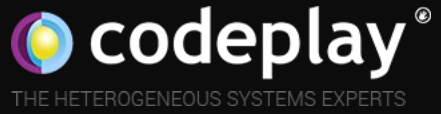


Extending TensorFlow to Heterogeneous Processors Using SYCL

Ralph Potter, Senior Research Engineer

RC4DL: Reconfigurable Computing for Deep Learning @ FPL'17

Leadership Products Enabling Advanced Applications on Complex Processor Systems

Company

High-performance software solutions for custom heterogeneous systems

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

Established 2002 in Scotland, UK

Markets

Vision Processing
Machine Learning
Data Compute
High Performance Computing (HPC)
Automotive (ISO 26262)
IoT, Smartphones & Tablets
Medical & Industrial

Products

ComputeCpp™

C++ platform with SYCL™, enabling vision & machine learning e.g. TensorFlow™

ComputeAorta™

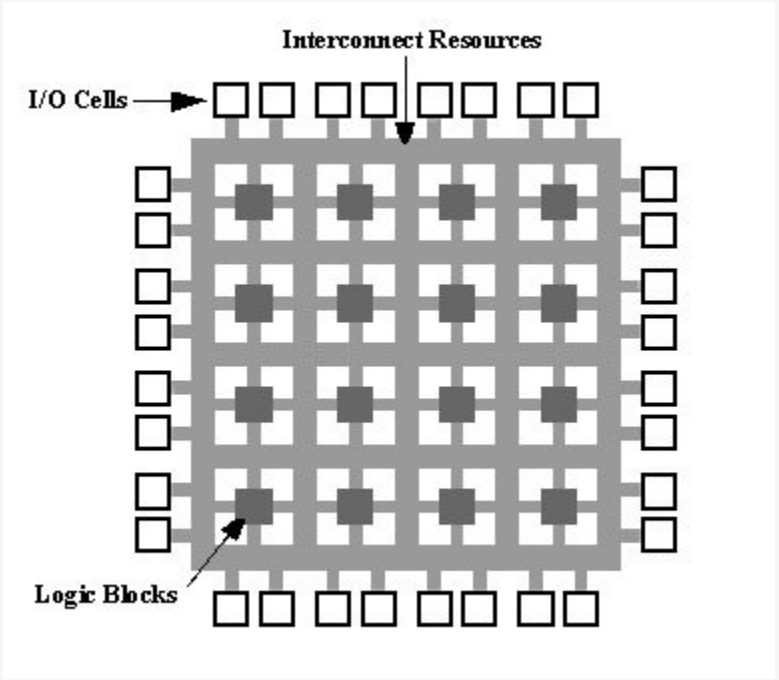
The heart of Codeplay's compute technology, enabling OpenCL™, SPIR™, HSA™ and Vulkan™

Customers/Partners

AMD
ARM
BROADCOM
Imagination
QUALCOMM
Movidius
Many Global Companies

Bridging the Gap

```
# Conv2D, with bias and relu
activation
conv = tf.nn.conv2d(conv, W1, ...)
conv = tf.nn.bias_add(conv, b1)
conv = tf.nn.relu(conv)
conv = tf.nn.max_pool(conv, ...)

# Fully connected layer
fc = tf.reshape(conv, ...)
out = tf.add(tf.matmul(fc, W2, ...),
                       b2)
```



https://commons.wikimedia.org/wiki/Category:Field_Programmable_Gate_Array_architecture#/media/File:Fpga1a.gif

This Presentation Will Cover:

- Work on bringing OpenCL acceleration to TensorFlow
- The tools required to make that possible
- Speculation on the applicability of this approach to FPGAs

Some Assembly Required

- All of the components described in this talk exist today
- Some of the components have been combined by teams at various companies...
  - Altera/Intel, Codeplay, Google, Xilinx
- ...but to my knowledge, no one has built the full stack

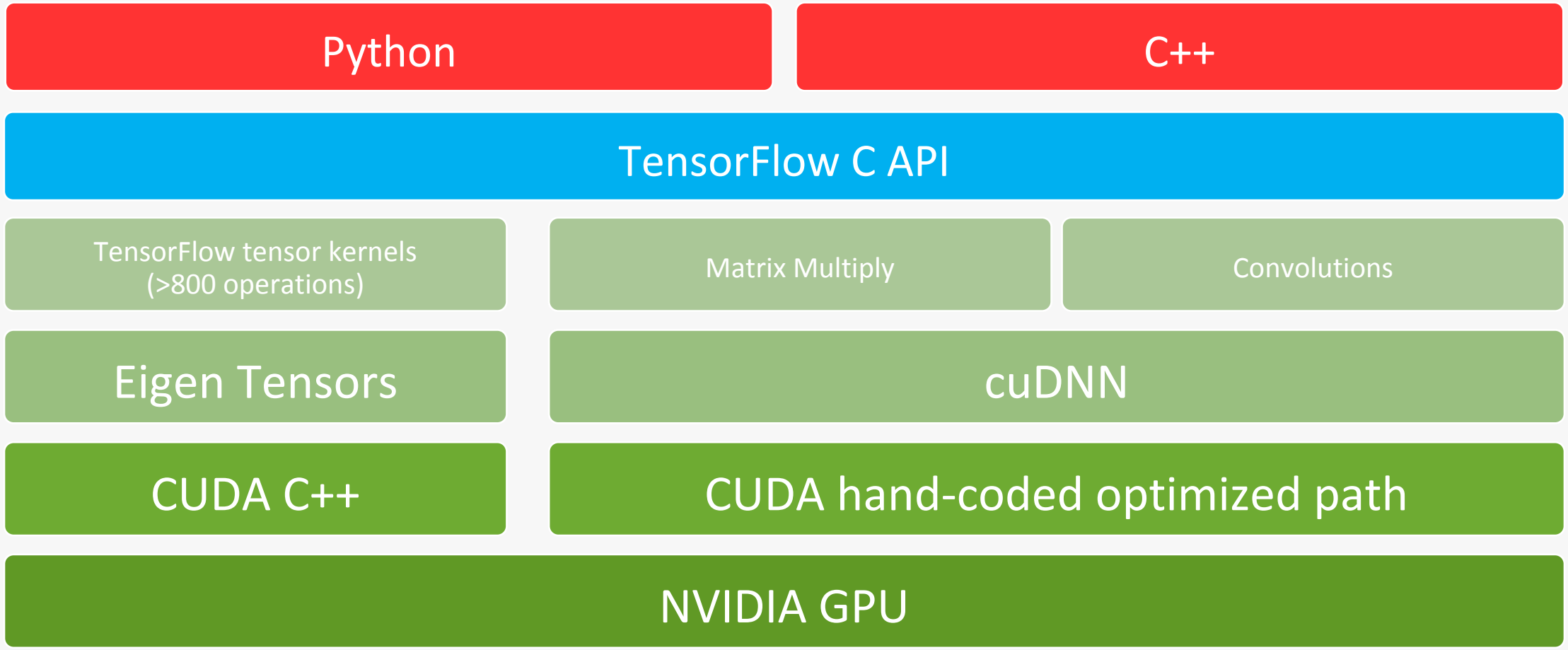
TensorFlow

- Widely used machine learning framework
- Initial design for distributed multi-core CPU systems
- Support for NVIDIA GPUs added later
- Limited support for Hexagon DSP and ARM Neon
- How do we add support for additional hardware devices, including FPGAs?

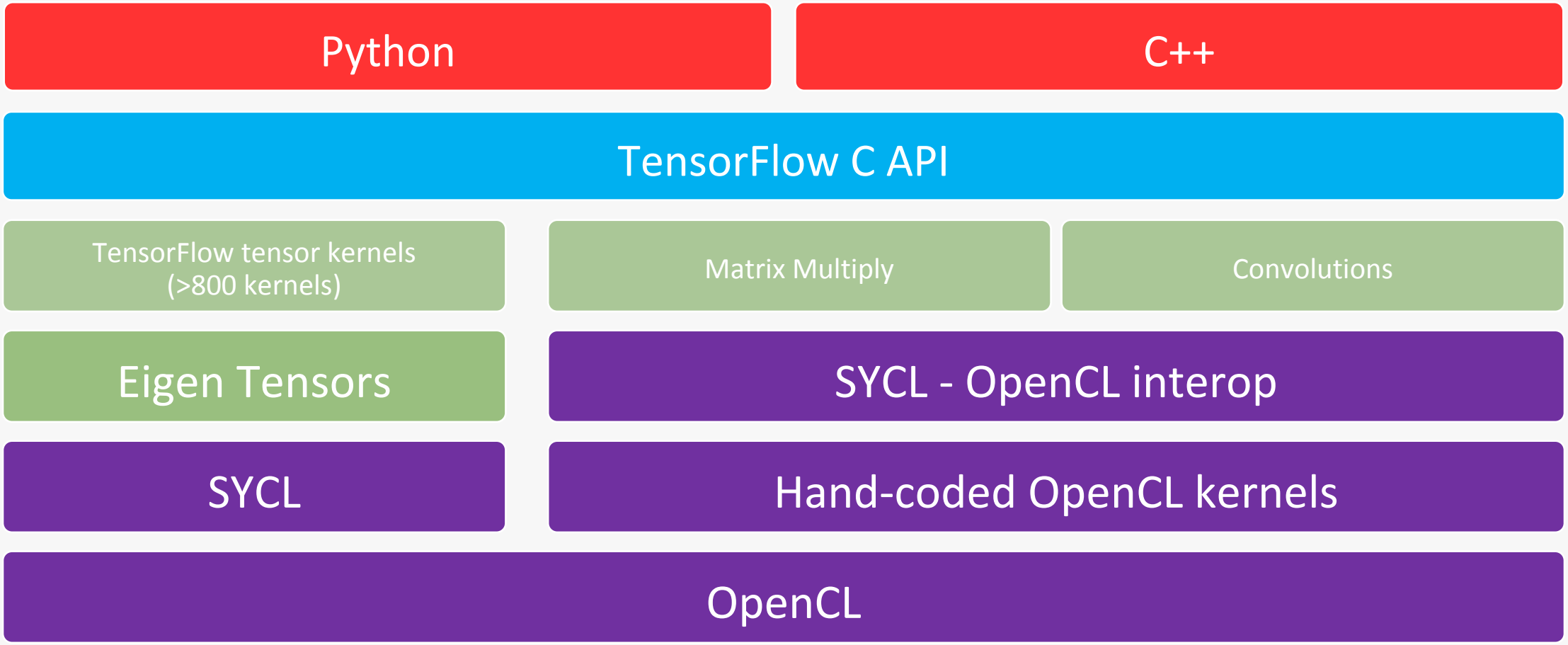
TensorFlow

- A graph constructed at runtime from a set of computational nodes
- Implementation of each node is static
- Runtime graph construction is not a good fit for hardware synthesis, but a per-node approach would be possible

TensorFlow on CUDA

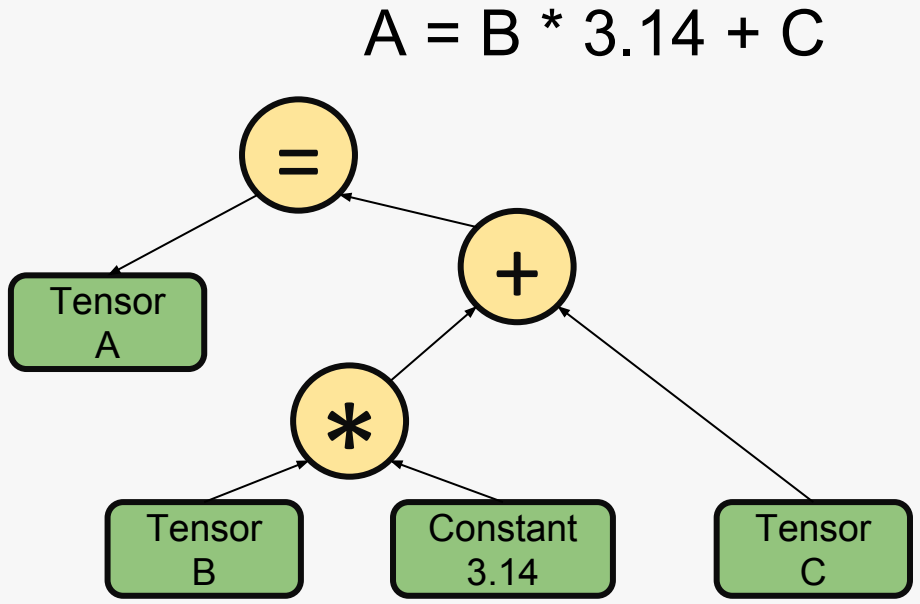


TensorFlow on Open Standards



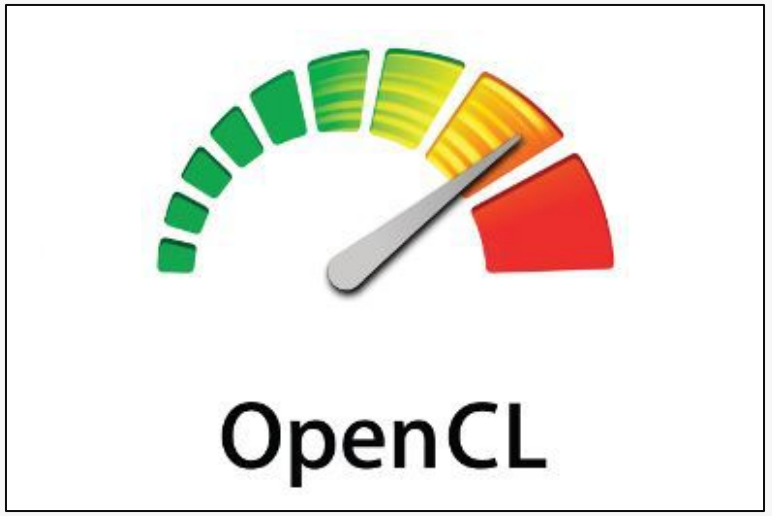
Eigen and TensorFlow

- Most computational nodes in TensorFlow are implemented as expressions in Eigens EDSL
- Encodes the structure and semantics of an expression in the C++ type system at compile time.
- A separate evaluator class provides the scheduling for a specific backend

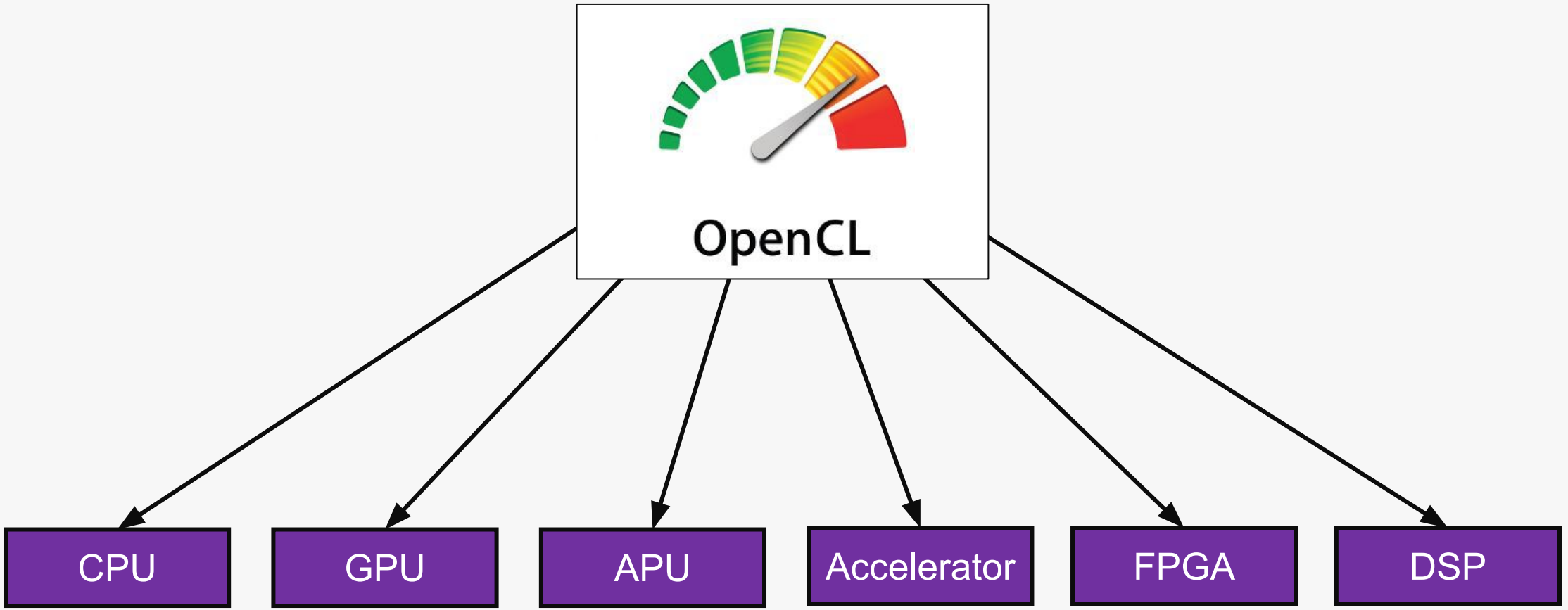



```
cl::sycl::gpu_selector s; // use OpenCL GPU
cl::sycl::queue q(s);
Eigen::SyclDevice sycl_device(q);

TensorMap<Tensor<DataType, 3, DataLayout>> A(gpu_in1_data, tensorRange);
TensorMap<Tensor<DataType, 3, DataLayout>> B(gpu_in2_data, tensorRange);
TensorMap<Tensor<DataType, 3, DataLayout>> C(gpu_in3_data, tensorRange);

// A = B * 3.14f + b*2.7f
A.device(sycl_device) =  B * B.constant(3.14f) + C;
```

OpenCL for FPGAs



SYCL Targets a Wide Range of OpenCL Devices



OpenCL for FPGAs

- Both of the major FPGA vendors provide OpenCL implementations
- Improved productivity over HDL…
  - …but not always performance parity
- OpenCL provides platform portability…
  - …but not necessarily performance portability
  - FPGA optimization guidance differs significantly from GPUs

Why not OpenCL C?

- Majority of TensorFlow's computational operators are implemented as expression trees encoded in the C++ type system
- Re-implementing all operators in OpenCL C is not tractable
  - Hundreds of operations, further parameterized by data type
  - Maintaining a separate OpenCL C code base presents a huge ongoing maintenance cost
- Libraries such as VexCL have handled similar use cases through runtime construction of kernel strings
  - A poor match for hardware synthesis

SYCL

- Open Standard from the Khronos Group
- Single-source C++ abstraction layer for OpenCL
  - Abstractions for all OpenCL features
  - Interop functionality for existing OpenCL applications
- Automatic management of data movement
- C++ template metaprogramming on host and accelerators
- Offline compilation
- SYCL 2.2 includes support for OpenCL pipes and shared virtual memory

Implementations

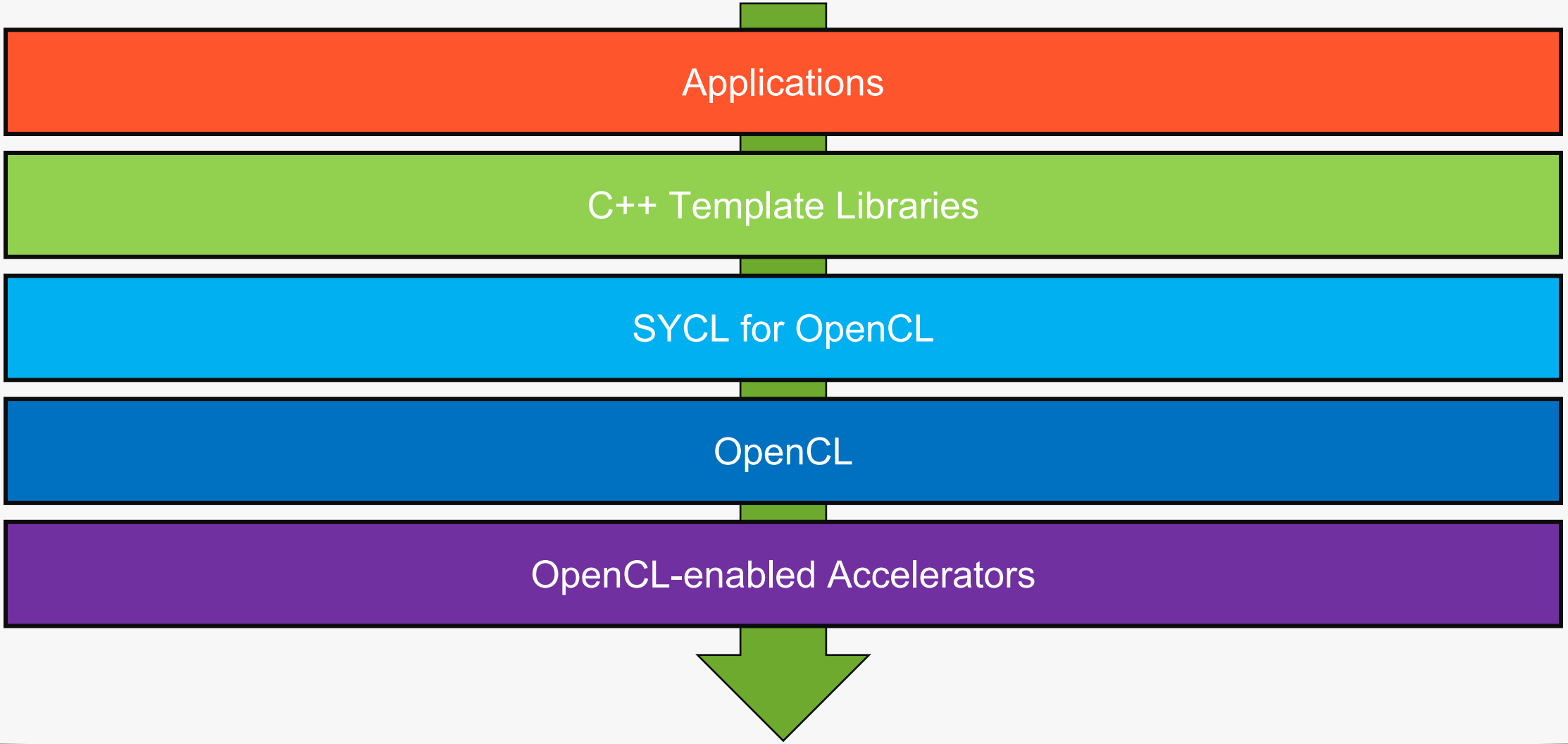
TriSYCL - Xilinx Research Labs

- OpenMP-based, with OpenCL interop support
- Open-source - https://github.com/triSYCL/triSYCL

ComputeCpp - Codeplay

- Standard OpenCL runtimes, requires SPIR or IHV compiler backend
- Free to end users, licensed to IHVs - https://computecpp.codeplay.com

SYCL Ecosystem



SYCL Example

```
// Create a device queue.
cl::sycl::queue device_queue;
// Create buffers.
cl::sycl::range<1> n_items{array_size};
cl::sycl::buffer<cl::sycl::cl_int, 1> in_buffer(in.data(), n_items);
cl::sycl::buffer<cl::sycl::cl_int, 1> out_buffer(out.data(), n_items);
// Asynchronously submit a kernel and associated data movement operations.
device_queue.submit([&](cl::sycl::handler &cgh) {
    // Defines the kernels access requirements.
    auto in_accessor  = in_buffer.get_access<cl::sycl::access::mode::read>(cgh);
    auto out_accessor = out_buffer.get_access<cl::sycl::access::mode::write>(cgh);
    // Defines the kernel itself.
    cgh.parallel_for<class VecScalMul>(n_items, [=](cl::sycl::id<1> wiID) {
        out_accessor[wiID] = in_accessor[wiID]*2;
    });
});
```

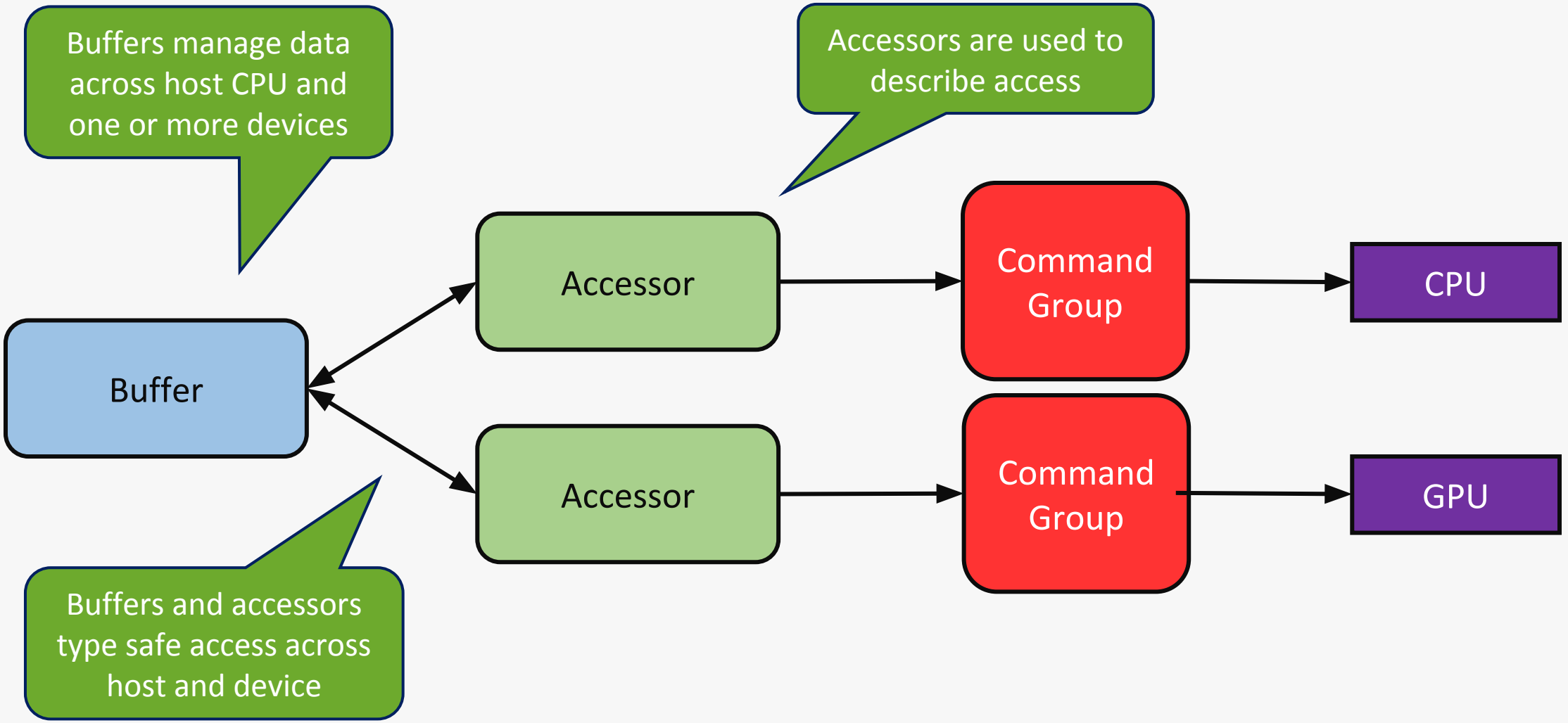
Different Forms of Parallelism

```
cgh.single_task<T>([=](){

  /* task executed by a single
     work item */

});
```

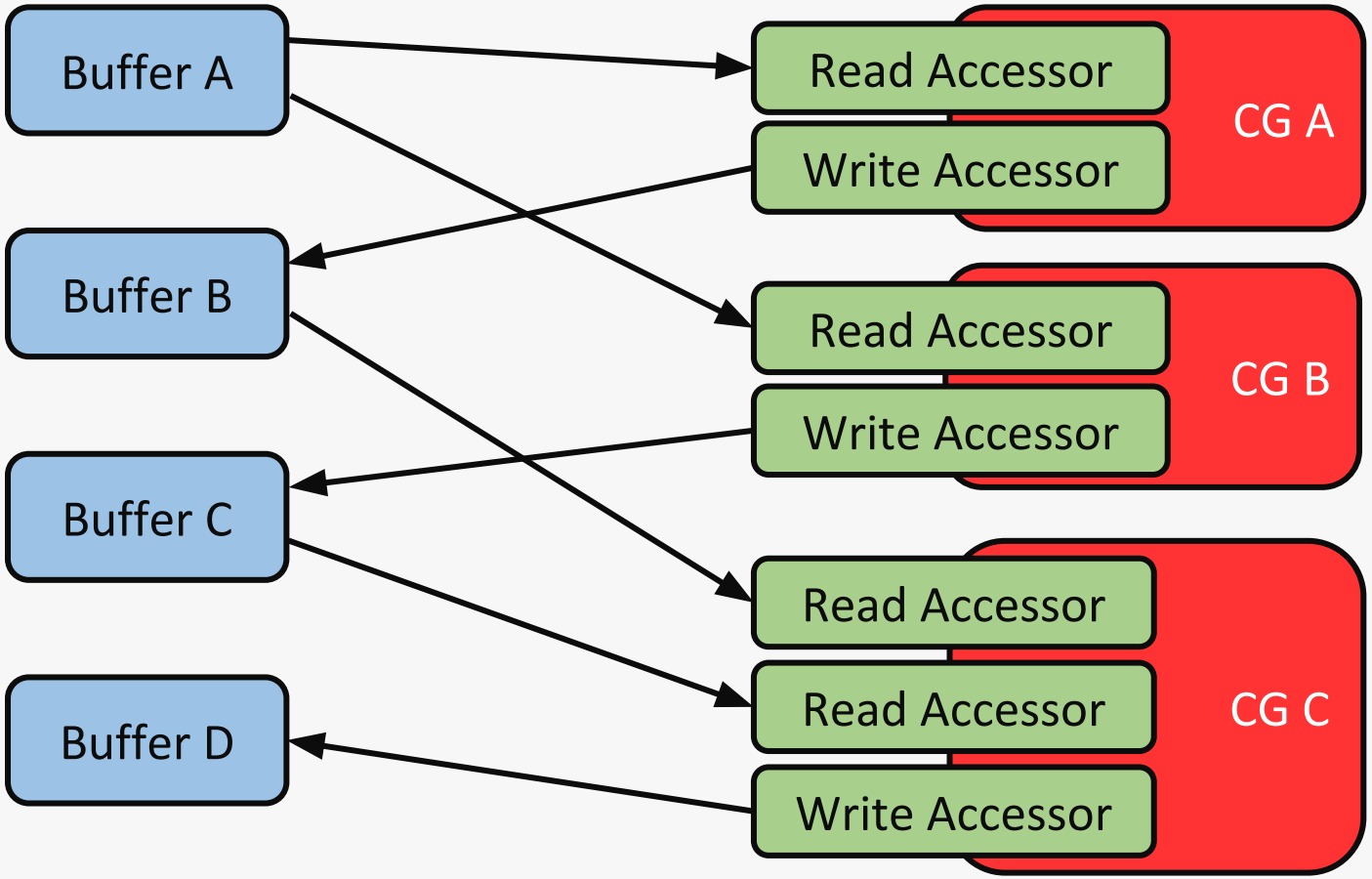
```
cgh.parallel_for<T>(range<2>(64, 64),
                    [=](id<2> idx){

  /* data parallel work executed
     across a range */

});
```

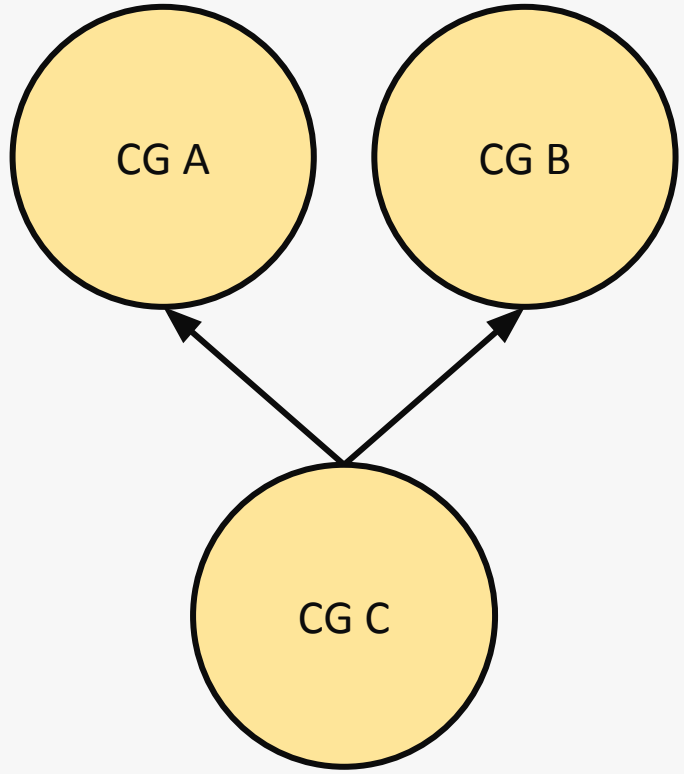
```
cgh.parallel_for_work_group(range<2>(64, 64),
                            [=](group<2> gp){

  /* data parallel work executed once
     per work group */

  parallel_for_work_item(gp, [=](item<2> it){

    /* data parallel work executed once
       per work item */

  });

});
```

Separating Storage & Access

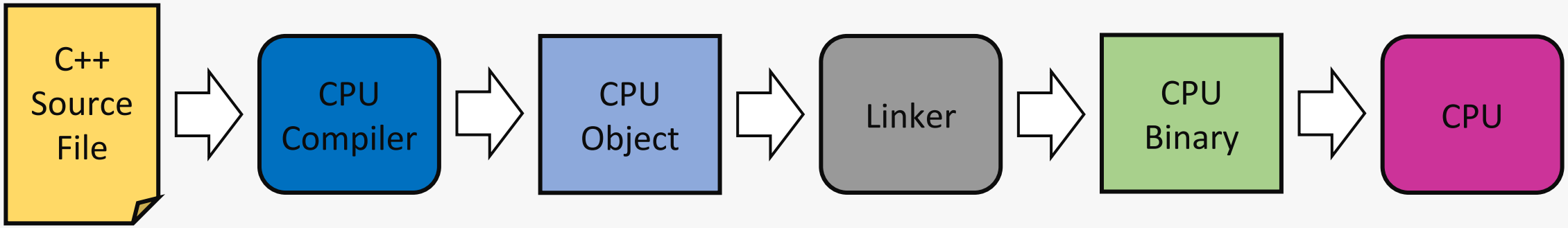


Data Dependency Task Graphs

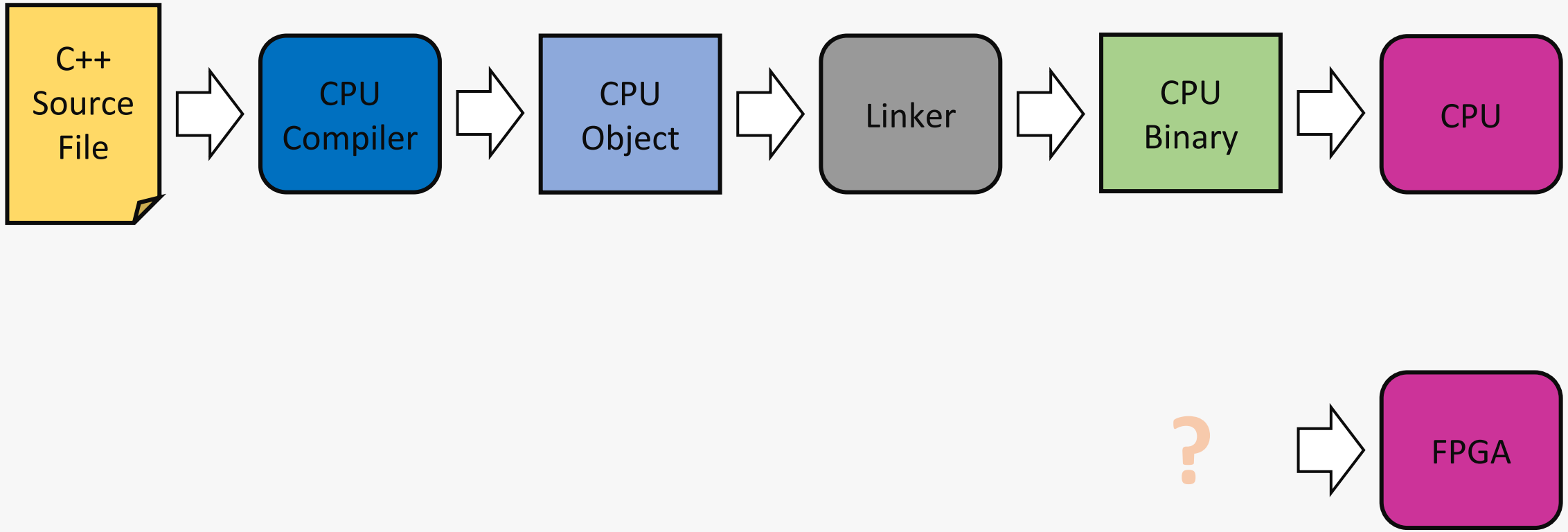




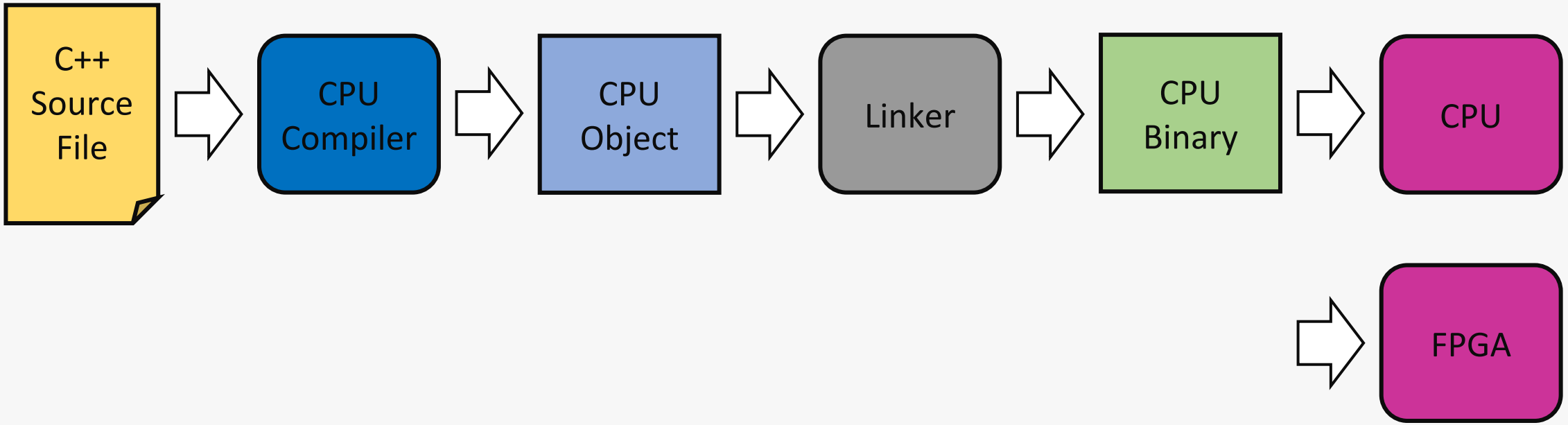
C++ Compilation Model



C++ Compilation Model

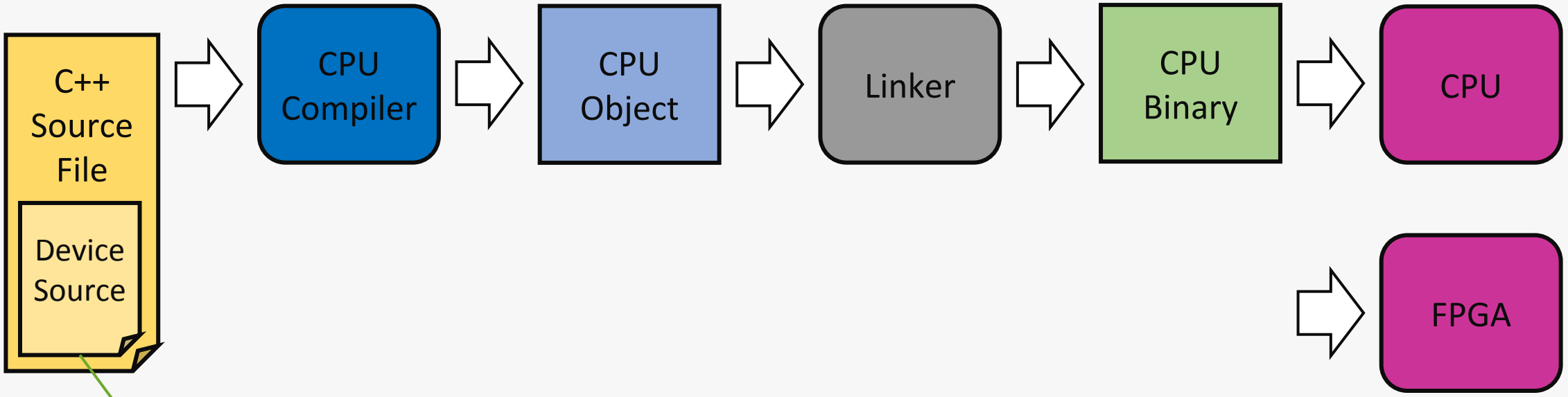


SYCL Compilation Model



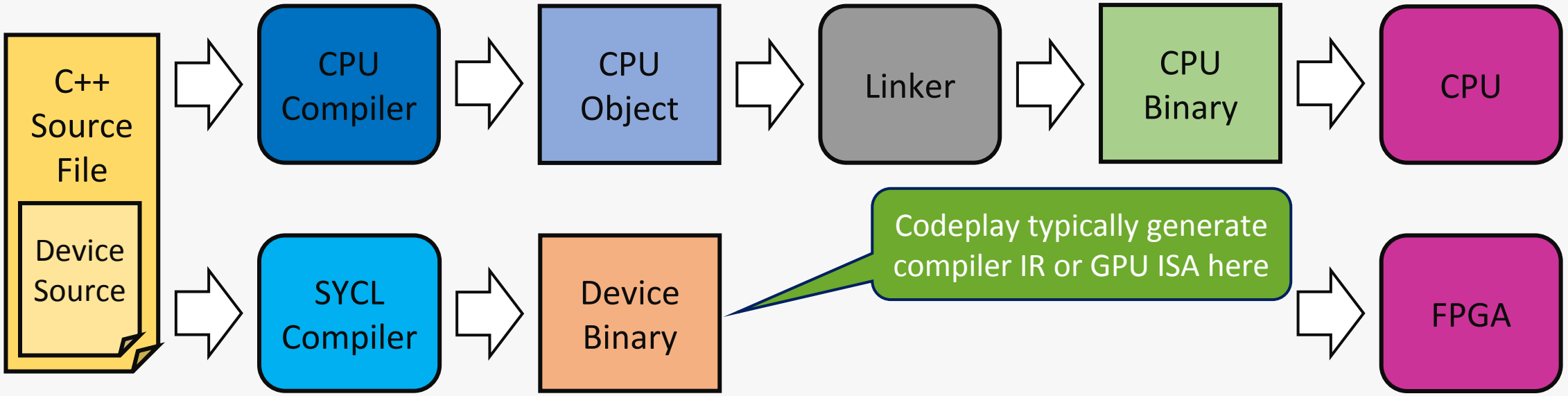

```
auto inA = bufA.get_access<access::mode::read>(cgh);
auto inB = bufB.get_access<access::mode::read>(cgh);
auto out = bufO.get_access<access::mode::write>(cgh);
cgh.parallel_for<class add>(range<2>(64, 64), [=](id<1> i) {
  out[i] = inA[i] + inB[i];
}));
```

SYCL Compilation Model



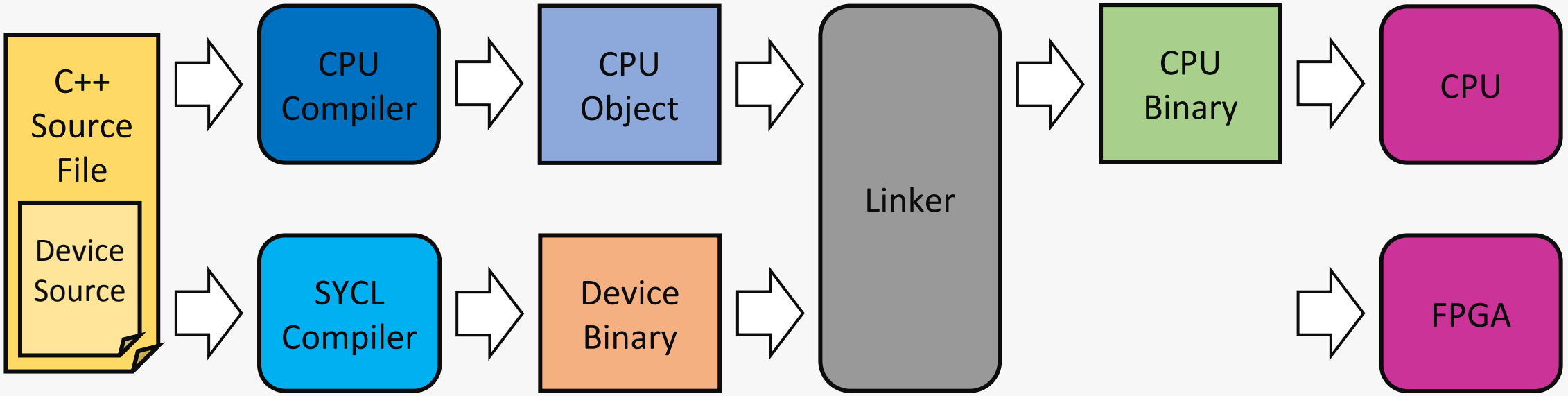

```
auto inA = bufA.get_access<access::mode::read>(cgh);
auto inB = bufB.get_access<access::mode::read>(cgh);
auto out = bufO.get_access<access::mode::write>(cgh);
cgh.parallel_for<class add>(range<2>(64, 64), [=](id<1> i) {
  out[i] = inA[i] + inB[i];
}));
```

SYCL Compilation Model



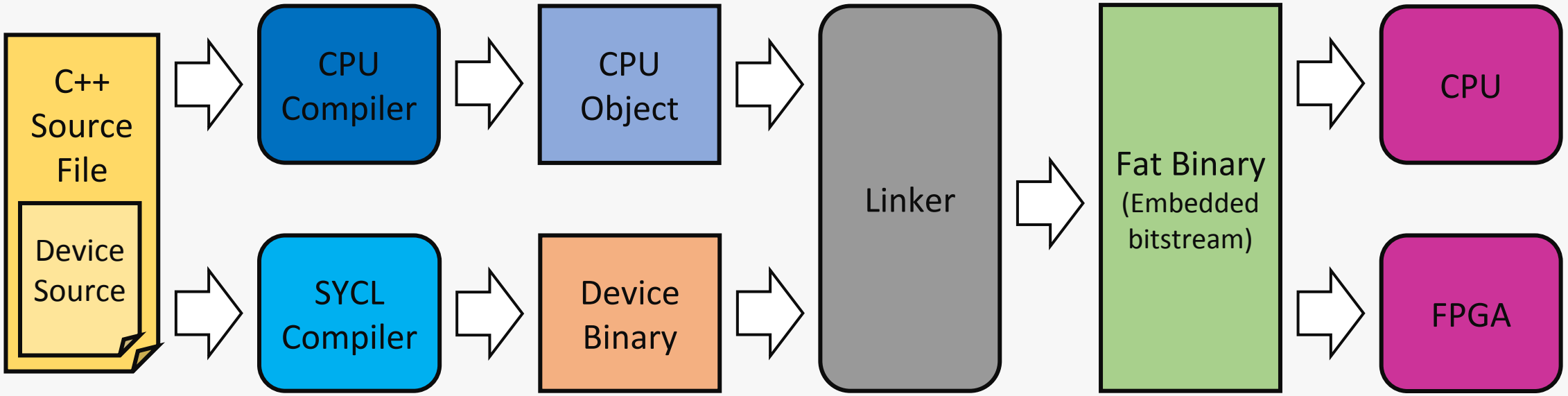

```
auto inA = bufA.get_access<access::mode::read>(cgh);
auto inB = bufB.get_access<access::mode::read>(cgh);
auto out = bufO.get_access<access::mode::write>(cgh);
cgh.parallel_for<class add>(range<2>(64, 64), [=](id<1> i) {
  out[i] = inA[i] + inB[i];
}));
```

SYCL Compilation Model




```
auto inA = bufA.get_access<access::mode::read>(cgh);
auto inB = bufB.get_access<access::mode::read>(cgh);
auto out = bufO.get_access<access::mode::write>(cgh);
cgh.parallel_for<class add>(range<1>(dA.size()), [=](id<1> i) {
  out[i] = inA[i] + inB[i];
}));
```

SYCL Compilation Model




```
auto inA = bufA.get_access<access::mode::read>(cgh);
auto inB = bufB.get_access<access::mode::read>(cgh);
auto out = bufO.get_access<access::mode::write>(cgh);
cgh.parallel_for<class add>(range<1>(dA.size()), [=](id<1> i) {
  out[i] = inA[i] + inB[i];
}));
```

Why SYCL?

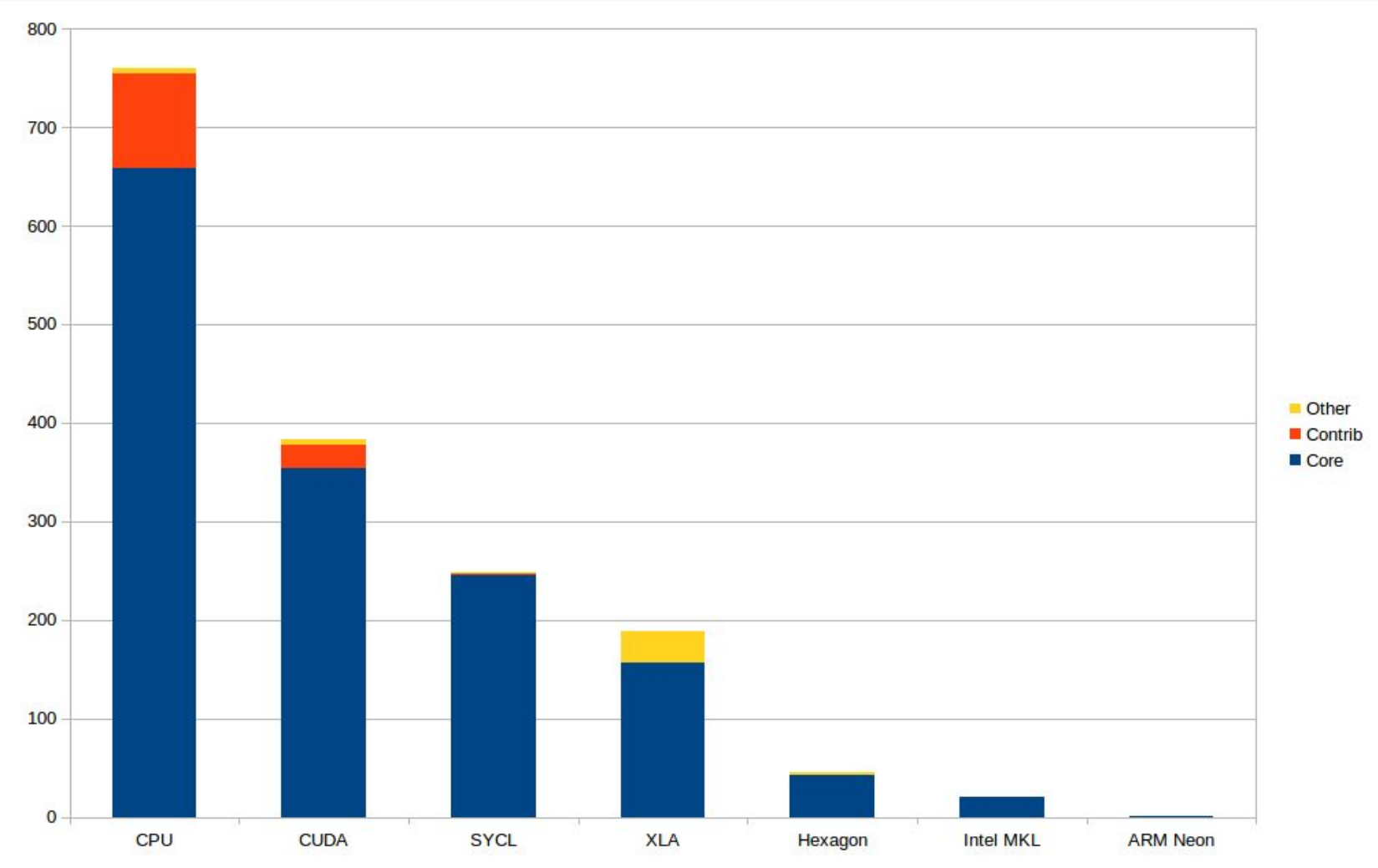
Makes heavy use of C++ expression trees

- Eigen tensors component
- Allows kernel fusion optimizations in C++

Alternative approach

- Extract TensorFlow graphs and recompile in a target-specific way
- This doesn't work for Google as it involves rewriting all their software

Implementation Status



SYCL for FPGAs?

- Modern single-source C++ gives us the tools to abstract optimization
- Expression templates allow us to statically generate kernels offline from user expressions
- TriSYCL has combined FPGA OpenCL kernels with SYCL scheduler
- ComputeCpp can be extended to any platform with an LLVM compiler

TensorFlow for FPGAs?

- Current TensorFlow and Eigen kernels are GPGPU-focused
  - Provide an alternative expression evaluator for FPGA kernels
- It seems unlikely that the entire set of kernels will fit on an FPGA
  - Restricting to the subset required for a single model seems tractable via SYCL

We're Hiring!
codeplay.com/careers/

codeplay
THE HETEROGENEOUS SYSTEMS EXPERTS

Questions?



@codeplaysoft info@codeplay.com codeplay.com

C++ Expression trees: Eigen Tensors

This creates an Eigen device

This code is the standard Eigen approach to linear algebra, which TensorFlow has adapted to tensors and accelerated with CUDA.

This is the code adapted to use SYCL

This expression is fused into a single kernel on the device

```
cl::sycl::gpu_selector s; // use OpenCL GPU
cl::sycl::queue q(s);
Eigen::SyclDevice sycl_device(q);

array<int, 3> tensorRange = {{100, 10, 20}};
Tensor<DataType, 3,DataLayout> in1(tensorRange);
Tensor<DataType, 3,DataLayout> in2(tensorRange);
Tensor<DataType, 3,DataLayout> in3(tensorRange);
Tensor<DataType, 3,DataLayout> out(tensorRange);

in2 = in2.random();
in3 = in3.random();

DataType *gpu_in1_data  = static_cast<DataType*>(sycl_device.allocate(in1.size()*sizeof(DataType)));
DataType *gpu_in2_data  = static_cast<DataType*>(sycl_device.allocate(in2.size()*sizeof(DataType)));
DataType *gpu_in3_data  = static_cast<DataType*>(sycl_device.allocate(in3.size()*sizeof(DataType)));
DataType *gpu_out_data =  static_cast<DataType*>(sycl_device.allocate(out.size()*sizeof(DataType)));

TensorMap<Tensor<DataType, 3, DataLayout>> gpu_in1(gpu_in1_data, tensorRange);
TensorMap<Tensor<DataType, 3, DataLayout>> gpu_in2(gpu_in2_data, tensorRange);
TensorMap<Tensor<DataType, 3, DataLayout>> gpu_in3(gpu_in3_data, tensorRange);
TensorMap<Tensor<DataType, 3, DataLayout>> gpu_out(gpu_out_data, tensorRange);

//a*3.14f + b*2.7f
gpu_out.device(sycl_device) =  gpu_in1 * gpu_in1.constant(3.14f) + gpu_in2 * gpu_in2.constant(2.7f);
sycl_device.memcpyDeviceToHost(out.data(),gpu_out_data,(out.size())*sizeof(DataType));
sycl_device.synchronize();
```

Tensor operation as functor on device

Tensor expression is in *Expr* type

Compile-time template magic to reconstruct tensor expression inside kernel

Evaluate element of tensor expression

```
template<typename Expr, typename FunctorExpr, typename TupleType >
struct ExecExprFunctorKernel {
  typedef typename internal::createPlaceHolderExpression<Expr>::Type PlaceHolderExpr;

  typedef typename Expr::Index Index;
  FunctorExpr functors;
  TupleType tuple_of_accessors;
  Index range;
  ExecExprFunctorKernel (Index range_, FunctorExpr functors_, TupleType tuple_of_accessors_)
    : functors (functors_), tuple_of_accessors (tuple_of_accessors_), range (range_){}

  void operator()(cl::sycl::nd_item<1> itemID) {
    typedef typename internal::ConvertToDeviceExpression<Expr>::Type DevExpr;
    auto device_expr = internal::createDeviceExpression<DevExpr, PlaceHolderExpr>(functors,
                                                                                  tuple_of_accessors);
    auto device_evaluator = Eigen::TensorEvaluator<decltype(device_expr.expr)
                                                   Eigen::DefaultDevice>(device_expr.expr,
                                                                         Eigen::DefaultDevice());
    typename DevExpr::Index gId = static_cast<typename DevExpr::Index> (
                                                  itemID.get_global_linear_id ());
    if (gId < range)
      device_evaluator.evalScalar(gId);
  }
};
```