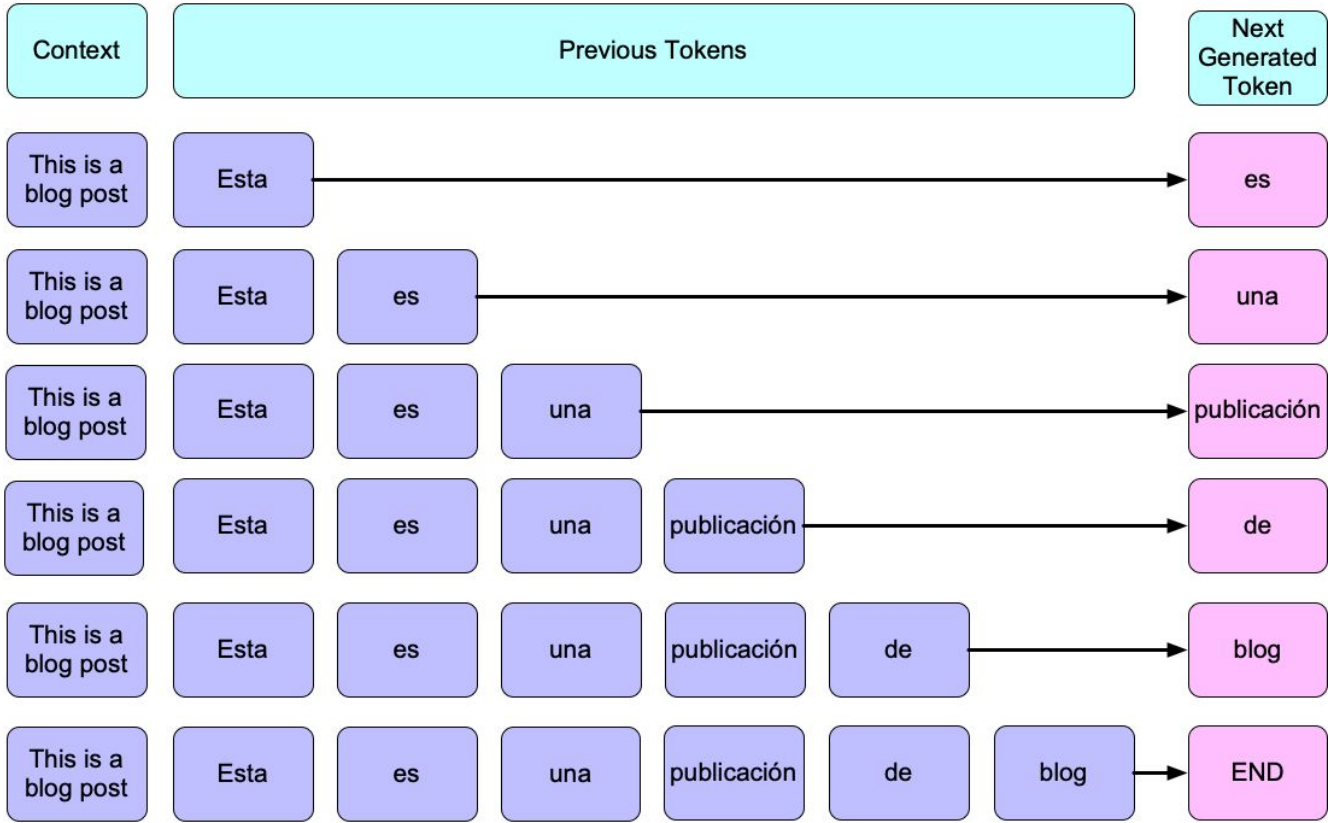
Memory Management in LLM Inference Serving

Hendrik Mayer, Yeda Song, Yuchen Xia

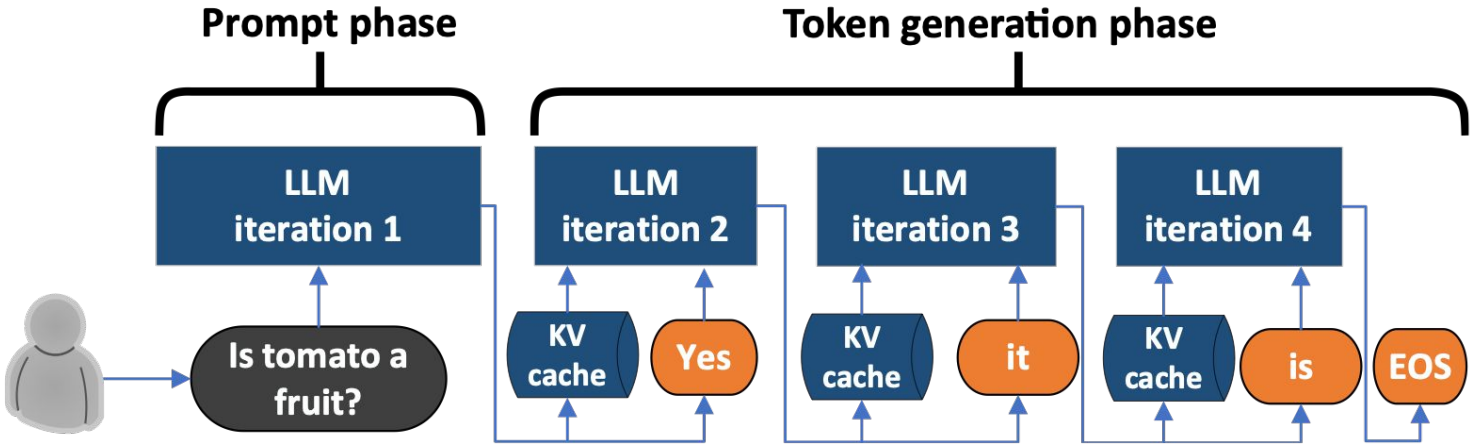
LLM Inference

- Inference refers to the process of decoding an output sequence from a prompt using a model
- The LLM models the probability of a list of tokens and uses these probabilities to generate next tokens



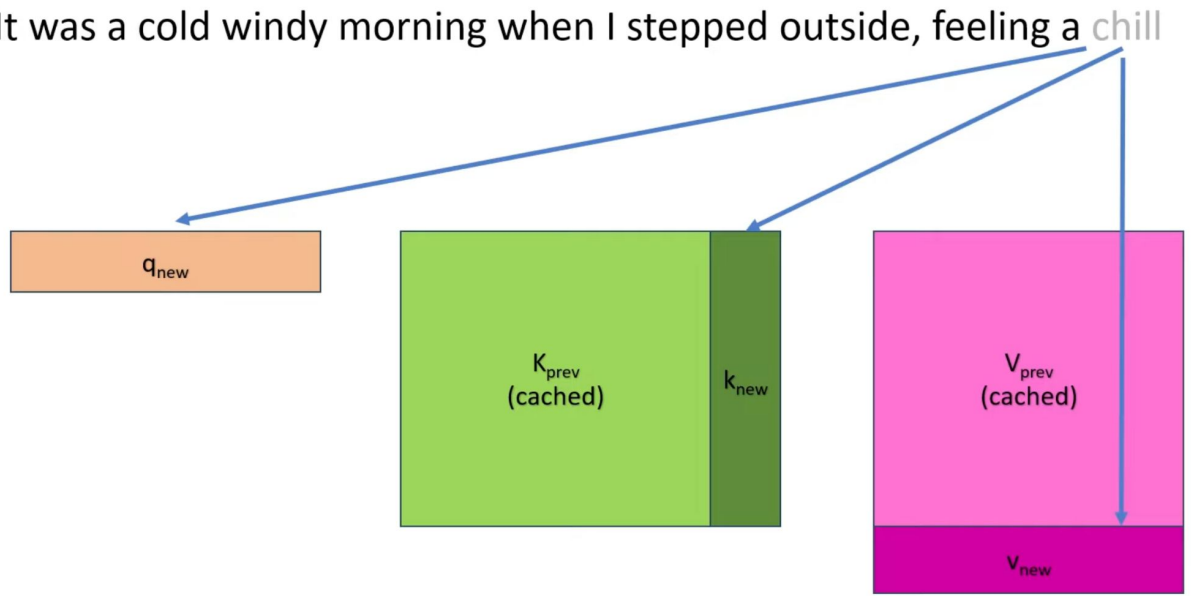
KV Cache

- Next token generation depends on all previous tokens' key-value vectors
- These vectors is stored in a KV cache to avoid repeated computation
  - Prompt phase (a.k.a. *prefill*)
  - Autoregressive generation phase (a.k.a. *decoding*)



KV Cache: Challenge

- KV cache grows **dynamically** by one token with each iteration, making pre-allocation of memory challenging and inefficient



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

Toward Efficient LLM Serving

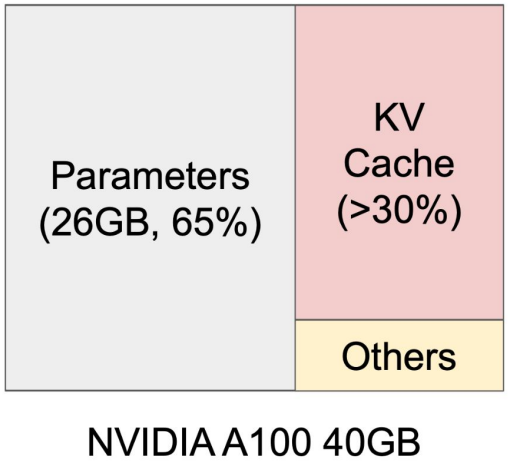
- Increasing throughput of LLM serving systems is important because
  - GPU computation power is often underutilized
  - Time spent processing LLM queries is expensive

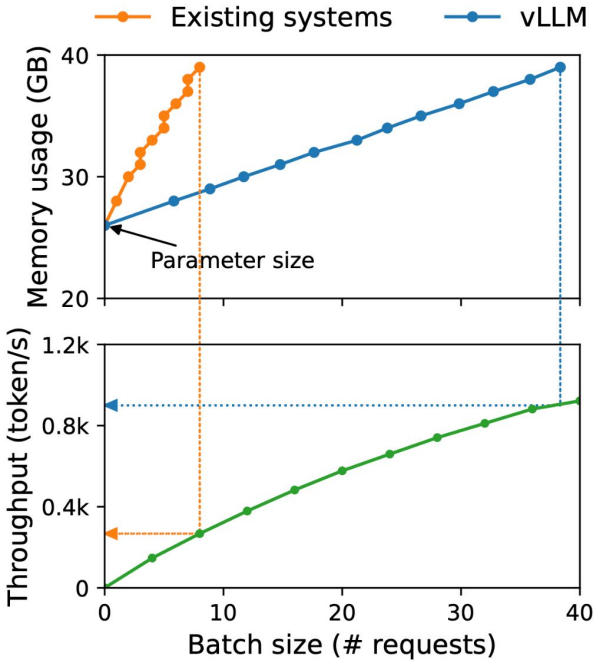
- Throughput can be increased by batching requests together

- PagedAttention provides efficient KV-cache memory management for batched requests

Key Insight

- Efficient **KV Cache** allows bigger **batch size**, which translates to higher **throughput**

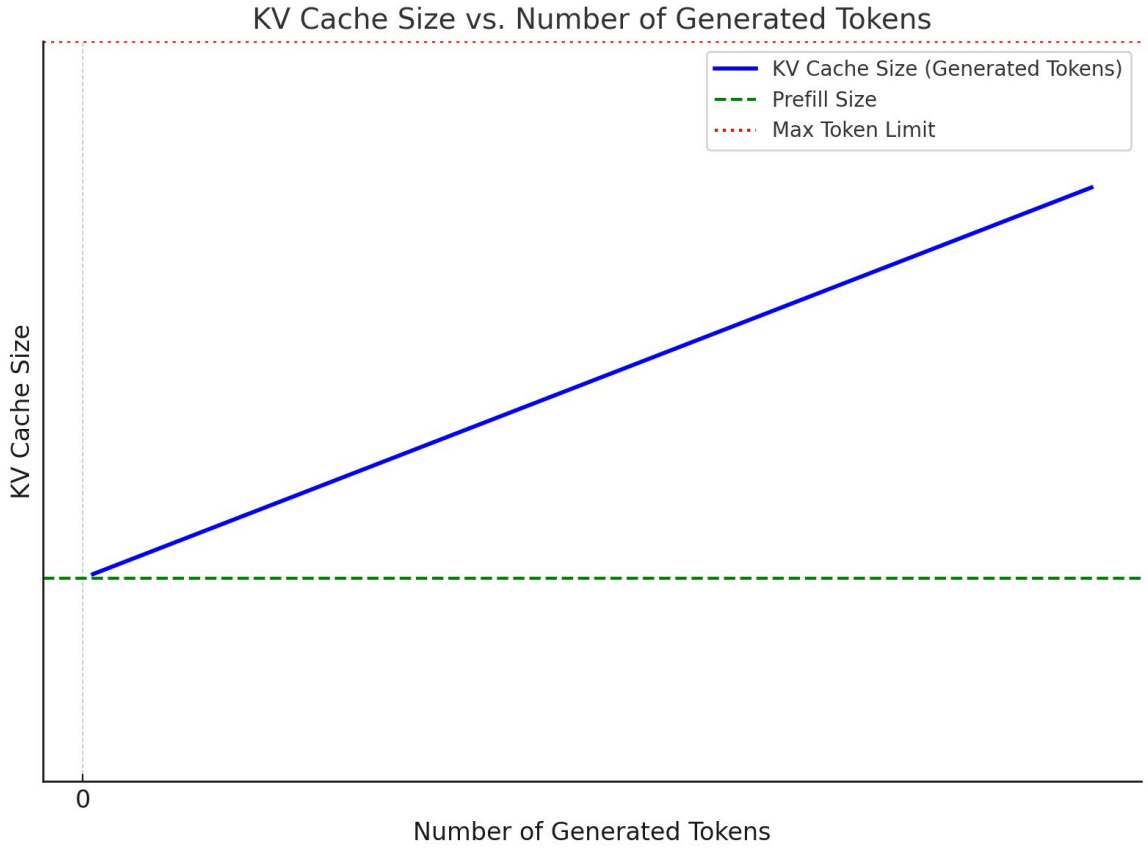




Problems with Existing Systems

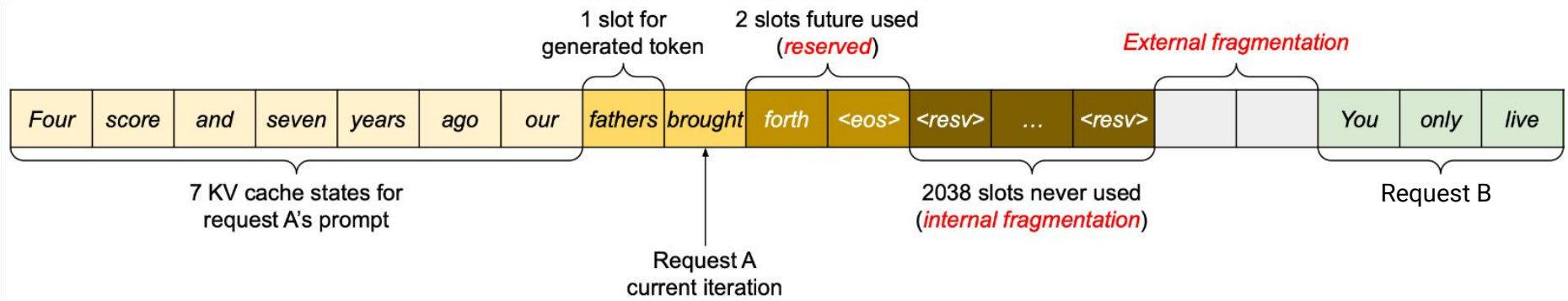
- KV cache dynamically grows and shrinks as the model generates next tokens

- Existing LLM serving systems store the KV cache for a request in contiguous memory
  - Internal and external memory fragmentation
  - Unable to exploit memory sharing



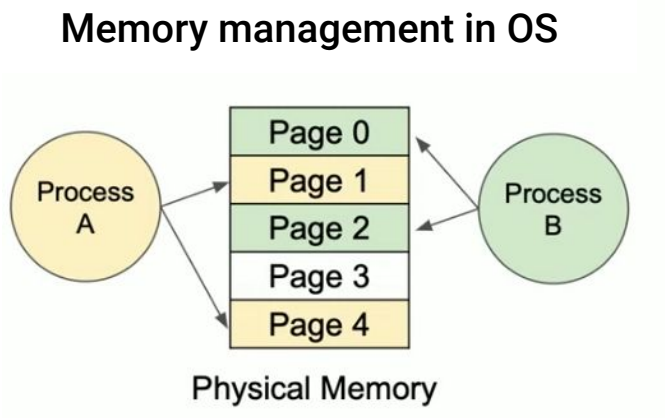
Limitations of Existing Systems Example

- Wastes of Memory
  - Reserved Slots: currently unused memory
  - Internal Fragmentation: memory over-allocation for a request
  - External Fragmentation: scattered free slots, no large contiguous memory



Solution: PagedAttention

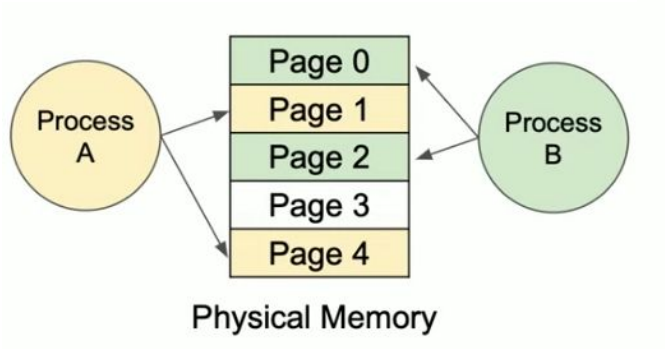
- Inspired by **virtual memory** with **paging**



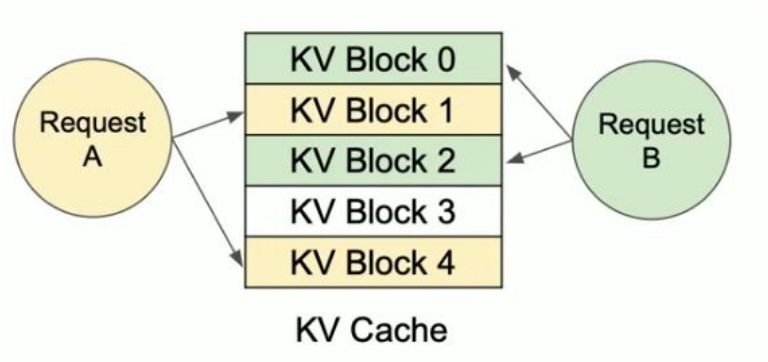
Solution: PagedAttention

- Inspired by **virtual memory** with **paging**

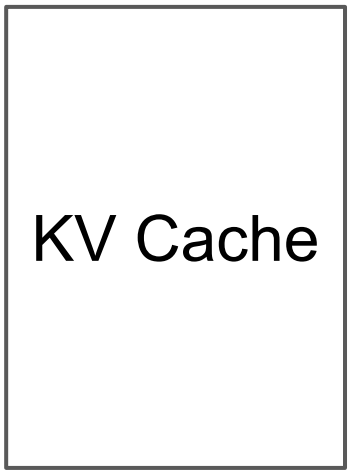
Memory management in OS



Memory management via PagedAttention

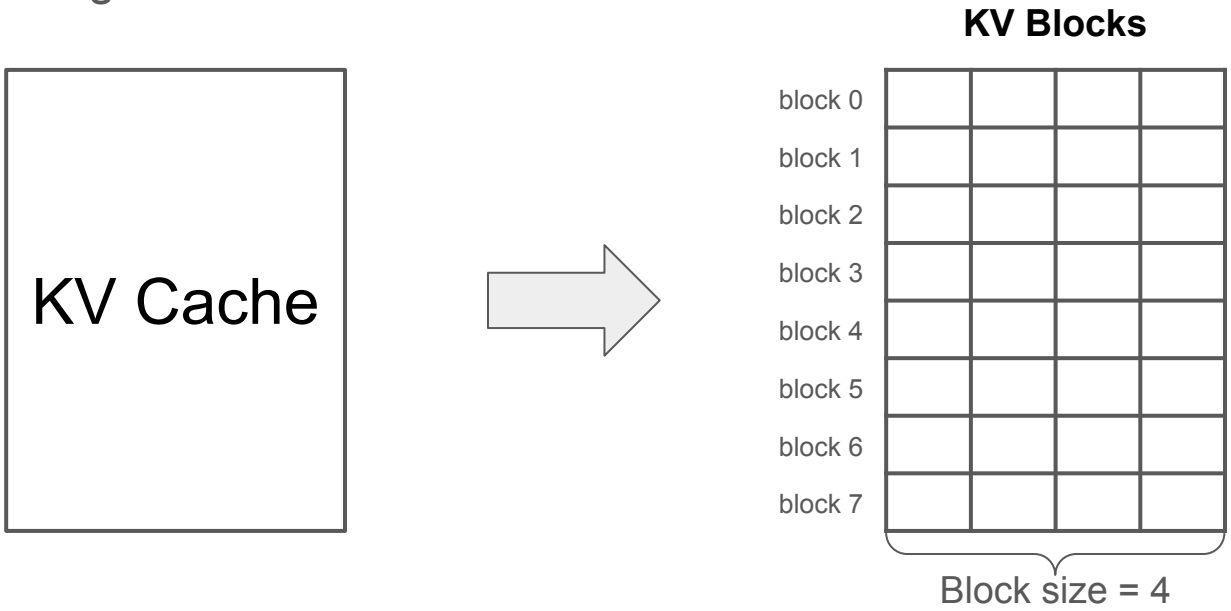


KV Blocks



KV Blocks

- A **fixed-size** contiguous chunk of memory that can store KV cache values from left to right



0. Before generation.

Seq A

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

Logical KV cache blocks

Block 0				
Block 1				
Block 2				
Block 3				

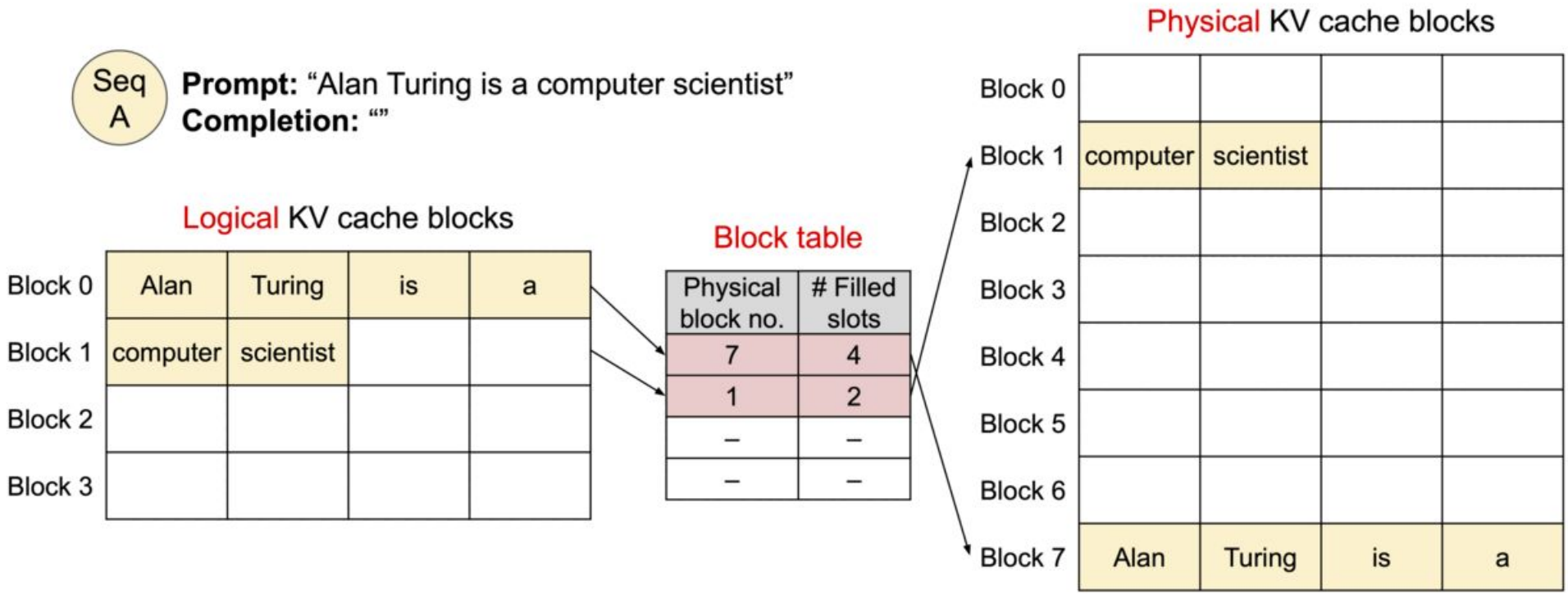
Block table

Physical block no.	# Filled slots
–	–
–	–
–	–
–	–

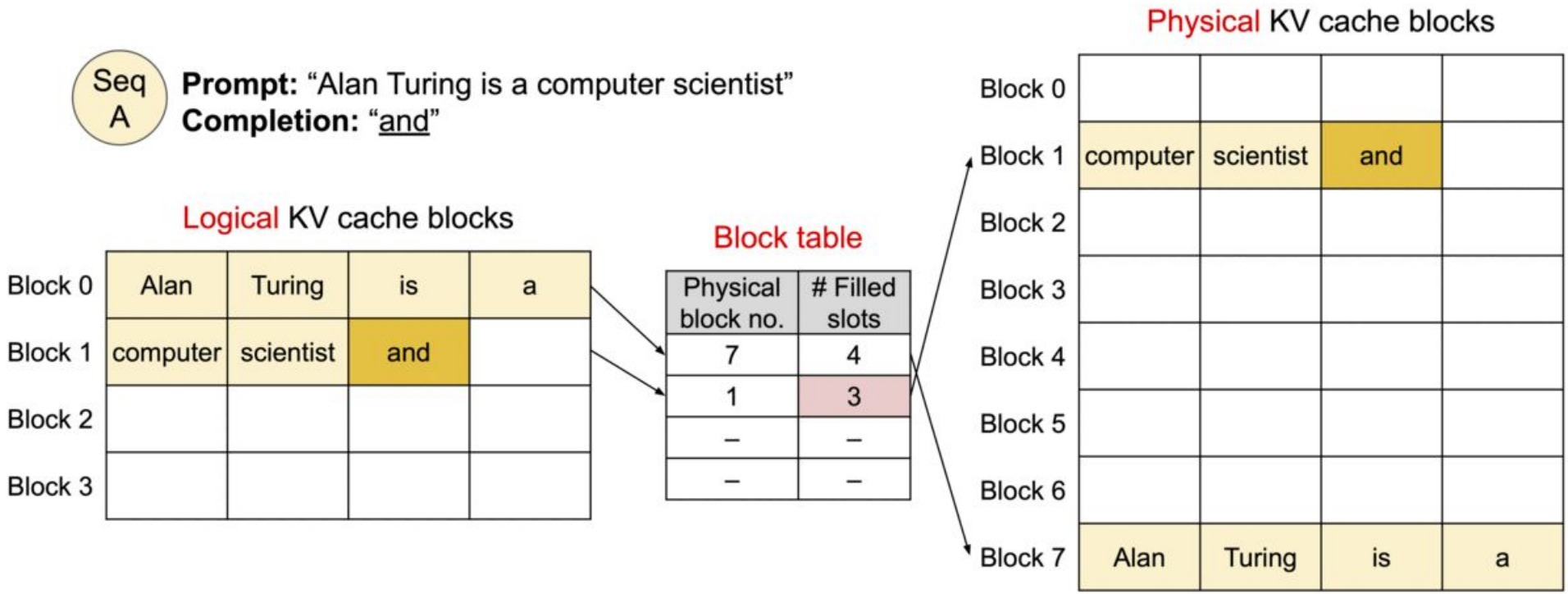
Physical KV cache blocks

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

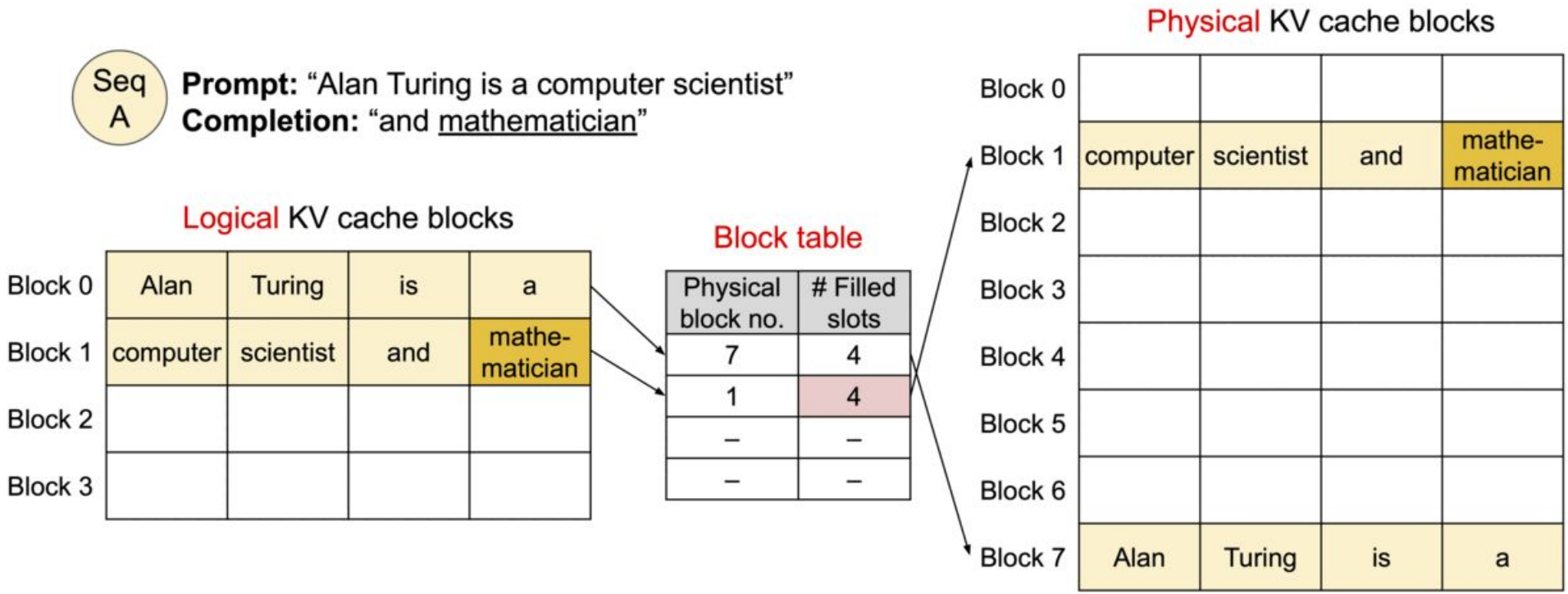
1. Allocate space and store the prompt's KV cache.



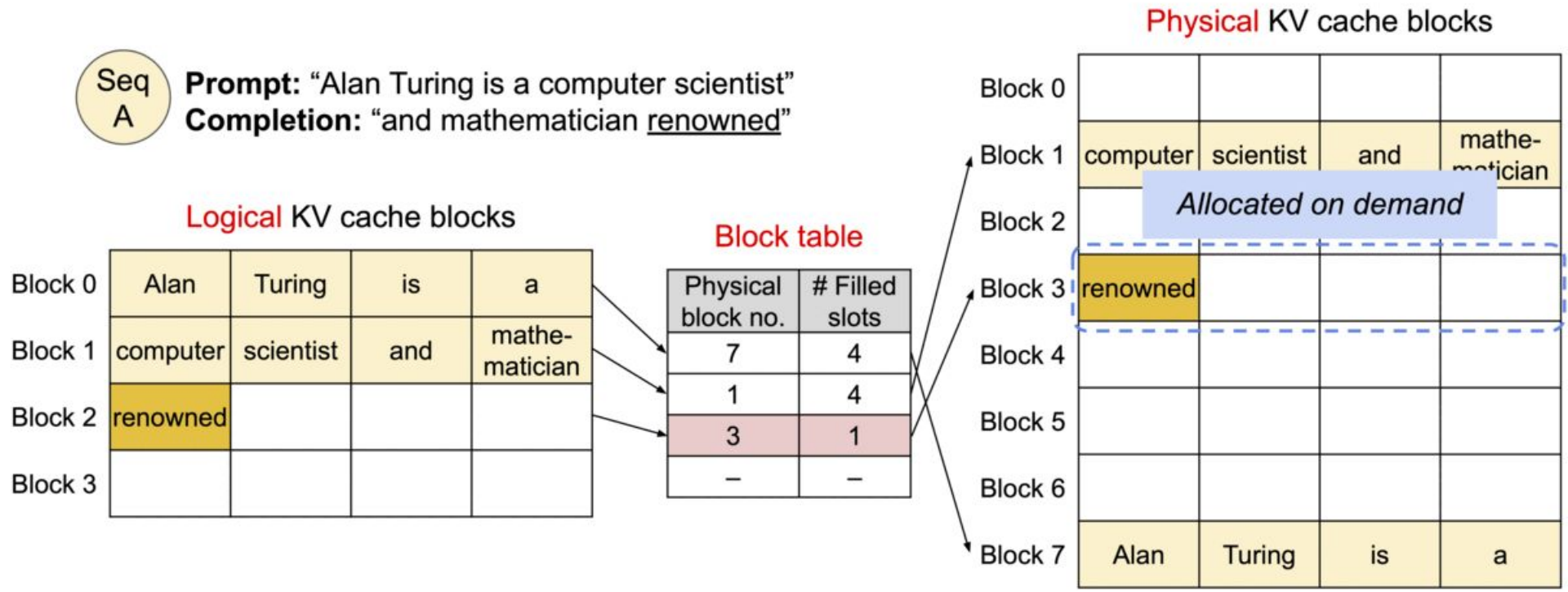
2. Generated 1st token.



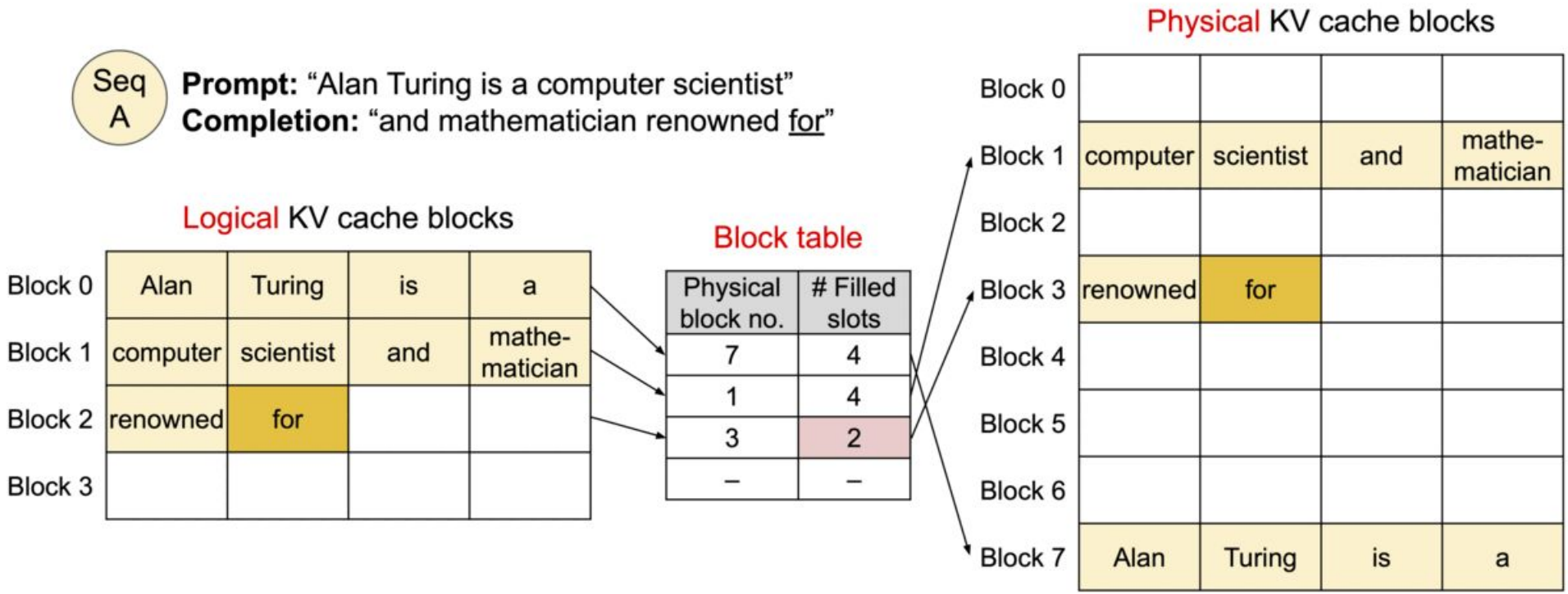
3. Generated 2nd token.



4. Generated 3rd token. Allocate new block.



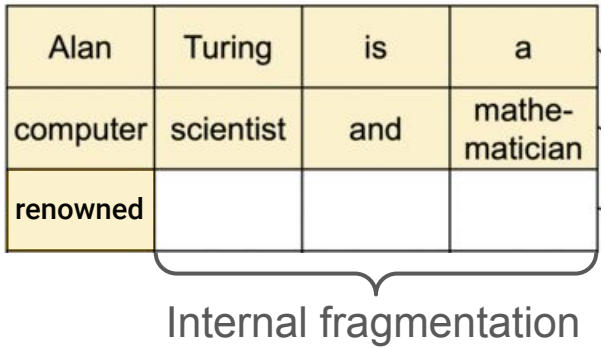
5. Generated 4th token.



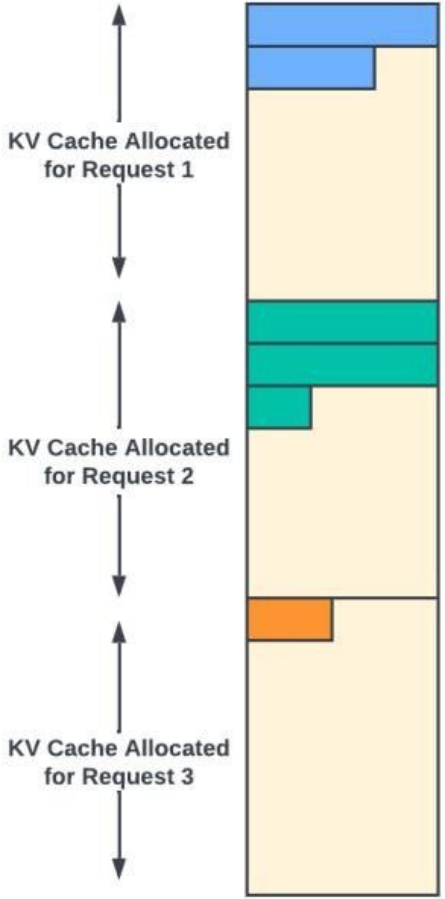
PagedAttention Handles Fragmentation

- Internal memory fragmentation **only occurs at the last block of the sequence**
  - # wasted tokens < block size (per sequence)

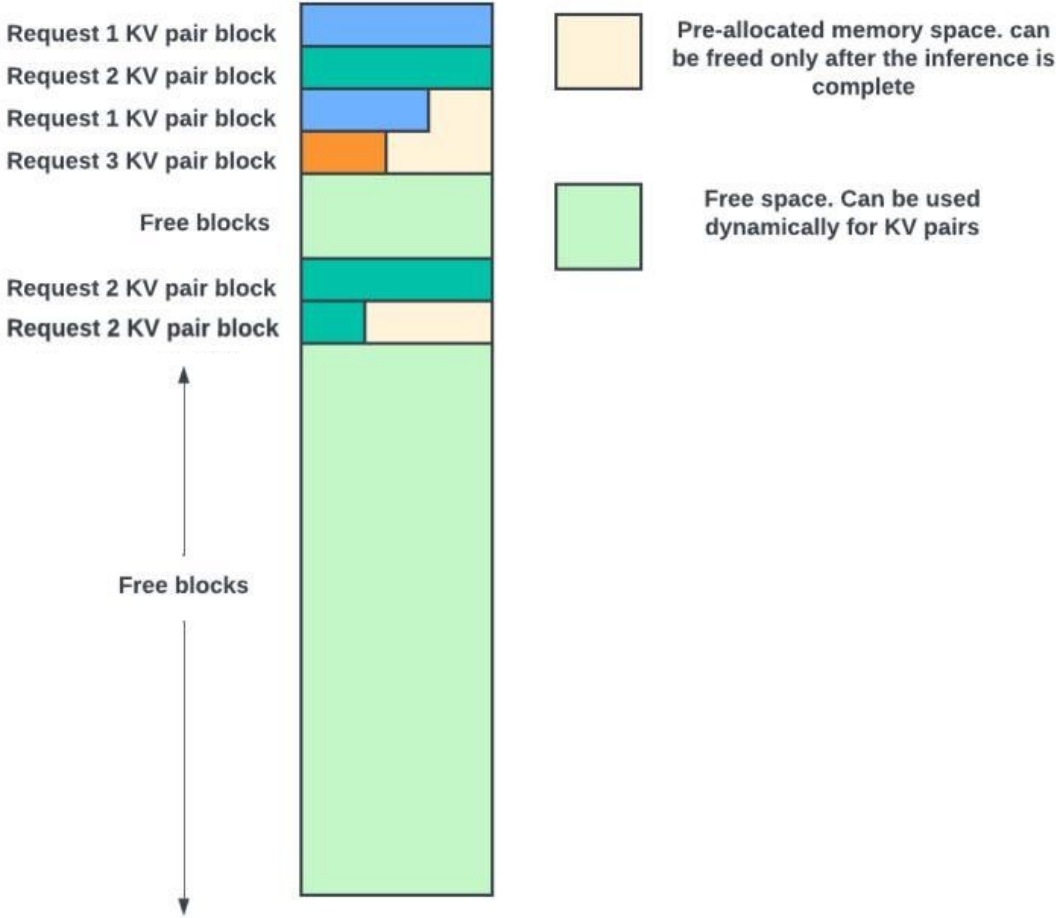
- **No** external memory fragmentation
  - Memory only becomes available in the same block size



Previous Systems

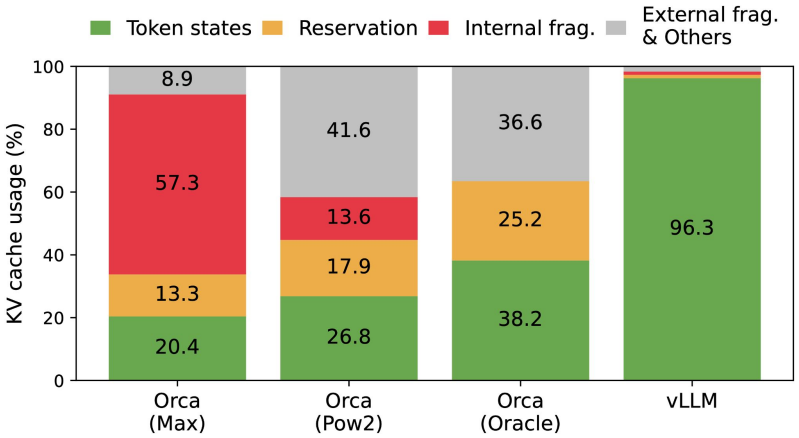


PagedAttention Systems

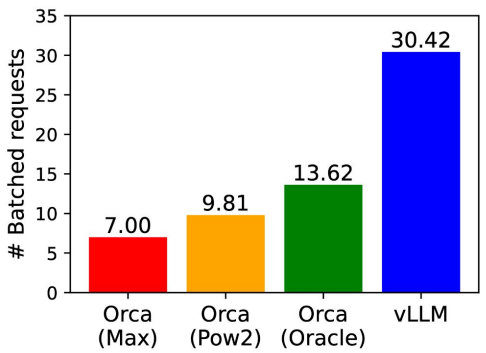


Efficient Memory Usage

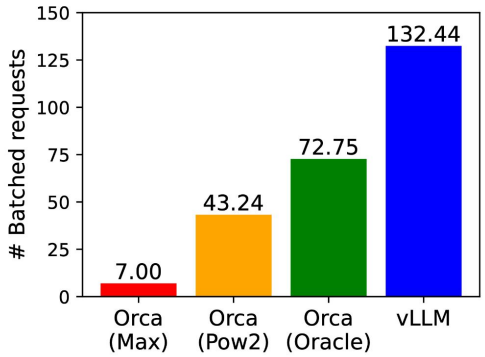
- Achieves near-zero waste of KV Cache



- Bigger batch sizes under same conditions



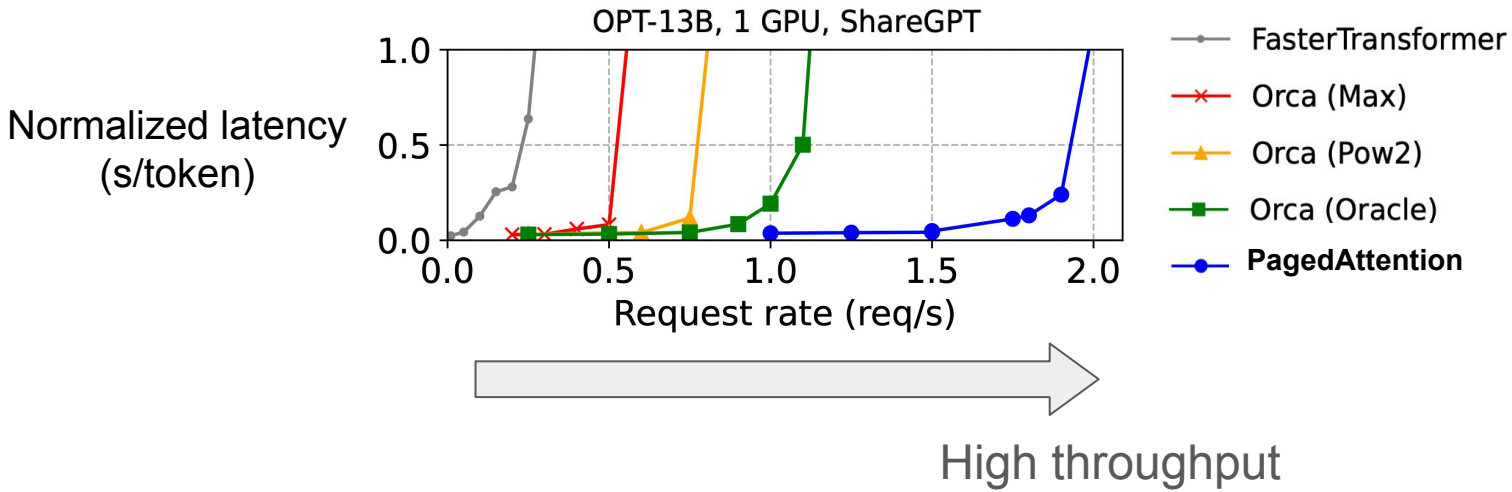
(a) ShareGPT



(b) Alpaca

High Throughput

$$\textit{Normalized latency}^{1} = \text{Avg}\left(\frac{\text{end-to-end latency}}{\text{output length}}\right)$$



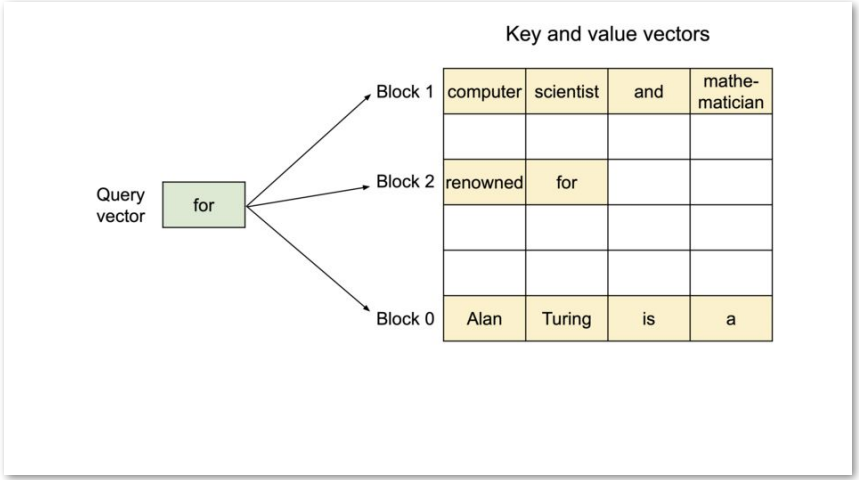
[1] Used in Orca (Yu et al.)

vAttention: Dynamic Memory Management for Serving LLMs without PagedAttention

Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, Ashish Panwar

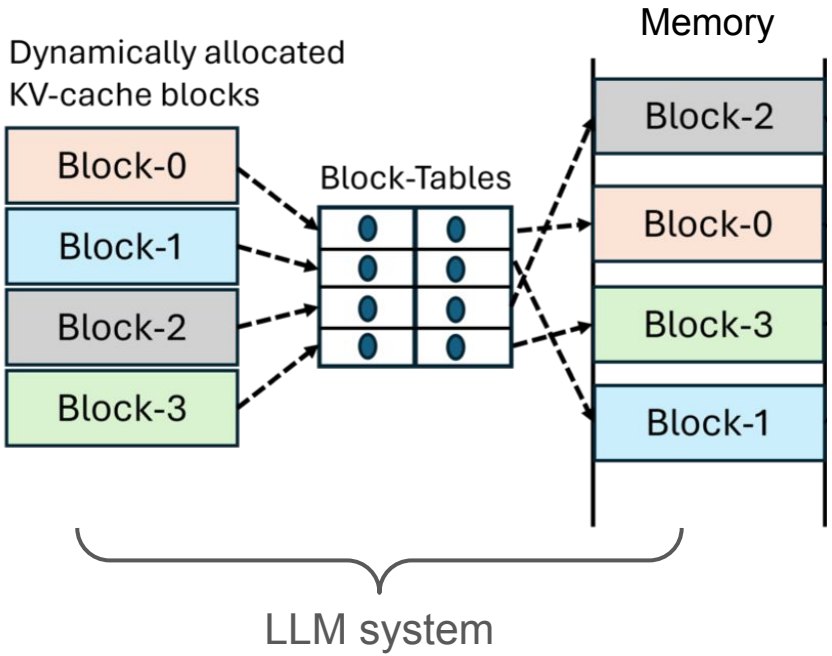
Performance Overhead of PagedAttention

- PagedAttention showed 20–28% **slower attention kernel** compared to non-paged systems
- Due to **non-contiguous virtual memory**



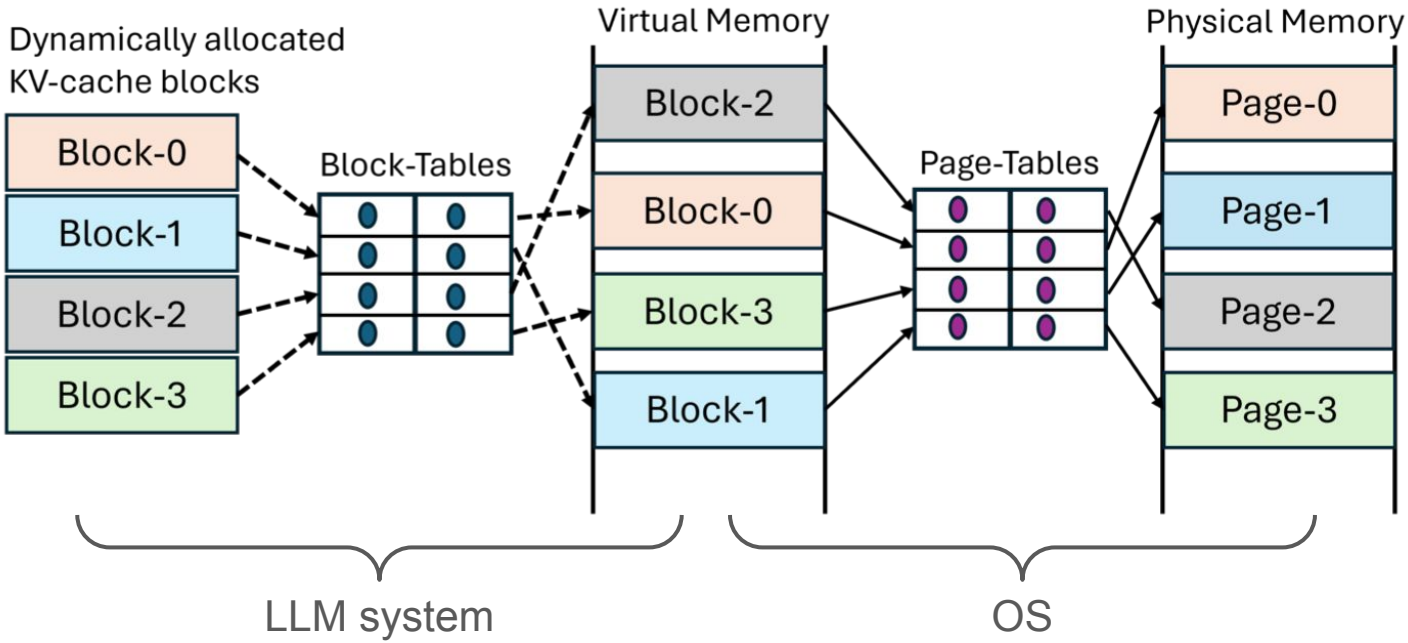
Decoding with PagedAttention

Design Redundancy of PagedAttention



Design Redundancy of PagedAttention

- **Two layers of memory management**, one in user space and one in OS kernel space



Pros and Cons of PagedAttention

(+)

- **Dynamic KV-cache memory management** for high throughput

Paged Attention

(-)

- Performance overhead due to **non-contiguous virtual memory**
- **Re-implementation** of OS memory management

Pros and Cons of PagedAttention

(+)

- **Dynamic KV-cache memory management** for high throughput

Paged Attention

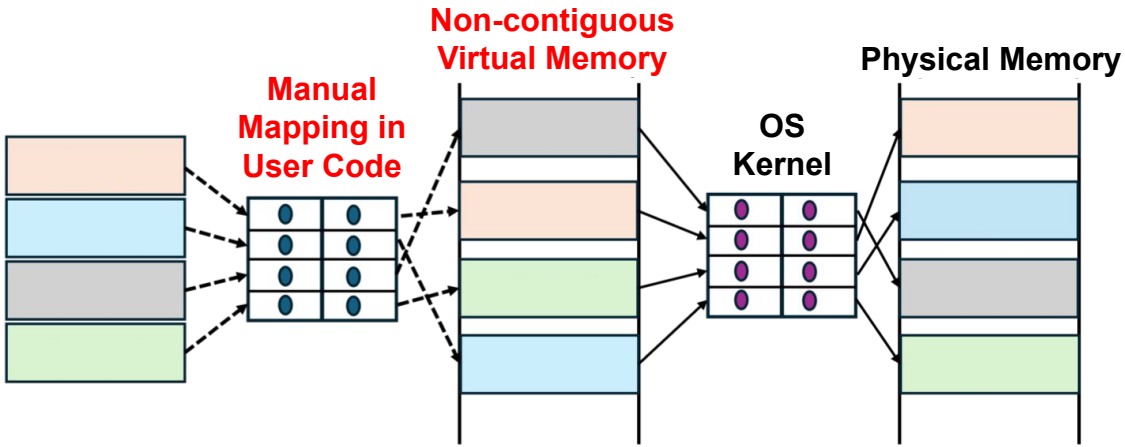
(-)

- Performance overhead due to **non-contiguous virtual memory**
- **Re-implementation** of OS memory management

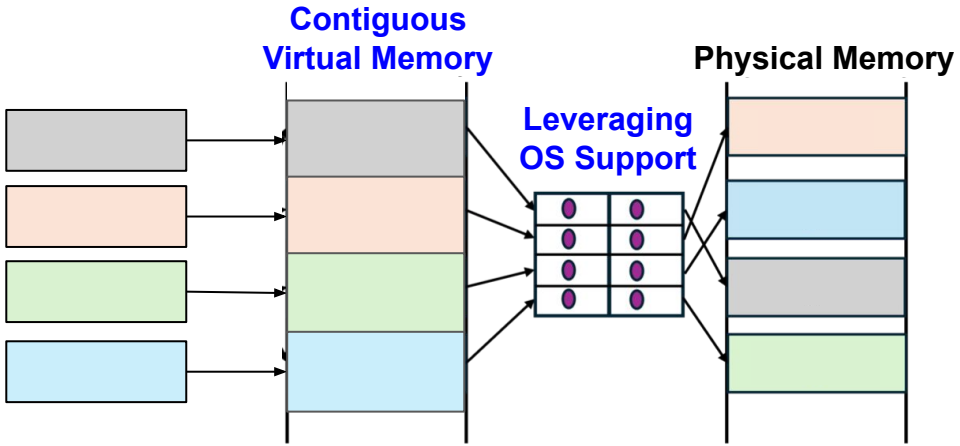
vAttention: Let's achieve **(+)** without **(-)**!

Key Differences

PagedAttention



vAttention



Design and Implementation

Number of virtual memory buffers: 2 × N

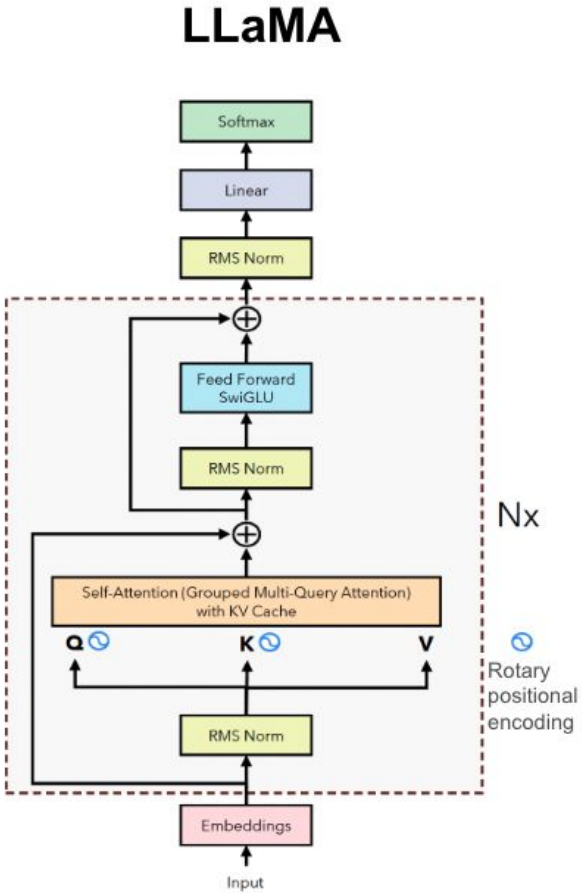
K cache and V cache

Number of layers

Maximum Size of virtual memory buffers: $BS = B \times L \times S$

B: maximum batch size
L: maximum context length

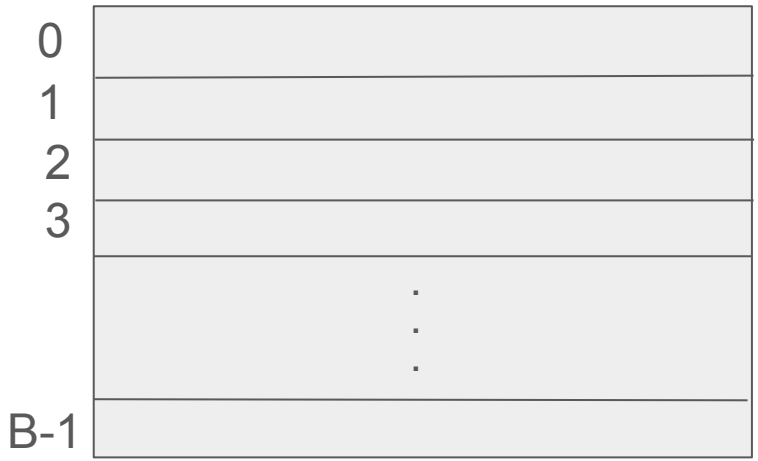
S = H × D × P
H: number of KV heads
D: dimension of each KV head
P: number of bytes



Design and Implementation

APIs	Description
`init`	Initializes vAttention with model parameters. arguments: N', B, L, H', P, `page_size`. return value: a list of KV-cache tensors.
`alloc_reqid`	Allocates an unused `reqId` and marks it active arguments: None return value: an integer `reqId`
`free_reqid`	Frees a `reqId` and marks it inactive arguments: an integer `reqId` return value: None
`step`	Ensures physical memory pages are mapped arguments: an array of size B containing sequence lengths of each `reqId` return value: 0 (success), -1 (failure).

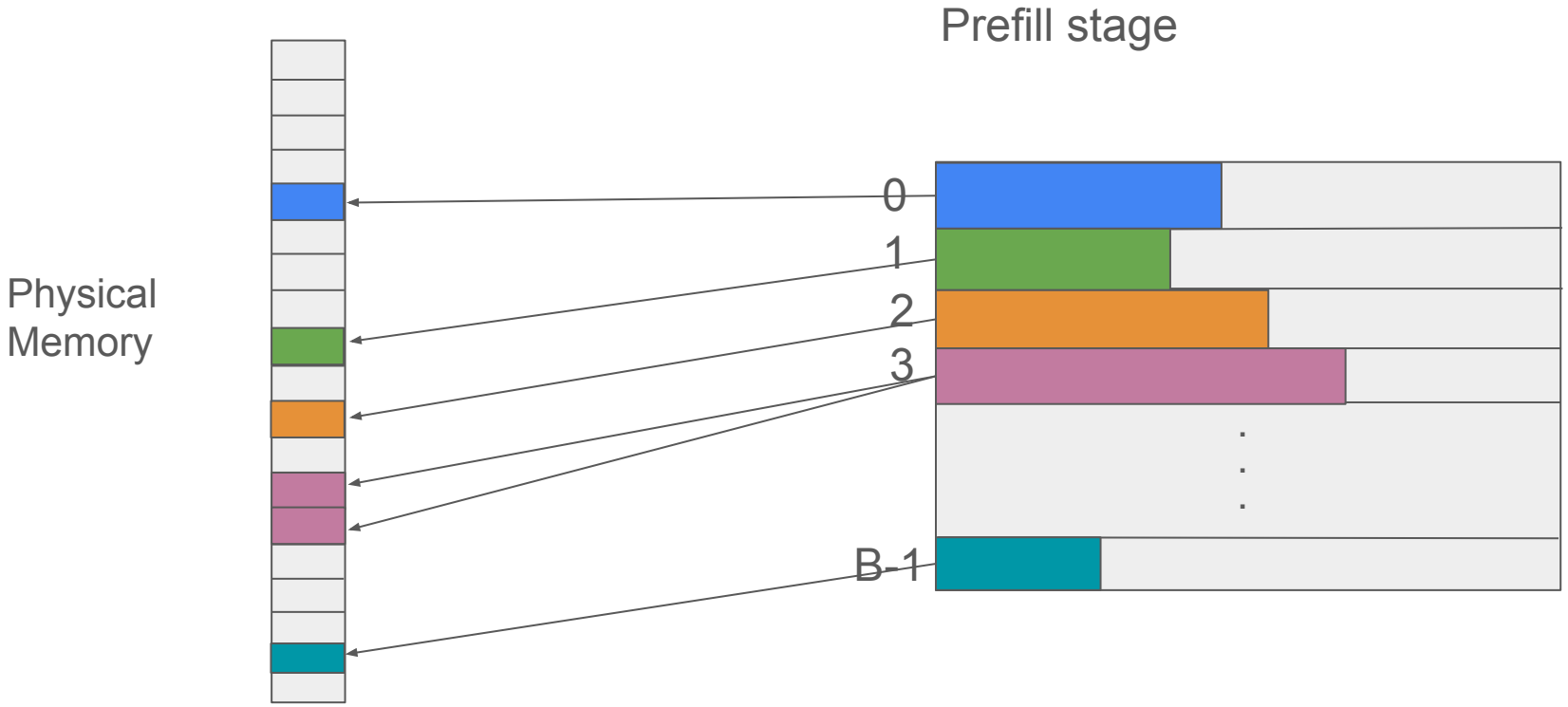
KV tensor : K-cache (or V-cache) of a layer for the maximum batch size



Different requests occupy different non-overlapping sub-regions

Sub-regions size: the maximum K-cache(or V-cache) size of a request. H * L * D * P

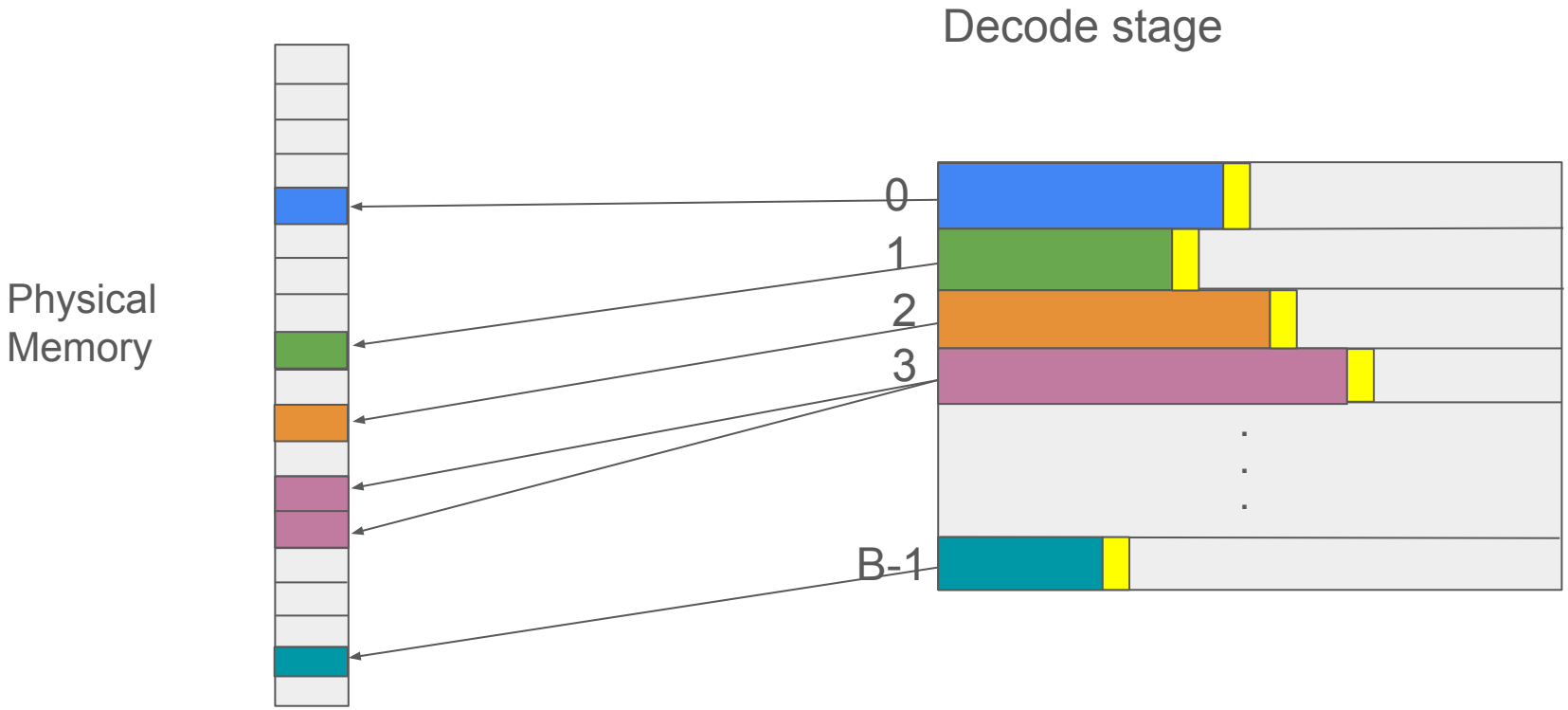
Design and Implementation



Different requests occupy different non-overlapping sub-regions

Sub-regions size: the maximum K-cache(or V-cache) size of a request. H * L * D * P

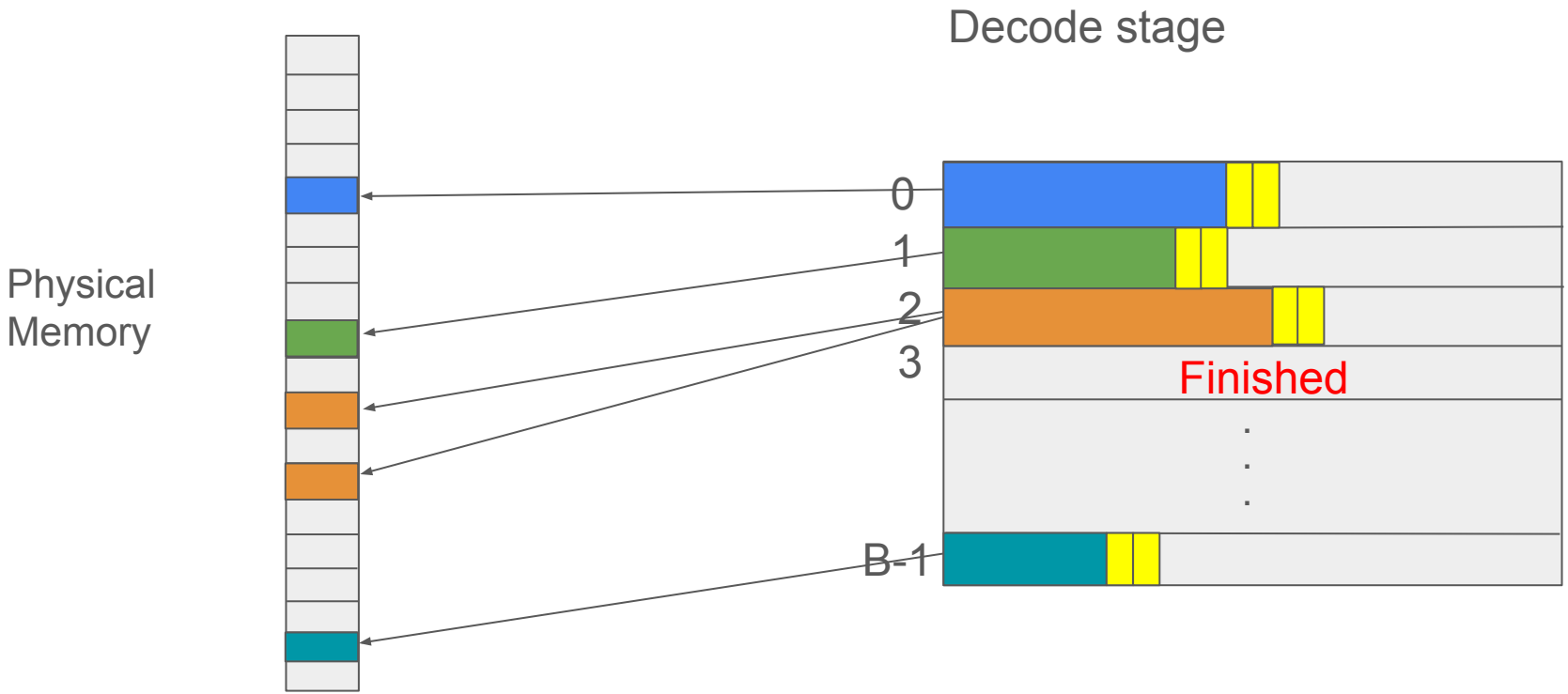
Design and Implementation



Different requests occupy different non-overlapping sub-regions

Sub-regions size: the maximum K-cache(or V-cache) size of a request. H * L * D * P

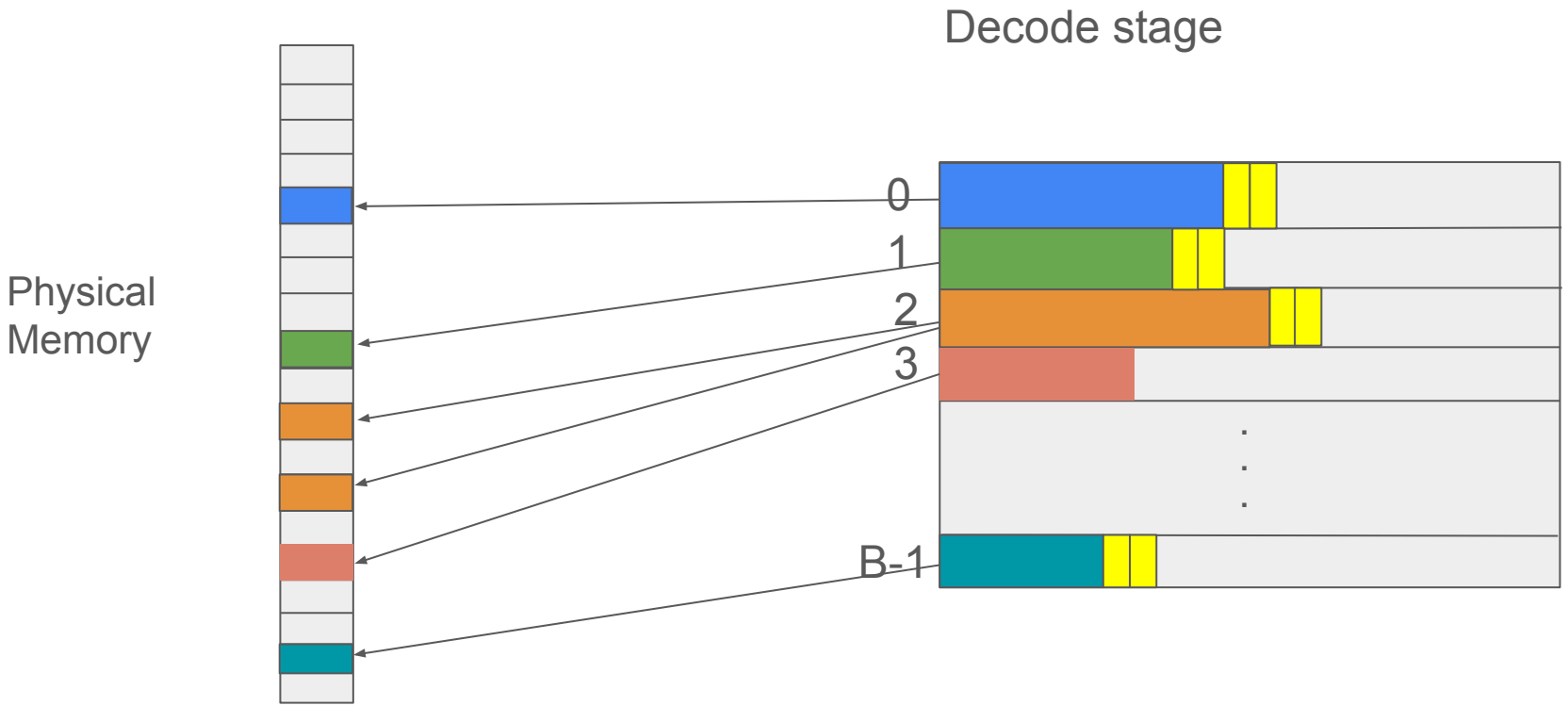
Design and Implementation



Different requests occupy different non-overlapping sub-regions

Sub-regions size: the maximum K-cache(or V-cache) size of a request. H * L * D * P

Design and Implementation



Different requests occupy different non-overlapping sub-regions

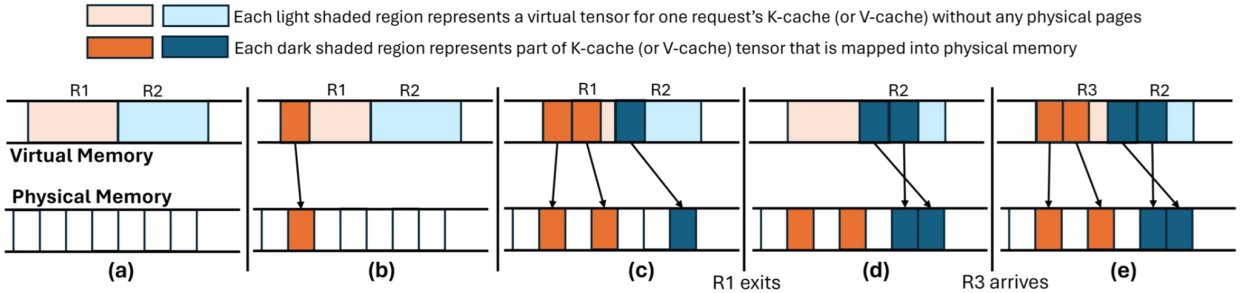
Sub-regions size: the maximum K-cache(or V-cache) size of a request. H * L * D * P

Optimizations

- Mitigating internal fragmentation

CUDA VM APIs	vAttention VM APIs	Description	Latency (microseconds) 64KB	128KB	256KB	2MB
cuMemAddressReserve *	vMemReserve *	Allocate a buffer in virtual memory	18	17	16	2
cuMemCreate *	vMemCreate *	Allocate a handle in physical memory	1.7	2	2.1	29
cuMemMap	vMemMap	Map a physical handle to a virtual buffer	8	8.5	9	2
cuMemSetAccess	-	Enable access to a virtual buffer	-	-	-	38
cuMemUnmap	-	Unmap physical handle from a virtual buffer	-	-	-	34
cuMemRelease *	vMemRelease *	Free physical pages of a handle	2	3	4	23
cuMemAddressFree *	vMemFree *	Free a virtual memory buffer	35	35	35	1

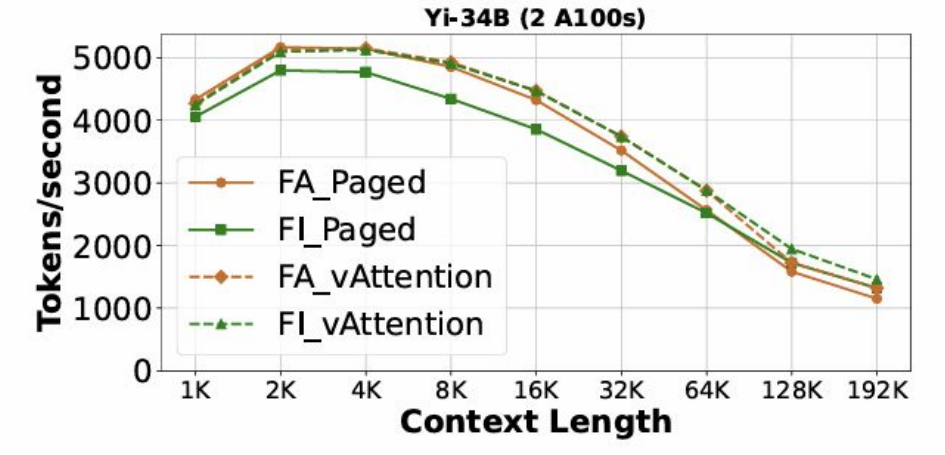
- Overlapping memory allocation with compute.
  - memory demand for a decode iteration is known ahead-of-time
- Deferred reclamation



Evaluation

Prefill stage throughput

(Higher the better)



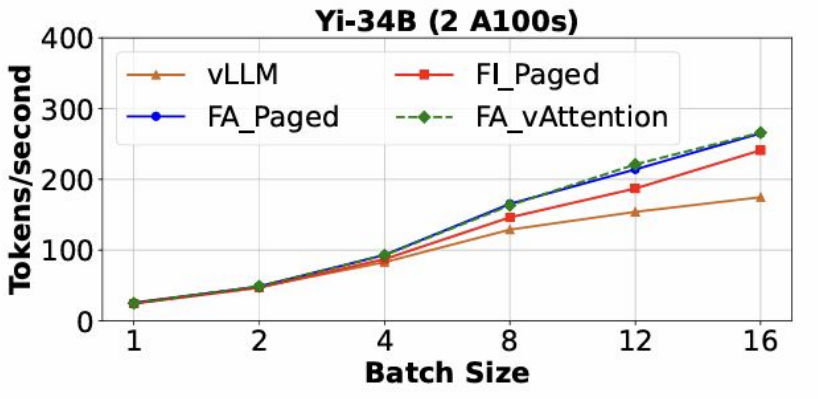
vAttention backed systems outperform the paged counterparts.

Throughput for longer contexts is lower due to the quadratic complexity of prefill attention.

Evaluation

Decoding stage throughput

(Higher the better)



Relative gains of FA_vAttention increase over vLLM (PagedAttention) with the batch size

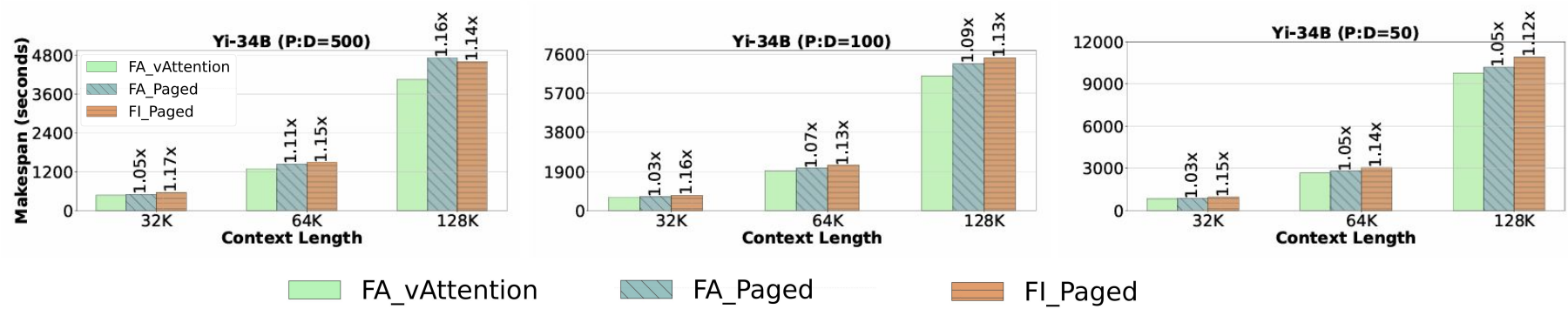
vAttention is only as good as the state-of the-art PagedAttention for decode throughput

- Memory bound nature of decode attention make it possible to hide the effect of additional compute that paging support requires

Evaluation

End-to-end Performance

(Lower the better)



- The ratio of prefill to decode tokens (referred as P:D).
- A higher P:D as well as a longer context indicates that the workload is more prefill bound.

Evaluation

```
-import flash_attn as fa
+import flashinfer as fi

-def flash_attn_prefill(q, k_cache, v_cache, kv_len):
-    k = k_cache[:, :kv_len, :, :]
-    v = v_cache[:, :kv_len, :, :]
-    return fa.flash_attn_func(q, k, v, causal=True)
+def flash_infer_prefill(q, k_cache, v_cache, kv_len):
+    q = q.squeeze(0)
+    k = k_cache.squeeze(0)[:kv_len, :, :]
+    v = v_cache.squeeze(0)[:kv_len, :, :]
+    return fi.single_prefill_with_kv_cache(q, k, v, causal=True)
```

- vAttention makes it feasible to replace one attention kernel with another with only a few lines of code changes
- While PagedAttention needs a paged kernel and hundreds lines of code changes

Q & A

References

- PagedAttention: https://doi.org/10.1145/3600006.3613165,

  https://blog.vllm.ai/2023/06/20/vllm.html

- vAttention: https://doi.org/10.48550/arXiv.2405.04437
- KV Cache figure: https://www.youtube.com/watch?v=80bIUggRJf4, https://arxiv.org/pdf/2311.18677
- Prefill Phase Figure: https://www.researchgate.net/figure/Prefill-and-decoding-phase-in-the-LLM-inference_fig2_382331612
- Inference Figure: https://iclr-blog-track.github.io/2022/03/25/text-gen-via-lfd/
- Dynamic Allocation Figure: https://www.linkedin.com/pulse/llm-inference-hwsw-optimizations-sharada-yeluri-wfdyc/

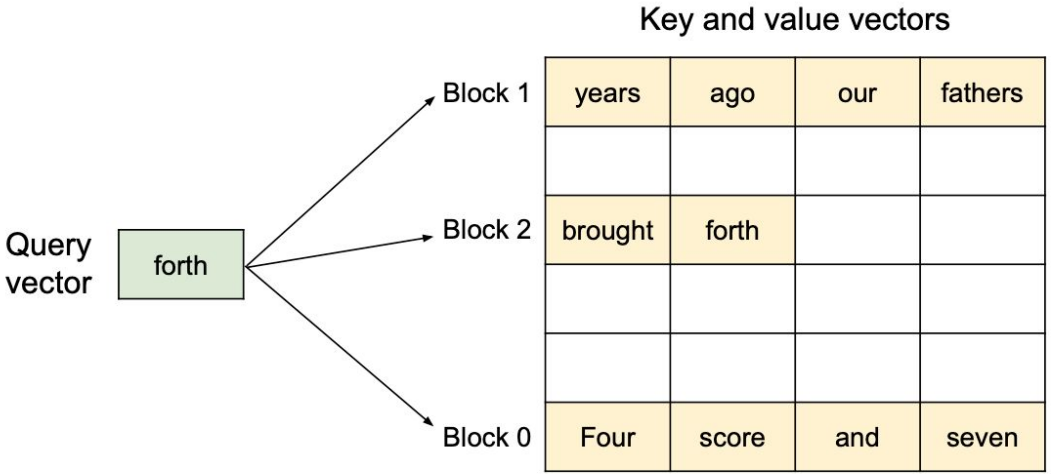
Backup Slides

Efficient management of KV cache

Hendrik Mayer, Yeda Song, Yuchen Xia

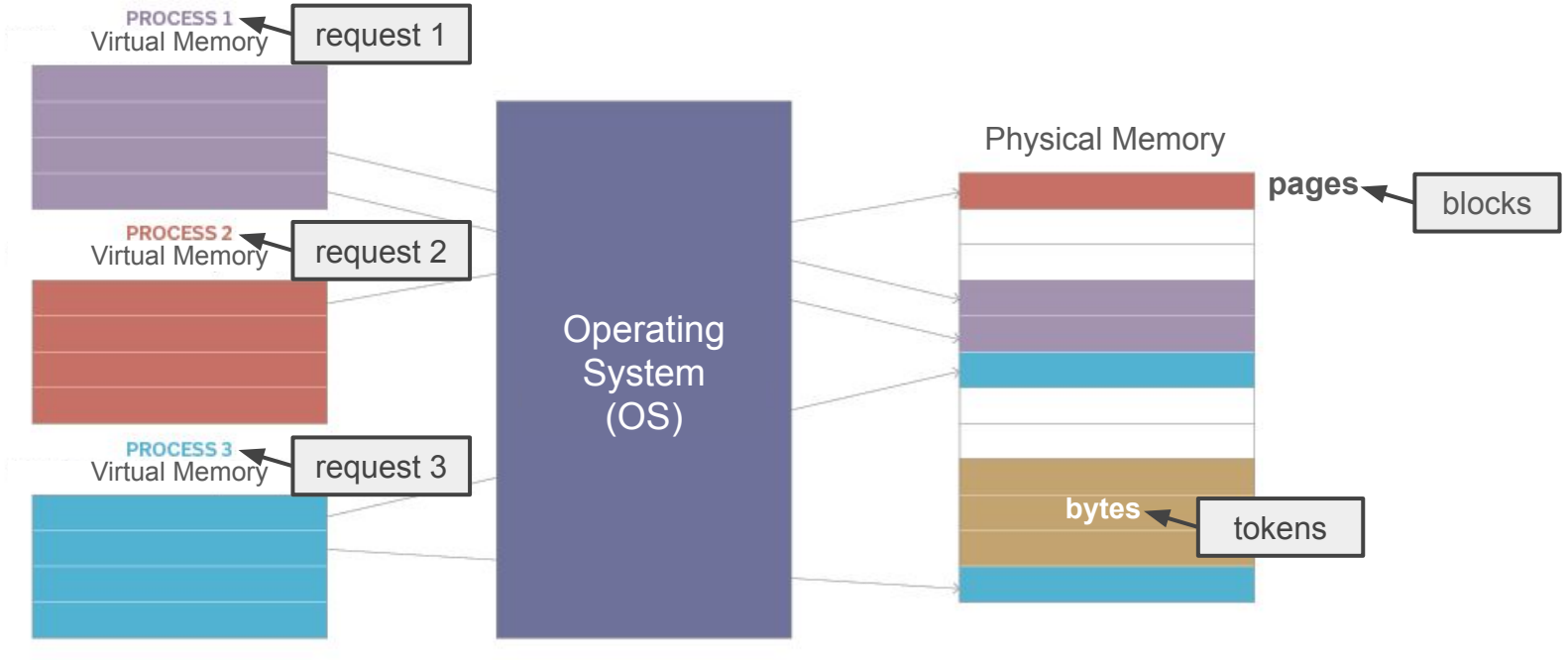
PagedAttention Overview

- PagedAttention divides the request's KV cache into small blocks, each of which can contain the attention keys and values of a fixed number of tokens
- Blocks not necessarily stored in contiguous memory
- Inspired by virtual memory with paging



Virtual Memory with Paging

- OS's solution to memory fragmentation and sharing: *virtual memory with paging*
- One can think of **requests** as ***processes,*** **blocks** as ***pages***, and **tokens** as ***bytes***



Serving LLMs with vAttention

Algorithm 1 Using vAttention in a serving framework.

```
1:  max_batch_size ← B
2:  cache_seq_len ← [0]*B
3:  req_batch_idx ← dict()
4:  vattention.init_cache(config_params)
5:  while !request_pool.is_empty() do
6:      for R_i in new_requests do
7:          if can_schedule(R_i) then
8:              idx ← vattention.alloc_reqid()
9:              req_batch_idx[R_i] ← idx
10:             cache_seq_len[idx] ← prompt_len(R_i)
11:         end if
12:     end for
13:     vattention.step(cache_seq_len)
14:     model.forward()
15:     for R_i in active_requests do
16:         idx ← req_batch_idx[R_i]
17:         if is_complete(R_i) then
18:             cache_seq_len[idx] ← 0
19:             vattention.free_reqid(idx)
20:         else
21:             cache_seq_len[idx] + = 1
22:         end if
23:     end for
24: end while
```

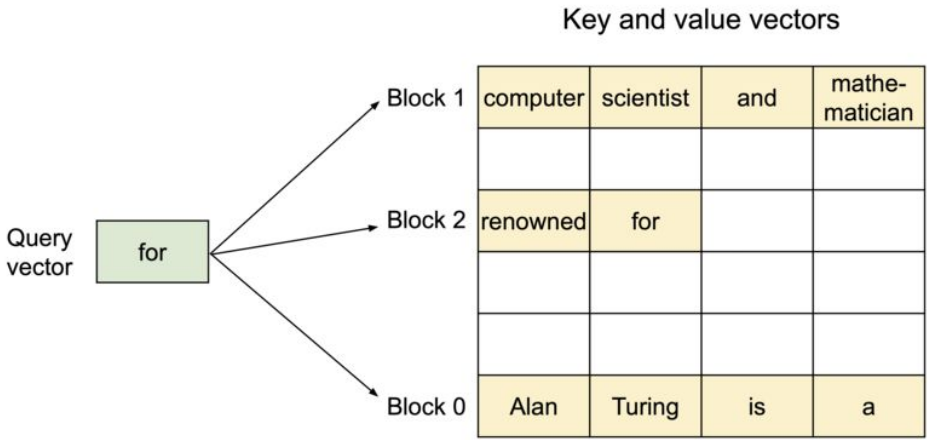
Assign an available reqid to new request

Cache context length of the request

Map and allocate physical memory

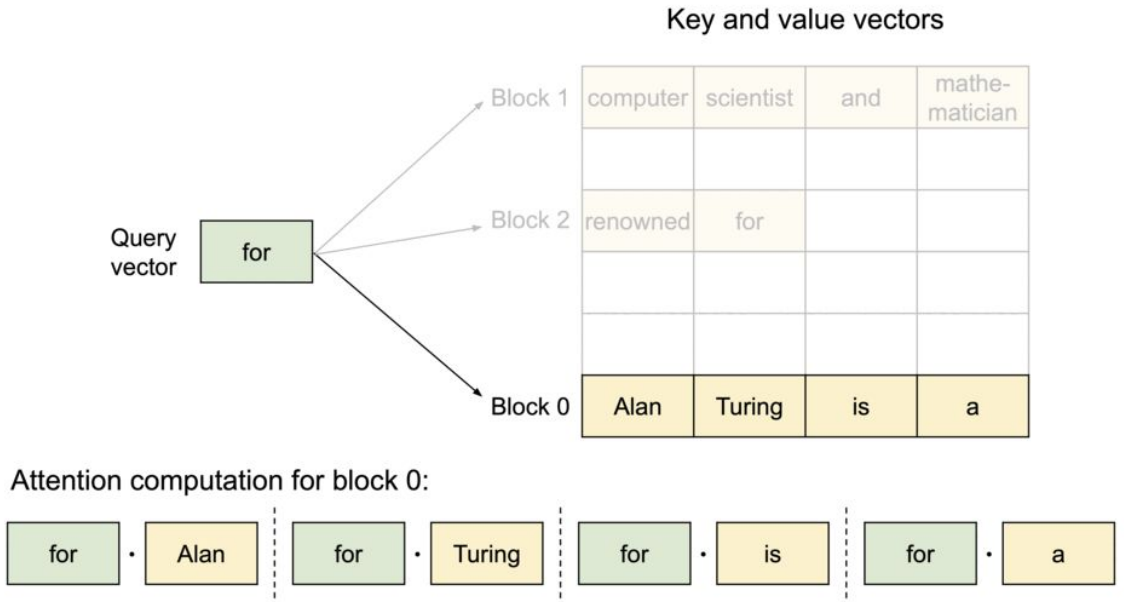
Attention Comptutation with PagedAttention

Sequence: Alan Turing is a computer scientist and mathematician renowned for



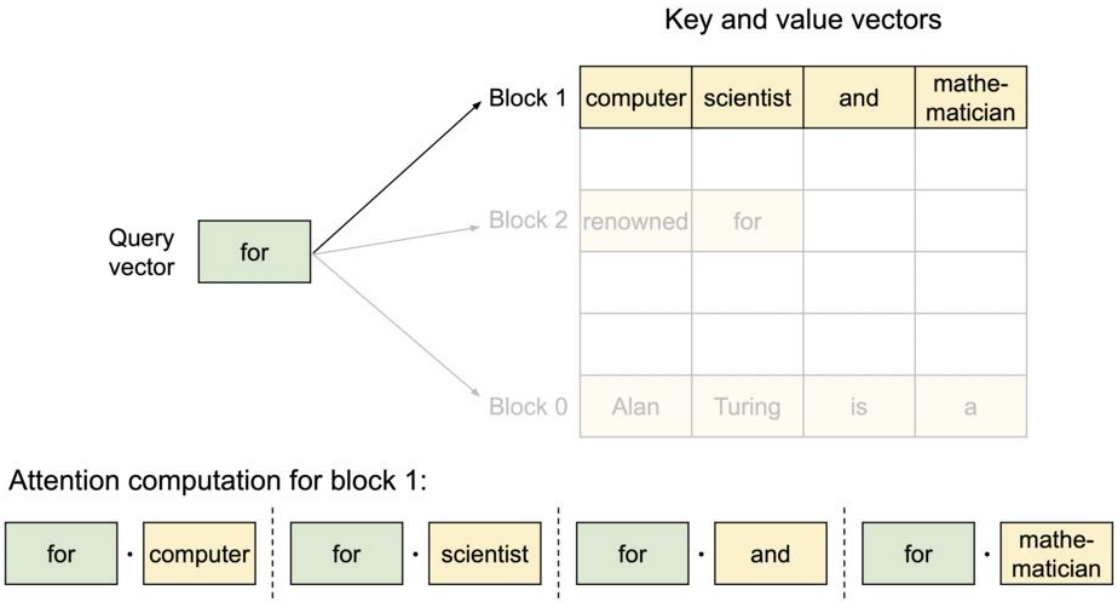
Attention Computation with PagedAttention

Sequence: Alan Turing is a computer scientist and mathematician renowned for



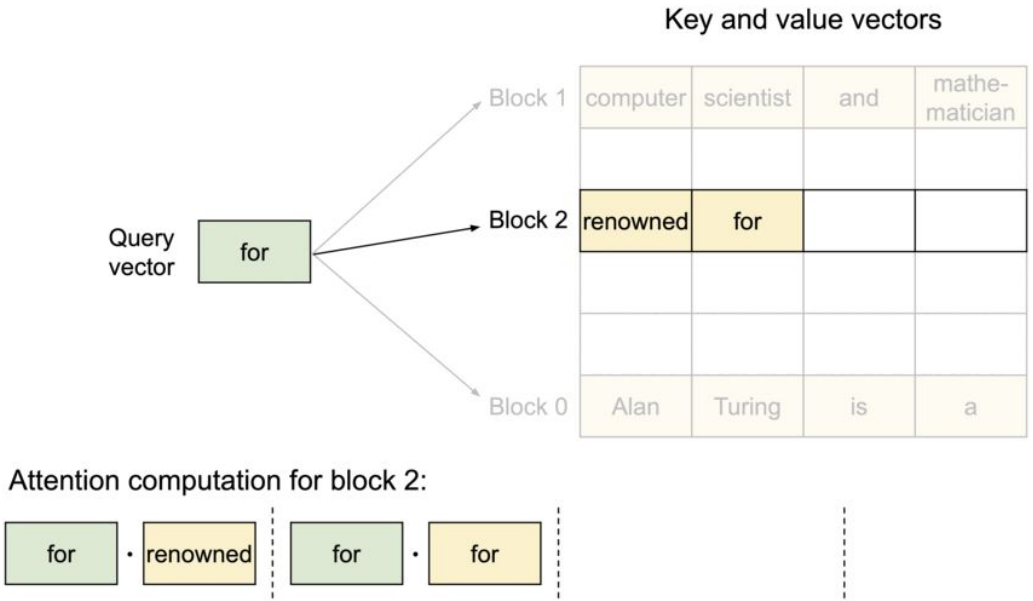
Attention Computation with PagedAttention

Sequence: Alan Turing is a computer scientist and mathematician renowned for

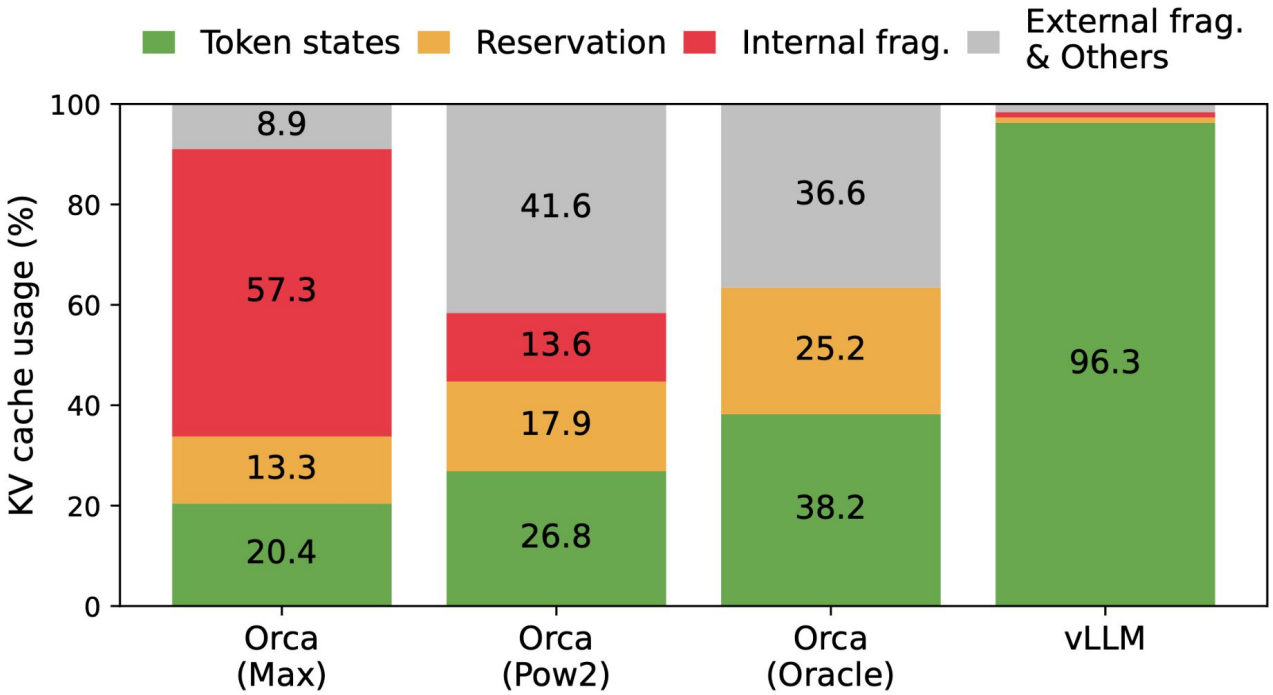


Attention Computation with PagedAttention

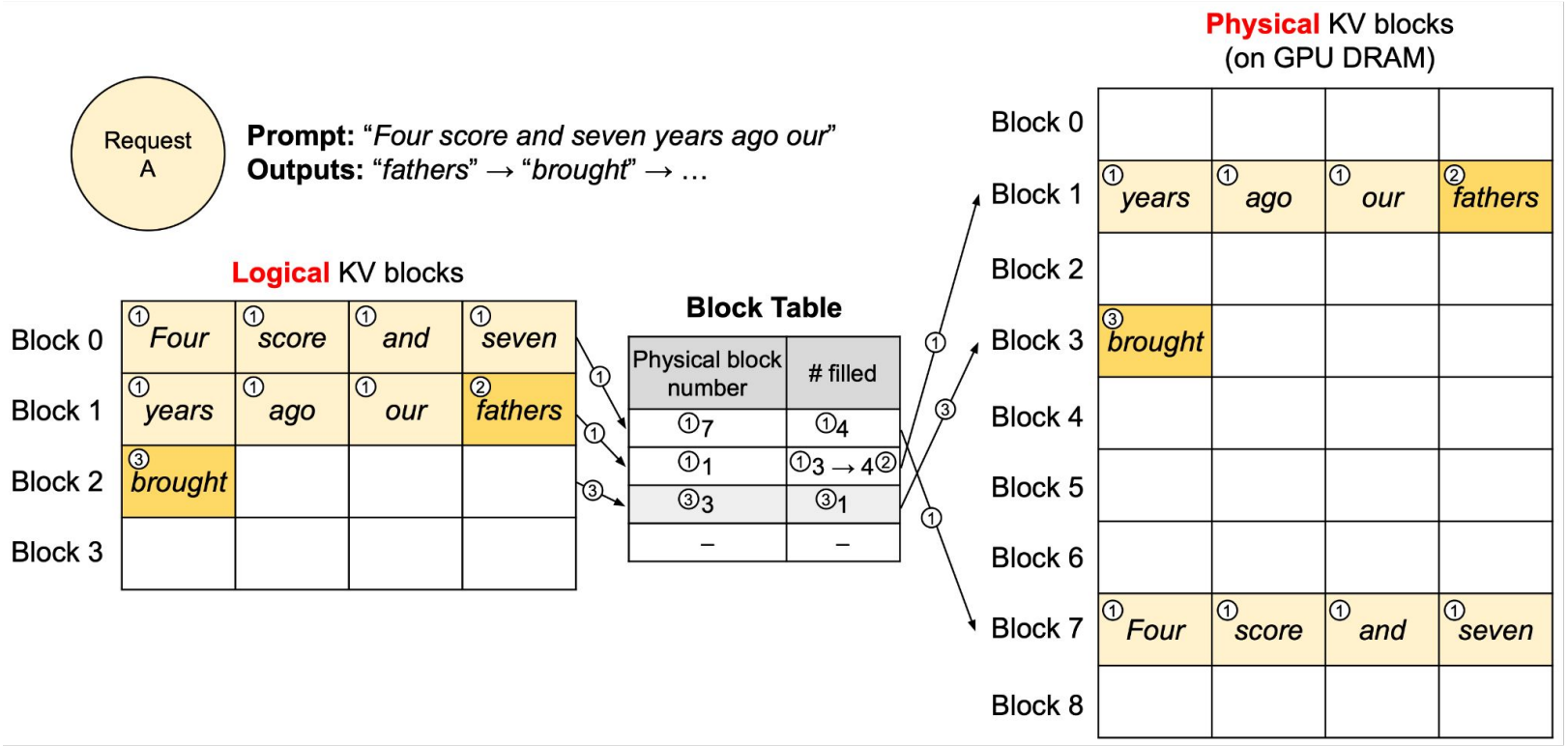
Sequence: Alan Turing is a computer scientist and mathematician renowned for



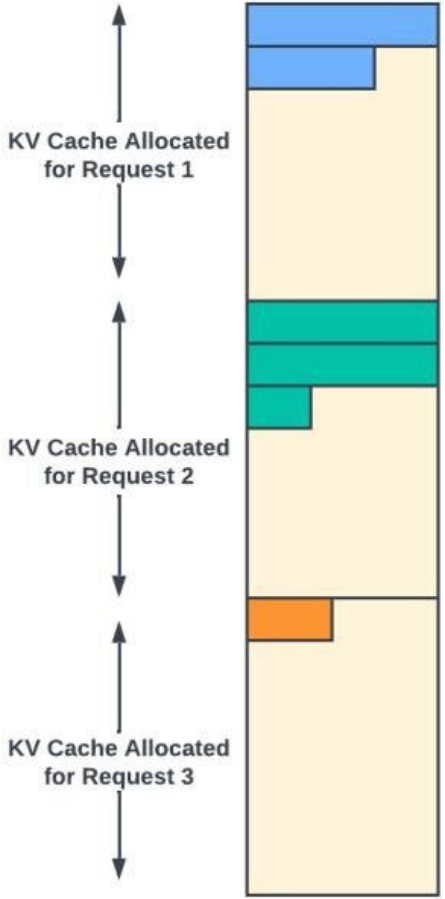
Memory Waste Comparison to Existing Systems



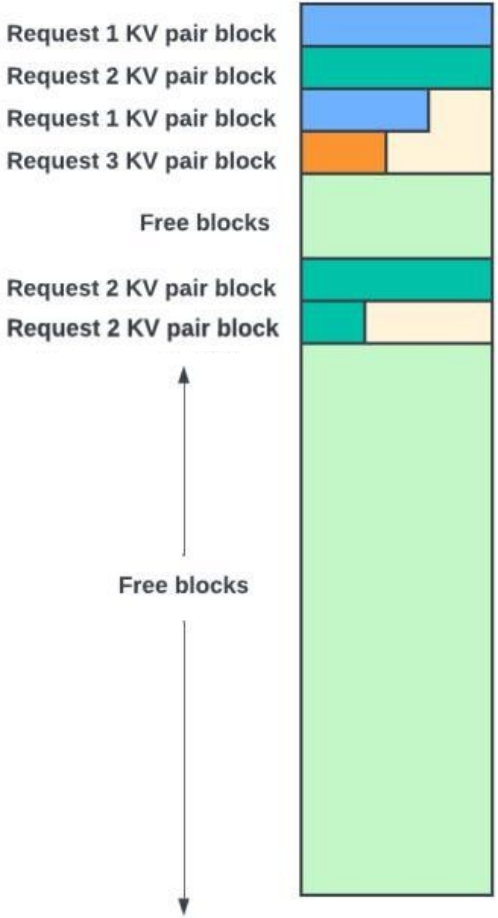
Dynamic Allocation of KV Blocks



Previous Systems



PagedAttention



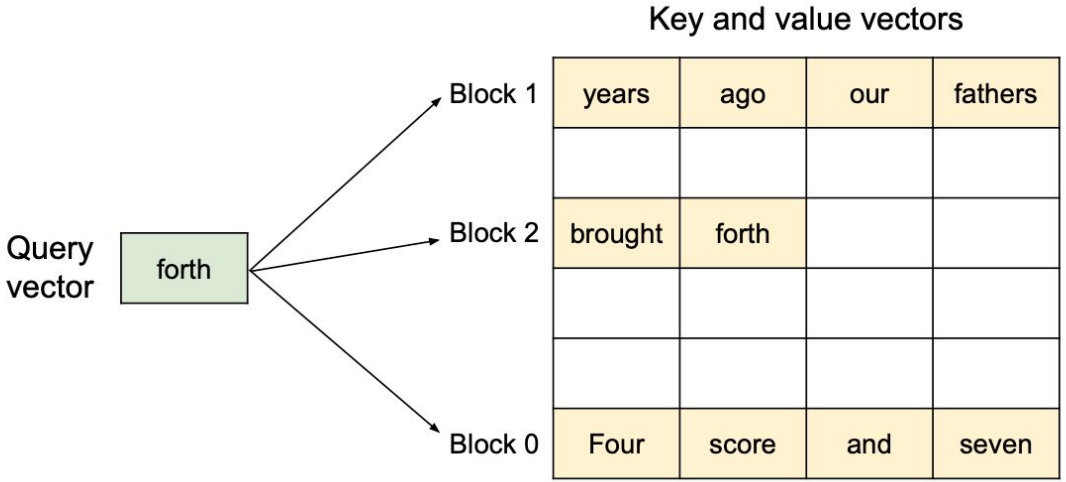
Pre-allocated memory space. can be freed only after the inference is complete

Free space. Can be used dynamically for KV pairs

Memory waste is minimized

PagedAttention Algorithm

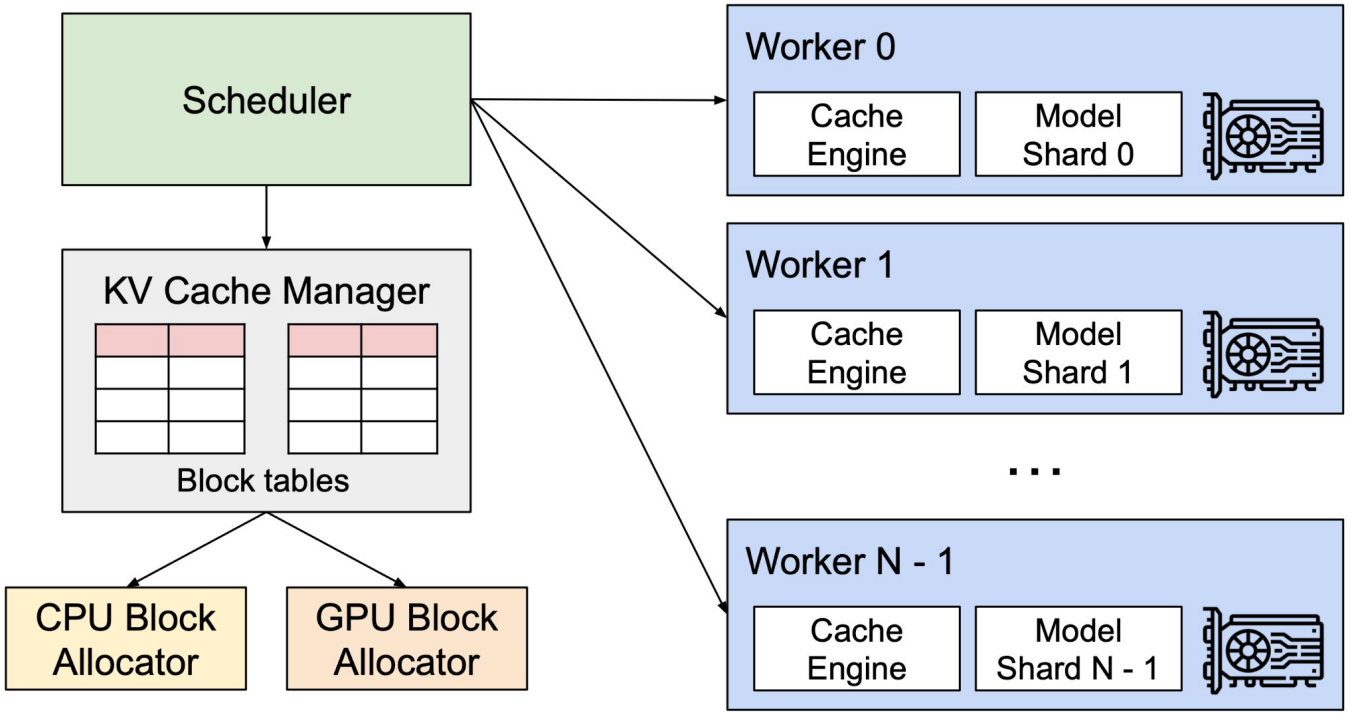
- Traditional attention computation is transformed into a blockwise computation
- The kernel multiplies some query vector q with the key values K_j in a block to compute the attention score vector A_{ij}
- A_{ij} is then multiplied by the value vector V_j in a block to derive that block's component to the attention output o_i



vLLM System Overview

- vLLM is a high-throughput distributed LLM serving engine using PagedAttention
- Increase in LLM serving by 2-4x compared to state-of-the-art systems
- The improvements over the state of the art are more pronounced with
    - Longer sequences
    - Larger models
    - More complex decoding algorithms

vLLM System Diagram



Scheduling and Preemption

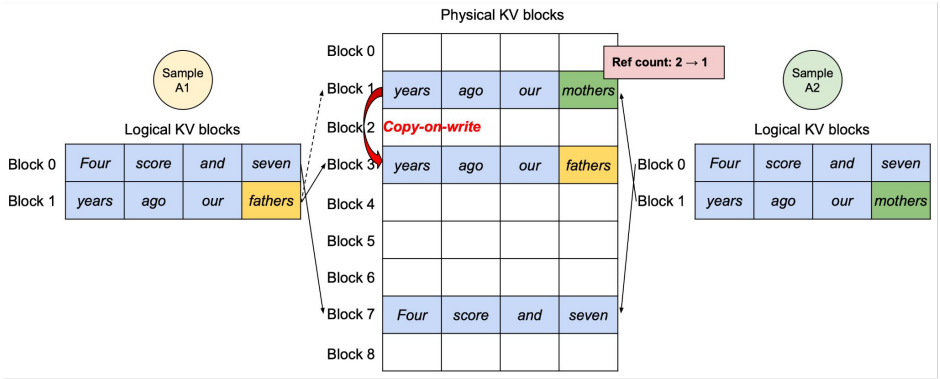
- vLLM adopts a first-come-first-serve (FCFS) scheduling policy for all requests
- Requests that arrive latest are preempted first to allow earlier requests to finish
- vLLM uses an all-or-nothing eviction policy where blocks of a sequence or sequence group (i.e. beam search candidates) are evicted together
- To recover an evicted block two techniques are considered
  - Swapping: evicted blocks are copied to CPU memory
  - Recomputation: sequences are rescheduled

Distributed Execution

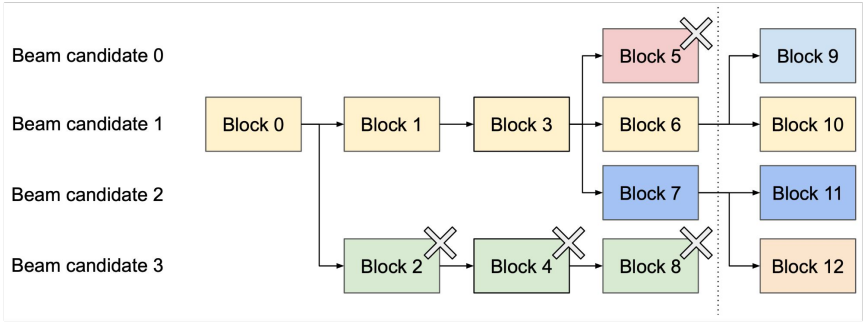
- A centralized KV Cache manager is used across all GPU workers, where each worker computes a portion of the KV cache
- The scheduler broadcasts control messages with input token IDs and block tables to GPU workers
- GPU workers obtain all necessary memory information at the beginning of a decoding iteration and then independently execute
- These GPU workers do the following:
    - Execute the model
    - Read the KV cache
    - Synchronize results using all-reduce communication

Other Decoding Scenarios

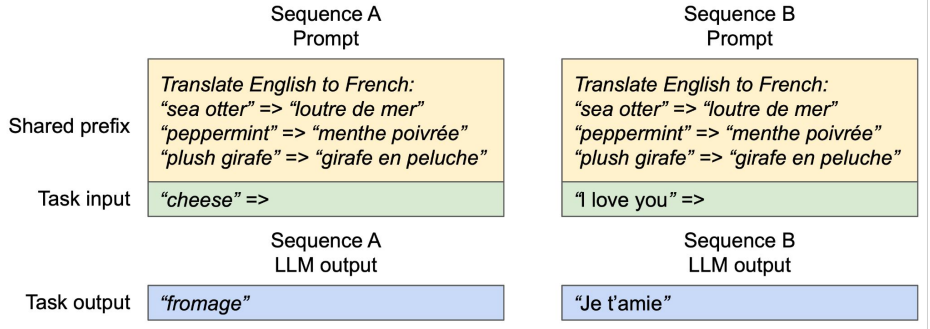
1. Parallel sampling



2. Beam search

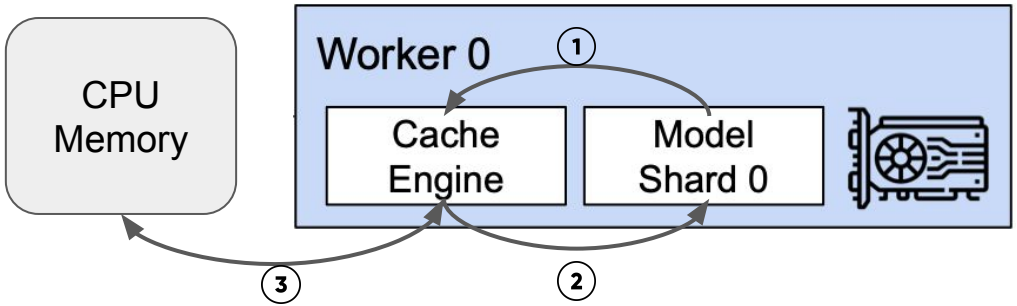


3. Shared prefix



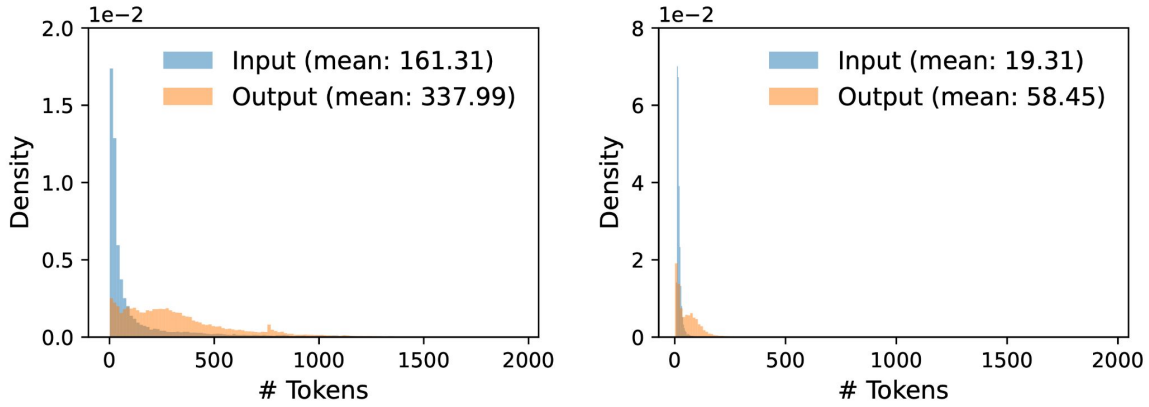
Kernel-level Optimizations during Implementation

- Existing GPU kernels do not optimize PagedAttention
- The following GPU kernels were developed:
  1. Fused reshape and block write
  2. Fusing block read and attention
  3. Fused block copy



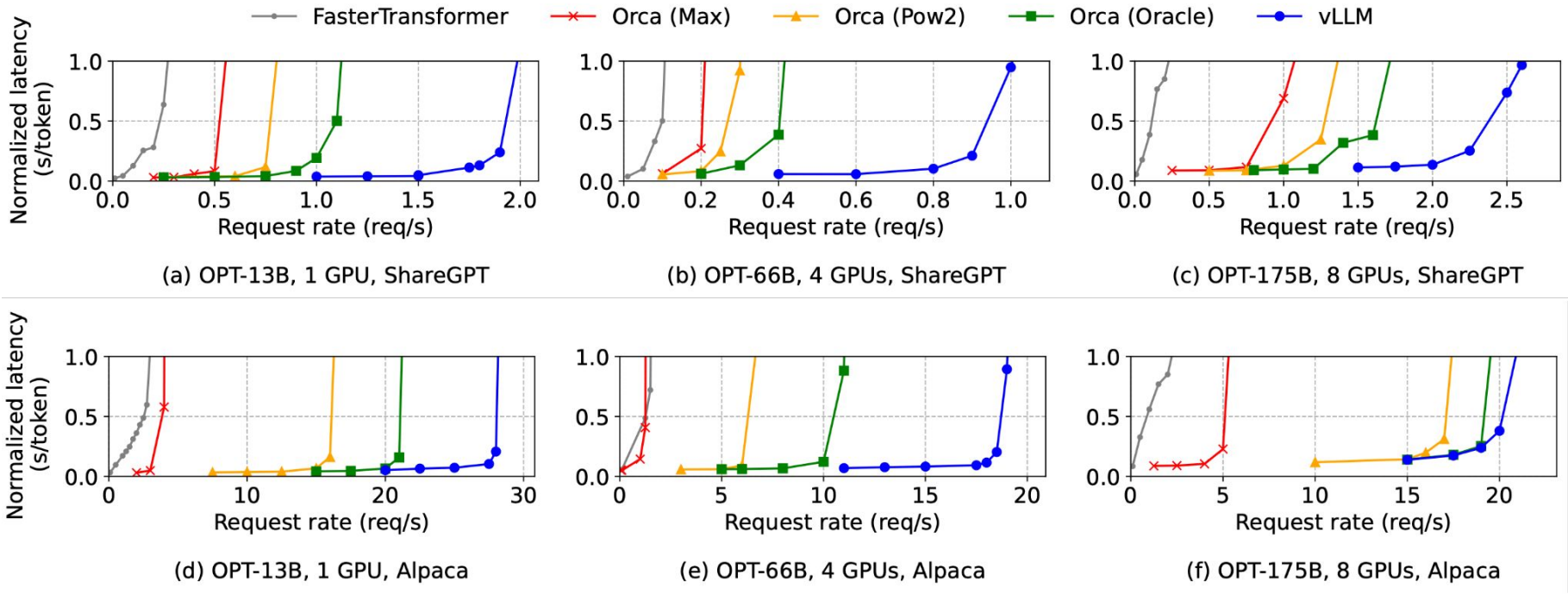
Workloads for Evaluation

- Synthesized dataset using real footprints (i.e., Input/Output lengths)
  - Request timestamp $\sim$ Poisson(λ), λ: request rate

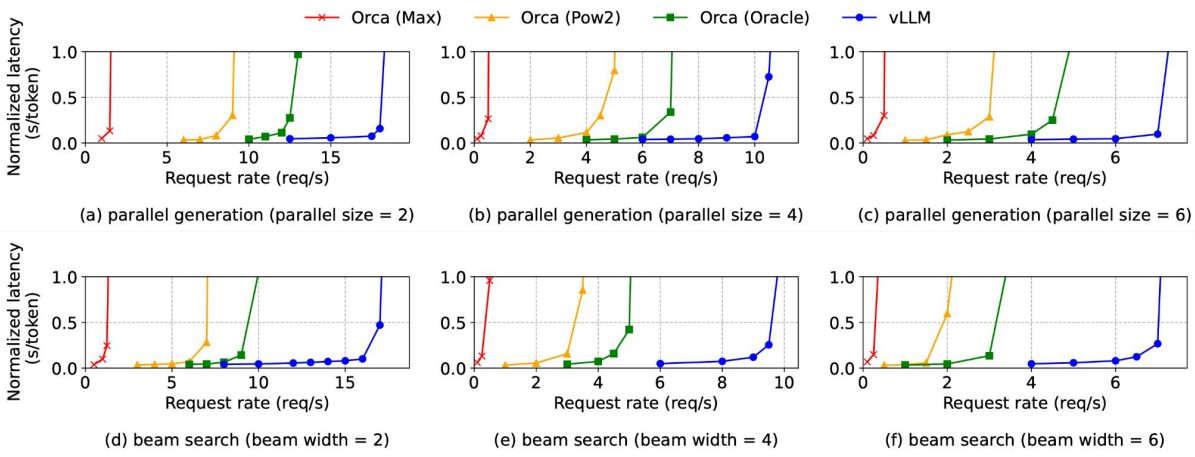


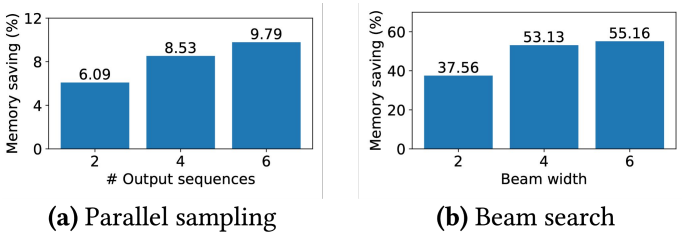
(a) ShareGPT **(b)** Alpaca

High Throughput of PagedAttention



Effectiveness of Memory Sharing

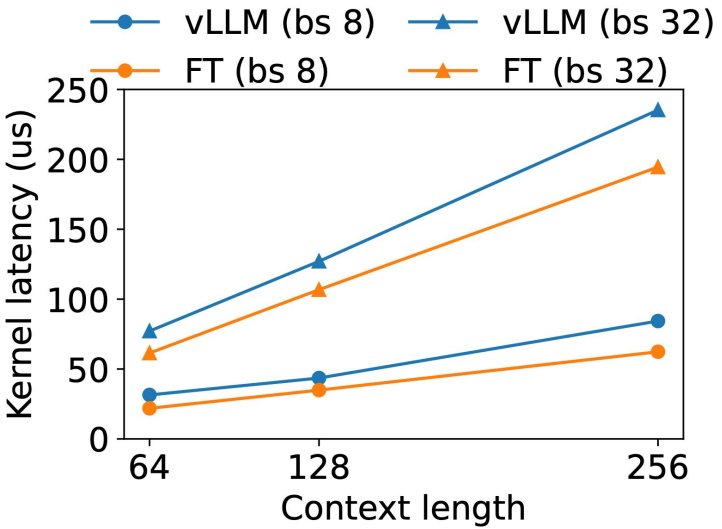




- PagedAttention retains higher throughput in parallel sampling and beam search scenarios
- Saved memory through PagedAttention

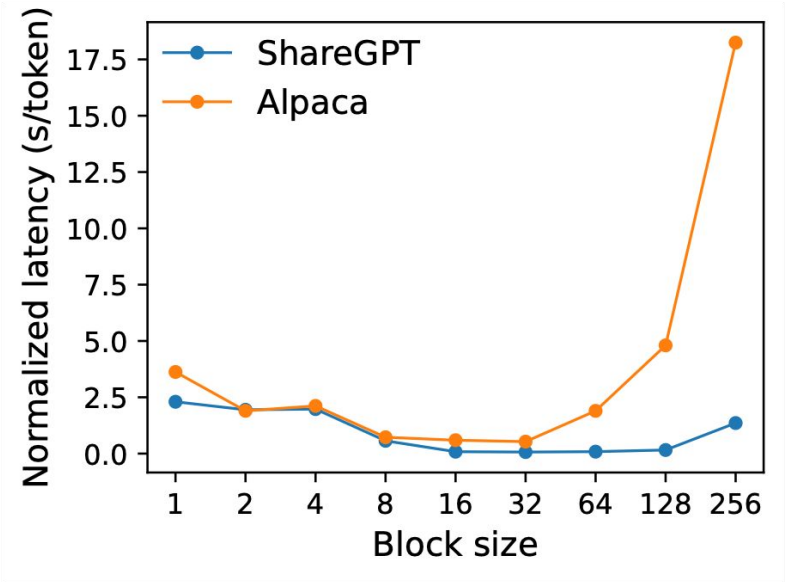
Kernel Microbenchmark

- 20–26% higher attention kernel latency compared to FasterTransformer



Latency of attention kernels

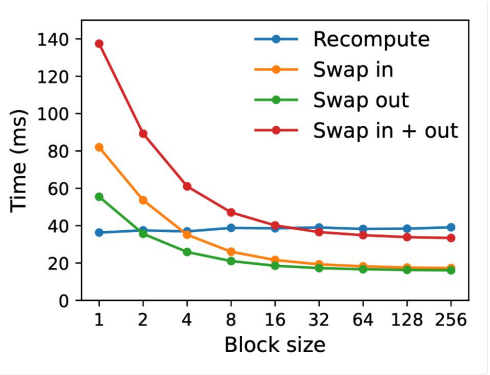
Ablation Study: Block Size



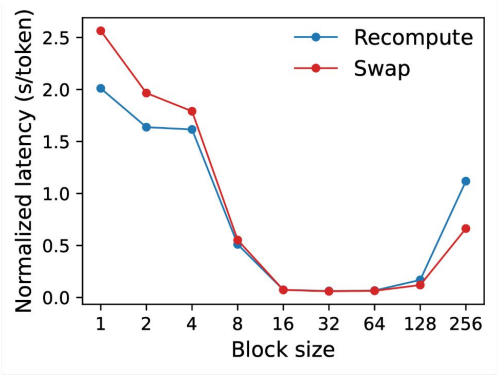
(b) End-to-end latency with different block sizes.

Ablation Study: Swapping v.s. Recomputation

- Swapping overhead increases with small block sizes
- Recomputation overhead remains constant across different block sizes
- They showed similar latency with the optimal block size of 16



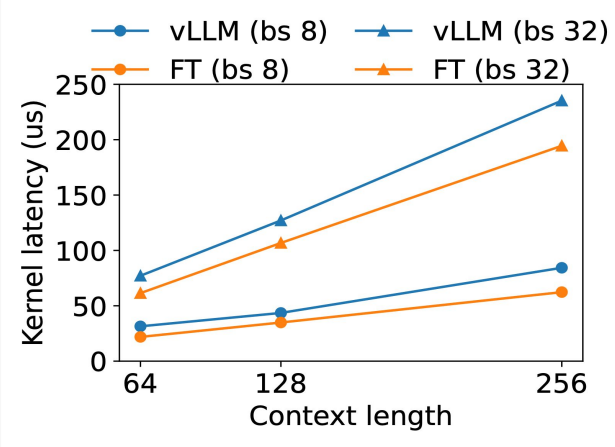
(a) Microbenchmark

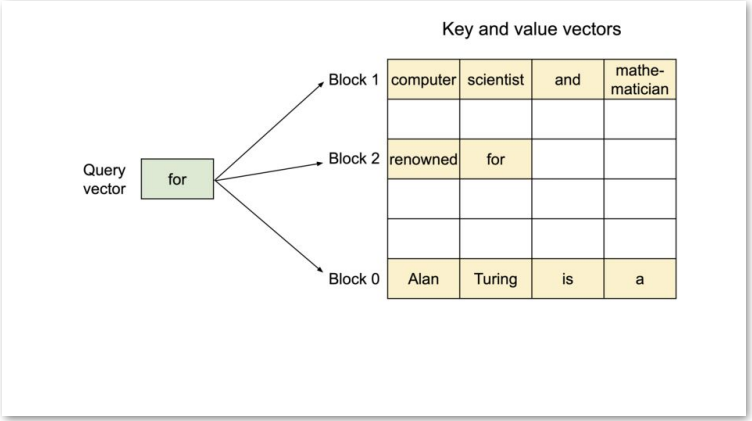


(b) End-to-end performance

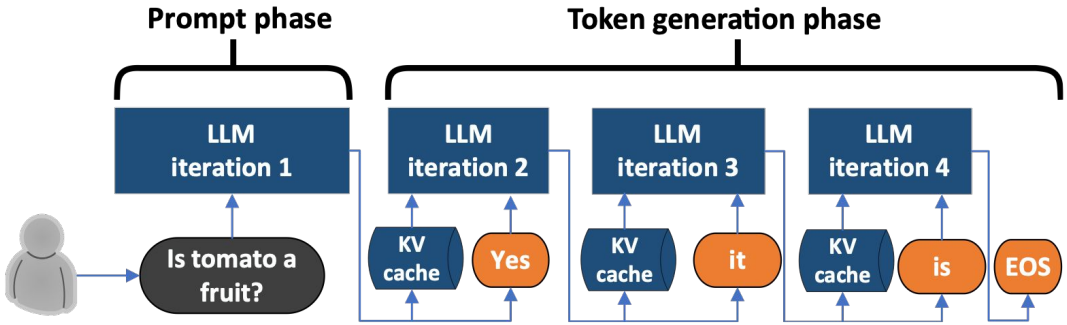
Performance Overhead of PagedAttention

- PagedAttention showed 20–26% **higher attention kernel latency** compared to FasterTransformer
- Due to the overhead of looking up block-tables, dealing with **non-contiguous virtual memory**





Insights into LLM Serving Systems



Key Observations:

1. KV-cache growth is known in advance: *uniformly* by one token per-iteration
2. KV-cache does not require high memory allocation bandwidth

Allocate virtual memory and physical memory separately.

Insights into LLM Serving Systems

Allocate virtual memory and physical memory separately.

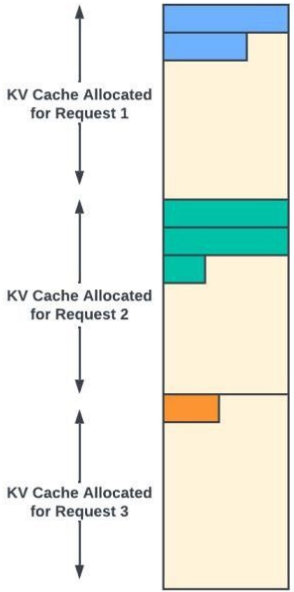
?

Insights into LLM Serving Systems

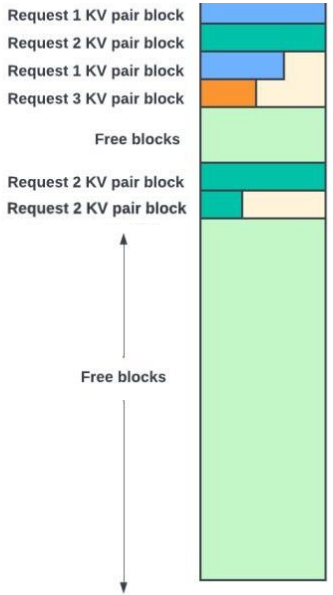
Allocate virtual memory and physical memory separately.

Virtual Memory

Pre-reserved allocation



Dynamic allocation



Physical Memory

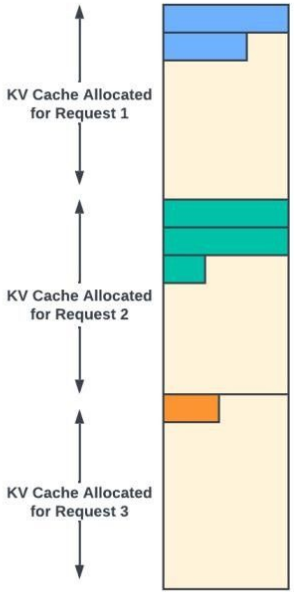
Insights into LLM Serving Systems

Allocate virtual memory and physical memory separately.

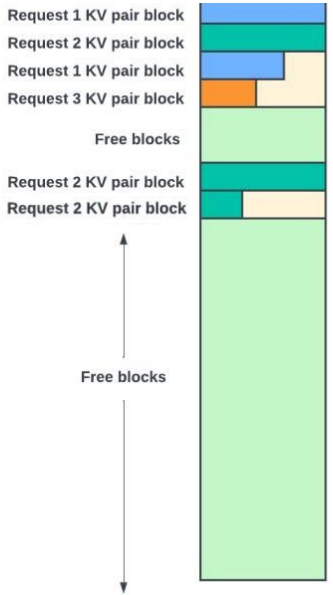
Virtual Memory

- **Contiguous**
- **Abundant**

Pre-reserved allocation



Dynamic allocation



Leveraging existing systems, reducing complexity

Physical Memory

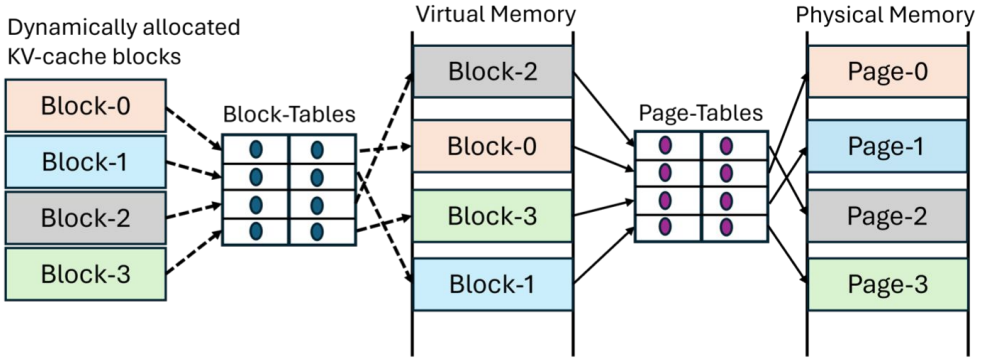
Design and Implementation

Allocate virtual memory and physical memory separately.

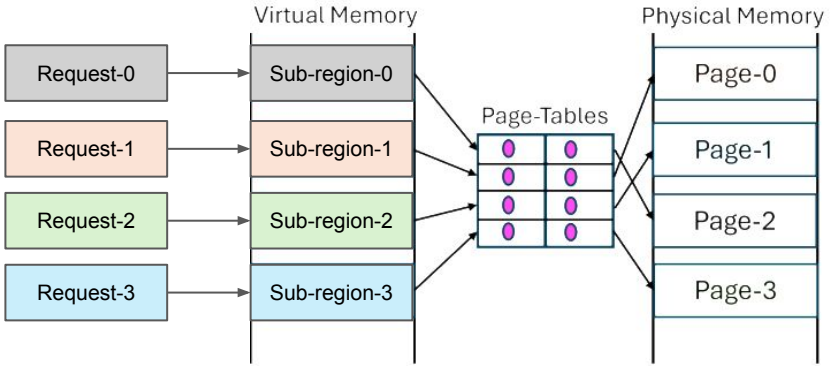
- Pre-reserving virtual memory
  - virtual memory is abundant (modern 64-bits systems provide a 128TB virtual memory per process)
  - pre-allocate it in size that is large enough to hold the KV-cache of the maximum batch size
- Deferring the allocation of physical memory to runtime
  - Efficient Use of Memory
- Leveraging the existing system support for demand paging
  - No need two-layer memory management
  - Better virtual contiguity of KV-cache

PagedAttention v.s. vAttention

PagedAttention

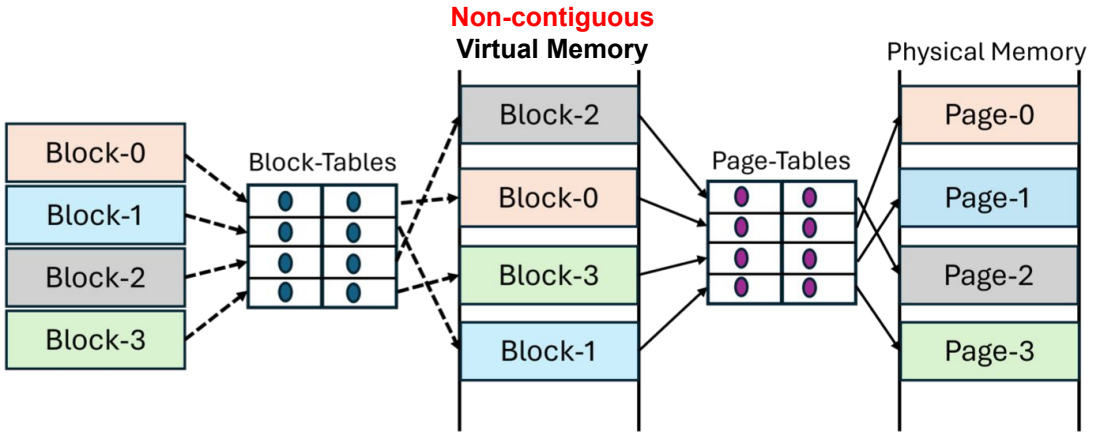


vAttention

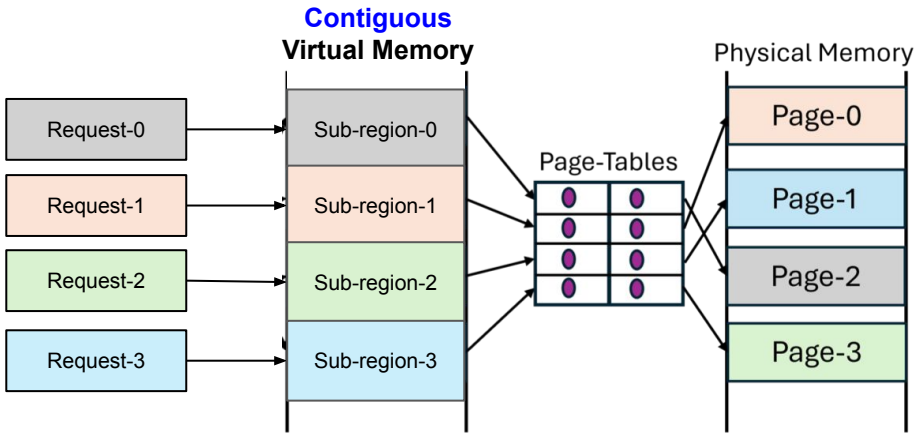


PagedAttention v.s. vAttention

PagedAttention

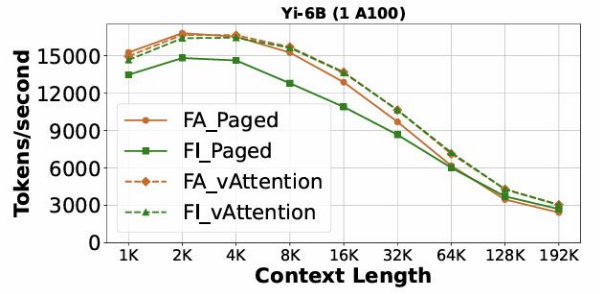


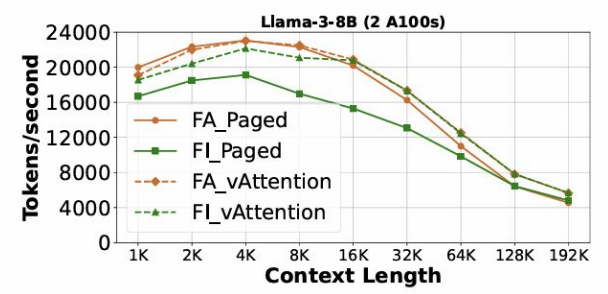
vAttention

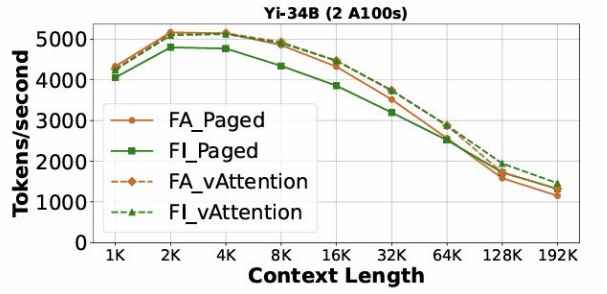


Evaluation

Prefill stage throughput





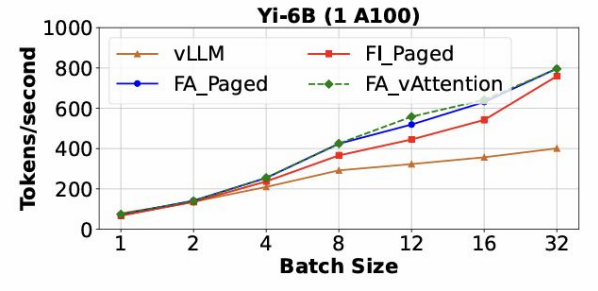


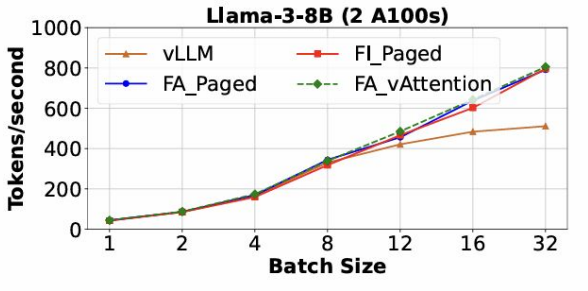
vAttention backed systems outperform the paged counterparts.

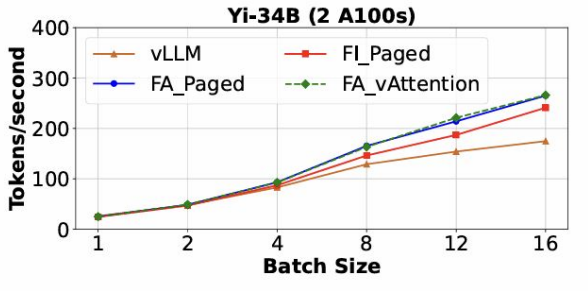
Throughput for longer contexts is lower due to the quadratic complexity of prefill attention.

Evaluation

Decoder stage throughput







FA_Paged and FA_vAttention outperform vLLM by up to 1.99×, 1.58× and 1.53× for Yi-6B, Llama-3 and Yi 34B, respectively.

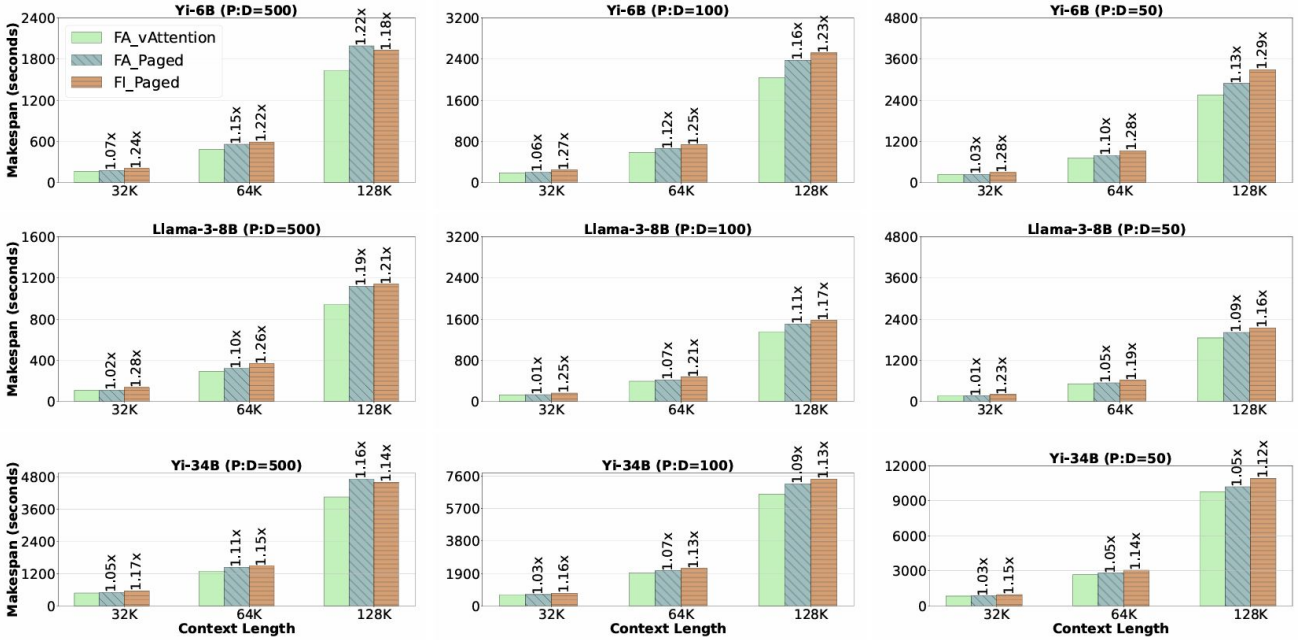
Relative gains of FA_vAttention and FA_Paged increase over vLLM with the batch size

vAttention is only as good as the state-of the-art PagedAttention for decode throughput

- Memory bound nature of decode attention make it possible to hide the effect of additional compute that paging support requires

Evaluation

End-to-end Throughput



- The ratio of prefill to decode tokens(referred as P:D).
- A higher P:D as well as a longer context indicates that the workload is more prefill bound.