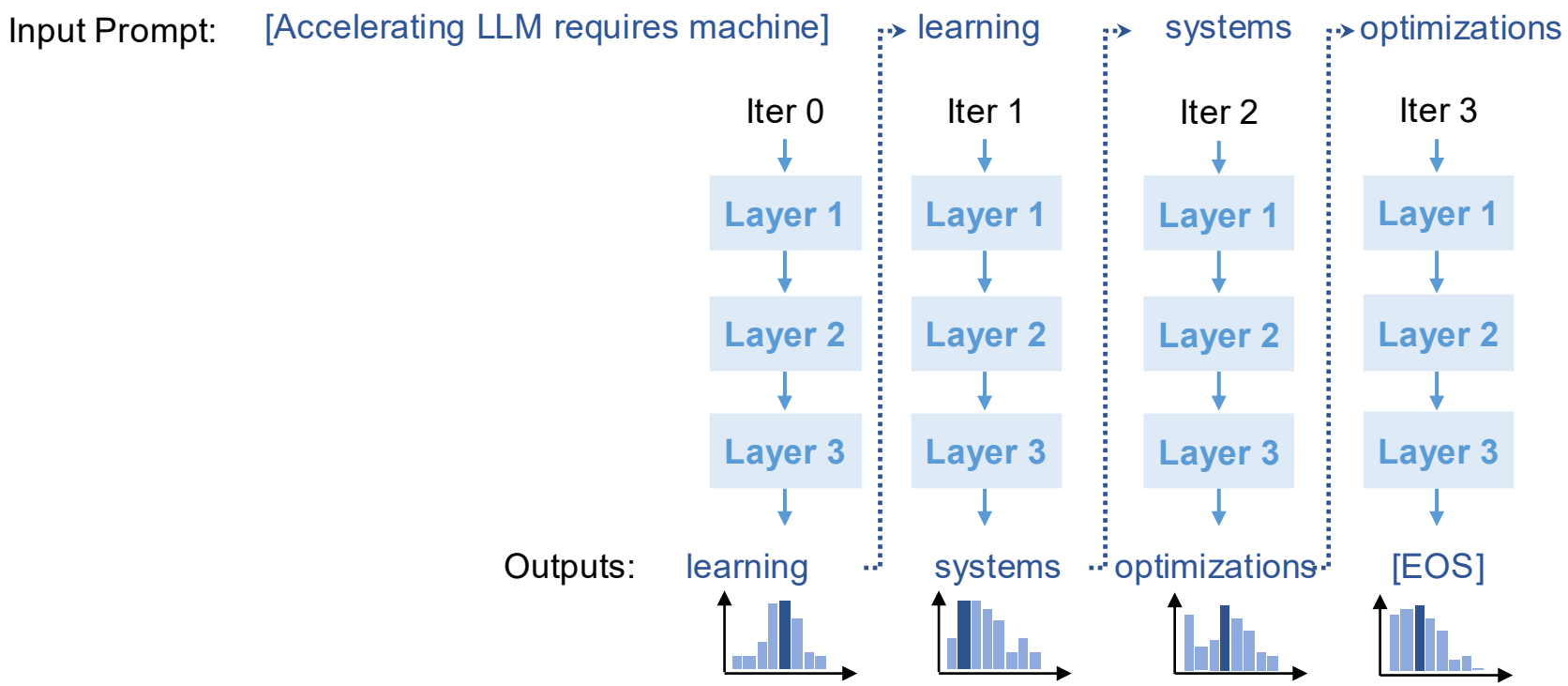
15-442/15-642: Machine Learning Systems

LLM Serving Techniques Part 1: Continuous Batching, PagedAttention, RadixAttention

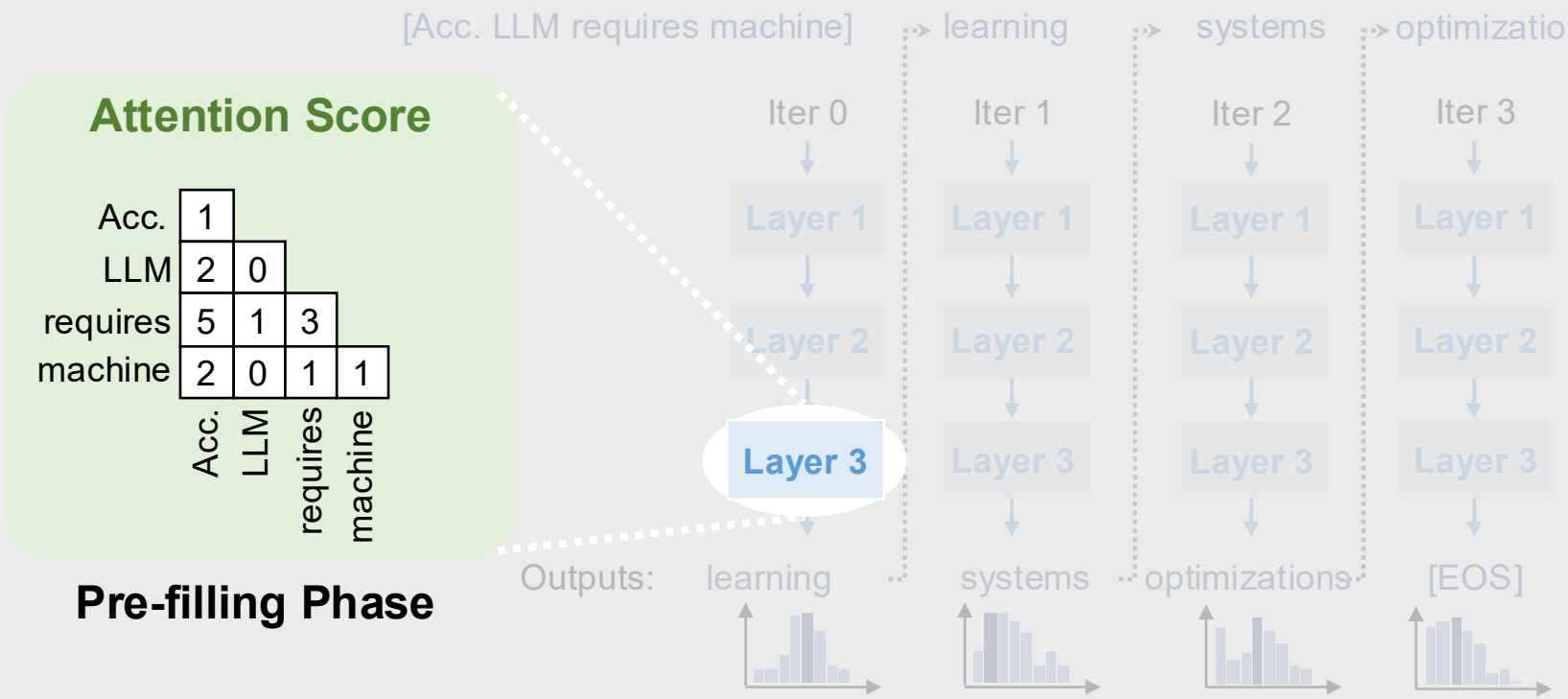
Tianqi Chen and Zhihao Jia

Carnegie Mellon University

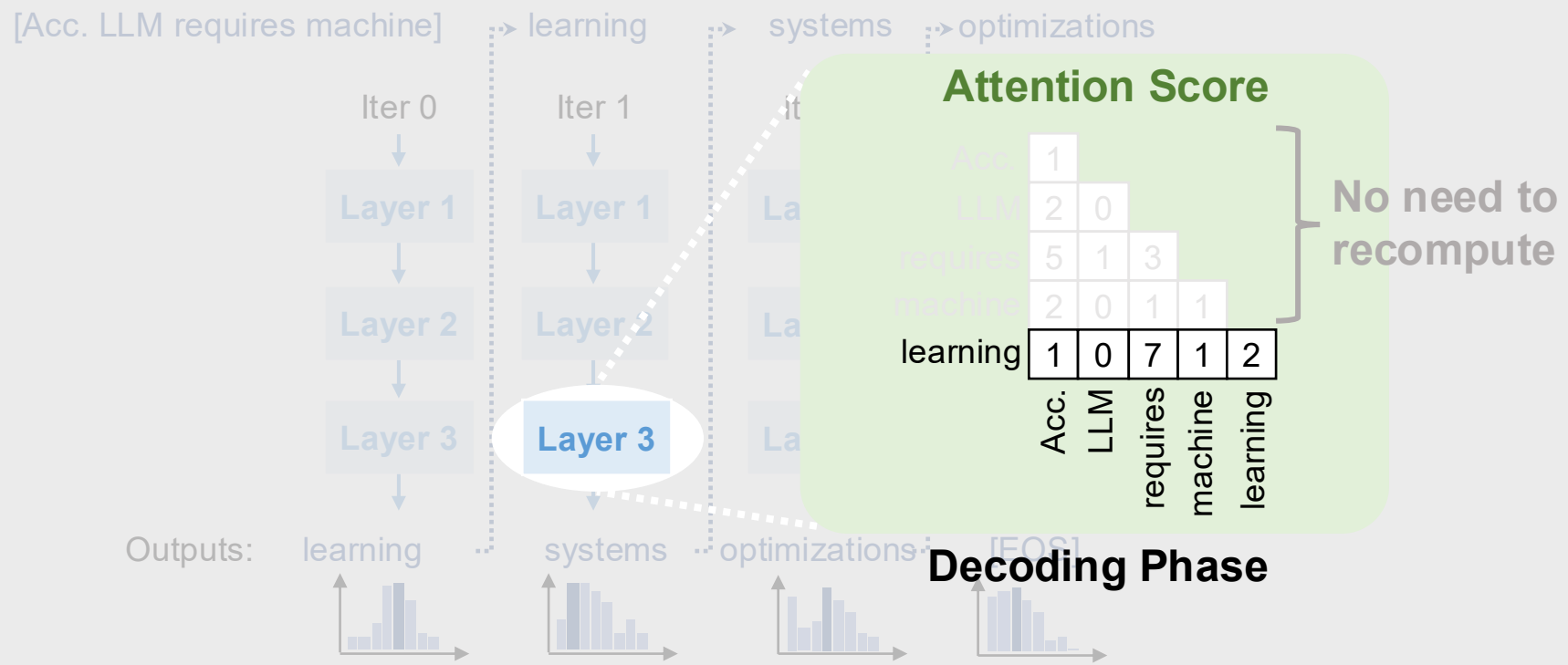
Generative LLM Inference: Autoregressive Decoding



Generative LLM Inference: Autoregressive Decoding



Generative LLM Inference: Autoregressive Decoding

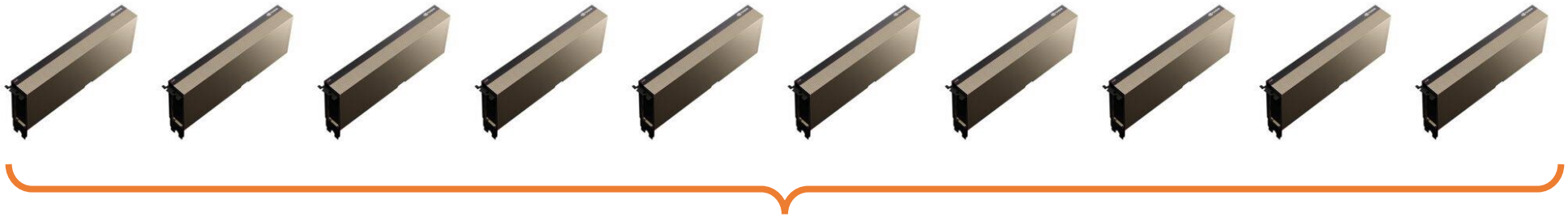


Generative LLM Inference: Autoregressive Decoding

- **Pre-filling phase** (first iteration):
  - Process ***all*** input tokens at once
- **Decoding phase** (all other iterations):
  - Process a ***single*** token generated from previous iteration
  - Use attention keys & values of all previous tokens

- Key-value cache:
  - Save attention keys and values for the following iterations to avoid recomputation

LLMs are Slow and Expensive to Serve

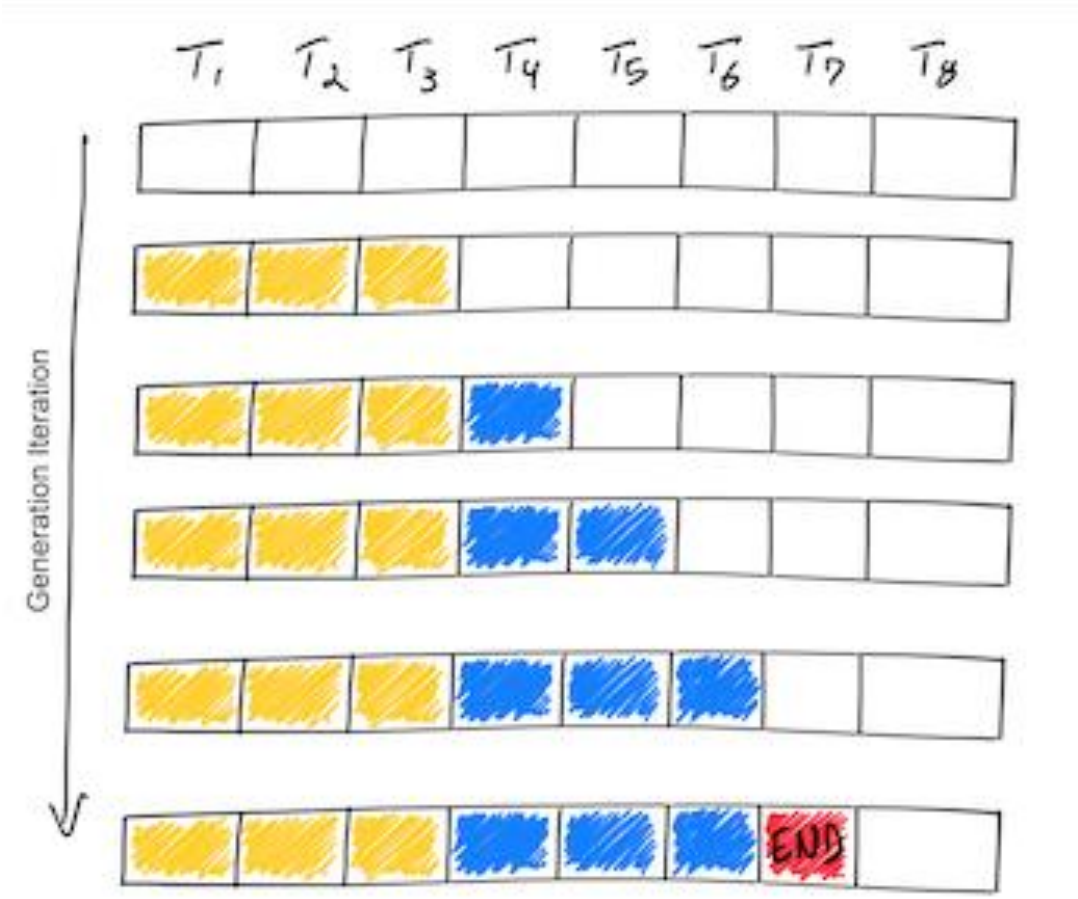


- **At least ten** A100-40GB GPUs to serve 175B GPT-3 in half precision
- Generating 256 tokens takes **~20 seconds**
- Cannot process many requests in parallel
  - Per-request key/value cache takes **3GB GPU memory**

Today's Lecture: LLMs Serving Techniques

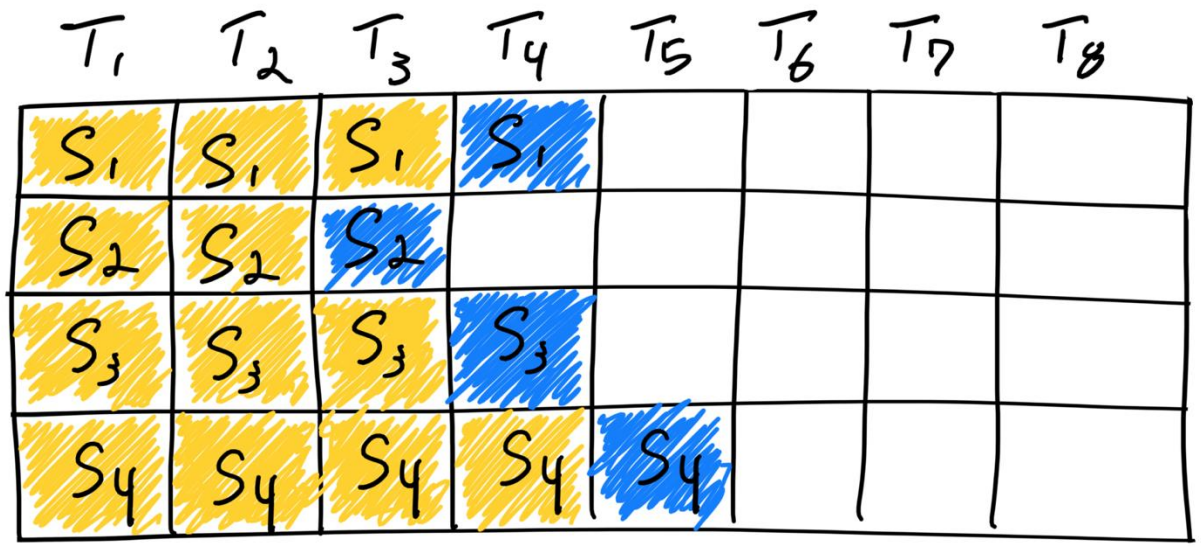
- **Continuous Batching**
- PagedAttention
- RadixAttention

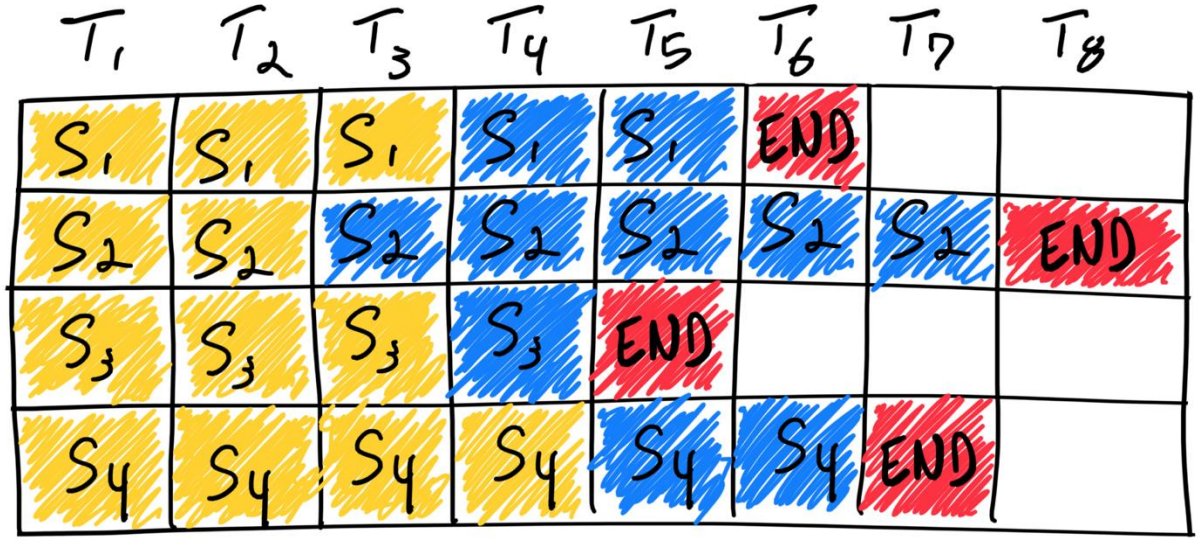
LLM Decoding Timeline



Prefill tokens **Decode tokens**

Batching Requests to Improve GPU Performance

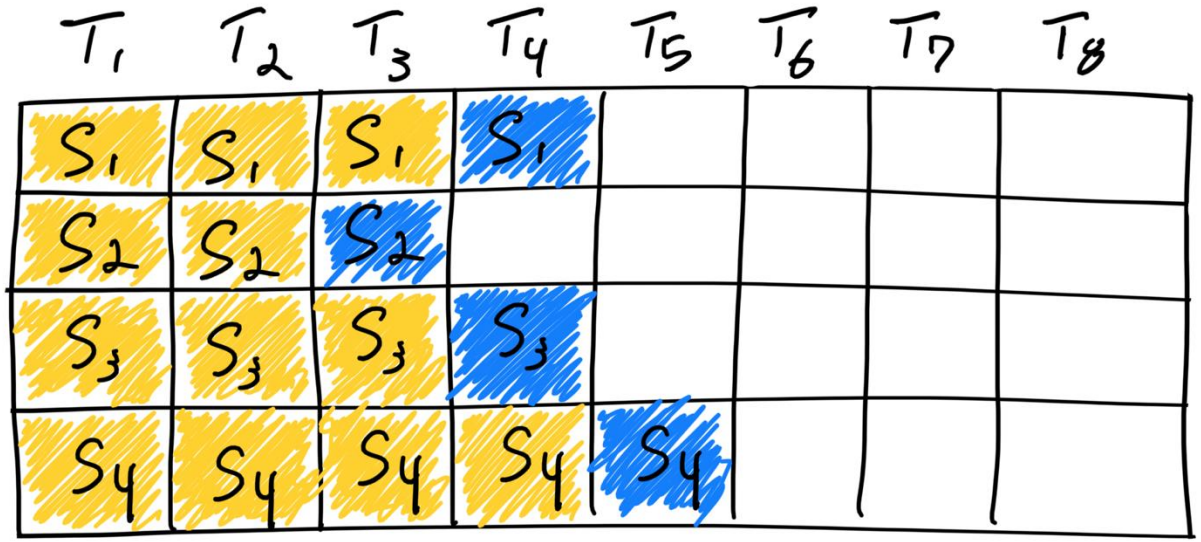


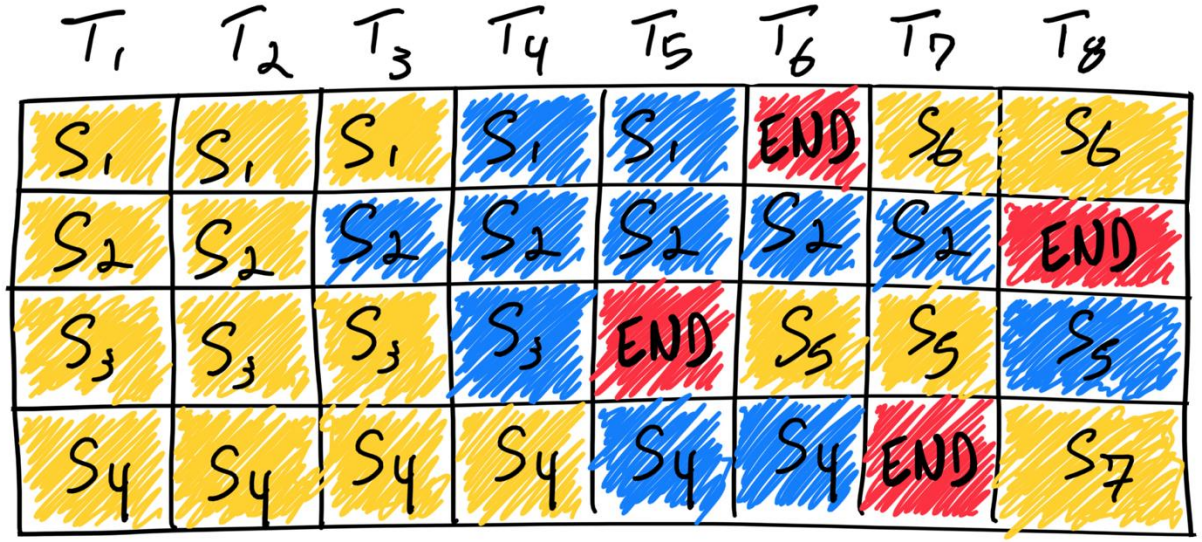


Issues with static batching:

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

Continuous Batching



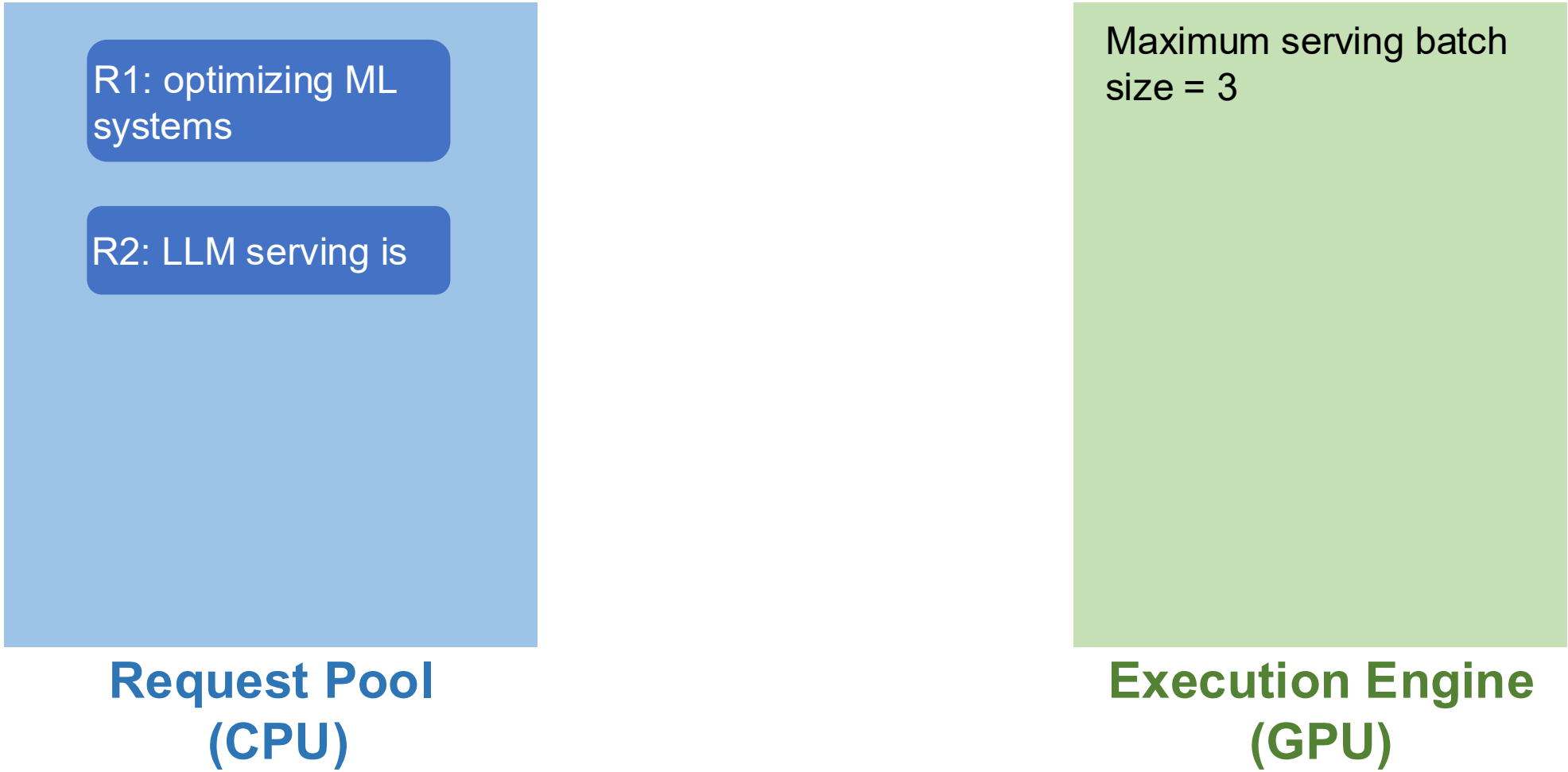


Benefits:

- Higher GPU utilization
- New requests can start immediately

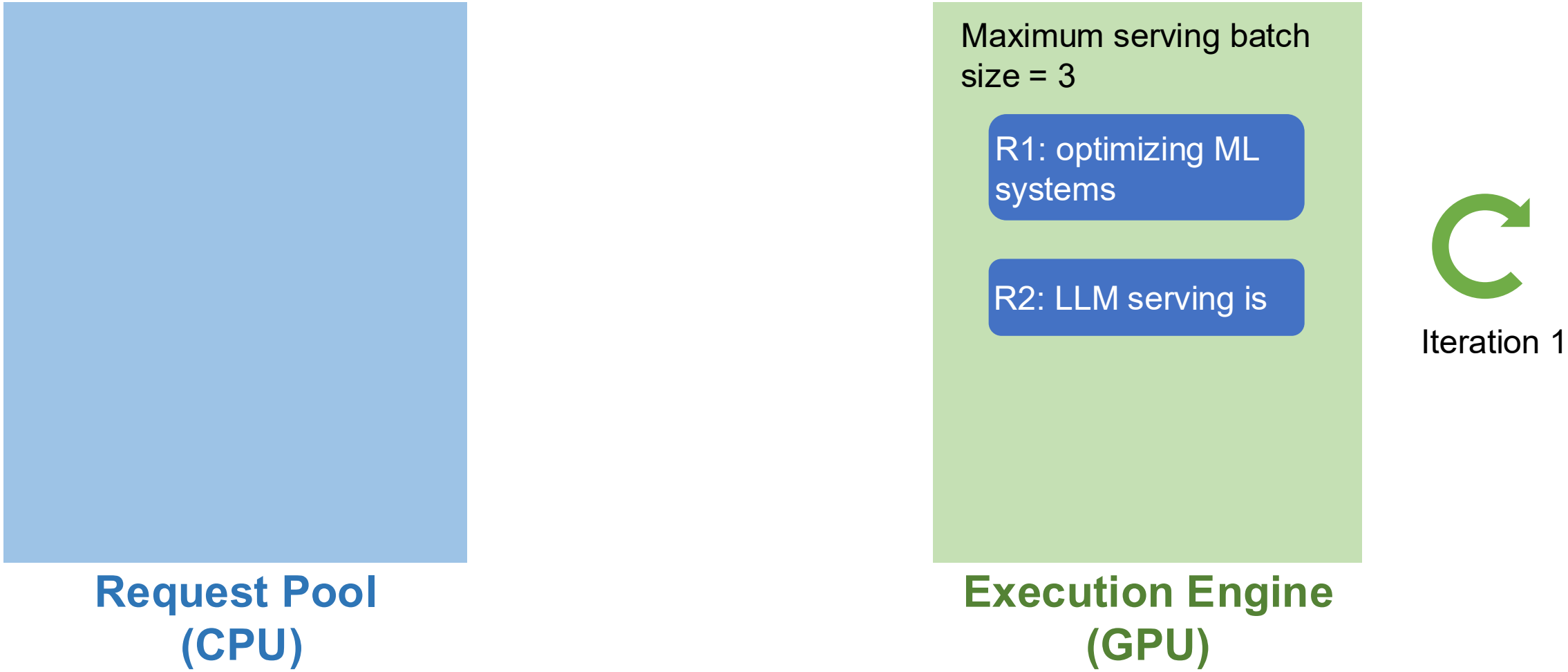
Continuous Batching Step-by-Step

- Receives two new requests R1 and R2



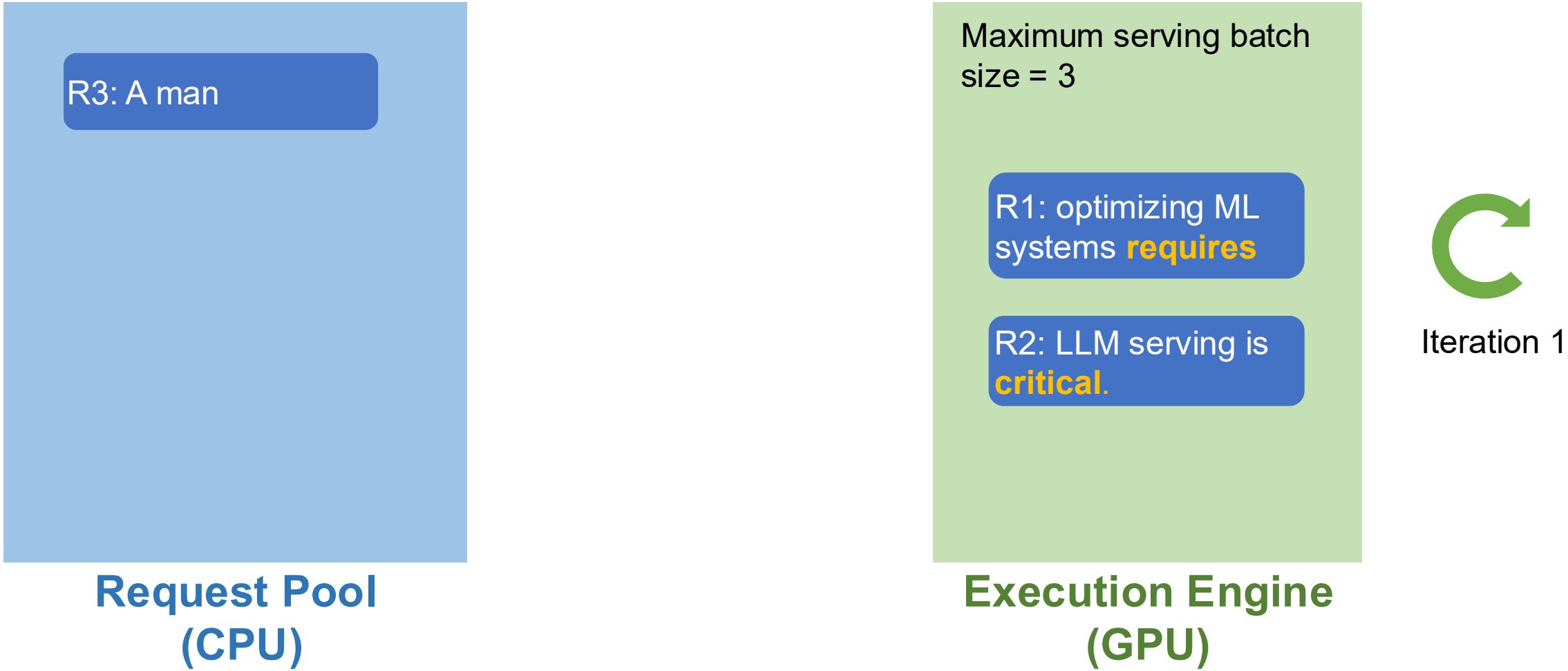
Continuous Batching Step-by-Step

- Iteration 1: decode R1 and R2



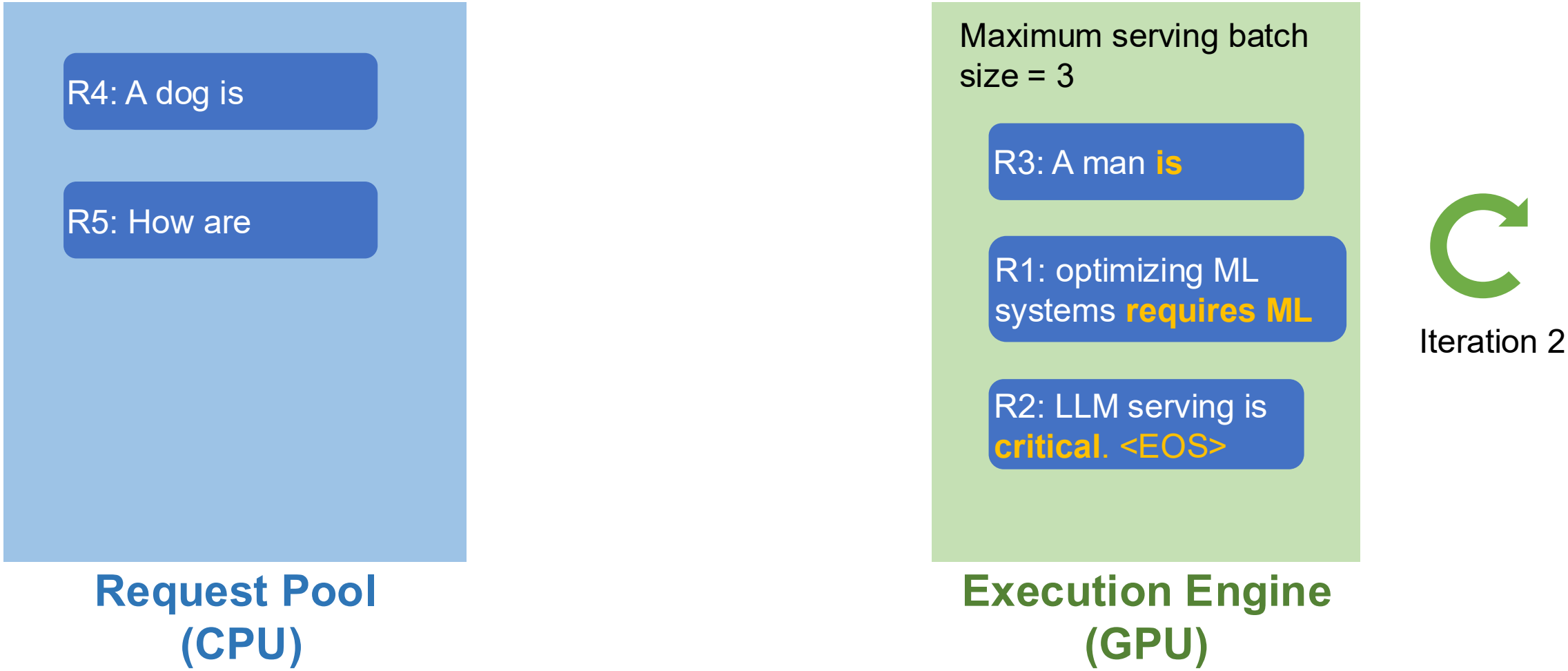
Continuous Batching Step-by-Step

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



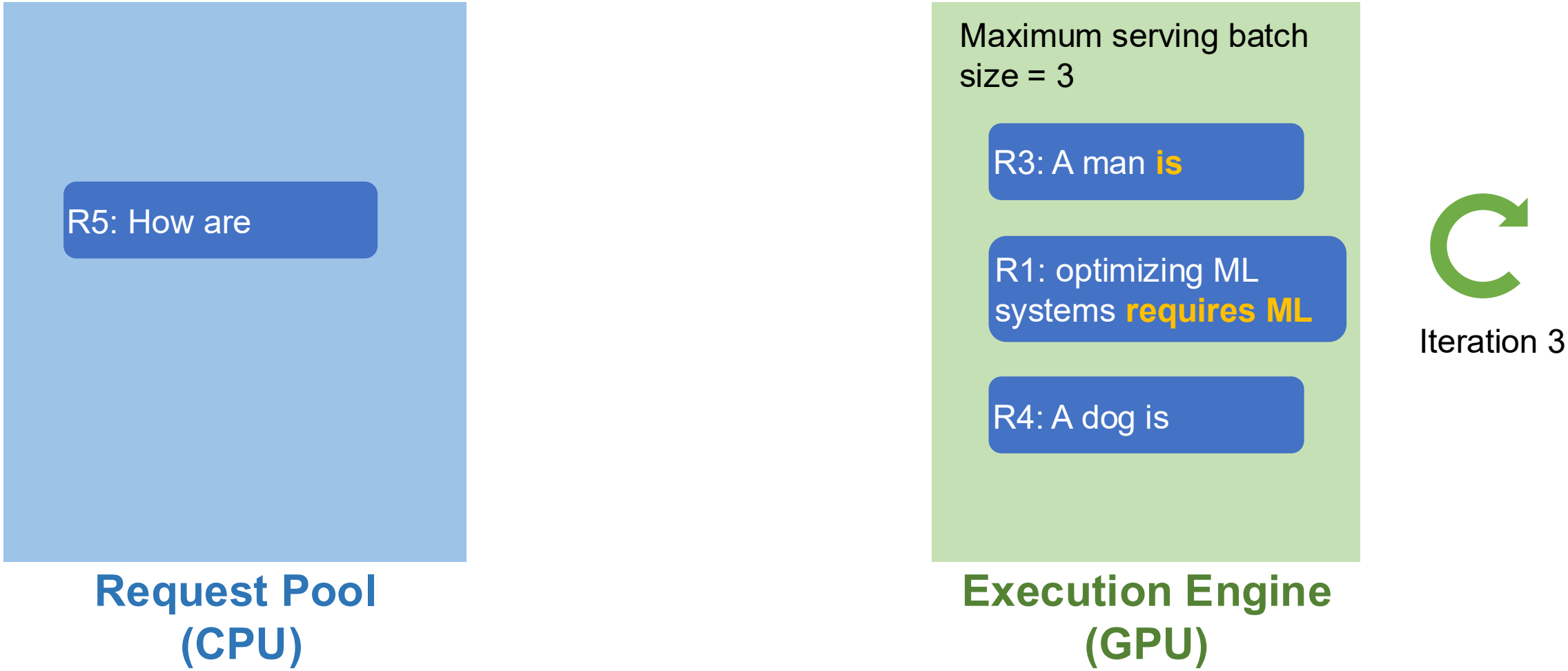
Continuous Batching Step-by-Step

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



Continuous Batching Step-by-Step

- Iteration 3: decode R1, R3, R4



Continuous Batching

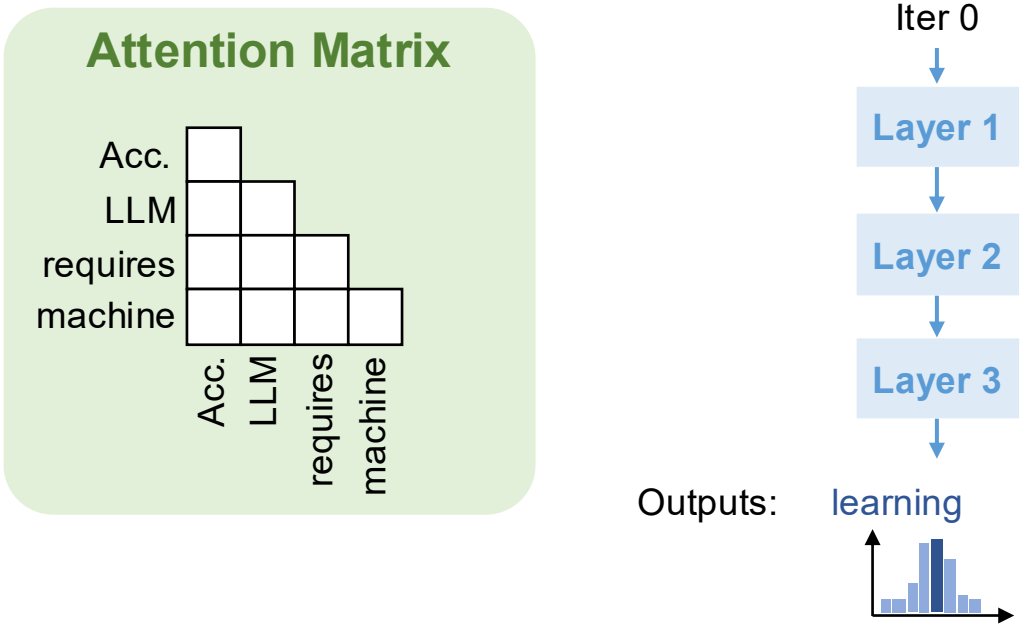
- Handle early-finished and late-arrived requests more efficiently
- Higher GPU utilization

Outline: LLMs Serving Techniques

- Continuous Batching
- **PagedAttention**
- RadixAttention

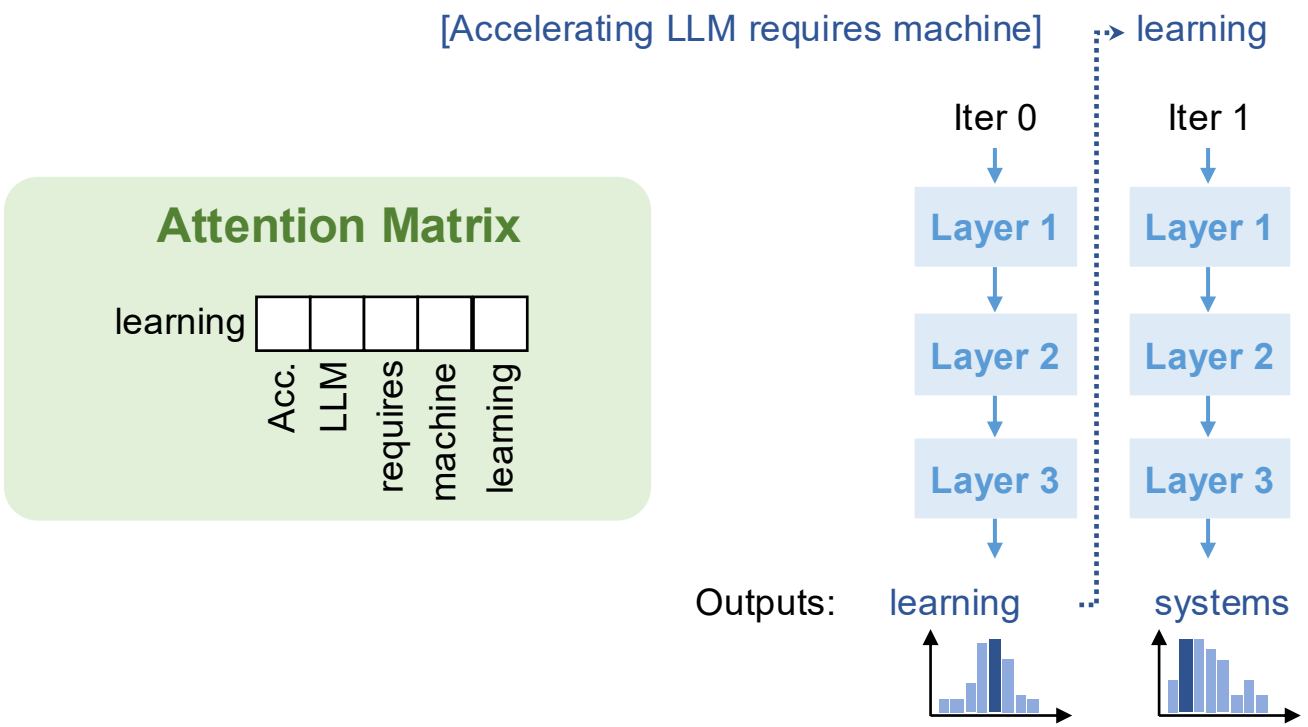
KV Cache Dynamically Grows and Shrinks

[Accelerating LLM requires machine]

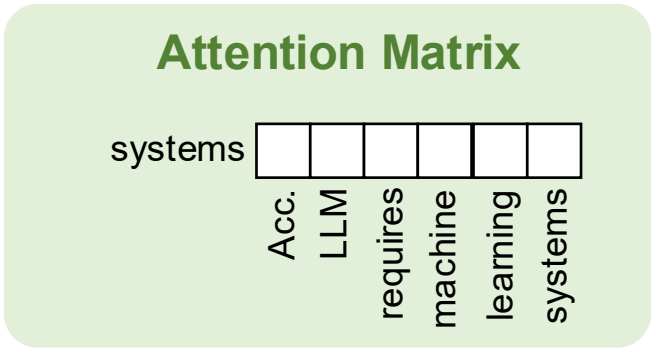


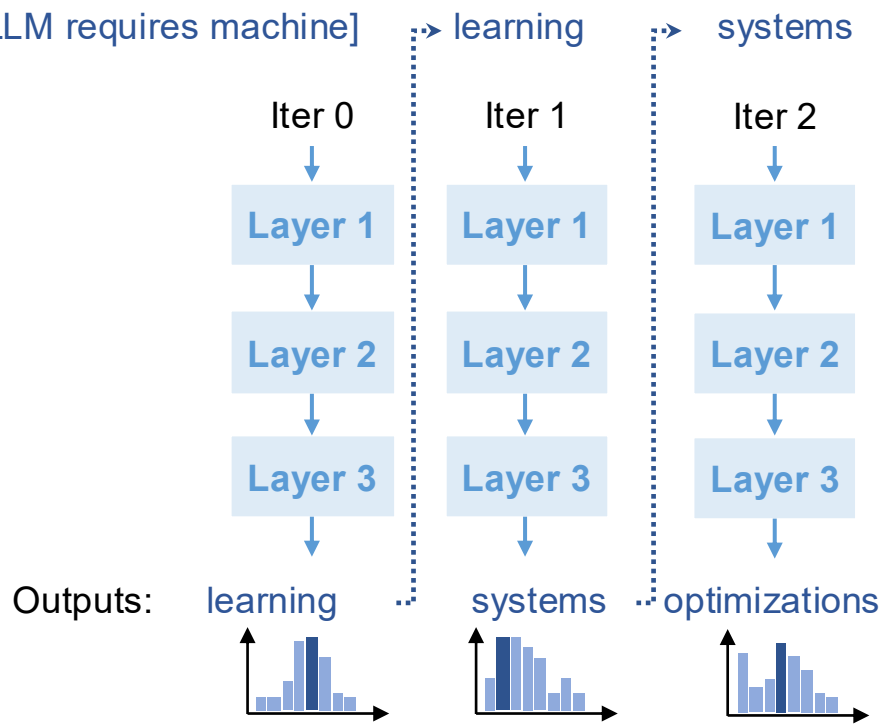
KV Cache

KV Cache Dynamically Grows and Shrinks

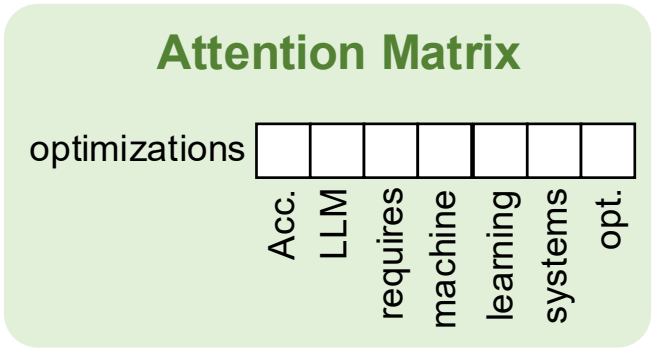


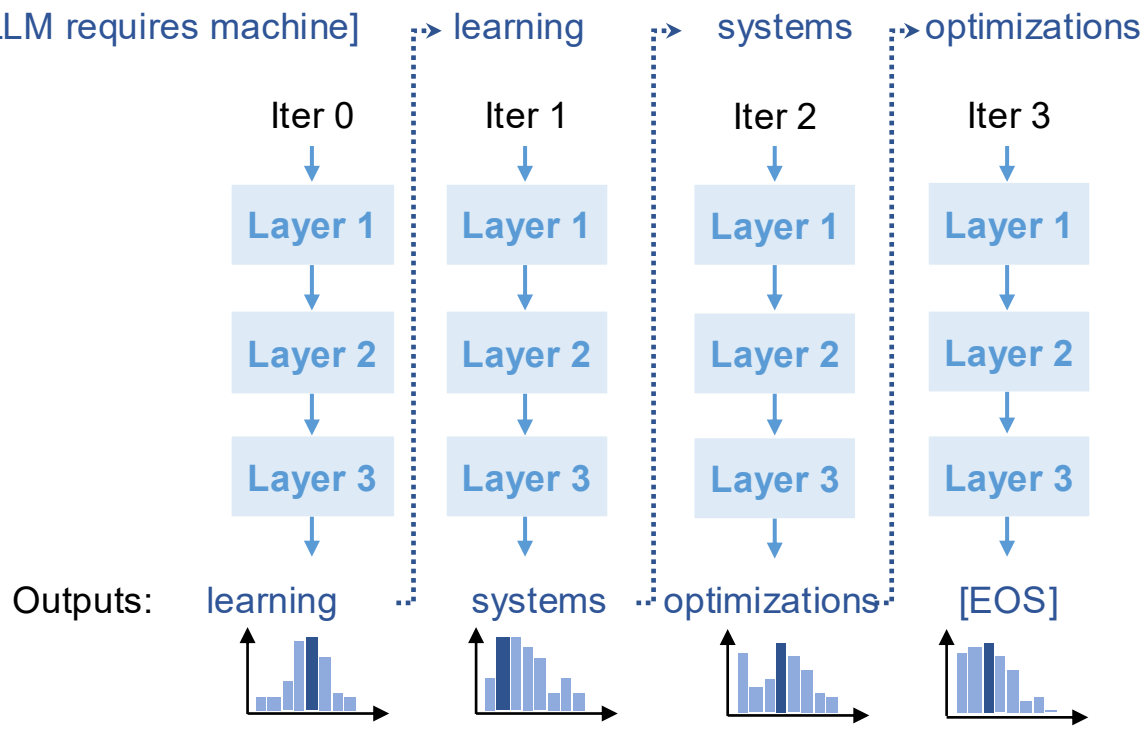
KV Cache Dynamically Grows and Shrinks



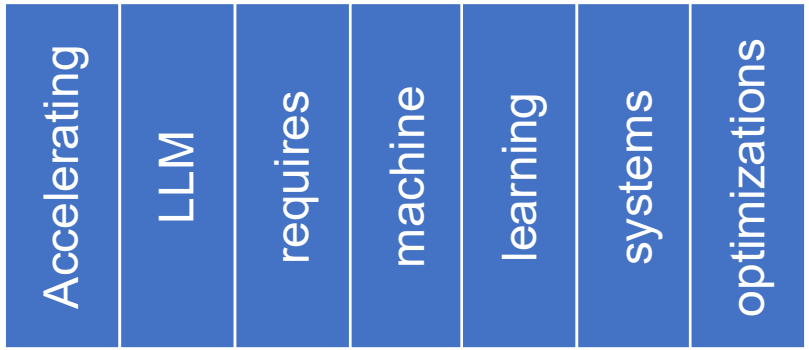


KV Cache Dynamically Grows and Shrinks

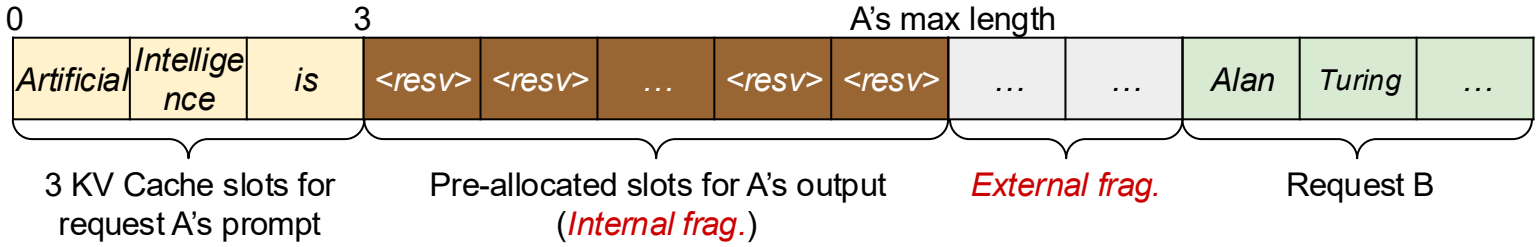




KV Cache



Static KV Cache Management Wastes Memory



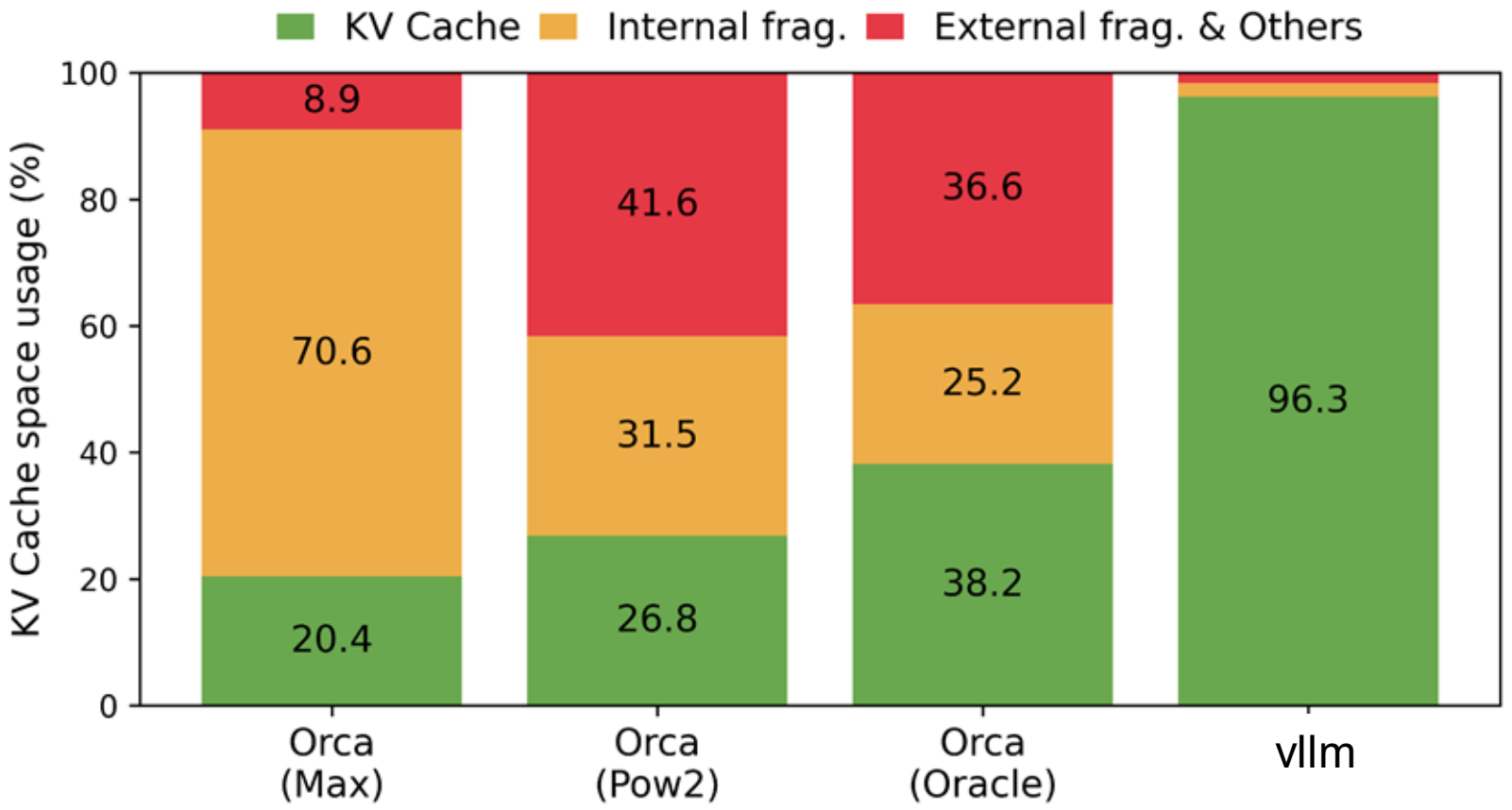
- **Pre-allocates contiguous** space of memory to the request's maximum length

- Memory fragmentation
  - **Internal fragmentation** due to unknown output length
  - **External fragmentation** due to non-uniform per-request max lengths

slides from vllm: Efficient Memory Management for Large Language Model Serving with PagedAttention

Significant Memory Waste in KV Cache

- Only 20-40% of KV cache is utilized to store actual token states

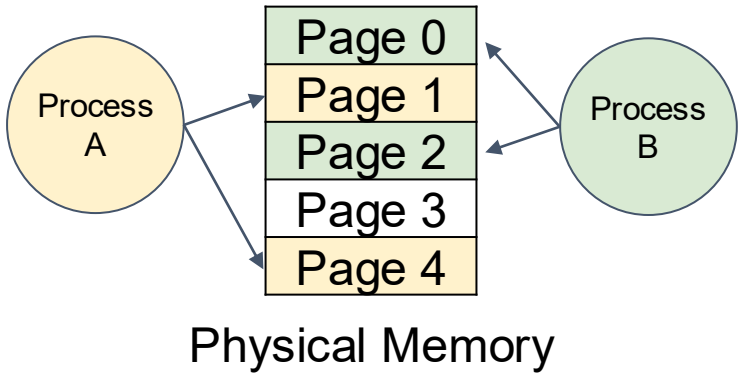


slides from vllm: Efficient Memory Management for Large Language Model Serving with PagedAttention

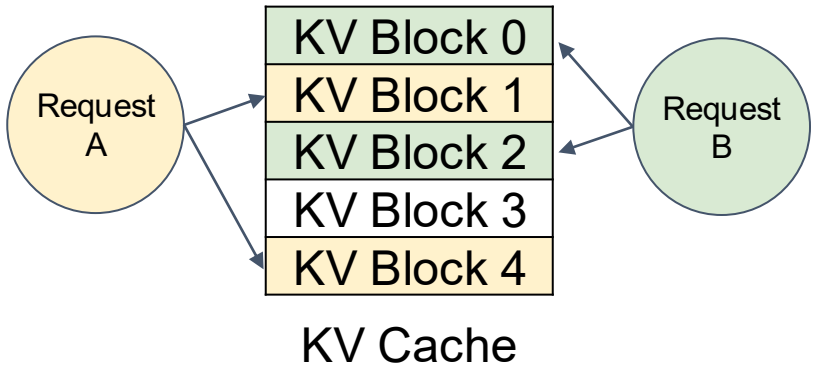
PagedAttention

- Application-level memory paging and virtualization for KV cache

Memory management in OS



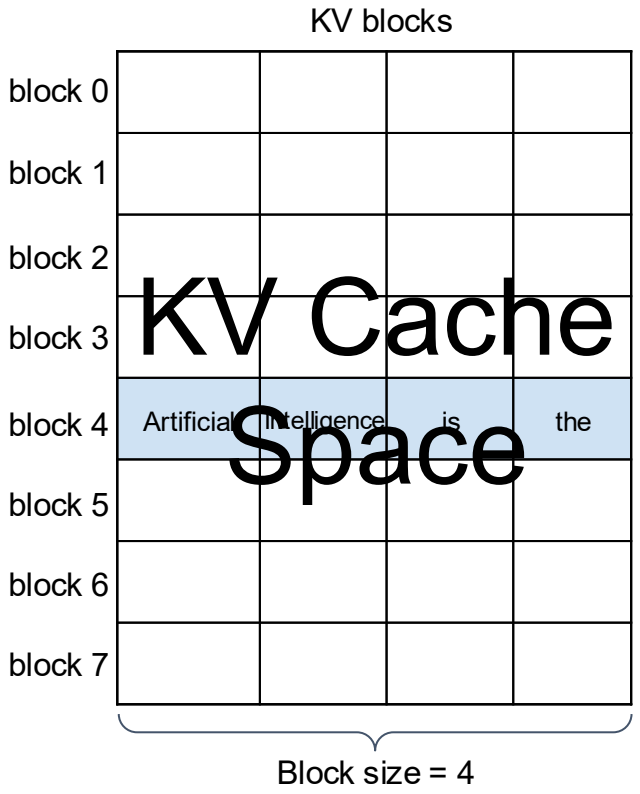
PagedAttention



slides from vllm: Efficient Memory Management for Large Language Model Serving with PagedAttention

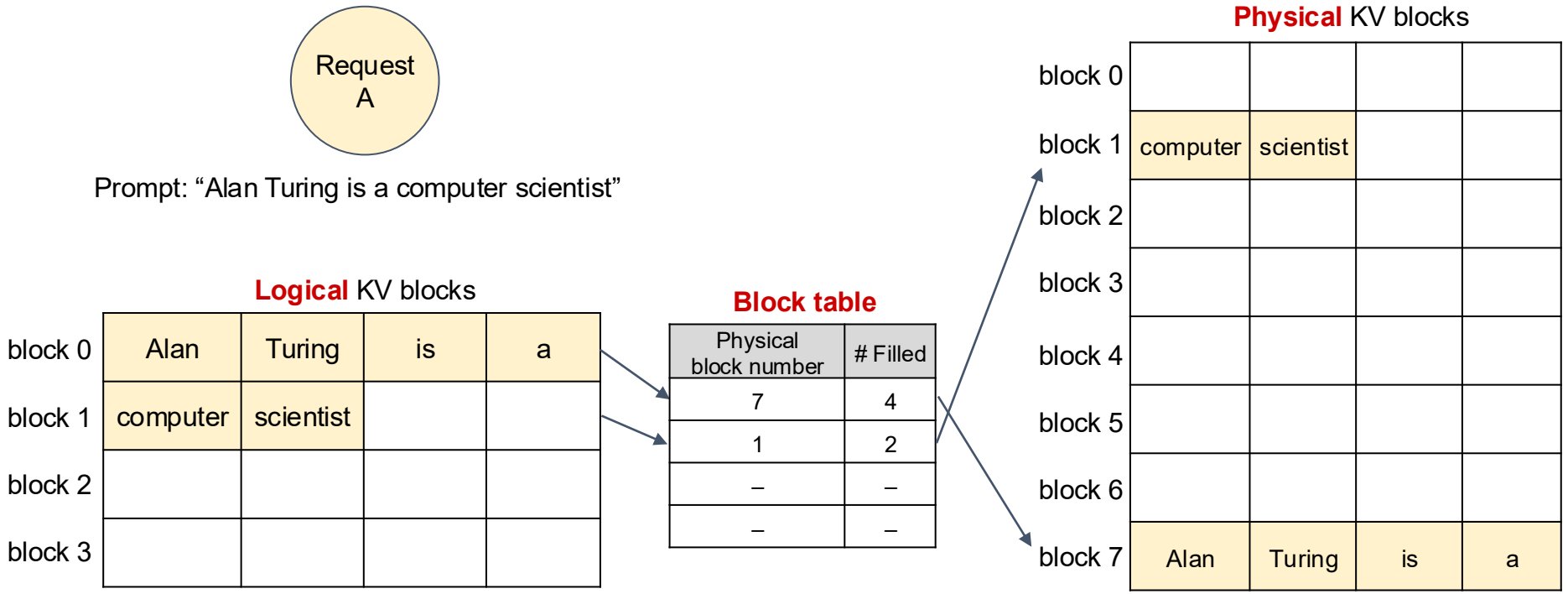
Paging KV Cache Space into KV Blocks*

- KV block is a **fixed-size** contiguous chunk of memory that stores KV states from **left to right**



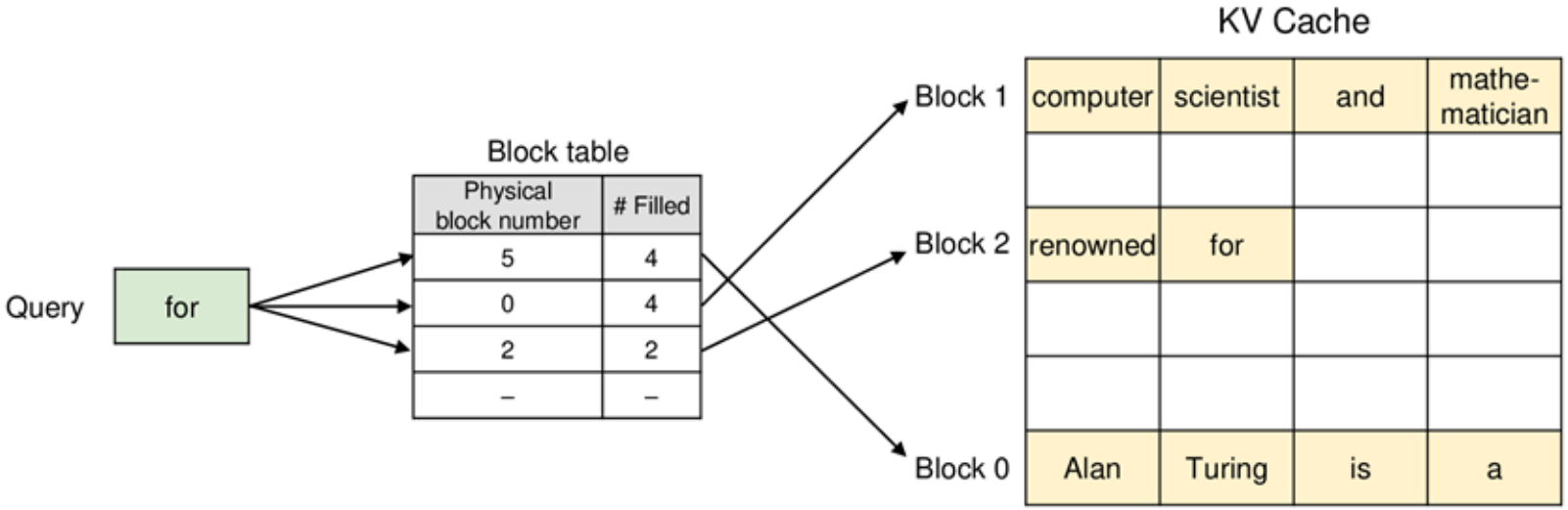
* The term ``block'' is overloaded in PagedAttention

Virtualizing KV Cache



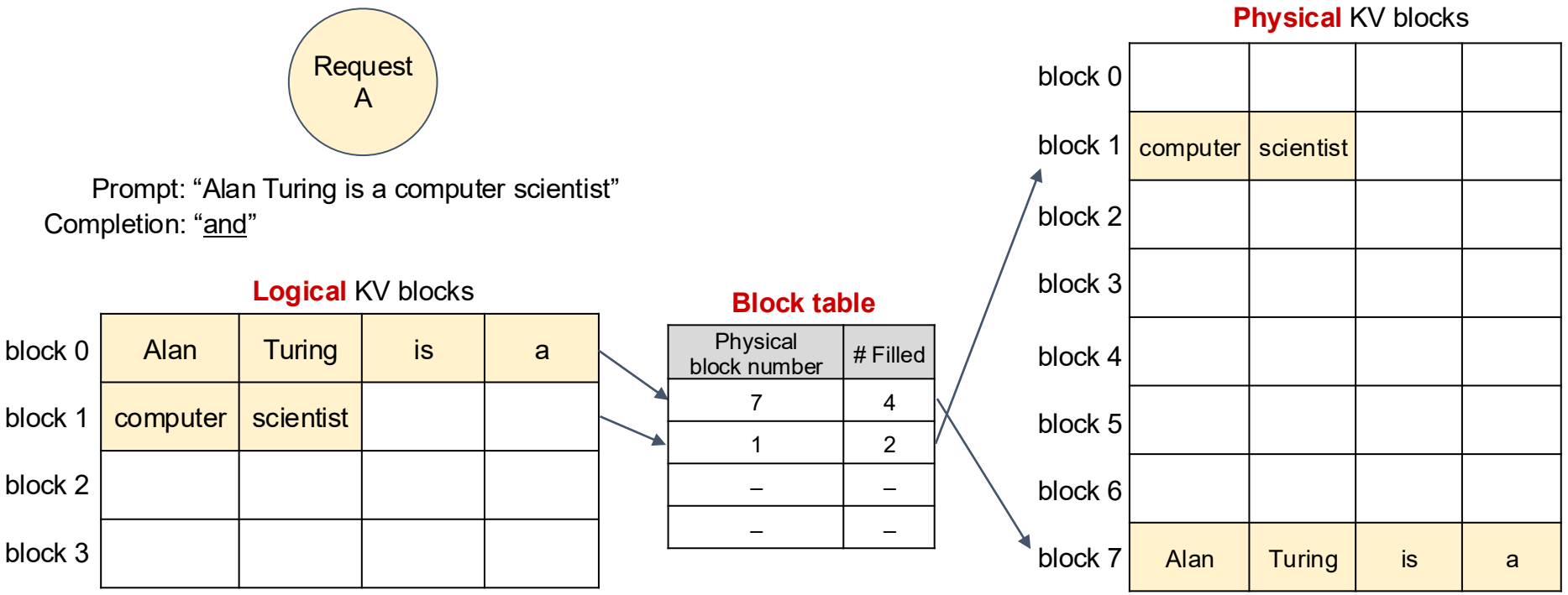
Attention with Virtualized KV Cache

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



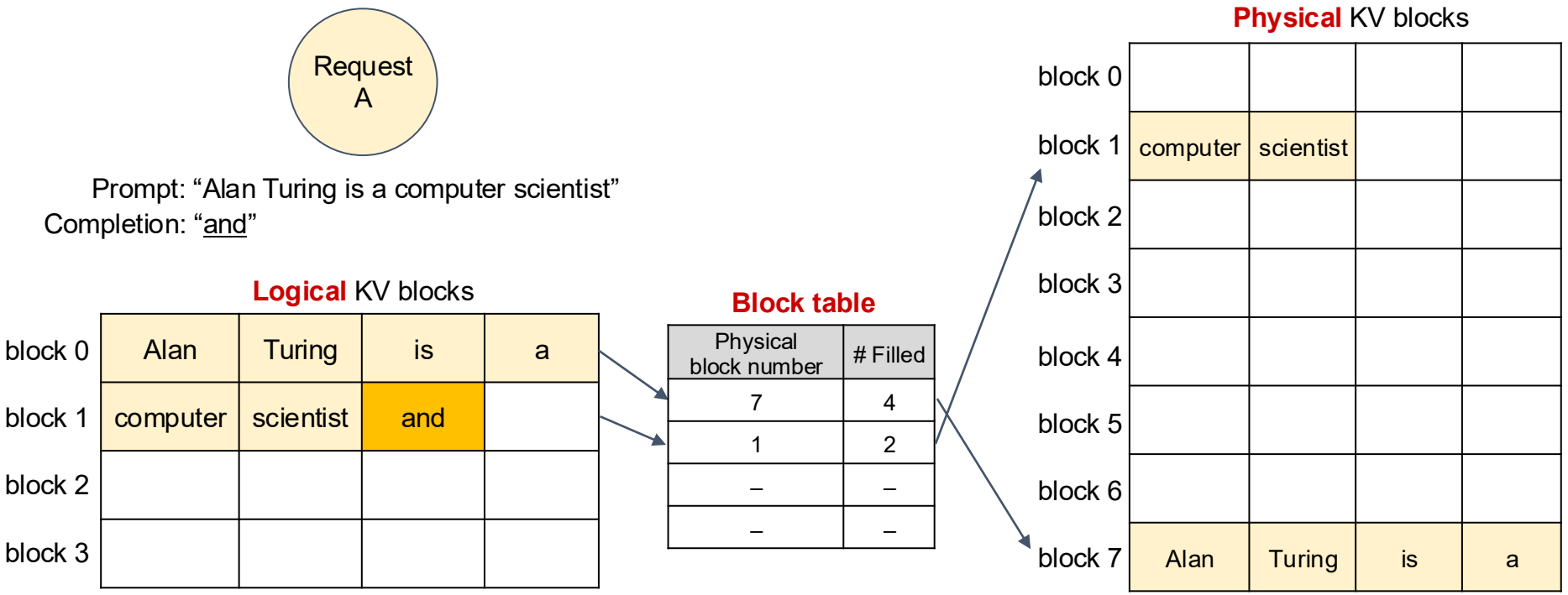
Key insight: attention is associative and commutative

Memory Management with PagedAttention

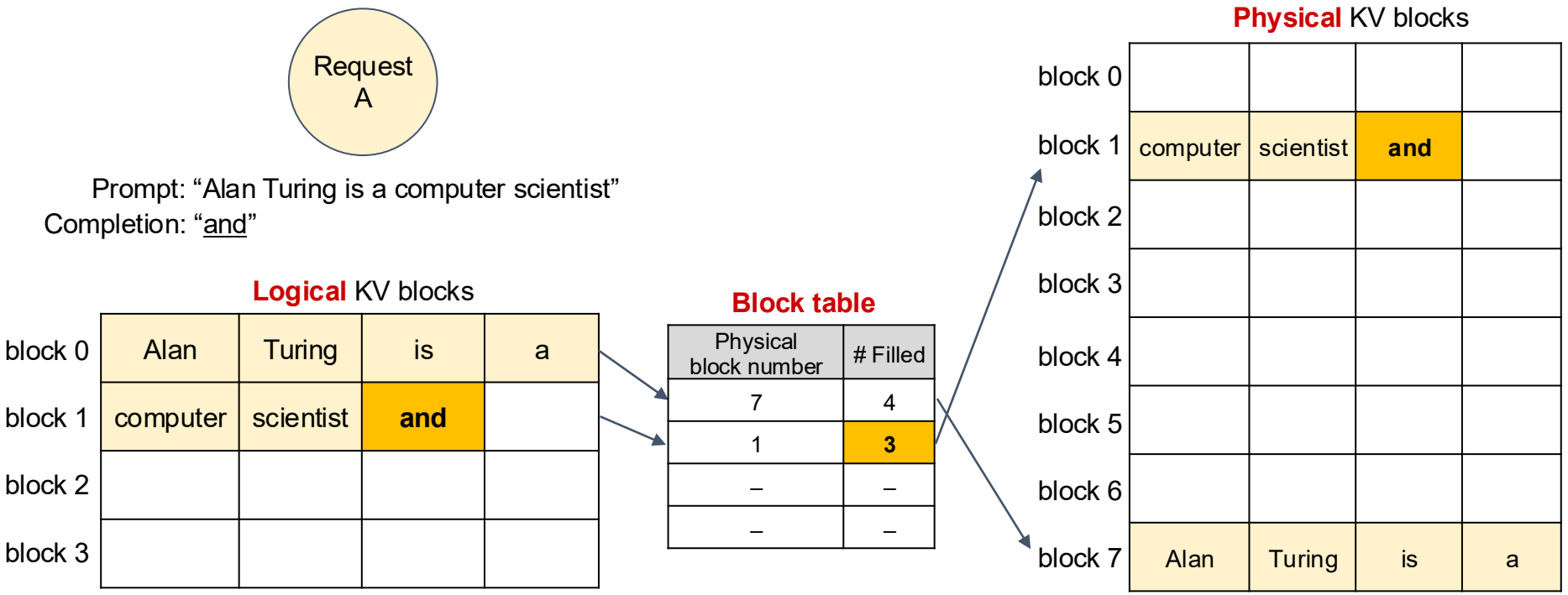


slides from vllm: Efficient Memory Management for Large Language Model Serving with PagedAttention

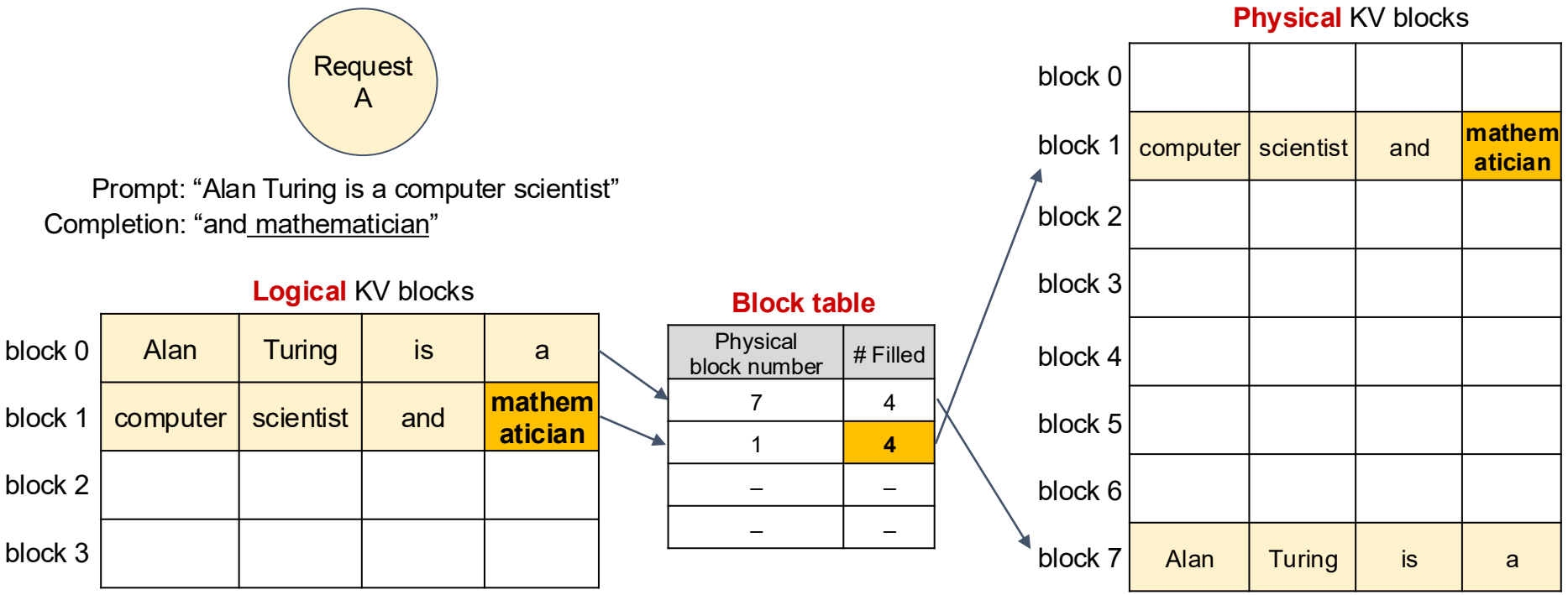
Memory Management with PagedAttention



Memory Management with PagedAttention

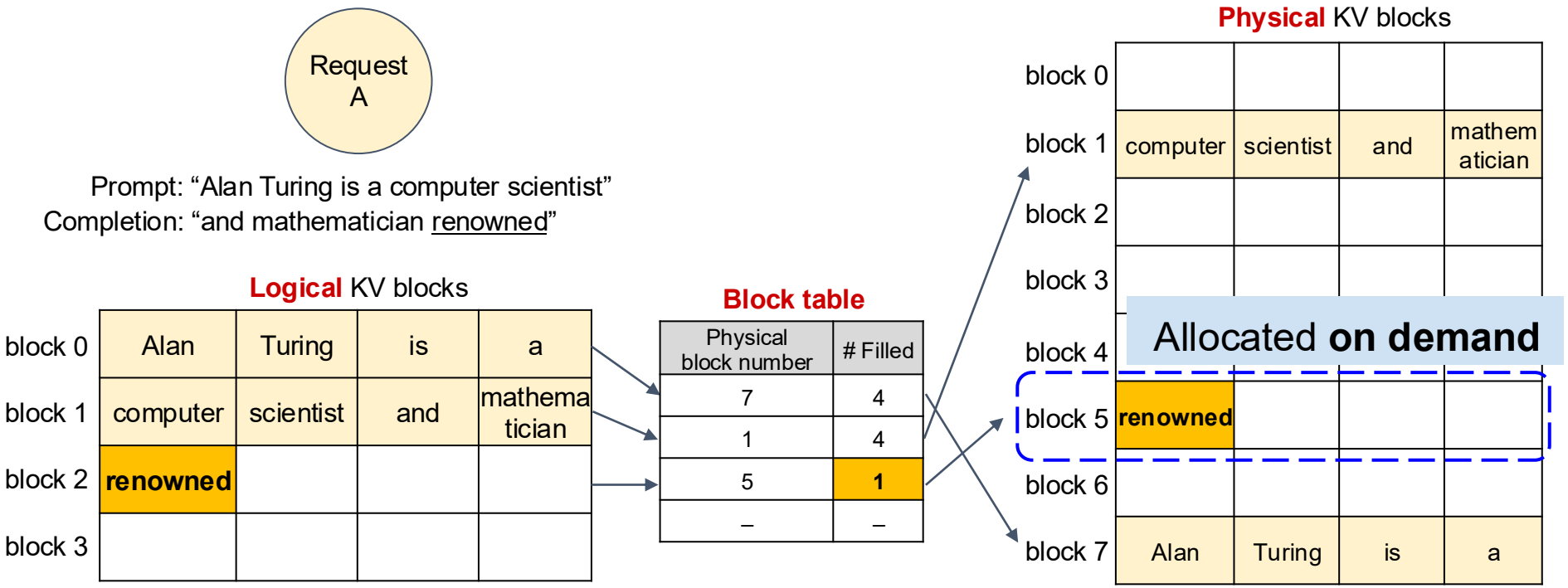


Memory Management with PagedAttention



slides from vllm: Efficient Memory Management for Large Language Model Serving with PagedAttention

Memory Management with PagedAttention



Memory Efficiency of PagedAttention

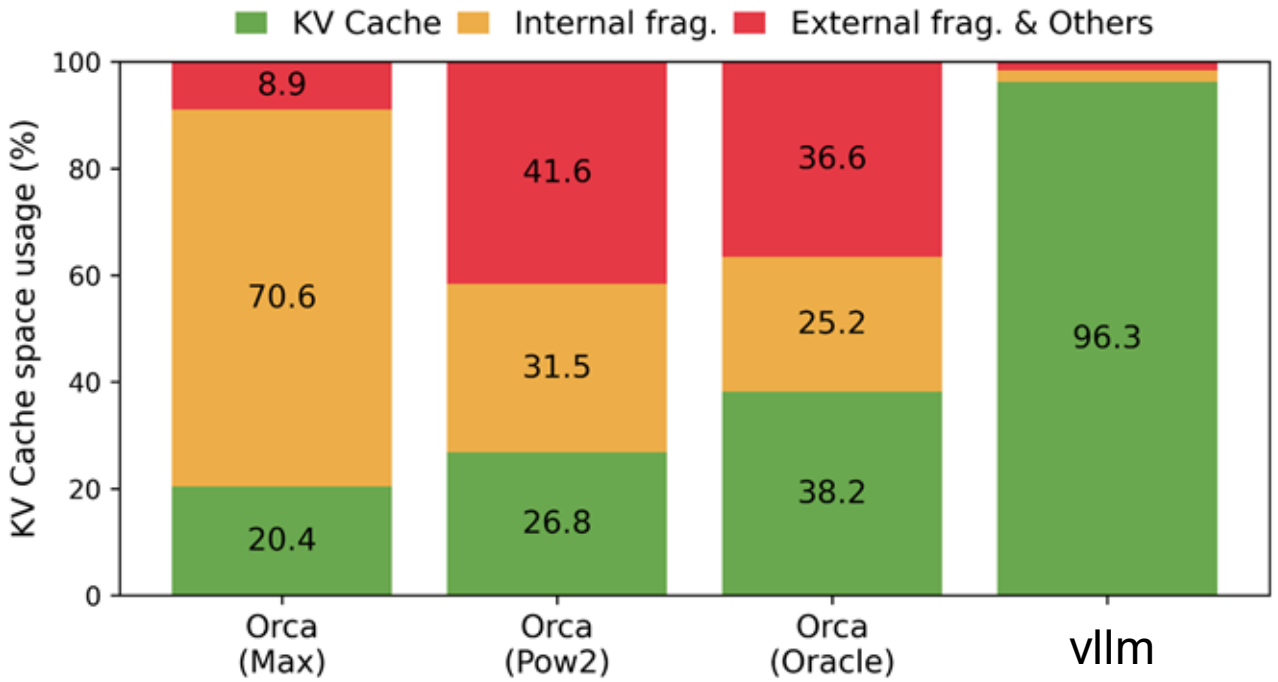
Minimal internal fragmentation

- Only happens at the last block of a sequence
- # wasted tokens per request < block size

No external fragmentation

Alan	Turing	is	a
computer	scientist	and	mathematician
renowned			

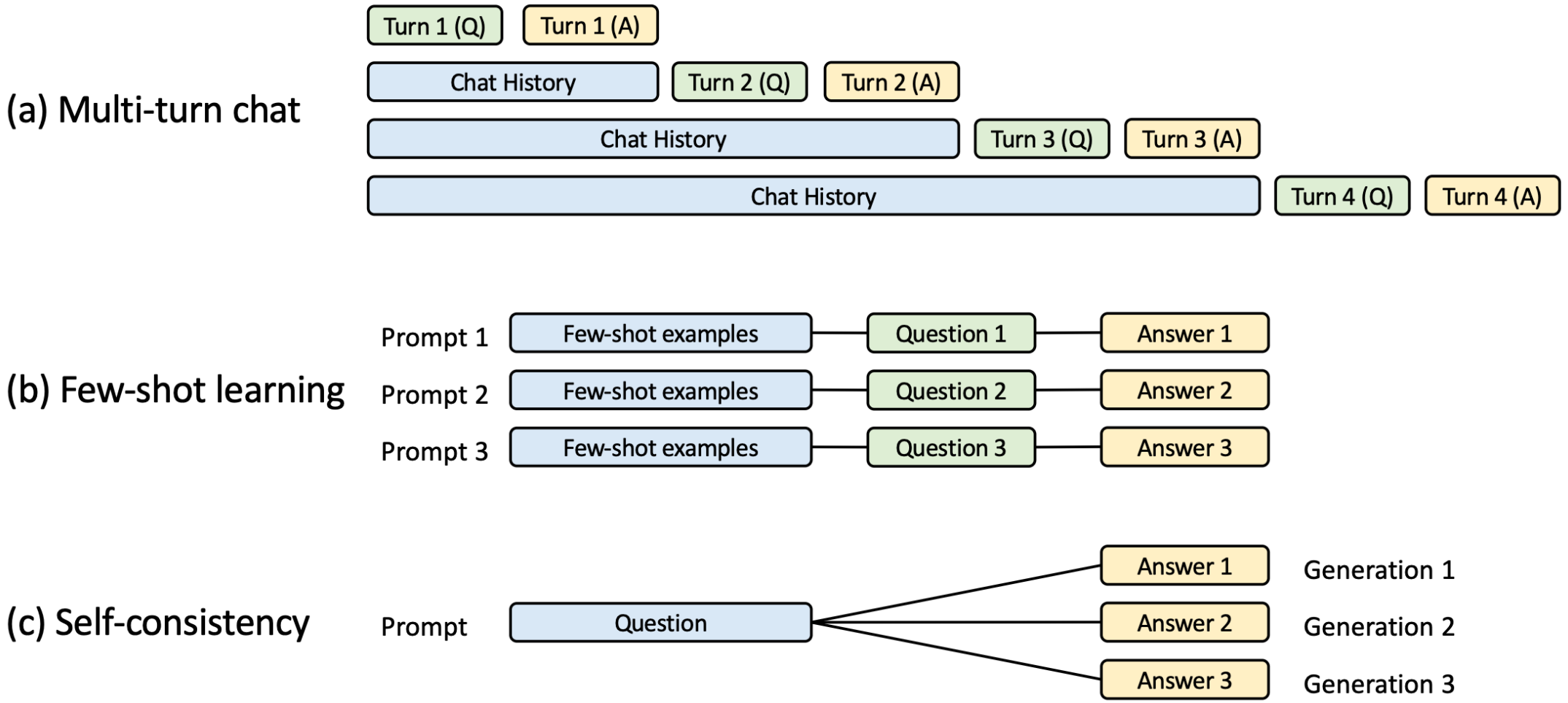
Internal fragmentation



Outline: LLMs Serving Techniques

- Continuous Batching
- PagedAttention
- **RadixAttention**

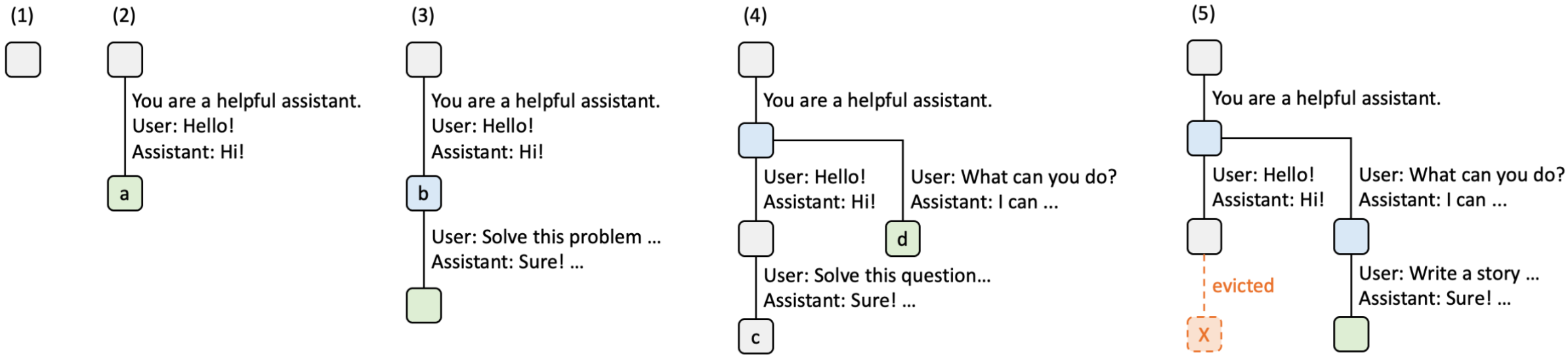
Opportunity: KV Cache Reuse



* Slides from SGLang: Efficient Execution of Structured Language Model Programs

RadixAttention: Attention with Prefix Sharing

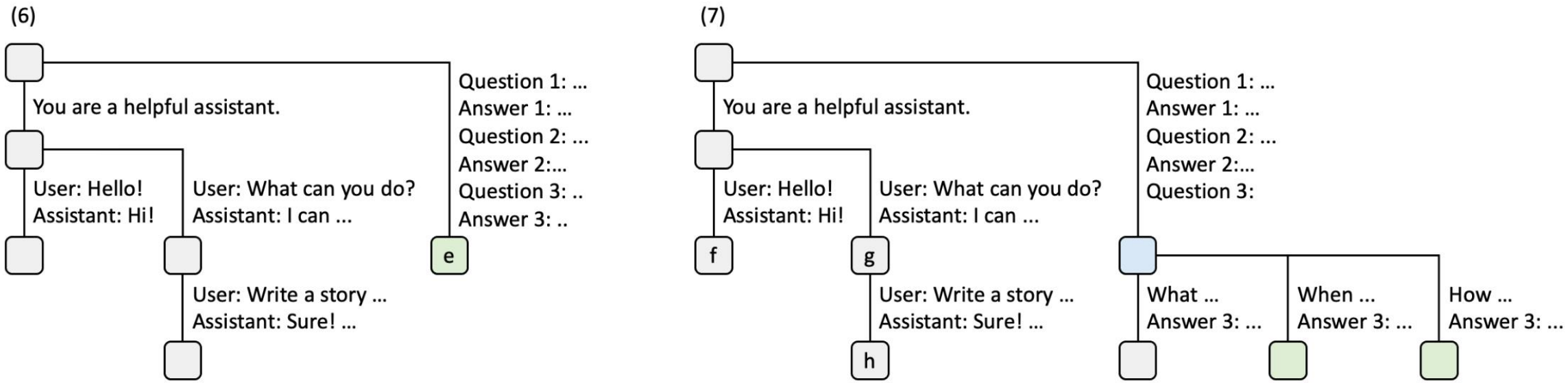
- Existing systems: discard KV cache after a request finishes
- RadixAttention: maintain an LRU cache for all requests in a radix tree (compact prefix tree)



* Slides from SGLang: Efficient Execution of Structured Language Model Programs

RadixAttention: Attention with Prefix Sharing

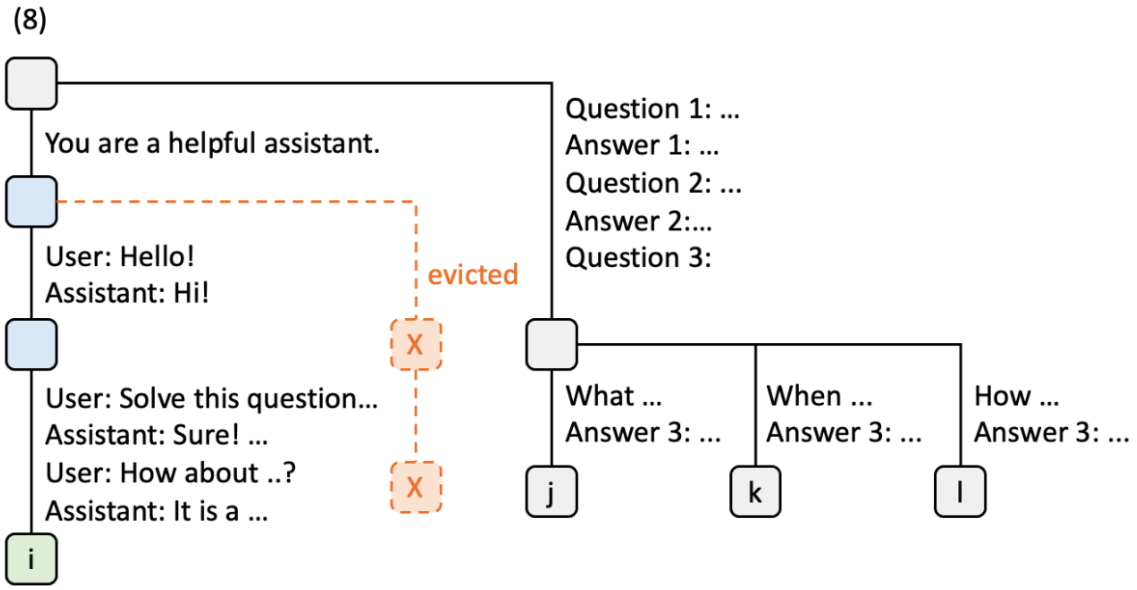
- Existing systems: discard KV cache after a request finishes
- RadixAttention: maintain an LRU cache for all requests in a radix tree (compact prefix tree)

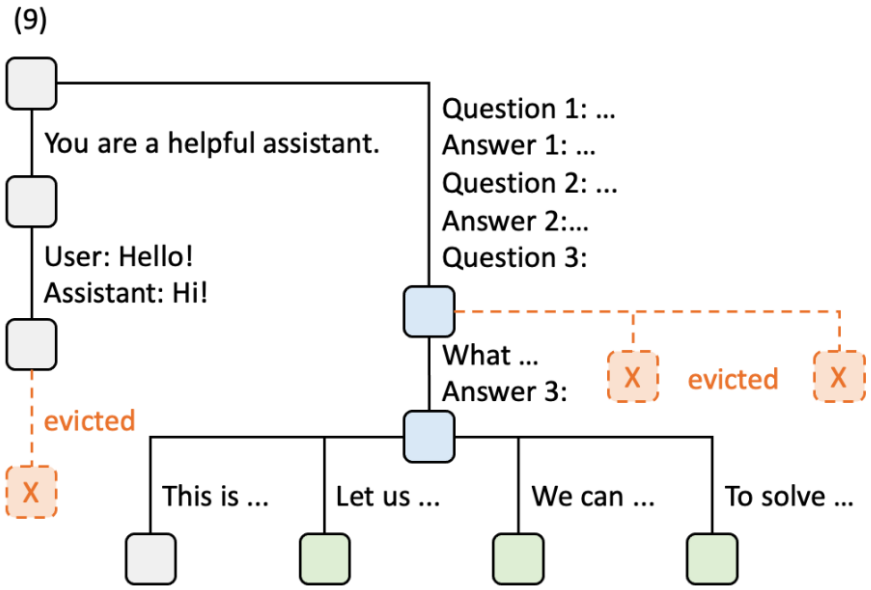


* Slides from SGLang: Efficient Execution of Structured Language Model Programs

RadixAttention: Attention with Prefix Sharing

- Existing systems: discard KV cache after a request finishes
- RadixAttention: maintain an LRU cache for all requests in a radix tree (compact prefix tree)

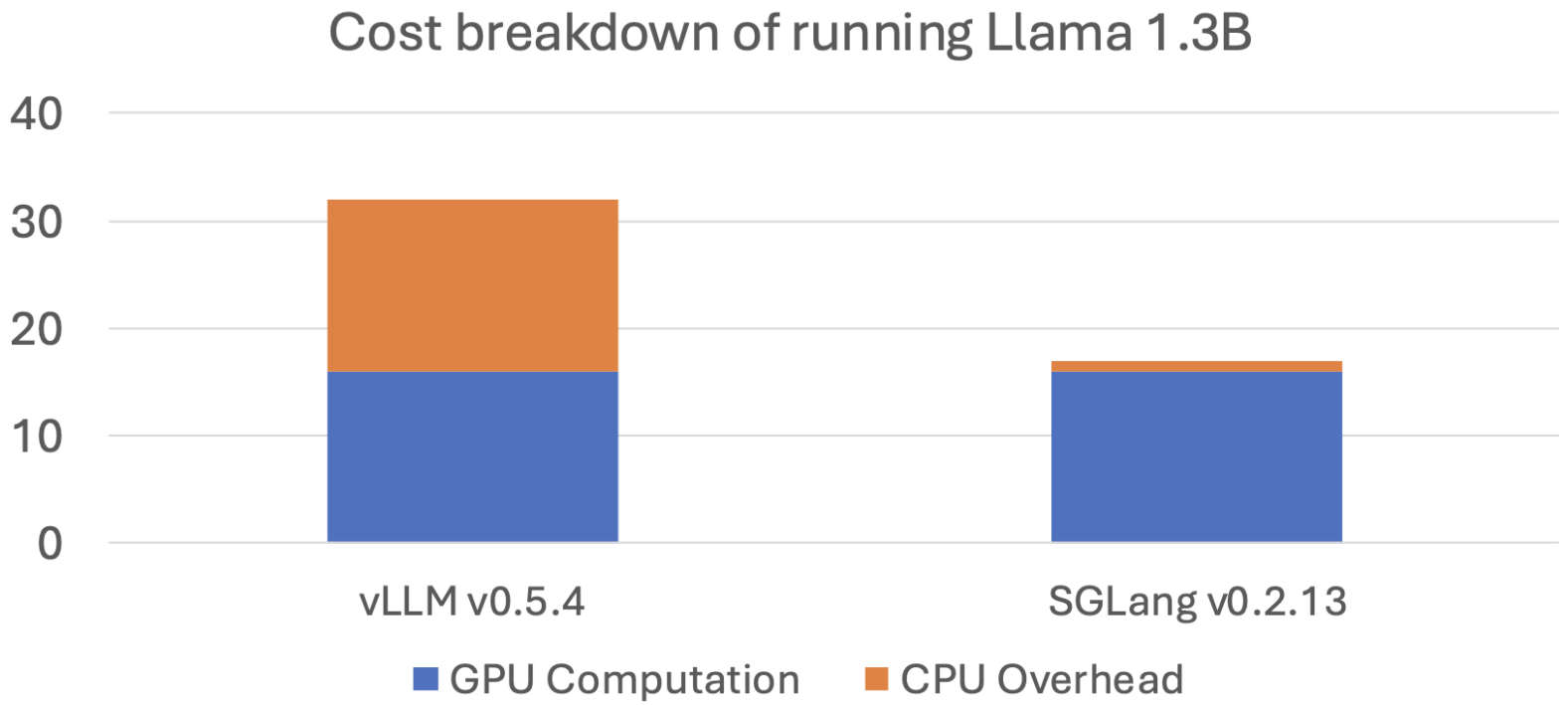




* Slides from SGLang: Efficient Execution of Structured Language Model Programs

CPU Overhead Hiding

- An unoptimized inference engine can waste more than 50% time on CPU scheduling



Source: https://mlsys.wuklab.io/posts/scheduling_overhead/

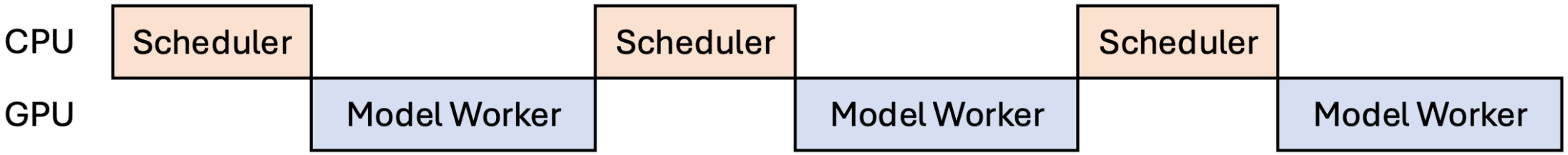
Understanding CPU Overhead

CPU scheduler iteratively

1. Receives input messages from users
2. Processes results from the model worker
3. Checks stop conditions
4. Runs prefix matching and request reorder
5. Allocates memory for the next batch

Pseudo code (every line is blocking)

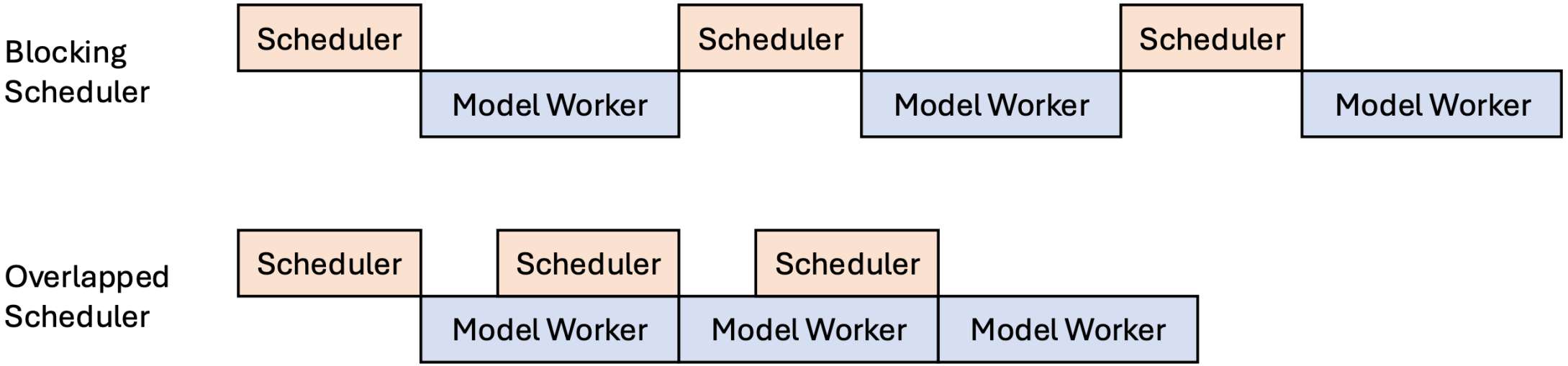
```
while True:
    recv_reqs = recv_requests()
    process_input_requests(recv_reqs)
    batch = get_next_batch_to_run()
    result = run_batch(batch)
    process_batch_result(batch , result)
```



* Slides from SGLang: Efficient Execution of Structured Language Model Programs

Overlapped Scheduler

- Hide CPU overhead by overlapping scheduler & model worker



* Slides from SGLang: Efficient Execution of Structured Language Model Programs

Overlapped Scheduling in SGLang

- Goal: overlap run_batch (GPU) and process_batch_result (CPU)

Normal version

```
while True:
    recv_reqs = recv_requests()
    process_input_requests(recv_reqs)
    batch = get_next_batch_to_run()
    result = run_batch(batch)
    process_batch_result(batch , result)
```

Overlap version

```
last_batch = None
while True:
    recv_reqs = recv_requests()
    process_input_requests(recv_reqs)
    batch = get_next_batch_to_run()
    result = run_batch(batch)
    result_queue.put((batch, result))
    if last_batch is not None:
        tmp_batch, tmp_result = result_queue.get()
        process_batch_result(tmp_batch, tmp_result)
    last_batch = batch
```

run_batch is non-blocking and returns a reference; overlapping run_batch of the current iteration with process_batch_result of the previous iteration

* Slides from SGLang: Efficient Execution of Structured Language Model Programs

Resolve Dependency

- **Key idea**: delay the finish condition check
- We assume a request did not finish and immediately run it in the next decoding batch
- We resolve dependency by paying the overhead of decoding one more useless token

* Slides from SGLang: Efficient Execution of Structured Language Model Programs

Recap: LLMs Serving Techniques

- Continuous Batching
  - Handle early-finished and late-arrived requests more efficiently
- PagedAttention
  - Application-level memory paging and virtualization for KV cache
- RadixAttention
  - Maintain an LRU cache for all requests in a radix tree