vLLM: PagedAttention

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

Presenters: Kevin He, Zac Sardi-Santos,
Itzel Sanchez, Elena Ghazi

What is KV-Cache?

Recall: Attention

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

$$Q = XW_Q$$

$$K = XW_K$$

$$V = XW_V$$

Animations

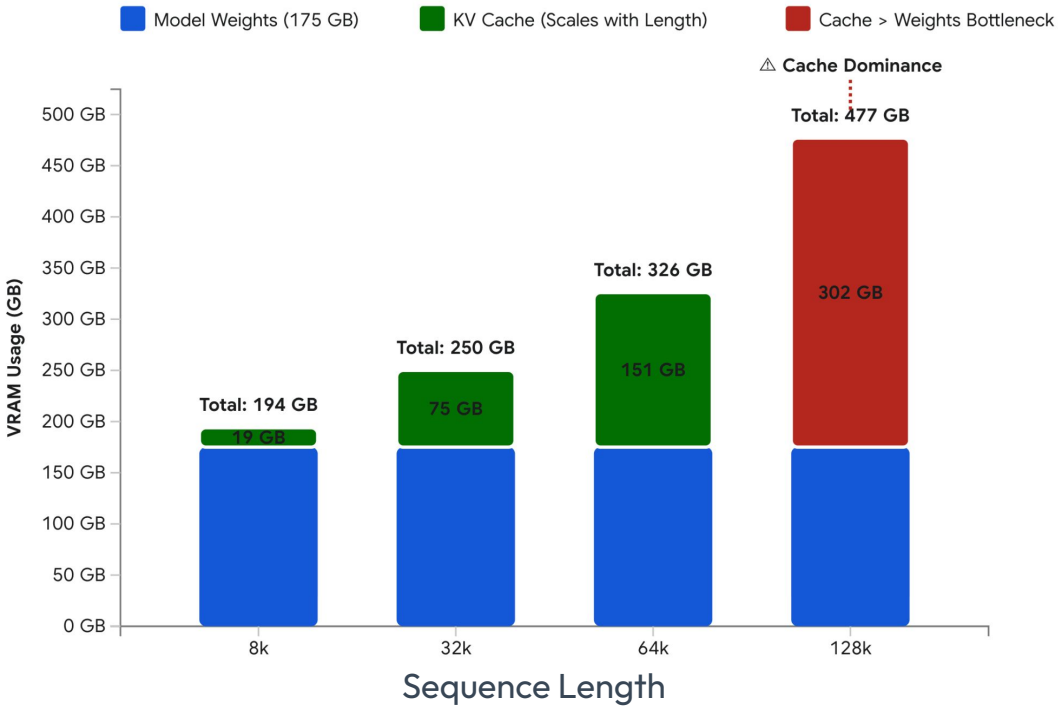
https://kevinhe.me/cs264-vllm/index.html

How does KV-cache scale?

KV-Cache size = 2 * batch size * **sequence length** * # layers * hidden size * precision

*for standard multi-head attention

GPT3: 175B parameters, INT8, Batch size = 16



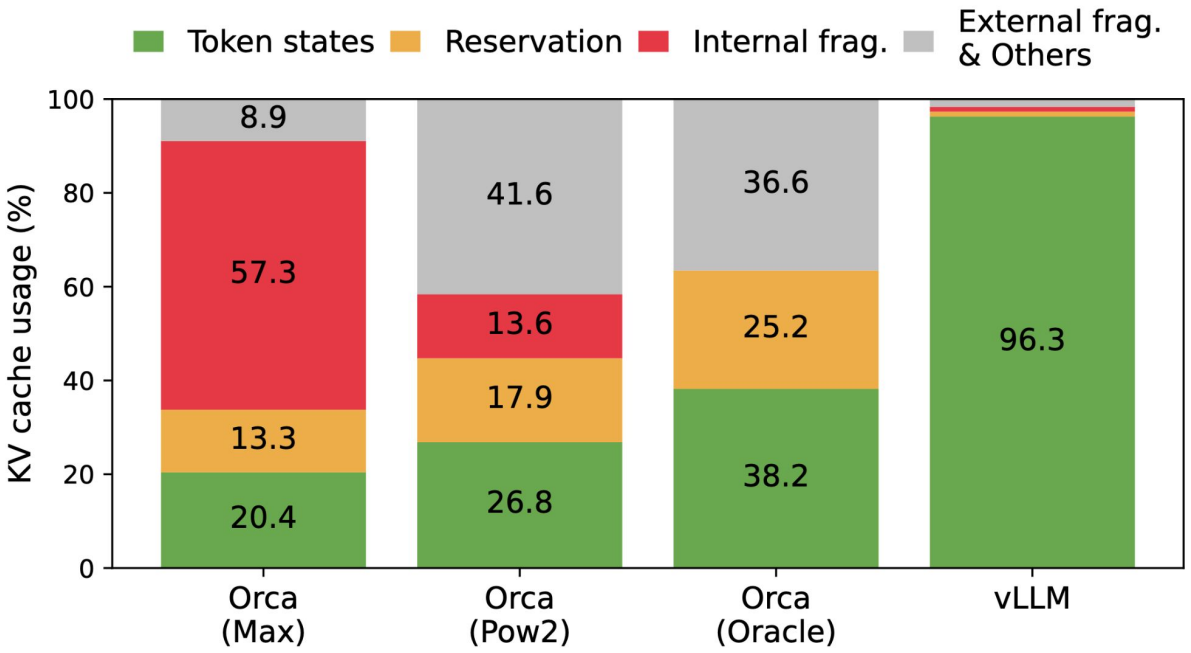
Discussion

How might we allocate GPU memory to the KV-Cache?

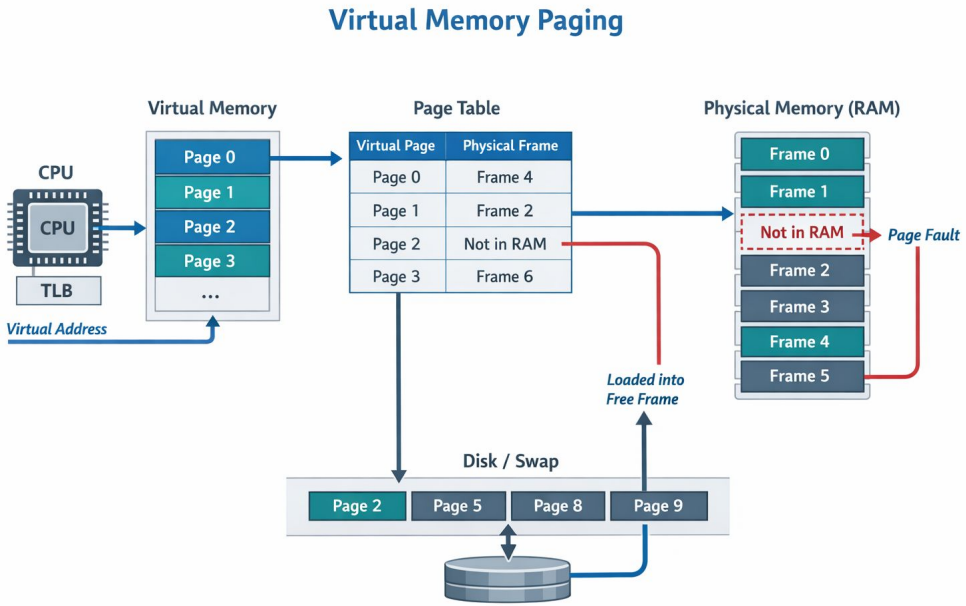
Why might dynamically growing/shrinking KV-cache lead to wasted memory?

What is the effect of wasted GPU memory on total token throughput?

Problem: Memory Fragmentation


Token states
Reservation
Internal frag.
External frag. & Others
KV cache usage (%)
100
80
60
40
20
0
8.9
57.3
13.3
20.4
41.6
13.6
17.9
26.8
36.6
25.2
38.2
96.3
Orca (Max)
Orca (Pow2)
Orca (Oracle)
vLLM

Recall: CPU Virtual Memory

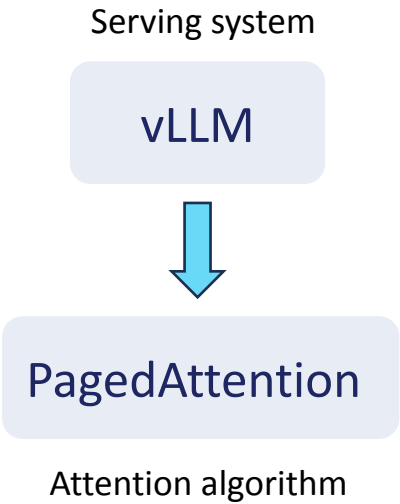


What if we applied this technique to KV-cache?

Solution: Paging KV-Cache and Attention

Parallel Sampling & Beam Search

Solutions: PagedAttention & vLLM



Rigid contiguous storage flexible block-based storage

OS virtual memory analogy

Operating systems

Program’s logical view

Logical page

Page table

Maps logical to physical

Physical page

Where data lives in RAM

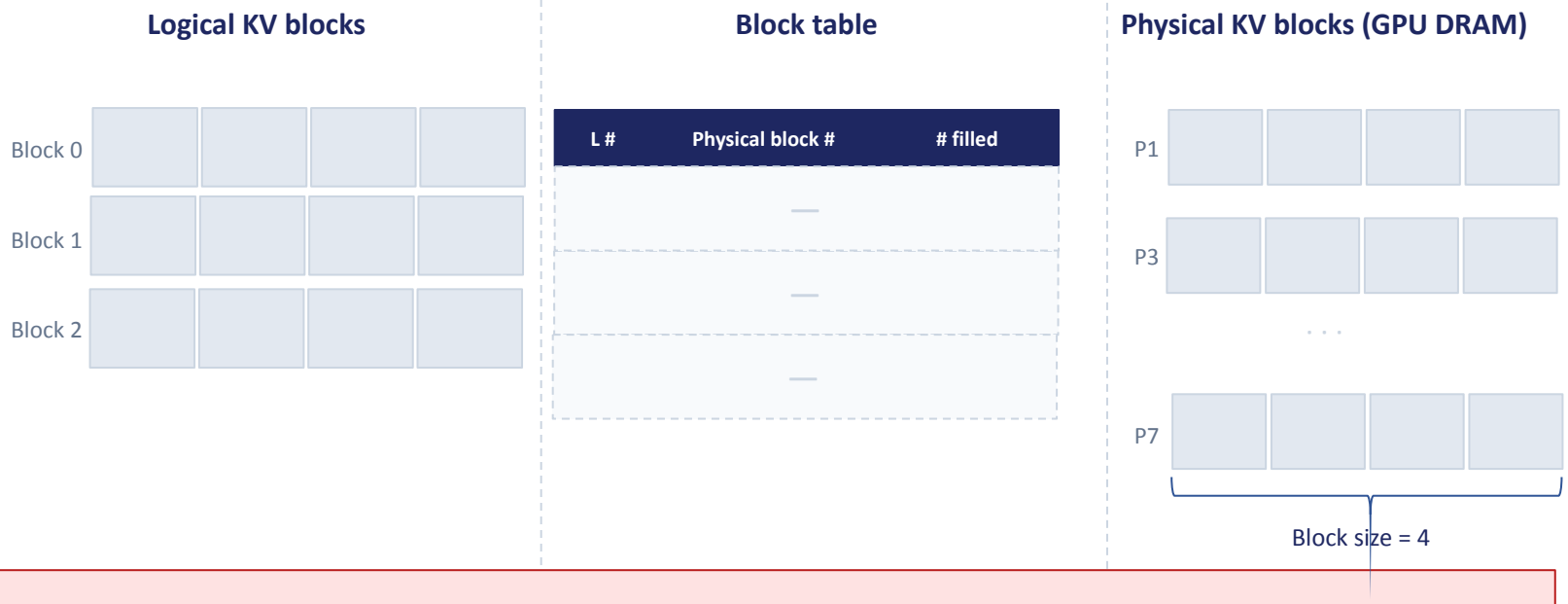
Program can behave as if memory is one continuous space.
OS does not have to reserve all physical memory in advance.

OS virtual memory analogy

Operating systems		vLLM
Program's logical view		Next logical chunk of the request's cache
Logical page	↔	Logical KV block
Page table Maps logical to physical	↔	Block table Maps logical to physical
Physical page Where data lives in RAM	↔	Physical KV block Where KV cache lives in GPU memory

Program can behave as if memory is one continuous space.
OS does not have to reserve all physical memory in advance.

Why is block-based KV-cache management useful?

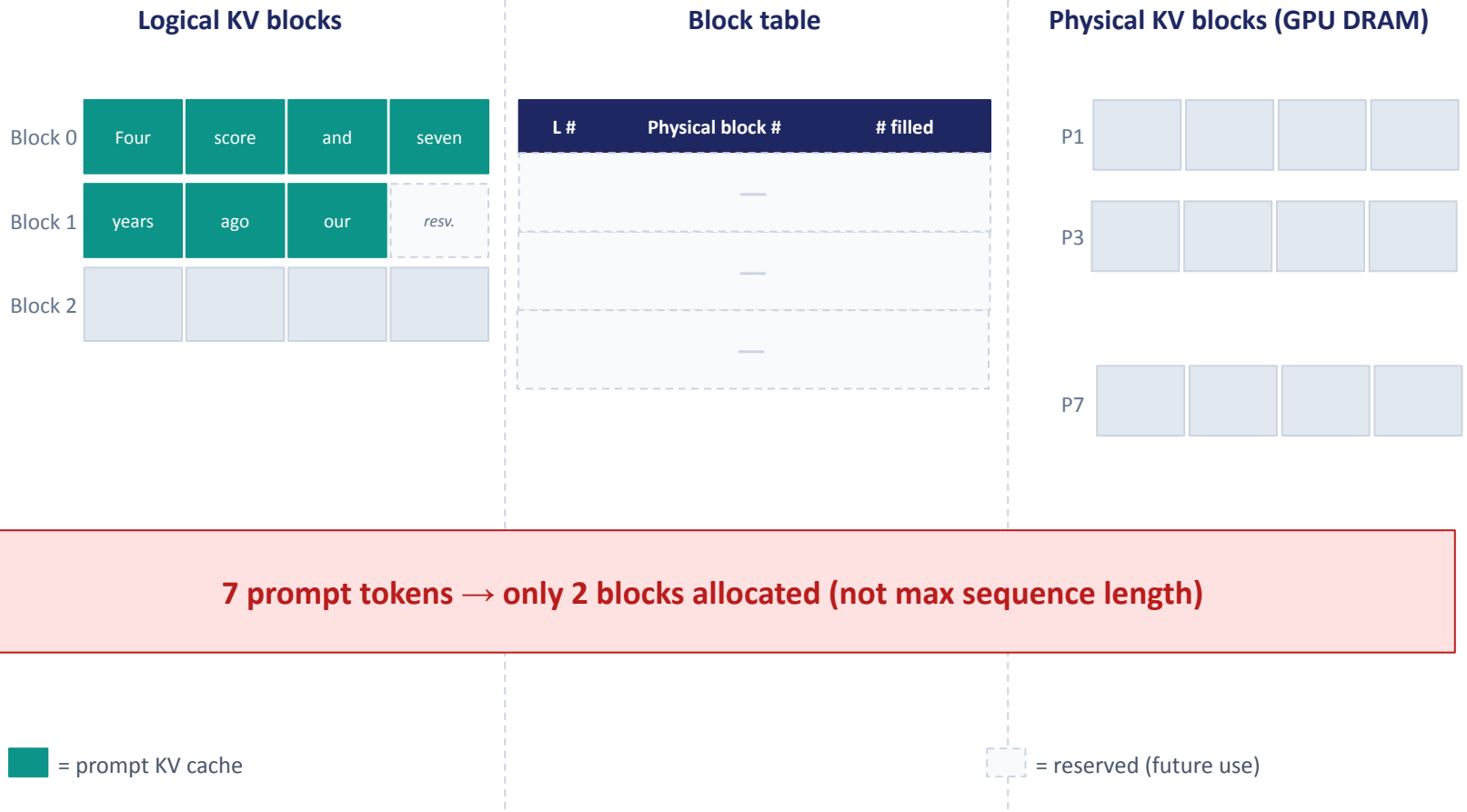


Allocate on demand + non-contiguous placement → less waste → more requests fit in memory

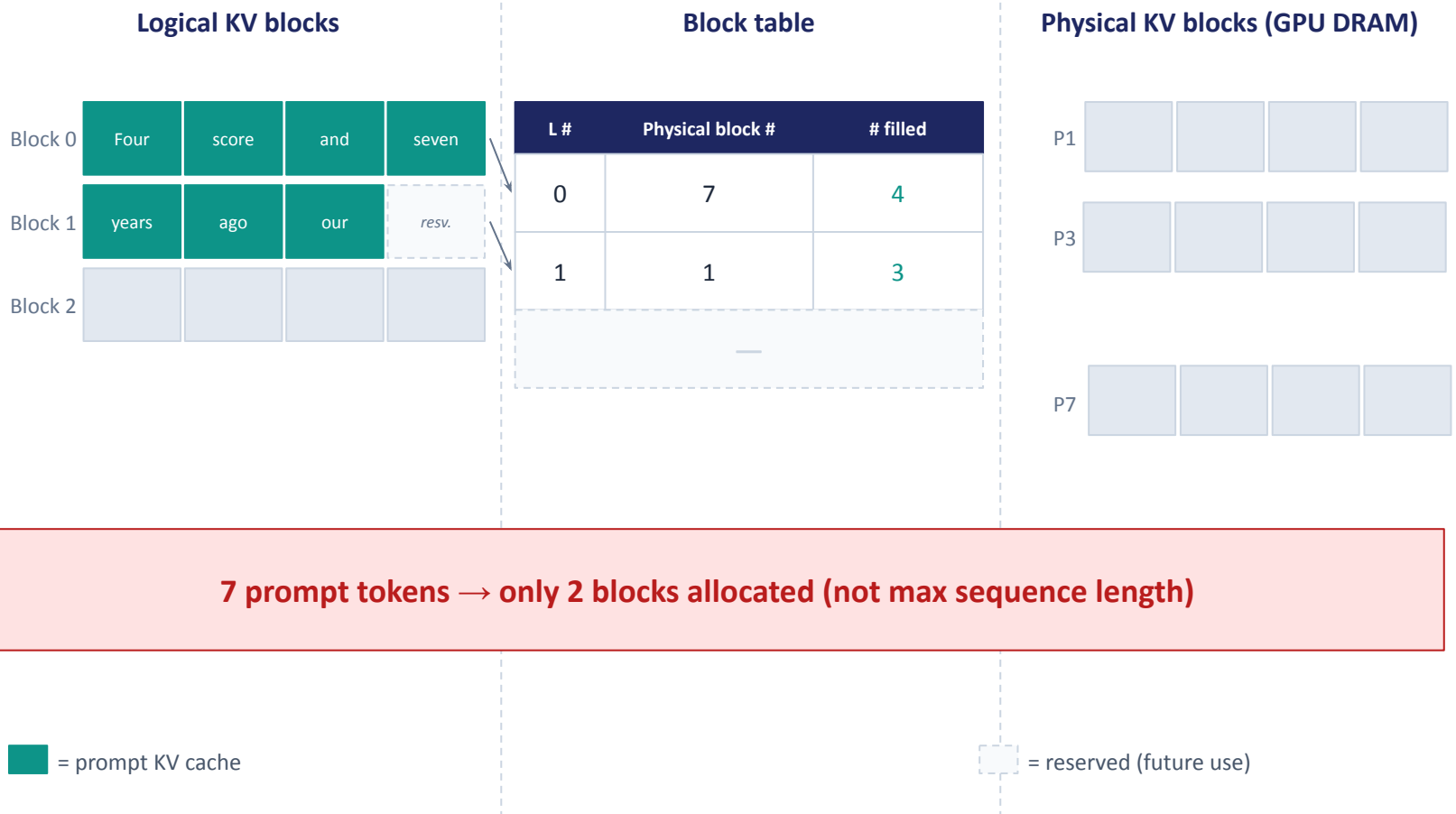
= prompt KV cache

= reserved (future use)

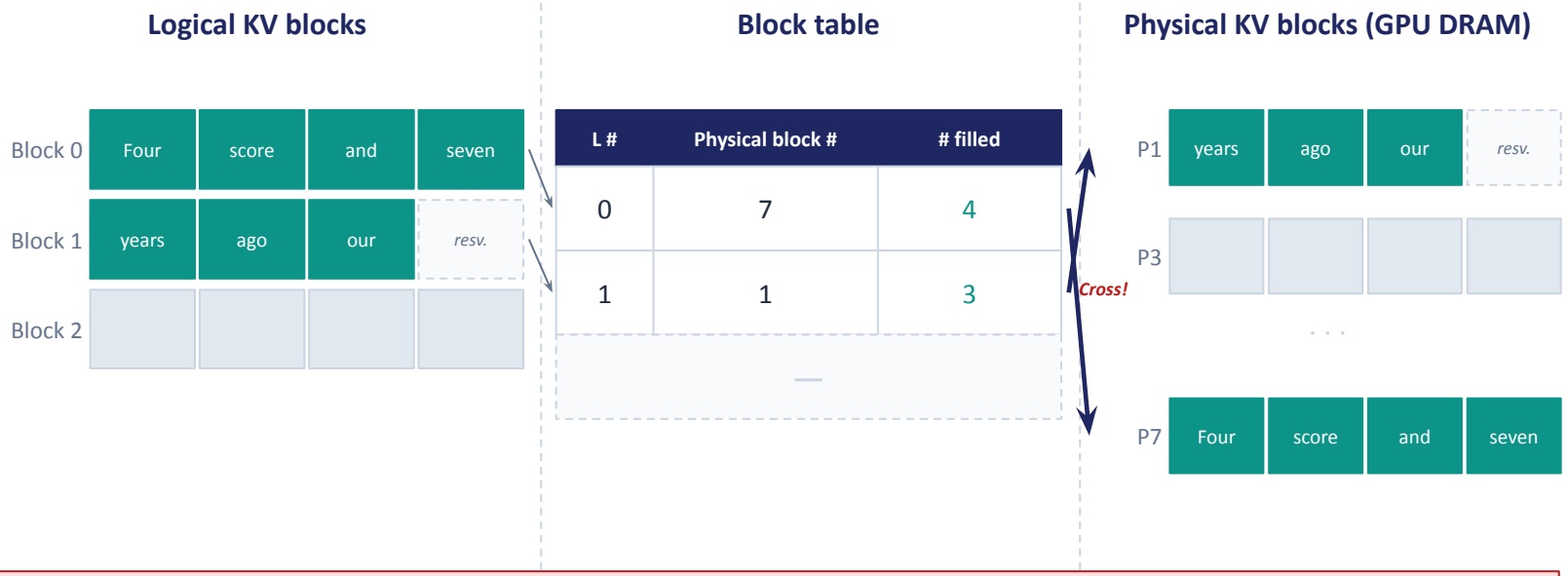
Step ① — Prefill: reserve only what the prompt needs



Step ① — Prefill: reserve only what the prompt needs



Step ① — Prefill: reserve only what the prompt needs

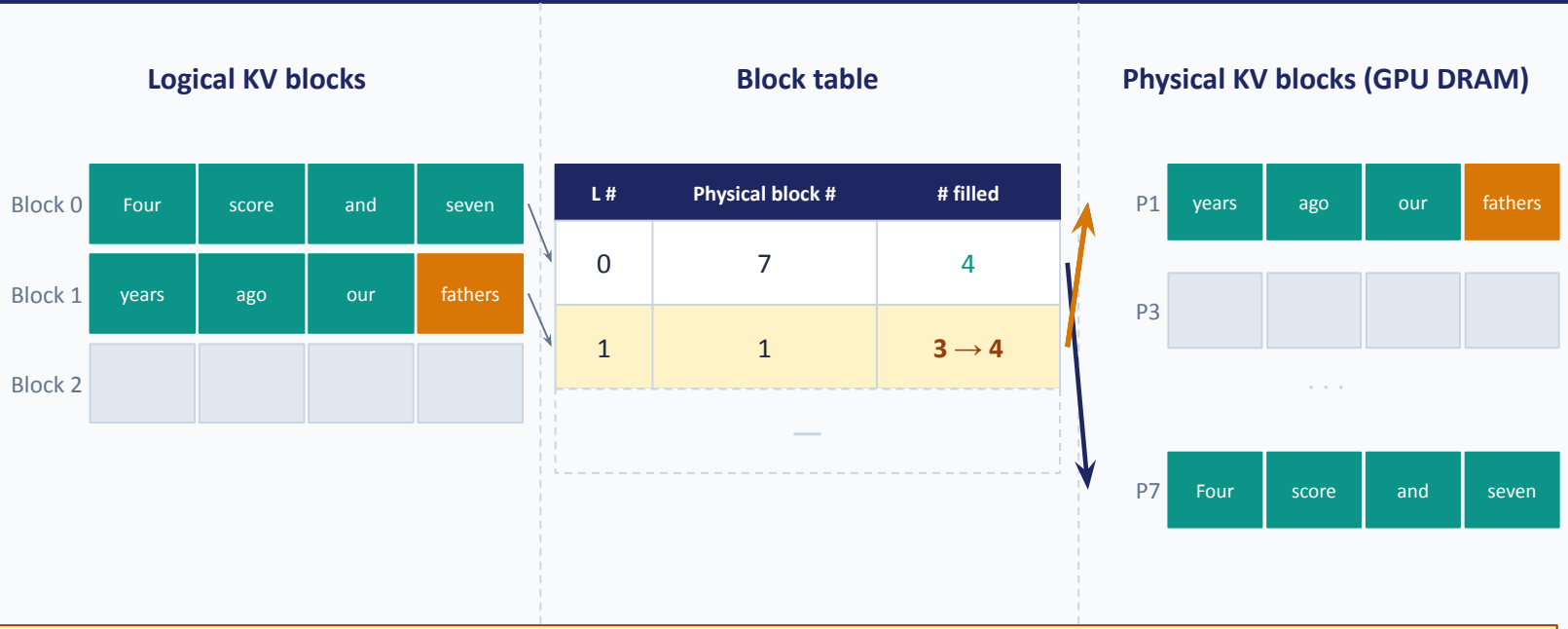


7 prompt tokens → only 2 blocks allocated (not max sequence length)

= prompt KV cache

= reserved (future use)

Step ② — First decode: “fathers” fills the last open slot



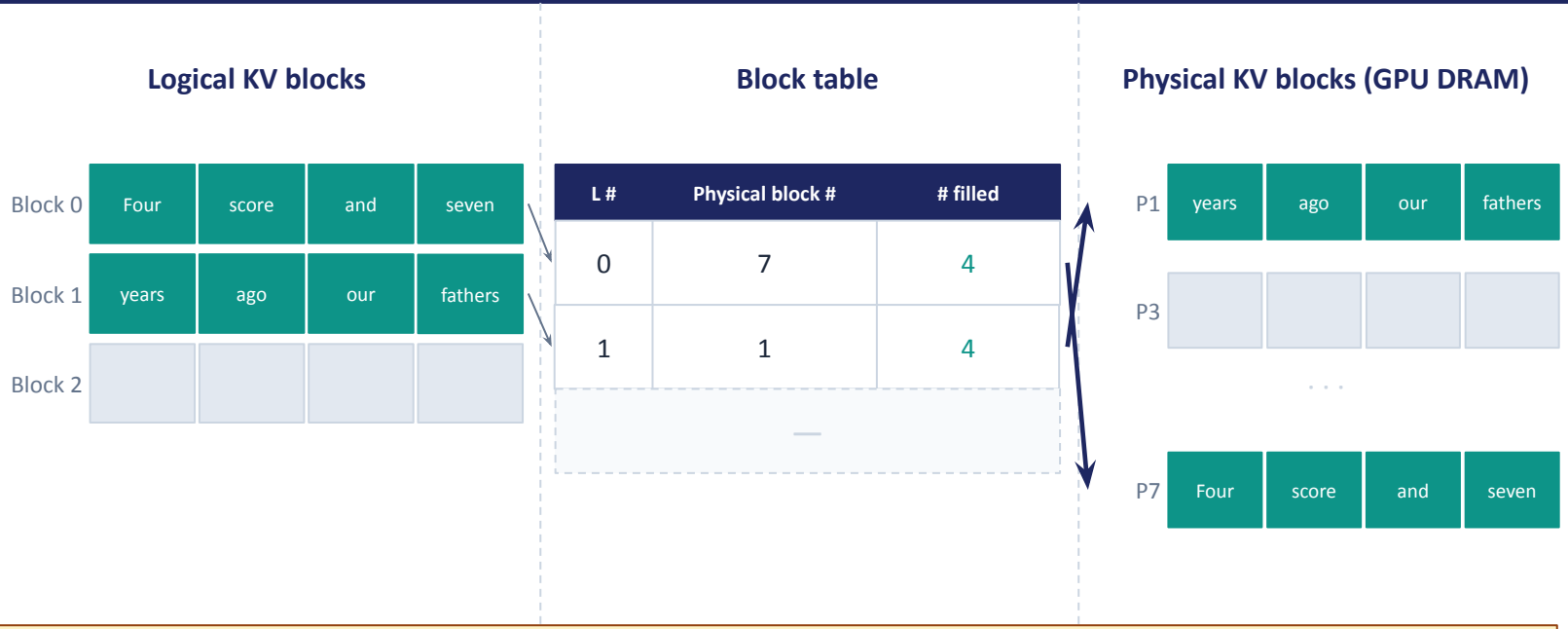
Reserved slot absorbs the new token — no allocation needed

= prompt KV cache

= new token (this step)

= reserved (future use)

Step ② — First decode: “fathers” fills the last open slot



Reserved slot absorbs the new token — no allocation needed

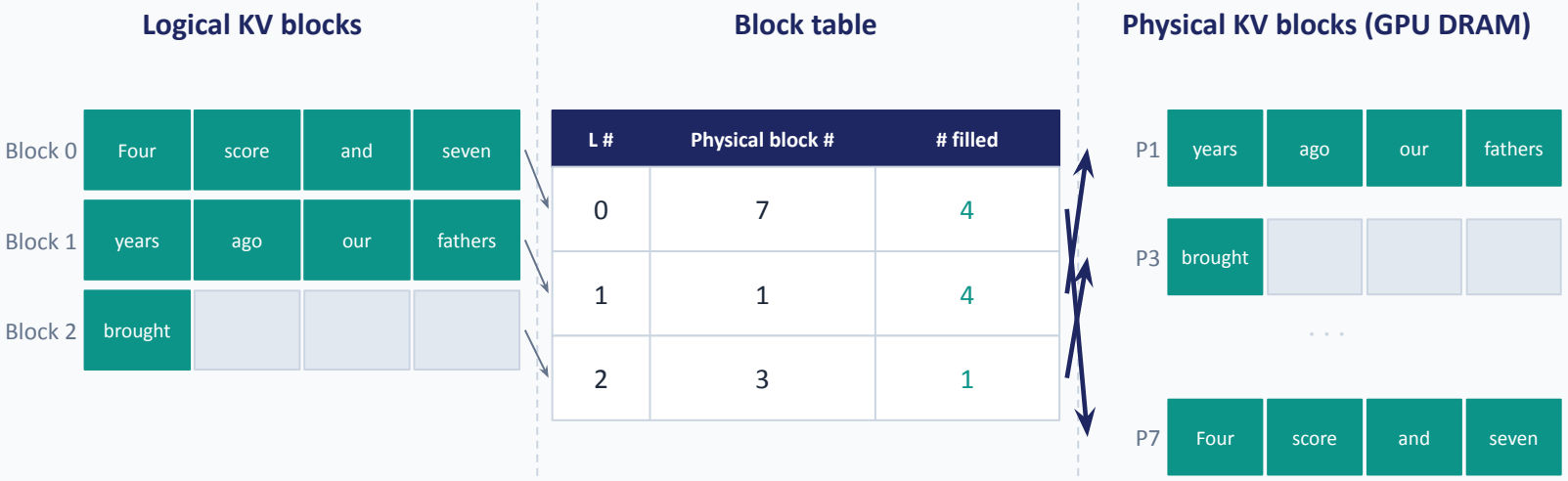
= prompt KV cache

= new token (this step)

= reserved (future use)

Step ③ — Second decode: block full → allocate on demand

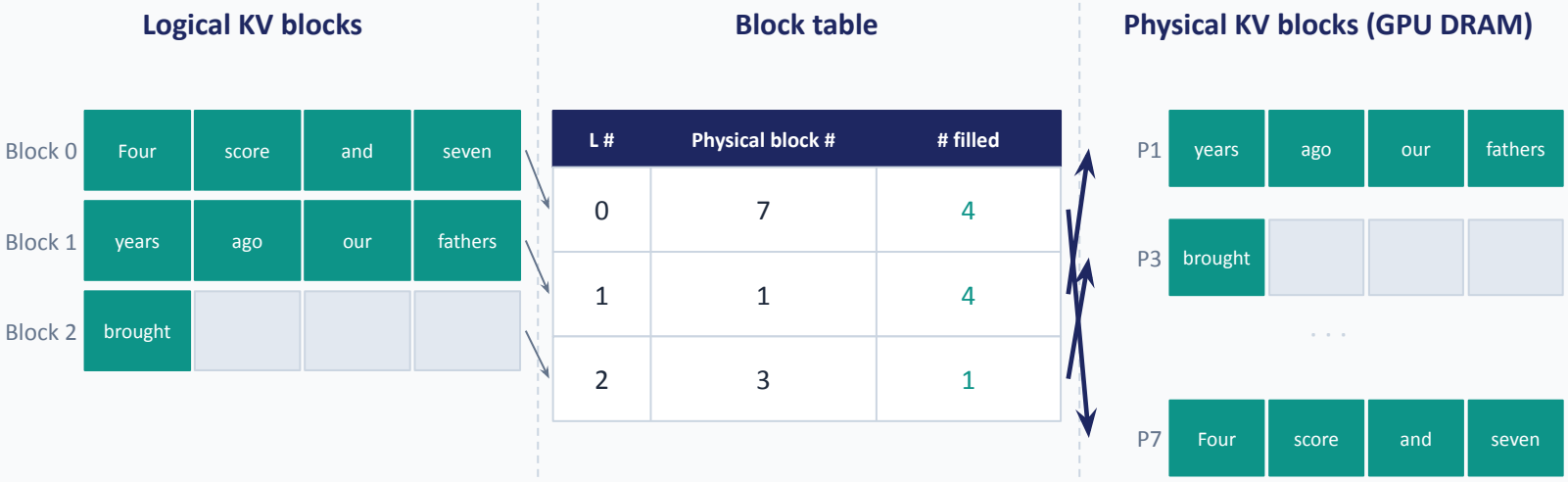
Key insight — Why does this design matter?



No pre-allocation

KV cache grows
block by block with the sequence

Key insight — Why does this design matter?

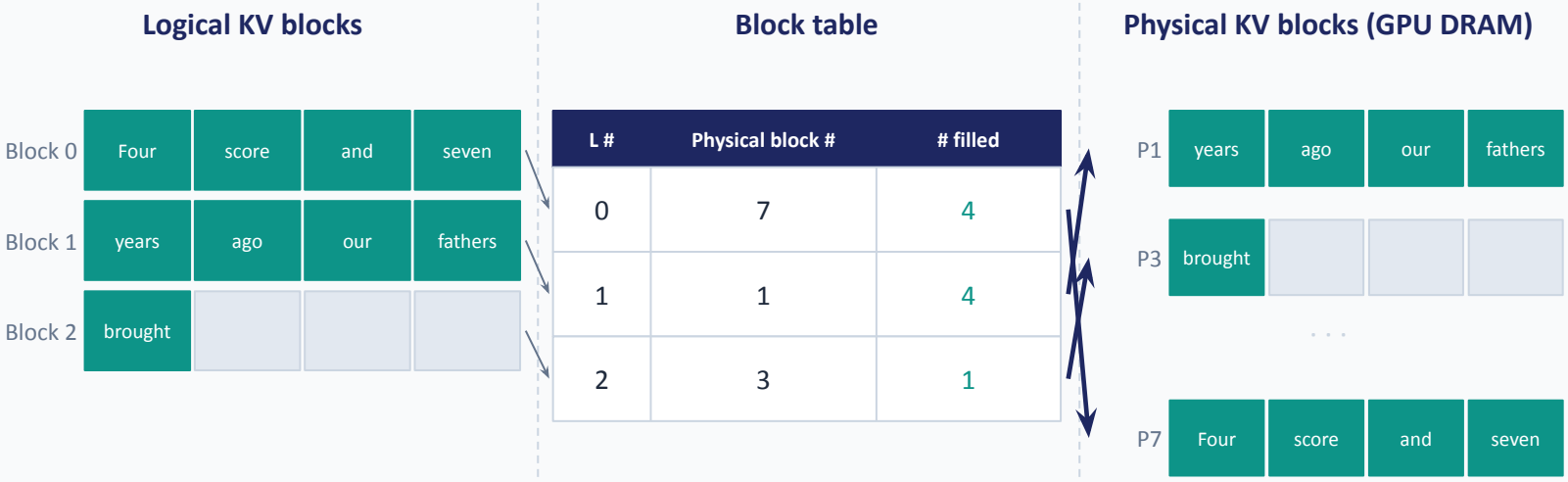


No pre-allocation

KV cache grows
block by block with the sequence

External fragmentation?

Key insight — Why does this design matter?



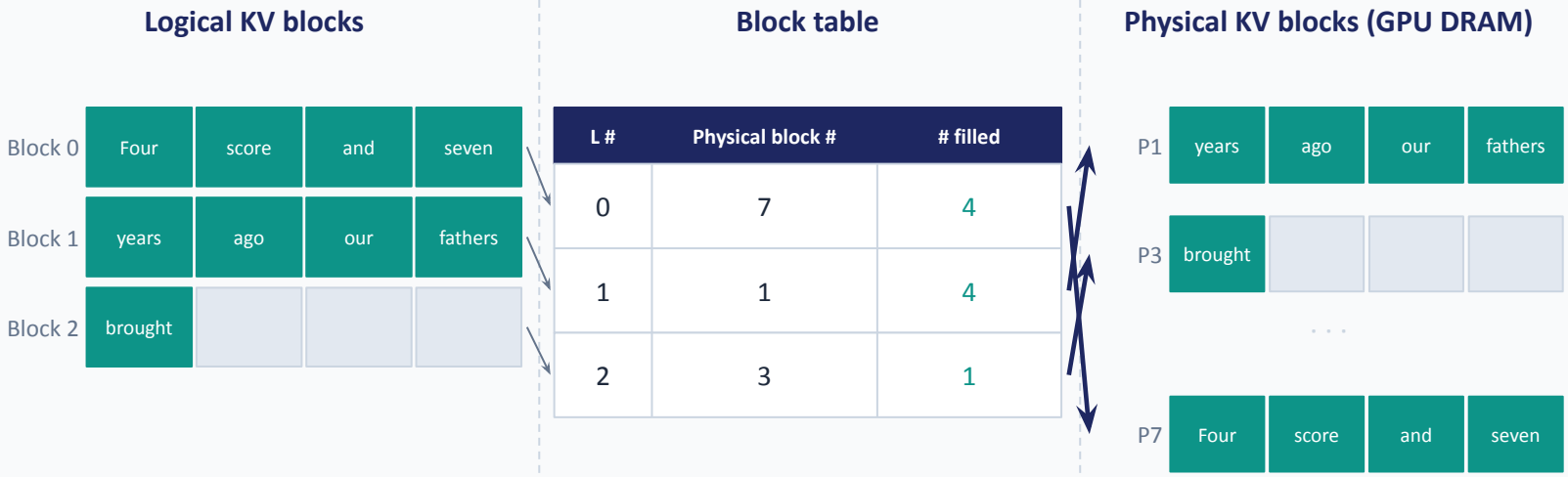
No pre-allocation

KV cache grows
block by block with the sequence

No external fragmentation

Any free physical block can be used for the next needed block

Key insight — Why does this design matter?



No pre-allocation

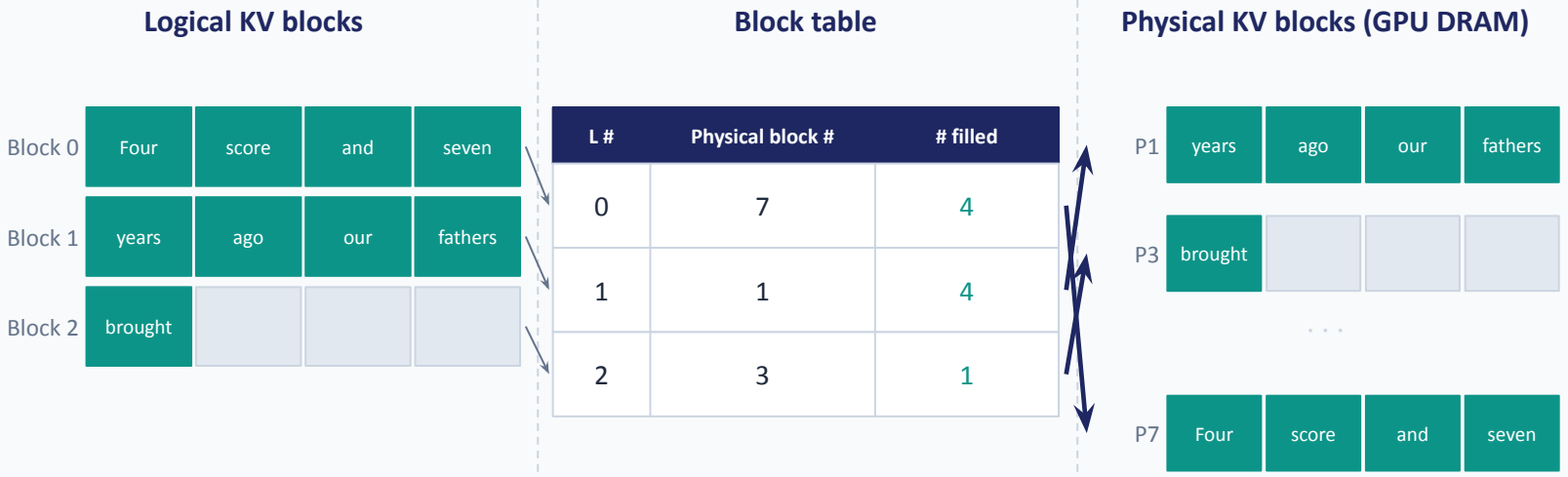
KV cache grows
block by block with the sequence

No external fragmentation

Any free physical block can be used for the next needed block

Request A has a partially filled last block
→ Request B uses a different physical block

Key insight — Why does this design matter?



No pre-allocation

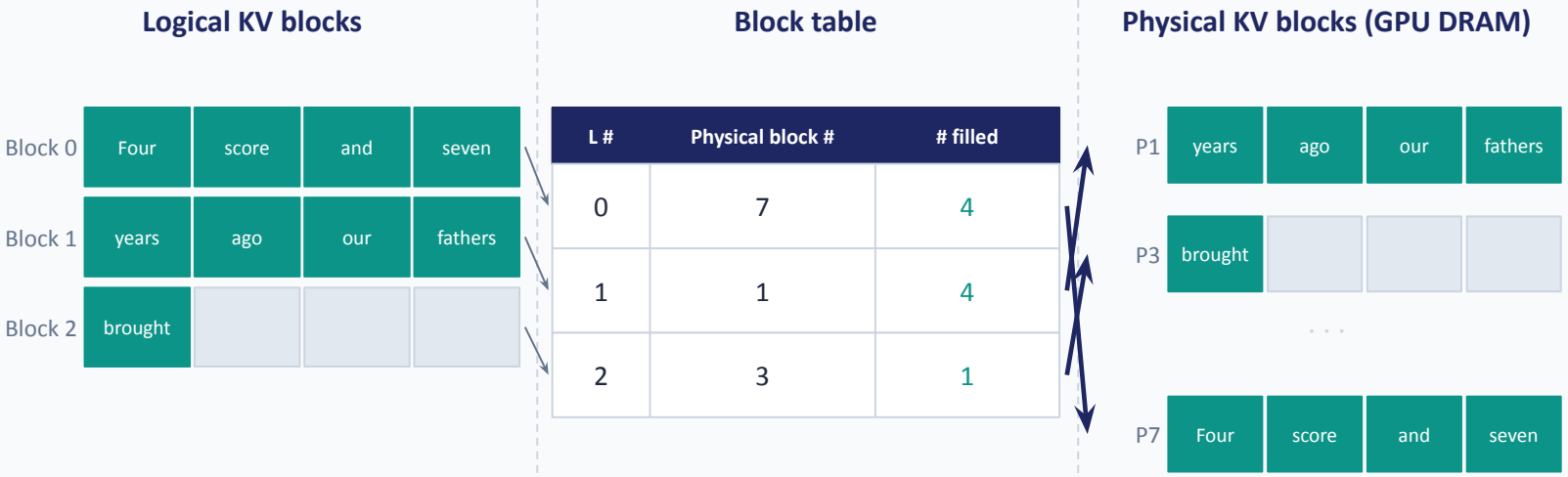
KV cache grows
block by block with the sequence

No external fragmentation

Any free physical block can be used for the next needed block

Internal fragmentation?

Key insight — Why does this design matter?



No pre-allocation

KV cache grows block by block with the sequence

No external fragmentation

Any free physical block can be used for the next needed block

Waste ≤ 1 block

Fill left-to-right; allocate a new block only when previous one is full

Parallel Sampling

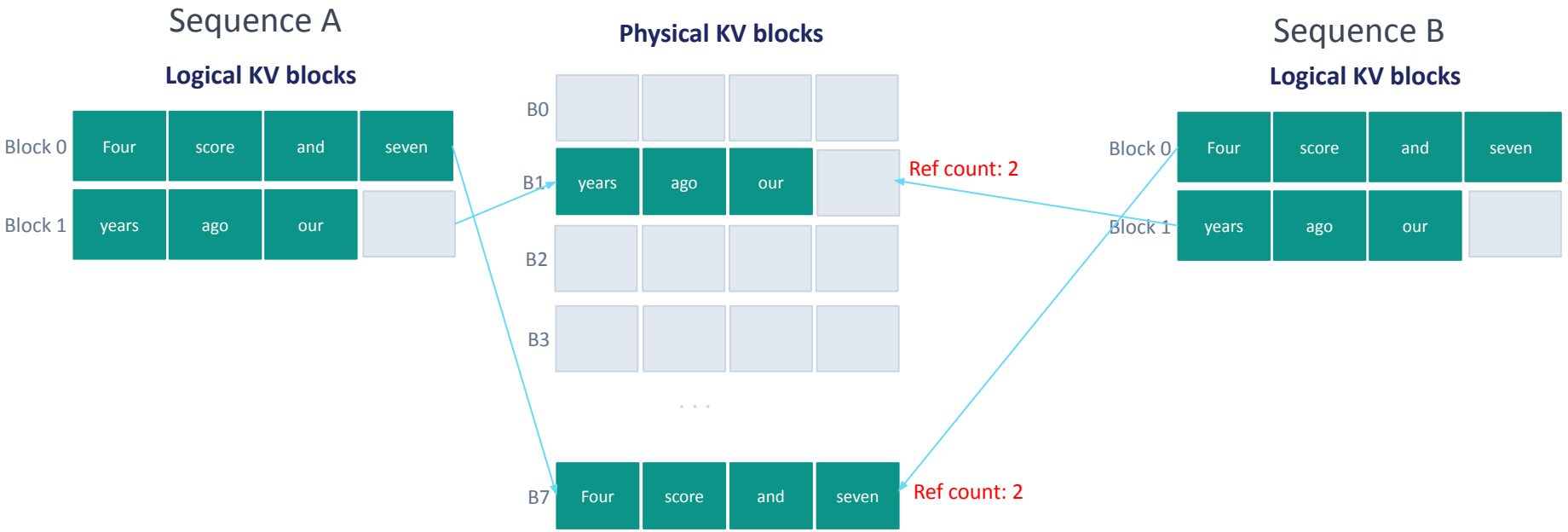
Parallel Sampling

One input prompt → Multiple outputs

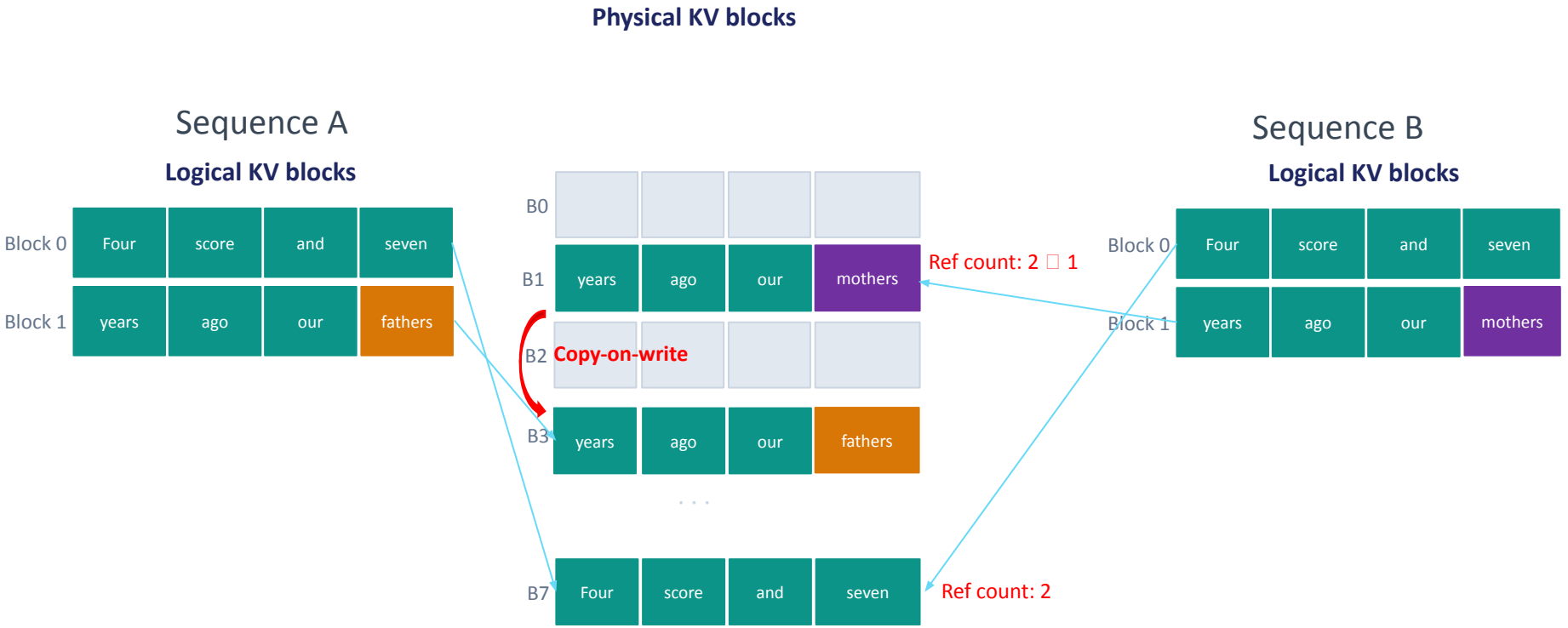
✓ Fixed allocation waste

Can the same design avoid duplication when one prompt produces multiple outputs?

Parallel Sampling



Parallel Sampling



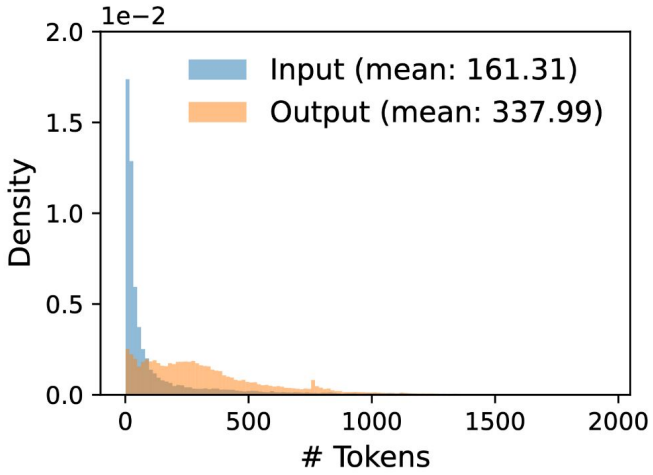
Swapping and Recomputation

Distributed Execution

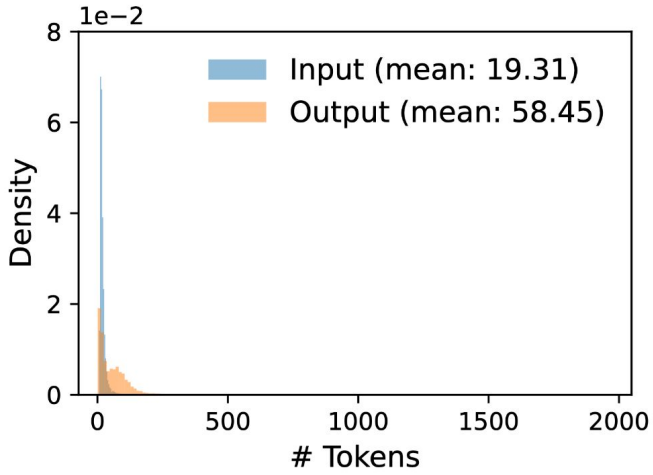
Evaluation

Datasets

Output and input length distributions on ShareGPU and Alpaca datasets



(a) ShareGPT



(b) Alpaca

Baselines and Server Configurations

1. **FasterTransformer**: Highly optimized transformer kernels from Nvidia. Authors added fixed batch size request-level scheduling.

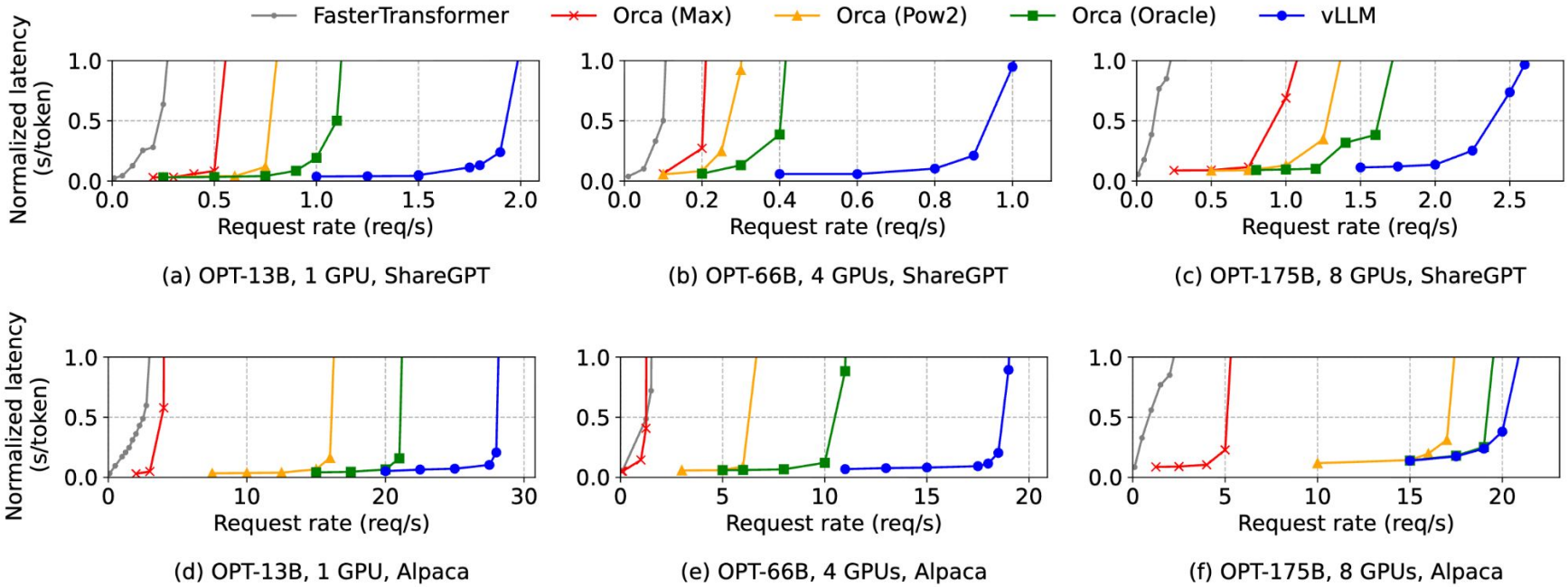
Iteration-Level Scheduling:

2. **Orca (Oracle)**: Allocates exactly enough memory for output length
3. **Orca (Pow2)**: Over-reserves memory by at most 2x
4. **Orca (Max)**: Reserves memory for the max possible memory up to the sequence length

Table 1. Model sizes and server configurations.

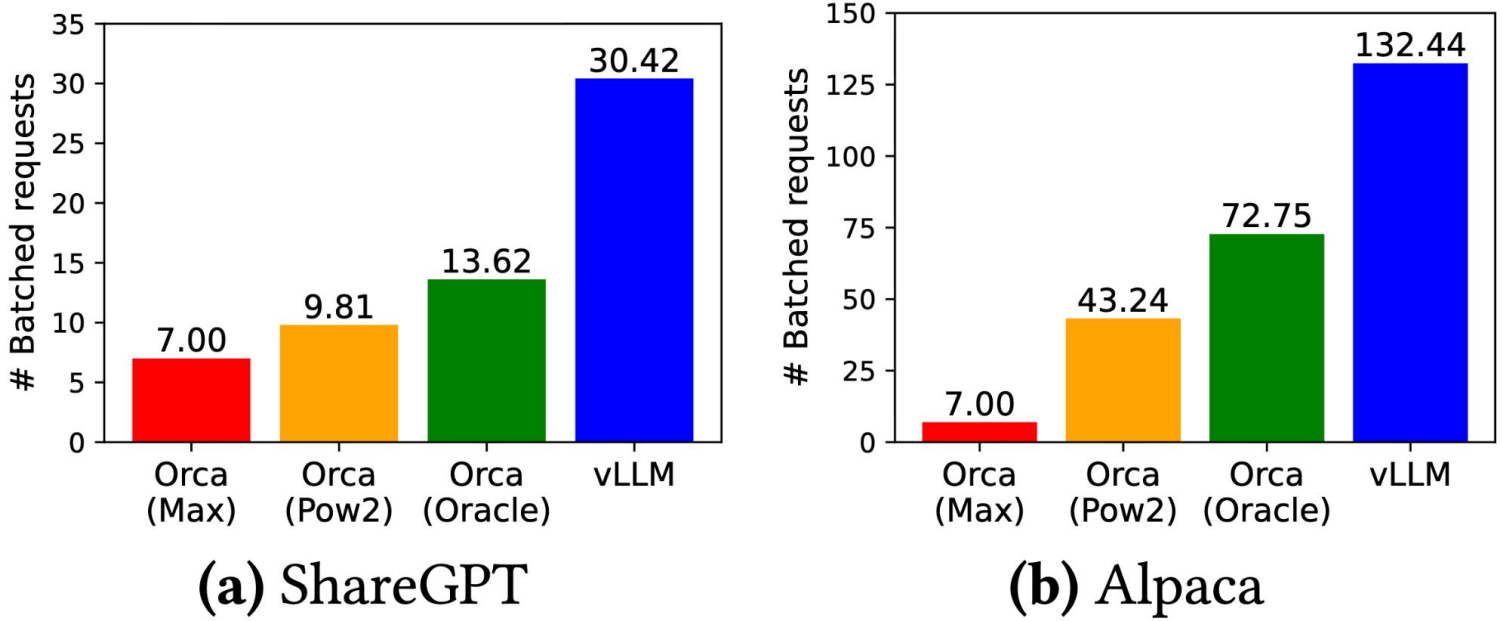
Model size	**13B**	**66B**	**175B**
GPUs	A100	4×A100	8×A100-80GB
Total GPU memory	40 GB	160 GB	640 GB
Parameter size	26 GB	132 GB	346 GB
Memory for KV cache	12 GB	21 GB	264 GB
Max. # KV cache slots	15.7K	9.7K	60.1K

How could vLLM outperform Orca (Oracle) if Orca (Oracle) perfectly allocates the right amount of memory?



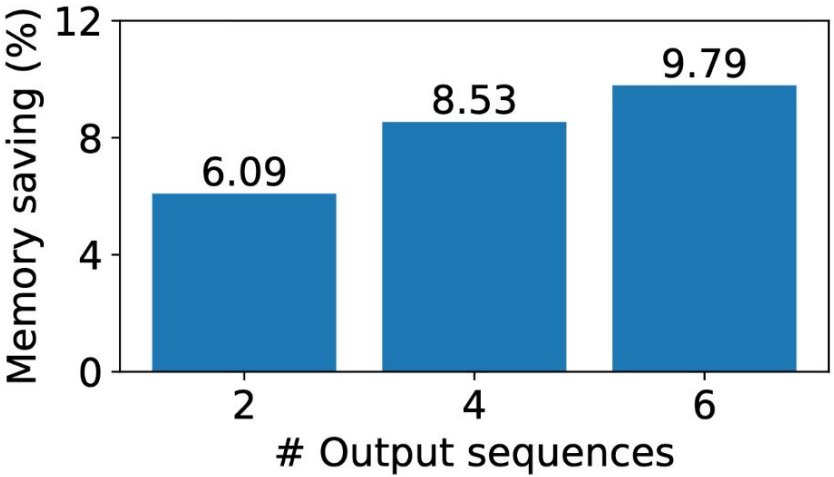
Why do we see the sharp upward trend in token latency as request rate increases?

Batch Sizes

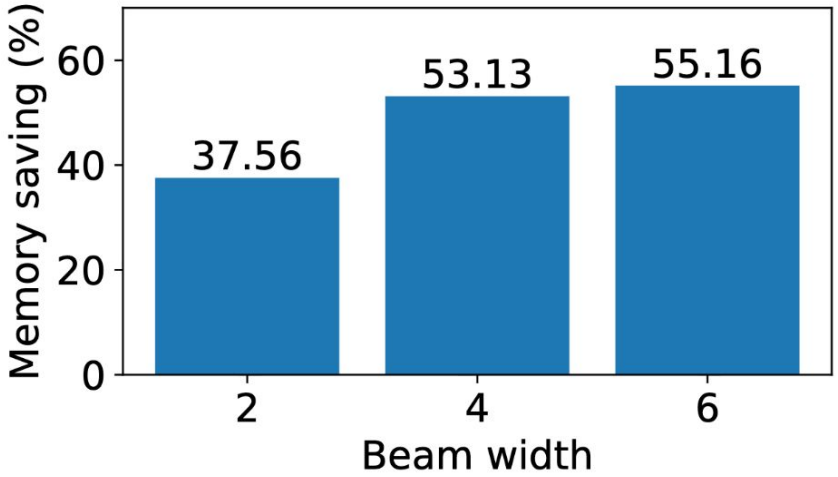


Average number of batched requests when serving OPT-13B for the ShareGPT (2 reqs/s) and Alpaca (30 reqs/s) traces

KV-Cache Page Reuse

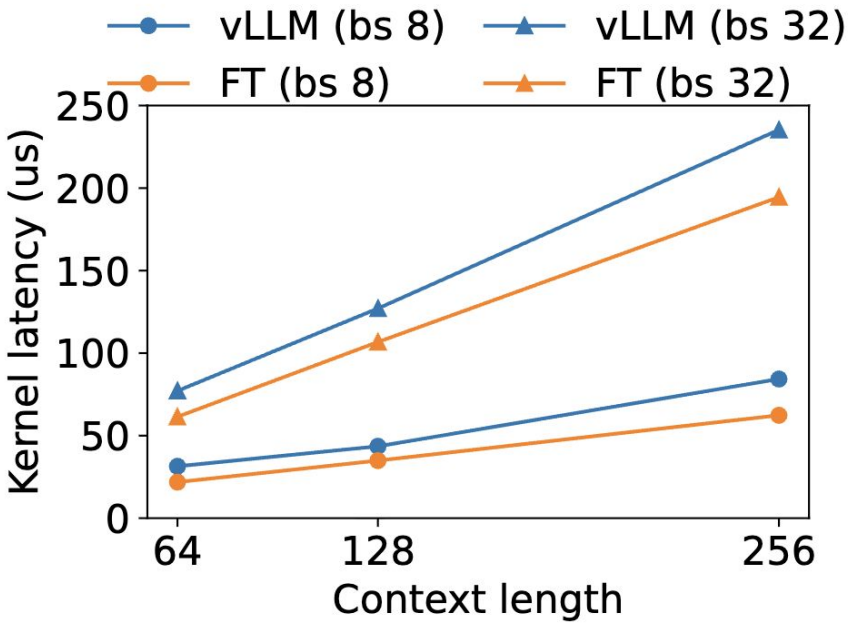


(a) Parallel sampling



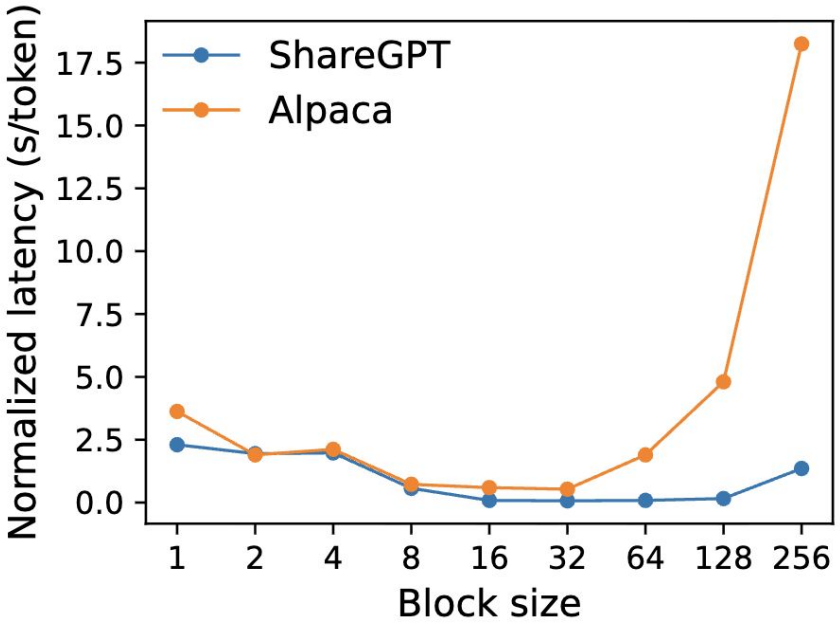
(b) Beam search

Paged Attention Kernel Benchmark



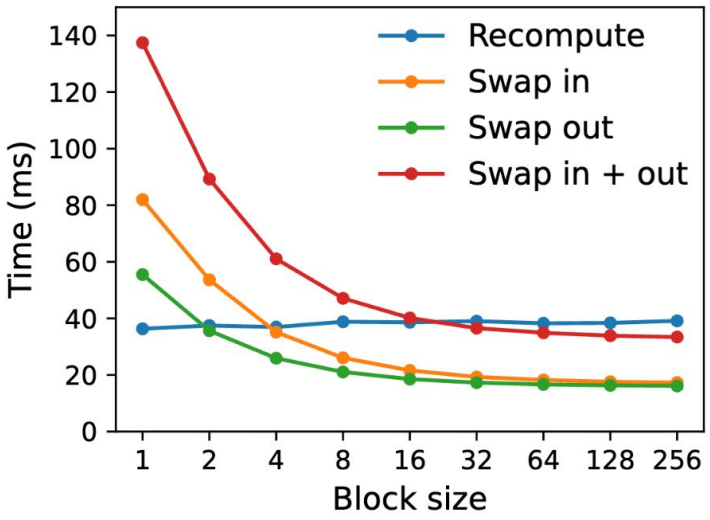
20–26% higher attention kernel latency due to paging compared to FasterTransformer (FT)

Ablation Study: Block Size

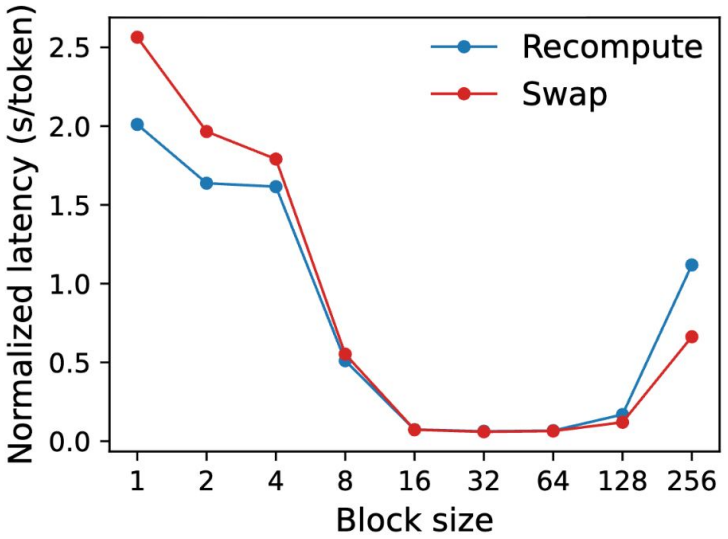


Why might very small or very large block sizes reduce performance?

Recomputation/Swap vs. Block Size



Overhead



End to End Performance

Where do we see more overhead with smaller block sizes? Why?

Thank you!