Generalized and Transparent AI Optimization Solutions with AI Compilers from Cloud Service Providers

Feb. 2022

Kai Zhu
tashuang.zk@Alibaba-inc.com

Agenda

- Background
  - Challenges as Cloud Service Providers
  - Motivations of a Dynamic Shape Compiler
- BladeDISC Features & Overview
- System Design
  - Decoupled Architecture
  - Dynamic Shape Support
  - Shape Constraints
  - Fusion Stitching Codegen
  - Runtime Abstraction Layer
  - Multiple Frontend Support
- Numbers
- Roadmap

Alibaba Group 阿里巴巴集团

Background

- Challenges in Large Scale Deployment as Cloud Service Providers
  - Diversified Workloads
  - Good Performance with Less Human Effort
  - Adaptation to Different Hardware
  - Ease of Use
    - Users with different background
    - Less complexity in deployment
    - Efficiency in optimization
  - Robustness
  - Multiple Frontends
    - Standard/Customized TF/PyTorch in different versions
  - Different Deploy Environments
    - Inference & single/multiple nodes training

A DL compiler, which:

1, Fully support dynamic shape semantics

2, Completely transparent to users

3, Support multiple frontend and backend

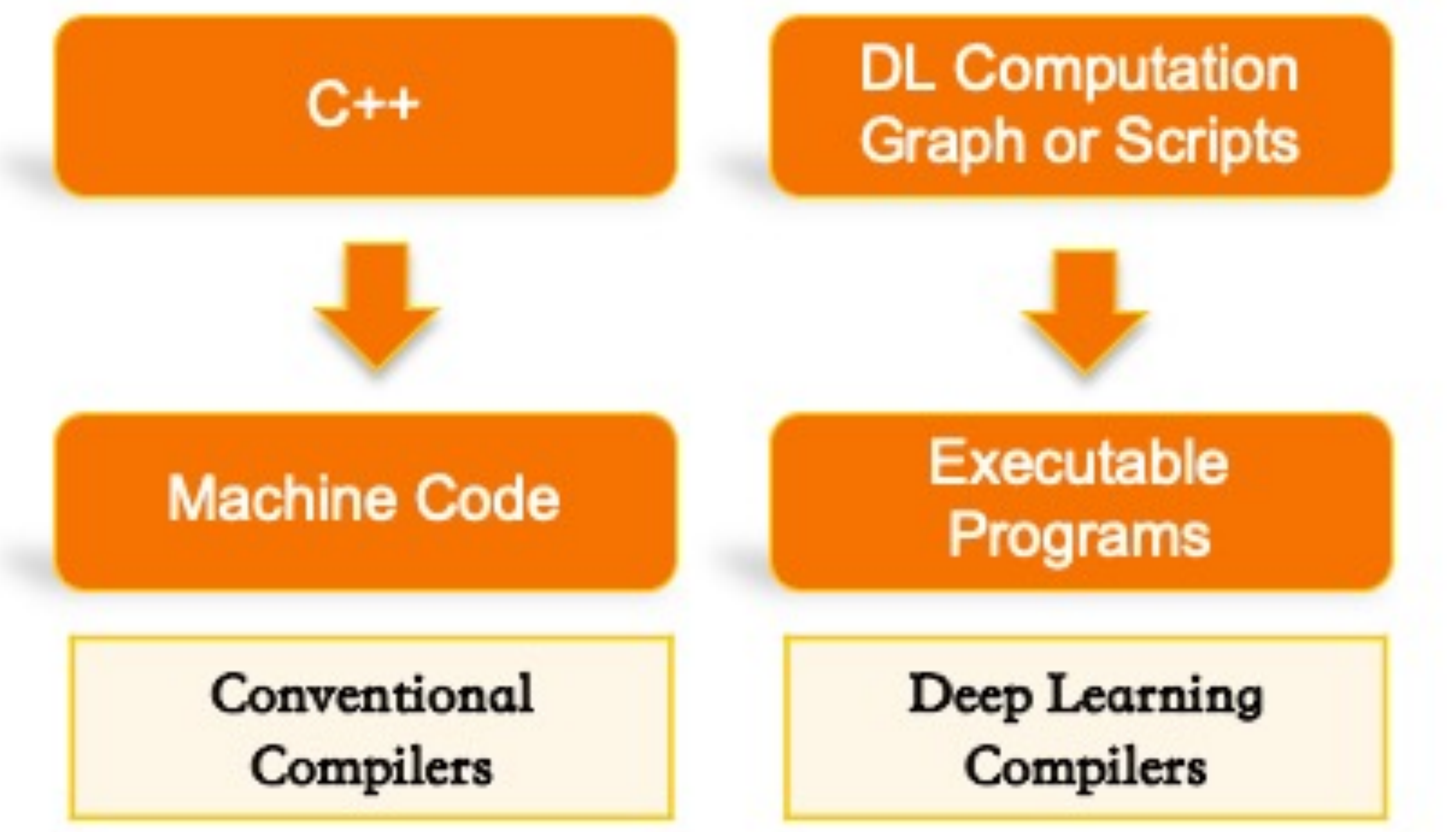
4, Decoupled, compiler as a plugin

5, Compile in a sandbox

Background

- AI Optimization Solutions
  - Manually crafted libraries and pattern matching based graph optimizers
    - TensorRT (ver < 8.0), MIGraph, OpenVINO, MNN
  - Deep Learning Compilers
    - Fill the gap between flexibility and performance
    - XLA, TVM, MLIR, IREE

- Deep Learning Compilers are Promising in
  - Good generality and scalability for a wide variety of fast evolving models
  - Easily adaptive to different backend devices
  - Common solution to fast-evolving frontend deep learning frameworks



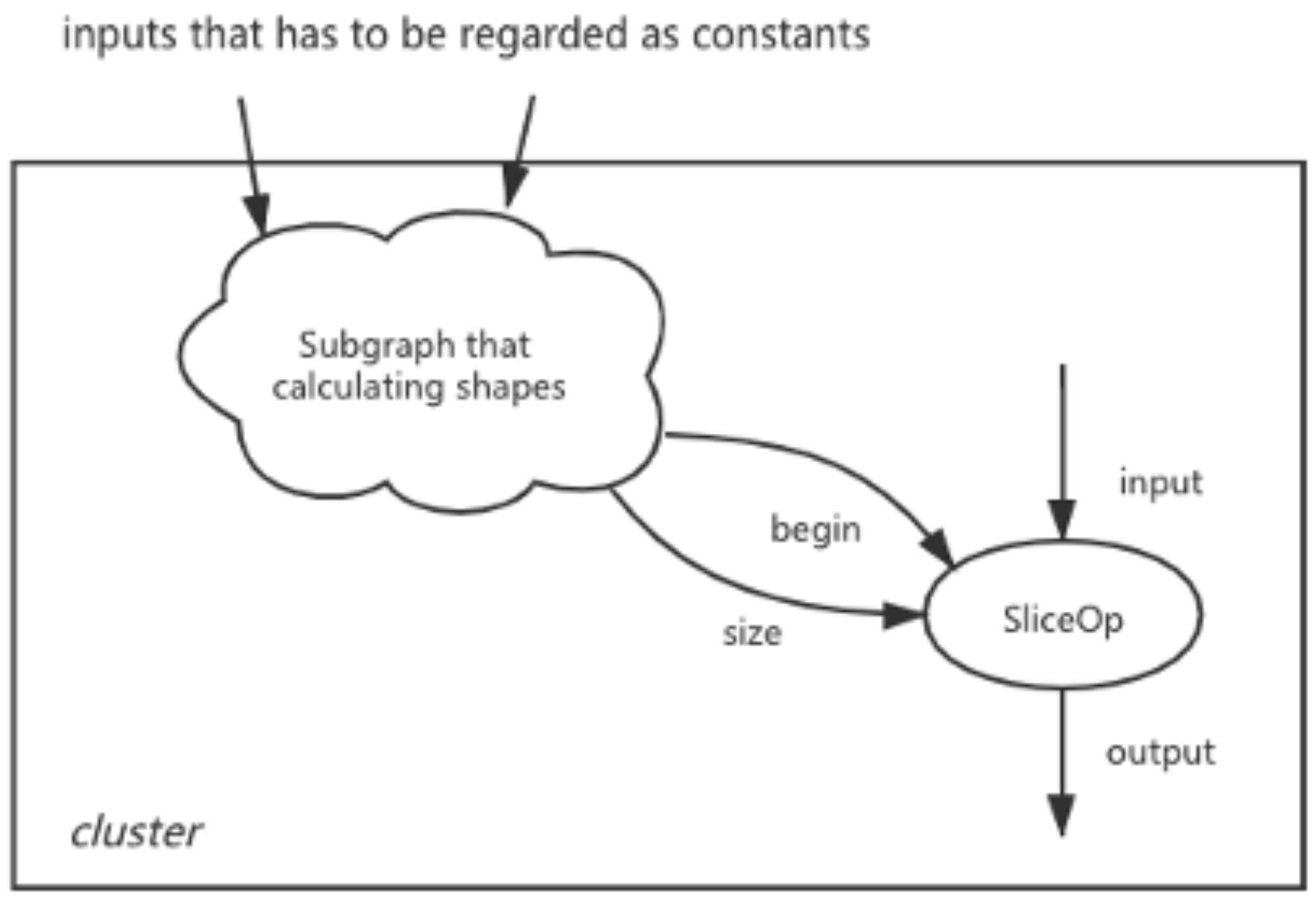
Motivations of a Dynamic Shape Compiler

- State of the art compilers are static shape oriented
  - Shapes are statically known at compile time
  - Static shape information benefits for:
    - Performance: graph level optimization, fusion decision, code generation, scheduling …
    - Memory optimization
  - However…

- A major problem that blocks the deployment and application
  - Compilation overhead
  - Problems on host / device memory usage
  - Complexity in model deployment
  - For some workloads, the amount of shapes is unlimited

Motivations of a Dynamic Shape Compiler

- Examples of workloads that suffer from static shape issues
  - CV workloads processing different image sizes, eg, object detection
  - Seq2seq models with varying input seq_len, output seq_len and batch size
  - TTS models with random shapes in the decoder even for fixed inputs
  - Sparse workloads with Unique ops generating varying shapes
    - tf.feature_column
    - Large scale embedding in distributed training



Alibaba Group 阿里巴巴集团

Features & Overview

- BladeDISC (Blade DynamIc Shape Compiler)
  - Multiple frontend support
    - TensorFlow & PyTorch
  - Multiple backend device support
    - GPGPU (CUDA & ROCM)
    - x86
  - Inference & training support
  - Fully dynamic shape semantics support
    - No restrictions on dynamic shape support
    - Without awareness of the semantics of dynamic dimensions (batchsize, sequence length etc.)
  - Deployment solutions
    - Plugin Mode: as a plugin of TensorFlow/PyTorch, with unsupported ops executed by TensorFlow/PyTorch runtime.
    - Standalone Mode: Standalone runtime for AOT application.

Features & Overview

- Transparency to Users
  - Plugin Mode: Only a few lines of codes on the original scripts are needed to turn on the compiler.

For TensorFlow Users

Only two lines of code are needed on native Tensorflow program as the following:

```
import numpy as np
import tensorflow as tf

## enable BladeDISC on TensorFlow program
import tensorflow_blade_disc as disc
disc.enable()

## construct TensorFlow Graph and run it
g = tf.Graph()
with g.as_default():
    ...
    with tf.session as sess:
        sess.run(...)
```

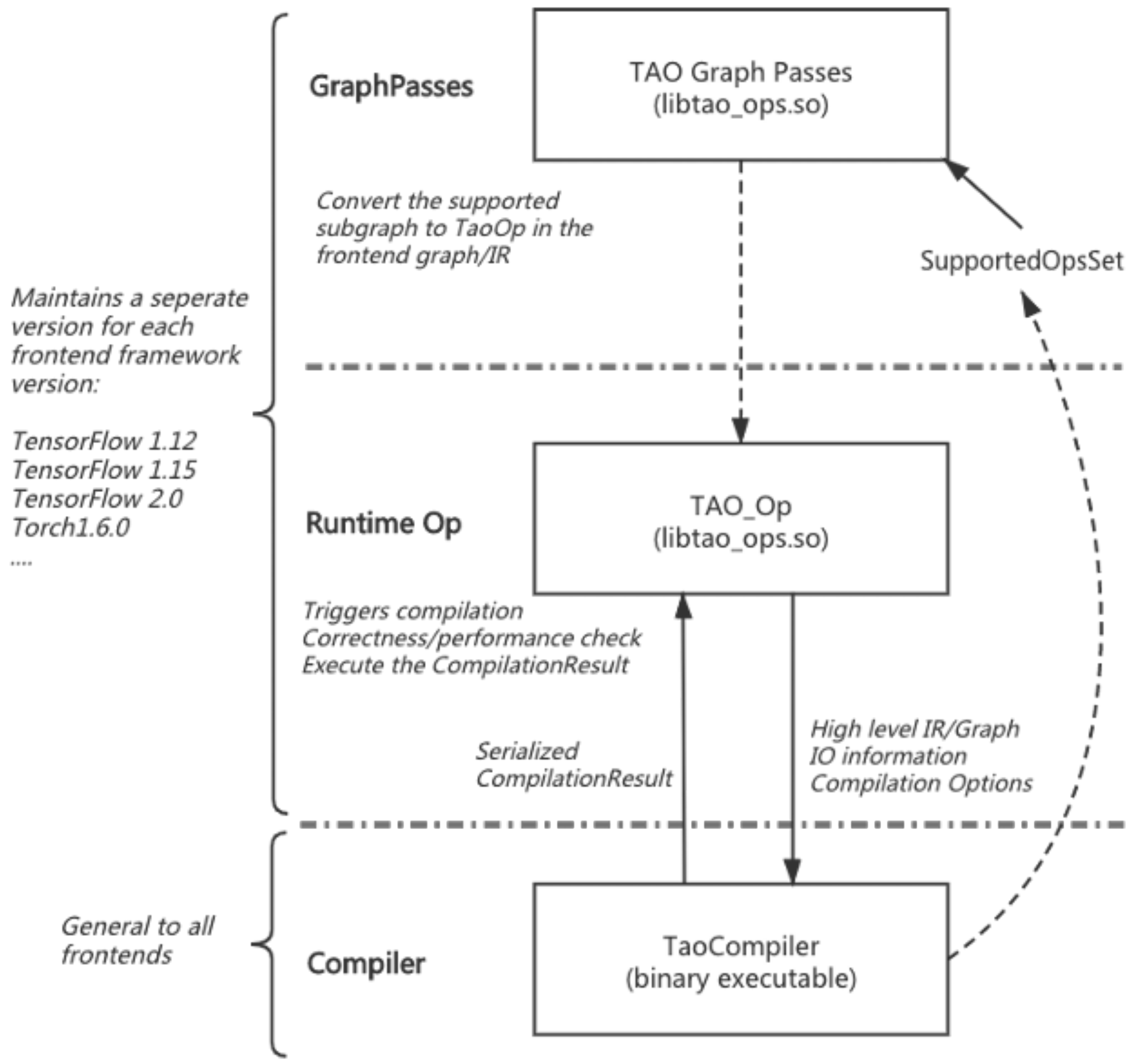
For PyTorch Users

PyTorch users only need the following few lines of code to enable BladeDISC:

```
import torch_blade
# construct PyTorch Module
class MyModule(nn.Module):
    ...

module = MyModule()

with torch.no_grad():
    # blade_module is the optimized module by BladeDISC
    blade_module = torch_blade.optimize(module, allow_tracing=True, model_inputs=(x, y))

# run the optimized module
blade_module(x, y)
```

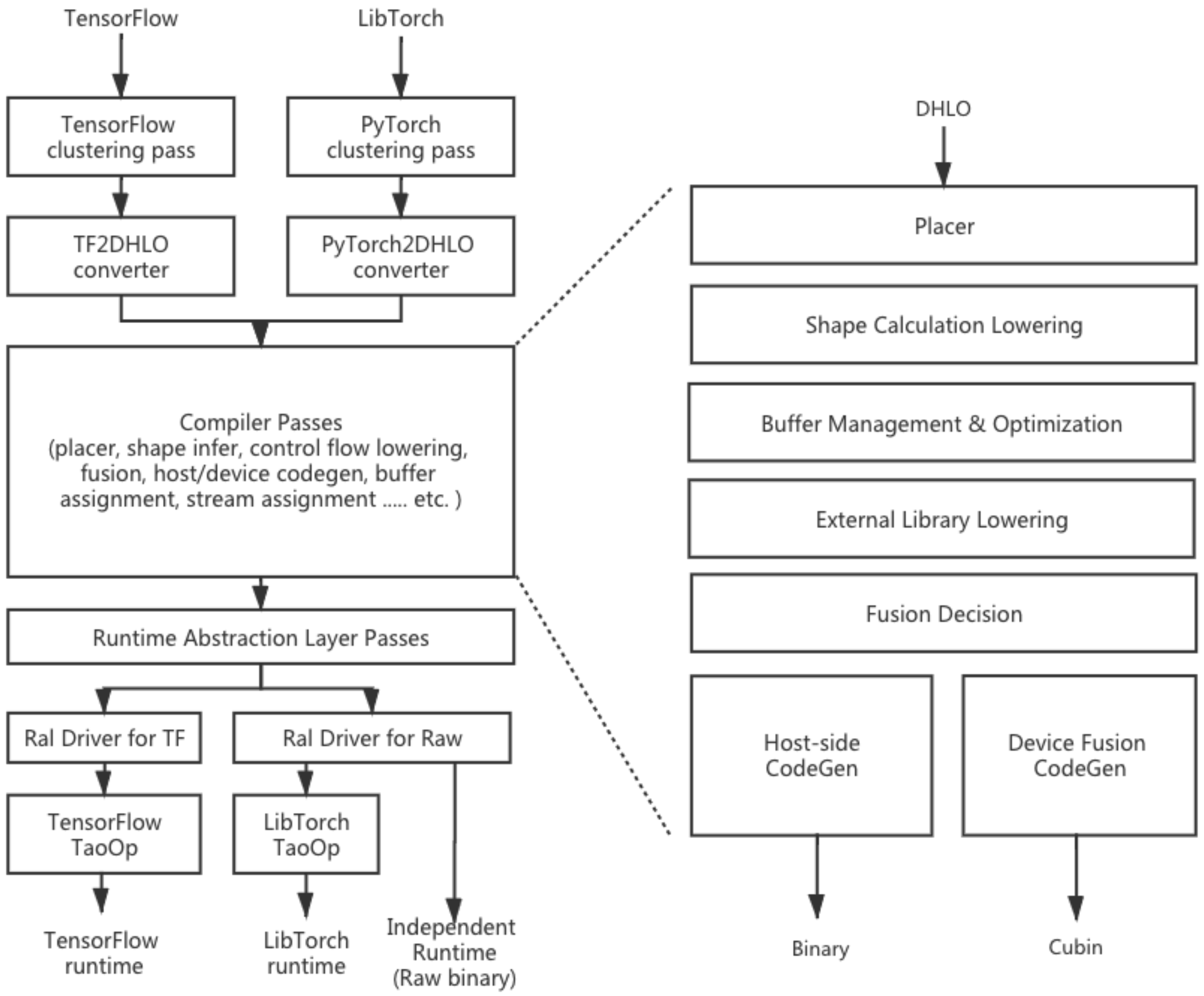
System Design

- Compiler as a plugin
  - Design Goal
    - Only maintains one copy of the compiler code
    - Adapting to different frontends easily
    - Fallback mechanism
  - Basic Ideas
    - Clustering based compilation
      - A graph rewriter pass to find candidate subgraphs
      - Following community best practice
      - Suitable for both training and inference
      - Friendly for custom op
    - Separating compilation & execution
      - A standalone compiler to do heavy lift things
      - A custom op to wrap execution logic
      - Avoid some engineering headaches (e.g. linking, compatibility)



System Design

- Compiler
  - Multiple framework support
    - MHLO as the centralized graph IR
  - Multiple backend hardware support
    - LLVM IR
  - Runtime Abstraction Layer
    - To isolate the compiler and different runtime environments
  - Kernel library integration
    - Cublas, cudnn etc.
    - A balance between complexity, flexibility & performance



System Design

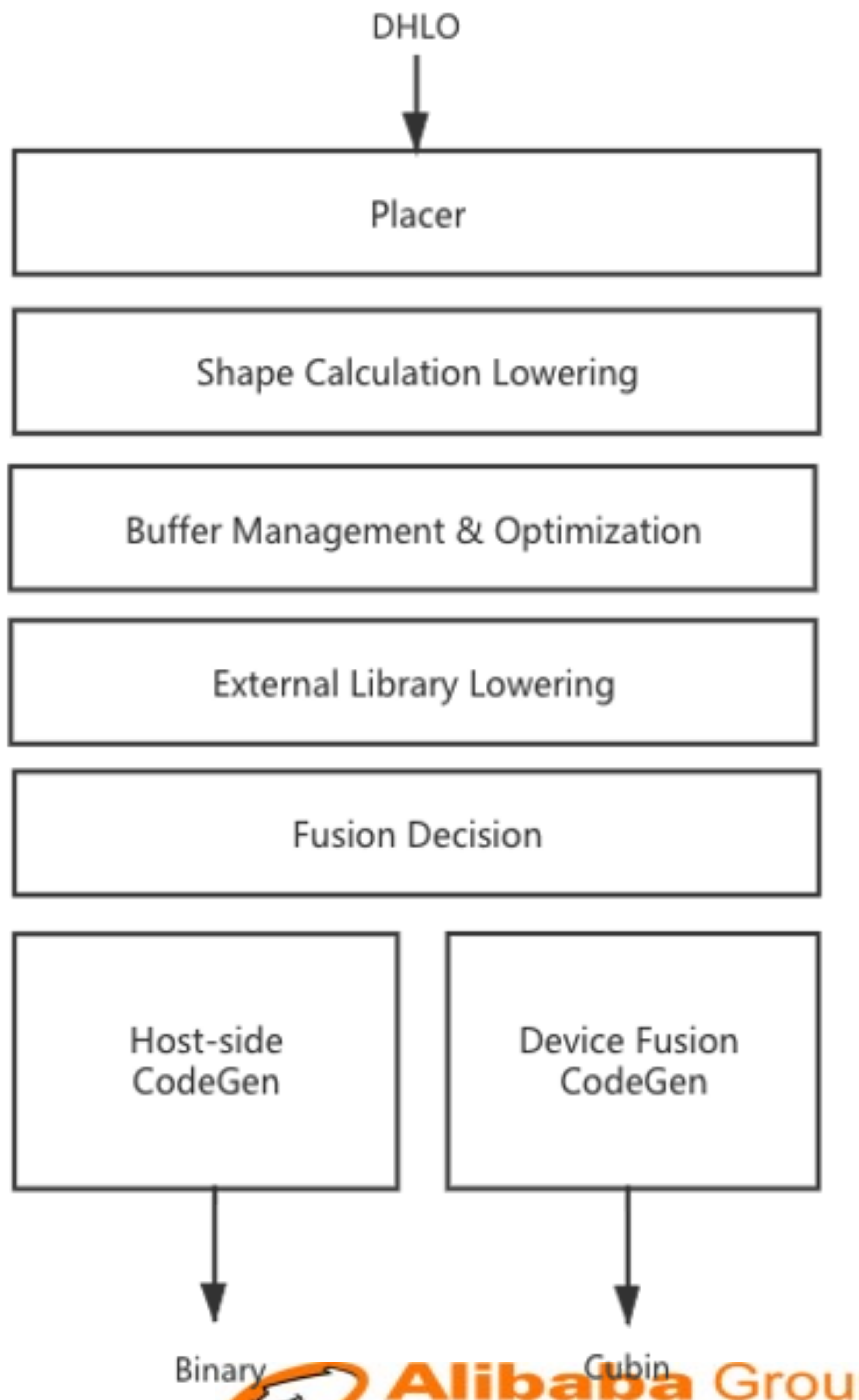
- **Fully Dynamic Shape Support**
  - The IRs that can fully represent dynamic shape semantics
    - Supplement of MHLO/LMHLO Dialect
  - Code generated runtime flow
    - Adaptive shape inference
    - Dynamic buffer management
    - Host-side control
  - Graph Optimization & CodeGen in dynamic shape
  - Fusion & code generation
    - Shape hints & constraints
    - Shape adaptive fusion configuration
  - Placer
  - Buffer Allocation & Scheduling

```
def HLO_SliceOp: HLO_Op<
      "slice",
      [NoSideEffect, SameOperandsAndResultElementType,
       AllTypesMatch<["start_indices", "limit_indices", "strides"]>,
       DeclareOpInterfaceMethods<InferTypeOpInterface>]> {
  let arguments = (ins
    HLO_Tensor:$operand,
    I64ElementsAttr:$start_indices,
    I64ElementsAttr:$limit_indices,
    I64ElementsAttr:$strides
  );
```

static shape semantics

```
def HLO_RealDynamicSliceOp: HLO_ShapedInterfaceOp<
      "real_dynamic_slice",
      [NoSideEffect, AllElementTypesMatch<["operand", "result"]>,
       AllTypesMatch<["start_indices", "limit_indices", "strides"]>]> {
  let summary = "Real Dynamic Slice operator";
  let description = [{
    The dynamic shape version of SliceOp. Extracts a sub-array from the input
    array according to start_indices, limit_indices and strides. Expect
    start_indices/limit_indices/strides to be statically shaped and matching
    the rank of the input.
  }];
  let arguments = (ins
    HLO_Tensor:$operand,
    HLO_DimensionTensor:$start_indices,
    HLO_DimensionTensor:$limit_indices,
    HLO_DimensionTensor:$strides
  );
```

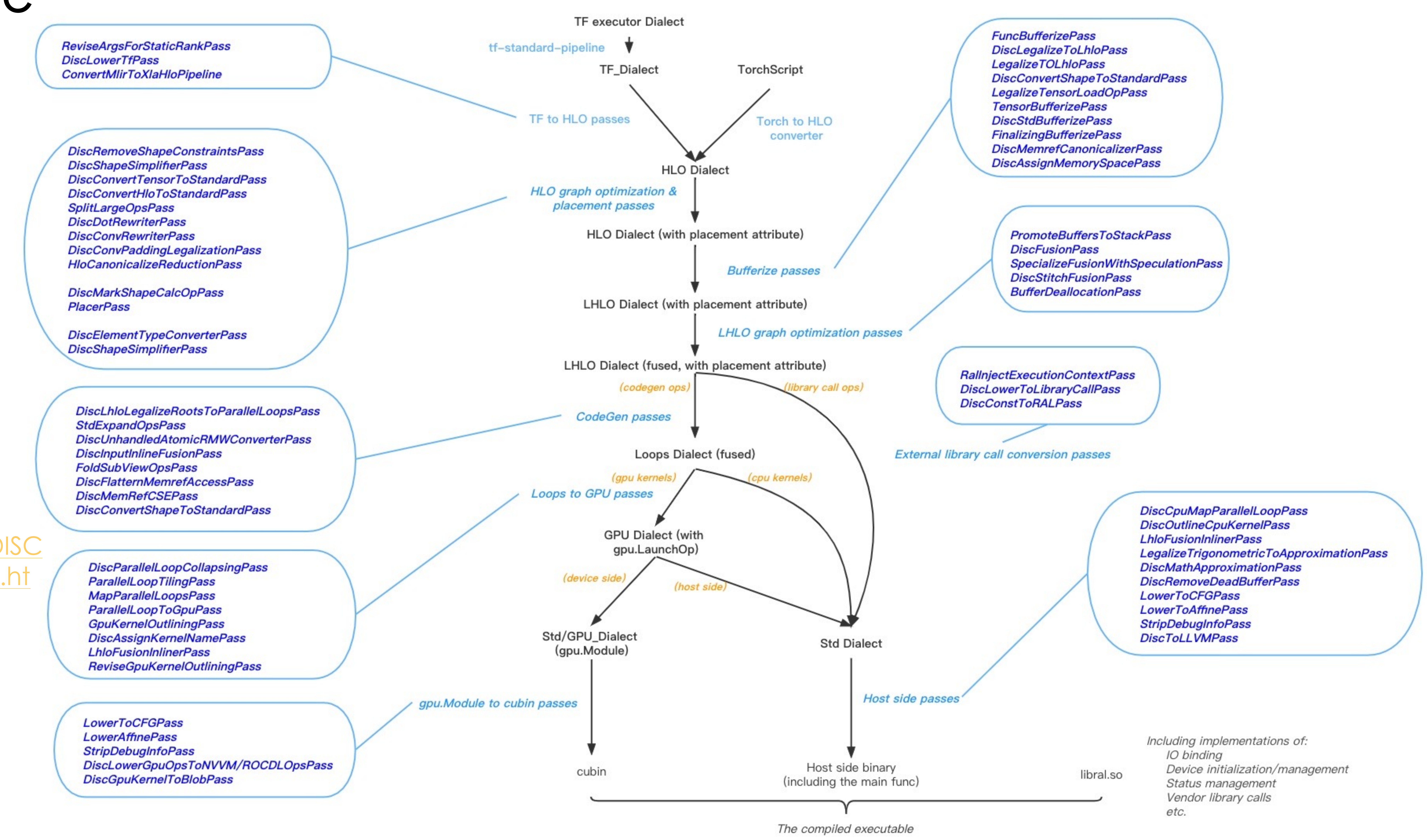
dynamic shape semantics



Alibaba Group 阿里巴巴集团

System Design

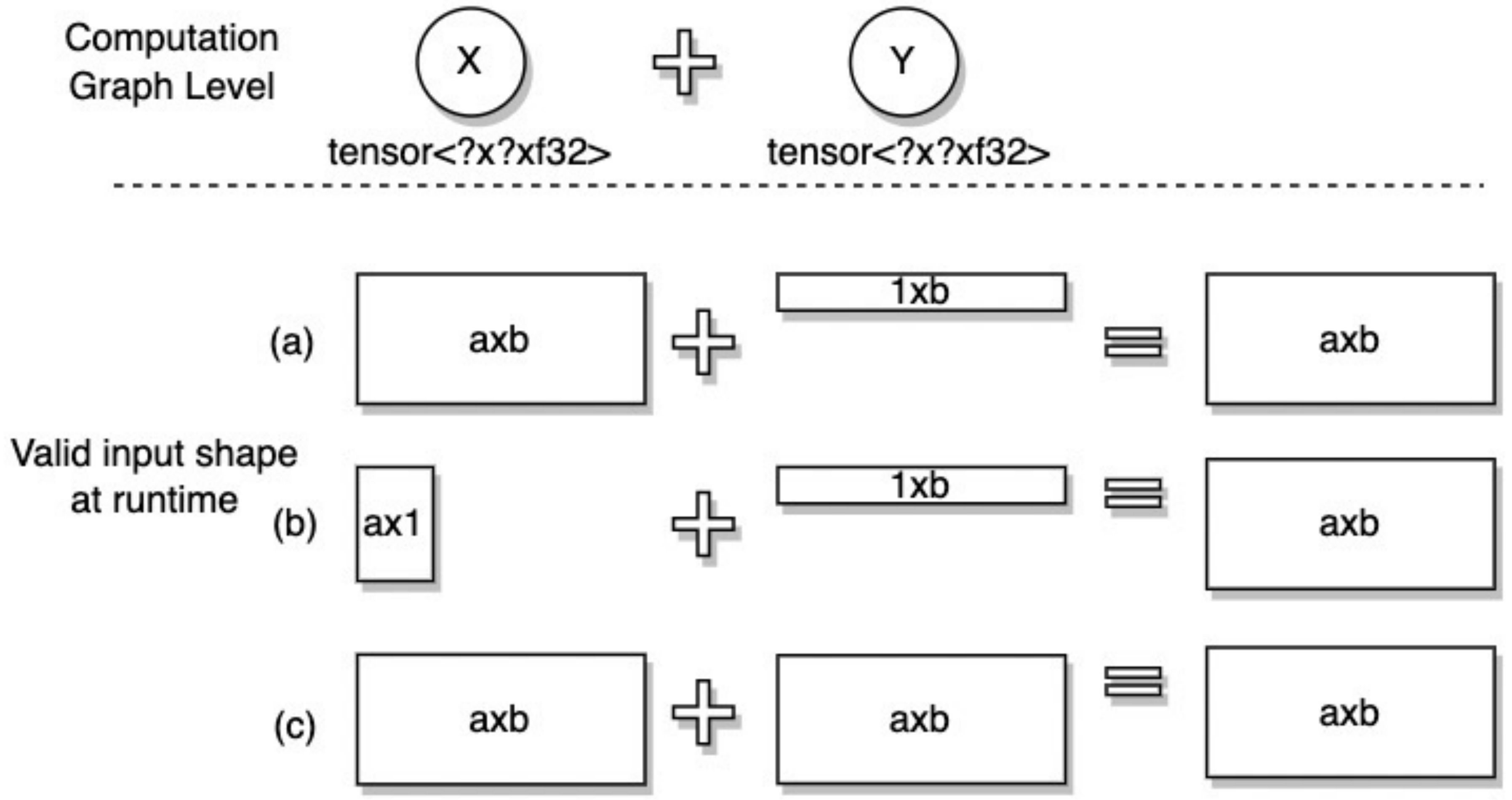
- Backbone Pass Pipeline
  - MLIR infra
    - Modular flexible infrastructure
    - Reusable & extensible
  - Major Dialects
    - DHLO Dialect
    - LDHLO Dialect
    - SCF Dialect
    - GPU Dialect
  - Tutorial of the Pass Pipeline
    - https://alibaba.github.io/BladeDISC/docs/developers/pass_pipeline.html



System Design

Challenges on Performance

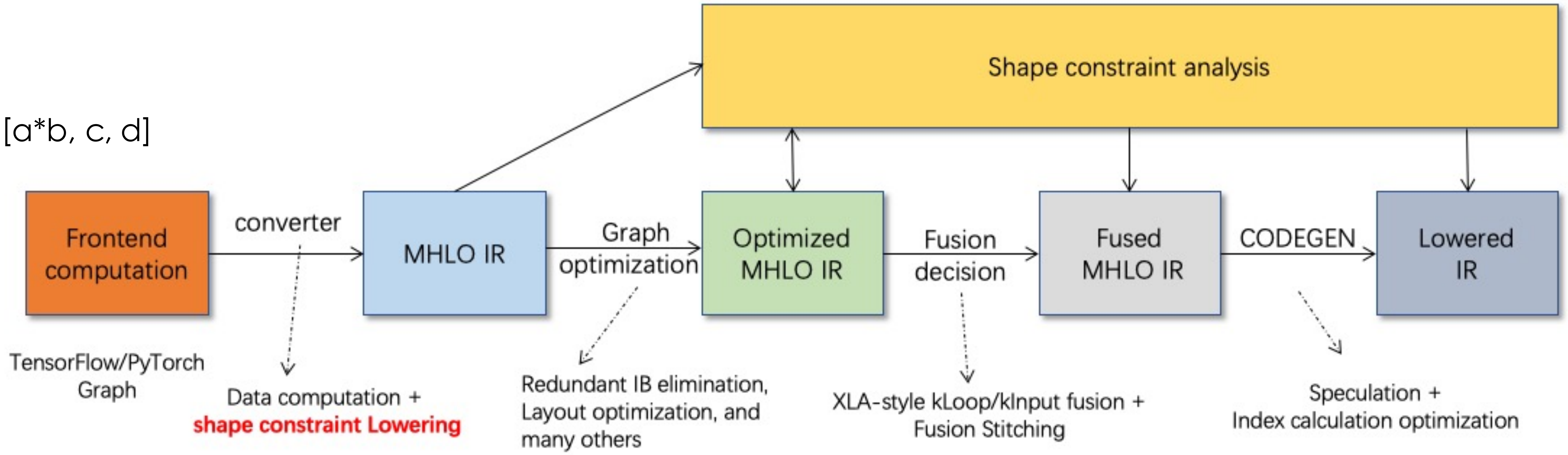
- More complicated computation graph
  - Mixed data computation & shape computation
- Optimization objective shifting
  - From peak performance to average performance, one-shape-one-solution vs transferable solution
- Less effective information/methods for optimization
  - Implicit Broadcast
  - Fusion strategy
  - Vectorization / Tiling strategies
  - Amount of index calculation instructions



An example for numpy style implicit broadcast

System Design

- Shape Constraints
  - BladeDISC optimization pipeline
    - Shape constraint centric
    - Widely used from graph level to instruction level optimizations
    - Crucial to performance in dynamic shape semantics
  - Different kinds of shape constraints
    - Structured shape constraint
      - Dimension size equality
      - Number elements equality
      - Symbolic equality: [a, b, c, d] to [a*b, c, d]
    - Shape distribution constraint
      - Dimsize %4 == 0
      - Likely values
      - Shape ranges



System Design

- Where to get shape constraints?
  - Semantics of MHLO Ops
  - Symbolic shape analysis
  - Injected by frontend converter
  - Provided by users
  - Injected at JIT compilation time

```
class HLO_UnaryElementwiseOp<string mnemonic, list<OpTrait> traits,
    Type TensorType> : HLO_Op<mnemonic, traits # [Elementwise,
    InferShapedTypeOpInterface, InferShapeEqualityOpInterface,
    SameOperandsAndResultShape]> {
  let arguments = (ins TensorType:$operand);
  let results = (outs TensorType);
```

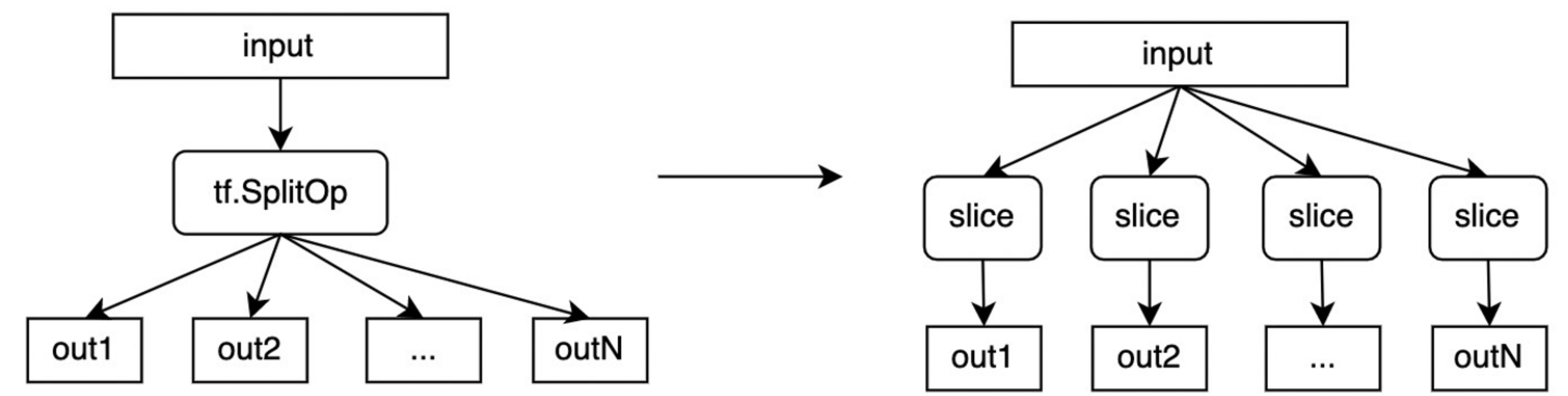
An example: infer shape constraint from the semantics of op definition

%0 = transpose %input {permutation = {1,0}}
%1 = mhlo.add %0, %input

An example for symbolic shape analysis: input shape should be squared

%0 = tensor.dim %input, %c0
%1 = tensor.dim %input, %c1
%2 = tensor.dim %input, %c2
%3 = mul %0, %1
%4 = tensor.from_elements %3, %2
%5 = mhlo.dynamic_reshape(%input, %4)

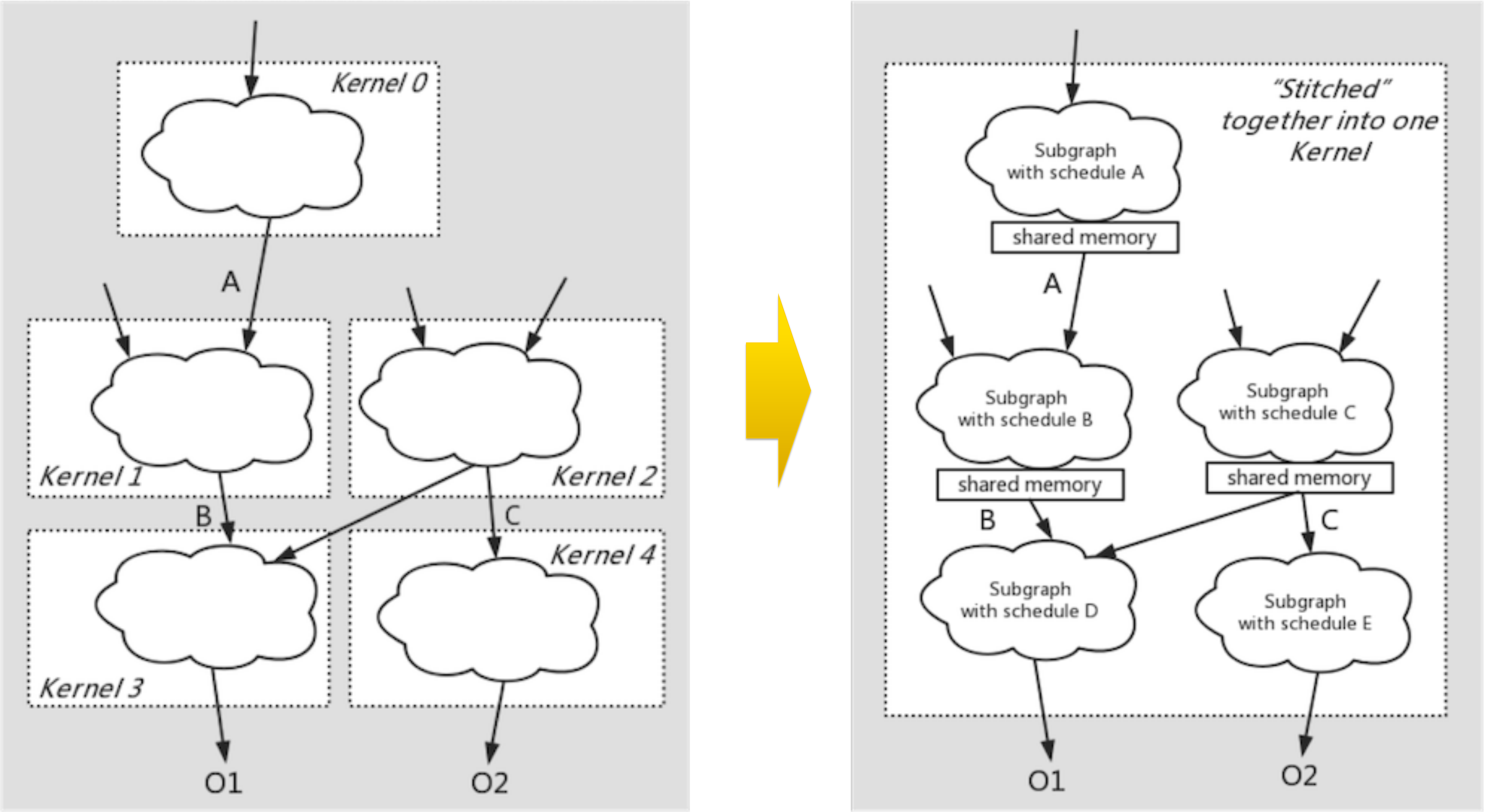
*An example for symbolic shape analysis: Reshape [a, b, c] -> [a*b, c]*



An example: shape constraints injected by frontend converter

System Design

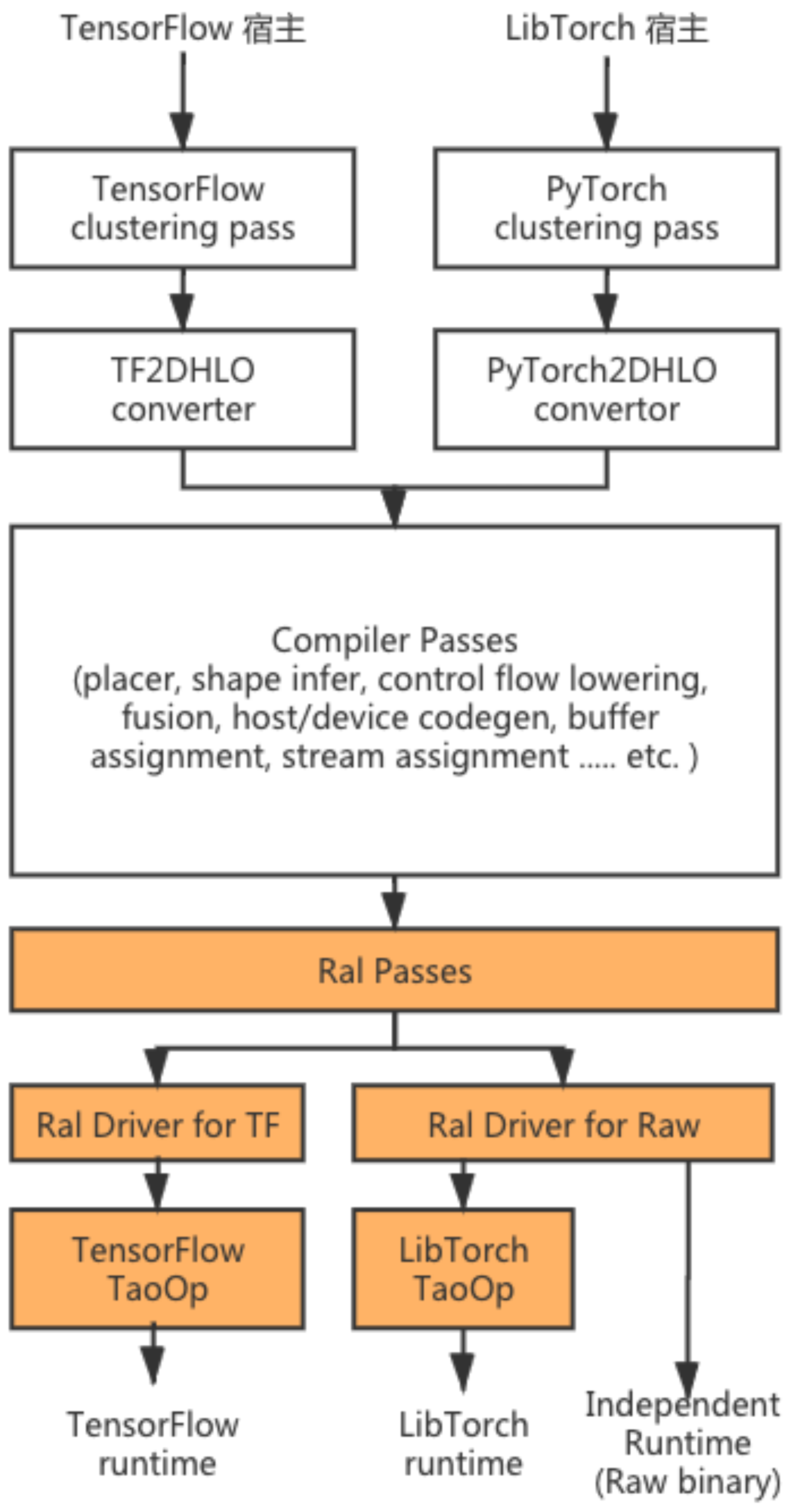
- FusionStitching CodeGen
  - Existing Works
    - Basic loop, input/output fusion
    - Less aggressive fusion, with guaranteed codegen quality
  - Major Challenges
    - More aggressive fusion granularity, while still close to the SOL of the device
    - An acceptable trade-off between compilation time and performance
  - Stitch multiple kernels into a bigger kernel
    - GPGPU - shared memory
    - CPU - local memory
  - Publications
    - https://dl.acm.org/doi/10.1145/3503222.3507723
    - https://arxiv.org/abs/2009.10924



Kernels	TensorFlow	XLA	BladeDISC
LSTM Cell	18	1 compute intensive 3 memory intensive	1 compute intensive 1 memory intensive
LayerNorm	42	6 memory intensive	1 memory intensive

System Design

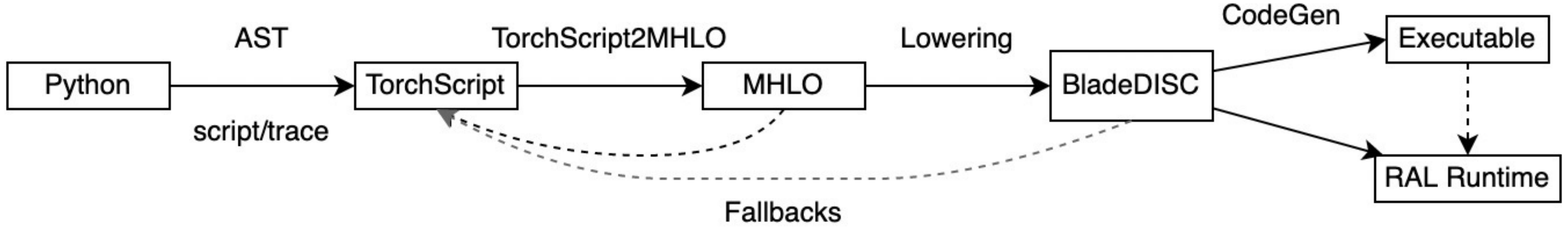
- Runtime Abstraction Layer
  - Compile Once and Run everywhere
    - As a TensorFlow Op
    - As a LibTorch Op
    - Raw independent binary
  - An abstraction to isolate compiler and runtime
    - Allocator, kernel launch, memcpy, io interface etc.
  - Stateless Compilation
    - State management are extracted to simplify the compilation
    - Constant, tuning cache etc.

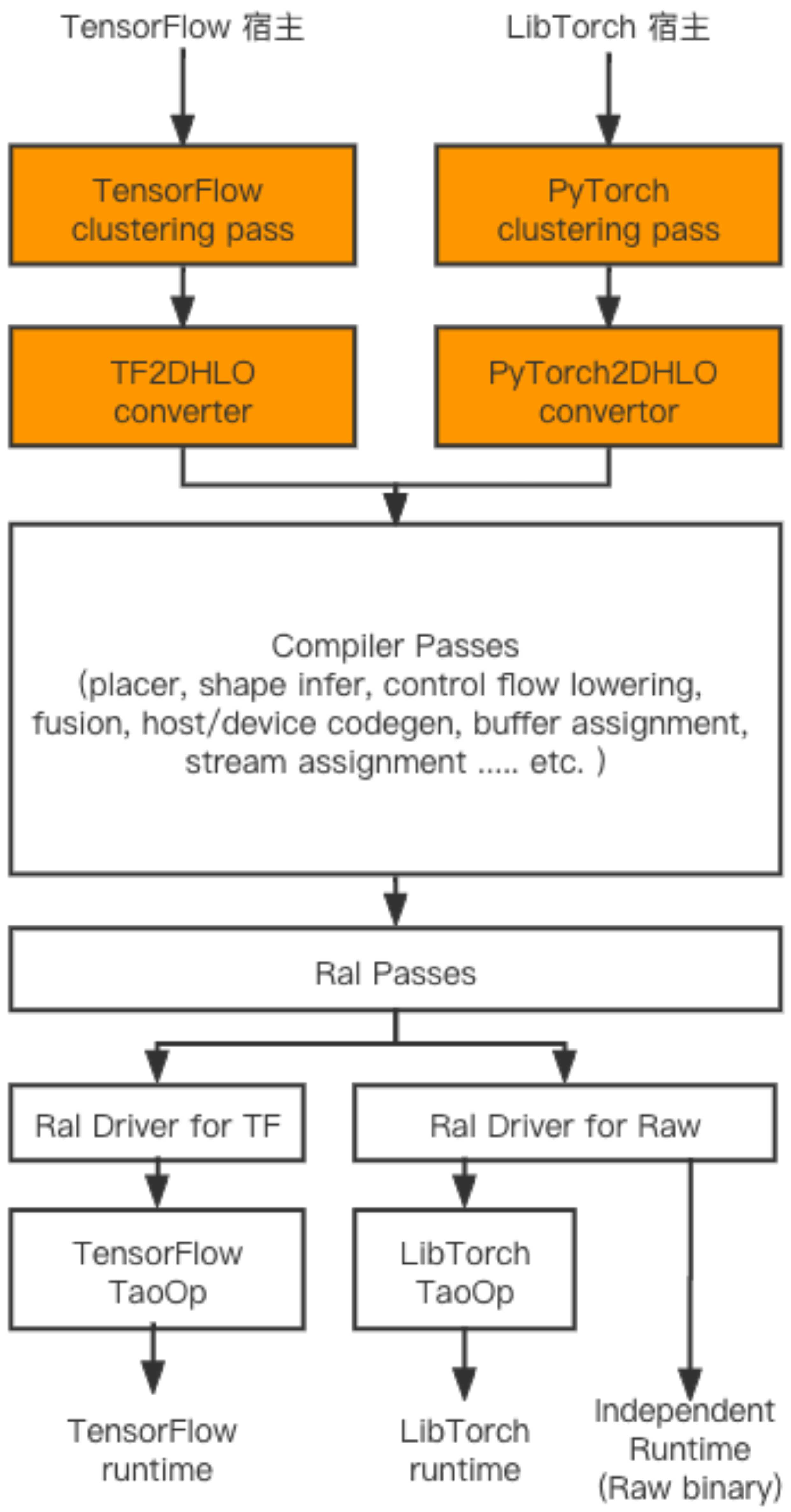


Alibaba Group 阿里巴巴集团

System Design

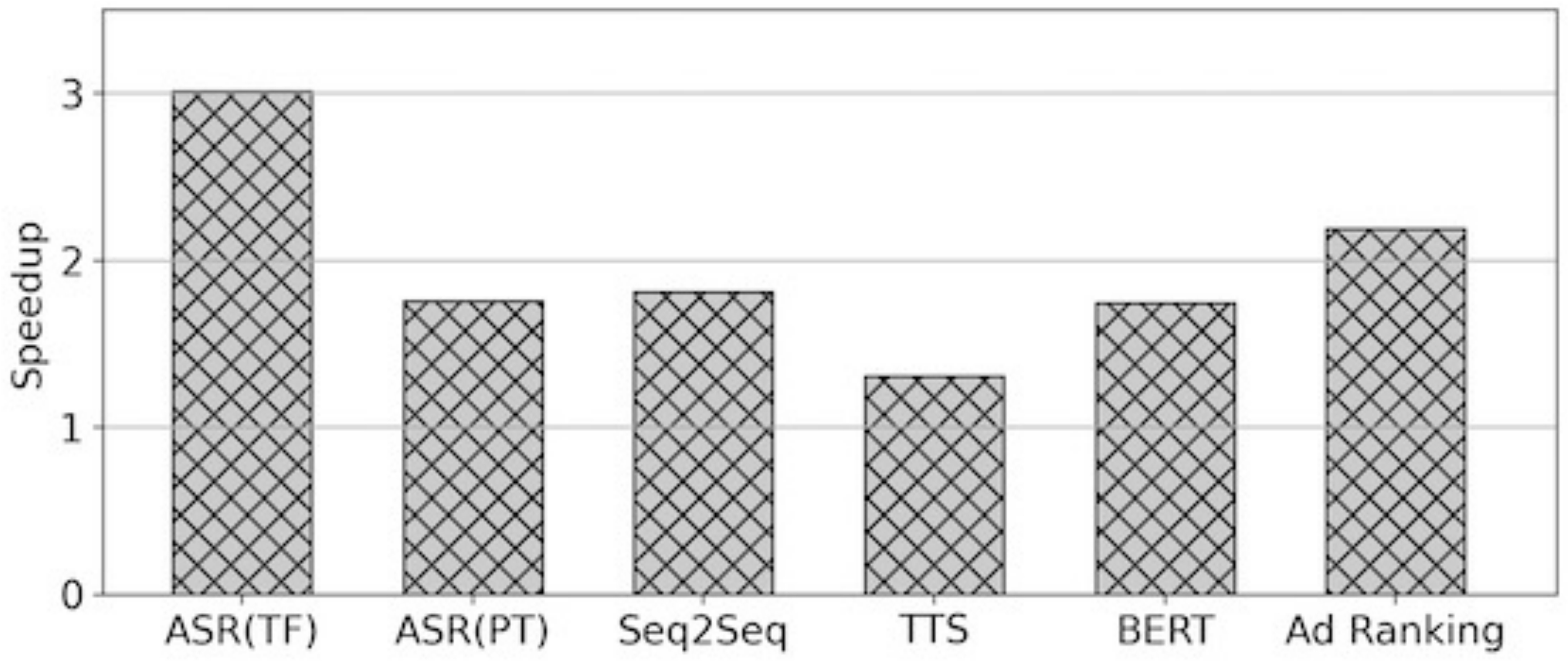
- Multiple Frontend Support
  - MHLO as a ‘Hub’ IR interfacing different frontends
  - Runtime Abstraction Layer adapts the compilation result to different runtimes





Numbers

- Up to 3x speedup compared with TensorFlow/PyTorch
- Comparing with static shape compiler (XLA)
  - In worst case, close enough (>80%) to XLA in our benchmarks
  - For some of the workloads, the performance even exceed due to large granularity fusion
- Comparing with TensorRT 8.X
  - Non-CV standard workloads (BERT etc.), typically 10% ~ 20% performance gap
  - Advantage in workload generality, dynamic shape support, and transparency of use
  - More detail numbers are under investigating and will be updated in our website

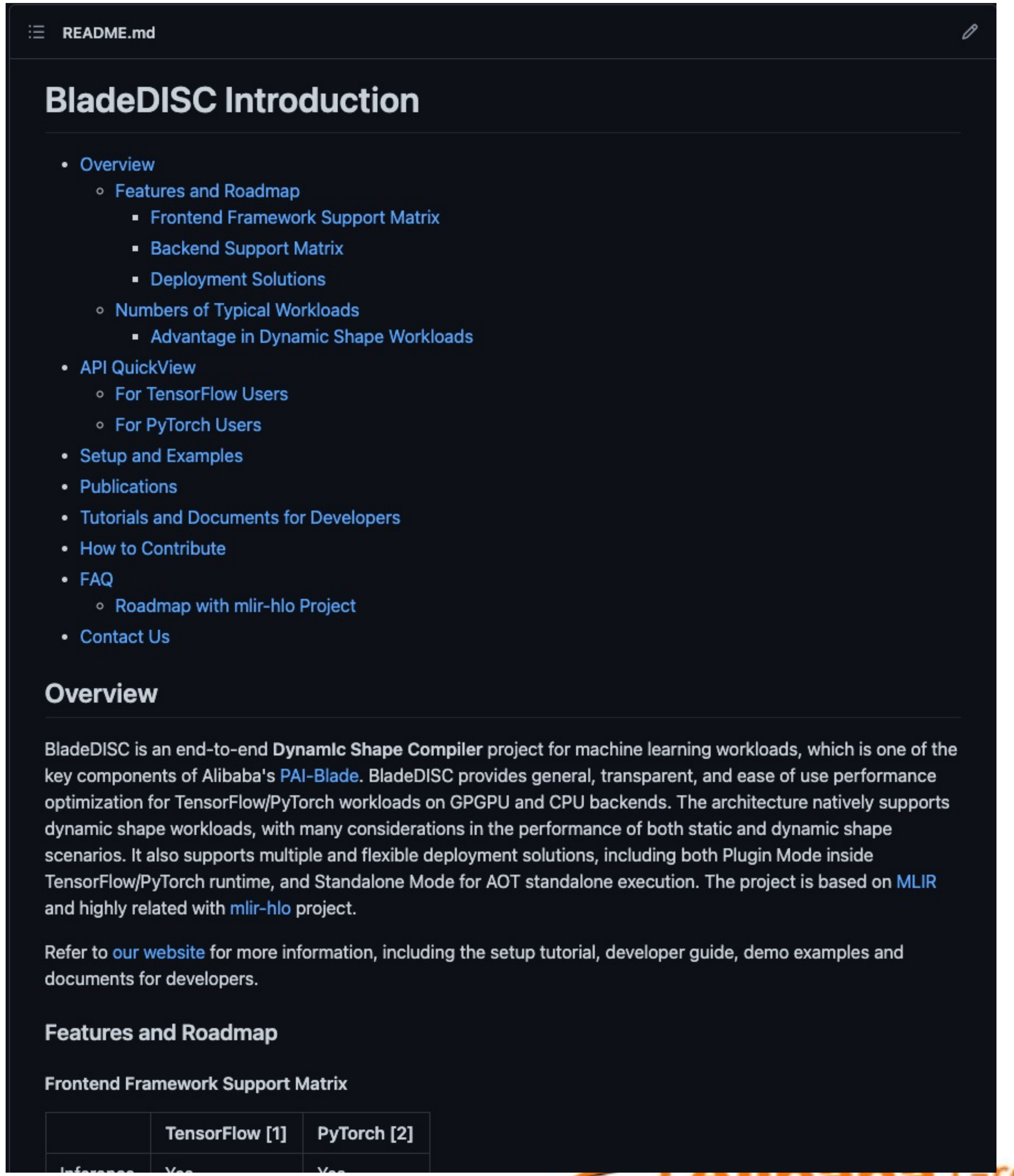


Speedup compared with TensorFlow/PyTorch

Roadmap

- Open Sourced at the End of 2021
  - Codebase
    - https://github.com/alibaba/BladeDISC
  - Documents Website
    - https://github.com/alibaba/BladeDISC
  - Welcome for a trial and technical cooperation
    - Mail group: bladedisc-dev@list.alibaba-inc.com

DingTalk group for support & discussion


README.md

BladeDISC Introduction

- Overview
  - Features and Roadmap
    - Frontend Framework Support Matrix
    - Backend Support Matrix
    - Deployment Solutions
  - Numbers of Typical Workloads
    - Advantage in Dynamic Shape Workloads
- API QuickView
  - For TensorFlow Users
  - For PyTorch Users
- Setup and Examples
- Publications
- Tutorials and Documents for Developers
- How to Contribute
- FAQ
  - Roadmap with mlir-hlo Project
- Contact Us

Overview

BladeDISC is an end-to-end **Dynamic Shape Compiler** project for machine learning workloads, which is one of the key components of Alibaba's PAI-Blade. BladeDISC provides general, transparent, and ease of use performance optimization for TensorFlow/PyTorch workloads on GPGPU and CPU backends. The architecture natively supports dynamic shape workloads, with many considerations in the performance of both static and dynamic shape scenarios. It also supports multiple and flexible deployment solutions, including both Plugin Mode inside TensorFlow/PyTorch runtime, and Standalone Mode for AOT standalone execution. The project is based on MLIR and highly related with mlir-hlo project.

Refer to our website for more information, including the setup tutorial, developer guide, demo examples and documents for developers.

Features and Roadmap

Frontend Framework Support Matrix

	TensorFlow [1]	PyTorch [2]

Roadmap

- Planned & Interested Future works
  - Continuously improvement on Op coverage, robustness, performance etc.
  - More frontend/backend support
  - PyTorch training support
  - Code generation on compute intensive part in dynamic semantics
  - Support for subgraphs with sparse features

Q & A