DeepSeekV3-FP8 Training

XiaoTonghuan, GongPing

USTC, CHINA
ADSLAB
先进数据系统实验室

Outline

- Background
- Challenges
- Design
- Implementation

Outline

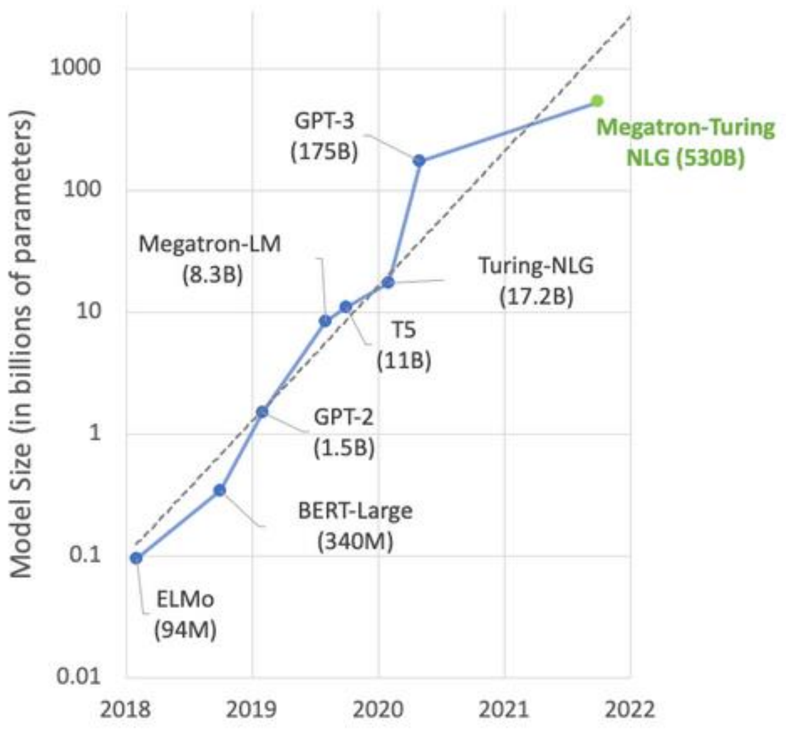
- Background
- Challenges
- Design
- Implementation

Background

- Why FP8 training?
  - Save memory
  - computing power

Background

- Why FP8 training?
  - Save memory



1B parameters occupy
4GB storage (32bit a parameter)

Example: GPT3(175B)
FP32: 700GB
FP8 : 175GB
Save 75% memory

Background

- Why FP8 training?
  - computing power

NVIDIA H100 specifications (vs. NVIDIA A100)

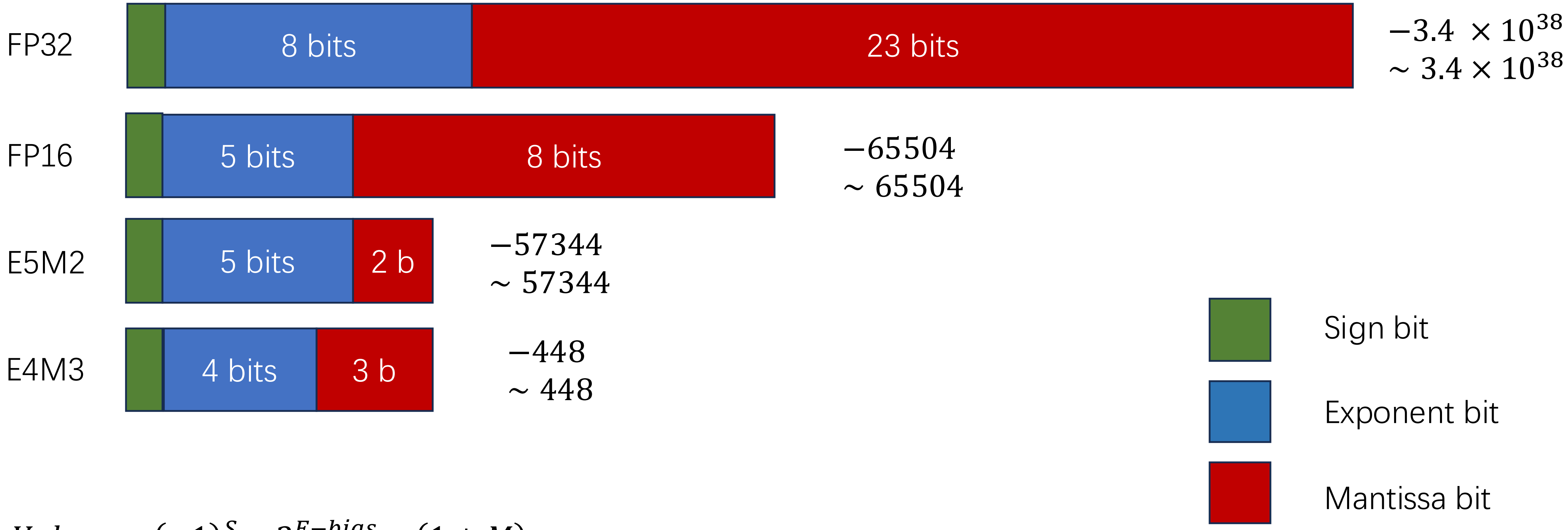
Data type	H100-SXM5 (TFLOPS)	A100-SXM4 (TFLOPS)	Difference
TF32	494	156	3.2x
BF16	989	312	3.2x
FP16	989	312	3.2x
FP8	1979	-	6.3x (vs BF16)
Bandwidth (GB/s)	3350	2039	1.6x

Table 1: FLOPS and memory bandwidth comparison between the NVIDIA H100 and NVIDIA A100. While there are 3x-6x more total FLOPS, real-world models may not realize these gains.

Computing power doubles

Background

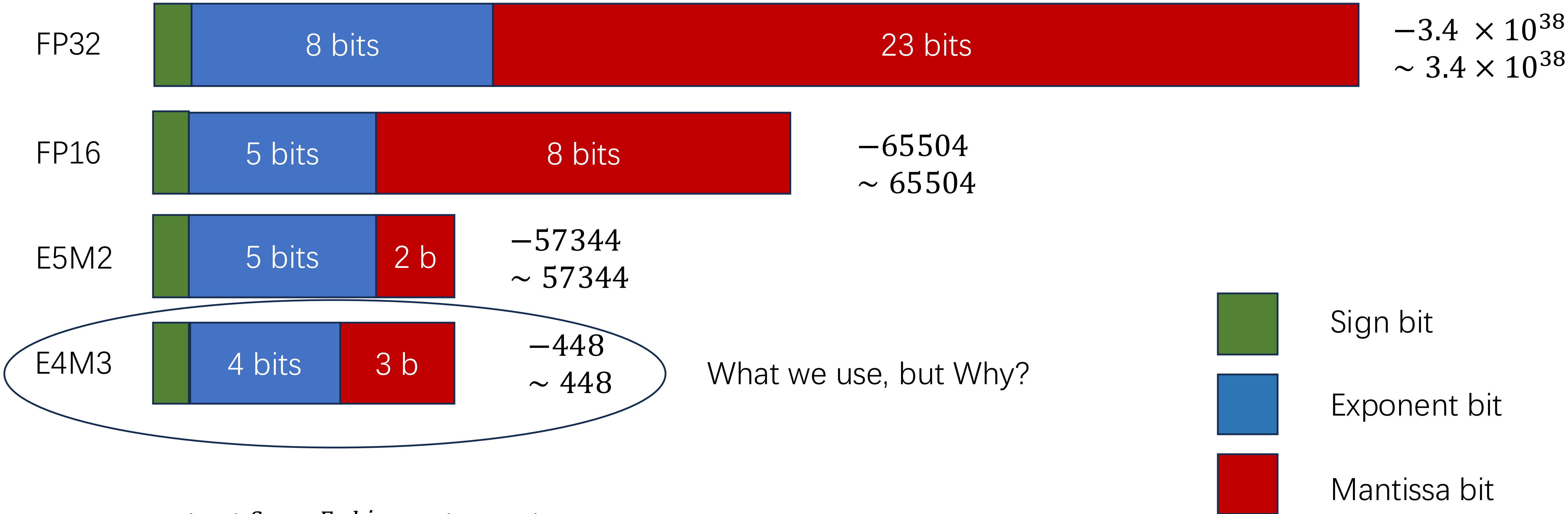
- Data Types
  - FP32, FP16, FP8(E5M2, E4M3)



$$Value = (-1)^S \times 2^{E-bias} \times (1 + M)$$

Background

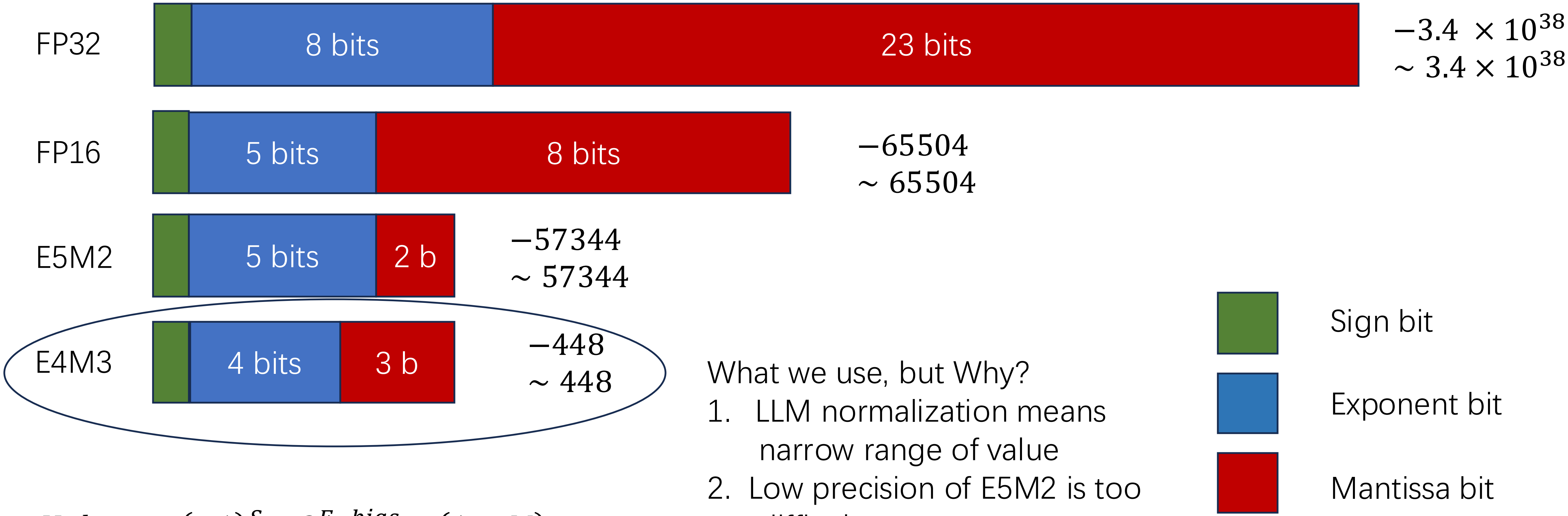
- Data Types
  - FP32, FP16, FP8(E5M2, E4M3)



$$Value = (-1)^S \times 2^{E-bias} \times (1 + M)$$

Background

- Data Types
  - FP32, FP16, FP8(E5M2, E4M3)



What we use, but Why?

1. LLM normalization means narrow range of value
2. Low precision of E5M2 is too difficult to overcome

$$Value = (-1)^S \times 2^{E-bias} \times (1+M)$$

Outline

- Background
- Challenges
- Design
- Implementation

Difficulty in using FP8(E4M3)

- FP8 low precision
  - Cannot satisfy all precision requirements in training
    - Precision problem is too hard to overcome, so not all FP8
      - Where to use FP8, Where to maintain original format(BF16,FP32)
    - Narrow range cause overflow/underflow in Conversion
      - Conversion between different Floating point numbers

Outline

- Background
- Challenges
- Design
  - How to handle mixed precision
    - Where to use FP8 ?
    - Conversion between different format
- Implementation
  - Deep GEMM
  - Mixed precision Framework

Outline

- Background
- Challenges
- Design
  - How to handle mixed precision
    - Where to use FP8 ?
    - Conversion between different format
- Implementation
  - Deep GEMM
  - Mixed precision Framework

Where to use FP8 ?

- Determine where to use FP8/BF16/FP32 by careful investigations
  - BF16/FP32
    - Embedding module(low utilization rate)
    - Output head(Low utilization rate)
    - MoE gating modules(only 1%~5% overhead of MoE)
    - normalization operators(A high precision requirement (e.g., 1e-6))
    - attention operators(A high precision requirement)
    - Weights(Weight update use FP32, Computation use FP8)
    - weight gradients(FP32, low computing power consume)
    - optimizer states(BF16, BF16 is enough for Optimizer DeepSeek experiment proof)
  - FP8
    - MLP on MoE
    - MLP before/after Attention

Where to use FP8 ?

- Determine where to use FP8/BF16/FP32 by careful investigations
  - BF16/FP32
    - Embedding module(low utilization rate)
    - Output head(Low utilization rate)
    - MoE gating modules(only 1%~5% overhead of MoE)
    - normalization operators(A high precision requirement (e.g., 1e-6))
    - attention operators(A high precision requirement)
    - Weights(Weight update use FP32, Computation use FP8)
    - weight gradients(FP32, low computing power consume)
    - optimizer states(BF16, BF16 is enough for Optimizer DeepSeek experiment proof)
  - FP8
    - MLP on MoE
    - MLP before/after Attention

What important is the law

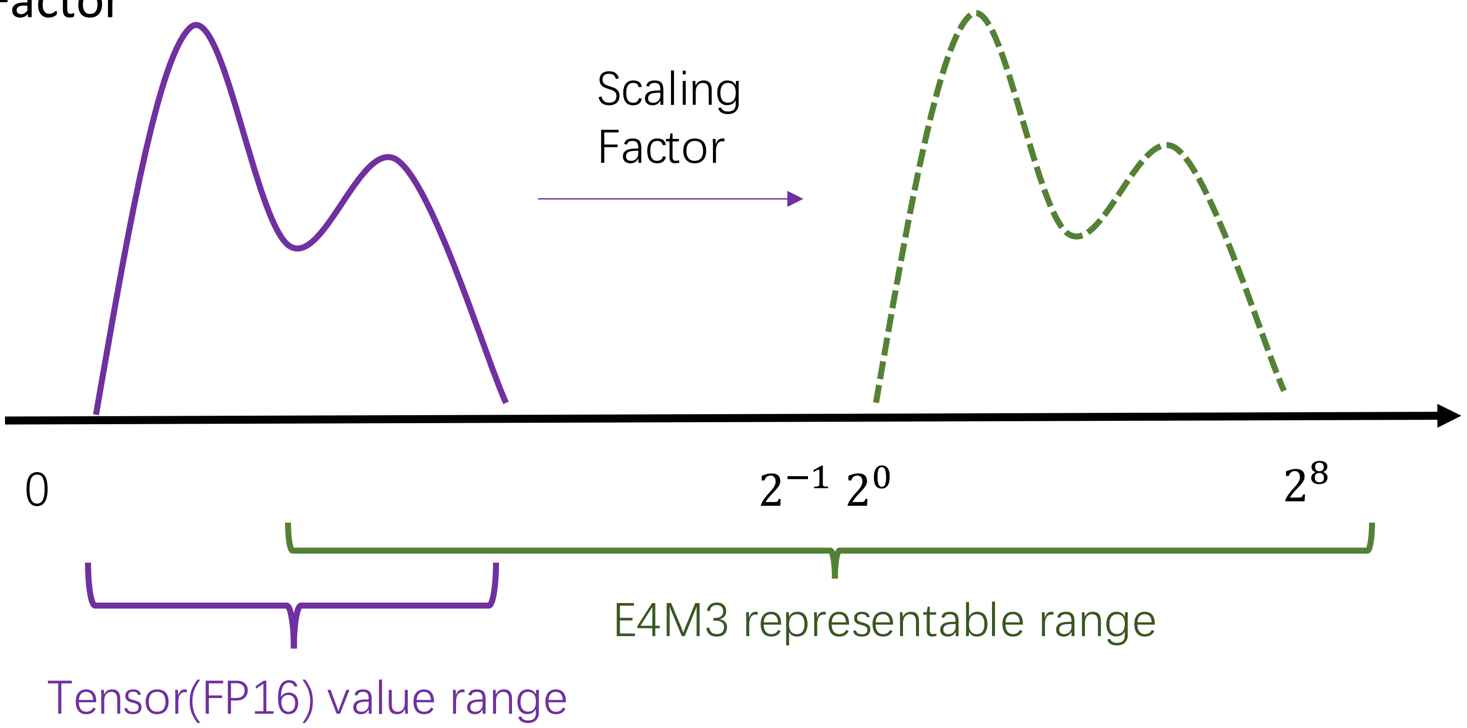
1. Not used in scenarios with low computational demands
2. Not used in applications requiring high precision

Outline

- Background
- Challenges
- Design
  - How to handle mixed precision
    - Where to use FP8 ?
    - Conversion between different format
- Implementation
  - Deep GEMM
  - Mixed precision Framework

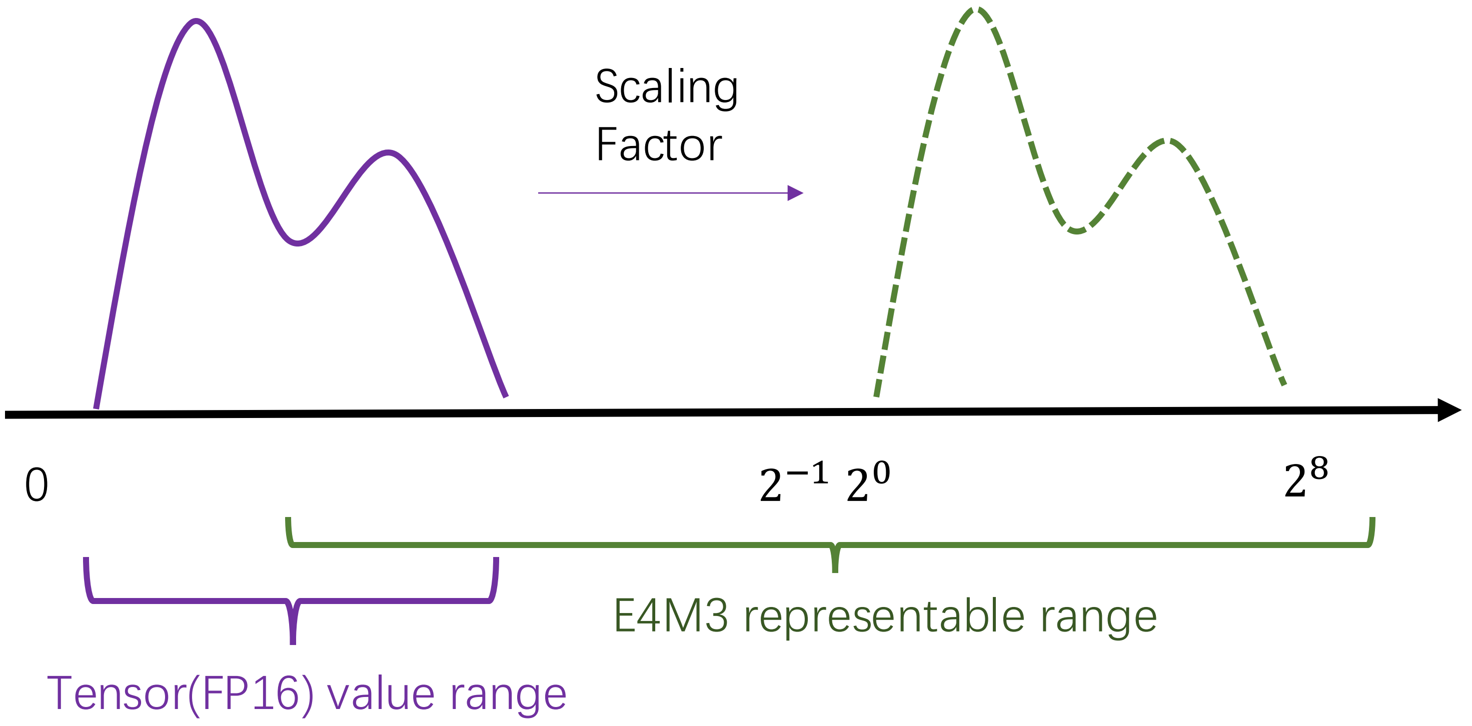
Overflow/Underflow

- Overflow/Underflow
  - What's the problem
  - Scaling Factor



Conversion between different format

- High precision to Low precision
  - Scaling (Make sure the value is within the FP8 range)
  - Cast(type conversion)



Conversion between different format

- High precision to Low precision
  - Scaling (Make sure the value is within the FP8 range)
  - Cast (Type casting)
- Low precision to High precision
  - Type casting to a wider scope
  - multiply Scaling Factor

Scaling Factor

- Use Dynamic Scaling Factor
  - Dynamic: Obtained during calculation
  - Scaling Factor: max(abs(x)) / MAX_E4M3

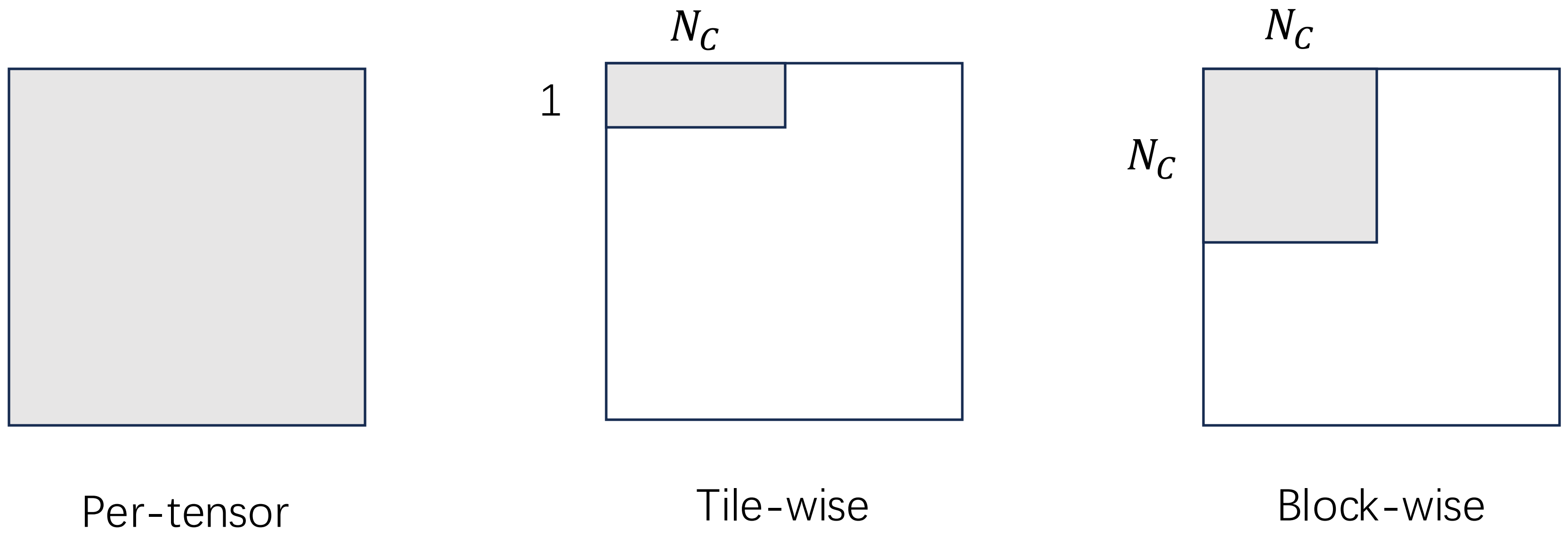
Scaling Factor

- Use Dynamic Scaling Factor
  - Dynamic: Obtained during calculation
  - Scaling Factor: max(abs(x)) / MAX_E4M3

What is X ?
Entire tensor or part of tensor ?

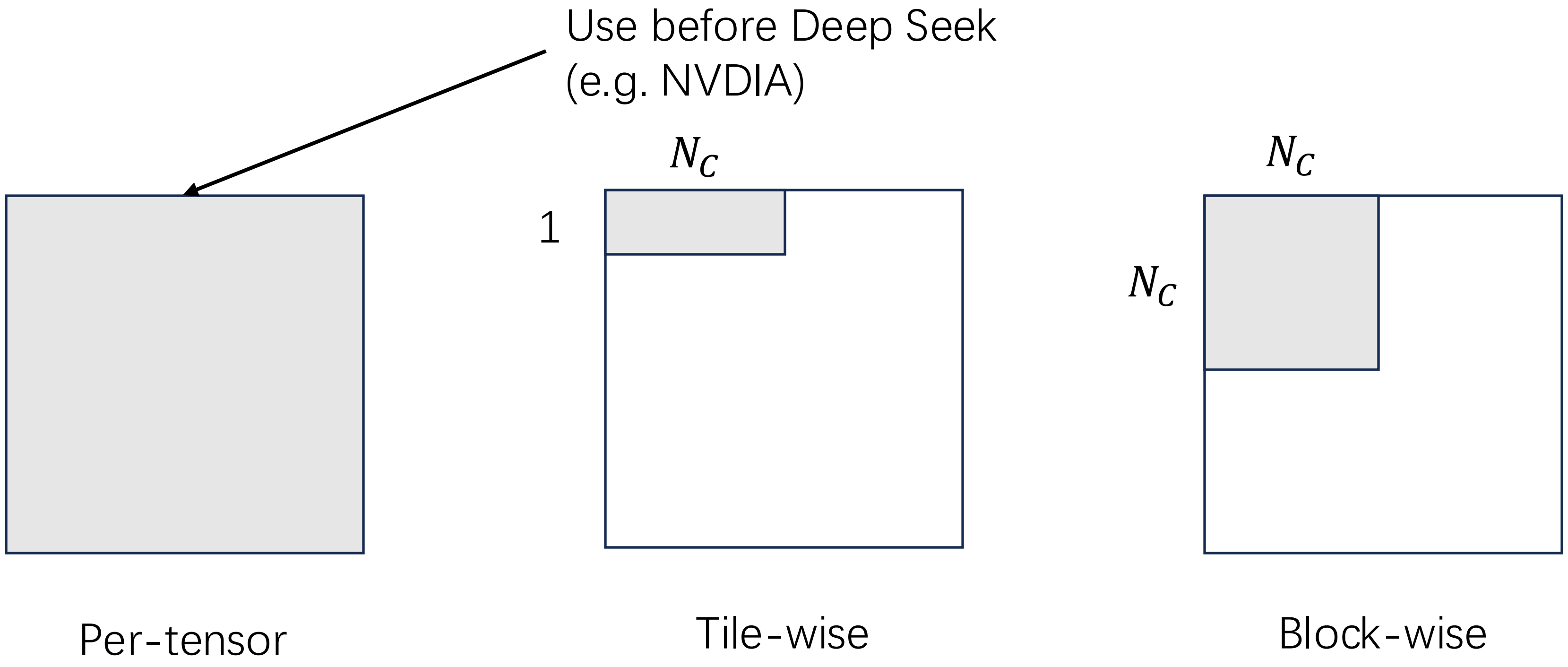
Scaling Factor

- What is X ? (Select an area to find scaling factor)



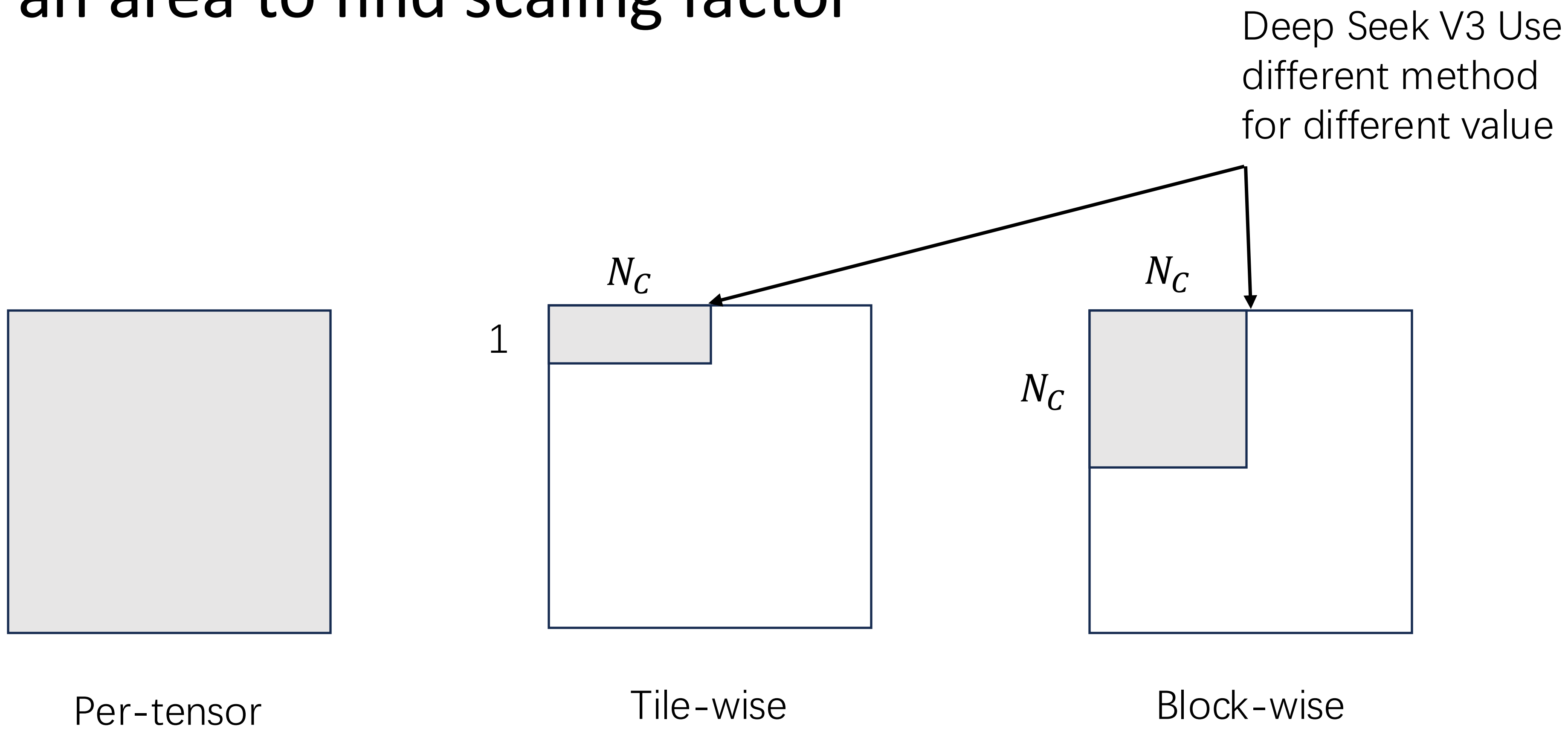
Scaling Factor

- Select an area to find scaling factor



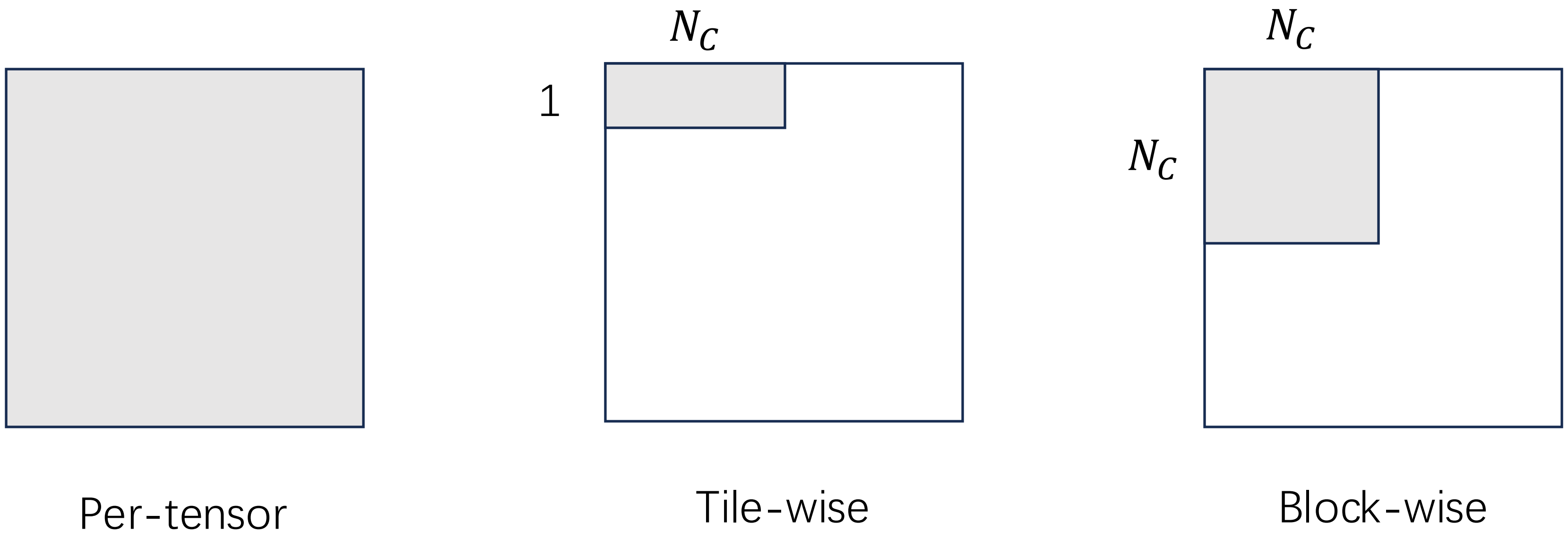
Scaling Factor

- Select an area to find scaling factor



Scaling Factor

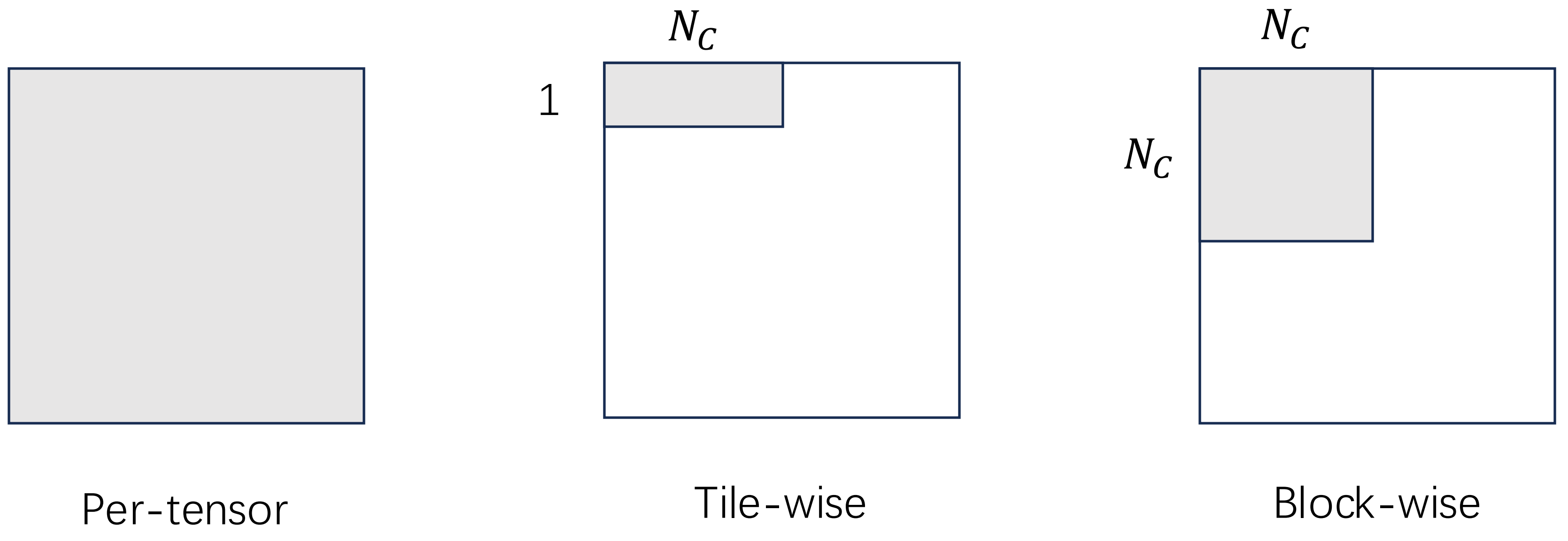
- Select an area to find scaling factor
  - ◆ Activation: tile-wise
  - ◆ Weight: block-wise



Scaling Factor

- Select an area to find scaling factor
  - Activation: tile-wise
  - Weight: block-wise

Different distribution of outlier between activation and weight



Outline

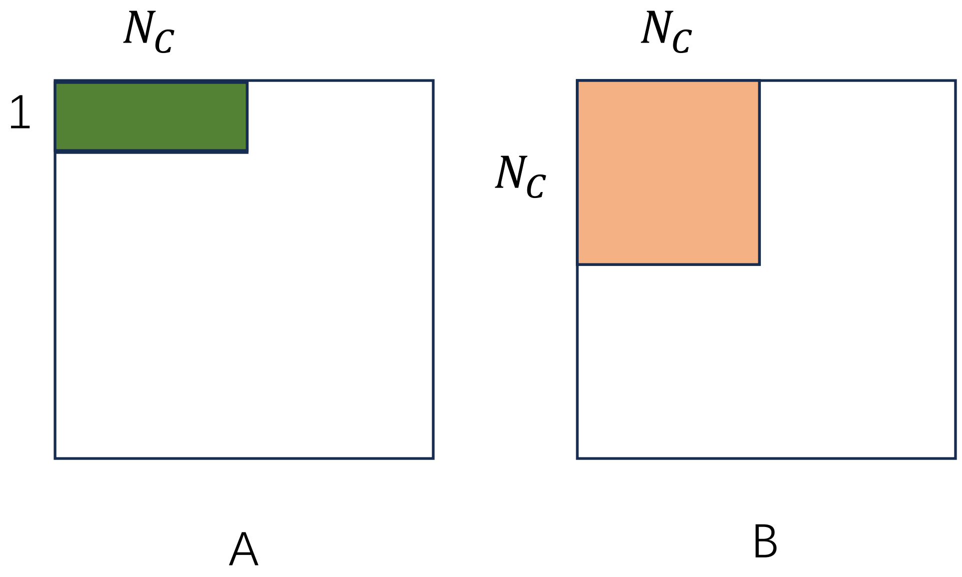
- Background
- Challenges
- Design
  - How to handle mixed precision
    - Where to use FP8 ?
    - Conversion between different format
- Implementation
  - Deep GEMM
  - Mixed precision Framework
  - Other

GeMM

- GeMM(General Matrix-Matrix Multiplication)
- A @ B = C(FP8,FP8, FP32)
  - Note: A, B with Scaling Factor
  - Blocks of matrix process MMA
    - MMA operation (wgmma instruction, FP32)
  - Merge block results
    - Multiply scaling factor
    - Combine partial sum
  - Use two warpgroup(one execute mma operation, another merge)

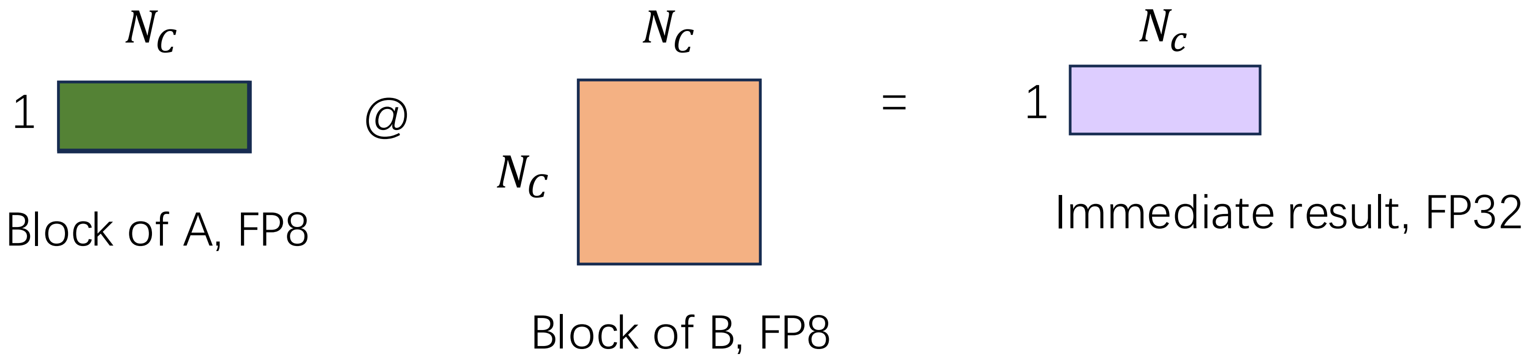
GeMM

- GeMM(General Matrix-Matrix Multiplication)
- A @ B = C
  - Blocks of matrix process MMA



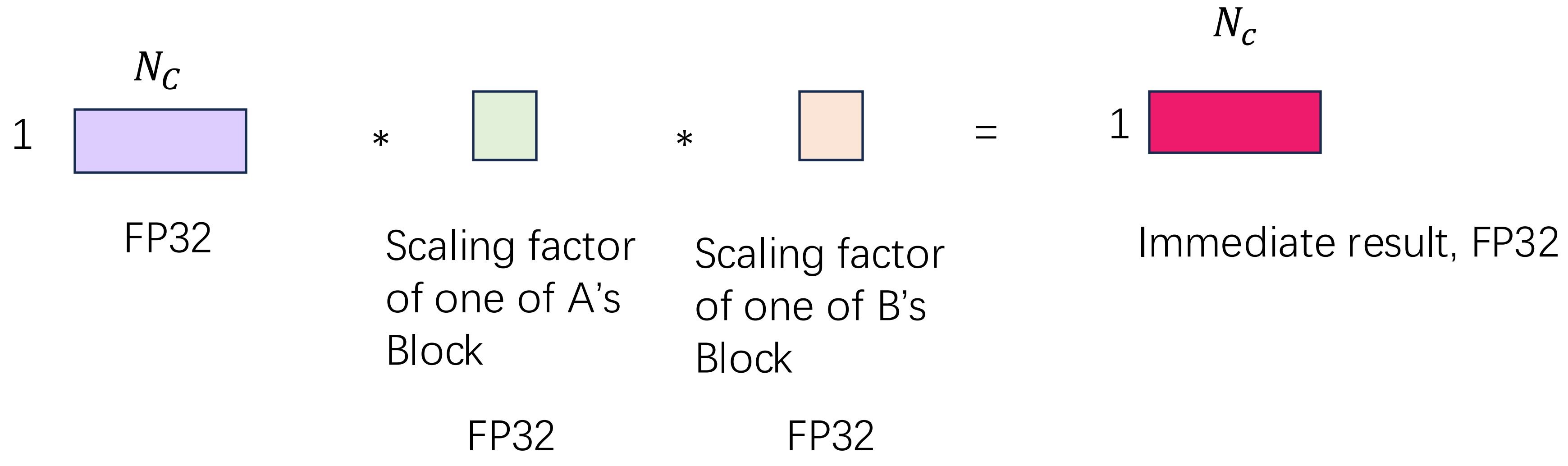
GeMM

- GeMM(General Matrix-Matrix Multiplication)
- A @ B = C
  - Blocks of matrix process MMA



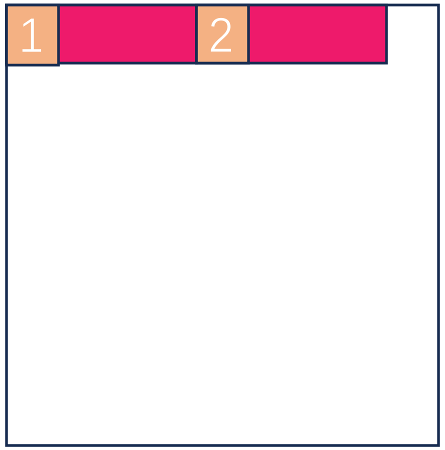
GeMM

- GeMM(General Matrix-Matrix Multiplication)
- A @ B = C
  - Promotion and Merge block results
    - Multiply scaling factor

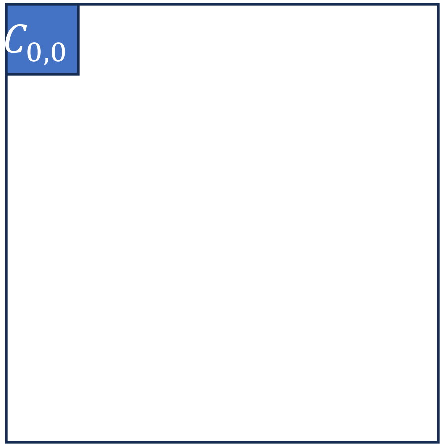


GeMM

- GeMM(General Matrix-Matrix Multiplication)
- A @ B = C
  - Promotion and Merge block results
    - Combine partial sum



Immediate result



C

$Add\ 1{,}2\ \ldots\ \text{to}\ C_{0,0}$

Outline

- Background & Challenges
- Design
  - Overflow/Underflow
  - Where to use FP8 ?
  - Conversion between different floating point
- Implementation
  - Deep GEMM
  - Mixed precision Framework
  - Other

Mixed precision Framework

- forward pass
- activation backward pass
- weight backward pass

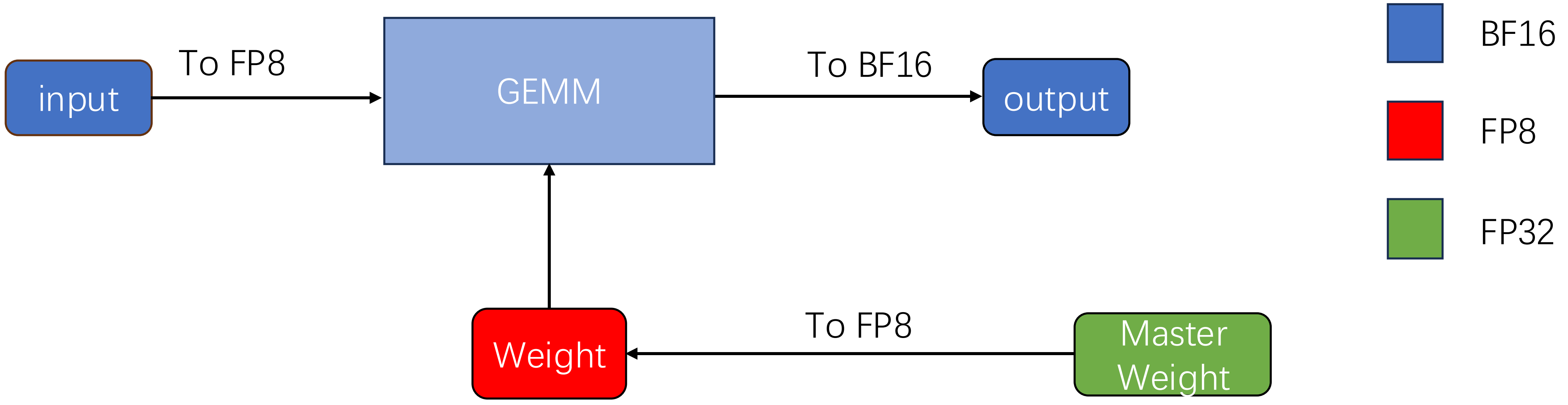
Mixed precision Framework

- forward pass
- activation backward pass
- weight backward pass

This partitioning is based on the fact that
backward passes rely on two key matrix multiplications.

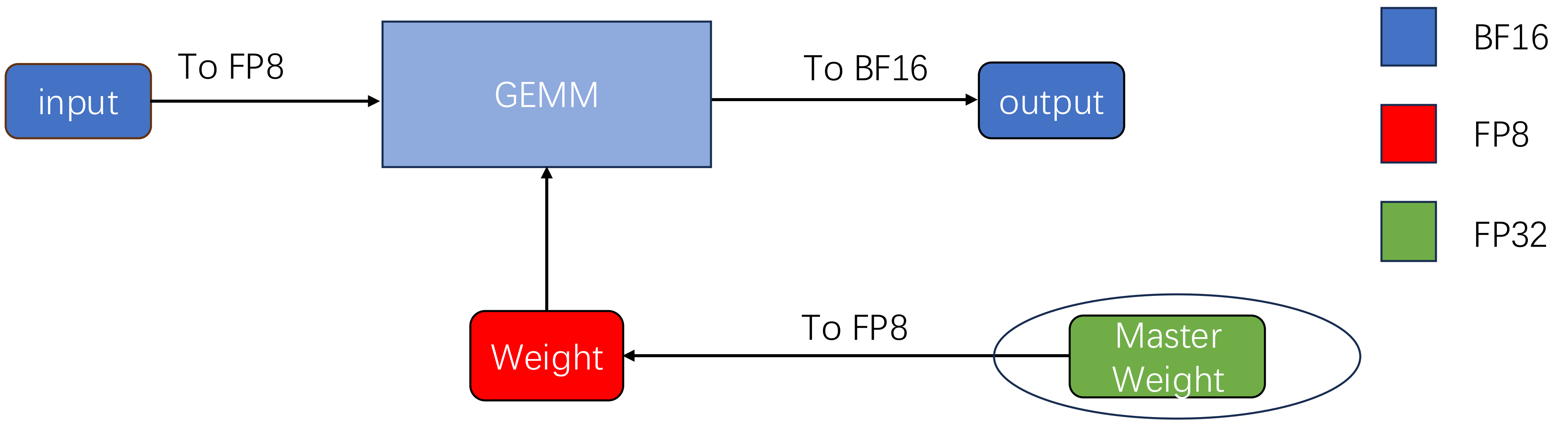
Mixed precision Framework

- Mixed precision Framework
  - forward pass



Mixed precision Framework

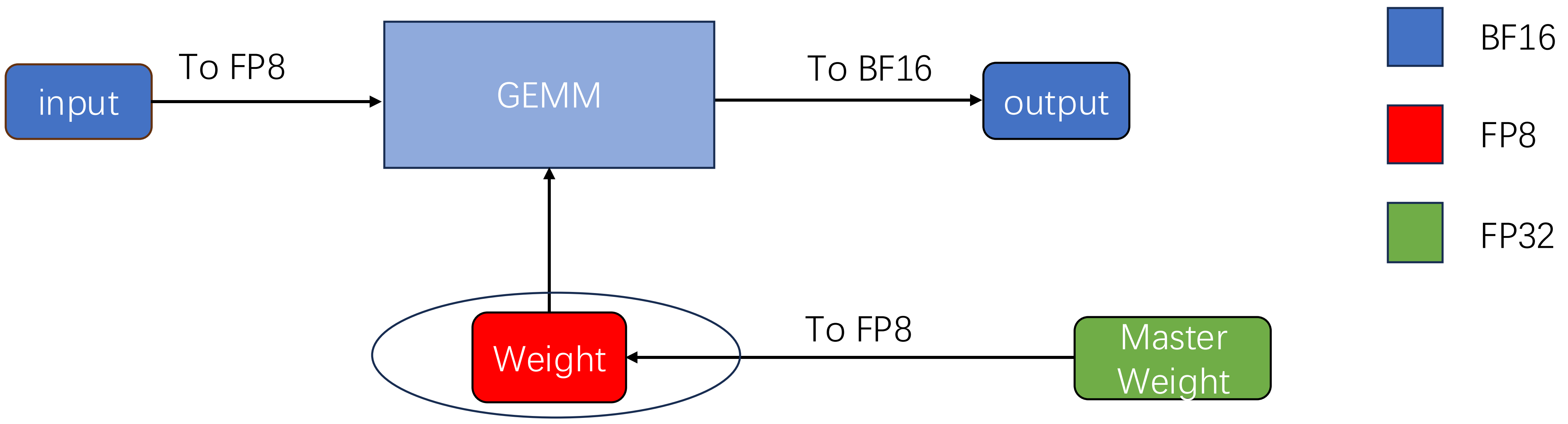
- Mixed precision Framework
  - ◆ forward pass



Saving and updating weights are performed on the master weights

Mixed precision Framework

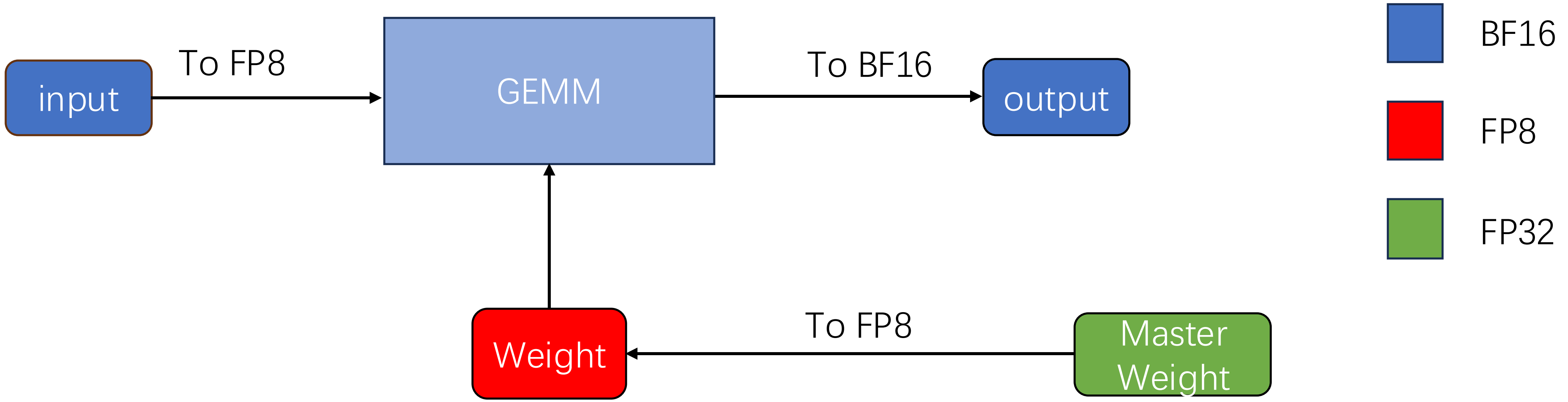
- Mixed precision Framework
  - forward pass



FP8 weight is used for computations.

Mixed precision Framework

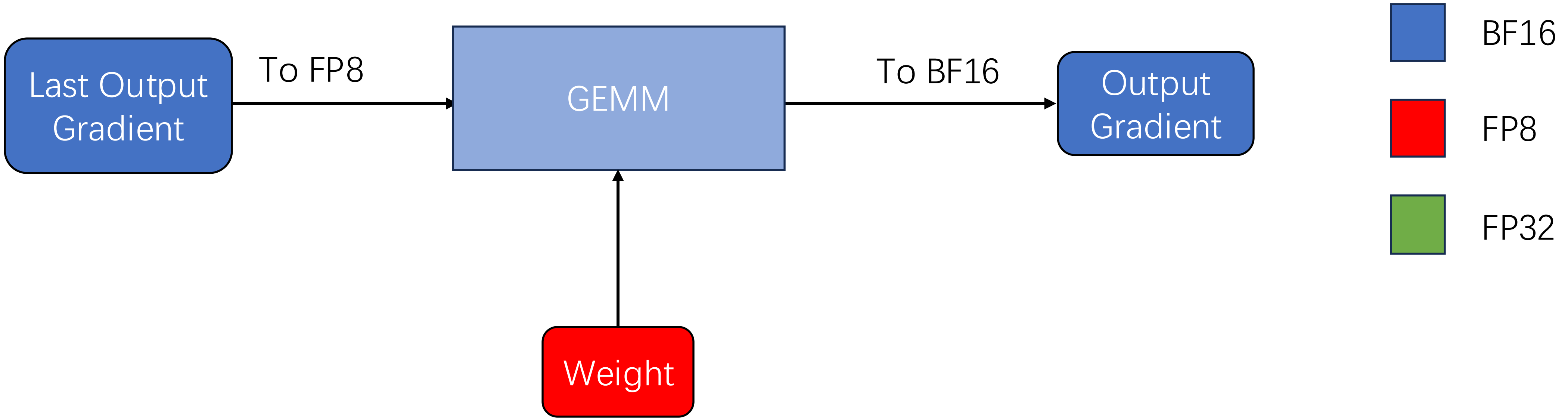
- Mixed precision Framework
  - forward pass



Directly updating weights in FP8 can lead to precision loss and vanishing/exploding gradient issues.

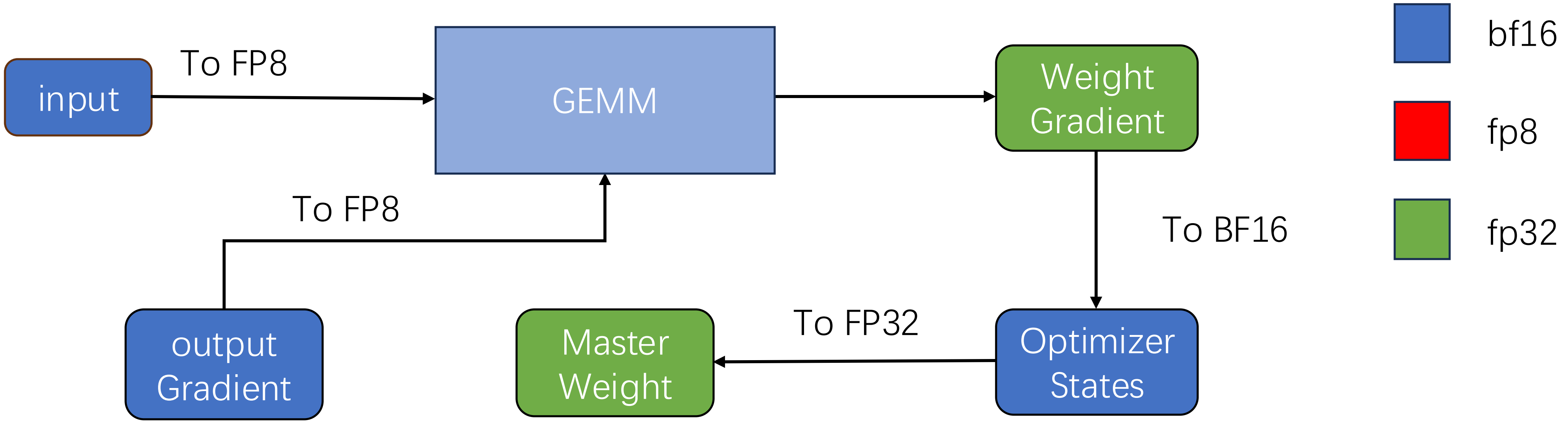
Mixed precision Framework

- Mixed precision Framework
  - activation backward pass



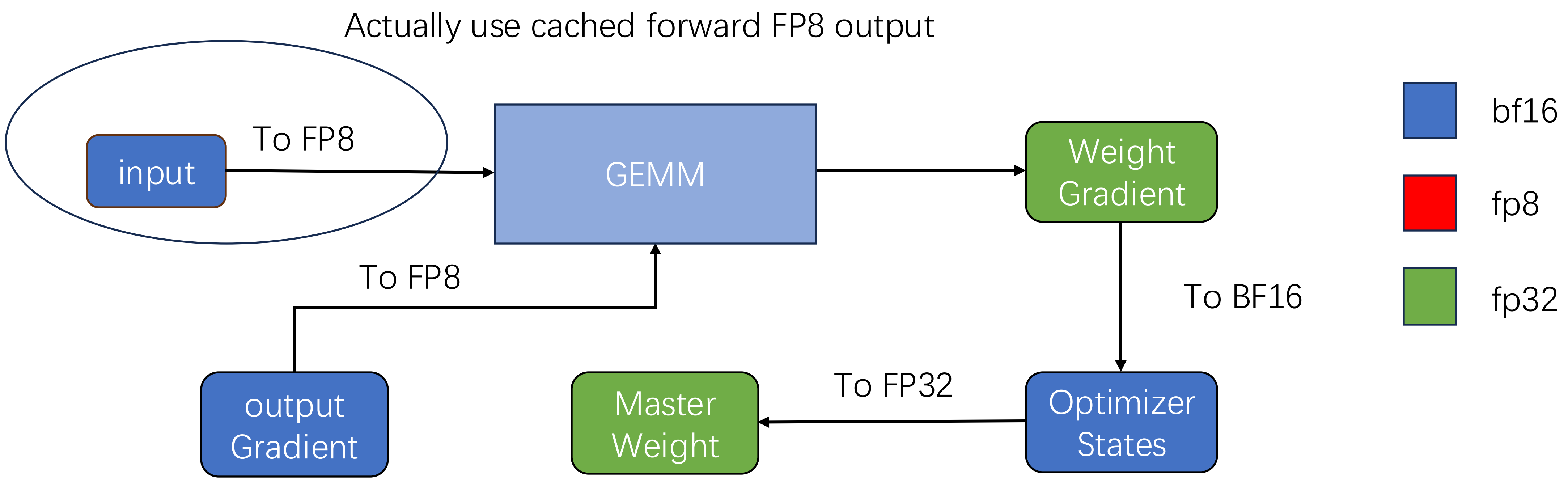
Mixed precision Framework

- Mixed precision Framework
  - weight backward pass



Mixed precision Framework

- Mixed precision Framework
  - weight backward pass



Outline

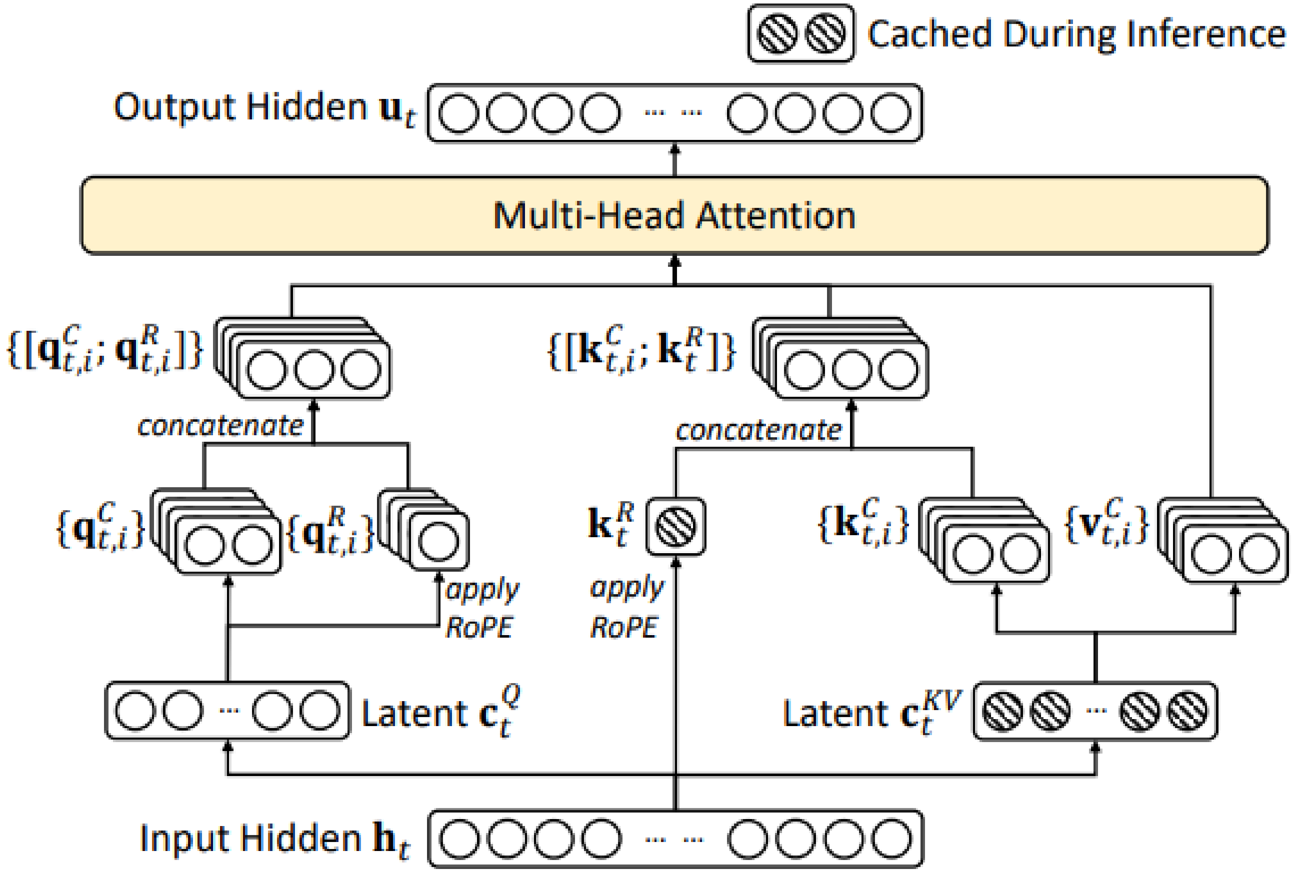
- Background & Challenges
- Design
  - Overflow/Underflow
  - Where to use FP8 ?
  - Conversion between different floating point
- Implementation
  - Deep GEMM
  - Mixed precision Framework
  - Other

Other

- Low-Precision Activation(Compared to BF16 activation)
  - When backward pass Cached FP8 Activation, special considerations are taken on several operators for low-cost high-precision training
  - Attention’s backward pass
  - Input of SwiGLU (MoE)

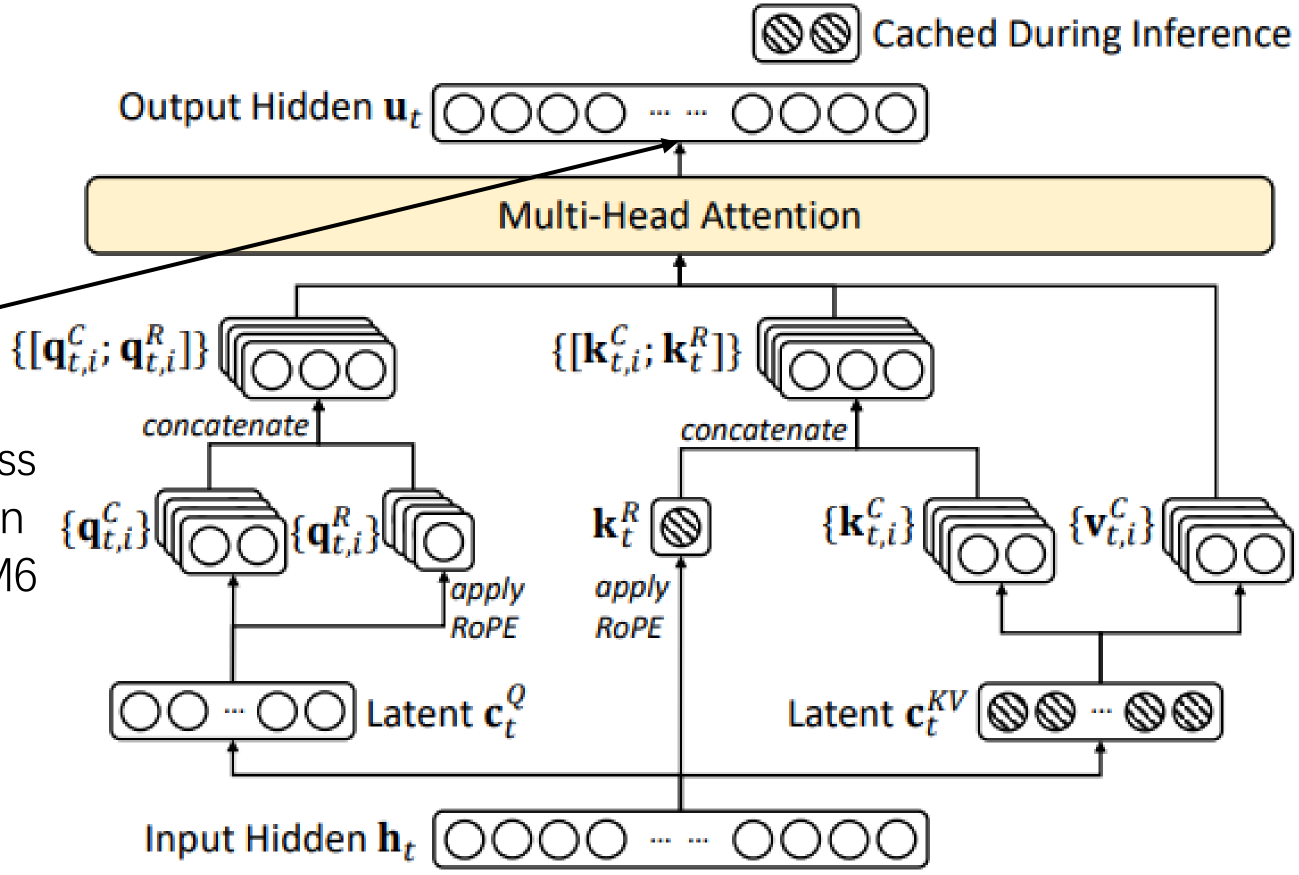
Other

- Low-Precision Activation(Compared to BF16 activation)
  - Attention's backward pass



Other

- Low-Precision Activation(Compared to BF16 activation)
  - Attention’s backward pass



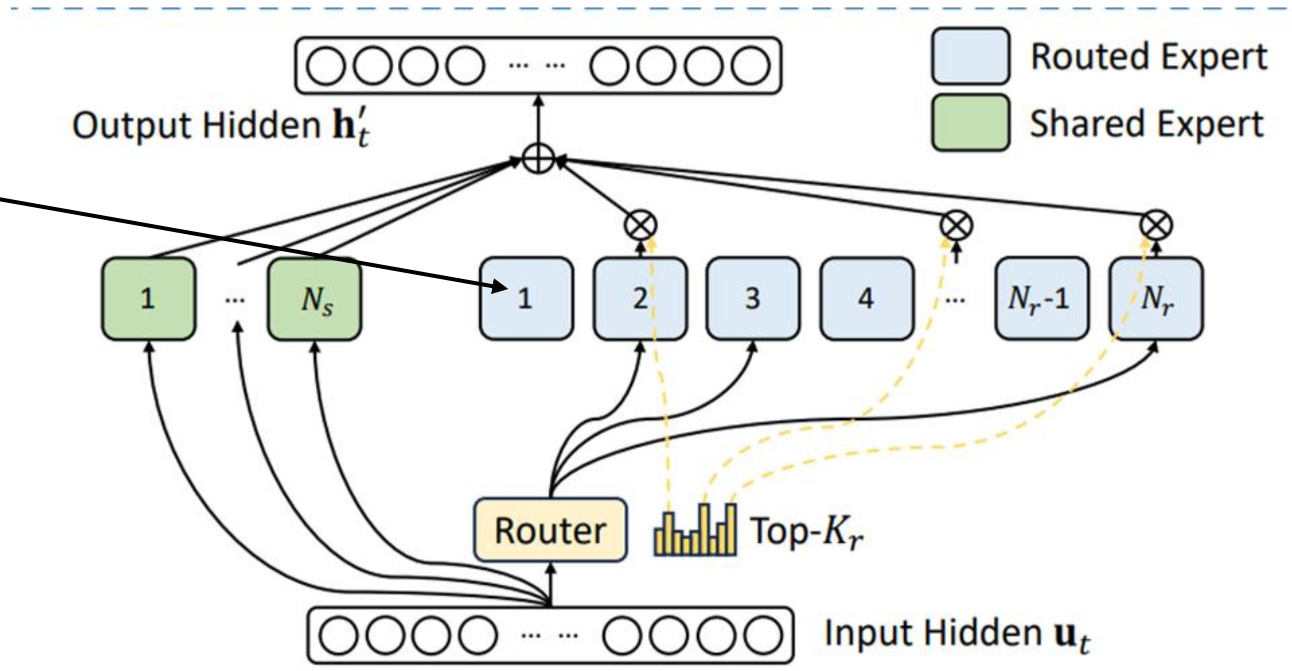
Here use attention output in backward pass
Don’t use FP8 activation, because attention need high precision, so deepseek use E9M6

Other

- Low-Precision Activation(Compared to BF16 activation)
  - ◆ Input of SwiGLU (MoE)

Expert:

```
self.w2(F.silu(self.w1(x)) * self.w3(x))
```



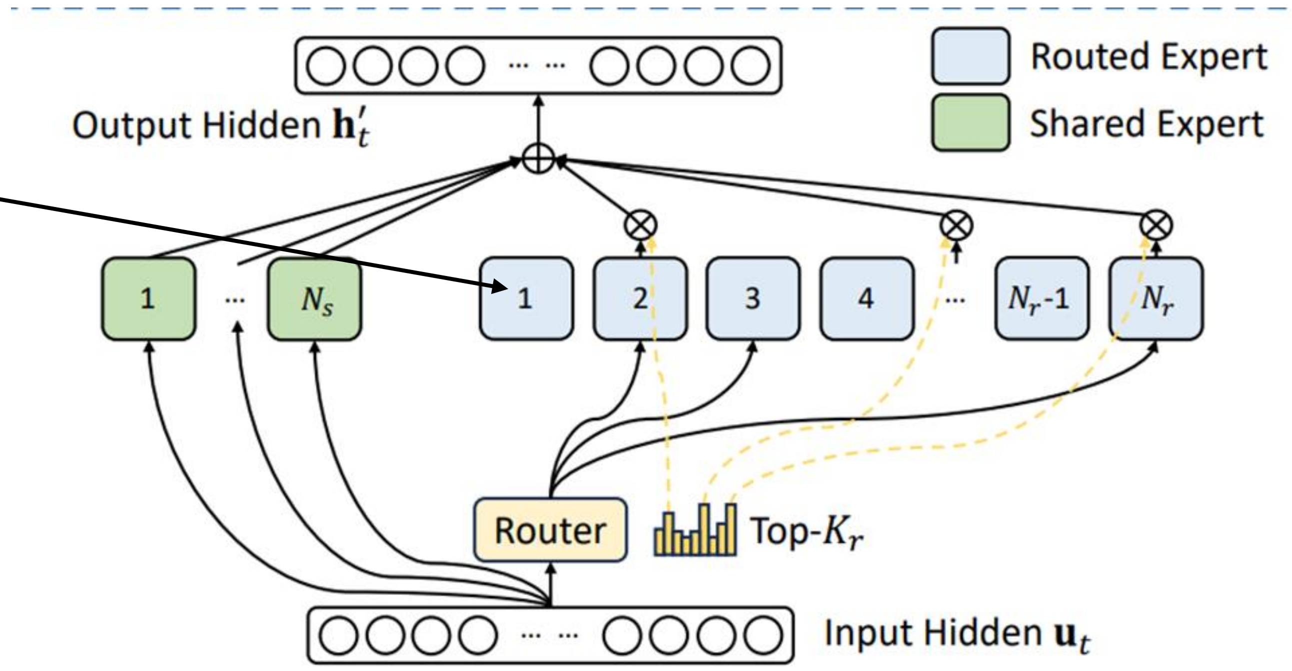
Other

- Low-Precision Activation(Compared to BF16 activation)
  - Input of SwiGLU (MoE)

Expert:

```
self.w2(F.silu(self.w1(x)) * self.w3(x))
```

SwiGLU(x)



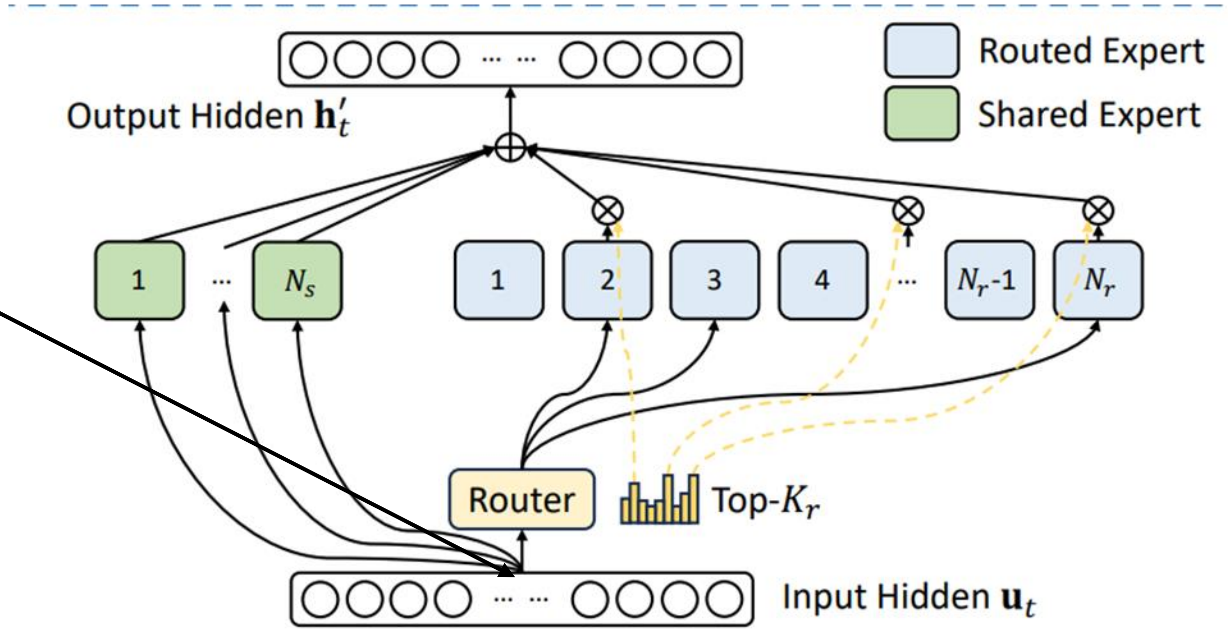
Other

- Low-Precision Activation(Compared to BF16 activation)
  - Input of SwiGLU (MoE)
    - cache the inputs of the SwiGLU operator and recompute its output in the backward (Saving its output incurs significant memory overhead)
    - striking a balance between memory efficiency and computational accuracy

Other

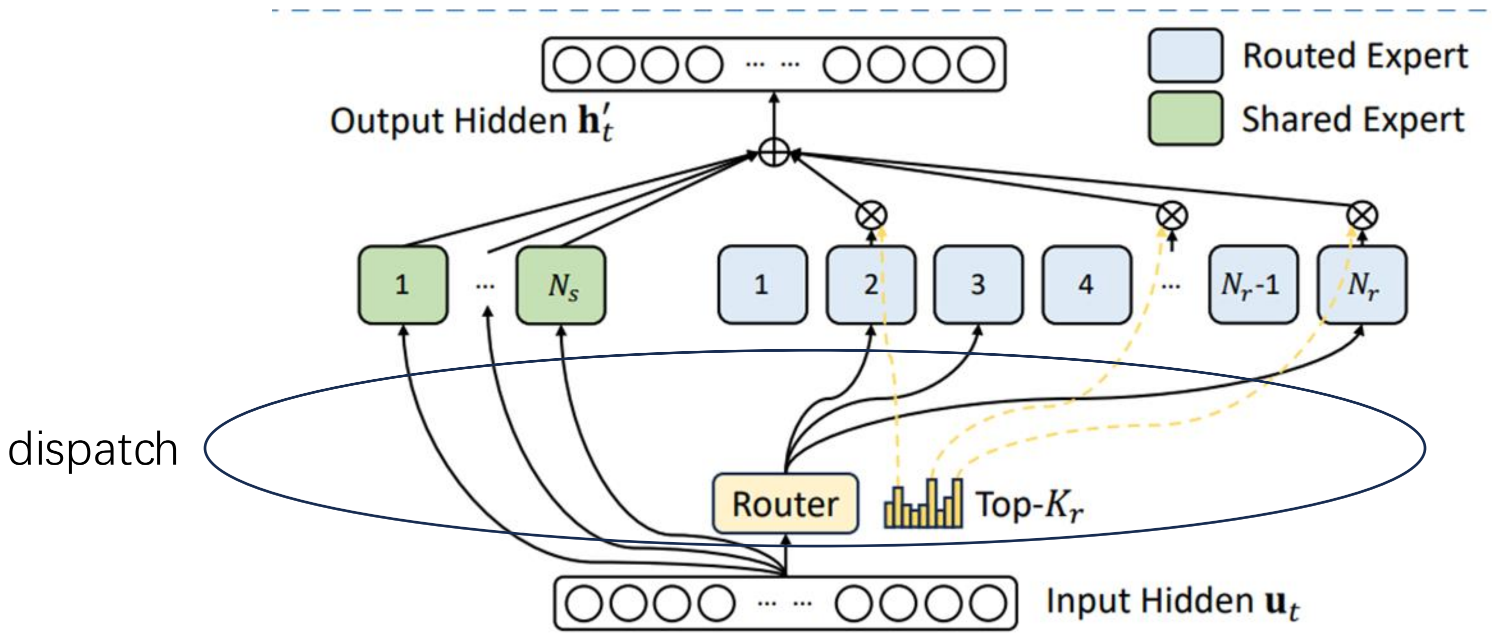
- Low-Precision Communication
  - Scale input of expert to FP8, then dispatch, which decrease communication overhead

Scale BF16 to FP8



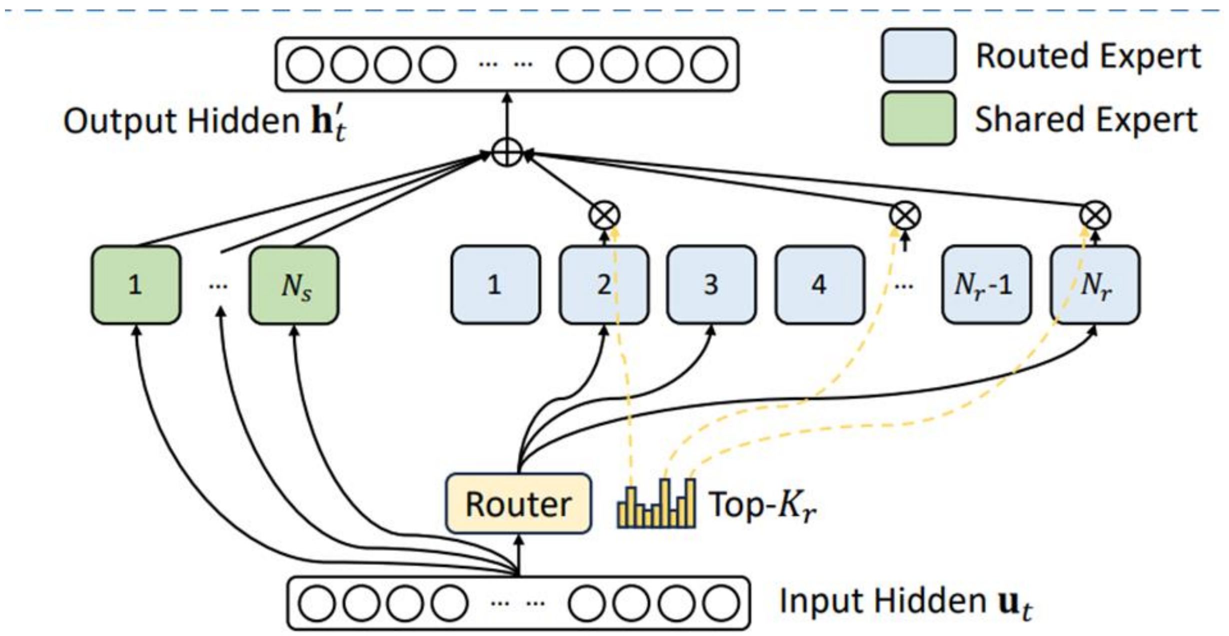
Other

- Low-Precision Communication
  - Scale input of expert to FP8, then dispatch, which decrease communication overhead



Other

- Low-Precision Communication
  - ◆ For combine component, retain them in BF16 to preserve training precision



Other

- Low-Precision Communication
  - For combine component, retain them in BF16 to preserve training precision