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

DSC 291: ML Systems
Spring 2024

- LLMs
- Parallelization
- Single-device Optimization
- Basics

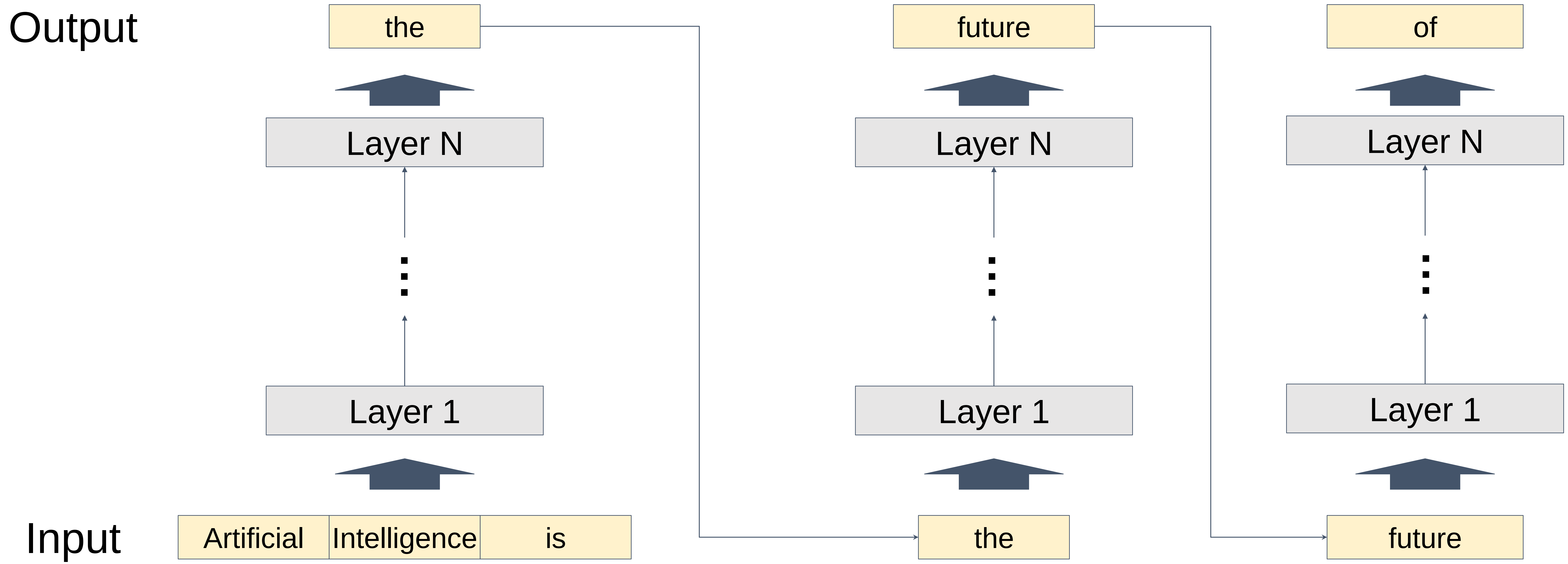
Course Evaluation

- Course evaluation is sent out
  - May 27 at 12:00 AM and Saturday, June 8
  - We are **47.62%** now --- need to reach 80% to get 2 points

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

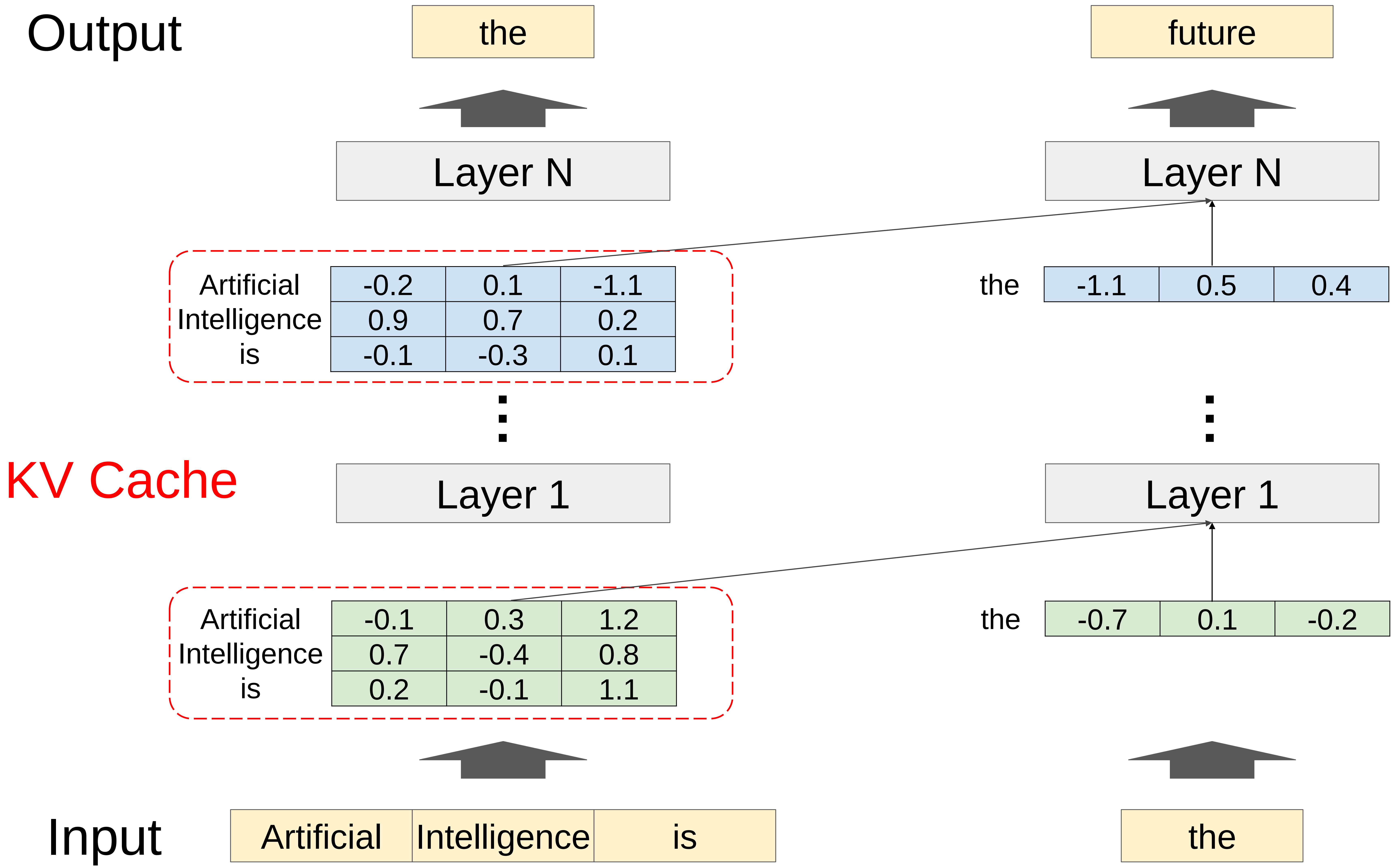
Inference process of LLMs



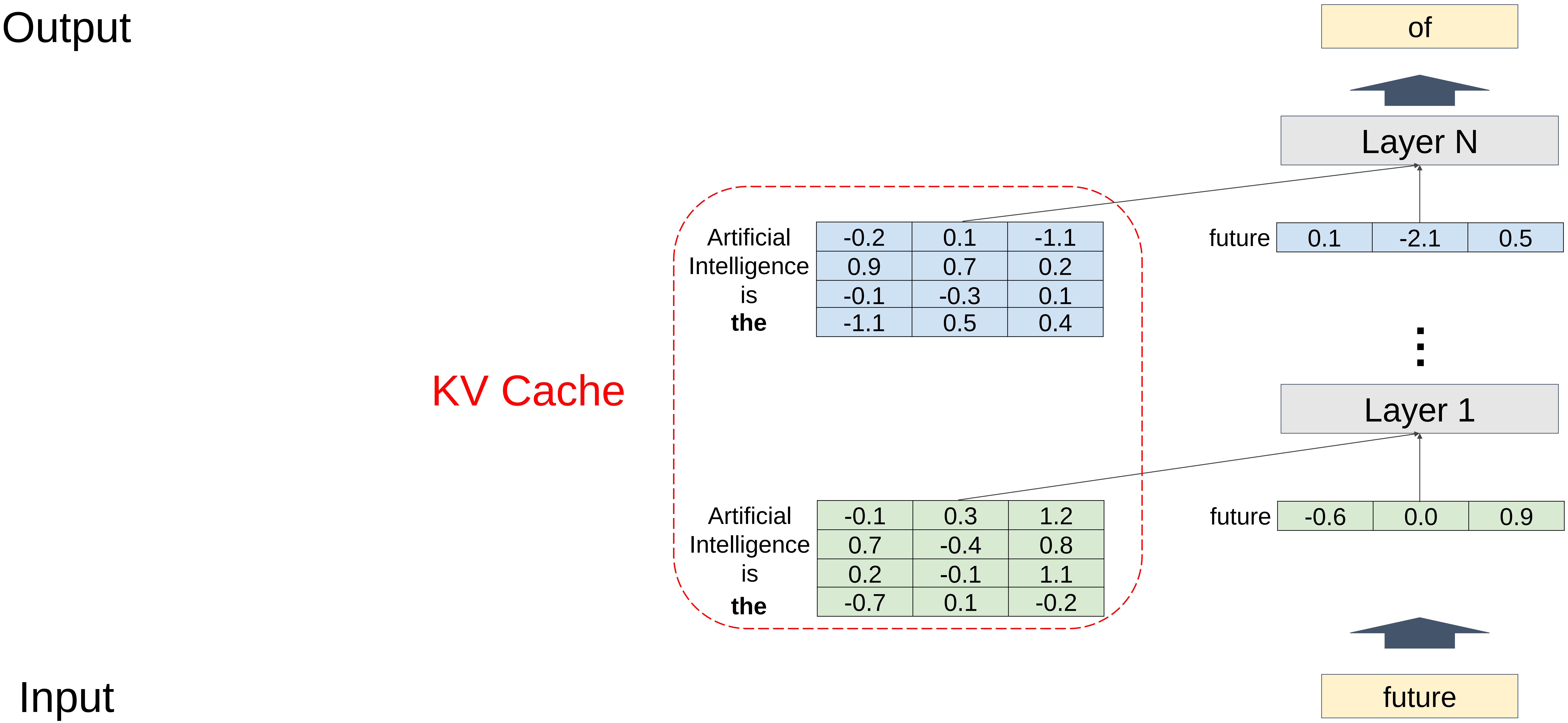
Repeat until the sequence

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

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

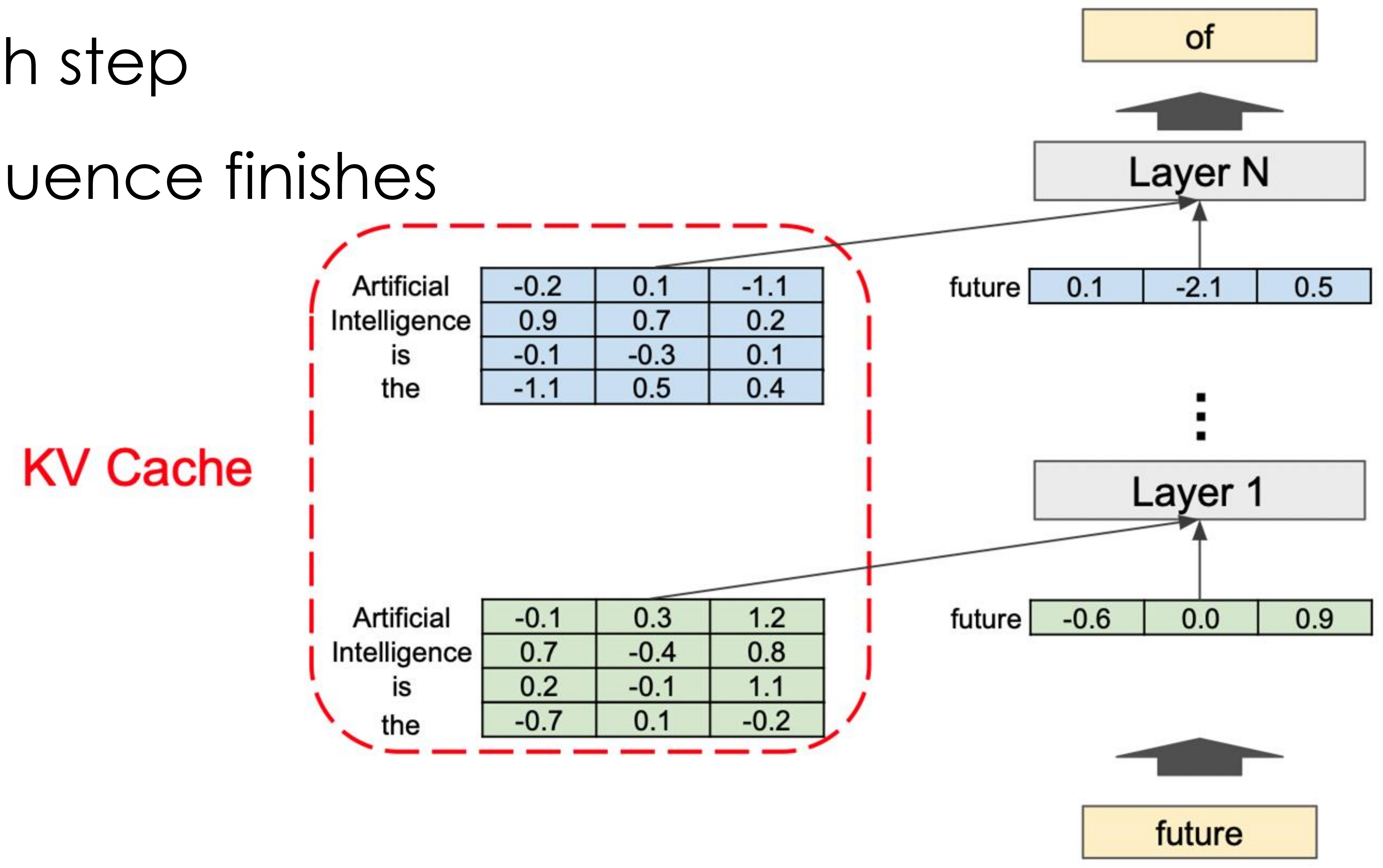
KV Cache



Output
of
Layer N
Artificial
Intelligence
is
the
-0.2 0.1 -1.1
0.9 0.7 0.2
-0.1 -0.3 0.1
-1.1 0.5 0.4
future
0.1 -2.1 0.5
KV Cache
Layer 1
Artificial
Intelligence
is
the
-0.1 0.3 1.2
0.7 -0.4 0.8
0.2 -0.1 1.1
-0.7 0.1 -0.2
future
-0.6 0.0 0.9
Input
future

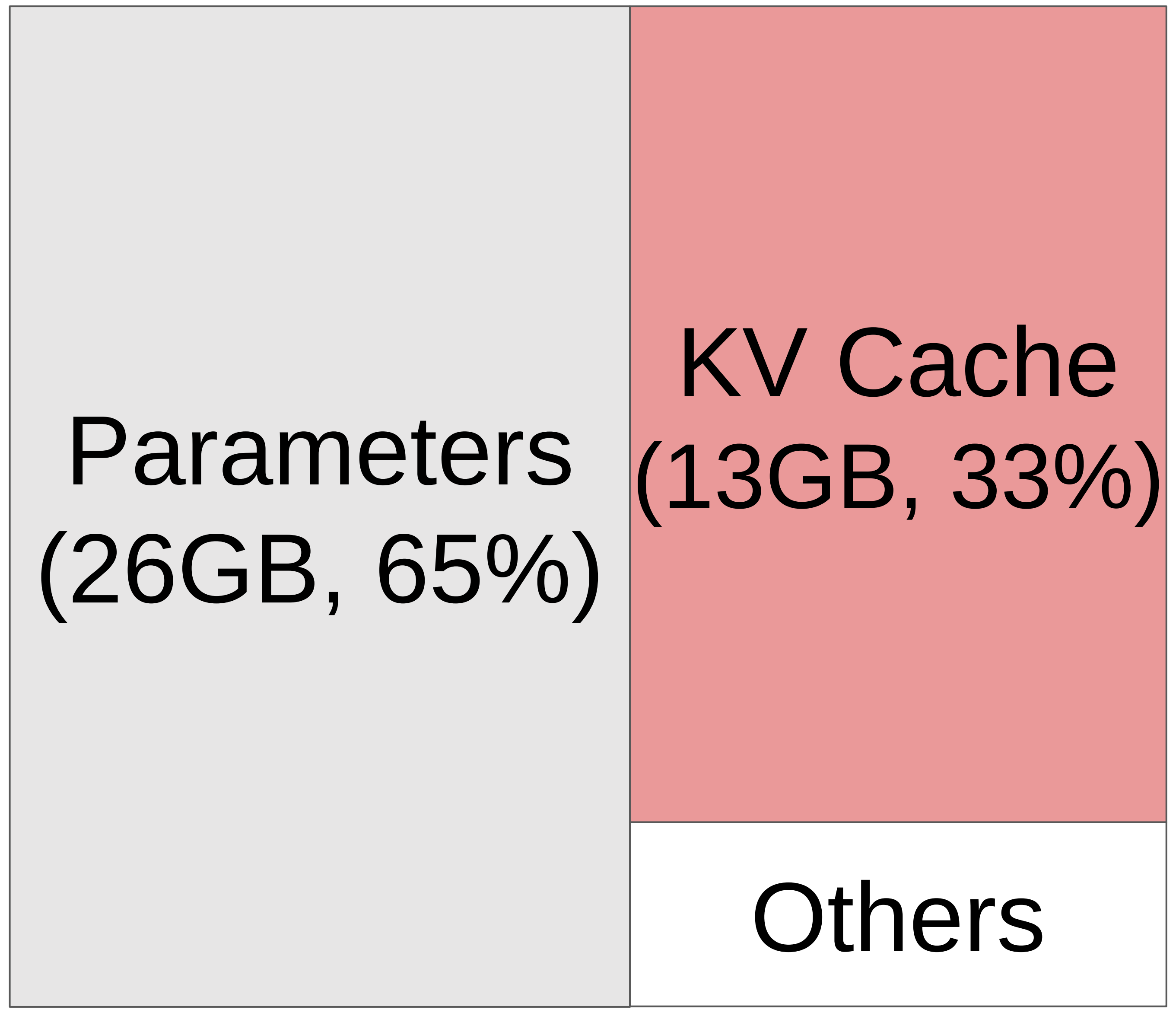
KV Cache

- 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

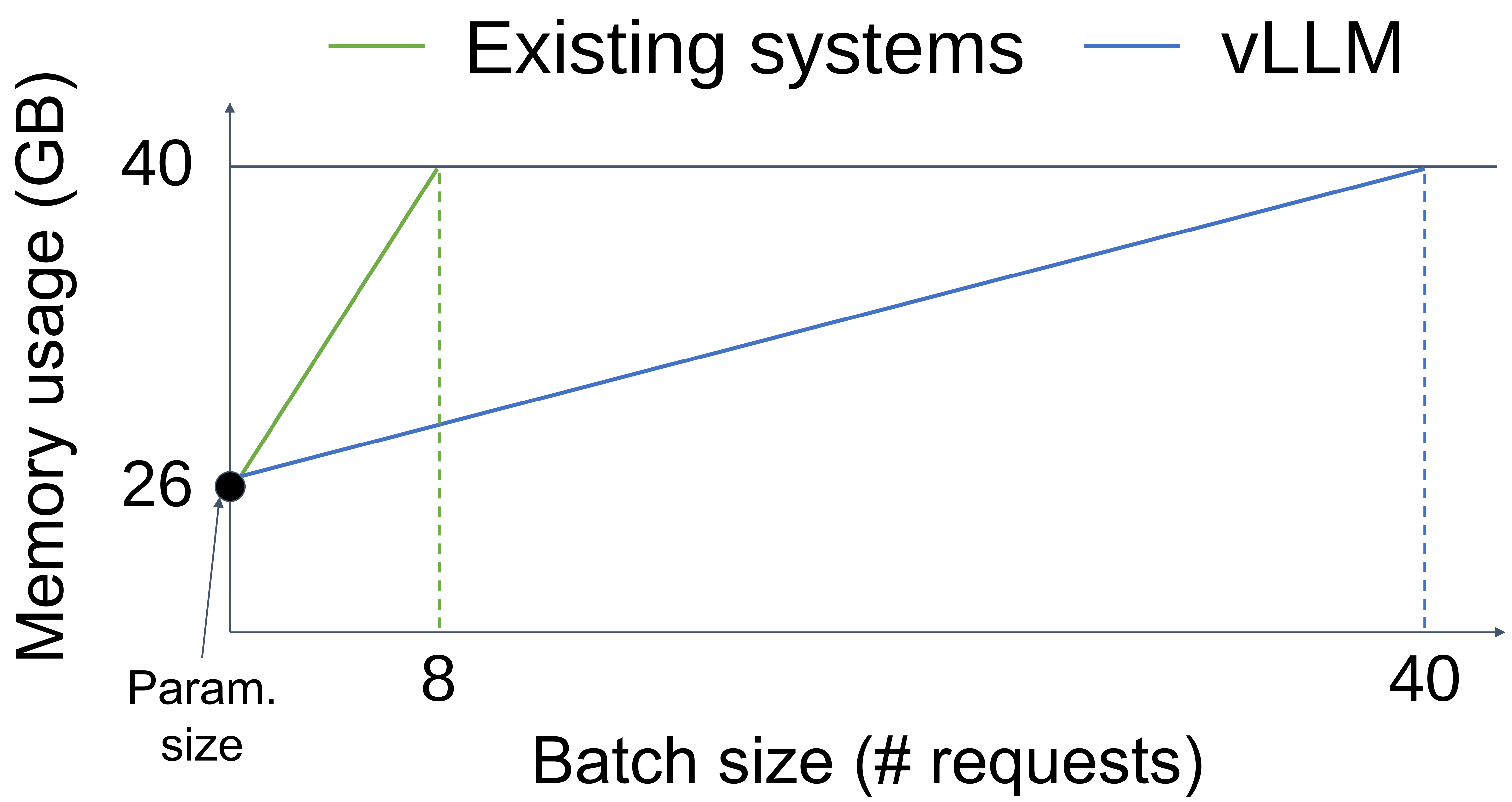


Key insight

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

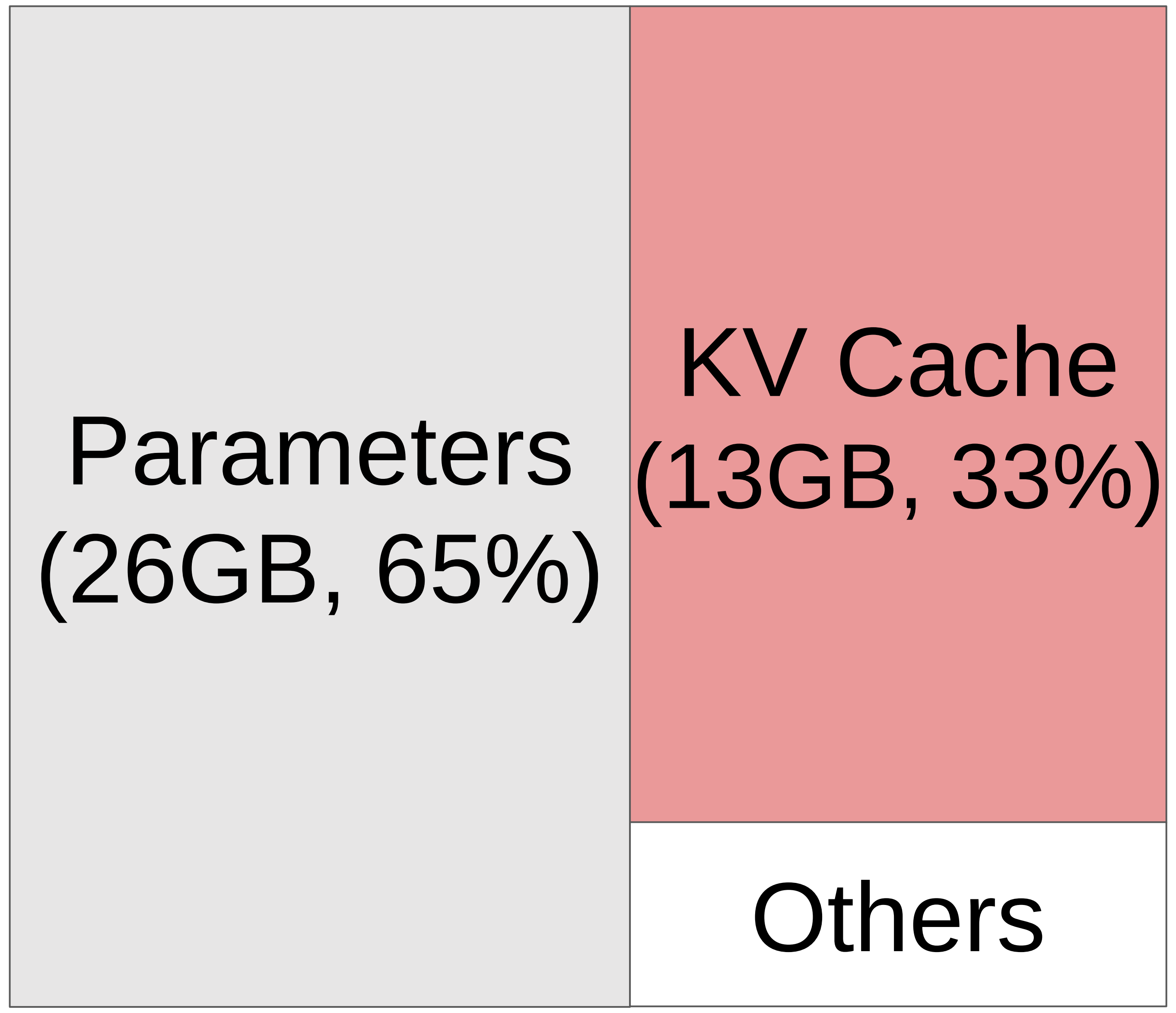


13B LLM on A100-40GB

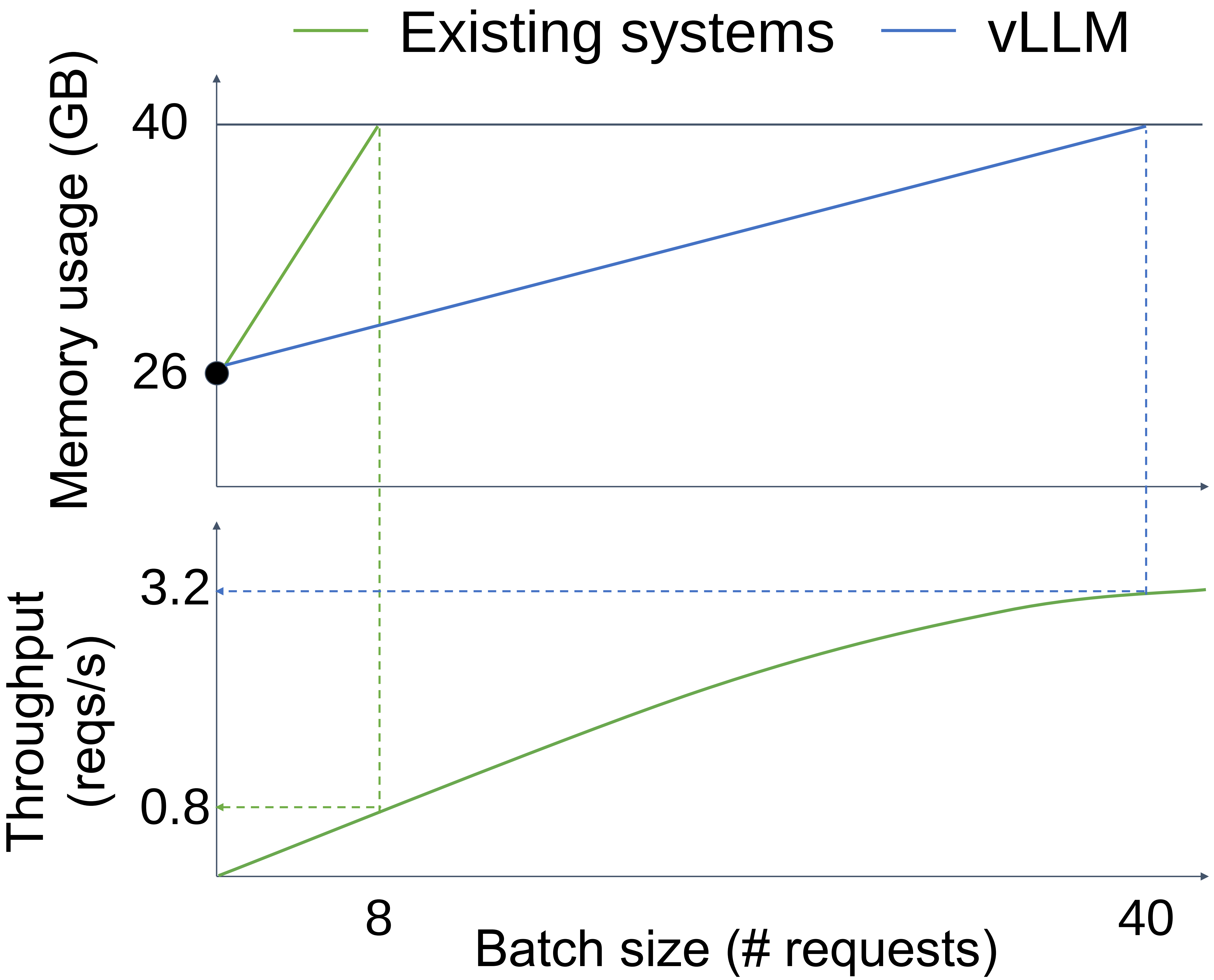


Key insight

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



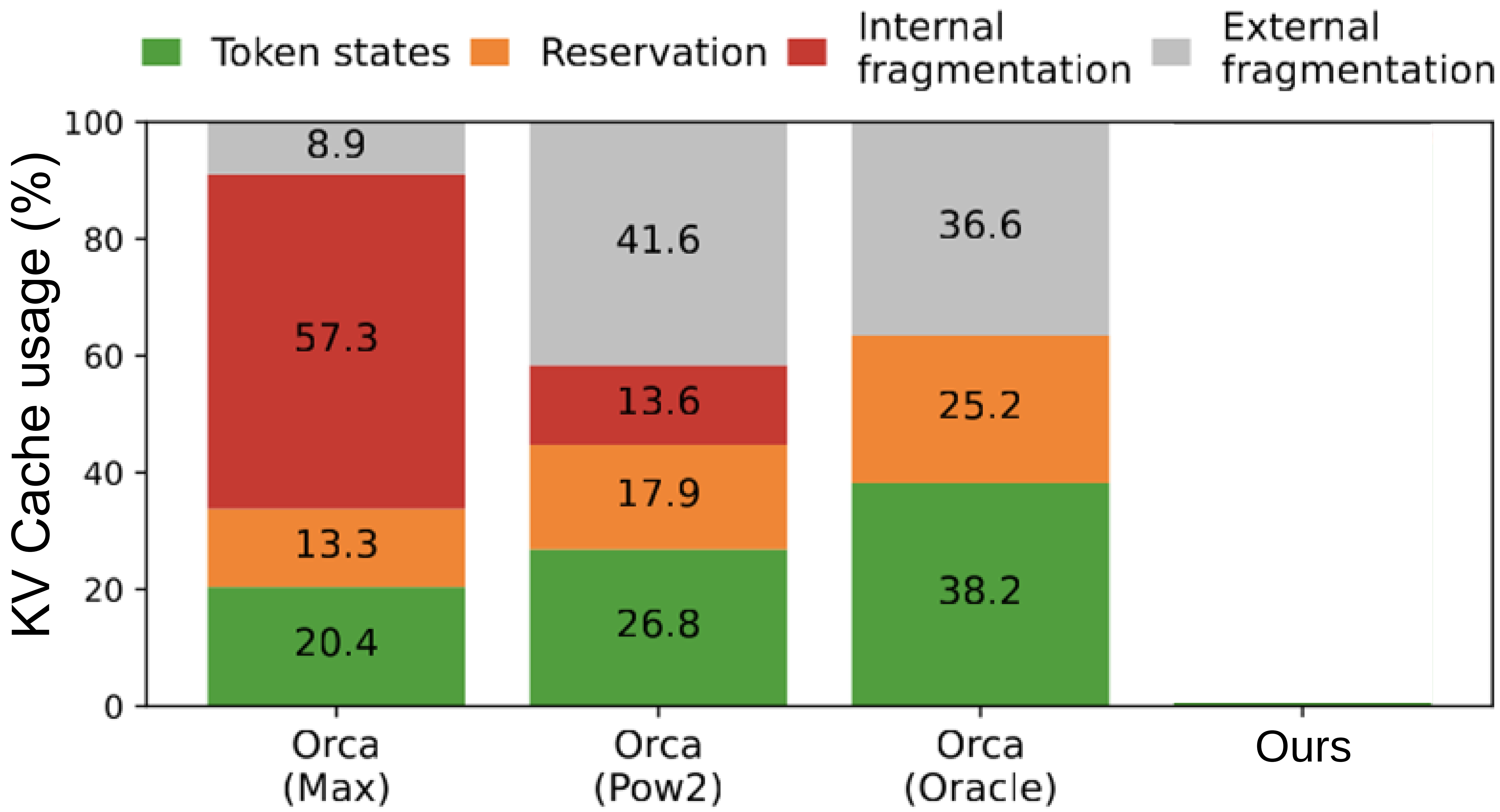
13B LLM on A100-40GB



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



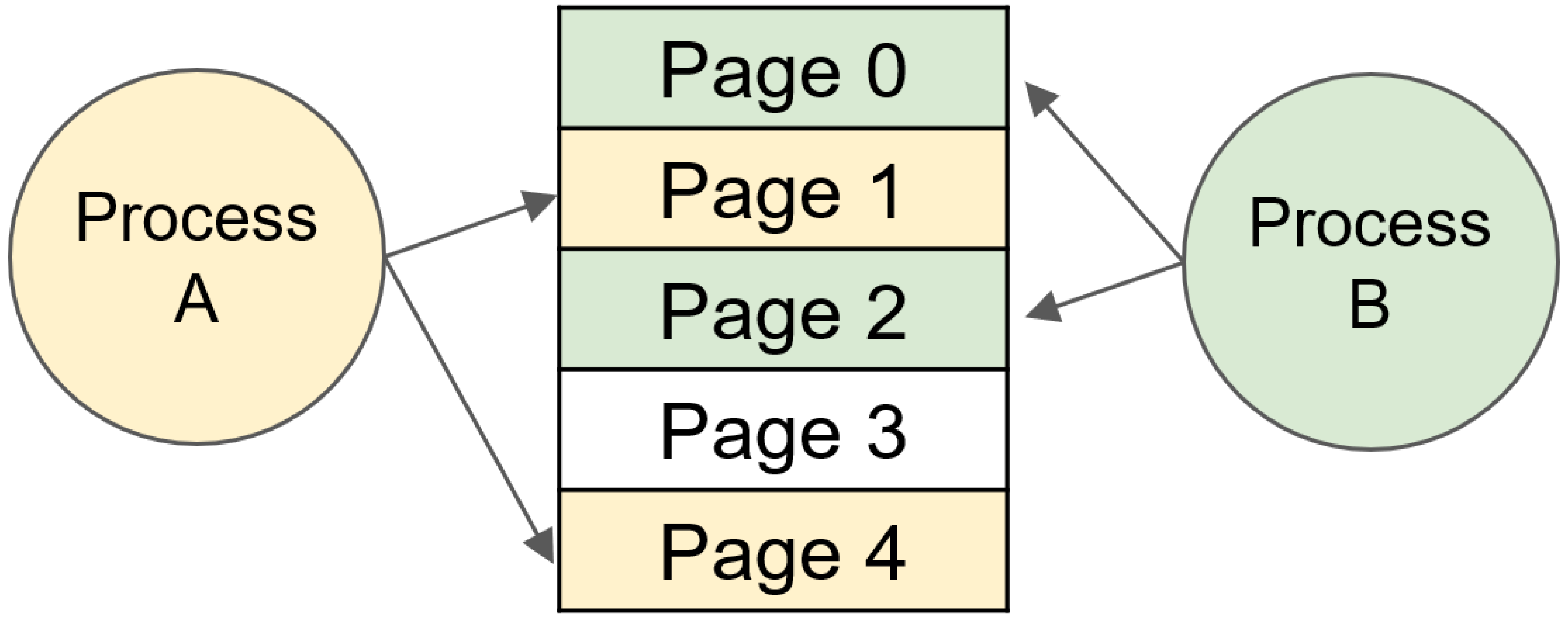
Only **20–40%** of KV cache is utilized to store token states

* 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

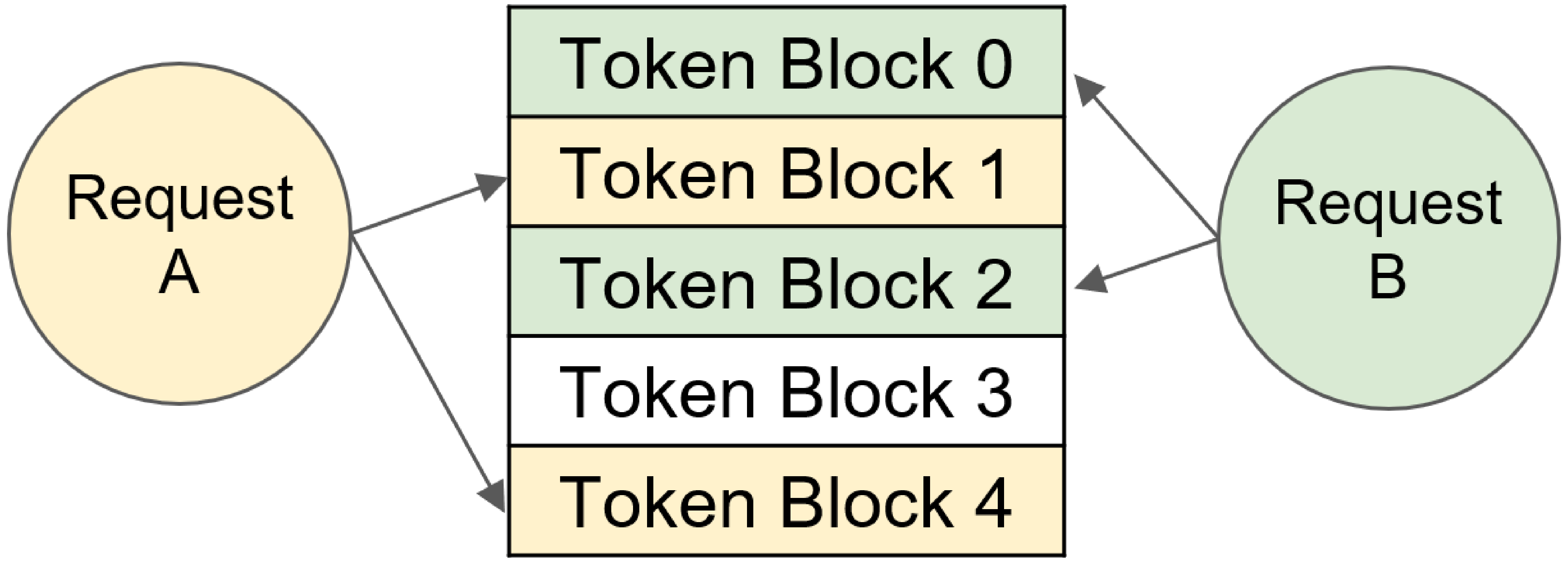
Inspired by **virtual memory** and **paging**

Memory management in OS



Physical Memory

Memory management in vLLM



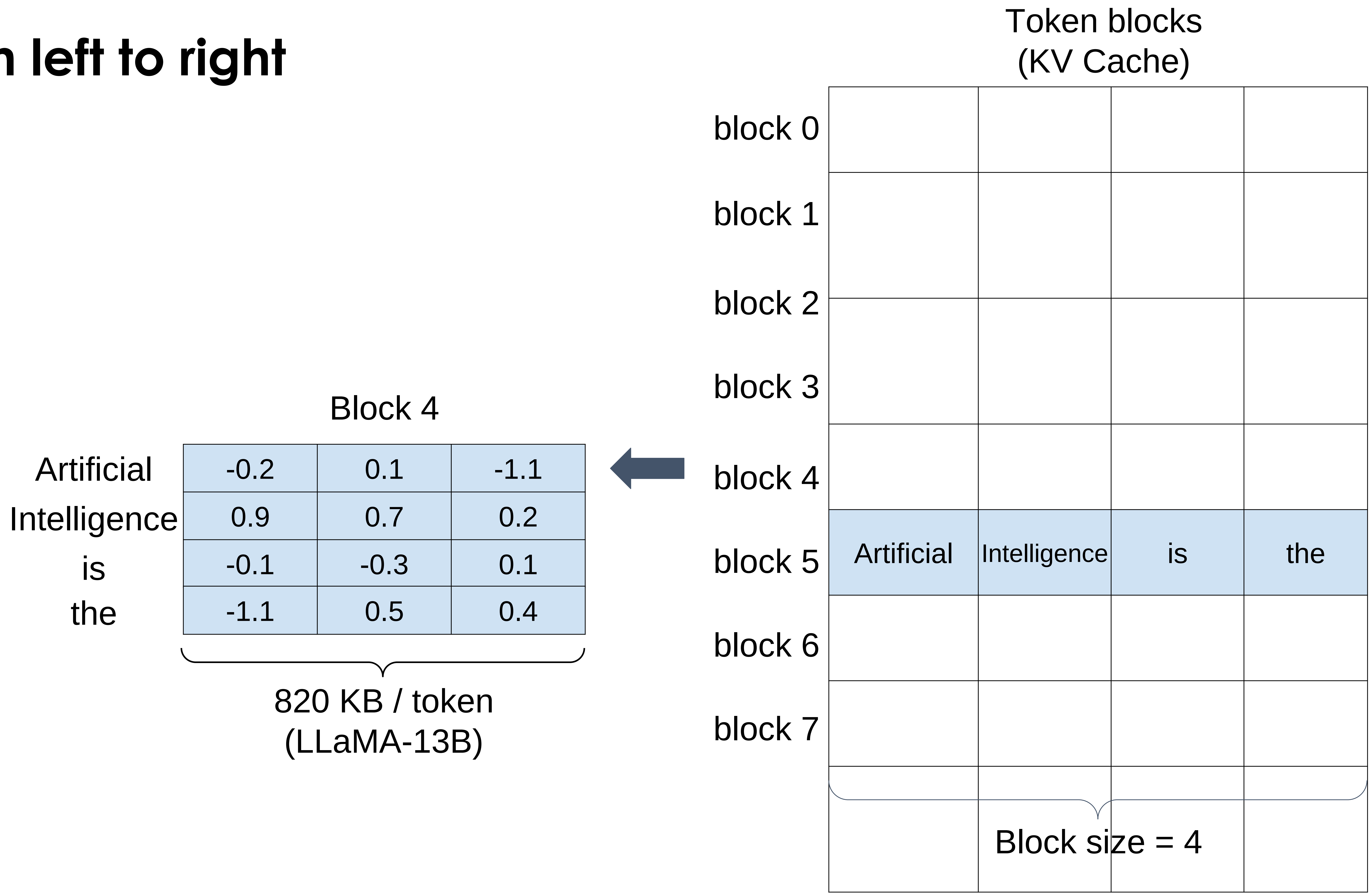
KV Cache

Token block

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

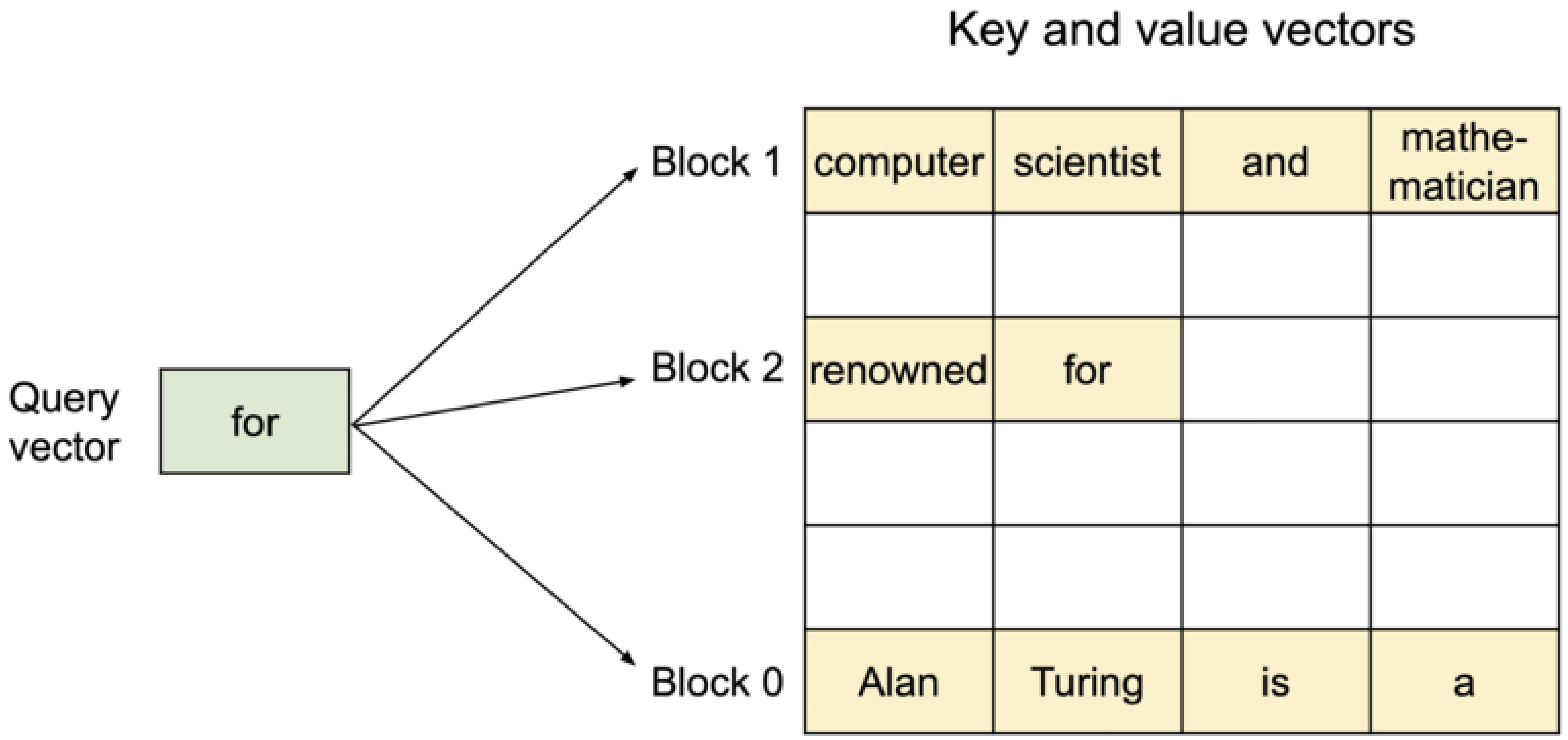
Token block

- 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: “<u>and</u>”

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: “<u>and</u>”

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: “<u>and</u>”

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 <u>mathematician</u>”

Logical token blocks

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

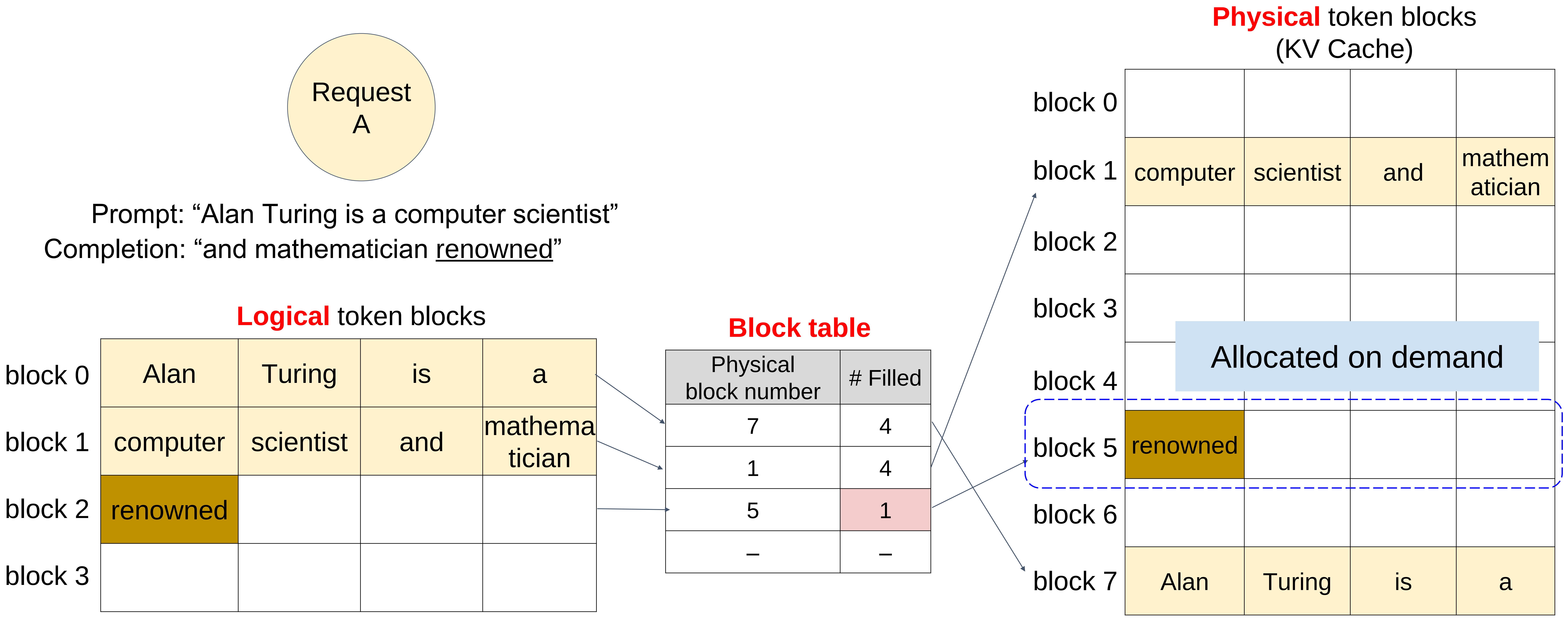
Block table

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

Physical token blocks (KV Cache)

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

Logical & physical token blocks



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

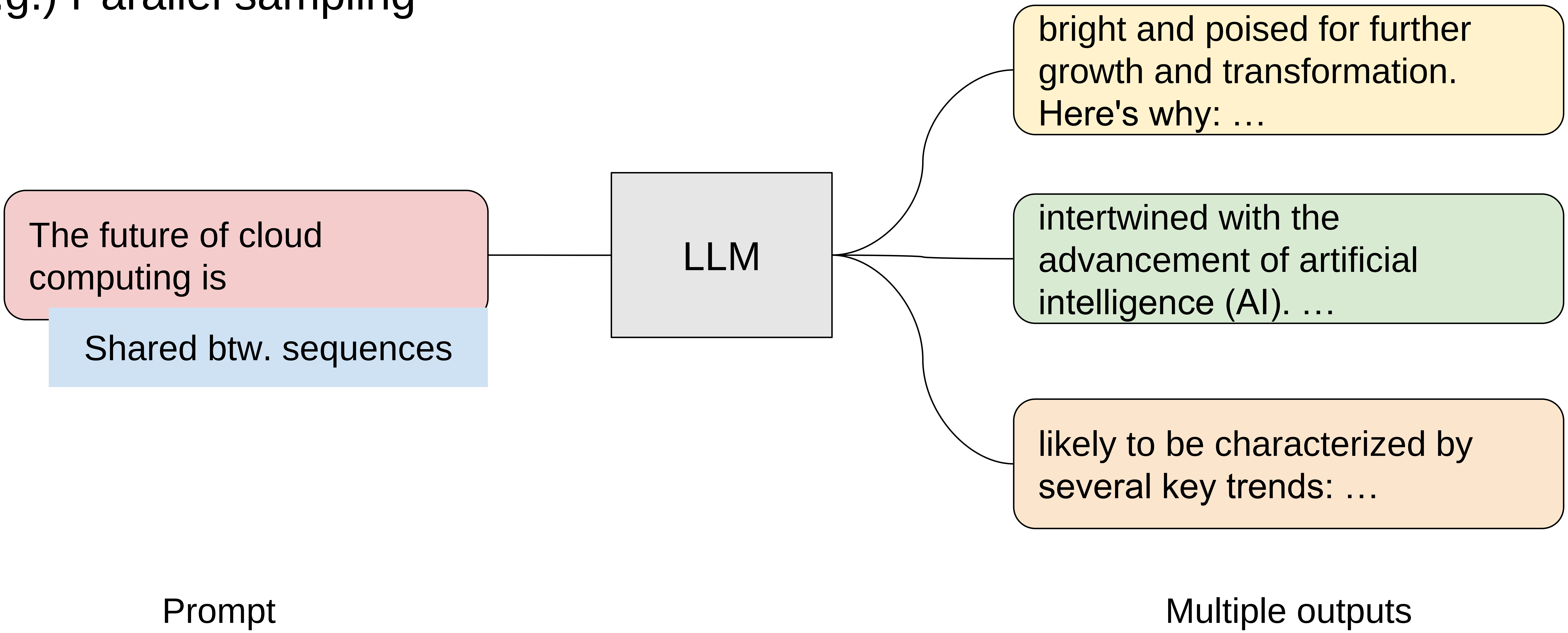
- Minimal internal fragmentation
  - Only happens at the last block of a sequence
  - **# wasted tokens / seq < block size**
    - Sequence: O(100) – O(1000) tokens
    - Block size: 16 or 32 tokens
- No external fragmentation

Alan	Turing	is	a
computer	scientist	and	mathematician
renowned			

Internal fragmentation

Dynamic block mapping enables sharing

E.g.) Parallel sampling



Sharing token blocks

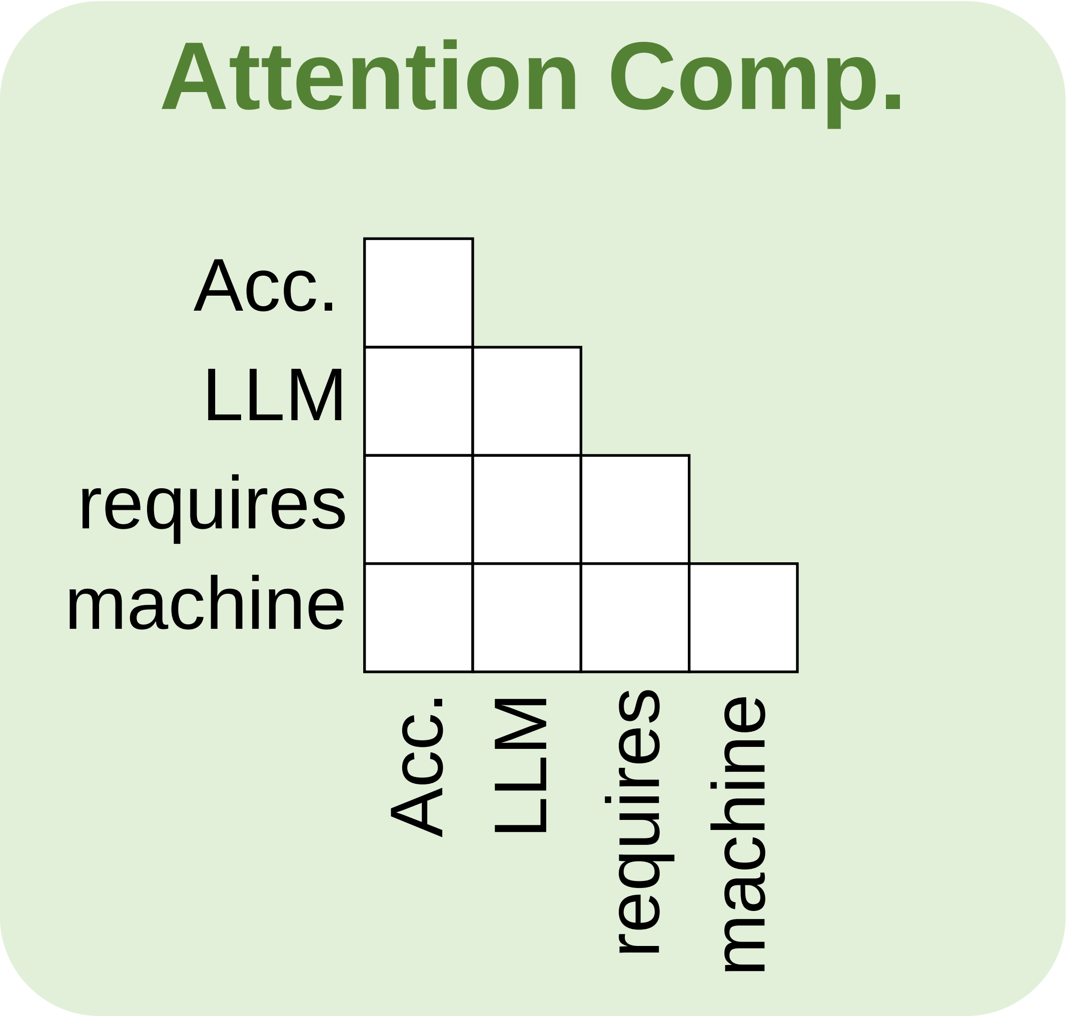
Sharing token blocks

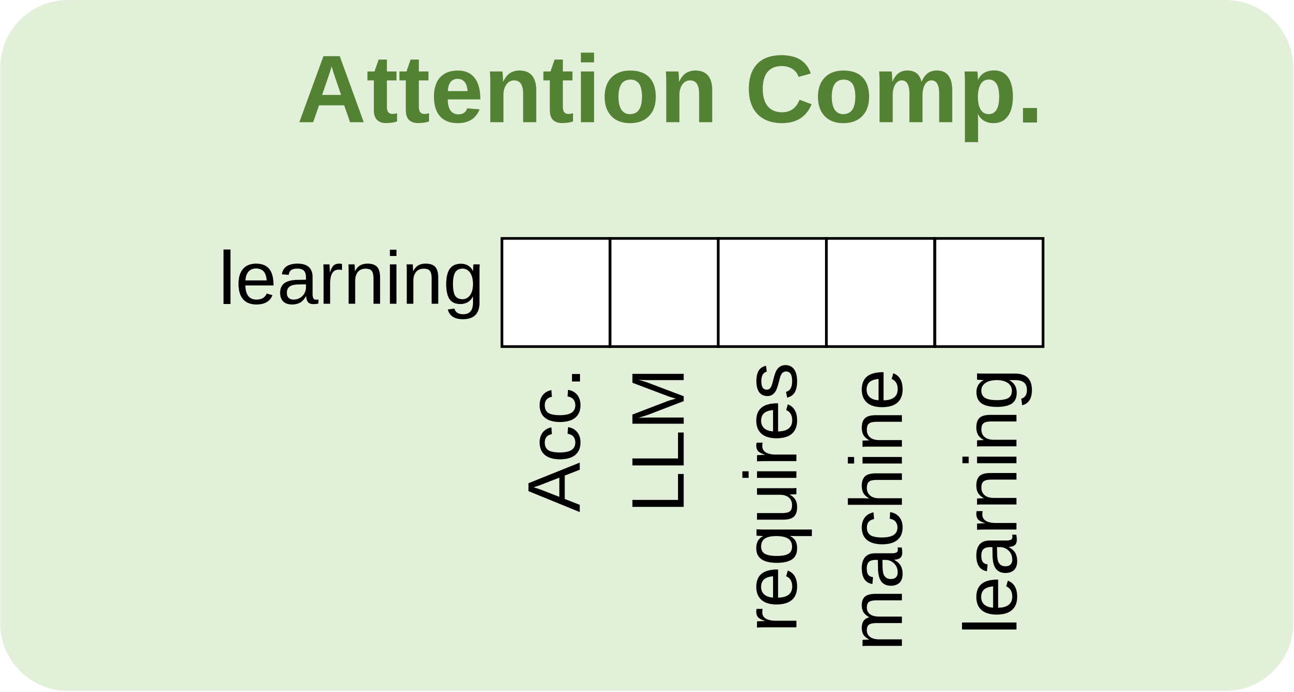
Sharing token blocks

Sharing token blocks

Sharing token blocks

Can We Apply FlashAttention to LLM Inference?





Pre-filling phase:

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

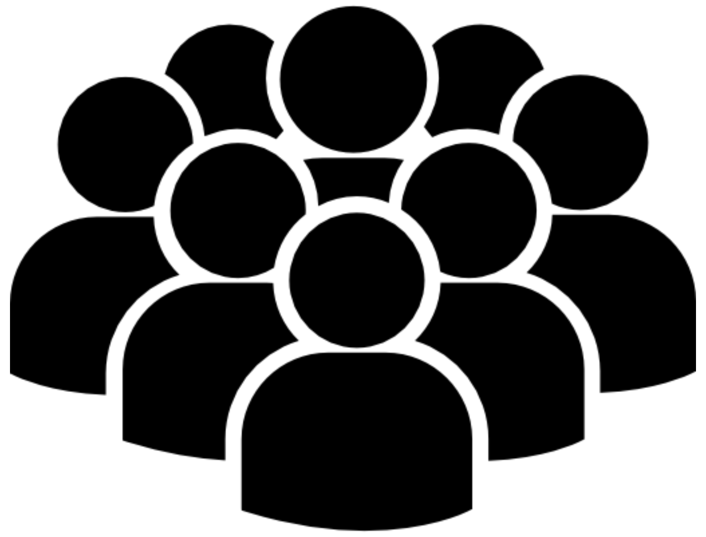
Decoding phase:

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

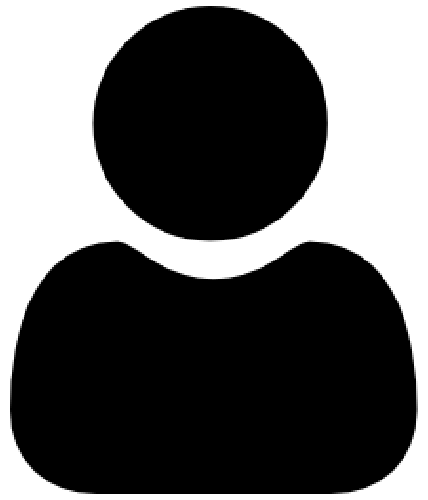
Summary: Autoregressive Decoding

- **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

Serving vs. Inference

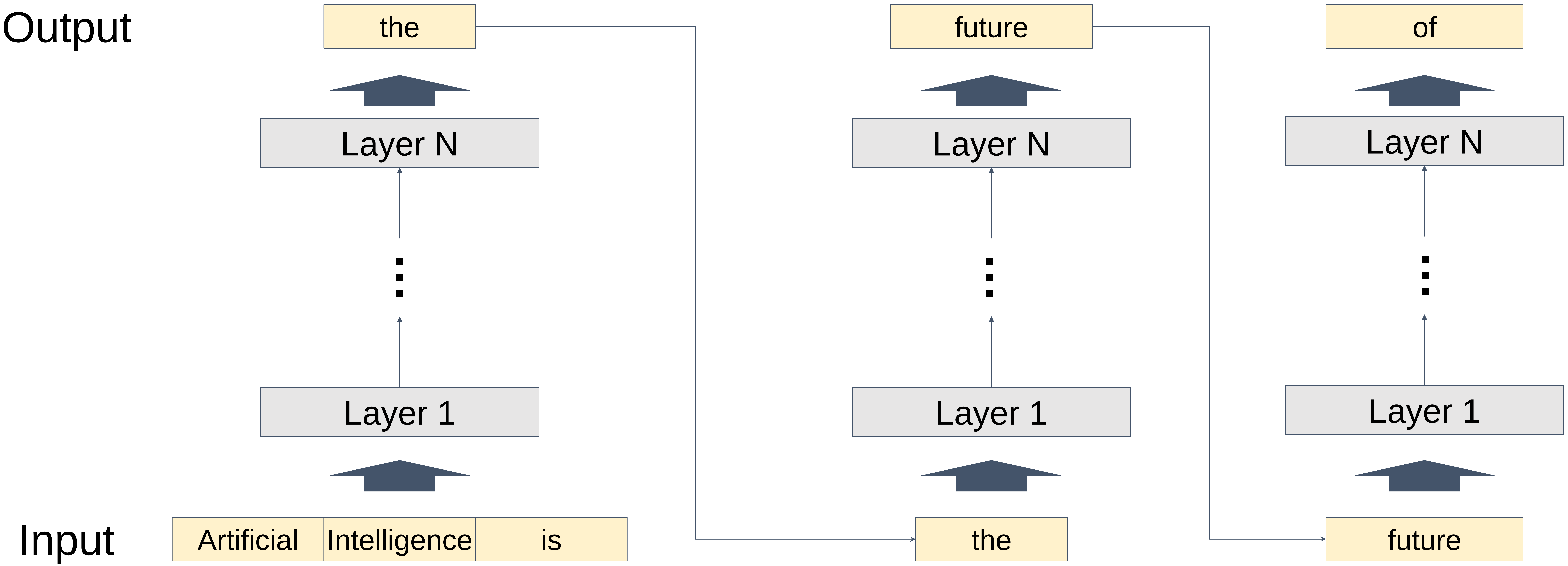


Serving: many requests, online traffic, emphasize cost-per-query



Inference: fewer request, low or offline traffic, emphasize latency

Inference process of LLMs



Repeat until the sequence

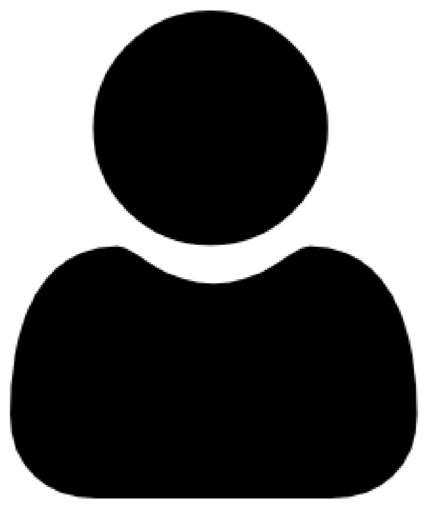
- Reaches its pre-defined maximum length (e.g., 2048 tokens)
- Generates certain tokens (e.g., “<|*end of sequence*|>”)

The Problem is harder than Thought

Even if only one request (and the system is not busy), we still *cannot* do better

Latency = step latency * # steps

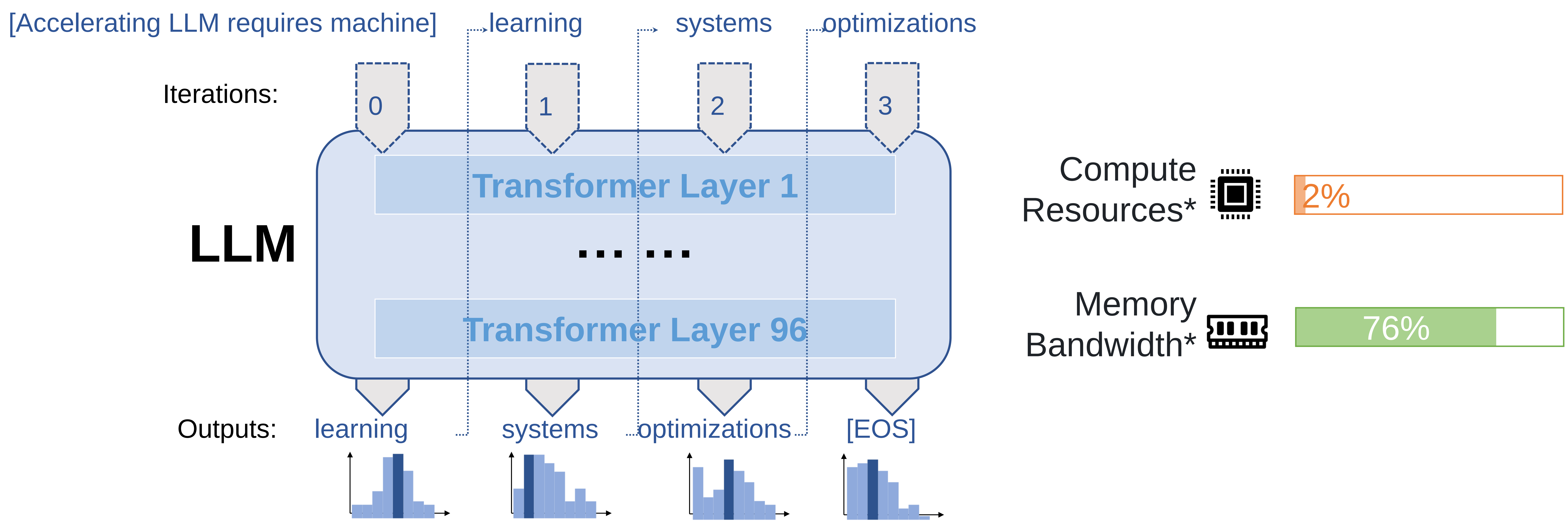
Can we do better?



Inference: fewer request

Why we cannot do better

Why we cannot do better: bottleneck



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

* Measured by serving LLAMA-2-70B on 4 A100 GPUs with 4K sequence length

Tradeoffs between Different Language Models

# Parameters	175B	13B	2.7B	760M	125M
TriviaQA	**71.2**	57.5	42.3	26.5	6.96
PIQA	**82.3**	79.9	75.4	72.0	64.3
SQuAD	**64.9**	62.6	50.0	39.2	27.5
latency	20 s	7.6s	2.7s	1.1s	**0.3s**
# A100s	10	1	1	1	**1**

Comparing multiple GPT-3 models*

Large models

Pro: better generative performance

Con: slow and expensive to serve

Small models

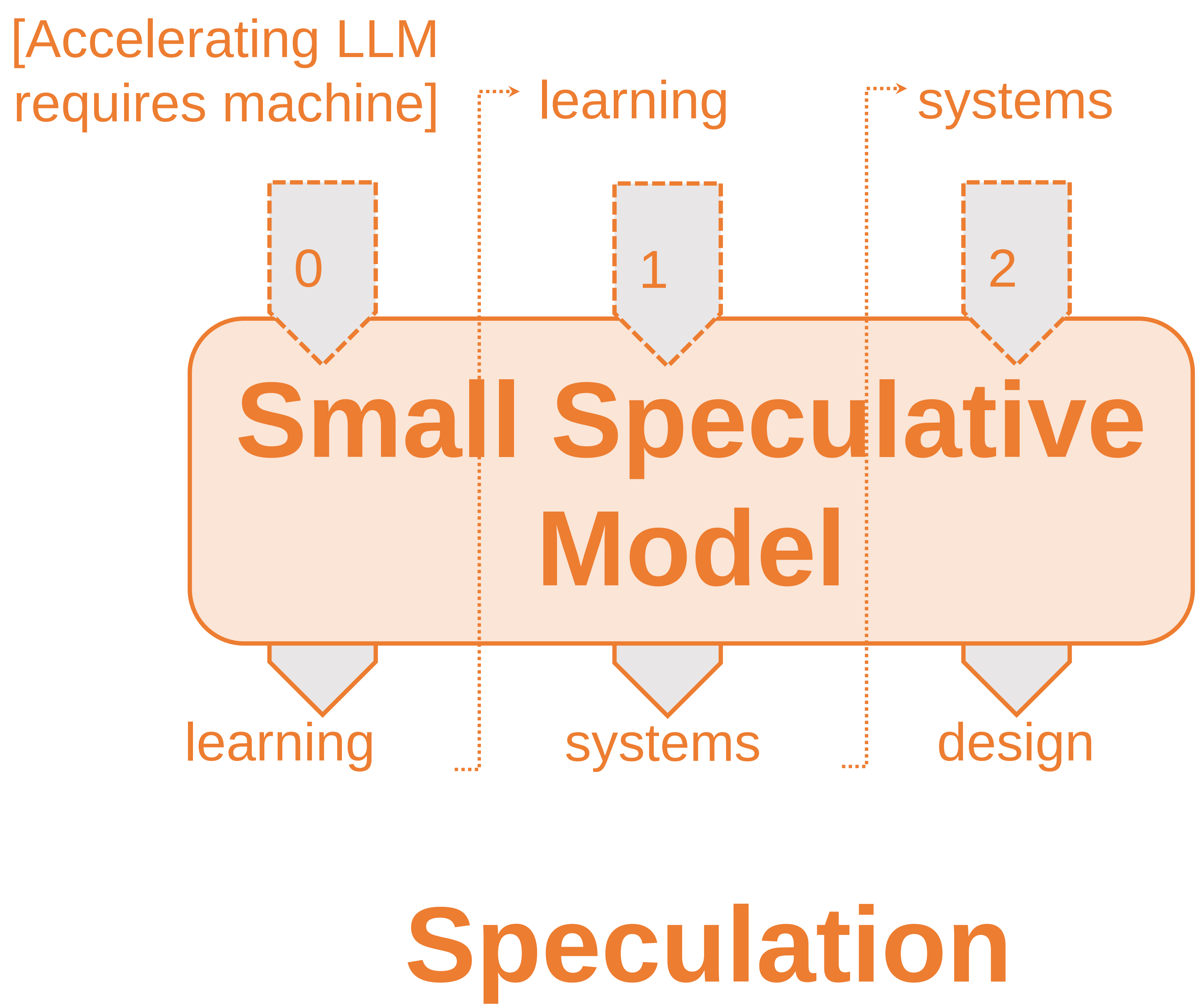
Pro: cheap and fast

Con: less accurate

* Language Models are Few-Shot Learners. Arxiv. 2005.14165

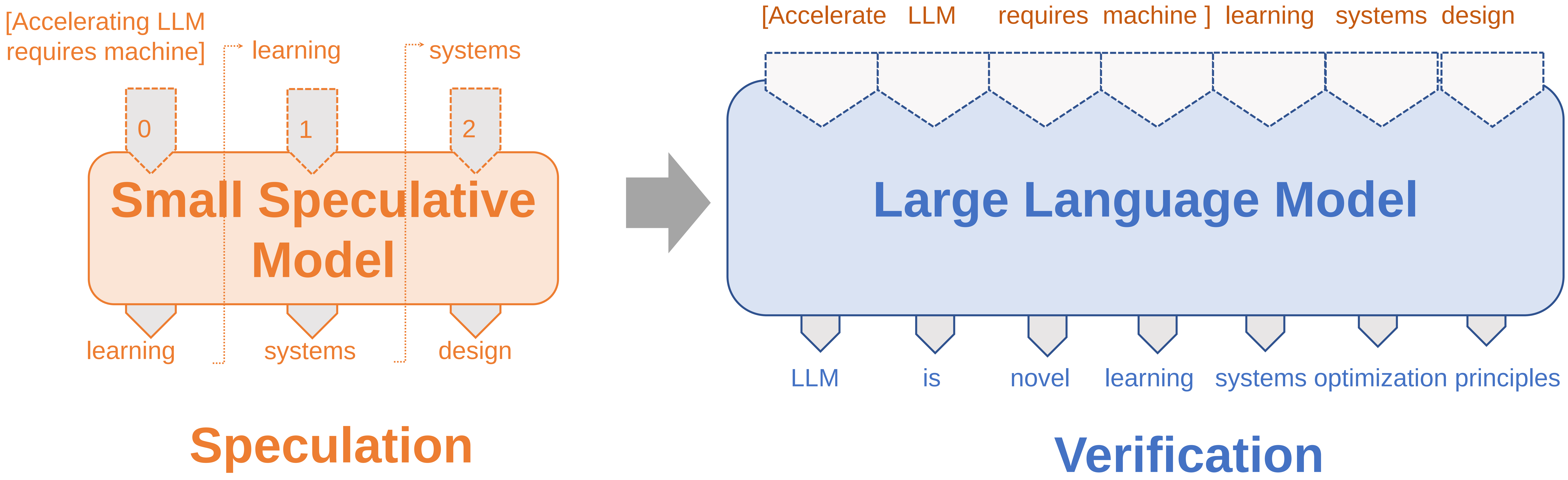
Speculative Decoding

1. Use a small speculative model (SSM) to predict the LLM's output
    - SSM runs much faster than LLM

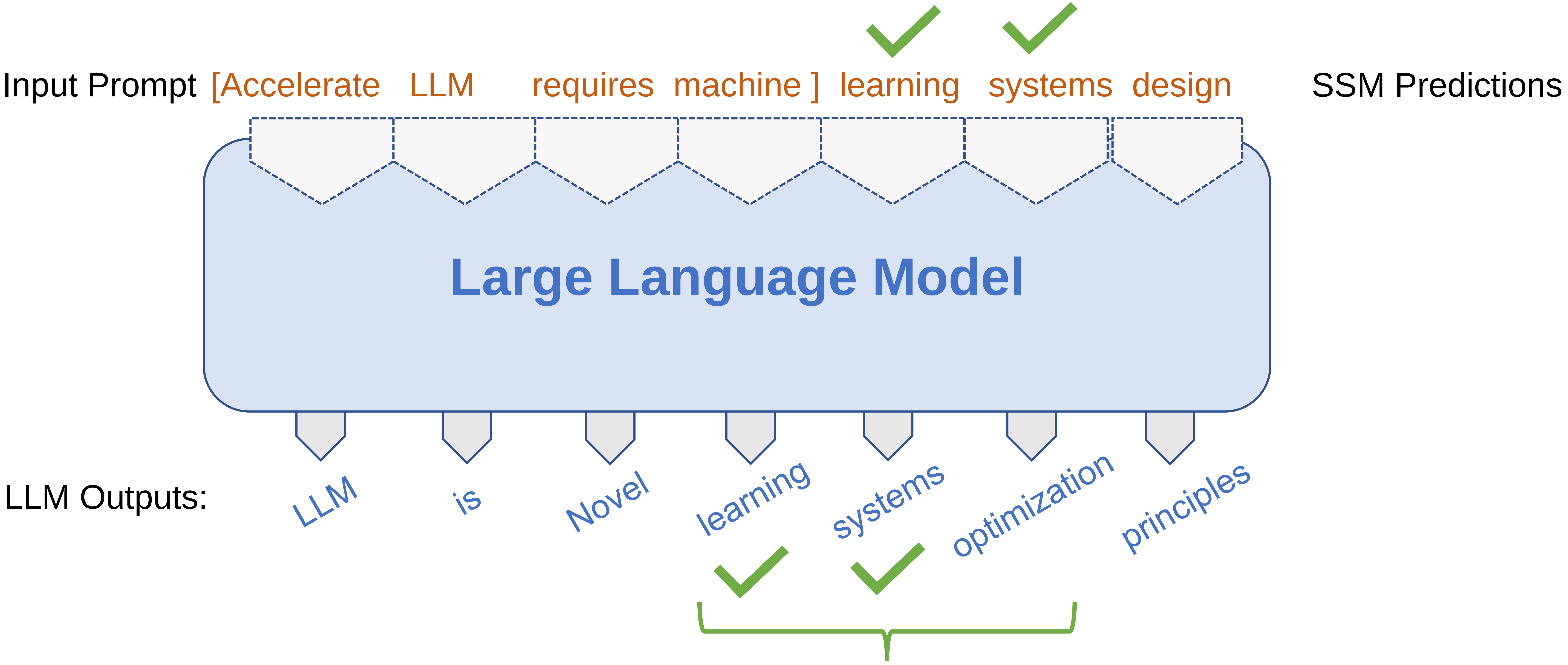


Speculative Decoding

1. Use a small speculative model (SSM) to predict the LLM's output
   - SSM runs much faster than LLM
2. Use the LLM to verify the SSM's prediction

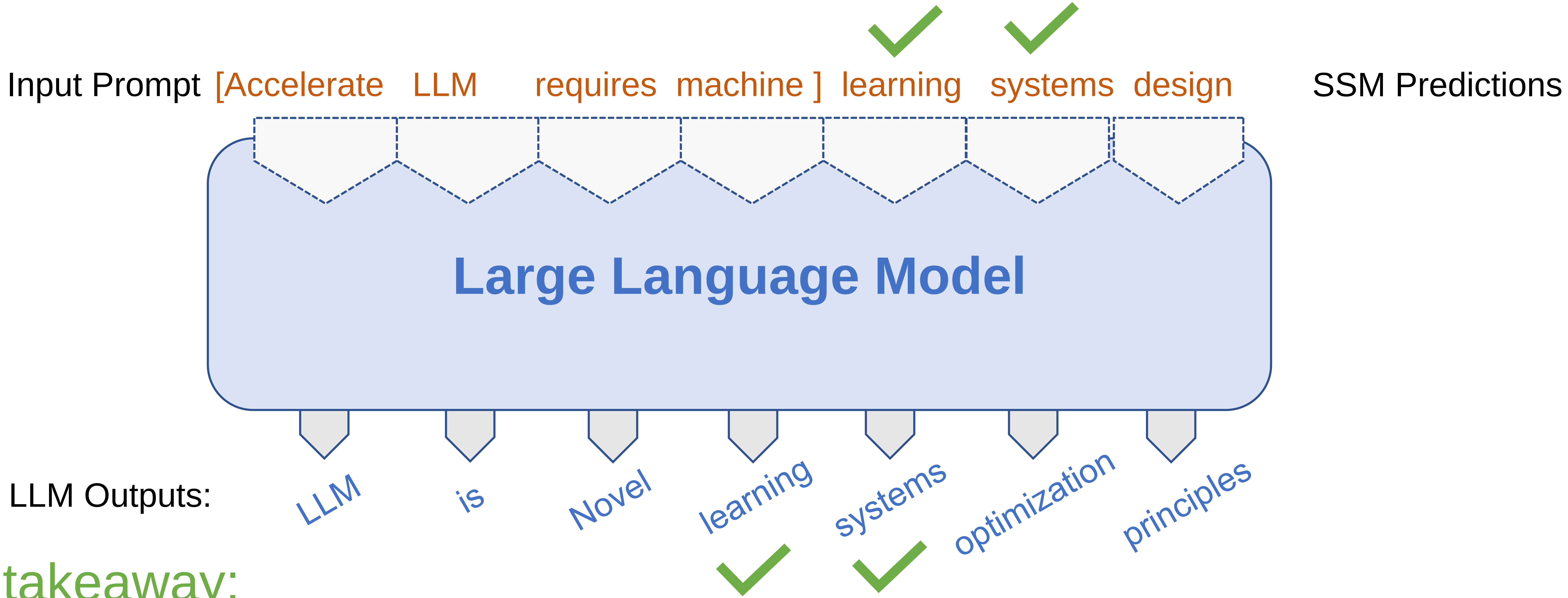


Verifying Speculative Decoding Results



Generate 3 new tokens in one LLM decoding step

Verifying Speculative Decoding Results



Key takeaway:

- LLM inference is bottlenecked by accessing model weights
- using LLM to decode multiple tokens to improve GPU utilization

A few questions

1. Can speculative decoding guarantee speedup and why?
2. When will speculative decoding bring speedup and when not?
3. Choice of draft model

Verification Algorithm

- Greedy Decoding: we already covered
- How about non-greedy decoding?
- How about verifying multiple candidates

Can we overcome autoregressive decoding?

- Self-speculative decoding
- Token-tree verification
- Medusa/Eagle: multi-head prediction
- Jacobi decoding
- Lookahead decoding

Rethink Autoregressive Decoding

Autoregressive Decoding (Greedy): decoding m tokens

$$y_i = \arg\max p_\theta(y_i \mid \mathbf{y}_{1:i-1}, \mathbf{x})$$

$$\Downarrow$$

$$\begin{cases} y_1 = \arg\max p_\theta(y_1 \mid \mathbf{x}) \\ y_2 = \arg\max p_\theta(y_2 \mid y_1, \mathbf{x}) \\ \vdots \\ y_m = \arg\max p_\theta(y_m \mid \mathbf{y}_{1:m-1}, \mathbf{x}) \end{cases}$$

Rethink Autoregressive Decoding

$\boldsymbol{x}$: prompt, $\boldsymbol{y} = [y_1, y_2, \dots, y_m]$: m tokens to decode, $p(\boldsymbol{y}|\boldsymbol{x})$: LLM distribution

Define: $f(y_i, \boldsymbol{y}_{1:i-1}, \boldsymbol{x}) = y_i - \operatorname{argmax} p(y_i|\boldsymbol{y}_{1:i-1}, \boldsymbol{x})$

$$\begin{cases} y_1 = \operatorname{argmax} p(y_1|\boldsymbol{x}) \\ y_2 = \operatorname{argmax} p(y_2|y_1, \boldsymbol{x}) \\ \quad\vdots \\ y_m = \operatorname{argmax} p(y_m|\boldsymbol{y}_{1:m-1}, \boldsymbol{x}) \end{cases} \equiv \begin{cases} f(y_1, \boldsymbol{x}) = 0 \\ f(y_2, y_1, \boldsymbol{x}) = 0 \\ \quad\vdots \\ f(y_m, \boldsymbol{y}_{1:m-1}, \boldsymbol{x}) = 0 \end{cases}$$

Autoregressive decoding

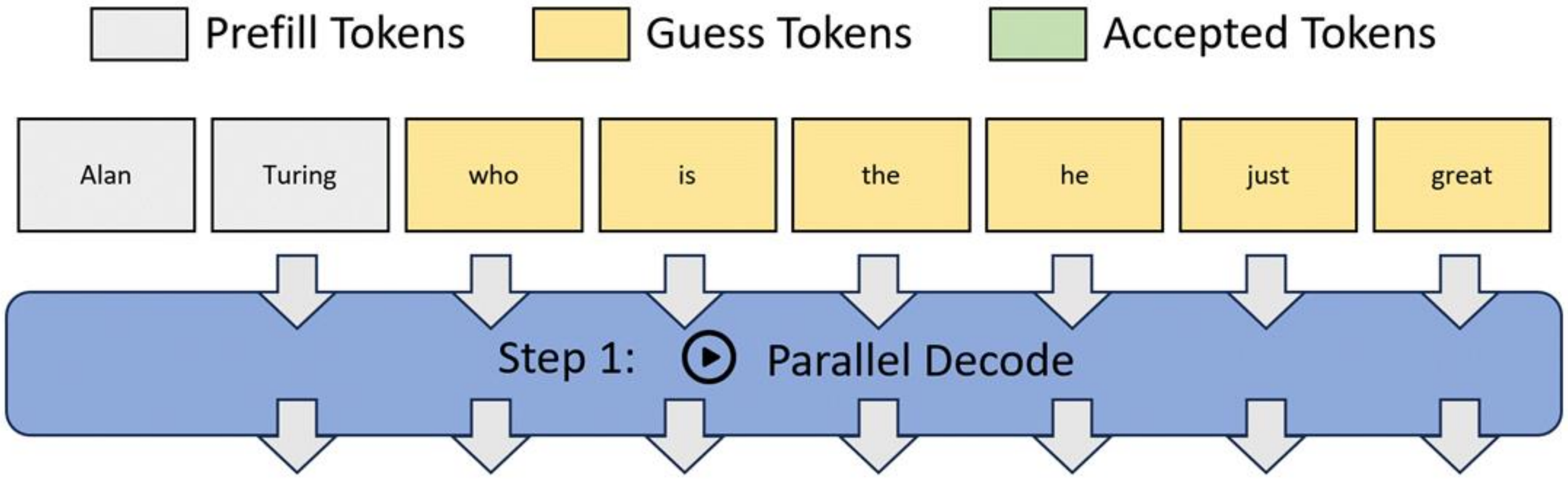
Non-linear system with m variables and m equations

One alternative: Jacobi Decoding

Algorithm 1 Jacobi decoding

Input: prompt $\mathbf{x}^0$, model p_M, generation length m
Initialize $\mathbf{y}^0 = (y_1^0, y_2^0, ..., y_m^0)$
Initialize $\mathbf{y}^{output} \leftarrow ()$
for $i = 1$ **to** m **do**
    $\mathbf{y}_{1:m}^i \leftarrow \text{argmax}(P_M(\mathbf{y}_{1:m}^i | \mathbf{y}_{1:m}^{i-1}, \mathbf{x}^0))$
    $\mathbf{o} \leftarrow \mathbf{y}^i$
    $stop \leftarrow$ STOPCONDITION$(\mathbf{y}^i, \mathbf{y}^{i-1})$
    if $stop$ **then**
        break
    end if
end for
Output: $\mathbf{o} = (y_1, y_2, ..., y_m)$

Jacobi Decoding Illustrated



Total Steps: 0
Total Accepted Tokens: 0

What are the trade-offs in Jacobi decoding?

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

Recall A few Important Problems (will be HW3)

- 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?

Motivation of Scaling Laws

- We have locked on transformers-based LLMs
  - Assuming you know the answers of the previous 3 Qs
  - Given a model architecture with #params, we know #flops, amount of memory given a model size
- 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

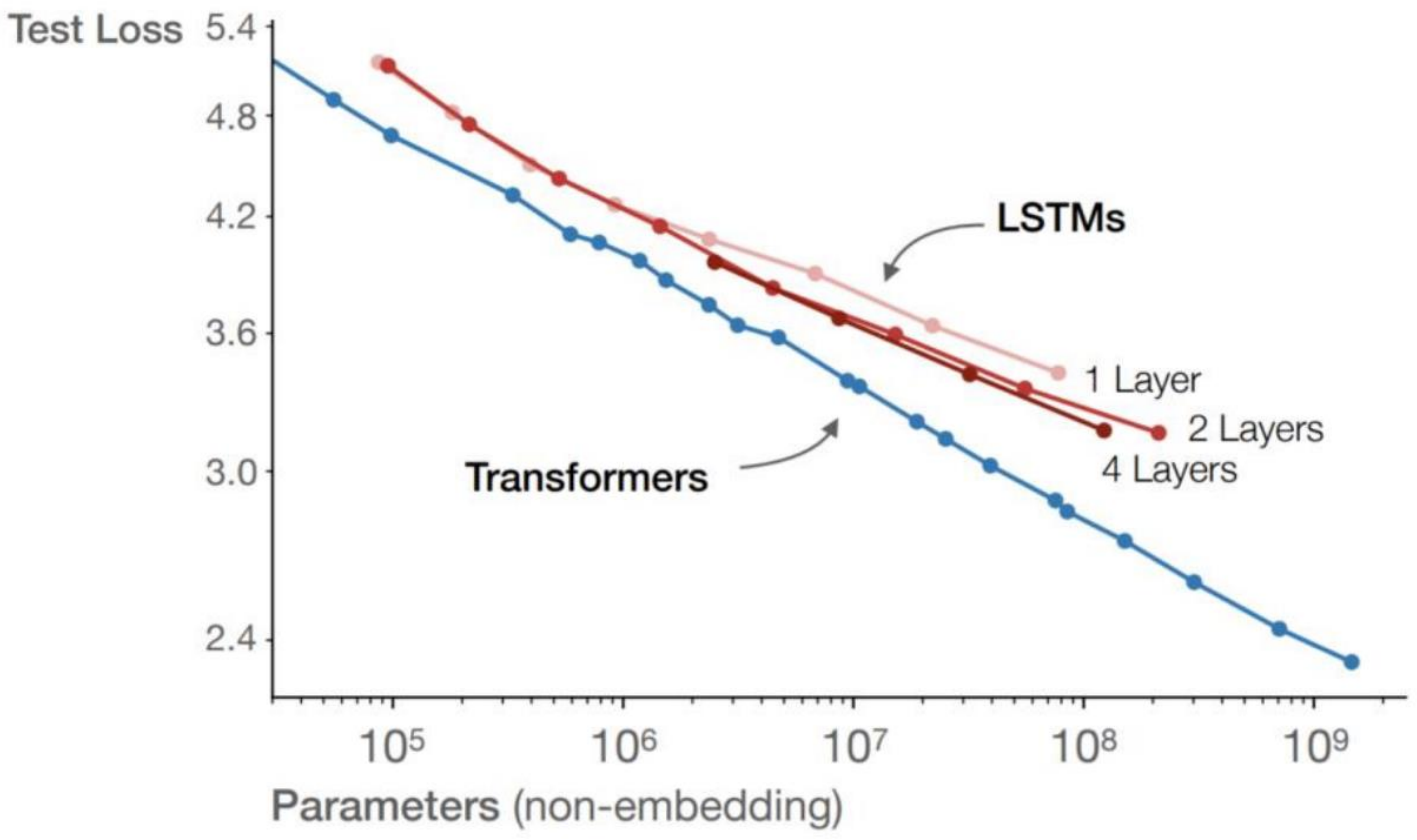
- Can we do this for transformers LLMs?

Model Scaling Laws

- Problem: How can we efficiently design huge LLMs?
  - LSTMs vs transformers
  - Adam vs SGD
- Problem: how should we allocate our limited resources:
  - Train models longer vs train bigger models?
  - Collect more data vs get more GPUs?

Transformers vs LSTMs

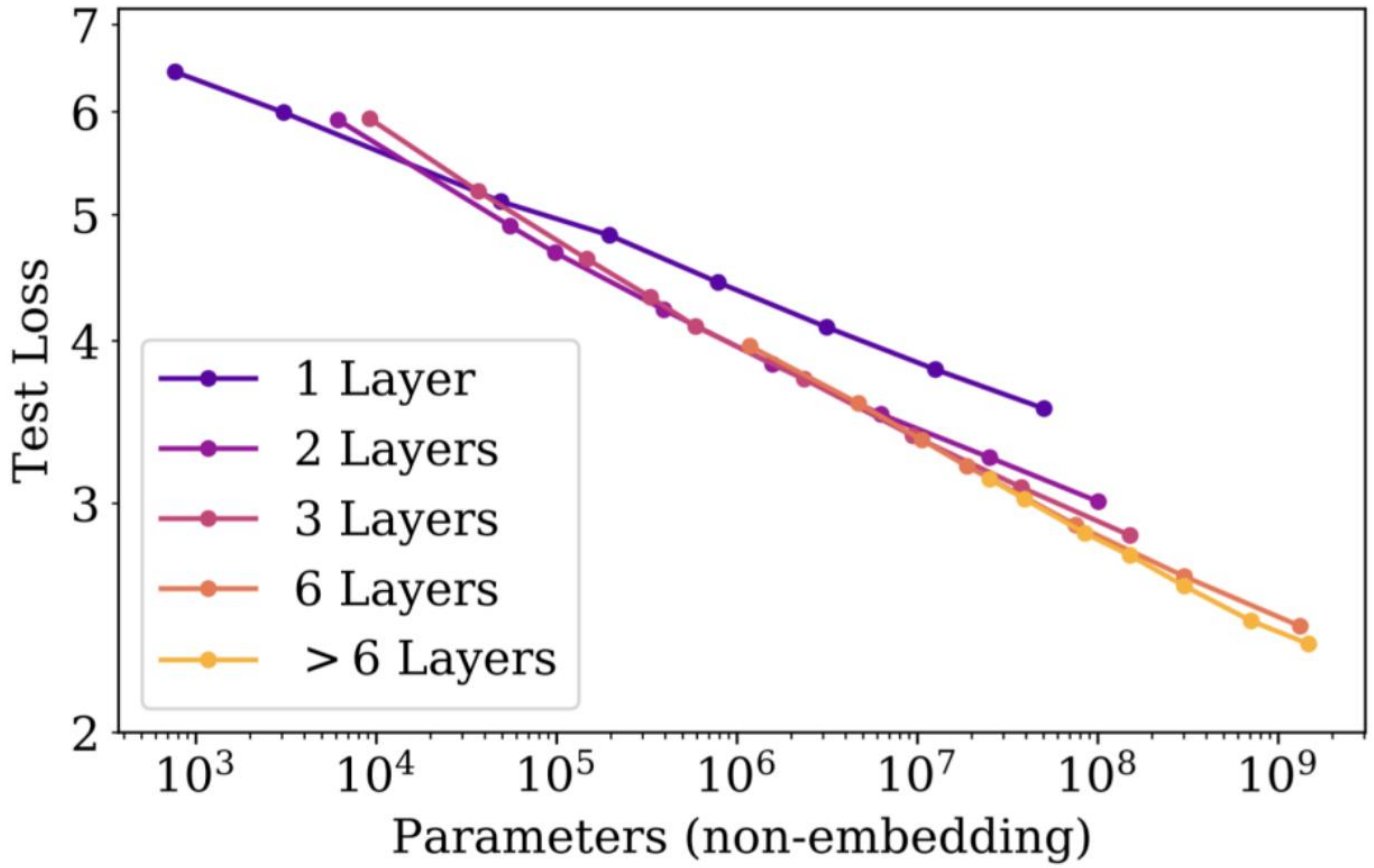
- Q: Are transformers better than LSTMs?
  - Brute force way: spend tens of millions to train a LSTM GPT-3
- Scaling law way:



[Kaplan+ 2021]

Number of Layers

- Does depth or width make a huge difference?
  - 1 vs 2 layers makes a huge difference. • More layers have diminishing returns below 107 params



The Scaling law way

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

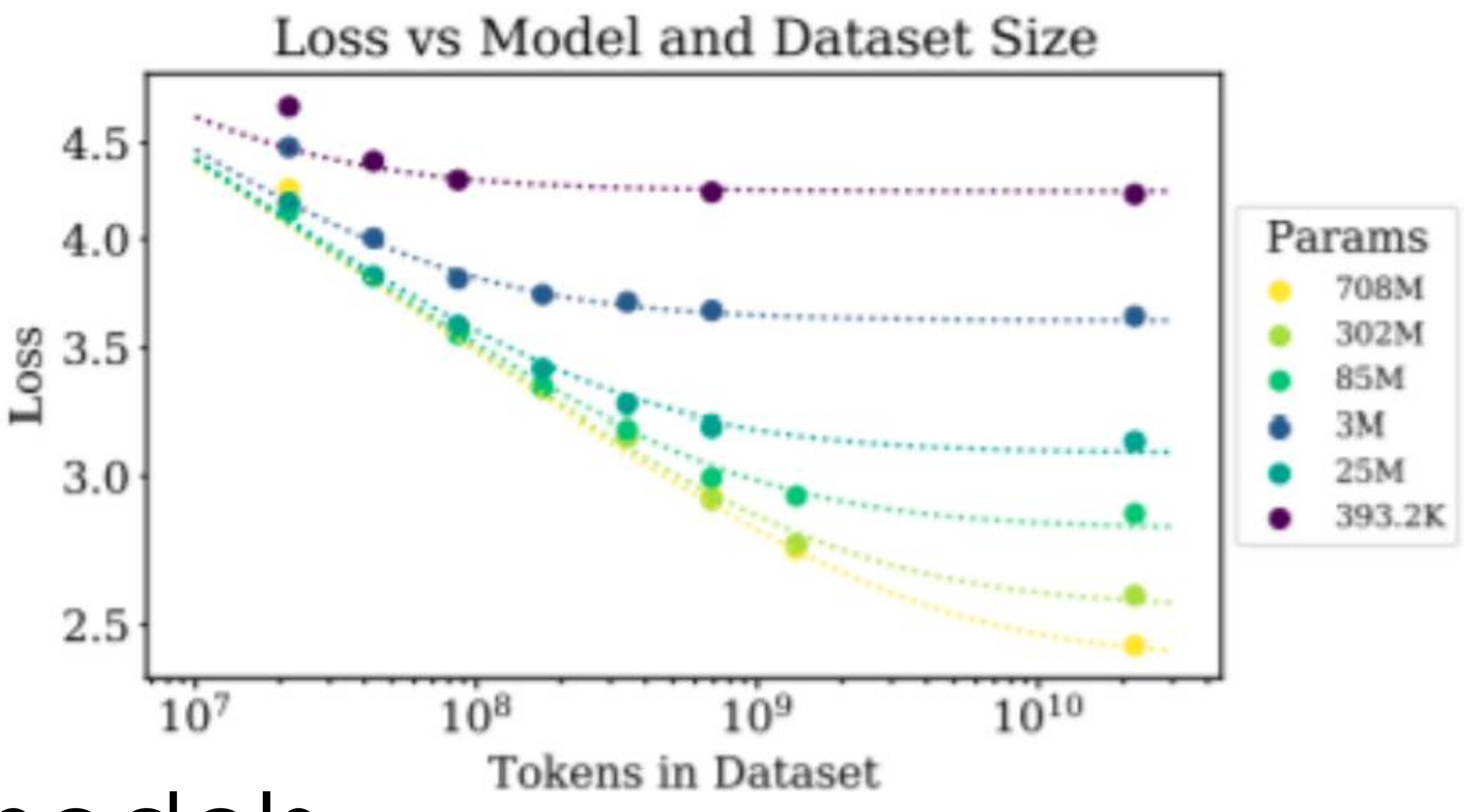
- Rationale
  - The effect of hyperparameters on big LMs can be predicted before training!
    - Optimizer choice
    - Model Depth
    - Arechitecture choice

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: model size data joint scaling

Model size data joint scaling



- Do we need more data or bigger model:
  - 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

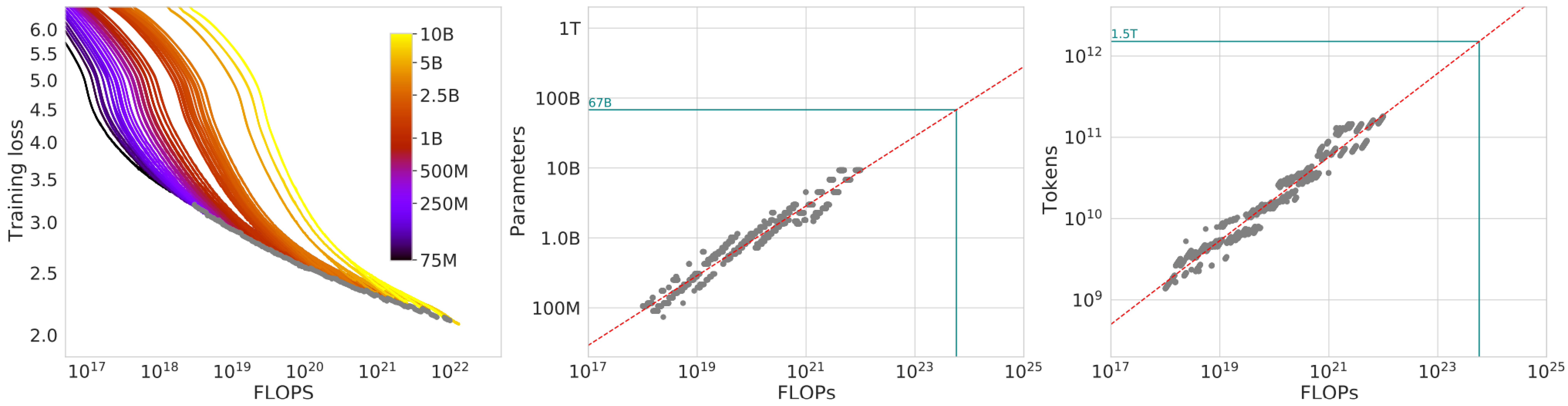


Figure 2 | **Training curve envelope.** On the **left** we show all of our different runs. We launched a range of model sizes going from 70M to 10B, each for four different cosine cycle lengths. From these curves, we extracted the envelope of minimal loss per FLOP, and we used these points to estimate 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}).

Final Remarks

- Scaling law: the physics behind LLMs
- Scaling law also represents a research approach transition:
  - Stats and theoretical analysis -> empirical laws
  - Exploration of different model architectures -> Scaling transformers
  - ML systems become essential

Recall A few Important Problems (will be HW3)

- 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