LLM Serving: Role of Memory

Arvind Krishnamurthy

Material adapted from slides by Woosuk Kwon and Umar Jamil
Paged Attention paper and Umar's talk

Lecture Outline

- **Memory management in GPUs for LLM serving**
- Augmented model memory through retrieval augmented generation (RAG)

Efficient Memory Management for Large Language Model Serving with PagedAttention

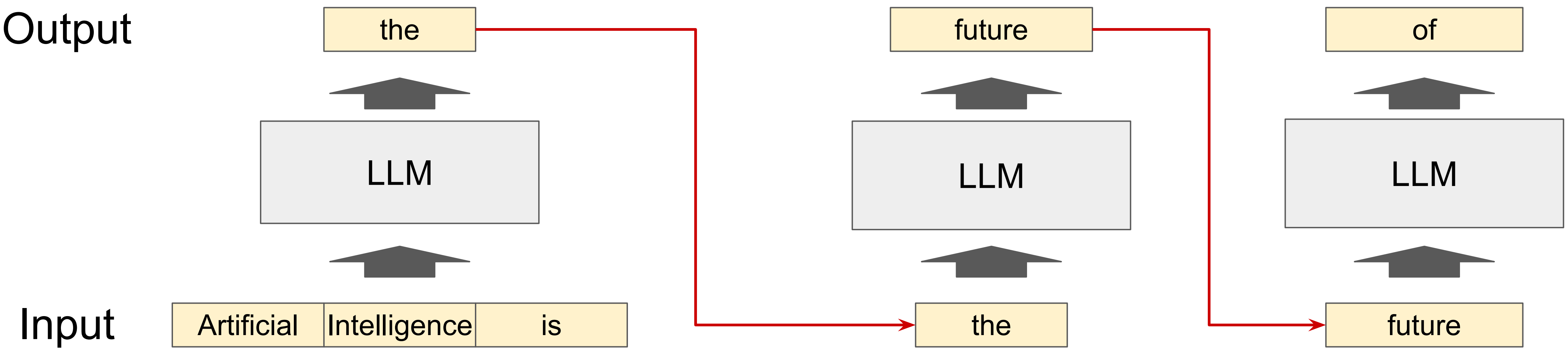
Woosuk Kwon[1,*] **Zhuohan Li**[1,*] Siyuan Zhuang[1] Ying Sheng[1,2] Lianmin Zheng[1]
Cody Hao Yu[3] Joseph E. Gonzalez[1] Hao Zhang[4] Ion Stoica[1]

[1]UC Berkeley [2]Stanford University [3]Independent Researcher [4]UC San Diego
**Equal contribution*

SOSP 2023

Berkeley
UNIVERSITY OF CALIFORNIA

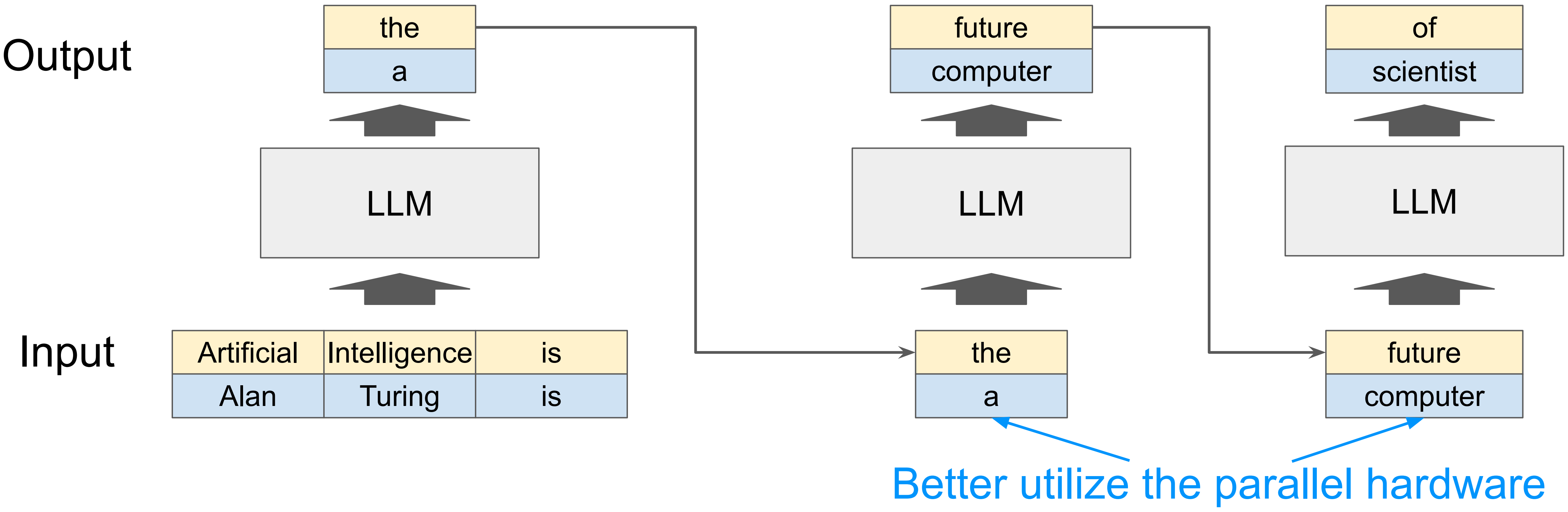
Inference process of LLMs



Why is it slow and inefficient?

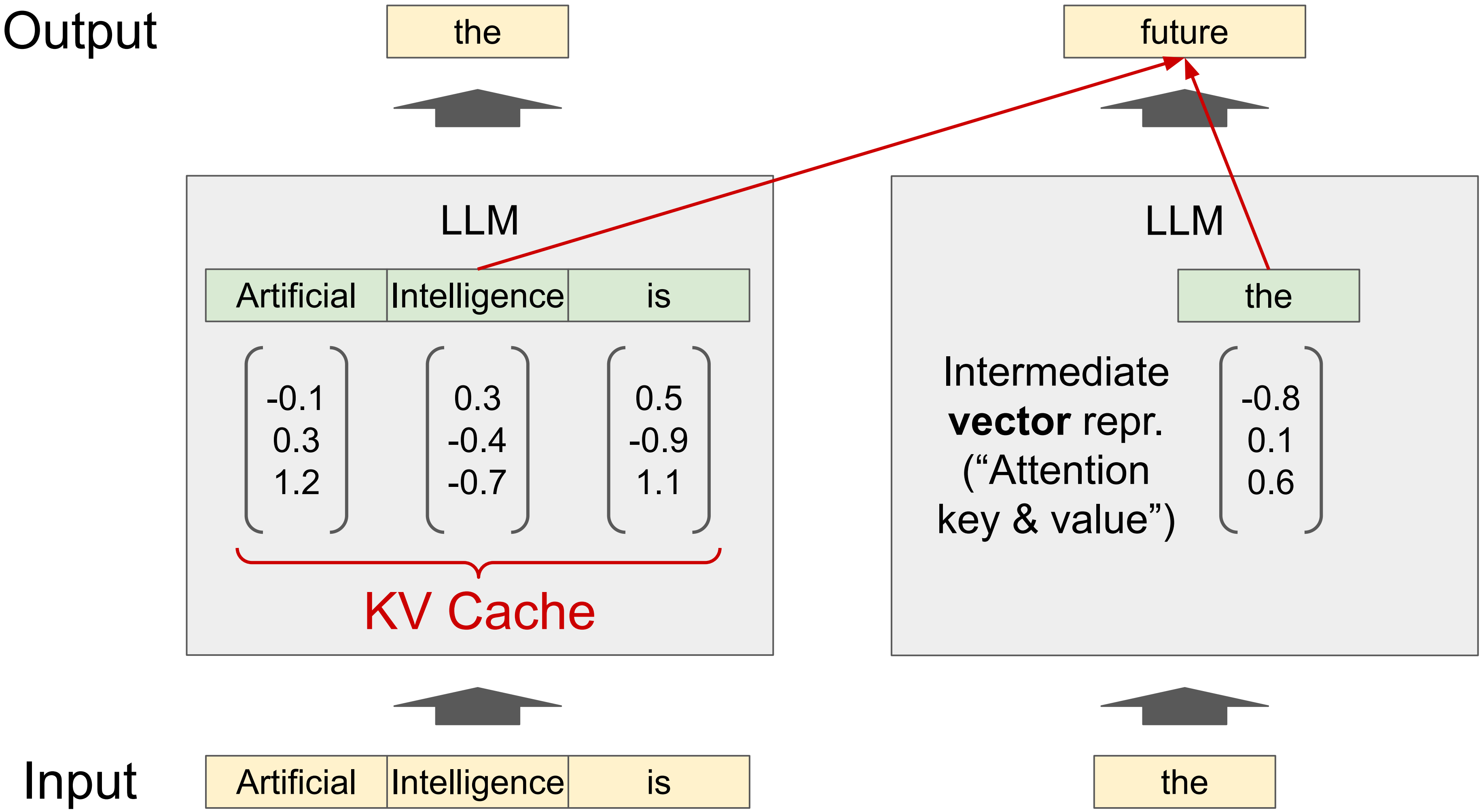
- The sequential dependency between output tokens makes it difficult to fully utilize the parallelism in GPUs

Solution: Batching multiple requests together

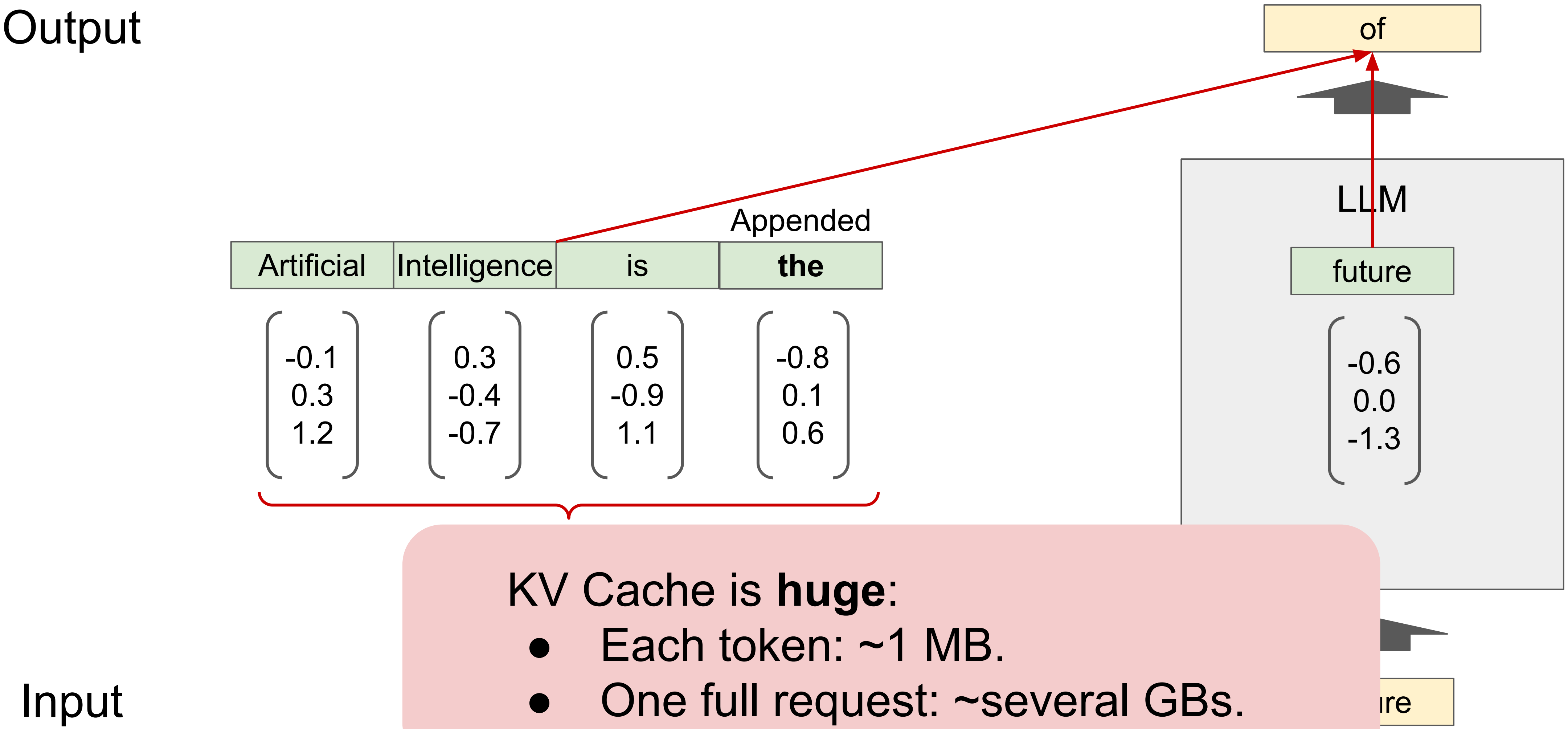


However, the batch size is significantly limited by the inefficient memory management for "***KV Cache***."

(Attention) KV Cache



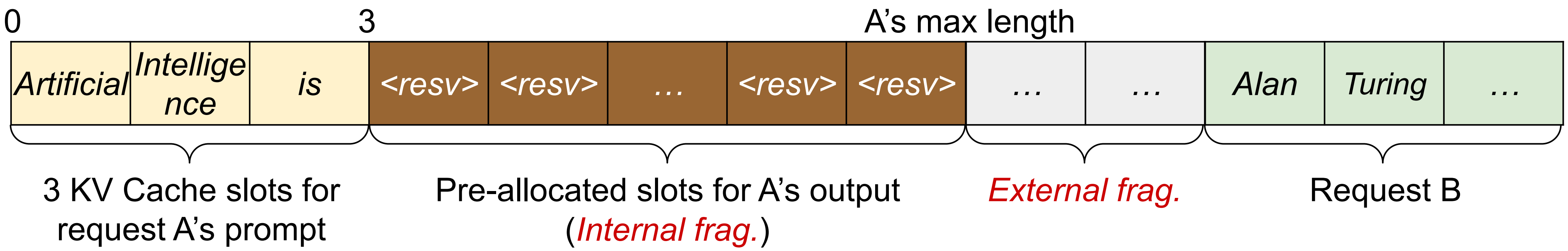
KV Cache dynamically grows and shrinks



KV Cache is **huge**:

- Each token: ~1 MB.
- One full request: ~several GBs.

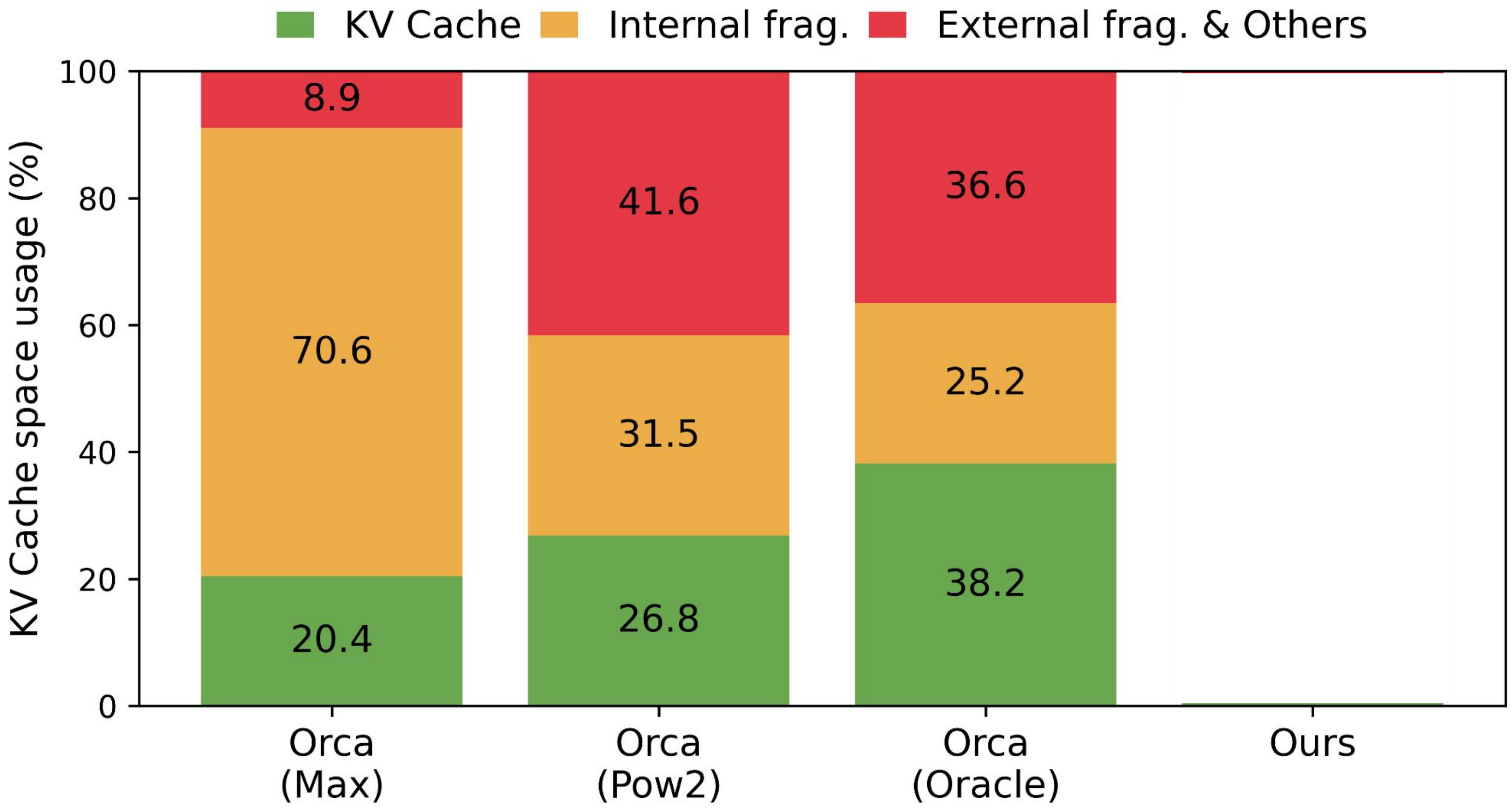
KV Cache management in previous systems



- **Pre-allocates contiguous** space of memory to the request’s maximum length
  - Useful convention in traditional deep learning workloads where the input/output shapes are **static** (e.g., fast pointer arithmetic, efficient memory access)
- Results in memory fragmentation
  - **Internal fragmentation** due to the **unknown** output length.
  - **External fragmentation** due to **non-uniform** per-request max lengths.

Significant memory waste in KV Cache space

- Only **20–40%** of KV Cache space is utilized to store actual token states

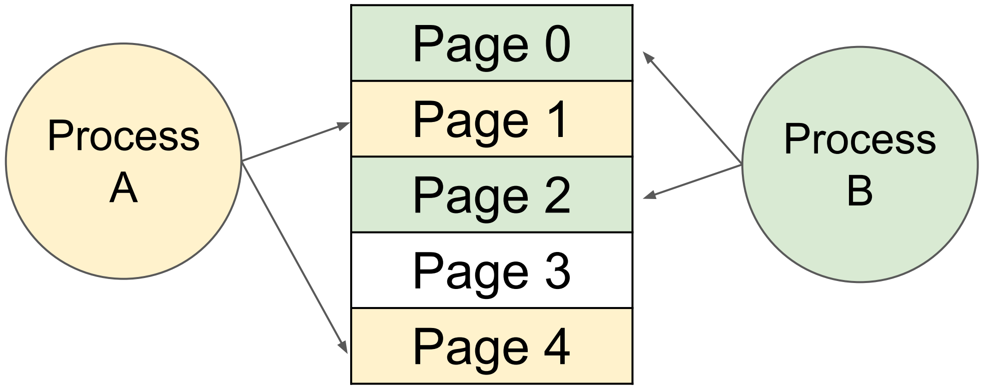


* Yu et al. “Orca: A Distributed Serving System for Transformer-Based Generative Models” (OSDI 22).

PagedAttention

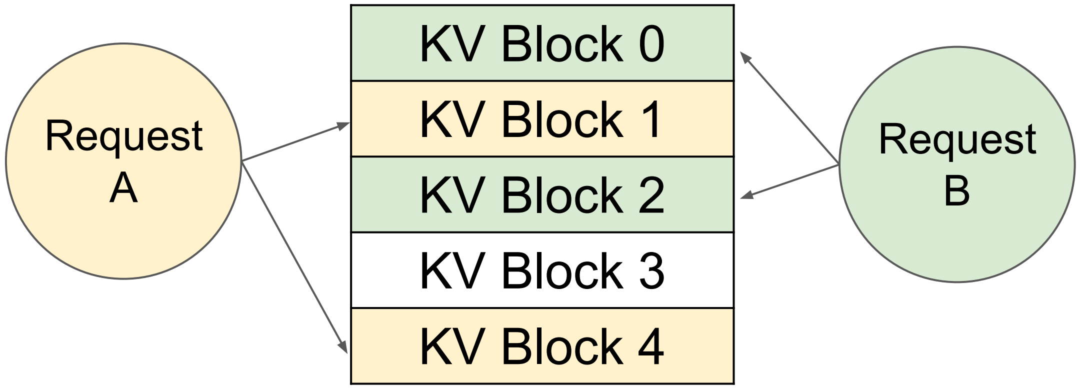
- **Application-level** memory **paging** and **virtualization** for attention KV Cache

Memory management in OS



Physical Memory

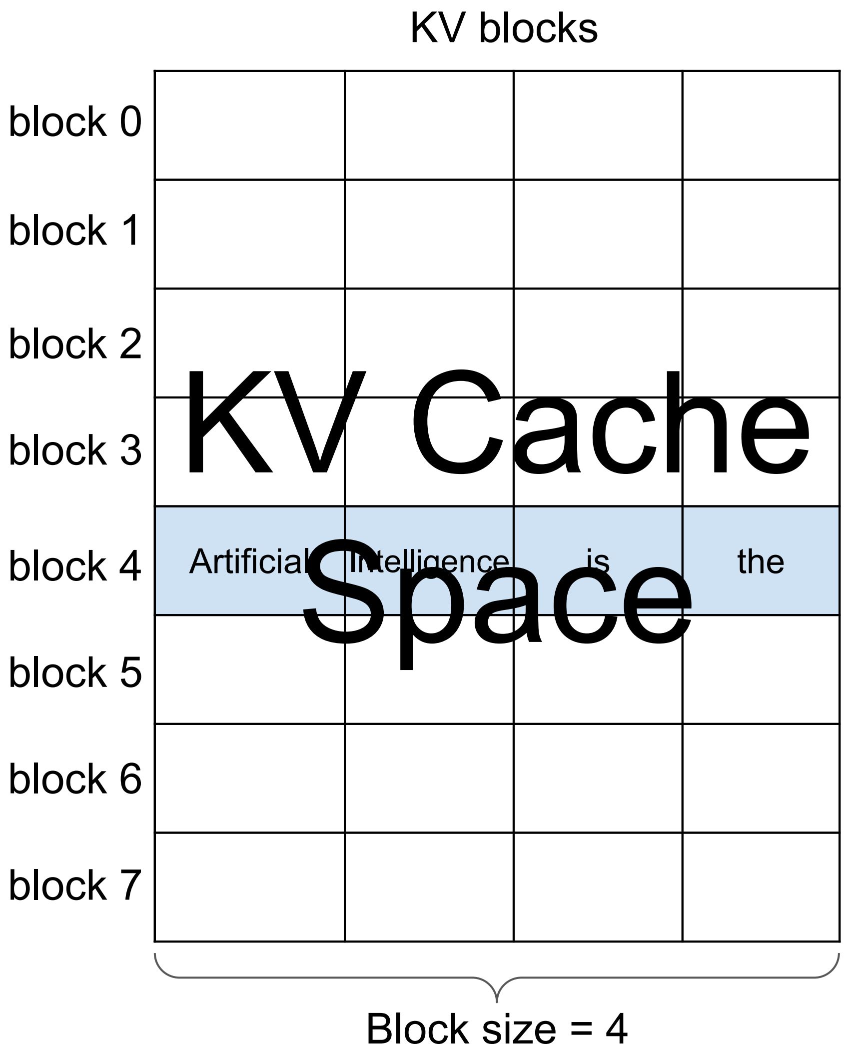
PagedAttention



KV Cache

Paging KV Cache space into KV blocks

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



Virtualizing KV Cache

Request A

Prompt: “Alan Turing is a computer scientist”

Logical KV blocks

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

Block table

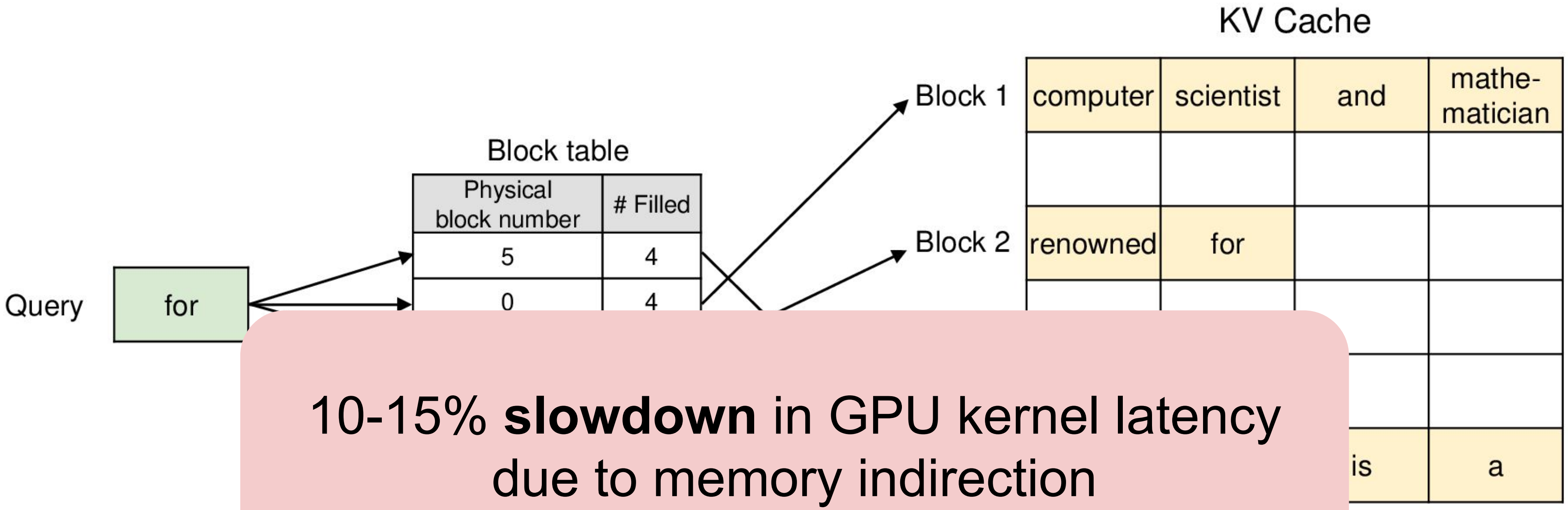
Physical block number	# Filled
7	4
1	2
–	–
–	–

Physical KV blocks

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

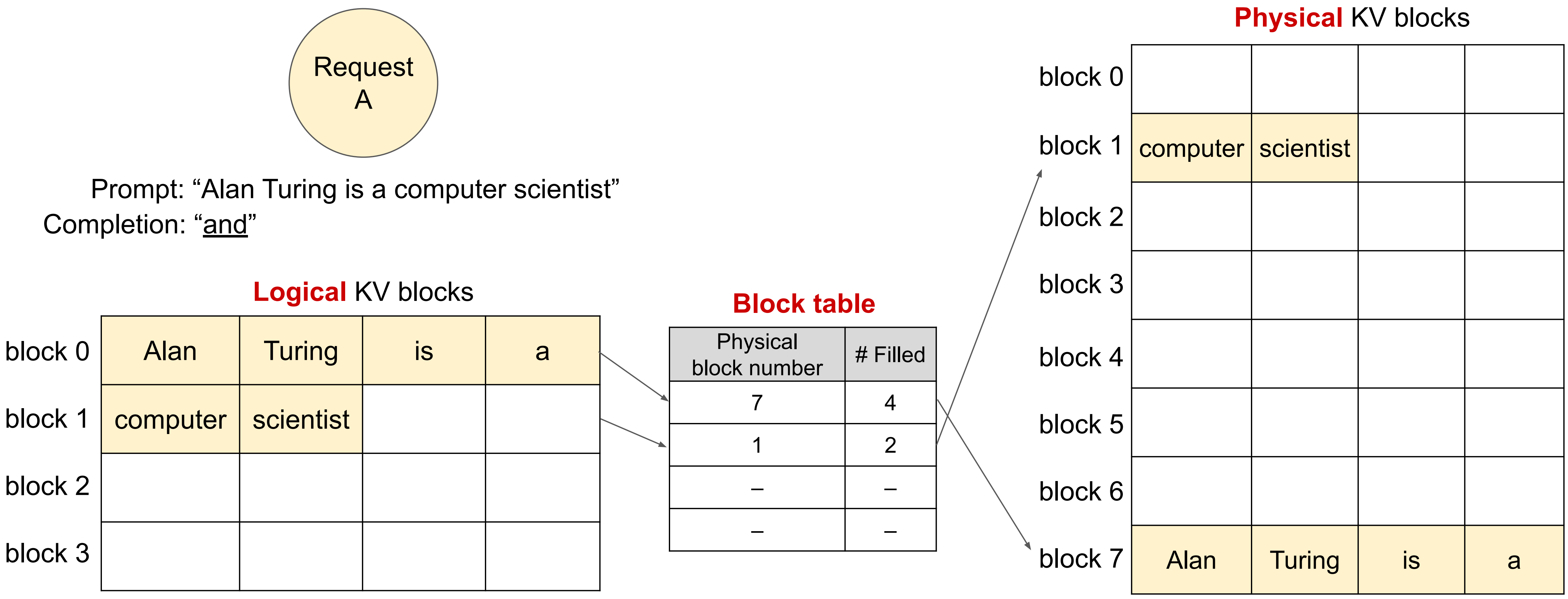
Attention mechanism with virtualized KV Cache

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

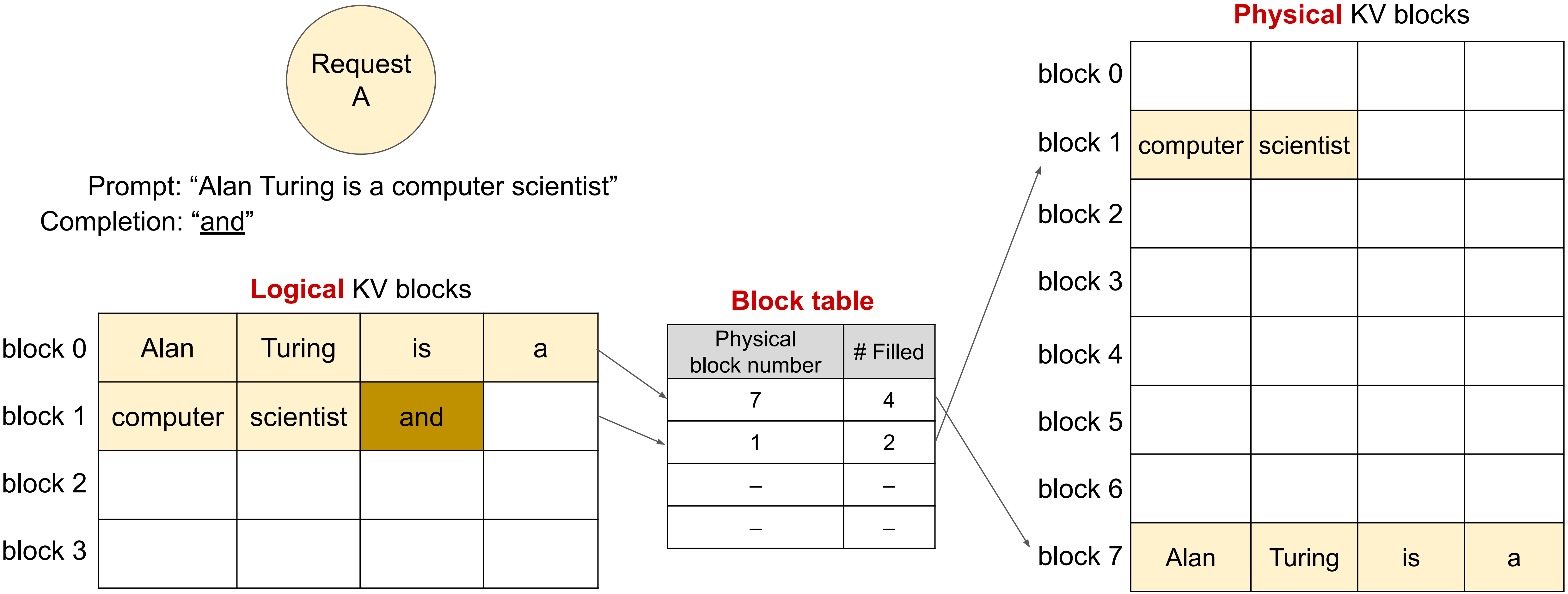


10-15% **slowdown** in GPU kernel latency due to memory indirection

Memory management with PagedAttention



Memory management with PagedAttention



Memory management with PagedAttention

Request A

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

Logical KV 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 KV blocks

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

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	mathema tician
block 2				
block 3				

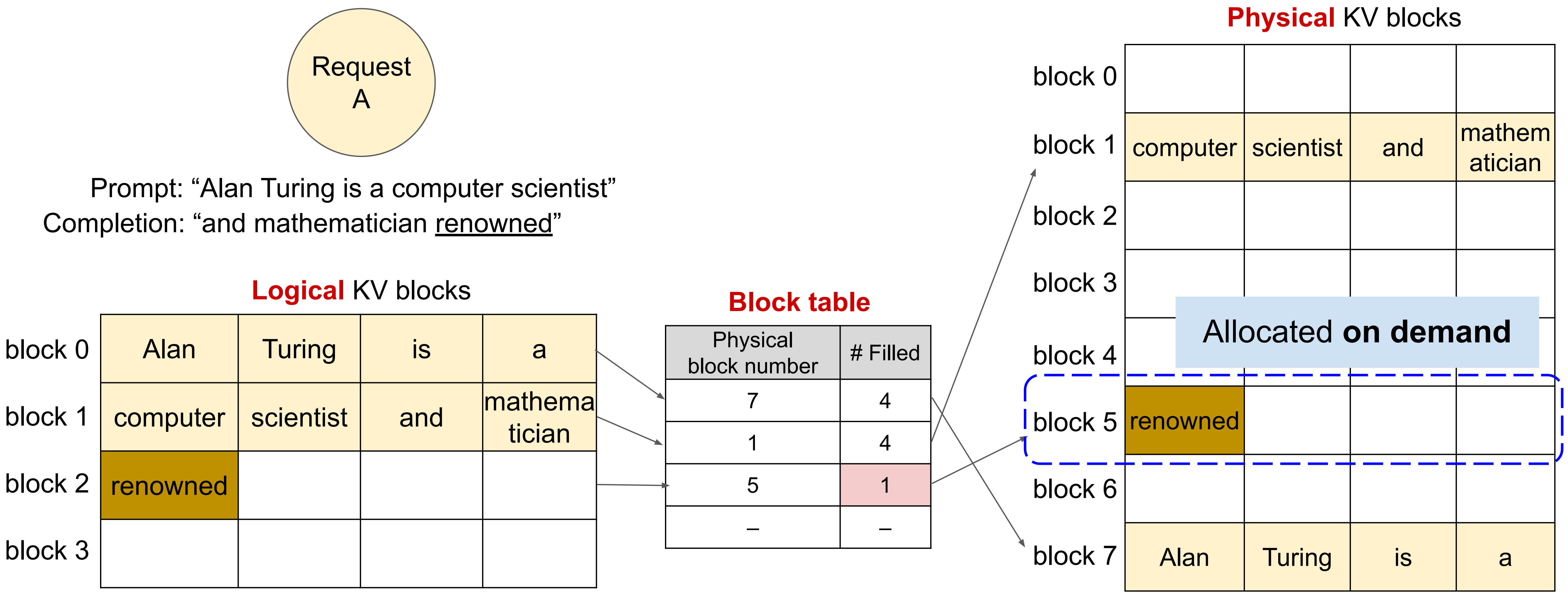
Block table

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

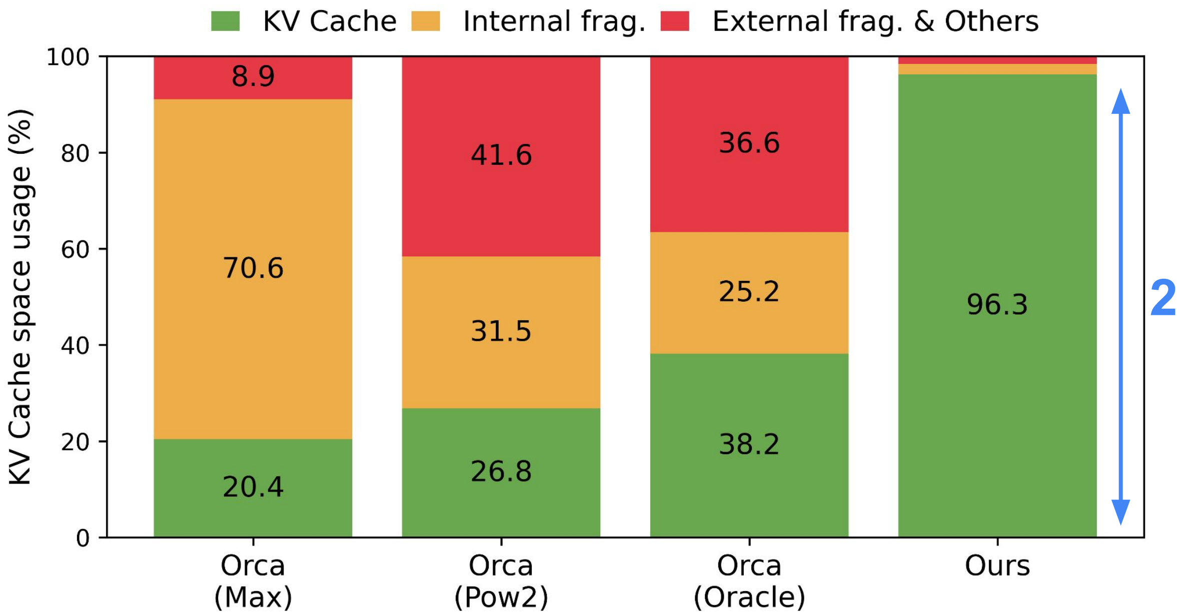


Memory efficiency of PagedAttention

- Minimal internal fragmentation
  - Only happens at the last block of a sequence
  - **# wasted tokens / seq < block size**
    - Sequence: O(100) – O(1000) tokens
    - Block size: O(10) tokens
- No external fragmentation

Alan	Turing	is	a
computer	scientist	and	mathematician
renowned			

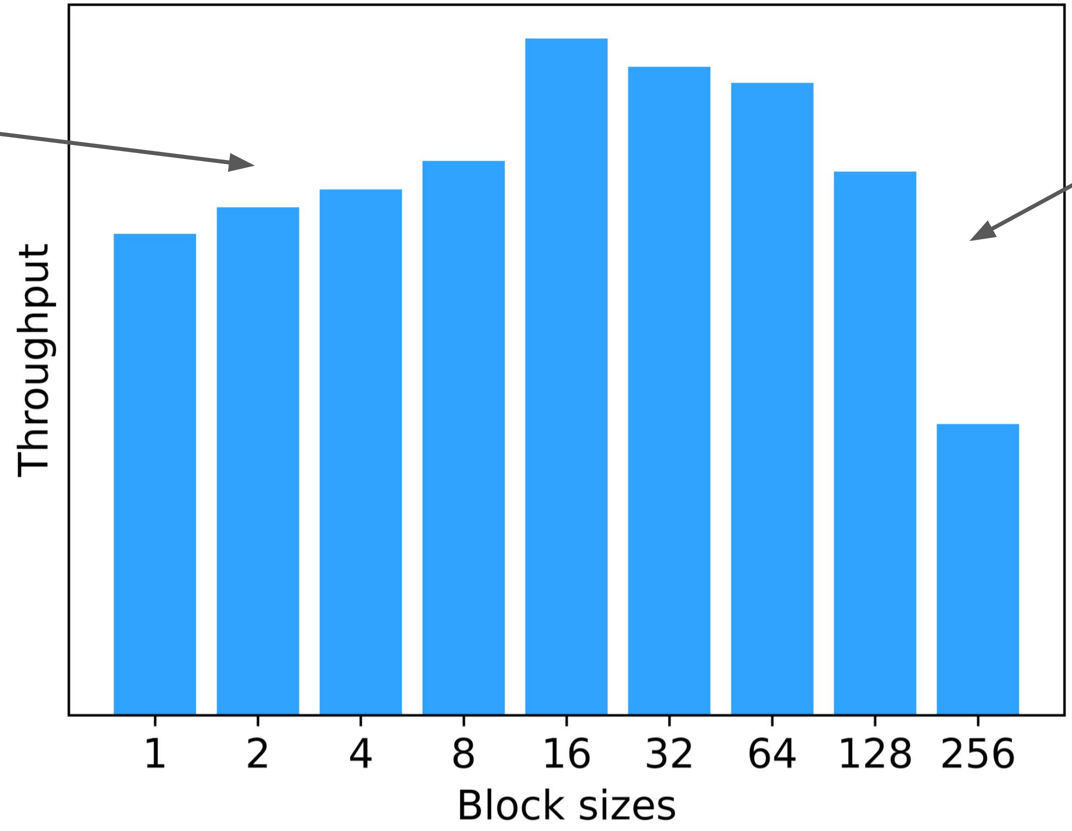
Internal fragmentation



2.5-5x improvement

Configuring the block size

Low spatial locality

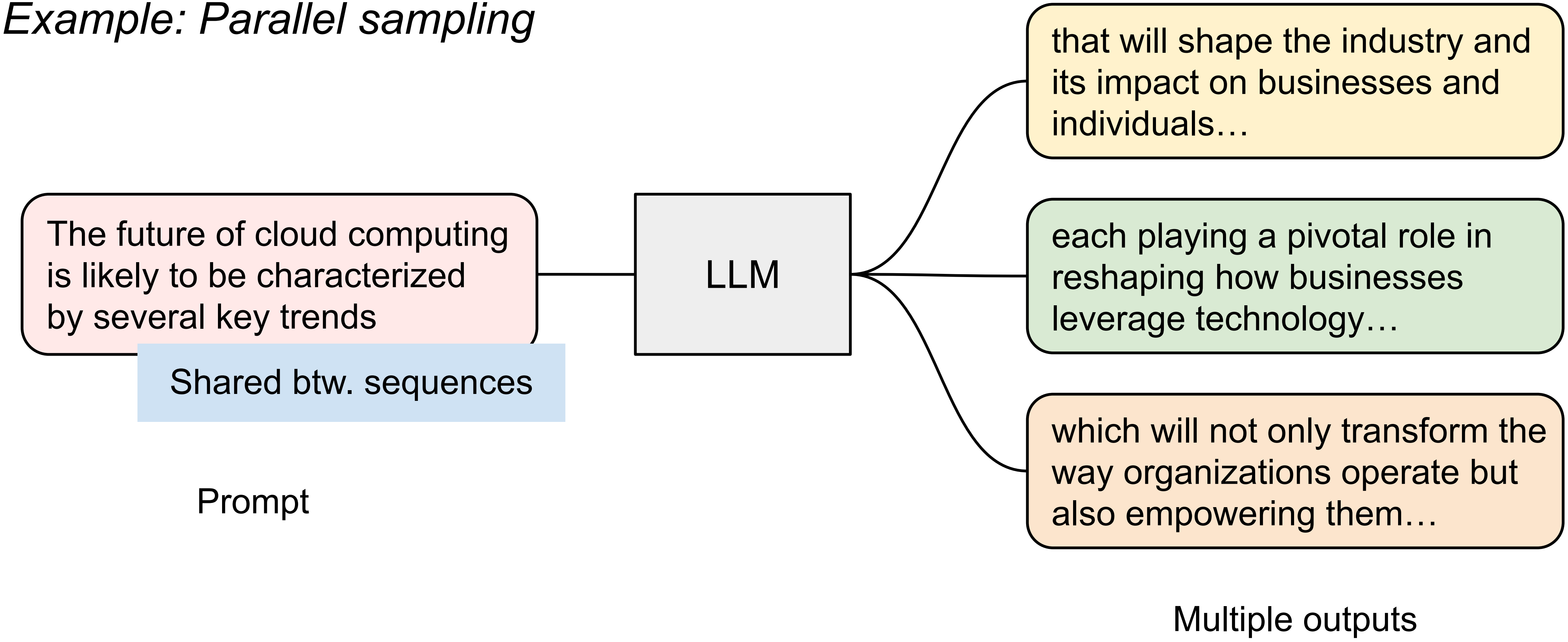


Large internal fragmentation

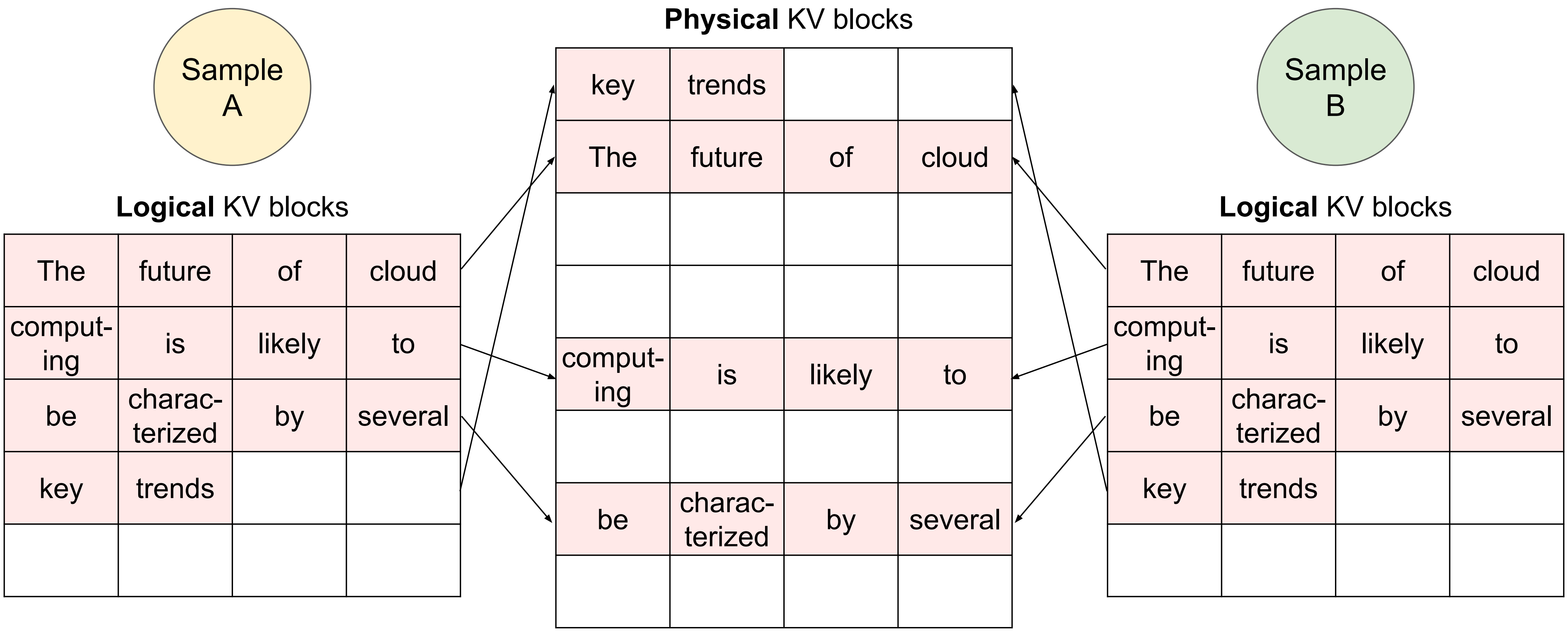
- **Block size 16** works generally well in practice

Paging enables sharing

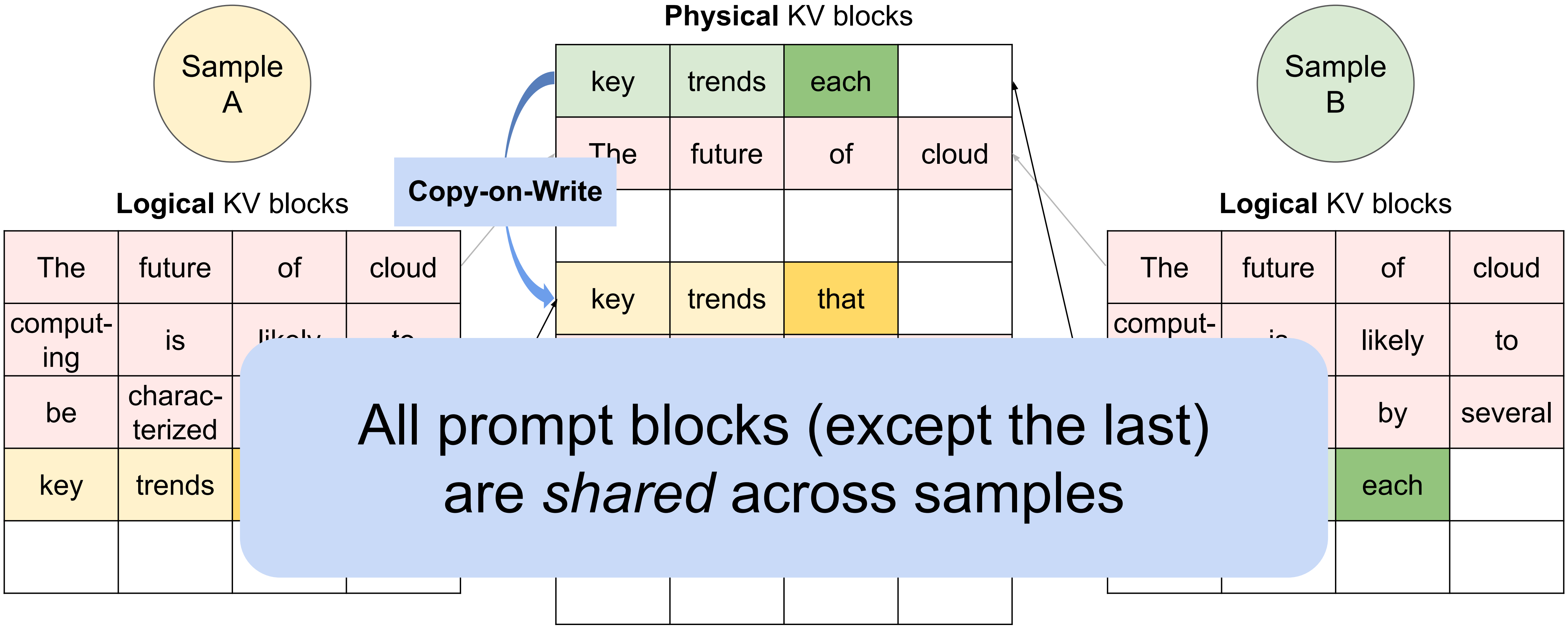
Example: Parallel sampling



Sharing KV blocks

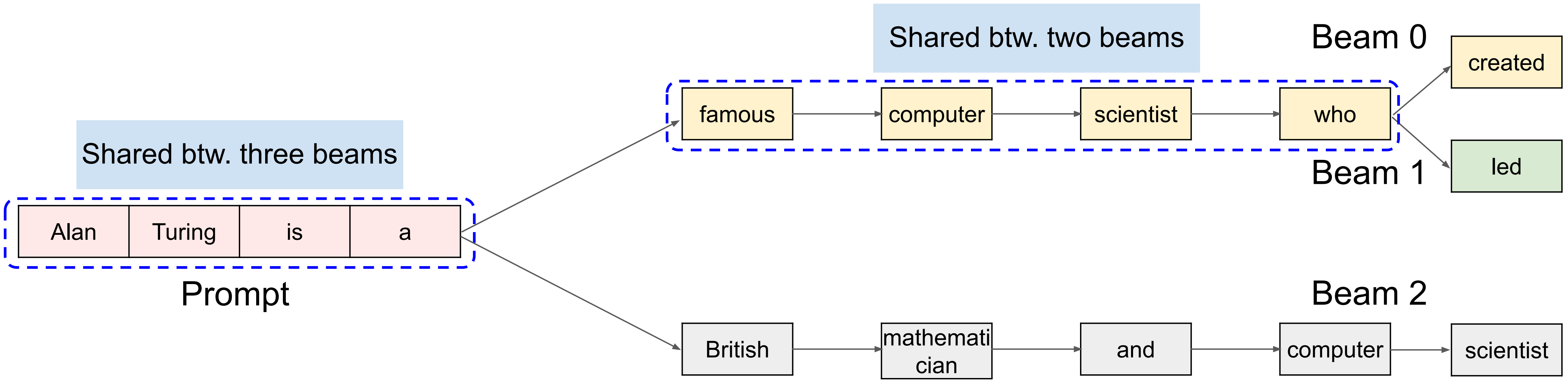


Sharing KV blocks



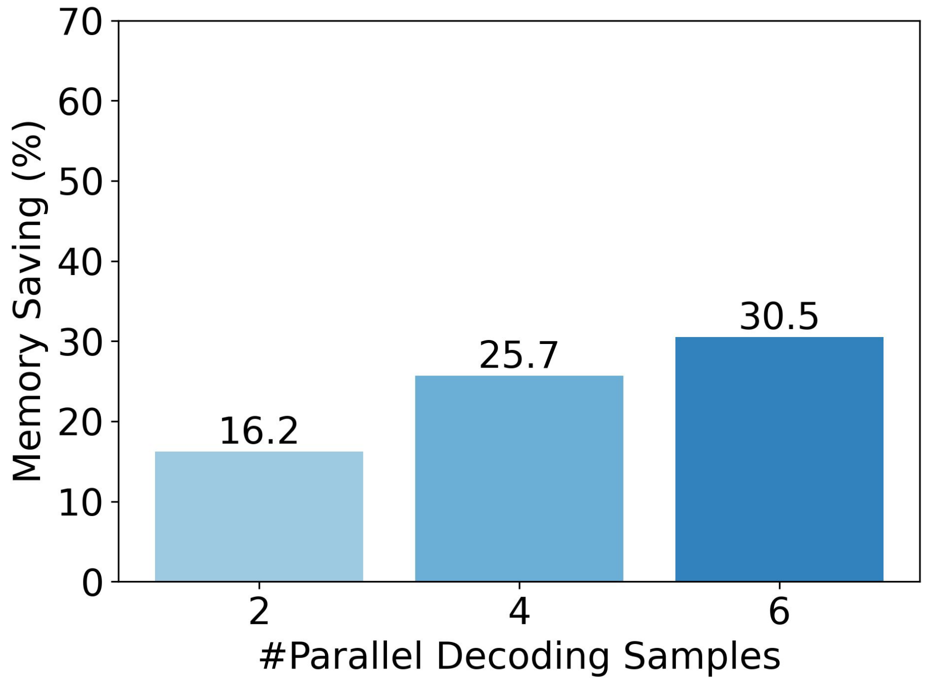
All prompt blocks (except the last) are *shared* across samples

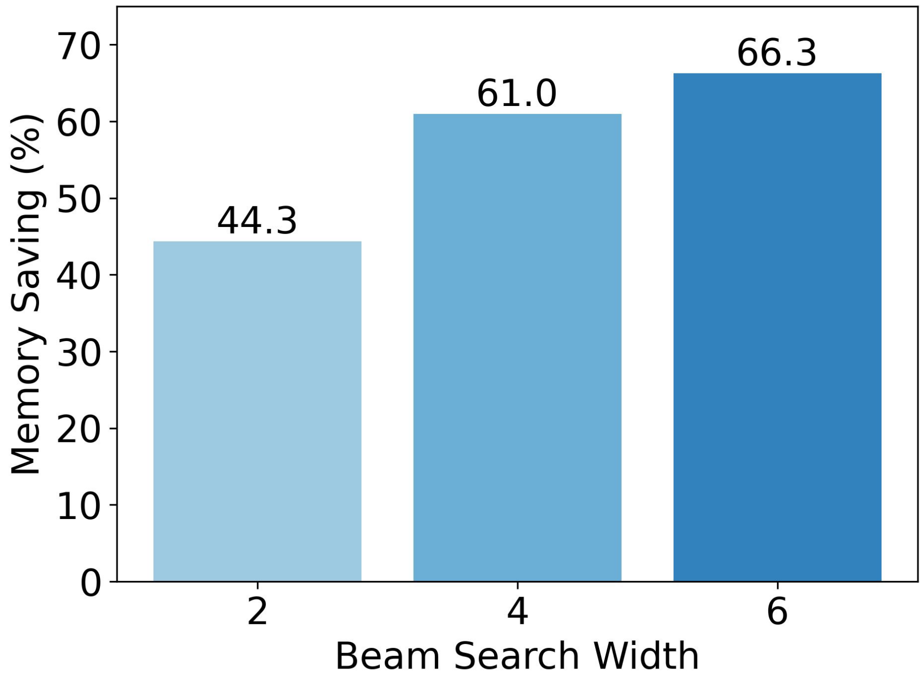
More complex sharing: beam search



- Similar to process tree (fork & kill)
- Efficiently supported by paged attention and copy-on-write mechanism

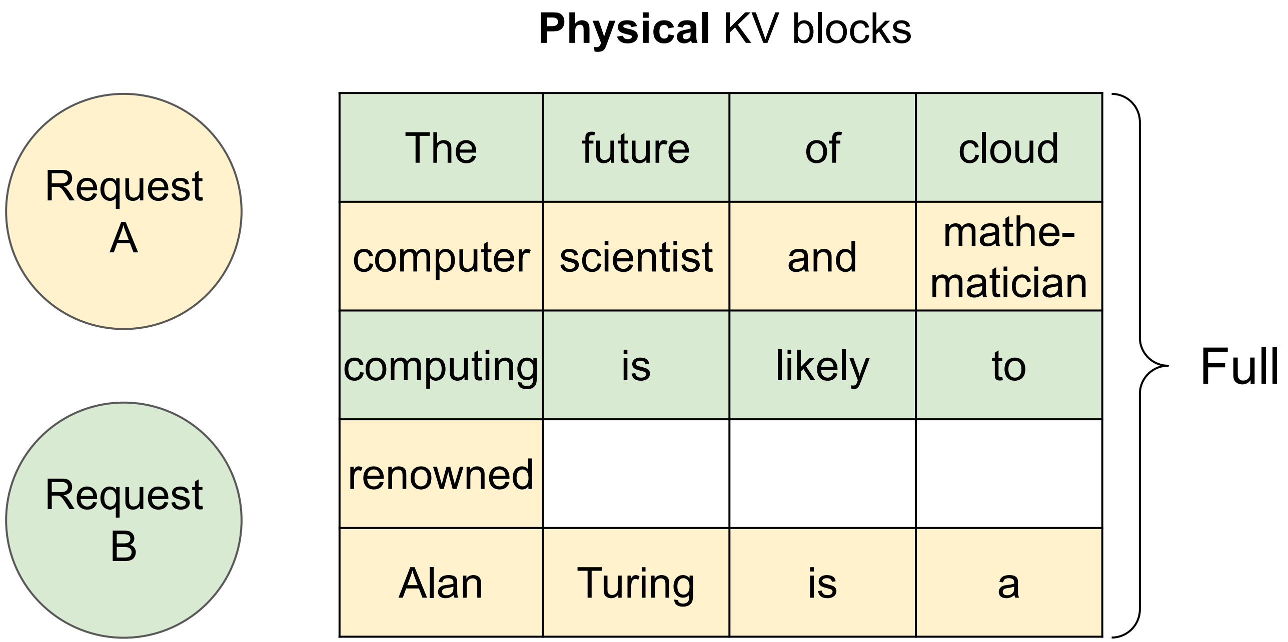
Memory saving via sharing



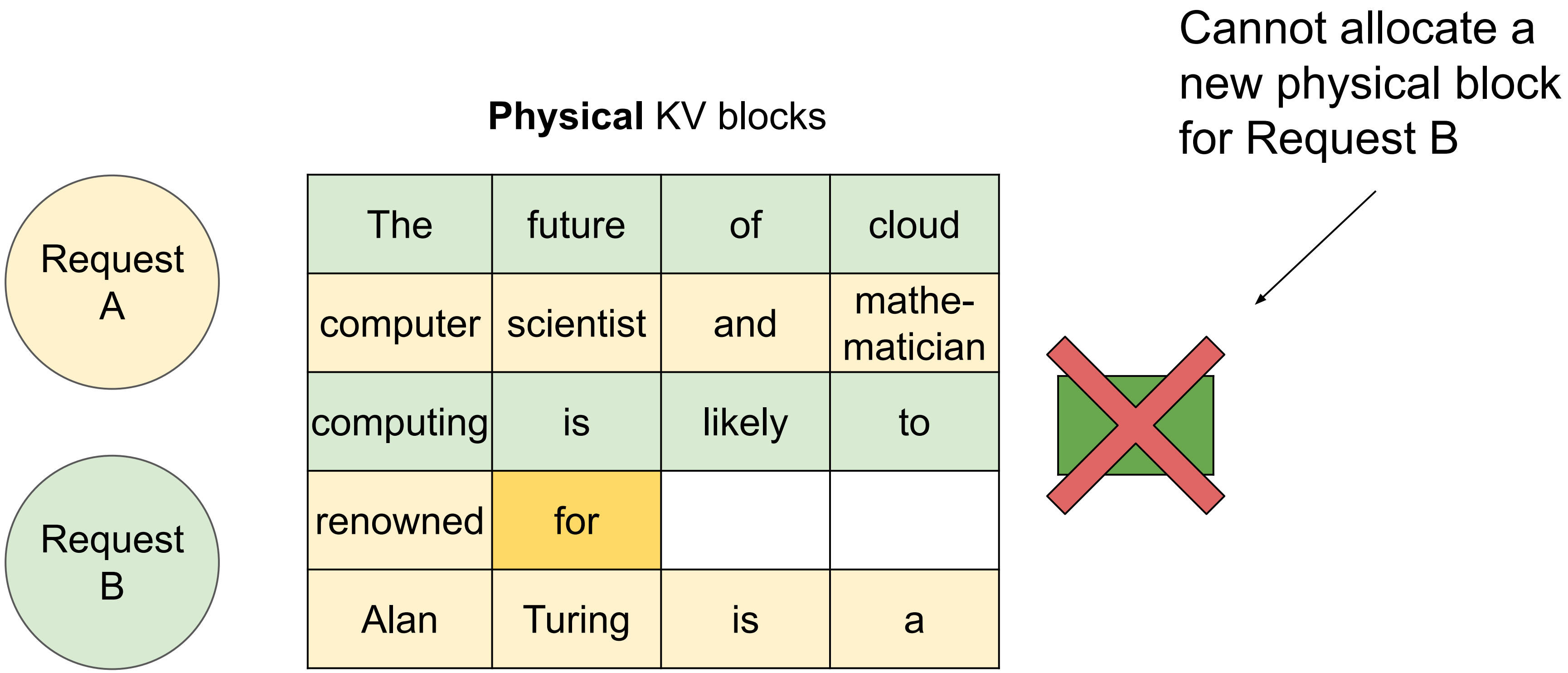


Percentage = (#blocks saved by sharing) / (#total blocks without sharing)
OPT-13B on 1x A100-40G with ShareGPT dataset

Out of KV Block Memory



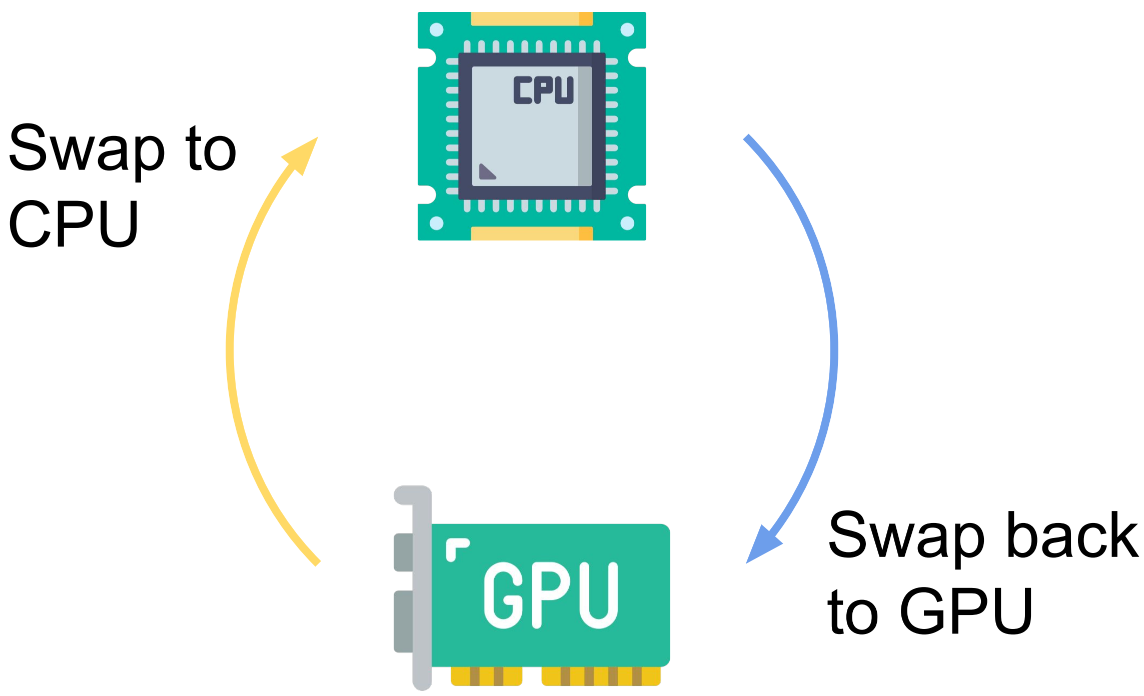
Out of KV Block Memory



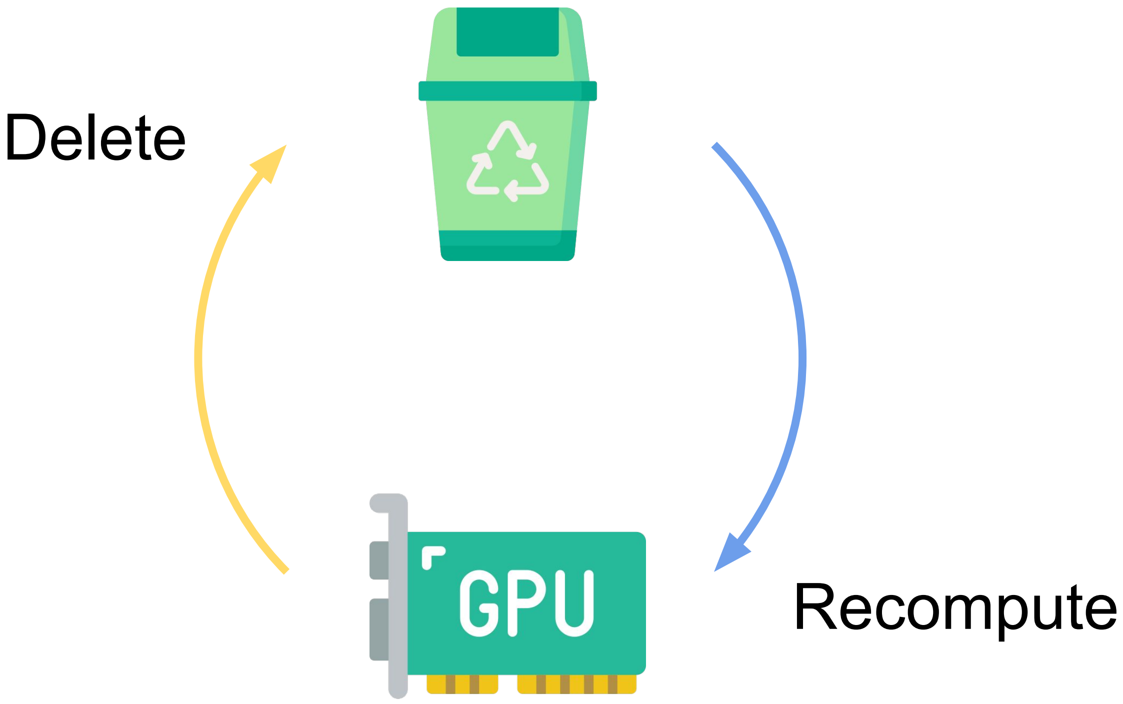
Request Preemption & Recovery

Goal: Free some requests' KV cache to let others run first.

Option 1: Swapping



Option 2: Recomputation



Notes on Preemption & Recovery

Swap/recompute the whole request, since all previous tokens are required every step.

Swapping: smaller block sizes → higher overhead due to small data transfers.

Recomputation: surprisingly fast since all token's KV cache can be computed in parallel.

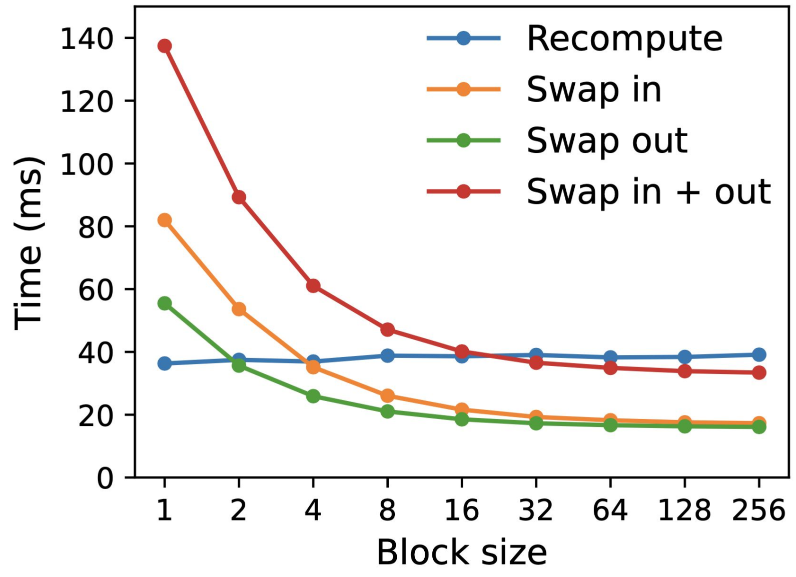
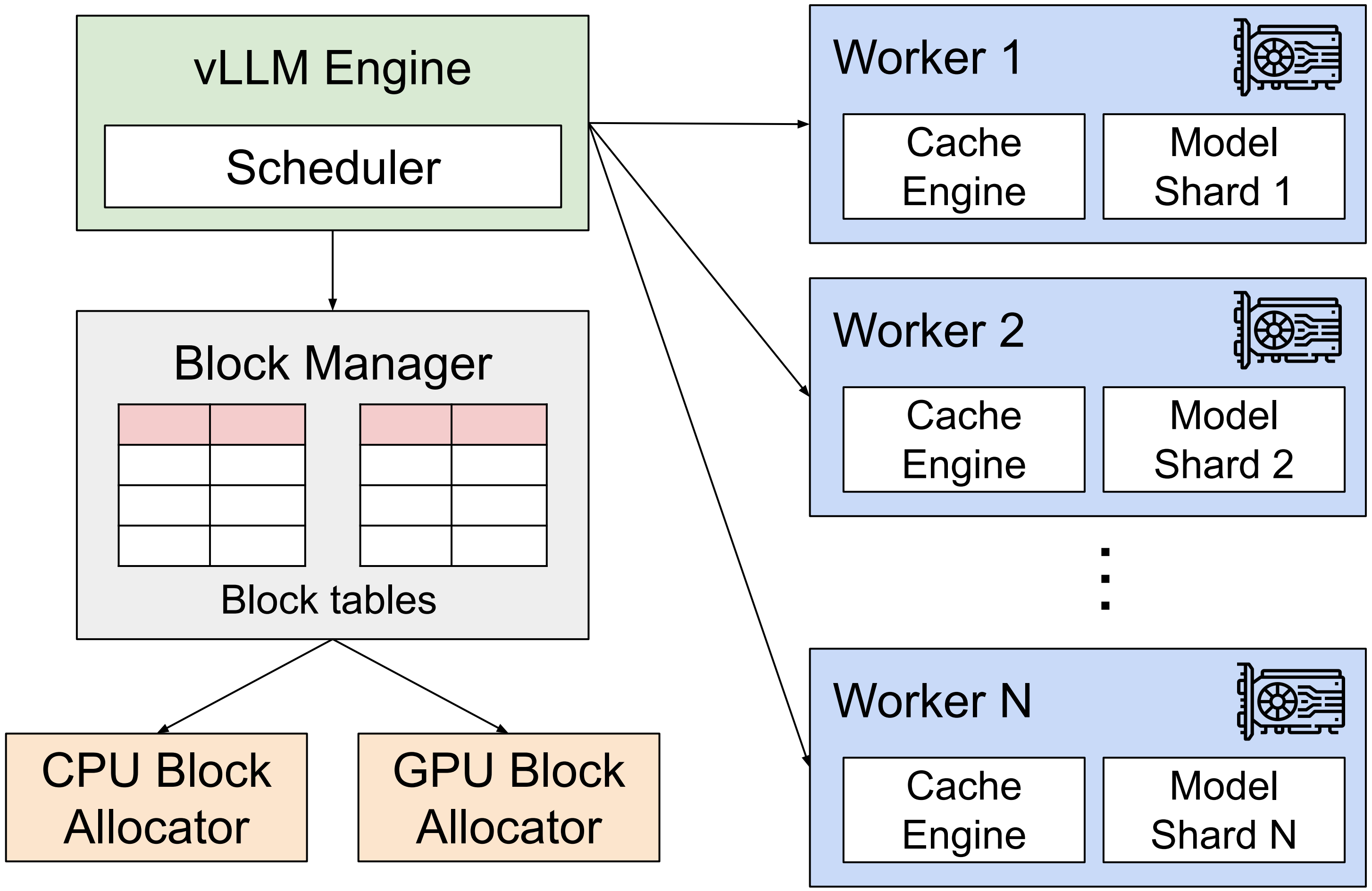
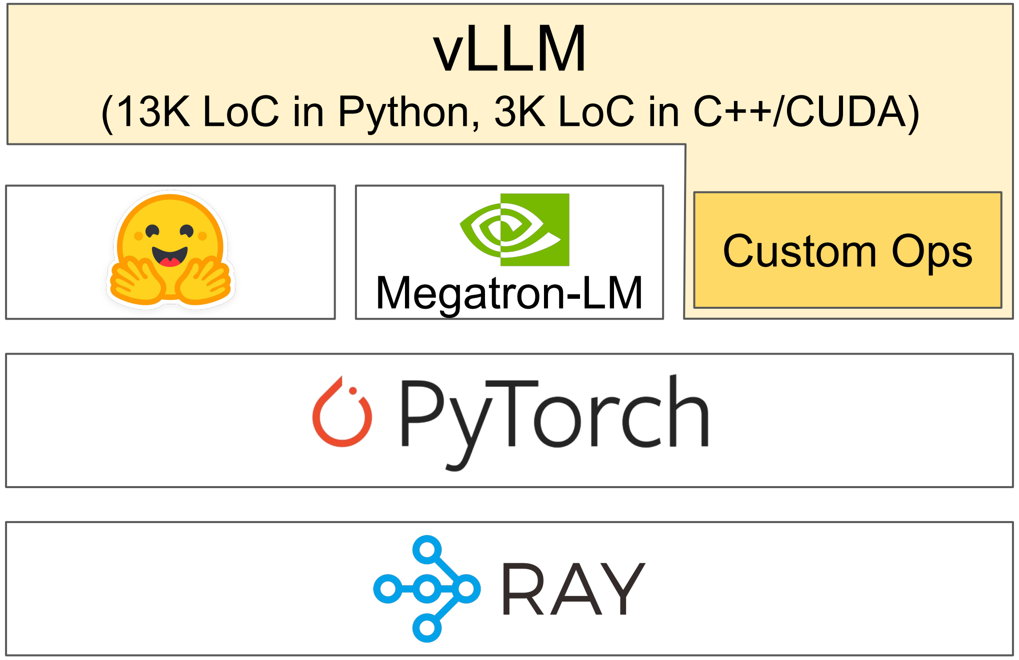


Figure: *Swap/Recomputation latency of 256 tokens.*

vLLM Strategy: Use recomputation when possible with FCFS policy

vLLM Distributed System Architecture & Implementation





Evaluation – Settings

Metric: Serving throughput

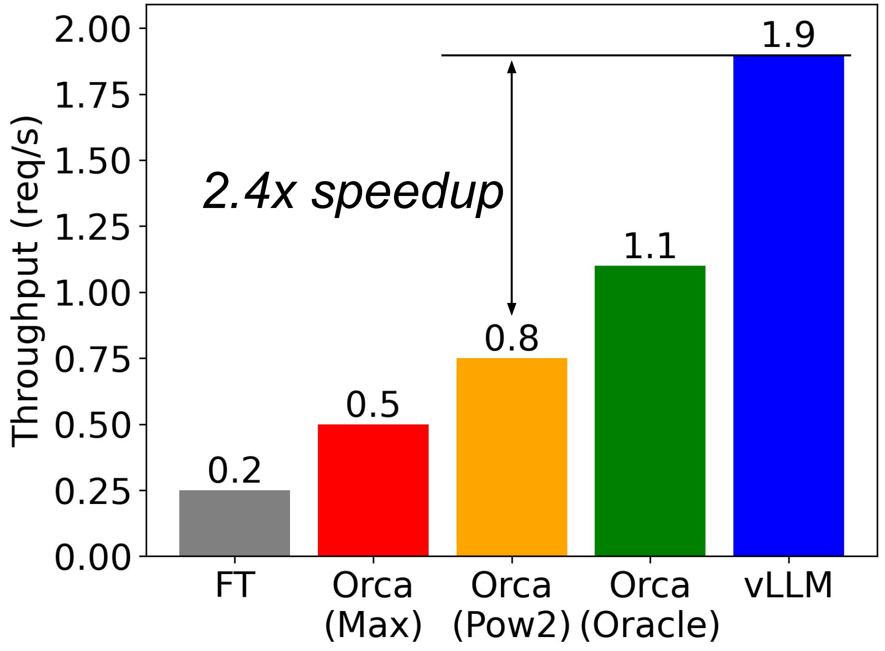
Input/Output Length Distribution

- Alpaca dataset (instruction-following)
- ShareGPT dataset (conversation)

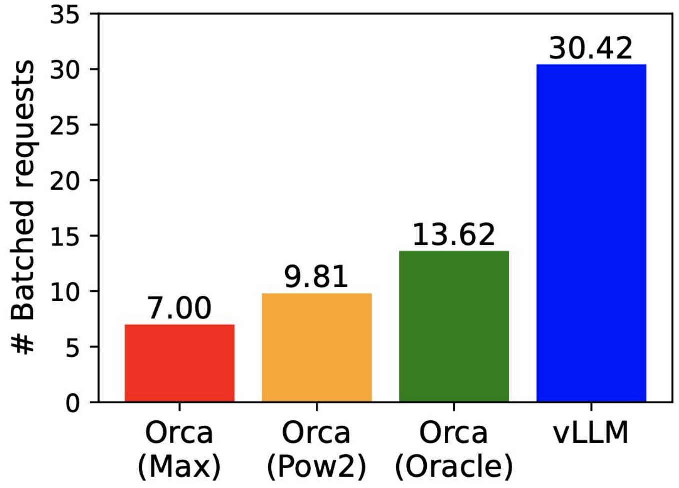
Baselines

- NVIDIA FasterTransformer (FT)
- Orca
  - Oracle: No over-reserve and know exact output lengths.
  - Pow2: Over-reserve the space for outputs by at most 2x.
  - Max: Over-reserve to the maximum possible output length.

Throughput – Greedy Decoding

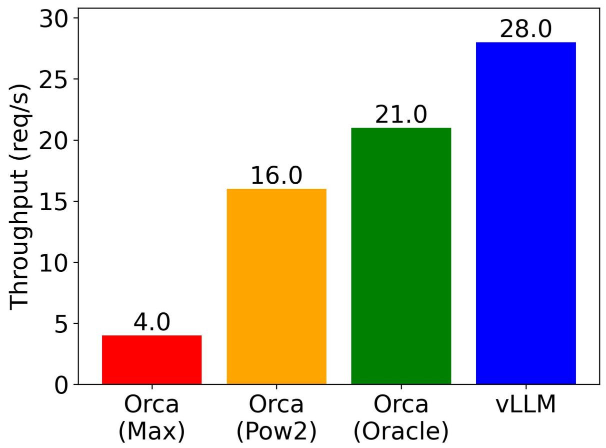


OPT-13B on 1xA100 40G with ShareGPT trace.

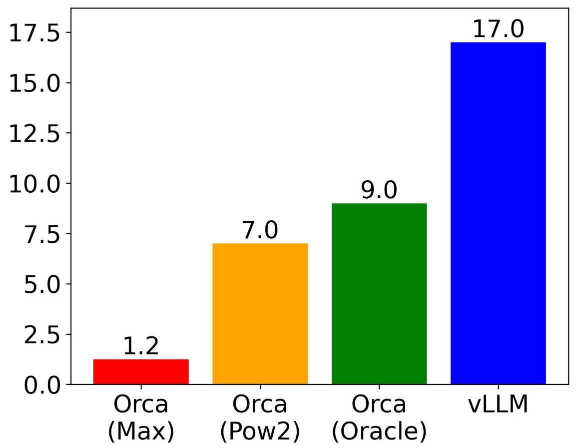


Average number of batched requests.

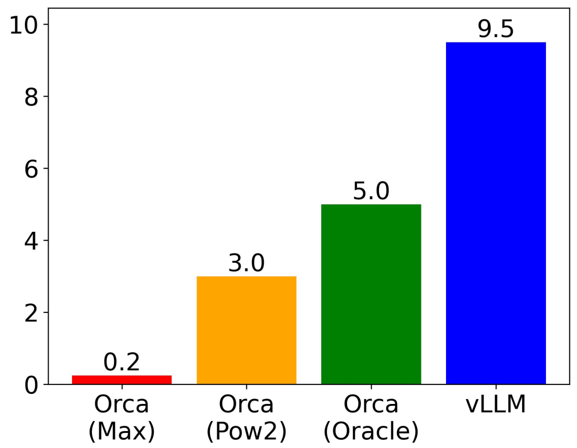
Throughput – Beam Search



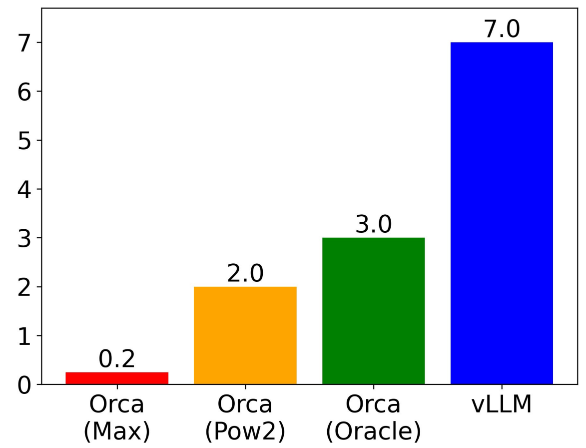
No beam search
***1.8x** speedup*



Beam width = 2
***2.4x** speedup*



Beam width = 4
***3.2x** speedup*



Beam width = 6
***3.5x** speedup*

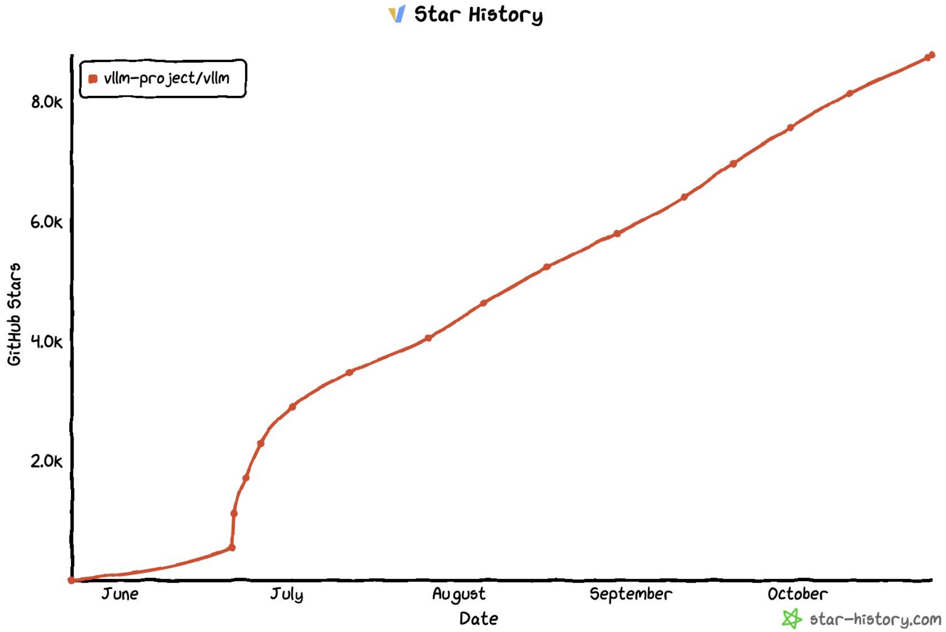
OPT-13B on 1xA100 40G with Alpaca trace.
Speedup: vLLM v.s. Orca(Pow2)

vLLM Open-Source Adoption

github.com/vllm-project/vllm

$ pip install vllm

8.8K Stars 95+ Contributors



Open-Source Projects

lm-sys/FastChat AI2 allenai/open-instruct

Companies

anyscale BentoML

C3.ai databricks Google Cloud

Lepton AI Meta Microsoft

Modal NVIDIA replicate

scale Twelve Labs Uber

...

Lecture Outline

- Memory management in GPUs for LLM serving
- **Augmented model memory through retrieval augmented generation (RAG)**

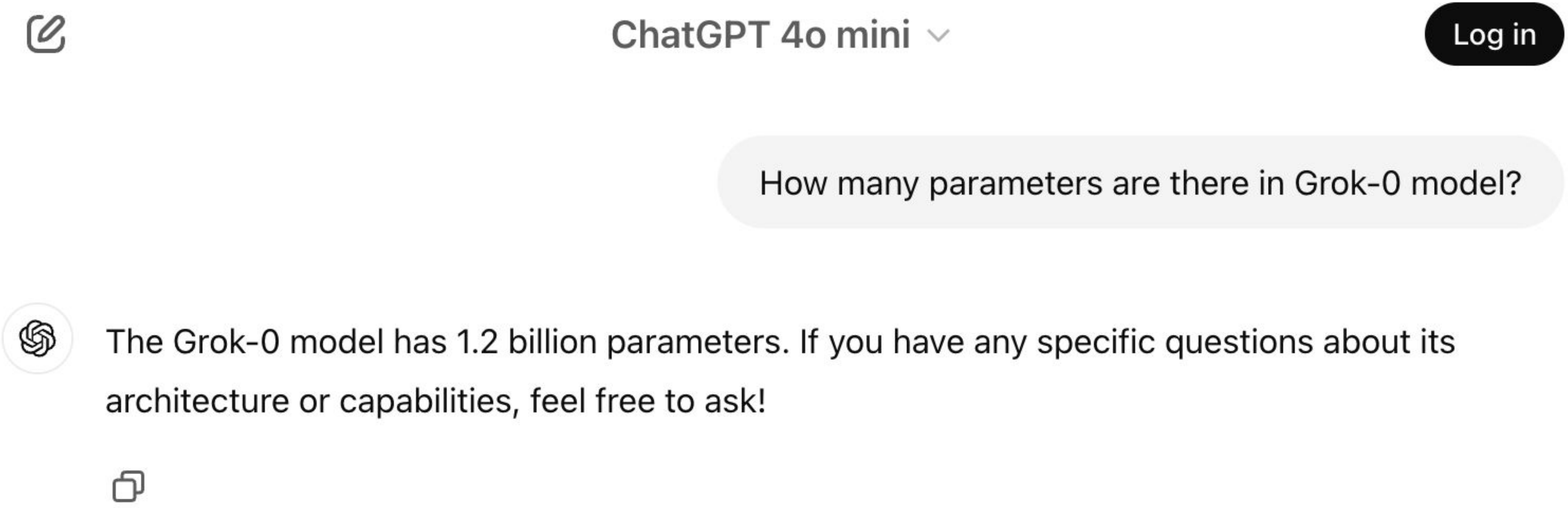
Pre-trained LLMs are great

- Capture knowledge in parameters
- Applicably broadly

But:

- They hallucinate
- Struggle to access new information and use updates

Sample Interaction



Inaccurate result as the LLM wasn’t trained with the data!

Possible Fixes

- Fine-tuning
- Prompt engineering
- Retrieval augmented generation

Fine Tuning

- Run additional training steps with additional data that is now available
  - Example: LLM trained in English cannot output French. To teach new concepts, add new training data to a pre-trained model and fine tune the model
- Downsides of fine tuning:
  - Expensive
  - Number of parameters might not be sufficient to incorporate new information
  - Language model can “forget” earlier capabilities

Prompt Engineering

Instructions

You're an assistant trained to answer questions using the given context.

Context

Context:
"The engine powering Grok is Grok-1, our frontier LLM, which we developed over the last four months. Grok-1 has gone through many iterations over this span of time.
After announcing xAI, we trained a prototype LLM (Grok-0) with 33 billion parameters. This early model approaches LLaMA 2 (70B) capabilities on standard LM benchmarks but uses only half of its training resources. In the last two months, we have made significant improvements in reasoning and coding capabilities leading up to Grok-1, a state-of-the-art language model that is significantly more powerful, achieving 63.2% on the HumanEval coding task and 73% on MMLU.
To understand the capability improvements we made with Grok-1, we have conducted a series of evaluations using a few standard machine learning benchmarks designed to measure math and reasoning abilities.

GSM8k: Middle school math word problems, (Cobbe et al. 2021), using the chain-of-thought prompt.
MMLU: Multidisciplinary multiple choice questions, (Hendrycks et al. 2021), provided 5-shot in-context examples.
HumanEval: Python code completion task, (Chen et al. 2021), zero-shot evaluated for pass@1.
MATH: Middle school and high school mathematics problems written in LaTeX, (Hendrycks et al. 2021), prompted with a fixed 4-shot prompt."

Question

Answer the following question: "How many parameters are there in Grok-0?"

Prompt

Answer

Grok-0, the prototype LLM mentioned in the provided context, is stated to have been trained with 33 billion parameters.

Possible to teach a new model to incorporate new information but limited by context lengths!

Retrieval is great

- Externally-retrieved information is useful for many NLP tasks
  - Precise and accurate knowledge access mechanism
  - Easy to update with new information

But:

- Lack ability to generate or “creativity”
- Need task-specific ways to integrate with applications/downstream tasks

RAG: Combine Retrieval with Generation

- Jointly learn to **retrieve** and **generate**
  - General recipe for any generation task
  - Latent retrieval - no labels needed for docs; just retrieval

RAG models combine parametric and non-parametric memory for knowledge intensive tasks

Widely used?

ChatGPT

Answer the following question: how many parameters does the Grok-0 LLM contain?

Searched 5 sites

"how many parameters does Grok-0 LLM contain"
bing.com

DailyAI — Grok LLM details and how it stacks up against ChatGPT | DailyAI
dailyai.com

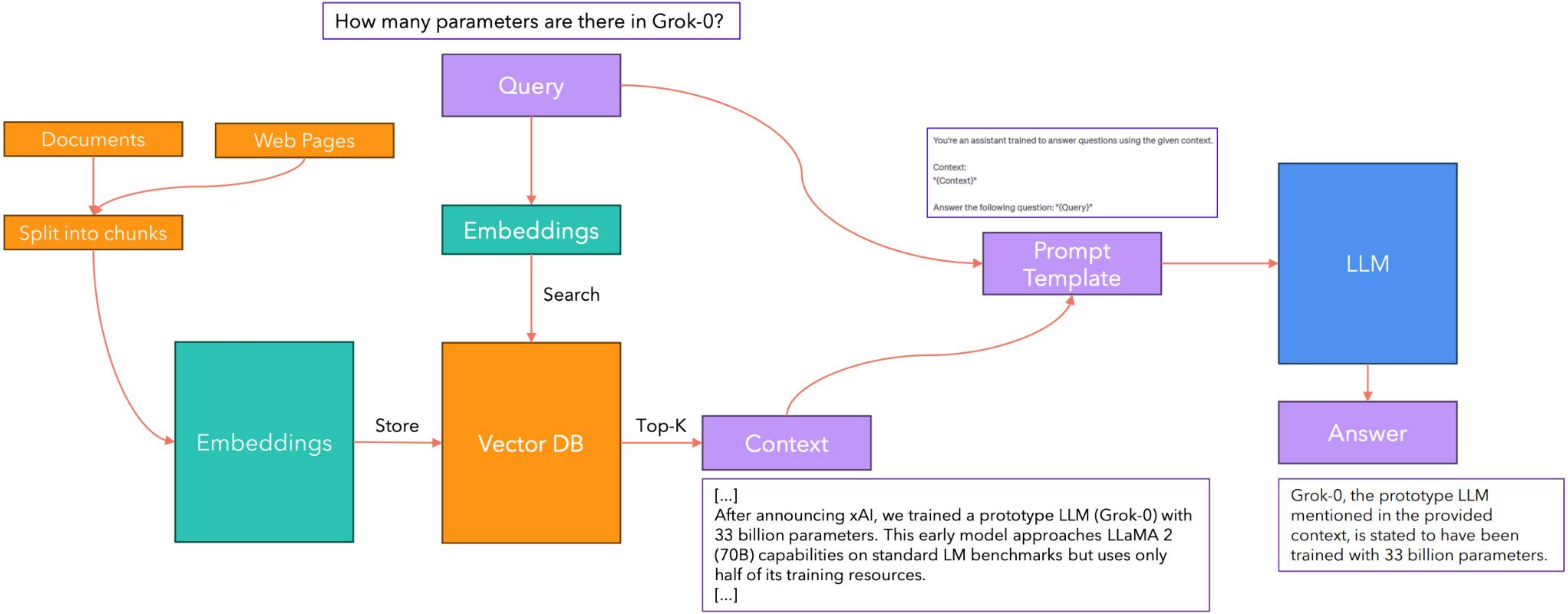
xAI — Announcing Grok
x.ai

Weights & Biases — xAI Launches Grok, a New LLM with Real-Time Knowledge
wandb.ai

InfoQ — xAI Introduces Large Language Model Grok - InfoQ
infoq.com

The Grok-0 model, developed by xAI, contains 33 billion parameters. This was the prototype for their later models, including Grok-1, which is expected to have a larger parameter count but specific details have not been disclosed. Despite its smaller size compared to models like GPT-3.5 (175 billion parameters), Grok-0 has shown competitive performance on several machine learning benchmarks DAILYAI INFOQ .

RAG Pipeline Illustration



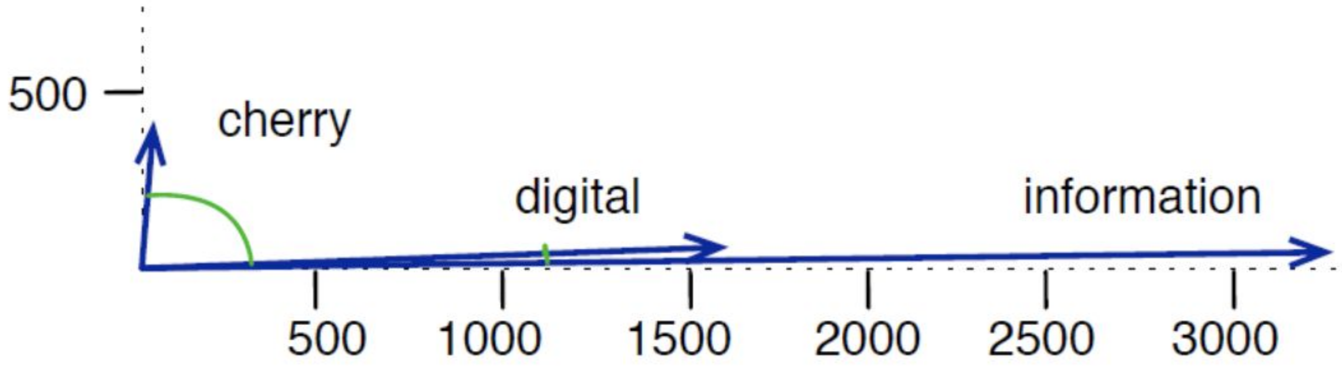
RAG Components

Need the following components:

- Pretrained generator model or language model
- Pretrained retriever model for generating embeddings of query, info
- Indexed knowledge base of text documents
- Use retriever model to identify relevant documents efficiently

Embedding Vectors

- Map semantic information of words to a high-dimensional space
- Related words will have "similar" vectors (e.g., cosine similarity or Euclidean distance) and can be used in the same context
- Words that appear in the same context can be inferred to be similar



Source: Speech and Language Processing 3rd Edition Draft, Dan Jurafsky and James H. Martin

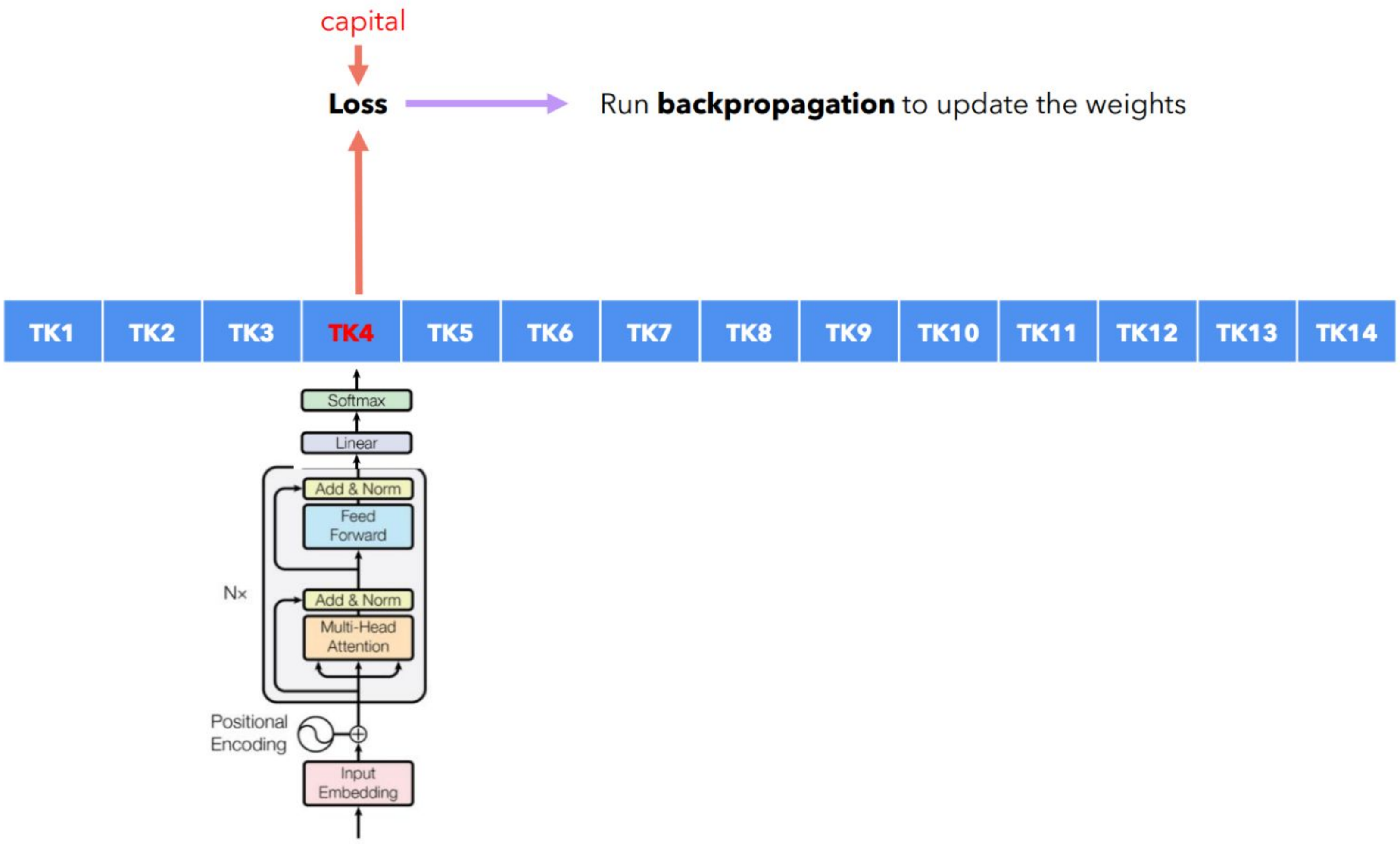
Generating Embeddings

- Train a model that can predict the missing word that is masked out
- For example, “Rome is the ___ of Italy, which is why it hosts many government buildings”
- Train an encode model (e.g., BERT)
- Use the Self-Attention mechanism to relate all the input tokens with each other

Generating Embeddings

Target (1 token):

Output (14 tokens):



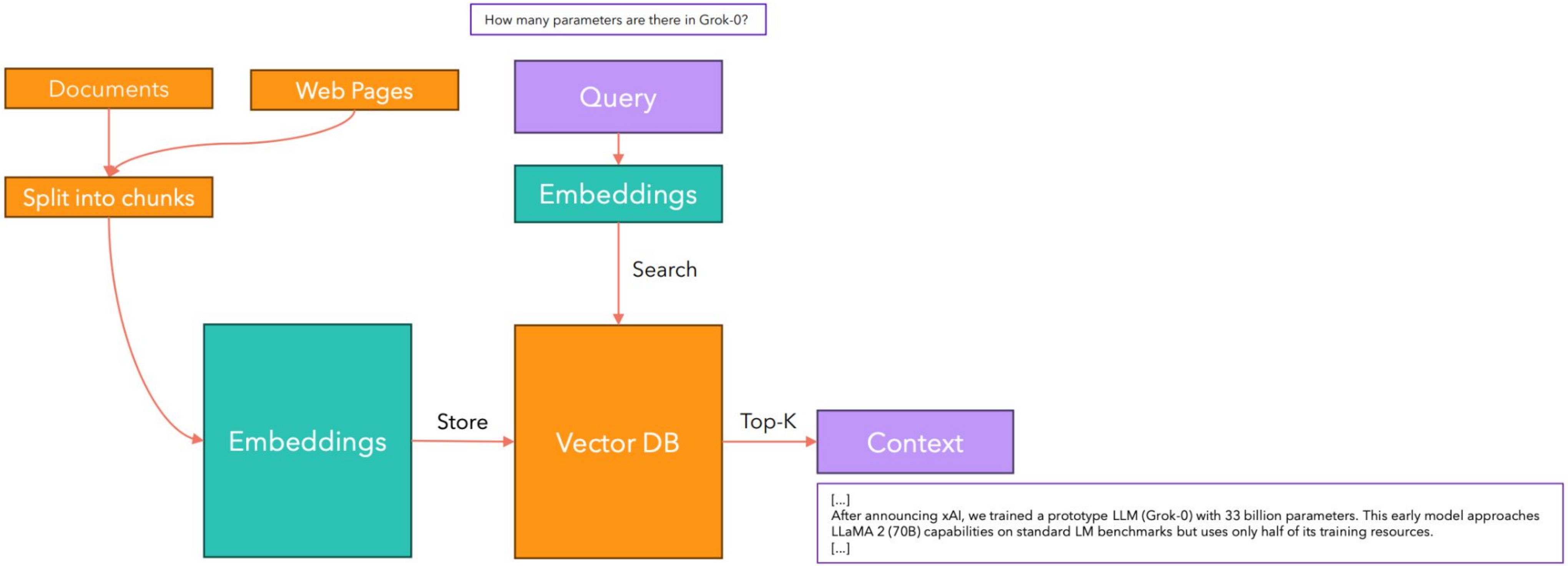
Input (14 tokens): Rome is the [mask] of Italy, which is why it hosts many government buildings.

Sentence Embeddings

- Use the Self-Attention mechanism also to capture entire sentences
- For example, encode every word and take the average of the embeddings
- But previous optimization is insufficient
  - Encodings should be such that the average across the words produces high cosine similarity with related words
- Sentence BERT: BERT specifically trained to generate similar embeddings for equivalent sentences

Vector DB

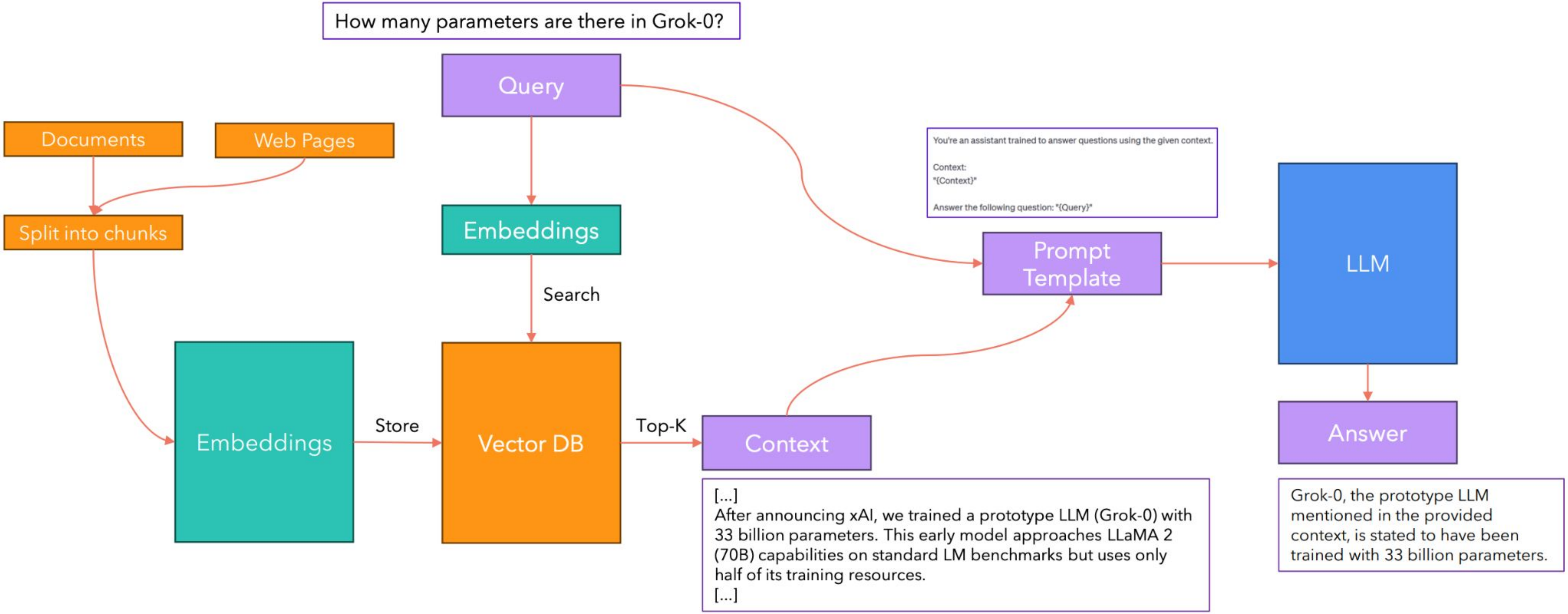
- Stores embedding vectors and supports similarity queries
- Used already for finding similar songs (Spotify), products (Amazon)



K-Nearest Neighbor Search

- Naive approaches uses exhaustive pairwise comparisons
- Approximate searches reduce the search space but at the cost of accuracy
  - For example, Hierarchical Navigable Small Worlds (HNSW)
  - Navigable small worlds builds a graph of entries with links connecting similar entries
    - Insert new entries by starting at random points and moving along directions that increase similarities
    - Establish connections between new entry and those that are closest to it
  - HNSW establishes a hierarchy of graphs with fewer elements in them, with each element randomly selected to be in the higher level graph
  - Search starts with the top-level of the hierarchy and proceeds to lower levels

RAG Pipeline Illustration



Updating LLMs using Fine Tuning and RAG