Efficient Memory Management for Large Language Model Serving with PagedAttention

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, Ion Stoica

Presented by Yuqi Xue

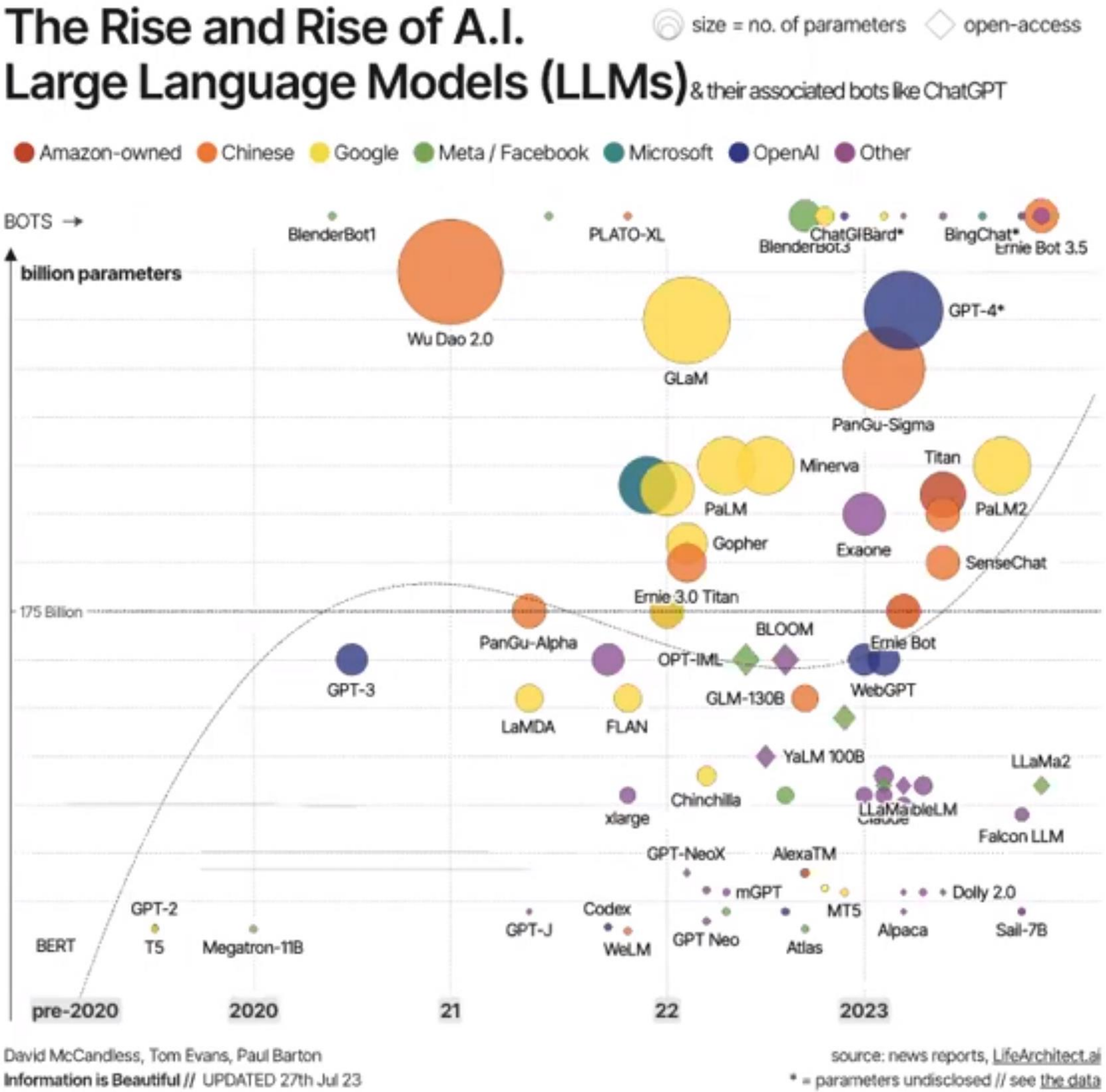
Feb 15, 2024

Paper Published at SOSP'23. Many slide pages are borrowed from the MLSys Seminar @SG talk: https://www.youtube.com/watch?v=Oq2SN7uutbQ.

The Era of LLMs

ChatGPT

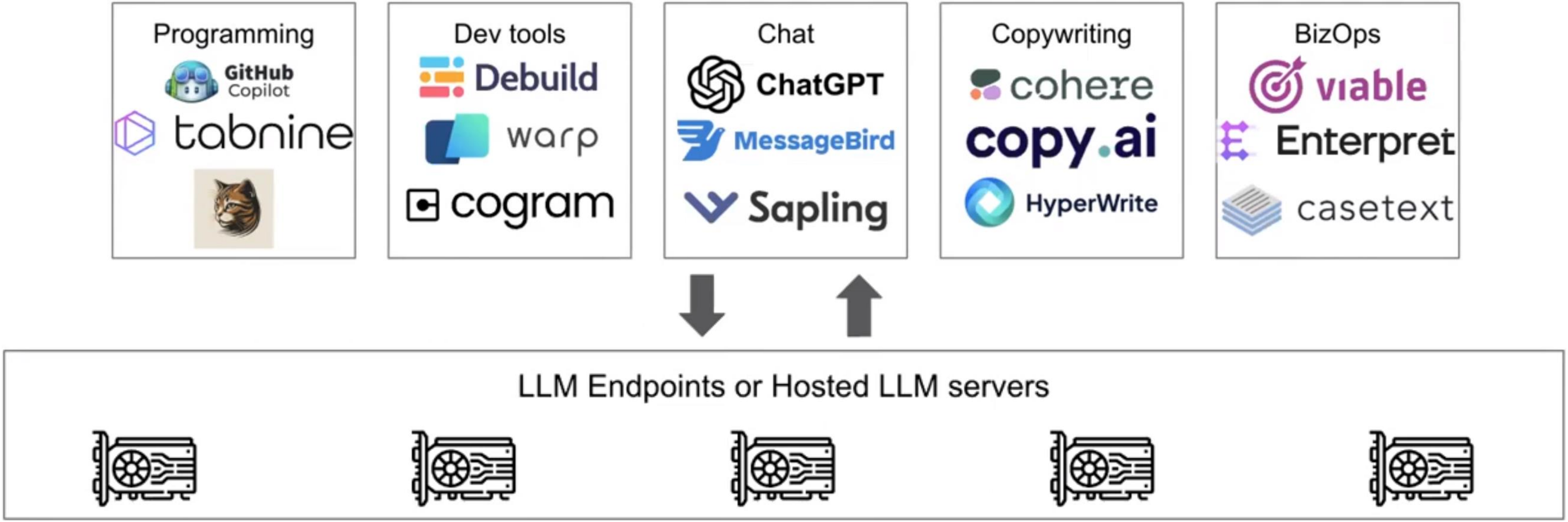
GitHub Copilot



LLM-Powered Services

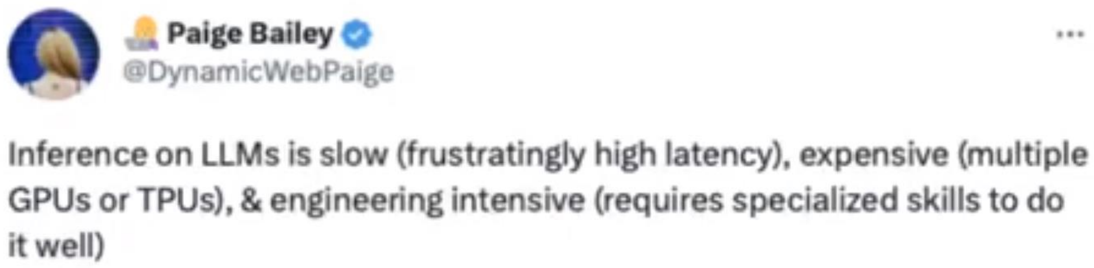
Programming	Dev tools	Chat	Copywriting	BizOps
GitHub Copilot	Debuild	ChatGPT	cohere	viable
tabnine	warp	MessageBird	copy.ai	Enterpret
	cogram	Sapling	HyperWrite	casetext

LLM-Powered Services



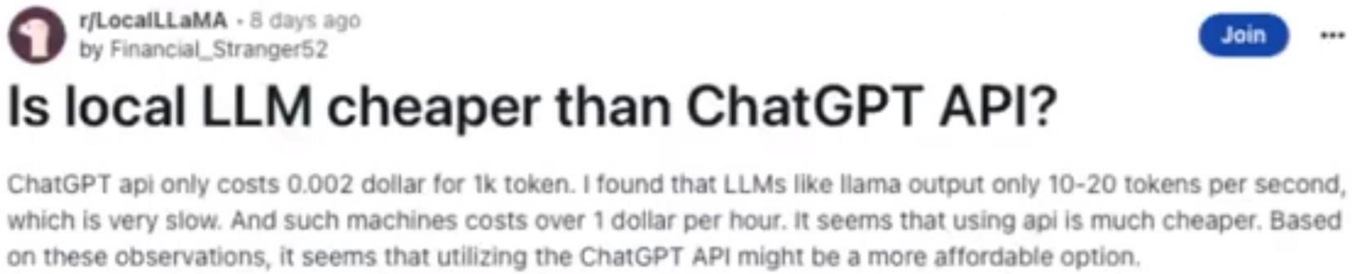
Serving LLMs is extremely expensive

- LLMs run on high-end GPUs such as NVIDIA A100
- Each GPU can only serve a handful of requests per second
  - For LLaMA-13B and moderate-size inputs, one A100 can process < 1 requests per second
- A ton of GPUs are required for production-scale LLM services



Paige Bailey
@DynamicWebPaige

Inference on LLMs is slow (frustratingly high latency), expensive (multiple GPUs or TPUs), & engineering intensive (requires specialized skills to do it well)



r/LocalLLaMA · 8 days ago
by Financial_Stranger52

Join

Is local LLM cheaper than ChatGPT API?

ChatGPT api only costs 0.002 dollar for 1k token. I found that LLMs like llama output only 10-20 tokens per second, which is very slow. And such machines costs over 1 dollar per hour. It seems that using api is much cheaper. Based on these observations, it seems that utilizing the ChatGPT API might be a more affordable option.

Inference process of LLMs

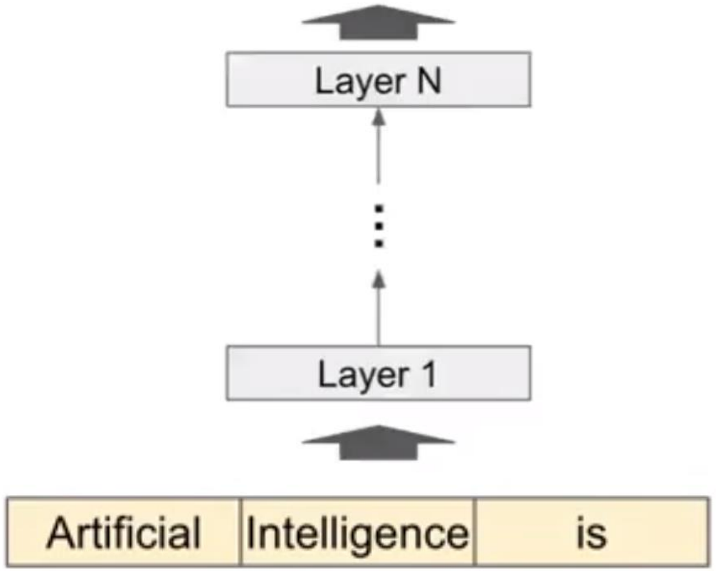
Output

Input

Artificial	Intelligence	is

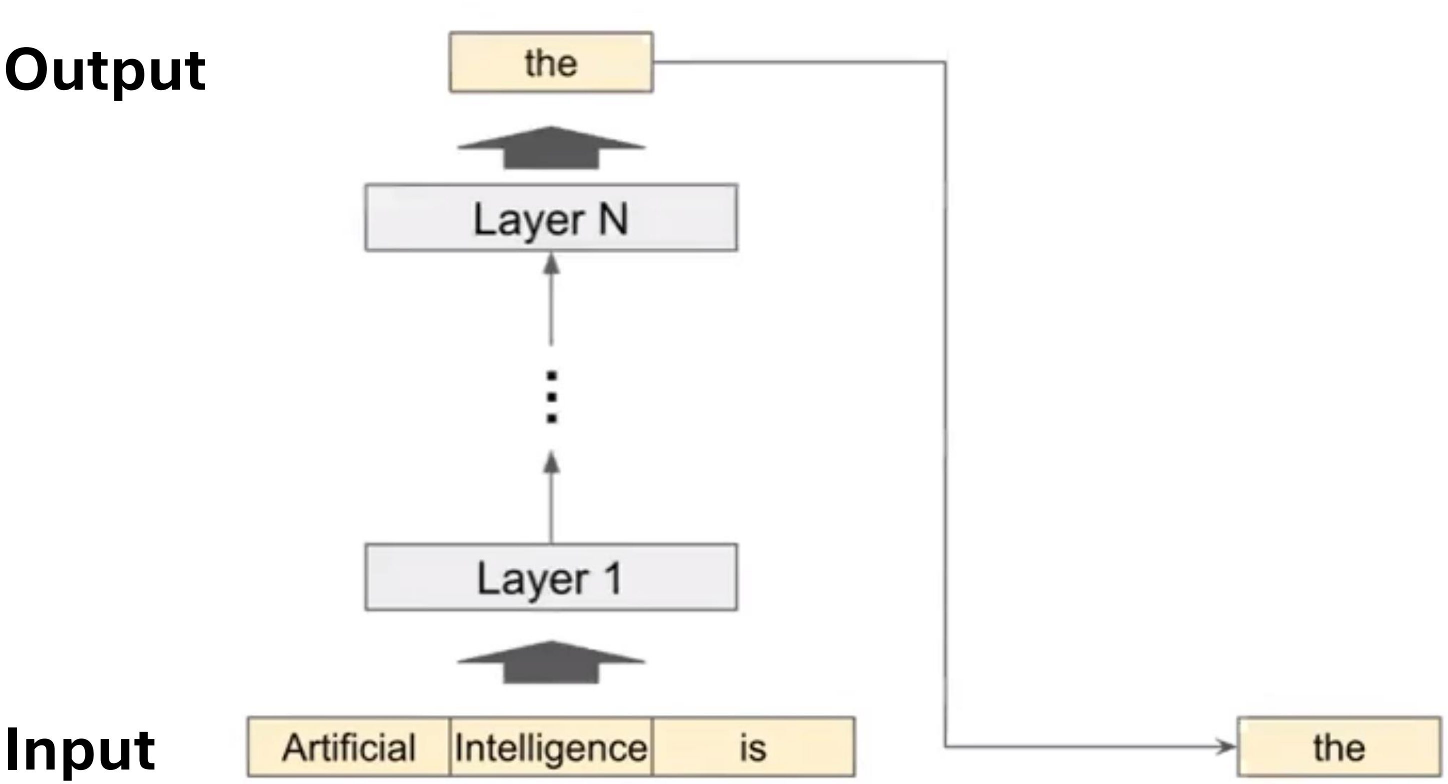
Inference process of LLMs

Output

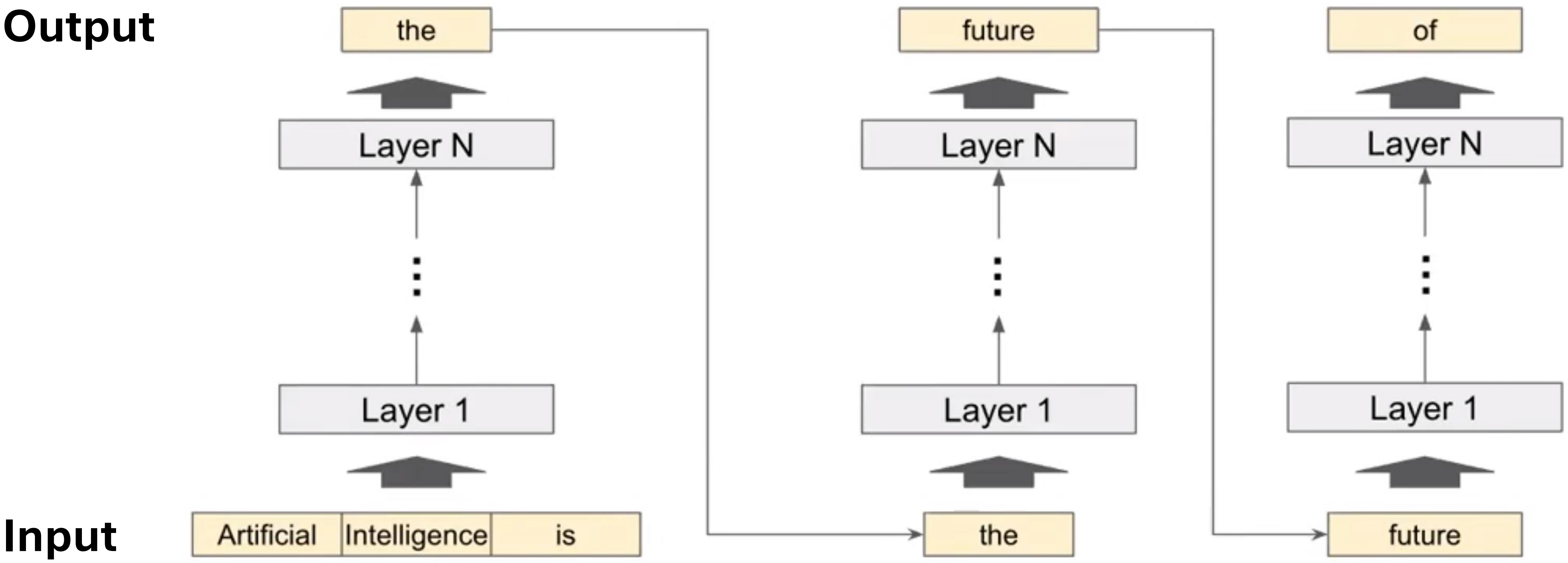


Input

Inference process of LLMs

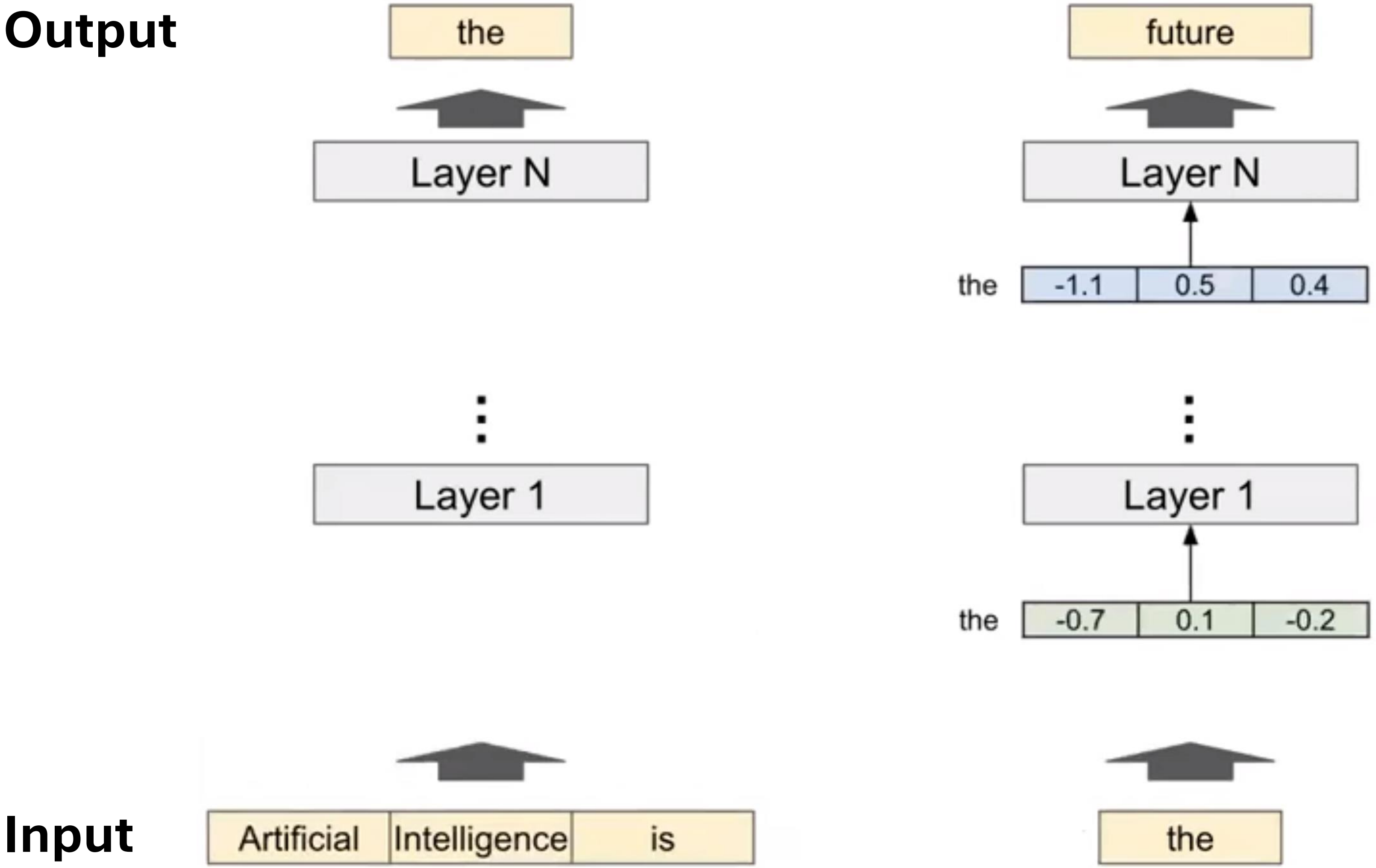


Inference process of LLMs

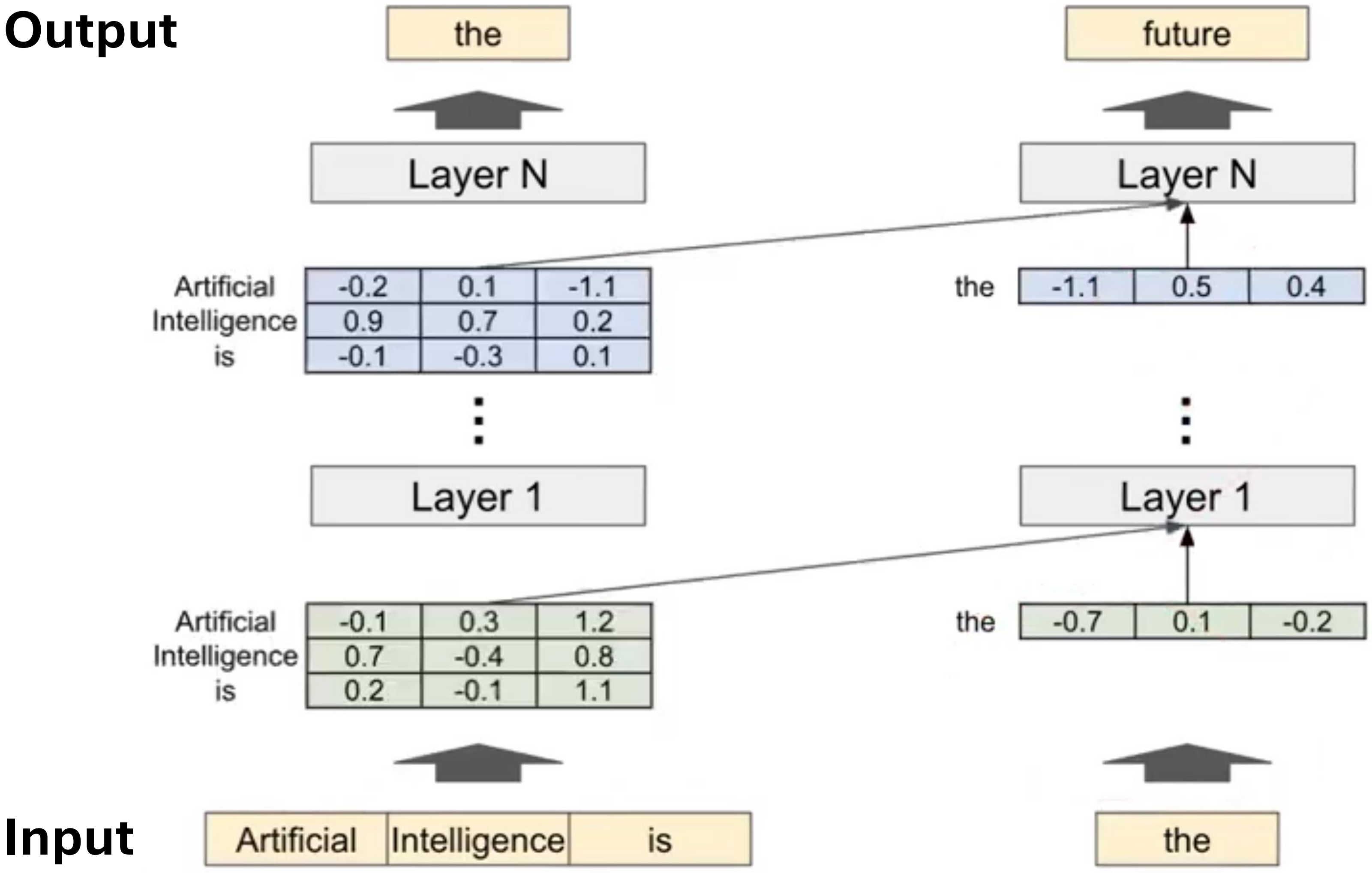


- Repeat until the sequence
  - Reaches its pre-defined max length (e.g., 4K tokens)
  - Generates an EOS (end of sequence) token

KV Cache

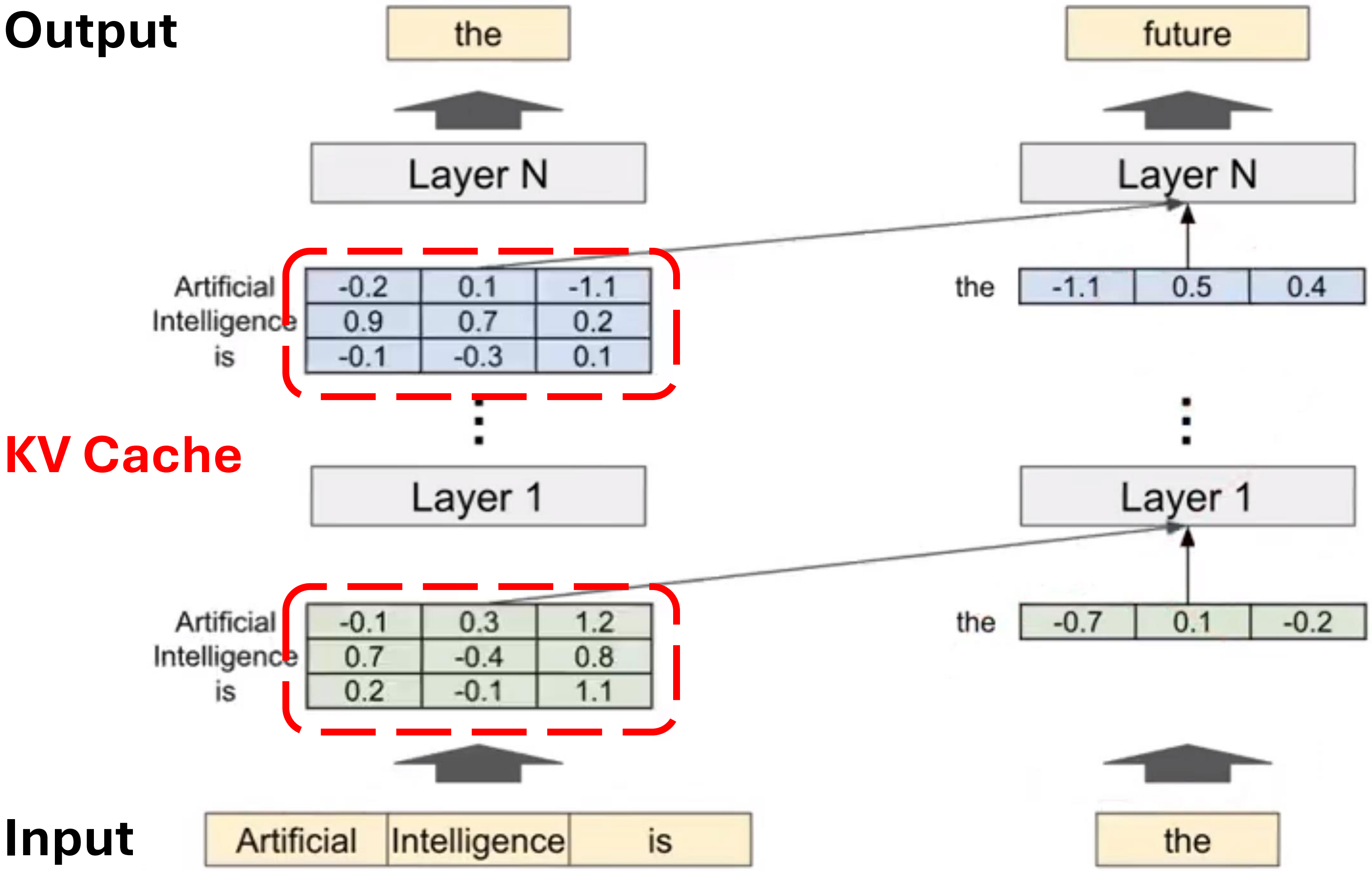

Output
the
Layer N
Layer 1
Input
Artificial
Intelligence
is
future
Layer N
the
-1.1
0.5
0.4
Layer 1
the
-0.7
0.1
-0.2
the

KV Cache



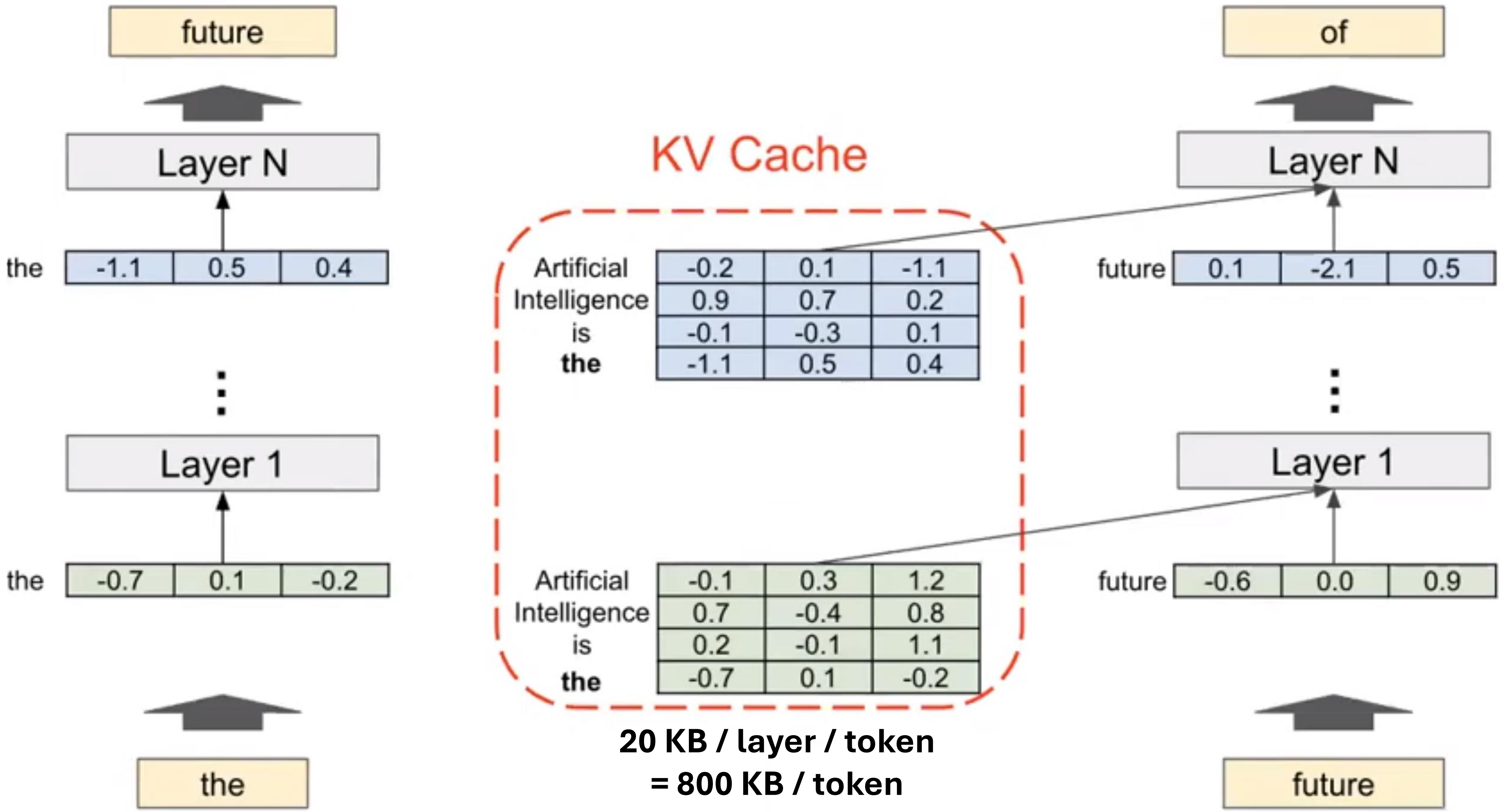
Output
the
future
Layer N
Layer N
Artificial
Intelligence
is
-0.2 0.1 -1.1
0.9 0.7 0.2
-0.1 -0.3 0.1
the
-1.1 0.5 0.4
Layer 1
Layer 1
Artificial
Intelligence
is
-0.1 0.3 1.2
0.7 -0.4 0.8
0.2 -0.1 1.1
the
-0.7 0.1 -0.2
Input
Artificial Intelligence is
the

KV Cache



Output
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

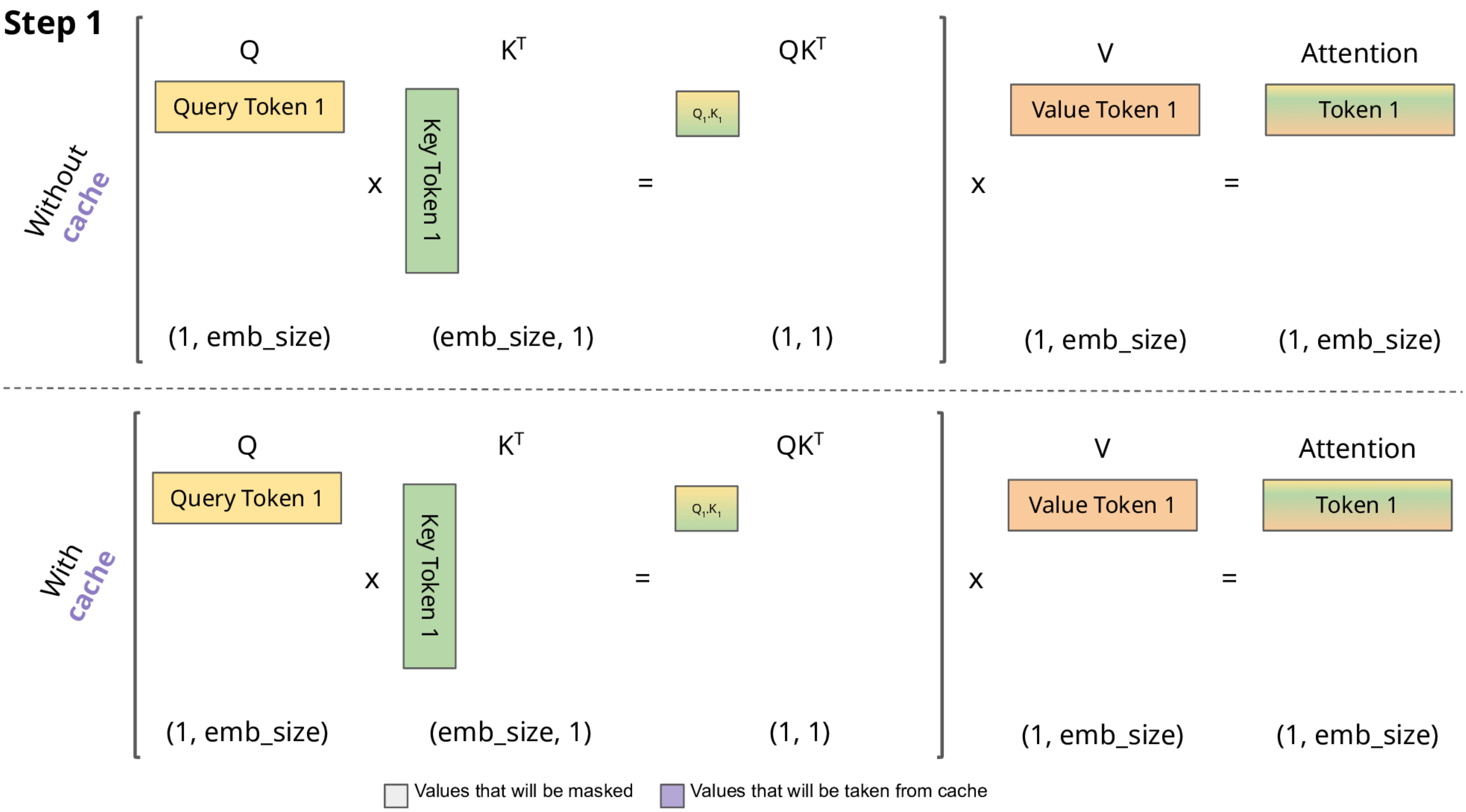
KV Cache



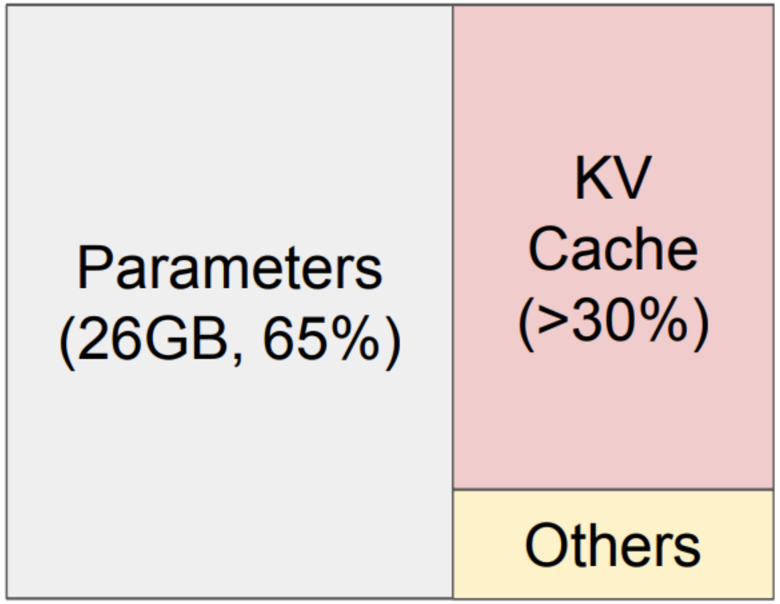
20 KB / layer / token
= 800 KB / token
(LLaMA-13B)

KV Cache

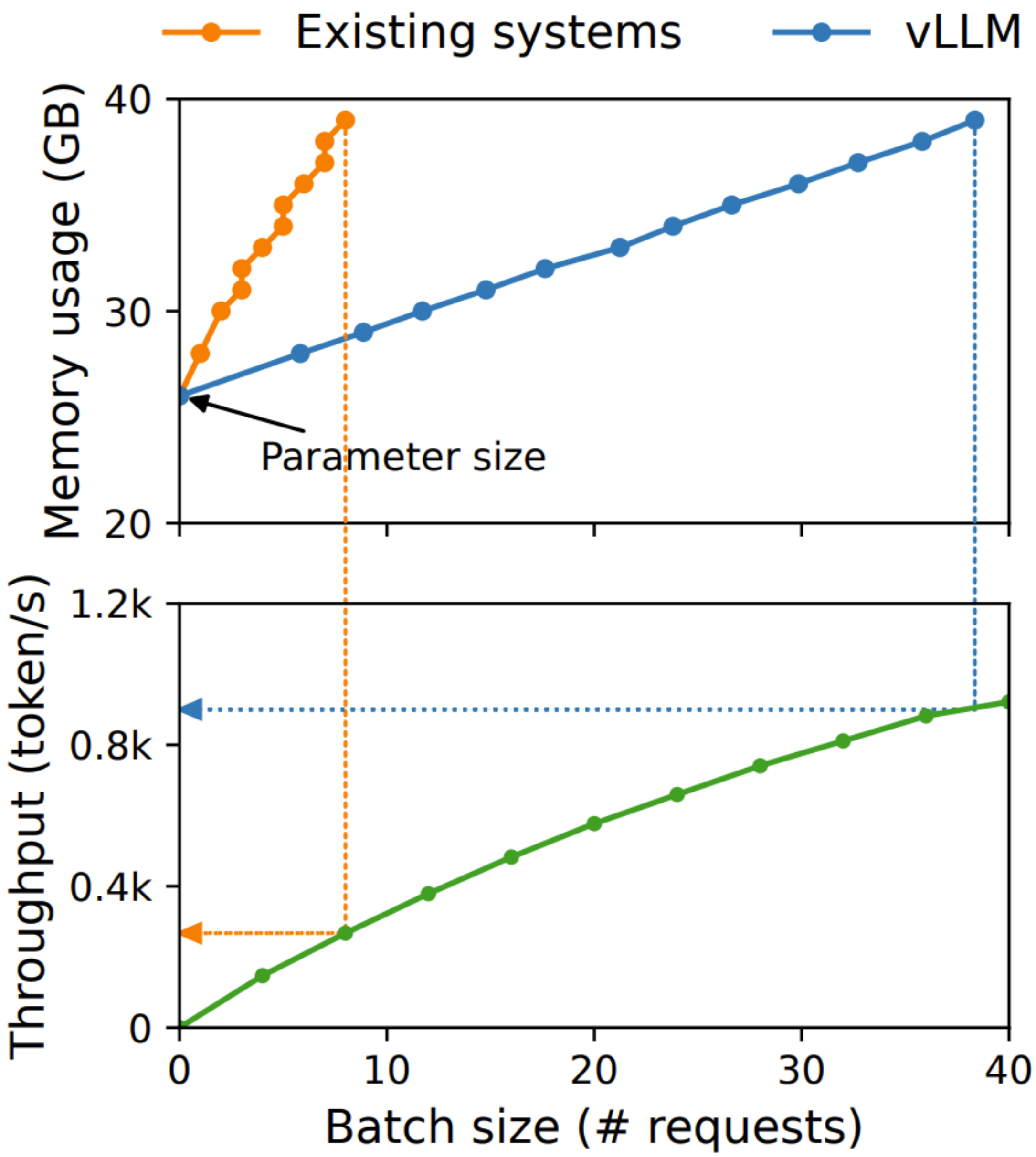
Animation source: https://medium.com/@joaolages/kv-caching-explained-276520203249



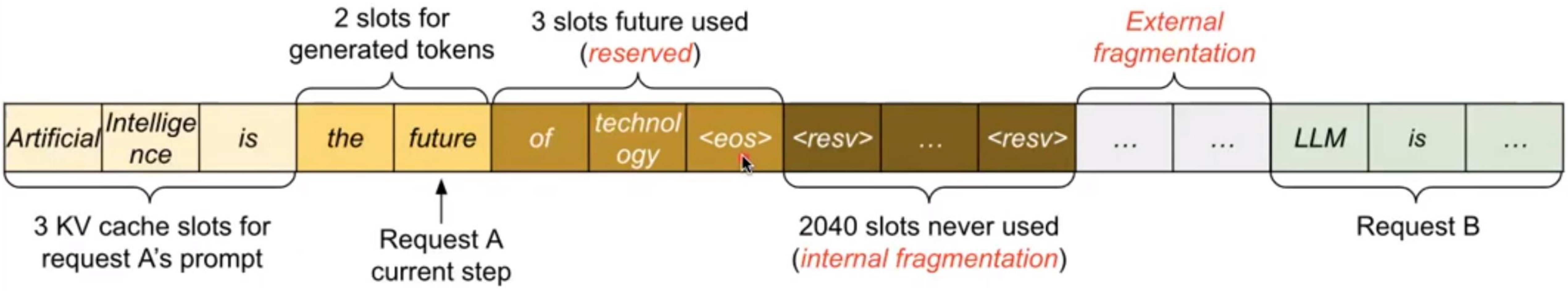
Key Insight



13B LLM on A100-40GB

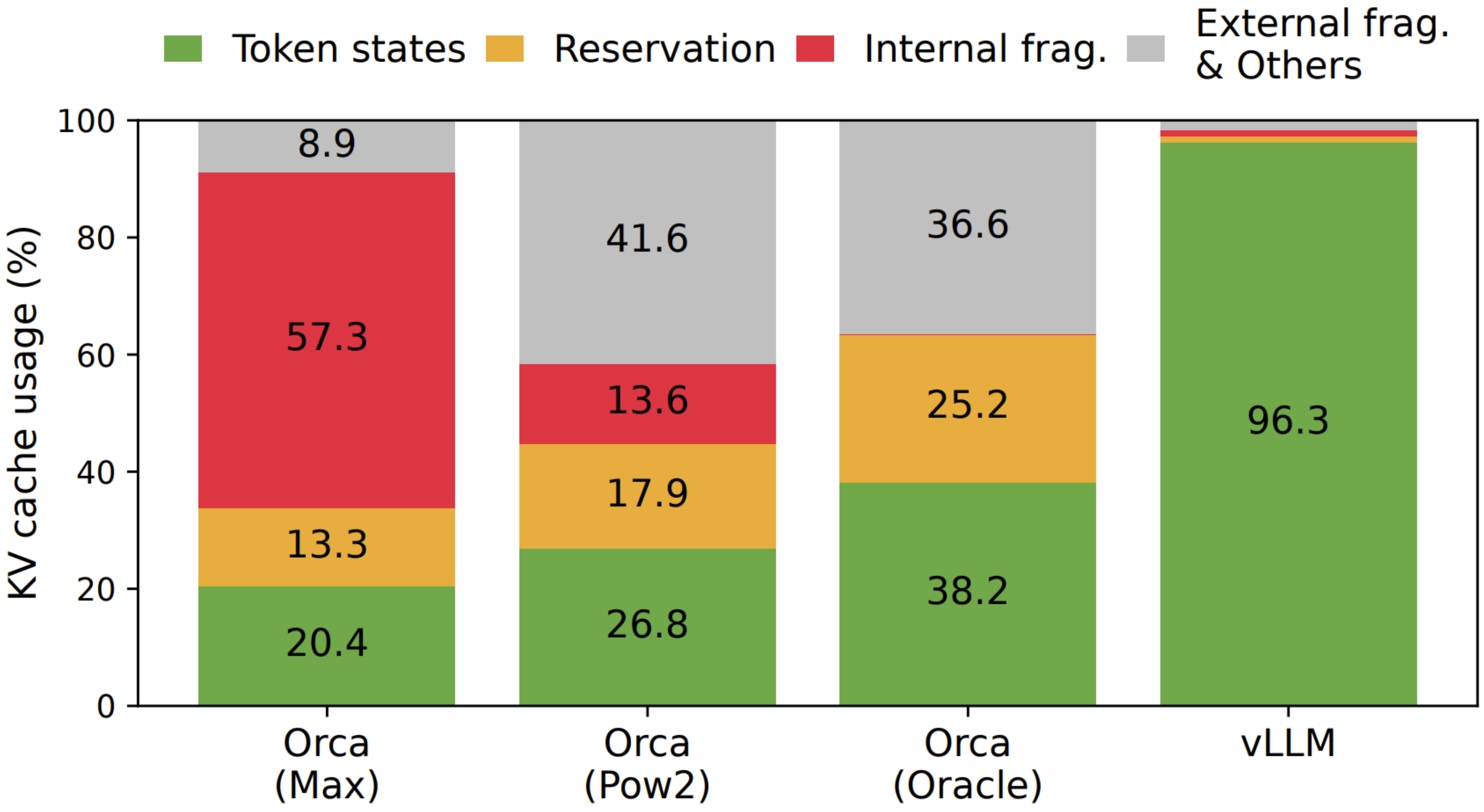


Memory Waste in KV Cache



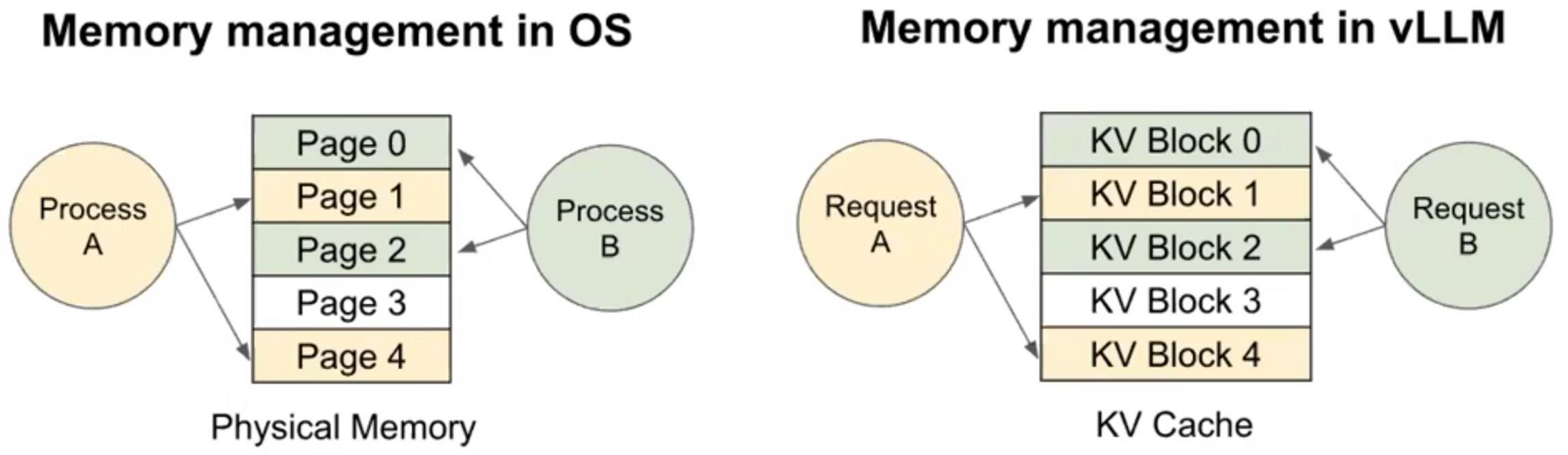
- **Internal fragmentation:** over-allocated due to the unknown output length
- **Reservation:** not used at the current step, but used in the future
- **External fragmentation:** due to different sequence lengths

Memory Waste in KV Cache



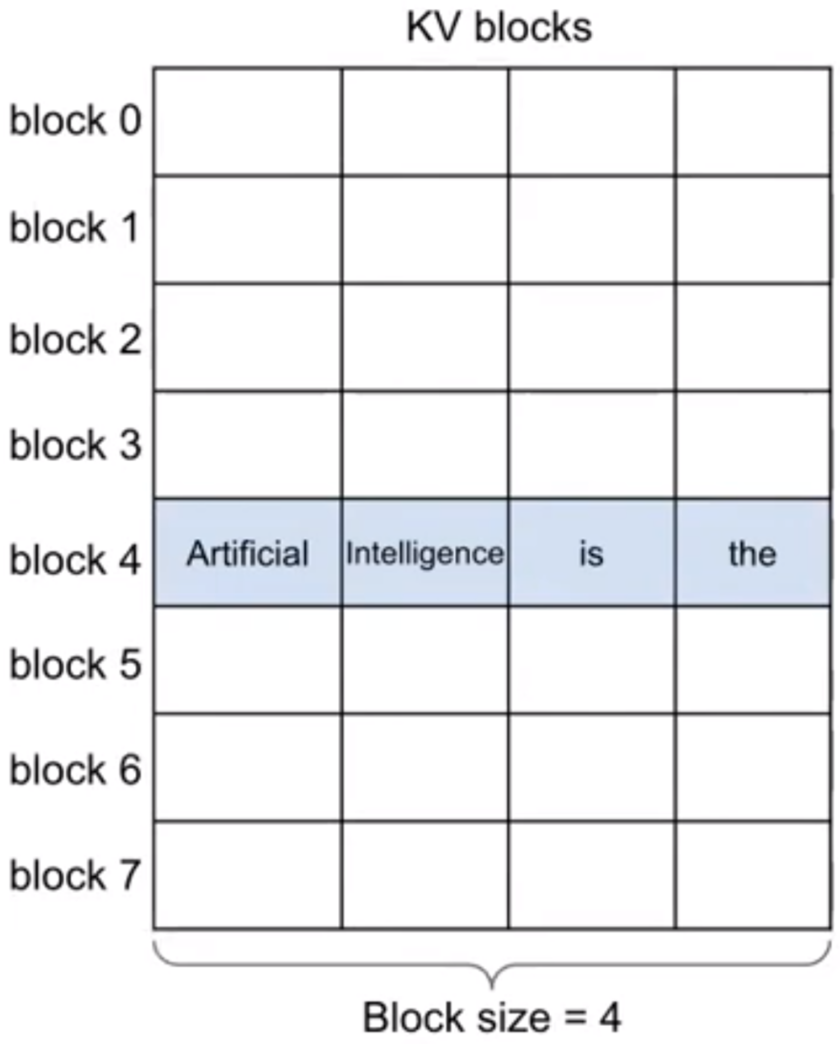
Only **20% - 40%** of KV cache is utilized to store token states

vLLM: Efficient Memory Management for LLM Inference



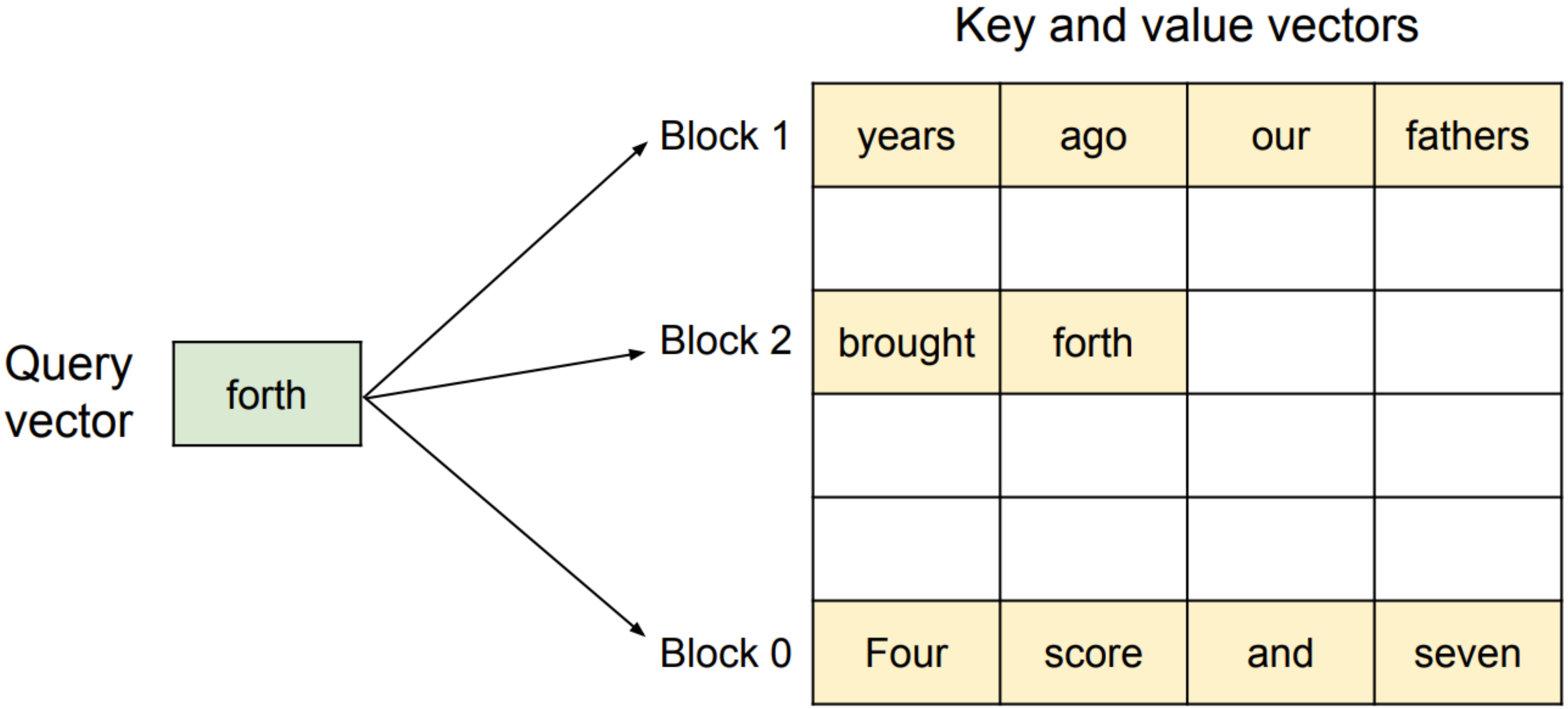
KV Block

- A fixed-size contiguous chunk of memory that stores the tokens' KV states
- Similar to the concept of a memory page



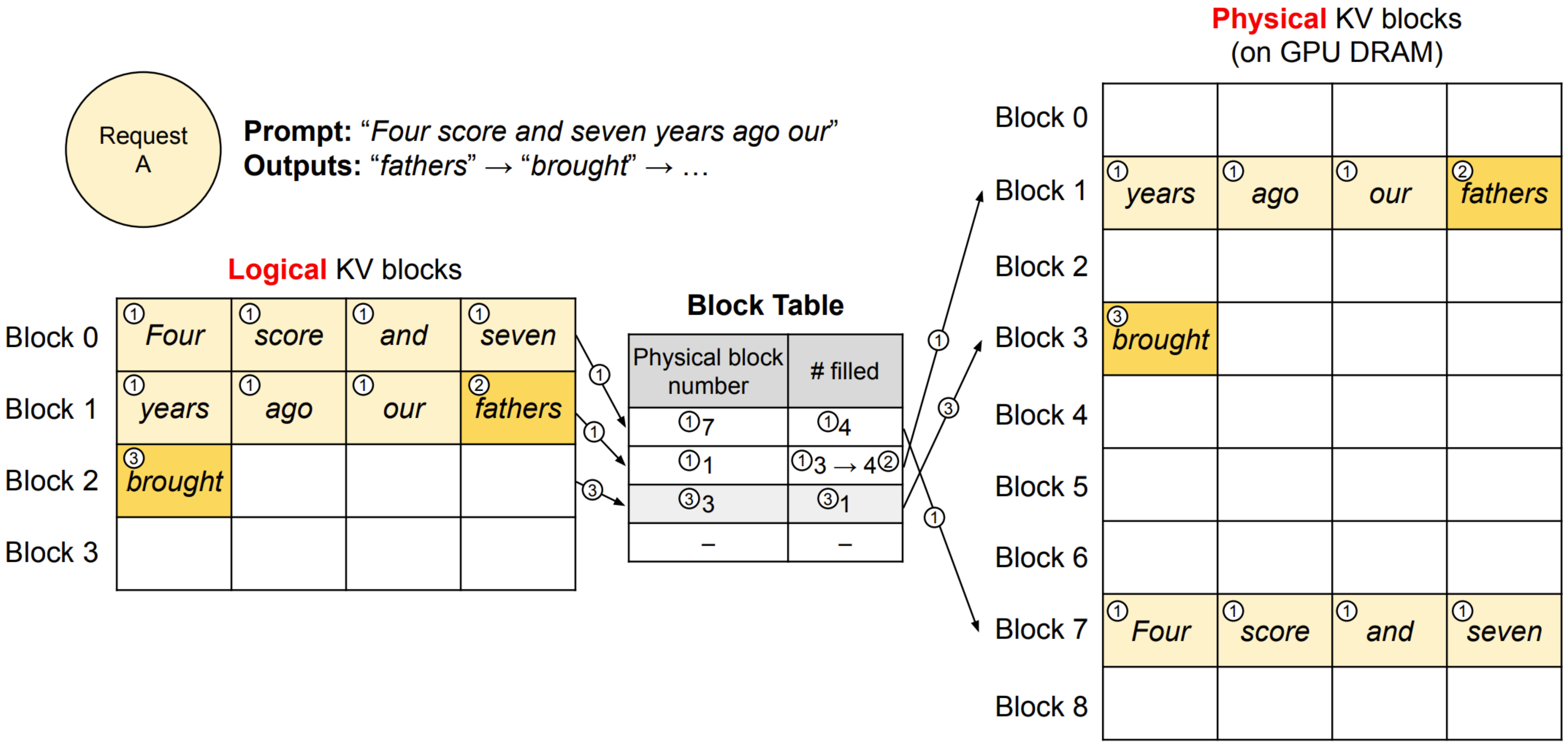
PagedAttention

- Manages KV cache in block granularity instead of sequence (i.e., request) granularity
- Allows storing logically contiguous KV blocks in non-contiguous physical memory

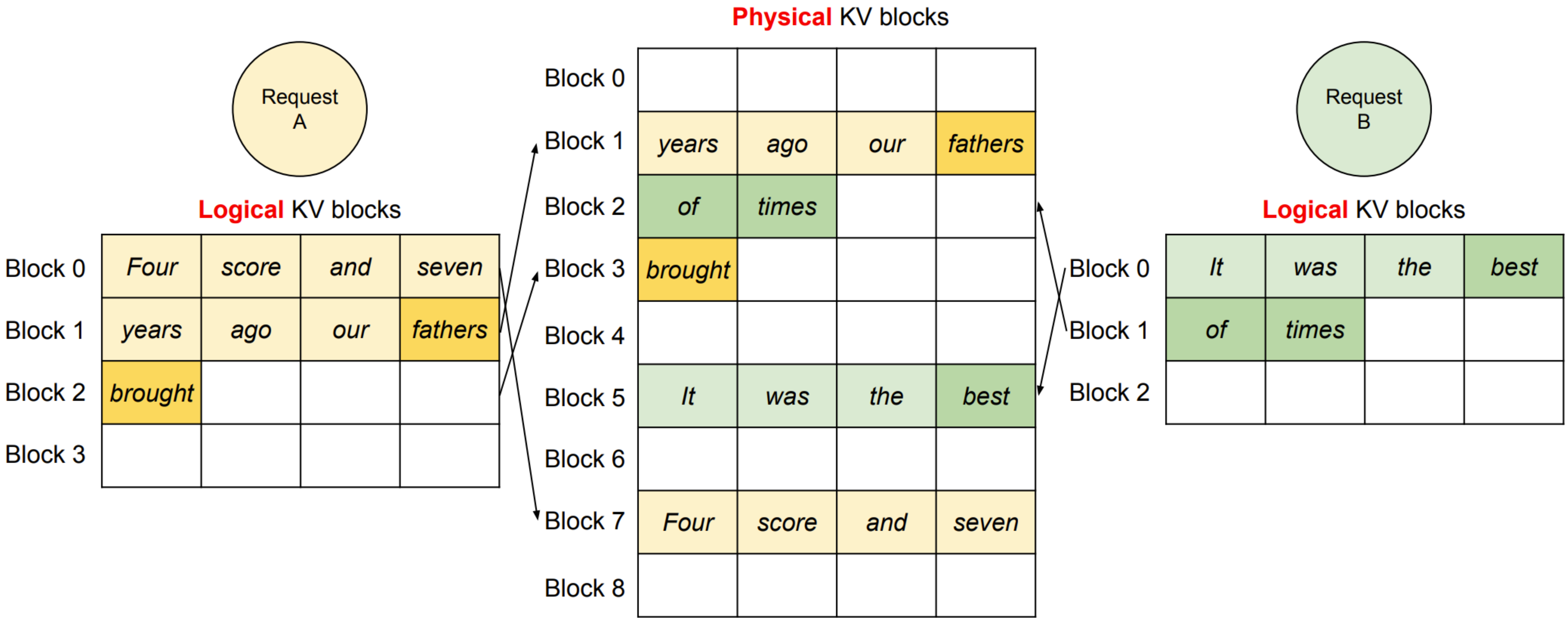


Example sequence: “Four score and seven years ago our | fathers brought forth”

Logical-to-Physical KV Block Translation

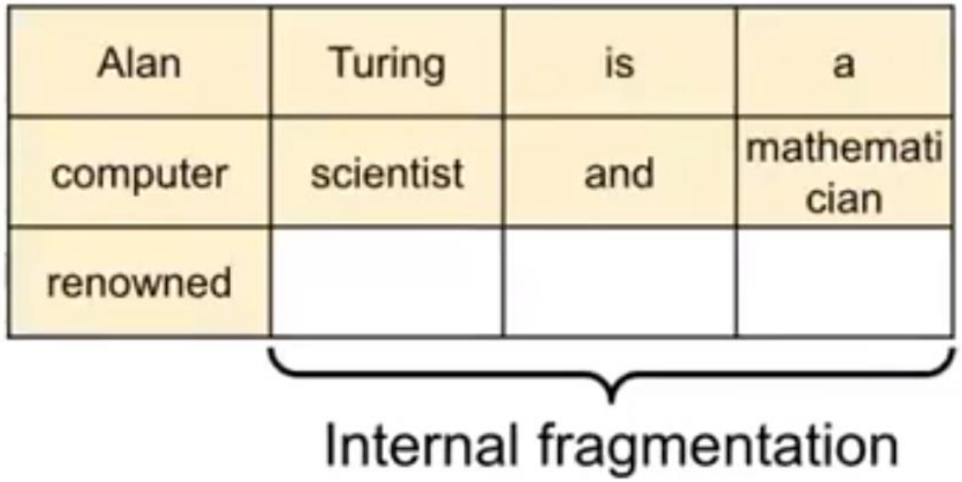


Serving Multiple Requests



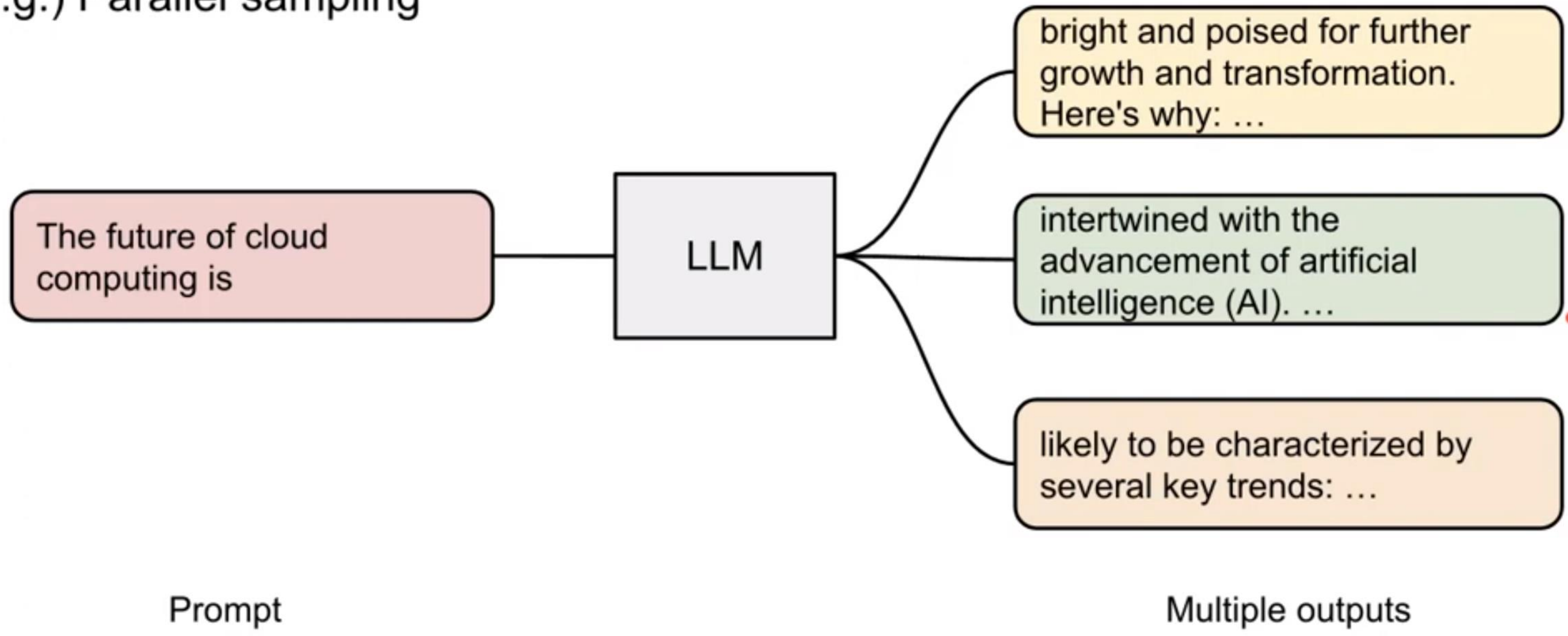
Memory Efficiency of vLLM

- Minimizes **internal fragmentation**
  - Only happens at the last block of a sequence
  - # wasted tokens per sequence < block size
    - Seq len: O(100) – O(1000) tokens
    - Block size: 16 or 32 tokens

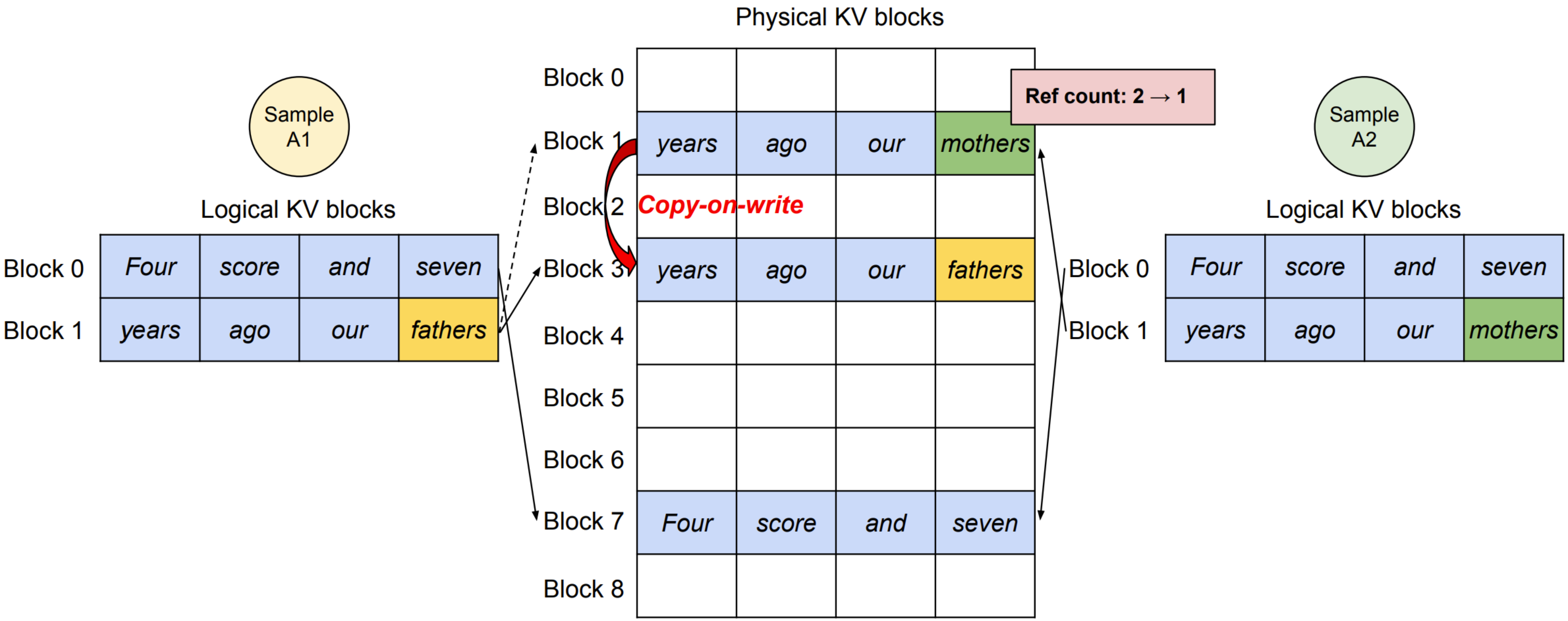


Dynamic Block Mapping Enables Sharing

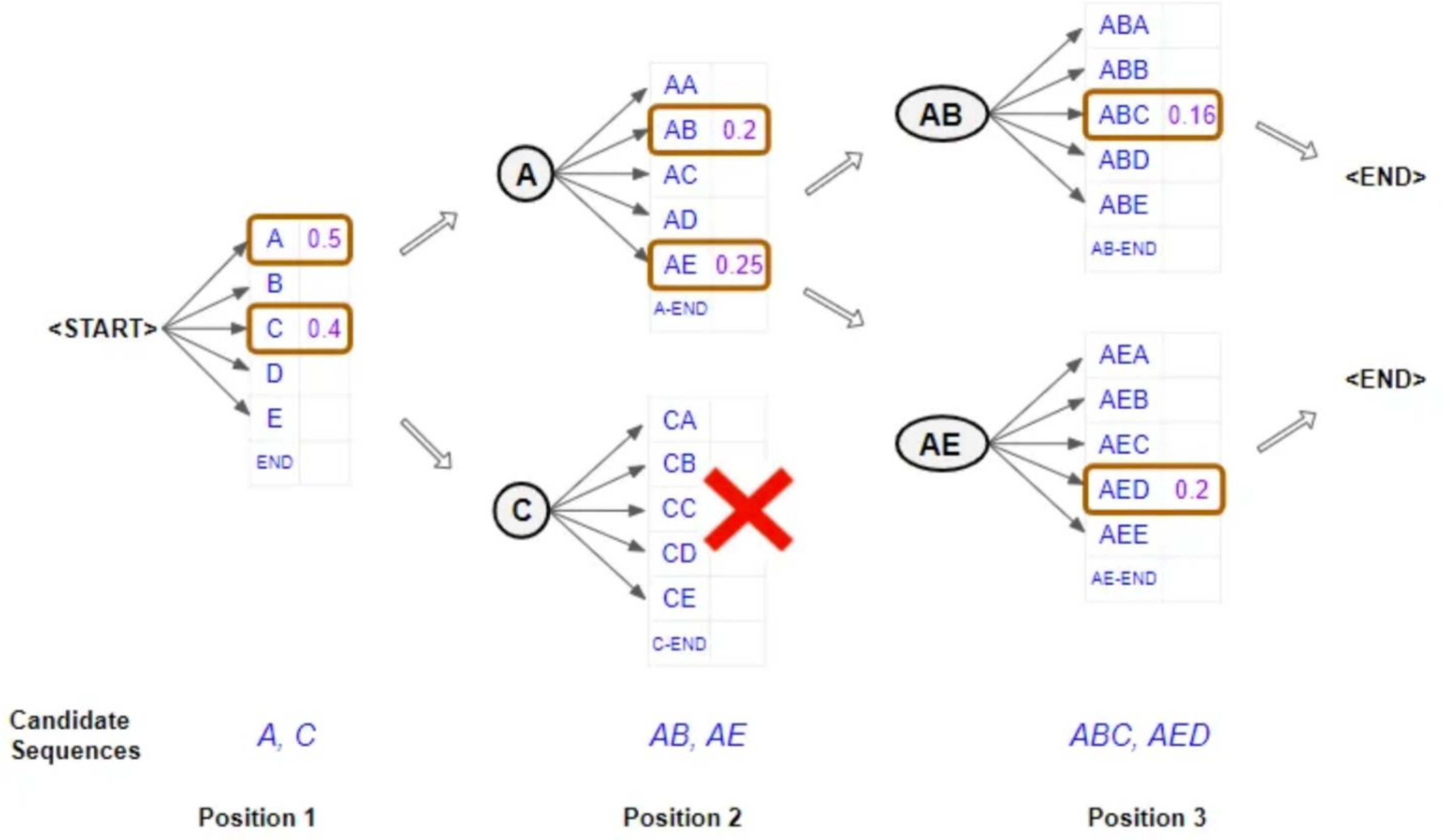
E.g.) Parallel sampling



KV Block Sharing for Parallel Sampling

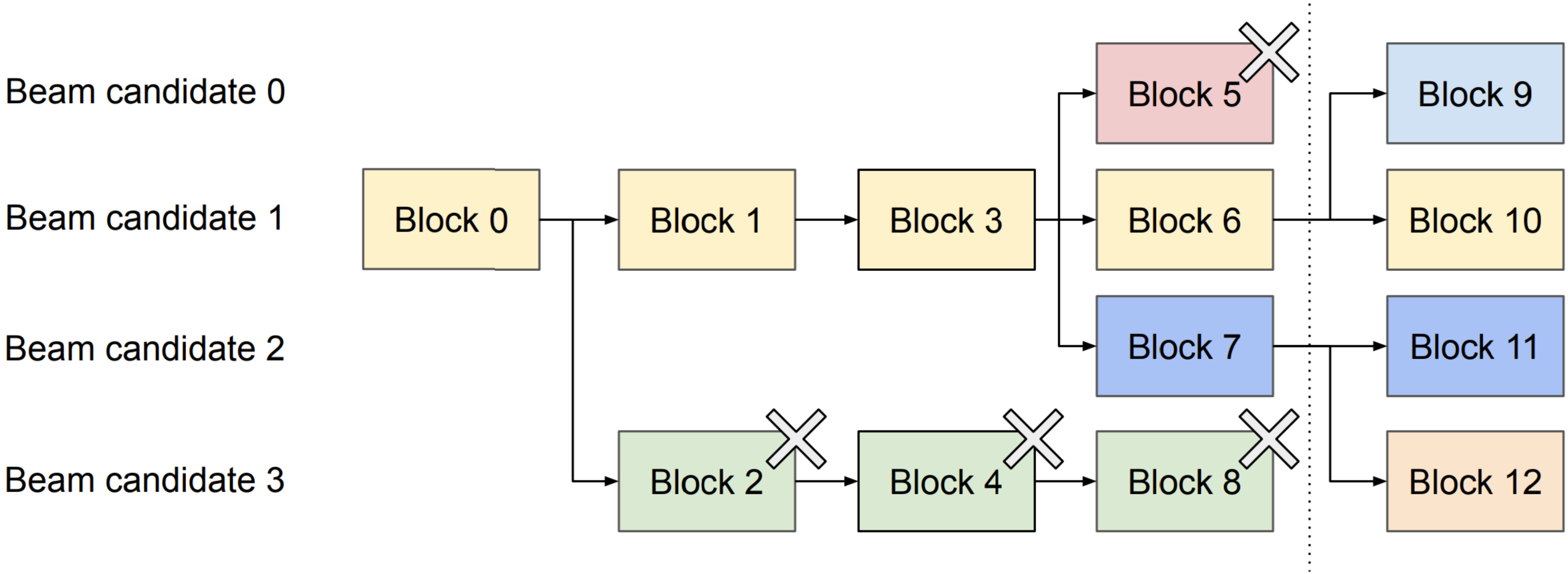


KV Block Sharing for Beam Search



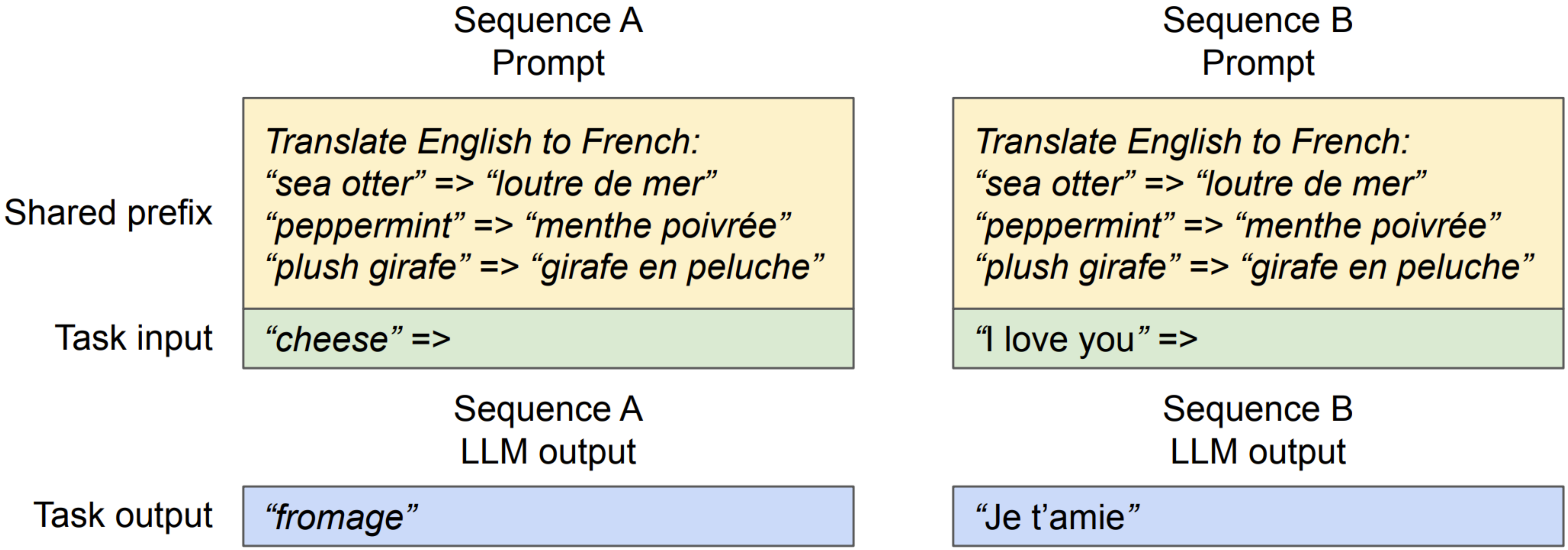
Picture source: https://towardsdatascience.com/foundations-of-nlp-explained-visually-beam-search-how-it-works-1586b9849a24.

KV Block Sharing for Beam Search



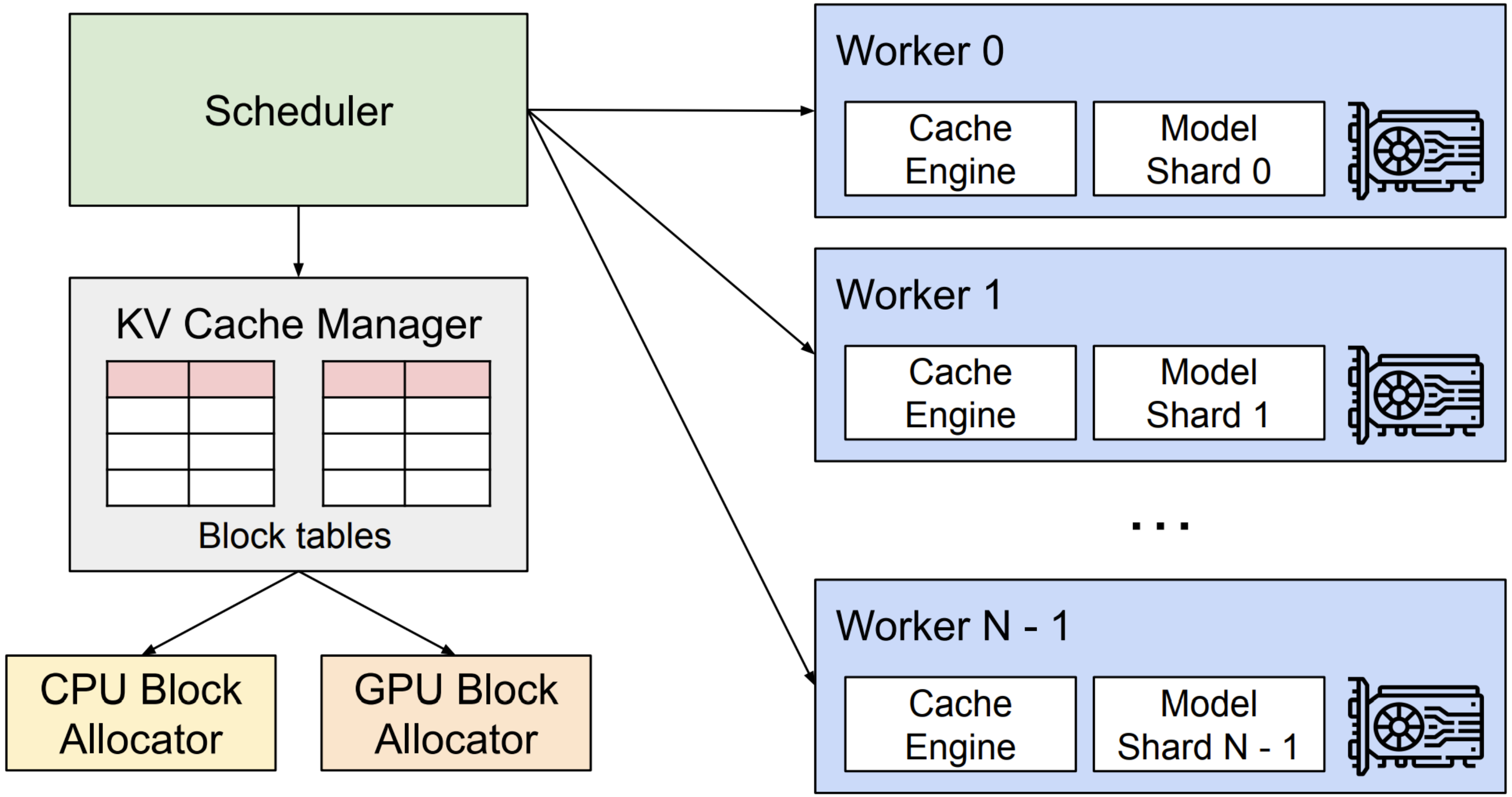
- Similar to process **fork** and **kill**

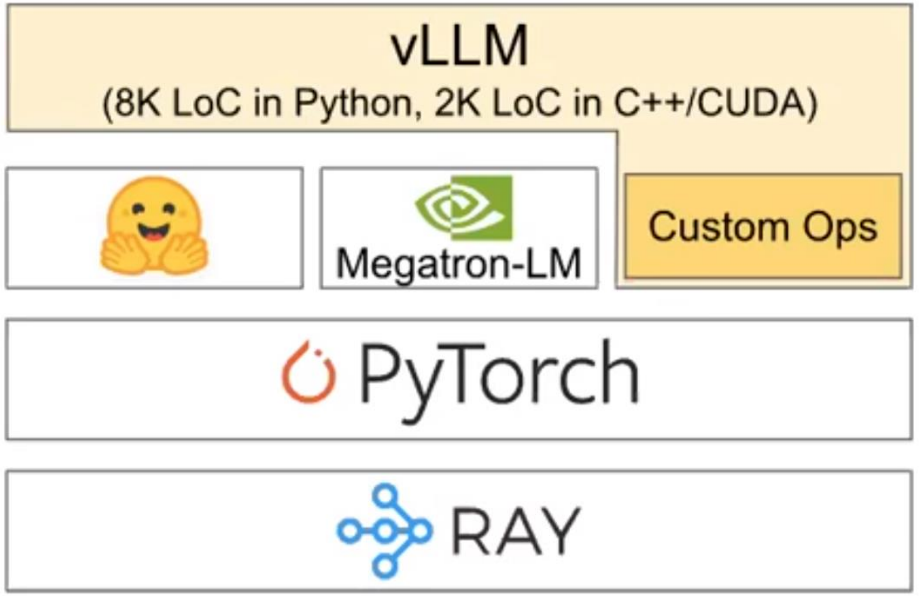
Shared Prompt



- Similar to shared libraries in OS

vLLM System Architecture & Implementation

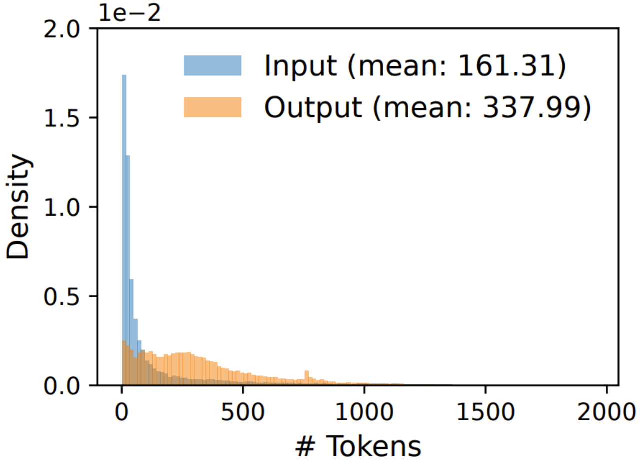




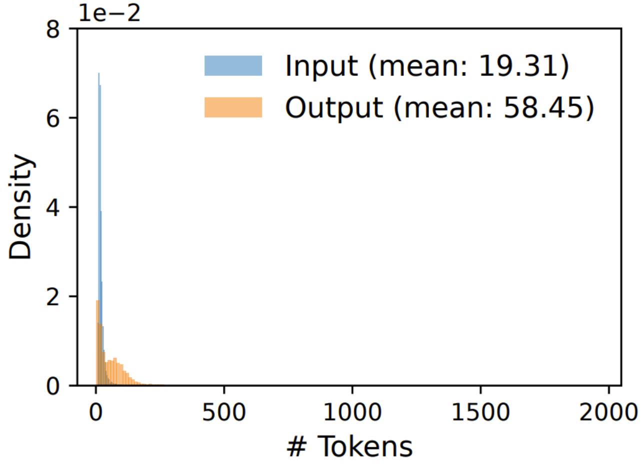
Evaluation

System configuration.

Model size	**13B**	**66B**	**175B**
GPUs	A100	4×A100	8×A100-80GB
Total GPU memory	40 GB	160 GB	640 GB
Parameter size	26 GB	132 GB	346 GB
Memory for KV cache	12 GB	21 GB	264 GB
Max. # KV cache slots	15.7K	9.7K	60.1K



(a) ShareGPT



(b) Alpaca

Figure 11. Input and output length distributions of the (a) ShareGPT and (b) Alpaca datasets.

vLLM Improves Inference Throughput by Enabling Larger Batch Size

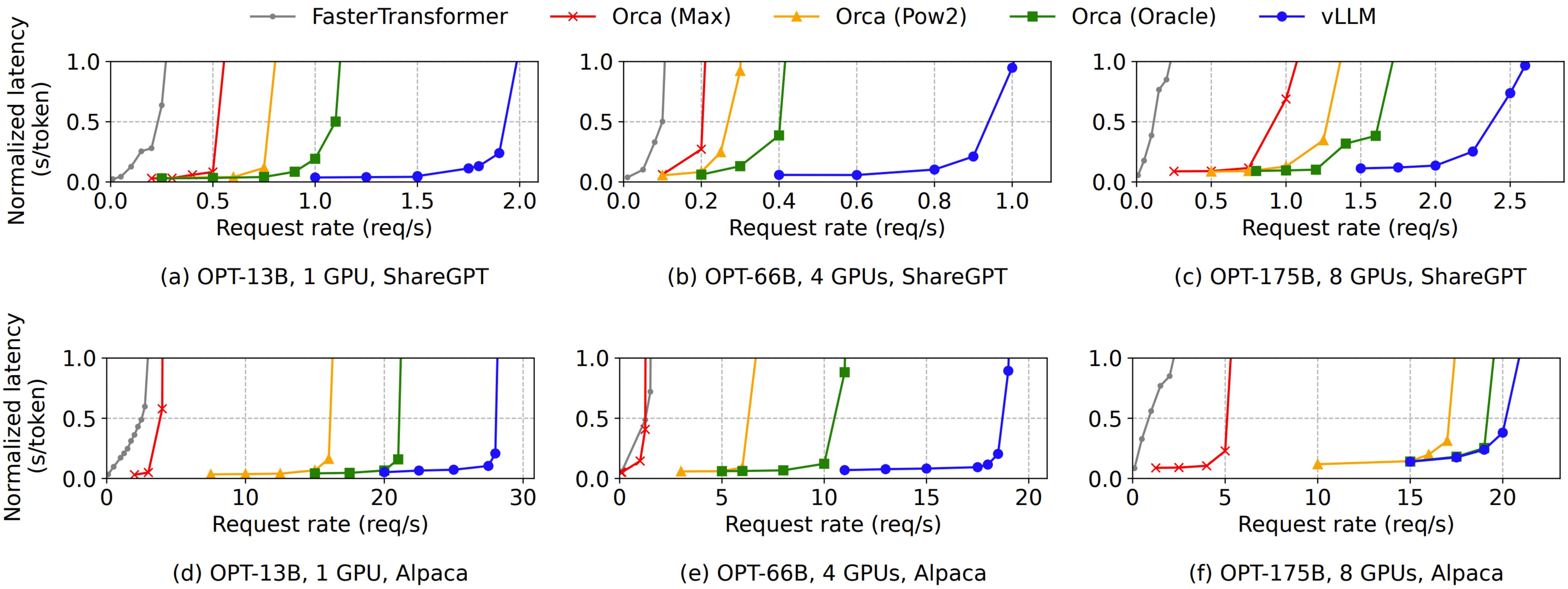


Figure 12. Single sequence generation with OPT models on the ShareGPT and Alpaca dataset

vLLM Improves Inference Throughput by Enabling Larger Batch Size

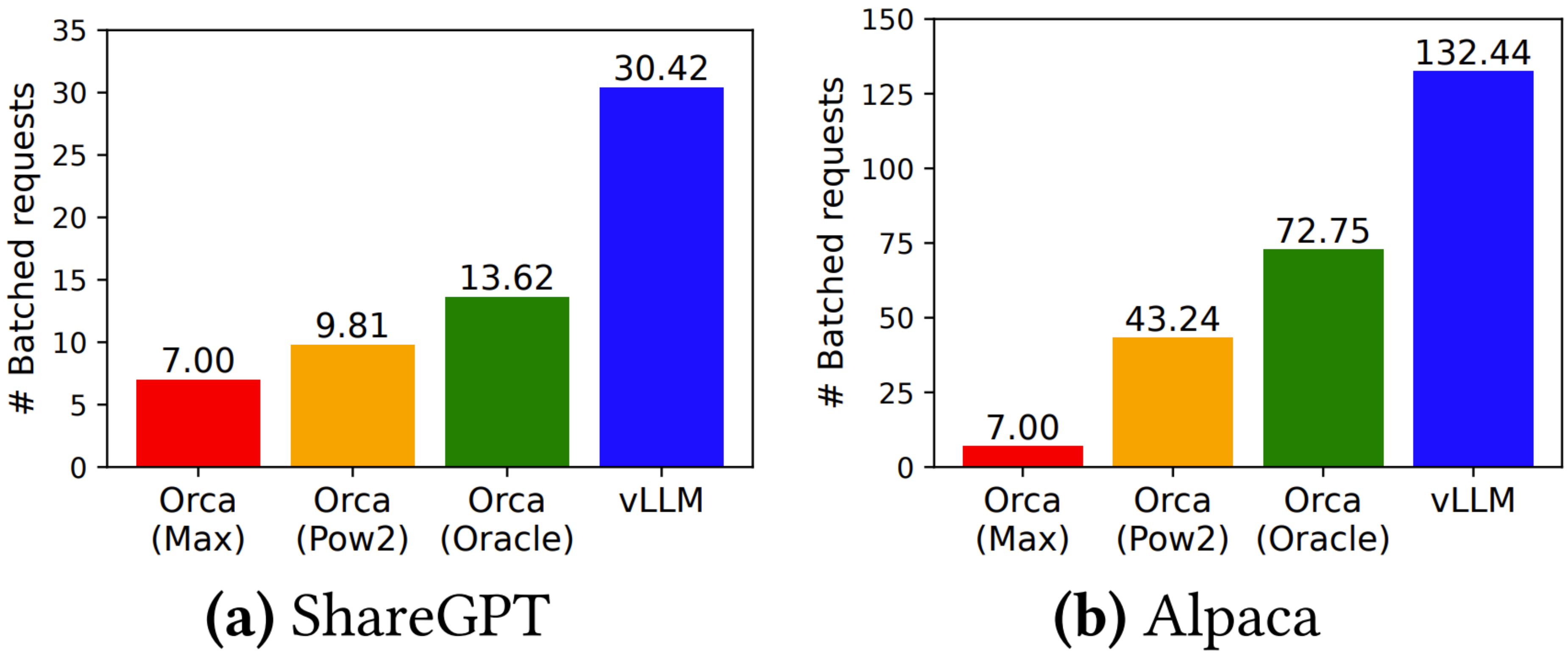


Figure 13. Average number of batched requests when serving OPT-13B for the ShareGPT (2 reqs/s) and Alpaca (30 reqs/s) traces.

Memory Saving of vLLM

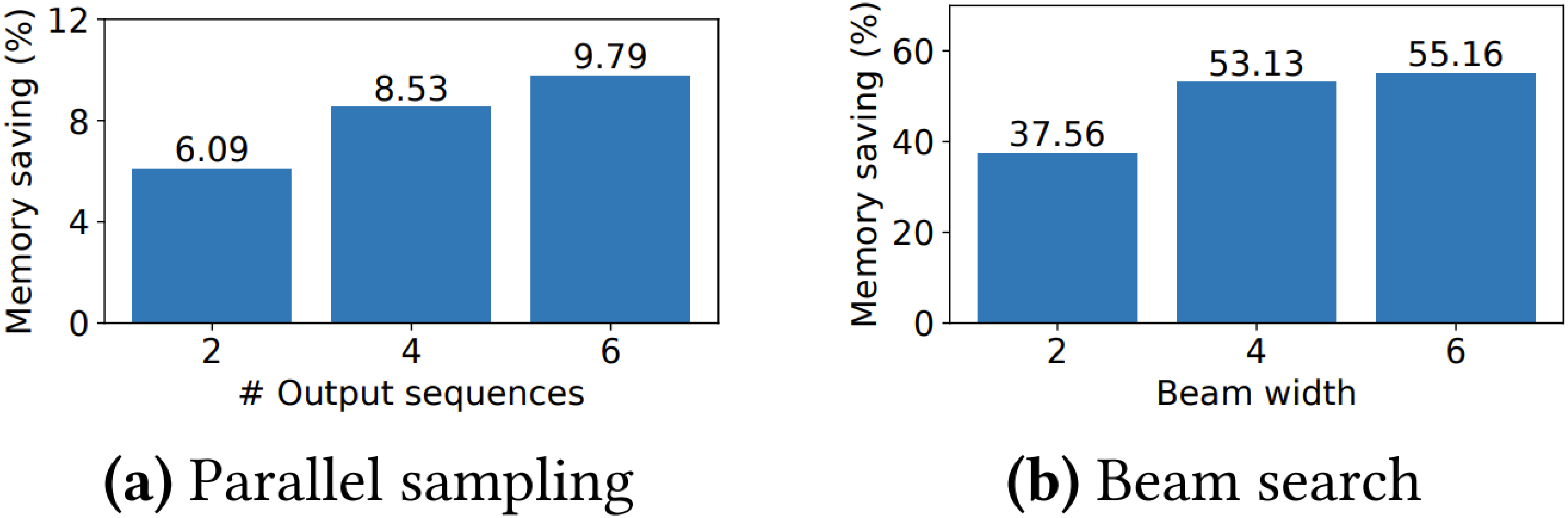


Figure 15. Average amount of memory saving from sharing KV blocks, when serving OPT-13B for the Alpaca trace.

Takeaways

- Strength
  - Interesting observation on the KV cache memory inefficiency
  - Analogy between KV cache management and OS paging
  - Open-source implementation
- Weakness
  - Cannot fundamentally improve inference latency
  - For multi-chip execution, vLLM assumes attention heads are sharded
    - If even a single attention head is too large, or we want to split it across multiple chips to improve latency, how can vLLM support sharding the KV cache?
  - The fundamental bottlenecks faced by LLM serving, **memory capacity** due to large model weights, and **memory bandwidth** due to low compute intensity of auto-regressive decoding, remain unsolved.
    - Speculative decoding?
    - New model architectures? (e.g., SSM, Mamba)
    - New hardware/architectural innovations? (e.g., processing-in-memory, NVIDIA's new patent that proposes stacking HBM dies on the processor die to expose a wider mem interface)

Q & A