15-442/15-642: Machine Learning Systems:

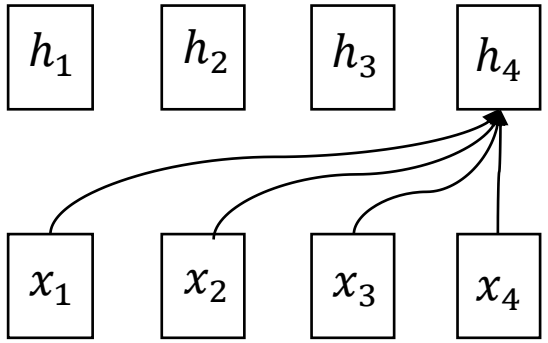
Transformer, Attention, and Optimizations

Tianqi Chen and Zhihao Jia
Carnegie Mellon University

Attention Mechanism

Refer to approach where individual states are combined using weights

Attention output



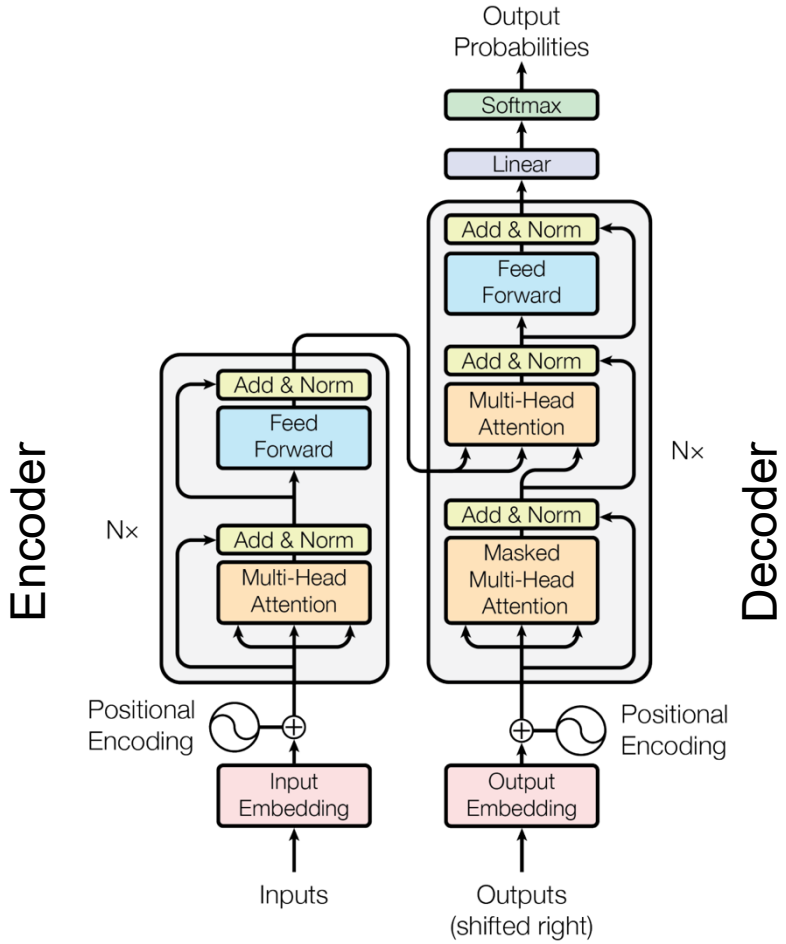
Hidden states from previous layer

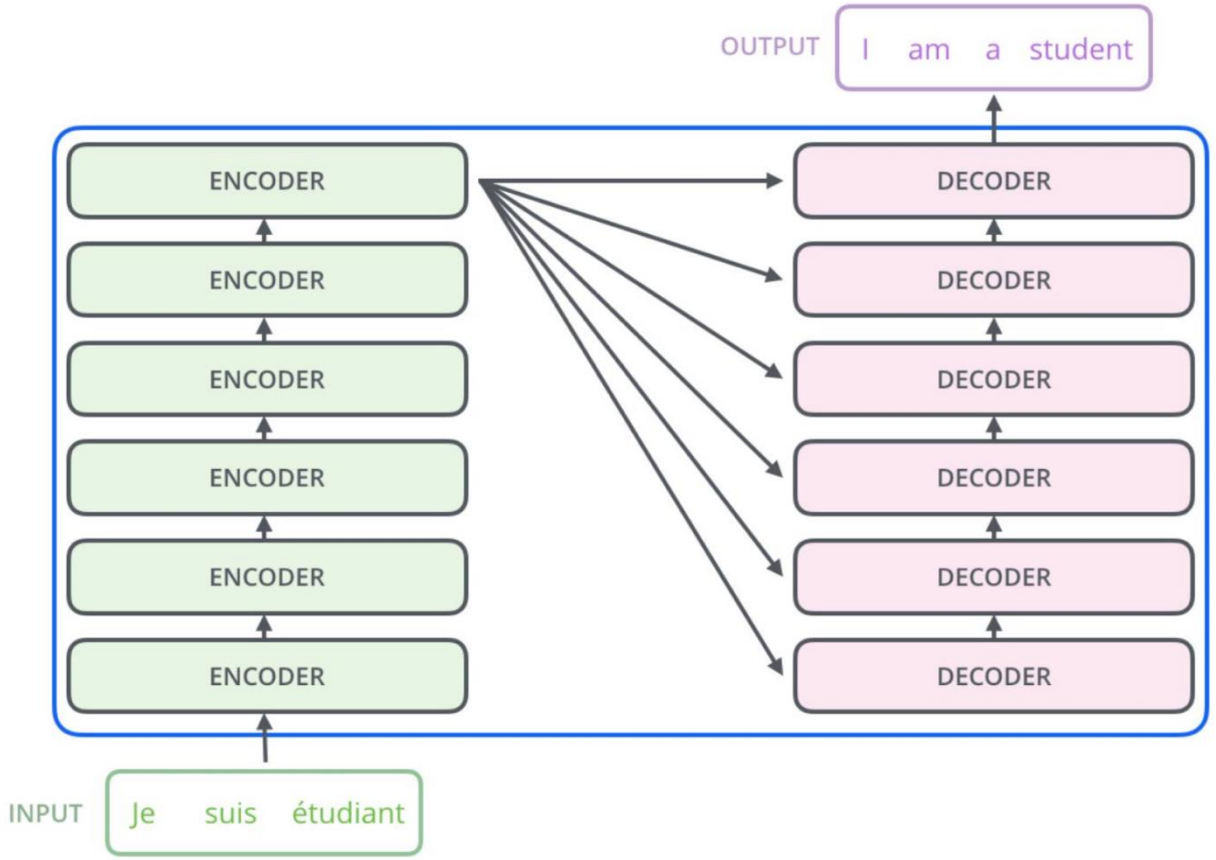
$$h_t = \sum_{i=1}^{t} s_i x_t$$

Intuitively s_i is "attention score" that computes how relevant the position i's input is to this current hidden output

There are different methods to decide how attention score is being computed

Transformer: Self-Attention Mechanism for Language Models

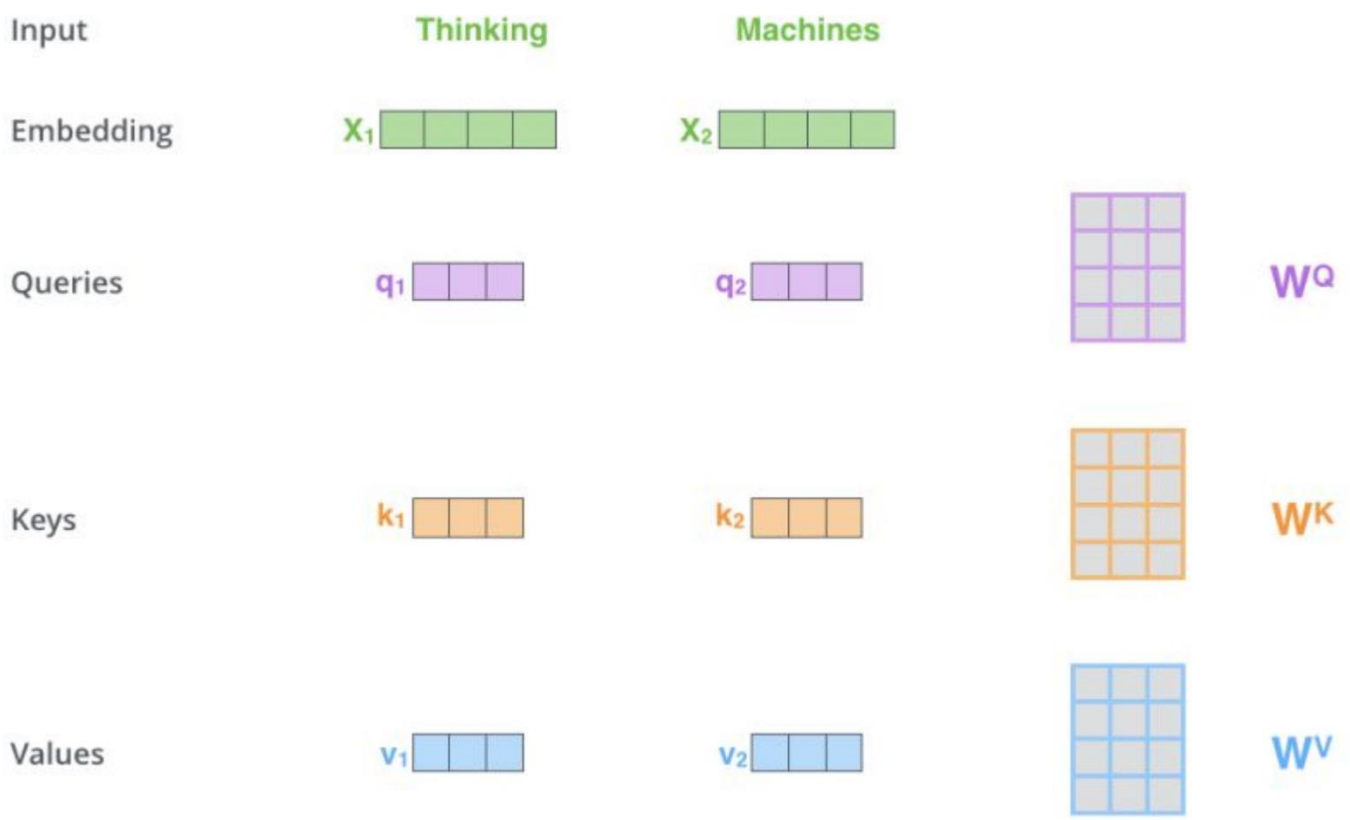




Ashish Vaswani et. al. Attention is all you need.

Self-Attention

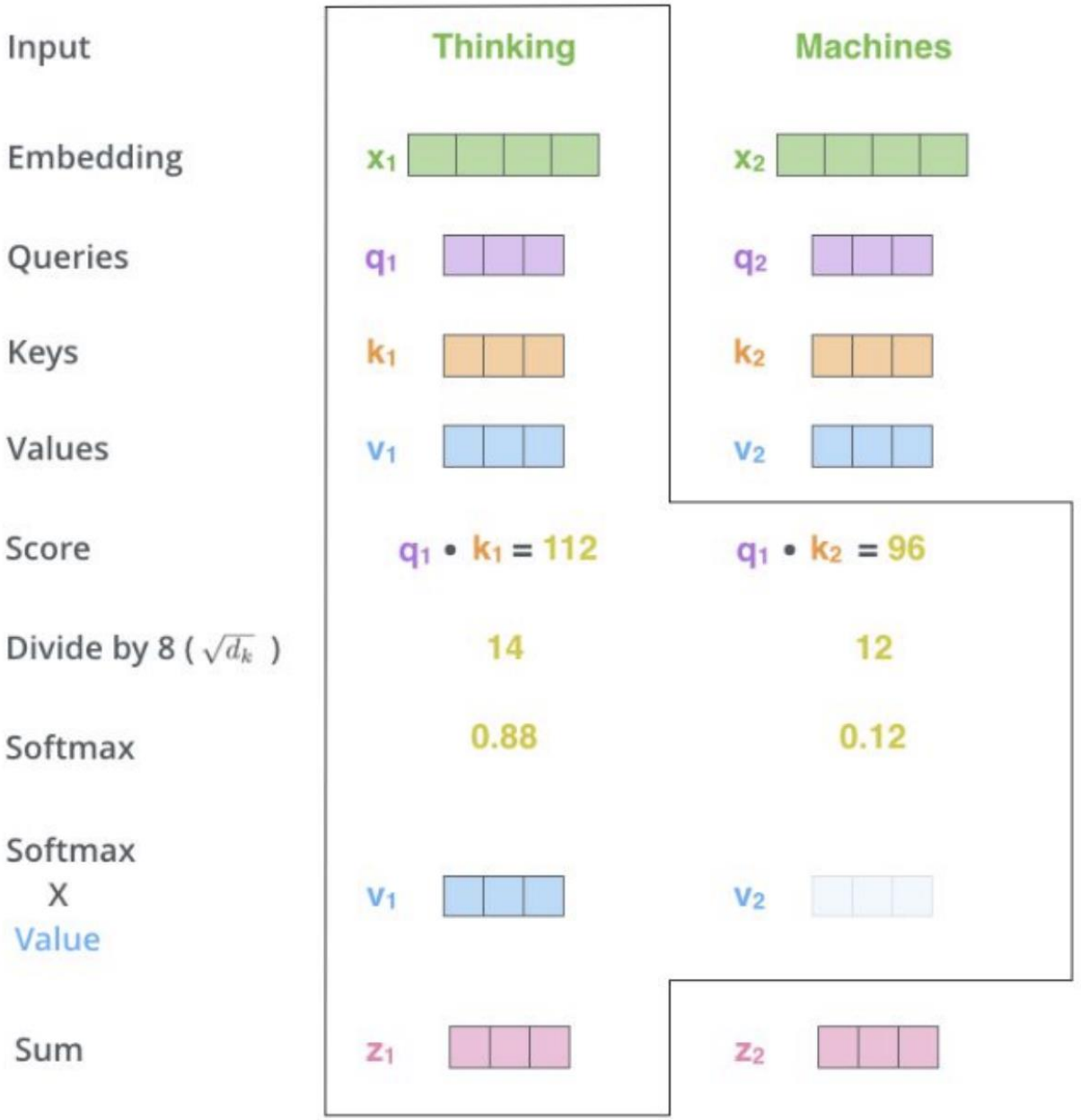
- Mapping a query and a set of key-value pairs to an output



Slide credit: Jay Allamar

Self-Attention

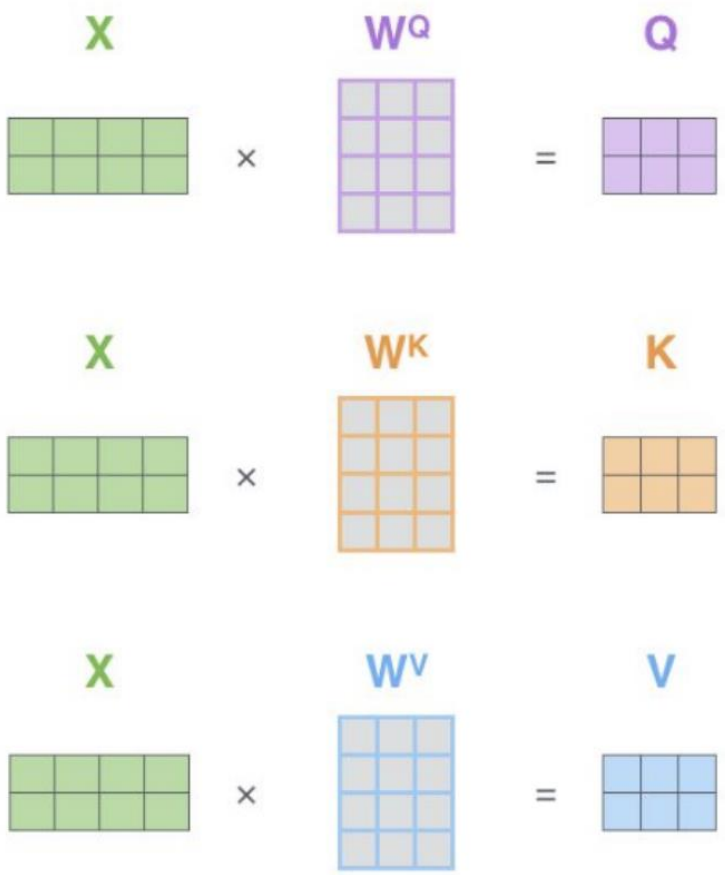
- Mapping a query and a set of key-value pairs to an output



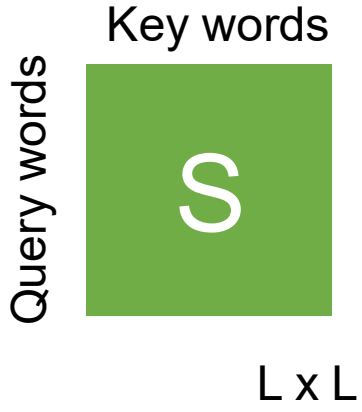
Slide credit: Jay Allamar

Self-Attention

- Multiple matrix multiplications

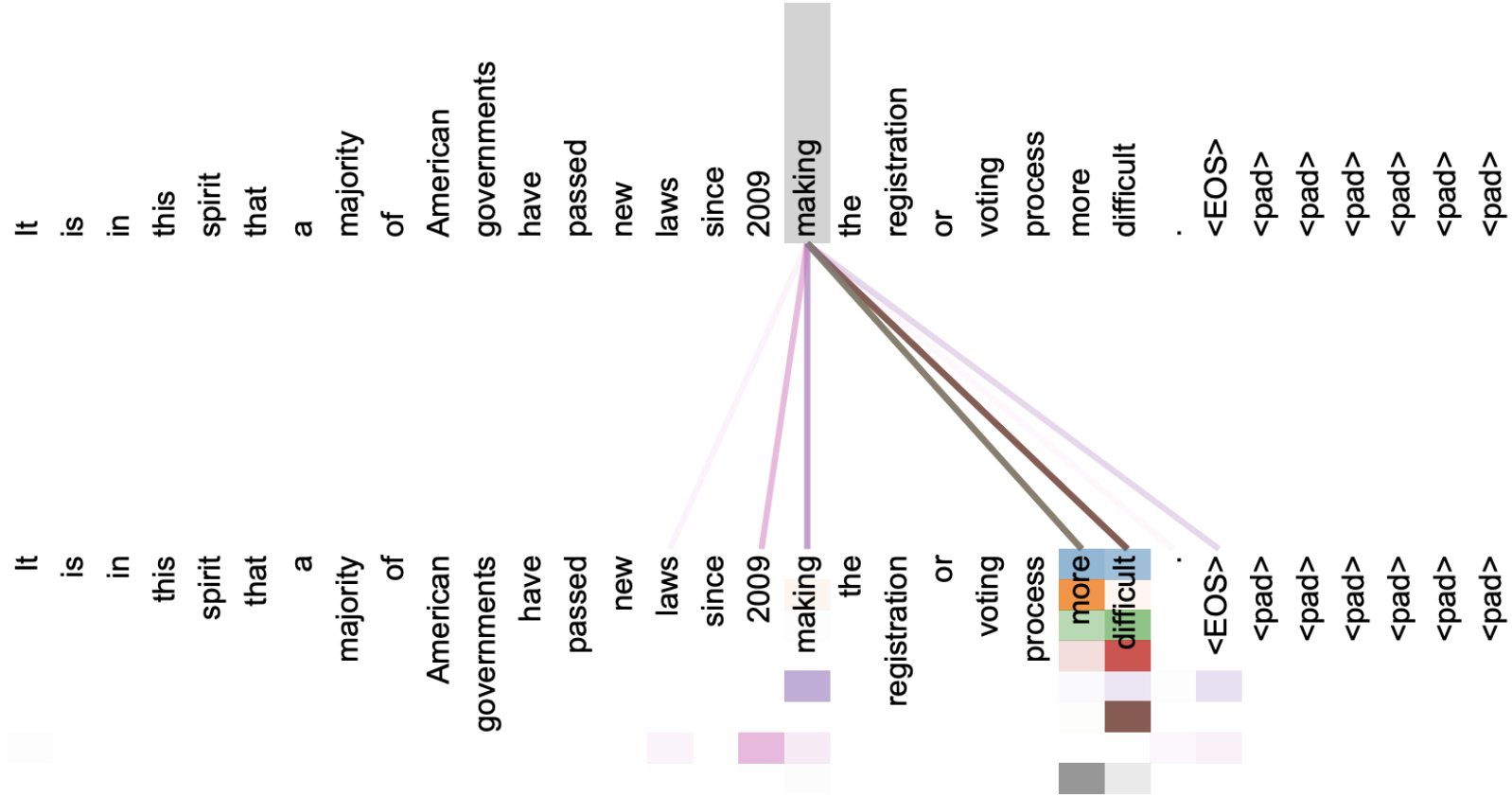


$$A(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$



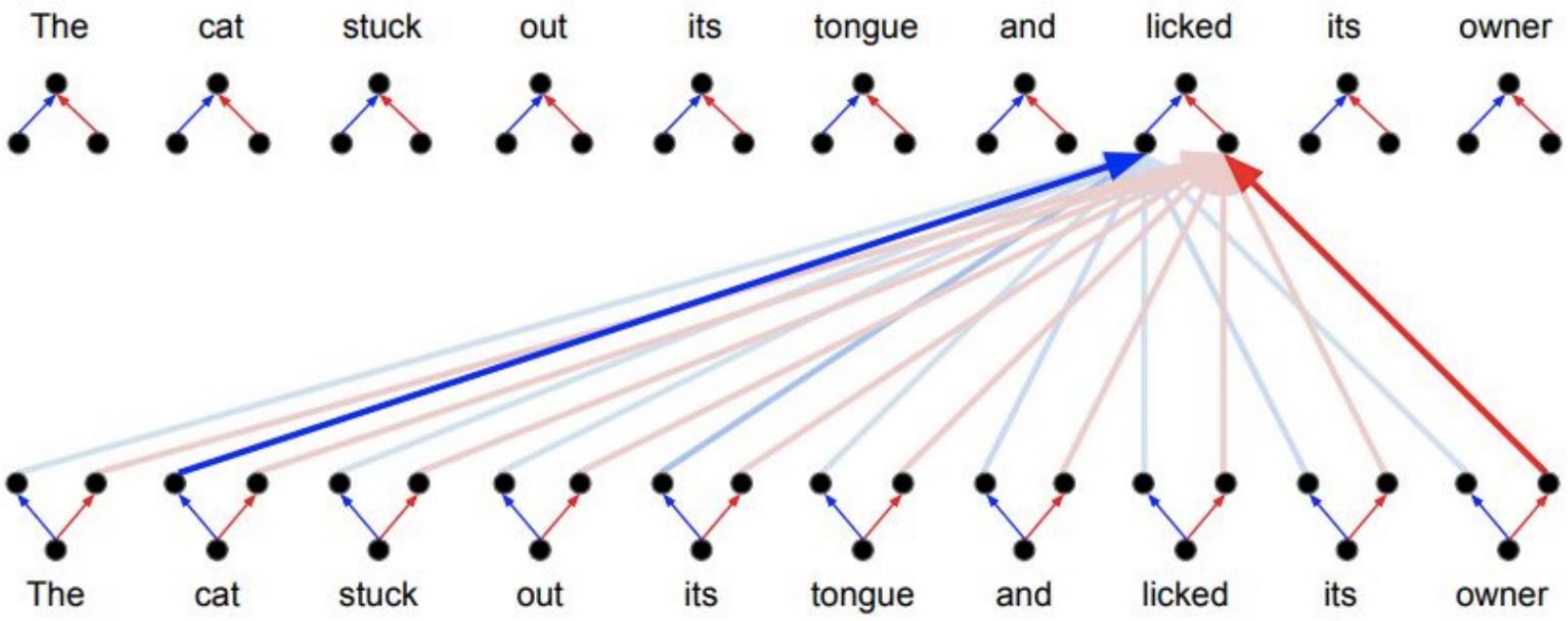
Slide credit: Jay Allamar

Self-Attention

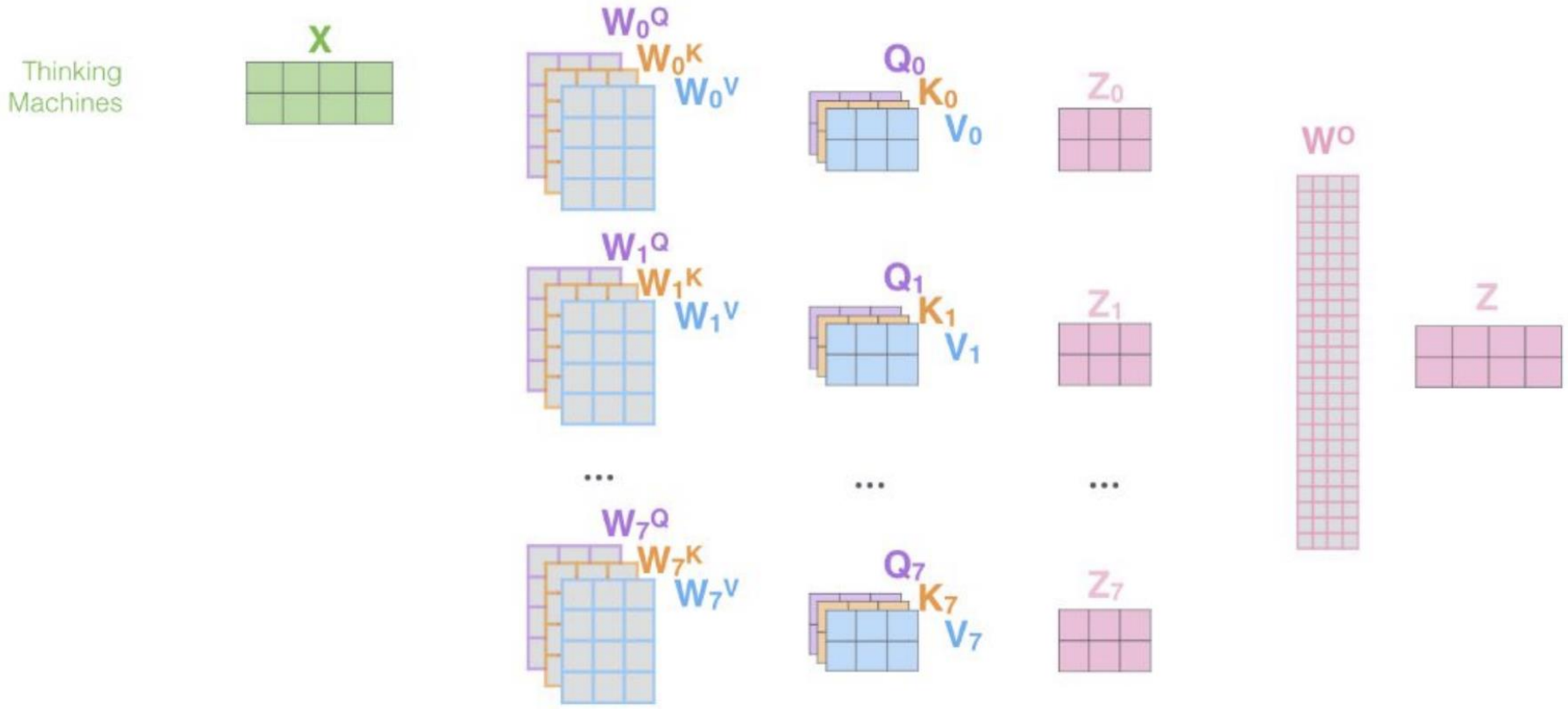


Multi-Head Self-Attention

- Parallelize attention layers with different linear transformations on input and output
- **Benefits: more parallelism, reduced computation cost**



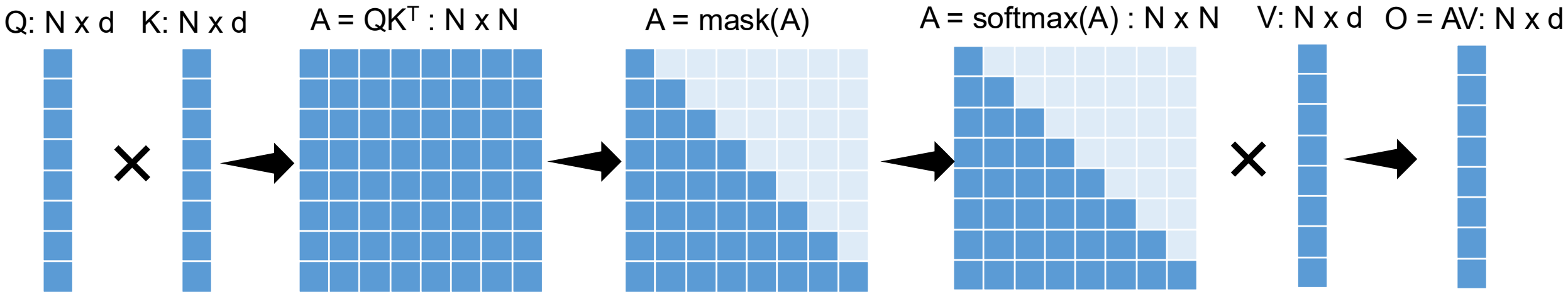
Multi-Head Self-Attention



$$Z_i = A(Q_i, K_i, V_i) = softmax\left(\frac{Q_i {K_i}^T}{\sqrt{d}}\right) V_i$$

$$Z = MultiHead(Q, K, V) = Concat(Z_0, \dots, Z_7) W^o$$

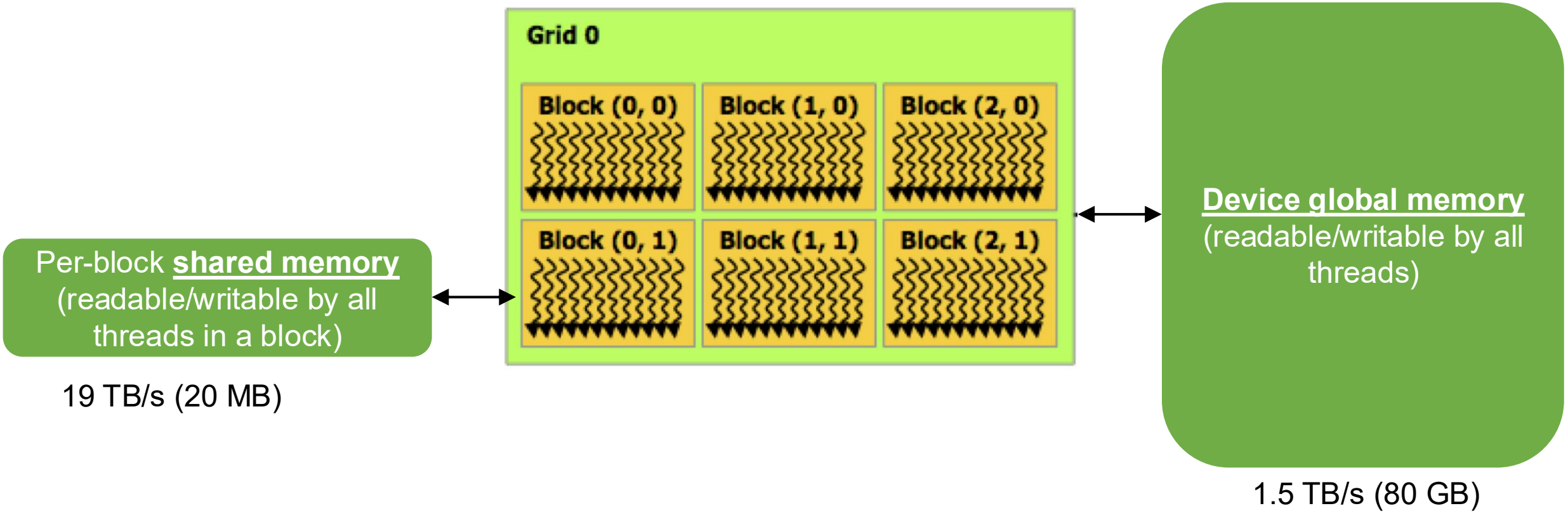
How to Compute Attention on GPUs?
$O = \text{Softmax}(QK^T)\,V$



Challenges:

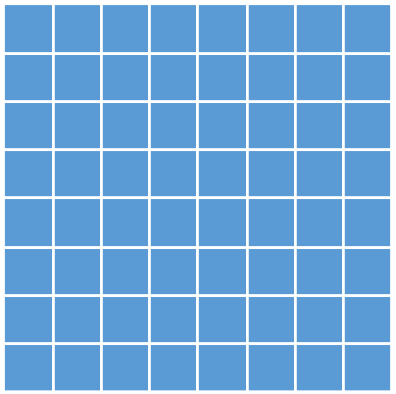
- Large intermediate results
- Repeated reads/writes from GPU device memory
- Cannot scale to long sequences due to O(N^2) intermediate results

Revisit: GPU Memory Hierarchy



FlashAttention

$A = \text{softmax}(QK^T)$



Key idea: compute attention by blocks to reduce global memory access

Two main Techniques:

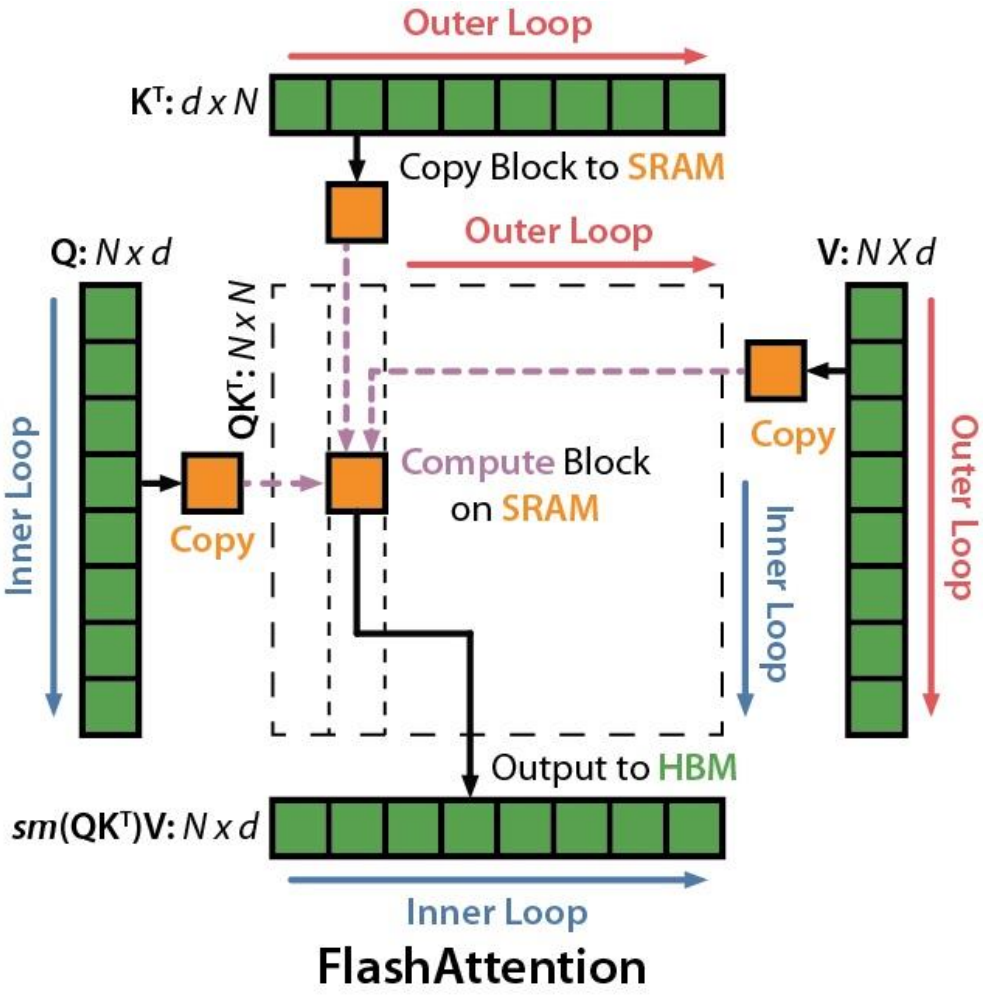
1. Tiling: restructure algorithm to load query/key/value block by block from global to shared memory

2. Recomputation: don't store attention matrix from forward, recompute it in backward

* FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

Tiling: Decompose Large Softmax into smaller ones by Scaling

1. Load inputs by blocks from global to shared memory
2. On chip, compute attention output wrt the block
3. Update output in device memory by scaling



Safe Softmax and Online Softmax $\frac{e^{x_i}}{\sum_j e^{x_j}}$

- **Issue**: maximum value for 16-bit floating point is 65504 ($< e^{12}$)
- To avoid overflow, the softmax of vector x is computed as

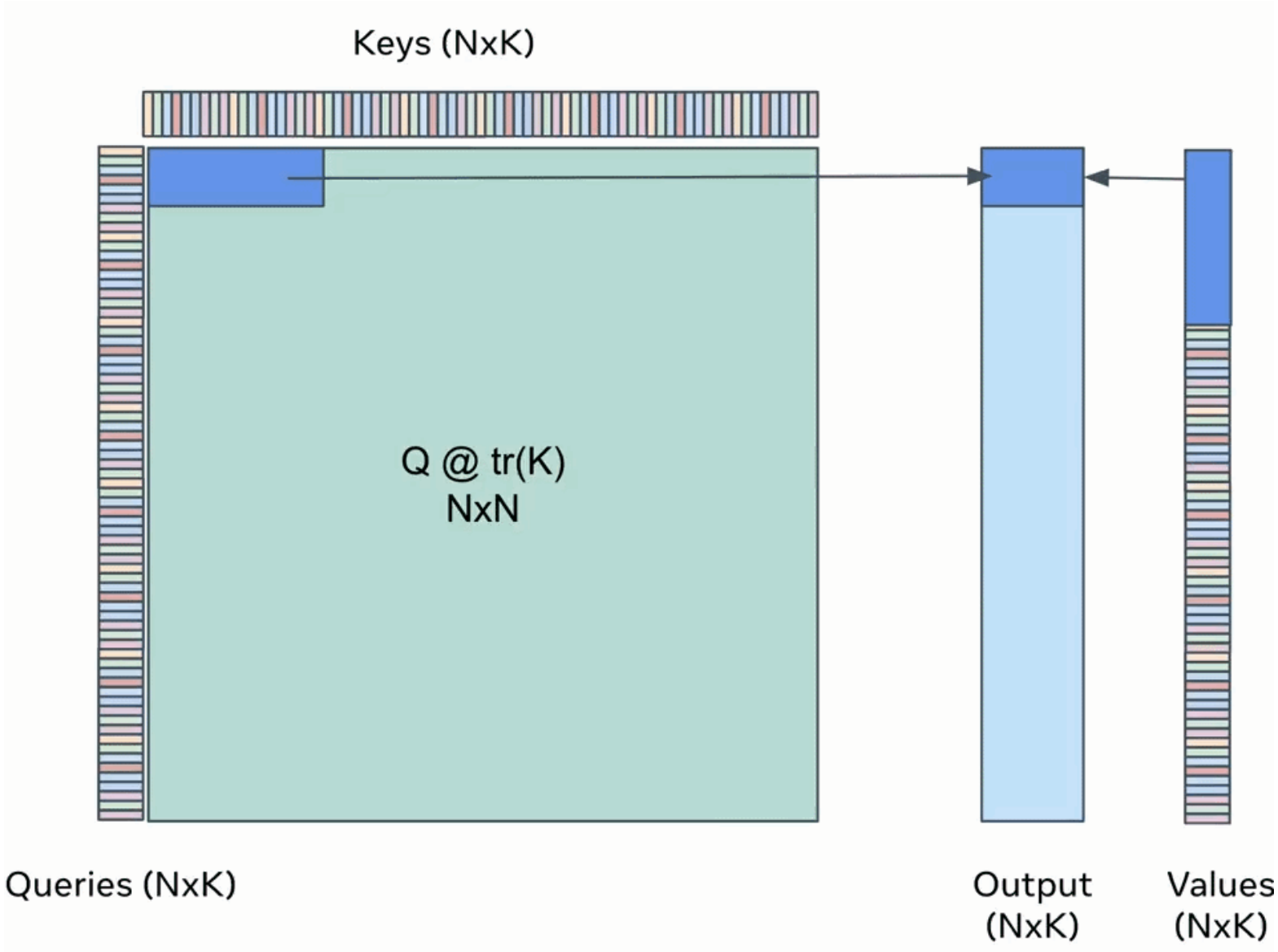
$$m(x) := \max_i \; x_i, \quad f(x) := \left[e^{x_1 - m(x)} \quad \dots \quad e^{x_B - m(x)}\right], \quad \ell(x) := \sum_i f(x)_i, \quad \text{softmax}(x) := \frac{f(x)}{\ell(x)}.$$

- For two vectors $x^{(1)}$ and $x^{(2)}$, we compute the softmax of $[x^{(1)}\ x^{(2)}]$

$$m(x) = m(\left[x^{(1)}\ x^{(2)}\right]) = \max(m(x^{(1)}), m(x^{(2)})), \quad f(x) = \left[e^{m(x^{(1)}) - m(x)} f(x^{(1)}) \quad e^{m(x^{(2)}) - m(x)} f(x^{(2)})\right],$$

$$\ell(x) = \ell(\left[x^{(1)}\ x^{(2)}\right]) = e^{m(x^{(1)}) - m(x)} \ell(x^{(1)}) + e^{m(x^{(2)}) - m(x)} \ell(x^{(2)}), \quad \text{softmax}(x) = \frac{f(x)}{\ell(x)}.$$

Tiling



Animation credit: Francisco Massa

FlashAttention 2 Algorithm

Algorithm 1 FLASHATTENTION-2 forward pass

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, block sizes B_c, B_r.

Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \ldots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \ldots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \ldots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.

Divide the output $\mathbf{O} \in \mathbb{R}^{N \times d}$ into T_r blocks $\mathbf{O}_i, \ldots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, and divide the logsumexp L into T_r blocks $L_i, \ldots, L_{T_r}$ of size B_r each.

for $1 \le i \le T_r$ **do**

  Load $\mathbf{Q}_i$ from HBM to on-chip SRAM.

  On chip, initialize $\mathbf{O}_i^{(0)} = (0)_{B_r \times d} \in \mathbb{R}^{B_r \times d}, \ell_i^{(0)} = (0)_{B_r} \in \mathbb{R}^{B_r}, m_i^{(0)} = (-\infty)_{B_r} \in \mathbb{R}^{B_r}$.

  for $1 \le j \le T_c$ **do**

    Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.

    On chip, compute $\mathbf{S}_i^{(j)} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $m_i^{(j)} = \max(m_i^{(j-1)}, \text{rowmax}(\mathbf{S}_i^{(j)})) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_i^{(j)} = \exp(\mathbf{S}_i^{(j)} - m_i^{(j)}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\ell_i^{(j)} = e^{m_i^{j-1} - m_i^{(j)}} \ell_i^{(j-1)} + \text{rowsum}(\tilde{\mathbf{P}}_i^{(j)}) \in \mathbb{R}^{B_r}$.

    On chip, compute $\mathbf{O}_i^{(j)} = \text{diag}(e^{m_i^{(j-1)} - m_i^{(j)}})^{-1} \mathbf{O}_i^{(j-1)} + \tilde{\mathbf{P}}_i^{(j)} \mathbf{V}_j$.

  end for

  On chip, compute $\mathbf{O}_i = \text{diag}(\ell_i^{(T_c)})^{-1} \mathbf{O}_i^{(T_c)}$.

  On chip, compute $L_i = m_i^{(T_c)} + \log(\ell_i^{(T_c)})$.

  Write $\mathbf{O}_i$ to HBM as the i-th block of $\mathbf{O}$.

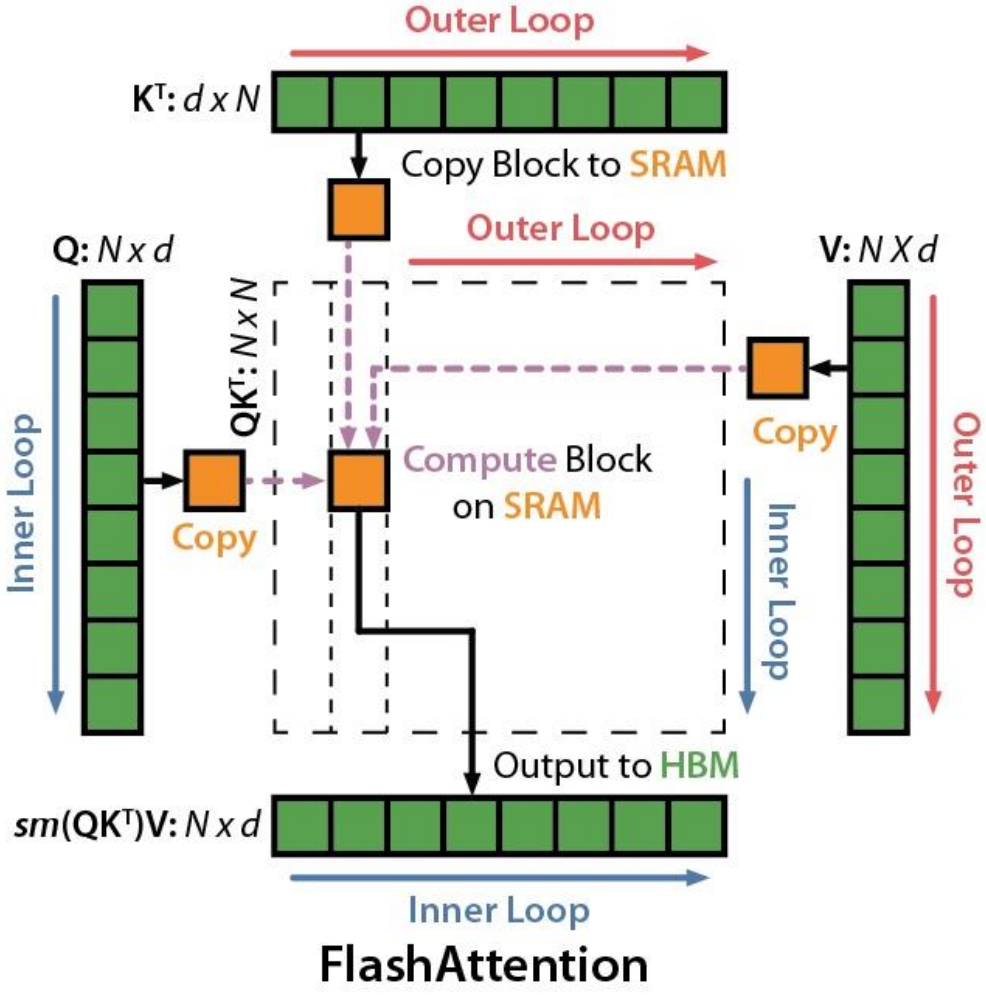
  Write L_i to HBM as the i-th block of L.

end for

Return the output $\mathbf{O}$ and the logsumexp L.

Recomputation: Backward Pass

By storing softmax normalization factors from forward (size N), recompute attention in the backward from inputs in shared memory



Attention	Standard	FlashAttention
GFLOPs	66.6	75.2
Global mem access	40.3 GB	4.4 GB
Runtime	41.7 ms	7.3 ms

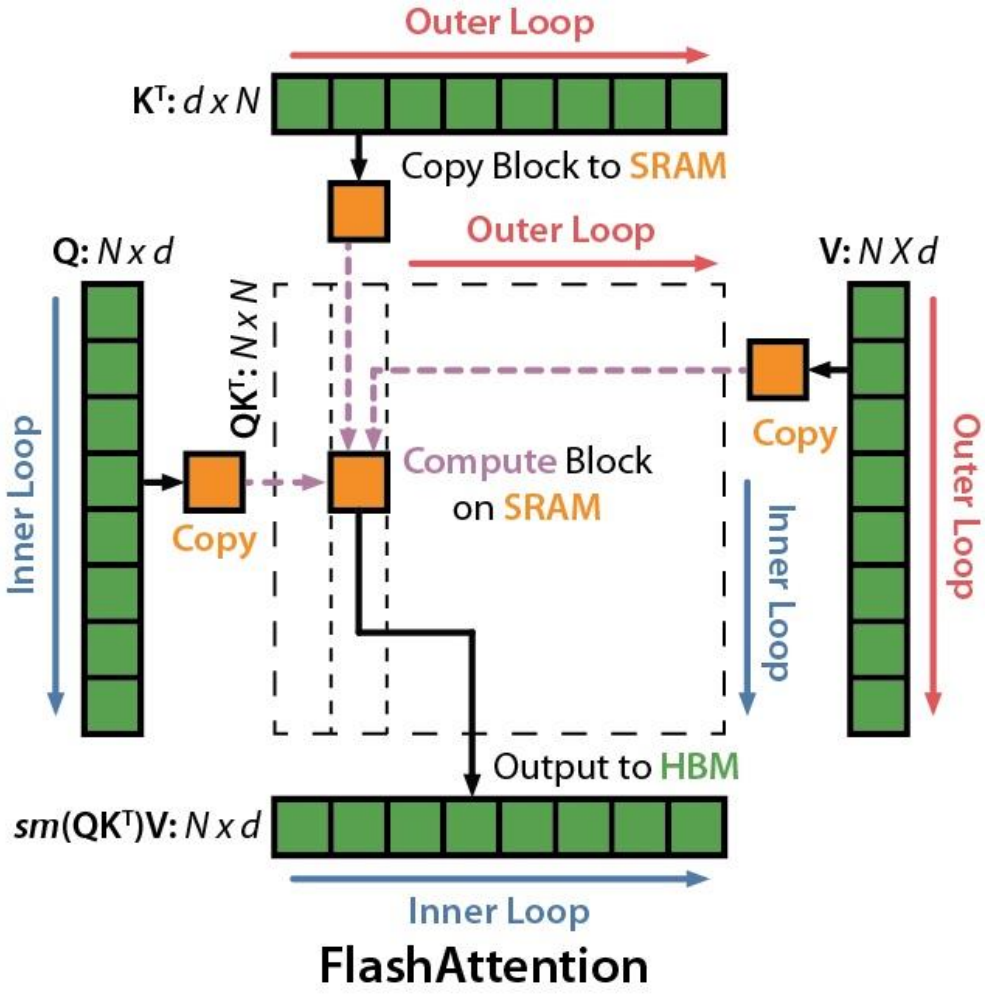
Speed up backward pass with increased FLOPs

FlashAttention: Threadblock-level Parallelism

How to partition FlasshAttention across thread blocks?

(An A100 has 108 SMMs -> 108 thread blocks)

- Step 1: assign different heads to different thread blocks (16-64 heads)

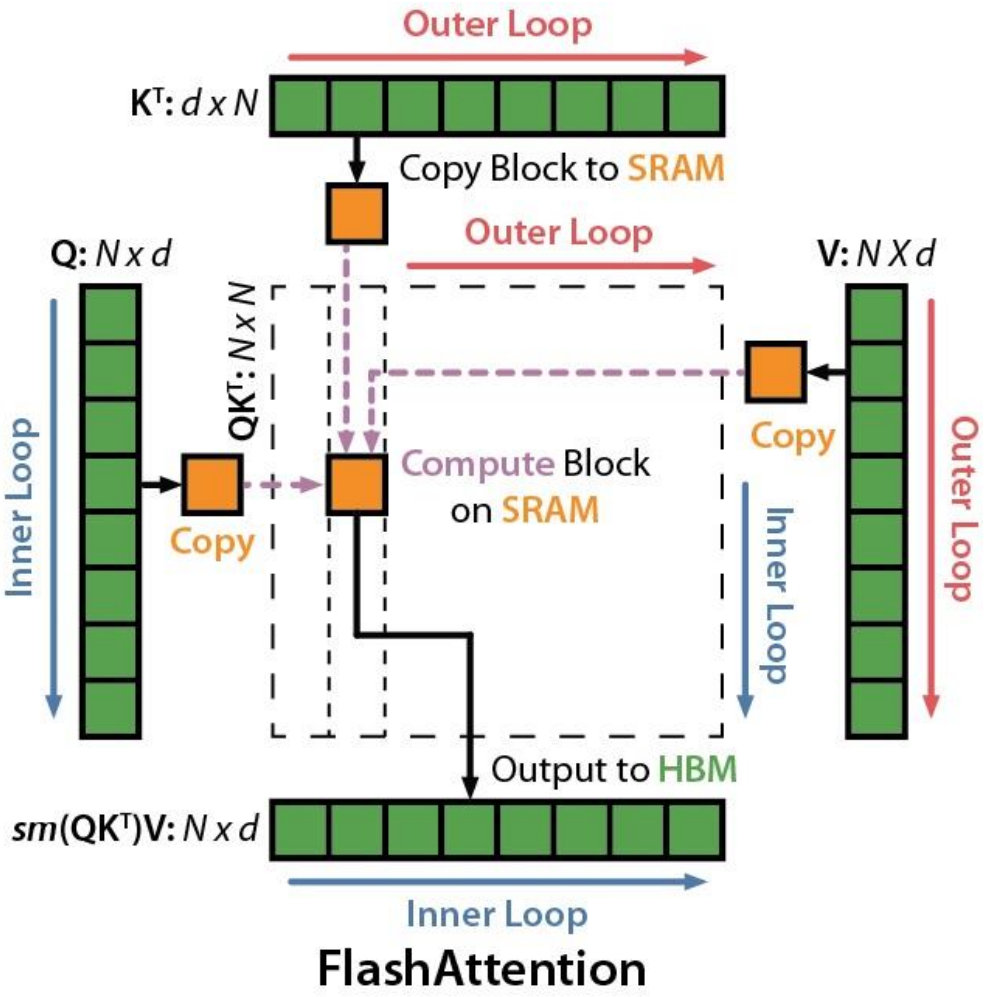


FlashAttention: Threadblock-level Parallelism

How to partition FlasshAttention across thread blocks?

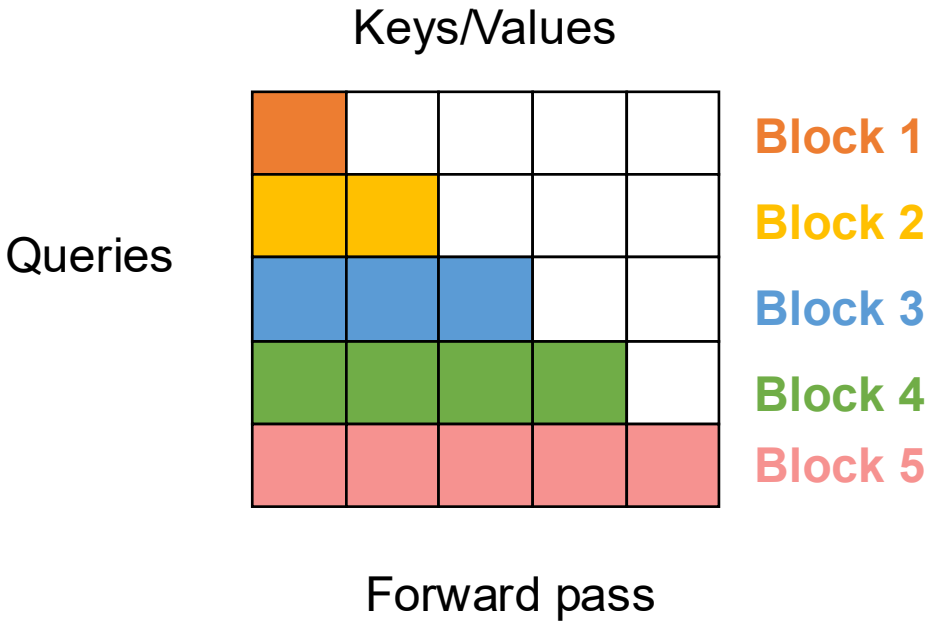
(An A100 has 108 SMMs -> 108 thread blocks)

- Step 1: assign different heads to different thread blocks (16-64 heads)
- Step 2: assign different queries to different thread blocks (Why?)



Thread blocks cannot communicate; cannot perform softmax when partitioning keys/values

FlashAttention: Threadblock-level Parallelism

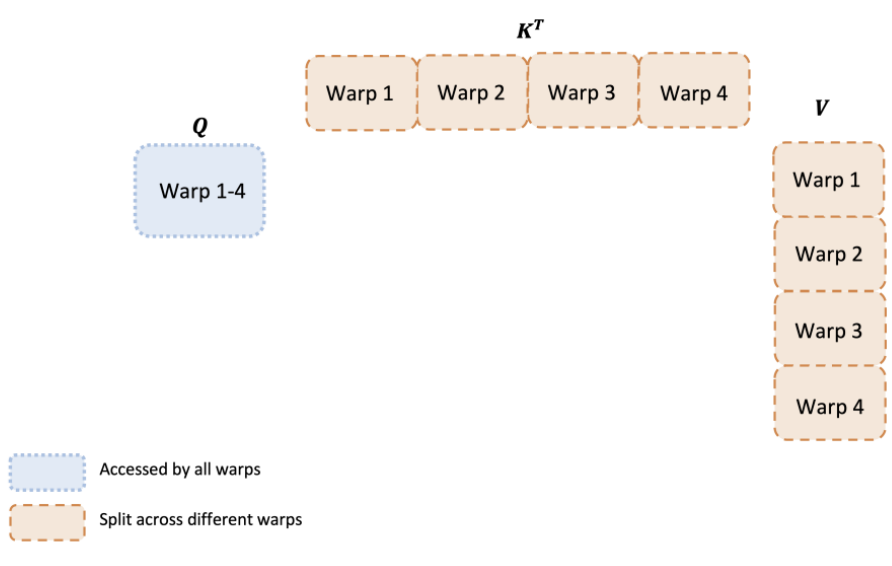


Do we need to handle workload imbalance?

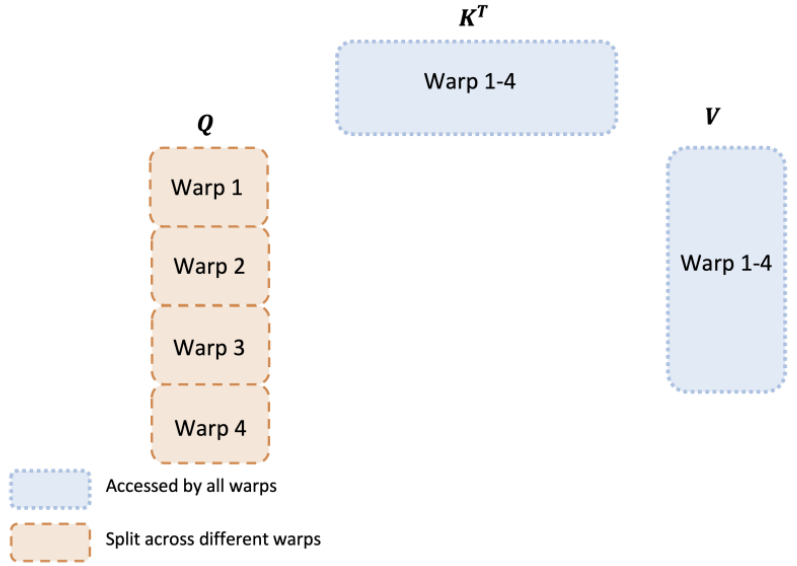
No. GPU scheduler automatically loads the next block once the current one completes.

FlashAttention: Warp-Level Parallelism

- How to partition FlashAttention across warps within a thread block?



(a) FlashAttention

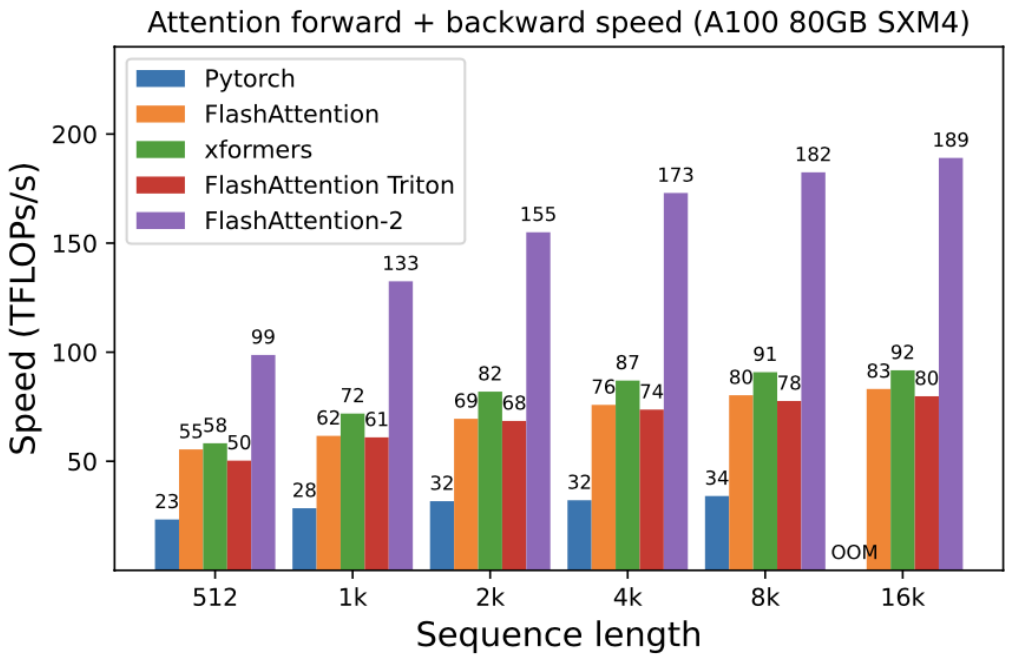


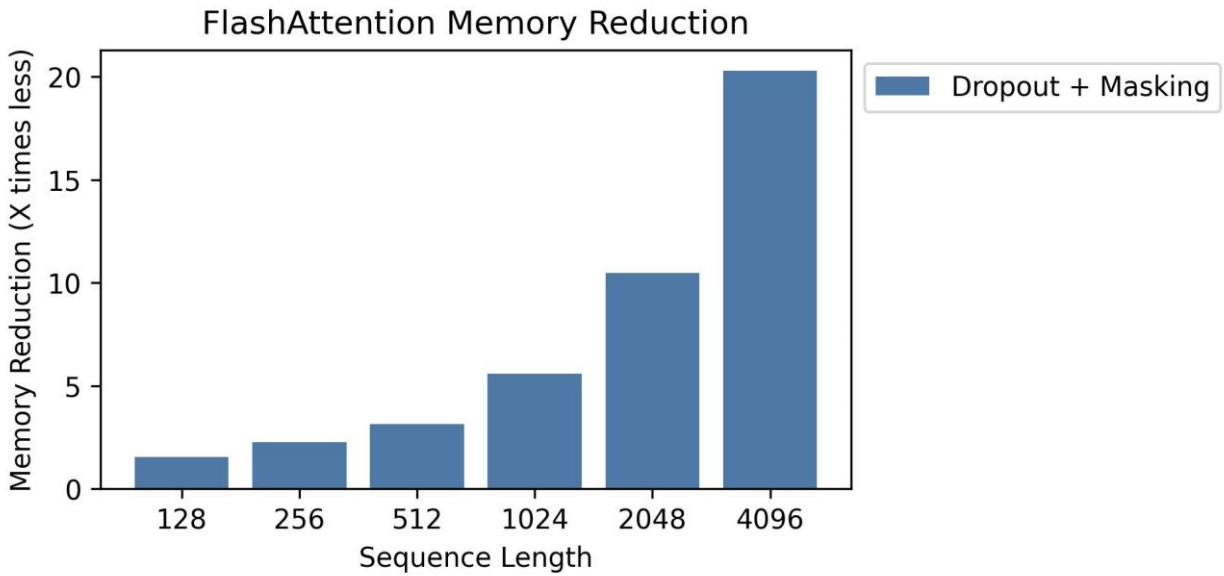
(b) FlashAttention-2

☹ Splitting across K/V requires communication to add results

Splitting across Q avoids communications ☺

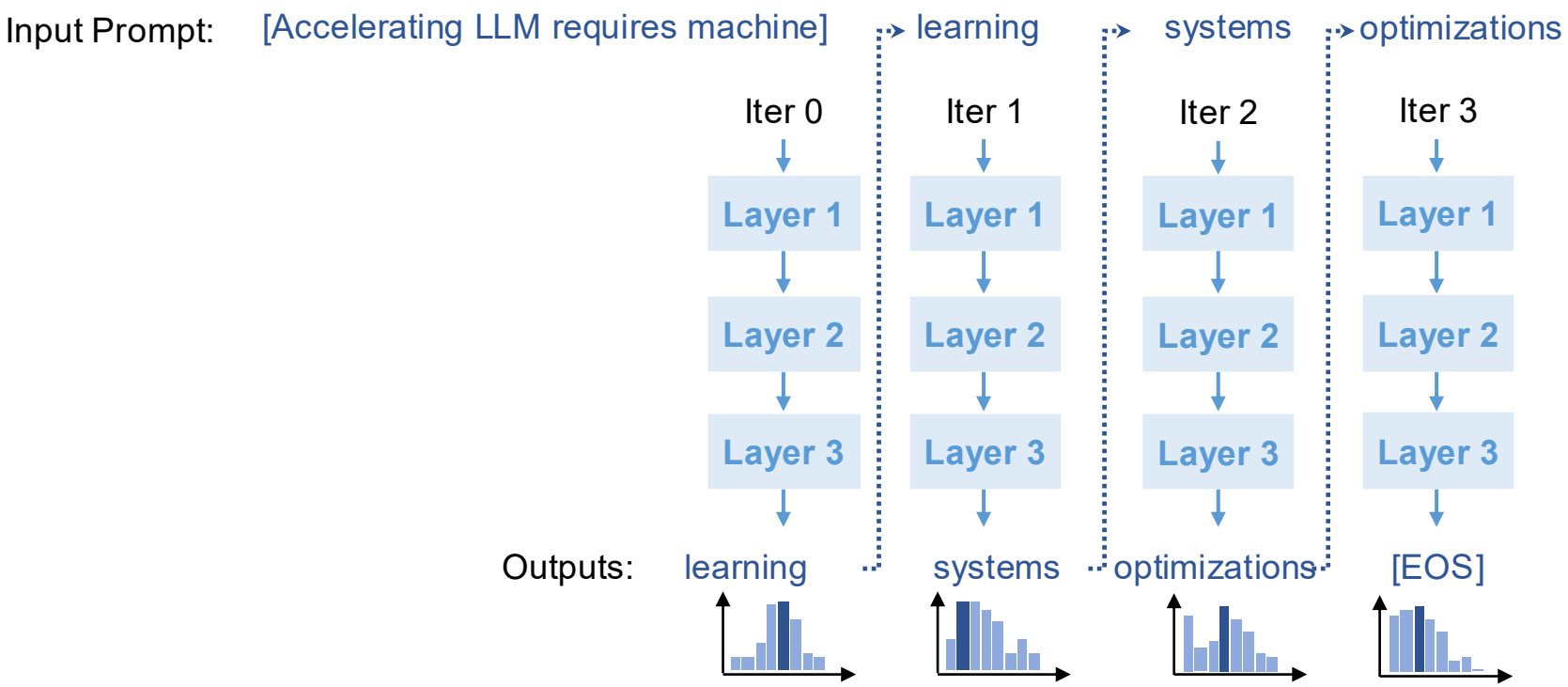
FlashAttention: 2-4x speedup, 10-20x memory reduction



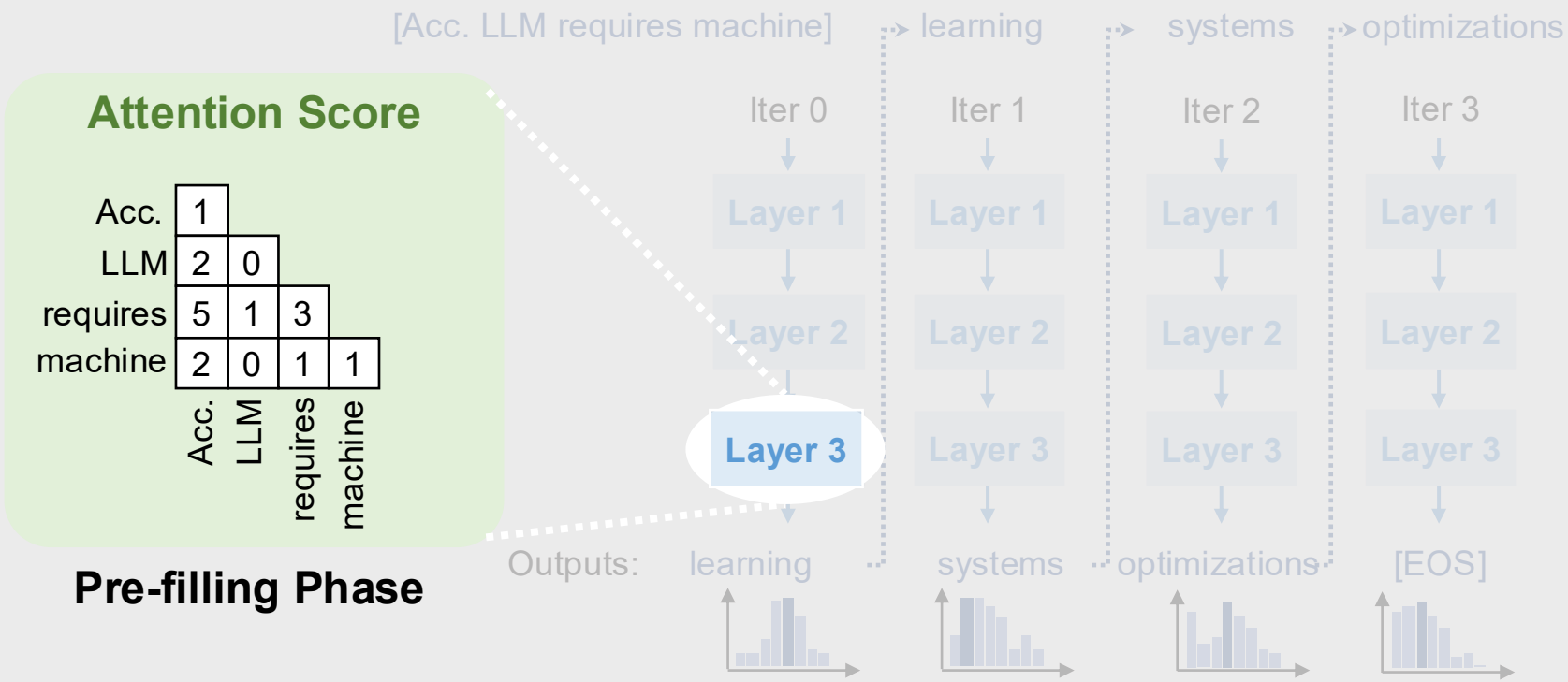


Memory linear in sequence length

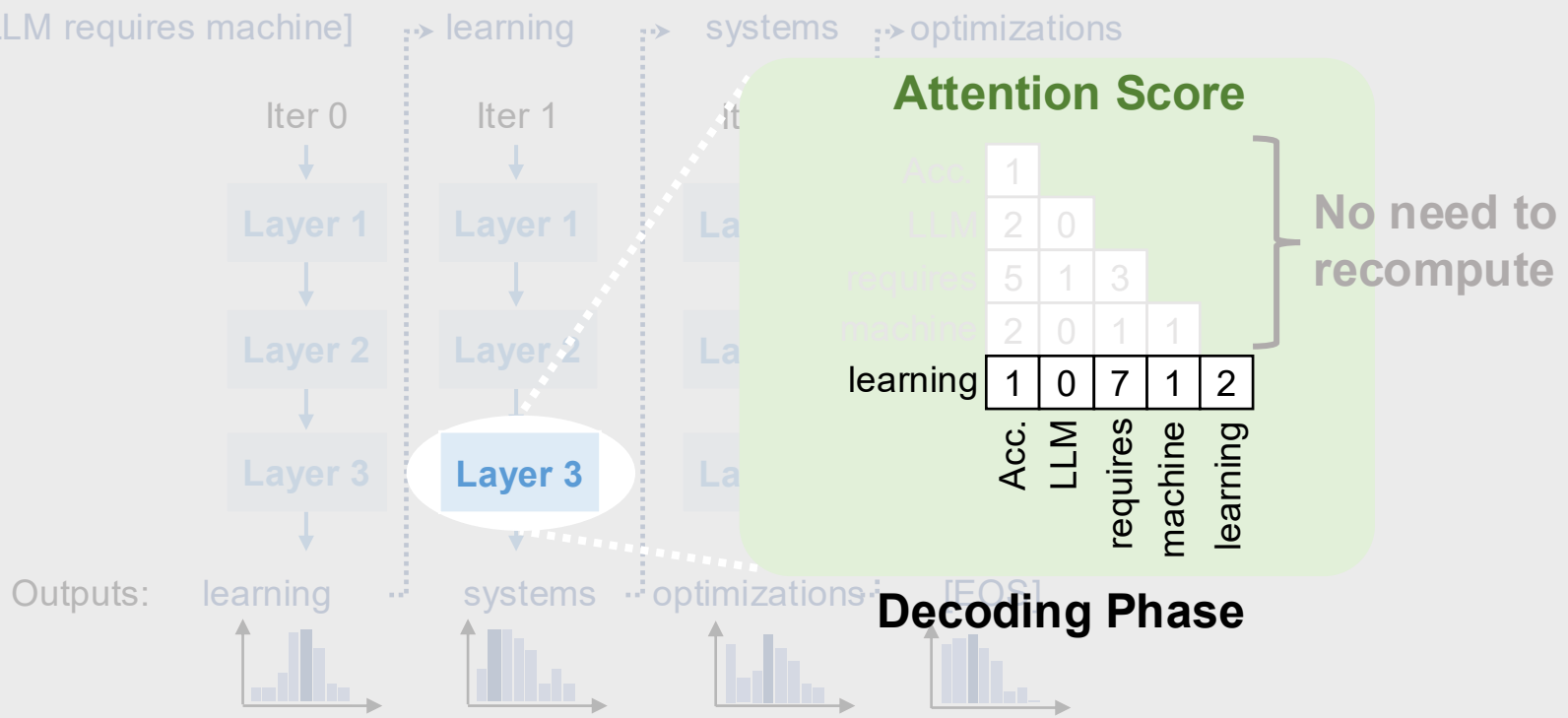
Generative LLM Inference: Autoregressive Decoding



Generative LLM Inference: Autoregressive Decoding



Generative LLM Inference: Autoregressive Decoding

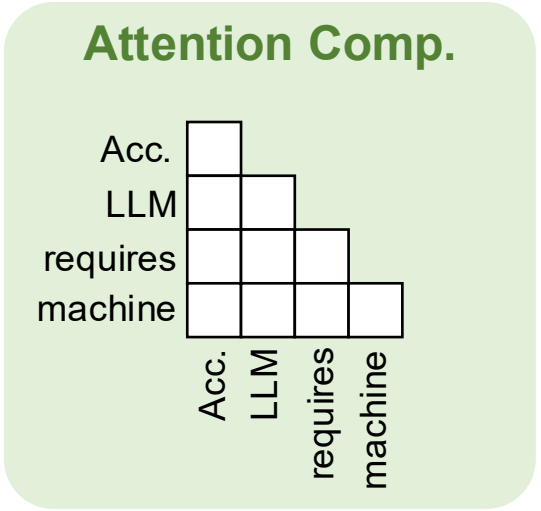


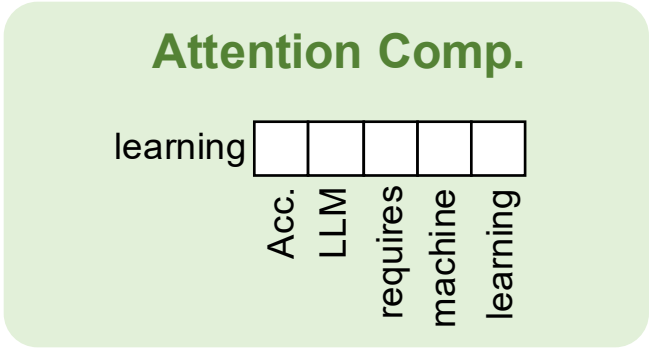
Generative LLM Inference: Autoregressive Decoding

- **Pre-filling phase** (0-th iteration):
  - Process ***all*** input tokens at once
- **Decoding phase** (all other iterations):
  - Process a ***single*** token generated from previous iteration
  - Use attention keys & values of all previous tokens

- Key-value cache:
  - Save attention keys and values for the following iterations to avoid recomputation

Can We Apply FlashAttention to LLM Inference?





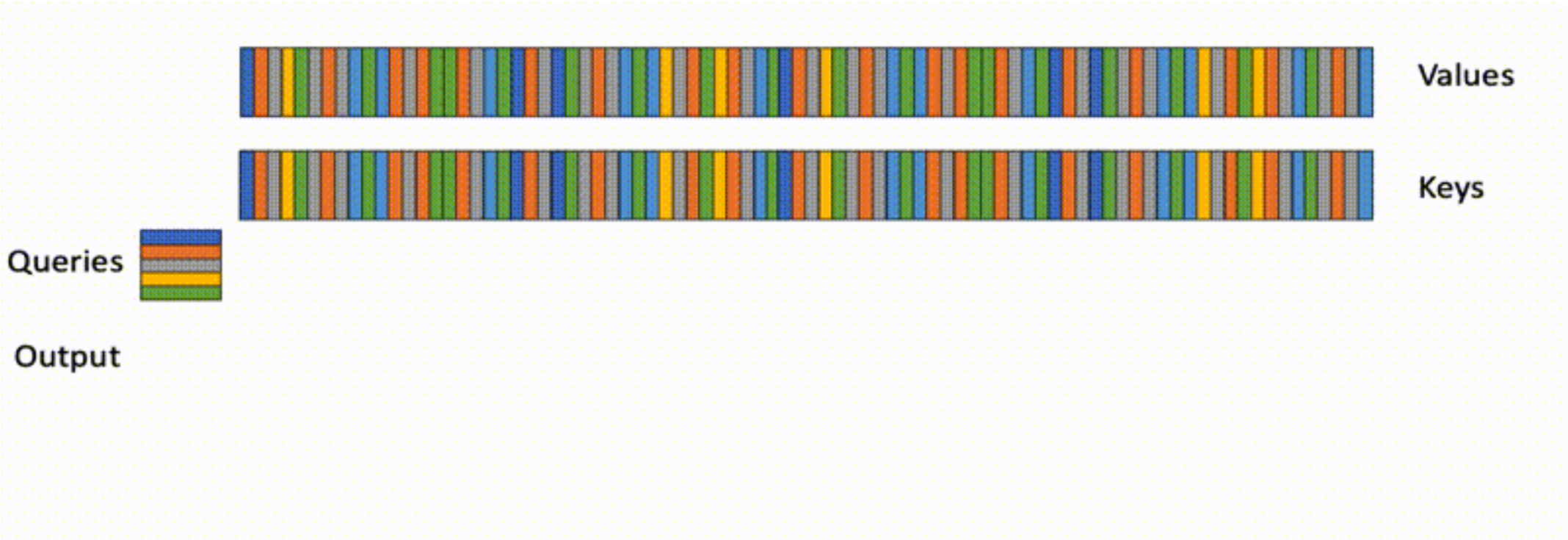
Pre-filling phase:

- Yes, compute different queries using different thread blocks/warps

Decoding phase:

- No, there is a single query in the decoding phase

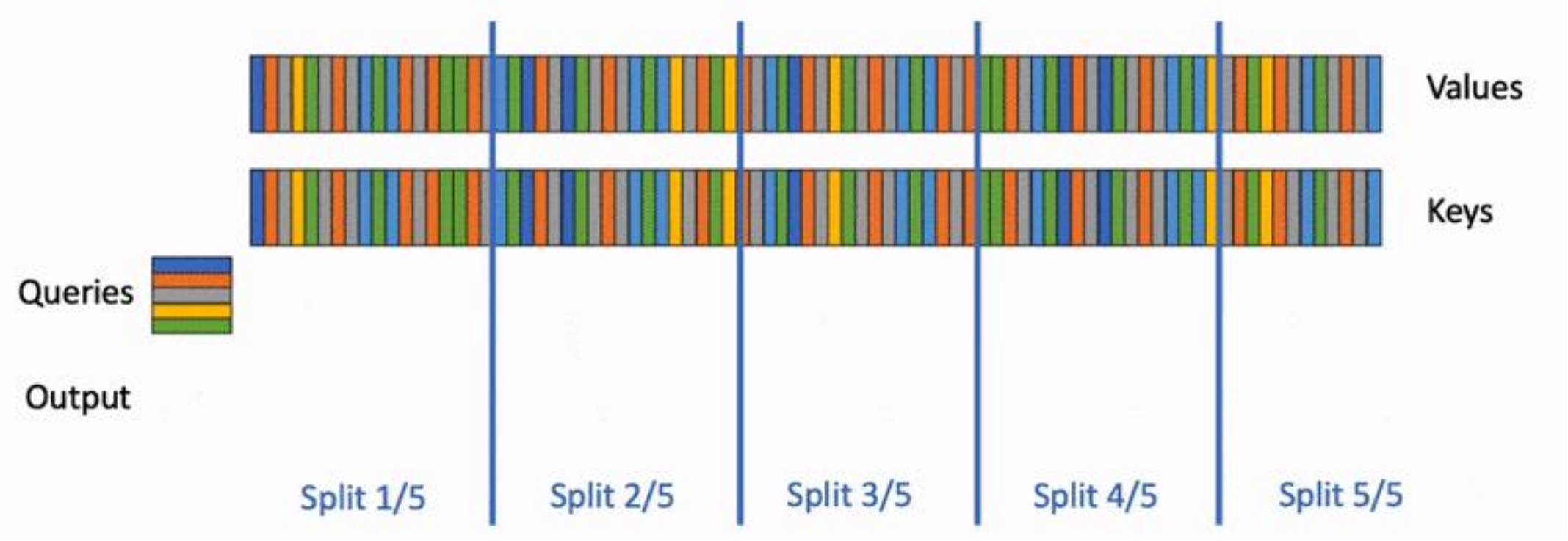
FlashAttention Processes K/V Sequentially



Inefficient for requests with long context (many keys/values)

Flash-Decoding Parallelizes Across Keys/Values

1. Split keys/values into small chunks
2. Compute attention with these splits using FlashAttention
3. Reduce overall all splits



Key insight: attention is associative and commutative

Flash-Decoding is up to 8x faster than prior work