[COMSE6998-015] Fall 2024 Introduction to Deep Learning and LLM based Generative AI Systems

Lecture 12

Parijat Dube, Chen Wang

Agenda: Efficient Serving of LLMs

Batching Techniques

- Static Batching, Dynamic Batching and Continuous Batching

Memory Optimization Techniques

- KV Cache
- Flash Attention
- PagedAttention

LLM Serving Frameworks

- vLLM

Serving Systems for Fine-tuning Models

- sLoRA, vLLM

GPU Sharing Techniques

- Space vs Time Partitioning
- Software vs Hardware
- LLM inference on MIGs

The era of Large Language Models (LLMs)

ChatGPT

GitHub Copilot

Chatting

LLM

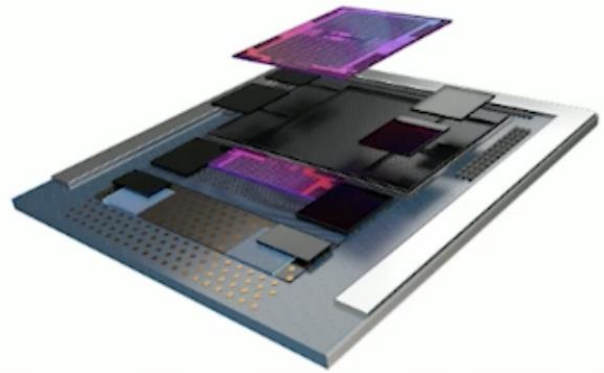
Programming

BizOps

Content creation

Dev tools

Served by GPUs

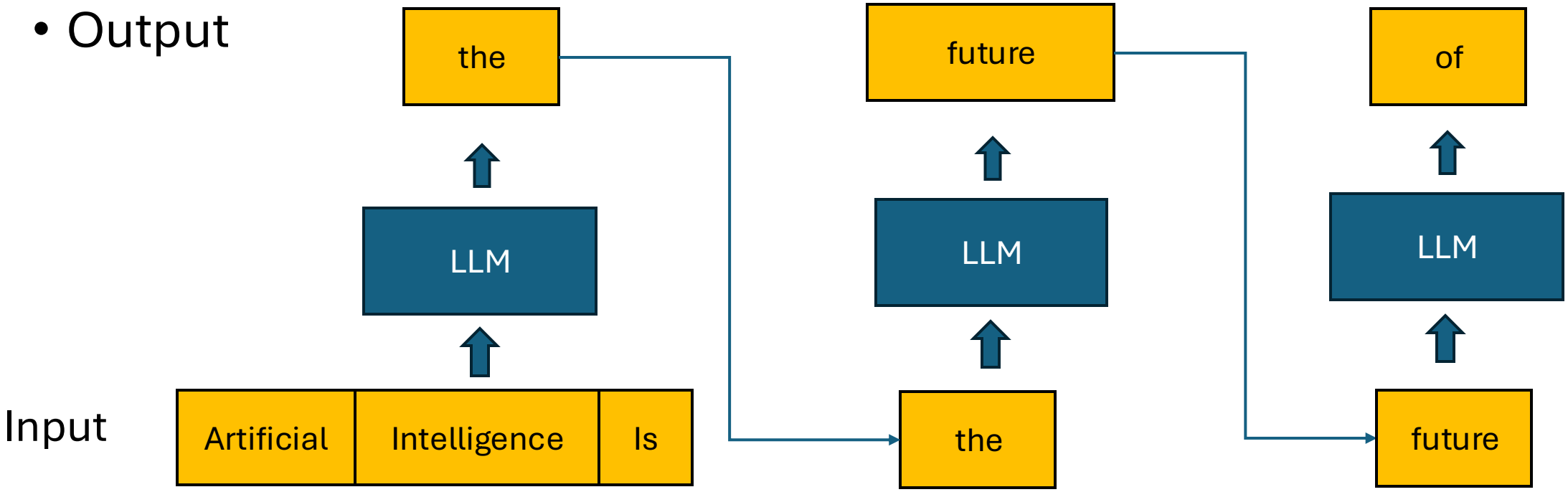


Serving LLMs is (surprisingly) slow and expensive

- A single A100 GPU can only serve <1 request per second
  - Moderate size of model (13 Billion parameters) and input
- A ton of GPUs are required for production-scale LLM services

Inference process of LLMs

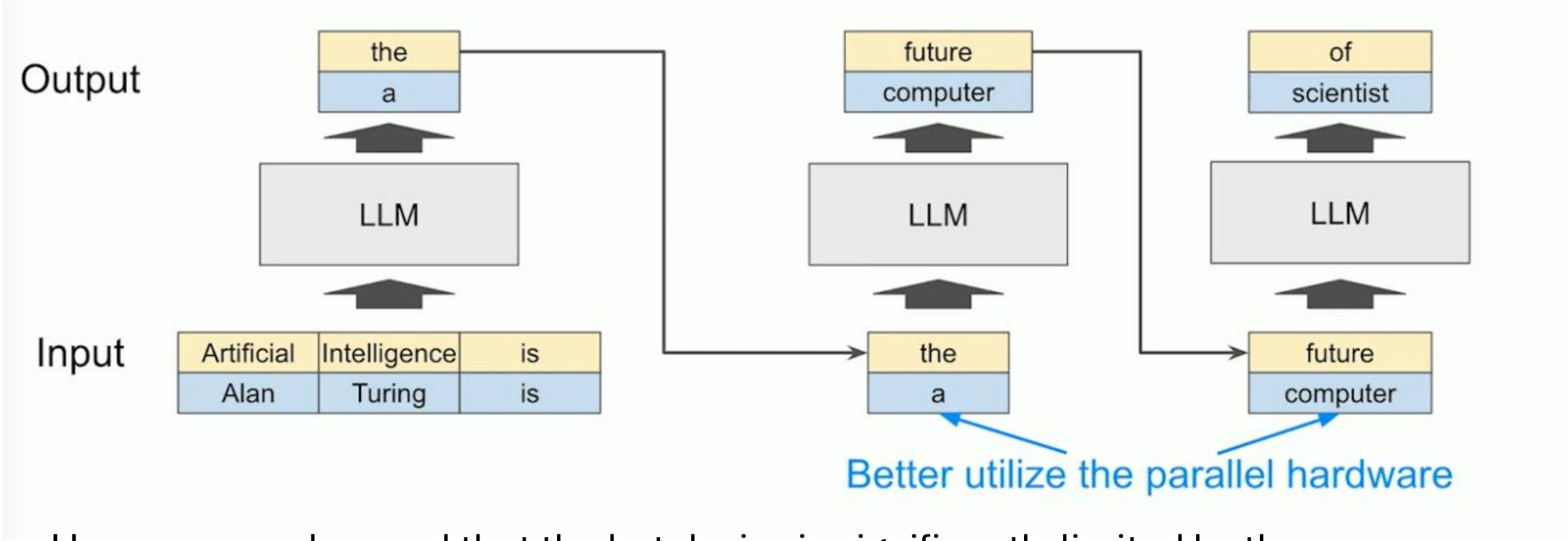
- Output



Why is it **slow** and **inefficient**?

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

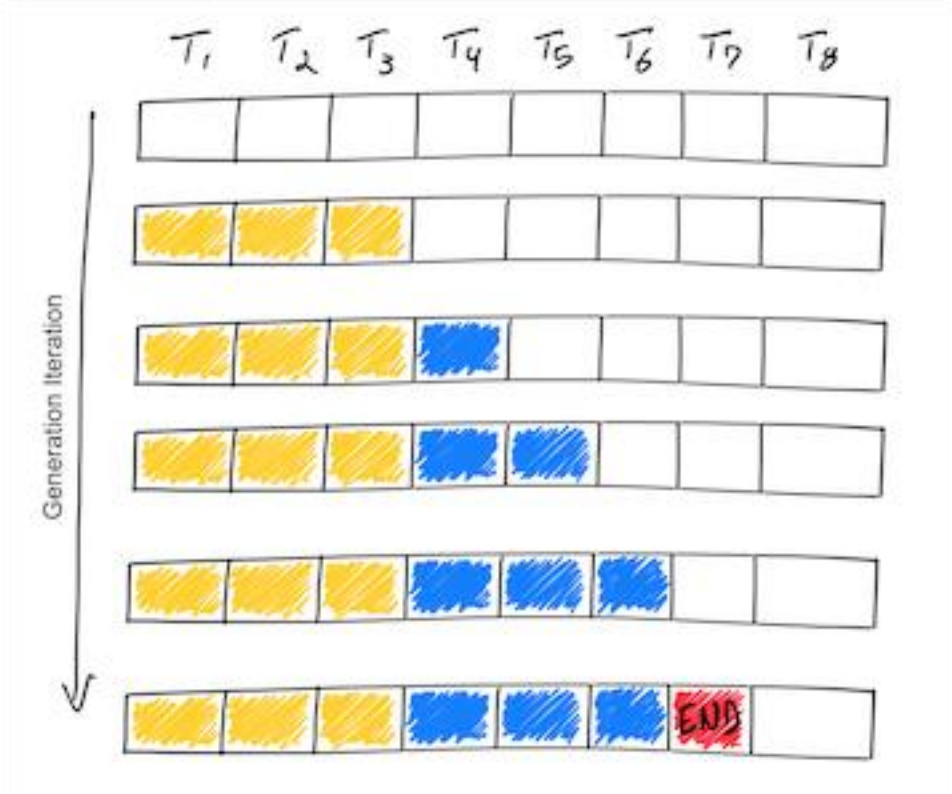
Solution: Batching multiple requests together



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

The basics of LLM inference

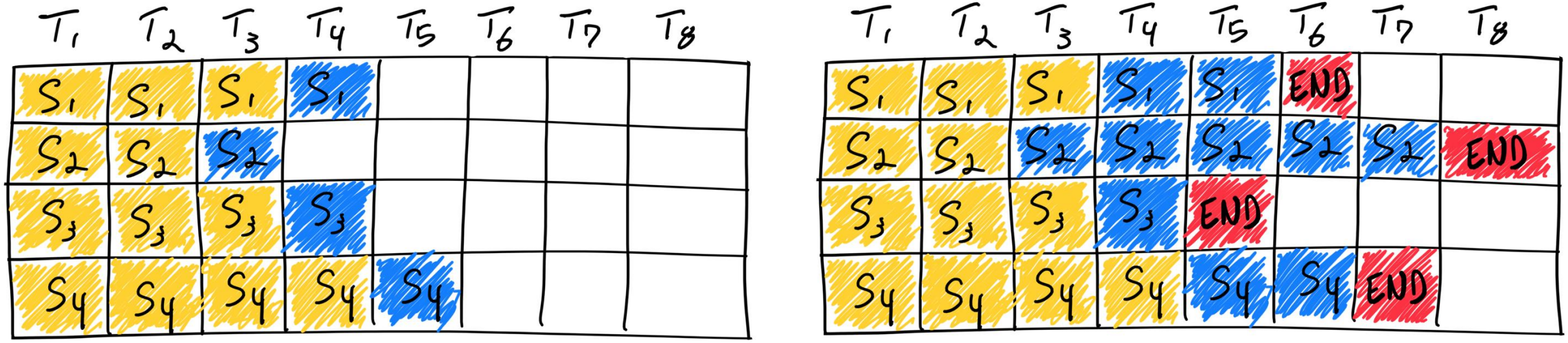
- Basic Process
  - LLM inference is an iterative process that generates one token at a time



- Three Critical Aspects
  - Prefill Phase
    - Initial prompt processing takes significant time
    - Pre-computes attention inputs that remain constant during generation
    - Uses GPU's parallel compute efficiently
  - Memory-IO Bound
    - Loading data to GPU cores takes longer than computation
    - Throughput depends on batch size fitting in GPU memory
    - Memory efficiency is crucial for performance2
  - Memory Scaling
    - GPU memory usage scales with model size + sequence length
    - A 13B parameter model uses ~1MB per token
    - On 40GB A100 GPU: 26GB for model parameters, 14GB for processing

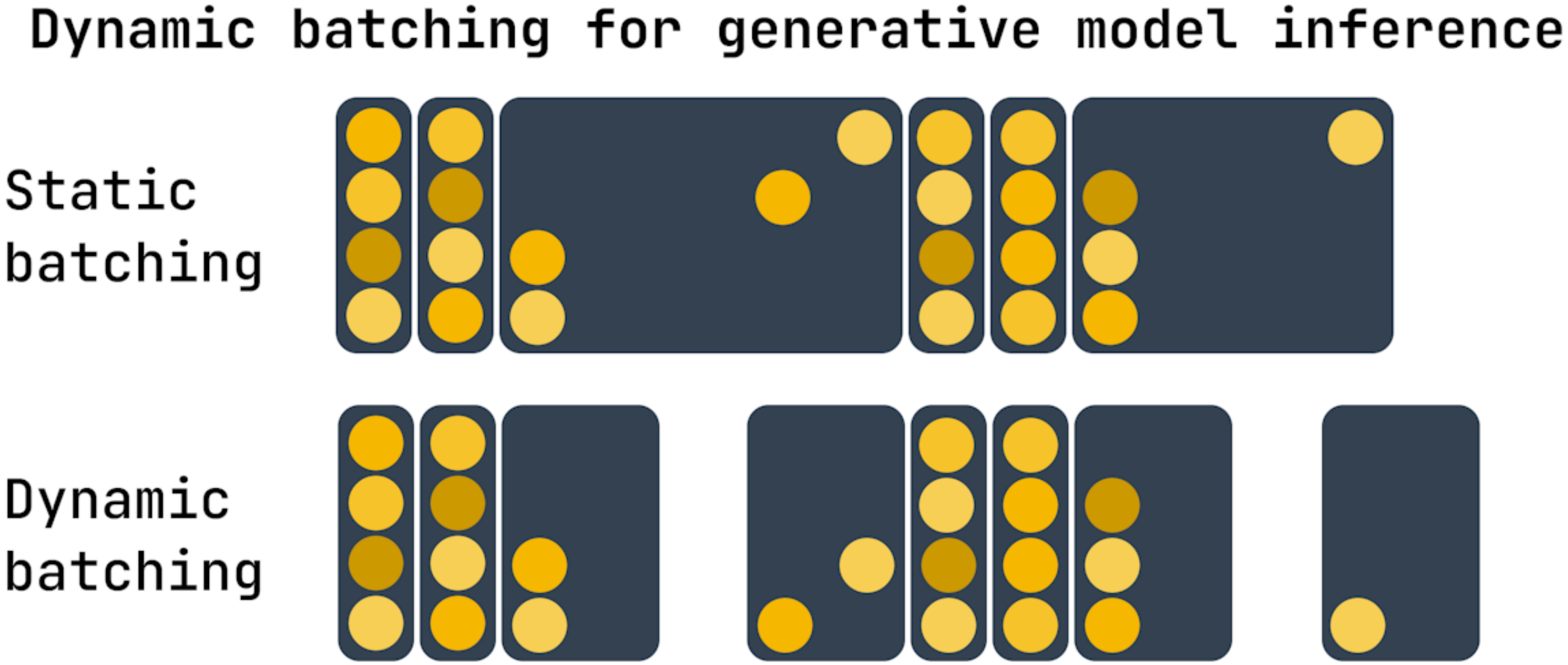
https://github.com/ray-project/llm-numbers#1-mb-gpu-memory-required-for-1-token-of-output-with-a-13b-parameter-model

Naïve batching / static batching

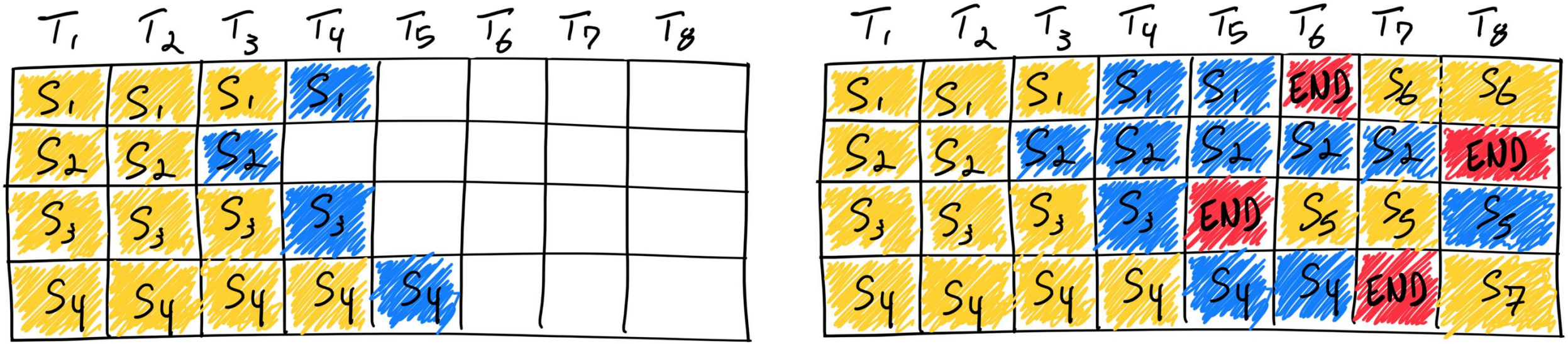


Completing four sequences using static batching. On the first iteration (left), each sequence generates one token (blue) from the prompt tokens (yellow). After several iterations (right), the completed sequences each have different sizes because each emits their end-of-sequence-token (red) at different iterations. Even though sequence 3 finished after two iterations, static batching means that the GPU will be underutilized until the last sequence in the batch finishes generation (in this example, sequence 2 after six iterations).

Static Batching vs Dynamic Batching



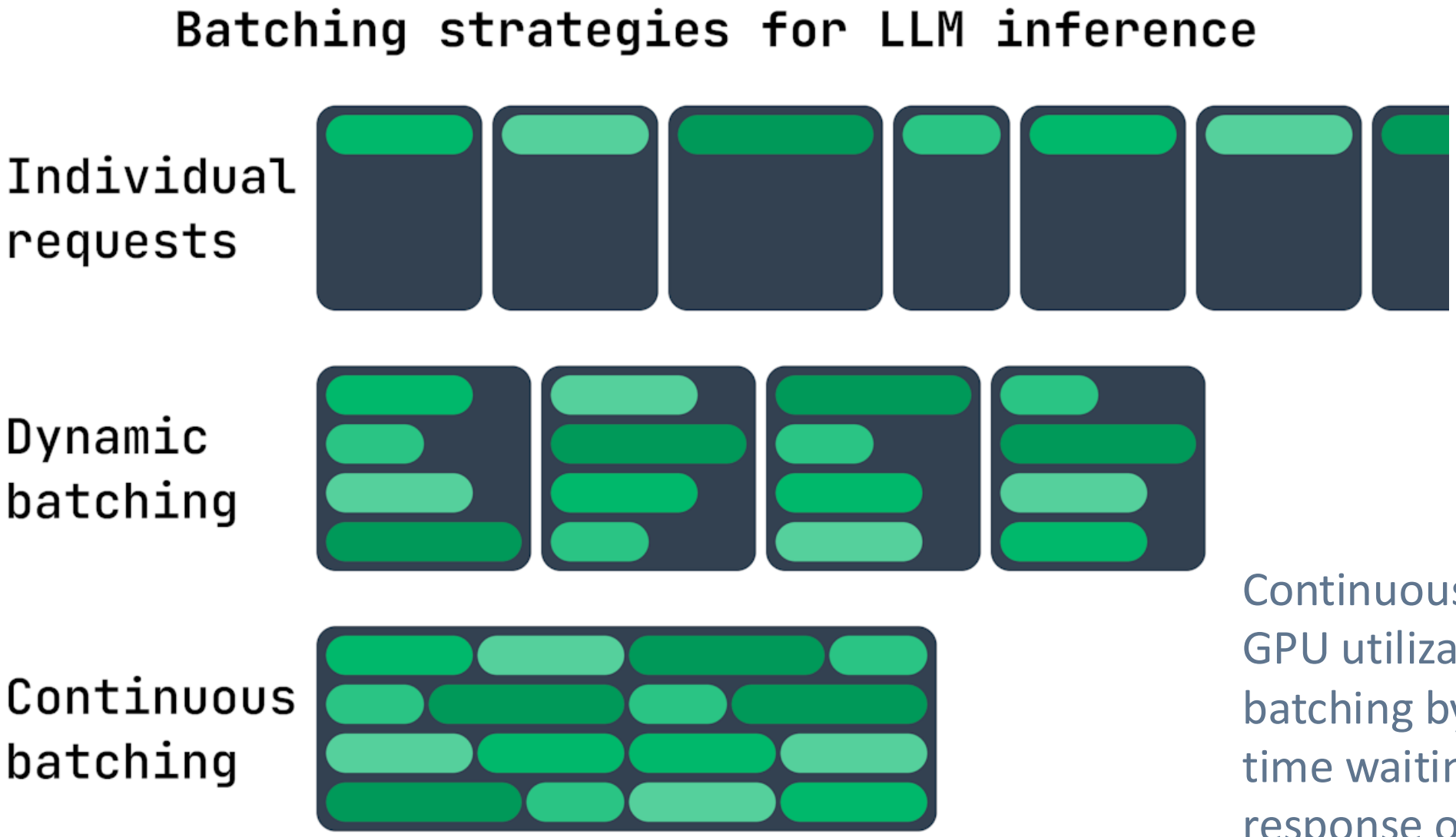
Continuous Batching



Completing seven sequences using continuous batching. Left shows the batch after a single iteration, right shows the batch after several iterations. Once a sequence emits an end-of-sequence token, we insert a new sequence in its place (i.e. sequences S5, S6, and S7). This achieves higher GPU utilization since the GPU does not wait for all sequences to complete before starting a new one.

Yu, Gyeong-In, et al. "Orca: A distributed serving system for {Transformer-Based} generative models." *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 2022.

Continuous Batching



Continuous batching improves GPU utilization over dynamic batching by eliminating the idle time waiting for the longest response of each batch to finish.

Types of LLM Inference Batching

Static Batching

- Waits for a fixed number of requests to accumulate before processing
- Processes all requests simultaneously as a single batch
- Best suited for offline workloads where latency isn't critical
- Like a bus that only departs when completely full

Dynamic Batching

- Uses both maximum batch size and time window parameters
- Processes requests when either batch is full or time window expires
- Balances between throughput and latency
- Ideal for consistent-length outputs like image generation

Continuous Batching

- Processes requests token-by-token at the sequence level
- Dynamically allocates GPU resources as tokens are generated
- Efficiently handles varying output lengths
- Optimizes model weight loading across multiple requests

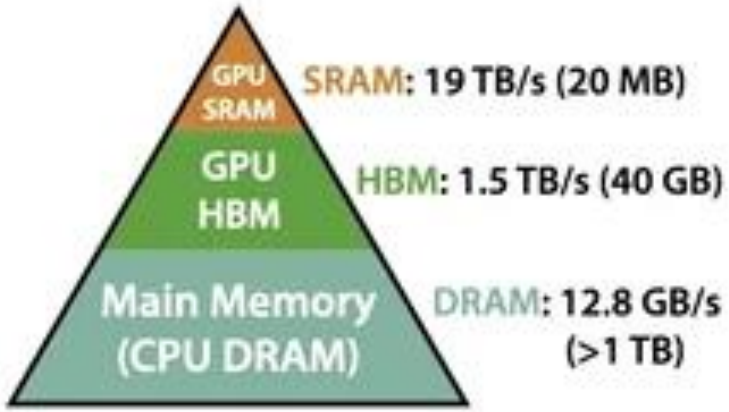
Performance Characteristics of Batching

Batching Type	Throughput	Latency	Use Case
Static	Highest	High	Offline processing, daily jobs
Dynamic	Medium	Medium	Real-time image generation
Continuous	High	Low	Interactive LLM applications

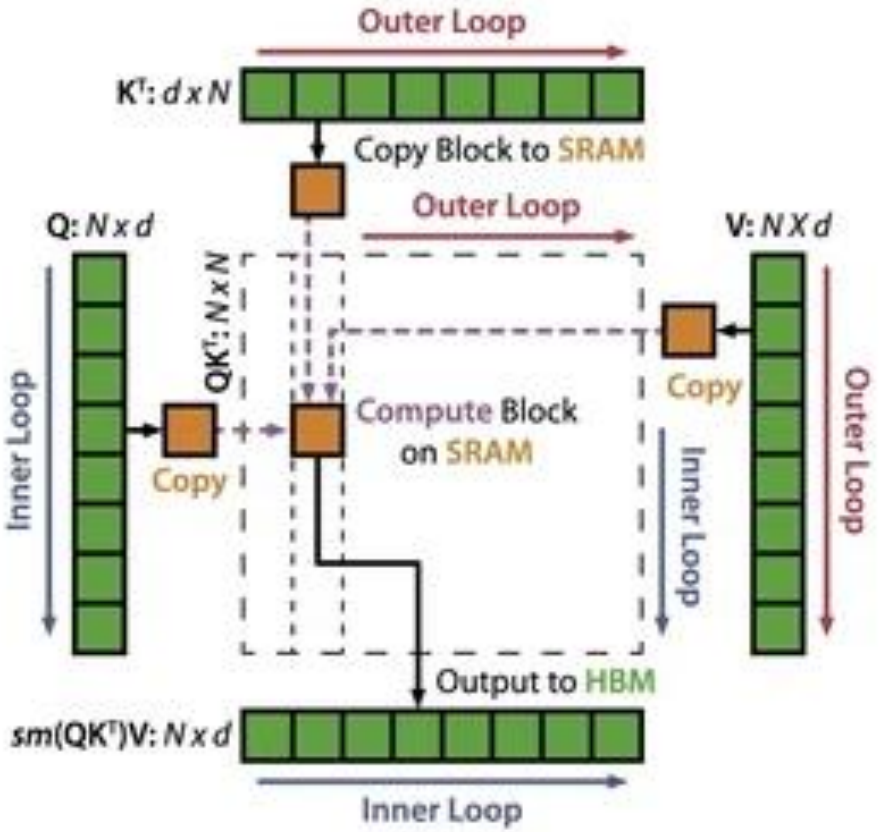
FlashAttention: Fast and Memory Efficient Exact Attention with IO-Awareness

FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness
Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, Christopher Ré
Paper: https://arxiv.org/abs/2205.14135

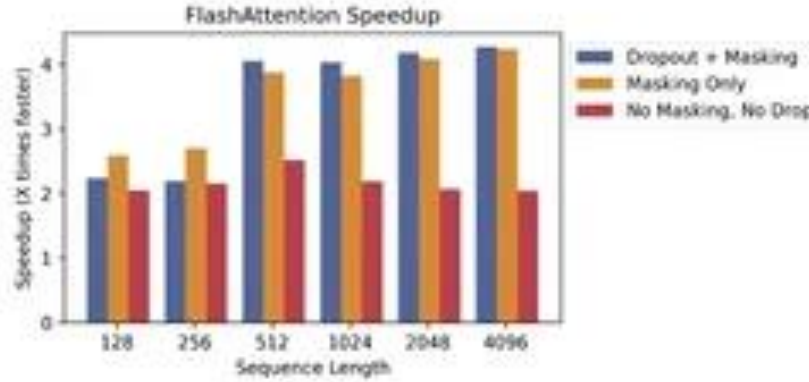
Attention: IO-Aware Tiling Yields Speed, Mem Savings, & Quality

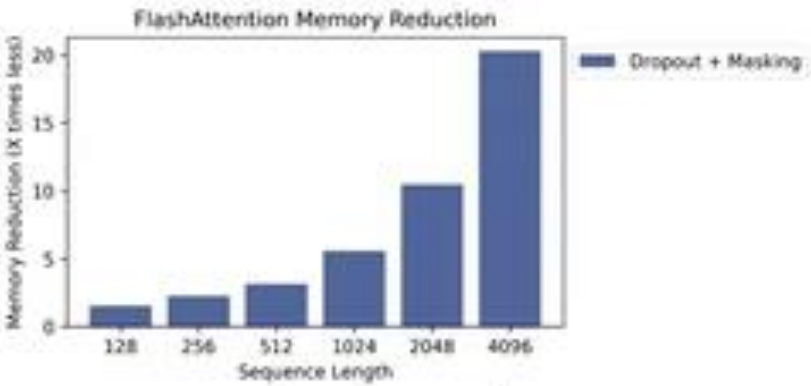


Memory Hierarchy with Bandwidth & Memory Size



FlashAttention





Model	Path-X	Path-256
Transformer	✗	✗
Linformer [81]	✗	✗
Linear Attention [48]	✗	✗
Performer [11]	✗	✗
Local Attention [77]	✗	✗
Reformer [49]	✗	✗
SMYRF [18]	✗	✗
FlashAttention	**61.4**	✗
Block-sparse FlashAttention	56.0	**63.1**

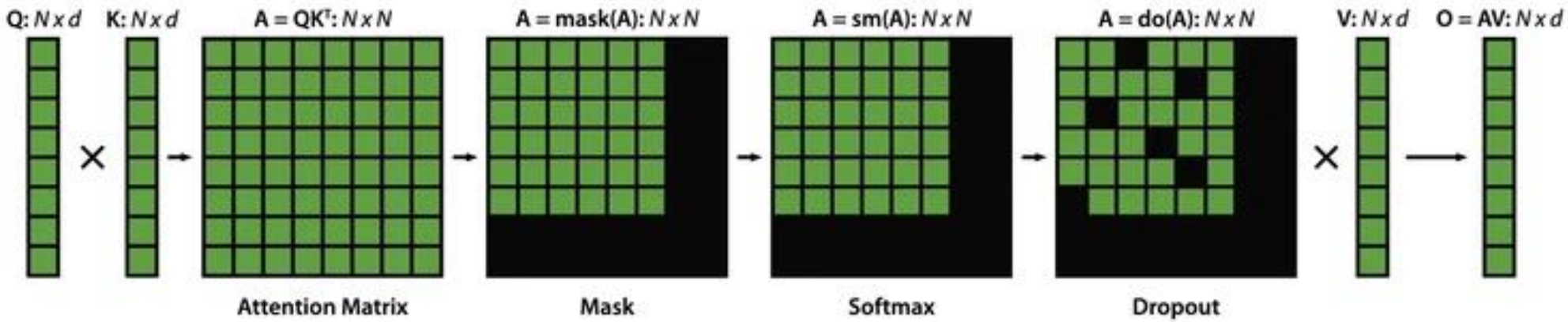
Motivation: Modeling Longer Sequences

Bridging new capabilities	Closing the reality gap	Opening new areas
NLP: Large context required to understand books, plays, instruction manuals.	Computer vision: higher resolution can lead to better, more robust insight.	Time series, audio, video, medical imaging data naturally modeled as sequences of millions of steps.

Challenge: How to scale Transformers to longer sequences?

Blog post: Can Longer Sequences Help Take the Next Leap in AI? https://ai.stanford.edu/blog/longer-sequences-next-leap-ai/

Background: Attention is Bottlenecked by Memory Reads/Writes



$$\mathbf{O} = \text{Dropout}(\text{Softmax}(\text{Mask}(\mathbf{QK}^T))\mathbf{V}$$

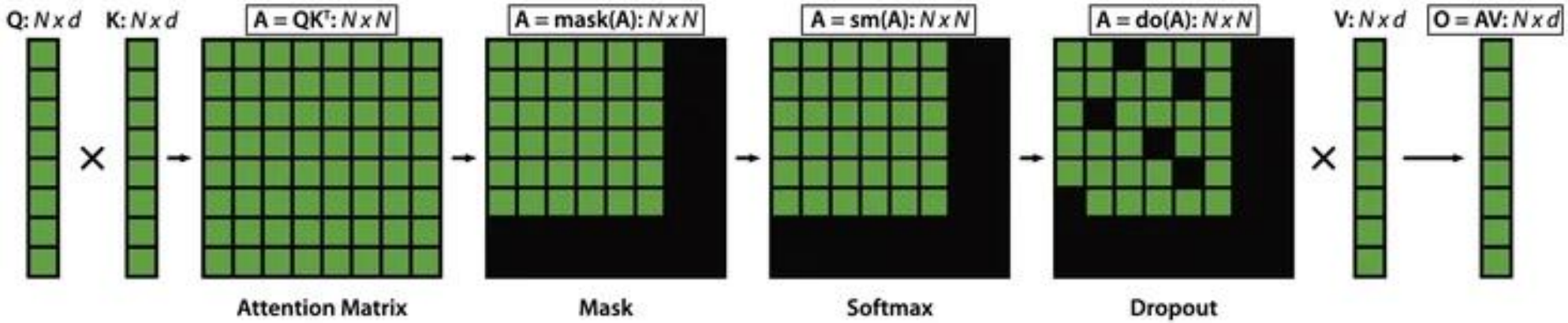
DATA MOVEMENT IS ALL YOU NEED: A CASE STUDY ON OPTIMIZING TRANSFORMERS

Andrei Ivanov [*1] **Nikoli Dryden** [*1] **Tal Ben-Nun** [1] **Shigang Li** [1] **Torsten Hoefler** [1]

ABSTRACT

Transformers are one of the most important machine learning workloads today. Training one is a very compute-intensive task, often taking days or weeks, and significant attention has been given to optimizing transformers. Despite this, existing implementations do not efficiently utilize GPUs. We find that data movement is the key bottleneck when training. Due to Amdahl's Law and massive improvements in compute performance, training has now become memory-bound. Further, existing frameworks use suboptimal data layouts. Using these insights, we present a recipe for globally optimizing data movement in transformers. We reduce data movement by up to 22.91% and overall achieve a 1.30× performance improvement over state-of-the-art frameworks when training a BERT encoder layer and 1.19× for the entire BERT. Our approach is applicable more broadly to optimizing deep neural networks, and offers insight into how to tackle emerging performance bottlenecks.

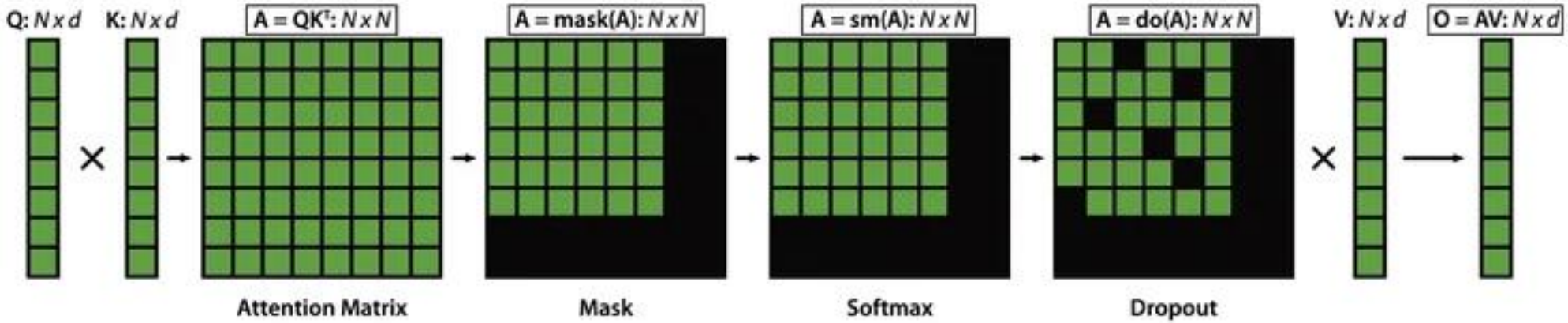
Background: Attention is Bottlenecked by Memory Reads/Writes



$$\mathbf{O} = \text{Dropout}(\text{Softmax}(\text{Mask}(\mathbf{QK}^T))\mathbf{V}$$

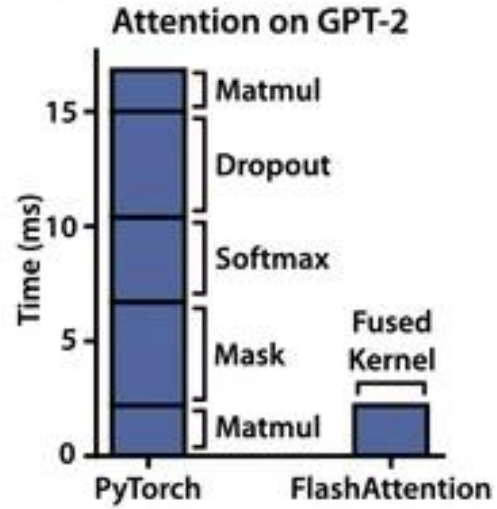
Naive implementation requires repeated R/W from slow GPU HBM

Background: Attention is Bottlenecked by Memory Reads/Writes

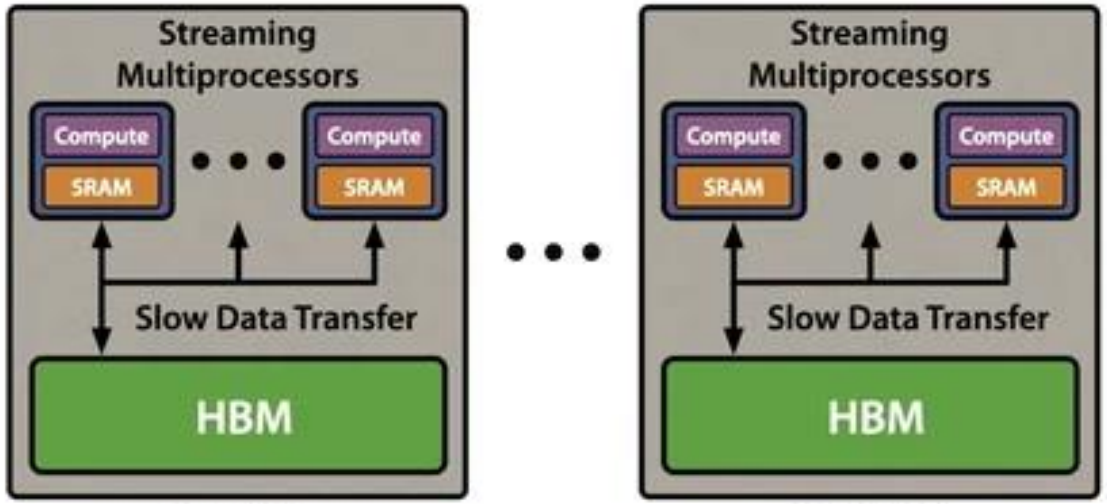


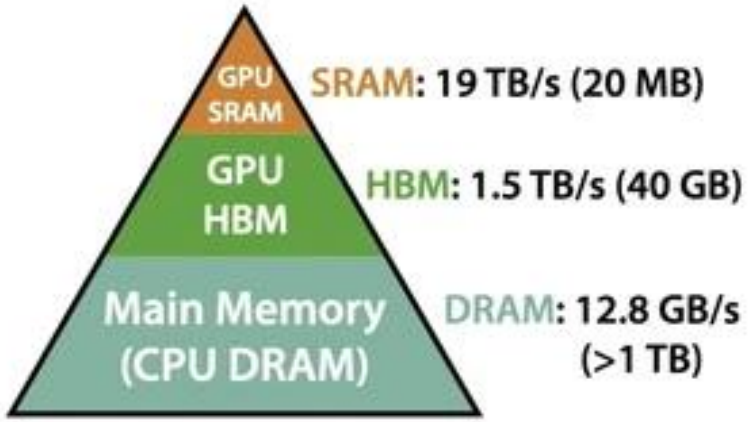
$$\mathbf{O} = \text{Dropout}(\text{Softmax}(\text{Mask}(\mathbf{QK}^T))\mathbf{V}$$

Naive implementation requires repeated R/W from slow GPU HBM



Background: GPU Compute Model & Memory Hierarchy

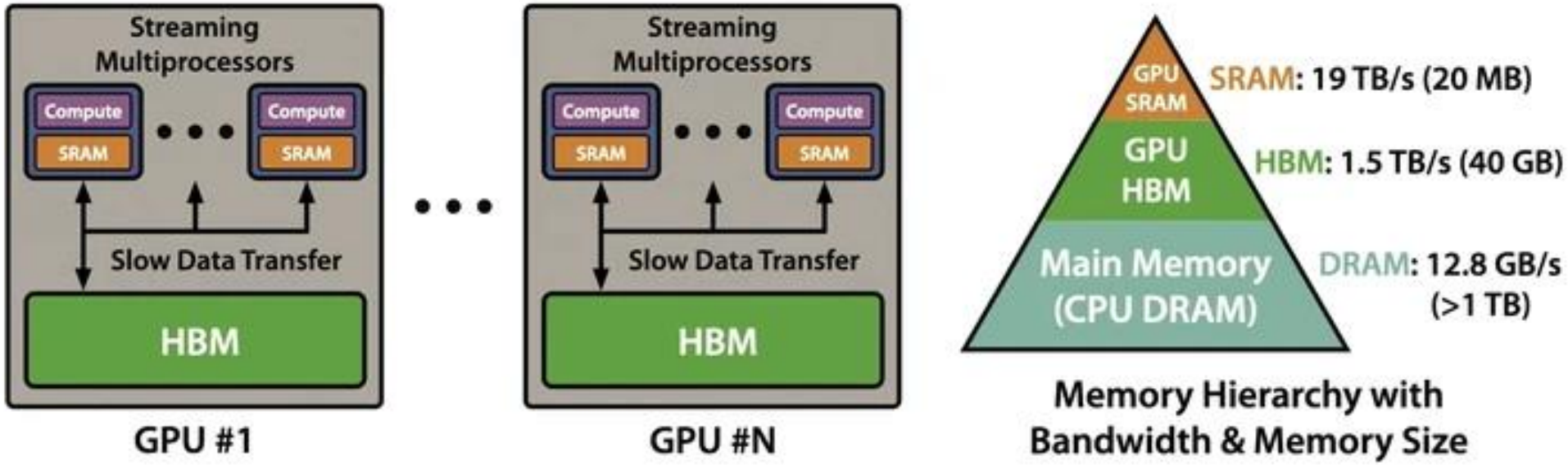




Memory Hierarchy with Bandwidth & Memory Size

Blogpost: Horace He, Making Deep Learning Go Brrrr From First Principles.

Background: GPU Compute Model & Memory Hierarchy



Can we exploit the memory asymmetry to get speed up?
With IO-awareness (accounting for R/W to different levels of memory)

Blogpost: Horace He, Making Deep Learning Go Brrrr From First Principles.

How to Reduce HBM Reads/Writes: Compute by Blocks

Challenges: (1) compute softmax reduction without access to full input.
(2) backward without the large attention matrix from forward.

How to Reduce HBM Reads/Writes: Compute by Blocks

Challenges: (1) compute softmax reduction without access to full input.
(2) backward without the large attention matrix from forward.

Tiling: Restructure algorithm to load block by block from HBM to SRAM to compute attention.

Recomputation: Don't store attn. matrix from forward, recompute it in the backward.

How to Reduce HBM Reads/Writes: Compute by Blocks

Challenges: (1) compute softmax reduction without access to full input.
(2) backward without the large attention matrix from forward.

Tiling: Restructure algorithm to load block by block from HBM to SRAM to compute attention.

Recomputation: Don't store attn. matrix from forward, recompute it in the backward.

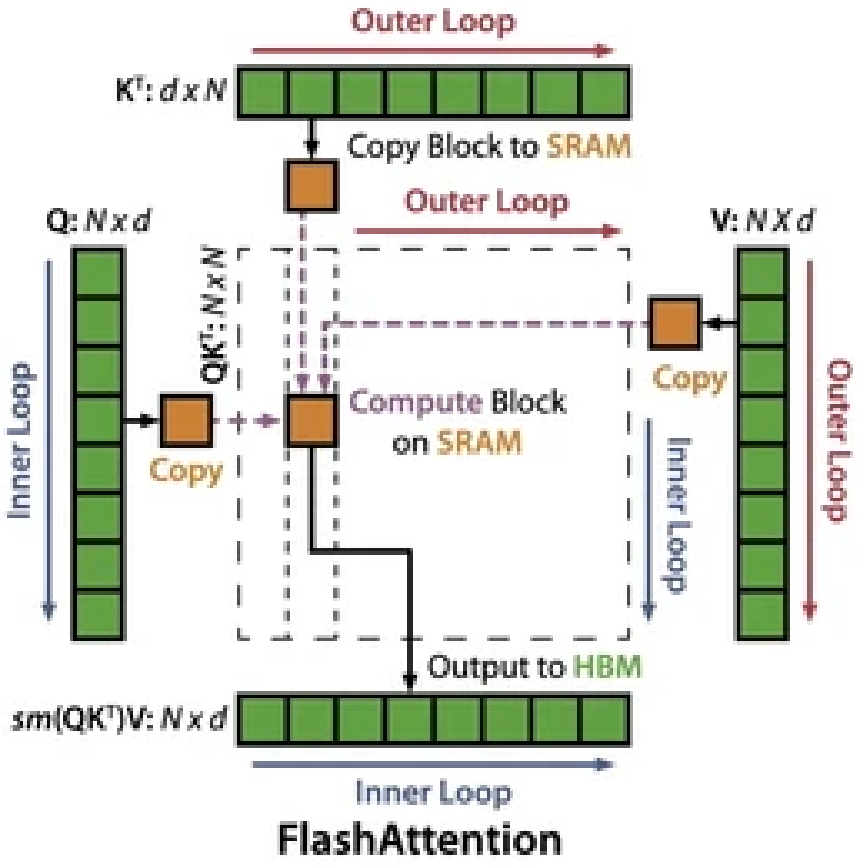
Implementation: fused CUDA kernel for fine-grained control of memory accesses.

Tiling

Decomposing large softmax into smaller ones by scaling.

$$\text{softmax}([A_1, A_2]) = [\alpha\ \text{softmax}(A_1),\ \ \beta\ \text{softmax}(A_2)]\,.$$

$$\text{softmax}([A_1, A_2]) \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \alpha\ \text{softmax}(A_1)\ V_1 + \beta\ \text{softmax}(A_2)\ V_2\,.$$



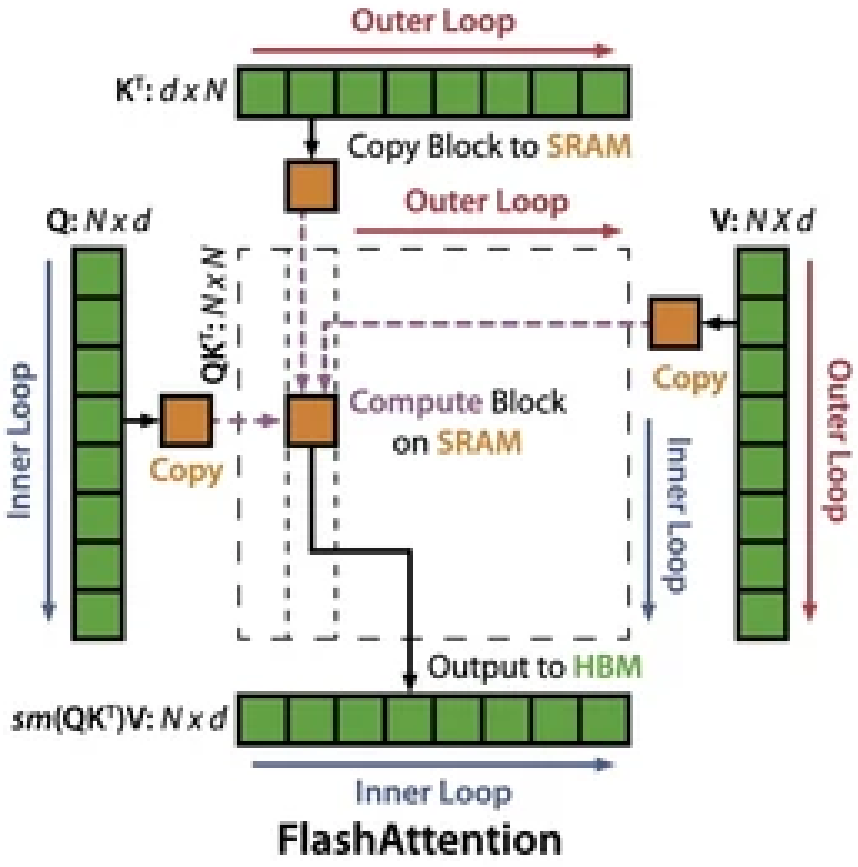
Tiling

Decomposing large softmax into smaller ones by scaling.

$$\text{softmax}([A_1, A_2]) = [\alpha\ \text{softmax}(A_1),\ \ \beta\ \text{softmax}(A_2)]\,.$$

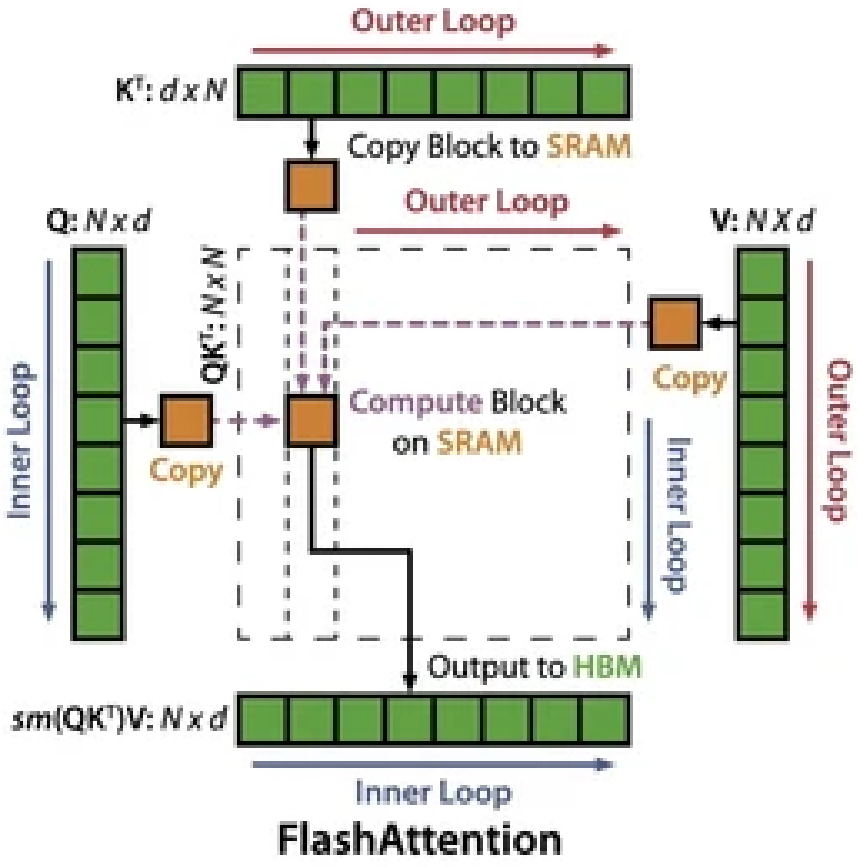
$$\text{softmax}([A_1, A_2]) \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \alpha\ \text{softmax}(A_1)\ V_1 + \beta\ \text{softmax}(A_1)\ V_2\,.$$

1. Load inputs by blocks from HBM to SRAM.
2. On chip, compute attention output wrt that block.
3. Update output in HBM by scaling.



Recomputation (Backward Pass)

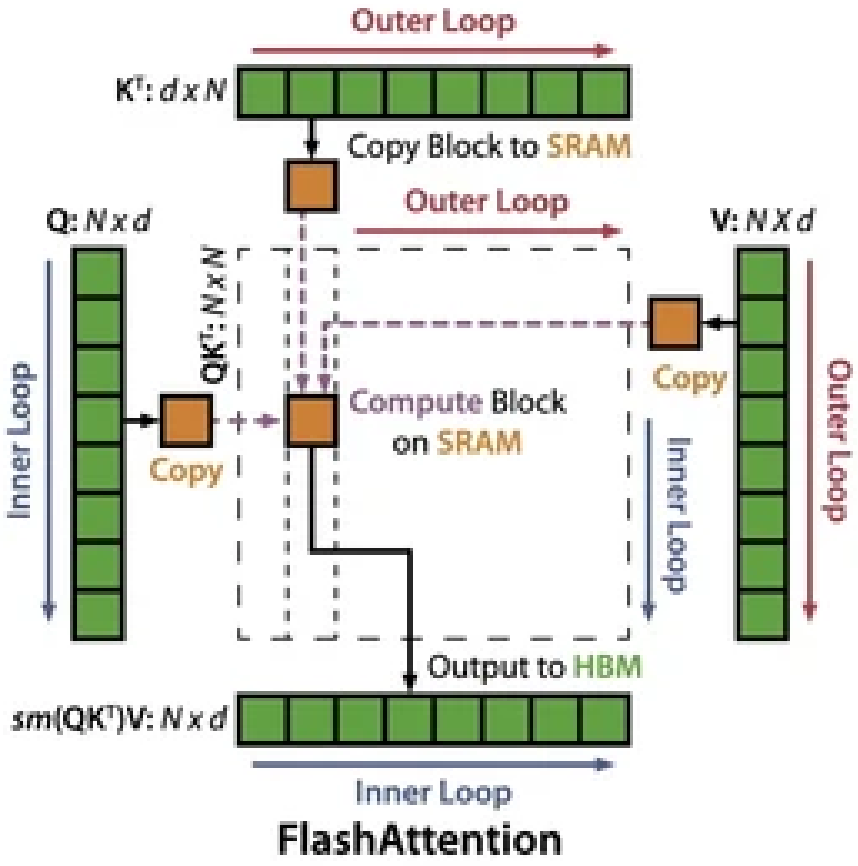
By storing softmax normalization factors from fwd (size N), quickly recompute attention in the bwd from inputs in SRAM.



Recomputation (Backward Pass)

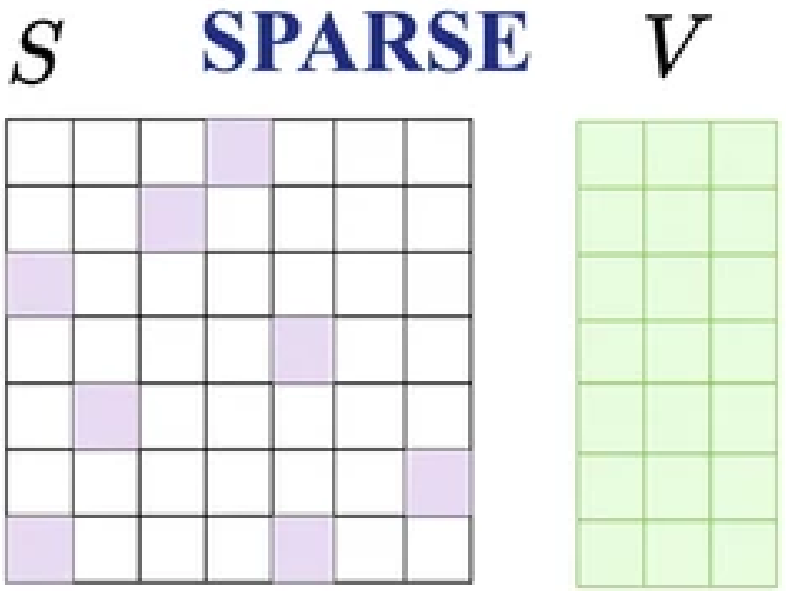
By storing softmax normalization factors from fwd (size N), quickly recompute attention in the bwd from inputs in SRAM.

Attention	Standard	FLASHATTENTION
GFLOPs	66.6	75.2
HBM R/W (GB)	40.3	4.4
Runtime (ms)	41.7	7.3

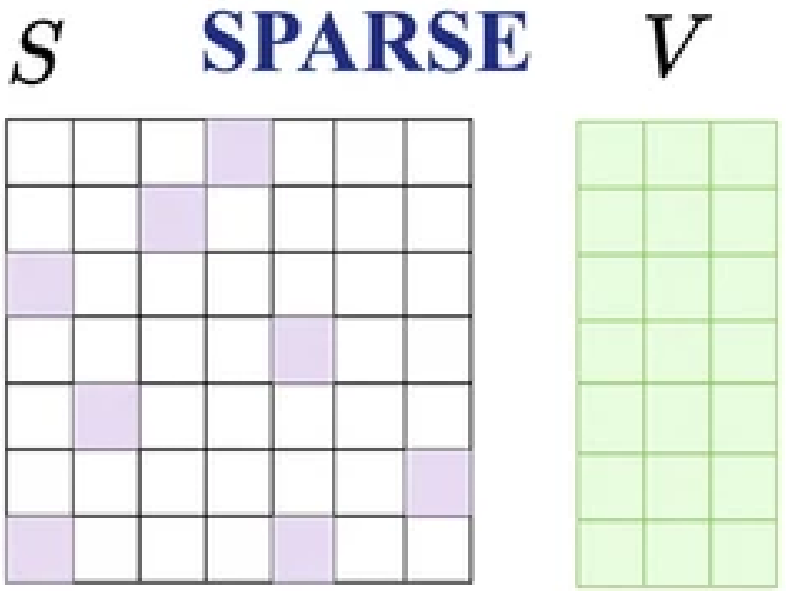


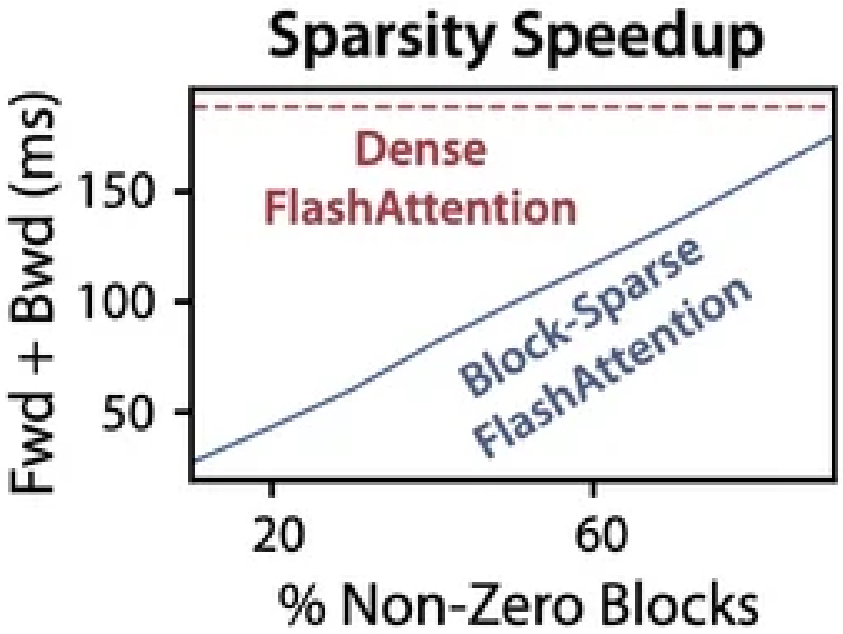
Speed up backward pass even with increased FLOPs.

Extension to Block-Sparse Attention



Extension to Block-Sparse Attention

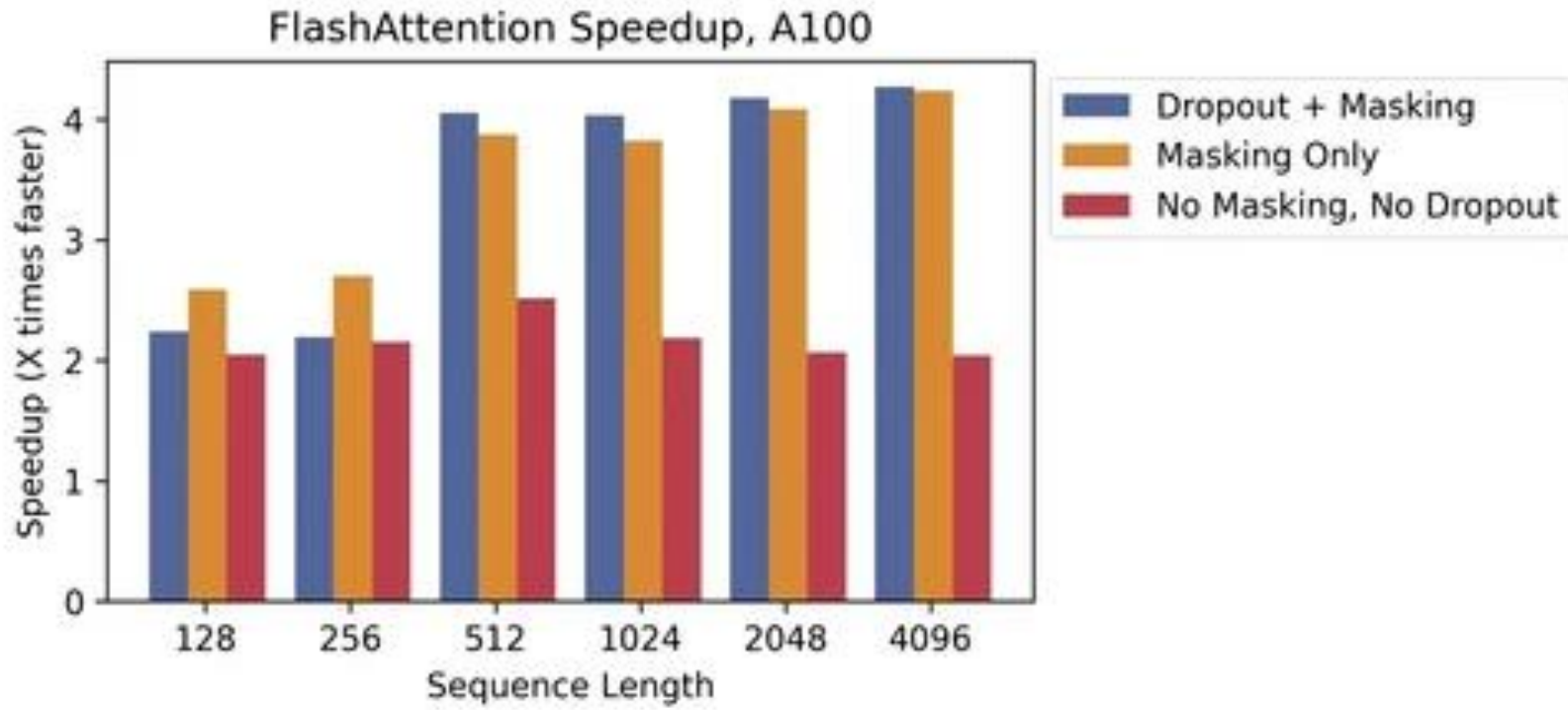




Just skip the zero blocks!

Attention Module: 2-4x speedup

Benchmarking **runtime** against sequence length



2-4x speedup over naive attention, esp. with dropout/masking

In paper: comparison with sparse/approximate attention

Faster Transformer Training: MLPerf Record for Training BERT

MLPerf: (highly optimized) standard benchmark for training speed

Time to hit an accuracy of 72.0% on MLM from a fixed checkpoint, averaged across 10 runs on 8xA100 GPUs

BERT Implementation	Training time (minutes)
Nvidia MLPerf 1.1 [56]	20.0 ± 1.5
FlashAttention (ours)	**17.4** ± 1.4

FlashAttention outperforms the previous MLPerf record by 15%

Mattson et al. *MLSys 2020*.

Faster Transformer Training: GPT-2

Training GPT-2 small and GPT-2 medium from scratch on OpenWebText, using 8xA100 GPUs

Model implementations	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Huggingface [84]	18.2	9.5 days (1.0×)
GPT-2 small - Megatron-LM [74]	18.2	4.7 days (2.0×)
GPT-2 small - FLASHATTENTION	18.2	**2.7 days (3.5×)**
GPT-2 medium - Huggingface [84]	14.2	21.0 days (1.0×)
GPT-2 medium - Megatron-LM [74]	14.3	11.5 days (1.8×)
GPT-2 medium - FLASHATTENTION	14.3	**6.9 days (3.0×)**

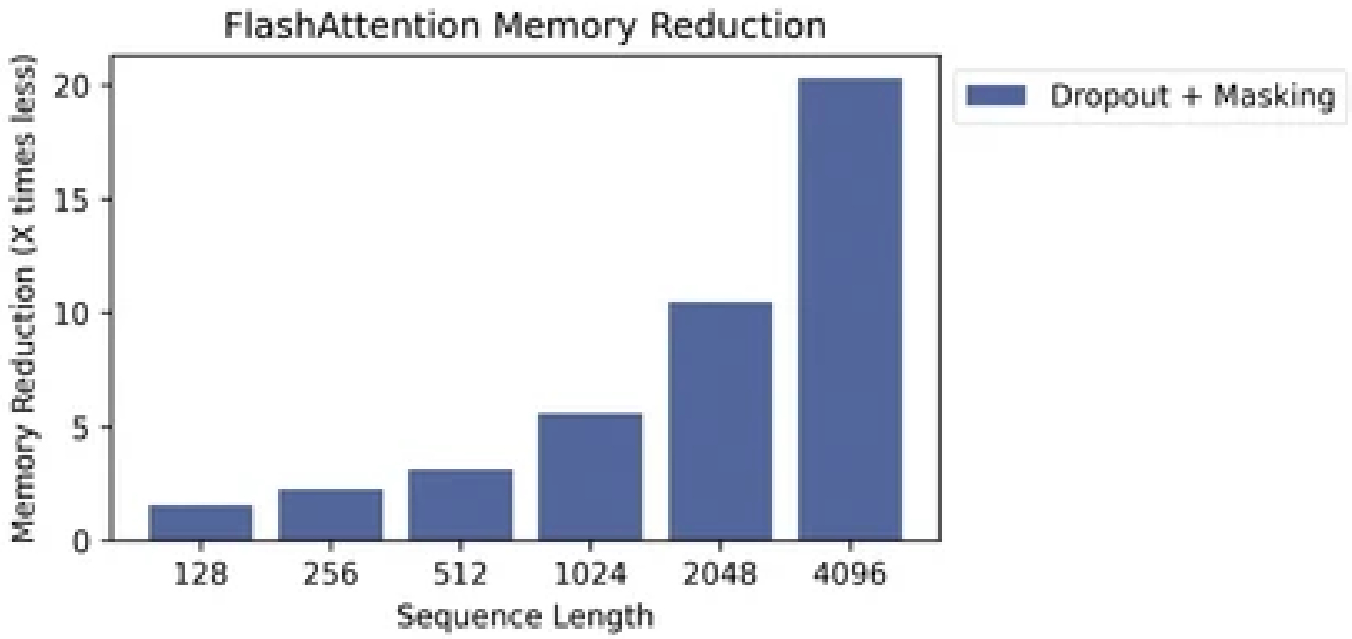
FlashAttention speeds up GPT-2 training by 2.0-3.5x

Longer sequence length: more speedup

Wolf et al. *EMNLP 2020.*
Shoeybi et al. *arXiv:1909.08053 2019.*

Attention Module: 10-20x memory reduction

Benchmarking **memory footprint** against sequence length



10-20x memory saving

Memory linear in sequence length — scaling up to 64K on one A100

Longer Sequences: GPT-2 with Long Context

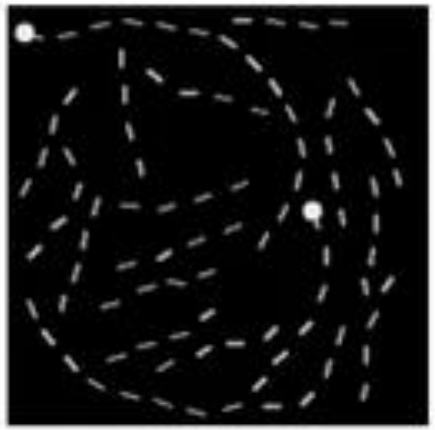
Training GPT-2 small with a longer context

Longer context: slower, but lower perplexity (better model)

Model implementations	Context length	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Megatron-LM	1k	18.2	4.7 days (1.0×)
GPT-2 small - FlashAttention	1k	18.2	**2.7 days (1.7×)**
GPT-2 small - FlashAttention	2k	17.6	3.0 days (1.6×)
GPT-2 small - FlashAttention	4k	**17.5**	3.6 days (1.3×)

FlashAttention with context length 4K is faster than Megatron-LM at 1K, and achieves lower perplexity

Longer Sequences: Path-X, Path-256



Path-X: Tell whether two dots are connected — designed to push Transformers (no patches)

Requires sequence length 16K/64K for Path-X/Path-256

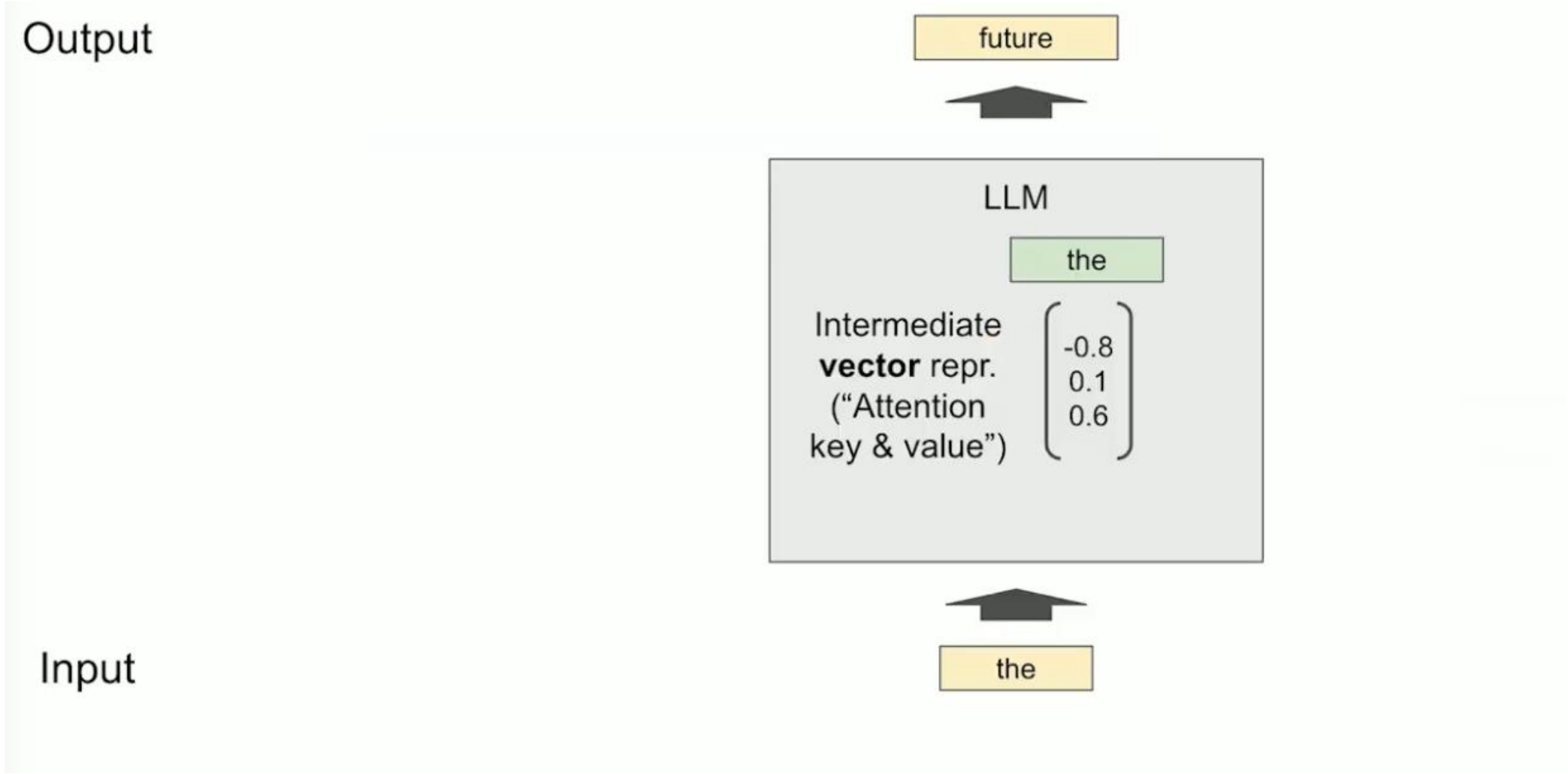
Model	Path-X	Path-256
Transformer	✗	✗
Linformer [81]	✗	✗
Linear Attention [48]	✗	✗
Performer [11]	✗	✗
Local Attention [77]	✗	✗
Reformer [49]	✗	✗
SMYRF [18]	✗	✗
FlashAttention	**61.4**	✗
Block-sparse FlashAttention	56.0	**63.1**

FlashAttention yields the first Transformer that can solve Path-X

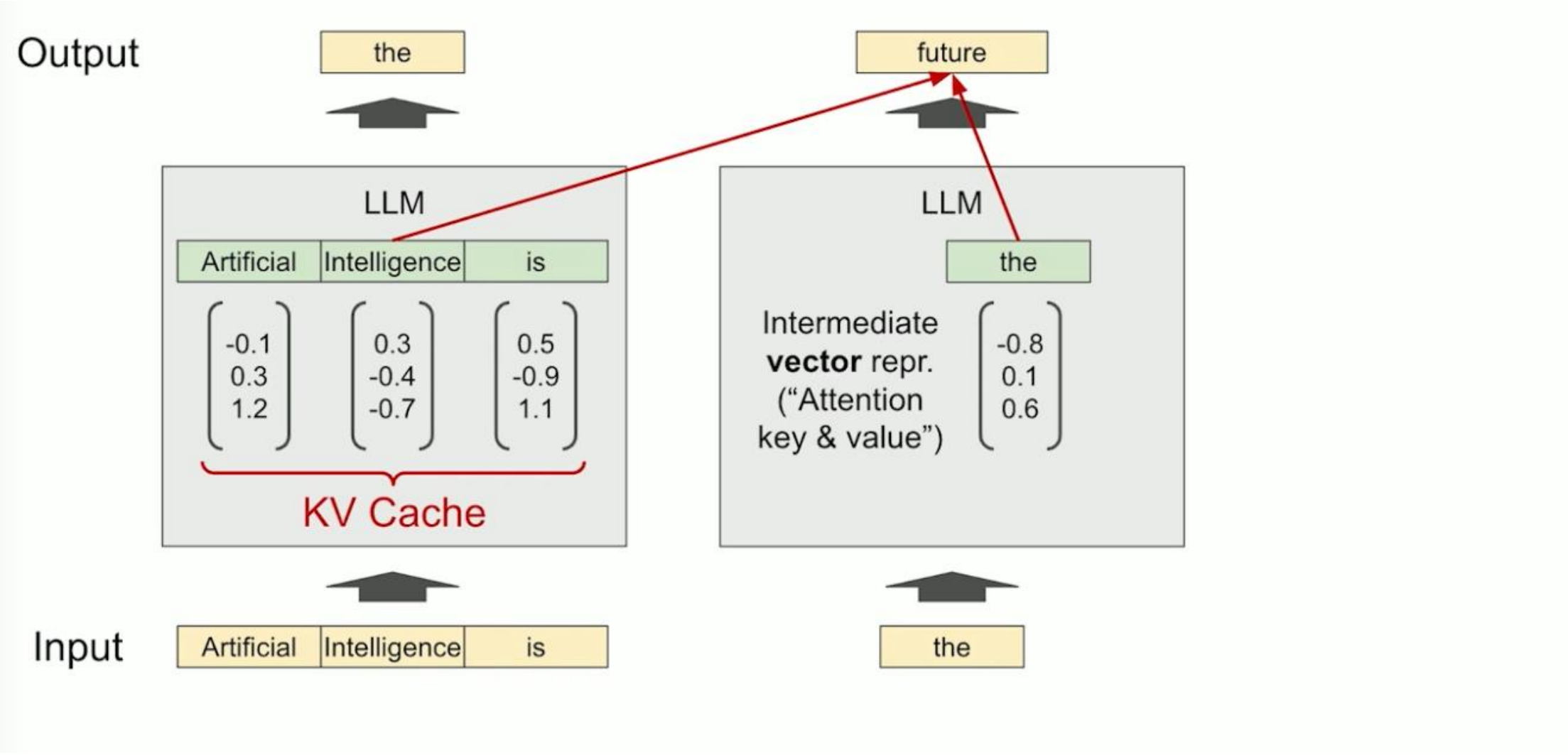
Block-sparse FlashAttention enables Path-256

Tay et al. *ICLR 2020*.

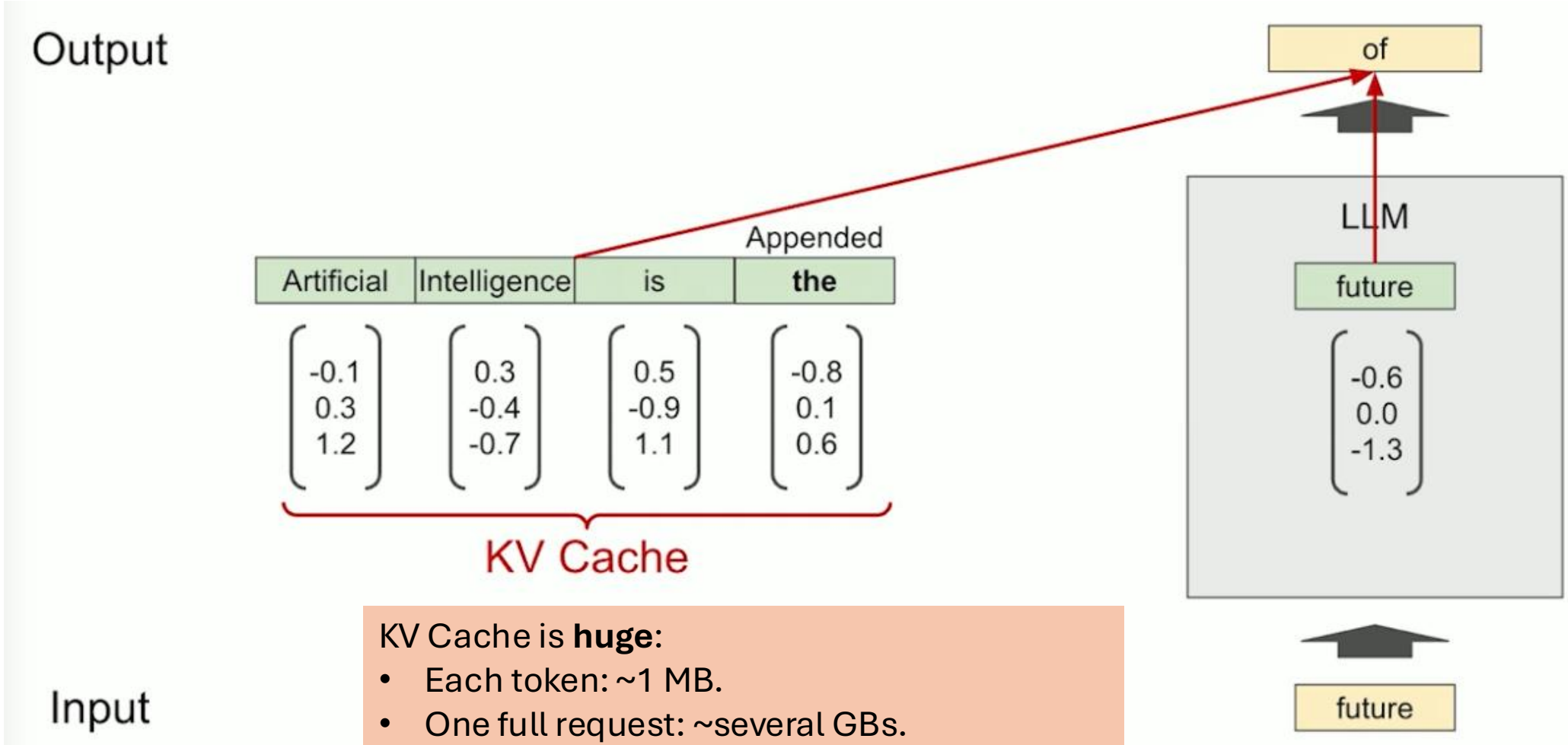
(Attention) KV Cache



(Attention) KV Cache



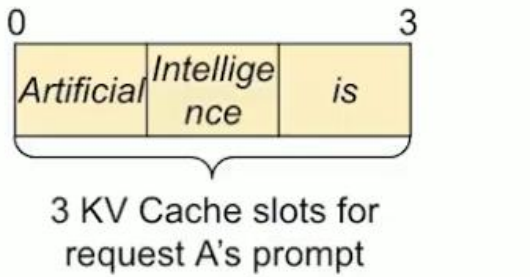
KV Cache dynamically grows and shrinks



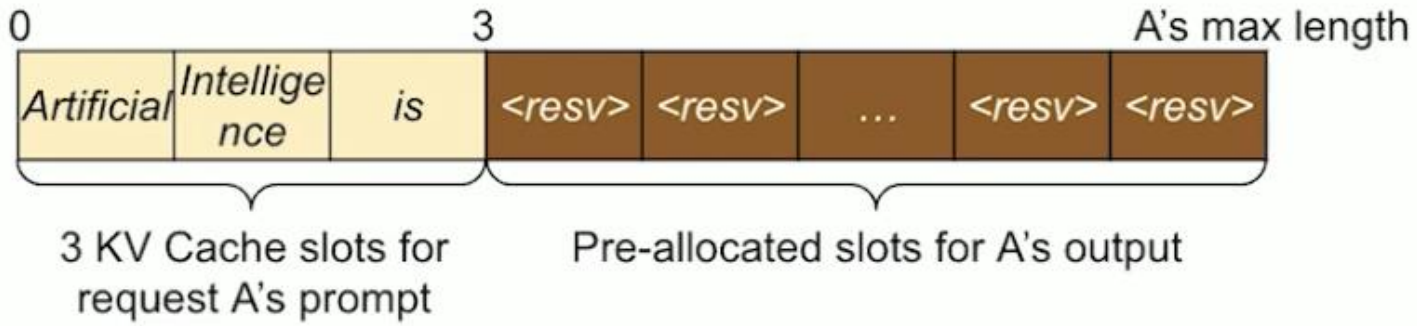
KV Cache is **huge**:

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

KV Cache management in previous sytems

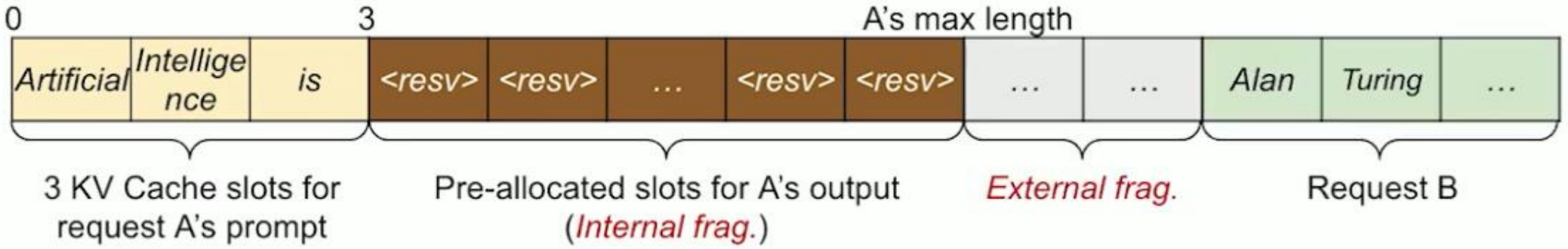


KV Cache management in previous systems



- **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., faster pointer arithmetic, efficient memory access)
- Results in memory fragmentation
  - **Internal fragmentation** due to the **unknown** output length.

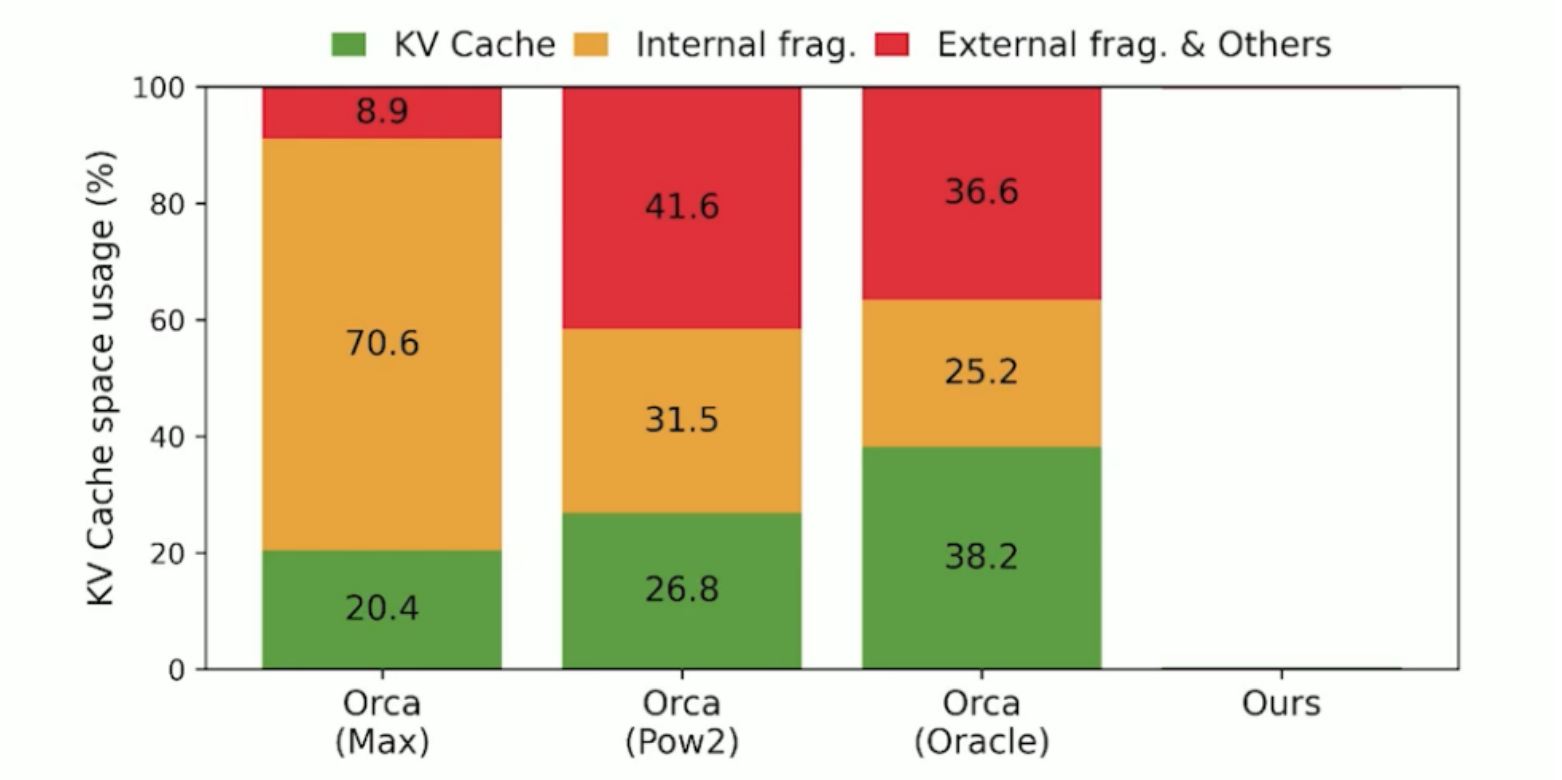
KV Cache management in previous systems



- **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., faster pointer arithmetic, efficient memory access)
- Results in memory fragmentation
  - **Internal fragmentation** due to the **unknown** output length.
  - **External fragmentation** due to **non-uniform** pre-request max lengths.

Significant memory waste in KV Cache space

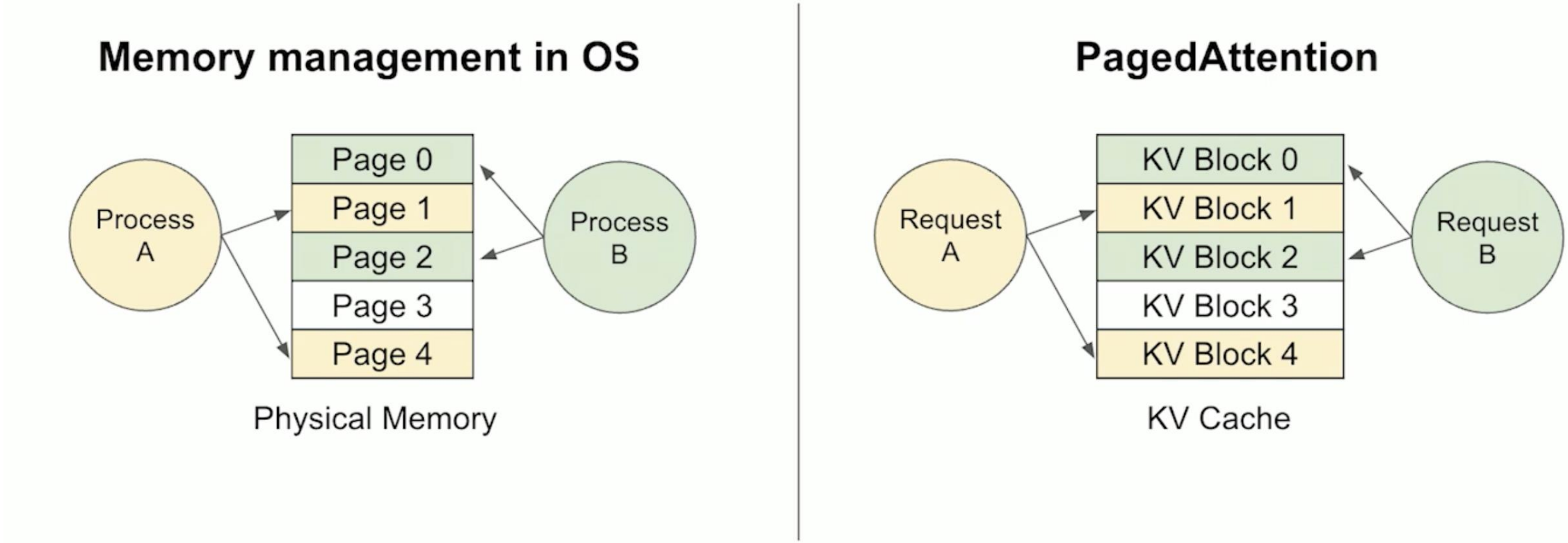
- Only **20-40%** of KV Cache space is utilized to store actual token states



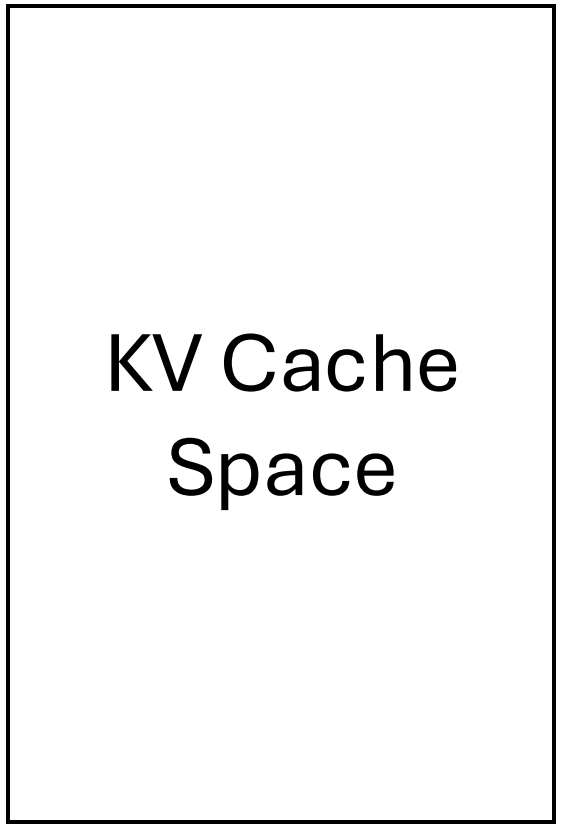
Yu, Gyeong-In, et al. "Orca: A distributed serving system for {Transformer-Based} generative models." *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 2022.

PagedAttention

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

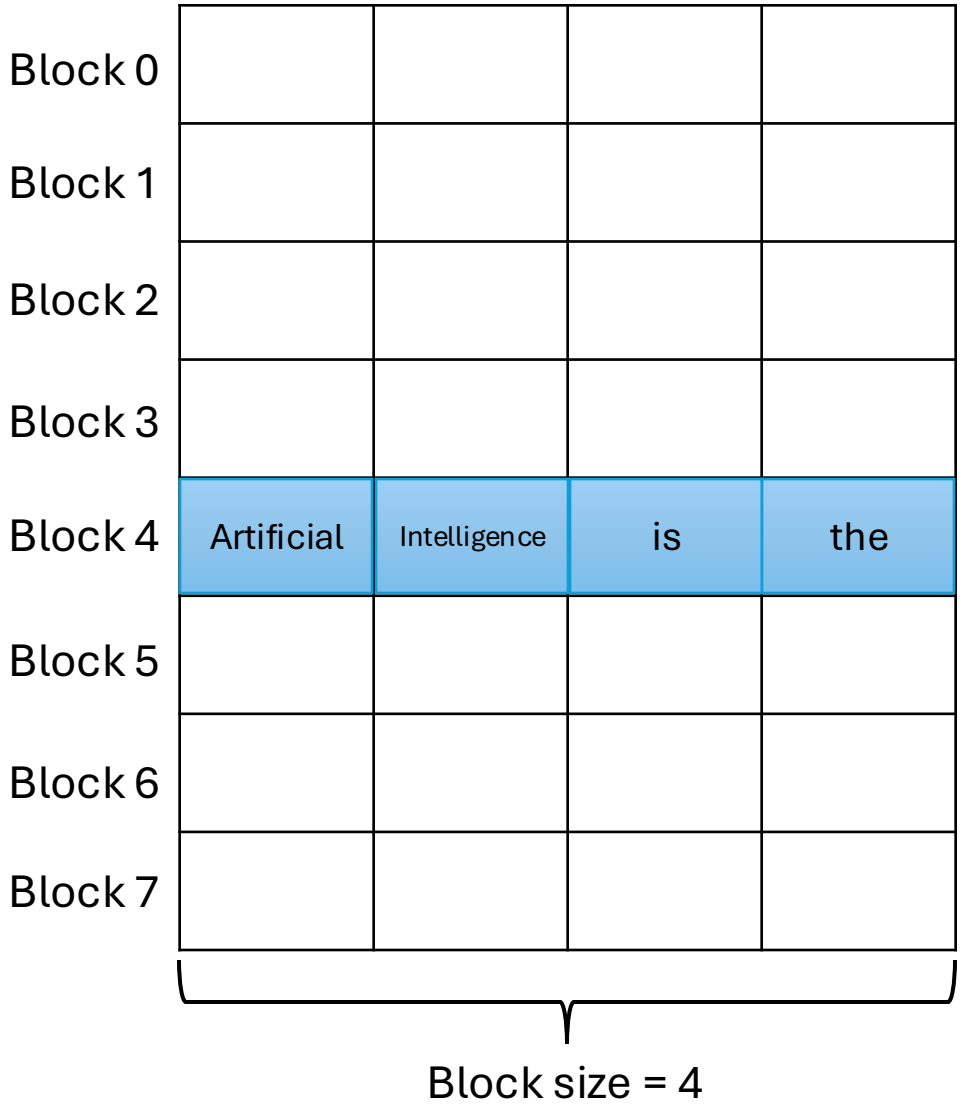


Paging KV Cache Space into KV Blocks

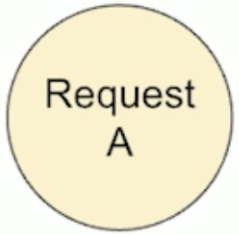


Paging KV Cache Space into KV Blocks

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



Virtualizing KV Cache



Prompt: “Alan Turing is a computer scientist”

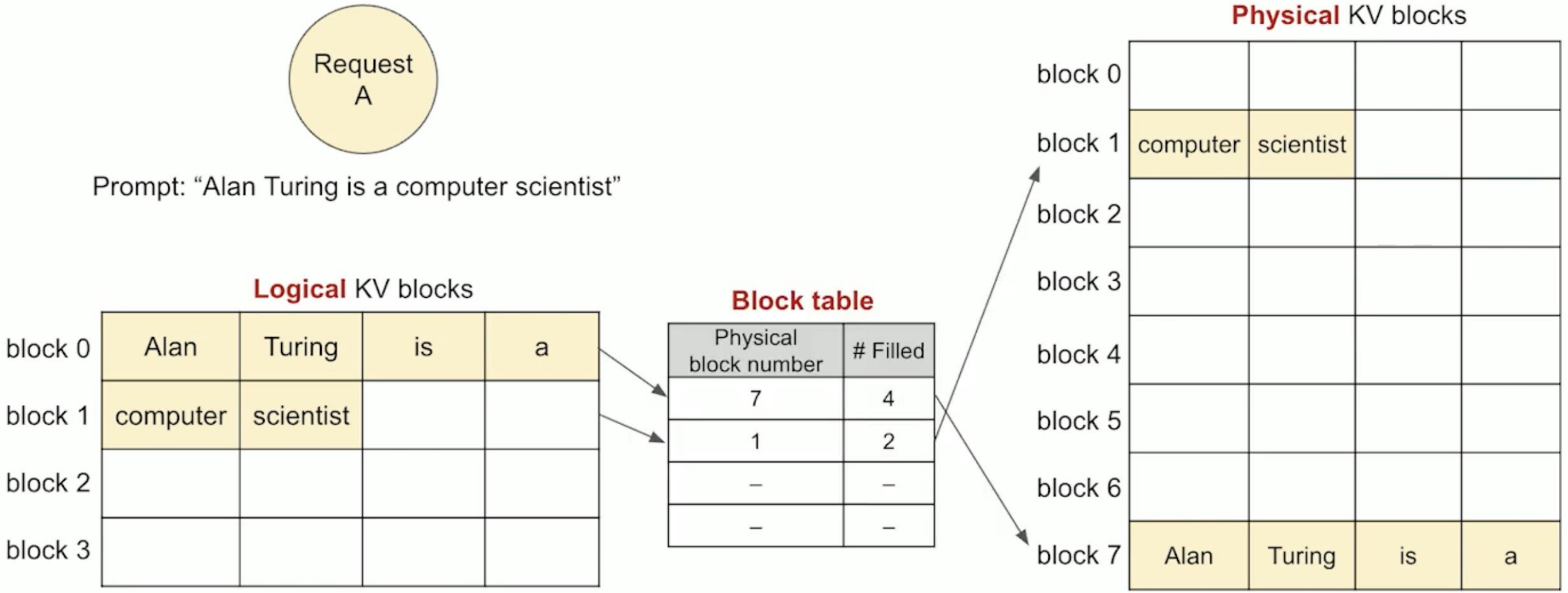
Logical KV blocks

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

Physical KV blocks

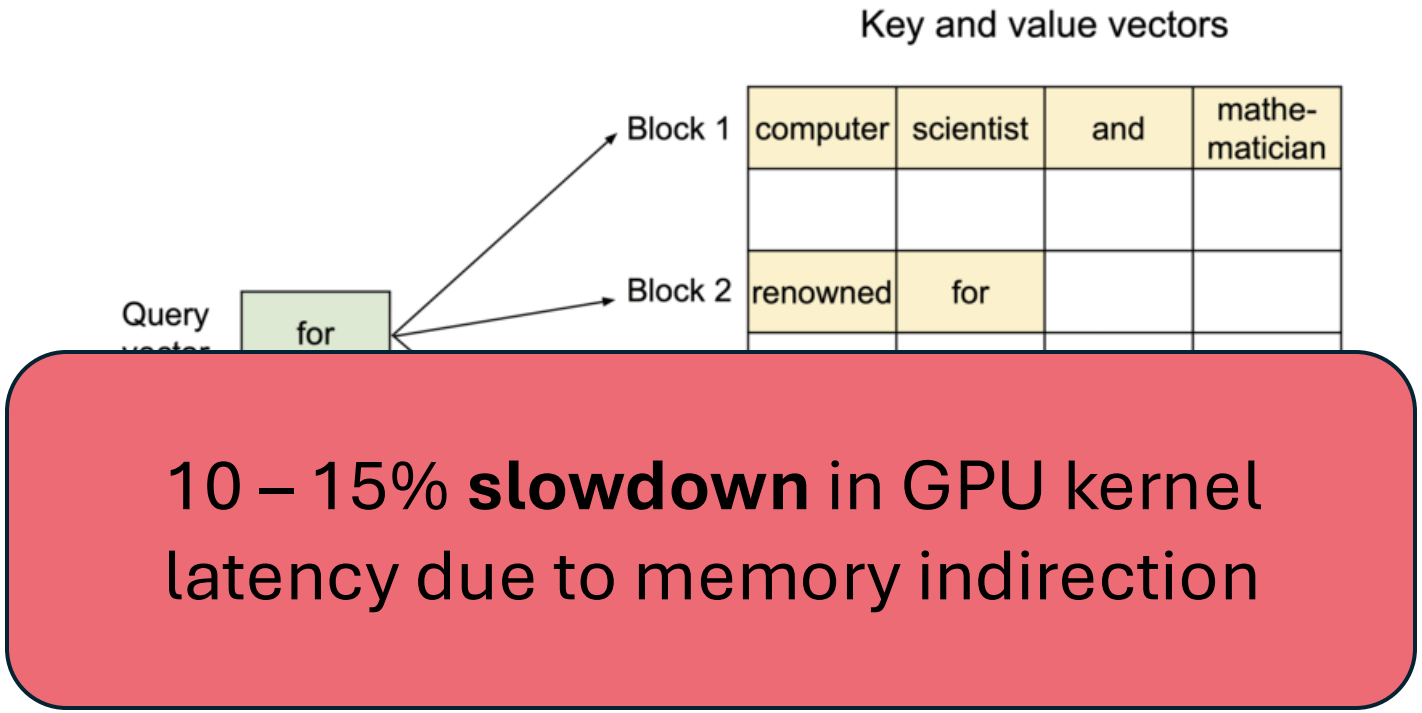
block 0				
block 1	computer	scientist		
block 2				
block 3				
block 4				
block 5				
block 6				
block 7	Alan	Turing	is	a

Virtualizing KV Cache



Attention mechanism with virtualized KV Cache

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



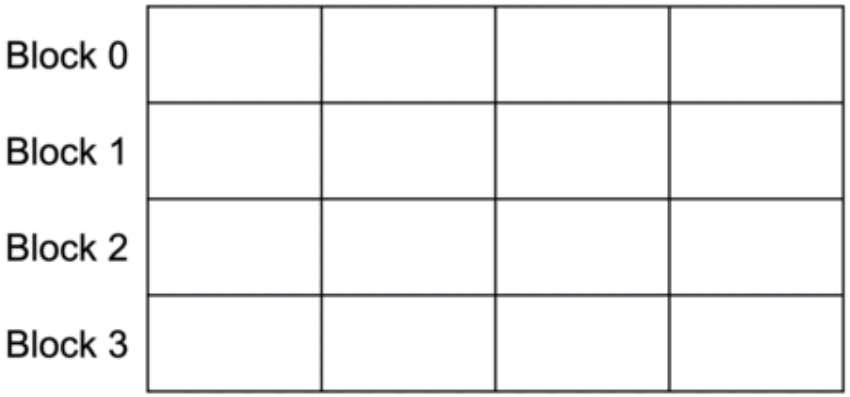
Memory management with PagedAttention

0. Before generation.

Seq A

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

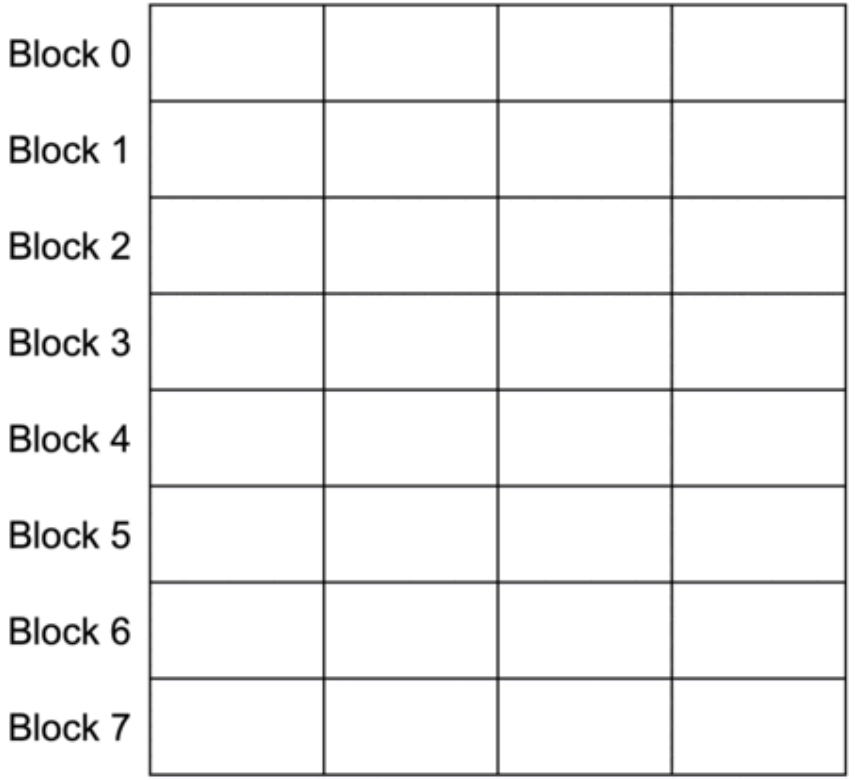
Logical KV cache blocks



Block table

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

Physical KV cache blocks

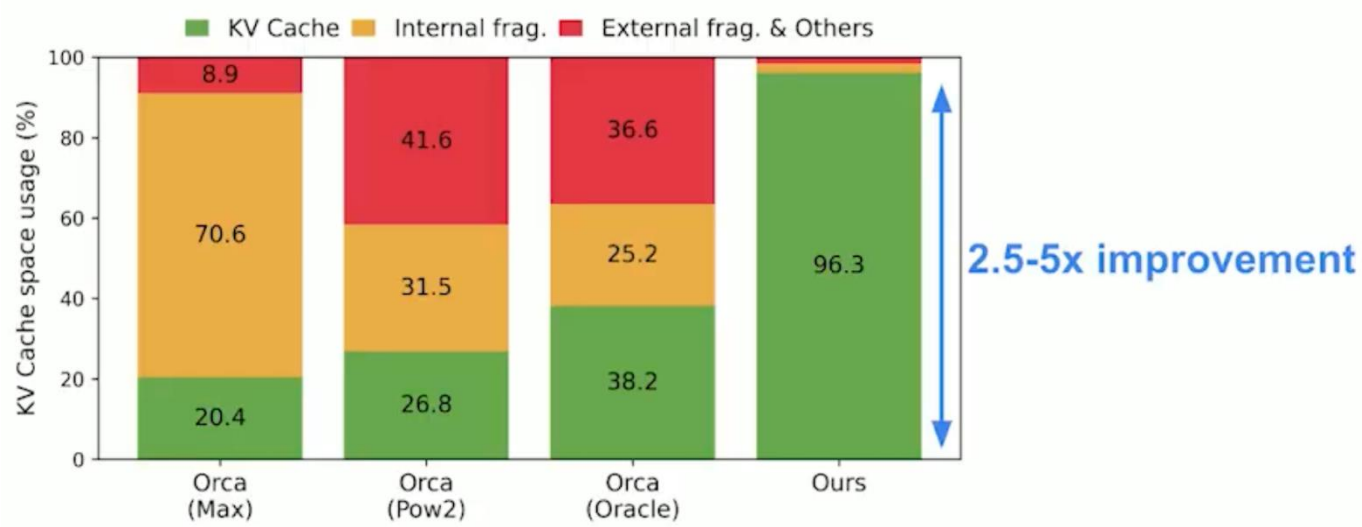


Memory efficiency of PagedAttention

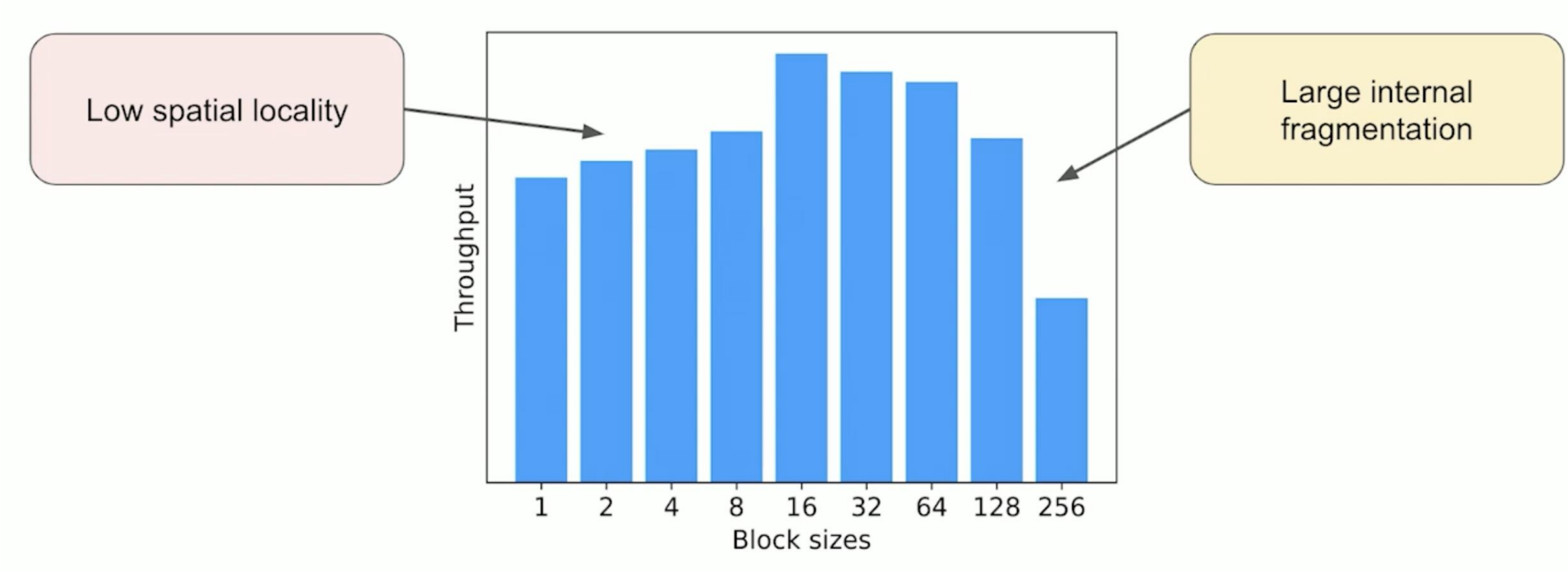
- Minimal internal fragmentation
  - Only happens at the last block of a sequence
  - # wasted tokens / seq < block size
    - Sequence: O(100) – O(1000) tokens
    - Block size: O(10) tokens
- No external fragmentation

Alan	Turing	is	a
computer	scientist	and	mathematician
renowned			

Internal fragmentation

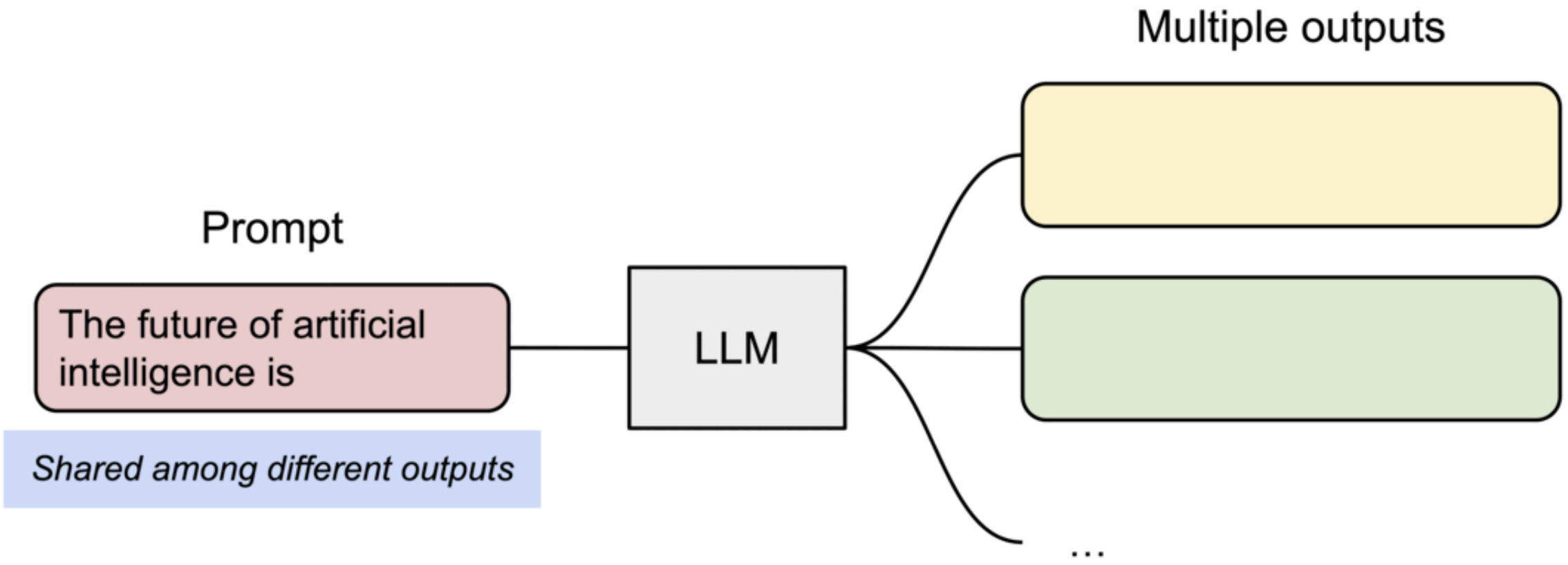


Configuring the block size



- Block size 16 works generally well in practice

Paging enables sharing



Sharing KV blocks

Sharing KV blocks

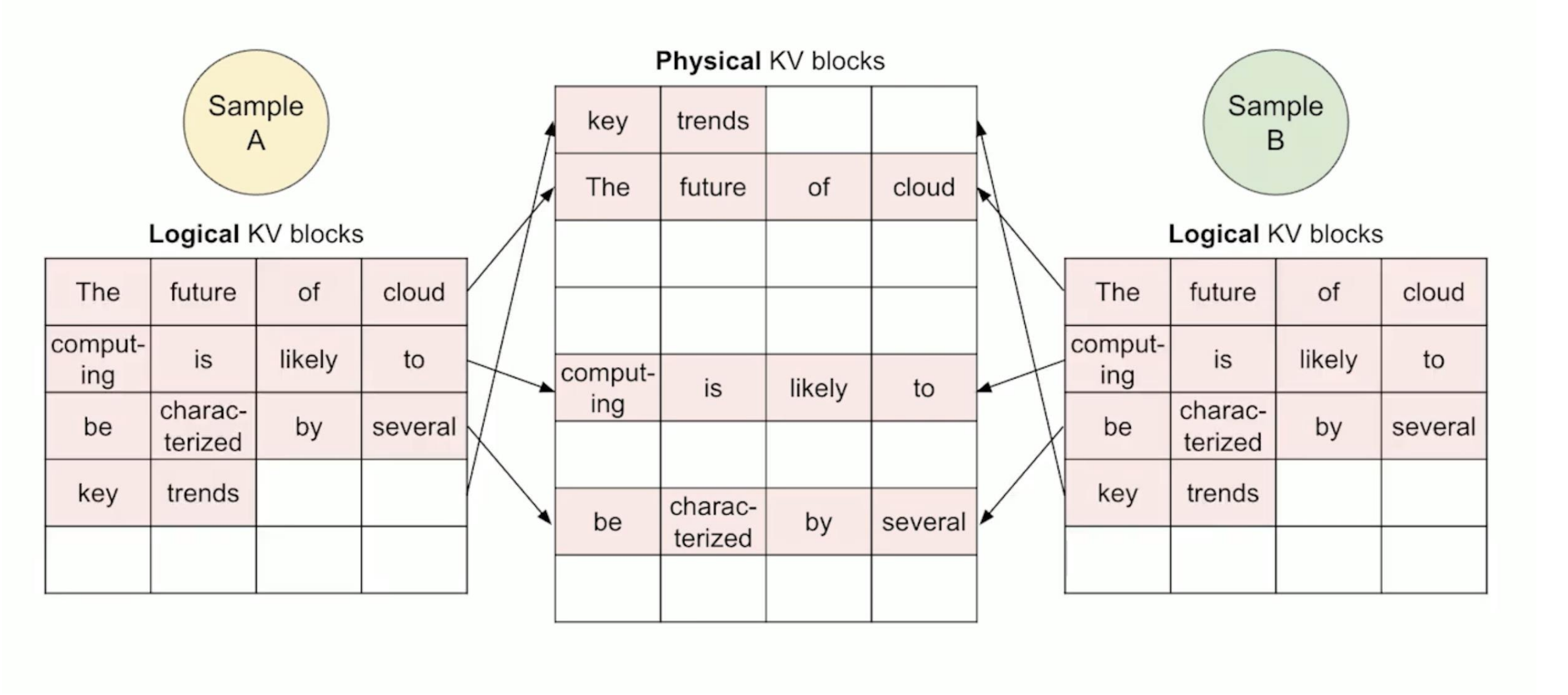
Sample A

Physical KV blocks

key	trends		
The	future	of	cloud
comput-ing	is	likely	to
be	charac-terized	by	several

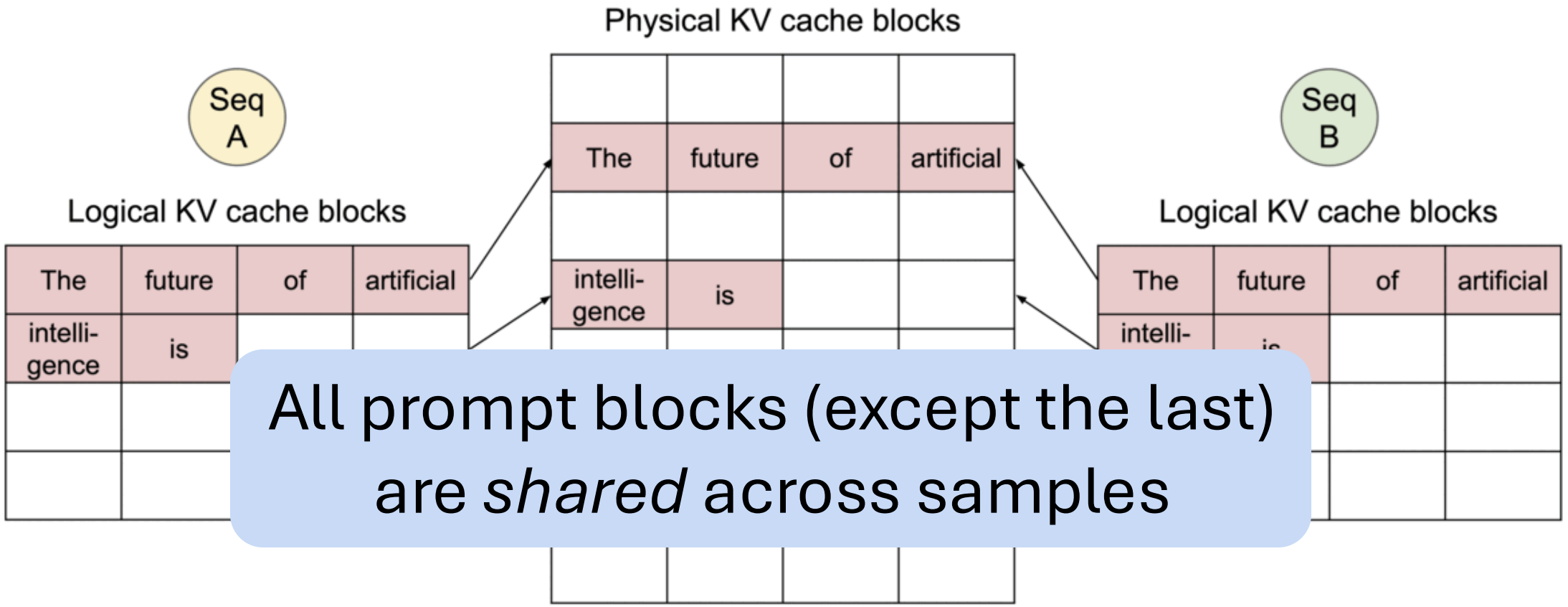
Sample B

Sharing KV blocks



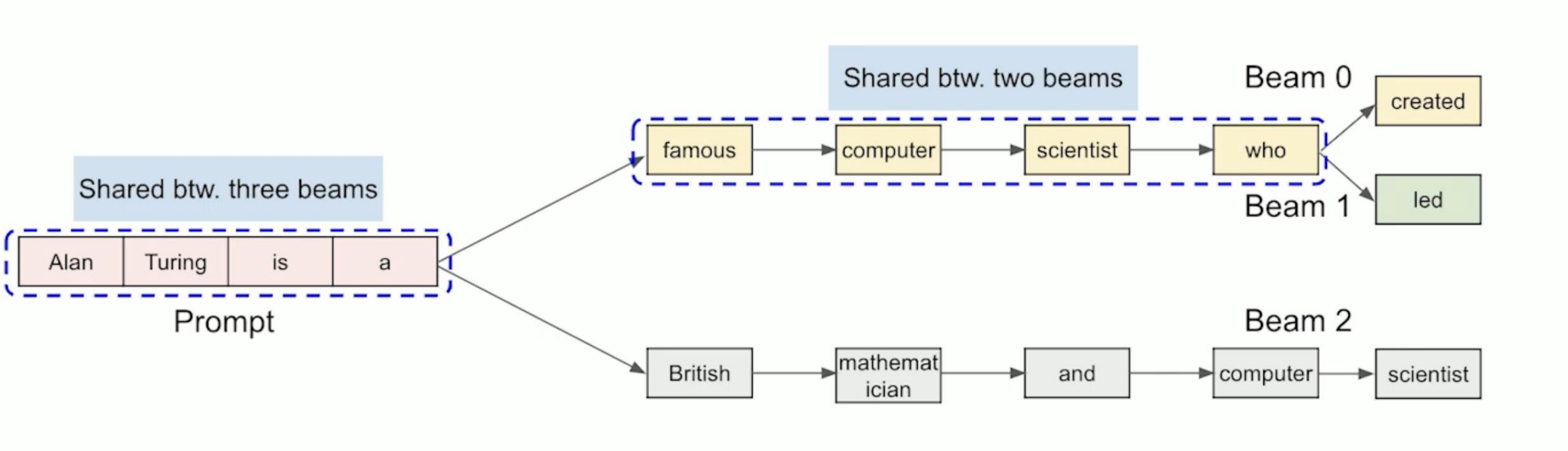
Sharing KV blocks

0. Shared prompt: Map logical blocks to the same physical blocks.



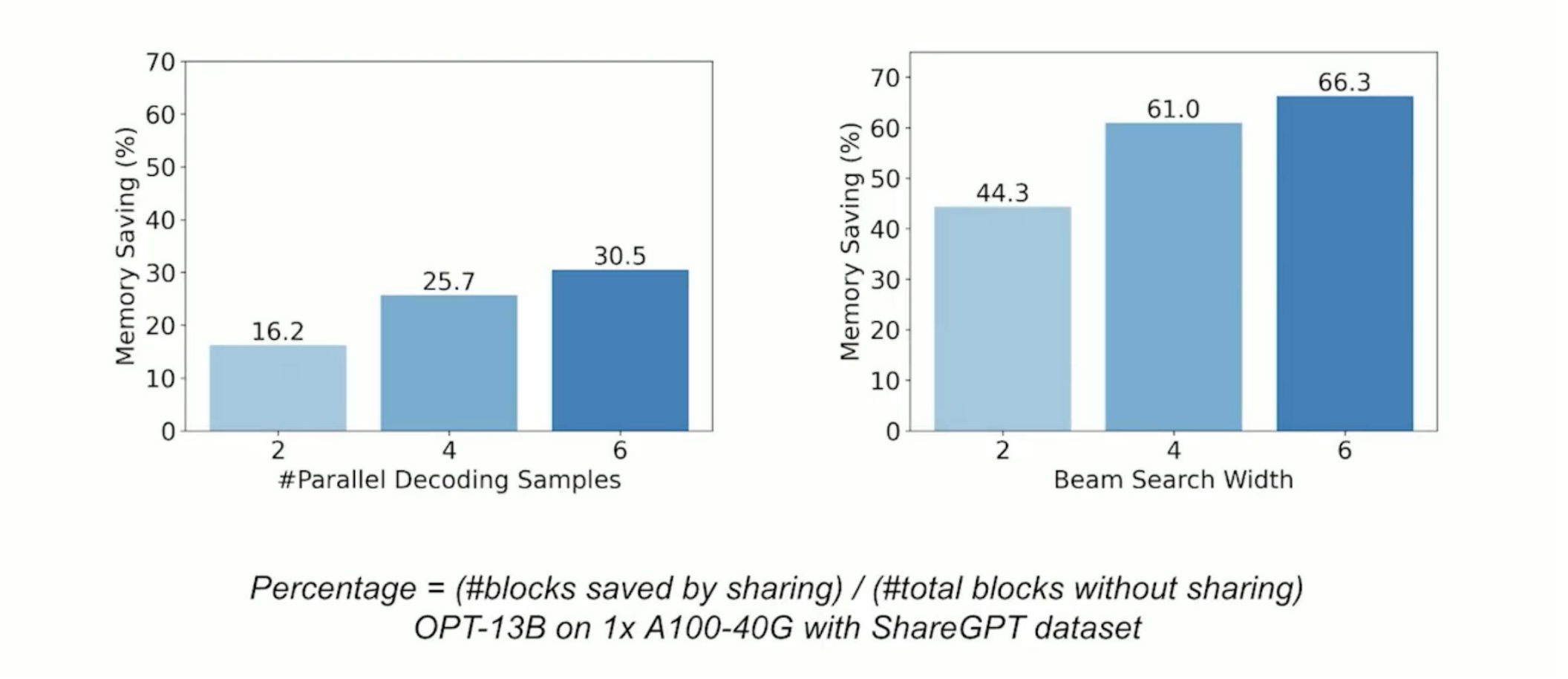
All prompt blocks (except the last) are *shared* across samples

More complex sharing: beam search



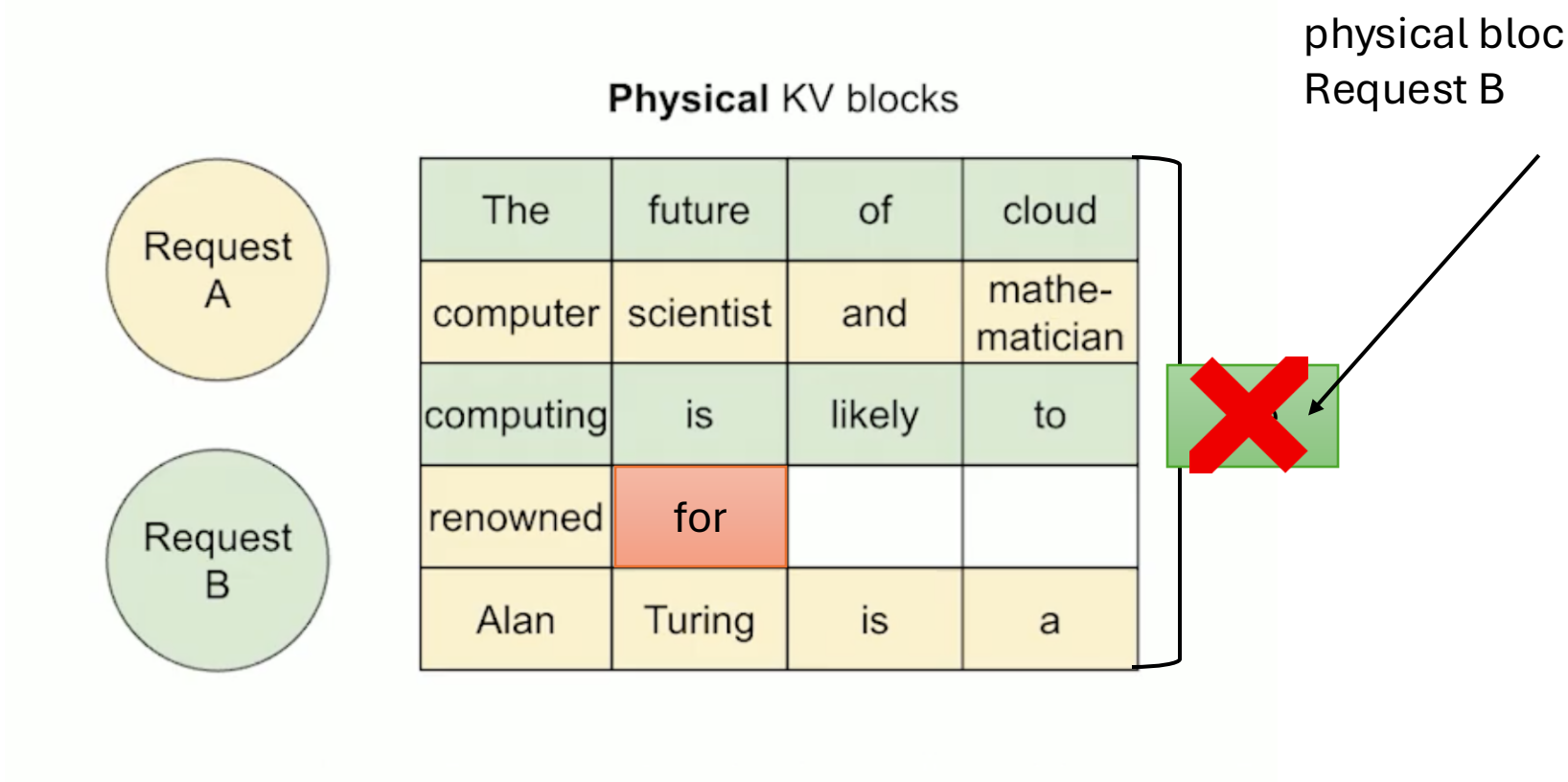
- Similar to process tree (fork & kill)
- Efficiently supported by paged attention and copy-on-write mechanism

Memory saving via sharing



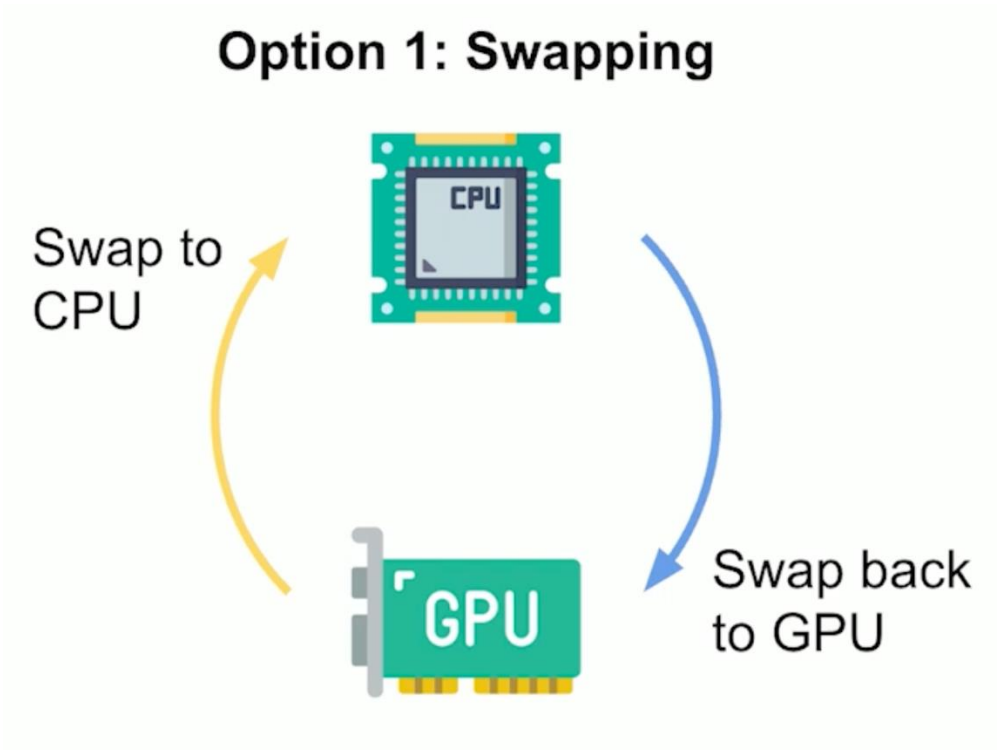
Kwon, Woosuk, et al. "Efficient memory management for large language model serving with pagedattention." *Proceedings of the 29th Symposium on Operating Systems Principles*. 2023.

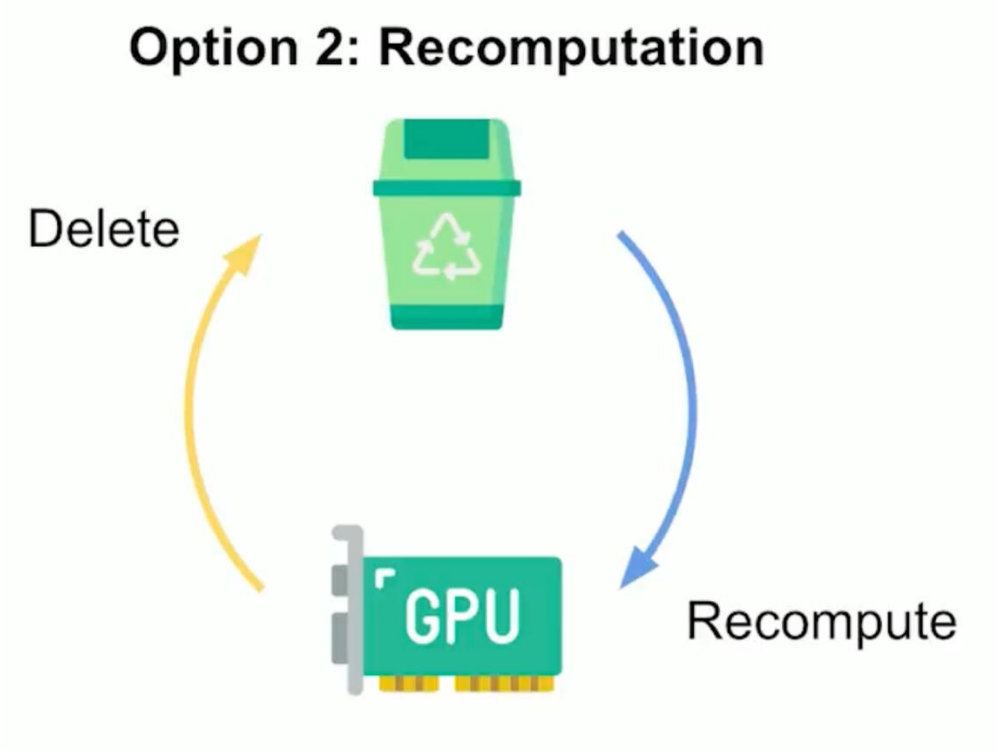
Out of KV Block Memory



Request Preemption & Recovery

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





Notes on Preemption & Recovery

- Swap/recompute the whole request since all previous tokens are required every step.
- **Swapping**: smaller block size → higher overhead due to small data transfers.
- **Recomputation**: surprisingly fast since all token's KV cache can be computed in parallel.

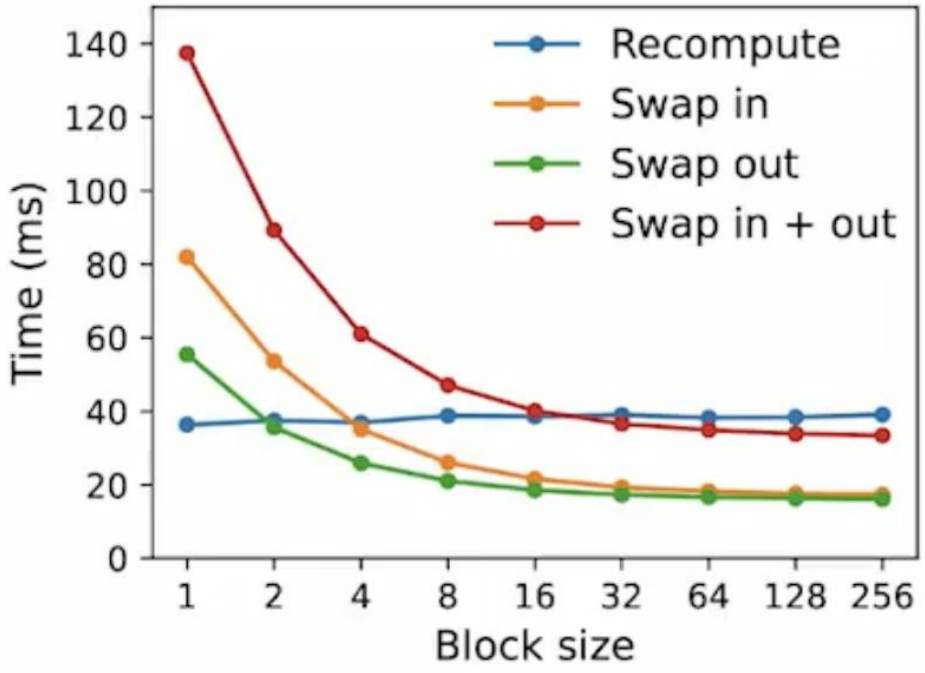


Figure: *Swap/Recomputation latency of 256 tokens.*

vLLM strategy: Use recomputation when possible with FCFS policy.

Comparison with Operating System Virtual Memory

Analogies

OS pages ⇔ KV blocks

- Reduce memory fragmentation

Shared pages across processes ⇔ Shared KV blocks across samples

- Reduce memory waste via sharing

Differences

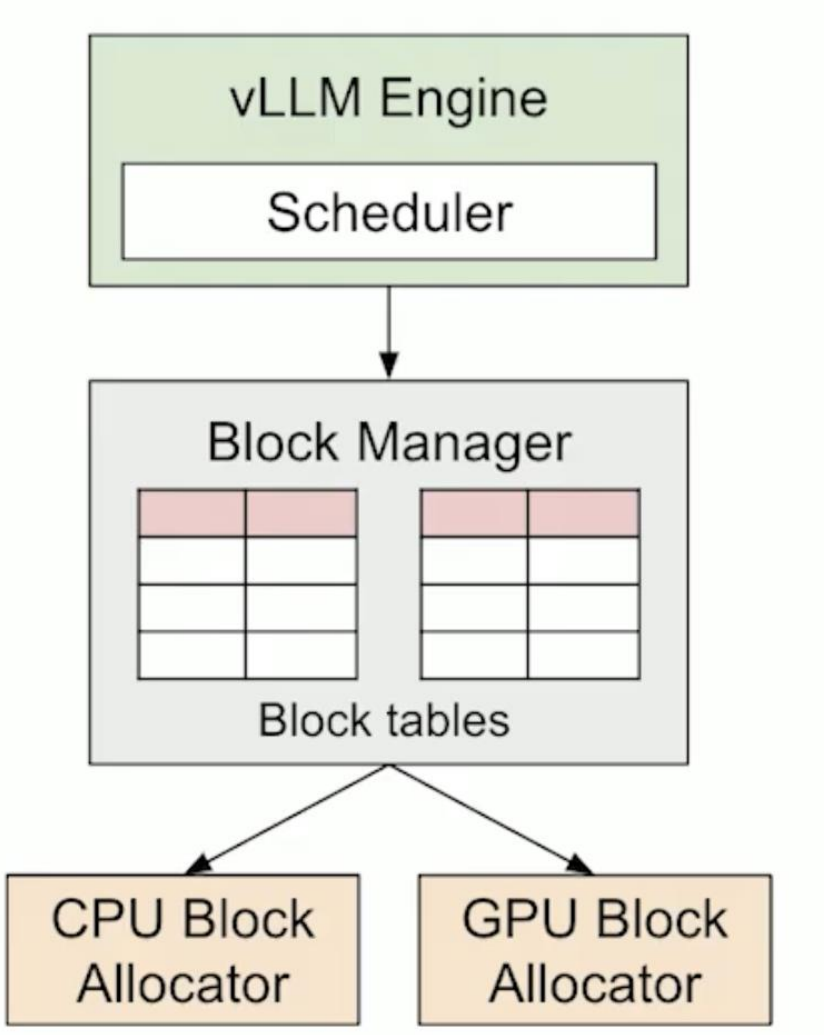
Single-level block table

- Block table is tiny compared to the actual data.

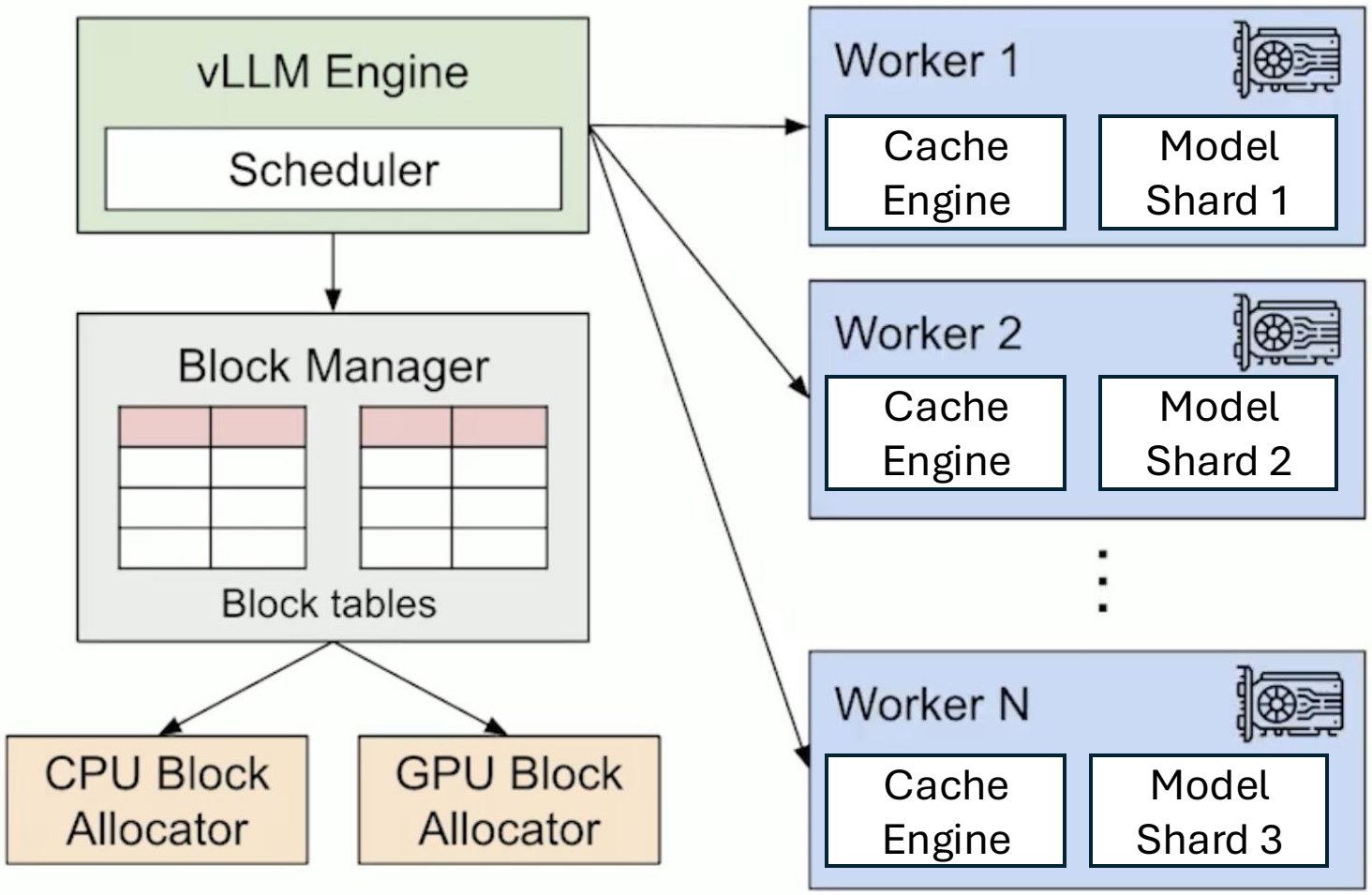
Preemption & Recovery

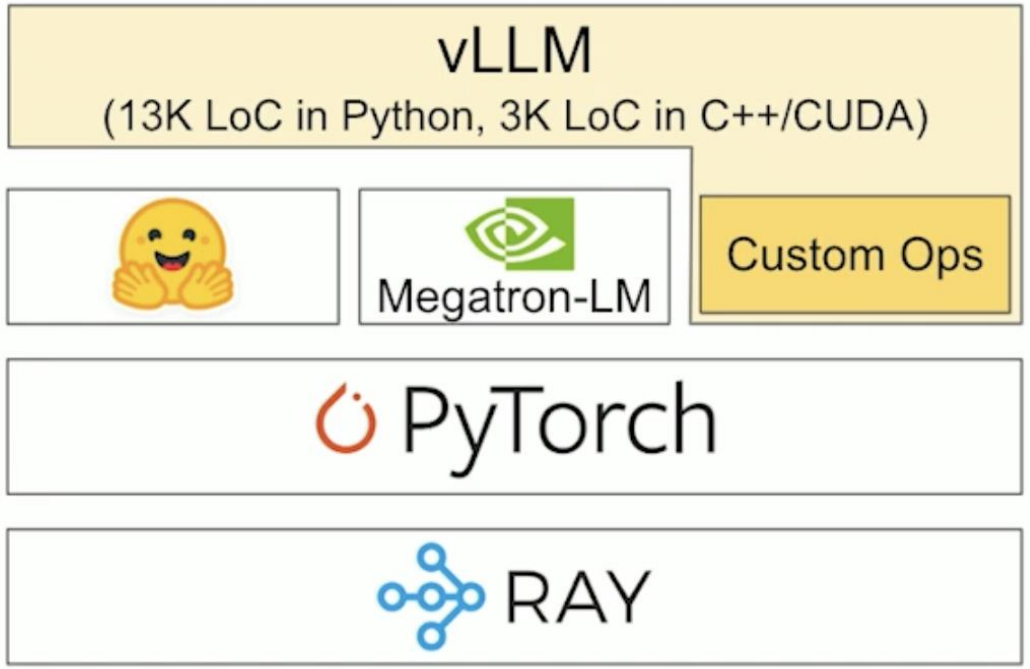
- Request-level preemption
- Recomputation-based recovery

vLLM Distributed System Architecture & Implementation



vLLM Distributed System Architecture & Implementation





Evaluation — Settings

Metric: Serving throughput

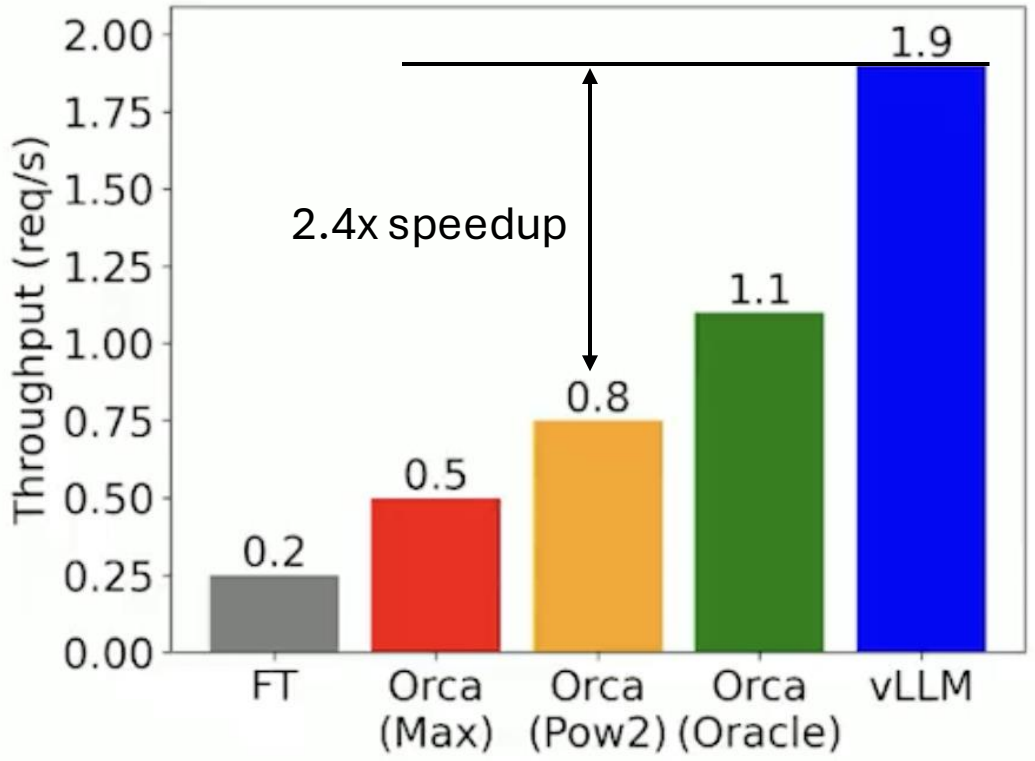
Input/Output Length Distribution

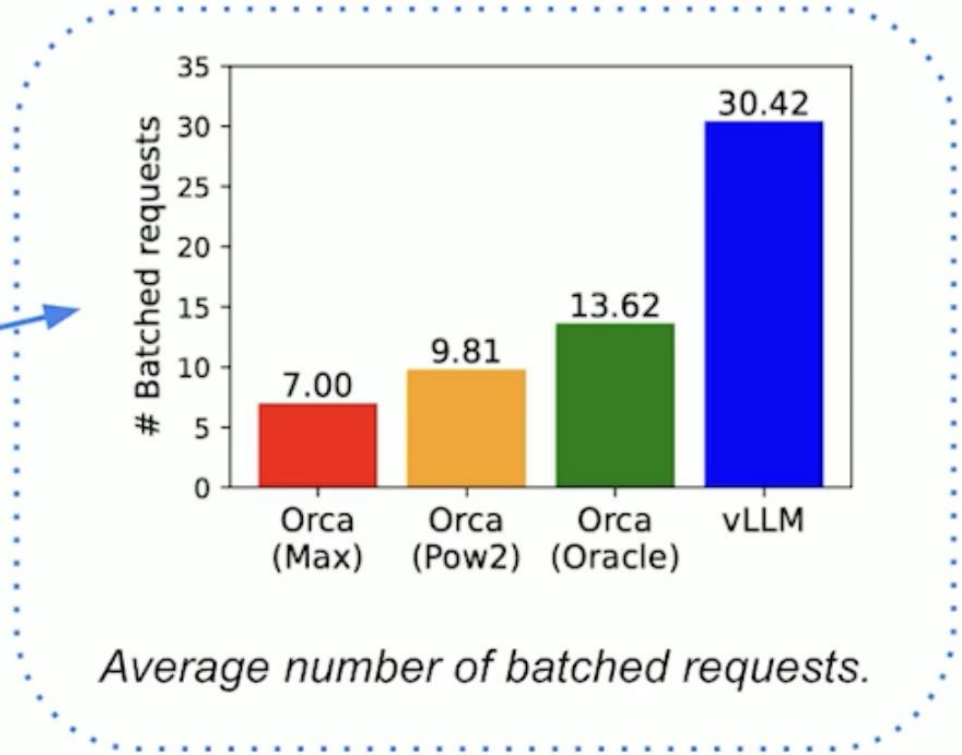
- Alpaca dataset (instruction-following)
- ShareGPT dataset (conversation)

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.

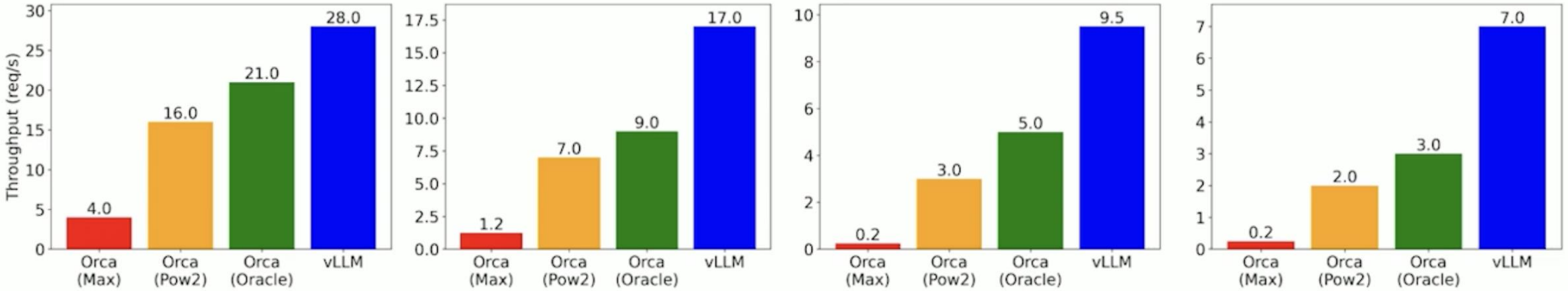
Throughput — Greedy Decoding





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

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*

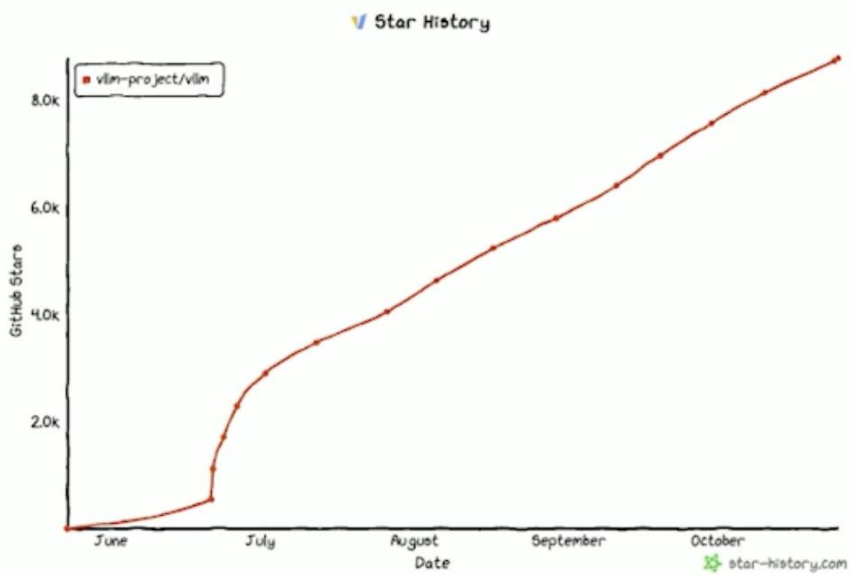
OPT-13B on 1xA100 40G with Alpaca trace.
Speedup: vLLM v.s. Orca(Pow2)

LLM 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

...

vLLM

- Improve memory efficiency by **2.5x–5x** by **reducing fragmentation** and **enabling sharing** with ***PagedAttention***
- Outperform SOTA by **up to 4x** in terms of **serving throughput**
- A vibrant open-source project that gets widely adopted


https://github.com/vllm-project/vllm

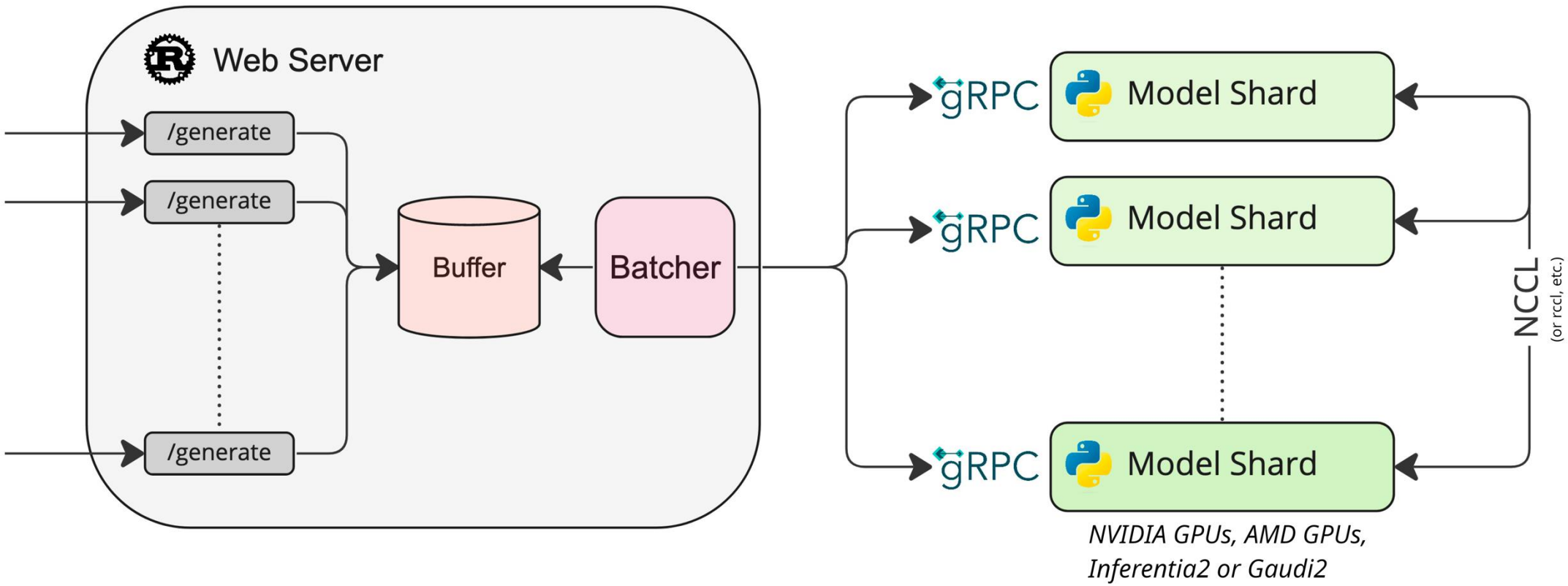
https://vllm.ai


https://arxiv.org/abs/2309.06180


https://discord.gg/jz7wjKhh6g

Text Generation Inference

Fast optimized inference for LLMs



microsoft/
DeepSpeed

DeepSpeed is a deep learning optimization library that makes distributed training and inference easy, efficient, and effective.

315 Contributors | 6k Used by | 193 Discussions | 32k Stars | 4k Forks

NVIDIA TensorRT

What is TensorRT?

- High-performance C++ library for GPU-accelerated inference
- Optimizes trained neural networks for faster execution on NVIDIA GPUs
- Supports models from major frameworks like TensorFlow, PyTorch, and ONNX

Key Features

- Graph optimizations and layer fusion
- Mixed precision capabilities (FP32, FP16, INT8, INT4)
- Automatic performance tuning
- Tensor and layer parallelism
- Support for various NVIDIA GPU architectures

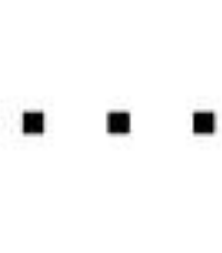
Performance Benefits

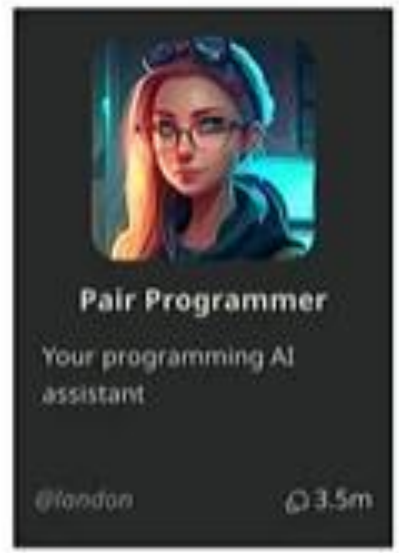
- Up to 36X faster inference compared to CPU-only platforms
- Significant latency reduction for real-time applications
- Improved throughput for data center deployments

SLoRA: Scalable Serving of Thousands of LoRA Adapters

Motivation: Customized Serving



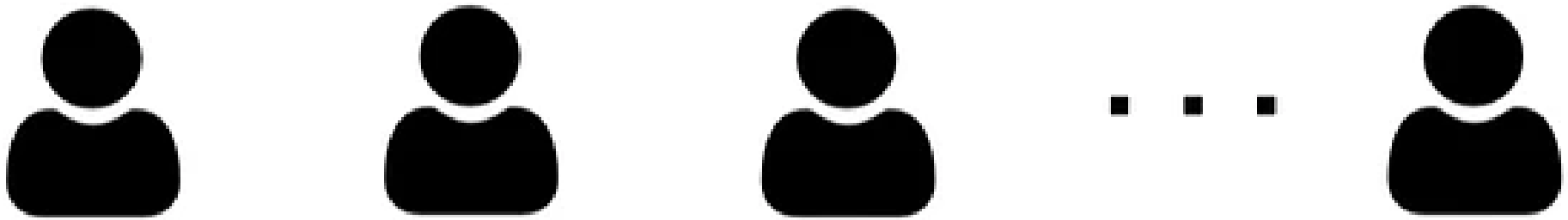



Motivation: Customized Serving

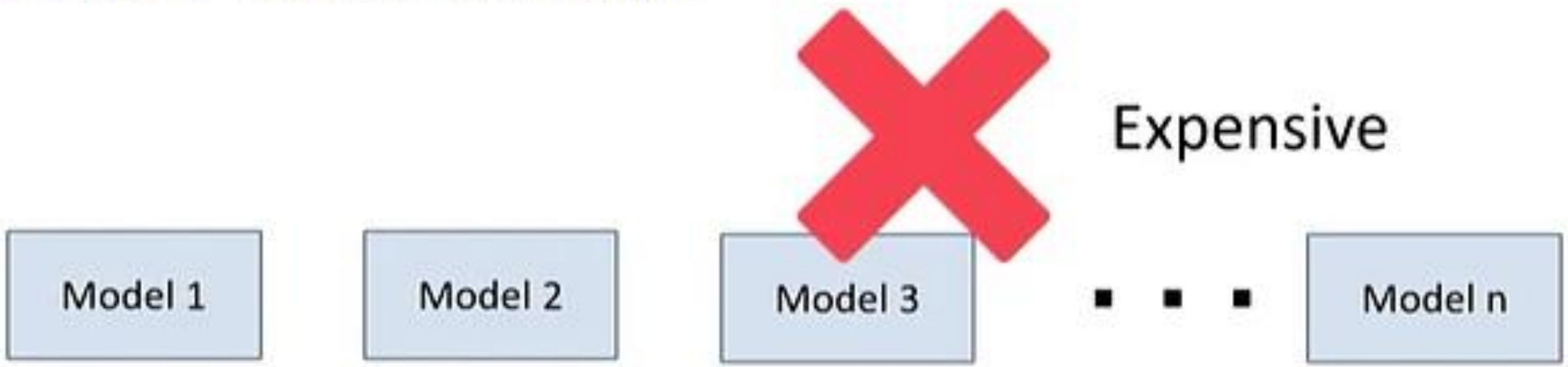
- Personalized writing style
- Personal habits and preferences
- Unique set of data

Naive: Full Fine-tuning for Each User



Naive: Full Fine-tuning for Each User

Not every user is active all the time



Expensive

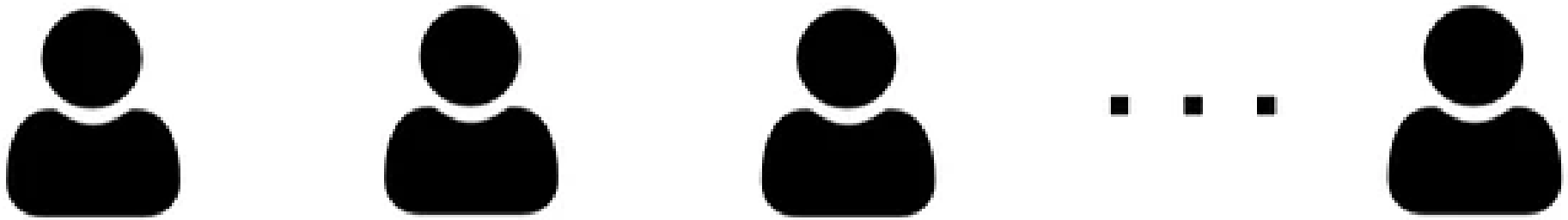
Naive: Full Fine-tuning for Each User

Not every user is active all the time

How do we let users with different needs share the resources?

Expensive

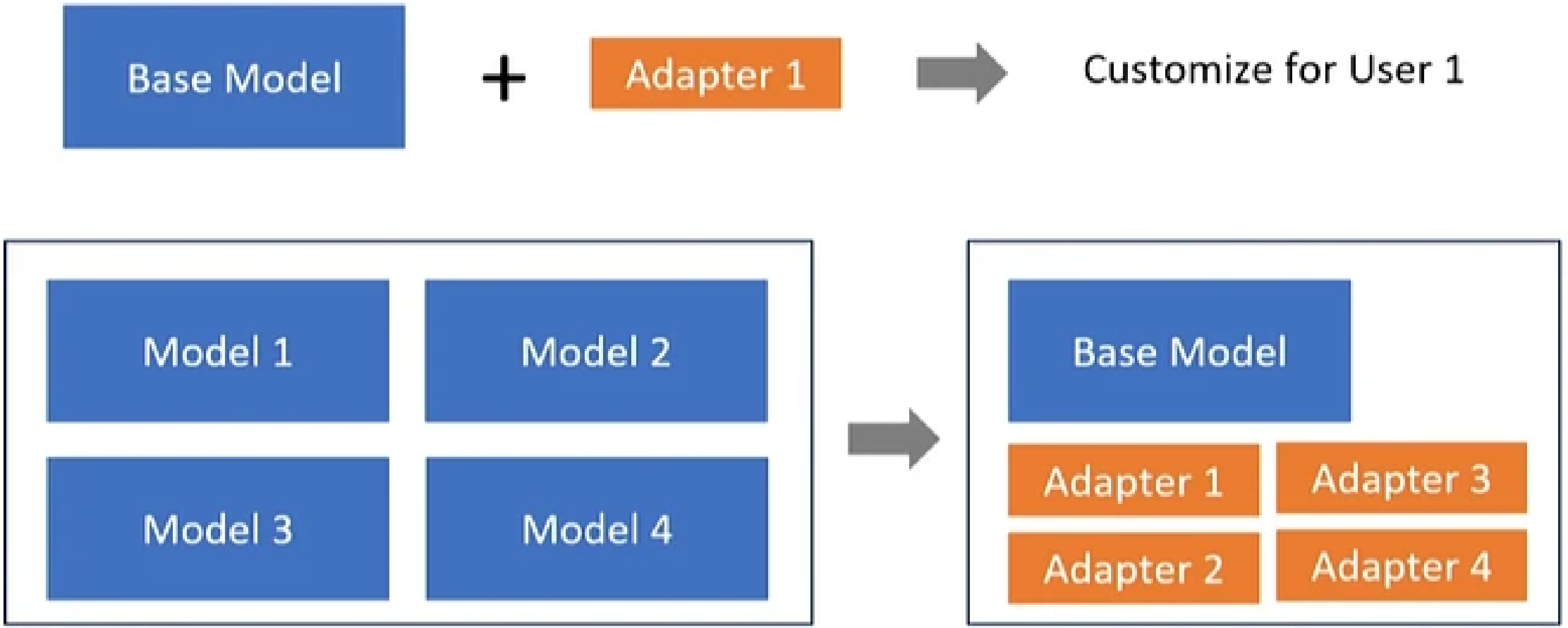
Desired: Share One Foundational Model



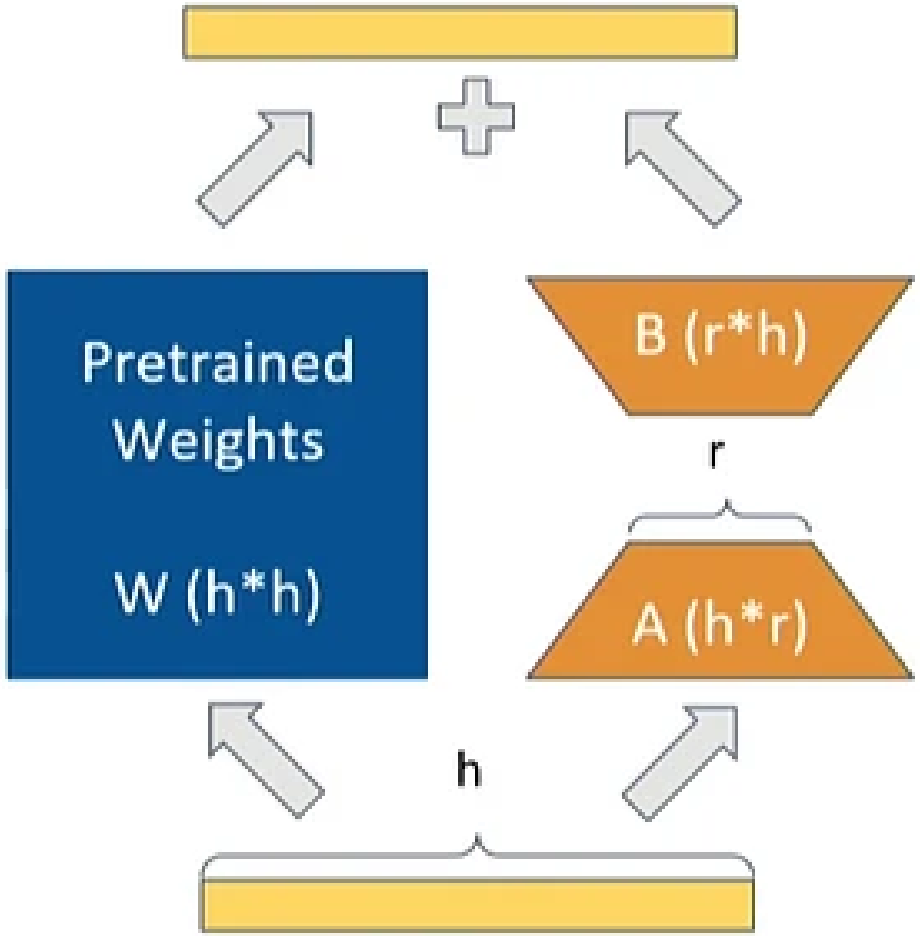
Base Model

Background: Low-Rank Adaptation (LoRA)

Background: Low-Rank Adaptation (LoRA)

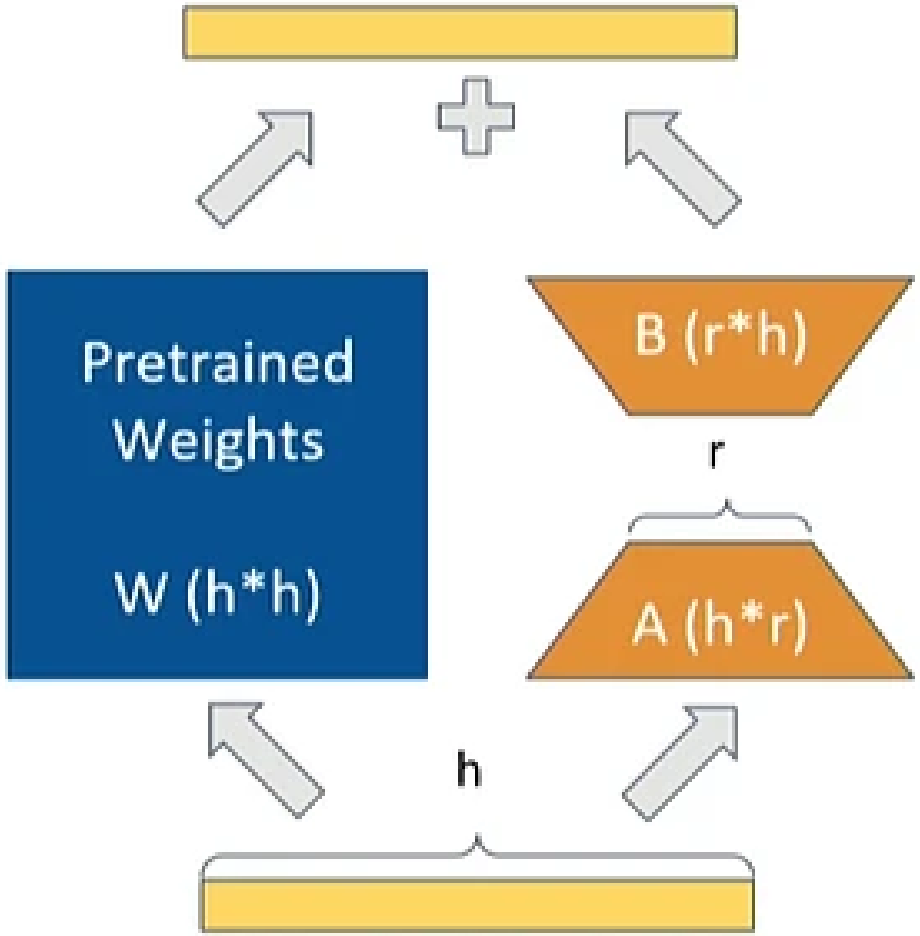


Background: Low-Rank Adaptation (LoRA)



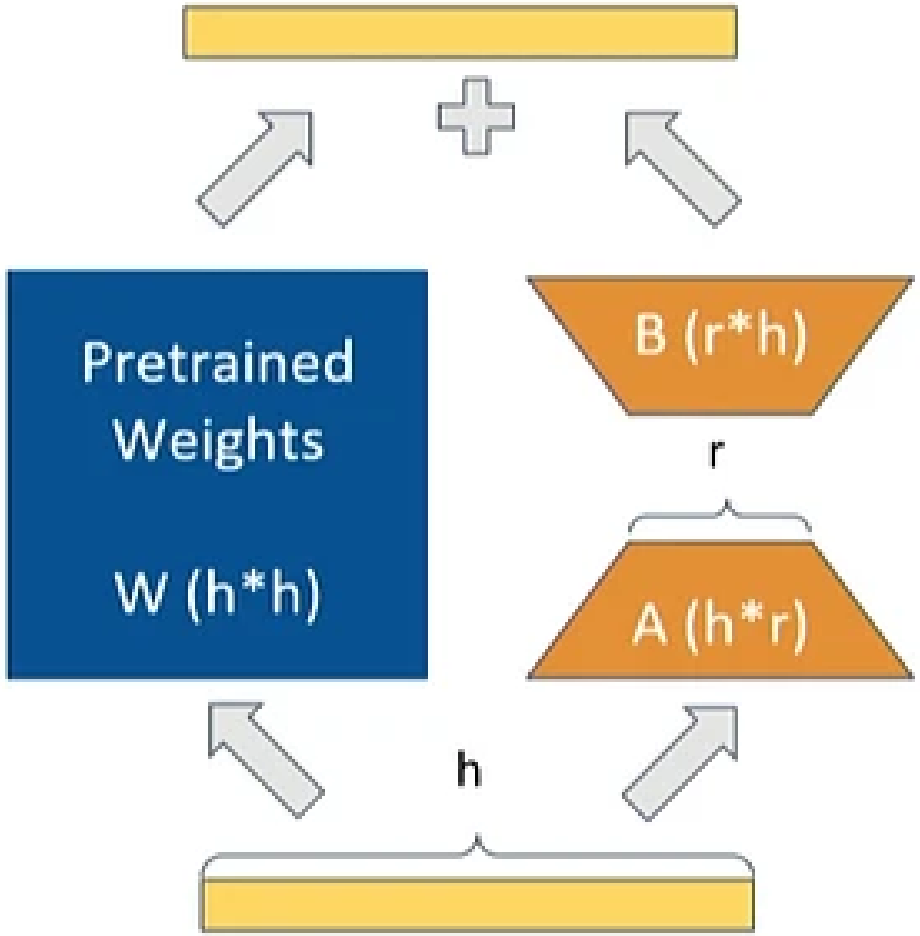
- Only update low-rank additive matrices (a small number of params)

Background: Low-Rank Adaptation (LoRA)



- Only update low-rank additive matrices (a small number of params)
- Performance on par with full-weight fine-tuning.

Background: Low-Rank Adaptation (LoRA)



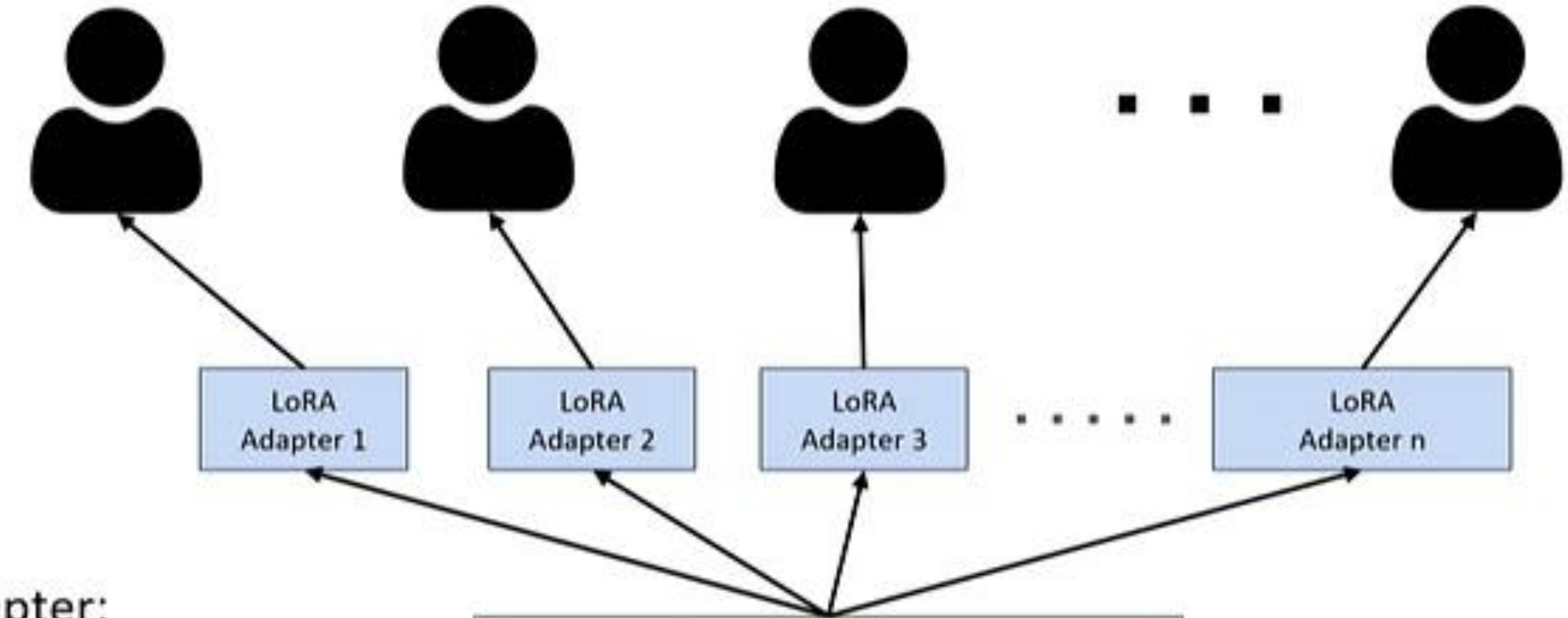
$$h = xW' = x(W + AB) \quad (1)$$
$$= xW + xAB. \quad (2)$$

- Only update low-rank additive matrices (a small number of params)
- Performance on par with full-weight fine-tuning.

Use Low-Rank Adaption (LoRA)

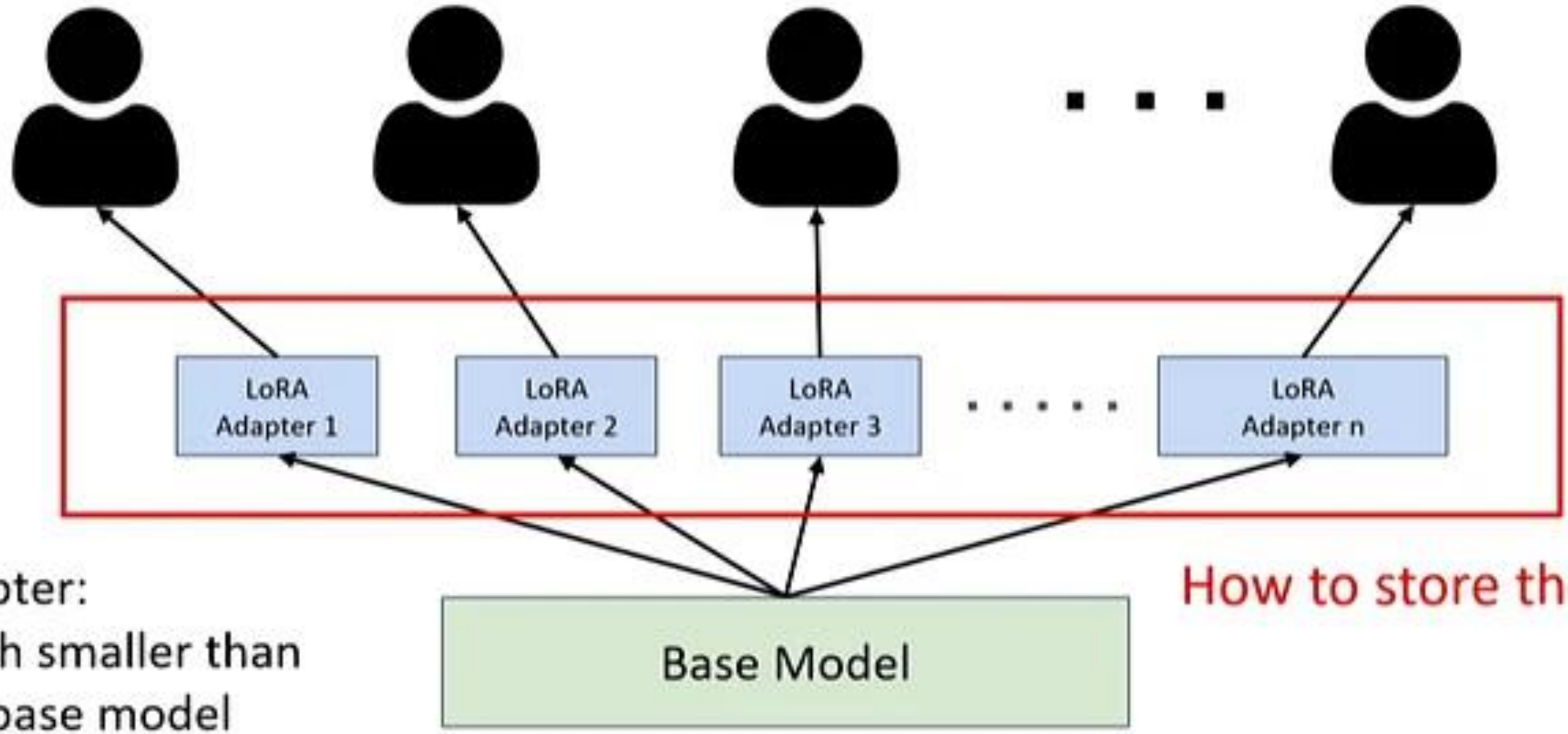
Adapter:
Much smaller than
the base model

Use Low-Rank Adaption (LoRA)



Adapter:
Much smaller than
the base model

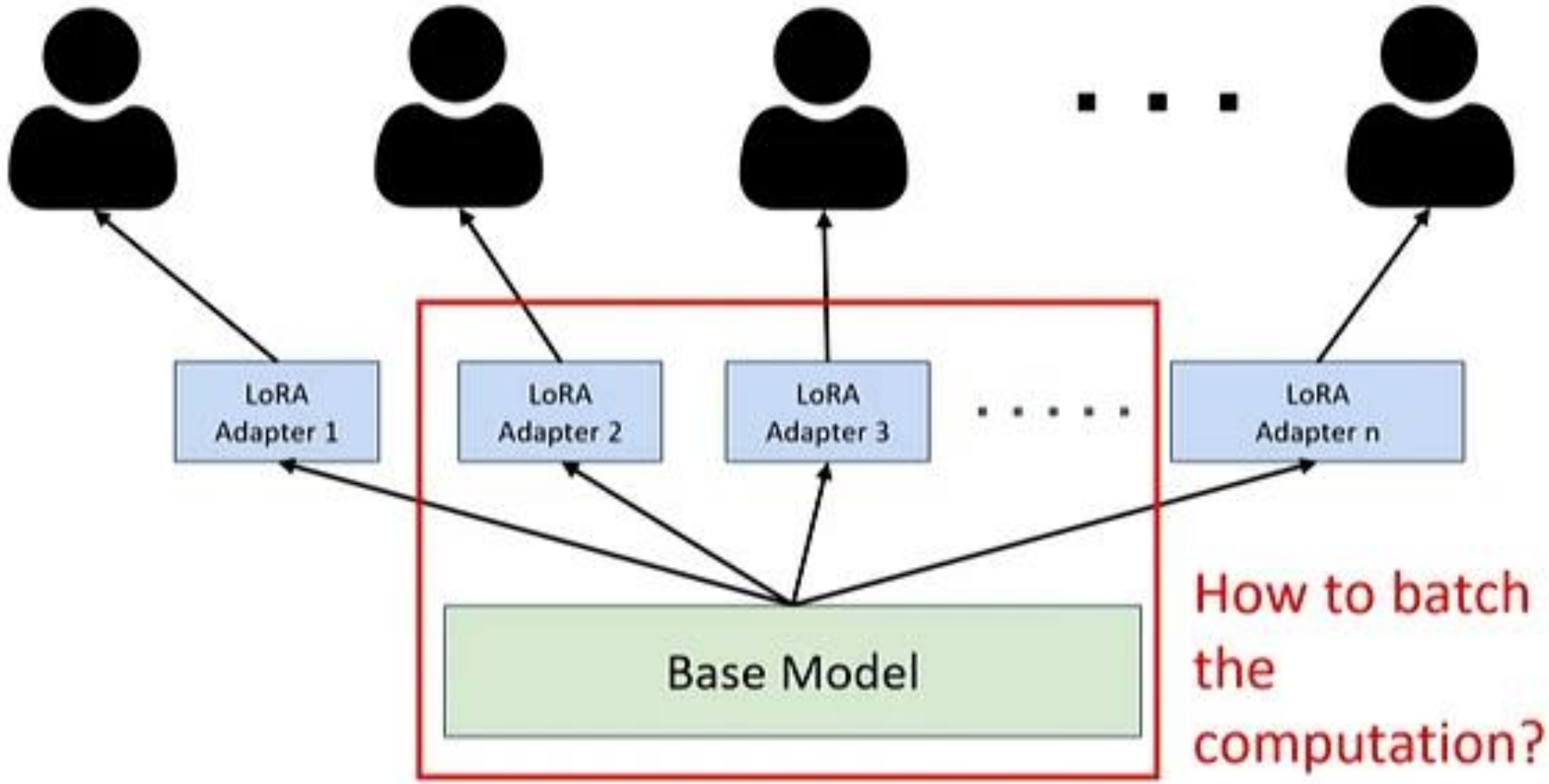
Use Low-Rank Adaption (LoRA)



Adapter:
Much smaller than the base model

How to store them?

Use Low-Rank Adaption (LoRA)



S-LoRA: Serving thousands of adapters on a single GPU

- ❑ **CPU/GPU Offloading**
- ❑ Unified Memory
- ❑ Heterogeneous Batching
  - ❑ Different adapters
  - ❑ Different ranks

- ➢ Adapters cannot fit on one GPU
- ➢ Or they occupy significant memory

S-LoRA: Serving thousands of adapters on a single GPU

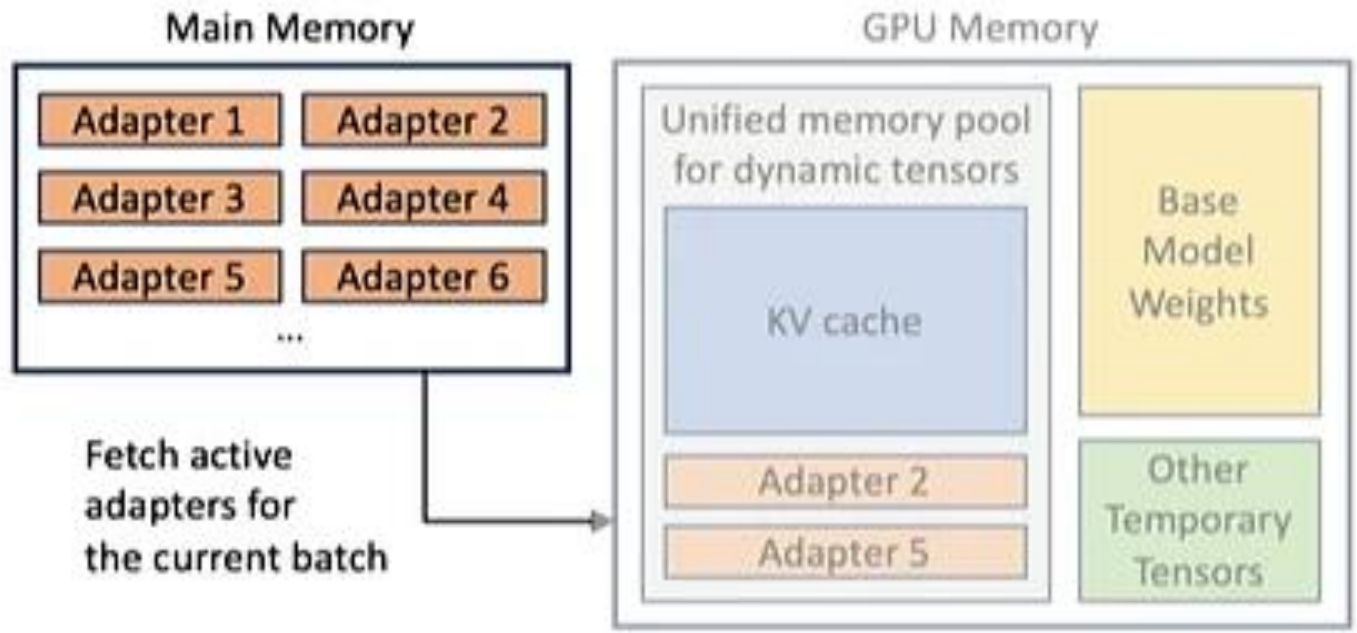
- ❑ **CPU/GPU Offloading**
- ❑ Unified Memory
- ❑ Heterogeneous Batching
  - ❑ Different adapters
  - ❑ Different ranks

- ➢ Adapters cannot fit on one GPU
- ➢ Or they occupy significant memory

Shrink the memory for the KV Cache
Limit the batch size

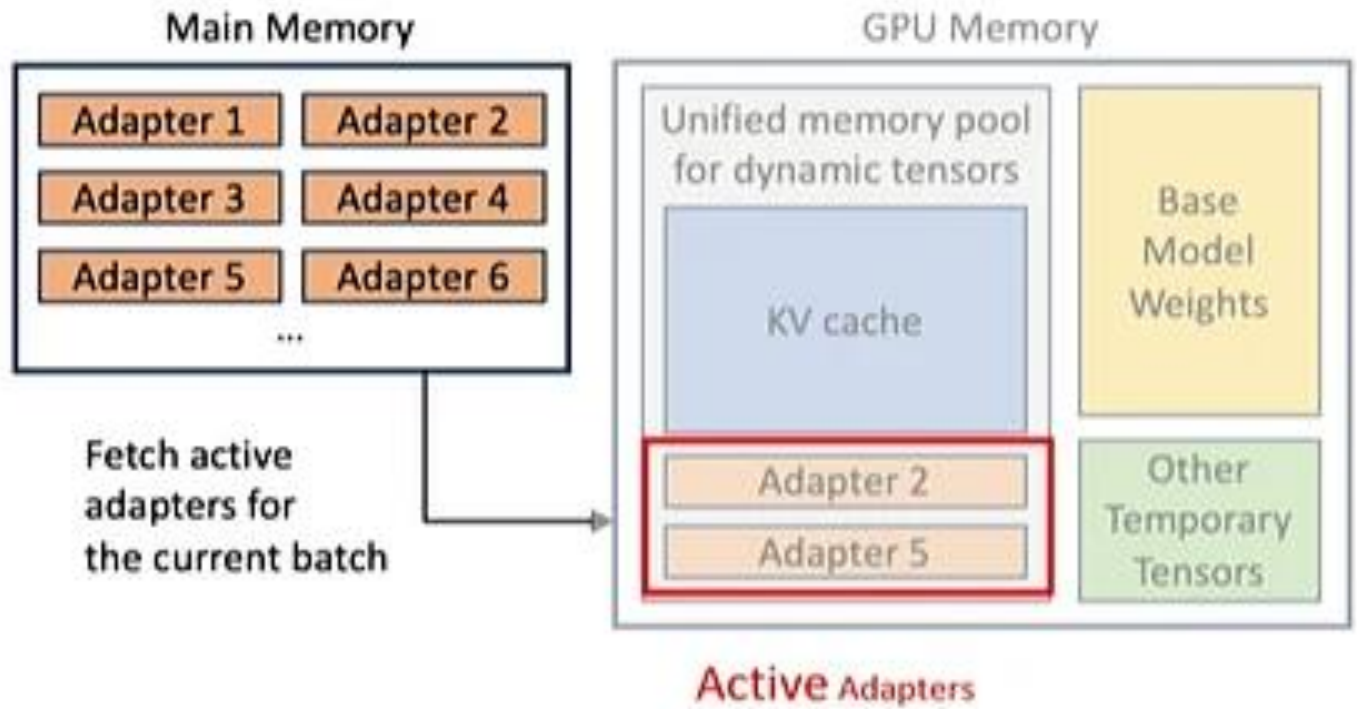
S-LoRA: Serving thousands of adapters on a single GPU

- ❑ **CPU/GPU Offloading**
- ❑ Unified Memory
- ❑ Heterogeneous Batching
  - ❑ Different adapters
  - ❑ Different ranks



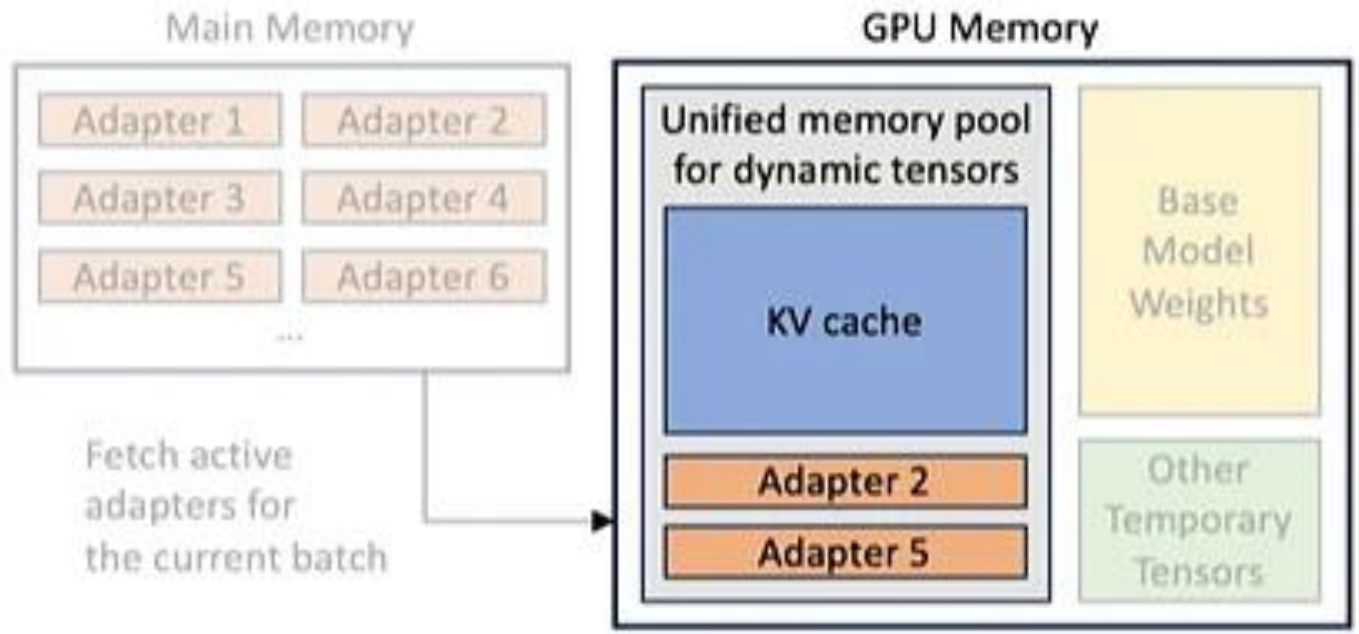
S-LoRA: Serving thousands of adapters on a single GPU

- ❑ **CPU/GPU Offloading**
- ❑ Unified Memory
- ❑ Heterogeneous Batching
  - ❑ Different adapters
  - ❑ Different ranks



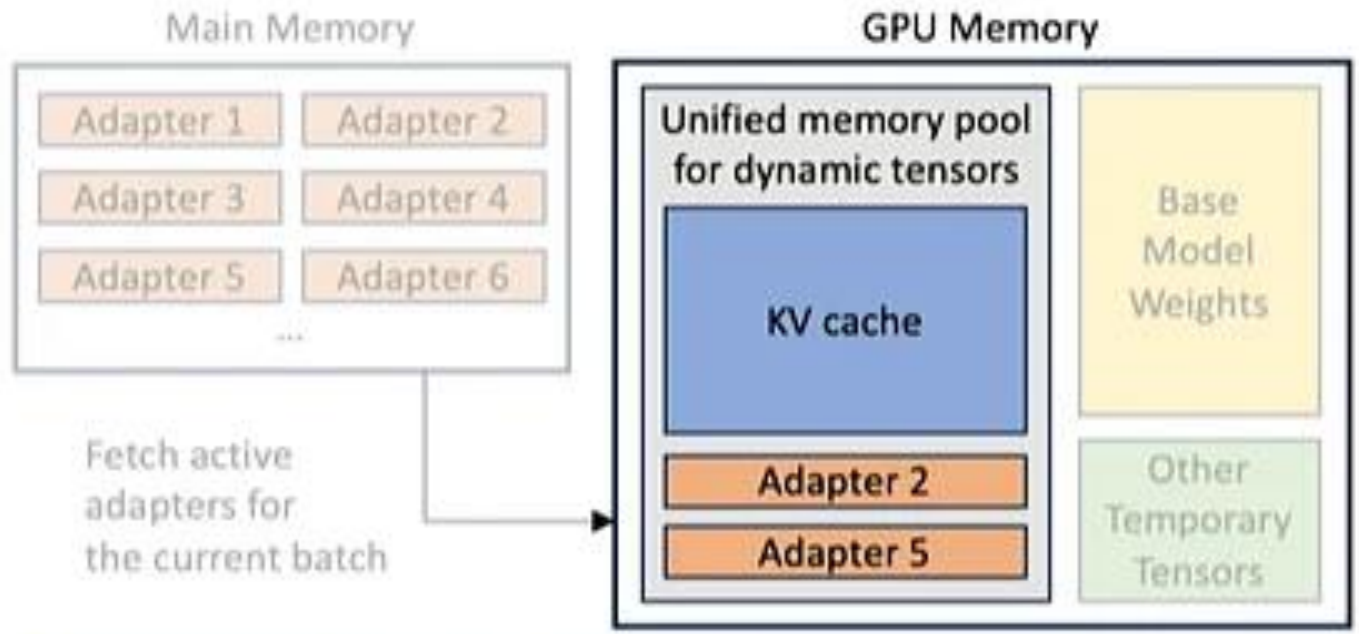
S-LoRA: Serving thousands of adapters on a single GPU

- ❑ CPU/GPU Offloading
- ❑ **Unified Memory**
- ❑ Heterogeneous Batching
  - ❑ Different adapters
  - ❑ Different ranks



S-LoRA: Serving thousands of adapters on a single GPU

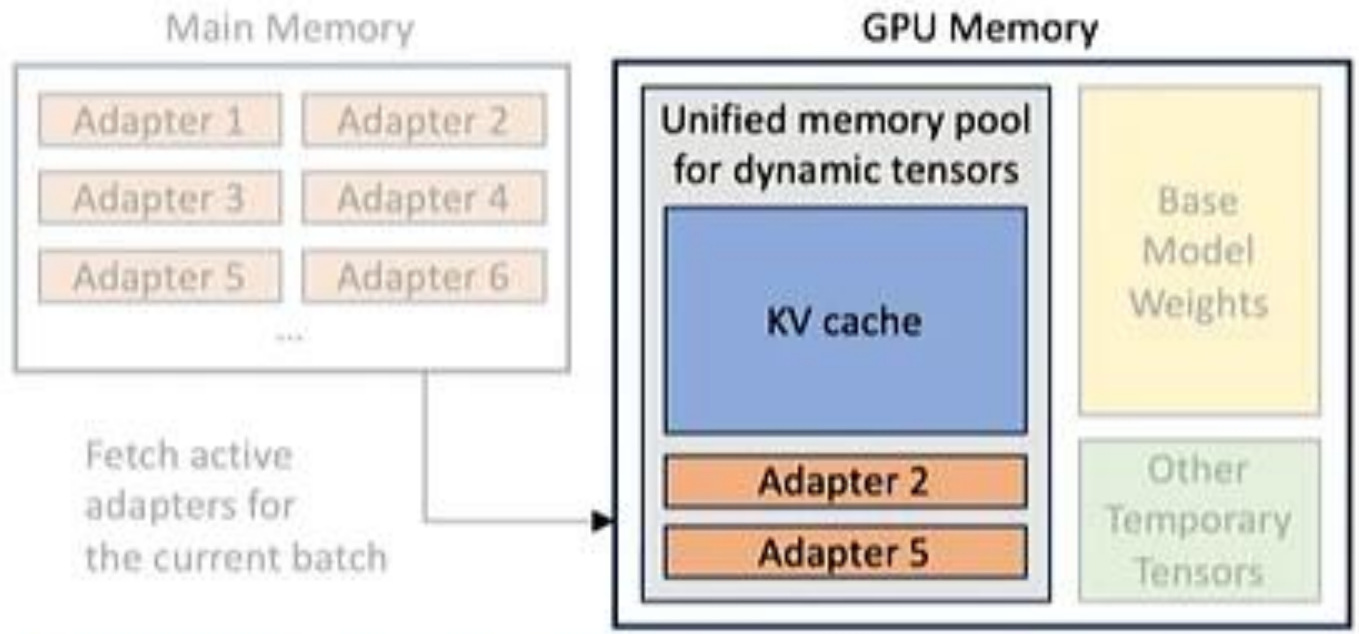
- ❑ CPU/GPU Offloading
- ❑ **Unified Memory**
- ❑ Heterogeneous Batching
  - ❑ Different adapters
  - ❑ Different ranks



➢ Memory use for KV cache and adapters are dynamic

S-LoRA: Serving thousands of adapters on a single GPU

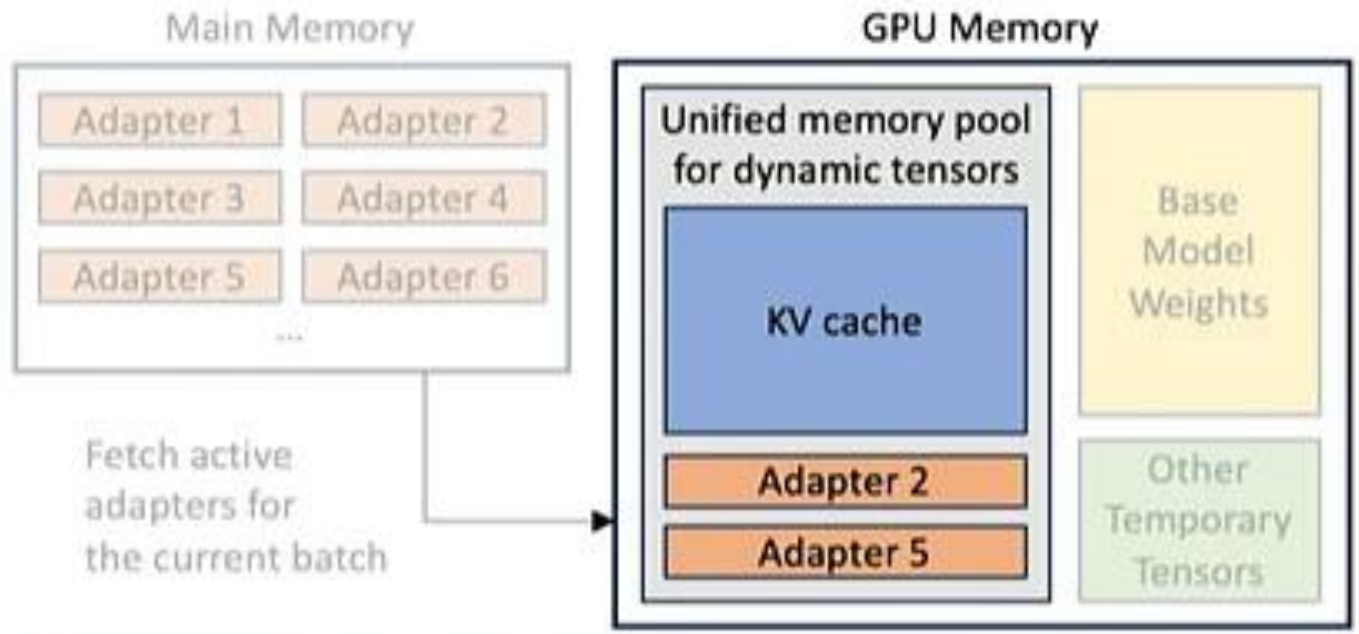
- ❑ CPU/GPU Offloading
- ❑ **Unified Memory**
- ❑ Heterogeneous Batching
  - ❑ Different adapters
  - ❑ Different ranks



- ➢ Memory use for KV cache and adapters are dynamic
- ➢ Separate memory pool cause more fragmentation

S-LoRA: Serving thousands of adapters on a single GPU

- ❑ CPU/GPU Offloading
- ❑ **Unified Memory**
- ❑ Heterogeneous Batching
  - ❑ Different adapters
  - ❑ Different ranks

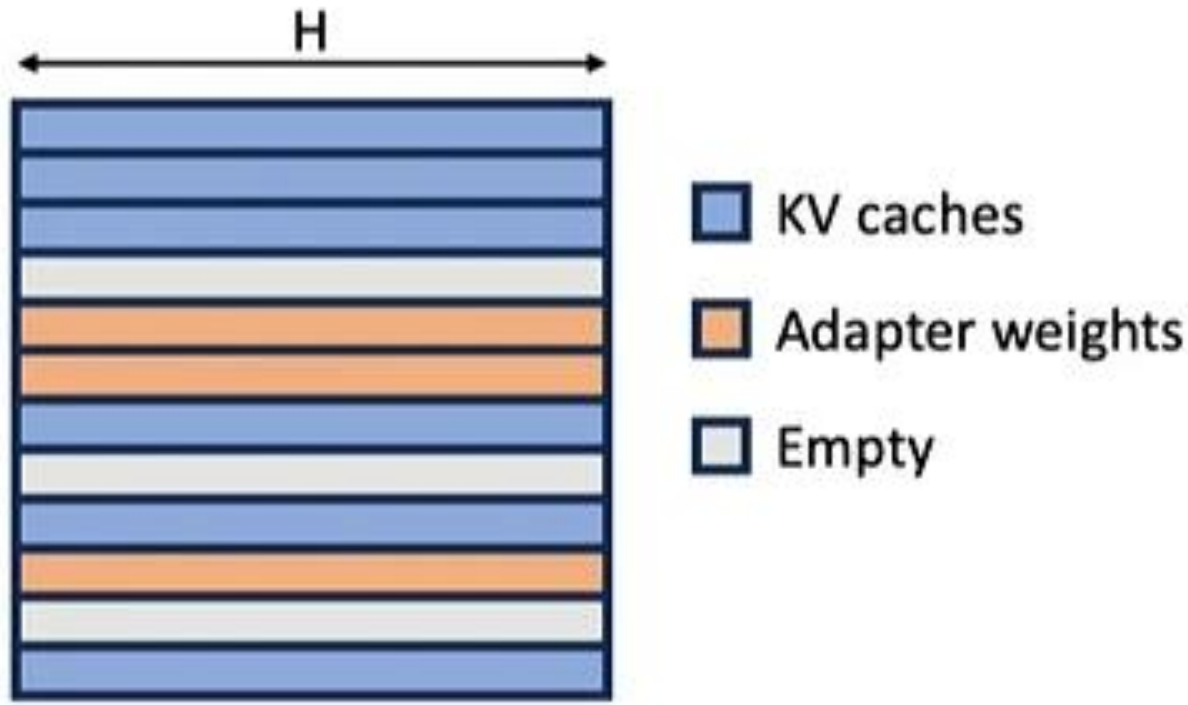


- ➢ Memory use for KV cache and adapters are dynamic
- ➢ Separate memory pool cause more fragmentation

Unified memory for dynamic tensors: Memory fragmentation -> 0

S-LoRA: Serving thousands of adapters on a single GPU

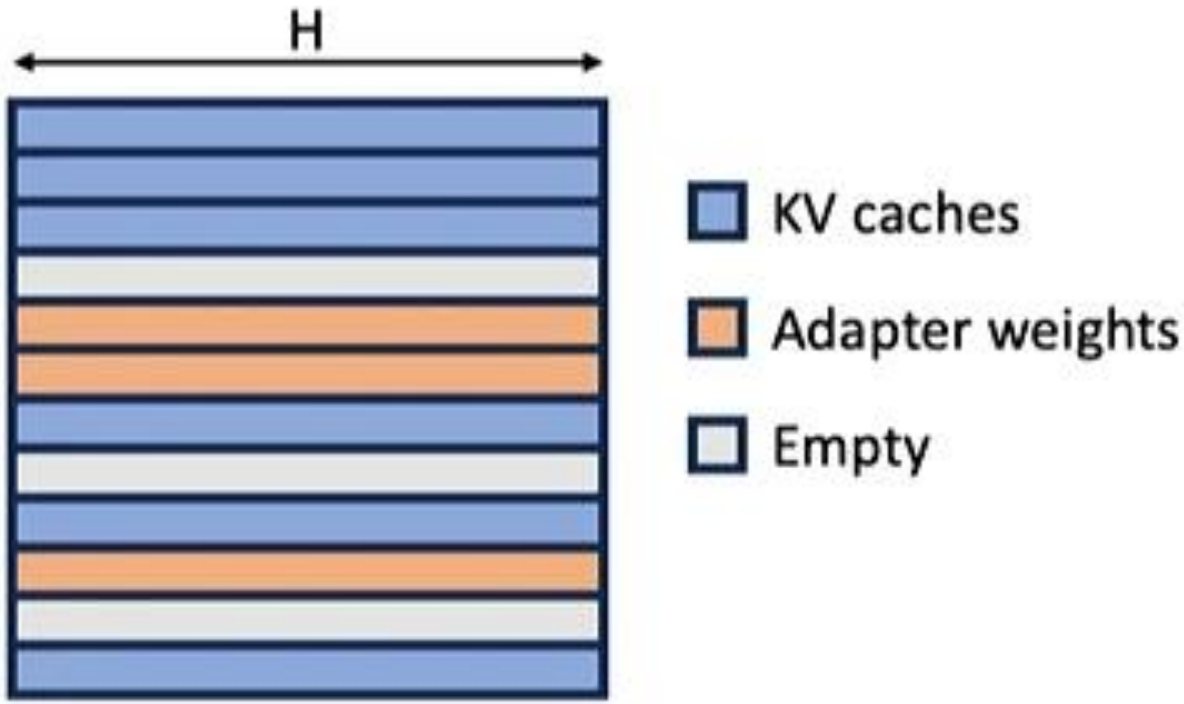
- ❑ CPU/GPU Offloading
- ❑ **Unified Memory**
- ❑ Heterogeneous Batching
  - ❑ Different adapters
  - ❑ Different ranks



Dynamic memory size for KV caches and adapters

S-LoRA: Serving thousands of adapters on a single GPU

- ❑ CPU/GPU Offloading
- ❑ **Unified Memory**
- ❑ Heterogeneous Batching
  - ❑ Different adapters
  - ❑ Different ranks



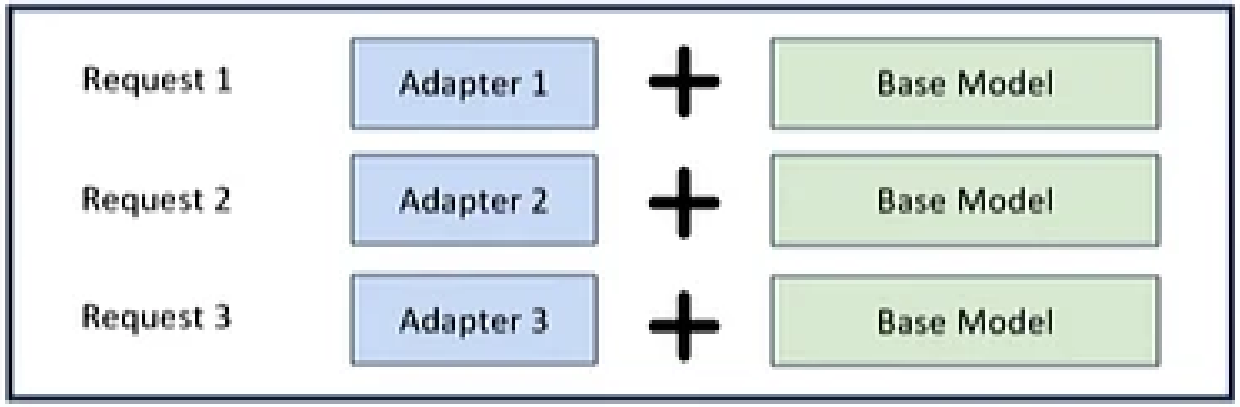
Dynamic memory size for KV caches and adapters

Paged KV cache (PagedAttention[1]) -> Unify paged KV cache and paged adapter weights

[1] vLLM: Woosuk Kwon*, Zhuohan Li*, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, Ion Stoica. SOSP'23

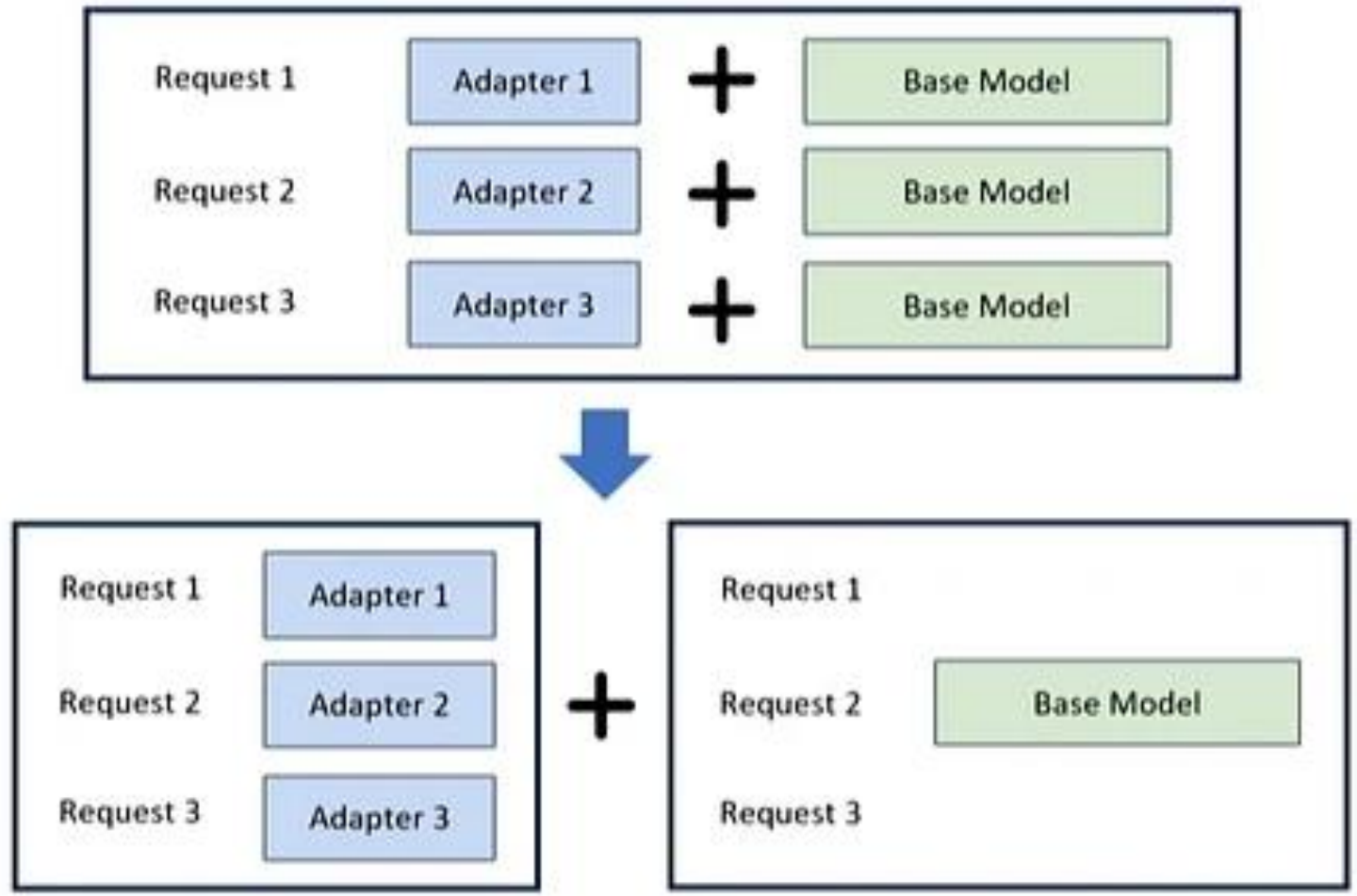
S-LoRA: Serving thousands of adapters on a single GPU

- CPU/GPU Offloading
- Unified Memory
- **Heterogeneous Batching**
  - Different adapters
  - Different ranks



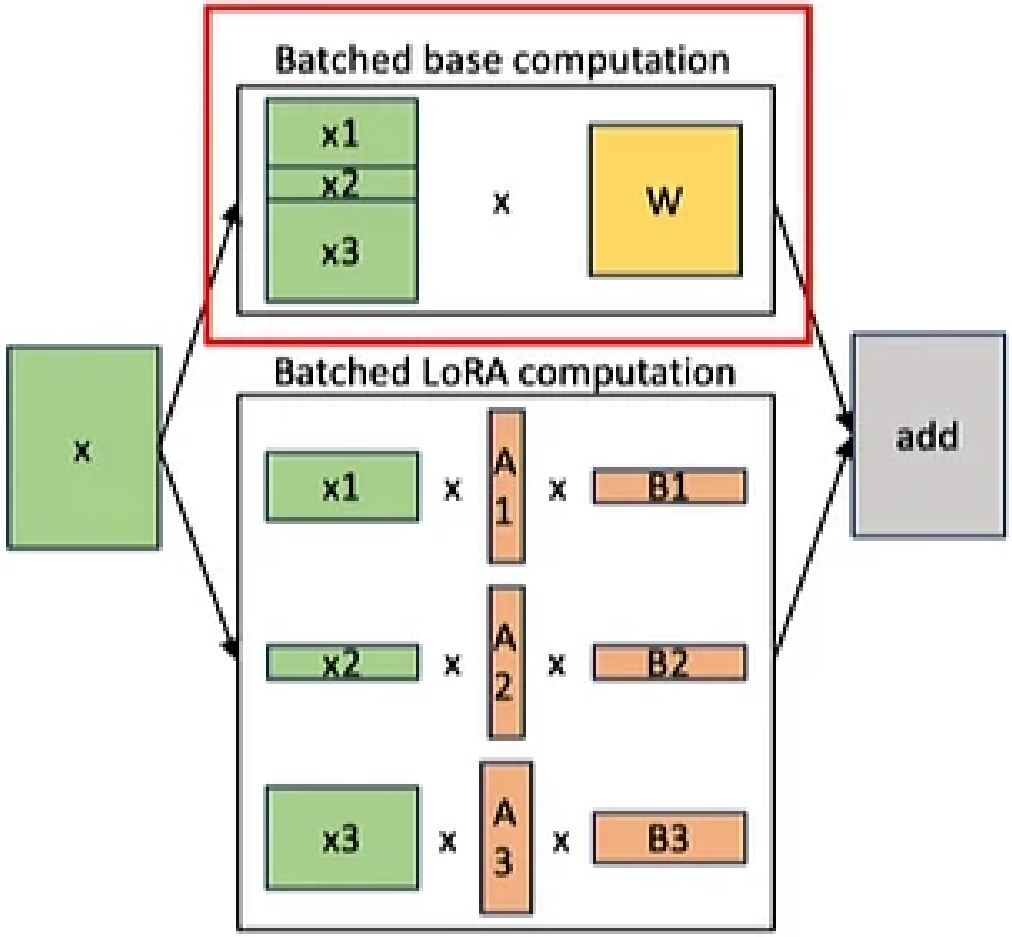
S-LoRA: Serving thousands of adapters on a single GPU

- ❑ CPU/GPU Offloading
- ❑ Unified Memory
- ❑ **Heterogeneous Batching**
  - ❑ Different adapters
  - ❑ Different ranks



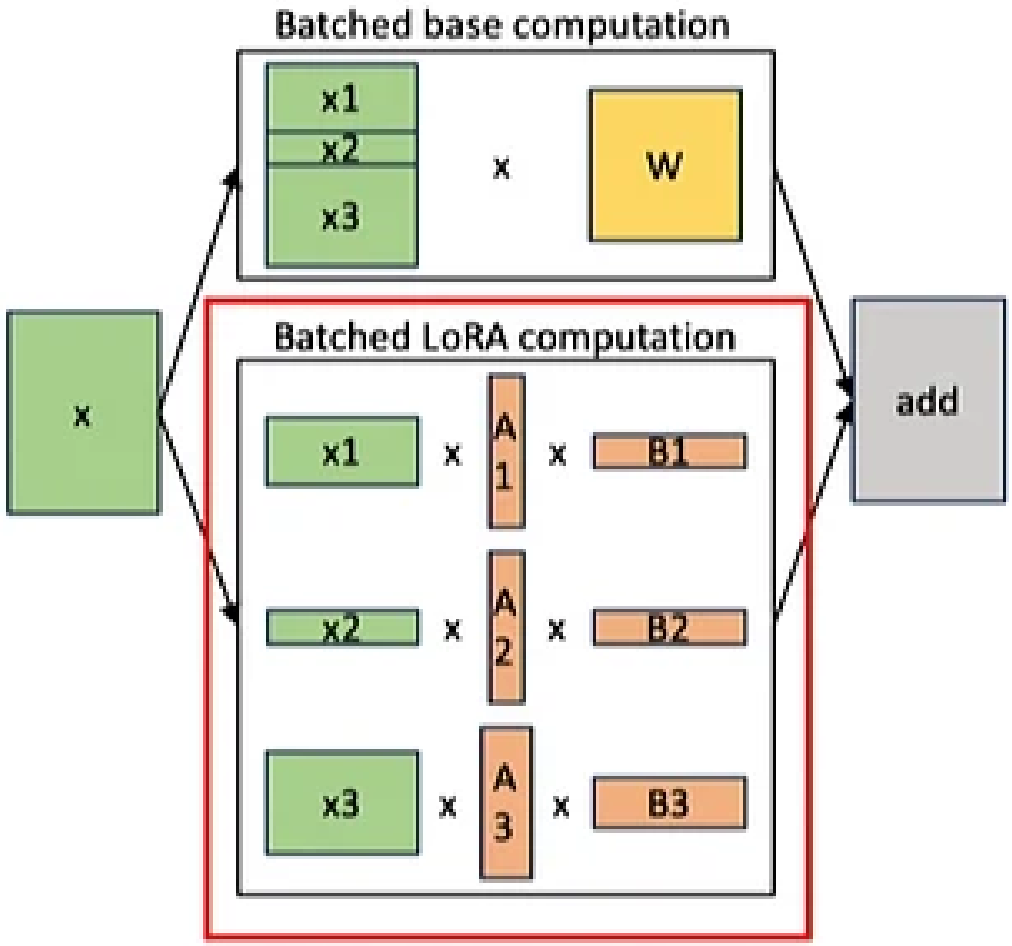
Batching different adapters use customized kernel

(extend from punica)



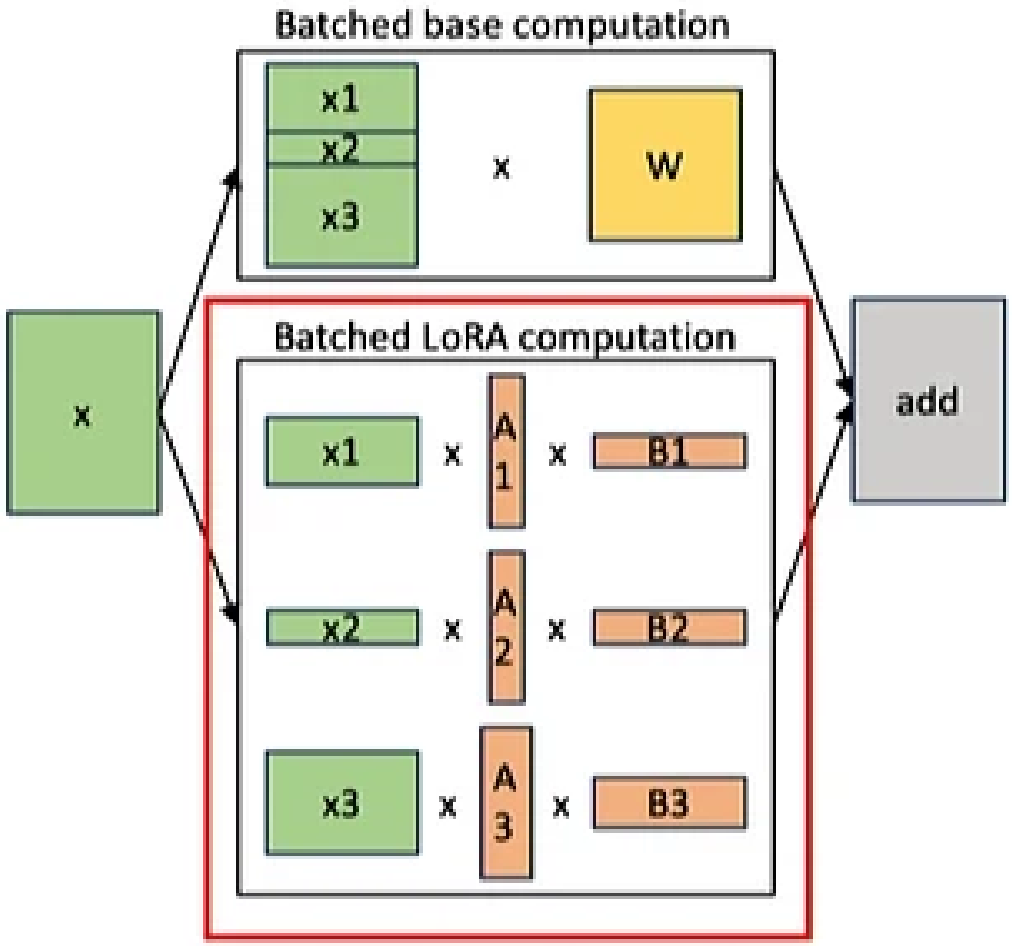
Batching different adapters use customized kernel

(extend from punica)



Batching different adapters use customized kernel

(extend from punica)

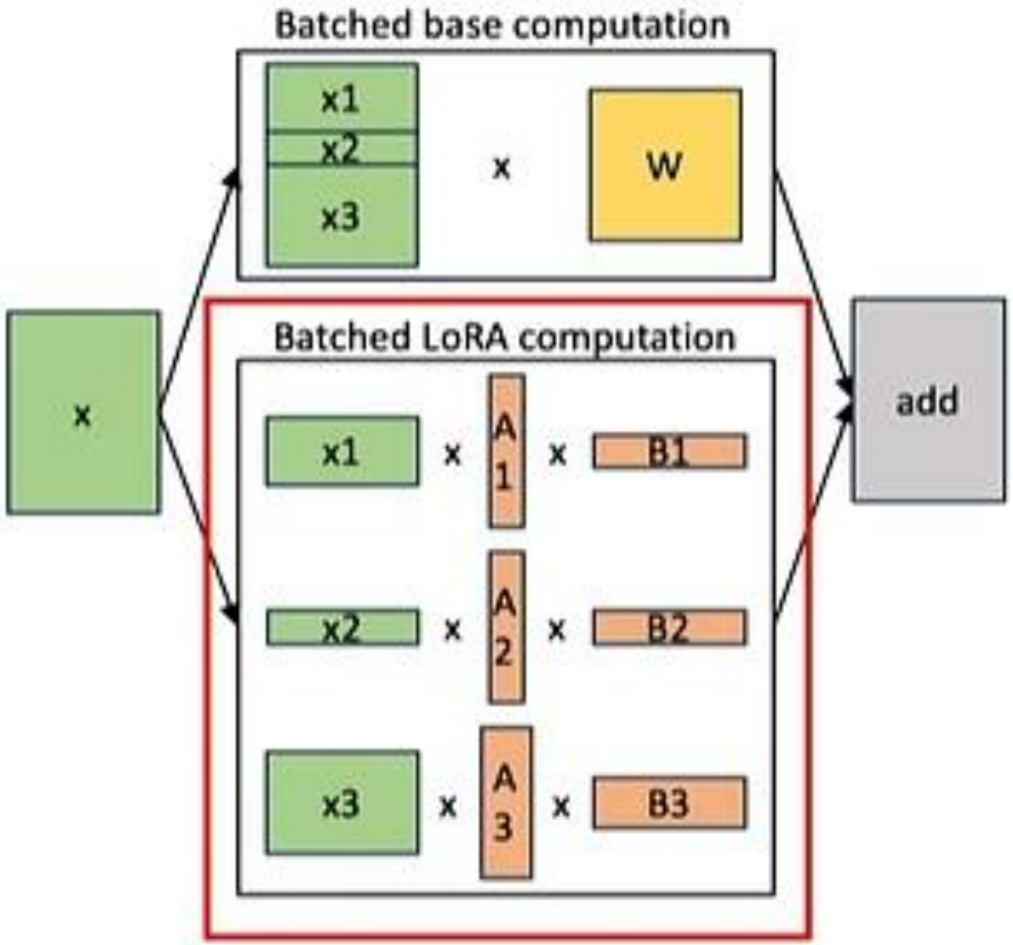


Example: xA for decode

```
x: [B, H], A_i: [H, R_i]
```

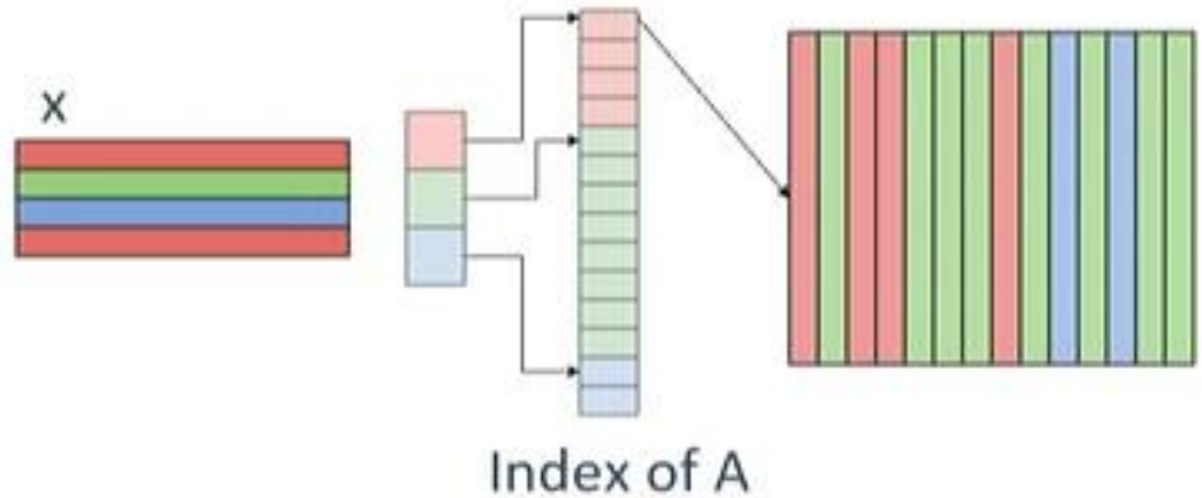
Batching different adapters use customized kernel

(extend from punica)



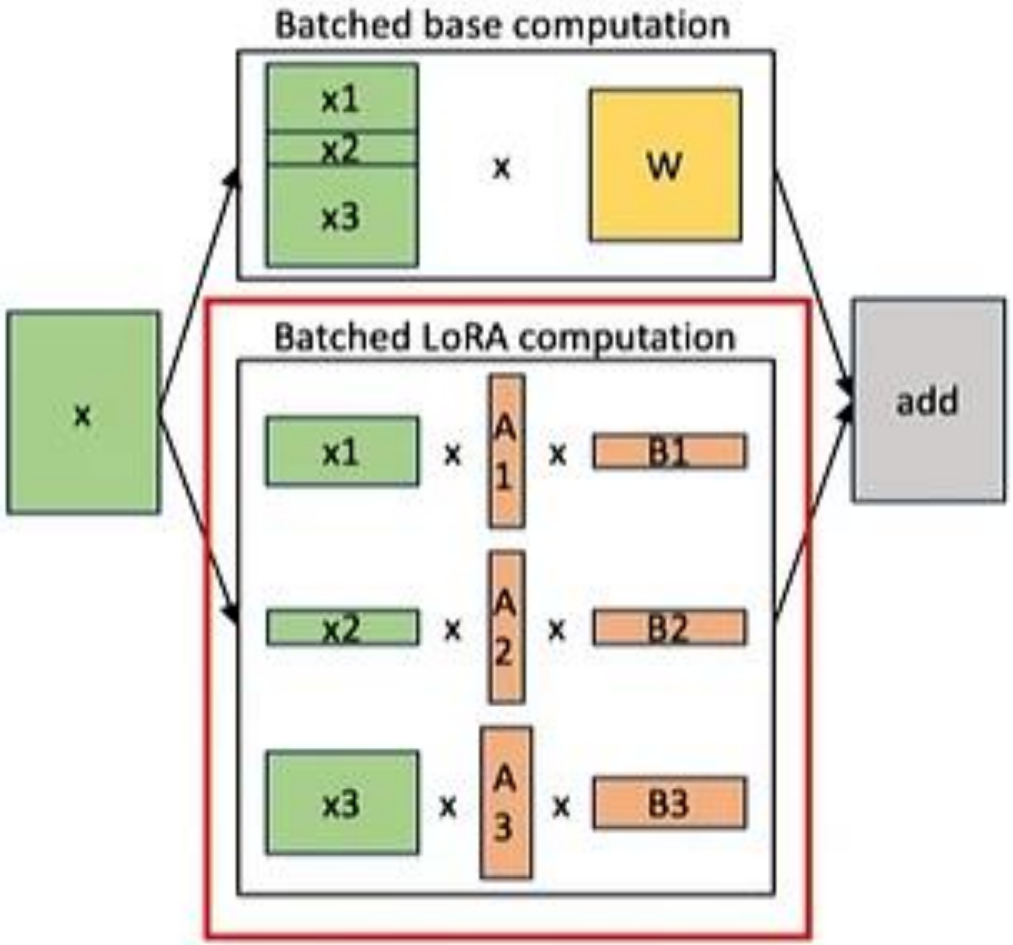
Example: xA for decode

```
x: [B, H], A_i: [H, R_i]
```



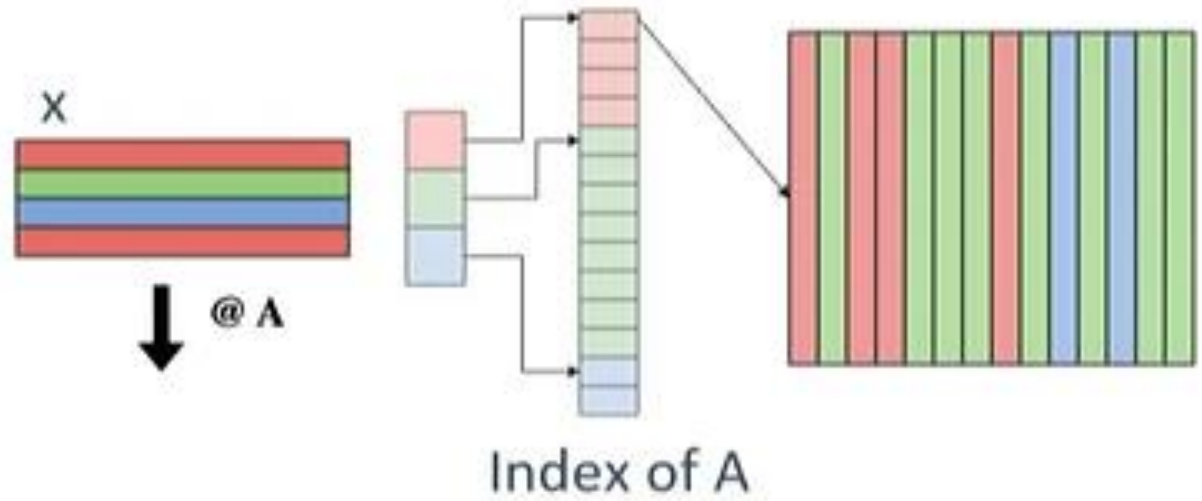
Batching different adapters use customized kernel

(extend from punica)



Example: xA for decode

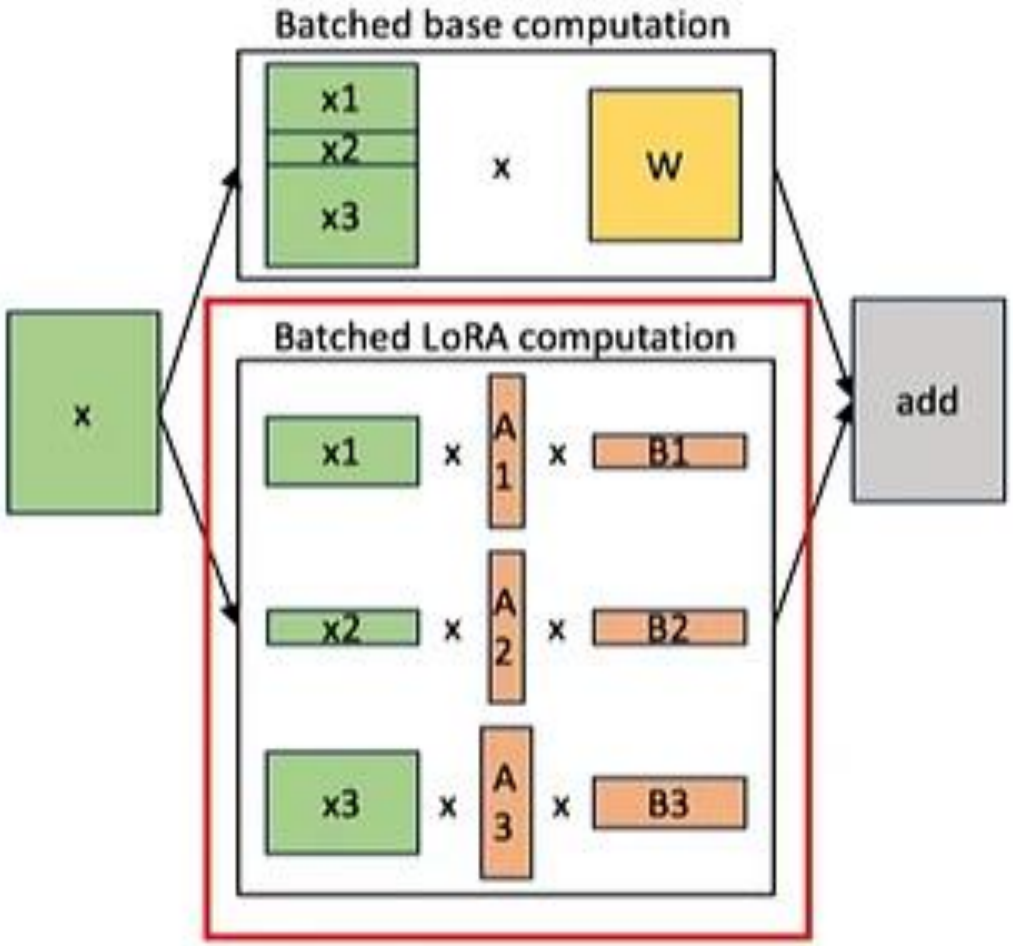
```
x: [B, H], A_i: [H, R_i]
```



Parallel each H*H vector-vector multiplication

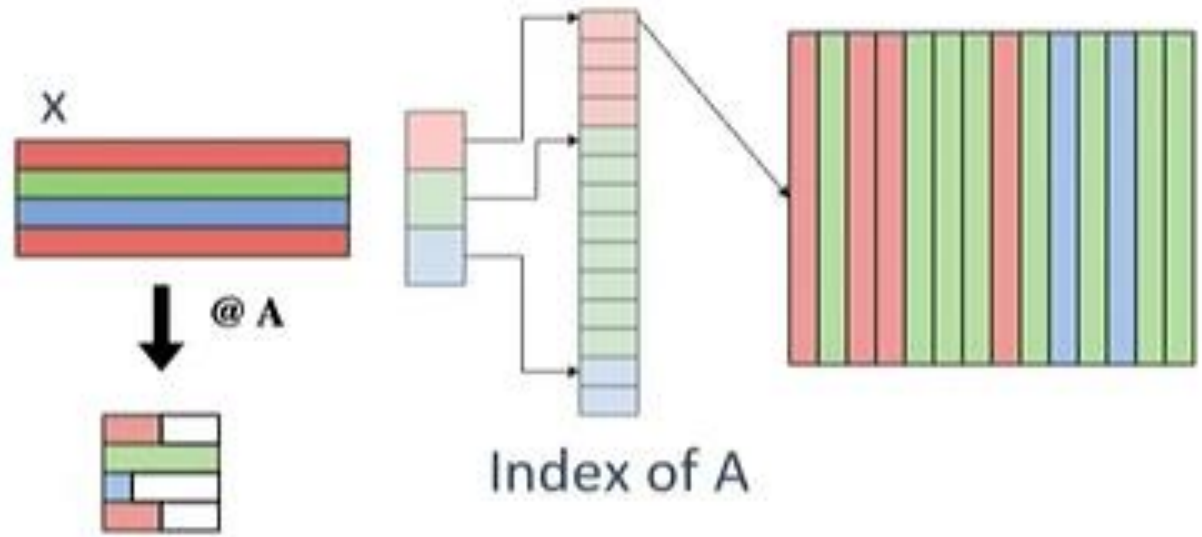
Batching different adapters use customized kernel

(extend from punica)



Example: xA for decode

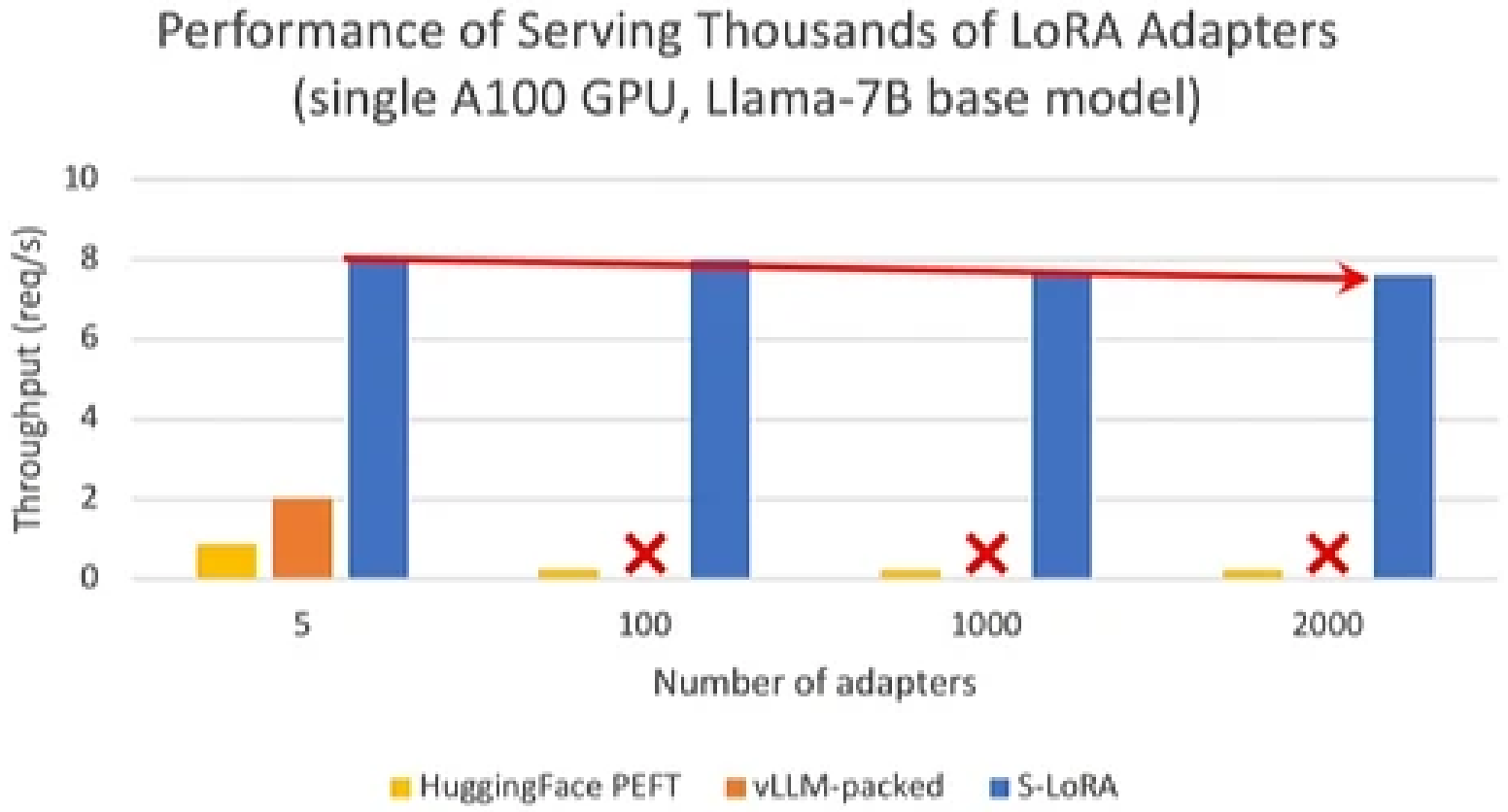
```
x: [B, H], A_i: [H, R_i]
```



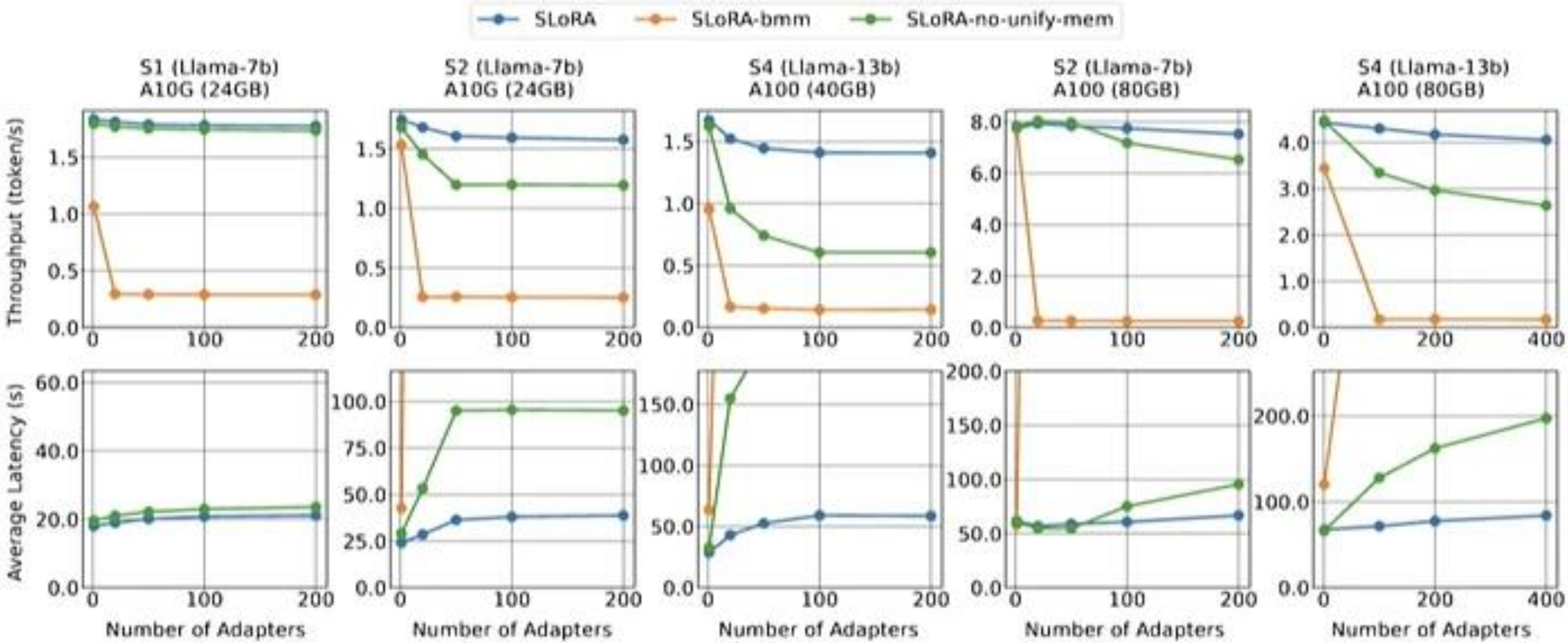

```
Grid = [max(R_i), B]
```

S-LoRA: Serving thousands of adapters on a single GPU

- Serve 100x more users
- 1000x users "for free"



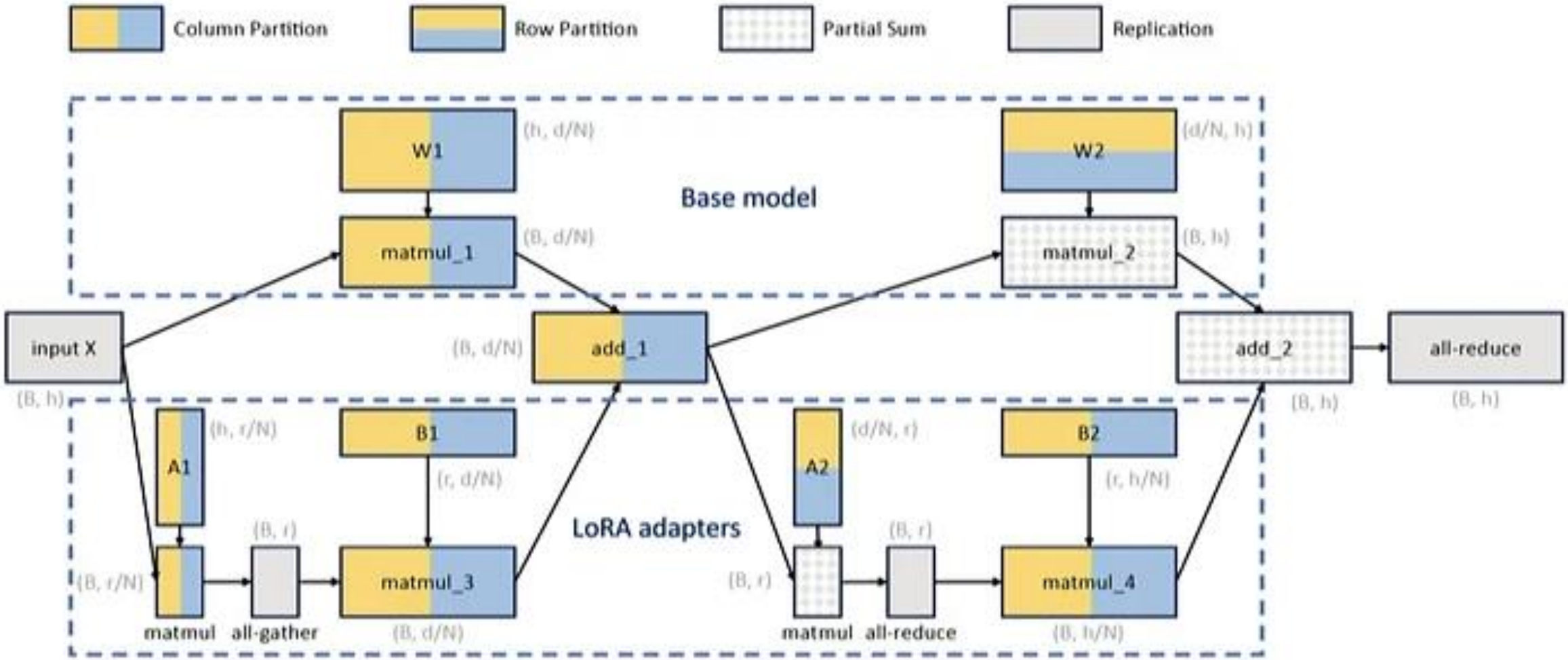
Evaluation



Real world trace give similar results

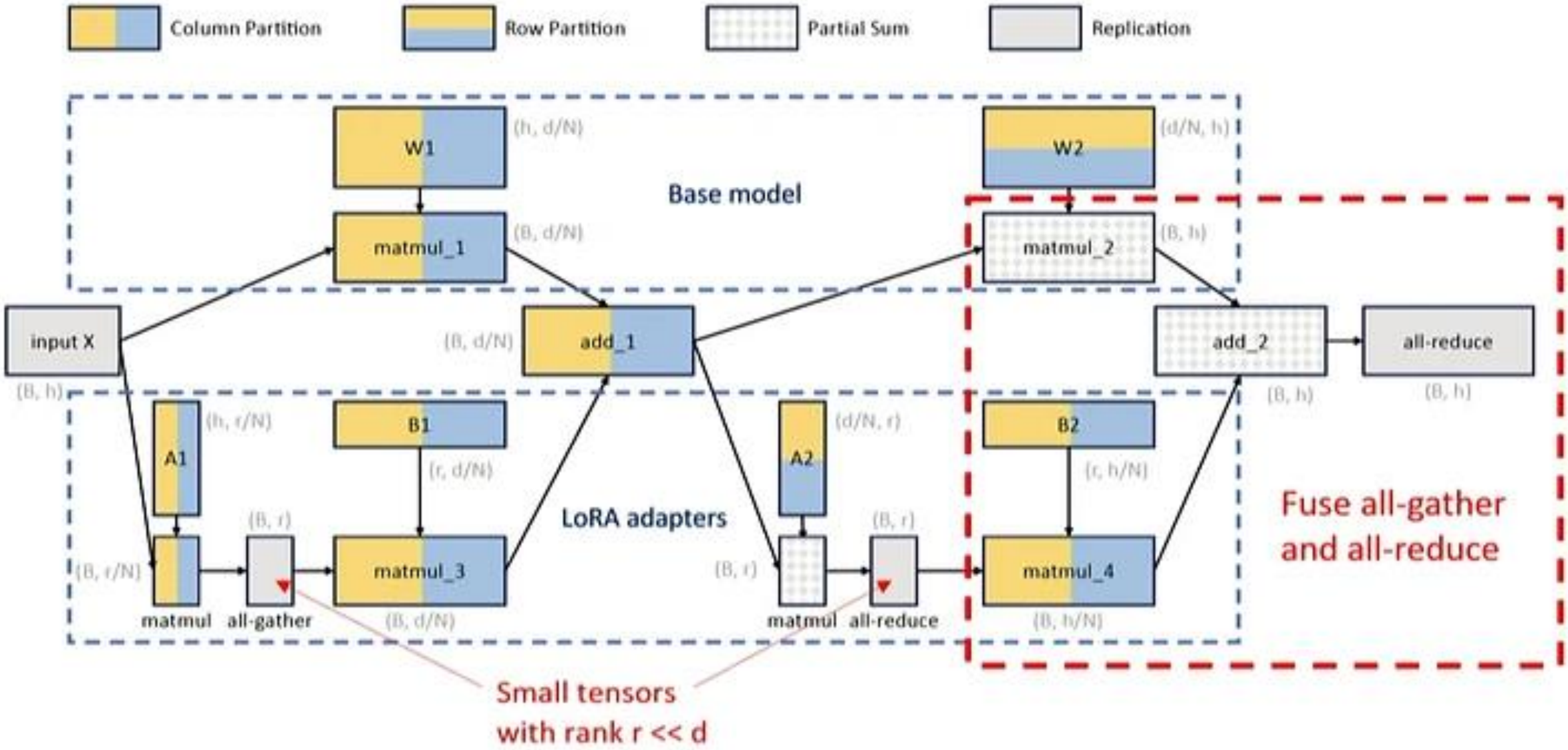
Tensor Parallelism

Base: 2(N-1)/N * Bh >> Adapter: 3(N-1)/N * Br + 2(N-1)/N * Br

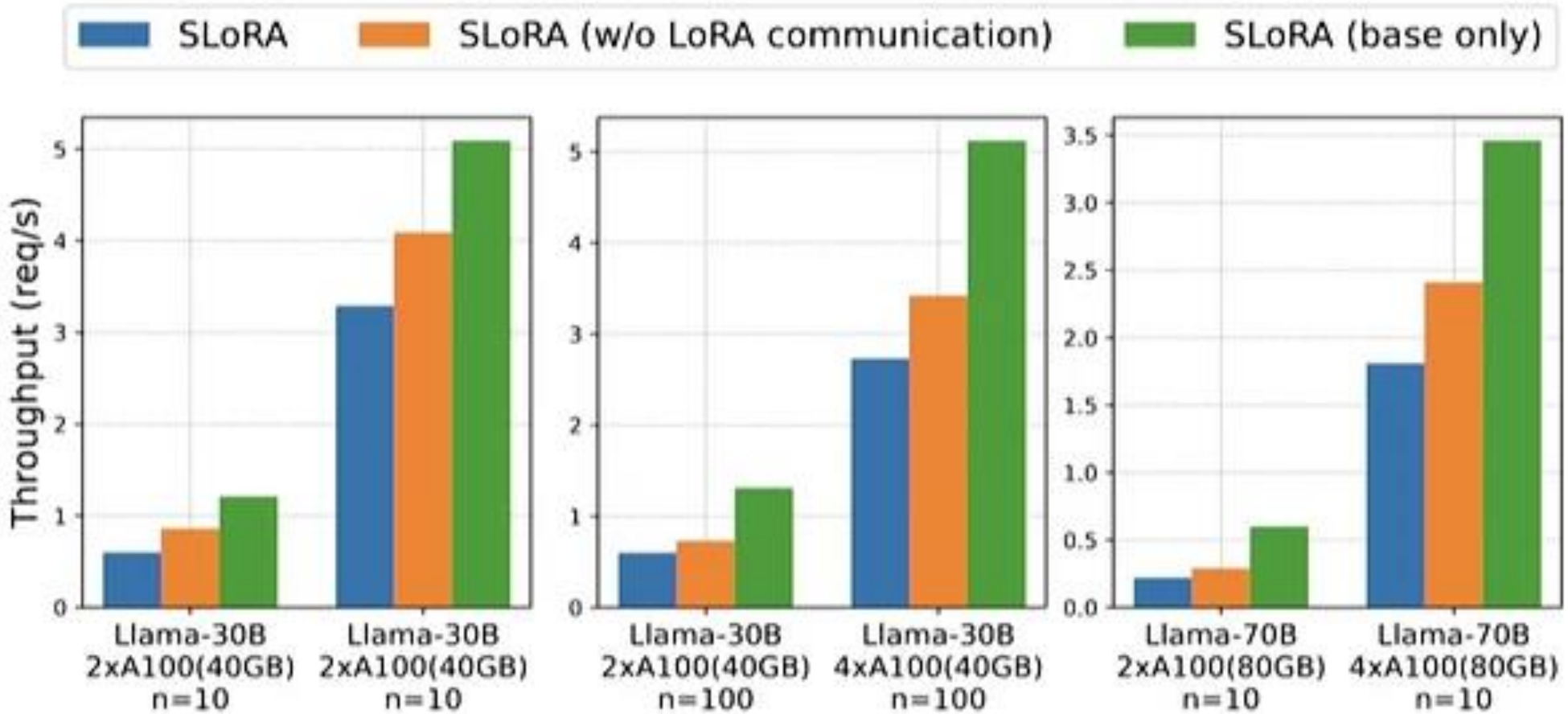


Tensor Parallelism

Base: 2(N-1)/N * Bh >> Adapter: 3(N-1)/N * Br + 2(N-1)/N * Br



Tensor Parallelism



Conclusion

- SLoRA: a system that can serve 1000x adapters with much higher throughput.
- Unified memory pool, heterogeneous batching, S-LoRA tensor parallelism.
- Batching different adapters improve the throughput by up to 4x, and increase the number of served adapters by several orders of magnitude.

vllm-project/vllm

#4068 [Feature]: Allow LoRA adapters to be specified as in-memor...

5 comments

jacobthebanana opened on April 14, 2024

vLLM vs sLoRA in serving LoRA adapters

Adapter Management

- vLLM: Merges LoRA weights into base model
- S-LoRA: Stores adapters in main memory, fetches as needed

Memory Efficiency

- vLLM: Limited to <5 adapters in GPU due to GPU memory constraints, but can swap adapters between GPU and CPU memory
- S-LoRA: Can serve thousands of adapters on a single GPU

Performance

- vLLM: Lower throughput, limited adapter capacity
- S-LoRA: Up to 4x higher throughput, stable with increasing adapters

Parallelism

- vLLM: Separate processes for each adapter
- S-LoRA: Novel tensor parallelism strategy (S-LoRA TP)

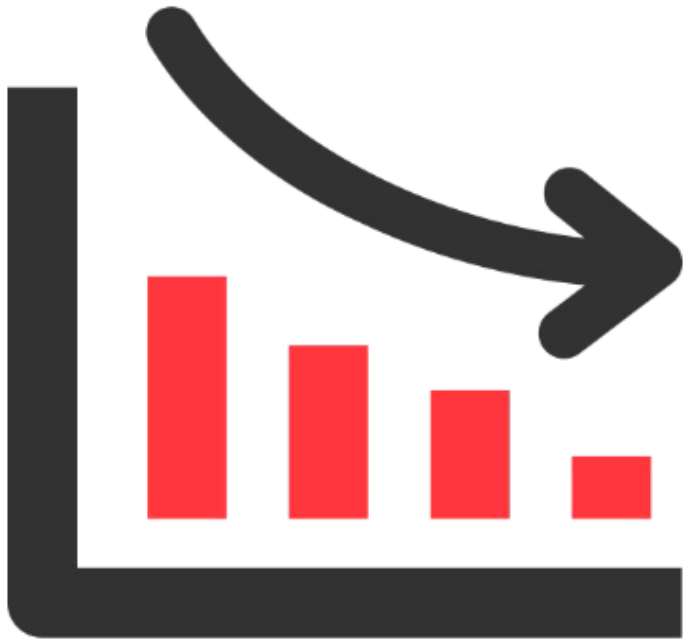
Optimization

- vLLM: No specific LoRA optimizations
- S-LoRA: Custom CUDA kernels for batched LoRA computation

Why Share GPUs?

Increase Utilization

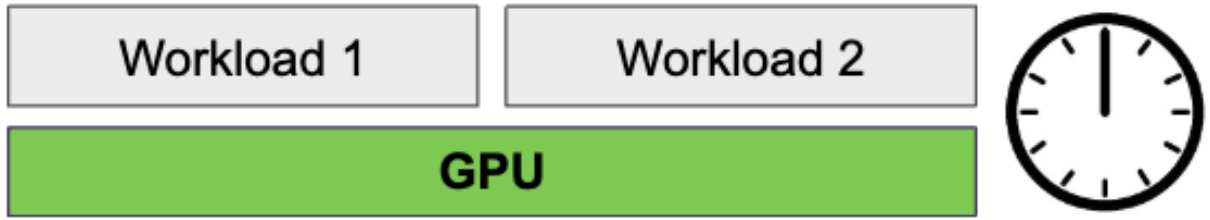
Decrease Costs



More Efficient Use of GPUs → Fewer GPUs to Get Work Done→ Lower Costs

Space vs. Time Partitioning

Space



Advantages:

- Fully resident at all times
- No context switch overhead
- Predictable performance

Disadvantages:

- Subset of GPU resources
- Limited number of clients

Space vs Time Partitioning

Advantages:

- Fully resident at all times
- No context switch overhead
- Predictable performance

Disadvantages:

- Subset of GPU resources
- Limited number of clients

Advantages:

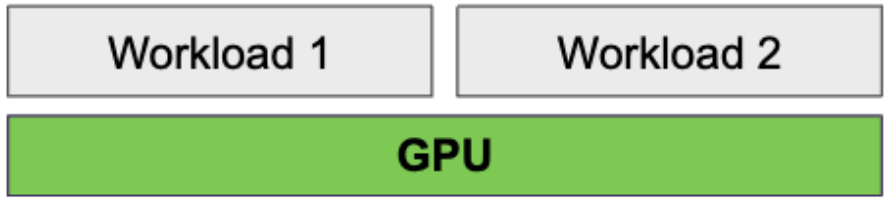
- Full set of GPU resources
- Unlimited number of clients

Disadvantages:

- Only resident a fraction of the time
- Additional context switch overhead
- Unpredictable performance

Space vs Time Partitioning

Space



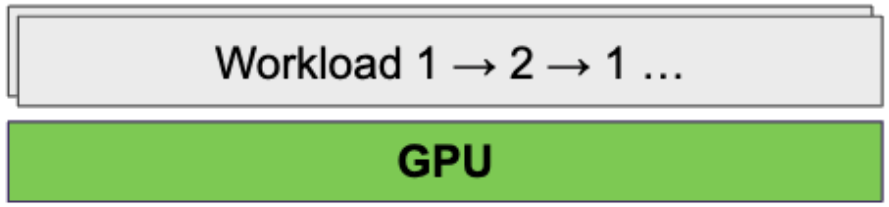
Advantages:

- Fully resident at all times
- No context switch overhead
- Predictable performance

Disadvantages:

- Subset of GPU resources
- Limited number of clients

Time



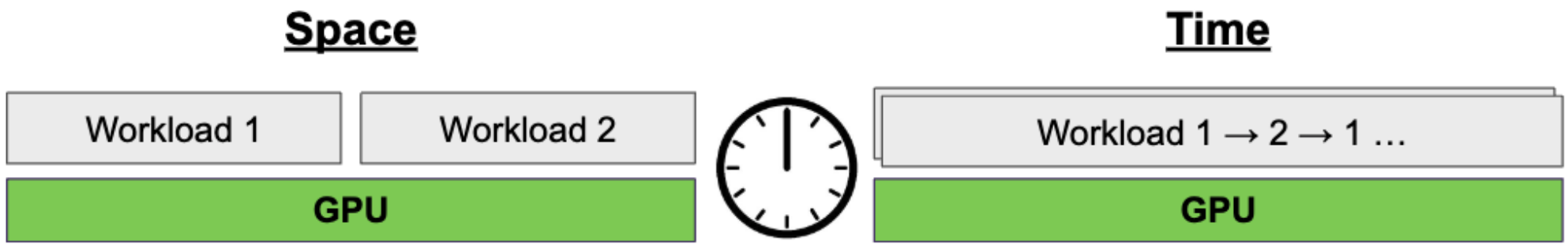
Advantages:

- Full set of GPU resources
- Unlimited number of clients

Disadvantages:

- Only resident a fraction of the time
- Additional context switch overhead
- Unpredictable performance

Space vs Time Partitioning



Advantages:

- Fully resident at all times
- No context switch overhead
- Predictable performance

Disadvantages:

- Subset of GPU resources
- Limited number of clients

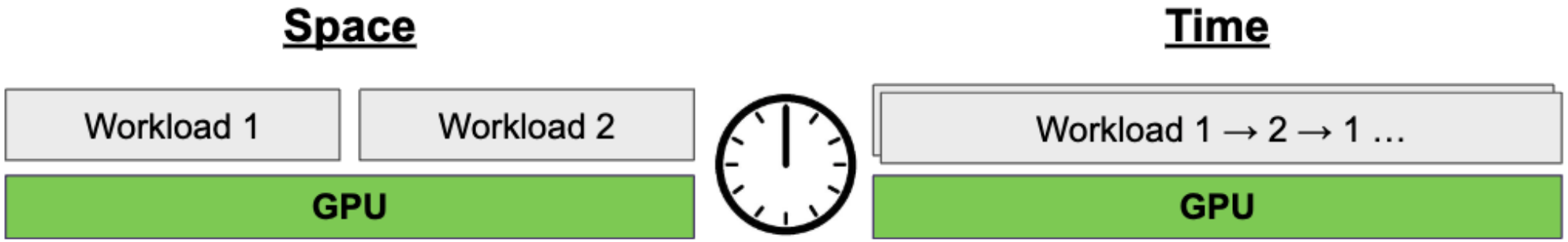
Advantages:

- Full set of GPU resources
- Unlimited number of clients

Disadvantages:

- Only resident a fraction of the time
- Additional context switch overhead
- Unpredictable performance

Space vs Time Partitioning



Advantages:

- Fully resident at all times
- No context switch overhead
- Predictable performance

Disadvantages:

- Subset of GPU resources
- Limited number of clients

Advantages:

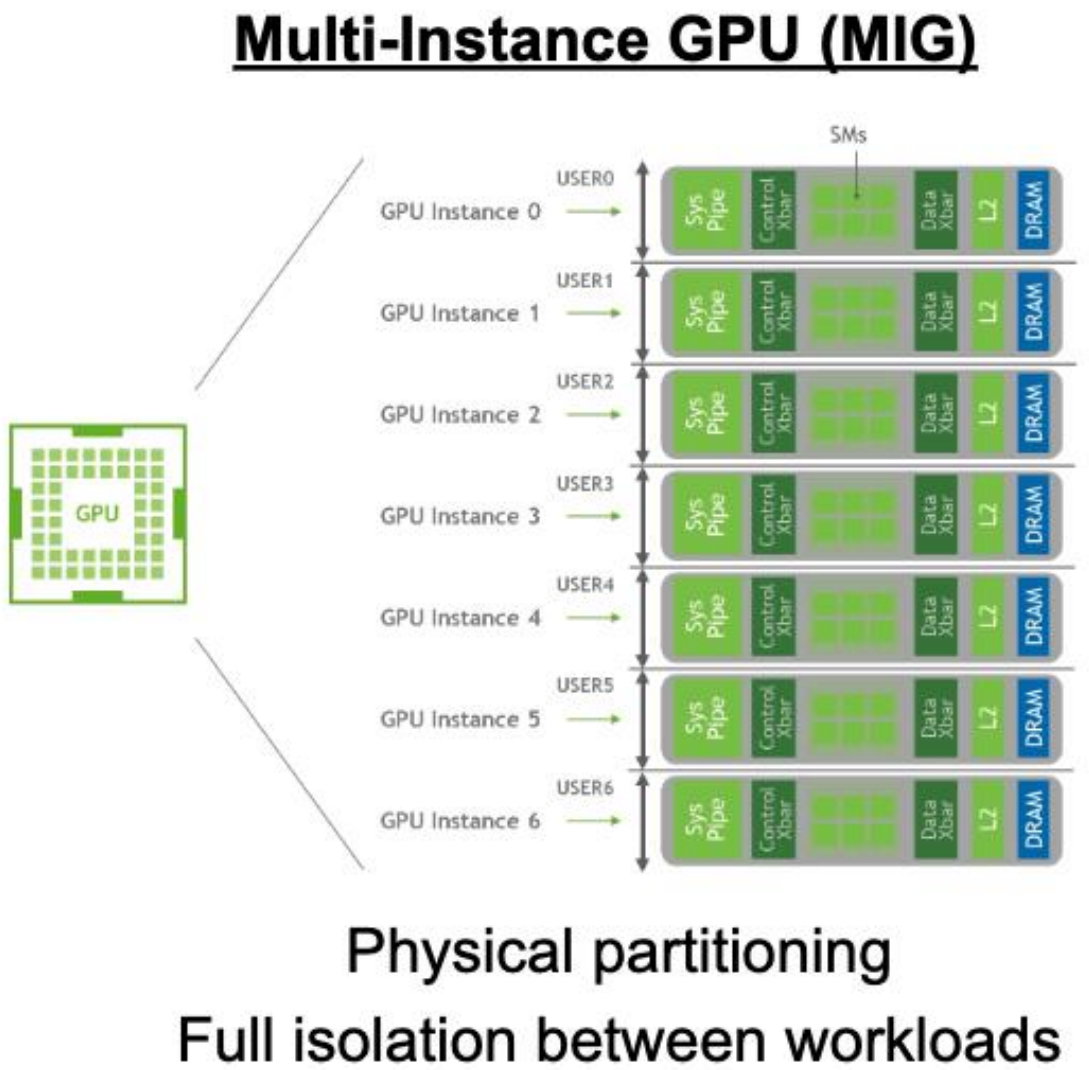
- Full set of GPU resources
- Unlimited number of clients

Disadvantages:

- Only resident a fraction of the time
- Additional context switch overhead
- Unpredictable performance

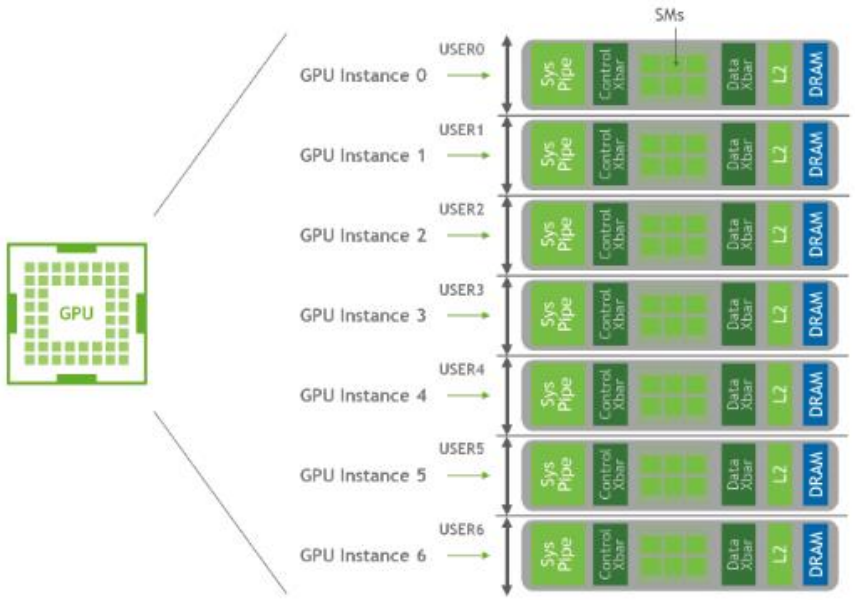
Inherent Tradeoff in Choosing One Strategy over the Other

Hardware vs. Software-Based Space Partitioning



Hardware vs. Software-Based Space Partitioning

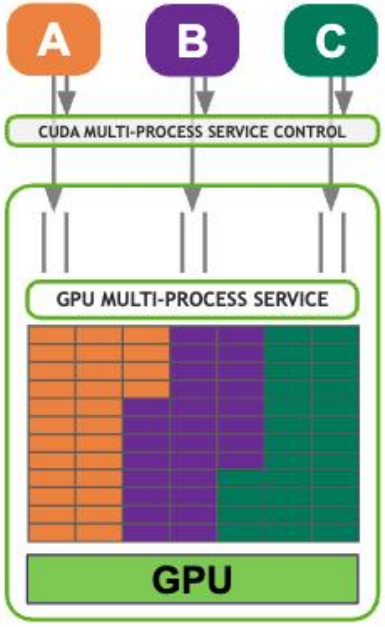
Multi-Instance GPU (MIG)



Physical partitioning

Full isolation between workloads

Multi-Process Service (MPS)



Logical partitioning

Limited isolation between workloads

Hardware vs. Software-Based Space Partitioning

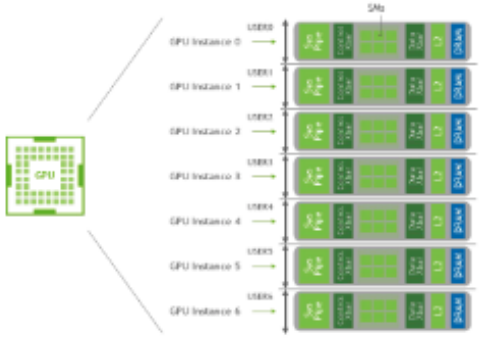
Multi-Instance GPU (MIG)

Advantages:

- Full Fault Isolation
- Guaranteed Memory Bandwidth QoS
- Suitable for Multi-Tenant Environments

Disadvantages:

- Fixed Size Partitions (multiple dimensions)



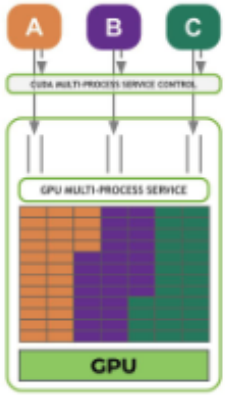
Multi-Process Service (MPS)

Advantages:

- Flexible Partition Size (multiple dimensions)

Disadvantages:

- Limited Fault Isolation
- No Memory Bandwidth QoS
- Not Suitable for Multi-Tenant Environments



Hardware vs. Software-Based Space Partitioning

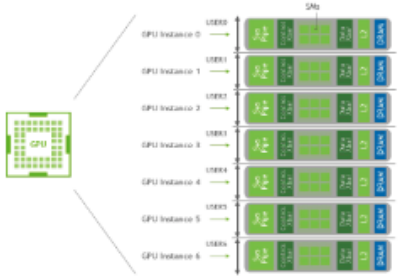
Multi-Instance GPU (MIG)

Advantages:

- Full Fault Isolation
- Guaranteed Memory Bandwidth QoS
- Suitable for Multi-Tenant Environments

Disadvantages:

- Fixed Size Partitions (multiple dimensions)



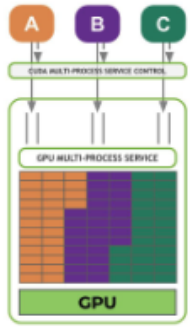
Multi-Process Service (MPS)

Advantages:

- Flexible Partition Size (multiple dimensions)

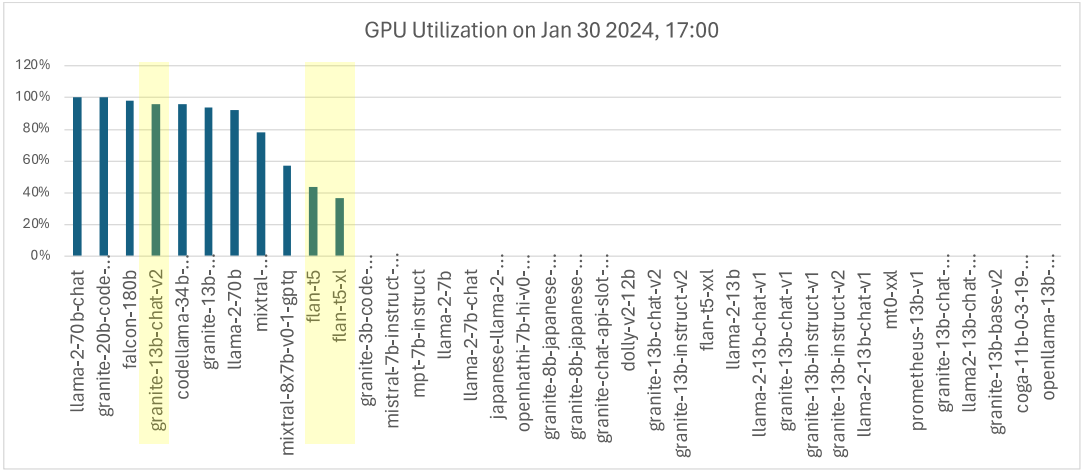
Disadvantages:

- Limited Fault Isolation
- No Memory Bandwidth QoS
- Not Suitable for Multi-Tenant Environments

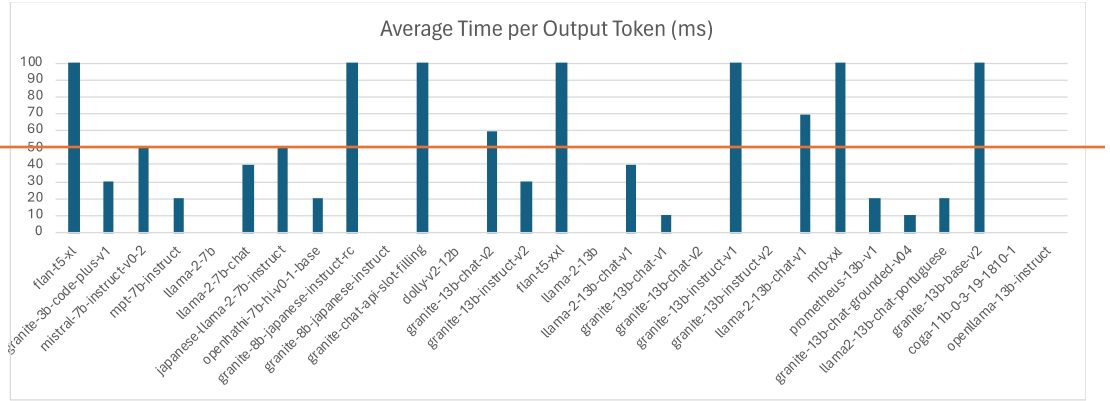


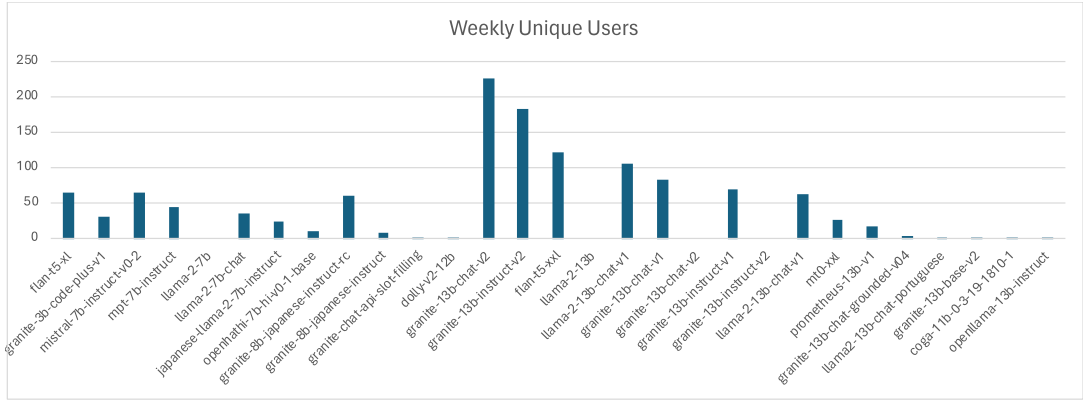
Motivation

- GPUs are idling for small models.
- Small Models (<= 13B)
  - 29 models out of 54 in Research clusters
- Usage & Performance
  - Active users per week < 60
  - Per request latency < 10 seconds
  - Per token latency < 50 ms



SLA = 50 ms





GPU Sharing Options

Unpredictable memory allocation may lead to exceptions when models are dynamically sharing the HBM Space.

	Time Sharing	MPS	MIG
Process Isolation	Good	Correlated Failures	Good
HBM Space	Dynamic Sharing	Dynamic Sharing	Static Partitioning
Compute Unites	Time Multiplexing	Space Multiplexing	Space Multiplexing
Mechanism	Software	Hybrid	Hardware

Paged and flash attention techniques allocate variable memory for requests, making memory usage unpredictable for sharing.

Limiting batch sizes to cap memory restricts the advantages of time-sharing compared to MIG.

Overview of MIG

- Multi-instance GPU (MIG)
  - Allow GPUs be securely partitioned up to 7 separate GPU instances
  - Separate and isolated paths through the entire memory system
  - Supported bare-metal, containers and Kubernetes
- MIG profile placement
  - Multiple GPU instances can be created from a mix and match of these profiles
  - A100 80GB: 7g80gb, 4g40gb, 3g40gb, 2g20gb, 1g20gb, 1g10gb, 1g10gb+me

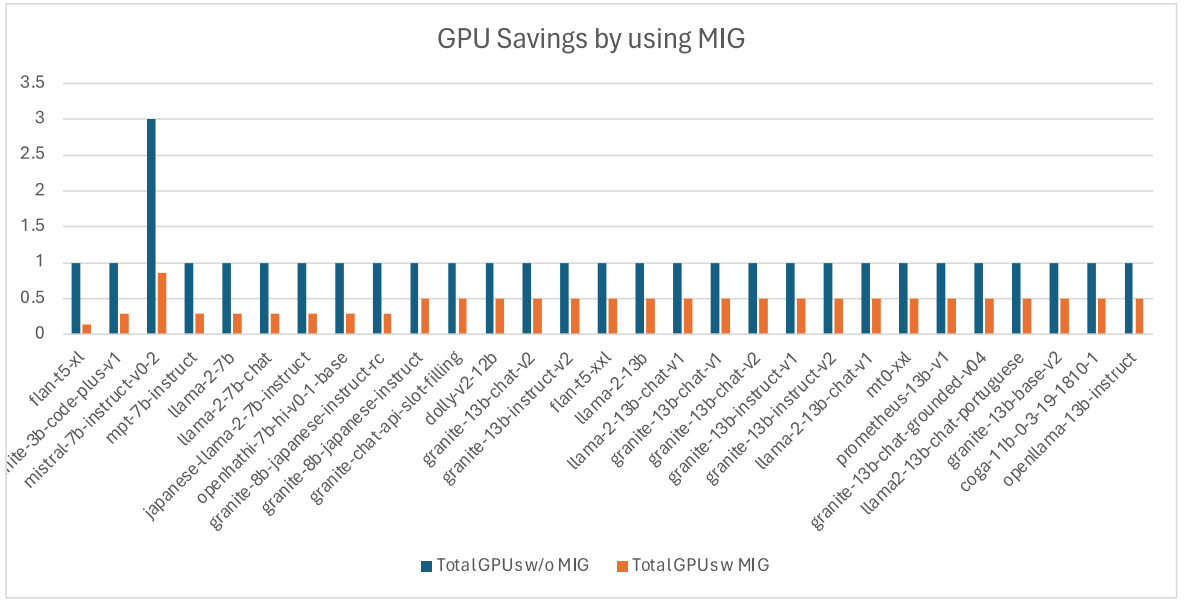
Config	GPC Slice #0	GPC Slice #1	GPC Slice #2	GPC Slice #3	GPC Slice #4	GPC Slice #5	GPC Slice #6
1	7						
2	4				3		
3	4				2		1
4	4				1	1	1
5	3			3			
6	3			2		1	
7	3			1	1	1	
8	2		2		3		
9	2		1	1	3		
10	1	1	2		3		
11	1	1	1	1	3		
12	2		2		2		1
13	2		1	1	2		1
14	1	1	2		2		1
15	2		1	1	1	1	1
16	1	1	2		1	1	1
17	1	1	1	1	2		1
18	1	1	1	1	1	2	
19	1	1	1	1	1	1	1

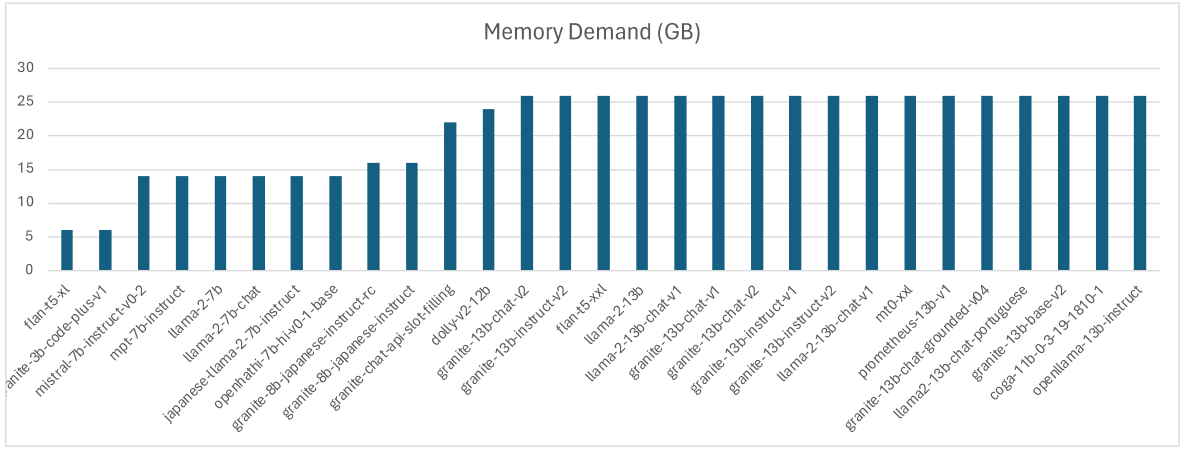
MIG Profiles on A100 [1]

[1] https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html

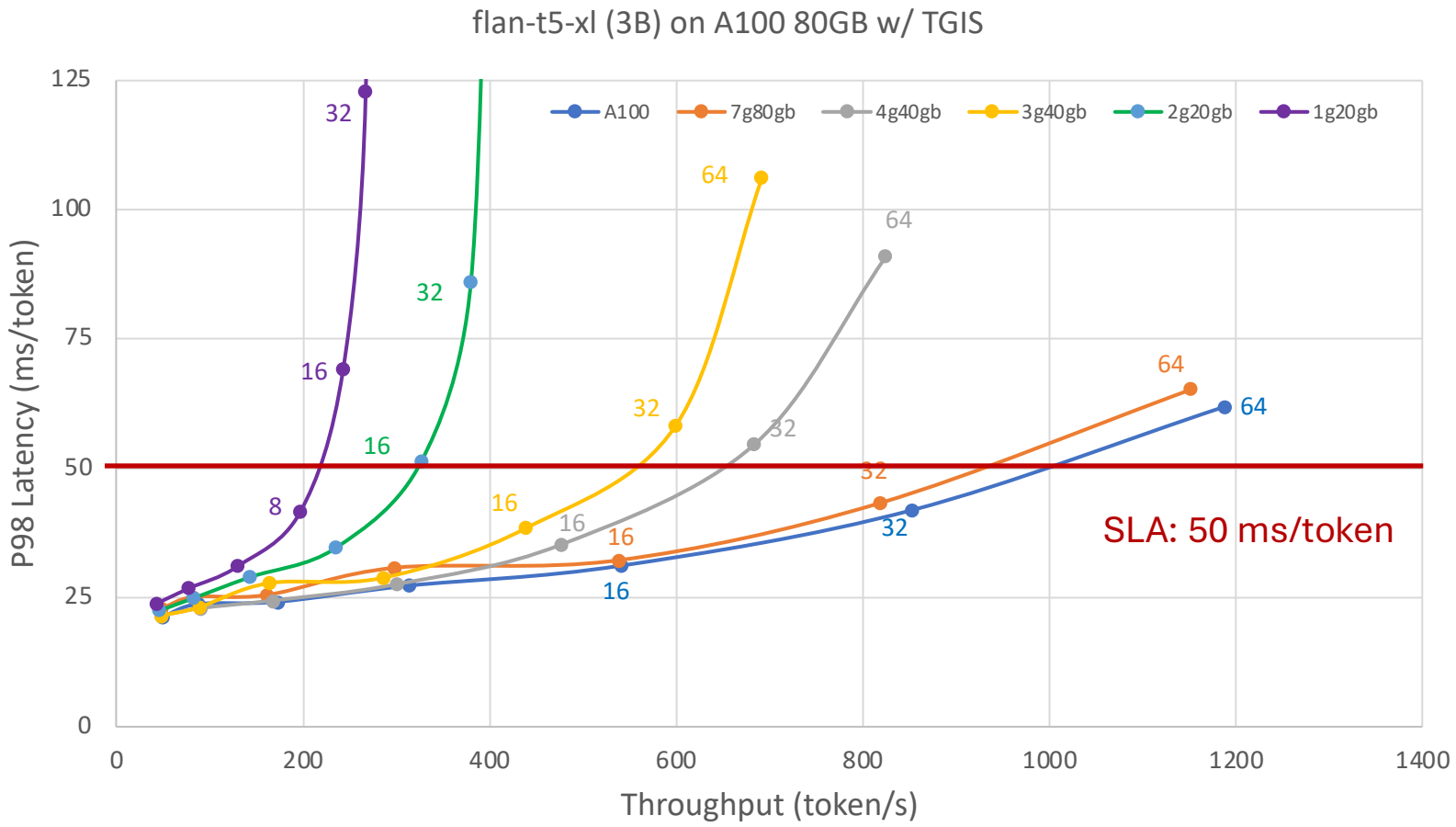
Potential Savings

- Candidate MIG sizes
  - 1g10gb, 2g20gb, 3g40gb, 4g40gb
- Potential Savings by using MIG
  - Without MIG: 31 A100 GPUs
  - With MIG: 13 A100 GPUs





Performance Analysis --- flan-t5-xl (3B)



What MIG size should we choose?

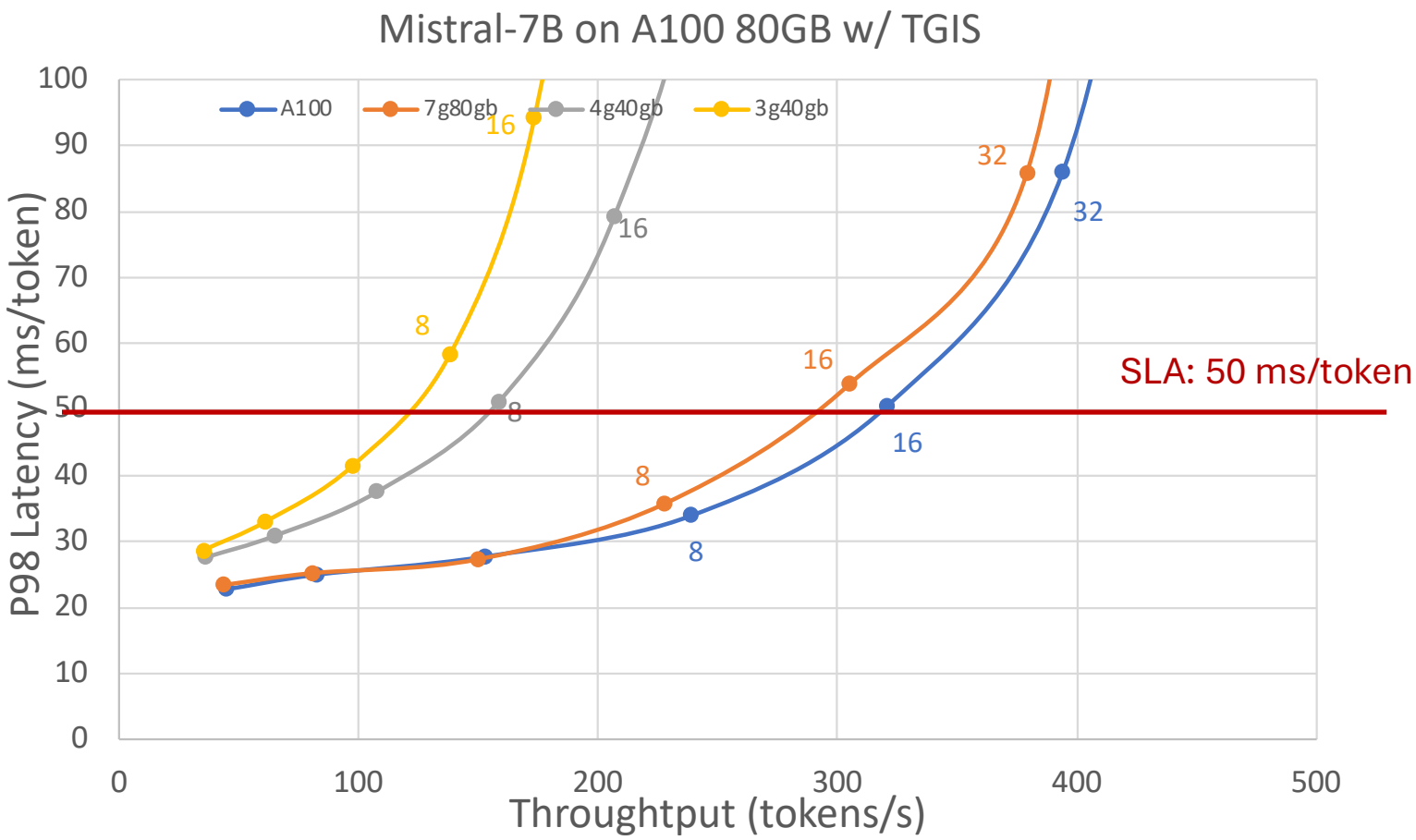
1g20gb can be chosen if the concurrent number of users is below **8**.

2g20gb can be chosen if the concurrent number of users is below **16**.

As the MIG size becomes *smaller*, the latencies *increase*, and the throughput *decreases*.

<u>It is worth switching to MIG when the load on the model is light.</u>

Performance Analysis --- Mistral (7B)



What MIG size should we choose?

4g40gb can be chosen if the concurrent number of users is below **8**.

As the MIG size becomes *smaller*, the latencies *increase*, and the throughput *decreases*.

It is worth switching to MIG when the load on the model is light.

How to enable MIG and configure slices on OpenShift cluster

When we apply any mig configuration, need all gpus on the target node to be free

How to enable MIG (**ref**)

- Check if the MIG mode is enabled on the node: oc rsh –n nvidia-gpu-operator nvidia-driver-daemonset-xxxxxxxx nvidia-smi
- If not, oc rsh –n nvidia-gpu-operator nvidia-driver-daemonset-xxxxxxxx nvidia-smi –mig 1
- Then reboot the node

How to configure custom MIG profiles

- oc create cm –n nvidia-gpu-operator configmap mig-custom-config –from-file custom-mig-config.yaml
  (if the configmap already exists, edit it)
- oc edit clusterpolicy and change spec.migManager.config.name to your
  "mig":{"strategy":"mixed"},
  "migManager":{"enabled":true, "config":{"name": "mig-custom-config"}}
- Note: If we want to mix mig-disabled gpus and mig-enabled gpus, need to start with disabled gpus in the mig config yaml file
- MIG_CONFIGURATION=custom1 && oc label node/${NODE_NAME} nvidia.com/mig.config=$MIG_CONFIGURATION –overwrite

Three ways to confirm if the configuration is successfully applied

- oc get nodes/$NODE_NAME --show-labels | tr ',' '\n' | grep nvidia.com
- oc get nodes/$NODE_NAME -ojsonpath={.status.allocatable} | jq -c 'with_entries(select (.key | startswith("nvidia.com")))' | jq
- oc logs nvidia-mig-manager-xxxx -n nvidia-gpu-operator

To collect DCGM metrics, verify if the installed dcgm-exporter supports MIG, and upgrade it if not

- The dcgm-exporter 2.4.0-rc.2 or later supports MIG
- The installed dcgm-exporter version by gpu-operator is possible to be older than that

Numbers 💯 every LLM Developer 🛠️ should know *

🪄 Prompts

40-90%	Amount saved by appending "Be Concise" to your prompt
1.3	Average tokens per word

💡 Training and Fine Tuning

~$1 million	Cost to train a 13 billion parameter model on 1.4 trillion tokens
<0.001	Cost ratio of fine tuning vs training from scratch

💰 Price

~50	Cost Ratio of GPT-4 to GPT-3.5 Turbo
5	Cost Ratio of generation of text using GPT-3.5-Turbo vs OpenAI embedding
10	Cost Ratio of OpenAI embedding to Self-Hosted embedding
6	Cost Ratio of OpenAI base vs fine tuned model queries
1	Cost Ratio of Self-Hosted base vs fine-tuned model queries

🧠 GPU Memory

16GB 24GB 40/80GB	V100 GRAM capacity A10G GRAM capacity A100 GRAM capacity
2x number of parameters	Typical GPU memory requirements of an LLM for serving
~1GB	Typical GPU memory requirements of an embedding model
>10x	Throughput improvement from batching LLM requests
1 MB	GPU Memory required for 1 token of output with a 13B parameter model

* Check out bit.ly/llm-dev-numbers for how we calculated the numbers