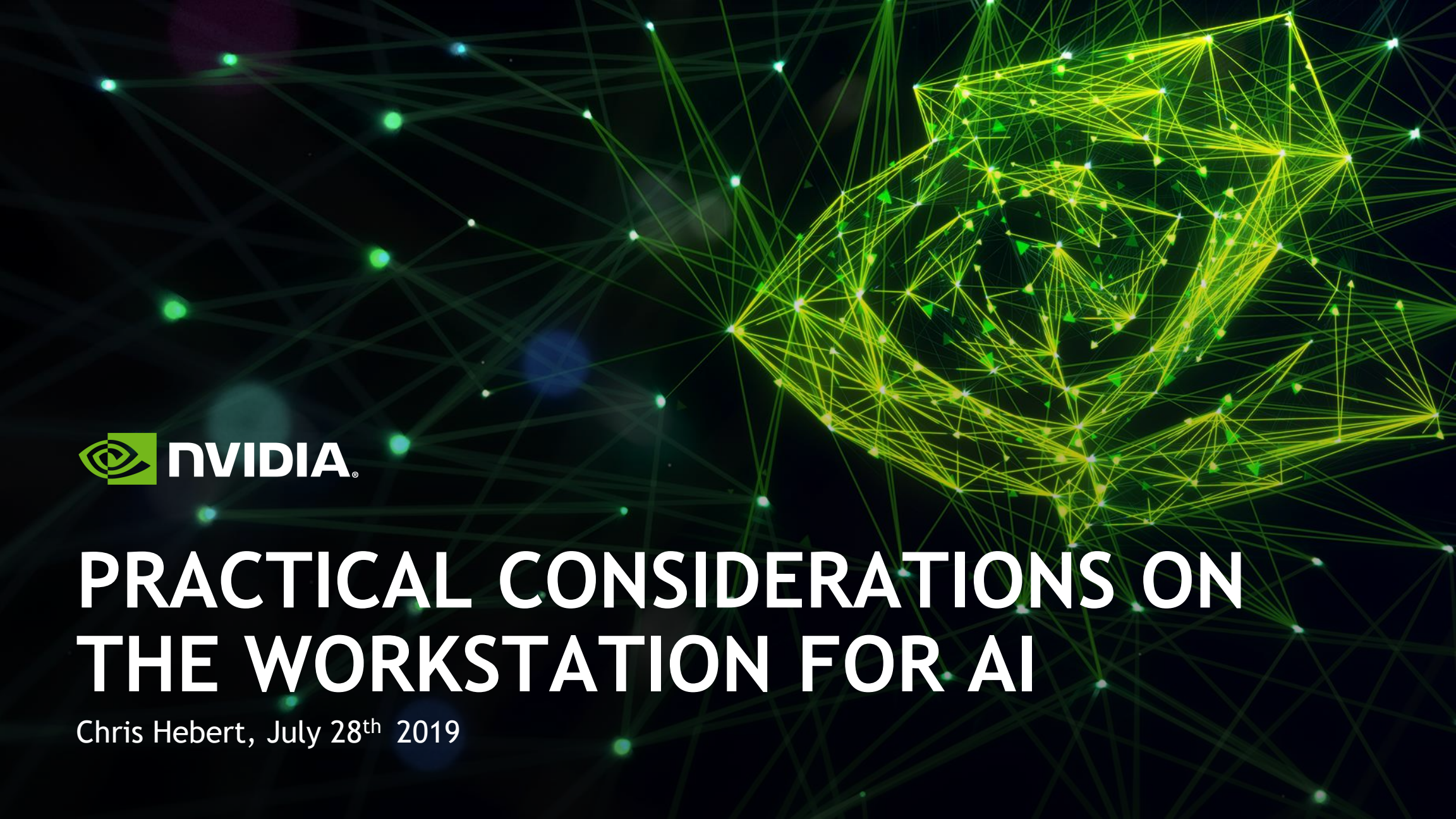


PRACTICAL CONSIDERATIONS ON THE WORKSTATION FOR AI

Chris Hebert, July 28th 2019

AGENDA

Practical Considerations For Workstation Deployment

- Research to Production
  - How do we deploy
  - What do we need
  - Scientist vs Engineer
  - Using ONNX and WinML
  - TensorRT and cuDNN
- The Last 10 Percent ...

From Research to Production: It just works ... or not?!

Summary

Research To Production

How do we deploy?

- Some constraints for workstation.
  - Well likely be sharing the GPU with many other tasks
  - We may not know ahead of time what hardware will be available
  - Our deployment solution will likely need to integrate with an existing codebase

Research To Production

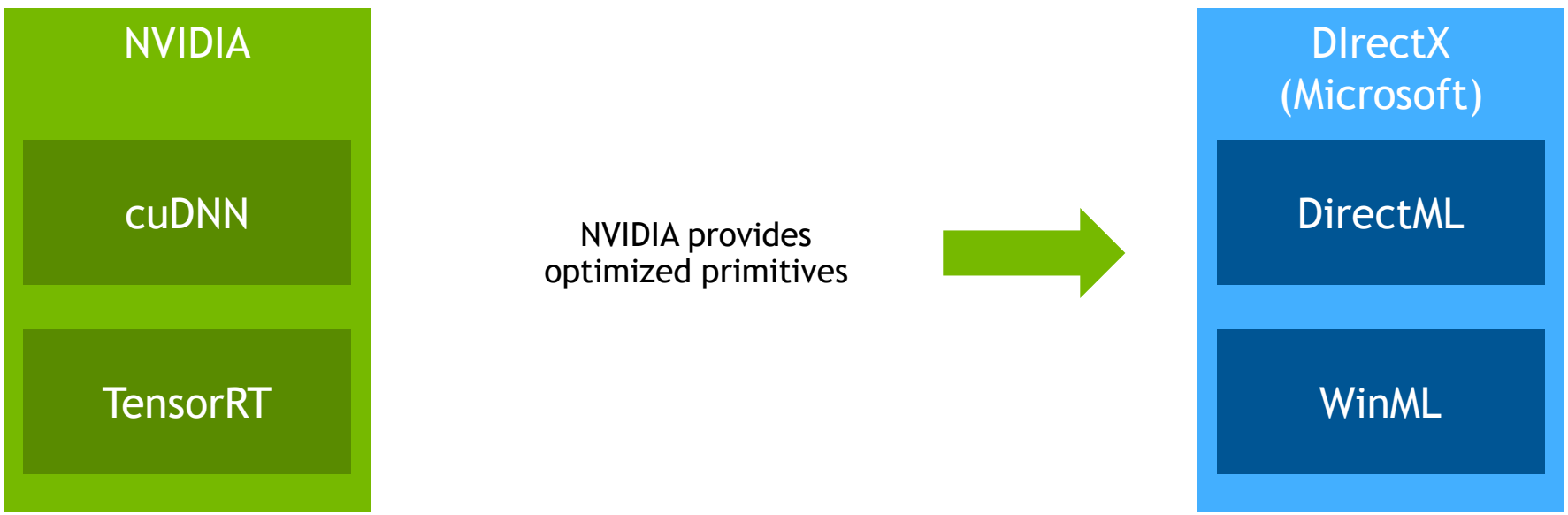
How do we deploy?

- Several solutions exist today.

Research To Production

How do we deploy?

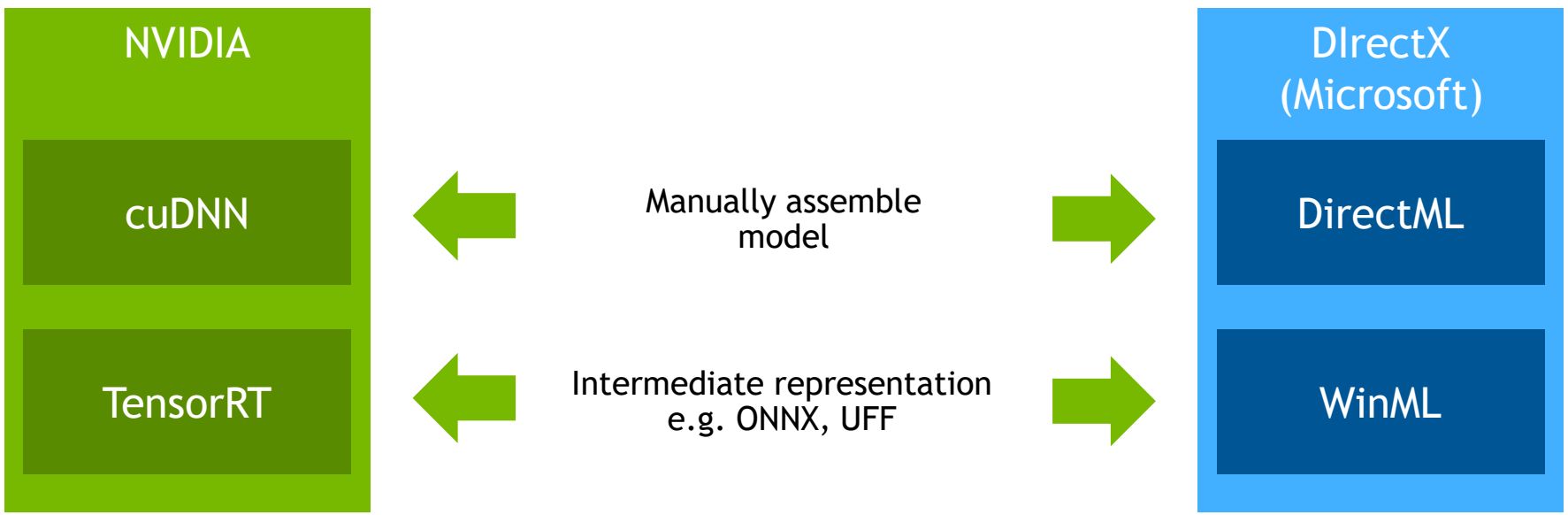
- Several solutions exist today.



Research To Production

How do we deploy?

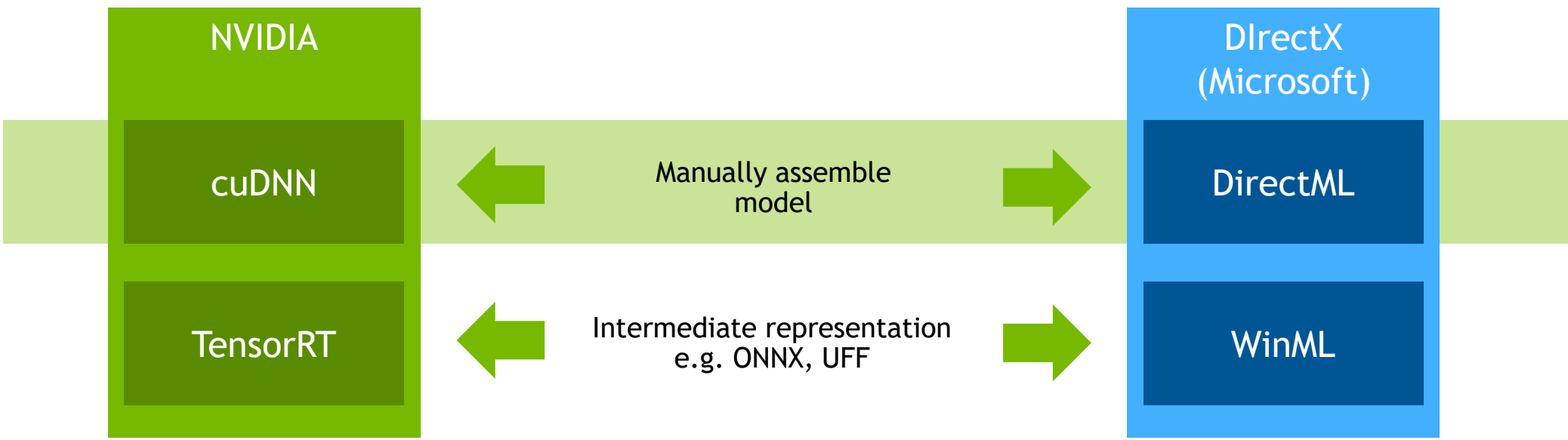
- Several solutions exist today.



Research To Production

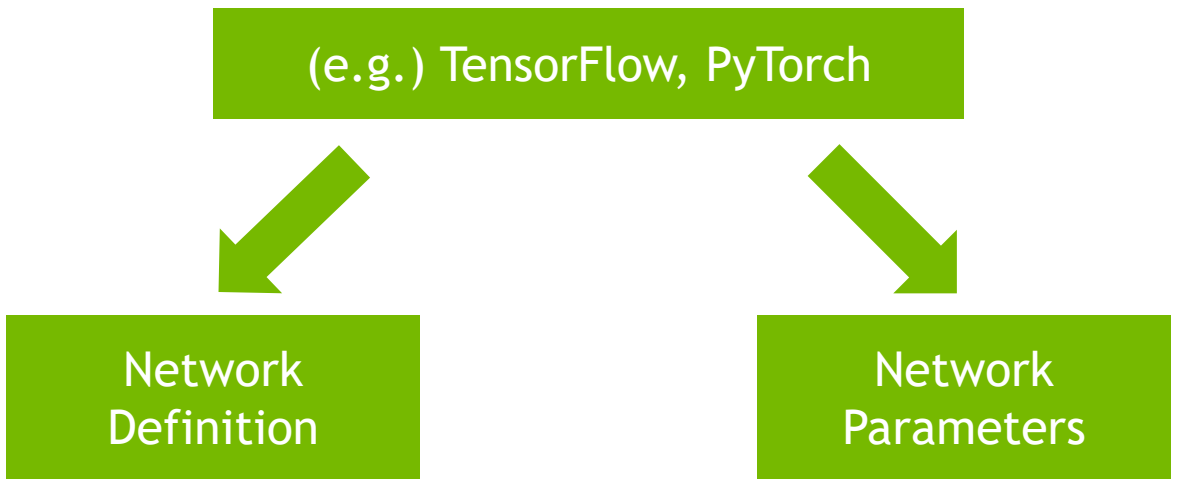
How do we deploy?

- Several solutions exist today.



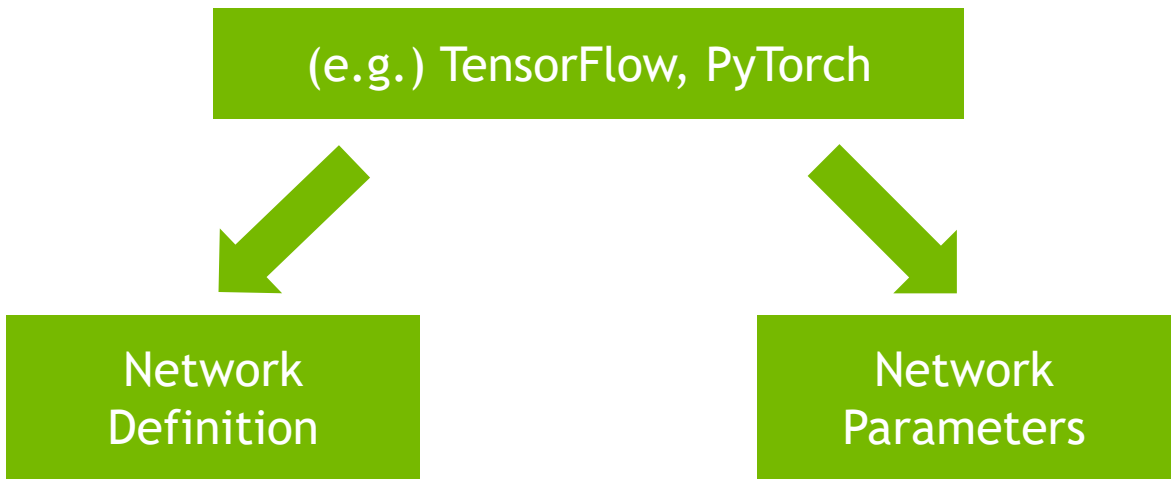
Research To Production

What do we need?



Research To Production

What do we need?



A few ways to do this.

Research To Production

C++ Parse The Protocol Buffers

- Most checkpoint/model file formats based on Protocol Buffers from Google
  - Check em out, they're awesome.
- Message format defined in the .proto file
- Compiled with the Protocol Buffer Compiler
- Manipulate the contents with the Protocol Buffer API
- Good tutorials for this at
  - https://developers.google.com/protocol-buffers/docs/cpptutorial

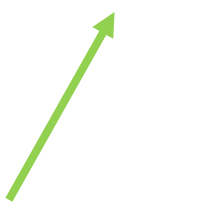
Research To Production

Write the params and arch straight from Python

PyTorch example (simplified)

Load the model in Python, open output file for writing

```
……
head_0_spade_0_mlp_shared_0,Weights,  176198144,2342400,     128,183,5,5
head_0_spade_0_mlp_shared_0,biases,   176197632,512,         1,1,1,128
……
```



Tensor Name

Offset,Size

Shape

Research To Production

Write the model architecture straight from Python

PyTorch example (simplified)

Load the model in Python, open output file for writing

```
input_file_path = <Path to Pytorch checkpoint>

ckpnt = torch.load(input_file_path,map_location="cpu")

out_path = "netG_params.txt"

        with open(out_path,"w") as f:
```

Research To Production

Write the model architecture straight from Python

PyTorch example (simplified)

Iterate the model, find the weights and biases

```
input_file_path = <Path to Pytorch checkpoint>

ckpnt = torch.load(input_file_path,map_location="cpu")

out_path = “netG_params.txt”

with open(out_path,"w") as f:

          for model_key in ckpnt :
                if model_key.find("weight") > 0 or model_key.find("bias") > 0:
                    cur_var = Variable(ckpnt[model_key])
                    var_size = cur_var.size()
                    size_len = len(var_size)
```

Research To Production

Write the model architecture straight from Python

PyTorch example (simplified)

Replace ‘.’ with ‘_’ (personal preference)

```
tensor_name = model_key.replace(".","_")
tensor_type = “”
if model_key.find("weight") > 0:
          tensor_type = "Weights"
          tensor_name = tensor_name.replace("_weight","")
if model_key.find("bias") > 0:
          tensor_type = "biases"
          tensor_name = tensor_name.replace("_bias","")

tensor_dims = ""
tensor_total_size = 1
```

Research To Production

Write the model architecture straight from Python

PyTorch example (simplified)

Record the tensor shape in a consistent manner

```
 if size_len < 4:
          size_len_delta = 4-size_len
          for s in range(size_len_delta):
                    tensor_dims += ",1"

          for s in range(size_len):
                    tensor_dims += ",{}".format(var_size[s])
                    tensor_total_size *= var_size[s]

tensor_total_file_size = tensor_total_size * 4

tensor_size_data = ",{},{}".format(tensor_offset,tensor_total_file_size)
```

Research To Production

Write the model architecture straight from Python

PyTorch example (simplified)

Write the name, offset, size and shape to the text file.

```
tensor_total_file_size = tensor_total_size * 4

tensor_size_data = ",{},{}".format(tensor_offset,tensor_total_file_size)
tensor_offset += tensor_total_file_size
tensor_name += ",{}".format(tensor_type)
f.write(tensor_name)
f.write(tensor_size_data)
f.write(tensor_dims)
f.write("\n")
```

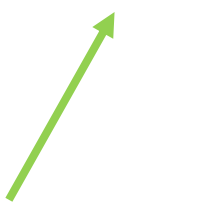
Research To Production

Write the params and arch straight from Python

PyTorch example (simplified)

Load the model in Python, open output file for writing

```
……
head_0_spade_0_mlp_shared_0,Weights,  176198144,2342400,     128,183,5,5
head_0_spade_0_mlp_shared_0,biases,   176197632,512,         1,1,1,128
……
```



Tensor Name

Offset,Size

Shape

Research To Production

Write the params and arch straight from Python

PyTorch example (simplified)

Load the model in Python, open output file for writing

```
……
head_0_spade_0_mlp_shared_0,Weights,  176198144,2342400,    128,183,5,5
head_0_spade_0_mlp_shared_0,biases,   176197632,512,        1,1,1,128
……
```

out,in,filter H/W

Tensor Name

Offset,Size

Shape

Research To Production

Write the model params straight from Python

PyTorch example (simplified)

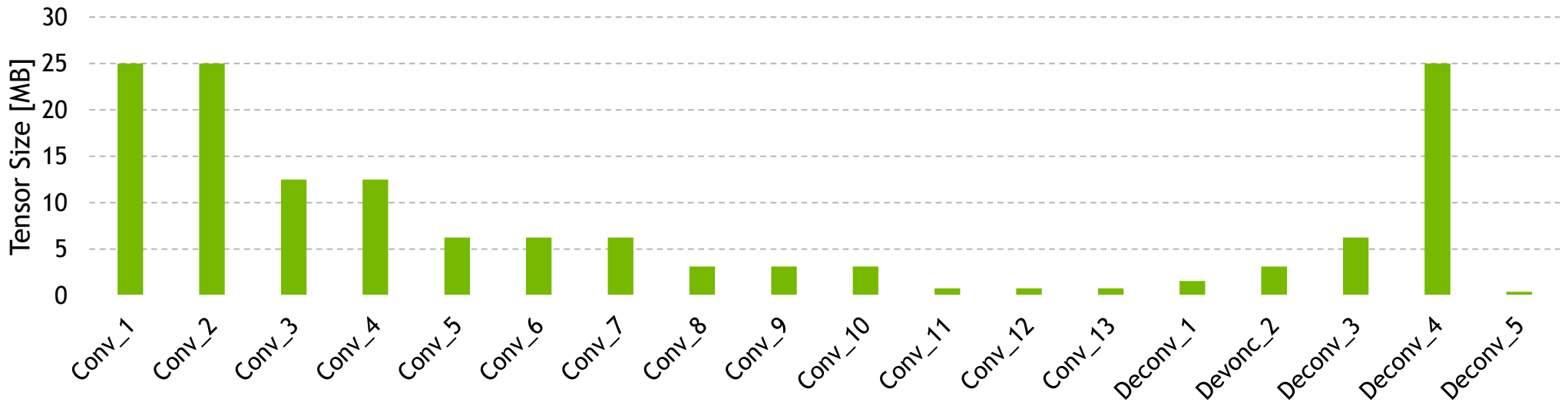
Similar loop as before but extract the weights from the .data member and write to a single file

```
(iterate model as before)

data = ckpt[model_key]
cur_var = Variable(data)
var_size = cur_var.size()
np_data = data.cpu().numpy()
f.write(np_data.tobytes())
```

TENSOR DEVICE MEMORY

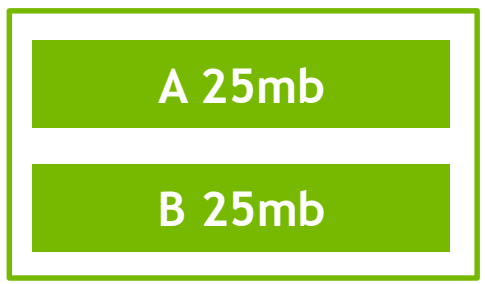
Deep Image Matting, Chris Hebert, GTC‘18



Combined size of all output tensors is **141.8 MB**

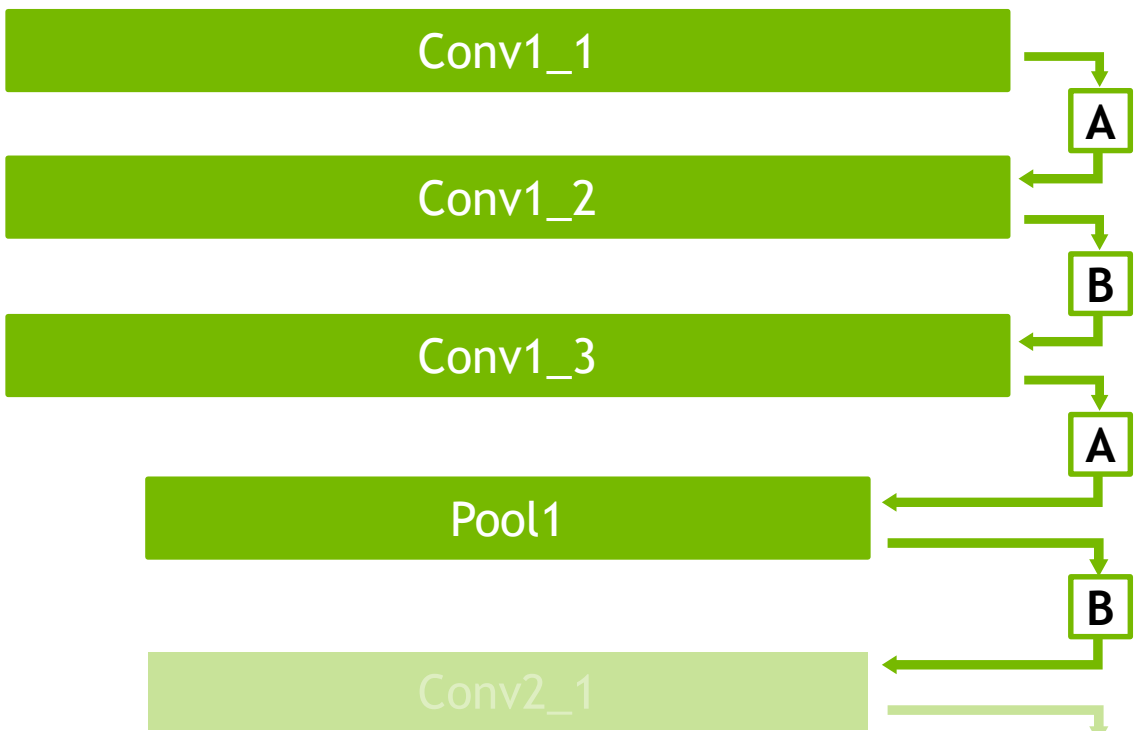
Only two tensors used at a time: input & output tensor

- Allocate two times the maximum tensor size: **50 MB**



MINIMIZING MEMORY FOOTPRINT

"Ping-Pong" Tensor Memory



Memory Pool
2x Largest Tensor

A 25mb

B 25mb

Doesn't work for cached tensors, e.g., skip links!

MINIMIZING MEMORY FOOTPRINT

Workspace Memory

Size of a convolution workspace varies, depending on multiple parameters:

- input and output tensor dimensions
- Precisions
- Convolution algorithm
- ...

But: workspace can be shared among layers

➢ **Allocate maximum workspace size!**

Research To Production

Scientist vs Engineer

- Scientists express their models in an algebraically correct manner.
  - They need to, that's how science shares research and advances.
  - But algebraically correct does not necessarily mean performant.
- Engineers need to identify when an algorithm can be restructured for performance.
  - That's our job.

Research To Production

Scientist vs Engineer

Example 1. Wx+b when you ONLY want the bias.

0.1762	0.0365	-0.0102	0.9191
-0.2388	-0.0010	0.7723	-0.5400
0.0012	-0.3333	-0.0001	0.0838
0.0019	-0.0095	0.0200	0.0211

X

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

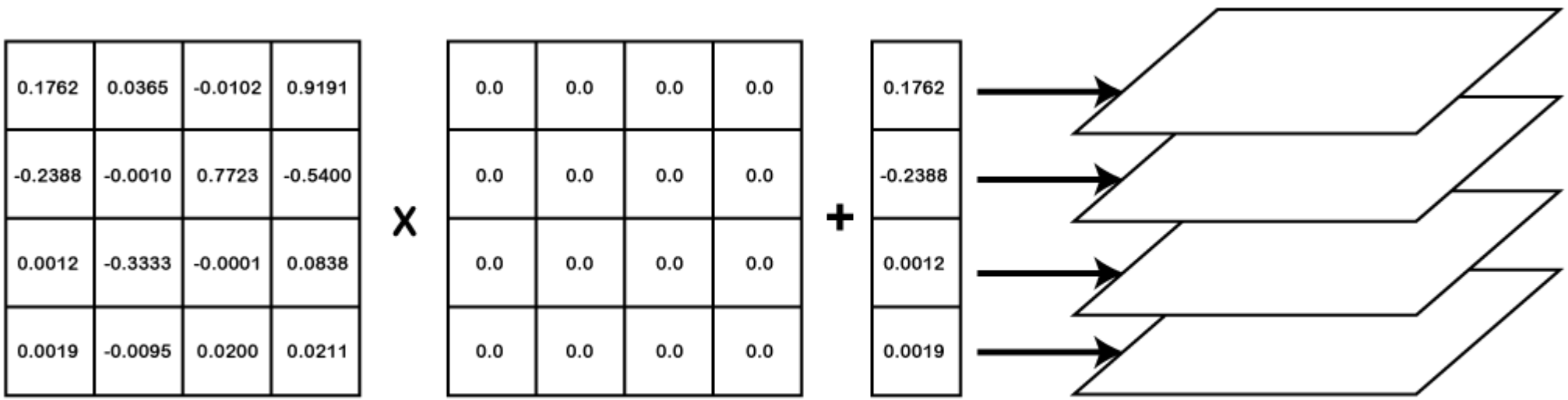
+

0.1762
-0.2388
0.0012
0.0019

Research To Production

Scientist vs Engineer

Example 1. Wx+b when you ONLY want the bias.



Research To Production

Scientist vs Engineer

Example 1. Wx+b when you ONLY want the bias.

In this particular case:

Use a bias ad op to a zero tensor if you have one

or

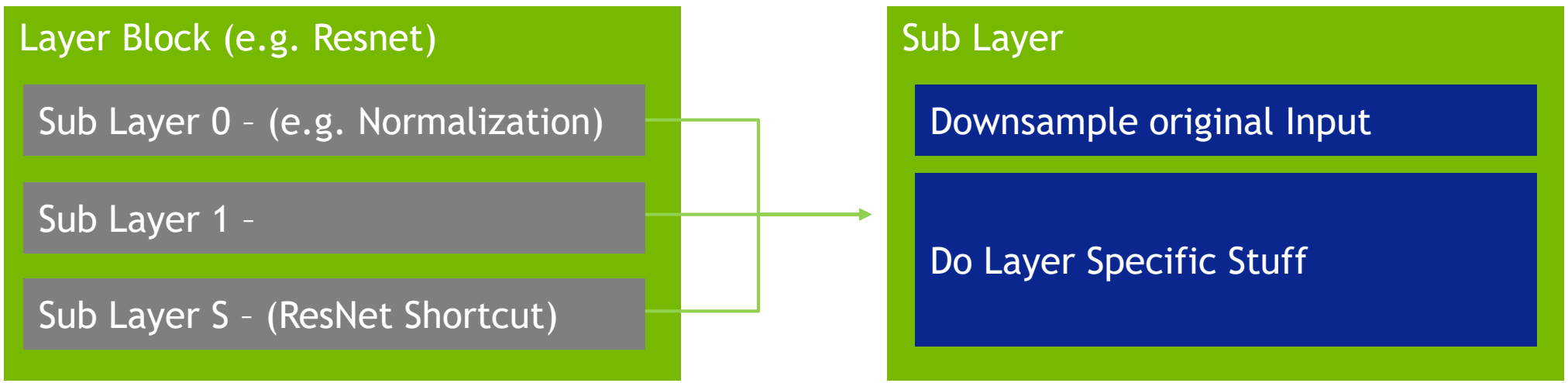
Write custom kernel to write the bias values.

And if possible fuse with previous and/or next step.

Research To Production

Scientist vs Engineer

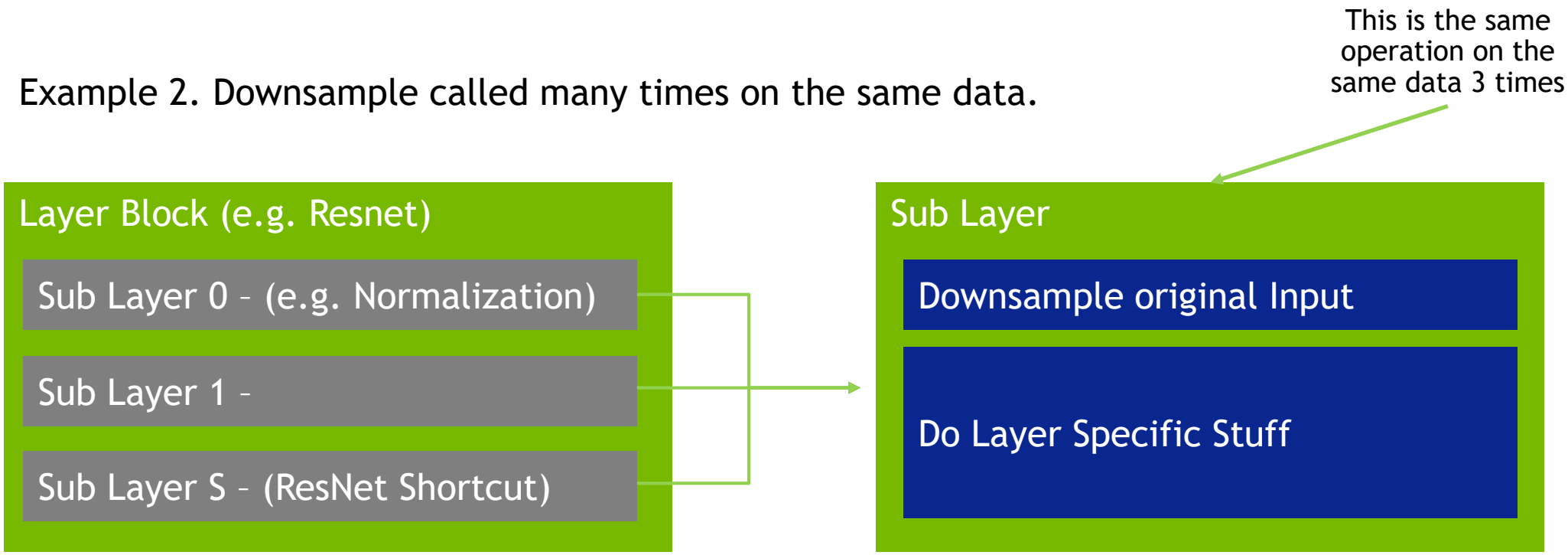
Example 2. Downsample called many times on the same data.



Research To Production

Scientist vs Engineer

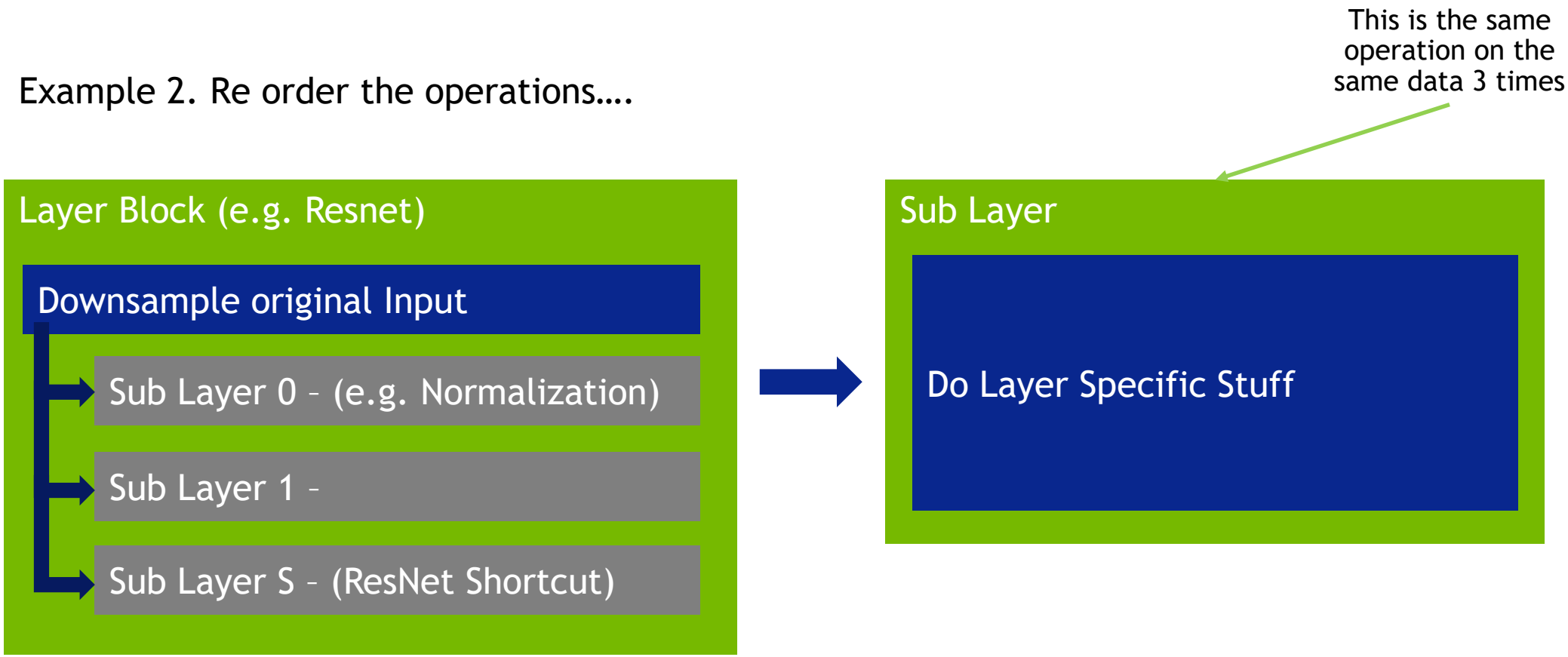
Example 2. Downsample called many times on the same data.



Research To Production

Scientist vs Engineer

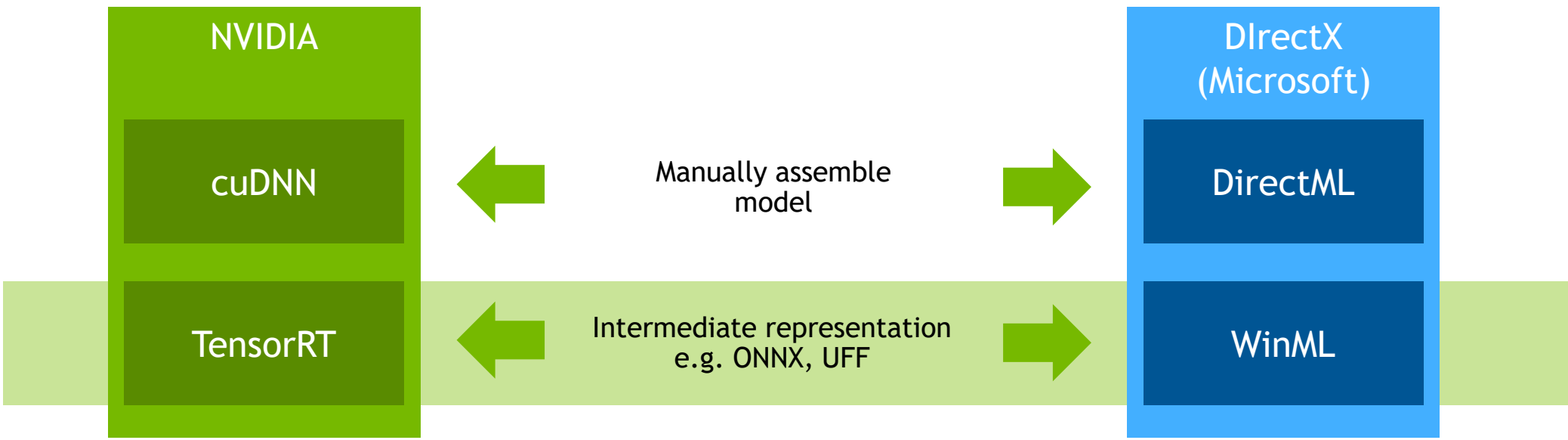
Example 2. Re order the operations….



Research To Production

How do we deploy?

- Several solutions exist today.



Research To Production

Using ONNX

- ONNX is great for a production workflow

Research To Production

Using ONNX

- ONNX is great for a production workflow
  - Designed to be seamless

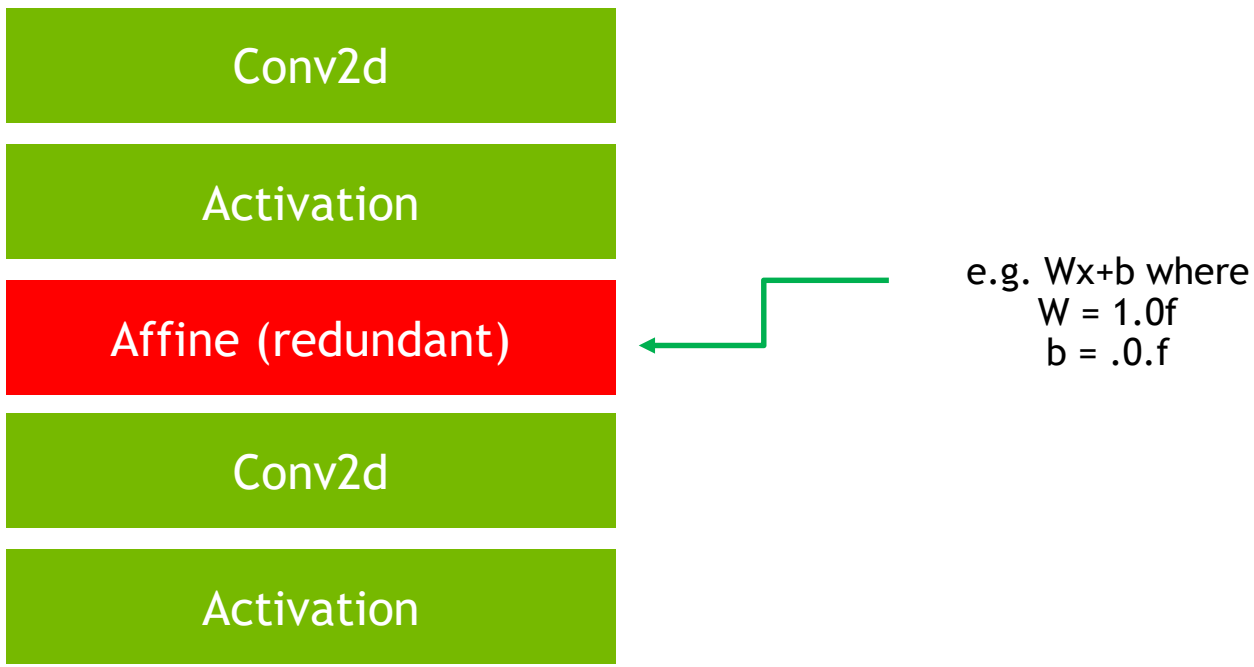
Research To Production

Parser vs Engineer

- Parsers and converters aren't always perfect
  - Though they are improving all the time
- Sometimes a resulting ONNX graph needs some closer inspection
- Numerous ways to perform surgery on ONNX graph
- ONNX graphs also based on protocol buffers
- Simple to create a custom ONNX parser
  - Analyze the graph
  - Make changes
  - Write out a new graph

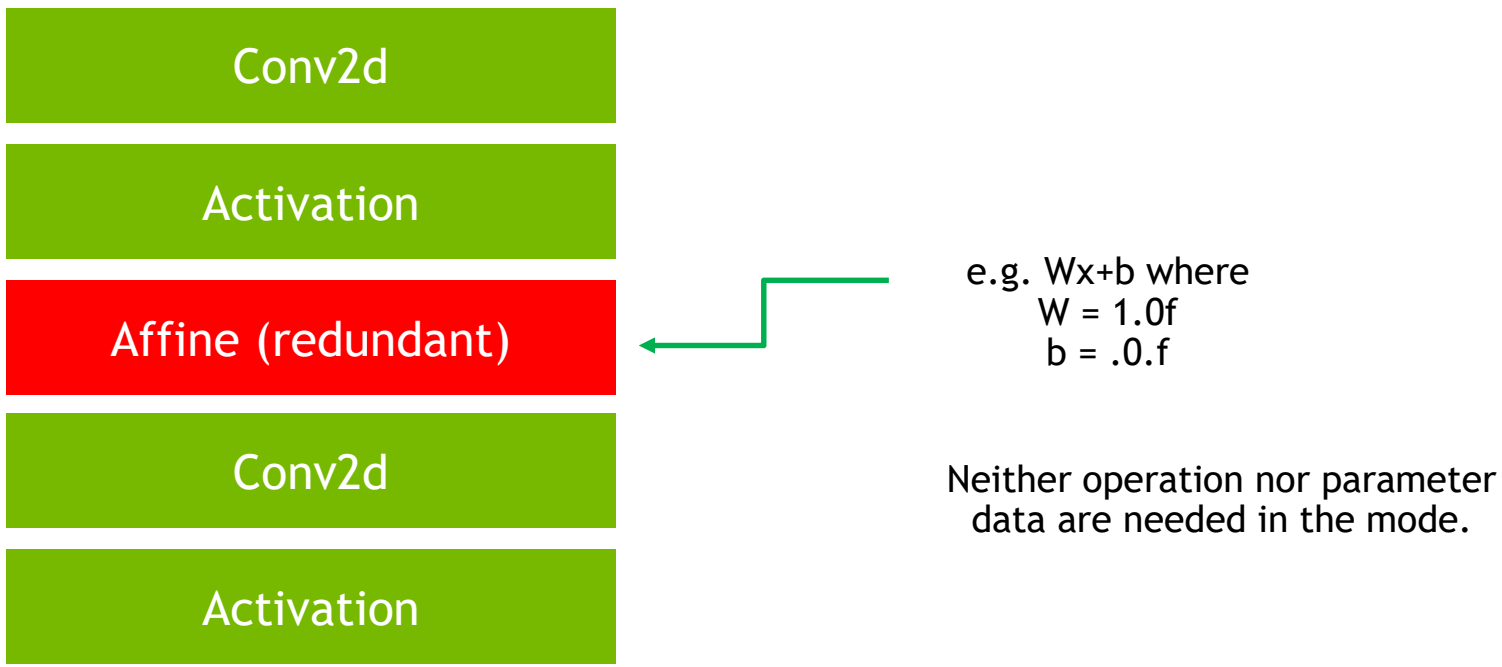
Research To Production

Parser vs Engineer



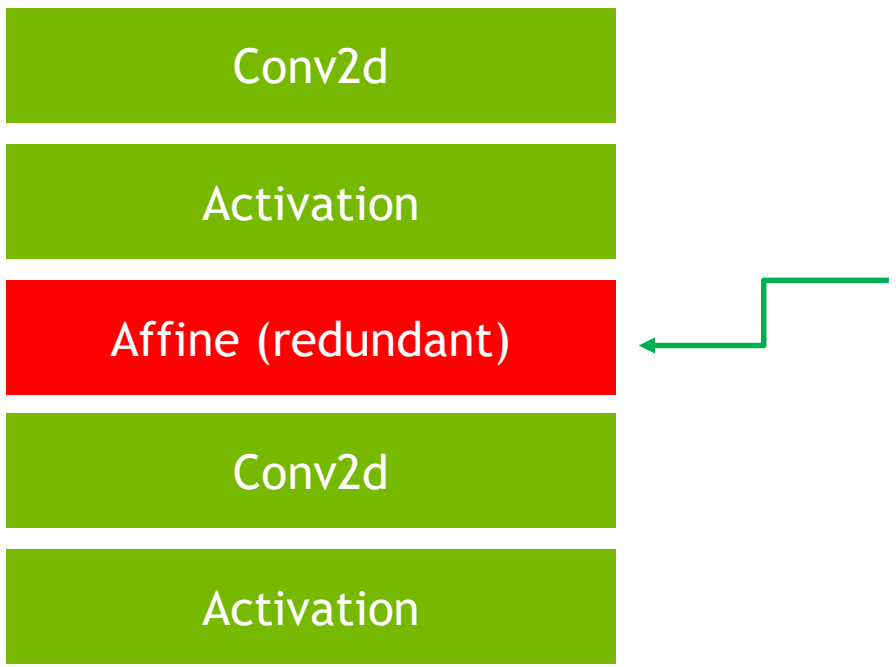
Research To Production

Parser vs Engineer



Research To Production

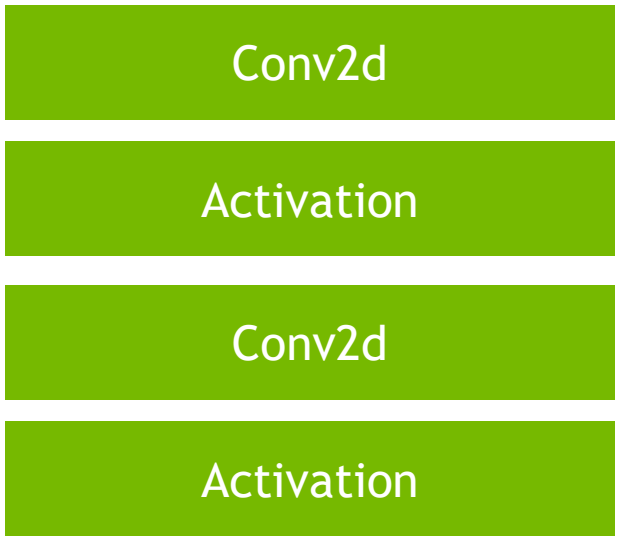
Parser vs Engineer



Manually edit the graph
Using e.g. protocol buffers API

Research To Production

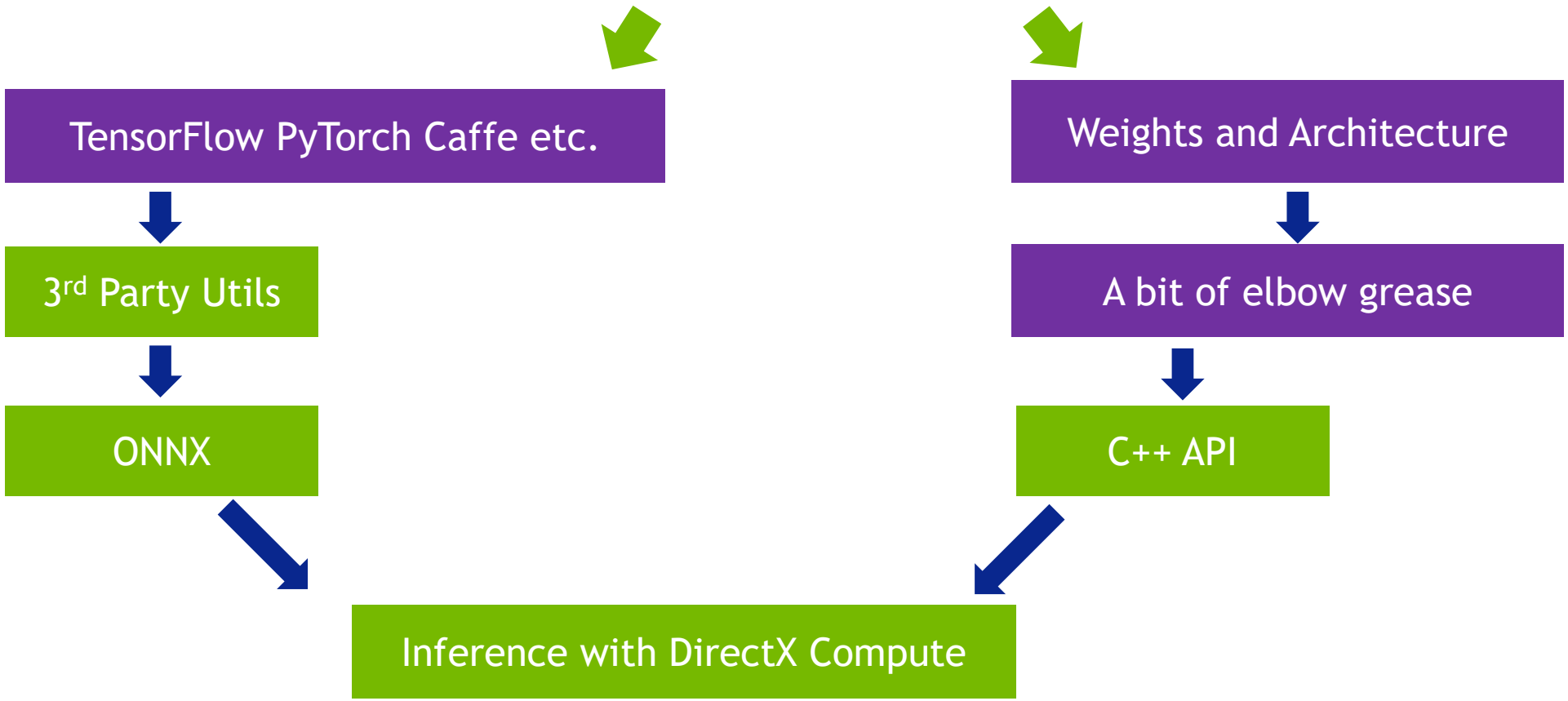
Parser vs Engineer



Now the model is smaller
And it’s going to run faster

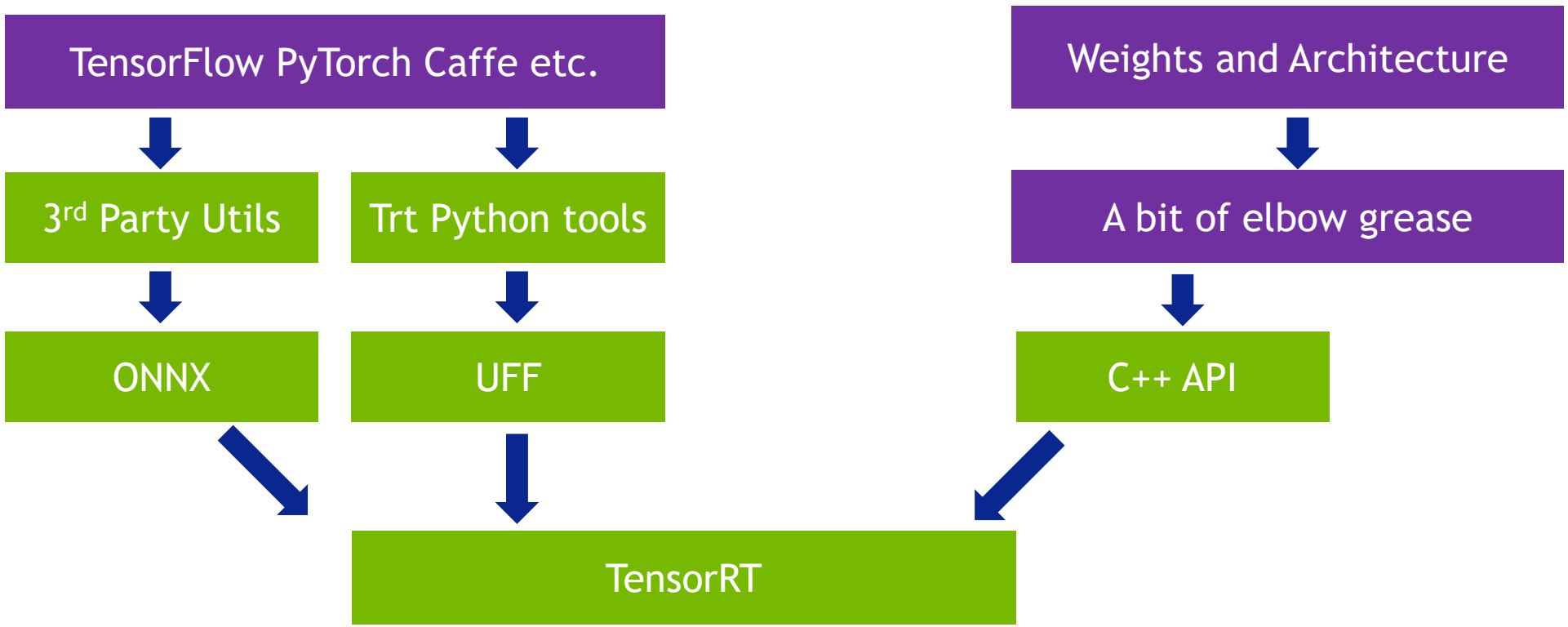
Research To Production

Porting to WinML and/or DirectML



Research To Production

Porting to TensorRT



Research To Production

UFF, ONNX or API …. Which to use….

- Most common architectures will import directly from TensorFlow/PyTorch etc
- Most common operations are already supported in TensorRT
- Convolution/Cross Correlation
- Activation
  - Sigmoid, Relu, Clipped Relu, TanH, ELU
- Batch Norm
  - Spatial, Spatial_persistent, Per Activation
- Pooling
  - Max, Average

Research To Production

UFF, ONNX or API …. Which to use….

- Sometimes it’s not that easy
- Sometimes some graph surgery is required.
  - Edit the graph to strip out e.g. pre/post processing at either end of the graph
- TensorRT provides a plug-in interface for custom layers
  - Name custom layers as per the incoming model (e.g. LeakyRelu)
  - From TrT 5.1 : The IPlugInV2 interface supports optimization.
- There is a simpler option

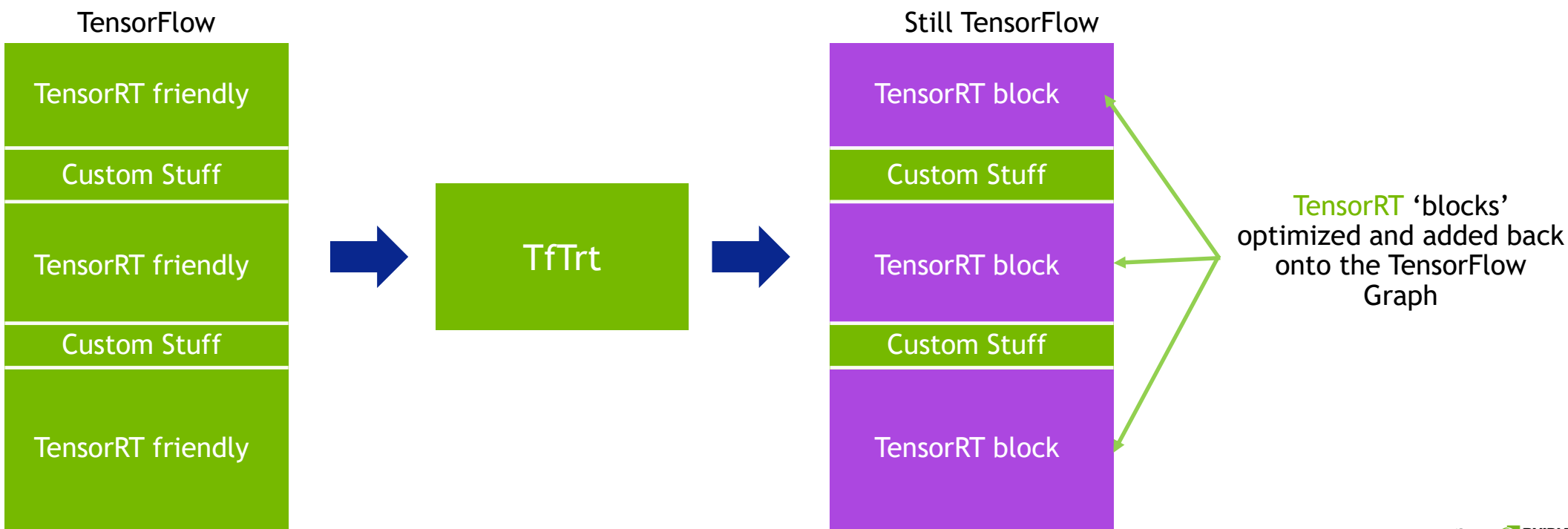
Research To Production

Porting to TensorRT Using TfTrt

- Converts TensorFlow graph into 1 or more TensorRT ‘blocks’
- Add’s these blocks back onto TensorFlow graph
- Inference of these blocks performed with TensorRT
- The rest use TensorFlow
- Worflow:
  - Load TensorFlow graph
  - Prepare for inference (freeze layers, convert variables to constants etc)
  - Call trt.create_inference_graph(input_graph_def, outputs, max_batch_size,max_workspace_size,precision_mode)

Research To Production

Porting to TensorRT Using TfTrt



Research To Production

Porting ‘Funky’ networks to TensorRT

Important takeaways from this

You don’t need generate a single monolithic graph with TensorRT

Generate graph snippets from TensorRT interleaved with custom CUDA

You can do this with the TensorRT API

Execute them in whatever sequence you need at run time.

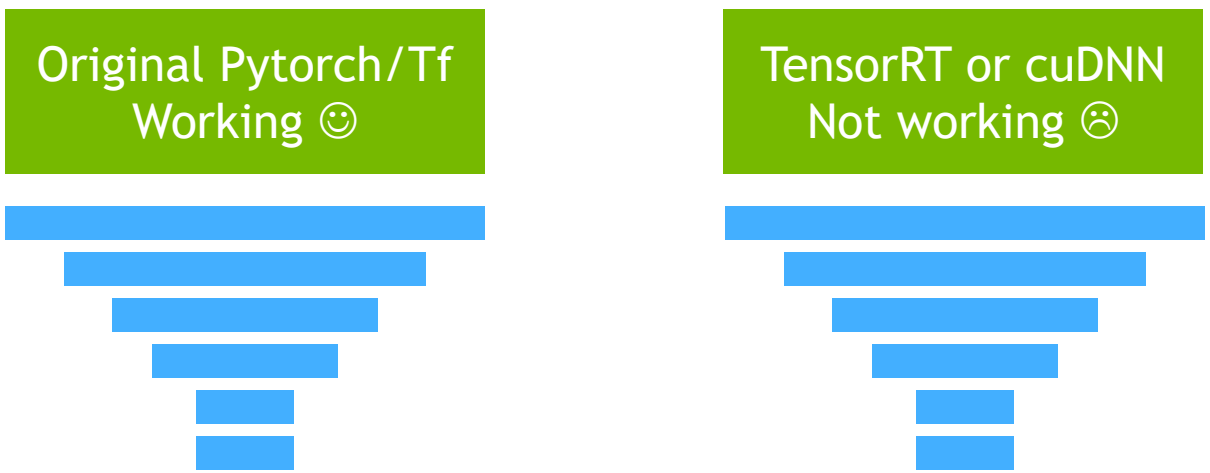
Allows you to create inference solutions with dynamic runtime behavior.

Keep all data on the GPU whenever possible.

Research To Production

When it doesn’t ……. Just work.

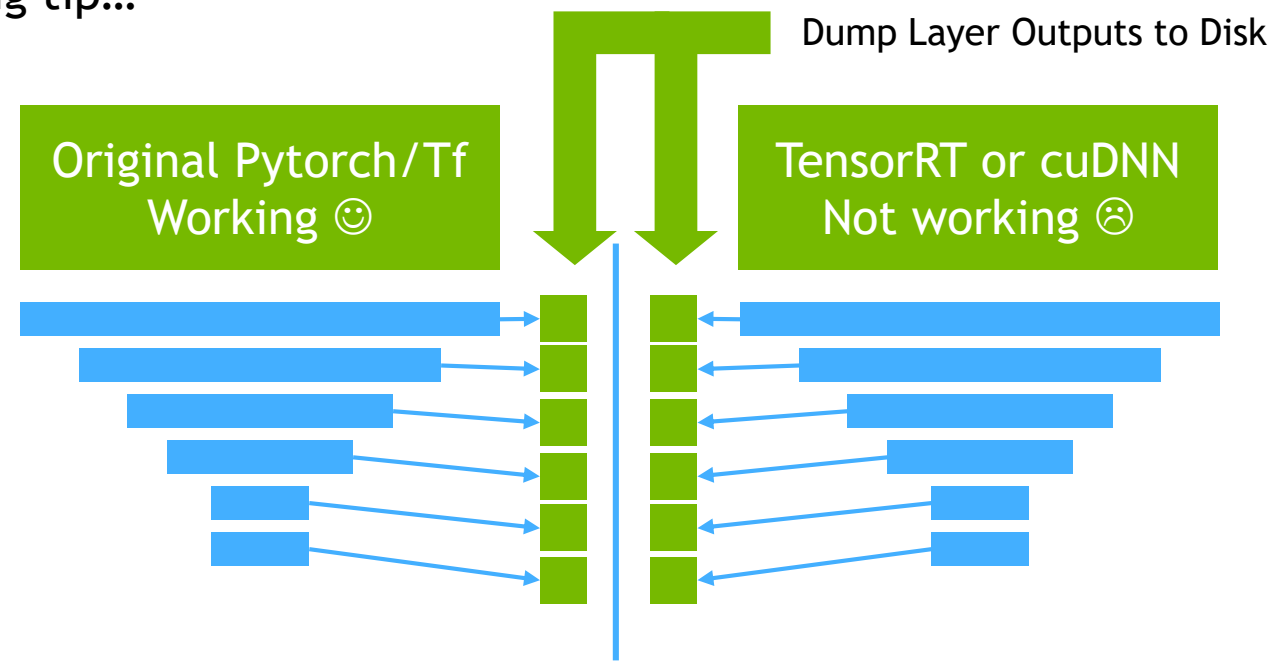
Here’s a debugging tip…



Research To Production

When it doesn't ……. Just work.

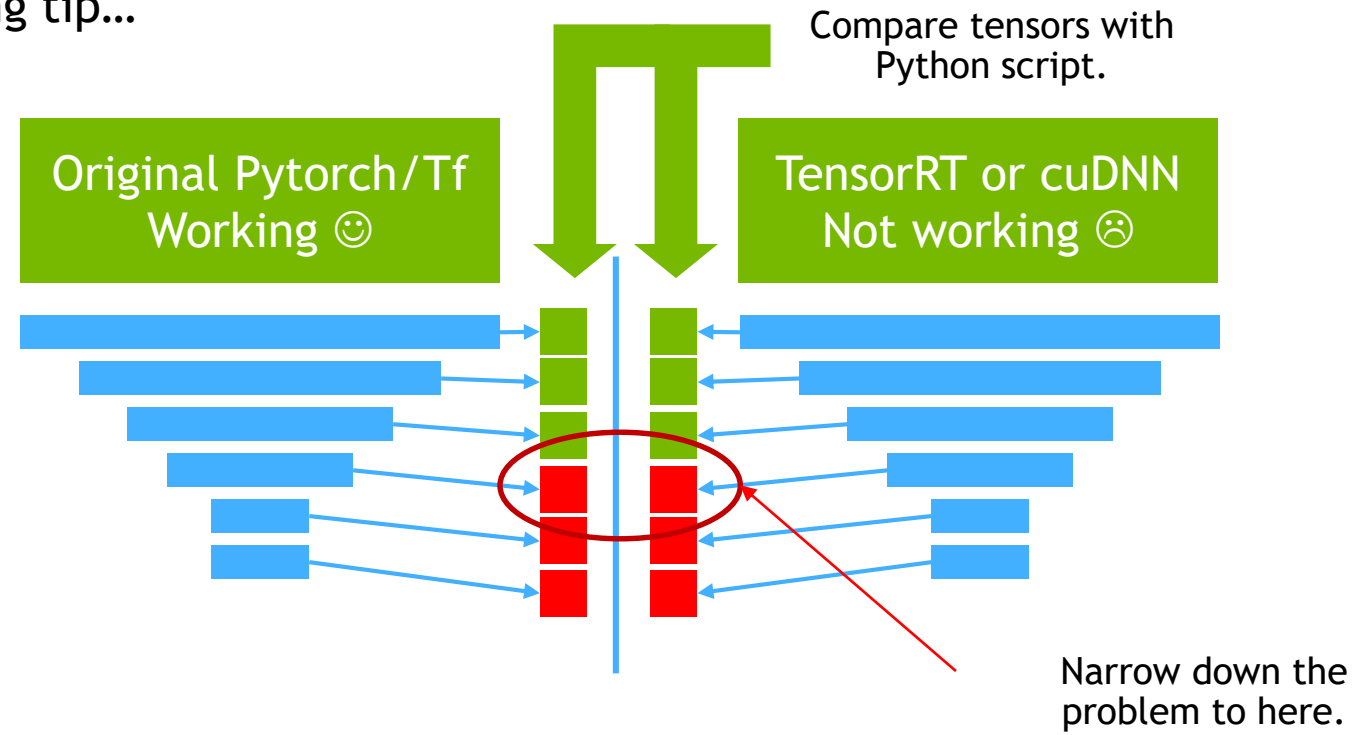
Here's a debugging tip…



Research To Production

When it doesn't ……. Just work.

Here's a debugging tip…



LOW PRECISION INFERENCE

In most cases FP16 / half provides more than adequate precision for image processing

As long as there is fairly low variance across the model

Volta and Turing have hardware for FAST fp16 – TRUE_HALF_CONFIG

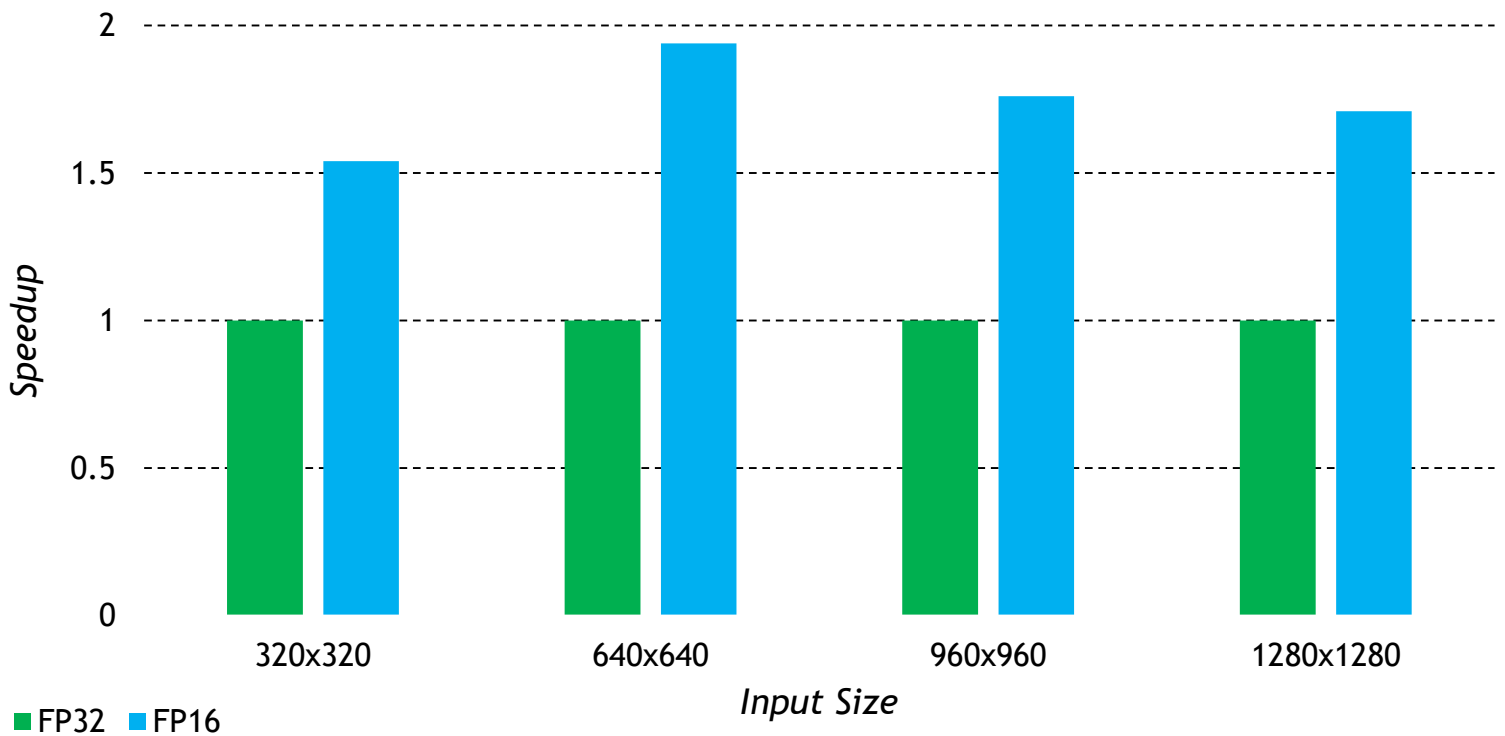
On Pascal and below, store in fp16 but process in fp32 – PSEUDO_HALF_CONFIG

Given an FP32 model, simply converting the weights to FP16 often retains decent quality

For best results ⇒ Retrain with FP16 precision

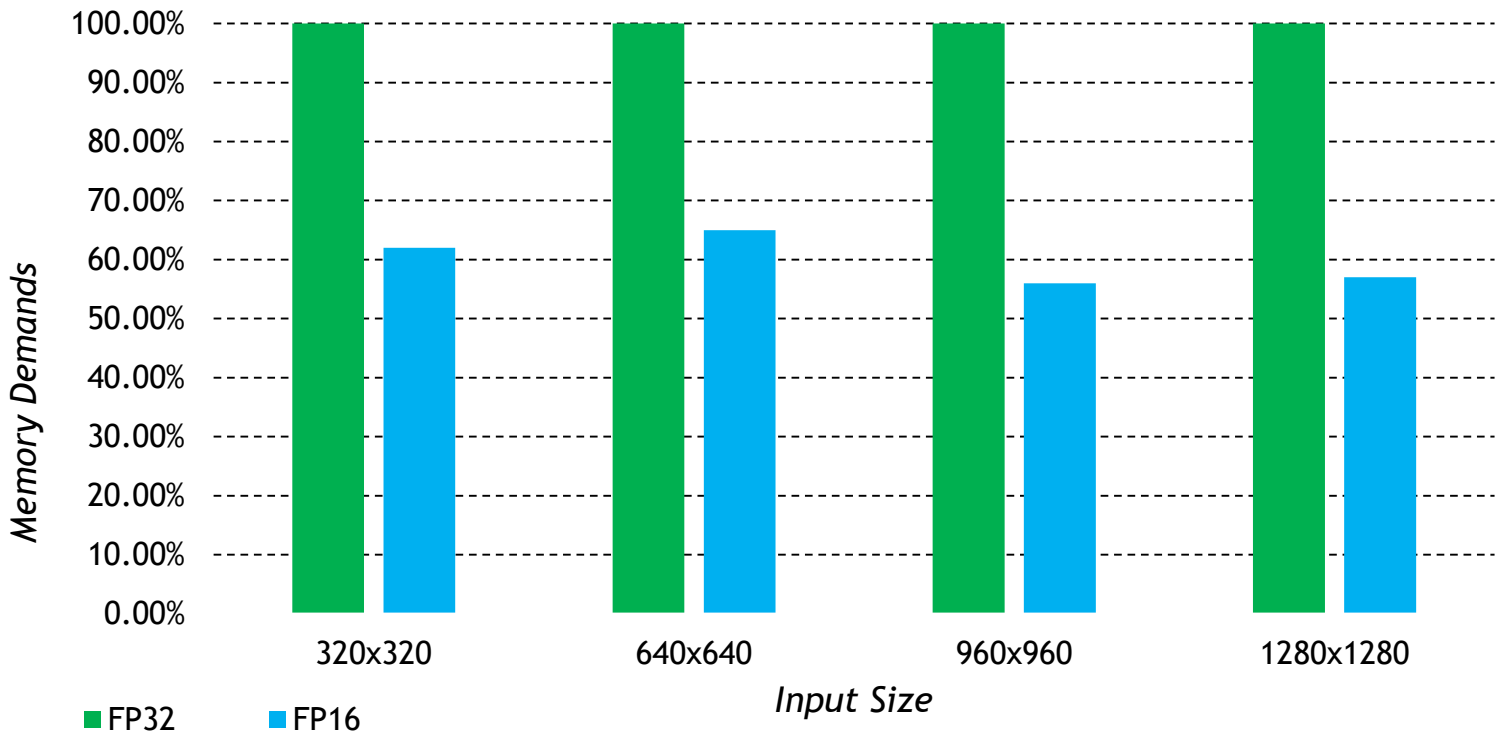
LOW PRECISION INFERENCE

Deep Image Matting, Chris Hebert, GTC‘18



LOW PRECISION INFERENCE

Deep Image Matting, Chris Hebert, GTC‘18

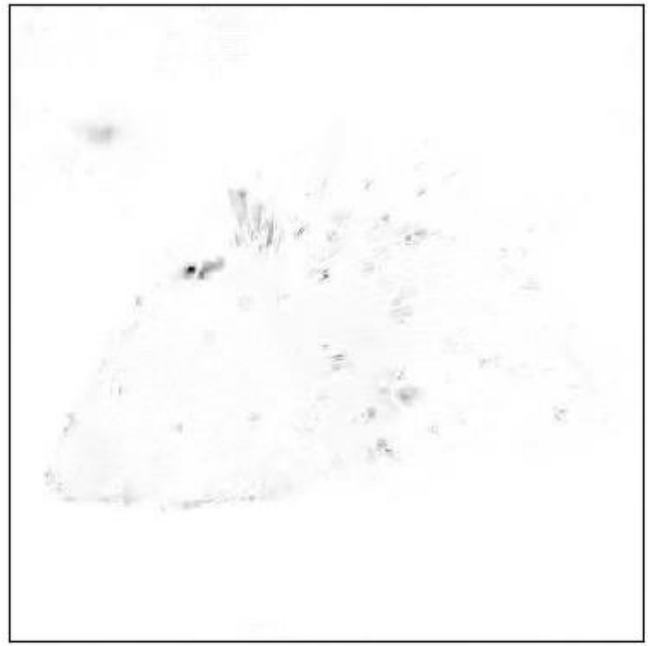


LOW PRECISION INFERENCE

Deep Image Matting

FP32

FP16



Abs difference x100

TENSOR CORES ON VOLTA & TURING

Tensor Cores perform FP16 matrix multiply accumulate (HMMA)

Turing also supports INT8 and INT4

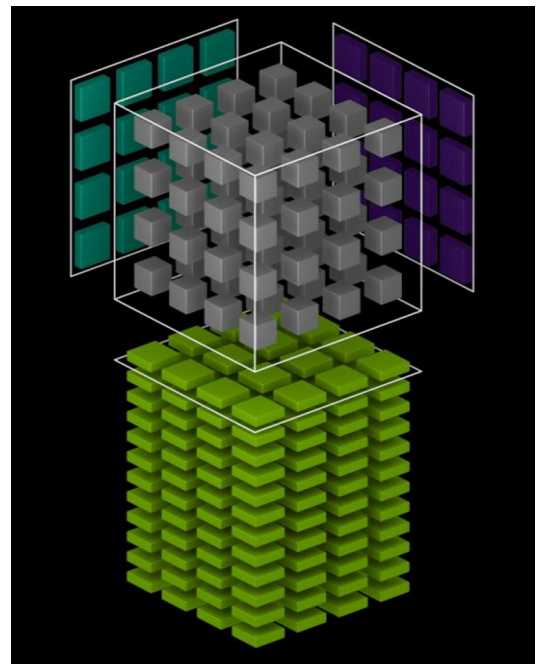
Only two algorithms supported:

`CUDNN_CONVOLUTION_FWD_ALGO_WINOGRAD_NONFUSED`

`CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM`

Number of Input and output channels must be multiple of eight!

Convolution math type must be set to `CUDNN_TENSOR_OP_MATH`



And it will be much MUCH faster!

TENSOR CORES ON VOLTA & TURING

8 Tensor Cores per sm.

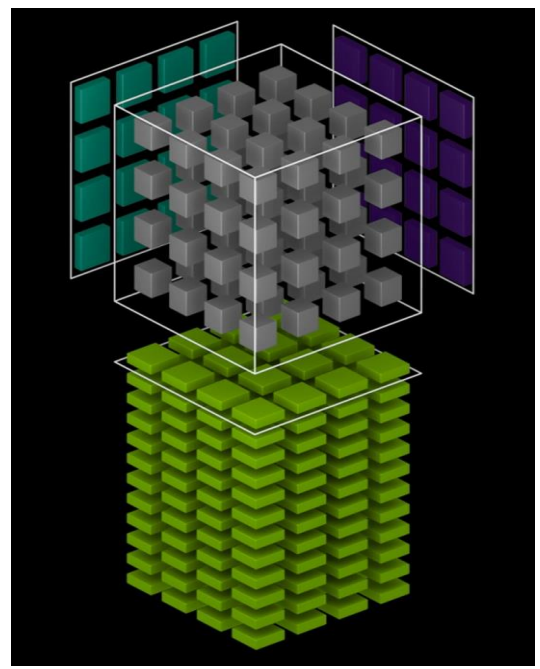
Each Tensor Core performs 64 FMA operations per clock

4x4x4 sub matrices

Each warp utilizes multiple Tensor Cores to accumulate results

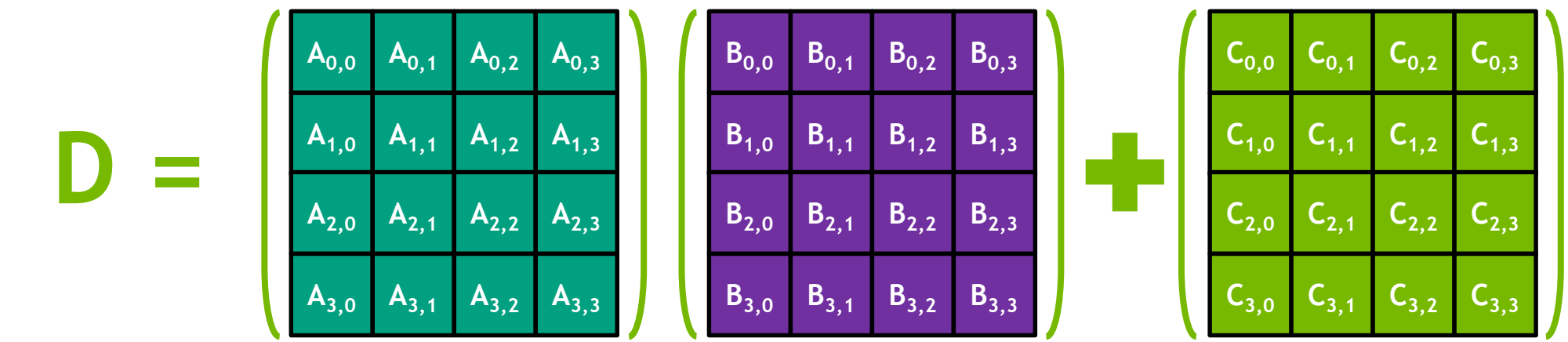
Hardware operates in FP16, cuDNN 7.3 onwards transparently converts from/to FP32:

`CUDNN_TENSOR_OP_MATH_ALLOW_CONVERSION`



TENSOR CORES ON VOLTA & TURING

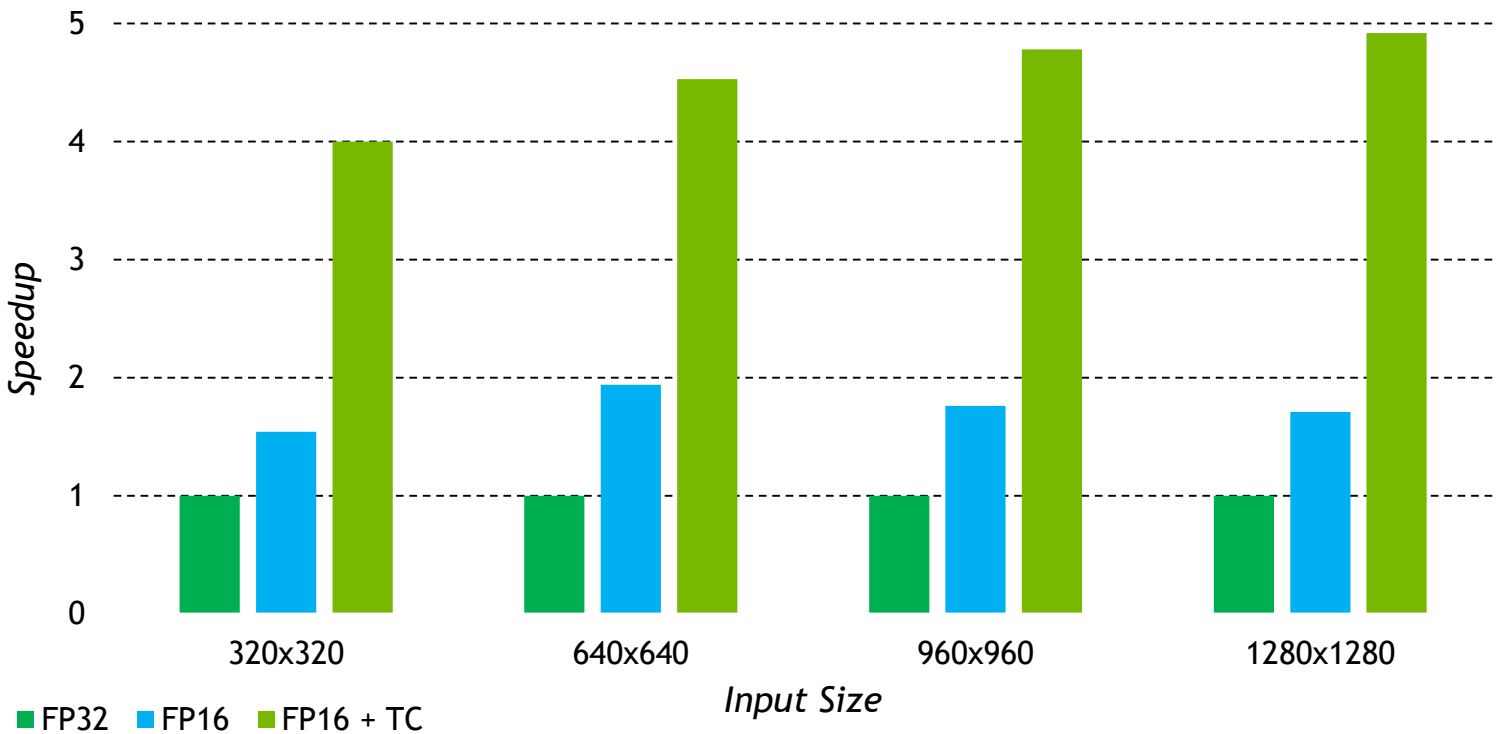
Mixed Precision Matrix Math
4x4 matrices



$$D = AB + C$$

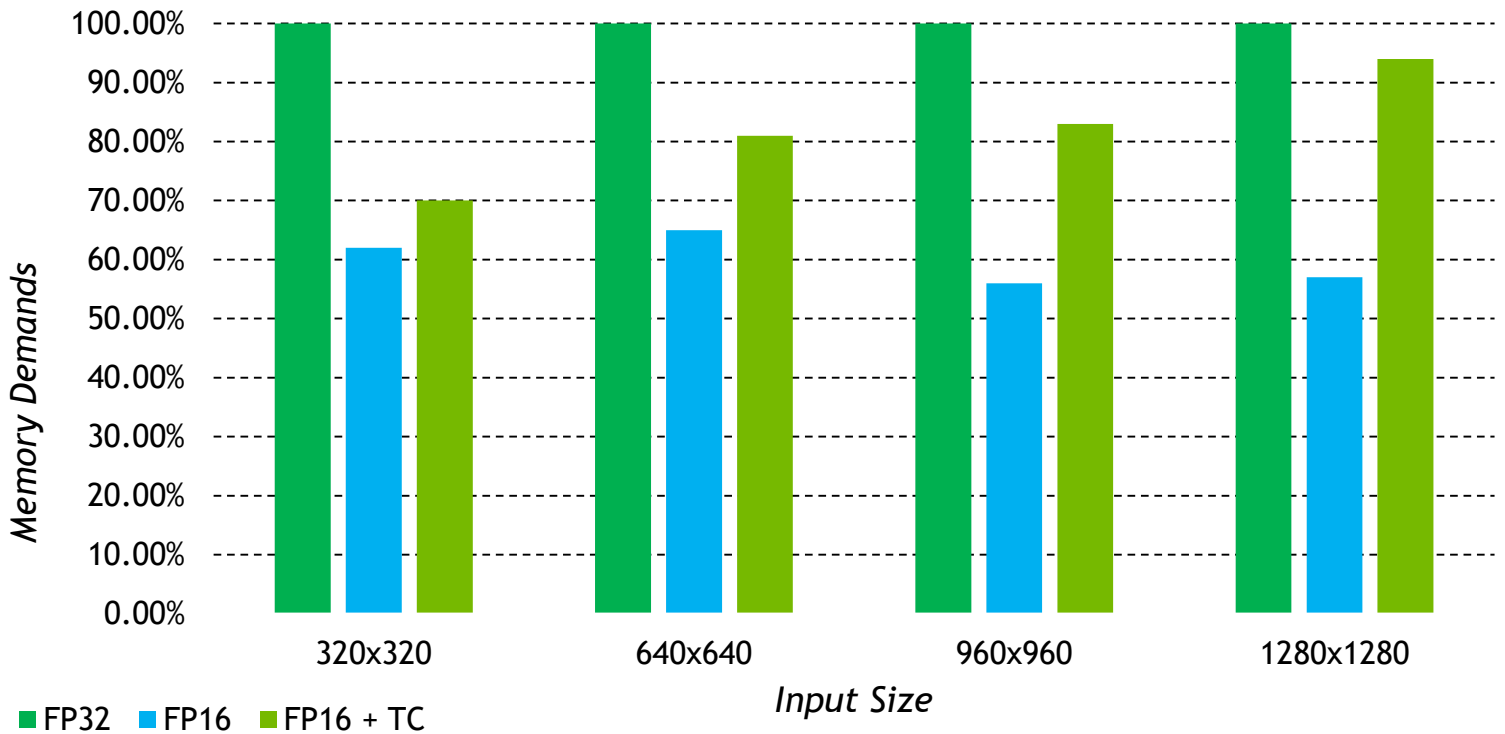
LOW PRECISION INFERENCE

Deep Image Matting, Chris Hebert, GTC‘18



LOW PRECISION INFERENCE

Deep Image Matting, Chris Hebert, GTC‘18



TENSOR CORES ON VOLTA AND TURING

NCHW vs NHWC

Input Tensor Size	Output Tensor Size	Filter Size	NCHW	NHWC
32 x 32 x 64	16 x 16 x 128	3 x 3	0.05 ms	0.04 ms
128 x 128 x 128	128 x 128 x 128	3 x 3	0.11 ms	0.08 ms
512 x 512 x 32	256 x 256 x 64	5 x 5	0.25 ms	0.15 ms
1920 x 1080 x 8	1920 x 1080 x 32	5 x 5	3.00 ms	2.31 ms
16 x 16 x 128	8 x 8 x 256	7 x 7	0.26 ms	0.11 ms
128 x 128 x 128	128 x 128 x 128	7 x 7	0.37 ms	0.34 ms
800 x 800 x 8	400 x 400 x 8	9 x 9	1.20 ms	2.53 ms

Convolution algorithm selected using cudnnFindConvolutionForwardAlgorithm(...)

AGENDA

Introduction to cuDNN

cuDNN Best Practices:

- Memory Management Done Right
- Choosing the Right Convolution Algorithm & Tensor Layout
- Tensor Cores: Low Precision Inference at Speed of Light
- The Last 10 Percent ...

From Research to Production: It just works ... or not?!

Summary/Demo

SUMMARY

Common DL frameworks often far from optimized for inference on GPUs

- Use cuDNN (or TensorRT) if you care about performance & memory!

Memory Management matters!

Lower your precision if possible!

Use hardware-specific optimizations, e.g. Tensor Cores on Volta & Turing!

You can never profile too much!


NVIDIA