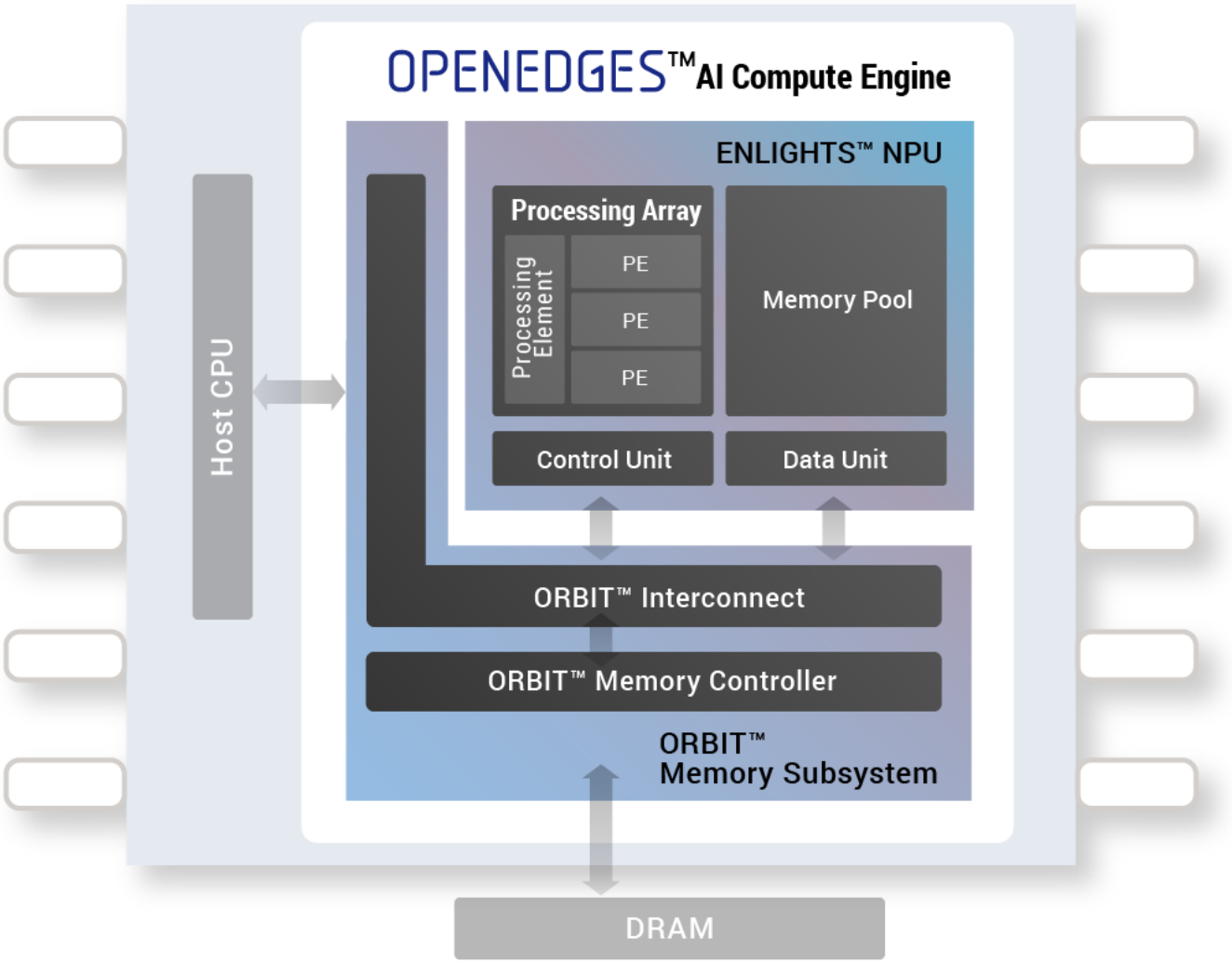
XLA 101

Tee Jung

tee.ty.jung@openedges.com



- Reference NN Models
- Quantization Simulator
- Quantize Aware Trainer
- Quantizer
- Compiler
- NPU Driver

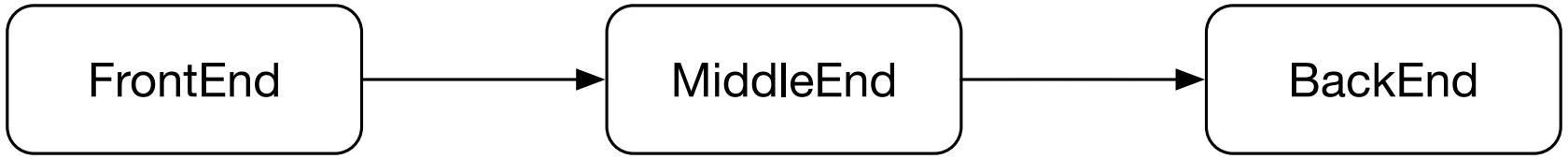
https://openedges.com/

XLA

- Optimizing Compiler for Machine Learning
- Supports:
  - Ahead-Of-Time Compilation
  - Just-In-Time Compilation

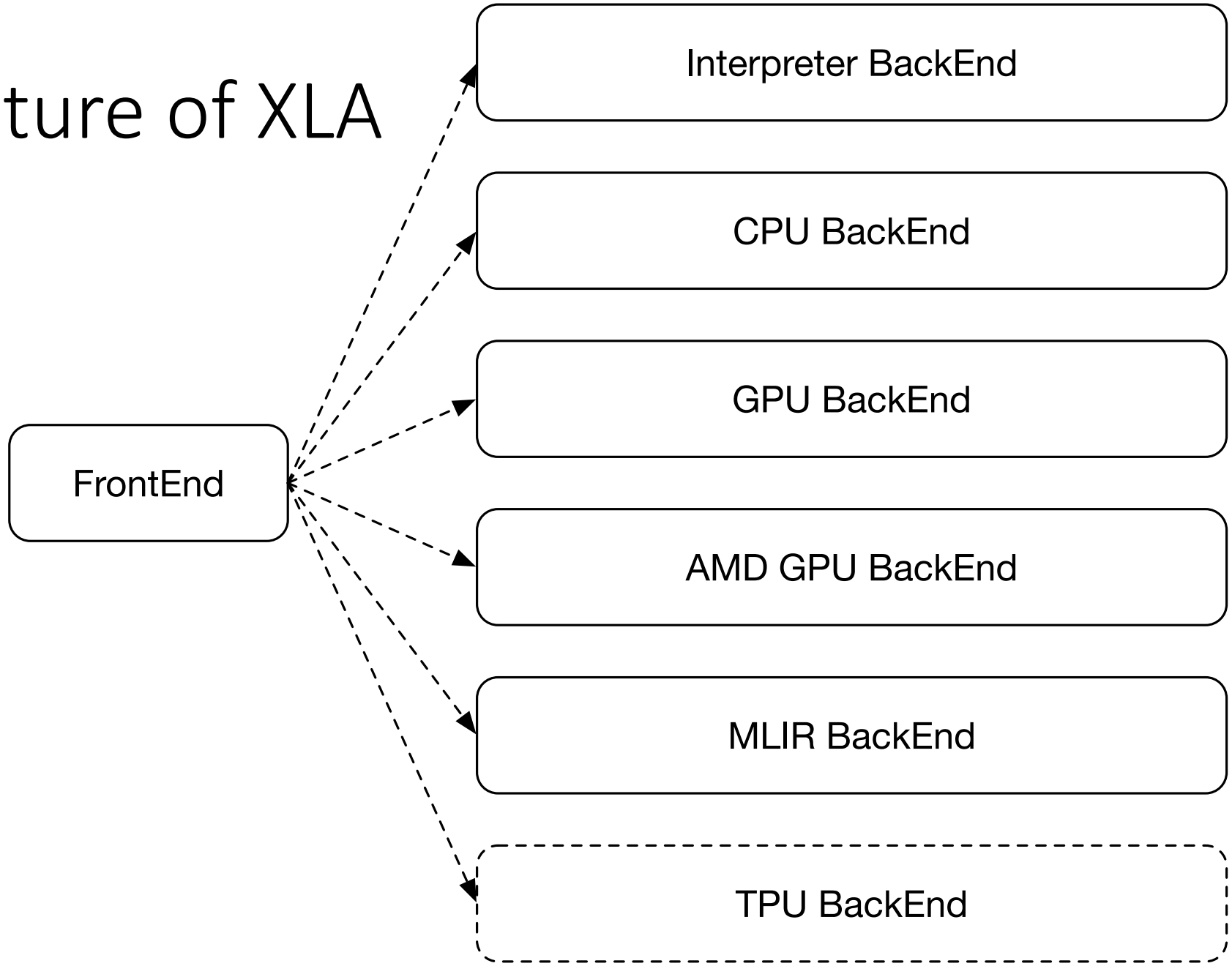


Structure of Compiler

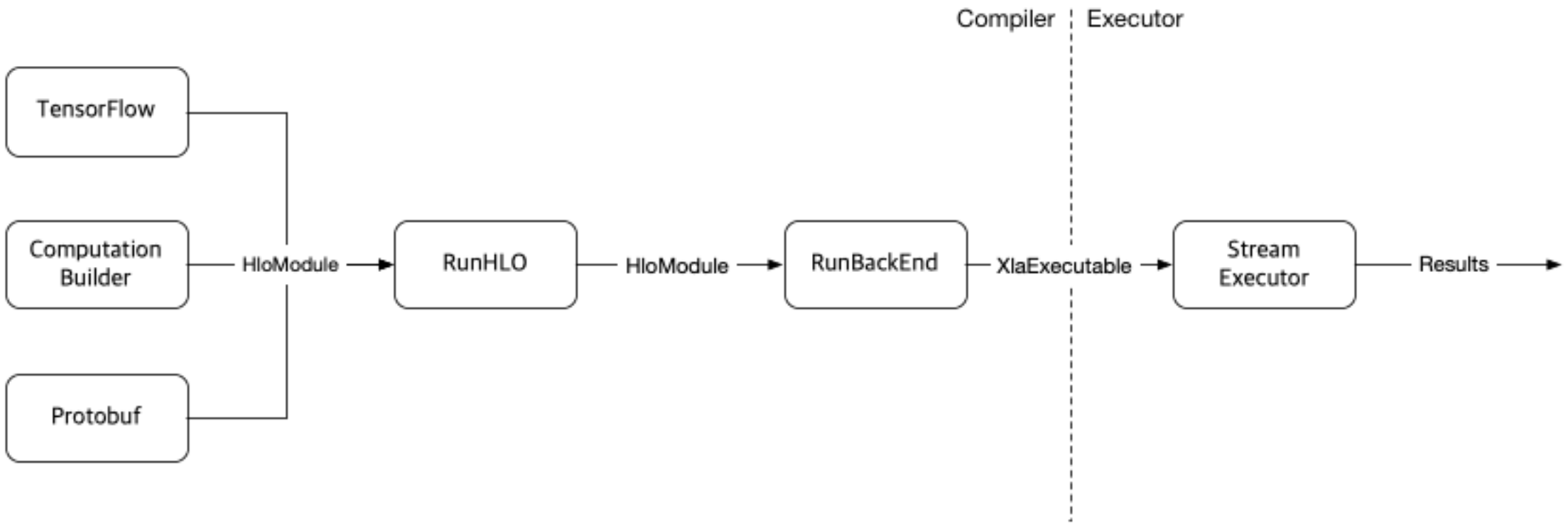


Structure of XLA

Structure of XLA

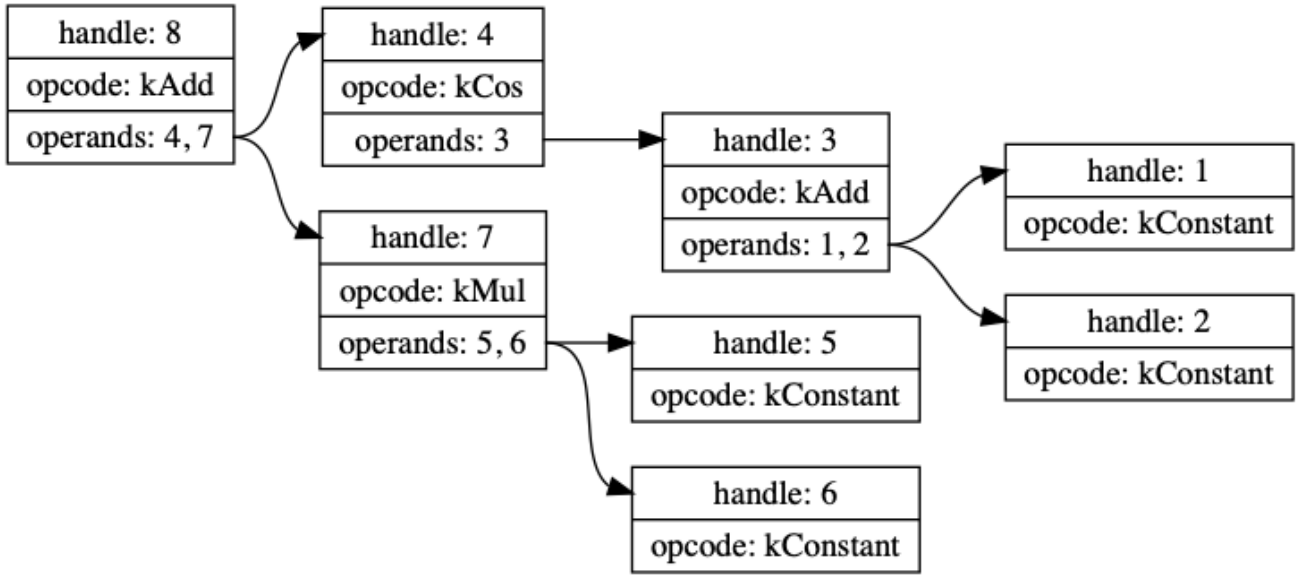


Structure of XLA



Structure of XLA: FrontEnd

- Building HloModule



Structure of XLA: MiddleEnd

- High-Level Optimization (HLO)
  - DeadCodeElimination
  - ConstantFolding
    - *(x + 1 + 2 + 3)* → *(x + 6)*
  - Flattening
    - *(a + b + c)* → *AddN(a, b, c)*
  - Hoisting
    - *(x*a + x*b)* → *x * (a + b)*
  - Simplication
    - *!(x > y)* → *(x <= y)*
  - Broadcast Minimization
    - *(Mtx1 + Scalar1) + (Mtx2 + Scalar2)* → *(Mtx1 + Mtx2) + (Scalar1 + Scalar2)*

See also: https://web.stanford.edu/class/cs245/slides/TFGraphOptimizationsStanford.pdf

Structure of XLA: MiddleEnd (Cont.)

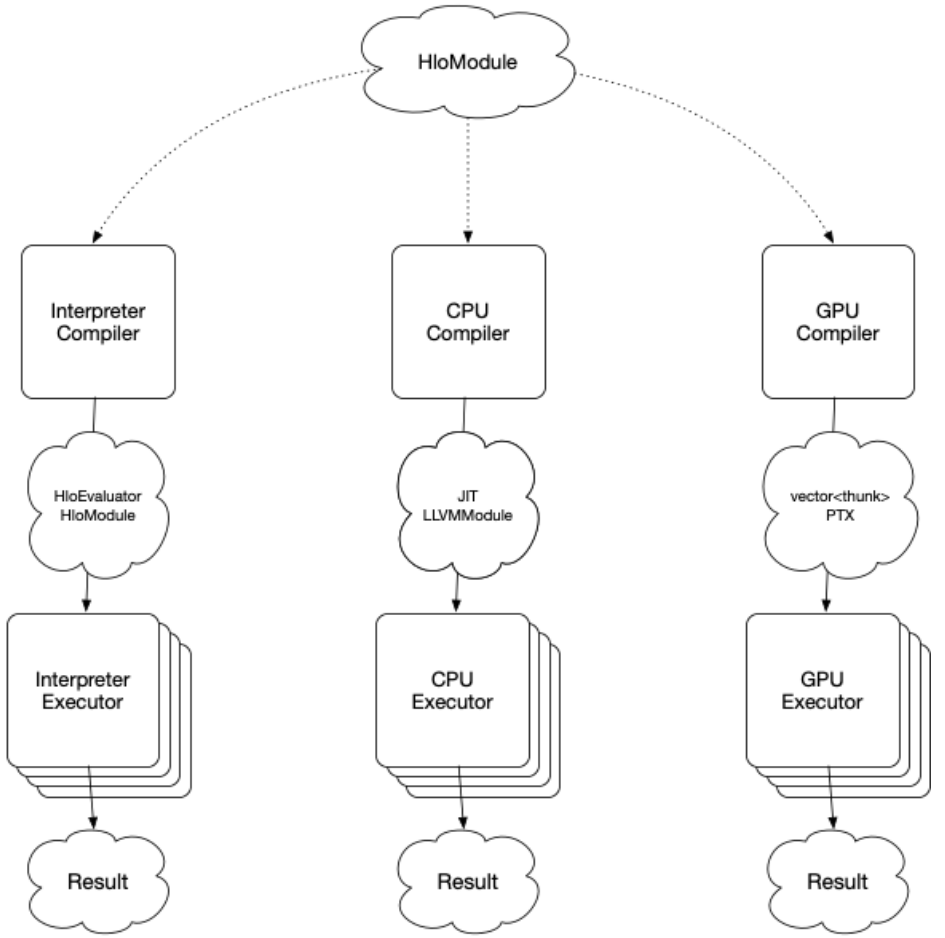
```
403  Status CpuCompiler::RunHloPassesAfterLayoutAssn(
404      HloModule* module, bool is_aot_compile,
405      LLVMTargetMachineFeatures* target_machine_features) {
406    HloPassPipeline pipeline("HLO passes after layout assignment");
407    // After layout assignment, use a layout-sensitive verifier.
408
409    pipeline.AddPass<HloPassPipeline>("after layout assignment")
410        .AddInvariantCheckerDebug<HloVerifier>(
411            /*layout_sensitive=*/true,
412            /*allow_mixed_precision=*/false);
413
414    // The LayoutAssignment pass may leave behind kCopy instructions which are
415    // duplicate or NOPs, so remove them with algebraic simplification and CSE.
416    {
417      auto& pass = pipeline.AddPass<HloPassFix<HloPassPipeline>>(
418          "simplification after layout assignment");
419      pass.AddInvariantCheckerDebug<HloVerifier>(
420          /*layout_sensitive=*/true,
421          /*allow_mixed_precision=*/false,
422          LayoutAssignment::InstructionCanChangeLayout);
423      AlgebraicSimplifierOptions options;
424      options.set_is_layout_sensitive(true);
425      options.set_enable_dot_strength_reduction(false);
426      pass.AddPass<HloPassFix<AlgebraicSimplifier>>(options);
427      pass.AddPass<HloDCE>();
428      pass.AddPass<HloCSE>(/*is_layout_sensitive=*/true);
429    }
```

Structure of XLA: MiddleEnd (Cont.)

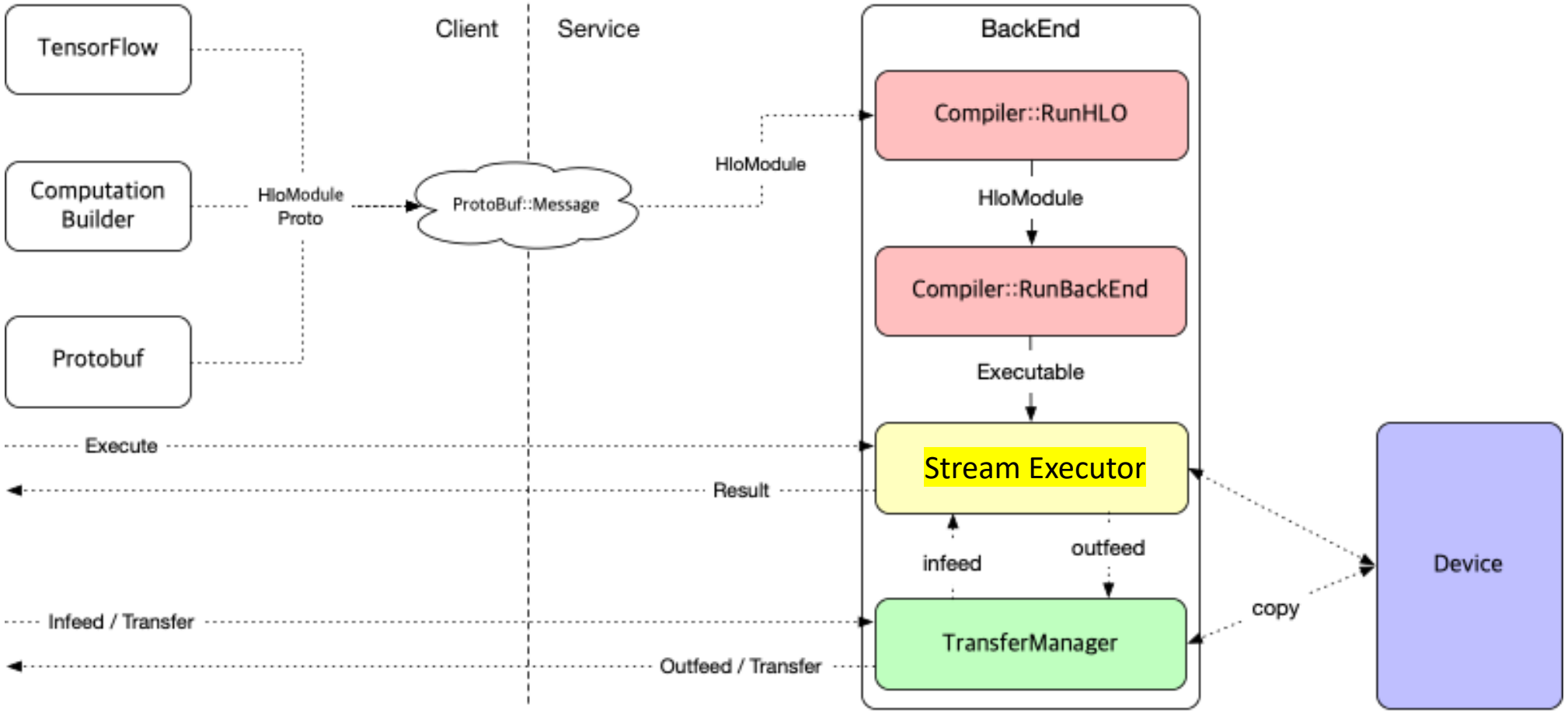
```
391     // Layout assignment uses alias analysis, which requires the call graph to be
392     // flattened.
393     pipeline.AddPass<FlattenCallGraph>();
394     pipeline.AddPass<CpuLayoutAssignment>(
395         module->mutable_entry_computation_layout(),
396         LayoutAssignment::InstructionCanChangeLayout, target_machine_features);
397
398     pipeline.AddPass<CpuInstructionFusion>();
399
400     return pipeline.Run(module).status();
401   }
```

Structure of XLA: BackEnd

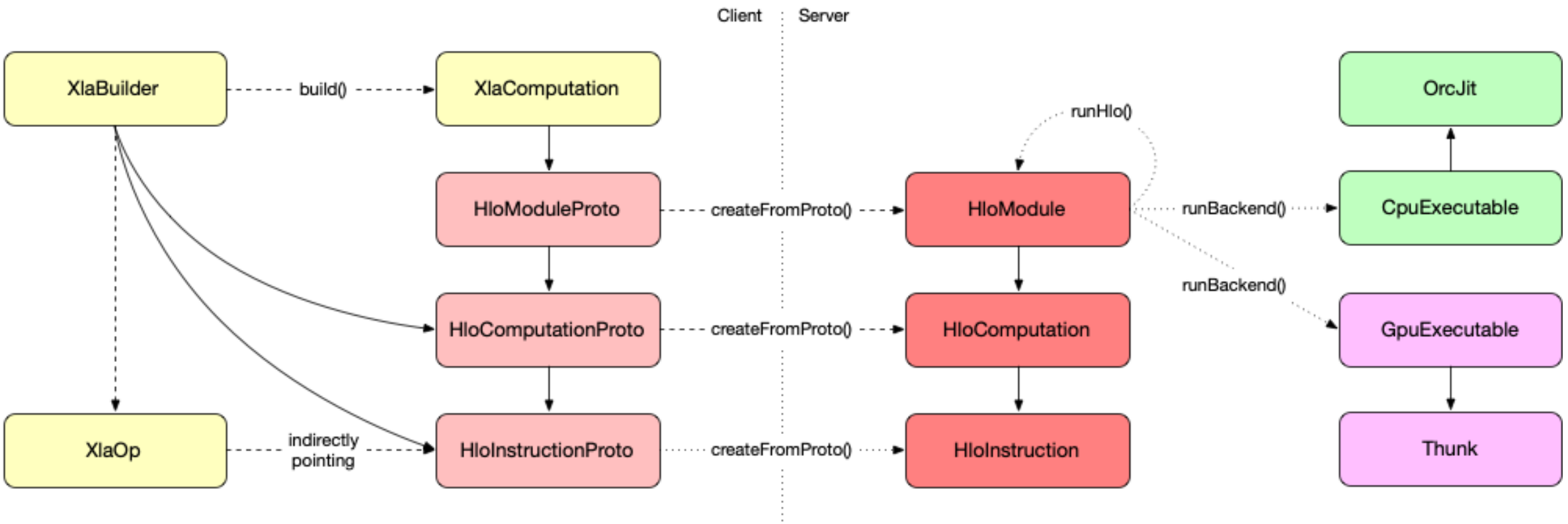
- Code Generation



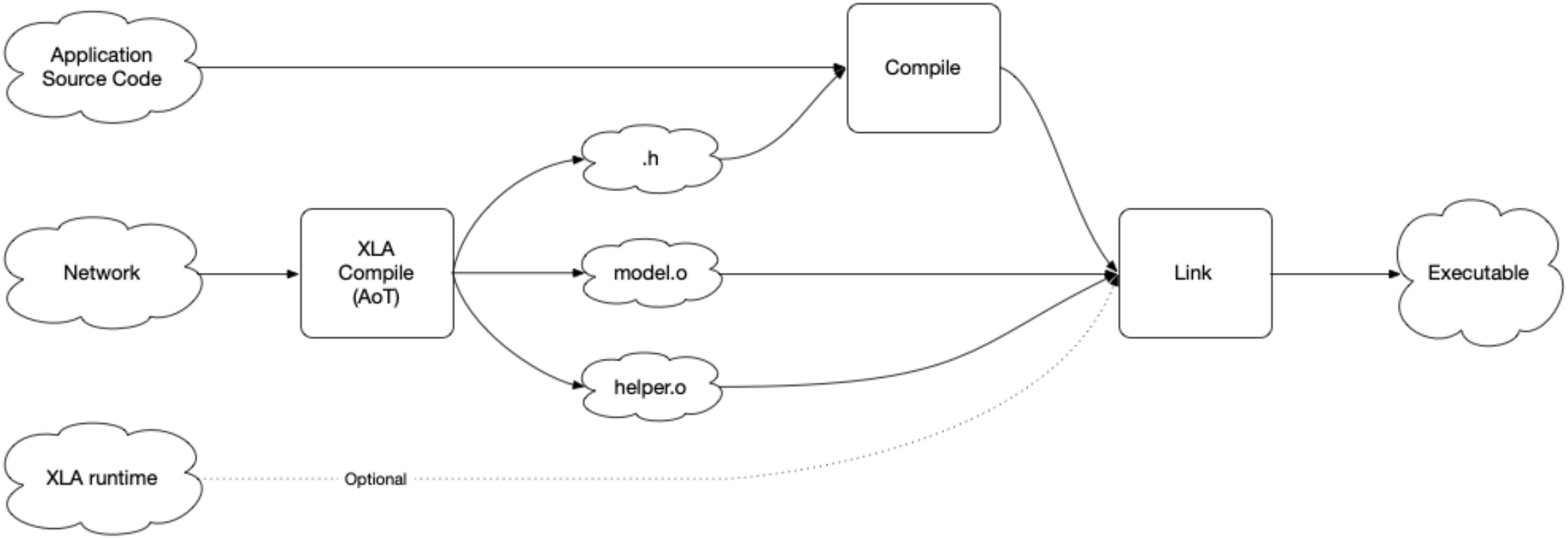
Structure of XLA: Data Path



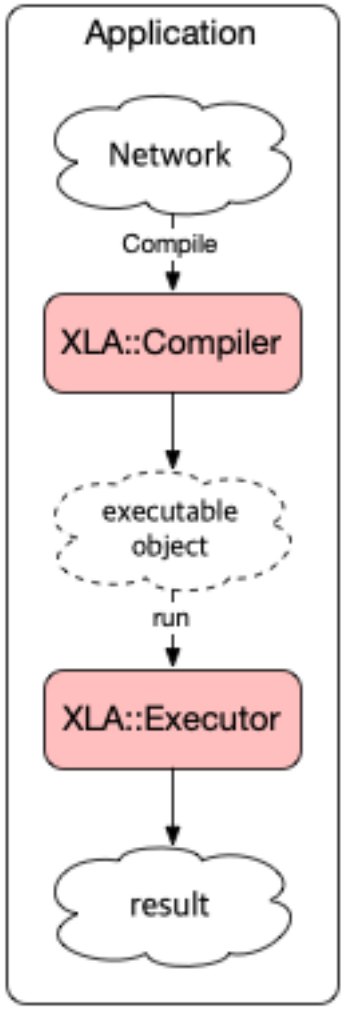
Structure of XLA: Data Structure

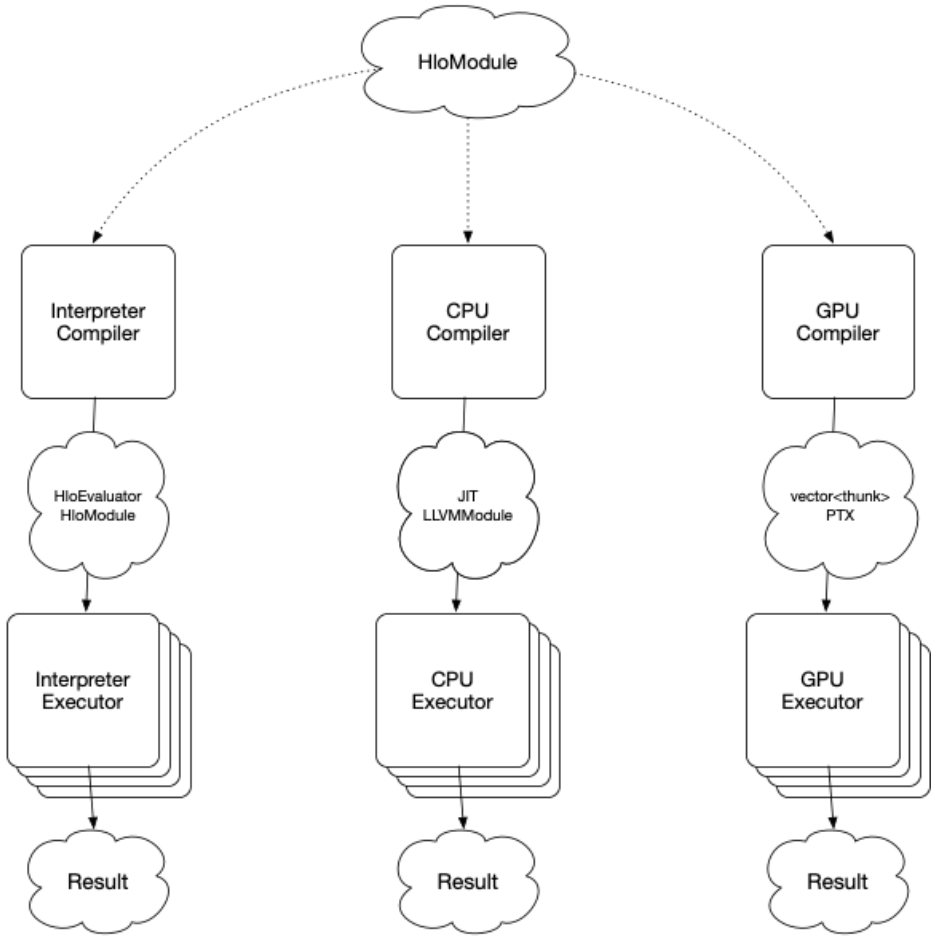


AoT Compilation w/ XLA



JiT Compilation w/ XLA





XLA in Practice: Building AoT Compiler

```
bazel build //tensorflow/compiler/aot:tfcompile
```

XLA in Practice: Writing Custom AoT Application

- Steps
  - Write Build Script (optional.)
  - Prepare Graph Config / Definition
  - Compile Graph
  - Write Custom Application
  - Compile & Link Custom Application
  - Run Custom Applcation

XLA in Practice: Writing Custom AoT Application

- Build Script (optional.)

```
load("//tensorflow/compiler/aot:tfcompile.bzl", "tf_library")

# Use the tf_library macro to compile your graph into executable code.
tf_library(
    # name is used to generate the following underlying build rules:
    # <name>           : cc_library packaging the generated header and object files
    # <name>_test      : cc_test containing a simple test and benchmark
    # <name>_benchmark : cc_binary containing a stand-alone benchmark with minimal deps;
    #                    can be run on a mobile device
    name = "test_graph_tfadd",
    # cpp_class specifies the name of the generated C++ class, with namespaces allowed.
    # The class will be generated in the given namespace(s), or if no namespaces are
    # given, within the global namespace.
    cpp_class = "foo::bar::tfadd",
    # graph is the input GraphDef proto, by default expected in binary format.  To
    # use the text format instead, just use the '.pbtxt' suffix.  A subgraph will be
    # created from this input graph, with feeds as inputs and fetches as outputs.
    # No Placeholder or Variable ops may exist in this subgraph.
    graph = "test_graph_tfadd.pbtxt",
    # config is the input Config proto, by default expected in binary format.  To
    # use the text format instead, use the '.pbtxt' suffix.  This is where the
    # feeds and fetches were specified above, in the previous step.
    config = "test_graph_tfadd.config.pbtxt",
)
```

XLA in Practice: Writing Custom AoT Application

- Graph Config Example

```
# Text form of tensorflow.tf2xla.Config proto.
feed {
  id { node_name: "x_const" }
  shape {
    dim { size: 1 }
  }
}
feed {
  id { node_name: "y_reshape" }
  shape {
    dim { size: 1 }
  }
}
fetch {
  id { node_name: "x_y_sum" }
}
```

XLA in Practice: Writing Custom AoT Application

- Graph Definition Example

```
node {
  name  : "y_reshape"
  op    : "Reshape"
  input : "y_const"
  input : "y_shape"
  attr { key: "T" value { type: DT_INT32 } }
  # Attribute TShape not specified; needs to be set to its default
  # by tfcompile.
}
node {
  name  : "x_y_sum"
  op    : "Add"
  input : "x_const"
  input : "y_reshape"
  attr {
    key  : "T"
    value {
      type: DT_INT32
    }
  }
}
versions {
  producer: 15
}
```

XLA in Practice: Writing Custom AoT Application

- Compile Graph with bazel

```
bazel build //tensorflow/compiler/aot/test_project:test_graph_tfadd
```

XLA in Practice: Writing Custom AoT Application

- Compile Graph w/o bazel

```
tfcompile --config=test_graph_tfadd.config.pbtxt \
          --graph=test_graph_tfadd.pbtxt \
          --cpp_class=foo::bar::tfadd \
          --out_header=test_graph_tfadd.h
```

XLA in Practice: Writing Custom AoT Application

- Compile Graph w/o bazel

option	description
--graph	Input GraphDef file
--config	Input file containing Config proto
--target_triple	Target platform, similar to the clang –target flag
--target_cpu	Target cpu, similar to the clang –mcpu flag
--target_features	Target features, e.g. +avx2 +neon
--entry_point	Name of generated function
--cpp_class	Name of the generated C++ class
--out_header	Output header file name

XLA in Practice: Writing Custom AoT Application

- Write Custom Application

```
#include <iostream>
#include "tensorflow/compiler/aot/my/test_graph_tfadd.h" // generated

int main(int argc, char** argv) {
    foo::bar::tfadd add;

    // Set up args and run the computation.
    const float args[12] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
    std::copy(args + 0, args + 1, add.arg0_data());
    std::copy(args + 6, args + 7, add.arg1_data());
    add.Run();

    // Check result
    if (add.result0(0) == 8) {
        std::cout << "Success" << std::endl;
    } else {
        std::cout << "Failed. Expected value 8. Got:"
        << add.result0() << std::endl;
    }

    return 0;
}
```

XLA in Practice: Writing Custom AoT Application

- Compile & Link Custom Application

```
load("//tensorflow/compiler/aot:tfcompile.bzl", "tf_library")

# Use the tf_library macro to compile your graph into executable code.
tf_library(
    name = "test_graph_tfadd",
    cpp_class = "tfadd",
    graph = "test_graph_tfadd.pbtxt",
    config = "test_graph_tfadd.config.pbtxt",
    tfcompile_flags = ["--xla_cpu_multi_thread_eigen=false"]
)

# The executable code generated by tf_library can then be linked into your c
cc_binary(
    name = "my_binary",
    srcs = [
        "my_app.cc",  # include test_graph_tfmatmul.h to access the generate
    ],
    deps = [
        ":test_graph_tfadd",  # link in the generated object file
    ]
)
```

XLA in Practice: Writing Custom AoT Application

- Compile & Link Custom Application

```
$ bazel build //tensorflow/compiler/aot/my:my_binary
```

```
$ ls -l bazel-bin/tensorflow/compiler/aot/my/my_binary
-r-xr-xr-x 1 dev dev 26160 Aug  5 18:55 bazel-bin/tensorflow/compiler/aot/my
```

XLA in Practice: Writing Custom JiT Application

- Steps
  - Write Build Script
  - Write Custom Application
  - Compile & Link Custom Application
  - Run Custom Applcation

XLA in Practice: Writing Custom JiT Application

- Step1: Write Build Script

```
# cat tensorflow/compiler/xla/mytest/BUILD

load("//tensorflow:tensorflow.bzl", "tf_cc_test")

tf_cc_test(
    name = "hello_xla",
    srcs = ["hello_xla.cc"],
    deps = [
        "//tensorflow/compiler/xla:literal",
        "//tensorflow/compiler/xla:shape_util",
        "//tensorflow/compiler/xla:statusor",
        "//tensorflow/compiler/xla:test_helpers",
        "//tensorflow/compiler/xla:xla_data_proto",
        "//tensorflow/compiler/xla/client:global_data",
        "//tensorflow/compiler/xla/client:local_client",
        "//tensorflow/compiler/xla/client:xla_builder",
        "//tensorflow/compiler/xla/client:xla_computation",
        "//tensorflow/compiler/xla/client/lib:arithmetic",
        "//tensorflow/compiler/xla/service:cpu_plugin",
        "//tensorflow/compiler/xla/tests:client_library_test_base",
        "//tensorflow/compiler/xla/tests:literal_test_util",
        "//tensorflow/core:lib",
        "//tensorflow/core:test",
    ],
)
```

XLA in Practice: Writing Custom JiT Application

- Step2: Write Custom Application

```
#include "tensorflow/compiler/xla/client/client_library.h"
#include "tensorflow/compiler/xla/client/xla_builder.h"
#include "tensorflow/compiler/xla/client/xla_computation.h"

#include <iostream>

int main(int, char**) {
    // build computation graph: "float + float"
    xla::XlaComputation computation;
    {
        xla::XlaBuilder builder("computation");

        auto l = xla::ConstantR0<float>(&builder, 3.0f);
        auto r = xla::ConstantR0<float>(&builder, 2.0f);

        auto add = xla::Add(l, r);

        computation = builder.Build().ConsumeValueOrDie();
    }

    // compile & run
    auto client = xla::ClientLibrary::GetOrCreateLocalClient().ValueOrDie();
    auto result = client->Execute(computation, {/* no argument */}).ConsumeValueOrDie();

    // print result
    auto result_literal = client->Transfer(*result).ConsumeValueOrDie();
    std::cout << result_literal << std::endl;

    return 0;
}
```

See also: https://github.com/tensorflow/tensorflow/blob/master/tensorflow/compiler/xla/client/client_library.cc#L82#L88

XLA in Practice: Writing Custom JiT Application

- Step3: Compile & Link Custom Application

```
bazel build //tensorflow/compiler/xla/mytest:hello_xla
```

XLA in Practice: Writing Custom JiT Application

- Step4: Run Custom Applcation

```
./bazel-bin/tensorflow/compiler/xla/mytest/hello_xla
2019-07-15 05:38:41.393933: I tensorflow/core/platform/profile_utils/cpu_uti
2019-07-15 05:38:41.394579: I tensorflow/compiler/xla/service/service.cc:149
2019-07-15 05:38:41.394603: I tensorflow/compiler/xla/service/service.cc:157
5
```

Thanks for attention