Santiago
Chile

SIGMOD
PODS
2024

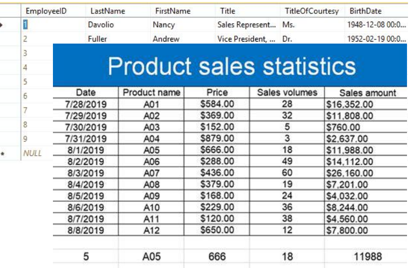
Vector Database Management Techniques and Systems

James Jie Pan[1] Jianguo Wang[2] Guoliang Li[1]

[1]Tsinghua University
[2]Purdue University

Modern DBMS Landscape

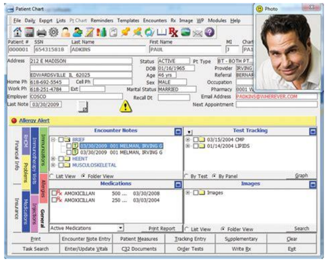
Modern DBMSs are designed for data that humans can understand



Product sales statistics

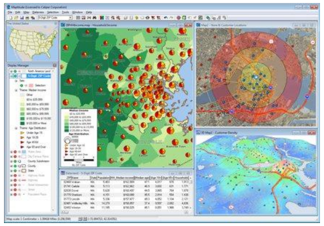
Date	Product name	Price	Sales volumes	Sales amount
7/28/2019	A01	$584.00	28	$16,352.00
7/29/2019	A02	$369.00	32	$11,808.00
7/30/2019	A03	$152.00	5	$760.00
7/31/2019	A04	$879.00	3	$2,637.00
8/1/2019	A05	$666.00	18	$11,988.00
8/2/2019	A06	$288.00	49	$14,112.00
8/3/2019	A07	$436.00	60	$26,160.00
8/4/2019	A08	$379.00	19	$7,201.00
8/5/2019	A09	$168.00	24	$4,032.00
8/6/2019	A10	$229.00	36	$8,244.00
8/7/2019	A11	$120.00	38	$4,560.00
8/8/2019	A12	$650.00	12	$7,800.00
5	A05	666	18	11988

Business Operations

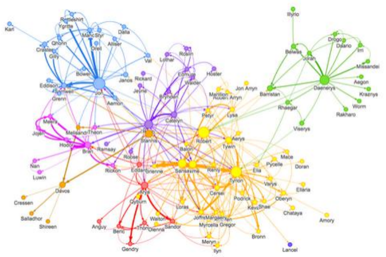


Medical Records

Financial Timeseries



Geospatial Vectors/Rasters



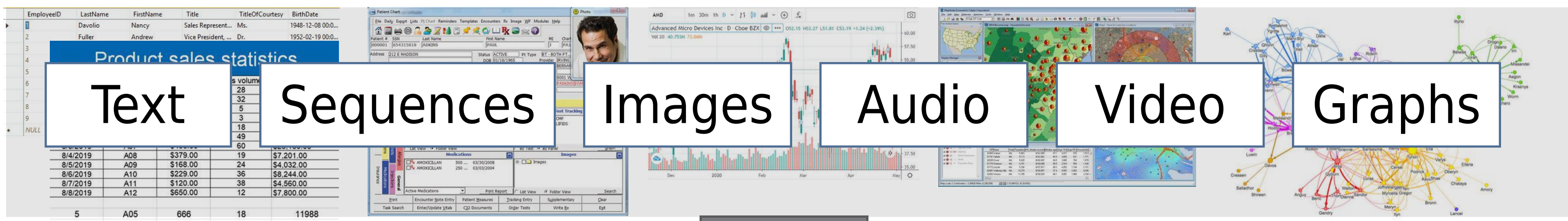
Social Networks

Relational/NewSQL

NoSQL/Specialty

Embeddings: Building Blocks of the Future

More and more applications rely on deep-learning **embedding vectors** that can only be understood by machines



DeepMind | Hugging Face | OpenAI

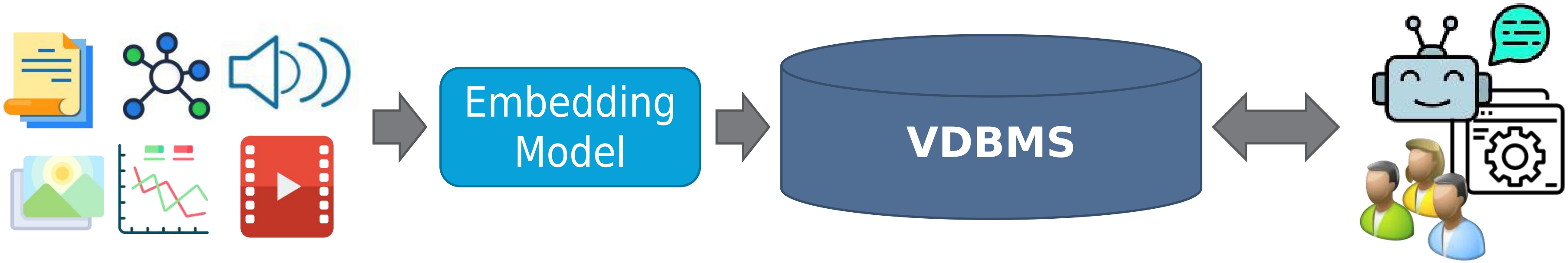
MiniLM-L6 | MPNet-Base | ResNet-50 | CLIP VIT | Whisper | Multimodal Receiver

Forecasting | Pattern Recognition | Object Detection | Segmentation

Classification | Reinforcement Learning | Semantic Retrieval | Generation

RecSys

Goal: Vector DBMS (VDBMS)

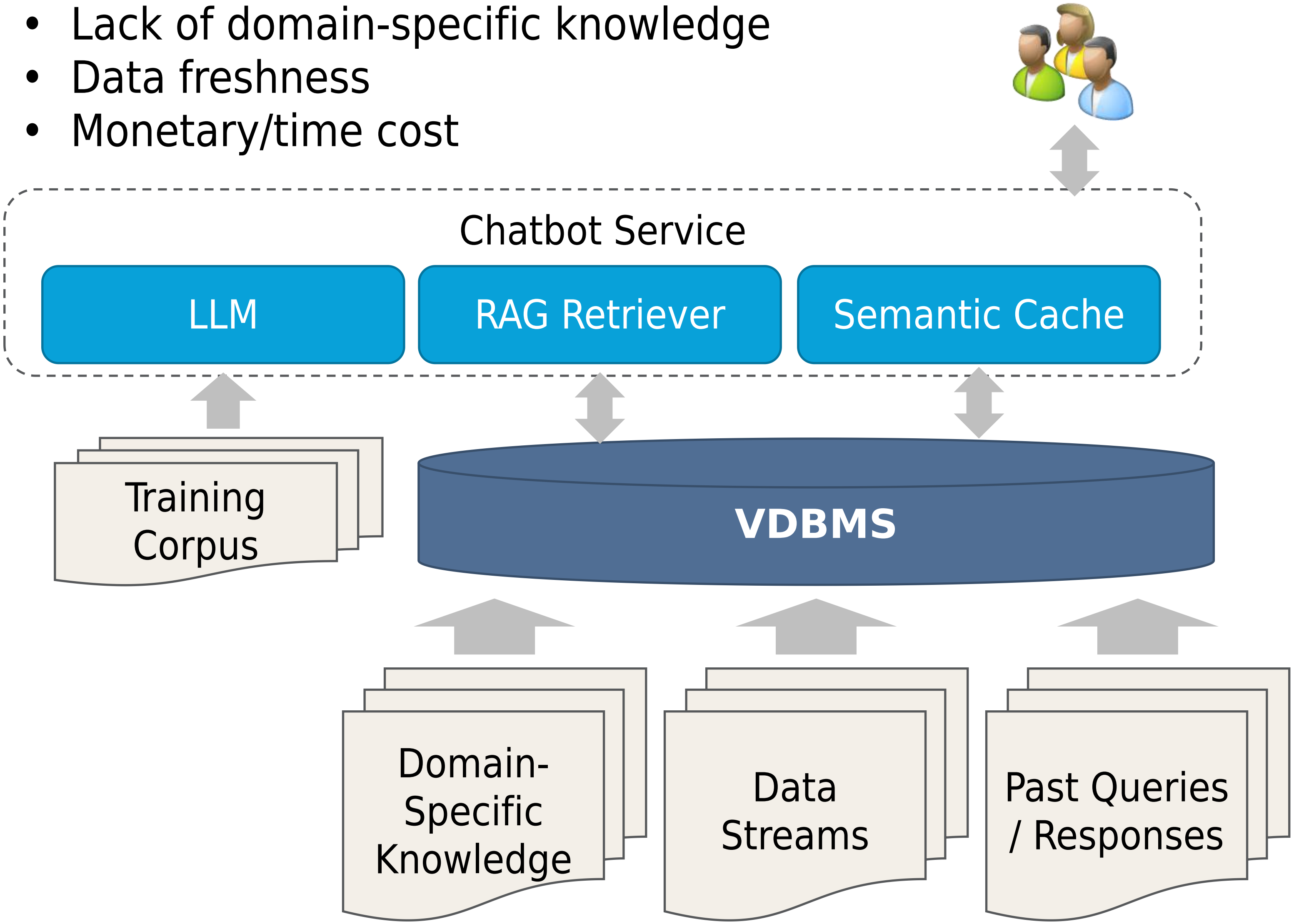


Store embeddings in a database and retrieve desired embeddings for whatever downstream task

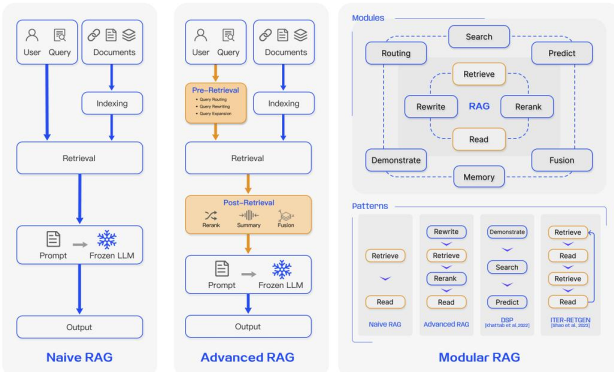
- Capabilities: similarity-based top-k/range retrieval, hybrid attribute-vector retrieval, multi-modal (multi-vector) retrieval
- Characteristics: read/write latency/throughput, retrieval accuracy, scalability, availability, consistency, fault tolerance, privacy & security, elasticity

Example: LLMs + VDBMS

- Lack of domain-specific knowledge
- Data freshness
- Monetary/time cost



Some of Today's Commercial Applications



LLM Retrieval-Augmented Generation (RAG)

https://arxiv.org/abs/2312.10997

E-Commerce & Recommendation Systems

Writing Assistant

News Classification

Photo/Video Search & Deduplication

Threat Detection

Why is Building a VDBMS Hard?

Embeddings are...

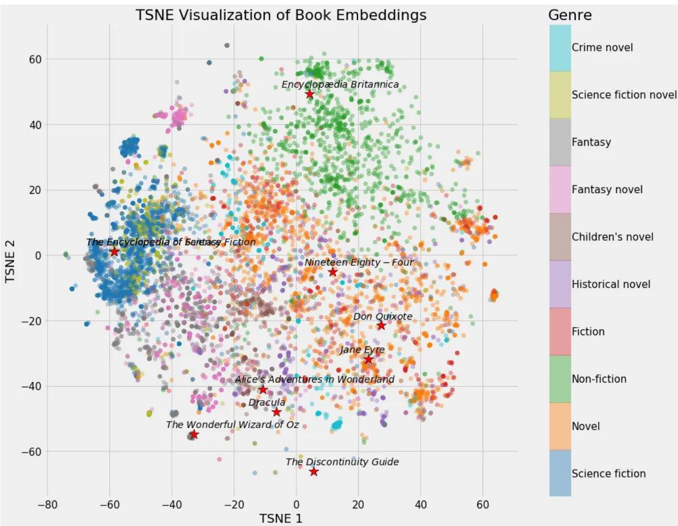
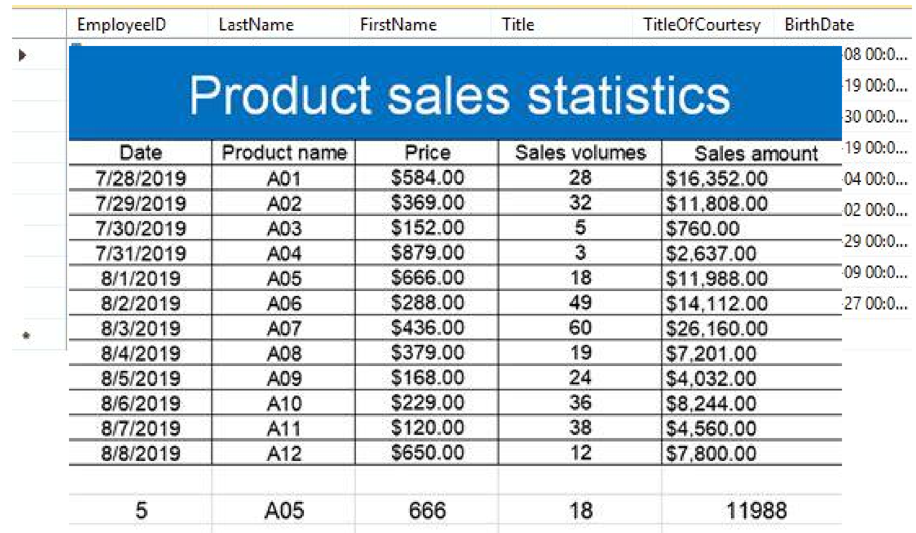


Figure: Will Koehrsen

VS



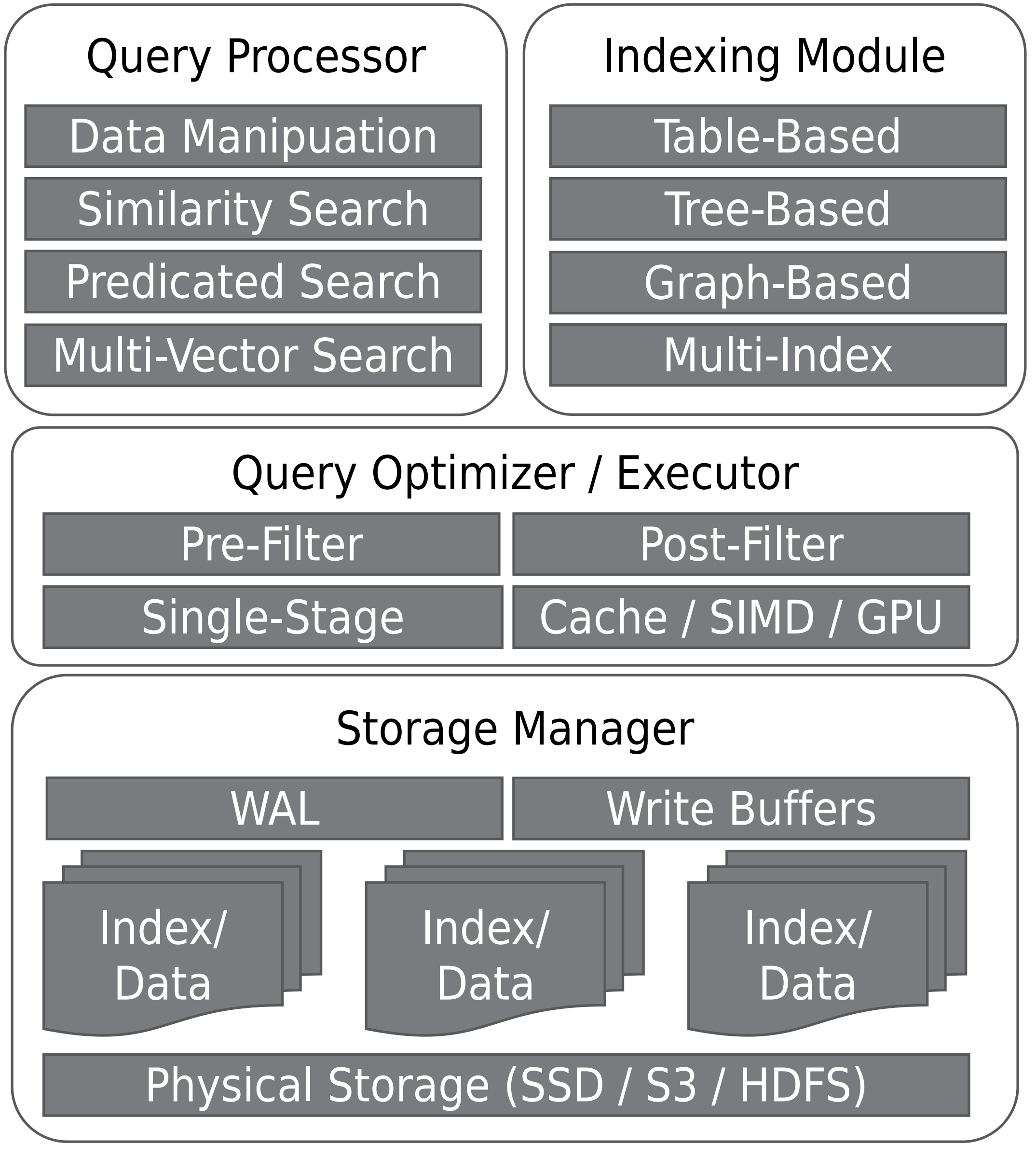
Product sales statistics

Date	Product name	Price	Sales volumes	Sales amount
7/28/2019	A01	$584.00	28	$16,352.00
7/29/2019	A02	$369.00	32	$11,808.00
7/30/2019	A03	$152.00	5	$760.00
7/31/2019	A04	$879.00	3	$2,637.00
8/1/2019	A05	$666.00	18	$11,988.00
8/2/2019	A06	$288.00	49	$14,112.00
8/3/2019	A07	$436.00	60	$26,160.00
8/4/2019	A08	$379.00	19	$7,201.00
8/5/2019	A09	$168.00	24	$4,032.00
8/6/2019	A10	$229.00	36	$8,244.00
8/7/2019	A11	$120.00	38	$4,560.00
8/8/2019	A12	$650.00	12	$7,800.00
5	A05	666	18	11988

- Huge (1024 x float64) → costly to move, clog storage
- Hard to retrieve without ambiguity
- Hard to index
- Costly to compare
- Hard to index together with attributes

similar to

Part 1: VDBMS Techniques



Overview of Query Processing

Query Processor

- Data Manipuation
- Similarity Search
- Predicated Search
- Multi-Vector Search

- High dimensionality
- Large data volume
- Low latency
- High accuracy

Query Definition

Similarity Score

- Metrical Scores
- Non-Metrical Scores

Query Type

- Data Manipulation
- Range Search
- (c,k)-Search
- Variants

Query Interface

- API, SQL

Operators & Algorithms

Vector Operators

- In/Up/Del
- Object Embedding
- Vector Math
- Vector Projection

Search Operators

- Table Scan, Top-K
- Index-Based Operators

Search Algorithms

- Brute-Force Search
- Index-Based Search

Query Definition: Similarity Scores

Similarity Score

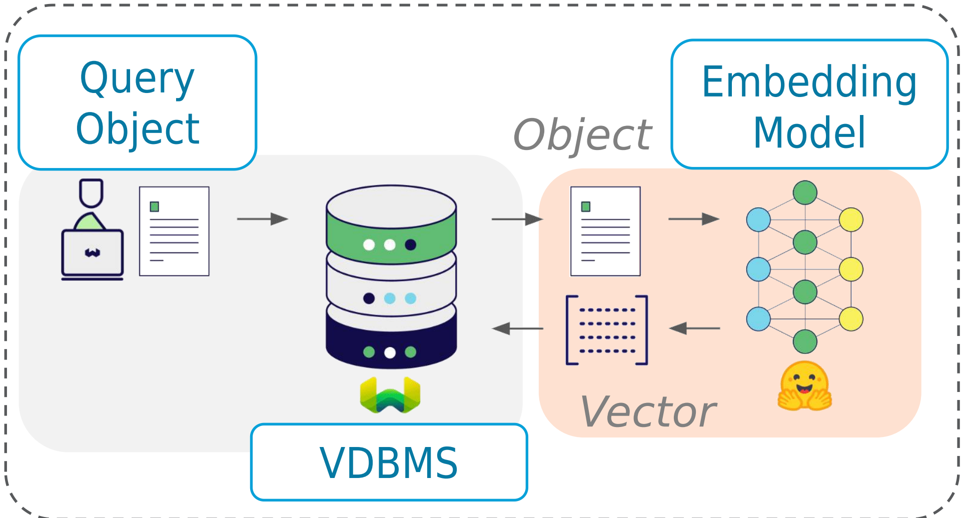
Non-Metrical Scores
- Inner Product, Cosine Similarity

Metrical Scores
- Metric axioms (id., positiv., sym., triangle ineq.)
- Hamming, Minkowski (Manhattan, Euclidean)

- A function $f: \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ indicating degree of similarity
- Similarity calculations are expensive
  - Sq. Euclidean (D=1024 floats) takes 62us on my machine (Intel i5 @ 2.3 GHz), about the same as SSD random seek

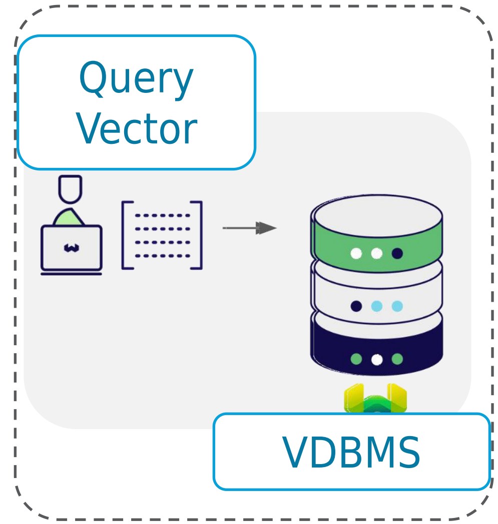
Query Types: Data Manipulation

"Indirect"



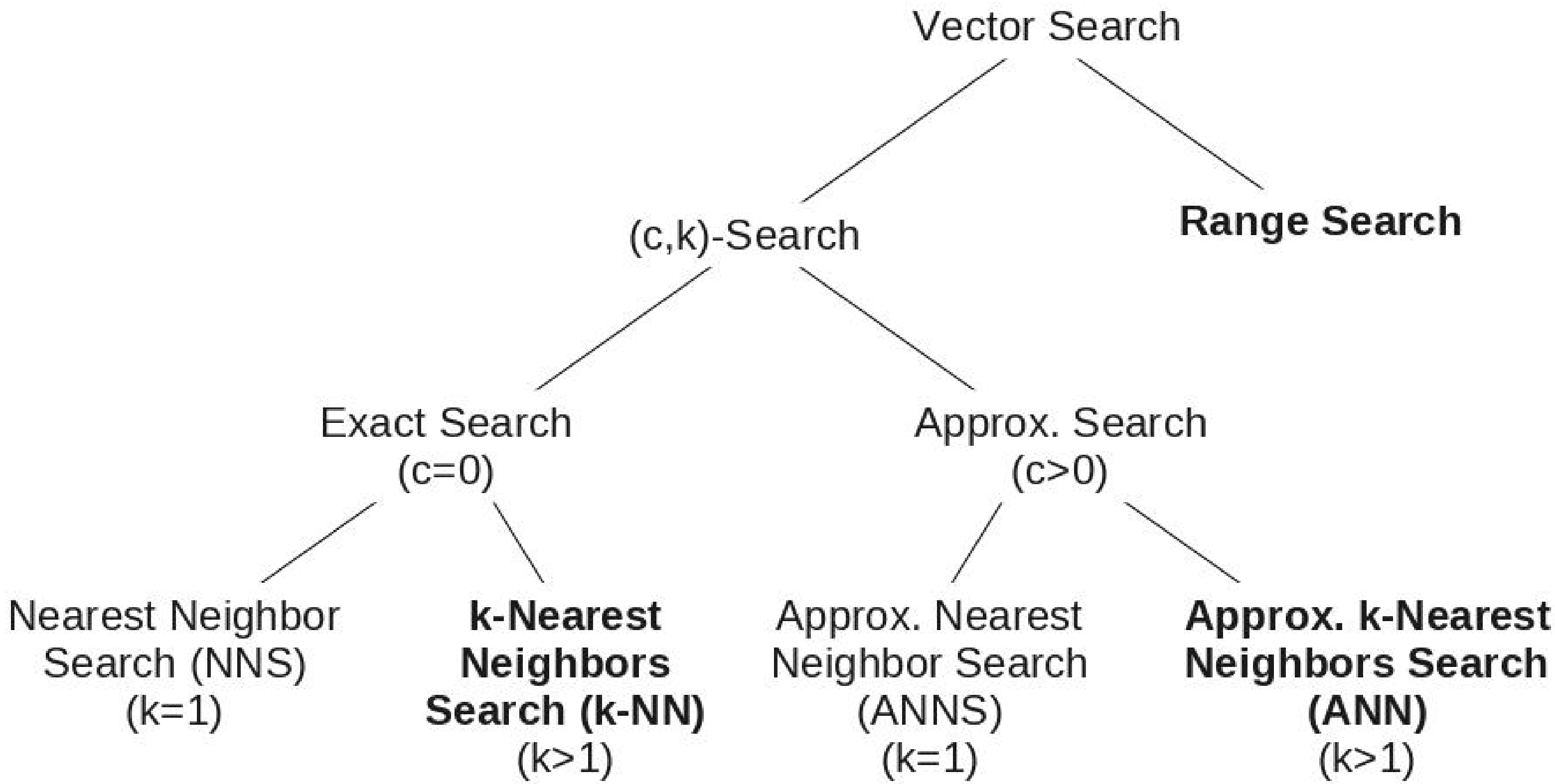
- VDBMS interacts with embedding model via plugin/add-on/extension
- More user friendly

"Direct"



- User is responsible for producing embeddings
- More controllable (e.g. custom embedding model)

Query Types: Vector Search Queries



Query Variants: predicated, batched, multi-vector

Query Interfaces

API-Based

Chroma API

count, add, get, peek, query, modify, update, upsert, delete

√ Less impedance

X Not portable

SQL-Based

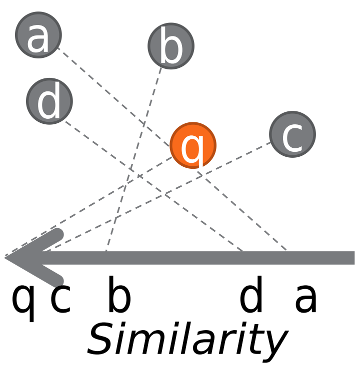
pgvector

- CREATE TABLE items (... **embedding** vector(128));
- UPDATE items SET embedding = '[1,2,3]' WHERE id = 1;
- DELETE FROM items WHERE id = 1;
- SELECT * FROM items **WHERE cat_id = 123 ORDER BY embedding <-> '[3,1,2]' LIMIT 5**;
- CREATE INDEX ON items **USING hnsw...**;

√ Potentially more portable

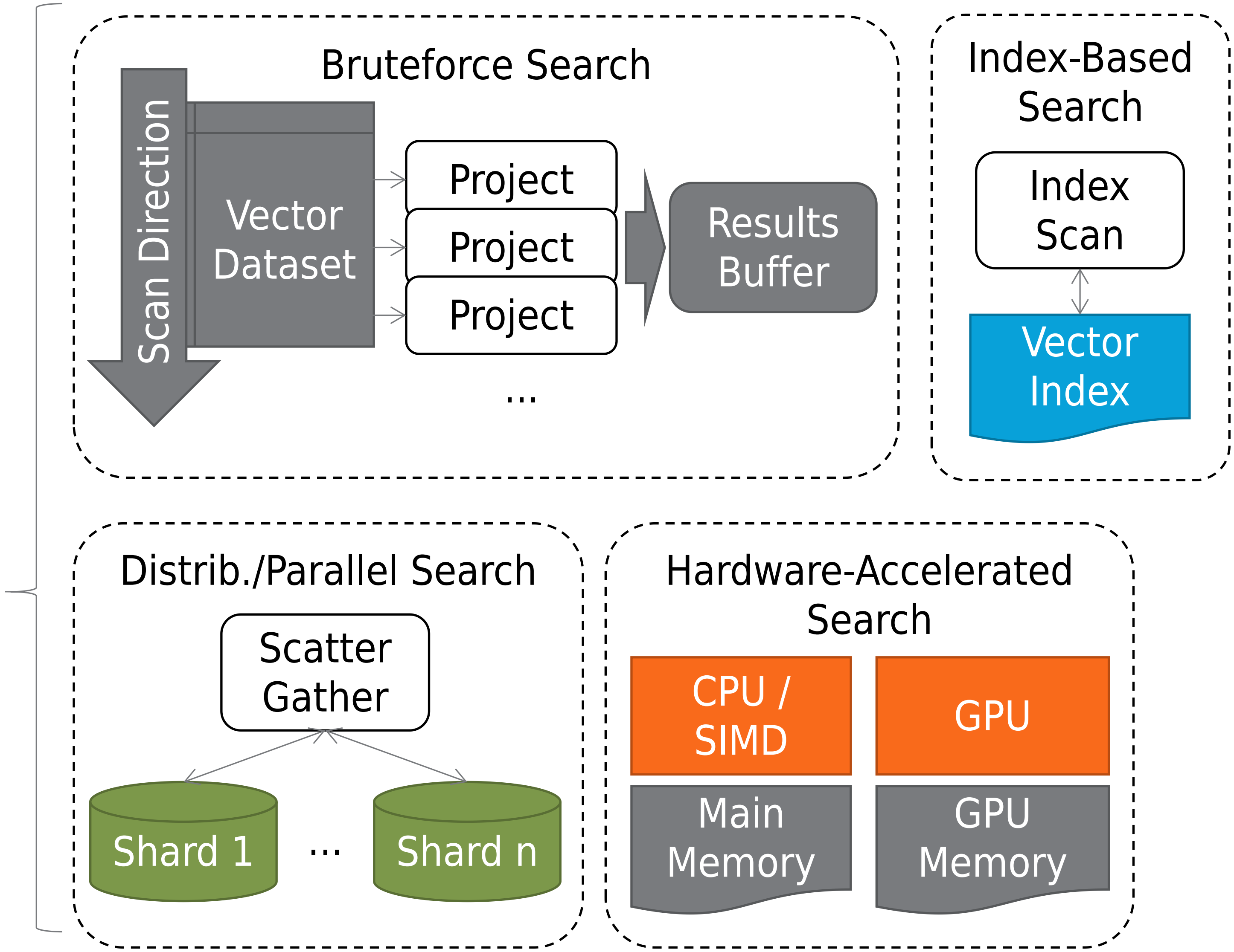
X Impedance mismatch

Operators & Algorithms



Project a vector onto similary score with respect to query vector

- O(D) for D-dimensions



Characteristics of Search Algorithms

Performance

- Amount of visited vectors, similarity comparisons

Accuracy

- Recall: (true positives) / (true positives + false negatives)
- Recall@K
  - Recall when k=K for k-NN, ANN (Li et al 2020)
  - Proportion of queries where 1-NN is ranked in first k results (Jegou et al 2011)
  - Proportion of true nearest-neighbors within the first K results of a k-NN or ANN query (K ≤ k) (RecSys)
- Precision: (true positives) / (true positives + false positives)

Challenges to Query Processing

Query Semantics
(e.g. “Amazon”)

Score Selection

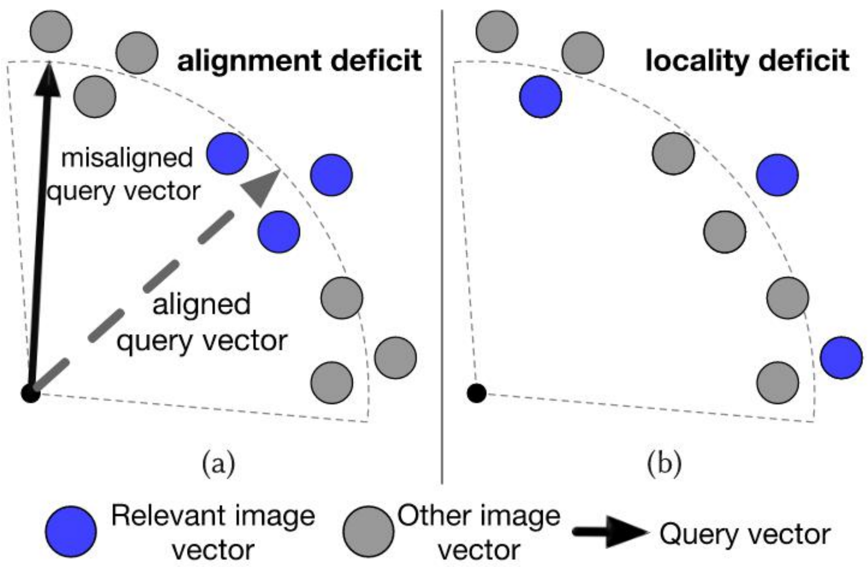


Figure: Moll et al 2024

Curse of Dimensionality

Score Design
(e.g. Metric Learning)

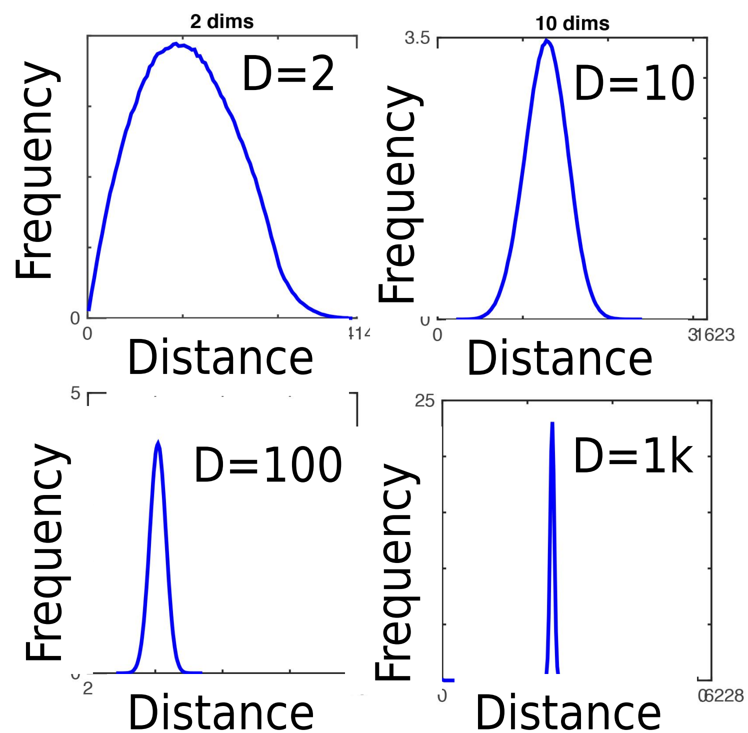


Figure: Cornell CS4780/CS5780 Lecture 2 (Fall 2018)

Overview of Storage & Indexing

Indexing Module

- Table-Based
- Tree-Based
- Graph-Based
- Multi-Index

- High dimensionality
- Large data volume
- Low latency
- High accuracy
- **Construction cost**
- **Storage cost**
- **Maintenance cost**

Construction / Search / Maint.

Construction

- Randomization
- Learned Partitions
- Navigable Partitions

Search

- Bucket Scan
- Defeatist Search
- Best-First Search

Maintenance

- Rebalancing

Logical / Physical Storage

Logical Structure

- Tables
- Trees
- Graphs

Physical Structure

- Quantization
- Disk-Resident Indexes

Table-Based Indexes

Randomized Partitioning

Locality-Sensitive Hashing
- Random Hyperplanes (E2LSH)
- Random Bits (Faiss IndexLSH)
- Random Balls (FALCONN)

Rely on probability amplification

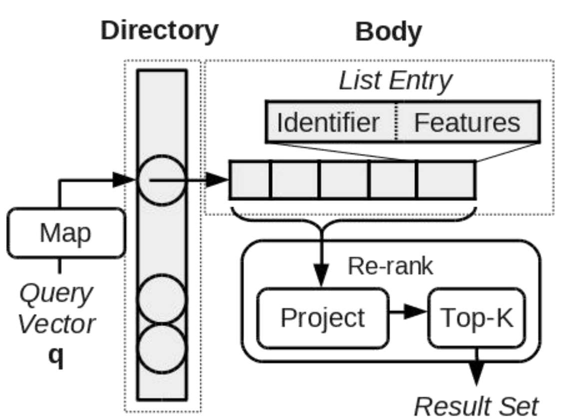
Learned Partitioning

Centroid-Based (Quantization)
- Nearest Centroid (SPANN, Faiss IVF*)
- Nearest Centroid Product (Faiss PQ)

Learned Hashing

Learn from data distribution

- Construction: O(DN);
- Search: $O(DN^c)$, $0 < c < 1$



- Data Drift

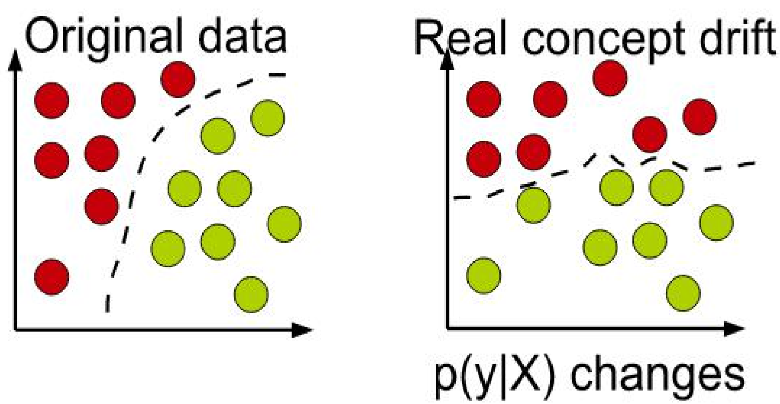
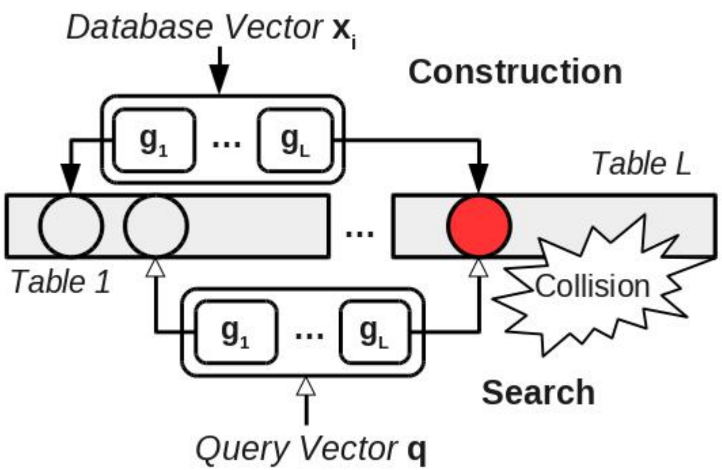
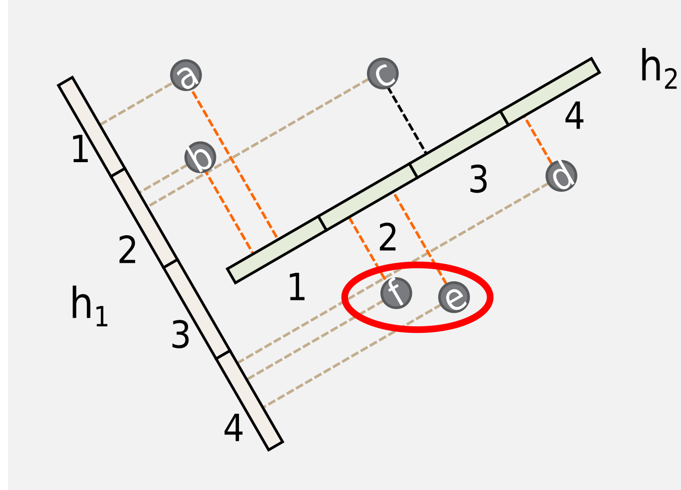


Figure: Gama et al, 2013

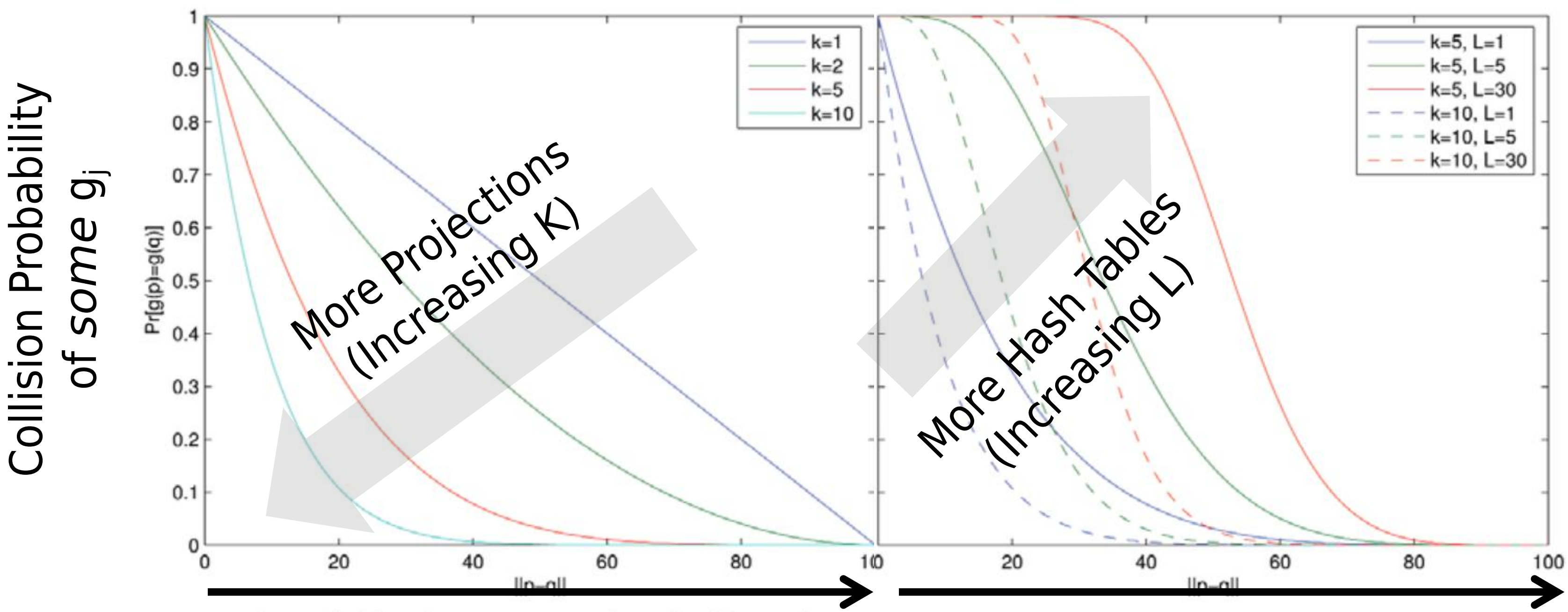
Locality-Sensitive Hashing (LSH)

e.g. E2LSH
L=2, K=4





Vec.	$g=(h_1,h_2)$
a	11
b	21
c	23
d	44
e	**42**
f	**42**



Distance

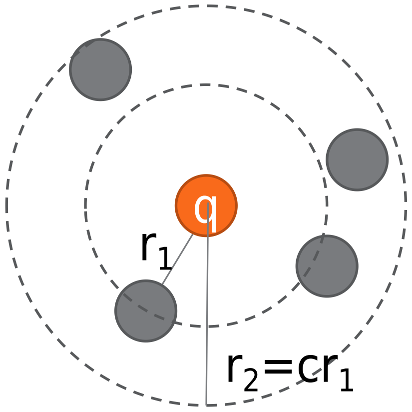
Distance

Figure: Andoni & Indyk, 2008

LSH Families

- "Hash Family": For any r_1 , r_2 , $x \in S$, and q:
  - if $d(x, q) \le r_1$, then collision prob. $\ge p_1$ Large is Better
  - if $d(x, q) \ge r_2$, then collision prob. $\le p_2$ Small is Better
- Typically storage ~ $O(DN^{1+\rho})$, search ~ $O(DN^{\rho})$ where

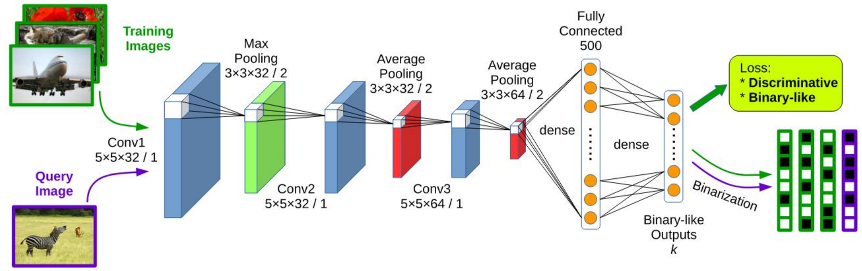
$$\rho = \frac{log(1/p_1)}{log(1/p_2)}$$



- Interesting families:
  - Hamming (Faiss IndexLSH) $\rho=1/c$
  - Random hyperplans (E2LSH) $\rho=1/c$
  - Spherical LSH (FALCONN) $\rho=1/(2c^2-1)$

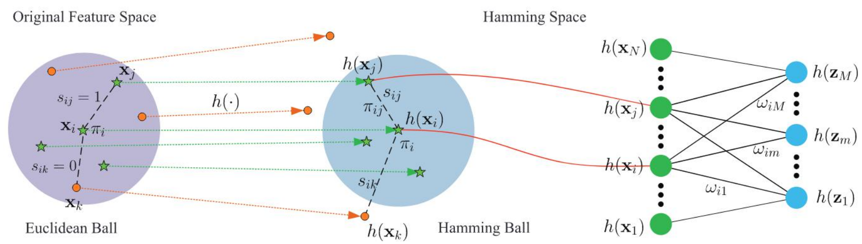
Learning to Hash (L2H)

Pairwise Similarity - Preserving



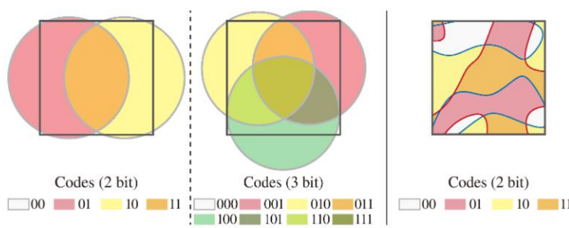
Deep Supervised Hashing, Liu et al 2016

Multiwise Similarity-Preserving



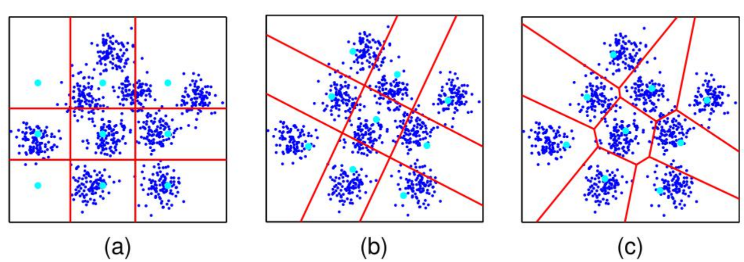
kNN Hashing, Ding et al 2015

Implicit Similarity-Preserving



Spherical Hashing, Heo et al 2012

Reconstruction Error-Minimizing



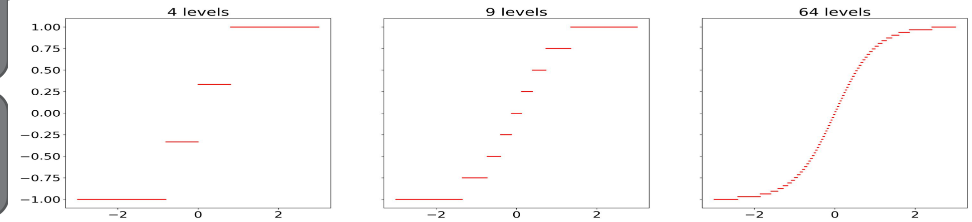
Quantization, Wang et al 2018

Quantization

Level Quantization

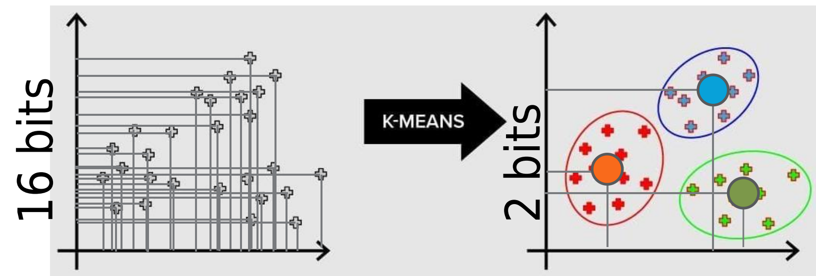
- Uniform (e.g. Faiss SQ)
- Non-Uniform

2 bits | 4 bits | 6 bits



Learned Quantization

- k-Means (e.g. Faiss IVFSQ)
- Product (e.g. Faiss PQ)

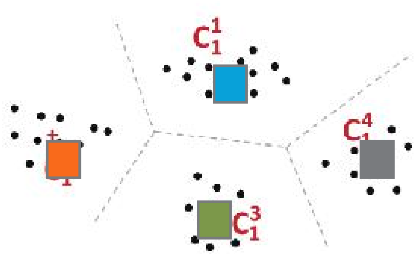


16 bits | 2 bits

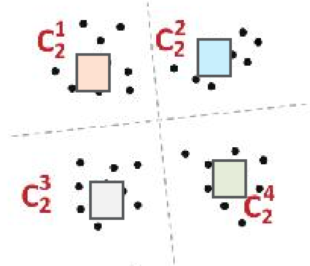
E.g. 4-Dimensional Vectors m=2, k=4

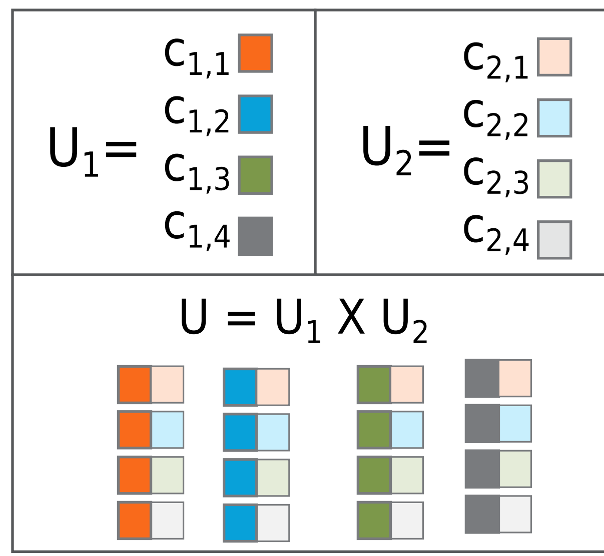
Dims [1, 2] | Dims [3, 4]

subspace 1

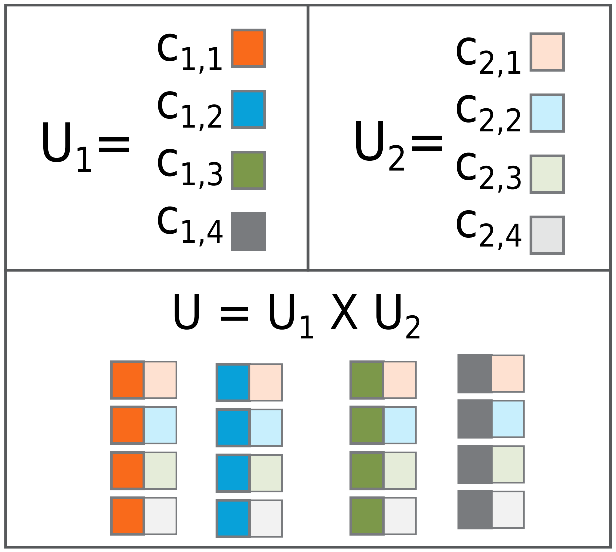


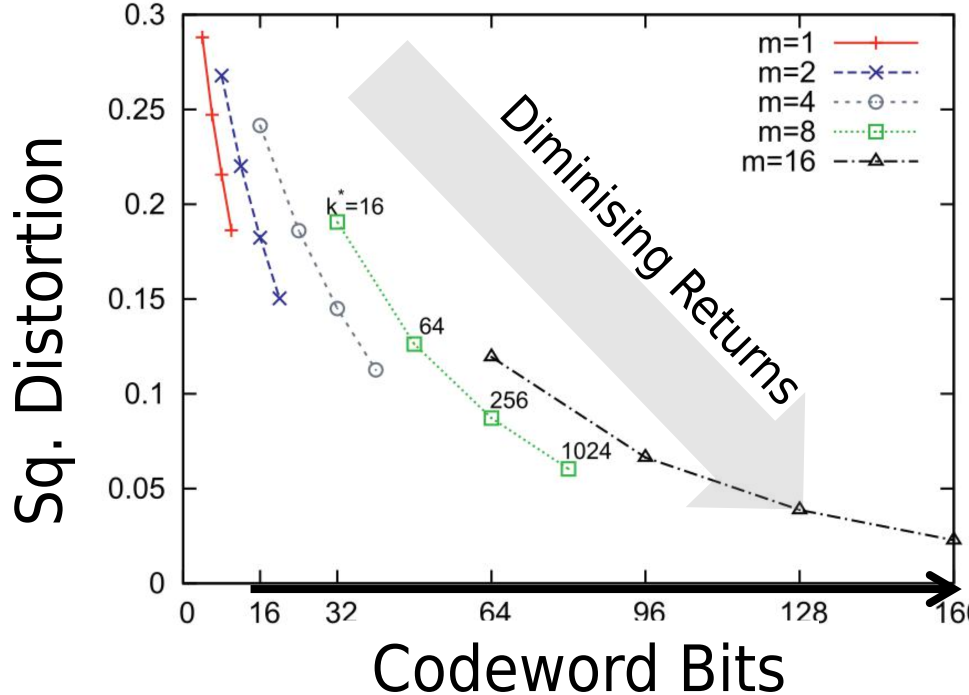
subspace 2





Product Quantization

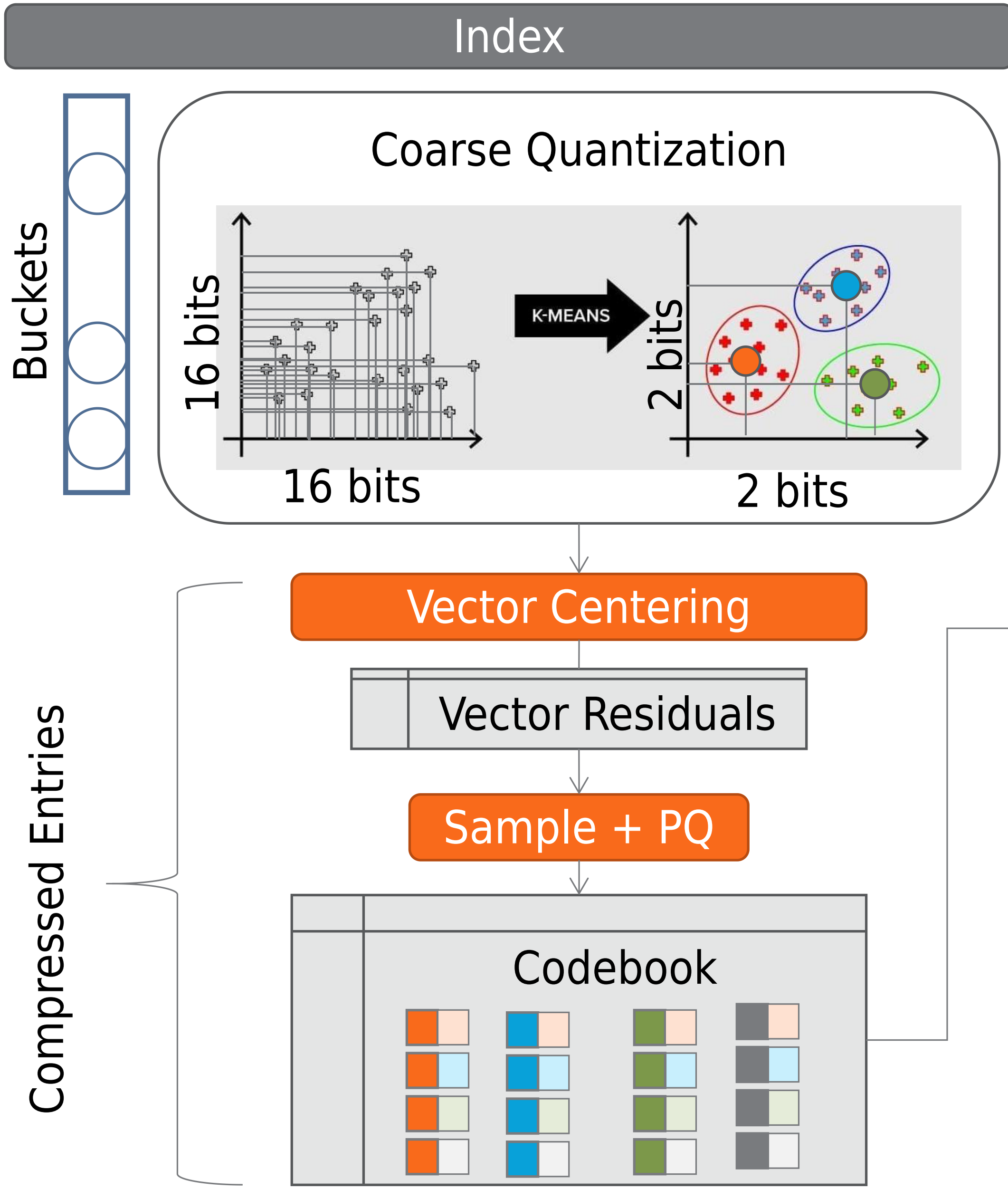


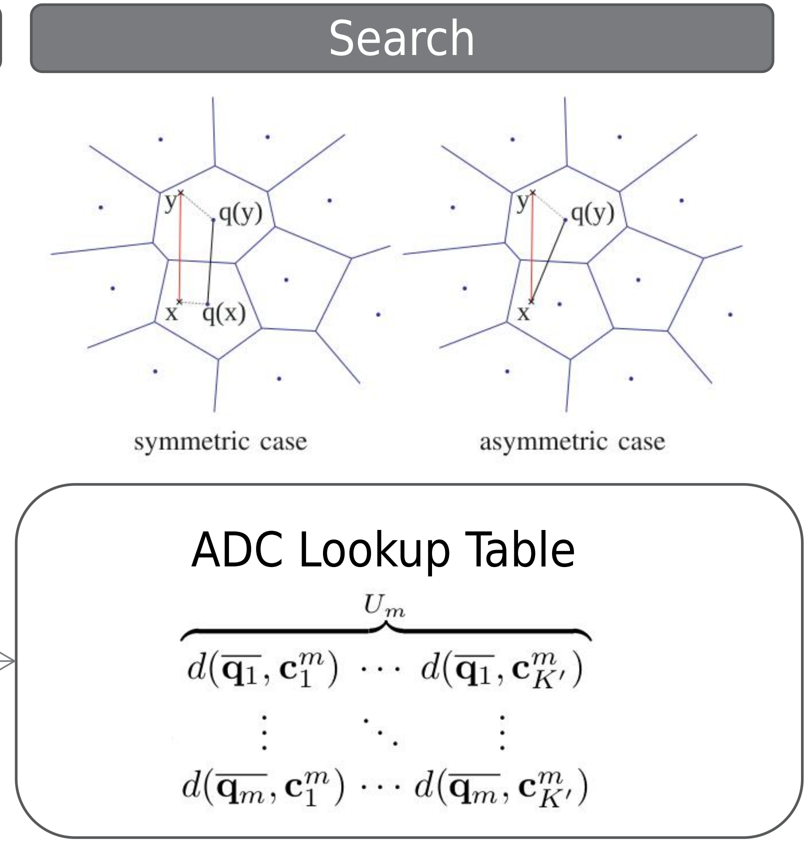


- Preserves dimensions, e.g. $R^4 = R^2 \times R^2$
- Faster training
  - 4 centroids per subspace = 16 total codes
  - k-means O(DN*k): 2(2N*4) = 16N vs 4N*16 = 64N

Figure: Jegou et al 2011

“IVFADC” Asymmetric Distance Comp.

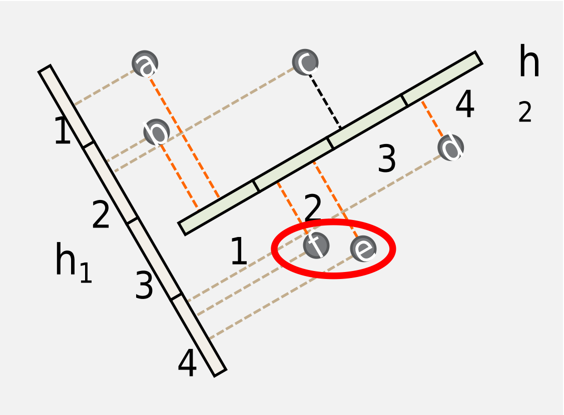




- Bucketing for fast search
- Lookup cache
- Sampling from residuals for faster training

Summary of Table-Based Indexes

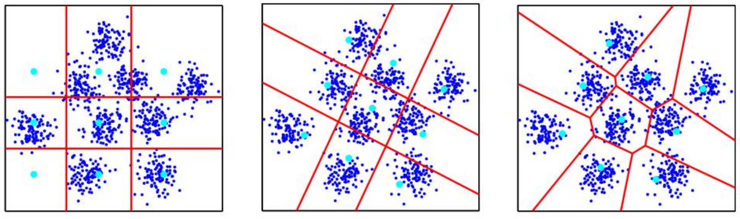
Randomization-Based



E.g. Faiss IndexLSH, E2LSH, FALCONN

√ Theoretical guarantees
√ No rebalancing
X High storage costs

Learning-Based



(a) (b) (c)

E.g. L2H, SQ, PQ, IVFADC

√ Low storage costs
√ Low latency
X Susceptible to data drift

Table-Based Indexes: Discussion

Advantages

√ Disk-friendly, E.g. LSH, SPANN

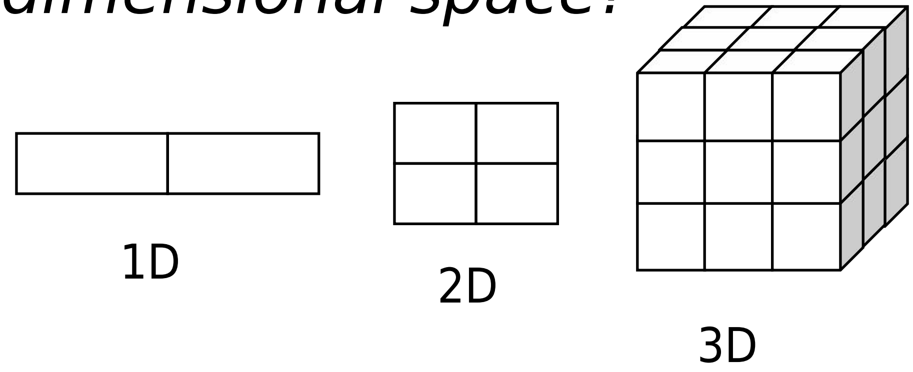
√ Readily supports in-distribution insert/update/delete

√ Easier to derive error bounds

Disadvantages

X Hard to deal with queries near borders/corners

- *How many buckets are adjacent to a corner in a D-dimensional space?*



Tree-Based Indexes

Splitting Planes

Splitting Points

Axis-Aligned
- k-d Tree
- PKD-tree (Principal Components)
- FLANN

Randomized
- RPTree
- ANNOY

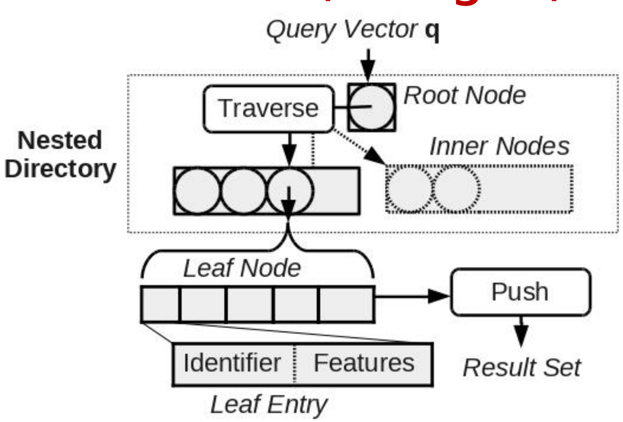
Fixed
- Medians
- Overlapping

Recursive space partitioning

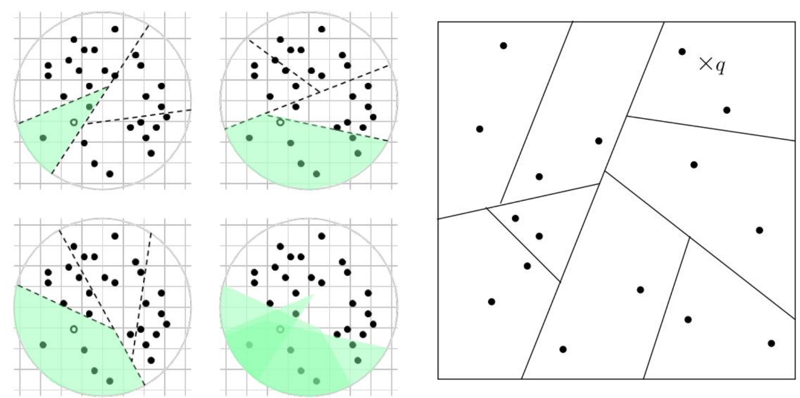
Rely on probability amplification

Balanced partitioning

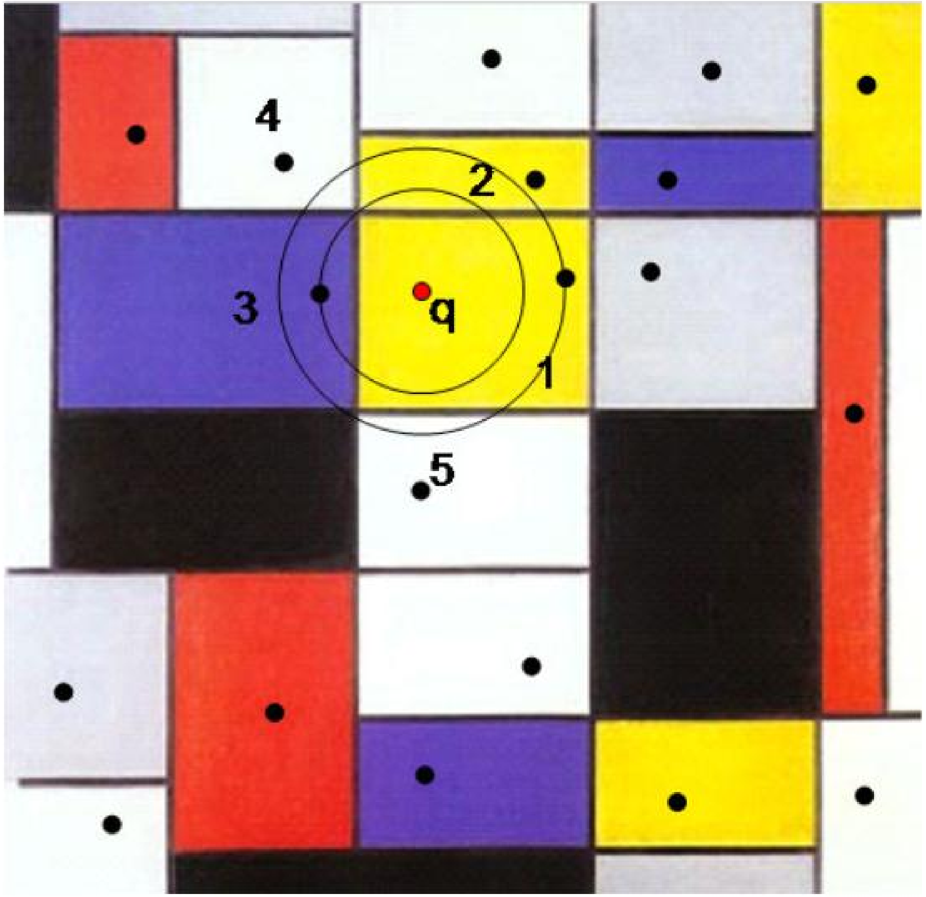
- Construction: O(DN log N)
- Search: O(D log N)



- Discernability



k-d Tree

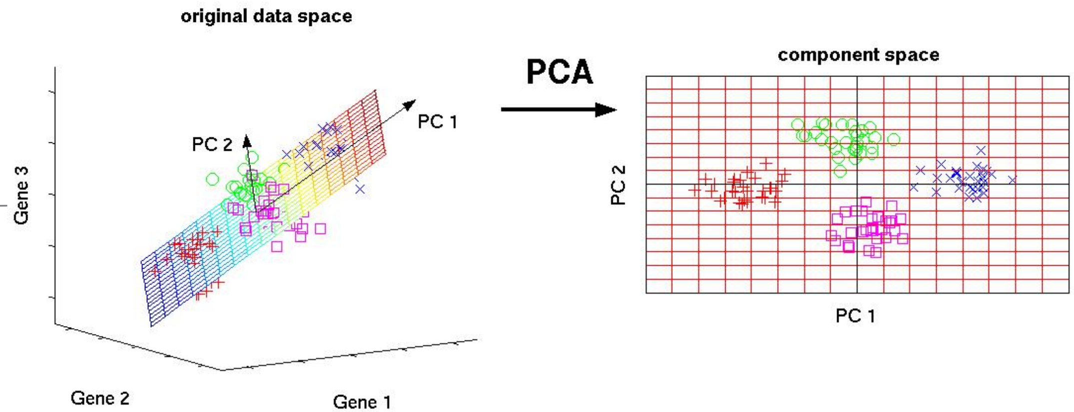


- $O(DN^{1-1/D})$ search, $O(DN \log N)$ construction
- Tends toward $O(DN)$ as D grows

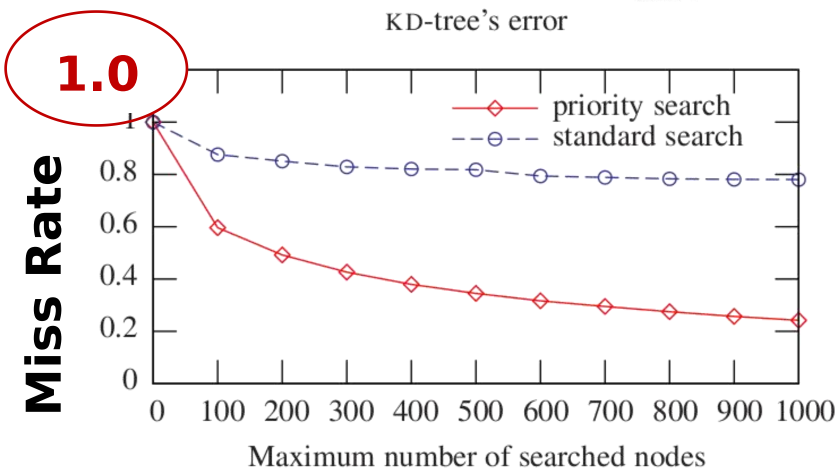
Figure: Silpa-Anan and Hartley, 2008

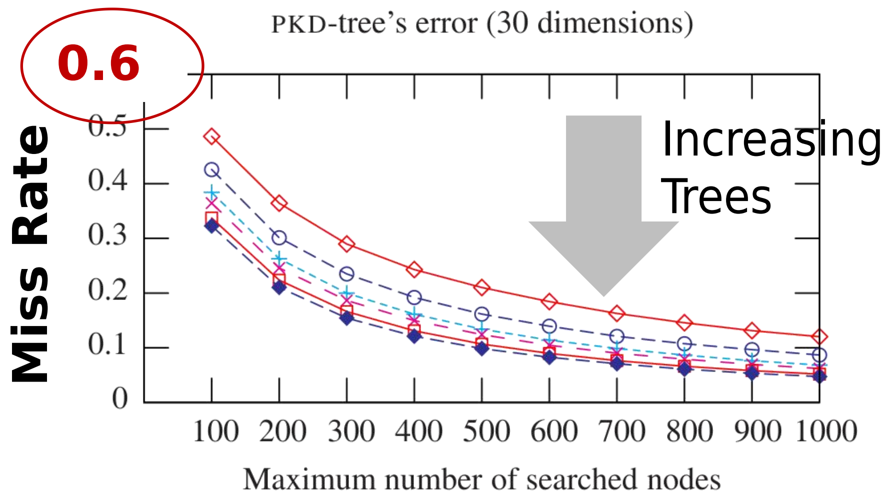
Principal Component Trees

Vector Rotation



k-d Tree Construction



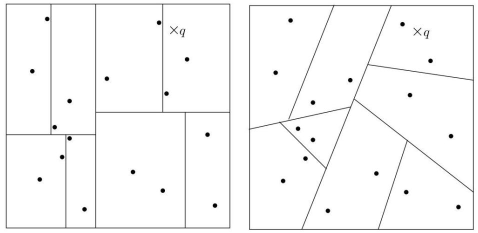


- More discernability by aligning to principal components
- No huge gains from multiple trees

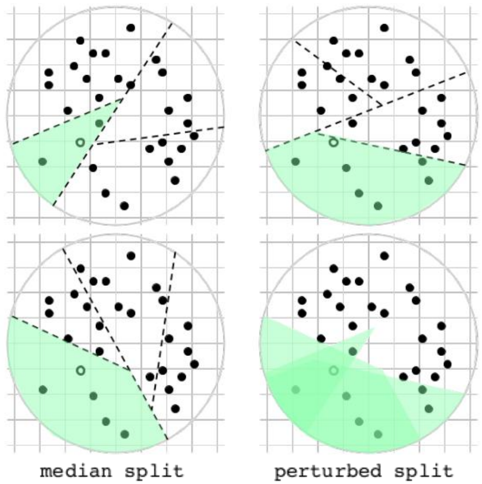
Figure: Silpa-Anan and Hartley, 2008 (bottom)

Random Projection Trees

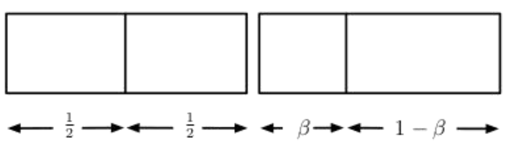
Random Projections



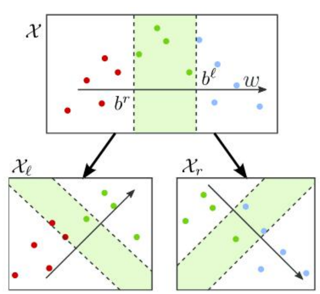
Random Rotations

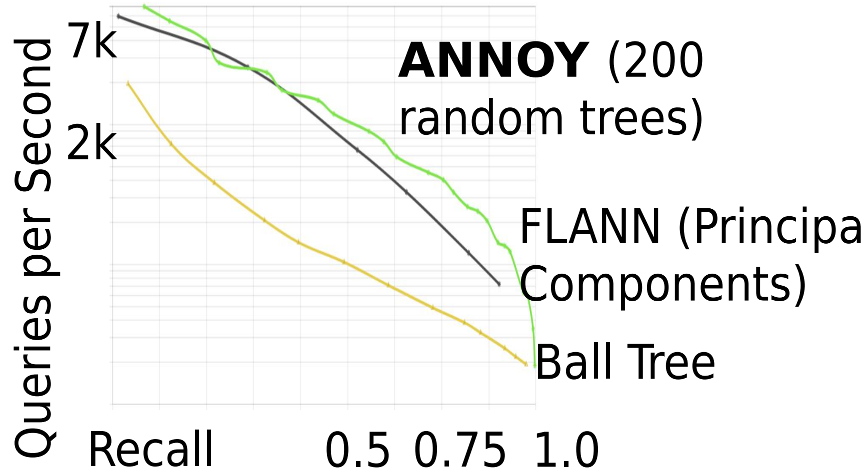


Random Splits

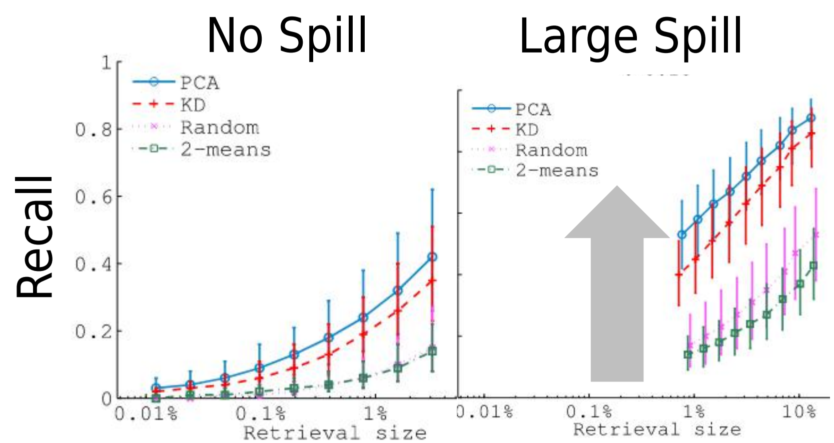


Overlapping Splits





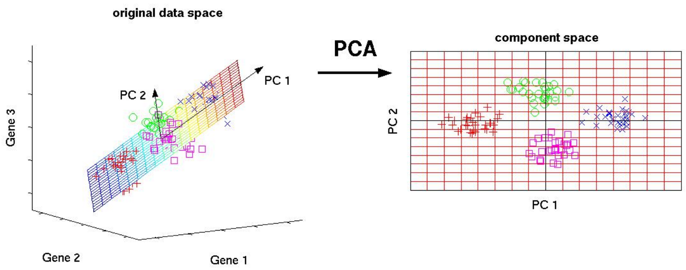
Source: ann-benchmarks.com



Figures: Dasgupta & Sinha 2013, Dasgupta & Freund 2008, Ram & Sinha 2019, McFee & Lanckriet 2011

Summary of Tree-Based Indexes

Axis-Aligned

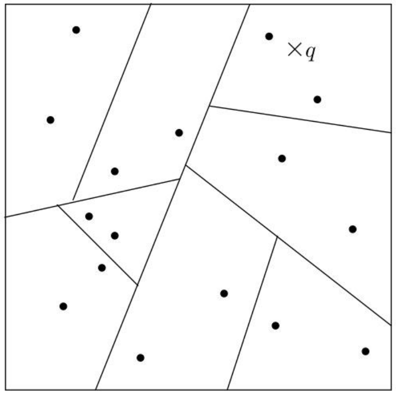


E.g. k-d Tree, PKD-Tree, FLANN

√ High recall for low dims.

X Inflexible

Randomized



E.g. ANNOY, RPTree

√ High recall for high dims.

X High storage (forests)

Tree-Based Indexes: Discussion

Advantages

√ Disk-friendly in principle (store together by leaf)

√ O(D log N) defeatist search

√ Supports in-distribution insert/update/delete

Disadvantages

X Hard to keep balanced following data drift

X Low recall for queries near leaf borders / corners

Graph-Based Indexes

Navigable Partitioning

k-Nearest Neighbor Graphs (kNNGs)
- KGraph
- EFANNA

Directly index the nearest neighbors

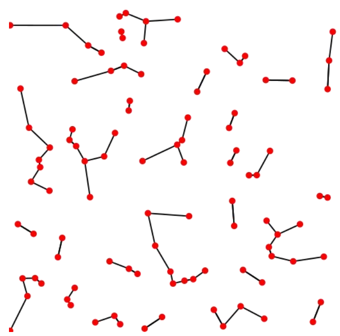
Monotonic Search Networks (MSNs)
- FANNG
- NSG
- Vamana

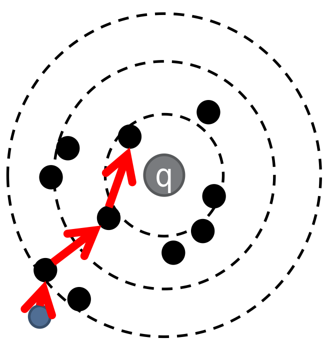
Support greedy depth-first search

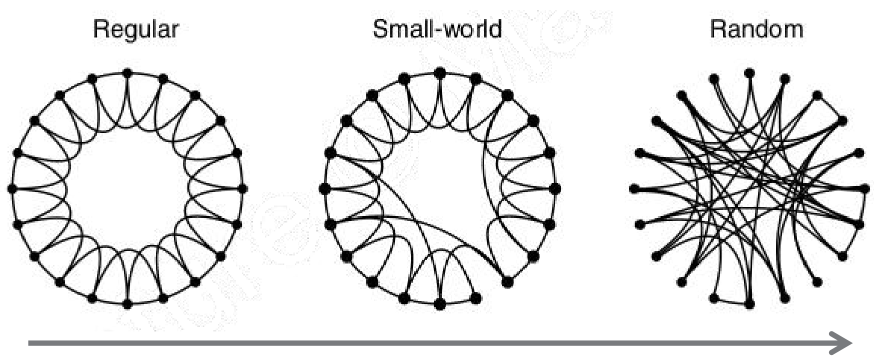
Small-World Graphs
- NSW
- HNSW

Exploit navigational small-world properties

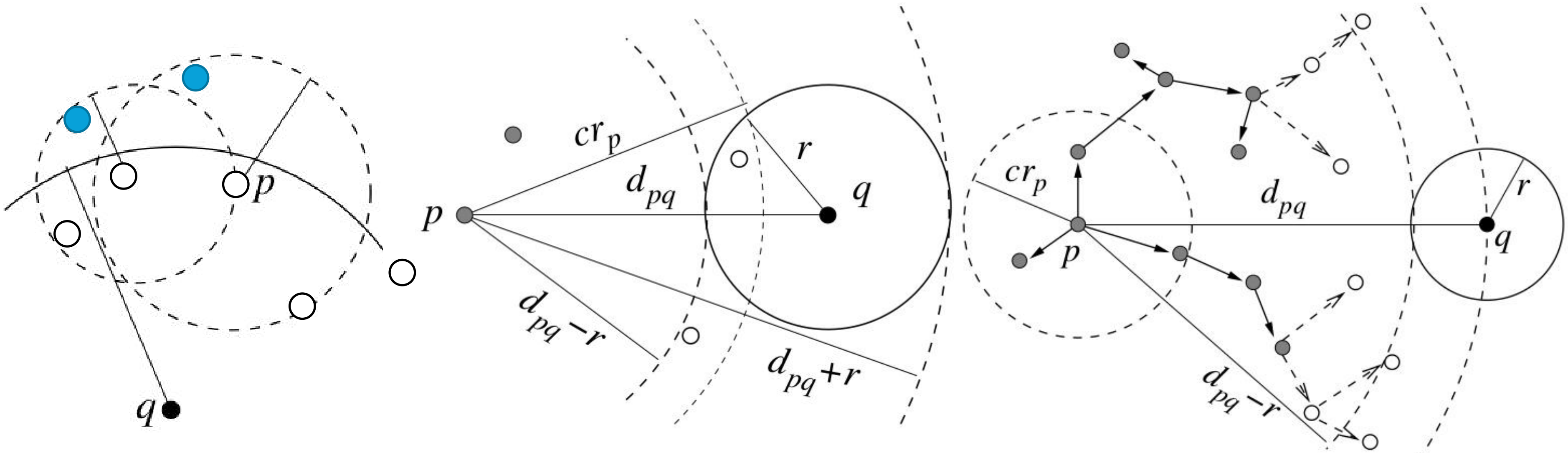
- Construction: ranges from $O(DN \log N)$ to $O(DN^{1+c})$
- Search: $O(D \log N)$ or $O(DN^c)$







k-Nearest Neighbor Graphs (kNNGs)



A. Sampled point inside query ball → check its neighbors

B. Point ball intersects query ball → check neighbors in the overlap

C. Point ball outside query ball → prune near neighbors of p

- O(1) search for queries in dataset, else O(DN^{1+c}) via sample and prune (see above)

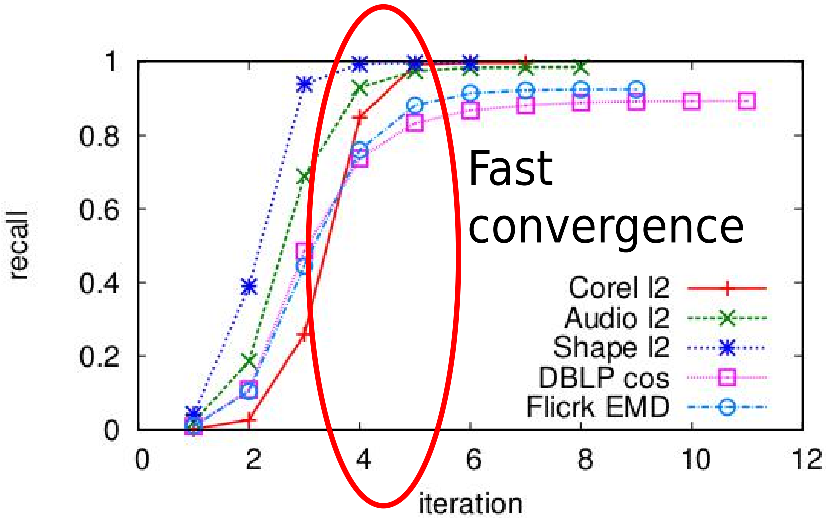
Construction

- Exact: O(DN^2)

Figure: Paredes & Chavez, 2005

KGraph (NNDescent)

“A neighbor of a neighbor is likely to also be a neighbor”



Dataset & Measure	Empirical Complexity
Corel/l_2	$O(n^{1.11})$
Audio/l_2	$O(n^{1.14})$
Shape/l_2	$O(n^{1.11})$
DBLP/cos	$O(n^{1.11})$
Flickr/EMD	$O(n^{1.14})$

Construction

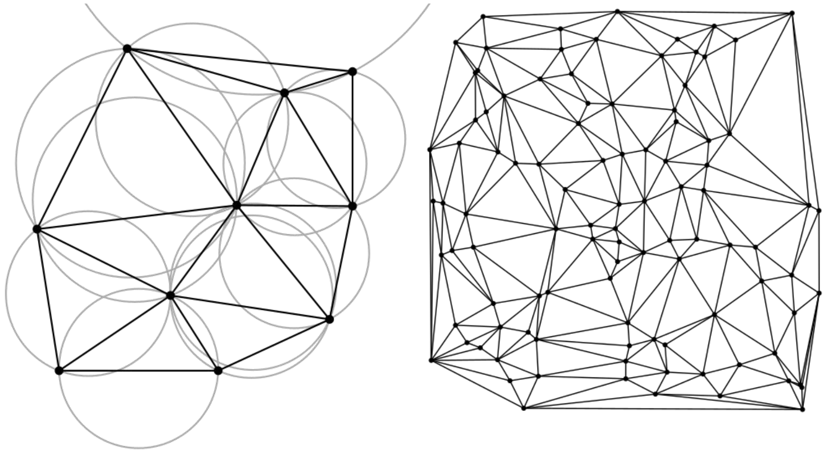
- Initialize random kNNG
- For each node, change any 2-hop neighbor into a 1-hop neighbor if it is a new k-nearest neigbor
- Repeat until convergence

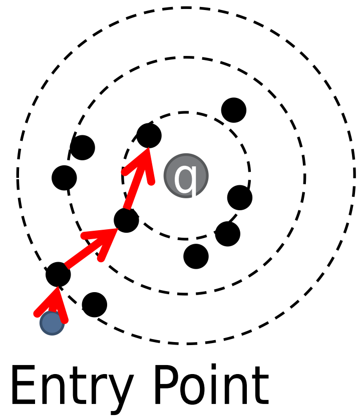
Figure: Dong et al, 2011

Monotonic Search Networks (MSNs)

"Greedy search is all you need"

Figures:
Delaunay
triangulation
(Wikipedia)

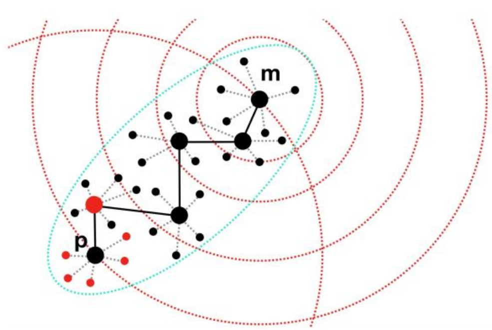




Random Search Trials

- Probe the graph by conducting searches from a random entry point to a random query point in the dataset, e.g. FANNG

Fixed Search Trials



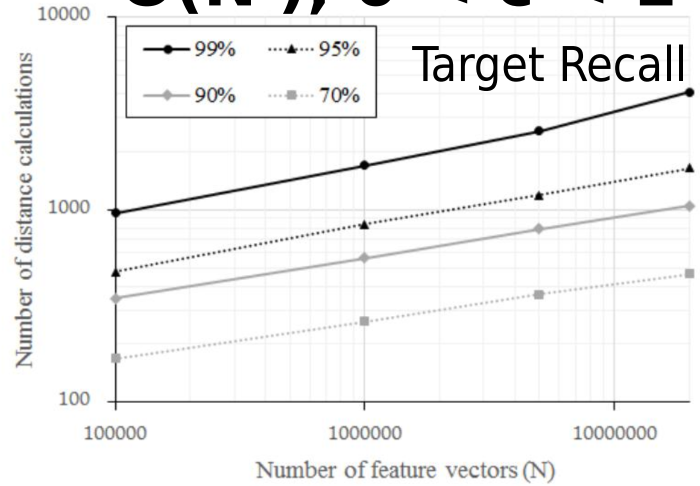
- Designate a point as the sole entry point for all search trials, e.g. NSG, Vamana

Fast ANN Graph (FANNG)

Search Latency after 50N Random Search Trials

$O(N^c)$, 0 < c < 1

Distance Calculations (Log Scale)



Dataset Size (Log Scale)

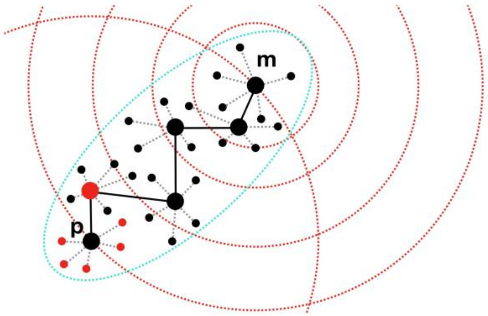
Construction

- O(N) trials, each trial $O(N^c)$ to yield $O(N^{1+c})$
- Occlusion rule prunes redundant edges to limit out-degrees

Figure: Harwood & Drummond, 2016

Navigating Spreading-Out Graph (NSG)

- Single source makes it easier to establish monotonic search paths from this node to all other nodes
- Spanning tree ensures connectivity



Construction

- $O(N^{1+c} \log N^c)$

Search

- ~O(log n) due to higher quality neighborhoods

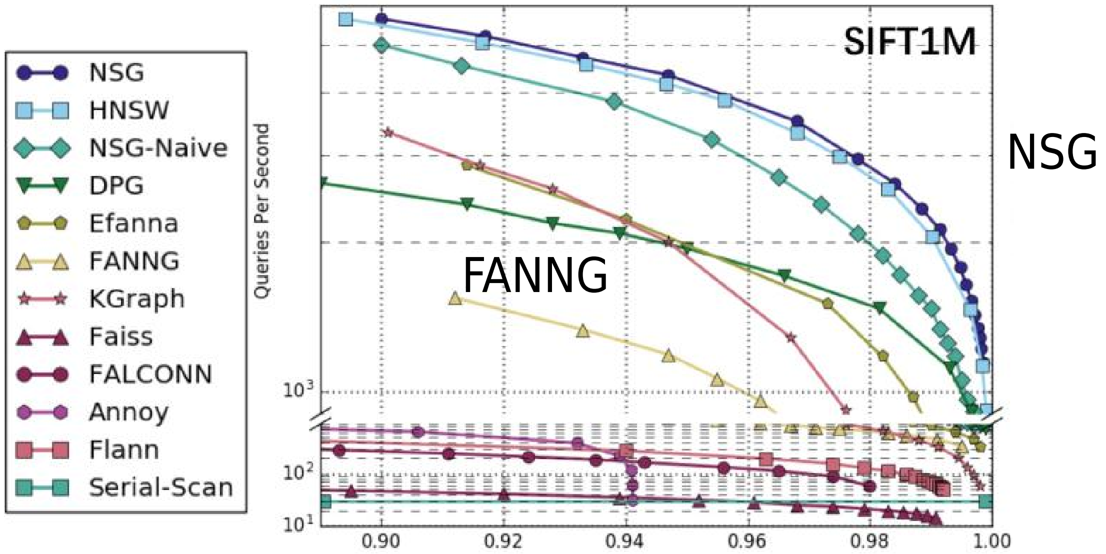
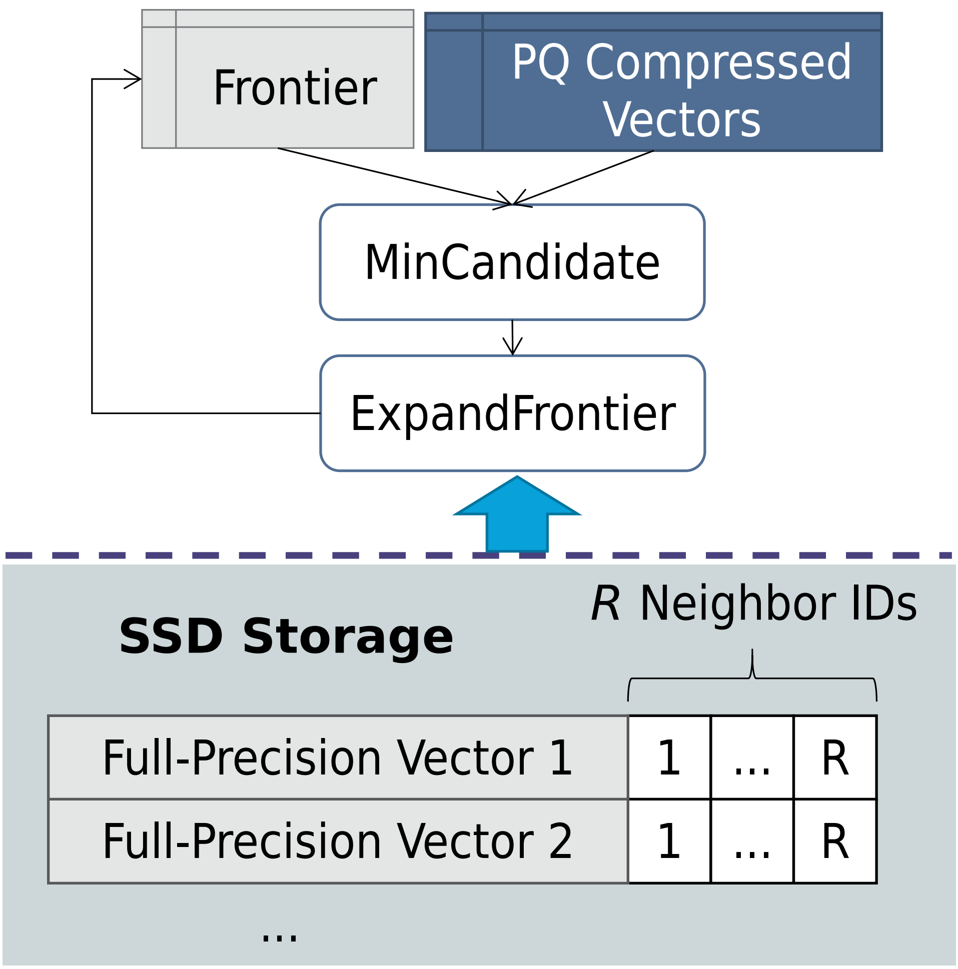


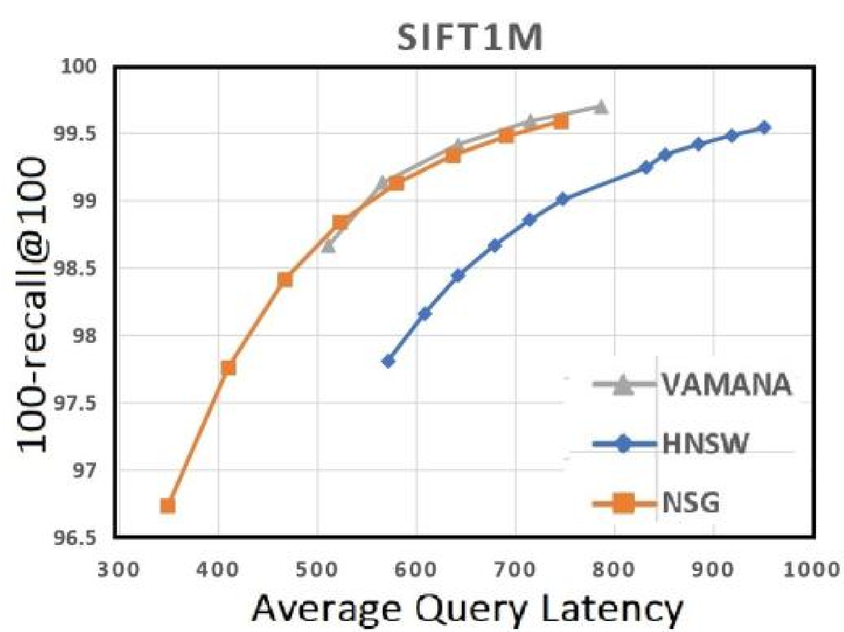
Figure: Fu et al, 2019

Vamana/DiskANN

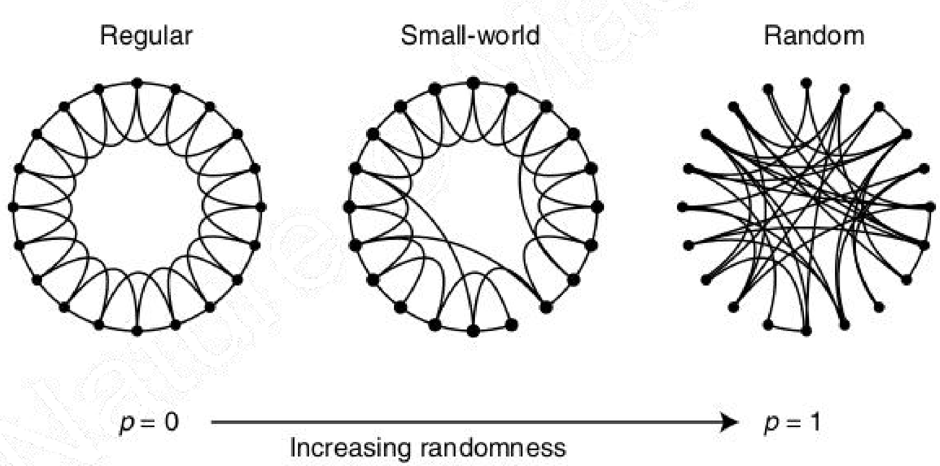
VamanaIndexScan

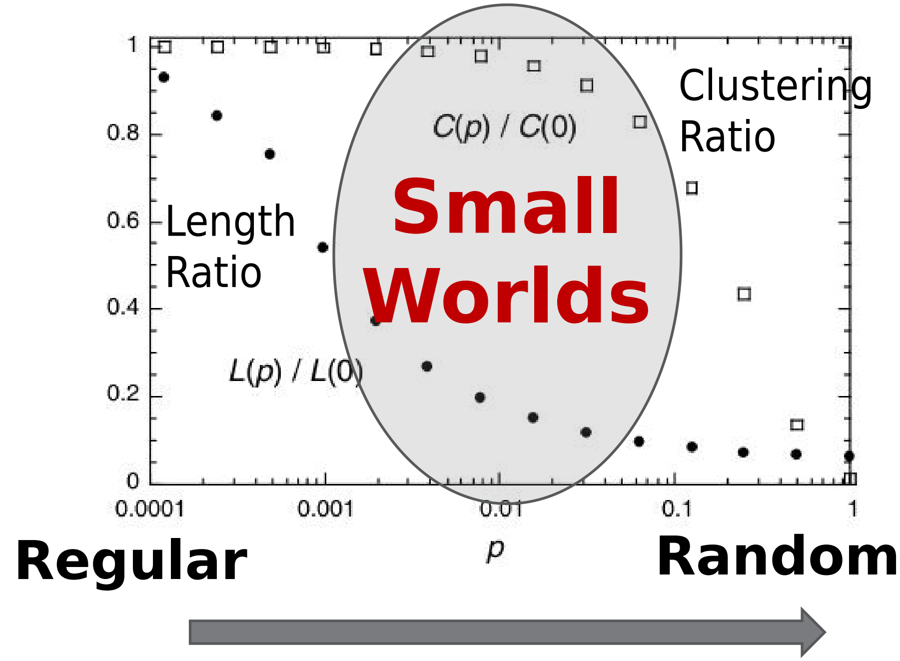


- Similar to NSG
- On-disk neighborhoods
- Edge traversal performed in memory using compressed vectors



Small-World Graphs





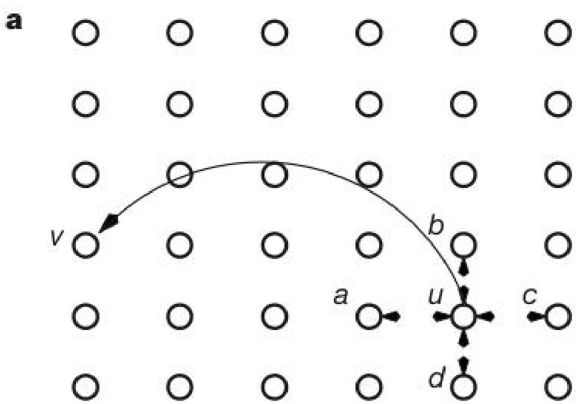
Probability of Random Rewiring

- Small characteristic path lengths (short shortest-paths)
- High clustering (friend of a friend is also my friend)

Figure: Watts and Strogatz, Nature 1998

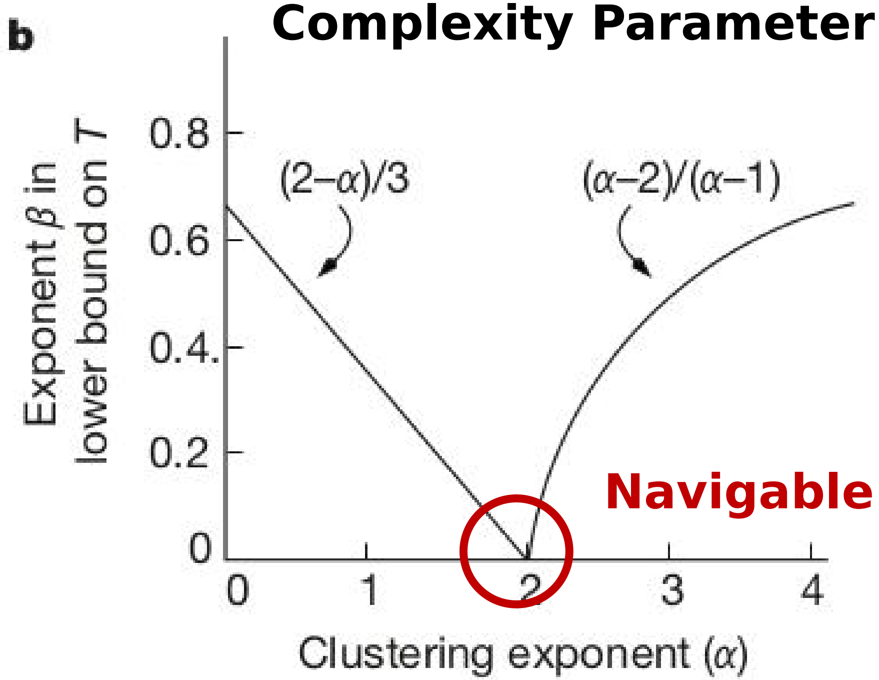
Navigable Small-World Graphs

Random edge probability α



Randomly add an edge from u to v with probability $|u,v|_1^{-\alpha}$

$O(N^{\beta})$ Greedy Search Complexity Parameter β



- Not all small-world graphs permit O(log N) greedy search
- (In Kleinberg's graph, only α=2 yields a navigable graph)

Figure: Kleinberg, Nature 2000

Hierarchical Navigable Small-World Graphs (HNSW)

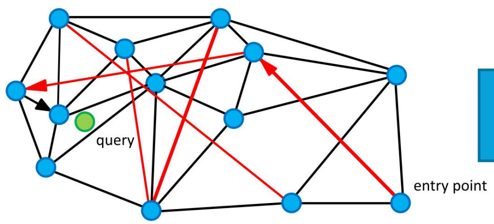


Fig. 1. Graph representation of the structure. Circles (vertices) are the data in metric space, black edges are the approximation of the Delaunay graph, and red edges are long range links for logarithmic scaling. Arrows show a sample path of the greedy algorithm from the entry point to the query (shown green). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

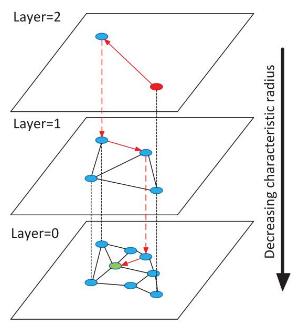
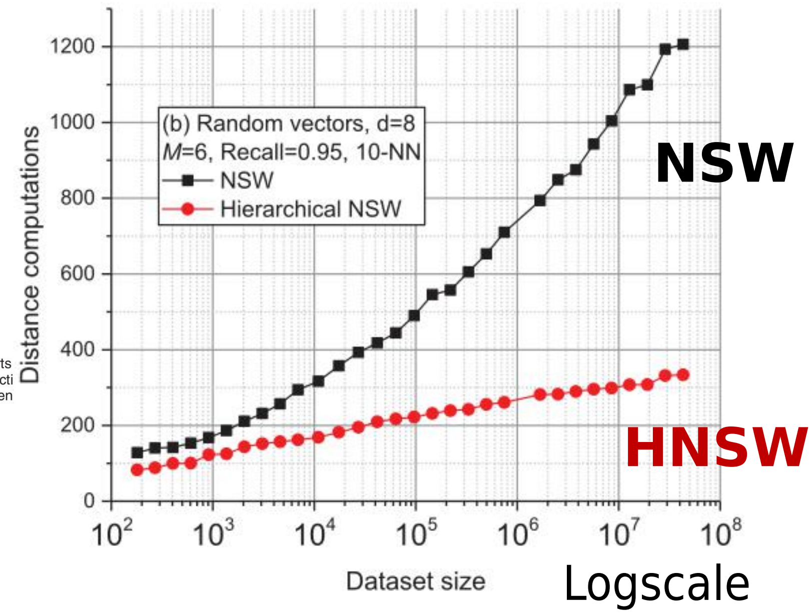


Fig. 1. Illustration of the Hierarchical NSW idea. The search starts an element from the top layer (shown red). Red arrows show directi the greedy algorithm from the entry point to the query (shown green

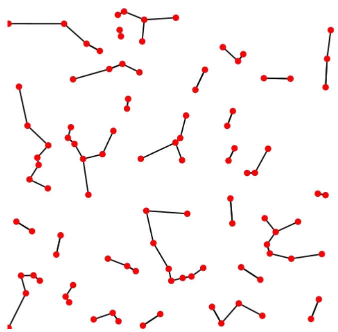


- Simply **inserting vectors one at a time**, connecting it to its k nearest neighbors already in the graph found via search trial, is navigable and small-world
- Hierarchical levels mitigates high out-degrees

Figures: Malkov et al 2014, Malkov & Yashunin 2020

Summary of Graph-Based Indexes

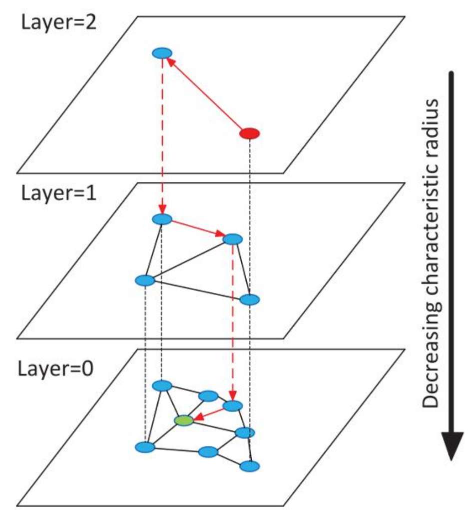
Nearest-Neighbor Graphs



E.g. KGraph (NNDescent)

- √ O(1) offline search
- √ Fast approx. construction
- X Slow for online queries

Monotonic Search Networks / Small Worlds



E.g. FANNG, NSG, Vamana, HNSW

- √ ~O(log N) online search
- X Slow construction

Graph-Based Indexes: Discussion

Advantages

√ Empirically state-of-art throughput/recall

Disadvantages

X Hard to adapt to disk

X Hard to support updates for many graphs

- For HNSW: accuracy degradation issue

X Long construction times for graphs based on search trials (incl. HNSW)

Summary of Indexes

Index Type	Search Efficiency	Search Accuracy	Write Friendliness	Disk Friendliness
Table-Based		↑	↑	↑
Tree-Based	↑			↑
Graph-Based	↑	↑		↓

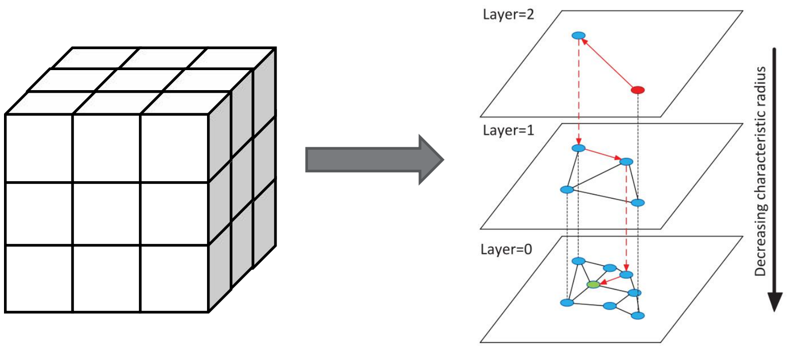
Index Type	Construction Efficiency	Storage Efficiency	Ease of Maintenance
Table-Based			↑
Tree-Based			
Graph-Based	↓		

Challenges to Storage & Indexing

Index Variety

Index Selection

E.g. How to handle workload shift?



Capabilities

Index Design

Examples:
- Disk-Resident Indexes
- Concurrent Indexes
- Distributed Indexes
- Indexes for Predicated Queries
- Etc.

Overview of Query Optimization

Predicated Vector Search Query
e.g. select * from items where price < \$10 order by simTo(query) limit k

Plan Enumeration

Plan Types

- Naive
- Pre-Filtering
- Post-Filtering
- Single-Stage Filtering

Plan Selection

Cost-Based

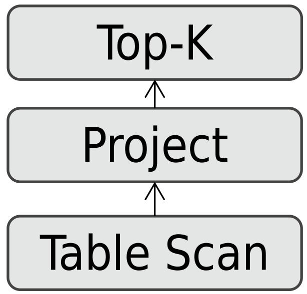
- Cost Model
- Operator Costs

Rule-Based

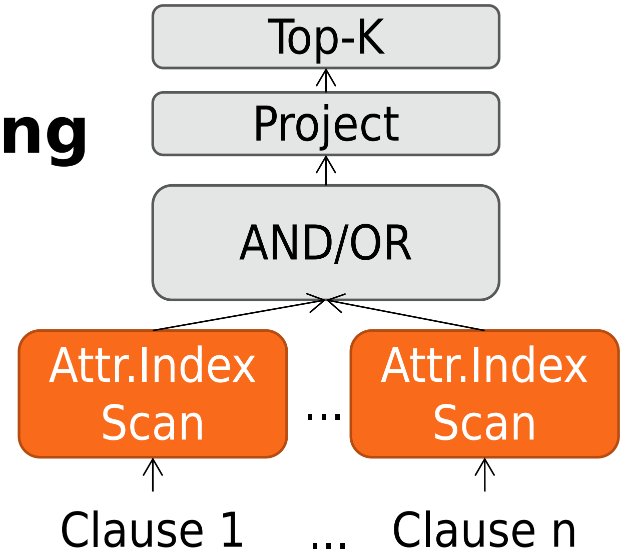
- Rule Design

Plan Types for Predicated Queries

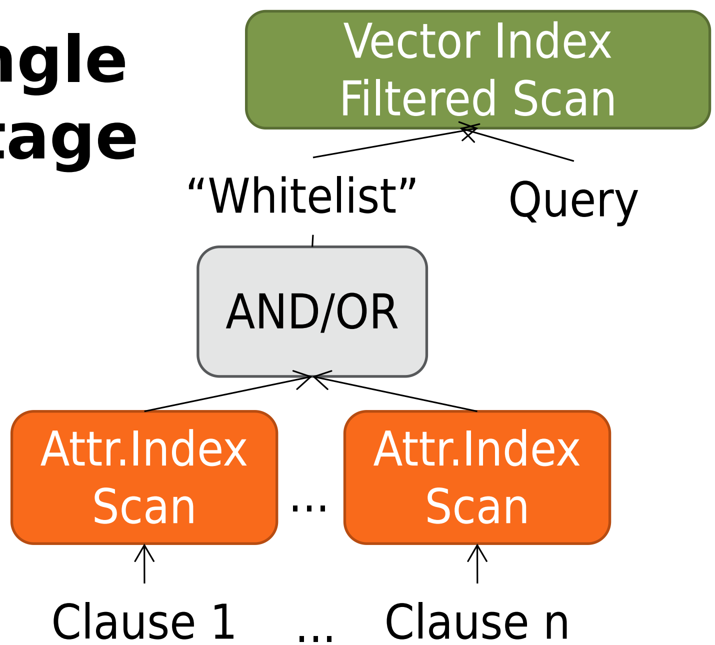
Brute-Force



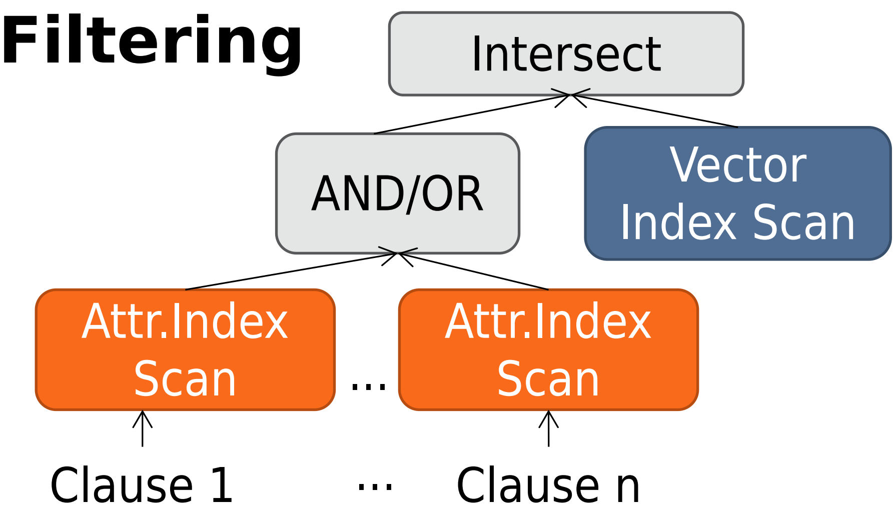
Pre-Filtering



Single-Stage



Post-Filtering



Summary of Plan Types

Plan Type	Advantages	Disadvantages
Brute-Force	√ Exact (100% Recall)	X High latency for weak filters
Pre-Filtering	√ Exact (100% Recall), efficient for strong filters	X High latency for weak filters
Post-Filtering	√ Efficient (Native vector search speed & native attribute filter speed)	X Low accuracy risk (e.g. empty intersection). Mitigation: collect (α k) similar vectors, not just k. E.g. ADBV
Single-Stage Filtering	√ No loss of recall, often more efficient than pre-filtering	X Possibly high latency for strong filters. Mitigation for graph-based indexes: Encourage visiting satisfying vectors, e.g. FilteredDiskANN, HQANN, NHQ; Increase reachability, e.g. ACORN; Decrease failures via partitioning, e.g. Milvus

Plan Enumeration

Predefined, e.g. Weaviate, Milvus, ADBV

- Use a single predefined plan for all predicated queries, e.g. Weaviate, Pinecone
- Predefine multiple plans and select which plan to use at query time, e.g. ADBV, Milvus

Automatic, e.g. PASE, pgvector (PostgreSQL)

- Let the optimizer automatically enumerate plans
  - Post-filter low-accuracy risk is real!

pashkinelfe commented on Sep 21, 2023 • edited Contributor ...

I suppose the case when post-filtering depletes all (or most) of ann-found tuples is completely legit. Though considering how often are the related complaints I'd suggest it to be mentioned in readme/manual explicitly. Maybe both filtering by other attribute or "filtering" due to dead heap tuples could be mentioned both.

Plan Selection

Rule-Based, e.g. Qdrant, Vespa

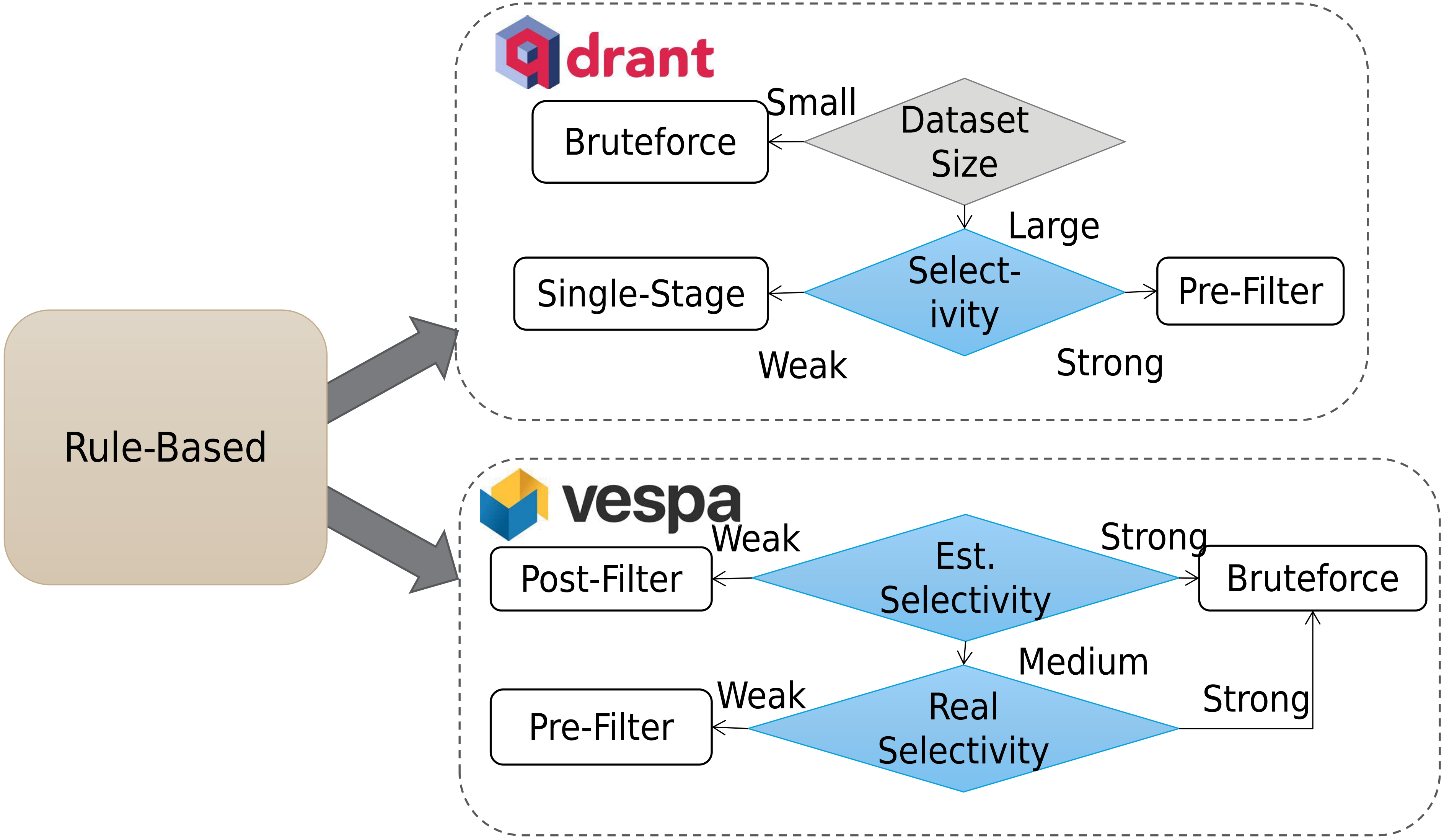
- Simple to implement
- Depends on accurate selectivity estimates

Cost-Based, e.g. ADBV, Milvus

- Select based on a cost model
- Generally more flexible
- Depends on accurate operator cost estimates

Rule-Based Plan Selection

Example: Qdrant and Vespa



Cost-Based Plan Selection in ADBV

Table 1: Notations used by hybrid query optimization

Notation	Meaning
n	the total number of tuples in database
α	the ratio between n and the number of records satisfying structured predicate
β	the visited subcells ratio during VGPQ index searching process in VGPQ Knn Bitmap Scan
γ	the visited subcells ratio during VGPQ index searching process in VGPQ Knn Scan
$\sigma_{\{B,C,D\}}$	amplification factors of *ANNS Scan* operators in $Plan\{B,C,D\}$
c_1	the total time cost to fetch a vector and compute pairwise distance
c_2	the total time cost to fetch a PQ code and run ADC

- Pre-Filter k-NN Scan

$$cost_A = T_0 + \alpha \times n \times c_1$$

- Single-Stage PQ Filtered Scan

$$cost_B = T_0 + \alpha \times n \times c_2 + \sigma_B \times k \times c_1$$

- Single-Stage VGPQ Filtered Scan

$$cost_C = T_0 + \beta \times n \times \alpha \times c_2 + \sigma_C \times k \times c_1$$

- Post-Filter VGPQ Scan

$$cost_D = \gamma \times n \times c_2 + \sigma_D \times k \times c_1$$

Challenges to Query Optimization

Cost Estimation

Single-Stage

- May be difficult to estimate cost of Vector Index Filtered Scan (i.e. backtracking)

Post-Filtering

- Hard to take into account low-accuracy risk

pashkinelfe
Pavel Borisov

I suppose the case when post-filtering depletes all (or most) of ann-found tuples is completely legit. Though considering how often are the related complaints I'd suggest it to be mentioned in readme/manual explicitly.

Source: pgvector Issue #263

Overview of Query Execution

Hardware Acceleration

Data Transfers
- CPU Cache

HW Parallelism
- SIMD/GPU

Data Manipulation

Execution Mode
- Immediate
- Deferred

Distributed Query Processing

Partitioning
- Random/Uniform
- Attribute-Based
- Learned

Consistency
- Strong
- Eventual

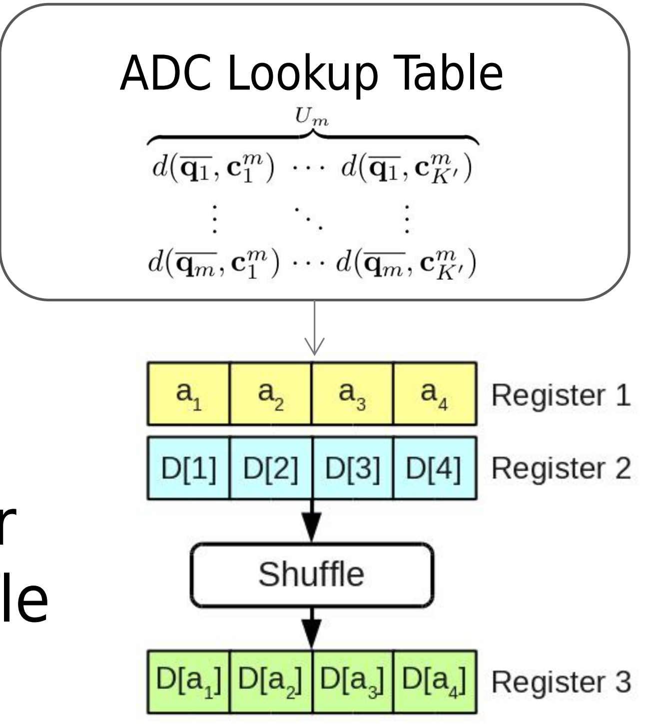
Hardware Acceleration

Data Transfers

- CPU Cache: “Query blocks” keep query vectors cache-resident while assigning threads to data vectors keeps data vectors cache-resident, e.g. Milvus

Data/Task Parallelism

- SIMD/GPU for IVFADC, e.g. Faiss
    - Parallelize lookups by keeping the lookup table inside the SIMD register and simulate lookups via SIMD shuffle (also avoids memory retrieval)
    - Parallelize summations over registers



Data Manipulation

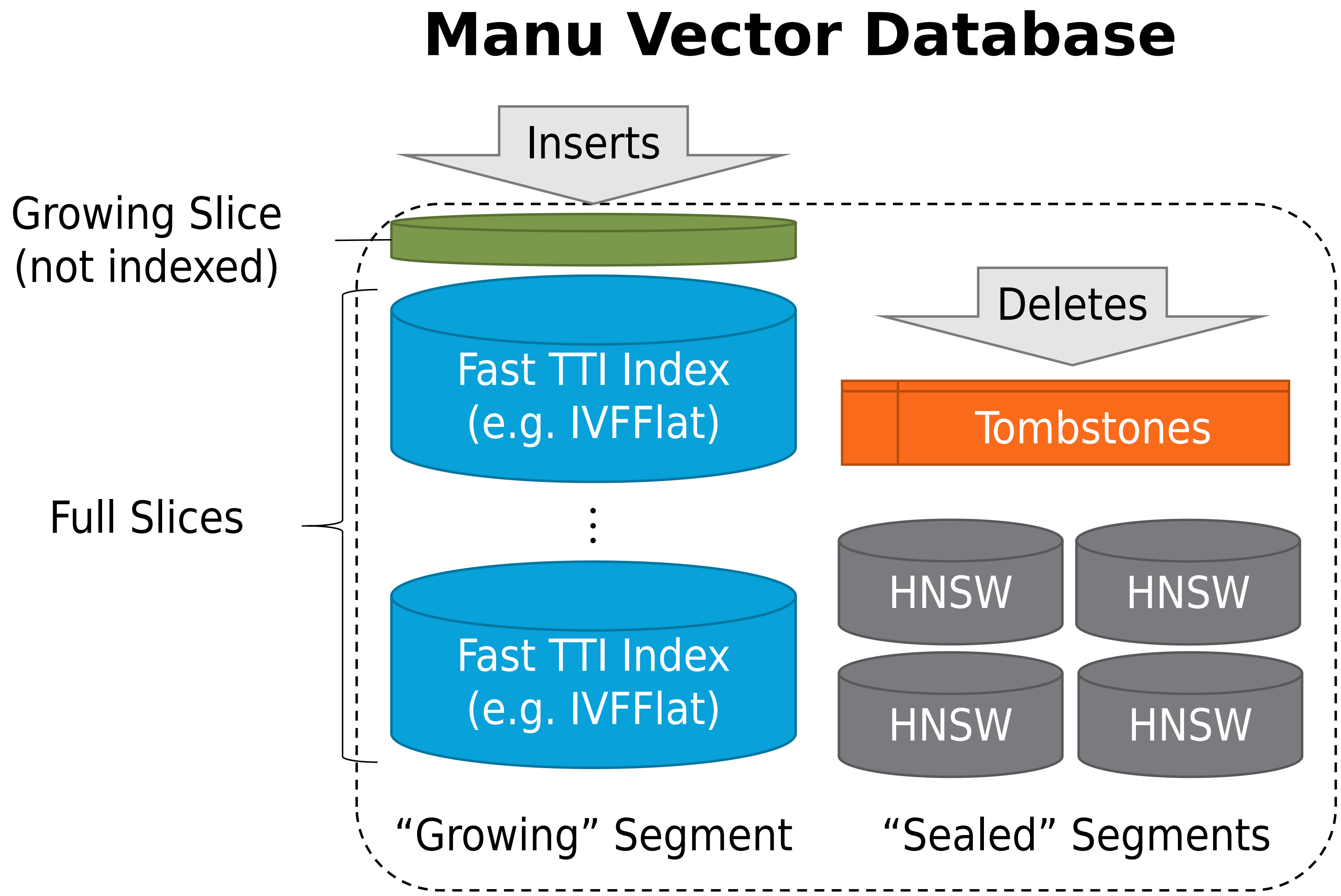
Streaming Updates

- Some indexes support in/up/del, e.g. HNSW
- Vearch: use tombstone deletes to avoid disconnecting the graph + periodic garbage collection

Batched Updates

- Perform in/up/del inside a fast-writeable slow-readable structure which also participates in search
- Reconcile into the slow-writeable fast-readable structure at a convenient time

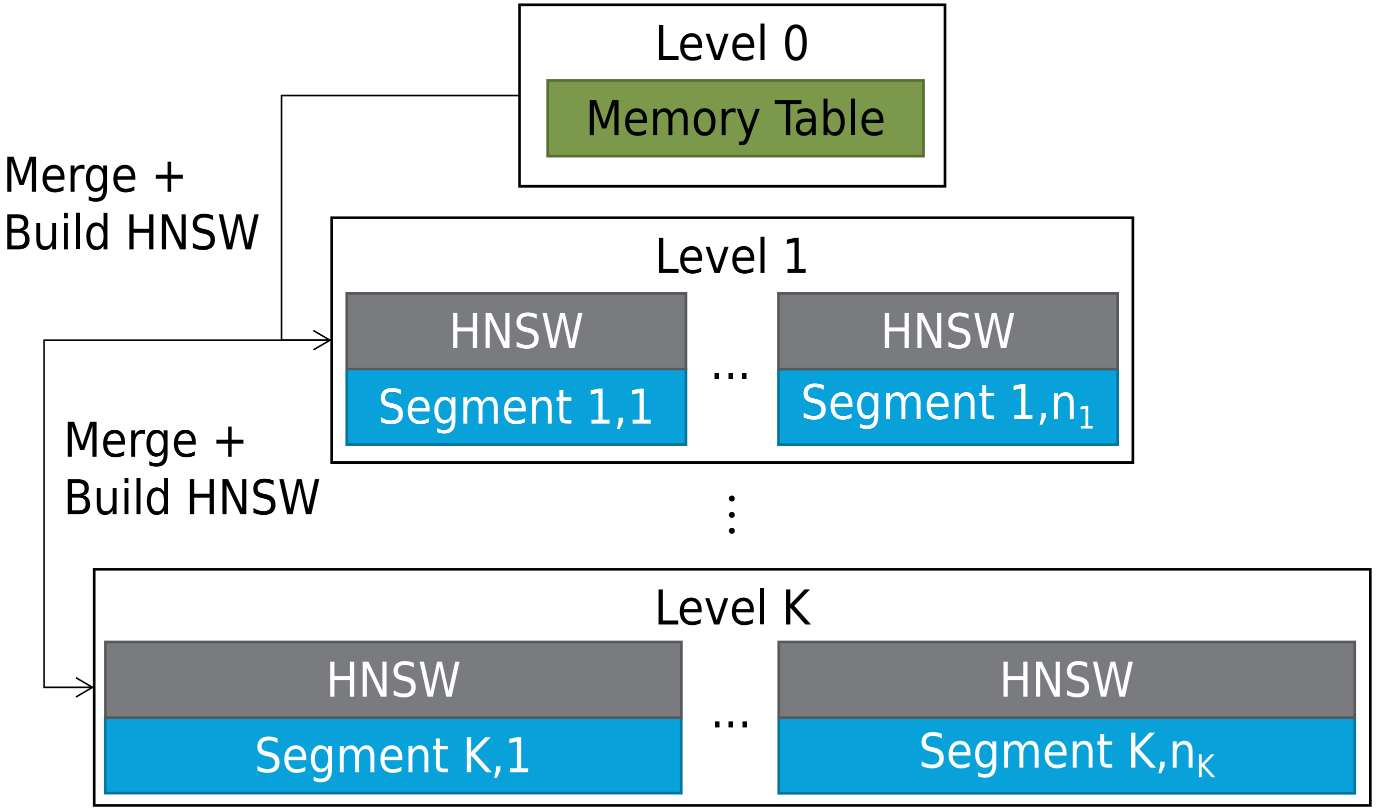
Fast Slices with Slow Segments



- HNSW is built over the growing segment once full, and then the temporary slices are discarded

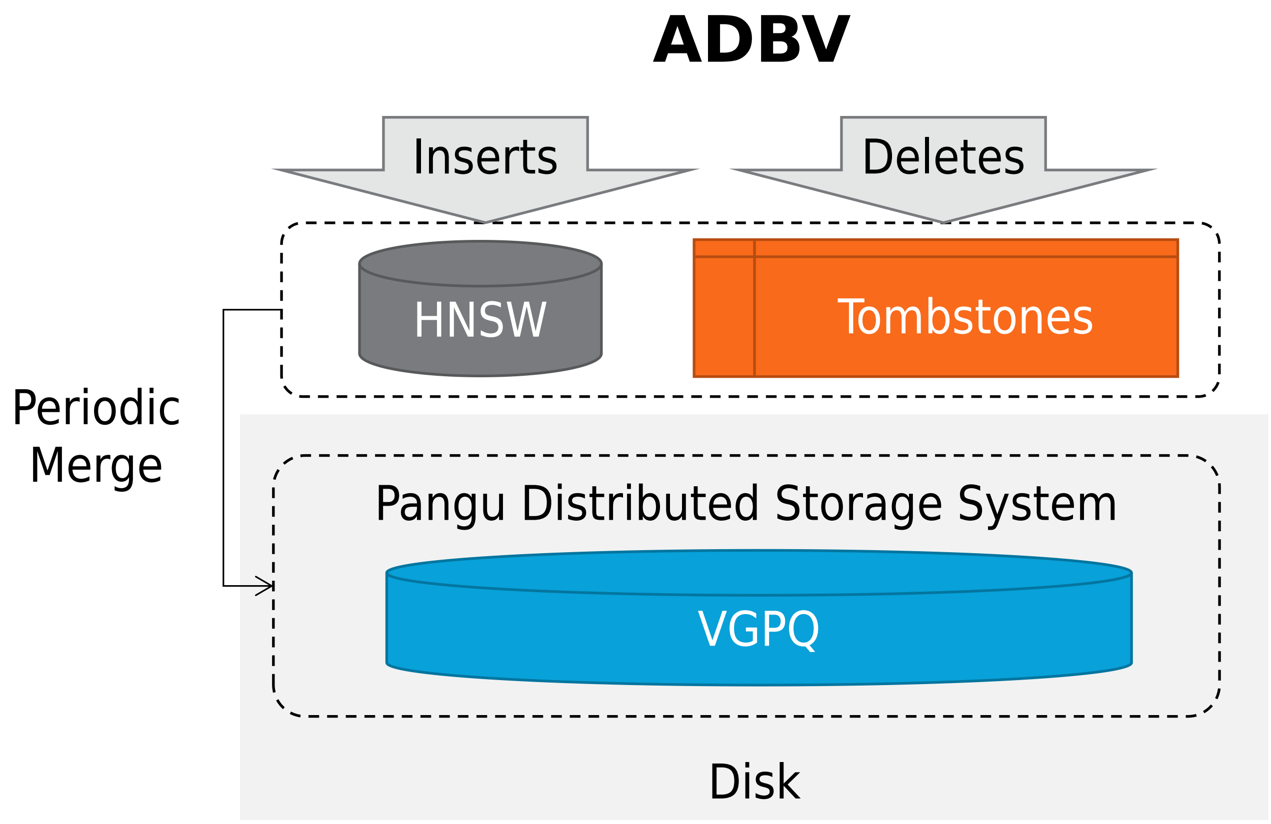
Log-Structured Merge (LSM) Tree

Milvus Vector Database



- HNSW built during segment compaction
- Tombstones are reconciled during compaction

In-Memory HNSW with Disk-Resident Index



- Designed for massive TB+ datasets
- HNSW serves as the fast-writeable structure while disk-resident VGPQ is the slow-writeable structure

Distributed Query Processing

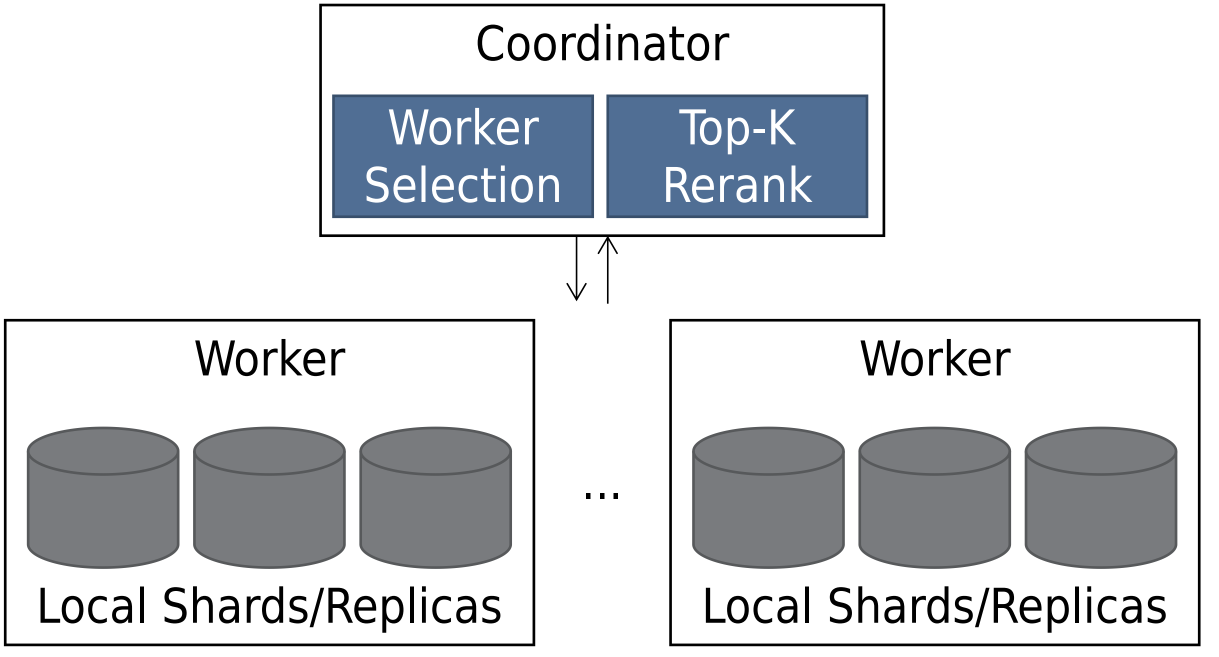
Partitioning

- By attribute if available, e.g. ADBV
- By k-means cluster, e.g. ADBV
- By memory availability, e.g. Vald
- By uniform hashing

Consistency

- Eventual consistency
    - By quorum, e.g. Weaviate
    - By timestamp delta, e.g. Manu
- Strong consistency
    - Concurrent HNSW via internal locks, e.g. Vearch

Scatter-Gather Search



- Hard to know beforehand which workers to select
- k-means partitioning can reduce searched partitions but needs rebalancing

Part 2: Commercial VDBMSs

Native

VEARCH
Pinecone
zilliz
EuclidesDB
milvus
marqo
Chroma
Weaviate
qdrant
Vald
nucliadb

Extended

ClickHouse
AnalyticDB
SingleStore
vespa

Search Engines / Libraries

APACHE LUCENE
elasticsearch
OpenSearch
Solr
Microsoft

VDBMS Types and Capabilities

Native Systems

- Mostly-Vector (Pinecone)
  - Limited query/index types, mainly read-heavy, limited or no predicated queries
- Mostly-Mixed (milvus)
  - Multiple query/index types, mixed workloads, predicated + attribute-only queries

Extended Systems

- NoSQL
- Relational

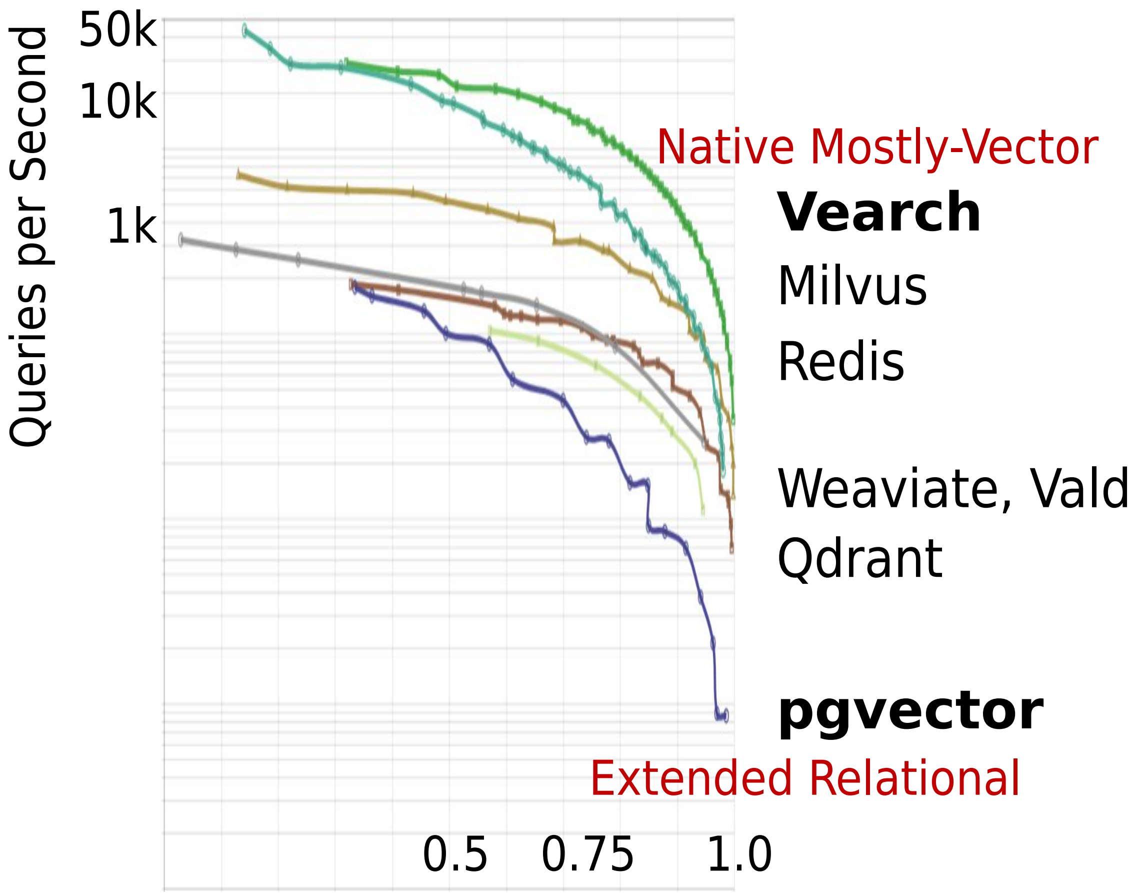
- Basic vector search capability, similar to Mostly-Vector systems
- Comprehensive capabilities, similar to Mostly-Mixed systems

Search Engines & Libraries

- Search Engines (APACHE LUCENE)
- Libraries

- Embedded at application-level
- Offers specific functionality, e.g. a single vector index

VDBMS Performance



Source: ann-benchmarks.com

- Native systems "tend to" outperform extended systems
- This view is already being challenged. Zhang et al ICDE'24: "...there is **no fundamental limitation in using a relational database** (e.g., PostgreSQL) to support efficient vector data management"

Design Considerations

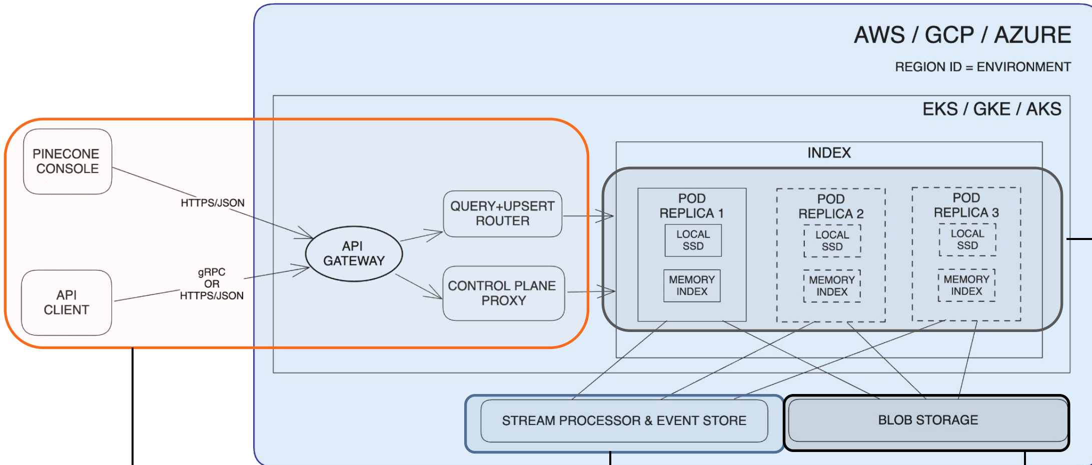
Database Management
(Distributed QP, Failure Recovery, Storage Management, Query Optimizer)

Vector Search Capability
(Operators, Indexes, Plans, Interface)

System Type	Examples	Features	Implementation
Native Mostly-Vector	Pinecone, Vearch	Distributed QP, Failure Recovery	Shards/reps, shared-storage persistence
Native Mostly-Mixed	Milvus, Weaviate	Distributed QP, Failure Recovery, Storage Management	Shards/reps, WAL, LSM-Tree
Extended NoSQL	Redis, Vespa	Single Query/Index Types	Bolt on HNSW
Extended Relational	PASE, ADBV	Multi. Query/Index Types, Operators	Tight Integration

Native Mostly Vector

Example: Pinecone



Pods (shards) and replicas for scatter-gather & availability

API serves as query interface

Stream Processor triggers vector indexing

Blob Storage holds persistent replicas for backup/recovery

Advantages

√ Low-latency searches, high search throughput

Limitations

X Systems with graph-based index may struggle with writes

X Systems with table-based index may struggle with latency/accuracy

Figure: pinecone.io

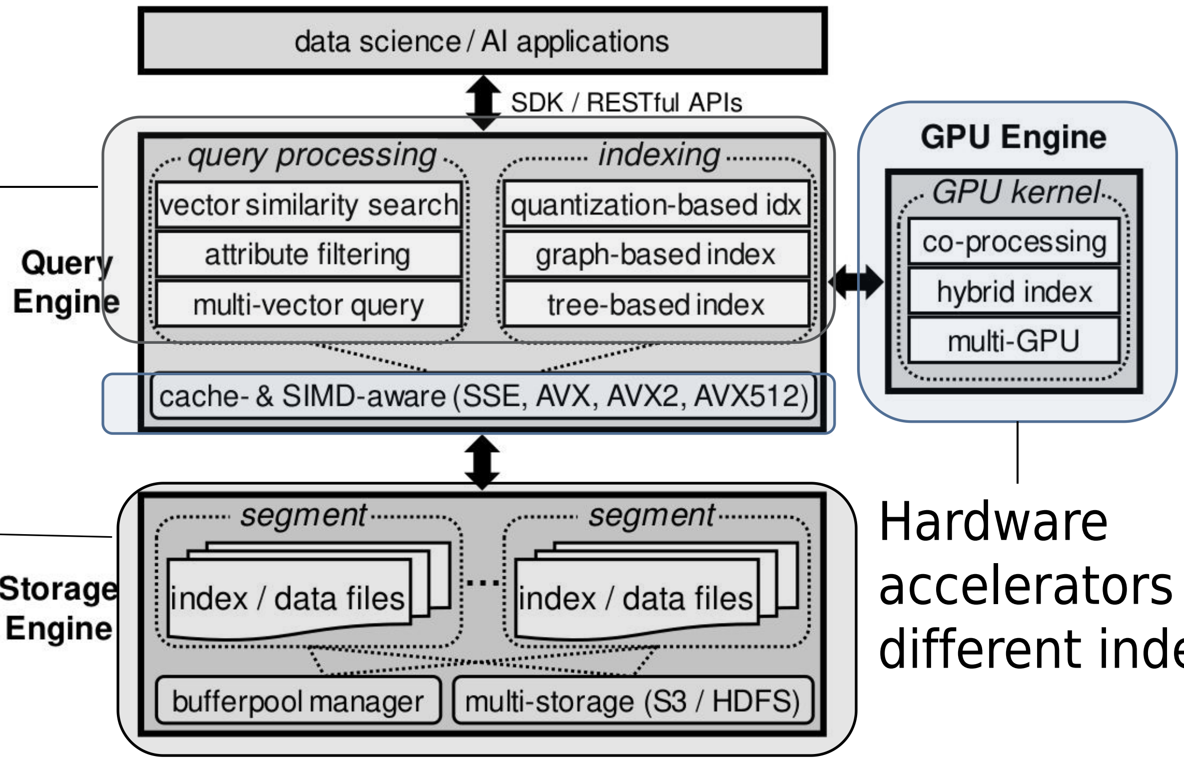
Other Mostly Vector Systems

- **Vald**: Architecturally similar to Pinecone, except uses NGT graph index
- **Vearch**
  - Li et al Middleware 2018: Architecturally similar, except uses table-based index and supports predicated search via post-filtering
  - Latest version: Adds support for attribute-only indexes/queries, pre-filtering, multiple index types (HNSW, IVFPQ)
- **EuclidesDB/Chroma**: On-premise centralized

Native Mostly Mixed

Example: milvus

More query capabilities and index types



More sophisticated storage to support fast writes in addition to reads

Hardware accelerators for different index scans

Advantages

√ Many supported query types, can be configured for both read-heavy and write-heavy workloads

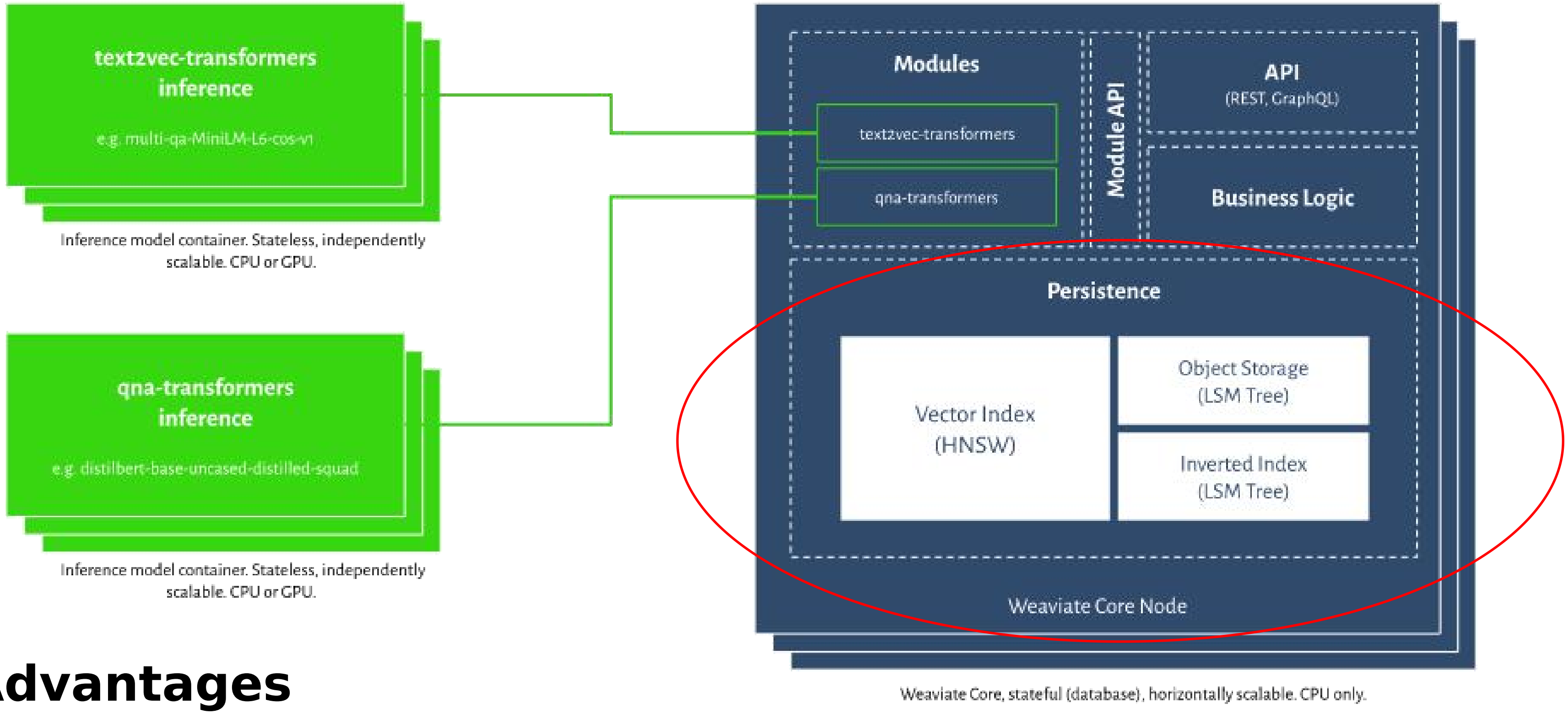
Limitations

X Can be resource-intensive due to more sophisticated storage/recovery

Figure: Wang et al "Milvus: A purpose-built vector data management system". SIGMOD 2021

Native Mostly Mixed

Example: Weaviate



Advantages

√ Many supported query types, can be configured for both read-heavy and write-heavy workloads

Limitations

X Can be resource-intensive due to more sophisticated storage/recovery

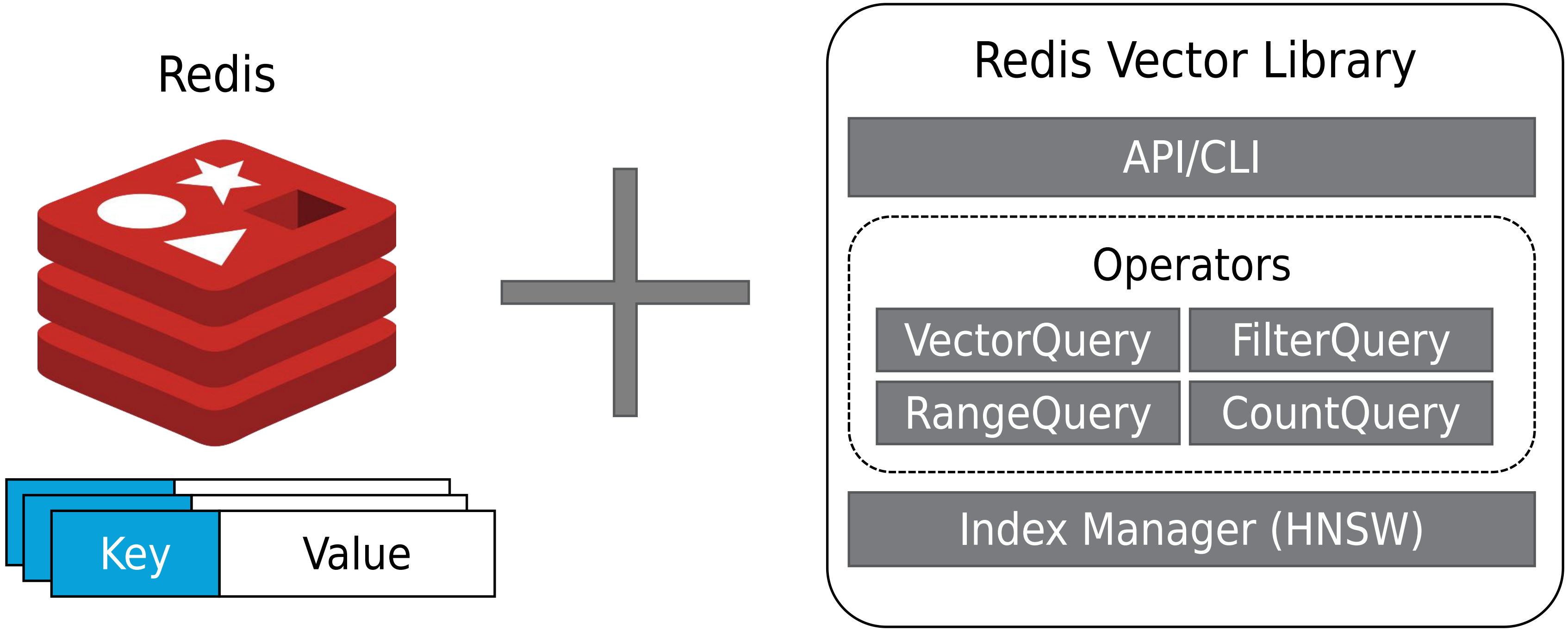
Figure: Wang et al "Milvus: A purpose-built vector data management system". SIGMOD 2021

Other Mostly Mixed Systems

- **Weaviate**
  - Targeted at documents over a graph model; supports both vector search and traditional graph queries via GraphQL
  - HNSW + LSM-Tree, used for raw records + inverted index over keywords and attributes
  - Pre-filtering for predicated search queries
- **Qdrant:** rule-based optimizer for predicated queries
- **NucliaDB/Marqo**

Extended NoSQL

Example: RedisVL



Advantages

√ High-performance vector search, similar to Native Mostly-Vector

√ Combined vector search + non-vector capabilities

Limitations

X As with Mostly-Vector, performance is tied to specific workload

Other NoSQL Systems

- **Vespa**
  - Document model
  - SQL-like query language for complex big data analytics
  - Rule-based optimizer for predicated queries
- **Cosmos DB:** proprietary vector index
- **MongoDB:** HNSW bolt-on
- **Neo4j:** HNSW bolt-on
- **Cassandra:** HNSW bolt-on

Extended Relational

Example: PASE

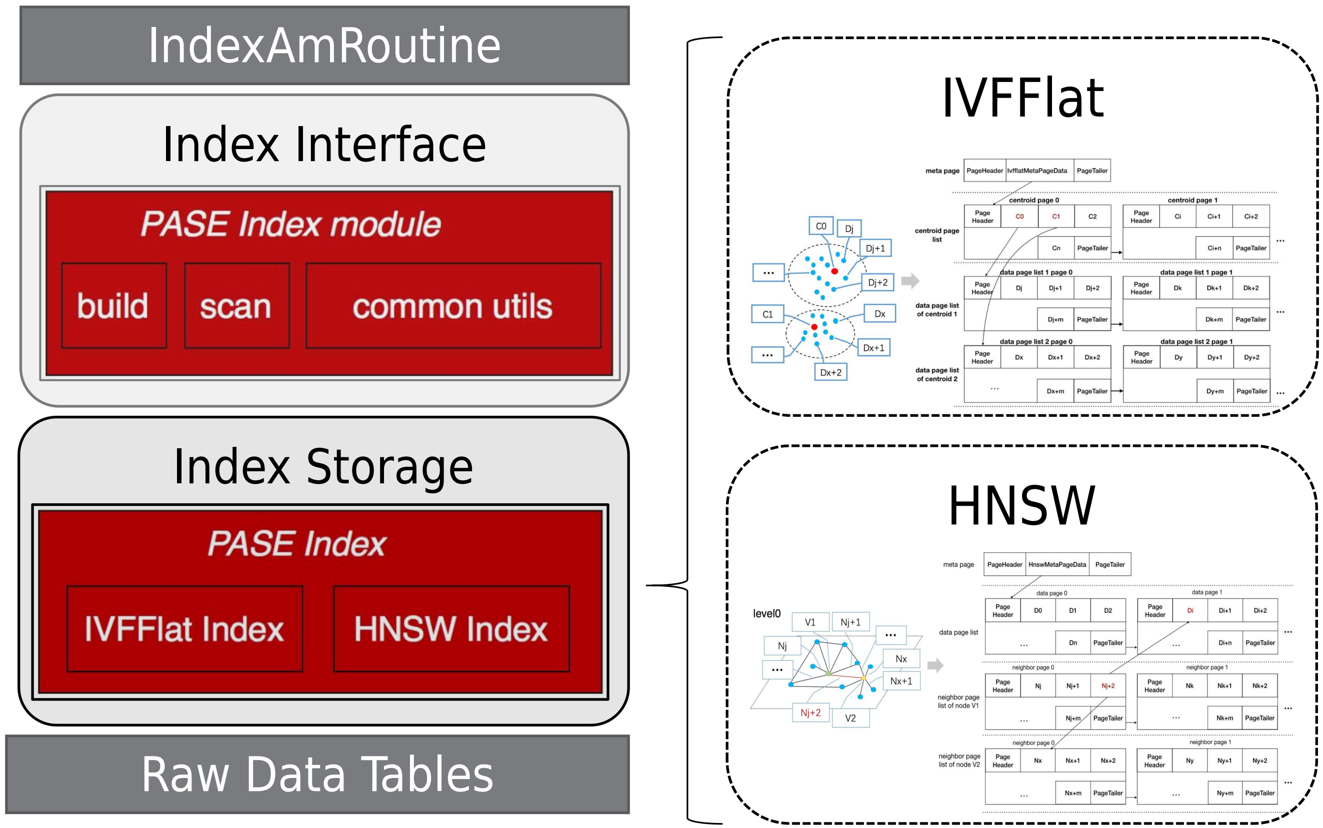


Figure from Yang et al "PASE…". SIGMOD 2020

Advantages

√ Adaptable to different types of workloads

√ As with Ext NoSQL systems, offers diverse capabilities

Limitations

X May suffer performance overhead (e.g. page indirection)

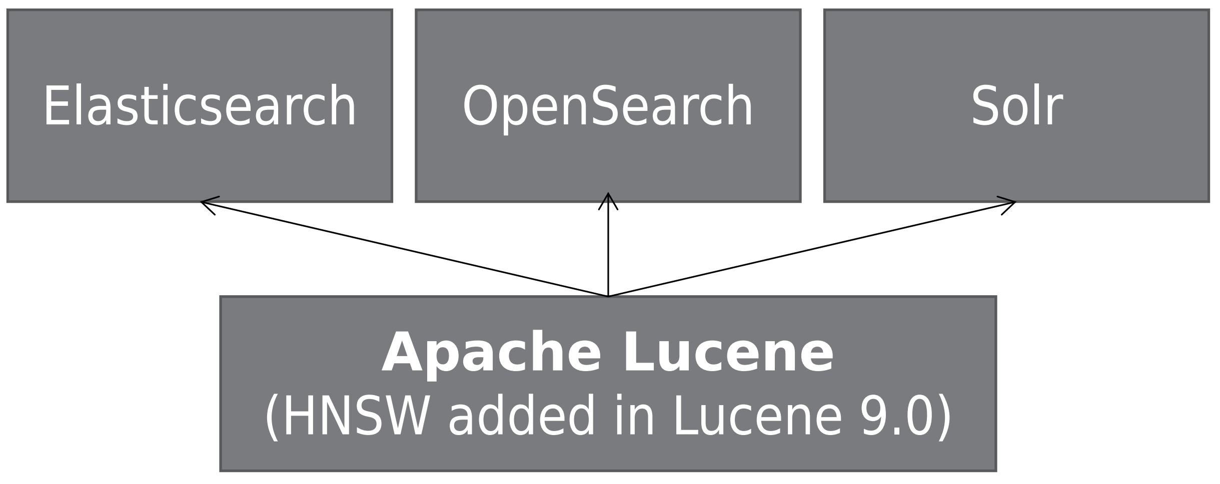
Other Relational Systems

- **pgvector:** similar to PASE
- **AnalyticDB+V**
  - Relational OLAP DBMS over disaggregated compute-storage
  - Adds indexing and fast-slow write structures for supporting real-time read/writes over slow-updateable vector indexes, plus accuracy-aware cost estimation model for ANN
- **SingleStoreDB:** Adds sim. scores to enable brute-force vector search
- **ClickHouse, MyScale**

Search Engines and Libraries

Lin et al “**Vector Search with OpenAI Embeddings: Lucene Is All You Need**” (2023) arXiv:2308.14963

Search Engines



Libraries: Meta Faiss, hnswlib, ANNOY, Microsoft SPTAG, KGraph, E2LSH, FALCONN, etc.

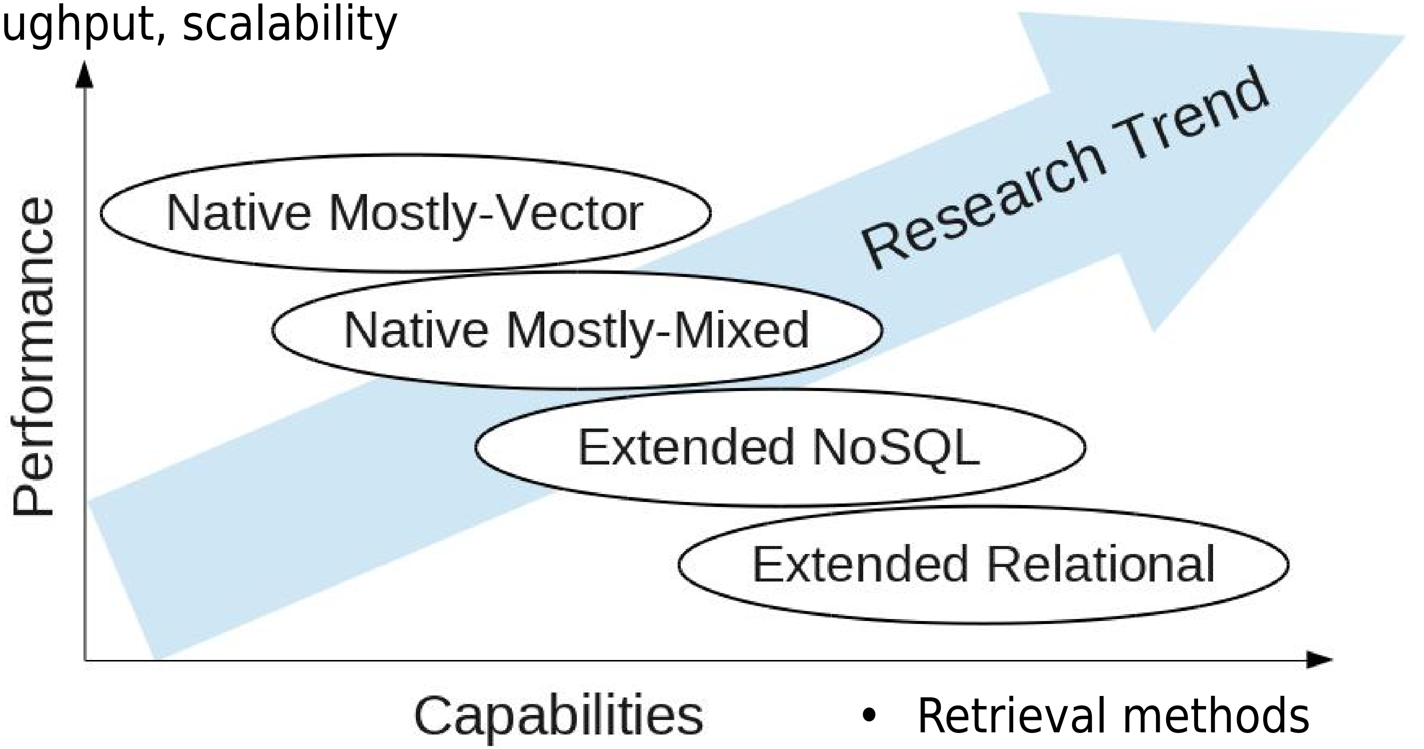
Benchmarks

- Surprisingly few benchmarks
- **ann-benchmarks.com**
  - Real implementations, highly implementation-dependent
- **Li et al TKDE 2020**
  - Idealized implementations

Li et al "Approximate nearest neighbor search on high dimensional data — Experiments, analyses, and improvement." IEEE Trans. Knowl. Data Eng. 32(8), 1475–1488 (2020)

Summary of Vector Database Systems

- Latency
- Throughput, scalability



- Retrieval methods
- Storage methods
- Recovery methods
- Elasticity, availability, consistency, security

Part 3: Challenges and Open Problems

Score Design/Selection

A particular score may not return maximally relevant results, even under high recall:

“The total prod [negative user feedback] rate are 7.3%, 32.6%, and 43.1% at top 1, 3, and 5... roughly 70% are generated by the EBR node during the retrieval stage”

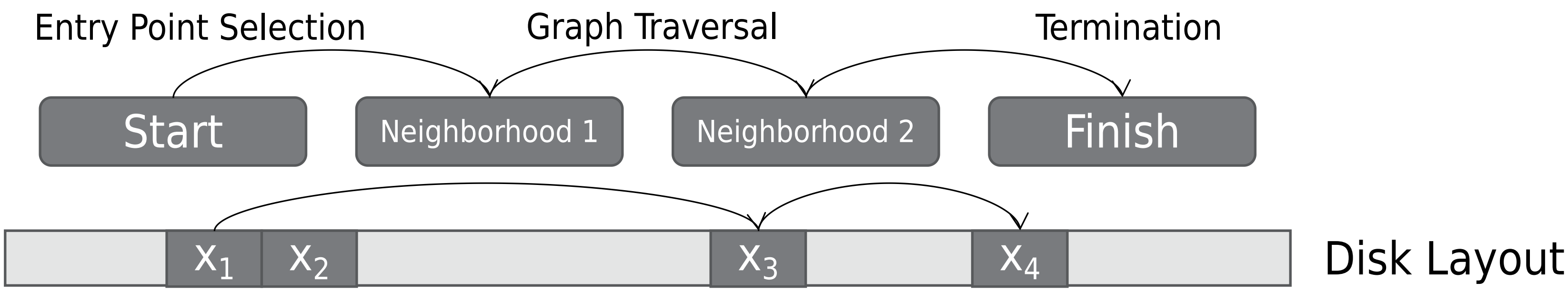
Table 1: Distribution of EBR failures

Failure reasons	Percentage	Category
irrelevant result (fuzzy text match)	53%	junkiness
Location mismatch	18%	junkiness
Language mismatch	4%	junkiness
Misinformation	10%	integrity
Untrustworthy results	10%	integrity
Offensive results	5%	integrity

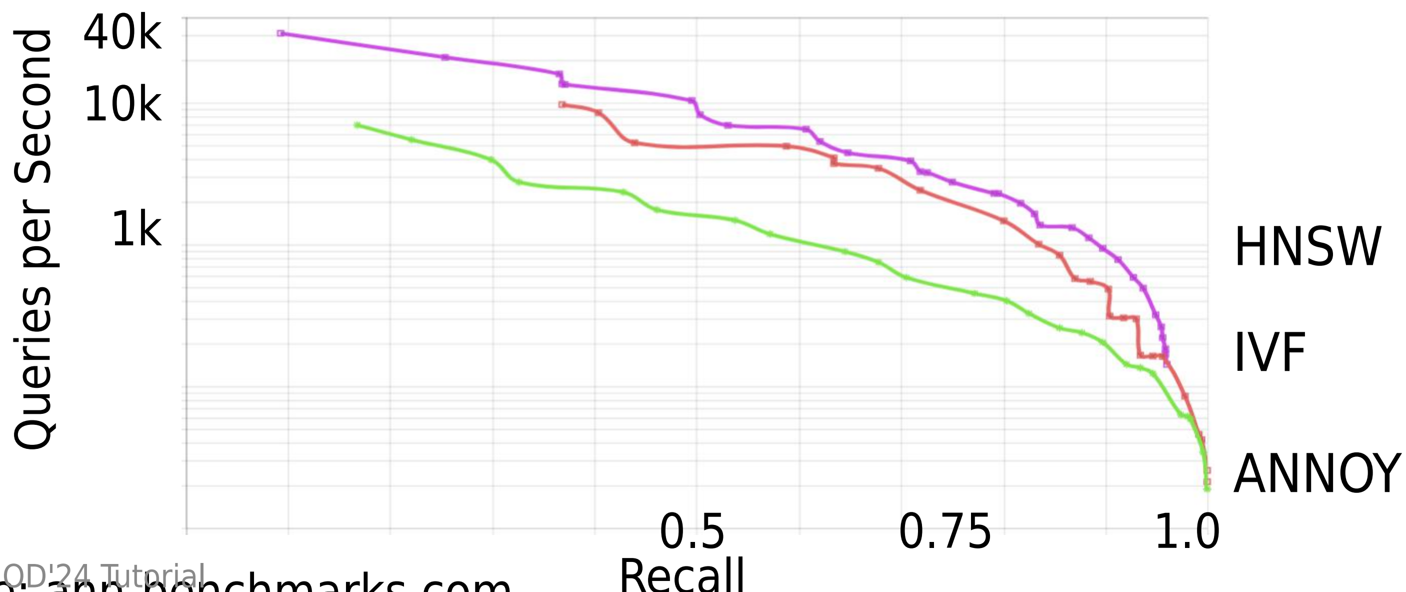
Wang et al “Integrity and junkiness failure handling for embedding-based retrieval: A case study in social network search”. SIGIR 2023

Disk-Friendly/Distributed Indexes

Graphs are slow for disk-resident datasets



Meanwhile, **trees/tables** are disk-friendly but have worse QPS/recall

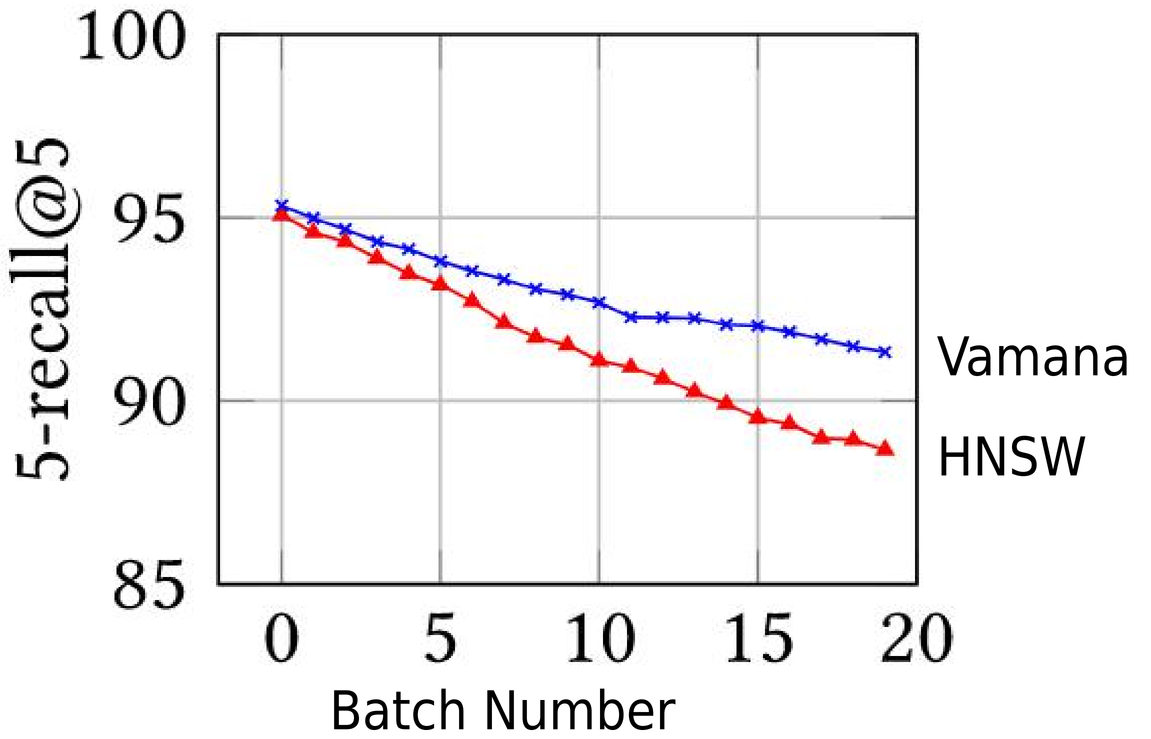


Source: ann-benchmarks.com

Update-Friendly Graphs

Accuracy degradation following series of updates

Effect of Repeated Delete-Reinsert Cycles



Singh et al "FreshDiskANN: A Fast and Accurate Graph-Based ANN Index for Streaming Similarity Search" 2021. arXiv 2105.09613

Easy-to-Build Graphs

- O(DN log N) still too high for huge N (billions)
- ANN_SIFT1B (128 dimensions):
  - Vamana single: 2 days @ 1100 GB peak memory
  - Vamana merged: 5 days @ 64 GB peak memory
- 200M subset of ANN_SIFT1B:
  - HNSW, ef=500: 5.6 hours @ 64 GB peak memory

Sources:

- Malkov & Yashunin "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs" IEEE Trans. Pattern Anal. and Mach. Intell. 2020
- Subramanya et al "DiskANN: Fast accurate billion-point nearest neighbor search on a single node". NeurIPS 2019

New Capabilities

- Multi-Vector Search
  - NRA only works with bounded scores (e.g. cosine)
- Incremental k-NN
- Secure k-NN
- VDBMS Benchmark

Thanks!

Q and A