SNIA DEVELOPER CONFERENCE

SDC 25

By Developers FOR Developers

September 15-17, 2025

Storage Implications for the New Generation of AI Applications

CJ Newburn, Distinguished Engineer, NVIDIA GPU Cloud

Wen-mei Hwu, Senior Distinguished Researcher, NVIDIA Research

Storage architectural context

- Storage parameters
- Node in a cluster
- EW vs. NS, RDMA

Storage parameters

What to consider when you create an architecture for storage

- Location: bandwidth Connectivity
  - PCIe peer-peer
  - NS, EW; RDMA

- Criticality
- Capacity
- Throughput (IOPS)
- Location: latency
- Buffering
- Granularity
- Power
- Computing near storage

Driven by apps

Storage in data center architecture

Different concerns and opportunities for each part of the data center storage architecture

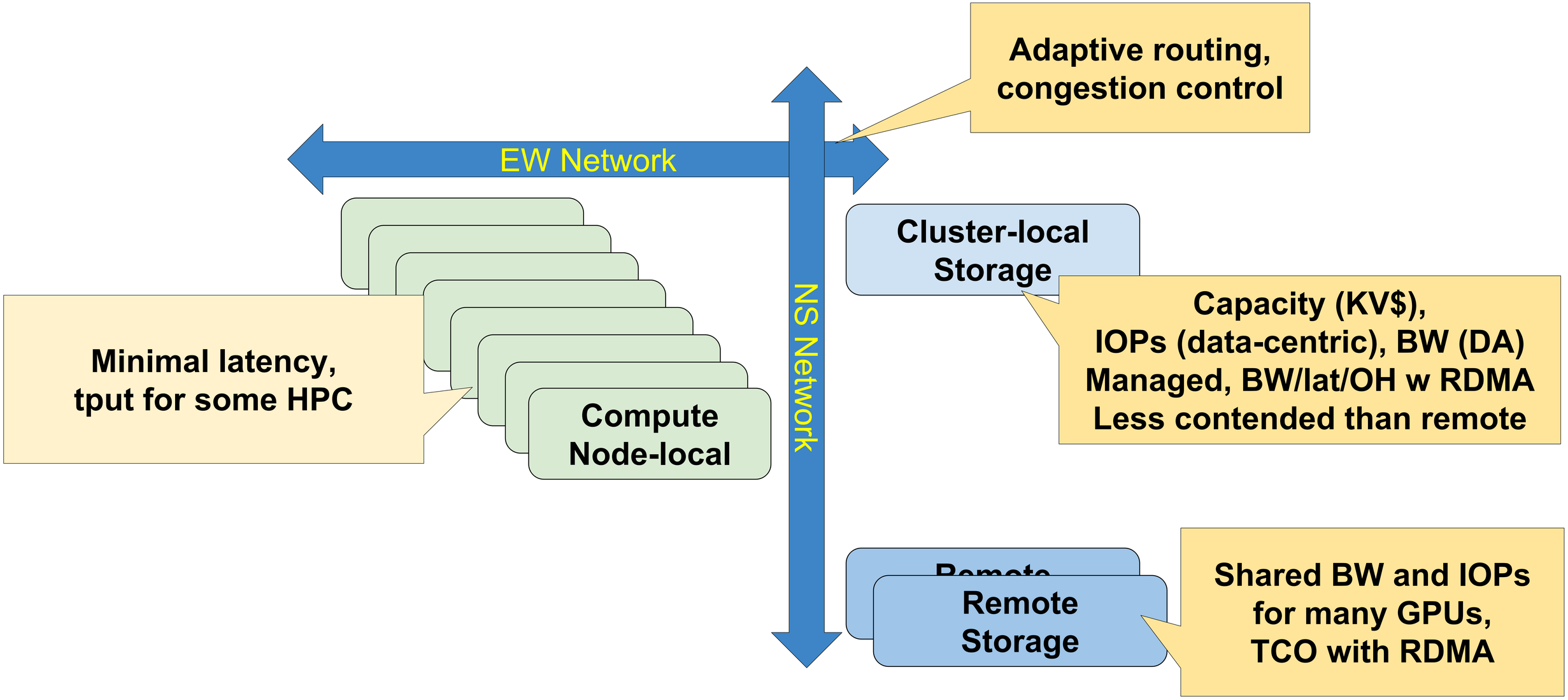


Illustration: Local vs. North-South vs. East-West RDMA

Consider separating 1) EW comms to “memory” and 2) NS for infra control and legacy storage

- Can’t go faster than Gen5 in a CPU today
- NS NICs from Gen5 match @ 400 Gbps
- NVMes hanging off CPU are Gen5

- GPUs are Gen6 (B300)
- EW NICs could match Gen6 @ 800 Gbps
- RDMA obviates the need to use CPU for TCP
- Gen6 NVMes in cluster-local storage can be accessible over RDMA via Gen6 NIC

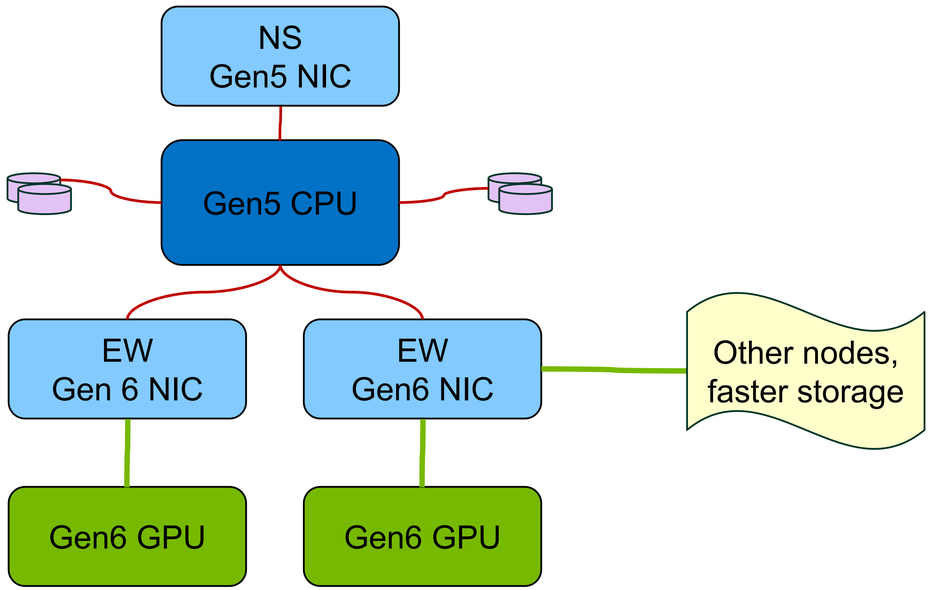
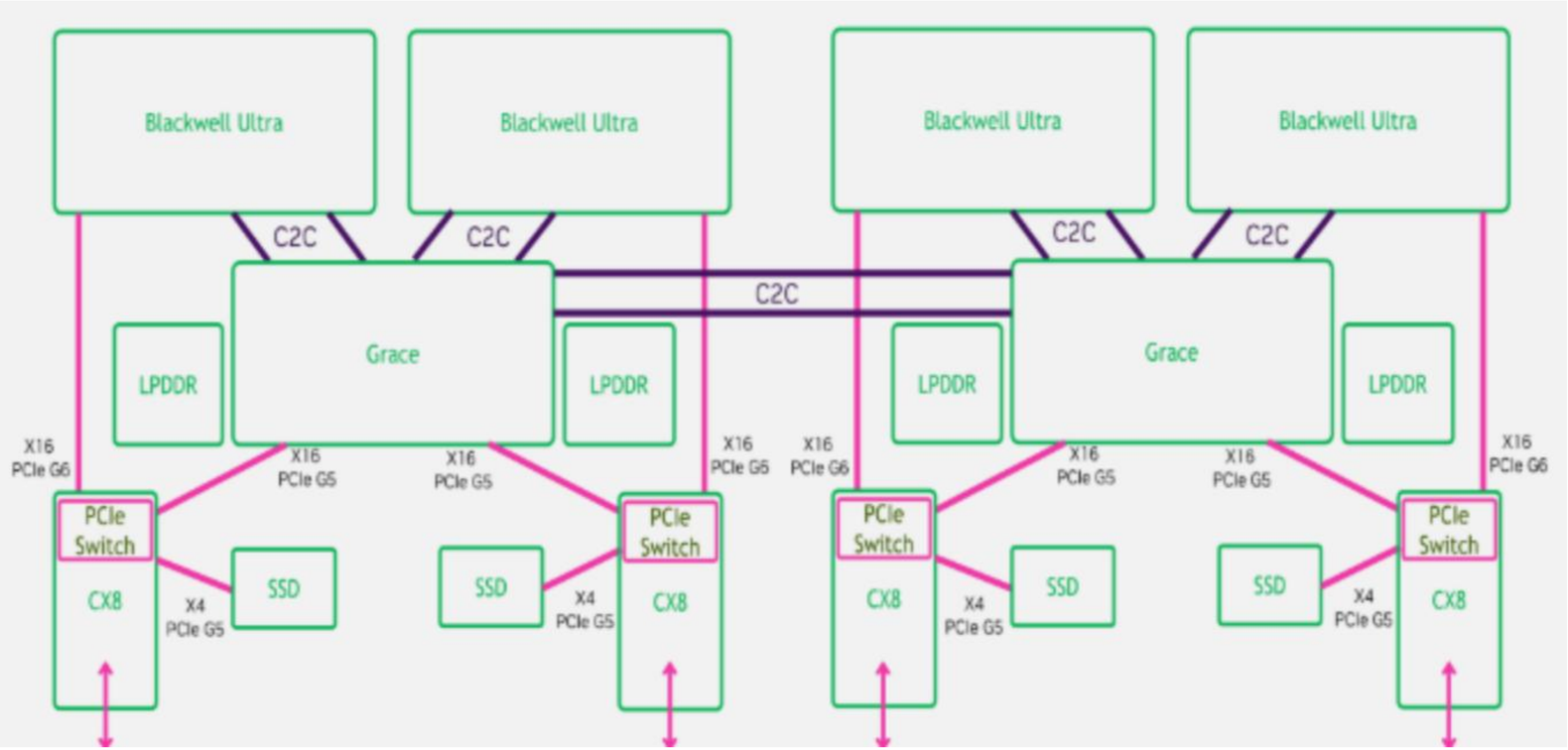


Illustration: GB300 node

800 Gbps access to network, NVMes



- Gen6 matches 800 Gbps; supports 100 GB/s, 200 MIOPs @ 512B
- **GPU**-CX8 on Gen6; supplied with and qualified for 4:1 NVMe:GPU Gen5
- The above is 2x of Grace **CPU**-NVMe/NW on Gen5 with CX7

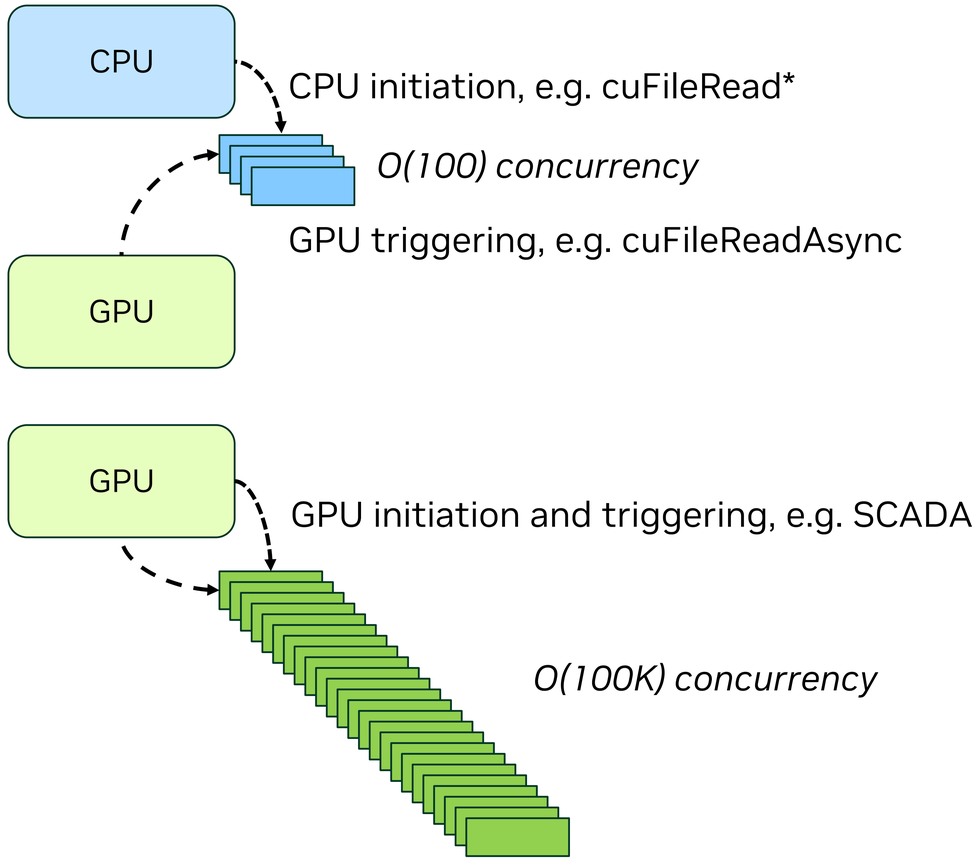
Interfaces to storage

- Copy into memory for sparse access
- Direct access datasets of unbounded size
- Computing elements
- CPU vs. GPU initiation

Work flow

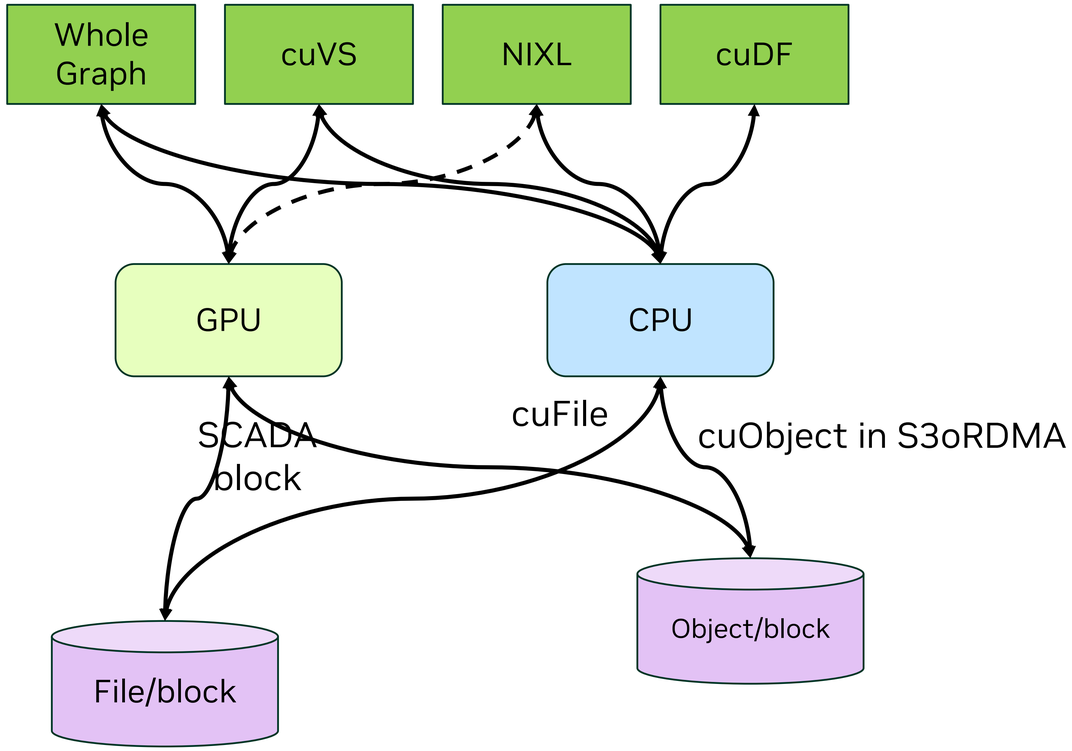
GPUs support 2-3 orders of magnitude more concurrency

- Initiation
  - Create work queue entry
  - Populate it with what to access
  - Populate data buffer for writes
  - Triggering from GPU enables >> concurrency
- Triggering
  - Unblock access and make data ready
  - Triggering from GPU minimizes sync overhead
- Completion
  - Processing completion queue entries from storage
  - Make data available to compute for reads
  - Often even more limiting for speed than initiation
  - Implicitly associated with initiating device

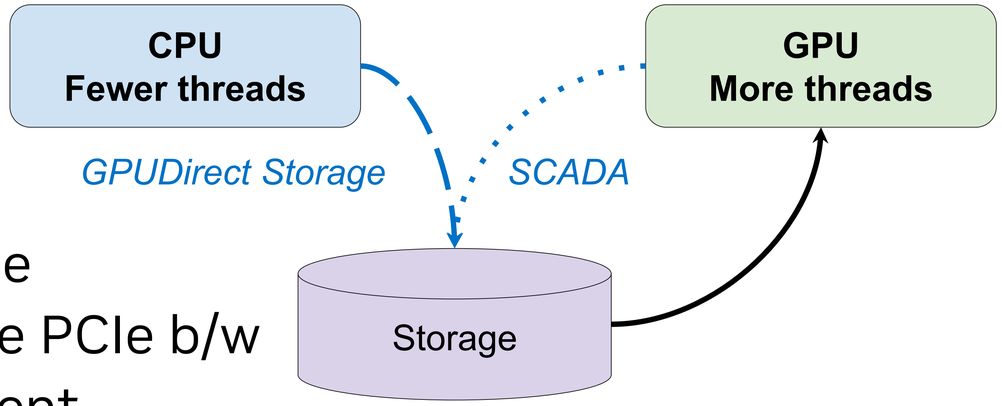


Application interfaces to storage

- CPU-initiated
  - cuFile - file
  - cuObject - S3oRDMA
- GPU-initiated
  - Block, but interoperable – SCADA
  - SCADA API now - contiguous linear
  - SCADA API WIP - key value
- Abstractions
  - RAPIDS.cuDF loader
  - NIXL for inference data and more
  - cuGraph.WholeGraph for GNN embedding data
  - cuVS for vector data and vector index
  - Thrust for CUDA C++ applications
  - …



Data access tech overview: initiators



- CPU-initiated - GPUDirect Storage (cuFile/cuObject)
  - Predictable - set of accesses is known ahead of time
  - Transfers of n pages – O(10) of threads saturate the PCIe b/w
  - Fewer threads, large IO size → lower IOPs requirement
  - Prefetch of content to GPU memory, while compute happening on previous batch

- GPU-initiated – SCADA
  - Compute-driven – requests identified as part of compute, known only to each GPU thread
  - Fine-grained – O(100K) threads busy saturating PCIe
  - O(100K) threads, small IO size → high IOPs: 100 GB/s @ Gen6 / 512B = 200 MIOPs
  - Leverage SIMT latency tolerance and HBM's bandwidth for fast data processing

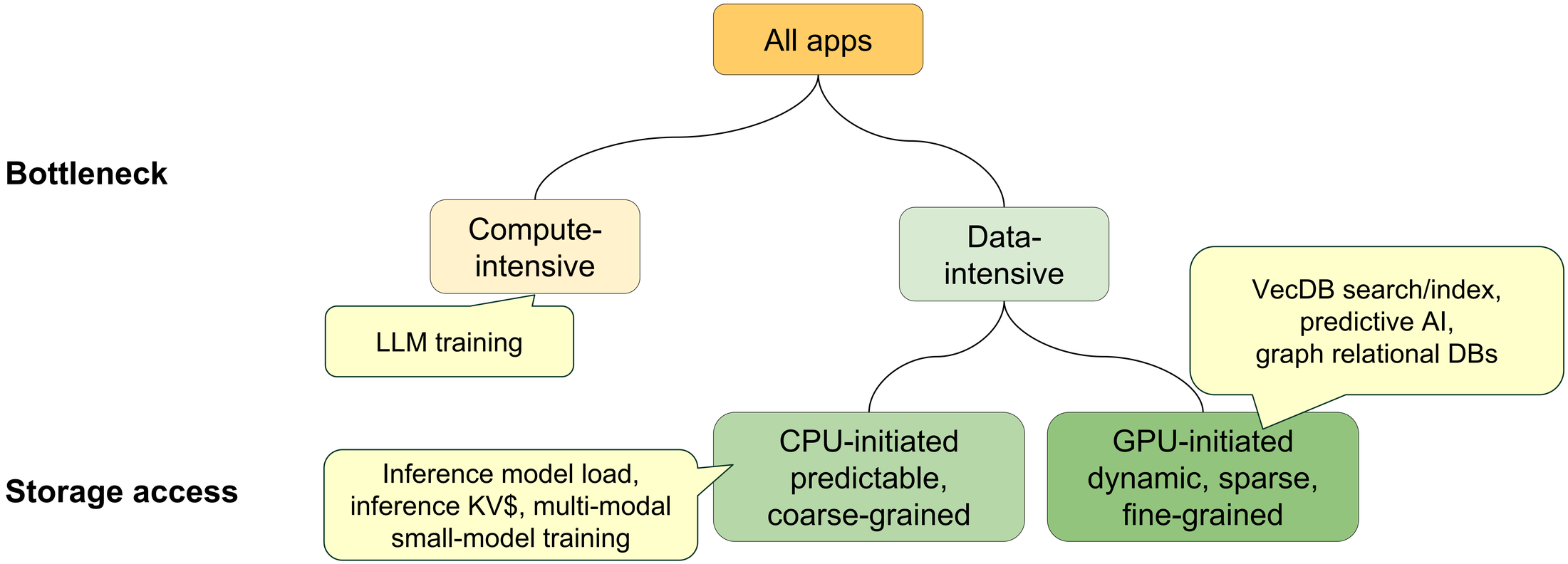
Problems → solutions for concurrency

Ushering in a new era of data-centric computing on large problems with GPUs

SC	Scaled	• O(100) thread CPUs limit concurrency → O(100K) GPU threads • Partitioning → automatic GPU-GPU communication • Memory fabric breaks at scale → introduce an API, support errors
A	Accelerated	• GPUs can't run a file system, can't ask CPU for help → GPU initiation • Can read into registers from IO → buffer in a cache on GPU • CPUs can't tolerate latency → GPUs do • Too many fine-grained accesses → aggregation
D	Data	• Limited capacity in memory → unbounded storage
A	Access	• RAF problem from sparse access to big chunks read in → direct access

App taxonomy: bottlenecks, dynamism, granularity

Designing storage solutions hinges on understanding app requirements



- Traditional focus for GPUs has been compute-intensive apps like training
- The explosion of innovation for VecDB, predictive AI drives new technologies to fill the gaps

Styles of compute node – storage interaction

Compute-intensive apps

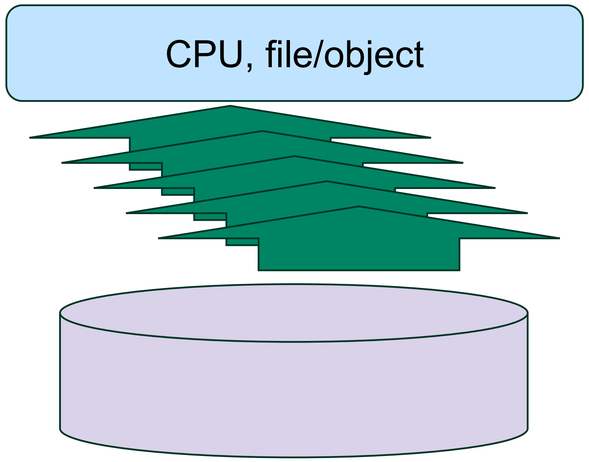
- *LLM training/inference*
- *Generative AI*

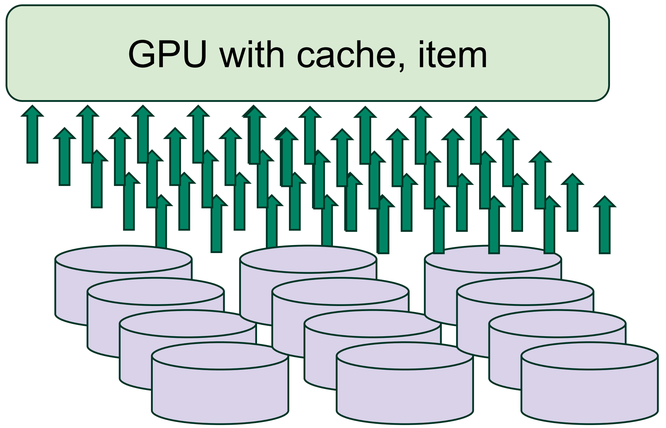
- Working sets fit in memory
- Low bandwidth, not perf critical
- cuFile/cuObject, POSIX, S3
- Coarse grained, standard NVMe
- NAND: large pages, low IOPs
- TB/TCO

Data-intensive apps

- *GNNs, vector DBs, graph relational DBs*
- *Predictive AI, search*

- Working sets spill to storage
- High bandwidth, perf critical
- SCADA
- Fine grained, customize front end
- NAND: many dies*planes, BCH, high IOPs
- IOPS/TCO





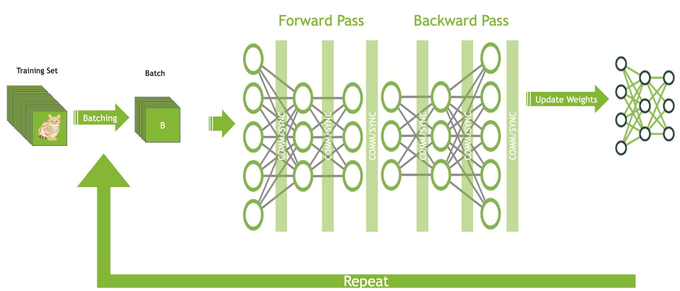
Getting ahead of the trend

Anticipating the needs of emerging usage models while sustaining core volume for legacy

Category	Disruptive trend	What we need to do
Data reach by compute	10TB → 1 PB	Access storage like memory with an API
Throughput and concurrency	O(100)/CPU → O(100K)/GPU Threads accessing data	New software stack with highly concurrent caching, batching, and queuing
Access pattern	Sparse, random on vector data	New storage SKUs optimized for sparse access IOPs/TCO
Granularity	Coarse → fine	
Programming model	GPU autonomy	GPU-initiated/completed, fine-grained (SCADA)

- Breakout out of artificial memory constraints
- But then storage needs to keep up with memory's sparse IOPs

GenAI Applications

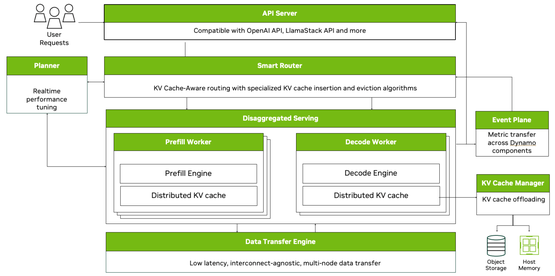


Pretraining

Models (weights and checkpoints)
input dataset (images, text, videos, audios)

Large datablobs
Coarse-grain sequentially accessed

Yesterday

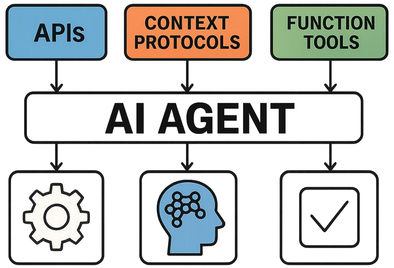


LLM Inference, RAG, vectorDB

KVcache, vector embedding,
vector index, document chunks,
Input dataset, models

Medium to large datablobs
Medium- to Fine-grain semi-randomly accessed

Today



Agentic tasks

Graph, KVcache, vector embedding,
vector indexed search/DB, fine-grain document chunks, Input,
model weight, long-term memory

Medium to huge datablobs
Fine-grain semi-randomly accessed
High throughput, low latency

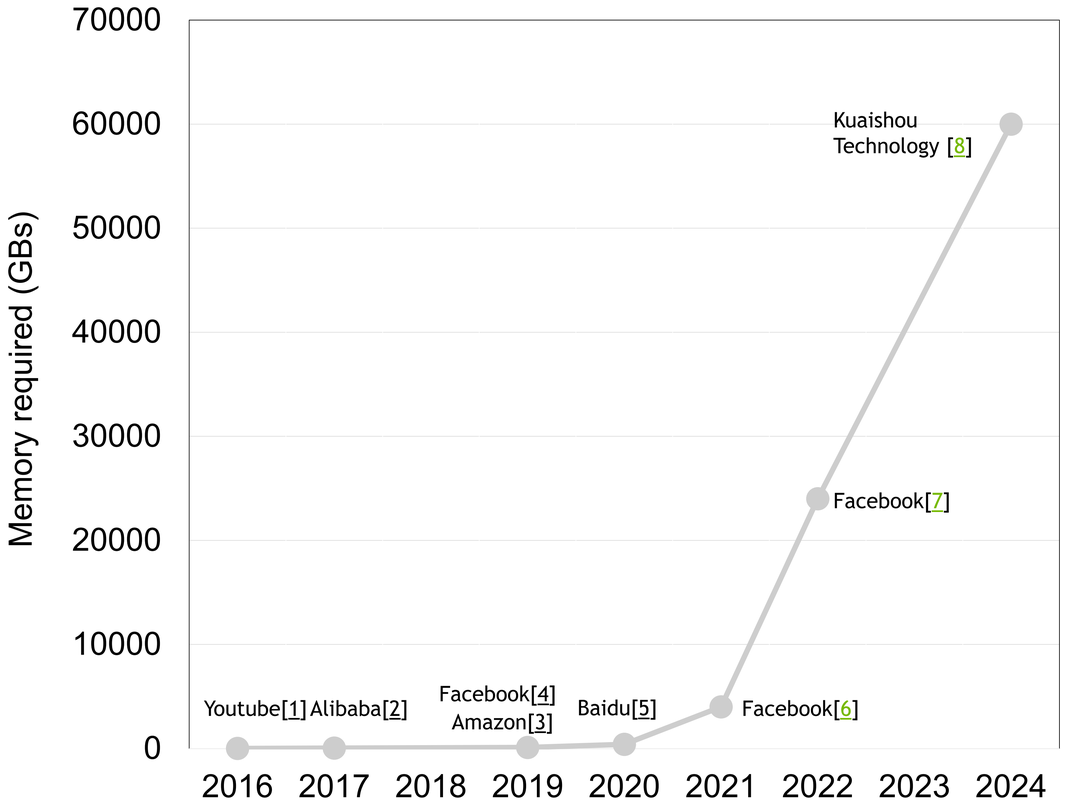
Future

AI apps: generation vs. prediction

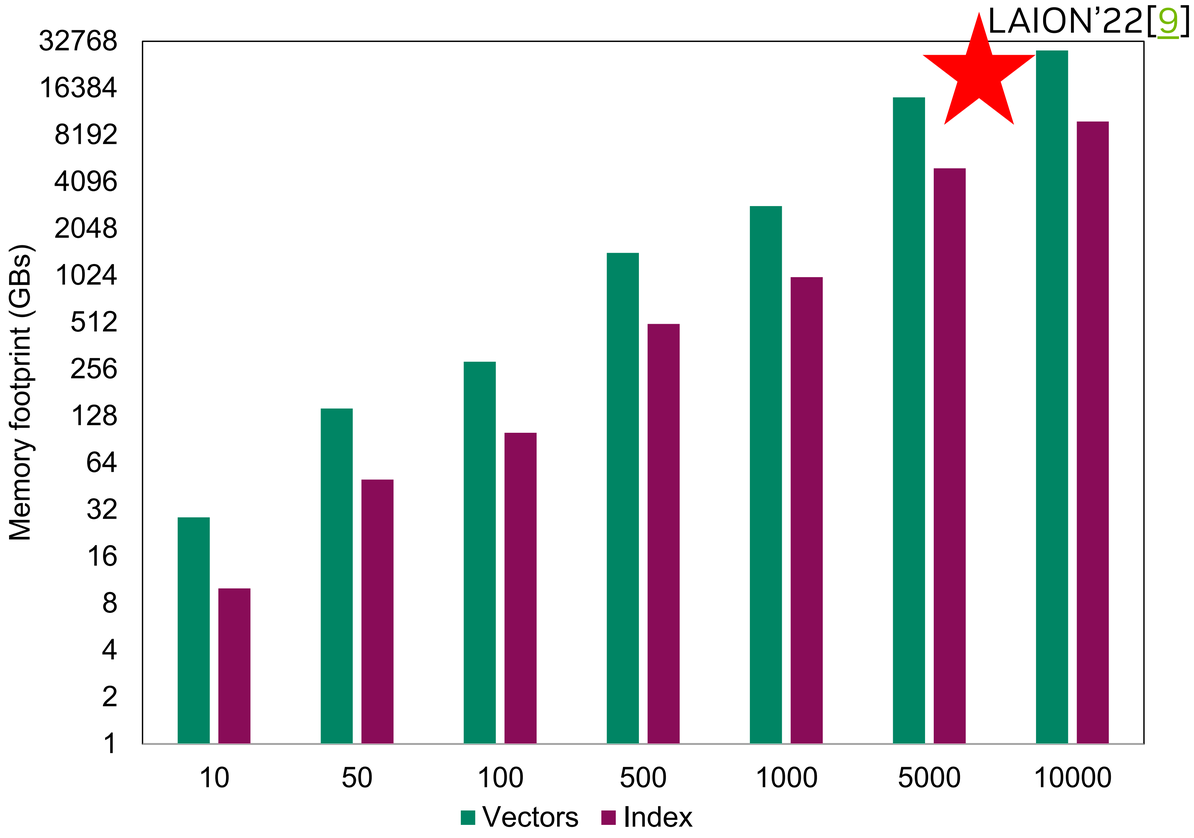
Prediction opens up new venues to revenue

	Generation	Prediction
Output	Query response Generated code Art, e.g. designs, music	Recommendation Filtered selection Trends and anomaly
Usages	Chatbots, reasoning	ECommerce, financial services, social networks, IT
Cost of error	Wrong, hallucinated answer Debugging, escaped bugs Artistic artifact	Mispredictions ok if usually right or better than current approaches
Infrastructure	LLM	GNN, Graph Transformer, Graph Relational DBs
Computational complexity	High	Modest
Data needed in memory	Model weights, KV	Knowledge graphs, graph relational DB, embeddings, model weights
Data needed in storage	KV$	Same as above, for bigger problems
Challenges	Running out of public data	Many new horizons ("big data"), data updates

Emerging workloads demand memory



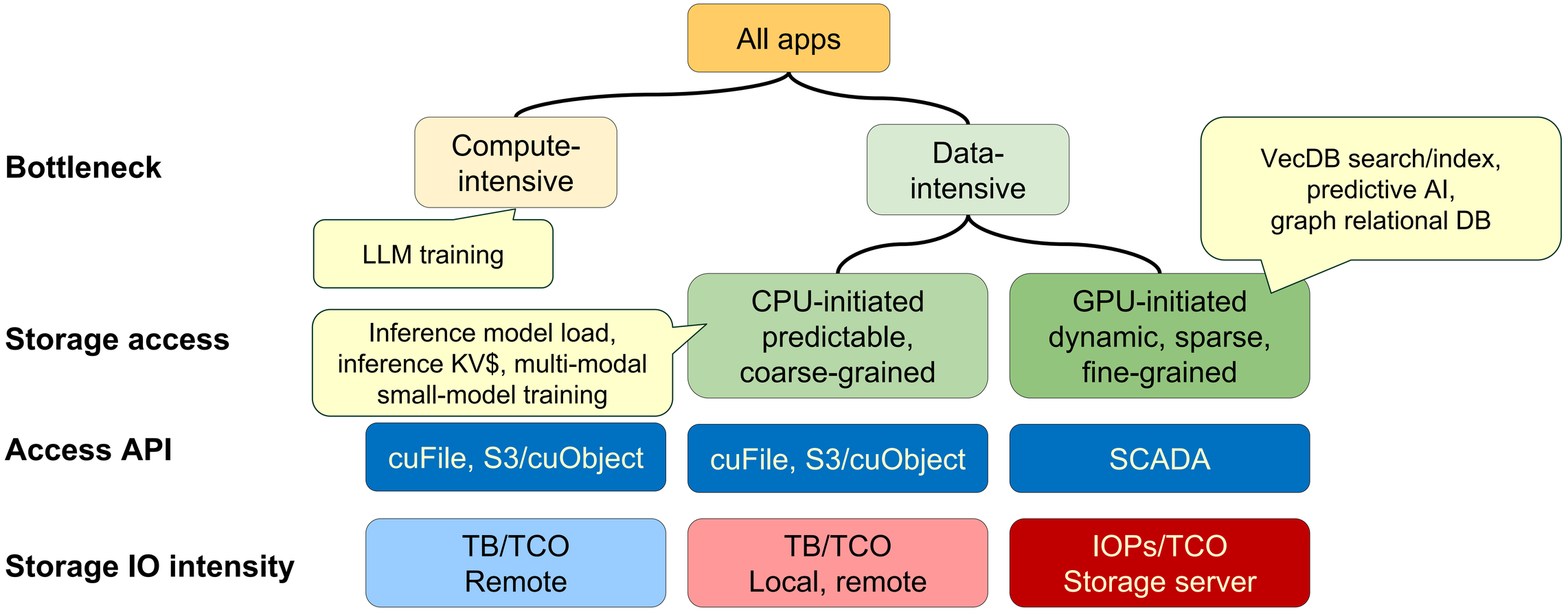
GenAI Recommender Systems
1TB to 100TBs



Vector databases
1TB to 100TBs

App taxonomy: bottlenecks+APIs, intensity

Designing storage solutions hinges on understanding app requirements

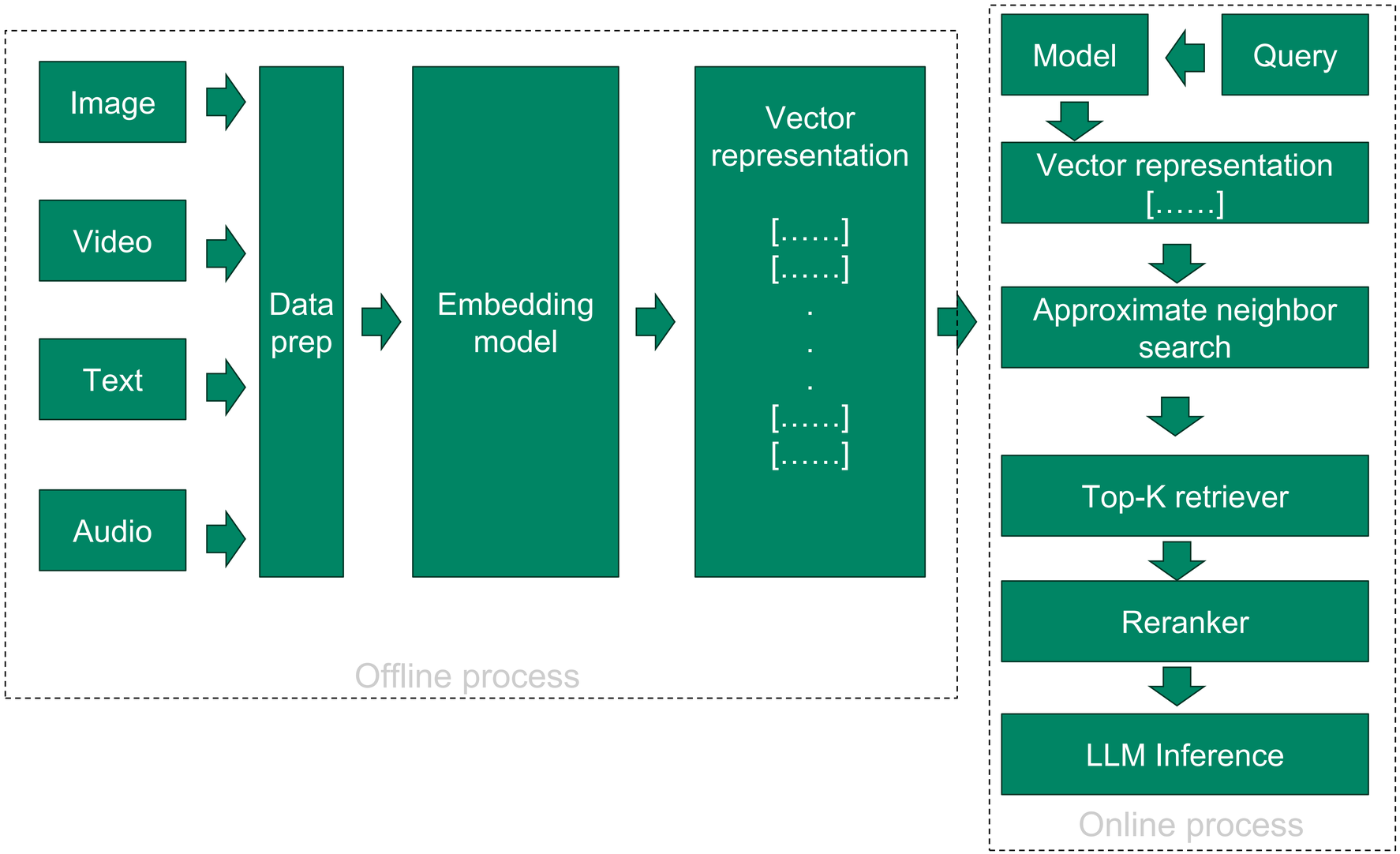


- Traditional focus for GPUs has been compute-intensive apps like Gen AI
- The explosion of innovation for VecDB, predictive AI drives new technologies to fill the gaps

Emerging AI applications

- VectorDB and RAG
- Predictive AI

Vector (semantic) search pipeline for RAG



Vector search / vector database

Where do GPUs benefit vector search & vector databases?

	Summary	Constraint	Applications	GPU Benefits
Online search	One-at-a-time queries submitted at moderately low volume	Individual search **latency** important (typical SLAs require <<100ms, usually closer to 10ms)	Recommender systems, semantic search (multimedia search using vector embeddings)	Consolidate CPU-centric hardware while maintaining 10x lower latency
Offline search	10s to thousands of queries submitted at a time	Query **throughput** is important (typical SLAs require 10s to 100s thousand vecs/s)	Data mining and exploratory data analysis algorithms such as data clustering and visualization	GPUs can process more queries in parallel than CPU, delivering 34x higher throughput
Index build	Training a machine learning model	Build **throughput** is important	Training models for predicting the approximate nearest neighbors fast and with high quality	GPUs can build larger and higher quality indexes 20x faster.

- Disruptive movement of compute from CPU to GPU
- GPU throughput matters for offline and index build but not online (RecSys, semantic search)
  - The total latency for traversing index graphs and fetching candidate neighbors matter.

NVIDIA cuVS

Library for GPU-Accelerated Vector Search & Clustering

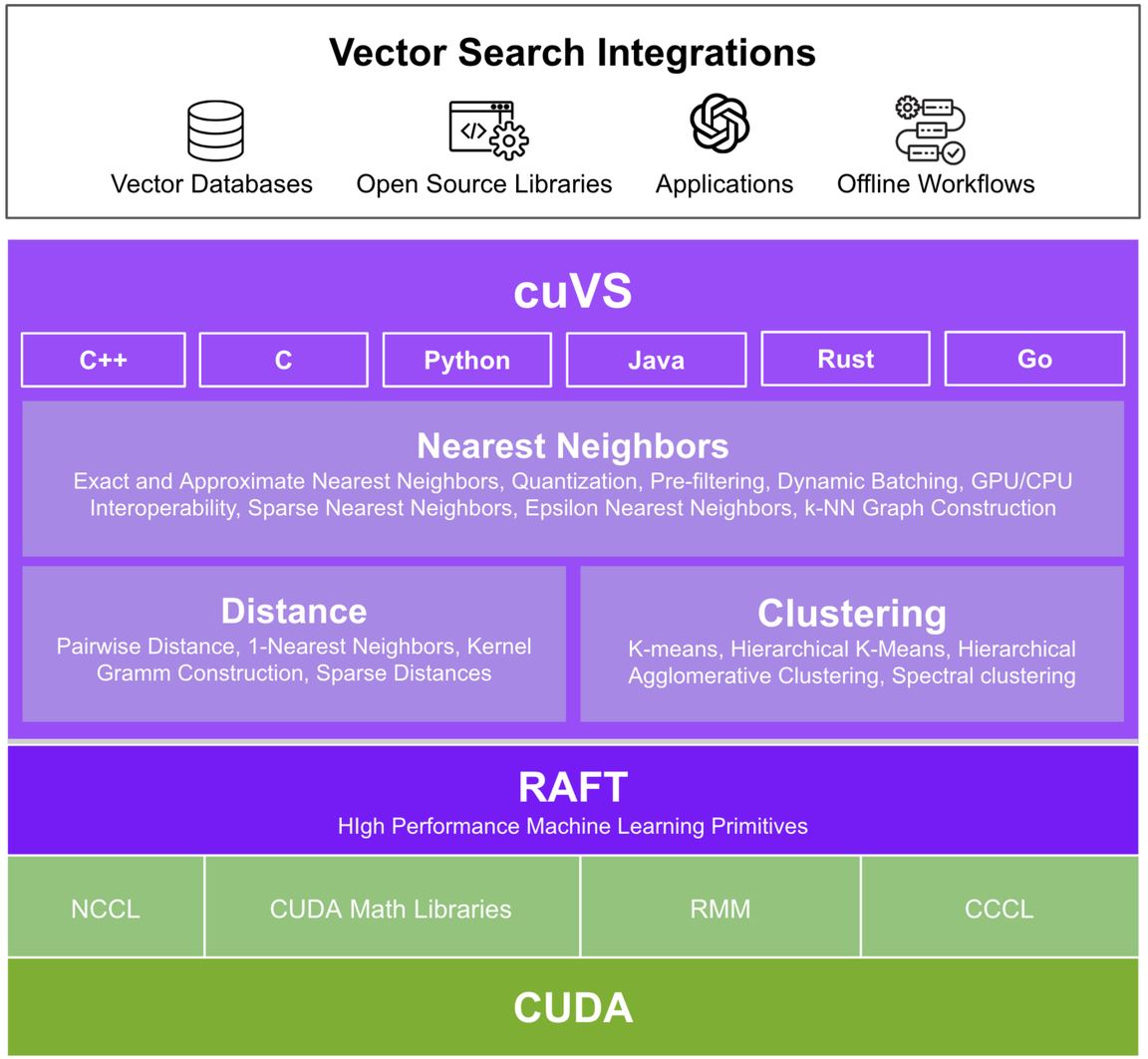
Open source

Apache 2.0 License

- Building blocks and end-to-end indexing APIs
- Multiple programming languages

https://github.com/rapidsai/cuvs

https://developer.nvidia.com/cuvs



Indexing of vector databases

- Vector databases may be huge (PB) and get updated 2% per unit time, where that interval is shrinking fast.
- Every time the VecDB changes, the index either needs to be recalculated or incrementally updated.
- Index build: Ingest all, write it once, read sequentially to build index, write index as sequentially as possible.
  - Log-structured, dynamic (Pgvector/Postgres), global index (VESPA)
- Index on disk update: read new data sequentially, **sparse R/W** of random locations in index
  - Actual writes are merges of coarser read data into a log structured merge tree as a flat index, with a coarse-grained write
  - Serialization of writes incurs a very non-linear address → storage location mapping that induces a subsequent random read access
  - Tiering of graph over flat indexes accumulate up to a threshold which triggers reindexing
- Index usage: random sparse fine-grained reads
- Premier resources used to build the vector database index may be different than cheaper resources used for lower-bandwidth vector search → modest persistence constraints

Vector database architectures

How Databases use approximate nearest neighbors (ANN) indexes

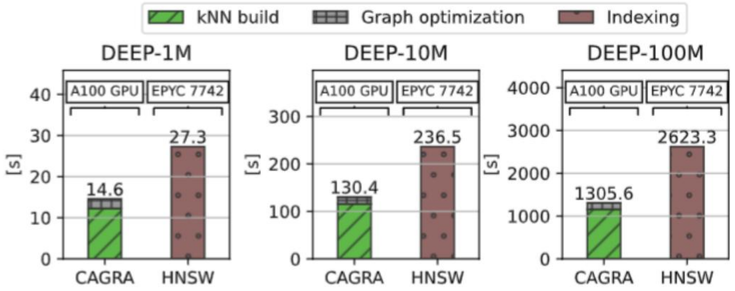
Architecture	Description	Typical Usage	Benefits	Drawbacks	Examples
Log-structured Merge (and derivatives)	Data ingested into segments, (shards) and indexes built from them. Larger number of vectors = larger number of segments and "indexes"	Distributed structured and lexical search	Partition tolerant for scale-out	Blind sharding of vectors requires all levels of the merge trees be visited during search	Milvus, Lucene, Solr, Elasticsearch
Dynamic (and tiered) Index	Builds single large ANN index that is dynamic (can support CRUD in-place).	Transactional and relational databases	Immediately consistent	Focus on scale-up; not partition tolerant	Oracle AI, Pgvector, AlloyDB
Global Index	Spatial locality is established up front by sorting vectors into various buckets. Multiple layers of buckets can be used to create a hierarchy	Distributed and scalable vector search systems	Highly partition tolerant and scalable	May need periodic reindexing to rebalance across buckets	Vespa, SPaNN, SCaNN, IVF, Cluster-based DiskANN

- Log-structured merge could make use of GPUDirect Storage – coarse grained from CPU
- Dynamic and global index could make use of SCADA. – fine-grained from GPU

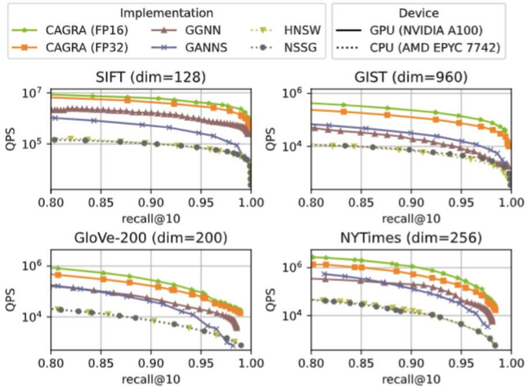
CAGRA

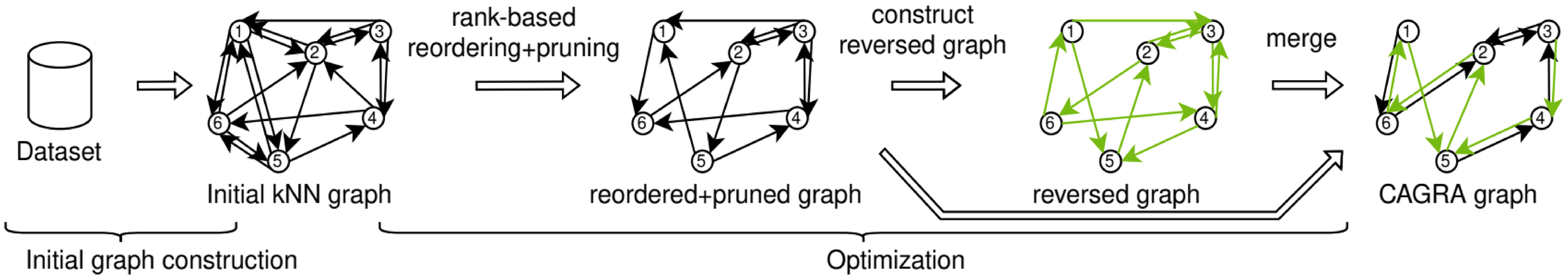
GPU-Accelerated State-of-the-Art Graph-Based ANN

- ***Individual queries*** parallelized during search
- Setting records for both ***single query*** and ***large batch*** performance
- ***Higher throughput*** than existing GPU Graph ANNs and ***lower latency*** than SOTA CPU Graph ANNs



Batches of 10k queries

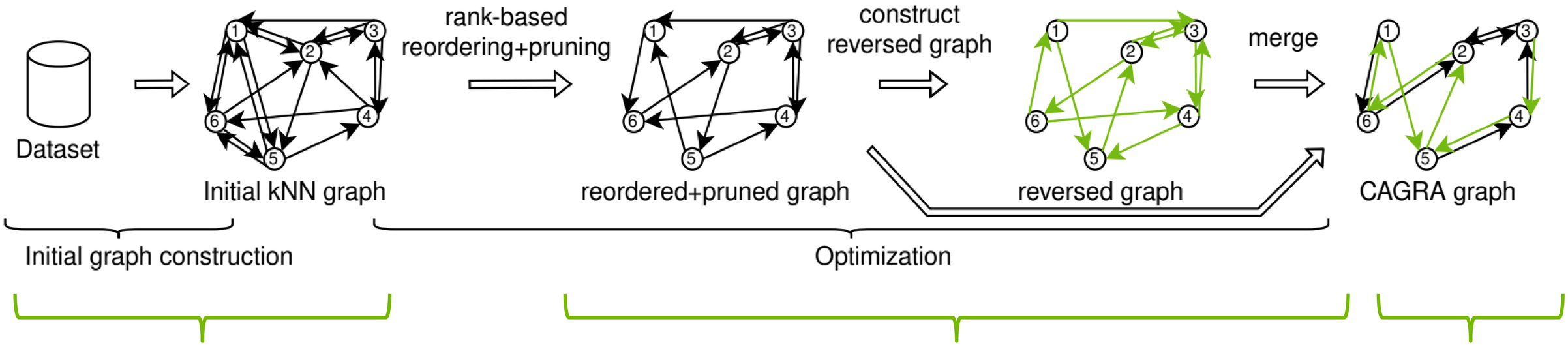




"CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search on the GPU", Ootomo et al., 2023

CAGRA Index Build

Graph based algorithms requiring fine-grain access



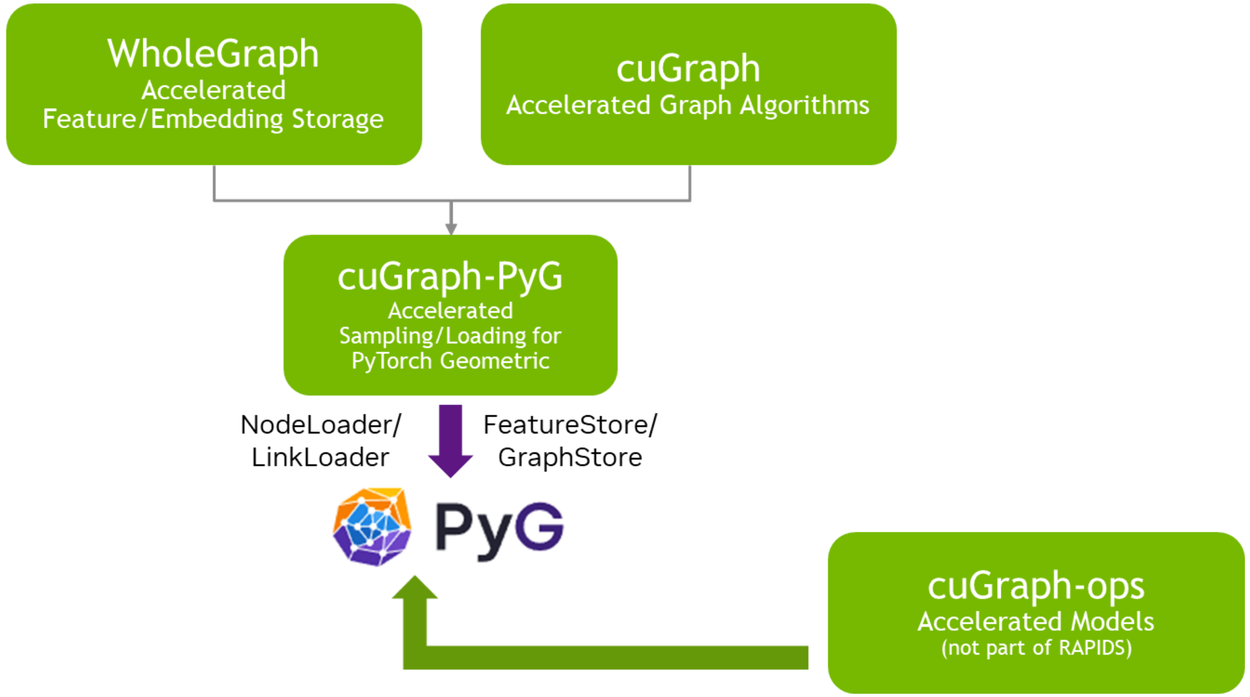
Highly parallel and memory intensive

Extremely memory access sensitive

Merge & Write only

SOTA GNN Software Stack

cuGraph/cuGraph-PyG



- Two main data structures in GNN
  - Graph - 100s Million to several Billion edges
  - Embeddings – Every node or edge has embedding of size 512B to 4KB
- Common algorithms
  - Subgraph creation using neighborhood sampling
    - Multi-hop random sampling for given set of nodes
  - Smaller (than LLM) model training on the gathered subgraph
    - Attention is partly provided by graph edges
- Access granularities
  - Graph is accessed in 8B to ~1024B
  - Embeddings in 512B to 4KB
    - Sizes don't compress much due to accuracy loss

Graph data

- Data: (x=[num_nodes, num_feats_n], edge_index=[2, num_edges], edge_attr=[num_edges, num_feats_e]
- Small Example: MLPerf IGBH Dataset: 547M nodes, 5.8B edges[1]
- Real world: nodes and edges can be added or existing ones can have their features modified
- Size
  - Initial graph structure - tends to fit across a set of compute nodes. Ex: 1T edges might be 8TB
  - Total embedding data - exceeds the HBM size for # GPUs needed for inference or GNN training.
    - Ex: (100B nodes + 1T edges) * 4KB = 4.4PB!
  - Subgraph Sampling:
    - (NeighborSampler(batch_size=1024, num_neighbors=[5, 10, 15])
      - **subgraphs w/ 1024*5*10*15 nodes = 750K nodes** ⟸ IO Parallelism per batch

[1]https://mlcommons.org/2024/06/gnn-for-mlperf-training-v4/

Emerging: combining GNNs and LLMs

GNNs enhance LLM accuracy with less space, compute, and complexity

- LLMs
  - Models are large
  - Single-hop: associative memory, e.g. P(token[n] | tokens[0, n-1])
    - Running out of public quality data
    - Public data is being poisoned by low quality LLM gen'd content
  - Encoding of relationships takes a lot of space (n^2)
  - Non-Deterministic: train("2,4,6,8..." & "2,6,18,54"), inference(2)-> ? (50/50)
  - Computational complexity is very high
    - Billions of params
- GNNs
  - Small SubGraphs are sampled from a large knowledge graph
  - Multi-hop: complex multi-hop relationships
  - Encoding of relationships is very efficient (explicit edges)
  - Deterministic
  - Computational complexity is relatively low
    - Millions of params

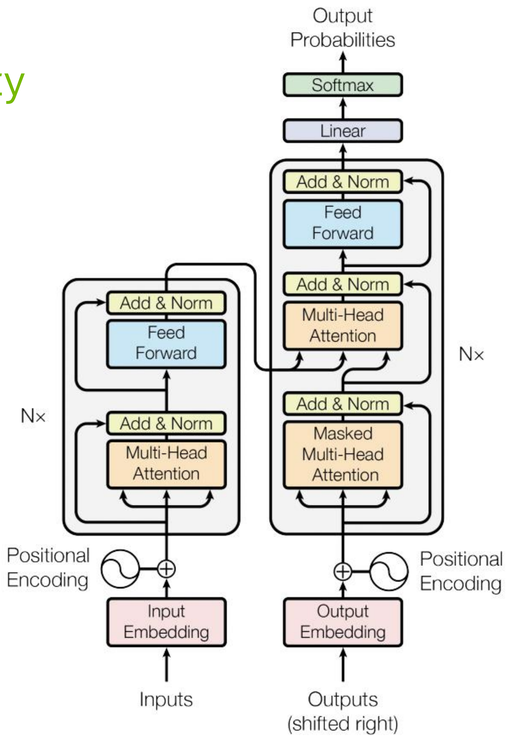
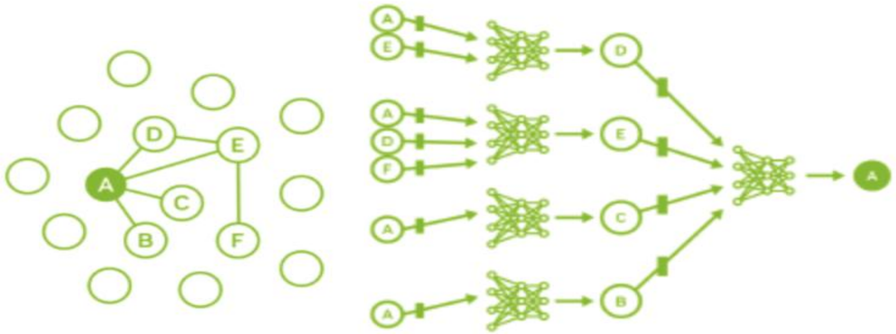
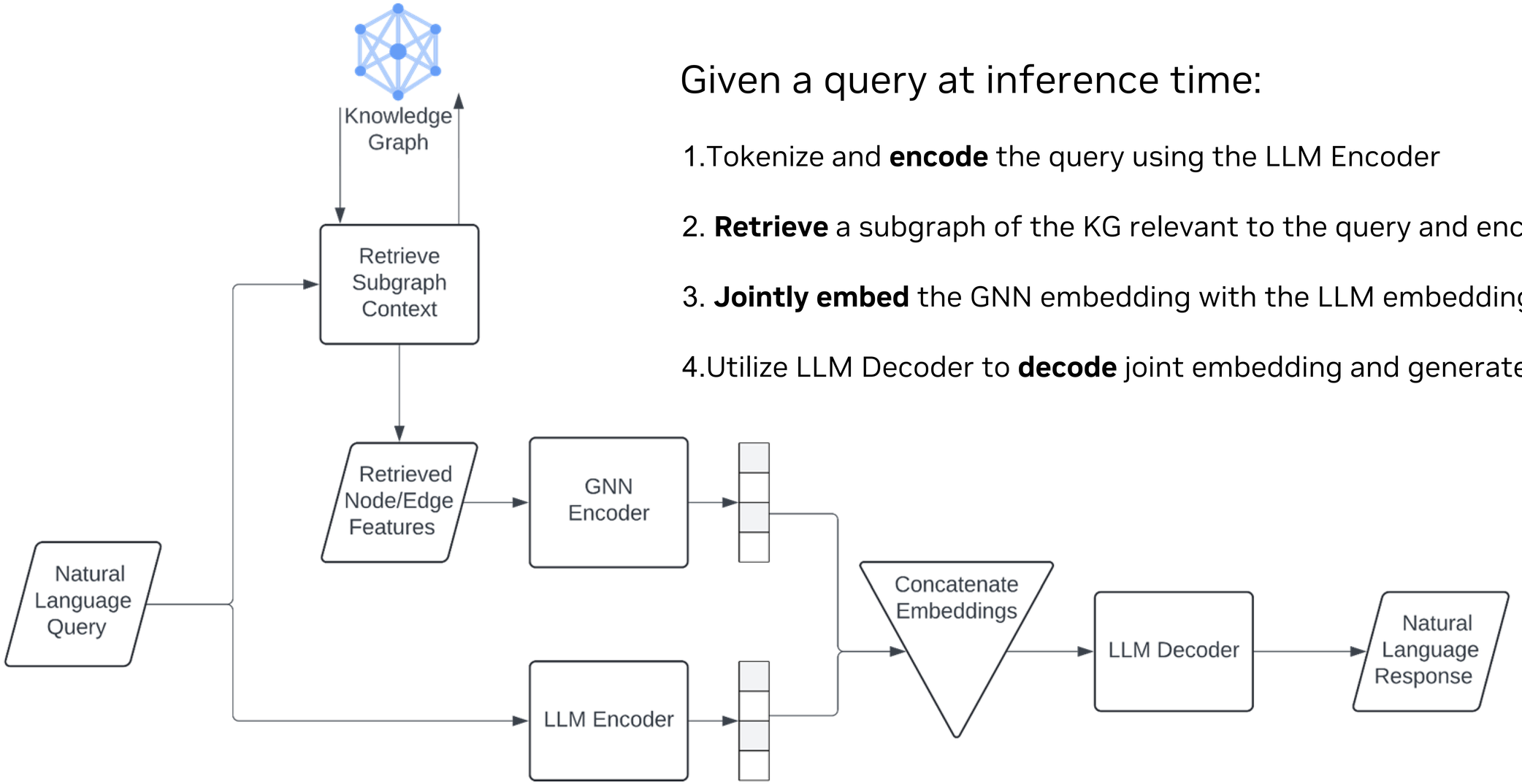


Figure 1: The Transformer - model architecture.



Example application of GNN RAG



Given a query at inference time:

1.Tokenize and **encode** the query using the LLM Encoder

2. **Retrieve** a subgraph of the KG relevant to the query and encode it using a GNN

3. **Jointly embed** the GNN embedding with the LLM embedding

4.Utilize LLM Decoder to **decode** joint embedding and generate a response

*RAG = Retrieval Augmented Generation

*G-retriever: https://arxiv.org/abs/2402.07630

GNN RAG

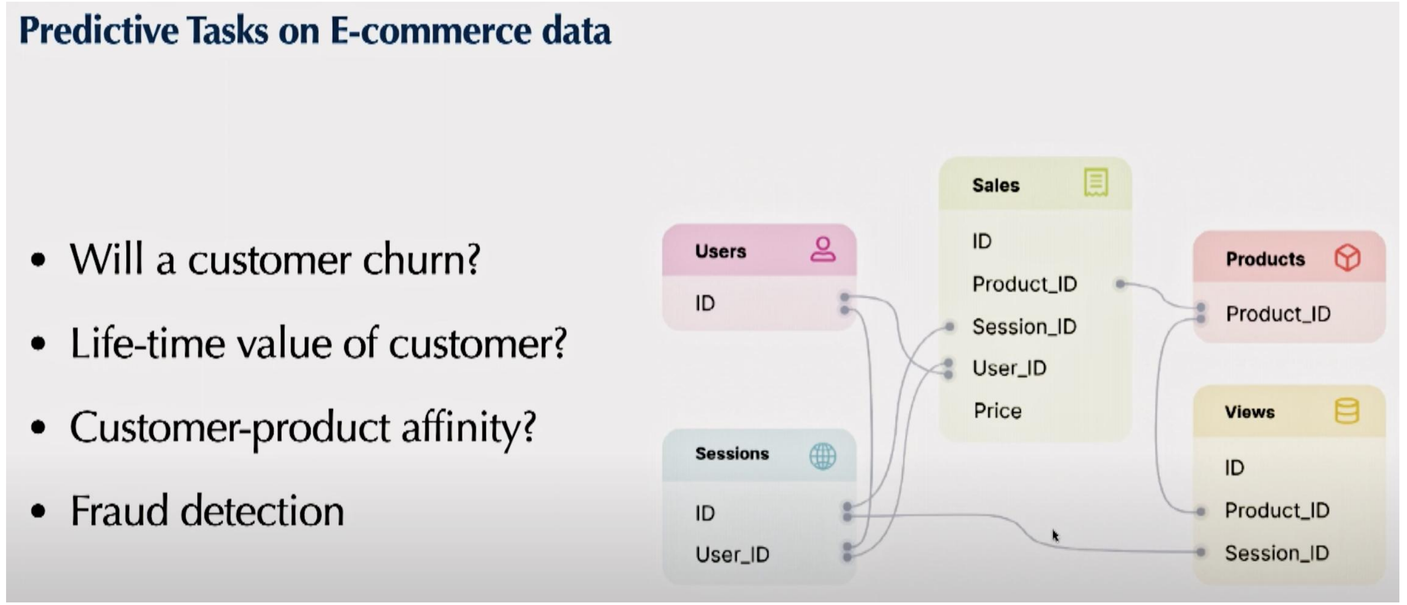
Early results are promising in this space

- GNNs can significantly improve LLM accuracy with small overhead
  - up to 2x accuracy seen so far,
- Requires knowledge graphs
  - Human curated, or LLM(docs) -> knowledge graph
  - Knowledge graph growing fast in size
  - New, promising area
    - Webinar: https://www.youtube.com/watch?v=uRIA8e7Y_vs

Graph Relational DB

Predictive AI using relationship information is gaining popularity

- Raw Data: SQL tables w/ keys linking them
- Node Data: Row in a table w/ time stamp
  - Each table: users, sessions, products, views, sales, ...
  - Each column is a feature
  - Each feature:
    - integers, floating points, or text/image embeddings
- Edge Data: Unique ID's linking nodes



Heterogeneous graph – massive in size for both structure and embeddings

https://www.youtube.com/watch?v=s_xGhXX_F3c

Graph Transformers

- Train Graph Transformer on many Graph Relational DBs → zero shot on new data
  - Both training and inference require accessing datasets
- Closed-source example, Jure Leskovec's company:
  - https://kumo.ai/company/news/kumo-relational-foundation-model/
  - Convert compute and datasets into real value, no effort
    - $200M increase in revenue for Reddit, 2x sales for DataBricks based on 5x accuracy
- Open-source w/ Jure Leskovec's post doc student: WIP (NVIDIA - Stanford collaboration)

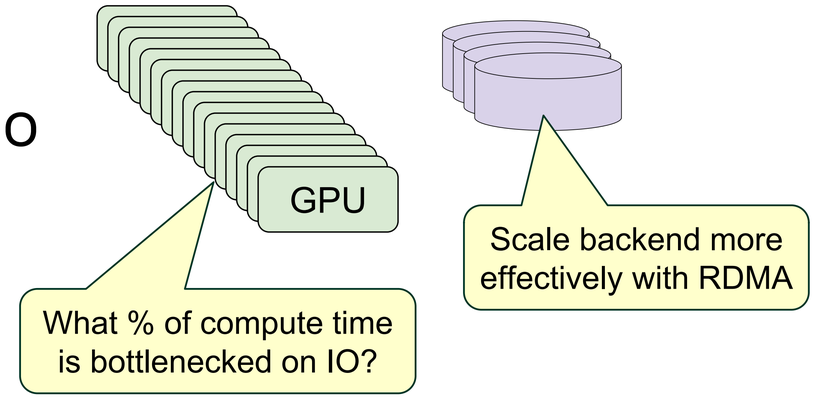
Storage parameters

- Location
  - Bandwidth, latency
- Connectivity
  - PCIe peer-peer
  - NS, EW; RDMA

- Criticality
- Capacity
- Throughput (IOPS)
- Location: latency
- Buffering
- Granularity
- Power
- Computing near storage

Summary of criticality at compute node

- Data centric vs. compute centric - IO::compute ratio
  - Low ratio LLM training
  - Usually low LLM inference, except for model load
  - High Search for predictive AI, vector DB indexing
- Factors impacting criticality
  - Size of data transfer
    - ↑ Example of large: KV cache with context data from 100x 100MB PDFs that are repeatedly queried
  - Total IO bandwidth dissipated across multiple GPUs
    - ↑ Ex: each GPU building an index for VecDB makes accesses from O(100K) threads
    - ↓ Ex: KV cache often loaded into multiple prefill GPUs
    - ↓ Ex: LLM training state of 8TB spread across 10K GPUs for checkpoint save/restore
  - Implications for TTFT wrt human perception
    - ↑ Ex: model load for MoE (2GB per B parameters)*(a few models) for new worker
    - ↓ Ex: 4GB KV$ (Llama 405B FP4 ISL of 32K) across 8 prefill GPUs with 100 Gbps = 40ms << 100-250ms

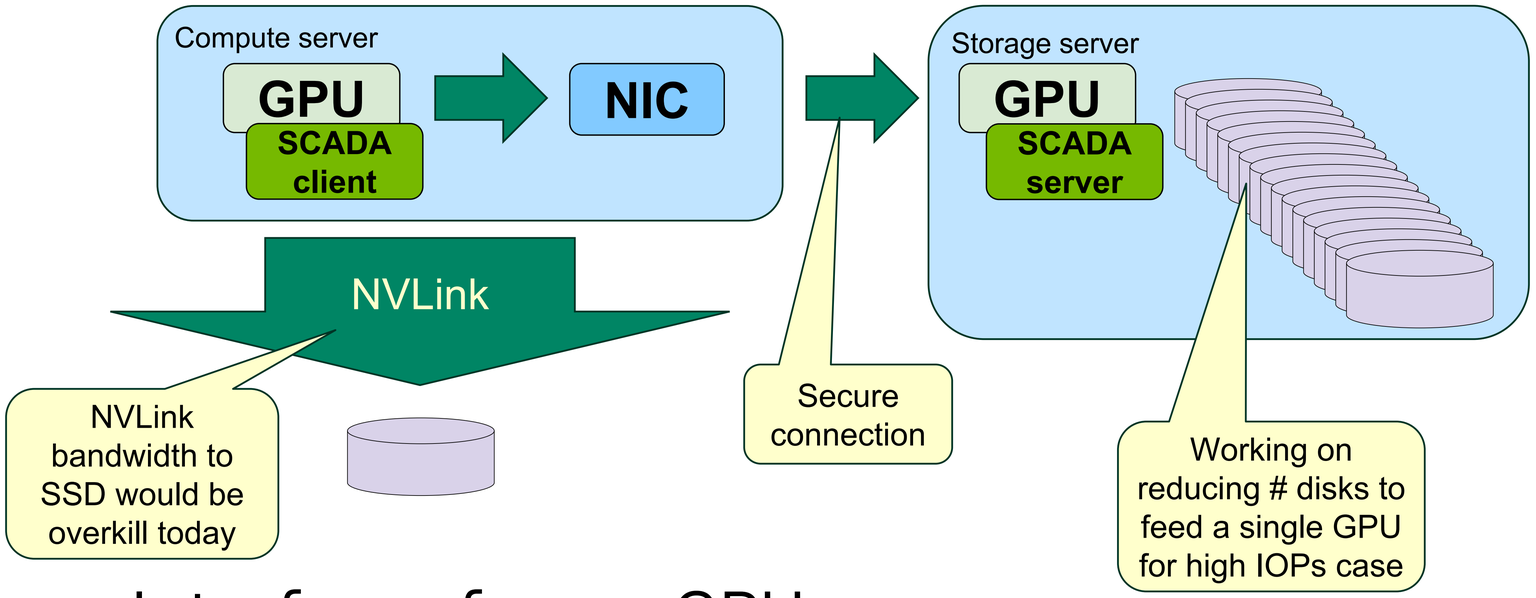


Summary of capacity

- Training data: usually remote
  - Can be 10TB – 1PB
  - Streamed in; only staged in nearby storage if used repeatedly in batches
- Key-value context cache (KV$): cluster local, not compute node
  - Individual context size based on
    - (past ISL)(KB/token for a model)(FP4 or FP8)
    - (16K-128K)(17 for DS R1v3, 123 for Llama 405B)(1 or 2)
  - Number of contexts based on (# concurrent users, e.g. 10K)(time between queries, e.g. 1s)
- Model parameters: compute node or cluster local
  - 2GB/1B parameters, for 405B is 810GB → can't fit in GPU
  - Increasingly common to have many models for mixture of experts (MoE)
- VectorDB and indexes: compute node or cluster local
  - 5GB-1PB per worker
- GNNs: compute node or cluster local
  - 100GB – 100TB per worker

GPU physical interfaces to storage

Main bottlenecks are number of SSDs and SSD IOPs, not connections to outside world

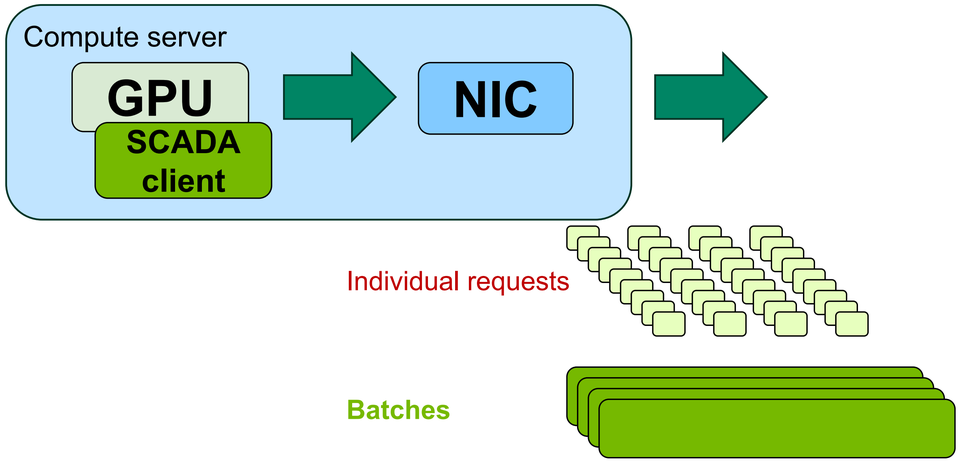


PCIe Gen	512B MIOPs for GPU	Drives/GPU legacy/optimized
5	100	32
6	200	40/10*
7	400	50/8*
8	800	60/4*

*Made-up numbers; not revealing proprietary info

- Interfaces from a GPU
  - PCIe or CXL (100 GB/s Gen6) Ability to saturate depends on GPU SW: app tuning and SCADA efficiency
  - NVLink 5 (1800GB/s) Won't be saturated with current apps
- Interfaces to SSDs
  - Hard enough to saturate wrt IOPs now, especially with finer-grained IOPs; NVLink would be overkill
  - Today, takes more SSDs to saturate fine-grained IOPs to GPU than can fit in a compute node
    - Gen6: 8 GPUs * 40 NVMes/GPU = 320 NVMes per compute node → move to storage server
    - Need 10x IOPs/NVMe to relieve bottleneck inside compute node

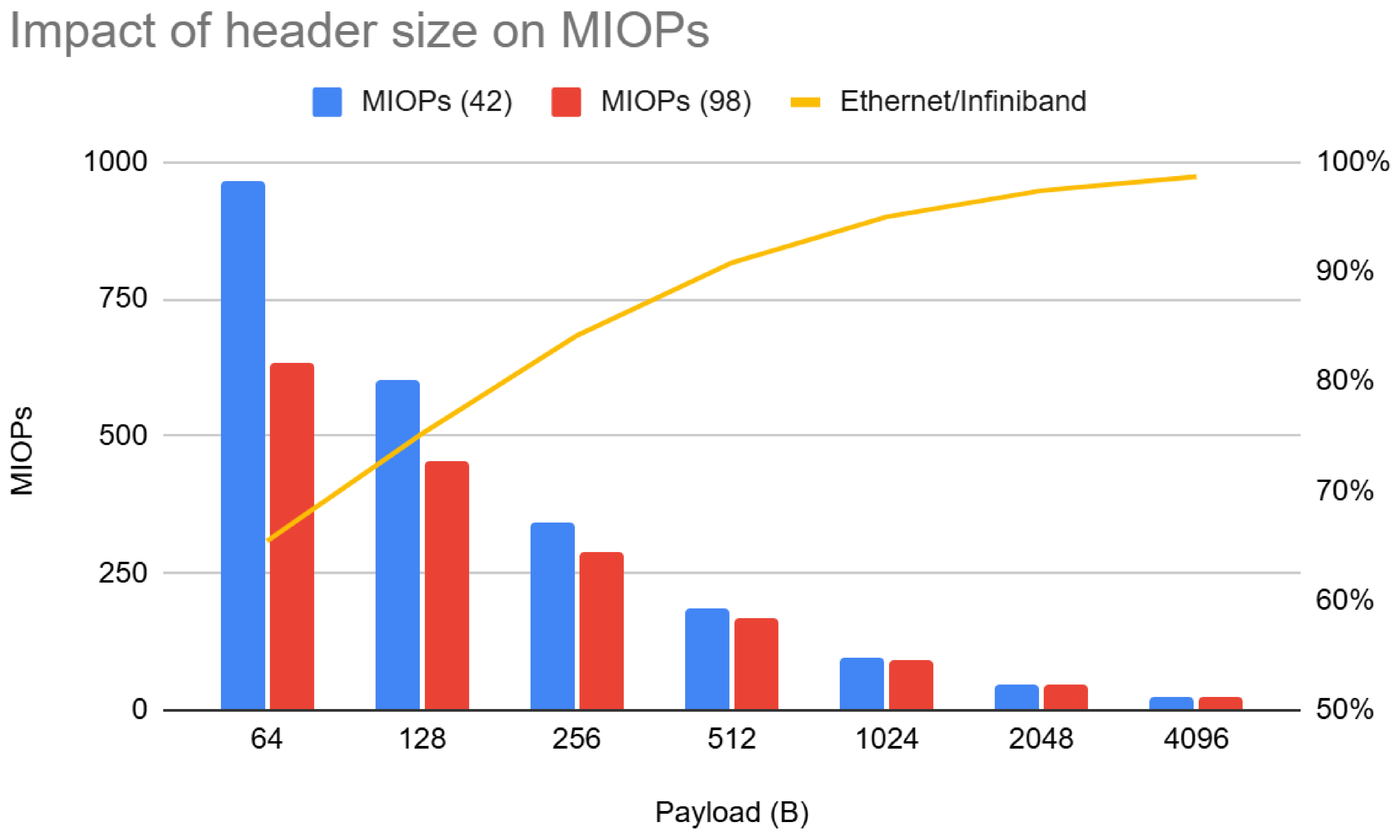
IOPs over the network: requests



- NICs are tuned for 4KB size
  - Theoretical max on Gen5 is 50 GB/s → 12.5 4KB MIOPs, 100 512B MIOPs
  - Measured with ib_write_bw, on CX7
    - 12 4KB MIOPs, 46 GB/s
    - 82 512B MIOPs, 40 GB/s
  - There's protocol overhead, models to 98B effective header size
- SCADA can gather and send bundles of 16K-64K requests in a single network "IOP"
  - Bigger IO size gets closer to peak
  - Unpack bundles of requests at server to boost effective bandwidth and IOPs
- GPU client → GPU server → GPU client (disk IO not a bottleneck)
  - Ib_write_bw 100%
  - DOCA GPUNetIO 95.3%
  - SCADANetIO 78.2%

IOPs over the network: network protocol comparison

Using IB vs. Ethernet can buy nearly 10%



- IOPs falls linearly with granularity

- EN packet header can be 98B
- IB packet header can be 42B
- At 64B, EN efficiency loss is 35%
- At 512B, EN efficiency loss is 9%

IOPs over the network: response aggregation

1.4x increase in IOPS and bandwidth for the combined coalescing of data and flag

- Responses and notifications are individual RDMAs
- Request could send 32 buffers of <32K each
- RDMA data response could be 512B
- RDMA notification could be 64B

- Can potentially coalesce either data responses or notifications
  - Applies to responses or notifications of consecutive 32 requests

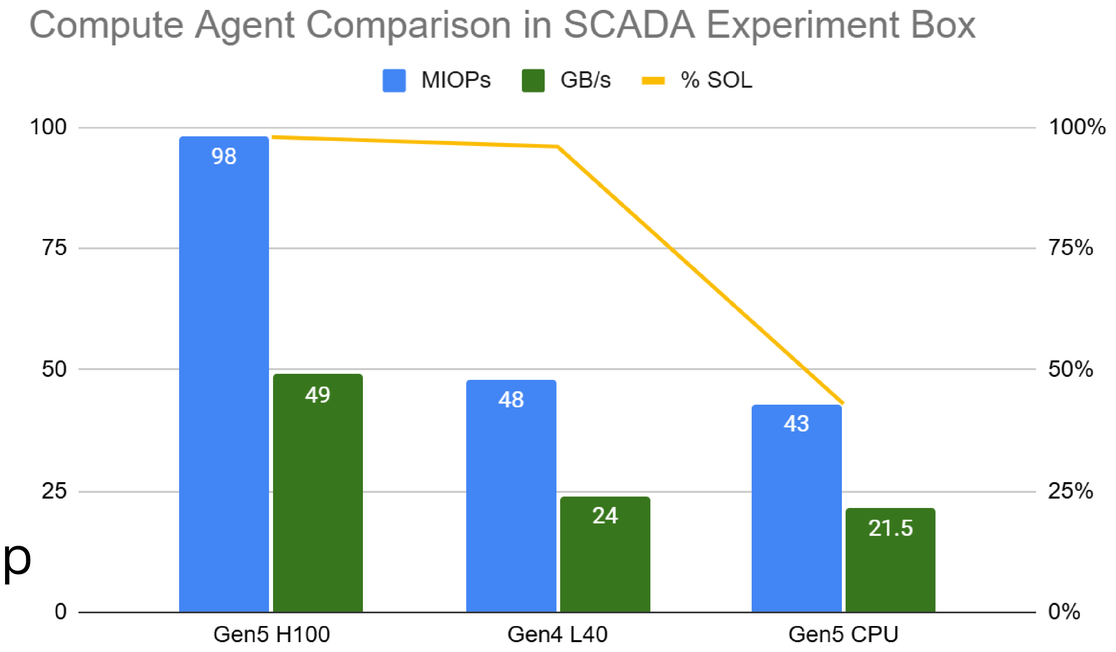
Performance results

- Saturating PCIe
- GPUs vs. CPUs
- E2E results with cuGraph.WholeGraph

GPUs near storage

GPUs accelerate not only app-level access, but also NVMe access, especially over the networks

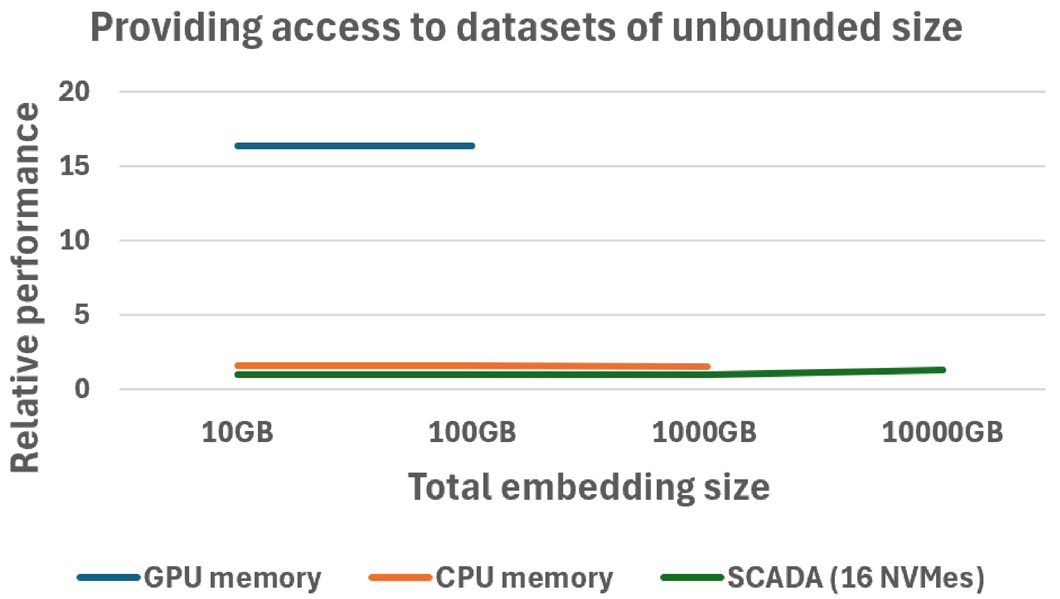
- 2x effectiveness at performing just NVMe IOPs
  - Can be even higher by adding more PCIe bandwidth to the RDMA datapath
- Need for PCIe switches unless IOQs and RDMA bounce buffers are in CPU memory
- AIDP recommends GPUs near storage to create insight platform near data
- Indexing is a data-intensive, GPU-accelerated app



End to end results with WholeGraph

Using one GPU

- Graph structure small enough to reside in GPU memory
- Node embedding data may grow beyond what could fit in many nodes' memory
  - CPU-GPU superchip or CXL would just be kicking the can down the road
- E2E results on 1 GPU
  - GPU memory with ld/st
  - CPU memory across PCIe to x86
  - SCADA to NVMes
- GPU vs. CPU memory from GPU is 10x faster
  - Limited GPU memory per node, fits 12GB IGBH-small
  - 100GB IGBH-medium won't fit in some GPUs
- **1.2x cost for unlimited size vs. mem-constrained**
  - 2.1 TB IGBH-full won't fit in CPU memory per node
  - IGBH-full runs a little faster, perhaps given more exposed parallelism
  - Early result with client/server on same GPU, IGBH 4KB, still in tuning



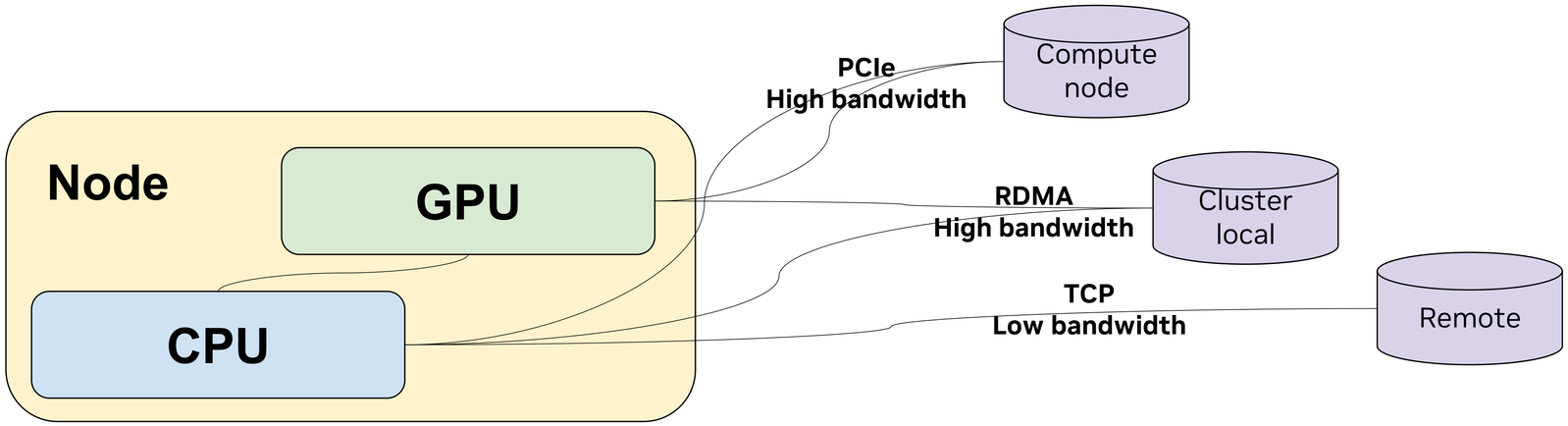
Performance take-aways

We've demonstrated a promising result, but need to map this to E2E performance on real apps

- IOPs to unlimited data size is approaching access to CPU over PCIe
  - Fundamentally changes what kinds of problems you can work on
  - Use HBM if you can fit, else CPU memory across as many nodes as necessary
  - But if you can't fit, use SCADA + Storage-Next drives
- Mileage will vary
  - 1.2x result is with E2E GNN training
  - Need to dig into app patterns and do evaluations
- Access pattern cases include
  - Max disk IOPs – each thread incurs multiple misses, fill to memory vs. registers
    - Not mdspan, used for this result and brought 1.3x-1.4x; more for less-constrained buffer size, e.g. 4K vs. 2K
  - High disk IOPs – each thread incurs a miss, but lack of coalescing may impact occupancy
  - Hit under miss – one miss per warp, other threads hit and coalesce
  - GPU cache hit - great temporal and spatial locality, very low disk IOPs
- Work ahead of us: map apps to cases like these and evaluate performance

Latency

Latency sensitivity varies; cluster-local is often adequate



Latency can matter

- When on critical path, e.g. TTFT
- Example: model load time for MoE

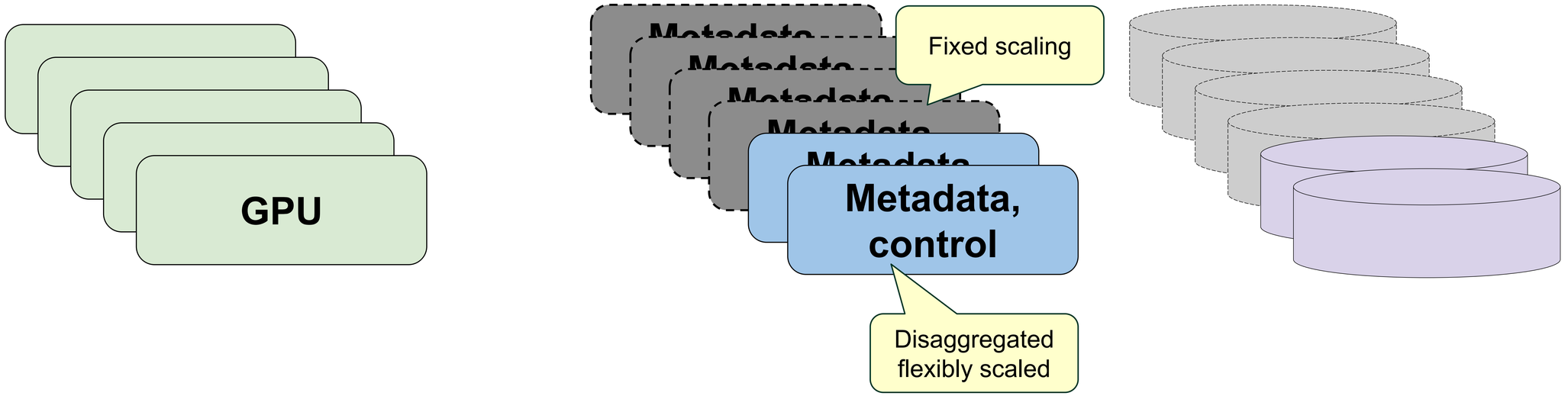
Latency is not necessarily king!

- May not matter if the access is predictable
- Once you are < SLA, it's all about tput efficiency
  - Data fetch implementations often just wait until the next time slot if not data ready
- Compare 10us to 8GB state/100Gbps/32 GPUs = 200ms
- Compare 10us to 100-250ms threshold of human perception

Use memory if really matters, else cluster local RDMA

Destination from GPU	Latency access+drive
Compute node-local	O(1us)+O(10us)
Cluster-local, TOR or nearby rack	O(10us) +O(10us)
Remote	O(1ms) +O(100us)

Disaggregation provides flexible scalability



- In compute server
  - Pressure to squeeze out local storage to enable higher density (Kyper)
  - Trend toward managed storage for ephemeral data
  - For KV$, can’t fit enough capacity in compute server
  - For some Predictive AI apps, can’t get high enough IOPs either
- In storage server, cluster-local or remote
  - Decouple compute/metadata (blue with solid outlines) from storage capacity
  - Adds flexibility, e.g. scaling for use of S3oRDMA
  - Contrast with fixed scaling (grey with dashed outlines), and implications of excess capacity, more cores with SW licenses than needed

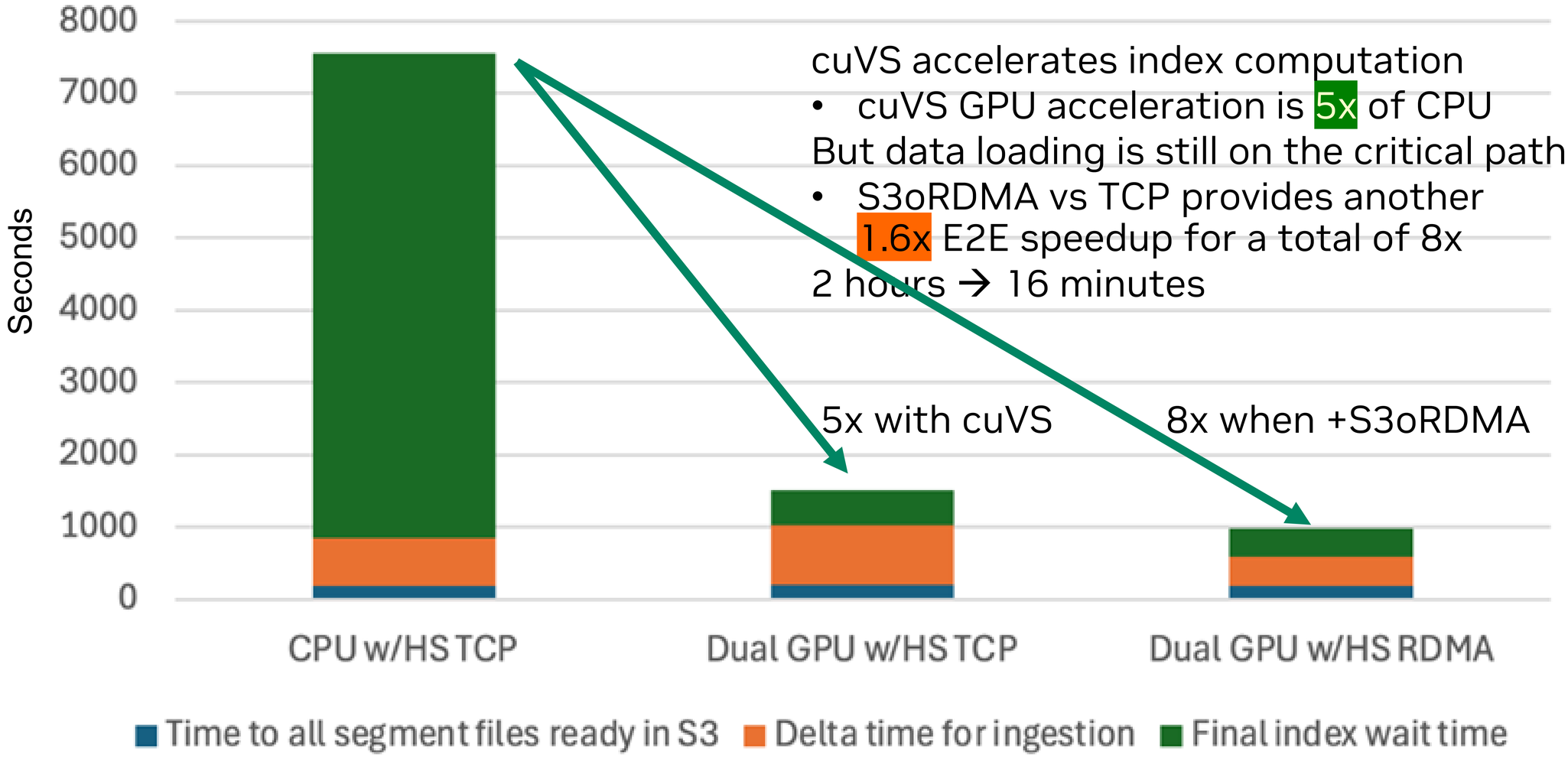
RDMA everywhere!

- Technology
  - TCP uses CPU software to implement a two-sided protocol
  - RDMA uses NIC hardware to implement a one-sided protocol
- Bandwidth
  - TCP bandwidth is limited by CPU
  - RDMA bandwidth is limited by network and storage
- Autonomy
  - TCP requires data path to path through CPU
  - RDMA enables data path to go straight to the GPU
- CPU load
  - Has some impact in the compute node
  - Has a significant impact in the storage server
    - Example: 1.8x fewer metadata nodes for a disaggregated architecture
    - Example: 6x fewer fixed building blocks for Cloudian

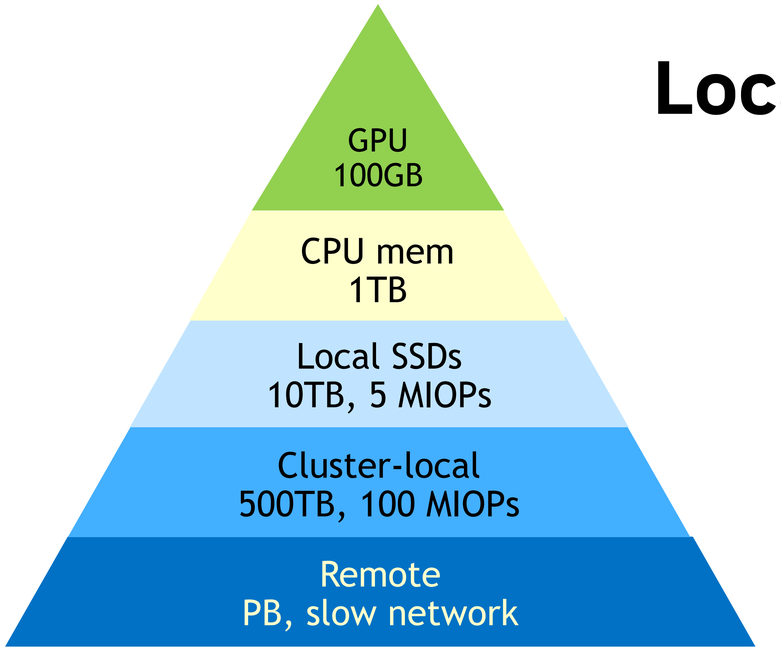
Category	TCP	RDMA
Active agent	SW on CPU	HW on NIC
Throughput	X	3X
Latency	X	3X
CPU load	20-50X	X
Critical resource	CPUs	Network/storage speeds and feeds

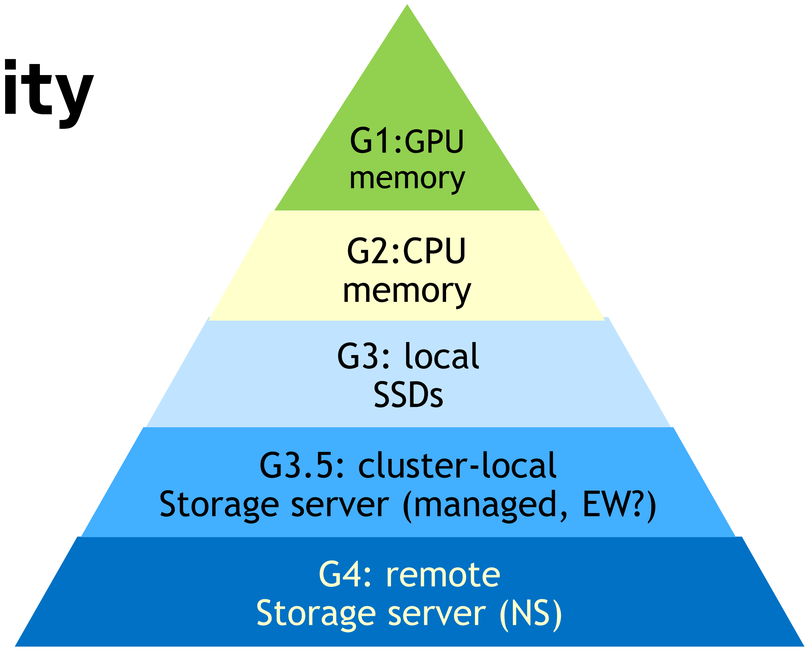
Example: Data loading for vector database indexing

Time to index/ store embedded data in a Milvus vector database with cuVS+Cloudian **S3oRDMA** vs. regular Milvus+S3



Location and connectivity

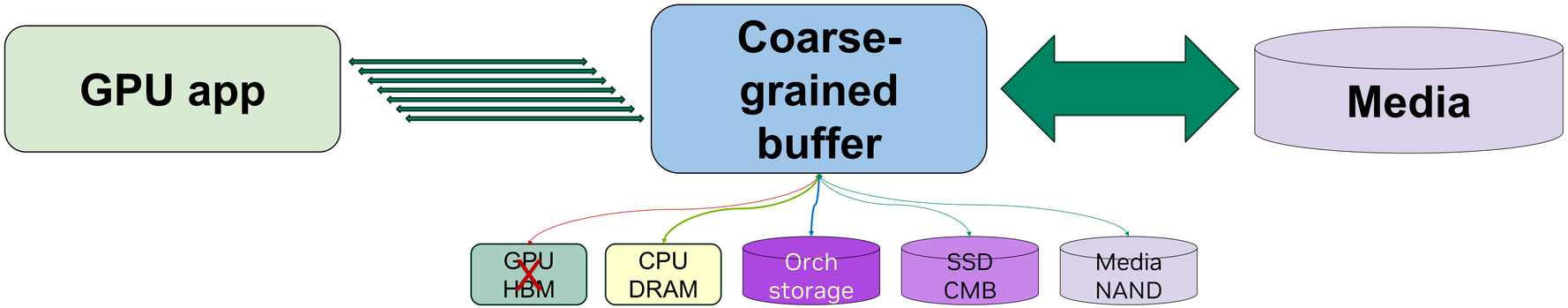




- G2 has phenomenal bandwidth, e.g. for access within a particular KV context
- Once problem doesn't fit in GPU or CPU memory, storage becomes extension of memory
  - Use of superchips or CXL may extend memory, but it doesn't fundamentally enable large problem size like storage does
- G3/3.5 is close enough wrt latency; tends to offer RDMA
- Have to go to G3.5 or G4 to get enough capacity for KV$
- KV$ gets enough bandwidth from NS connection to many prefill nodes
- Data-intensive apps can't be adequately fed by a limited NS bandwidth & IOPs; need EW
  - Putting specialized storage as memory on EW network is consistent with being used for compute data

Fine-grained access to coarse-grained buffers

Broaden the focus of Storage-Next beyond fine-grained IOPs to the media itself

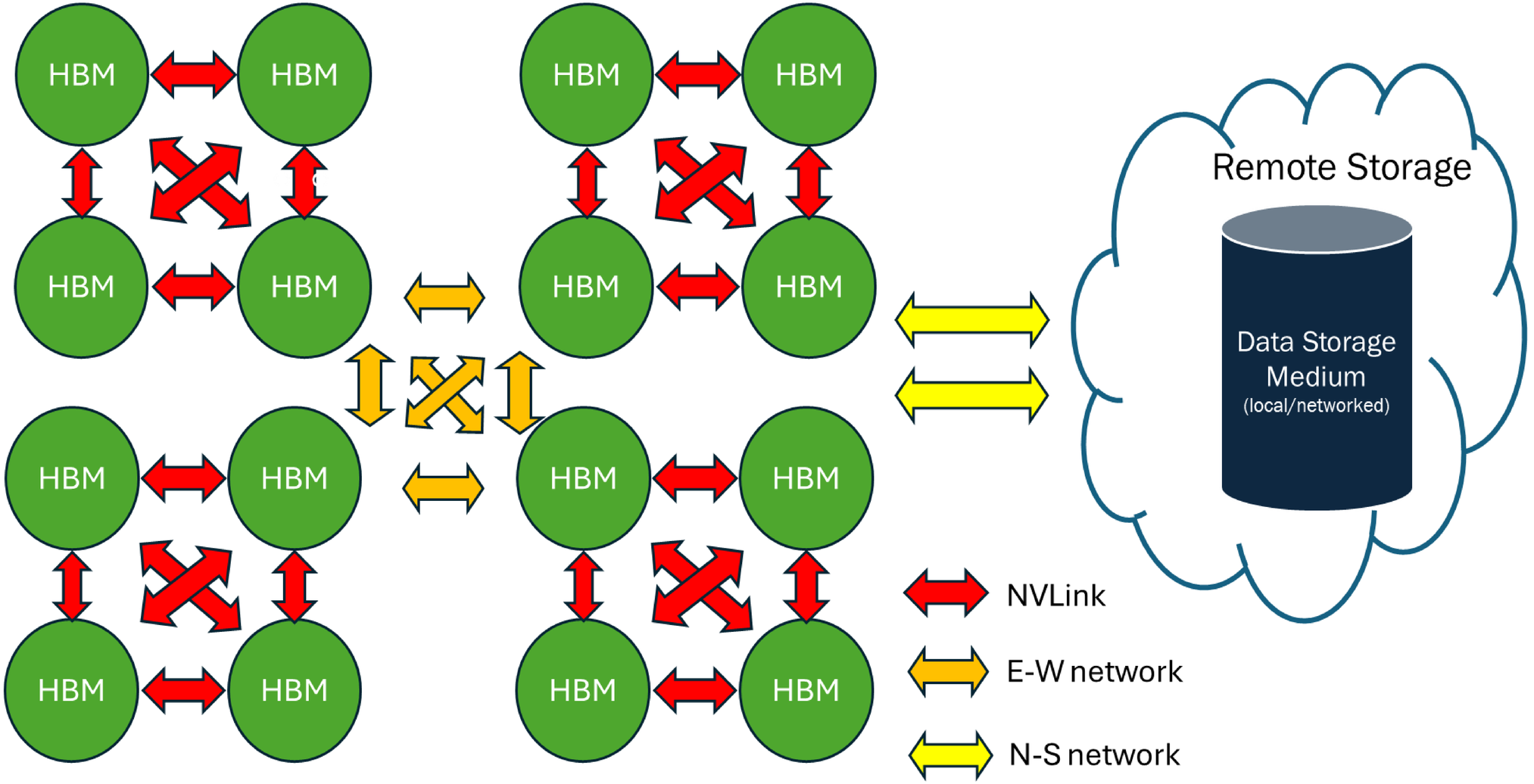


- Bulk access from media
  - Option for good locality: 16KB-32KB NAND page size for greater efficiency
  - Option for poor locality: ≤ 4KB codeword to save power from moving useless bits
- Fine-grained access once in memory/as-if memory
  - Stage in CPU if accesses are predictable and concurrency is low, e.g. for KV$
  - Direct access from GPU if accesses are dynamic and have high concurrency, e.g. vector databases/indexes and GNN embedding data

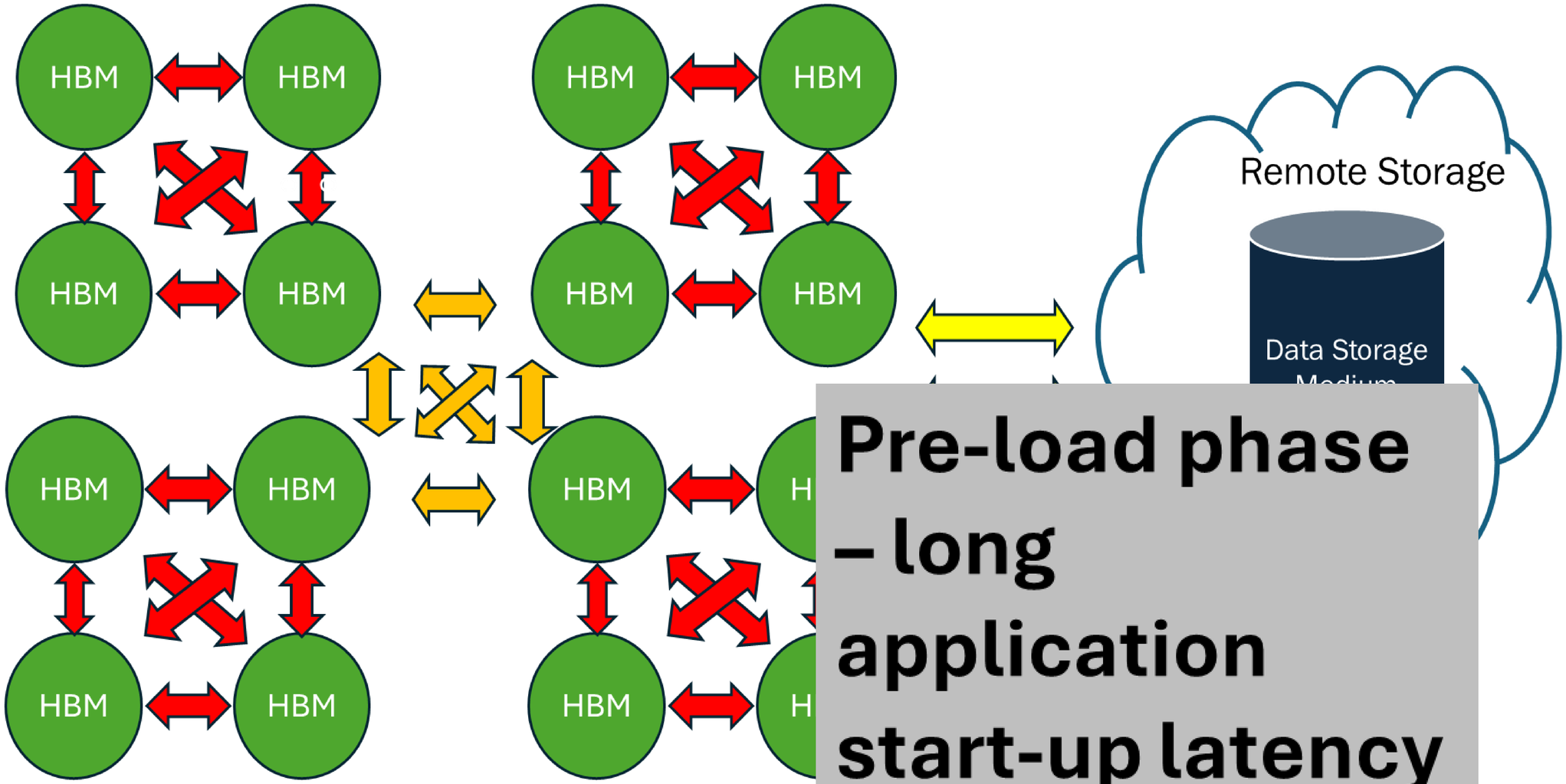
SCADA

- Problem and solutions
- Architecture
- Ecosystem

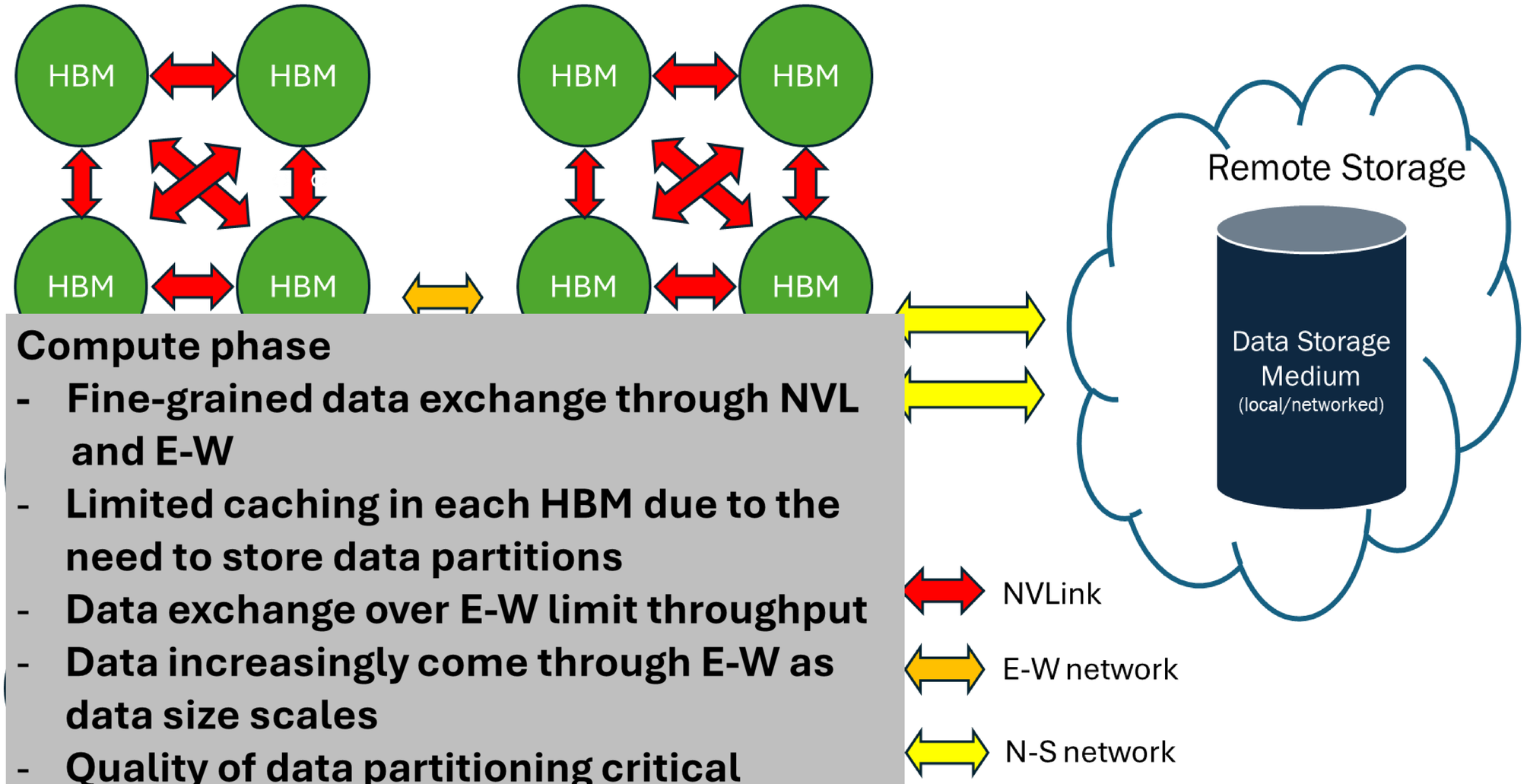
Old Way – DRAM Pooling and Fine-Grain Access from DRAM



Old Way – DRAM Pooling and Fine-Grain Access from DRAM



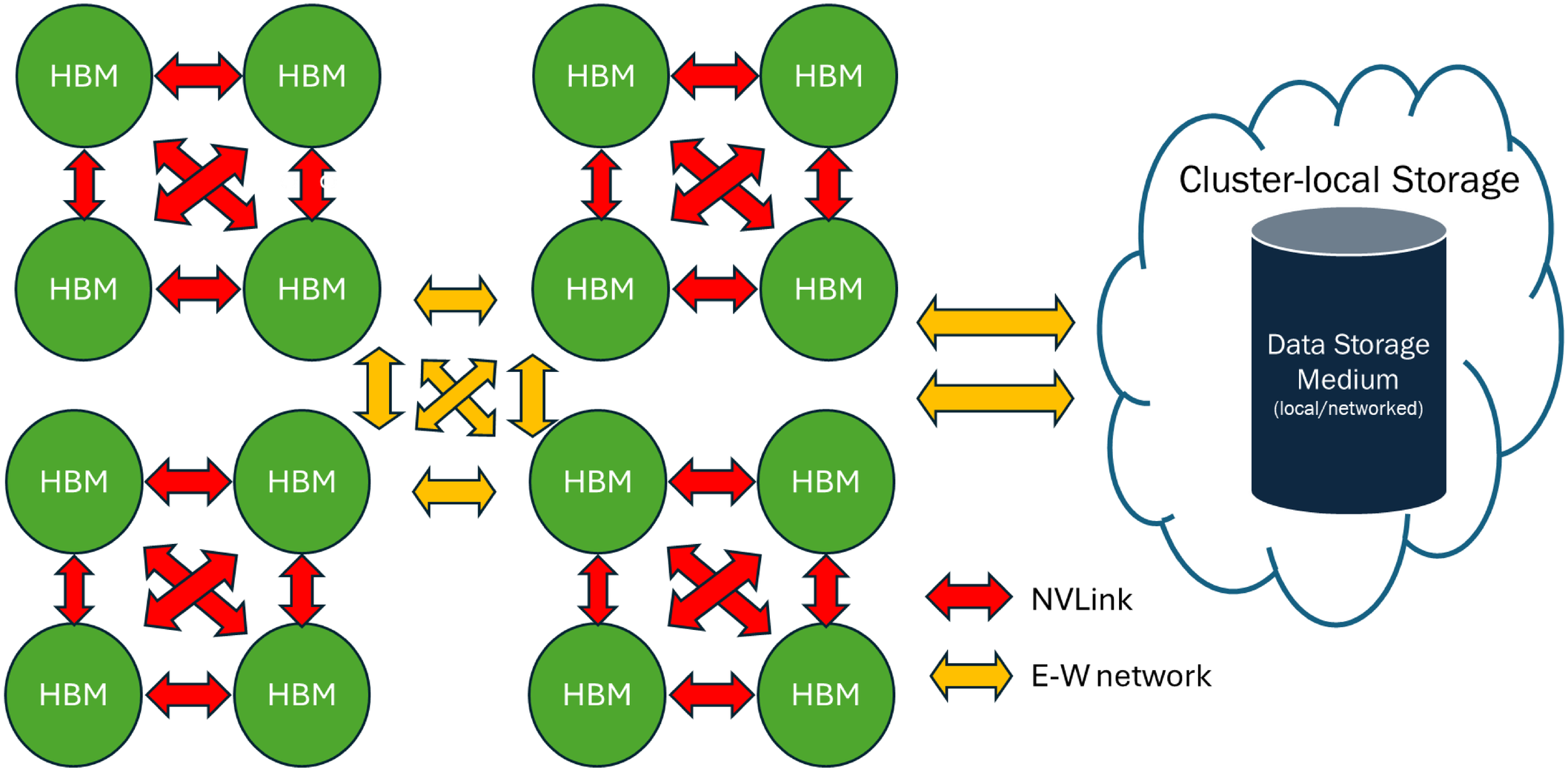
Old Way – DRAM Pooling and Fine-Grain Access from DRAM



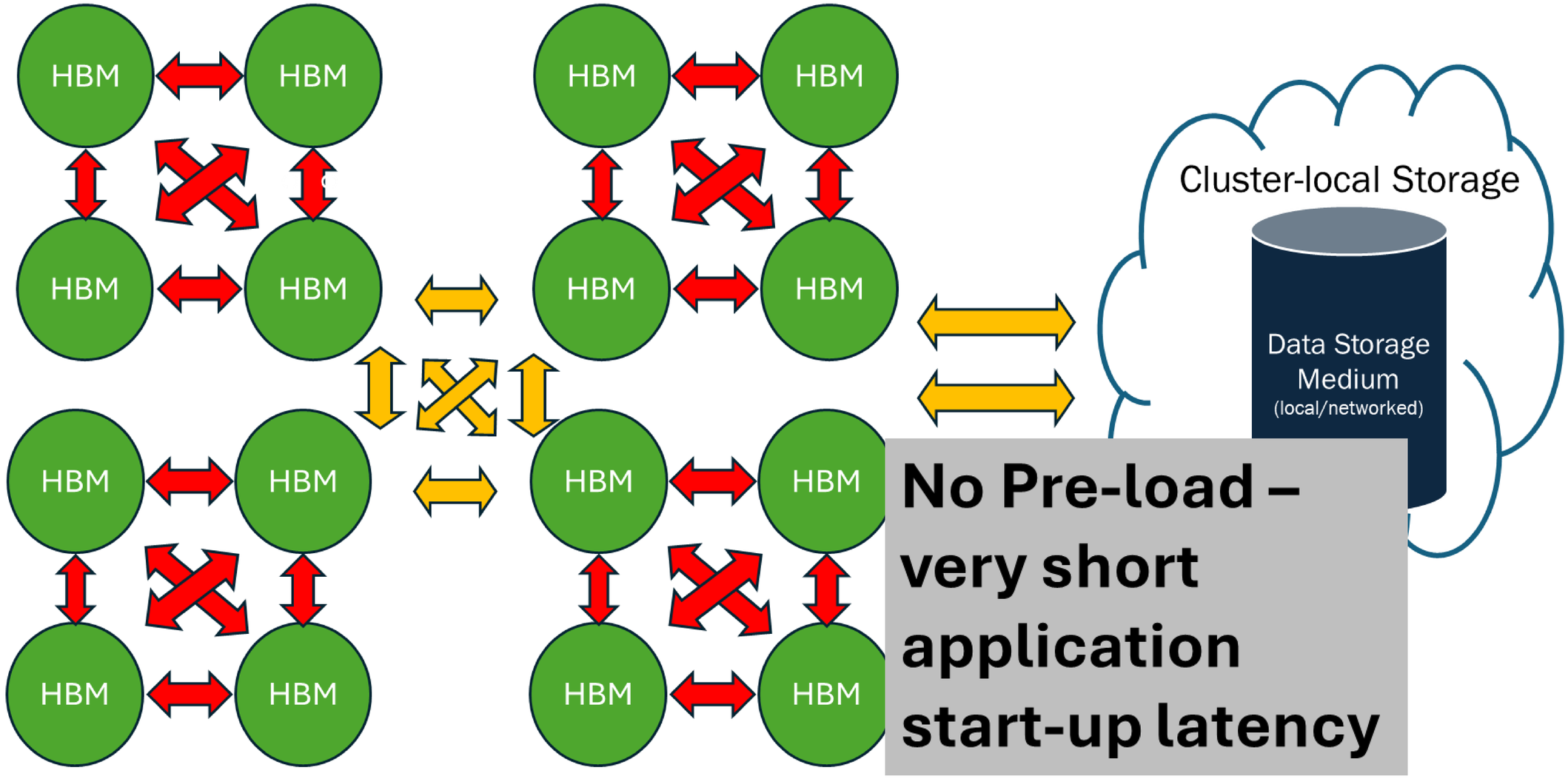
Compute phase

- **Fine-grained data exchange through NVL and E-W**
- **Limited caching in each HBM due to the need to store data partitions**
- **Data exchange over E-W limit throughput**
- **Data increasingly come through E-W as data size scales**
- **Quality of data partitioning critical**

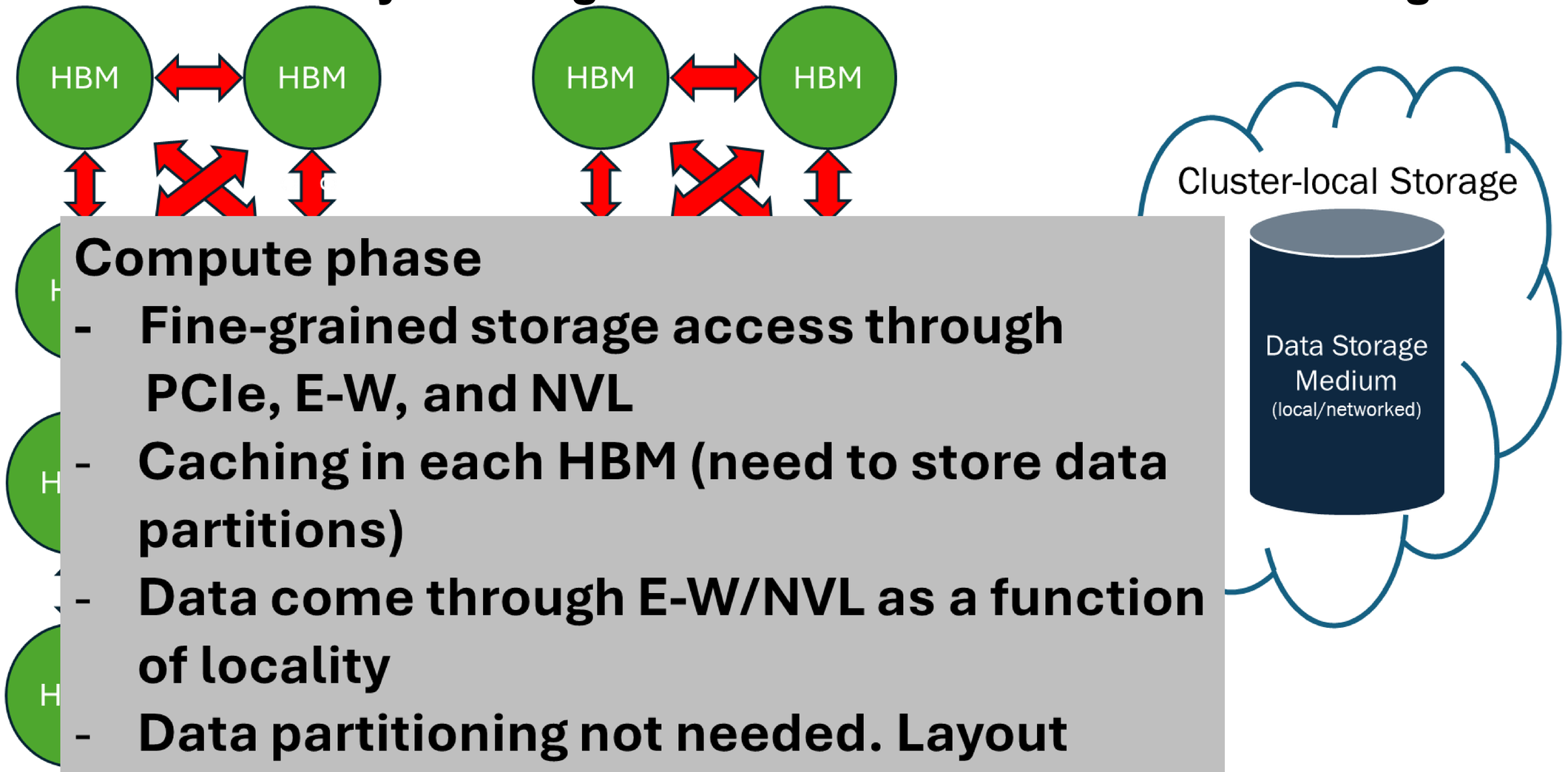
New Way – Fine-grain access from cluster local Storage



New Way – Fine-grain access from cluster local Storage



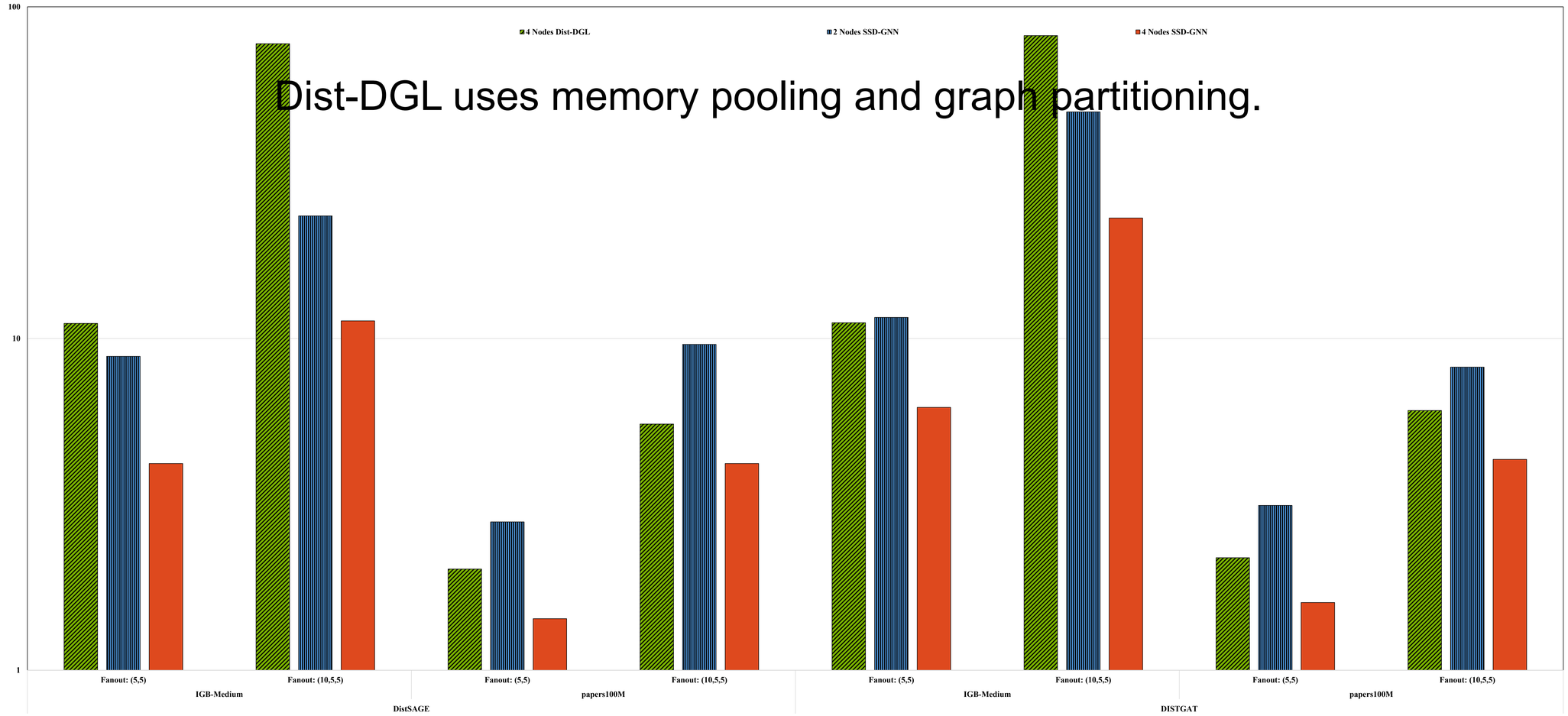
New Way – Fine-grain access from cluster local Storage



Compute phase

- Fine-grained storage access through PCIe, E-W, and NVL
- Caching in each HBM (need to store data partitions)
- Data come through E-W/NVL as a function of locality
- Data partitioning not needed. Layout optimizations can improve speed.

Example: BaM multi-GPU software cache for GNN Training



SSD-GNN is faster with fewer GPUs for large graphs

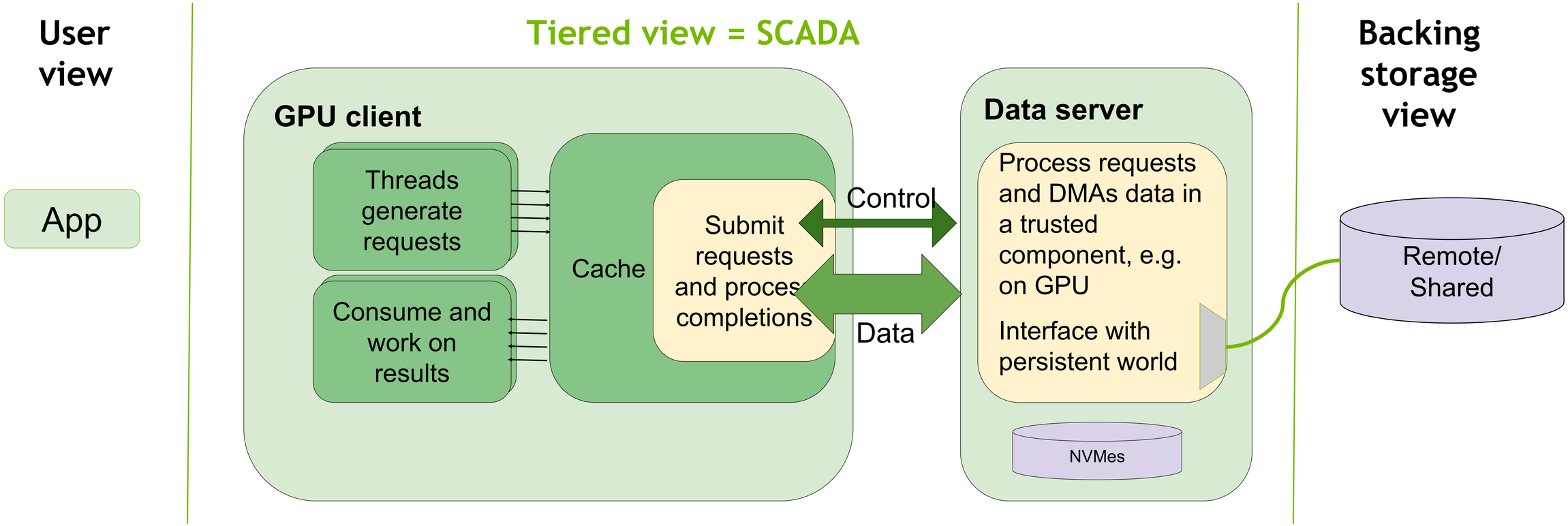
Problems → solutions

Ushering in a new era of data-centric computing on large problems with GPUs

SC	Scaled	• O(100) thread CPUs limit concurrency → O(100K) GPU threads • Partitioning → automatic GPU-GPU communication • Memory fabric breaks at scale → introduce an API, support errors
A	Accelerated	• GPUs can't run a file system, can't ask CPU for help → GPU initiation • Can read into registers from IO → buffer in a cache on GPU • CPUs can't tolerate latency → GPUs do • Too many fine-grained accesses → aggregation
D	Data	• Limited capacity in memory → unbounded storage
A	Access	• RAF problem from sparse access to big chunks read in → direct access

SCADA abstraction

GPU becomes an autonomous highly parallel data access engine



- User view: app-centric API, e.g. contiguous linear/mdspan, key-value (coming)
- Tiered view: implement caching, block access, many disks, map to outside world
- Backing storage view: interop with file/object systems

Ecosystem

- Data persistence
  - Input data may span multiple processes, can’t stream a PB through GPU when you’ll only access 2% of it
  - Output data, e.g. VecDB index, may be produced by one (GPU) process and consumed by another (CPU) process
- Interoperability
  - Ingest data – maybe object
  - Perform ETL – extract data of interest
  - Layout data for efficient linear block access
  - Co-ownership: SCADA while processing, storage system to move data around
- Data integrity
  - Deal with bad or slow blocks
  - Provide data checking and correction between SSD and GPU
  - Must keep up with GPU concurrency → compiled into GPU SCADA server
- Key-value store
  - Extension to client-side API, client-side cache, server-side mapping, integrated into GPU SCADA client and server

Initiatives

- Storage-Next
- TCO
- SKUs
- Roadmap planning

Storage-Next

Accelerating a rich mix of technologies to pull in usable deployment across the industry

- Goals
  - HW: define and refine IOPs/TCO-optimized SSD
    - This includes both fine-grained IOPs in the media itself and fine-grained IOPs to intermediate buffers with coarser-grained media access
  - SW: evaluate, optimize, generalize SCADA
  - Joint exploration and input to mature technology enough to standardize
- Participants
  - Direct technical stakeholders bringing solutions, not a spectator sport
  - Does not imply endorsement of any given requirement or solution
  - Media and controller vendors, for SSDs
  - Storage providers, for interop
  - Other: key value and data integrity that are GPU based
- Phases
  - Nothing mature enough to propose standard: NDA partners of NVIDIA
  - When we have something solid: broad participation as we move toward standard

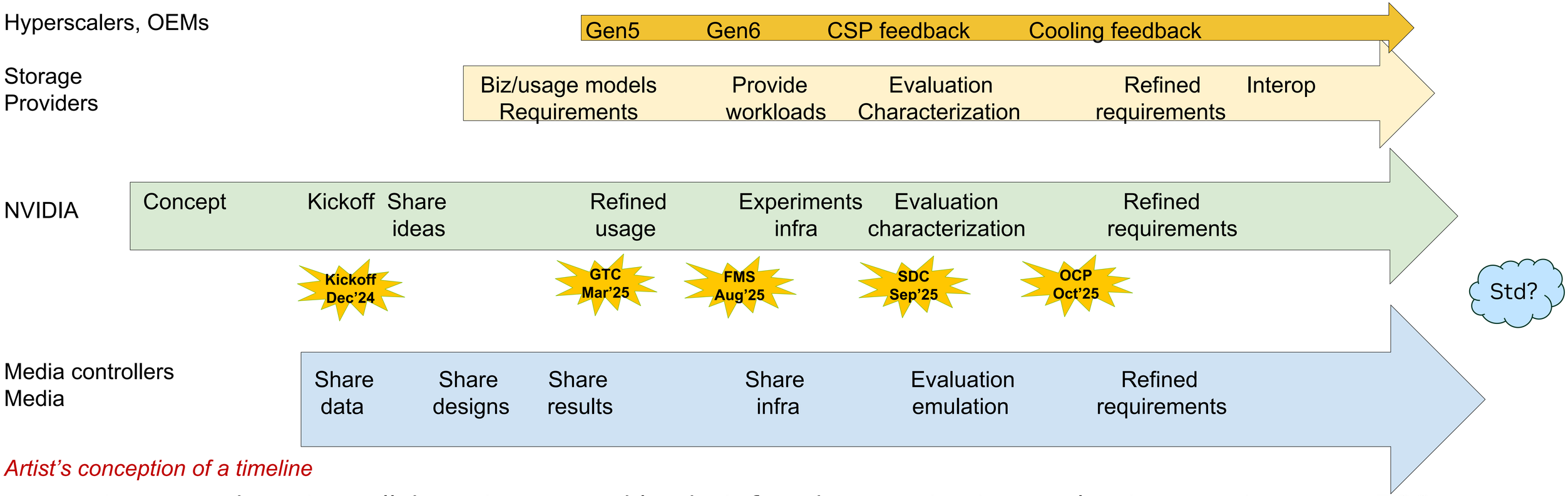
Total cost of ownership (TCO)

Components of the objective function that partners can exercise full freedom in optimizing

- 2 ongoing SKUs
  - TB/TCO Compute-centric, course-grained accesses
  - IOPs/TCO Data-centric, fine-grained accesses
- TCO as {BOM, volume, power}
- Implications for high volume of fine-grained requests
  - IOPs Gen5/6/7 PCIe x16 has 50/100/200 GB/s → 100/200/400 MIOPs at 512B
  - Drive/GPU Gen 5/6 @ 3.3/5 MIOPs → 32/40 x4 drives
  - Power 40 x 25W = 1kW
  - Problem size One drive may be enough to spill from memory to disk for a large problem size
  - Location Compute nodes left unchanged; full freedom to optimize IOPs/TCO in remote storage server
- TCO analysis is left to partners

The path to Storage-Next

IOPs and TB with improved TCO from BOM, volume, power improvements

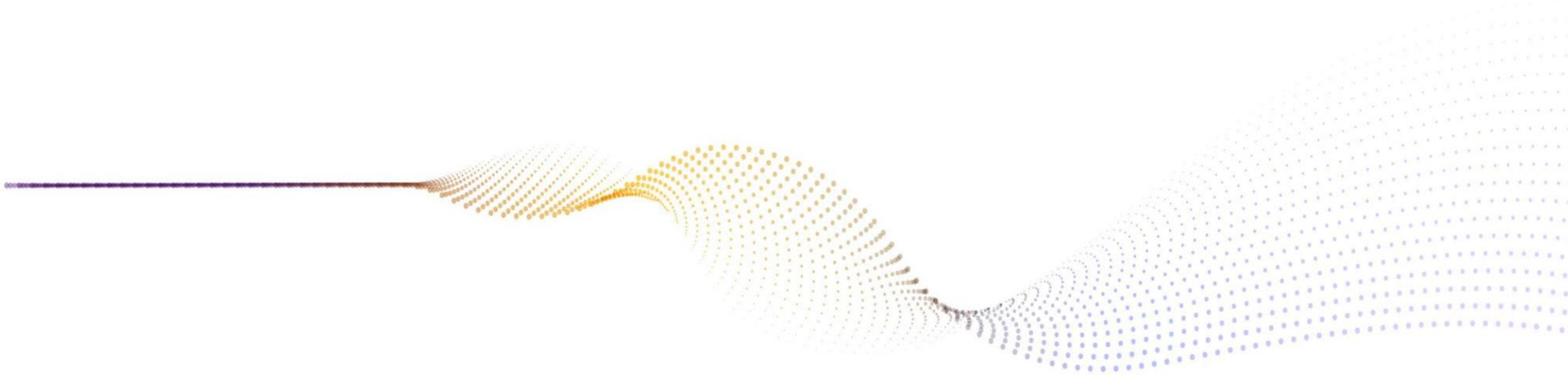


Artist's conception of a timeline

- Interested parties collaborating on workloads, infra, characterization, evaluation, requirements, TCO
- Open to all direct contributors: media vendors, storage controller vendors, OEMs, hyperscalers

Call to action

- Understand app implications
  - Storage criticality, bandwidth/latency/CPU load at compute server and storage server, granularity
    - Ex: well-tuned checkpoint perf doesn’t matter at compute node
    - Ex: KV$ perf at compute node may matter for summarization but not in many other cases
    - Ex: model load matters when workers are spun up, otherwise may be in memory already
- Bring us your applications and frameworks and help evaluate them!
  - Carefully evaluate criticality based on fundamental algorithm
  - Contribute evaluation infra and collaborate on characterization
  - See where there are opportunities to relieve bottlenecks to make storage more critical – check out Wen-mei’s book!
  - Map apps onto access patterns
- Strike the right balance in storage solutions across app domains
  - New objectives: IOPs, power efficiency in Storage-Next



Thank you for attending!

Please remember to rate this session. You get access the presentations at
http://sniadeveloper.org/conference