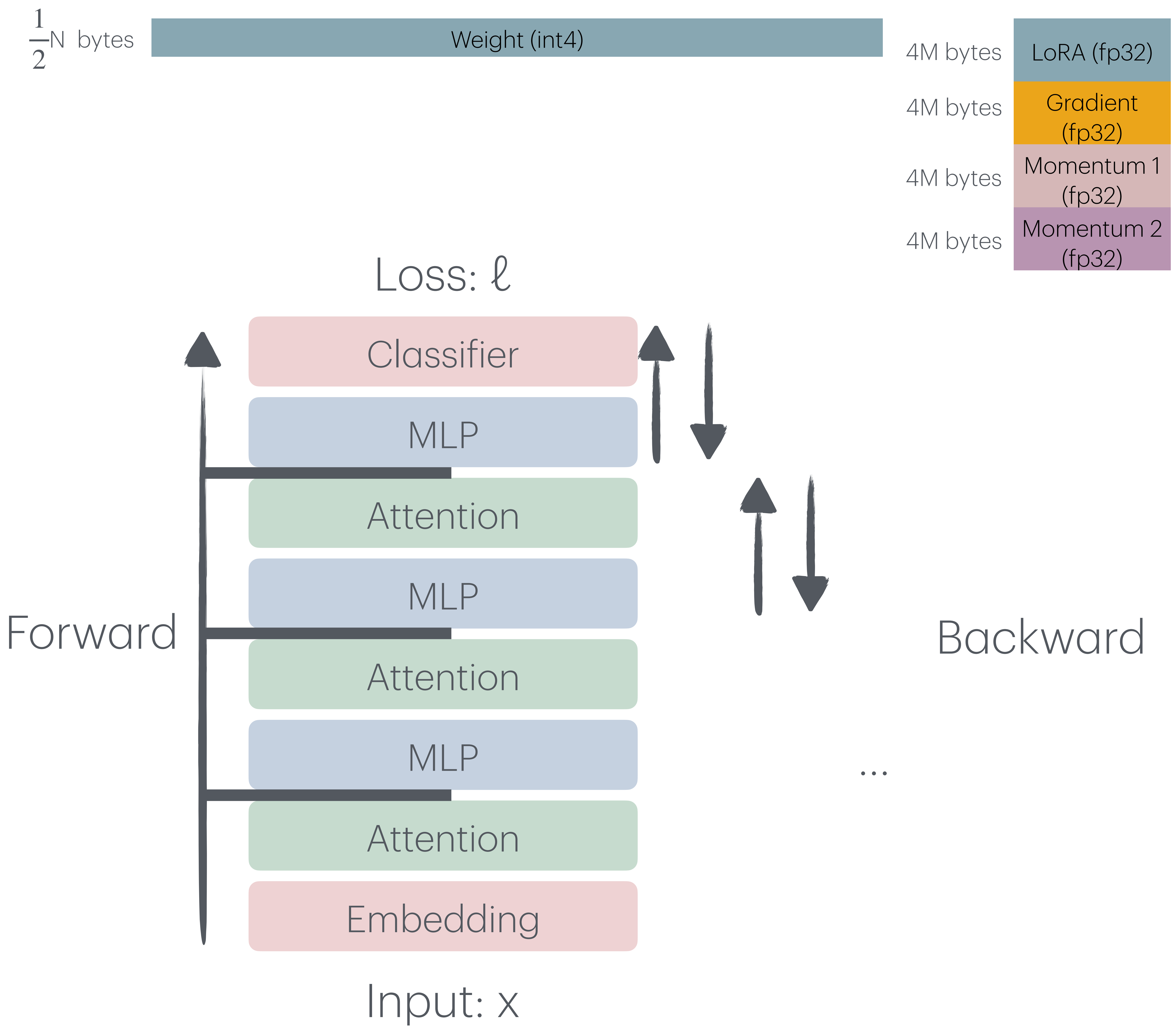
Flash Attention

Philipp Krähenbühl, UT Austin

Memory efficient model training

Memory requirements

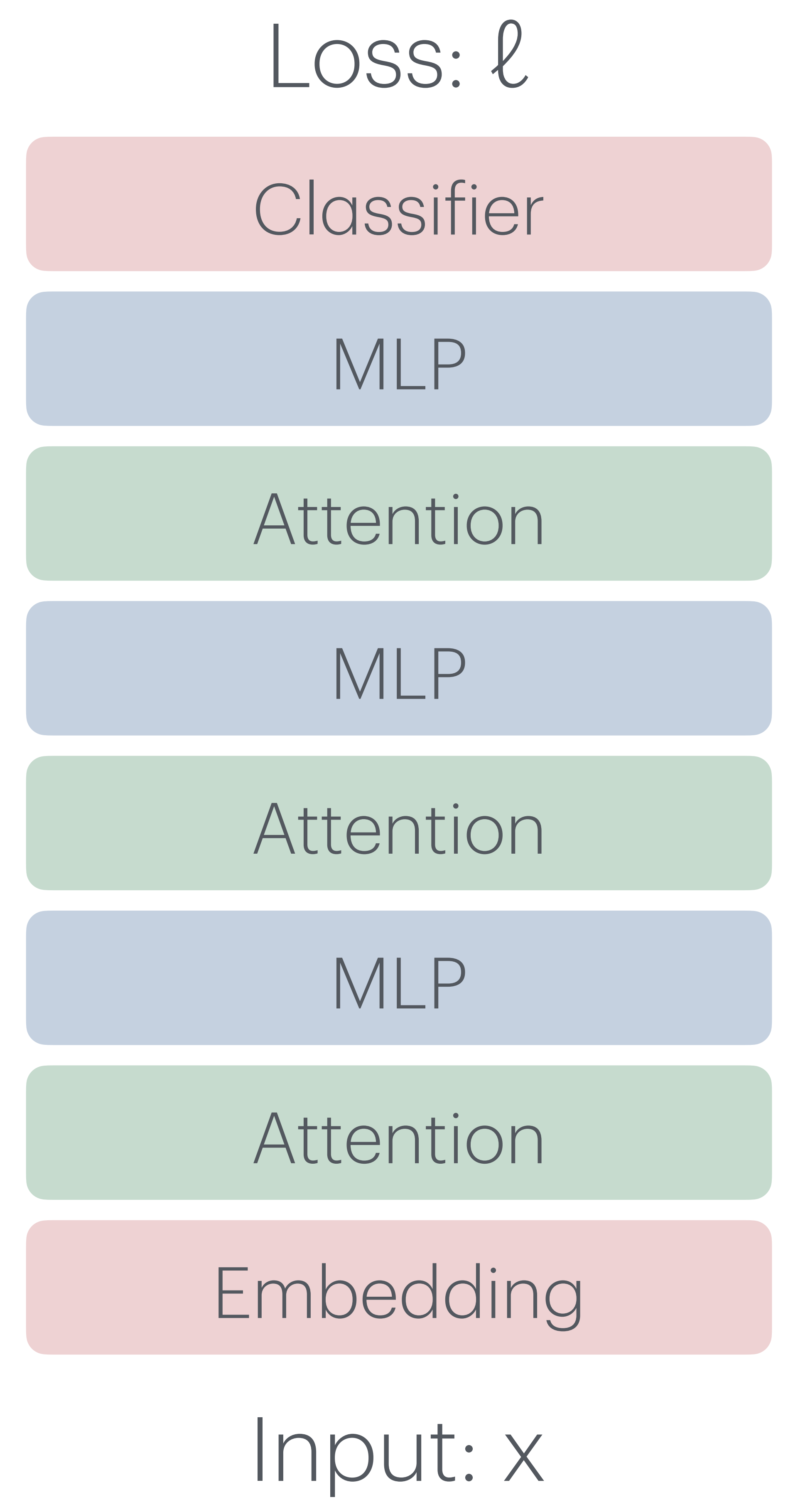
- QLoRA
  - Model parameters: N, LoRA param M
  - Weights: N int4, M floats
  - Gradients: M floats
  - Momentum: M floats
  - 2nd momentum (ADAM): M floats
- $\frac{1}{2}$N+16M bytes; M often ~1-5% of N
- $O(\sqrt{D})$ gradient checkpoint; two forward passes



Almost the full picture

Memory use

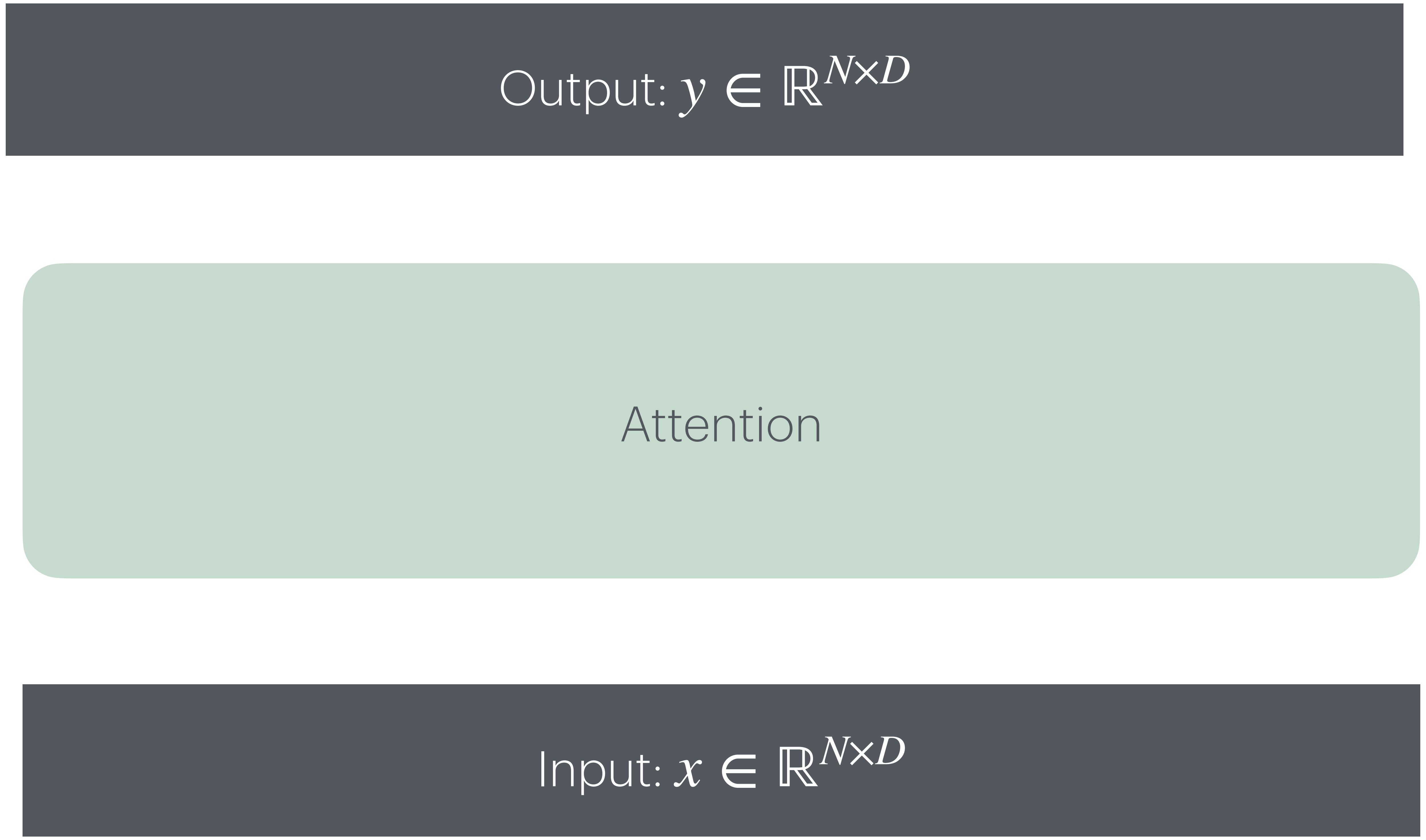
- Weights, Gradients, Momentum
  - Quantization, LoRA, Galore
- Activations
  - Checkpointing
- Intermediate/Scratch memory
  - Specialized implementations



Attention

Intermediate Memory

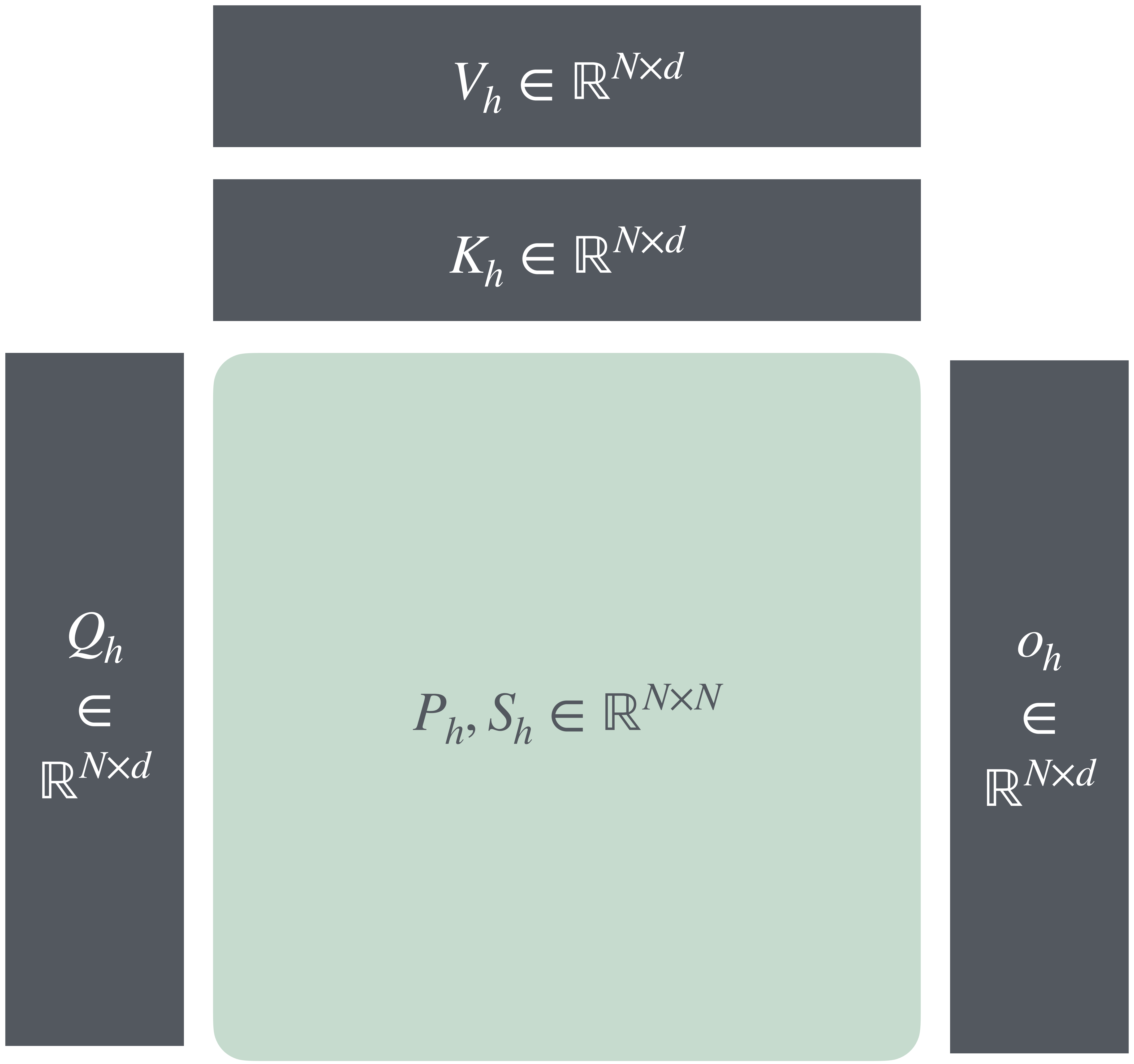
- Attention (per head)
  - $K_h = W_K x \quad V_h = W_V x \quad Q_h = W_Q x$; $K_h, V_h, Q_h \in \mathbb{R}^{N\times d}$
  - $S_h = Q_h K_h^\top$; $S_h \in \mathbb{R}^{N\times N}$
  - $P_h = \text{softmax}(S_h)$; $P_h \in \mathbb{R}^{N\times N}$
  - $o_h = P_h V_h$; $o_h \in \mathbb{R}^{N\times d}$
- $y = \sum_h W_h o_h$



Attention

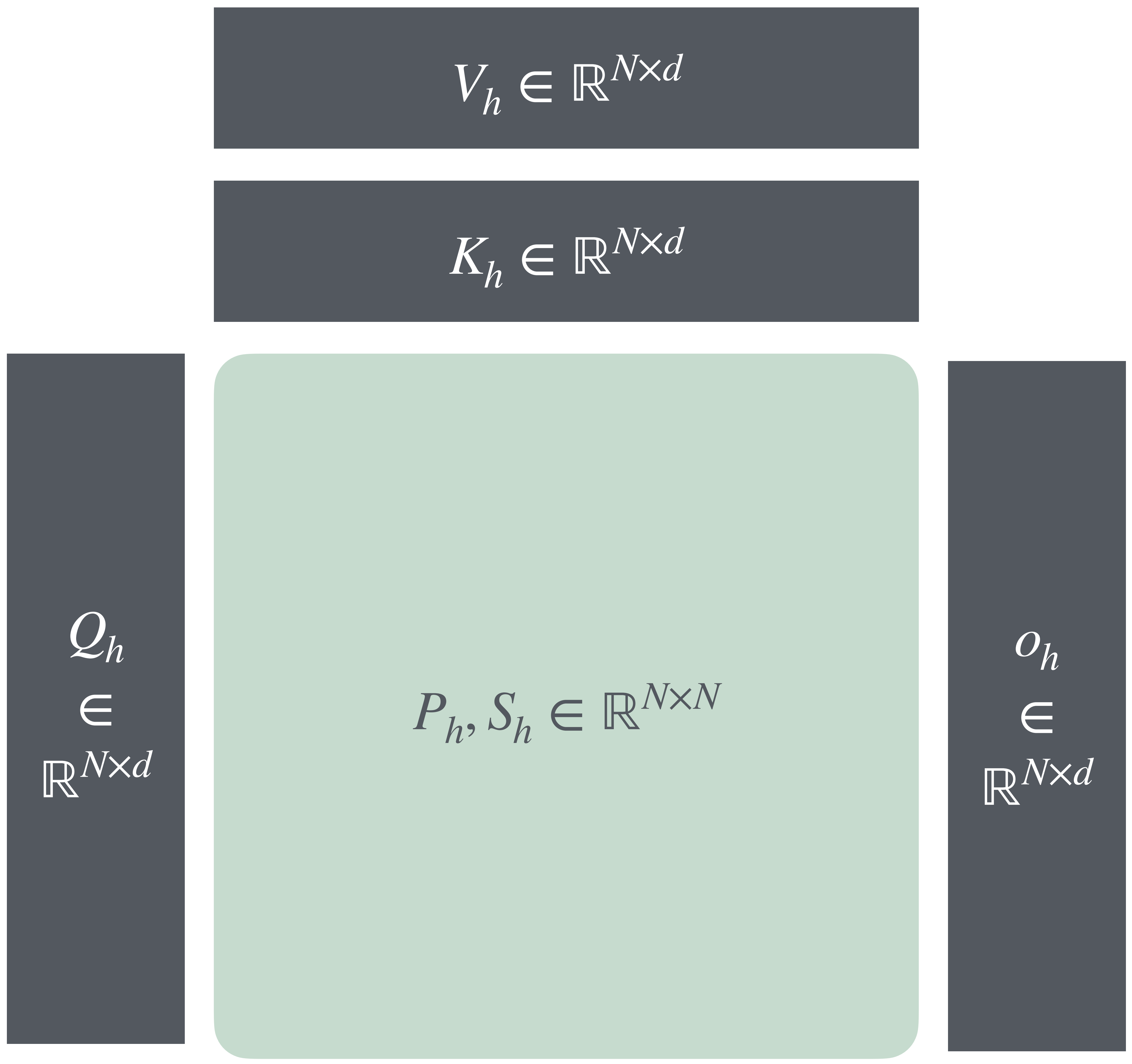
Intermediate Memory

- Attention (per head)
  - $K_h = W_K x \quad V_h = W_V x \quad Q_h = W_Q x;$
    $K_h, V_h, Q_h \in \mathbb{R}^{N \times d}$
  - $S_h = Q_h K_h^\top;\ S_h \in \mathbb{R}^{N \times N}$
  - $P_h = \text{softmax}(S_h);\ P_h \in \mathbb{R}^{N \times N}$
  - $o_h = P_h V_h;\ o_h \in \mathbb{R}^{N \times d}$
- $y = \sum_h W_h o_h$



Flash Attention

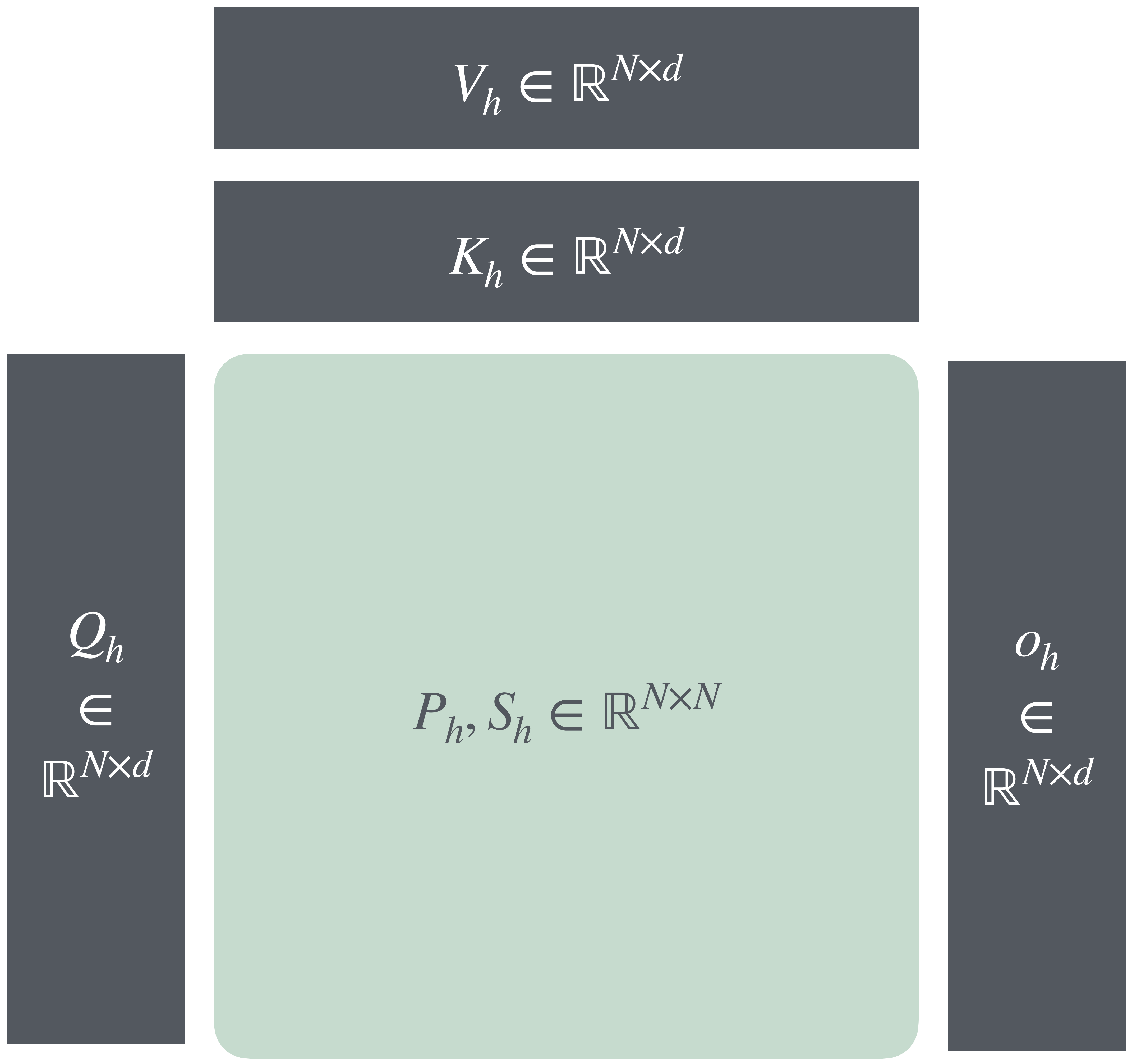
- Never compute $S_h, P_h \in \mathbb{R}^{N \times N}$ explicitly
- Easy on CPU
- Tricky on GPU



Flash Attention

On CPU (for illustration only)

```
Inputs: V, K, Q

Output: o

for i = 1…N:

 S_i = K @ Q_i

 P_i = exp(S_i) / sum(exp(S_i))

 o_i = V.mT @ P_i
```



Flash Attention

On CPU (for illustration only)

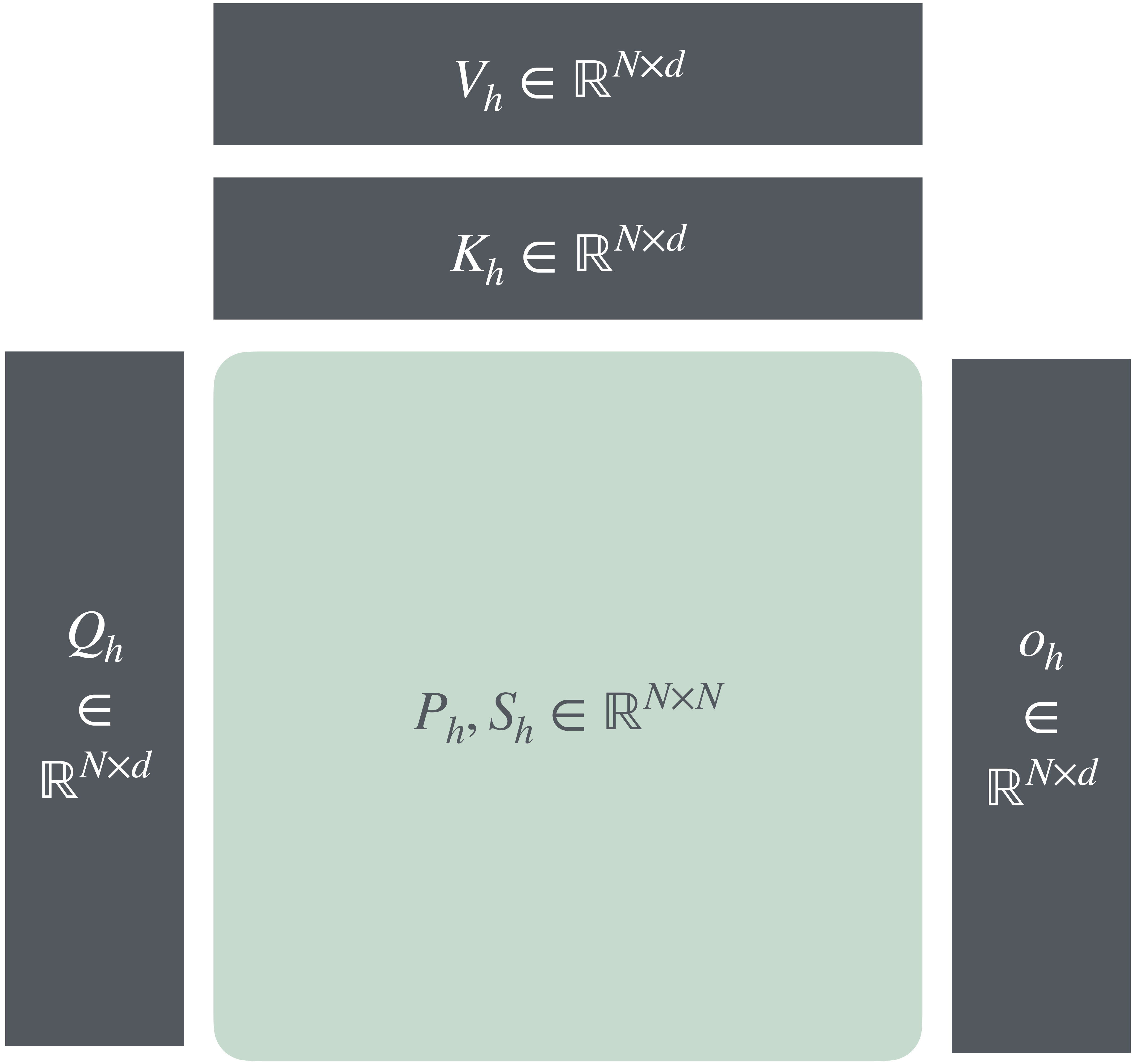
```
Inputs: V, K, Q

Output: o

for i = 1…N:

 S_i = K @ Q_i

 P_i = exp(S_i) / sum(exp(S_i))

 o_i = V.mT @ P_i
```

Memory (local)	**Memory (global)**
	O(ND)
	O(ND)
O(N)	
O(N)	
O(N)	

Flash Attention

On CPU (for illustration only)

```
Inputs: V, K, Q

Output: o

for i = 1…N:

 o_i = 0

 n = 0

 for j = 1…N:

   S_ij = K_j @ Q_i

   o_i += exp(S_ij) V_j

   n += exp(S_ij)

 o_i /= n
```



Flash Attention

On CPU (for illustration only)

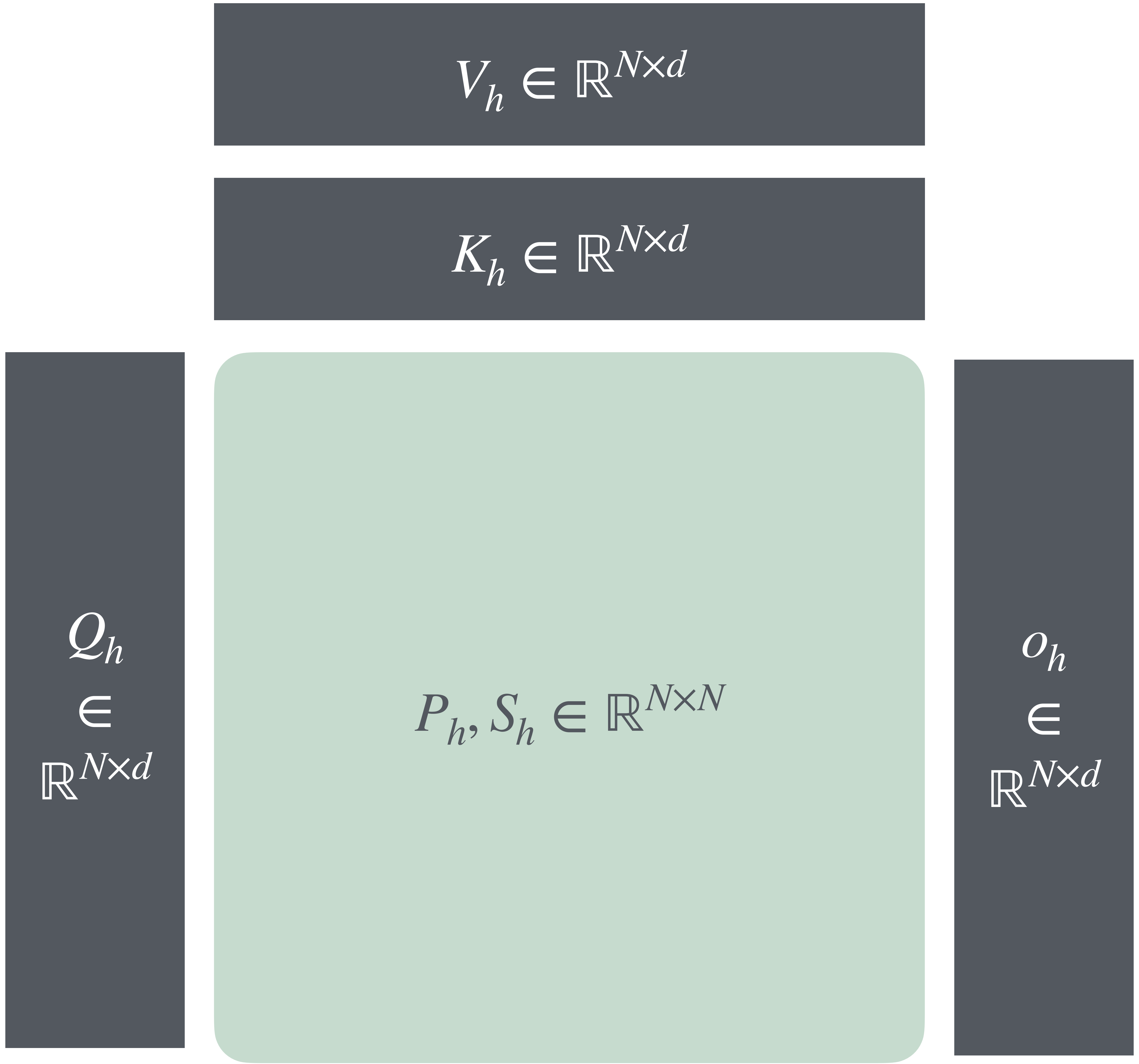
```
Inputs: V, K, Q

Output: o

for i = 1…N:

 o_i = 0

 n = 0

 for j = 1…N:

   S_ij = K_j @ Q_i

   o_i += exp(S_ij) V_j

   n += exp(S_ij)

 o_i /= n
```

Memory (local)	Memory (global)
	O(ND)
	O(ND)
O(D)	
O(1)	
O(1)	
O(D)	
O(1)	

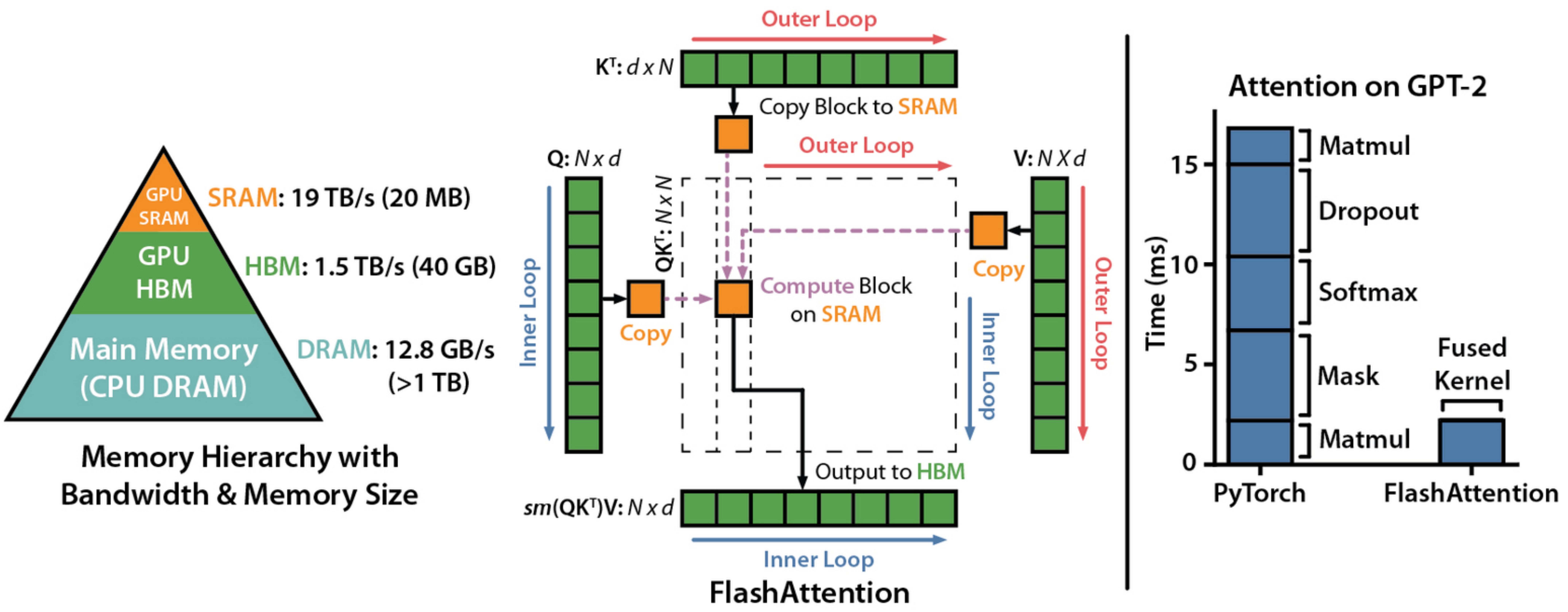
Attention

Backward

- Regular attention stores (for backward)
  - $S_h = Q_h K_h^\top;\ S_h \in \mathbb{R}^{N \times N}$
  - $P_h = \mathrm{softmax}(S_h);\ P_h \in \mathbb{R}^{N \times N}$
- FlashAttention
  - Recomputes S_h, P_h
  - Saves memory



Flash Attention

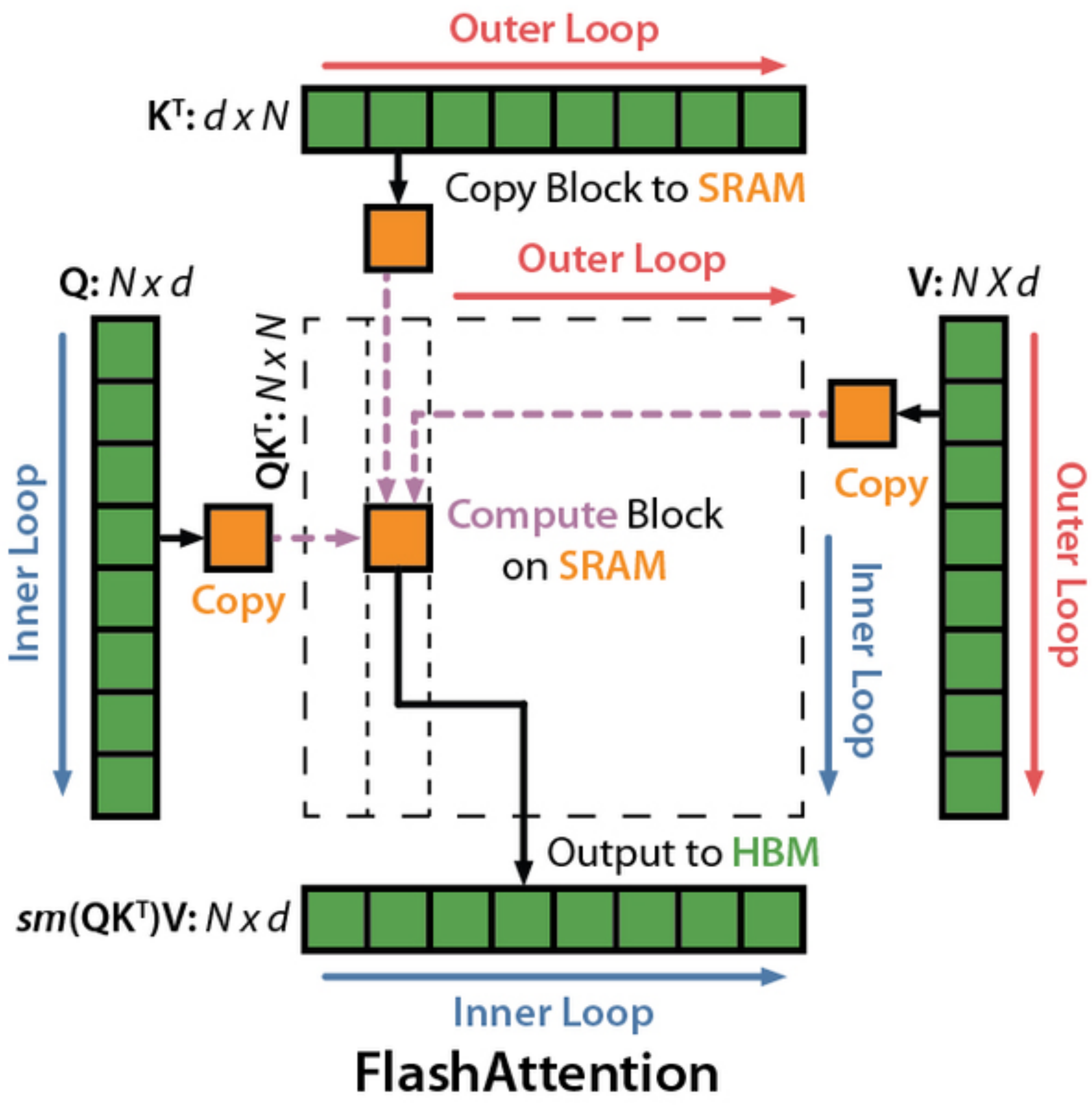


[1] Tri Dao, et al. FlashAttention: Fast and memory-efficient exact attention with io-awareness. 2022.

Flash Attention

Discussion

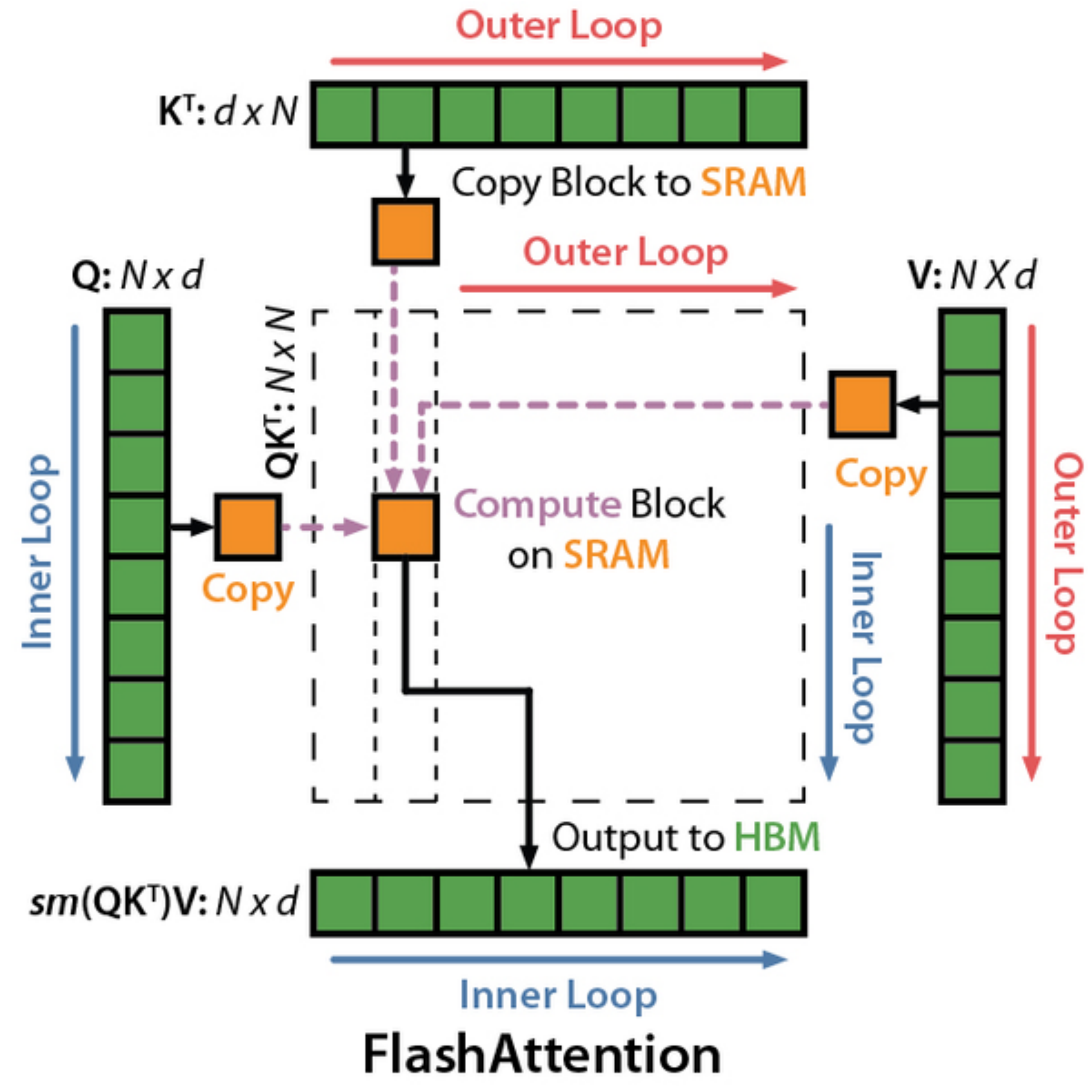
- Faster than regular attention
  - ~2x PyTorch Attention
  - 30-40% of just matmul (upper bound)
- More memory efficient
  - Attention computed but never stored

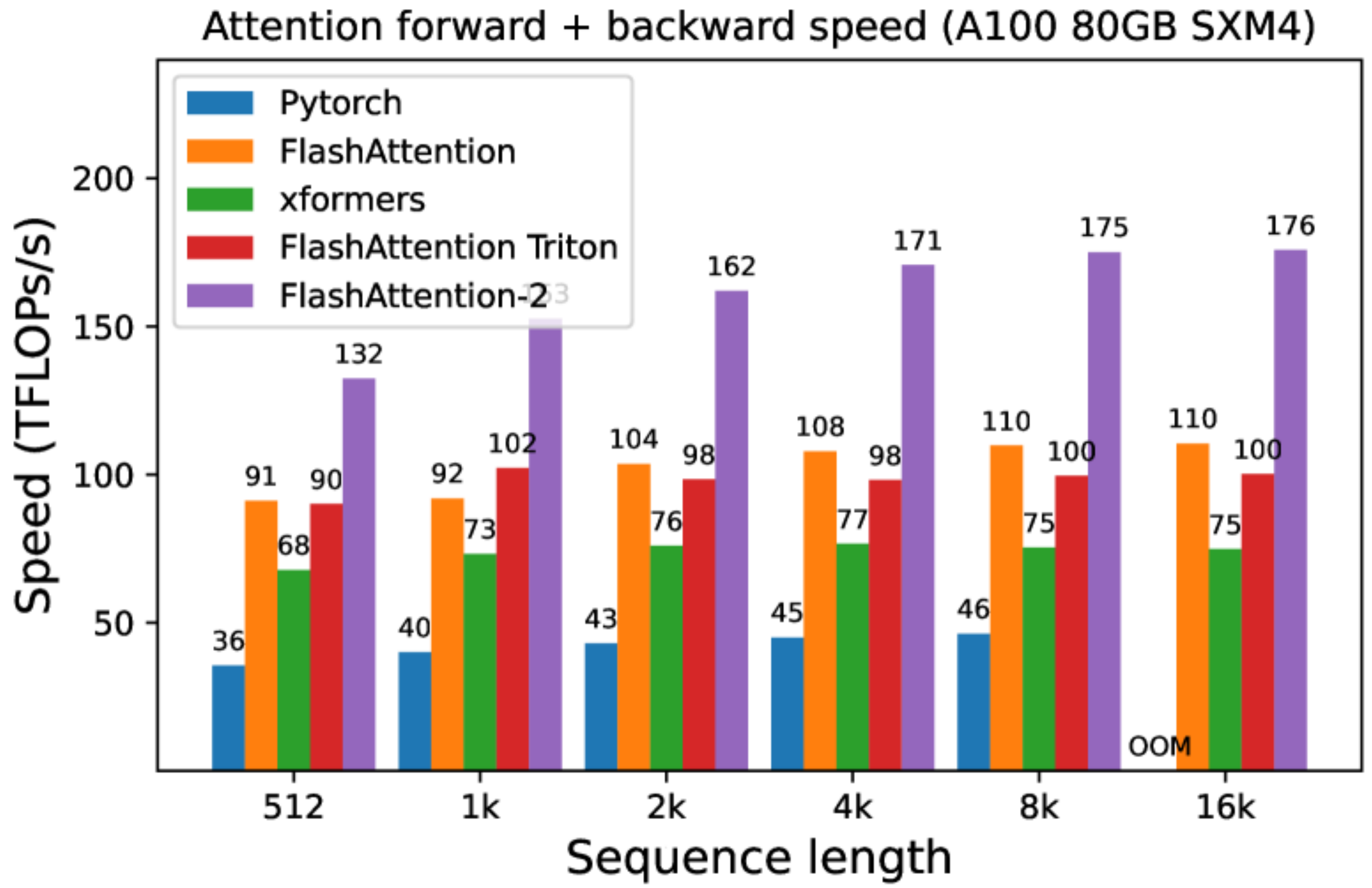


[1] Tri Dao, et al. FlashAttention: Fast and memory-efficient exact attention with io-awareness. 2022.

Flash Attention 2

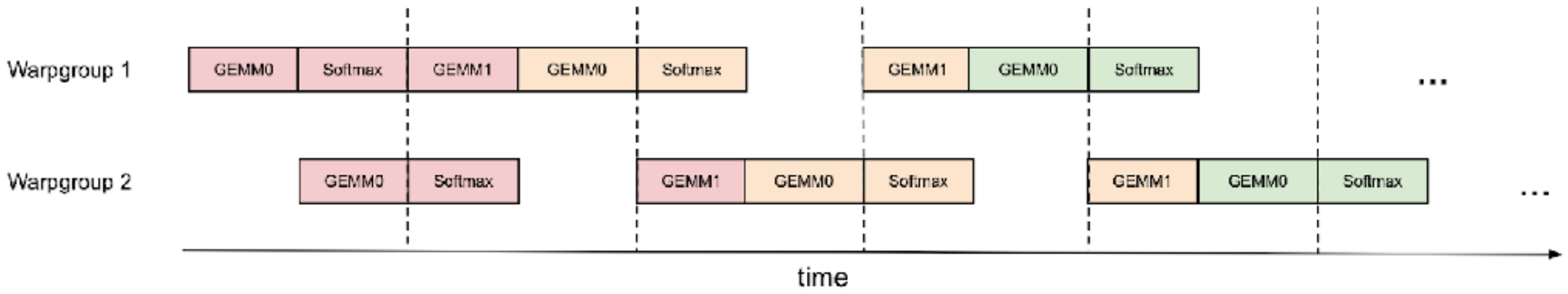
- Flash Attention
  - Compute output band-by-band
- Flash Attention 2
  - Compute output block-by-block
  - Fewer general purpose OPs
  - More matmuls
  - More parallelism; 70% of max possible



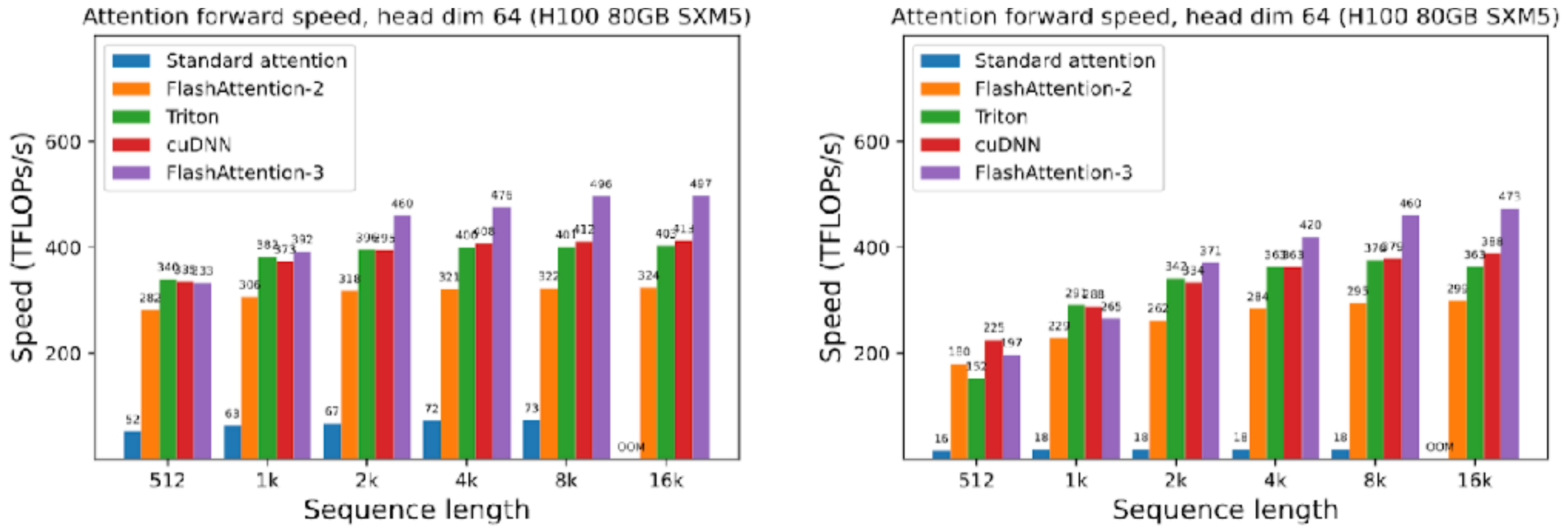


[1] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. 2023.

Flash Attention 3



- H100 GPU specific improvements
  - Better scheduling
    - Run GEMM + softmax in parallel
  - New GPU specific ops
  - More parallelism; 75% of max possible



(a) Forward, without causal mask, head dim 64

(b) Forward, with causal mask, head dim 64

[1] Jay Shah, et al. FlashAttention-3: Fast and accurate attention with asynchrony and low-precision. 2024

Other operations

- Fused operations
  - Special kernels: e.g. Matmul + non-linear
  - E.g. https://github.com/linkedin/Liger-Kernel
- Chunking
  - Breaking single forward call into multiple calls with less memory each

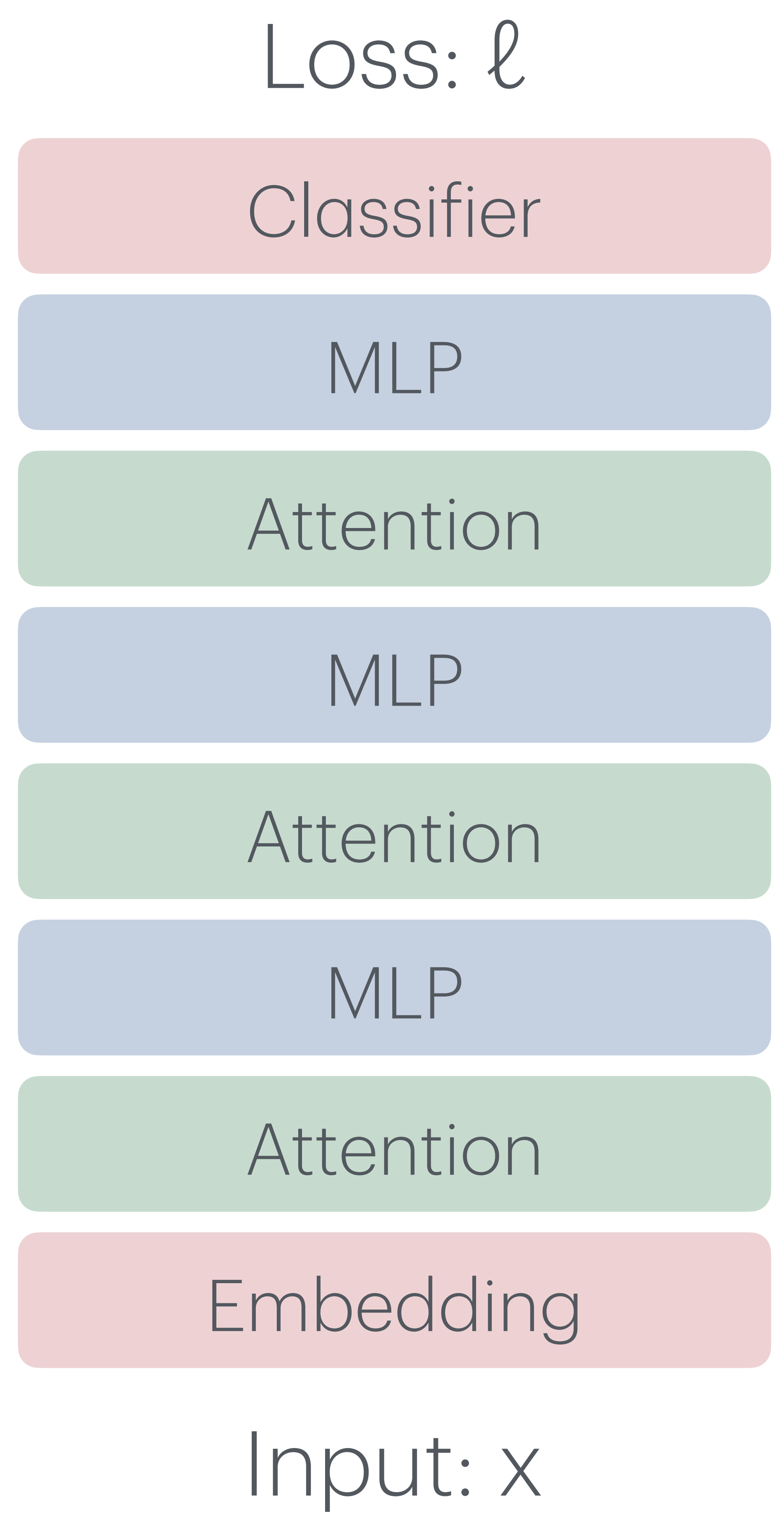
torch.compile

- Optimize PyTorch model automatically
  - Fuse OPs in Triton
  - Backend / GPU specific ops
  - Translation to runtime engines (e.g. TensorRT)
- Similar to programming lang. compilers
  - Not as fast as custom kernels yet, but soon

The full picture

Memory use

- Weights, Gradients, Momentum
  - Quantization, LoRA, Galore
- Activations
  - Checkpointing
- Intermediate/Scratch memory
  - Specialized implementations
  - torch.compile



References

- [1] Tri Dao, et al. FlashAttention: Fast and memory-efficient exact attention with io-awareness. 2022. (link)
- [2] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. 2023. (link)
- [3] Jay Shah, et al. FlashAttention-3: Fast and accurate attention with asynchrony and low-precision. 2024. (link)