Scaling Workloads with a Memory Lake

Dr. Skyler Windh
Senior Systems Engineer – SMS Pathfinding
May 8th, 2025

micron

Content

Machine Learning's Scaling Challenge

Micron's Memory Centric Architecture Vision

Prototype Memory Lake System

Early Results of Memory Lake Scaling

Work supported by:

U.S. DEPARTMENT OF ENERGY | Office of Science

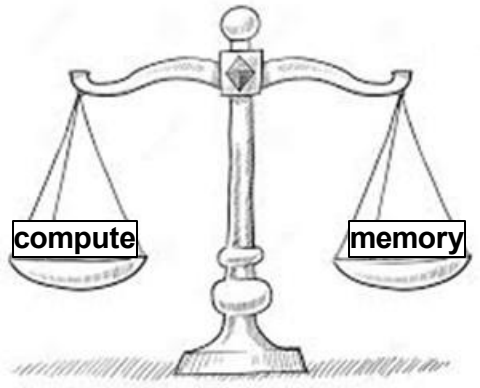
Machine Learning's Effect on Scaling

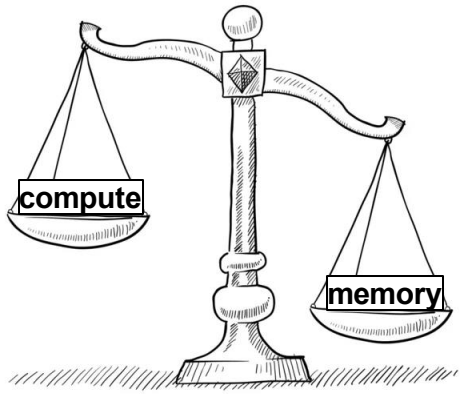
LLMs break balanced soft scaling

- Traditional High-Performance Computing (HPC) systems use a balanced scaling approach, where workloads have a set ratio of compute to memory capacity/bandwidth.
  - This ensures efficient resource use for *most* applications.

- This approach struggles with Machine Learning (ML) applications, especially Large Language Model (LLM) workloads, which have different phases of compute intensity vs memory bandwidth intensity

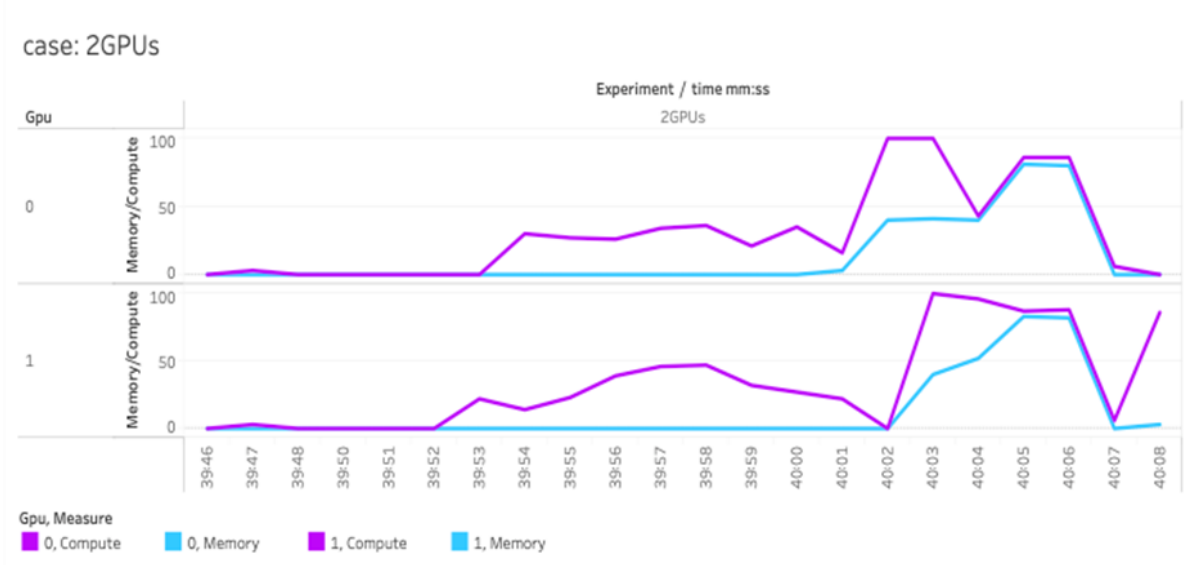
- This imbalance can lead to underutilized compute resources and excessive data movement, affecting performance and energy efficiency.



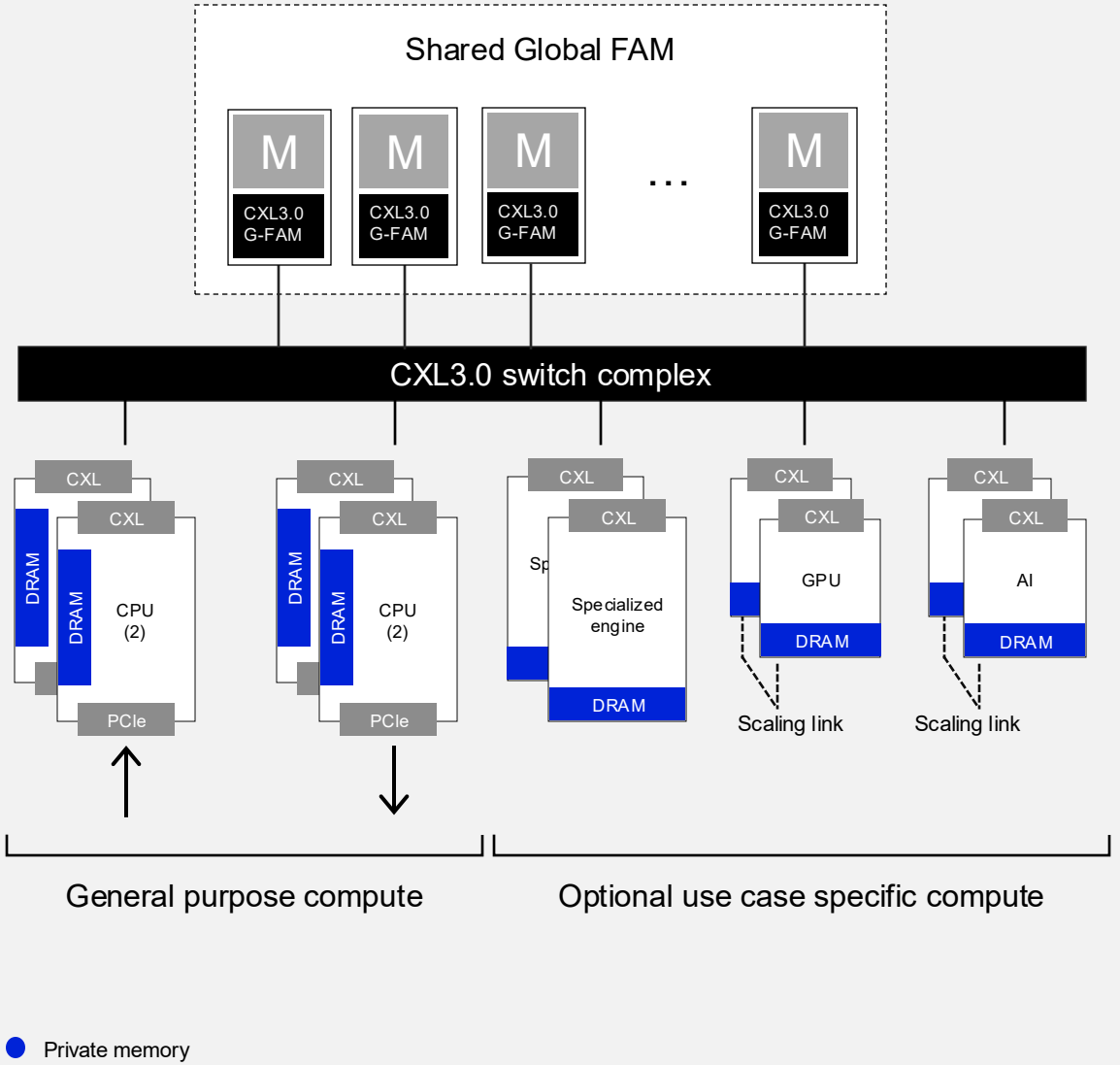


Stranded compute resources

- In this LLM example, the model is so large that it cannot be efficiently run on a 2 GPU configuration, resulting in underutilization of GPU compute resources.
- CPU offload (e.g., Microsoft's ZeRO) is a technology that enhances the efficiency of distributed training for deep-learning models. It achieves this by offloading some data from the GPU to the CPU, enabling larger model sizes per GPU to be trained.
- The results with GPU + CPU offloading show complete utilization of the GPU compute power.



CXL* enables a memory-centric system architecture



*These concepts are not limited to just CXL, but could be implemented on any fine-grained load/store fabric

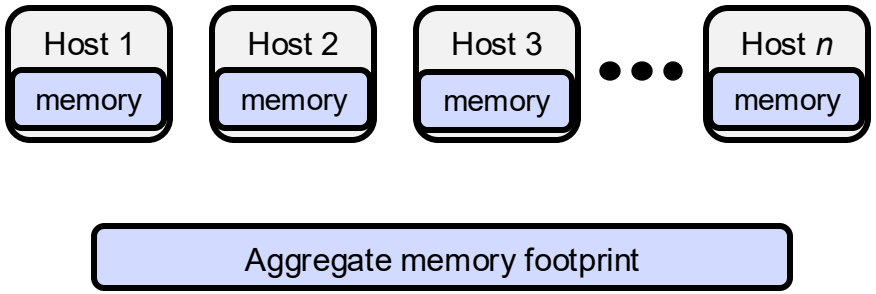
Addressing AI/ML scalability challenges with Global Fabric Attached Memory

- Imbalanced memory and compute requirements lead to underutilized compute resources and excessive data movement, affecting performance and energy efficiency.

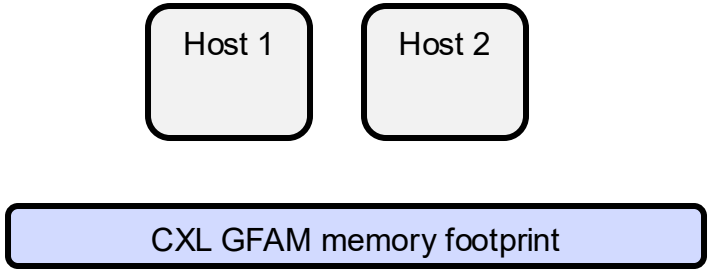
- Global Fabric Attached Memory (GFAM) allows independent scaling of compute and memory resources, leading to efficient resource utilization for memory-intensive workloads like LLMs.

- GFAM also minimizes the need for data movement.
  - Prepared data can be left in GFAM in zero-copy formats (e.g., NumPy Ndarray, pandas DataFrames, Arrow, etc.)

- However, adopting GFAM requires strategic **placement of memory and compute resources** and potential **software stack adjustments**.



Large # of compute resources to achieve required total memory

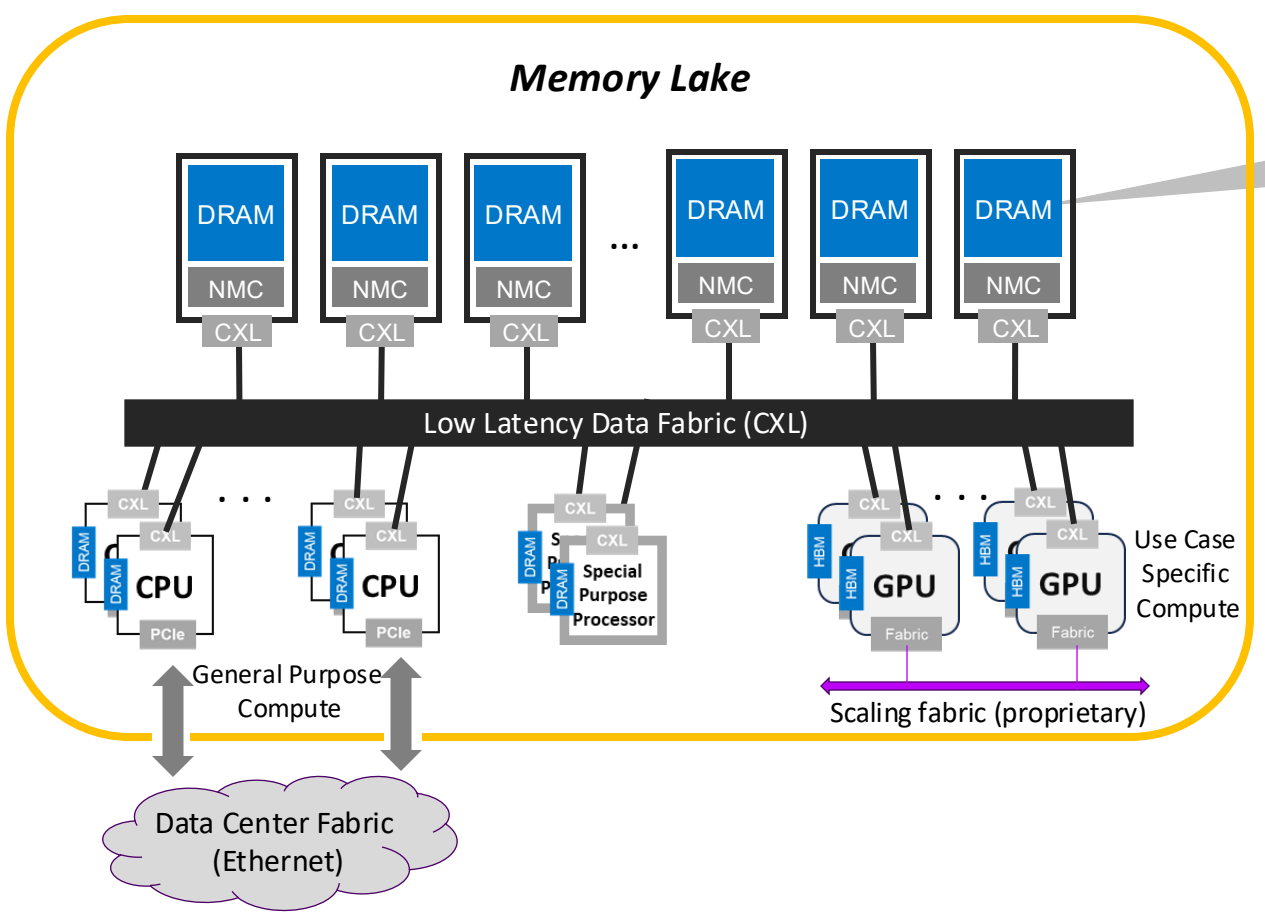


Memory size independent of # of compute resources

Micron’s Memory Centric Pathfinding Architecture

Micron's Memory Centric Architecture Vision

Enabling scaling of emerging applications



"BADS" module
DRAM
Near Memory Computing

- Very large, Scalable TB to PB
- Global In-Memory Data Store
- Zero copy Data Sharing
- Near Memory Compute (NMC)
- NMC Synchronization mechanism
- Usage specific topologies possible
- LD/ST access with <500 nsec access time

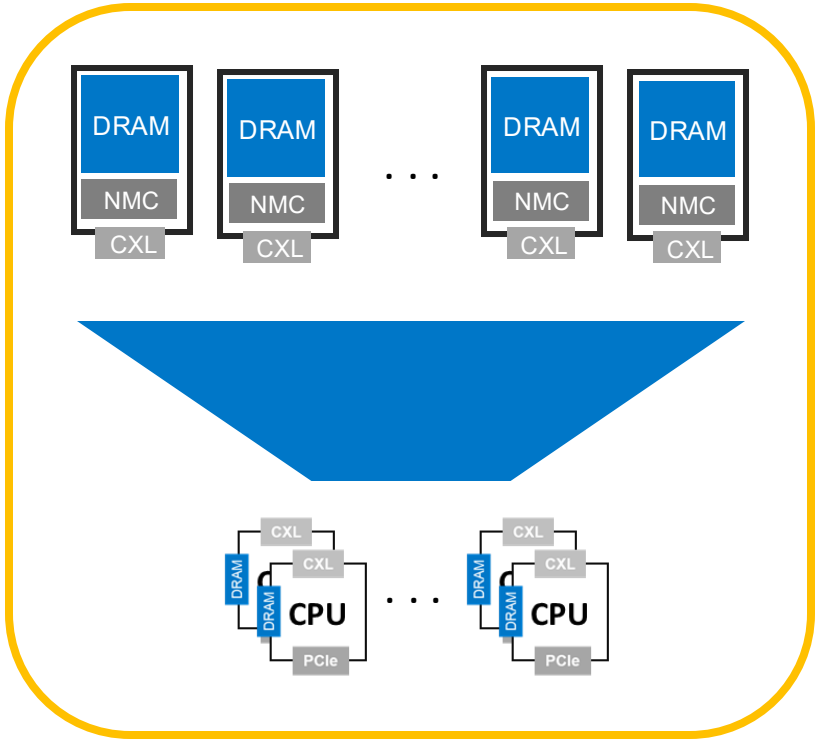
BADS – Byte Addressable Data Store

NMC – Near Memory Computing

Motivation for near memory computing

- Leverage excess disaggregated memory bandwidth to reduce application time-to-solution
- Reduce data transferred to host systems
  - Less fabric cost / energy consumption
- NMC on local memory
  - Reduce memory access latency
- Command Manager (CM) for dispatching
  - Memory Atomics, Event Messaging, Thread dispatching
- Custom RISC-V barrel processor
  - Latency tolerant random memory access (Graph analytics)
  - CXL device peer-to-peer accesses

Higher Aggregate Memory Bandwidth



Lower Aggregate Processor Bandwidth

2-10x Excess Memory Bandwidth

Three-tier computational model

1. Host servers

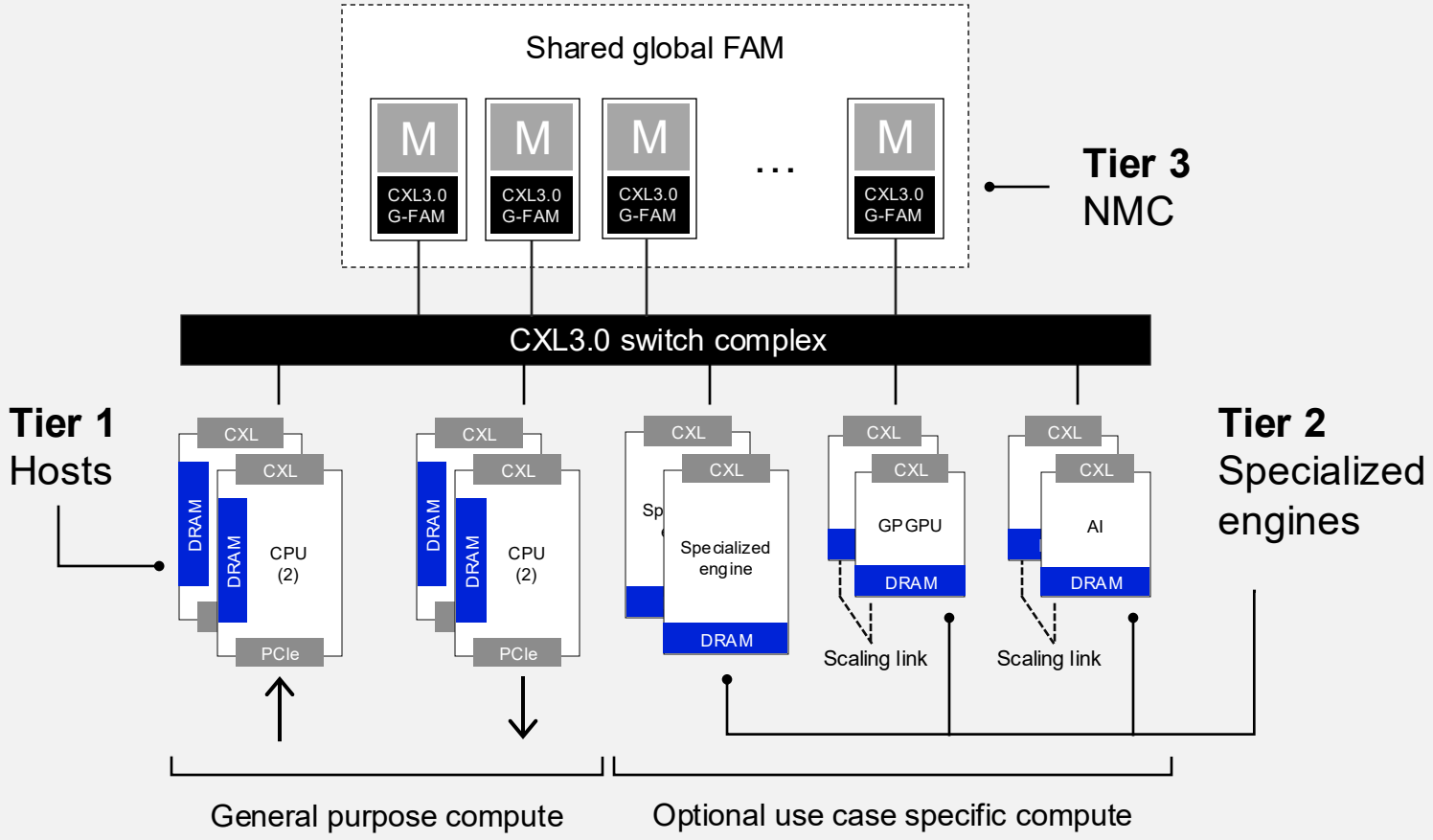
- General purpose compute
- Handles input/output for system
- Manages specialized compute resources

2. Specialized engines

- Domain-specific compute engines
  - GPGPU
  - AI
  - Other

3. Near-memory compute

- Processing within the memory module
- Leverage local memory bandwidth



Why heterogeneity? One size does not fit all

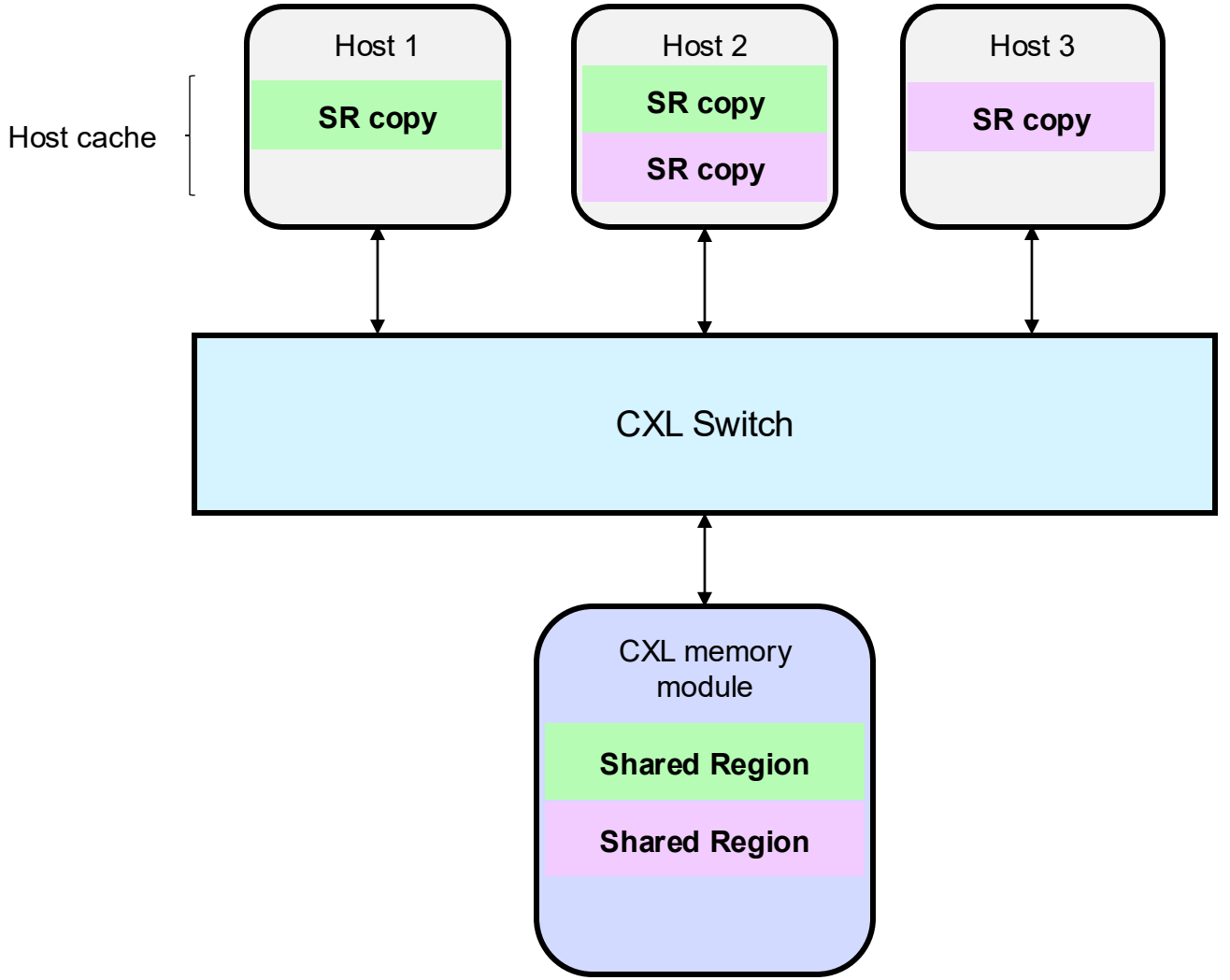
- Today, GPGPUs are used by applications in kernels with high compute density
- In the future, with CXL 3.0 fabrics, memory access latency will impact application performance
  - Large CXL 3.0 fabrics could have idle latencies in the micro-second range

How will fabric latency be mitigated to optimize performance?

1. **Modern processor have prefetch instructions**
   Limited value as the access latency increases – ability to see far enough in advance is limited as latency grows
2. **Modern processor have data movers**
   Good for large transfer size – small transfers will not require enough outstanding requests to overcome latency
3. **Near-memory compute (NMC)**
   Good for applications with sharded data – applications where data can be sharded can take advantage of NMC
4. **Specialized engine with latency tolerant compute model**
   Good for applications where data sharing is not possible such as graphs or sparse data structures

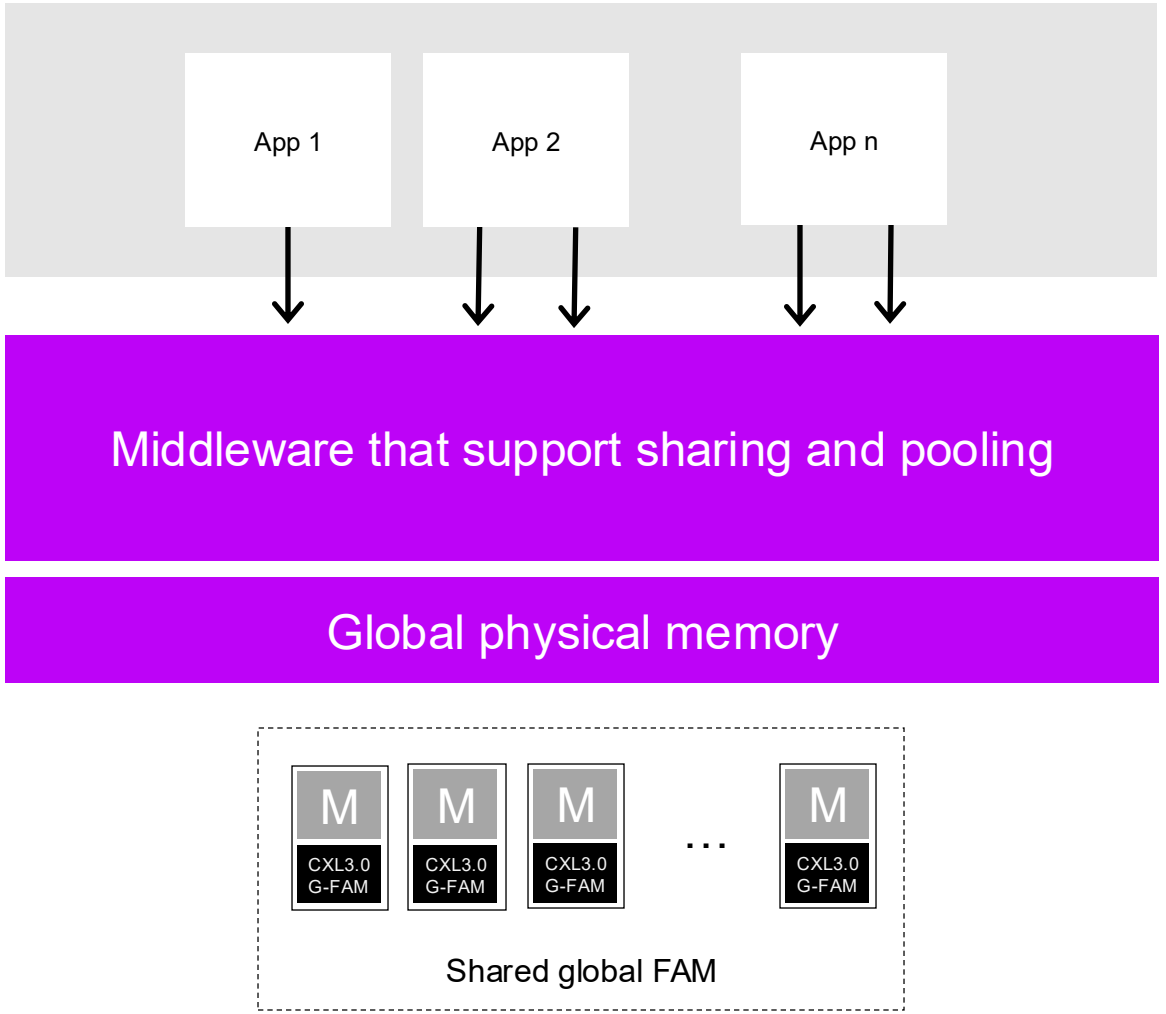
What about coherency?

- CXL 3.0 supports both hardware- and software-based coherency for shared memory
- For small systems hardware coherency is feasible
- For large systems general hardware coherency is not feasible
  - A single device error brings down the entire cluster
  - Load and store instructions have specific timeout requirements
  - Smaller islands of coherency as a possible mitigation
- Vendors will likely start with software coherency



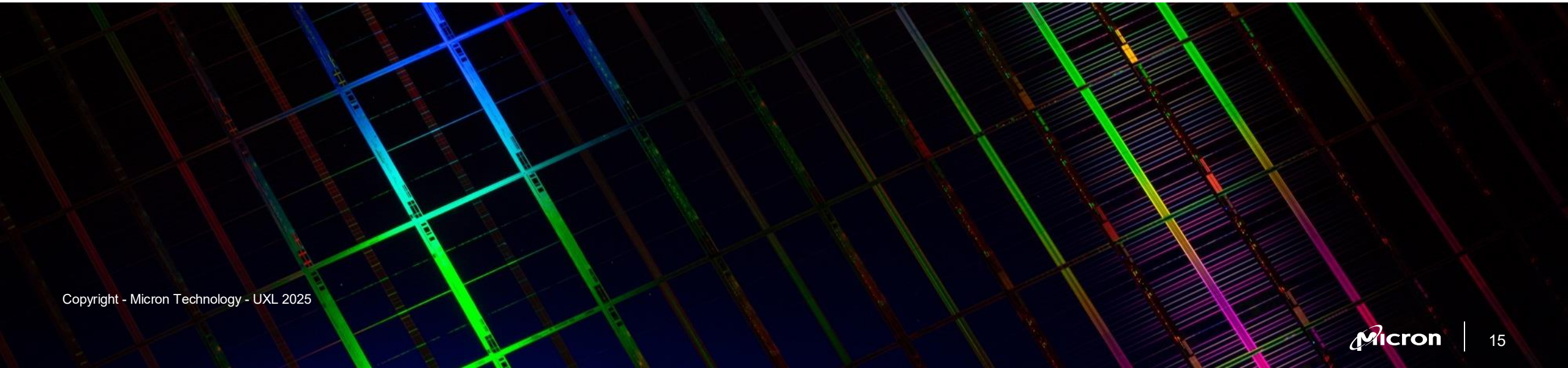
Emerging GFAM applications will require a new class of middleware

- Middleware that supports interhost memory sharing and pooling will be required
  - Check out the Famfs abstraction our colleague John Groves is working to upstream to linux
- Middleware must support mapping the GFAM memory into an application's virtual address space with sharing across hosts
- Visibility to system level interactions will be required to understand and optimize an applications performance



Application spaces benefiting from large shared memory

- Machine learning training pipeline
- Recommendation system
- Machine learning inference pipeline
- Large-scale graph analytics
- Graph inference system
- Similarity search
- Hyperdimensional data analytics (MDDB)
- Fraud detection
- High-frequency trading
- Quantum Chemistry Simulation



Use case
ML training pipeline

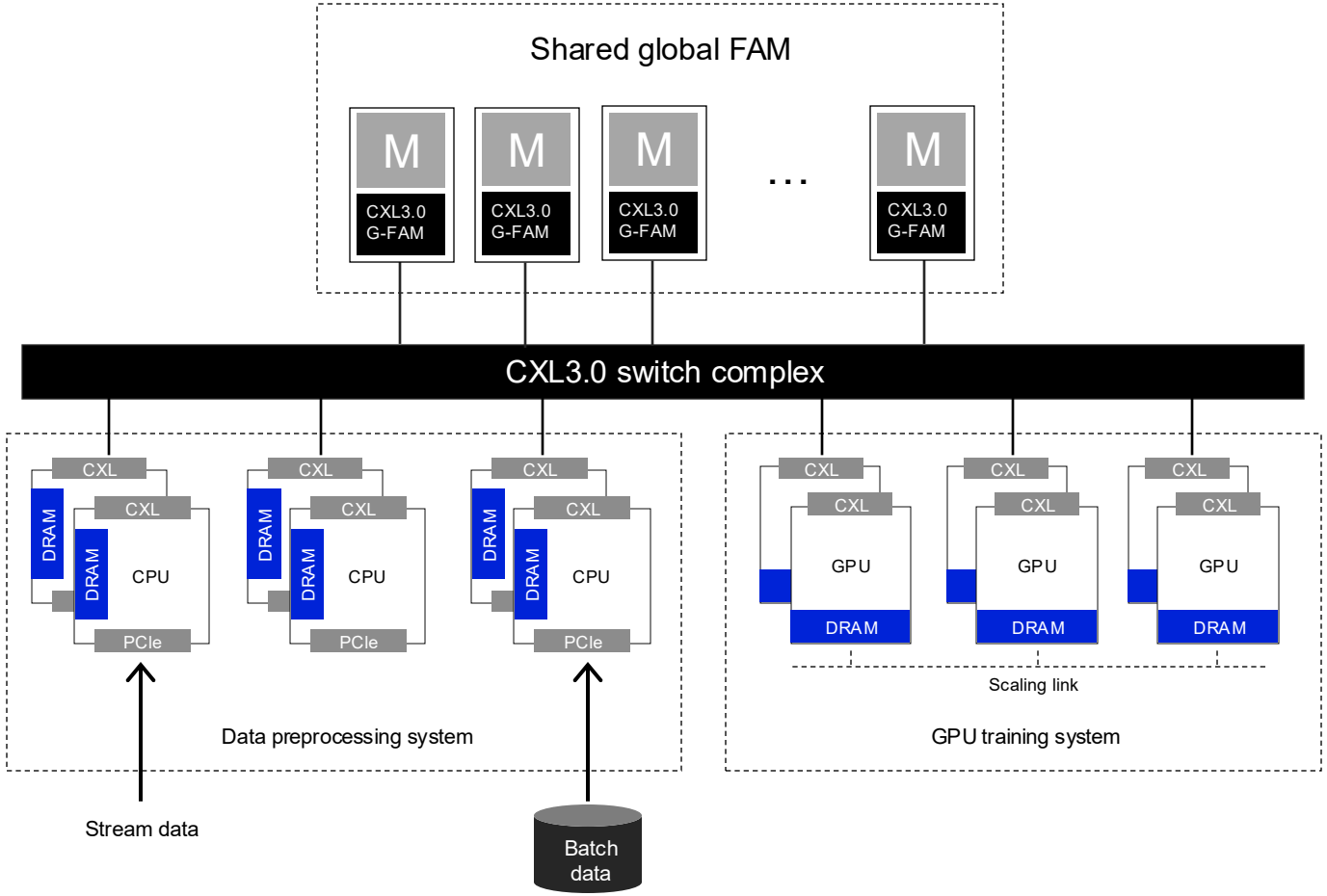
ML pipeline stages

Model
- Training
- Evaluation
- Deployment
- Retraining

Data preprocessing
- Cleaning
- Reduction
- Transformation

Example solution
- Shared global memory
- Data preprocessing cluster
- GPU cluster

CXL 3.0 shared memory benefits
- Time-to-insight improvement
  - In-memory connector between preprocessing and training stages
  - Reduces latency and improves bandwidth



Use case Recommendation system

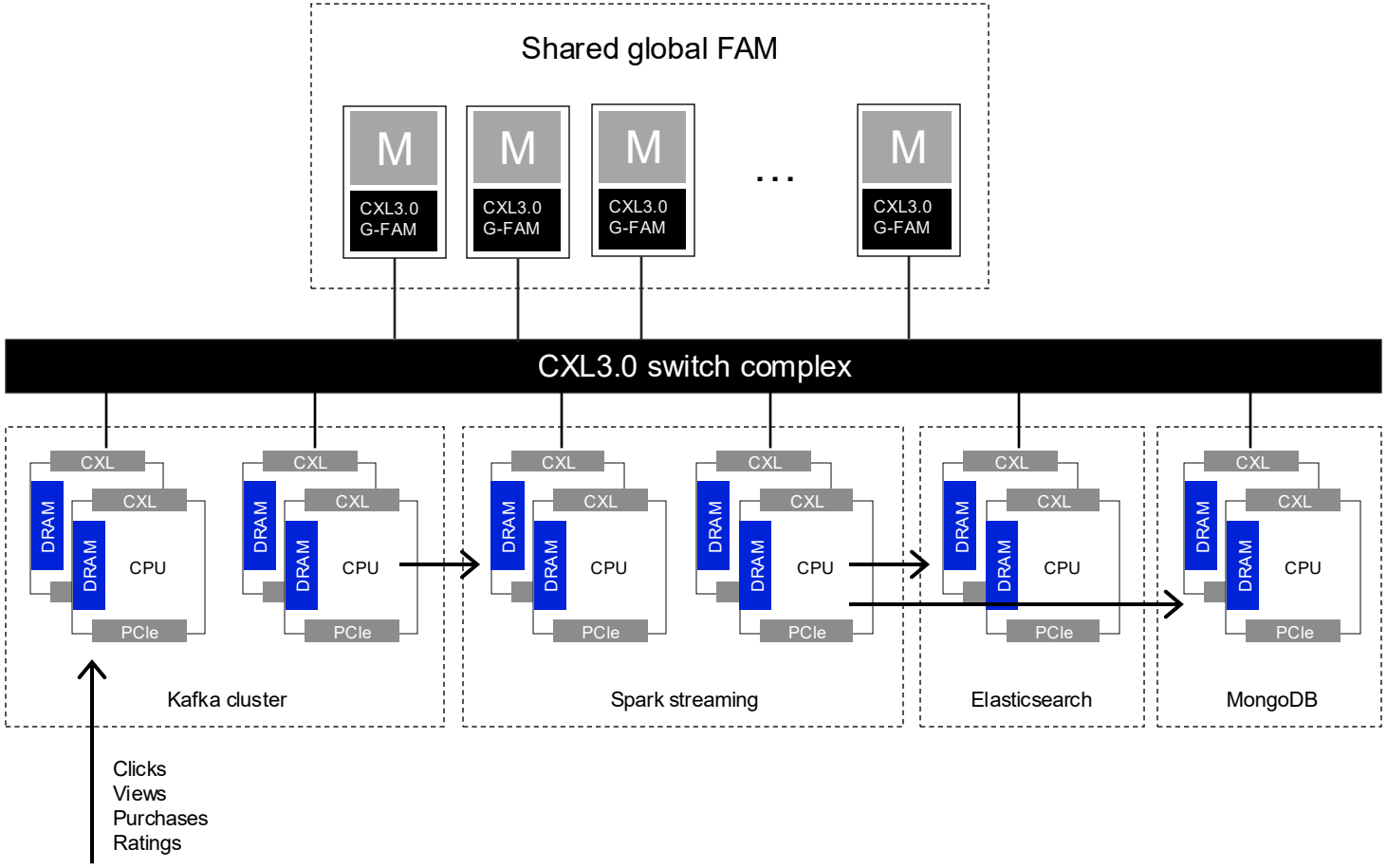
Information filtering system that seeks to predict the "rating" or "preference" that a user would give to an item

Example solution

- Tracking stored in a Kafka topic
- Spark streaming reads and process the data from Kafka
- Spark SQL writes the rating to MongoDB and Elasticsearch

CXL 3.0 shared memory benefits

- Time-to-insight improvement
  - In-memory connector between Kafka cluster and Spark streaming allows zero copy transfers
- Pub/sub acceleration
  - Improves performance and scale for delivery to subscribers



Visit Recommender system for more information

Use case
Graph analytics system

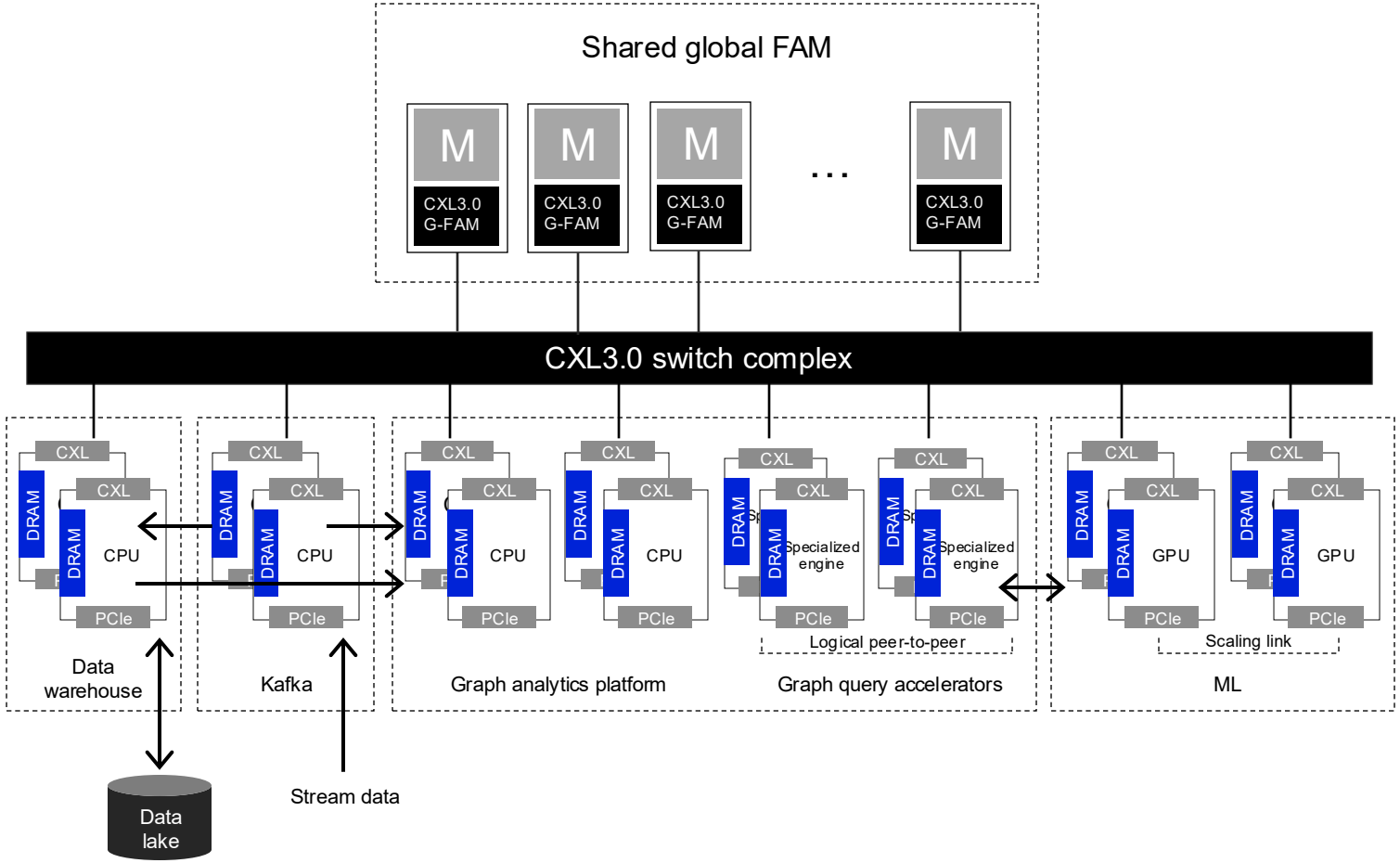
An analytics and machine learning (ML) solution that analyzes relationships in data to improve predictions and discover insights

Example solution

- Data warehouse – historical data
- Kafka – connectors to graph analytics and data warehouse
- Graph analytics platform – connector to ML
- ML – graph-computed features

CXL 3.0 shared memory benefits

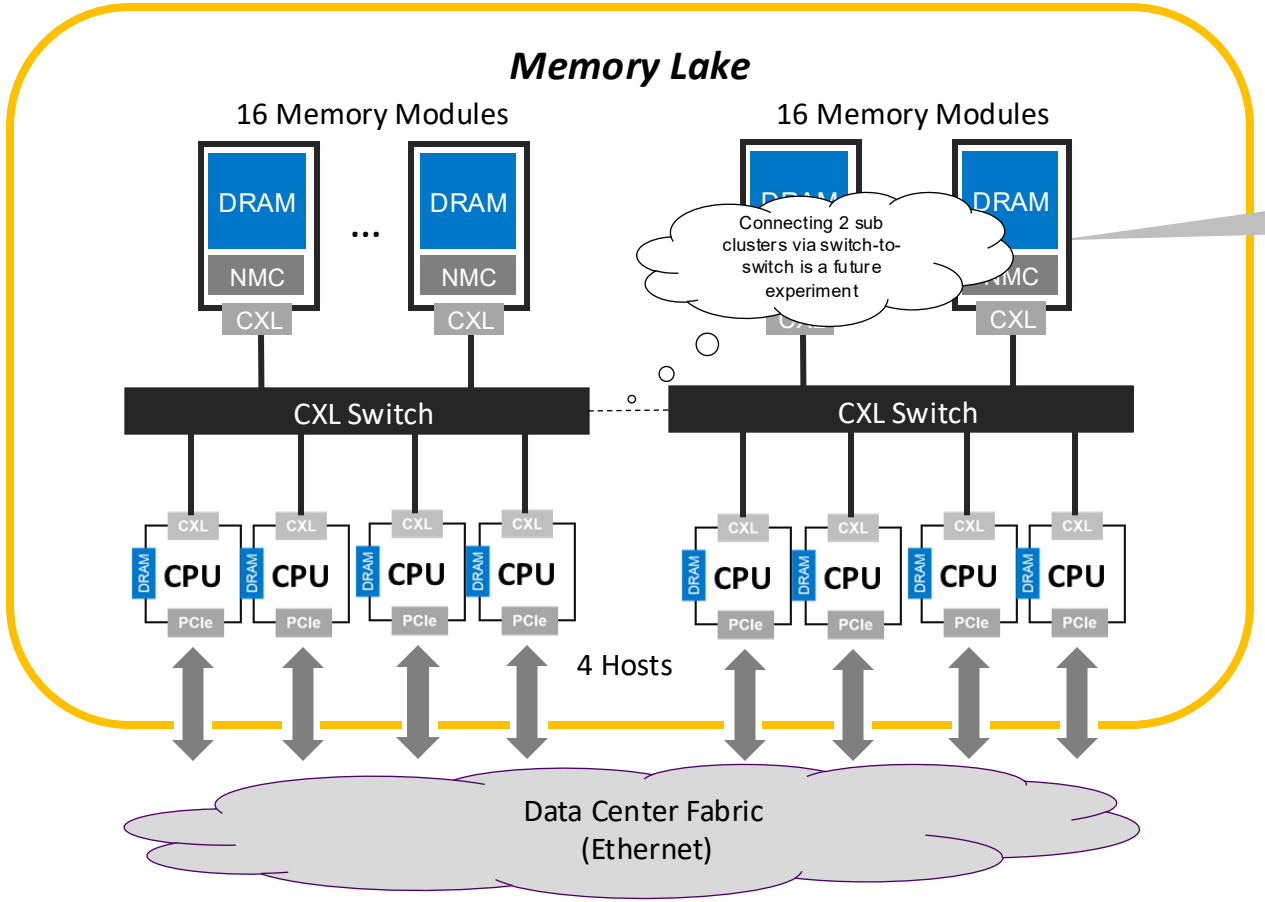
- Time-to-insight improvement
- In-memory connectors allow zero copy transfers
- In-memory graph analytics



Prototype Memory Lake System

Prototype Memory Lake System

Expected PNNL availability May 2025



"BADS" module
DRAM
Near Memory Computing

- Near Memory Compute
  - In memory atomics
  - Synchronization mechanism
  - Threading engines

XCONN XC50256 Technologies Switch Support for “CXL 2-Plus”

“CXL 2-Plus” is a term MemVerge uses to talk about Multi-host sharing with CXL 2.0 switches

CXL Specification Feature Summary

Features	CXL 1.0 / 1.1	CXL 2.0	CXL 3.0
Release date	2019	2020	August 2022
Max link rate	32GTs	32GTs	64GTs
Flit 68 byte (up to 32 GTs)	✓	✓	✓
Flit 256 byte (up to 64 GTs)			✓
Type 1, Type 2 and Type 3 Devices	✓	✓	✓
Memory Pooling w/ MLDs		✓	✓
Global Persistent Flush		✓	✓
CXL IDE		✓	✓
Switching (Single-level)		✓	✓
Switching (Multi-level)			✓
Direct memory access for peer-to-peer			✓
Enhanced coherency (256 byte flit)			✓
Memory sharing (256 byte flit)			✓
Multiple Type 1/Type 2 devices per root port			✓
Fabric capabilities (256 byte flit)			✓

Not Supported ✓ Supported

Compute Express Link® and CXL® are trademarks of the Compute Express Link Consortium.

CXL Sharing of Single Logical Device (SLD)

- XCONN’s EndPoint Mode
- Multi-host software cache coherency
  - CXL repeater cards
  - Cluster management server

[1]Sharing with Intel GNR and EMR processors requires directory updating to be disabled. Socket Local Memory access goes up to > 100ns

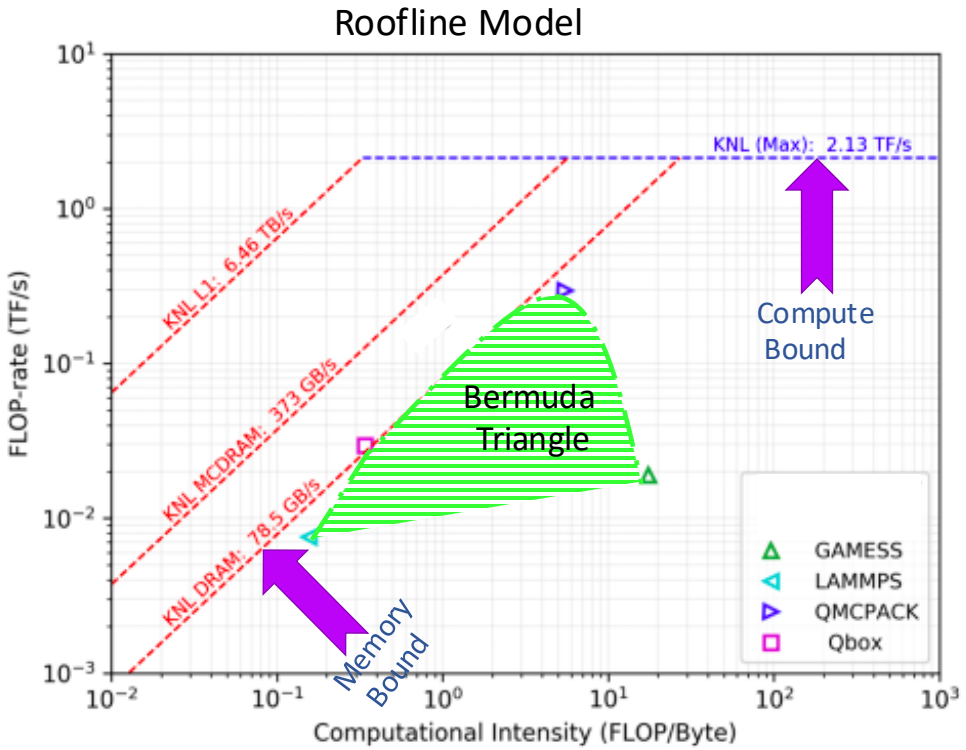
Two Clusters Running in the Lab

- Micron currently has 2 clusters running in the lab
  - Shared memory is stable
  - NMC drivers currently progressing through bring-up
  - Cabling is impressive

DOE's Accelerate (collab PNNL, MS, Micron) Computational Chemistry Performance Insights

- Roofline Models seem to indicate performance bottlenecks due to **memory wall**
  - Low to moderate computational intensity
  - Low locality, preventing use of performant higher memory levels

- Memory Wall Trifecta
  - Capacity
  - Bandwidth
  - Latency



Ref. ALCF Theta Cray XC40 (CUG 2019 J. KWACK, T. APPLENCOURT, C. BERTONI, Y. GHADAR, H. ZHENG, C. KNIGHT, S. PARKER)

mods by A. Márquez, PNNL

Slide provided by Andres Marquez, PNNL

Early Results of Memory Lake Scaling

“OpenMP with Remote Offload” on CXL Shared Memory outperforms RDMA

Linear scale out across servers using FAM by removing network copies

The OpenMP target construct allows for code regions to be executed on a device and to support multiple devices, device teams, and device synchronization

Micron has modified the LLVM compiler to enable executions of code regions on multiple x86 servers (based on prior research[1])

This modified compiler uses CXL FAM to share memory which eliminates the need to copy data between servers

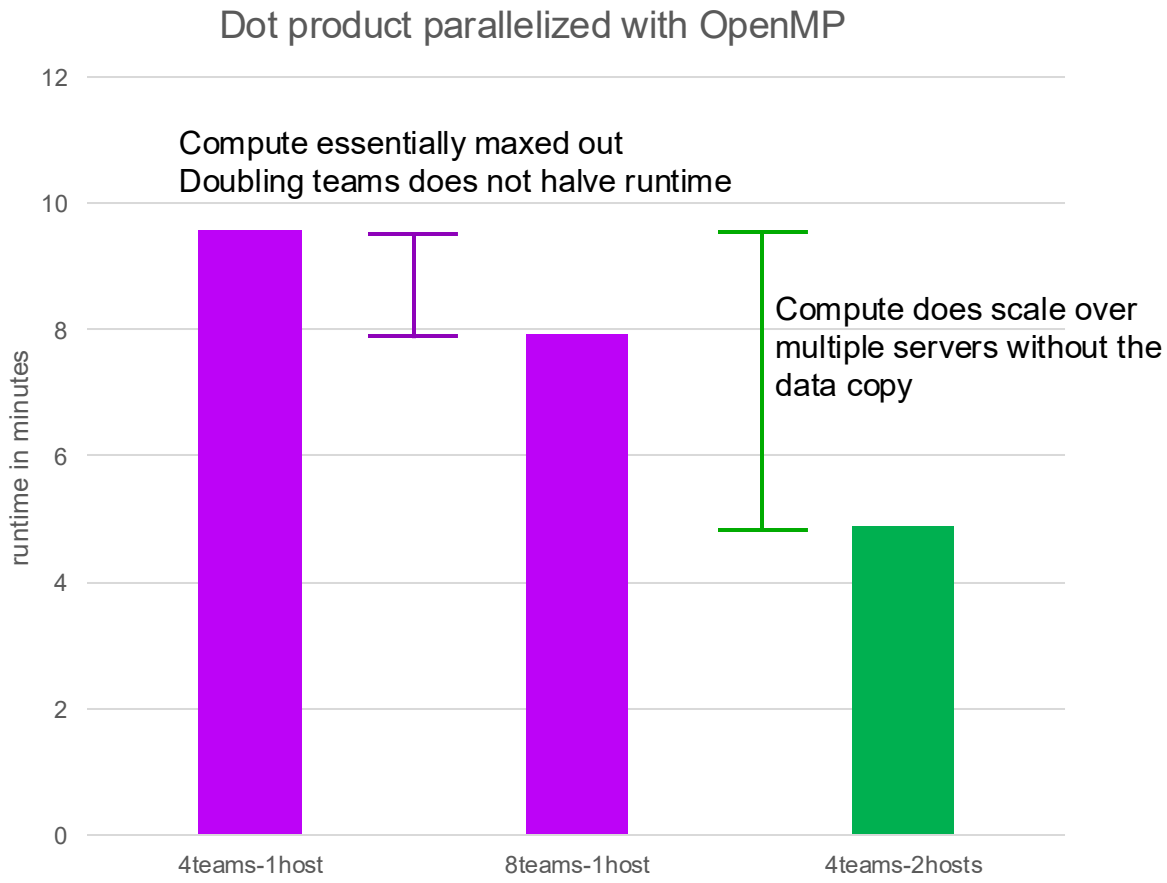
```
  for (i0 = 0; i0 < N; i0 += block_size) {
    // 4 teams start on remote host
    if (omp_get_team_num() < 4) {
//#pragma omp target device(omp_get_team_num()) map(to: B[:N], C[:N]) defaultmap(tofrom:scalar)
#pragma omp target device(omp_get_team_num()) map(to: B, C) defaultmap(tofrom:scalar)
#pragma omp parallel for reduction(+:sum) num_threads(32)
      for (i= i0; i < min(i0+block_size, N); i++)
```

Map full array by a copy

FAM enables mapping by pointer

This makes it possible to run existing OpenMP applications across a CXL FAM cluster.

Requires the modification/addition of OpenMP target pragmas, adding SW coherency where hosts share data (can be challenging), followed by recompilation.



1 Patel, A., Doerfert, J. (2022). Remote OpenMP Offloading. In: Varbanescu, AL., Bhatele, A., Luszczek, P., Marc, B. (eds) High Performance Computing. ISC High Performance 2022. Lecture Notes in Computer Science, vol 13289. Springer, Cham. https://doi.org/10.1007/978-3-031-07312-0_16

Chemistry Applications and Shared Memory

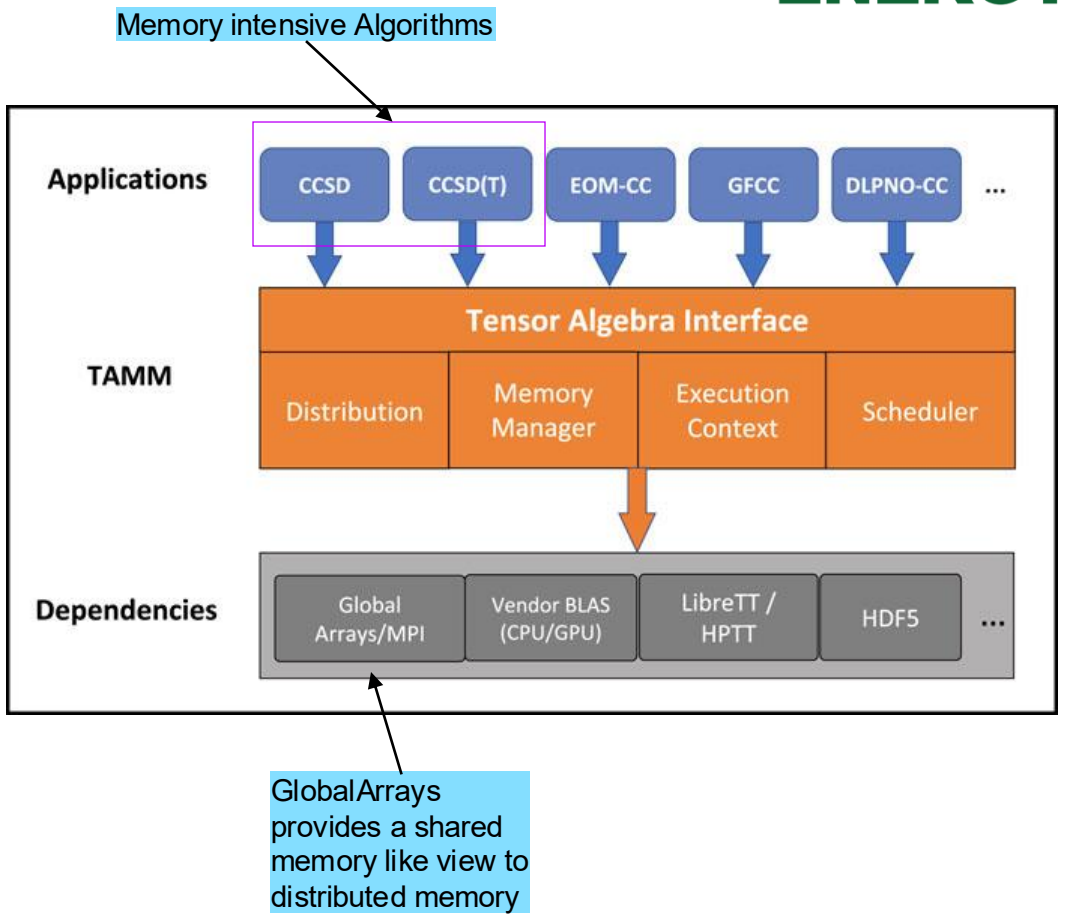
Porting to FAM saw a **~2.5X speedup** due to removing data copies and network buffering

Under the ACCELERATE contract with PNNL, Micron is helping port Chemistry apps to shared memory

Tensor **A**lgebra for **M**any-body **M**ethods (TAMM)[1]

- Provides a tensor algebra interface and core algorithms for chemistry workloads

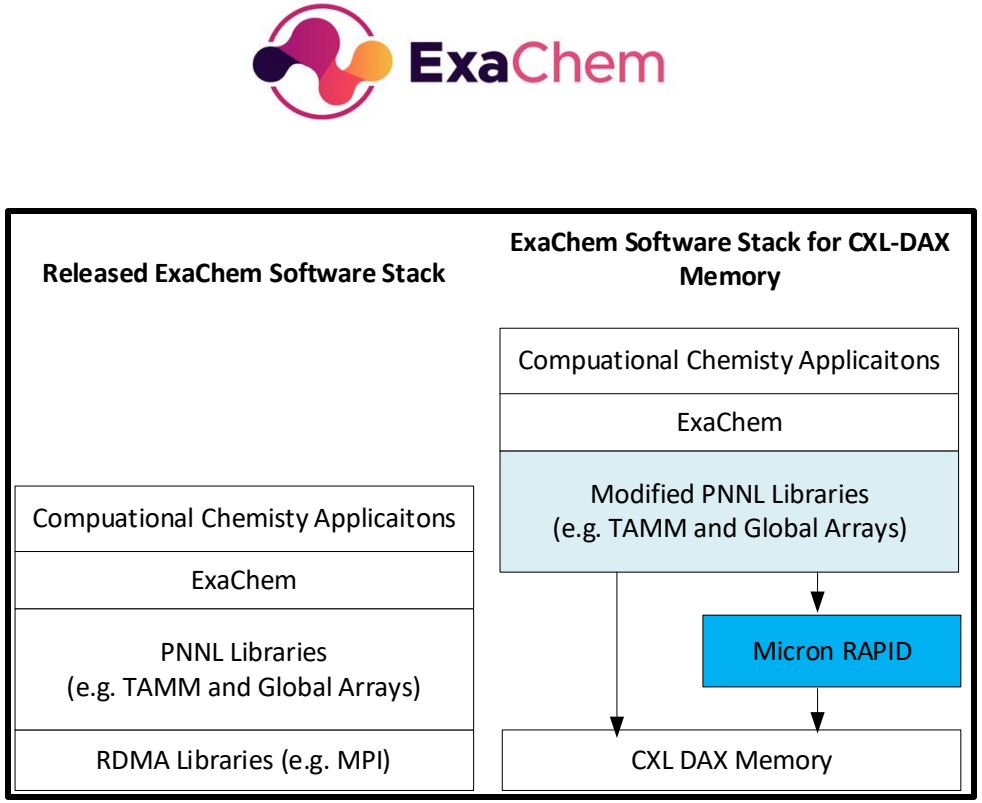
- CCSD algorithm was the first focus as it is **memory intensive**. Cleanly partitions over shared memory w/o SW coherency semantics



1 NWChemEx/TAMM: Tensor Algebra for many-body methods

Chemistry Applications and Shared Memory

Single node large memory clusters help customers solve problems



ExaChem[1] is a chemistry software that builds on top of TAMM

Micron recently finished porting to shared memory cluster using sw coherency (perf analysis pending)

- However, **single host-large memory pool is still providing useful science**

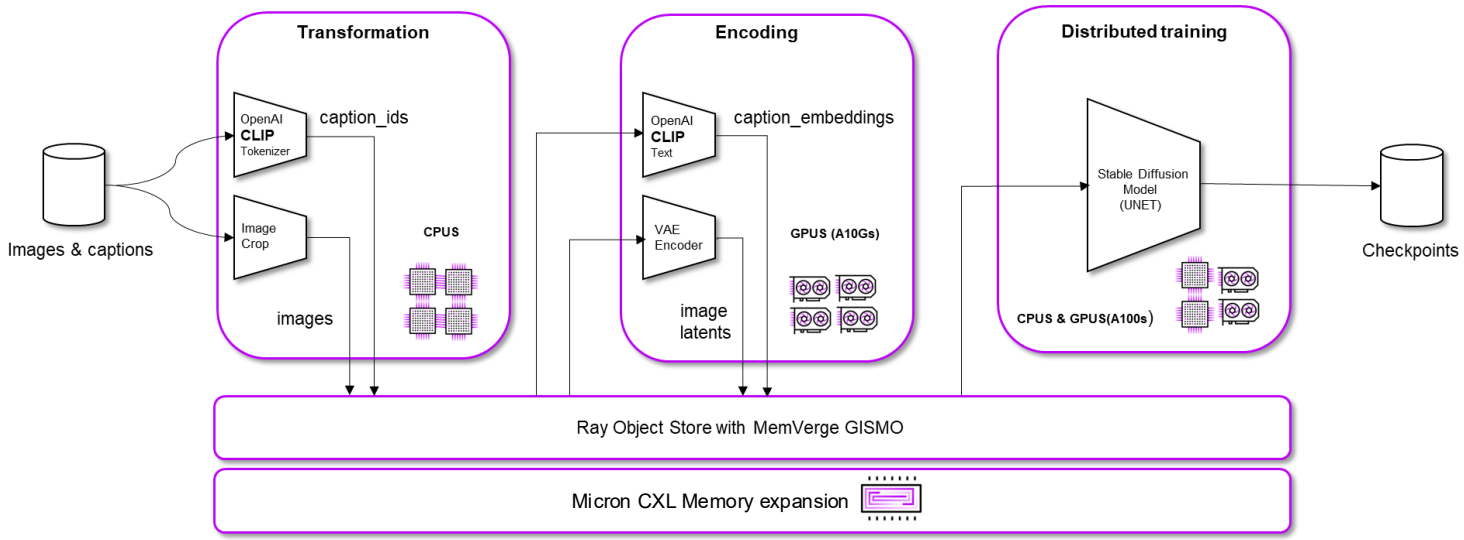
Memory intensive workloads are run on the Micron cluster and compute intensive work is run on the Microsoft Azure Quantum Elements (AQE) cluster

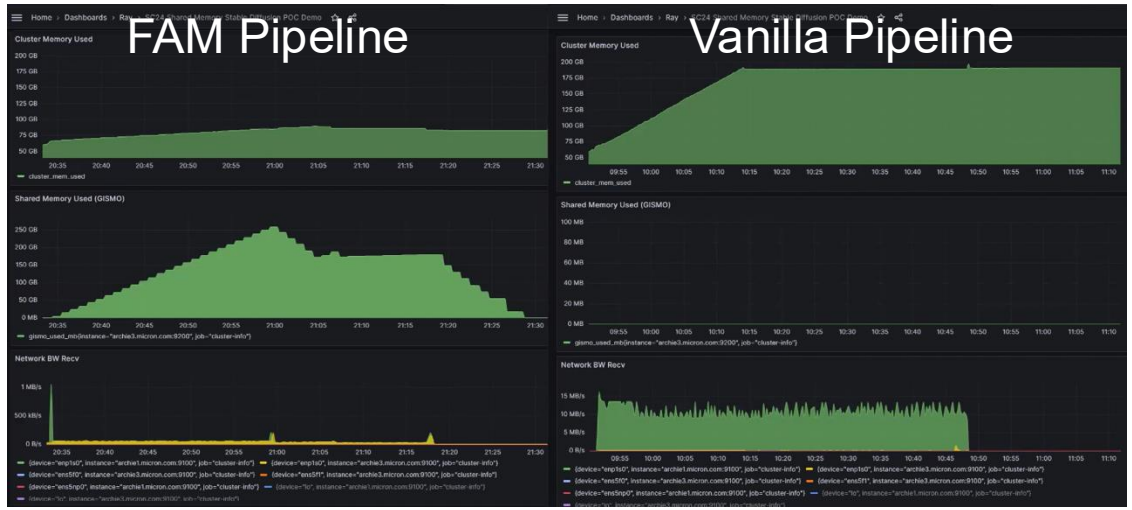
- 2 papers in progress with PNNL and Microsoft, one related to processing PFAS “forever chemicals”

1 ExaChem/exachem: Open Source Exascale Computational Chemistry Software

SC24: Stable Diffusion Training with Ray/MemVerge GISMO

Early POC – training using CXL switched shared memory between heterogeneous GPU systems.





- **No code changes to the Ray pipeline**
  - Install Ray with GISMO[1] support
  - Startup cluster with feature turned on
  - Run same code that ran on vanilla Ray
- Share memory between heterogenous systems (e.g., A10G node and A100 node)
- Avoid spilling to storage when server hits memory capacity
- Perf was +/- 10% of vanilla. GPUs were the bottleneck and CXL switch is heavily throttled. Looks promising to revisit with latest switch and heterogeneous GPU setup

1 Memory Machine CXL Fabric-Attached Memory - MemVerge

Summary

Middleware can simplify future scalable programming

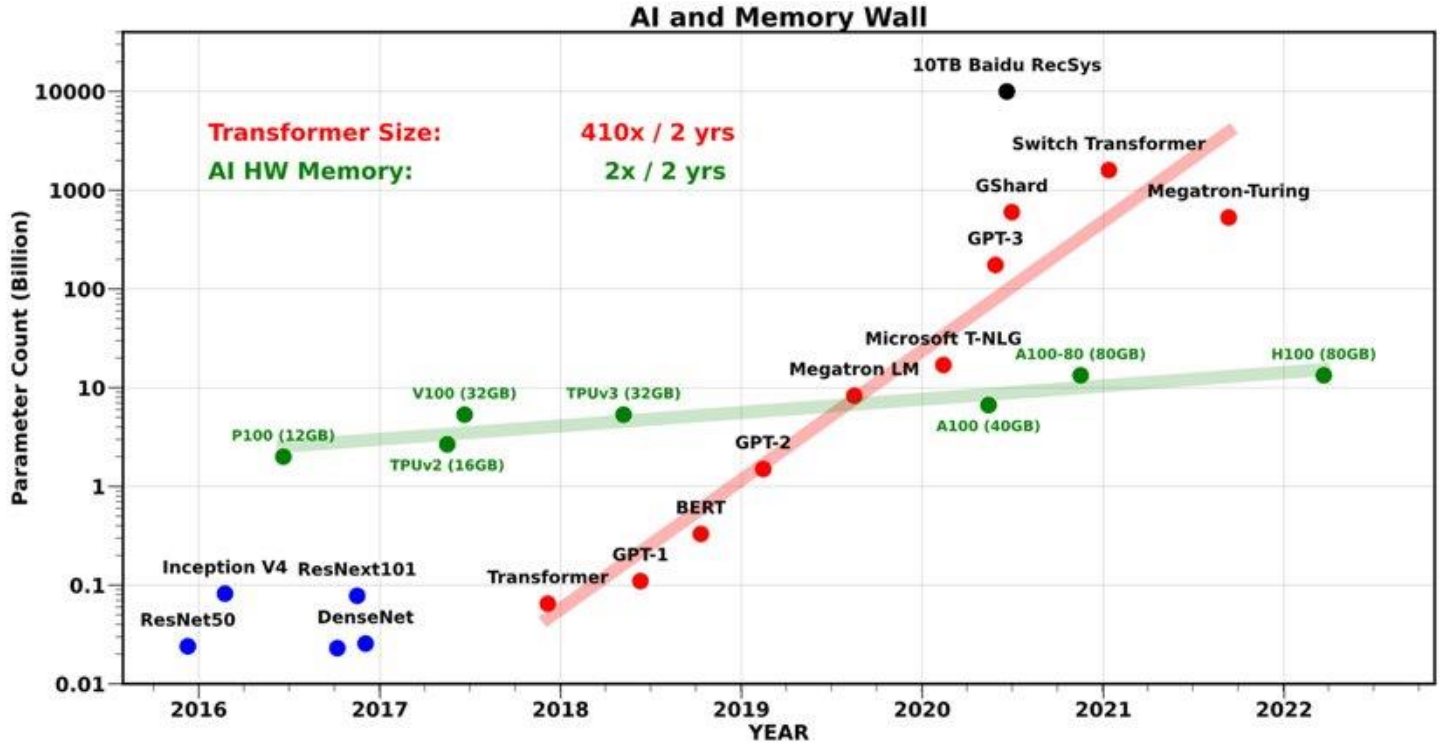
- Memory and compute allocation is a challenging aspect of scaling large applications

- Sharing fabrics enable finer-grained placement and utilization of both memory and compute resources
  - Enables sending “compute threads” to where the memory resides
  - Enables plugging in custom accelerators for specific compute needs
  - Reduces data movement and improves efficiency

- Shared memory software coherency can be hard
  - The flush/fence pattern is simple, but finding all the memory touch points is hard
  - Clean abstractions and debug tools are sorely needed

AI has hit a memory wall

- Compute needed to train Transformer models has been growing at a rate of 750x/2yrs.
- There is an emerging challenge with training and serving these models: memory and communication bottlenecks.
  - AI applications are becoming bottlenecked by intra/inter-chip and communication across/to AI accelerators rather than compute.
- LLM model sizes has been increasing at a rate of 410x every 2 years.
- Large Recommendation System models have reached O(10) TB parameters. DRAM memory has only scaled at a rate of 2x every 2 years.



*Source: Gholami A, Yao Z, Kim S, Mahoney MW, Keutzer K. AI and Memory Wall. RiseLab Medium Blog Post. University of Califonia Berkeley. 2021, March 29.

LLM Tokenization

In-GFAM object creation/consumption scales to multiple servers, requires single transformation

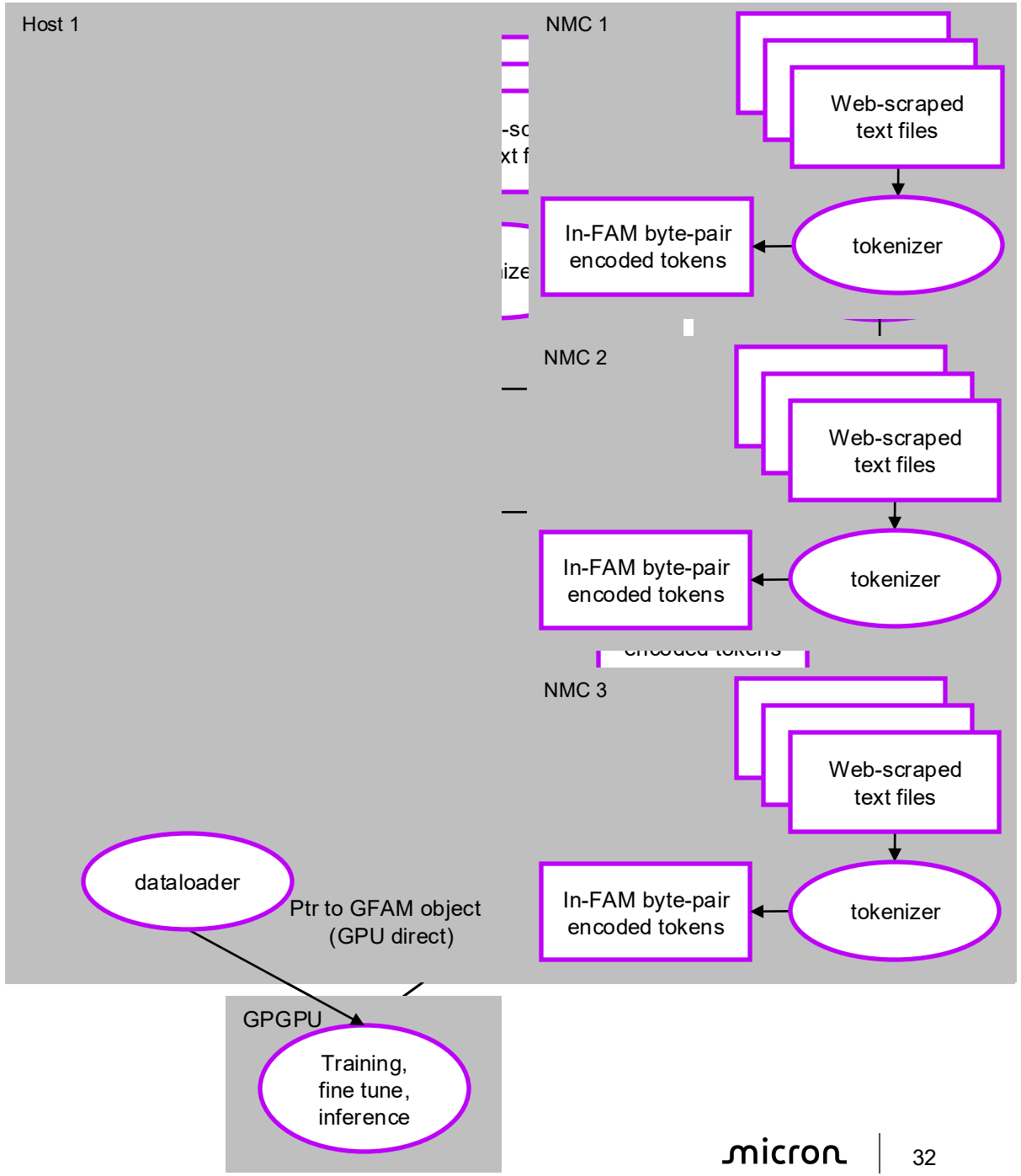
Tokenization leveraging GFAM provides maximum parallelization opportunities. Bandwidth usage is minimized with in-FAM memory object creation/consumption.

Near-memory compute capabilities facilitate further optimization

- **Shard input text files onto separate NMC devices**
- **In-GFAM token creation entirely within a NMC device**

Hugging Face GPT-NeoX tokenizer and c4 dataset

- 1024 files, 770GB
- Tokenization scaleup limited by memory requirements
  - ~175GB per tokenizer



CXL FAM Scaling Demonstration

Linear scaling while scaling out

Scaling out compute across servers has been limited by the overhead of copying data.

Using shared CXL FAM provides the utility of shared DRAM while **enabling computational scale out across servers**.

This also facilitates scale out using *domain specific accelerators* (e.g., image scaler)