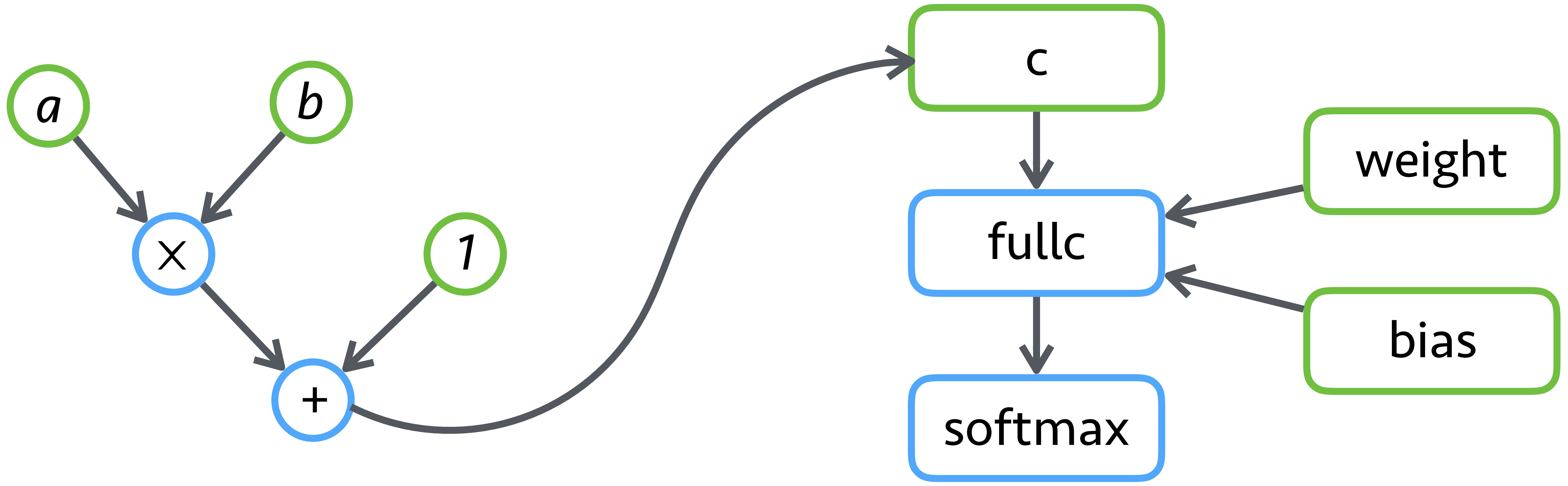
Back-end System

Front-end

```python
import mxnet as mx
a = mx.nd.zeros((100, 50))
b = mx.nd.ones((100, 50))
c = a * b
c += 1
```

```python
import mxnet as mx
net = mx.symbol.Variable('data')
net = mx.symbol.FullyConnected(
          data=net, num_hidden=128)
net = mx.symbol.SoftmaxOutput(data=net)
texec = mx.module.Module(net)
texec.forward(data=c)
texec.backward()
```

Back-end

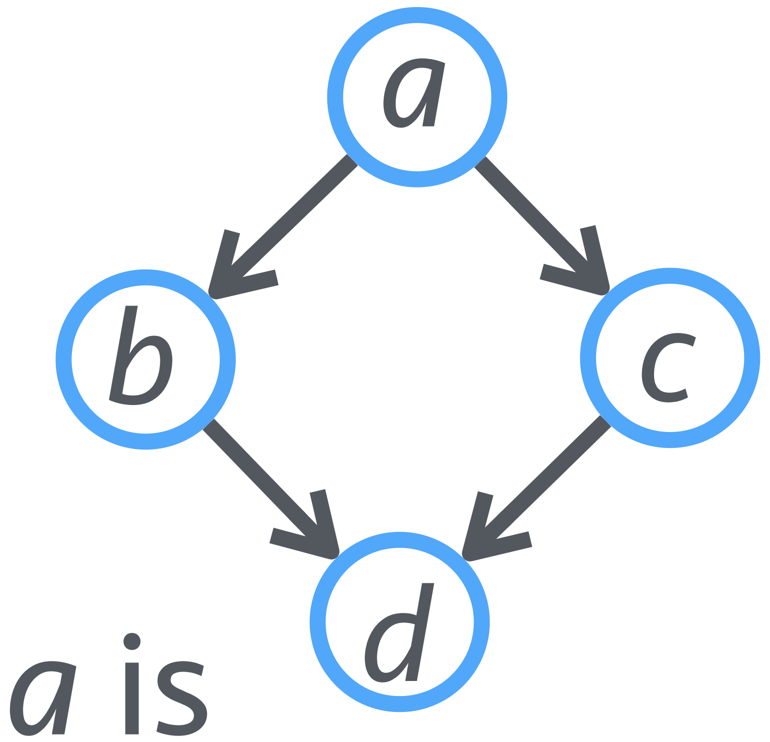


- ✧ Optimization
  - ✓ Memory optimization
  - ✓ Operator fusion
- ✧ Scheduling
  - ✓ Auto-parallelization

Memory Optimization

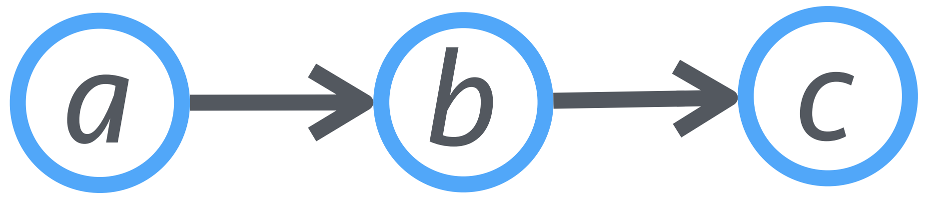
Traverse the computation graph to reduce the memory footprint with time complexity linear in graph size

aliveness analysis



now *a* is deletable

shared space between variables

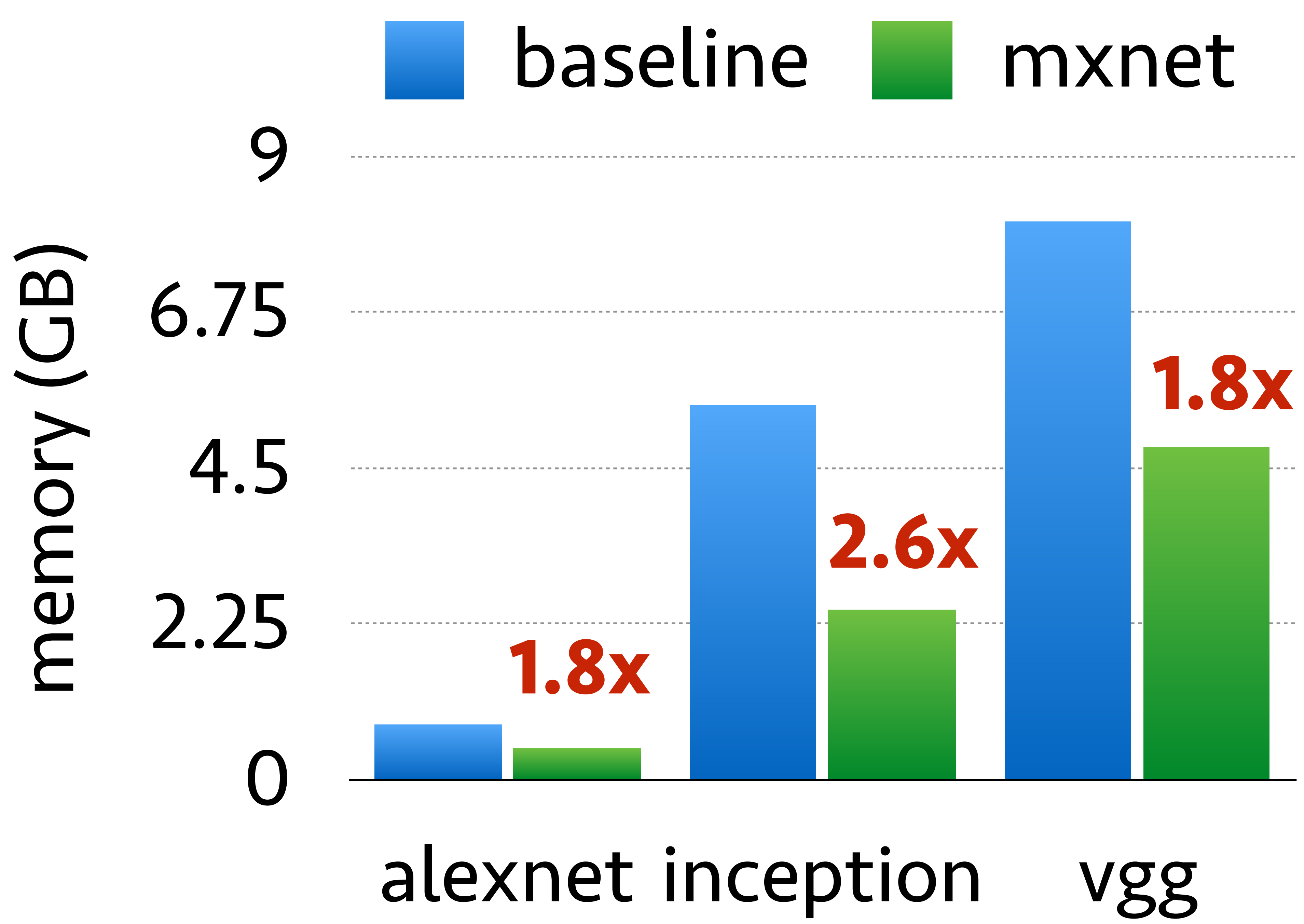


share *a* and *b*

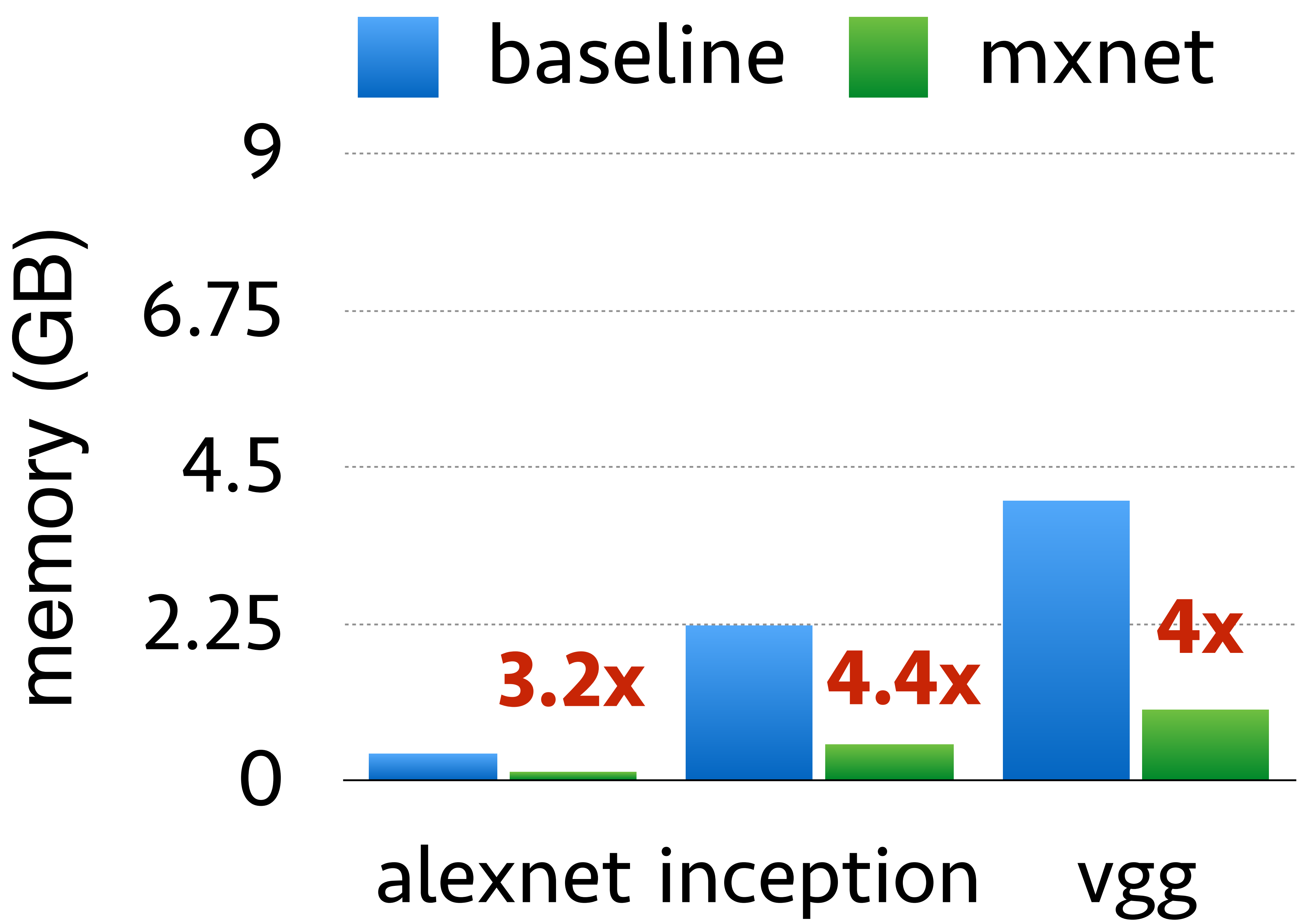
Results for Deep CNNs

IMAGENET winner neural networks

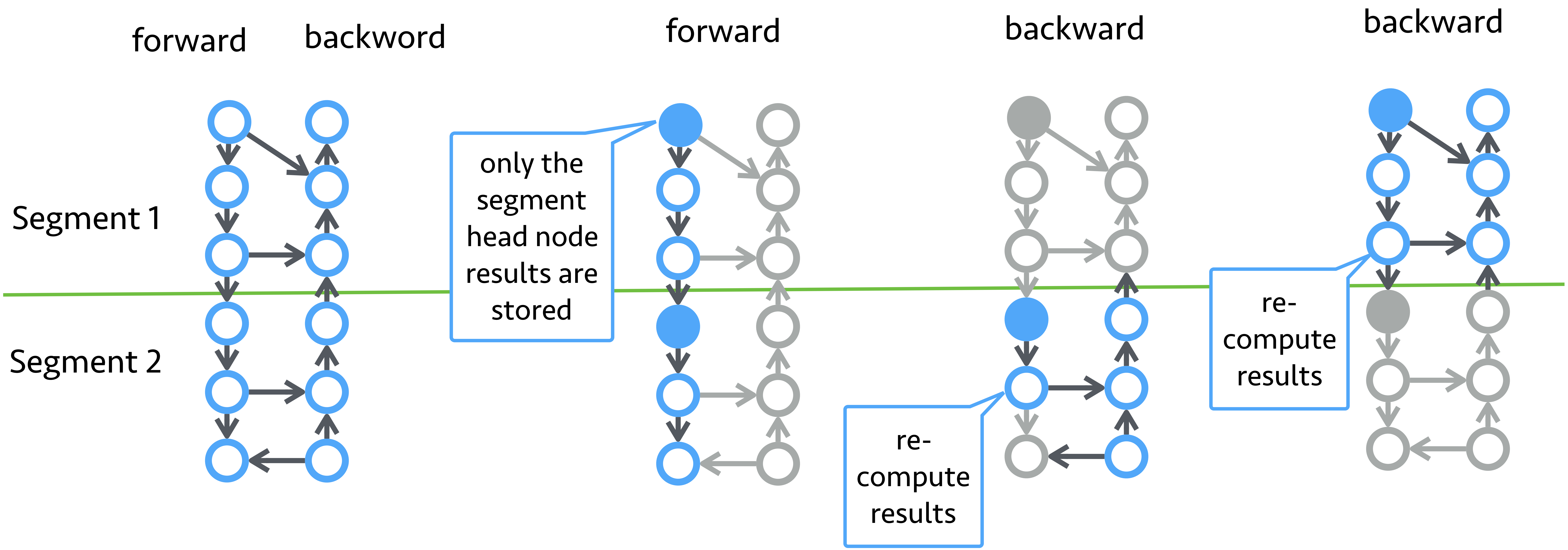
Training



Prediction



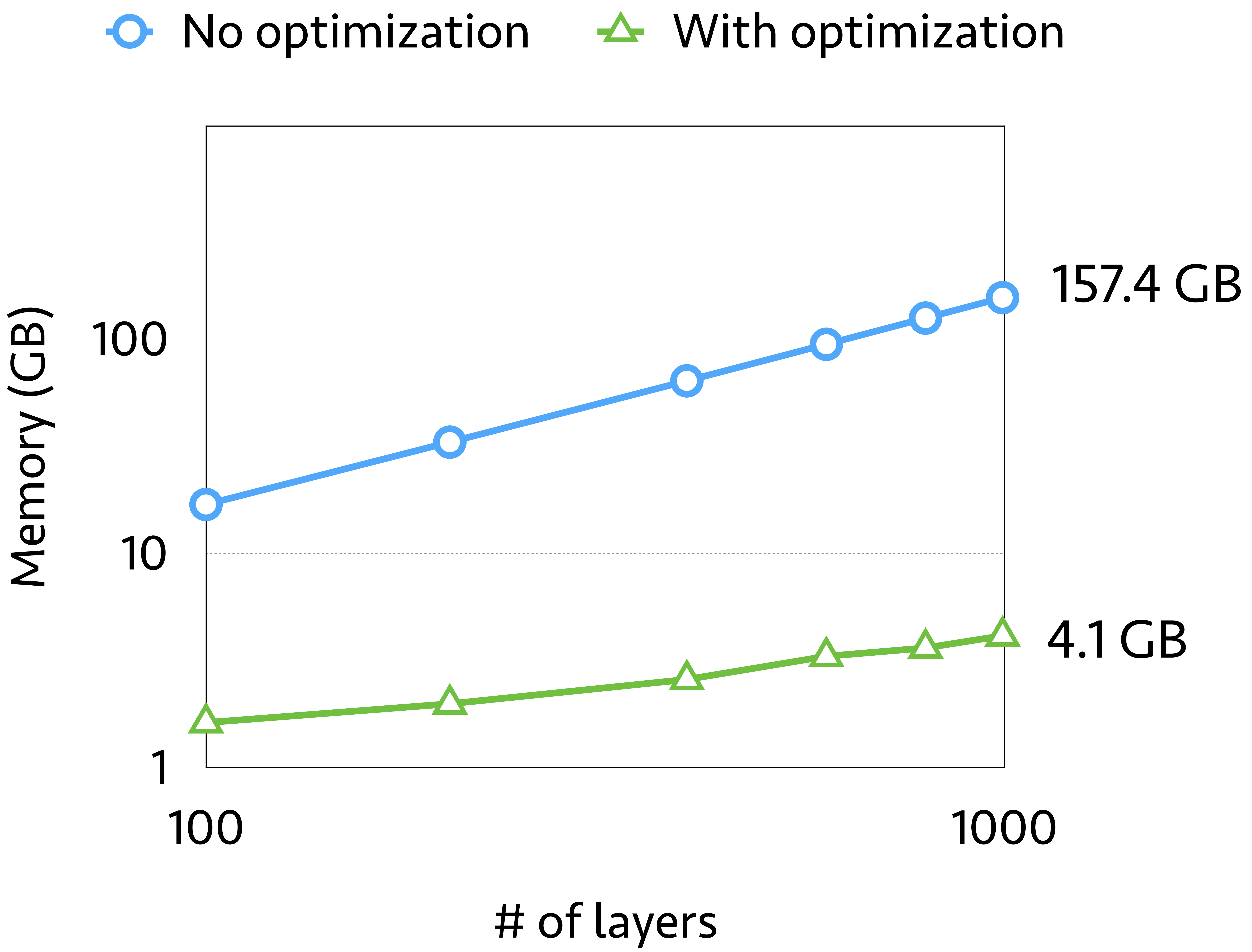
Trade Computation for Memory



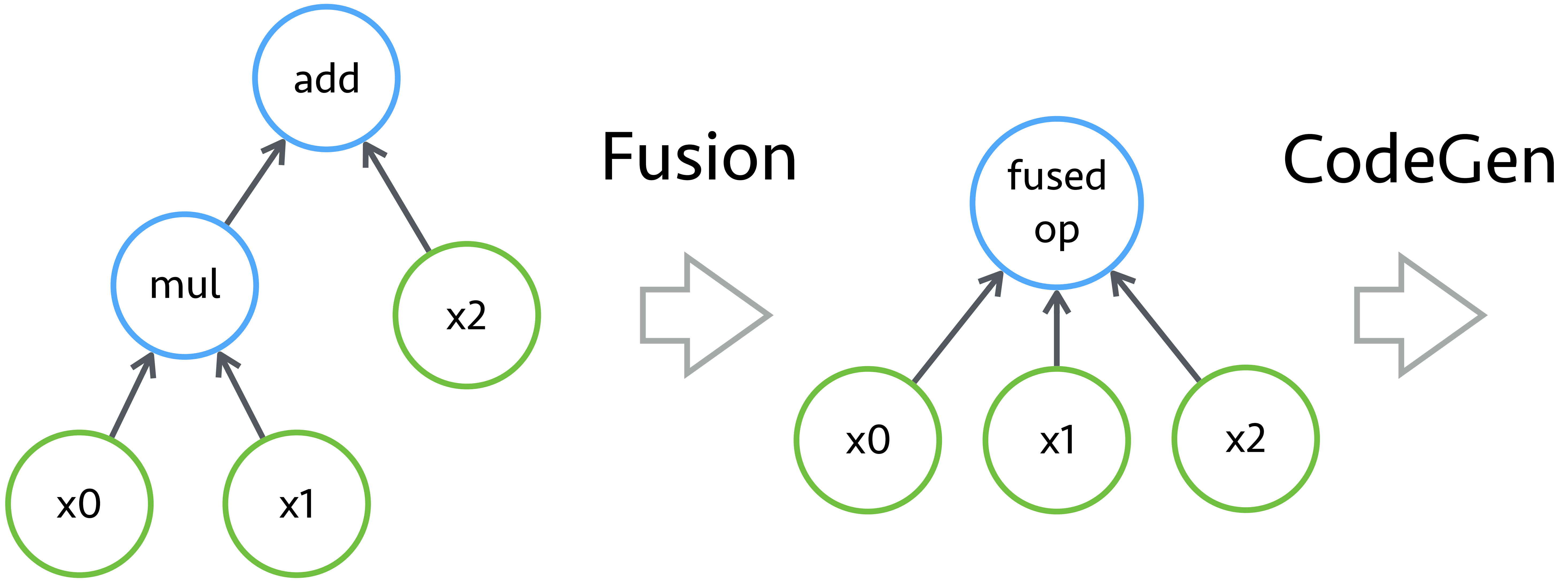
- Needs an extra forward pass
- Reduces the memory complexity from $O(n)$ to $O(\sqrt{n})$, where *n* is the number of layers

Results on ResNet

- Batch size = 32
- Increase 30% computation cost when optimization is applied



Operator Fusion and Runtime Compilation




```
extern "C" __global__ fusion_kernel (uint32_t num_element,
    float *x0, float *x1, float *x2, float *y) {
  int global_idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (global_idx < num_element)
    y[global_idx] = (x0[global_idx] * x1[global_idx]) + x2[global_
}
```

Fuse Adam into a single operator

```
variance *= self.beta2
variance += (1 - self.beta2) * square(grad, out=grad)

coef1 = 1. - self.beta1**t
coef2 = 1. - self.beta2**t
lr *= math.sqrt(coef2)/coef1
```

20% performance improvement on ResNet