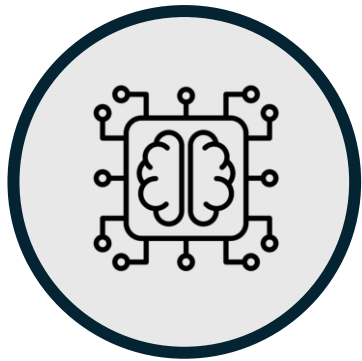


Storage Scale for Training and Inference



Training
&
Inference

+



IBM Storage Scale

- Vasily Tarasov, Yue Zhu, Jonathan Turner, Jeremy Cohn, Frank Schmuck, Animesh Trivedi, Radu Stoica, Haris Pozidis, Tom Parnell, Alex Merenstein, Frank Schmuck, Marc Eshel, Lei Pan, Leo Luan, Thanh Pham, Veera Deenadhayalan, Scott Guthridge, Marc Dombrowa, Ali Sydney, Alim Alim, Bengi Karacali-Akyamac, Rick Welp, Seelam Seetharami, Sophia Wen, Swami Sundararaman, Talia Gershon

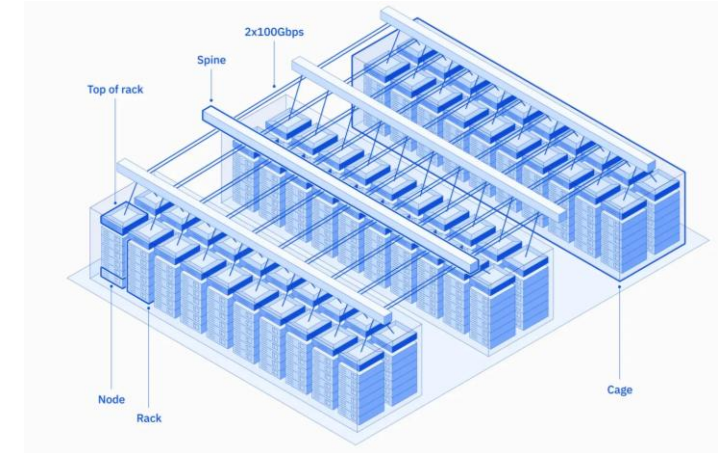
IBM Research - Hybrid Cloud Infrastructure

- Kevin O'Connor, Abdoulaye Traore, Chris Laibinis, Brent Wolfe, Carlos Fonseca
IBM Research - Emerging Technology Engineering
- Guy Margalit, Piyush Chowdhary, Khanh Ngo
IBM Storage – CTO office
- Chirstof Schmidt, Anthony Hsu
IBM Storage – Scale
- Brian Reitz, Steve Pritko, Piyush Shivam
IBM Cloud – Block Storage

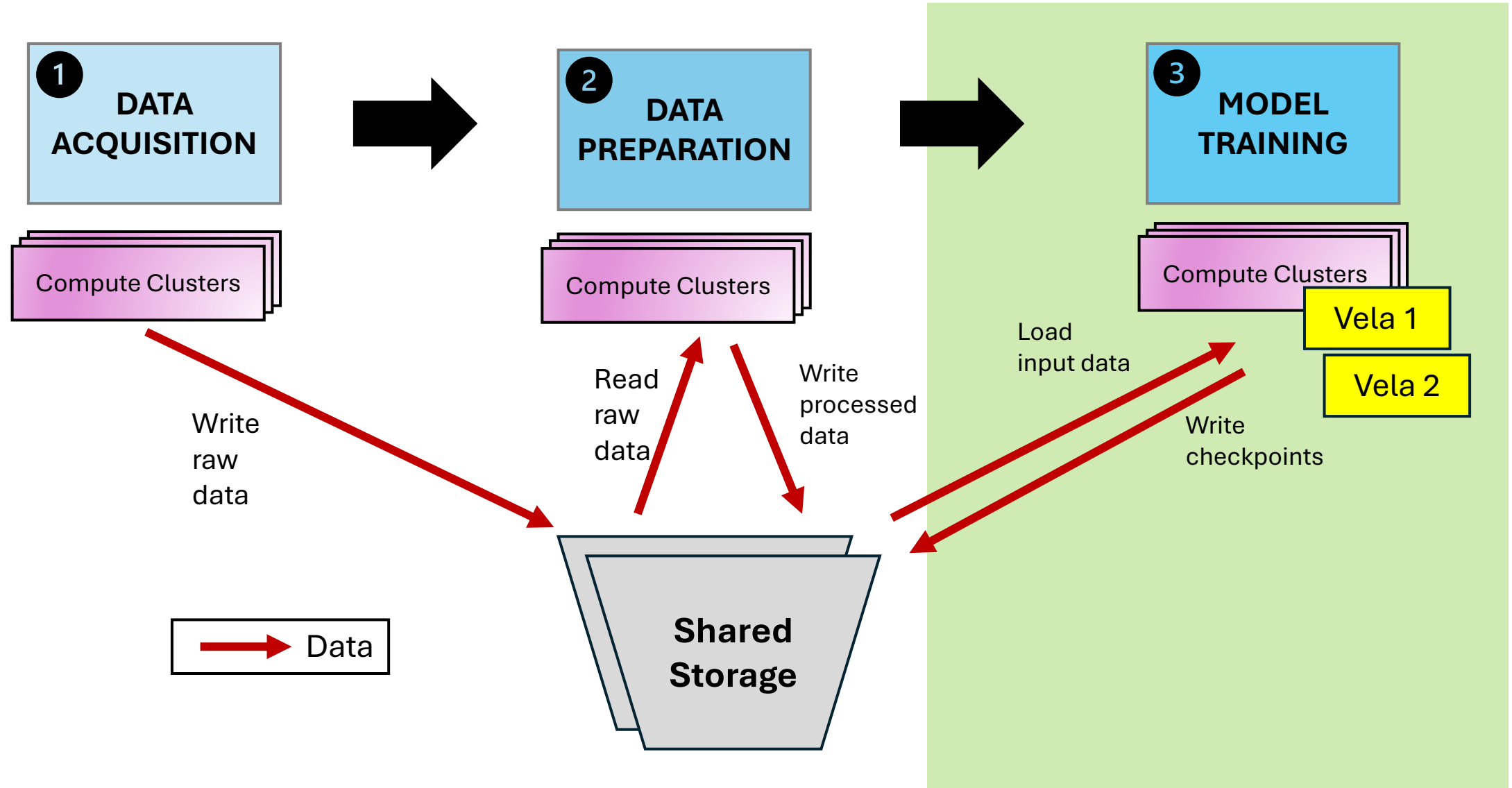
Training

Cloud Vela(s): Cloud-native AI Training Cluster(s)

- IBM Research needs infrastructure to train ML models
 - Large Language Models (LLMs) have billions of parameters
- Can we build a **cloud-native** training cluster?
- Of-the-shelf GPU-rich host servers added to IBM Cloud
 - 200 nodes, 8× NVIDIA A100 / 60 nodes, NVIDIA H100 80GB
- KVM-based Virtual Machines as building blocks
- Standard Ethernet networking
 - RoCE for GPU-GPU communication
- Red Hat OpenShift (OCP) for resources and training job management
 - MLBatch / Kueue MLBatch queuing and quota management system
<https://github.com/project-codeflare/mlbatch/blob/main/CODEFLARE.md#mlbatch-for-codeflare-users>
- IBM Granite models
 - <https://huggingface.co/ibm-granite>



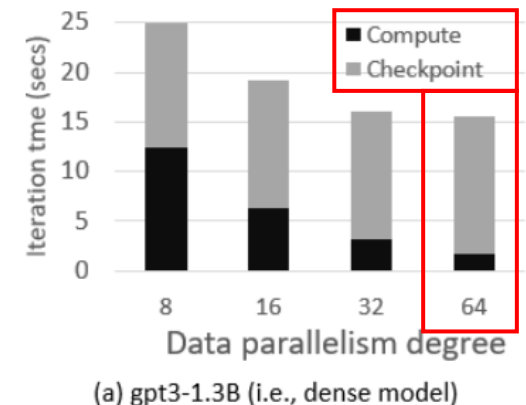
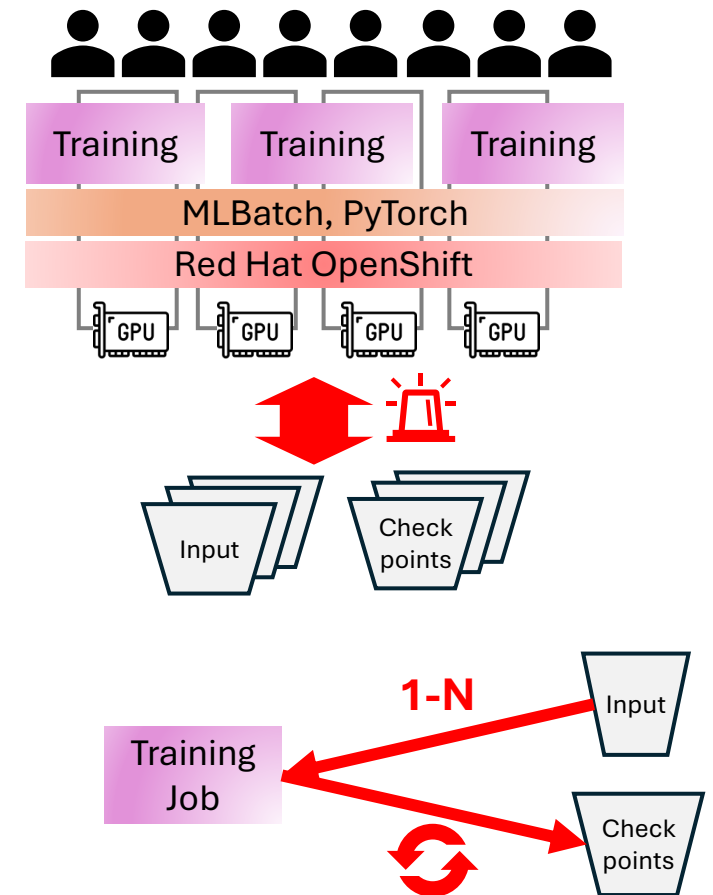
Data Access in AI workflows



Storage for Model Training

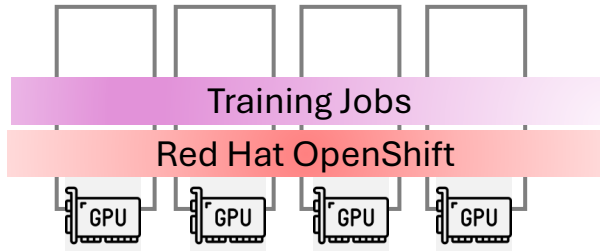
- Multiple users run concurrent training jobs
 - Some jobs for weeks and months
- Native shared storage services in IBM Cloud
 - **IBM Cloud Object Storage (COS)**
 - IBM Cloud File Storage (NFS) – 1GB/s per share only
 - IBM Cloud Block Storage – no shared namespace
- Typical training job uses
 - 1 bucket for input data – read all input data a few times
 - 1 bucket for checkpoints – many periodic writes, infrequent reads
- Increasingly large datasets
 - Input data (more tokens): 1TB → 20TB
 - Checkpoints (larger models): 100GB → 1TB
- **Problems**
 - ✗ Slow checkpointing
 - ✗ Slow training data loading
 - ✗ Storage backend overload
 - ✗ Slow restore from checkpointing

- Extended training times
- Idling GPUs
- Can't perform checkpoints as frequently as desired



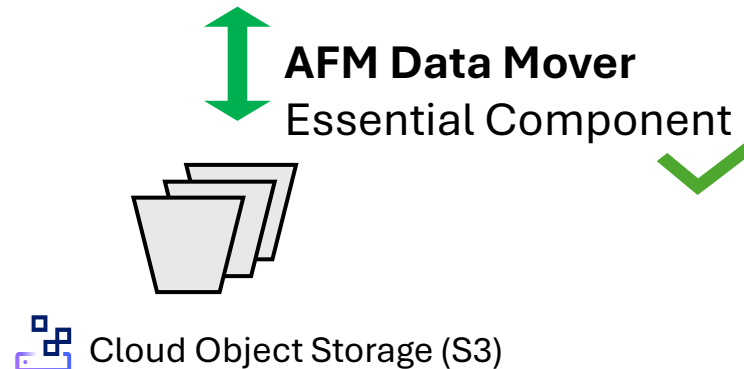
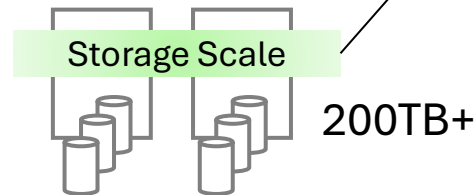
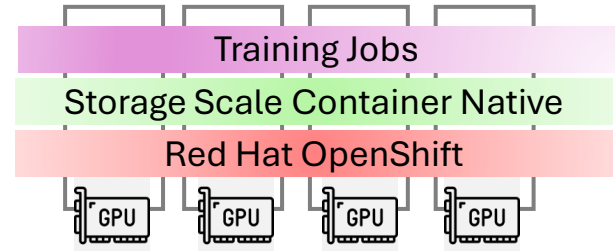
Multi-user Elastic Cache w/ Storage Scale

BEFORE



 Cloud Object Storage (S3)

NOW



 Cloud Object Storage (S3)

**Large, persistent,
and high-performance
cache dedicated
to the AI cluster**

- ✓ 1. Consistently fast checkpoints
- ✓ 2. Fast data load from cache
- ✓ 3. No backend overload

- ✓ 4. Automated data movement

Users now call it
"Infinite file system"

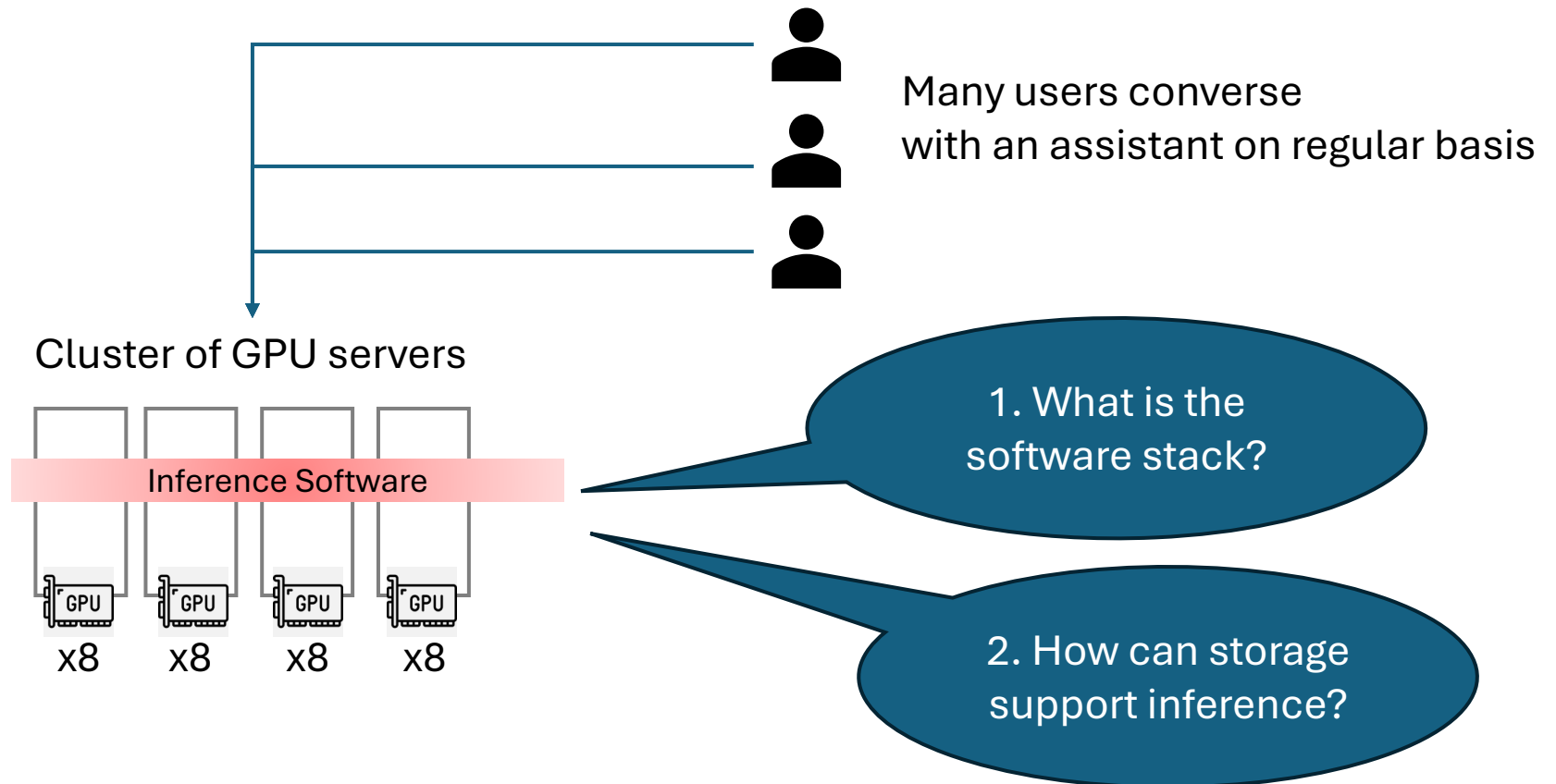
Notable facts on how we use Scale

- We run multiple clusters over 2 years, in different regions, approaching 1PB in total
- Virtual instances for storage servers, not baremetal
- New, Ceph-based, networked block storage
- 16-node cluster, 96 block volumes, 240TB, 96GB/s throughput
- No file system replication, no ECE
- Multi-Rail over TCP (MROT) to use multiple vNICs
- Largest Container Native Storage Access (CNSA) cluster
- AFM and Scale Cache Volumes (since version 5.2.3)
 - Users can create PVCs that refer to the bucket that needs to be cached
 - PoC that integrates data prefetch with Job scheduler
- PoC of Scale as a Managed Service

Inference

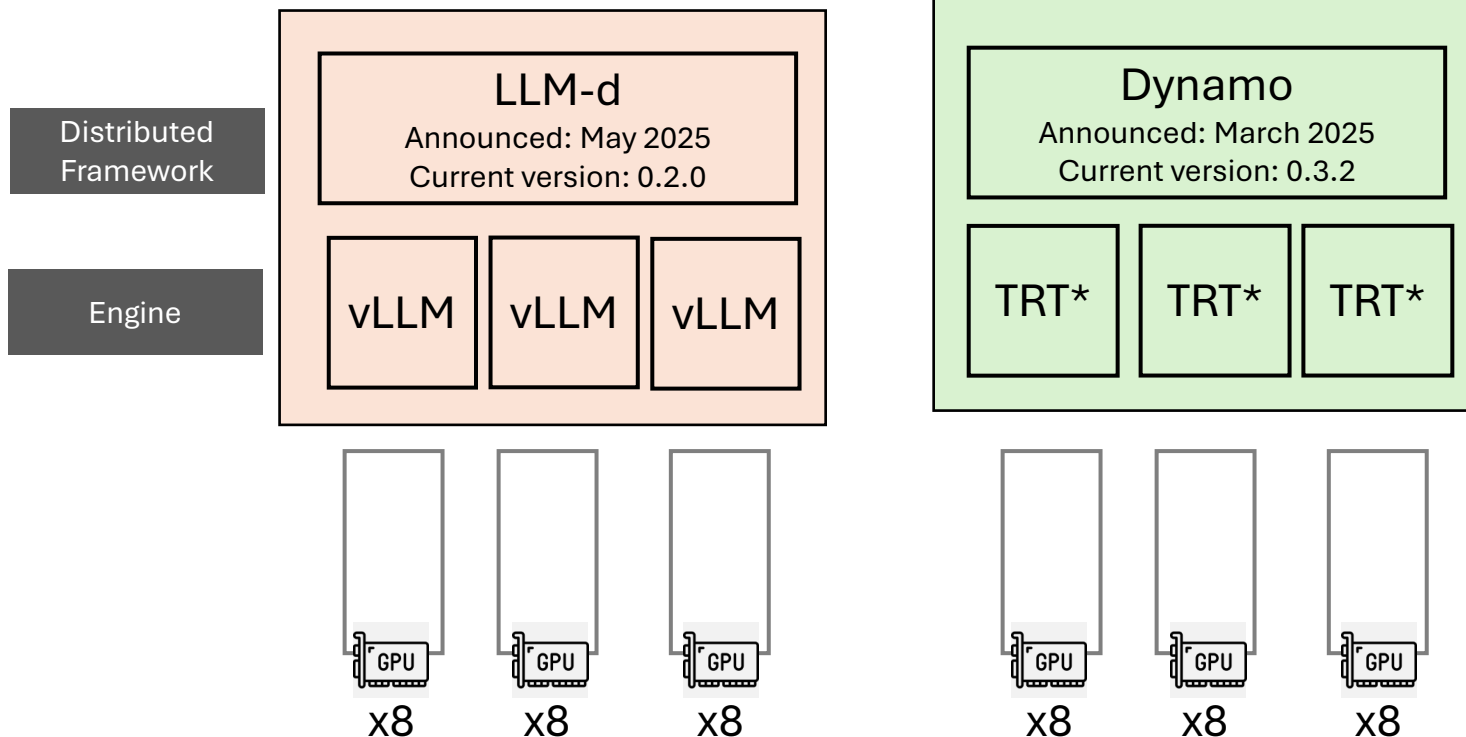
Running Example

- An organization is building an inference cluster
- Target use: conversations w/ personal assistant or therapist



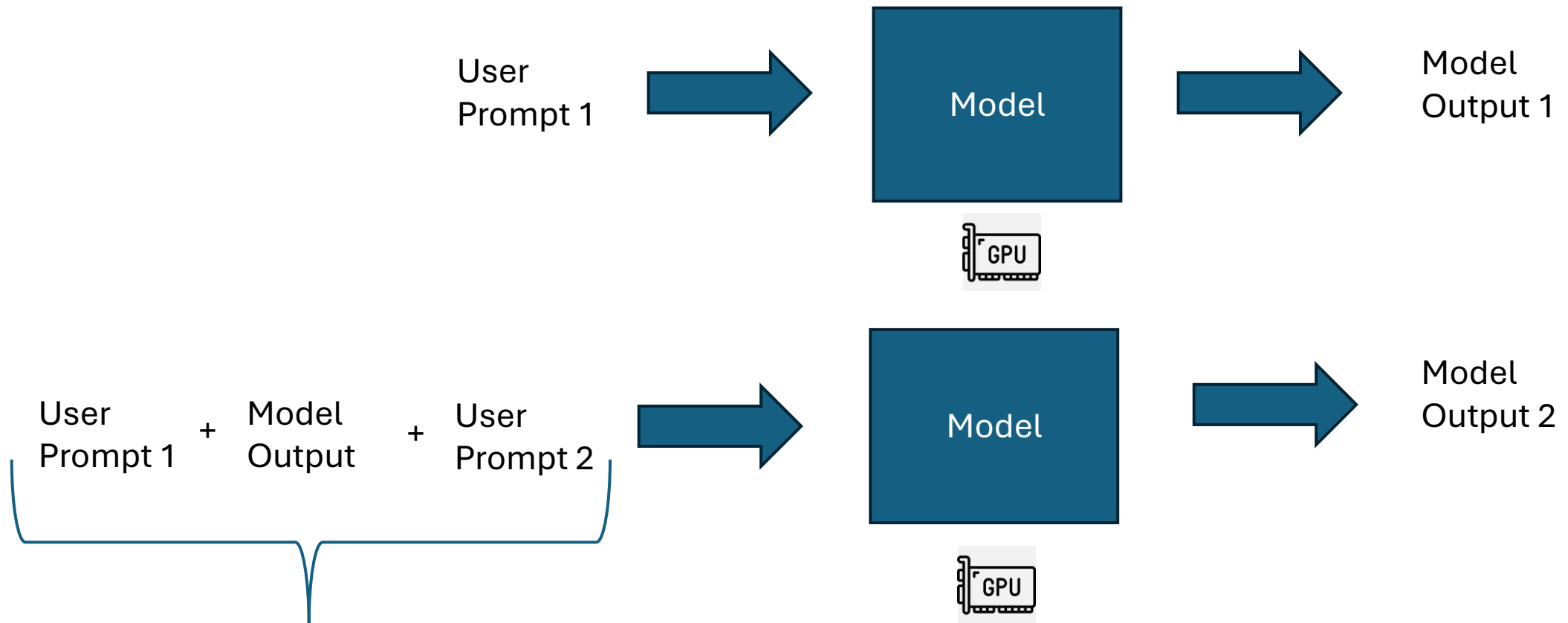
Inference Stacks

Two major alternative inference stacks under **active** development



- Maximize overall cluster inference throughput (TPS) while providing latencies within required SLAs (TTFT, TBT)
- Inference Gateway
- Smart request routing
- Disaggregated prefill and decode
- Dynamic GPU scheduling
- Accelerated data transfer

Prompt size continuously grows



Input grows as the conversation continues!

Context window is the limit: 100K – 1M tokens

Tokenization and Embedding

- Tokenization: text converted to a stream of tokens

- Vocabulary: 10K-100K tokens
- <https://tokenizer.model.box/>

- Embeddings

- High-dimensional vectors: 1K to 10K of int32
- Represent “the semantics” of the word
- Models include embedding matrix

How is the weather today in San Jose?

4438, 374, 279, 9282, 3432, 304, 5960, 11097, 30

(for Llama-3-8B)

123	213	554	670	348	908	498	798	135
484	800	345	345	798	987	109	490	194
.
.
.
808	324	123	942	213	155	908	150	841
990	098	598	123	412	129	142	890	185

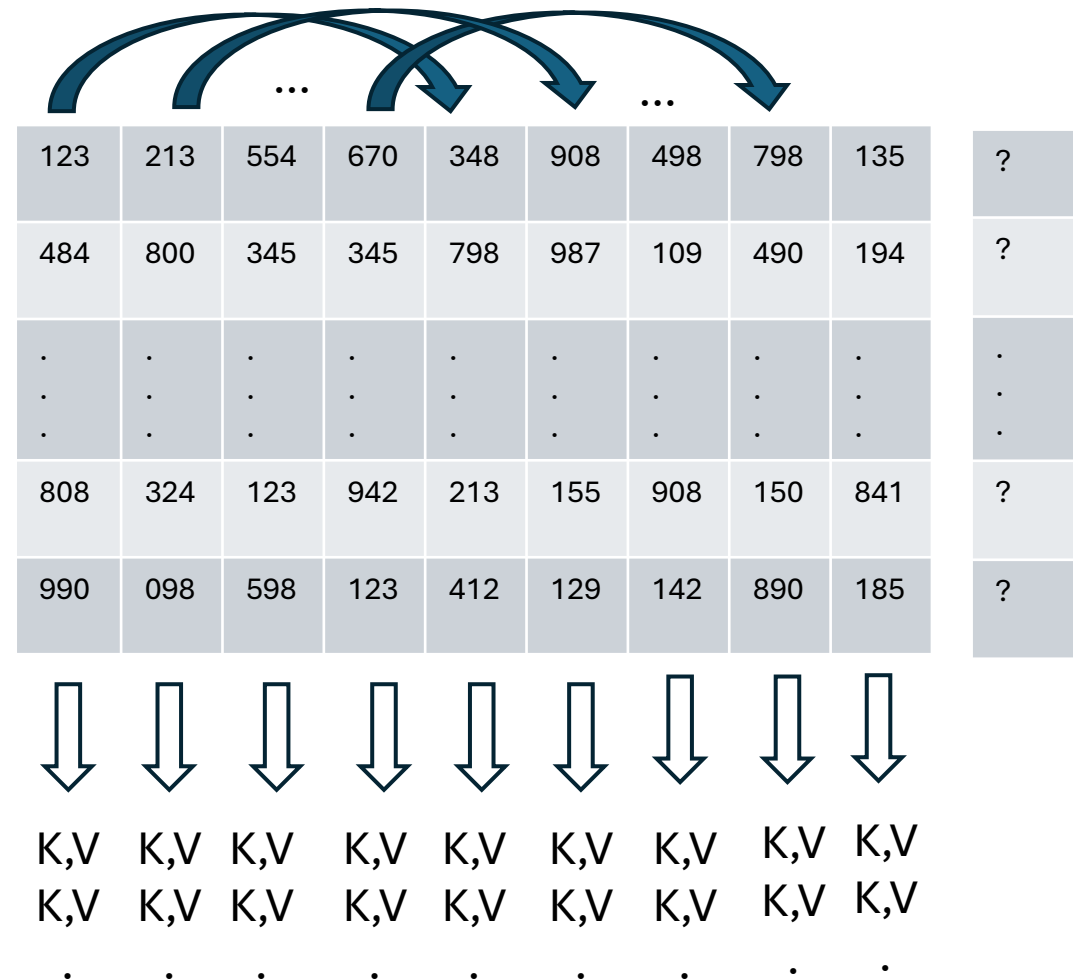
Prefill and decode

- Embeddings are **contextualized** within the context window
 - Vector values change depending on the surrounding vectors
- Self-attention mechanism
- Multiple K and V matrices computed for each embedding
 - Multiplications of embeddings and weight matrices
 - Computationally intensive, parallelizable
- Llama-3-8b, 100K tokens, **12GB**
 - https://lmcache.ai/kv_cache_calculator.html
- Decoding stage
 - Predict next token using Ks and Vs

How is the weather today in San Jose?

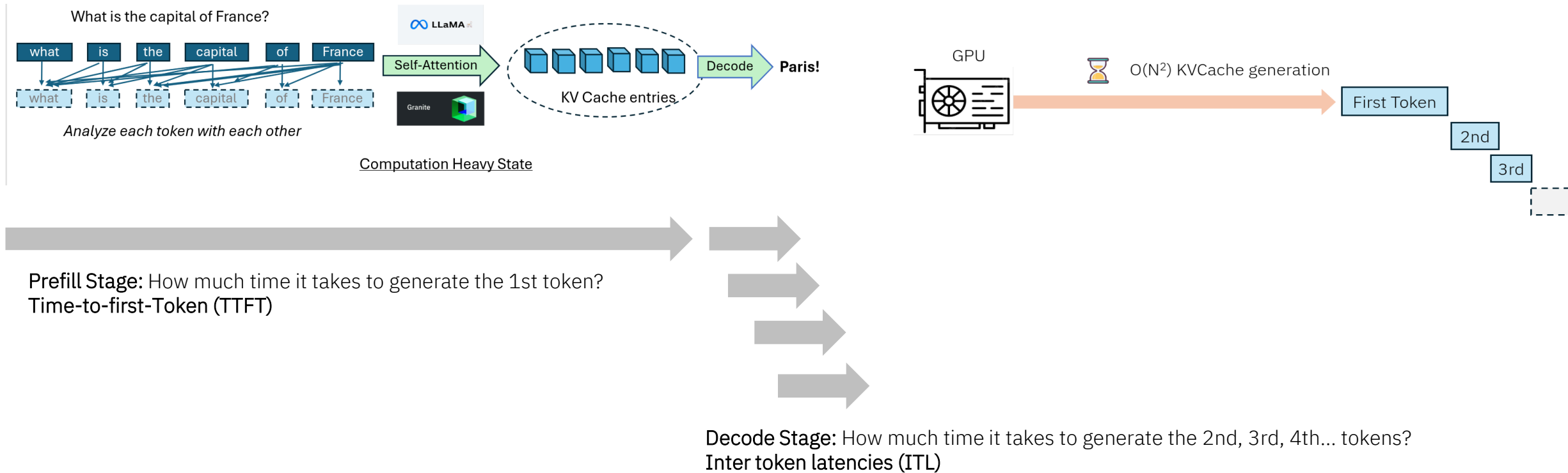
4438, 374, 279, 9282, 3432, 304, 5960, 11097, 30

(for Llama-3-8B)



KV Cache

- KV Cache is the "**Runtime State**" of an inference request - Computationally expensive to build

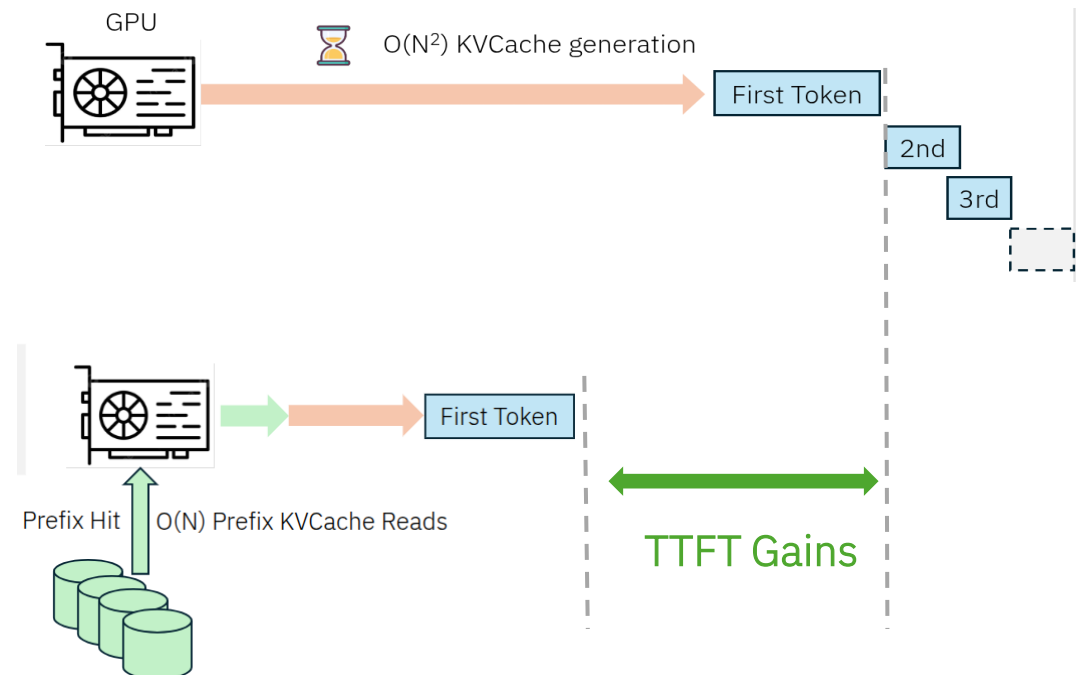
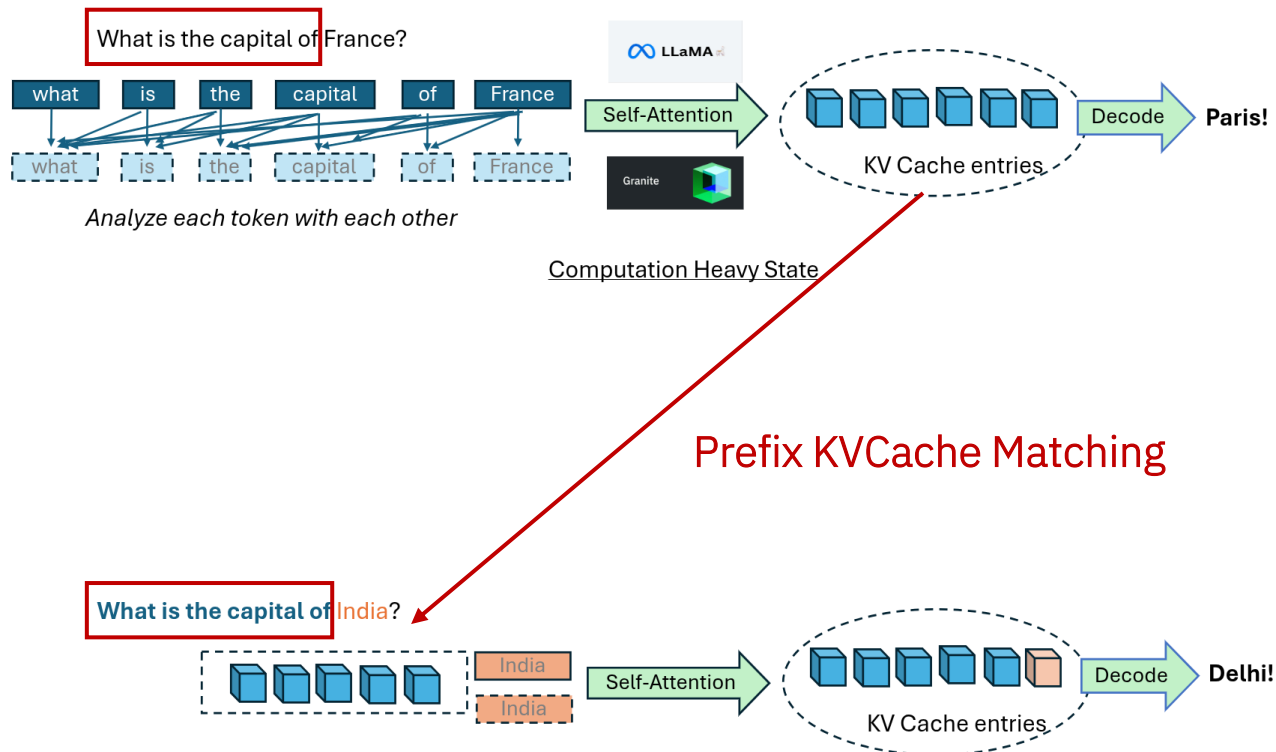


KV Cache Management

- For the duration of prefill and decode
 - Keep in GPU memory
- After the active computation is done?
 1. Discard
 2. Keep in GPU memory – if the user returns or shared prefix
 3. Offload to CPU memory, local storage, remote storage
- (Re)computation of inference state (KV Cache) gets more expensive
 - Growing model sizes and context windows
 - Agentic computing, multi-turn conversations

Benefits of KV Cache Offloading

- Prefix Cache stores KV Caches for future re-use
- KVCache is the "**Runtime State**" of an inference request - Computationally expensive to build



"What if" Analysis : Opal Simulator

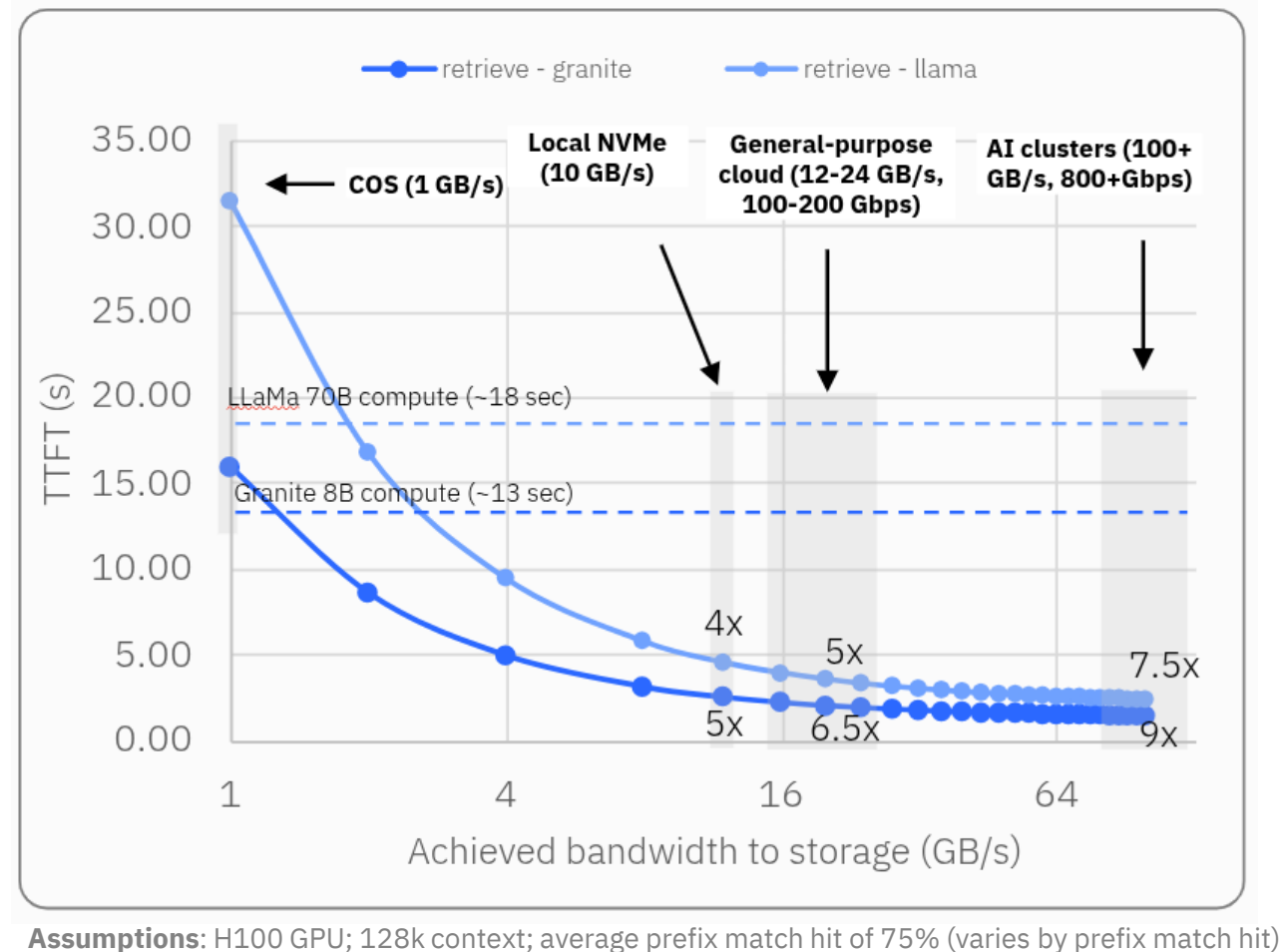
Large performance design space, multiple hardware choices, deployment scenarios

What is the potential performance gain if one has

- GPUs: H100, A100, L40s...
- Storage BW: 1, 10, 100 ...GBps
- Model: Llama, Granite ...
- Prompt and prefix sizes ...

Goals:

- Identify high-gains combinations quickly
- Demonstrate gains in practice with the IBM stack



High-Performance Storage Delivers Performance Gains with Lower TTFT by 4 - 9x

Inference Stacks



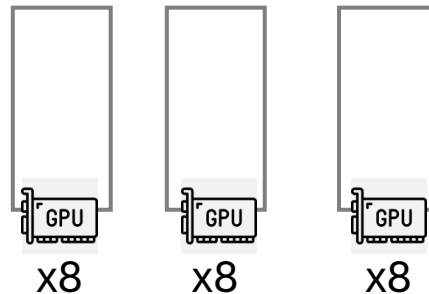
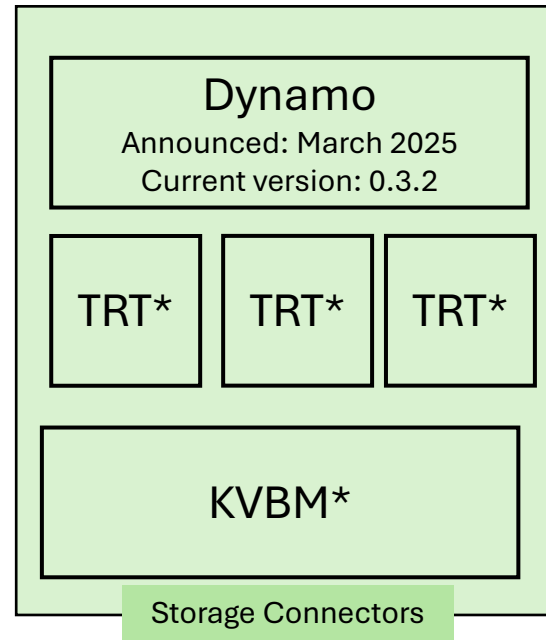
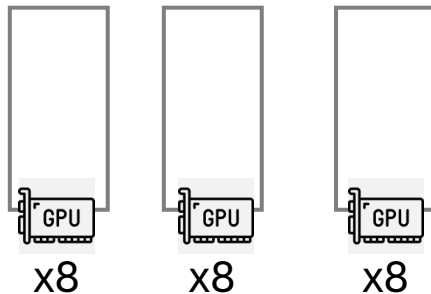
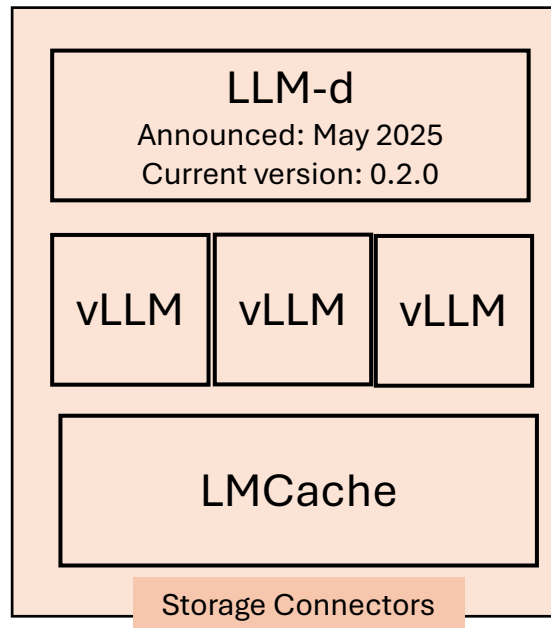
Red Hat “Red Stack”



NVIDIA “Green Stack”

Distributed
Framework

Engine



- Each stack includes technologies for KVCache offloading – LMCache and KVBM
- Each stack introduces the concept of storage connectors to work with different storage vendors – from generic file/object to storage specific.

Using Scale for KV Cache Offloading

- Inference frameworks (LLM-d, Dynamo) support file system connectors
- Just mount Scale and specify the mount point as a place to store KV Cache
- CNSA for K8s-native frameworks
- GPU Direct Storage supported
- The fact that the file system is distributed allows frameworks to route requests more freely – better TPS

Preliminary results

