DEVIEW
2019

# Faster Transformer: CUDA-Centric BERT Inference Optimization

이민석
NVIDIA

# CONTENTS

DEVIEW 2019

# 1. Background

# 1.1 What is BERT?

DEVIEW 2019

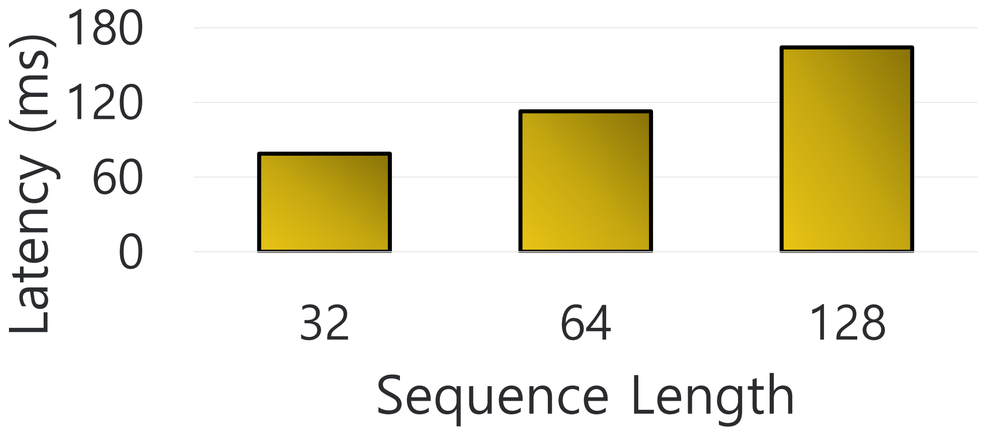
## One of the Most Popular Large-Scale Language Model

- Based on Transformer Encoder
- Provide a leap in accuracy for various NLP tasks beyond conversational AI
- Companies across industries are trying to use the model in production

# 1.2 Challenge in Production

DEVIEW 2019

- Quality of Service: Accuracy + Latency
- BERT requires significant amounts of computation during inference
- Obstacle for companies to deploy BERT in its real-time applications




BERT-base latency on **CPU**

- # Layers: 12
- Batch Size: 1
- # Heads: 12
- Size per Head: 64

# 1.3 Characteristics of Inference



- Compute capability possibly different from Training's
  e.g., training with multiple V100s vs. inference with a single T4
- No backward pass
- Inference-specific optimization is necessary and possible

DEVIEW
2019

# 2.Performance Analysis of BERT Inference on GPU

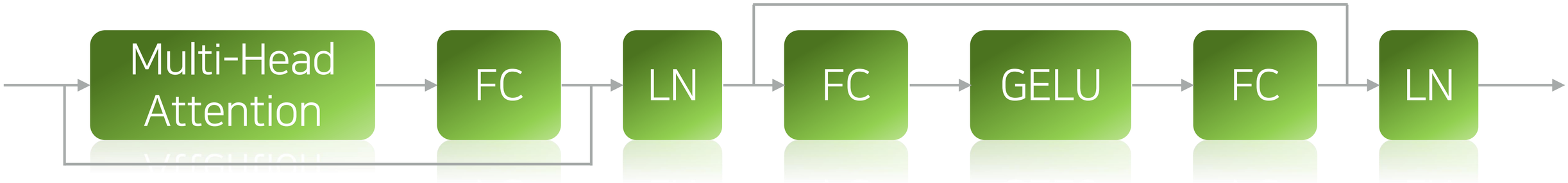
# 2.1 Purpose of Analysis



- To check if there exists opportunities for latency reduction and get some hints for the performance optimization
- To verify if the applied techniques are really effective
- Profiling tools such as Nsight Systems can be useful

# 2.2 BERT Encoder Cell

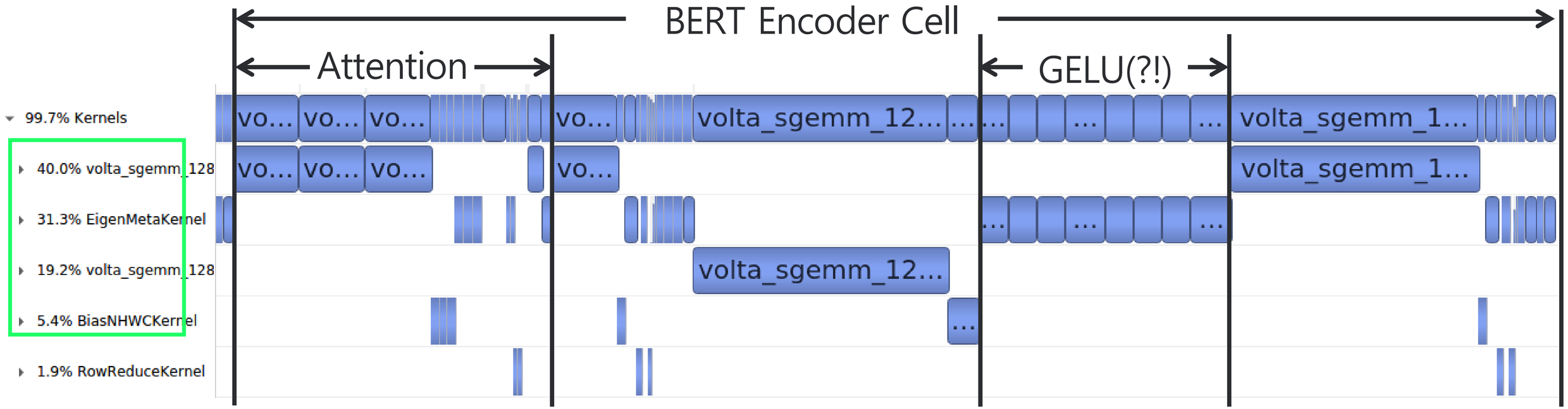




# 2.2.1 Profiling BERT Encoder Cell



- 1 encoder cell leads to > 40 CUDA kernels!



# 2.3. GELU

# 2.3.1 GELU Activation



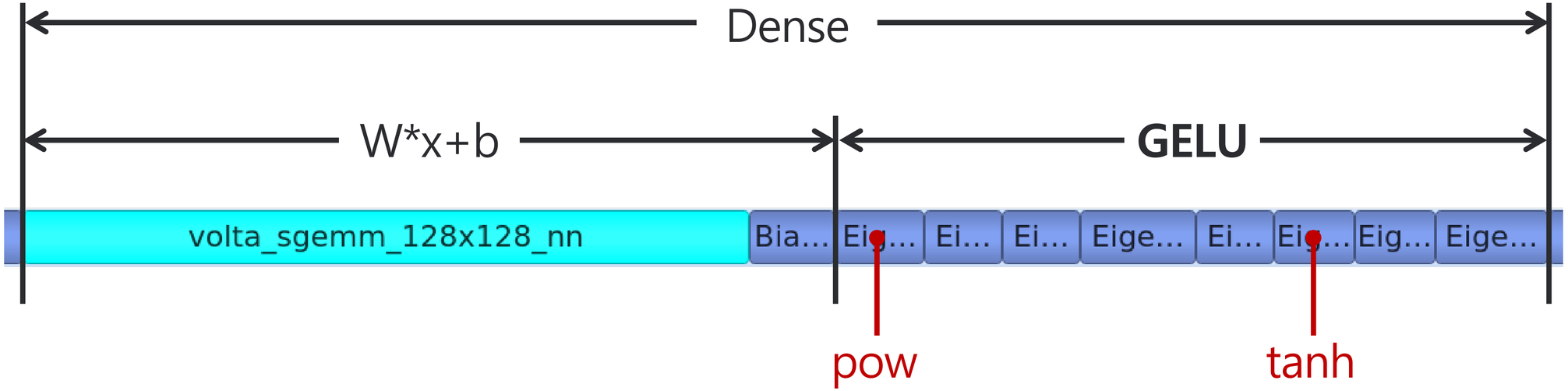
- Easy-to-write element-wise op in Tensorflow by compositing existing ops
- But how about performance?

```
def gelu(x):
  cdf = 0.5 * (1.0 + tf.tanh(
      (np.sqrt(2 / np.pi) * (x + 0.044715 * tf.pow(x, 3)))))
  return x * cdf
```

# 2.3.2 Profiling GELU on GPU

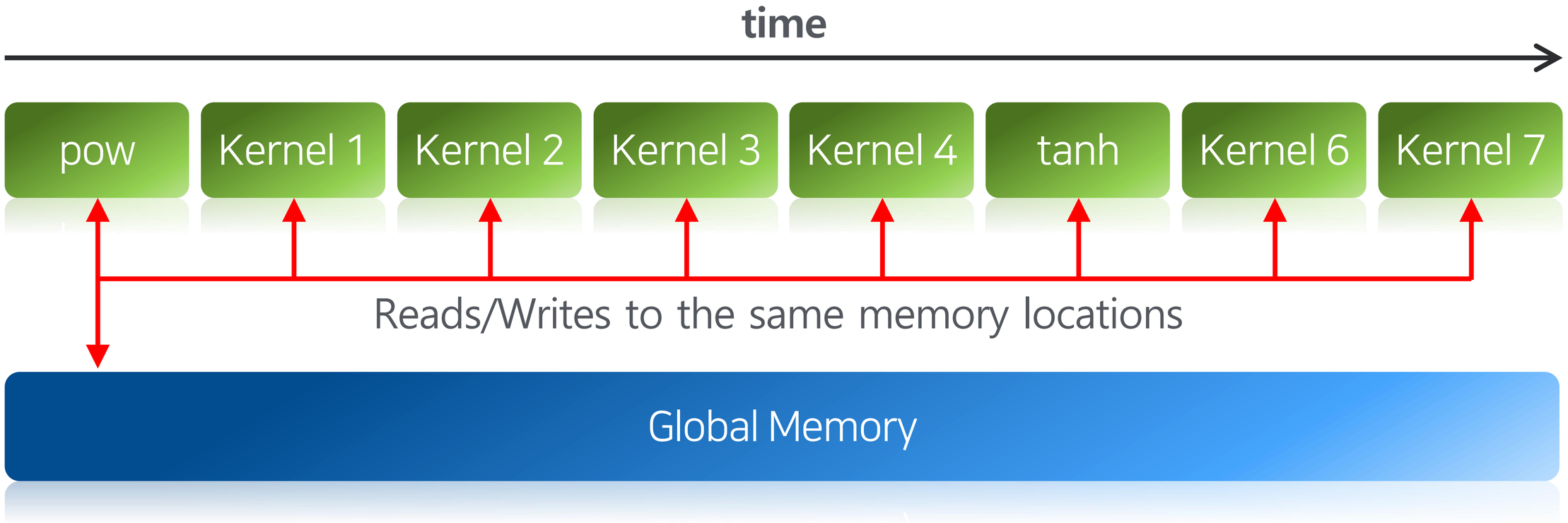


- GELU consists of 8 CUDA kernels
- Aggregated runtime is almost equivalent to W*x+b



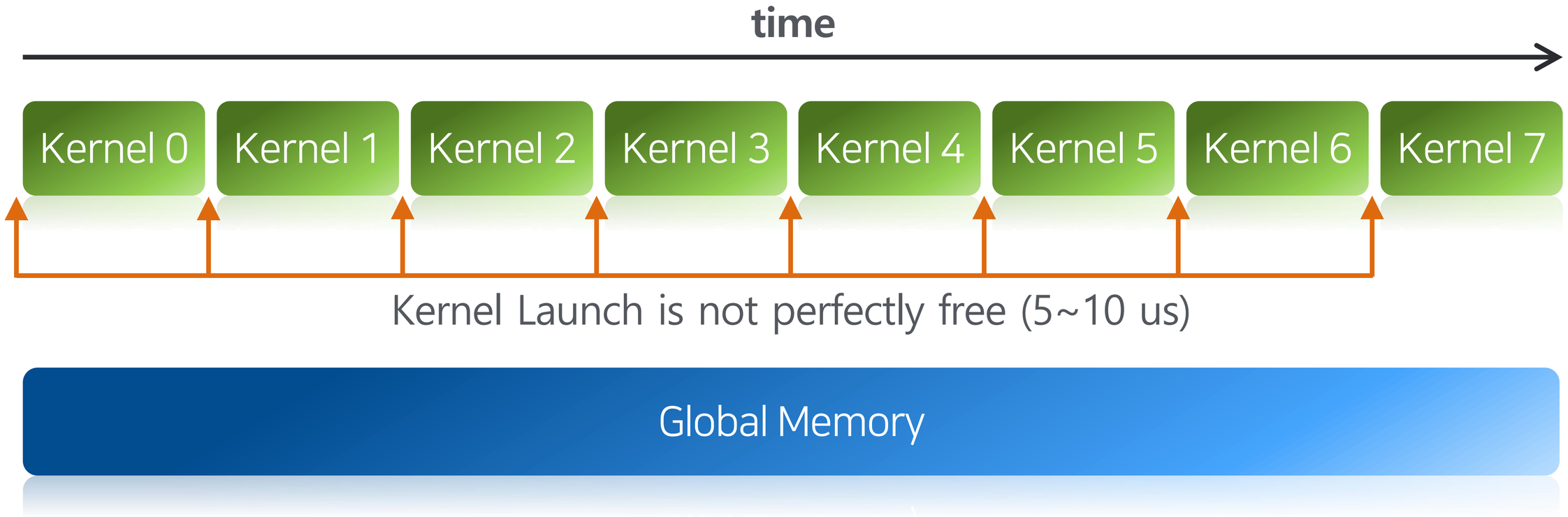
# 2.3.3 Memory Access of naïve GELU





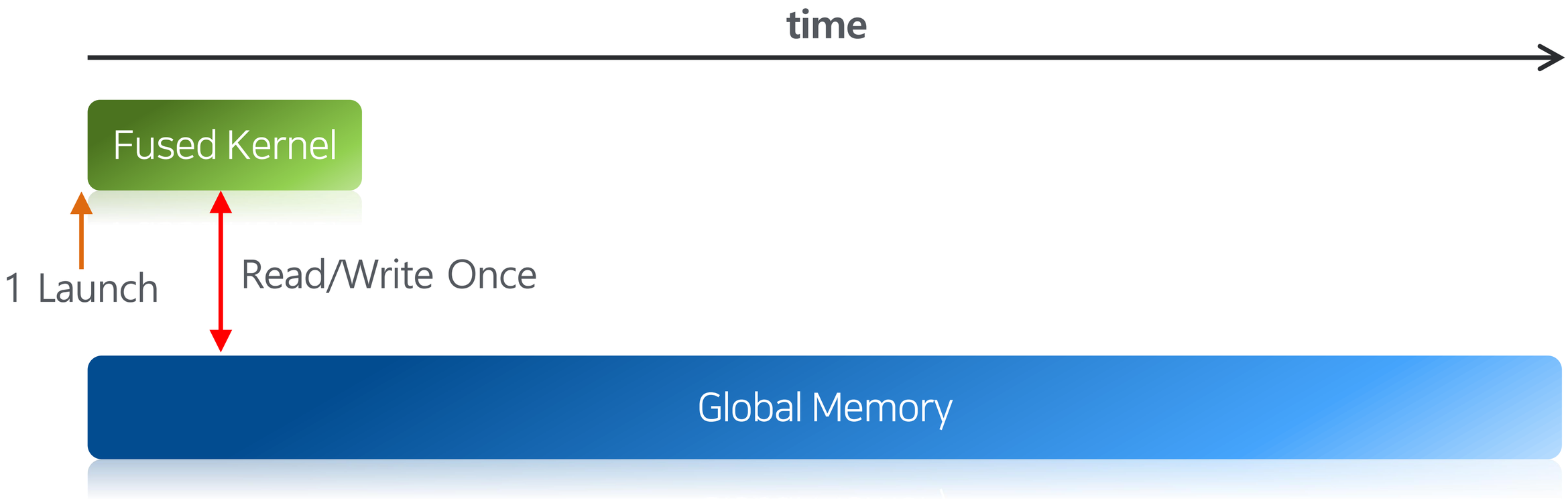
# 2.3.4 Kernel Launch Overhead





# 2.3.5 Fused GELU

DEVIEW 2019



# 2.3.6 Fused GELU CUDA C++ Function

DEVIEW 2019

- All the operations are done in on-chip registers

```
template <typename T>
__inline__ __device__ why not __global__? (explained later)
T gelu(T x)
{
  float cdf = 0.5f * (1.0f + tanhf((0.7978845608028654f * (x + 0.044715f * x * x * x))));
  return x * cdf;
}
```

Faster than pow(x, 3)

2 / np.pi

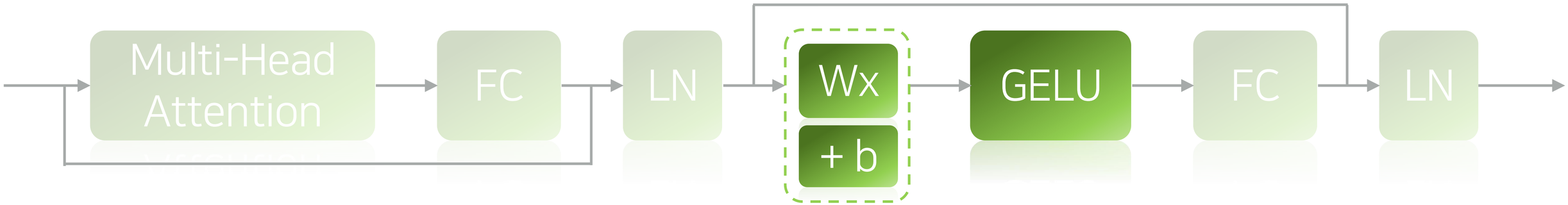
# 2.4. addBias + GELU

# 2.4.1 BERT Encoder Cell revisited



Fully-Connect Layer

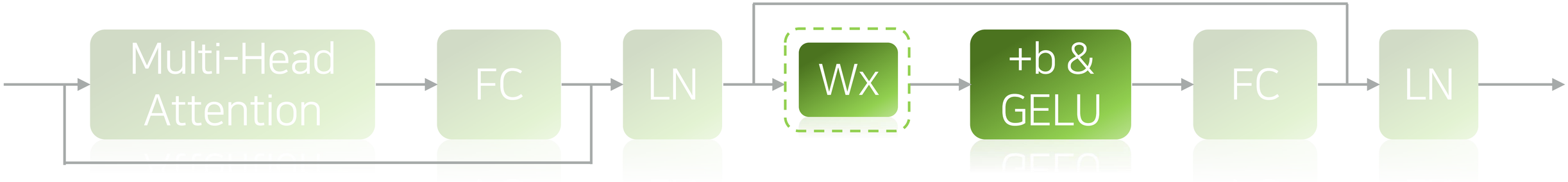
- Wx: highly optimized CUBLAS GEMM
- +b: simple addBias CUDA kernel



# 2.4.2 Fusion of addBias and GELU



- Simply call GELU device function inside your addBias kernel!
- Wx still relies upon CUBLAS GEMM

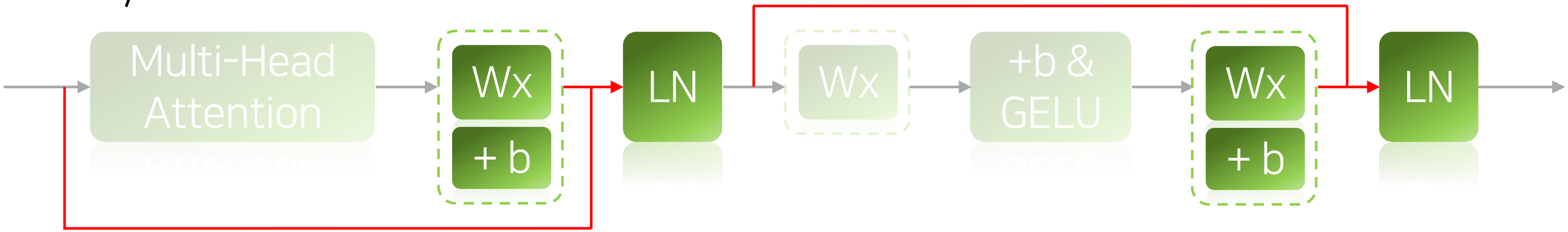


# 2.5. addBias + LayerNorm

# 2.5.1 BERT Encoder Cell revisited



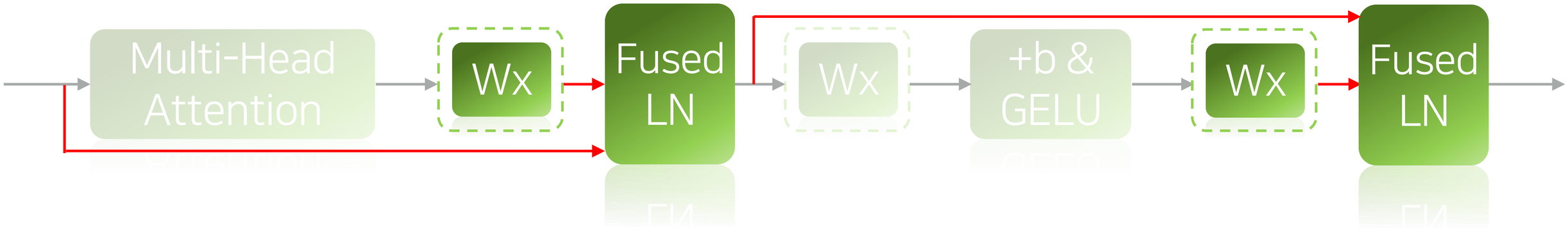
- Residual connection
- addBias
- Layer Normalization



# 2.5.2 Fused Layer Normalization

DEVIEW 2019

- Residual connection + addBias + LayerNorm



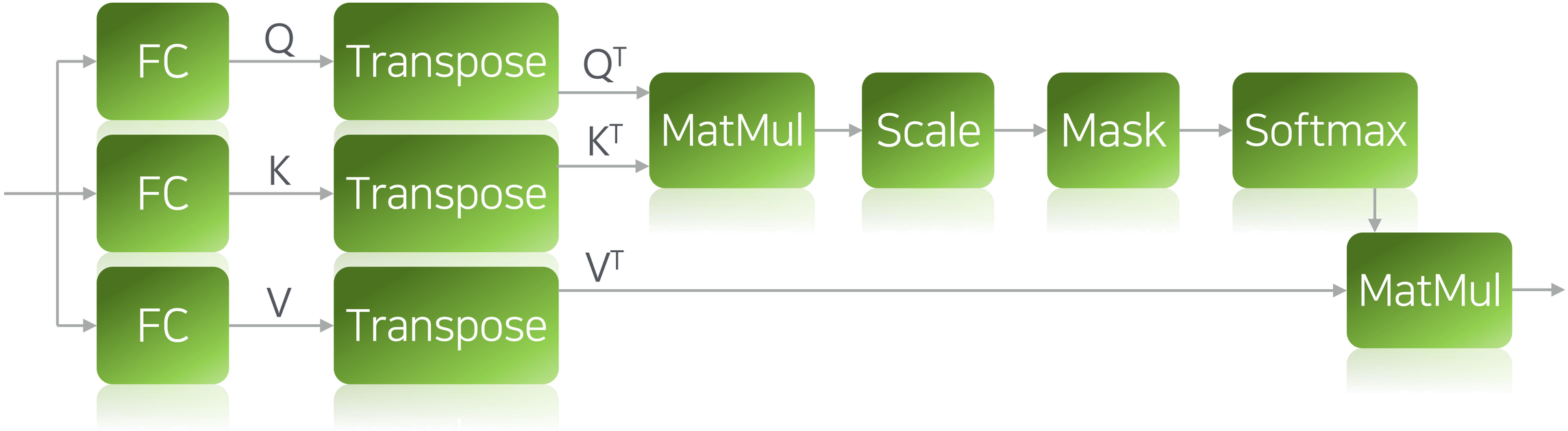
DEVIEW
2019

# 2.6. Multi-Head Attention

# 2.6.1 Multi-Head Attention



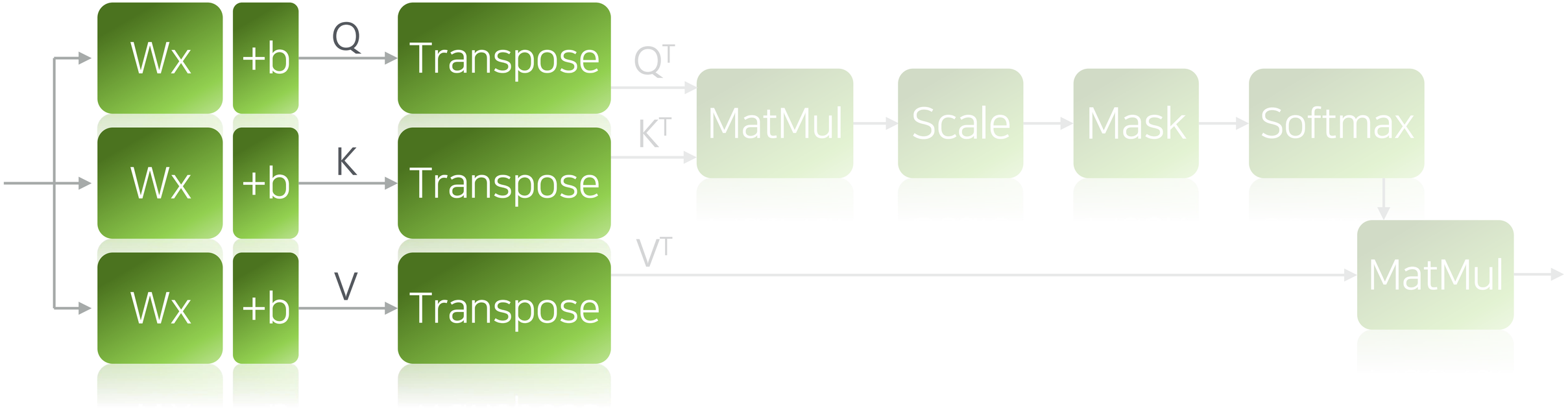
- Input: (BxSxNxH)



# 2.6.1 Multi-Head Attention

DEVIEW 2019

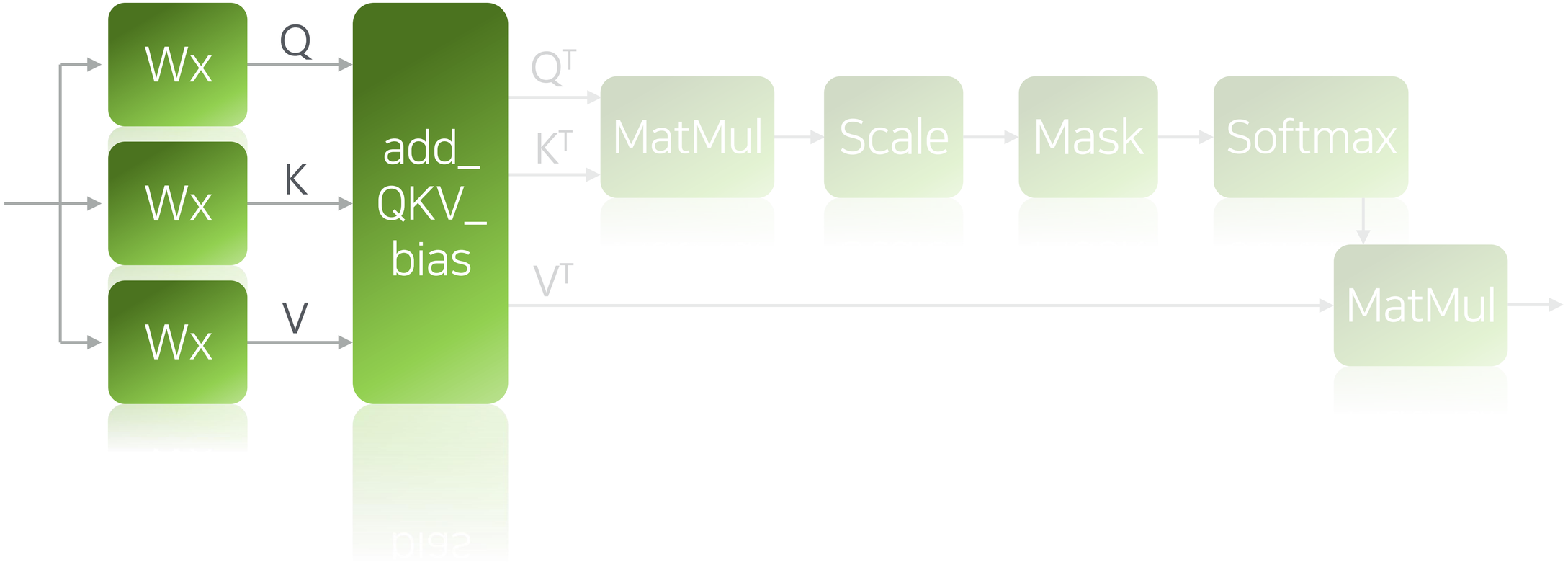
- Input: (BxSxNxH)



# 2.6.2 Fusion of addBias and Transpose



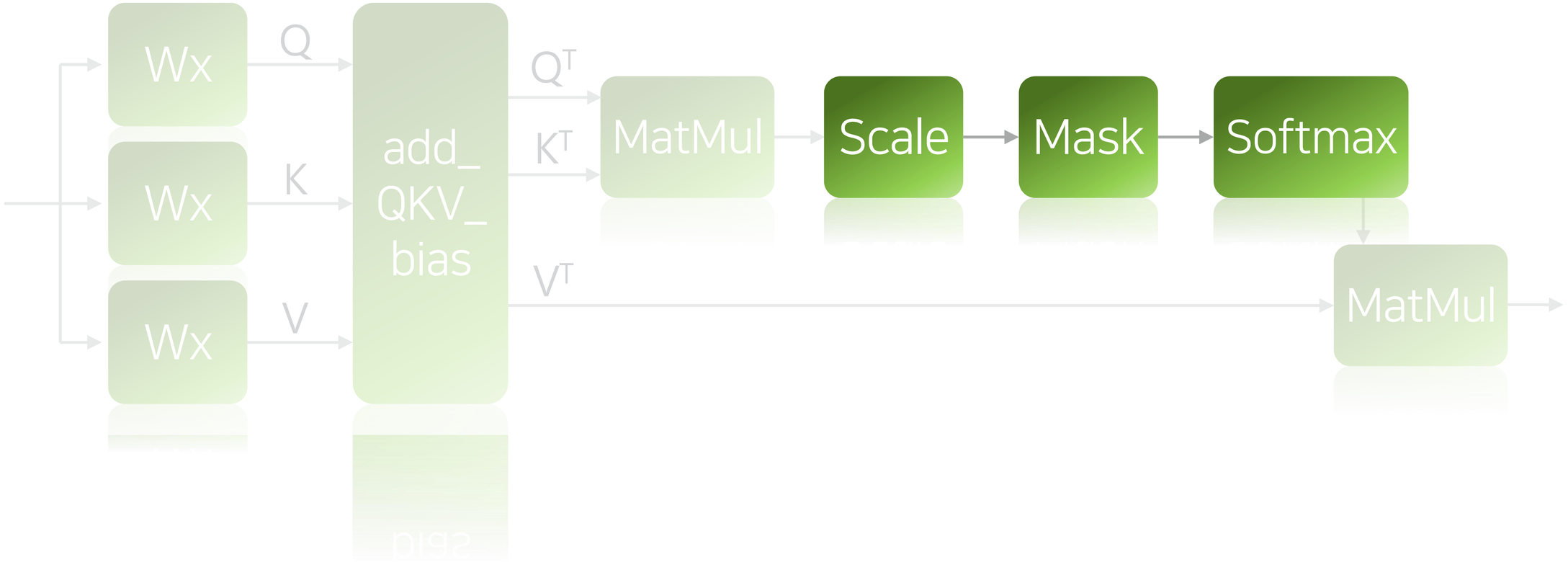
- Improve thread-level parallelism, launch overhead and memory efficiency



# 2.6.3 Scale, Mask and Softmax

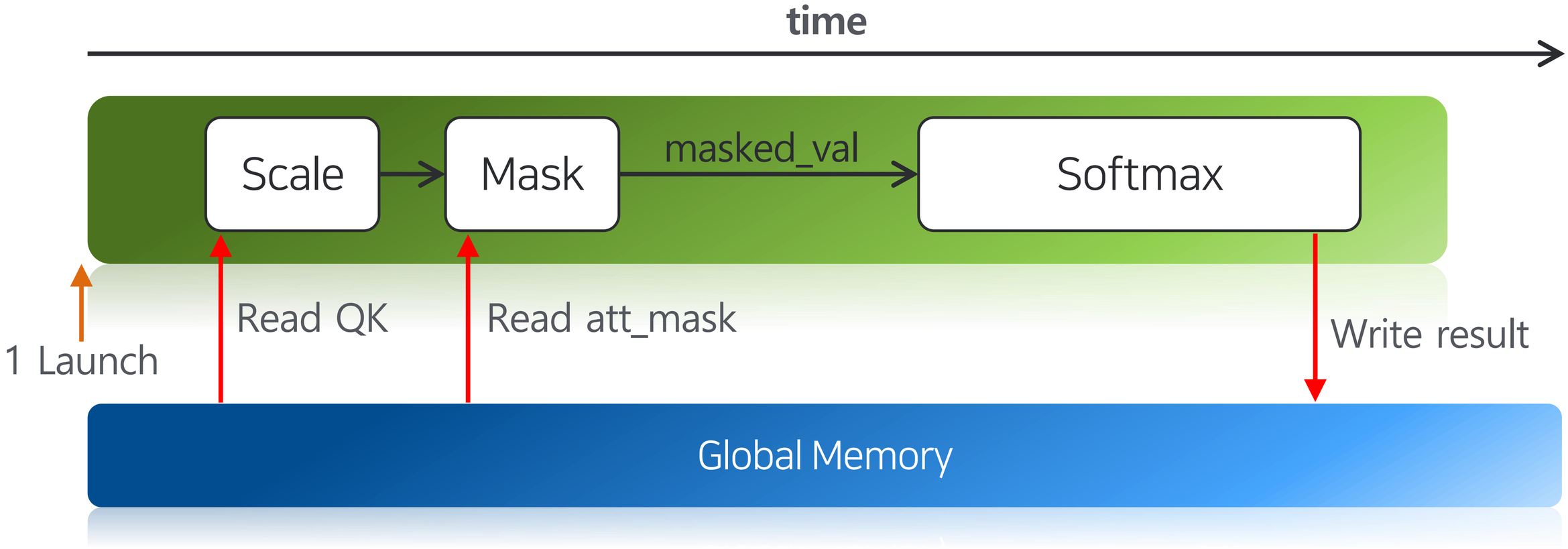


- Scale and Mask are element-wise operations



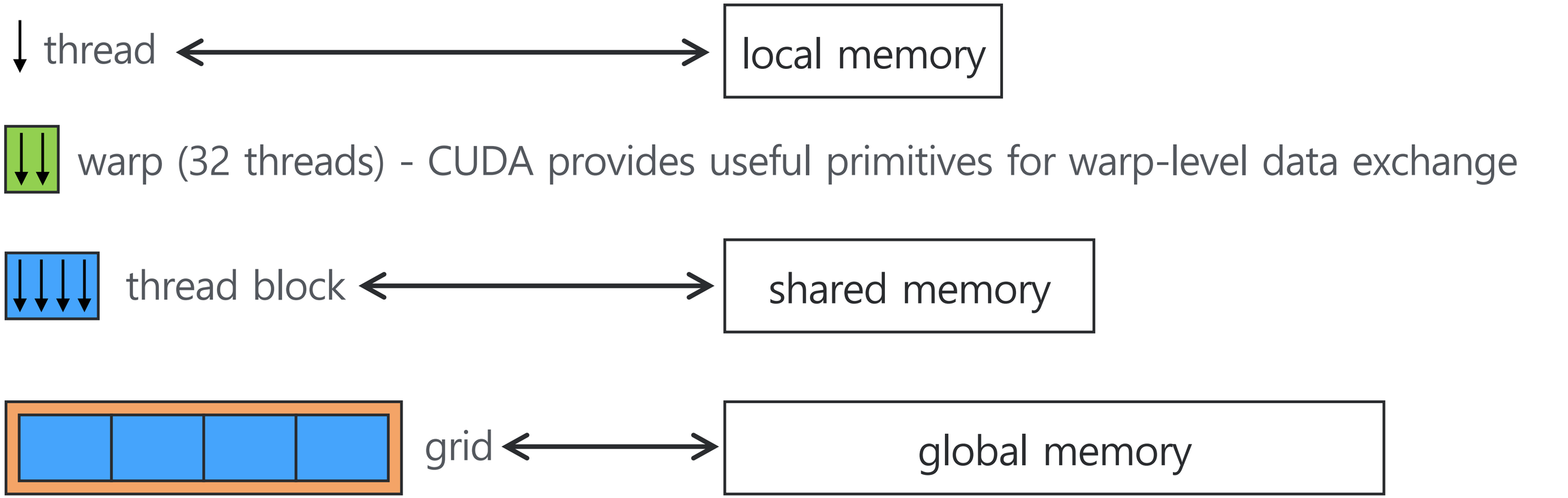
# 2.6.5 Fused Softmax





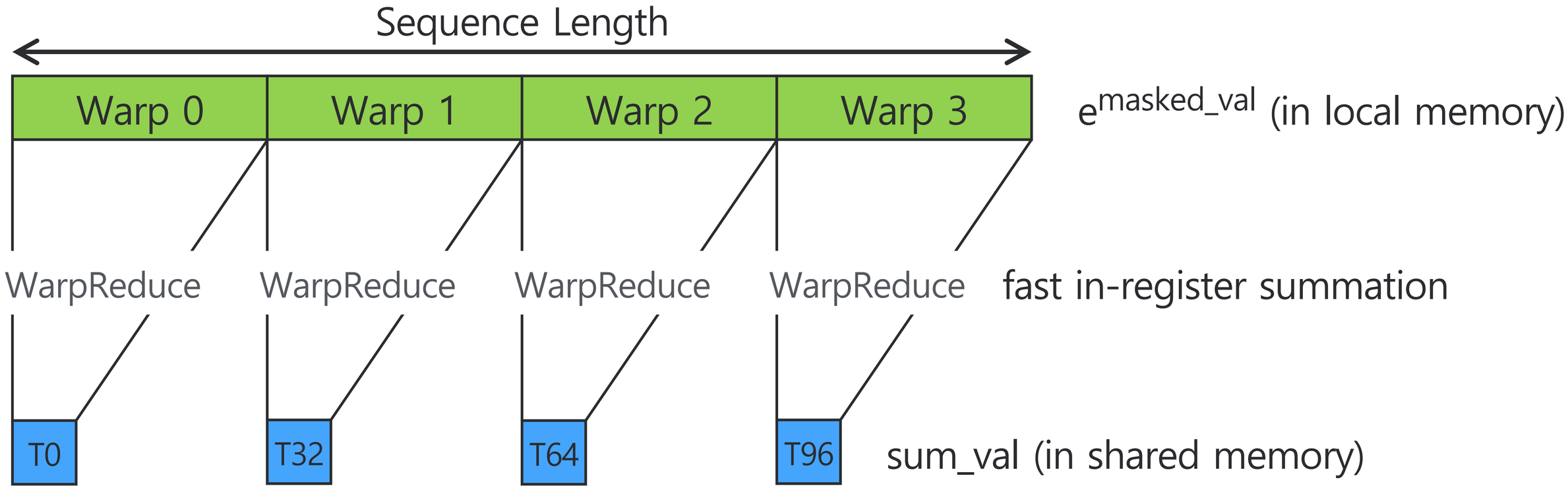
# 2.6.6 CUDA Thread/Memory Hierarchy





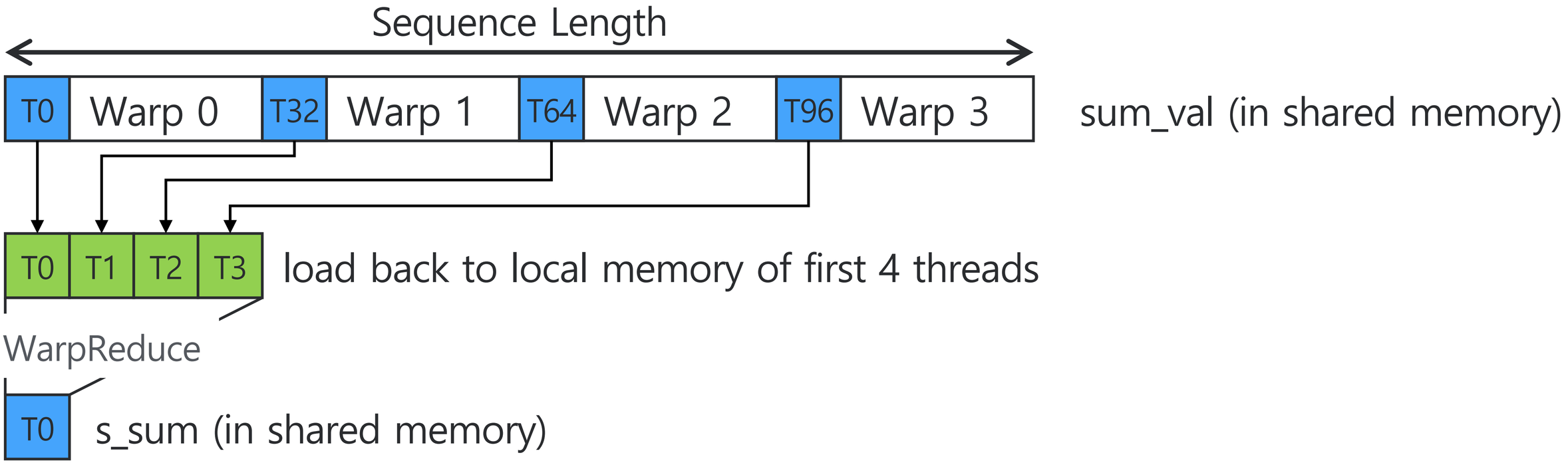
# 2.6.7 Softmax Implementation Sketch

DEVIEW 2019



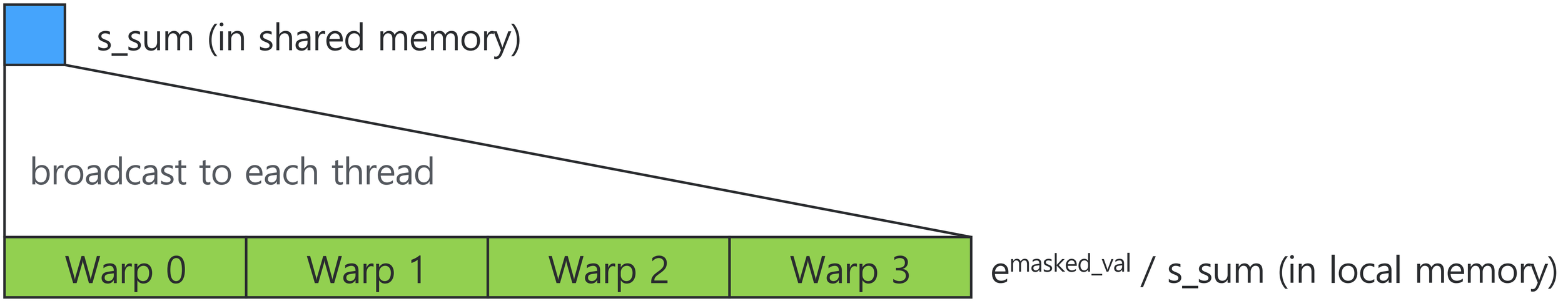
# 2.6.7 Softmax Implementation Sketch





# 2.6.7 Softmax Implementation Sketch

DEVIEW 2019



# 2.6.8 Task-Specific Optimization



Input Tensor Shape: [batch_size, head_num, seq_len, seq_len]

Tasks with large batch sizes: already have enough thread blocks

- # blocks: (batch_size x head_num), block size: (seq_len x seq_len)

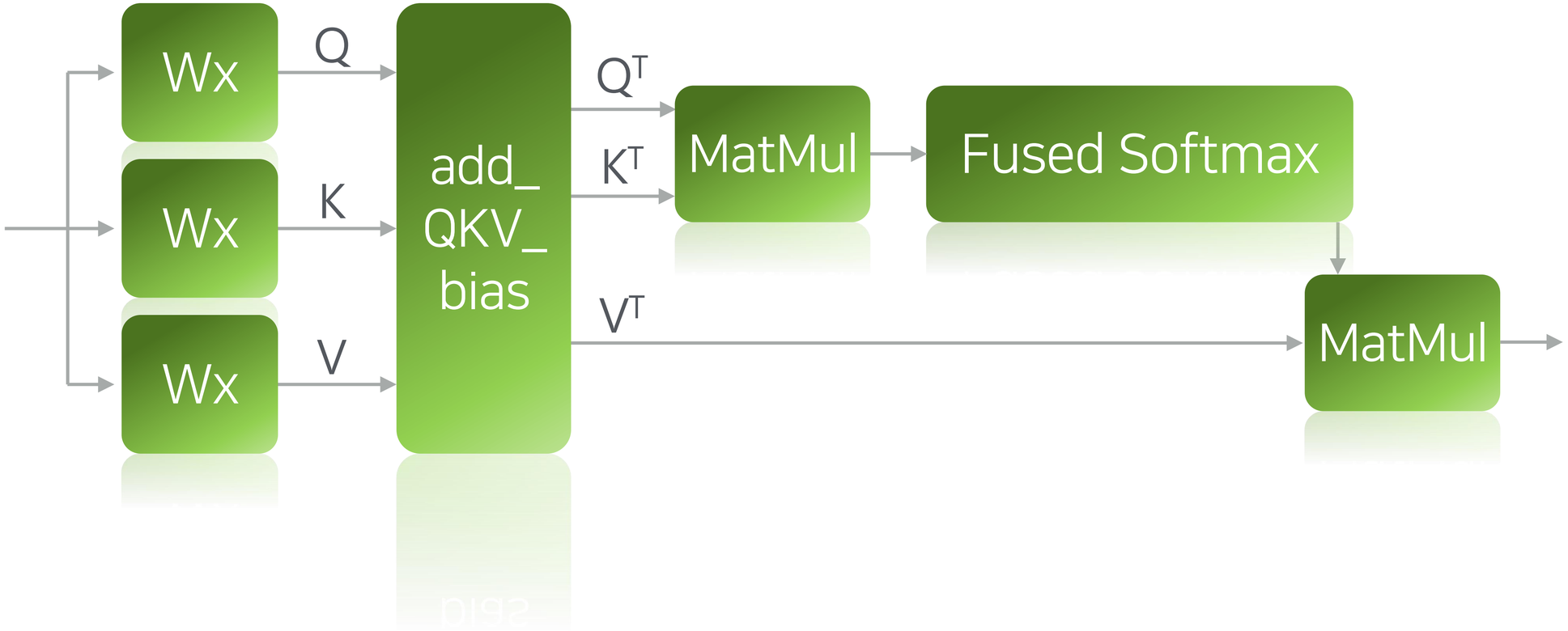
Tasks with small batch sizes: need to improve # thread blocks

- # blocks: (batch_size x head_num x seq_len), block size: (seq_len)

# 2.7 Fused Multi-Head Attention



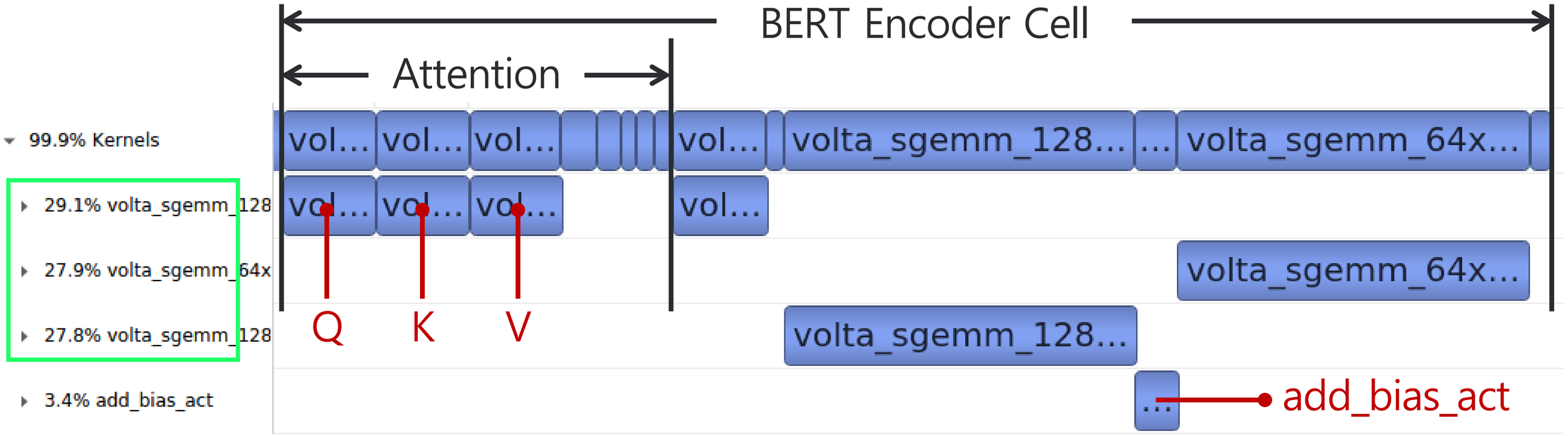
- Wx and MatMul are the calls to highly optimized CUBLAS GEMMs



# 2.7.1 Profiling FasterTransformer



- cuBLAS GEMMs represent >= 80% of execution time
- What can we do to improve the performance further?



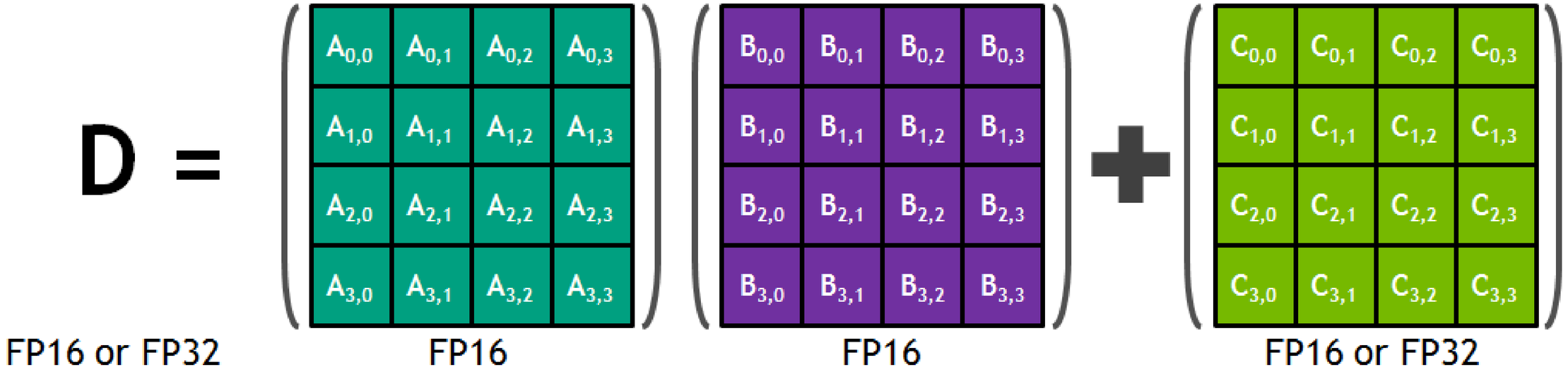
DEVIEW
2019

# 2.8. Lower Precision

# 2.8.2 Tensor Cores in NVIDIA GPUs

DEVIEW 2019

- Matrix-multiply-and-accumulate units available in Volta/Turing Archs
- To improve throughput by using lower precisions, e.g., FP16



# 2.8.3 Enable FP16 in cuBLAS GEMMs



```
cublasStatus_t cublasGemmEx(cublasHandle_t handle,
                            cublasOperation_t transa,
                            cublasOperation_t transb,
                            int m,
                            int n,
                            int k,
                            const void    *alpha,
                            const void     *A,
                            cudaDataType_t Atype,
                            int lda,
                            const void     *B,
                            cudaDataType_t Btype,
                            int ldb,
                            const void    *beta,
                            void           *C,
                            cudaDataType_t Ctype,
                            int ldc,
                            cudaDataType_t computeType,
                            cublasGemmAlgo_t algo)
```

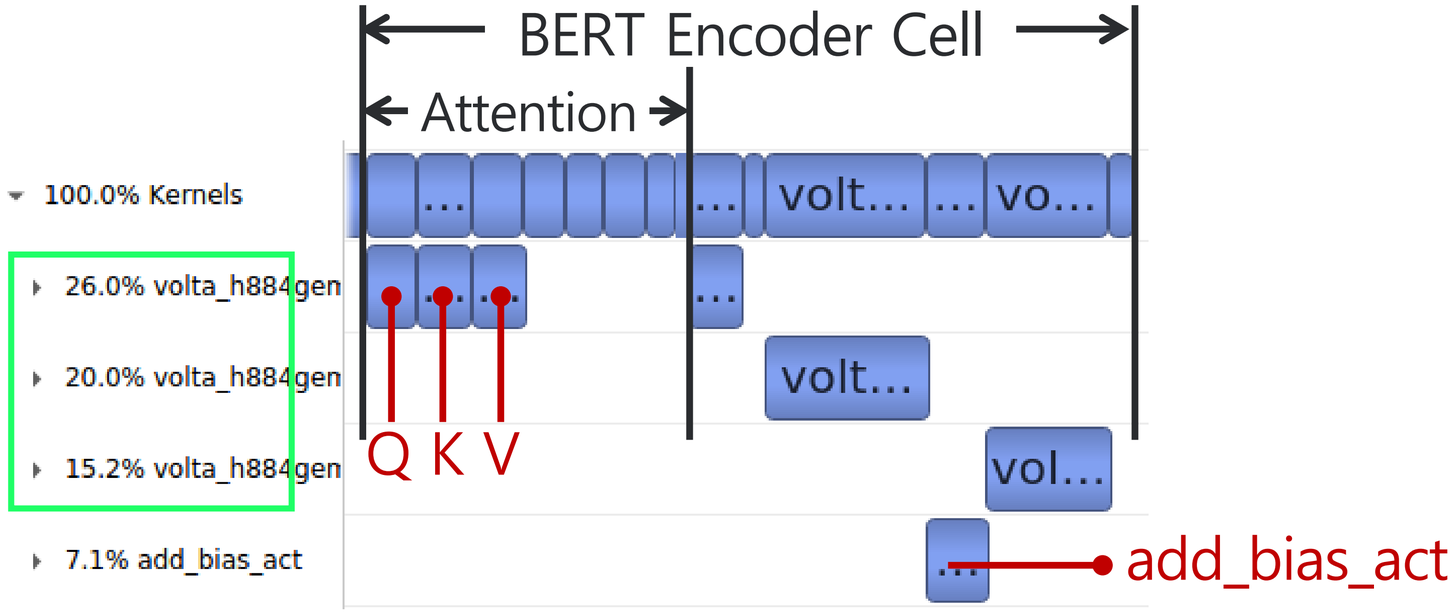
In/Out Matrices

| Atype/Btype | Ctype |
|---|---|
| CUDA_R_32F | CUDA_R_32F |
| CUDA_R_16F | CUDA_R_16F |
| CUDA_R_16F | CUDA_R_32F |

# 2.8.4 Profiling Again

DEVIEW 2019

- Reduced the latency of GEMMs by halving the precision (FP32->FP16)



# 3. Evaluation

# 3.1. Methodology

DEVIEW 2019

- Baseline: Tensorflow w/ XLA
- Model: BERT Base
- GPU: V100, P4, and T4
- CPU: Intel Xeon Gold 6132 CPU @ 2.60GHz

# 3.2. Performance Comparison on V100

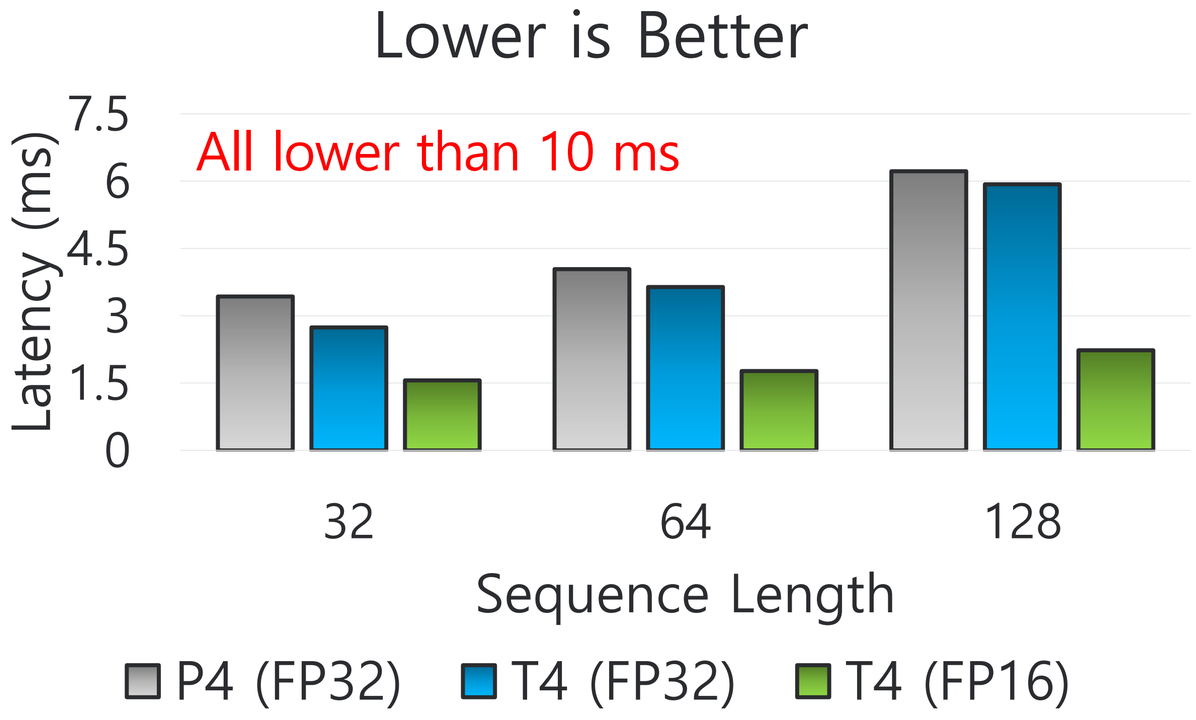



- # Layers: 12
- Sequence Length: 32
- # Heads: 12
- Size per Head: 64

- Memory Clock: 877MHz
- Processor Clock: 1380MHz

# 3.3. Faster Transformer on T4






- # Layers: 12
- Batch Size: 1
- # Heads: 12
- Size per Head: 64

- Memory Clock: 5000MHz
- Processor Clock: 1590MHz

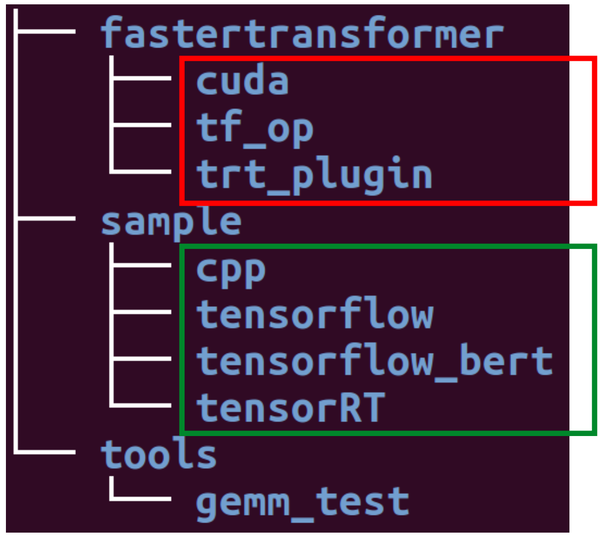
DEVIEW
2019

# 4. Faster Transformer Repository

# 4.1. Code Structure



- All that I explained has been open-sourced as Faster Transformer!
  https://github.com/NVIDIA/DeepLearningExamples/tree/master/FasterTransformer




CUDA C++ Implementation,

Tensorflow and TensorRT interfaces

Sample illustrating how to use it

in C++, Tensorflow, and TensorRT

# 4.2. Use Case: PingAn's PA-Occam-Bert

DEVIEW 2019

| Rank | 1-example Latency (milliseconds) | Model | Framework |
|---|---|---|---|
| 1 Jul 2019 | 7.5790 | PA-Occam-Bert *Ping An Technology Occam Platform* source | Tensorflow 1.13.0 |
| 2 Feb 2019 | 7.9000 | FastFusionNet *Wu et al. (Cornell, SayMosaic, Google)* source | Pytorch v0.3.1 |
| 3 Oct 2017 | 100.0000 | BiDAF *Stanford DAWN* source | TensorFlow v1.2 |

- DAWNBench SQuAD
- F1 score 75.80
- Faster Transformer Integration on 1X Tesla V100
- https://github.com/geekerzli/PA-Occam-Bert

- https://dawn.cs.stanford.edu/benchmark/#squad-inference-time

# 4.3. Upcoming Faster Transformer 2.0

DEVIEW 2019

- Transformer Decoder will be included!

- Your feedback is highly appreciated
  https://github.com/NVIDIA/DeepLearningExamples/issues

DEVIEW
2019

# Q & A

DEVIEW
2019

# Thank You