中国软件技术大会

ByteIR: 迈向端到端的AI编译

字节跳动 机器学习系统工程师 刘渊强

2023.12

ByteDance字节跳动

1. What is ByteIR and advantages of ByteIR

2. Design and technical details

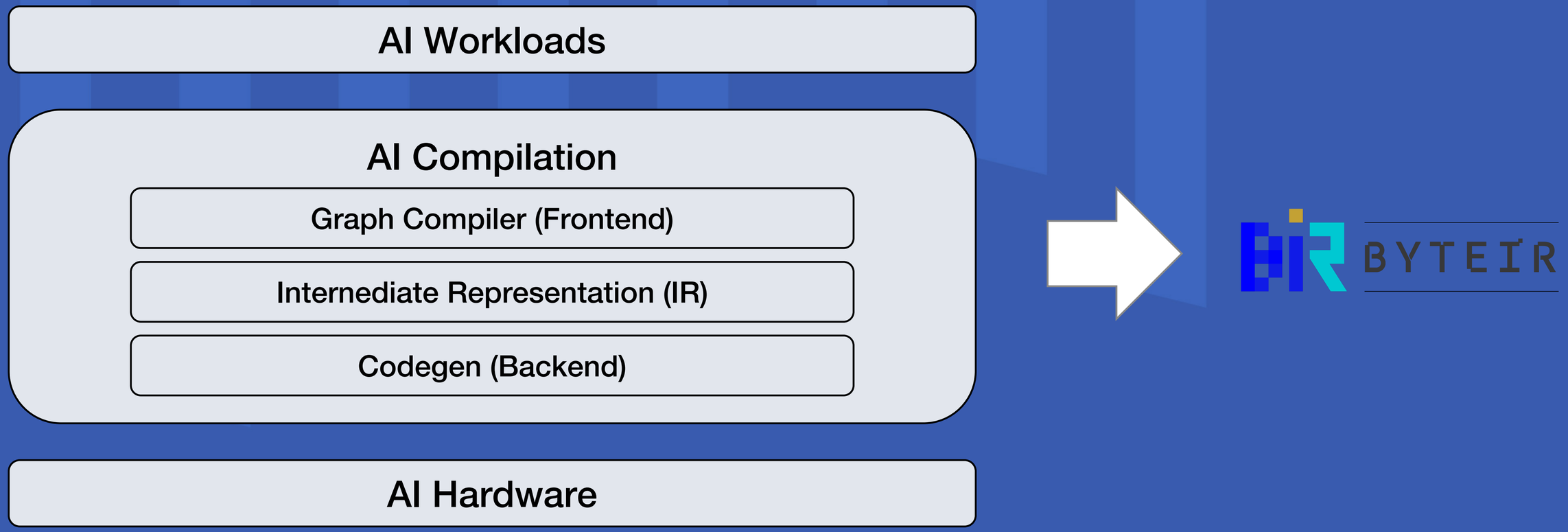
3. LLM training example and performance

What is ByteIR

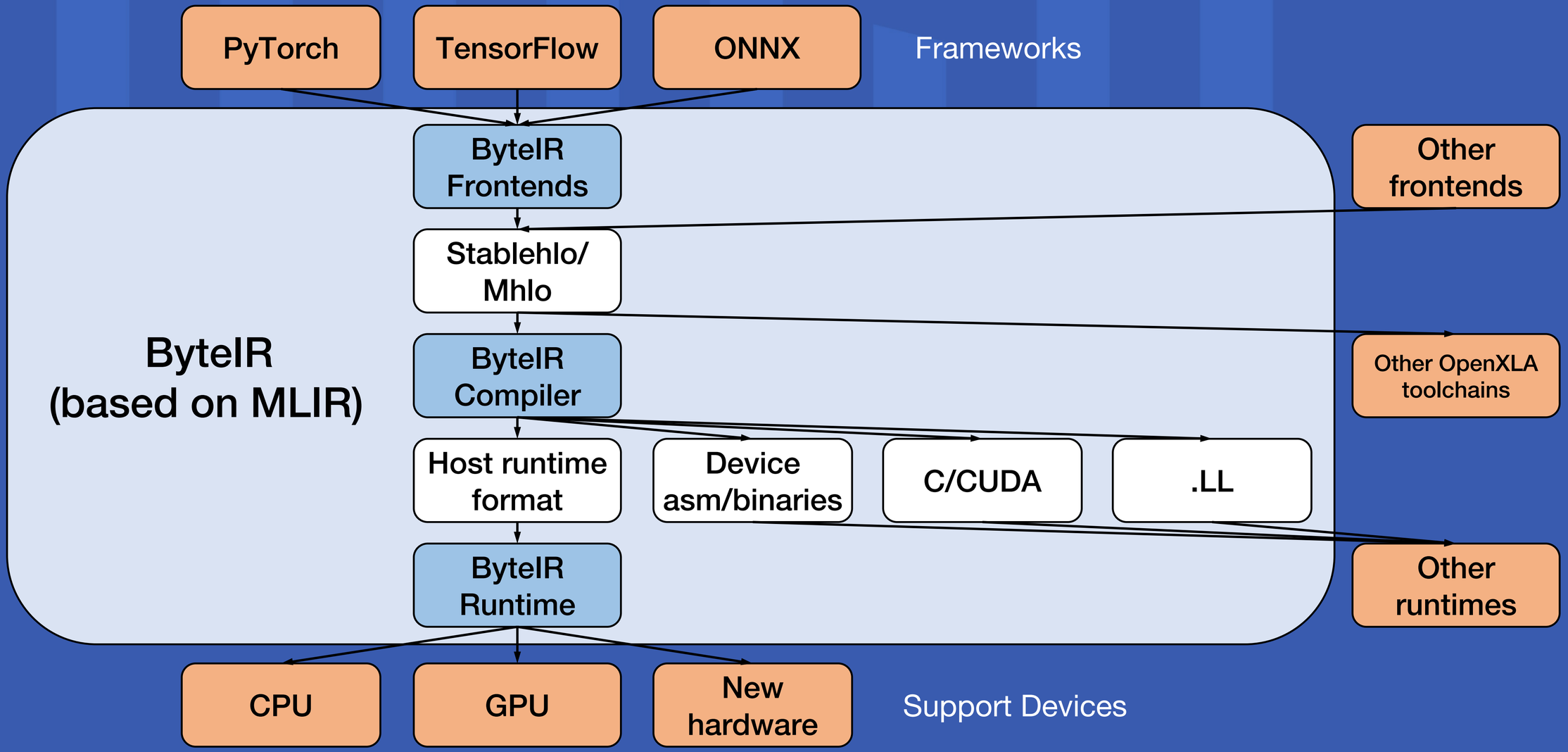
is NOT a specific IR (Intermediate Representation)

is our solution for framework-to-hardware compilation

AI Compilation: NN graph to HW



ByteIR Architecture



Advantages of ByteIR

1. Embrace open source, upstream first

   Contribute to llvm, tensorflow, pytorch, torch-mlir, onnx-mlir, stablehlo

2. Well support for PyTorch, both inference and training

3. Friendly to new hardware (ASIC/NPU)

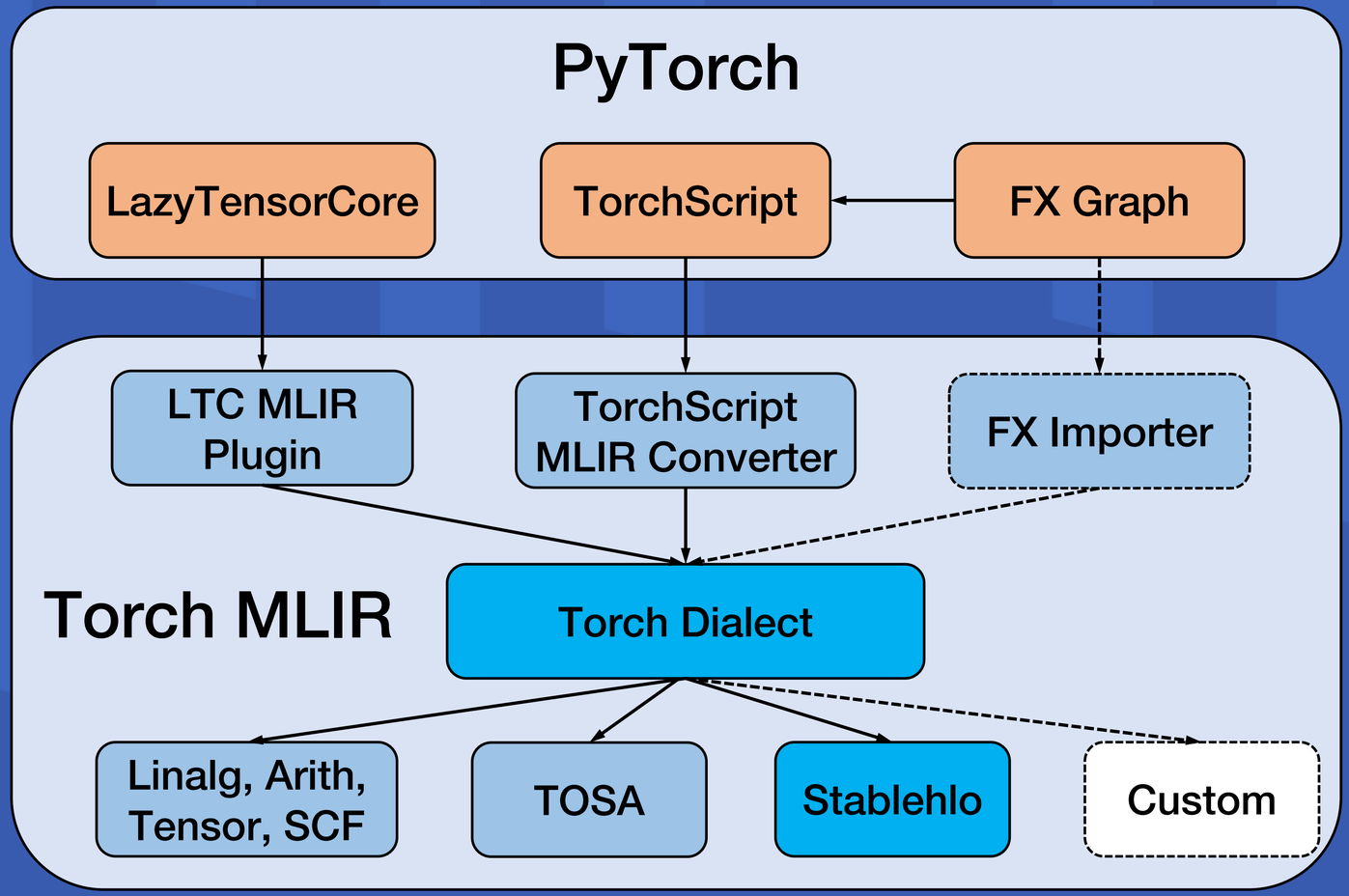
4. Flexible, extensible, high performance

1. What is ByteIR and advantages of ByteIR

2. Design and technical details

3. LLM training example and performance

ByteIR Torch Frontend – Torch MLIR



Github: https://github.com/llvm/torch-mlir

Torch MLIR Lowering

torch

```
class Module(torch.nn.Module):
        def __init__(self):
                super().__init__()
                self.linear = nn.Linear(10, 20)
        def forward(self, x):
                return self.linear(x)
```

torch.jit.scirpt

MLIR importer

torch dialect

```
func.func @forward(%arg0: !torch.tensor,
%arg1: !torch.tensor) -> !torch.tensor {
        %0 = torch.aten.mm %arg0,
%arg1 : !torch.tensor, !torch.tensor -> !torch.tensor
        return %0 : !torch.tensor
}
```

Shape/dtype inference

Value semantics analysis

torch dialect

```
func.func @forward(%arg0: !torch.vtensor<[20,
10], f32>, %arg1: !torch.vtensor<[10, 20], f32>) -
> !torch.vtensor<[20, 20], f32> {
        %0 = torch.aten.mm %arg0,
%arg1 : !torch.vtensor<[20, 10],
f32>, !torch.vtensor<[10, 20], f32> -
> !torch.vtensor<[20, 20], f32>
        return %0 : !torch.vtensor<[20, 20], f32>
}
```

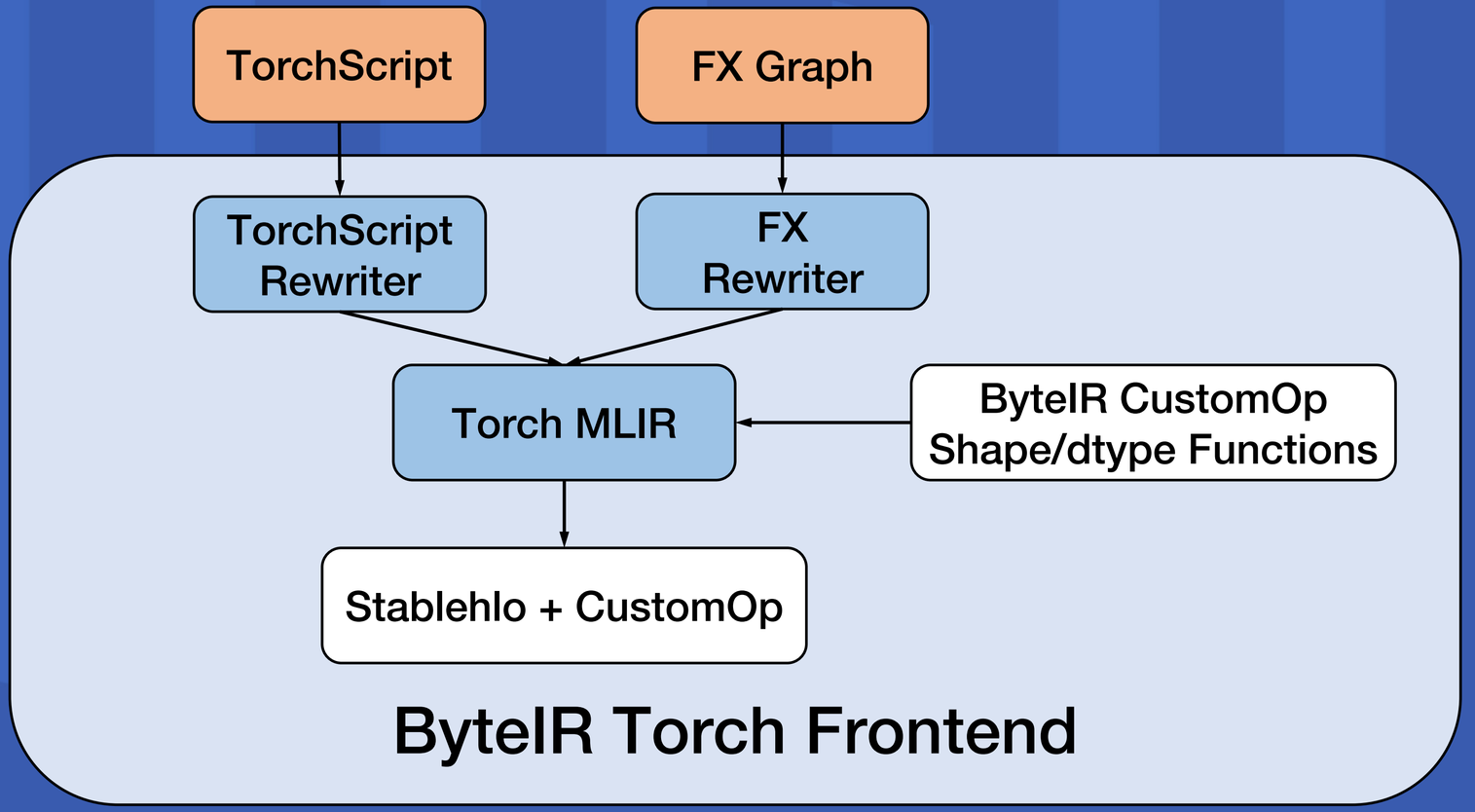
Decomposition

Lowering

stablehlo dialect

```
func.func @forward(%arg0: tensor<20x10xf32>,
%arg1: tensor<10x20xf32>) ->
tensor<20x20xf32> {
        %0 = ”stablehlo.dot"(%arg0, %arg1) :
tensor<20x10xf32>, tensor<10x20xf32> ->
tensor<20x20xf32>
        return %0 : tensor<20x20xf32>
}
```

ByteIR Torch Frontend



Able to provide coarse-grained operators

Corner Cases

```
def forward(self, x):
    x += 1
    return x
```

Success, but no effect on input

```
def forward(self, x, y):
    x += y
    z = x.view(-1, 4)
    x += 1
    return x, z
```

Failed, really overwrite same memory

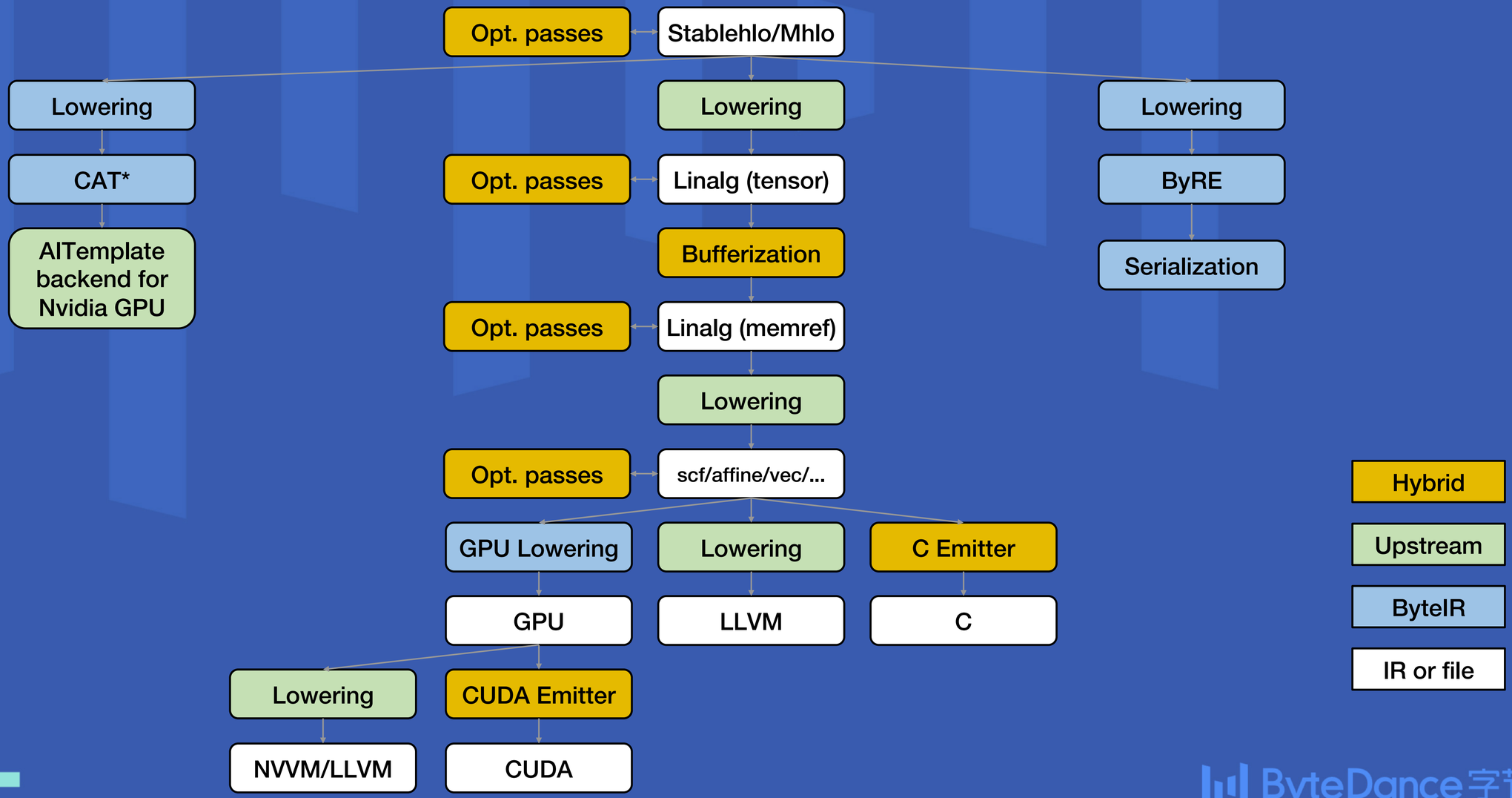
```
def forward(self, x):
    y = torch.as_strided(x, size, stride)
    return y
```

Failed, no abstraction of pointer/storage

- Don’t support dynamic if/for/while

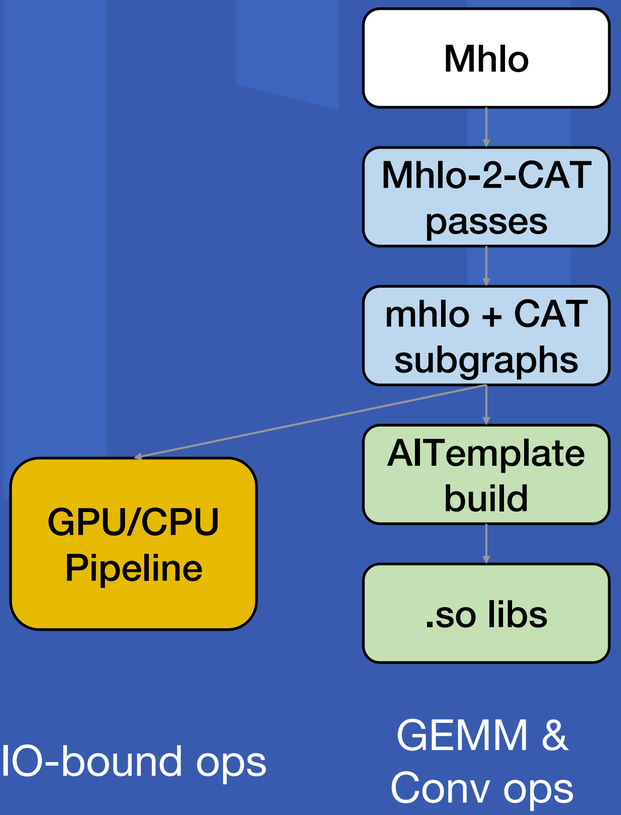
……

ByteIR Compiler Overview



Integrating AITemplate

We introduce **CAT** (Composable Algebra Template) dialect

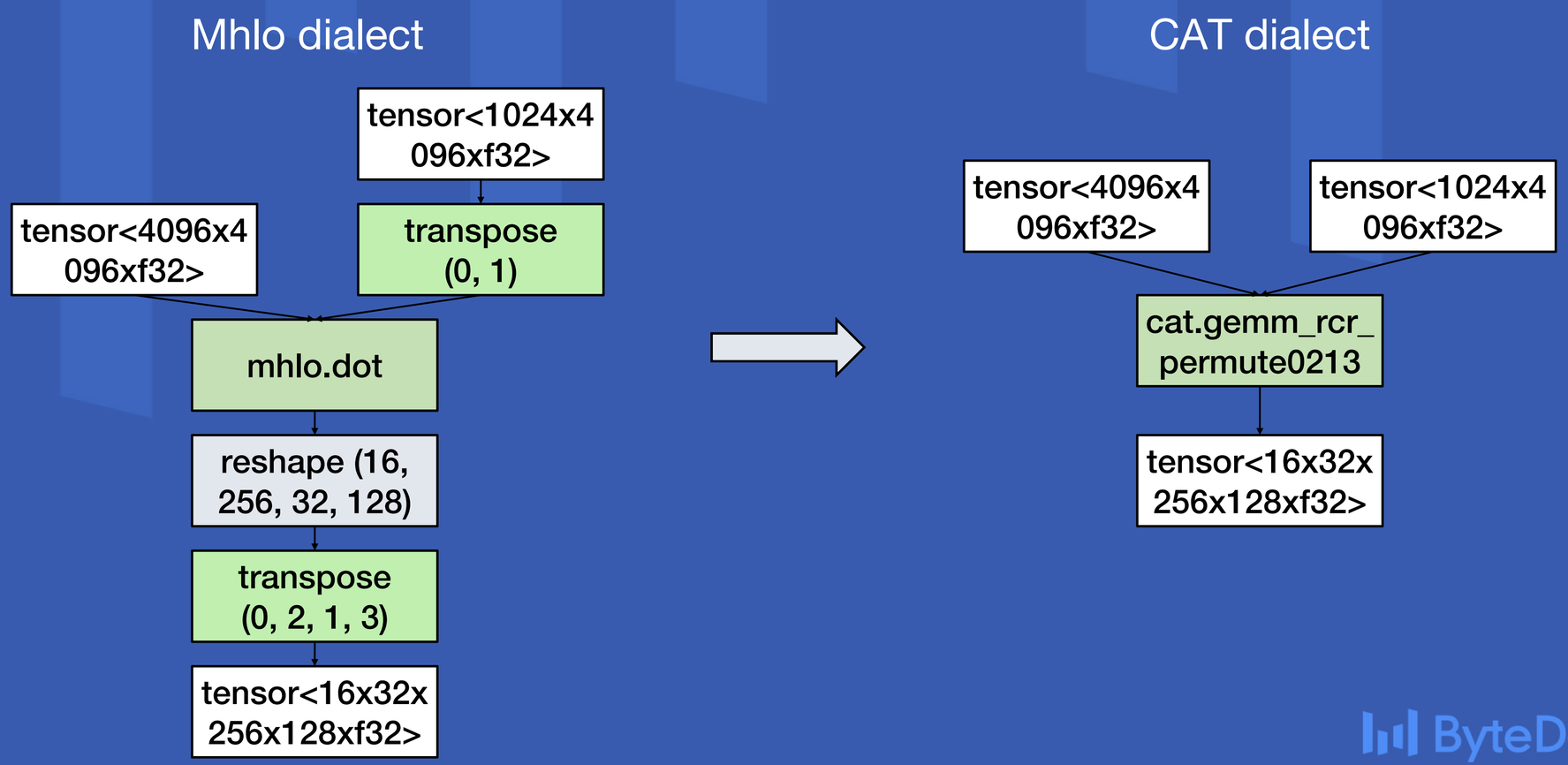


AITemplate: Meta's inference framework converting PyTorch NN to CUDA code (based on CUTLASS)

Mhlo-2-CAT Passes

Mhlo-2-CAT:

- Convert Mhlo ops to CAT ops (one CAT op corresponds to one AIT op)
- Eleminate redundant transpose/permuate ops



Linalg Tiling and Fusion

Mhlo op:

```
func.func @fuse_element(%arg0 ..., %arg1 ...)
    %0 = mhlo.some_elemwise_binary_1(%arg0, %arg1)
    %1 = mhlo.some_elemwise_binary_2(%0, %arg1)
    return %1
```

Linalg transformation

Linalg op:

```
func.func @fuse_element(%arg0 ..., %arg1 ...)
    %1 = linalg.generic {indexing_maps = ...,
                         iterator_types = ...} ins(%arg0, %arg1) outs... {
    ^bb0(%in0, %in1, %out)
        %2 = arith.some_elemwise_binary_1(%in0, %in1)
        %3 = arith.some_elemwise_binary_2(%2, %in1)
        linalg.yield %3 ...
    }
    return %1
```

ByetIR's Linalg Extension

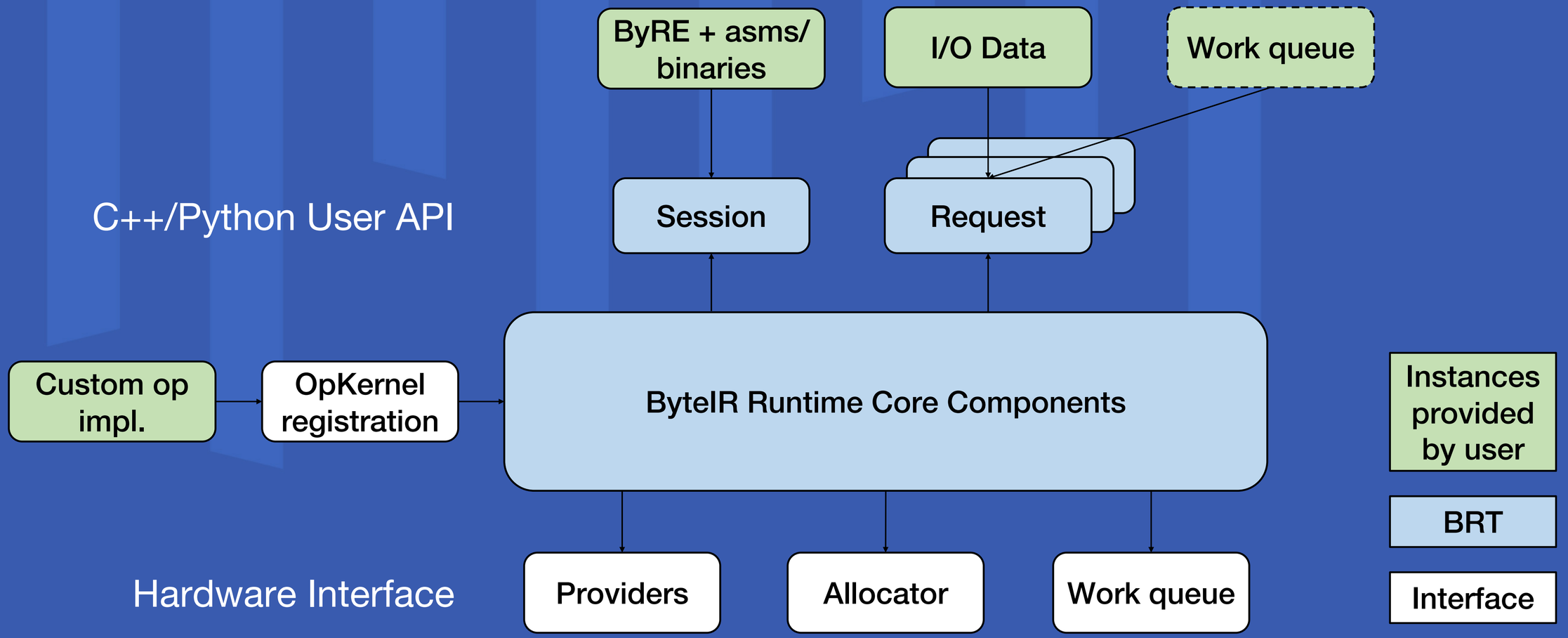
More ops:
- Alias, Diag, Scan, Scatter, Softmax, TopK
- support transformations of extended ops

Enhanced fusion transformations:
- producer-consumer & input-sharing fusion
- tiling along reduction axis correction
- intermediates as outputs within fusion
- intermediate tensor dim simplification
- map ops to generic ops conversion
- ...

Other introduced transformations:
- Collapse dims transformation
- Fuse operands transformation
- ...

Benefits:
- Extreme IO-bound op fusion
- Lower overhead for fused ops (Exploit GPU DRAM bandwidth)

ByteIR Runtime (BRT) Overview



BRT Interface for Hardware

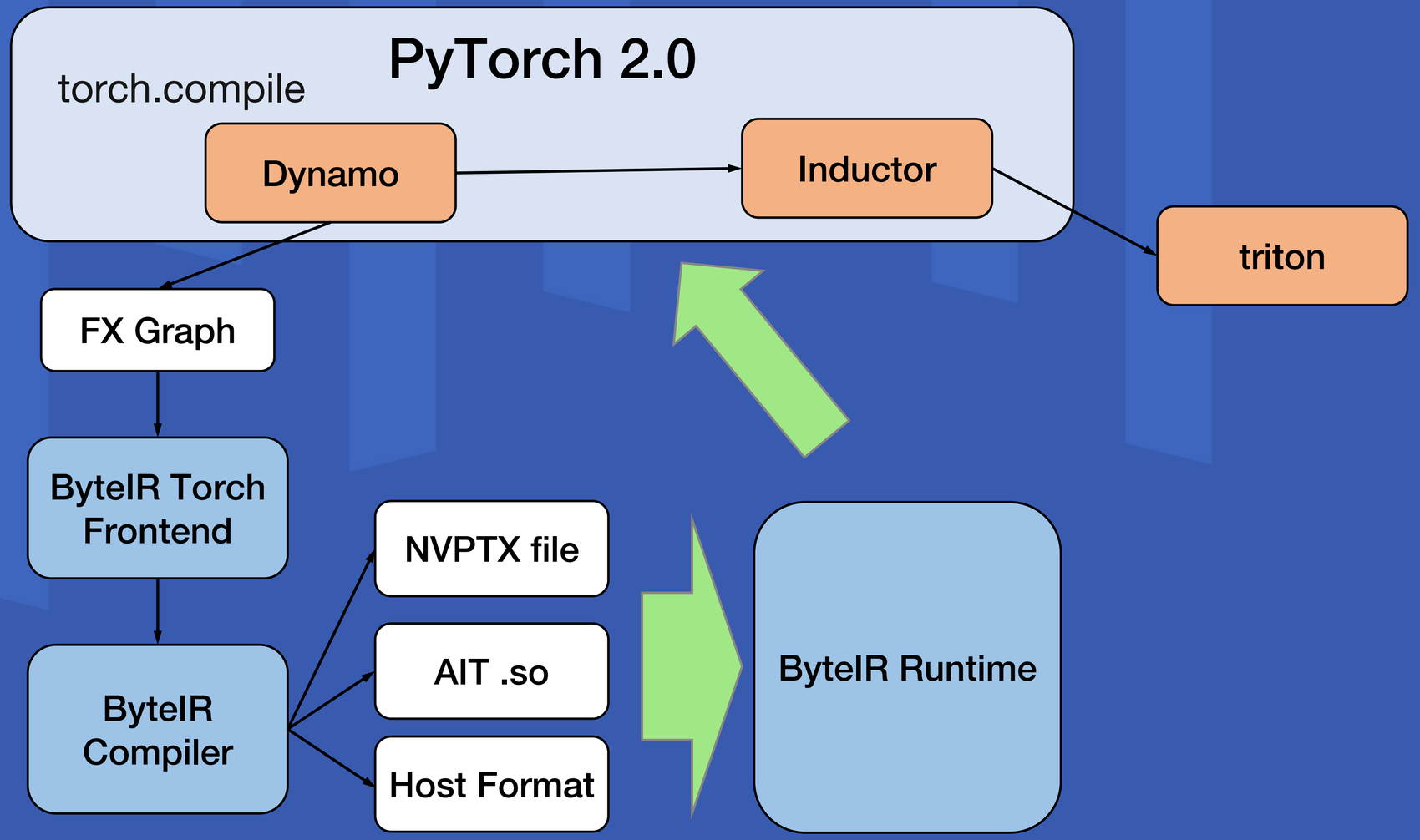
Provider	A collection of op implementation • e.g., mm, d2h/h2d memcpy
Work Queue	Abstraction(like CUDAStream) for execution order
Allocator	Memory Allocate/Free

1. What is ByteIR and advantages of ByteIR

2. Design and technical details

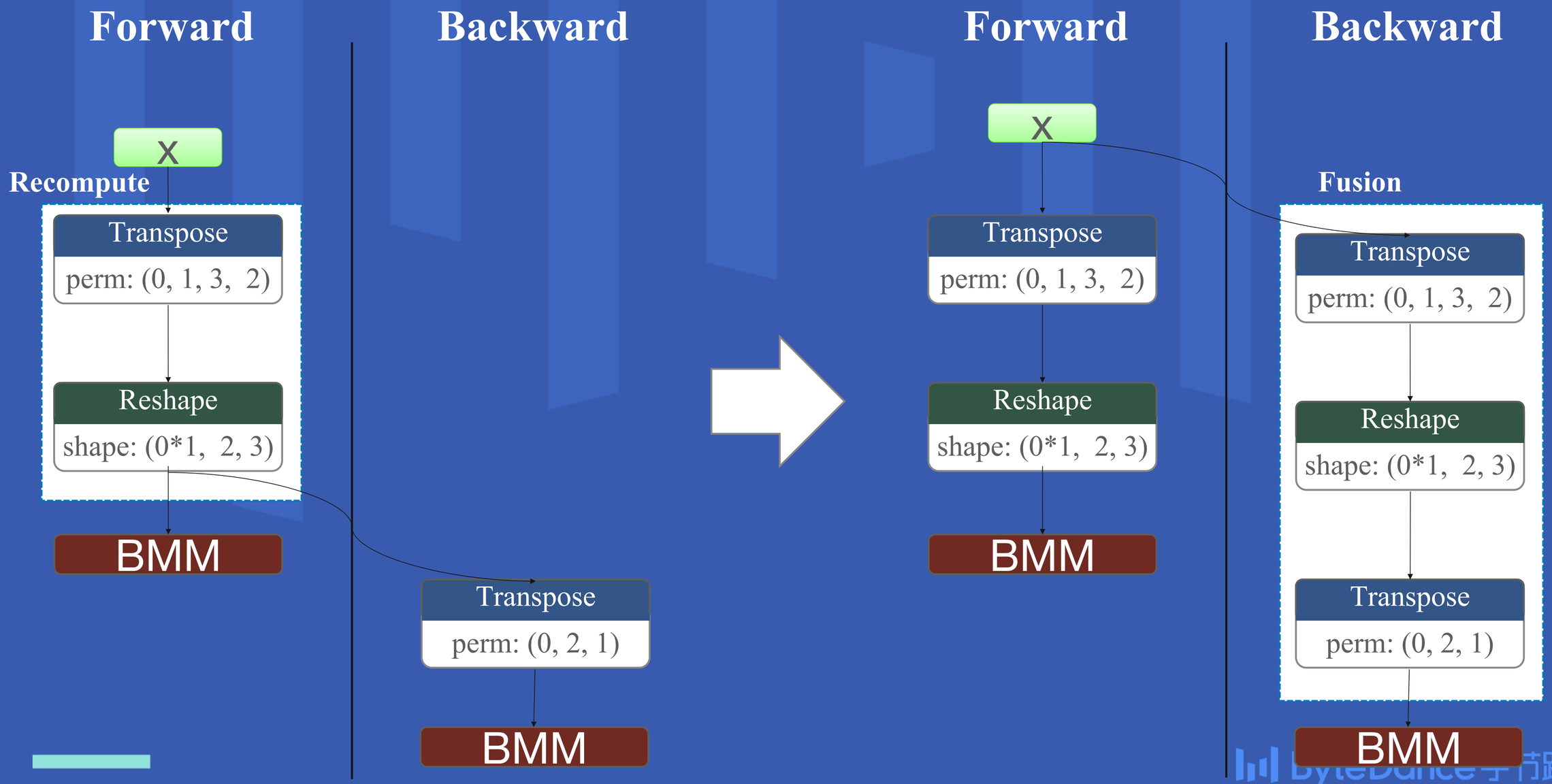
3. LLM training example and performance

ByteIR LLM Training Compilation Pipeline



Demo Link: https://github.com/bytedance/byteir/tree/main/frontends/torch-frontend/examples/demo

Optimization in FW/BW Partition



One case of our partition strategy

ByteIR PyTorch Compile Example

```
from byteir import byteir_compile_fx

model = make_model(model_name)

# 1, compile with byteir
optimized_model = torch.compile(model, backend=byteir_compile_fx)

# 2, execution as usual
data = make_data(optimized_model, model_name, device)
model.zero_grad(set_to_none=True)
with torch.cuda.amp.autocast(enabled=True, dtype=torch.float16):
    # forward compile
    loss = compute_loss(optimized_model, data)
    # backward compile
    loss.backward()
```

Demo Link: https://github.com/bytedance/byteir/tree/main/frontends/torch-frontend/examples/demo

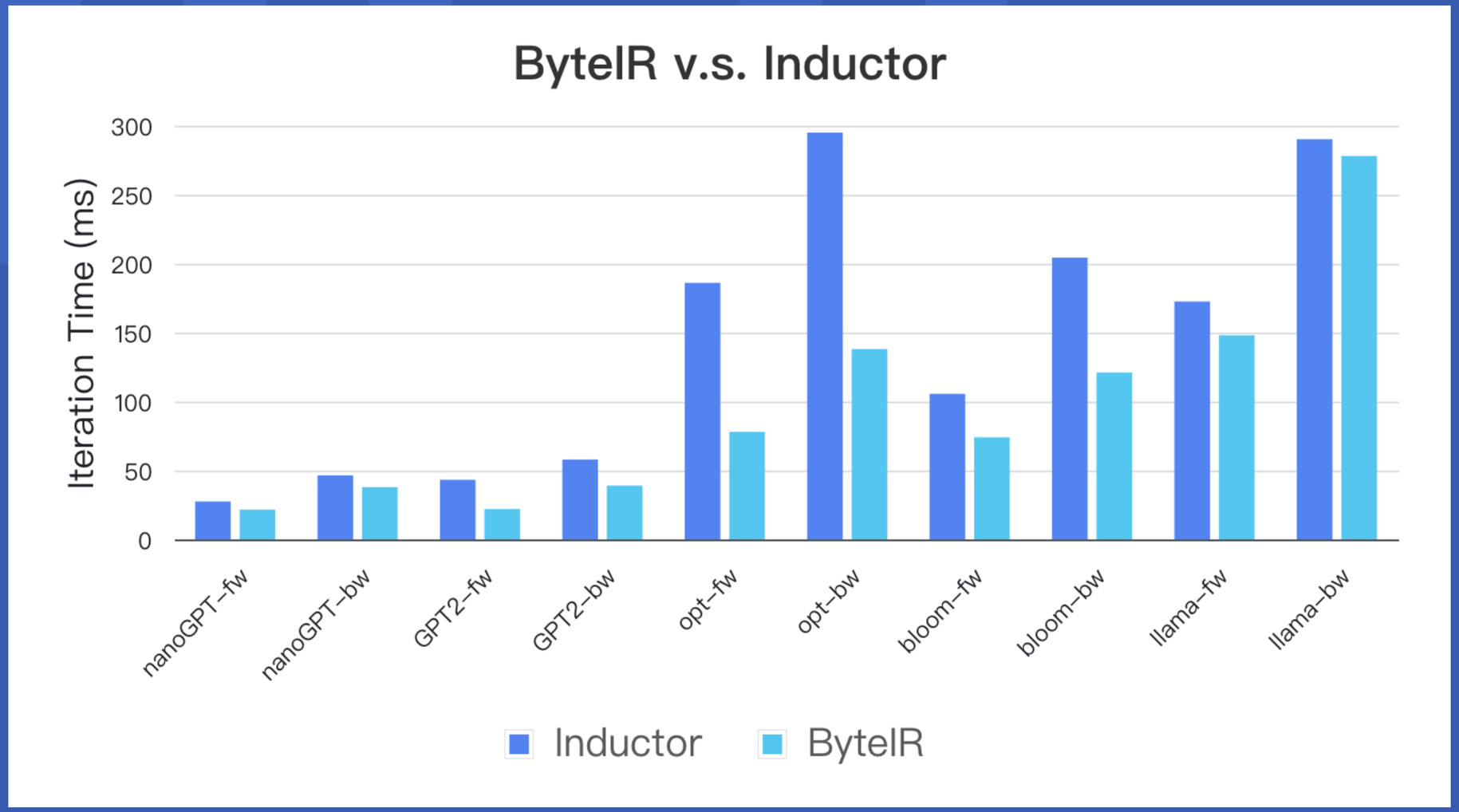
ByteIR Runtime Example

```
import brt

session = brt.Session()
session.load(byre_model_path)
req = session.new_request_context(torch.cuda.current_stream())
inputs, outputs = [], []

# init input/output data
for offset in session.get_input_arg_offsets():
    inputs.append(torch.randn(session.get_static_shape(offset),
                  dtype=dtype, device="cuda"))
    req.bind_arg(offset, inputs[-1].data_ptr())
...

req.finish_io_binding()
req.run()
req.sync()
```

Demo Link: https://github.com/bytedance/byteir/tree/main/frontends/torch-frontend/examples/demo

Performance

- Flash Attention 2
- Elementwise Fusion
- AITemplate
- Reduce Codegen



Demo Link: https://github.com/bytedance/byteir/tree/main/frontends/torch-frontend/examples/demo

Conclusion & Future Work

We introduce ByteIR: a framework-to-hardware compiler solution

- Friendly to PyTorch and GPU/ASIC
- PyTorch 2.0 training/inference demo on LLM

Future Work:

- Distributed support
- TensorCore MMA Codegen

Website: https://byteir.ai

Github: https://github.com/bytedance/byteir

ByteDance字节跳动

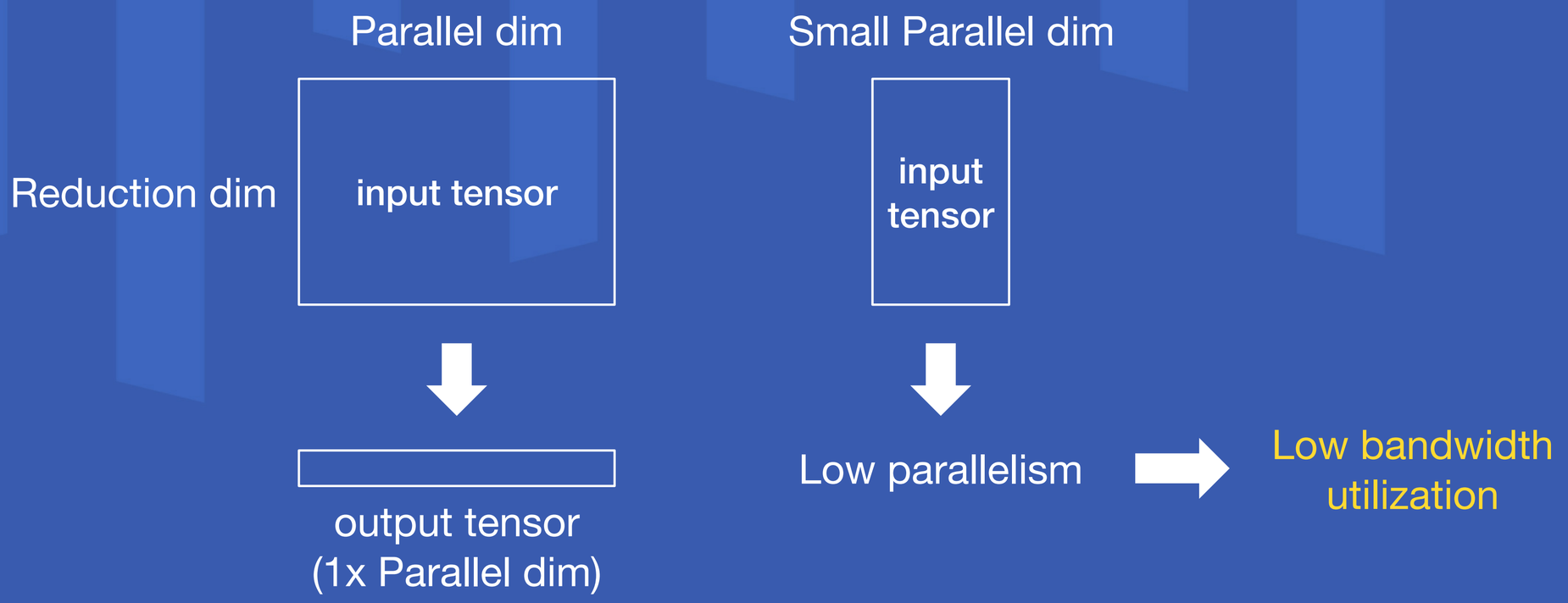
THANKS

https://byteir.ai

ByteDance字节跳动

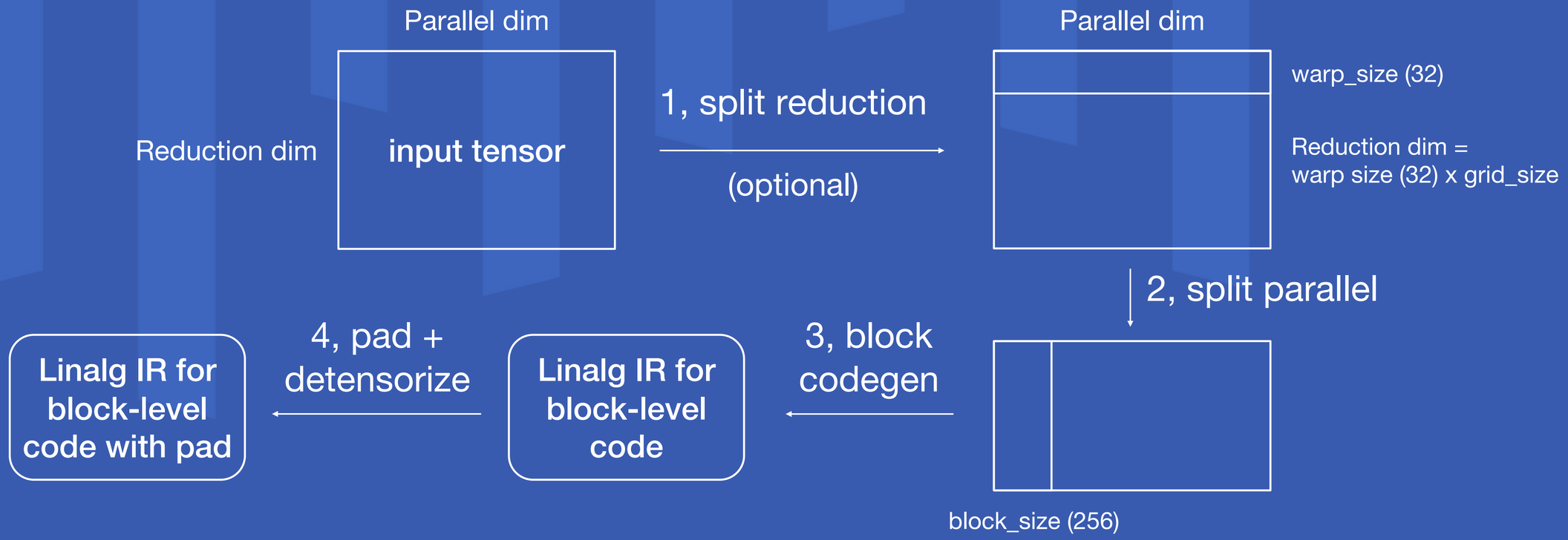
Appendix 1: Reduce Op Optimization (Fusion)

Optimization 1: Fusing reduce op with producer ops



Appendix 2: Reduce Op Optimization (Tiling)

Optimization 2: Parallelizing reduce dimension



We use utilize our LinalgExt transformations to achieve best tiling efficiency