https://hao-ai-lab.github.io/dsc291-s24/

DSC 291: ML Systems
Spring 2024

LLMs

Parallelization

Single-device Optimization

Basics

Logistics

- Please start preparing your final project talk
  - We will use week 10 (may need additional time) to go through each team's talk
  - The talk orders have been out
  - Please upload your slides to TAs by next Tuesday
  - TAs has distributed some rubrics/guidelines

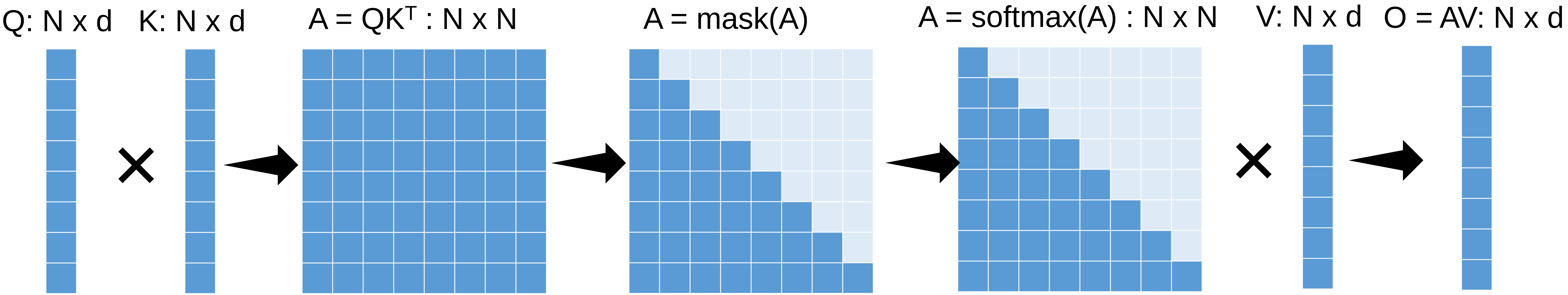
Course Evaluation

- Course evaluation is sent out
  - May 27 at 12:00 AM and Saturday, June 8
- Please fill the course evaluation
  - It is important for you:
    - Get your 2% if 80% of you filled the survey
  - It is important for TAs!
  - It is important for me!

Where We Are: LLMs

- Transformers and Attentions
- LLM Training Optimizations
  - **Flash attention**
  - 3D parallelism
- LLM Inference and Serving
  - Paged attention
  - Continuous batching
  - Speculative decoding
- Scaling Laws
- Long context

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



Attention Computation

Algorithm 0 Standard Attention Implementation

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

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.

2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.

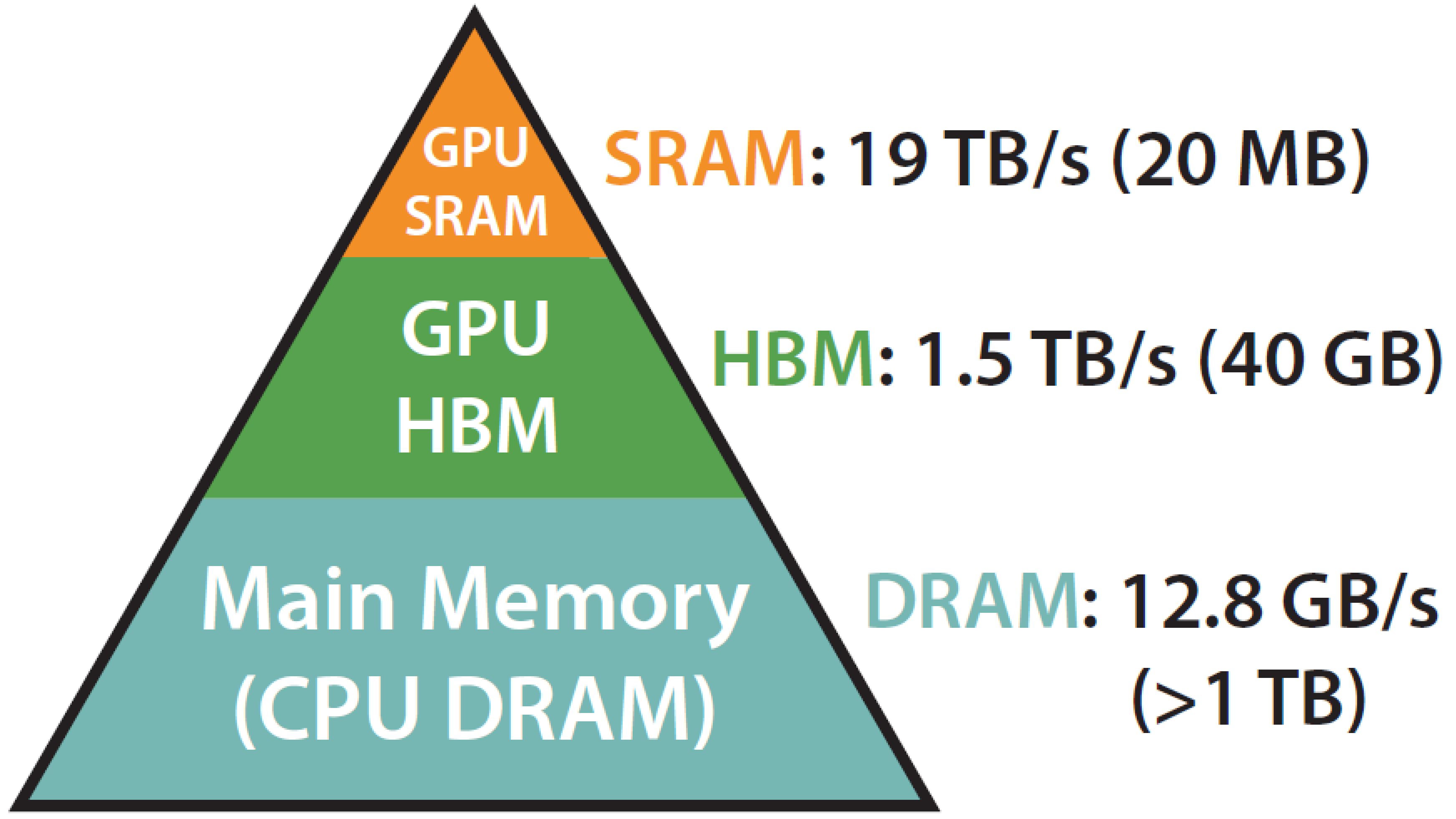
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.

4: Return $\mathbf{O}$.

Challenges:

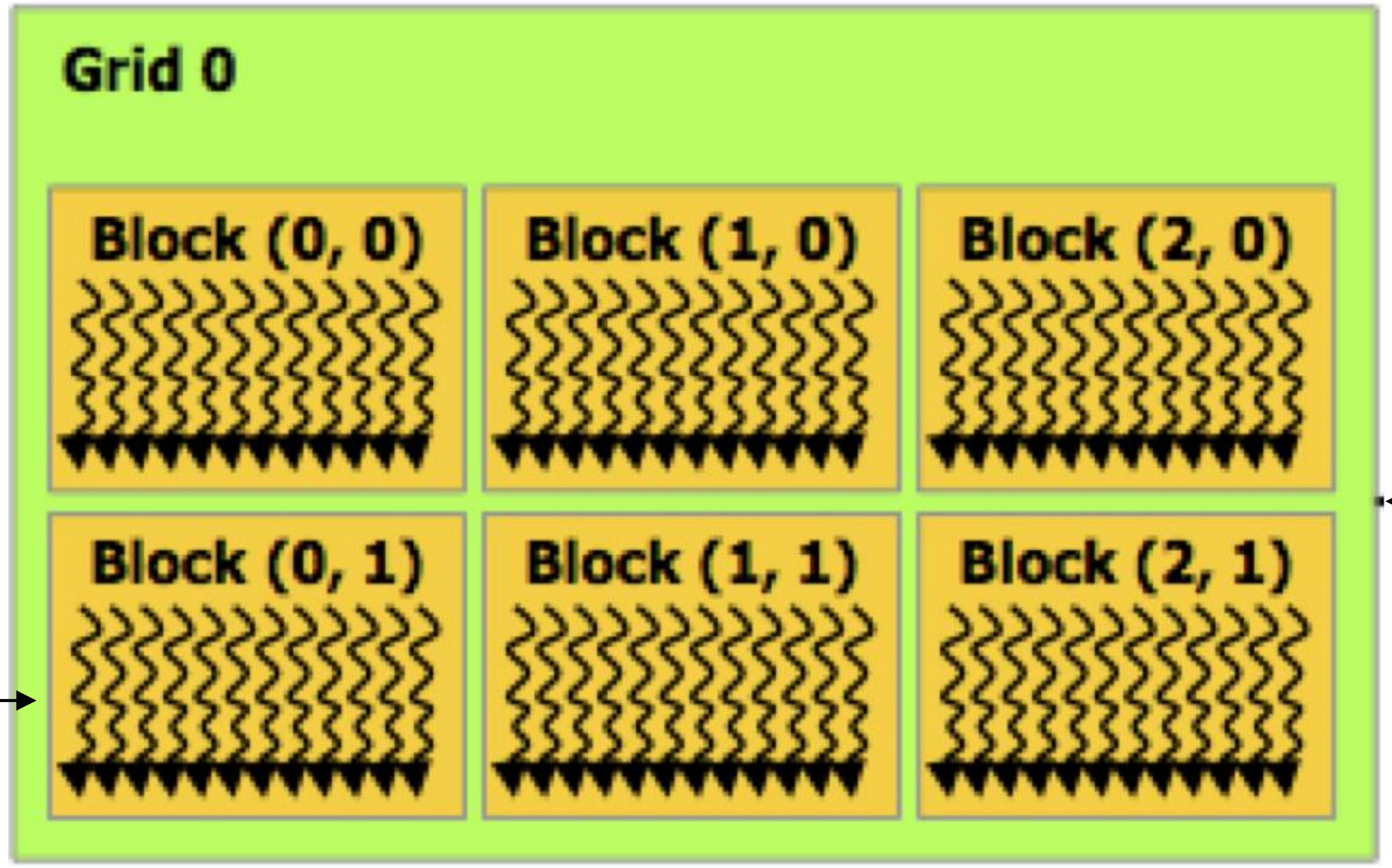
- Large intermediate results
- Repeated reads/writes from GPU device memory
- Cannot scale to long sequences due to O(N^2) intermediate results

Revisit: GPU Memory Hierarchy



Memory Hierarchy with
Bandwidth & Memory Size

Revisit: GPU Memory Hierarchy



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

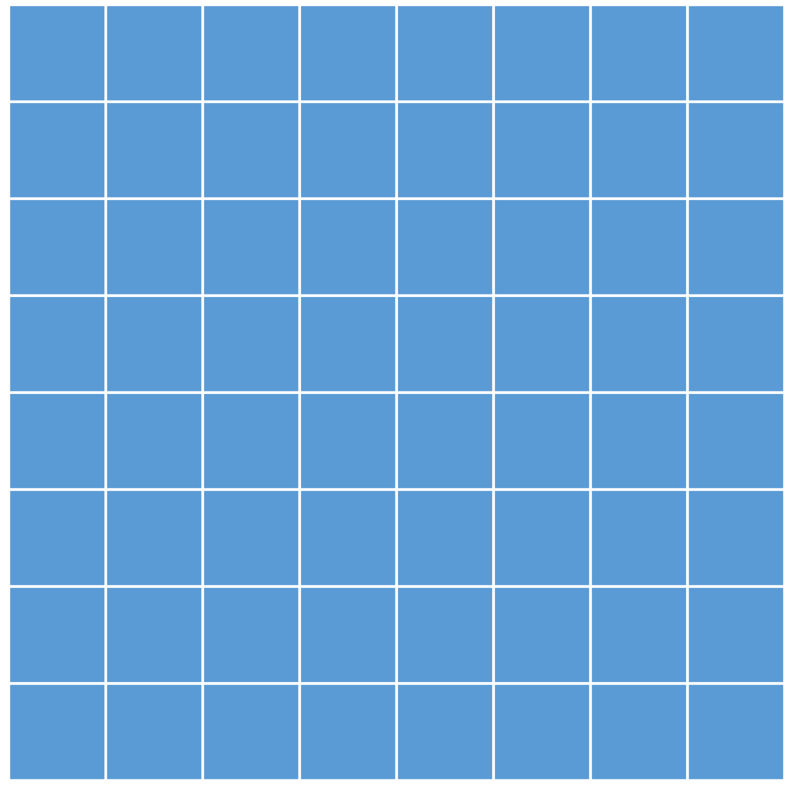
19 TB/s (20 MB)

Device global memory (readable/writable by all threads)

1.5 TB/s (80 GB)

FlashAttention

A = softmax(QK^T)



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

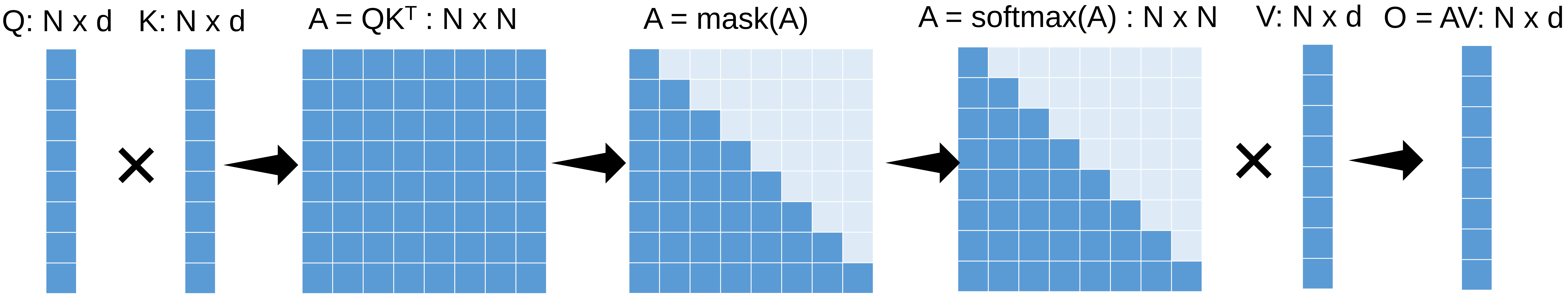
Two main Techniques:

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

Problem: How to tile softmax?



Challenges

- We must avoid materializing NxN while still get the precise softmax results
    - Compute softmax reduction w/o access to NxN at forward
- Backward without the NxN softmax forward activations

How to Implement Softmax

Algorithm 1 Naive softmax

1: $d_0 \leftarrow 0$
2: **for** $j \leftarrow 1, V$ **do**
3: $\quad d_j \leftarrow d_{j-1} + e^{x_j}$
4: **end for**
5: **for** $i \leftarrow 1, V$ **do**
6: $\quad y_i \leftarrow \frac{e^{x_i}}{d_V}$
7: **end for**

Problem

- Can easily go overflow because of sum (e^x)

Safe Softmax

$$y_i = \frac{e^{x_i - \max_{k=1}^{V} x_k}}{\sum_{j=1}^{V} e^{x_j - \max_{k=1}^{V} x_k}}$$

Algorithm 2 Safe softmax

```
m_0 ← -∞
for k ← 1, V do
    m_k ← max(m_{k-1}, x_k)
end for
d_0 ← 0
for j ← 1, V do
    d_j ← d_{j-1} + e^{x_j - m_V}
end for
for i ← 1, V do
    y_i ← e^{x_i - m_V} / d_V
end for
```

Can we fuse?

Create alternative sequence

$$d_i' := \sum_{j=1}^{i} e^{x_j - m_i}$$

With:

$$d_V' = d_V$$

Algorithm 2 Safe softmax

$m_0 \leftarrow -\infty$
for $k \leftarrow 1, V$ **do**
    $m_k \leftarrow \max(m_{k-1}, x_k)$
end for
$d_0 \leftarrow 0$
for $j \leftarrow 1, V$ **do**
    $d_j \leftarrow d_{j-1} + e^{x_j - m_V}$
end for
for $i \leftarrow 1, V$ **do**
    $y_i \leftarrow \frac{e^{x_i - m_V}}{d_V}$
end for

But:

Create alternative sequence

$$d'_i := \sum_{j=1}^{i} e^{x_j - m_i}$$

With:
$d'_V = d_V$

$$\begin{aligned} d'_i &= \sum_{j=1}^{i} e^{x_j - m_i} \\ &= \left(\sum_{j=1}^{i-1} e^{x_j - m_i} \right) + e^{x_i - m_i} \\ &= \left(\sum_{j=1}^{i-1} e^{x_j - m_{i-1}} \right) e^{m_{i-1} - m_i} + e^{x_i - m_i} \\ &= d'_{i-1} \, e^{m_{i-1} - m_i} + e^{x_i - m_i} \end{aligned}$$

d_V does not depend on m_V

Online, Safe Softmax

Algorithm 3 Safe softmax with online normalizer calculation

$m_0 \leftarrow -\infty$
$d_0 \leftarrow 0$
for $j \leftarrow 1, V$ **do**
  $m_j \leftarrow \max\left(m_{j-1}, x_j\right)$
  $d_j \leftarrow d_{j-1} \times e^{m_{j-1} - m_j} + e^{x_j - m_j}$
end for
for $i \leftarrow 1, V$ **do**
  $y_i \leftarrow \frac{e^{x_i - m_V}}{d_V}$
end for

Self attention

NOTATIONS

$Q[k,:]$: the k-th row vector of Q matrix.

$K^T[:,i]$: the i-th column vector of K^T matrix.

$O[k,:]$: the k-th row of output O matrix.

$V[i,:]$: the i-th row of V matrix.

$\{\boldsymbol{o}_i\}$: $\sum_{j=1}^{i} a_j V[j,:]$, a row vector storing partial aggregation result $A[k,:i] \times V[:i,:]$

BODY

for $i \leftarrow 1, N$ **do**

$$\begin{aligned} x_i &\leftarrow Q[k,:]\, K^T[:,i] \\ m_i &\leftarrow \max(m_{i-1}, x_i) \\ d'_i &\leftarrow d'_{i-1}\, e^{m_{i-1}-m_i} + e^{x_i - m_i} \end{aligned}$$

end

for $i \leftarrow 1, N$ **do**

$$\begin{aligned} a_i &\leftarrow \frac{e^{x_i - m_N}}{d'_N} \\ \boldsymbol{o}_i &\leftarrow \boldsymbol{o}_{i-1} + a_i\, V[i,:] \end{aligned}$$

end

$$O[k,:] \leftarrow \boldsymbol{o}_N$$

Self attention

NOTATIONS
$Q[k,:]$: the k-th row vector of Q matrix.
$K^T[:,i]$: the i-th column vector of K^T matrix.
$O[k,:]$: the k-th row of output O matrix.
$V[i,:]$: the i-th row of V matrix.
$\{\boldsymbol{o}_i\}$: $\sum_{j=1}^{i} a_j V[j,:]$, a row vector storing partial aggregation result $A[k,:i] \times V[:i,:]$

BODY
for $i \leftarrow 1, N$ **do**

$$x_i \leftarrow Q[k,:]\, K^T[:,i]$$
$$m_i \leftarrow \max(m_{i-1}, x_i)$$
$$d'_i \leftarrow d'_{i-1}\, e^{m_{i-1}-m_i} + e^{x_i - m_i}$$

end
for $i \leftarrow 1, N$ **do**

$$a_i \leftarrow \frac{e^{x_i - m_N}}{d'_N}$$
$$\boldsymbol{o}_i \leftarrow \boldsymbol{o}_{i-1} + a_i V[i,:]$$

end

$$O[k,:] \leftarrow \boldsymbol{o}_N$$

$$\boldsymbol{o}_i := \sum_{j=1}^{i} \left(\frac{e^{x_j - m_N}}{d'_N} V[j,:] \right)$$

Create alternative sequence with $o_N = o'_N$

$$\boldsymbol{o}'_i := \left(\sum_{j=1}^{i} \frac{e^{x_j - m_i}}{d'_i} V[j,:] \right)$$

But

$$
\begin{aligned}
\boldsymbol{o}'_i &= \sum_{j=1}^{i} \frac{e^{x_j - m_i}}{d'_i} V[j,:] \\
&= \left(\sum_{j=1}^{i-1} \frac{e^{x_j - m_i}}{d'_i} V[j,:] \right) + \frac{e^{x_i - m_i}}{d'_i} V[i,:] \\
&= \left(\sum_{j=1}^{i-1} \frac{e^{x_j - m_{i-1}}}{d'_{i-1}} \frac{e^{x_j - m_i}}{e^{x_j - m_{i-1}}} \frac{d'_{i-1}}{d'_i} V[j,:] \right) + \frac{e^{x_i - m_i}}{d'_i} V[i,:] \\
&= \left(\sum_{j=1}^{i-1} \frac{e^{x_j - m_{i-1}}}{d'_{i-1}} V[j,:] \right) \frac{d'_{i-1}}{d'_i} e^{m_{i-1} - m_i} + \frac{e^{x_i - m_i}}{d'_i} V[i,:] \\
&= \boldsymbol{o}'_{i-1} \frac{d'_{i-1} e^{m_{i-1} - m_i}}{d'_i} + \frac{e^{x_i - m_i}}{d'_i} V[i,:]
\end{aligned}
$$

for $i \leftarrow 1, N$ **do**

$$
\begin{aligned}
x_i &\leftarrow Q[k,:]\, K^T[:, i] \\
m_i &\leftarrow \max\left(m_{i-1}, x_i\right) \\
d'_i &\leftarrow d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i} \\
\boldsymbol{o}'_i &\leftarrow \boldsymbol{o}'_{i-1} \frac{d'_{i-1} e^{m_{i-1} - m_i}}{d'_i} + \frac{e^{x_i - m_i}}{d'_i} V[i,:]
\end{aligned}
$$

end

$$O[k,:] \leftarrow \boldsymbol{o}'_N$$

Flash Attention

for $i \leftarrow 1, N$ **do**

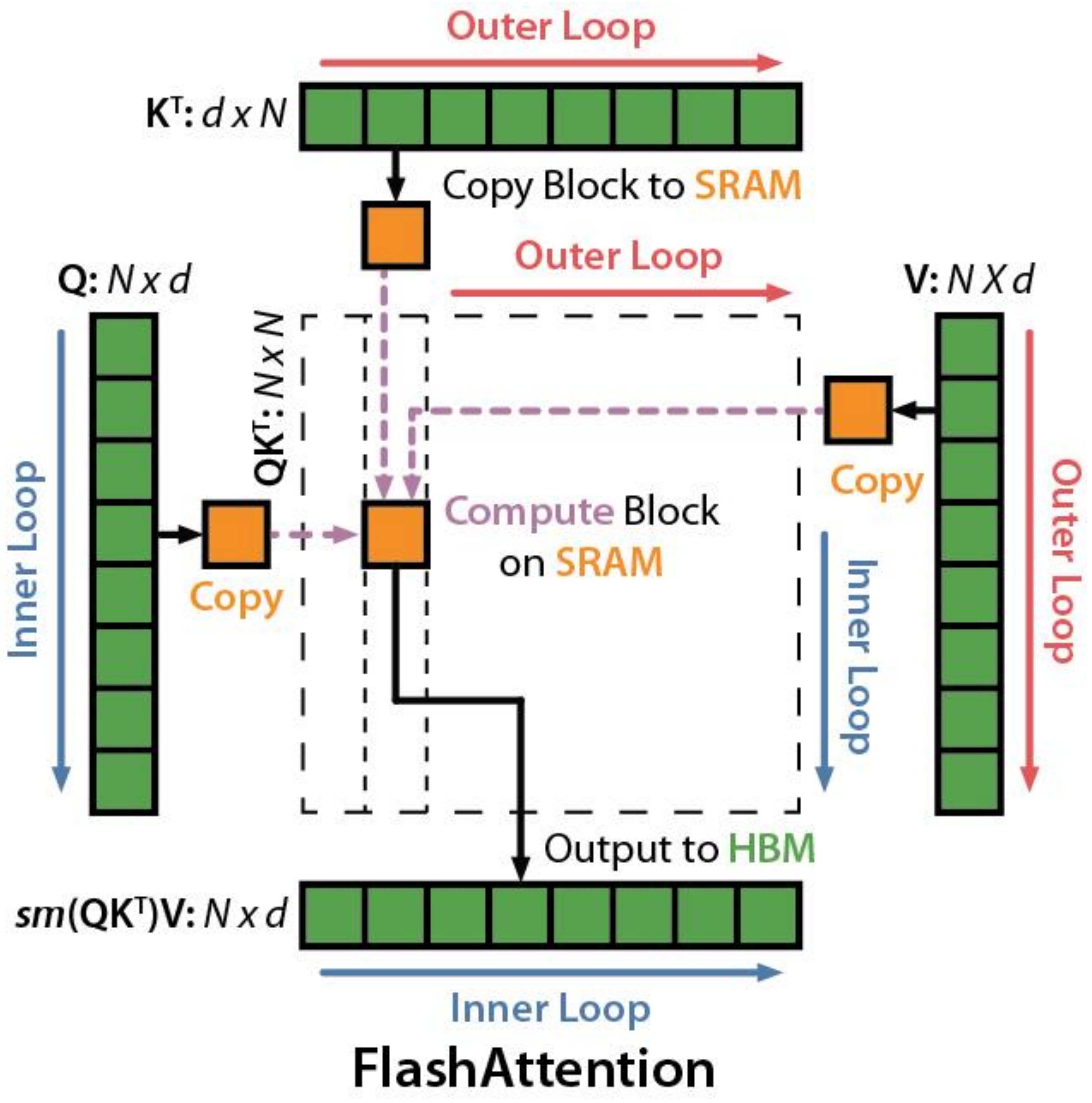
$$
\begin{aligned}
x_i &\leftarrow Q[k,:]\, K^T[:,i] \\
m_i &\leftarrow \max\left(m_{i-1}, x_i\right) \\
d_i' &\leftarrow d_{i-1}'\, e^{m_{i-1}-m_i} + e^{x_i - m_i} \\
\boldsymbol{o}_i' &\leftarrow \boldsymbol{o}_{i-1}' \frac{d_{i-1}'\, e^{m_{i-1}-m_i}}{d_i'} + \frac{e^{x_i-m_i}}{d_i'} V[i,:]
\end{aligned}
$$

end

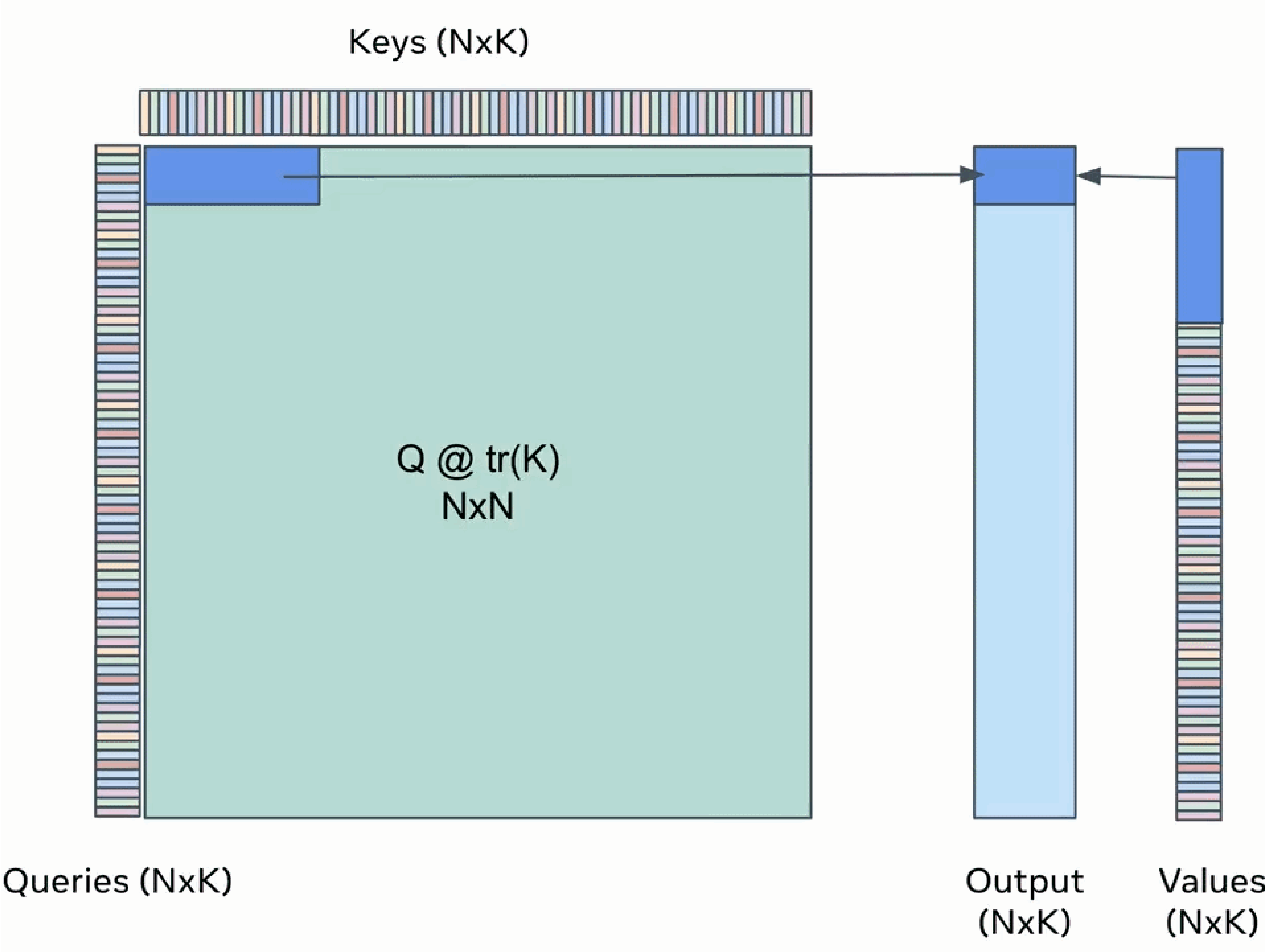
$$O[k,:] \leftarrow \boldsymbol{o}_N'$$

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



Tiling

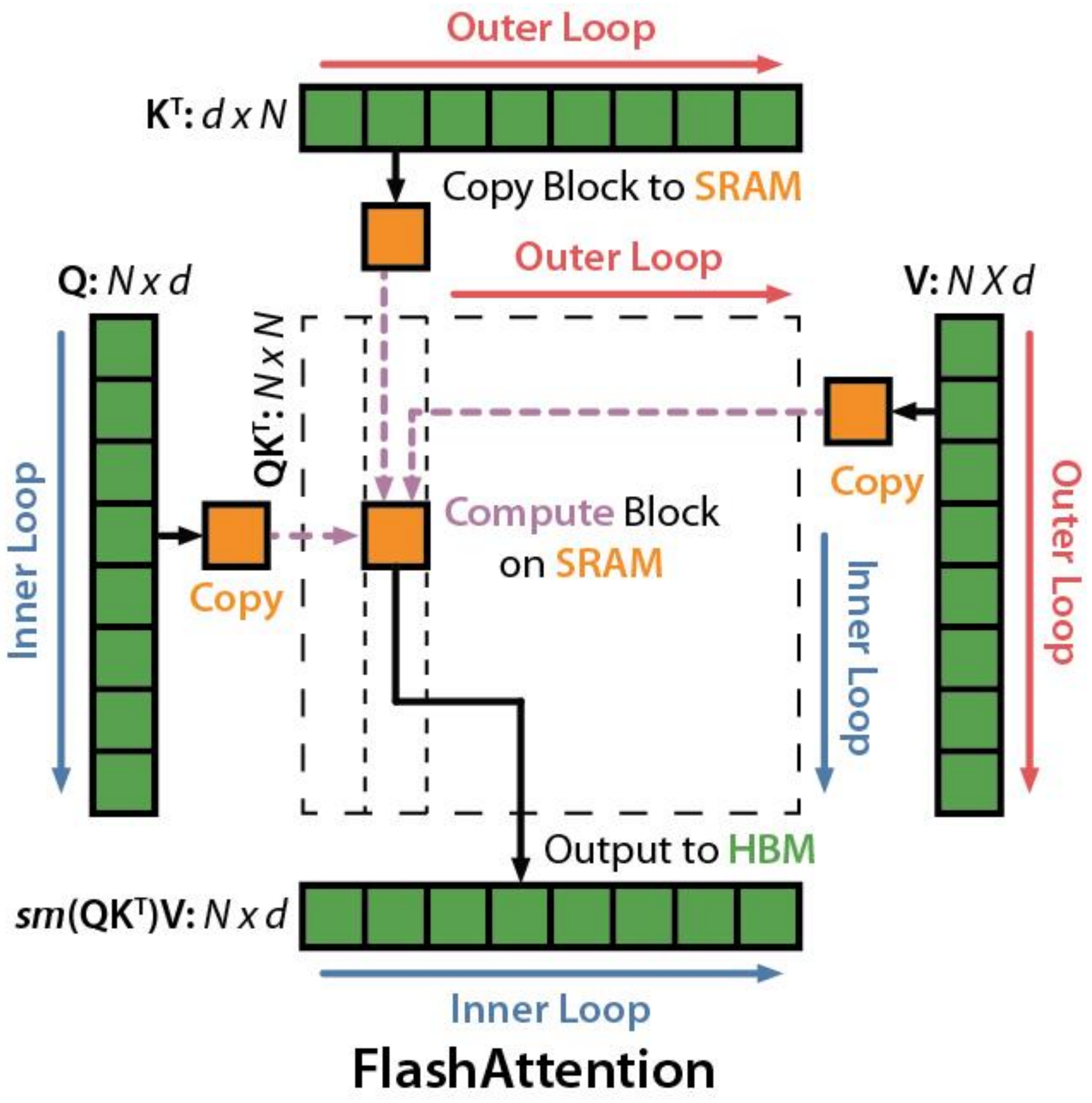


Animation credit: Francisco Massa

Recomputation: Backward Pass

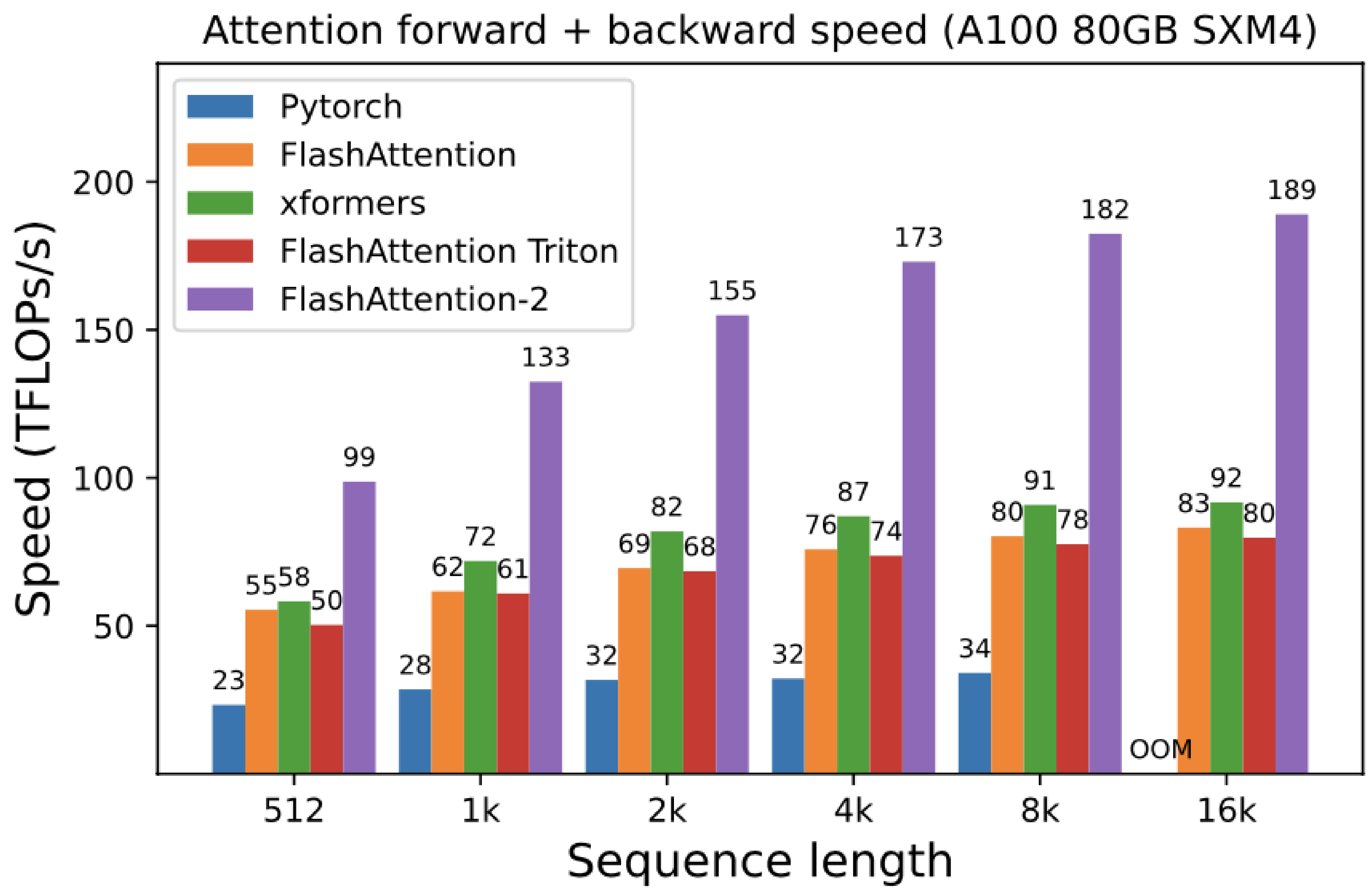
By storing softmax normalization factors from forward (size N), recompute attention in the backward from inputs in shared memory

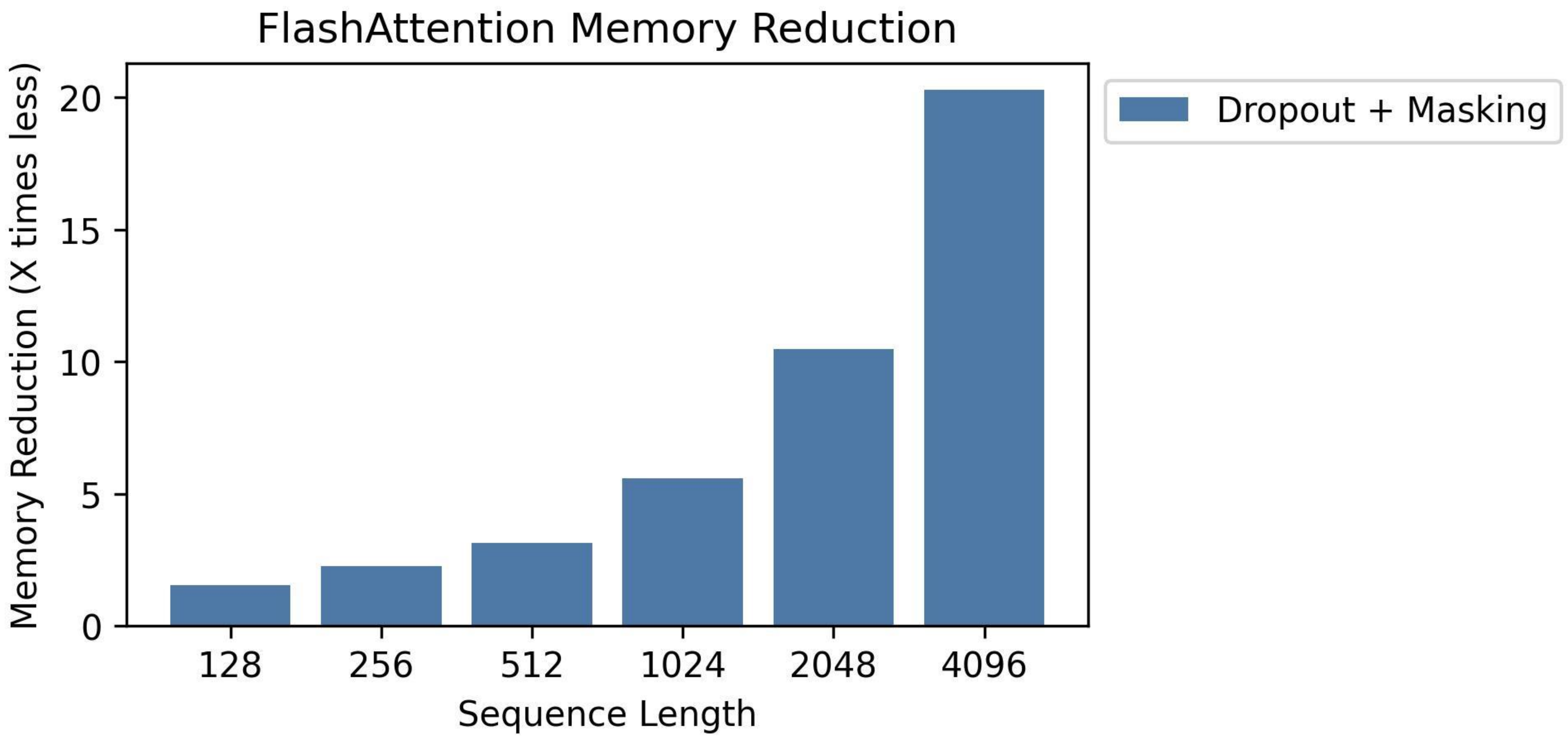
Attention	Standard	FlashAttention
GFLOPs	66.6	75.2
Global mem access	40.3 GB	4.4 GB
Runtime	41.7 ms	7.3 ms



Speed up backward pass with increased FLOPs

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



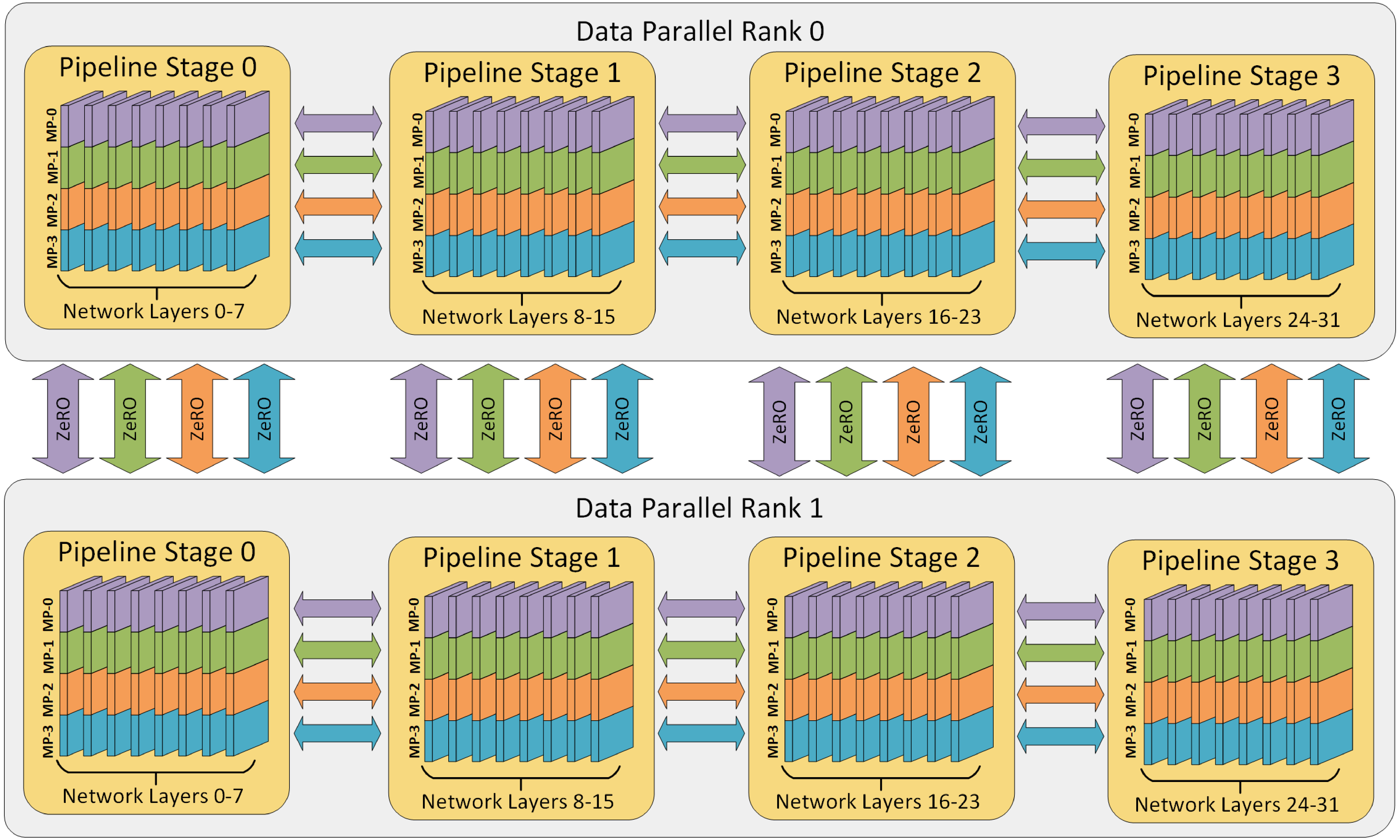


Memory linear in sequence length

Where We Are: LLMs

- Transformers and Attentions
- LLM Training Optimizations
  - Flash attention
  - **3D parallelism**
- LLM Inference and Serving
  - Paged attention
  - Continuous batching
  - Speculative decoding
- Scaling Laws
- Long context

How LLMs are trained today



Summary: How LLMs are trained today

- Outer Loop 1:
  - Inter-op parallelism + 1F1B
- Outer Loop 2: Intra-op parallelism based on model architecture
    - Zero-2 / Zero-3 + data parallelism
    - Megatron-LM tensor parallelism or Expert parallelism
- Outer Loop 3:
  - Gradient checkpointing and recomputation at backward
- Inner Loop 4:
  - Graph fusion
- Inner Loop 5:
  - Operator-level optimization: tiling, flash attention, etc.

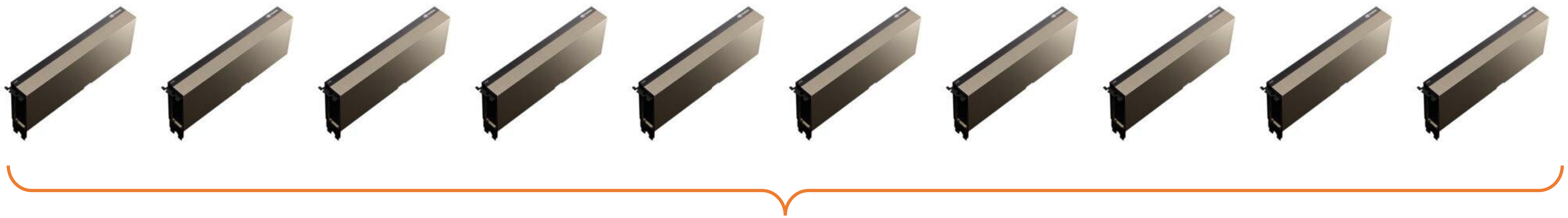
Side effects of Flash Attention

- Because we do not materialize the N x N intermediate matrix, we decrease peak memory
- Because of decreased peak memory, we can use a larger micro batch size (significantly larger, e.g., 1 -> 32)
- Because of large per-device batch size, much higher AI

Where We Are: LLMs

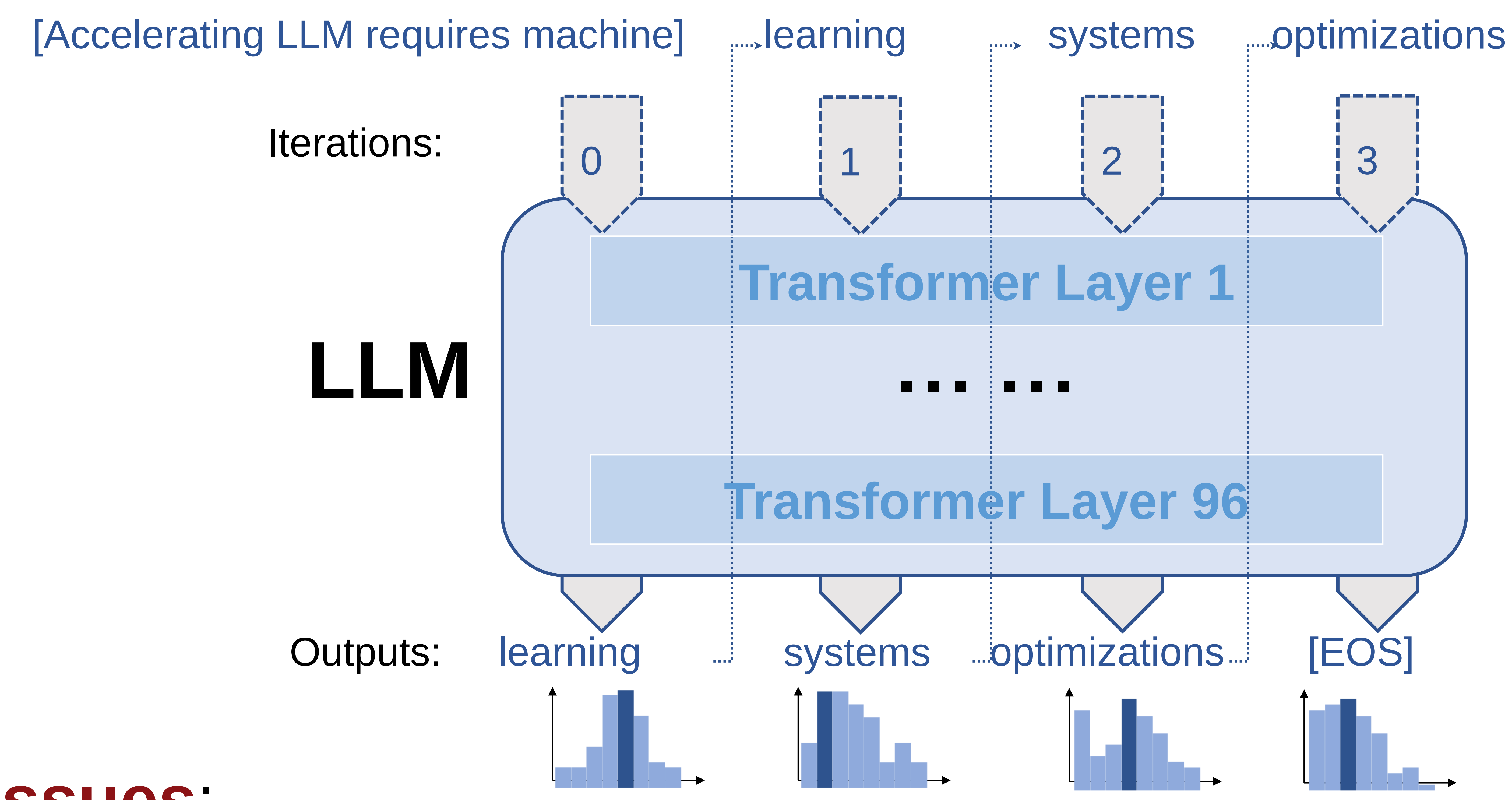
- Transformers and Attentions
- LLM Training Optimizations
  - Flash attention
  - 3D parallelism
- **LLM Inference and Serving**
  - Continuous batching
  - Paged attention
  - Speculative decoding
- Scaling Laws
- Long context

LLMs are Slow and Expensive to Serve



- **At least ten** A100-40GB GPUs to serve 175B GPT-3 in half precision
- Generating 256 tokens takes **~20 seconds**
- Cannot process many requests in parallel
  - Per-request key/value cache takes **3GB GPU memory**

Recall: Incremental Decoding



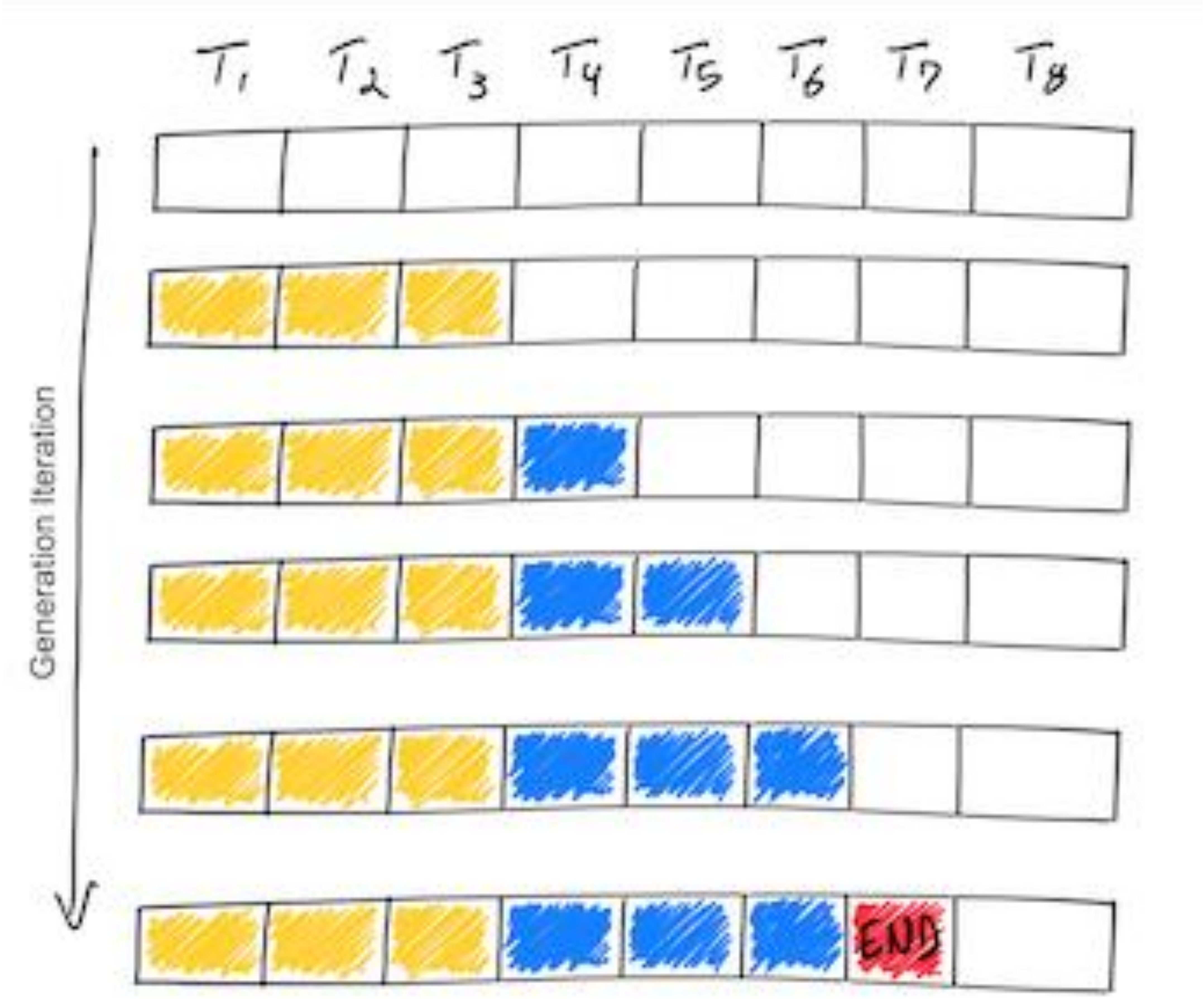
Main issues:

- Limited degree of parallelism → underutilized GPU resources
- Need all parameters to decode a token → bottlenecked by GPU memory access

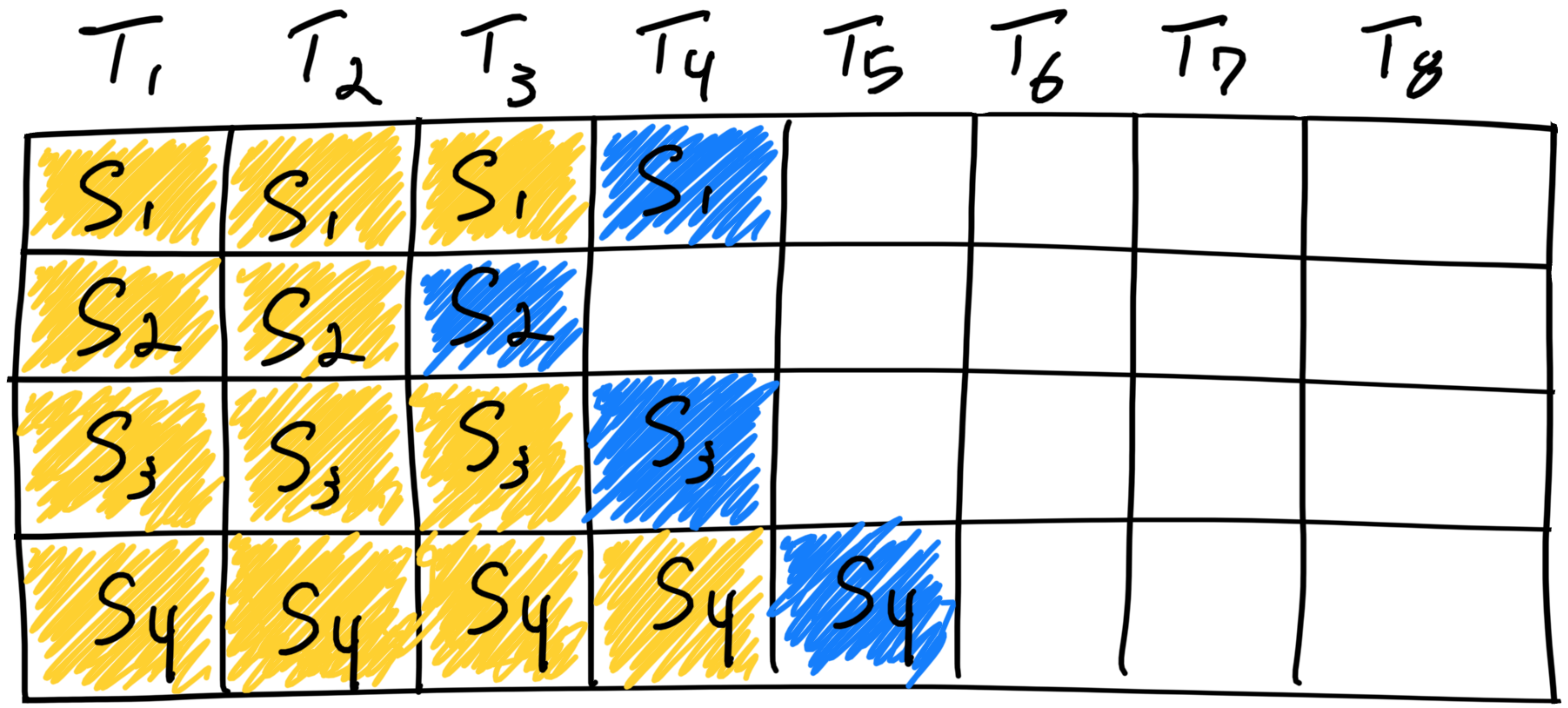
Outline: LLMs Serving Techniques

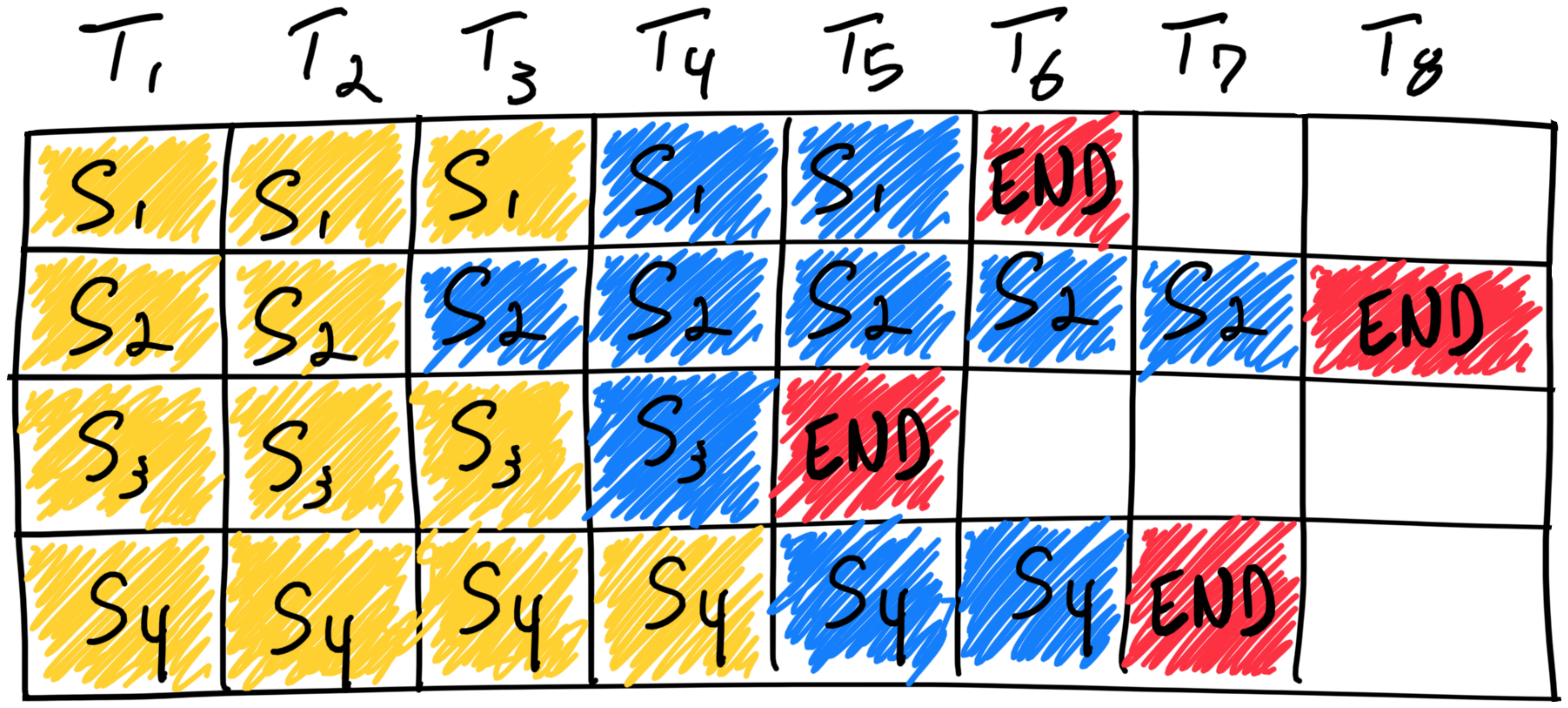
- **Continuous Batching**
- Paged Attention
- Speculative Decoding

LLM Decoding Timeline



Batching Requests to Improve GPU Performance

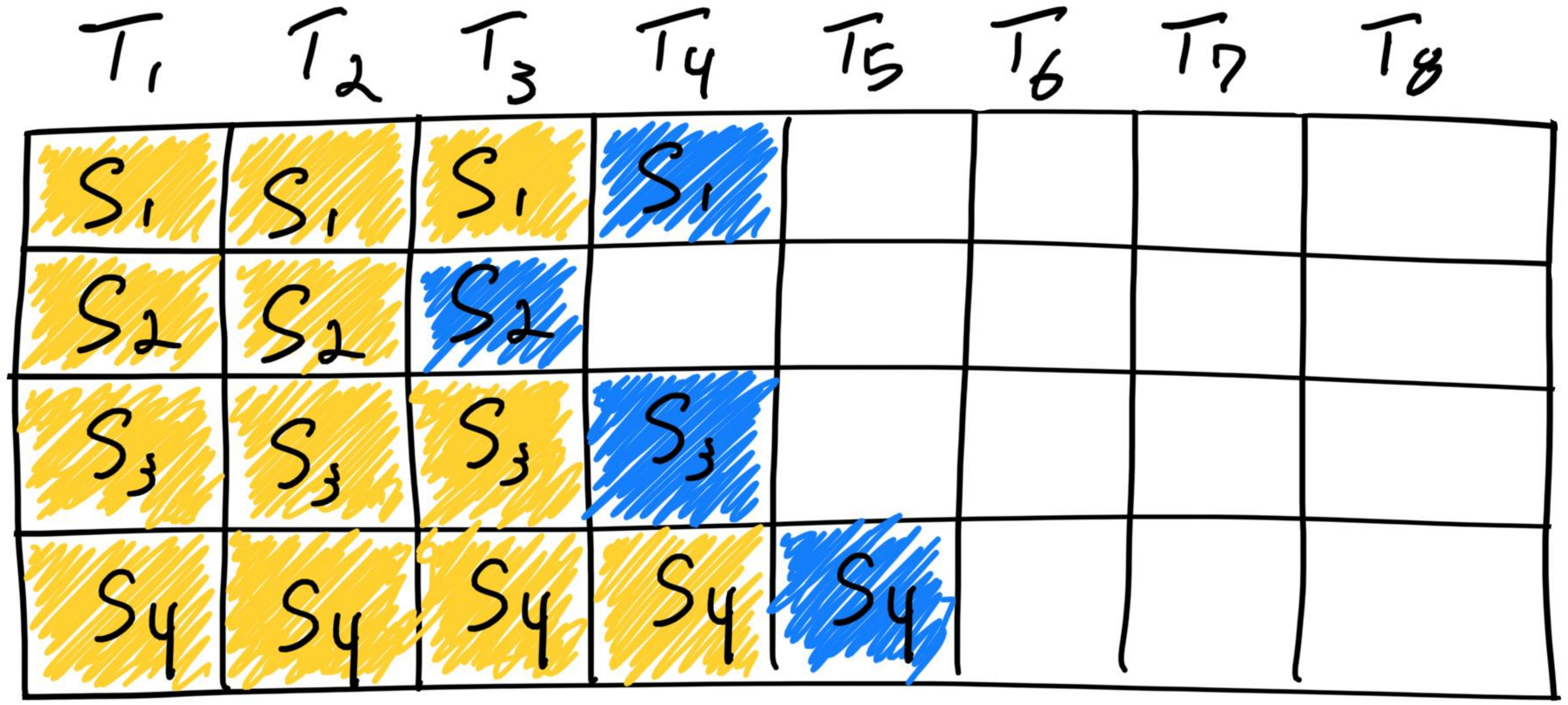


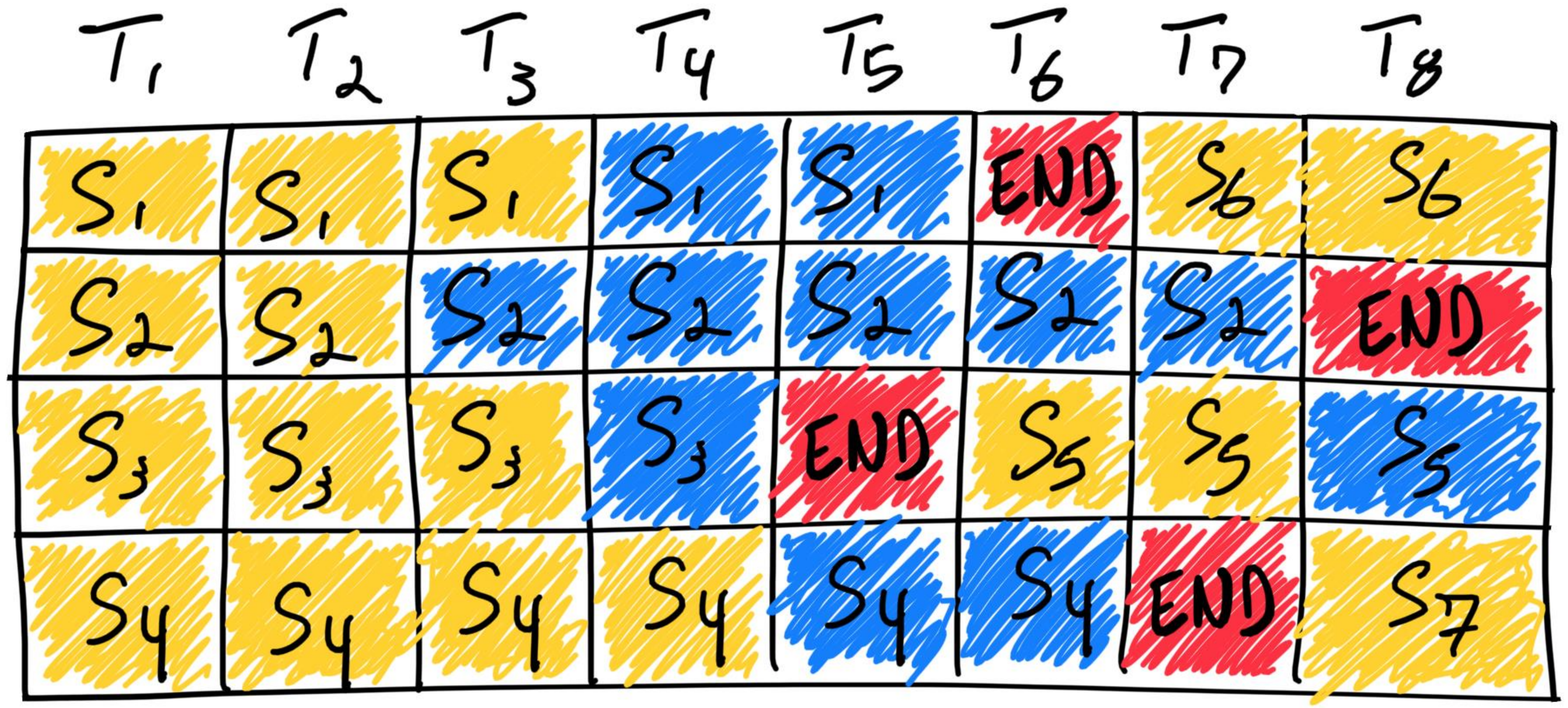


Issues with static batching:

- Requests may complete at different iterations
- Idle GPU cycles
- New requests cannot start immediately

Continuous Batching



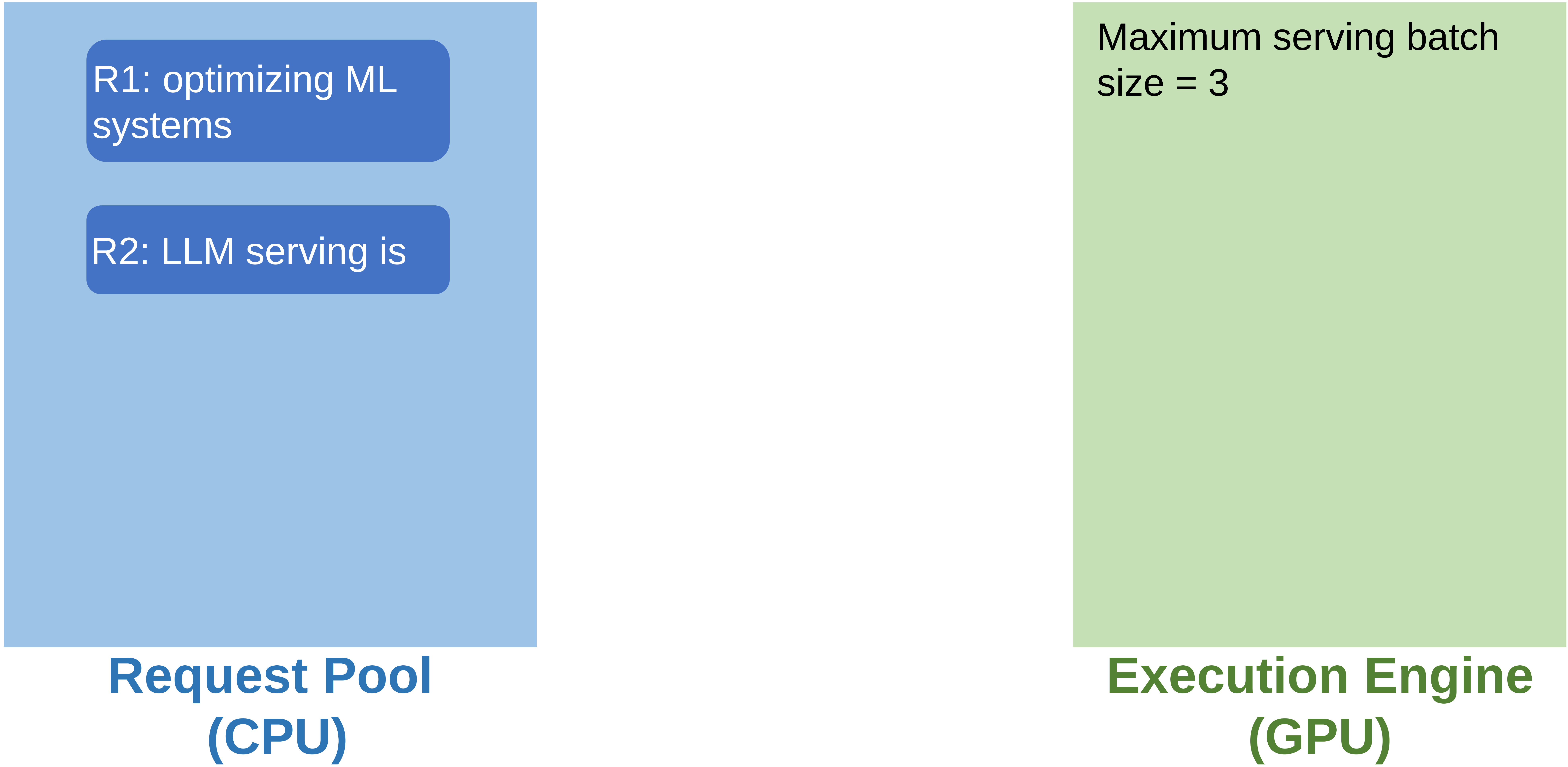


Benefits:

- Higher GPU utilization
- New requests can start immediately

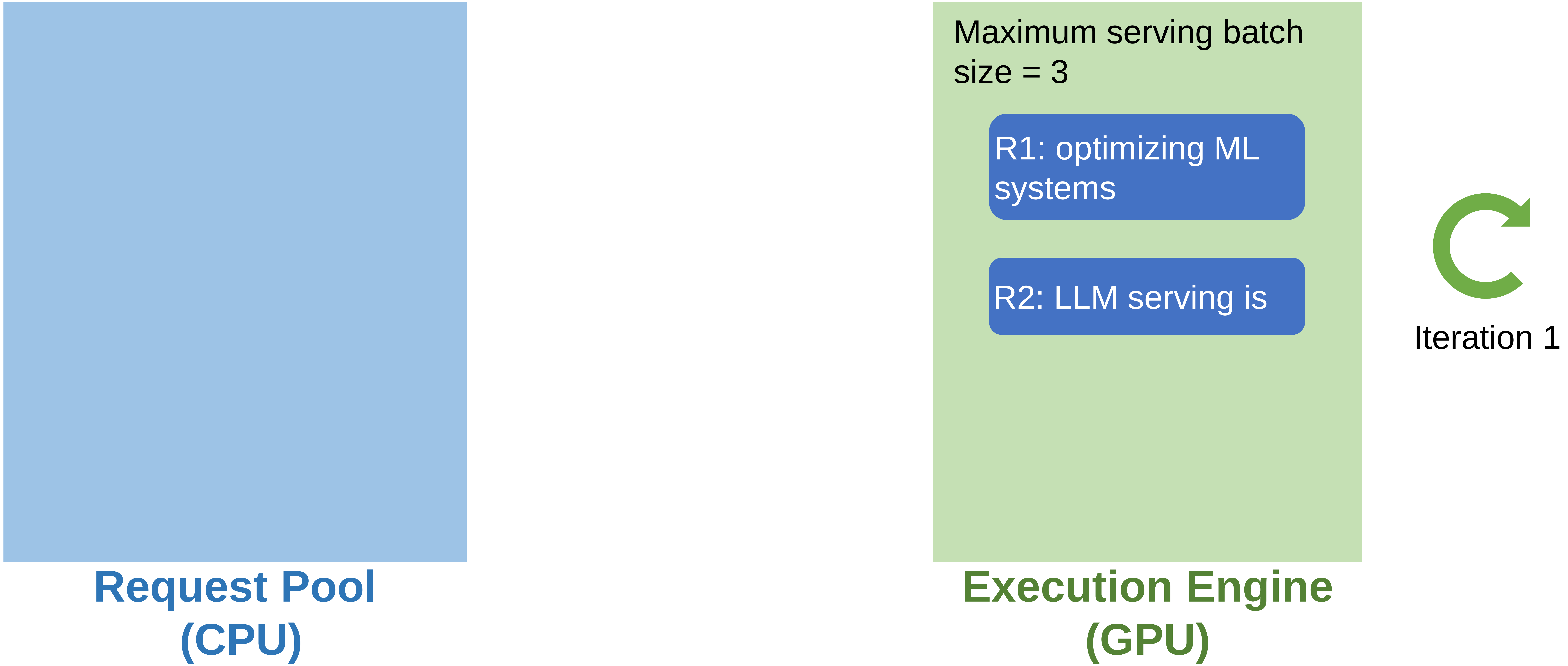
Continuous Batching Step-by-Step

- Receives two new requests R1 and R2



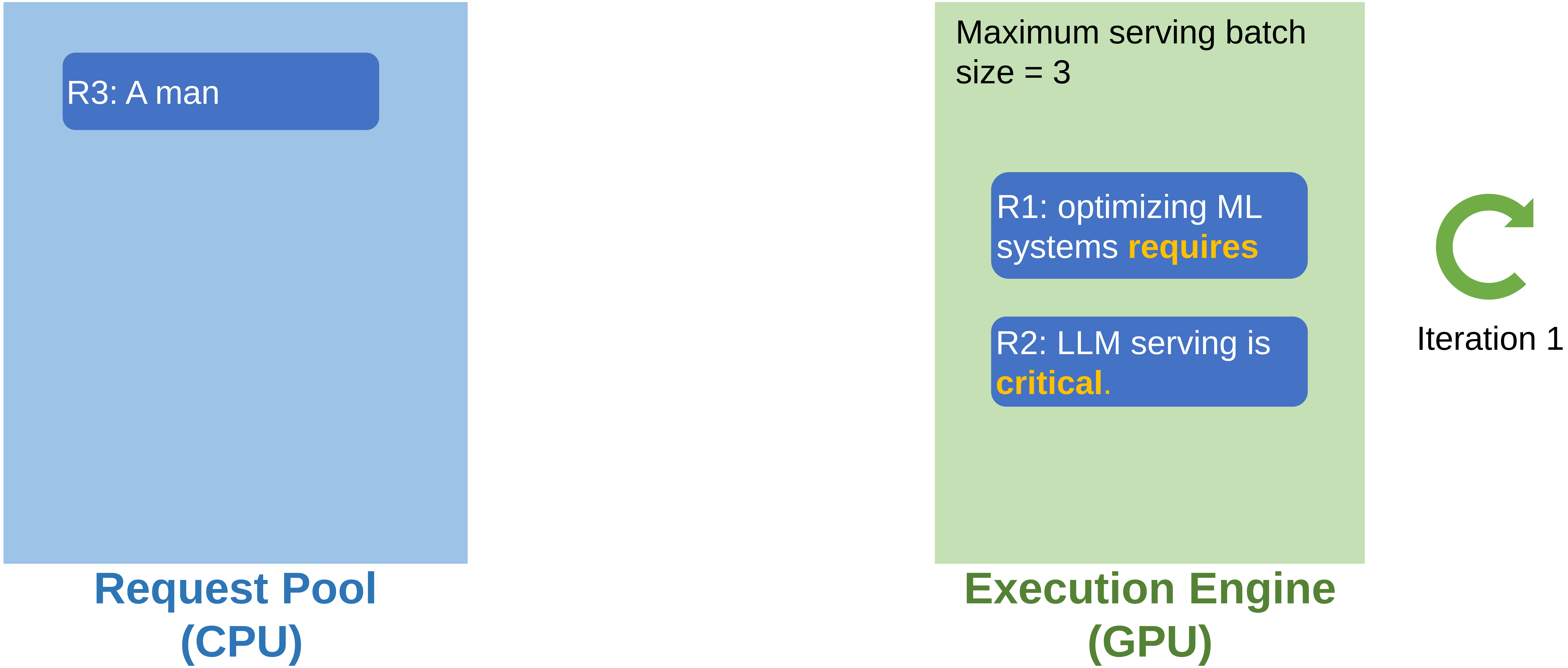
Continuous Batching Step-by-Step

- Iteration 1: decode R1 and R2



Continuous Batching Step-by-Step

- Receive a new request R3; finish decoding R1 and R2



Continuous Batching Step-by-Step

- Iteration 2: decode R1, R2, R3; receive R4, R5; R2 completes

R4: A dog is

R5: How are

Request Pool (CPU)

Maximum serving batch size = 3

R3: A man **is**

R1: optimizing ML systems **requires ML**

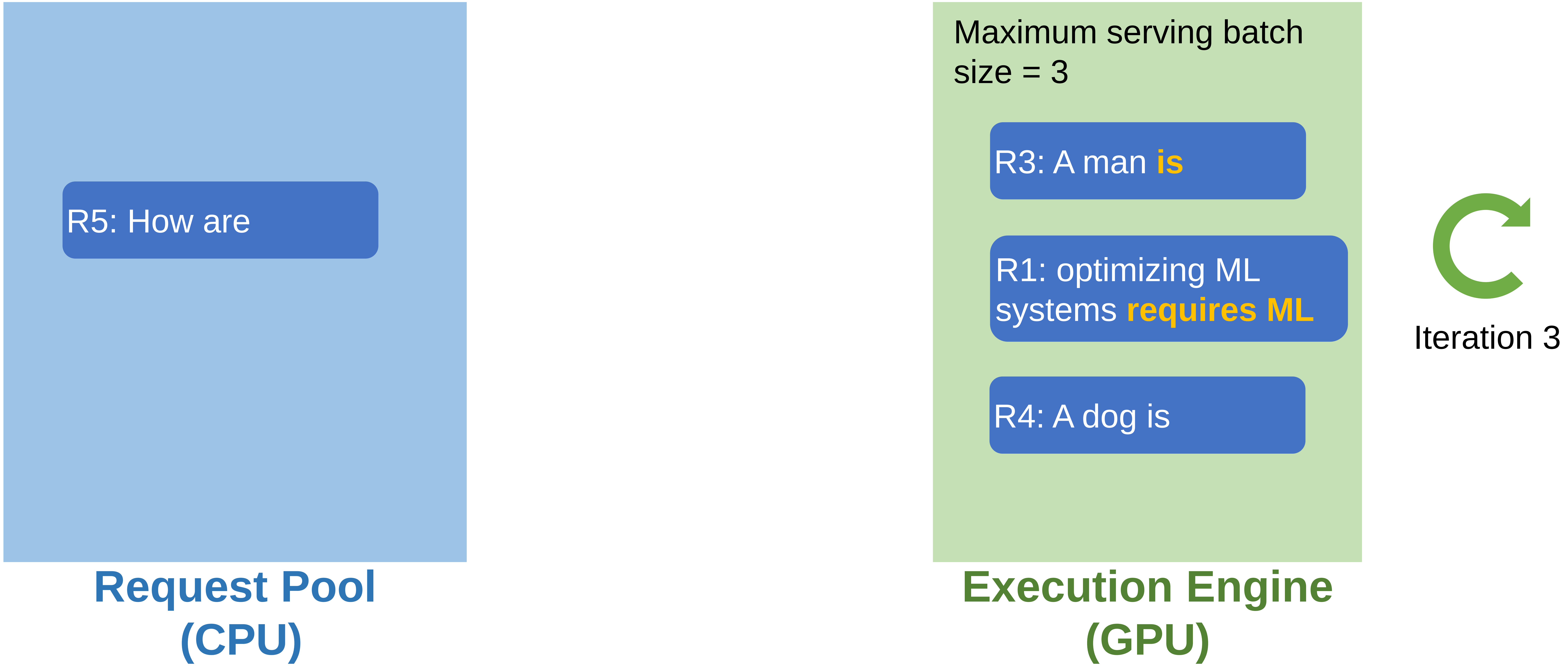
R2: LLM serving is **critical**. <EOS>

Execution Engine (GPU)

Iteration 2

Continuous Batching Step-by-Step

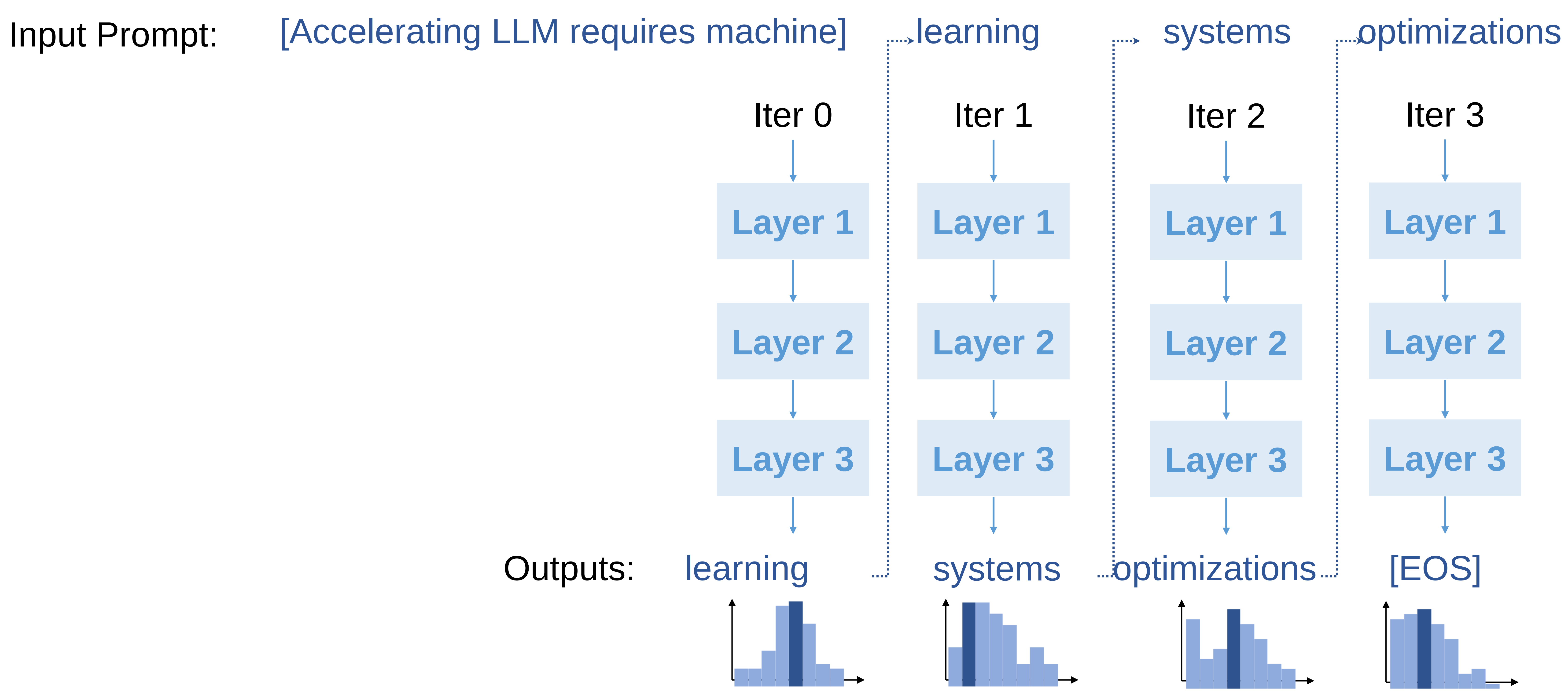
- Iteration 3: decode R1, R3, R4



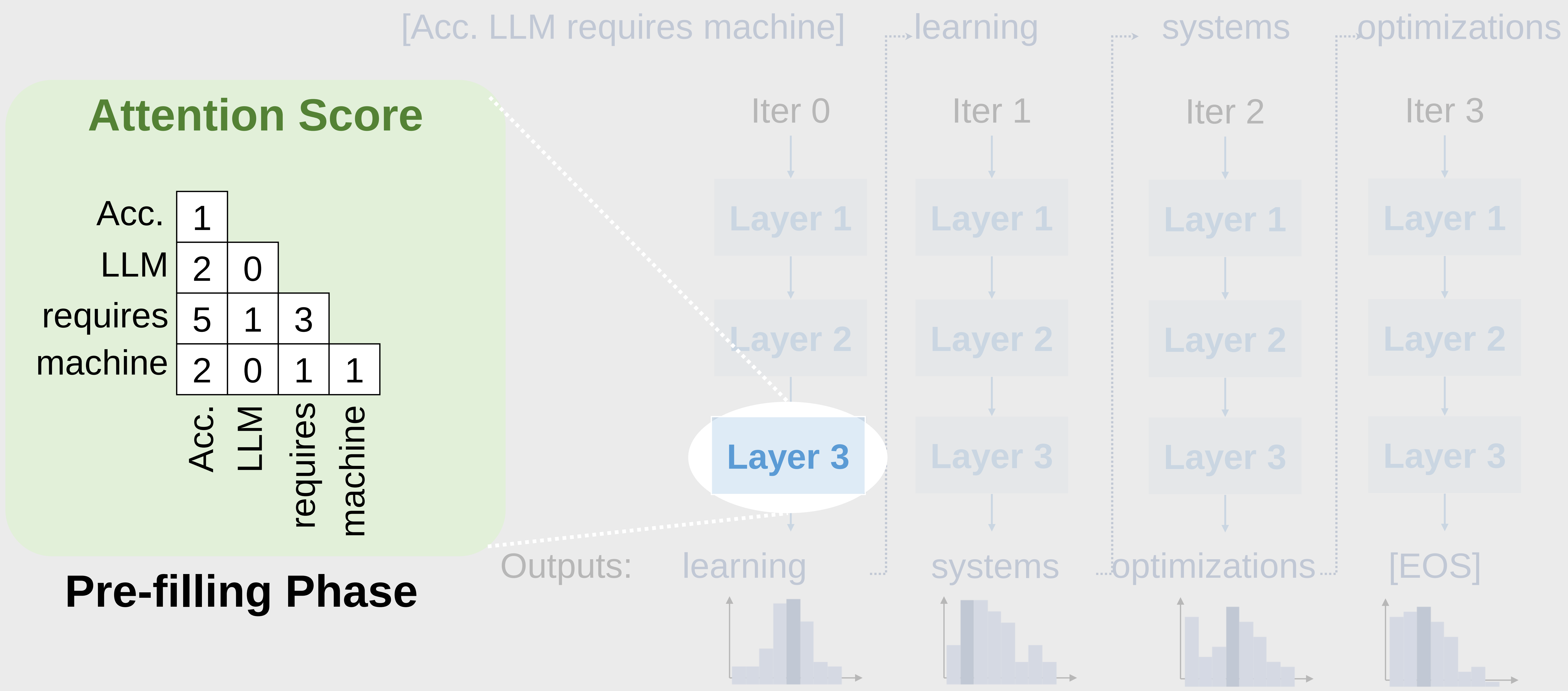
Continuous Batching

- Handle early-finished and late-arrived requests more efficiently
- Higher GPU utilization

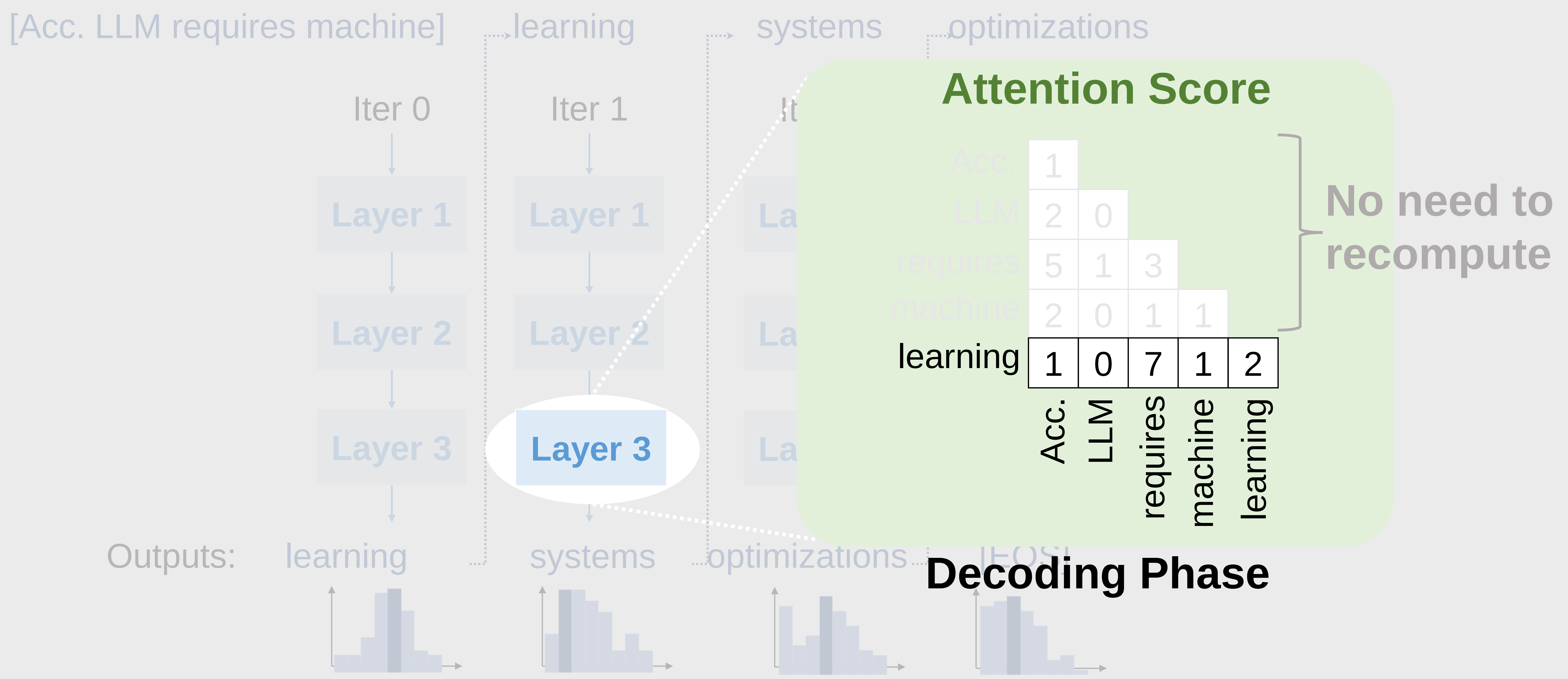
Generative LLM Inference: Autoregressive Decoding



Generative LLM Inference: Autoregressive Decoding



Generative LLM Inference: Autoregressive Decoding

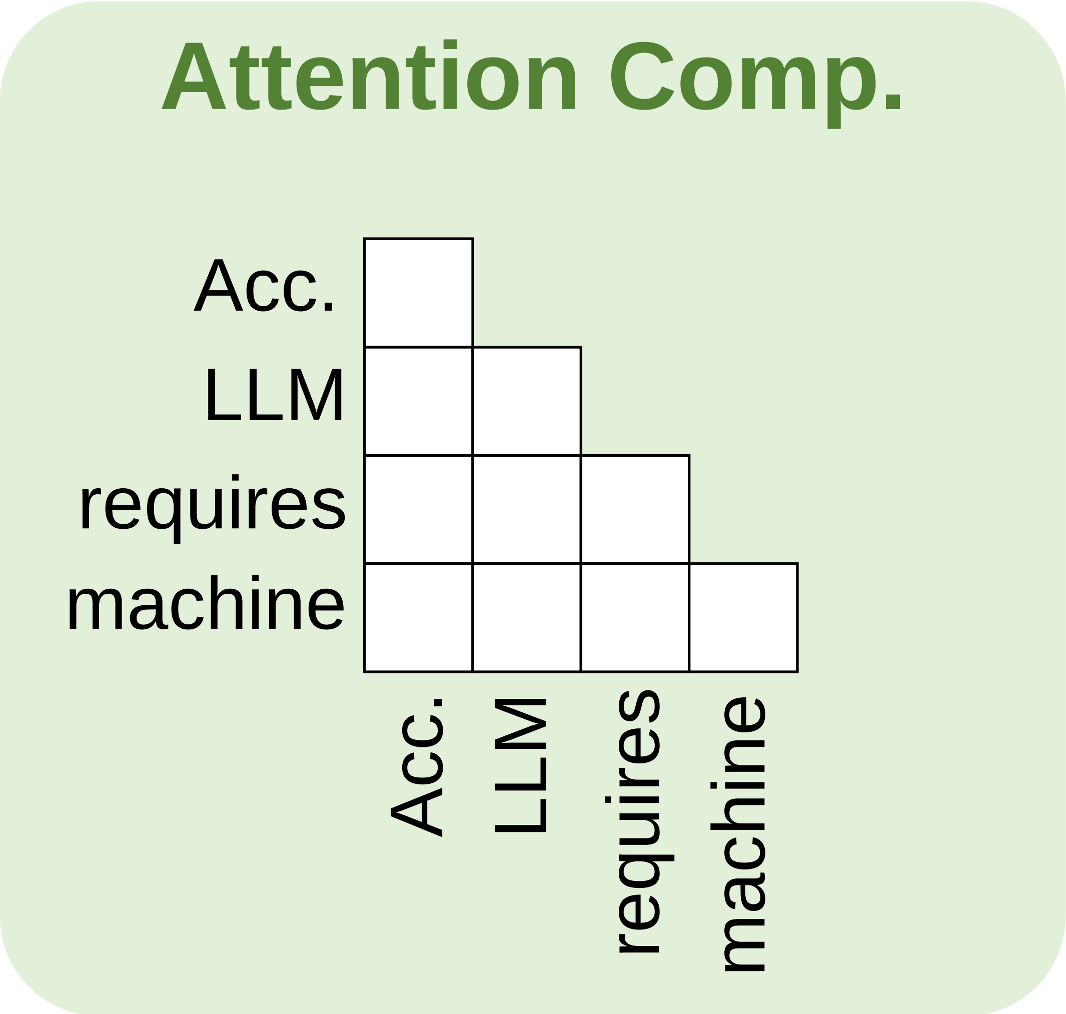


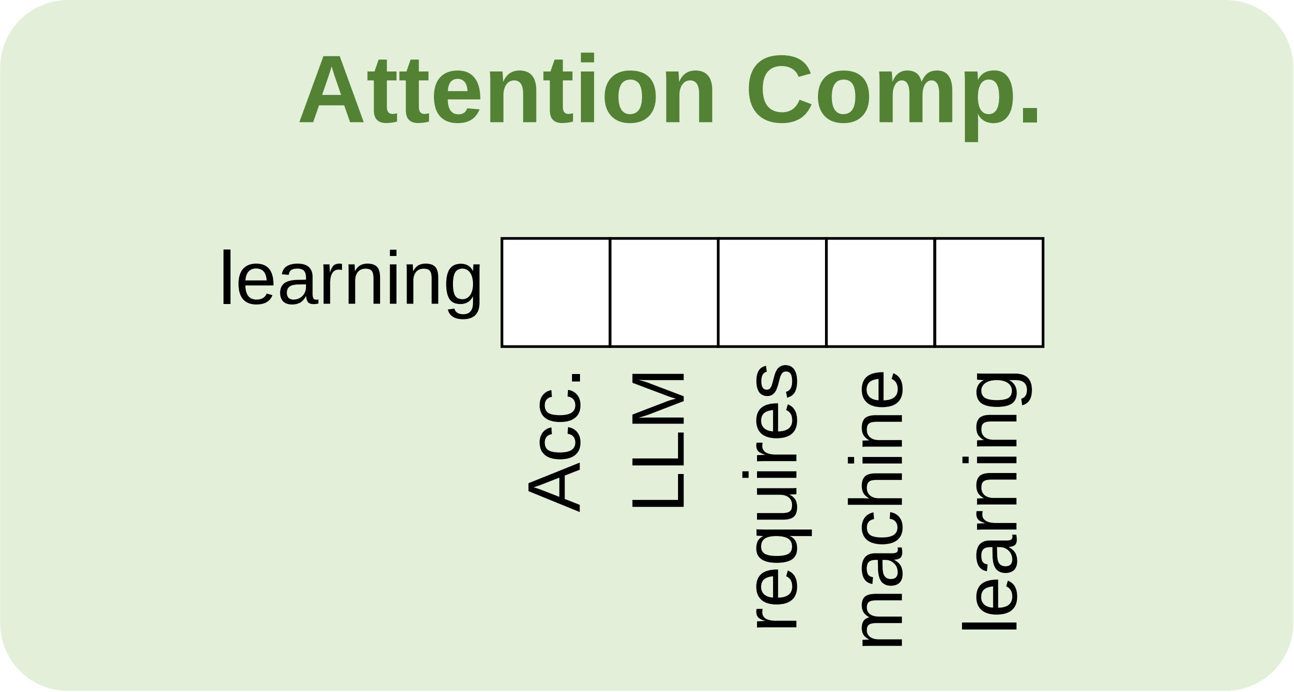
Generative LLM Inference: Autoregressive Decoding

- **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?





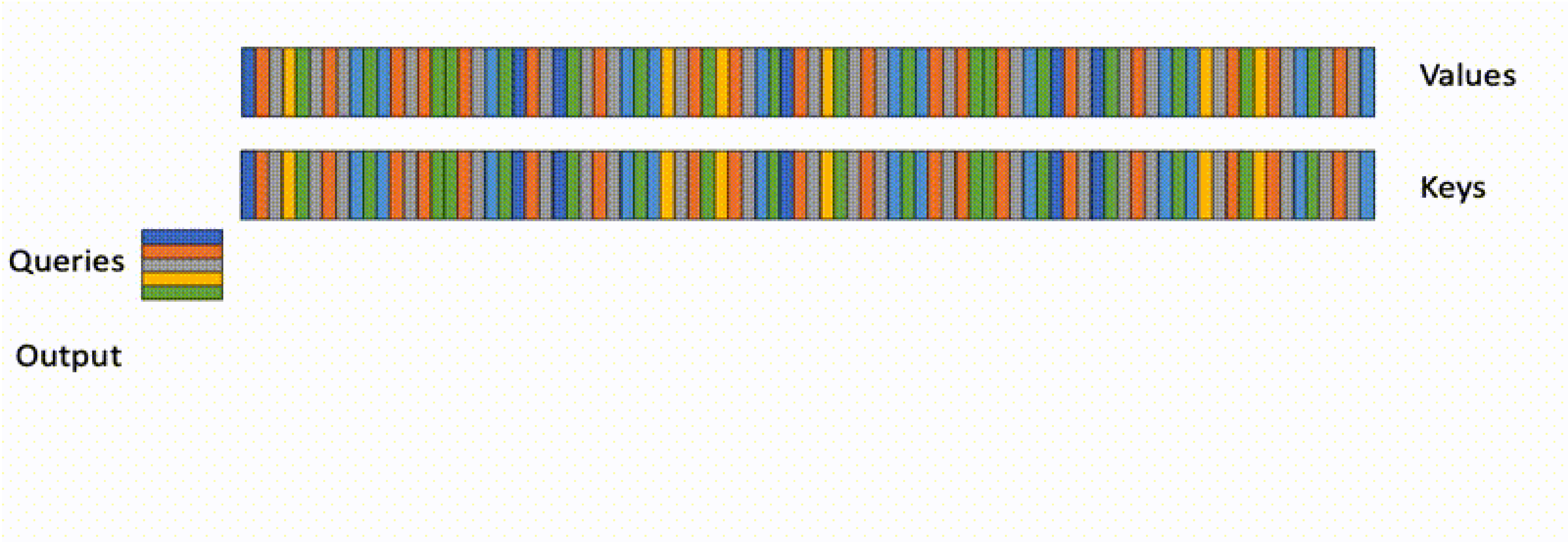
Pre-filling phase:

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

Decoding phase:

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

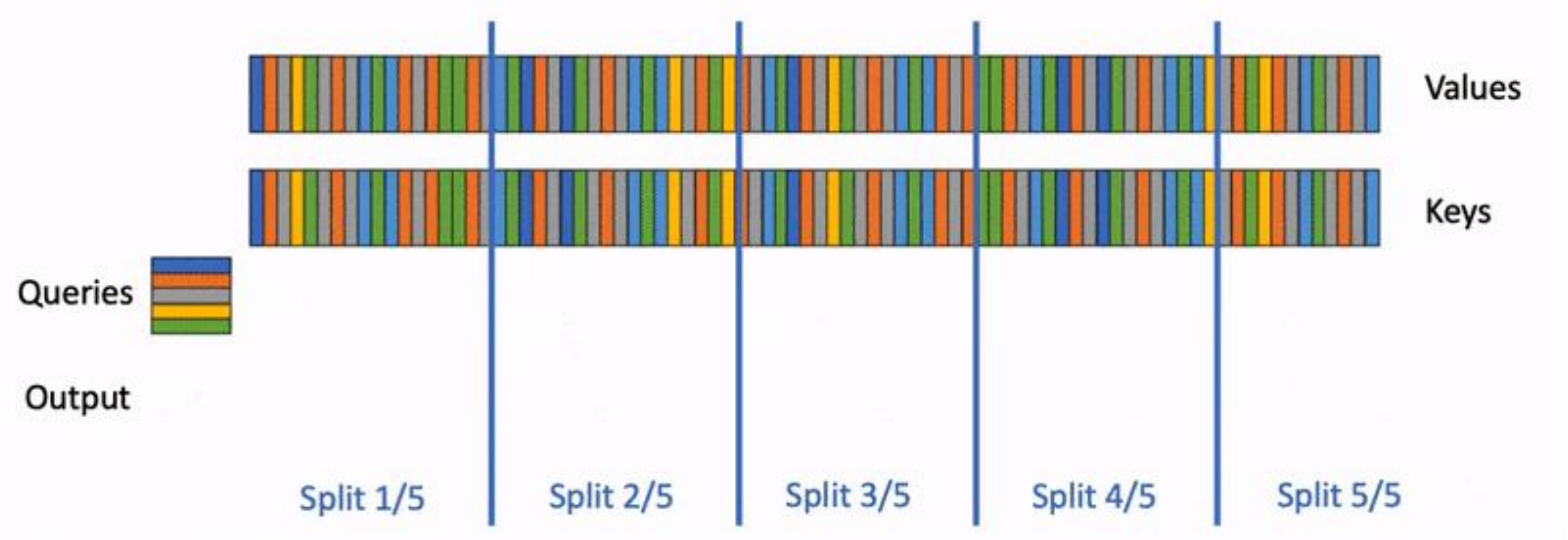
FlashAttention Processes K/V Sequentially



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

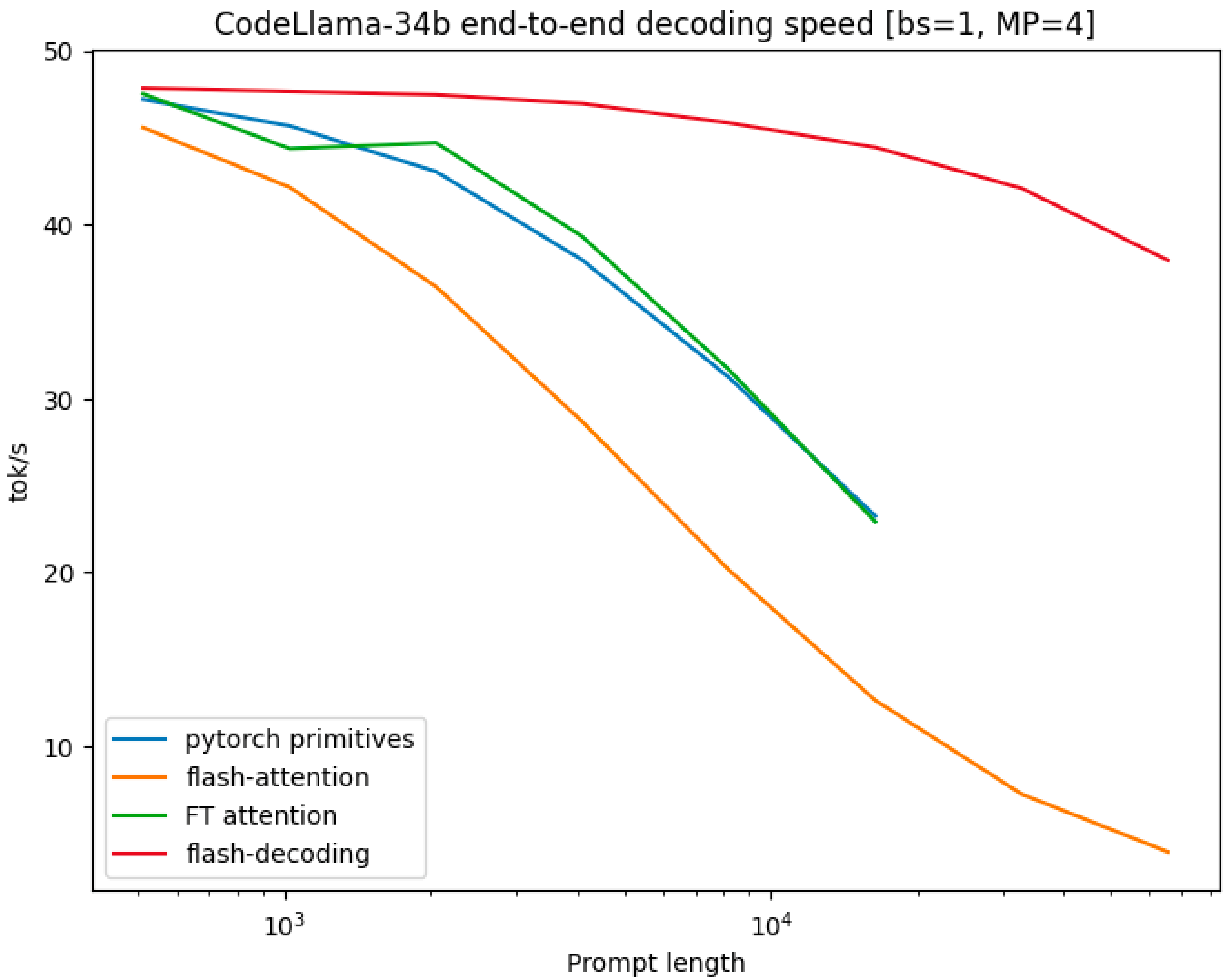
Flash-Decoding Parallelizes Across Keys/Values

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



Outline: Attention Optimizastions

Part 1: LLM Training

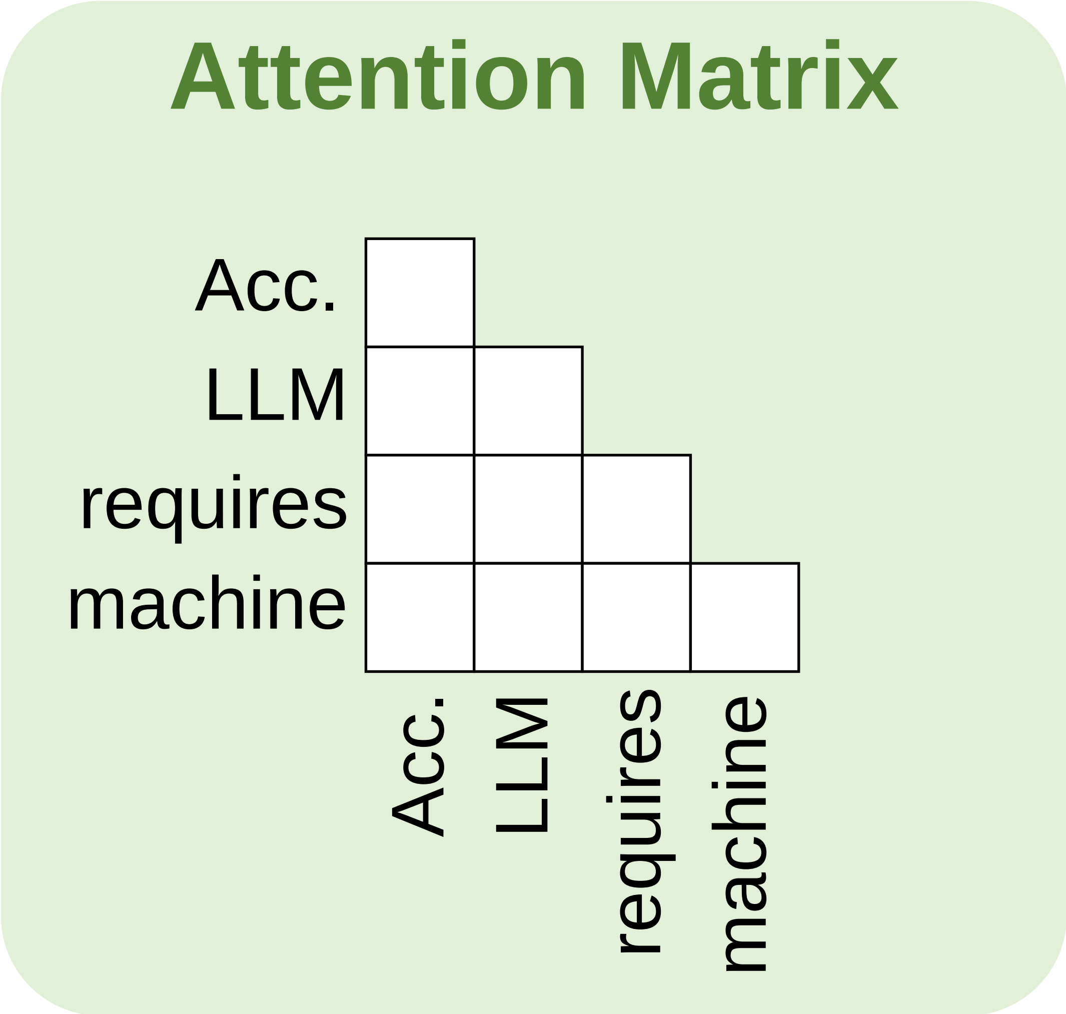
- FlashAttention

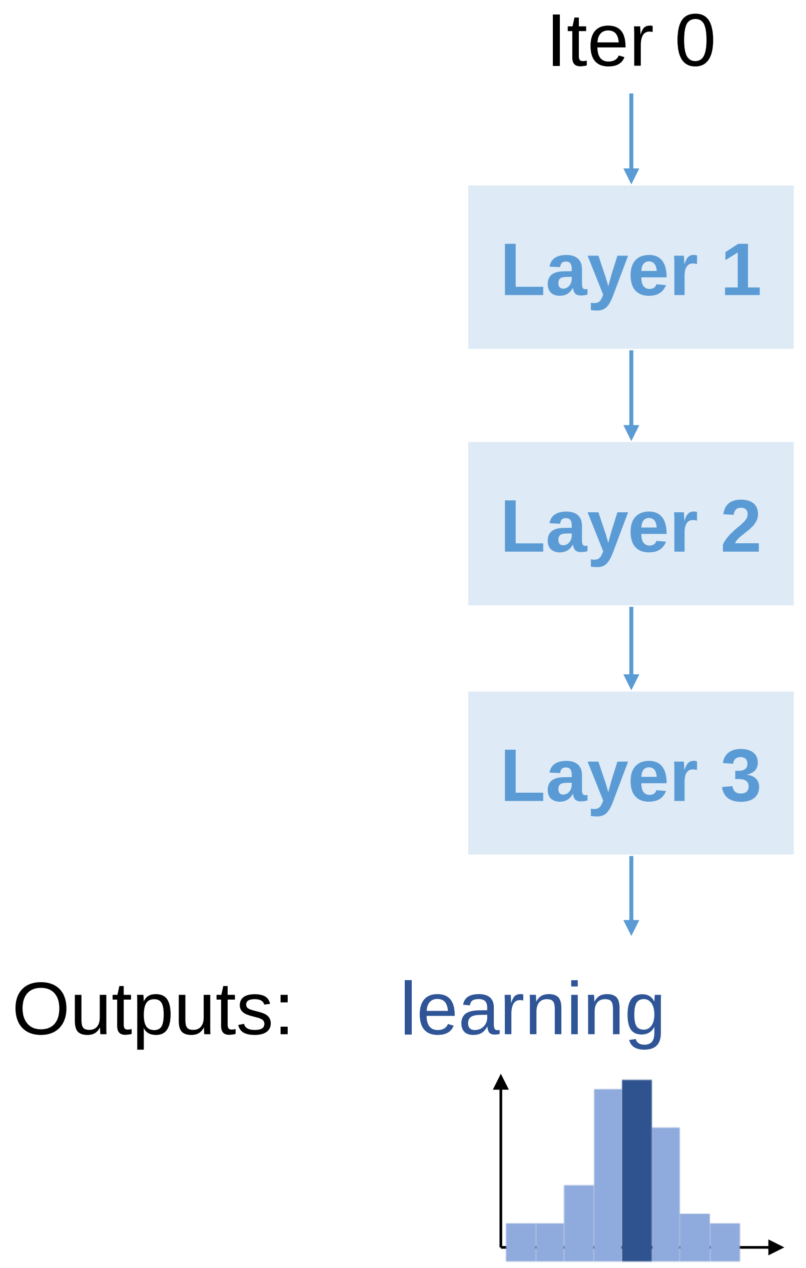
Part 2: LLM Inference (Auto-regressive Decoding)

- Flash-Decoding
- **PagedAttention**

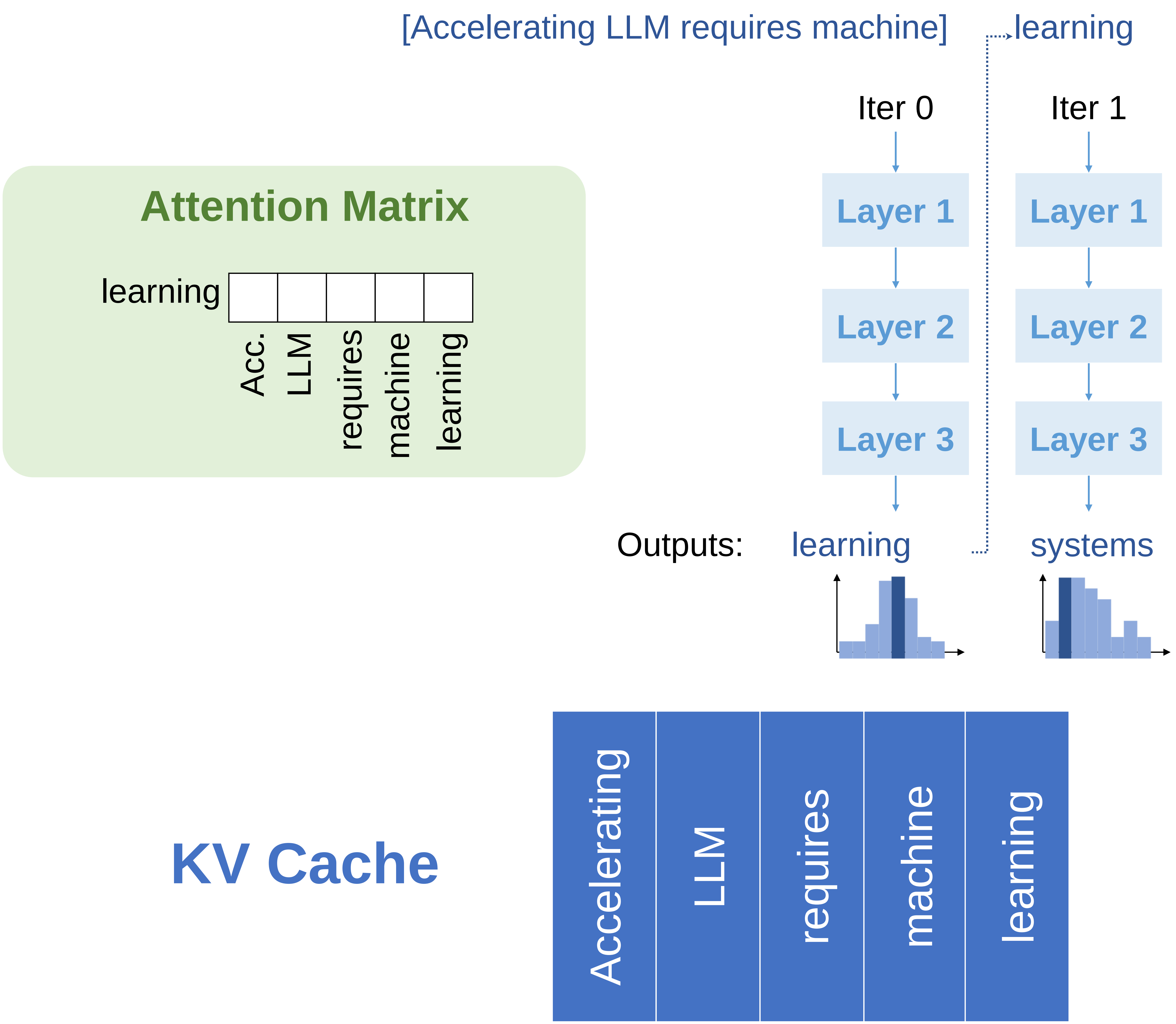
KV Cache Dynamically Grows and Shrinks

[Accelerating LLM requires machine]

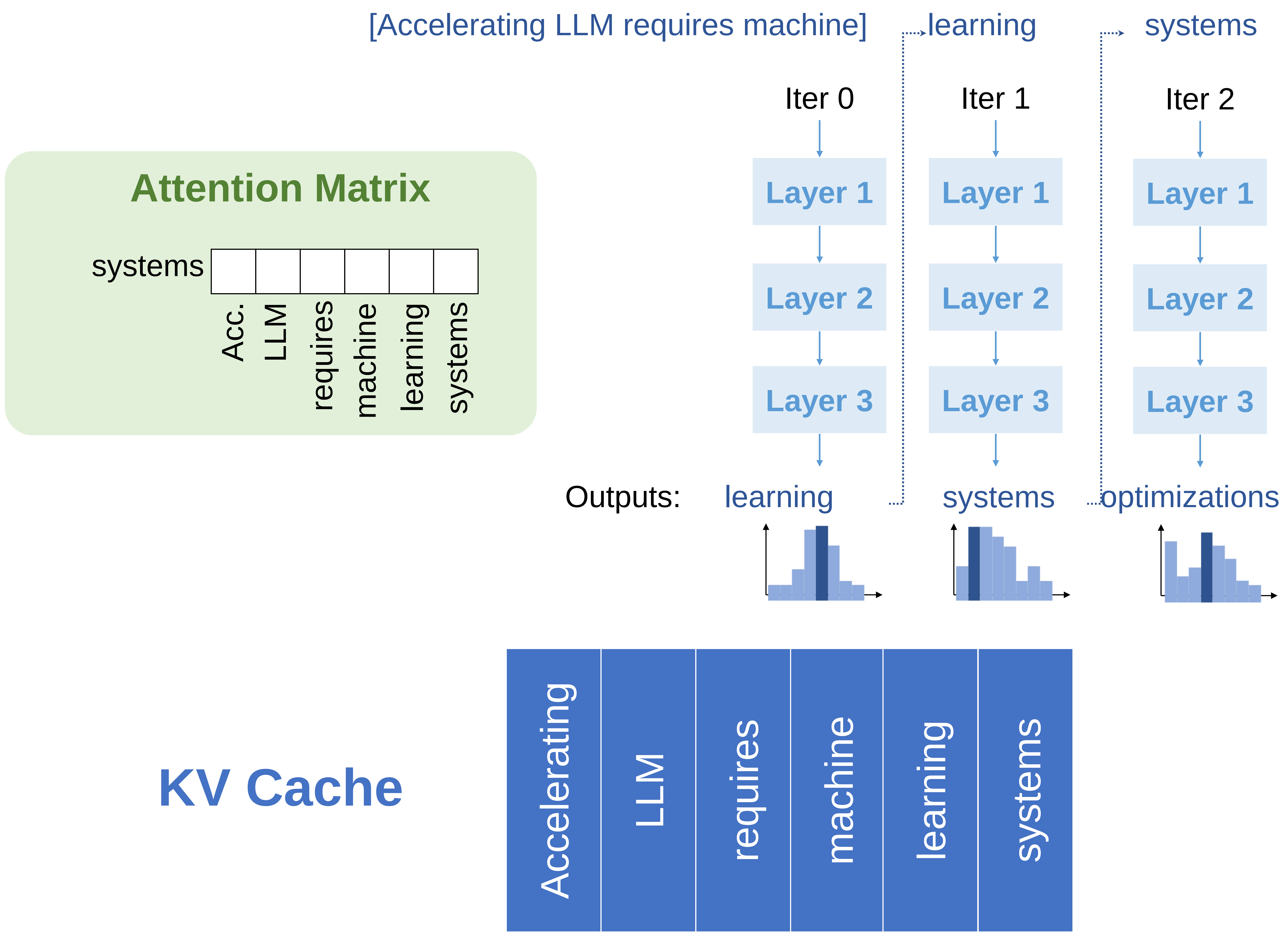




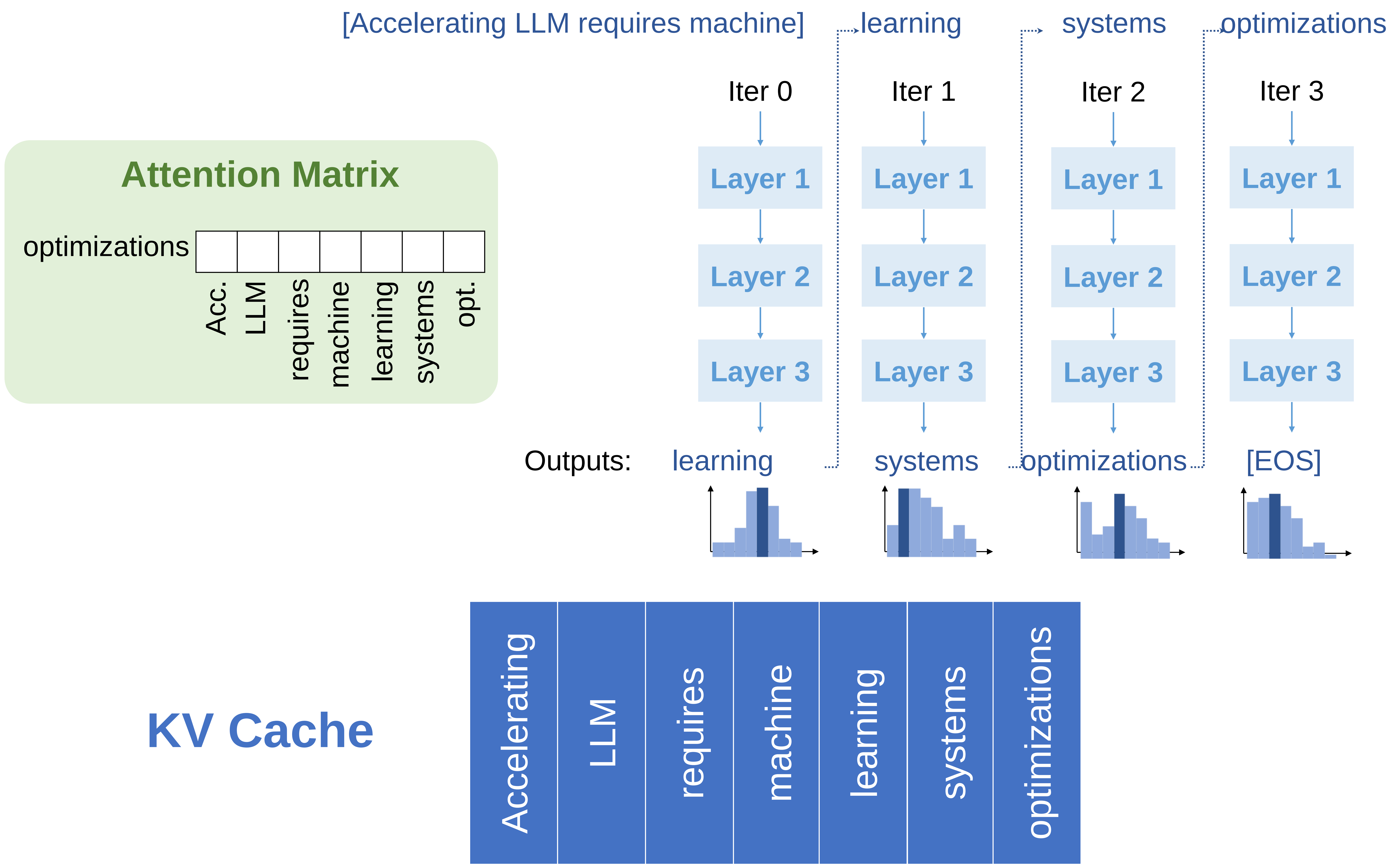
KV Cache Dynamically Grows and Shrinks



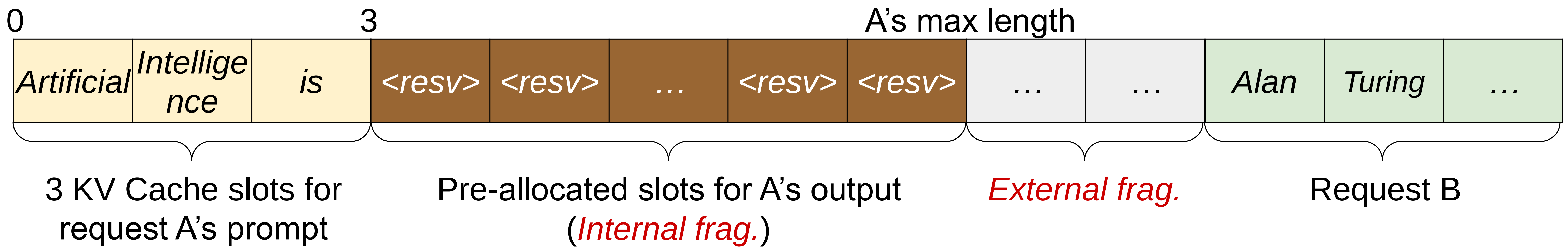
KV Cache Dynamically Grows and Shrinks



KV Cache Dynamically Grows and Shrinks



Static KV Cache Management Wastes Memory

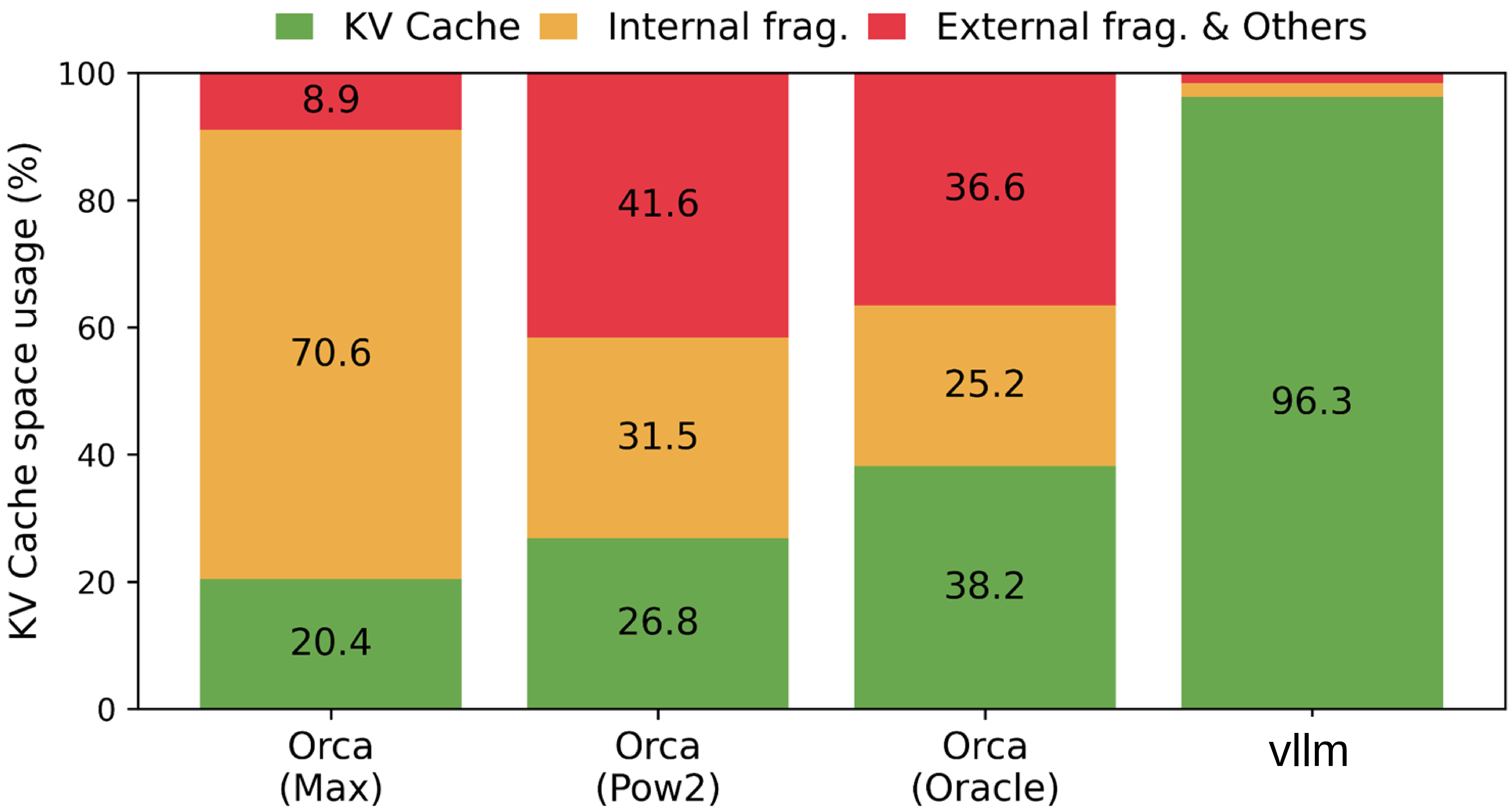


- **Pre-allocates contiguous** space of memory to the request's maximum length

- Memory fragmentation
  - **Internal fragmentation** due to unknown output length
  - **External fragmentation** due to non-uniform per-request max lengths

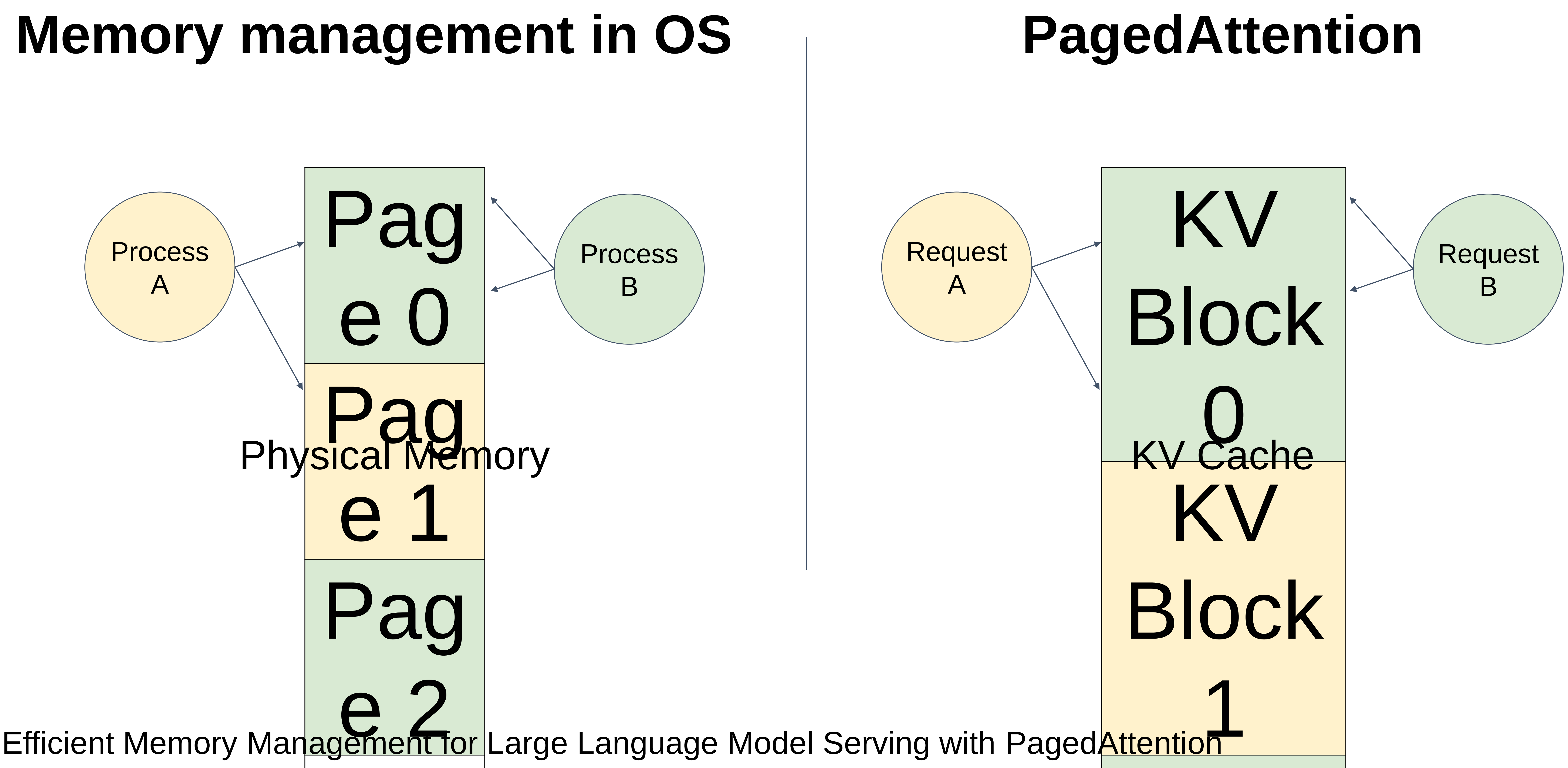
Significant Memory Waste in KV Cache

- Only 20-40% of KV cache is utilized to store actual token states



PagedAttention

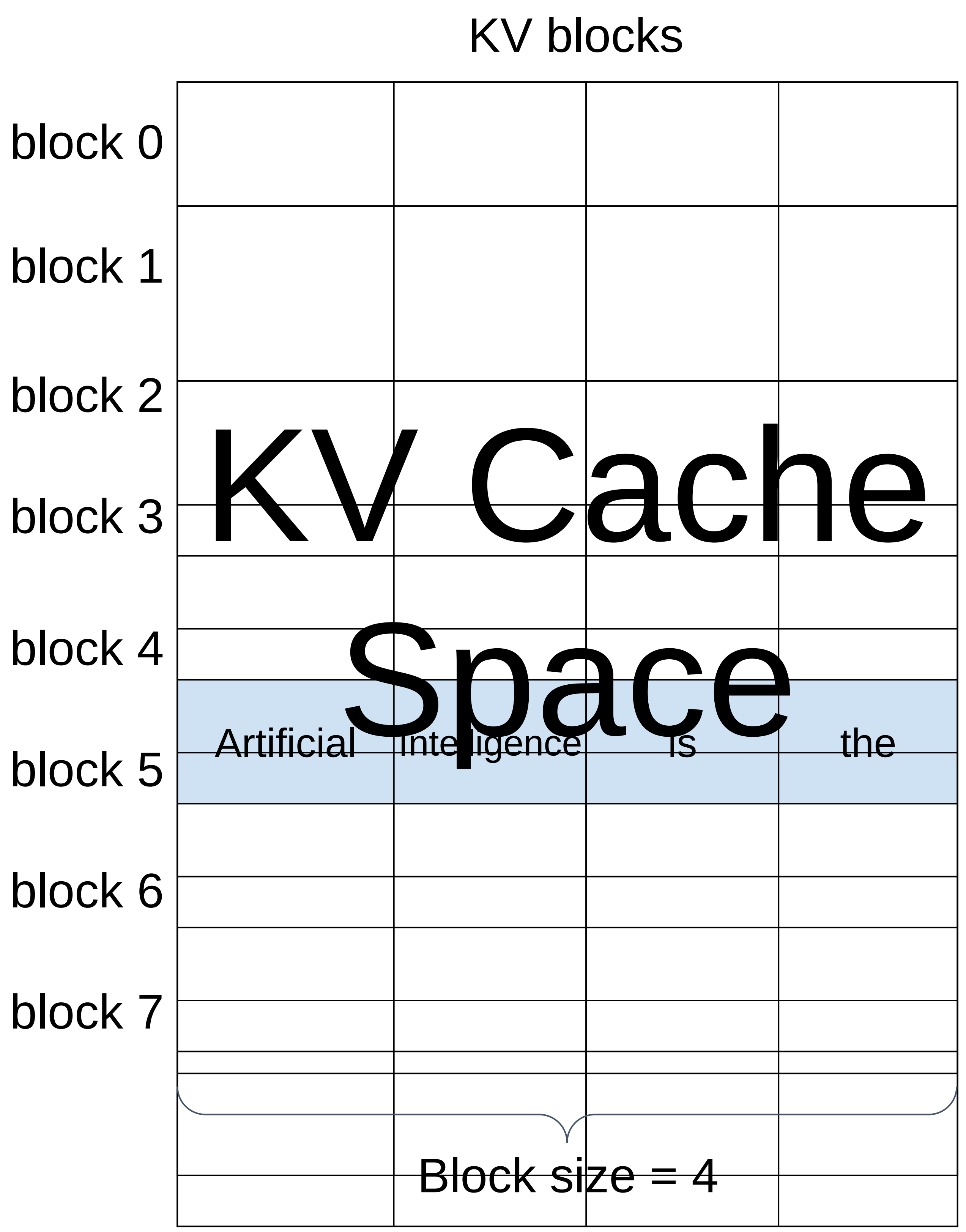
- Application-level memory paging and virtualization for KV cache



slides from vllm: Efficient Memory Management for Large Language Model Serving with PagedAttention

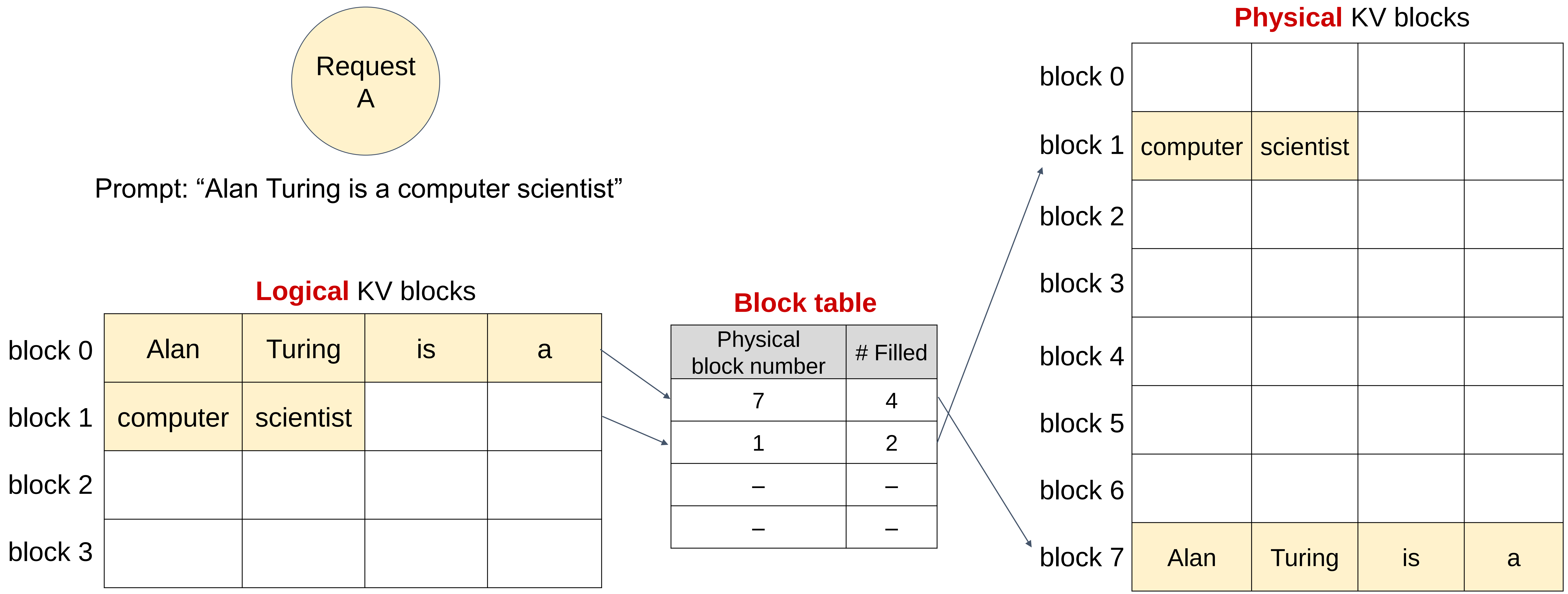
Paging KV Cache Space into KV Blocks*

- KV block is a **fixed-size** contiguous chunk of memory that stores KV states from **left to right**



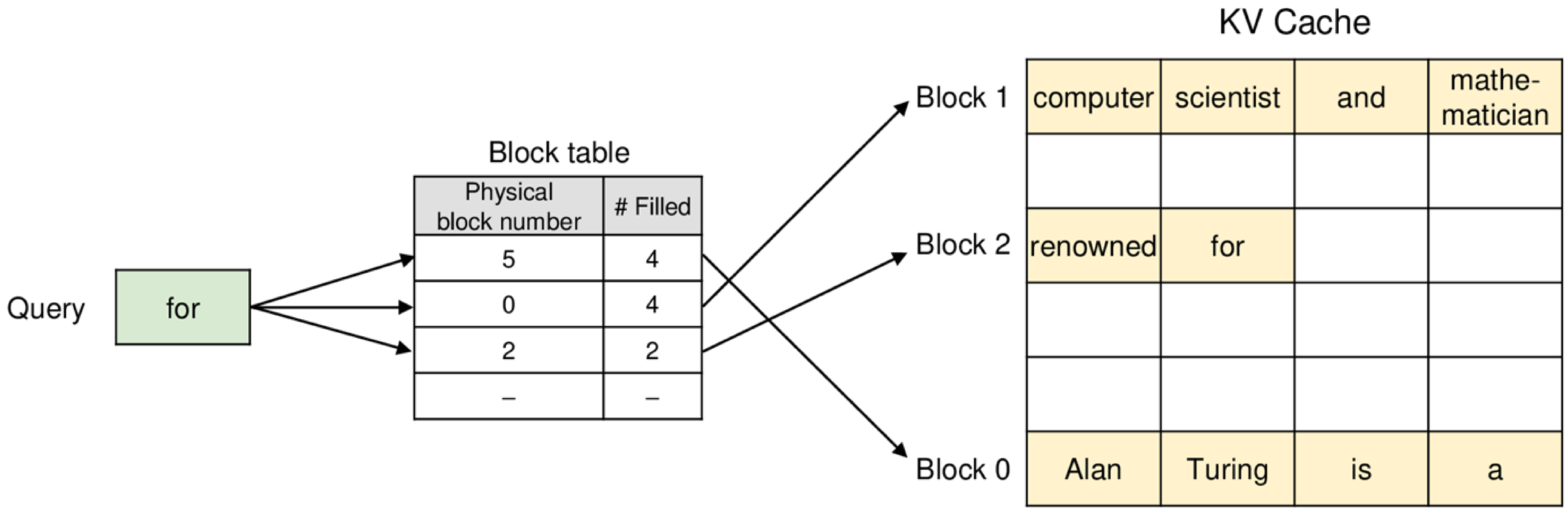
* The term ``block'' is overloaded in PagedAttention

Virtualizing KV Cache



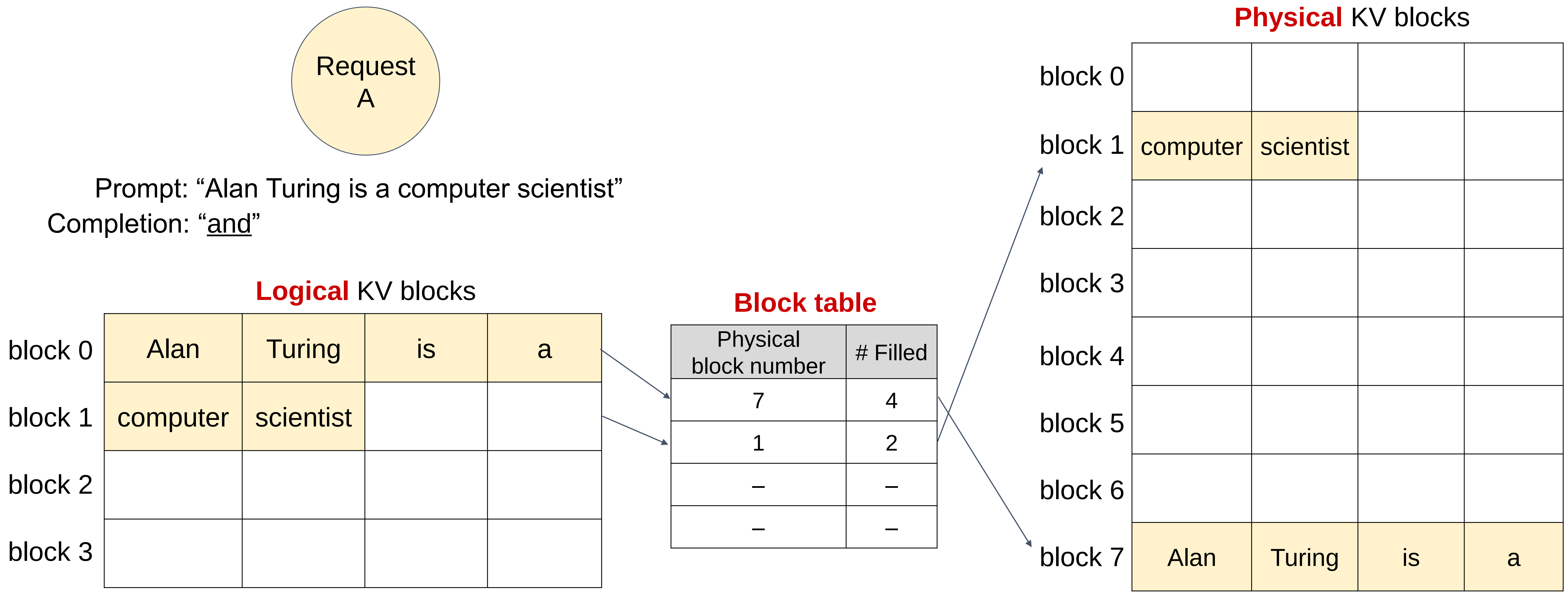
Attention with Virtualized KV Cache

1. Fetch non-contiguous KV blocks using the block table
2. Apply attention on the fly

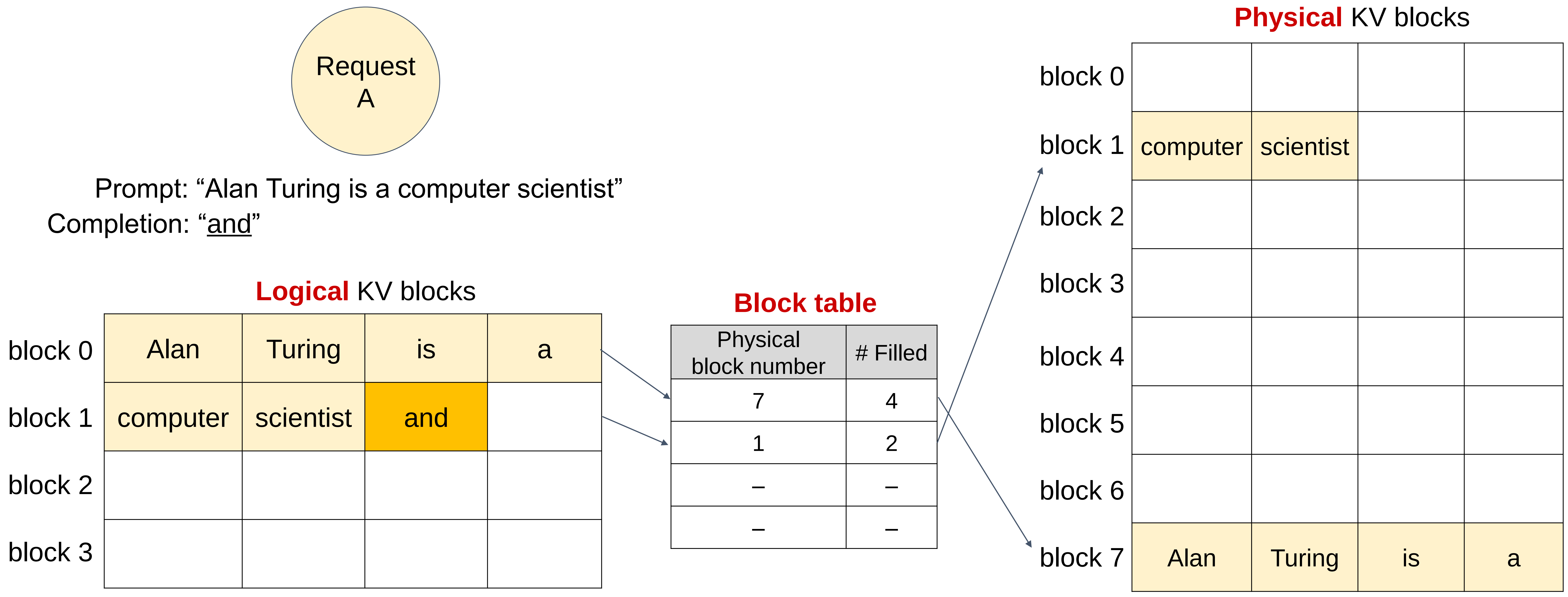


Key insight: attention is associative and commutative

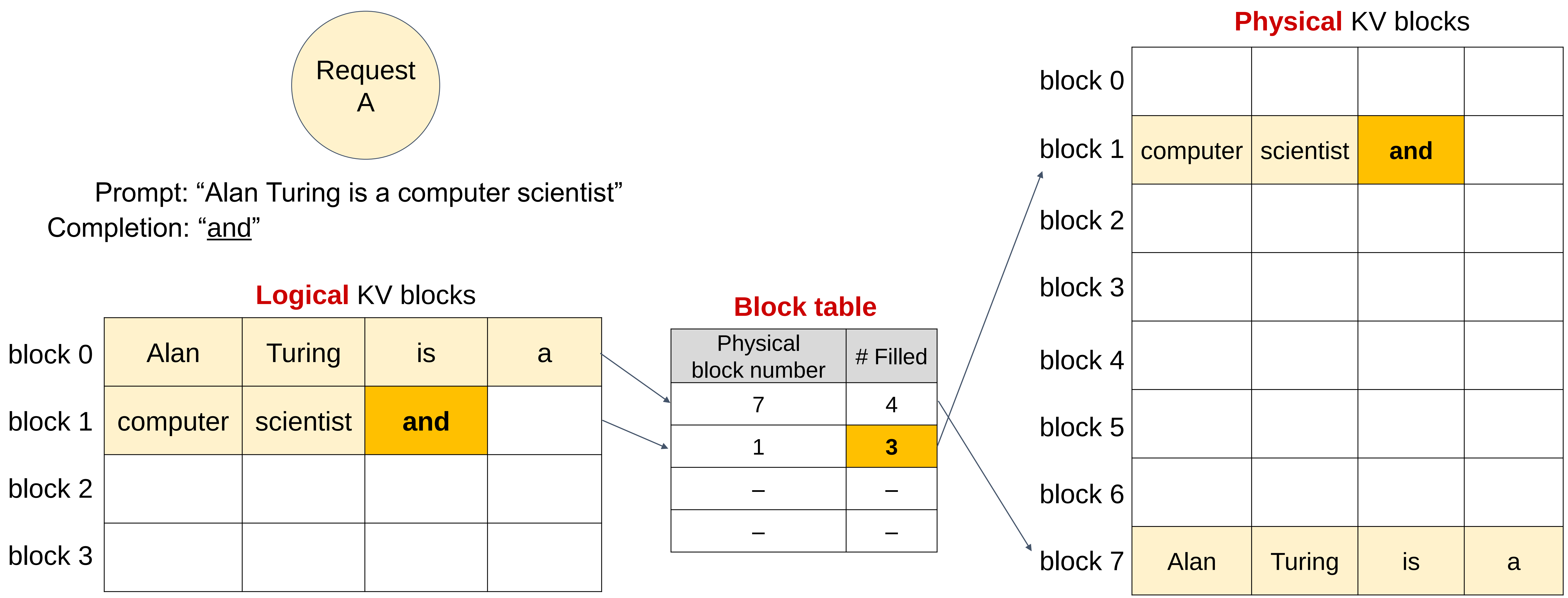
Memory Management with PagedAttention



Memory Management with PagedAttention



Memory Management with PagedAttention



Memory Management with PagedAttention

Request A

Prompt: “Alan Turing is a computer scientist”
Completion: “and mathematician”

Logical KV blocks

block 0	Alan	Turing	is	a
block 1	computer	scientist	and	**mathem atician**
block 2				
block 3				

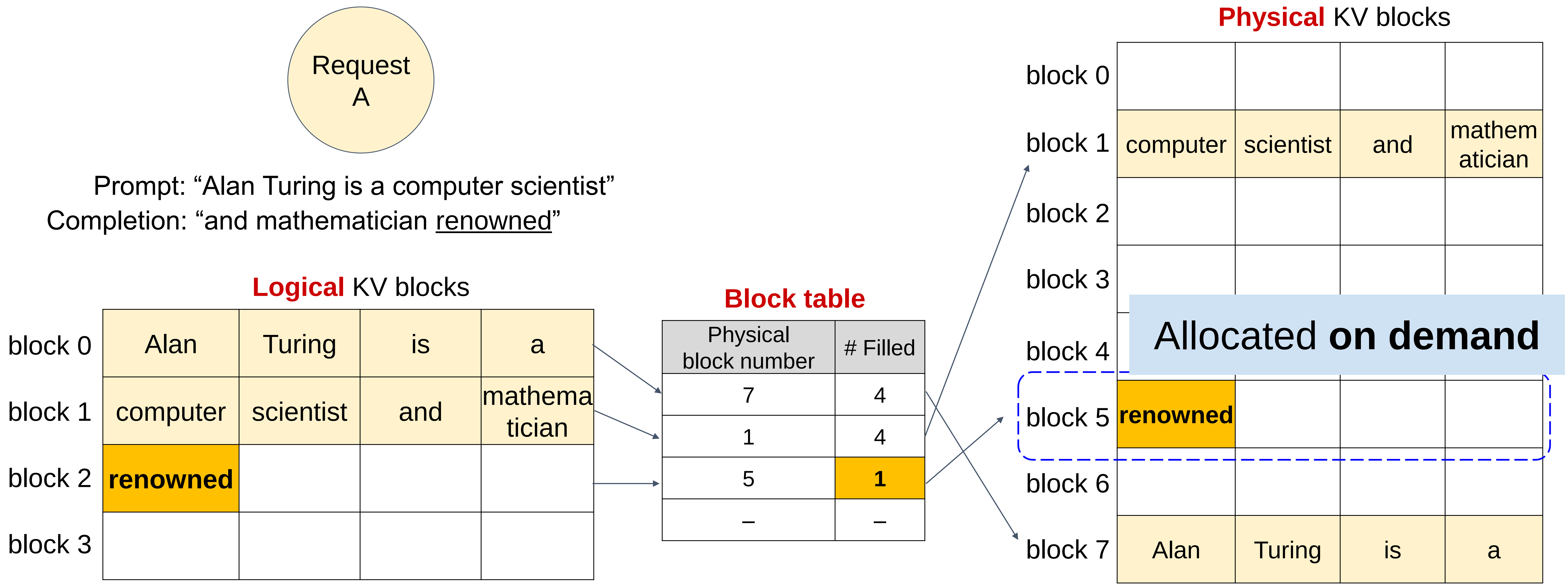
Block table

Physical block number	# Filled
7	4
1	**4**
–	–
–	–

Physical KV blocks

block 0				
block 1	computer	scientist	and	**mathem atician**
block 2				
block 3				
block 4				
block 5				
block 6				
block 7	Alan	Turing	is	a

Memory Management with PagedAttention



Memory Efficiency of PagedAttention

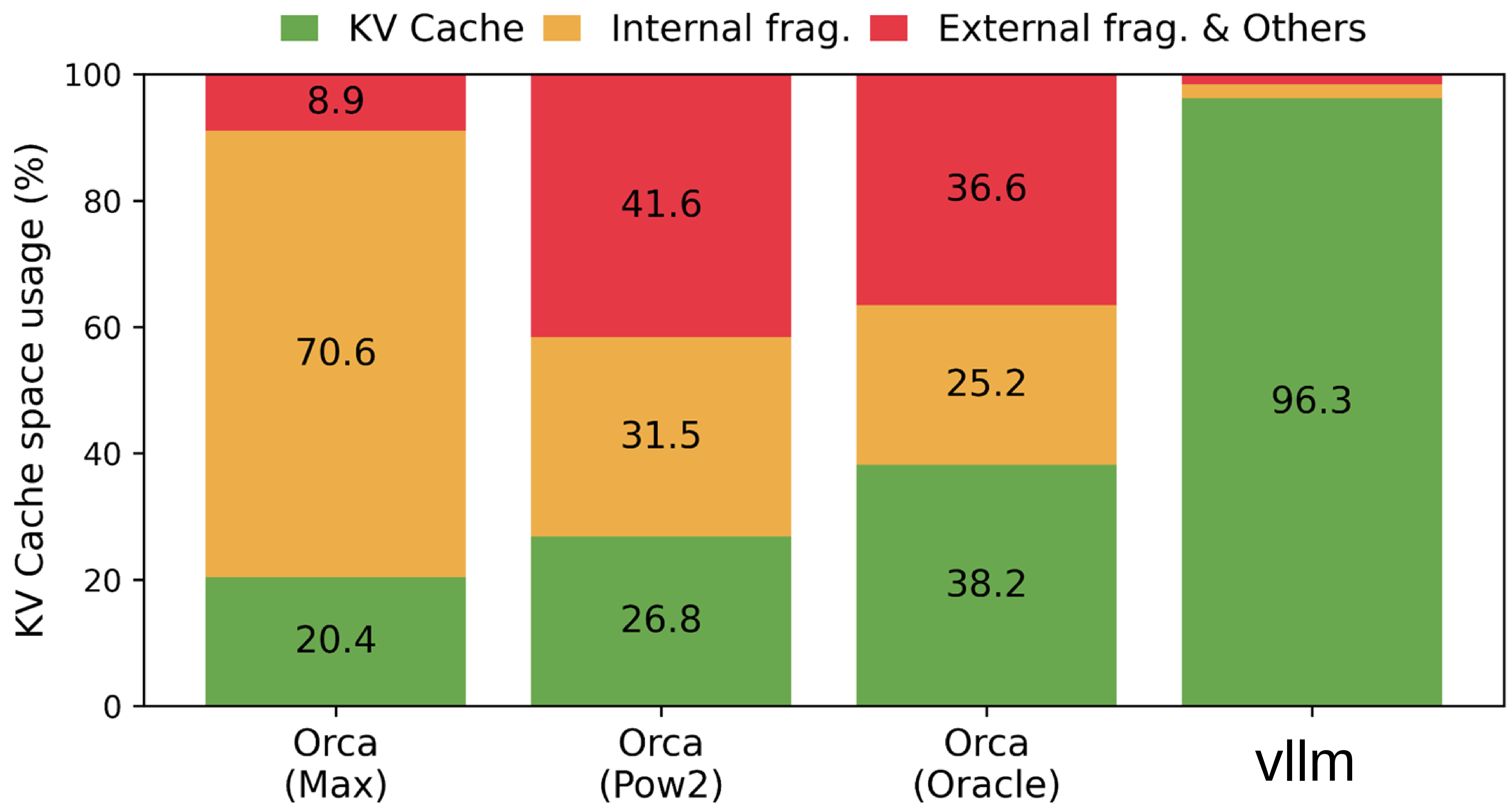
Minimal internal fragmentation

- Only happens at the last block of a sequence
- # wasted tokens / seq < block size

No external fragmentation

Alan	Turing	is	a
computer	scientist	and	mathematician
renowned			

Internal fragmentation



A few Important Problems (will be HW3)

- How to estimate the number of parameters of an LLM?
  - Embedding: position + word
  - Transformers layers:
    - attention Wq,Wk,Wv
    - MLP: up project, down project
    - Layernorm parameters
- How to estimate the flops needed to train an LLM?
- How to estimate the memory needed to train a transformer?