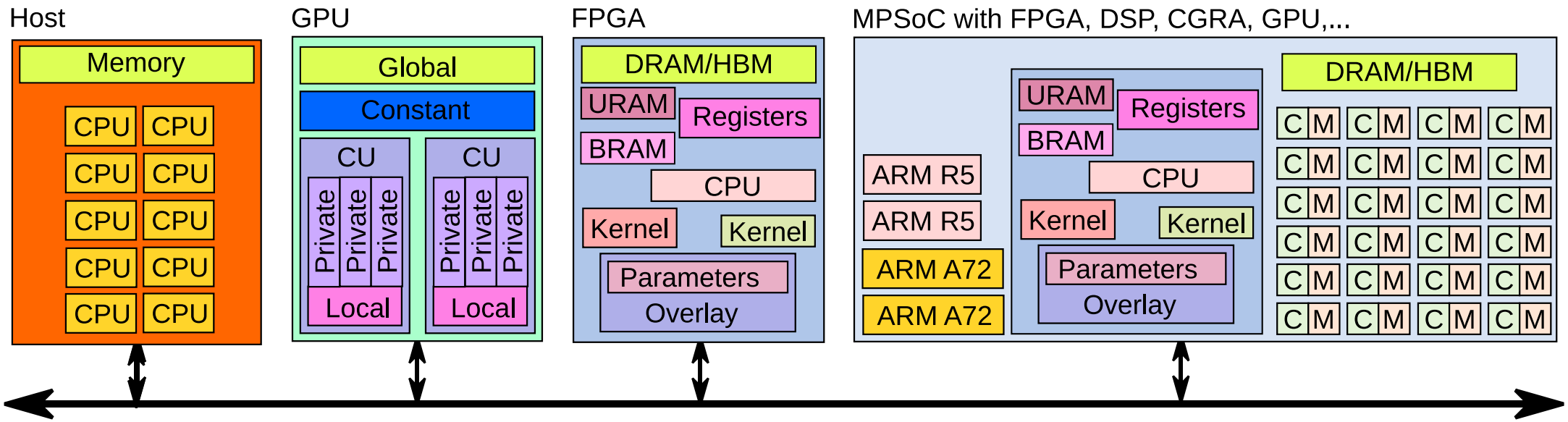
SYCL 2020

Inclusive heterogeneous computing with C++

Ronan Keryell
Khronos SYCL specification editor & Xilinx Research Labs (San José, California)
2021/04/26 oneAPI Developer Summit

Address the need for real inclusive programming standard



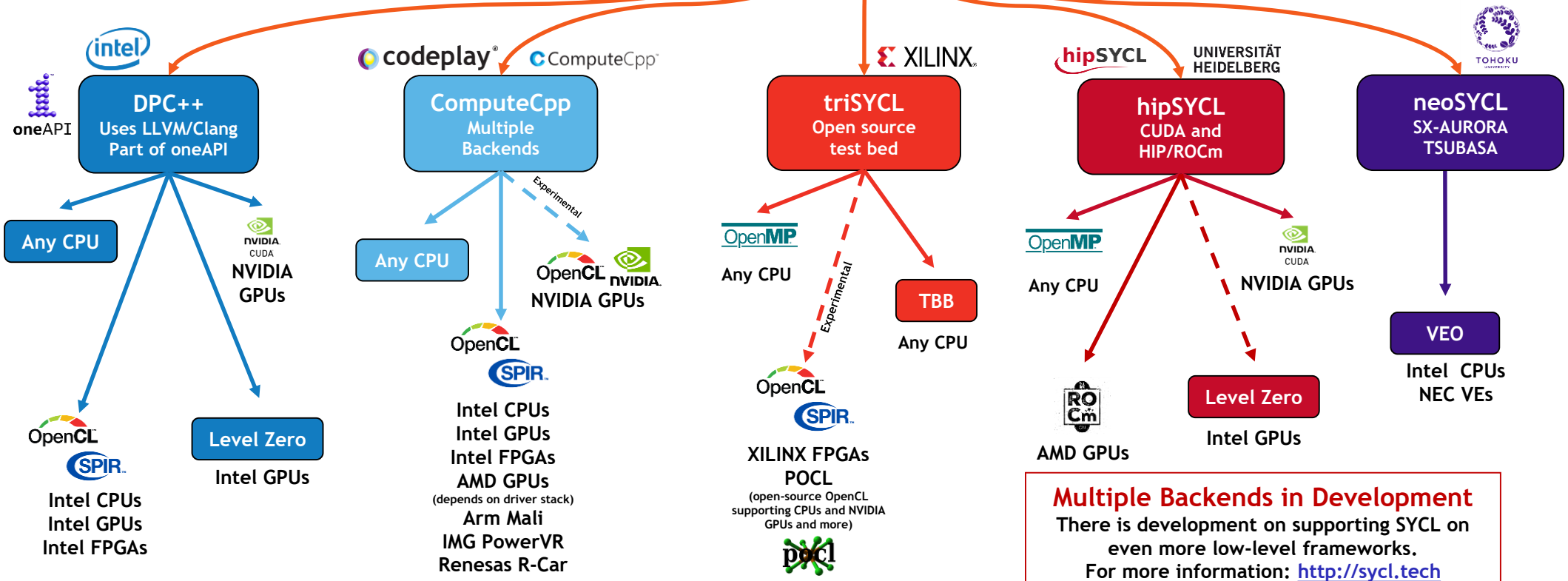
- From embedded systems up to HPC and data-center systems
- Complex systems with multiple devices from multiple vendors **at the same time!**
- No performance compromise
- Add your own devices on the picture!

SYCL ecosystem is growing

SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies

SYCL Source Code

SYCL enables Khronos to influence ISO C++ to (eventually) support heterogeneous compute



Multiple Backends in Development
There is development on supporting SYCL on even more low-level frameworks.
For more information: http://sycl.tech

+ Celerity: SYCL on MPI+SYCL

- https://www.khronos.org/blog/sycl-2020-what-do-you-need-to-know

SYCL 2020 ≡ heterogeneous simplicity with C++17 and up

- Abstract storage

```
#include <sycl/sycl.hpp>
#include <iostream>
using namespace sycl;
constexpr int N = 32;

int main () {
  buffer<int> buf { N };
  queue {}.submit([&](auto &h) {
     accessor a { buf, h, write_only, no_init };
     h.parallel_for(N, [=](auto i) { a[i] = i; });
  });
  for (host_accessor a { buf }; auto e : a)
     std::cout << e << std::endl;
}
```

- Code executed on device (“kernel”)
- “Single-source”
  - Seamless integration in host code
  - Type-safety

- Accessors
  - Express access intention
  - Implicit data flow graph
  - Automatic data transfers across devices
  - Overlap computation & communication

SYCL 2020 with unified shared memory (USM)

```
// Using buffers and accessors
#include <sycl/sycl.hpp>
#include <iostream>
using namespace sycl;
constexpr int N = 32;

int main () {
  buffer<int> buf { N };
  queue {}.submit([&](auto &h) {
     accessor a { buf, h, write_only, no_init };
     h.parallel_for(N, [=](auto i) { a[i] = i; });
  });
  for (host_accessor a { buf }; auto e : a)
     std::cout << e << std::endl;
}
```

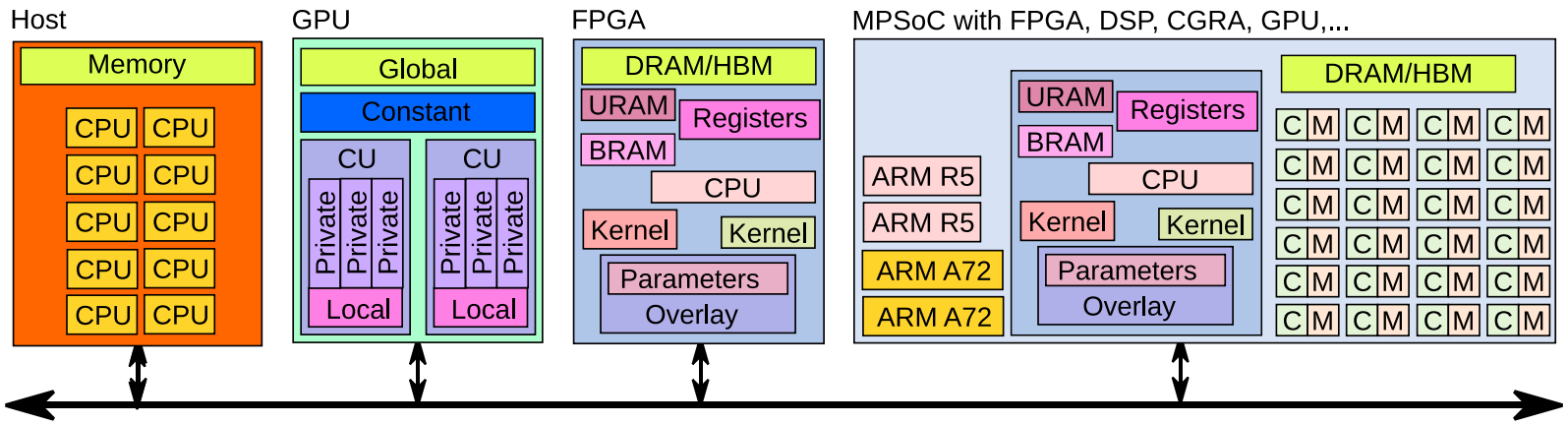
```
// Using USM only
#include <sycl/sycl.hpp>
#include <iostream>
using namespace sycl;
constexpr int N = 32;

int main () {
  queue q;
  int *a = malloc_shared<int>(N, q);
  q.parallel_for(N, [=](auto i) { a[i] = i; });
  q.wait();
  for (int i = 0; i < N; i++)
     std::cout << a[i] << std::endl;
}
```

SYCL 2020: reductions & group operations

```
buffer<int> valuesBuf { 1024 };
{
  host_accessor a { valuesBuf };
  std::iota(a.begin(), a.end(), 0);
}
int sumResult = 0;
buffer<int> sumBuf { &sumResult, 1 };
int maxResult = 0;
buffer<int> maxBuf { &maxResult, 1 };
myQueue.submit([&](handler& cgh) {
  auto inputValues = valuesBuf.get_access<access_mode::read>(cgh);
  auto sumReduction = reduction(sumBuf, cgh, plus<>());
  auto maxReduction = reduction(maxBuf, cgh, maximum<>());
  cgh.parallel_for(range<1>{1024}, sumReduction, maxReduction,
    [=](id<1> idx, auto& sum, auto& max) {
      sum += inputValues[idx];
      max.combine(inputValues[idx]);
  });
});
assert(maxBuf.get_host_access()[0] == 1023 && sumBuf.get_host_access()[0] == 523776);
```

SYCL 2020: Multiple backends



- **Need to work with any framework**
  - Not only OpenCL like old SYCL
- **Need to control implementation details for maximum performance**

Execution on Intel CPU + Xilinx FPGA (OpenCL) + Nvidia GPU (CUDA)

```
#include <iostream>
#include <sycl/sycl.hpp>
int main() {
 sycl::buffer<int> v { 10 };

 auto run = [&](auto sel, auto work) {
   sycl::queue { sel }.submit([&](auto& h) {
     auto a = sycl::accessor { v, h };
     h.parallel_for(a.get_count(), [=](auto i) { work(i, a); });
   });
 };
 run(sycl::host_selector {}, [](auto i, auto a) { a[i] = i; }); // CPU
 run(sycl::accelerator_selector {}, [](auto i, auto a) { a[i] = 2*a[i]; }); // FPGA
 run(sycl::gpu_selector {}, [](auto i, auto a) { a[i] = a[i] + 3; }); // GPU

 sycl::host_accessor acc { v };
 for (int i = 0; i != v.get_count(); ++i)
   std::cout << acc[i] << ", ";
 std::cout << std::endl;
}
```

- No `template` or `typename` or `class` or… ☺
- Generic & type-safe
- No explicit data motion or boiler-plate code
- Different accelerators/vendors in same program!

clang++ -std=c++20 -fsycl -fsycl-targets=nvptx64-nvidia-cuda-sycldevice,fpga64_sw_emu FPGA_GPU_CPU.cpp -o FPGA_GPU_CPU

oneAPI DPC++ + triSYCL https://github.com/triSYCL/sycl

SYCL 2020 backend interoperability

- **Possible to build SYCL objects from native backend objects**

```
xrt::device d = open_Xilinx_VCK1500();
xrt::bo xrt_bo = get_some_AIE_or_FPGA_storage(d);
sycl::queue { sycl::make_device(d) }.submit([&](auto &cgh) {
  sycl::accessor acc { sycl::make_buffer(xrt_bo) };
  cgh.single_task([=] { /* some work on acc */ });
});
```

  - Allow integration of SYCL application into CUDA/OpenCL/L0/ROCm/Vulkan/XRT/ADF/AIR/…
  - Kernel code can call directly native code and libraries, like Nvidia TensorCore…

- **Possible to get underlying native backend object from SYCL object**

```
sycl::buffer<int> buf { 1000 };
auto xrt_bo = buf.get_native<sycl::backend::xrt>(buf);
my_super_optimized_HLS_or_RTL_FPGA_function(xrt_bo);
```

  - Allow direct integration of OpenCL/CUDA/L0/ROCm/Vulkan/XRT/… application into SYCL
  - Obvious to use native optimized libraries

Documentation & resources

- https://www.khronos.org/sycl
  - The specification is open-source! https://github.com/KhronosGroup/SYCL-Docs
- **FAQ** https://www.khronos.org/blog/sycl-2020-what-do-you-need-to-know
- **SYCL-related community resource web site** http://sycl.tech
  - Learning
  - Community
  - Implementations
  - Tools
  - Conferences
  - Applications

Conclusion

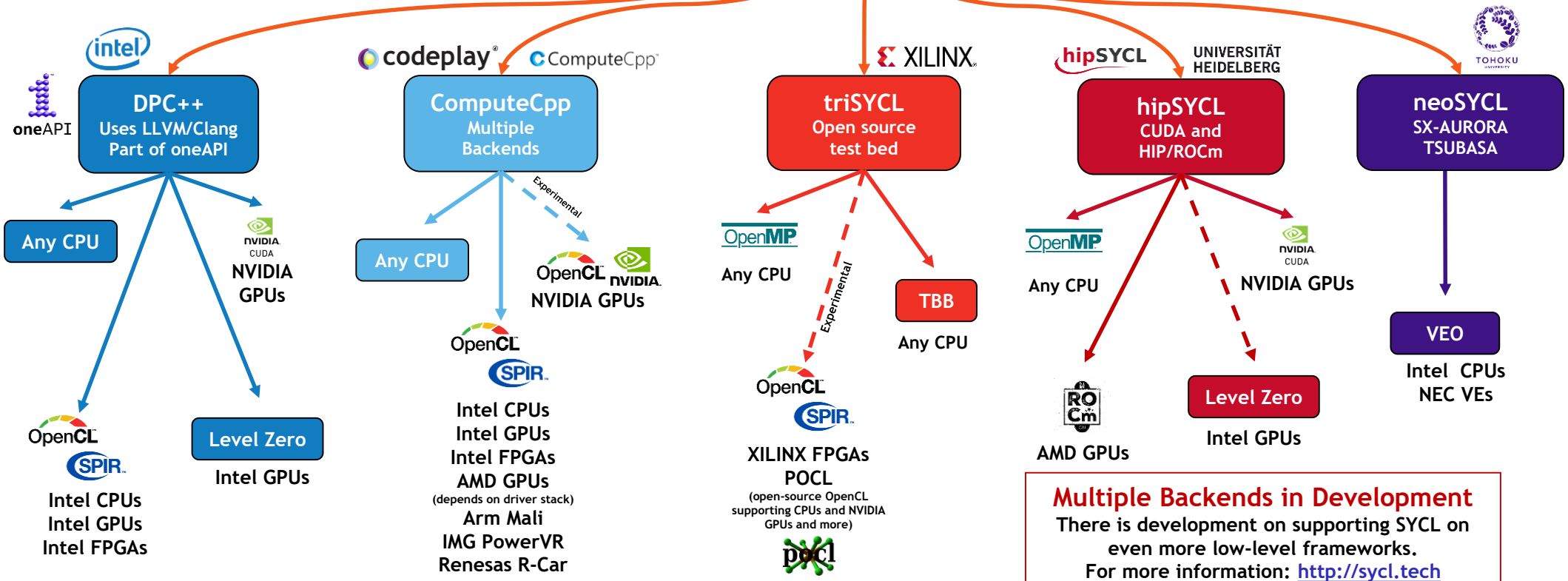
- **40+ new features with SYCL 2020**
- **The most inclusive C++ standard for heterogeneous computing**
  - Any device from any vendor at the same time
- **Backends & extensions for maximum performance**
  - No transistor left behind!
  - Leverage existing optimized libraries & tools/frameworks
- **Single-source C++ for simplicity and safety**
  - You deserve it!
- **Modern C++ but still plain C++**
  - Compatible with old C & C++
  - Interoperable with usual embedded & HPC frameworks (Fortran!)
  - Debug on CPU with usual software tools
- **Try it, adopt it & get involved into the standard!**

And now some news from the implementors…

SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies

SYCL Source Code

SYCL enables Khronos to influence ISO C++ to (eventually) support heterogeneous compute



Multiple Backends in Development
There is development on supporting SYCL on even more low-level frameworks.
For more information: http://sycl.tech

+ Celerity: SYCL on MPI+SYCL

- https://www.khronos.org/blog/sycl-2020-what-do-you-need-to-know