CS 498: Machine Learning System
Spring 2025

Minjia Zhang

The Grainger College of Engineering

Today

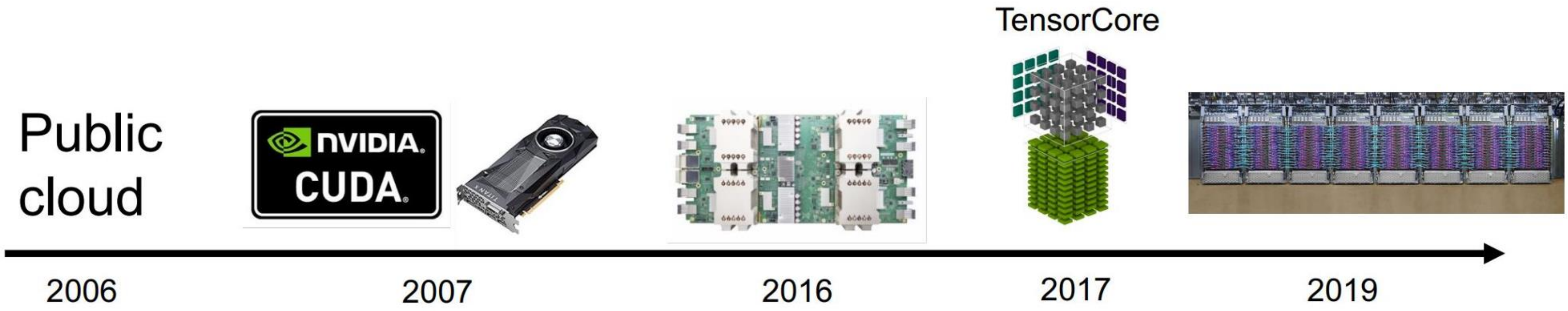
Mixed Precision Training

Outline

- **Mixed Precision Hardware**
- **What is Mixed Precision Training?**
- **Considerations for Mixed Precision**
- **Mixed Precision Software**

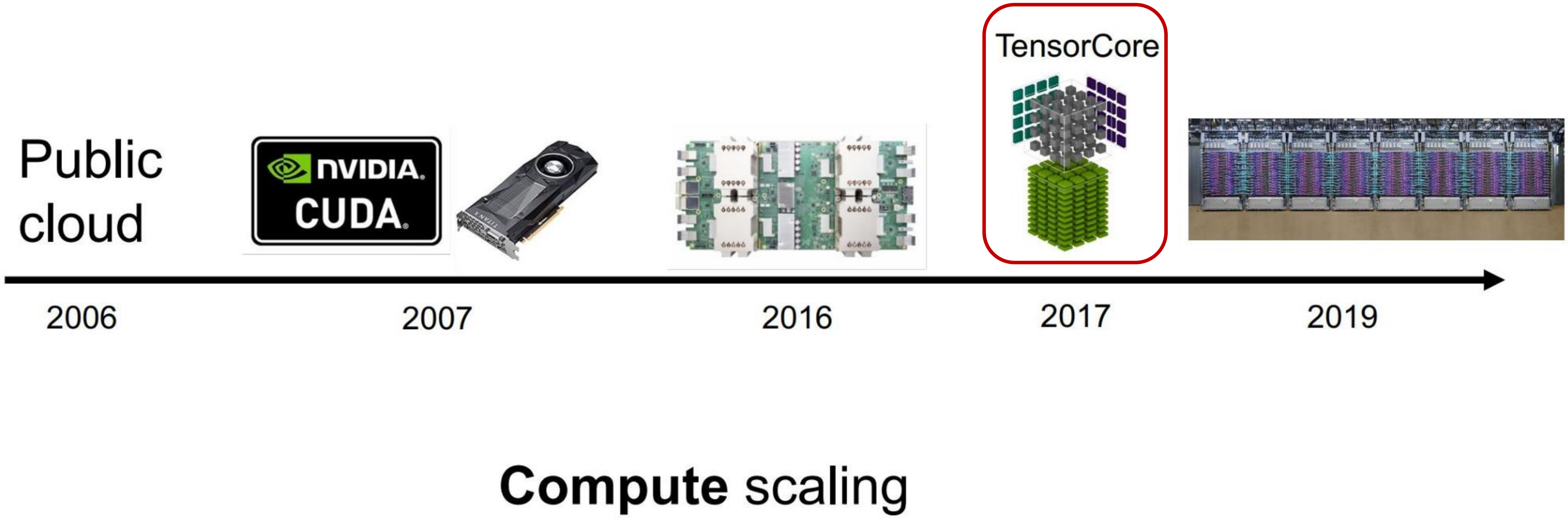
Mixed Precision Hardware

2006 – Now: Compute and Scaling



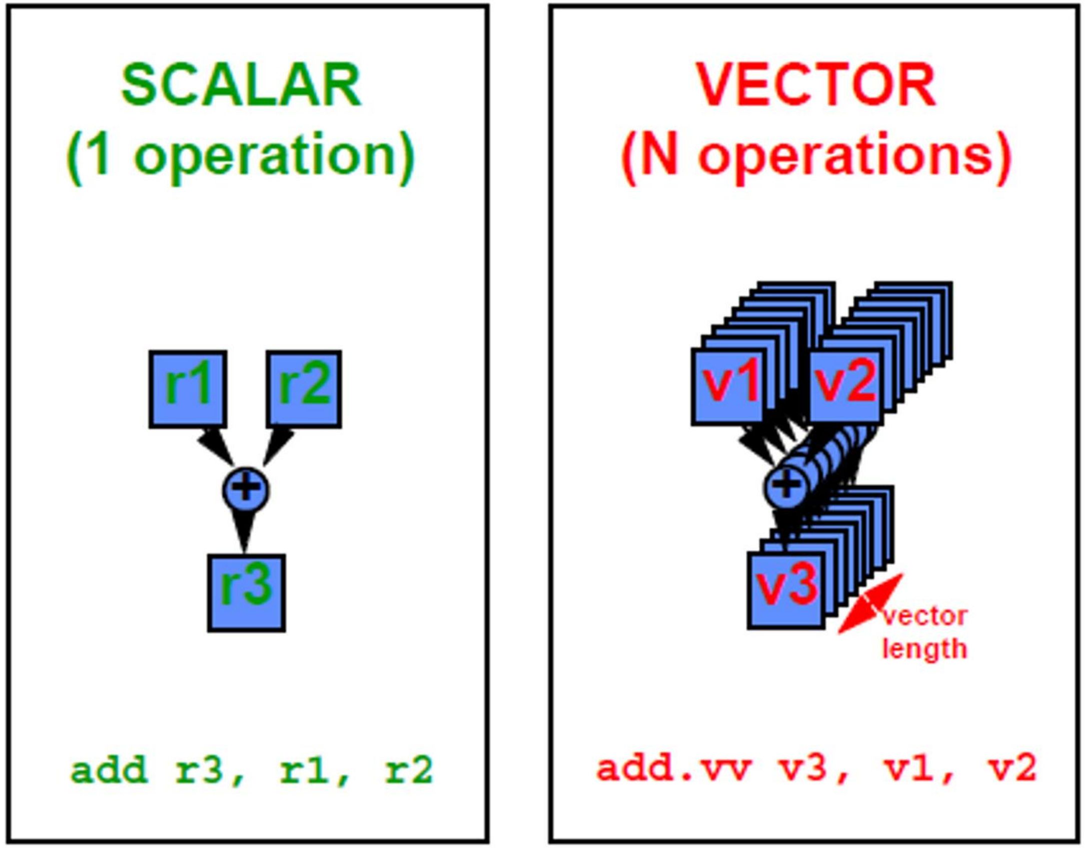
Compute scaling

2006 – Now: Compute and Scaling



Vector Processing Unit

- A processing unit that operates on an entire vector in one instruction (SIMD)
- The operand to the instructions are vectors instead of a scalar
- Work done in parallel



What are Tensor Cores? Matrix Multiplication Units

VOLTA ARCHITECTURE

What are Tensor Cores? Matrix Multiplication Units

- **Tensor cores are:**
  - Built to accelerate deep learning
  - Special hardware execution units
  - Execute matrix multiply operations (fused multiply-add on small matrices) in one instruction cycle

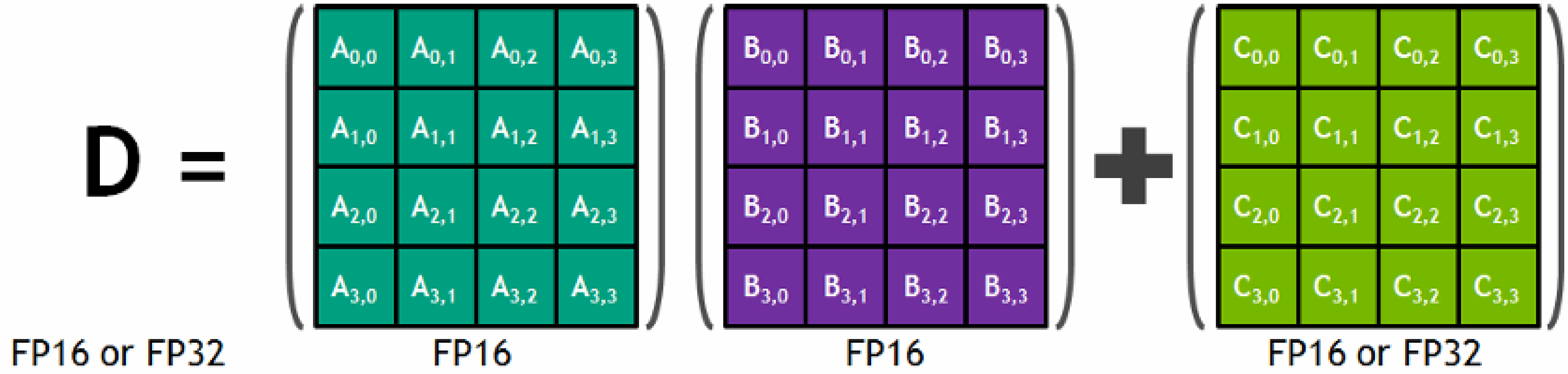
What are Tensor Cores? Matrix Multiplication Units

- **Tensor cores are:**
  - Built to accelerate deep learning
  - Special hardware execution units
  - Execute matrix multiply operations (fused multiply-add on small matrices) in one instruction cycle

- **Different flavors**
  - (V100) Volta Tensor Cores FP16
  - (T4) Turing Tensor Cores FP16/INT8/INT4
  - …

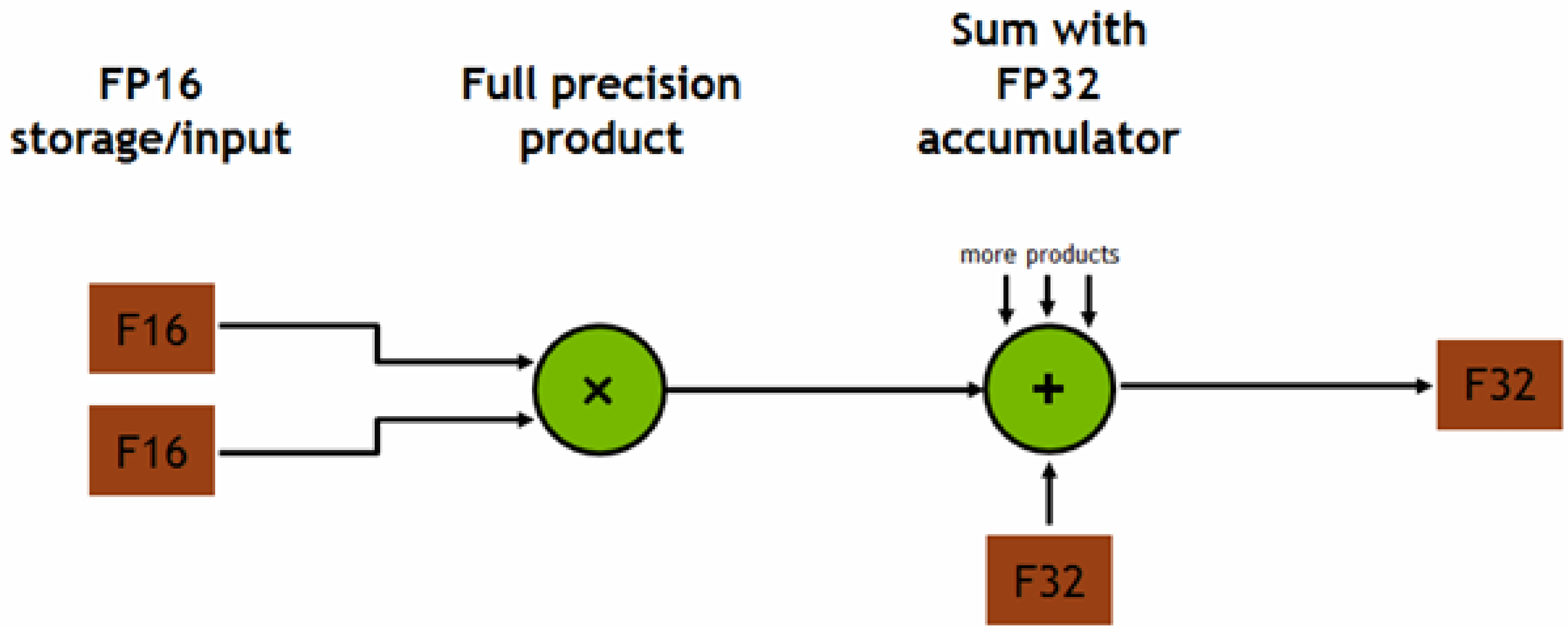
VOLTA ARCHITECTURE

Tensor Cores: Mixed Precision Matrix Math on 4x4 Matrices



$$D = AB + C$$

Internals of Tensor Cores (Volta)



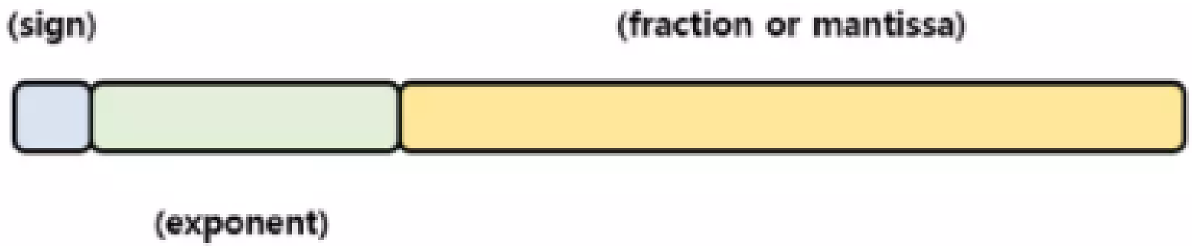
- **Accelerate matrix multiplications and convolutions**
- **Tensor core optimized libraries: cuDNN, cuBLAS, CUTLASS**

https://devblogs.nvidia.com/programming-tensor-cores-cuda-9/

What is Mixed Precision Training?

IEEE 754 Floating Point Representation Recap

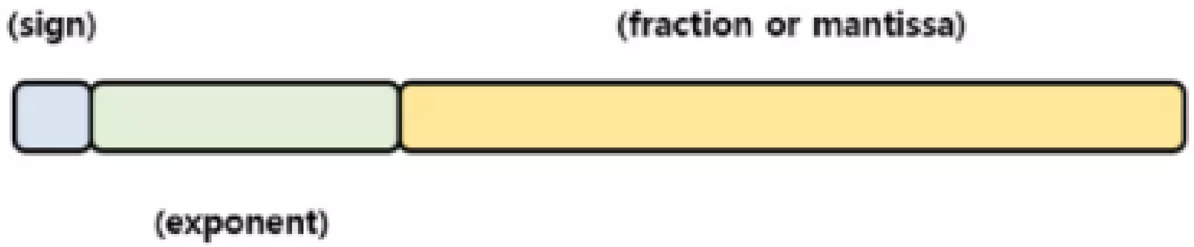
- Number can be represented by $(-1)^S * (1.M) * 2^{(E - Bias)}$



Half Precision (Floating Point 16)	1 bit	5 bit	10 bit
Single Precision (Floating Point 32)	1 bit	8 bit	23 bit
Double Precision (Floating Point 64)	1 bit	11 bit	52 bit
Quadruple Precision (Floating Point 128)	1 bit	15 bit	113 bit

IEEE 754 Floating Point Representation Recap

- Number can be represented by $(-1)^S * (1.M) * 2^{(E - Bias)}$

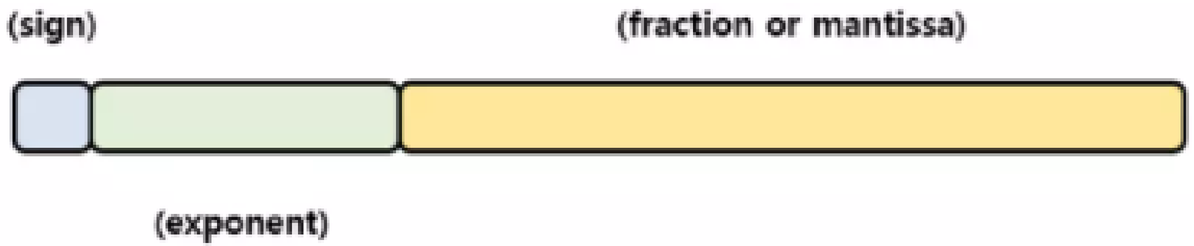


0 10000001 10100000000000000000000
Assume bias is 127

Half Precision (Floating Point 16)	1 bit	5 bit	10 bit
Single Precision (Floating Point 32)	1 bit	8 bit	23 bit
Double Precision (Floating Point 64)	1 bit	11 bit	52 bit
Quadruple Precision (Floating Point 128)	1 bit	15 bit	113 bit

IEEE 754 Floating Point Representation Recap

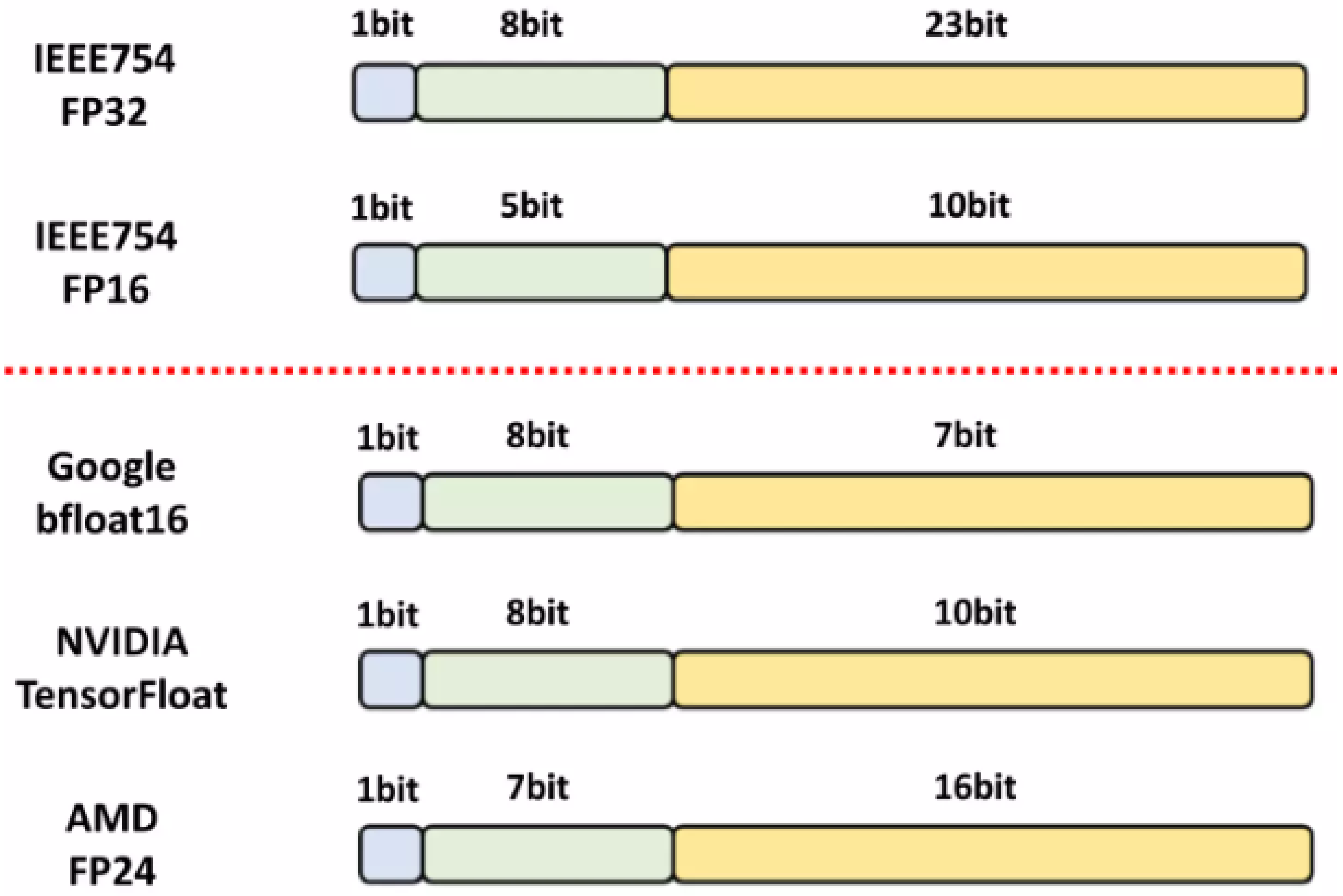
- Number can be represented by $(-1)^S * (1.M) * 2^{(E-Bias)}$



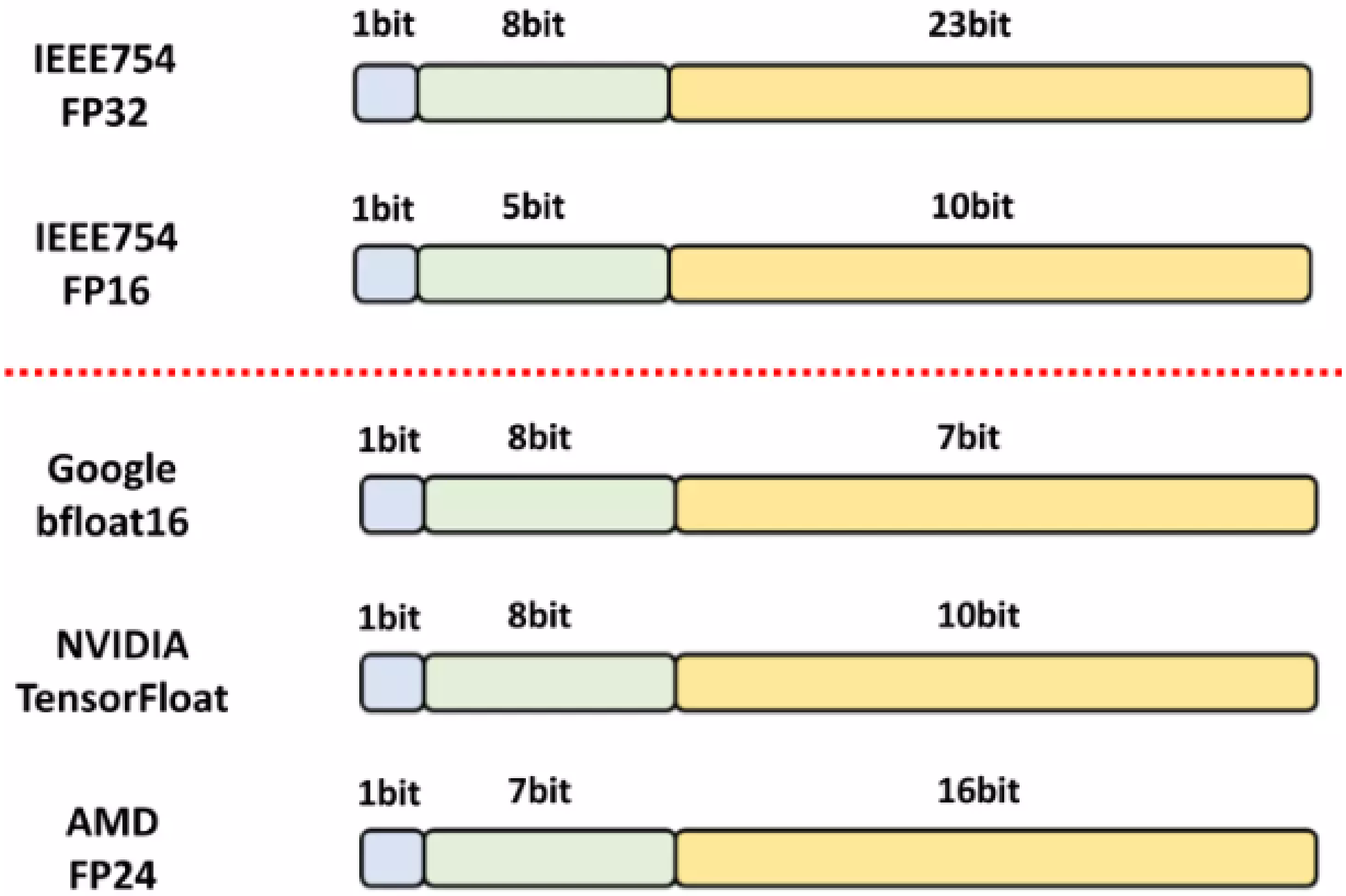
0 10000001 10100000000000000000000
Assume bias is 127
(-1)^0 * (1.625) * 2^(129 - 127) = 1.625 * 4
= 6.5

Half Precision (Floating Point 16)	1 bit	5 bit	10 bit
Single Precision (Floating Point 32)	1 bit	8 bit	23 bit
Double Precision (Floating Point 64)	1 bit	11 bit	52 bit
Quadruple Precision (Floating Point 128)	1 bit	15 bit	113 bit

New Floating-Point Format



New Floating-Point Format



Reduced storage and bandwidth

What is Mixed Precision Training? In a Nutshell

- Idea that you can train deep neural networks in multiple precisions:
  - Make precision decisions per layer or operation
  - Full precision (Fp32) where needed to *maintain task-specific accuracy*
  - Reduced precision (Fp16) everywhere else for speed and scale

What is Mixed Precision Training? In a Nutshell

- Idea that you can train deep neural networks in multiple precisions:
  - Make precision decisions per layer or operation
  - Full precision (Fp32) where needed to *maintain task-specific accuracy*
  - Reduced precision (Fp16) everywhere else for speed and scale

- By using *multiple* precisions, we can have the best of both worlds: **speed** and **accuracy**

- Goal: accelerate deep neural network training with mixed precision under the constraints of **matching accuracy of full precision training** and **no changes to how model is trained**

Mixed Precision Training Benefits

- **Accelerates speed**
  - Tensor cores are 8x faster than FP32

Mixed Precision Training Benefits

- **Accelerates speed**
  - Tensor cores are 8x faster than FP32

- **Reduces memory bandwidth pressure**
  - FP16 halves memory traffic compared to FP32

Mixed Precision Training Benefits

- **Accelerates speed**
  - Tensor cores are 8x faster than FP32
- **Reduces memory bandwidth pressure**
  - FP16 halves memory traffic compared to FP32
- **Reduces memory consumption**
  - FP16 halves the size of activation and gradient tensors
  - Enables larger models, mini batches or inputs

Why Use Mixed Precision? Pure FP16?

- Models cannot always converge properly in pure FP16
  - FP16 has a narrower dynamic range than FP32
  - May cause underflow/overflow issues and other arithmetic issues

Why Use Mixed Precision? Pure FP16?

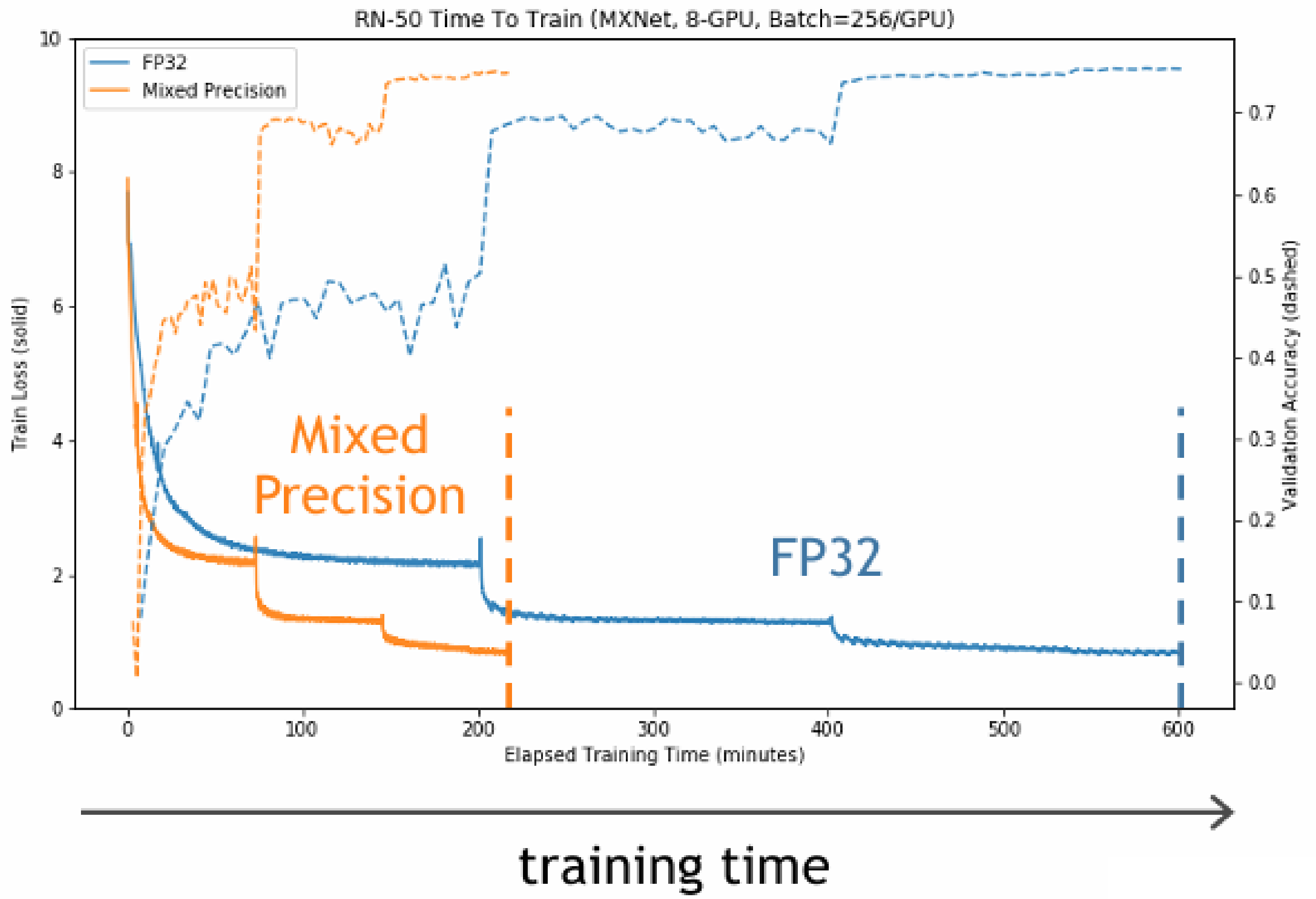
- Models cannot always converge properly in pure FP16
  - FP16 has a narrower dynamic range than FP32
  - May cause underflow/overflow issues and other arithmetic issues
- **Weight updates**
  - Optimizer takes very **small increments** when search narrows into a solution
  - Late updates often cannot be represented in FP16, but can be crucial for accuracy

Why Use Mixed Precision? Pure FP16?

- Models cannot always converge properly in pure FP16
  - FP16 has a narrower dynamic range than FP32
  - May cause underflow/overflow issues and other arithmetic issues
- **Weight updates**
  - Optimizer takes very **small increments** when search narrows into a solution
  - Late updates often cannot be represented in FP16, but can be crucial for accuracy
- **Reductions**
  - Large sums of values, e.g., in linear layers and convolutions, can be **too big** for FP16
  - Adding small values to a large sum can lead to **rounding errors**

Mixed Precision Training Example

- ResNet-50 training for ImageNet classification
  - 8 GPUs on DGX-1
- Comparing to FP32 training
  - **~3x speedup**
  - **Equal accuracy**
  - No hyperparameters changed



Mixed Precision is Generalizable

- Works across a wide range of tasks, problem domains, deep neural network architectures

Image Classification
AlexNet
DenseNet
Inception
MobileNet
NASNet
ResNet
ResNeXt
ShuffleNet
SqueezeNet
VGG
Xception

Detection / Segmentation
DeepLab
Faster R-CNN
Mask R-CNN
SSD
NVIDIA Automotive
RetinaNet
UNET

Recommendation
DeepRecommender
NCF

Generative Models (Images)
DLSS
GauGAN
Partial Image Inpainting
Progress GAN
Pix2Pix

Speech
Deep Speech 2
Jasper
Tacotron
Wave2vec
WaveNet
WaveGlow

Language Modeling
BERT
BigLSTM
Gated Convolutions
mLSTM
RoBERTa
Transformer XL

Translation
Convolutional Seq2Seq
Dynamic Convolutions
GNMT (RNN)
Levenshtein Transformer
Transformer (Self-Attention)

Mixed Precision Trains Faster: Drastically Reduces Training Time

Task	Model	Speedup
Image Classification	ResNet-50	3.6x
	DenseNet 201	2.2x*
	Xception	2.1x*
Detection / Segmentation	SSD	2.5x**
	Mask R-CNN	1.5x
	RetinaNet	2.0x

* 2x batch size ** Larger batch size

Task	Model	Speedup
Translation	GNMT	2.3x
	Transformer	2.9x 4.9x**
	Convolutional Seq2Seq	2.5x*
Speech	Deep Speech 2	4.5x**
	Wav2letter	3.0x*
	WaveGlow	1.9x
Language Modeling	mLSTM	4.0x**
	BERT	3.3x

Weeks to days

Days to hours

Hours to minutes

Mixed Precision Advances DL Research

"This paper shows that reduced mixed precision and large batch training can speedup training by nearly 5x on a single 8-GPU machine with careful tuning and implementation."

Scaling Neural Machine Translation, Facebook

"We train with mixed precision floating point arithmetic on DGX-1 machines [...] using 1024 V100 GPUs for approximately one day."

RoBERTa, Facebook

"leverages mixed precision training [...] largest transformer based language model ever trained at 24x the size of BERT and 5.6x the size of GPT-2."

MegatronLM, NVIDIA

Considerations for Mixed Precision

Considerations for Mixed Precision Training

- Goal #1: Make FP16 training general purpose, not only for limited class of applications
- Goal #2: With no changes to hyperparameters or the model architecture

Three parts:

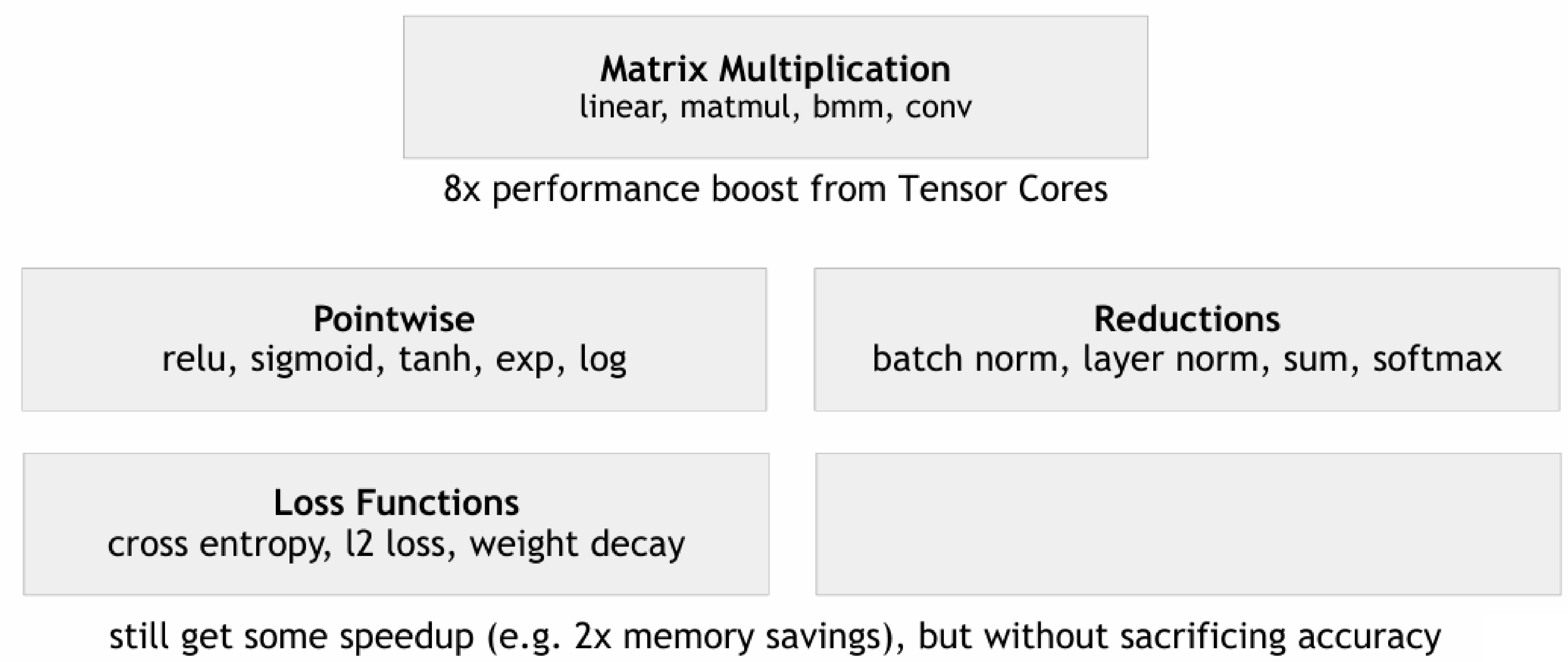
PRECISION OF OPS	MASTER WEIGHTS	LOSS SCALING
Decide which operations to compute in FP16 and FP32.	Keep an FP32 copy of the model weights.	Scale the loss value to retain small gradients.

Precision Choices for Different Classes of Operations

Matrix Multiplication
linear, matmul, bmm, conv

8x performance boost from Tensor Cores

Precision Choices for Different Classes of Operations



Precision of OPS: Conservative Recommendations

- Operations that can use FP16
  - Matrix multiplications
  - Most element-wise operations (e.g., relu, tanh, add, sub, mul)

Precision of OPS: Conservative Recommendations

- Operations that can use FP16
  - Matrix multiplications
  - Most element-wise operations (e.g., relu, tanh, add, sub, mul)

- Operations that need FP32 mantissa
  - Reduction operations

Precision of OPS: Conservative Recommendations

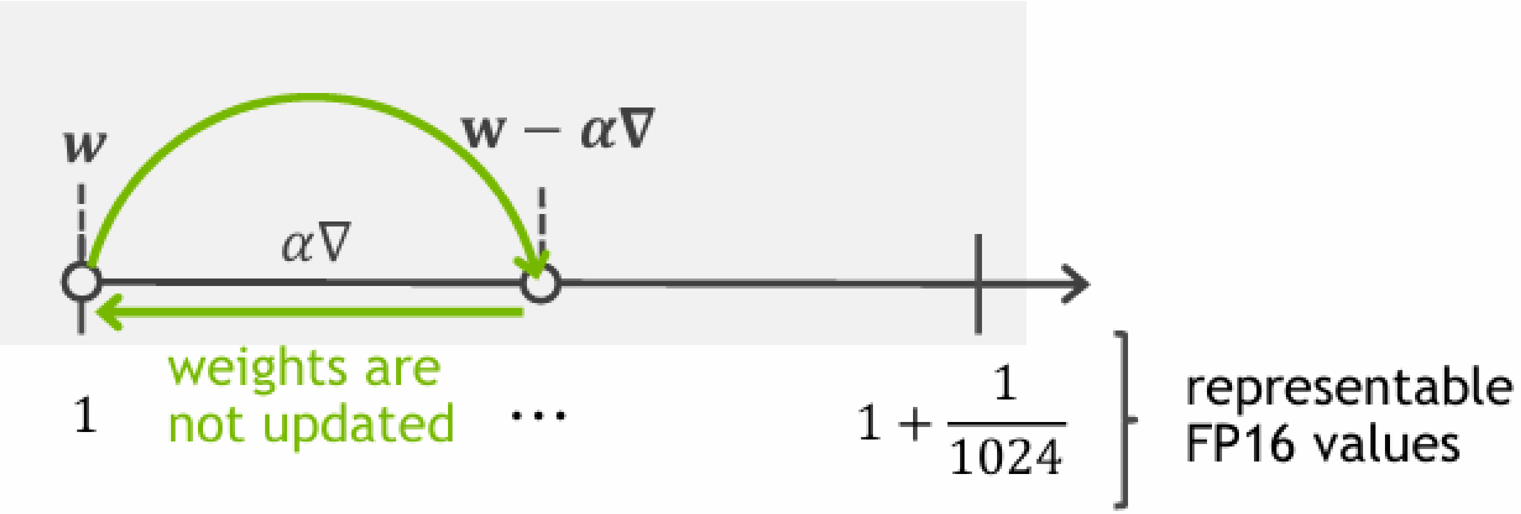
- Operations that can use FP16
  - Matrix multiplications
  - Most element-wise operations (e.g., relu, tanh, add, sub, mul)
- Operations that need FP32 mantissa
  - Reduction operations
- Operations that need FP32 range
  - Element-wise operations where $|f(x)| >> |x|$, e.g., exp, log, pow
  - Loss functions

Master Weights

- **Problem:** in late stages of training, weight updates become too small for addition in FP16
- **Consequence:** weight update gets clipped to zero when $w \gg \alpha\nabla$

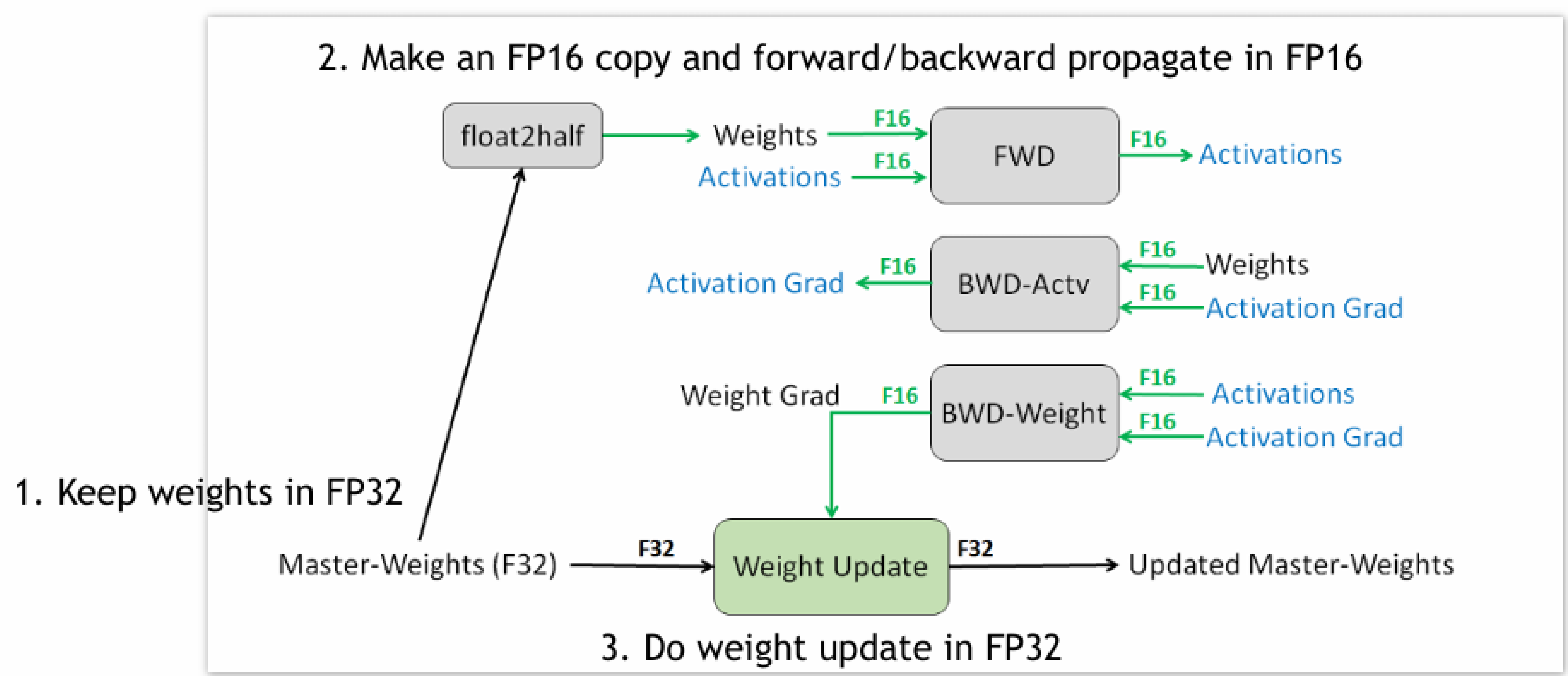
Master Weights

- **Problem:** in late stages of training, weight updates become too small for addition in FP16
- **Consequence:** weight update gets clipped to zero when $w \gg \alpha\nabla$



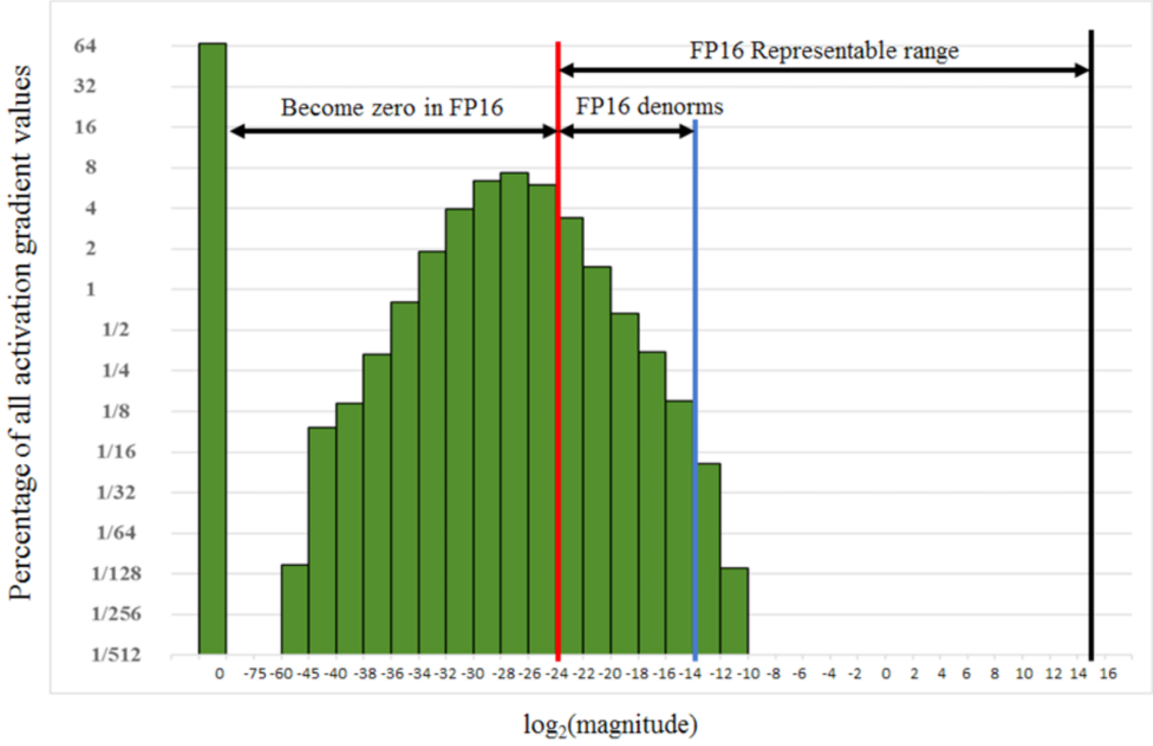
- **Conservative solution:** keep master copy of weights in FP32 so small updates can accumulate

Master Weights: Illustration



Loss Scaling: Put All Tensors in FP16 Range

- Range representable in FP16: ~40 powers of 2
- Gradients are small:
  - If cast to Fp16, some lost to zero
  - Much of fp16 representable ranges was left unused

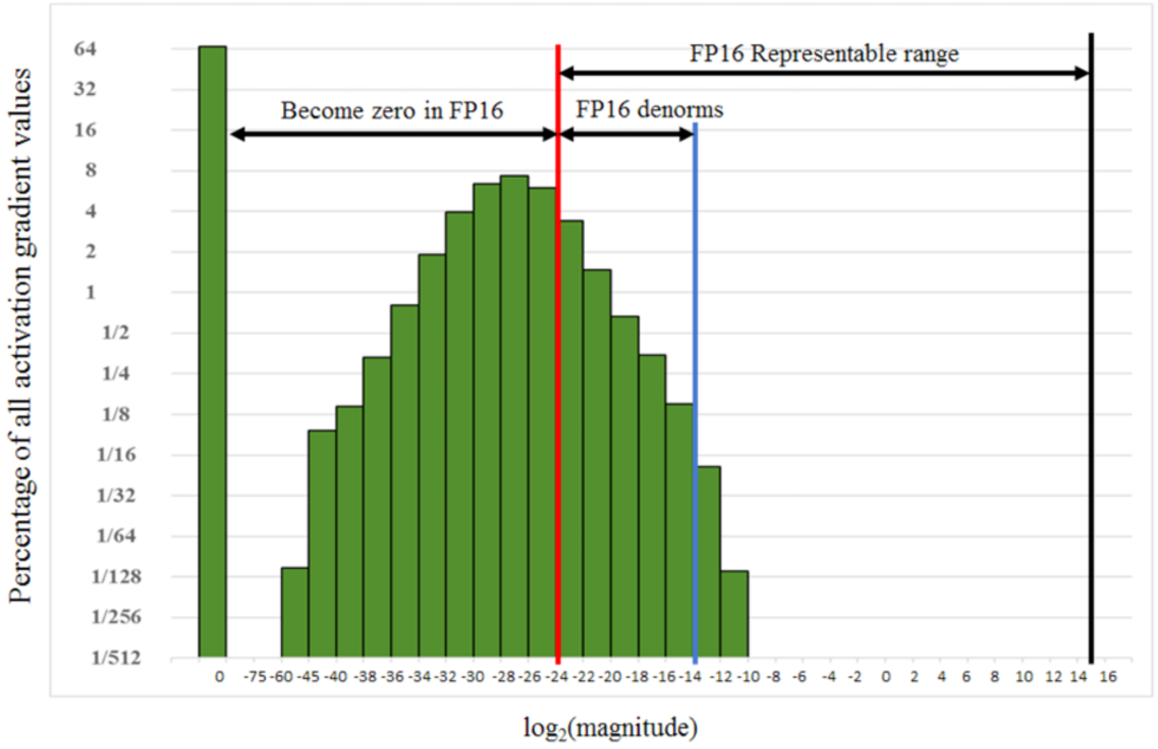


Histogram of activation gradient values

Loss Scaling: Put All Tensors in FP16 Range

- Range representable in FP16: ~40 powers of 2
- Gradients are small:
  - If cast to Fp16, some lost to zero
  - Much of fp16 representable ranges was left unused

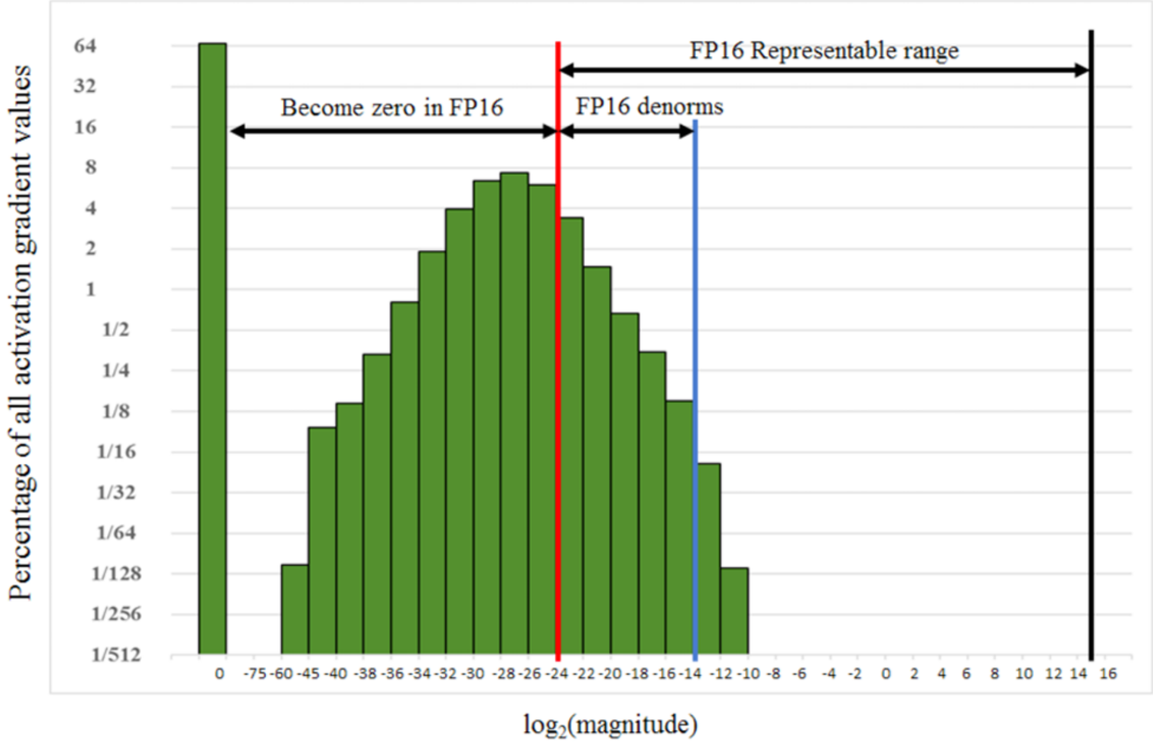
Question: How can we make better use of FP16 representable range?



Histogram of activation gradient values

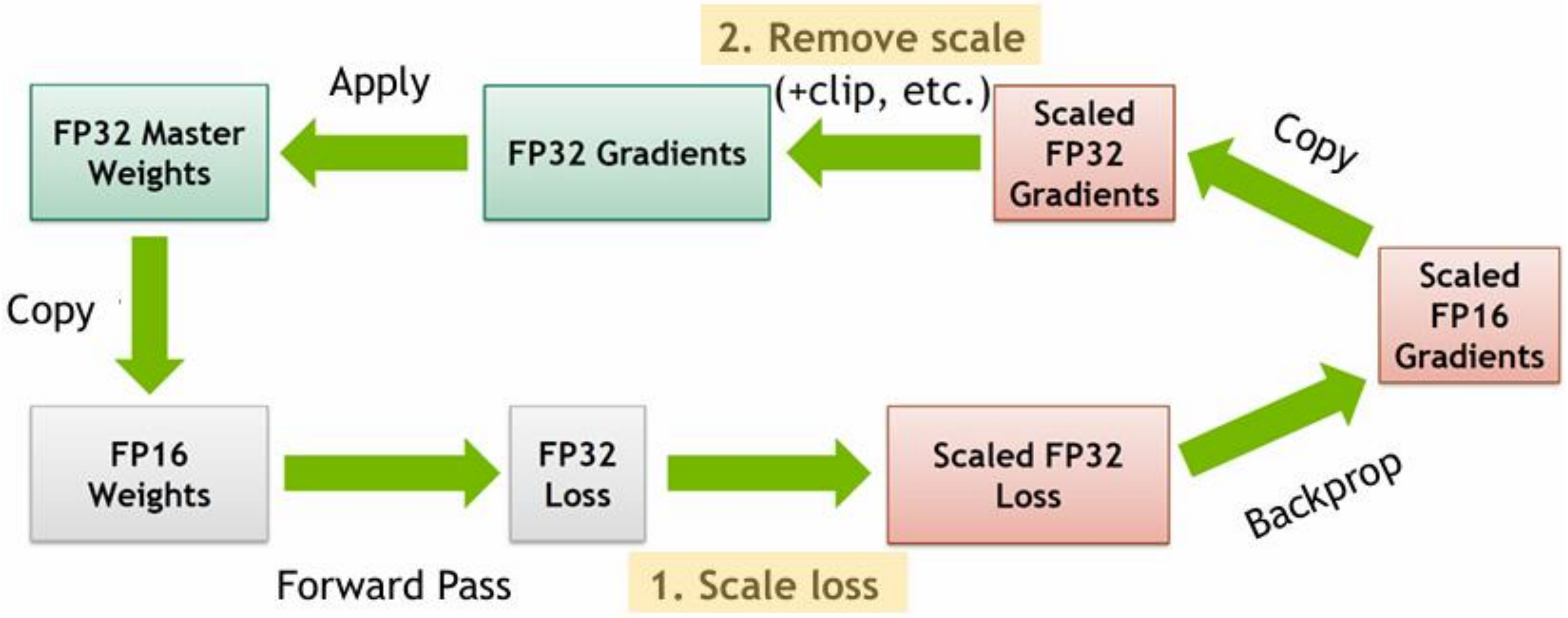
Loss Scaling: Put All Tensors in FP16 Range

- Range representable in FP16: ~40 powers of 2
- Gradients are small:
  - If cast to Fp16, some lost to zero
  - Much of fp16 representable ranges was left unused
- Solution:
  - Move small gradients values to fp16 range
  - Scaling gradients during backpropagation to prevent underflow
  - Gradients are scaled before backpropagation begins and rescaled before updating weights



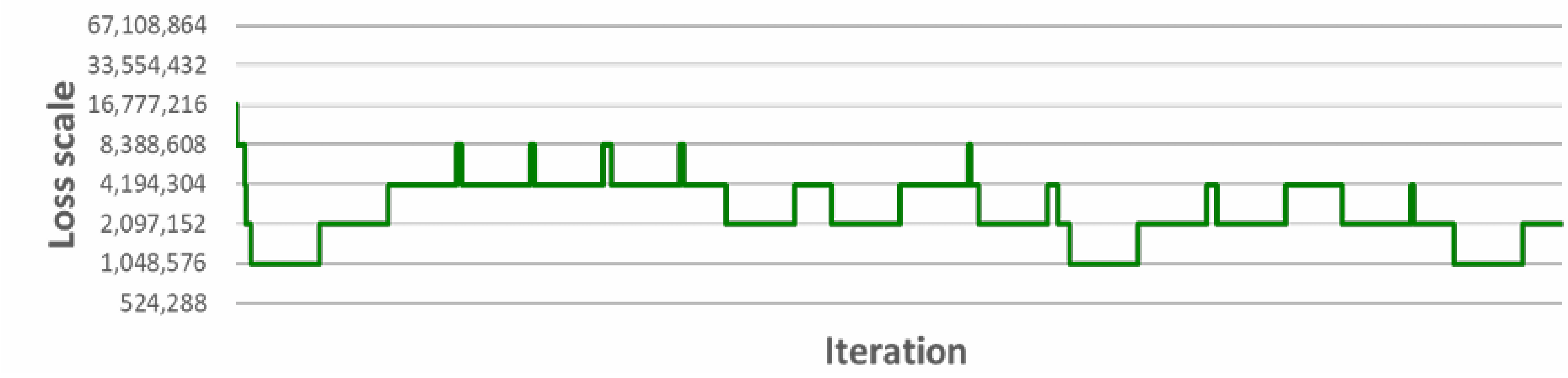
Histogram of activation gradient values

Loss Scaling: Algorithm



Automatic Loss Scaling: Frees Users from Choosing a Scaling Factor

- 1. Start with very large scale factor (e.g., 2^24)
- 2. If gradients overflows (with Inf or a Nan): decrease the scale by 2 and skip the update
- 3. If no overflows have occurred for some time: increase the scale by 2 (e.g., 2000 iterations)



[Train With Mixed Precision - NVIDIA Docs]

Mixed Precision Software

Mixed Precision Software

- Goal: Make mixed precision training easy with minimum effort for DL practitioners

- Available for major DL frameworks

PyTorch|DeepSpeed|Megatron

- Works with all types of optimizers
  - SGD, Adam, etc

- Works with multiple models, optimizers, and losses

Enabling Mixed Precision

- Traditionally a lot of work (recap on methodology)
- Model conversion
  - Switch everything to run on FP16 values
  - Cast to FP32 for loss functions, normalization, and element-wise ops that need full precision
- Master weights
  - Keep FP32 copy of model parameters
  - Make FP16 copies during forward/backward passes
- Loss scaling
  - Scale the loss value, unscale the gradients in FP32
  - Check gradients at each iteration to adjust loss scale and skip on overflow

Automatic Mixed Precision (AMP)

- Automates everything from the previous slides
- Framework software to run mixed precision automatically
- Details vary by framework, but core idea is the same
- Automatic Mixed Precision (AMP) does two things:

AUTOMATIC LOSS SCALING

Wraps the optimizer in order to:
- scale loss value & unscale gradients
- adjust scale & skip on gradient overflow

AUTOMATIC CASTING

Wraps model and operations in order to:
- cast data to FP16
- switch *everything* to run on FP16
- keep certain operations in FP32
- keep master copy of weights in FP32

Automatic Casting

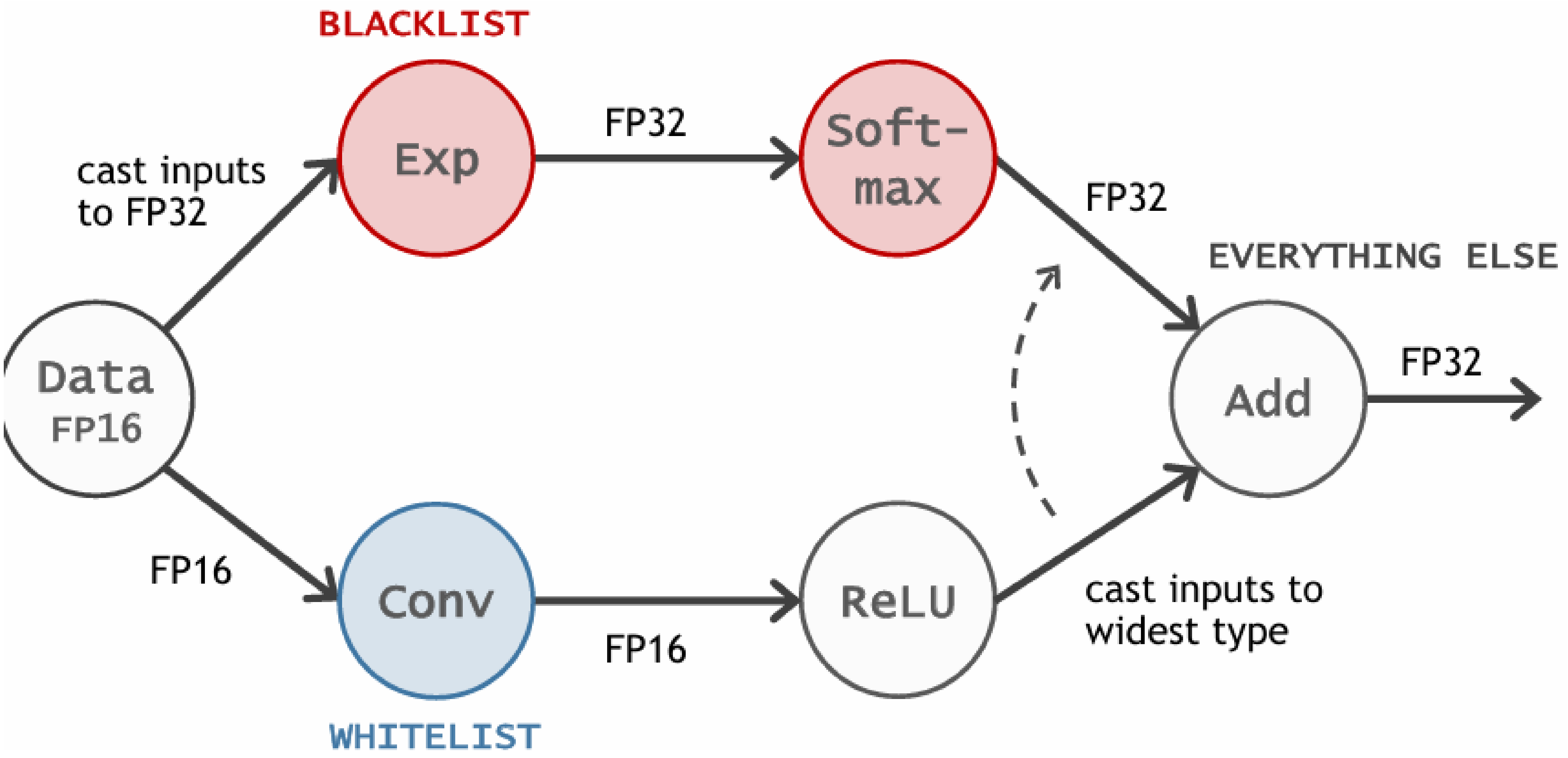
- Make type decisions for each operation a prior (static graph) or at runtime (eager execution)

- Define conservative set of rules to replace “by-hand” mixed precision

Divide the universe of operations into three kinds:

WHITELIST	BLACKLIST	EVERYTHING ELSE
FP16 enables Tensor Cores.	FP32 is needed for accuracy.	Can run in FP16, but only if inputs already in FP16.
e.g. linear, bmm, convs	e.g. loss, exp, sum, softmax	e.g. relu, add, maxpool
Rule: always run in FP16, cast if necessary.	Rule: always run in FP32, cast if necessary.	Rule: run in existing input type.

Automatic Casting



AMP for PyTorch

To control the operations being casted

```
model, optimizer = amp.initialize(model, optimizer, opt_level="O1")
```

O0	O1	O2
FP32 Training Leave everything in FP32.	**Mixed Precision Training** FP16 whitelist and FP32 blacklist ops.	**FP16 Training** FP16 model/data with FP32 batchnorm.

Questions?