vAttention: Dynamic Memory Management for Serving LLMs

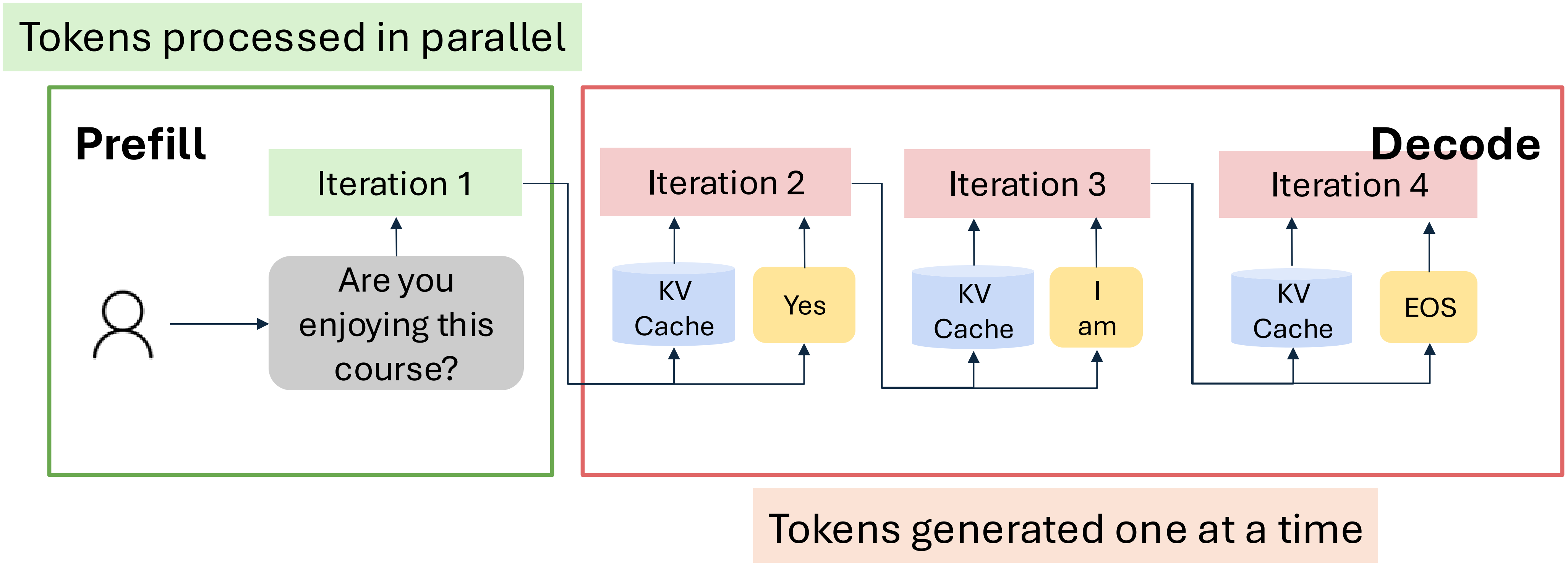
Presenters: Rahul Bothra, Chengyi Wang
11th October 2024

Our **comments** in purple boxes

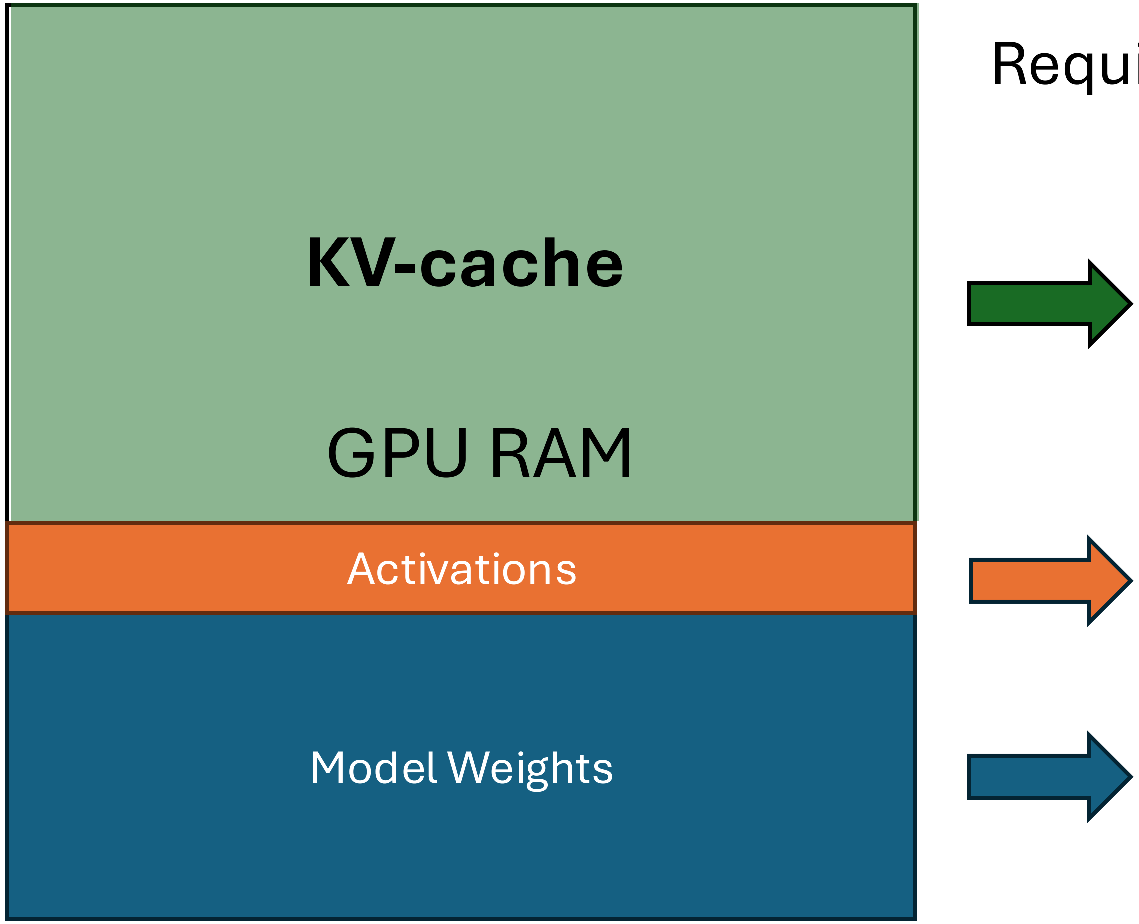
Contents (Remove names later)

1. (Rahul) LLM Inference primer + KV Cache
2. (Chengyi)Prior work on memory management (Orca and Paged Attention)
3. (Rahul) vAttention Design (Design and Challenges (CUDA paging size))
4. (Chengyi) vAttention Evaluation
5. (Chengyi + Rahul) Analysis and Future work

LLM Inference Primer



LLM Inference memory footprint



Requires careful memory management

- ❑ Large
- ❑ Dynamic
- ❑ Size unknown

Dynamic (small)

Static

KV Cache Memory Management

- KV-cache is **large, dynamic** and **size** is **unknown/variable**
- GPT-3: 1000 tokens = 4.5GB memory
- Grows one-token at a time (autoregressive decoding)
- Don't know request lengths in advance

Why care about memory management?

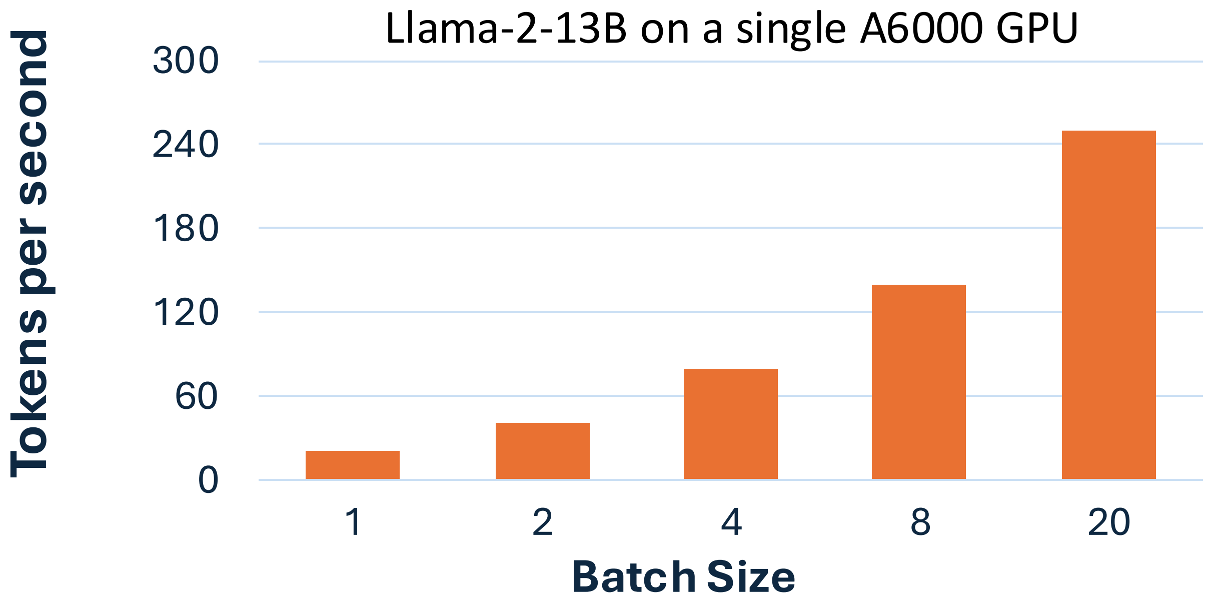
Would have been interesting to see ways of predicting the final KV cache size before it's full

KV-cache memory management

- KV-cache is **large, dynamic** and **size** is **unknown/variable**
- GPT-3: 1000 tokens = 4.5GB memory
- Grows one-token at a time (autoregressive decoding)
- Don’t know request lengths in advance

Why care about memory management?

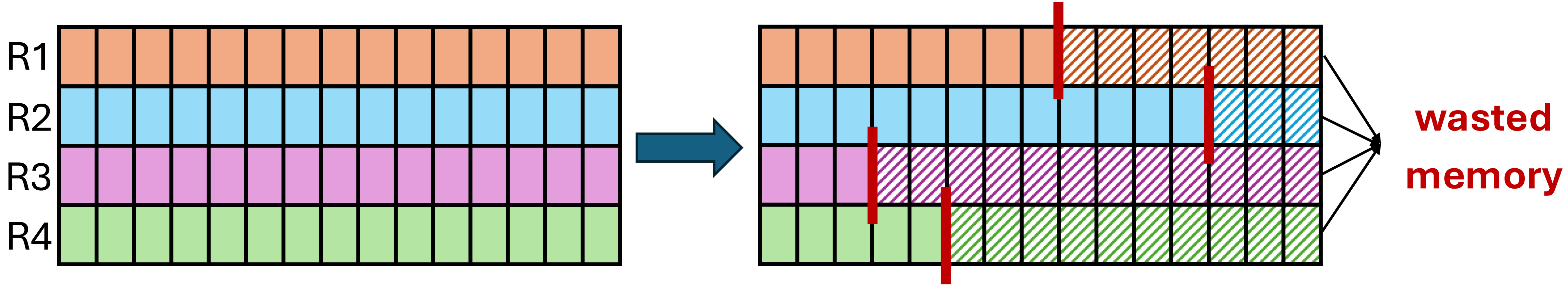
- Throughput depends on batch size



- Batch size depends on memory (KV-cache) allocator

A simple KV-cache memory manager

- **Assume** length(R_i)== max context length
- Allocate all memory upfront
  - e.g., max model length for GPT-3 = 4K
  - allocate 18GB memory for each request (= 4*4.5GB)



Can't serve more requests (though memory is underutilized)

A better approach: vLLM [SOSP’23]

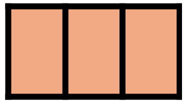
- Dynamic fine-grained memory allocation for KV-cache

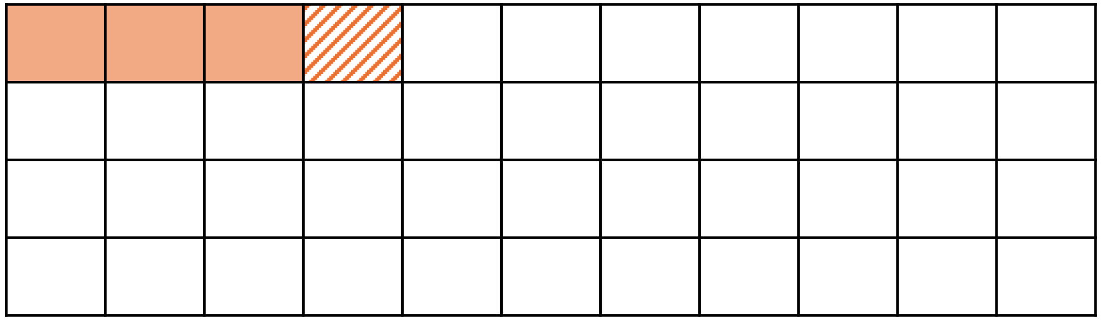
- **Dynamic:**
  - On demand allocation

- **Fine-grained:**
  - Divide memory into fixed-size blocks (e.g., 16 tokens)
  - Allocate one block at a time

Memory allocation with vLLM

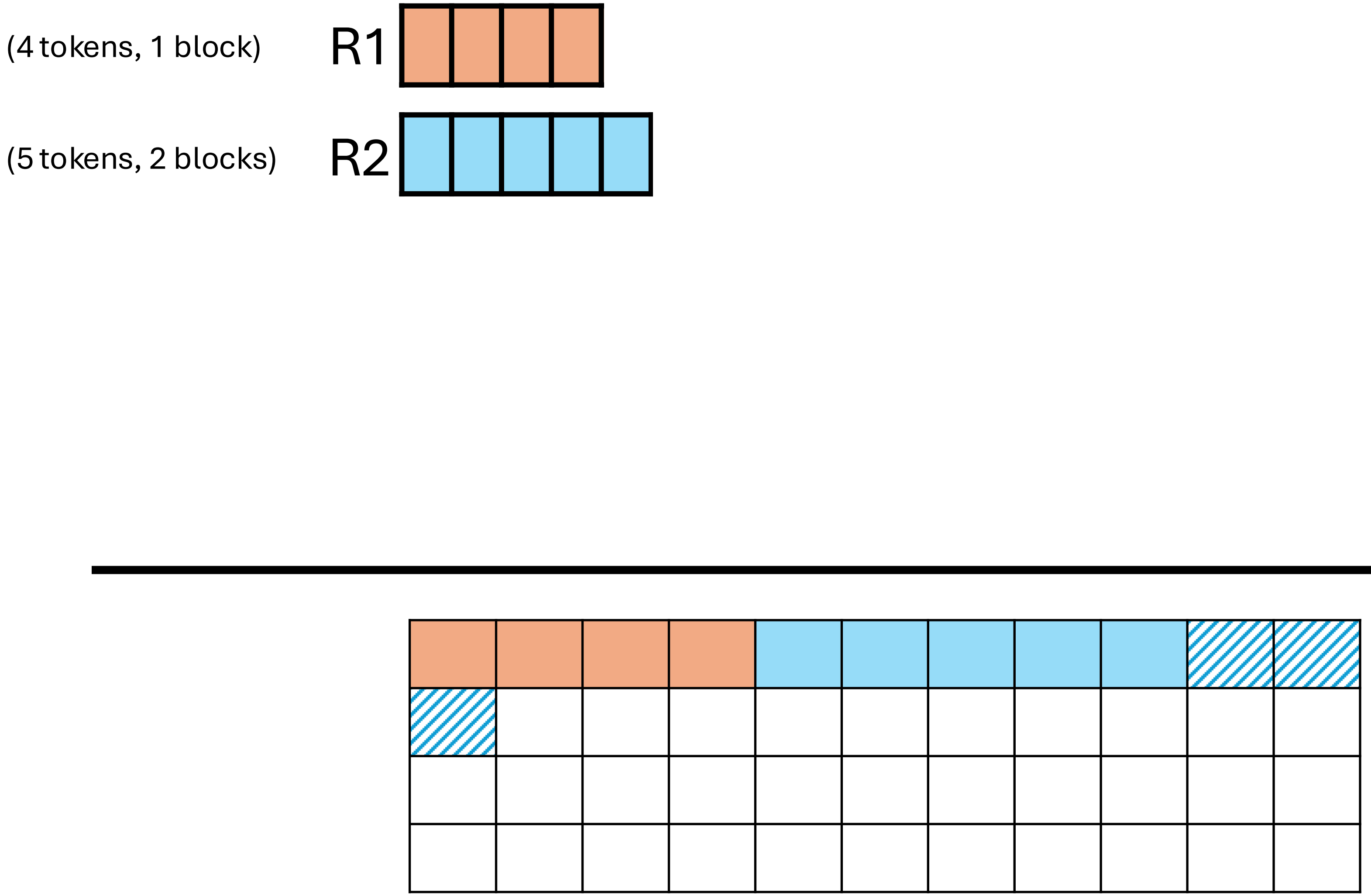
(3 tokens, 1 block) R1





GPU Memory (block size = 4)

Memory allocation with vLLM



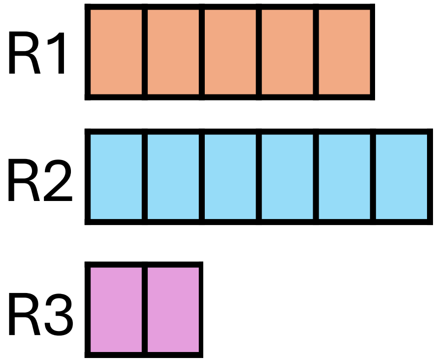
GPU Memory (block size = 4)

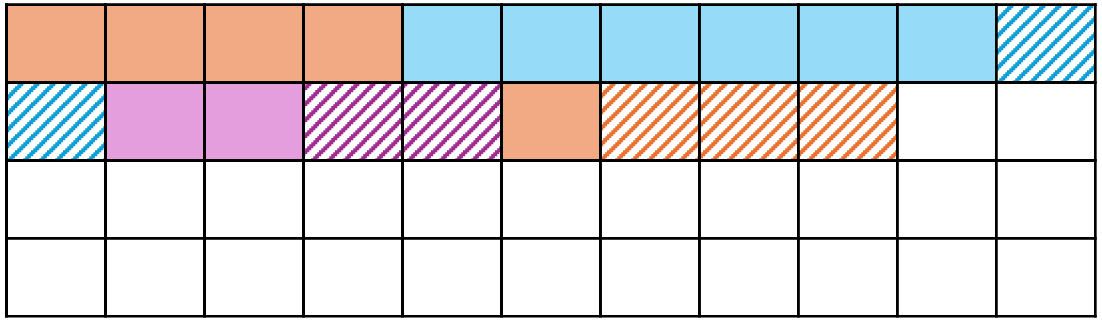
Memory allocation with vLLM

(5 tokens, 2 blocks)

(6 tokens, 2 blocks)

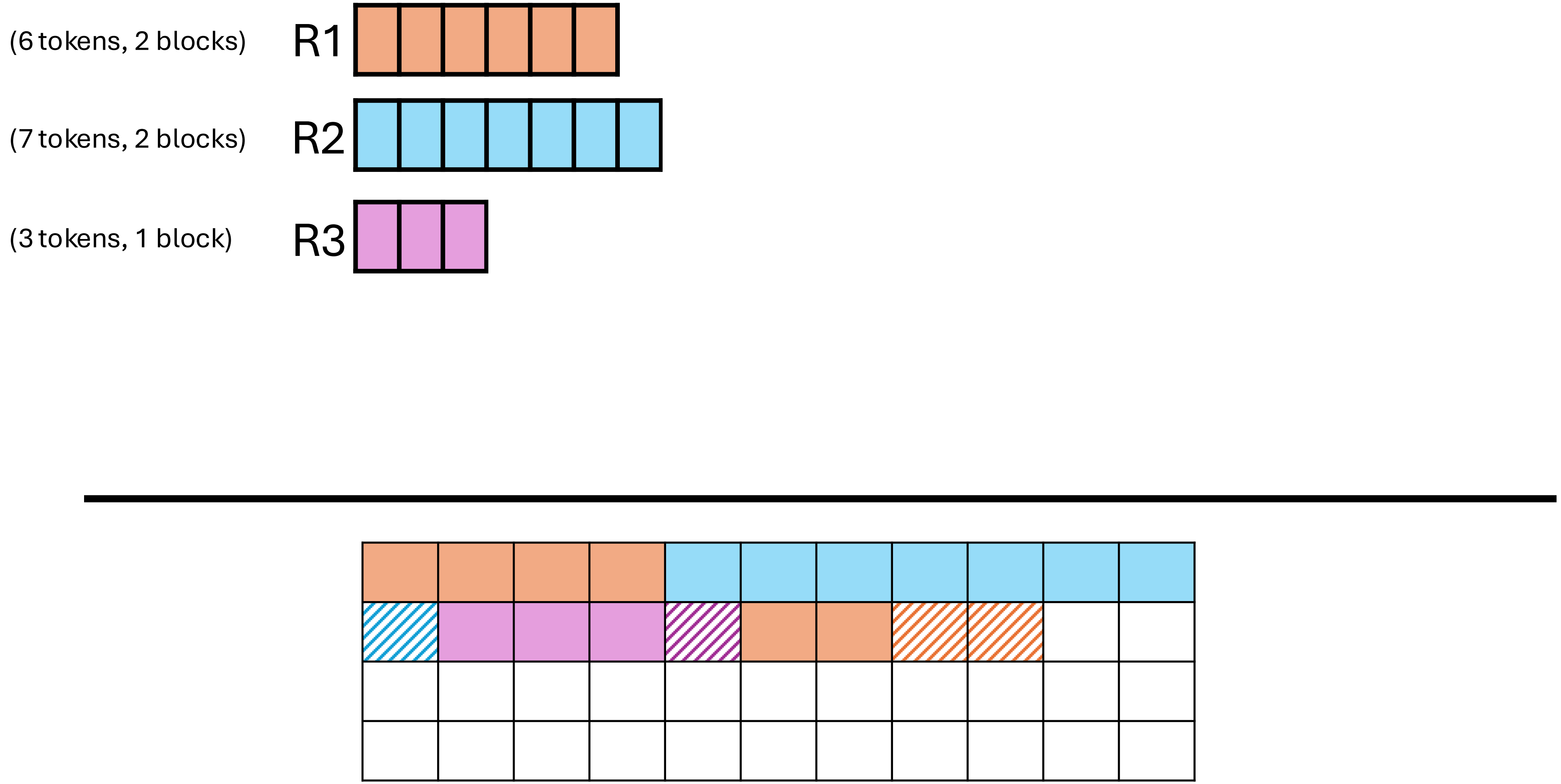
(2 tokens, 1 block)





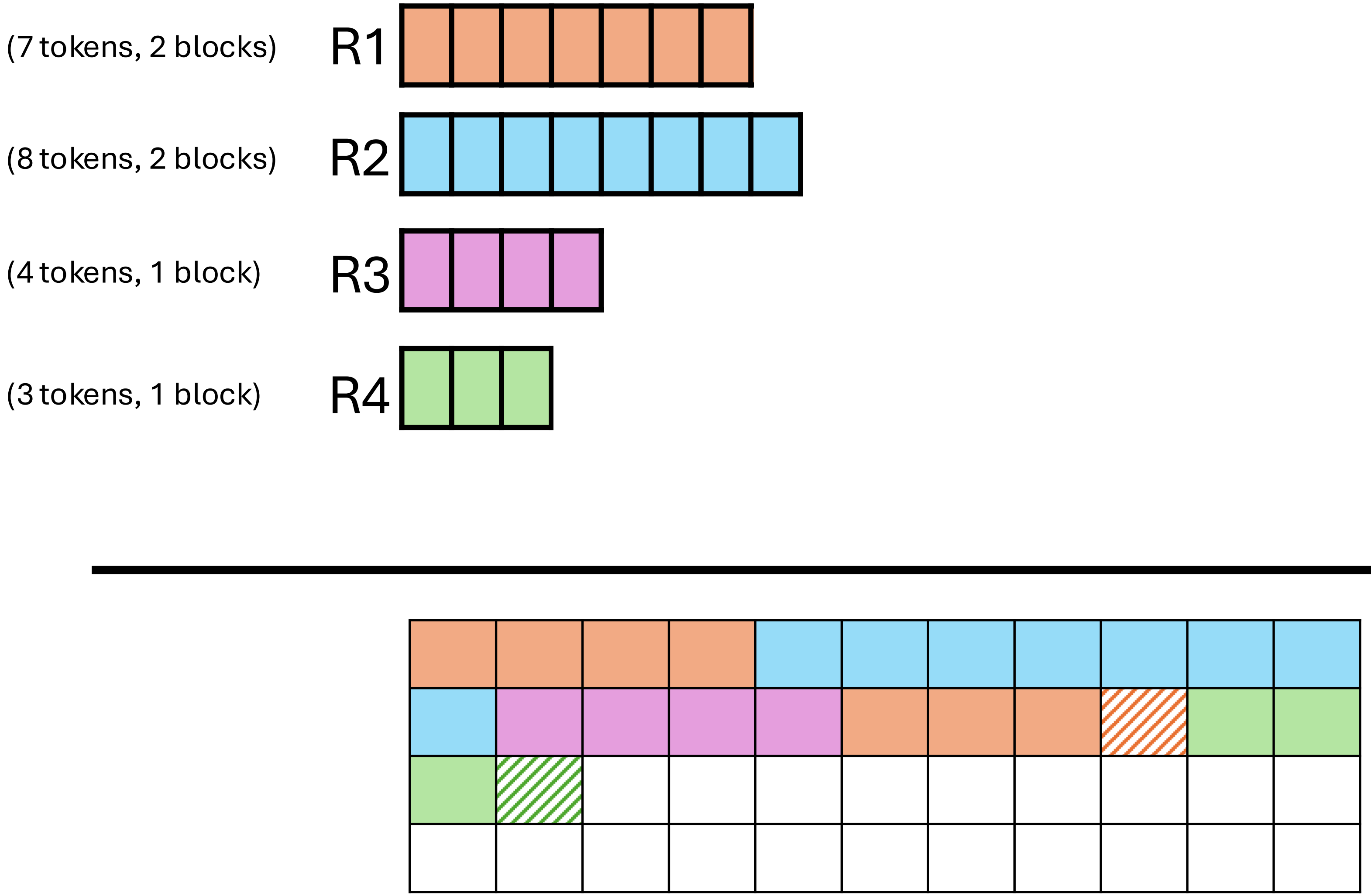
GPU Memory (block size = 4)

Memory allocation with vLLM



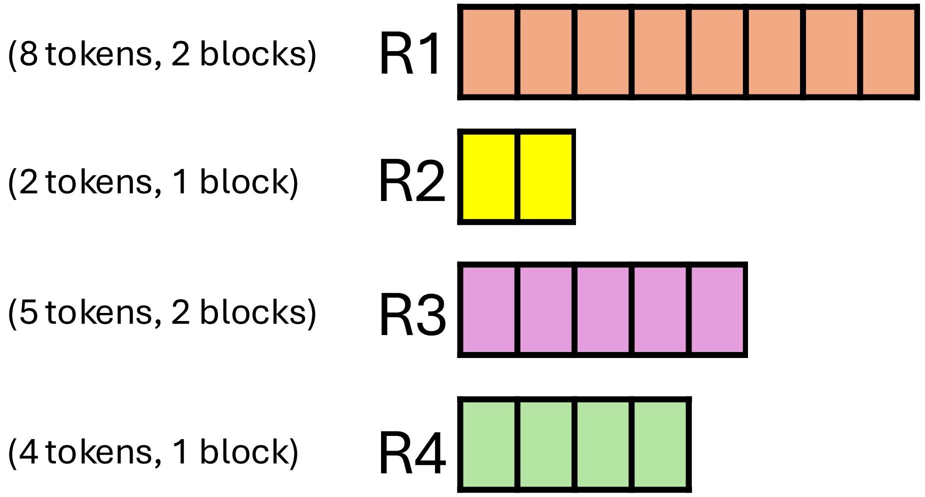
GPU Memory (block size = 4)

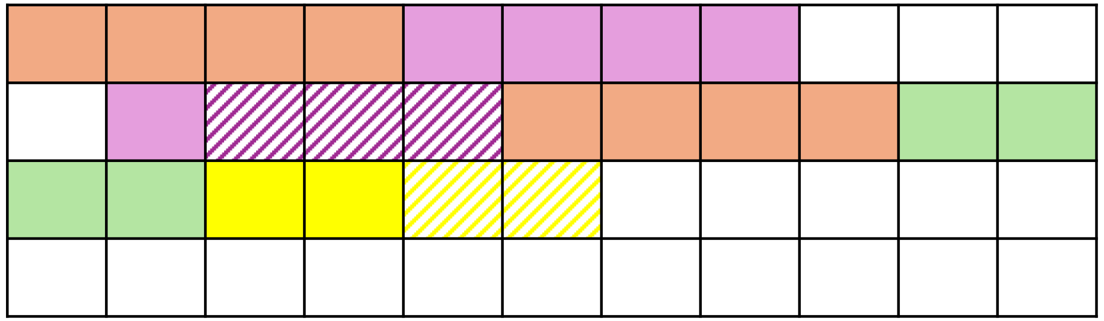
Memory allocation with vLLM



GPU Memory (block size = 4)

Memory allocation with vLLM





GPU Memory (block size = 4)

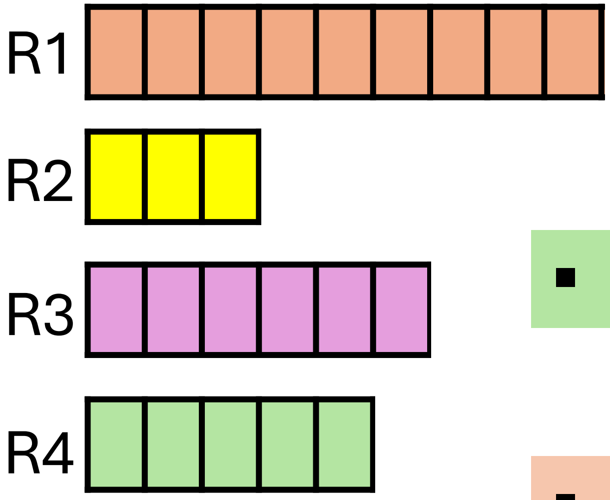
Memory allocation with vLLM

(9 tokens, 3 blocks)

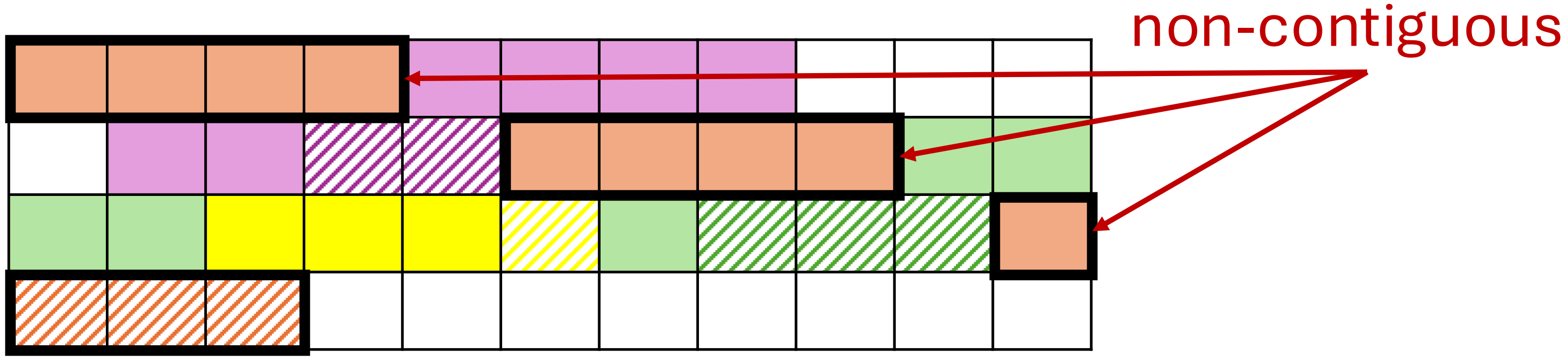
(3 tokens, 1 block)

(6 tokens, 2 blocks)

(5 tokens, 1 block)



- Dynamic allocation eliminates fragmentation
- Makes KV-cache **non-contiguous**



GPU Memory (block size = 4)

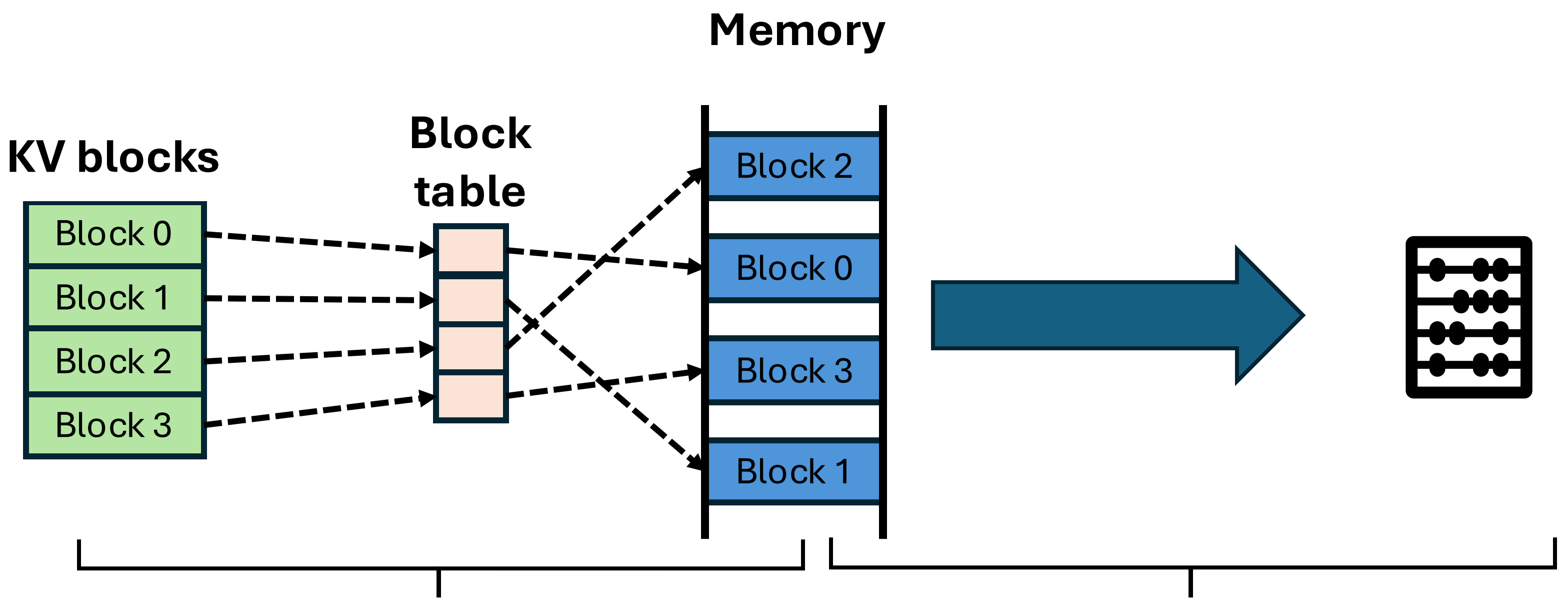
vLLM and PagedAttention

$$Attention(q_i, K, V) = softmax(\frac{q_i K^T}{scale})V$$

- Conventional implementations expect **contiguous K and V**
  - No longer possible in vLLM

- **PagedAttention**
  - Compute attention over non-contiguous blocks of K and V

Programming overhead



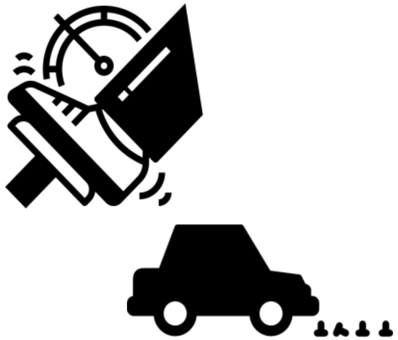
Serving stack needs to implement a memory manager

Attention kernel needs to fetch K and V from non-contiguous memory blocks

Issues with PagedAttention

Programming

Writing performant GPU code is non-trivial

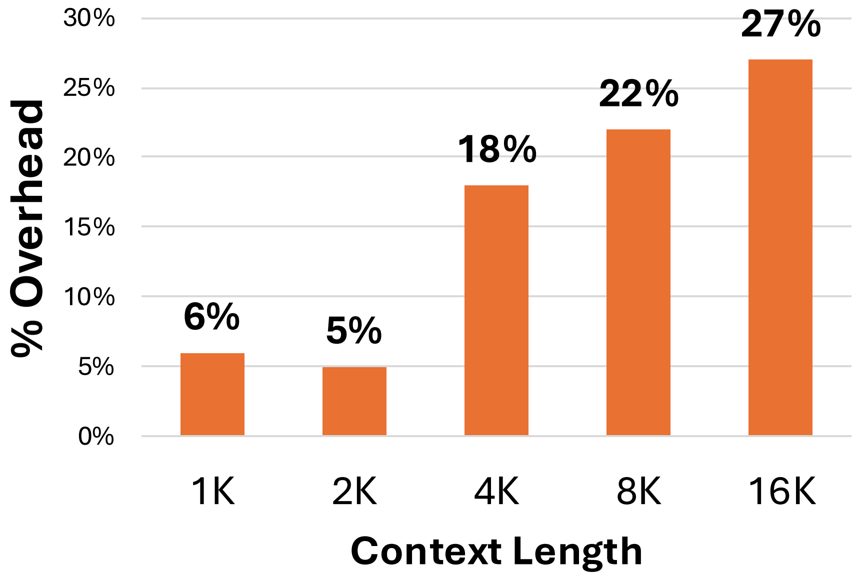


Performance

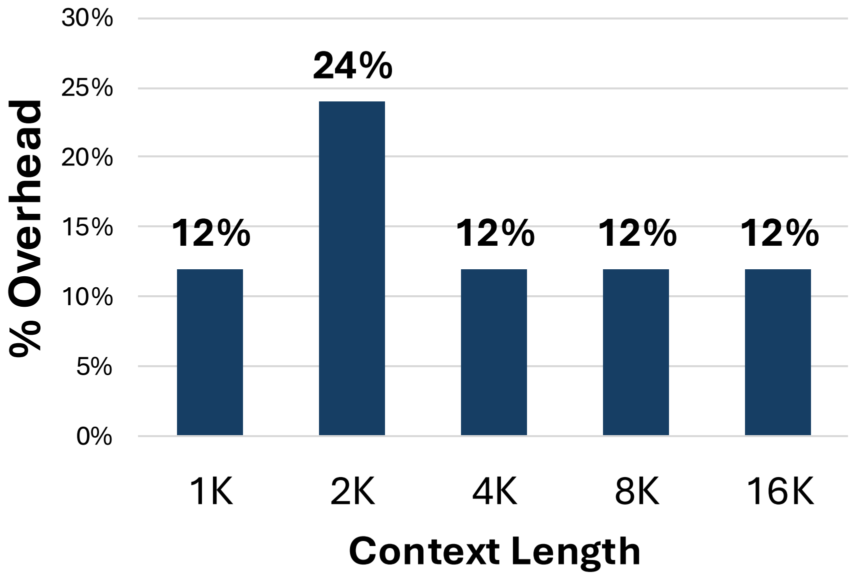
Redundant address translation has a cost

Performance overhead

FlashAttention-2*



FlashInfer**



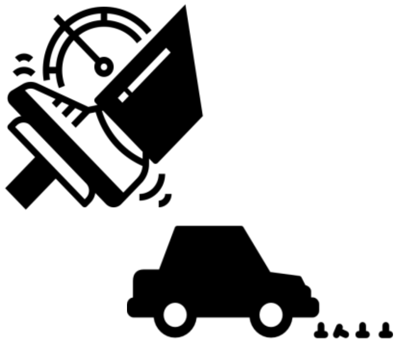
[* Dao-AILab/flash-attention: Fast and memory-efficient exact attention (github.com)]()

** [flashinfer-ai/flashinfer: FlashInfer: Kernel Library for LLM Serving (github.com)]()

Issues with PagedAttention

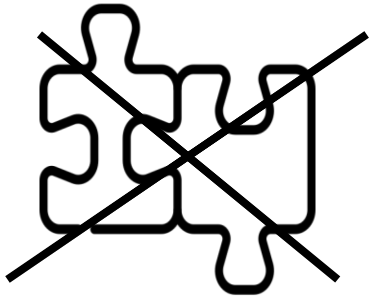
Programming

Writing performant GPU code is non-trivial



Performance

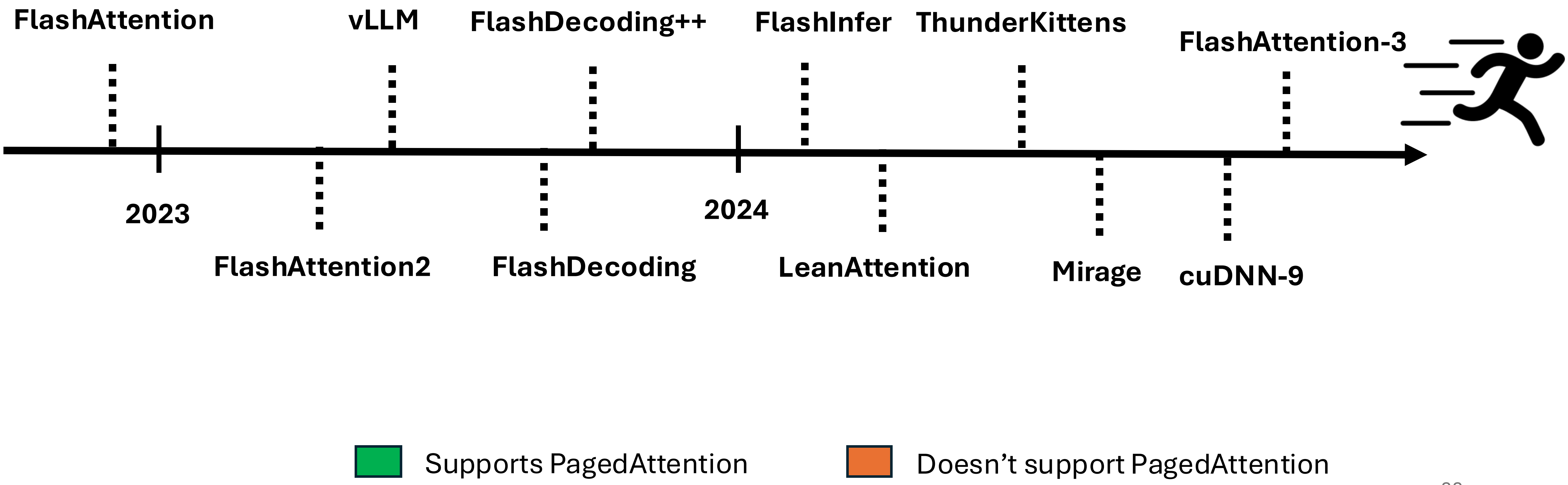
Redundant address translation has a cost



Portability

Kernels are not compatible!

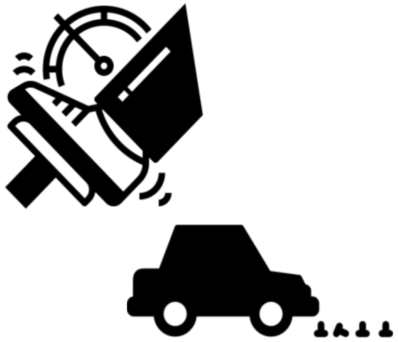
Why care about Portability?



Issues with PagedAttention

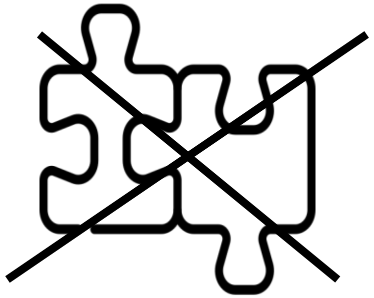
Programming

Writing performant GPU code is non-trivial



Performance

Redundant address translation has a cost



Portability

Kernels are not compatible (different formats)

Can we do better?

- Non-contiguous layout is not ideal

- Ideal solution:
  - **Dynamic** memory allocation
  - **Contiguous** memory layout

These goals are usually conflicting

Can we resolve the conflict?

Issues with PagedAttention

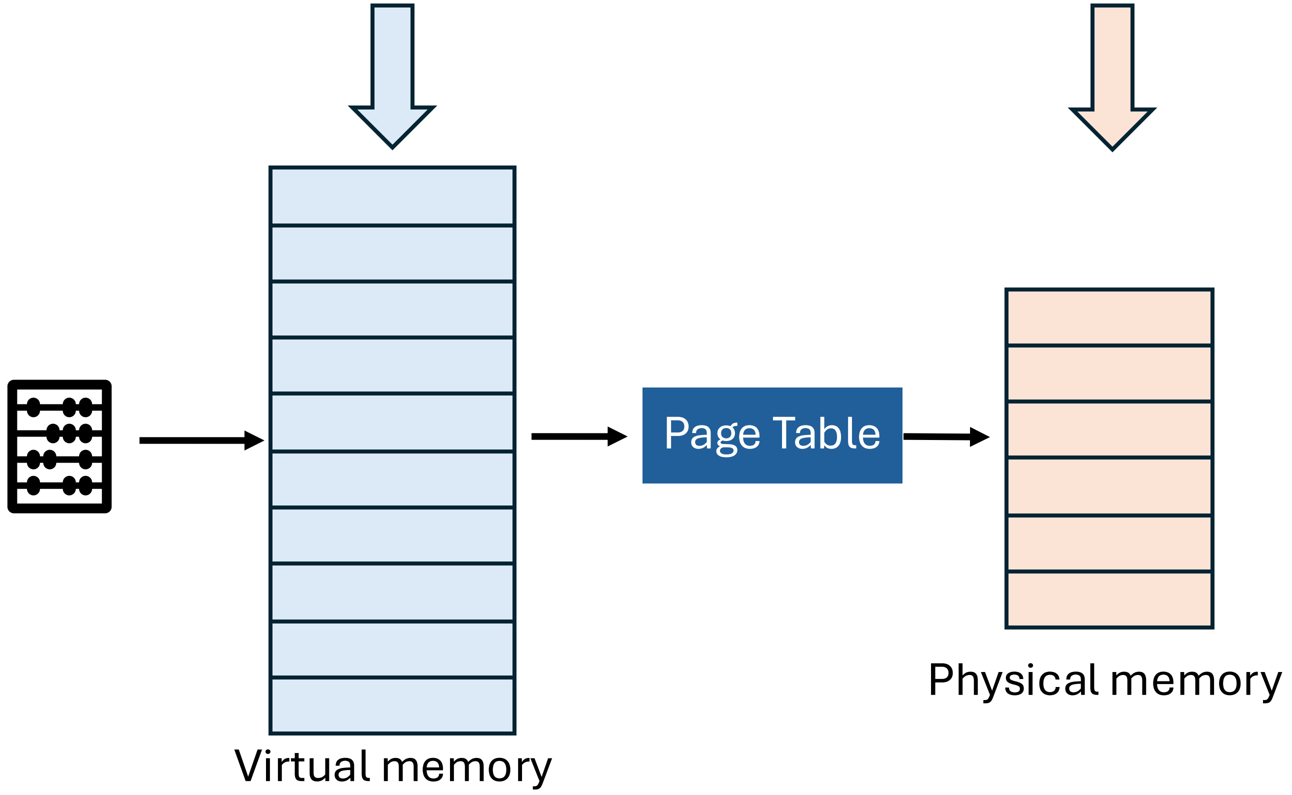
Programming

Writing performant GPU code is non-trivial

Enabling contiguous dynamic allocation

Contiguous layout

Dynamic allocation



Virtual memory is abundant
(128TB per process)

allocate large chunks, ahead-of-time

Physical memory is limited
(80GB per GPU)

allocate small chunks, on demand

Key idea: Let's allocate them separately

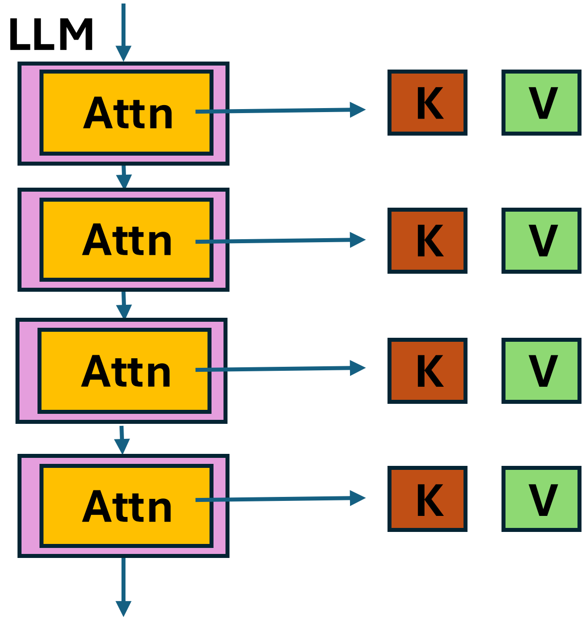
vAttention

Decoupling virtual and physical memory allocation

Leveraging system support for demand paging

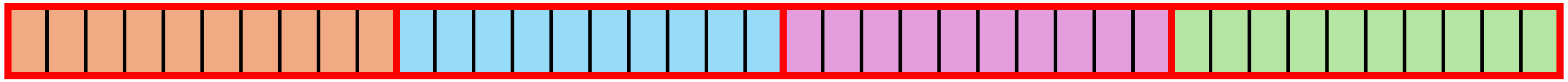
Optimizations (co-design with LLM inference)

Memory allocation in vAttention



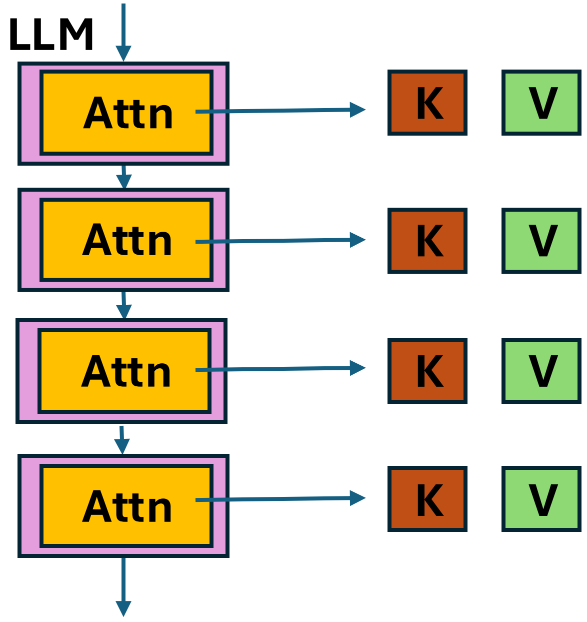
- Each worker allocates 2*N virtual tensors
  - N = number of layers hosted on the worker
  - Separate tensors for K and V at each layer
  - Size based on max context length and batch size

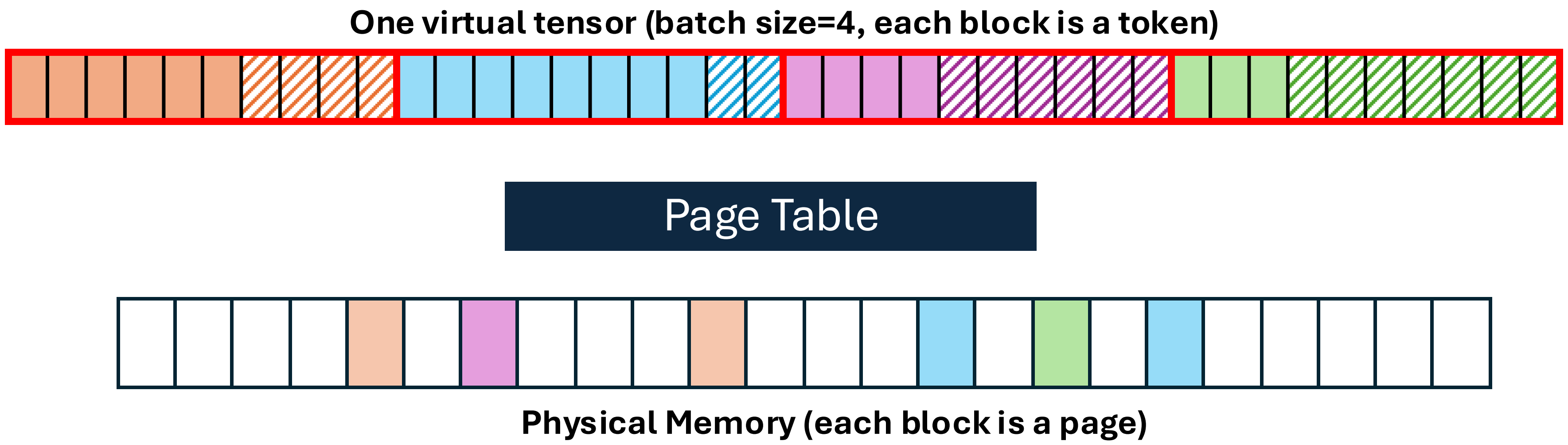
One virtual tensor (batch size=4, each block is a token)



Memory allocation in vAttention

- Each worker allocates 2*N virtual tensors
  - N = number of layers hosted on the worker
  - Separate tensors for K and V at each layer
  - Size based on max context length and batch size





Challenges for vAttention

Allocating a physical memory page requires a syscall → **High Latency**

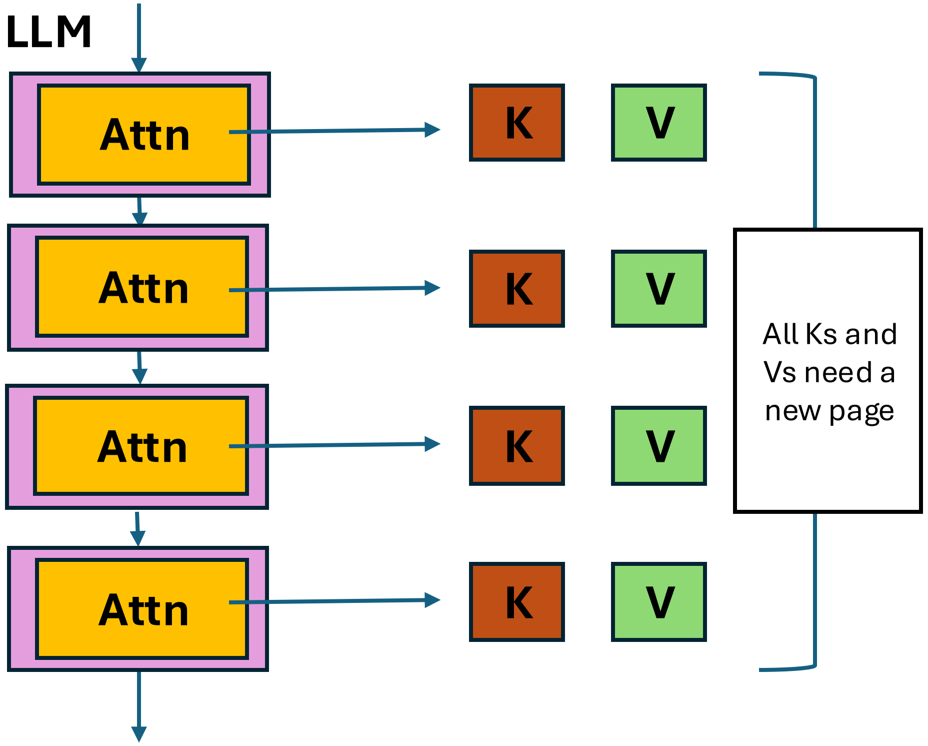
CUDA drivers allocate only large pages (>=2MB) → **High Fragmentation**

Challenge-1: Memory allocation latency

- Allocating one page takes **~40 μs**

- Need to allocate multiple pages at once
  - Latency overhead grows proportionally

- Example: **Yi-34B**
  - 60 layers == 120 virtual tensors
  - **5ms** (=120*40 μs) latency overhead per request
  - **50ms** overhead if 10 requests need new pages at once



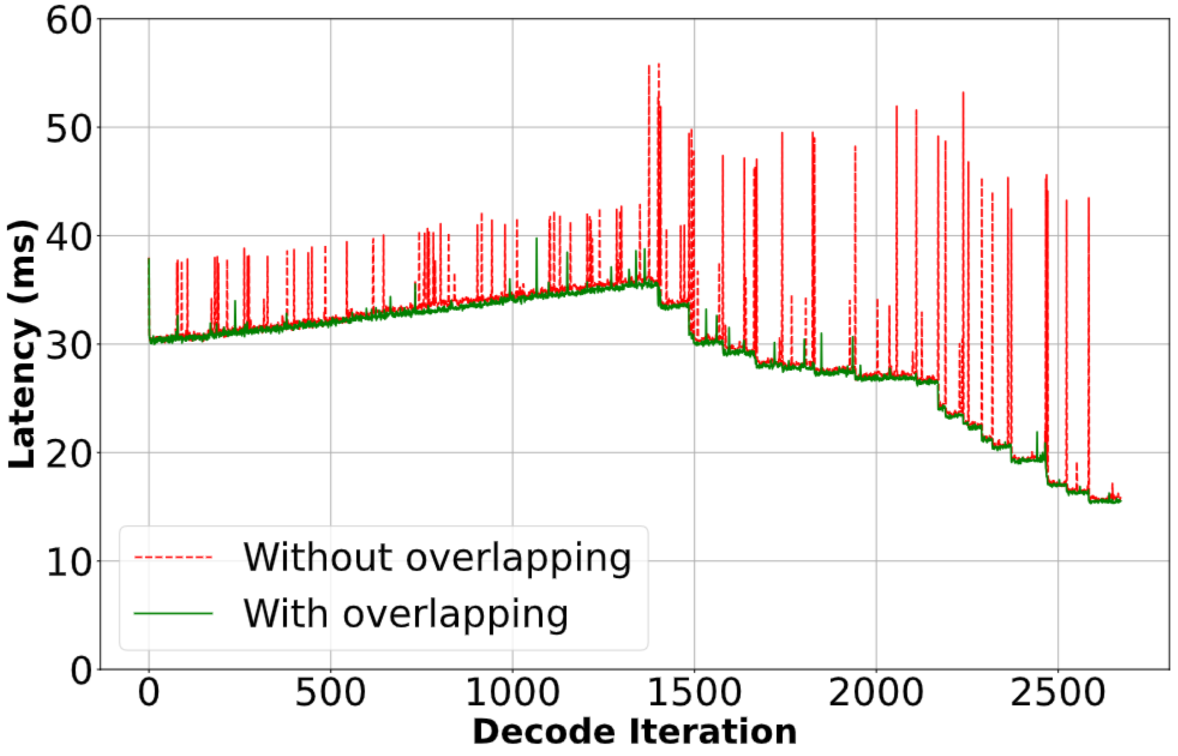
Optimization: Overlap allocation with compute

- Each decode iteration generates **one** token (per request)

- Memory requirement is known ahead-of-time (dream property!)

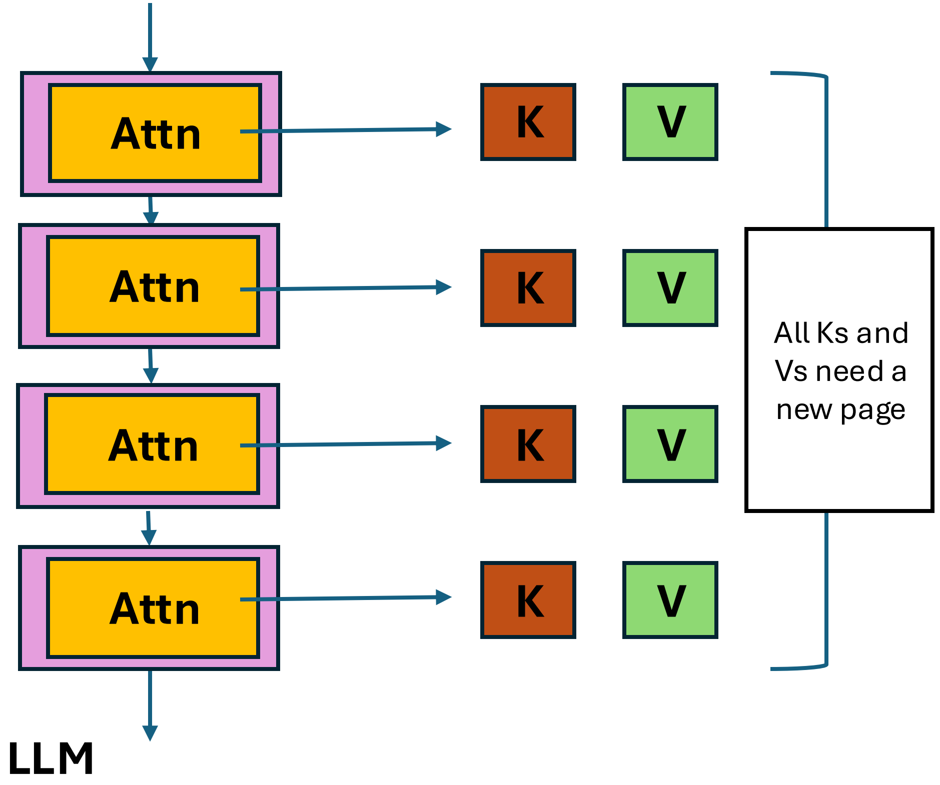
💡 Track progress of each request to determine when a new page is required

💡 **Asynchronously** allocate pages for iteration **i+1** when iteration **i** is executing



Challenge-2: Fragmentation (physical memory)

- Min page size in CUDA is **2MB**
- vAttention allocates 2*N pages at once
- Fragmentation proportional to:
  - Number of layers
  - Degree of tensor-parallelism



Challenge-2: Fragmentation (physical memory)

- Example: **Yi-34B**
  - 60 layers == 120 virtual tensors (per TP-worker)

Maximum memory wasted (per request) for Yi-34B

TP Dimension	Max Memory wasted
1	240MB
2	480MB
4	**960MB**
8	**1920MB**

Optimization: Allocate smaller physical pages

- GPUs natively support 4KB, 64KB and 2MB pages

💡 **Update CUDA drivers to allocate small (64KB) pages**

Maximum memory wasted (per request) for Yi-34B

TP dimension	64KB	2MB
1	7.5MB	240MB
2	15MB	480MB
4	30MB	960MB
8	**60MB**	**1920MB**

Up to 96% reduction in memory wastage

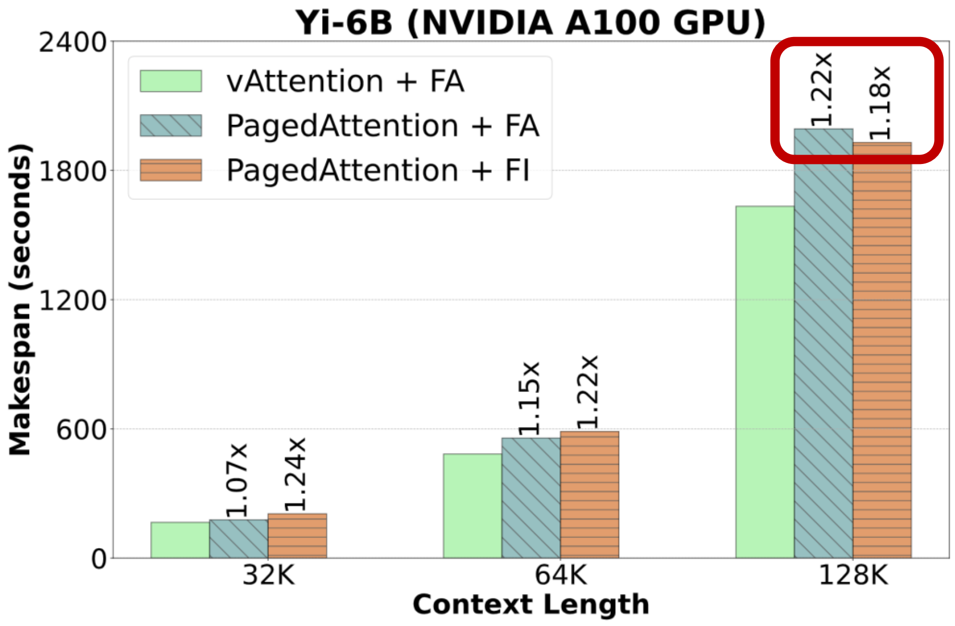
Challenges for vAttention

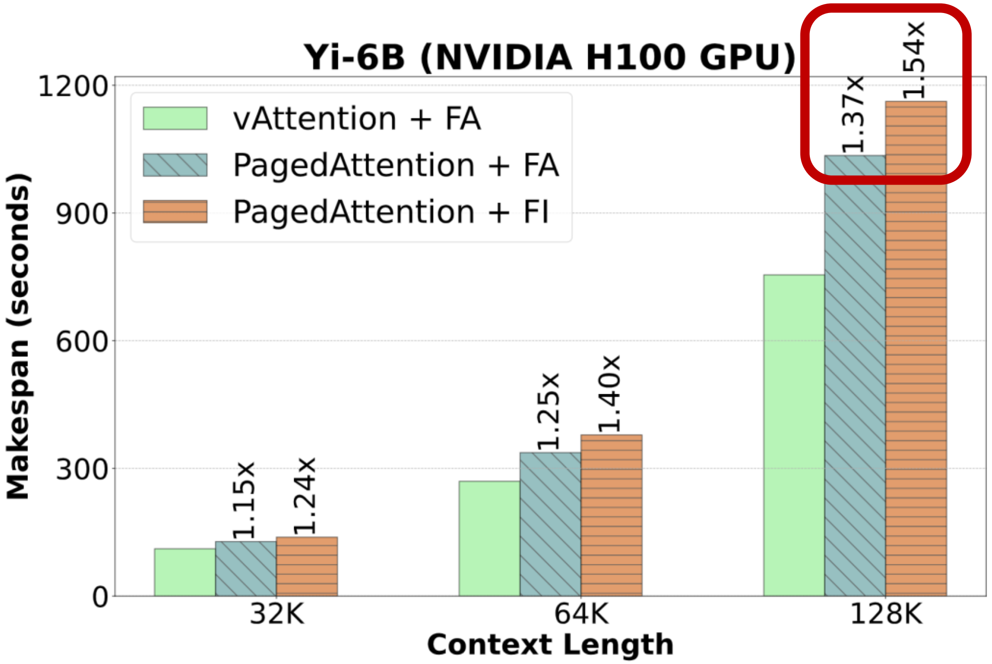
Allocating a physical memory pages requires a syscall → **High Latency** → **Async allocation**

CUDA drivers allocate only large pages (>=2MB) → **High Fragmentation** → **Small pages**

Evaluation

- FlashAttention-3 released on 11-July-2024 (up to 2x faster on H100)
- Does not support PagedAttention





FA: FlashAttention
FI: FlashInfer

vAttention supports FlashAttention-3 out-of-the-box

Summary

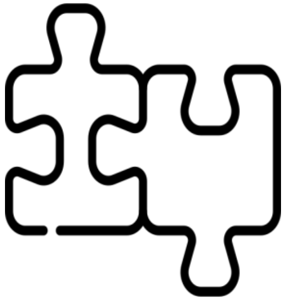
- **vAttention:** An alternative to PagedAttention
  - Leveraging system support for demand paging

Programming

Supports unmodified GPU implementations

Performance

Not impacted by dynamic memory allocation



Portability

Adopting new kernels is very easy