SNIA DEVELOPER CONFERENCE
SDC 25
By Developers FOR Developers

Hyatt Regency Santa Clara, CA
September 15-17, 2025

Towards memory efficient RAG pipelines with CXL technology

Arun George, Roshan Nair, Vikash Kumar and Siamak Tavallaei

www.sniadeveloper.org

SAMSUNG

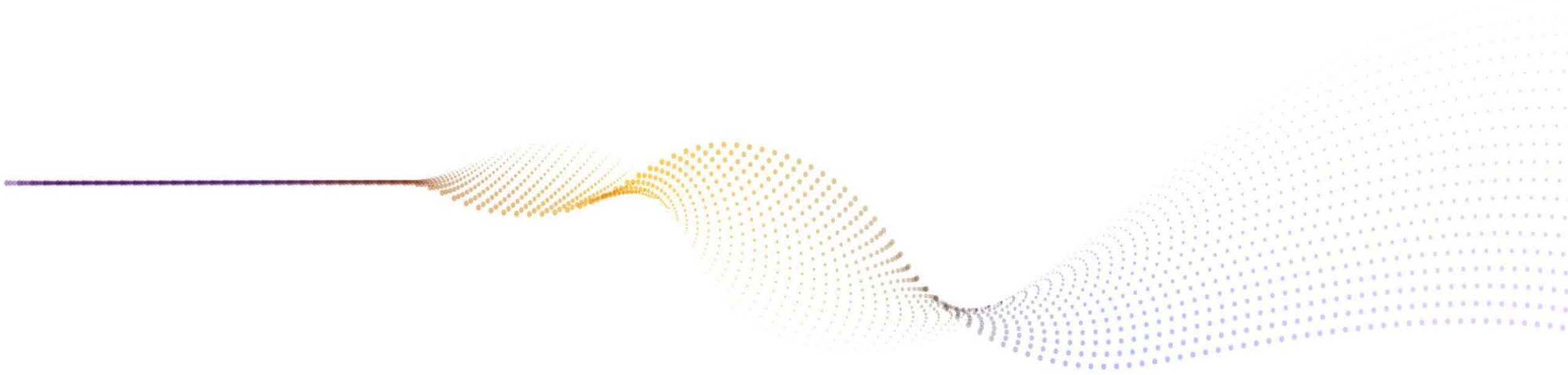
Motivation: Why we have this session?

- AI Infra Market
  - The global AI infrastructure market is projected to grow fast
  - From USD 46.15 billion in 2024 to USD 356.14 billion (7X) by 2032 [1]
- Memory fabrics is a key enabler for AI Infra
  - Proprietary standards: NVLink, UALink, Infiniband etc.
  - Open standard: CXL
  - Enables composable memory infra for AI
  - Opens new possibilities for
    - Scalable memory expansion
    - Flexible, disaggregated memories
    - TCO savings, use of cheaper memory media etc.
- CXL
  - We discuss some interesting use cases of CXL in AI Infrastructures.

[1]: Source: https://www.fortunebusinessinsights.com/ai-infrastructure-market-110456

What will we cover?

- RAG Pipelines in AI use cases
- Memory demand scenarios in RAG
- CXL memory pooling
- CXL Compressed memories
- Takeaways



RAG Pipelines in AI Use cases

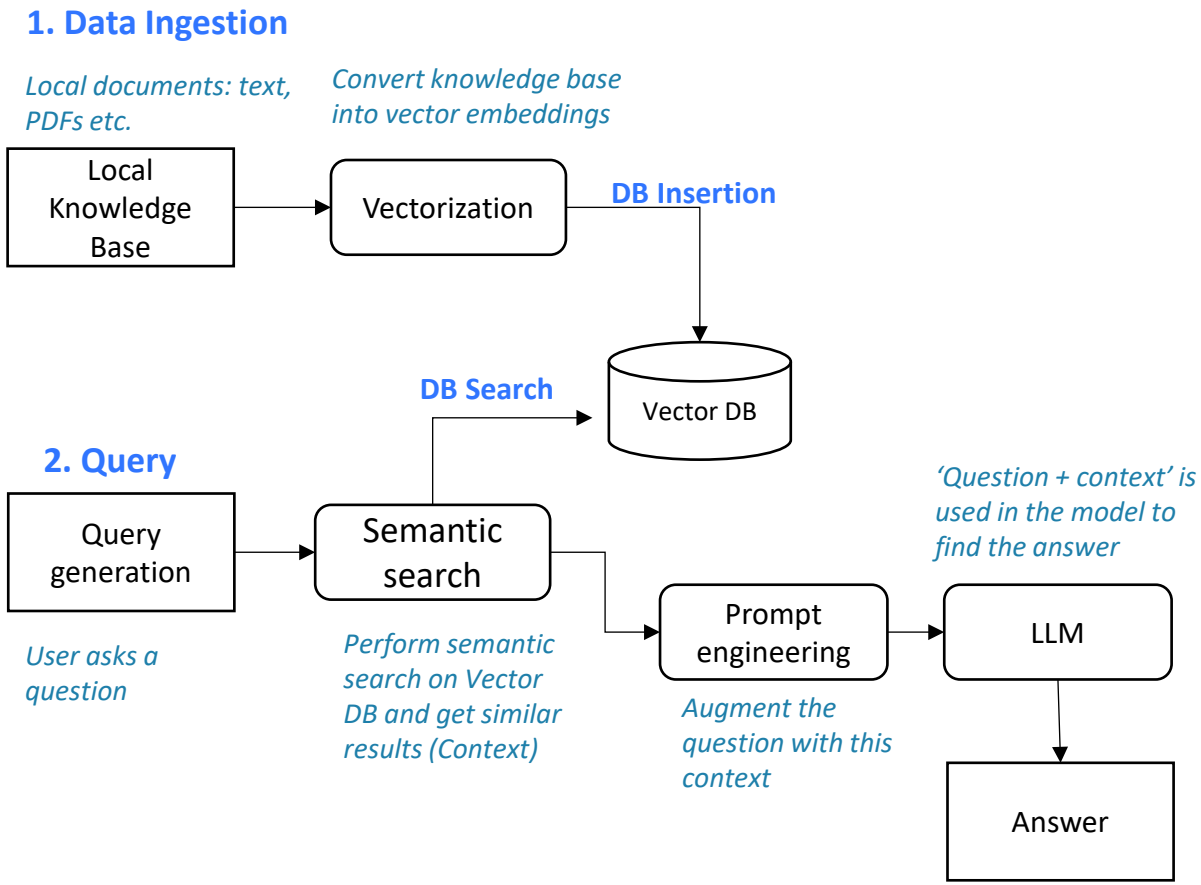
Memory demand scenarios in RAG

Primer: RAG Pipeline on AI Use-cases

- Why is RAG relevant?
  - Improves AI workflows
  - Generally employed on LLM models
- What are LLM models good at?
  - Generating content through natural language processing (NLP)
- What if we query LLM on a data that it never encountered before?
  - This data might be domain specific information or just up-to-date information
  - The LLM model will probably hallucinate
- RAG (Retrieval Augmented Generation)
  - RAG acts as a framework for the LLM to provide accurate and relevant information when generating the response.

RAG: Architecture overview

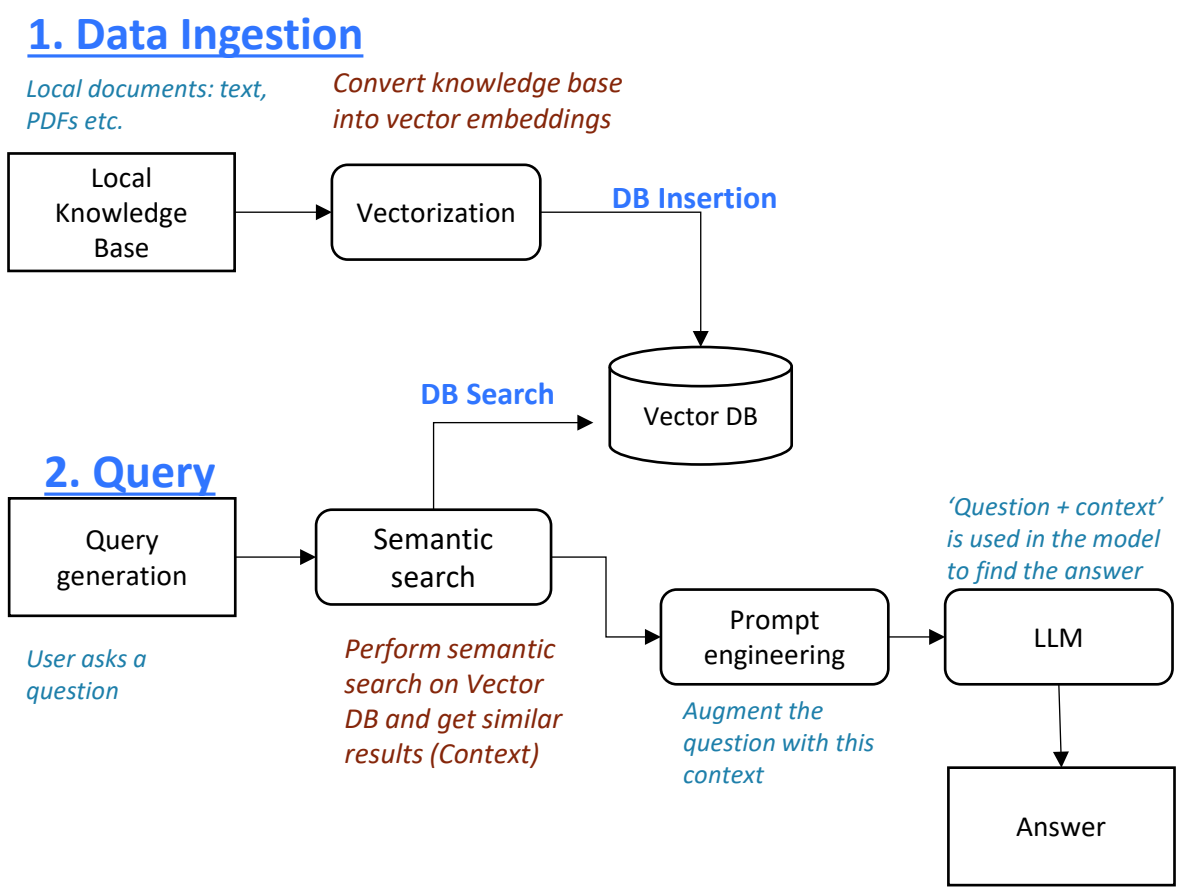
- "G" – Generation
  - It is the LLM which generates text in response to a user query
- "RA" - Retrieval Augmented
  - Instead of relying on what the LLM has been trained on, RAG provide the LLM some additional context
  - This prevents hallucinations
- Data Ingestion Phase
  - Ingest and vectorize local documents
  - Store in a VectorDB for fast retrieval
- Query Phase
  - Leverage this context for user prompts
  - VectorDB search for finding context
  - LLM is fed 'prompt + context'



SAMSUNG

Transient memory usages in RAG

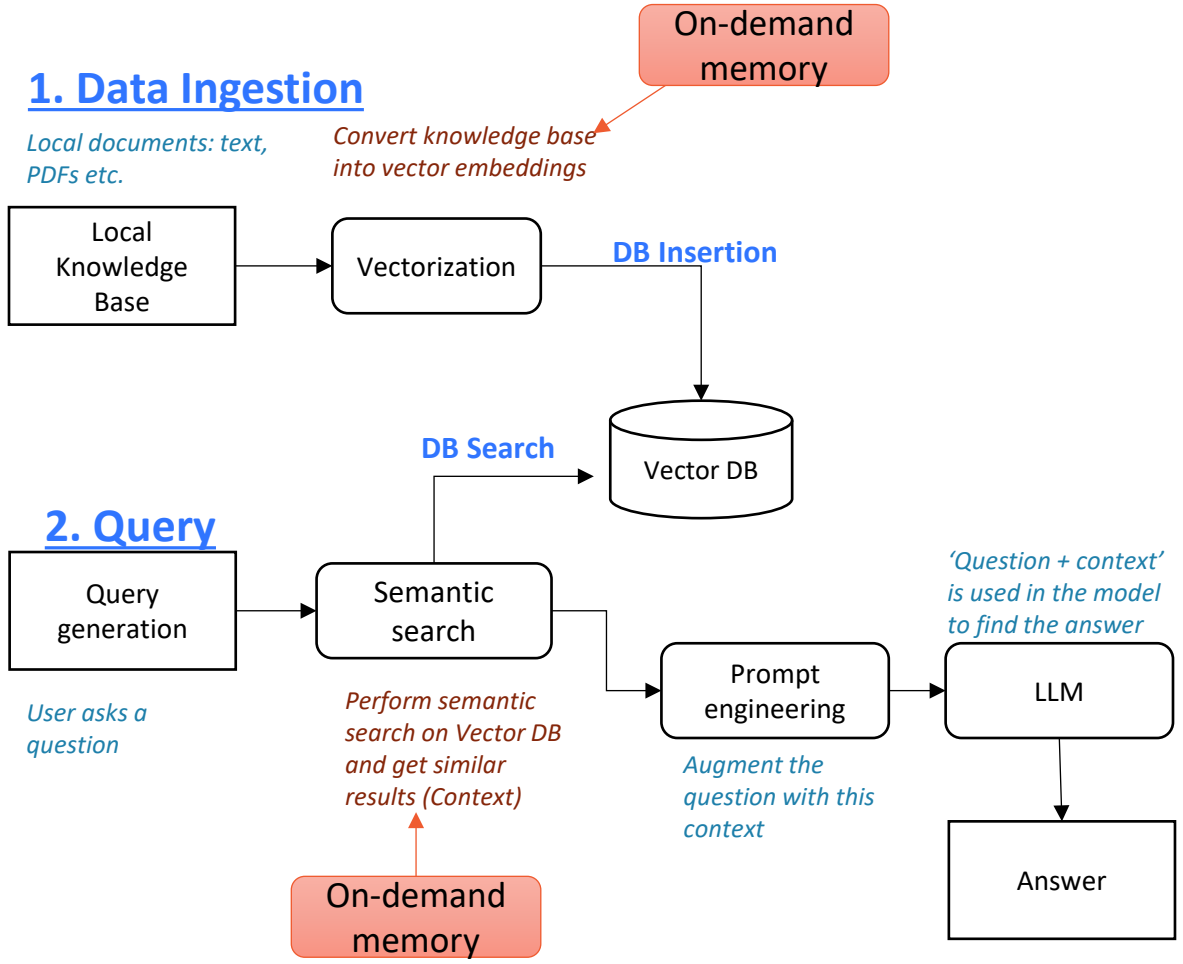
- We focus on the memory usage scenarios
- In particular, Vector DB use cases are interesting
- We observe transient memory demand scenarios
- 1. Data Ingestion phase
  - Huge transient memory demand
  - Large memory required for creation and handling of Vector embeddings
- 2. Query phase
  - Transient memory demands based on workload



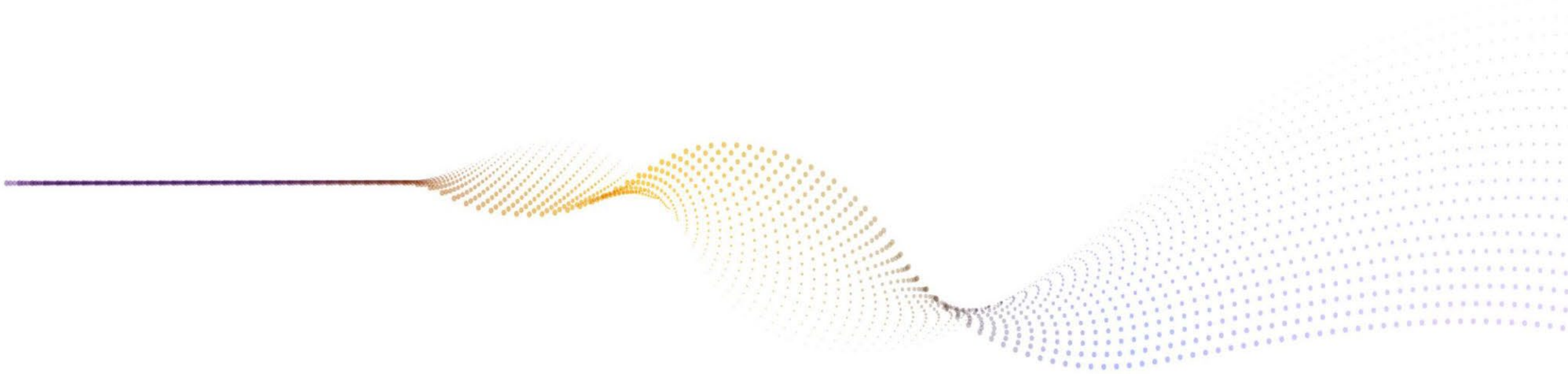
SAMSUNG

Case for using On-demand memory

- On-demand memory can solve transient memory pressures
- Need a solution to temporarily provide memory
- CXL Technology can help here
  - CXL based memory pooling provides temporary memory provisioning



SAMSUNG

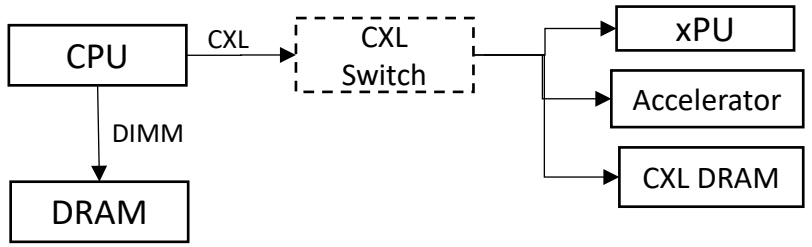


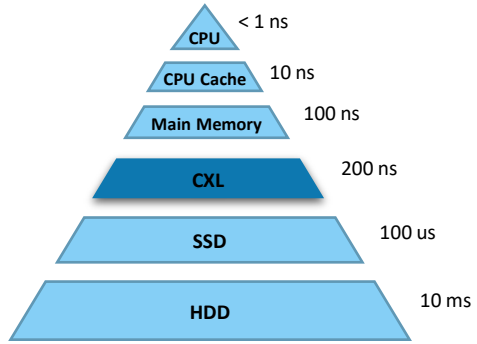
CXL Memory Pooling

Memory provisioning on demand

Primer: CXL Technology

- CXL Technology
  - Open standard interconnect
  - Designed for high-speed, cache-coherent communication between processors, memory, and accelerators
- Memory expansion through CXL
  - It leverages existing PCIe physical layer
    - Makes the adoption easier
- CXL in Memory hierarchy
  - CXL adds a layer in memory hierarchy in terms of access latency and typical capacity
  - General idea is to leverage CXL to expand memory with a latency cost but with capacity benefits





Memory Hierarchy Diagram

Primer: CXL Technology (contd..)

- Three CXL sub protocols: to ease the connection of different classes of devices
  - CXL.io – device enumeration
  - CXL.cache – coherent device memory
  - CXL.mem – memory expansion

- Defines three device types: to help devices to participate with appropriate protocols
  - Type 1: caching devices such as Accelerators and SmartNICs
  - Type 2: GPUs and FPGAs that have memories like DDR and HBM attached to the device.
  - Type 3: memory expansion devices

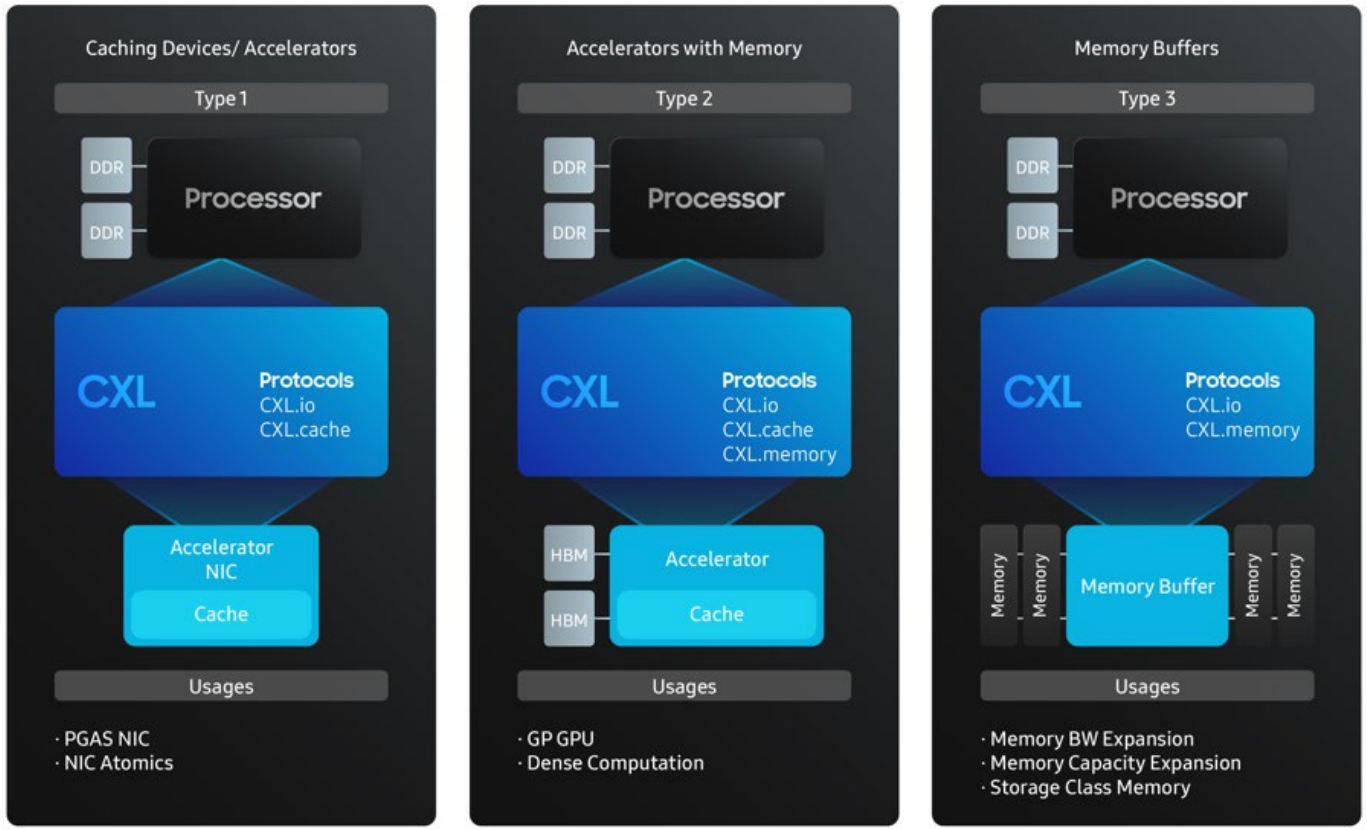
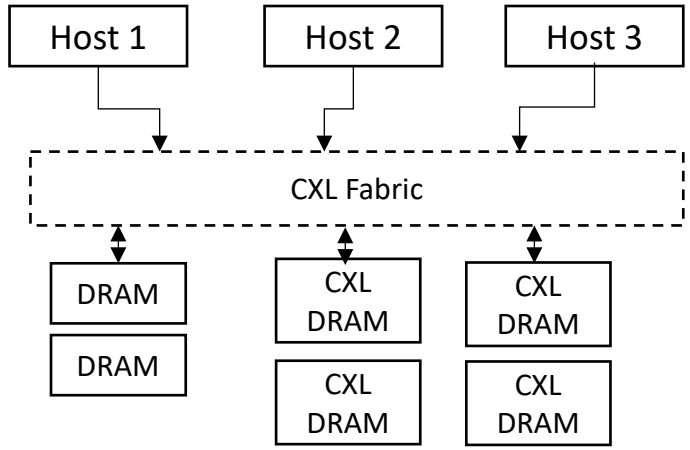


Fig: CXL Usages

CXL Memory Pooling

- Disaggregation: CXL Memory Pools
  - CXL enables disaggregated memory access
  - It enables on-demand memory expansion through the use of common memory pools
- Resource sharing: common pool avoids bottlenecks
  - Memory resources be pooled together and shared across different hosts
- Dynamic capacity: CXL enables memory expansion/shrink
  - CXL supports memory pooling from v2.0
  - CXL supports Dynamic Capacity Devices (DCD) feature from v3.0
    - CXL DCD methodology enables the host to handle the memory expansion/release gracefully.

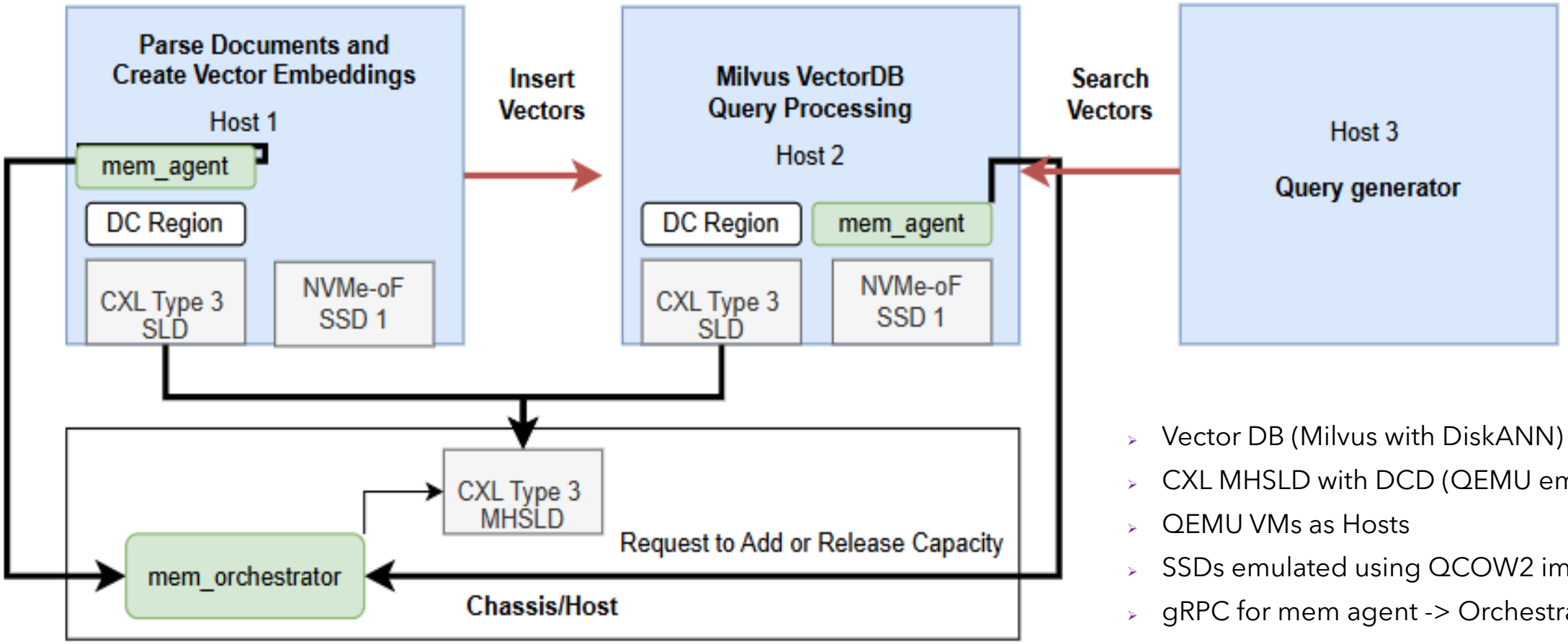
- Takeaway: Memory expansion/release of a pooled memory using CXL DCD would be ideal solution for RAG memory demands.



RAG+CXL Memory Pooling

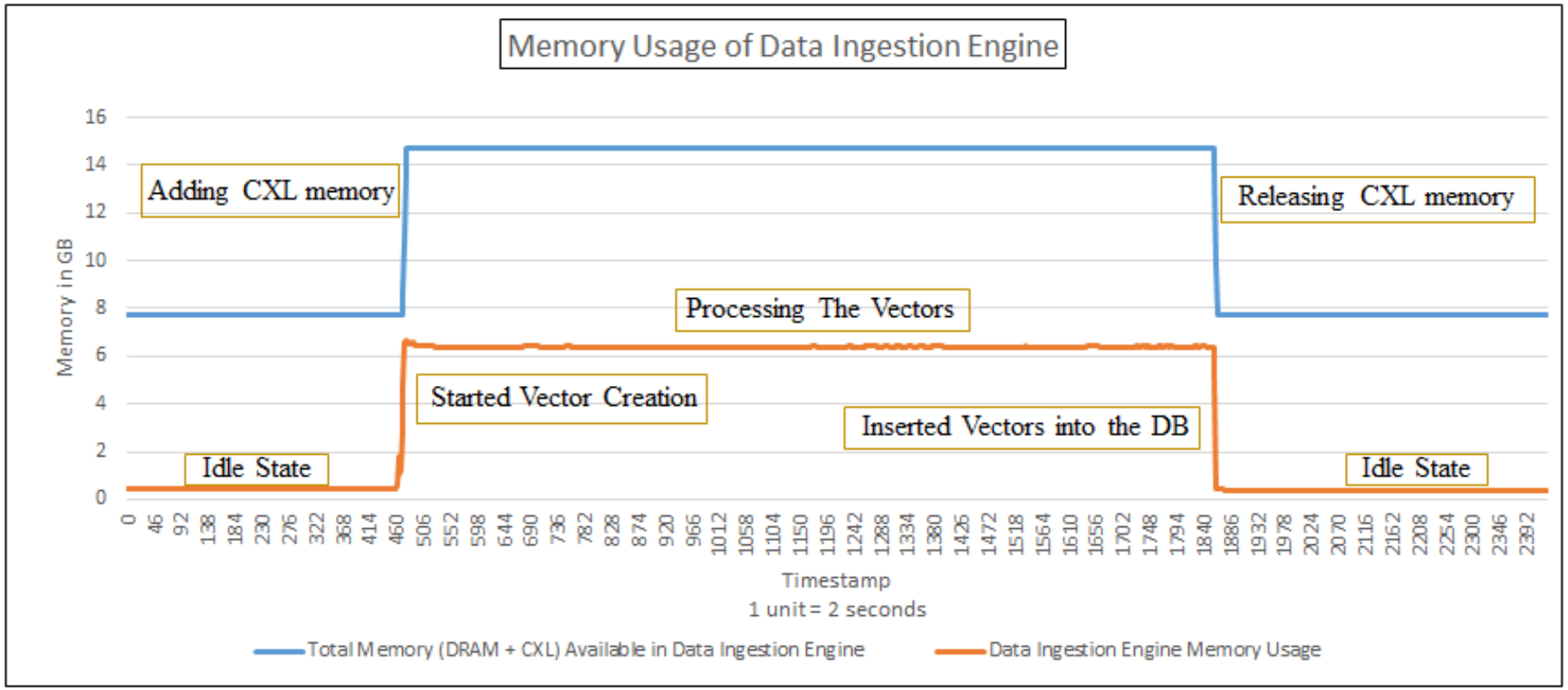
- We focus on VectorDB use cases in RAG pipeline
  - VectorDB insertion and Query stages are considered due to their memory demand scenarios
  - MLPerf storage benchmarking scripts from MLCommons used
- VectorDB
  - Milvus DB is a popular vectorDB
  - DiskANN algorithm is chosen for experiments
    - DiskANN uses DRAM temporarily before flushing the vectors to Disk
    - So, it has more transient memory scenarios than algorithms like HNSW
- CXL 3.0 on QEMU
  - CXL v3.0 supports DCD (Dynamic Capacity Devices) to handle on-demand scenarios
  - CXL 3.0 devices are only slowly arriving though
  - QEMU emulation is an easy way to prove the solution quickly

RAG+CXL Pooling - Experimental Setup



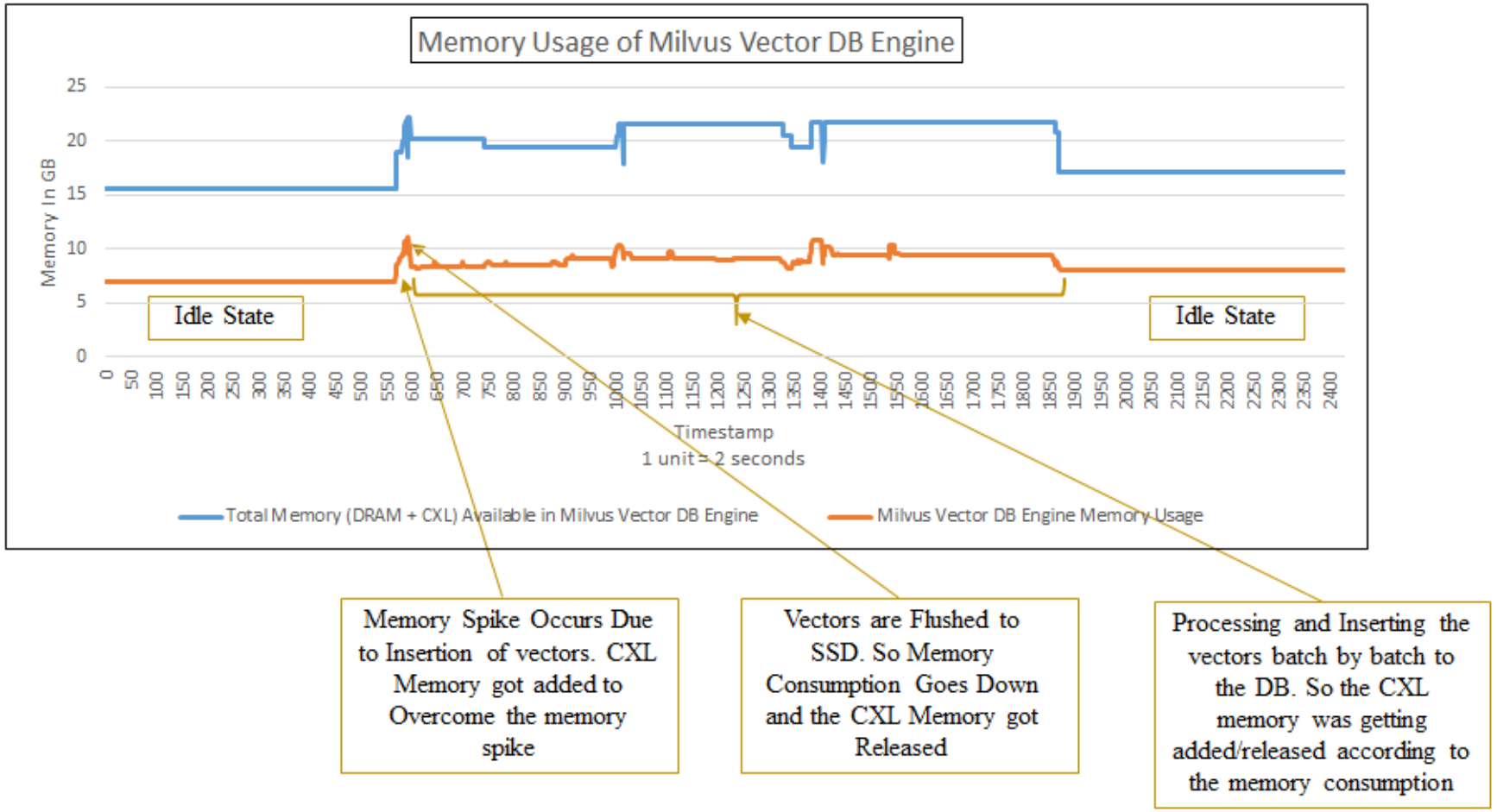
- Vector DB (Milvus with DiskANN)
- CXL MHSLD with DCD (QEMU emulation)
- QEMU VMs as Hosts
- SSDs emulated using QCOW2 images
- gRPC for mem agent -> Orchestrator

RAG+CXL Pooling – Memory usage results: processing engine



- Takeaway: CXL Memory is provisioned and released as per the demand from VectorDB insertion stage

RAG+CXL Pooling – Memory usage results: VectorDB instance



- Takeaway: Memory is provisioned and released as per the instantaneous memory demands

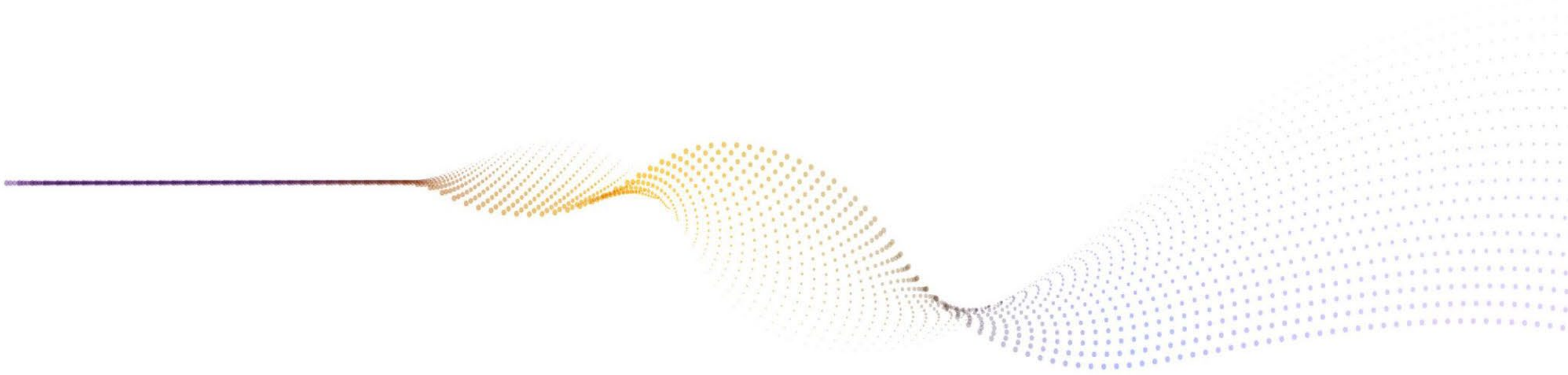
RAG + CXL Pooling – Setup details

1. CXL Memory Pooling Setup
   a. QEMU:
      - Branch: cxl-2025-02-20 (which has MHSLD patch applied already.)
      - URL: https://gitlab.com/jic23/qemu.git
   b. Linux Kernel:
      - Version: 6.12.0, with DCD patches from linux-mm
   c. Ndctl:
      - Branch: dcd-region2-2024-12-10, URL: https://github.com/weiny2/ndctl.git
2. Vector DB: Milvus Standalone Docker: https://milvus.io/docs/v2.0.x/install_standalone-docker.md
3. Vector DB Bench: Developed as a part of ML Commons – Storage (under preview): https://github.com/wvaske/vdb-bench
4. Server Setup Used:
   - Ubuntu version: 22.04.05 LTS | Kernel: 6.1.64 | DRAM: 512GB |CPU: 32 x Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz
5. Configuration of QEMU Nodes with a MHSLD Type 3 device 32 GB size (RAM backed).
   - QEMU 1: 64GiB DRAM, 16 vCPUs connected to head0
   - QEMU 2: 8GiB DRAM, 16 vCPUS connected to head1

TCO Advantages

- 41.67% TCO savings estimated
  - DDR-connected, Local DRAM TCO is 41.67% lower when memory pooling employed
  - Setup (DiskANN, 1M vectors)
    - Non-CXL: 80 GiB local DRAM for insertion host, 16GiB local DRAM for Vector DB instance
    - CXL (64 GiB DCD): 48GiB local DRAM for insertion Host, 8GiB local DRAM for Vector DB
  - Memory pool setup cost is not considered
    - Since it is a shared entity, setup cost can be ignored for this calculation

- Takeaway:
  - CXL memory-pooling enables smart memory management in RAG with TCO benefits

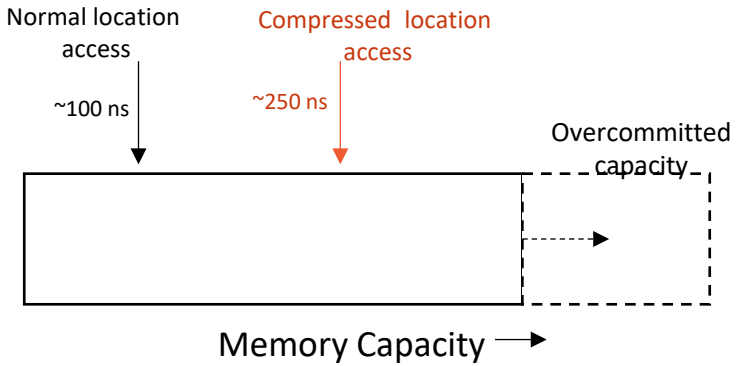


CXL Compressed Memories

Memory expansion in a smart fashion

Primer: CXL Compressed Memories

- Tiered Memory expanders
  - Google and Meta introduced “Hyperscale CXL Tiered Memory Expander” proposal in OCP [1]
  - This proposal talks about CXL memories with inline compression
  - Key point 1: memory over-commitment
    - Effective memory capacity can be increased by thin-provisioning memory
    - Host sees a variable effective capacity in physical address space.
  - Key point 2: variable access latency due to compressed memory locations



[1] https://computeexpresslink.org/wp-content/uploads/2024/10/CXL_Q3-Webinar_Making-Memories-at-HyperScale-with-CXL_FINAL.pdf

Primer: CXL Compressed Memories…

- Tiered Memory expanders - benefits
  - Improves memory TCO
  - Cheaper capacity expansion with slight latency costs
  - Useful for cold memory tiers
  - Why CXL: scalability
    - Allows a pool of cheap memories connected this way, apart from direct CPU connected
- Takeaway:
  - Compressed memories over CXL enables memory over commitment and improves TCO

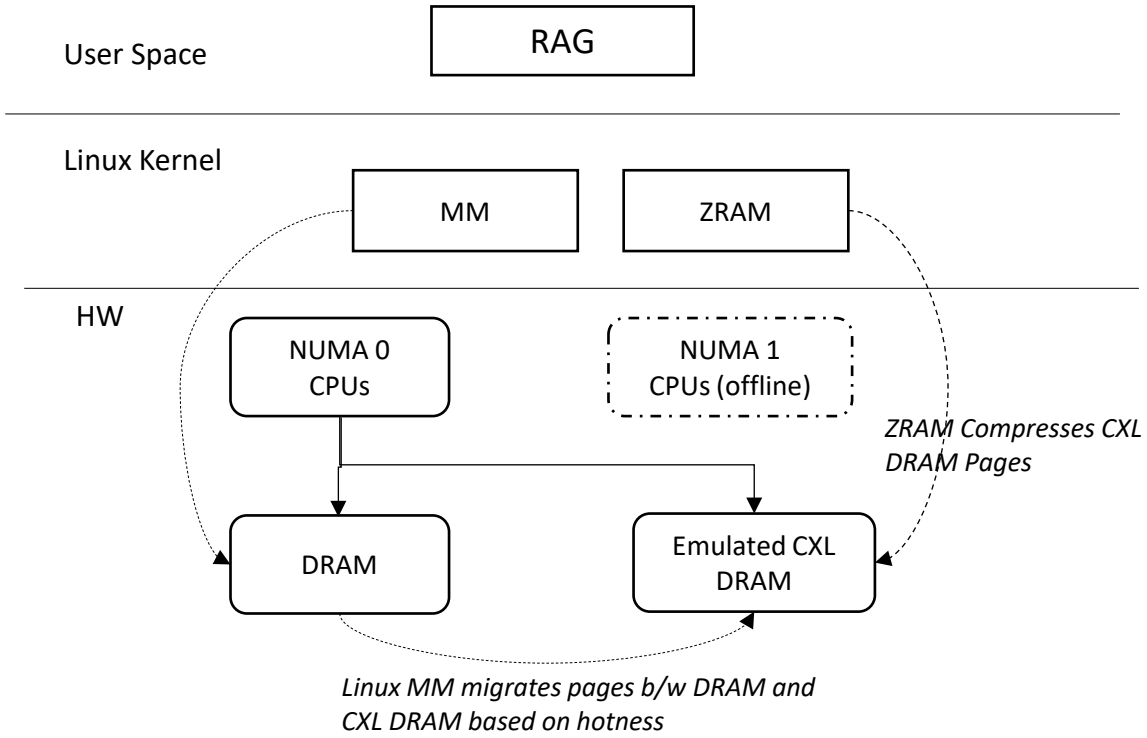
[1] https://computeexpresslink.org/wp-content/uploads/2024/10/CXL_Q3-Webinar_Making-Memories-at-HyperScale-with-CXL_FINAL.pdf

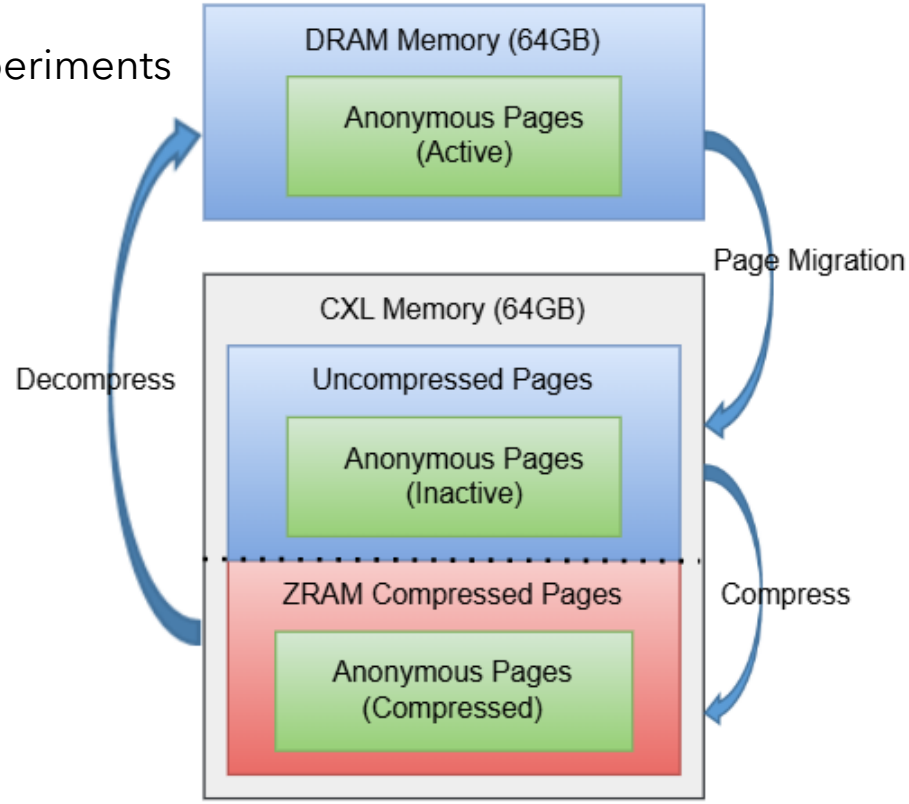
Exploring CXL compression memories for RAG

- Compression improves effective capacity
  - Enables memory expansion
  - More headroom for memory demand scenarios
  - VectorDB insertion and query phases should benefit from increased capacity
  - Insertion phase is not latency sensitive and most of the data is accessed once or only a few times
- Takeaway: Compressed CXL memories could help the memory demand scenarios of RAG

Setup for RAG+CXL Compressed Memory experiments

- CXL Emulation using CPU-less NUMA memory
  - Emulates comparable latency and functionality
- Compression feature emulation using Linux ZRAM
  - ZRAM provides a compressed swap backend on ram
    - Helps to verify the effectiveness of compression for use cases
  - ZRAM is made to work solely on emulated CXL memory for this experiments





SAMSUNG

Experiment 1 - without compression

DRAM : 64GB
CXL : 64GB
Workload : VectorDB bench

```
Every 1.0s: numactl -H

available: 2 nodes (0-1)
node 0 cpus: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46
node 0 size: 65329 MB
node 0 free: 62282 MB
node 1 cpus:
node 1 size: 65524 MB
node 1 free: 64313 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

RAG: Vector generation started

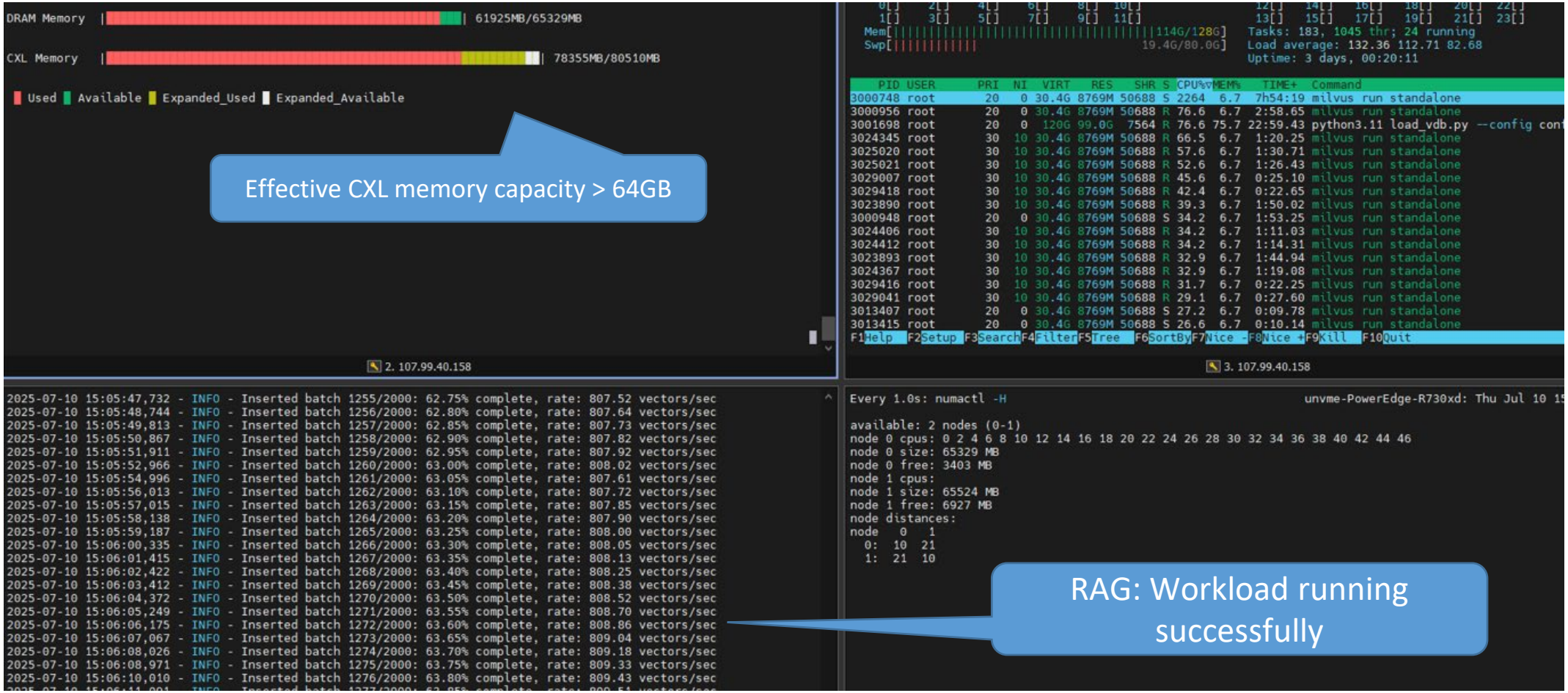
```
root@unvme-PowerEdge-R730xd:/home/unvme/vikash/vector-db/vdb-bench/vdbbench# python3.11 load_vdb.py --config confi
gs/10m.yaml --force
Loaded vdbbench configuration from configs/10m.yaml
2025-07-10 14:12:20,309 - INFO - Connected to Milvus server at 127.0.0.1:19530
2025-07-10 14:12:20,309 - WARNING - FLOAT16 data type not available in this version of pymilvus. Using FLOAT_VECTO
R instead.
2025-07-10 14:12:20,527 - INFO - Dropped existing collection: mlps_10m_10shards_1536dim_uniform
2025-07-10 14:12:20,658 - INFO - Created collection 'mlps_10m_10shards_1536dim_uniform' with 1536 dimensions and 1
0 shards
2025-07-10 14:12:20,658 - INFO - Creating index with parameters: {'index_type': 'HNSW', 'metric_type': 'COSINE', '
params': {'M': 16, 'efConstruction': 200}}
2025-07-10 14:12:21,618 - INFO - Index creation command completed in 0.96 seconds
2025-07-10 14:12:21,619 - INFO - Generating 2000000 vectors with 1536 dimensions using uniform distribution
2025-07-10 14:13:06,302 - INFO - Vector size = 6144000000
Killed
root@unvme-PowerEdge-R730xd:/home/unvme/vikash/vector-db/vdb-bench/vdbbench#
```

```
[257841.356003] oom-kill:constraint=CONSTRAINT_NONE,nodemask=(null),cpuset=user.slice,mems_allowed=0-1,global_oo
m,task_memcg=/user.slice/user-1000.slice/session-2.scope,task=python3.11,pid=2988358,uid=0
[257841.356232] Out of memory: Killed process 2988358 (python3.11) total-vm:130347004kB, anon-rss:126042420kB, f
ile-rss:592kB, shmem-rss:0kB, UID:0 pgtables:250156kB oom_score_adj:0
[257846.405608] oom_reaper: reaped process 2988358 (python3.11), now anon-rss:0kB, file-rss:592kB, shmem-rss:0kB
```

OOM: Process got killed

Takeaway: VectorDB Bench requires more than 128GB of memory (DRAM + CXL)

Experiment 2 - with compression



Takeaway: Compression increases effective capacity and enables VectorDB Bench to run successfully

SAMSUNG

CXL Compression: Takeaways

- Expansion
  - Compression enables capacity expansion in a cost-effective fashion
- Latency
  - Very useful for non latency sensitive scenarios
- Scalability
  - Enables scalable memory expansion using cheaper memories

Summary

Key takeaways

Learning: More efficient RAG Pipelines with CXL

- CXL Memory Pooling
  - Handles transient memory demands
- CXL Compressed memories
  - Memory expansion at reduced cost
- TCO savings
  - Both solutions result in significant memory TCO savings
- Scalable memory expansion solutions
  - Both offers scalable solutions which can be achieved with cheap memories

Thank you for attending!
Questions?

