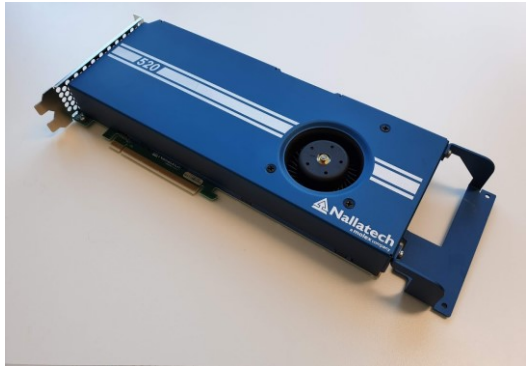
Australian National University

Accelerator Programming for HPC: Course Summary

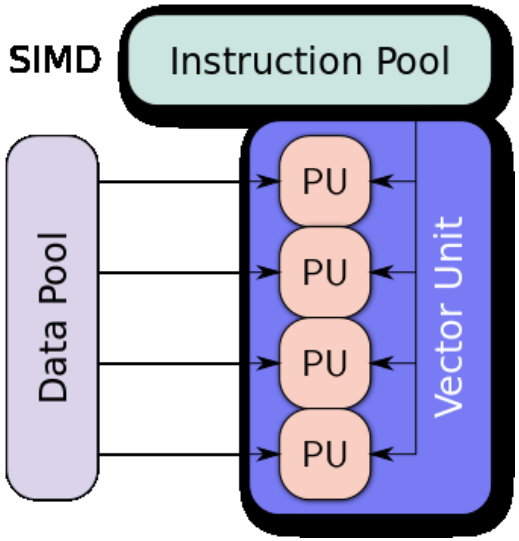
Accelerators for Parallel Computing

- Power Wall / Frequency Wall / ILP Wall ⇒ Go Parallel!
- SIMD: greater widths
- SIMT: greater numbers of processing elements
- Multicore / many-core: greater numbers of cores



SIMD (Vector Instructions)

- Data parallelism: apply the same operation to multiple data items at the same time
- More efficient: single instruction fetch and decode for all data items



Vadikus, SIMD2, CC BY-SA 4.0

GPU Architectures

- Streaming Multiprocessors / Compute Units
- Hardware threads (CUDA cores)
  - Arithmetic Logic Units (ALU)
- Memory types
  - Registers
  - Shared Memory
  - Global Memory
  - Constant / Texture Memory

Shared Memory Programming

- Explicit thread creation (pthreads):

```
pthread t thread;
pthread create(&thread, &attr, some_function, (void *)
&result);
…
pthread join(thread, NULL);
```

- Tasks (C++ 11):

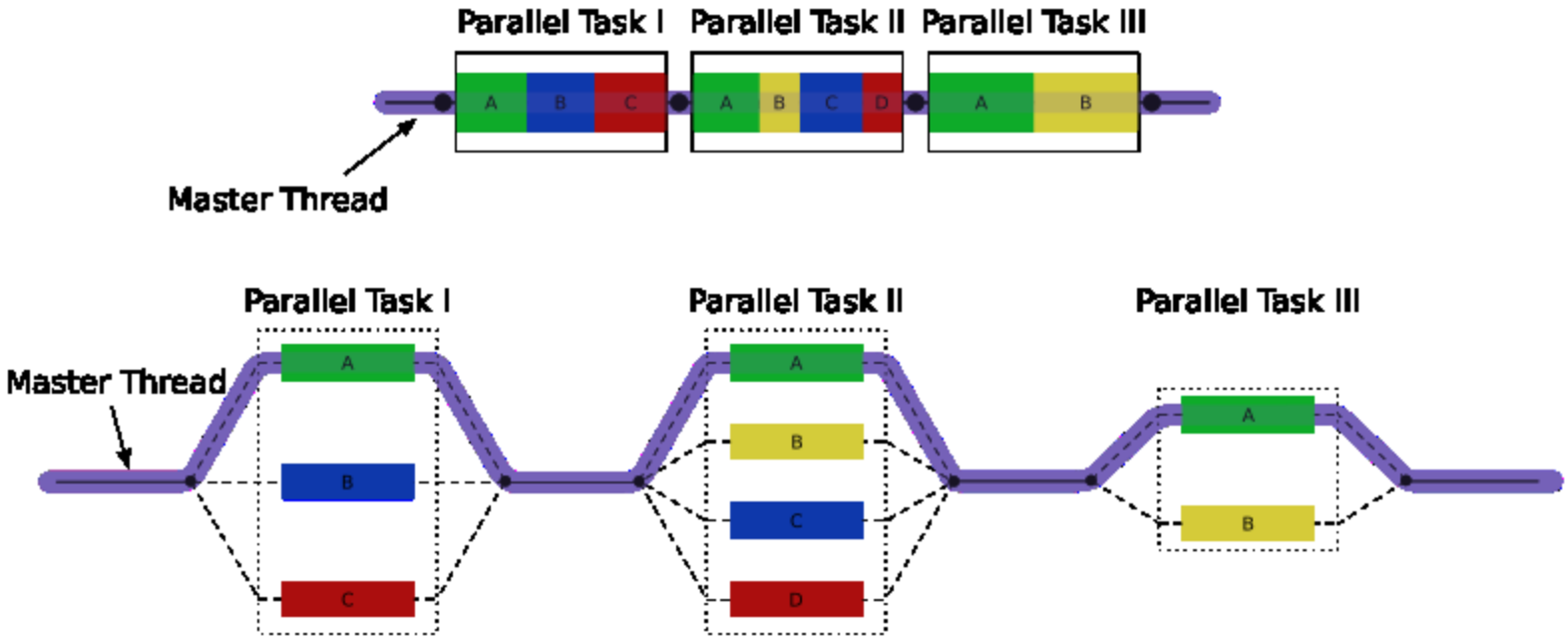
```
auto handle = std::async(std::launch::async, some_code, ...);
auto result = handle.get();
```

- Kernels (CUDA):

```
__global__
void someKernel(...) {
  int idx = blockIdx.x*blockDim.x + threadIdx.x
  // execute code for given index
}
```

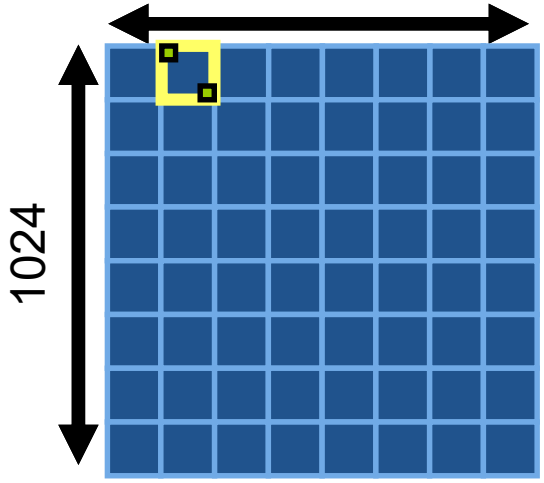
OpenMP

- Expose data parallelism through *parallel sections* and for loops



OpenCL

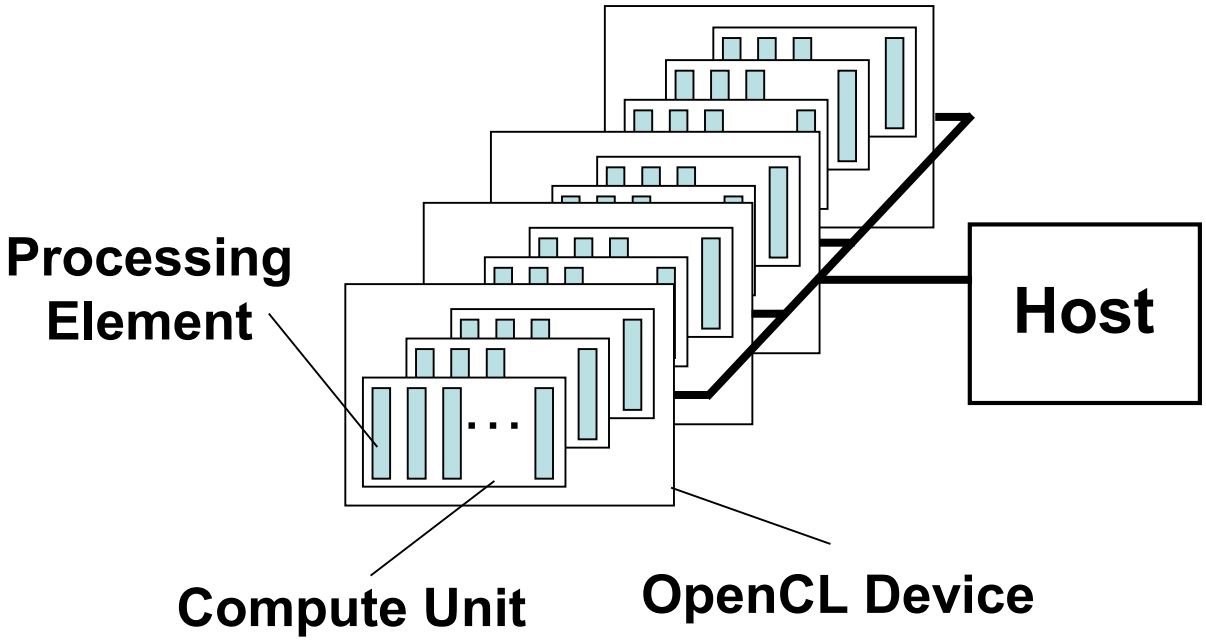
- Kernel programming to expose large amounts of potential parallelism to hardware




```
__kernel void
mul(__global const float *a,
    __global const float *b,
    __global       float *c)
{
  int i =                (0);
  c[i] = a[i] * b[i];
}
// many instances of the kernel,
// called work-items, execute
// in parallel
```

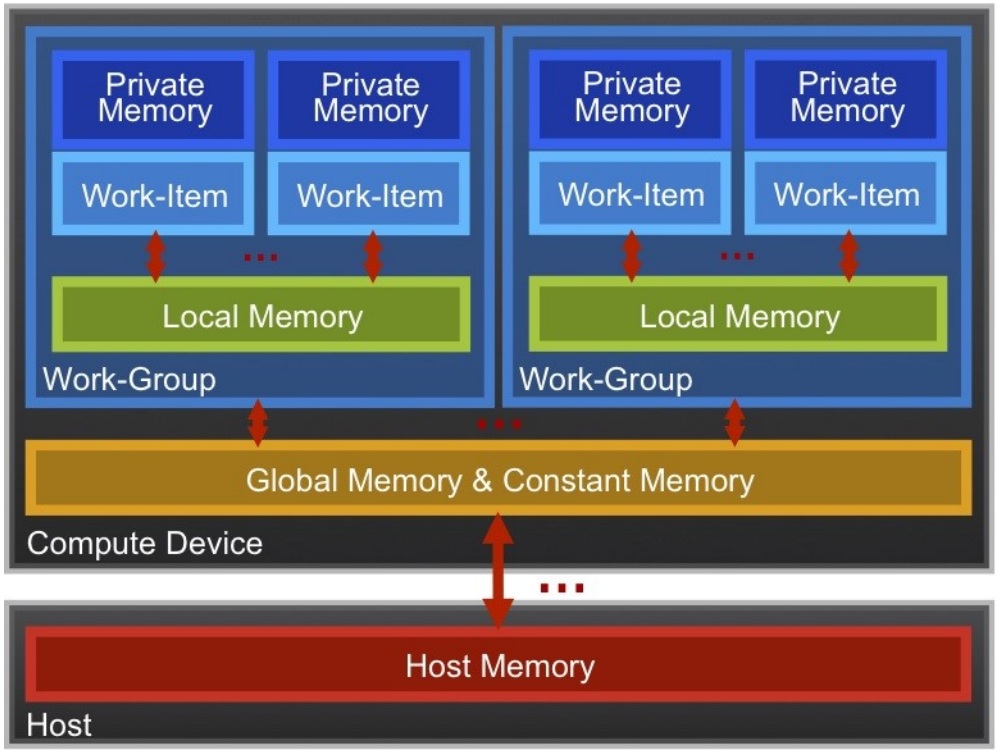
OpenCL Platform Model

- Single abstract platform model across wide range of device types (CPU, GPU, FPGA, MIC, …)

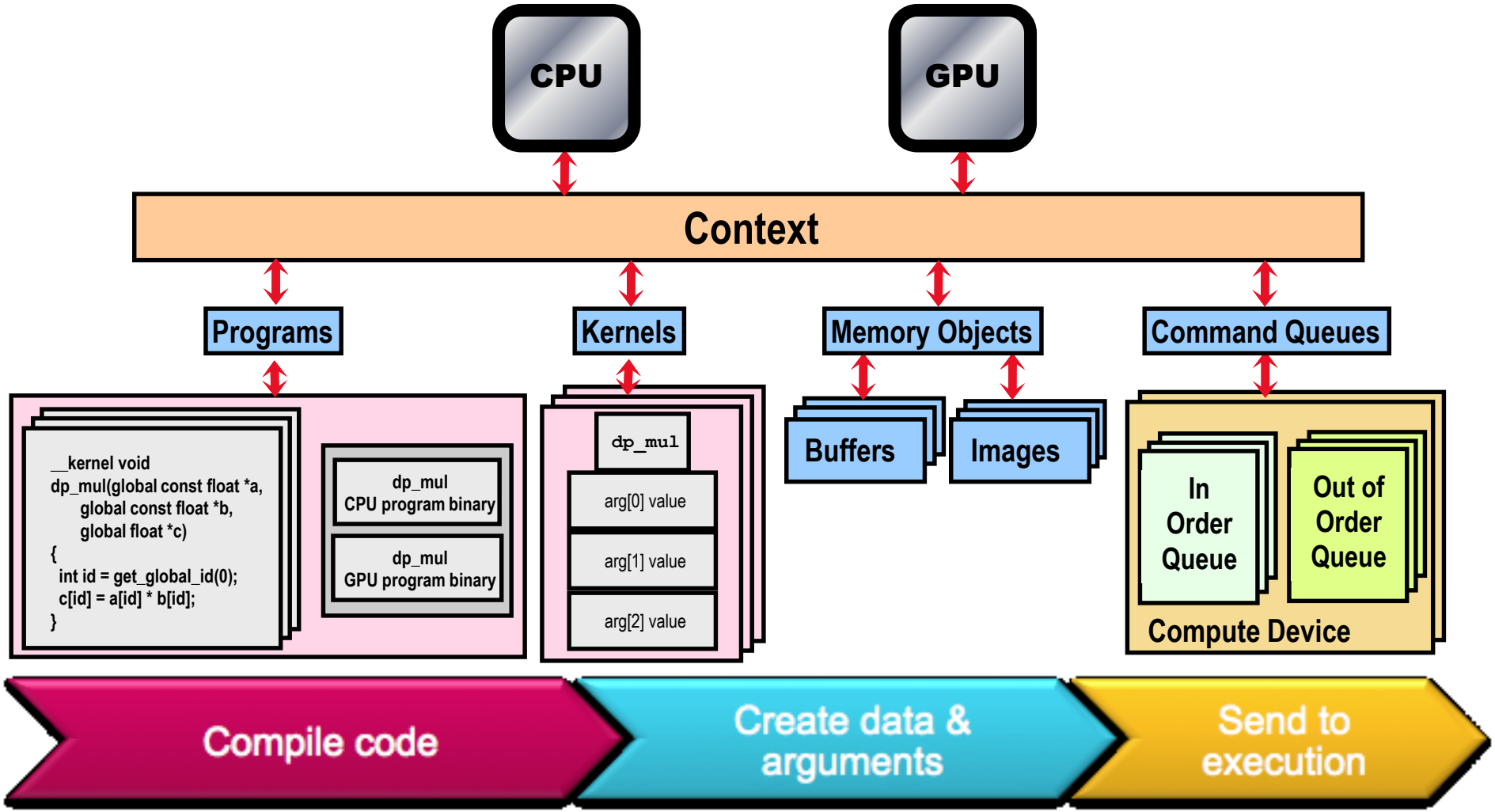


OpenCL Memory Model

- ***Private Memory***
  - Per work-item
- ***Local Memory***
  - Shared within a work-group
- ***Global Memory / Constant Memory***
  - Visible to all work-groups
- ***Host memory***
  - On the CPU



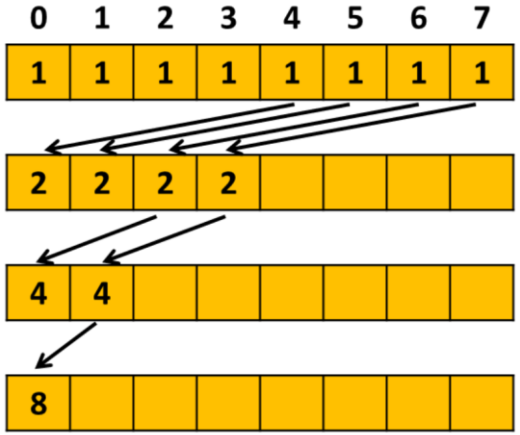
OpenCL Runtime APIS



CUDA Programming

- Launching kernels

```
__global__ void addone(int n, int *data) {
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx < n) data[idx] = data[idx] + 1;
}
addone<<<10,256>>>(n, data);
```



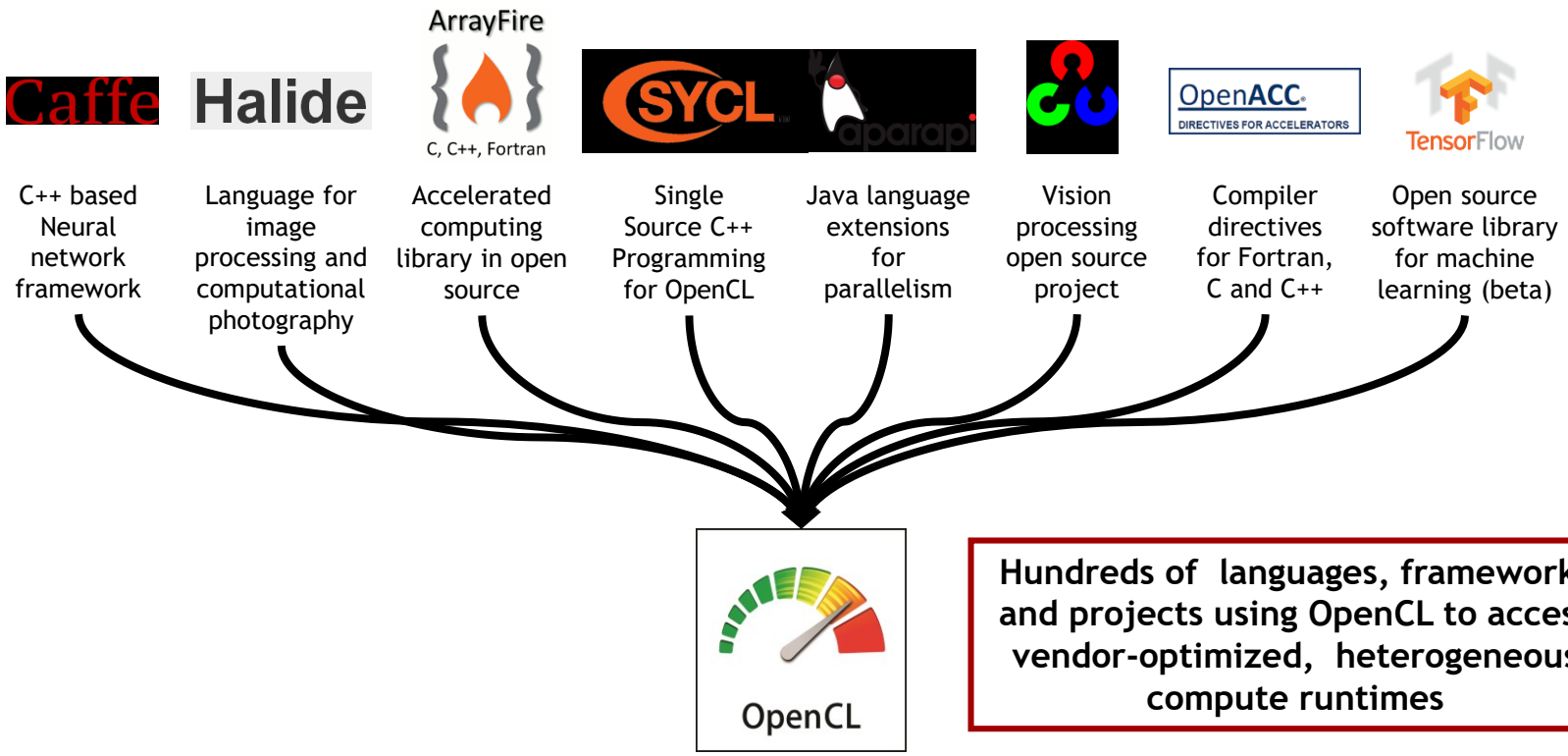
- Atomics
- Thread Divergence
- Streaming
- Parallel Reduction
- Warp-Level Synchronization

```
unsigned __ballot_sync(unsigned mask, int predicate);
T __shfl_sync(unsigned mask, T var, int srcLane);
```

https://devblogs.nvidia.com/faster-parallel-reductions-kepler/

Outlook:
Future Accelerator Architectures and Programming Models

OpenCL as Language/Library Backend



Source: Neil Trevett (2018). *OpenCL and Ecosystem: State of the Nation*. Khronos Group

SPIR-V Ecosystem

Open source tools and translators

https://github.com/KhronosGroup/SPIRV-Tools

Third party kernel and shader languages

GLSL **HLSL**

glslang **DXC**

SPIR-V
- Khronos defined cross-API IR
- Native graphics and parallel compute
- Easily parsed/extended 32-bit stream
- Data object/control flow retained for effective code generation/translation

MSL

HLSL

GLSL

SPIRV-Cross

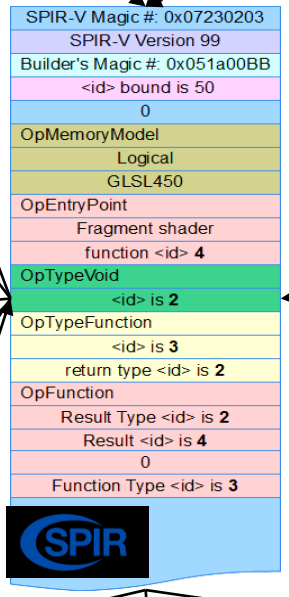
SPIR-V (Dis)Assembler

SPIR-V Validator

SPIRV-opt | SPIRV-remap

SPIR-V Optimizations
- Inlining (exhaustive)
- Store/Load Elimination
- Dead Code Elimination
- Dead Branch Elimination
- Common Uniform Elimination
- Loop Unrolling and Constant Folding
- Common Subexpression Elimination

Additional Intermediate Forms



OpenCL C Front-end **OpenCL C++ Front-end** **SYCL Front-end**

LLVM to SPIR-V Bi-directional Translators

Khronos liaising with Clang/LLVM Community E.g. discussing SPIR-V as supported Clang target

SPIR-V 1.3 released with Vulkan 1.1 inc. subgroups

OpenGL OpenCL Vulkan

Source: Neil Trevett (2018). *OpenCL and Ecosystem: State of the Nation*. Khronos Group

SYCL

- Single-source programming for heterogeneous systems
- Standard C++17

```
sycl::queue deviceQueue;
sycl::range<1> numOfItems{N};
std::array<sycl::cl_float, array_size> A, B, C;

sycl::buffer<sycl::cl_float, 1> bufferA(VA.data(), numOfItems);
sycl::buffer<sycl::cl_float, 1> bufferB(VB.data(), numOfItems);
sycl::buffer<sycl::cl_float, 1> bufferC(VC.data(), numOfItems);
deviceQueue.submit([&](sycl::handler& cgh) {
  auto accessorA = bufferA.template get_access<sycl_read>(cgh);
  auto accessorB = bufferB.template get_access<sycl_read>(cgh);
  auto accessorC = bufferC.template get_access<sycl_write>(cgh);
  cgh.parallel_for<class SimpleVadd<sycl::cl_float>>(numOfItems,
                  [=](sycl::id<1> wiID) {
    accessorC[wiID] = accessorA[wiID] + accessorB[wiID];
  });
});
```

Intel XE GPU

- 10nm process
- AVX512 SIMD instructions
- Cache-coherent GPU memory with CXL
- Programmed using OpenMP / OpenCL / SYCL
- Will provide majority of FP compute for DoE Exascale Computer ‘Aurora A21’

Source: Digital Trends

FPGAs for Custom Pipelines and Arithmetic

- FPGAs are suited to custom compute pipelines, including:
  - Low-latency, high-bandwidth connectivity ⇒ near-data processing
  - Logical operations & fixed-precision arithmetic
  - Variable precision, non-standard arithmetic
- For these reasons, FPGAs will become a standard part of data processing for computational science and engineering etc.
- Programmability is still a major challenge
- OpenCL may provide a path forward
  - OpenCL Next will allow vendors to support any range of OpenCL feature sets