CS4221
Modern Databases II.
Vector Databases

Yao LU

2024 Semester 2

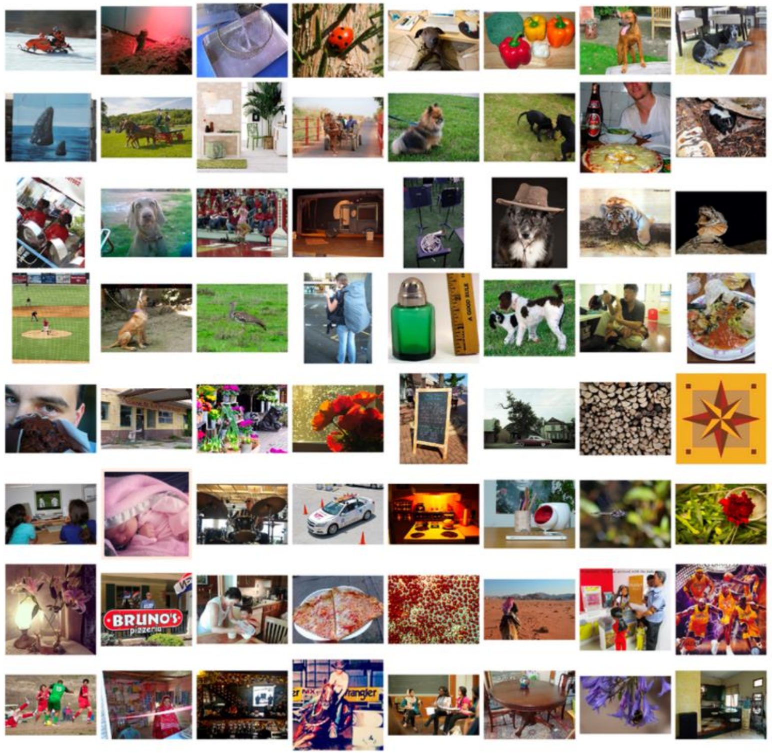
National University of Singapore
School of Computing

Recent vector databases

Pinecone
Milvus
Faiss
chroma
qdrant
Weaviate
vespa
ClickHouse
ORACLE
AlloyDB
Azure SQL
AnalyticDB
SingleStore
MongoDB.
Atlas
PostgreSQL
databricks
redis
Alipay
elasticsearch
OpenSearch
cassandra

Motivation 1: vector embedding

Unstructured Data



Vectors

0.03, 0.12, 0.01, ...

0.01, 0.01, 0.02, ...

0.02, 0.02, 0.03, ...

0.04, 0.11, 0.05, ...

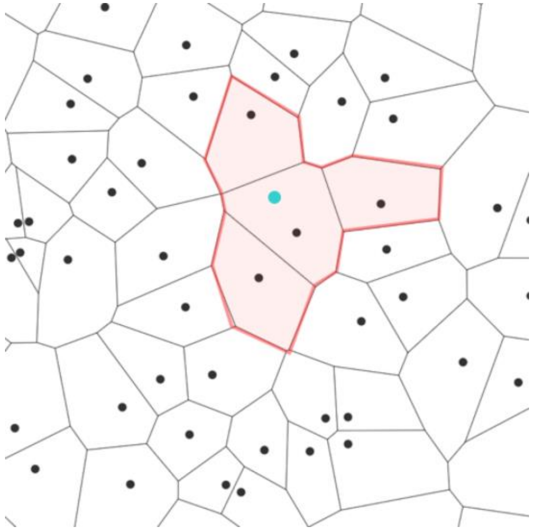
⋮

0.11, 0.13, 0.01, ...

0.06, 0.01, 0.02, ...

0.02, 0.02, 0.03, ...

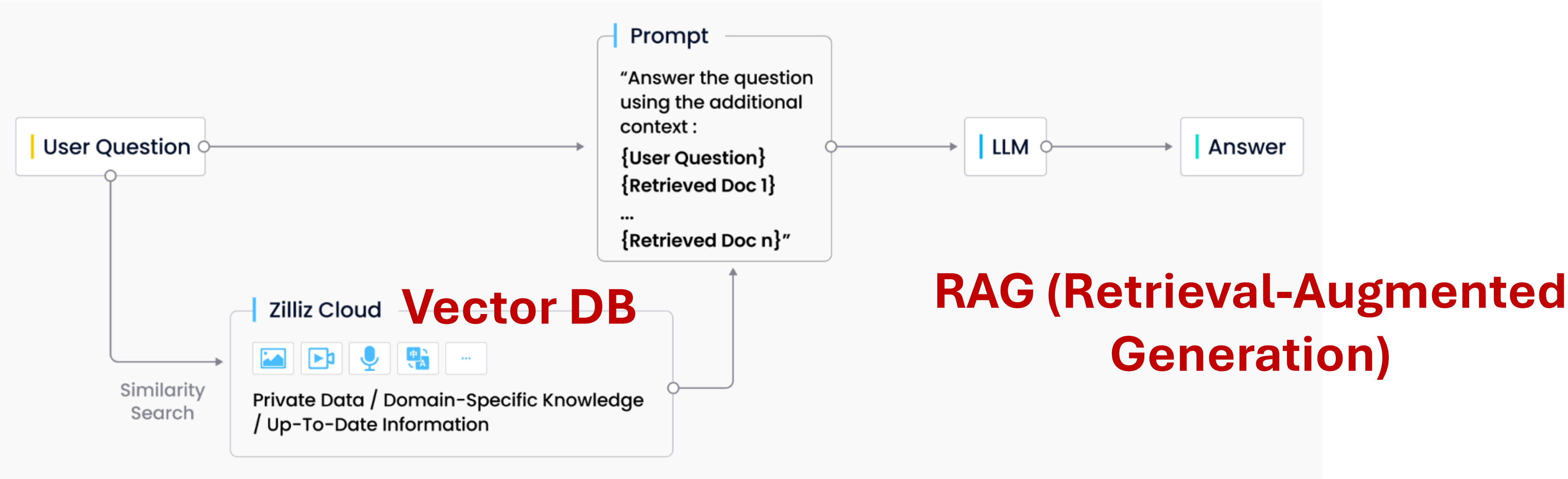
Analytics in Vector Space



Known as "vector embedding" (due to deep learning)

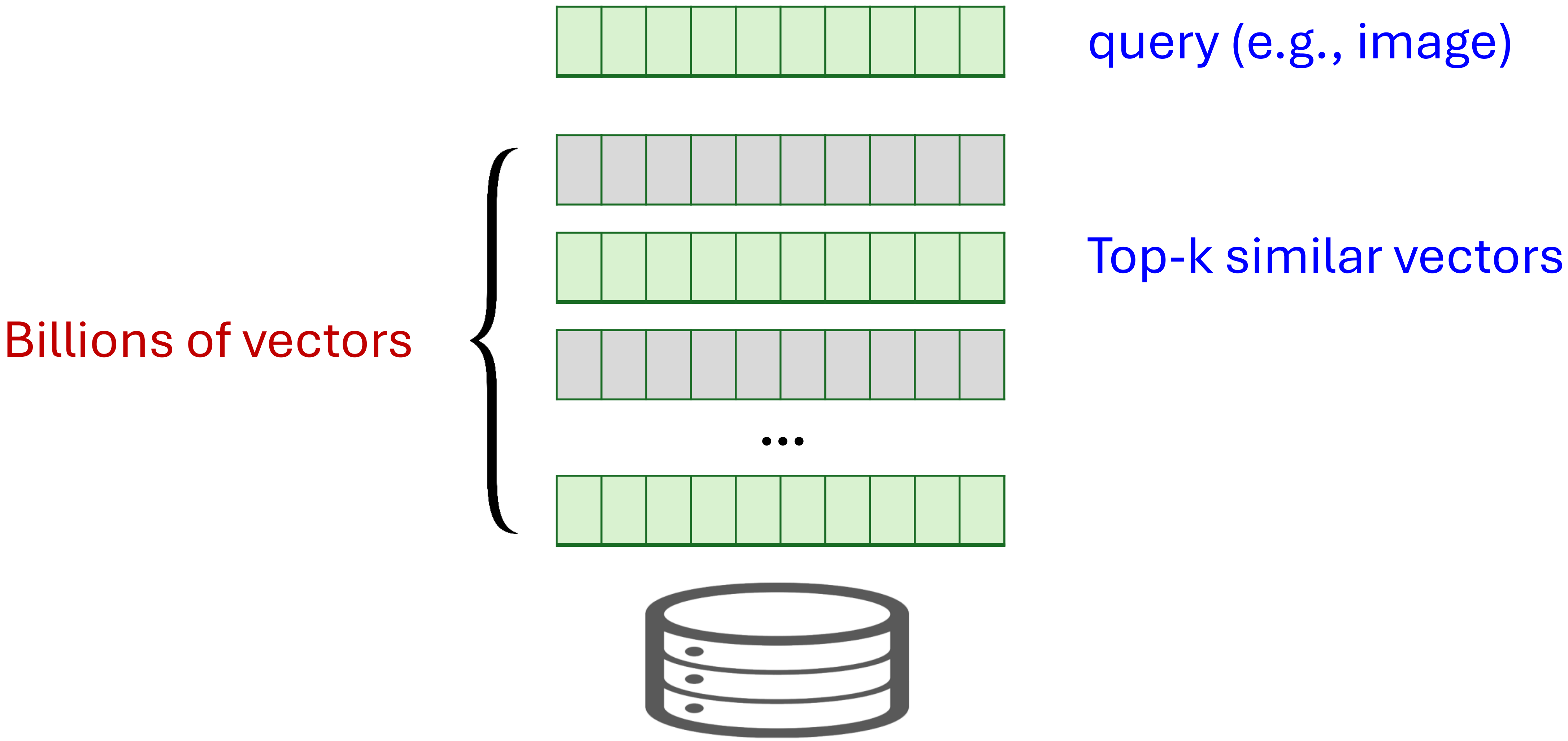
Motivation 2: large language models

- Vector DBs & RAGs address many critical limitations of LLMs
  - Hallucination: incorrect or fabricated answer
  - Lacking domain-specific knowledge
  - Up-to-date information



https://zilliz.com/use-cases/llm-retrieval-augmented-generation

Key operator in vector DBs: vector similarity search



Evolution of vector data(base)

1999
(Content-based information retrieval)

2013
(Embedding)

2023
(LLM)

Similarity Search in High Dimensions via Hashing

Aristides Gionis * Piotr Indyk† Rajeev Motwani‡
Department of Computer Science
Stanford University
Stanford, CA 94305
{gionis,indyk,rajeev}@cs.stanford.edu

Locality-sensitive hash

Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Retrieval OpenAI

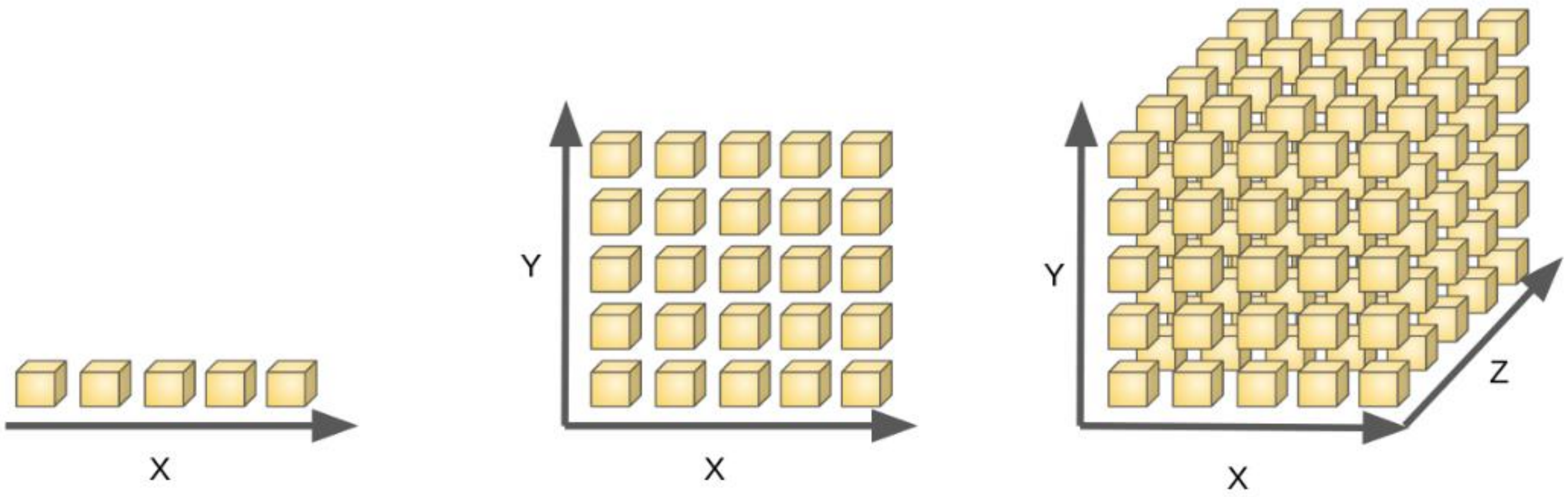
The open-source retrieval plugin enables ChatGPT to access personal or organizational information sources (with permission). It allows users to obtain the most relevant document snippets from their data sources, such as files, notes, emails or public documentation, by asking questions or expressing needs in natural language.

As an open-source and self-hosted solution, developers can deploy their own version of the plugin and register it with ChatGPT. The plugin leverages OpenAI embeddings and allows developers to choose a vector database (Milvus, Pinecone, Qdrant, Redis, Weaviate or Zilliz) for indexing and searching documents. Information sources can be synchronized with the database using webhooks.

Why are vector DBs challenging?

- Easy to get started, but very challenging to achieve high performance, accuracy, and efficiency
- Three unique properties that contribute to the challenges of vector DBs
  - Property P1: Curse of Dimensionality
  - Property P2: Approximation
  - Property P3: Advanced Vector Data Analytics

P1: Curse of dimensionality

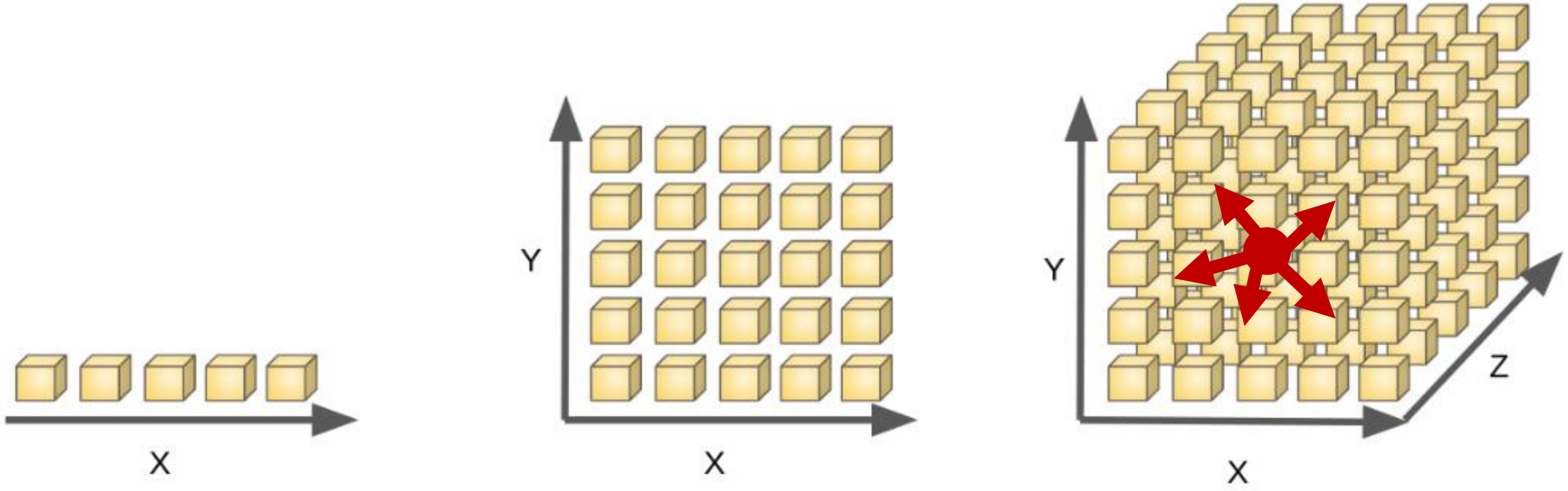


All techniques fail on high-d space (for exact answers)
➔ approximate answer!

All vector DBs return approximate answer

https://medium.com/@soumiksanku08/curse-of-dimensionality-293d0d16fe2a

P1: Curse of dimensionality



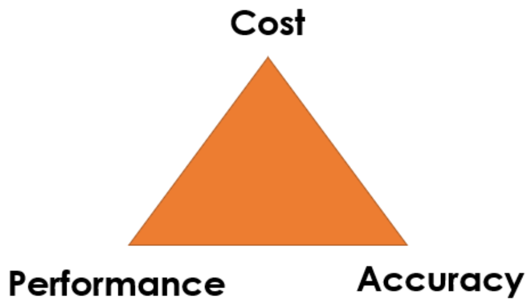
Querying in high-dimensional space
➔ no locality ➔ hard to leverage tiered storage

Almost all vector DBs only use DRAM ➔ too expensive (in the cloud)

https://medium.com/@soumiksanku08/curse-of-dimensionality-293d0d16fe2a

P2: Approximation

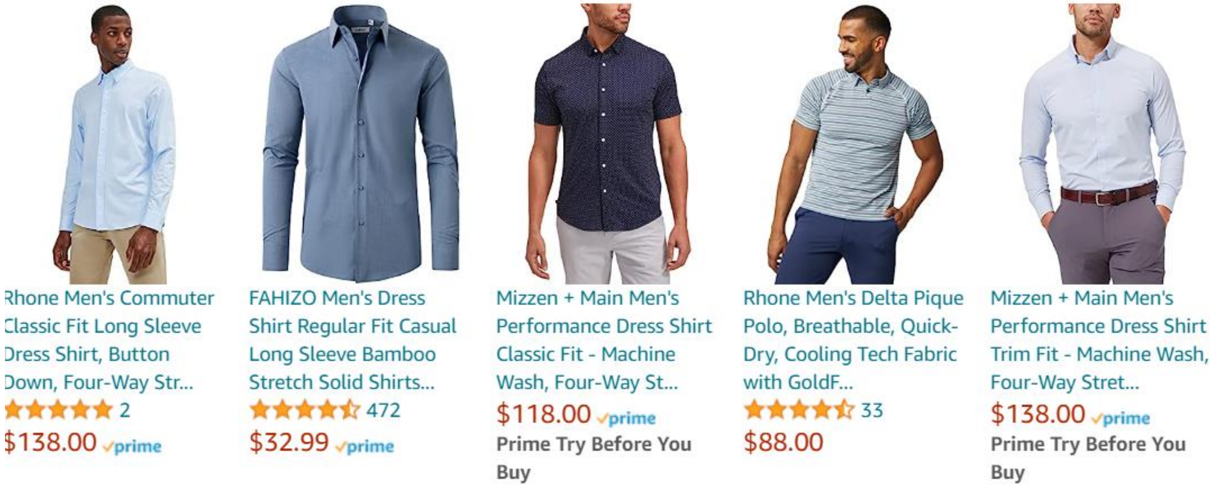
- Approximation introduces a new design tradeoff (in addition to performance and cost)
  - Complicates system design
  - Different from traditional databases
  - Approximate query processing is still not the mainstream
- Many vector indexes are developed with different tradeoffs
- How to make the right tradeoff between CAP?
  - Existing databases rely on users' manual selection



P2: Approximation

- Need to consider approximation from the ground-up
  - Index type selection
  - Index parameter tuning
  - Caching
    - If there’s a cache miss, do you want to return current results or go to disk to compute accurate answer?
  - Compression
  - Consistency
  - Visibility
  - …

P3: Advanced query processing



Finding the T-shirts similar to a given image vector that also cost less than $100 and have text descriptions containing specific keywords

- Query is more than pure vector search
- Can contain filters (attribute filters / range filters), and other non-vector data (e.g., relational / document / graph / spatial data)
- Necessary to support advanced RAGs

Outline

- Introduction
- **Main-memory vector index**
- Disk-based vector index
- Generalized vector DBs
- Specialized vector DBs

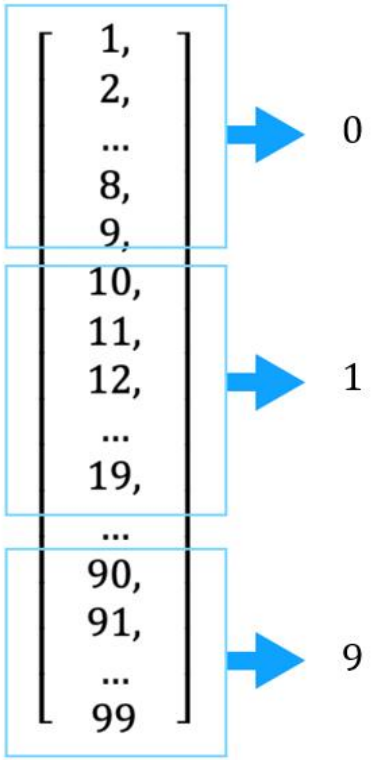
Vector indexes (main memory)

- Quantization-based indexes
  - E.g., IVF_FLAT, IVF_PQ
- Graph-based indexes
  - E.g., NSW, HNSW
- Tree-based indexes
- Hash-based indexes

Widely used in vector DBs

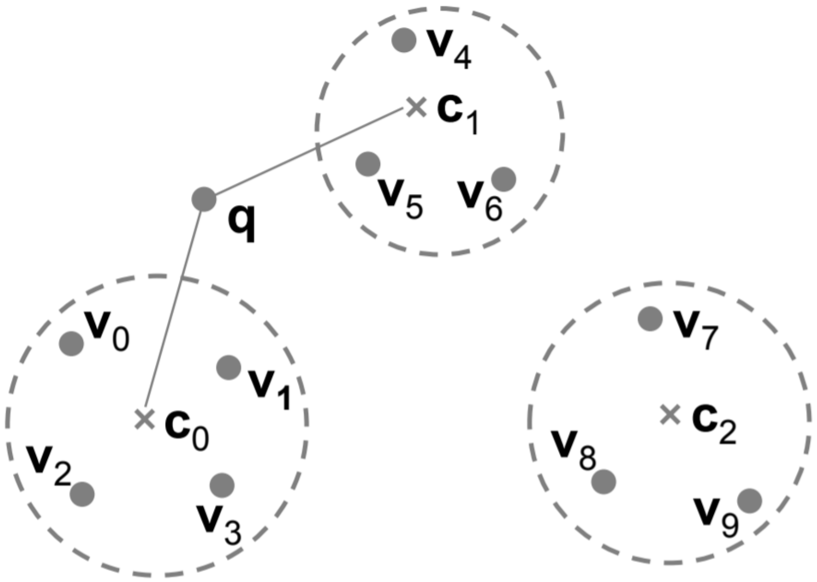
Quantization

- What's quantization?
  - A way of approximation
- Let's look at quantization in 1-dimensional space
  - $Q(x) = \left\lfloor \frac{x}{10} \right\rfloor$, where x is an input value
  - input = 3, $Q(3) = \left\lfloor \frac{3}{10} \right\rfloor = \lfloor 0.3 \rfloor = 0$
  - input = 3, $Q(91) = \left\lfloor \frac{91}{10} \right\rfloor = \lfloor 9.1 \rfloor = 9$
  - Those 99 integers can be quantized into a smaller set of 10 buckets



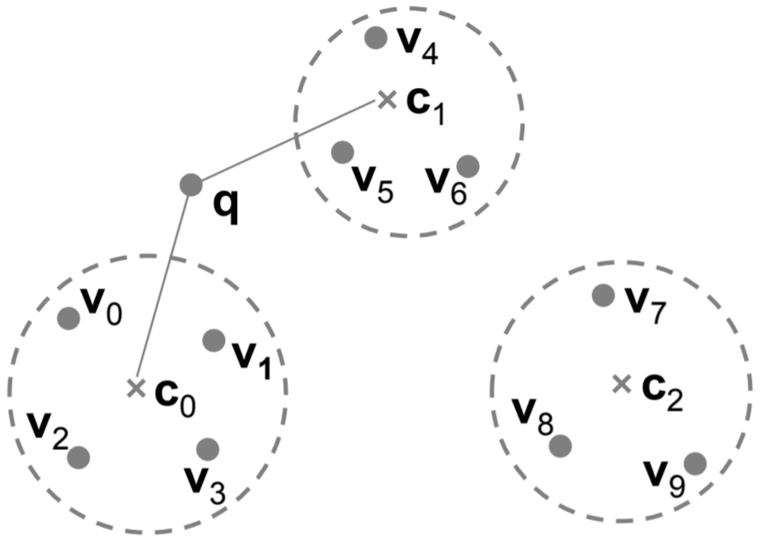
Quantization

- What's quantization in high-dimensional space?
  - It's basically clustering, e.g., k-means



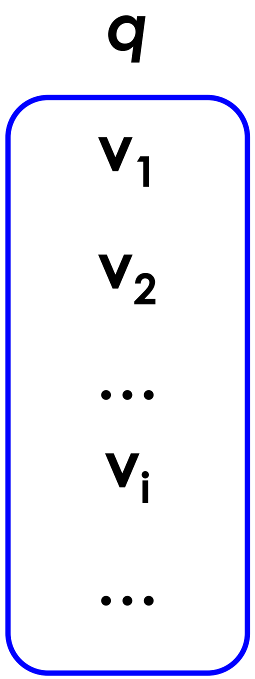
IVF_FLAT

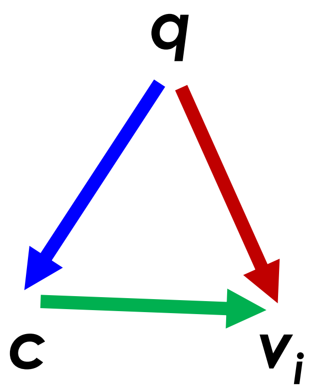
- Index phase
  - Cluster n vectors into K clusters (quantization)
  - Centroids: $c_0 \ldots c_{K-1}$
- Search phase
  - Given a query q, find the closest u clusters based on centroids
    - u: user-defined parameter
  - Only scan the vectors in the u clusters



IVF_FLAT

- Question: how to quickly compute the similarity between $\mathbf{q}$ and a vector $\mathbf{v}_i$ in a cluster?
- Naïve approach
  - A for-loop to compute dist($\mathbf{q}$,$\mathbf{v}_i$)
  - *d* steps (where *d* is dimensionality, e.g., d = 1000)
- Better solutions?
  - Remember, we know the centroid $\mathbf{c}$
  - We can pre-compute the distance of dist($\mathbf{c}$,$\mathbf{v}_i$)
- Then dist($\mathbf{q}$,$\mathbf{v}_i$) = dist($\mathbf{q}$,$\mathbf{c}$) + dist($\mathbf{c}$,$\mathbf{v}_i$) (approx.)
  - Only need 1 step to compute distance for all v_i





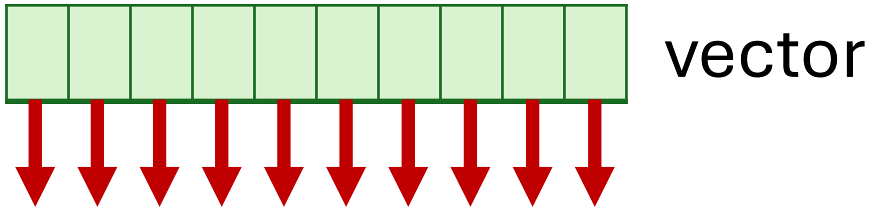
Compression

- How to reduce the space overhead of IVF_FLAT?
  - Compression
- Example
  - Youtube-8M data includes 1.4 billion vectors
  - Each vector takes 1024 dimensions (each float takes 32 bits)
  - 5.6TB space (memory!)

https://dl.acm.org/doi/10.14778/3424573.3424580

Compression: basic idea

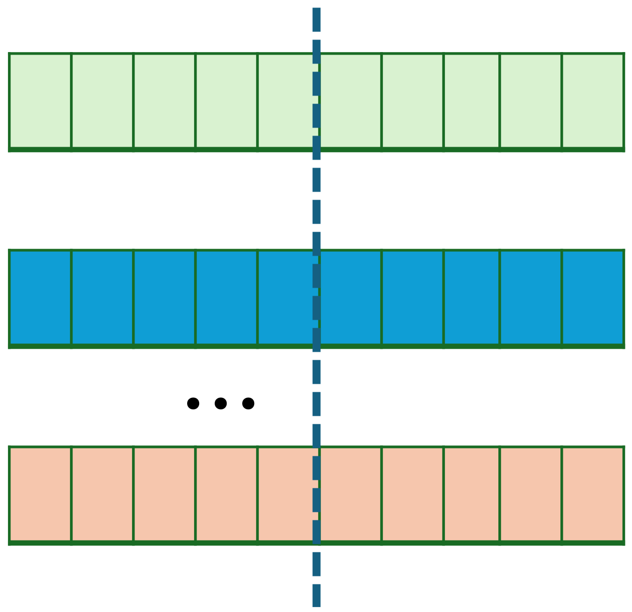
- Instead of using 32 bits to represent a float number
- Use L bits (e.g., $L = 8$)
- Think of 1-d quantization
- Every float number in a vector is quantized into $[0 \ldots 2^L-1]$
- The 1.4billion vectors will take 1.4TB space (if $L = 8$)



Every float number is mapped to [0…255] (8 bits per number)

Compression: product quantization (PQ)

- How to further reduce the space overhead?
- Product quantization (PQ)
  - Key idea: compress between multiple dimensions
  - Every vector is partitioned into M subvectors, e.g., $M = 8$
  - Every subvector is compressed using L bits (e.g., $L = 8$)

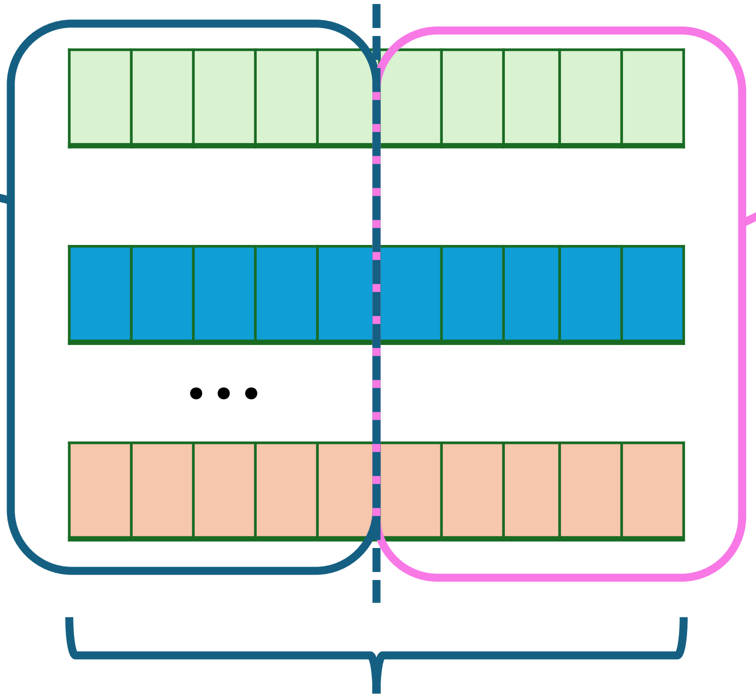


Compression: product quantization (PQ)

- How to compress subvectors?
- Each vector v_i is partitioned into *M* subvectors $v_i^0 \ldots v_i^{M-1}$
  - *M* subspace
- All the vectors in the same subspace are compressed together using high-dimensional quantization (clustering)
  - All $v_0^0, v_1^0, v_2^0 \ldots, v_{n-1}^0$ are compressed together
  - All $v_0^1, v_1^1, v_2^1 \ldots, v_{n-1}^1$ are compressed together
  - Every subvector is represented using the centroid ID

Compression: product quantization (PQ)

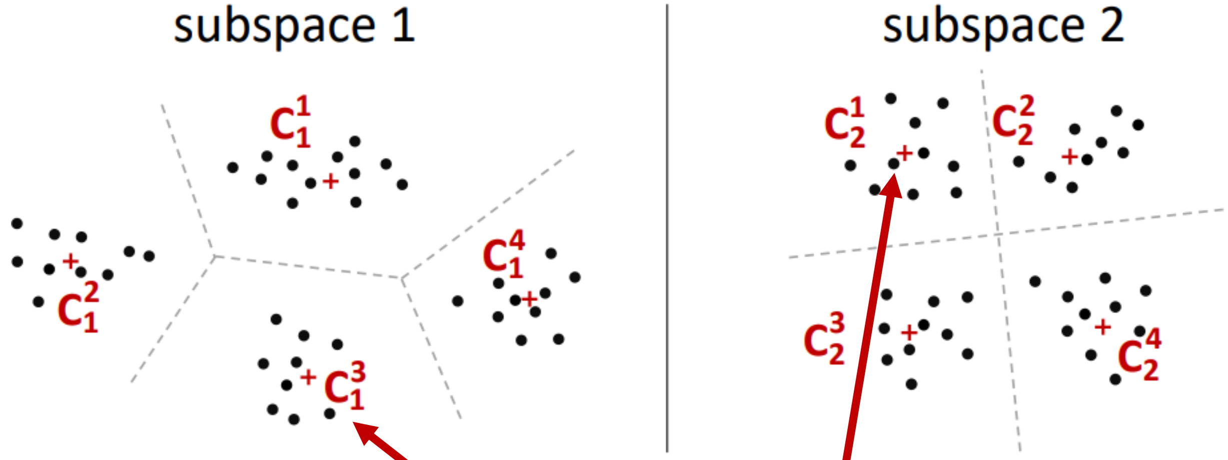
K-means clustering (every subvector is encoded using the centroid ID)



K-means clustering (every subvector is encoded using the centroid ID)

Every vector is split into 2 parts: head and tail vector
All the head vectors will be compressed together
All the tail vectors will be compressed together

Compression: product quantization (PQ)



C_1^1 := <2.25, 2.51>	C_2^1 := <9.10, 7.29>
C_1^2 := <8.35, 0.37>	C_2^2 := <7.76, 1.98>
C_1^3 := <5.00, 6.19>	C_2^3 := <0.98, 6.36>
C_1^4 := <8.86, 5.00>	C_2^4 := <1.29, 0.60>

$\langle \underbrace{5.53,\ 7.65,}_{\text{the subvector in subspace 1}}\ \underbrace{9.04,\ 6.80}_{\text{the subvector in subspace 2}} \rangle \rightarrow$ PQ-code: (*3*, *1*)

indexes of nearest centroids in each subspace (C_1^3 and C_2^1 resp.)

Original vector is compressed as <5.00, 6.19, 9.10, 7.29>

https://dl.acm.org/doi/10.14778/3424573.3424580

Compression: product quantization (PQ)

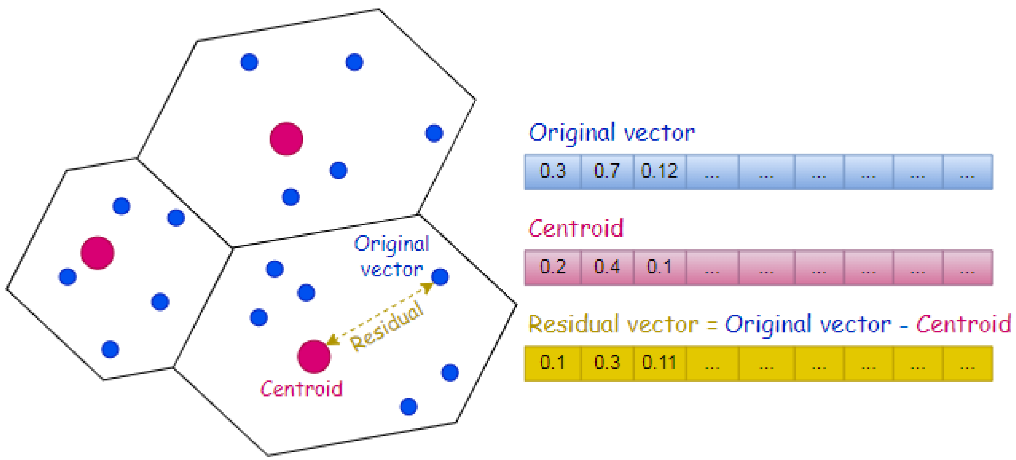
- Each vector is compressed using M*L bits
  - E.g., M = 8, L = 8
- Regardless of the dimensionality
  - But the parameters can be tuned based on dimensionality
- Example: Consider the 1.4 billion vectors again
  - Each vector will take 8*8 bits (M = 8, L = 8), i.e., 8 bytes
  - The 1.4billion vectors will take 11.2GB space

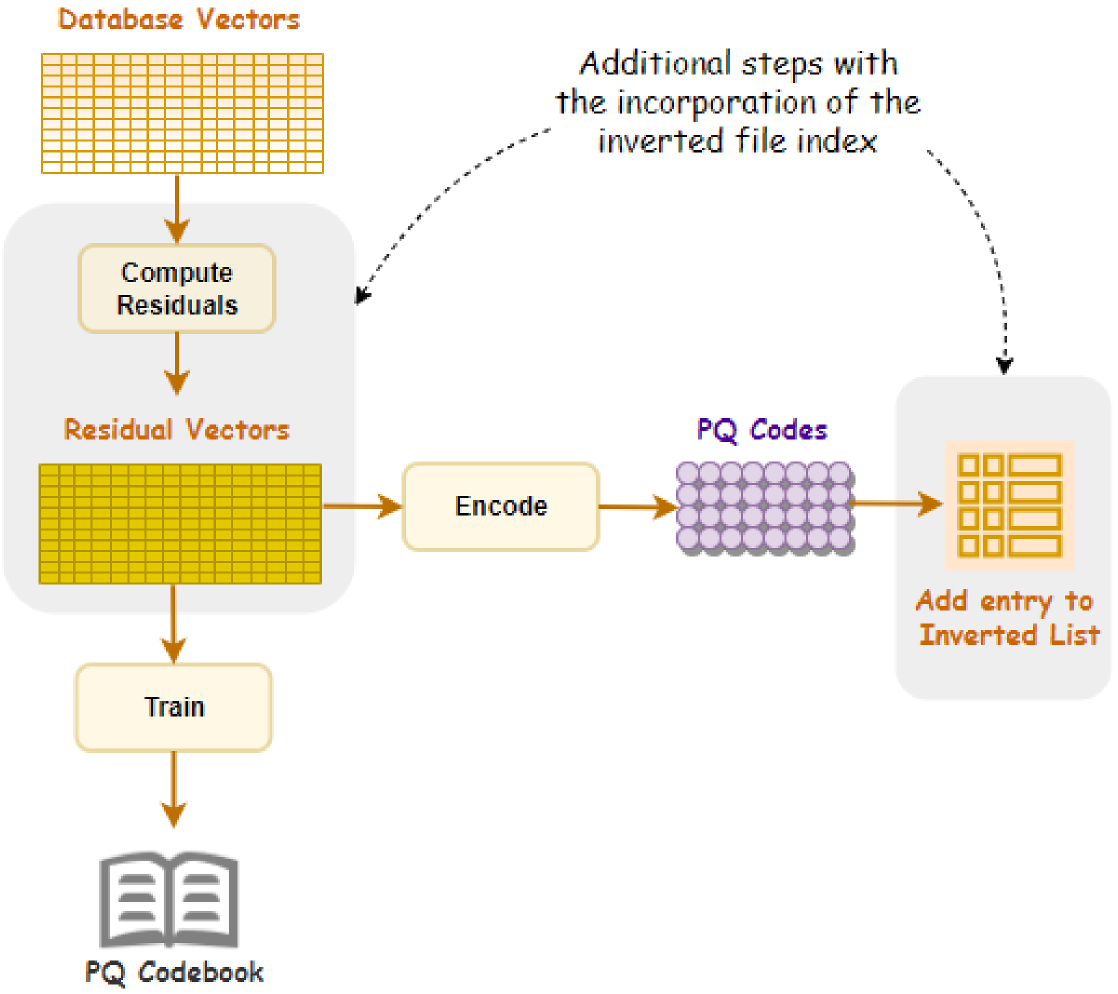
Compression: product quantization (PQ)

- What's the tradeoff? Space vs. accuracy
- Another benefit of PQ: Fast distance computation
  - All the distance in subspace can be precomputed
  - Example:
    - Vector X → PQ code (3, 1)
    - Vector Y → PQ code (1, 5)
    - Dist(X,Y) = dist(3,1) + dist(1,5), where each part can be precomputed

IVF_PQ

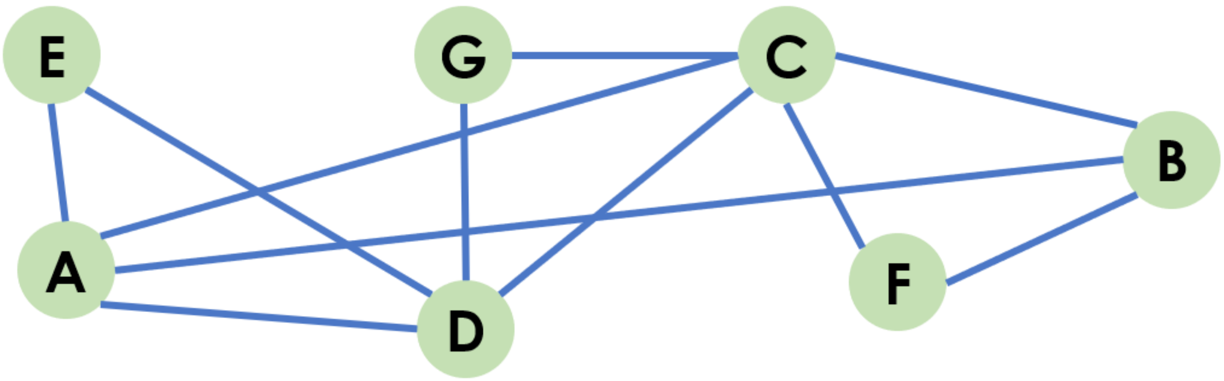
- Similar as IVF_FLAT
- Difference is that
  - Each cluster applies PQ
  - using residual vectors
- Search process is the same





Graph-based vector index

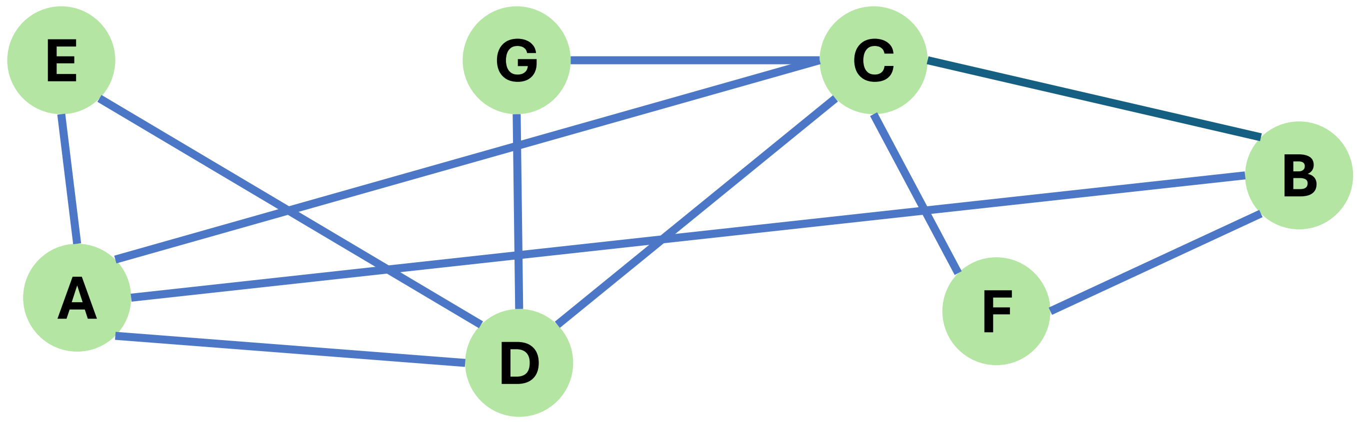
- Key ideas
  - For each vector, pre-compute the nearest neighbors
  - Connect them using a graph
  - Convert vector search problem to graph traversal problem



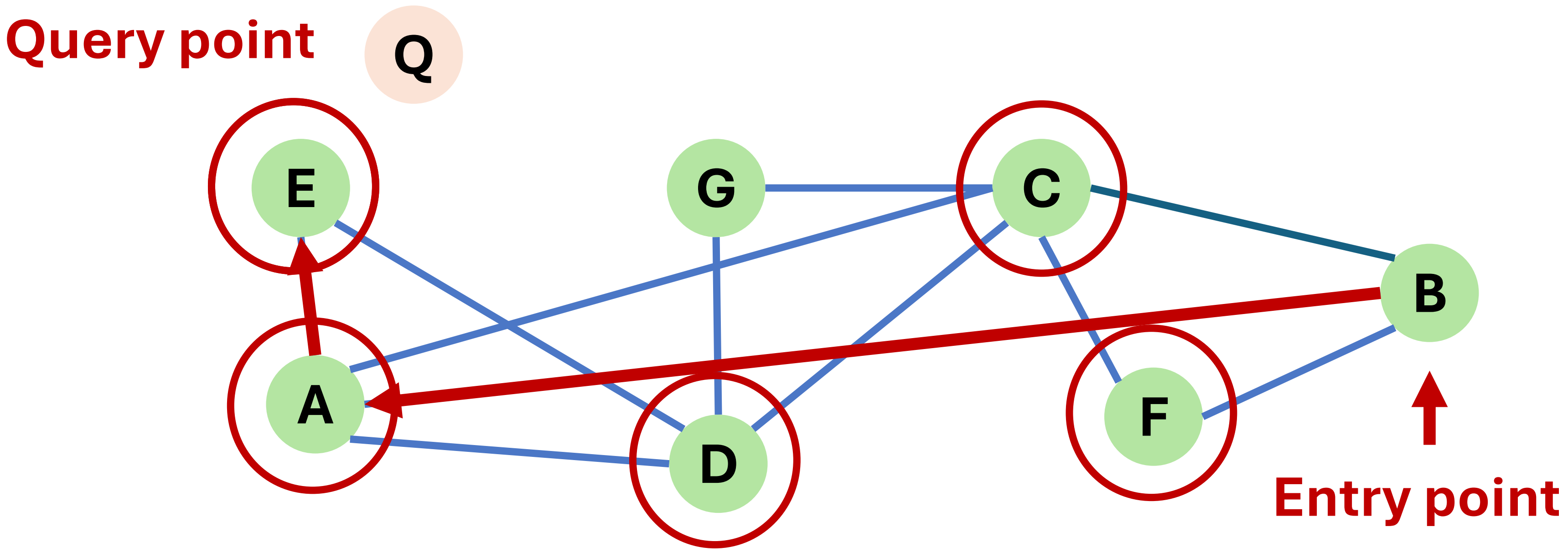
Graph-based vector index

- Navigable Small Worlds (NSW)
  - Add new vertices to the index
  - For each new vertex (vector), find the closest *m* neighbors seen so far and connect with them
  - Balance: index construction time & query performance

***m* = 2**



Graph-based vector index



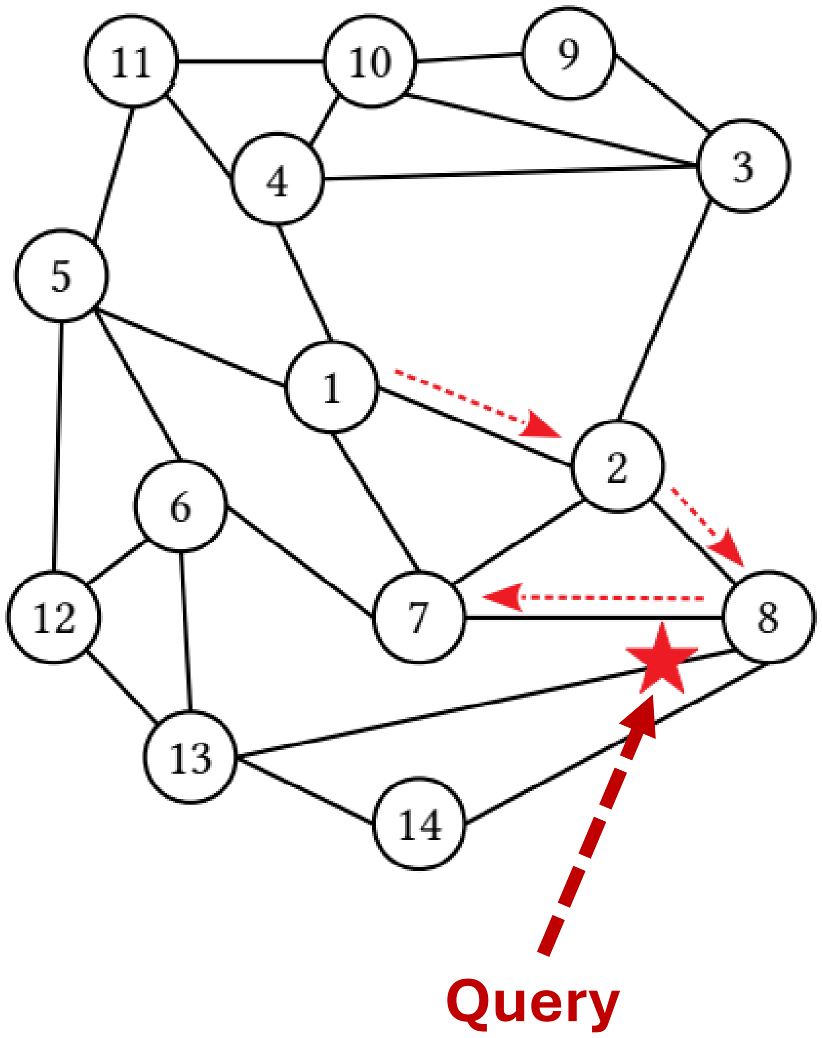
Can be extended to *k*NN by maintaining a result set and a candidate set

Terminate if the max distance in the result set < min distance in the candidate set

Candidate set

Result set

Initialization	*q*:	1
	topk:	∅
	visited:	1
Iteration 1	*q*:	2 7 4 5
	topk:	1
	visited:	1 2 4 5 7
Iteration 2	*q*:	8 7 3 4 5
	topk:	1 2
	visited:	1 2 3 4 5 7 8
Iteration 3	*q*:	7 3 4 5 14 13
	topk:	1 2 8
	visited:	1 2 3 4 5 7 8 13 14
Iteration 4	*q*:	3 4 5 6 14 13
	topk:	7 2 8
	visited:	1 2 3 4 5 6 7 8 13 14

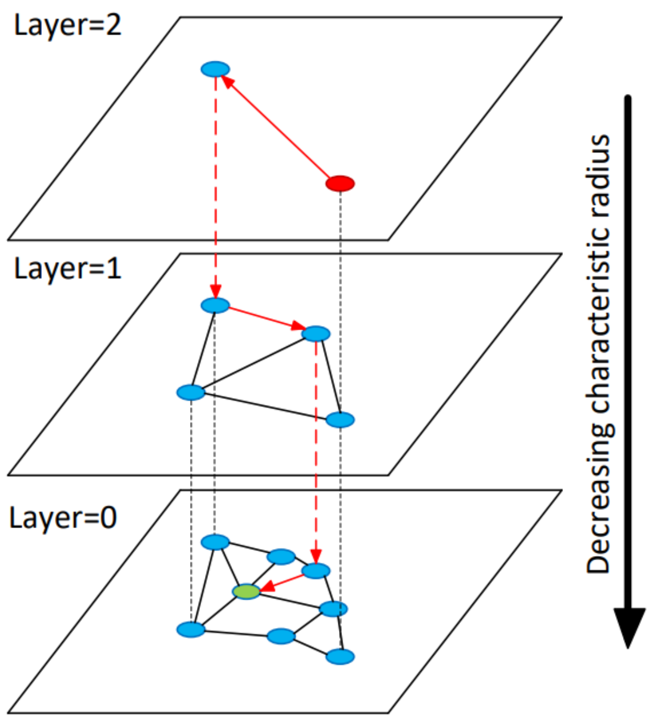


K = 3

https://ieeexplore.ieee.org/abstract/document/9101583

Graph-based vector index

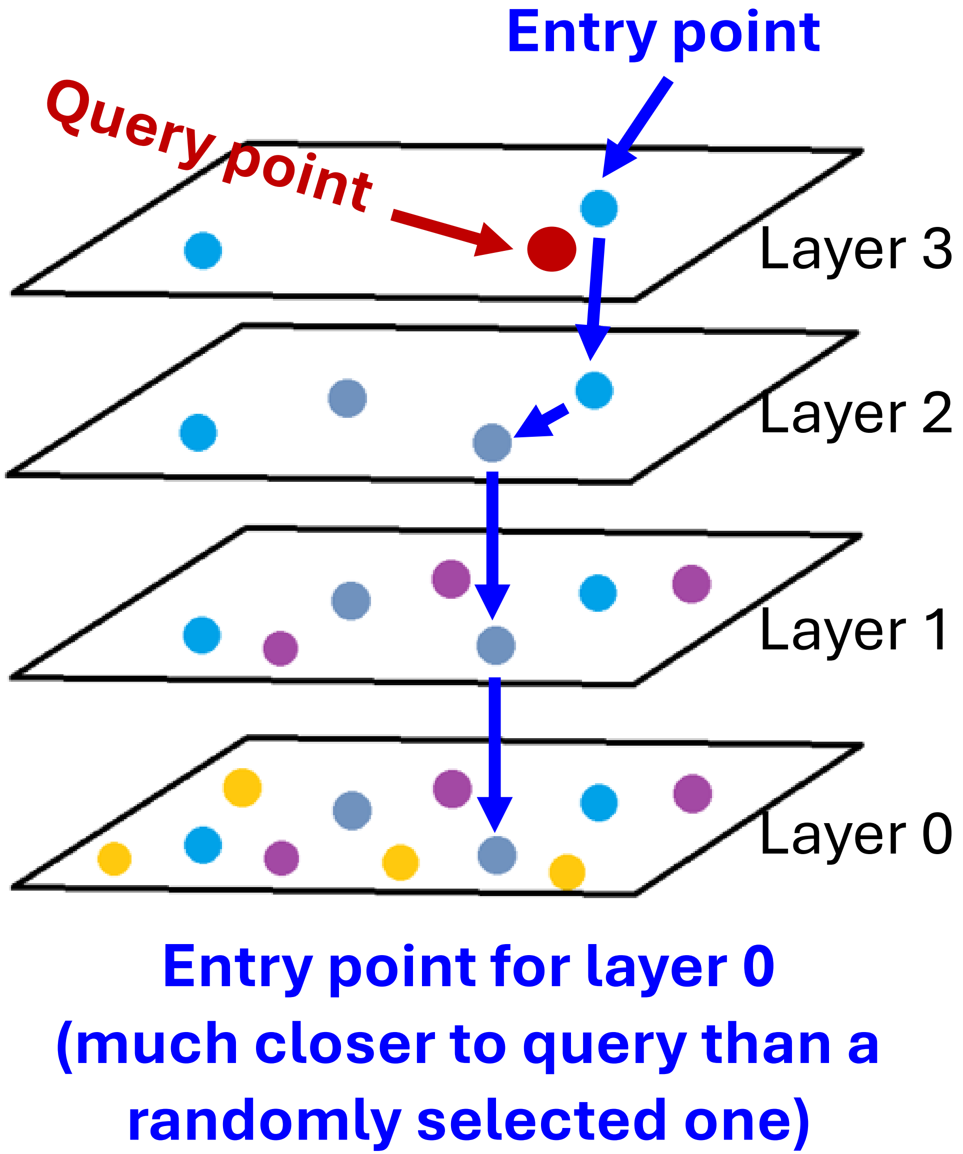
- Hierarchical Navigable Small Worlds (HNSW)
  - Skip list + NSW
  - Multi-layered NSW
  - Address the “bad” entry point issue
    - If the entry point is not selected properly, the search path is long



https://arxiv.org/ftp/arxiv/papers/1603/1603.09320.pdf

Graph-based vector index

- Every layer is an NSW on the sampled vertices
- Find the nearest vector in each layer, which will serve as the entry point for the next layer
- What if I choose a bad entry point from the top layer?
    - Slow, but acceptable
    - As the num of points is small in top layer



Outline

- Introduction
- Main-memory vector index
- **Disk-based vector index**
- Generalized vector DBs
- Specialized vector DBs

Overview of disk-based vector indexes

- Motivation: existing memory-based vector indexes consume too much memory (can be TBs of memory) to achieve high performance and recall
- Goal: reducing memory overhead while maintaining high performance and recall
- DiskANN (NeurIPS 2019): graph-based
- SPANN (NeurIPS 2021): quantization-based
- Starling (SIGMOD’24): graph-based

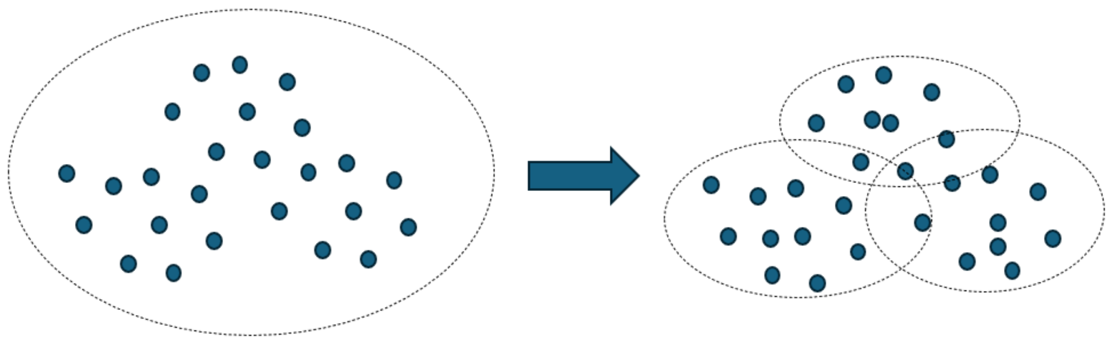
SPANN: Highly-efficient Billion-scale Approximate Nearest Neighbor Search

Qi Chen[1, *] **Bing Zhao**[1, 2, †] **Haidong Wang**[1] **Mingqin Li**[1] **Chuanjie Liu**[1, 3, †]
Zengzhong Li[1] **Mao Yang**[1] **Jingdong Wang**[1, 4, *, †]
[1]Microsoft [2]Peking University [3]Tencent [4]Baidu
[1]{cheqi, haidwa, mingqli, jasol, maoyang}@microsoft.com
[2]its.bingzhao@pku.edu.cn [3]liu.chuanjie@outlook.com [4]wangjingdong@outlook.com

NeurIPS 2021

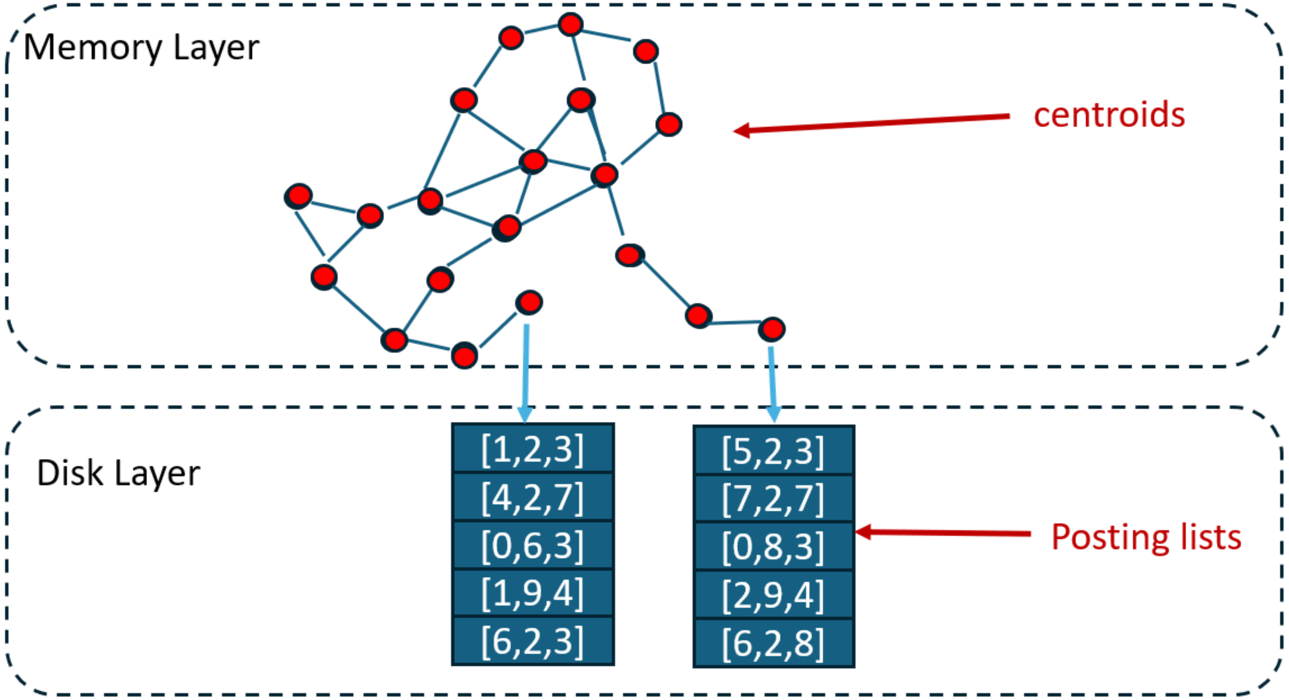
Key ideas

- Based on quantization
- Clustering the vectors into buckets (aka posting lists)
  - But the buckets are overlapped a bit (for optimizations)
- Centroids are stored in memory (organized by SPTAG index)
  - SPTAG: A tree-based index structure
- Posting lists are stored on disk



Key ideas

- Memory layer: SPTAG (for centroids)
- Disk layer: posting lists
- Search process
  - Search *m* nearest centroids from in-memory SPTAG
  - Load those *m* posting lists from disk

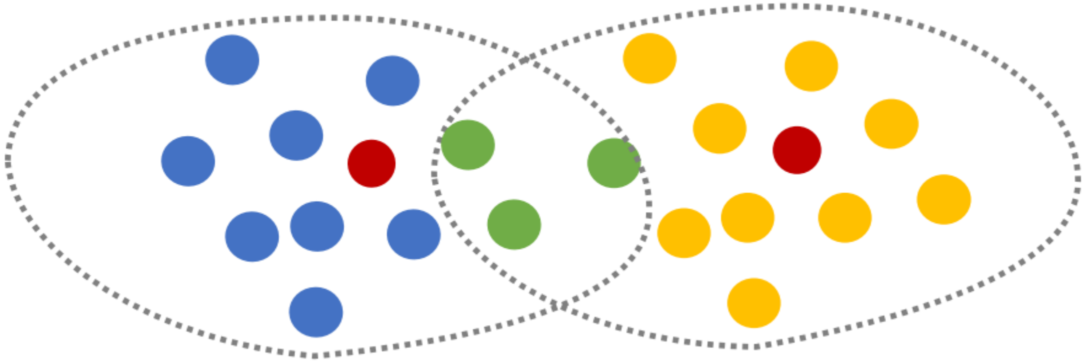


Optimizations in SPANN

- How to reduce disk access?
  - Some posting lists can be very long
- Solution
  - Partition the entire vectors into a large number of posting lists (so that each list is not very long)
  - Use multi-constraint balanced clustering to make sure some posting lists are not too long

Optimizations in SPANN

- How to improve recall?
  - Quantization-based indexes are difficult to achieve high recall
- Solution
  - Replicate boundary vectors into multiple posting lists
  - Overlapped clustering
  - How?



Green vectors are duplicated in two nearby clusters

Optimizations in SPANN

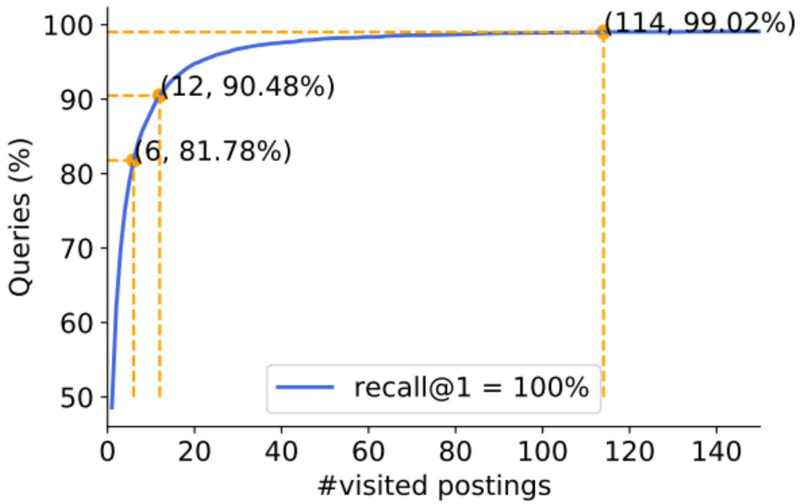
- Assign a vector to multiple closest clusters instead of only the closest one if the distance between the vector and these clusters are nearly the same

Nearest one

$$\mathbf{x} \in \mathbf{X}_{ij} \iff \mathrm{Dist}(\mathbf{x}, \mathbf{c}_{ij}) \le (1 + \epsilon_1) \times \mathrm{Dist}(\mathbf{x}, \mathbf{c}_{i1}),$$
$$\mathrm{Dist}(\mathbf{x}, \mathbf{c}_{i1}) \le \mathrm{Dist}(\mathbf{x}, \mathbf{c}_{i2}) \le \cdots \le \mathrm{Dist}(\mathbf{x}, \mathbf{c}_{iK})$$

Optimizations in SPANN

- How many posting lists to load for a given query?
  - Different queries may need different number
- Solution: query-aware dynamic pruning
  - Observation: Some queries only need to search several posting lists to find true neighbors, while some search a lot



80% of queries only need to search 6 posting lists

Optimizations in SPANN

- Query-aware dynamic pruning
  - Instead of searching closest *m* posting lists for all queries, dynamically decide a posting list to be searched only if the distance between its centroid and query is almost the same as the distance between query and the closest centroid

Nearest one

$$\mathbf{q} \xrightarrow{search} \mathbf{X}_{ij} \iff \mathrm{Dist}(\mathbf{q}, \mathbf{c}_{ij}) \leq (1 + \epsilon_2) \times \mathrm{Dist}(\mathbf{q}, \mathbf{c}_{i1}),$$
$$\mathrm{Dist}(\mathbf{q}, \mathbf{c}_{i1}) \leq \mathrm{Dist}(\mathbf{q}, \mathbf{c}_{i2}) \leq \cdots \leq \mathrm{Dist}(\mathbf{q}, \mathbf{c}_{iK})$$

DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node

Suhas Jayaram Subramanya*
Carnegie Mellon University
suhas@cmu.edu

Devvrit*
University of Texas at Austin
devvrit.03@gmail.com

Rohan Kadekodi*
University of Texas at Austin
rak@cs.texas.edu

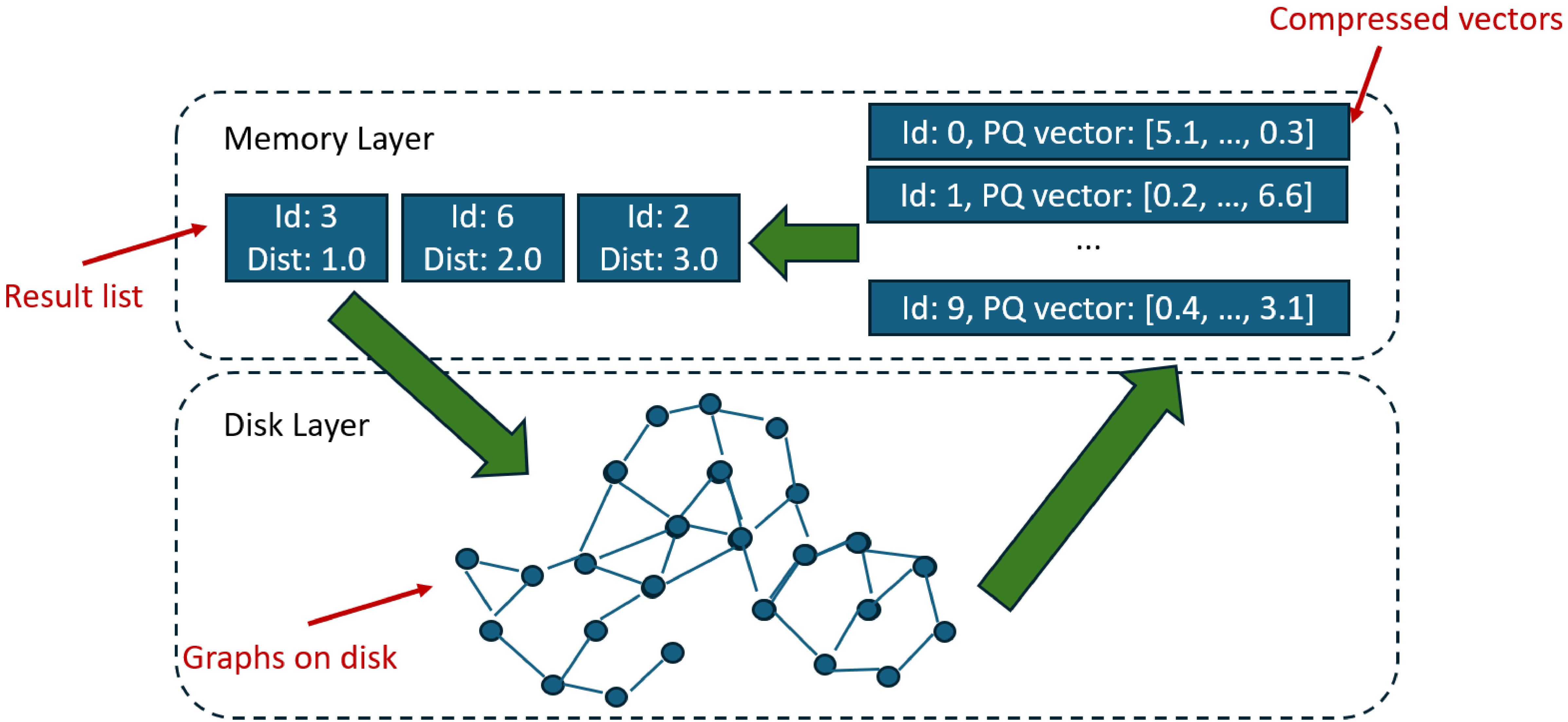
Ravishankar Krishaswamy
Microsoft Research India
rakri@microsoft.com

Harsha Vardhan Simhadri
Microsoft Research India
harshasi@microsoft.com

NeurIPS 2019

Key ideas

- Graph-based index
- Disk layer: Graph structure
  - Graph structure is similar to HNSW (called Vamana)
  - With full-precision vectors
  - Structure: vector itself followed by adjacent vector IDs
- Memory layer: Compressed vectors
  - PQ-compressed vectors



Compressed vectors
Memory Layer
Id: 0, PQ vector: [5.1, ..., 0.3]
Id: 1, PQ vector: [0.2, ..., 6.6]
...
Id: 9, PQ vector: [0.4, ..., 3.1]
Id: 3
Dist: 1.0
Id: 6
Dist: 2.0
Id: 2
Dist: 3.0
Result list
Disk Layer
Graphs on disk

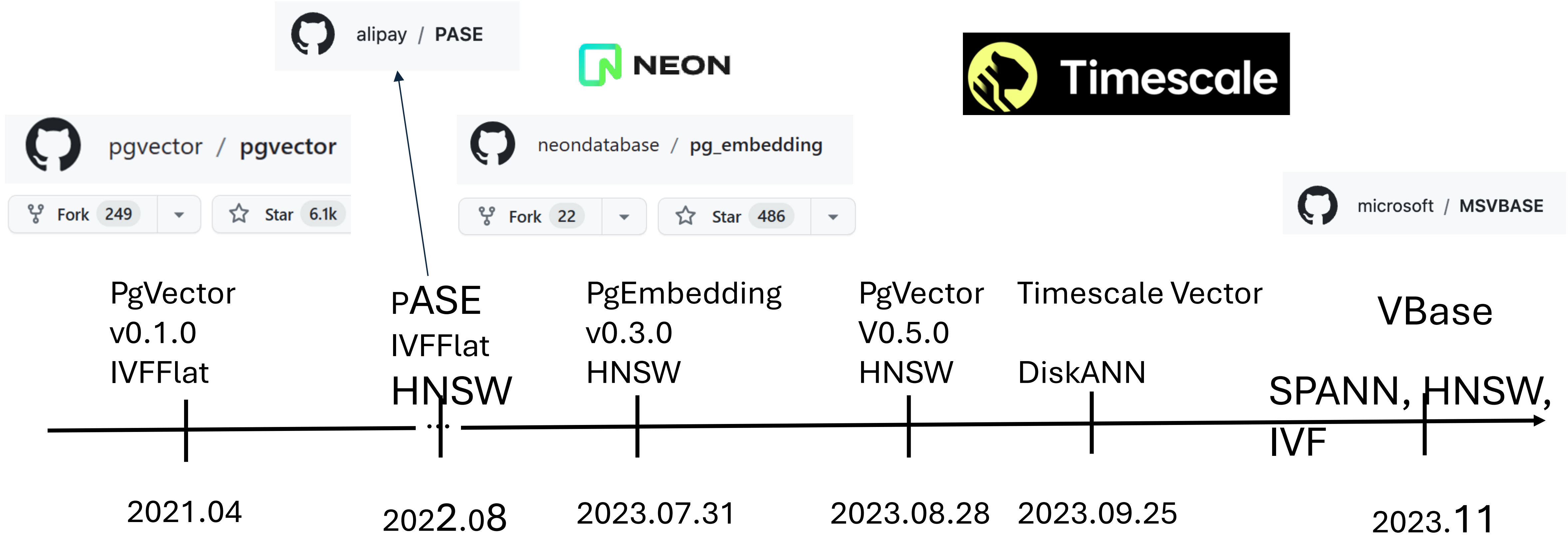
Outline

- Introduction
- Main-memory vector index
- Disk-based vector index
- **Generalized vector DBs**
- Specialized vector DBs

Vector databases: specialized vs. generalized

- **Specialized vector databases**
  - Explicitly designed for vector data
- **Generalized vector databases**
  - Support vector search within relational databases
  - One-size-fits-all

Vector search in PostgreSQL



Vector search in PostgreSQL

pgvector is similar to PASE

PASE: PostgreSQL Ultra-High-Dimensional Approximate Nearest Neighbor Search Extension

Wen Yang
Ant Financial
yangwen.yw@antfin.com

Tao Li
Ant Financial
lyee.lit@antfin.com

Gai Fang
Ant Financial
fanggai.fg@alibaba-inc.com

Hong Wei
Ant Financial
weihong.wh@antfin.com

SIGMOD'20

PASE

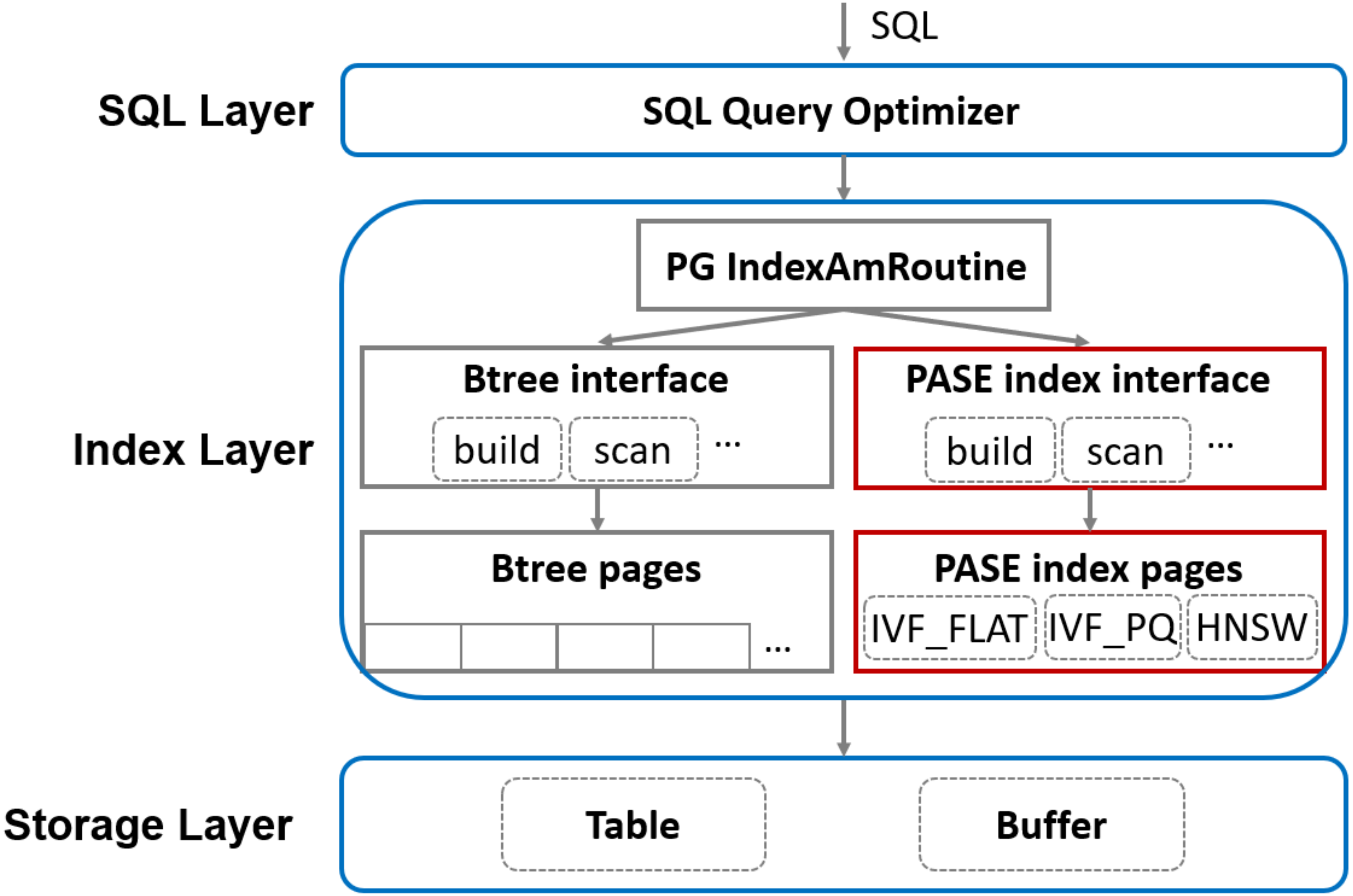
- Key ideas
  - Store vectors in a separated column
  - Build high-d indexes on that vector column
  - Similar to B-trees on other columns
  - Intuitively, should have very high performance

ID (Int Type)	Vector (Array Type)
0	[0.1, 0.5, 0.6, 0.3]
1	[0.3, 0.2, 0.9, 0.1]
2	[0.5, 0.5, 0.3, 0.4]
3	[0.9, 0.1, 0.3, 0.2]
4	[0.6, 0.4, 0.3, 0.8]

Challenges in PASE

- How to make the newly-built high-d index recognizable by the SQL query optimizer?
- How to configure the internal parameters of high-d indexes?
- How to define and specify similarity functions?
- How to represent vector search using SQL?

PASE architecture



SQL layer of PASE

- Extend SQL syntax for vector search

```sql
CREATE TABLE T (id int, vec float[]);

SELECT   id
FROM     T
ORDER BY vec <op> '0.1,0.2,0.3,0.4'::PASE ASC
LIMIT    10;
```

<op> is a special operator to compute the similarity between two vectors.

Index layer of PASE

```
CREATE INDEX ivfflat_idx ON T
       USING ivfflat_fun(vec)
       WITH  (distance_type = 0, dimension = 128,
             clustering_params = "10,256");
```

Sampling ratio: 10/1000
256: # of clusters in IVF_FLAT

Index Layer of PASE

- To be recognizable by SQL optimizer:
  - Implement certain index interfaces, e.g., build(), insert(), delete(), scan(), via PG's IndexAmRoutine()
  - The index needs to follow PG's index page structure in order to be accessed via the buffer manager and storage engine
- These restrictions can affect performance

Storage layer of PASE

- Store vector data same as other attributes in a table
- Tables and indexes are stored on disk, but frequently accessed pages are cached in memory via the buffer manager

Try Timescale Vector

PostgreSQL++ for AI Applications

Get started for free

Blog Categories

All posts
AI
Announcements
Cloud
Developer Q&A
Engineering
General
Grafana
Observability
PostgreSQL
Product Updates

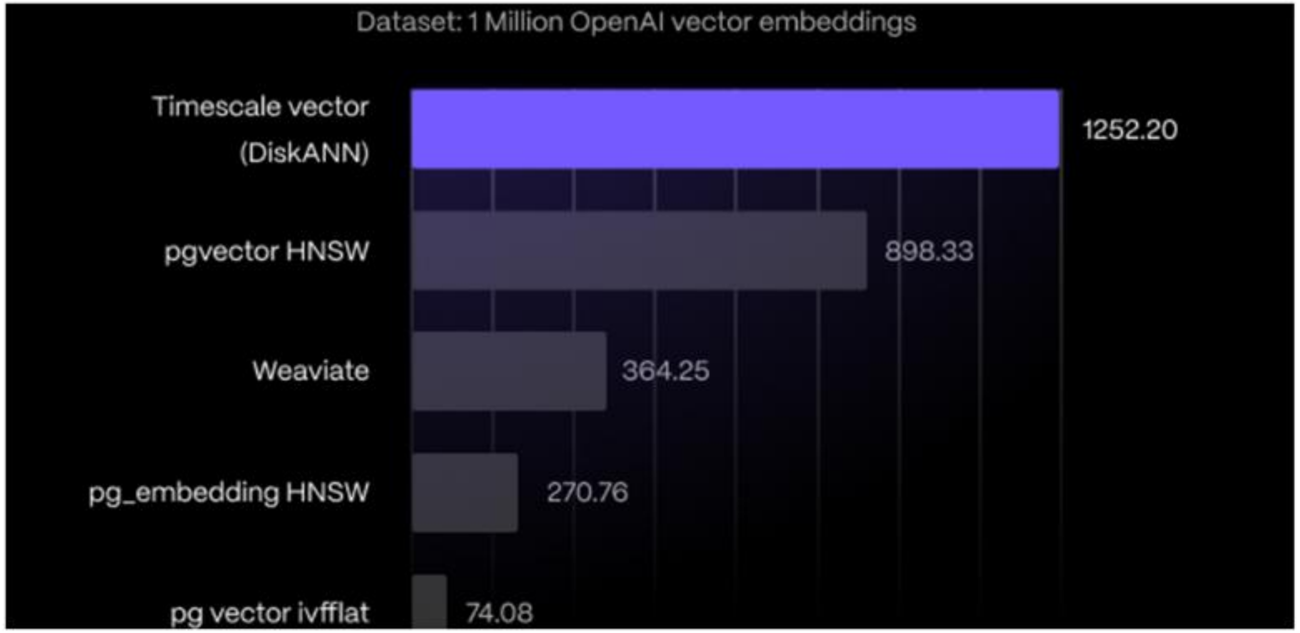
How We Made PostgreSQL a Better Vector Database

25 Sep 2023
24 min read

AI

Contributors
Avthar Sewrathan
Matvey Arye
Samuel Gichohi
Maheedhar PV

Share



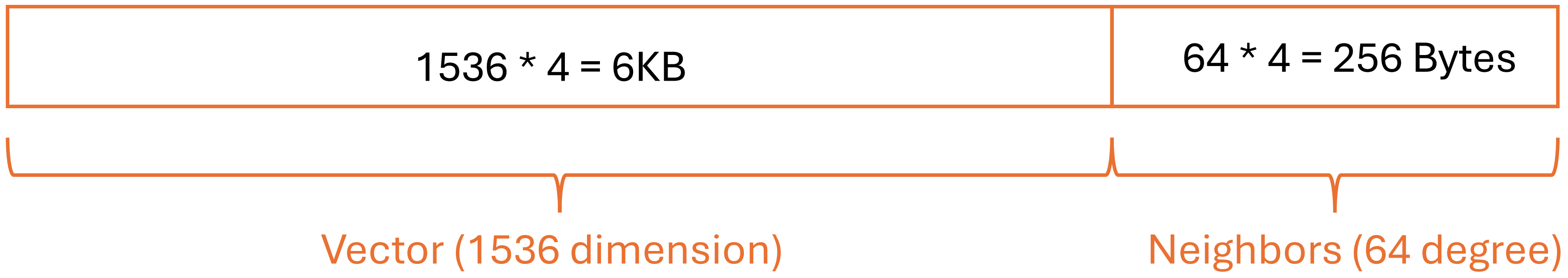
Introducing Timescale Vector, PostgreSQL++ for production AI applications. Timescale Vector enhances pgvector with faster search, higher recall, and more efficient time-based filtering, making PostgreSQL your new

How We Made PostgreSQL a Better Vector Database (timescale.com)

Timescale-vector

[1]Jayaram Subramanya, Suhas, et al. "Diskann: Fast accurate billion-point nearest neighbor search on a single node." Advances in Neural Information Processing Systems 32 (2019).

- Inspired by DiskANN[1] (Optimized for disk)
  - On-disk data layout
    - Cluster each node vector with its neighboring links
  - Single layer graph can further augment the cache's efficiency
    - Unlike HNSW's hierarchical structure



Outline

- Introduction
- Main-memory vector index
- Disk-based vector index
- Generalized vector DBs
- **Specialized vector DBs**

Milvus: A Purpose-Built Vector Data Management System

Jianguo Wang*, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, Charles Xie

* Zilliz & Purdue University

* *csjgwang@{zilliz.com; purdue.edu}*

Zilliz

{firstname.lastname}@zilliz.com

SIGMOD 2021

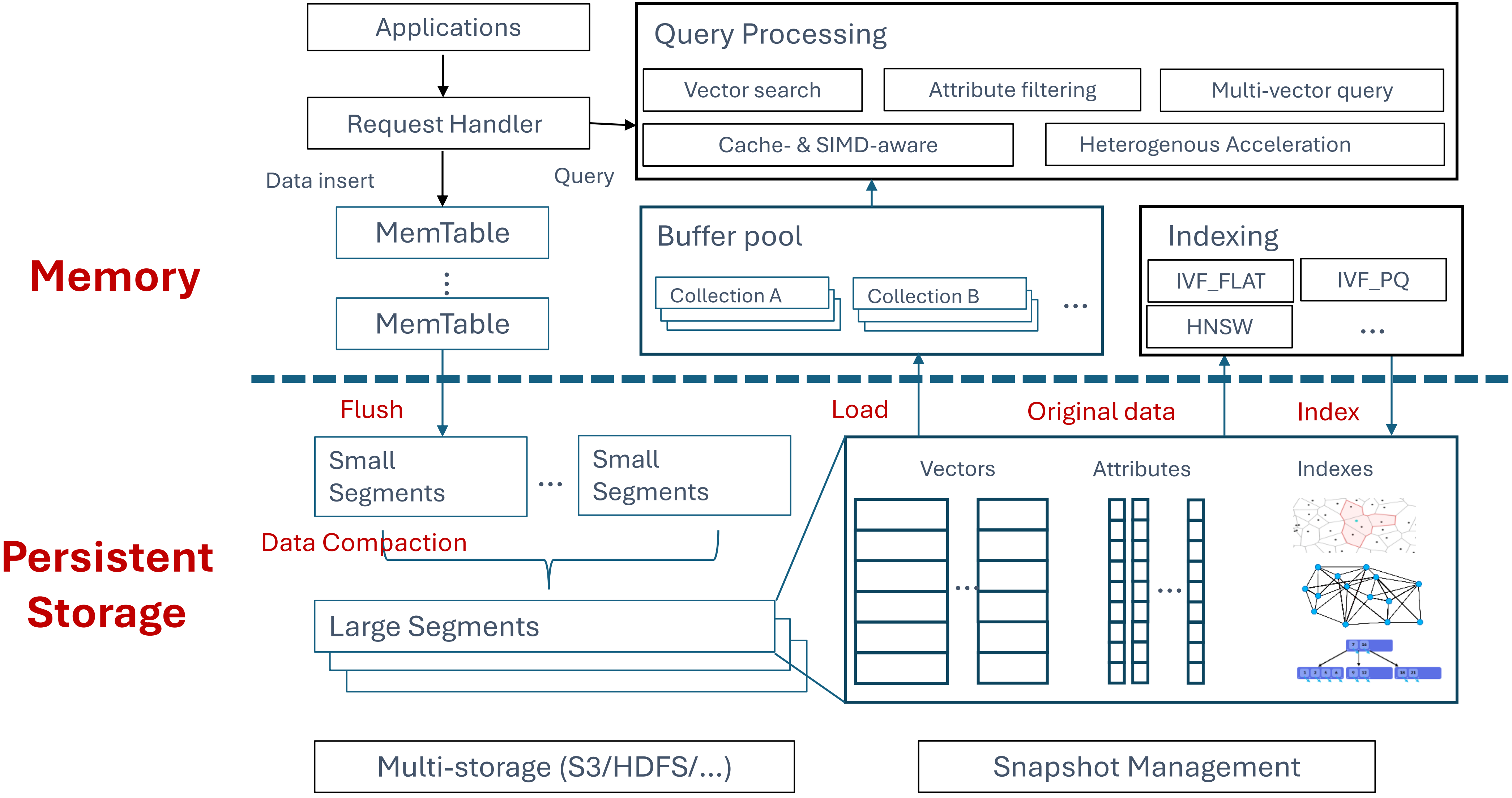
Motivation

- The motivation in 2021 was different from today
  - ❶ Explosive growth of unstructured data
  - ❷ Vector embedding is everywhere (e.g., item2vec, word2vec, doc2vec, graph2vec)
- Research question
  - How to efficiently manage large-scale vector data?

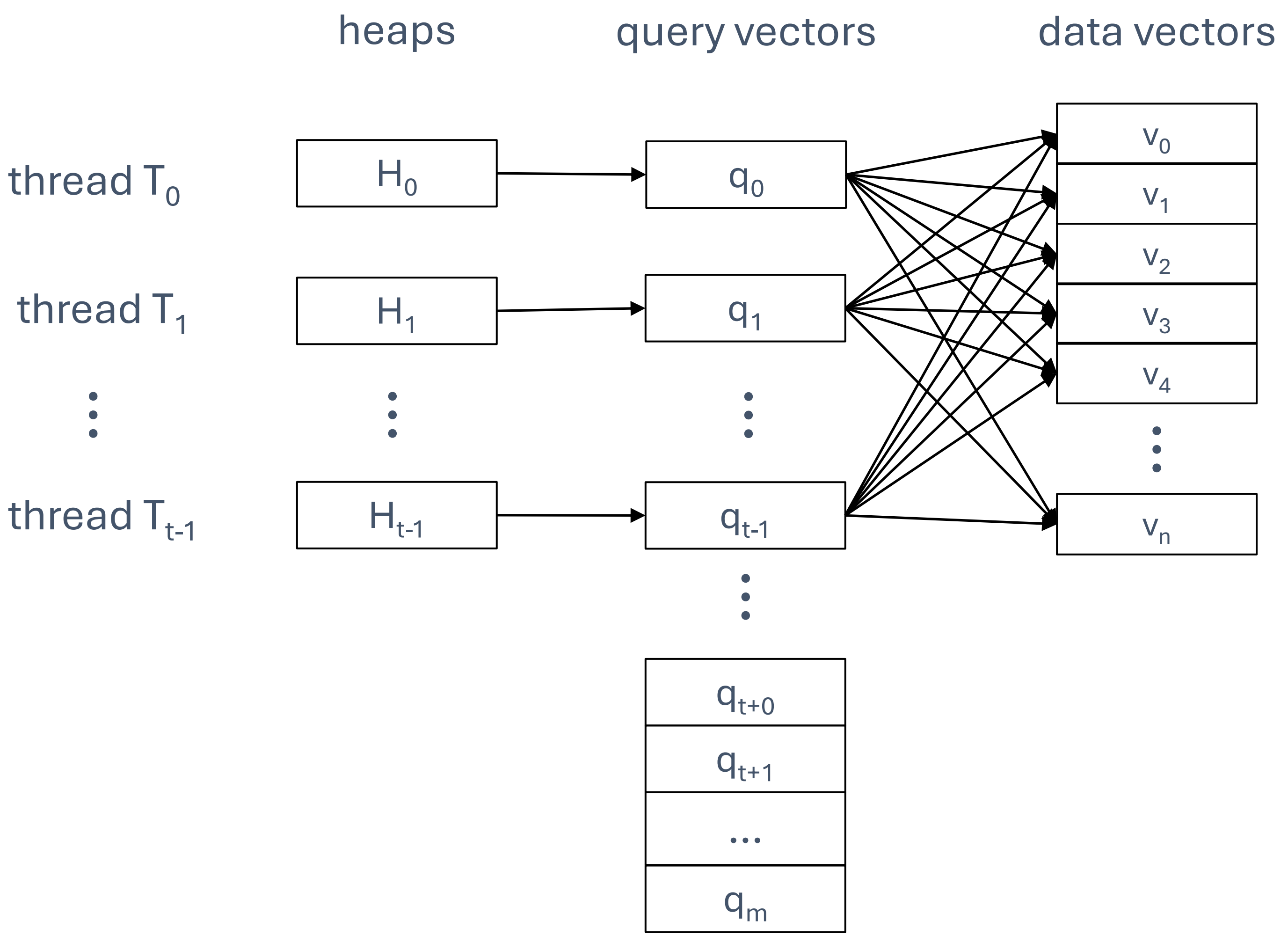
Requirements

- **Efficiency**
  - Large-scale vector data
  - Dynamic vector data management
- **Advanced query semantics**
  - Vector similarity search
  - Attribute filtering (filtered vector search)
  - Multi-vector search

Systems overview

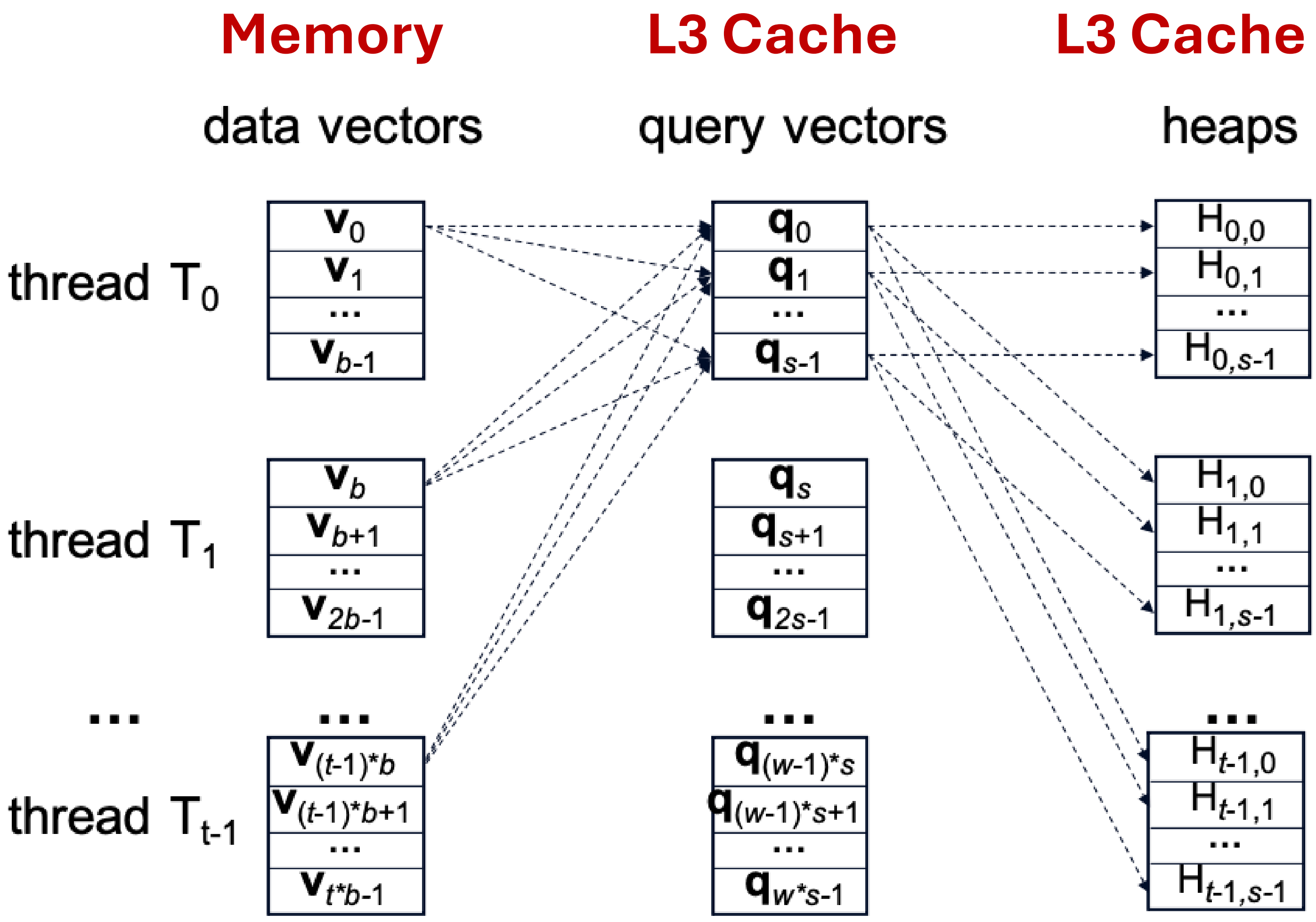


Cache-aware design: naïve solution

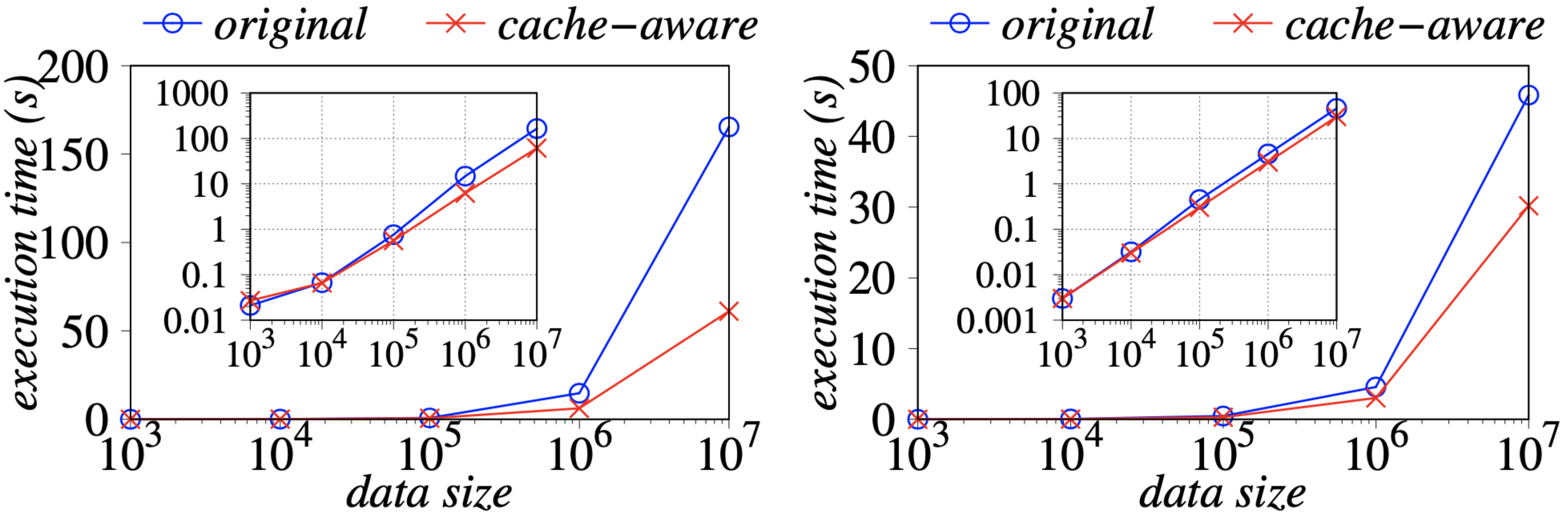


- *m* queries, *n* vectors, *t* threads, find for each vector its top-k
  - Basic operation in scanning a bucket
- Naïve solution
  - Assign one query to a thread every time
  - Compare query vector with all data vectors
- Limitations
  - High cache miss rate
  - Limited parallel when *m* is small

Cache-aware design in Milvus



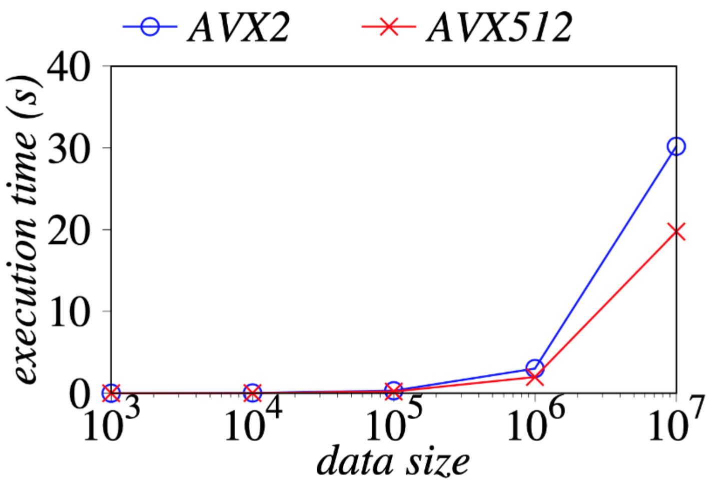
- Divide both data vectors and query vectors into blocks
- Process one query vector block per time
- Similar to block nested loop join
  - Query block: inner loop
  - Data block: outer loop
- Assign data vector blocks to threads (usually n > m)
- One heap per thread per query vector (avoid lock)



(a) 12MB L3 cache

(b) 35.75MB L3 cache

SIMD-aware optimizations



- Milvus supports SIMD SSE, AVX, AVX2, and AVX512
- Automatic SIMD-instruction selection
  - Challenge: how to make it automatically invoke the suitable SIMD instructions on any CPU processor?
  - Faiss: users need to manually specify the SIMD flag during compilation time
  - Milvus factors out the common functions (e.g., similarity computing) that rely on SIMD accelerations
  - Also, for each function, it implements four versions (i.e., SSE, AVX, AVX2, AVX512) and puts each one into a separated source file, which is further compiled individually with the corresponding SIMD flag

CPU and GPU co-design for vector search

- Limitations in the GPU design in Faiss
  - The PCIe bandwidth is not fully utilized, e.g., our experiments show that the measured I/O bandwidth is only 1～2GB/s while PCIe 3.0 supports up to 15.75GB/s
  - It is not always beneficial to execute queries on GPU (than CPU) considering the data transfer

CPU and GPU co-design for vector search

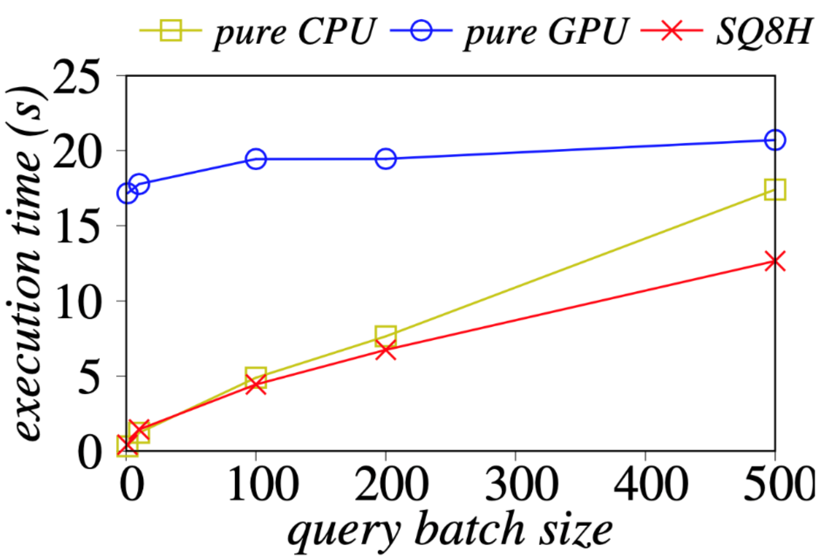
- Addressing limitation 1
  - Copy multiple buckets from CPU to GPU every time (while Faiss copies buckets one by one)
- Addressing limitation 2
  - Observation: GPU outperforms CPU if the query batch size is large enough considering the expensive data movement

CPU and GPU co-design for vector search

- CPU-GPU co-design

Algorithm 1: SQ8H

```
let n_q be the batch size;
if n_q ≥ threshold then
    run all the queries entirely in GPU (load multiple buckets
    to GPU memory on the fly);
else
    execute the step 1 of SQ8 in GPU: finding n_probe buckets;
    execute the step 2 of SQ8 in CPU: scanning every relevant
    bucket;
```



Higher ratio of computation-to-I/O

Multi-vector search

- In some applications, each entity has multiple vectors
  - E.g., each person is described using multiple vectors to describe the front face, side face, and posture
- Another source of multi-vector is using multiple embedding models to represent the same object

https://butterflymx.com/blog/multi-camera-security-system/

Multi-vector search

- Problem
  - Both data entries *v* and queries *q* contain *m* vectors. Given a vector similarity function *f* and a score aggregation function *g*, find *k* entries with highest score value:

$$g(f(v_1, q_1), f(v_2, q_2), \dots, f(v_\mu, q_\mu))$$

  - E.g., *g* can be weighted sum

Multi-vector search

- Vector fusion
  - Merge the multiple subvectors into a single vector $\mathbf{v} = [e.\mathbf{v}_0, e.\mathbf{v}_1, ..., e.\mathbf{v}_{\mu-1}]$
  - Perform regular vector search
  - However, it requires the similarity function is decomposable, e.g., dot product

$$g\left(f(v_1, q_1), f(v_2, q_2), \dots, f(v_\mu, q_\mu)\right)$$
$$= f\left(h(v_1, \dots, v_\mu), h'(q_1, \dots, q_\mu)\right)$$

f: inner product
g: weighted sum

h: concatenation,
h': weighted concatenation

$$[w_0 \times q.\mathbf{v}_0, w_1 \times q.\mathbf{v}_1, ..., w_{\mu-1} \times q.\mathbf{v}_{\mu-1}]$$

Multi-vector search

- Use Fagin's algorithm
- But it relies on getNext(), which is inefficient on vector index
- Milvus develops an iterative merging algorithm that bypasses getNext()

R_1	
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

R_2	
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

R_3	
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

https://www.sciencedirect.com/science/article/pii/S0022000003000266

Multi-vector search

- Search top-k’ result for each q.subvector
- Try to find results with Fagin's algorithm (NRA)
- If k results can be determined
  - Return results
- Otherwise
  - Increase k’ and repeat

Credits

- Jianguo Wang, Purdue
- Silu Huang, ByteDance