SNIA DEVELOPER CONFERENCE

SDC 25

By Developers FOR Developers

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

Scaling RAG with NVME

DISKANN's hybrid approach to Vector Database Indexing

www.sniadeveloper.org

Agenda

- Embeddings & Vectors
- Vector DB & Similarity Search
- Indexing Techniques
- Comparing Performance of Vector Indexes
- DISKANN

Vector Database

Purpose-built databases meant to conduct approximate nearest neighbor search.

Search across a large dataset of high-dimensional vectors.

Vectors meant to represent semantics of unstructured data.

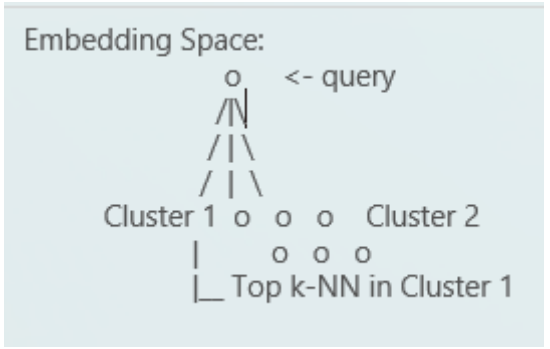
Similarity search requires a data structure known as vector index.

Vectors Key Differences

Aspect	Physical Science	Data Science
Meaning of Dimensions	Spatial (x , y, z)	Abstract Features
Typical dimensionality	2D or 3D	50-1000+ dimensions
Use of vector Math	Geometric	Similarity Search , clustering
Vector Operations	Dot / cross products, vector fields	Cosine , L2 distance
Interpretable Axes?	Yes	Often No

Vector Similarity Search

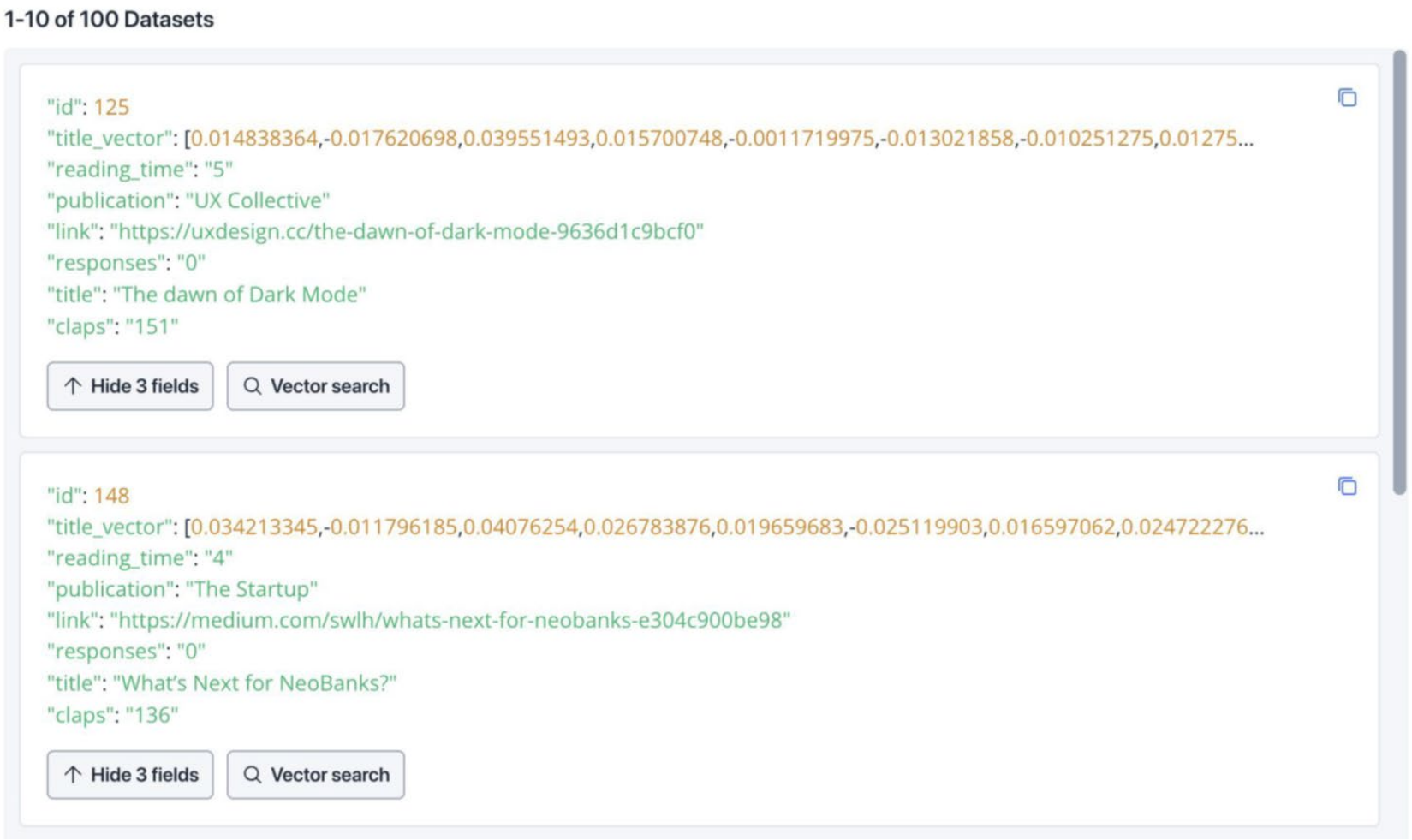
Concept	Role In Vector Search
Embeddings	Represent data in vector space
Similarity	Measure of “closeness” (cosine, L2)
K-NN	Returns top-k closest vectors
Clustering	Partitions space for scalable search
Indexes	Element to speed search. Type – HNSW,IVF,DISKANN,etc…
Recall	Measure of Model Quality



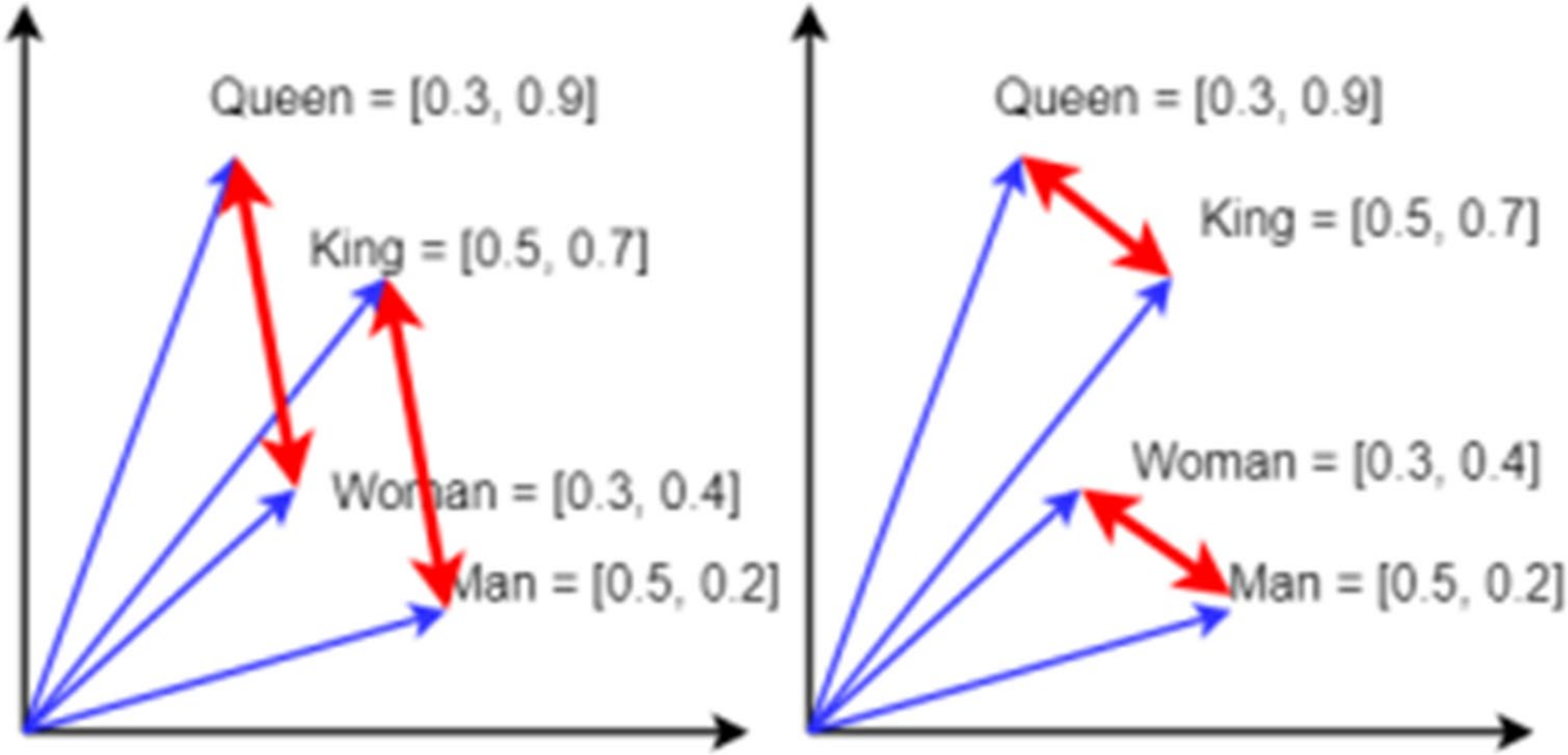
Embeddings

- Mapping from a complex, high-dimensional input into a dense vector.
- Semantic Similarity – Similar inputs are placed together in vector space.
  - It can give the impression of insight / reasoning
- x= raw input / v= embedding vector /d= dimensionality

Embedding – What does it look like?



Similarity as Vector Arithmetic



Indexing Techniques

Brute Force (FLAT)

Tree-Based

Graph-Based

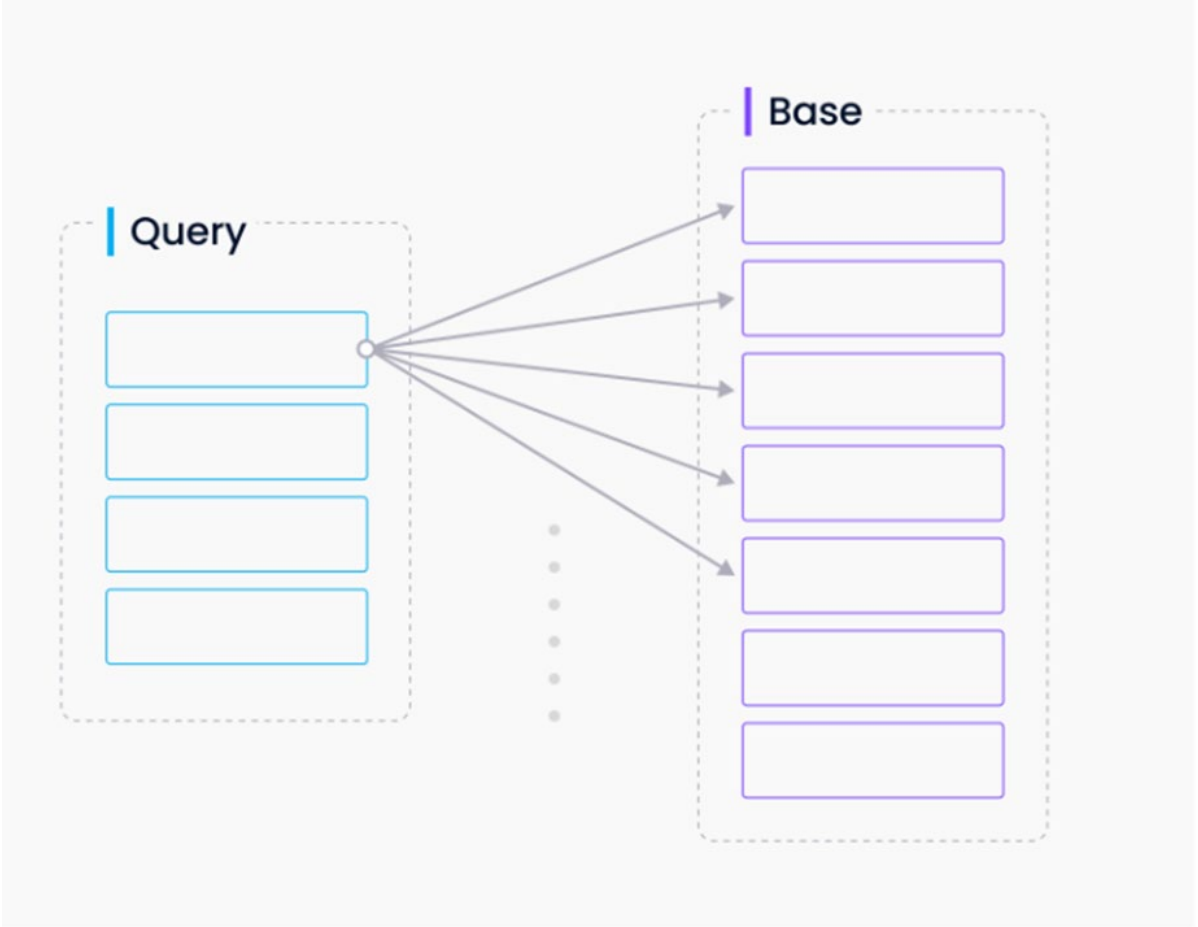
Clustering Based

Quantization Based

Hybrid

FLAT – BRUTE FORCE

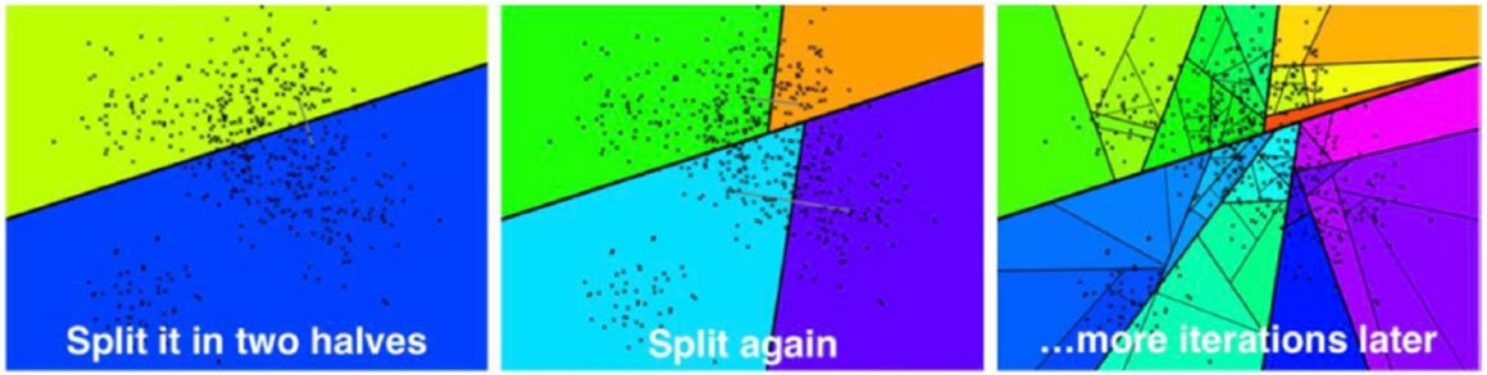
- Good to small datasets
- When exact nearest neighbor is required.
- Basically, an exhaustive matrix multiplication.
- Can quickly become the slowest.
- Can take advantage of high level of parallelism.



Flat visualized

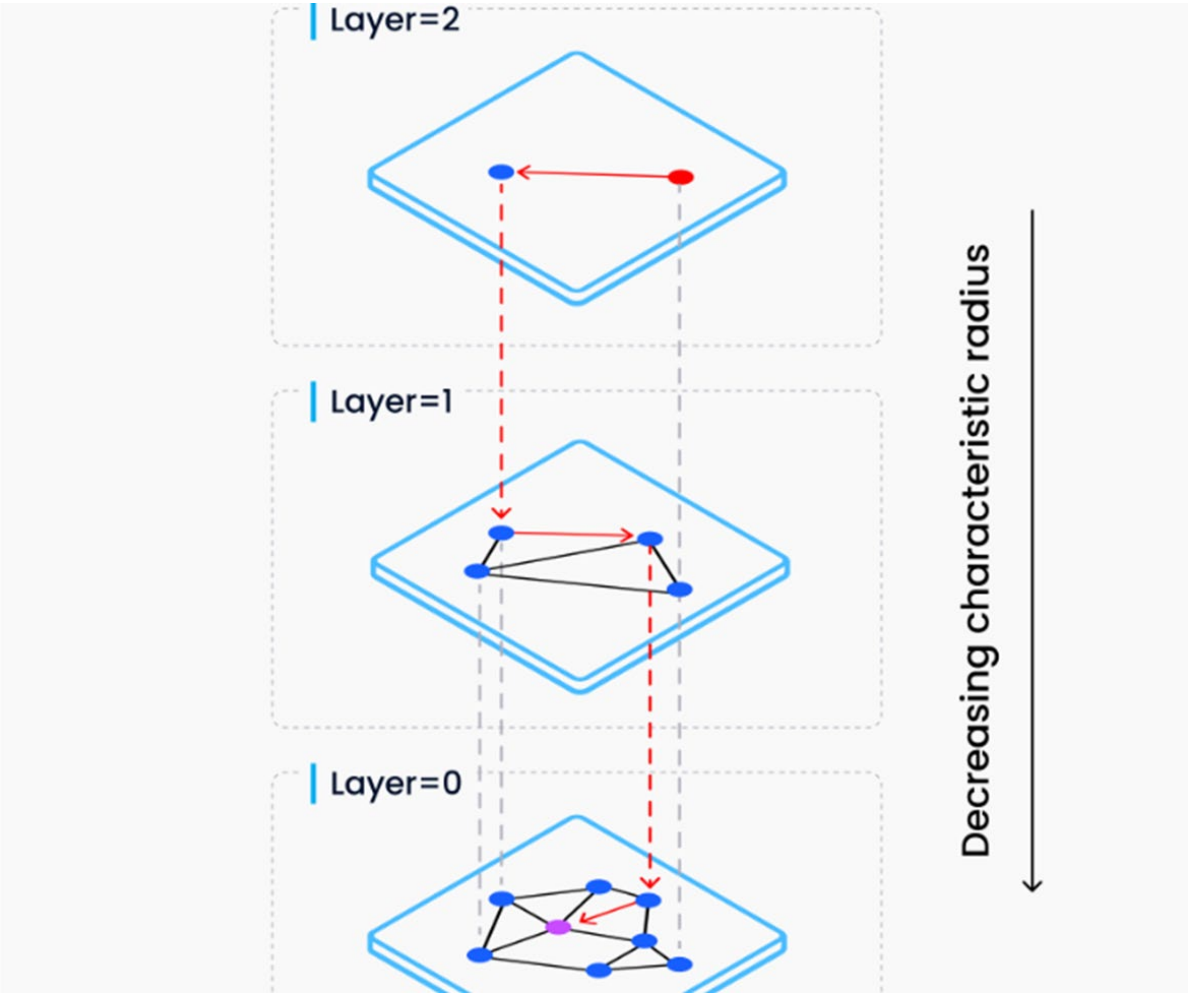
Tree Based Index (ANNOY)

- Uses binary search.
- It partitions the vector space recursively to create a binary tree.
- Each node is split by a hyperplane equidistant from 2 randomly selected child vectors.
- ANN – Approximate Nearest Neighbors

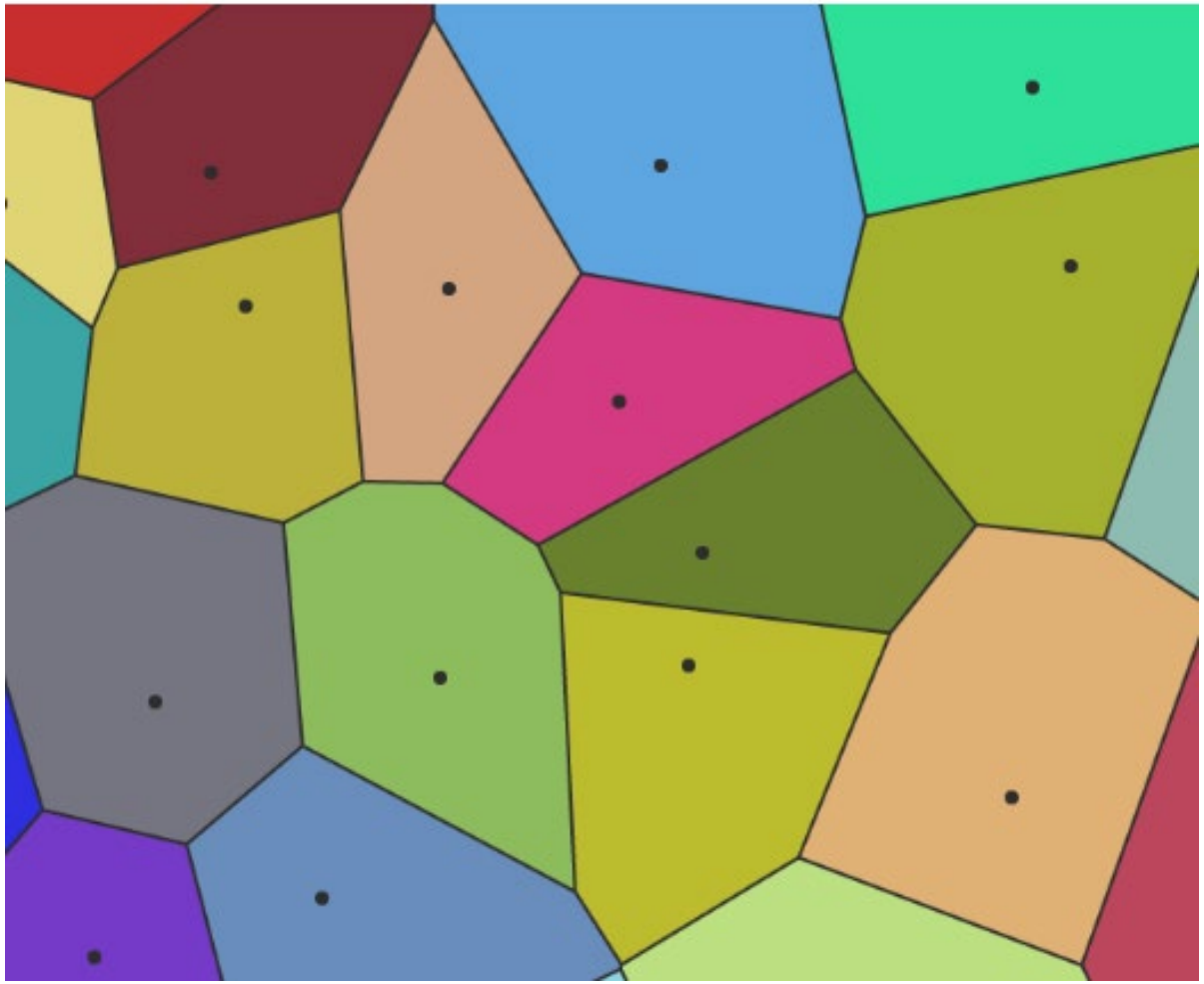


Graph Based

- One of the most popular today.
- Skip List and Navigable Small Graphs (NSG)
- Starts at the top layer and moving downwards until find closest match.



Cluster Based Index (IVF)



- Partition Based indexing strategy.
- Assigns All Database vectors to partition with the closest centroid.
- Cluster centroids are determined using unsupervised clustering (k-means)

Quantization Based

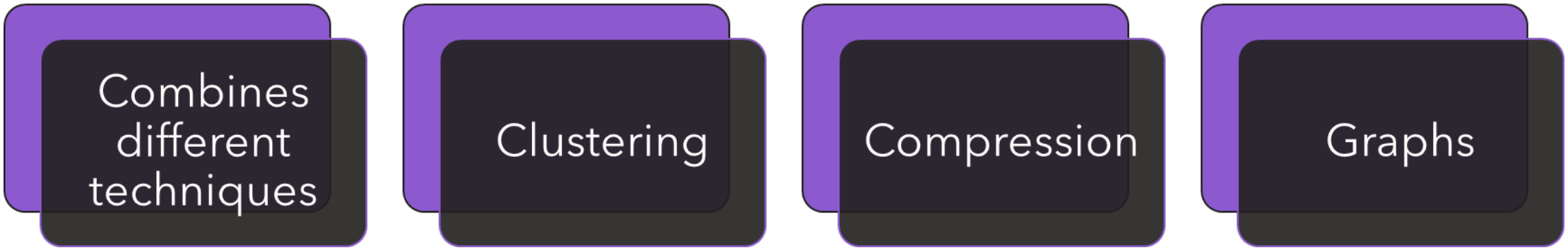
Scalar Quantization (SQ)

- Converts floating-point vectors into integer vectors by dividing each dimension into bins.
- Determine each dimension min and max values.
- Calculate start values and step sizes.
- Perform quantization by subtracting start values and dividing by step sizes.
- Normally uses 8-bit unsigned integers.

Product Quantization (PQ)

- SQ disregards distribution along each vector dimension, potential underutilized bins.
- More powerful alternative that performs both compression and reduction.
- Typically involves splitting vectors, applying k-means across all splits and convert centroid indices.

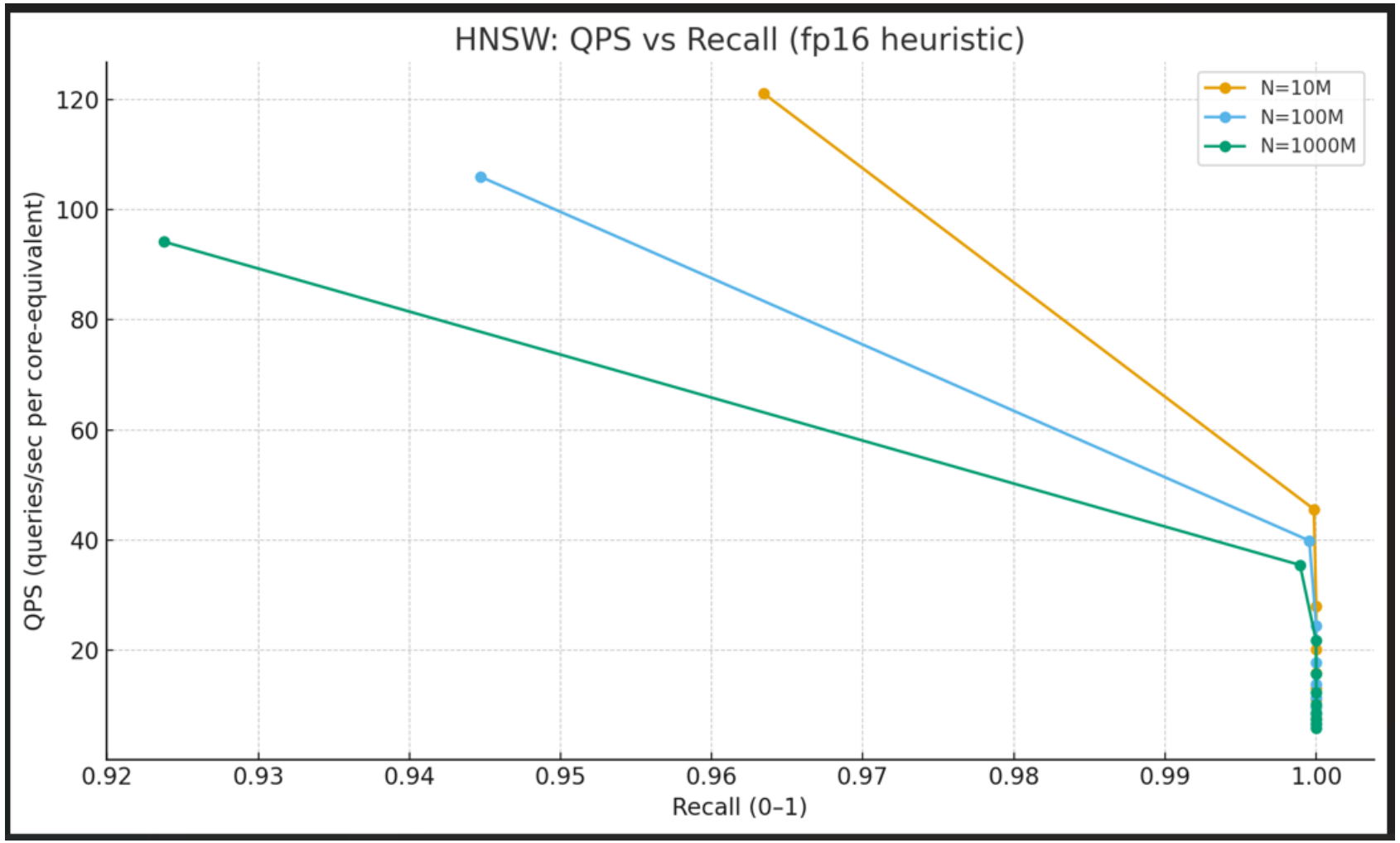
Hybrid (IVF+PQ, HNSW+PQ, DISKANN)



Comparing Performance of Vector Indexes (Search)

- Latency and Throughput can only be compared at similar levels of recall.
- Tend to compare recall within a small set of potential buckets.
  - 85% - 89% , 90% - 94%
- Compare against the model best-case search performance , high recall and throughput plus low latency.
  - The best result is dependent upon the requirements.
  - Try different settings for Index creation and Search.

QPS x Recall – HNSW (heuristic)



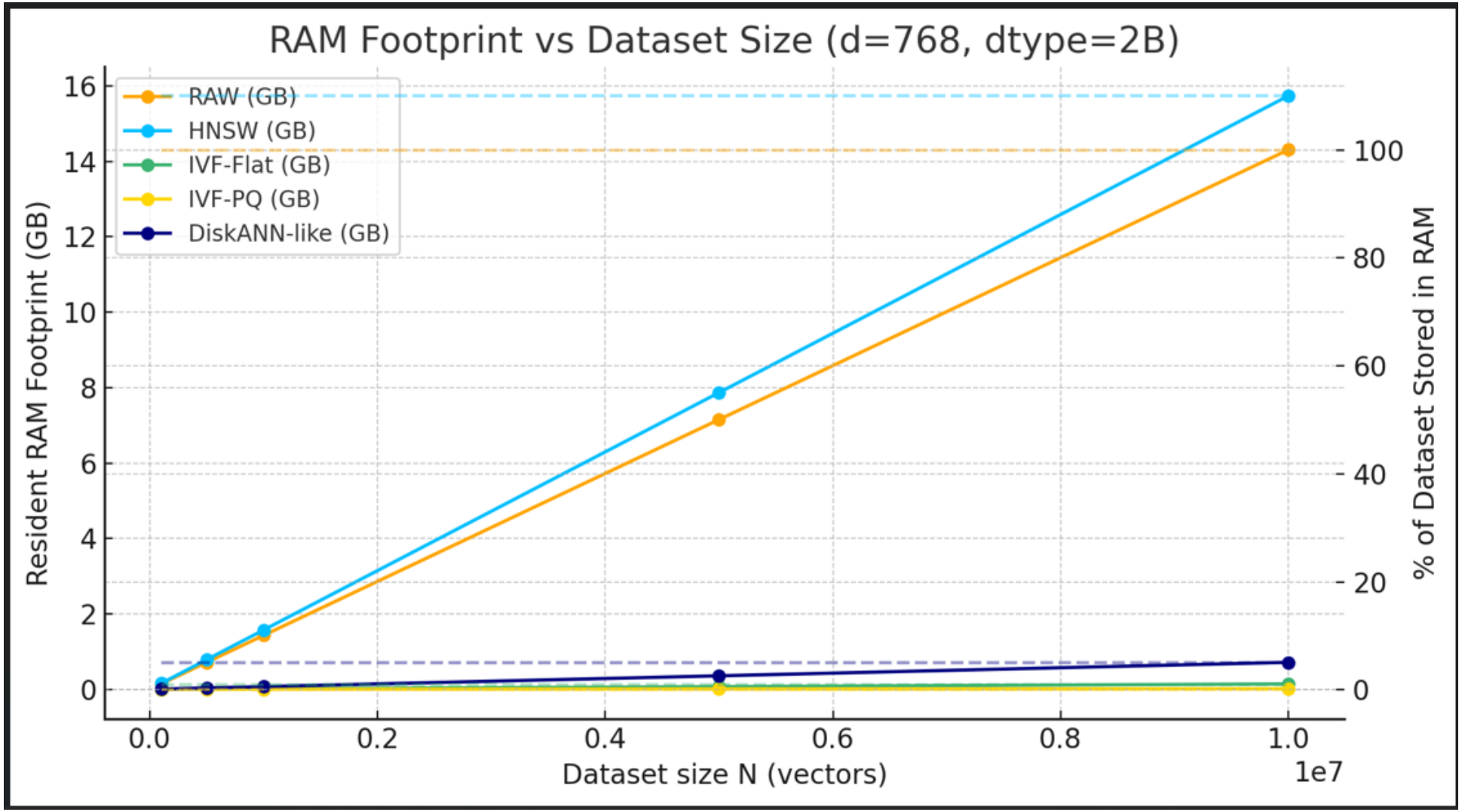
Memory Wall to DISKANN

Current state of the art ANN search algorithms generate indices that MUST be stored in Main Memory.

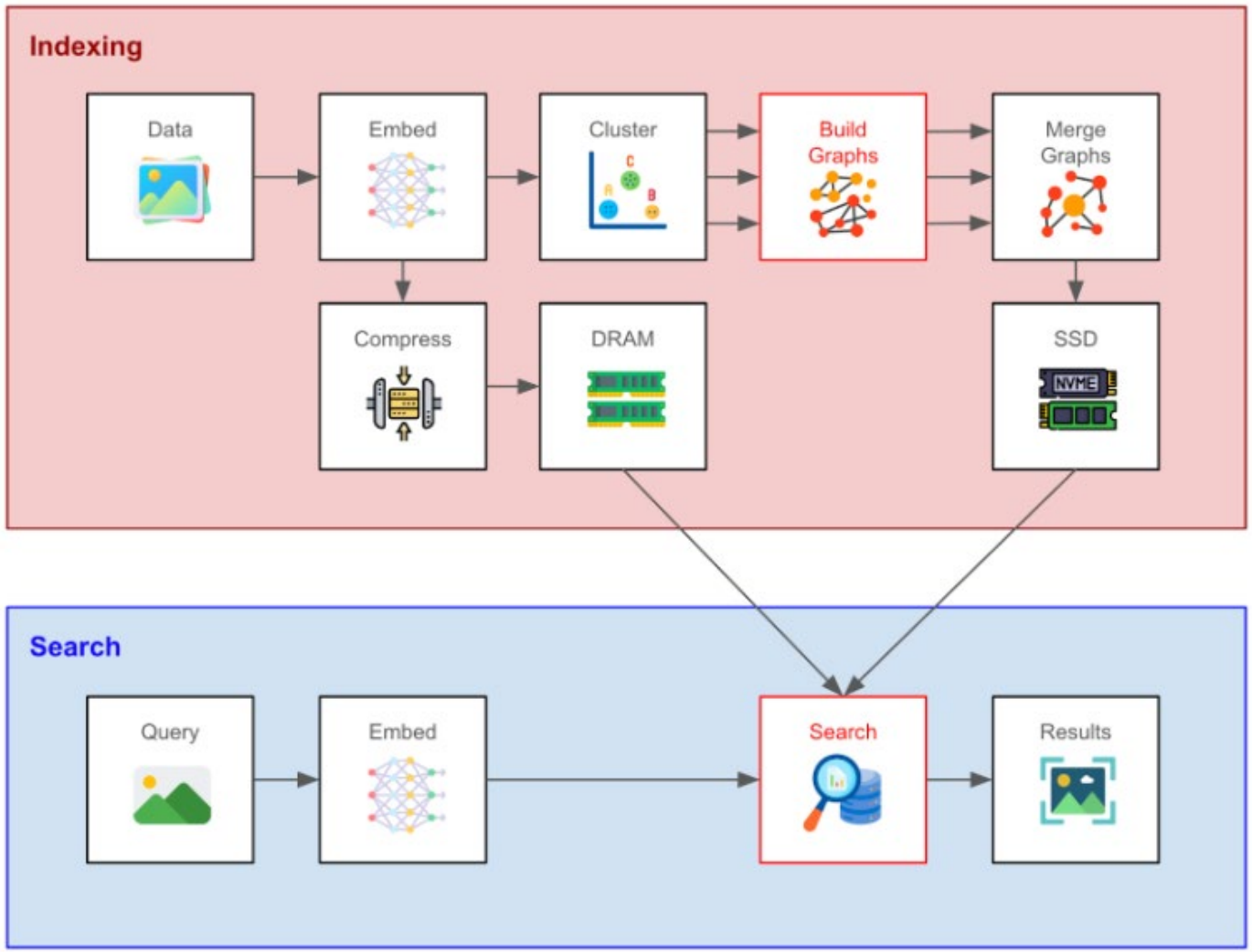
DISKANN – NVME based indices can meet requirements for a large-scale dataset.

- Local Main Memory – 64GB RAM
- 1B data points (vectors) – SIF1B big ann dataset.
- 5000 queries per second < 3ms @95% recall rate.

RAM Footprint



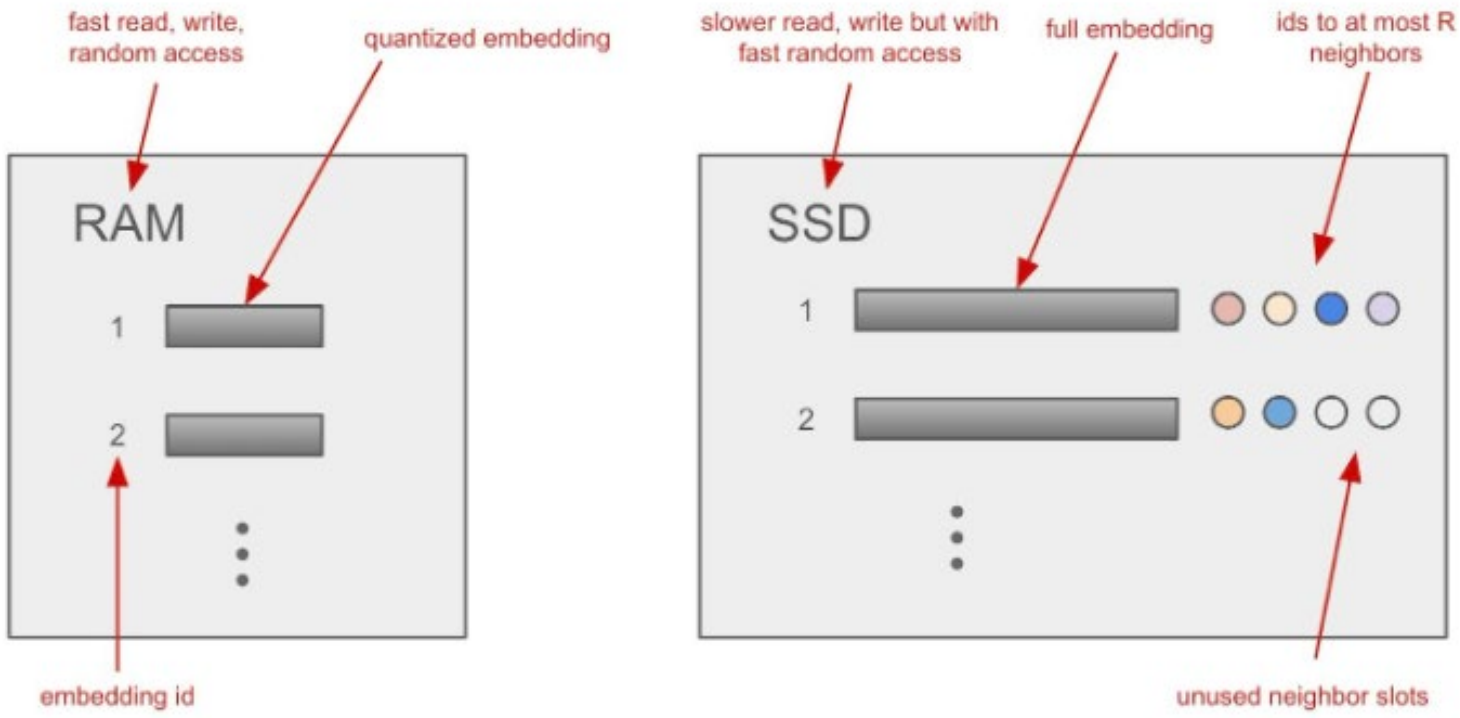
DISKANN Workflow



DISKANN - Architecture

- VAMANA construction algorithm.
- Memory + Disk Hierarchy
  - Memory – Medoid Graph (Compressed representation of points)
  - Disk – Resident Flat Graph (Vamana Graph)
- Optimized Disk Layout
  - Offline reordering
  - Graph connections stored in a sequential layout
- Beam Search instead of Greedy Search
  - Non-hierarchical beam search over a single-layer flat graph

DISKANN – Vector Index DRAM + Block Device

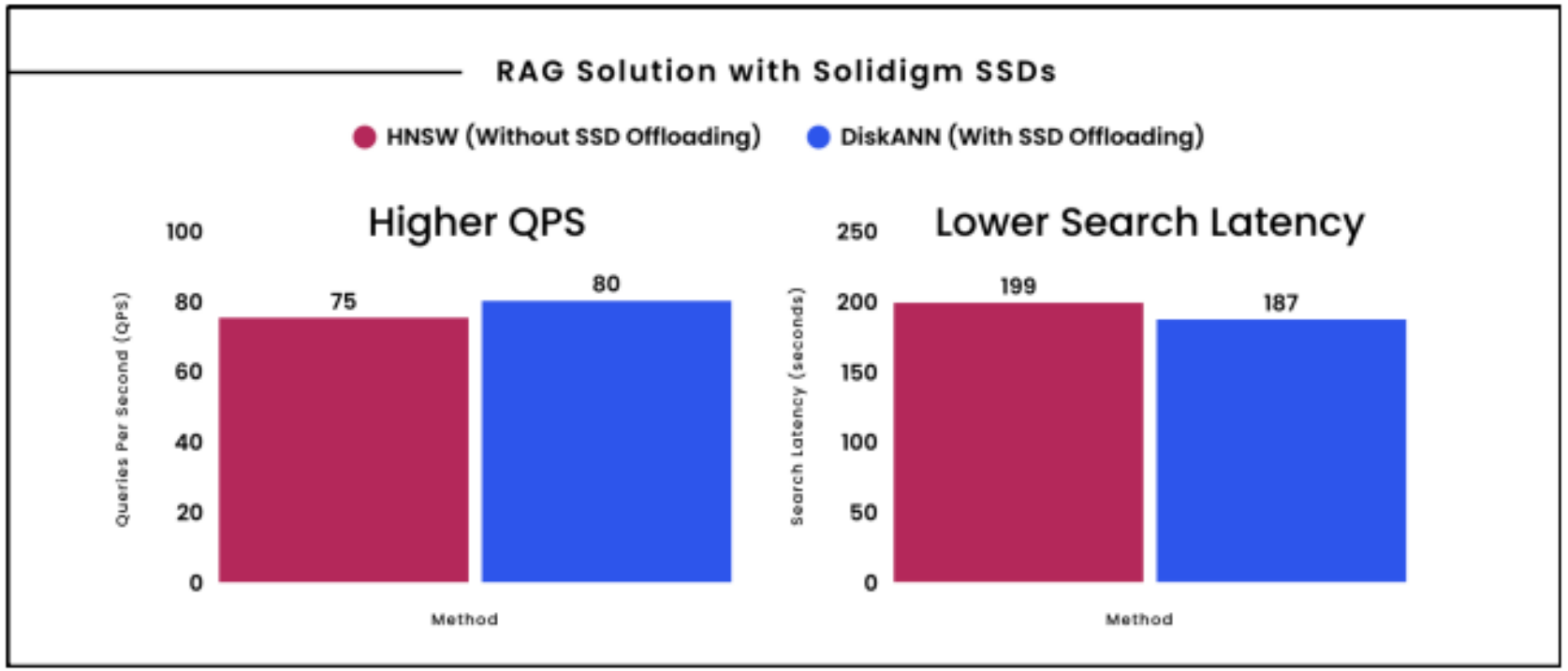


DISKANN - Enhancements

- Regular Search Operation – High Disk IO Operations
  - Fetching the neighborhood information from Disk.
  - Distance calculation to guide the best vertices and neighborhoods to read from the disk
- Fetch neighbors sequentially – Low Disk Operations
  - Fetch a small number of closest points in one operation.
- Cache data associated with a subset of vertices in DRAM.

DISKANN - QPS / Latency @ 97% Recall

- 10M Vectors – 768 dimensions



DISKANN Parameters (Milvus)

Parameter	Description	Value Range	Default Value
MaxDegree	Maximum degree of the Vamana graph. A larger value offers a higher recall rate but increases the size of and time to build the index.	[1,512]	56
SearchListSize	Size of the candidate list. A larger value increases the time spent on building the index but offers a higher recall rate. Set it to a value smaller than MaxDegree unless you need to reduce the index-building time.	[1,int32_max]	100
PQCodeBudgetGBRatio	Size limit on the PQ code. A larger value offers a higher recall rate but increases memory usage.	[0.0,0.25]	0.125
SearchCacheBudgetGBRatio	Ratio of cached node numbers to raw data. A larger value improves index-building performance with increased memory usage.	[0.0,0.3]	0.10
BeamWidthRatio	Ratio between the maximum number of IO requests per search iteration and CPU number.	[1,max(128/CPU number,16]	4.0

Index Memory Footprint (MILVUS Sizing)

768 Dimensions	DISKANN (GB)	HNSW(GB)
1M Vectors	18	18
10M	29	74
100M	332	780
1B	1638	4362

MILVUS Sizing Tool – 1B Vector / 768 Dimension

HNSW

- Estimated Monthly Cost – US$ 23K

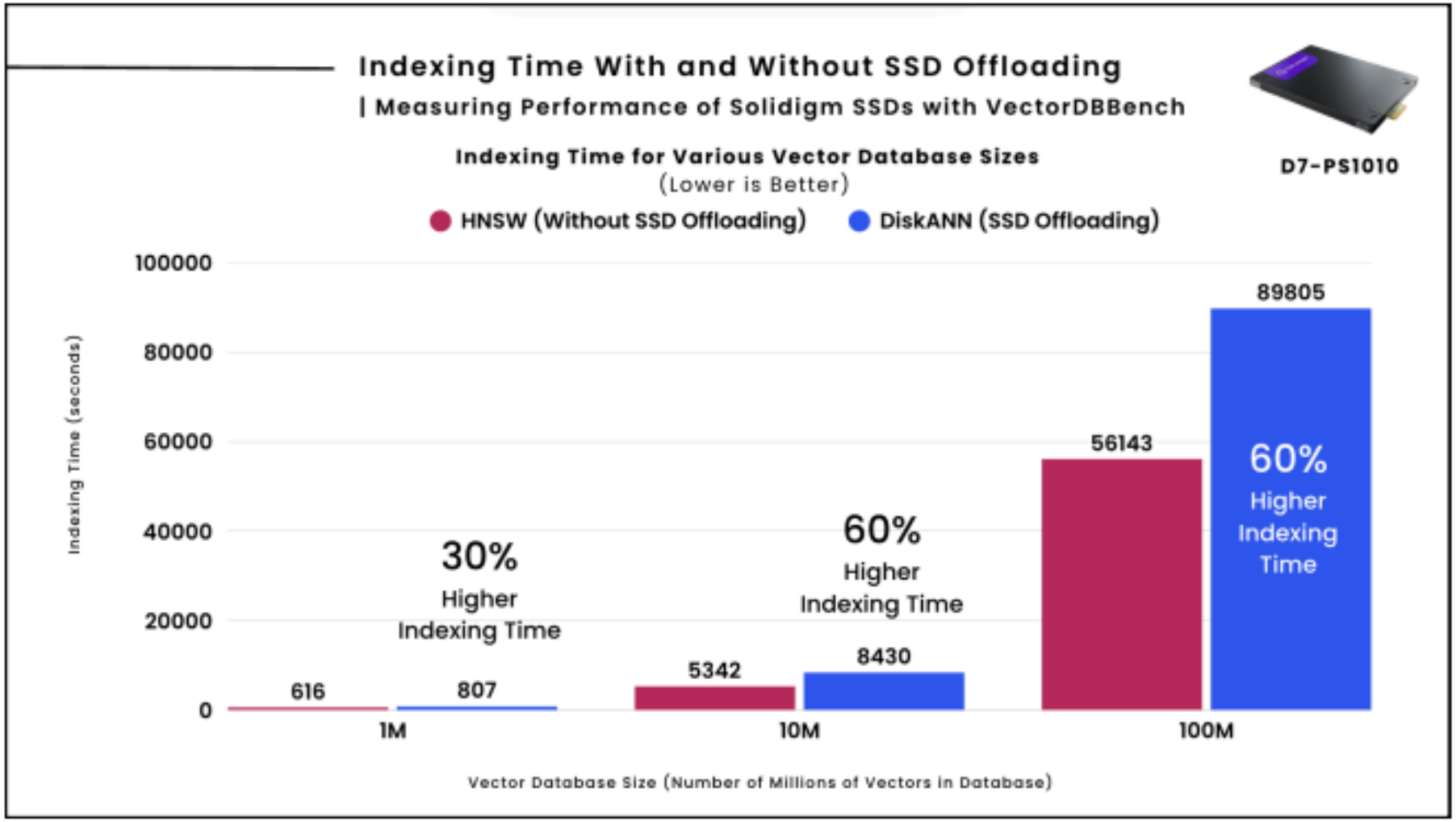
Cores	Memory	Storage	Local Disk
1100	4.26 TB	33.87 TB	0

DISKANN

- Estimated Monthly Cost – US$ 9.3K

Cores	Memory	Storage	Local Disk
340	1.26 TB	23.21 TB	3TB

DISKANN – Building Time Compare

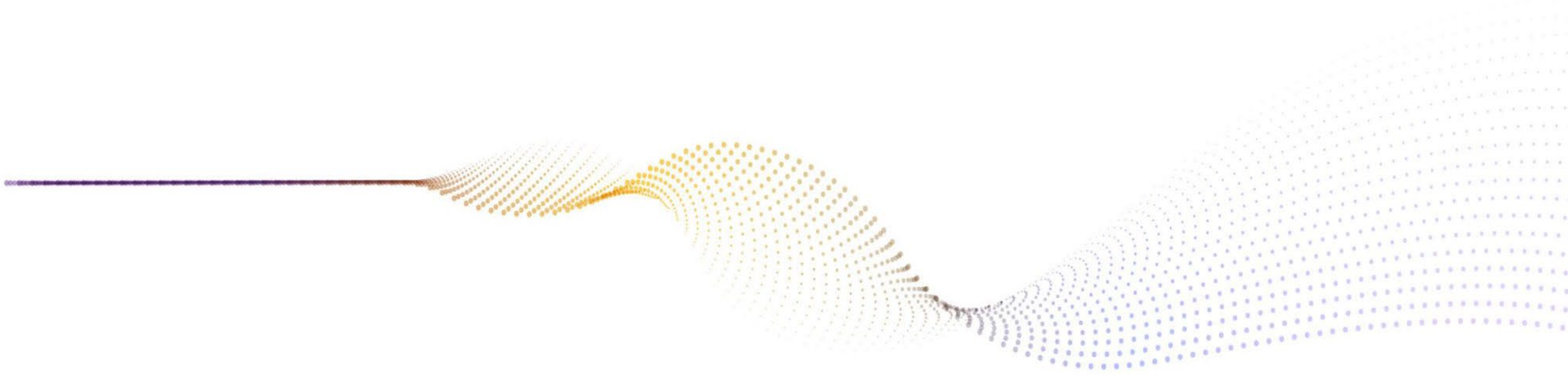


MLPERF Storage VectorDB Benchmark

- Initiative to add VectorDB benchmarking into MLPERF Storage suite of tests.
  - Based on VectorDBBench.
    - Currently only supporting MILVUS DB
  - https://github.com/mlcommons/storage/blob/main/mlpstorage/benchmarks/vectordbbench.py

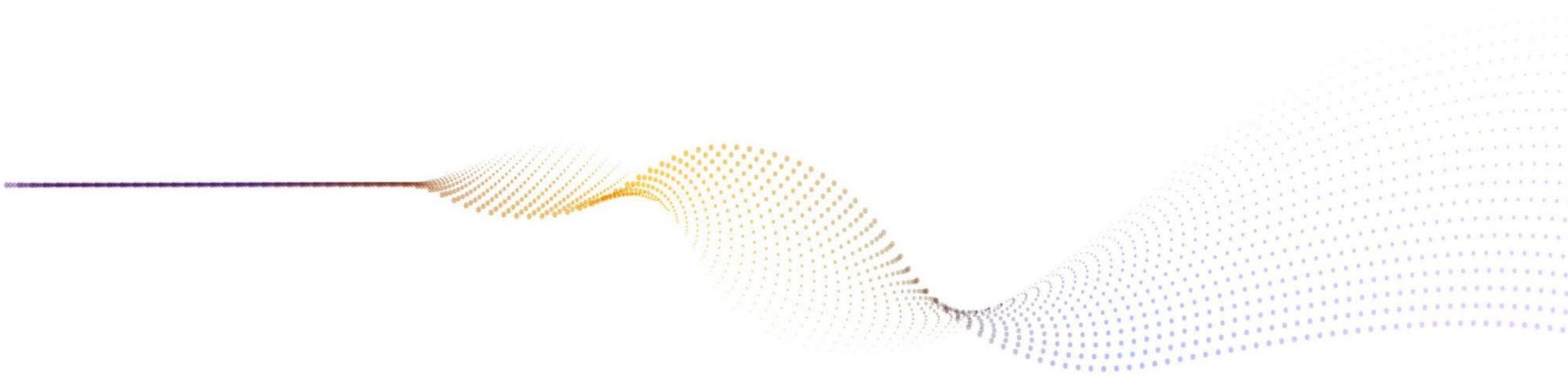
Call To Action

- Realtime updates needed → FreshDiskANN
- Indexing Build Time → CuVS Indexing using GPU
- Support / Integration to larger audience.
  - VectorDBs and RDMS



Thank you for attending!

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



BACKUP

