Attention Optimizations

Chien-Yu Lin

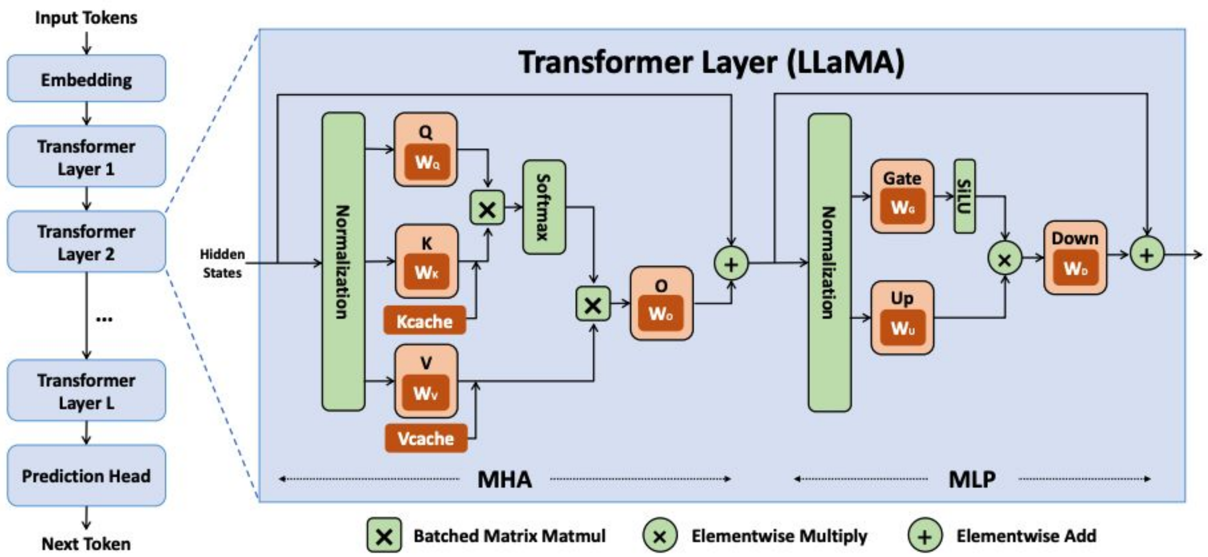
Zihao Ye

Slides are largely contributed by

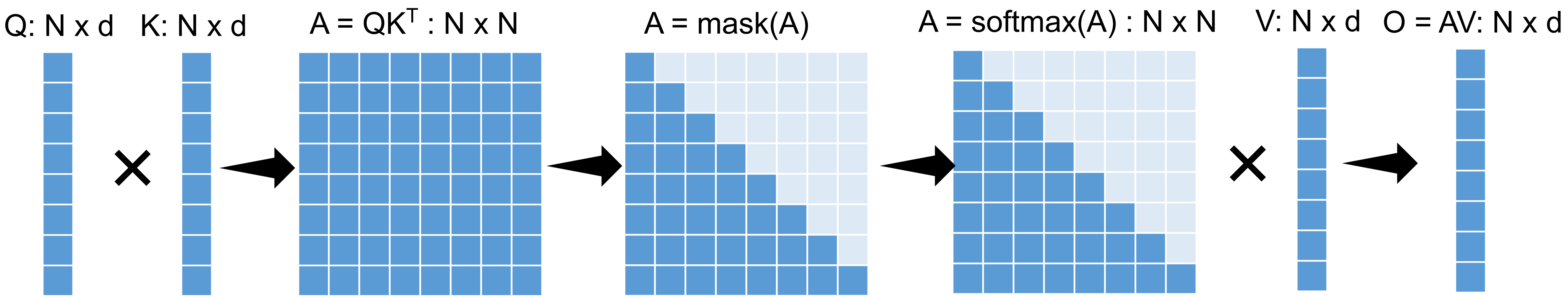
Tianqi Chen and Zhihao Jia from CMU

Why Optimizing Attention?

- Compute and memory complexity are quadratic to sequence length (N)
  - Compute = (4*d+3)*N^2
  - Loads/stores = 8*N^2 + 8*N*d
- Dominant runtime when sequence is long
  - For both training and inference



Attention: $O = \text{Softmax}(QK^T)\ V$



Challenges:

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

Outline: Attention Optimizations

Part 1: LLM Training

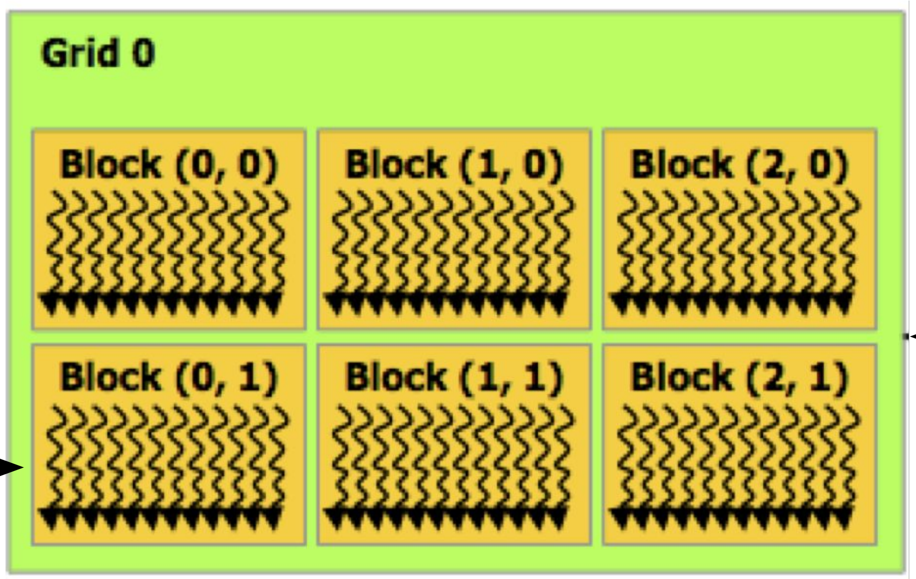
- FlashAttention

Part 2: LLM Inference

- Flash Decoding
- FlashInfer (Zihao)

These techniques are highly tailored for GPUs

GPU Memory Hierarchy



Per-block **shared memory** (readable/writable by all threads in a block)

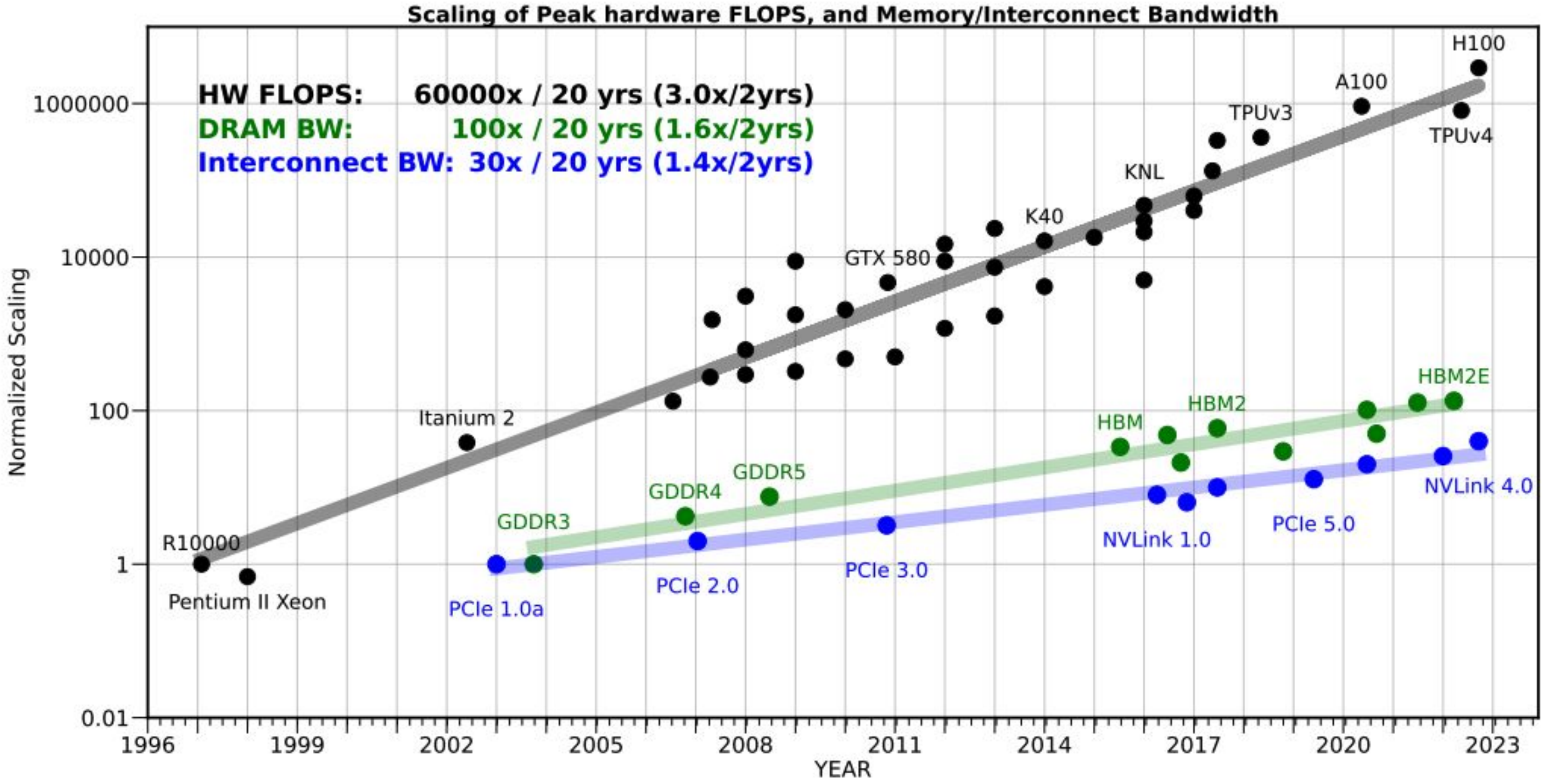
19 TB/s (20 MB)

Device global memory (readable/writable by all threads)

1.5 TB/s (80 GB)

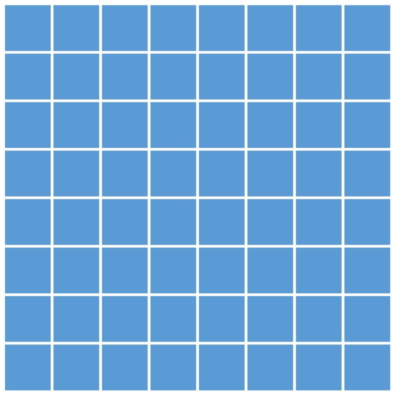
GPU FLOPs and Memory Bandwidth Trend

- Growth of compute outpace memory bandwidth



FlashAttention

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



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

Two main Techniques:

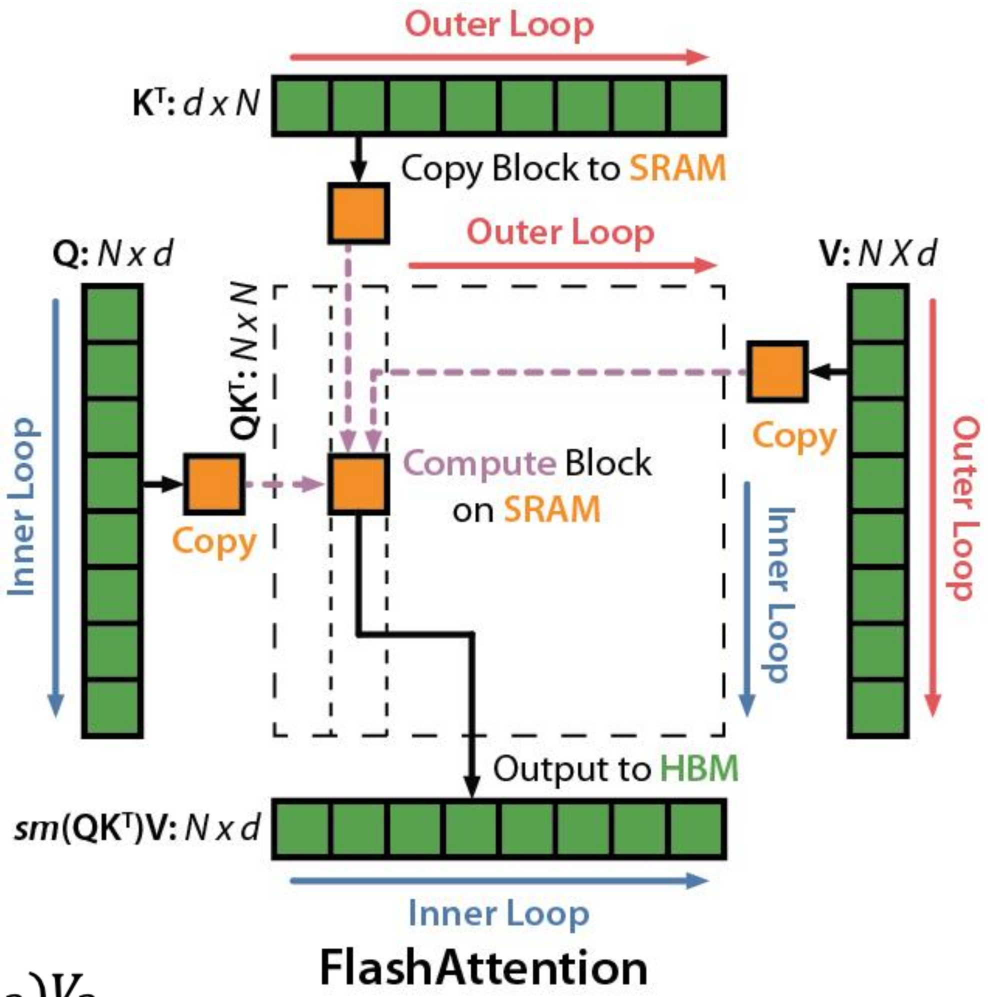
1. **Tiling:** restructure algorithm to load query/key/value block by block from global to shared memory

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

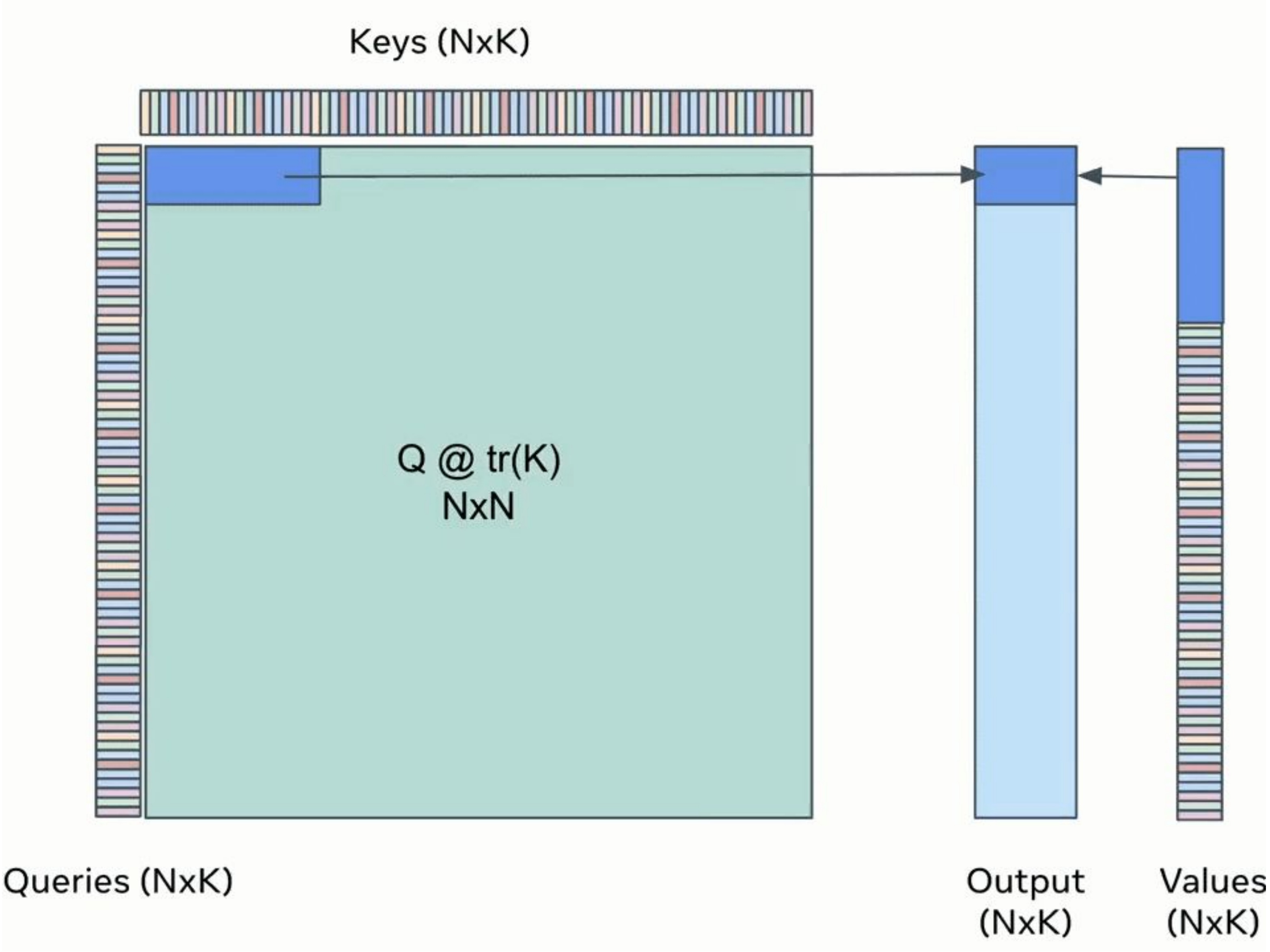


$$softmax([A_1, A_2]) = [\alpha \times softmax(A_1), \beta \times softmax(A_2)]$$

$$softmax([A_1, A_2]) \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \alpha \times softmax(A_1)V_1 + \beta \times softmax(A_2)V_2$$

https://courses.cs.washington.edu/courses/cse599m/23sp/notes/flashattn.pdf

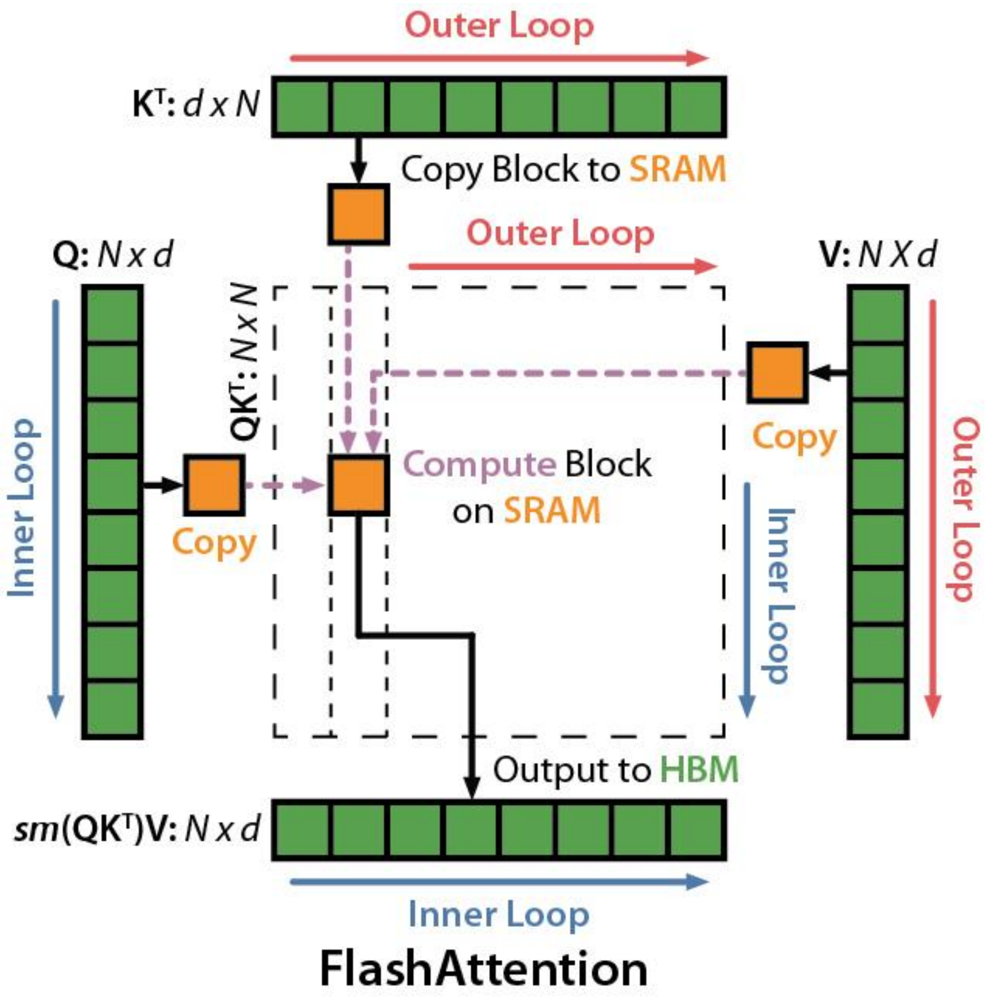
Tiling



Animation credit: Francisco Massa

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

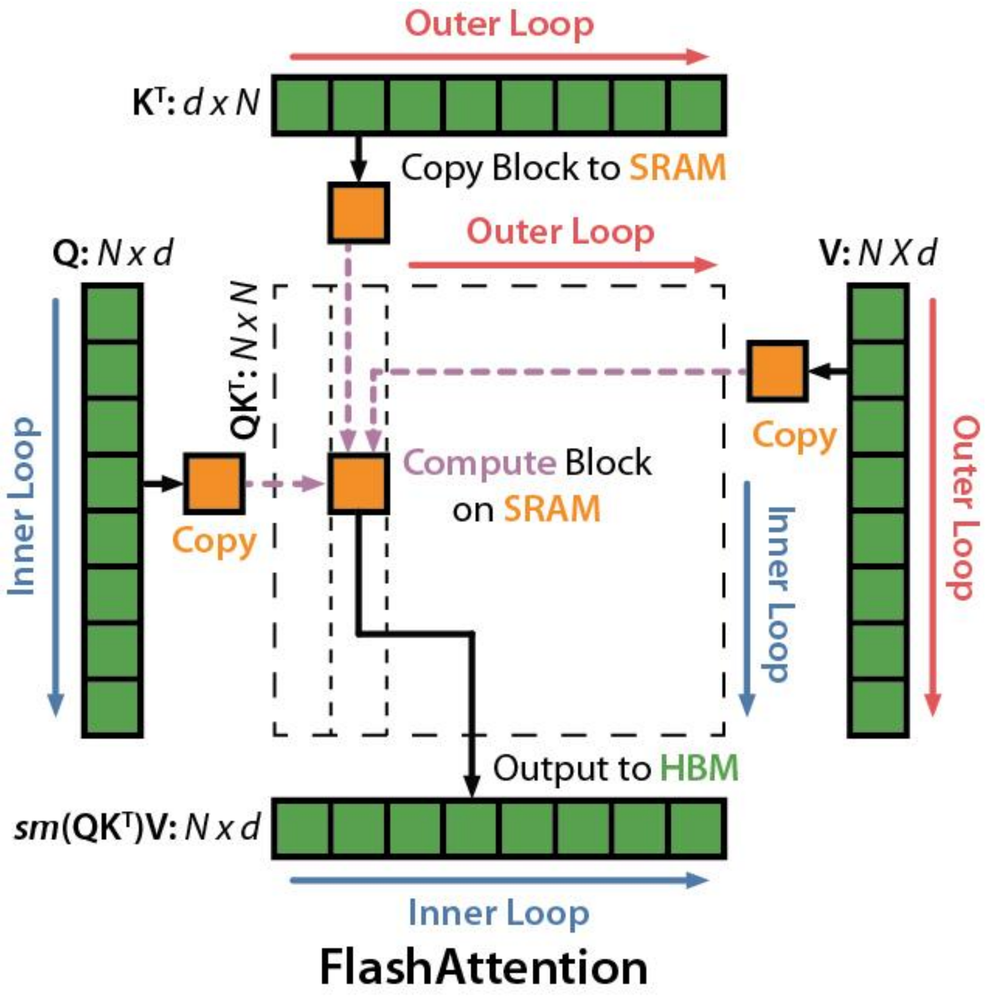
Speed up backward pass with increased FLOPs

FlashAttention: Threadblock-level Parallelism

How to partition FlasshAttention across thread blocks?

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

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

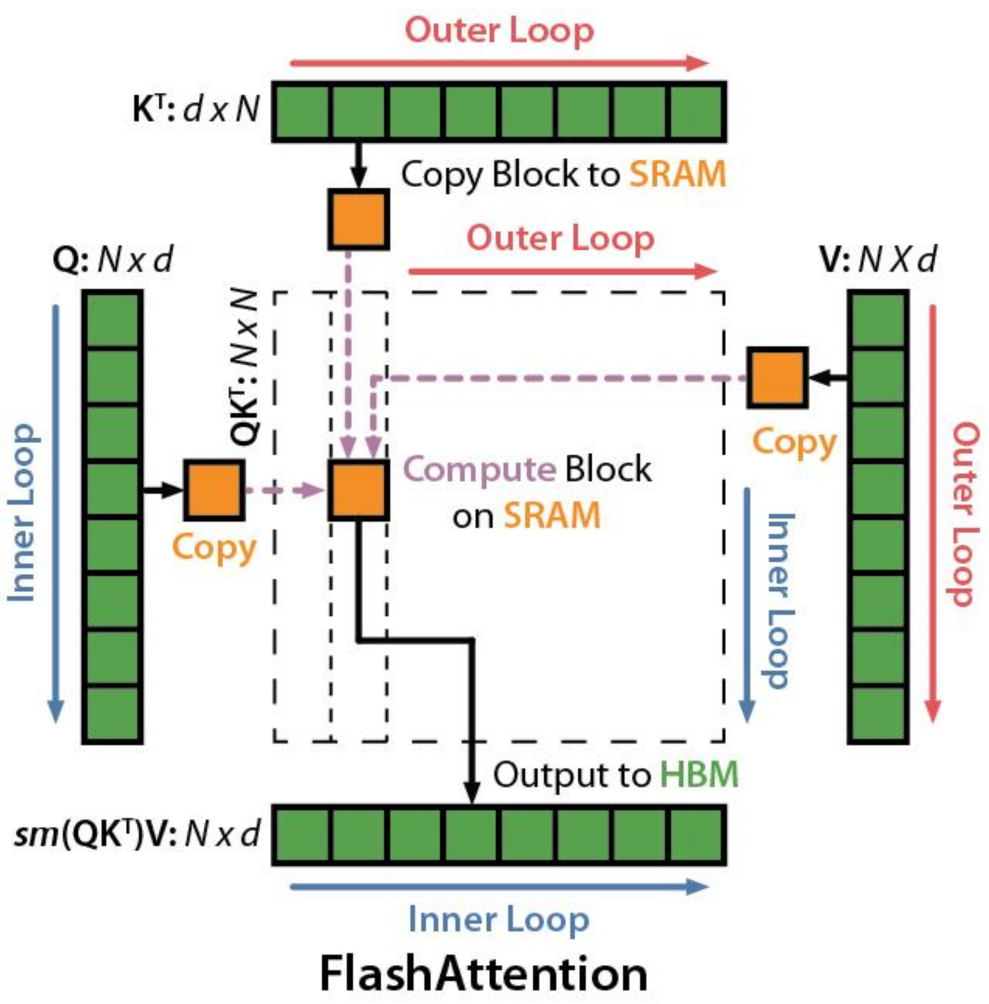


FlashAttention: Threadblock-level Parallelism

How to partition FlasshAttention across thread blocks?

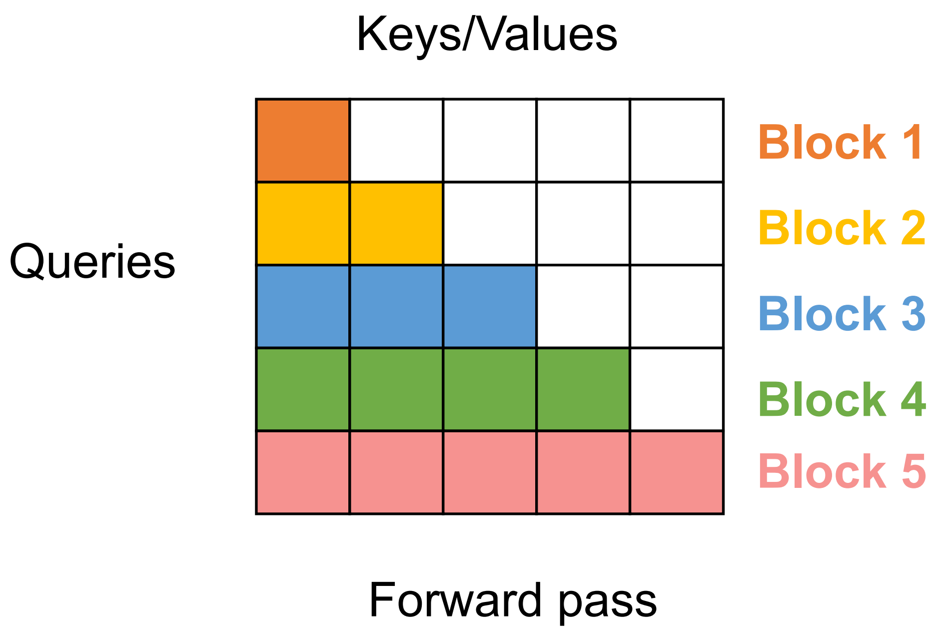
(An A100 has 108 SMs -> 108 thread blocks)

- 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

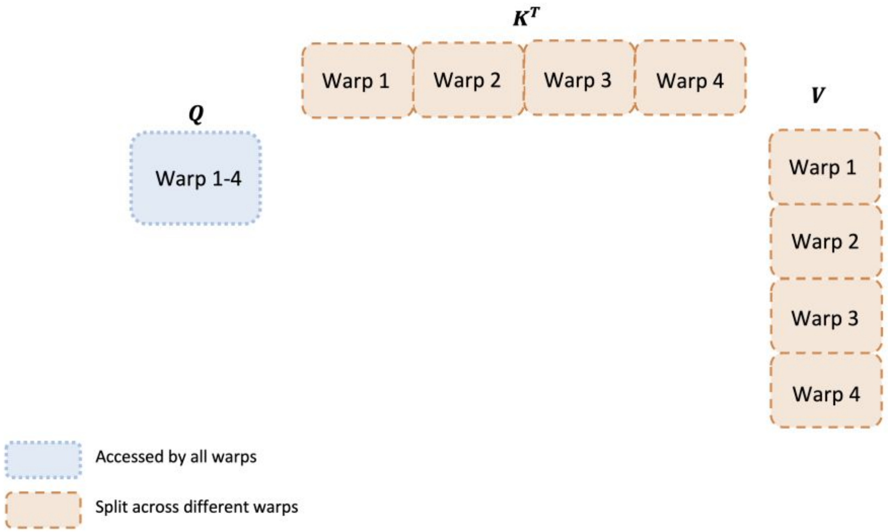


Do we need to handle workload imbalance?

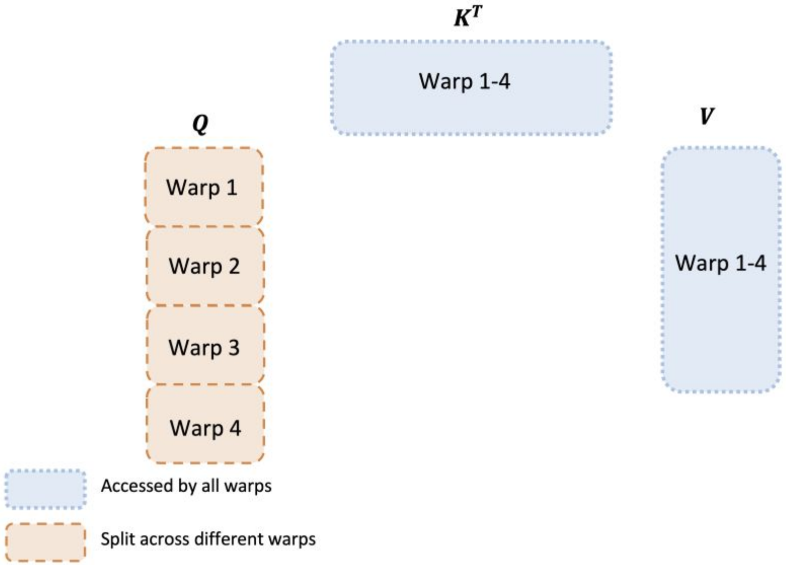
No. GPU scheduler automatically loads the next block once the current one completes.

FlashAttention: Warp-Level Parallelism

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



(a) FlashAttention

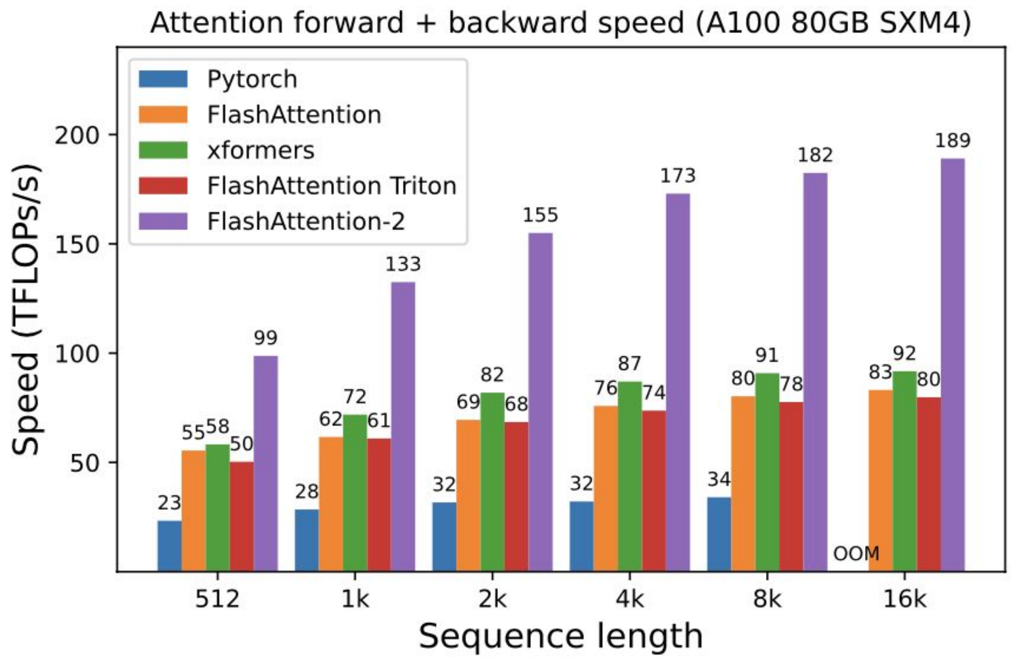


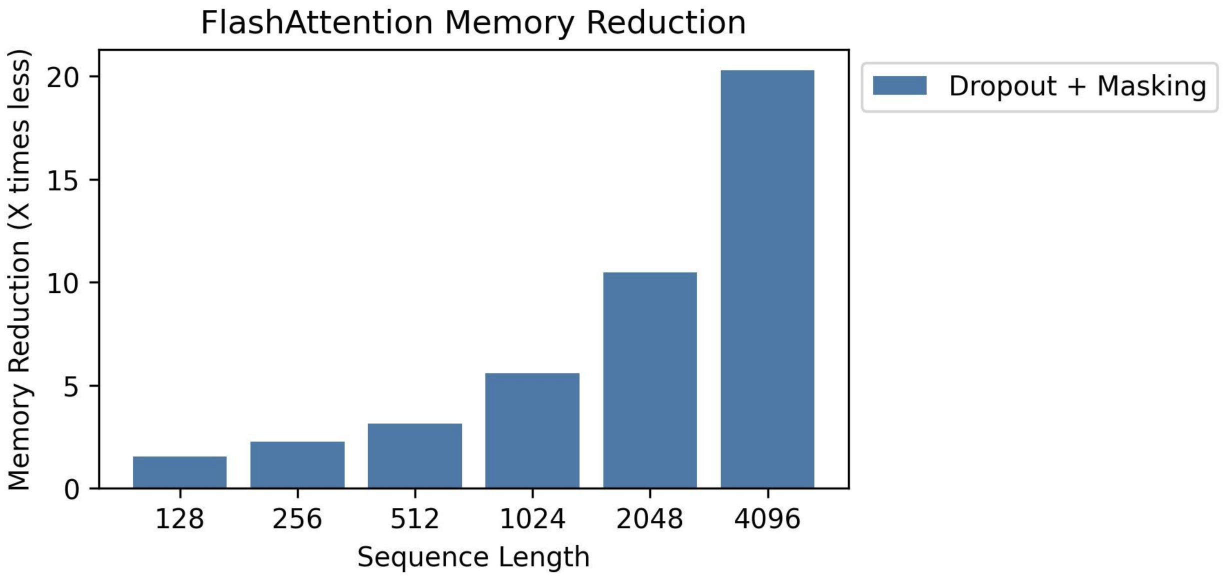
(b) FlashAttention-2

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

Outline: Attention Optimizastions

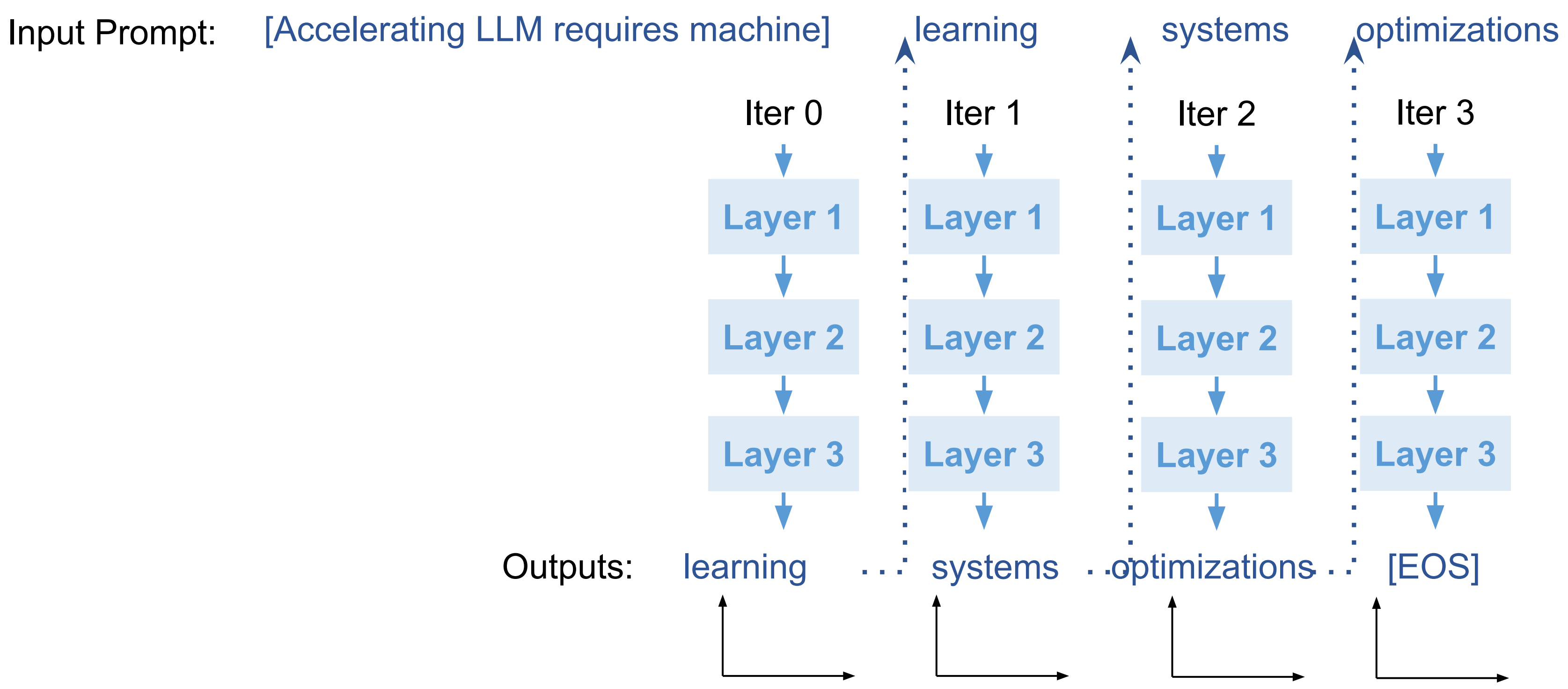
Part 1: LLM Training

- FlashAttention

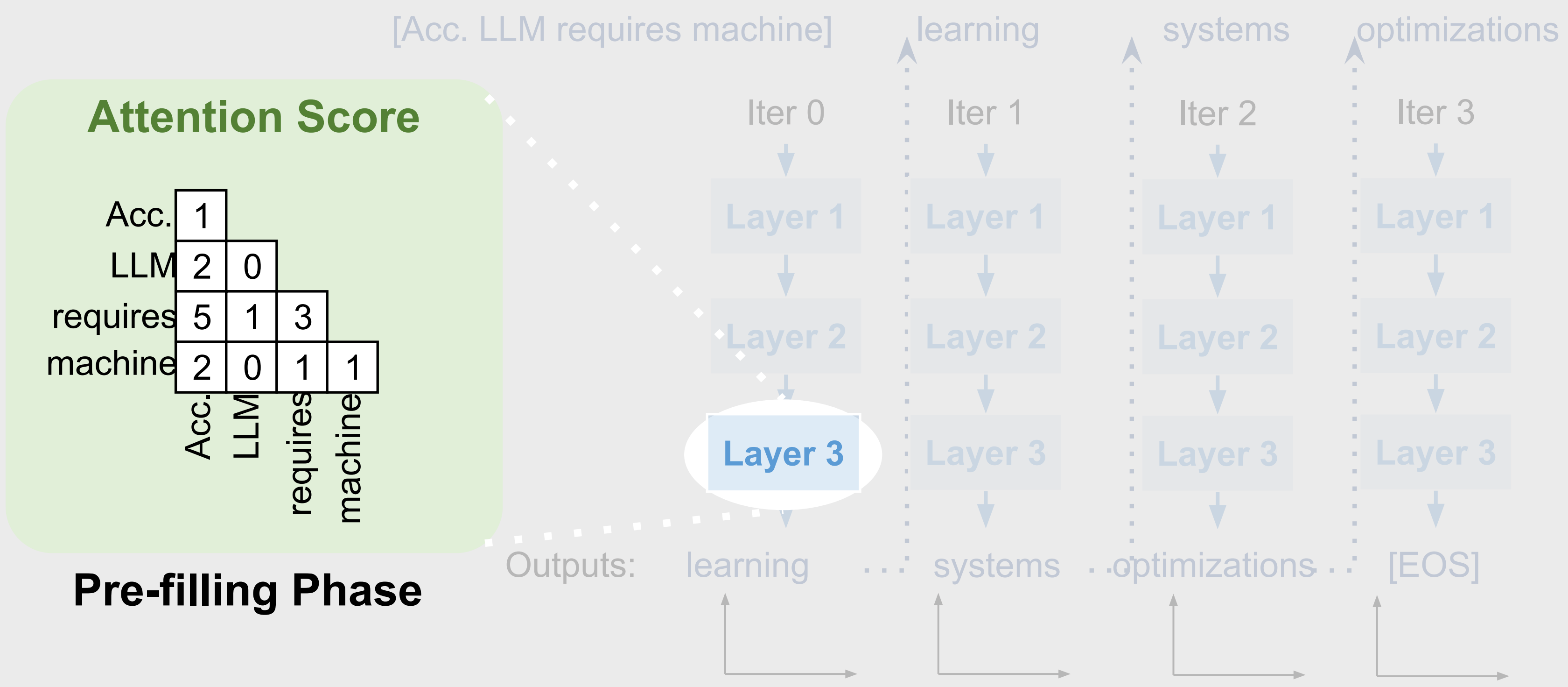
Part 2: LLM Inference (Auto-regressive Decoding)

- Flash Decoding
- FlashInfer (Zihao)

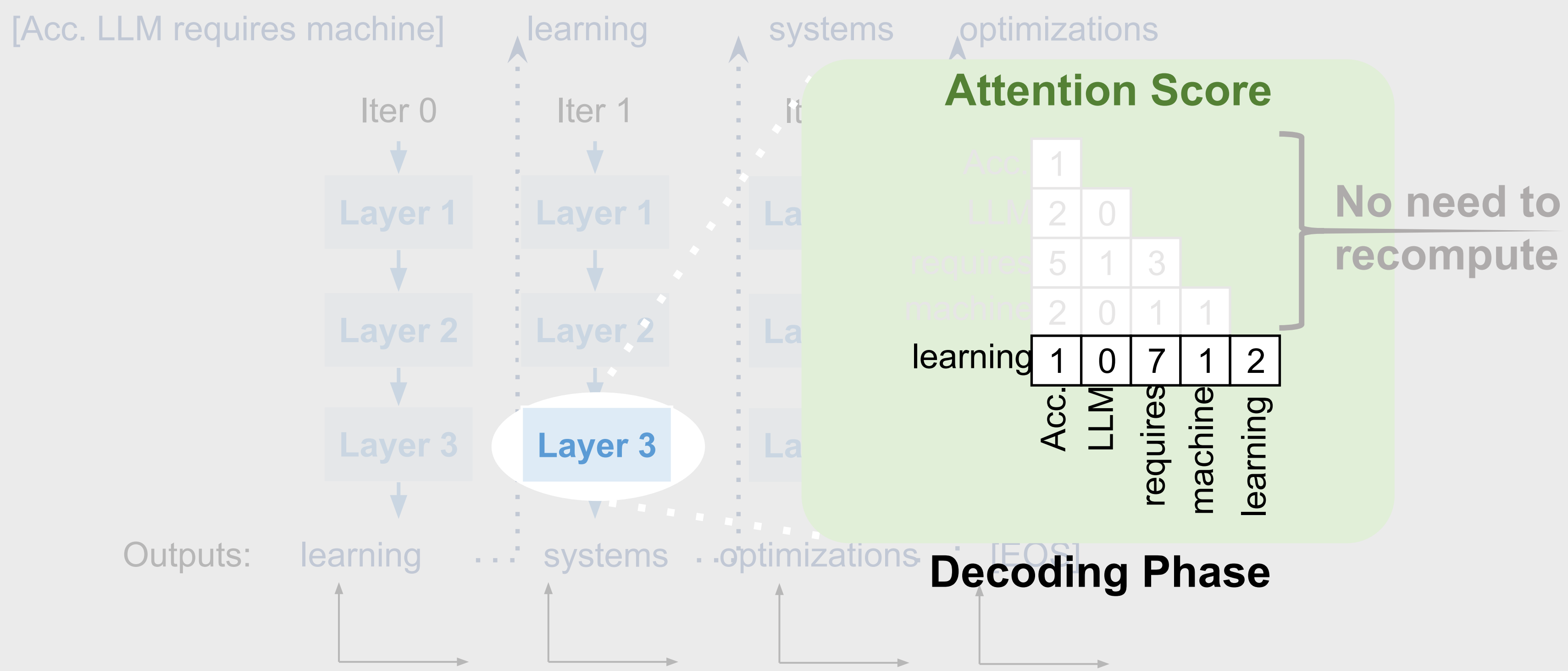
Generative LLM Inference: Autoregressive Decoding



Generative LLM Inference: Autoregressive Decoding



Generative LLM Inference: Autoregressive Decoding

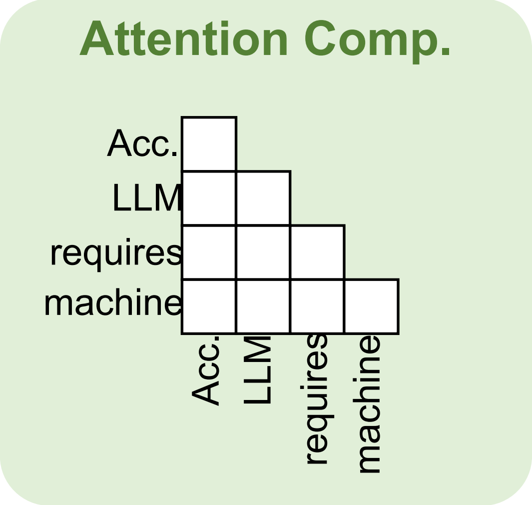


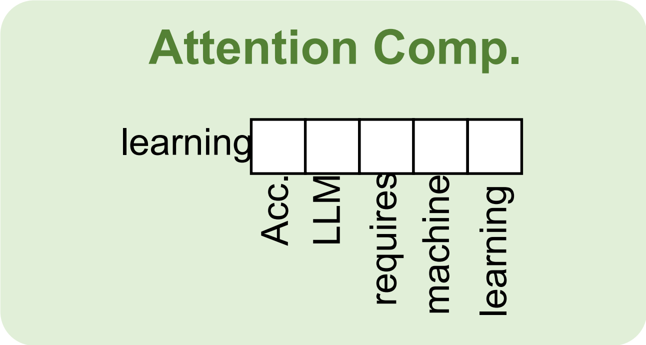
Generative LLM Inference: Autoregressive Decoding

- **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:

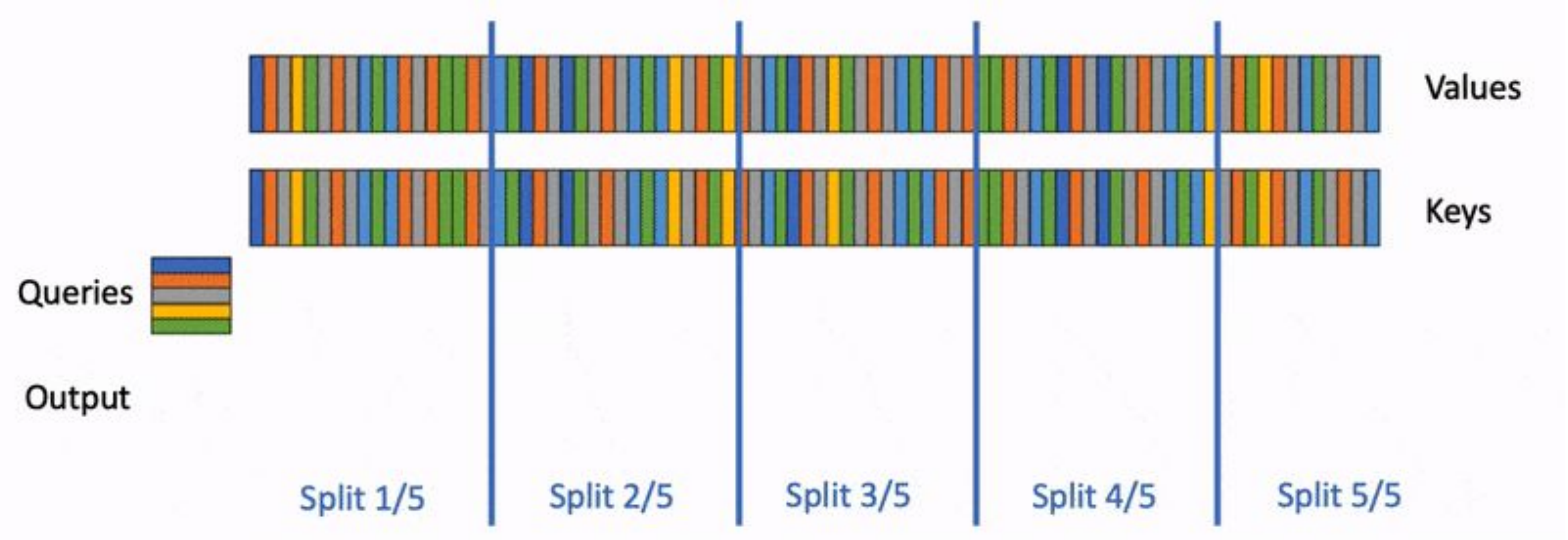
- No, there is a single query in the decoding phase

FlashAttention Processes K/V Sequentially

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

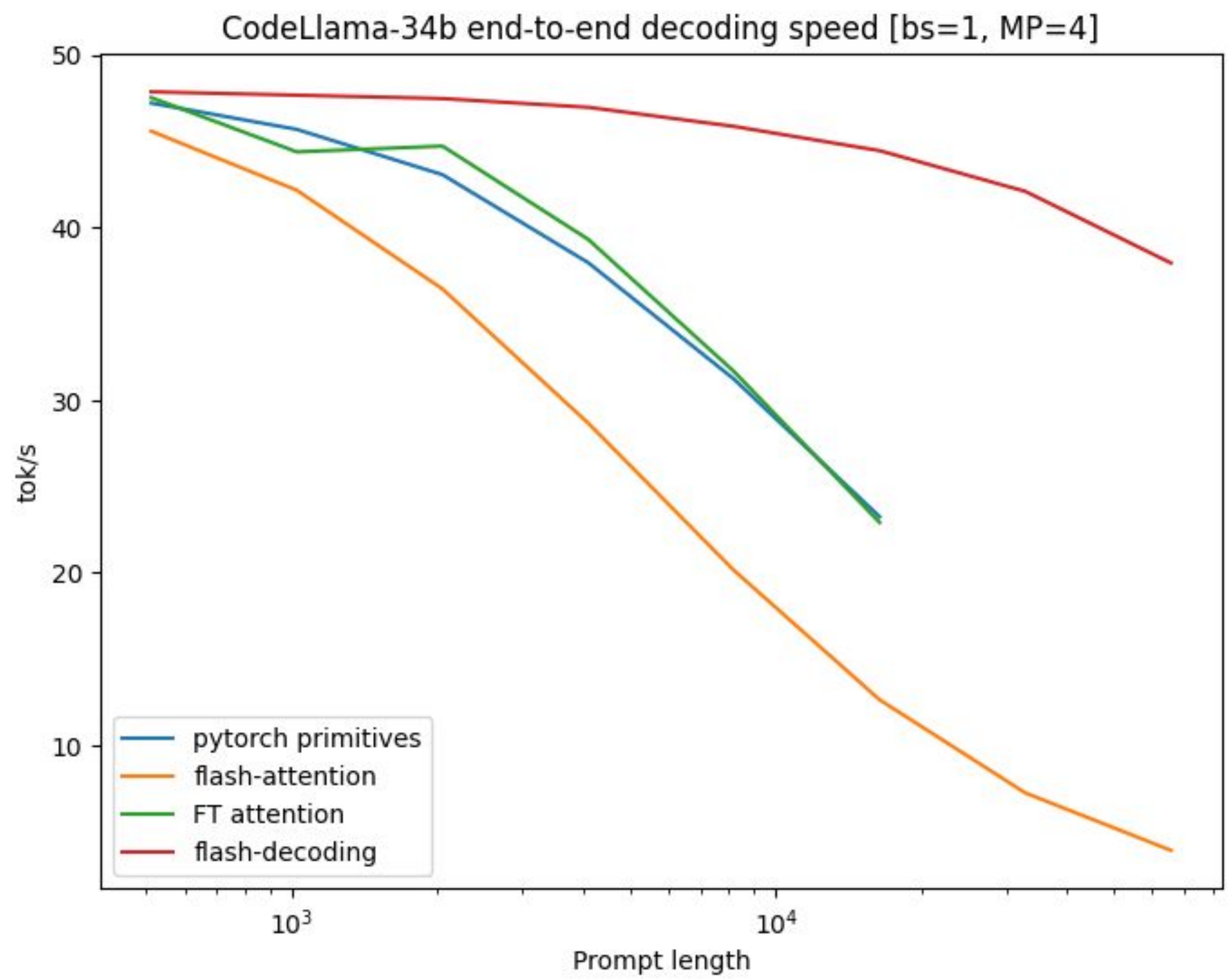
Flash-Decoding Parallelizes Across Keys/Values

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



Key insight: attention is associative and commutative

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



Advanced Attention Optimizations

FlashInfer

https://github.com/flashinfer-ai/flashinfer

Recap: Techniques for Optimizing Attention

- **FlashAttention**: tiling to reduce GPU global memory access
- **Auto-regressive Decoding**: pre-filling and decoding phases, KV cache
- **FlashDecoding**: improving attention's parallelism by splitting keys/values
- **PagedAttention**: paging and virtualization to reduce KV cache's memory requirement