Hewlett Packard
Enterprise

LLM Serving With Efficient KV-Cache Management Using Triggered Operations

Aditya Dhakal, Pedro Bruel, Gourav Rattihalli,
Sai Rahul Chalamalasetti and Dejan Milojicic

May DD, 2024

Large Language Models : Introduction

- Large Language Models (LLMs) are widely used now for applications such as Chatbot, Knowledge summary, etc.

- Popular LLM model such as LLaMA, GPT, etc generate response based on *user's prompts* and *context of the conversation*

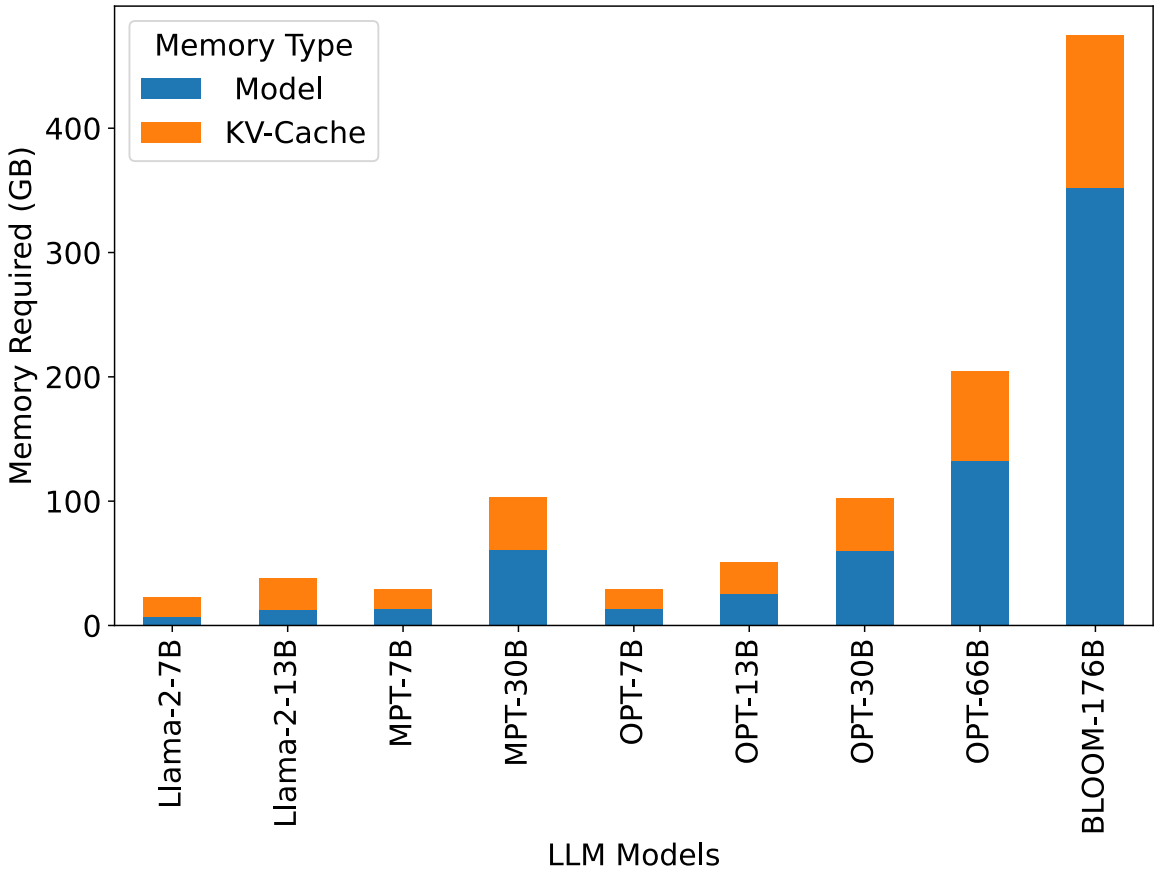
- State-of-the-art LLM models are very large in terms of memory and compute requirements
  - 70 billion parameters LLAMA models require 8 GPU clusters

- GPUs are expensive (estimates for March, 2024)

GPU Model	Cost GPU/hr (USD)	8X GPUs/hr
NVIDIA A100	$2.21	**$17.86**
NVIDIA H100	$4.76	**$38.80**

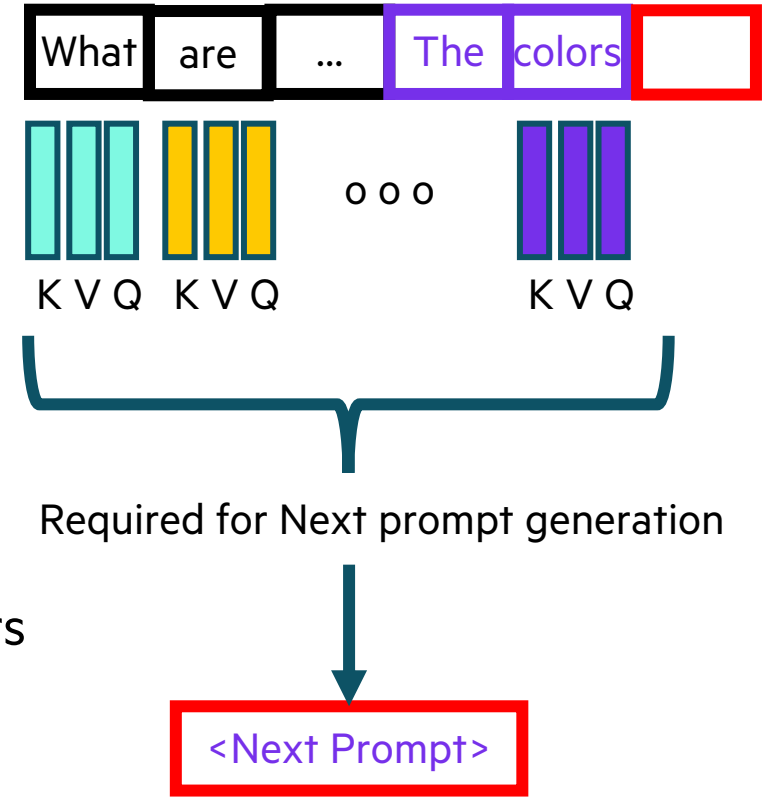
Multiplexing the GPUs

- Sharing the GPUs to run multiple user's queries will increase GPU utilization
- However, LLMs also requires large amount of memory just to operate
- One of the component for increased memory consumption is Key-Value cache (KV-cache)



Key Value (KV) Cache

- Key-value (KV) in LLM context is different from ***Key-value pairs***
- Key value vectors are created in *attention layer* in the generation phase
- Example:
  - Query: ***"What are the colors in the Rainbow?"***
  - Generated prompt: **"The colors …"**
- Here, for the computation of next token after *"The colors"* uses KV vectors of previous tokens:
  - ***"What are the colors in the Rainbow? The colors"***
- These KV vectors are invariant for same tokens. Thus, caching the KV vectors will greatly reduce the computation
  - KV-caches allow linear scaling



Why is KV-cache important?

- Lowers the computation for inference during **generation phase**
- Why save it then?
  - **Preserves the context of the conversation**
- We present an example
- KV-cache can also be a security issue

User: Please provide me with a recipe for making cheesecake.

LLM: For the cheesecake... 350◦F...

User: Only provide temperature in °C from now on

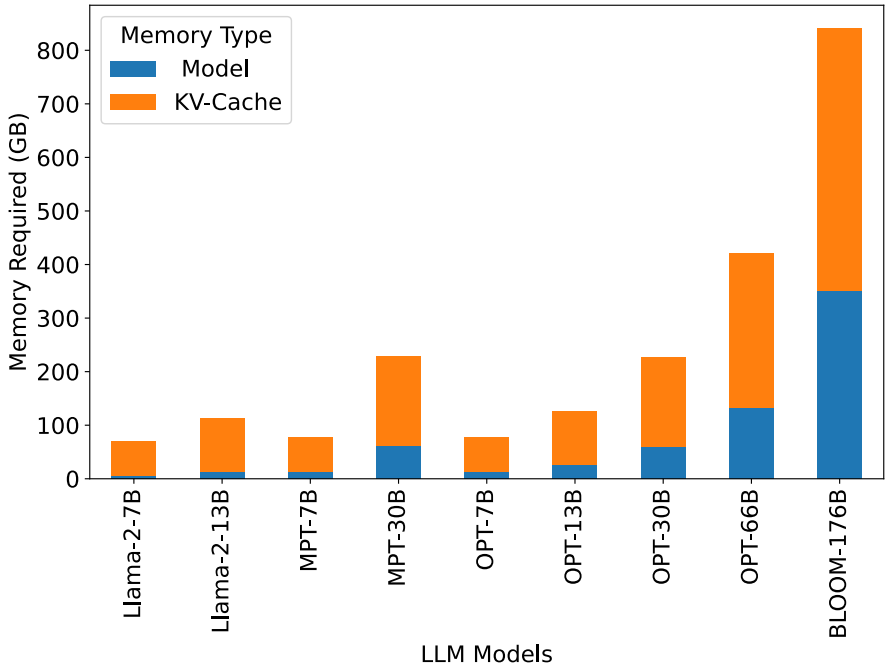
LLM: OK.

...

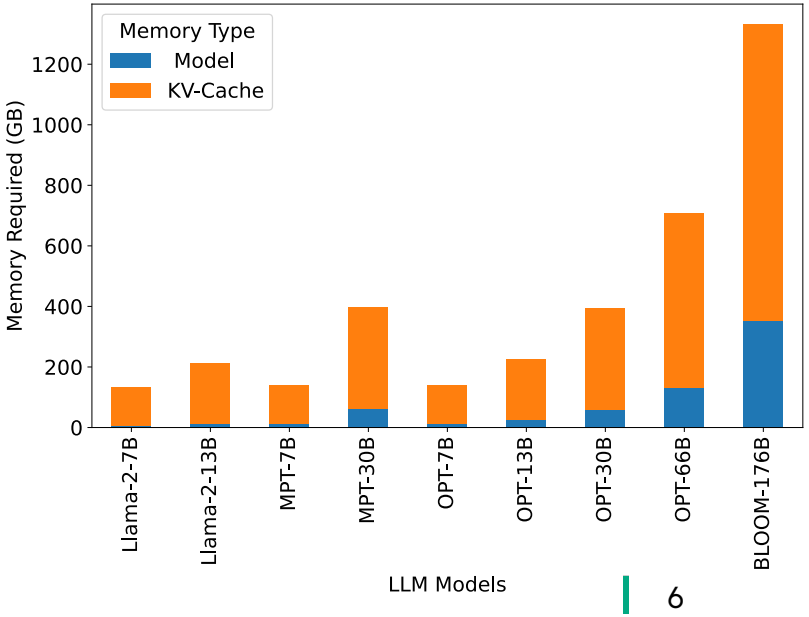
User: What temperature should I set the oven on?

Key Value Size

- KV-caching is compute-storage tradeoff
  - KV-cache size can easily exceed model size
- GPU memory is still limited (Max size: 192 GB)
- Swapping the KV-cache out of GPU
  - another user's queries to be processed (multiplexing)
- Using CPU to transfer data out of GPU to storage node increases latency
  - Use of CPU increases processing, data movement, network stack and context switching overhead



KV-cache with 4K token size



KV-cache with 8K token size

Triggered Operations for Data Transfer

- **Triggered operations** lets application enqueue data transfer requests in the NIC
  - Origin from Quadrics network
- Defer the transfer to future event (usually a counter or a flag called “Trigger”)
- HPE Cassini NICs provide trigger operations
  - Currently supported with libfabric communication library
- Application must assign the memory buffer to transfer
- Trigger operations do not involve the CPU other than initial trigger setup

fi_trigger(3) Libfabric Programmer's Manual

NAME

fi_trigger - Triggered operations

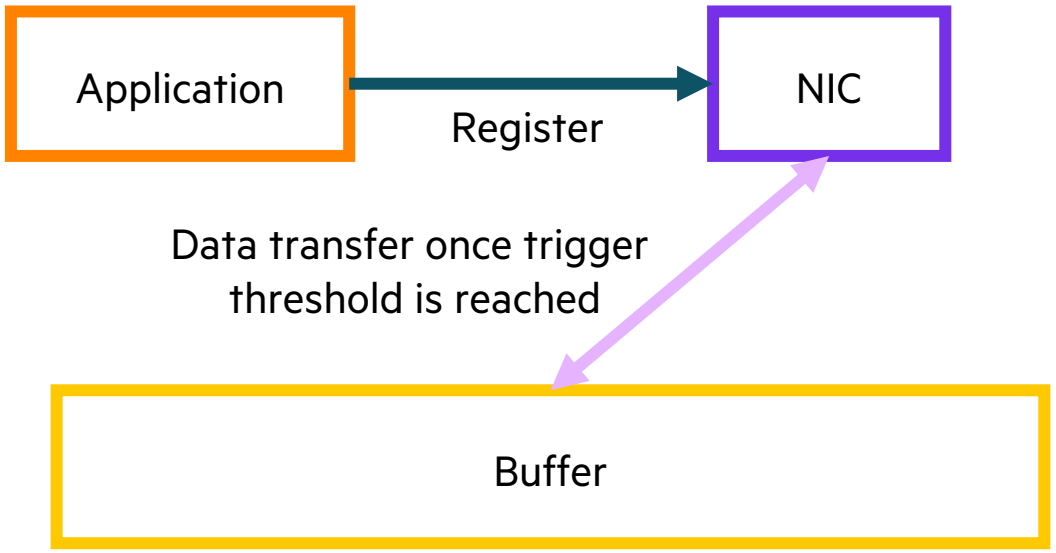
SYNOPSIS

```
#include <rdma/fi_trigger.h>
```

DESCRIPTION

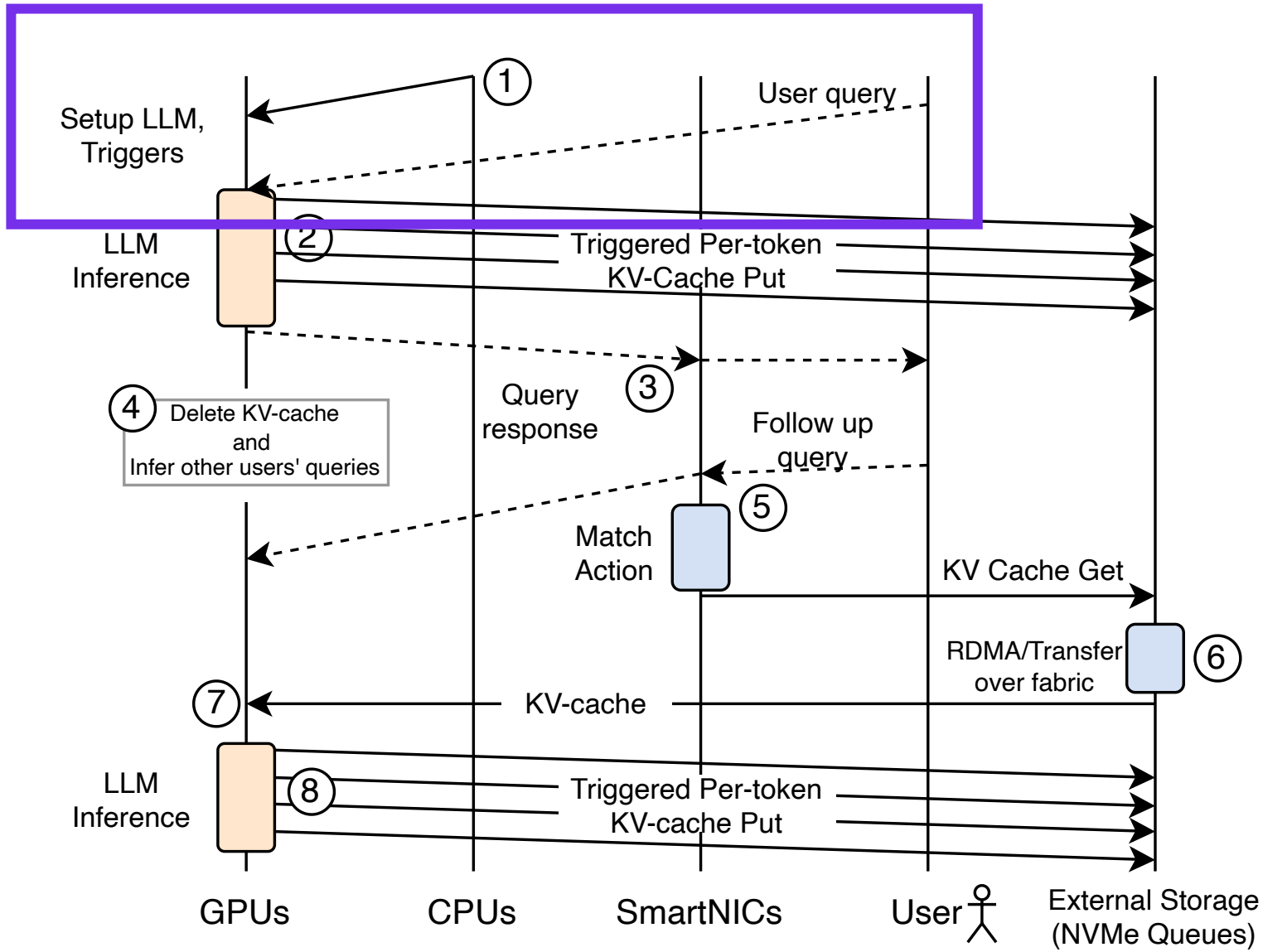
Triggered operations allow an application to queue a data transfer request that is deferred until a specified condition is met. A typical use is to send a message only after receiving all input data.

Libfabric description of trigger ops support
source: https://ofiwg.github.io/libfabric/v1.9.1/man/fi_trigger.3.html



Our Workflow

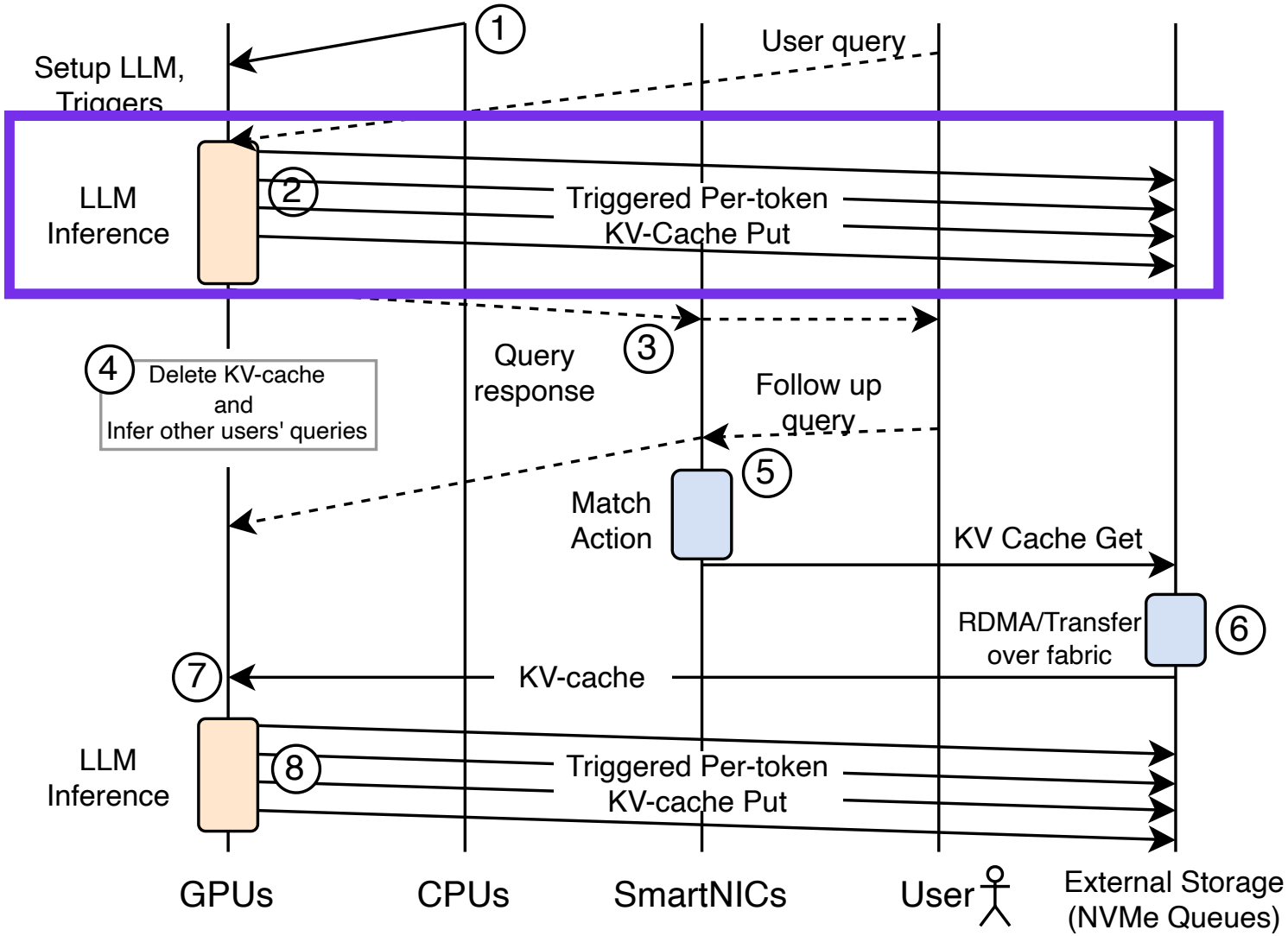
1. User sets up the LLM and triggers
   - User sends their query for inference



Our Workflow

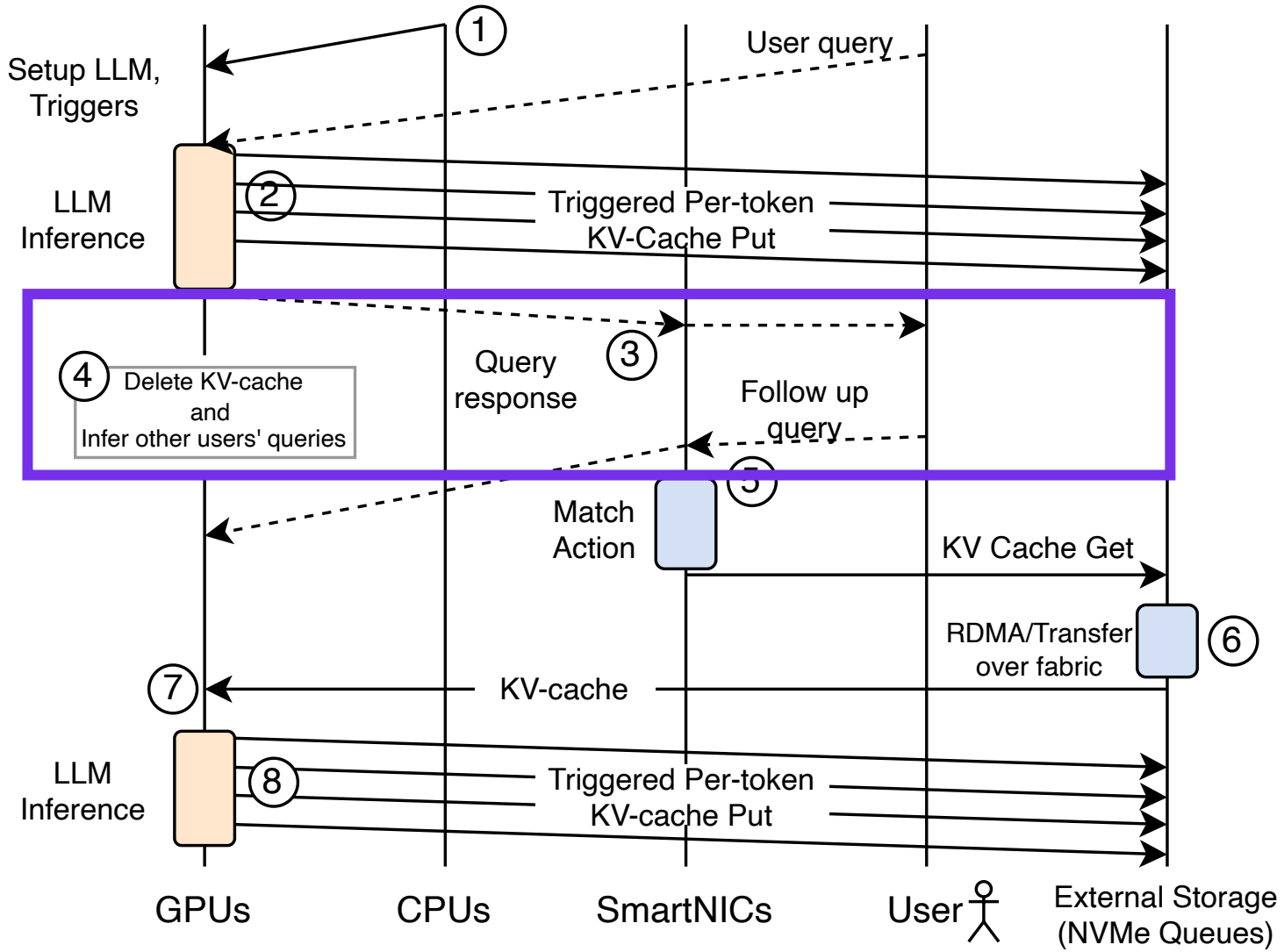
1. User sets up the LLM and triggers
   - User sends their query for inference

2. LLM inference: query response is generated
   - The newly generated KV-Cache is transferred out of GPU by triggered operations



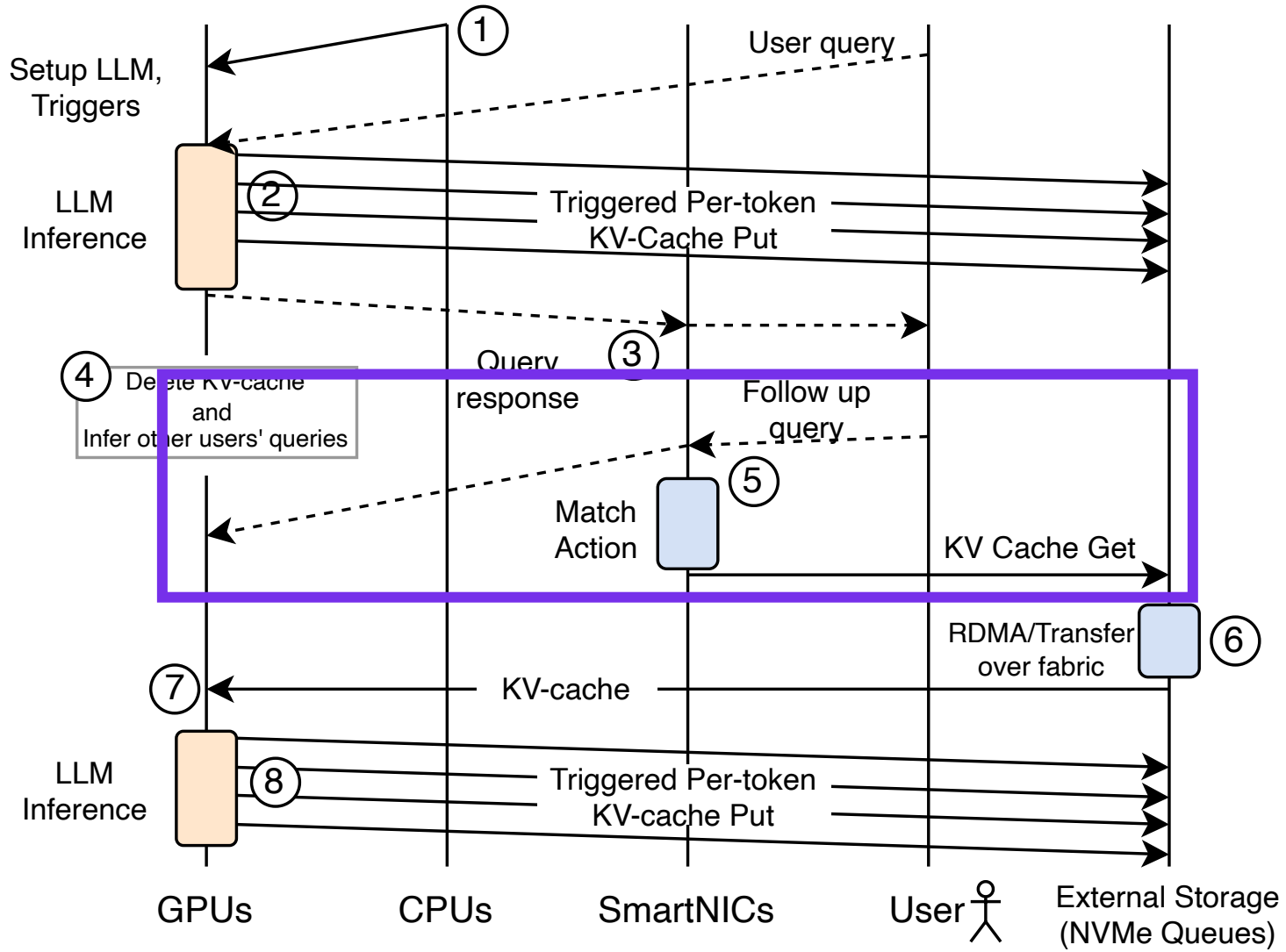
Our Workflow

1. User 1 sets up the LLM and triggers
   - User 1 sends their query for inference
2. LLM inference: query response is generated
   - The newly generated KV-Cache is transferred out of GPU by triggered operations
3. User 1 receives query response
4. Delete KV-cache and infer user 2's request



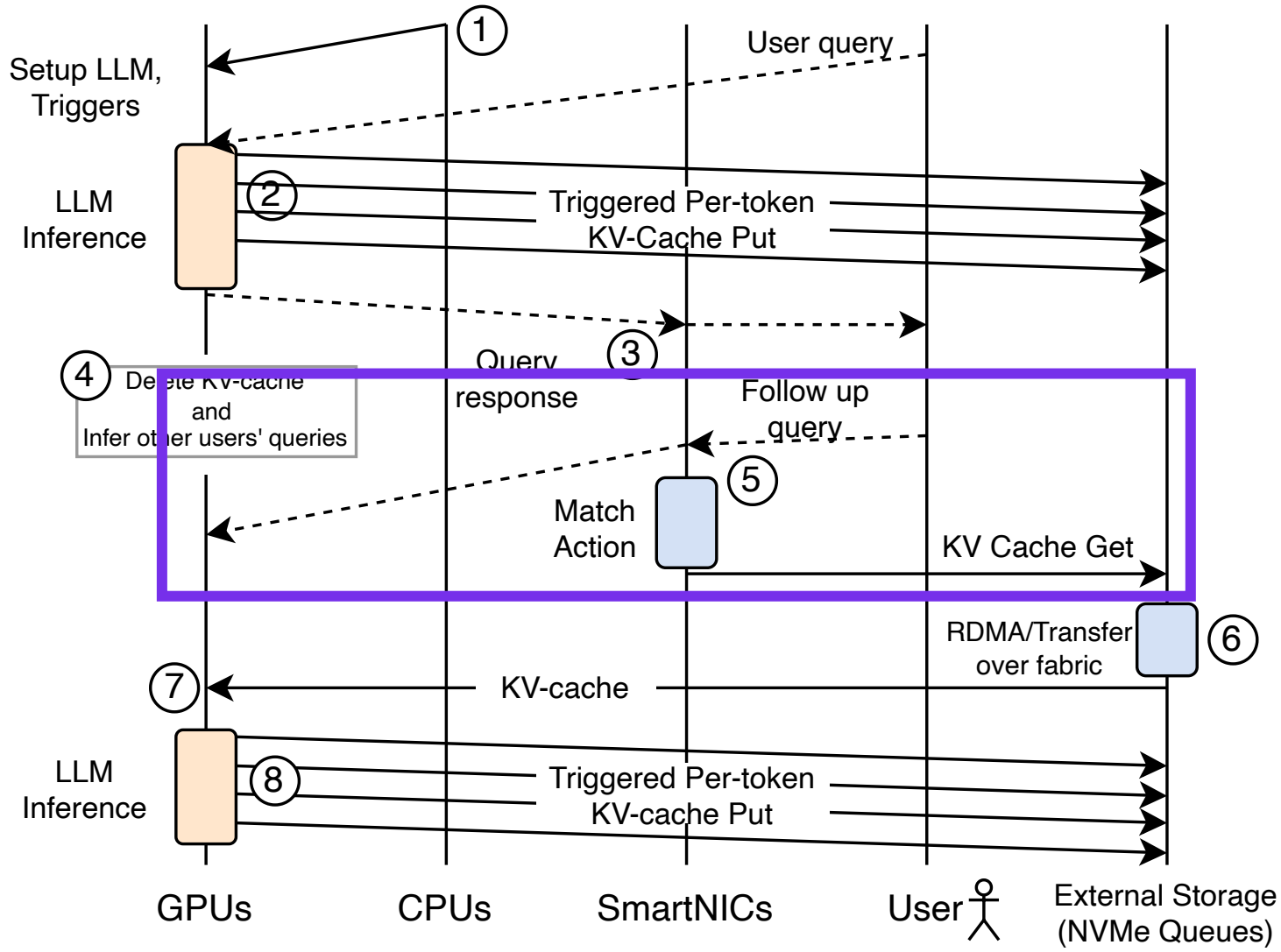
Our Workflow

1. User 1 sets up the LLM and triggers
   - User 1 sends their query for inference
2. LLM inference: query response is generated
   - The newly generated KV-Cache is transferred out of GPU by triggered operations
3. User 1 receives query response
4. Delete KV-cache and infer user 2's request
5. User 1 sends follow up request



Our Workflow

5. User 1 sends follow up request
   - The match-action module in SmartNIC will match the user request
   - The NIC will then fetch the KV-cache for the User 1 from storage

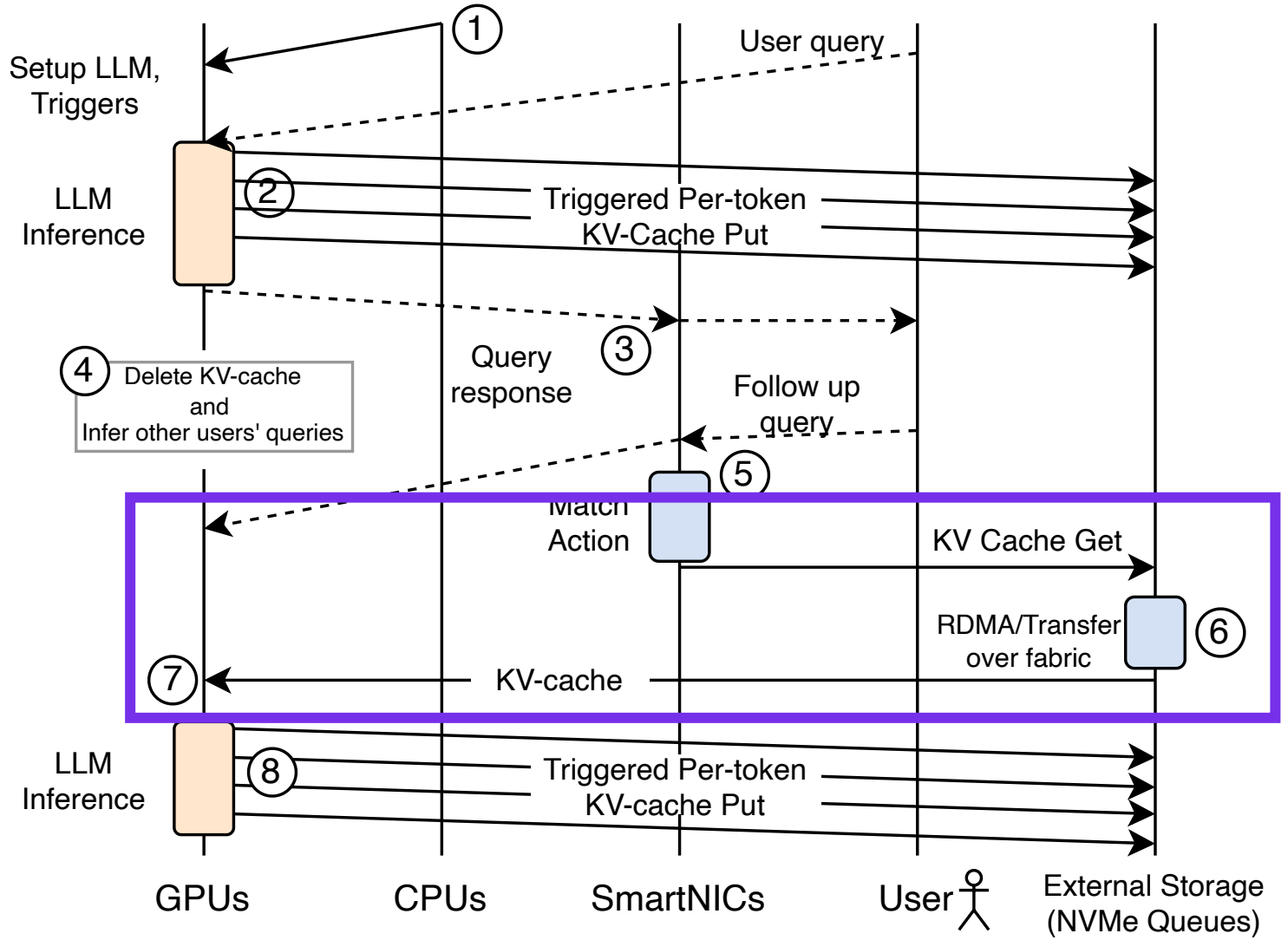


Our Workflow

5. User 1 sends follow up request
   - The match-action module in SmartNIC will match the user request
   - The NIC will then fetch the KV-cache for the User 1 from storage

6. The storage side smartNIC will transfer the data via RDMA to the compute accelerators

7. The accelerators receive the KV-cache



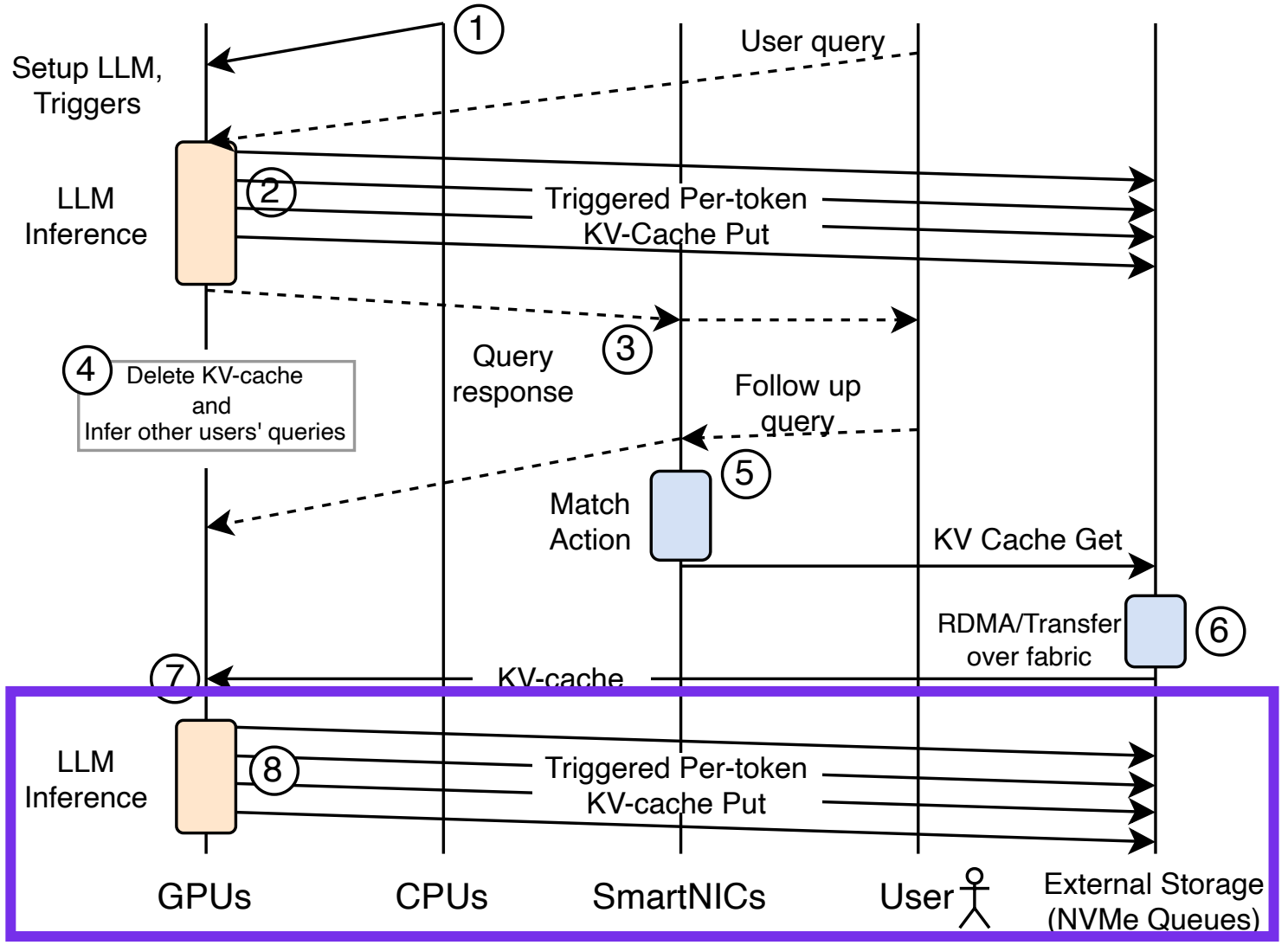
Our Workflow

5. User 1 sends follow up request
   - The match-action module in SmartNIC will match the user request
   - The NIC will then fetch the KV-cache for the User 1 from storage

6. The storage side smartNIC will transfer the data via RDMA to the compute accelerators

7. The accelerators receive the KV-cache

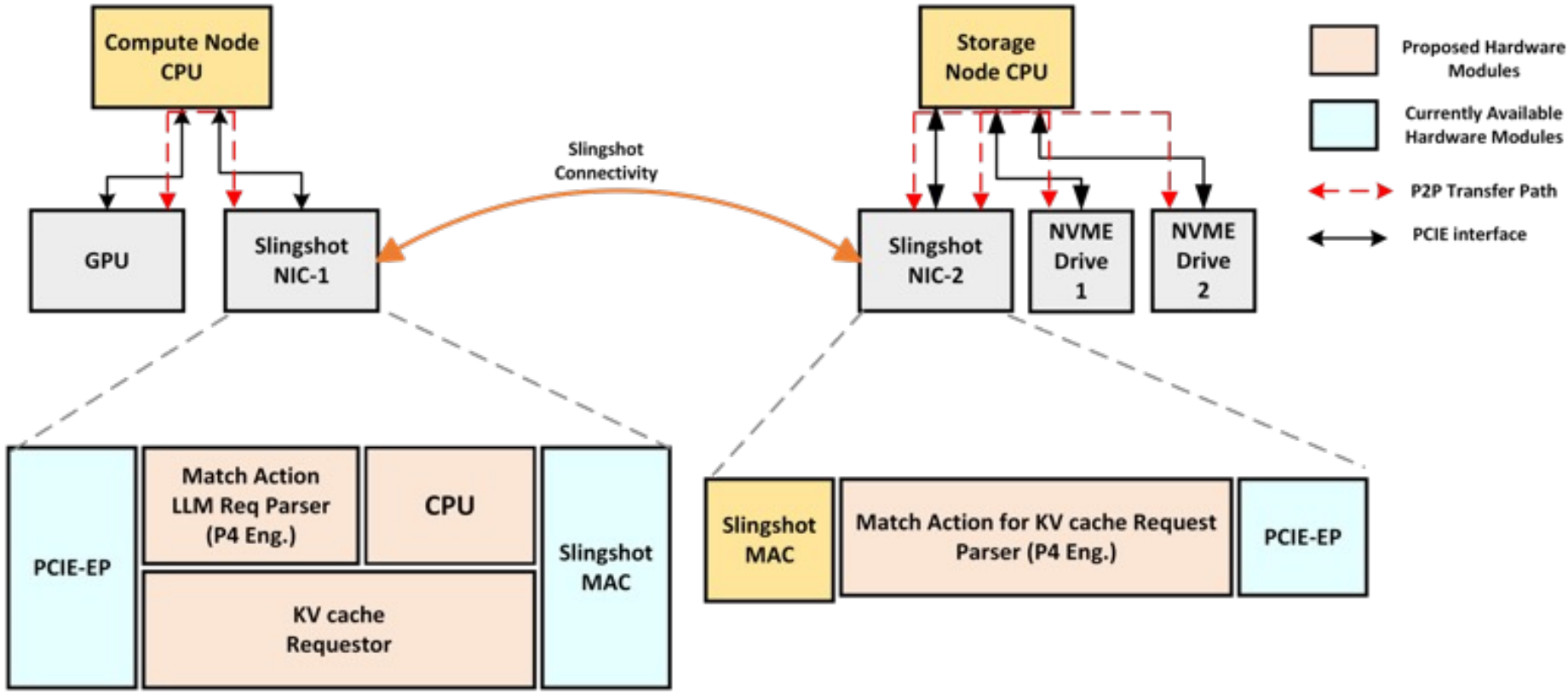
8. LLM inference is conducted on new query and KV-cache is again stored with triggered ops



System Architecture

- Hardware/software components in SmartNICs to be able to transfer KV-cache to remote storage node

- We divide requirements for compute and storage node



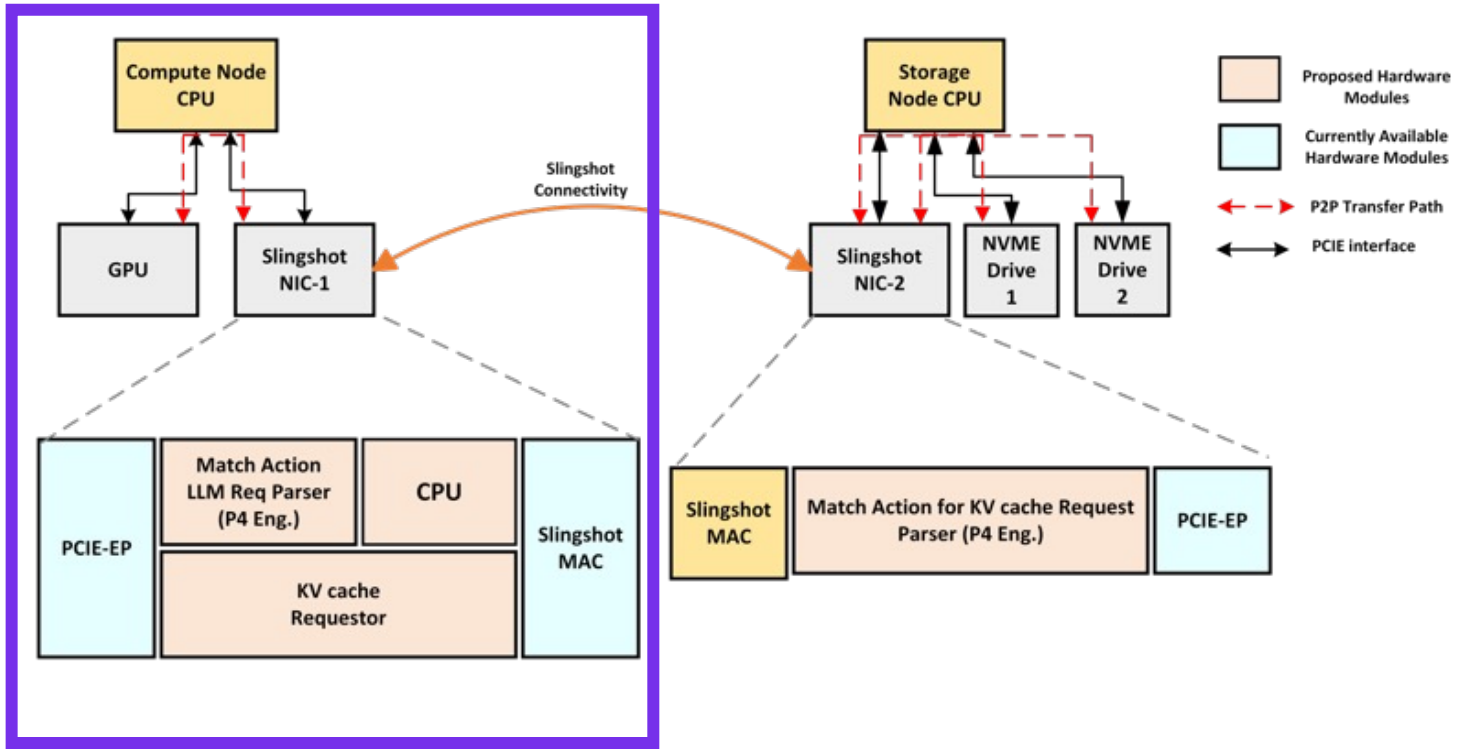
System Architecture

- **Compute Node**

- **Match Action Module:** matches incoming request and looks up where the KV-cache is located

- **KV-Cache Requestor**: Requests KV-cache from multiple storage nodes

- **CPU:** SmartNIC CPU to facilitate Match Action and KV-cache requestor

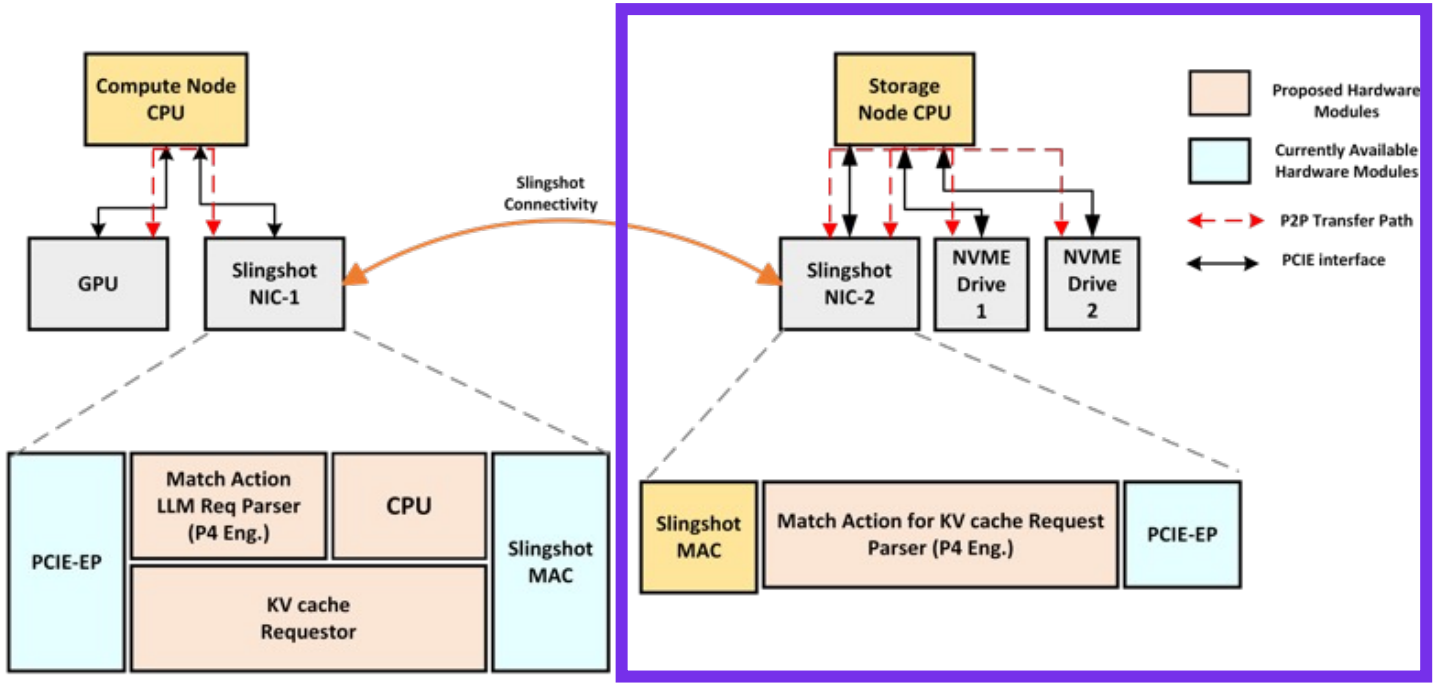


System Architecture

- **Storage Node**

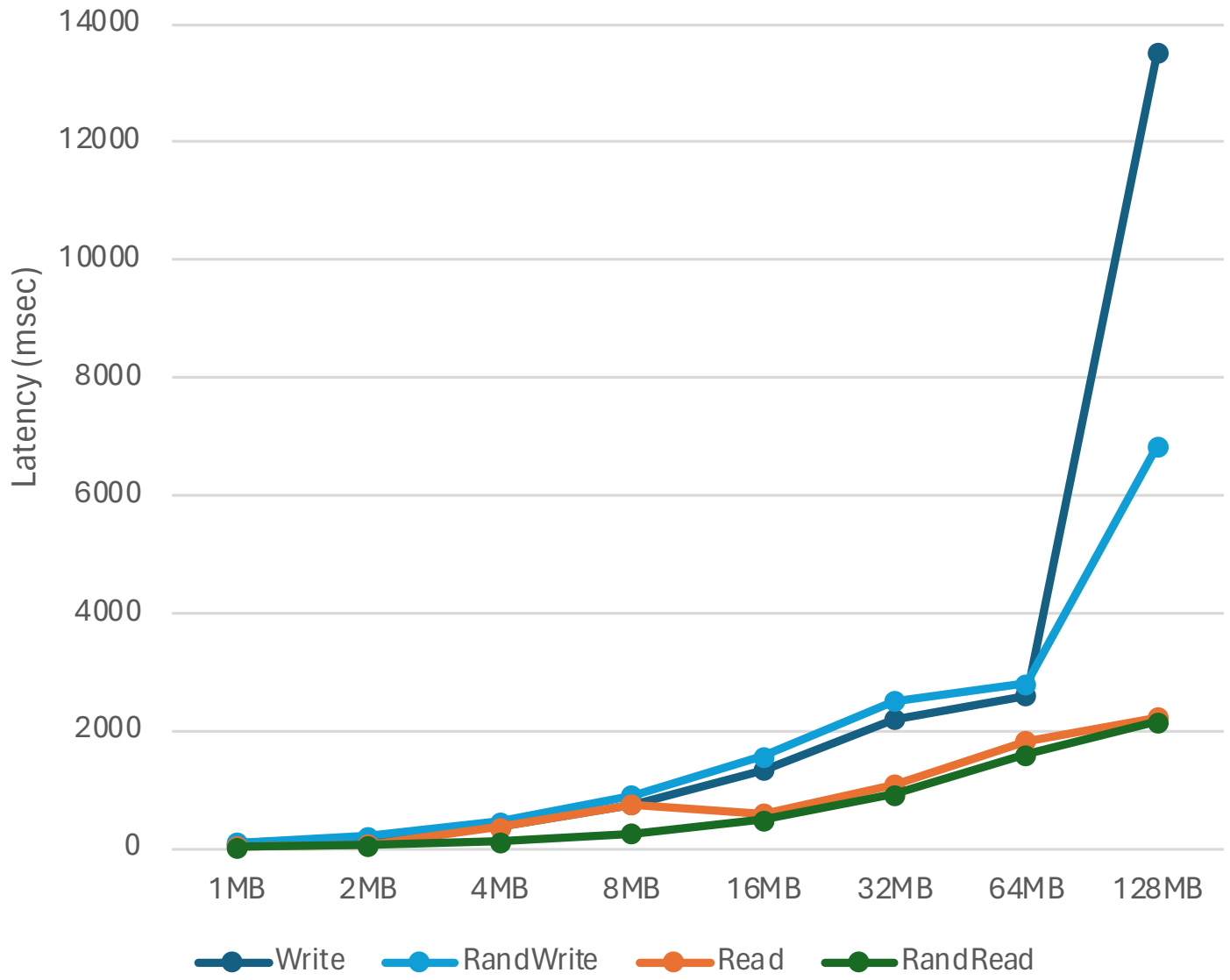
- **Match Action Module:** matches incoming KV-cache request

- **NVMe P2P transfer module**: Peer-to-peer transfer software support for SmartNIC



NVMe Measurement Results

- Measured read and write latency to NVMe drive from same socket CPU
- We want to populate our model with realistic values
- We see the RandRead is about same as Read
- Writes are however, much slower than read and latency increasing with larger data sizes (128MB)
- This influenced our decision to store chunks of KV-cache at a time than whole KV-cache



Simulation Parameters

- To create our data transfer model, we benchmarked relevant hardware and chose the values in Table II and Table III
- We used bandwidth and latencies value for different hardware in the system
- We simulated the results for 8 H100 GPU and 1 NIC per GPU.
- We modeled the link between compute and storage server with a 100 Gbps connection

TABLE II: Bandwidth parameters used in simulation, where m is the message size in bytes.

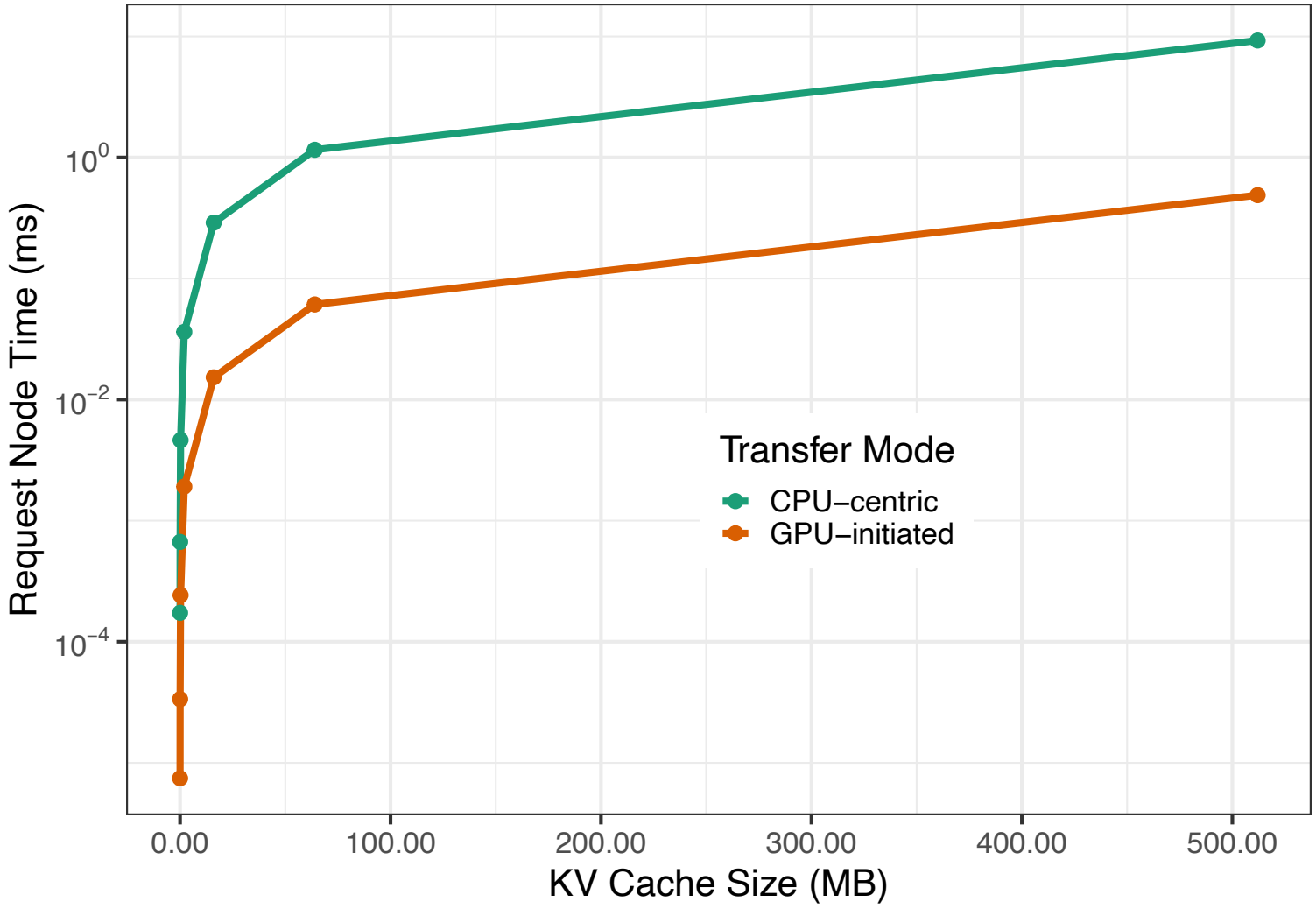
Parameter	Value $\left(\frac{1}{\text{bytes/s}}\right)$	Description
β_{PCIE}	8.12×10^{-12}	PCIE bandwidth
β_{VRAM}	4.55×10^{-13}	GPU memory bandwidth
β_{CPU}	4.55×10^{-13}	CPU estimated bandwidth
β_{DRAM}	1.21×10^{-12}	Node memory bandwidth
$\beta_{\text{NIC\_serial.}}$	5.13×10^{-11}	NIC serialization bandwidth
$\beta_{\text{NIC\_mem}}$	1.43×10^{-11}	NIC memory bandwidth
$\beta_{\text{NIC\_alu}}$	5.22×10^{-12}	NIC ALU bandwidth
$\beta_{\text{NVMe\_write}}$	$\frac{1}{2.34 \times 10^{9} + 6.83m}$	NVMe write bandwidth model
$\beta_{\text{NVMe\_read}}$	$\frac{1}{5.22 \times 10^{9} + 19.4m}$	NVMe read bandwidth model

TABLE III: Latency parameters used in simulation, where m is the message size in bytes.

Parameter	Value (s)	Description
λ_{PCIE}	3.2×10^{-8}	PCIE latency
$\lambda_{\text{NVMe\_write}}$	$3.3 + 4.10 \times 10^{-7}m$	NVMe write latency model
$\lambda_{\text{NVMe\_read}}$	$2.03 + 2.63 \times 10^{-7}m$	NVMe read latency model
λ_{network}	1.21×10^{-12}	Network latency

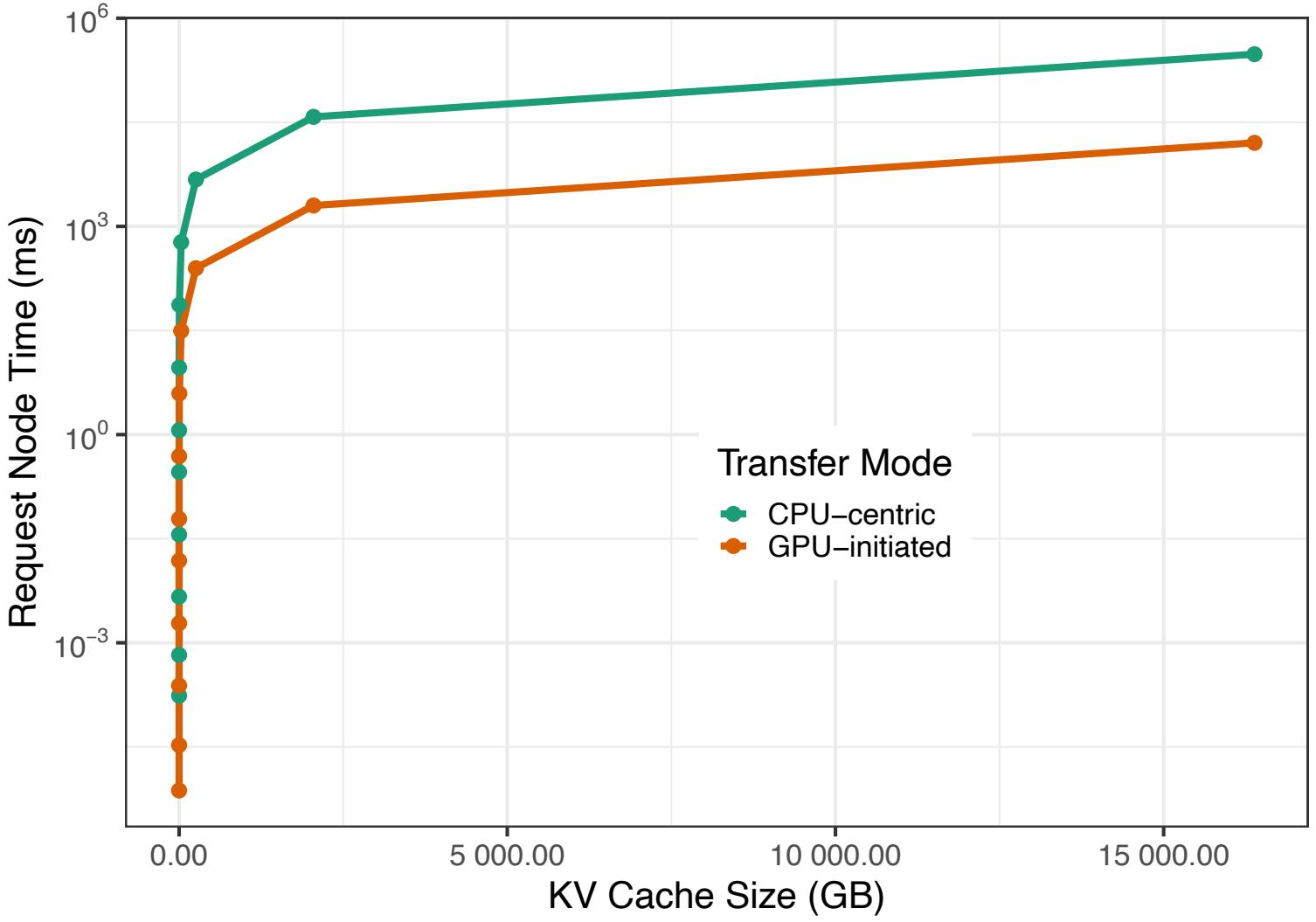
Simulated Results

- GPU-initiated (Trigger ops) transfer shows much lower latency at smaller data size



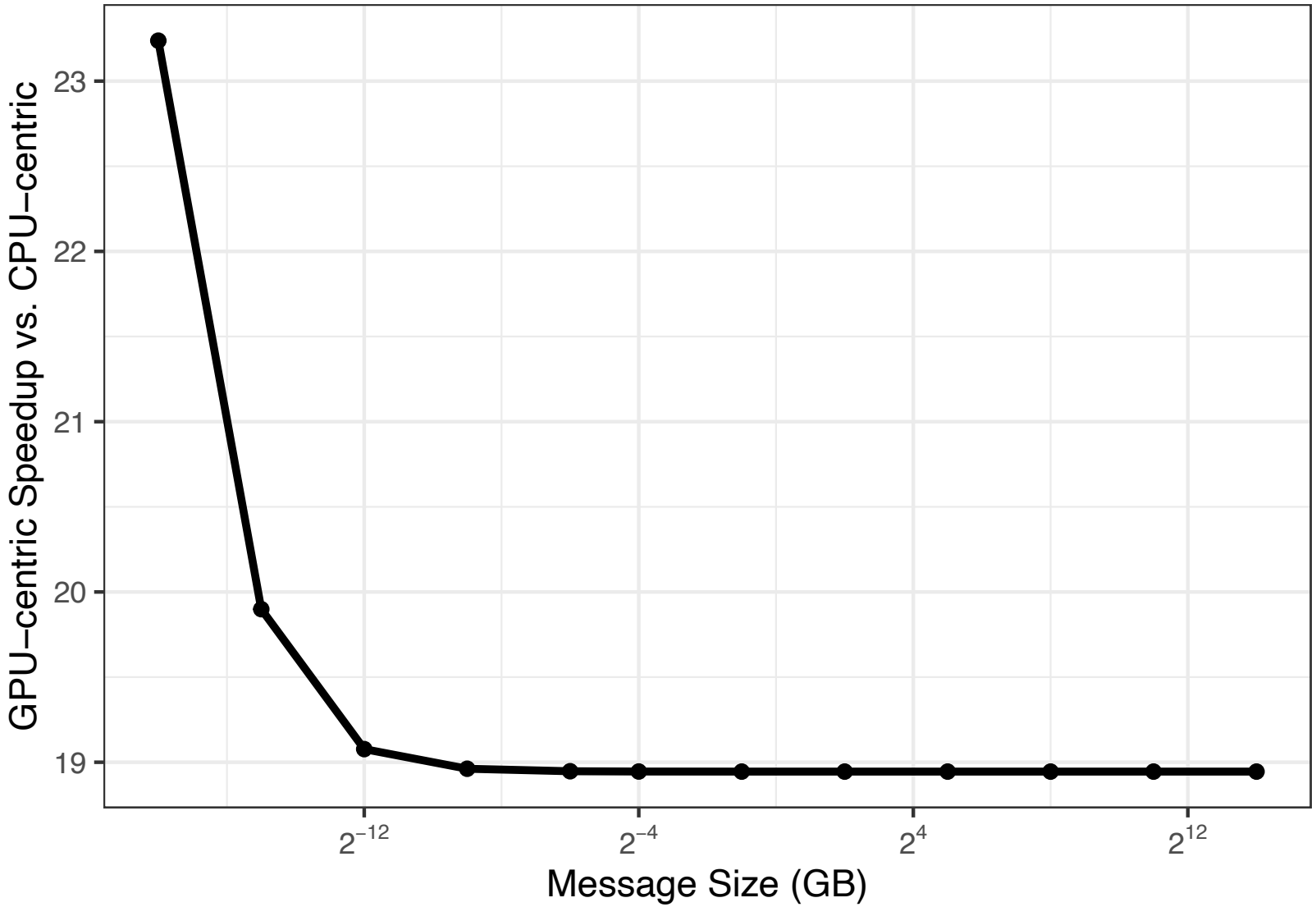
Simulated Results

- GPU-initiated (Trigger ops) transfer shows lower latency at higher and more practical data sizes as well



Speedup

- We get 19x speed up with Triggered-ops data transfer vs. CPU-centric data transfer



Next Steps

- Current results are simulation result and only show improvement over data transfer
- We are adapting vLLM and Deepspeed Inference to have end to end results
  - vLLM and Deepspeed Inference already manger KV-Cache within GPUs
- Working with triggered operations with Cassini NICs and libfabric application for our workflow

Conclusion

- Proposed system that will transfer KV-Cache generated during LLM inference to storage
- Swap the KV-cache back when user sends follow up response
- Use of triggered operation to remove CPU overhead
- Future work on implementing the system to LLM inference platforms

Thank you

Aditya Dhakal
aditya.dhakal@hpe.com