S62457 What's New in Transformer Engine and FP8

Przemyslaw Tredak, NVIDIA | March 20th 2024

Agenda

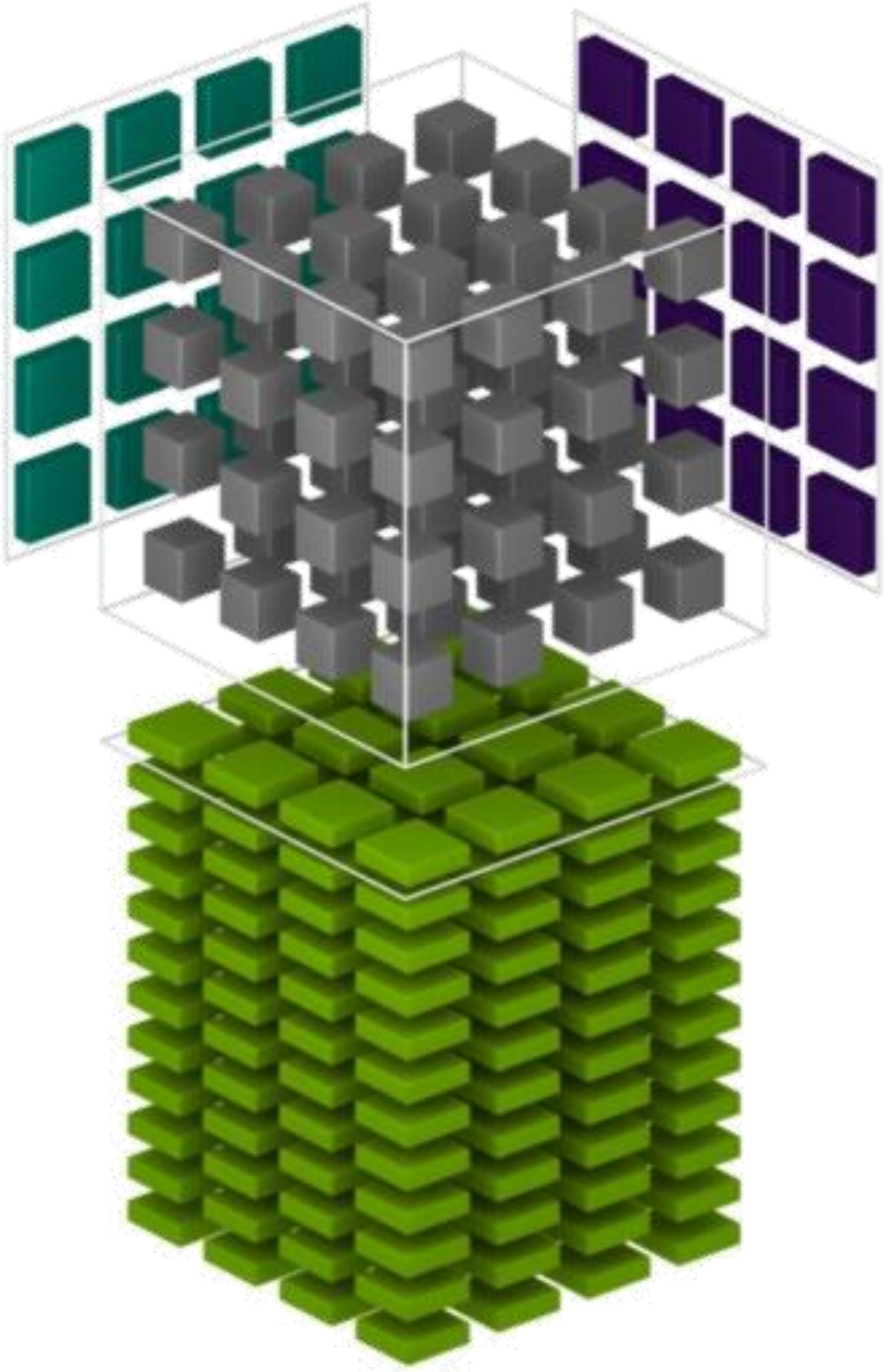
- FP8 Training
- Transformer Engine
- What’s new in TE?
- Future of TE

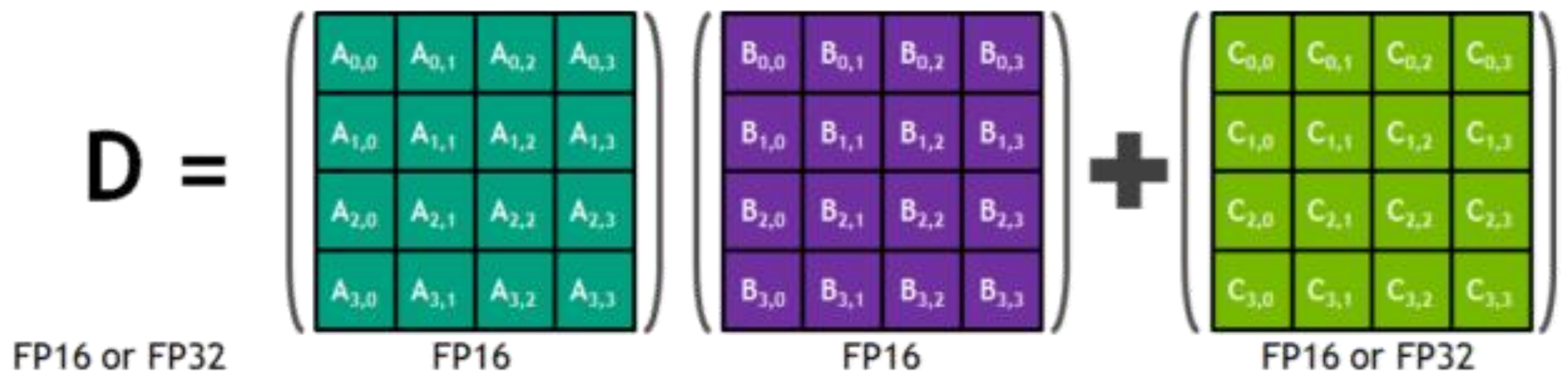
Training in FP8

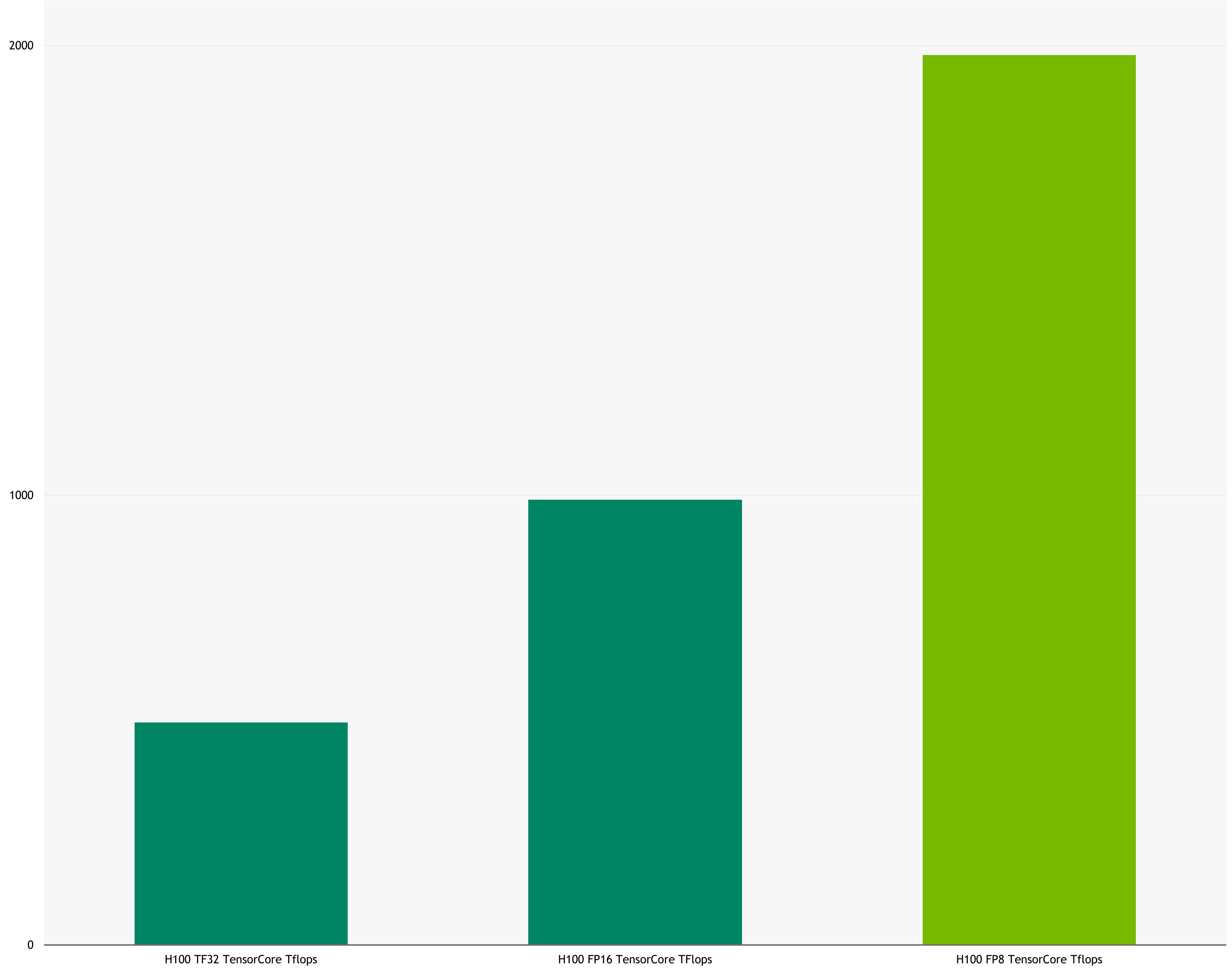
TensorCores and Mixed Precision

Motivation

- Starting with Volta, NVIDIA GPUs feature **TensorCores**
- They greatly speed up matrix multiplication and convolution
- To get the maximum performance, training in **mixed precision** is required







FP8

A little theory

FP32 = 0.3952

	sign		exponent									mantissa						
FP16	0	0	1	1	0	1	1	0	0	1	0	1	0	0	1	1	= 0.395264	More precision
BF16	0	0	1	1	1	1	1	0	1	1	0	0	1	0	1	0	= 0.394531	More range
FP8 E4M3	0	0	1	0	1	1	0	1									= 0.40625	More precision
FP8 E5M2	0	0	1	1	0	1	1	0									= 0.375	More range

Mixed Precision Recipe

The FP16 way

- Partition the DL network graph into safe and unsafe regions
  - Safe regions contain operations benefitting from reduced precision and whose outputs' dynamic ranges are similar to the inputs
- Use the scaling factor during the backward pass
  - Scaling factor is used to avoid over- and underflows in the value distribution of the tensors

NVIDIA

Mixed Precision Recipe

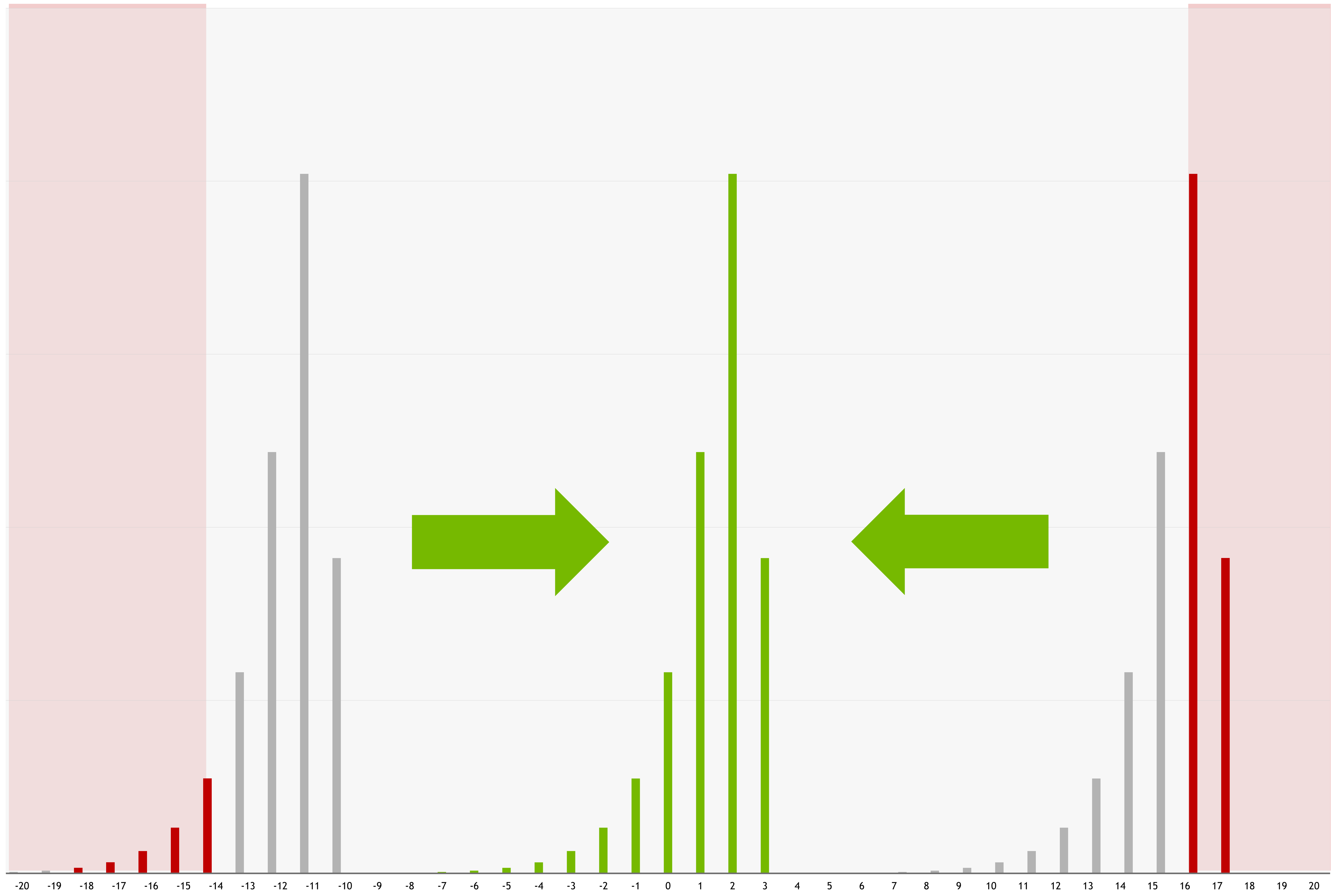
The FP16 way

- Partition the DL network graph into safe and unsafe regions
  - Safe regions contain operations benefitting from reduced precision and whose outputs' dynamic ranges are similar to the inputs
- Use the scaling factor during the backward pass
  - Scaling factor is used to avoid over- and underflows in the value distribution of the tensors

Mixed Precision Recipe

The FP16 way

- Partition the DL network graph into safe and unsafe regions
  - Safe regions contain operations benefitting from reduced precision and whose outputs’ dynamic ranges are similar to the inputs
- Use the scaling factor during the backward pass
  - Scaling factor is used to avoid over- and underflows in the value distribution of the tensors



Mixed Precision Recipe

Enter FP8

- Partition the DL network graph into safe and unsafe regions
  - Unsafe region does not necessarily need to be FP32, FP8 training recipe can be combined with FP16/BF16 recipe
  - Explicit casts are not enough - FP8 operators need to use higher precision internally and be able to output higher precision output
- Use the per-tensor scaling factors
  - Scaling factors are needed in both passes
    - E4M3 for forward, E5M2 for backward
  - A single scaling factor is no longer enough

Mixed Precision Recipe

Enter FP8

- Partition the DL network graph into safe and unsafe regions
  - Unsafe region does not necessarily need to be FP32, FP8 training recipe can be combined with FP16/BF16 recipe
  - Explicit casts are not enough - FP8 operators need to use higher precision internally and be able to output higher precision output
- Use the per-tensor scaling factors
  - Scaling factors are needed in both passes
    - E4M3 for forward, E5M2 for backward
  - A single scaling factor is no longer enough

Mixed Precision Recipe

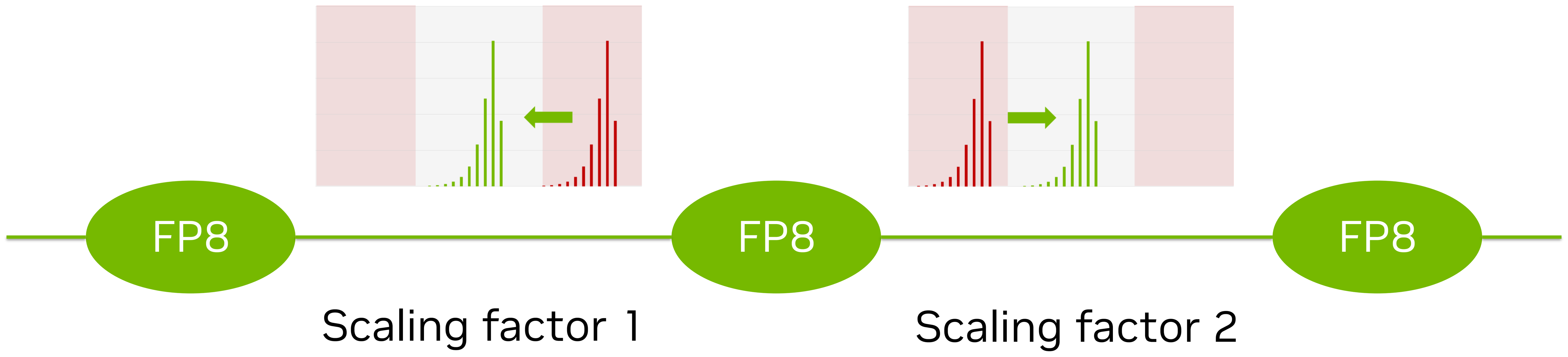
Enter FP8

- Partition the DL network graph into safe and unsafe regions
  - Unsafe region does not necessarily need to be FP32, FP8 training recipe can be combined with FP16/BF16 recipe
  - Explicit casts are not enough - FP8 operators need to use higher precision internally and be able to output higher precision output
- Use the per-tensor scaling factors
  - Scaling factors are needed in both passes
    - E4M3 for forward, E5M2 for backward
  - A single scaling factor is no longer enough

Mixed Precision Recipe

Enter FP8

- Partition the DL network graph into safe and unsafe regions
  - Unsafe region does not necessarily need to be FP32, FP8 training recipe can be combined with FP16/BF16 recipe
  - Explicit casts are not enough - FP8 operators need to use higher precision internally and be able to output higher precision output
- Use the per-tensor scaling factors
  - Scaling factors are needed in both passes
    - E4M3 for forward, E5M2 for backward
  - A single scaling factor is no longer enough



Additional Challenges

Choosing the scaling factor

- Choosing the scaling factor for the FP8 output is simple conceptually, but hard in practice

Additional Challenges

Choosing the scaling factor

- Choosing the scaling factor for the FP8 output is simple conceptually, but hard in practice
- Impossible to keep the entire high precision output in the high speed memory to find the maximum and scale with it

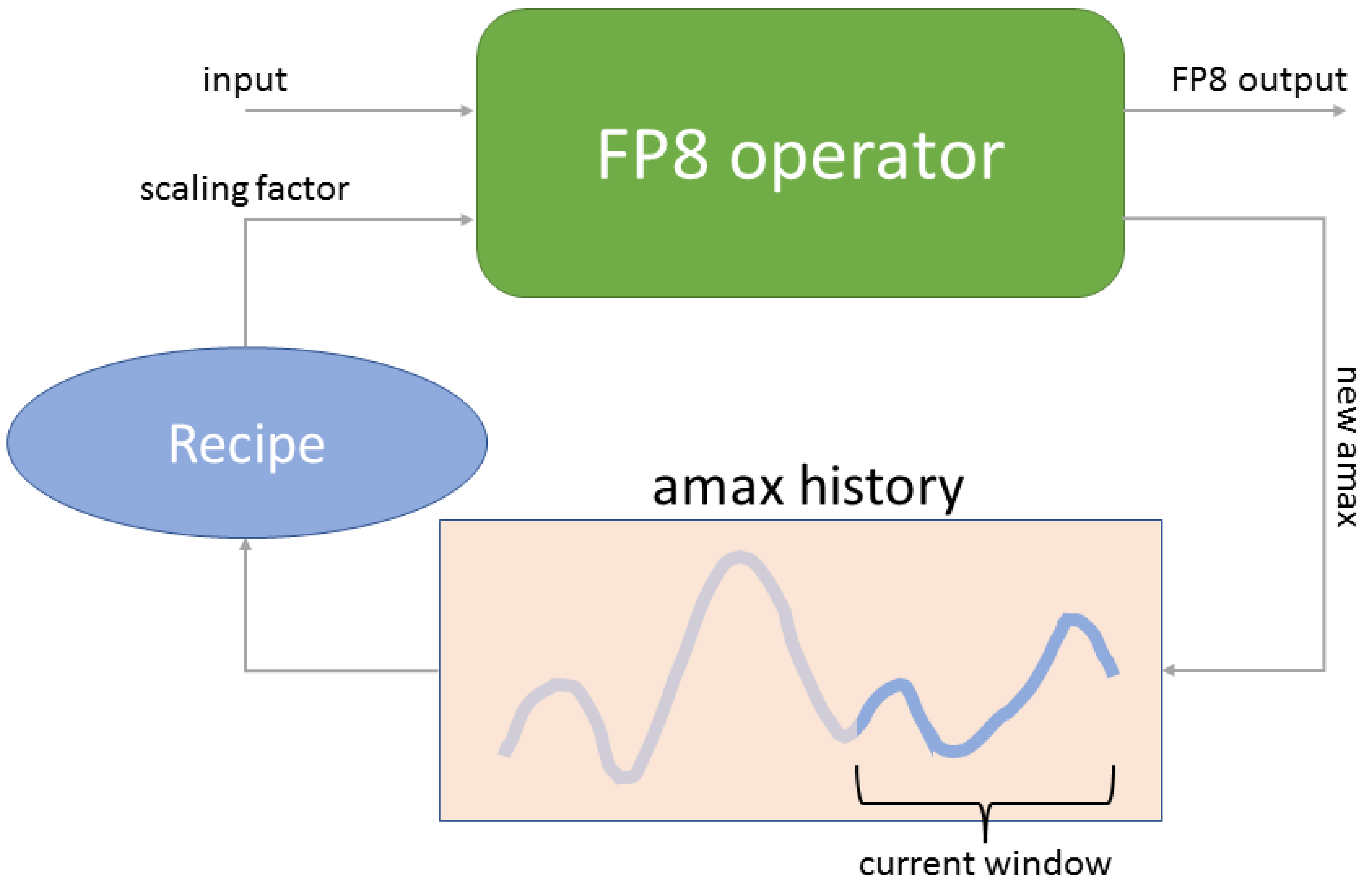
Additional Challenges

Choosing the scaling factor

- Choosing the scaling factor for the FP8 output is simple conceptually, but hard in practice
- Impossible to keep the entire high precision output in the high speed memory to find the maximum and scale with it
- To overcome that we need to know the scaling factor before seeing the output

Additional Challenges

Choosing the scaling factor

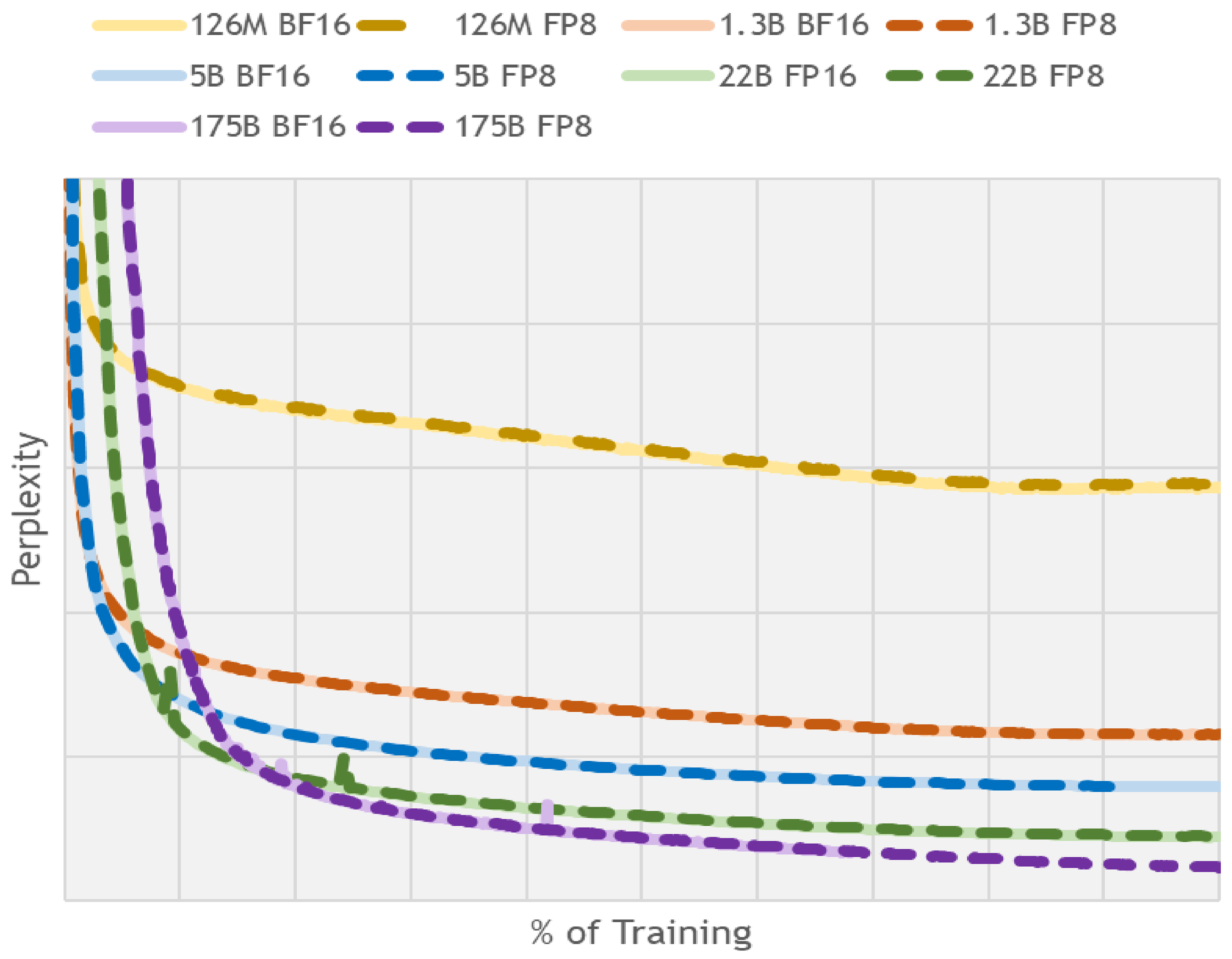


Transformer Engine

Transformer Engine

The introduction

- An open-source library implementing the FP8 recipe for Transformer building blocks
- Optimized for FP8 and other datatypes
- Supports pyTorch, JAX and PaddlePaddle
- Composable with the native framework operators
- Supports different types of model parallelism
- **https://github.com/NVIDIA/TransformerEngine**




```python
import torch
import transformer_engine.pytorch as te
from transformer_engine.common import recipe

# Set dimensions.
in_features = 768
out_features = 3072
hidden_size = 2048

# Initialize model and inputs.
model = te.Linear(in_features,
                  out_features,
                  bias=True)
inp = torch.randn(hidden_size,
                  in_features,
                  device="cuda")

# Create FP8 recipe.
fp8_recipe = recipe.DelayedScaling()

# Enable autocasting to FP8.
with te.fp8_autocast(enabled=True,
                     fp8_recipe=fp8_recipe):
    out = model(inp)

loss = out.sum()
loss.backward()
```

NVIDIA

Transformer Engine

The API

Transformer Engine

FP8 autocast

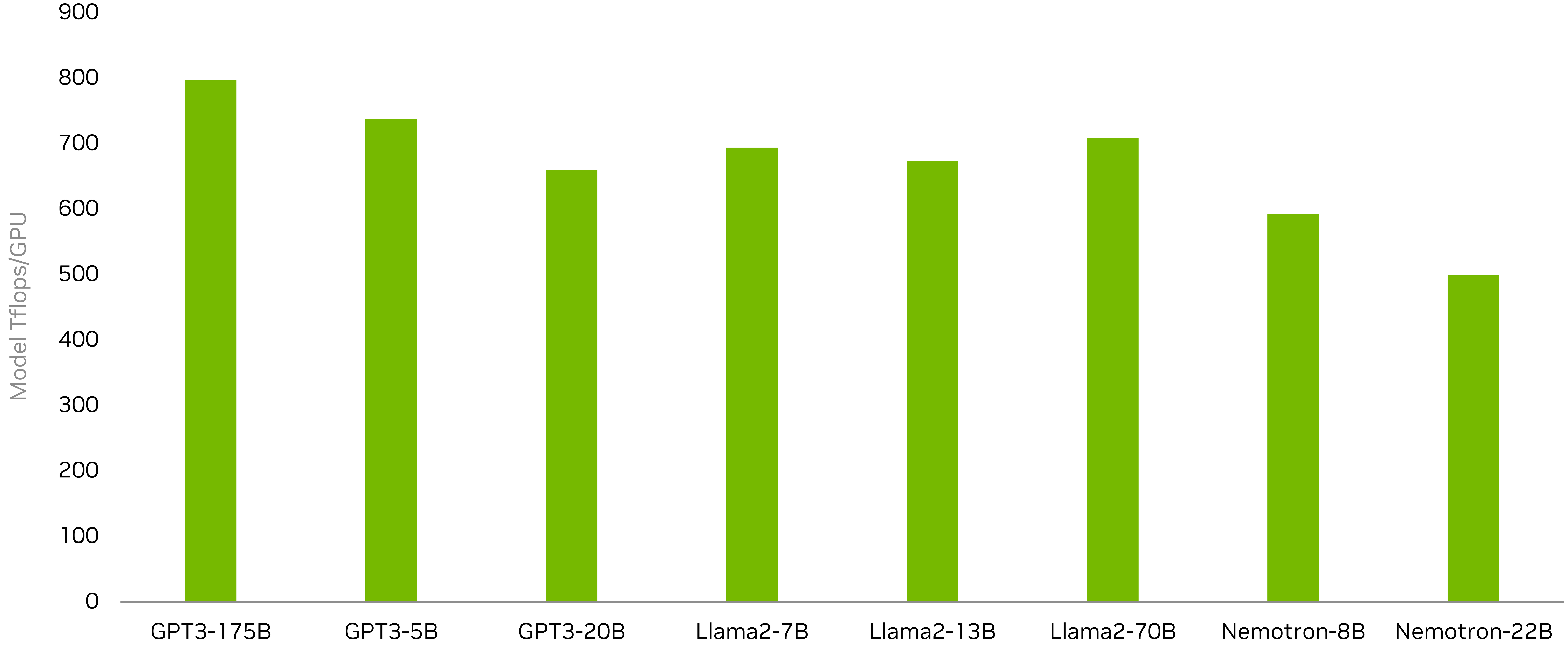
```
# Enable autocasting to FP8.
with te.fp8_autocast(enabled=True,
                     fp8_recipe=fp8_recipe):
    out = model(inp)

loss = out.sum()
loss.backward()
```

- `fp8_autocast` context manager tells Transformer Engine's modules to use FP8 computation internally
- Enables specifying the details about the chosen FP8 recipe, like a different size of the history window or a different algorithm for calculating the scaling factors
- It does not change anything else in the model
  - For example, for a mixed FP16/FP8 training it needs to be combined with native framework's AMP or casting of the model!
- Backward pass inherits the settings of the forward pass and should be outside of the `fp8_autocast` region

Performance

Pretraining using H100 and FP8 with NeMo

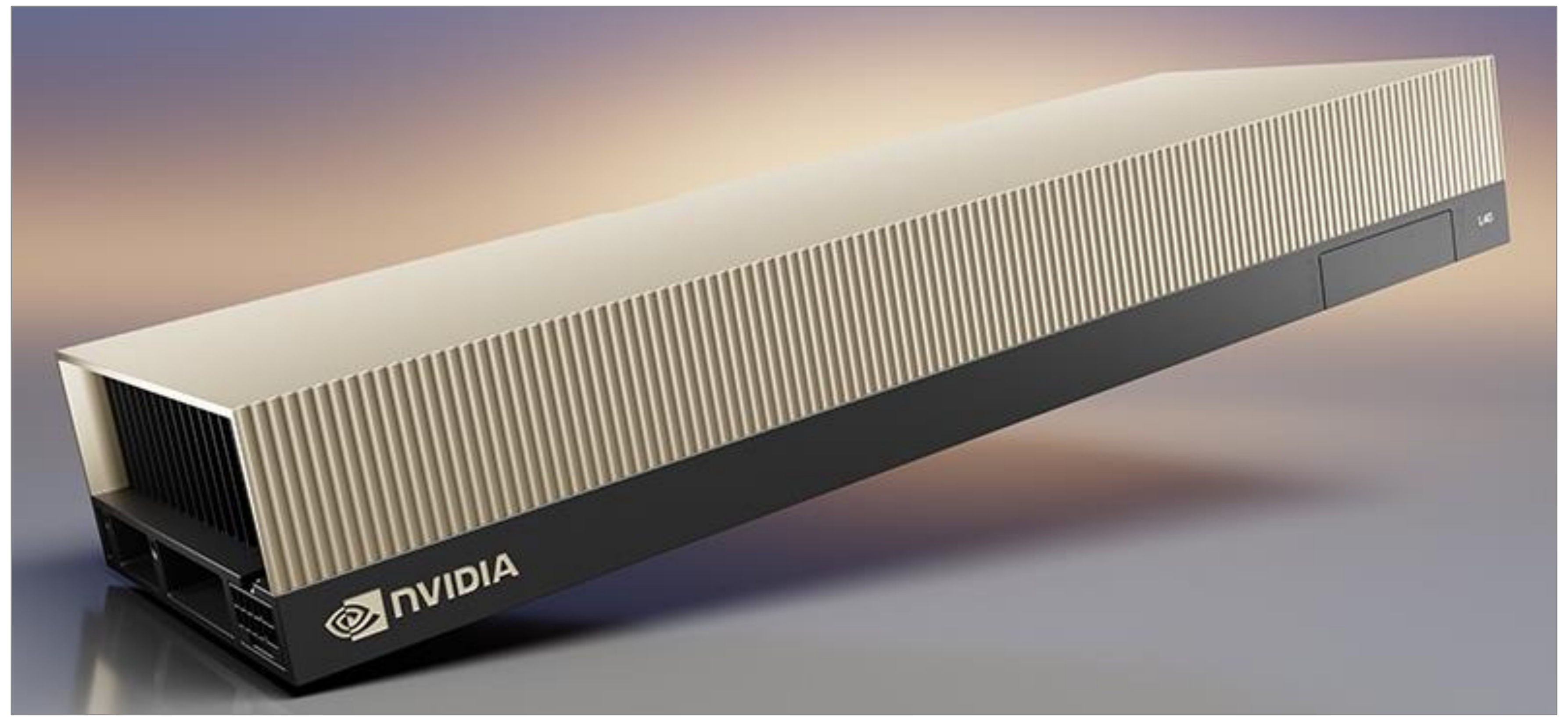


Source: https://github.com/NVIDIA/NeMo-Megatron-Launcher/tree/master/examples#benchmark-performance-numbers-pretraining

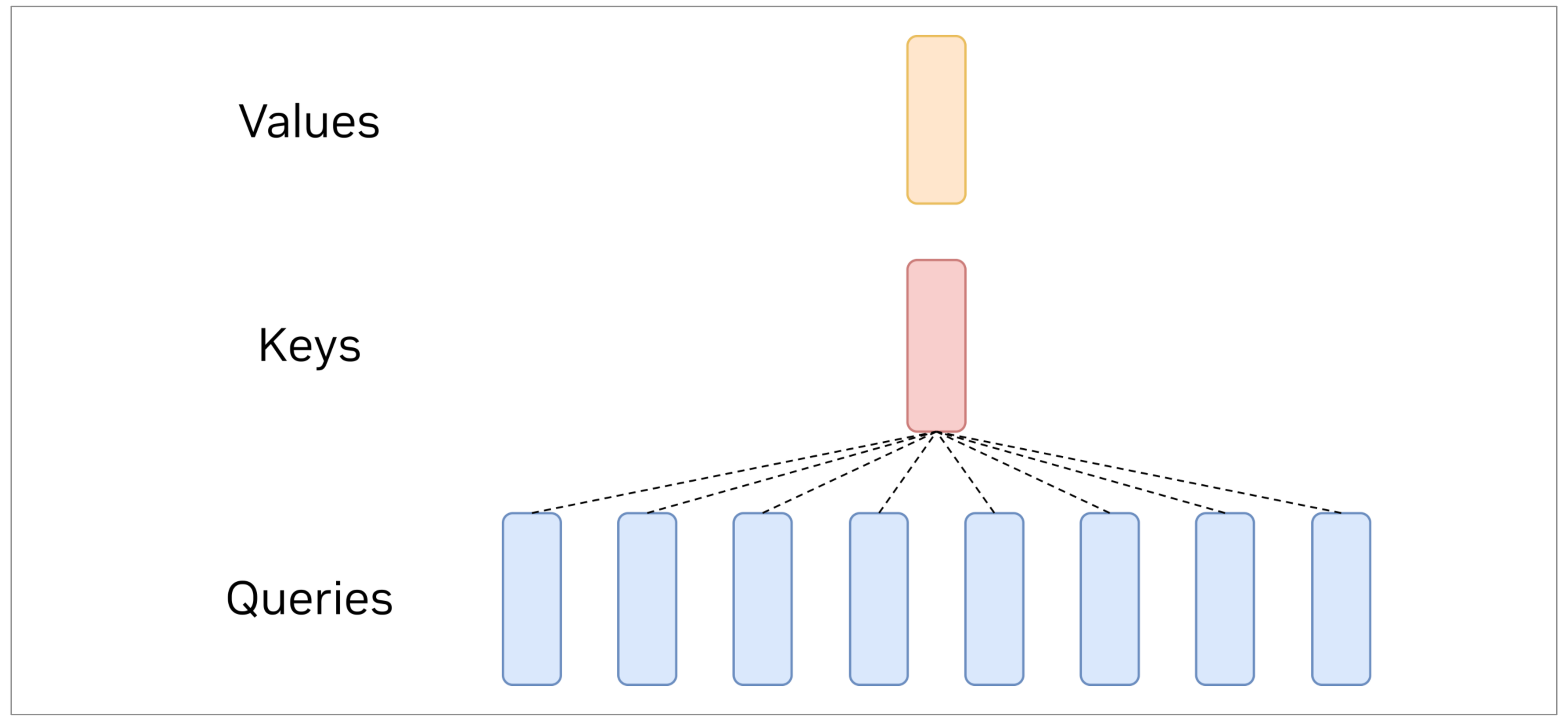
What’s new in TE

New functionality

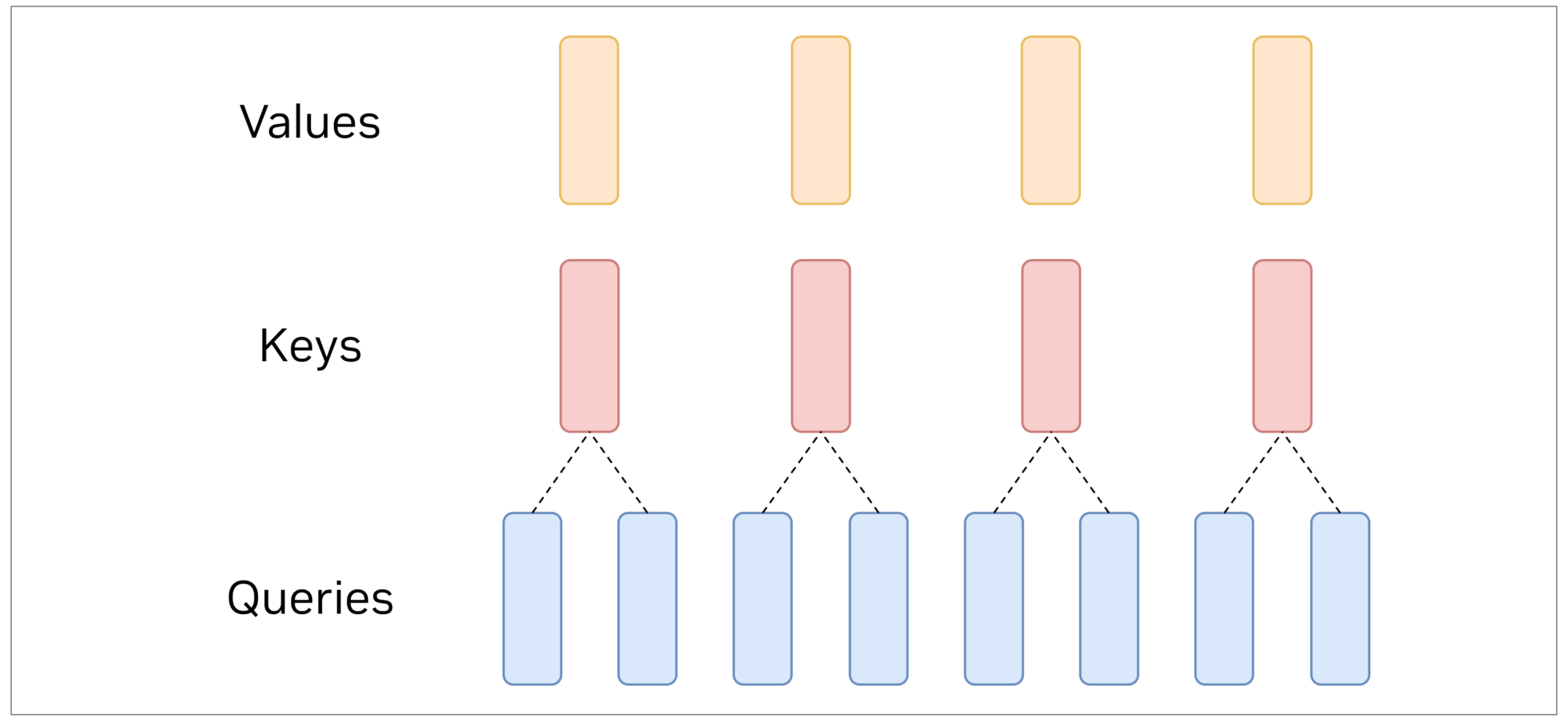
Since last GTC



Support FP8 on Ada GPUs



Multi-query attention



Grouped-query attention

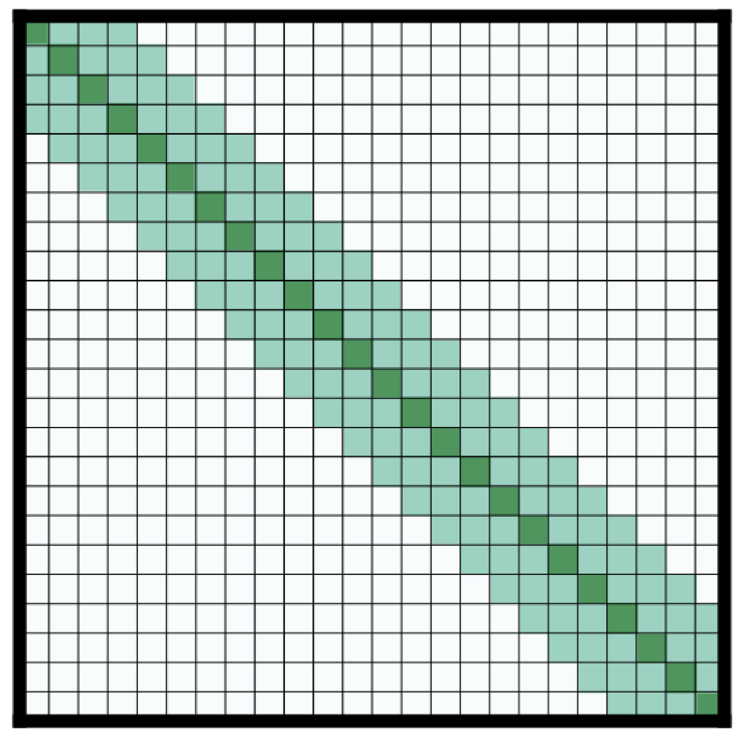
$$y = \frac{x}{RMS_\varepsilon(x)} * \gamma$$

where

$$RMS_\varepsilon(x) = \sqrt{\frac{1}{n}\sum_{i=0}^{n} x_i^2 + \varepsilon}$$

RMSNorm

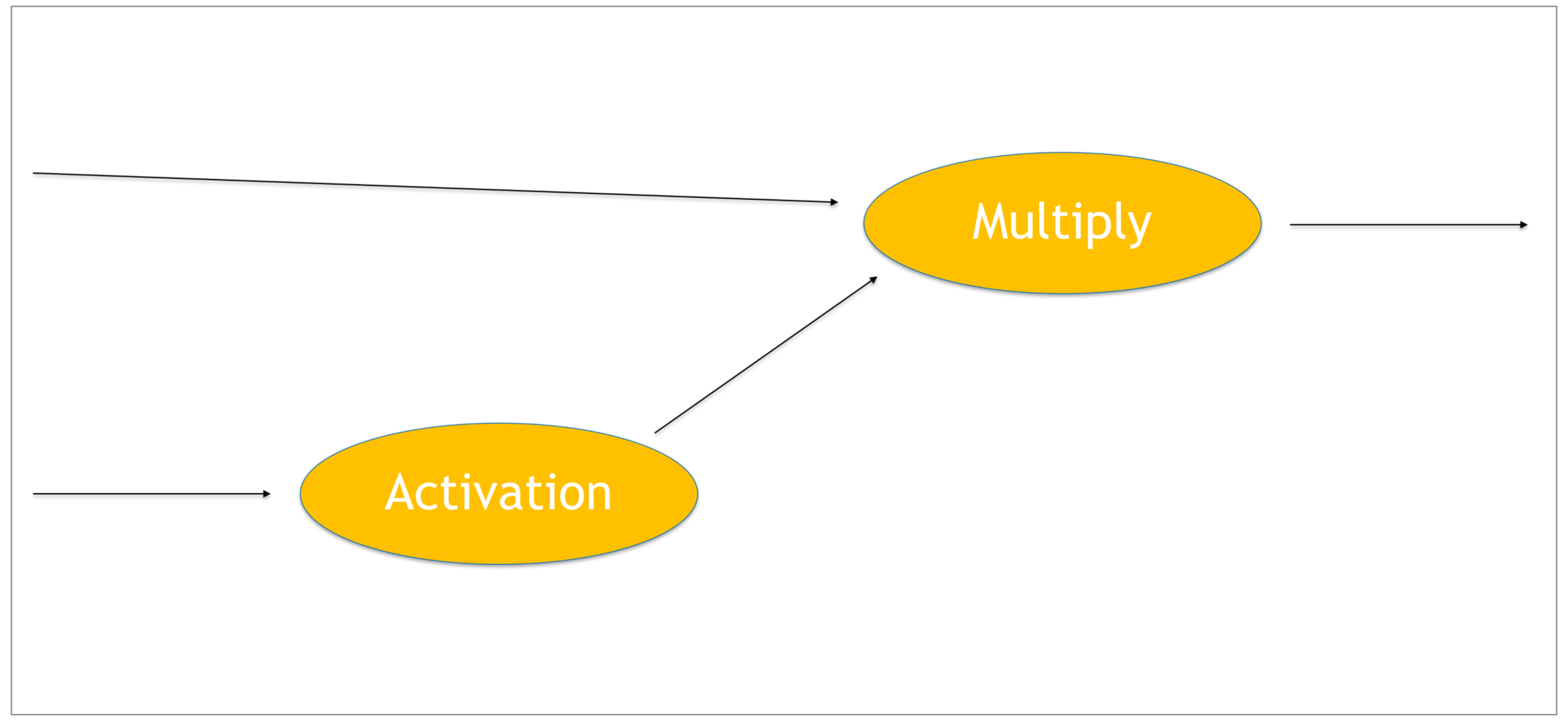
Including zero-centered gamma support



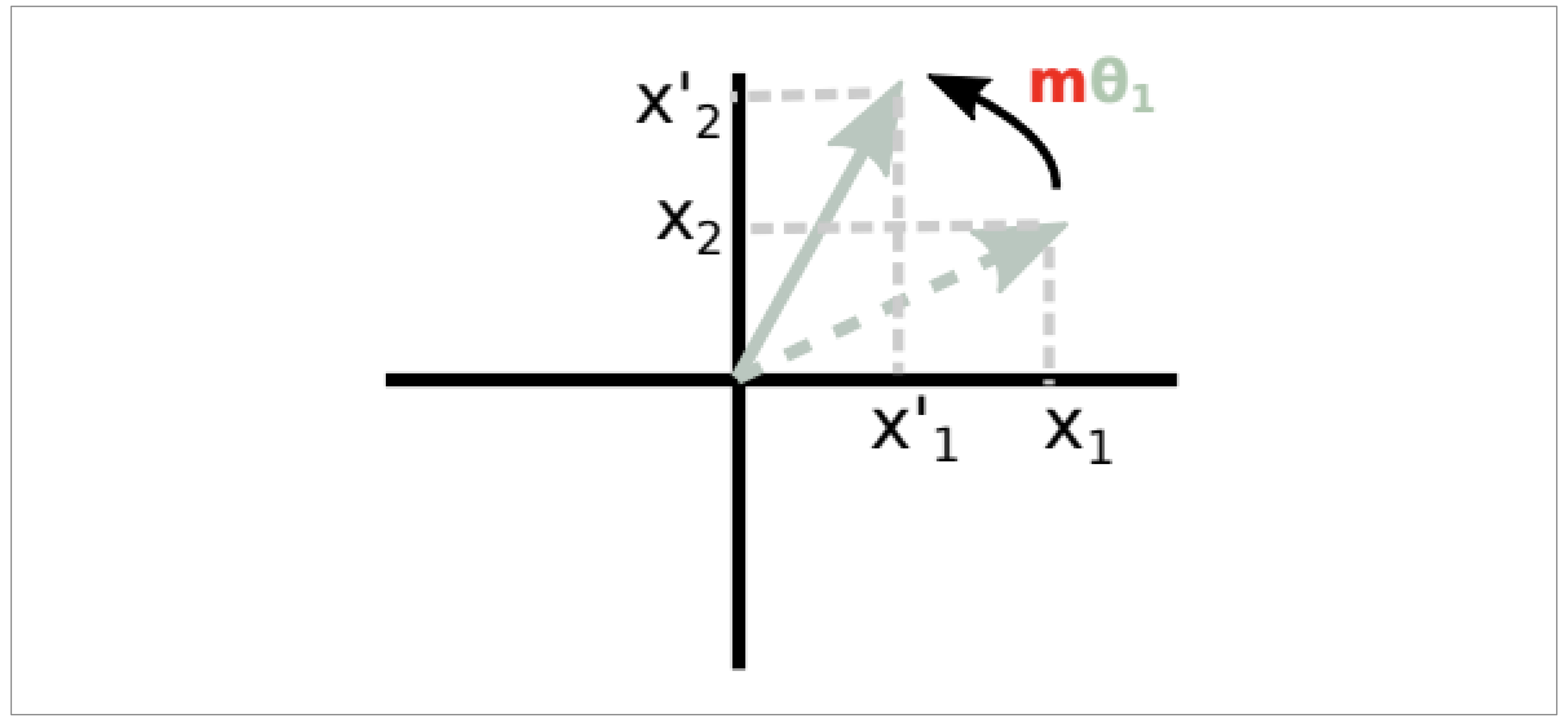
Sliding Window attention

New functionality

Since last GTC

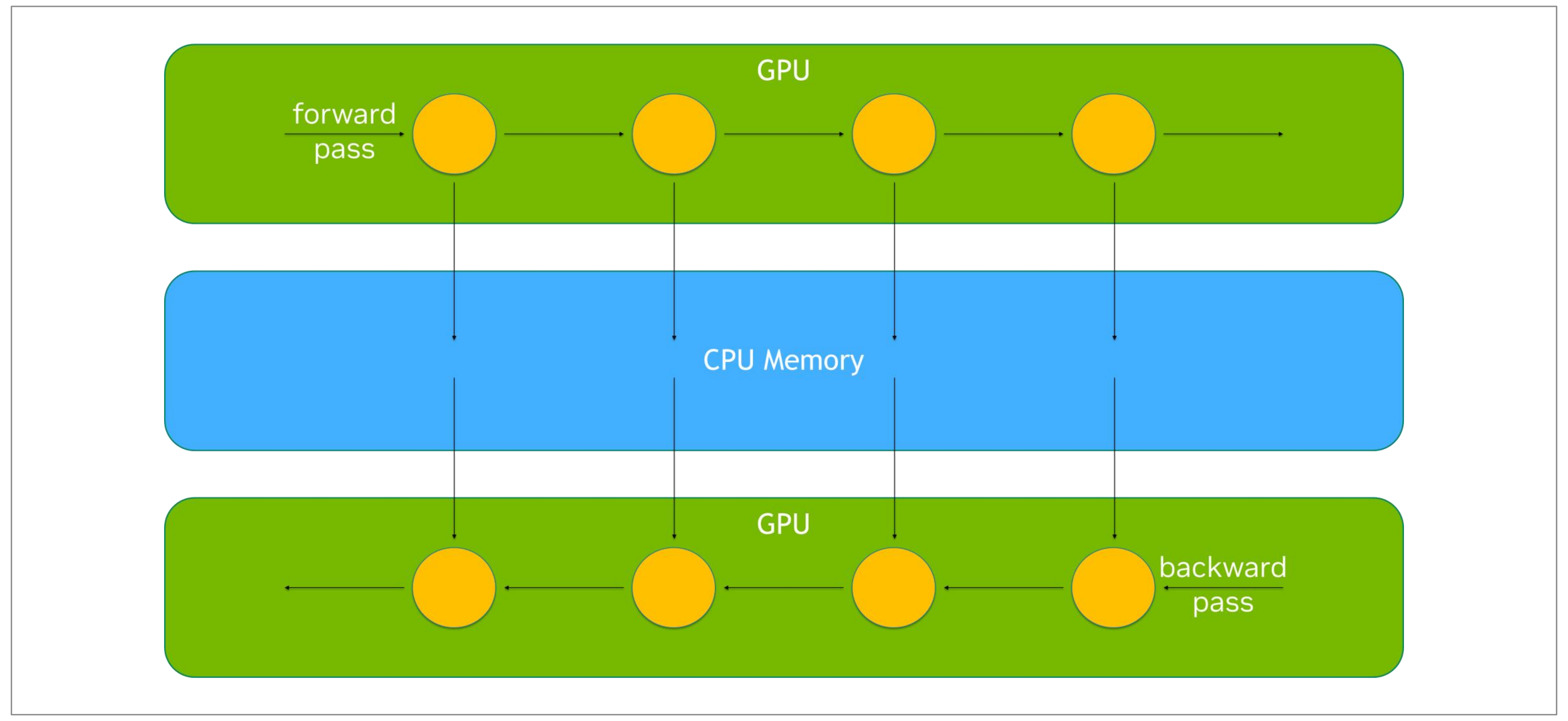


GLU activations
ReGLU, GeGLU, SwiGLU

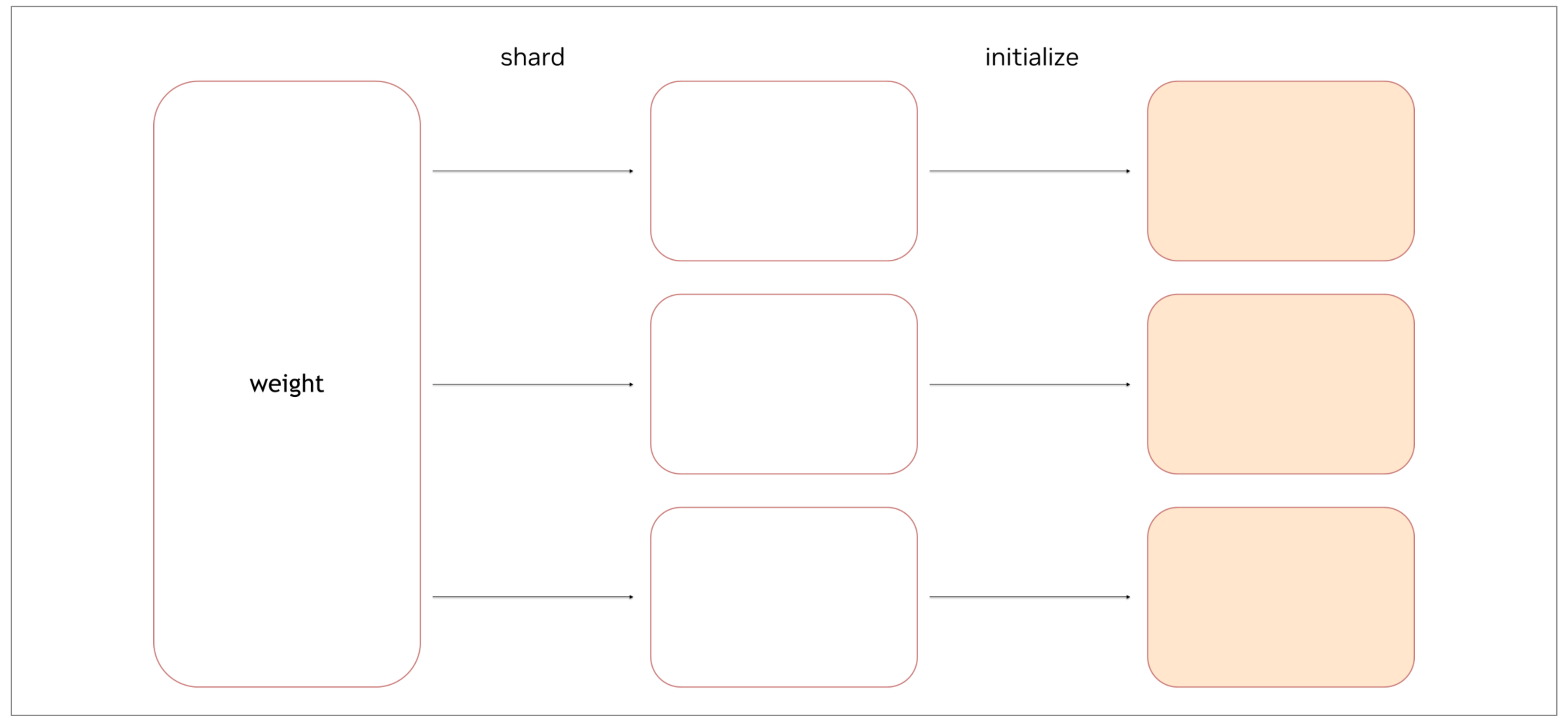


Rotary Position Embeddings
Fast fused implementation

Parallel Transformer
Falcon architecture



CPU offloading of activations



Lazy weight initialization in PyTorch
Using "meta" device

Performance improvements

Faster execution

Performance improvements

Faster execution

Performance improvements

Faster execution

Measured using CUDA 12.3, cuDNN 9.0.0.306, flash-attn==2.5.1 and Transformer Engine 1.4 on DGX H100. Other Q, K and V tensor dimensions: batch size 2, number of heads 16.

Performance improvements

More parallelism

Tensor parallelism and
Sequence parallelism
(TP4)

Performance improvements

More parallelism

Tensor parallelism and
Context parallelism
(TP2CP2)

Performance improvements

Saving memory

In [1]:
```
import transformer_engine.pytorch as te

# Create the model
model = te.Linear(128, 128)
```

In [2]:
```
print(model.weight)
```

```
Parameter containing:
tensor([[ 2.5218e-02, -3.0896e-02,  3.0437e-03,  ..., -5.0314e-02,
          8.4104e-03, -5.4243e-03],
        [ 1.2365e-02, -2.4354e-02,  5.6630e-03,  ...,  6.4632e-03,
          1.9270e-02,  8.8925e-03],
        [ 2.1083e-03,  6.1130e-05, -3.8648e-02,  ..., -7.9366e-04,
          1.1135e-02,  4.9120e-03],
        ...,
        [ 2.2397e-03, -2.5749e-02,  2.0430e-02,  ..., -2.7015e-02,
          8.1024e-03, -1.2790e-02],
        [-1.8441e-02,  4.0443e-02, -5.9455e-03,  ...,  3.2744e-02,
          7.7802e-04,  4.5076e-02],
        [-4.0411e-03, -1.4284e-03,  2.1492e-02,  ...,  9.7799e-03,
         -3.3097e-02,  5.5765e-03]], device='cuda:0', requires_grad=True)
```

- By default, Transformer Engine’s modules create their weights in high precision type (similarly to AMP)
- That is wasteful if the high precision copy is already stored in the optimizer or during inference

Performance improvements

Saving memory

In [1]:
```python
import transformer_engine.pytorch as te

# Store model weights in FP8
with te.fp8_model_init(enabled=True):
    model = te.Linear(128, 128)
```

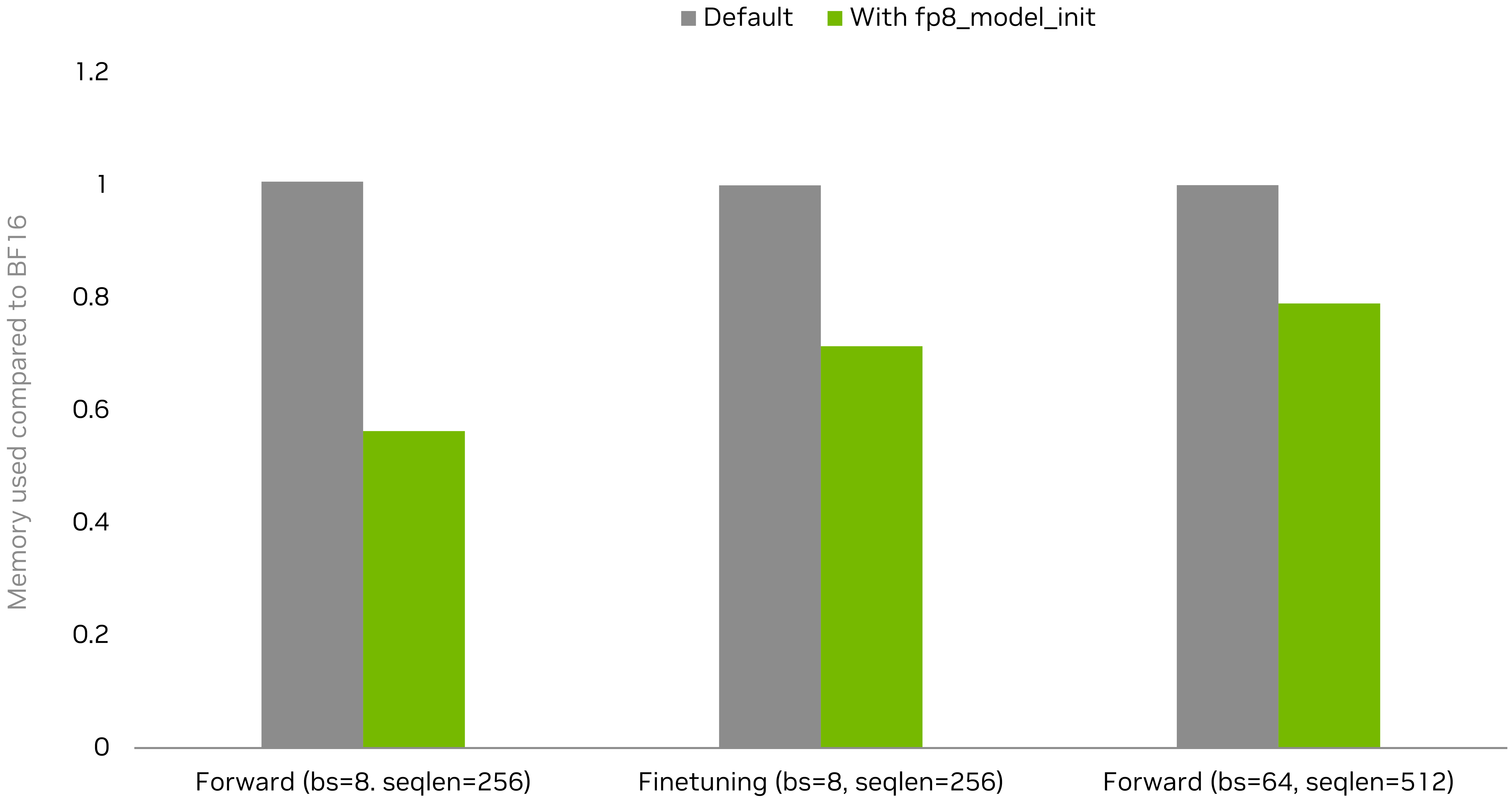
In [2]:
```python
print(model.weight)
```

```
Float8Tensor(fp8_dtype=DType.kFloat8E4M3, scale_inv=1.0, data=tensor([[ 0.0234, -0.0020, -0.0059,  ...,  0.009
8,  0.0547,  0.0176],
        [ 0.0098,  0.0020, -0.0059,  ..., -0.0059,  0.0195,  0.0098],
        [-0.0430,  0.0195, -0.0039,  ...,  0.0078, -0.0352,  0.0234],
        ...,
        [-0.0469,  0.0117, -0.0098,  ..., -0.0469, -0.0020,  0.0176],
        [-0.0215,  0.0039, -0.0195,  ..., -0.0215, -0.0430, -0.0293],
        [-0.0039, -0.0293,  0.0137,  ..., -0.0137,  0.0020, -0.0039]],
       device='cuda:0', grad_fn=<_FromFloat8FuncBackward>))
```

- `fp8_model_init` context manager tells Transformer Engine's modules to create their weights in FP8 only, without high precision copy
- Enables full memory savings from FP8

Performance improvements

Saving memory



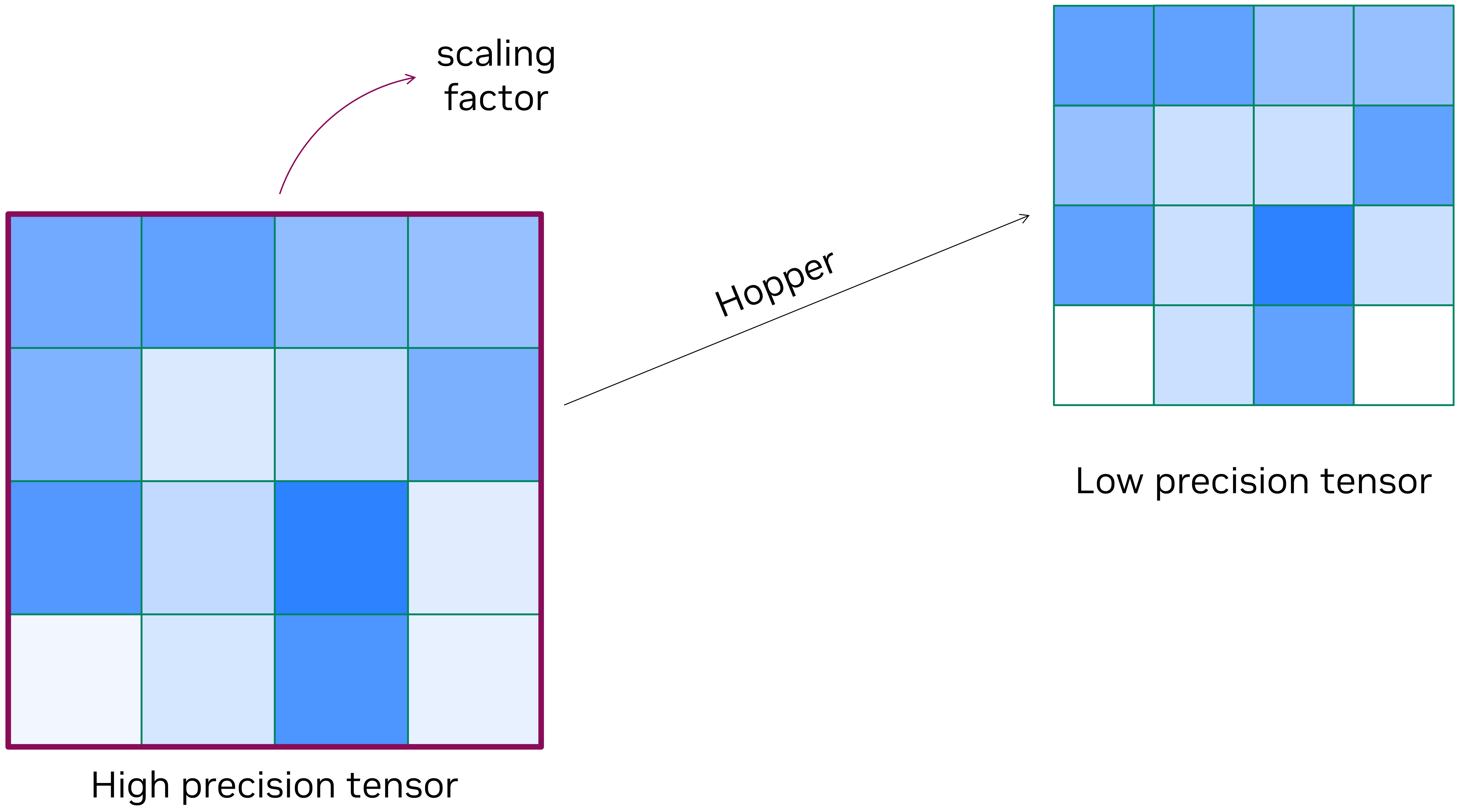
Measured using Llama 7B model on a single H100 80GB GPU

Future of TE

Future of TE

Blackwell

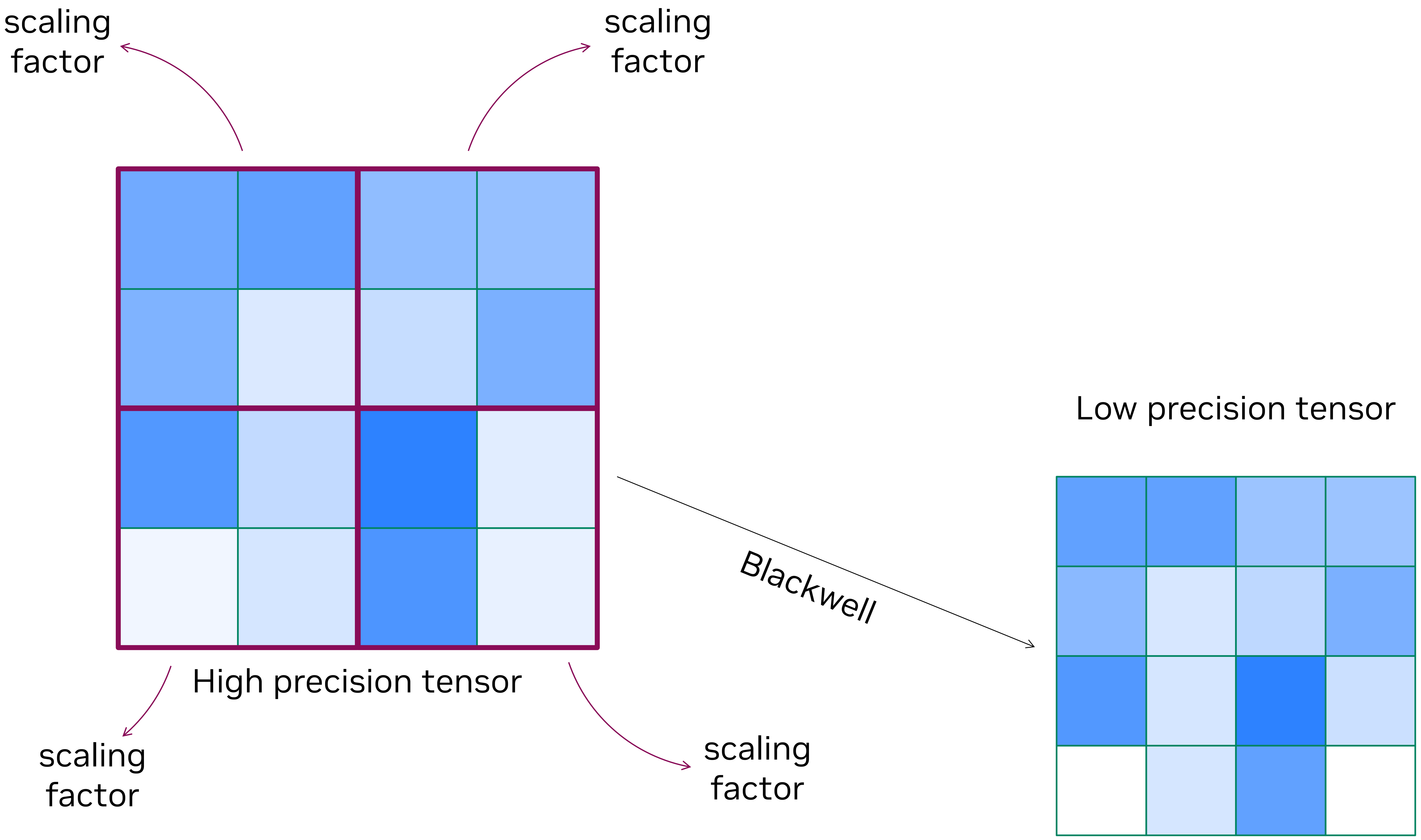
- Blackwell introduces the second-generation Transformer Engine with FP8, FP6 and FP4



Future of TE

Blackwell

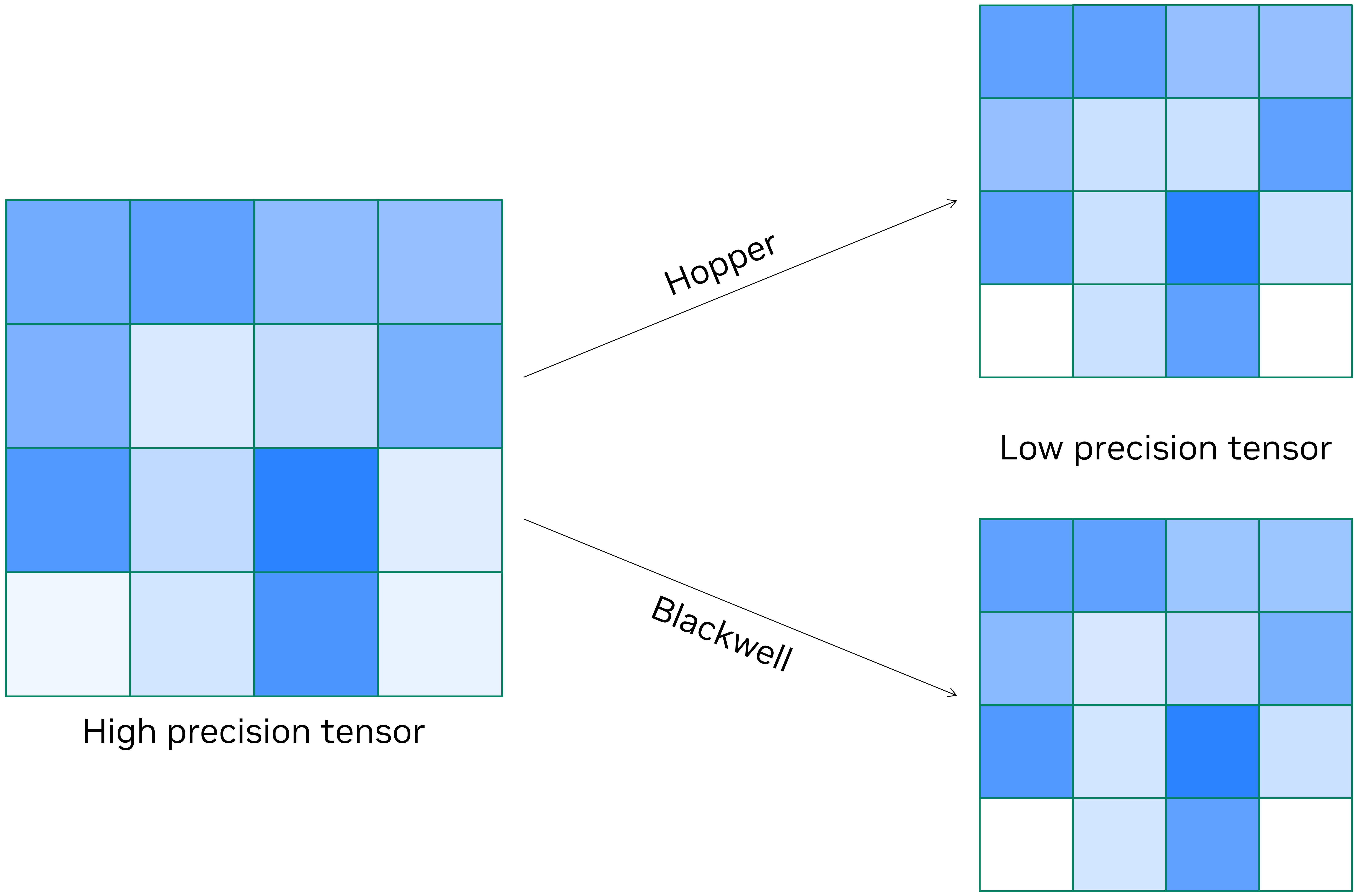
- Blackwell introduces the second-generation Transformer Engine with FP8, FP6 and FP4



Future of TE

Blackwell

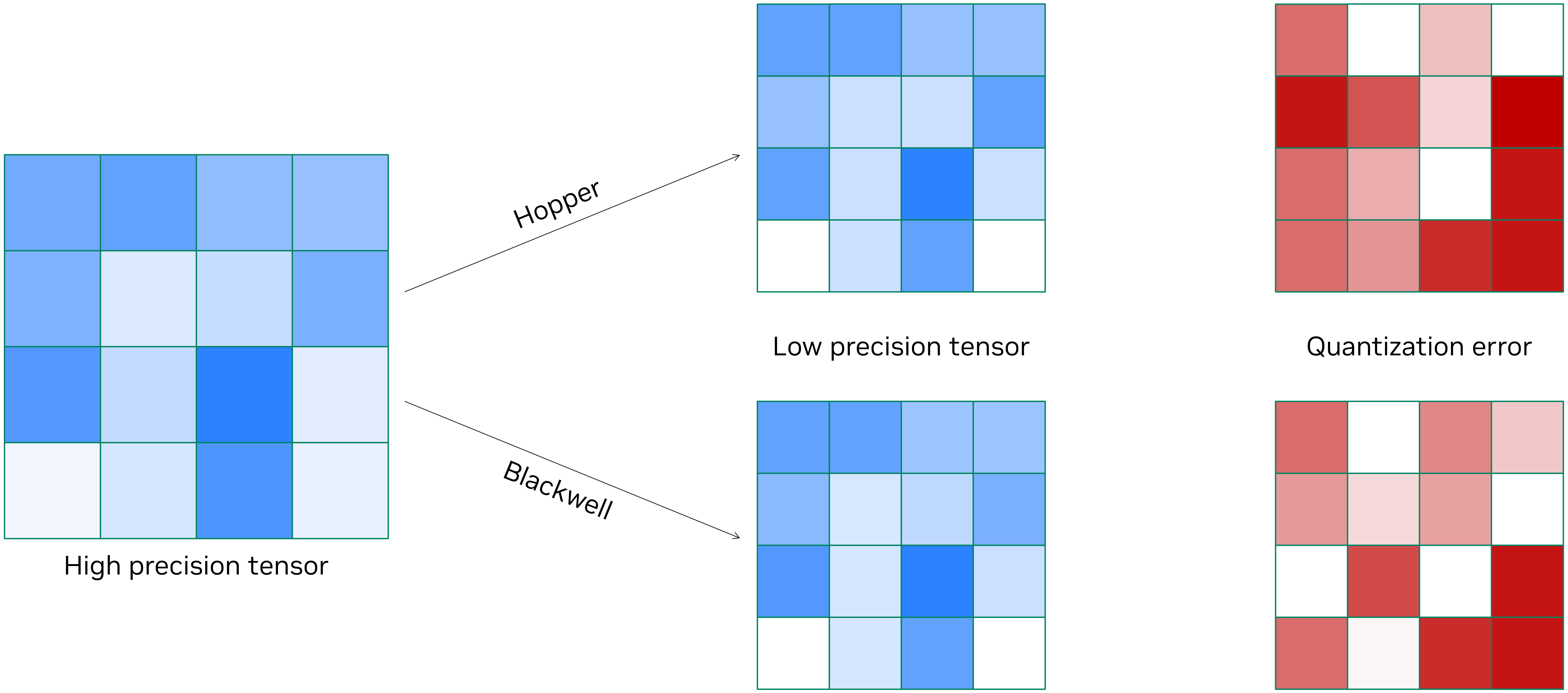
- Blackwell introduces the second-generation Transformer Engine with FP8, FP6 and FP4



Future of TE

Blackwell

- Blackwell introduces the second-generation Transformer Engine with FP8, FP6 and FP4



Future of TE

Beyond Transformers

- Focus on large context length makes regular attention a bottleneck, as it scales with a square of the sequence length.
- Many proposed techniques to overcome this – sliding window attention, sparse attention, dilated attention…
- State space models (SSMs, e.g. Mamba) are an interesting new family of language models that aims to tackle this issue by replacing attention.
- On the TE side, we want to aid that exploration with optimized building blocks for SSMs.

Future of TE

And more…

- Full CUDA graphs support
- Full support for FSDP in pyTorch
- FP8 optimizer
- Support for FP8 in MoE
- Better communication overlap in JAX
- More modular API
- More performance optimizations

NVIDIA