神经网络图优化与量化

6.2.1 啥是计算图

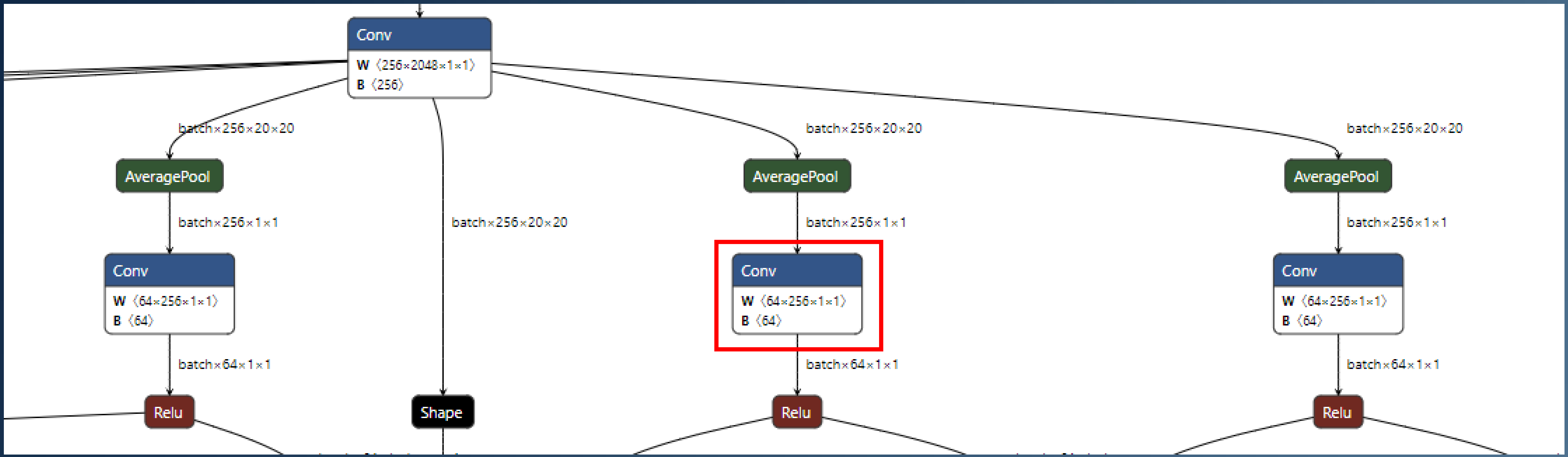
What is a Computational Graph.

C = {N, E, I, O} 一个计算图可以表示为一个由节点、边集、输入边、输出边组成的四元组。

- 计算图是一个有向联通无环图，其中节点也被称作为算子。
- 算子必定有边相连，输入边，输出边不为空。
- 计算图中可以有重边。

6.2.1 算子

What is a Computational Graph.



6.2.1 算子

What is a Computational Graph.

Operator	Since version
Abs	13, 6, 1
Acos	7
Acosh	9
Add	14, 13, 7, 6, 1
And	7, 1
ArgMax	13, 12, 11, 1
ArgMin	13, 12, 11, 1
Asin	7
Asinh	9
Atan	7
Atanh	9
AveragePool	11, 10, 7, 1
BatchNormalization	15, 14, 9, 7, 6, 1
Constant	13, 12, 11, 9, 1
ConstantOfShape	9
Conv	11, 1
ConvInteger	10
ConvTranspose	11, 1
Cos	7
Cosh	9
CumSum	14, 11
DepthToSpace	13, 11, 1
DequantizeLinear	13, 10
Det	11
Div	14, 13, 7, 6, 1
Dropout	13, 12, 10, 7, 6, 1

https://github.com/onnx/onnx/blob/main/docs/Operators.md

6.2.1 算子

What is a Computational Graph.

算子是神经网络的最小调度单位，但很遗憾的是，它并不是原子的：一个复杂的算子可以被更细粒度的算子所表示：

我们总是以算子为单位去支持你的网络。

在你的网络中你应该尽量避免使用特殊算子。

6.2.2 算子融合加速

Graph Fusion.

```
__declspec(noinline) void MatMul(
        ELEMENT_TYPE** input, ELEMENT_TYPE** weight,
        ELEMENT_TYPE** output, const unsigned int num_of_elements) {
        for (unsigned int i = 0; i < num_of_elements; i++)
                for (unsigned int j = 0; j < num_of_elements; j++)
                        for (unsigned int k = 0; k < num_of_elements; k++)
                                output[i][j] += input[i][k] * weight[k][j];
}

__declspec(noinline) void BiasAdd(
        ELEMENT_TYPE** input, ELEMENT_TYPE* bias,
        ELEMENT_TYPE** output, const unsigned int num_of_elements) {
        for (unsigned int i = 0; i < num_of_elements; i++)
                for (unsigned int j = 0; j < num_of_elements; j++)
                        output[i][j] += bias[i];
}
```

6.2.2 算子融合加速

Graph Fusion.

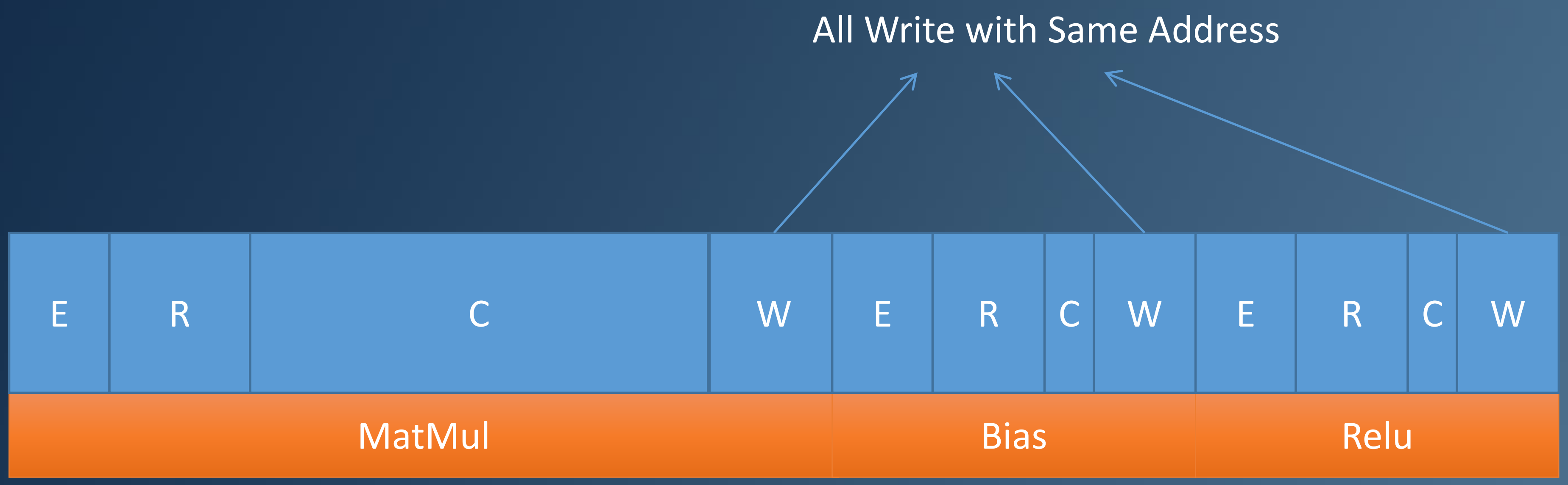
```
__declspec(noinline) void Relu(
        ELEMENT_TYPE** input, ELEMENT_TYPE** output,
        const unsigned int num_of_elements) {
        for (unsigned int i = 0; i < num_of_elements; i++)
                for (unsigned int j = 0; j < num_of_elements; j++)
                        output[i][j] = input[i][j] * (input[i][j] > 0);
}
```

在Matmul + Bias + Relu的子网中，如果不融合算子，output将至少被写入3次

并且启动3个算子的速度也不是很快。

6.2.2 算子融合加速

Graph Fusion.



E: Task Emission

R: Read

C: Computing

W: Write

6.2.2 算子融合加速

Graph Fusion.

```
__declspec(noinline) void MatMul(
        ELEMENT_TYPE** input, ELEMENT_TYPE** weight,  ELEMENT_TYPE* bias,
        ELEMENT_TYPE** output, const unsigned int num_of_elements) {
        for (unsigned int i = 0; i < num_of_elements; i++){
                for (unsigned int j = 0; j < num_of_elements; j++){
                        int accumulator = 0;
                        for (unsigned int k = 0; k < num_of_elements; k++){
                                accumulator  += input[i][k] * weight[k][j];
                        }
                        output[i][j] = accumulator + bias[j] > 0 ? accumulator + bias[j]: 0;
                }
        }
}
```

6.2.2 算子融合加速

Graph Fusion.

图融合减少访存，以及算子overhead

6.2.3 常见计算图优化

Widely - used Graph Optimization

6.2.3.1 激活函数融合

6.2.3.2 移除 Batchnorm 与 Dropout

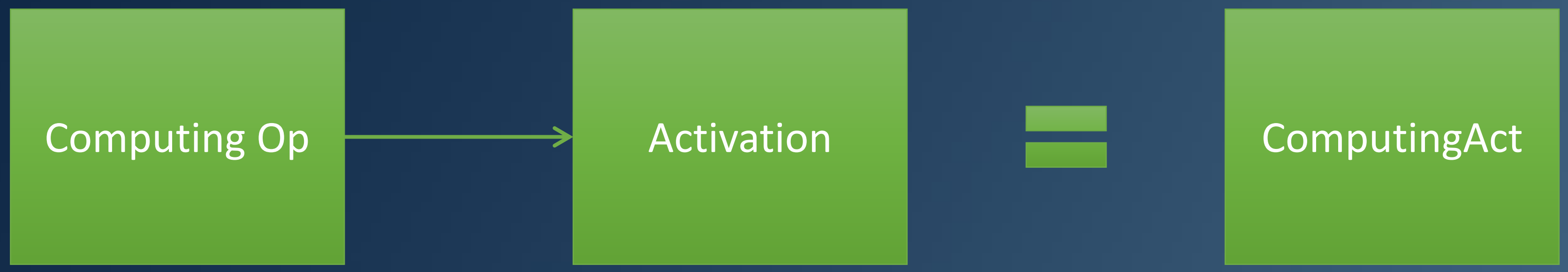
6.2.3.3 常量折叠

6.2.3.4 矩阵乘融合

6.2.3.5 Conv - Add 融合

6.2.3 常见计算图优化

Widely - used Graph Optimization



常见计算算子：Conv，ConvTranpose，Gemm

常见激活函数：Relu，Clip(Relu6)，PRelu，Tanh，Sigmoid，Swish

```
output[i][j] = activation_fn(accumulator + bias[j]);
```

6.2.3 常见计算图优化

Widely - used Graph Optimization

6.2.3 常见计算图优化

Widely - used Graph Optimization

计算算子：$Y = WX + B$

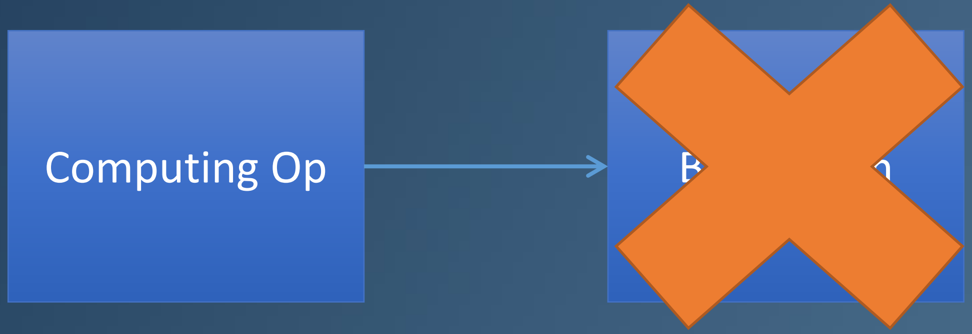
Batchnorm：$Y' = gamma * \frac{Y - mean}{var} + beta$

Merged：$Y' = gamma * \frac{WX + B - mean}{var} + beta = \frac{gamma}{var} WX + \frac{gamma}{var}(B - mean) + beta$

6.2.3 常见计算图优化

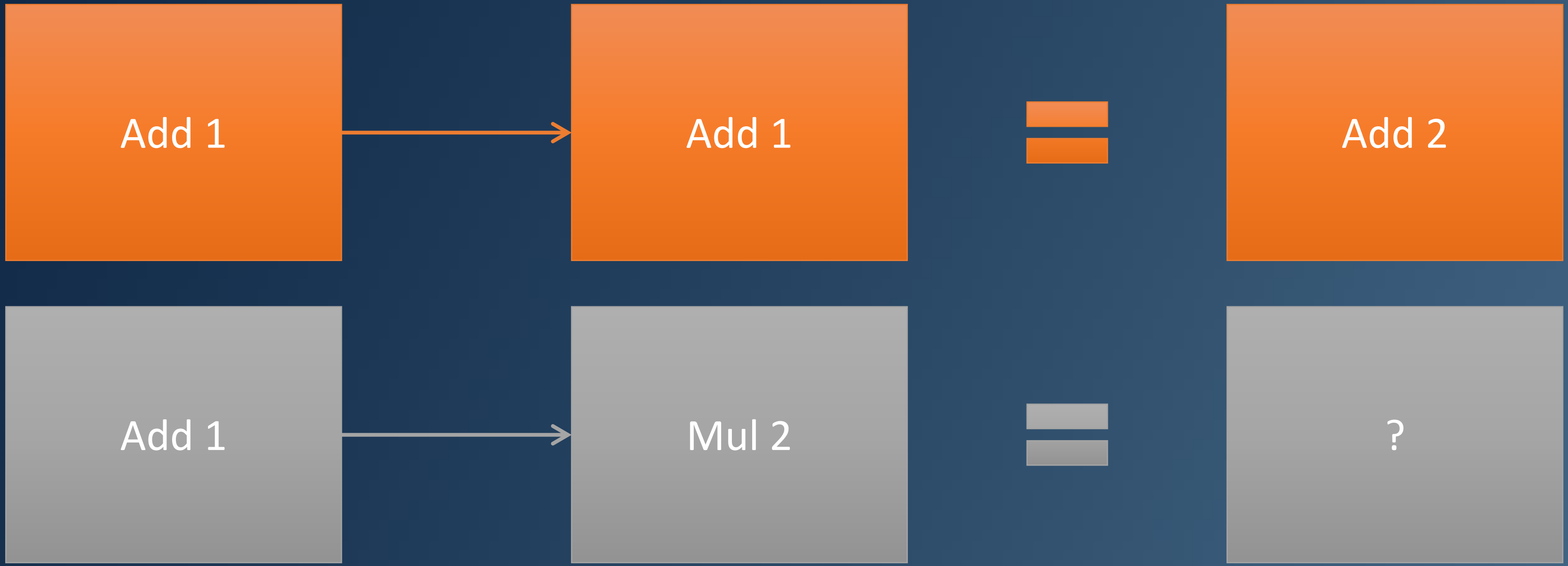
Widely - used Graph Optimization

```
alpha  = bn_op.parameters[0].value
beta    = bn_op.parameters[1].value
mean  = bn_op.parameters[2].value
var      = bn_op.parameters[3].value
if computing_op.type == 'Conv':
        # calculate new weight and bias
        scale = alpha / np.sqrt(var + epsilon)
        w = w * scale.reshape([-1, 1, 1, 1])
        b = alpha * (b - mean) / np.sqrt(var + epsilon) + beta
elif computing_op.type == 'Gemm':
        # calculate new weight and bias
        scale = alpha / np.sqrt(var + epsilon)
        if computing_op.attributes.get('transB', 0): w = w * scale.reshape([-1, 1])
        else: w = w * scale.reshape([1, -1])
        b = alpha * (b - mean) / np.sqrt(var + epsilon) + beta
```



6.2.3 常见计算图优化

Widely - used Graph Optimization



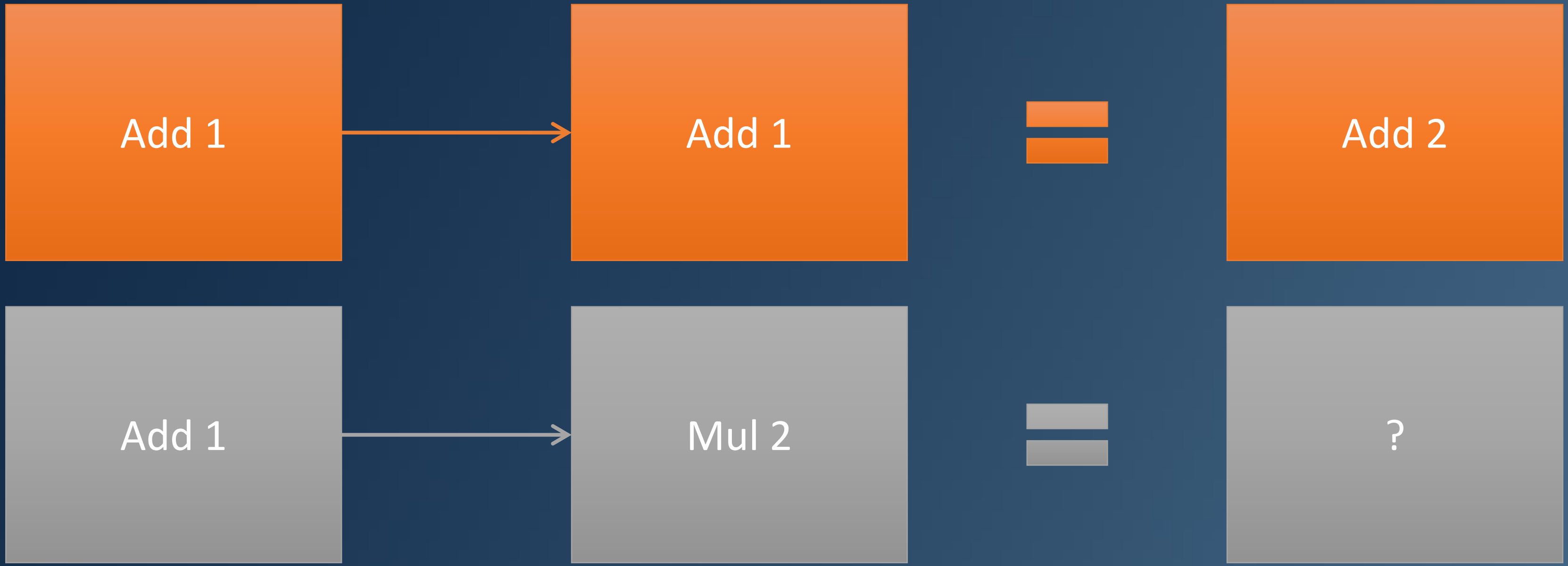
Add1 ：$Y = X + 1$

Add2 ：$Y' = Y + 1$

Merged ：$Y' = X + 2$

6.2.3 常见计算图优化

Widely - used Graph Optimization

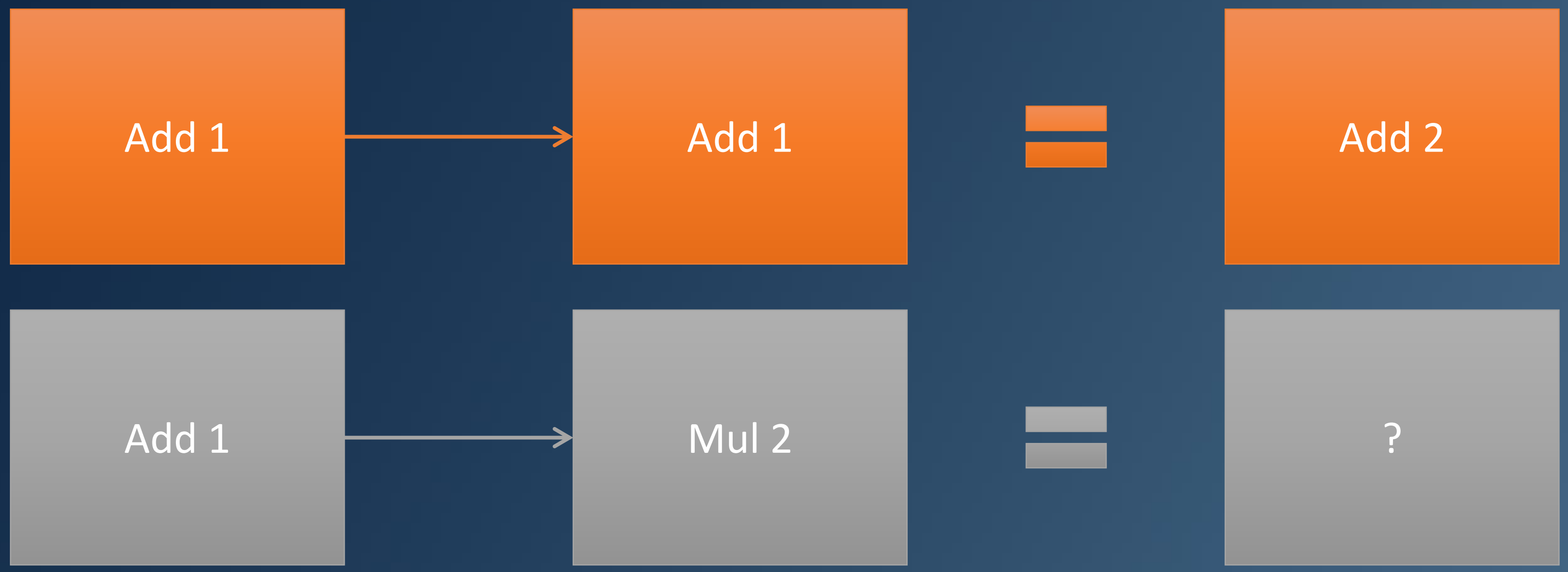


Add1 ：$Y = X + 1$ Mul2 ：$Y' = Y * 2$

Merged ：$Y' = (X + 1) * 2 = X * 2 + 2$

6.2.3 常见计算图优化

Widely - used Graph Optimization



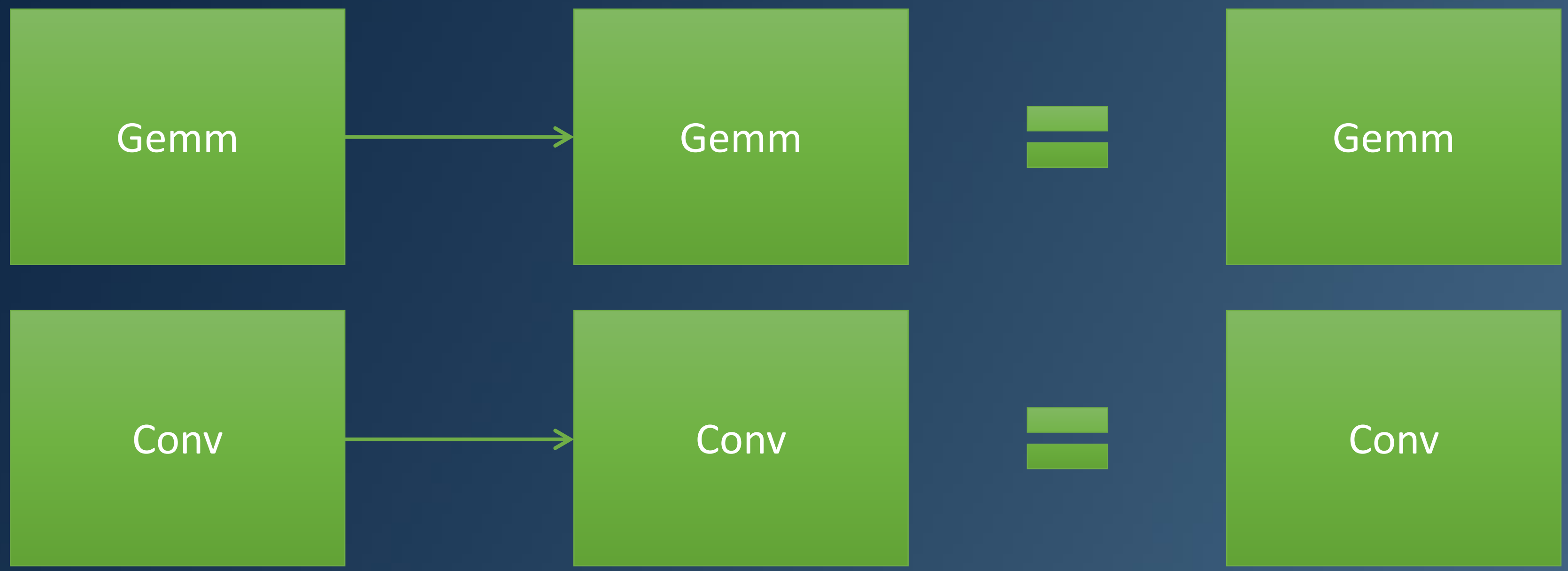
Add1 ： $Y = X + 1$　　　　Mul2 ： $Y' = Y * 2$

Merged ： $Y' = (X + 1) * 2 = X * 2 + 2?$

EMM… WE DO NOT HAVE AN OP TO RUN X*2+2!

6.2.3 常见计算图优化

Widely - used Graph Optimization



计算算子1：$Y = W_1X + B_1$

计算算子2：$Y' = W_2Y + B_2$

融合后：$Y' = W_2(W_1X + B_1) + B_2$

6.2.3 常见计算图优化

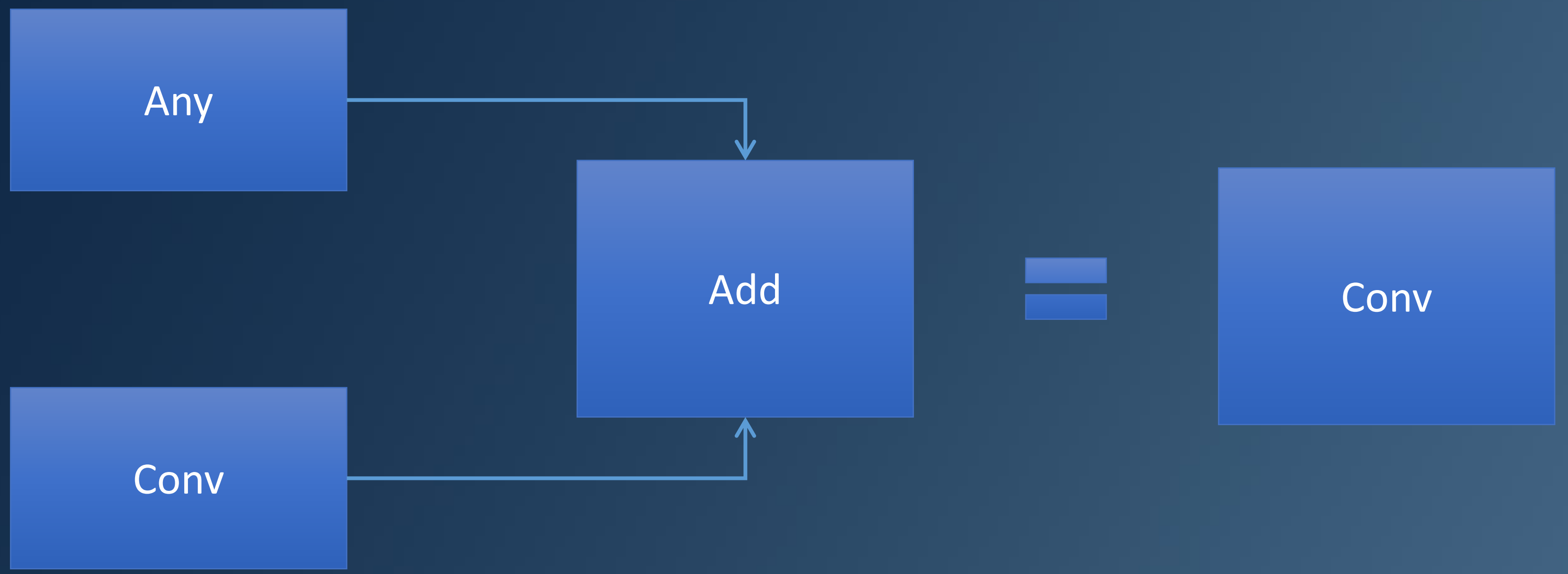
Widely - used Graph Optimization

```
def svd_for_factorization(self, w: torch.Tensor):
    assert w.ndim == 2
    u, s, v = torch.svd(w)
    a = torch.matmul(u, torch.diag(torch.sqrt(s)))
    b = torch.matmul(torch.diag(torch.sqrt(s)), v.transpose(0, 1))
    print(a.max(), b.max(), w.max())
    return a, b

if operation.type == 'Gemm':
        w = operation.parameters[0].value
        w = w.transpose(0, 1)
        if self.method == 'svd':
            a, b = self.svd_for_factorization(w)
```

6.2.3 常见计算图优化

Widely - used Graph Optimization



Conv：$Y_1 = W_1X_1 + B_1$

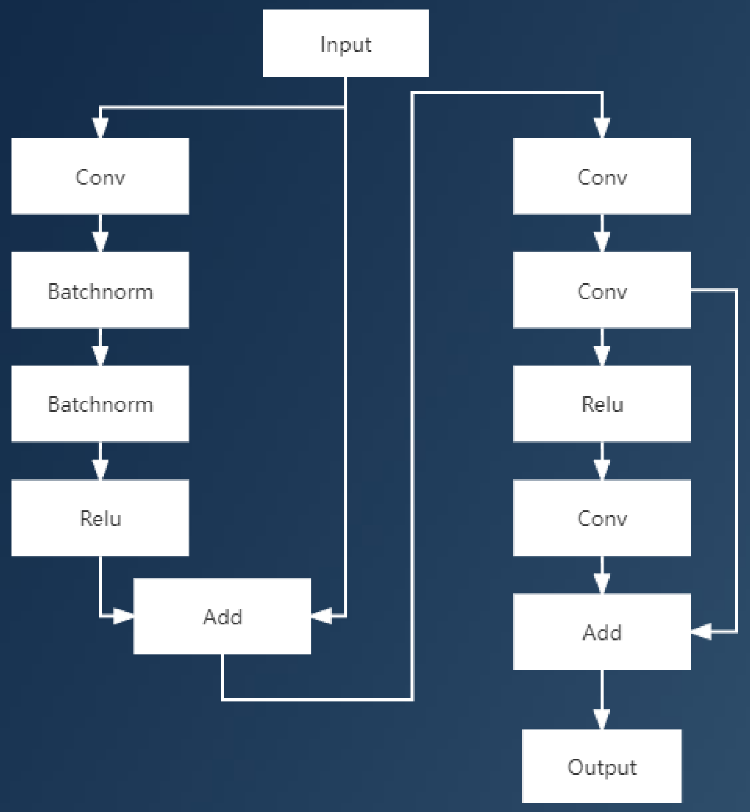
Any：Y_2

$Y = Y_1 + Y_2$

融合后：$Y = W_1X_1 + (Y_2 + B_1)$

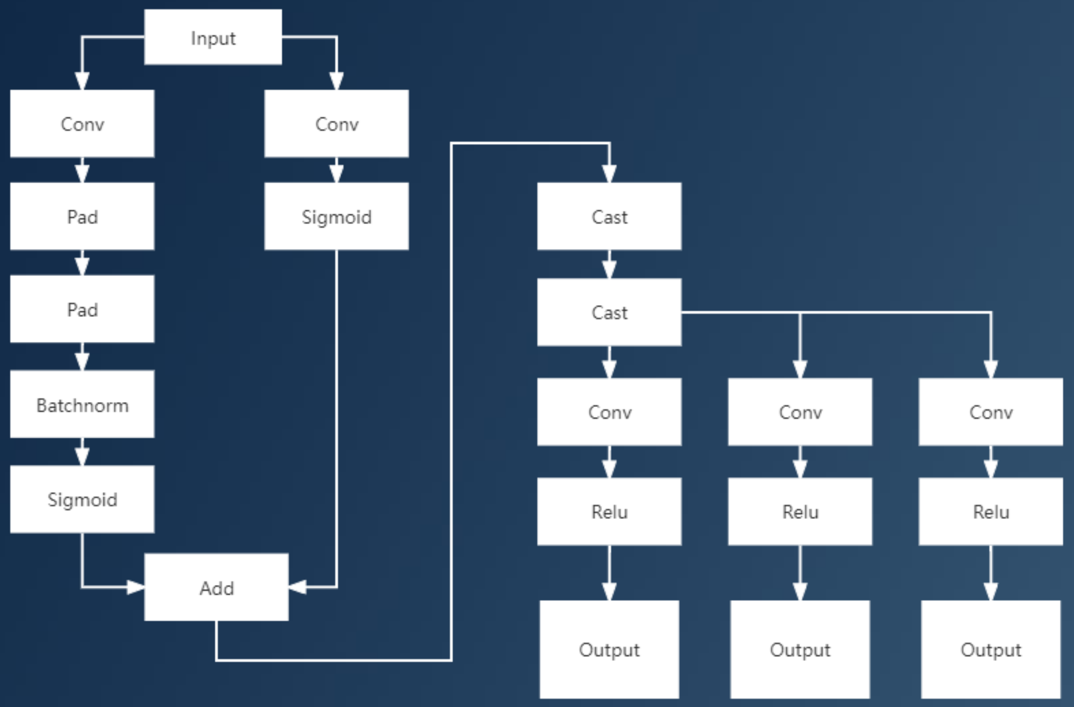
6.2.3 常见计算图优化

Widely - used Graph Optimization



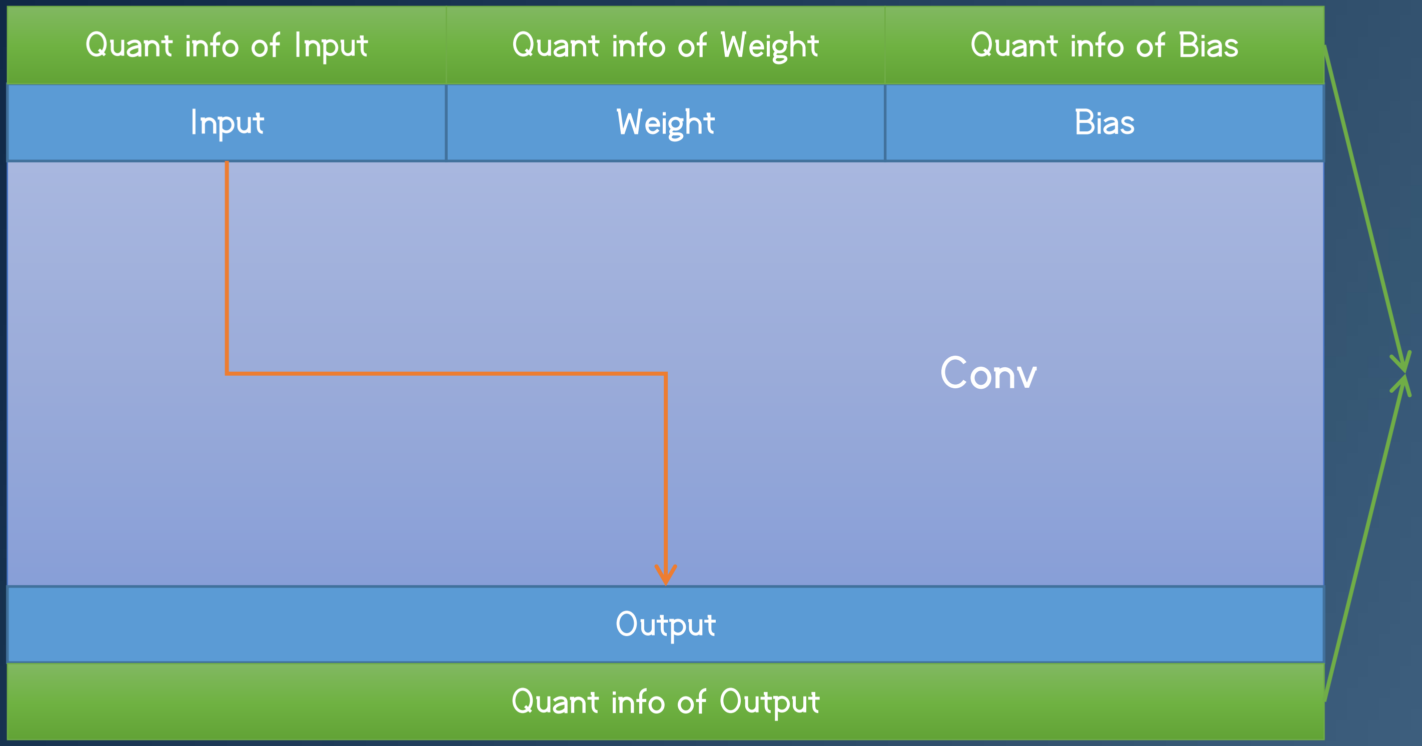
6.2.3 常见计算图优化

Widely-used Graph Optimization



6.2.4 联合定点

Union - Quantize



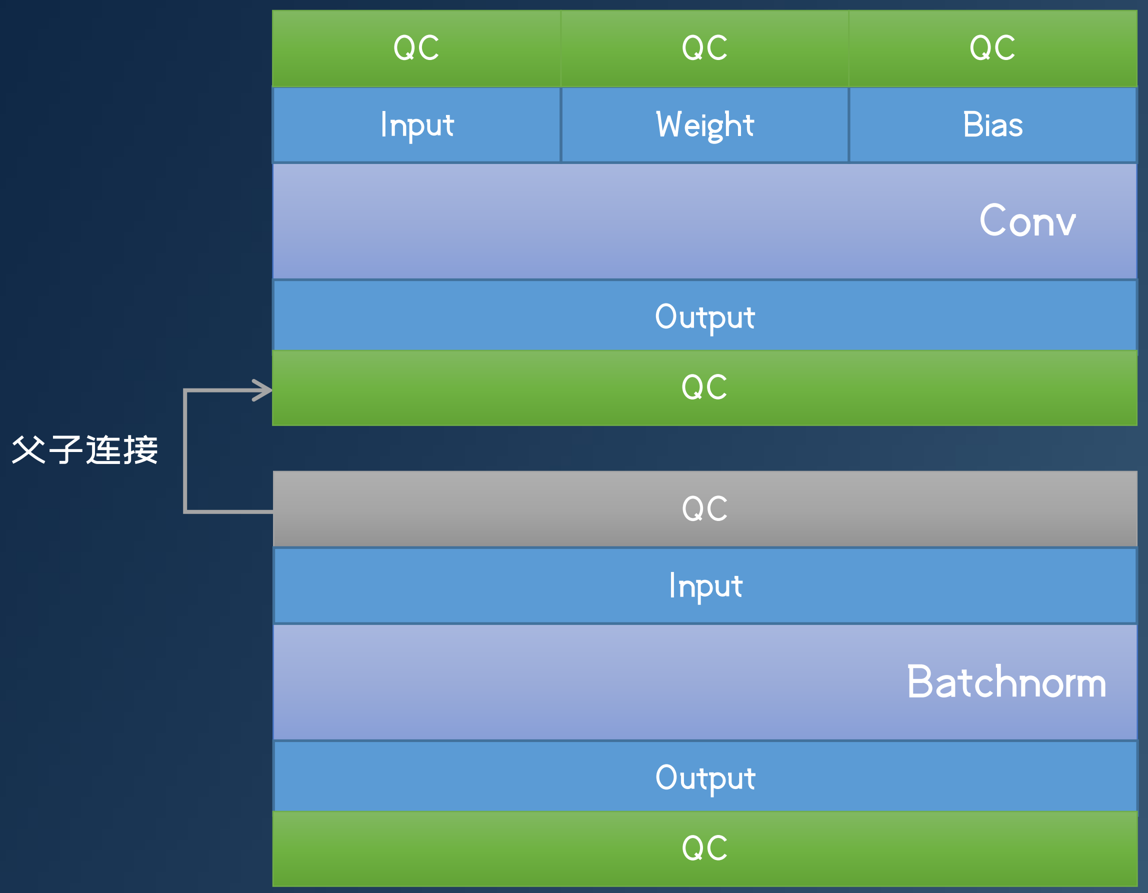

```
class TensorQuantizationConfig(Serializable):
        self._policy = policy
        self._num_of_bits = num_of_bits
        self._scale = scale
        self._offset = offset
        self.state = state
        self._rounding = rounding
        self._quant_min = quant_min
        self._quant_max = quant_max
        self._father_config = self # union - find
```

6.2.4 联合定点

Union - Quantize

6.2.4 联合定点

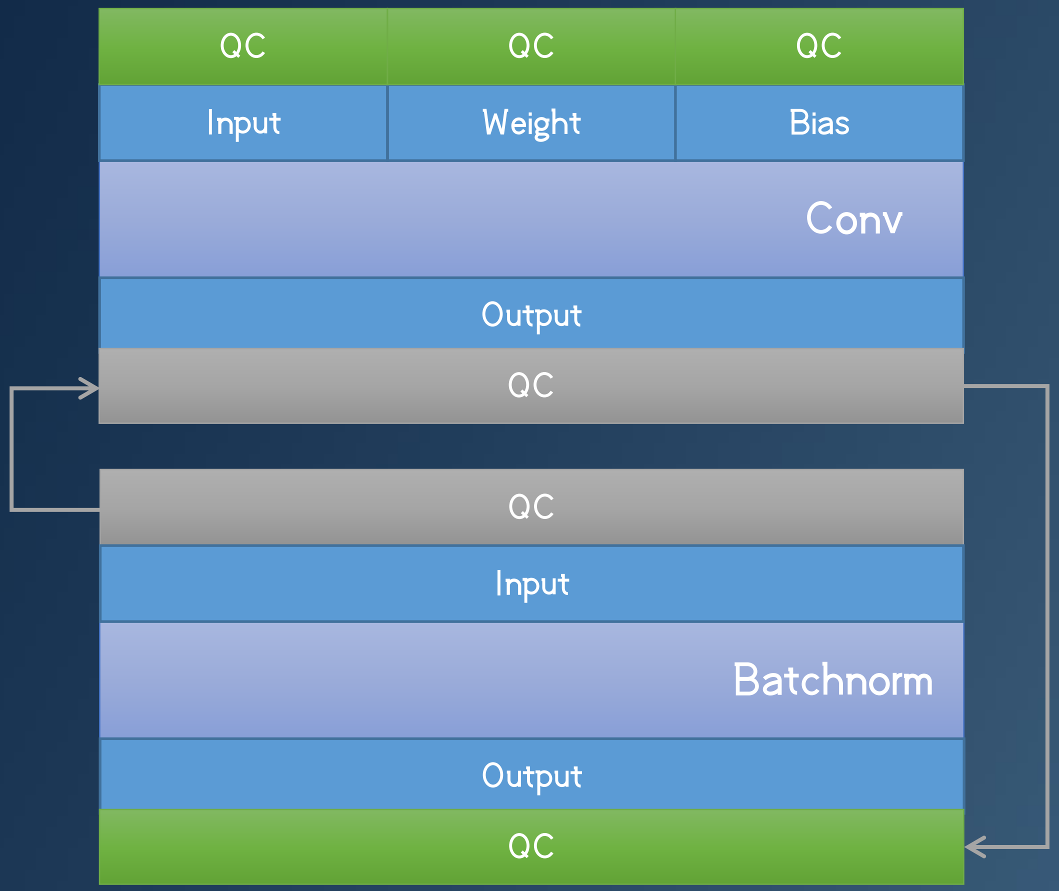
Union - Quantize



输入已经被量化，定点信息停用

6.2.4 联合定点

Union - Quantize



考虑到Conv与Batchnorm图融合
输出定点被停用

级联父子连接（并查集）

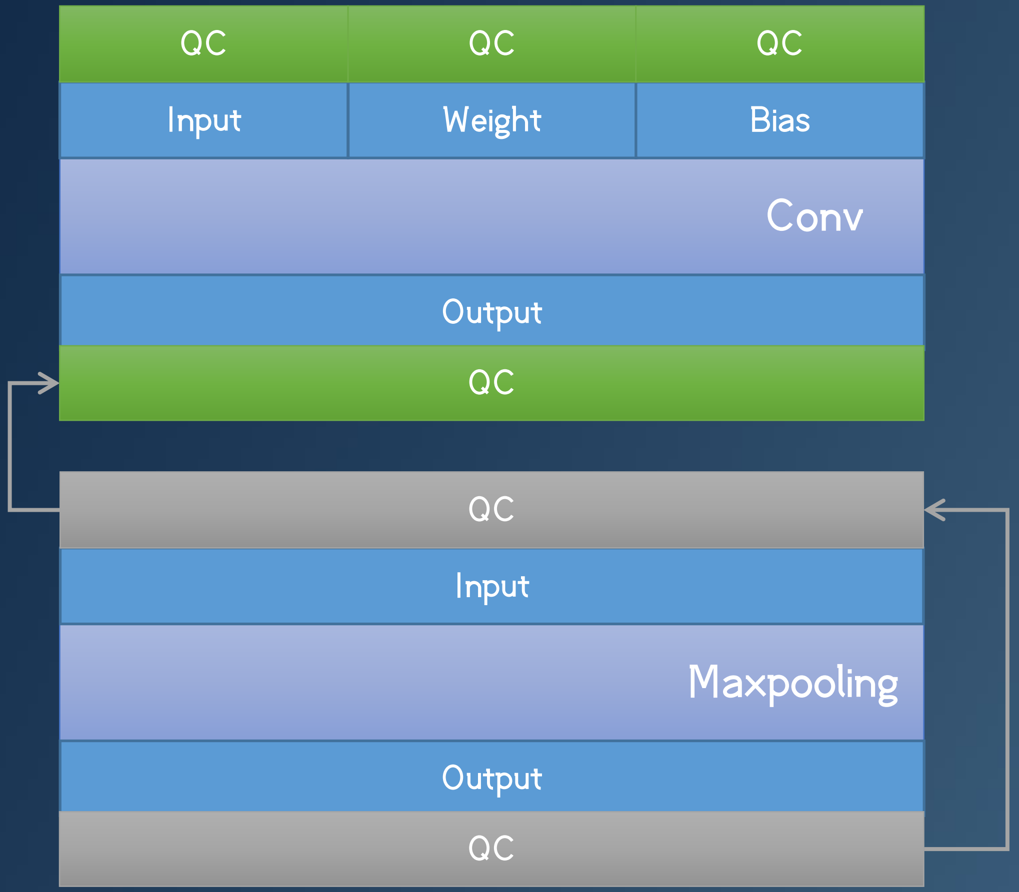
6.2.4 联合定点

Union - Quantize

类似地，激活函数联合定点

6.2.4 联合定点

Union - Quantize

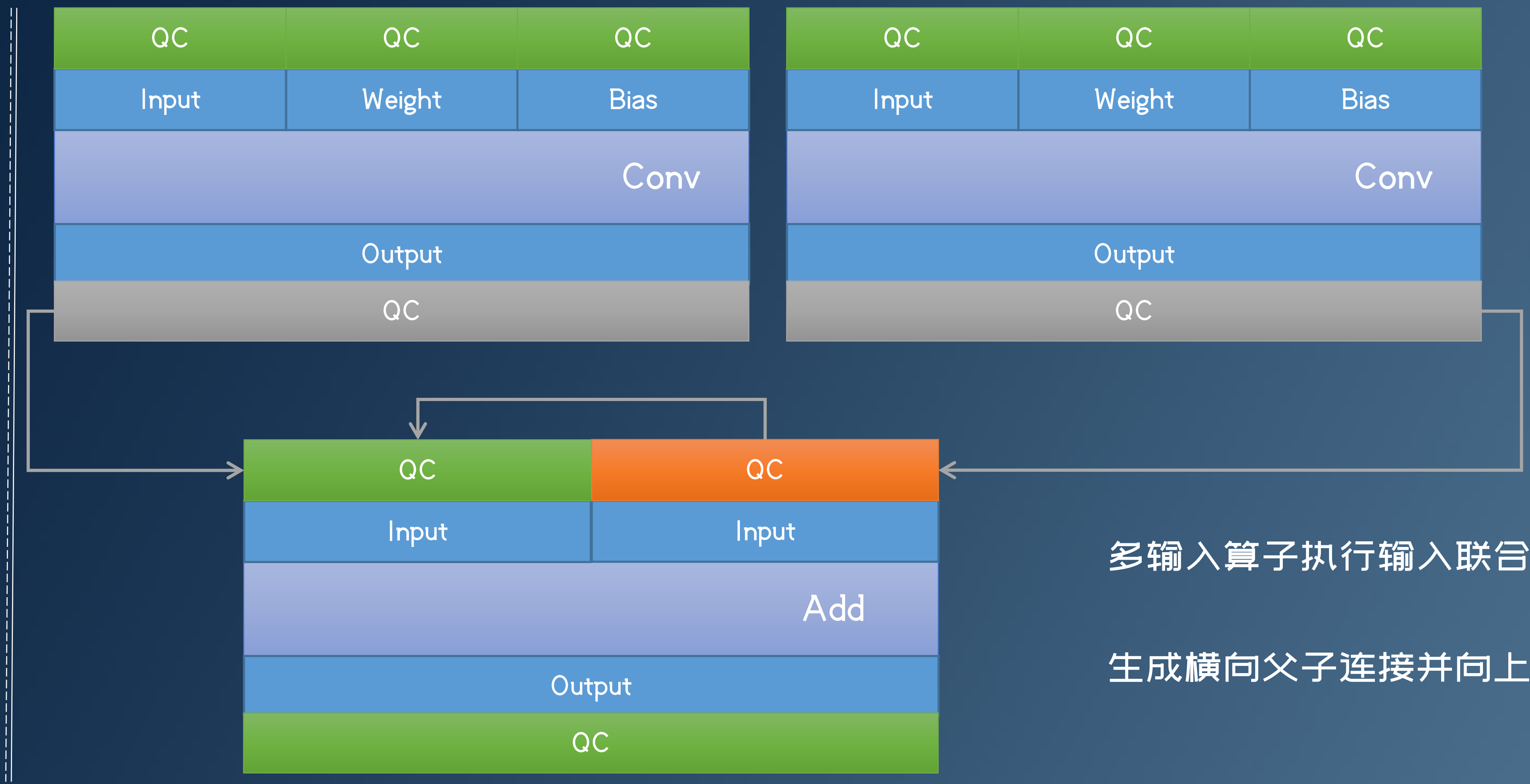


被动算子量化定点信息全部停用

向上生成级联父子连接

6.2.4 联合定点

Union - Quantize

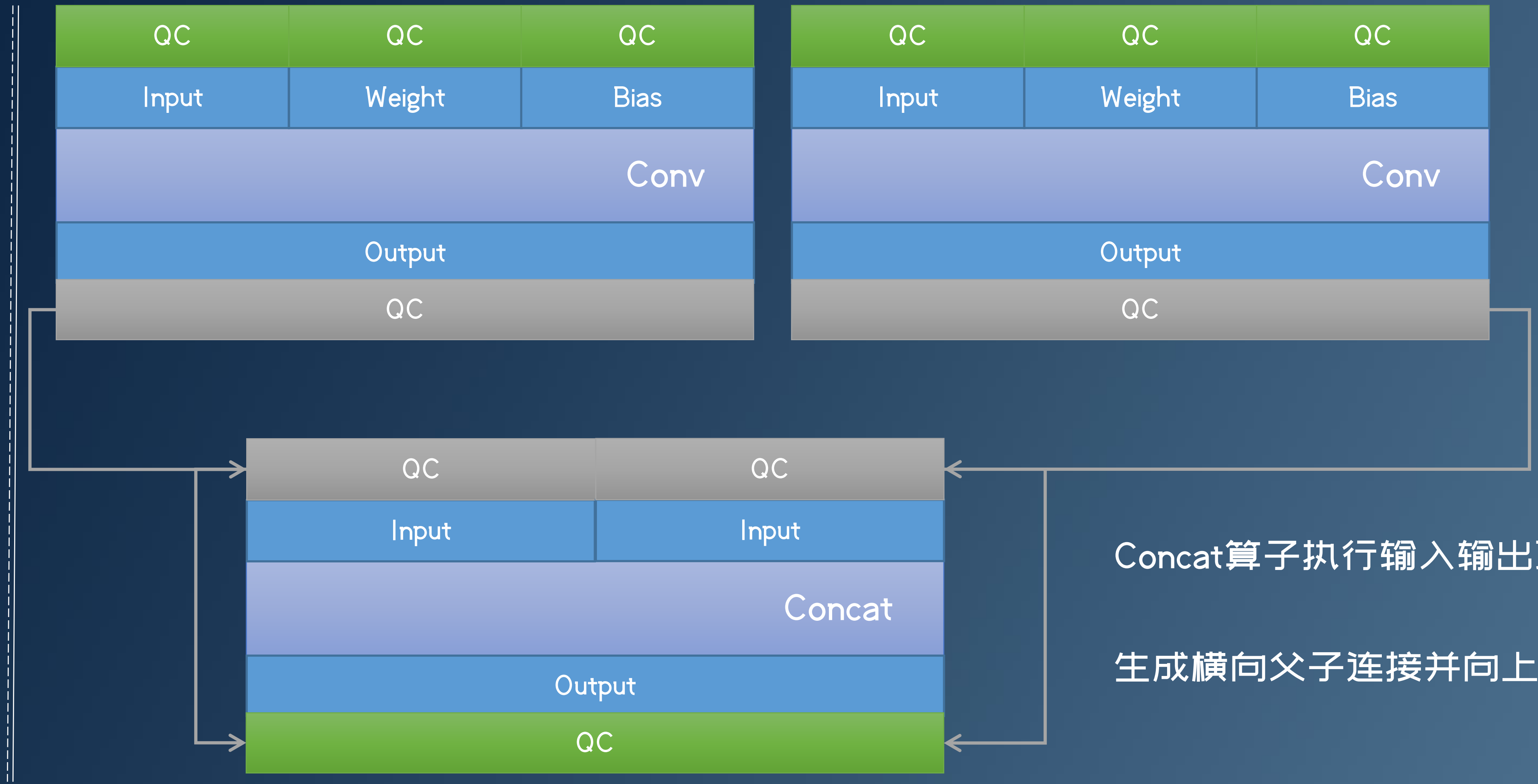


多输入算子执行输入联合定点

生成横向父子连接并向上覆盖（如可能）

6.2.4 联合定点

Union - Quantize

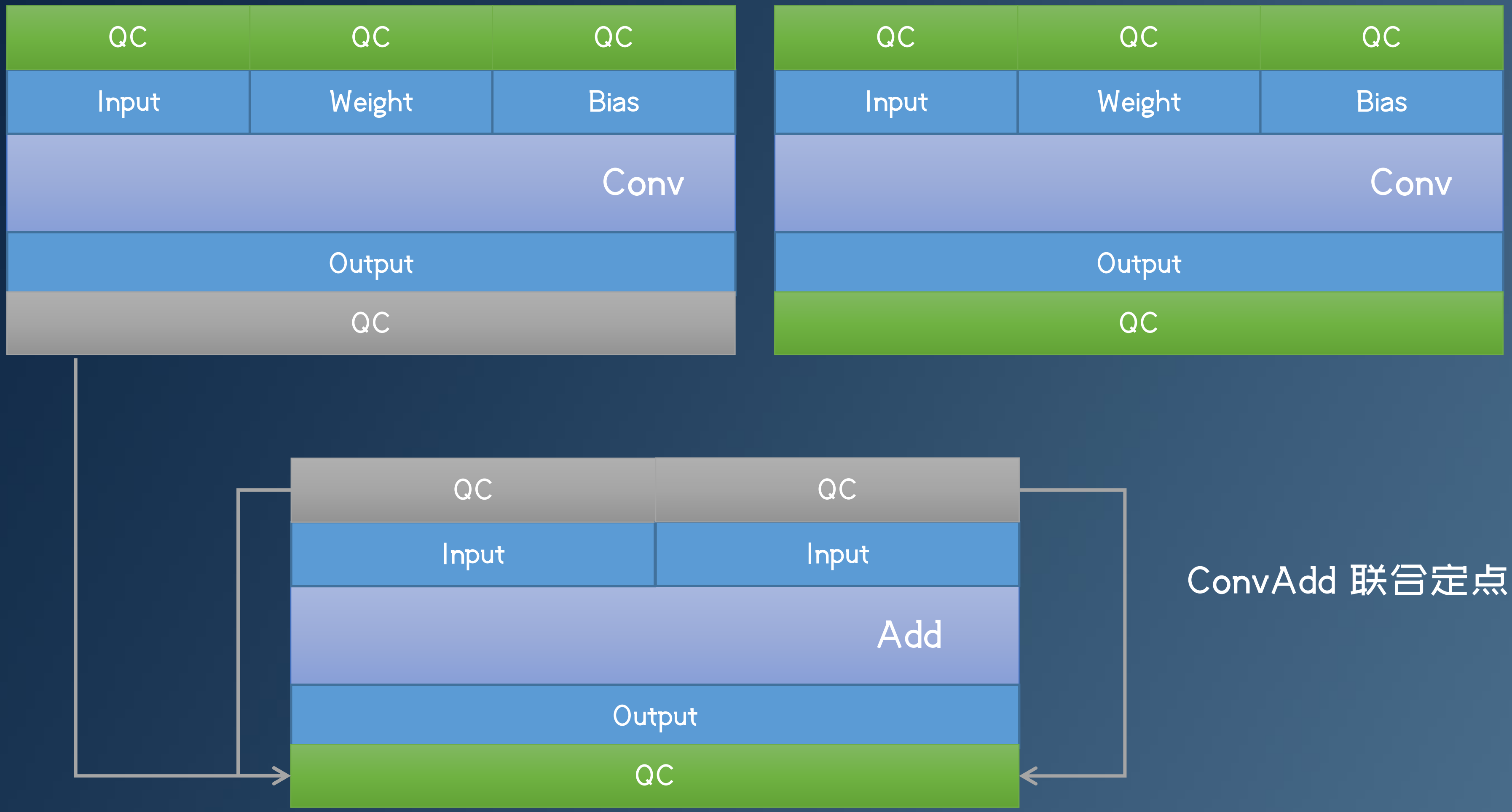


Concat算子执行输入输出联合定点

生成横向父子连接并向上覆盖（如可能）

6.2.4 联合定点

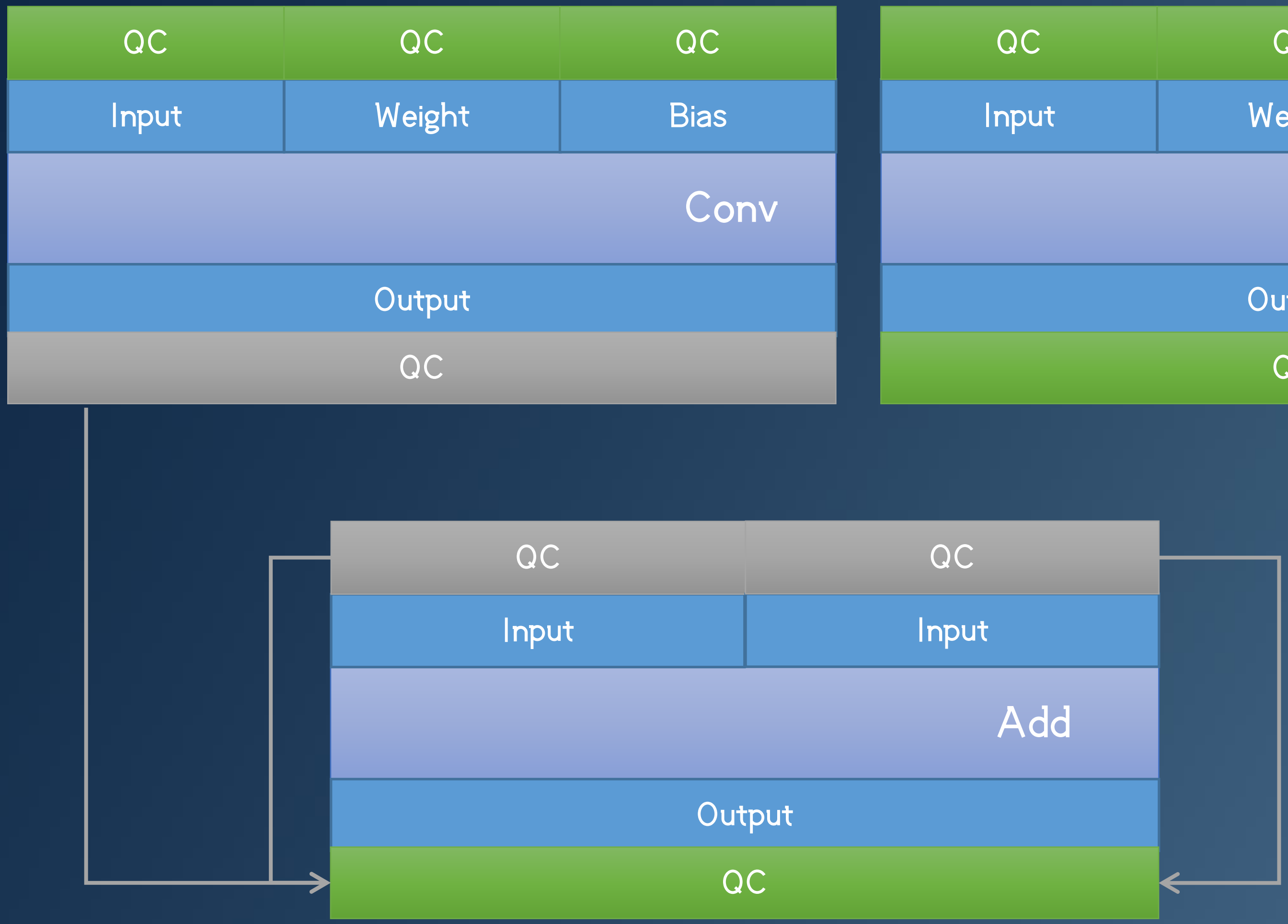
Union - Quantize



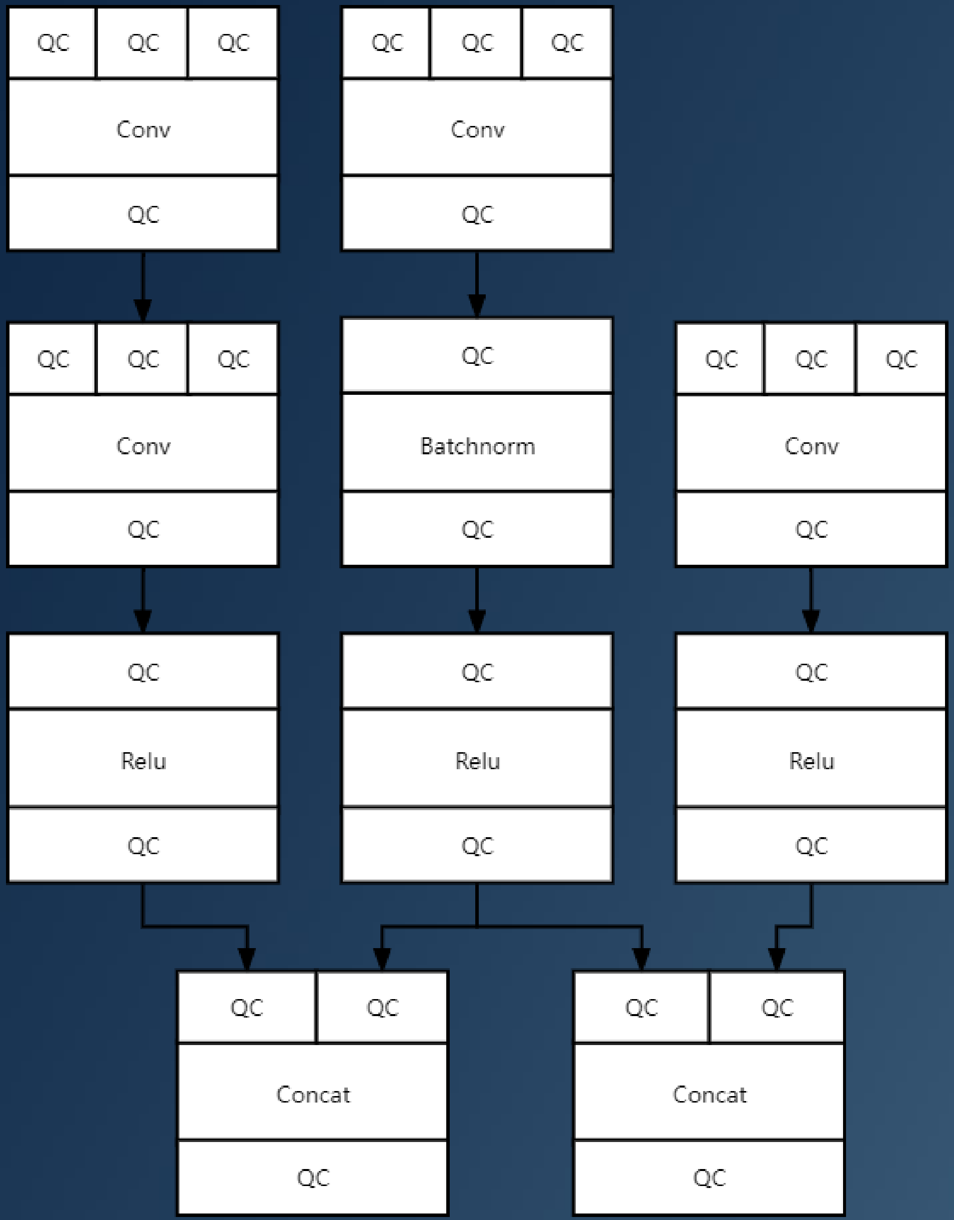
ConvAdd 联合定点（如可能）

6.2.4 联合定点

Union - Quantize



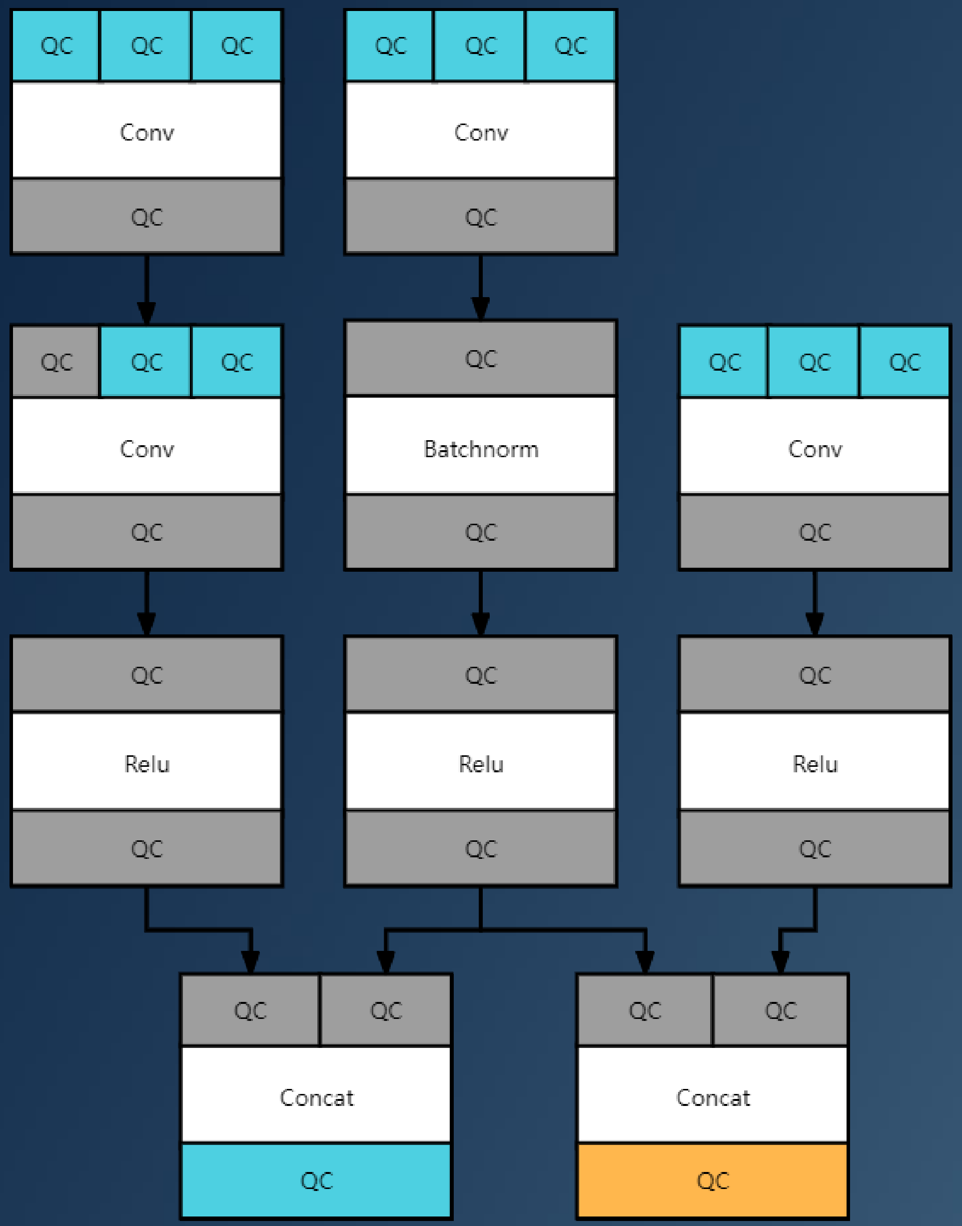
ConvAdd 联合定点（如可能）



已知硬件存在

- conv - conv融合
- conv - relu融合
- conv - batchnorm融合
- concat联合定点

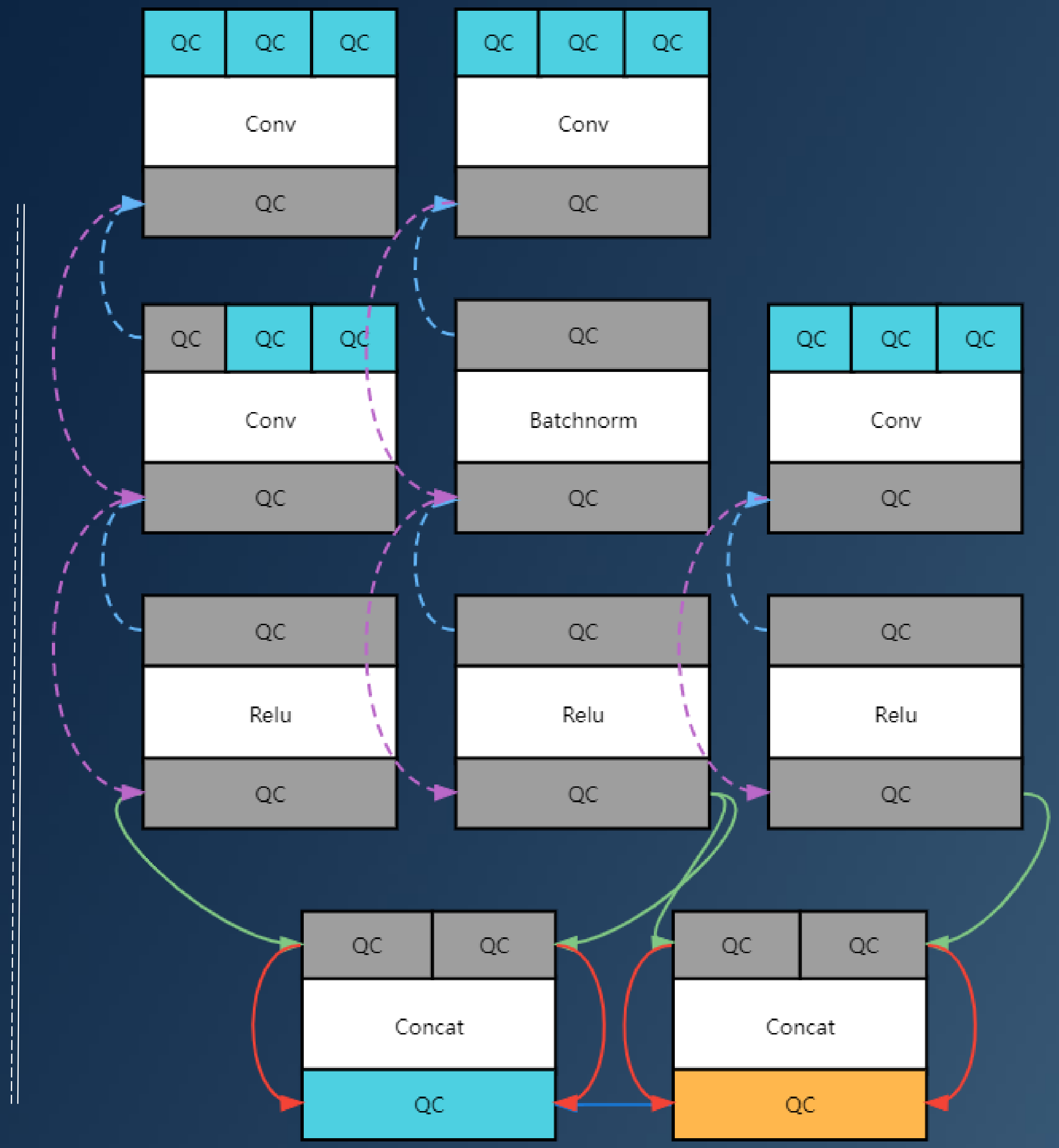
在如此规则下，左图应当如何联合定点？



已知硬件存在

- conv - conv融合
- conv - relu融合
- conv - batchnorm融合
- concat联合定点

在如此规则下，左图应当如何联合定点？

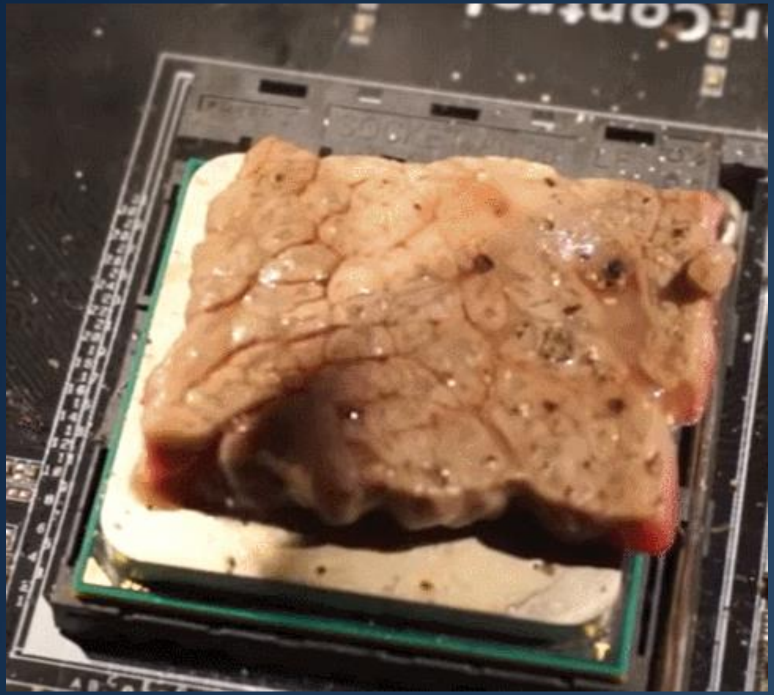


已知硬件存在

- conv - conv融合
- conv - relu融合
- conv - batchnorm融合
- concat联合定点

在如此规则下，左图应当如何联合定点？

联系我们 https://github.com/openppl-public



广告位招租

微信群

QQ群（入群密令OpenPPL）