https://hao-ai-lab.github.io/cse234-w25/

CSE 234: Data Systems for Machine Learning
Winter 2025

LLMSys

Optimizations and Parallelization

MLSys Basics

Logistics

- If 80% of you finish the course eval, all get +2 points in final score!
  - Currently: we are 50%
- TA will hold a recitation for exam – make sure to attend

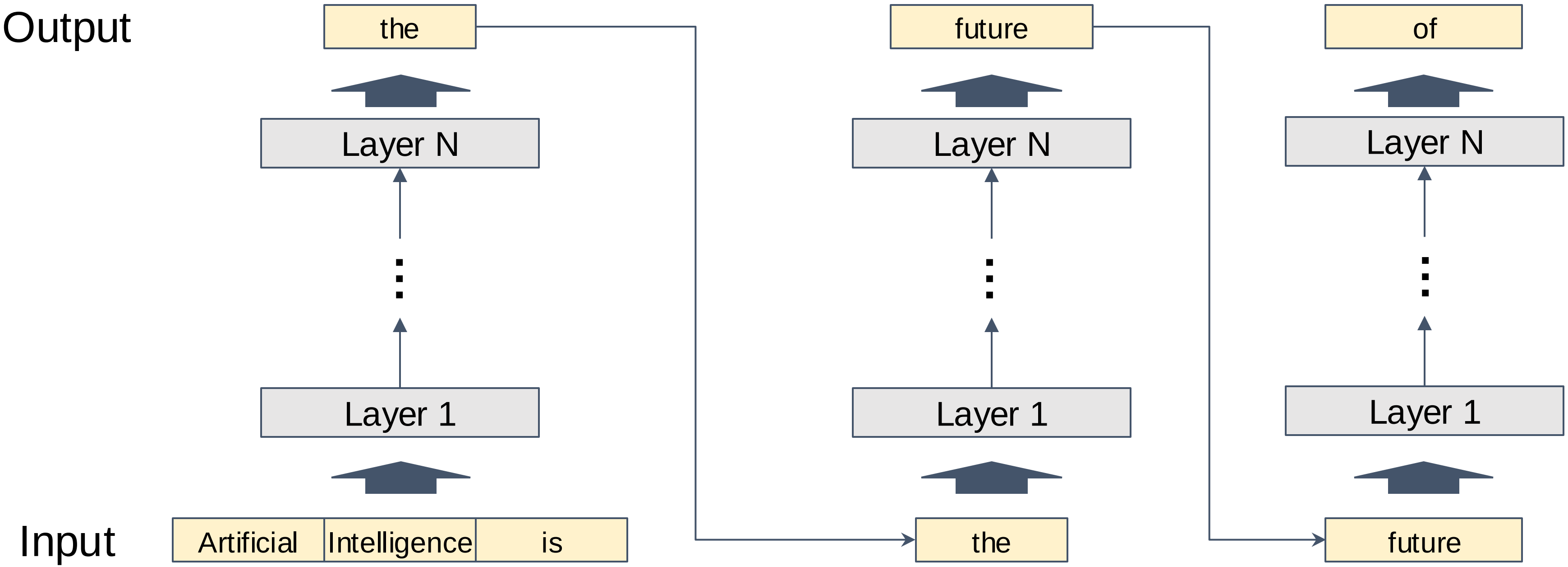
Recap: Next Token Prediction

Probability("San Diego has very nice weather")
= P("San Diego") P("has" | "San Diego")P("very" | "San Diego has")P("city" | . . .). . .P("weather" | . . .)

$$\text{Max}\,Prob(x_{1:T}) = \boxed{\prod_{t=1}^{T} P(x_{t+1}|x_{1\ldots t})}$$

This is model we got – capable of "predicting the next token".

Inference process of LLMs



Repeat until the sequence

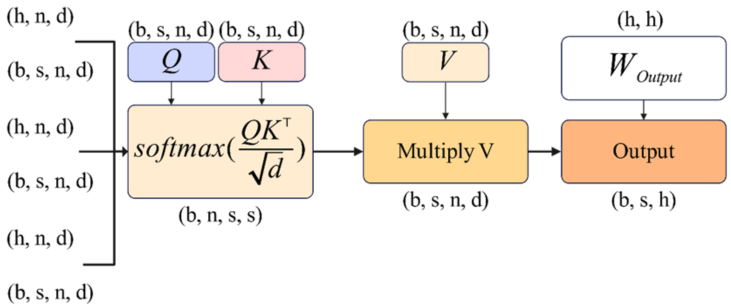
- Reaches its pre-defined maximum length (e.g., 2048 tokens)
- Generates certain tokens (e.g., “<|*end of sequence*|>”)

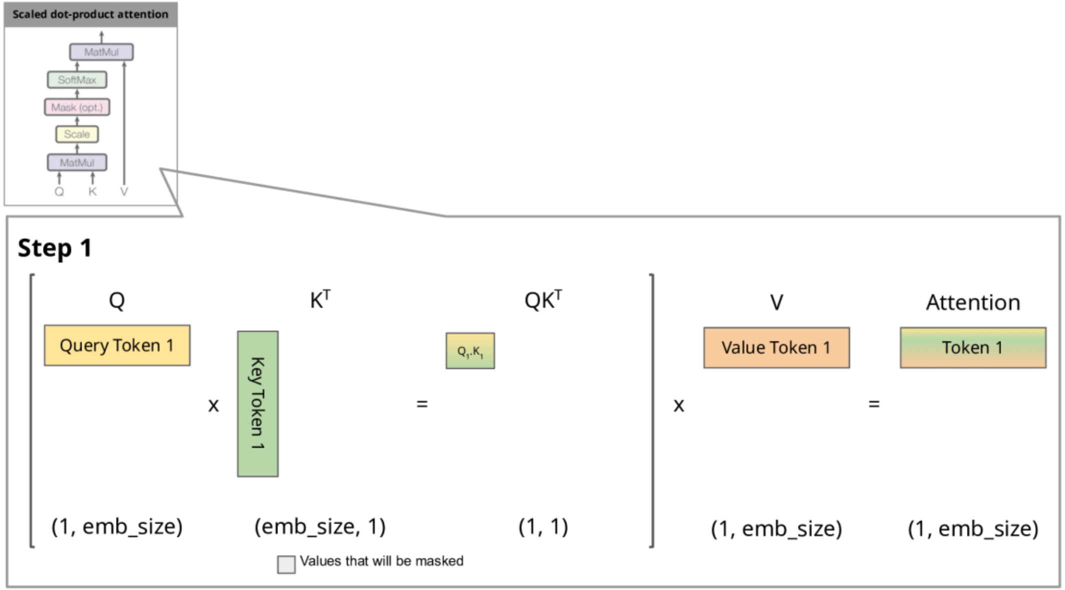
Generative LLM Inference: Autoregressive Decoding

- Pre-filling phase (0-th iteration):
  - Process *all* input tokens at once
- Decoding phase (all other iterations):
  - Process a *single* token generated from previous iteration

- Key-value cache:
  - Save attention keys and values for the following iterations to avoid recomputation
  - what is KV cache essentially?

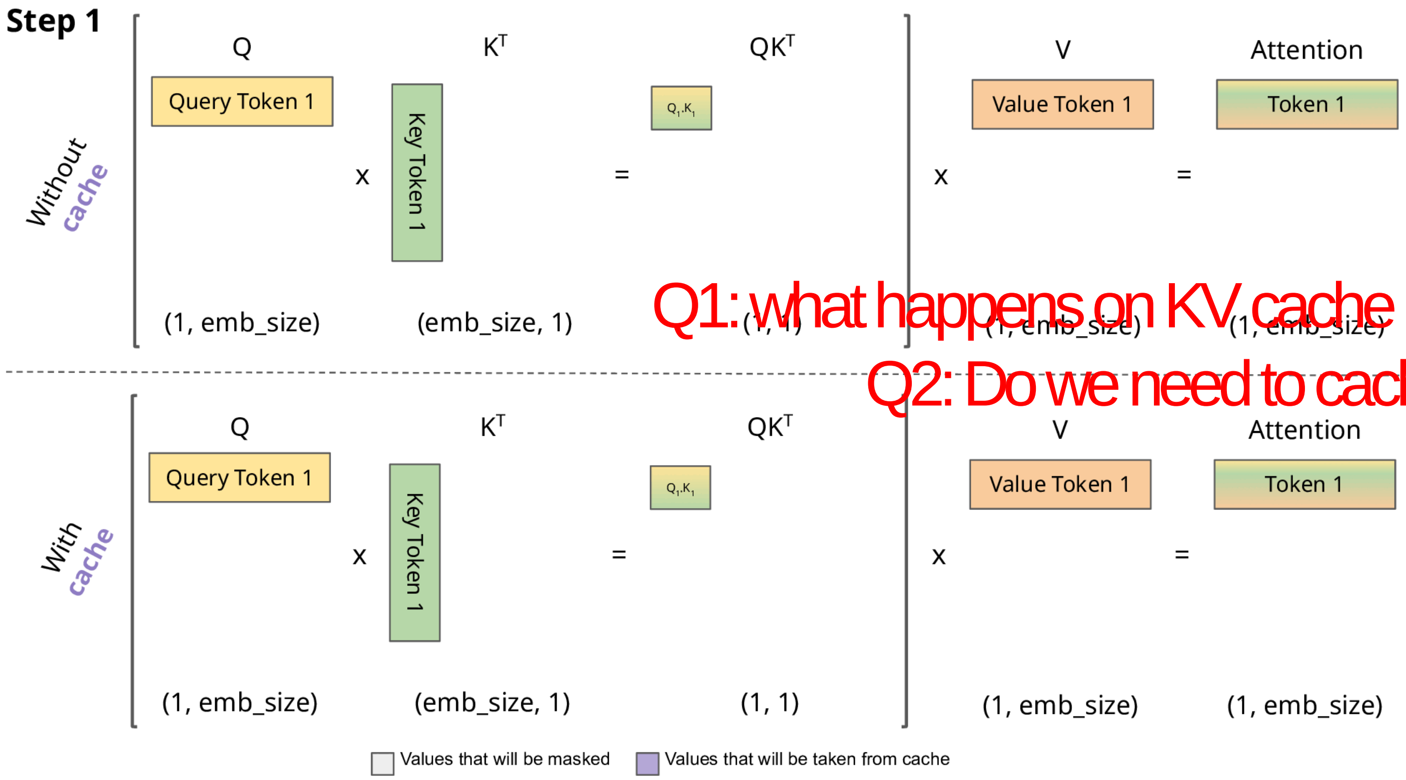
w/ KV Cache vs. w/o KV Cache





Zoom-in! (simplified without Scale and Softmax)

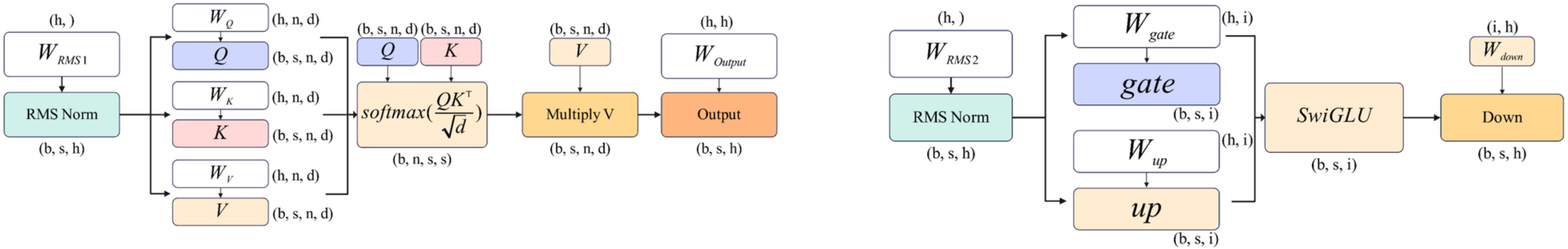
w/ KV Cache vs. w/o KV Cache



Q1: what happens on KV cache in prefill phase?

Q2: Do we need to cache Q?

Potential Bottleneck of LLM Inference?



- Compute:
  - Prefill: largely same with training
  - Decode: s = 1
- Memory
  - New: KV cache
- Communication
  - mostly same with training

Q? how about batch size b?

Serving vs. Inference

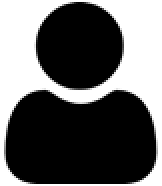
large b

Serving: many requests, online traffic, emphasize cost-per-query.

s.t. some mild latency constraints

emphasize **throughput**

b=1

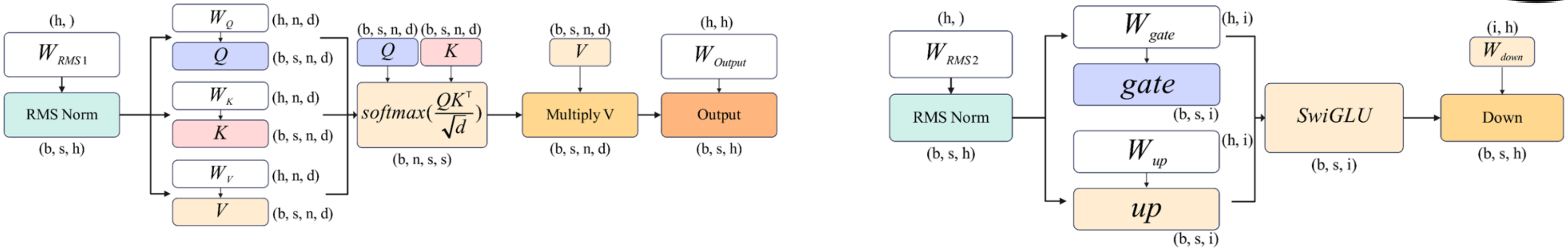


Inference: fewer request, low or offline traffic,

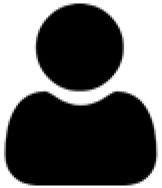
emphasize **latency**

large b

Potential Bottleneck of LLM Inference in Serving

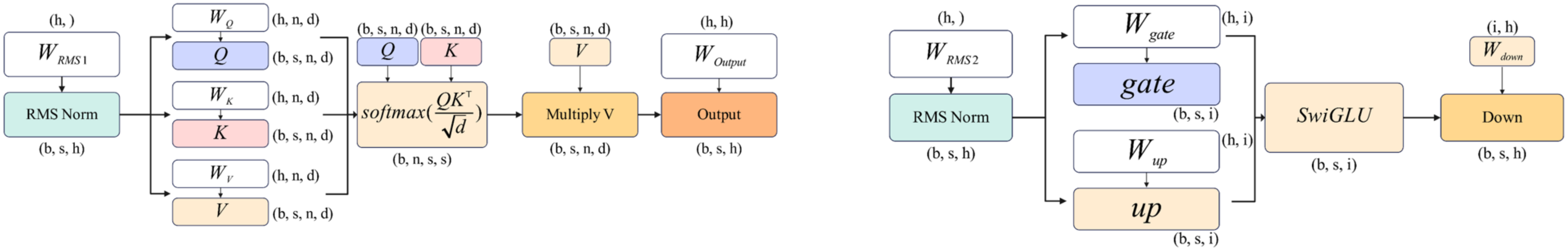


- Compute:
  - Prefill:
    - Different prompts have different length: how to batch?
  - Decode
    - Different prompts have different, unknown #generated tokens
    - s = 1, b is large
- Memory
  - New: KV cache
  - b is large -> KV is linear with b -> will KVs be large?
- Communication
  - mostly same with training



b=1

Potential Bottleneck of LLM Inference in Serving

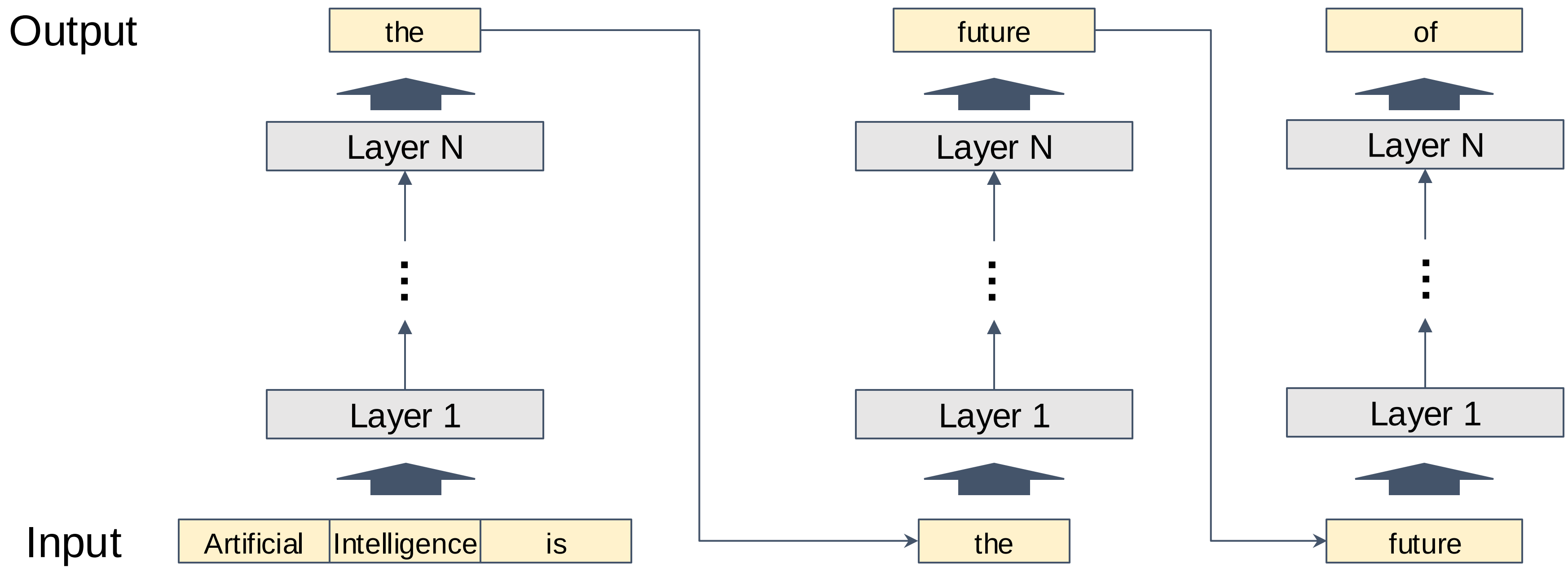


- Compute:
  - Prefill:
    - ~~Different prompts have different length: how to batch?~~
  - Decode
    - Different prompts have different, unknown #generated tokens
    - s = 1, b=1
- Memory
  - New: KV cache
  - ~~b =1 -> KV is linear with b -> will KVs be large?~~
- Communication
  - mostly same with training

Problems of bs = 1

max AI = #ops / #bytes

Recap: Inference process of LLMs



Repeat until the sequence

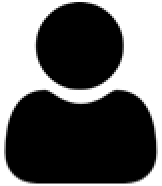
- Reaches its pre-defined maximum length (e.g., 2048 tokens)
- Generates certain tokens (e.g., “<|*end of sequence*|>”)

Problem of bs = 1

b=1

Latency = step latency * # steps

Speculative decoding reduces this, hence amortize the memory moving cost (but it may increase compute cost) b=1



Large Language Models

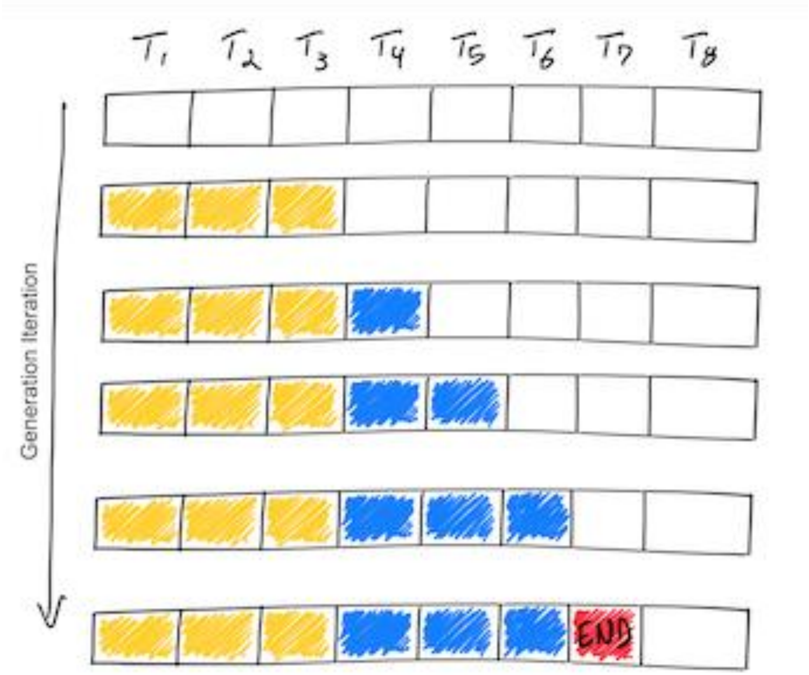
- Transformers, Attentions
- Scaling Law
  - MoE
- Connecting the dots: Training Optimizations
  - Flash attention ← come back to this later next week
- Serving and inference optimization
  - Continuous batching and Paged attention
  - Speculative decoding (Guest Lecture)
- Connecting the dots: Deepseek-v3
- Hot topics

large b

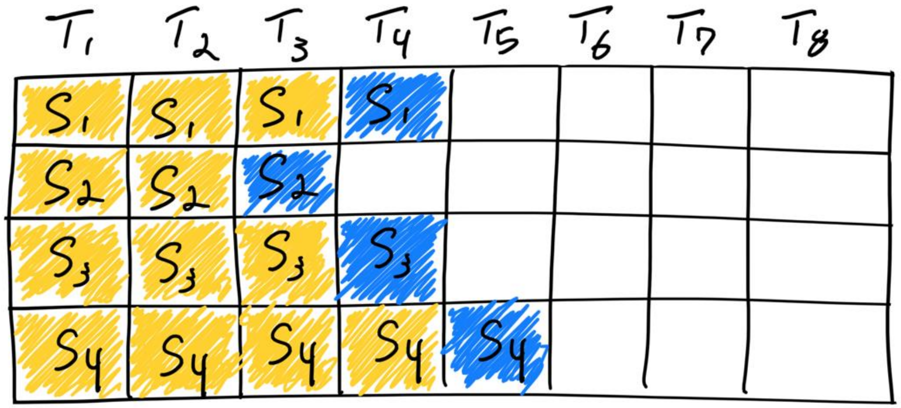
Large Language Models

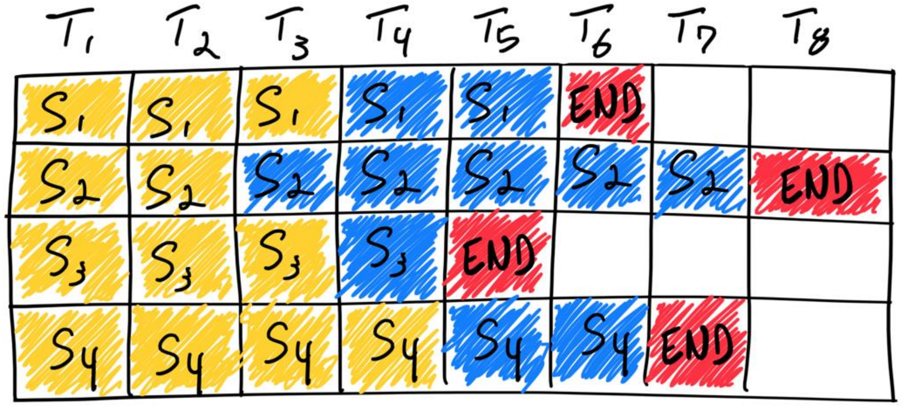
- Transformers, Attentions
- Scaling Law
  - MoE
- Connecting the dots: Training Optimizations
  - Flash attention ← come back to this later next week
- Serving and inference optimization
  - **Continuous batching and Paged attention**
  - Speculative decoding (Guest Lecture)
- Connecting the dots: Deepseek-v3
- Hot topics

LLM Decoding Timeline



Batching Requests to Improve GPU Performance

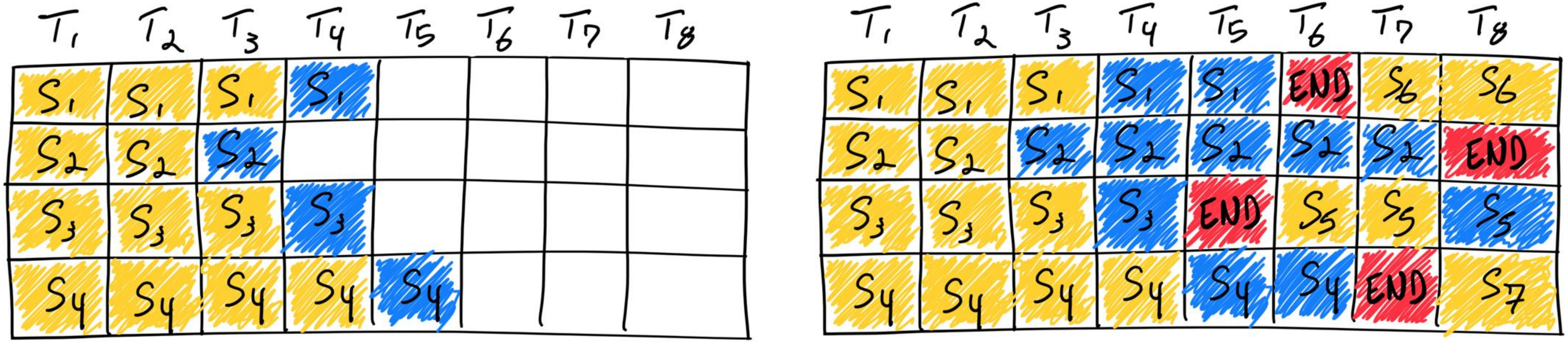




Issues with static batching:

- Requests may complete at different iterations
- Idle GPU cycles
- New requests cannot start immediately

Continuous Batching



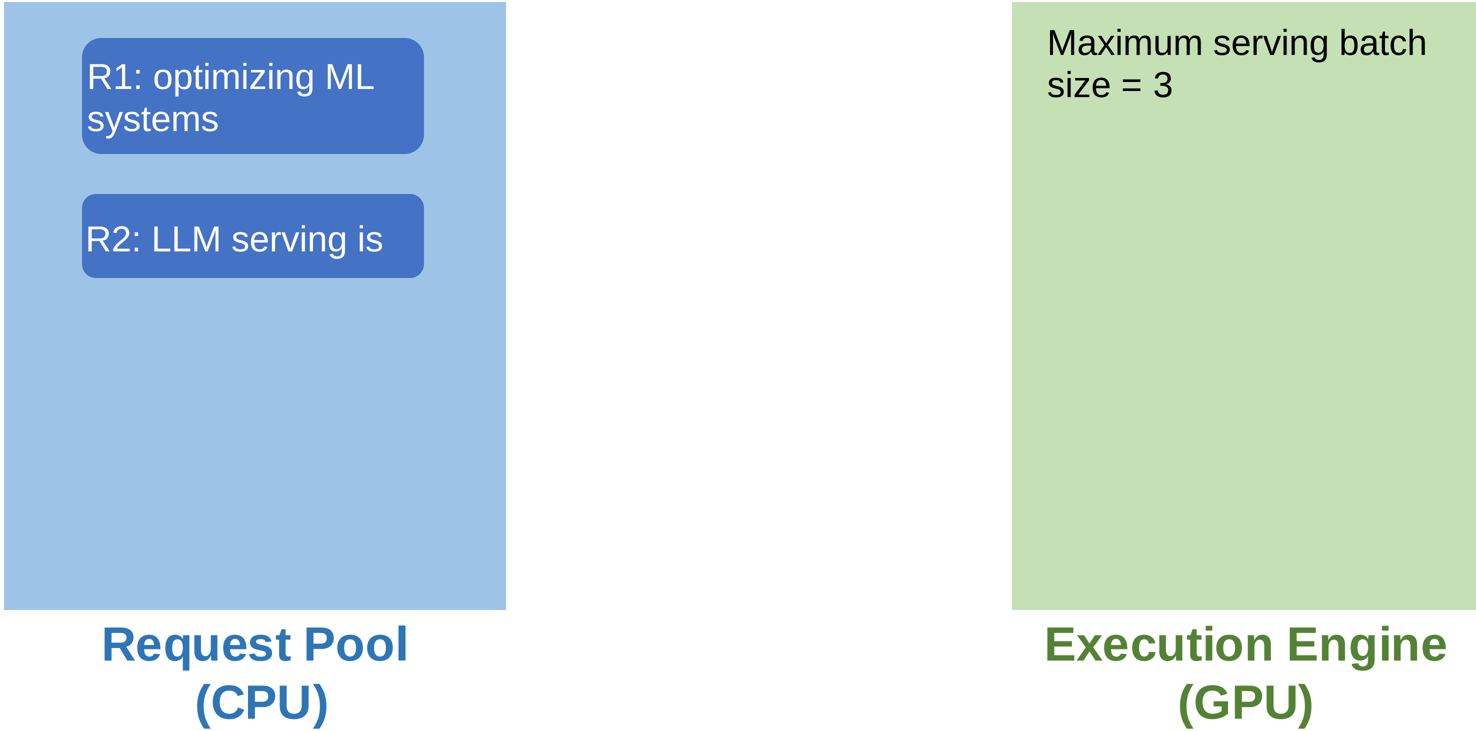
Benefits:

- Higher GPU utilization
- New requests can start immediately

Orca: A Distributed Serving System for Transformer-Based Generative Models. OSDI'22

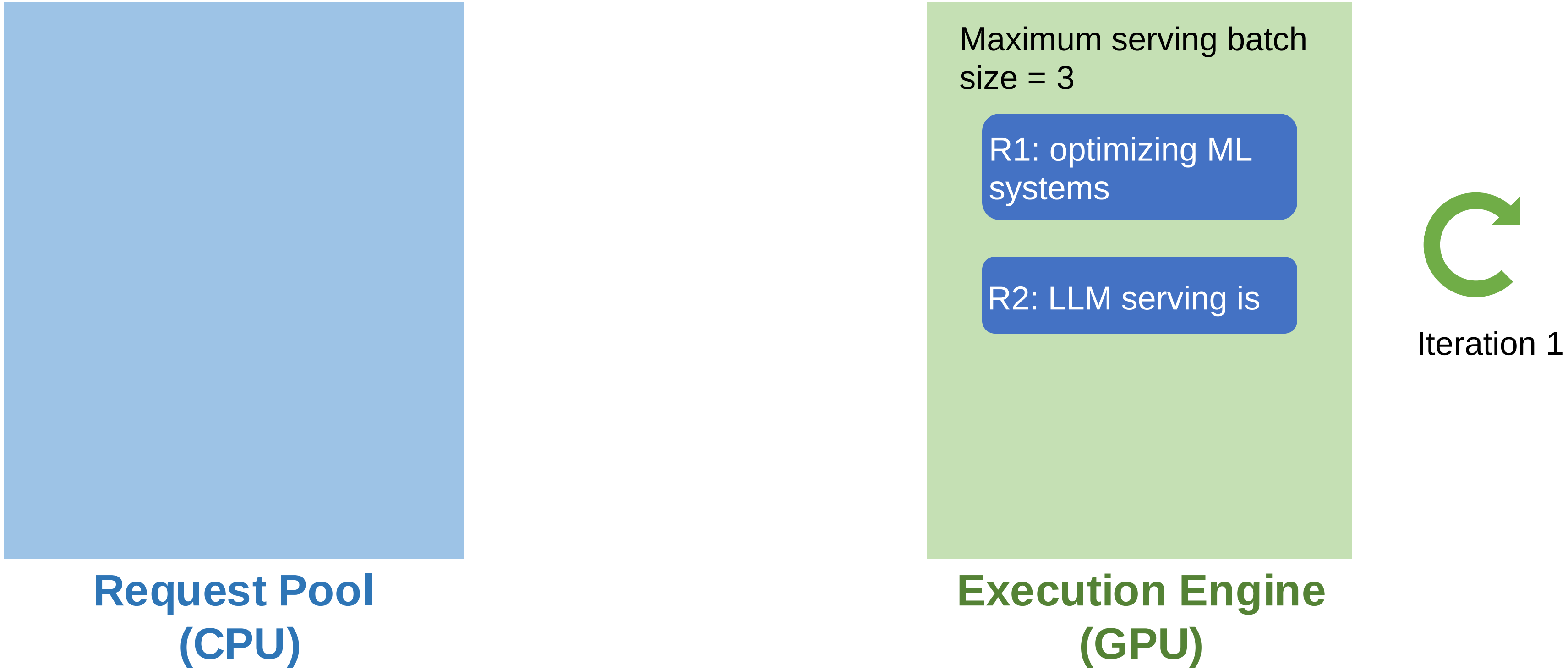
Continuous Batching Step-by-Step

- Receives two new requests R1 and R2



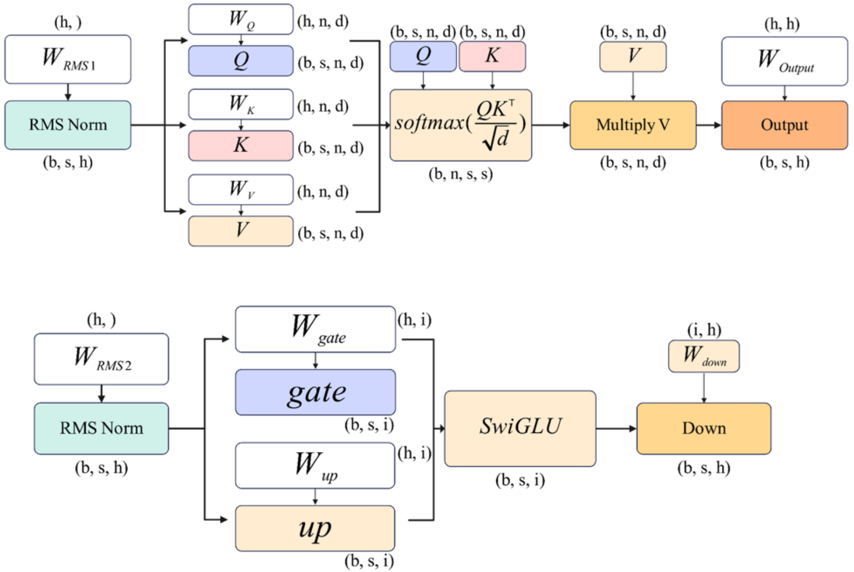
Continuous Batching Step-by-Step

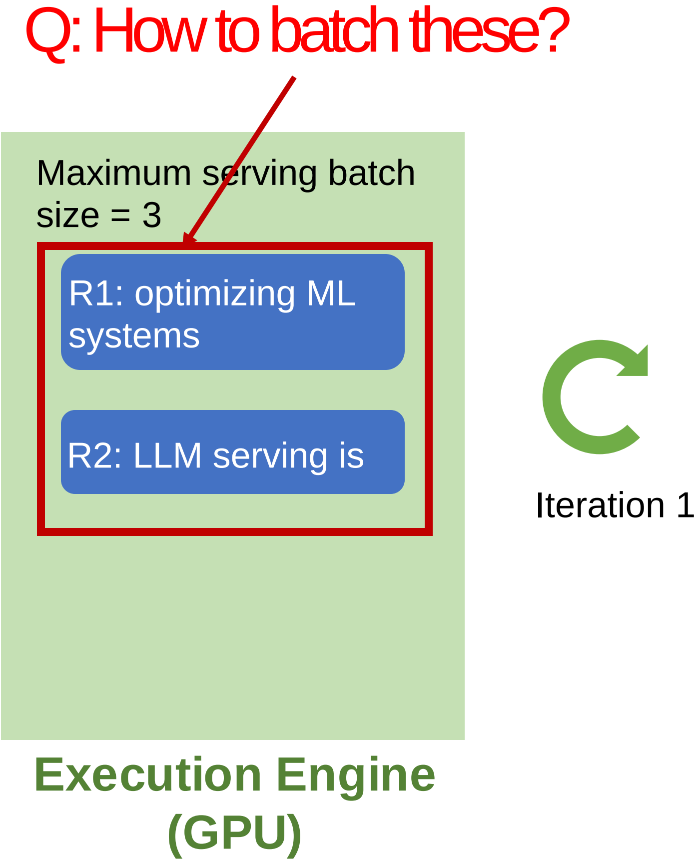
- Iteration 1: decode R1 and R2



Continuous Batching Step-by-Step

- Iteration 1: decode R1 and R2





Continuous Batching Step-by-Step

- Receive a new request R3; finish decoding R1 and R2

R3: A man

Request Pool (CPU)

Maximum serving batch size = 3

R1: optimizing ML systems **requires**

R2: LLM serving is **critical**.

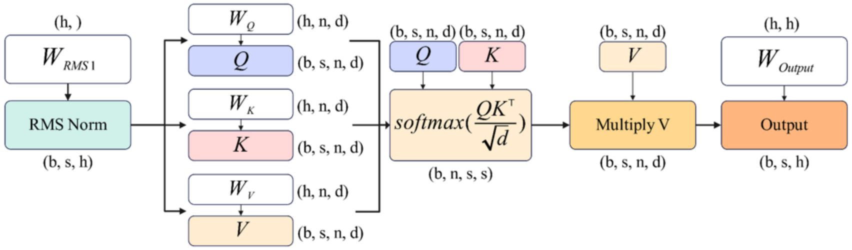
Execution Engine (GPU)

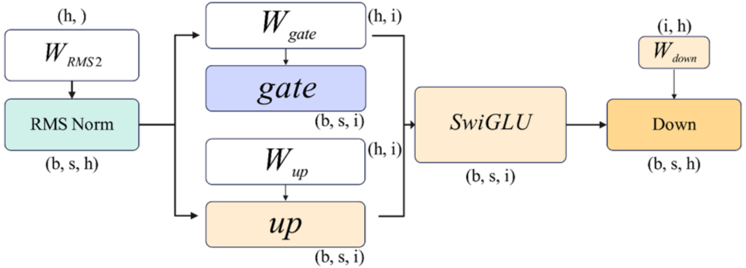
Iteration 1

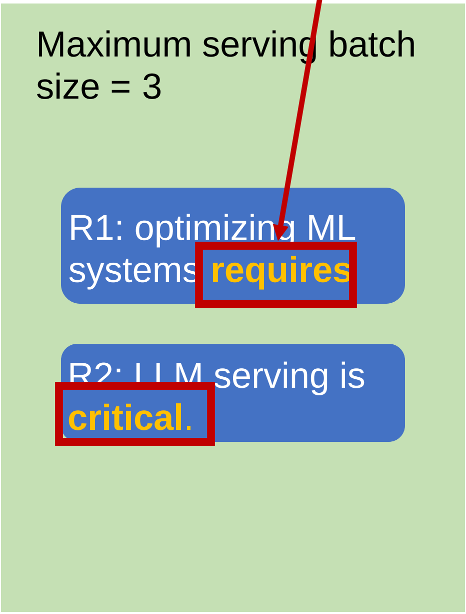
Continuous Batching Step-by-Step

Q: How to batch these?

- Receive a new request R3; finish decoding R1 and R2







Execution Engine (GPU)

Iteration 1

Traditional Batching

- Receive a new request R3; finish decoding R1 and R2

R3: A man

R4: A dog is

R5: How are

Request Pool (CPU)

Maximum serving batch size = 3

R1: optimizing ML systems **requires ML**

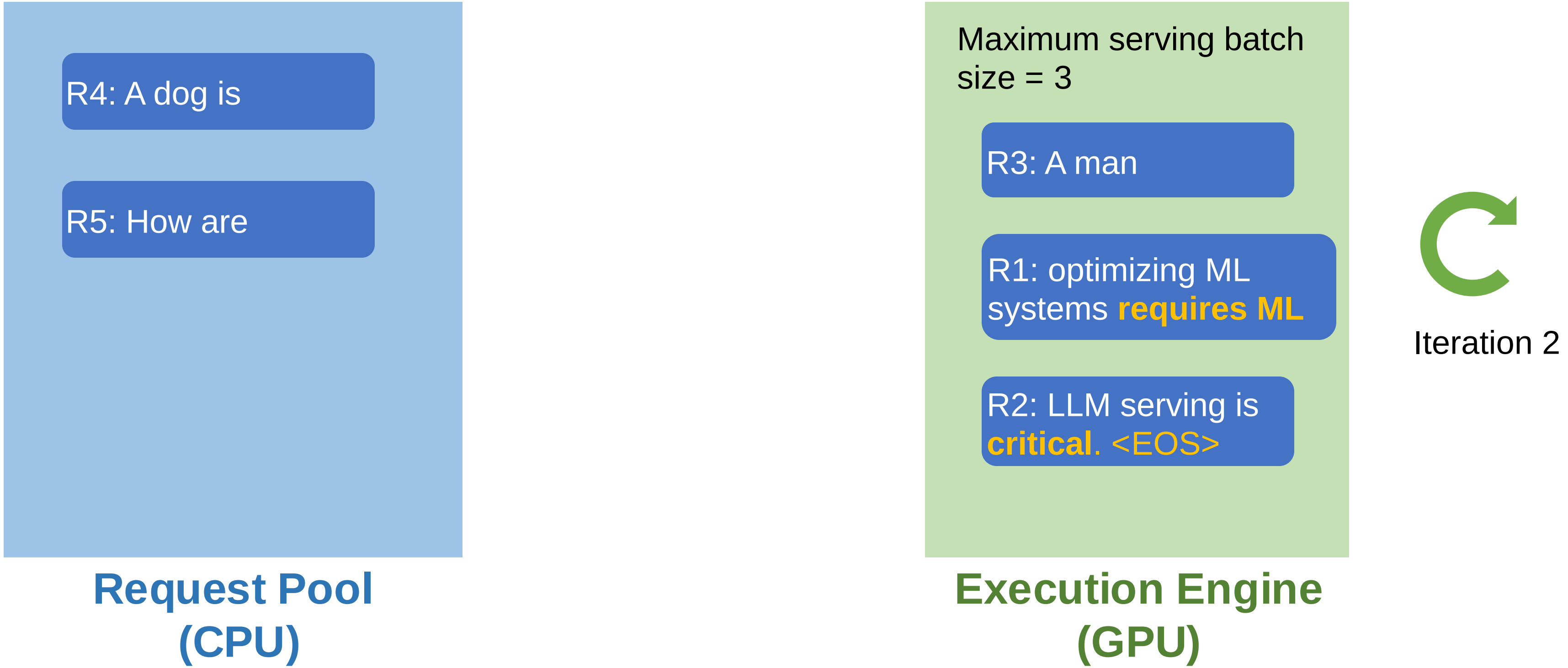
R2: LLM serving is **critical**. <EOS>

Execution Engine (GPU)

Iteration 2

Continuous Batching

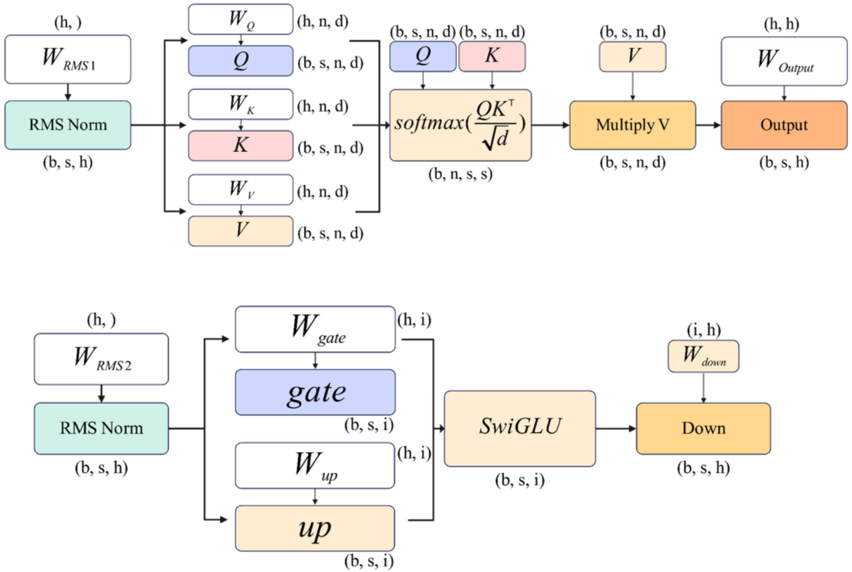
- Iteration 2: decode R1, R2, R3; receive R4, R5; R2 completes



Continuous Batching

Q: How to batch these?

- Iteration 2: decode R1, R2, R3; receive R4, R5; R2 completes



Maximum serving batch size = 3

R3: A man

R1: optimizing ML systems **requires ML**

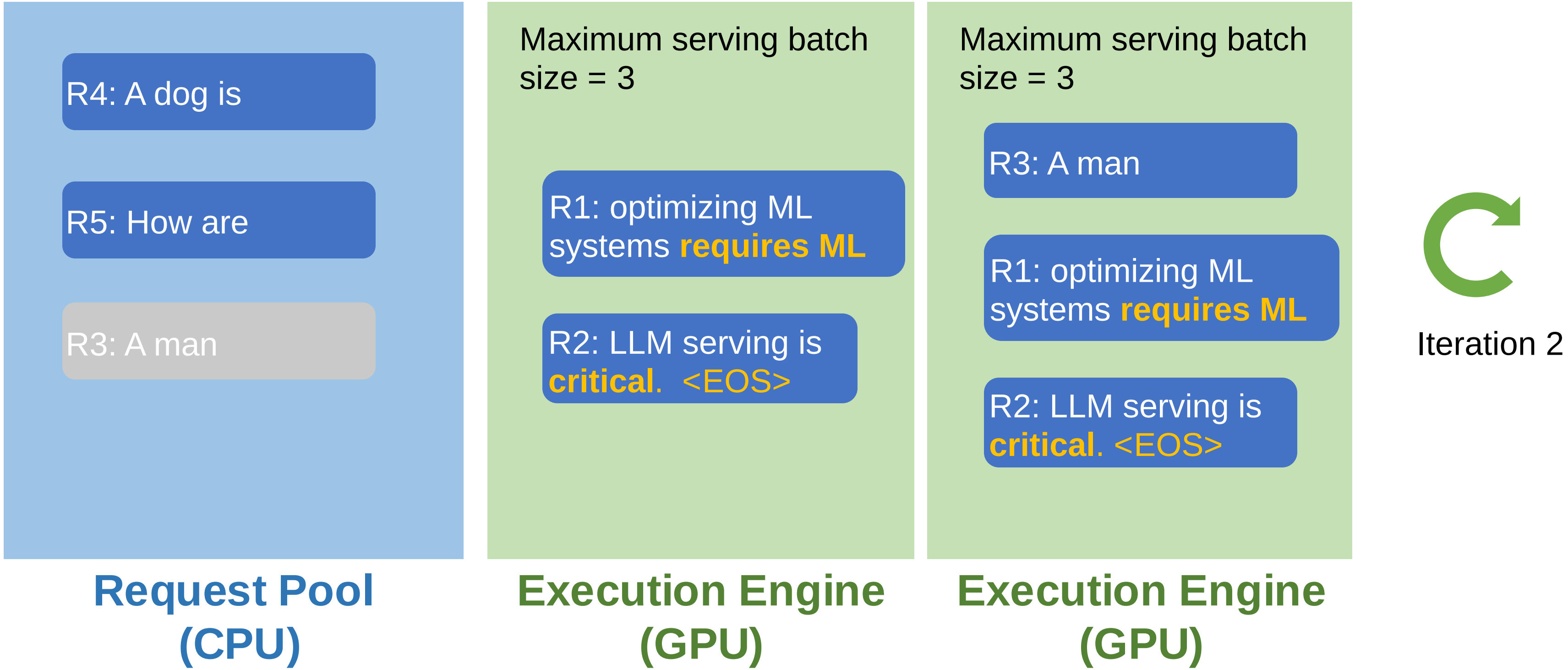
R2: LLM serving is **critical** <EOS>

Iteration 2

Execution Engine (GPU)

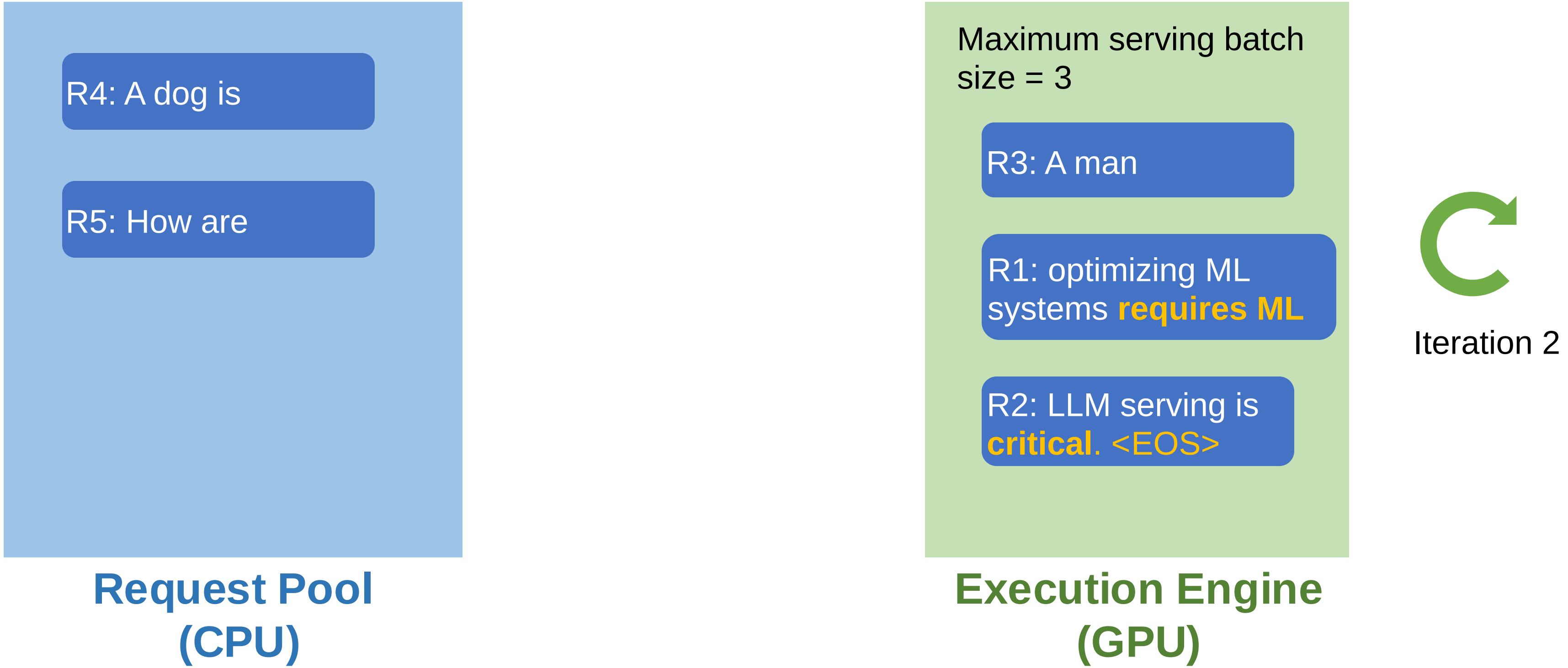
Traditional vs. Continuous Batching

- Iteration 2: decode R1, R2, R3; receive R4, R5; R2 completes



Continuous Batching

- Iteration 2: decode R1, R2, R3; receive R4, R5; R2 completes



Continuous Batching Step-by-Step

- Iteration 3: decode R1, R3, R4

R5: How are

Request Pool (CPU)

Maximum serving batch size = 3

R3: A man **is**

R1: optimizing ML systems **requires ML**

R4: A dog is

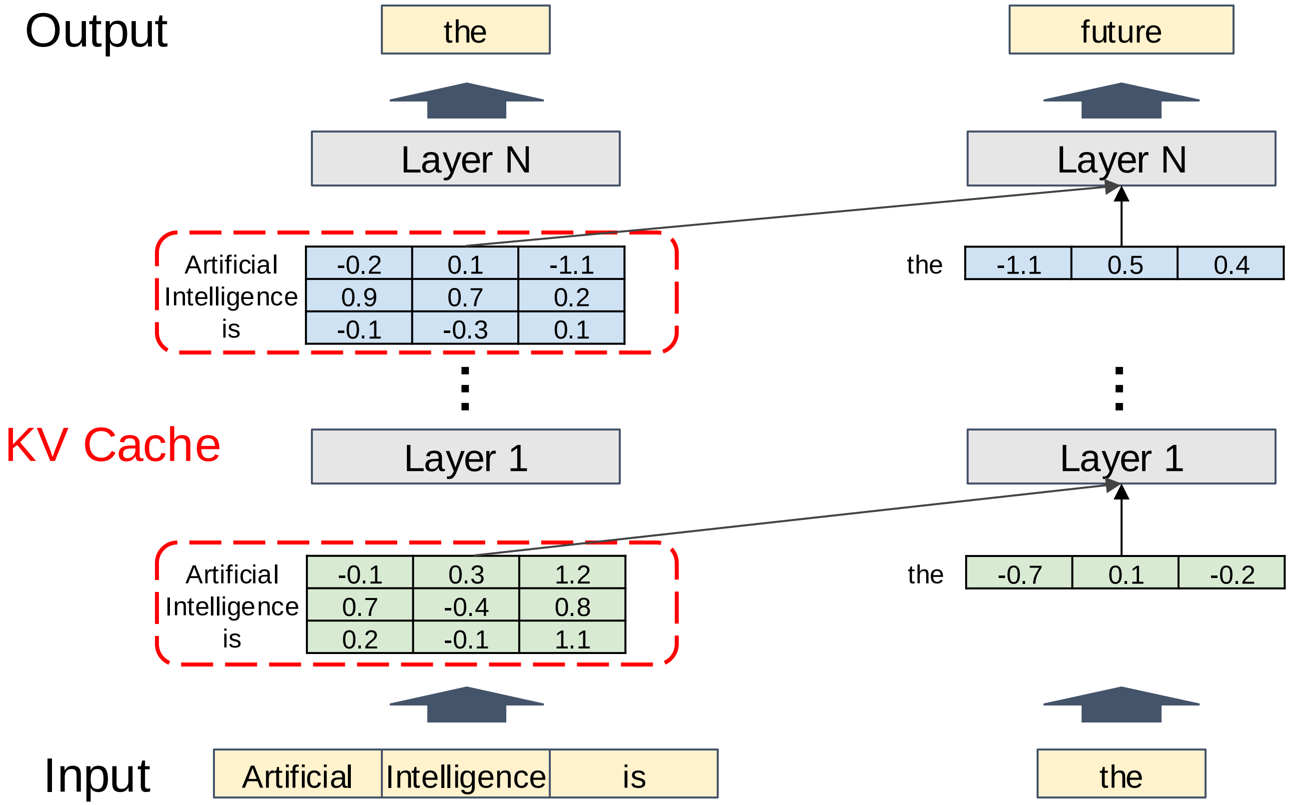
Execution Engine (GPU)

Iteration 3

Summary: Continuous Batching

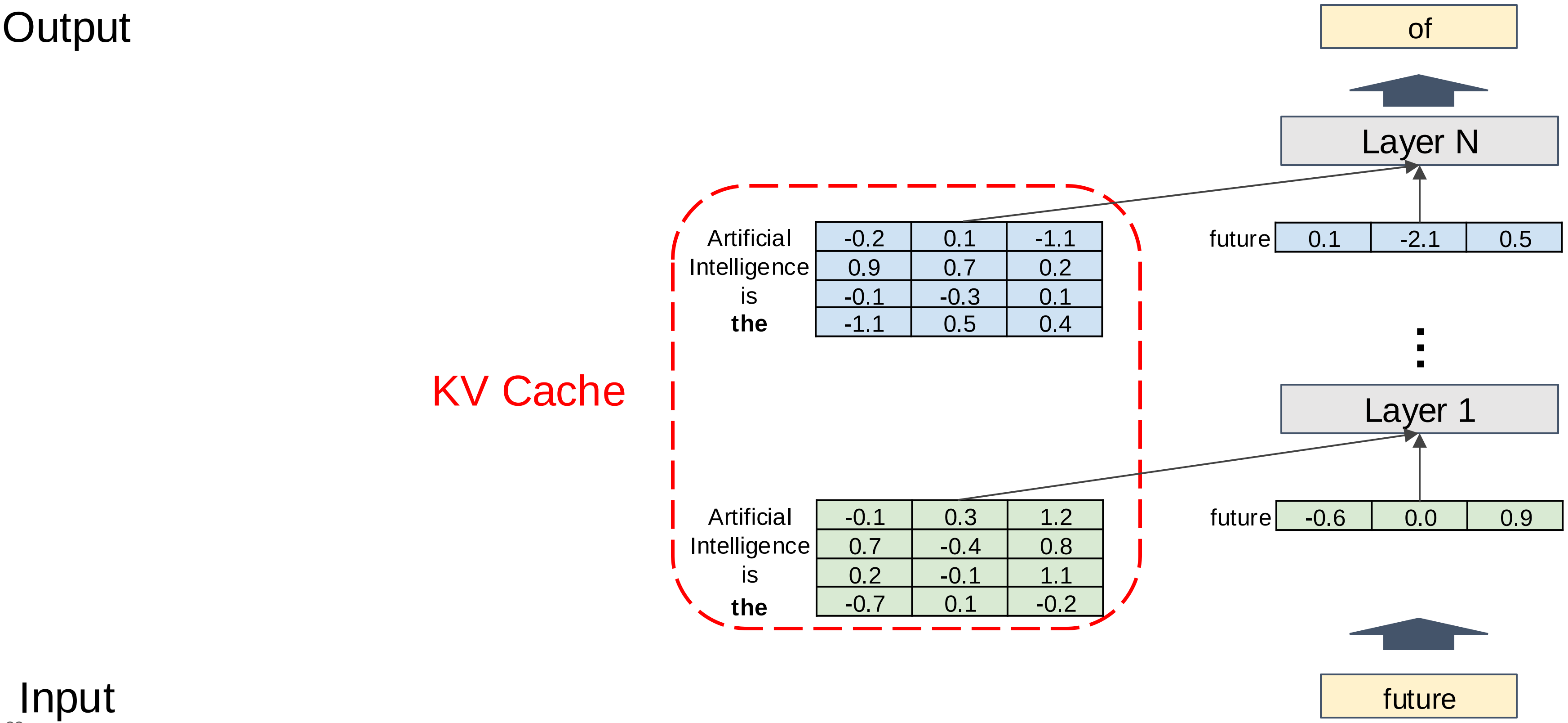
- Handle early-finished and late-arrived requests more efficiently
- Improve GPU utilization
- Key observation
  - MLP kernels are agnostic to the sequence dimension

KV Cache



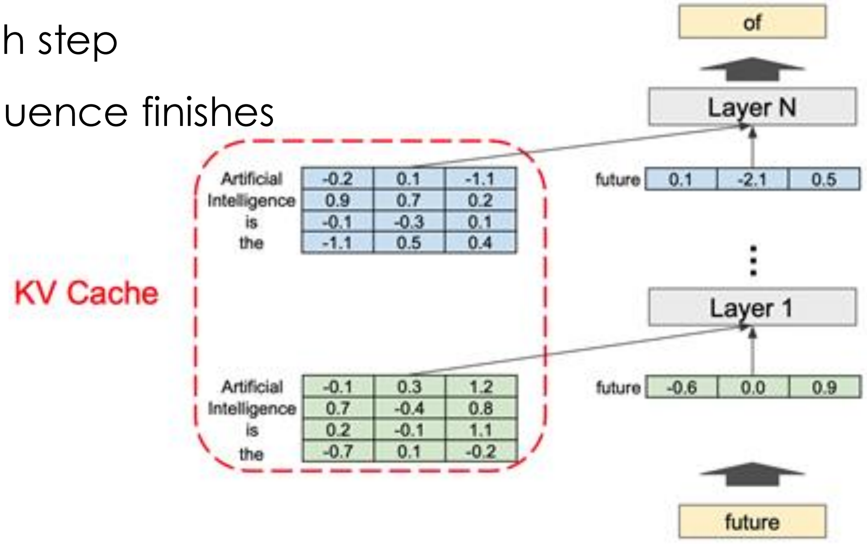
Output
the
future
Layer N
Layer N
Artificial
Intelligence
is
-0.2
0.1
-1.1
0.9
0.7
0.2
-0.1
-0.3
0.1
the
-1.1
0.5
0.4
KV Cache
Layer 1
Layer 1
Artificial
Intelligence
is
-0.1
0.3
1.2
0.7
-0.4
0.8
0.2
-0.1
1.1
the
-0.7
0.1
-0.2
Input
Artificial
Intelligence
is
the

KV Cache


Output
of
Layer N
Artificial -0.2 0.1 -1.1
Intelligence 0.9 0.7 0.2
is -0.1 -0.3 0.1
the -1.1 0.5 0.4
future 0.1 -2.1 0.5
KV Cache
Layer 1
Artificial -0.1 0.3 1.2
Intelligence 0.7 -0.4 0.8
is 0.2 -0.1 1.1
the -0.7 0.1 -0.2
future -0.6 0.0 0.9
future
Input

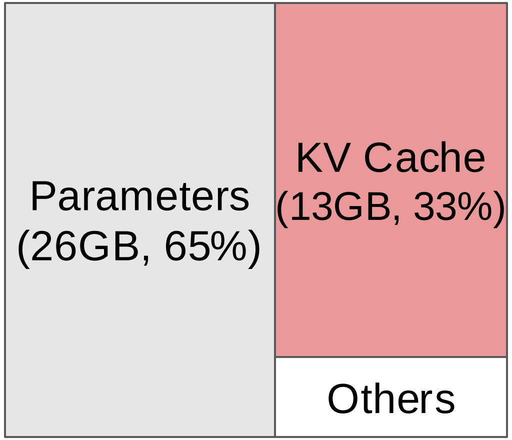
KV Cache

- Memory space to store intermediate vector representations of tokens
  - **Working set** rather than a "cache"
- The size of KV Cache dynamically grows and shrinks
  - A new token is appended in each step
  - Tokens are deleted once the sequence finishes

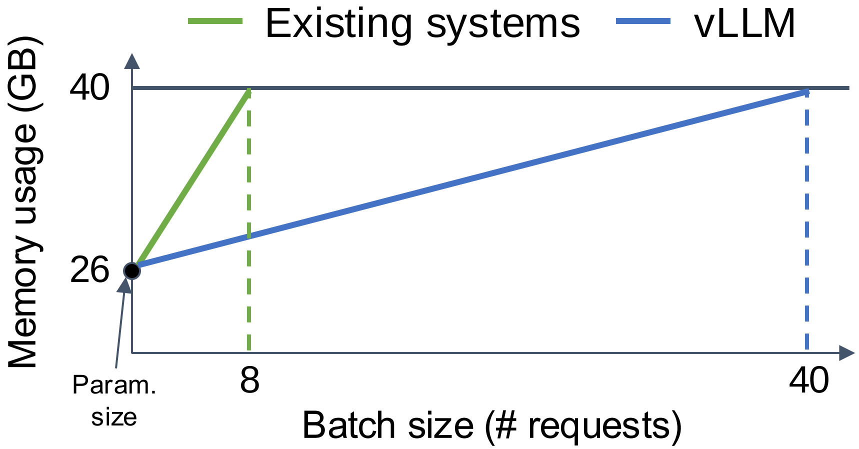


Key insight

Efficient management of KV cache is crucial for high-throughput LLM serving

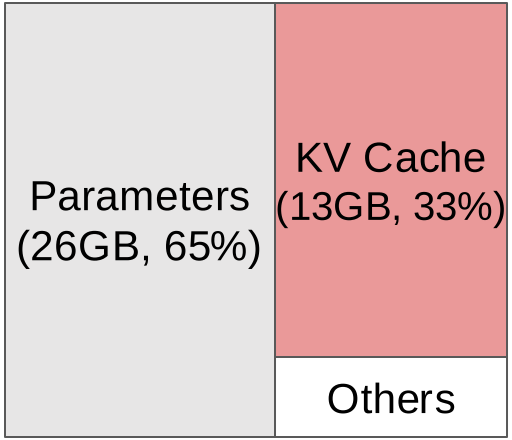


13B LLM on A100-40GB

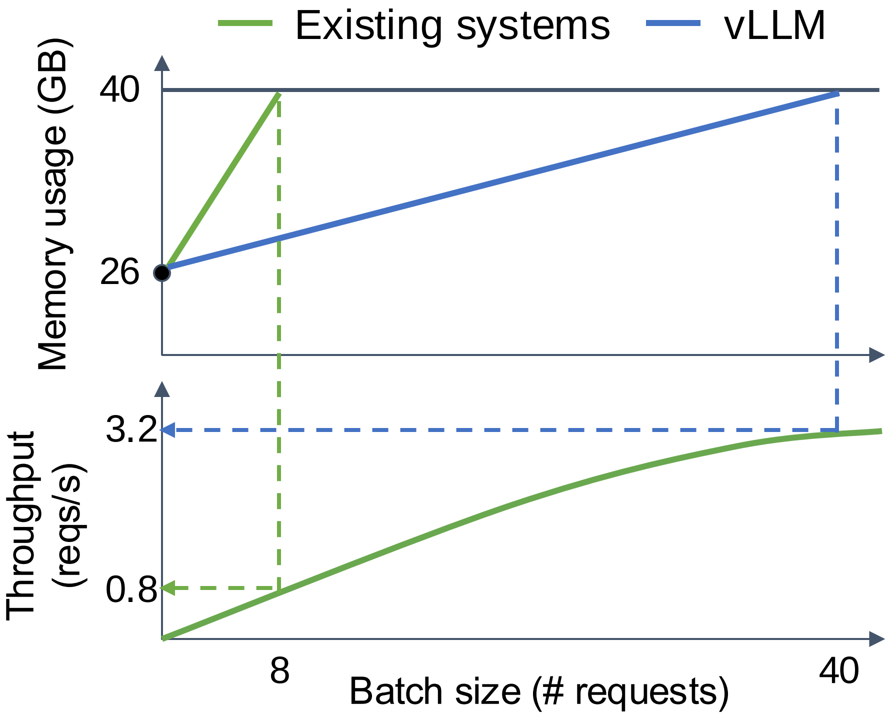


Key insight

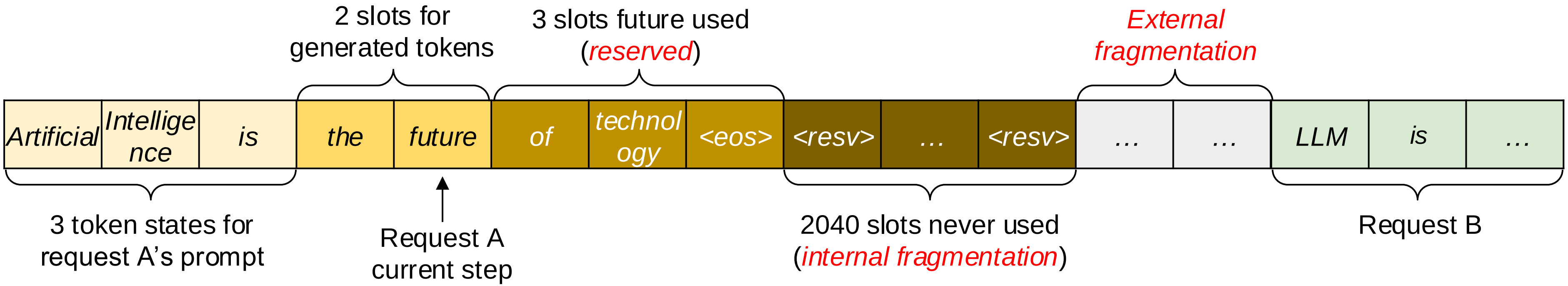
Efficient management of KV cache is crucial for high-throughput LLM serving



13B LLM on A100-40GB

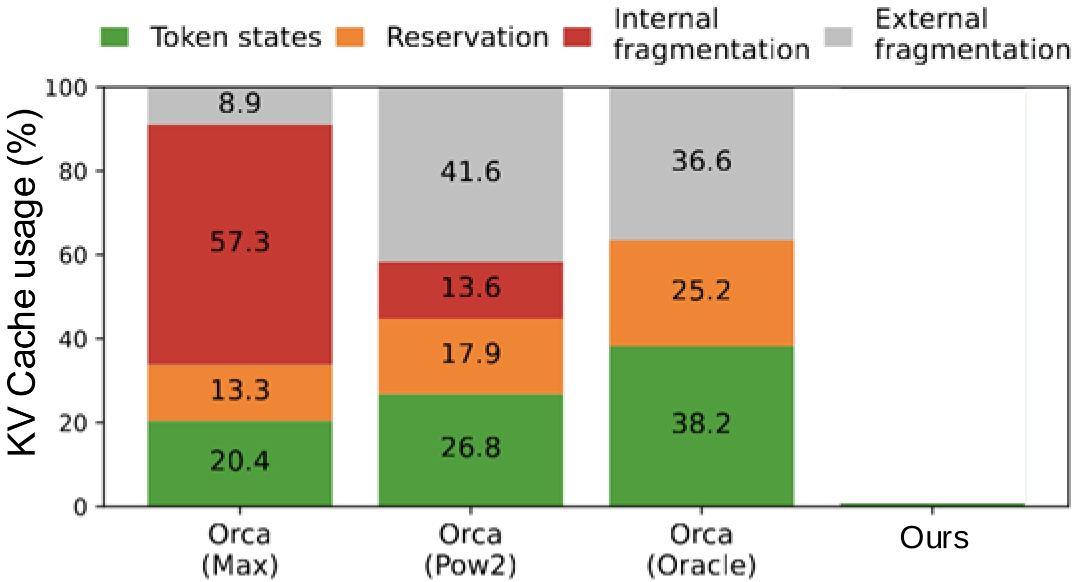


Memory waste in KV Cache



- Reservation: not used at the current step, but used in the future
- Internal fragmentation: over-allocated due to the unknown output length.

Memory waste in KV Cache

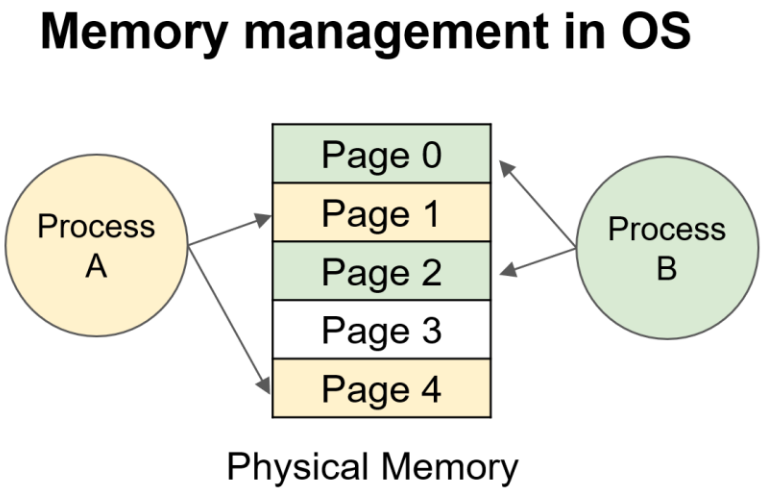


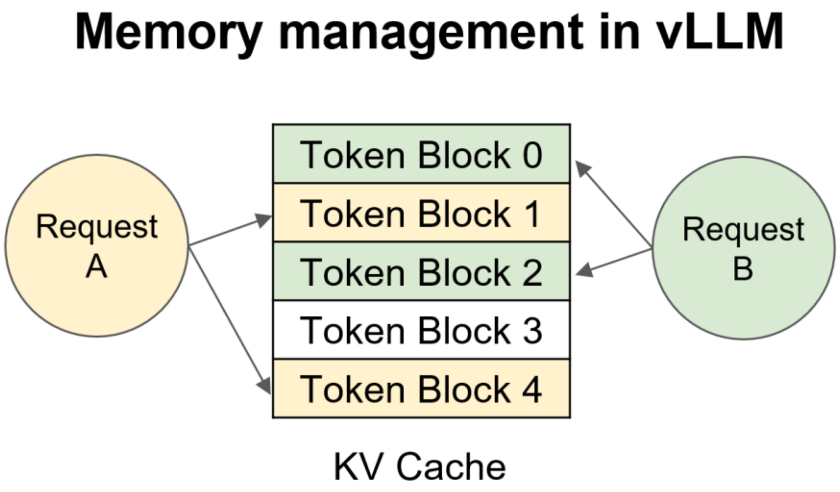
Only **20–40%** of KV cache is utilized to store token states

* Yu, G. I., Jeong, J. S., Kim, G. W., Kim, S., Chun, B. G. "Orca: A Distributed Serving System for Transformer-Based Generative Models" (OSDI 22).

vLLM: Efficient memory management for LLM inference

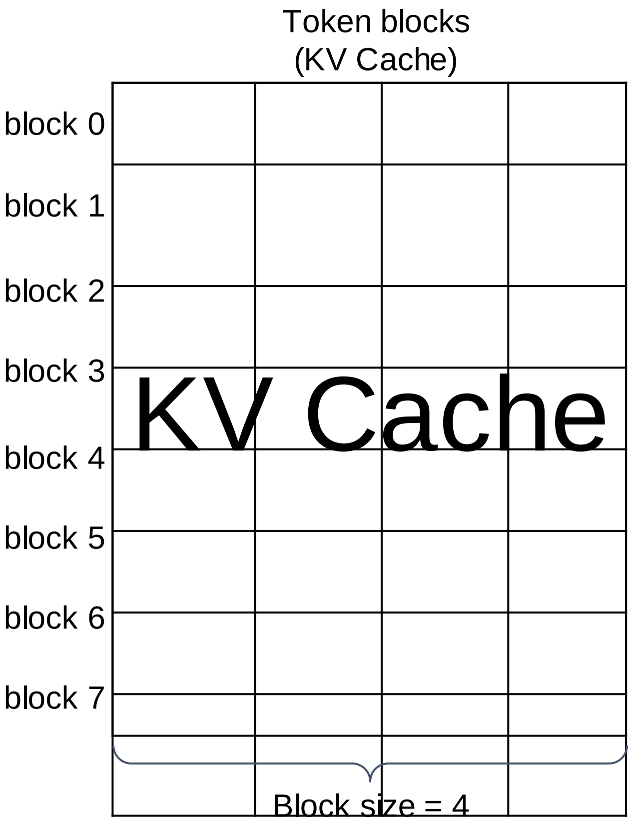
Inspired by **virtual memory** and **paging**





Token block

- A **fixed-size** contiguous chunk of memory that can store token states **from left to right**



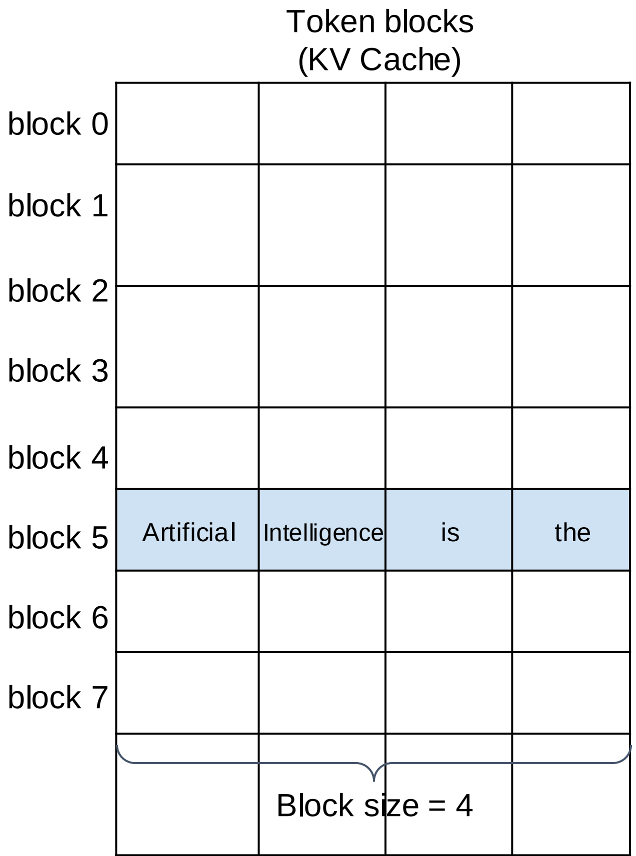
Token block

- A **fixed-size** contiguous chunk of memory that can store token states **from left to right**

Block 4

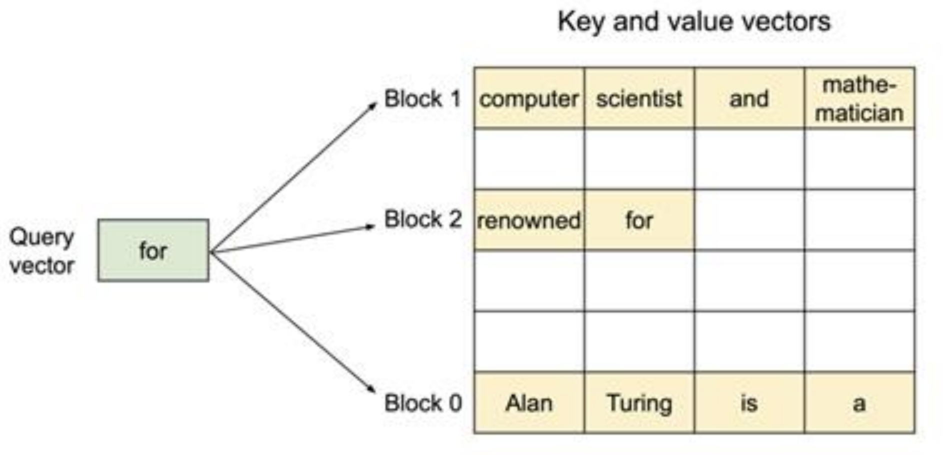
Artificial	-0.2	0.1	-1.1
Intelligence	0.9	0.7	0.2
is	-0.1	-0.3	0.1
the	-1.1	0.5	0.4

820 KB / token
(LLaMA-13B)



Paged Attention

- An attention algorithm that allows for storing continuous keys and values in non-contiguous memory space



Logical & physical token blocks

Request A

Prompt: “Alan Turing is a computer scientist”

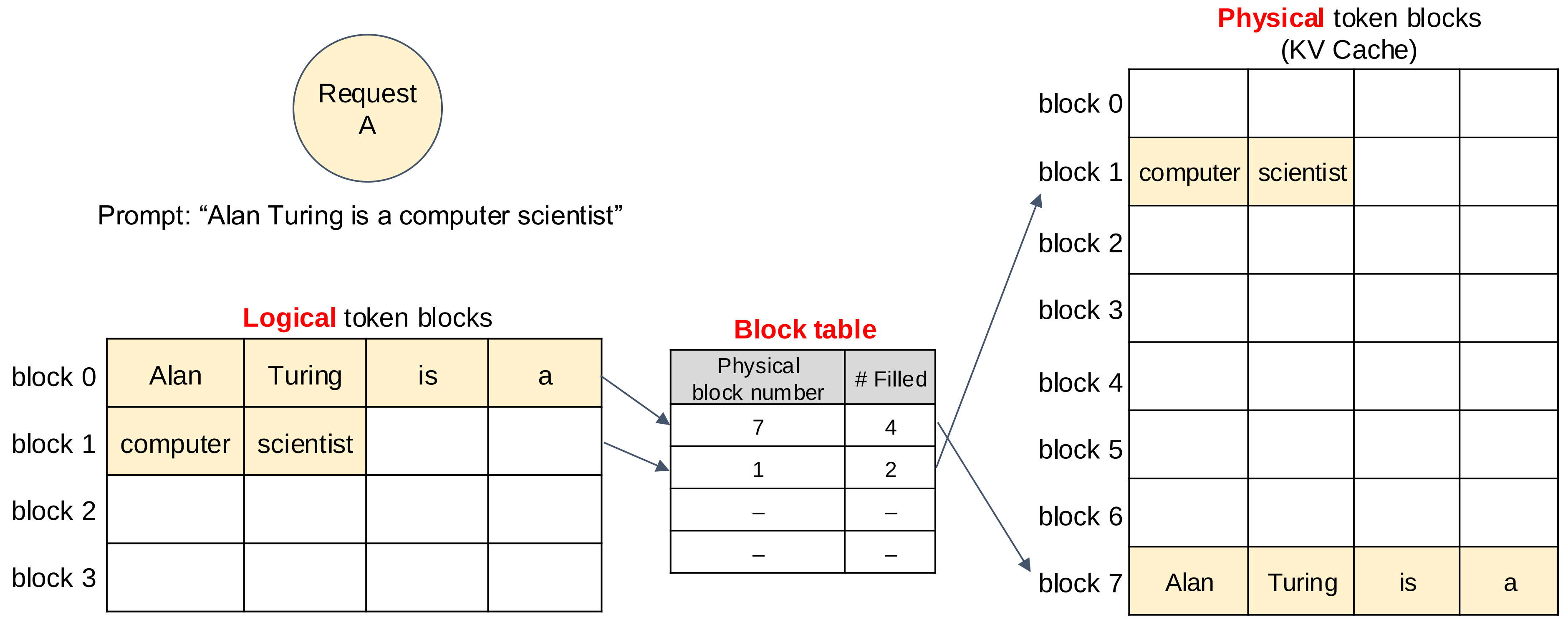
Logical token blocks

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

Physical token blocks
(KV Cache)

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

Logical & physical token blocks



Logical & physical token blocks

Request A

Prompt: “Alan Turing is a computer scientist”
Completion: “<u>and</u>”

Logical token blocks

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

Block table

Physical block number	# Filled
7	4
1	2
–	–
–	–

Physical token blocks
(KV Cache)

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

Logical & physical token blocks

Request A

Prompt: “Alan Turing is a computer scientist”
Completion: “<u>and</u>”

Logical token blocks

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

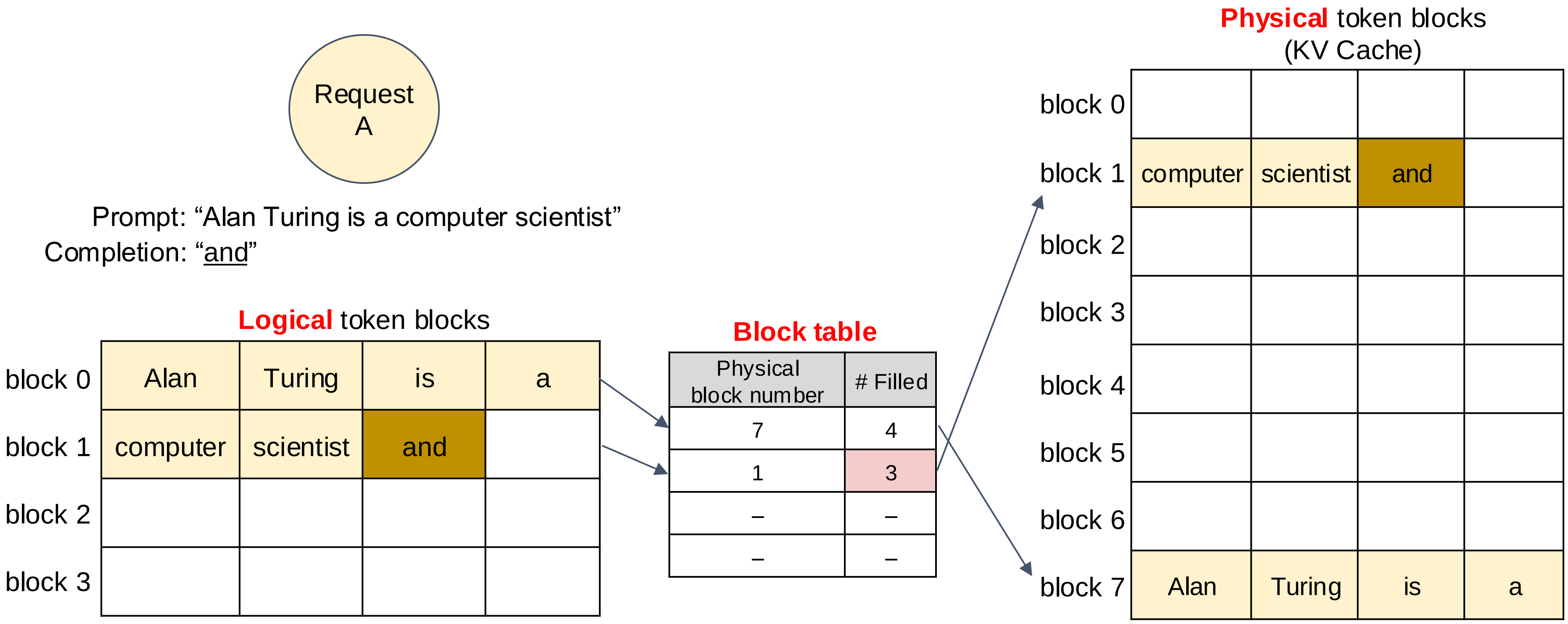
Block table

Physical block number	# Filled
7	4
1	2
–	–
–	–

Physical token blocks
(KV Cache)

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

Logical & physical token blocks



Logical & physical token blocks

Request A

Prompt: “Alan Turing is a computer scientist”
Completion: “and <u>mathematician</u>”

Logical token blocks

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

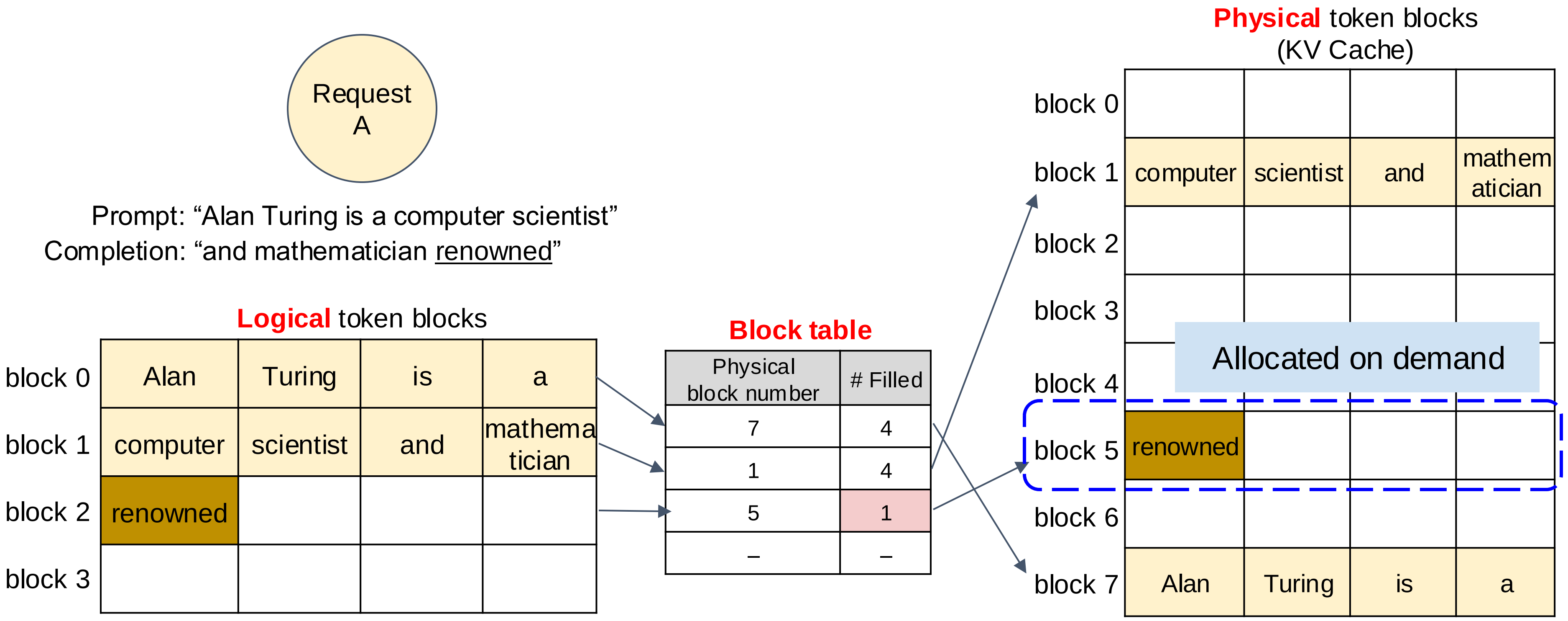
Block table

Physical block number	# Filled
7	4
1	4
–	–
–	–

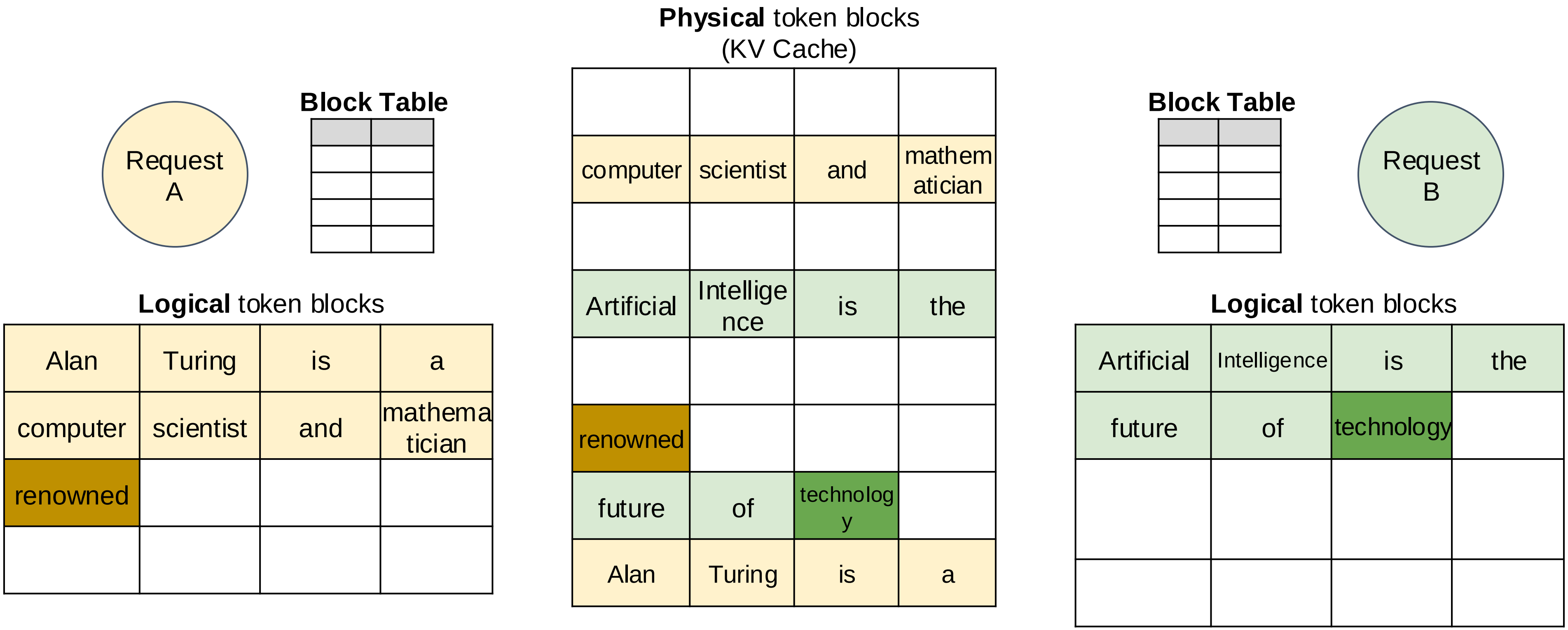
Physical token blocks (KV Cache)

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

Logical & physical token blocks



Serving multiple requests



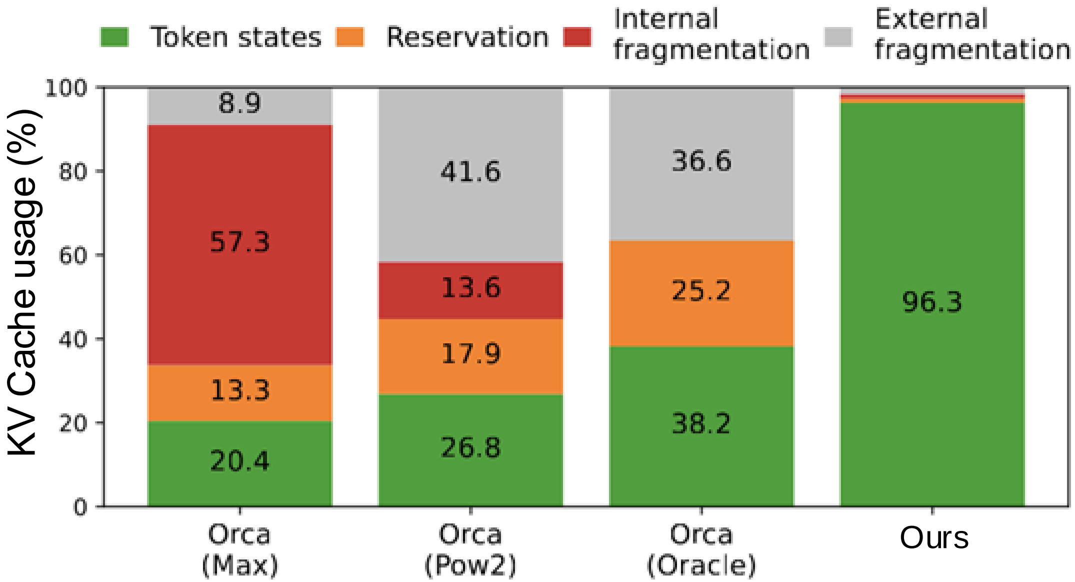
Memory efficiency of vLLM

- 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: 16 or 32 tokens
- No external fragmentation

Alan	Turing	is	a
computer	scientist	and	mathematician
renowned			

Internal fragmentation

Effectiveness of PagedAttention



96.3% KV cache utilization

large b

Large Language Models

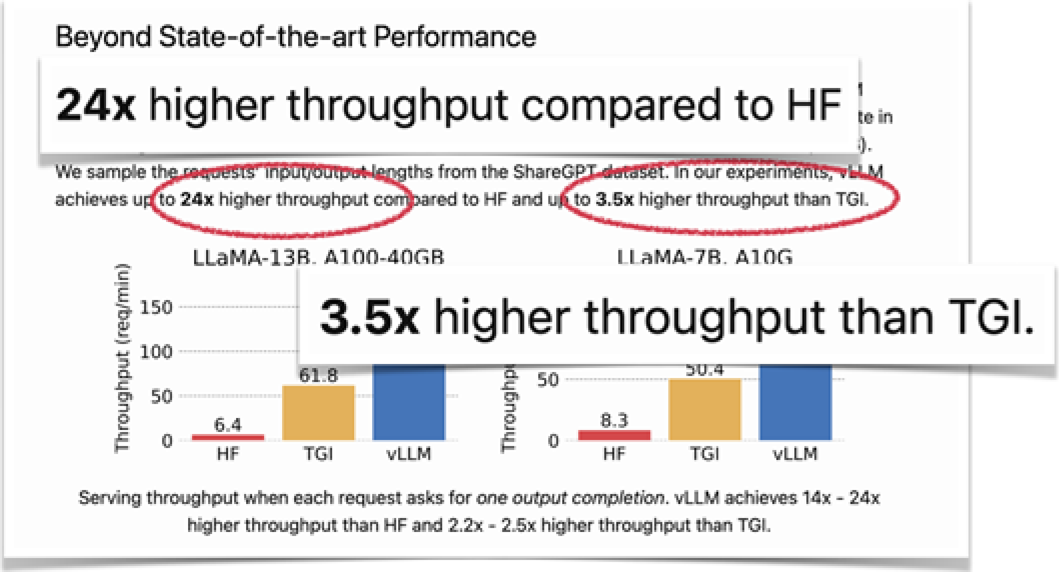
- Transformers, Attentions
- Scaling Law
  - MoE
- Connecting the dots: Training Optimizations
  - Flash attention ← come back to this later next week
- Serving and inference optimization
  - **Continuous batching and Paged attention**
  - Speculative decoding (Guest Lecture)
- Connecting the dots: Deepseek-v3
- Hot topics
  - Prefill-decode disaggregation

LLM System Today Optimize **Throughput**

vLLM

DeepSpeed MII

NVIDIA.



Motivation: Applications have Diverse SLO

- **TTFT**

Time to first token

Initial response time

- **TPOT**

Time per output token

Average time between two subsequent generated tokens

Chatbot

Fast initial response

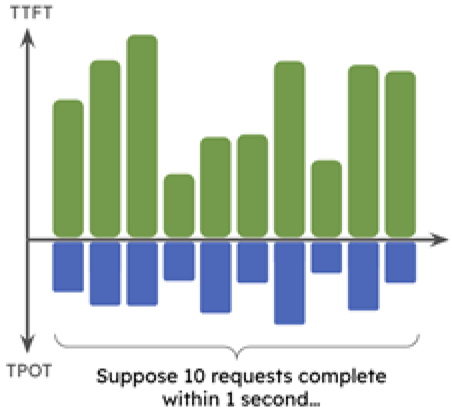
Human reading speed (P99 latency = 250ms)

Summarization

User can tolerate longer initial response

Data output generation (P99 latency = 35ms)

High Throughput ≠ High Goodput



Throughput = 10 rps

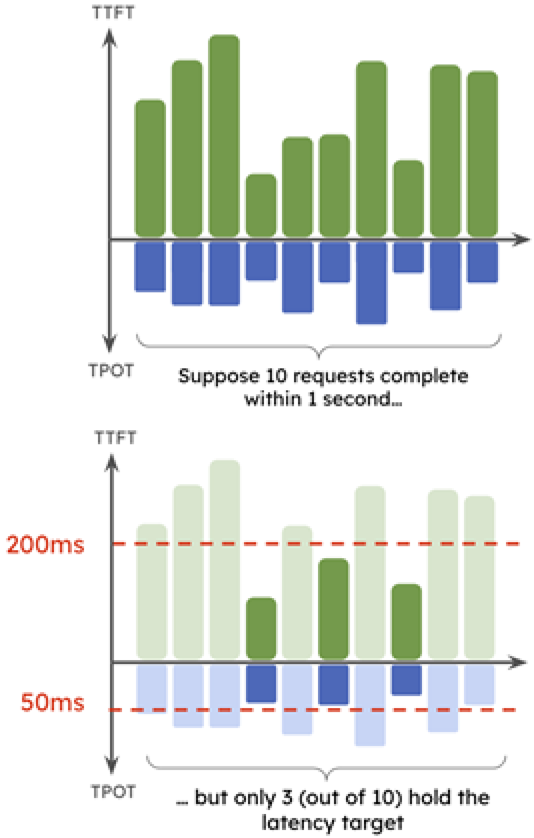
= completed request / time

High Throughput

System

...

High Throughput ≠ High Goodput



Throughput = 10 rps
= completed request / time

under SLO
criteria

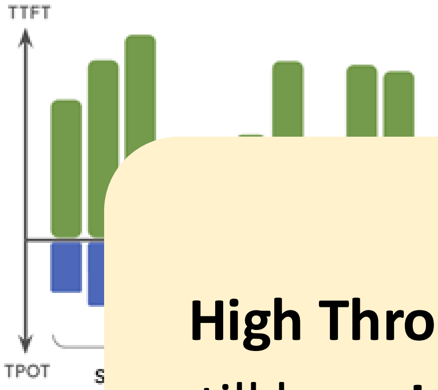
Goodput = 3 rps
= completed request **within SLO** / time

High Throughput
System

...

can have
Low Goodput!

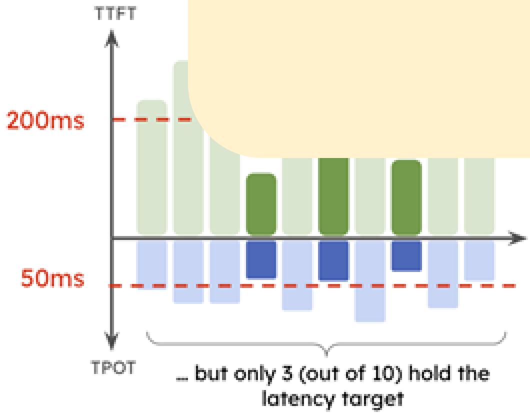
High Throughput ≠ High Goodput



High Throughput

High Throughput can still have **Low Goodput**

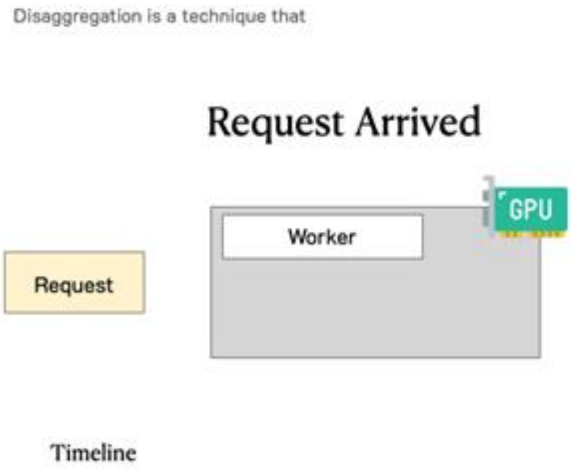
⇒ Poor UX 💔



Goodput = 3 rps

= completed request **within SLO** / time

Background: Continuous Batching



Prefill and Decode have Distinct Characteristics

- **Prefill**

Compute-bound

One prefill saturates compute.

- **Decode**

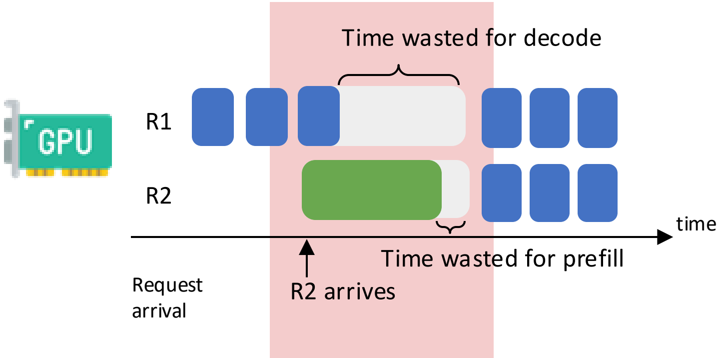
Memory-bound

Must batch a lot of requests together to saturate compute

Continuous Batching Cause Interference

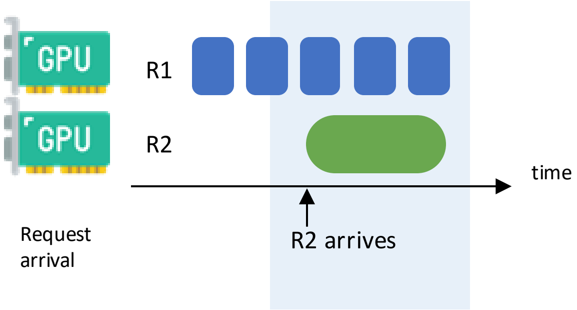
Continuous Batching

Batch R1 and R2 together in 1 GPU



Separate prefill / decode

R1 and R2 in separate GPUs

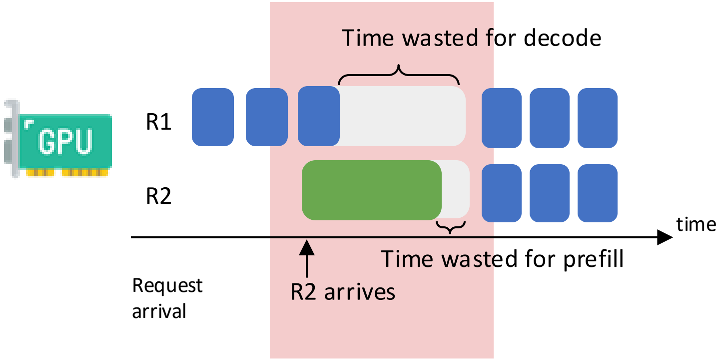


No Interference

Continuous Batching Cause Interference

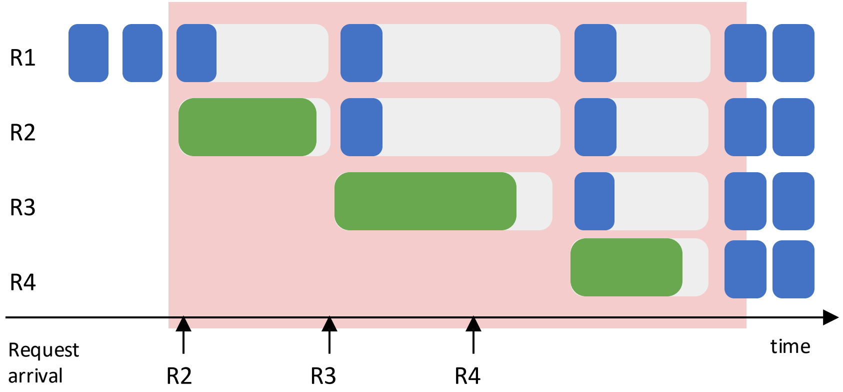
Continuous Batching

Batch R1 and R2 together in 1 GPU



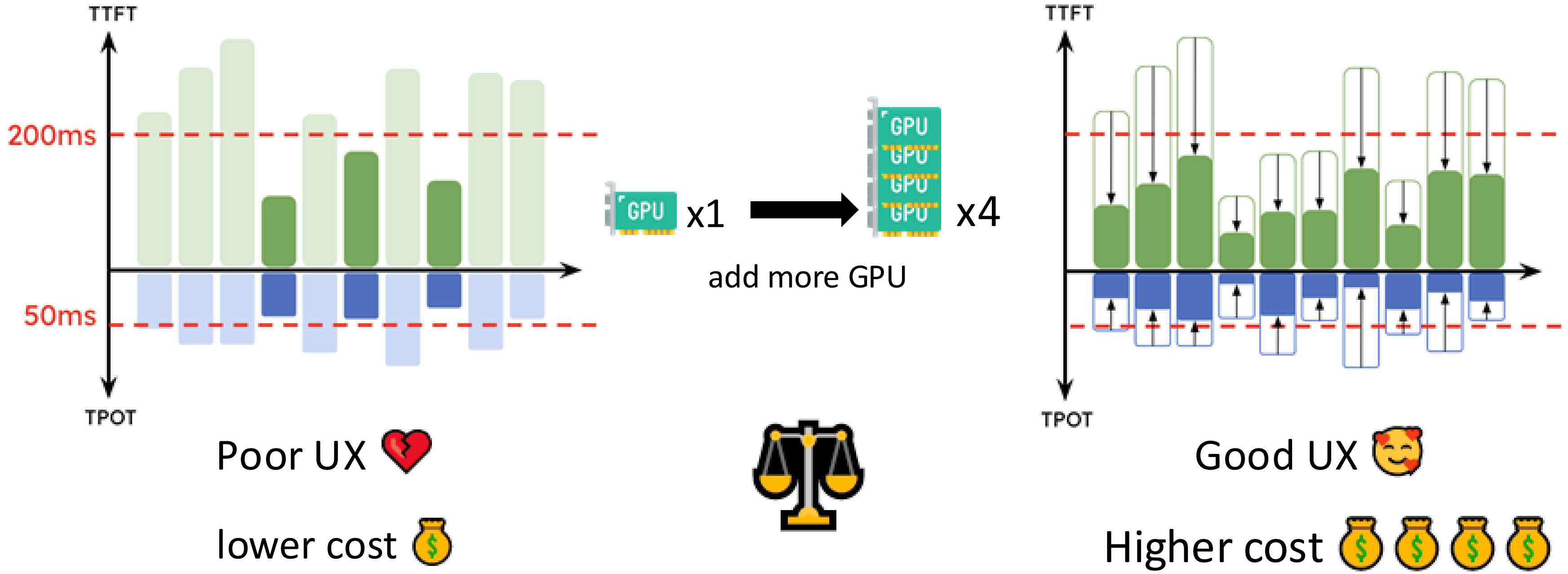
Continuous Batching

Batch R1~R4 together in 1 GPU

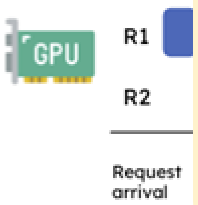


wasted time

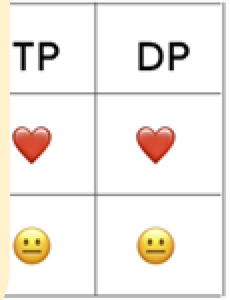
Colocation → Overprovision Resource to meet SLO



Summary: Problems caused by Colocation



Is there a better way to achieve better **Goodput per GPU**?

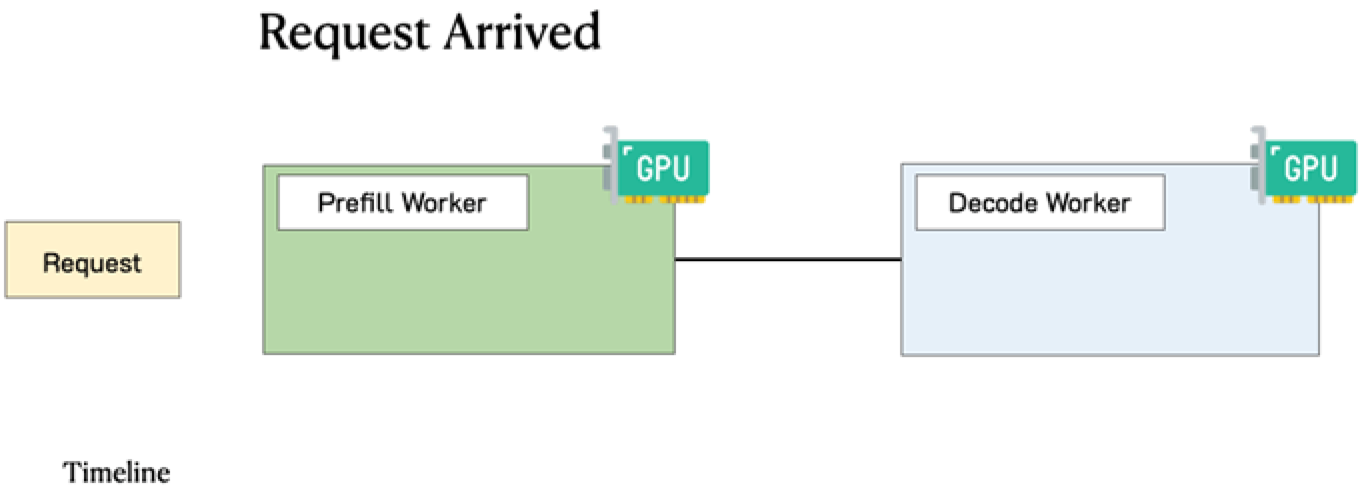


Continuous Batching Cause Interference

Coupled Parallelism Strategy

Disaggregating Prefill and Decode

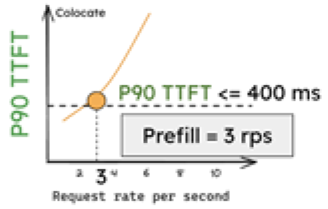
Disaggregation is a technique that

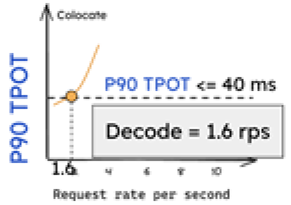


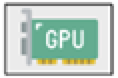
Disaggregation achieves better goodput

Colocate

1 GPU for both Prefill and Decode







Max System goodput

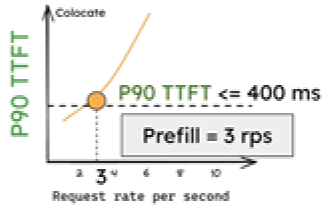
= Min(Prefill, Decode)

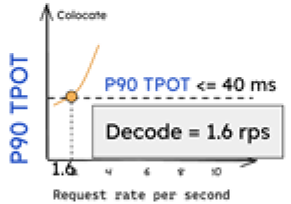
= 1.6 rps / GPU

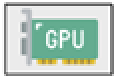
Disaggregation achieves better goodput

Colocate

1 GPU for both Prefill and Decode







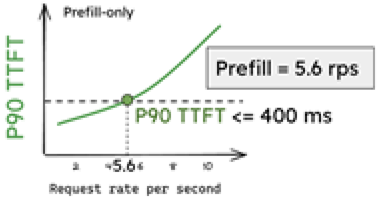
Max System goodput

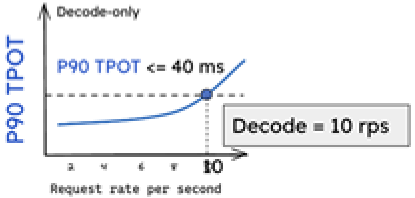
= Min(Prefill, Decode)

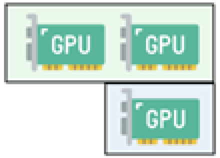
= 1.6 rps / GPU

Disaggregate (2P1D)

2 GPU for Prefill + 1 GPU for Decode







Disaggregate (2P1D) goodput

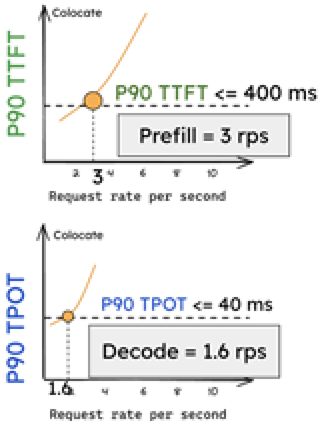
= Min (5.6 x **2**, 10) rps / 3 GPU

= 3.3 rps / GPU

Disaggregation achieves better goodput

Colocate

1 GPU for both Prefill and Decode



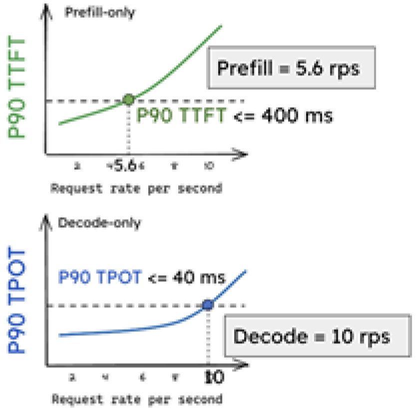
Max System goodput

= Min(Prefill, Decode)

= 1.6 rps / GPU

Disaggregate (2P1D)

2 GPU for Prefill + 1 GPU for Decode



Simple Disaggregation achieves **2x** goodput (per GPU)

Disaggregate (2P1D) goodput

= Min (5.6 x **2**, 10) rps / 3 GPU

= 3.3 rps / GPU

Disaggregation

- Published in 2024 at UCSD (yes, Hao’s lab)
- Soon become the default architecture replacing continuous batching at large scale
- Deepseek v3 uses prefill-decode disaggregation combined with different parallelisms.