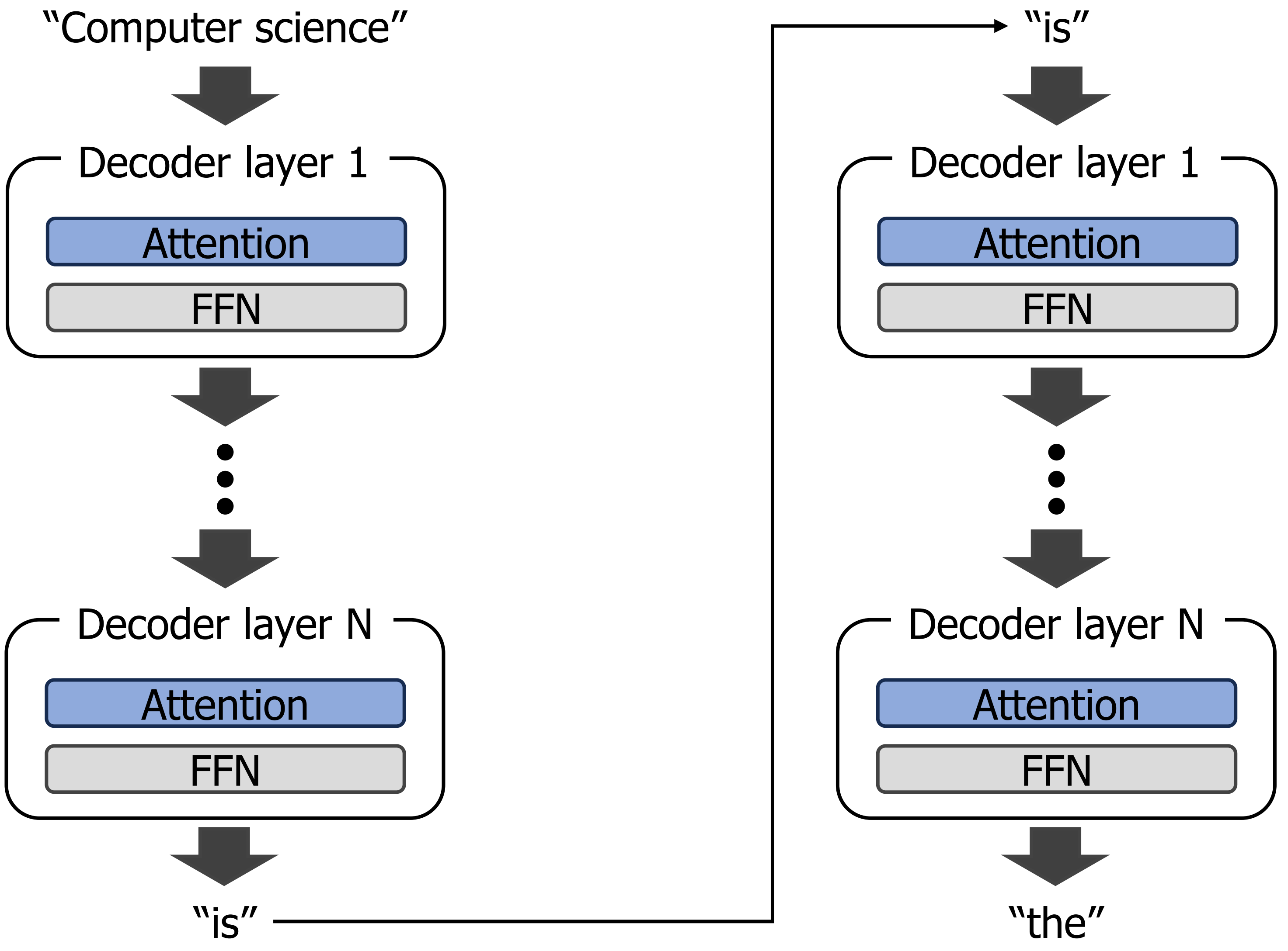
Accelerating LLM Serving for Multi-turn Dialogues with Efficient Resource Management

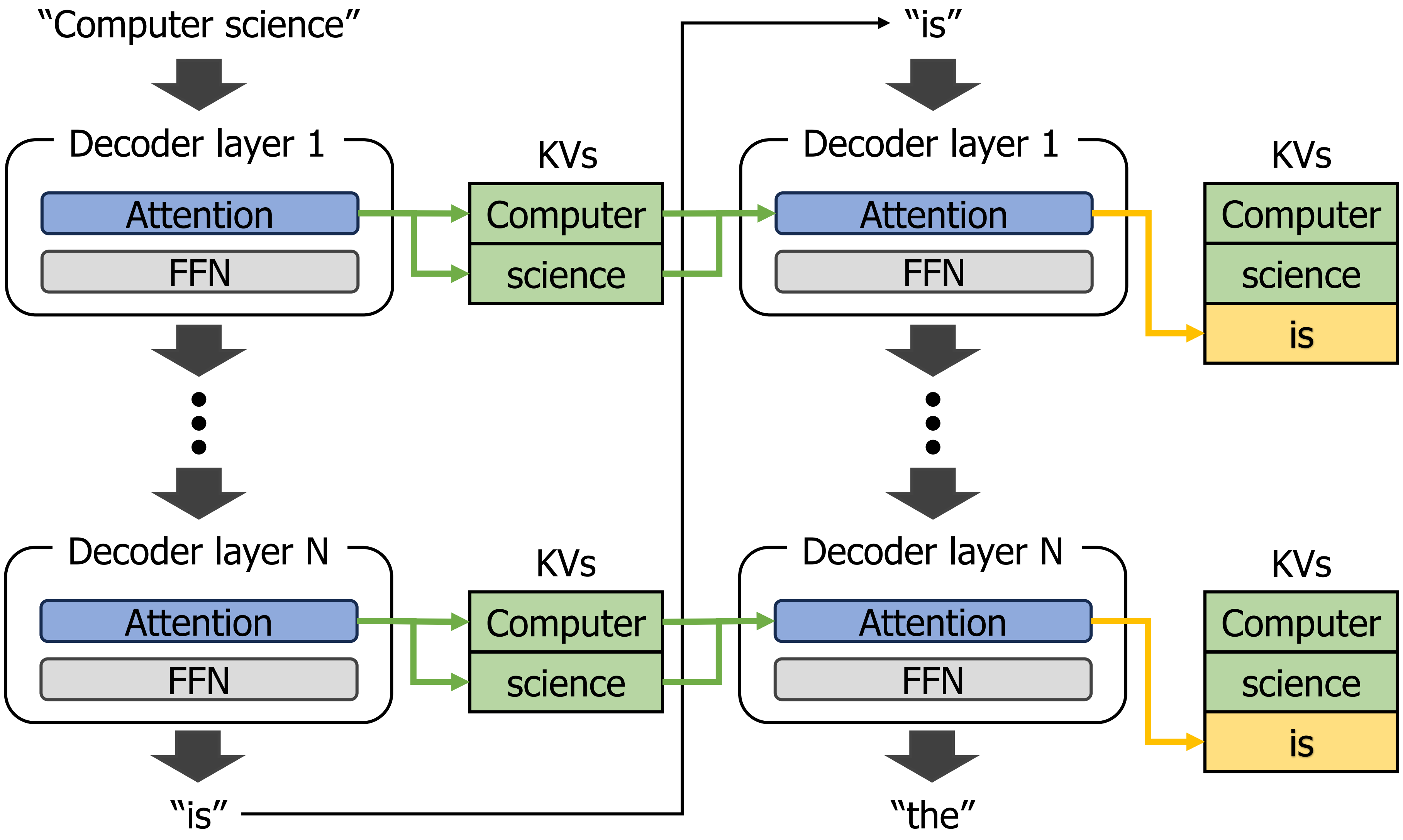
Jinwoo Jeong Jeongseob Ahn

KOREA UNIVERSITY

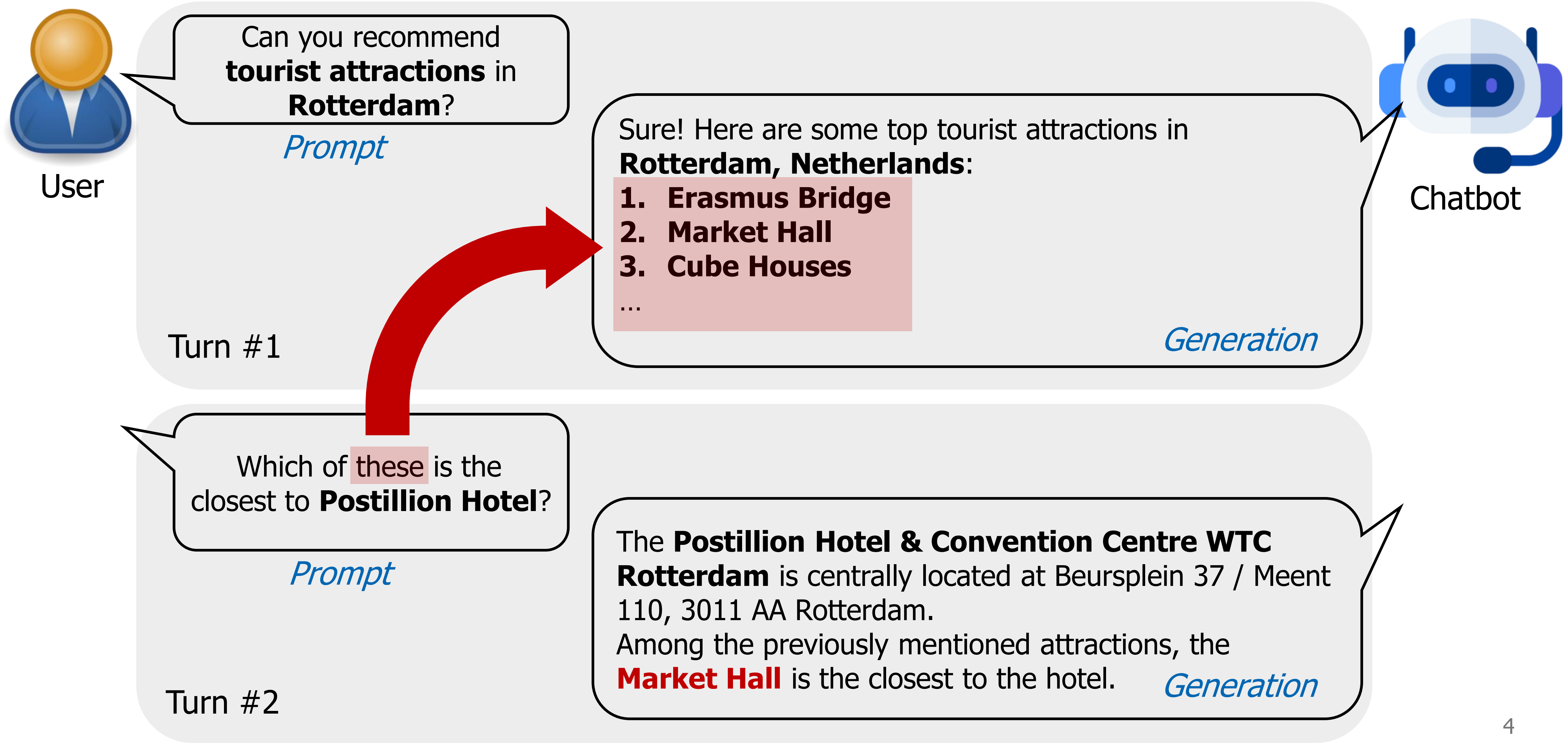
Transformer-based Text Generation



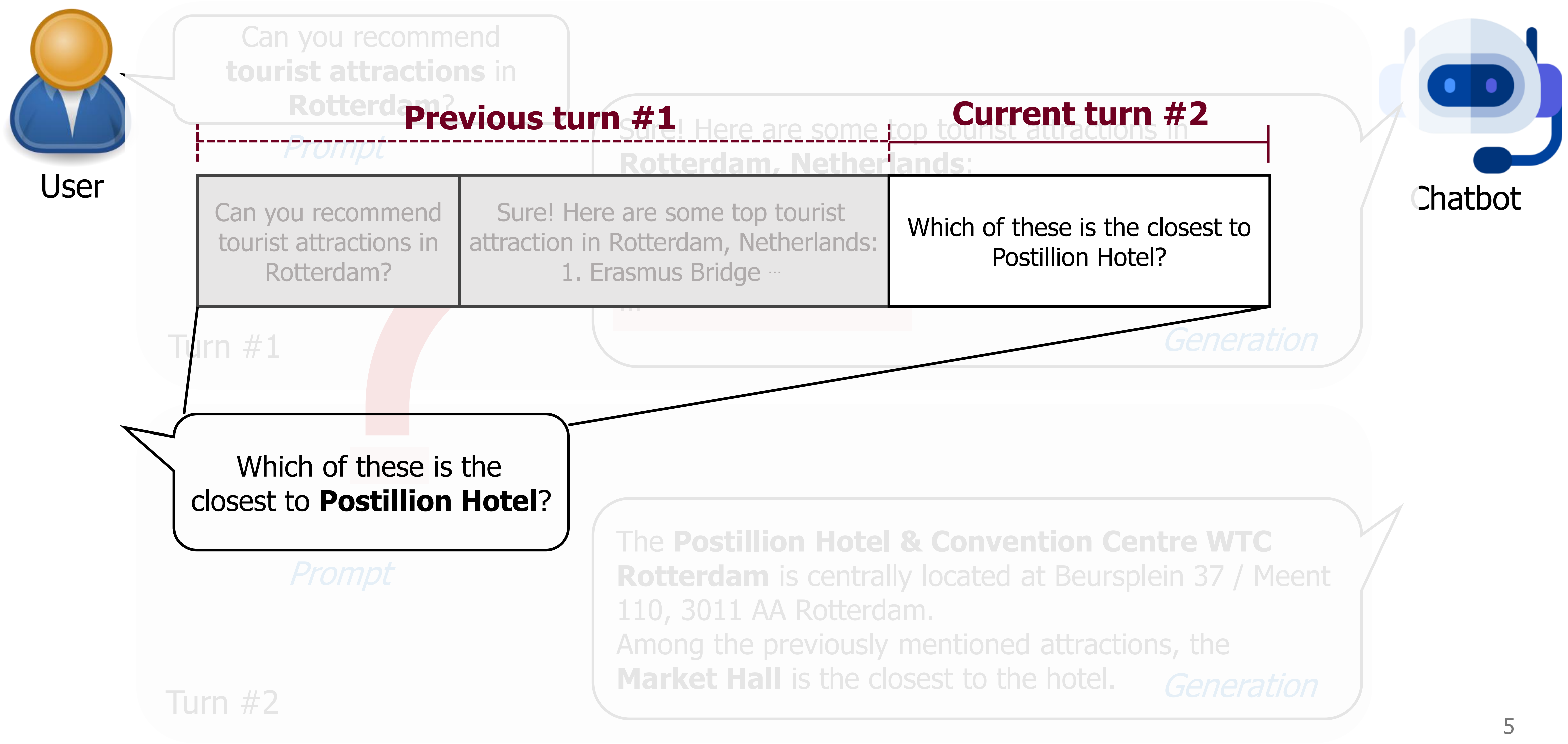
Transformer-based Text Generation



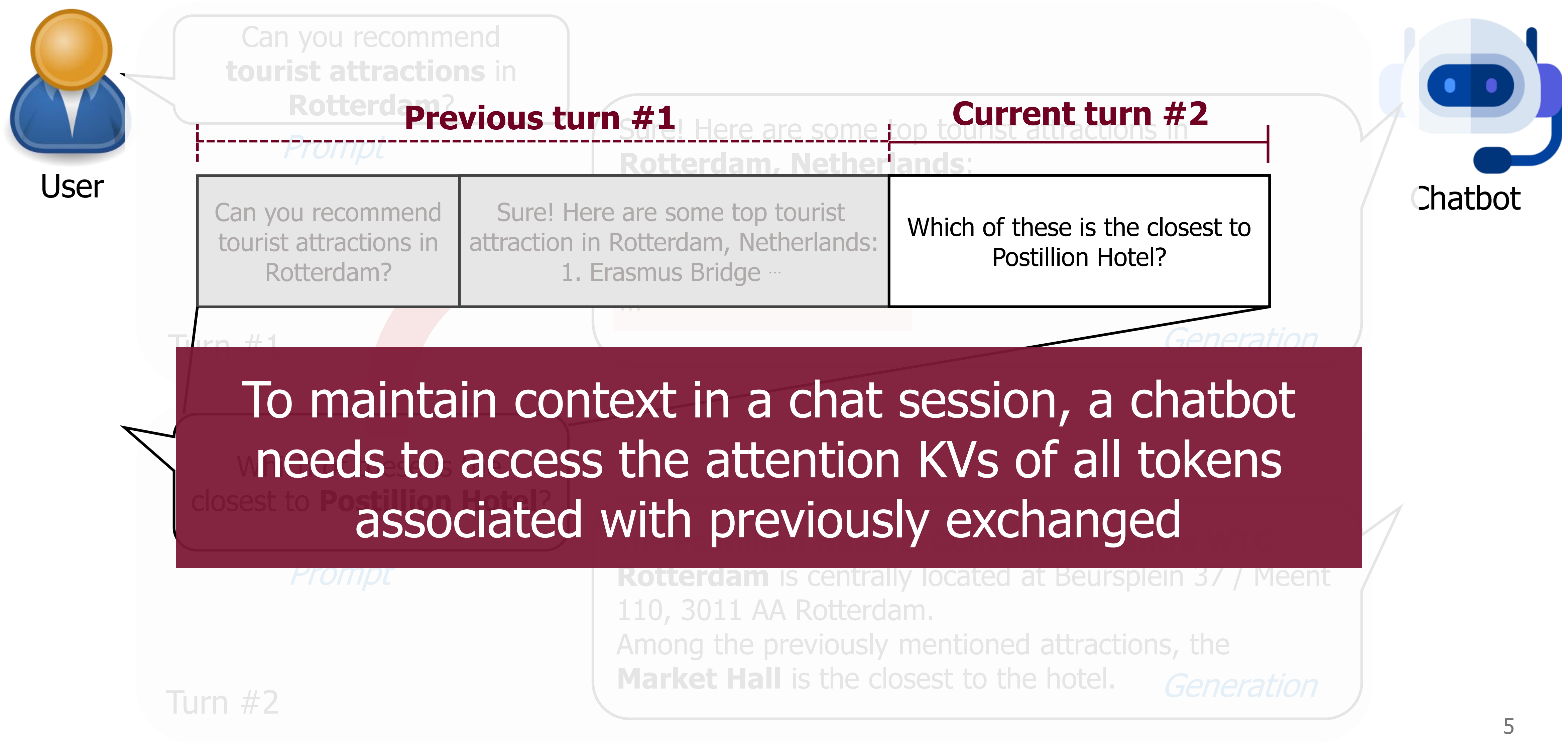
Multi-turn Dialogues with a Chatbot



Multi-turn Dialogues with a Chatbot



Multi-turn Dialogues with a Chatbot



To maintain context in a chat session, a chatbot needs to access the attention KVs of all tokens associated with previously exchanged

Challenges of LLM Serving in Multi-turn

1. Repeatedly access the history to generate context-aware answer
   - Problem: Existing LLM serving systems recompute history KVs at every turns
   - Solution: Retaining history KVs in the memory hierarchy

     ➔ **FlashGen-Cache**

2. Amplifying prompt length due to accumulation of history
   - Problem: Exacerbate a head-of-line blocking problem caused by FCFS
   - Solution: Reordering shorter prompt to fill available free space

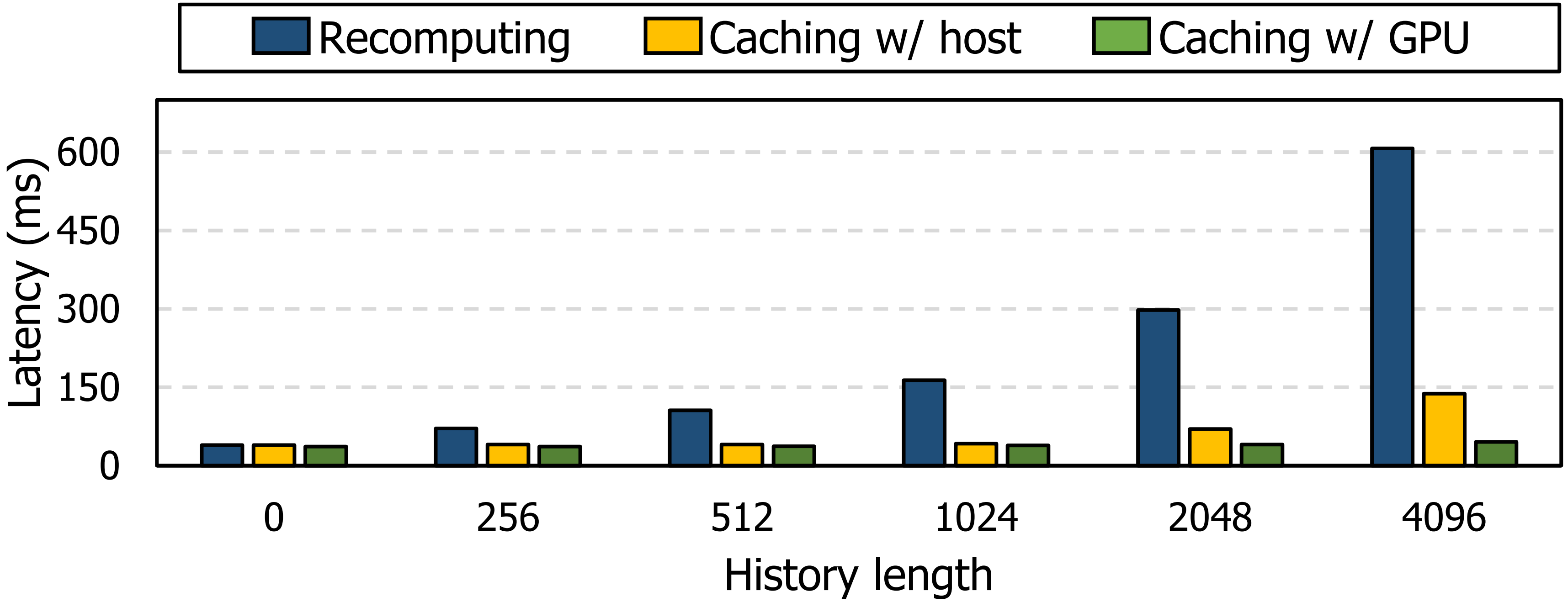
     ➔ **FlashGen-Sched**

Our solution: **FlashGen**

➔ Integrates FlashGen-Cache and -Sched for efficient multi-turn LLM serving

Cost of Handling Multi-turn Prompts

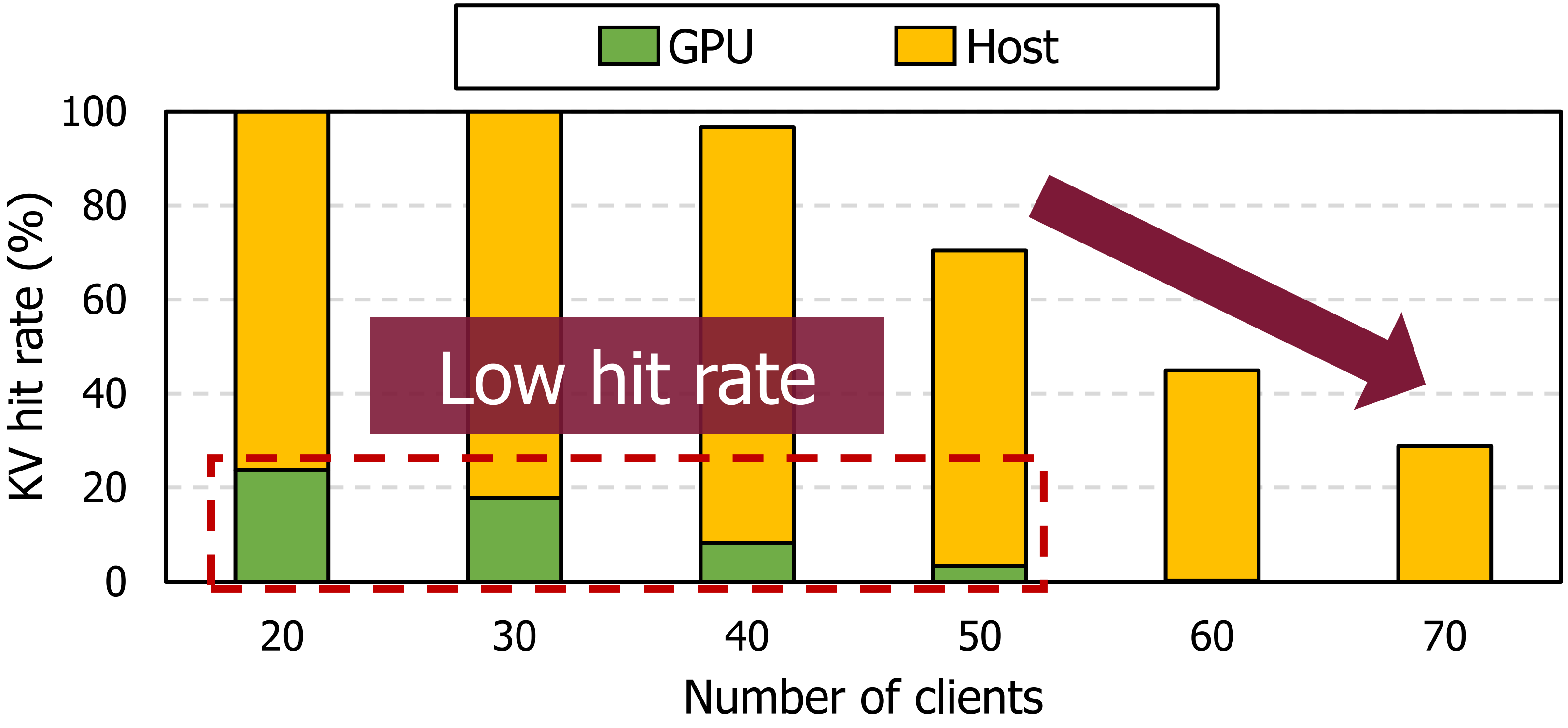
- Current LLM serving frameworks (e.g., vLLM and TensorRT-LLM) takes an approach that **recompute** previous turns
  - ➔ Leading to significant performance overhead due to recomputation



- Model: OPT-13B
- GPU: A100 (80GB)
- Prompt length: 256

History KV Cache Hit Rate

- Under high request loads (e.g., increased concurrent users)
  - Both GPU and host memory become insufficient for caching history KVs



- Dataset: ShareGPT
- Model: OPT-30B
- GPU: A100 (80GB)
- Host memory: 224GB

FlashGen-Cache

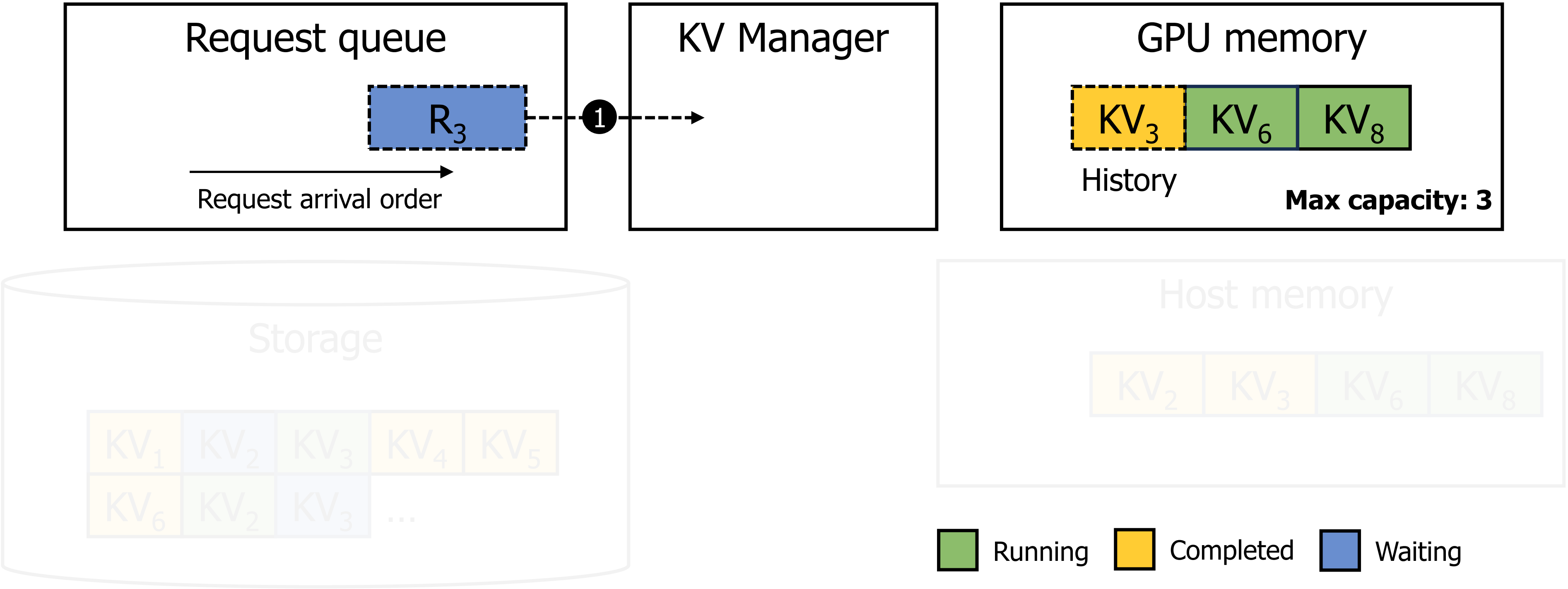
- Multi-level attention KV cache
  - Leverage **GPU memory**, **host memory**, and even **storage**

- KV Cache hit scenario

1. **GPU memory** → serve request immediately
2. **Host memory** → load KVs to GPU memory
3. **Storage** → stage KVs in host memory and load KVs to GPU memory

Scenario 1: Requested KVs in GPU Memory

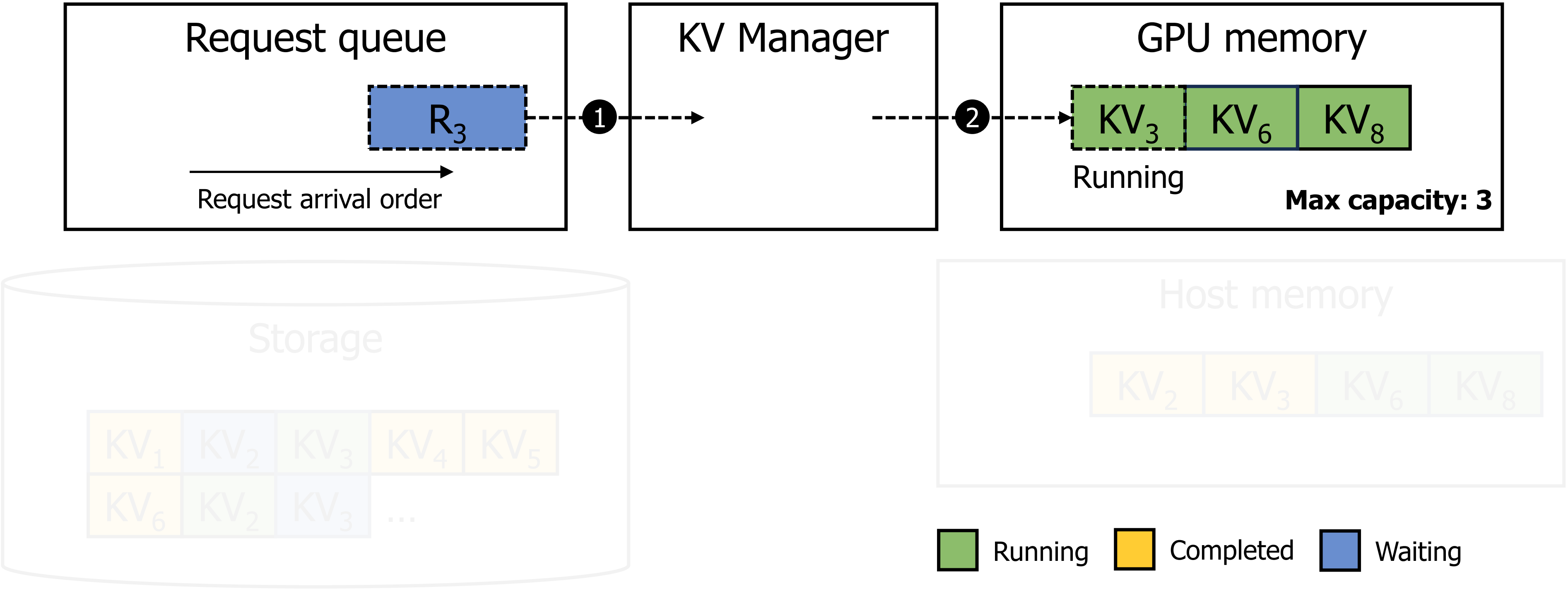
- Upon a GPU cache hit
  - Serve the request with history KVs cached in GPU memory without recomputing them



*subscription means *session ID*

Scenario 1: Requested KVs in GPU Memory

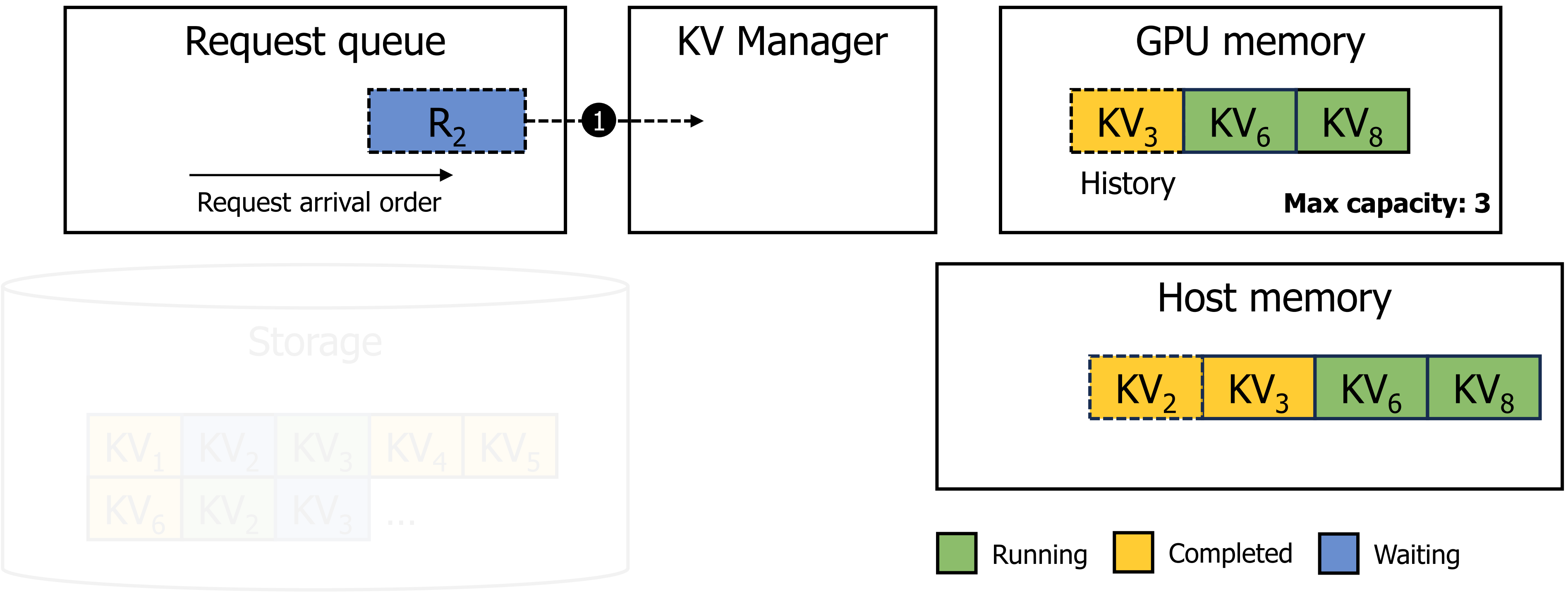
- Upon a GPU cache hit
  - Serve the request with history KVs cached in GPU memory without recomputing them



*subscription means *session ID*

Scenario 2: Requested KVs in Host Memory

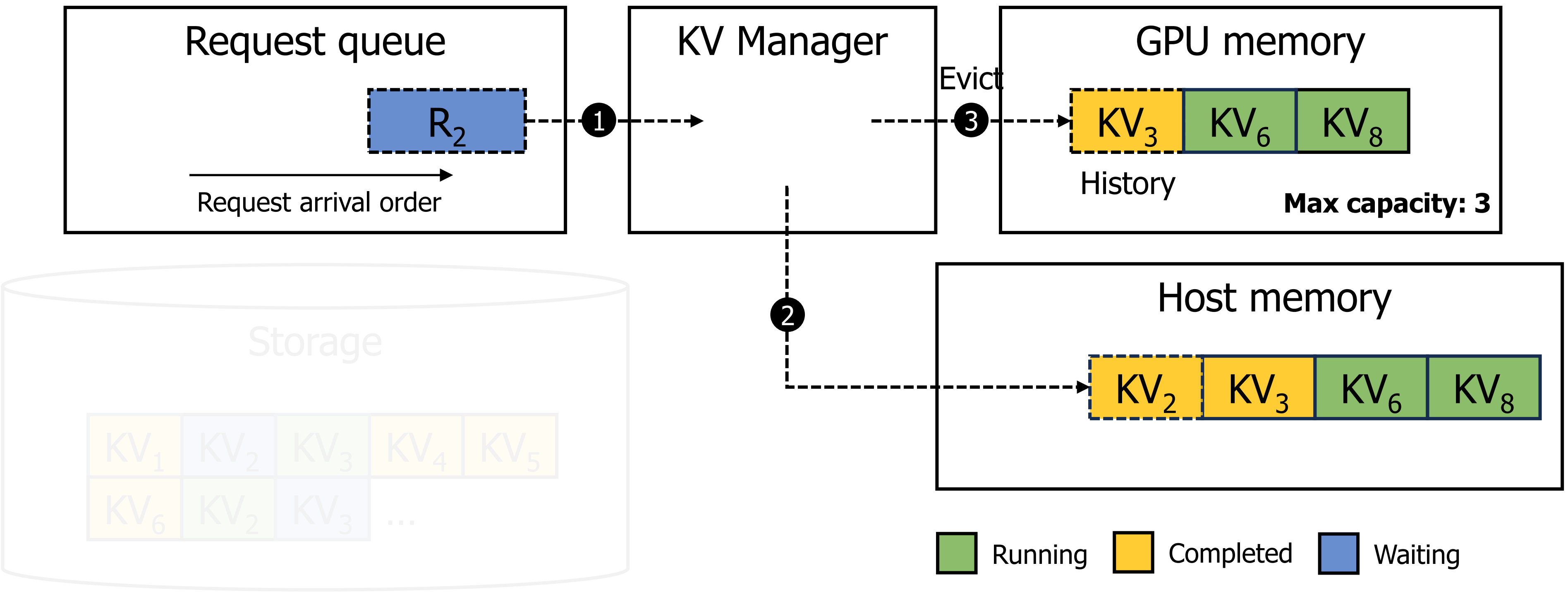
- Upon a GPU cache miss
  - If the history KVs are in host memory, transfer them to GPU memory



*subscription means *session ID*

Scenario 2: Requested KVs in Host Memory

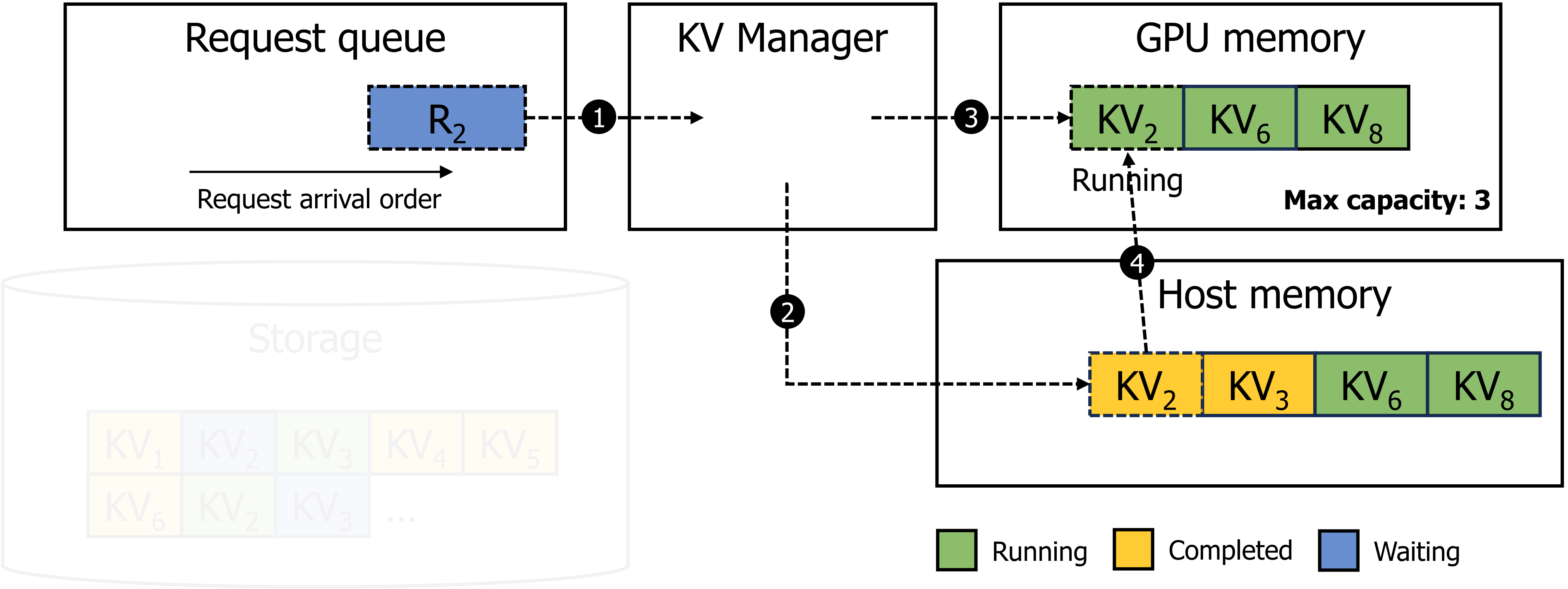
- Upon a GPU cache miss
  - If the history KVs are in host memory, transfer them to GPU memory



*subscription means *session ID*

Scenario 2: Requested KVs in Host Memory

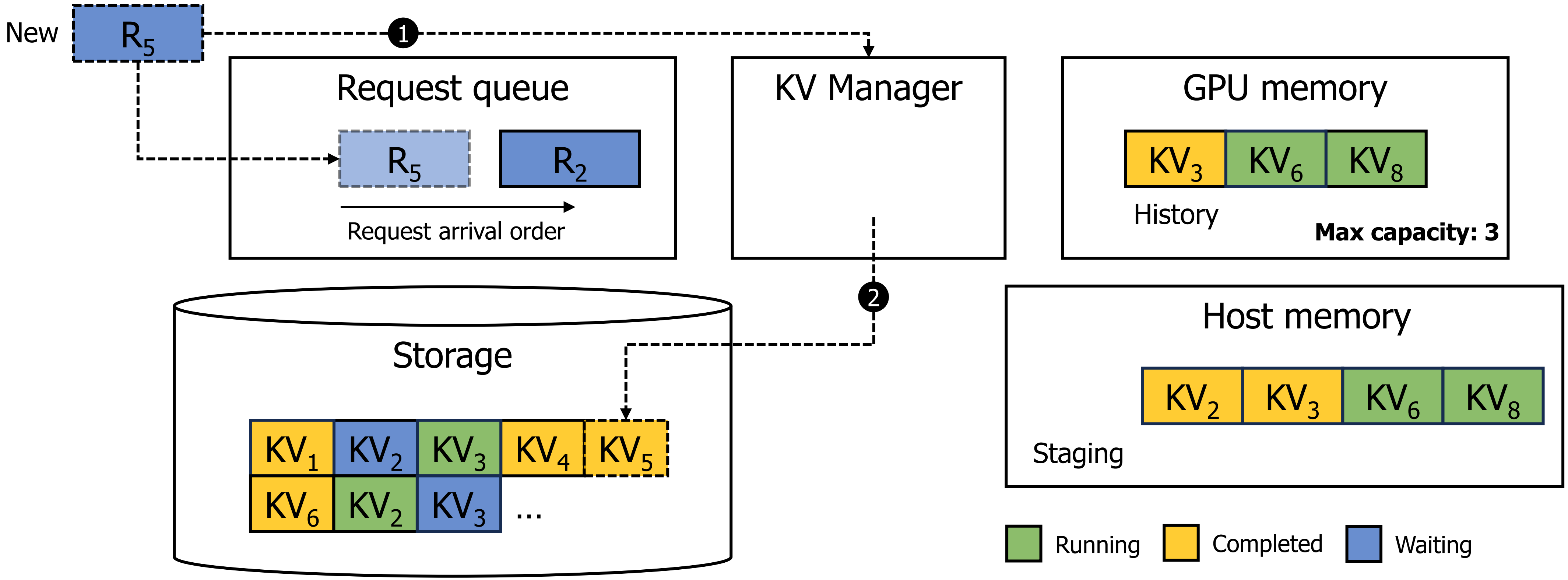
- Upon a GPU cache miss
  - If the history KVs are in host memory, transfer them to GPU memory



*subscription means *session ID*

Scenario 3: Requested KVs in Storage

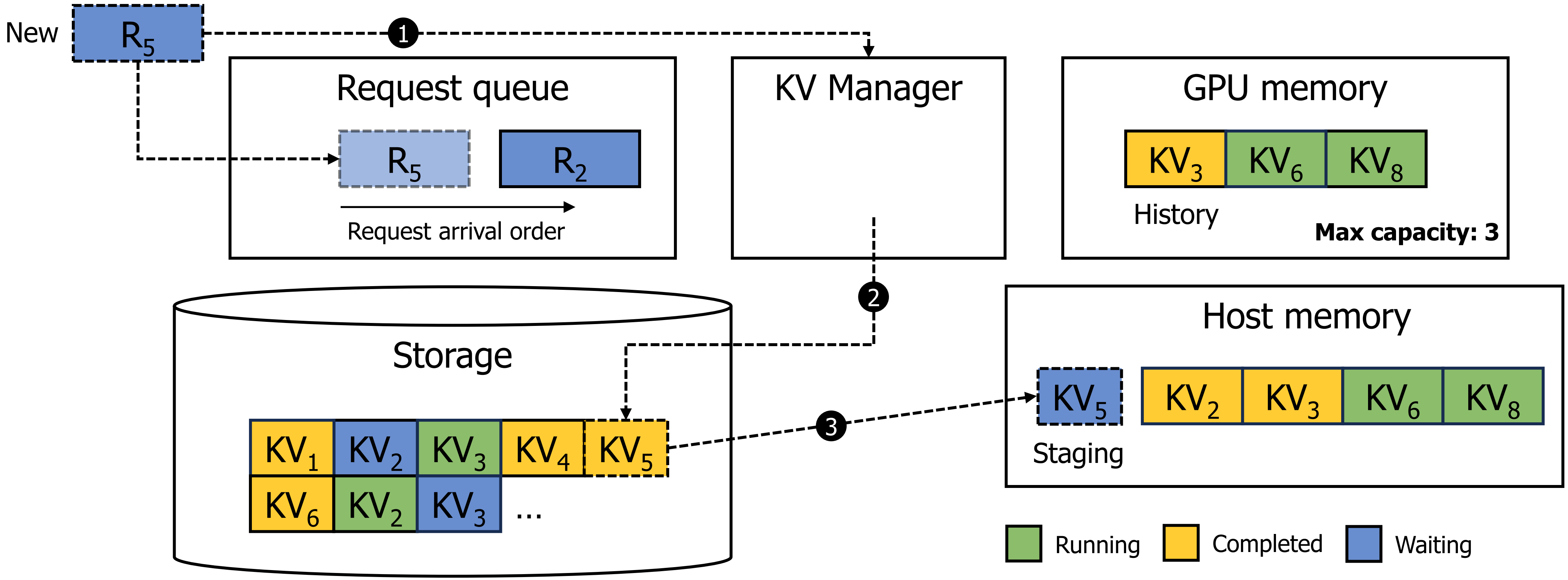
- Upon receiving a request
  - If its history KVs are not in host memory, stage them into host memory



*subscription means *session ID*

Scenario 3: Requested KVs in Storage

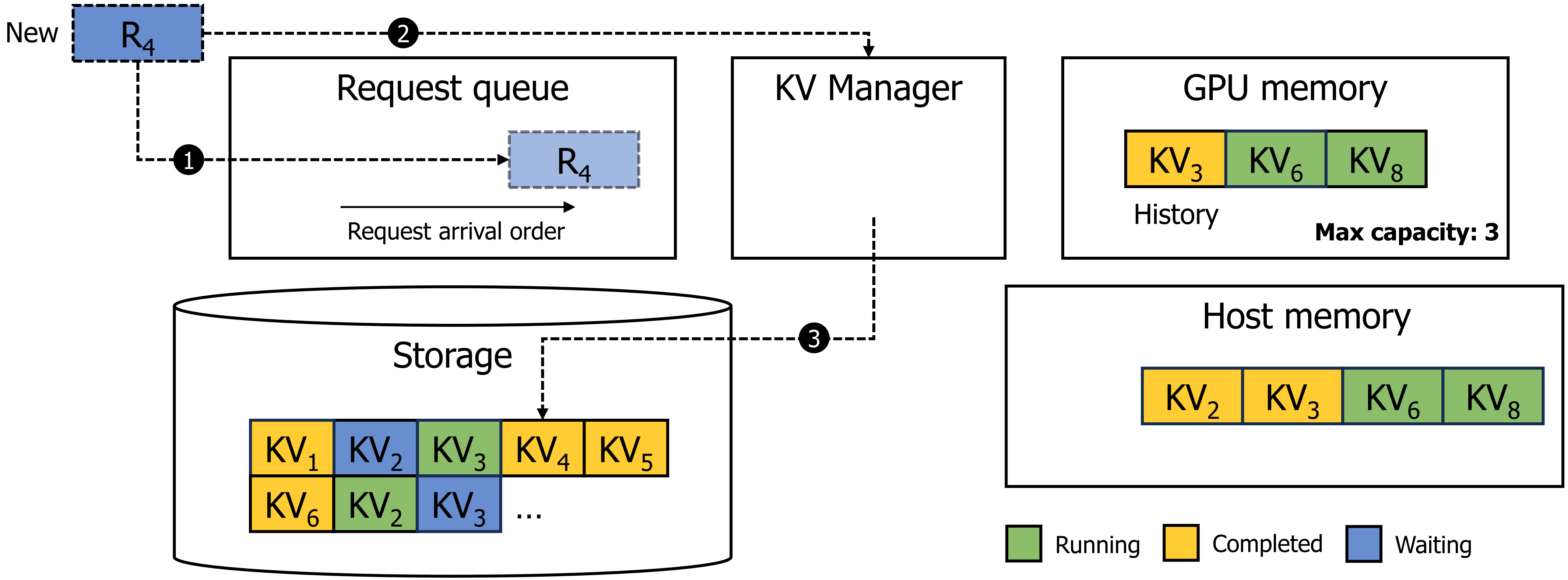
- Upon receiving a request
  - If its history KVs are not in host memory, stage them into host memory



*subscription means *session ID*

Scenario 3: Requested KVs in Storage

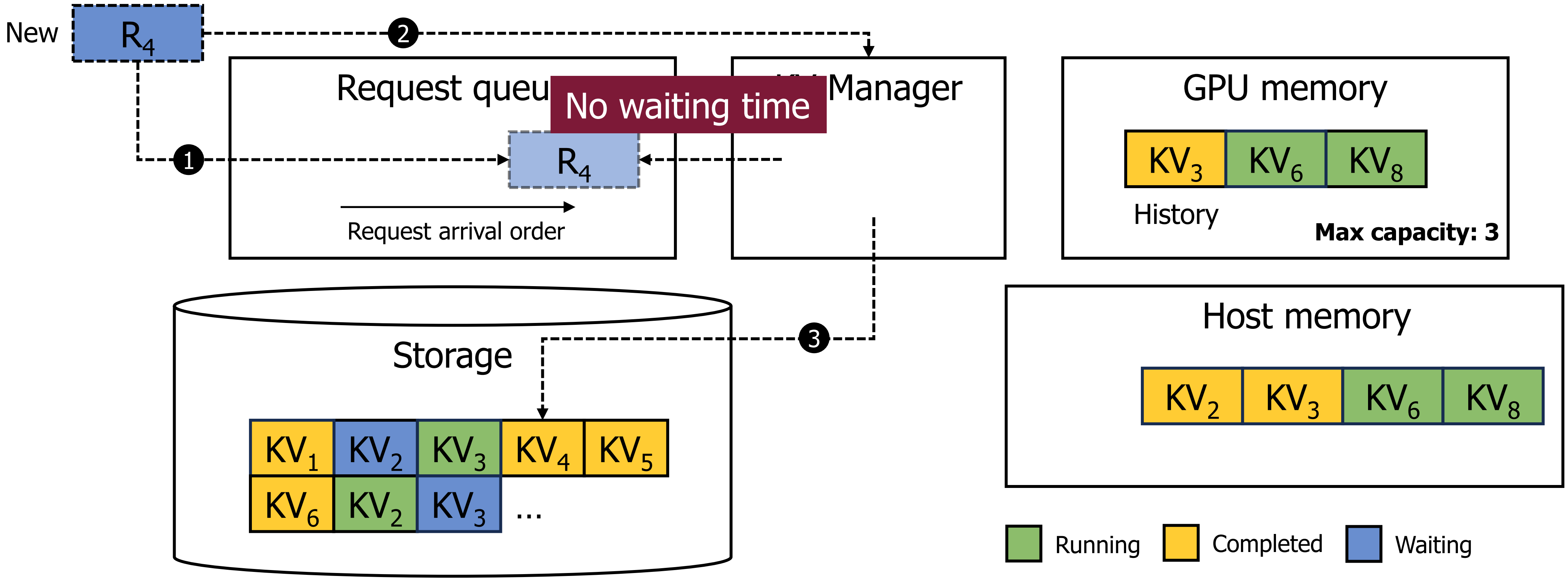
- If the staging phase cannot be hidden (i.e., waiting reqs < 1)
  - Recompute instead of retrieving from storage due to bandwidth limits



*subscription means *session ID*

Scenario 3: Requested KVs in Storage

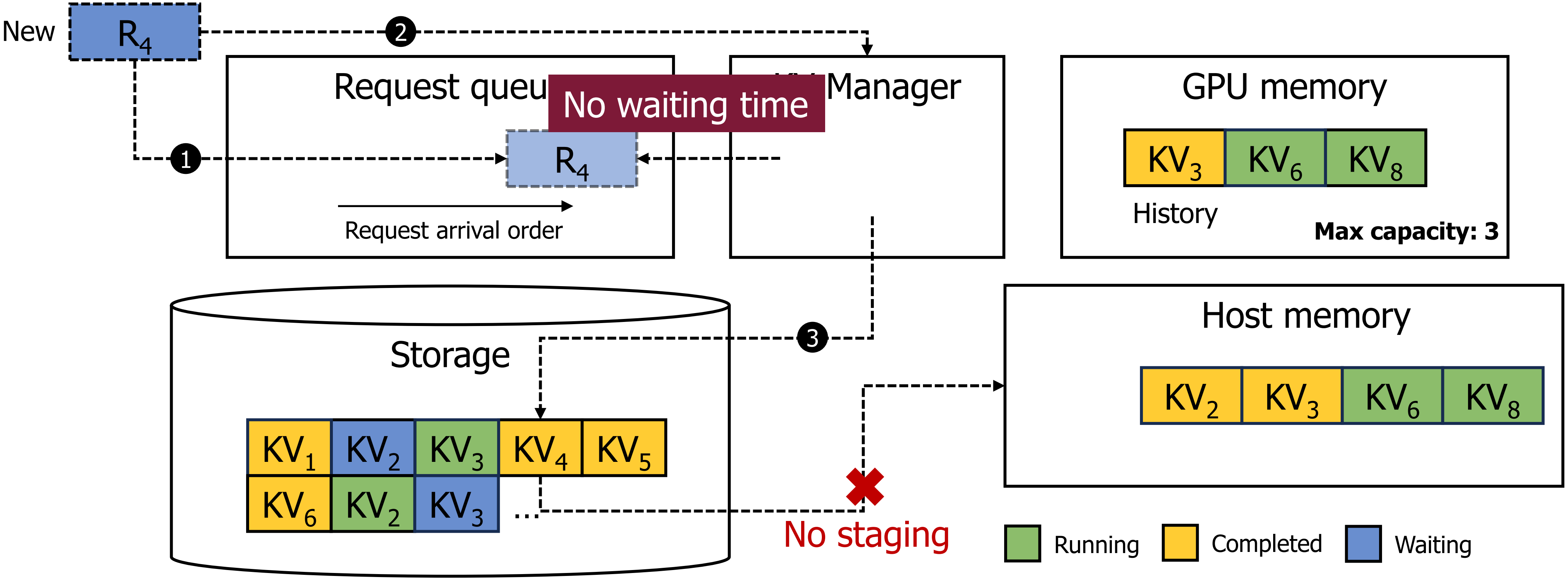
- If the staging phase cannot be hidden (i.e., waiting reqs < 1)
  - Recompute instead of retrieving from storage due to bandwidth limits



*subscription means *session ID*

Scenario 3: Requested KVs in Storage

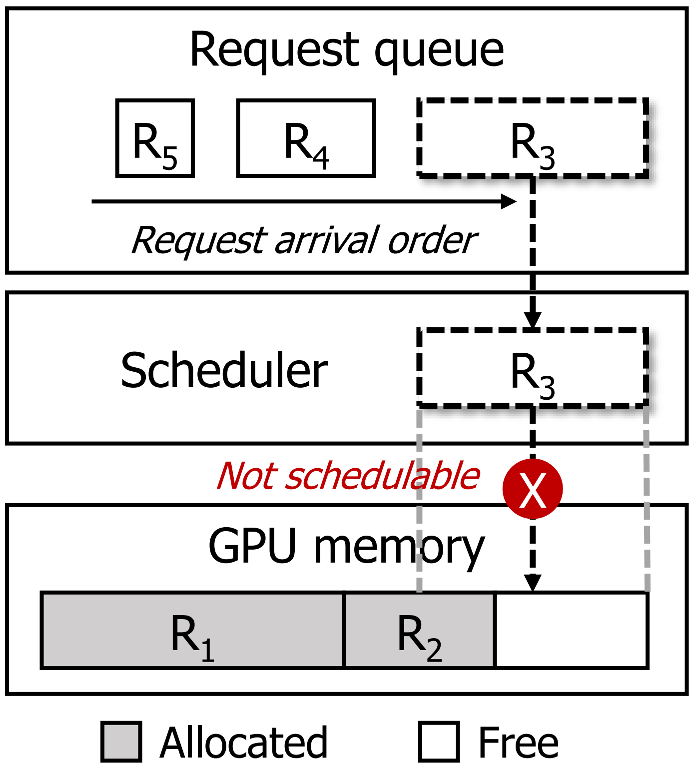
- If the staging phase cannot be hidden (i.e., waiting reqs < 1)
  - Recompute instead of retrieving from storage due to bandwidth limits



*subscription means *session ID*

Head-of-Line Blocking

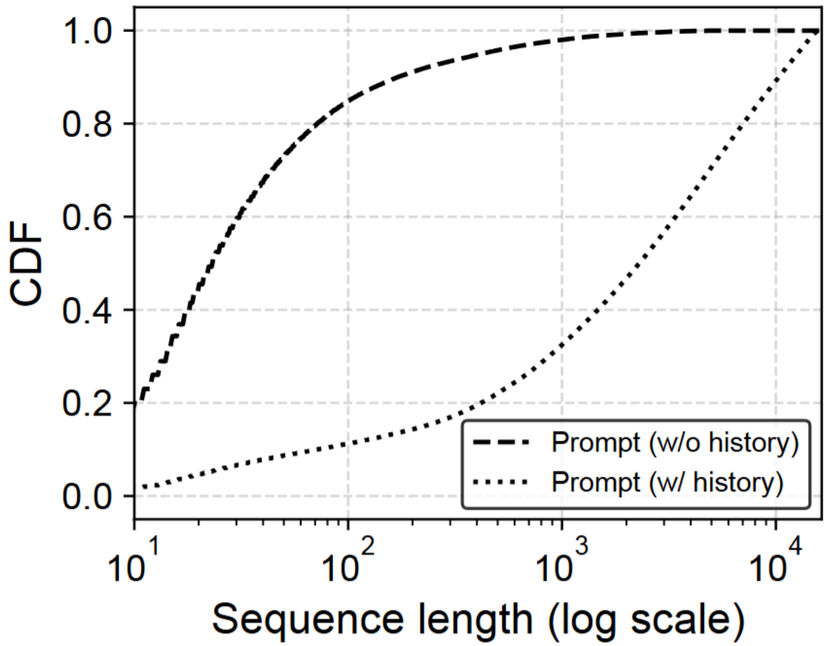
- FCFS scheduling causes the head-of-line blocking problem

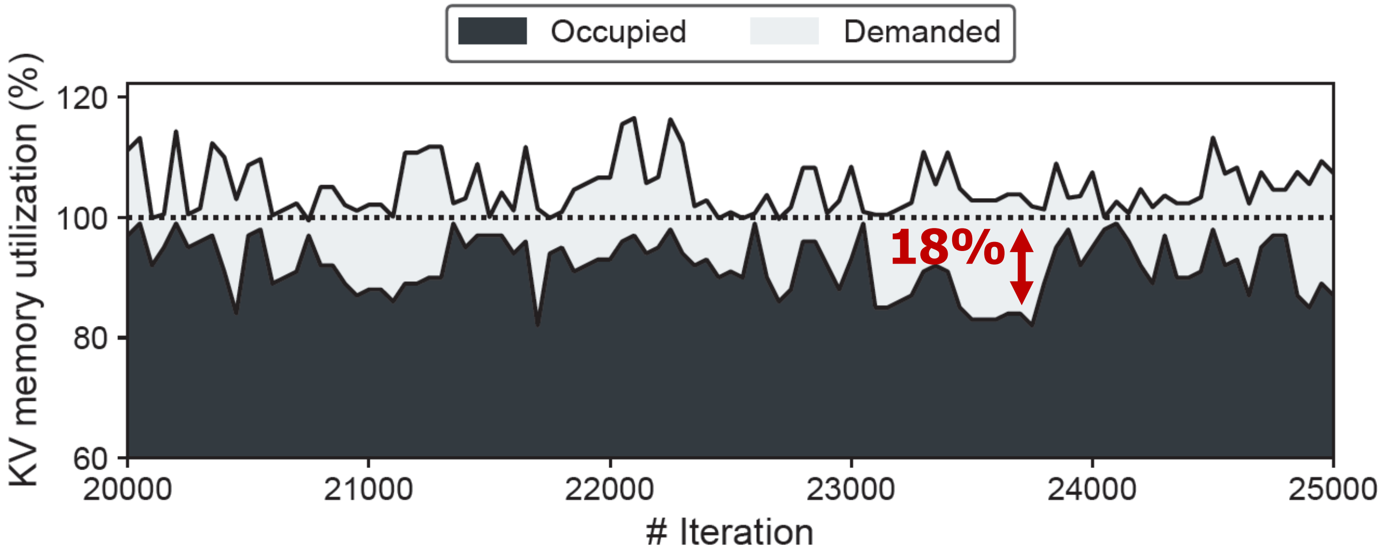


Amplified Prompt Length in Multi-turns

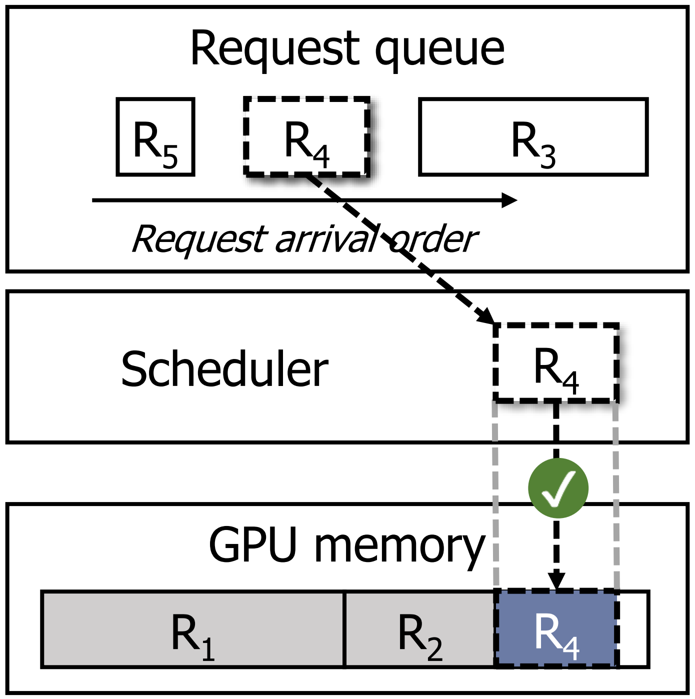
- Multi-turn dialogues amplify the prompt length of user queries
  ➔ Leading to underutilization of GPU memory

Dataset: ShareGPT





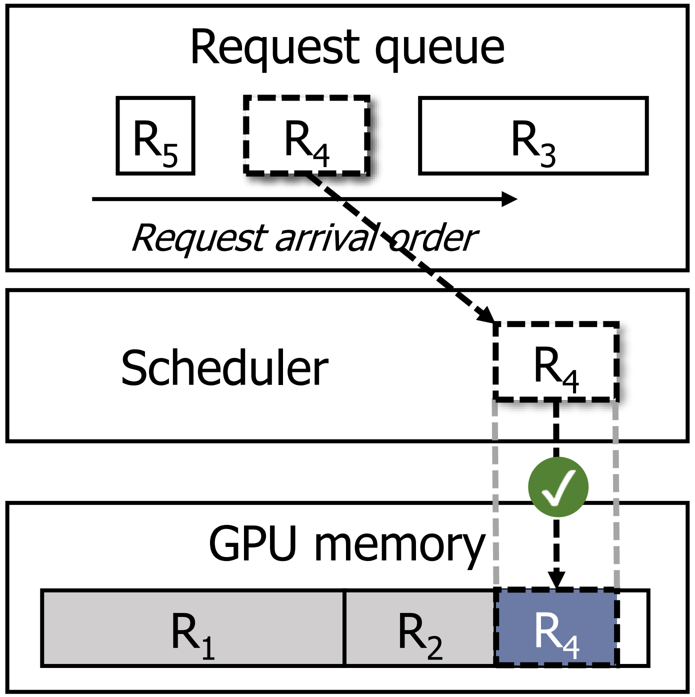
FlashGen-Sched: Request Reordering



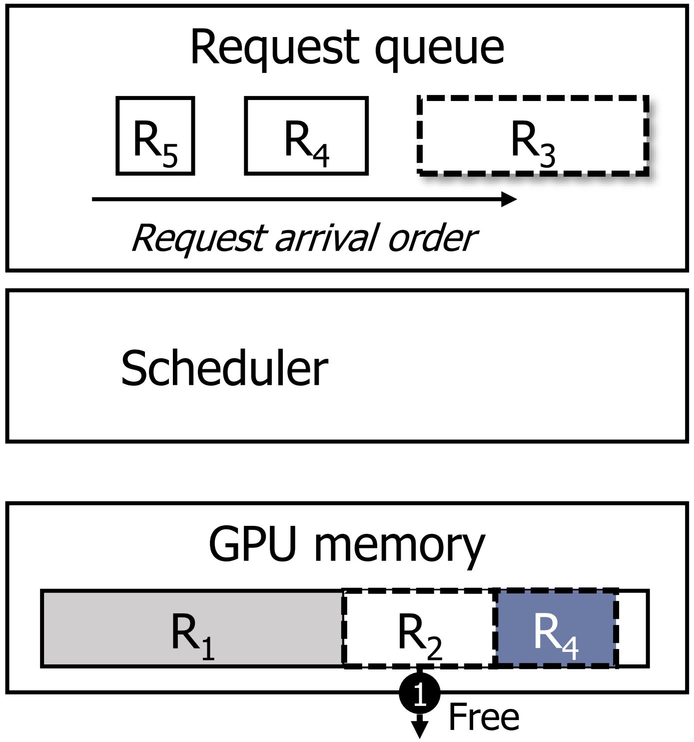
Reordering execution

FlashGen-Sched: Starvation-free Scheduling

- Upon R2 is completed, the occupied space is freed



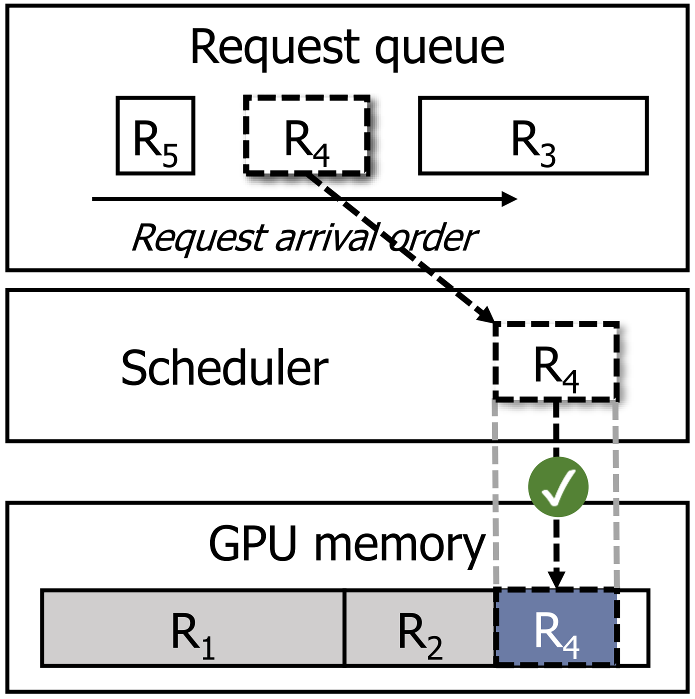
Reordering execution



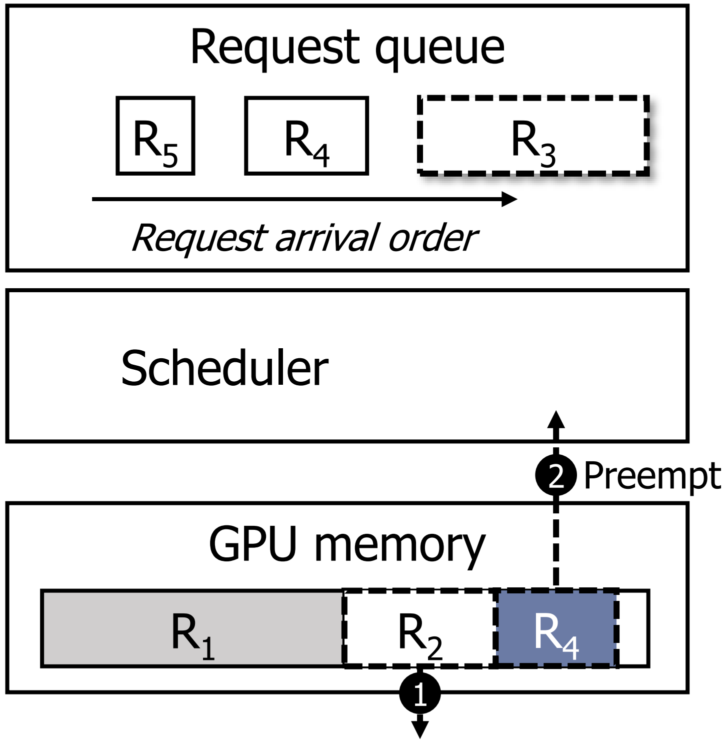
Starvation-free scheduling

FlashGen-Sched: Starvation-free Scheduling

- If the total memory of the promoted request and the remaining free space is enough for R3, preempt the prompted request



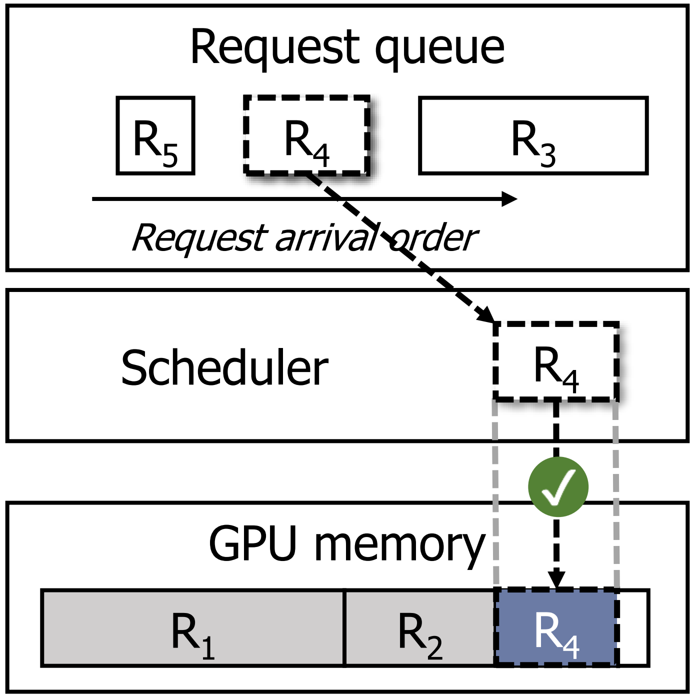
Reordering execution



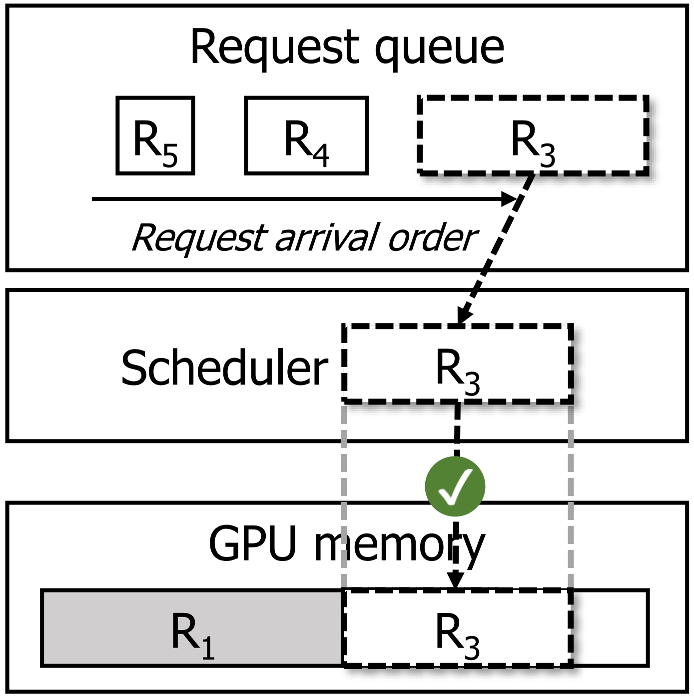
Starvation-free scheduling

FlashGen-Sched: Starvation-free Scheduling

- Once preemption is completed, dispatch R3



Reordering execution



Starvation-free scheduling

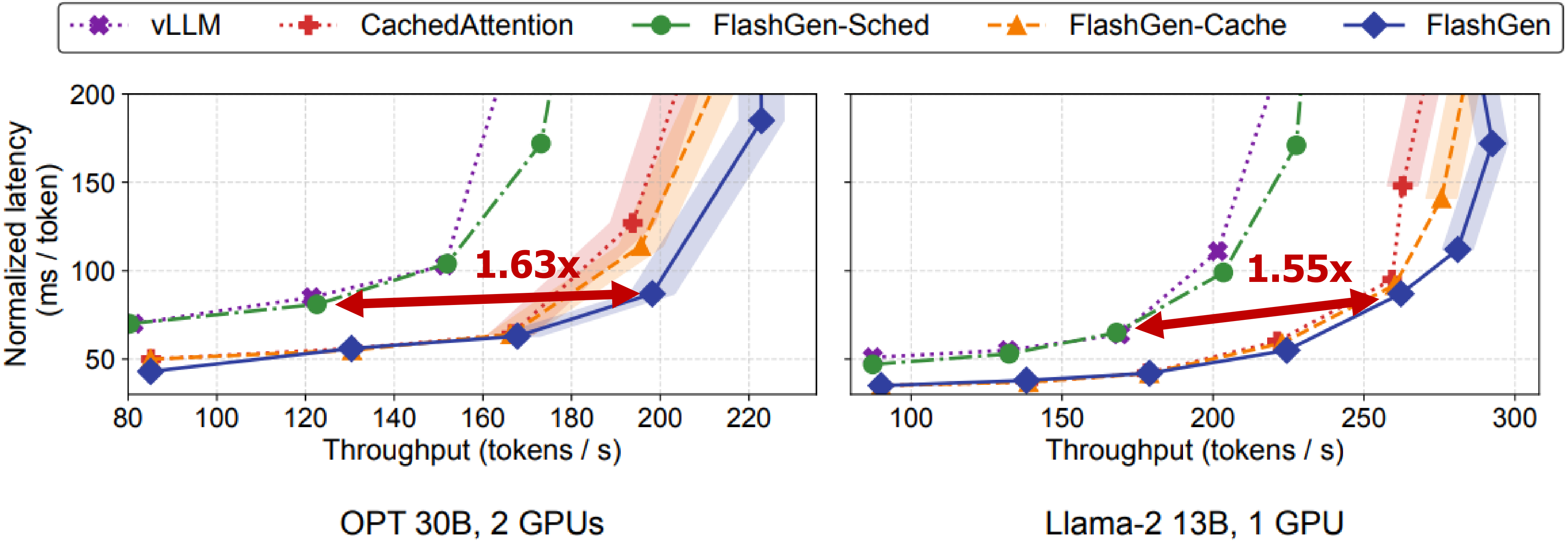
Evaluation Setup

<table>
<tr><td rowspan="4">Hardware Setup</td><td colspan="2">Azure instance: standard_NC48ads_A100_v4</td></tr>
<tr><td>GPUs</td><td>A100 (80GB) x 2ea</td></tr>
<tr><td>DRAM</td><td>440GB -> use 224 GB (50%) for caching KVs</td></tr>
<tr><td>Storage</td><td>NVMe SSD: 960GB x 2ea (RAID-0)</td></tr>
<tr><td>Comparison</td><td colspan="2">vLLM, *CachedAttention, FlashGen-Sched, FlashGen-Cache, FlashGen</td></tr>
<tr><td rowspan="3">Workloads</td><td>Dataset</td><td>ShareGPT</td></tr>
<tr><td rowspan="2">Models</td><td>OPT: 13B, 30B</td></tr>
<tr><td>Llama-2: 13B, 70B</td></tr>
</table>

* B. Gao et al. Cost-Efficient Large Language Model Serving for Multi-turn Conversations with CachedAttention (ATC'24)

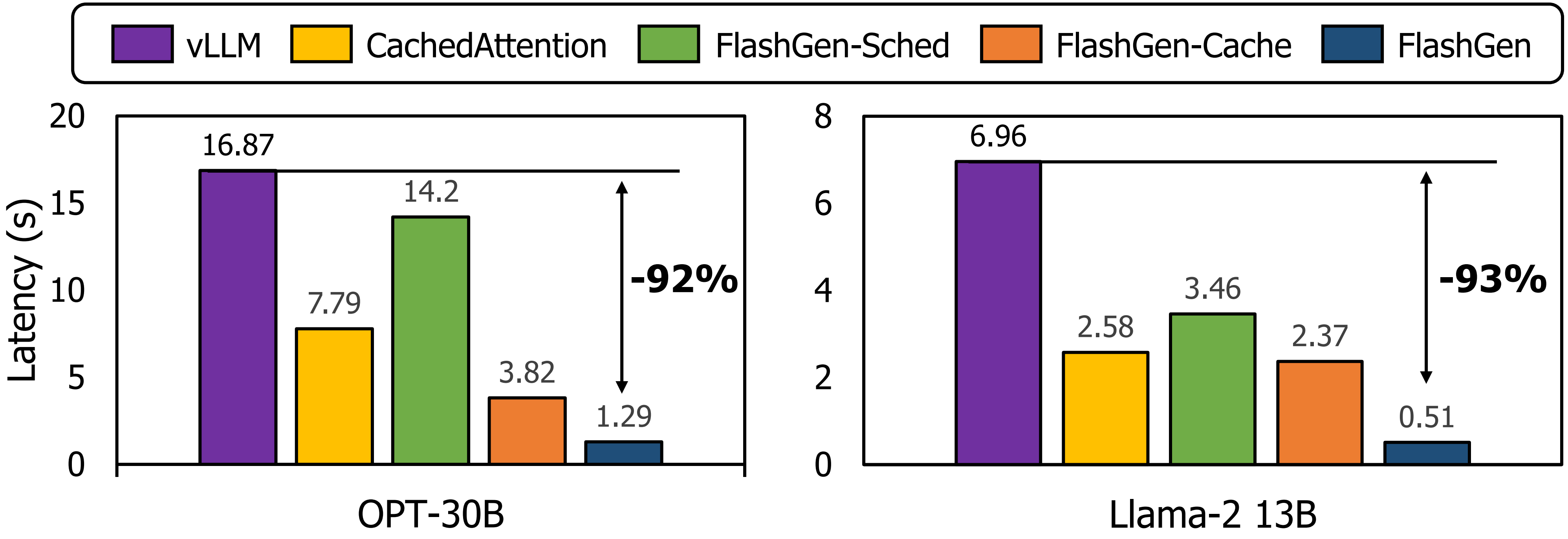
End-to-End Latency & Throughput

Shaded points indicate SSD involvement



All **FlashGen** schemes outperform **vLLM** in both latency & throughput

P95 Time To First Token (TTFT)



FlashGen drastically improves the responsiveness

Effectiveness of FlashGen-Sched

- FlashGen-Sched increases memory utilization
  - ➔ Allows for batching more requests, leading to higher throughput

	OPT-13B	OPT-30B	Llama-2 13B	Llama-2 70B	Average
vLLM	90.44	88.80	91.669	91.70	90.65
FlashGen-Sched	96.99	95.71	98.426	95.21	96.58

(a) Average GPU memory utilization

	OPT 13B	OPT 30B	Llama-2 13B	Llama-2 70B
FlashGen-Sched	1.15x	1.15x	1.06x	1.06x

(b) Increase in the average number of batched requests

Conclusion

- **Problem**
  - Existing LLM frameworks are inefficient in serving multi-turn dialogues
  - Increasing conversation turns lead to larger attention KV contexts and longer prompts
- **Solution:**
  - **Multi-level caching** stores attention KVs in GPU, CPU, and SSD to minimize recomputation
  - **Request reordering** improves GPU memory efficiency and reduces waste
- **Result**
  - **FlashGen** achieves 1.63x better throughput while in a similar latency boundary

Thank You!

Accelerating LLM Serving for Multi-turn Dialogues with Efficient Resource Management

Jinwoo Jeong
jwjeong@csl.korea.ac.kr

Jeongseob Ahn
jsahn@csl.korea.ac.kr

KOREA UNIVERSITY