MXNET-AMP AND MXNET-TRT FOR FAST DEEP LEARNING TRAINING AND INFERENCE

Wenming Ye (Amazon), Przemyslaw Tredak (NVIDIA), 11.06.2019

DEEP LEARNING FRAMEWORKS

The landscape

aws

mxnet

NVIDIA

DEEP LEARNING FRAMEWORKS

The landscape

DEEP LEARNING FRAMEWORKS

- Fully open source (Apache incubating project)
- Mixes imperative and declarative programming
- Combines ease of use with speed
- Ready for deployment

APACHE MXNET

Mixing programming paradigms

Static graph

Hybrid

Imperative

APACHE MXNET

Mixing programming paradigms

Static graph → Hybrid ← Imperative

Eager mode (→) JIT (←)

APACHE MXNET

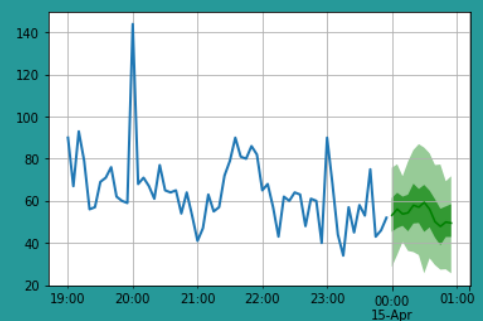
Gluon Ecosystem

GLUON

GluonCV

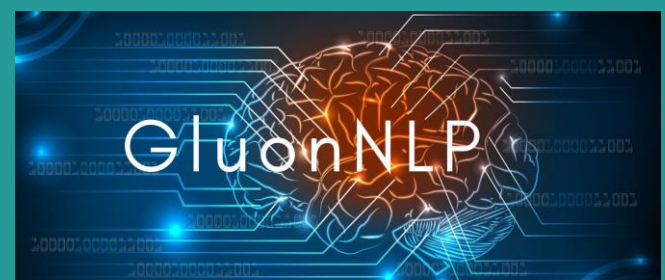
- Over 190 pretrained models with SOTA accuracy
- Training scripts
- Tutorials

GluonTS



- Time series forecasting

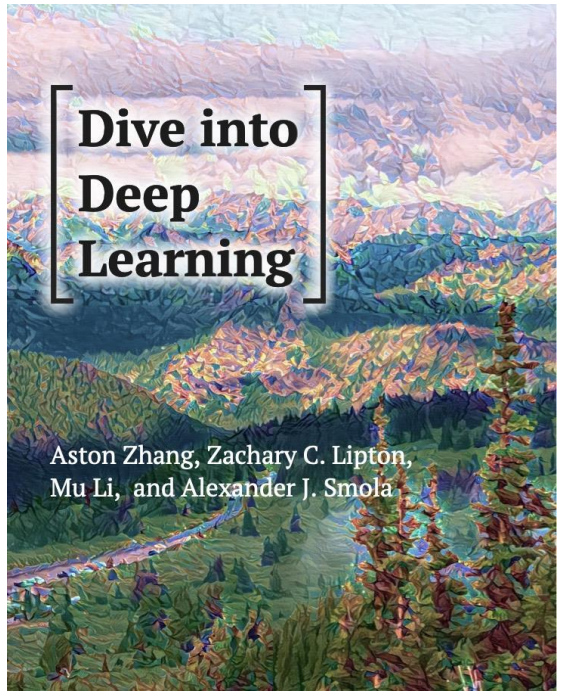
GluonNLP



- Word embeddings
- Language models
- Neural Machine Translation
- BERT, ELMO, XLNet, ERNIE, GPT-2…
- Reproducible scripts

APACHE MXNET

Gluon Ecosystem



http://d2l.ai

MULTI-LANGUAGE SUPPORT

Java | Perl | Julia | Clojure

Python | Scala | C++ | R

Frontend

Backend

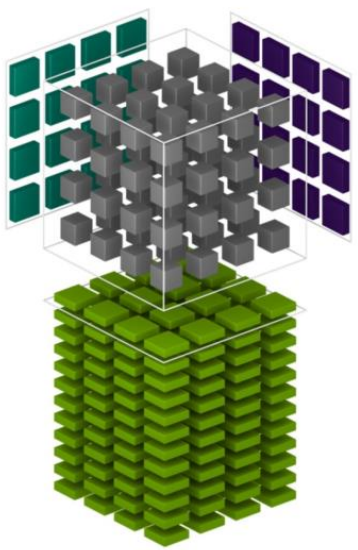
C++


AUTOMATIC MIXED
PRECISION IN MXNET

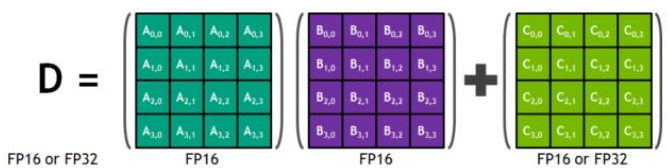
TENSORCORES AND MIXED PRECISION

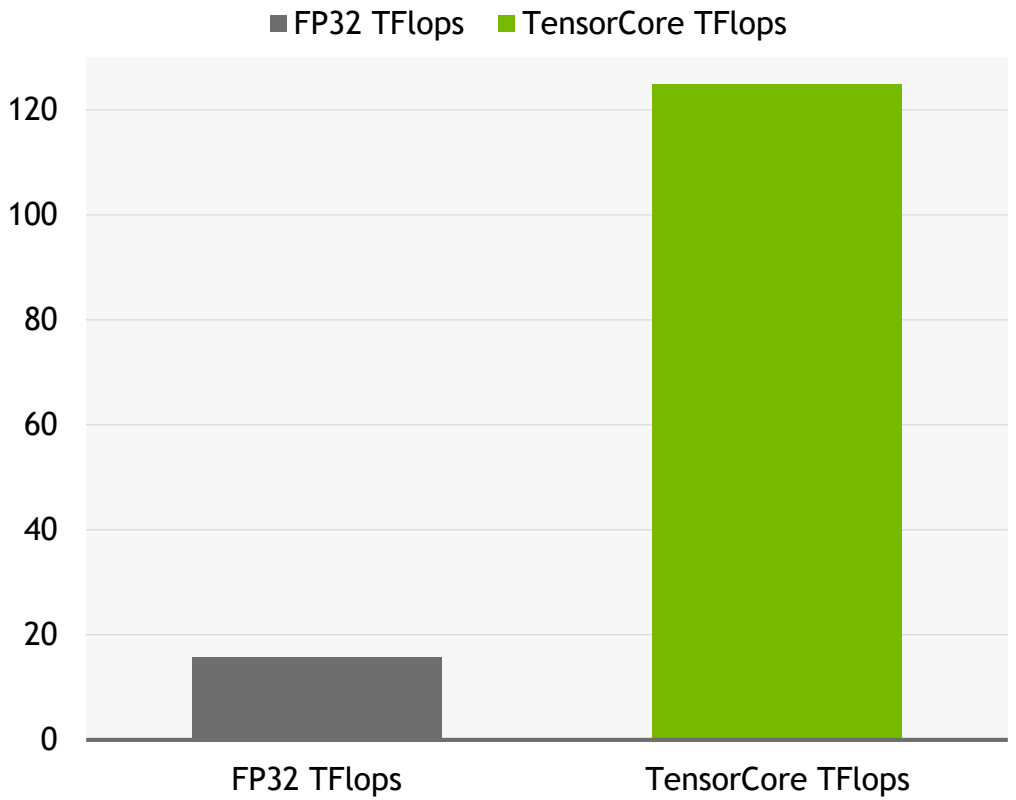
Starting with Volta, NVIDIA GPUs feature TensorCores

They greatly speed up matrix multiplication



Using them requires mixed precision





MIXED PRECISION RECIPE

Theory

- Cast input to FP16, cast back to FP32 before the softmax
- Keep “master copy” of the weights in FP32
- Scale the loss to keep gradients in FP16 dynamic range

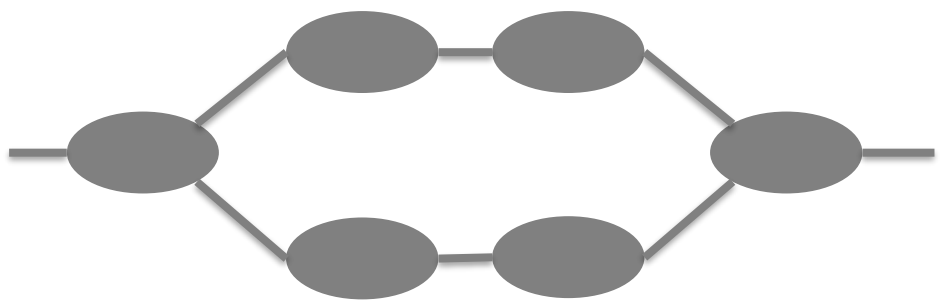
MIXED PRECISION RECIPE

In practice

- Cast input to FP16, cast back to FP32 before the softmax
  - What about Norm, Mean, etc.?
- Keep “master copy” of the weights in FP32
  - optimizer.multi_precision=True
- Scale the loss to keep gradients in FP16 dynamic range
  - What should the loss scale be? How to make it dynamic?

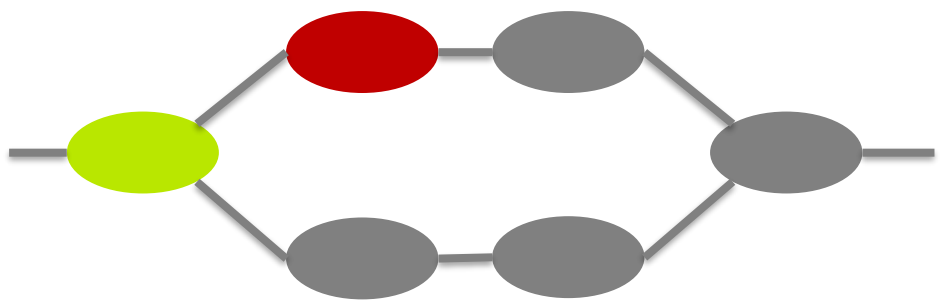
AMP: AUTOMATIC MIXED PRECISION

- Automatic casting of the model
  - Convolution, FullyConnected -> FP16
  - Norm, Mean, SoftMax, etc. -> FP32
  - Add, Mul etc. -> Cast to widest type
- Utilities for dynamic loss scaling



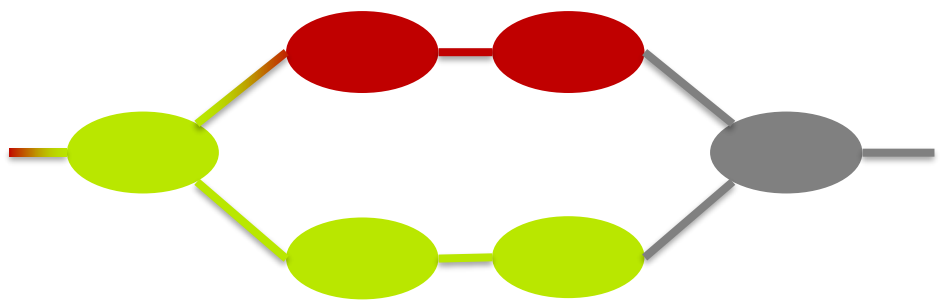
AMP: AUTOMATIC MIXED PRECISION

- Automatic casting of the model
  - Convolution, FullyConnected -> **FP16**
  - Norm, Mean, SoftMax, etc. -> **FP32**
  - Add, Mul etc. -> Cast to widest type
- Utilities for dynamic loss scaling



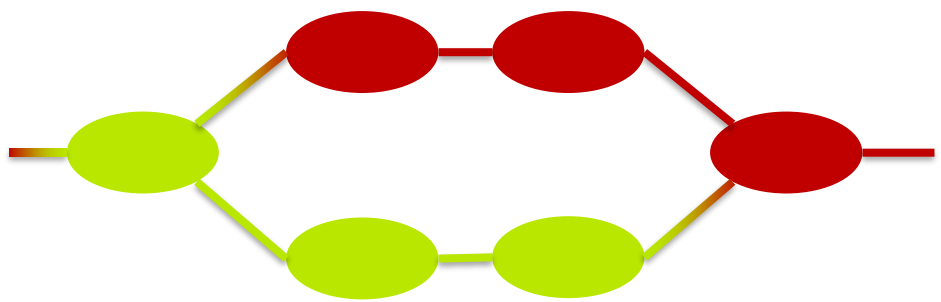
AMP: AUTOMATIC MIXED PRECISION

- Automatic casting of the model
  - Convolution, FullyConnected -> **FP16**
  - Norm, Mean, SoftMax, etc. -> **FP32**
  - Add, Mul etc. -> Cast to widest type
- Utilities for dynamic loss scaling



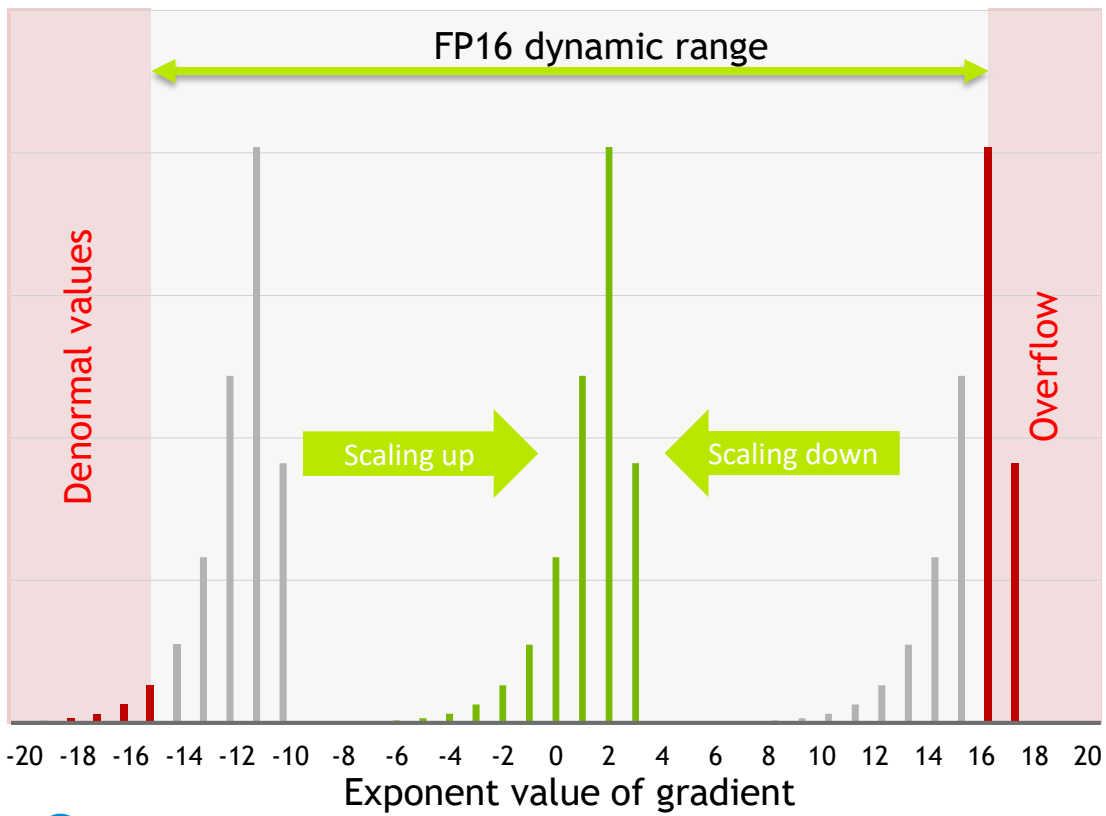
AMP: AUTOMATIC MIXED PRECISION

- Automatic casting of the model
  - Convolution, FullyConnected -> **FP16**
  - Norm, Mean, SoftMax, etc. -> **FP32**
  - Add, Mul etc. -> Cast to widest type
- Utilities for dynamic loss scaling



AMP: AUTOMATIC MIXED PRECISION

- Automatic casting of the model
  - Convolution, FullyConnected -> **FP16**
  - Norm, Mean, SoftMax, etc. -> **FP32**
  - Add, Mul etc. -> Cast to widest type
- Utilities for dynamic loss scaling



MIXED PRECISION RECIPE

AMP

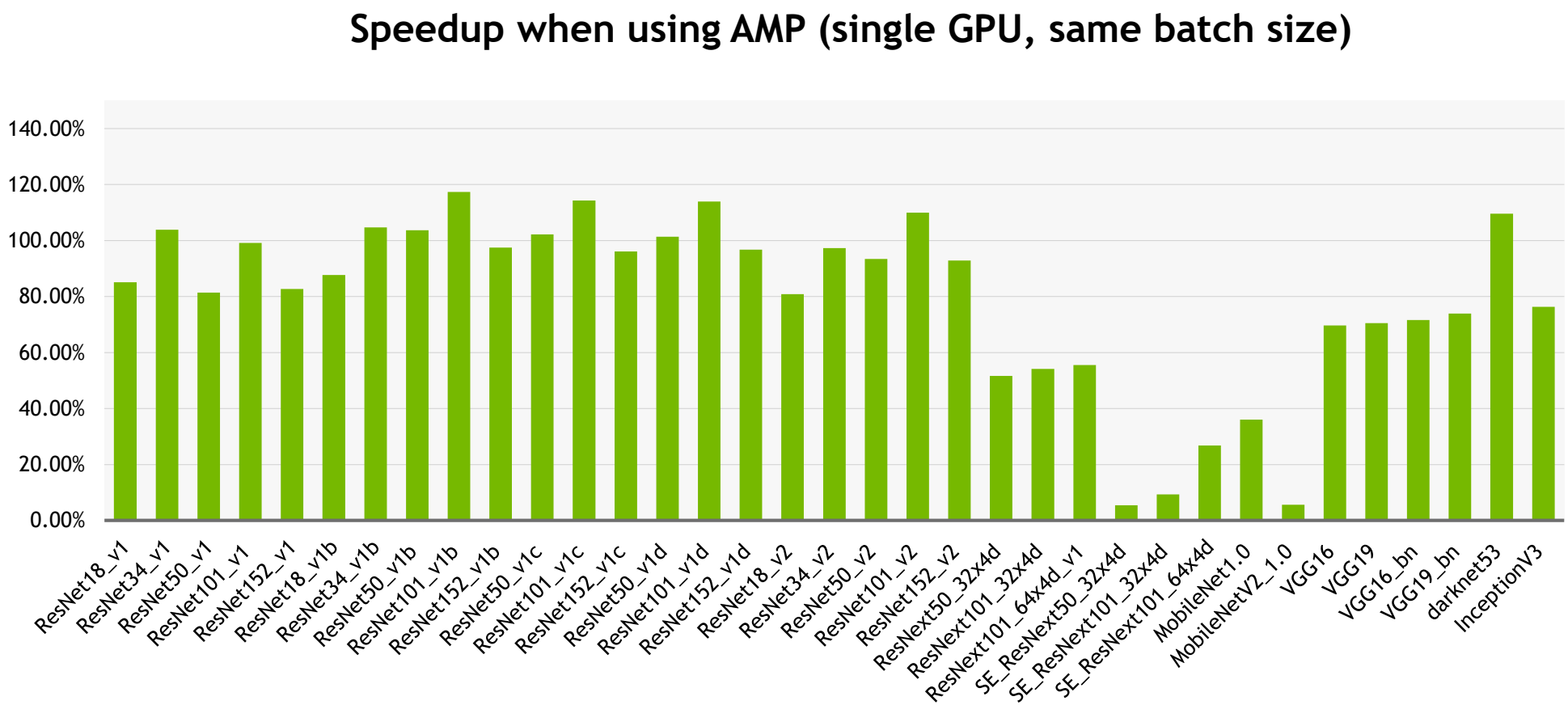
```
net = get_network()
trainer = mx.gluon.Trainer(...)

with autograd.record(True):
    out = net(data)
    l = loss(out, label)

    autograd.backward(loss)
```

MIXED PRECISION RECIPE

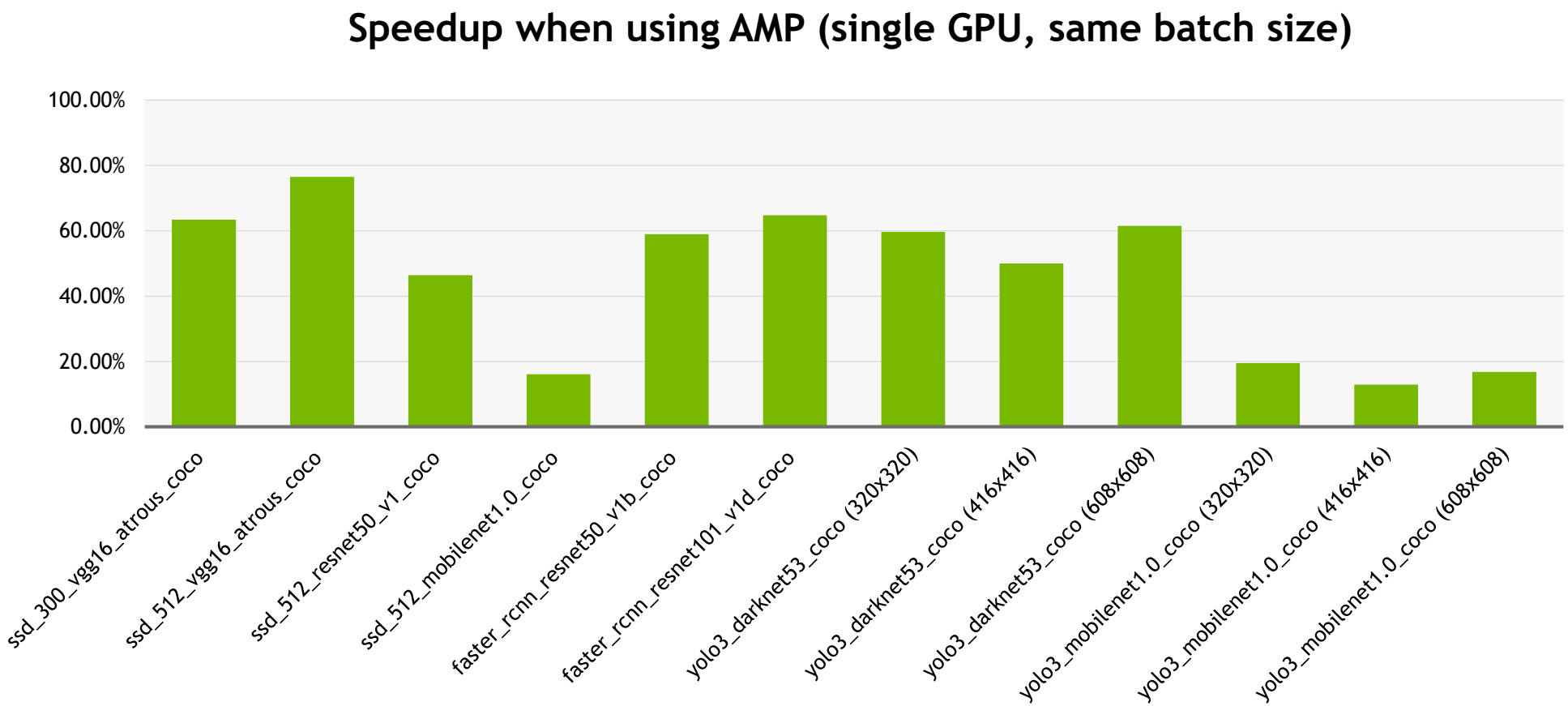
AMP

```
amp.init()

net = get_network()

trainer = mx.gluon.Trainer(...)

amp.init_trainer(trainer)

with autograd.record(True):

    out = net(data)

    l = loss(out, label)

    with amp.loss_scale(loss, trainer) as scaled_loss:

        autograd.backward(scaled_loss)
```

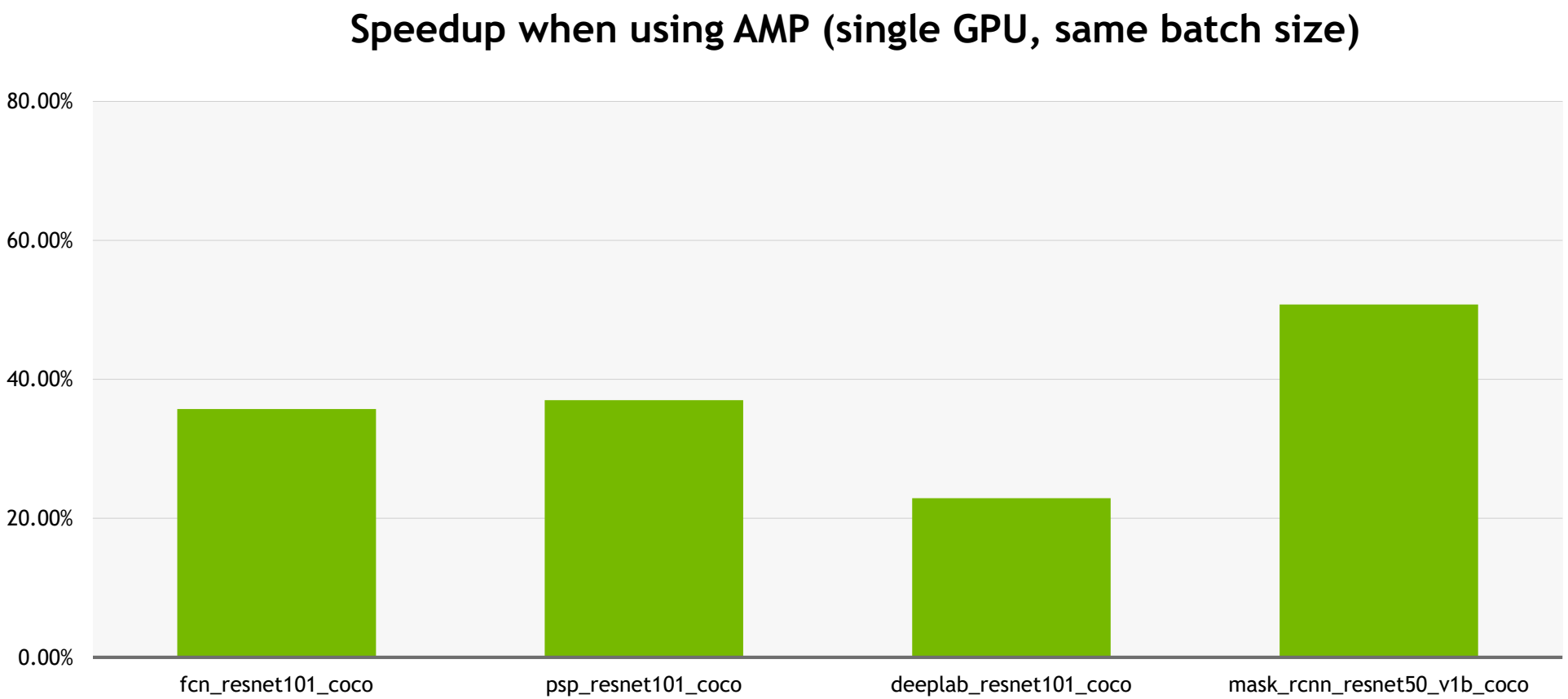
PERFORMANCE - CLASSIFICATION



PERFORMANCE - DETECTION



PERFORMANCE - SEGMENTATION




MXNET-AMP:
LOOKING FORWARD

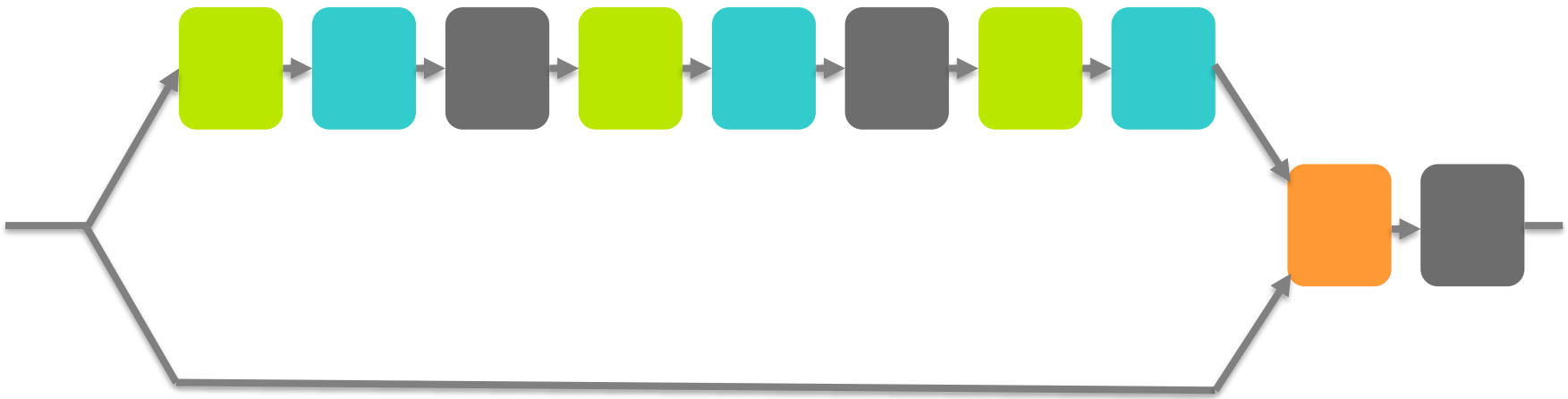
TRAINING OPTIMIZATION

MLPerf

MLPerf

TRAINING OPTIMIZATION

MLPerf ResNet 50 under the hood



Convolution

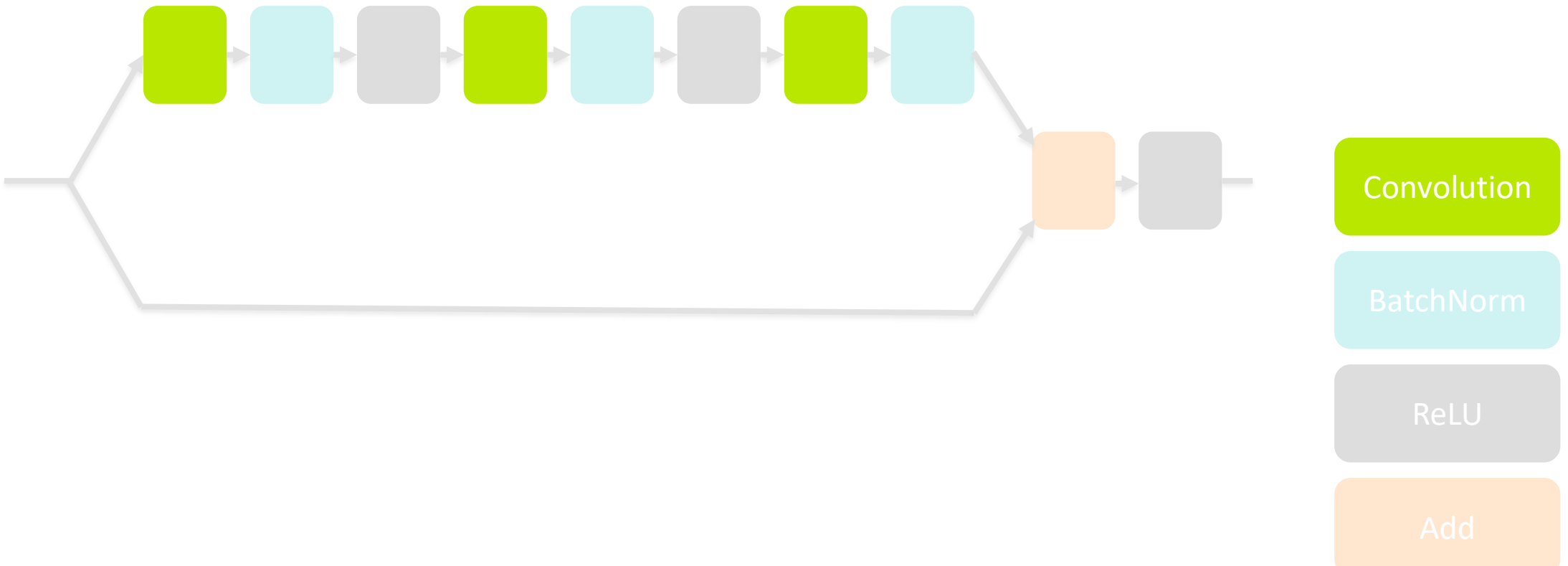
BatchNorm

ReLU

Add

TRAINING OPTIMIZATION

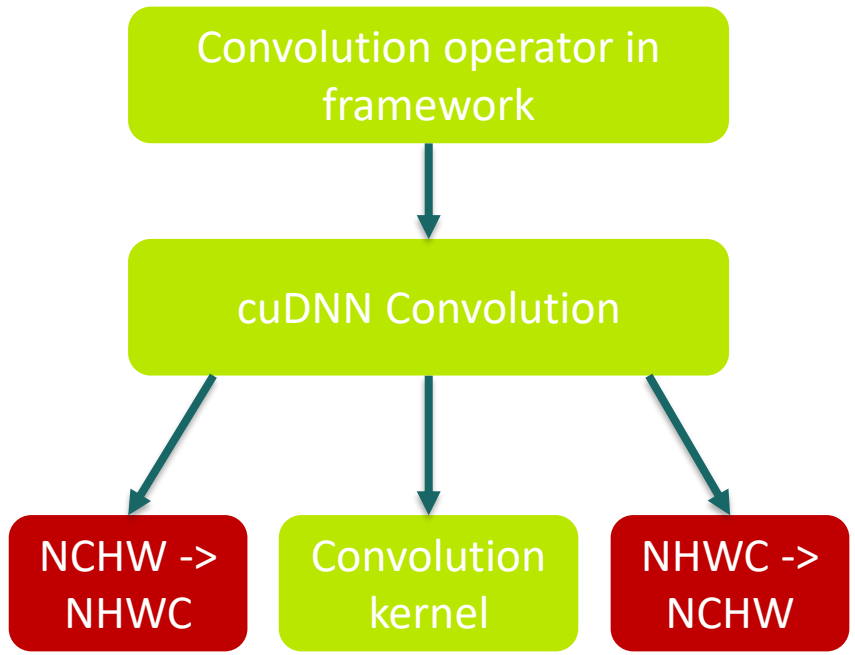
Anatomy of TensorCore convolution



TRAINING OPTIMIZATION

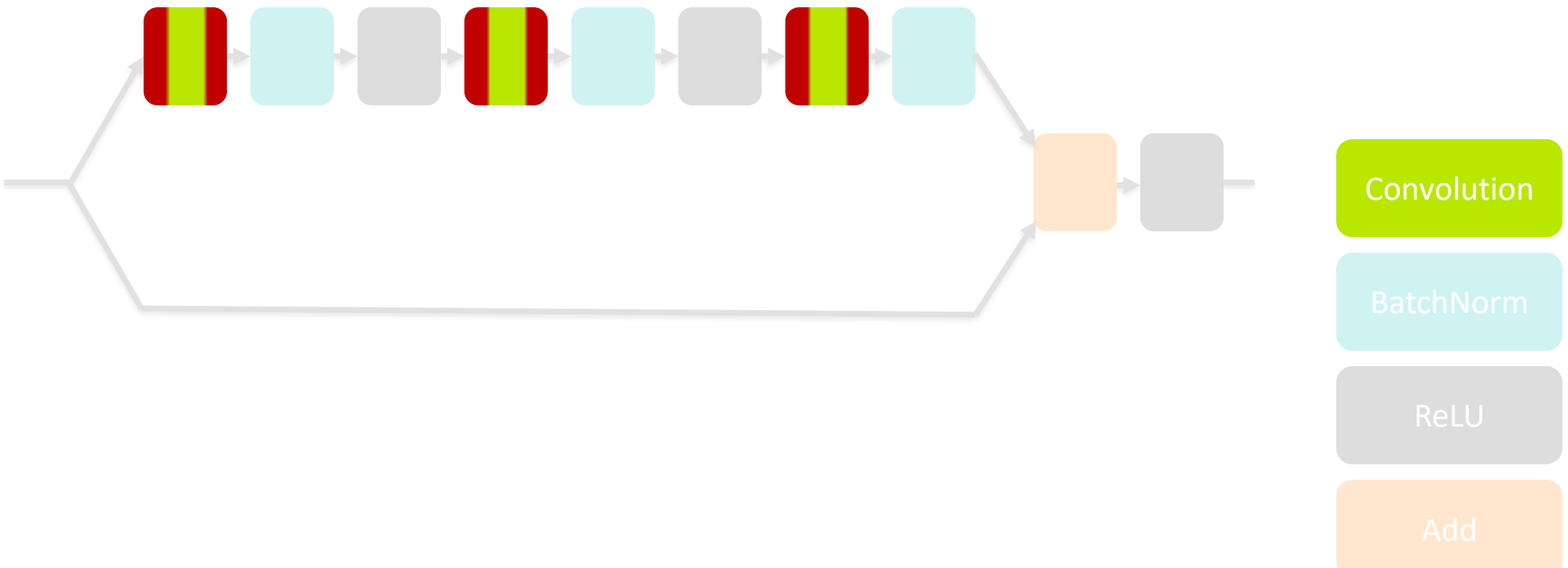
Anatomy of TensorCore convolution

- TensorCore convolution requires special data layout (NHWC) for efficient loading of inputs and weights
- DL frameworks use NCHW data layout, since it is the best choice for FP32
- cuDNN internally transposes data before calling convolution kernel



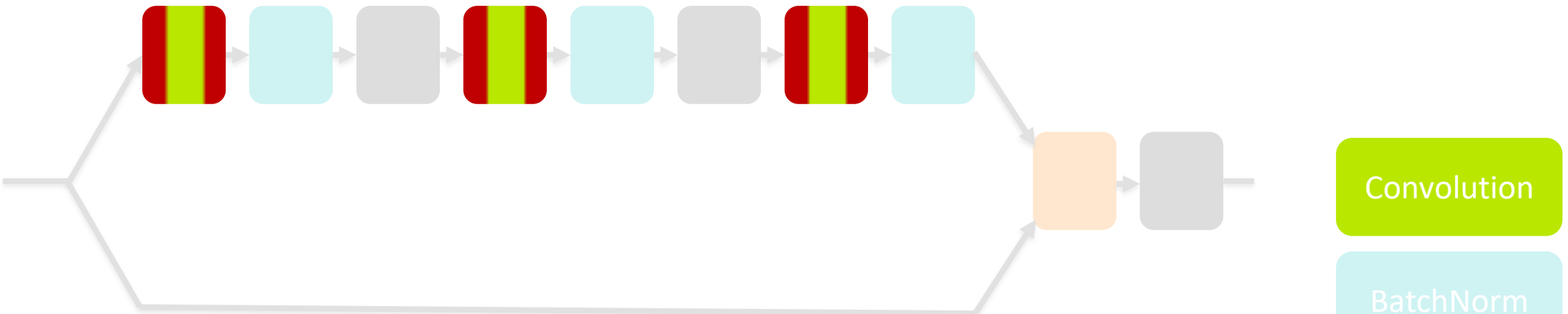
TRAINING OPTIMIZATION

Anatomy of TensorCore convolution



TRAINING OPTIMIZATION

Anatomy of TensorCore convolution



Convolution

BatchNorm

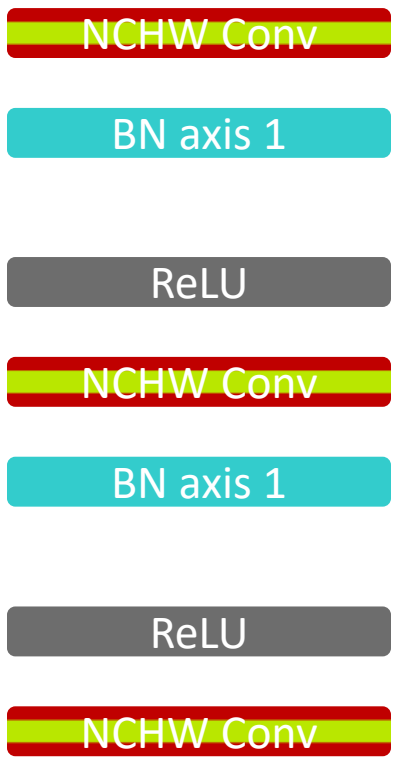
ReLU

Add

AMP should use NHWC!

TRAINING OPTIMIZATION

Automatic Layout Management

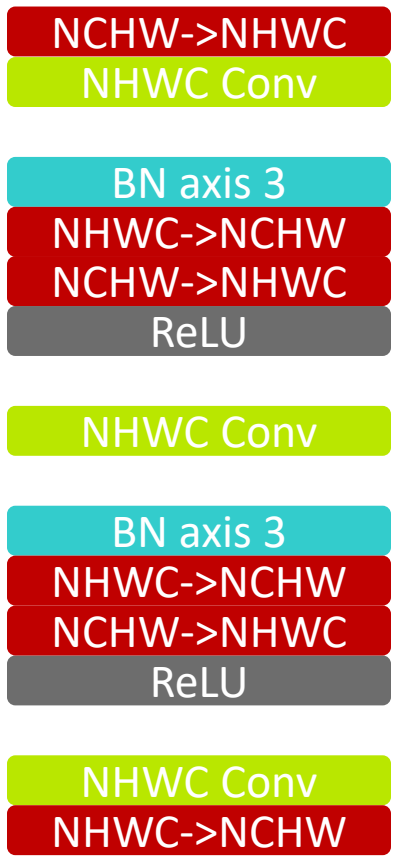


TRAINING OPTIMIZATION

Automatic Layout Management

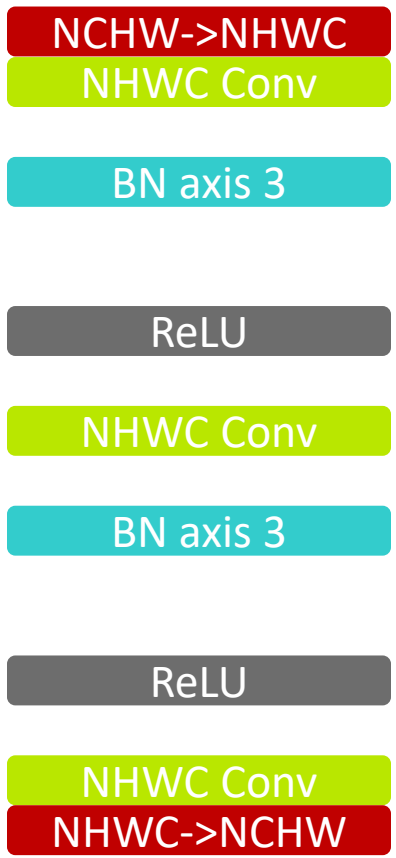
TRAINING OPTIMIZATION

Automatic Layout Management

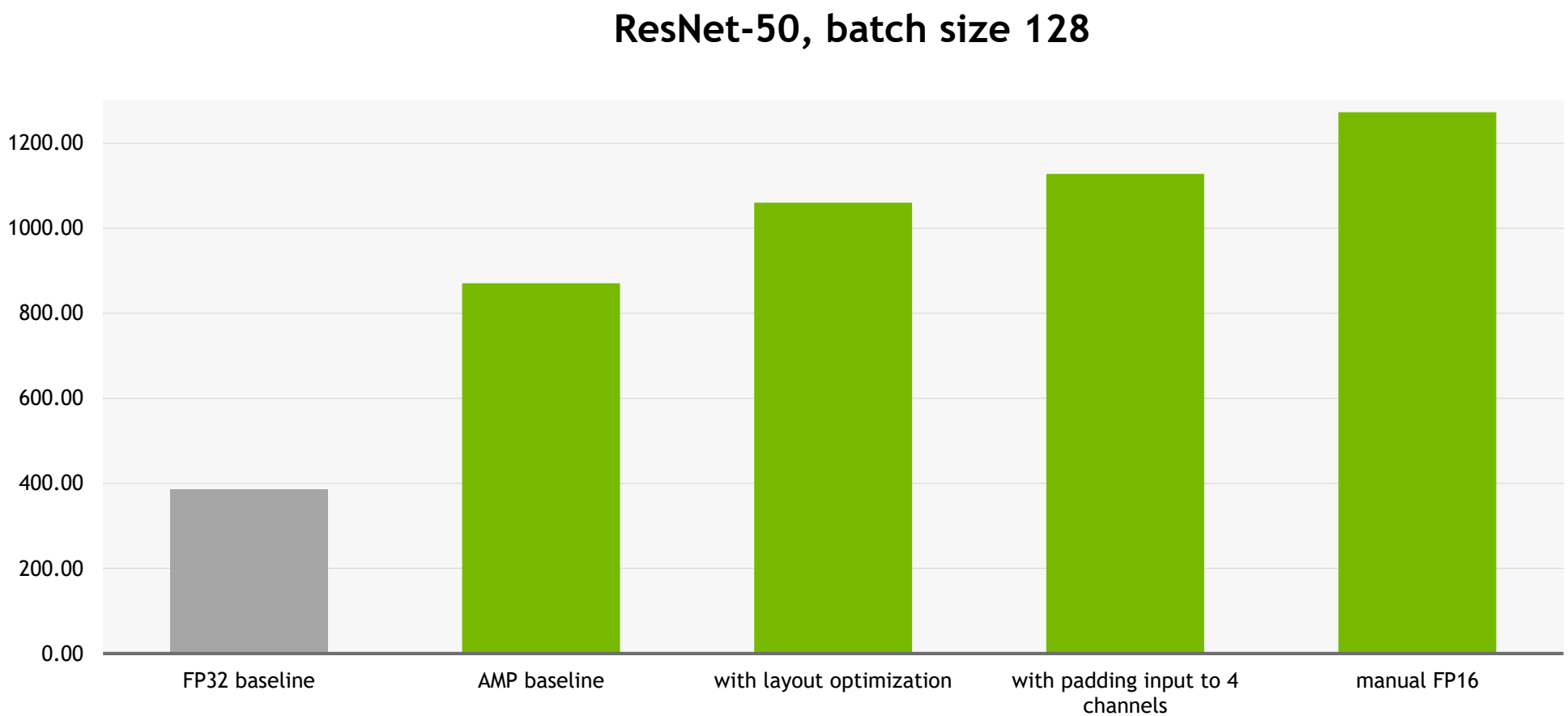


TRAINING OPTIMIZATION

Automatic Layout Management



PRELIMINARY PERFORMANCE RESULTS




MXNET TENSORRT
INTEGRATION

INFERENCE OPTIMIZATION

Similar to training optimizations but with a few extras

- **At inference time we make use of same speedups as in training (TensorCores)**

Inference time only:

Operators have fixed values after training (additional opportunity to fuse operators)

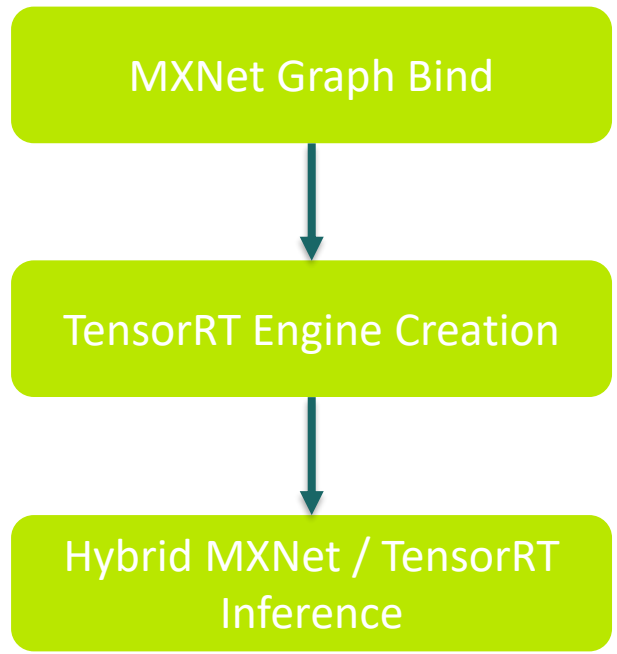
Many intermediate memory buffers needed for gradient storage can be removed (yields a storage savings, and potentially memory read / write savings)

- Our goal is to get many of these inference-only optimization for free by integrating with TensorRT (has been shown to provide dramatic inference-time speedups across a variety of workloads)
- **We also want to enable these speedups with minimal developer effort or code change**

MXNET TENSORRT INTEGRATION

Subgraph compilation

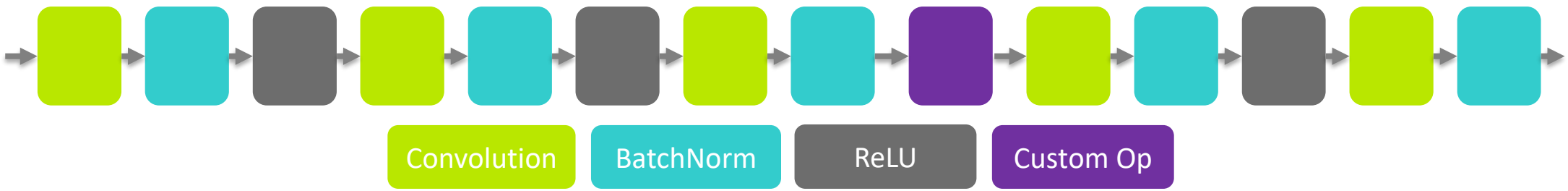
- The MXNet community developed a generic subgraph-API for MXNet and TensorRT has been integrated as a first class citizen (Supports Vendor specific implementation)
- This API lets vendor devs tell MXNet which operators are supported by their acceleration software
- While running inference MXNet will select subgraphs automatically based on this support level, and fall back to CUDA implementations for unsupported or custom operators (region of interest, tasks, etc)



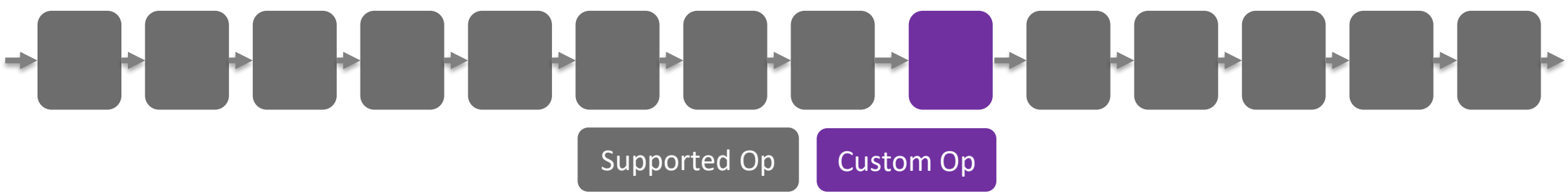
MXNET TENSORRT INTEGRATION

Subgraph compilation – under the hood

- We start with a normal computation graph for an MXNet model



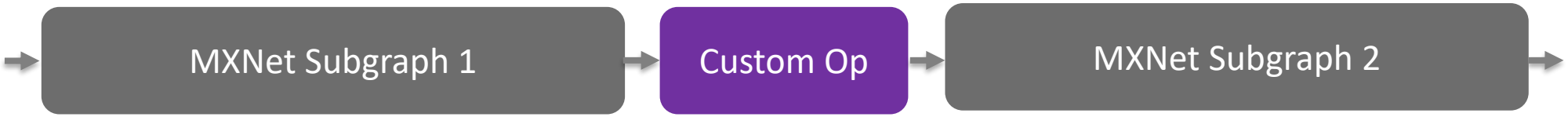
- We search for subgraphs of supported operators



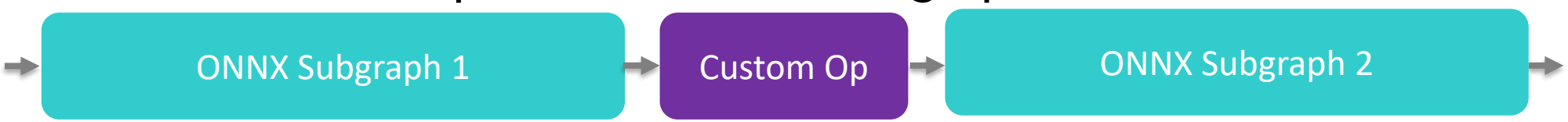
MXNET TENSORRT INTEGRATION

Subgraph compilation – under the hood cont.

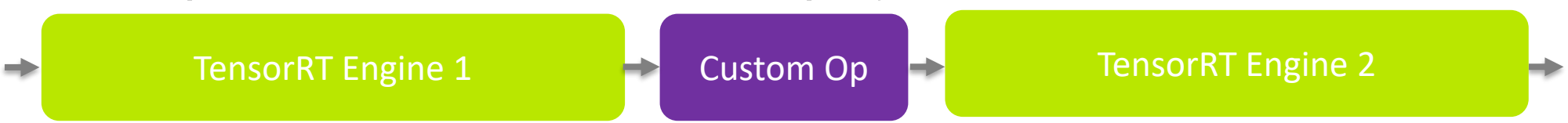
- We now have our full graph partitioned into a number of subgraphs



- We convert the operators in each subgraph to ONNX

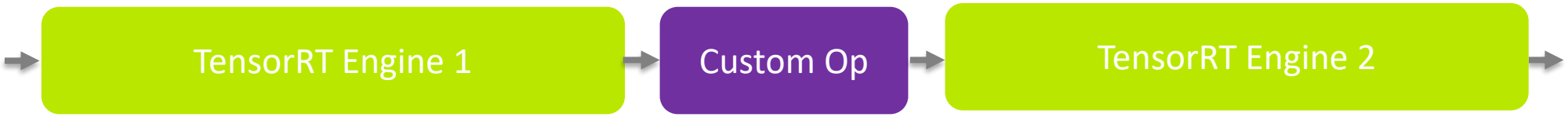


- Finally, we convert the ONNX graphs to TensorRT engines using the open source ONNX-TensorRT project



MXNET TENSORRT INTEGRATION

Subgraph compilation



- When inference is run on this graph all TensorRT engine nodes will be executed with a library call to TensorRT
- Many standard models will collapse the entire graph to a single TensorRT node
- Custom ops or unsupported ops will continue to run as normal, using their default MXNet CUDA implementations
- All of this conversion happens with little coding effort required by MXNet users
- This process is focused on high-performance inference scenarios, so the API is exposed from Python, our C API, and our C++ package

TENSORRT INTEGRATION RECIPE

Normal Inference in MXNet

```
executor = sym.simple_bind(ctx=mx.gpu(), data=batch_shape, grad_req='null',
                           force_rebind=True)
executor.copy_params_from(arg_params, aux_params)
y_gen = executor.forward(is_train=False, data=input)
```

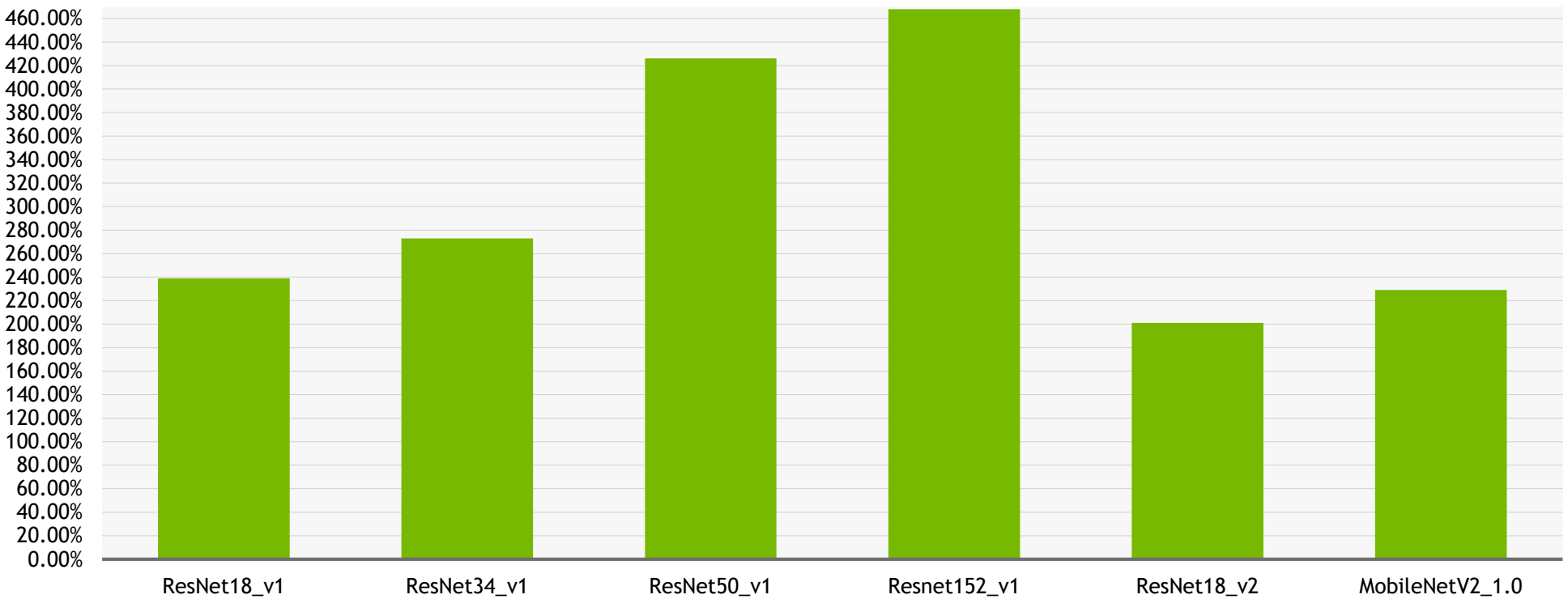
TENSORRT INTEGRATION RECIPE

TensorRT enabled Inference in MXNet

```
trt_sym = sym.get_backend_symbol('TensorRT')
arg_params, aux_params = mx.contrib.tensorrt.init_tensorrt_params(trt_sym,
                                                                  arg_params,
                                                                  aux_params)
executor = trt_sym.simple_bind(ctx=mx.gpu(), data=batch_shape, grad_req='null',
                               force_rebind=True)
executor.copy_params_from(arg_params, aux_params)
y_gen = executor.forward(is_train=False, data=input)
```

PERFORMANCE - CLASSIFICATION

Speedup when using TensorRT Integration (single GPU, same batch size)




NVIDIA