https://haoailab.com/cse291-s26/

CSE/DSC 291: Deep Learning Systems
Spring 2026

LLM, diffusion, and case studies

Optimizations and Parallelization

Basics

Logistics

- PA3 was released, enjoy!

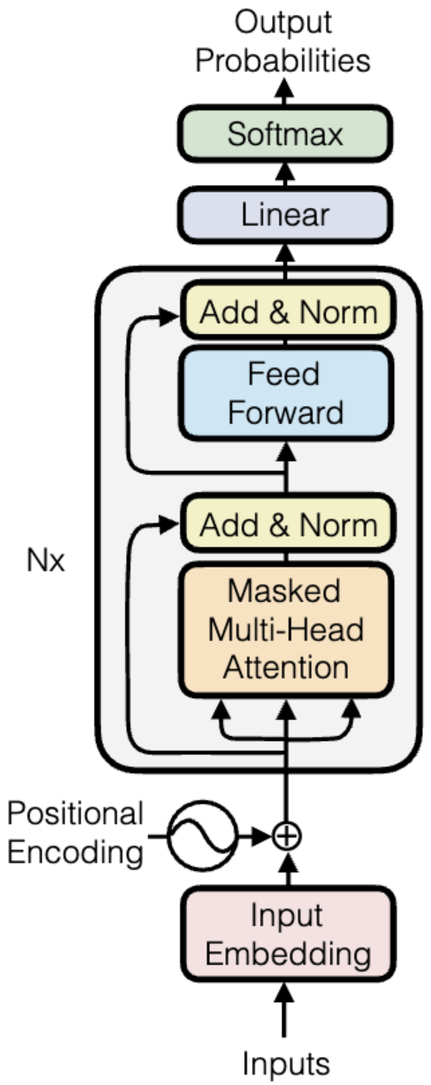
- Next Tuesday (Zoom): Flash attention and training optimization

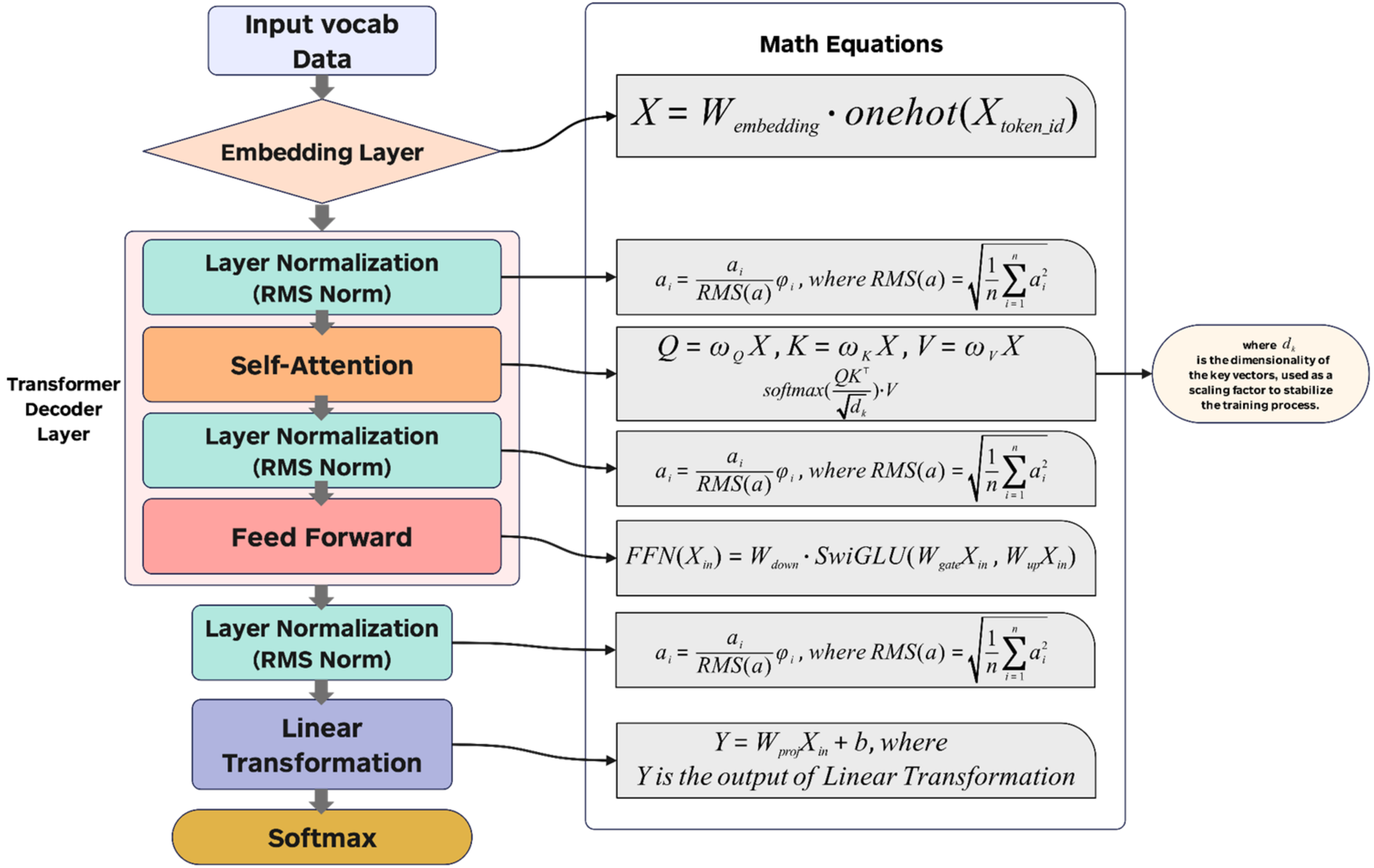
- Next Thursday (Guest Lecture): Zihao Ye -- flashinfer creator

Connecting the Dots: Compute/Comm characteristic of LLMs

Key characteristics: compute, memory, communication

- calculate the number of parameters of an LLM?
- calculate the flops needed to train an LLM?
- calculate the memory needed to train an LLM?




Input vocab Data
Embedding Layer
Transformer Decoder Layer
Layer Normalization (RMS Norm)
Self-Attention
Layer Normalization (RMS Norm)
Feed Forward
Layer Normalization (RMS Norm)
Linear Transformation
Softmax
Math Equations
$X = W_{embedding} \cdot onehot(X_{token\_id})$
$a_i = \frac{a_i}{RMS(a)} \varphi_i$, where $RMS(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}$
$Q = \omega_Q X, K = \omega_K X, V = \omega_V X$
$softmax(\frac{QK^T}{\sqrt{d_k}}) \cdot V$
where d_k is the dimensionality of the key vectors, used as a scaling factor to stabilize the training process.
$a_i = \frac{a_i}{RMS(a)} \varphi_i$, where $RMS(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}$
$FFN(X_{in}) = W_{down} \cdot SwiGLU(W_{gate} X_{in}, W_{up} X_{in})$
$a_i = \frac{a_i}{RMS(a)} \varphi_i$, where $RMS(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}$
$Y = W_{proj} X_{in} + b$, where
Y is the output of Linear Transformation

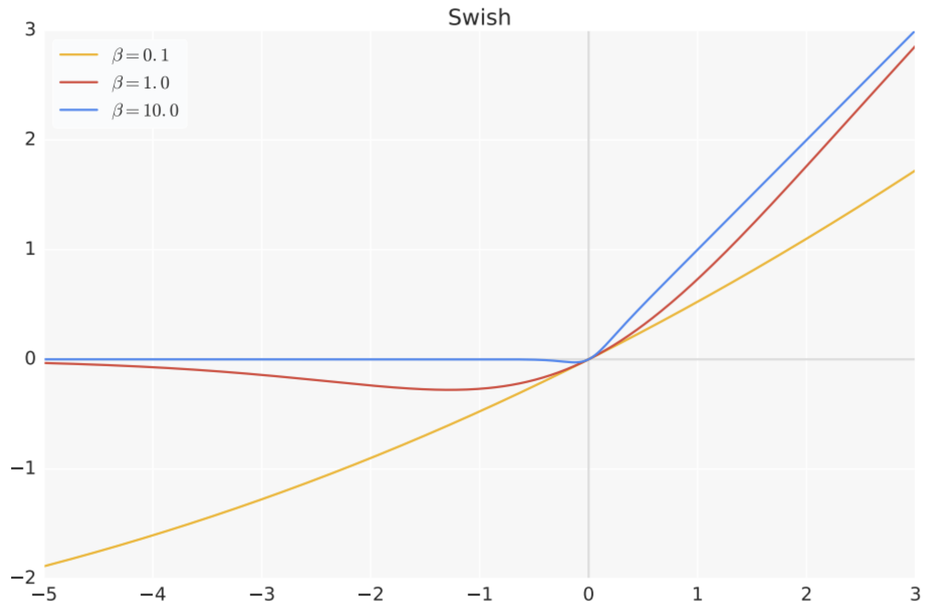
Feed Forward SwiGLU

The general formula for SwiGLU is:

$$\mathrm{SwiGLU}(x) = \mathrm{Swish}(xW_1 + b_1) \odot (xW_2 + b_2)$$

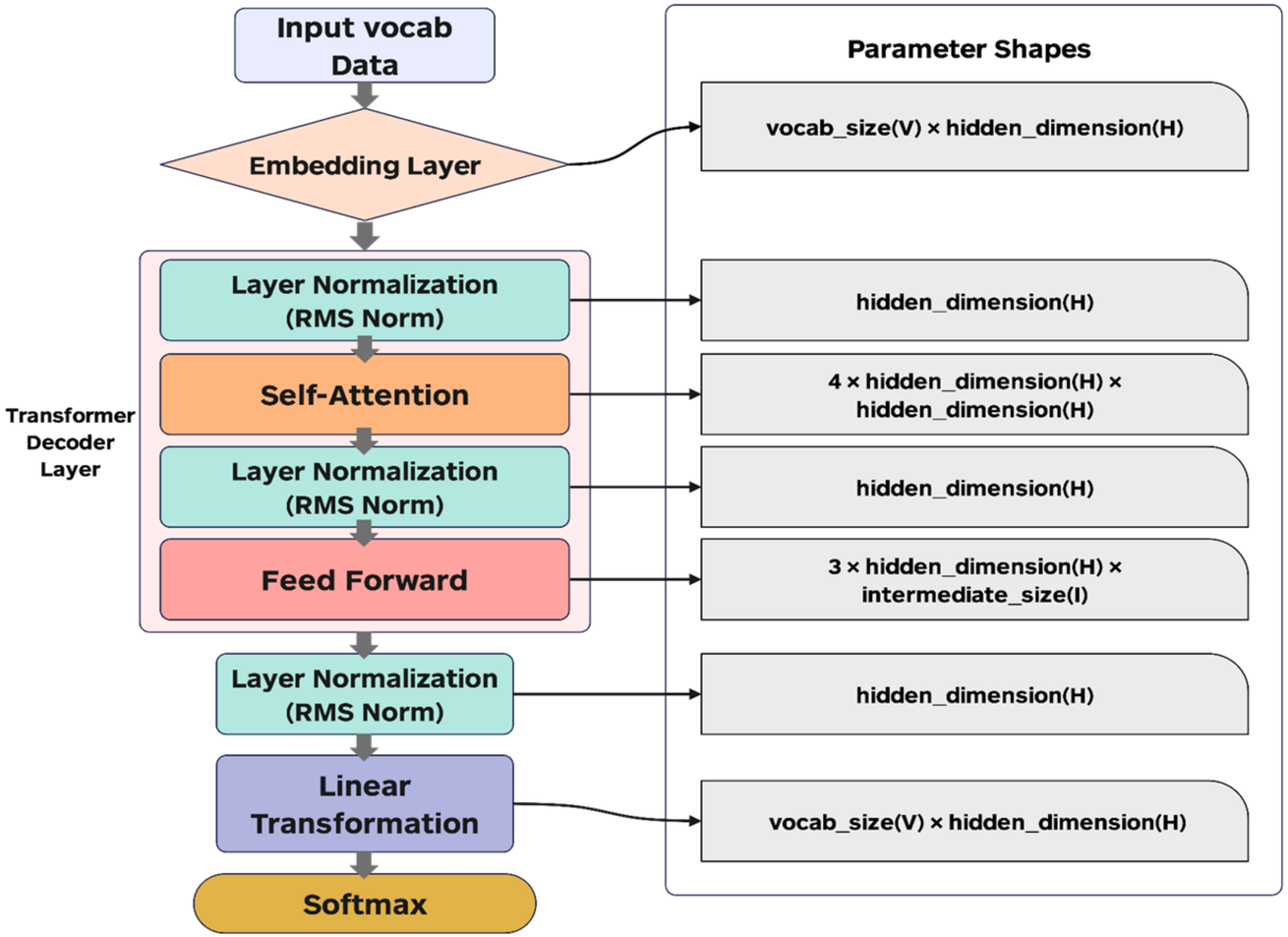
Swish is the activation function applied to one branch, defined as:

$$\mathrm{Swish}(z) = z \cdot \sigma(z)$$

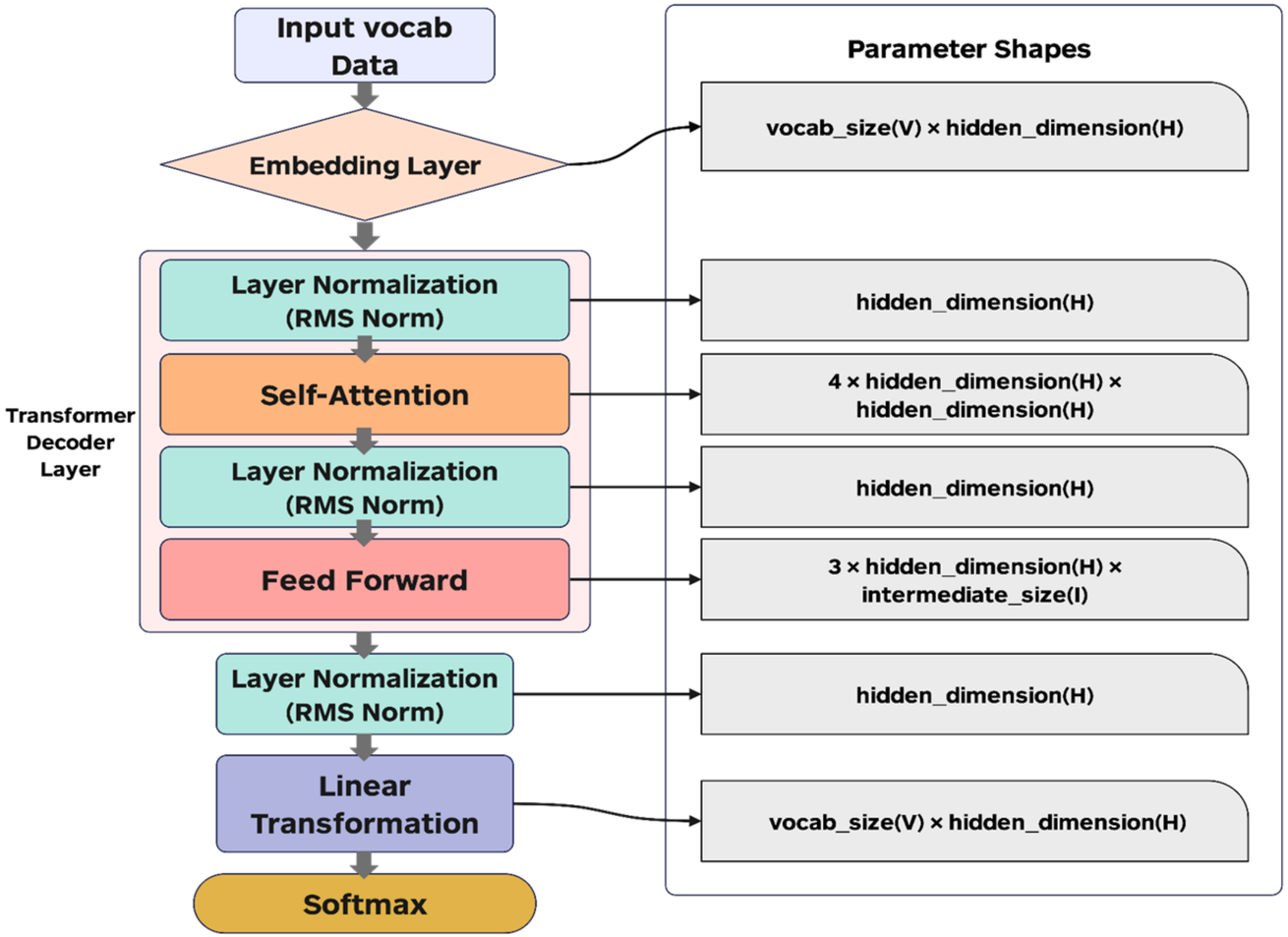


- SwiGLU helps the model capture more complex patterns by selectively gating information
- Swish is smoother than traditional activations ReLU

Summary


Input vocab Data
Embedding Layer
Transformer Decoder Layer
Layer Normalization (RMS Norm)
Self-Attention
Layer Normalization (RMS Norm)
Feed Forward
Layer Normalization (RMS Norm)
Linear Transformation
Softmax
Parameter Shapes
vocab_size(V) × hidden_dimension(H)
hidden_dimension(H)
4 × hidden_dimension(H) × hidden_dimension(H)
hidden_dimension(H)
3 × hidden_dimension(H) × intermediate_size(I)
hidden_dimension(H)
vocab_size(V) × hidden_dimension(H)

Scaling Up: Where is the Potential Bottleneck?

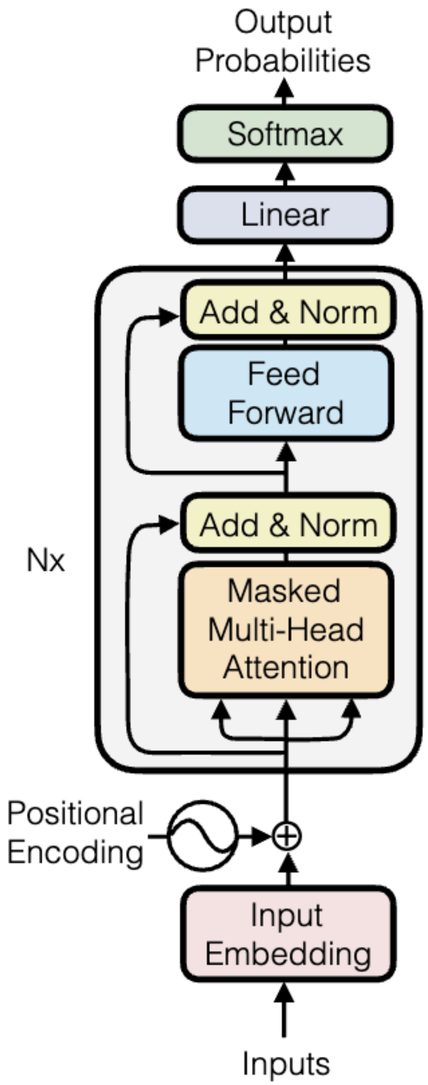


In PA3, you will implement this function😉

Connecting the Dots: Compute/Comm characteristic of LLMs

Key characteristics: compute, memory, communication

- calculate the number of parameters of an LLM?
- calculate the flops needed to train an LLM?
- calculate the memory needed to train an LLM?

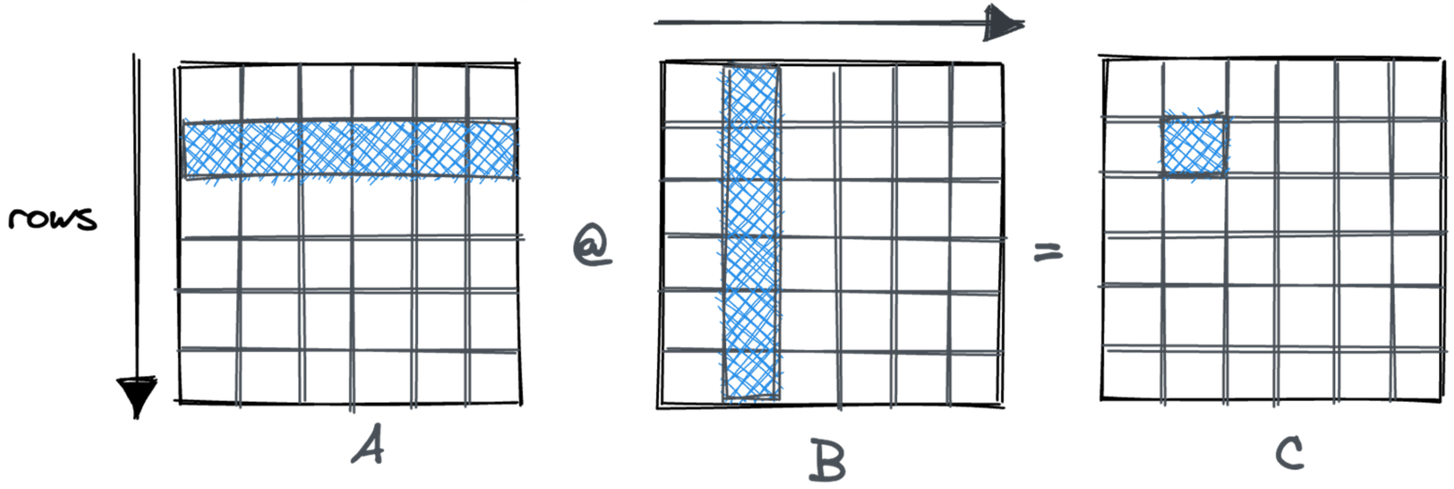


Estimate the Compute: FLOPs

The FLOPs for multiplying two matrices of dimensions m×n and n×h can be calculated as follows:

$$\text{FLOPs} = m \times h \times (2n - 1)$$

So the total number of FLOPs is roughly $\text{FLOPs} \approx 2m \times n \times h$



LLama 2 7B Flops Forward Calculation (Training)

Hyperparameters:

Batch size: b

Sequence length: s

The number of attention heads: n

Hidden state size of one head: d

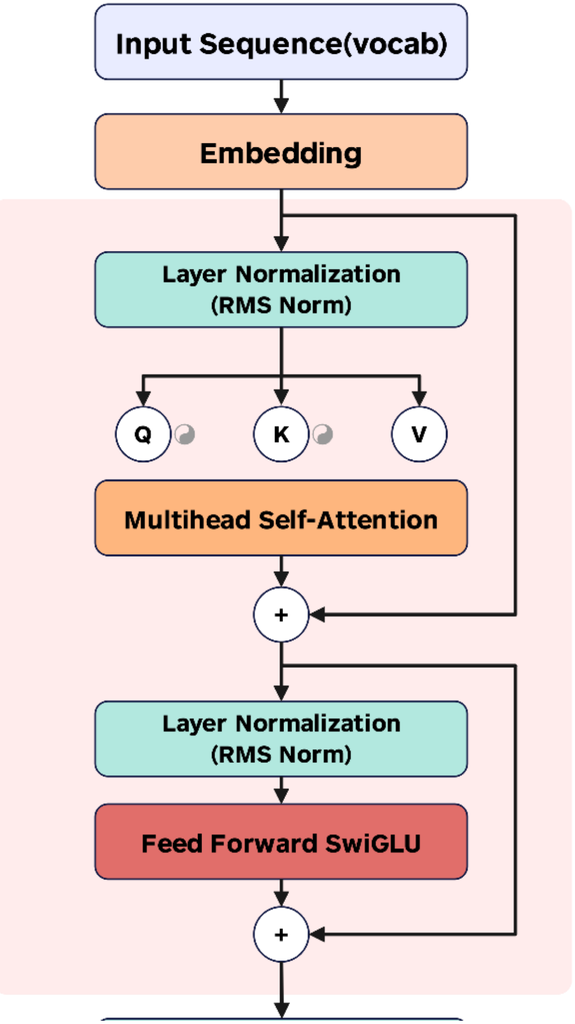
Hidden state size: h (h = n * d)

SwiGLU proj dim: i

Vocab size: v

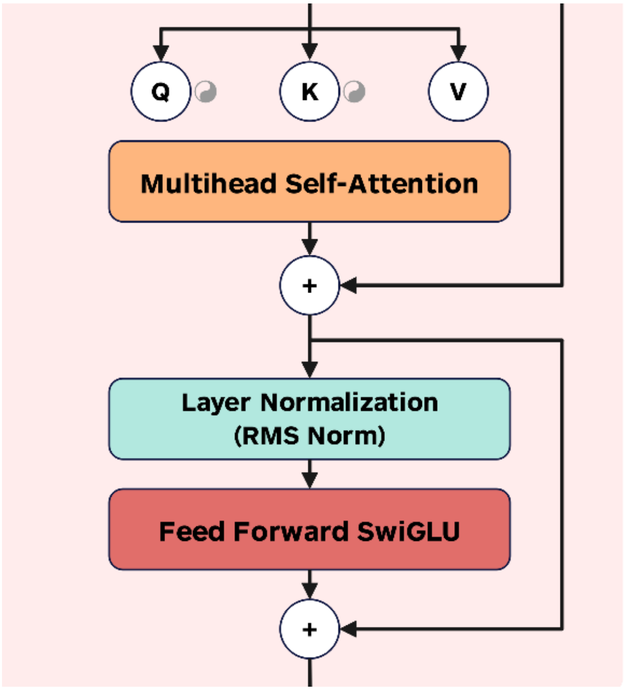
Input:	Output Shape:	FLOPs
X	(b, s, h)	0
Self Attention:		
XW_Q, XW_K, XW_V	(b, s, h)	$3 * 2bsh^2$
RoPE	(b, n, s, d)	$3bsnd$
$P = Softmax(QK^T/\sqrt{d})$	(b, n, s, s)	$2bs^2nd + 3bs^2n$
PV	(b, n, s, d)	$2bs^2nd$
AW_O	(b, s, h)	$2bsh^2$
Residual Connection:	(b, s, h)	bsh

Batch size: b
Sequence length: s
# of attention heads: n
Hidden state dim of one head: d
Hidden state dim: h



Batch size: b
Sequence length: s
Hidden state dim: h
SwiGLU proj dim: i

Output from Self Attn:	Output Shape:	FLOPs
X	(b, s, h)	
Feed-Forward		0
SwiGLU:	(b, s, i)	2 * 2bshi
XW_{gate}, XW_{up}	(b, s, i)	4bsi
Swish Activation	(b, s, i)	bsi
Element-wise *	(b, s, h)	2bshi
XW_{down}		4bsh + 2bs



RMS Norm:

$$\text{SwiGLU}(x) = \text{Swish}(xW_1 + b_1) \odot (xW_2 + b_2)$$

1. **Calculate Root Mean Square:**
   - $\text{RMS}(x) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}$
2. **Normalize:**
   - $\text{RMSNorm}(x) = \frac{x}{\text{RMS}(x)+\epsilon} \cdot \gamma$

LLama 2 7B Flops Forward (Training)

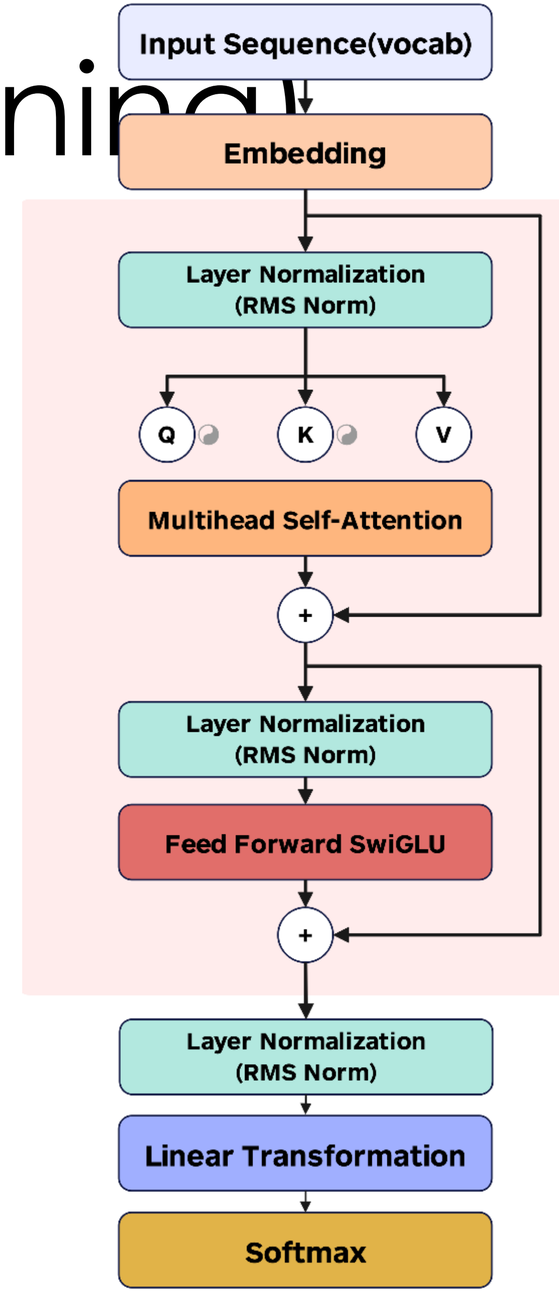
Total Flops ≈ #num_layers * (Attention block + SwiGLU block)

+ Prediction head

= #num_layers * ($6bsh^2 + 4bs^2h + 3bs^2n + 2bsh^2$)

+ #num_layers (6bshi)

+ 2 bshv



LLama 2 7B Flops Forward Calculation (Training)

Hyperparameters:

Batch size: b=1

Sequence length: s=4096

The number of attention heads: n=32

Hidden state size of one head: d=128

Hidden state size: h =4096

SwiGLU proj dim: i=11008

Vocab size: v=32000

The number of layers: N=32

$$\text{Total Flops} \approx N * (6bsh^2 + 4bs^2h + 3bs^2n + 2bsh^2)$$

$$+ N\ (6bshi)$$

$$+\ 2\ bshv$$

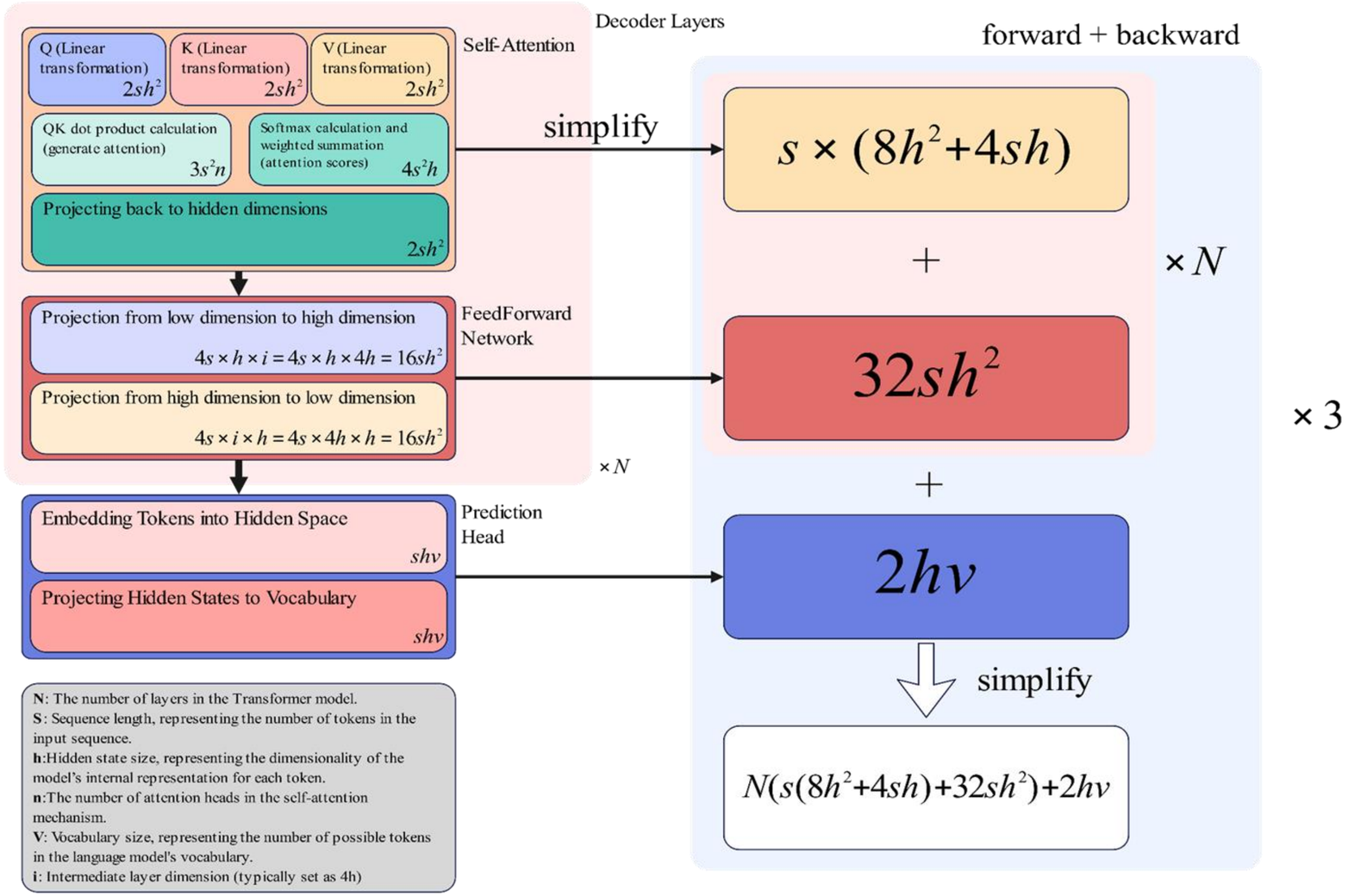
$$\approx 63\ \text{TFLOPs}$$

Flops Distribution

Training Computational Costs Breakdown:

- **Total Training TeraFLOPs:** 192.17 TFLOPs
- **FLOP Distribution by Layer:**
  - **Embedding Layer:** 1.676%
  - **Normalization:** 0.007%
  - **Residual:** 0.003%
  - **Attention:** 41.276%
  - **MLP (Multi-Layer Perceptron):** 55.361%
  - **Linear:** 1.676%

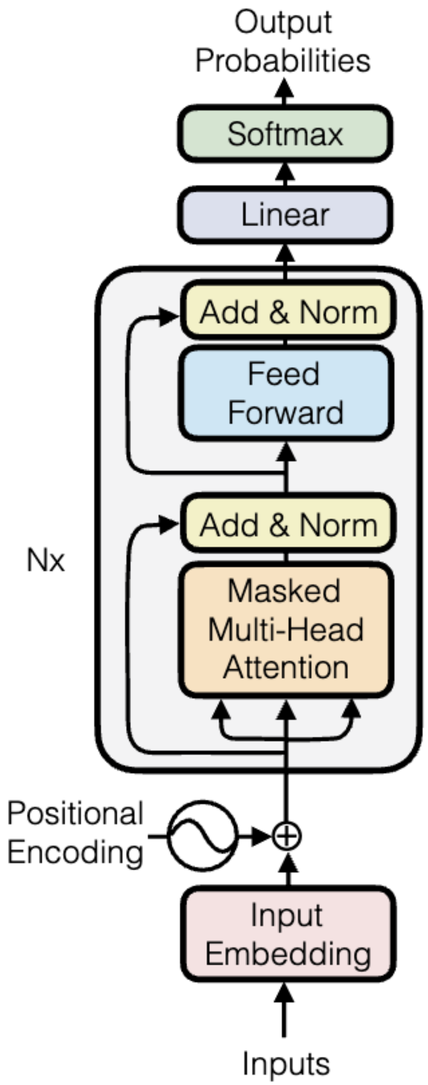
Scaling Up: Where is the Potential Bottleneck?



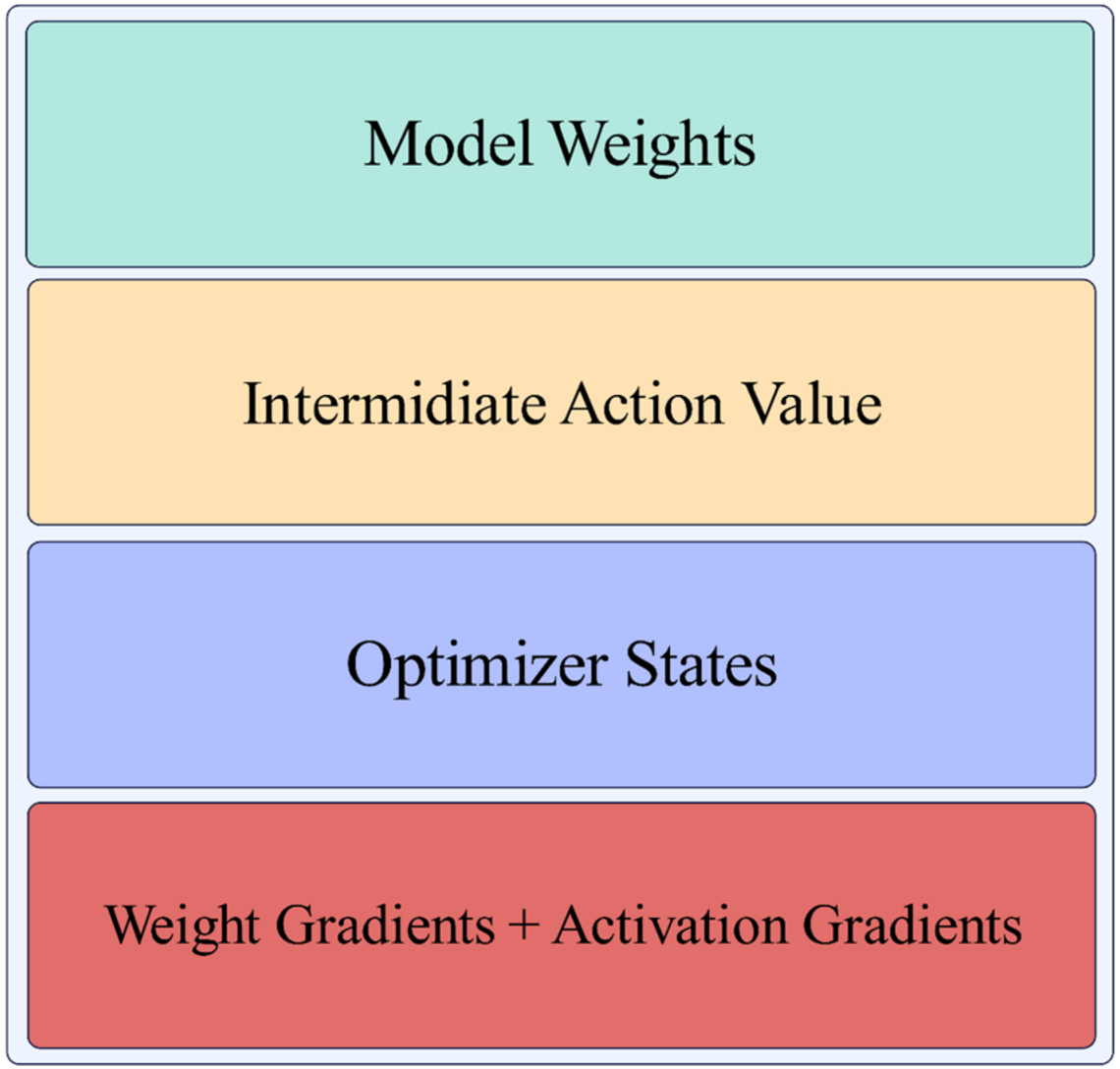
Connecting the Dots: Compute/Comm characteristic of LLMs

Key characteristics: compute, memory, communication

- calculate the number of parameters of an LLM?
- calculate the flops needed to train an LLM?
- calculate the memory needed to train an LLM?



Composition of Memory Usage (Training)



Large Language Models

- LLM Internals
- **Scaling Law**
- Serving and inference optimization

Some Observations

- compute is a function of: h, i, b
- #parameter is a function of: h, i
- Hence: compute correlates with #parameters
  - more parameters, more compute
  - more data, more compute (of course)
- Problem: we have limited compute ($)
- how should we allocate our limited resources:
  - Train models longer vs train bigger models?
  - Collect more data vs get more GPUs?

Motivation of Scaling Laws

- We want to know:
  - how large a model should we train…
  - How many data should we use…
  - To achieve a given performance…
  - Subject to a compute budget ($)?

How do we do that in traditional ML: data scaling law

Input: $x_1 \ \ldots x_n \ \sim N(\mu, \sigma^2)$

Task: estimate the average as $\hat{\mu} = \frac{\sum_i x_i}{n}$

What's the error? By standard arguments..

$$\mathrm{E}[(\hat{\mu} - \mu)^2] = \frac{\sigma^2}{n}$$

This is a scaling law!!

$$\log(Error) = -\log n + 2\log\sigma$$

More generally, any polynomial rate $1/n^{\alpha}$ is a scaling law

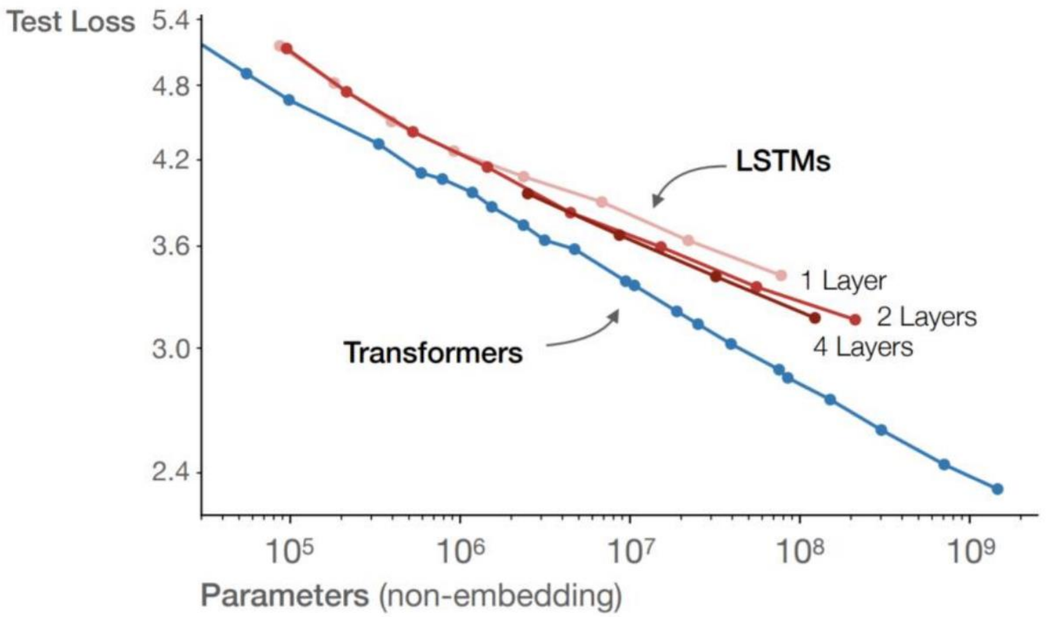
- Can we do this for transformers LLMs?
- **Unfortunately NO**

Think in this way

Mathematics vs. Physics

Transformers vs LSTMs

- Q: Are transformers better than LSTMs?
  - Brute force way: spend tens of millions to train a LSTM GPT-3
- Scaling law way:



[Kaplan+ 2021]

The Scaling law way: Physics Way

- Approach:
  - Train a few smaller models
  - Establish a scaling law (LSTM vs. transformers)
  - Select optimal hyperparam based on the scaling law prediction.

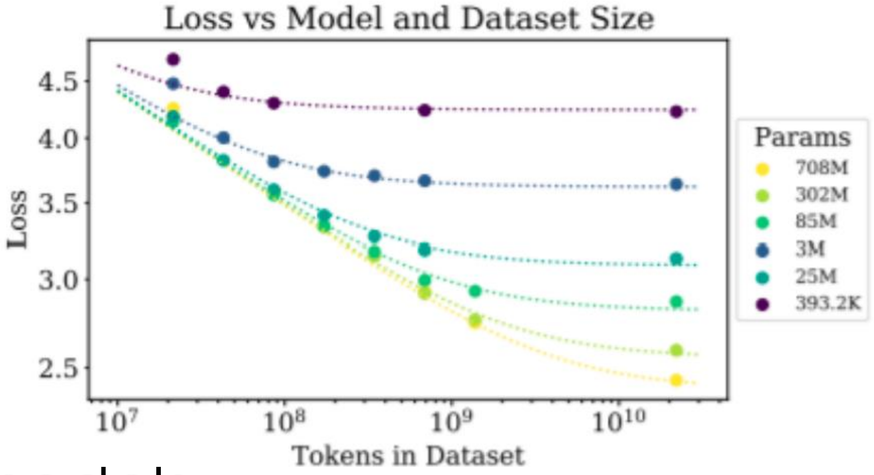
- Rationale
  - The effect of hyperparameters on big LMs can be predicted before training!
    - Optimizer choice
    - Model Depth
    - Arechitecture choice

Back to our problem:

- how large a model should we train…
- How many data should we use…
- To achieve a given performance…
- Subject to a compute budget?

- Approach: estimate a law between model size data joint scaling

Model size data joint scaling



- Do we need more data or bigger models?
  - Clearly, lots of data is wasted on small models
- Joint data-model scaling laws describe ho the two relate

From Rosenfeld+ 2020,

$$Error = n^{-\alpha} + m^{-\beta} + C$$

From Kaplan+ 2021

$$Error = [m^{-\alpha} + n^{-1}]^{\beta}$$

Provides surprisingly good fits to model-data joint error.

Compute Trade-offs

- Q: what about other resources? Compute vs. performance?

- For a fixed compute budget…
  - Big models that's undertrained vs small model that's well trained?
  - Solving the following optimization?

$$N_{opt}(C), D_{opt}(C) = \underset{N,D \text{ s.t. } \mathrm{FLOPs}(N,D)=C}{\operatorname{argmin}} L(N, D).$$

Approach: empirical scaling law

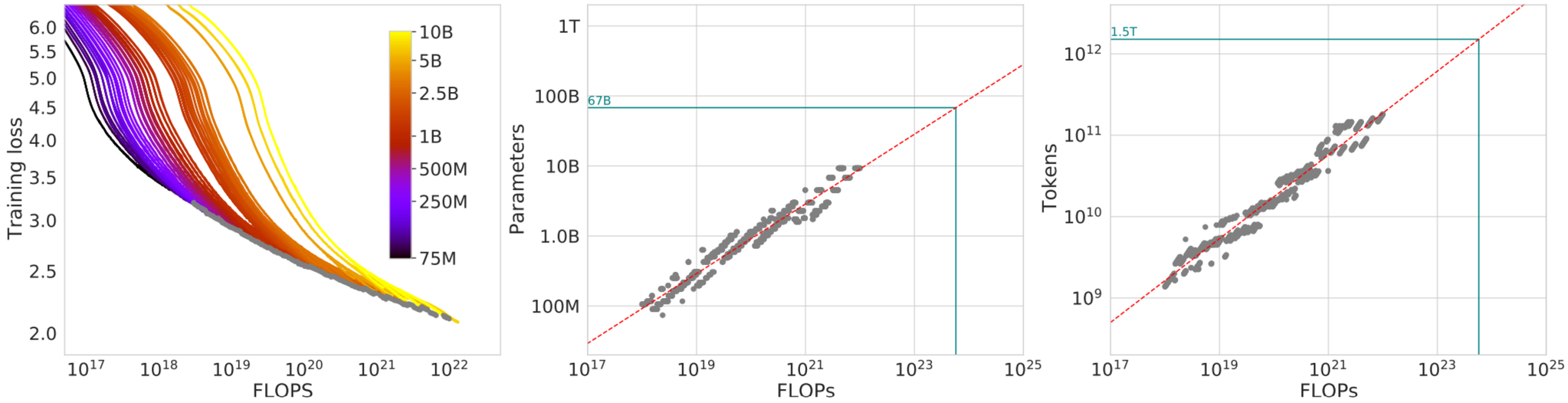


Figure 2 | **Training curve envelope.** On the **left** we show all of our different runs. We launched a range of model sizes going from 70M to 10B, each for four different cosine cycle lengths. From these curves, we extracted the envelope of minimal loss per FLOP, and we used these points to estimate the optimal model size (**center**) for a given compute budget and the optimal number of training tokens (**right**). In green, we show projections of optimal model size and training token count based on the number of FLOPs used to train *Gopher* (5.76×10^{23}).

Today's SoTA Law

$$L(N, D) = \frac{406.4}{N^{0.34}} + \frac{410.7}{D^{0.29}} + 1.69$$

Summary

- Scaling law: the physics of ML
- Scaling law marks a new era of ML research:
  - Rigorous theoretical analysis -> empirical laws
  - Exploration of different model architectures -> Scaling transformers
  - Due to scaling law: ML systems become essential

PA3: Q3

You already know:

- How to estimate the number of parameters of an LLM?
- How to estimate the flops needed to train an LLM?
- How to estimate the memory needed to train a transformer?

- We will give you a scaling law and compute budget
  - Task: design your optimal LLM

Large Language Models

- LLM Internals
- Scaling Law
- **Serving and inference optimization**

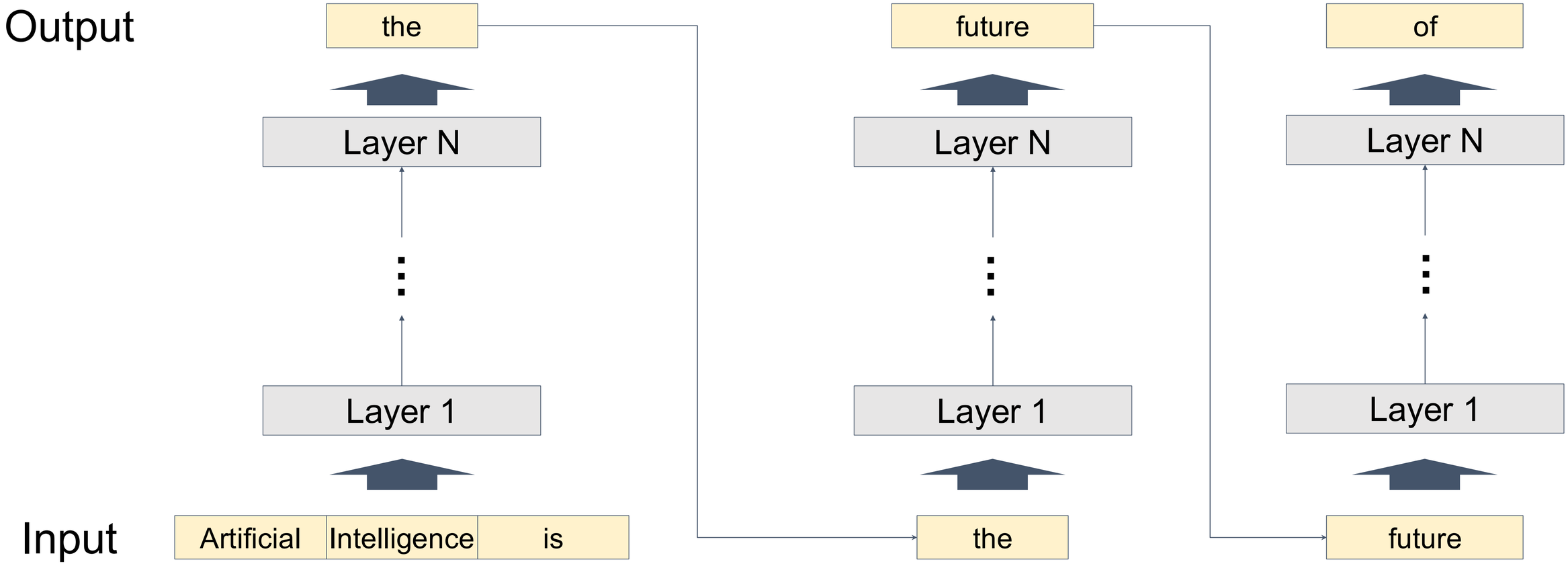
Recap: Next Token Prediction

Probability("San Diego has very nice weather")
= P("San Diego") P("has"|"San Diego")P("very"|"San Diego has")P("city"|…)…P("weather"|…)

$$\text{Max}\,Prob(x_{1:T}) = \prod_{t=1}^{T} P(x_{t+1}|x_{1\ldots t})$$

This is model we got – capable of "predicting the next token".

Inference process of LLMs



Repeat until the sequence

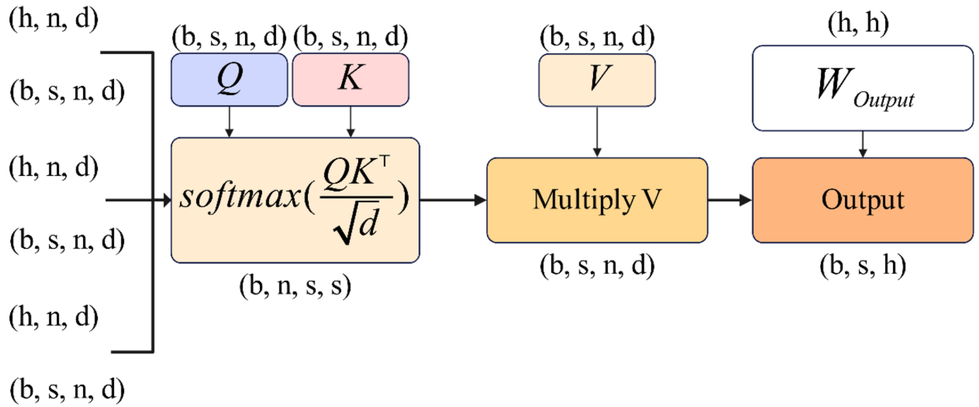
- Reaches its pre-defined maximum length (e.g., 2048 tokens)
- Generates certain tokens (e.g., “<|*end of sequence*|>”)

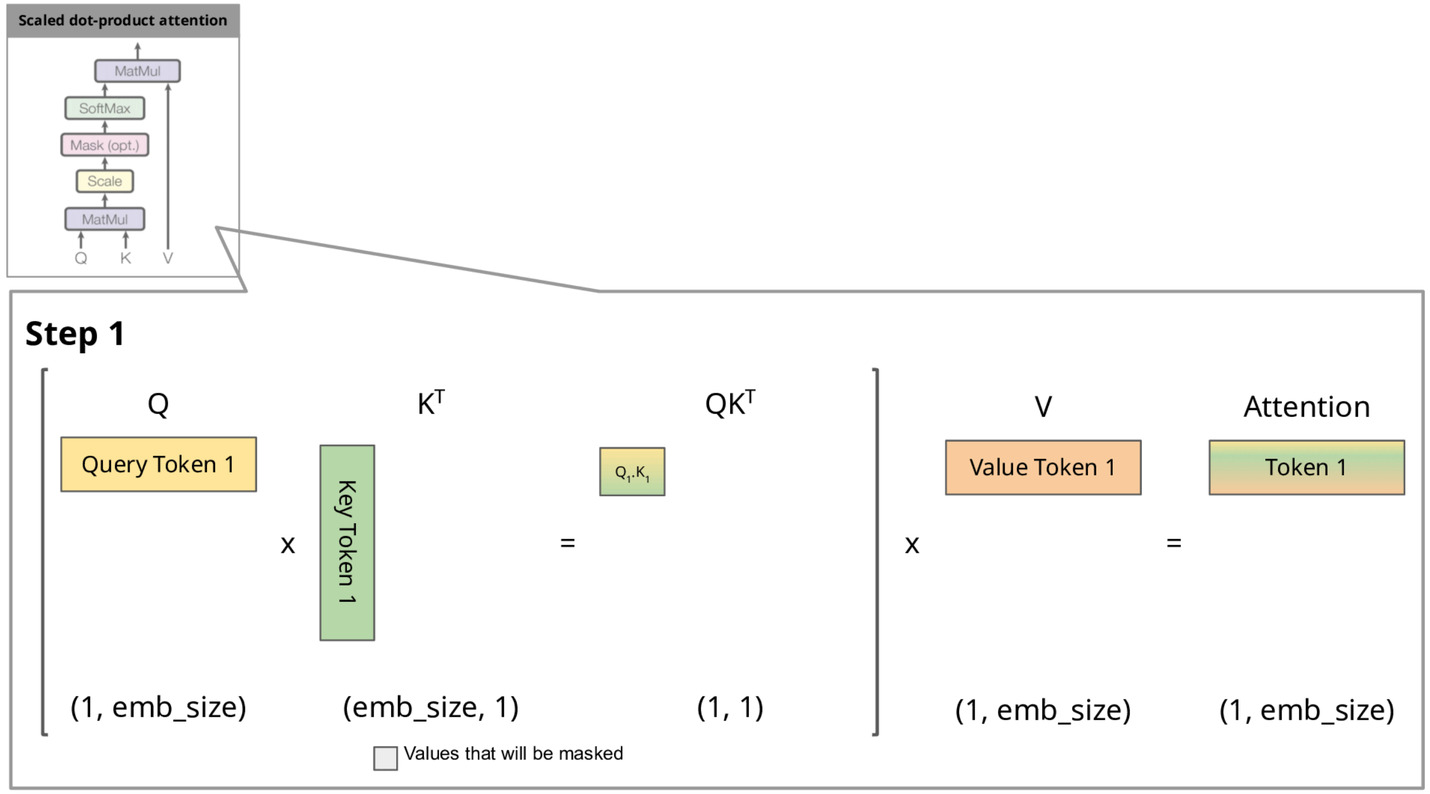
Generative LLM Inference: Autoregressive Decoding

- 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

- Key-value cache:
  - Save attention keys and values for the following iterations to avoid recomputation
  - what is KV cache essentially?

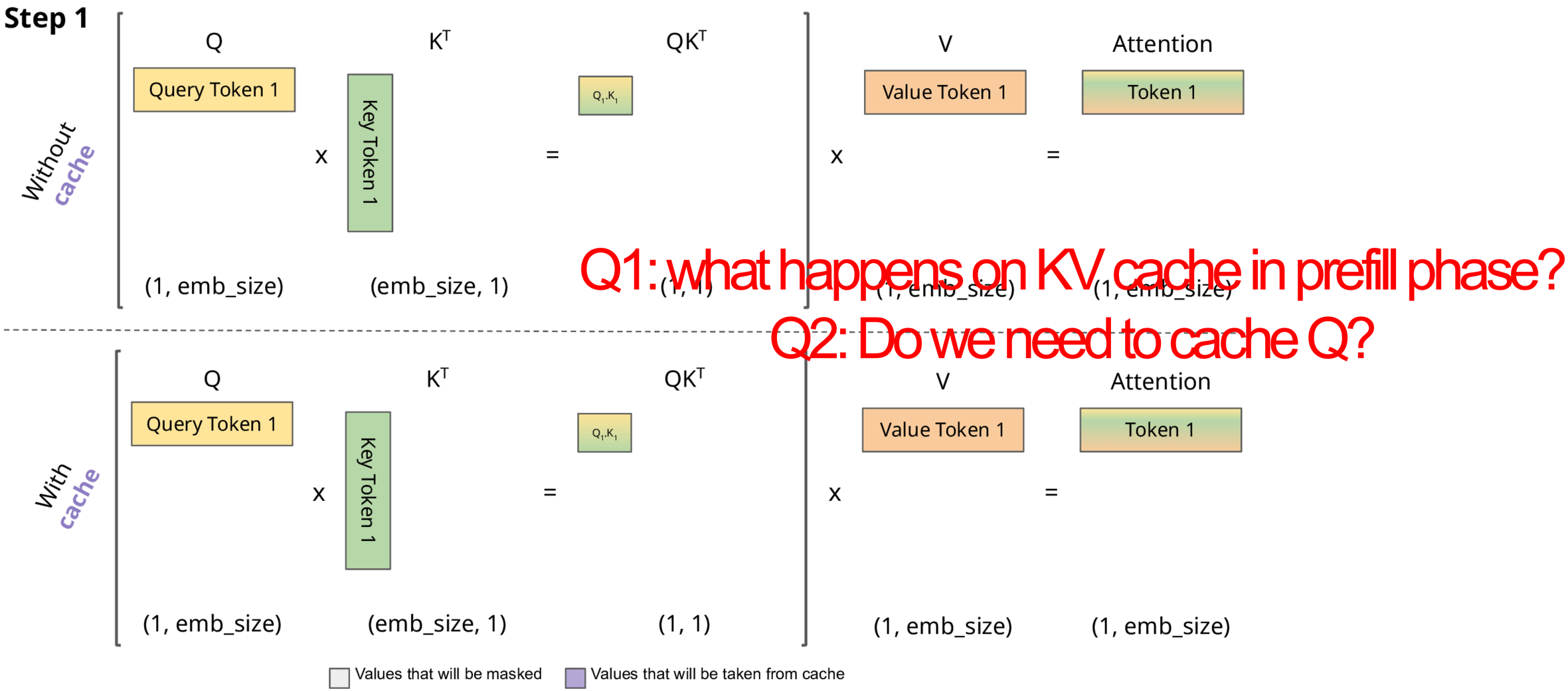
w/ KV Cache vs. w/o KV Cache





Zoom-in! (simplified without Scale and Softmax)

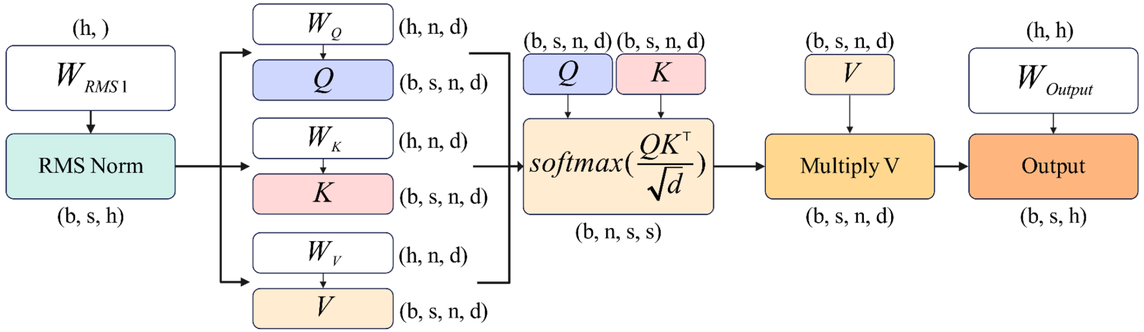
w/ KV Cache vs. w/o KV Cache

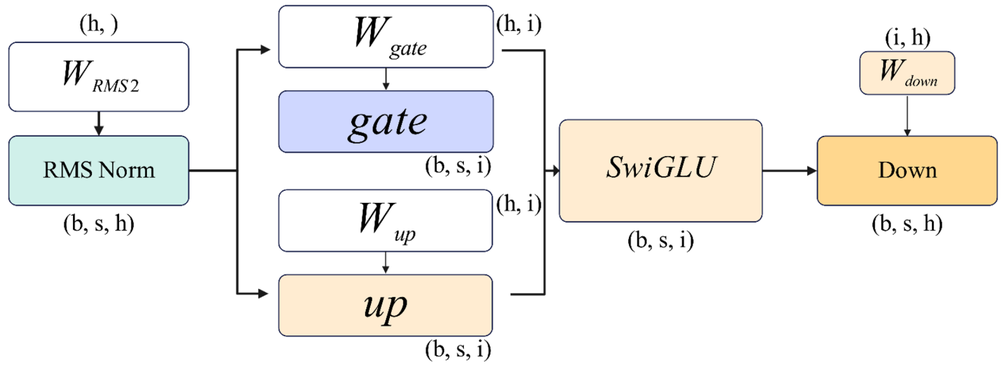


Q1: what happens on KV cache in prefill phase?

Q2: Do we need to cache Q?

Potential Bottleneck of LLM Inference?





- Compute:
  - Prefill: largely same with training
  - Decode: s = 1
- Memory
  - New: KV cache

Q? how about batch size b?

Serving vs. Inference

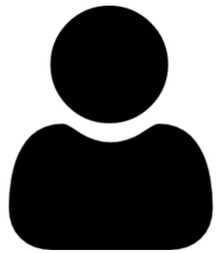
large b

Serving: many requests, online traffic, emphasize cost-per-query.

s.t. some mild latency constraints

emphasize **throughput**

b = 1

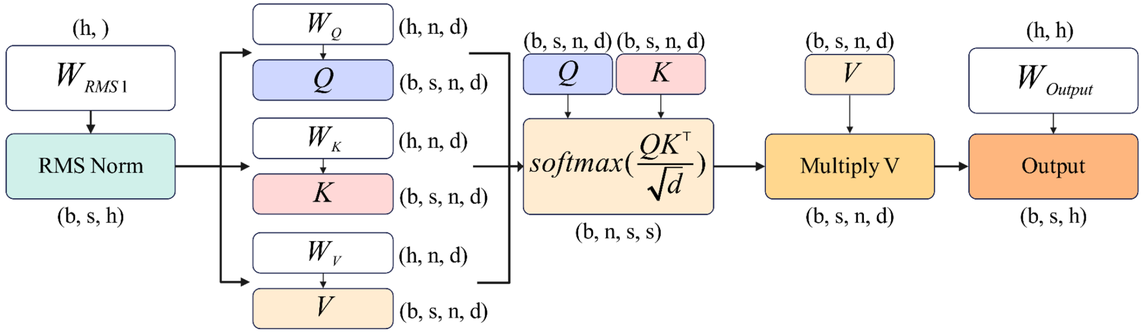


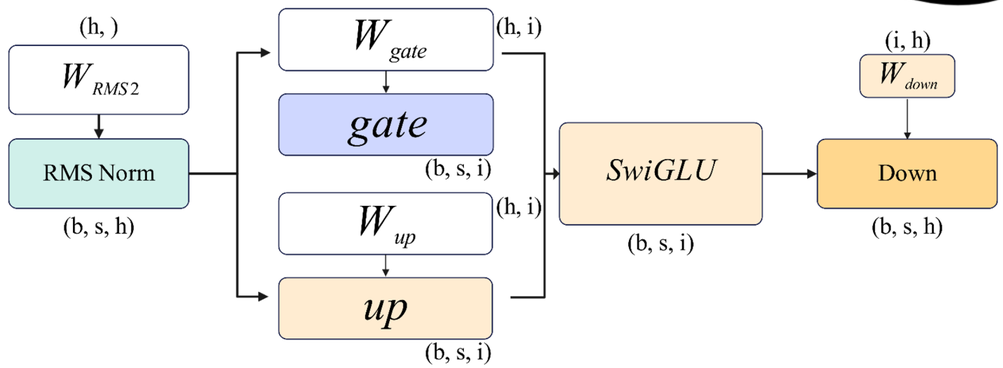
Inference: fewer request, low or offline traffic,

emphasize **latency**

large b

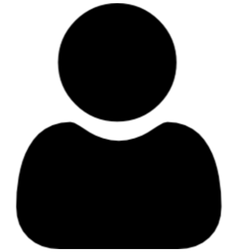
Potential Bottleneck of LLM Inference in Serving



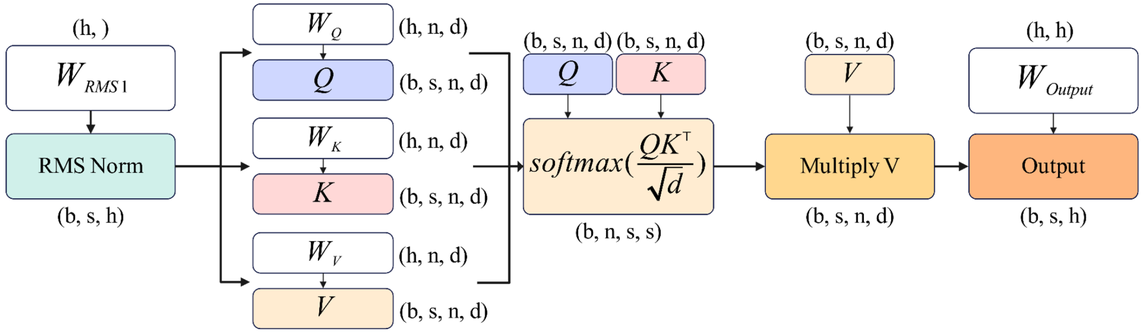


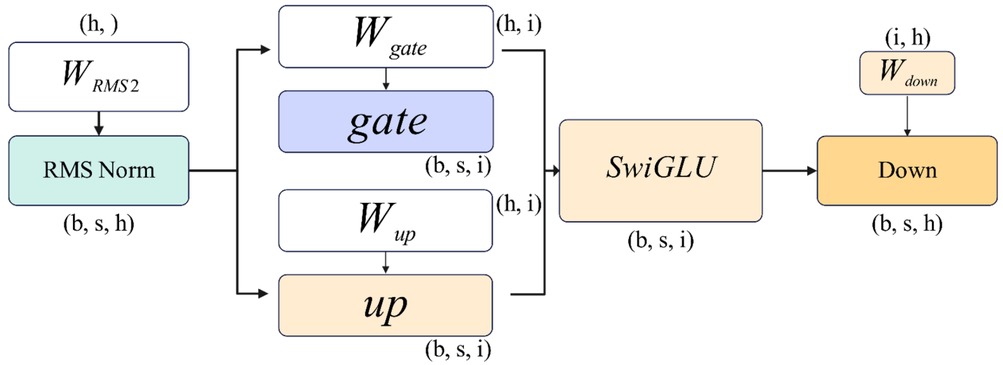
- Compute:
  - Prefill:
    - Different prompts have different length: how to batch?
  - Decode
    - Different prompts have different, unknown #generated tokens
    - s = 1, b is large
- Memory
  - New: KV cache
  - b is large -> KV is linear with b -> will KVs be large?

b=1



Potential Bottleneck of LLM Inference in Serving



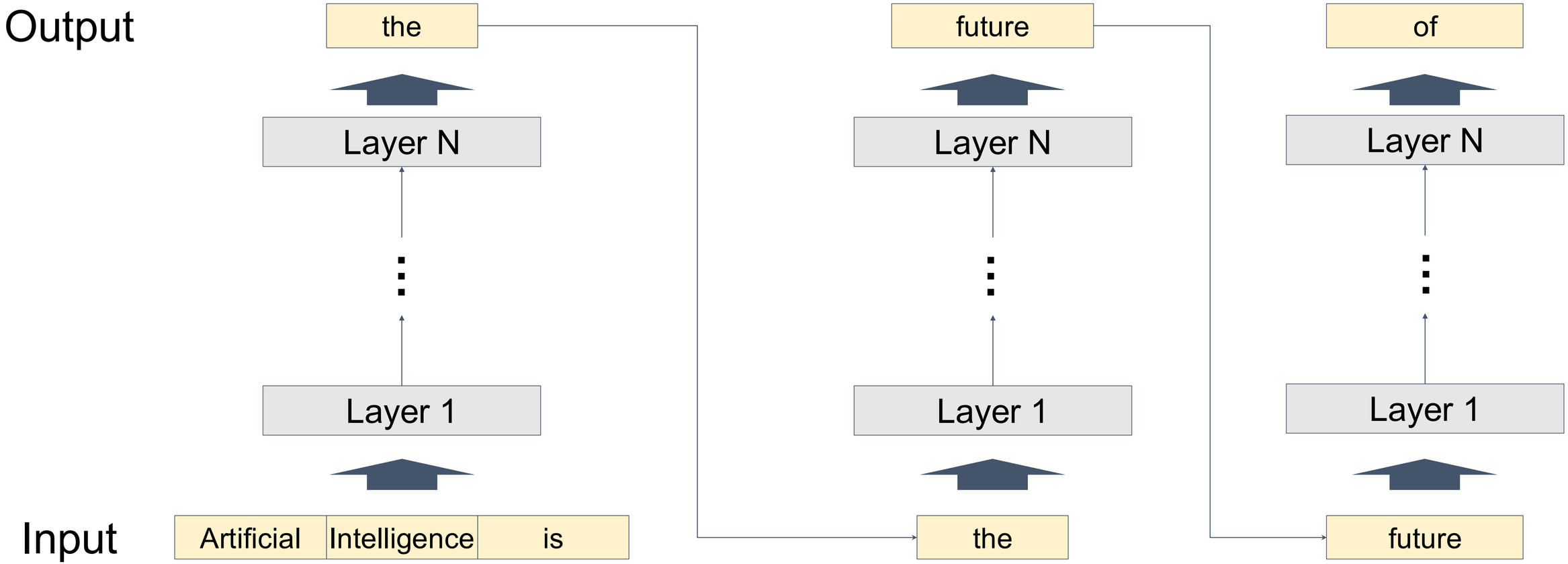


- Compute:
  - Prefill:
    - ~~Different prompts have different length: how to batch?~~
  - Decode
    - Different prompts have different, unknown #generated tokens
      - s = 1, b=1
- Memory
  - New: KV cache
    - ~~b =1 -> KV is linear with b -> will KVs be large?~~

Problems of bs = 1

max AI = #ops / #bytes

Recap: Inference process of LLMs

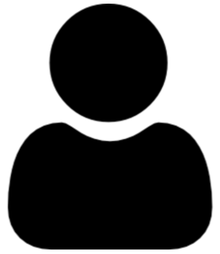


Repeat until the sequence

- Reaches its pre-defined maximum length (e.g., 2048 tokens)
- Generates certain tokens (e.g., “<|*end of sequence*|>”)

Problem of bs = 1

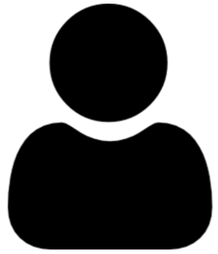
b=1



Latency = step latency * # steps

Speculative decoding reduces this, hence amortize the memory moving cost (but it may increase compute cost)

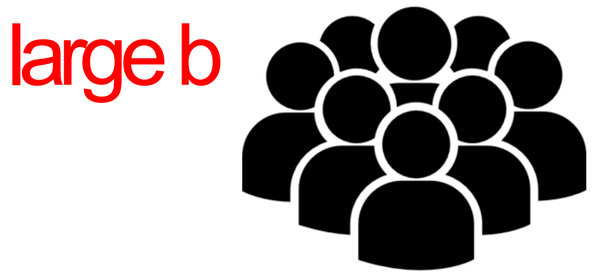
b=1



Large Language Models

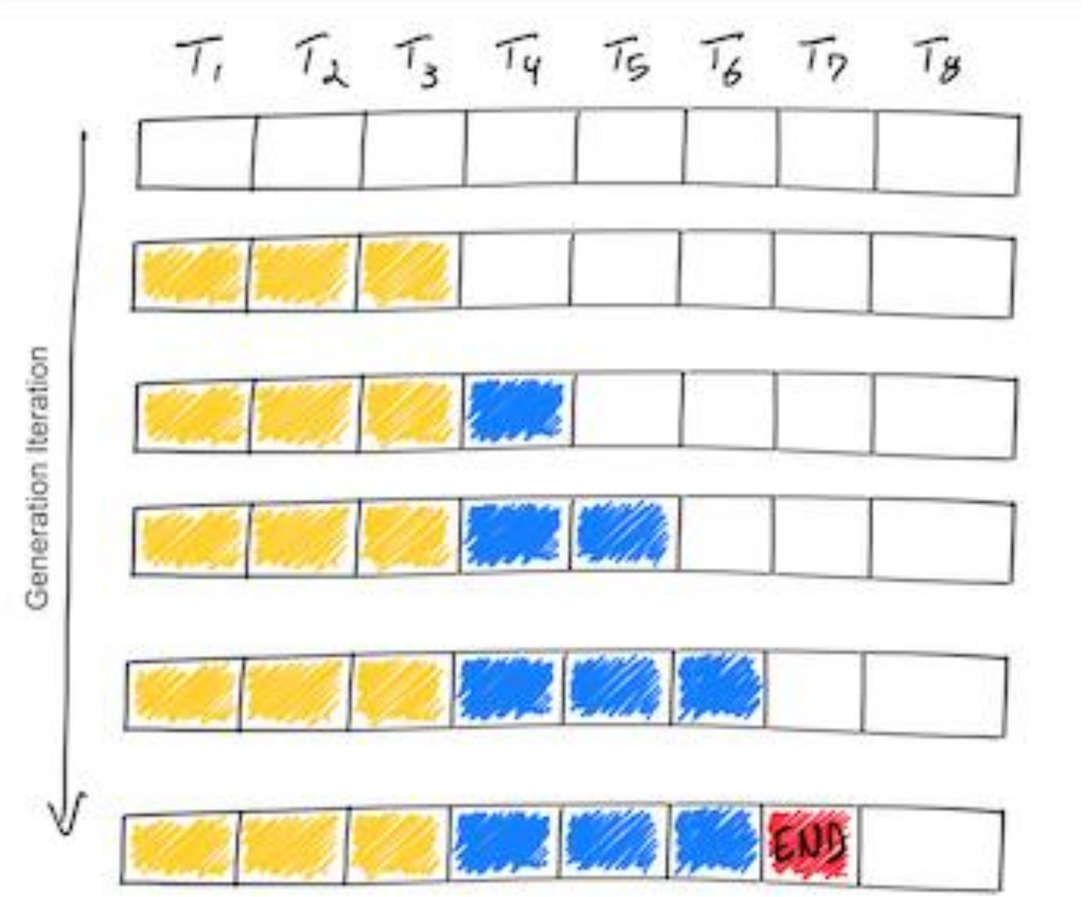
- LLM Internals
- **Scaling Law**
- Serving and inference optimization
  - Speculative decoding (question in your PA3)

Large Language Models

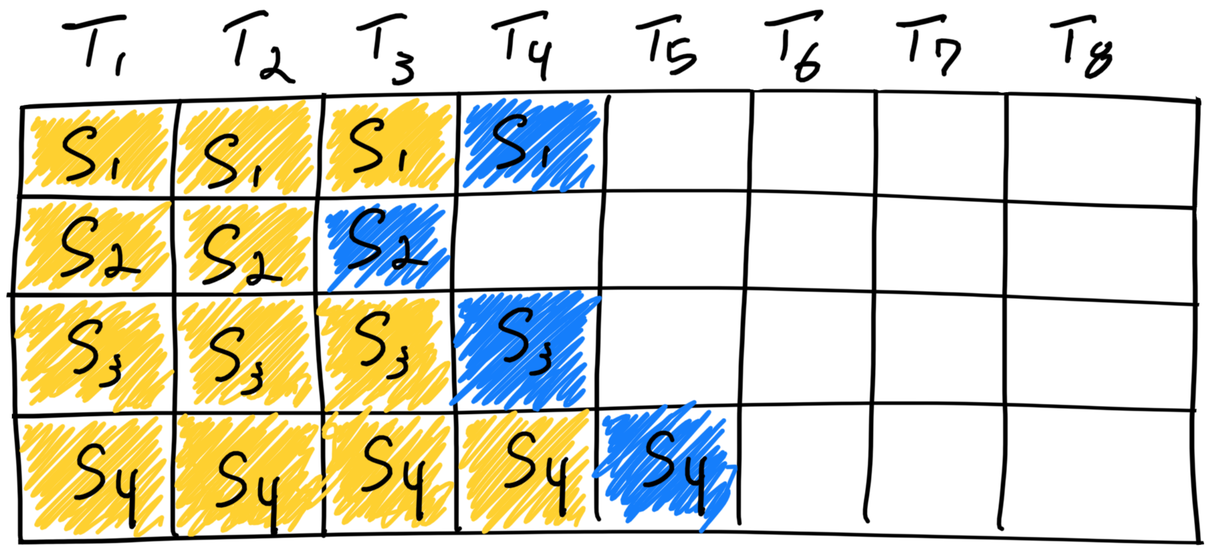


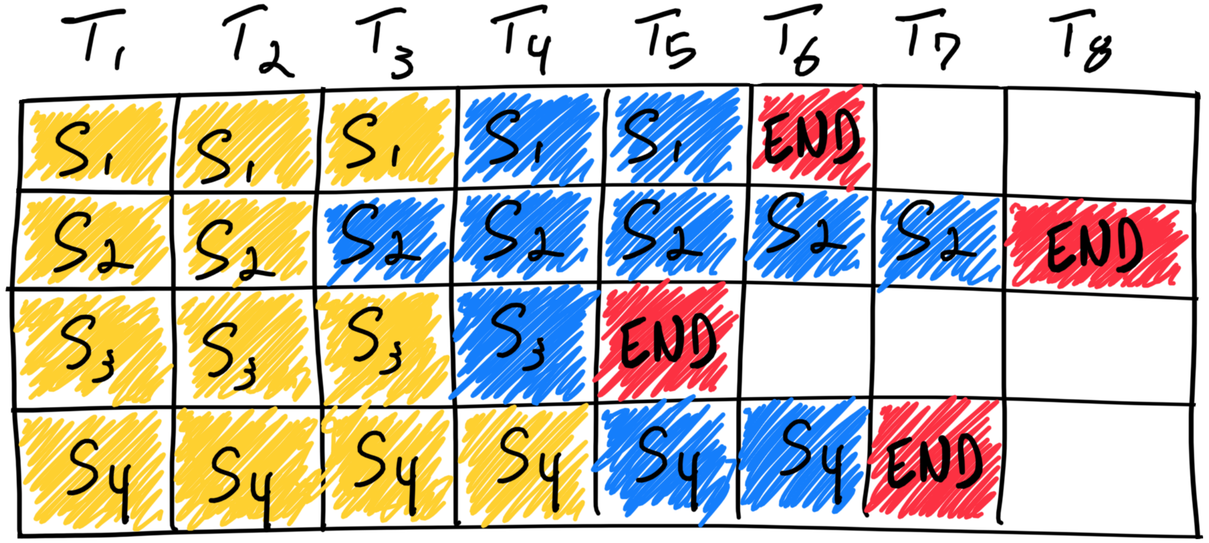
- LLM Internals
- **Scaling Law**
- Serving and inference optimization
  - Speculative decoding (question in your PA3)
  - **Continuous batching and Paged attention**

LLM Decoding Timeline



Batching Requests to Improve GPU Performance

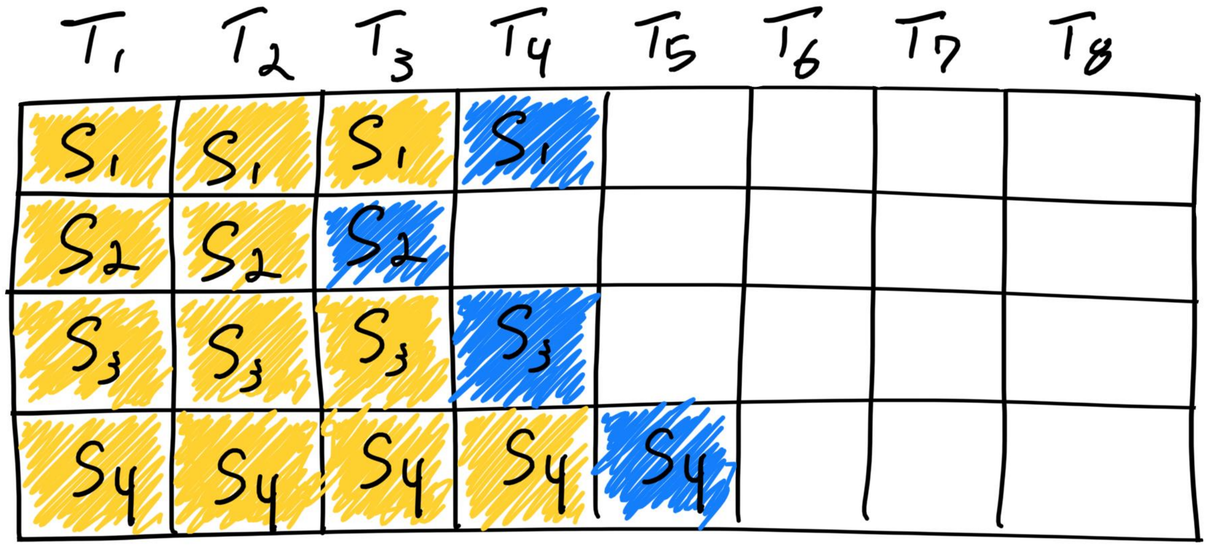


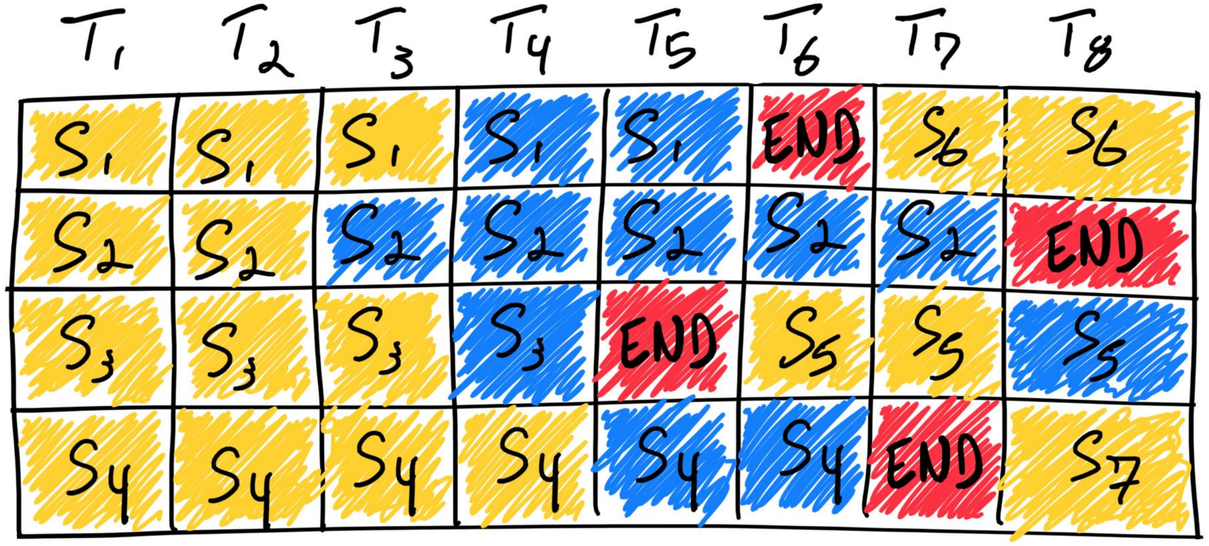


Issues with static batching:

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

Continuous Batching



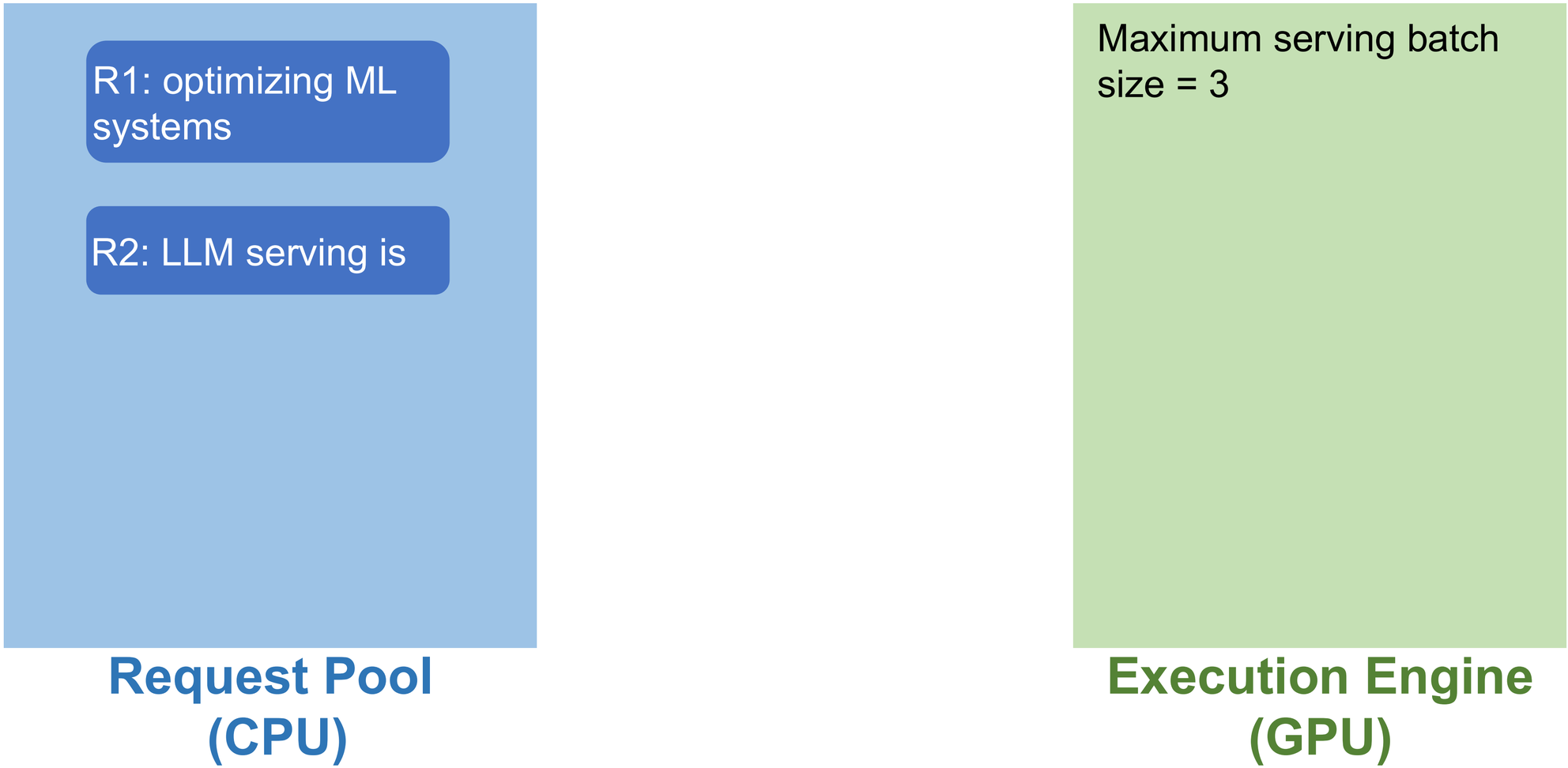


Benefits:

- Higher GPU utilization
- New requests can start immediately

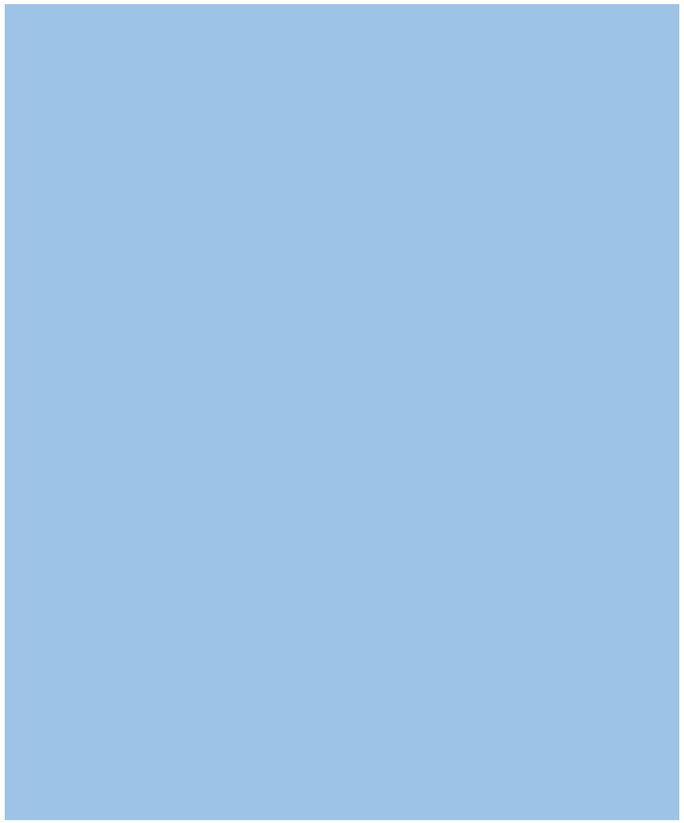
Continuous Batching Step-by-Step

- Receives two new requests R1 and R2



Continuous Batching Step-by-Step

- Iteration 1: decode R1 and R2



Request Pool (CPU)

Maximum serving batch size = 3

R1: optimizing ML systems

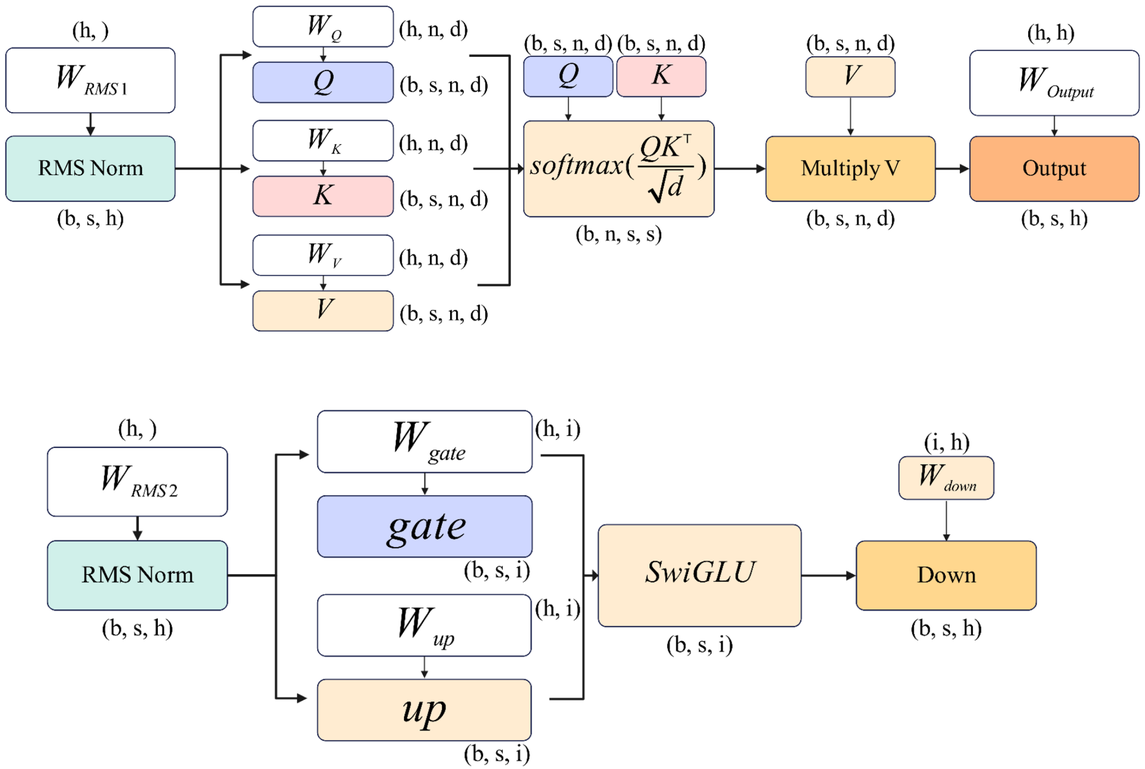
R2: LLM serving is

Execution Engine (GPU)

Iteration 1

Continuous Batching Step-by-Step

- Iteration 1: decode R1 and R2



Q: How to batch these?

Maximum serving batch size = 3

R1: optimizing ML systems

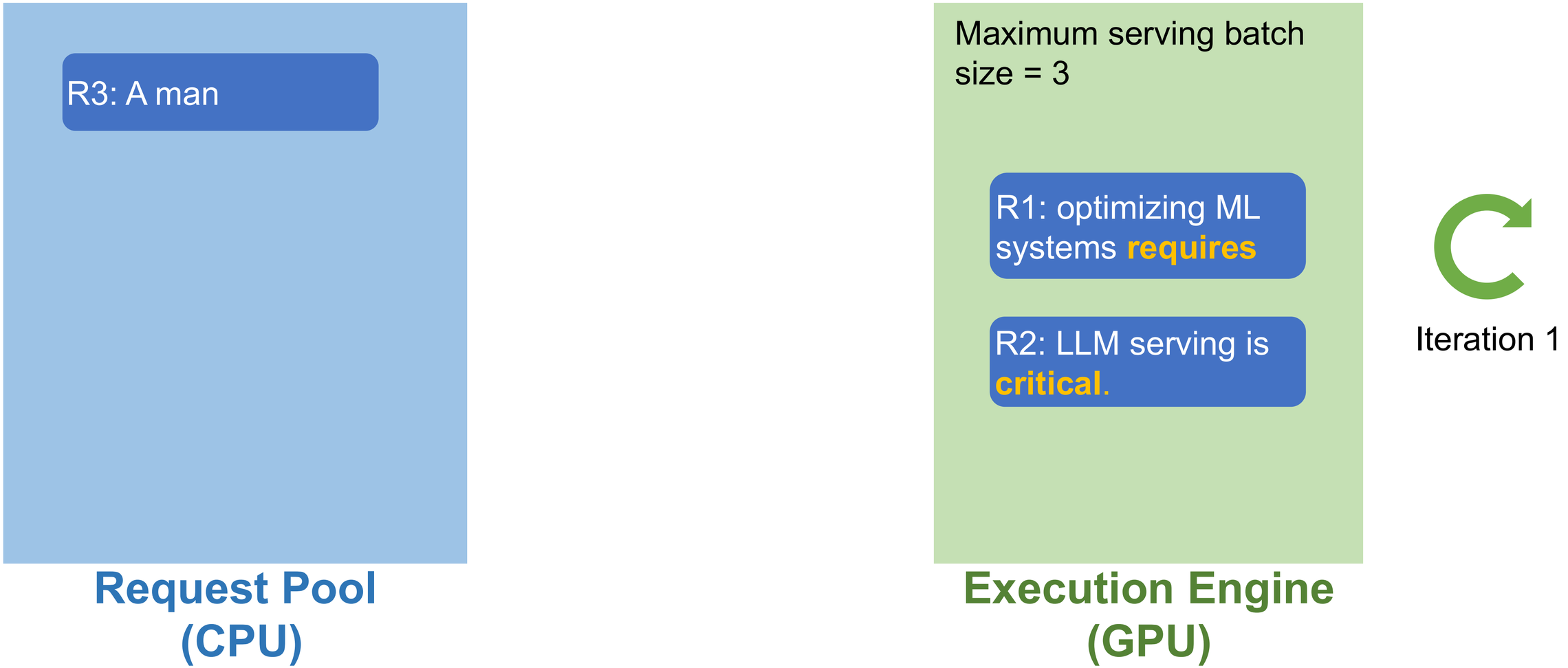
R2: LLM serving is

Iteration 1

Execution Engine (GPU)

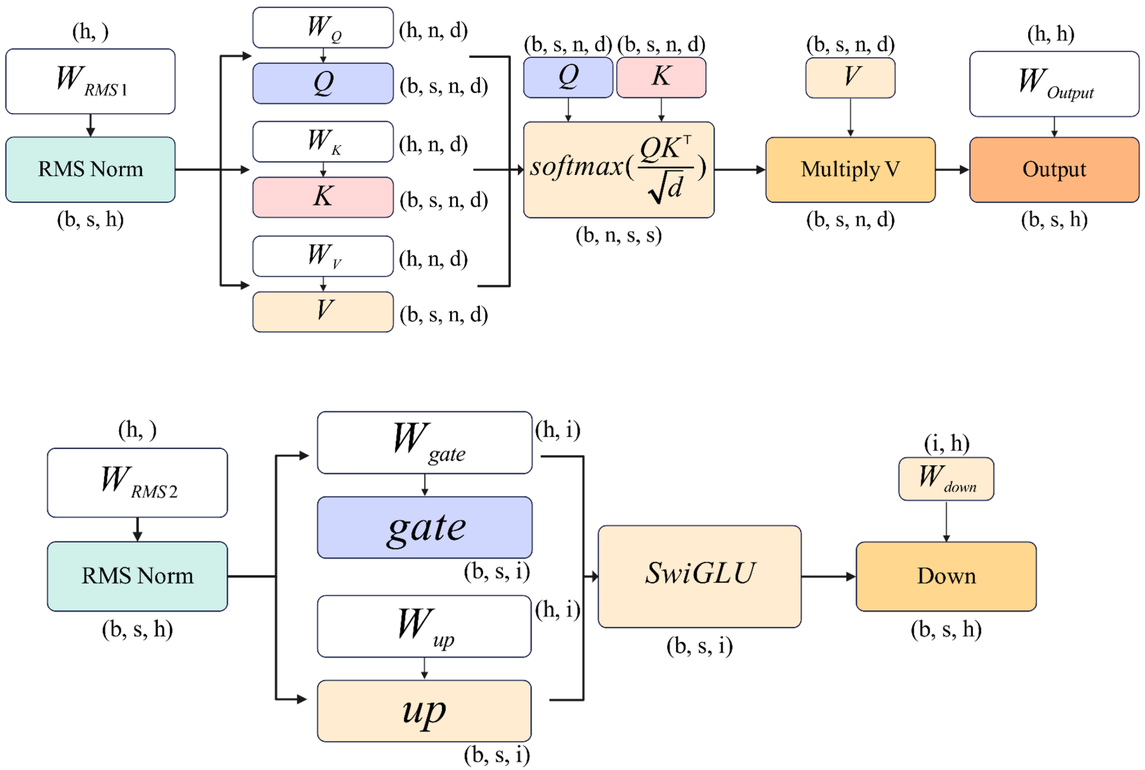
Continuous Batching Step-by-Step

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



Continuous Batching Step-by-Step

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



Q: How to batch these?

Maximum serving batch size = 3

R1: optimizing ML systems **requires**

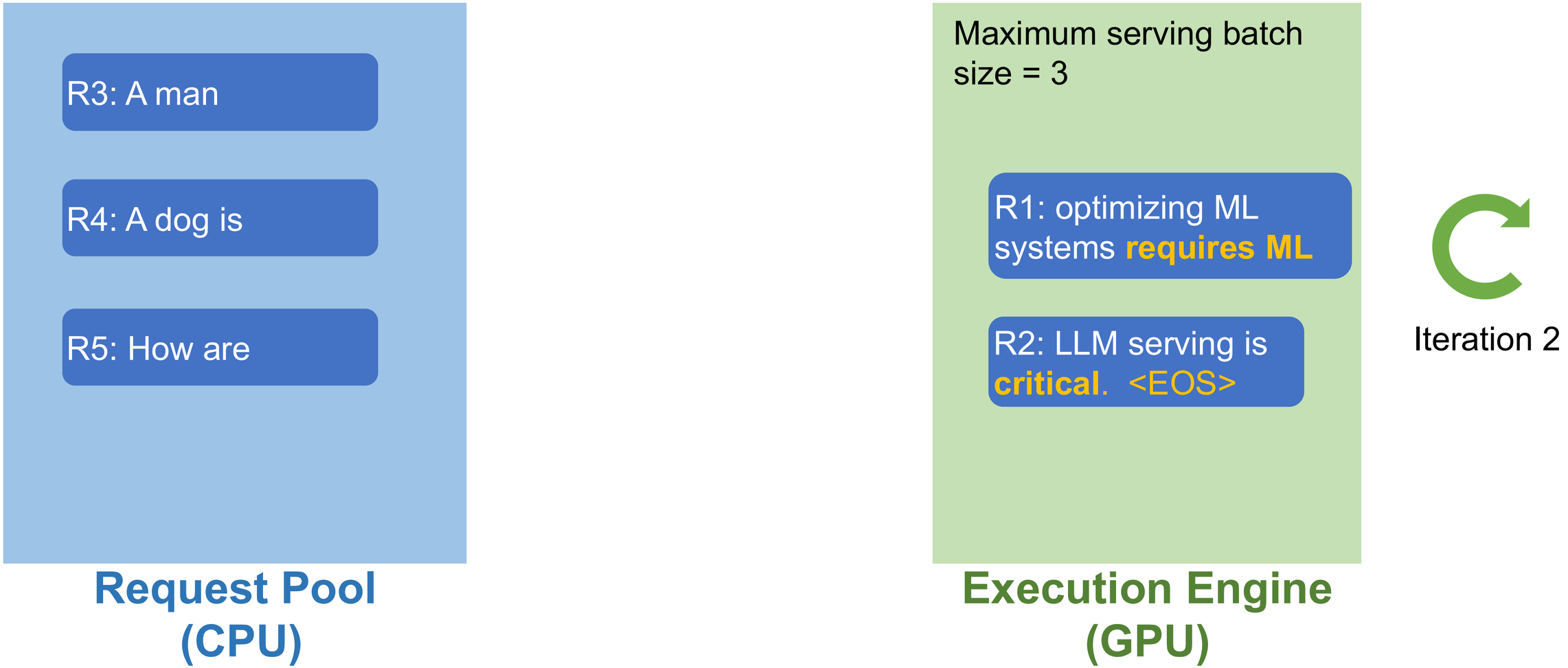
R2: LLM serving is **critical**.

Iteration 1

Execution Engine (GPU)

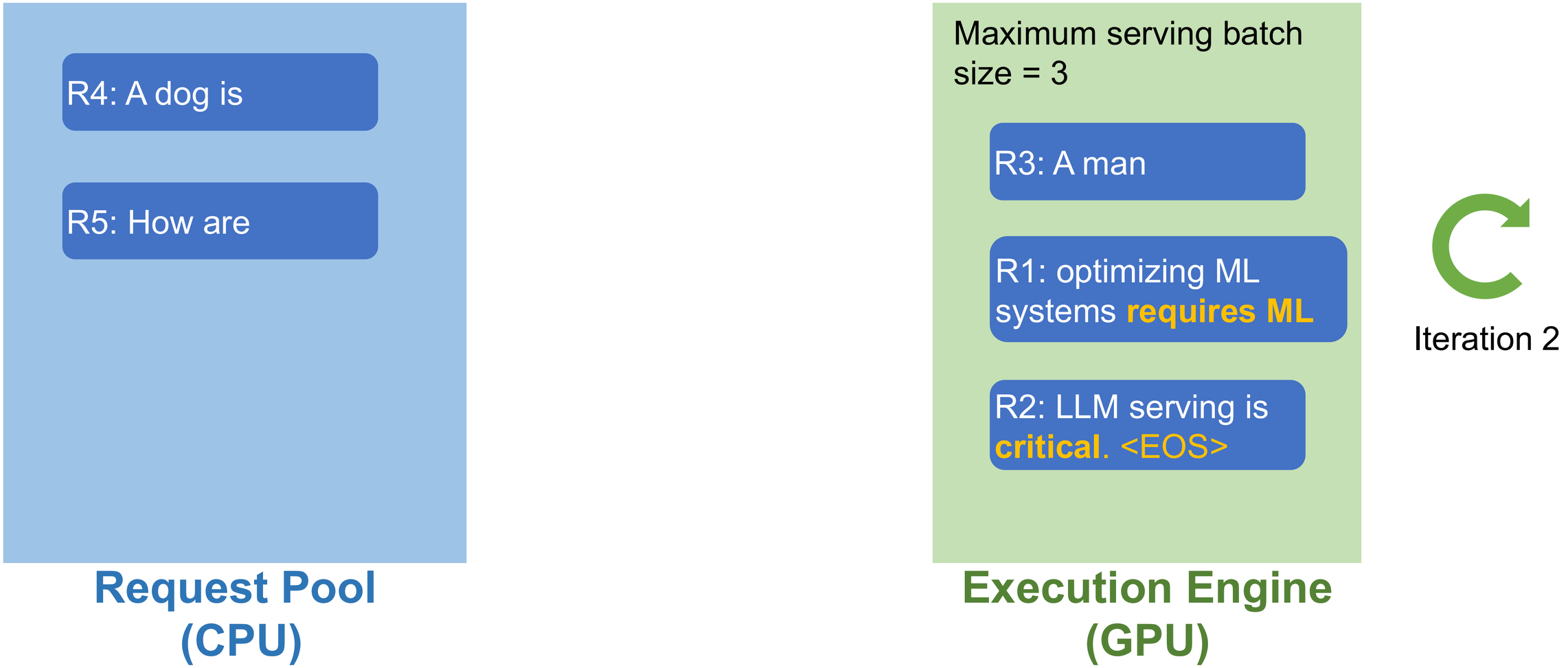
Traditional Batching

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



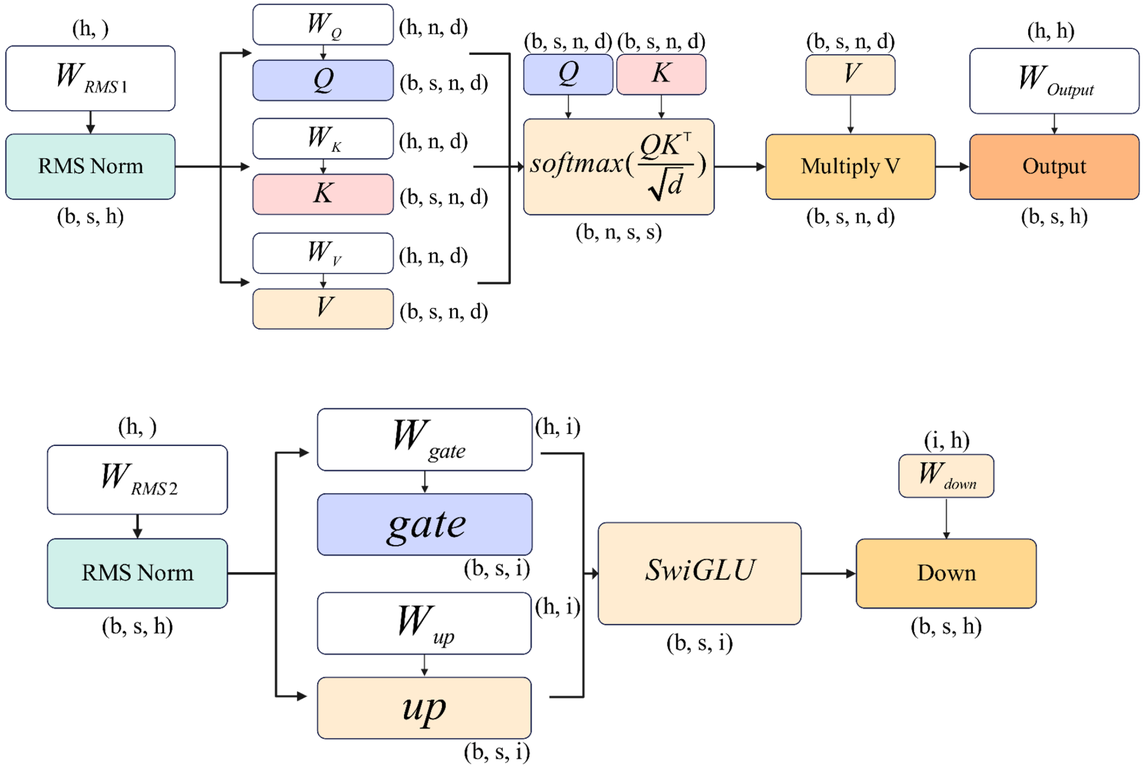
Continuous Batching

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



Continuous Batching

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



Q: How to batch these?

Maximum serving batch size = 3

R3: A man

R1: optimizing ML systems **requires ML**

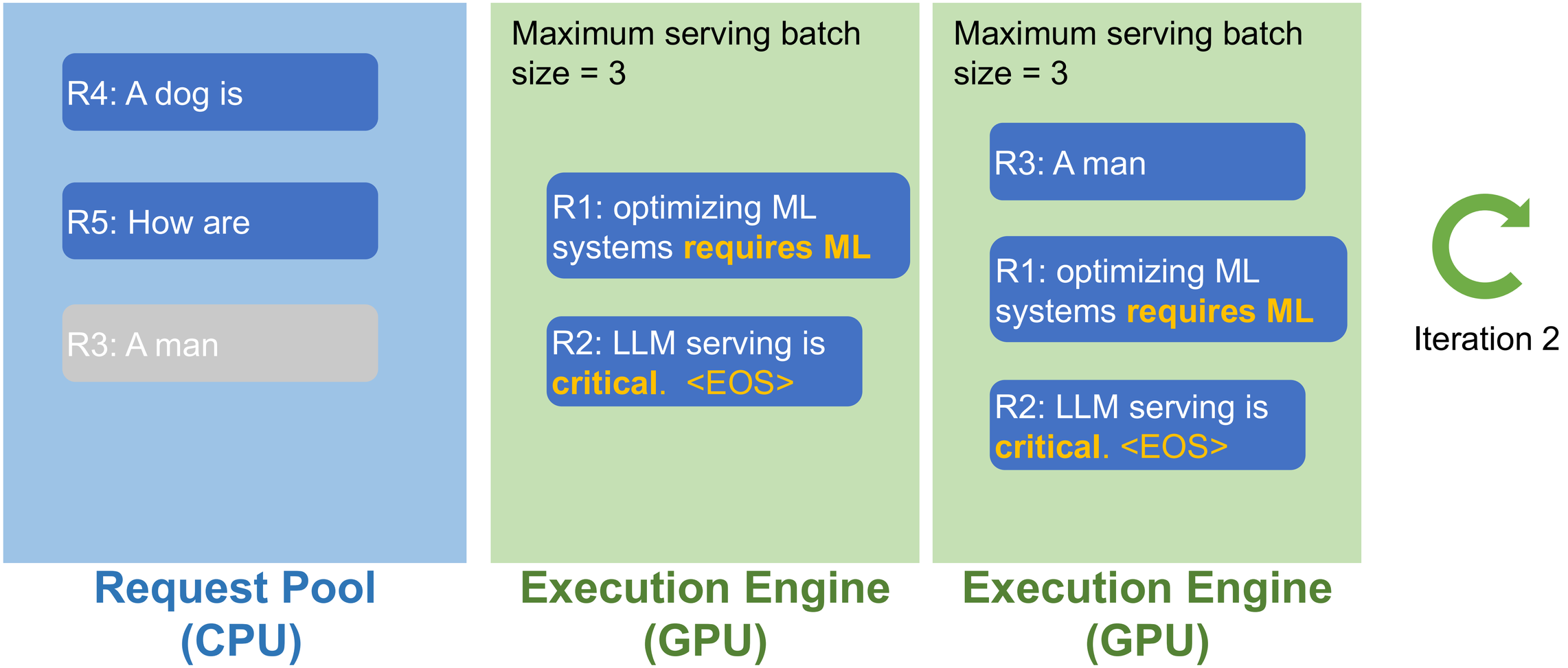
R2: LLM serving is **critical** <EOS>

Iteration 2

Execution Engine (GPU)

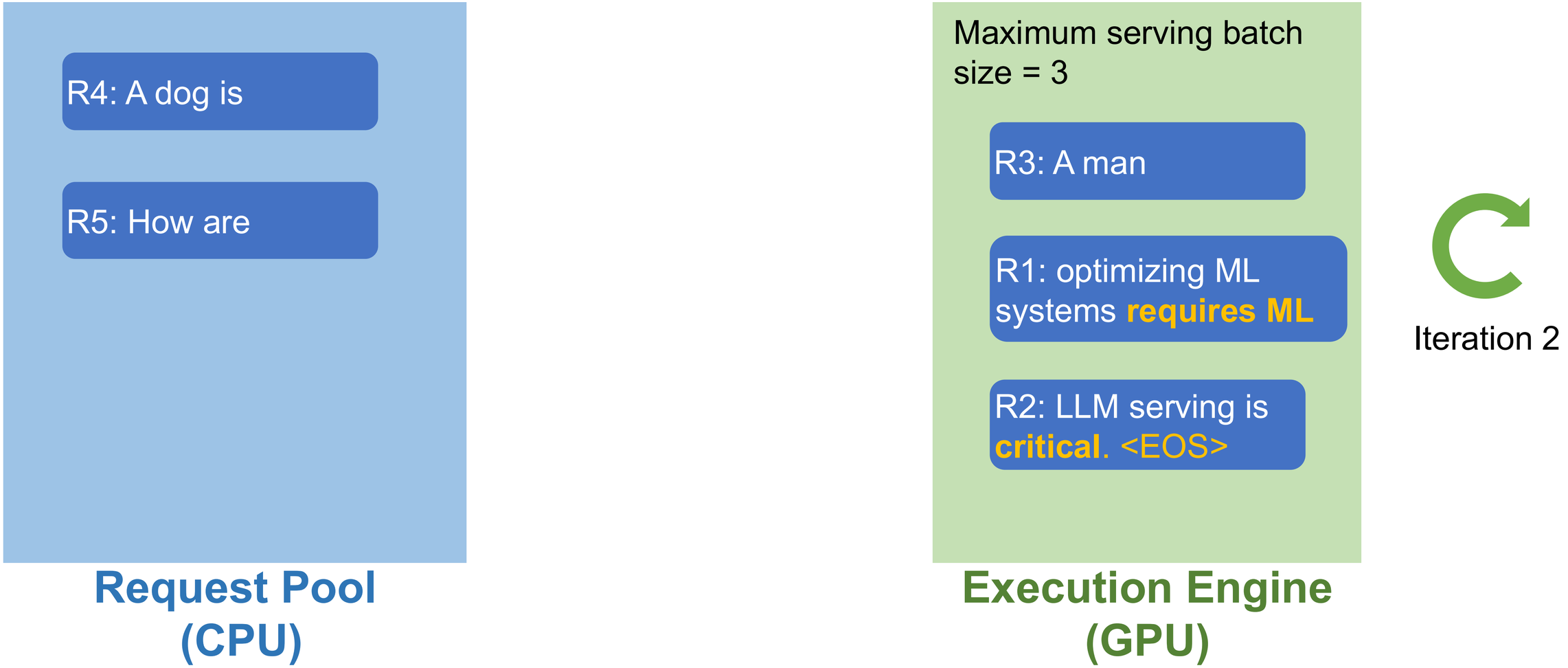
Traditional vs. Continuous Batching

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



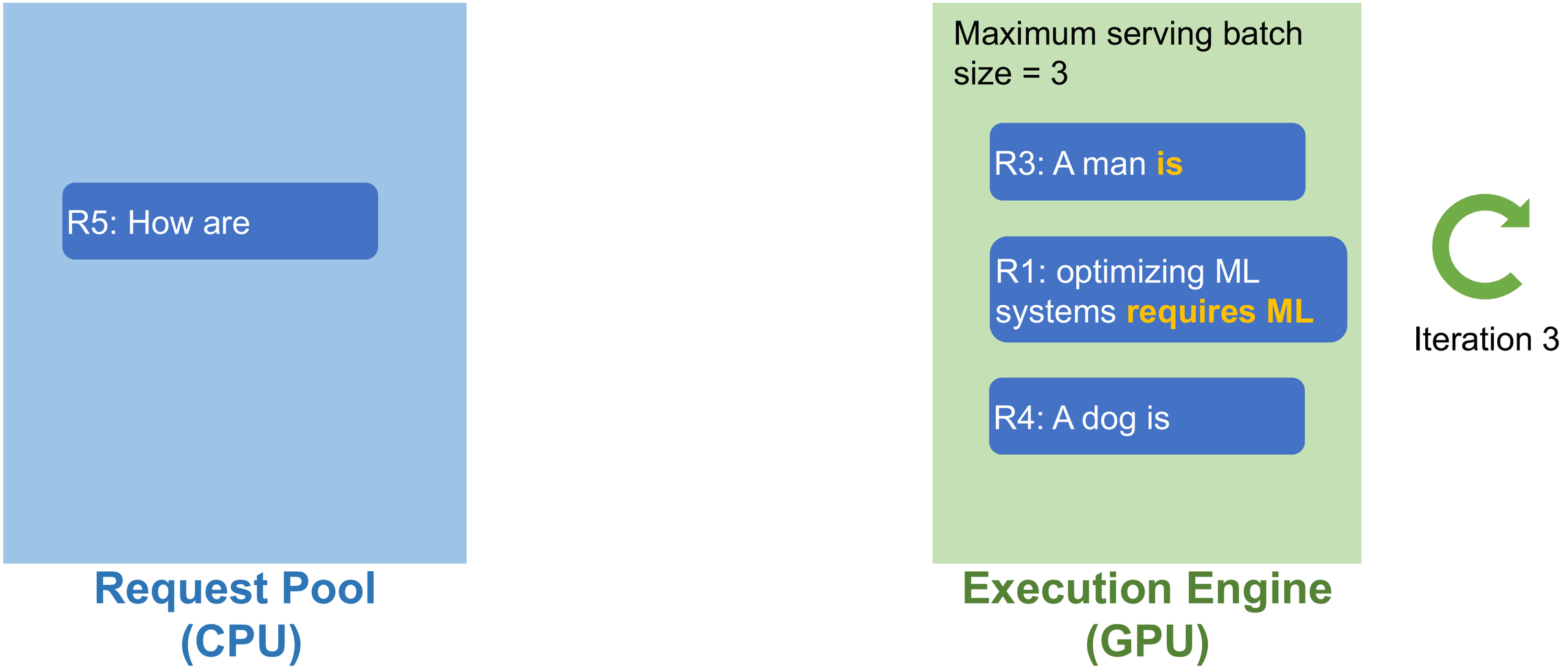
Continuous Batching

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



Continuous Batching Step-by-Step

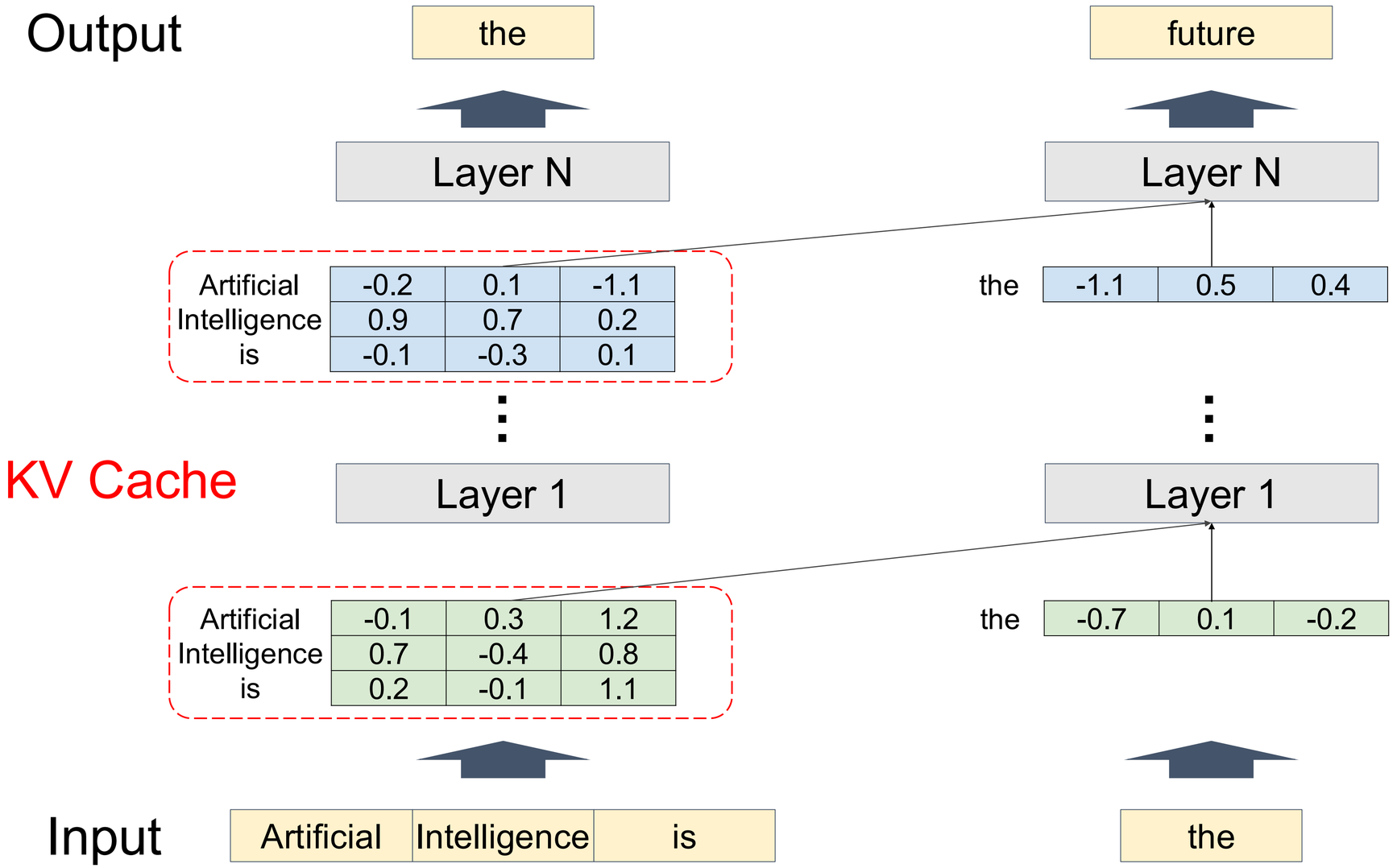
- Iteration 3: decode R1, R3, R4



Summary: Continuous Batching

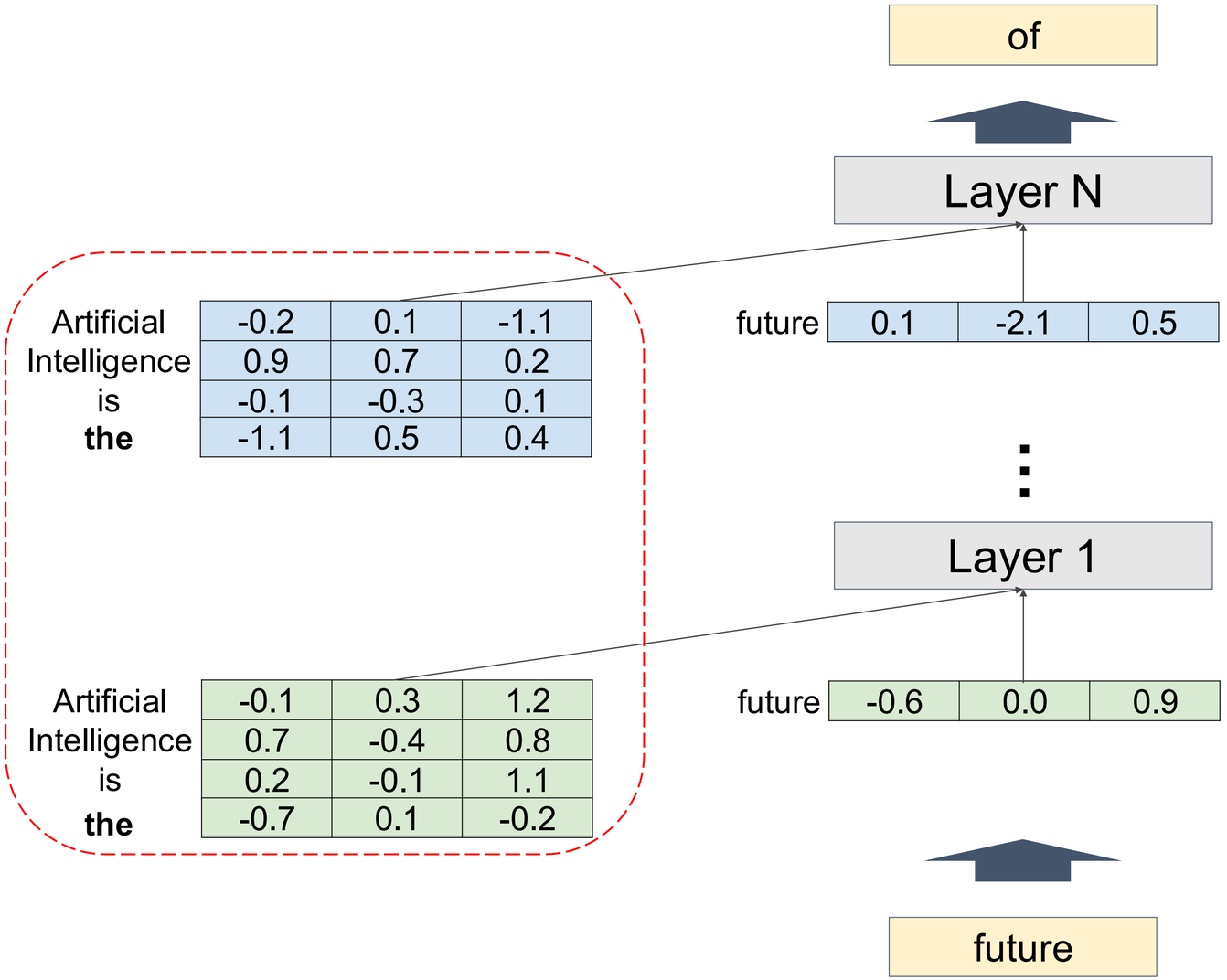
- Handle early-finished and late-arrived requests more efficiently
- Improve GPU utilization
- Key observation
  - MLP kernels are agnostic to the sequence dimension

KV Cache


Output
the
future
Layer N
Layer N
Artificial
Intelligence
is
-0.2
0.1
-1.1
0.9
0.7
0.2
-0.1
-0.3
0.1
the
-1.1
0.5
0.4
KV Cache
Layer 1
Layer 1
Artificial
Intelligence
is
-0.1
0.3
1.2
0.7
-0.4
0.8
0.2
-0.1
1.1
the
-0.7
0.1
-0.2
Input
Artificial
Intelligence
is
the

KV Cache

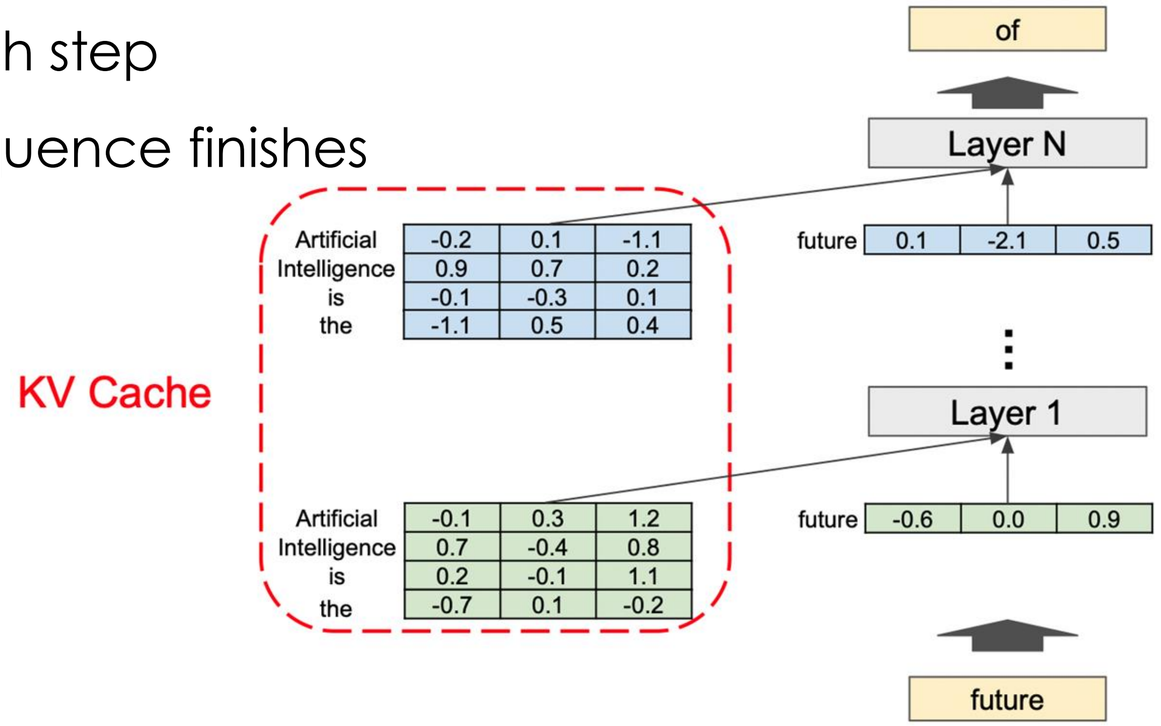
Output



Input

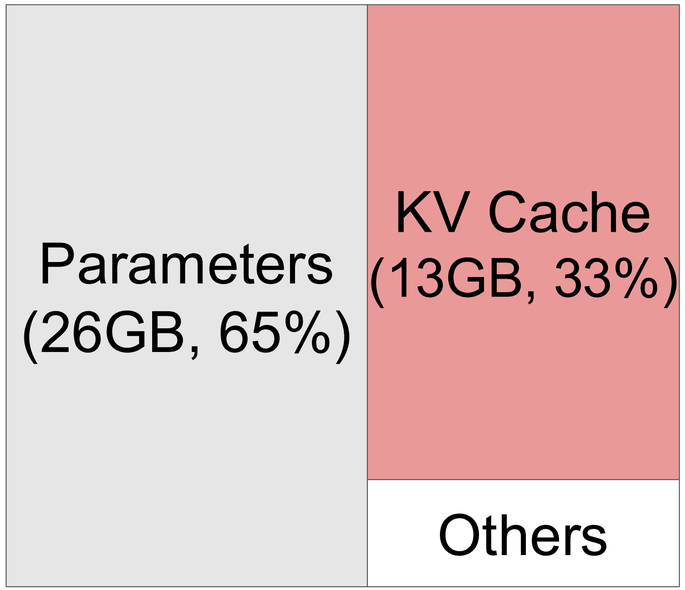
KV Cache

- Memory space to store intermediate vector representations of tokens
  - **Working set** rather than a "cache"
- The size of KV Cache dynamically grows and shrinks
  - A new token is appended in each step
  - Tokens are deleted once the sequence finishes

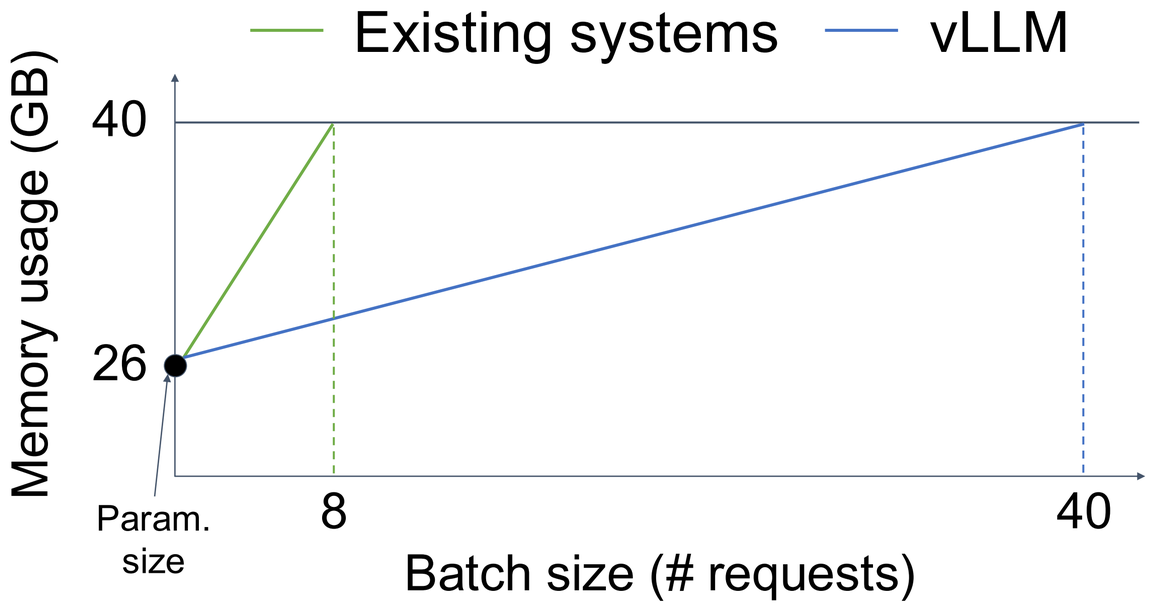


Key insight

Efficient management of KV cache is crucial for high-throughput LLM serving

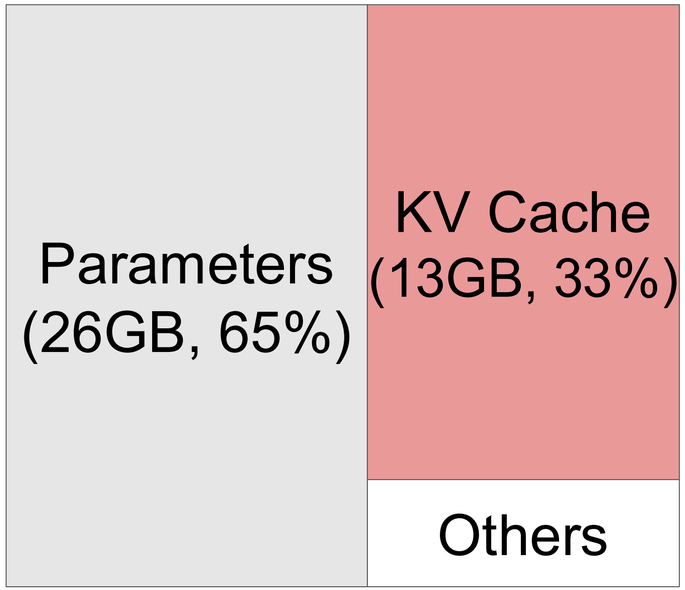


13B LLM on A100-40GB

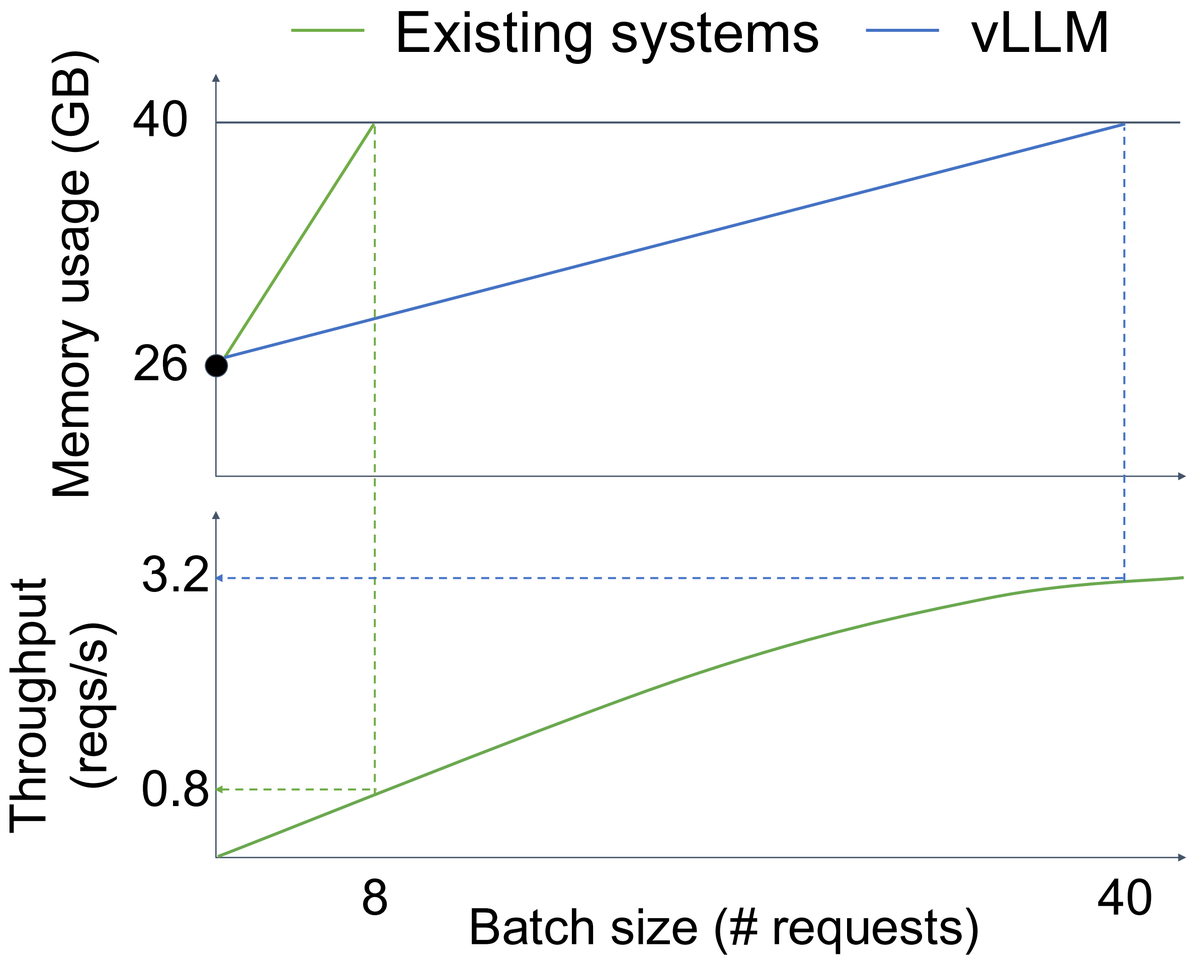


Key insight

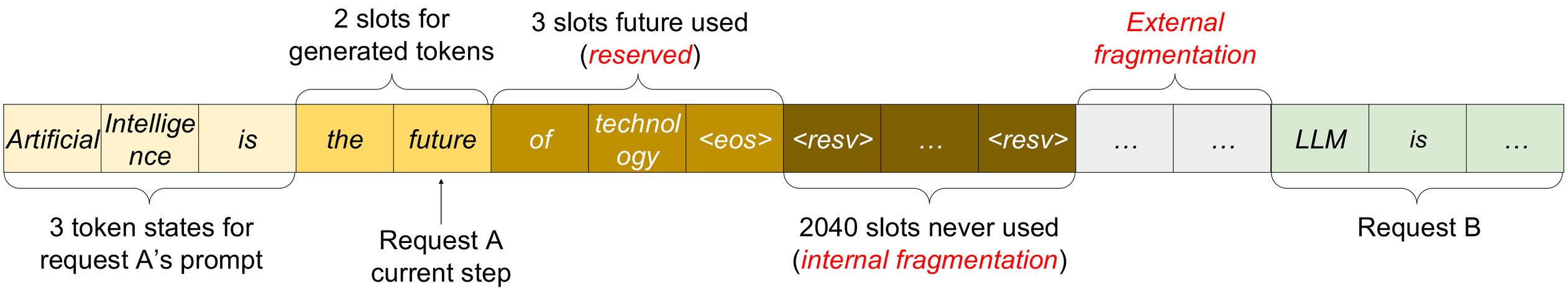
Efficient management of KV cache is crucial for high-throughput LLM serving



13B LLM on A100-40GB

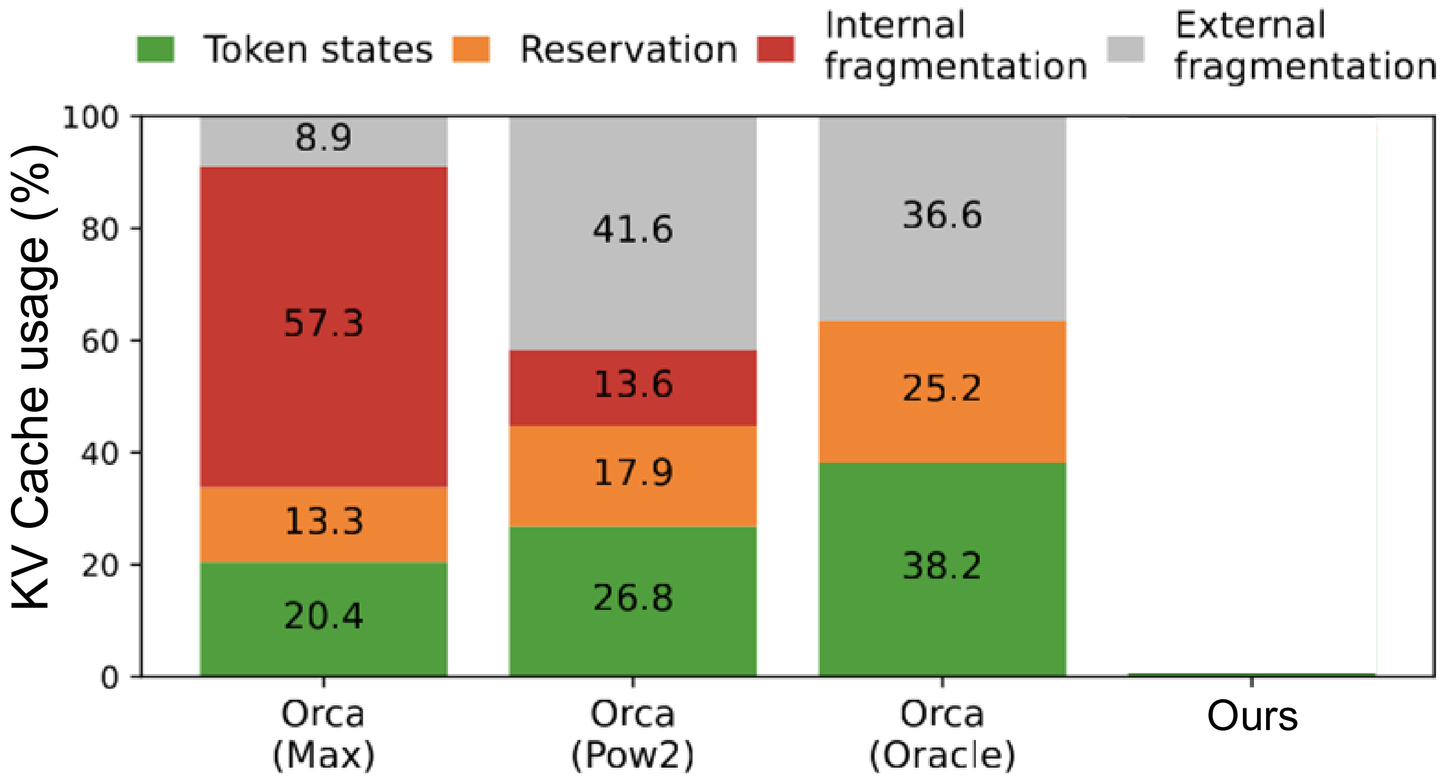


Memory waste in KV Cache



- Reservation: not used at the current step, but used in the future
- Internal fragmentation: over-allocated due to the unknown output length.

Memory waste in KV Cache

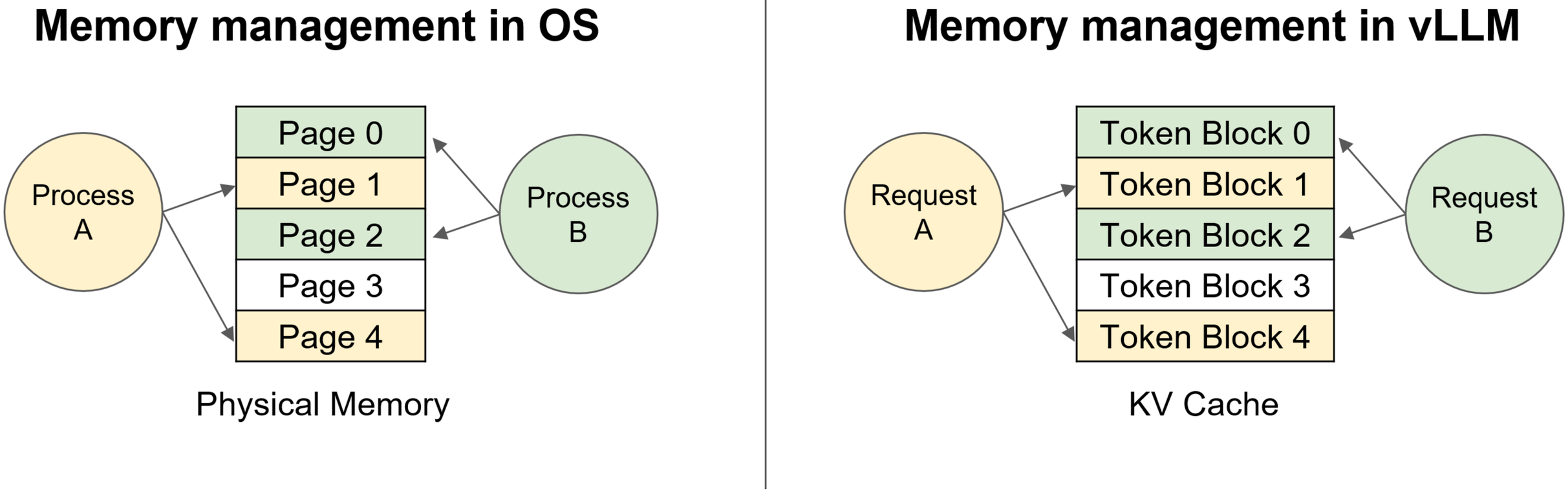


Only **20–40%** of KV cache is utilized to store token states

* Yu, G. I., Jeong, J. S., Kim, G. W., Kim, S., Chun, B. G. “Orca: A Distributed Serving System for Transformer-Based Generative Models” (OSDI 22).

vLLM: Efficient memory management for LLM inference

Inspired by **virtual memory** and **paging**

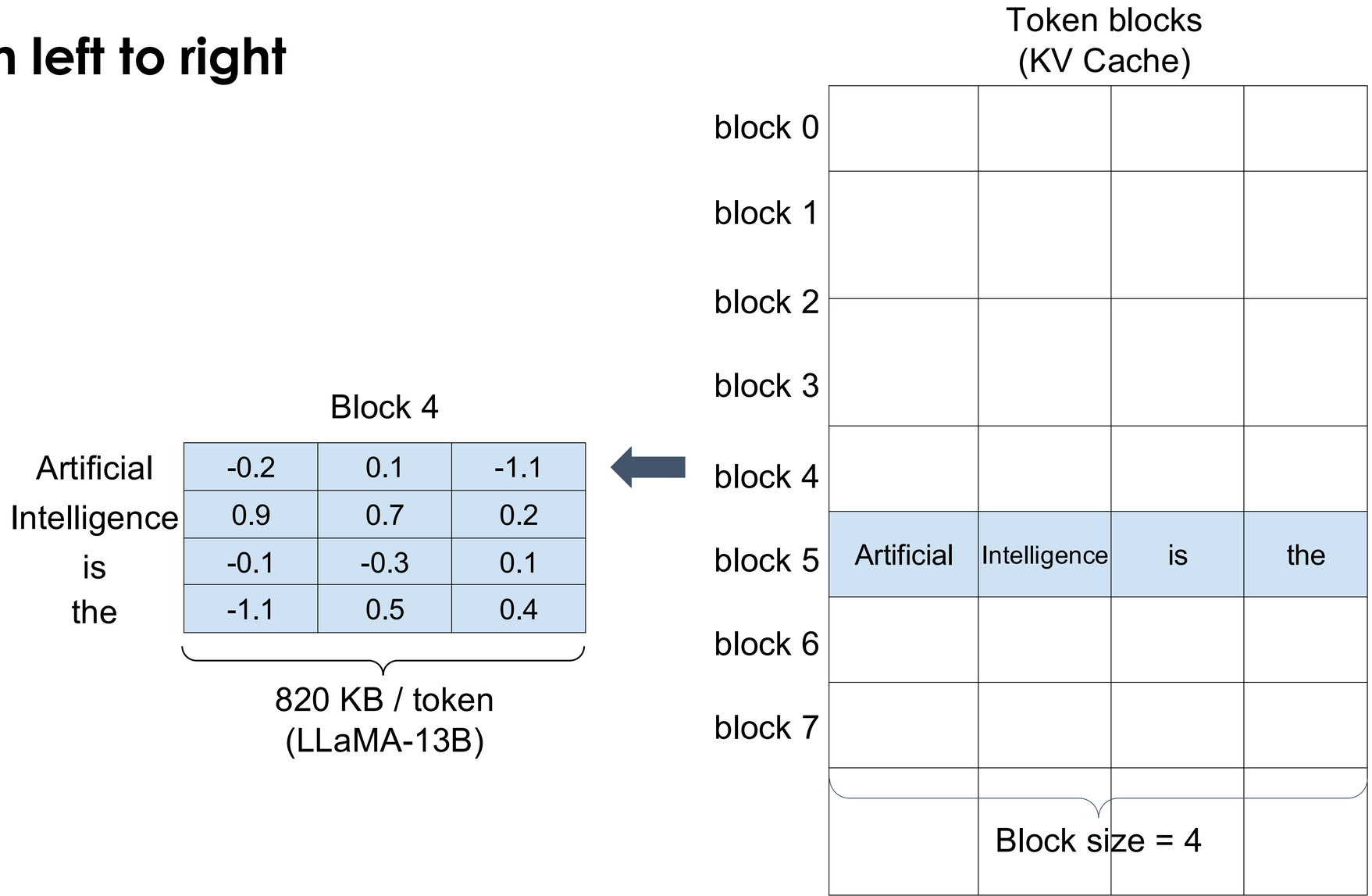


Token block

- A **fixed-size** contiguous chunk of memory that can store token states **from left to right**

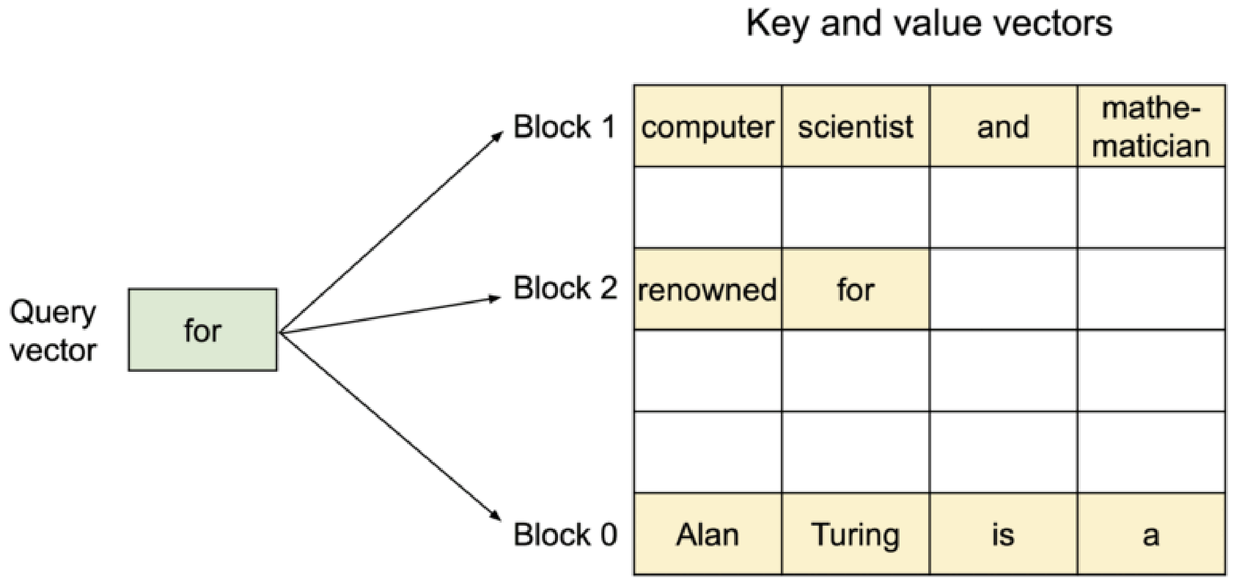
Token block

- A **fixed-size** contiguous chunk of memory that can store token states **from left to right**



Paged Attention

- An attention algorithm that allows for storing continuous keys and values in non-contiguous memory space



Logical & physical token blocks

Request A

Prompt: “Alan Turing is a computer scientist”

Logical token blocks

block 0	Alan	Turing	is	a
block 1	computer	scientist		
block 2				
block 3				

Physical token blocks (KV Cache)

block 0				
block 1				
block 2				
block 3				
block 4				
block 5				
block 6				
block 7				

Logical & physical token blocks

Request A

Prompt: “Alan Turing is a computer scientist”

Logical token blocks

block 0	Alan	Turing	is	a
block 1	computer	scientist		
block 2				
block 3				

Block table

Physical block number	# Filled
7	4
1	2
–	–
–	–

Physical token blocks (KV Cache)

block 0				
block 1	computer	scientist		
block 2				
block 3				
block 4				
block 5				
block 6				
block 7	Alan	Turing	is	a

Logical & physical token blocks

Request A

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

Logical token blocks

block 0	Alan	Turing	is	a
block 1	computer	scientist		
block 2				
block 3				

Block table

Physical block number	# Filled
7	4
1	2
–	–
–	–

Physical token blocks (KV Cache)

block 0				
block 1	computer	scientist		
block 2				
block 3				
block 4				
block 5				
block 6				
block 7	Alan	Turing	is	a

Logical & physical token blocks

Request A

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

Logical token blocks

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

Block table

Physical block number	# Filled
7	4
1	2
–	–
–	–

Physical token blocks (KV Cache)

block 0				
block 1	computer	scientist		
block 2				
block 3				
block 4				
block 5				
block 6				
block 7	Alan	Turing	is	a

Logical & physical token blocks

Request A

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

Logical token blocks

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

Block table

Physical block number	# Filled
7	4
1	3
–	–
–	–

Physical token blocks (KV Cache)

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

Logical & physical token blocks

Request A

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

Logical token blocks

block 0	Alan	Turing	is	a
block 1	computer	scientist	and	mathema tician
block 2				
block 3				

Block table

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

Physical token blocks (KV Cache)

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

Logical & physical token blocks

Request A

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

Logical token blocks

block 0	Alan	Turing	is	a
block 1	computer	scientist	and	mathema tician
block 2	renowned			
block 3				

Block table

Physical block number	# Filled
7	4
1	4
5	1
–	–

Physical token blocks (KV Cache)

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

Allocated on demand

Serving multiple requests

Request A

Block Table

Logical token blocks

Alan	Turing	is	a
computer	scientist	and	mathematician
renowned			

Physical token blocks
(KV Cache)

computer	scientist	and	mathematician
Artificial	Intelligence	is	the
renowned			
future	of	technology	
Alan	Turing	is	a

Block Table

Request B

Logical token blocks

Artificial	Intelligence	is	the
future	of	technology	

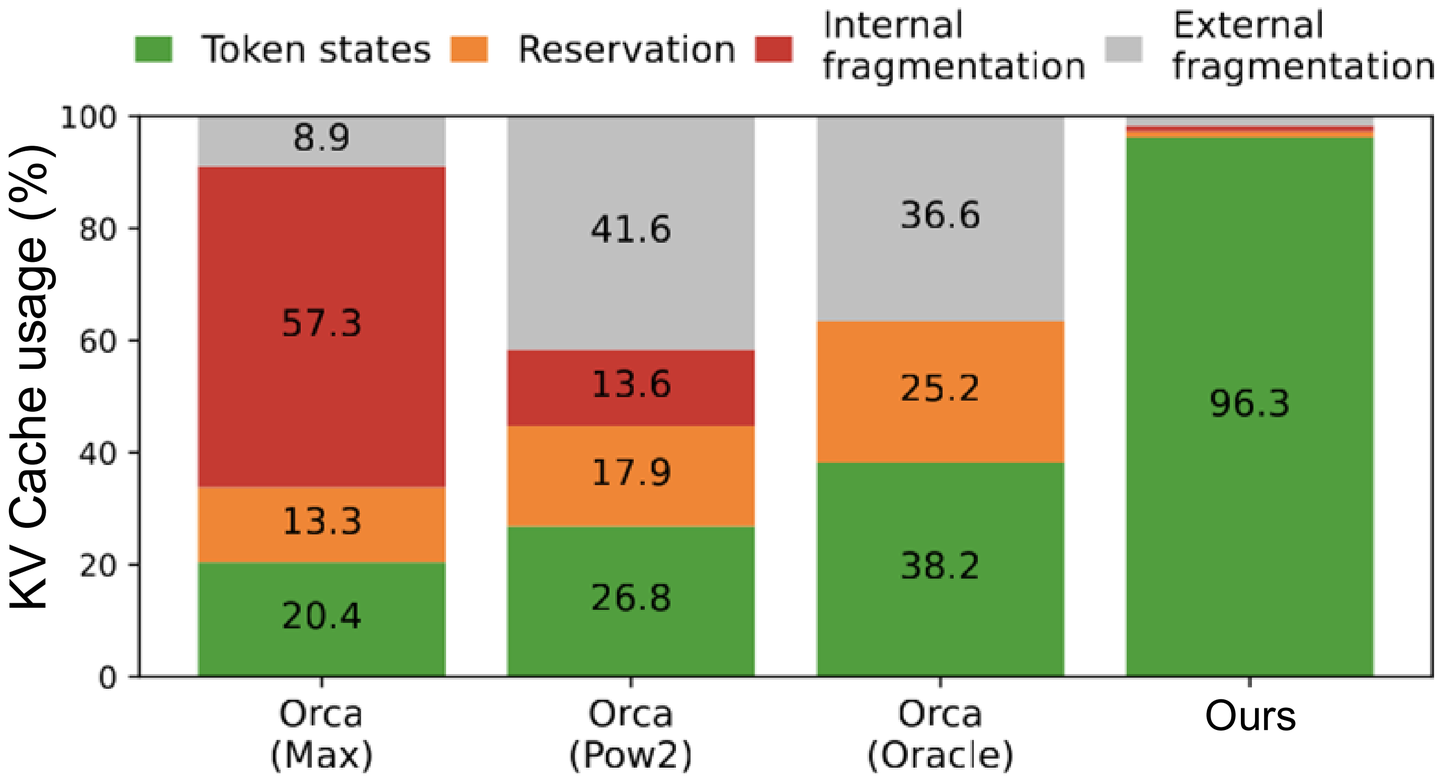
Memory efficiency of vLLM

- Minimal internal fragmentation
  - Only happens at the last block of a sequence
  - **# wasted tokens / seq < block size**
    - Sequence: O(100) – O(1000) tokens
    - Block size: 16 or 32 tokens
- No external fragmentation

Alan	Turing	is	a
computer	scientist	and	mathematician
renowned			

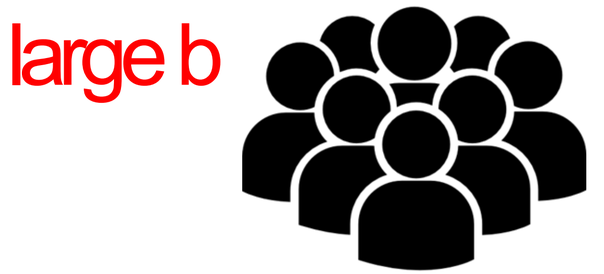
Internal fragmentation

Effectiveness of PagedAttention



96.3% KV cache utilization

Large Language Models



- LLM Internals
- **Scaling Law**
- Serving and inference optimization
  - Speculative decoding (question in your PA3)
  - **Continuous batching and Paged attention**
  - Disaggregation (in reading)
  - Chunked prefill
  - Kernels.. (next lecture)