JIT in MegEngine

王彪
wangbiao@megvii.com

MEGVII 旷视

天元 MegEngine

MegEngine

PYTHON PROGRAMMING

训练

推理

天元 MegEngine

- ✓ 训推一体
- ✓ 动静结合
- ✓ 多平台高性能支持

天元官网:https://megengine.org.cn/

GitHub(欢迎star):https://github.com/MegEngine

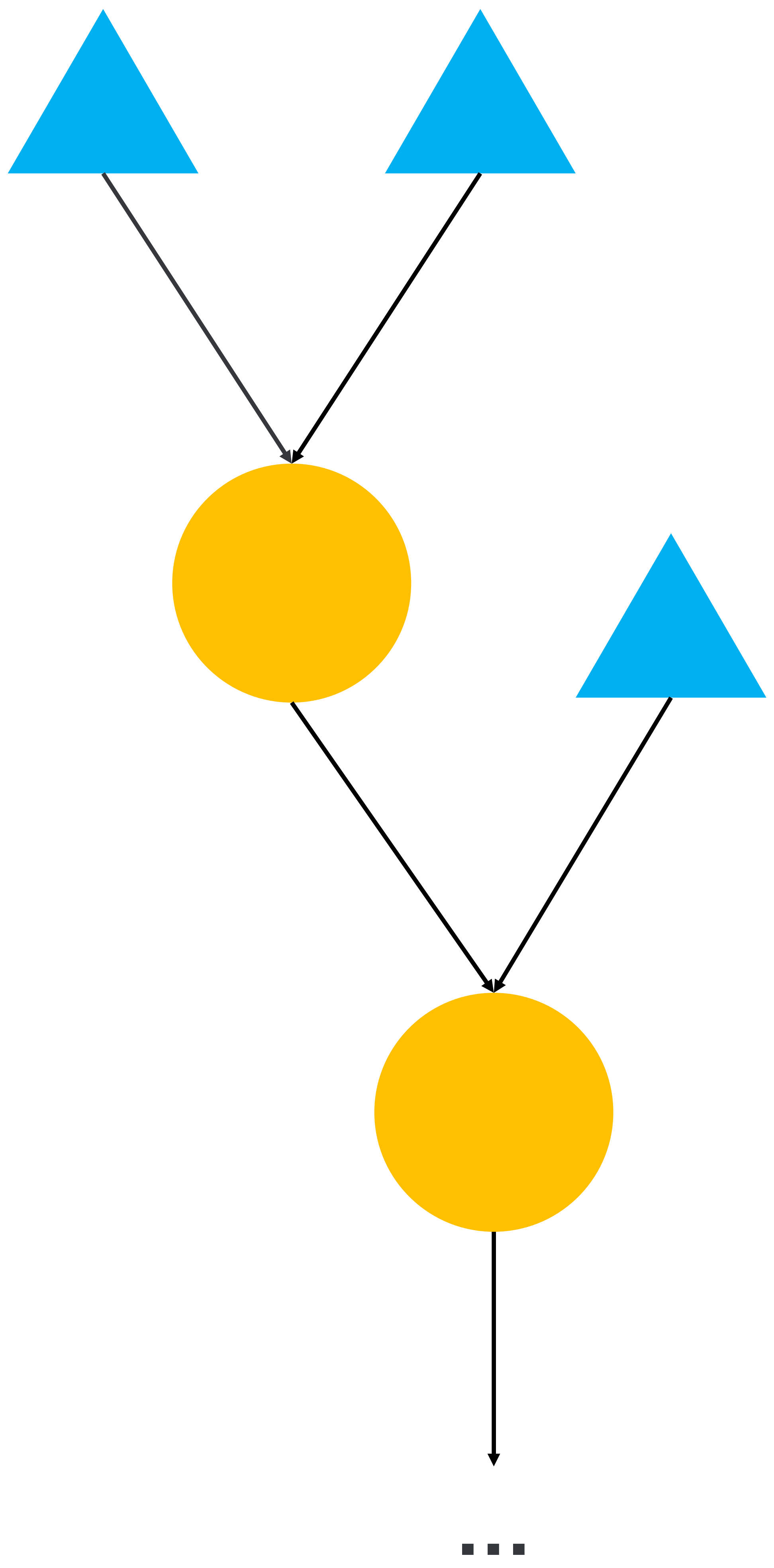
MegEngine 架构

- **计算接口**
  - API 接口
    - Python 接口
    - C++ 接口
- **图表示**
  - 动态图表示
  - 静态图表示
- **优化与编译**
  - 核心计算图
    - 自动求导器
    - 图优化
    - 图编译
- **运行时管理**
  - 计算调度
    - 执行流
    - 调度器
  - 内存管理
    - 静态内存管理
    - 动态内存管理
    - 亚线性优化
- **计算内核**
  - 高性能算子
    - X86
    - CUDA
    - ARM
    - ASIC
    - …
  - 高性能异构通信库

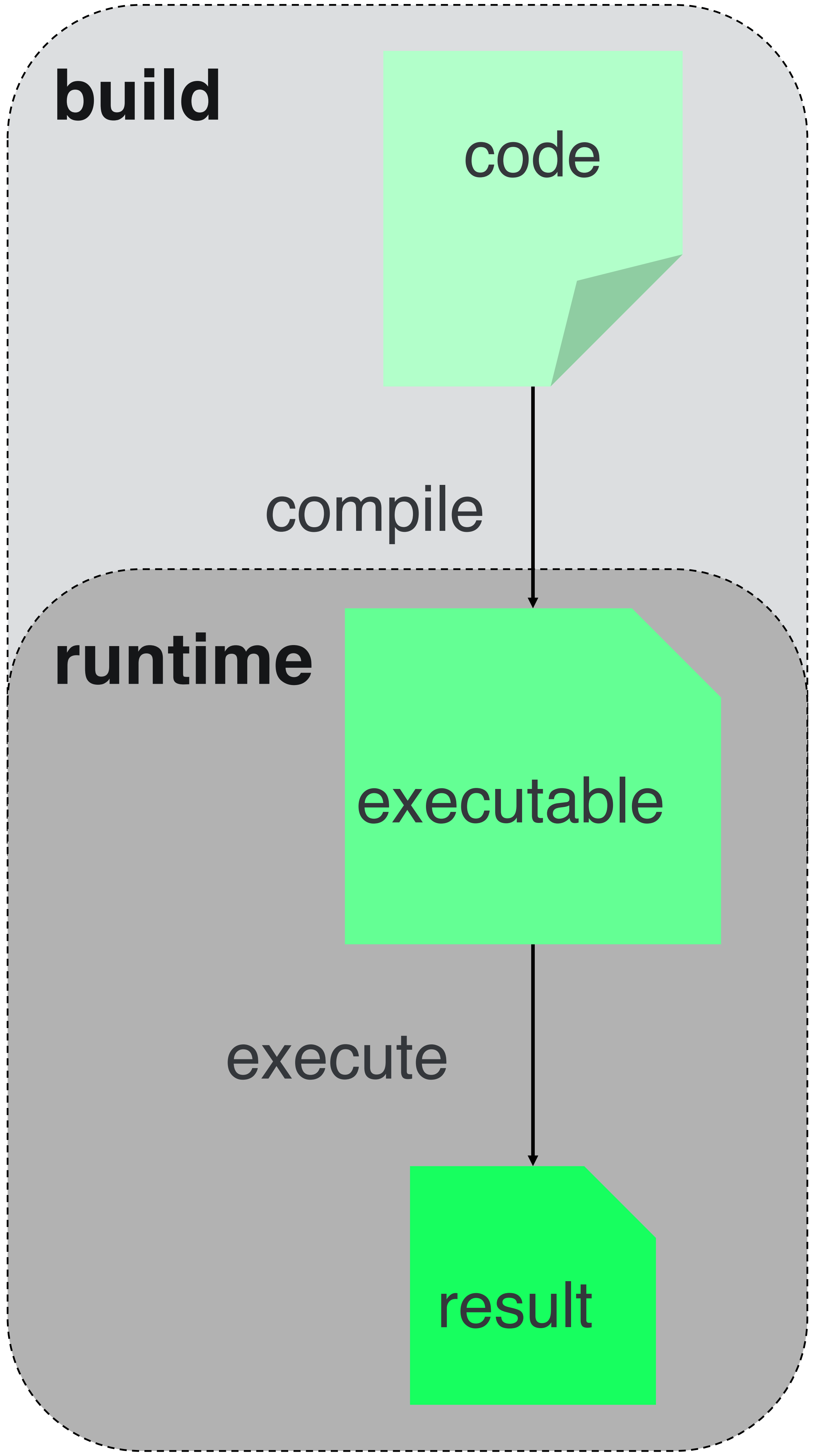
MegEngine 静态图模式

MEGVII 旷视

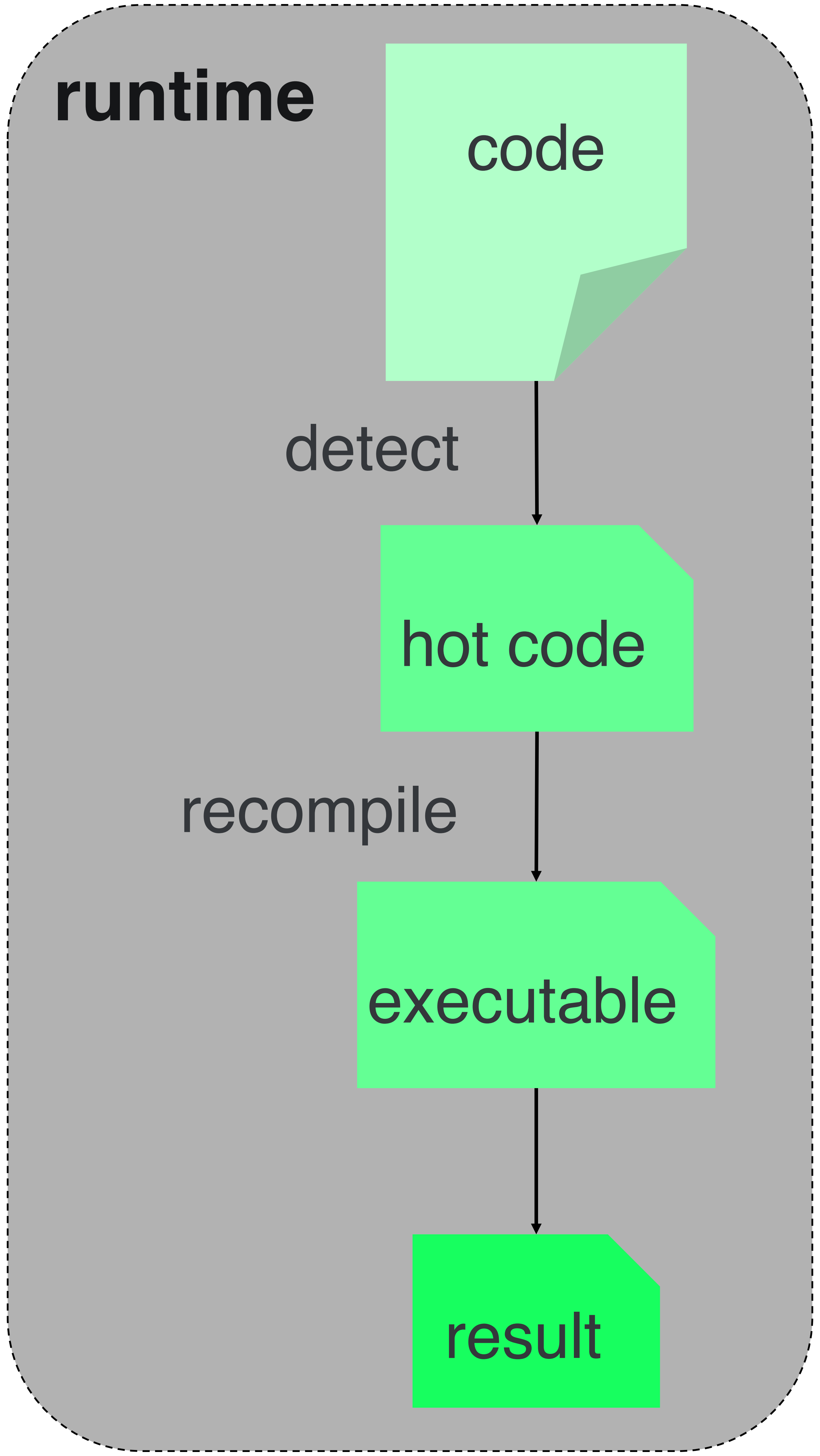
```
if __name__ == '__main__':
    gm = ad.GradManager().attach(model.parameters())
    opt = optim.SGD(model.parameters(), lr=0.0125, momentum=0.9,
weight_decay=1e-4,)
    # 通过 trace 转换为静态图
    @trace(symbolic=True)
    def train():
        with gm:
            logits = model(image)
            loss = F.loss.cross_entropy(logits, label)
            gm.backward(loss)
        opt.step()
        opt.clear_grad()
        return loss
    loss = train()
    loss.numpy()
```



AOT and JIT



Ahead-of-time (AOT)



Just-in-time (JIT)

Outline

MEGVII 旷视

① Motivation

② Detect Fusion

③ JIT Compilation

④ Evaluation

MegEngine 中的 Elemwise 模式

MEGVII 旷视

Unary	
RELU	NEGATE
ABS	SIGMOID
ACOS	SIN
ASIN	TANH
CEIL	FAST_TANH
COS	ROUND
EXP	ERF
EXPM1	ERFINV
FLOOR	ERFC
LOG	ERFCINV
LOG1P	HSWISH

Binary	
ABS_GRAD	SWISH_GT0
ADD	TANH_GRAD
FLOOR_DIV	TRUE_DIV
MAX	LOG_SUM_EXP
MIN	LT
MOD	LEQ
EXP	EQ
MUL	SHL
POW	SHR
SIGMOID_GRAD	FAST_TANH_GRAD
SUB	RMULH
ATAN2	H_SWISH_GRAD

Fused Binary	
FUSE_ADD_RELU	max(x+y, 0)
FUSE_ADD_SIGMOID	1/(1+exp(-(x+y)))
FUSE_ADD_TANH	tanh(x+y)
FUSE_ADD_H_SWISH	hawish(x+y)

More	
COND_LEQ_MOV	x <= y ? z : 0
FUSE_MUL_ADD3	a * b + c
FUSE_MUL_ADD4	a * b + c * d

CNN 中的 Elemwise 分析

MEGVII 旷视

model	batchsize	elemwise computation / total computation(%)	elemwise time/total time(%)
resnet50	1	0.102054232	4.6
	8	0.102085366	10.8
	16	0.102080177	11.9
mobilenetV2	1	0.703333333	4.1
	8	0.704592902	8.9
	16	0.704627697	11.9
vgg16	1	0.02927408	5.8
	8	0.029277172	7.1
	16	0.029342568	9.4

training 图纯 device kernel 执行时间

Elemwise Fusion

设子图中的 opetator 集合为 $\mathbf{O_i}$ (0 <= i <= N)

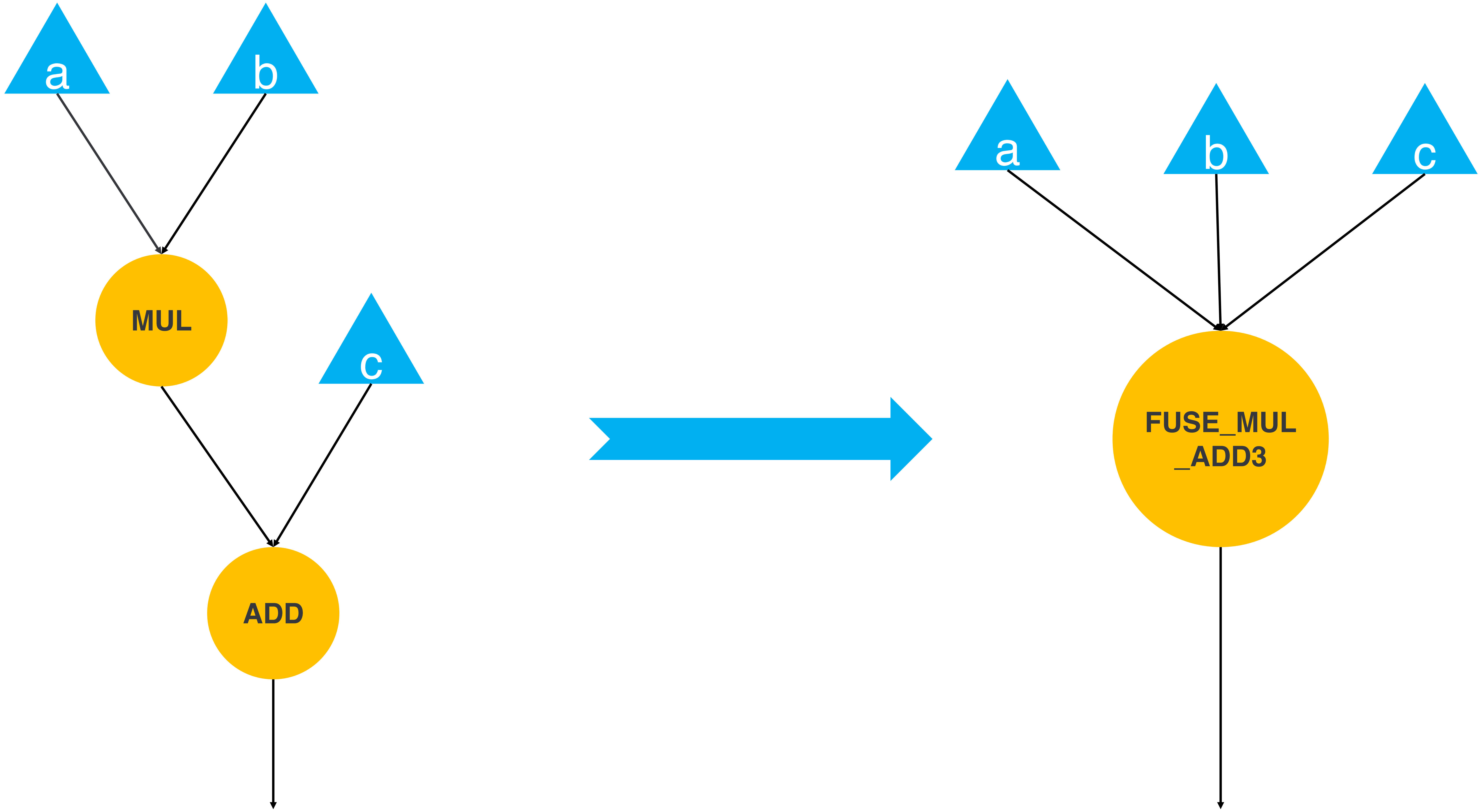
INP_i 表示第 i 个 operator 的 input 个数

OUP_i 表示第 i 个 operator 的 output 个数

则整个子图的读次数为 $\sum_{i=0}^{N} \mathrm{INP_i}$，写次数为 $\sum_{i=0}^{N} \mathrm{OUP_i}$.

Elemwise Fusion 可以减少子图中的 operator 的数量，自然可以减少 operator 之间的读写次数.

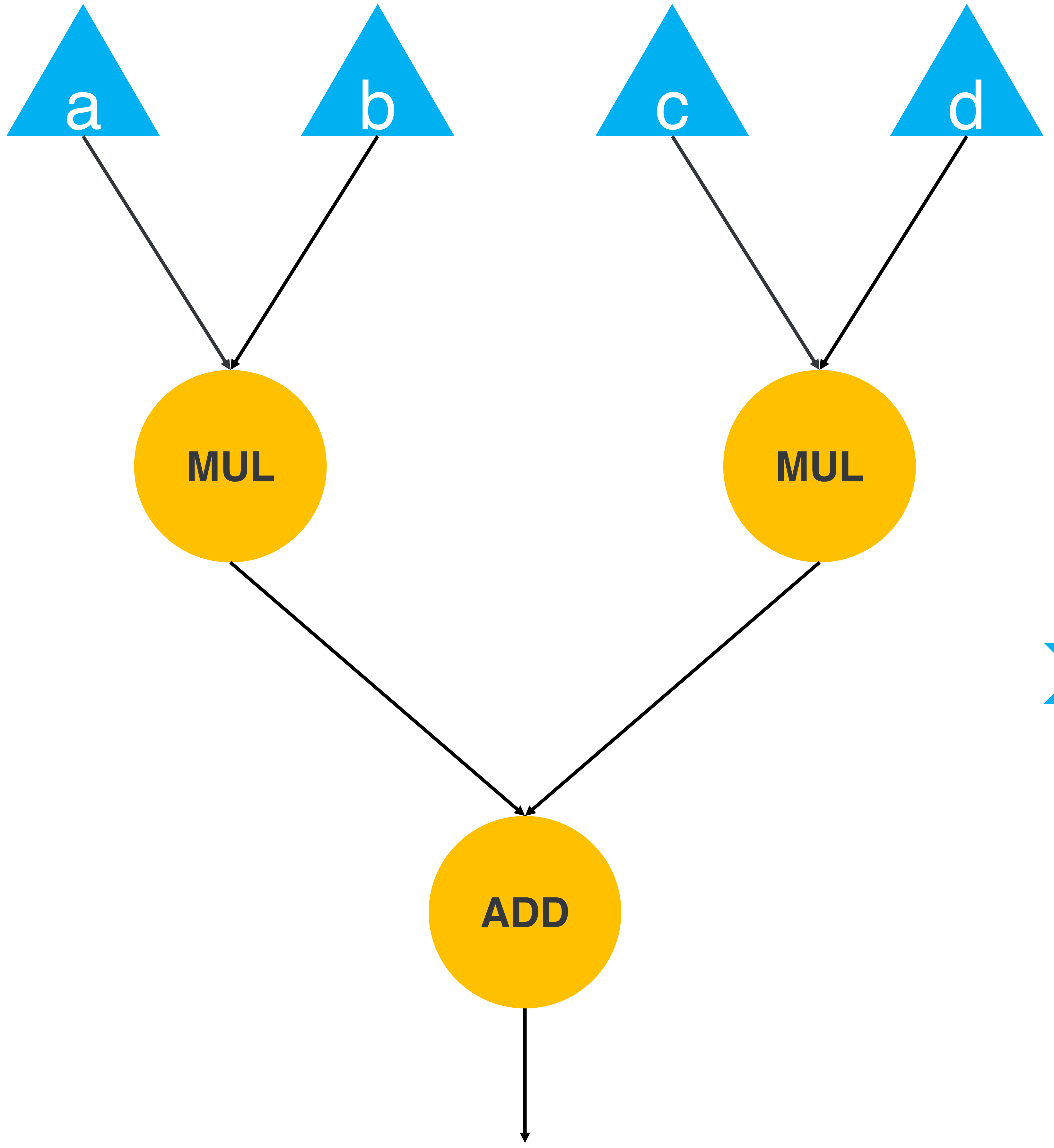
Elemwise Fusion



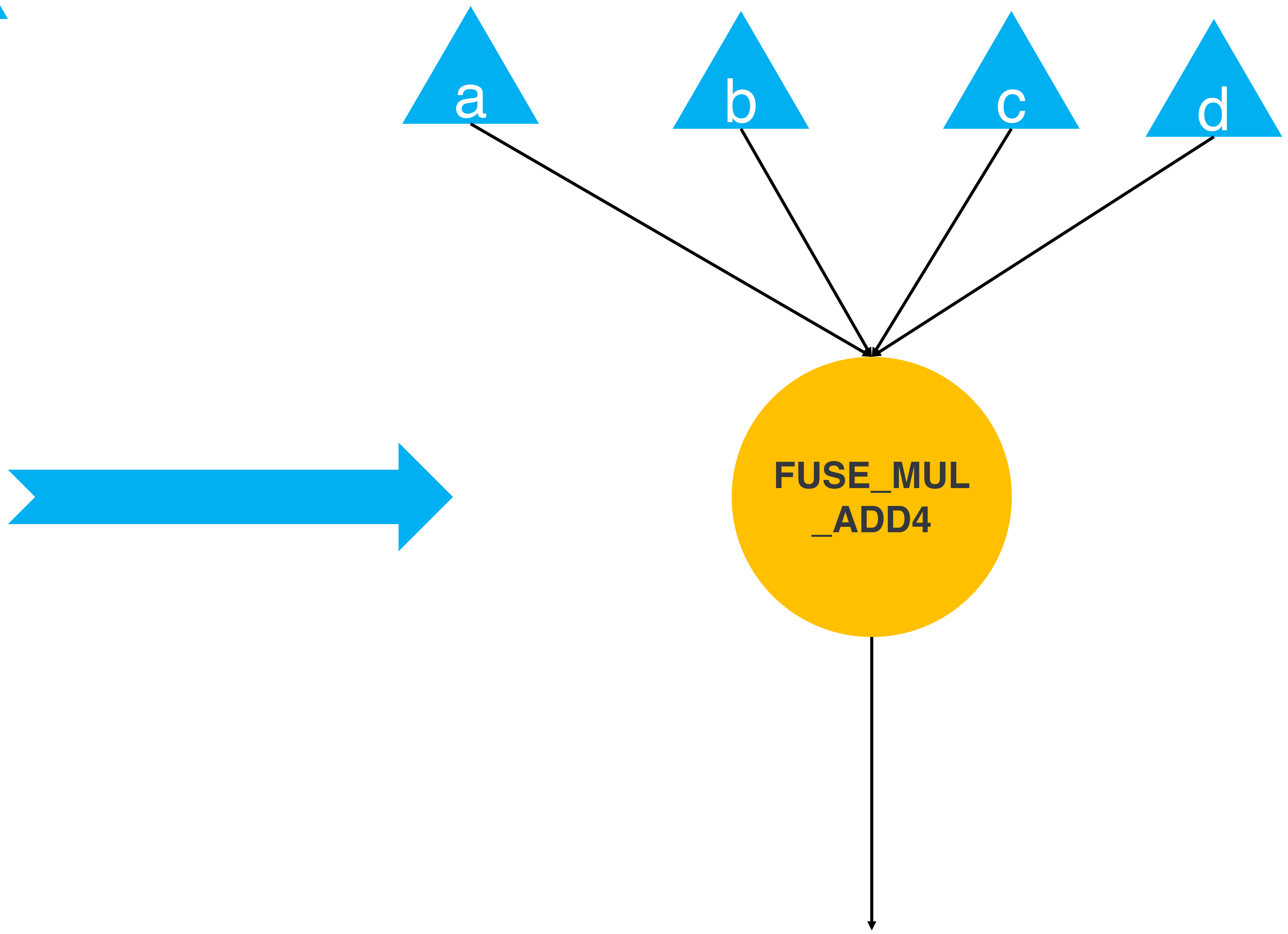
4 次读, 2 次写

3 次读, 1 次写

Elemwise Fusion

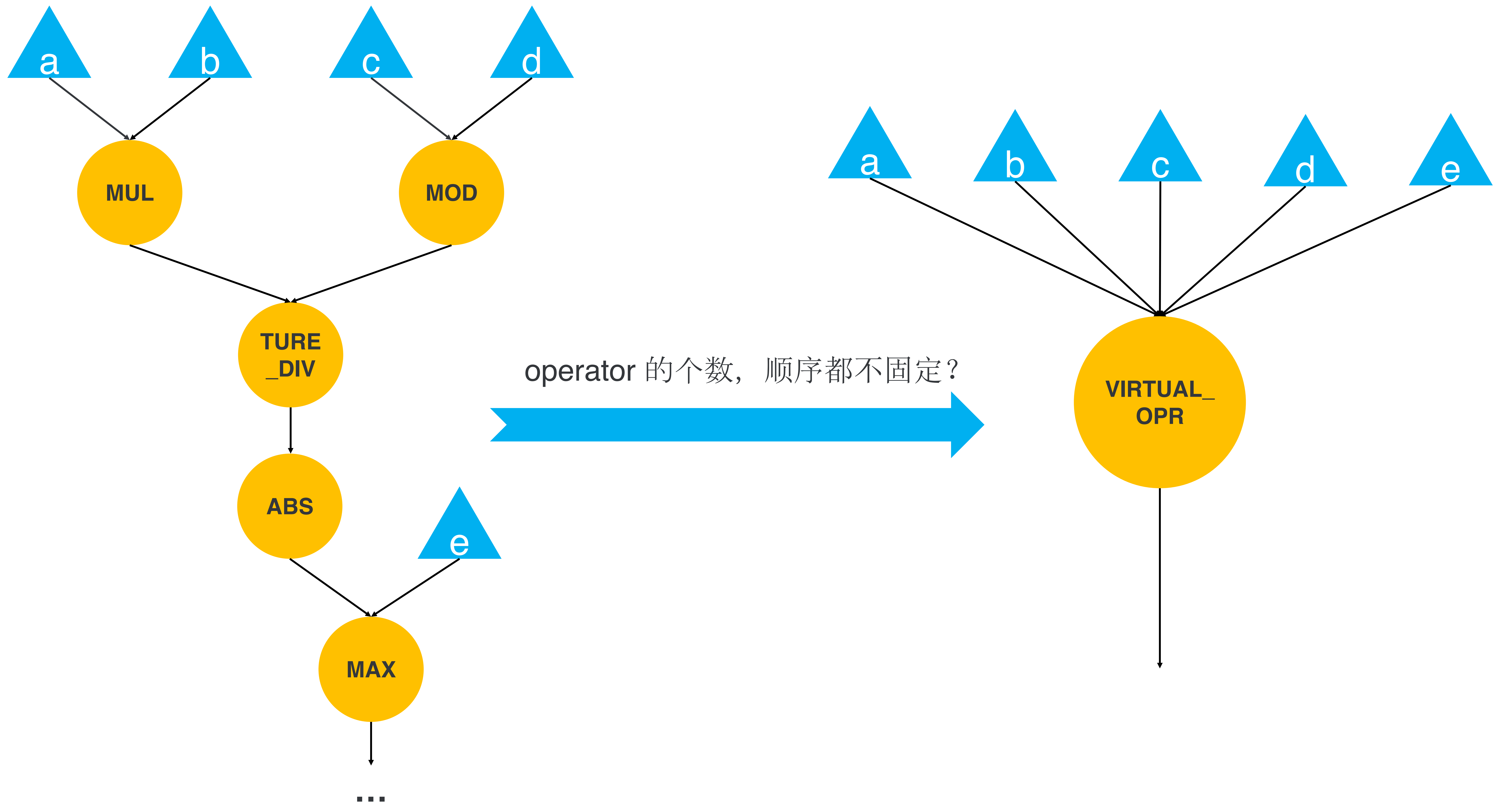


6 次读, 3 次写



4 次读, 1 次写

Elemwise Fusion



为啥用 JIT

CNN 模型特征

- 静态图模式下模型结构一般不变
- 模型训练过程历经很多个 iter/min_batch, 输入数据的 shape 不变

使用 JIT 的好处

- 仅需要在首次运行时编译一次
- 有较强的可移植性
- 解决 elemwise 模式组合爆炸的问题

OUTLINE

MEGVII 旷视

① Motivation

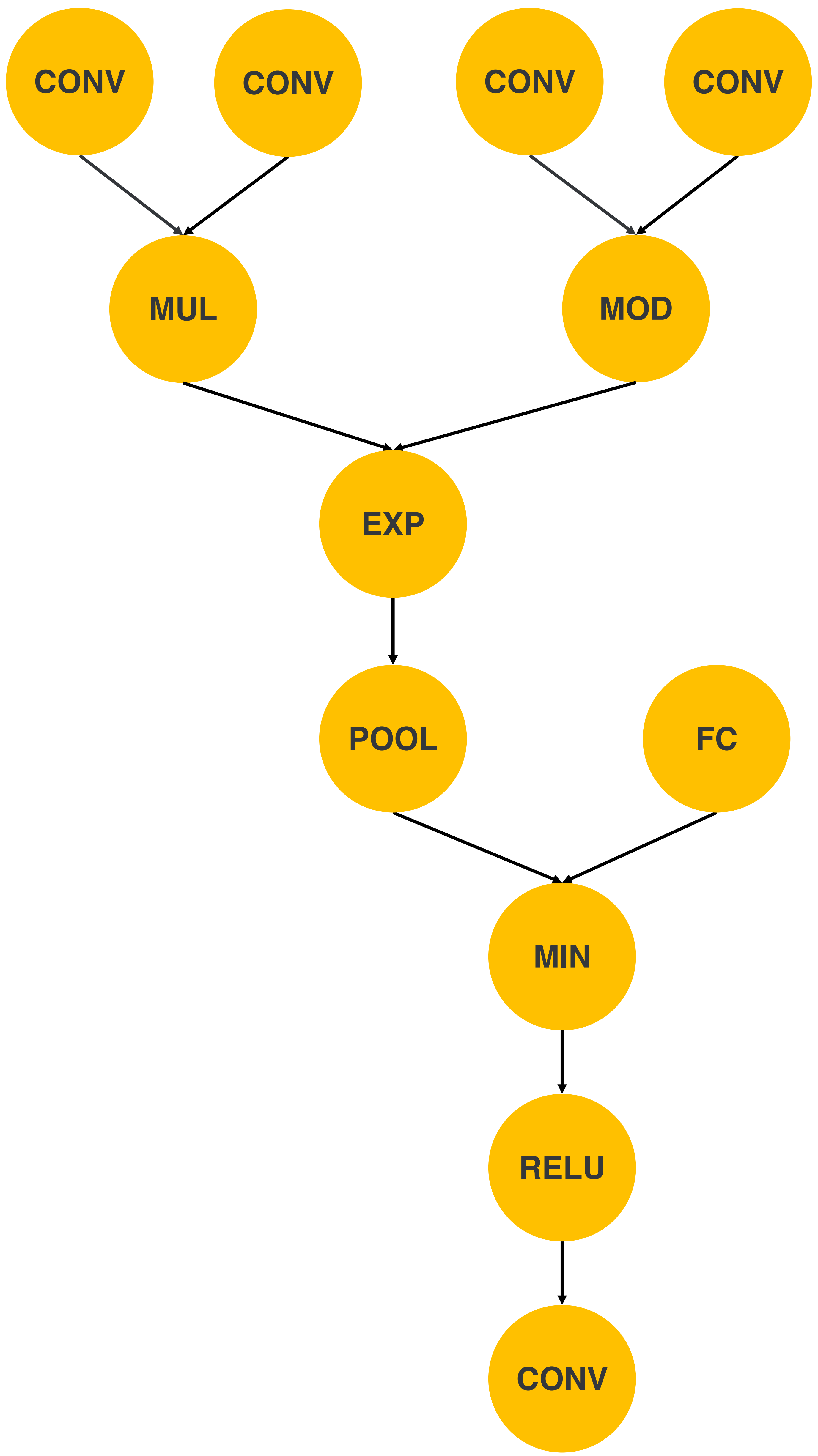
② Detect Fusion

③ JIT Compilation

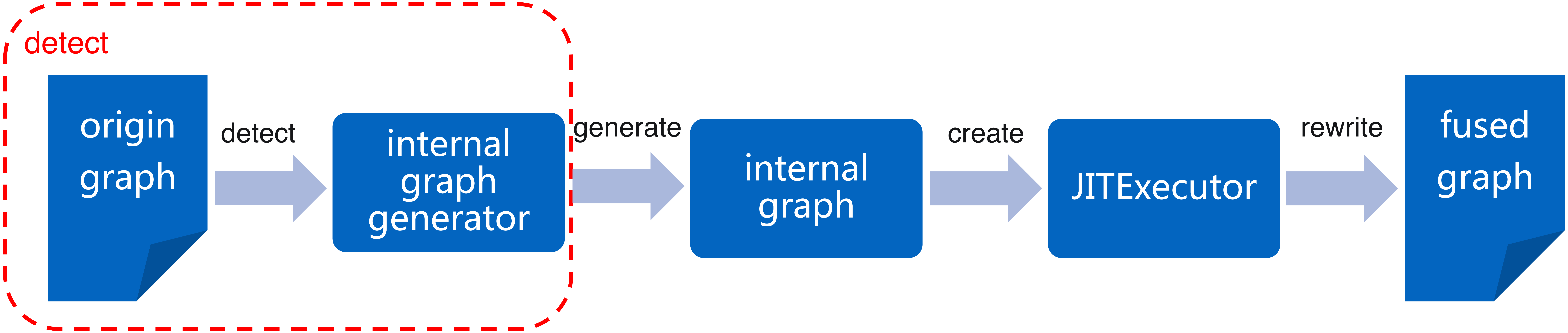
④ Evaluation

Detect Fusion

该计算图中可融合的子图是？



Detect Fusion



Detect

设

G : 计算图

opr : 图 G 中的算子

var : opr 的输入/输出

Detect

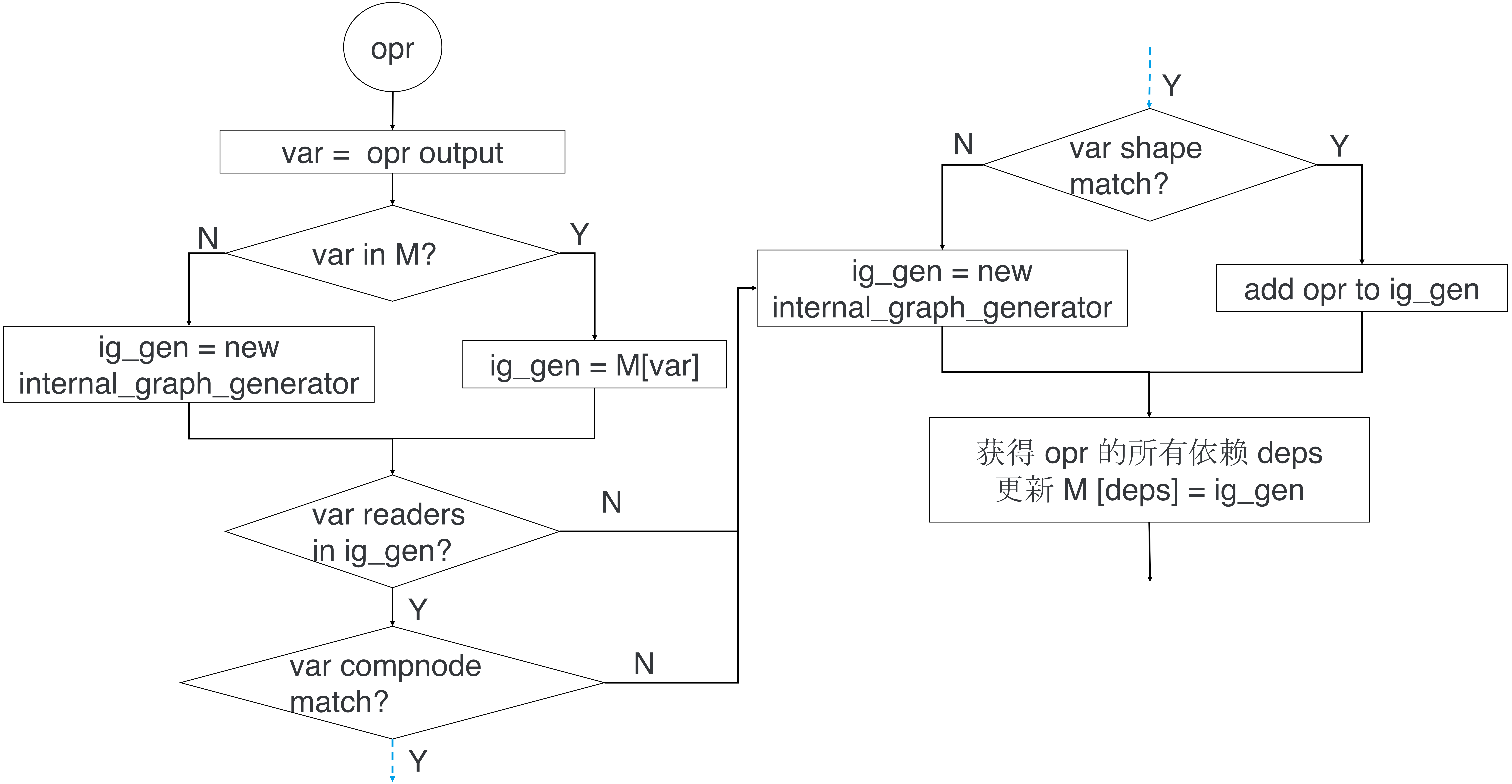
Algorithm : detect – 检测可以 fuse 的子图

input : G – 原始计算图

output : M – 子图的 output var 与对应的 internal graph generator

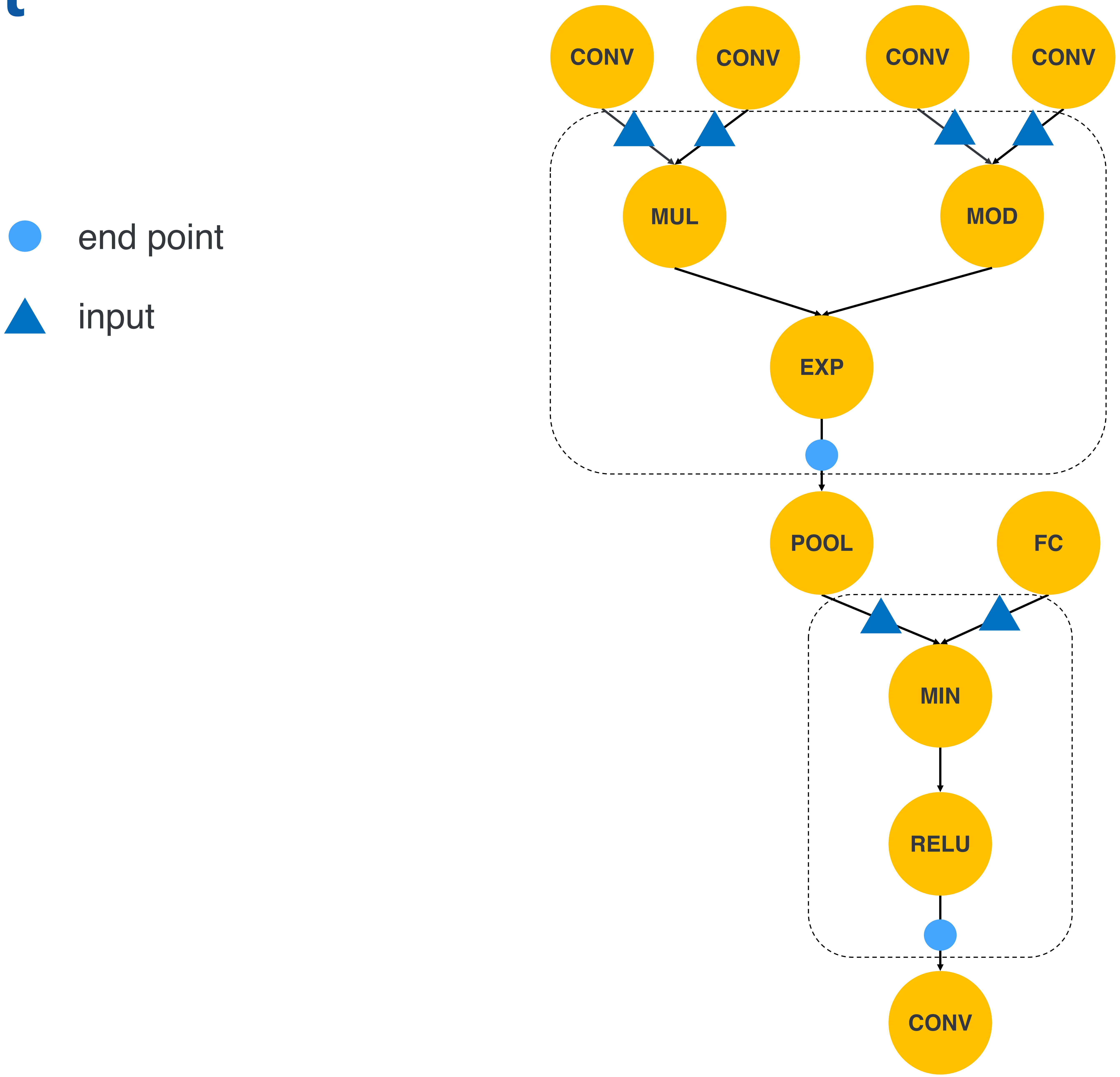
1. 按照逆拓扑序列遍历图 G 中的算子 opr
2. 如果 opr 不是 Elemwise/PowC/TypeCvt/Reduce/Dimshuffle/JITExecutor，返回步骤1
3. 如果 opr 的 input/output 数据类型不是 float32/float16，返回步骤1
4. **process_opr(opr)**
5. 转到步骤 1

Detect

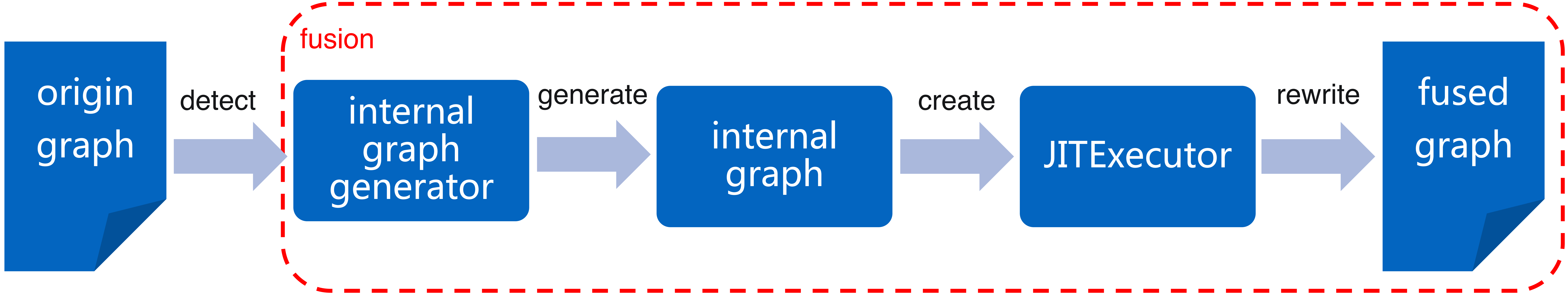


opr
var = opr output
var in M?
N
Y
ig_gen = new internal_graph_generator
ig_gen = M[var]
var readers in ig_gen?
N
Y
var compnode match?
N
Y
Y
var shape match?
N
Y
ig_gen = new internal_graph_generator
add opr to ig_gen
获得 opr 的所有依赖 deps
更新 M [deps] = ig_gen

Detect


end point
input
CONV
CONV
CONV
CONV
MUL
MOD
EXP
POOL
FC
MIN
RELU
CONV

Detect Fusion



Fusion

Algorithm : fusion – 将检测出的子图 fuse 成一个 opr

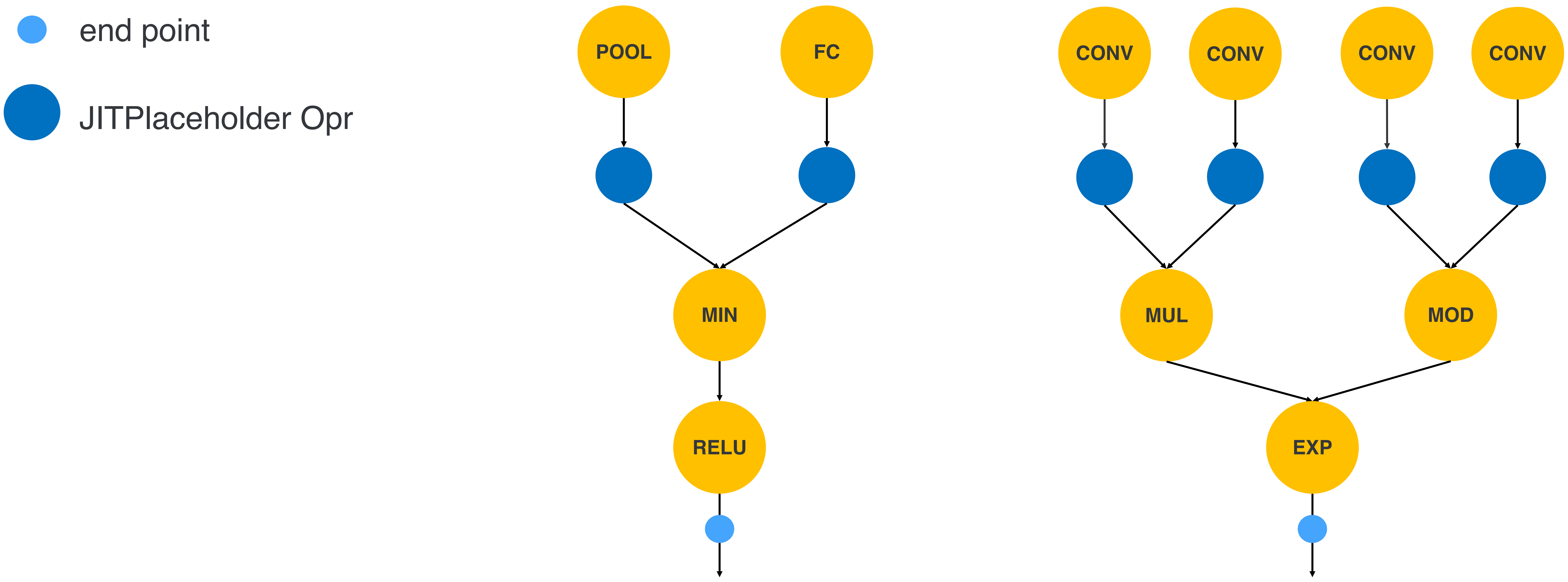
input : G – 原始计算图； M – 子图的 output var 与对应的 internal graph generator ;

output : fuse 后的计算图 G'

1. 按照拓扑序列遍历图 G 中的算子 opr
2. 若 opr 的 input var 不是 endpoint, 返回步骤 1
3. 从 M 中拿到 var 对应的 internal graph generator, 生成 internal graph
4. 从 internal graph 创建 JITExecutor
5. 写回原始的计算图 G
6. 转到步骤 1

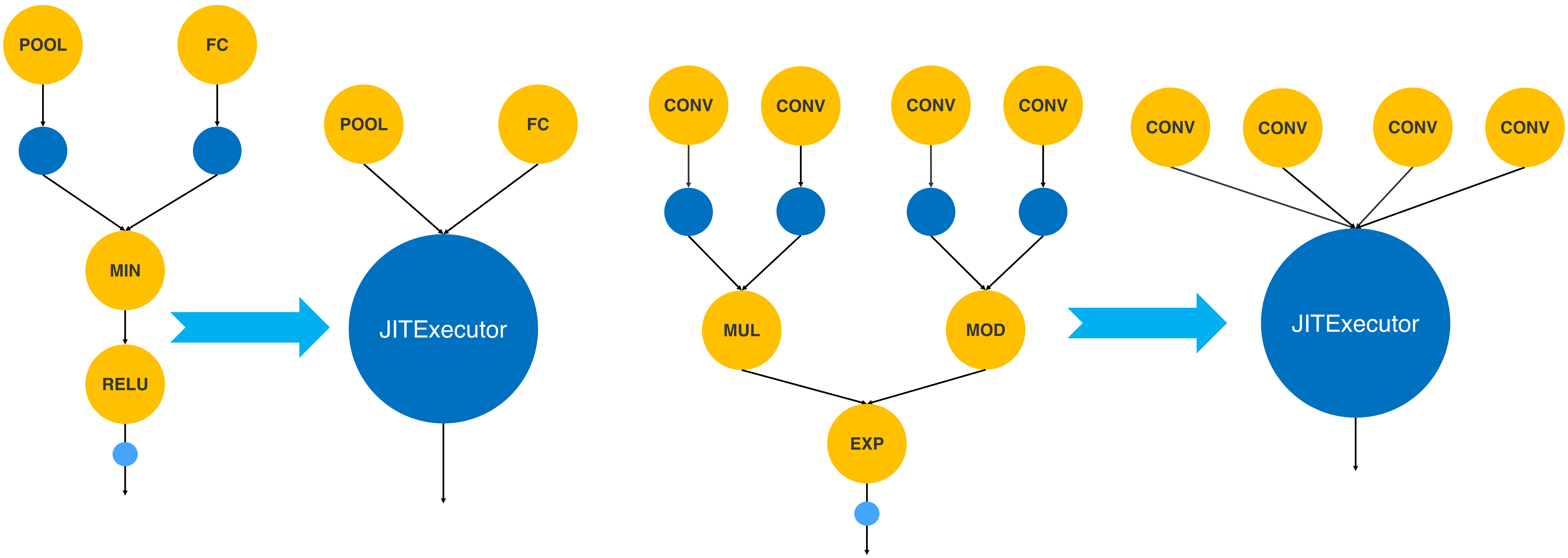
Fusion

internal graph generator -> internal graph



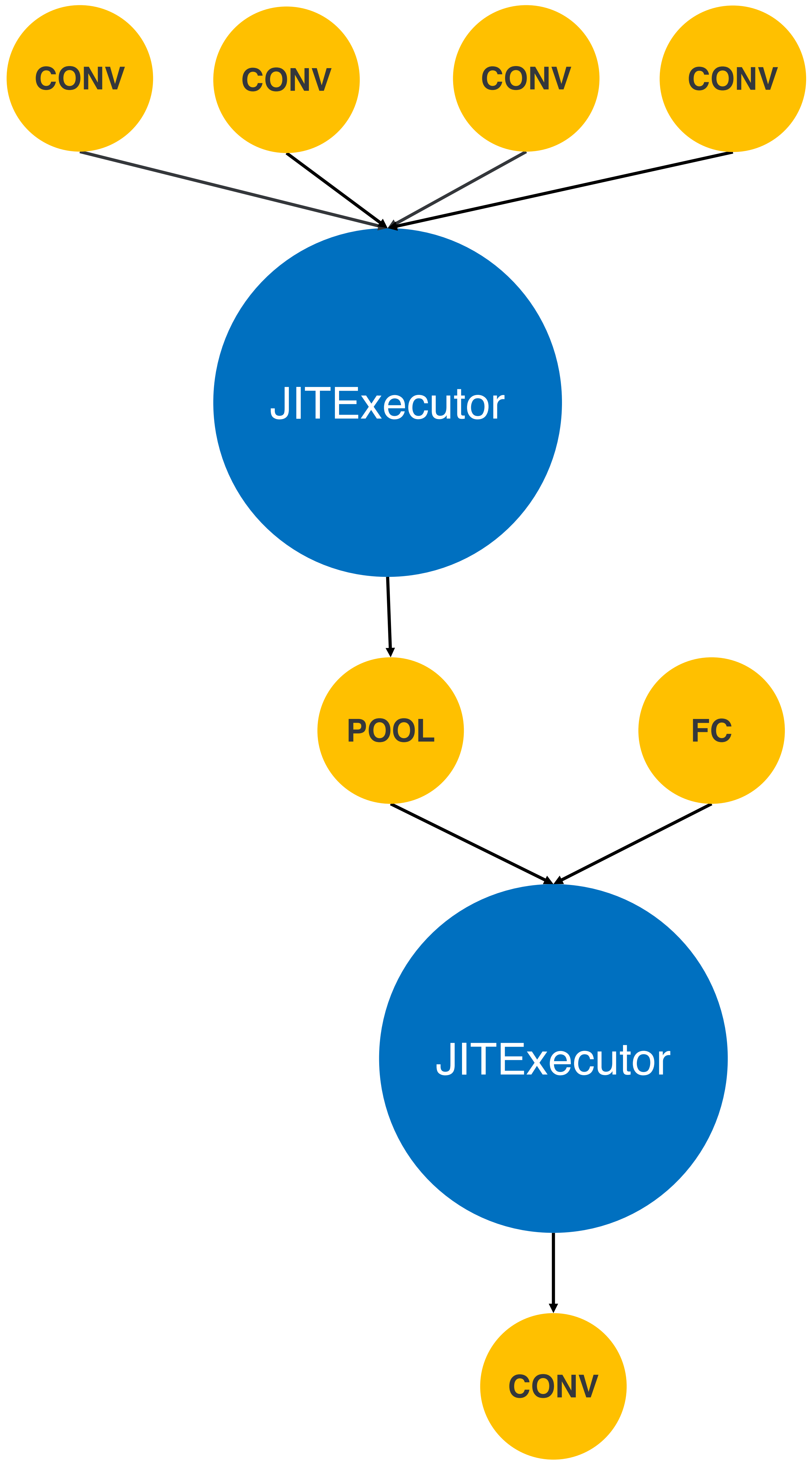
Fusion

internal graph -> JITExecutor



Fusion

写回计算图



OUTLINE

MEGVII 旷视

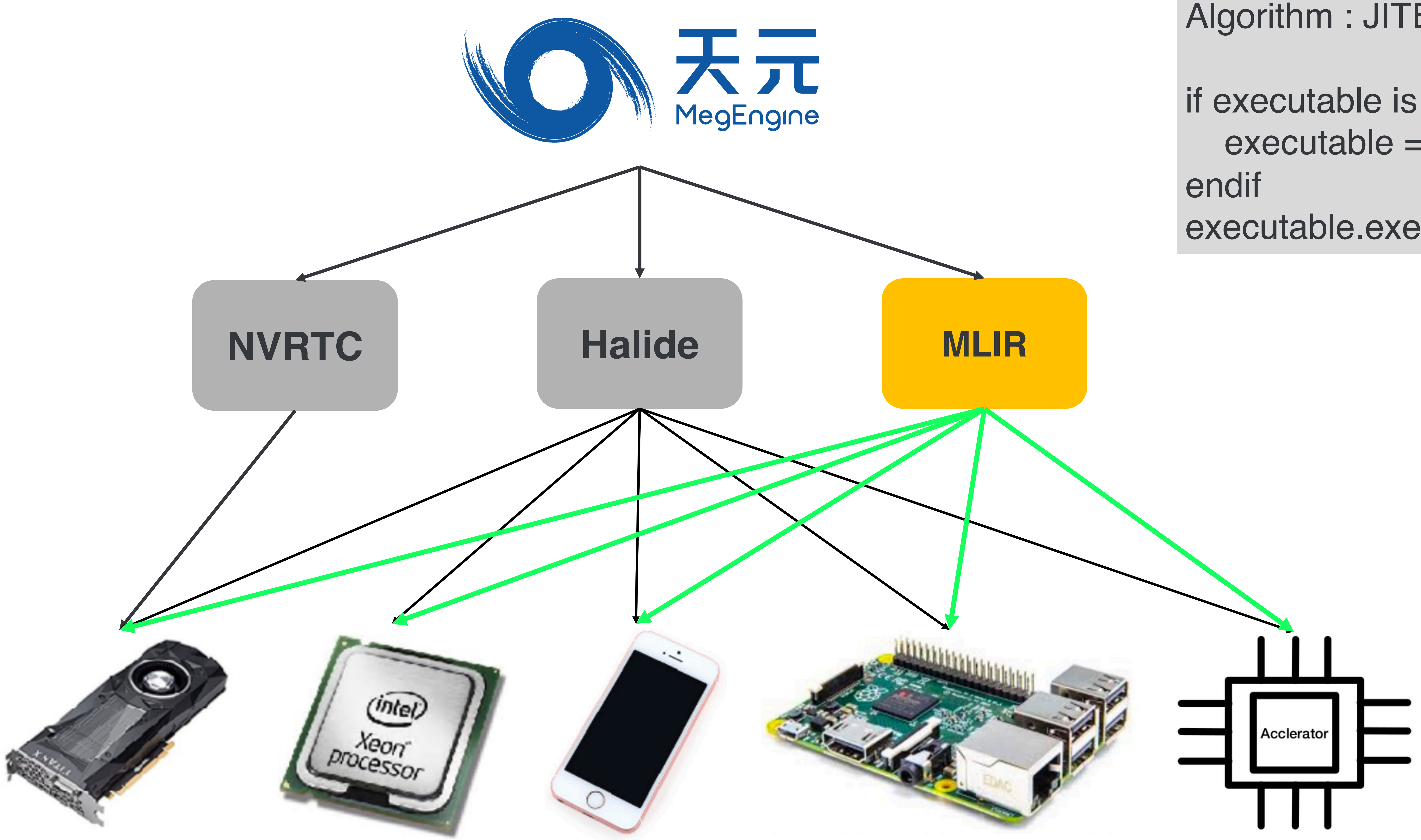
① Motivation

② Detect Fusion

③ JIT Compilation

④ Evaluation

JIT Compilation




```
Algorithm : JITExecutor::execute

if executable is not valid:
    executable = compiler.compile()
endif
executable.execute()
```

MLIR

MLIR(**M**ulti-**L**evel **I**ntermediate **R**epresentation)

1. Operator level

```
%C = "mgb.matmul"(%A, %B)
   : (memref<2048x2048xf32>, memref<2048x2048xf32>) ->
tensor<2048x2048xf32>
```

2. Loop level

```
affine.for %i = 0 to 2048 {
   affine.for %j = 0 to 2048 {
      affine.for %k = 0 to 2048 {
         %a = affine.load %A[%i, %k] : memref<2048x2048xf32>
         %b = affine. load %B[%k, %j] : memref<2048x2048xf32>
         %c = affine. load %C[%i, %j] : memref<2048x2048xf32>
         %p = mulf %a, %b : f32
         %co = addf %c, %p : f32
         affine. store %co, %C[%i, %j] : memref<2048x2048xf32>
      }
   }
}
```

3. Low-level : 类似汇编语言

```
%a = load %A[%i2, %i3] : memref<2048x2048xf32>
%b = load %B[%i2, %i3] : memref<2048x2048xf32>
%c = addf %a, %b : f32
store %c, %C[%i2, %i3] : memref<2048x2048xf32>
```

MLIR

MLIR 可以做：

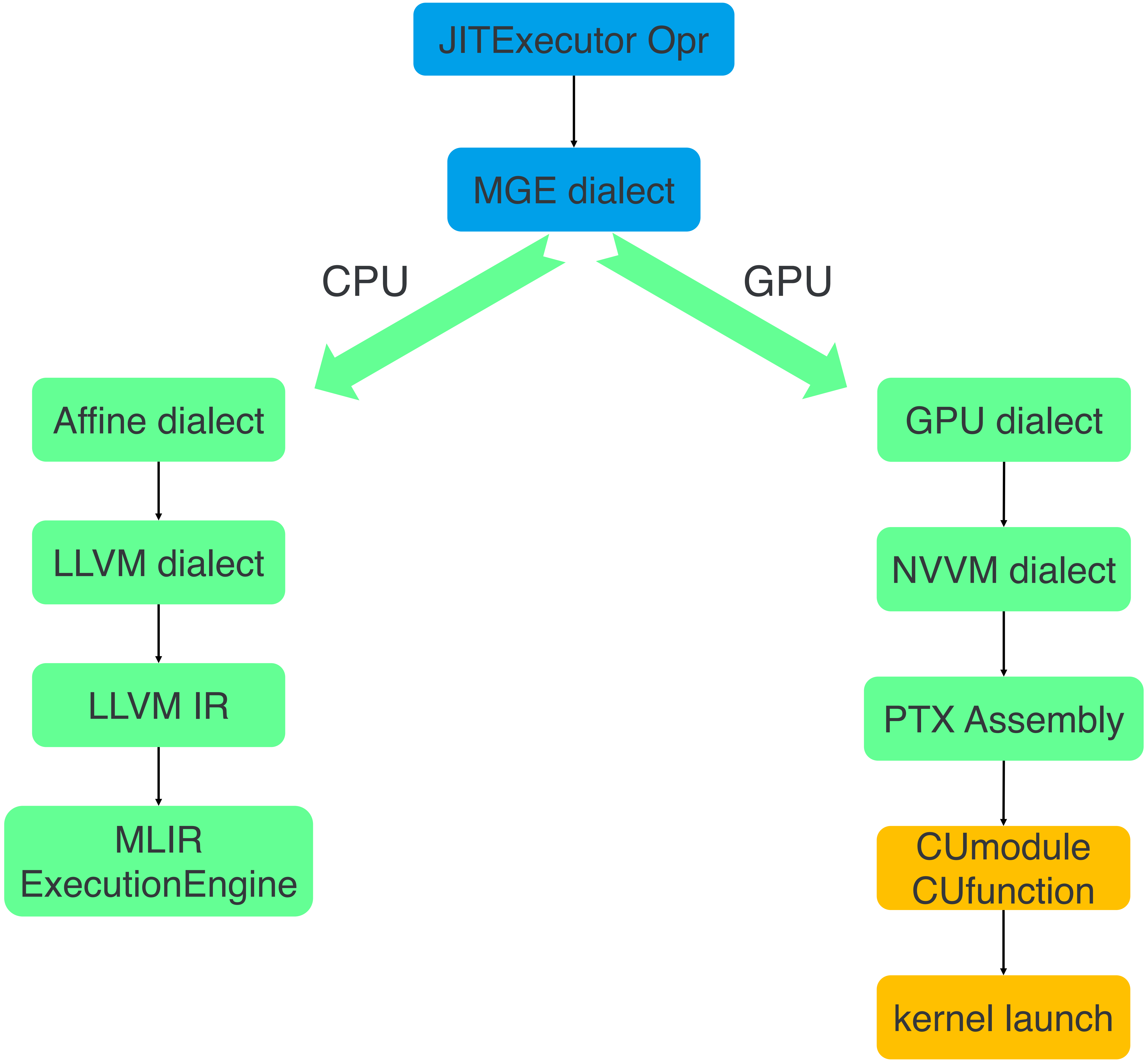
- 表达数据流图（如静态图模式下的 MegEngine Graph）
- 进行各种优化如算子融合（kernel fusion）、循环融合、分块和内存格式（memory layout）转换等
- 自动代码生成、自动向量化
- 表达适用于专用加速器的语言

Dialect : 一个算子、类型、属性的抽象集合（ MGE dialect, Affine dialect, GPU dialect, LLVM dialect）

Lowering : dialect 之间的转换，例如 MGE dialect -> Affine dialect

Lowering is all we need !

JIT Compilation



OUTLINE

MEGVII 旷视

① Motivation

② Detect Fusion

③ JIT Compilation

④ Evaluation

Evaluation

MEGVII 旷视

1. cmake 源码编译 MegEngine（仅 MLIR 需要）

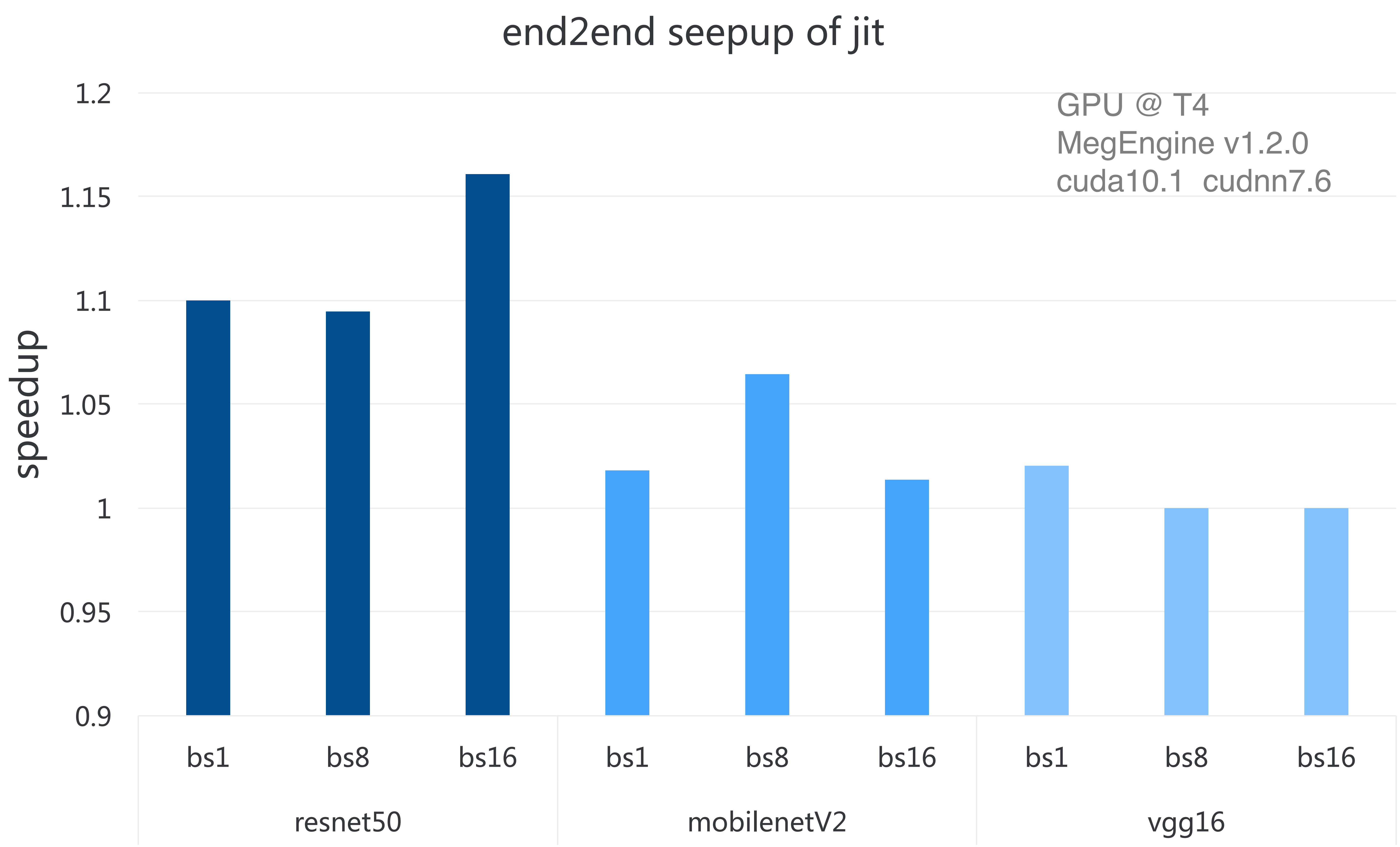
2. 改 python 代码

```
if __name__ == '__main__':
    gm = ad.GradManager().attach(model.parameters())
    opt = optim.SGD(model.parameters(), lr=0.0125, momentum=0.9, weight_decay=1e-4,)
    # 通过 trace 转换为静态图
    @trace(symbolic=True, opt_level=3)
    def train():
        with gm:
            logits = model(image)
            loss = F.loss.cross_entropy(logits, label)
            gm.backward(loss)
        opt.step()
        opt.clear_grad()
        return loss
    loss = train()
    loss.numpy()
```

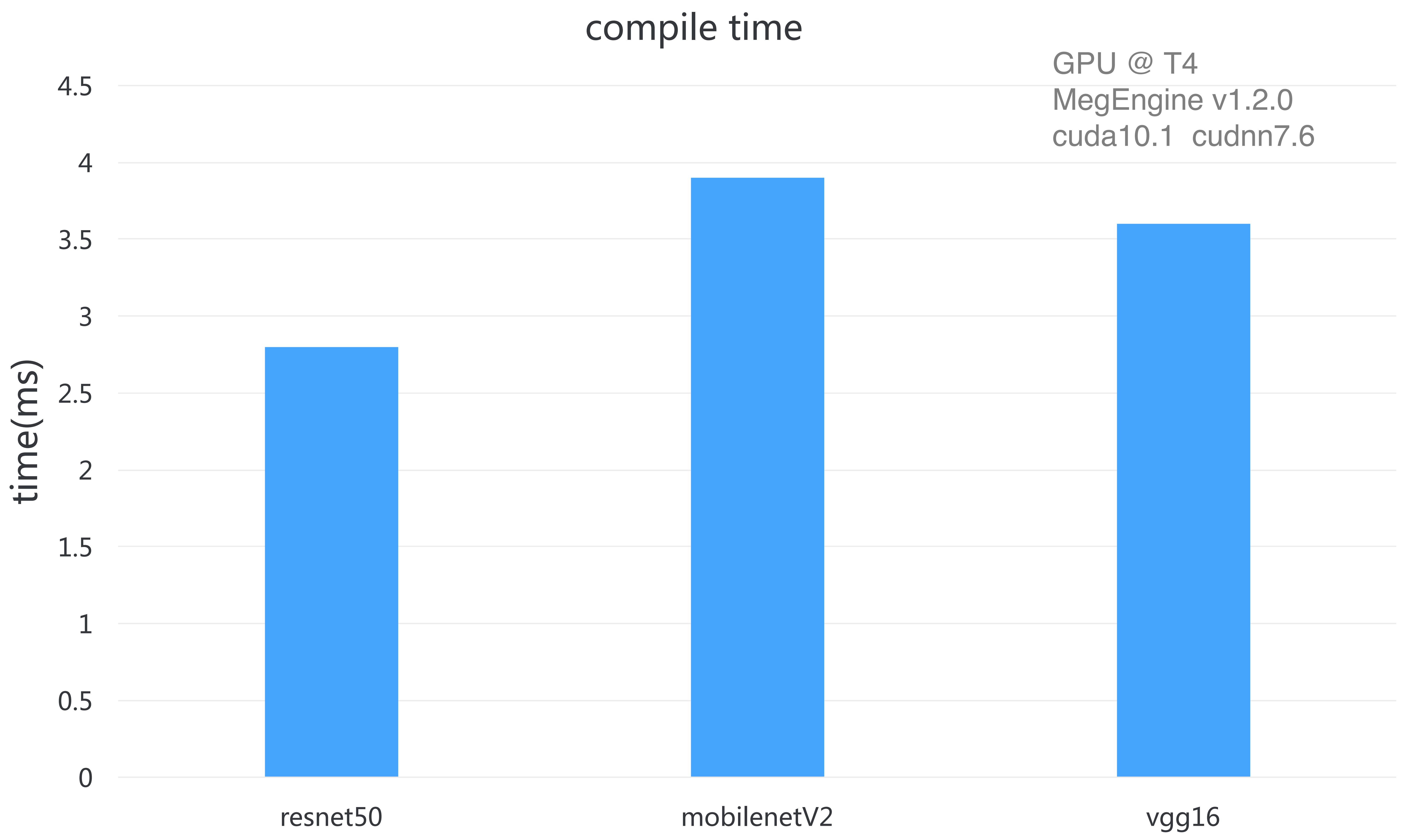
3. export MGB_JIT_BACKEND= "MLIR" / "NVRTC" / "HALIDE"

https://megengine.org.cn/doc/advanced/how_to_use_codegen.html

Evaluation


end2end seepup of jit
GPU @ T4
MegEngine v1.2.0
cuda10.1 cudnn7.6
speedup
1.2
1.15
1.1
1.05
1
0.95
0.9
bs1
bs8
bs16
bs1
bs8
bs16
bs1
bs8
bs16
resnet50
mobilenetV2
vgg16

Evaluation



compile time
GPU @ T4
MegEngine v1.2.0
cuda10.1 cudnn7.6
time(ms)
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
resnet50
mobilenetV2
vgg16

Future work

MEGVII 旷视

- 将 jit 编译的结果离线保存，解决线上首次运行编译慢的问题
- jit 支持更多的算子
- jit 支持更多数据类型
- 动态图 jit（真jit），即检测热点代码并编译

MEGVII 旷视

天元 MegEngine

天元

天元开发者交流群

群号：1029741705

扫一扫二维码，加入群聊。