltd.



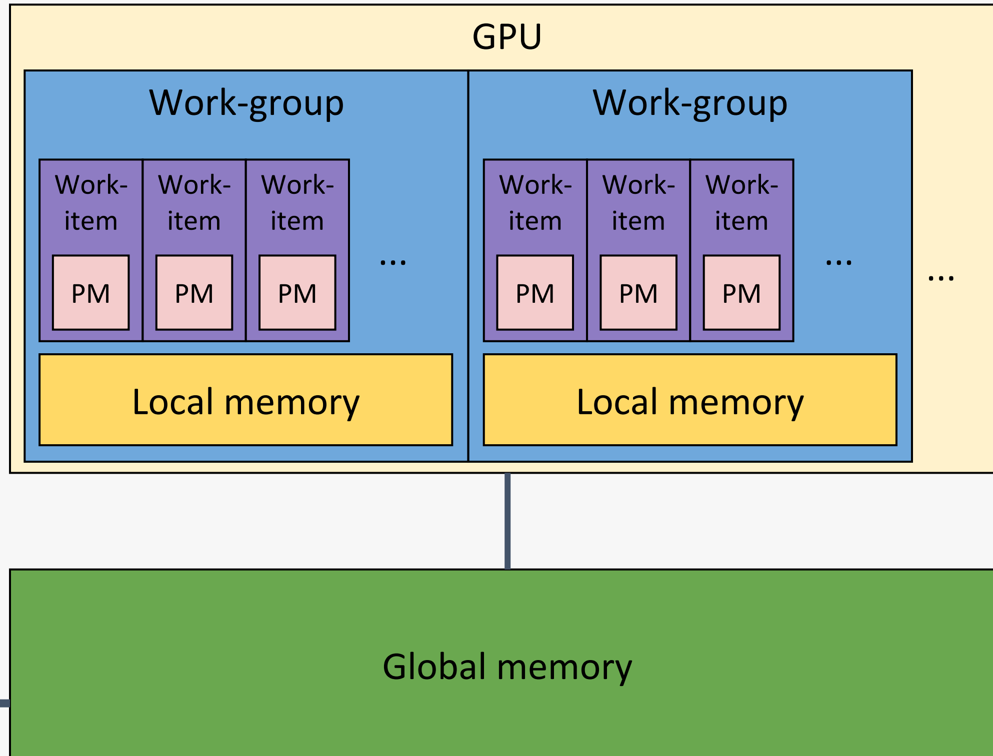
Processing elements on are mapped to work-items

© 2019 Codeplay Software Ltd.



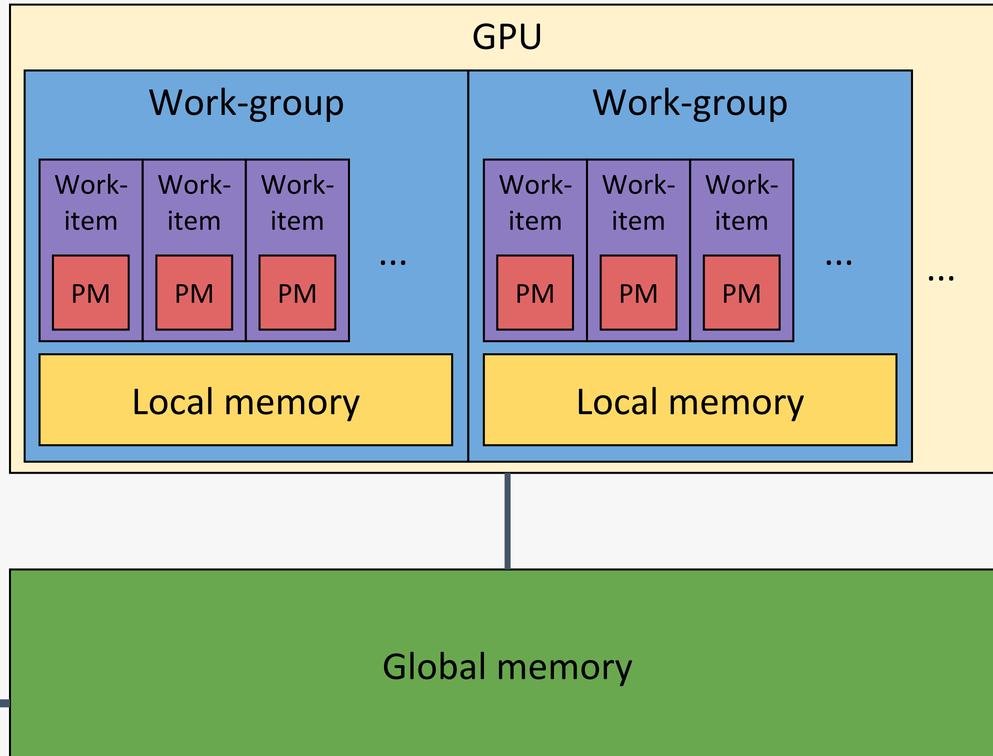
DDR memory is mapped to global memory

© 2019 Codeplay Software Ltd.



Compute unit shared memory is mapped to local memory

© 2019 Codeplay Software Ltd.



Processing element local memory is mapped to private memory

© 2019 Codeplay Software Ltd.

Agenda

Emergent hardware for AI in automotive

Overview of OpenCL/SYCL programming model

Mapping typical hardware to the OpenCL/SYCL programming model

The Renesas R-Car architecture

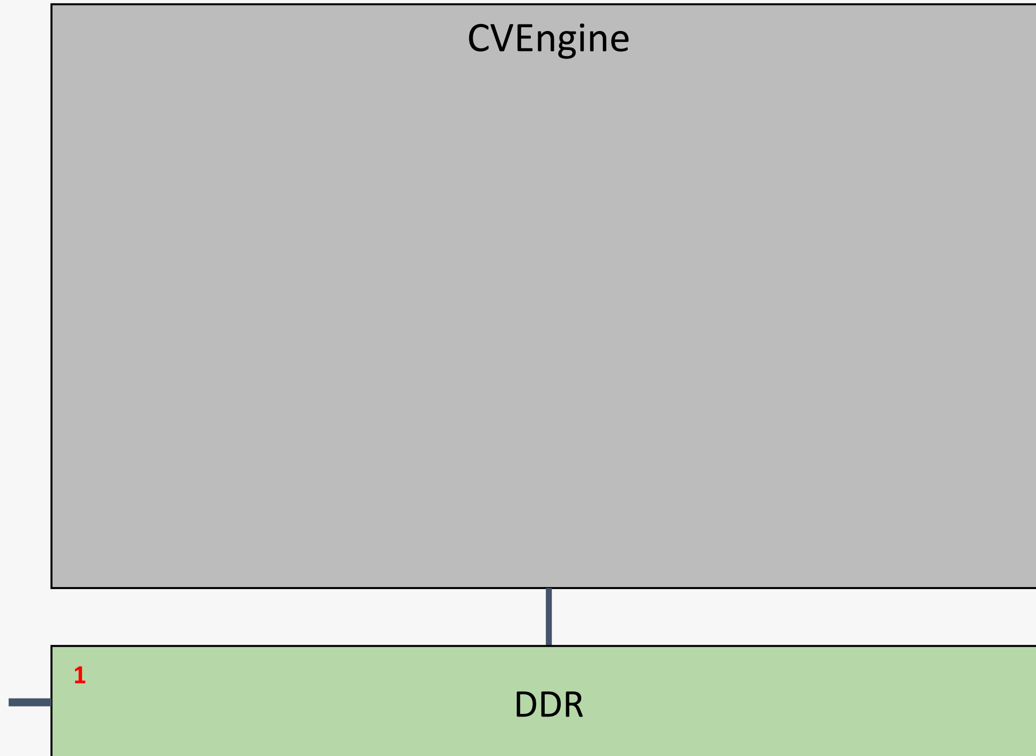
Extending OpenCL & SYCL for R-Car

Optimising machine learning algorithms using R-Car

© 2019 Codeplay Software Ltd.

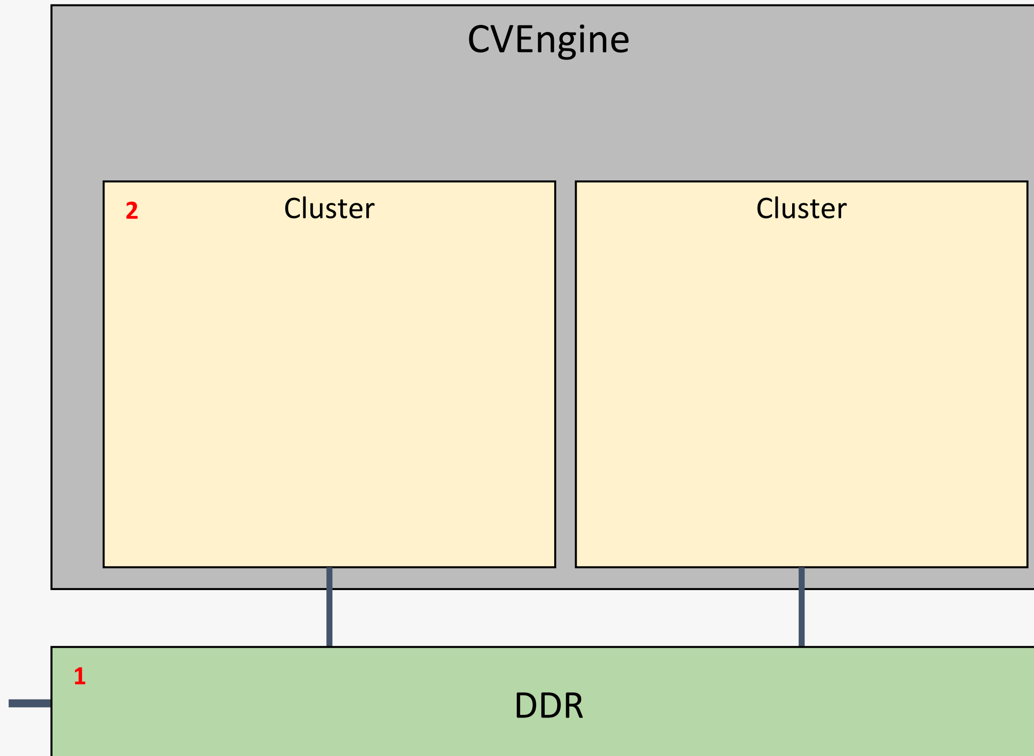
CVEngine

© 2019 Codeplay Software Ltd.



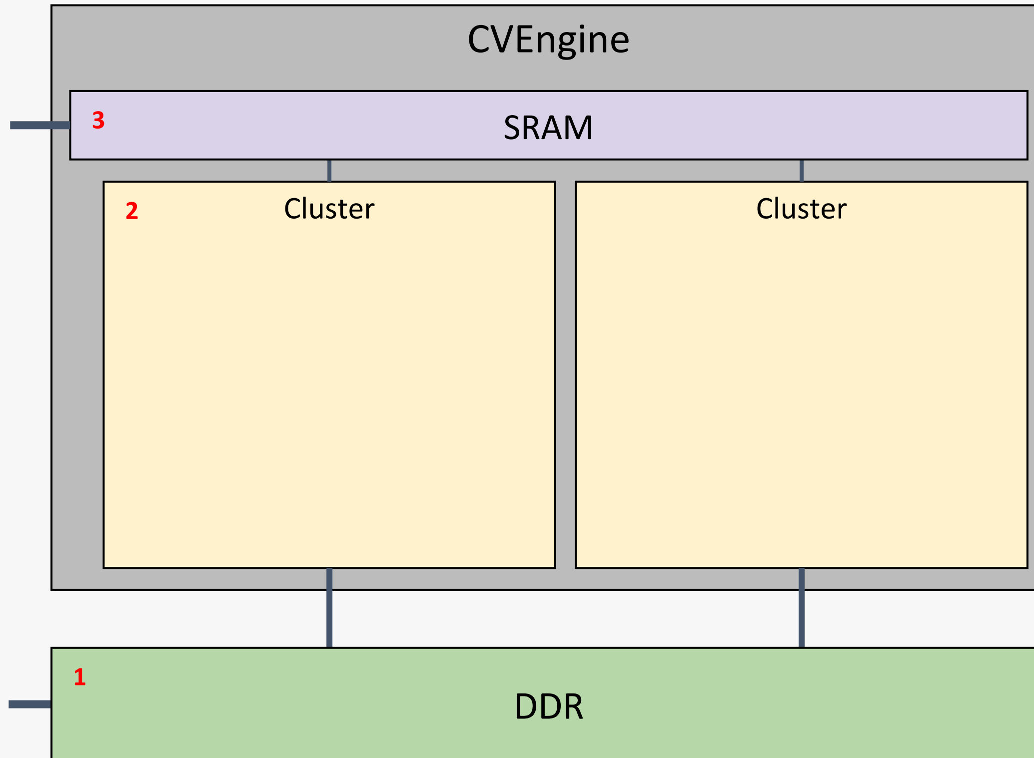
1. The CVEngine has is connected to off-chip DDR which is connected to the CPU

© 2019 Codeplay Software Ltd.



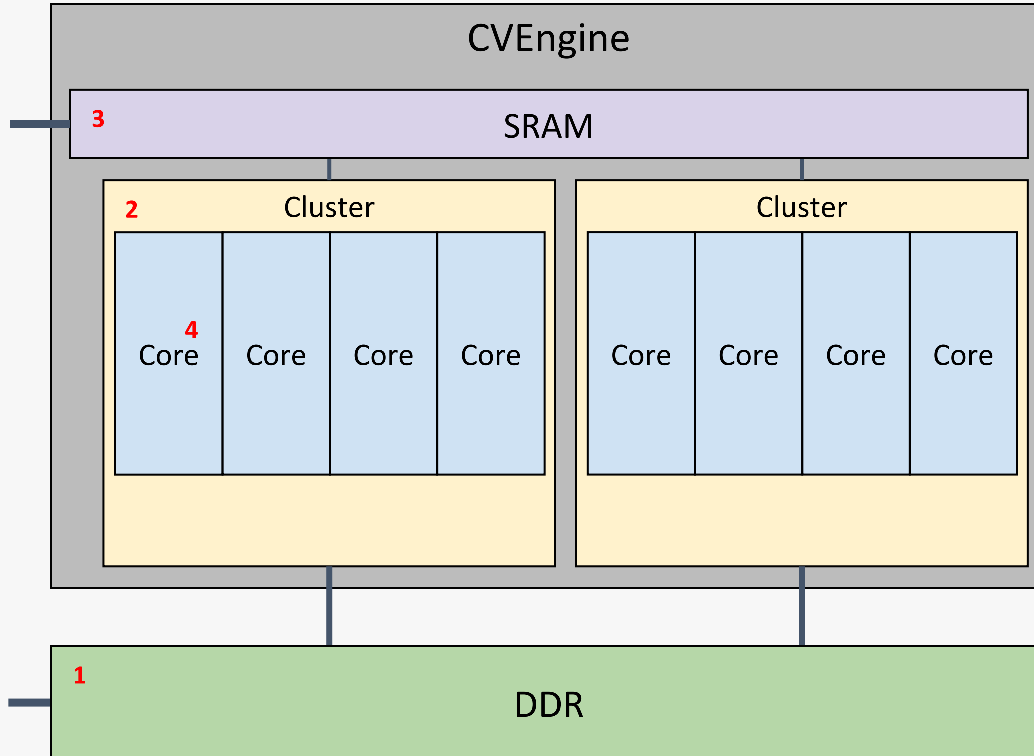
1. The CVEngine has is connected to off-chip DDR which is connected to the CPU
2. The CVEngine has a number of clusters

© 2019 Codeplay Software Ltd.



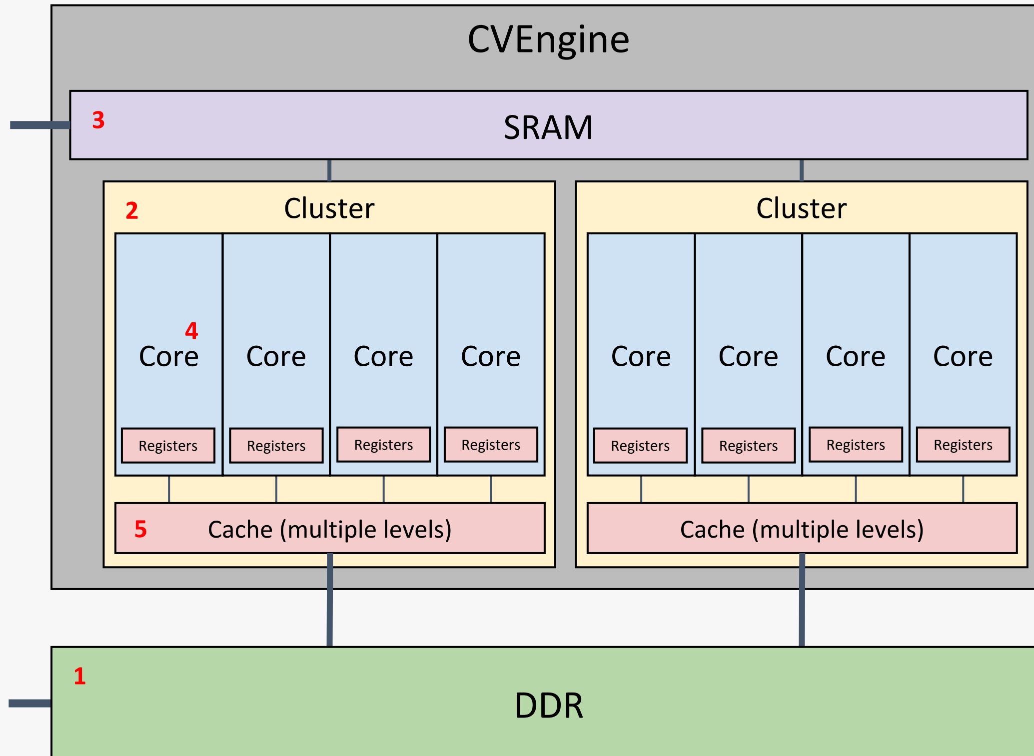
1. The CVEngine has is connected to off-chip DDR which is connected to the CPU
2. The CVEngine has a number of clusters
3. The CVEngine has a region of on-chip SRAM, also connected to the CPU

© 2019 Codeplay Software Ltd.



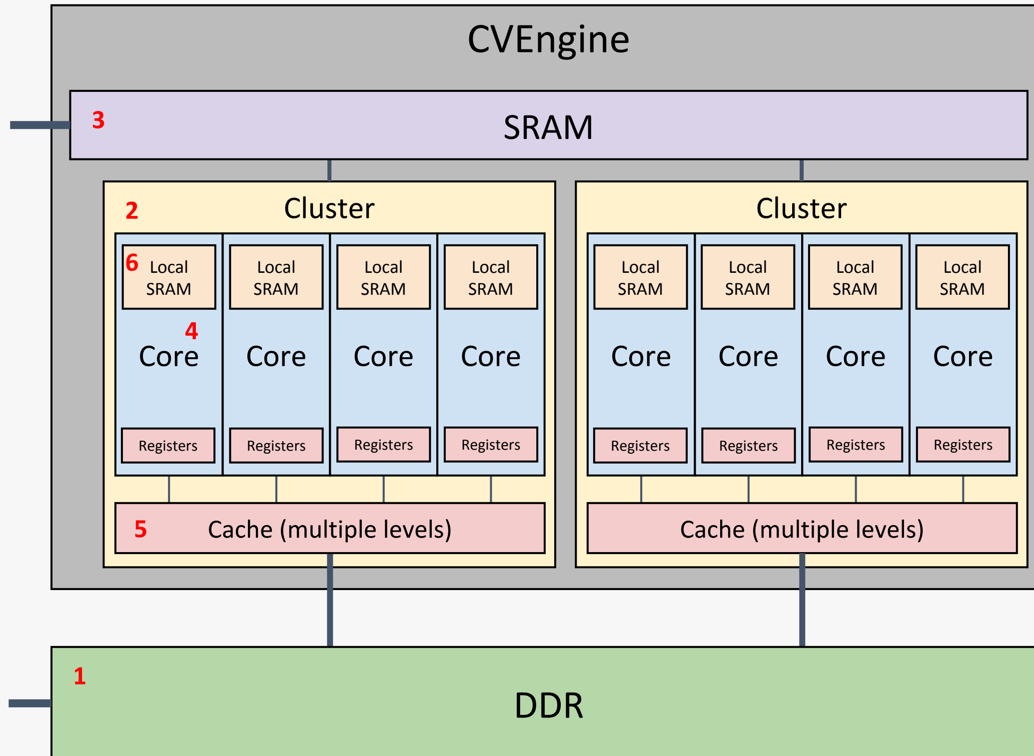
1. The CVEngine has is connected to off-chip DDR which is connected to the CPU
2. The CVEngine has a number of clusters
3. The CVEngine has a region of on-chip SRAM, also connected to the CPU
4. Each cluster has 4 cores each with a number of processing elements

© 2019 Codeplay Software Ltd.



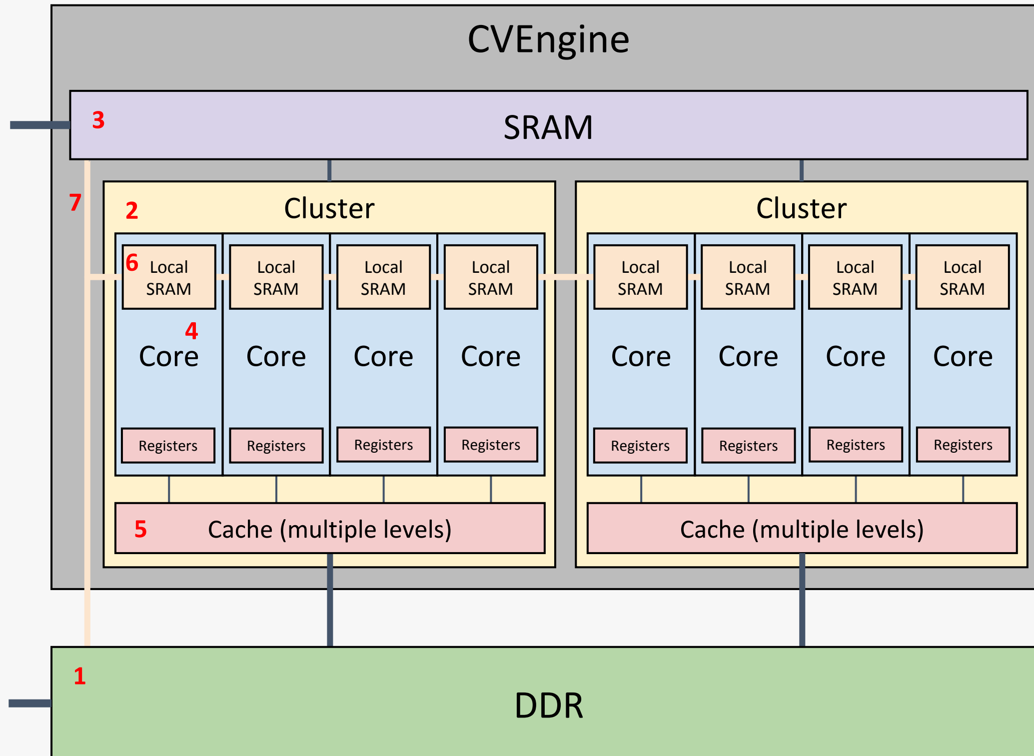
1. The CVEngine has is connected to off-chip DDR which is connected to the CPU
2. The CVEngine has a number of clusters
3. The CVEngine has a region of on-chip SRAM, also connected to the CPU
4. Each cluster has 4 cores each with a number of processing elements
5. Each core has dedicated registers and can access DDR memory via caches

© 2019 Codeplay Software Ltd.



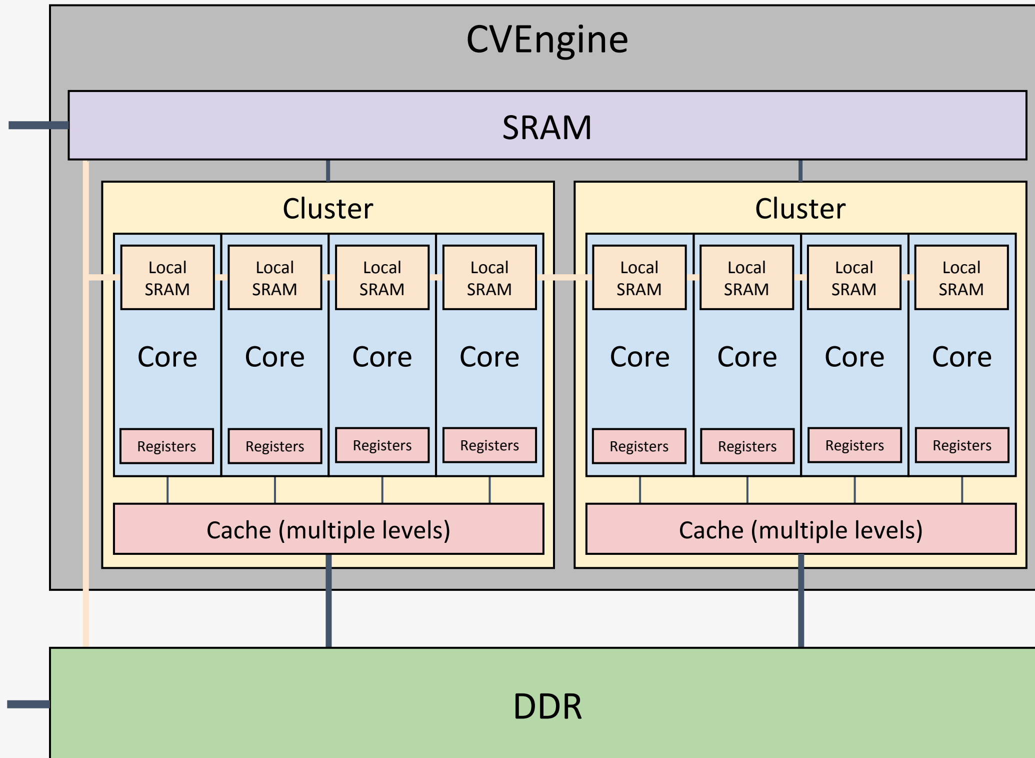
1. The CVEngine has is connected to off-chip DDR which is connected to the CPU
2. The CVEngine has a number of clusters
3. The CVEngine has a region of on-chip SRAM, also connected to the CPU
4. Each cluster has 4 cores each with a number of processing elements
5. Each core has dedicated registers and can access DDR memory via caches
6. Each core also has dedicated local SRAM

© 2019 Codeplay Software Ltd.

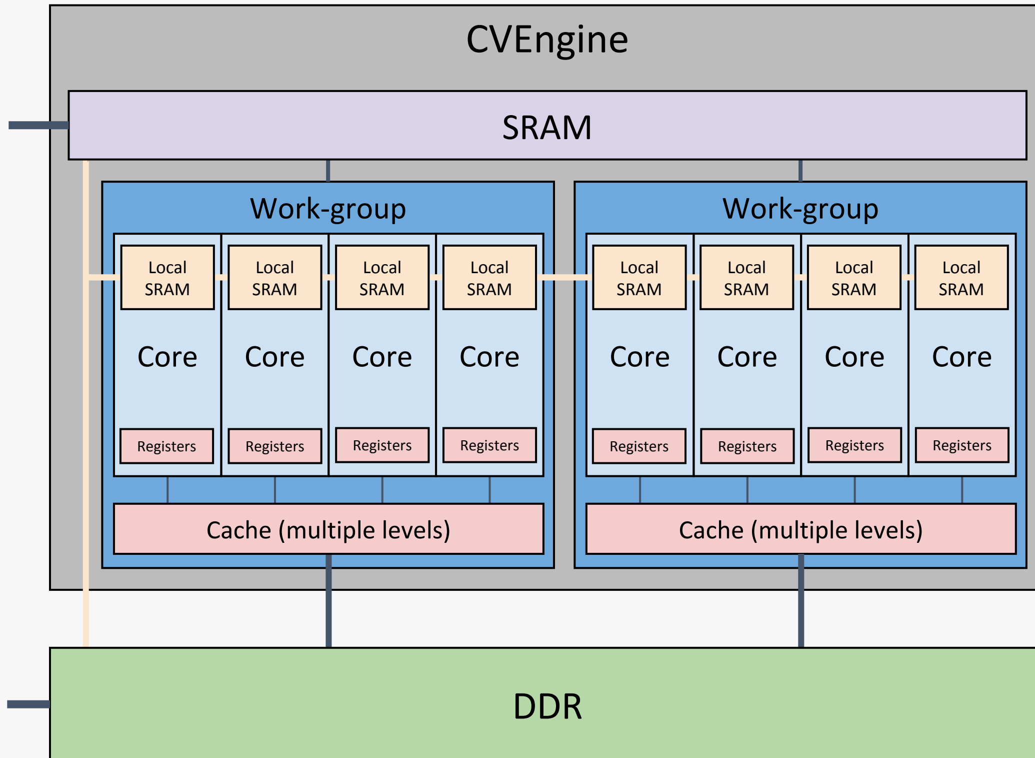


1. The CVEngine has is connected to off-chip DDR which is connected to the CPU
2. The CVEngine has a number of clusters
3. The CVEngine has a region of on-chip SRAM, also connected to the CPU
4. Each cluster has 4 cores each with a number of processing elements
5. Each core has dedicated registers and can access DDR memory via caches
6. Each core also has dedicated local SRAM
7. The local SRAM is connected to the on-chip SRAM and DDR via DMA

© 2019 Codeplay Software Ltd.

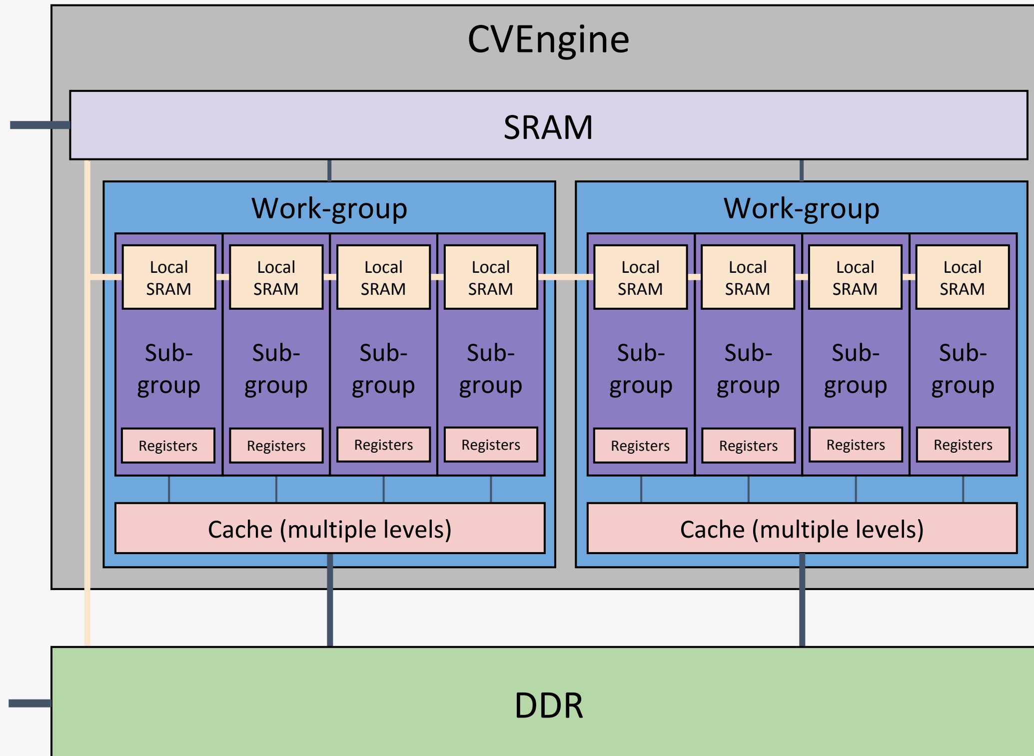


© 2019 Codeplay Software Ltd.



Each cluster maps to the optimal work-group size

© 2019 Codeplay Software Ltd.

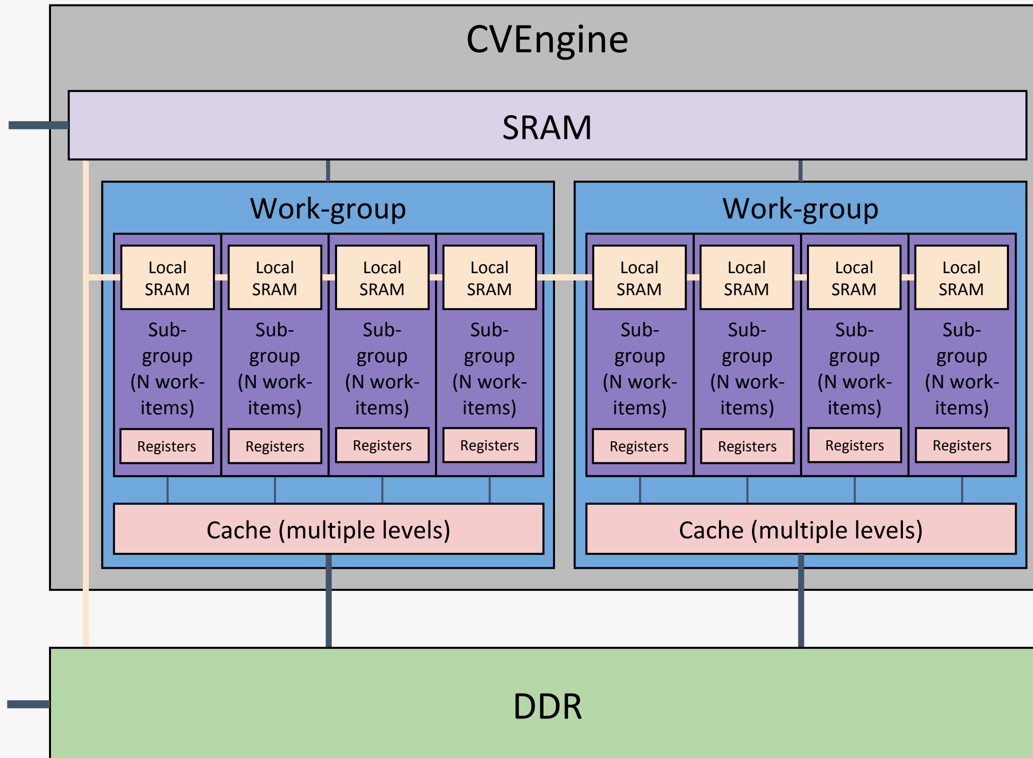


Cores provide an extra level of subdivision within work-groups

So each core maps to a sub-group

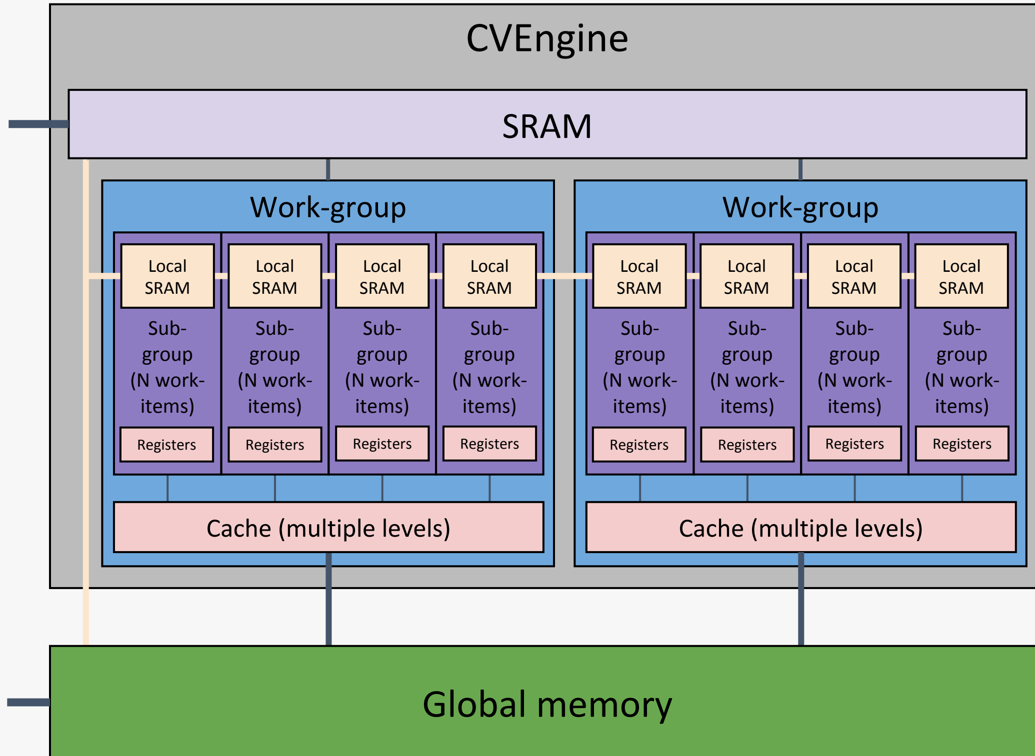
Sub-groups are available in OpenCL 2.x but not yet available in SYCL so this will require an extension

© 2019 Codeplay Software Ltd.



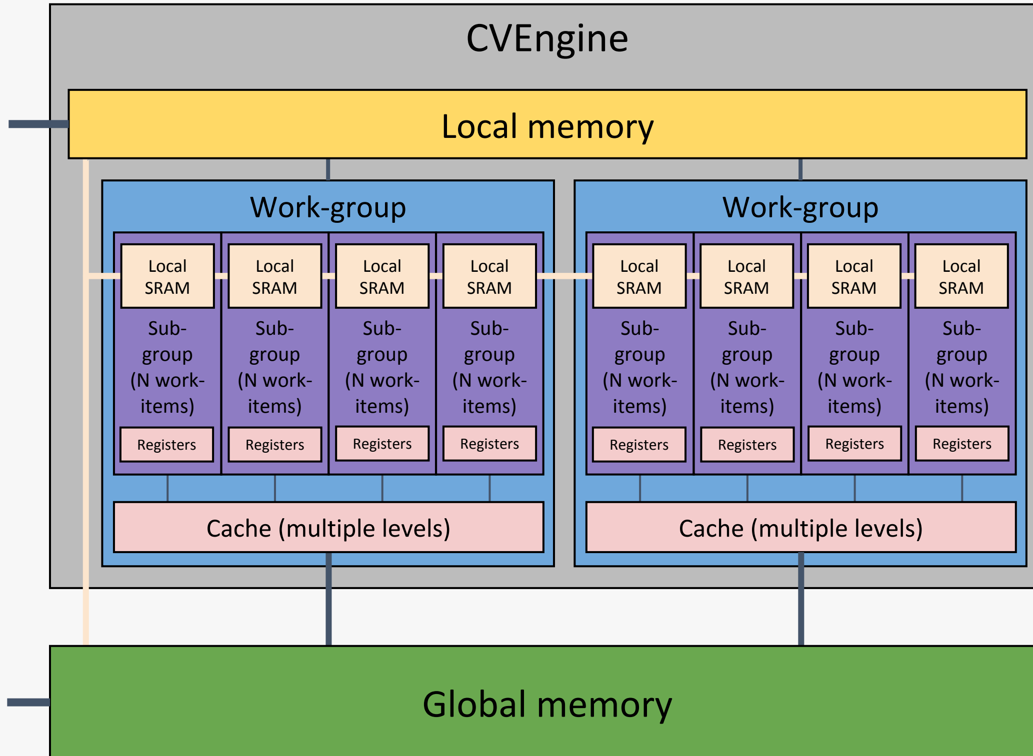
Each processing element within a core maps to a single work-item

© 2019 Codeplay Software Ltd.



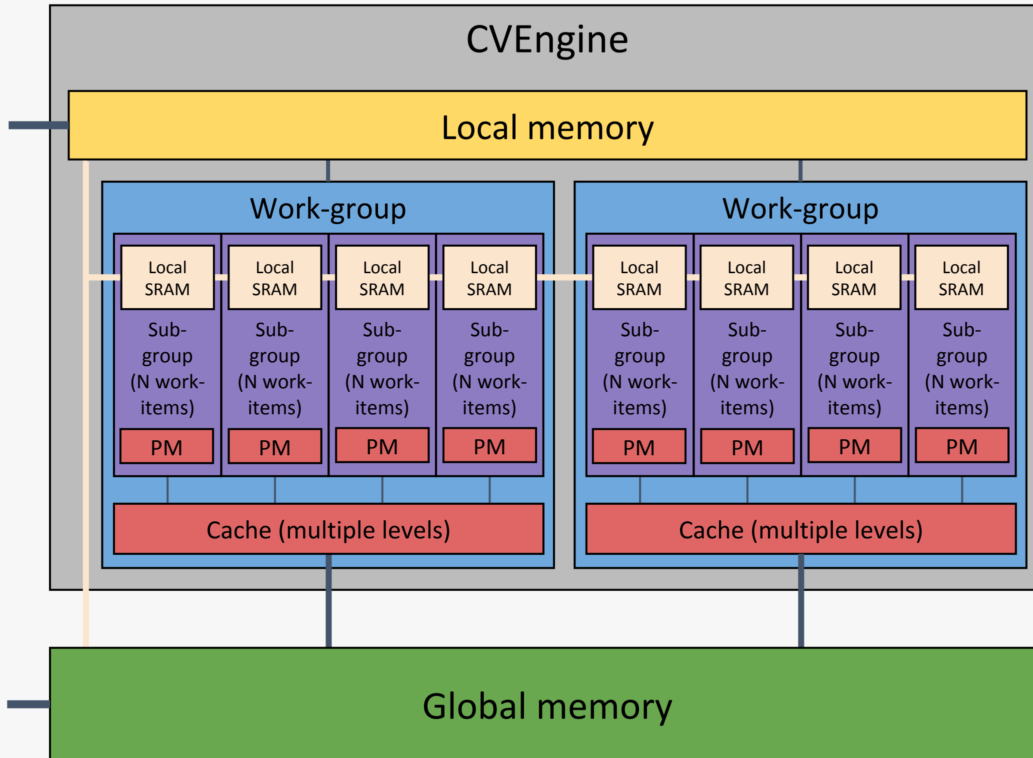
Off-chip DDR memory is mapped to global memory

© 2019 Codeplay Software Ltd.



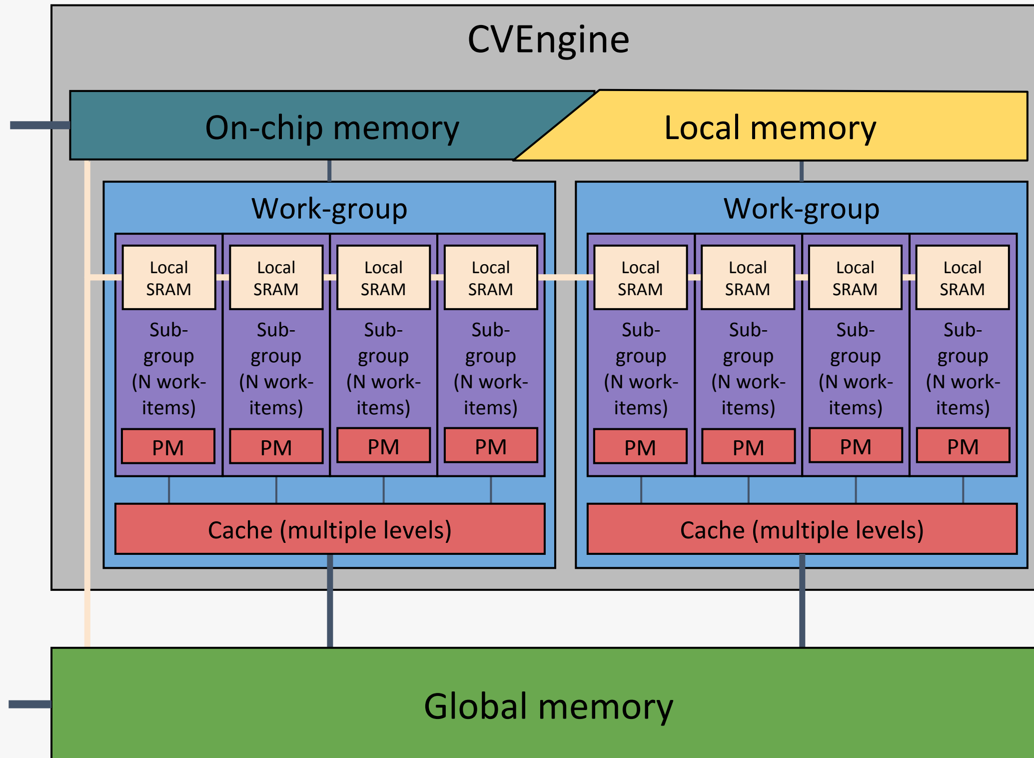
On-chip SRAM memory is mapped to local memory

© 2019 Codeplay Software Ltd.



Registers are mapped to private memory

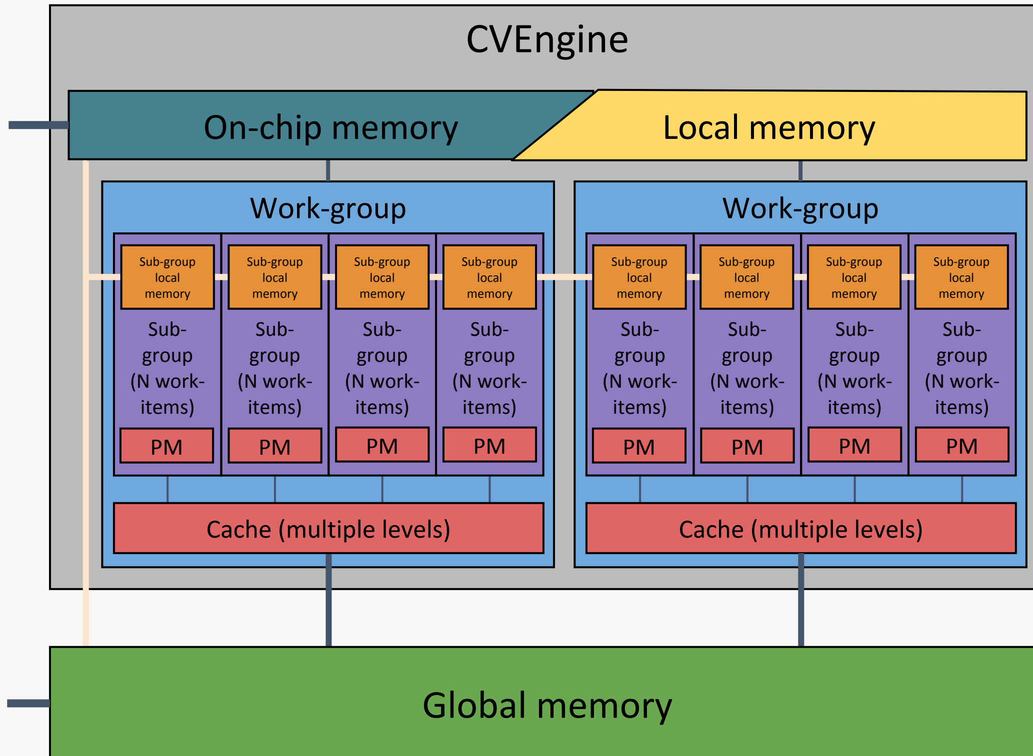
© 2019 Codeplay Software Ltd.



Since SRAM can be written to and read from the CPU and can be accessed by all work-groups similar to global memory

SRAM can also be used to allocate low-latency on-chip buffers

© 2019 Codeplay Software Ltd.

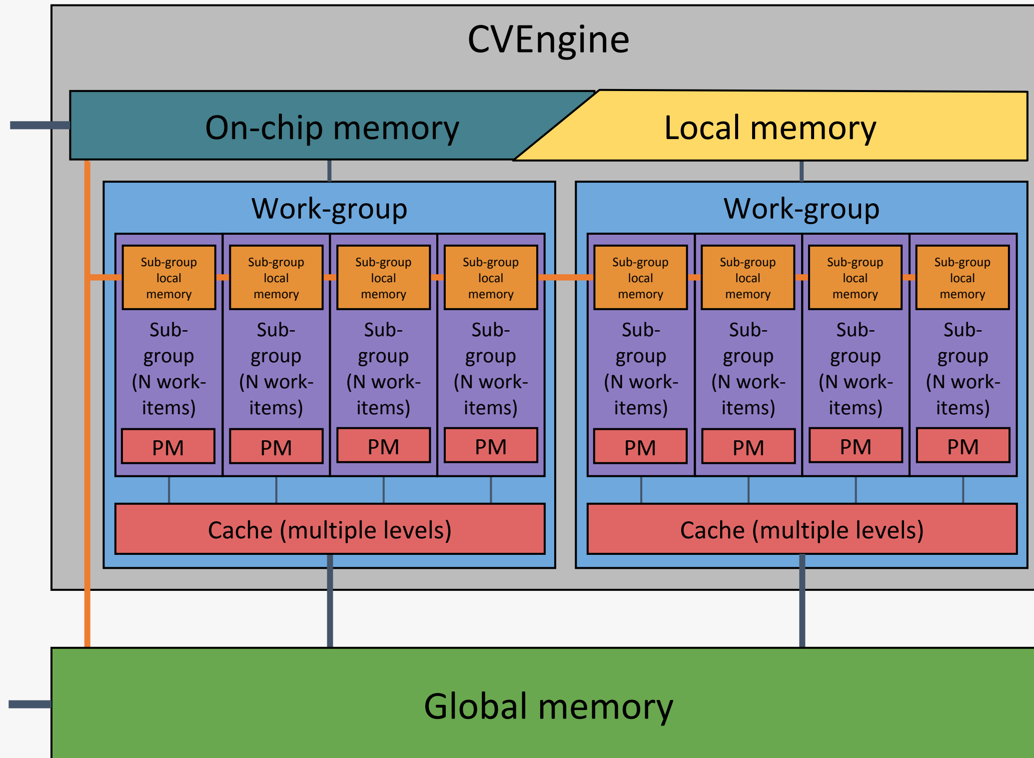


Since each core has its own dedicated local SRAM

Local SRAM can be mapped to a sub-group local memory

Sub-group local memory is not yet available in OpenCL or SYCL so this will require an extension

© 2019 Codeplay Software Ltd.

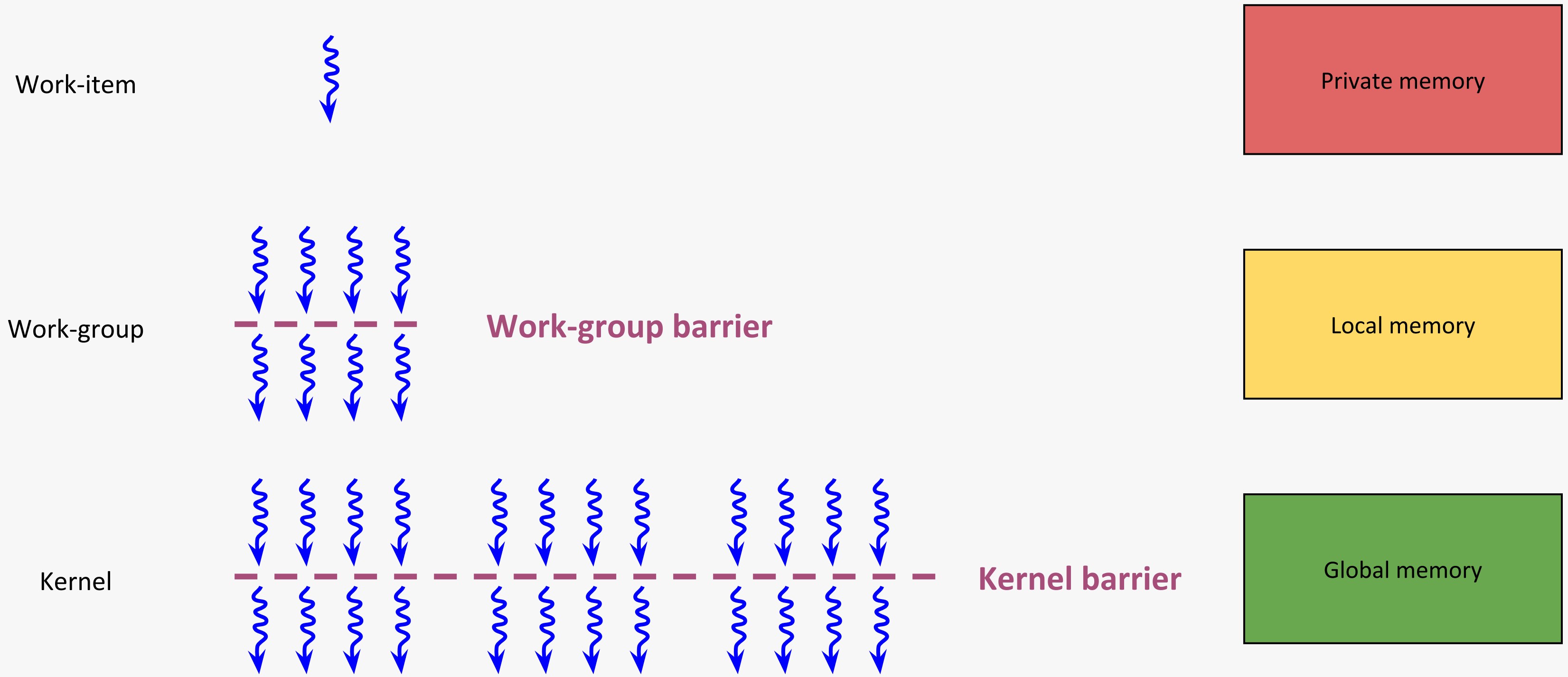


Since there is DMA connections from SRAM and DDR to local SRAM

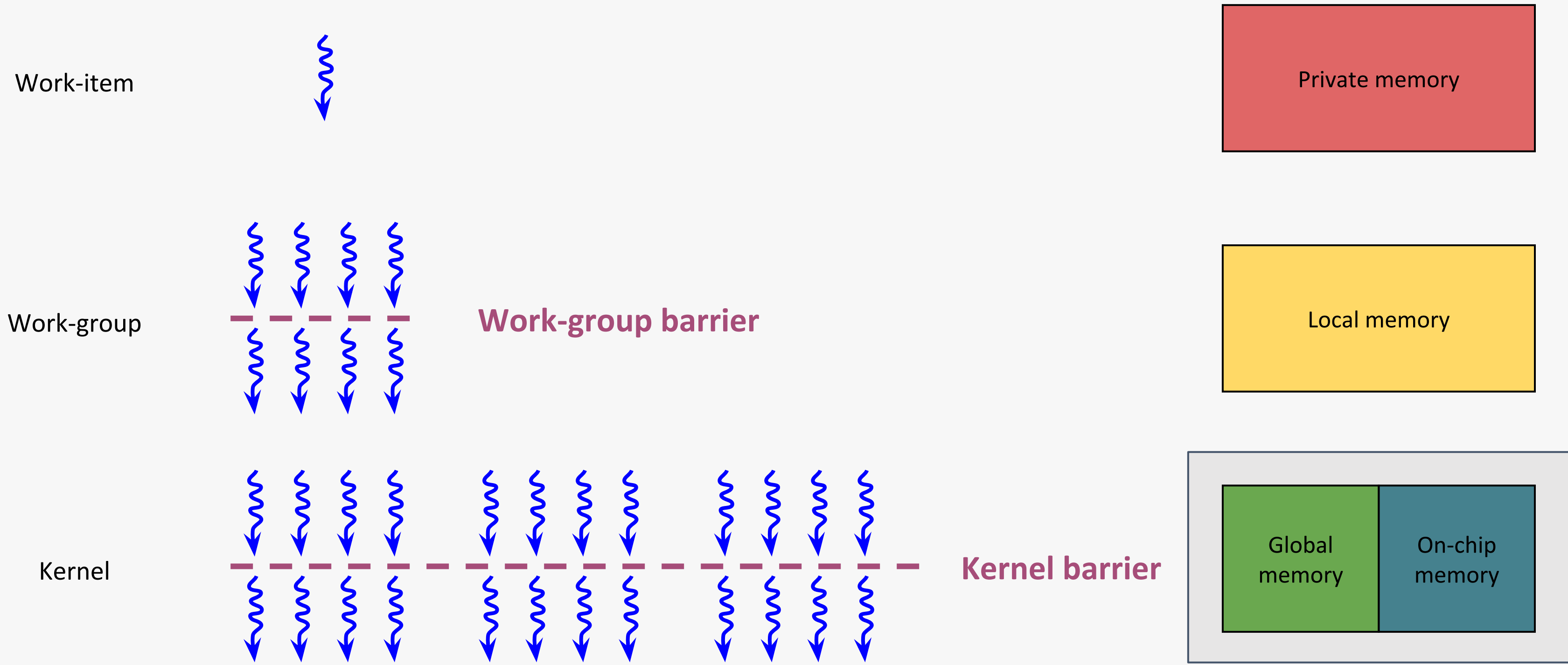
The CVEngine can support asynchronous memory copies from on-chip memory buffers and global memory buffers into sub-group local memory and vise versa

These asynchronous copies cannot be represented in OpenCL or SYCL so will require an extension

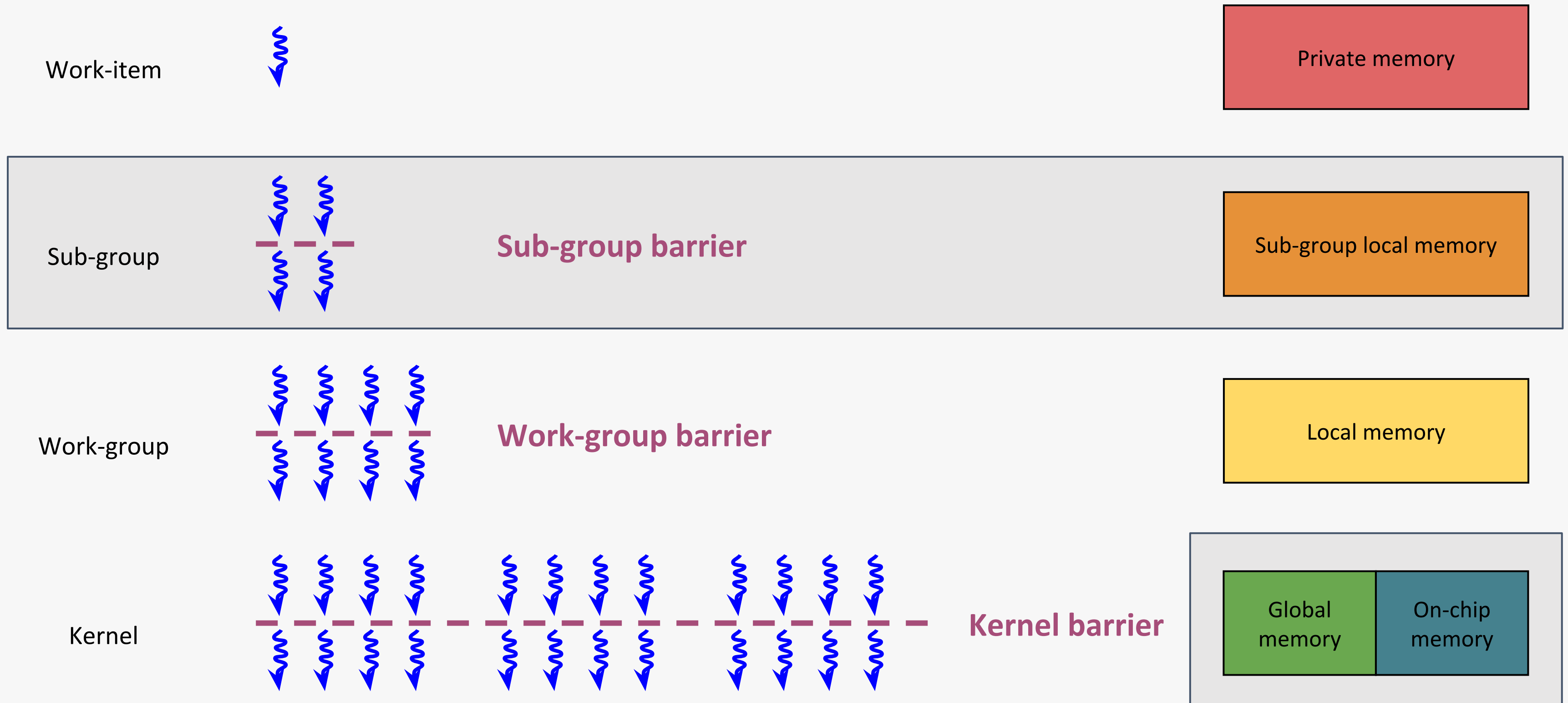
© 2019 Codeplay Software Ltd.



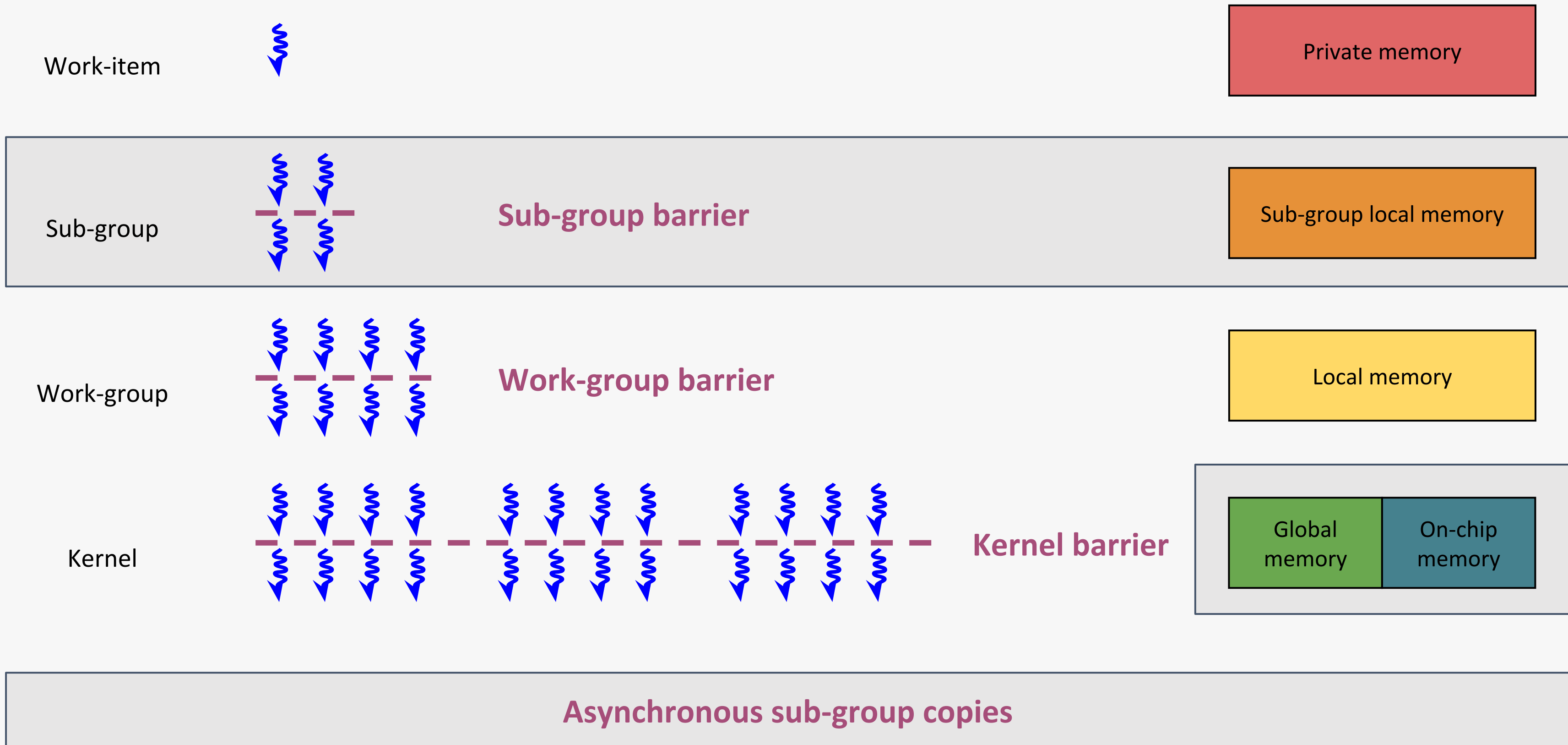
© 2019 Codeplay Software Ltd.



© 2019 Codeplay Software Ltd.



© 2019 Codeplay Software Ltd.



© 2019 Codeplay Software Ltd.

Agenda

Emergent hardware for AI in automotive

Overview of OpenCL/SYCL programming model

Mapping typical hardware to the OpenCL/SYCL programming model

The Renesas R-Car architecture

Extending OpenCL & SYCL for R-Car

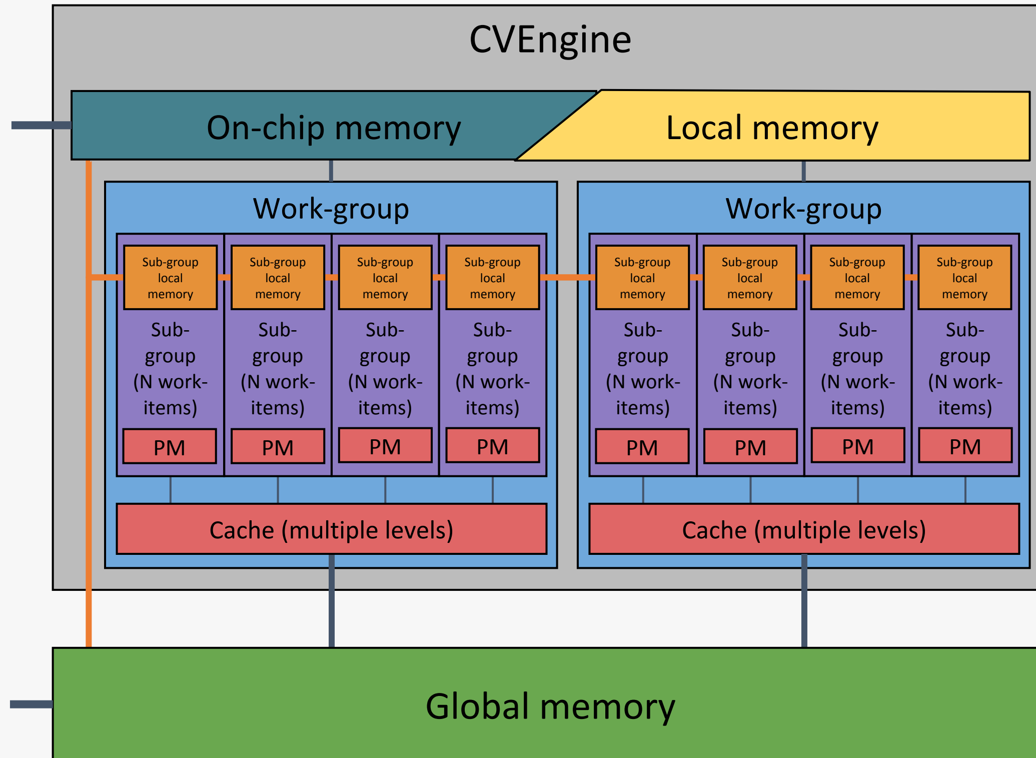
Optimising machine learning algorithms using R-Car

© 2019 Codeplay Software Ltd.

Disclaimer

The features that I present here are Codeplay extensions and are not standard SYCL features

© 2019 Codeplay Software Ltd.



© 2019 Codeplay Software Ltd.

- On-chip memory
    - On-chip memory is allocated in OpenCL/SYCL similarly to regular buffers
        - ComputeAorta (OpenCL) provides an extension API
        - ComputeCpp (SYCL) provides an extension buffer property **use_onchip_memory**
    - On-chip memory buffers are accessed in OpenCL/SYCL kernels in the same way as regular buffers

© 2019 Codeplay Software Ltd.

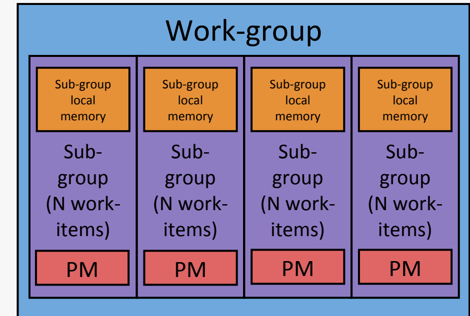
```
class kernel;

using namespace cl::sycl;

{
  queue deviceQueue;

  buffer<float, 1> onchipBuffer(hostData, size,
    {codeplay::property::buffer::use_onchip_memory(
      codeplay::property::require)});

  deviceQueue.submit([&](handler &cgh){

    auto onchipAcc =
      onchipBuffer.get_access<access::mode::read_write>(cgh);

    cgh.parallel_for<kernel>(range<1>(size), [=](id<1> idx){
      onchipAcc[idx] = onchipAcc[idx] * onchipAcc[idx];
    }):
  });
}
```

We construct a SYCL buffer as normal, but provide the **use_onchip_memory** buffer property

This property takes an enumeration; either **require**, which means that SYCL runtime has to use it or **prefer**, which means the SYCL runtime should try to use it

© 2019 Codeplay Software Ltd.

- Sub-groups
  - Sub-groups are exposed following the OpenCL 2.x feature and as a natural extension to the SYCL execution model
    - ComputeAorta (OpenCL) provides kernel builtins for querying sub-group info and invoking a sub-group barrier
    - ComputeCpp (SYCL) provides an extension to **nd_item** to expose a **sub_group** object, similar to group, which exposes member functions for querying sub-group info and invoking a sub-group barrier
  - The size of sub-groups cannot be specified explicitly by the users, they are determined by the implementation



© 2019 Codeplay Software Ltd.

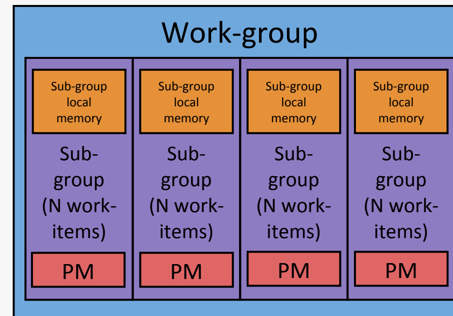
```
class kernel;

using namespace cl::sycl;

{
  queue deviceQueue;

  buffer<float, 1> deviceBuffer(hostData, size);

  deviceQueue.submit([&](handler &cgh){

    auto deviceAcc=
      deviceBuffer.get_access<access::mode::read_write>(cgh);

    cgh.parallel_for<kernel>(nd_range<1>(range<1>(size), range<1>(32)),
      [=](nd_item<1> ndItem){
      …
      auto subGroup = ndItem.get_sub_group();
      auto subGroupRange = subGroup.get_group_range();
      auto subGroupId = subGroup.get_group_id();
      subGroup.barrier();
      …
    }):
  });
}
```

We query in-kernel sub-group information and invoke sub-group barriers via the **sub_group** class and the nd_item has a member function called **get_sub_group** that will return a **sub_group** object

If an implementation does not support sub-groups using **sub_group** is undefined

© 2019 Codeplay Software Ltd.

- Sub-group local memory
  - Sub-group local memory is exposed with extensions which follow the OpenCL/SYCL memory model
    - ComputeAorta (OpenCL) provides a new address space which can be used to allocate sub-group local memory
    - ComputeCpp (SYCL) provides a new accessor access target; **access::target::subgroup_local**, that behaves similarly to **access::target::local**



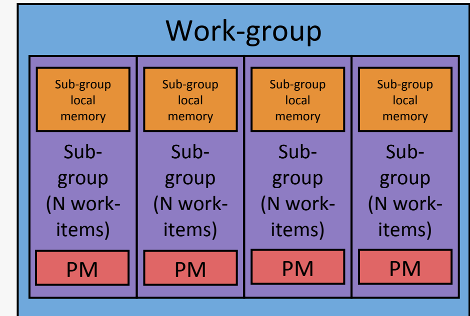
© 2019 Codeplay Software Ltd.

```
class kernel;

using namespace cl::sycl;

{
  queue deviceQueue;

  buffer<float, 1> deviceBuffer(hostData, size);

  deviceQueue.submit([&](handler &cgh){

    auto deviceAcc=
      deviceBuffer.get_access<access::mode::read_write>(cgh);

    auto subGroupLocalMem = accessor<float, 1, access::mode::read_write,
      access::target::subgroup_local>(cgh, range<1>(32));

    cgh.parallel_for<kernel>(nd_range<1>(range<1>(size), range<1>(32)),
      [=](nd_item<1> ndItem){
      …
      subGroupLocalMem[idx] = …;
      …
    }):
  });
}
```

We allocate sub-group local memory by constructing an accessor with the **subgroup_local** access target

© 2019 Codeplay Software Ltd.

- Asynchronous sub-group copies
  - Asynchronous sub-group copies are exposed following the OpenCL/SYCL feature for asynchronous work-group copies
    - ComputeAorta (OpenCL) provides a **plane_t** type to represent a non-accessible buffer and kernel builtins for invoking an asynchronous in-kernel copies between a **plane_t** and a sub-group local memory allocation
    - ComputeCpp (SYCL) provides a new accessor access target; **access::target::plane**, and a member function to the **sub_group** extension; **async_sub_group_copy**, to perform an asynchronous in-kernel copy from a plane accessor to a sub-group local accessor



© 2019 Codeplay Software Ltd.

```
class kernel;
using namespace cl::sycl;

{
  queue deviceQueue;

  buffer<float, 1> deviceBuffer(hostData, size);

  deviceQueue.submit([&](handler &cgh){

    auto devicePlane =
      deviceBuffer.get_access<access::mode::read_write,
        access::target::plane>(cgh);

    auto subGroupLocalMem = accessor<float, 1, access::mode::read_write,
      access::target::subgroup_local>(cgh, range<1>(32));

    cgh.parallel_for<kernel>(nd_range<1>(range<1>(size), range<1>(32)),
      [=](nd_item<1> ndItem){
      …
      auto subGroup = ndItem.get_sub_group();
      auto event = subGroup.async_sub_group_copy(subGroupLocalMem,
        devicePlane, range<1>(32));
      …
      event.wait();
    }):
  });
}
```

We construct a plane accessor using the **plane** access target

We perform asynchronous in-kernel sub-group copies by calling the **sub_group** member function **async_sub_group_copy**

This returns a device_event that can be used to wait on the copy to complete.

© 2019 Codeplay Software Ltd.

Agenda

Emergent hardware for AI in automotive

Overview of OpenCL/SYCL programming model

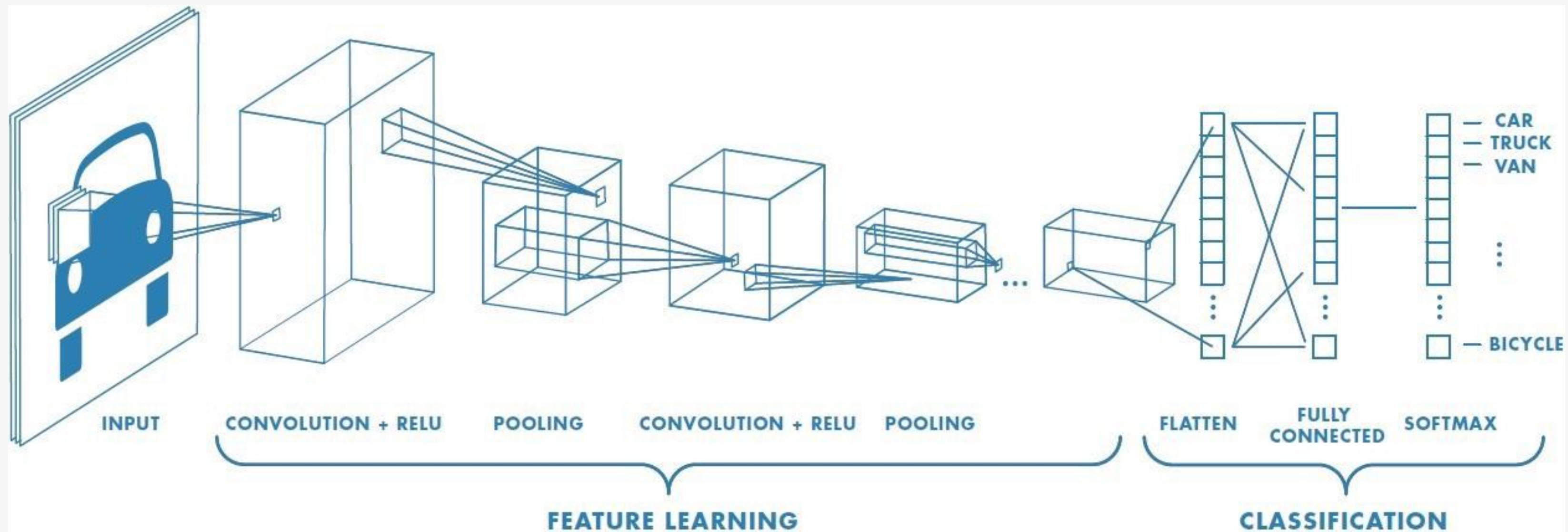
Mapping typical hardware to the OpenCL/SYCL programming model

The Renesas R-Car architecture

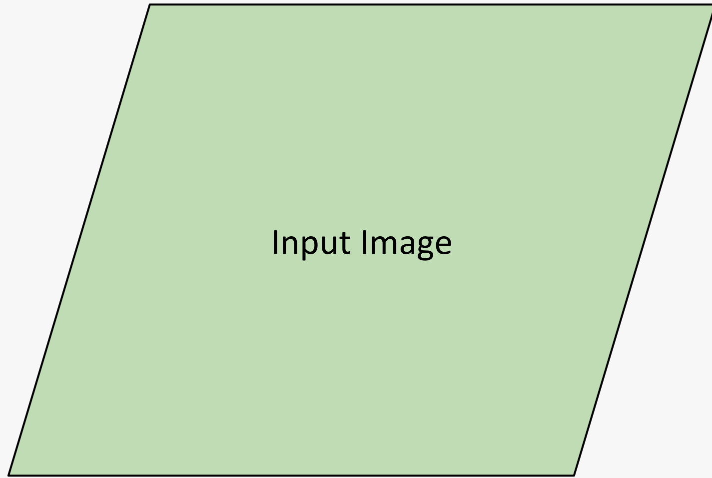
Extending OpenCL & SYCL for R-Car

Optimising machine learning algorithms using R-Car

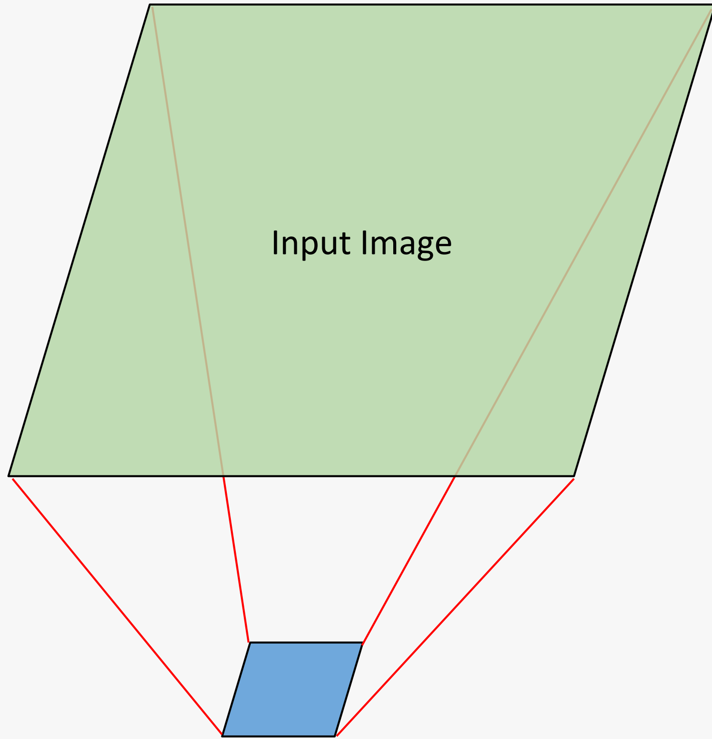
© 2019 Codeplay Software Ltd.



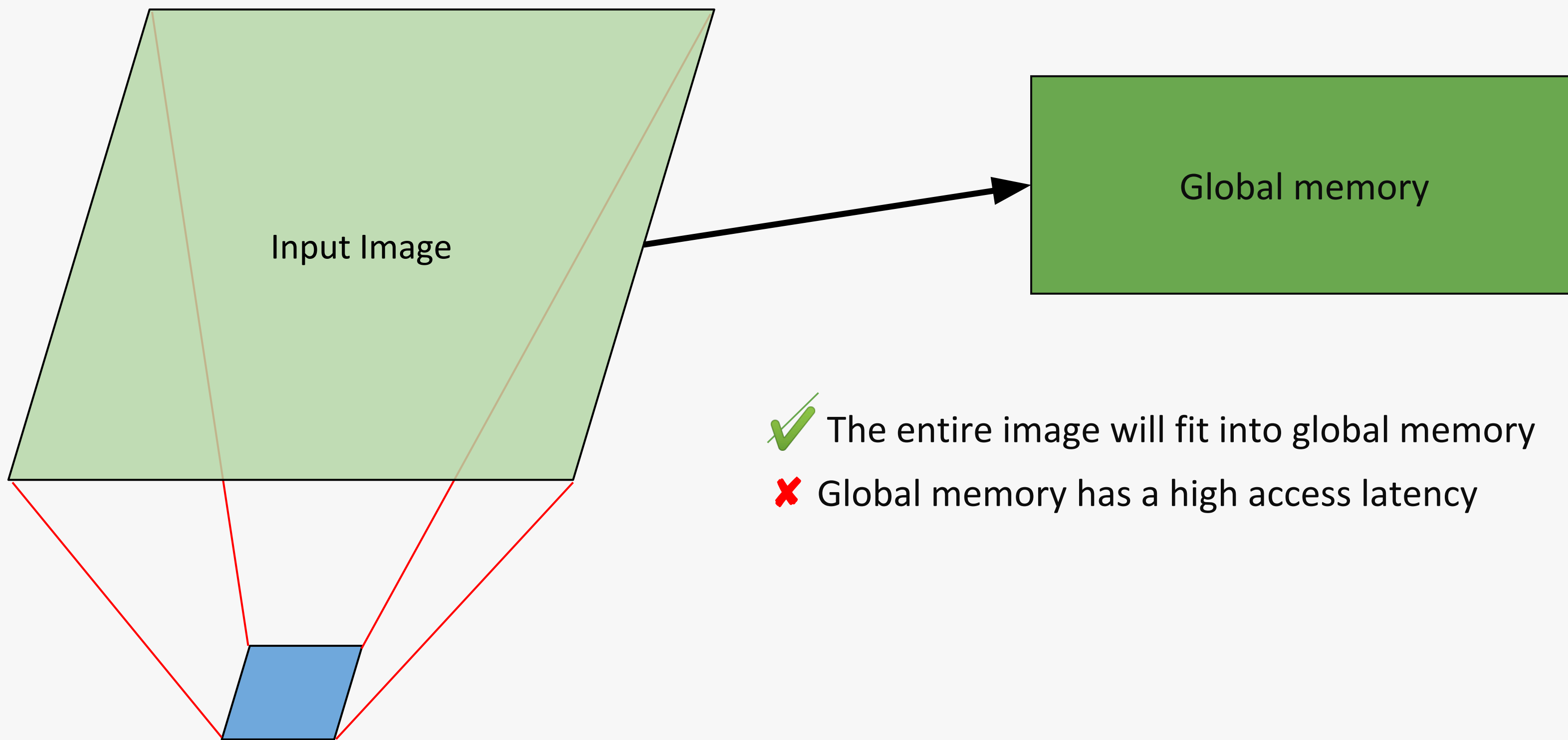
© 2019 Codeplay Software Ltd.



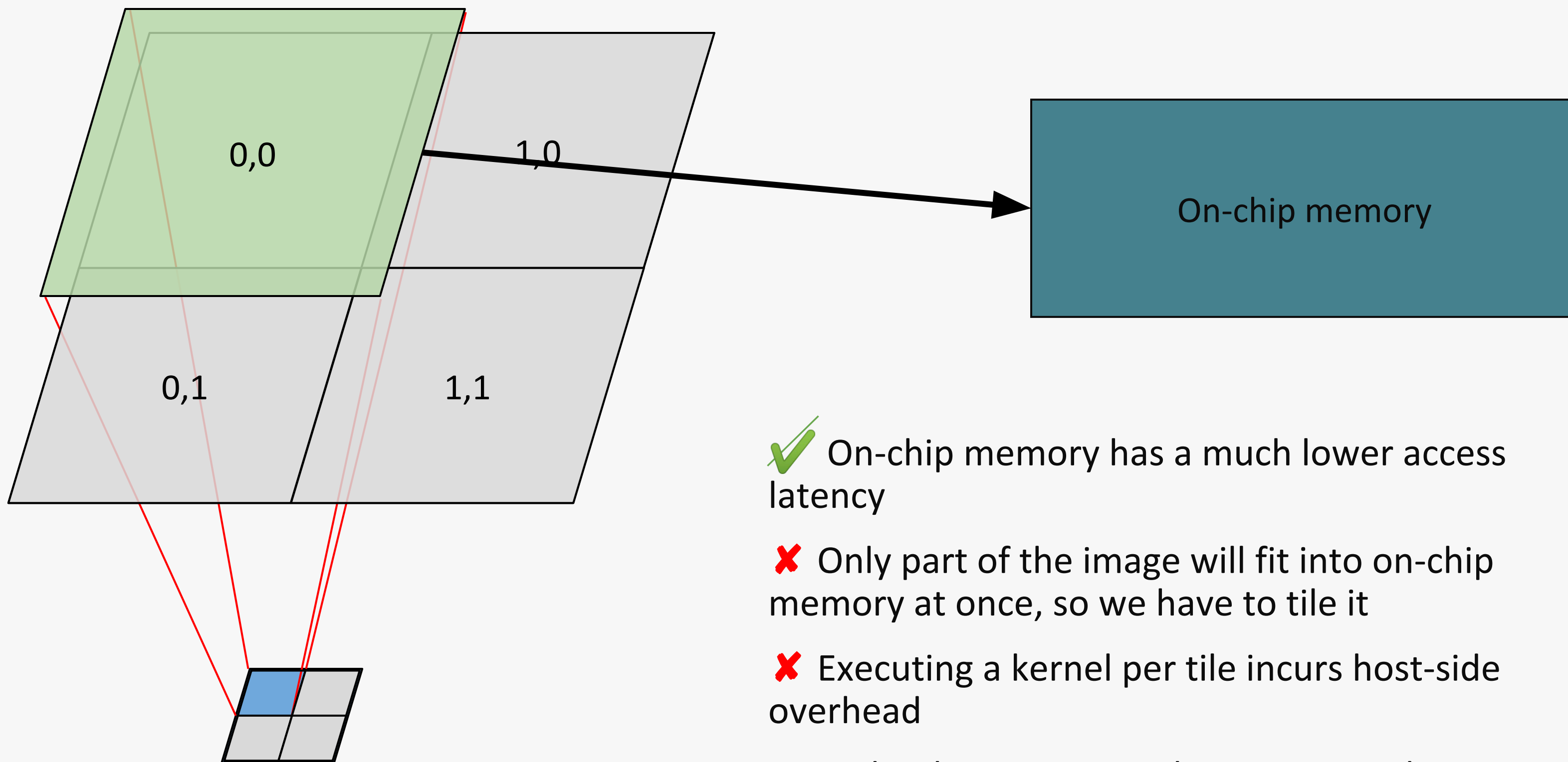
© 2019 Codeplay Software Ltd.



© 2019 Codeplay Software Ltd.



© 2019 Codeplay Software Ltd.



✔ On-chip memory has a much lower access latency

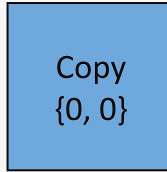
✘ Only part of the image will fit into on-chip memory at once, so we have to tile it

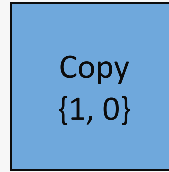
✘ Executing a kernel per tile incurs host-side overhead

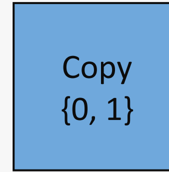
Note that because convolutions are gather operations the input data much include a halo

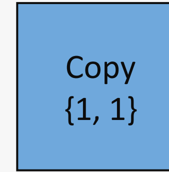
© 2019 Codeplay Software Ltd.

Copy

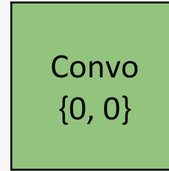


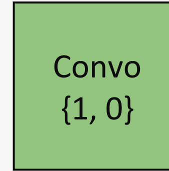


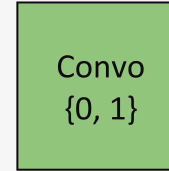


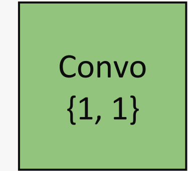


Compute



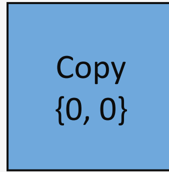


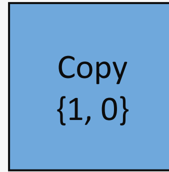


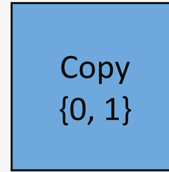


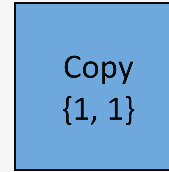
© 2019 Codeplay Software Ltd.

Copy

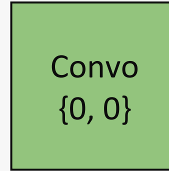


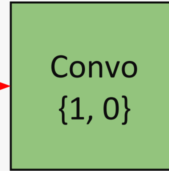


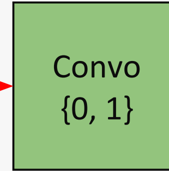


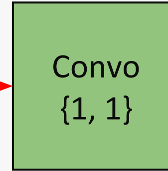


Compute



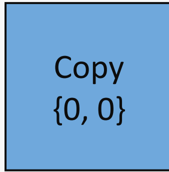


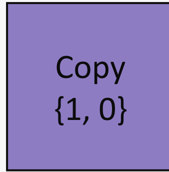


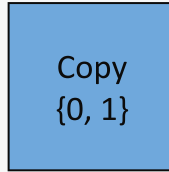


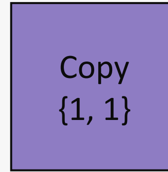
© 2019 Codeplay Software Ltd.

Copy

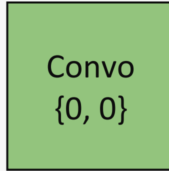


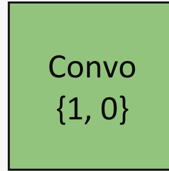


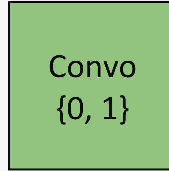


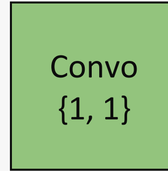


Compute



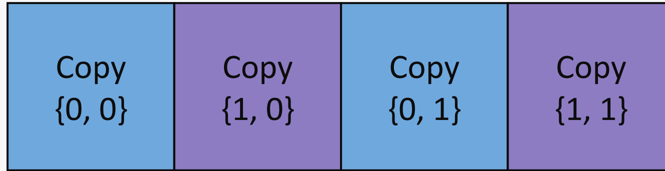




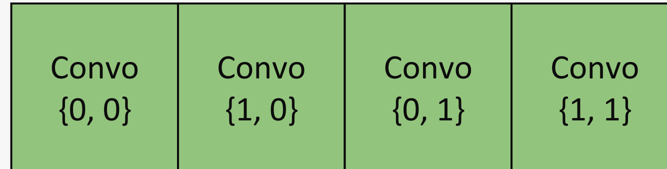


© 2019 Codeplay Software Ltd.

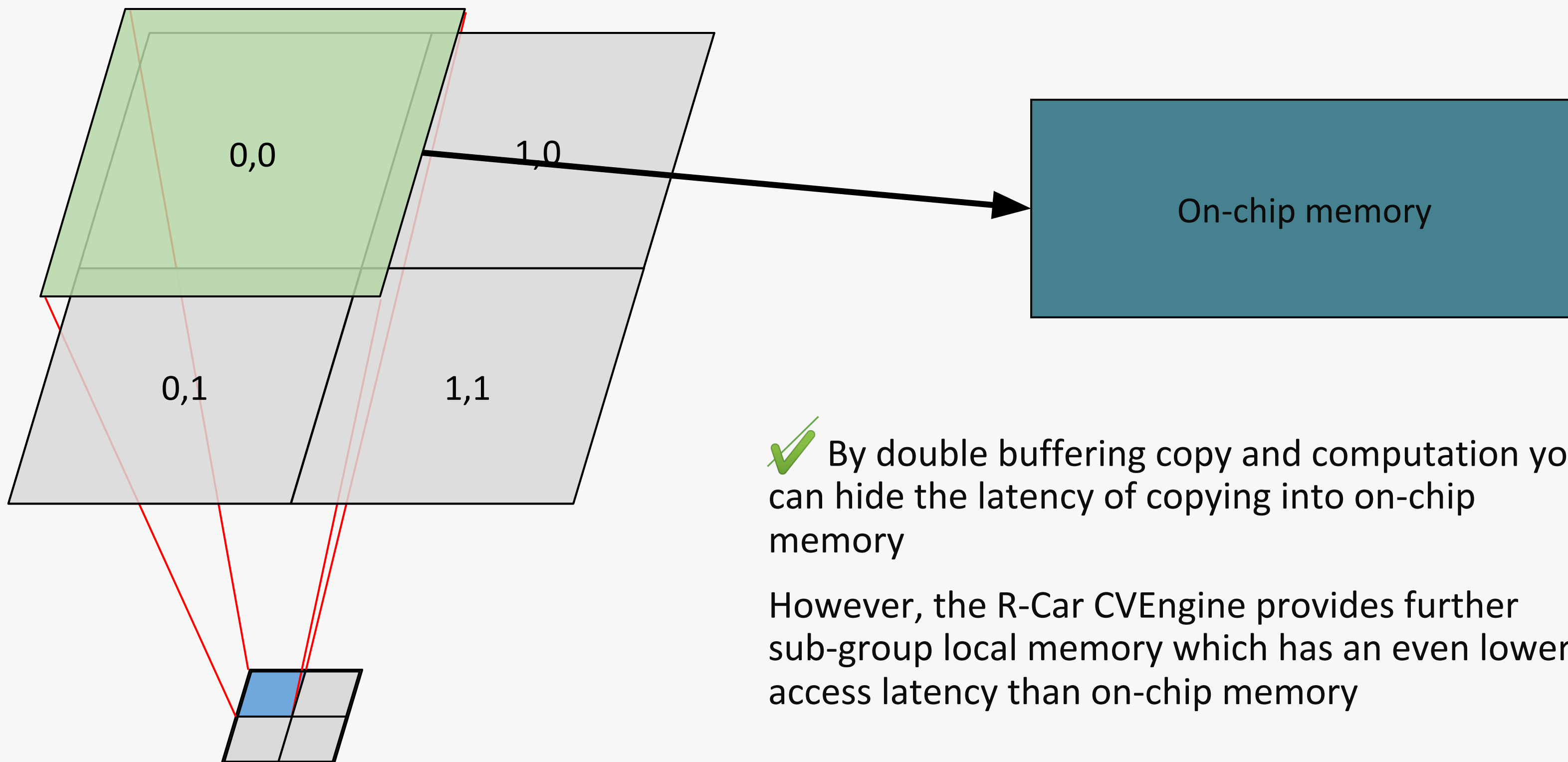
Copy



Compute



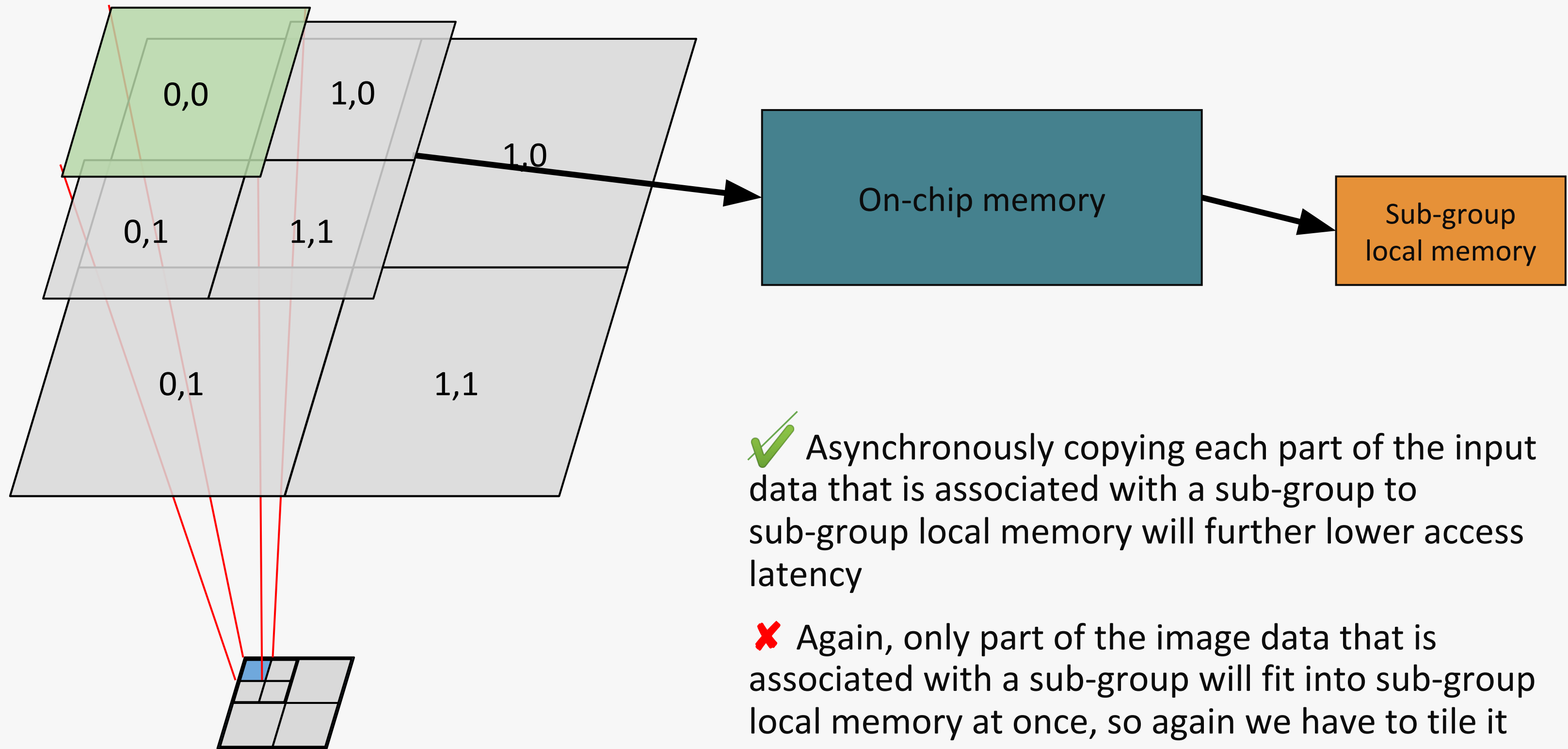
© 2019 Codeplay Software Ltd.



By double buffering copy and computation you can hide the latency of copying into on-chip memory

However, the R-Car CVEngine provides further sub-group local memory which has an even lower access latency than on-chip memory

© 2019 Codeplay Software Ltd.



✓ Asynchronously copying each part of the input data that is associated with a sub-group to sub-group local memory will further lower access latency

✗ Again, only part of the image data that is associated with a sub-group will fit into sub-group local memory at once, so again we have to tile it

In this case the tiling is done in-kernel

© 2019 Codeplay Software Ltd.

```
cgh.parallel_for<convo2d>(ndRange, [=](nd_item<1> ndItem){
  auto subGroup = ndItem.get_sub_group();

  auto numTiles = calculate_num_tiles(subGroup.get_group_range(), TILE_SIZE);

  auto currentTileRange = calculate_tile_range(subGroup, 0);
  auto nextTileRange = calculate_tile_range(subGroup, 1);

  subGroup.async_sub_group_copy(currentTileLocalMem, currentTilePlain, currentTileRange)
    .wait();
  copyEvent = subGroup.async_sub_group_copy(nextTileLocalMem, nextTilePlain,
      nextTileRange);

  for (int tile = 0; tile < numTiles; ++tile) {
    compute_tile(subGroup, currentTileRange, output);

    copyEvent.wait();

    if (tile == (numTiles - 1)) {
      copyEvent = subGroup.async_sub_group_copy(nextTileLocalMem, nextTilePlain,
        nextTileRange);

      currentTileRange = nextTileRange;
      nextTileRange = calculate_tile_range(subGroup, tile + 1);

      swap(currentTileLocalMem, nextTileLocalMem);
      swap(currentTilePlain, nextTilePlain);
    }
  }
});
```


© 2019 Codeplay Software Ltd.

```
cgh.parallel_for<convo2d>(ndRange, [=](nd_item<1> ndItem){
  auto subGroup = ndItem.get_sub_group();

  auto numTiles = calculate_num_tiles(subGroup.get_group_range(), TILE_SIZE);

  auto currentTileRange = calculate_tile_range(subGroup, 0);
  auto nextTileRange = calculate_tile_range(subGroup, 1);

  subGroup.async_sub_group_copy(currentTileLocalMem, currentTilePlain, currentTileRange)
    .wait();
  copyEvent = subGroup.async_sub_group_copy(nextTileLocalMem, nextTilePlain,
      nextTileRange);

  for (int tile = 0; tile < numTiles; ++tile) {
    compute_tile(subGroup, currentTileRange, output);

    copyEvent.wait();

    if (tile == (numTiles - 1)) {
      copyEvent = subGroup.async_sub_group_copy(nextTileLocalMem, nextTilePlain,
        nextTileRange);

      currentTileRange = nextTileRange;
      nextTileRange = calculate_tile_range(subGroup, tile + 1);

      swap(currentTileLocalMem, nextTileLocalMem);
      swap(currentTilePlain, nextTilePlain);
    }
  }
});
```

This kernel is operating on a tile that is stored in on-chip memory

© 2019 Codeplay Software Ltd.

```
cgh.parallel_for<convo2d>(ndRange, [=](nd_item<1> ndItem){
  auto subGroup = ndItem.get_sub_group();

  auto numTiles = calculate_num_tiles(subGroup.get_group_range(), TILE_SIZE);

  auto currentTileRange = calculate_tile_range(subGroup, 0);
  auto nextTileRange = calculate_tile_range(subGroup, 1);

  subGroup.async_sub_group_copy(currentTileLocalMem, currentTilePlain, currentTileRange)
    .wait();
  copyEvent = subGroup.async_sub_group_copy(nextTileLocalMem, nextTilePlain,
      nextTileRange);

  for (int tile = 0; tile < numTiles; ++tile) {
    compute_tile(subGroup, currentTileRange, output);

    copyEvent.wait();

    if (tile == (numTiles - 1)) {
      copyEvent = subGroup.async_sub_group_copy(nextTileLocalMem, nextTilePlain,
        nextTileRange);

      currentTileRange = nextTileRange;
      nextTileRange = calculate_tile_range(subGroup, tile + 1);

      swap(currentTileLocalMem, nextTileLocalMem);
      swap(currentTilePlain, nextTilePlain);
    }
  }
});
```

We want to perform the computation of the part of the input that each sub-group corresponds to in sub-group local memory

But all the memory required may not fit into sub-group local memory at once

So we calculate how many tiles are required for a sub-group

© 2019 Codeplay Software Ltd.

```
cgh.parallel_for<convo2d>(ndRange, [=](nd_item<1> ndItem){
  auto subGroup = ndItem.get_sub_group();

  auto numTiles = calculate_num_tiles(subGroup.get_group_range(), TILE_SIZE);

  auto currentTileRange = calculate_tile_range(subGroup, 0);
  auto nextTileRange = calculate_tile_range(subGroup, 1);

  subGroup.async_sub_group_copy(currentTileLocalMem, currentTilePlain, currentTileRange)
    .wait();
  copyEvent = subGroup.async_sub_group_copy(nextTileLocalMem, nextTilePlain,
      nextTileRange);

  for (int tile = 0; tile < numTiles; ++tile) {
    compute_tile(subGroup, currentTileRange, output);

    copyEvent.wait();

    if (tile == (numTiles - 1)) {
      copyEvent = subGroup.async_sub_group_copy(nextTileLocalMem, nextTilePlain,
        nextTileRange);

      currentTileRange = nextTileRange;
      nextTileRange = calculate_tile_range(subGroup, tile + 1);

      swap(currentTileLocalMem, nextTileLocalMem);
      swap(currentTilePlain, nextTilePlain);
    }
  }
});
```

First we initiate and wait on the copy of the first tile so we can perform the computation on it

Then we initiate, but don’t wait for the copy of the second tile, so that copy will happen in parallel to the computation of the first tile

© 2019 Codeplay Software Ltd.

```
cgh.parallel_for<convo2d>(ndRange, [=](nd_item<1> ndItem){
  auto subGroup = ndItem.get_sub_group();

  auto numTiles = calculate_num_tiles(subGroup.get_group_range(), TILE_SIZE);

  auto currentTileRange = calculate_tile_range(subGroup, 0);
  auto nextTileRange = calculate_tile_range(subGroup, 1);

  subGroup.async_sub_group_copy(currentTileLocalMem, currentTilePlain, currentTileRange)
    .wait();
  copyEvent = subGroup.async_sub_group_copy(nextTileLocalMem, nextTilePlain,
      nextTileRange);

  for (int tile = 0; tile < numTiles; ++tile) {
    compute_tile(subGroup, currentTileRange, output);

    copyEvent.wait();

    if (tile == (numTiles - 1)) {
      copyEvent = subGroup.async_sub_group_copy(nextTileLocalMem, nextTilePlain,
        nextTileRange);

      currentTileRange = nextTileRange;
      nextTileRange = calculate_tile_range(subGroup, tile + 1);

      swap(currentTileLocalMem, nextTileLocalMem);
      swap(currentTilePlain, nextTilePlain);
    }
  }
});
```

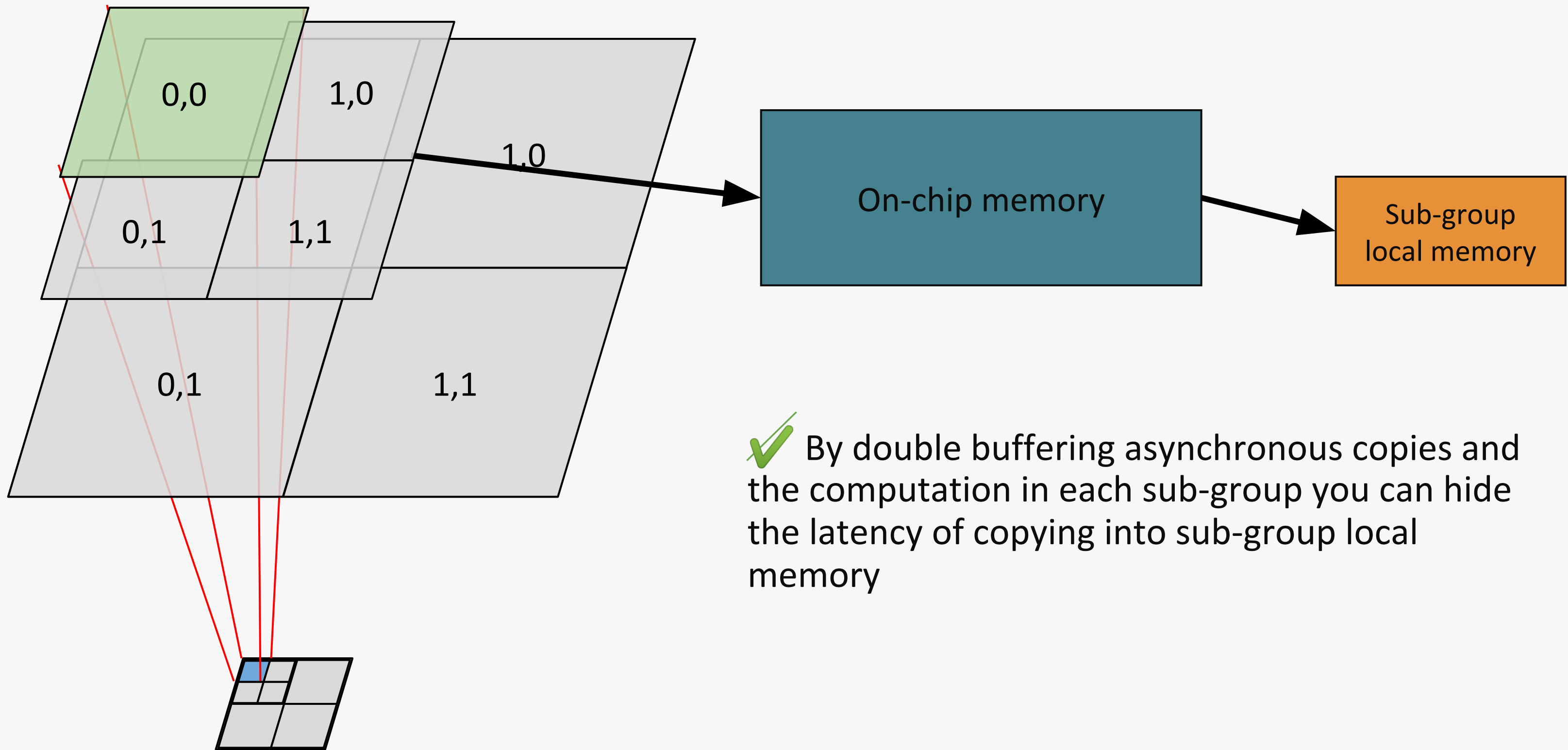
Then we iterate over the tiles, performing the computation of the current tile and then waiting on the copy for the next tile

© 2019 Codeplay Software Ltd.

```
cgh.parallel_for<convo2d>(ndRange, [=](nd_item<1> ndItem){
  auto subGroup = ndItem.get_sub_group();

  auto numTiles = calculate_num_tiles(subGroup.get_group_range(), TILE_SIZE);

  auto currentTileRange = calculate_tile_range(subGroup, 0);
  auto nextTileRange = calculate_tile_range(subGroup, 1);

  subGroup.async_sub_group_copy(currentTileLocalMem, currentTilePlain, currentTileRange)
    .wait();
  copyEvent = subGroup.async_sub_group_copy(nextTileLocalMem, nextTilePlain,
      nextTileRange);

  for (int tile = 0; tile < numTiles; ++tile) {
    compute_tile(subGroup, currentTileRange, output);

    copyEvent.wait();

    if (tile == (numTiles - 1)) {
      copyEvent = subGroup.async_sub_group_copy(nextTileLocalMem, nextTilePlain,
        nextTileRange);

      currentTileRange = nextTileRange;
      nextTileRange = calculate_tile_range(subGroup, tile + 1);

      swap(currentTileLocalMem, nextTileLocalMem);
      swap(currentTilePlain, nextTilePlain);
    }
  }
});
```

Finally, if there are further tiles to be processed, then we initiate the copy for the next tile and then swap the accessors for the next iteration of the loop

© 2019 Codeplay Software Ltd.



By double buffering asynchronous copies and the computation in each sub-group you can hide the latency of copying into sub-group local memory

© 2019 Codeplay Software Ltd.

Conclusion

- The Renesas R-Car CVEngine is designed to efficiently accelerate complex machine learning algorithms in a low power environment
- The OpenCL/SYCL programming memory can be efficiently applied and extended when necessary to support very unique hardware architectures
- This allows automotive systems to take advantage of AI software stacks based on open standards

© 2019 Codeplay Software Ltd.

We're Hiring!
codeplay.com/careers/

Thank you for listening

@codeplaysoft

/codeplaysoft

codeplay.com