Inference in the Age of Reasoning Models

Dr. Séverine Habert

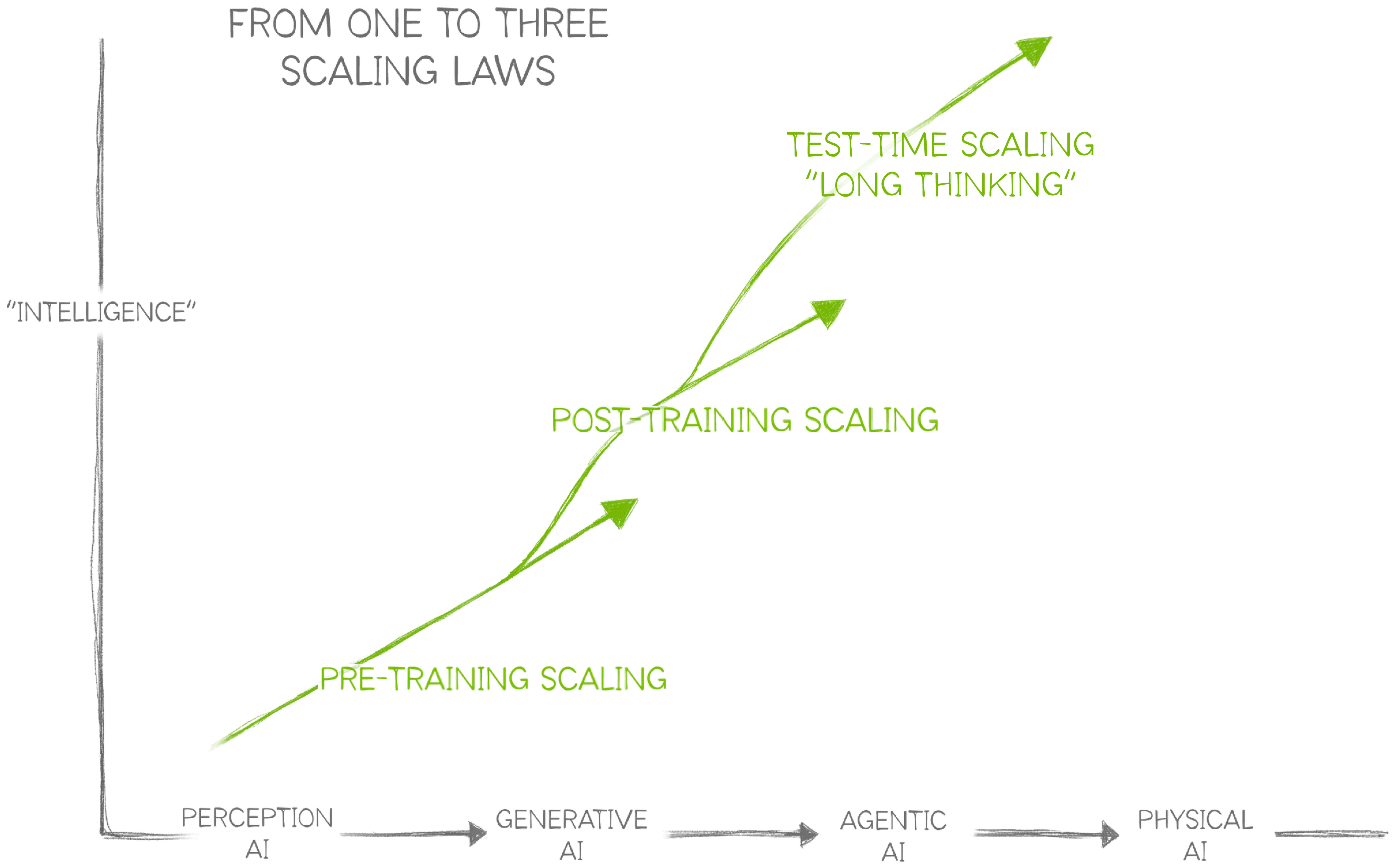
Senior Solutions Architect

Frédéric Parienté

Director, Solutions Architecture and Engineering

FAU

AI Scaling Laws Drive Exponential Demand for Compute



Inference Compute Requirements Scaling Exponentially

Fueled by reasoning models and AI agents

Larger Models

Hundreds of billions of parameters

Long Thinking Time

100x more thinking tokens

Larger Context

Millions of input tokens

Agents

One user prompt involves multiple model executions

Two Stages of LLM Execution

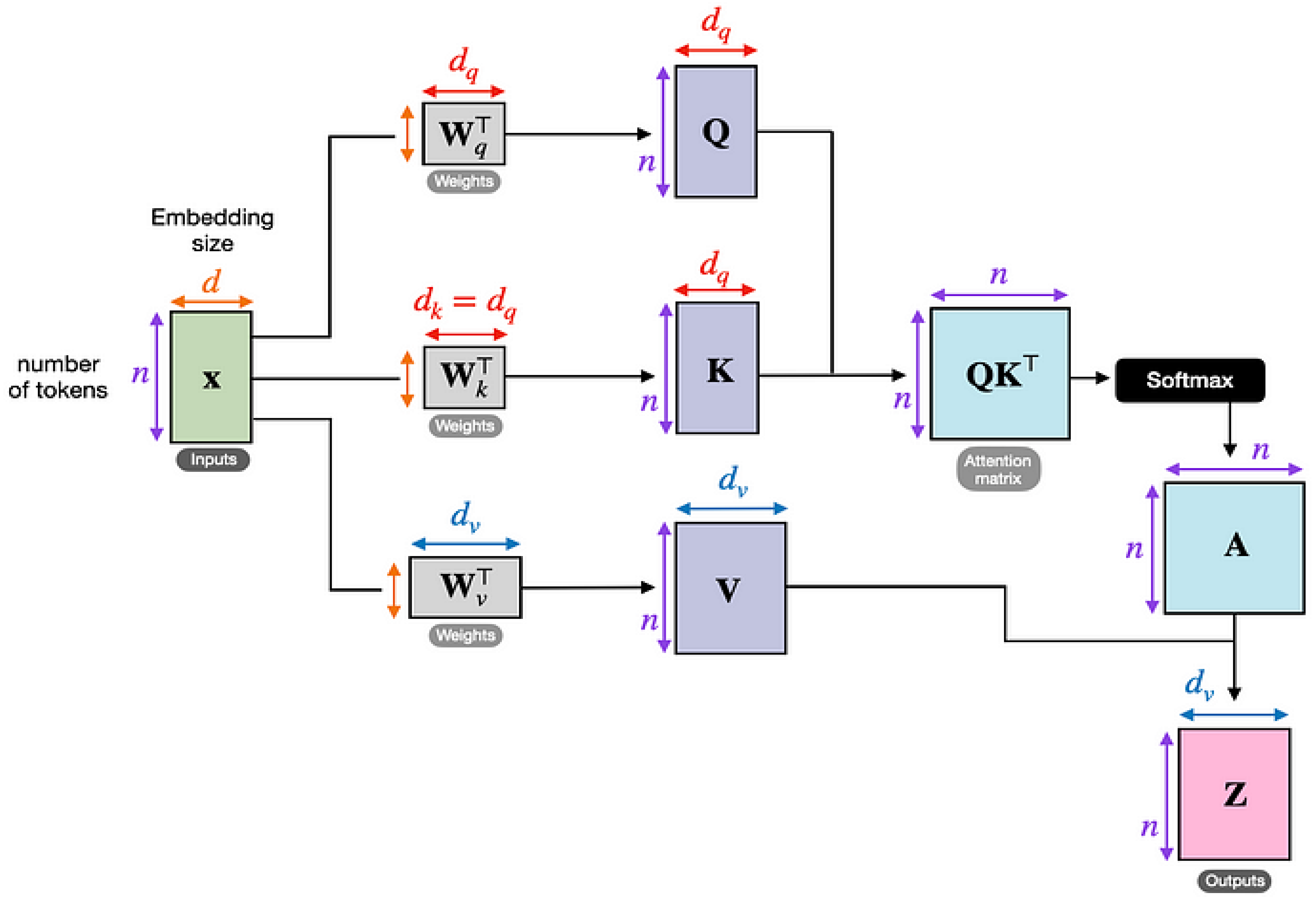
Prefill vs Decoding and the use of KV-cache

- **Prefill** = time to first token
  - Loading the user prompt into the system
  - Populate KV-cache for all the tokens from the prompt.
- **Decoding** = inter-token latency
  - Generating the response token by token
  - Reuse KV-cache to generate the next token

LLM inference is made of prefill followed by decode

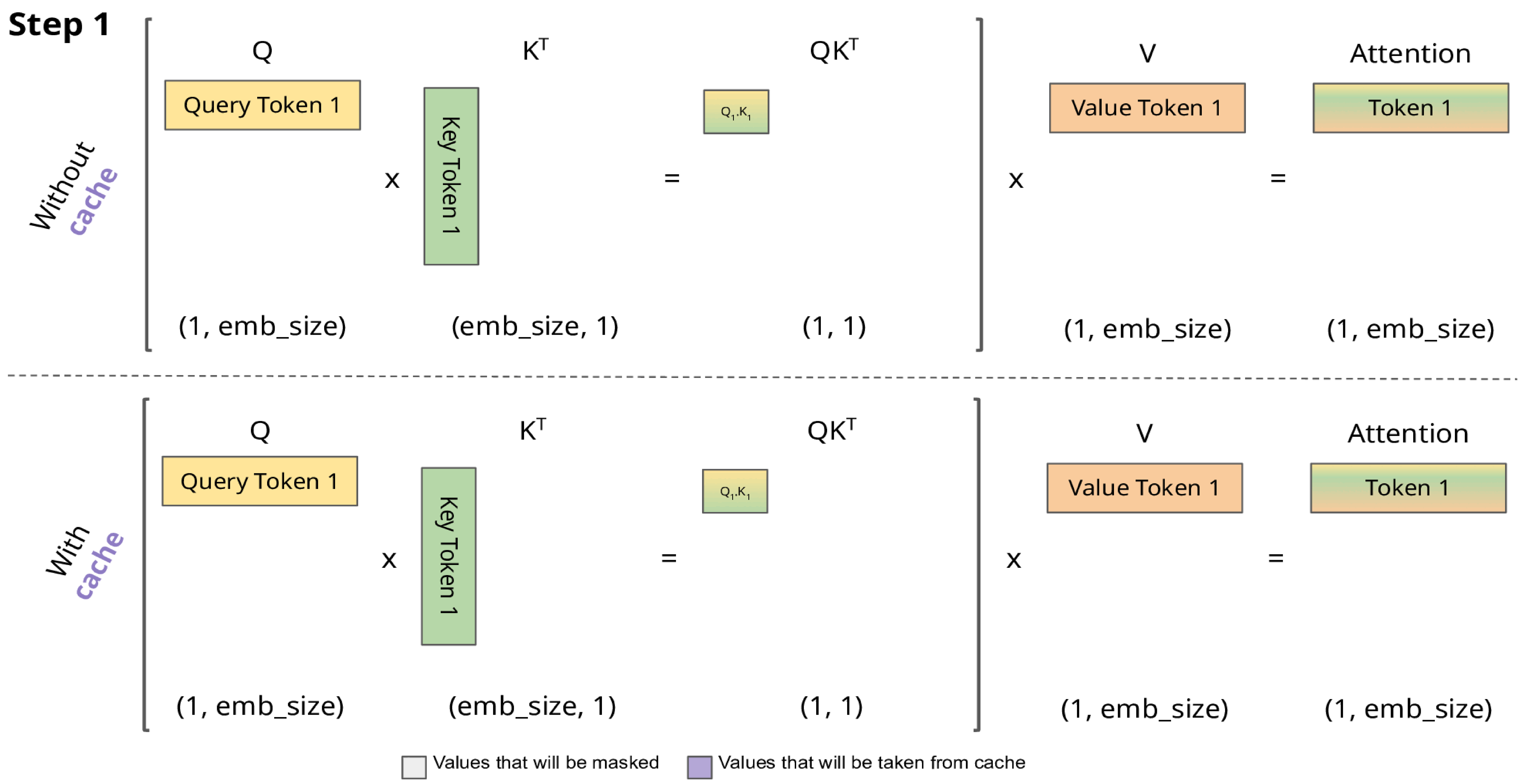
KV-cache

A look at the attention layer



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

KV-cache



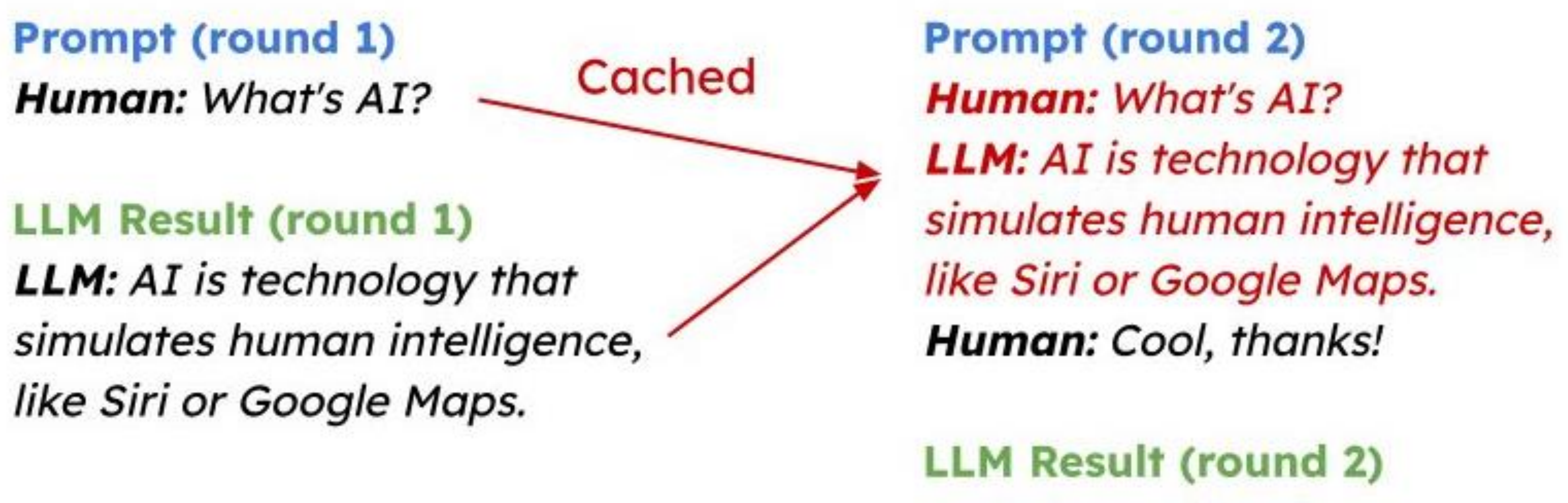
Step 1
Without cache
Q
Query Token 1
(1, emb_size)
x
K^T
Key Token 1
(emb_size, 1)
=
QK^T
$Q_1.K_1$
(1, 1)
x
V
Value Token 1
(1, emb_size)
=
Attention
Token 1
(1, emb_size)
With cache
Q
Query Token 1
(1, emb_size)
x
K^T
Key Token 1
(emb_size, 1)
=
QK^T
$Q_1.K_1$
(1, 1)
x
V
Value Token 1
(1, emb_size)
=
Attention
Token 1
(1, emb_size)
Values that will be masked
Values that will be taken from cache

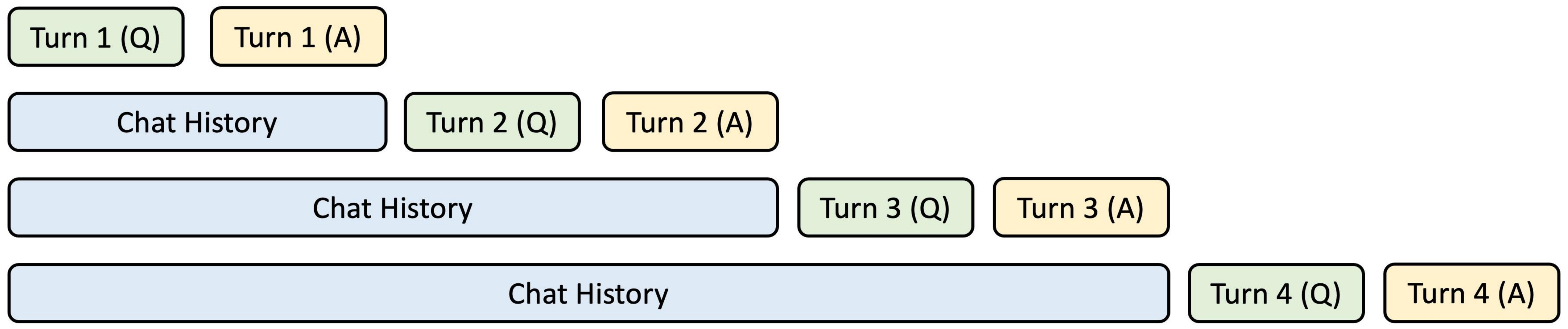
Prefix caching

KV cache reuse

KV cache are stored and can be used when generating different responses to the prompts that contain similar prefix

Multi-turn conversation





System prompt

Request A *A chat between a curious user and an artificial intelligence assistant. The assistant gives helpful, detailed, and polite answers to the user's questions. User: Hello!*

Request B *A chat between a curious user and an artificial intelligence assistant. The assistant gives helpful, detailed, and polite answers to the user's questions. User: How are you?*

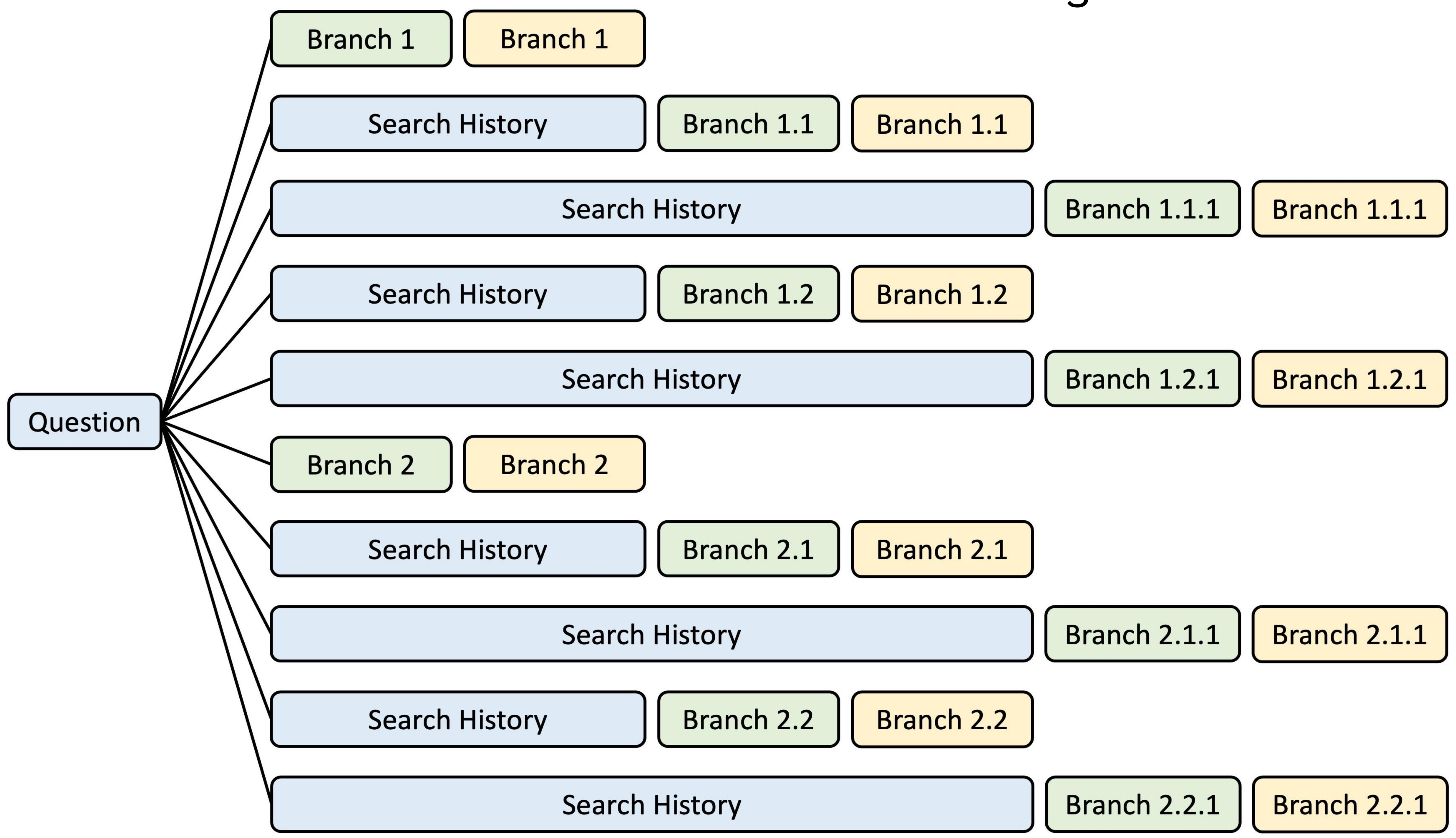
Request C *A chat between a curious user and an artificial intelligence assistant. The assistant speaks French. User: Bonjour!*

Prefix caching

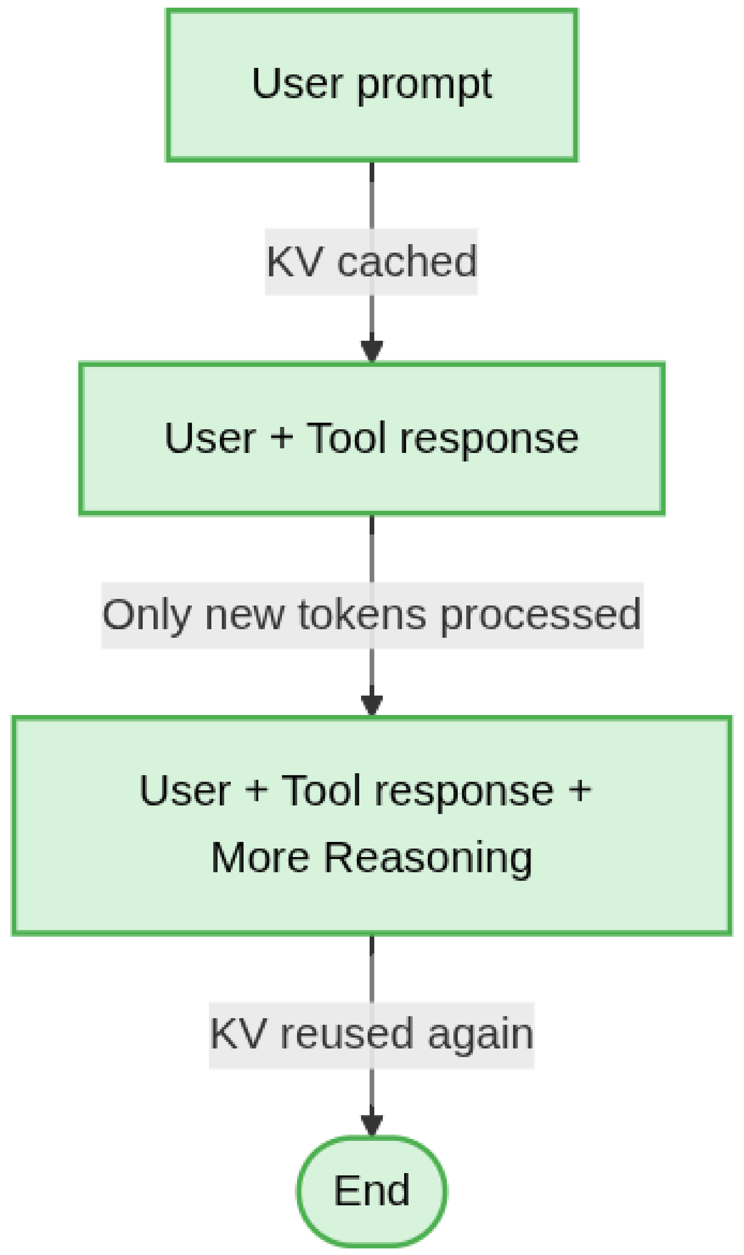
KV cache reuse

KV cache are stored and can be used when generating different responses to the prompts that contain similar prefix

Tree-of-thought

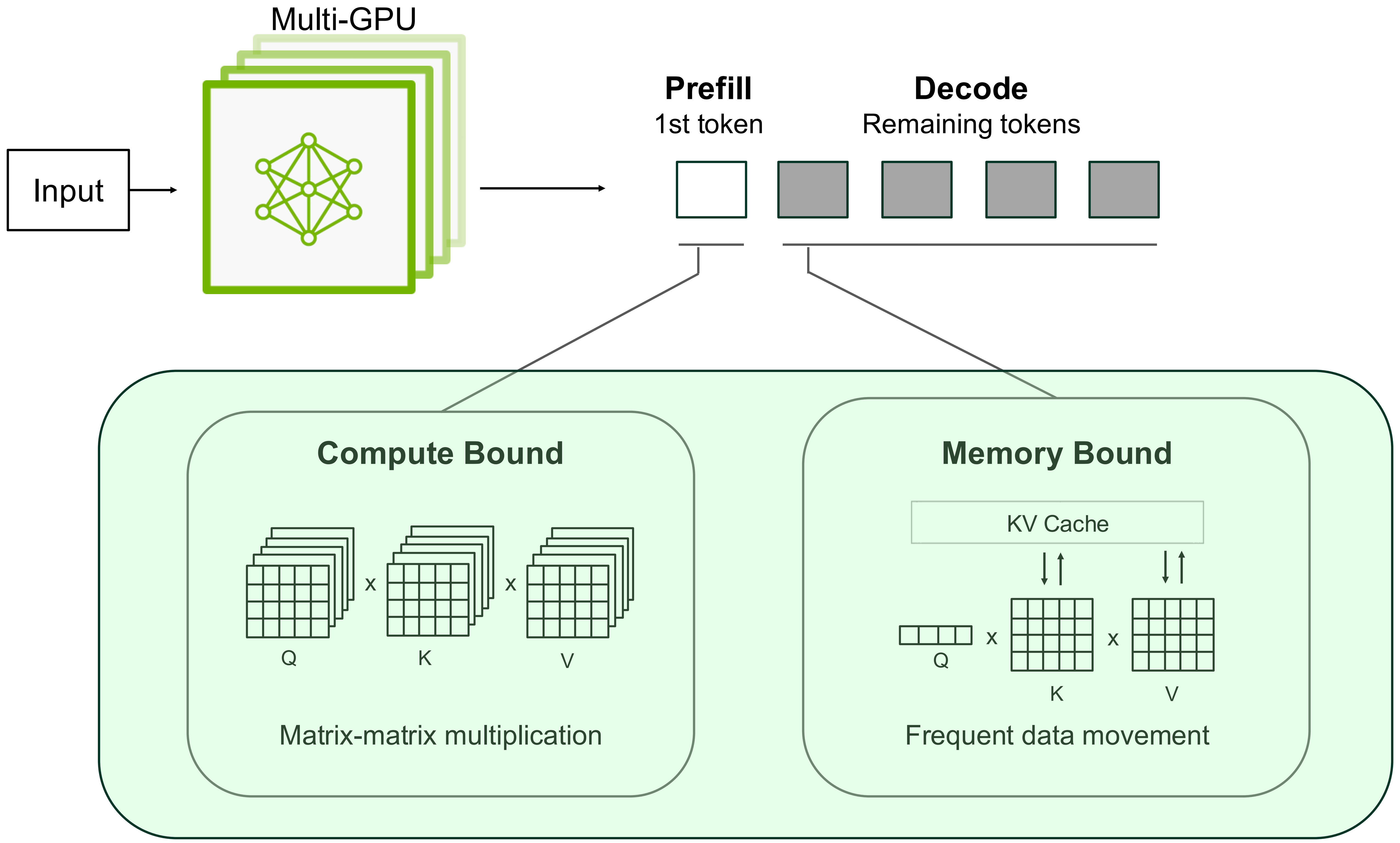


Agent reuse after tool usage



LLM inference

The effect of monolithic inference

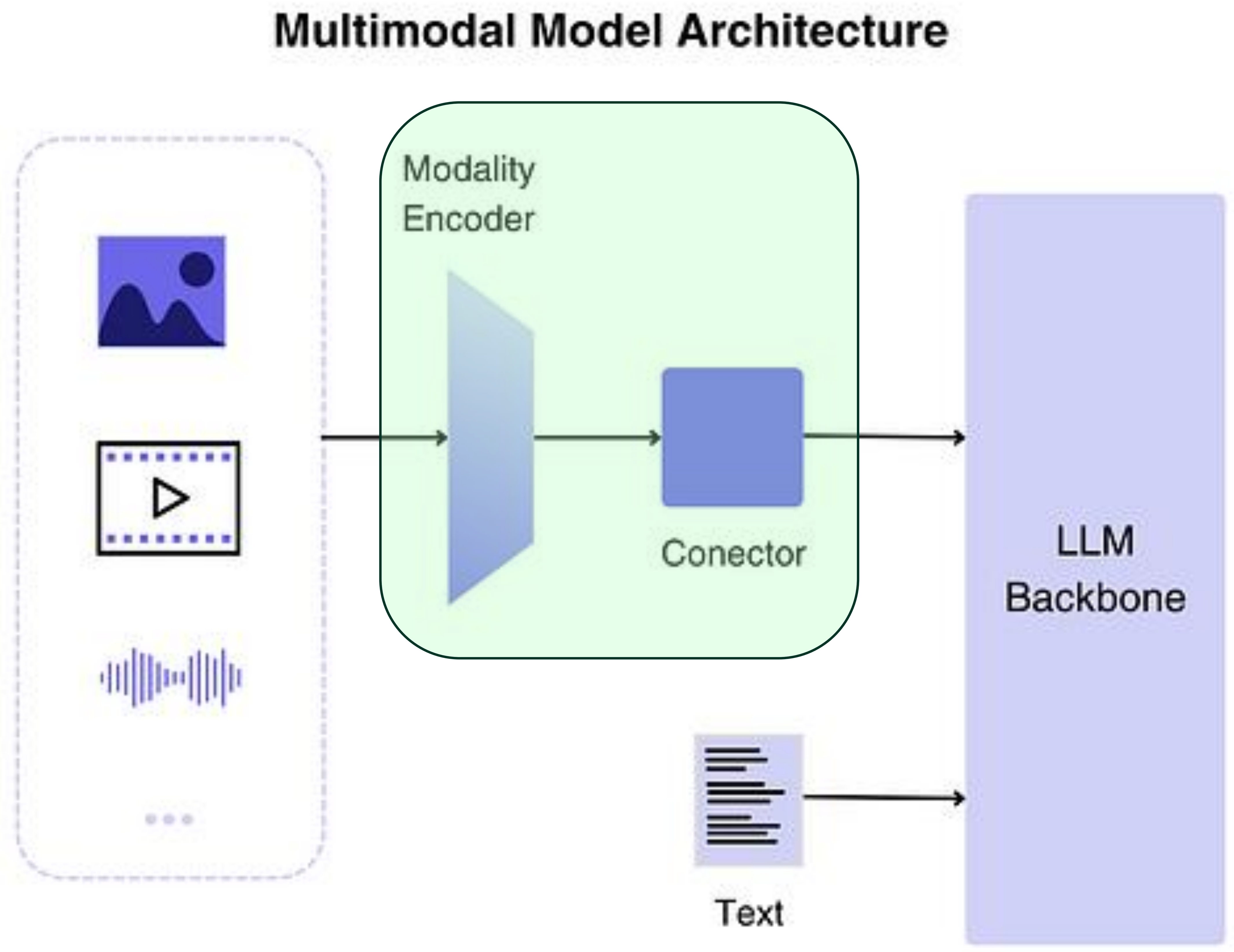


Share same parallelism strategy

Multimodality

Diverse elements require separate optimizations

- Beyond the LLM backbone, multimodal models have image or video encoders
- Encoders are compute bound.



Estimating the size of the KV Cache

Total size of KV cache in bytes = $2 * sizeof(precision) * n_{layers} * d_{model} * seqlen * batch$

$$2 = two\ matrices\ of\ K\ and\ V$$

$$precision = bytes/parameter\ (FP16 = 2bytes)$$

$$n_{layers} = layers\ in\ the\ model$$

$$d_{model} = Dimension\ of\ the\ embeddings$$

$$seqlen = length\ of\ context\ in\ tokens\ (input\ prompt + generated\ output)$$

$$batch = batch\ size$$

Example of a KV Cache size for LLaMa2 7B model in FP16 and a batch size of 1

$$2 * 2 * 4096 * 32 * 4096 * 1 = 2GB$$

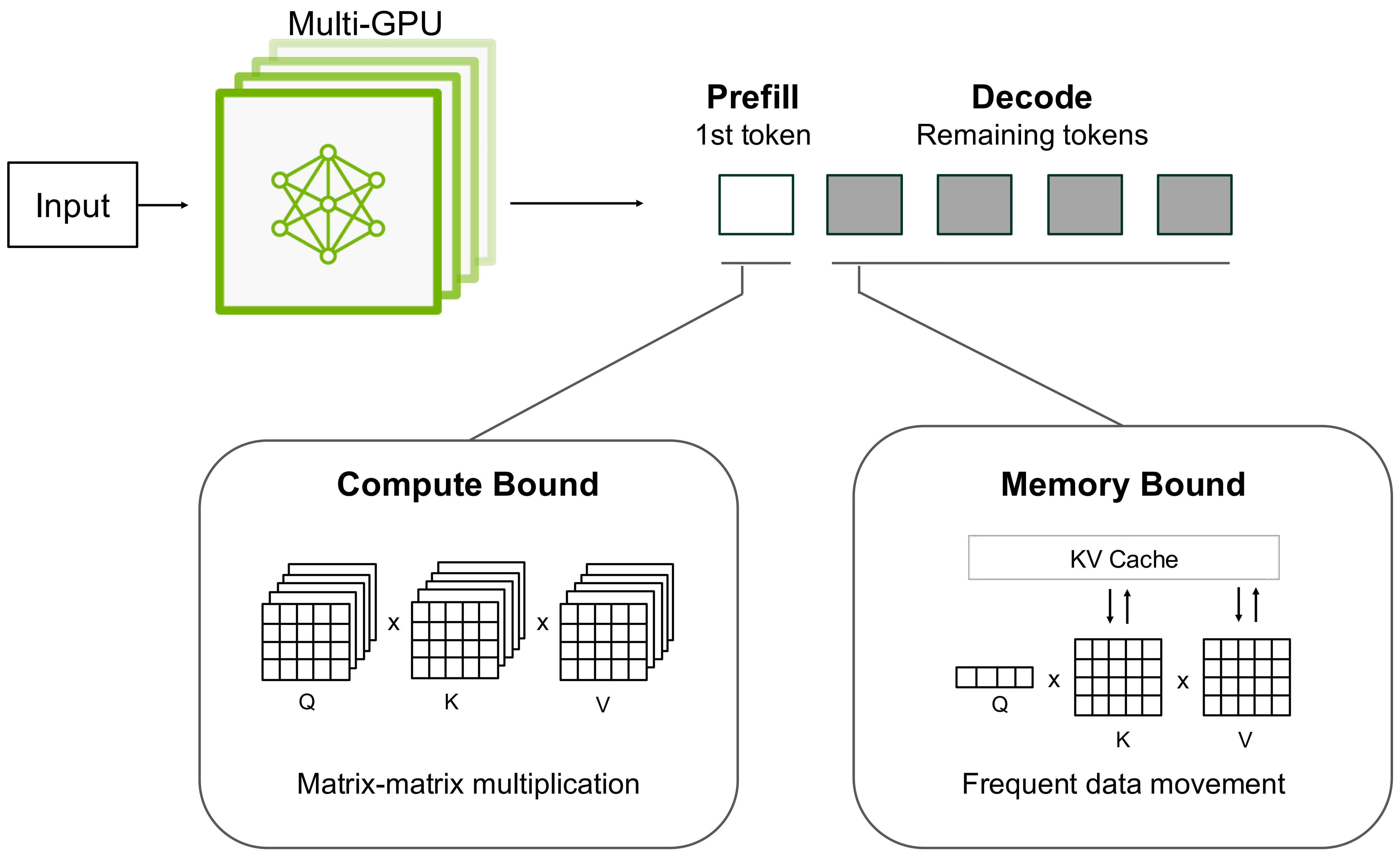
Source: Mastering LLM Techniques: Inference Optimization

NVIDIA

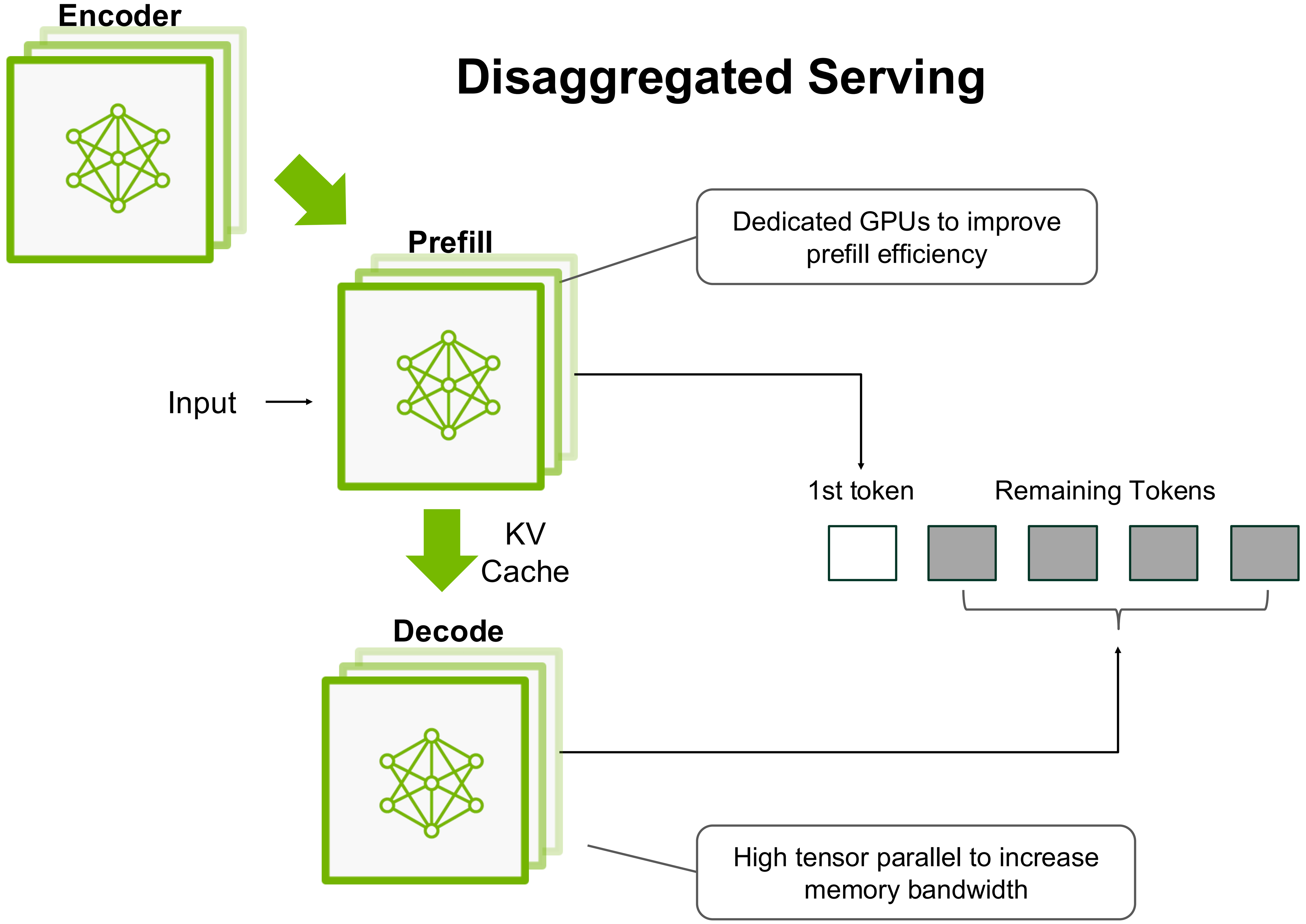
New Inference Optimization Techniques to Boost Inference

Disaggregated serving separates prefill and decode allowing each to be optimized independently

Traditional Serving



Disaggregated Serving



More flexibility to optimize cost and user experience

Inferencing in the End-to-end AI Workstream

AI Training

AI Inference

DATA PROCESSING

Collect, cleanse and transform training data set

TRAIN

Select model and fine tune for desired accuracy

OPTIMIZE

Optimize model for target HW accelerator

DEPLOY

Serve model for optimal user experience & TCO

Inference Engine

Inference Engines ecosystem

vLLM: https://catalog.ngc.nvidia.com/orgs/nvidia/containers/vllm/tags?version=25.10-py3
SGLang: https://catalog.ngc.nvidia.com/orgs/nvidia/containers/sglang?version=25.10-py3

TensorRT-LLM in the *DL Compiler* Ecosystem

TensorRT-LLM builds on TensorRT Compilation

- **TensorRT-LLM**
  - Inference runtime & compiler specifically designed for LLMs
  - LLM specific optimizations:
    - KV Caching & Custom MHA Kernels
    - Inflight batching, Paged KV Cache (Attention)
    - Multi-GPU, Multi-Node
    - Grammar support
    - *& more*
  - *ONLY for LLMs*

- **TensorRT**
  - General purpose Deep Learning Inference Compiler
    - Graph rewriting, constant folding, kernel fusion
    - Optimized GEMMs & pointwise kernels
    - Kernel Auto-Tuning
    - Memory Optimizations
    - *& more*
  - *All AI Workloads*

TensorRT-LLM

LLM specific optimizations:

- KV Caching
- Multi-GPU, Muti-Node
- Custom MHA optimizations
- Paged KV Cache (Attention)
- *etc…*

TensorRT

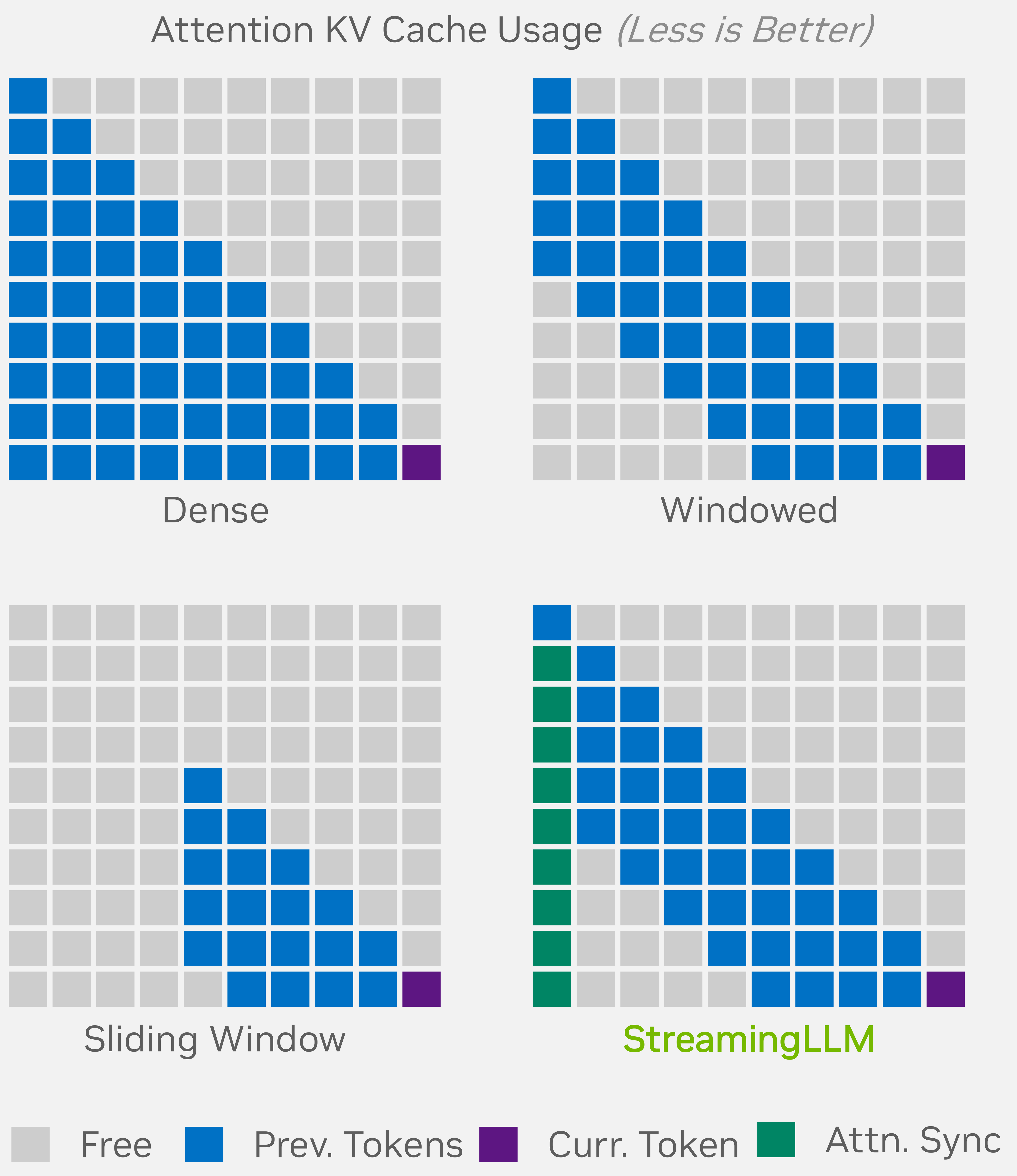
General Purpose Compiler

- Optimized GEMMs & general kernels
- Kernel Fusion
- Auto Tuning
- Memory Optimizations
- Multi-stream execution

KV Cache & Attention Techniques

(Sliding) Window Attention, & Streaming LLM

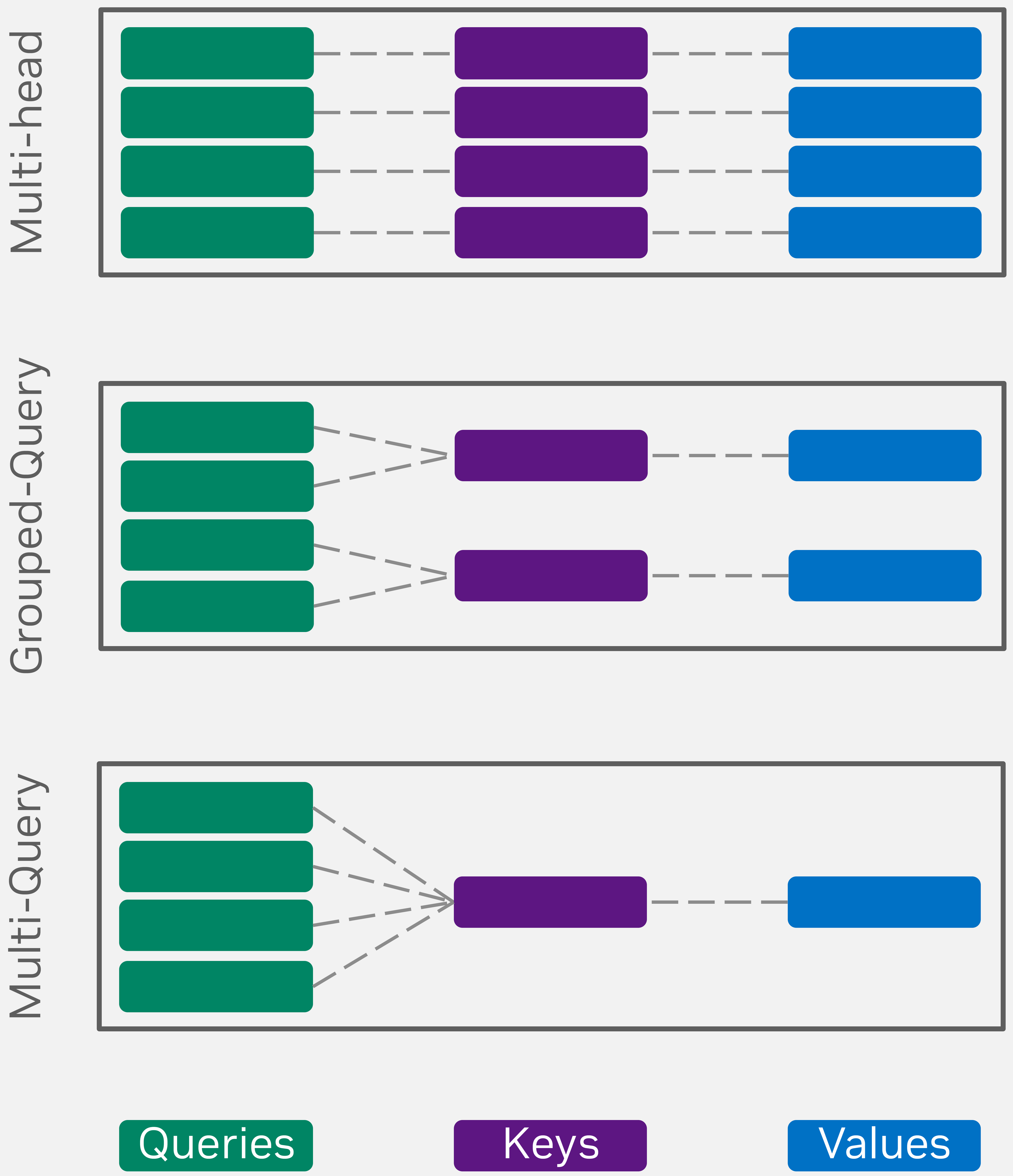
- Allow for longer (sometimes unlimited) sequence length
  - Reduces KV Cache Memory usage
  - Avoids OOM Errors
- (Sliding) Windowed Attention evict tokens based on arrival
  - Significantly reduces memory usage
  - Can negatively impact accuracy or require recomputing KV
- Streaming-LLM allows for unlimited sequence length
  - Does not evict Attention Sinks (important elements)
  - KV Cache stays constant size
  - Does not require recompute & does not impact accuracy
  - Particulary beneficial for multi-turn (ie. chat) usecases



Optimized Attention

Custom Implementations for Attention

- Custom optimized CUDA kernels for Attention
  - Similar to FlashAttentionV2
- Optimized for A100 & H100 & B200
- Kernels for Encoder & Decoder, as well as context & prefill
- Supports MHA, MQA, GQA

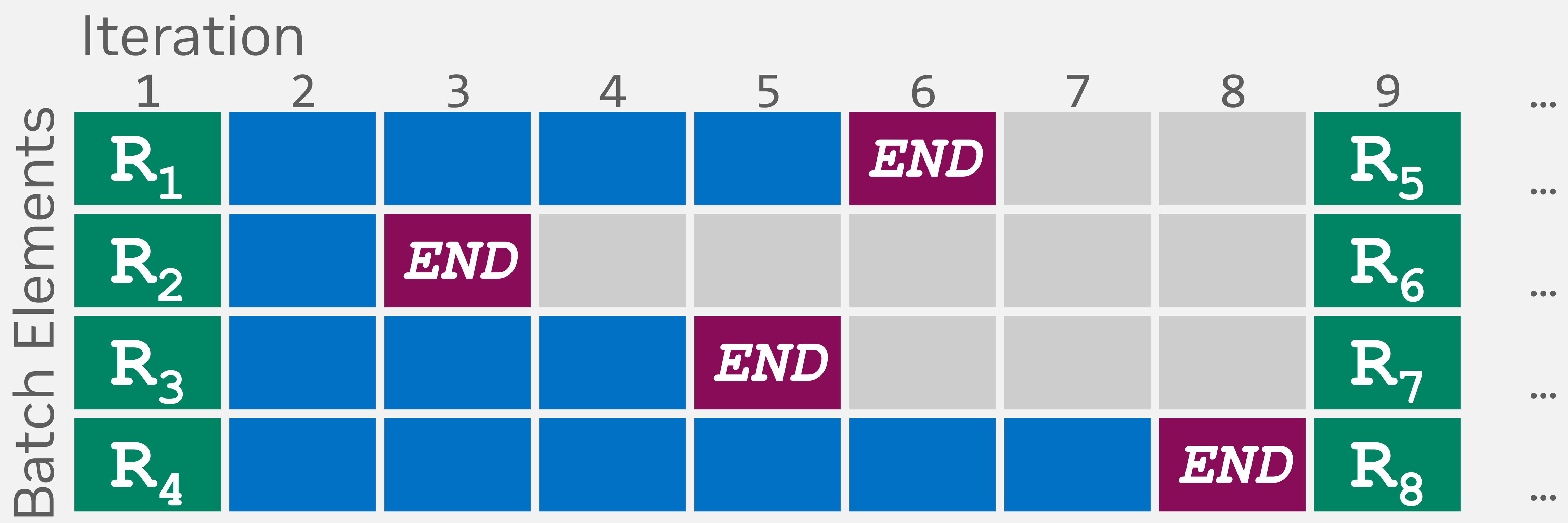


Inflight Batching

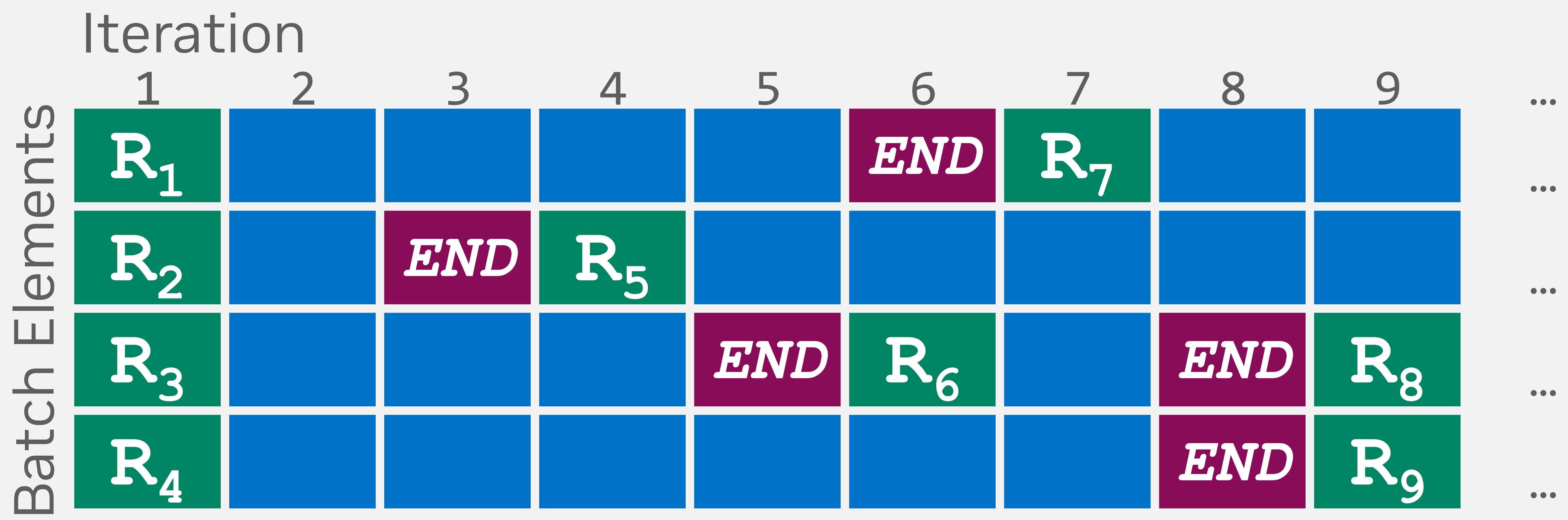
Maximizing GPU Utilization during LLM Serving

TensorRT-LLM provides custom Inflight Batching to optimize GPU utilization during LLM Serving

- Replaces completed requests in the batch
  - Evicts requests after EoS & inserts a new request
- Improves throughput, time to first token, & GPU utilization
- Integrated directly into the TensorRT-LLM Triton backend
- Accessible though the TensorRT-LLM Batch Manager



Static Batching



Inflight Batching

Context Gen EoS NoOp

KV Cache Optimizations

Paged & Quantized KV Cache

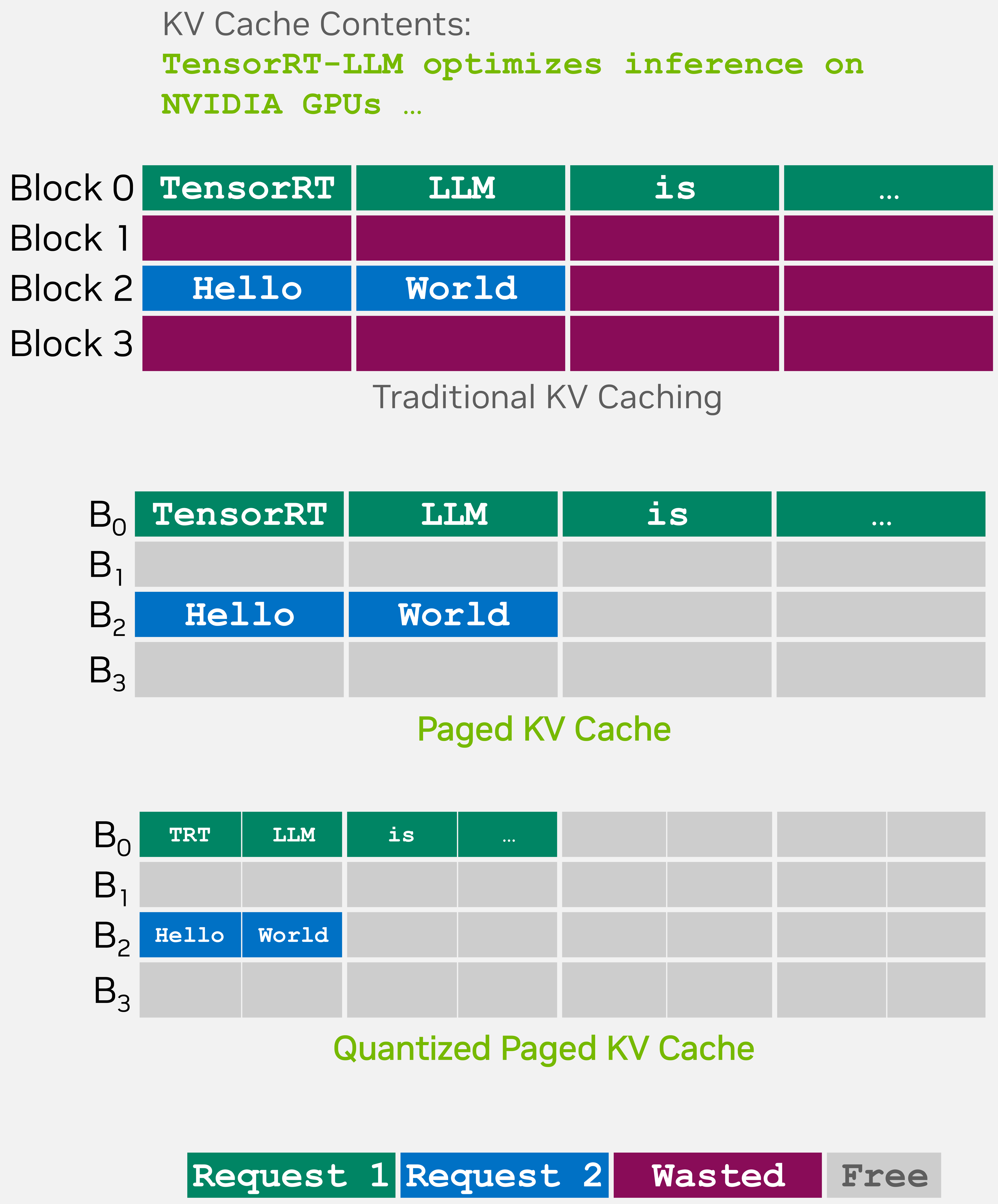
Paged KV Cache improves memory consumption & utilization

- Stores keys & values in non-contiguous memory space
- Allows for reduced memory consumption of KV cache
- Allocates memory on demand

Quantized KV Cache improves memory consumption & perf

- Reduces KV Cache elements from 16b to 8b (or less!)
- Reduces memory transfer improving performance
- Supports INT8 / FP8 KV Caches

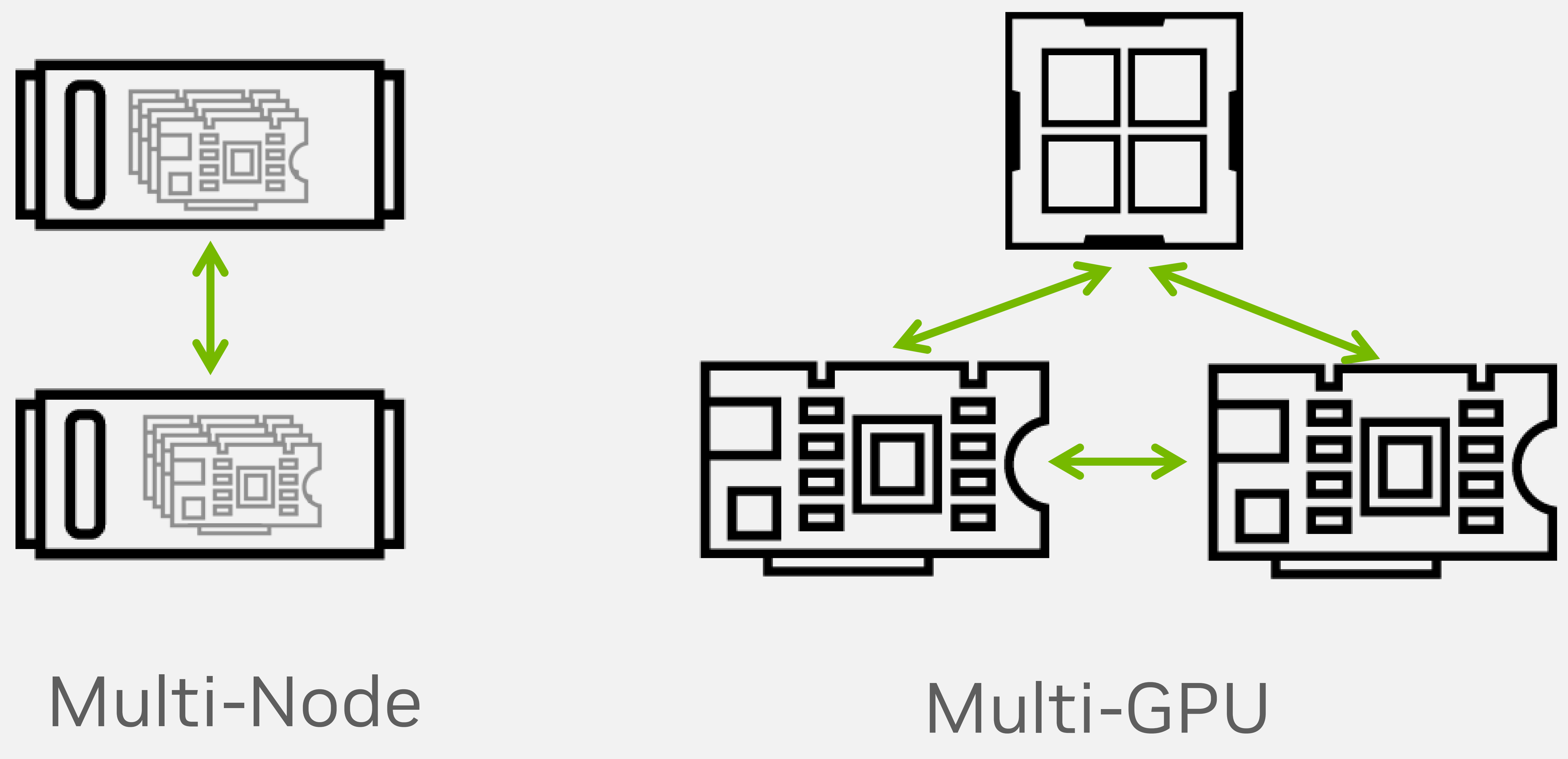
Both allow for increased peak performance

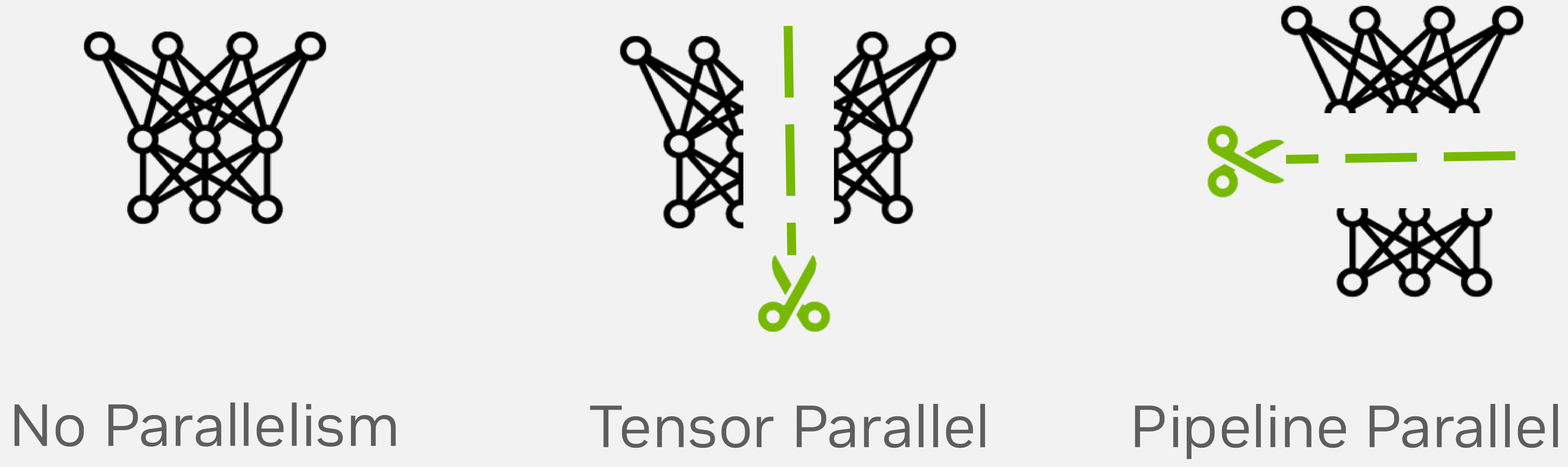


Multi-GPU Multi-Node

Sharding Models across GPUs

- Supports Tensor & Pipeline parallelism
- Allows for running very large models (tested up to 530B)
- Supports multi-GPU (single node) & multi-node
- TensorRT-LLM handles communication between GPUs
- Examples are parametrized for sharding across GPUs





Inference serving

NVIDIA Dynamo Platform

The Operating System for AI Factories

NVIDIA Dynamo Platform

NVIDIA Dynamo

Distributed and Disaggregated Generative AI Serving

NVIDIA Dynamo Triton

(formerly Triton Inference Server)

Standardized Model Deployment Across Every AI Workload

NVIDIA NIM

Fastest and Easiest Way to Deploy NVIDIA Dynamo Platform in Production

NVIDIA Dynamo Triton (formerly Triton Inference Server)

Deploy models from all popular frameworks across GPUs and CPUs

Any Framework

Supports Multiple Framework Backends Natively e.g., TensorFlow, PyTorch, TensorRT, XGBoost, ONNX, Python, TensorRT-LLM, vLLM & More

Any Query Type

Optimized for Real Time, Batch, Streaming, Ensemble Inferencing

Any Platform

X86 CPU | Arm CPU | NVIDIA GPUs | MIG

Linux | Windows | Virtualization

Public Cloud, Data Center and Edge/Embedded (Jetson)

DevOps & MLOps

Integration With Kubernetes, KServe, Prometheus & Grafana

Available Across All Major Cloud AI Platforms

Performance & Utilization

Model Analyzer for Optimal Configuration

Optimized for High GPU/CPU Utilization, High Throughput & Low Latency

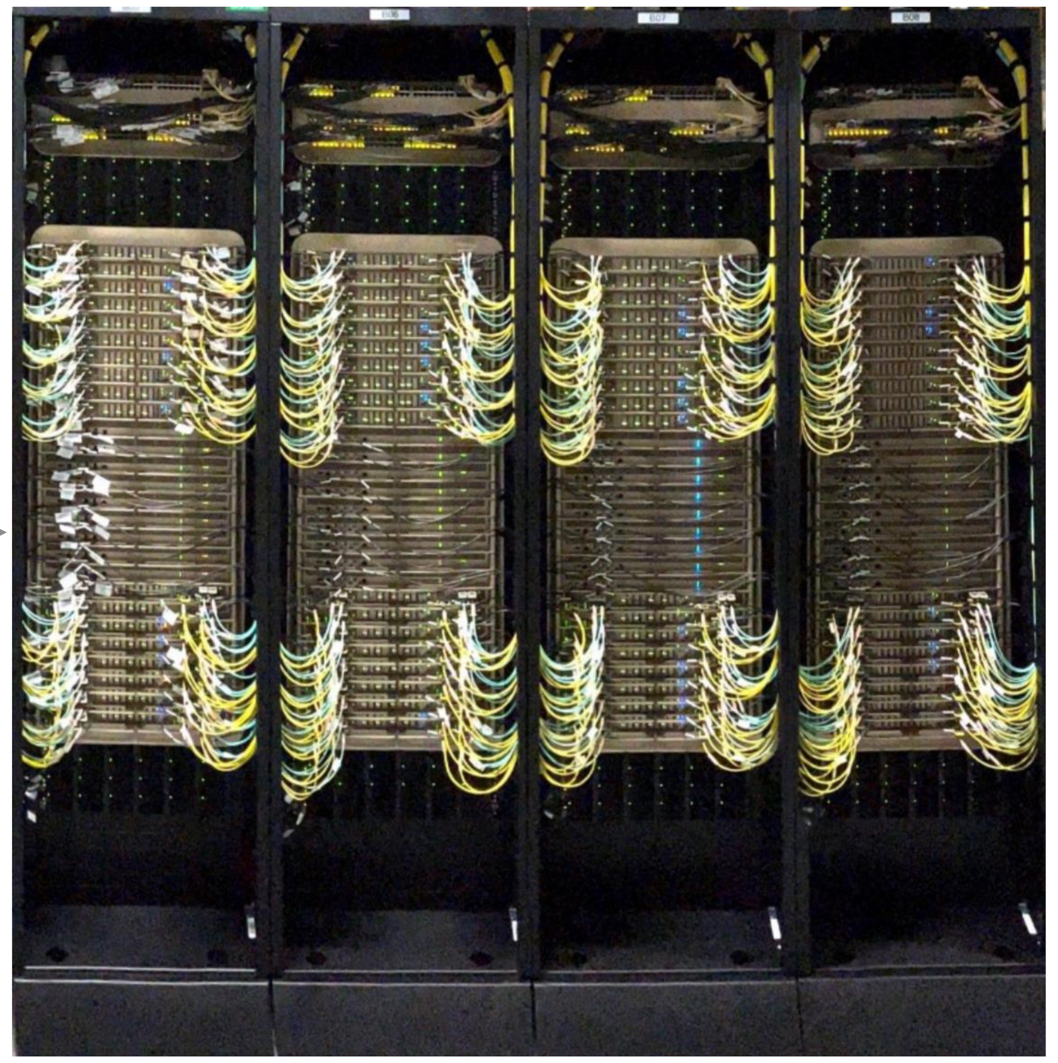
Announcing NVIDIA Dynamo

AI Inference Software for Reasoning Inference at Scale

PyTorch | SGL | TensorRT | vLLM

NVIDIA Dynamo

- Smart Router
- GPU Planner
- Low-latency Communication Library
- KV-Cache Offload Manager



Blackwell Cluster

1000+

GPU Scale for a single query

2X

Throughput & Revenue

Llama Models On Hopper[2]

vLLM, FP8, ISL/OSL: 3K/50 . Without Dynamo: Inflight Batching, TP8DP2. With Dynamo: Disaggregated Serving, Context: TP2DP4, Generation: TP8. 2- vLLM, FP8, ISL/OSL: 3K/50 .

Architecture and Components

Benefits of disaggregation on multiple nodes

Benchmark on multiple nodes

Key result:

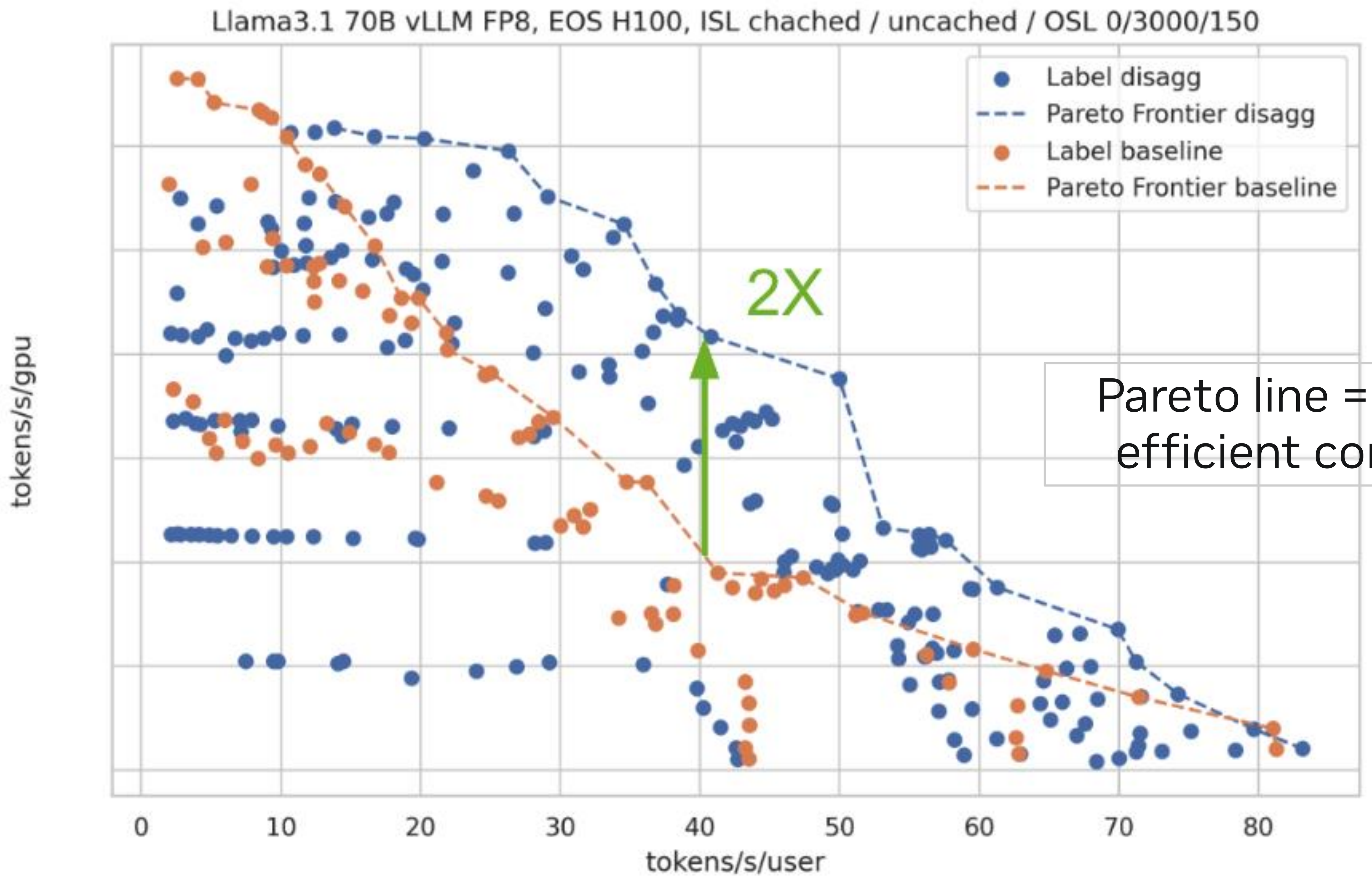
- disaggregated consistently outperforms aggregated
- Up to 2× throughput per GPU improvement

Config insights:

- Aggregated best: TP8 DP2
- Disaggregated best: prefill TP2 DP4, decode TP8

Prefill favors more data parallelism for batch efficiency
Decode favors higher tensor parallelism for GPU utilization

Two Nodes

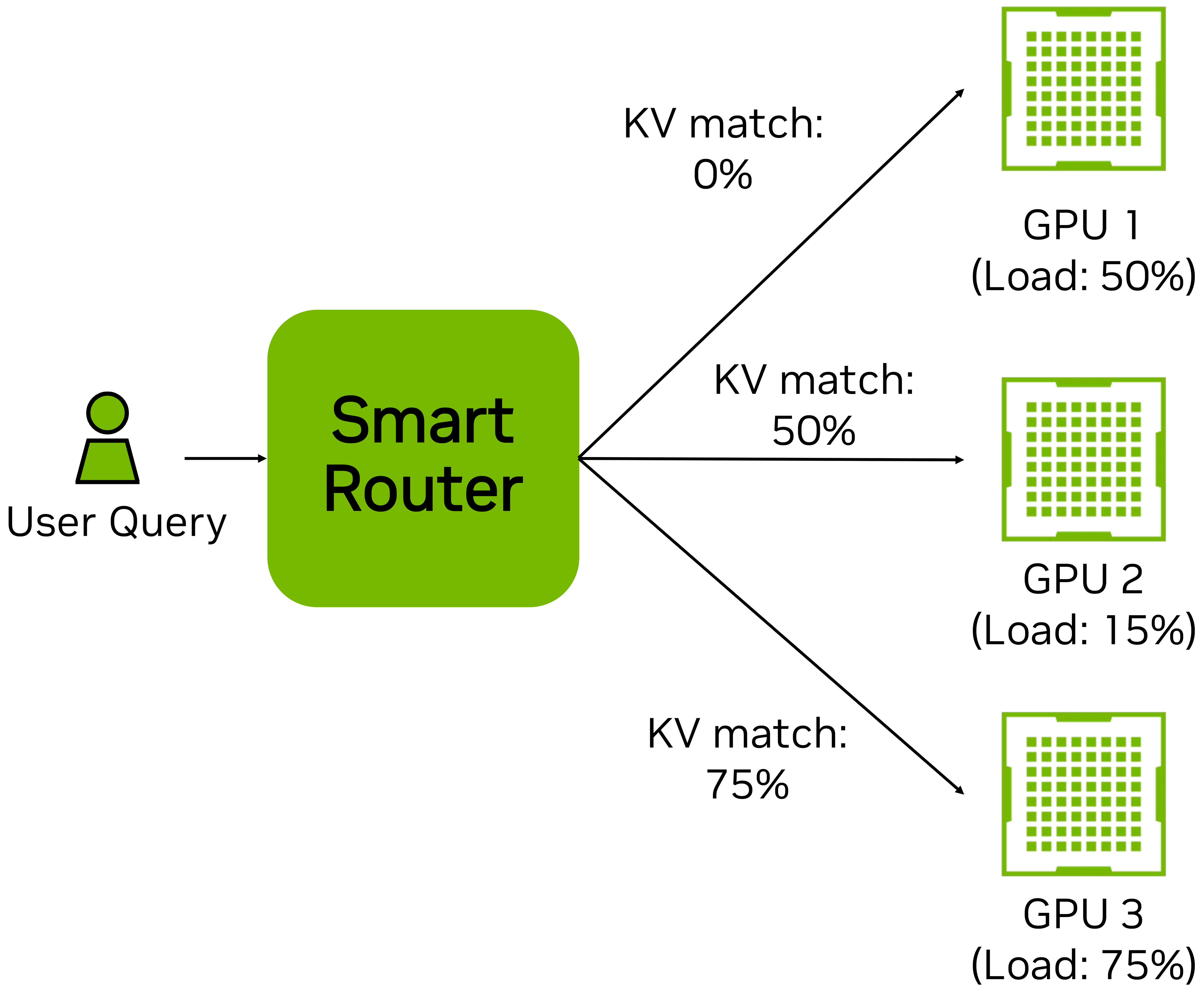


Pareto line = most efficient configs

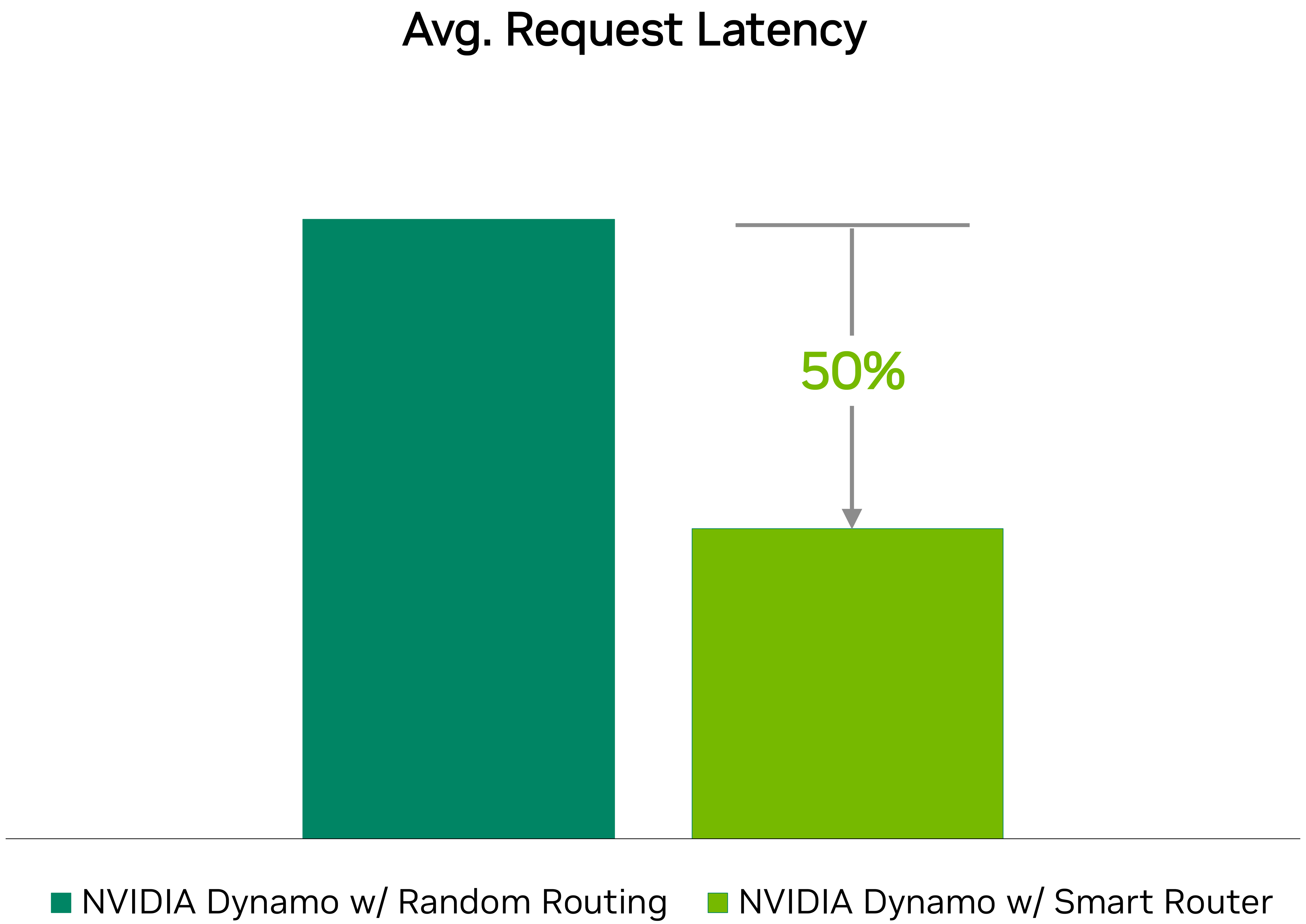
Each dot = different config (TP, PP, DP)

NVIDIA Dynamo: Smart Router

Reducing costly re-computation of KV cache



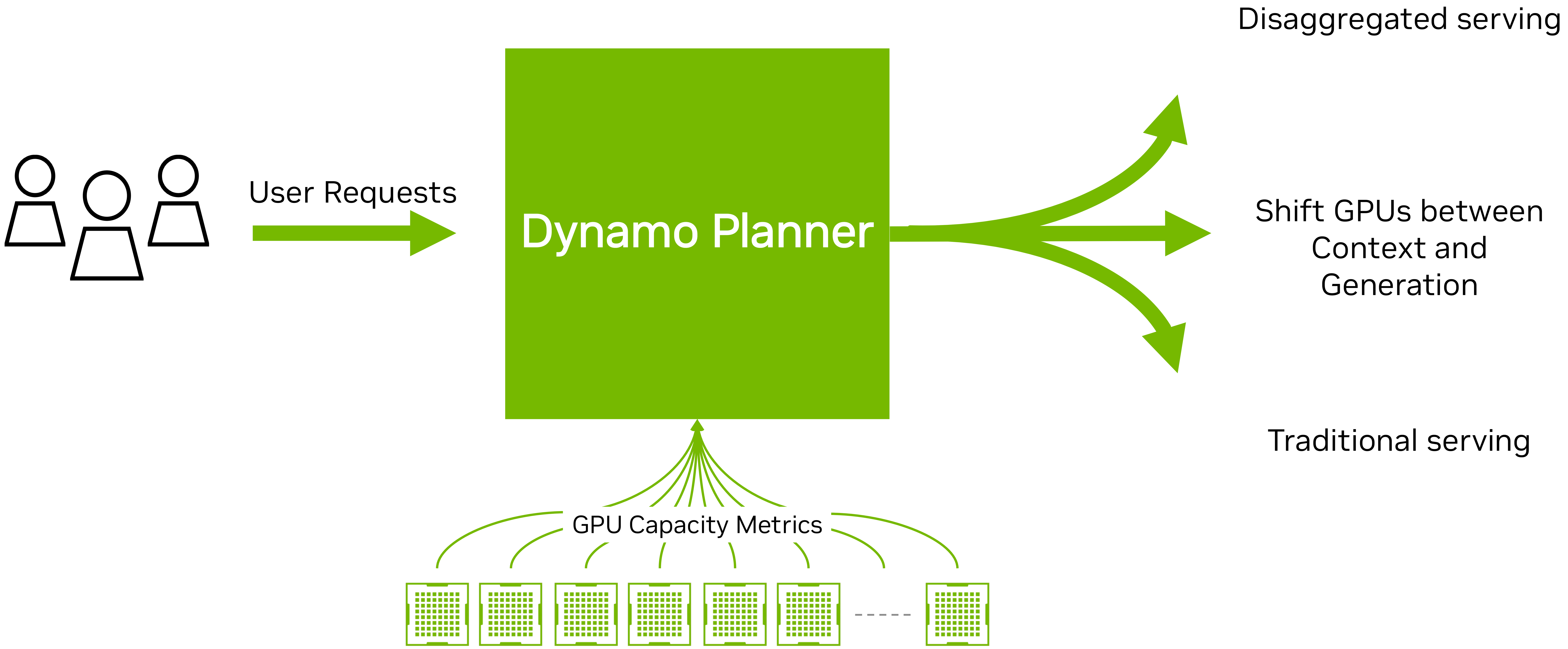
DeepSeek-R1 Distill Llama 70B | NVIDIA HGX-H100
(Lower is Better)



2x HGX-H100 nodes
8x DeepSeek-R1- Distill-Llama-70B. vLLM, FP8, Tensor Parallel: 2
Data Source: 100K R1 requests,

NVIDIA Dynamo: GPU Planner

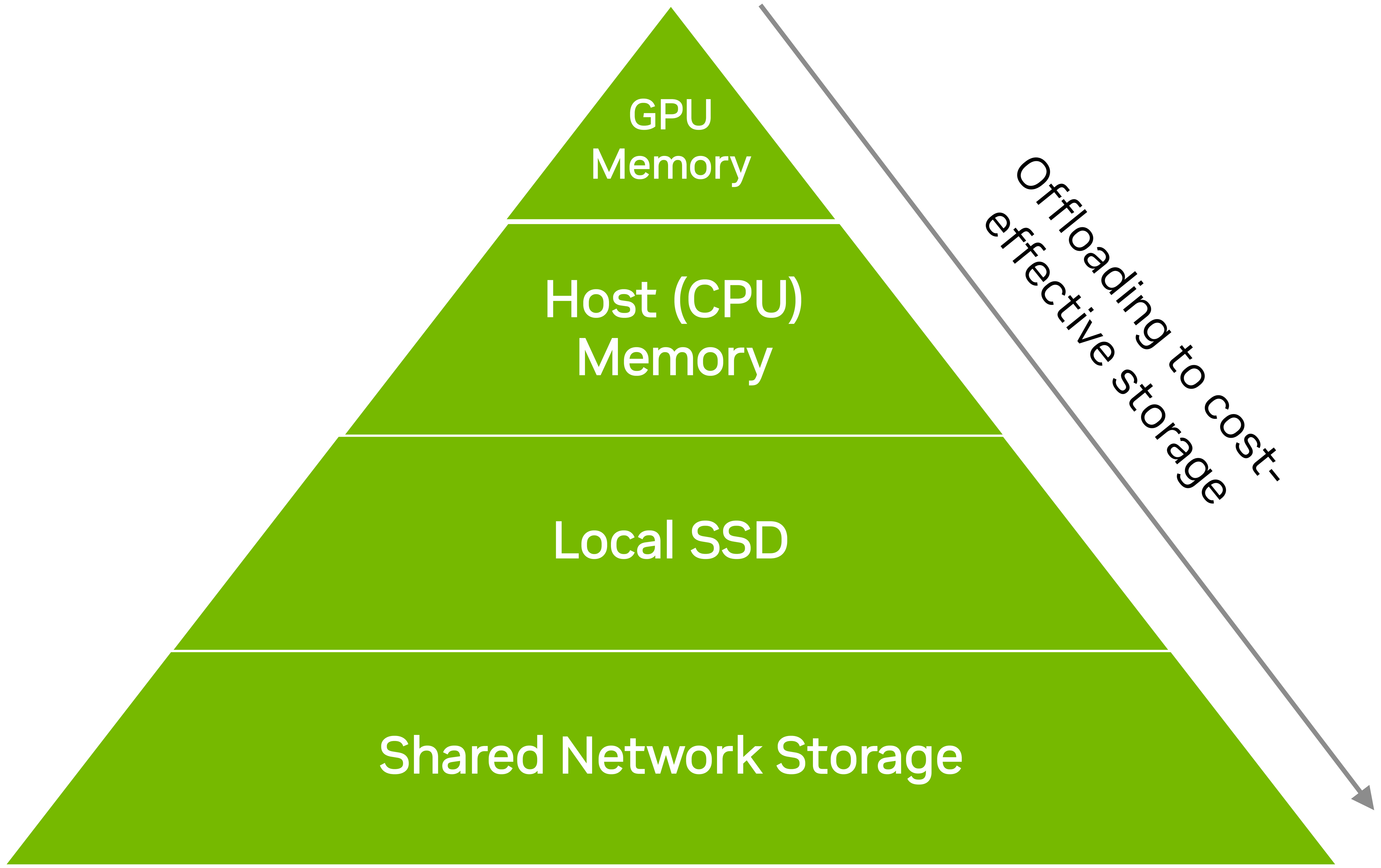
Optimizing GPU resources for distributed inference



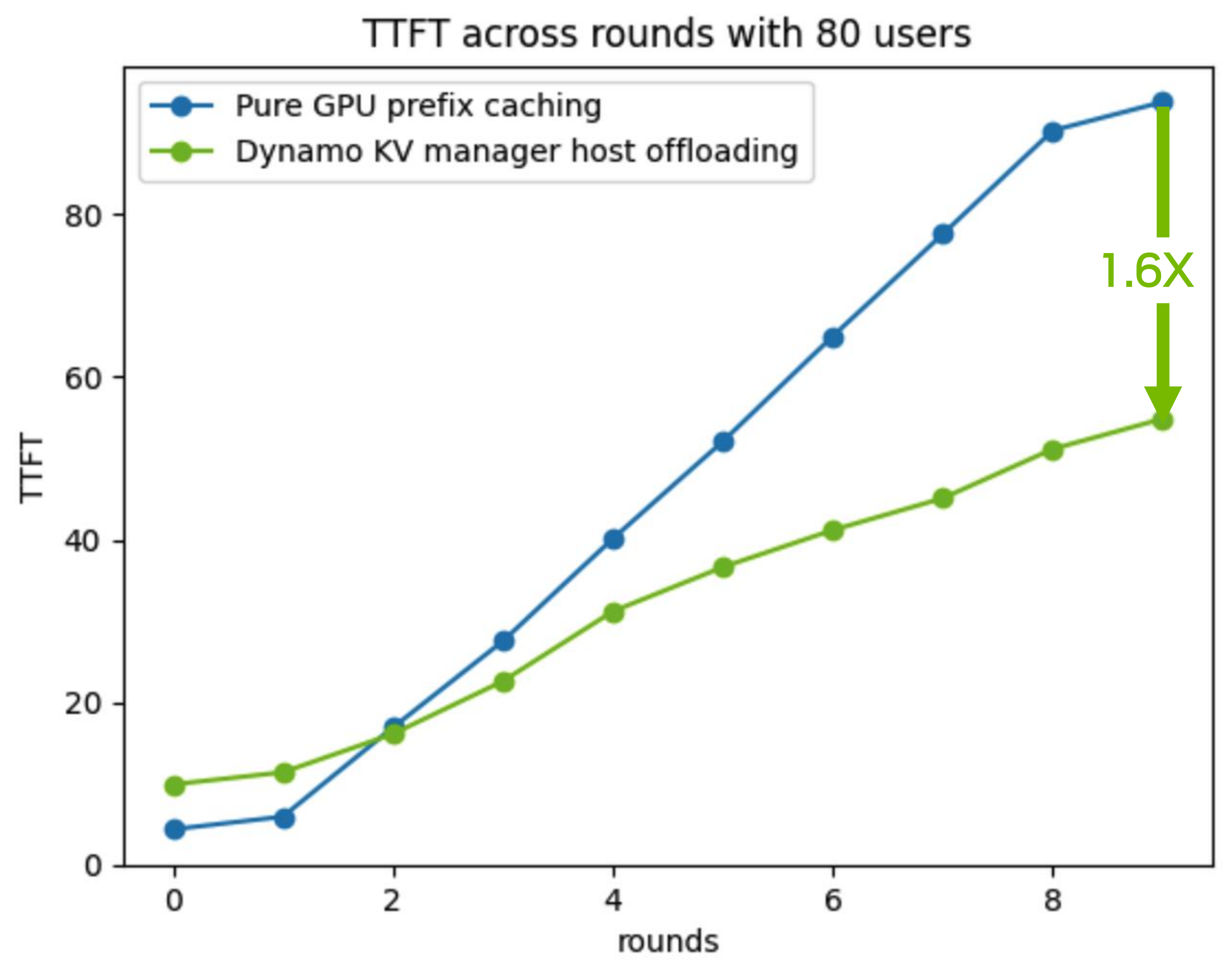
Efficient Resource Allocation | Adjust to Fluctuating Demand | Lower Inference Costs

NVIDIA Dynamo: KV Cache Manager

Offloading KV cache to cost-effective storage



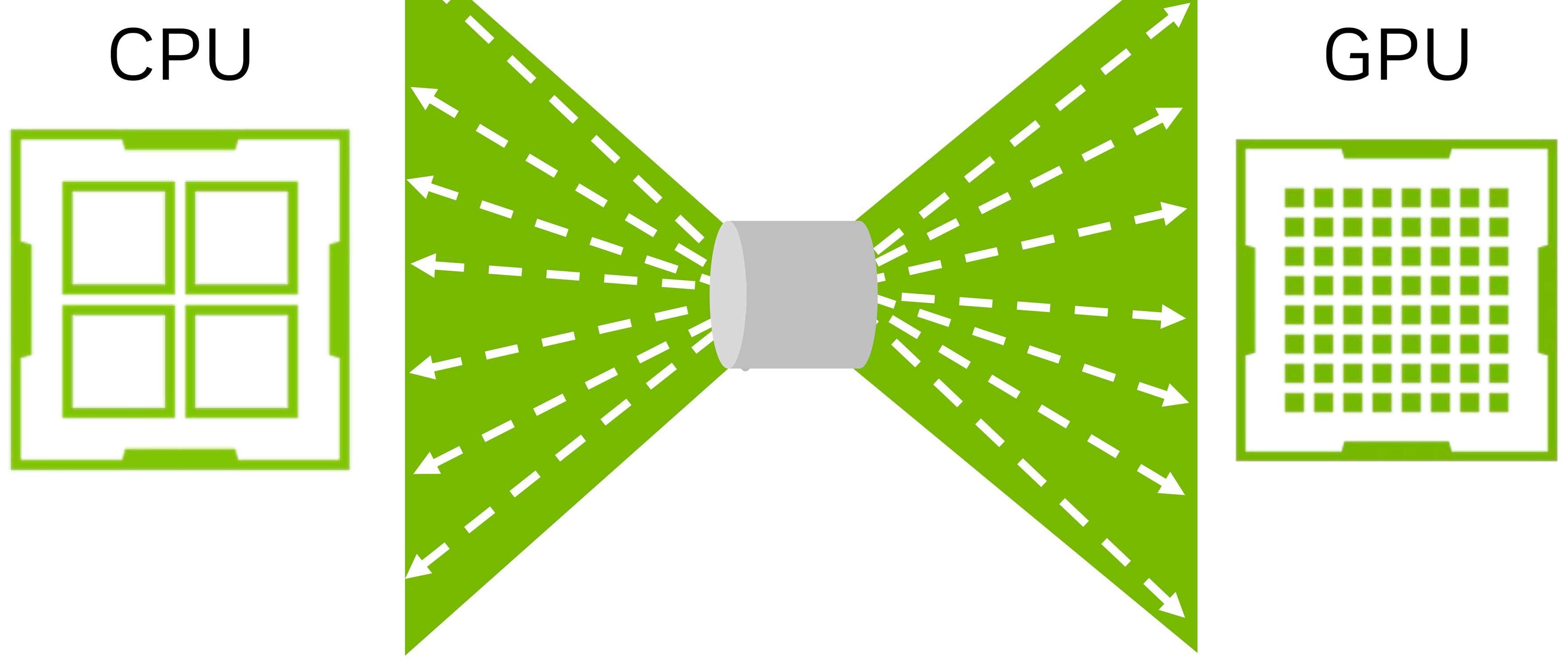
Llama 8B | NVIDIA HGX-H100



Grace Blackwell NVLink-C2C is Ideal For Inference

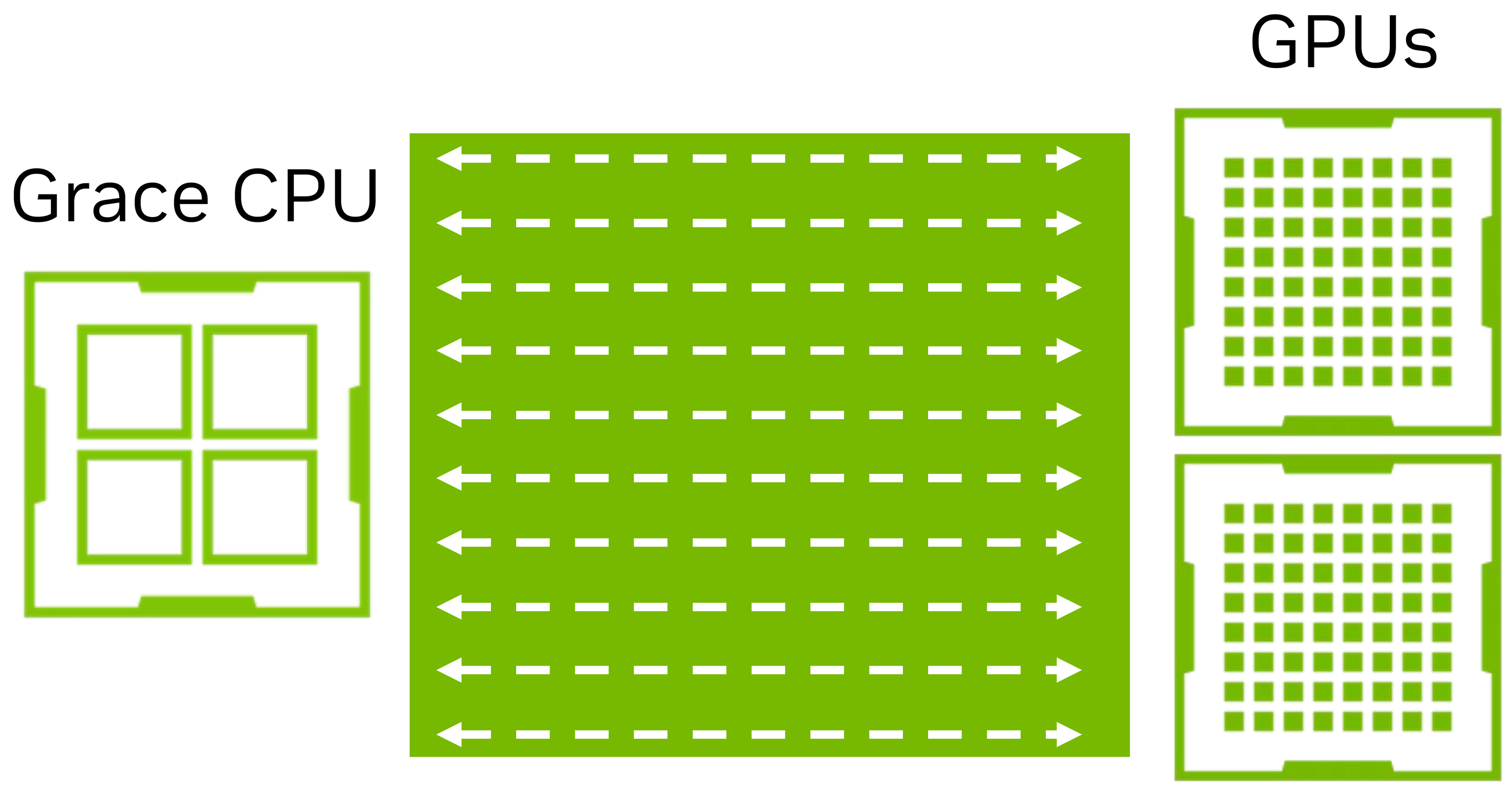
Avoids KV Cache re-computation by offloading to CPU memory

Traditional Architecture



PCIe bottlenecks CPU-GPU communication

NVIDIA Grace Blackwell Superchip Architecture



7x faster CPU-GPU KV Cache Transfers

NVIDIA Dynamo Breakthrough Features

A modular generative AI inference server designed for distributed and disaggregated serving

NVIDIA Dynamo

Distributed Inference Serving	Seamlessly scale LLMs from a single GPU to thousands of GPUs
GPU Planning & Scheduling	Meet changing demand patterns w/o over or under provisioning of resources
Smart Request Router	Free up GPU resources by reducing re-computations for similar requests
Low-latency Inference Data Transfer Library	Accelerate GPU-to-GPU communication to enhance user experience
KV Cache Manager	Preserve GPU memory by offloading context (KV$) to cheaper storage

Demo with AI perf

AI Configurator

AIConfigurator for DisAgg vs Agg Performance

```
pip install aiconfigurator

aiconfigurator cli
  –model QWEN3_32B
  –system h200_sxm –total_gpu 512
```

```
********************************************************************************
*                      Dynamo Configurator Final Results                       *
********************************************************************************
  ----------------------------------------------------------------------------
  Input Configuration & SLA Target:
    Model: QWEN3_32B (is_moe: False)
    Total GPUs: 512
    I/O Length (tokens): Input=4000, Output=500
    SLA Target: TTFT <= 300.0ms, TPOT <= 10.0ms
  ----------------------------------------------------------------------------
  Overall best system chosen: disagg at 862.95 tokens/s/gpu (2.36x better)
    - Agg Actual Best: 365.24 tokens/s/gpu  101.23 tokens/s/user | TTFT: 185.09ms TPOT: 9.88ms
    - Disagg Actual Best: 862.95 tokens/s/gpu  107.79 tokens/s/user | TTFT: 244.45ms TPOT: 9.28ms
  ----------------------------------------------------------------------------
  Pareto Frontier:
               QWEN3_32B Pareto Frontier: tokens/s/gpu vs tokens/s/user
       ┌────────────────────────────────────────────────────────────────────────┐
1850.0-┤dd Disagg                                                               │
       │aa Agg                                                                  │
       │XX Best                                                                 │
       │                                                                        │
1541.7-┤    a                                                                   │
       │    aa                                                                  │
       │      aaadddddddd                                                       │
       │        a      d                                                        │
1233.3-┤         aa     dddddddd                                                │
       │          aa            d                                               │
       │            aa           dddddddd                                       │
       │              aaa               dd                                      │
 925.0-┤                 a                ddddddddX                             │
       │                  a                       ddd                           │
       │                   aaa                       d                          │
       │                     aaa                      d                         │
 616.7-┤                        aaa                    d                        │
       │                           aaaaa                ddd                     │
       │                                aaaaa              d                    │
       │                                     aaa            dd                  │
 308.3-┤                                        aaaa          ddd               │
       │                                            aaaaaaaa                    │
       │                                                    aaaaaaa             │
       │                                                                        │
   0.0-┤                                                                        │
       └┬─────────────────┬─────────────────┬────────────────┬─────────────────┬┘
        0                 45                90               135               180
tokens/s/gpu                         tokens/s/user
```

Key Takeaways

🔍 Summary

NVIDIA enables scalable, efficient LLM inference through:

- KV caching and prefix reuse to reduce compute.
- Disaggregated serving to optimize prefill and decode separately.
- TensorRT-LLM and NVIDIA Dynamo for high-performance, distributed inference.

shabert@nvidia.com