Systems Problems in ML Inference

COS 316: Principles of Computer System Design
Lecture 19

Nicolaas Kaashoek

Inference Presents New Challenges

Large language models have two key step:

- Training: Creating the model
- Inference: Using the model

Training challenges: Parallelism, Fault Tolerance, Storage

Inference challenges? Very different!

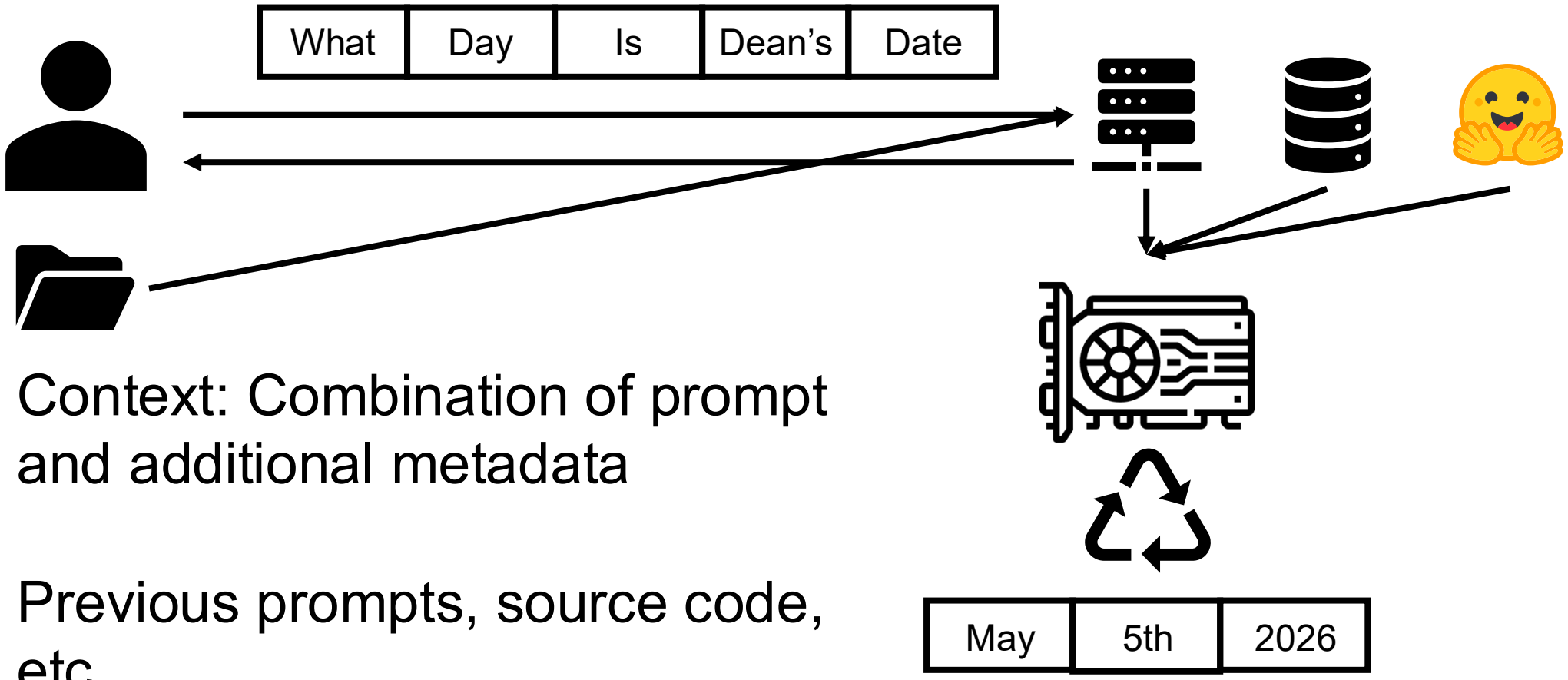
What is Inference

Recall: We train LLMs to do prediction

Example: “What day is Dean’s Date? ____”

Inference is the process that gets from input *context* to response tokens

Inference: Bird's Eye View



Context: Combination of prompt and additional metadata

Previous prompts, source code, etc.

Post-Inference Work

Lots of post-inference steps:

- Your context gets stored somewhere
- Feedback sent to the training/refinement pipeline
- Update additional metadata to improve future inferences
- Guardrails to check the output

Inference Metrics

Important inference metrics are latency based

Time to First Token (TTFT):
- How long to go from all data to the first output token

Time Between Tokens (TBT):
- How long each iteration of the loop takes

End-to-End Latency: Total time from client request to response

Inference: Pre-Fill and Decode

Inference broken into two stages, Pre-Fill and Decode

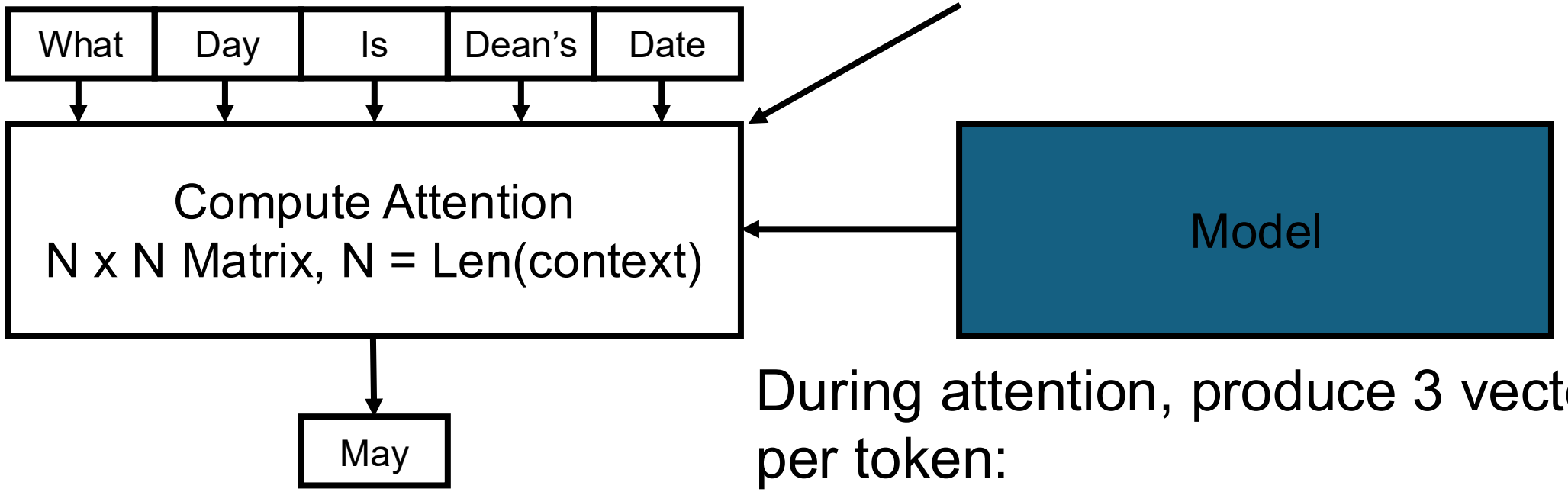
Pre-Fill: Setup a cache and get the first token (TTFT)

Decode: The loop that generates the next N tokens (TBT)

Pre-Fill and Decode have wildly different performance characteristics!

Pre-Fill

Attention step is highly parallel!



KV Cache

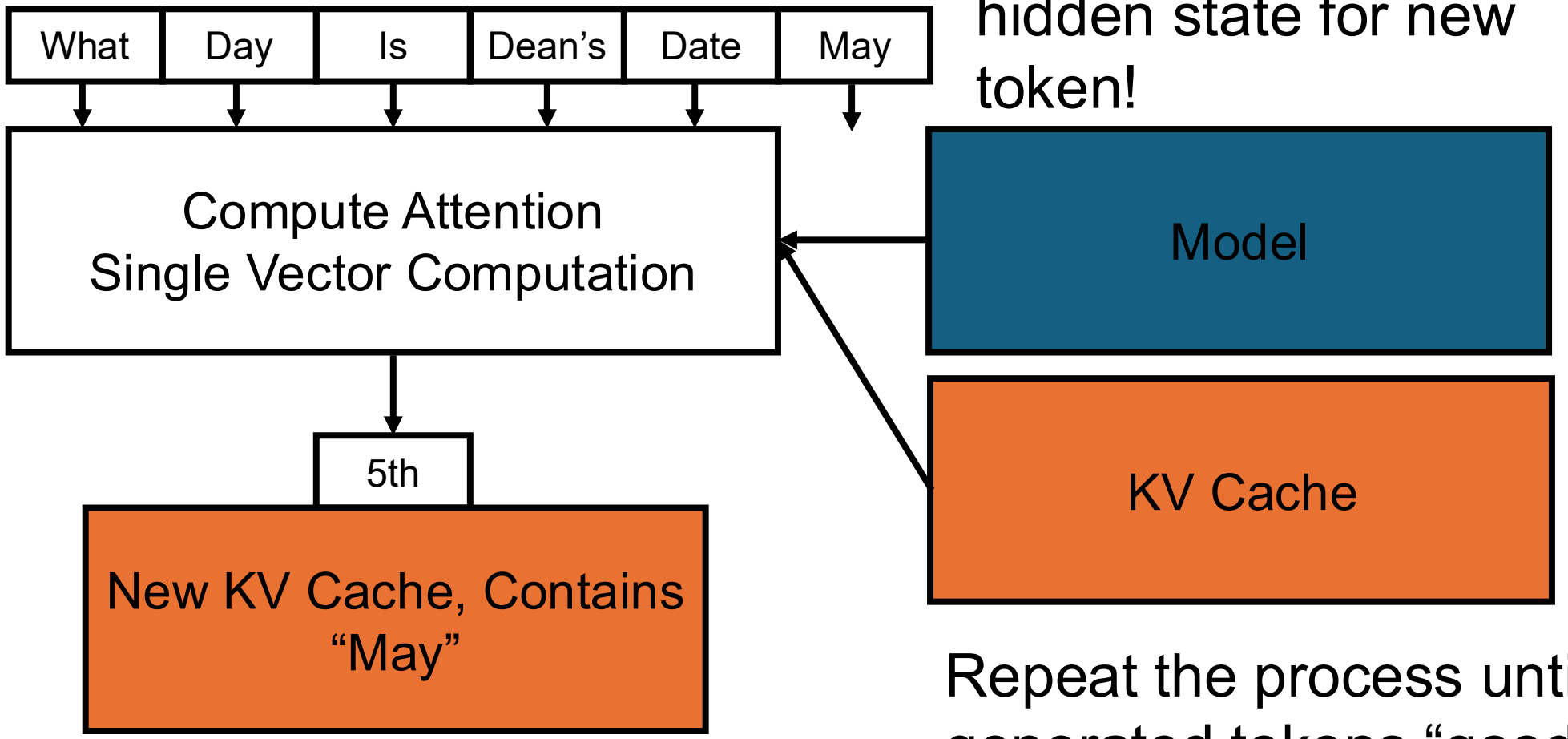
Layer 1: $K_{0 \rightarrow 4}$ | $V_{0 \rightarrow 4}$
Layer 2: $K_{0 \rightarrow 4}$ | $V_{0 \rightarrow 4}$
...
Layer N: $K_{0 \rightarrow 4}$ | $V_{0 \rightarrow 4}$

During attention, produce 3 vectors per token:

- Query (Q), Key (K), Value (V)
- Reuse them with caching!
- KV Cache stores Keys and Values

Previous keys and values used to predict next token

Decode



Only need to compute hidden state for new token!

Repeat the process until generated tokens "good enough"

LLM with/without KV Cache

Computing the KV cache ahead of time is a huge time saver!

- $O(N^2)$ matrix multiplication for each new token, to vector
- Also frees up GPU cores!

Concrete numbers from DeepSeek R1 on Azure:

- Time to First Token (Pre-Fill): 1 second
- Time Between Tokens (Decode): 20ms

Inference vs. Training

Training exploits parallelism to make the job computable

Inference has one parallel task, one sequential!

Training: Do it once, then you’re done

Inference: Do it many, many times

Inference at Scale

How many people are doing inference at any one time?

- ChatGPT: 30k requests per second (2.5 billion per day)

How many GPUs do we need to support this?

- 30k? No way!

Systems Challenges for Inference

Challenge 1: Scheduling/Load Balancing

- How can we make better use of our GPUs to support necessary throughput?

Challenge 2: GPU Memory Management

- How do we properly cache the KV Cache when it gets big and we have *lots* of them

Challenge 3: Prevent attacks on/with LLMs

Challenge 1 - Scheduling

Inference receives lots and lots of requests, constantly

- Pre-Fill eats up GPU cycles (parallelism)
- Decode takes the most time (98% of total inference time)

Challenge: Head of line blocking

- Need lots of GPUs to do the Pre-Fill step
- Can starve the decode step

Solution: Heterogenous Resource Pools

Schedulers are a classic systems problem:

- How do I decide what to run, and where?

Insight: Separate pools of resources for pre-fill and decode

ML Schedulers: Complex!

- Different GPUs have different models and KV Caches
- Inference time is unpredictable
- Different workloads have different characteristics (text vs. image)

Solution: Batching

Each decode cycle uses ~1 GPU core, GPUs have 18k cores!

Insight: Batch!

- Classic systems solution: merge small jobs into big one
- Run decodes from multiple prompts in one batch

Continuous batching: Keep GPUs occupied by constantly sending new decode jobs

Challenge: Multiple KV caches?

Challenge 2: KV Caches Get Big

175 billion parameter model -> Multiple GB KVCache

- 13B parameters: 1.6GB KVCache

Goal: Run many decode jobs on one GPU

Challenge: H100s only have ~80GB of memory!

- No way to fit all those KV caches into memory

Moving data from main memory to GPU memory is slow

Solution: PagedAttention

Storing the KV caches for multiple requests is challenging:

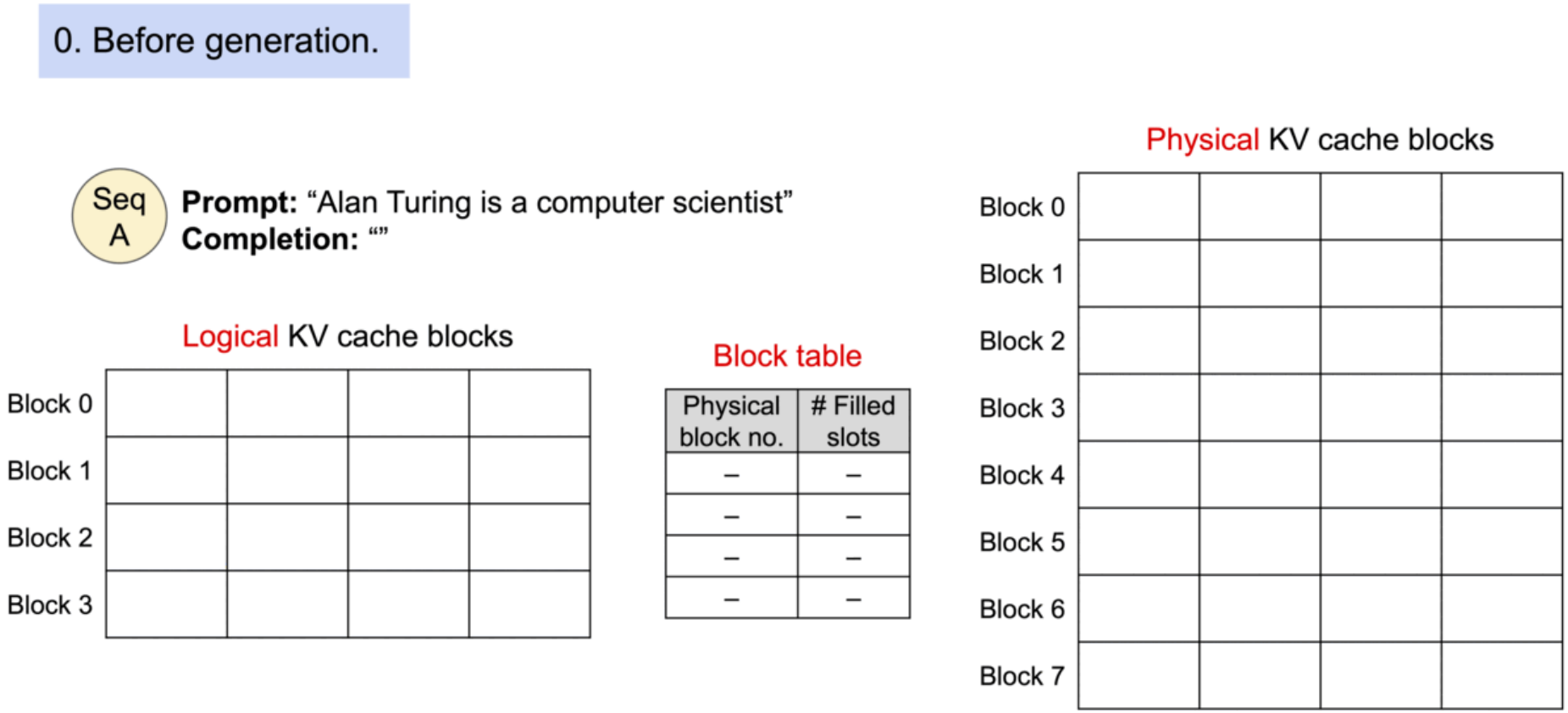
- Size is unpredictable, and always grows
- Laying them out contiguously in memory is a bad idea

How can we manage them all efficiently?

- Idea: Split them up into chunks, manage on chunk level
- Now Decode has to find the right chunks
- Also need to allocate new chunks on the fly
- And manage them once they exist

This is a familiar problem…

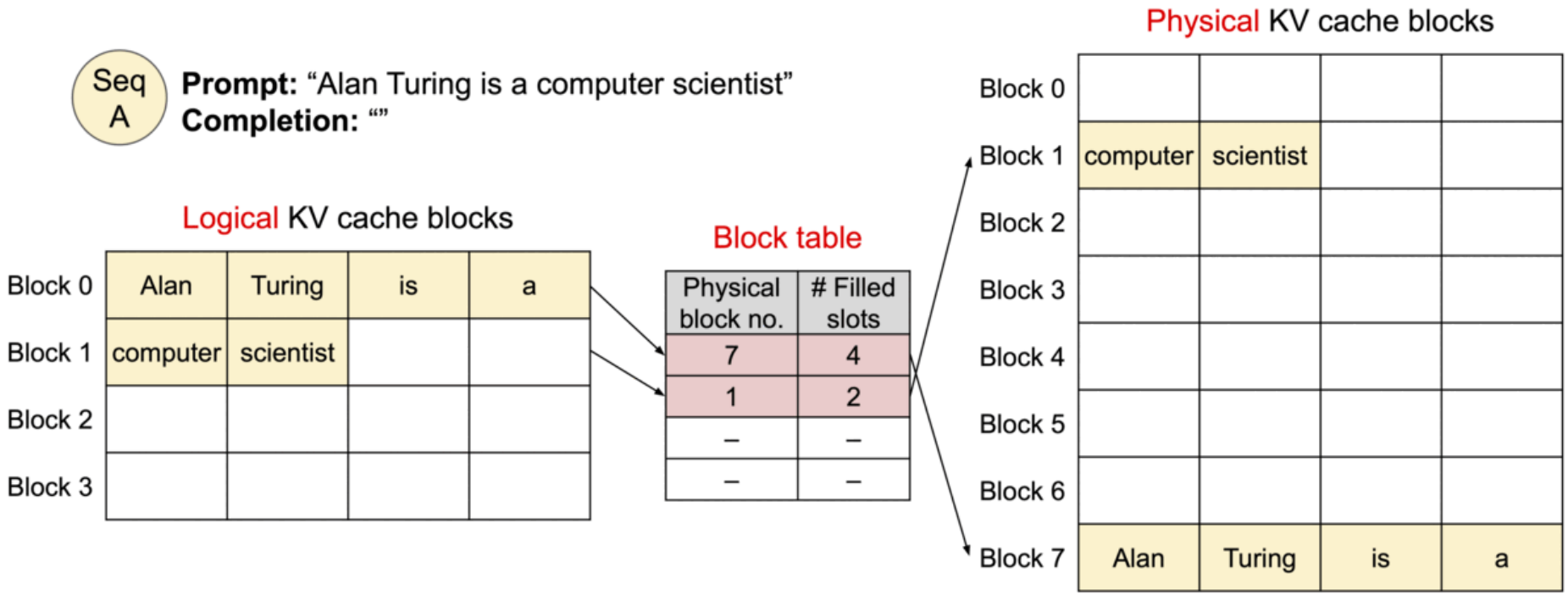
PagedAttention: Memory Management



Source: https://vllm.ai/blog/vllm

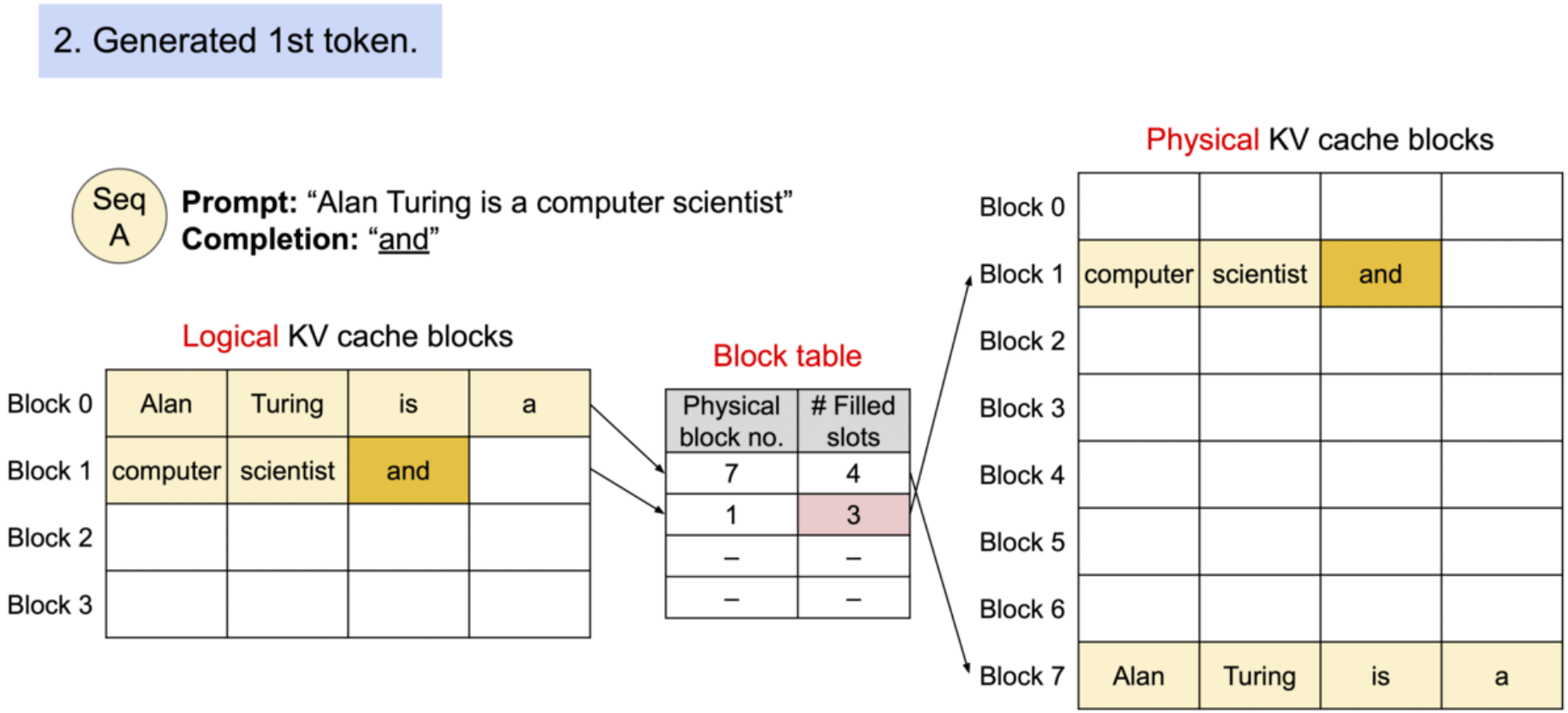
PagedAttention: Memory Management

1. Allocate space and store the prompt's KV cache.



Source: https://vllm.ai/blog/vllm

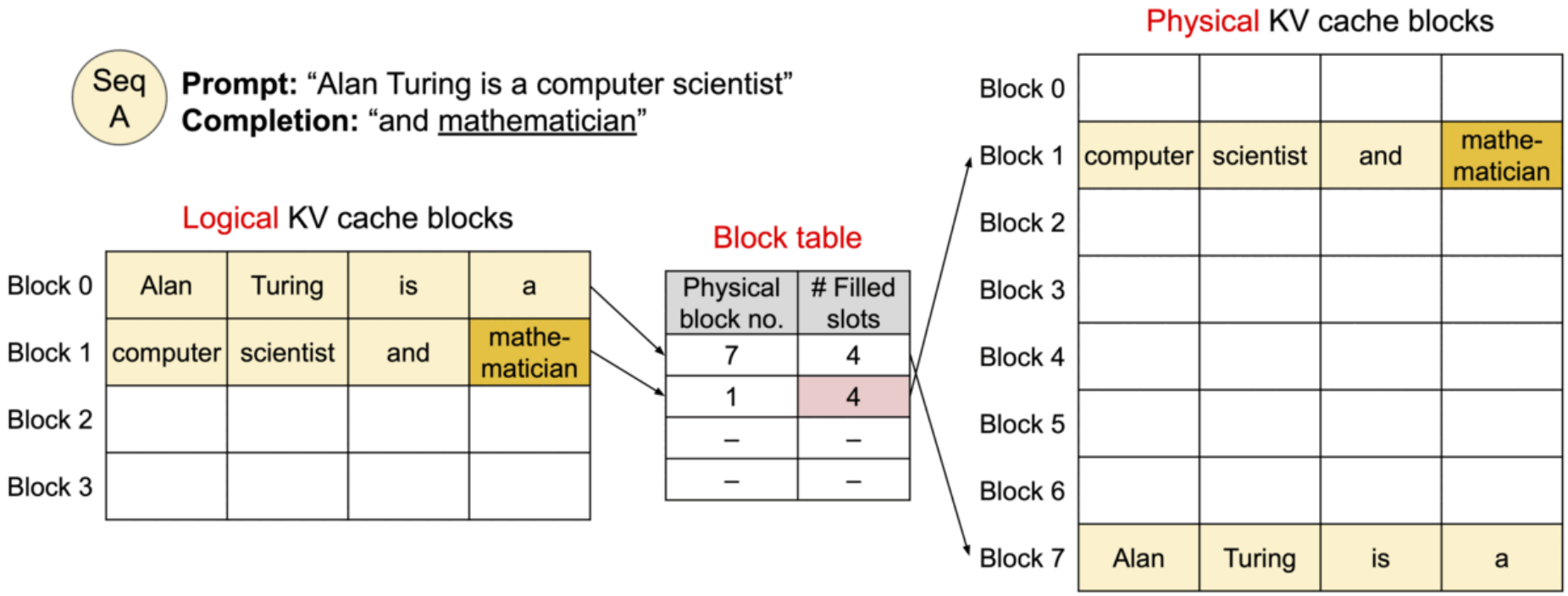
PagedAttention: Memory Management



Source: https://vllm.ai/blog/vllm

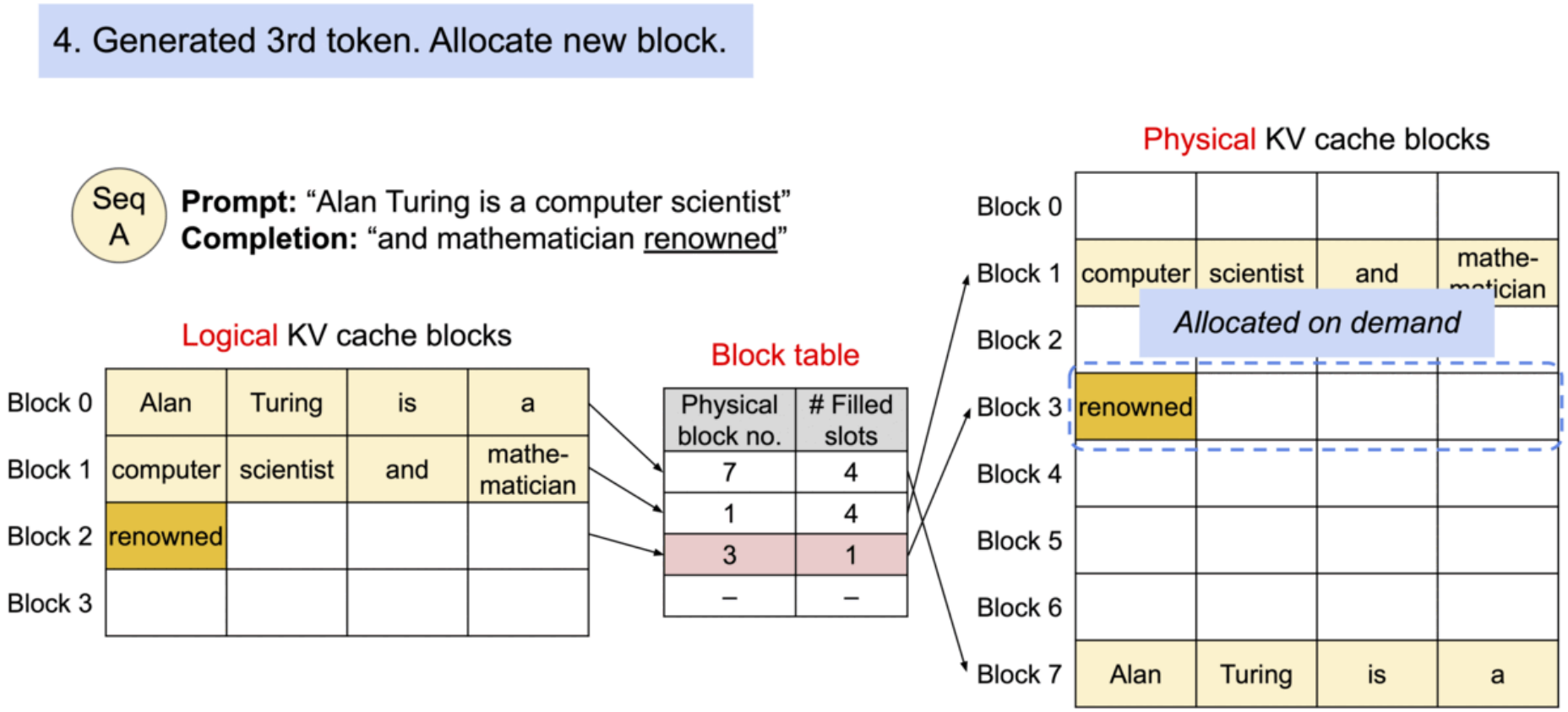
PagedAttention: Memory Management

3. Generated 2nd token.



Source: https://vllm.ai/blog/vllm

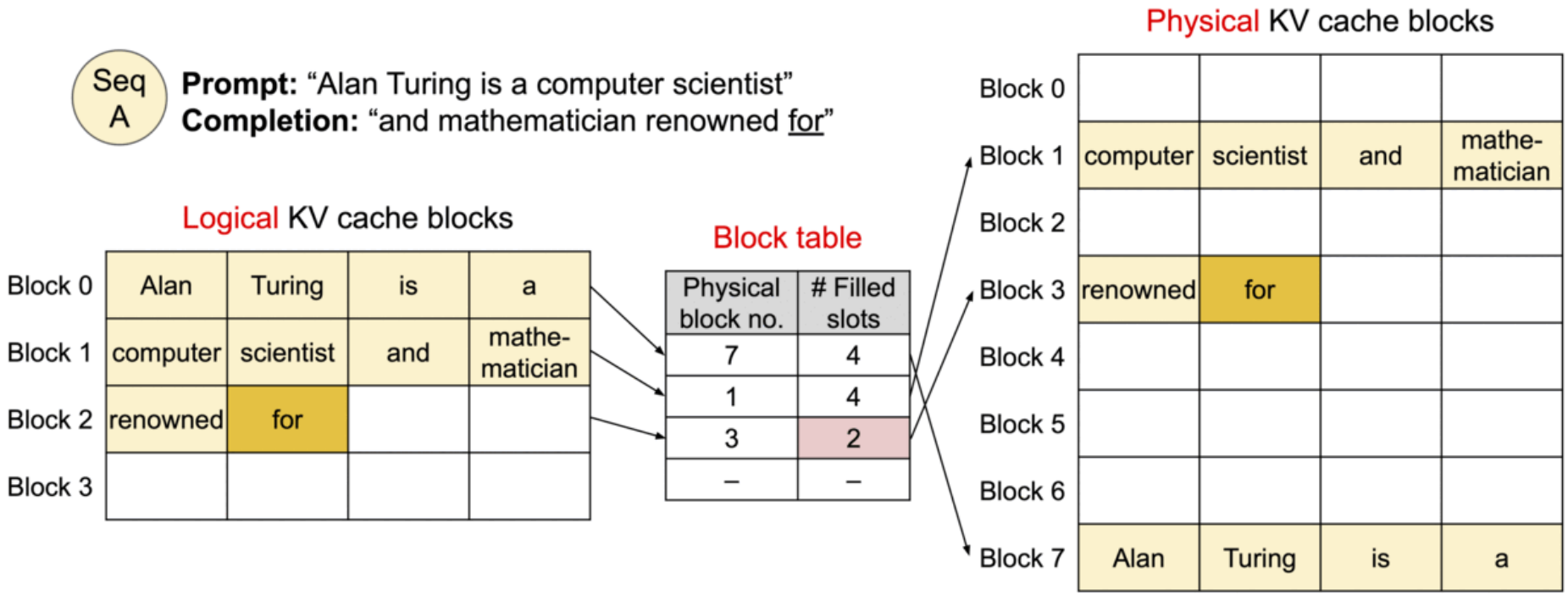
PagedAttention: Memory Management



Source: https://vllm.ai/blog/vllm

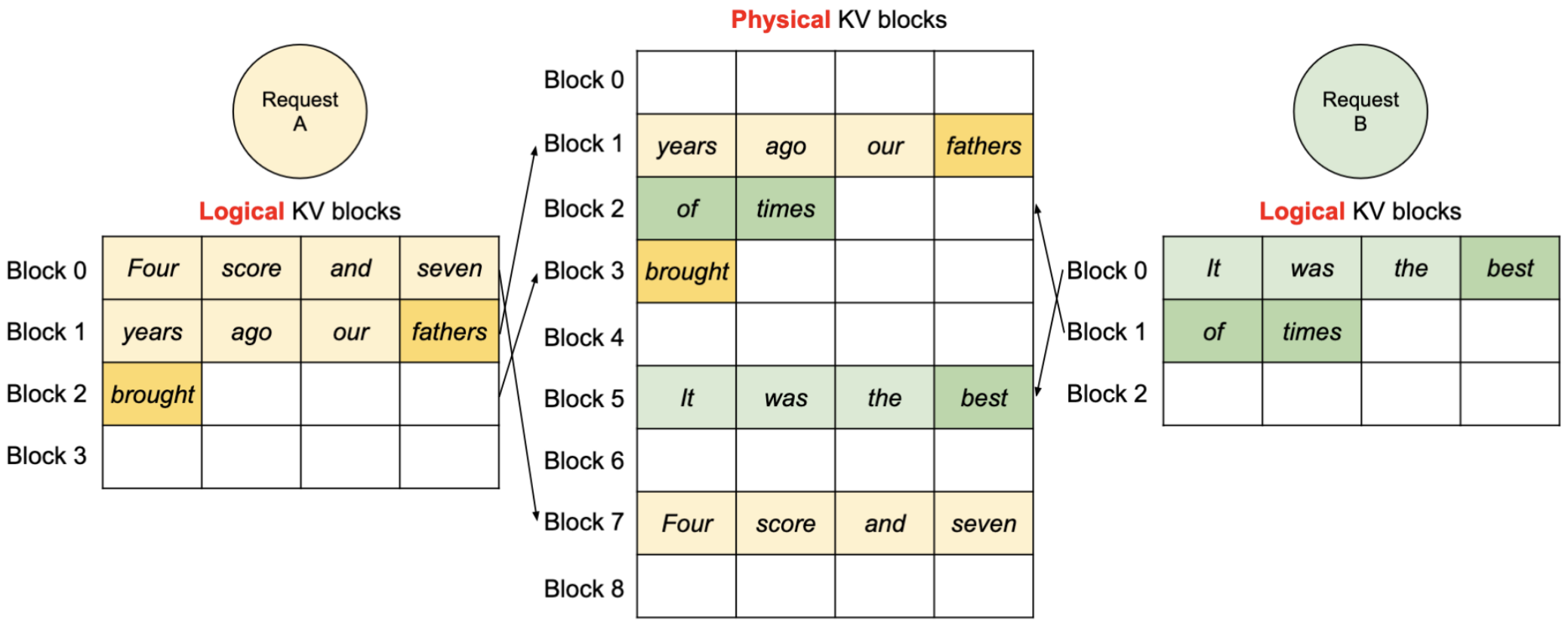
PagedAttention: Memory Management

5. Generated 4th token.



Source: https://vllm.ai/blog/vllm

PagedAttention: Efficient Memory Usage



Source: https://arxiv.org/pdf/2309.06180

PagedAttention: Next Steps

PagedAttention doesn’t fully solve our problem, but it enables a whole bunch of other solutions

Operating with blocks rather than full KVCaches is useful:

- Can page blocks between CPU and GPU
- Share blocks for common parts of prompt
- Compress context to evict old blocks

Without PagedAttention, none of the above would be possible!

Challenge 3: Access Control

LLMs are trained on enormous amount of data, designed to be extremely generic

Some tasks should not be doable with LLM:

- “How do I build a bomb”
- Common term: “AI Alignment”

Challenge: Stop the model from giving up answers that are “bad”

Solution: Guardrails

Put checks in place to prevent the model from returning that to the user if it's "bad"

- Reinforcement during training to incentivize certain kinds of responses
- Input sanitization to avoid responding to obvious attacks
- Output sanitization to catch harmful responses

Sanitization techniques range from simple (Grep) to complex (Llama-Guard 3)

LLMs Accelerate Attack and Defense!

Every day, new attacks are successfully mounted using LLMs

- Yesterday: Lovable (allegedly)
- Sunday: Vercel
- Saturday: FortiSandbox (CVE-2026-39808)

LLMs are also used to fix vulnerabilities daily ☺

This is an area with huge potential for major research impact!

Aside: Specialized Hardware

GPUs are fantastic at the pre-fill step:

- Super parallelizable
- Prompts are large, big matrices use all the GPU cores

Decode doesn't fit this pattern, and there's an emerging concern over the amount of decode that's happening

- Training new models has diminishing returns
- Inference *will* dominate in the future, and decode dominates inference

Probably need new hardware!

ML Problems Benefit From Systems Solutions

Systems ideas reusable in ML context, with modification

Looked at some today, there are many more!

- Speculative execution -> Speculative decoding
- Kernel Bypass -> Used all over the place
- Copy-on-write -> Active area of research