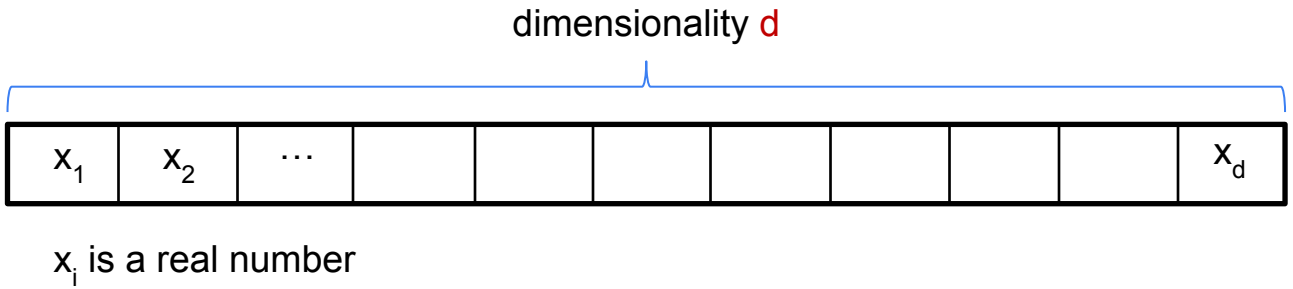
The Quest for F a s t e r ANN Vector Search

Manos Chatzakis, Francesca Del Gaudio, Sophia Sideri, Themis Palpanas

LIPADE, Université Paris Cité

Vector

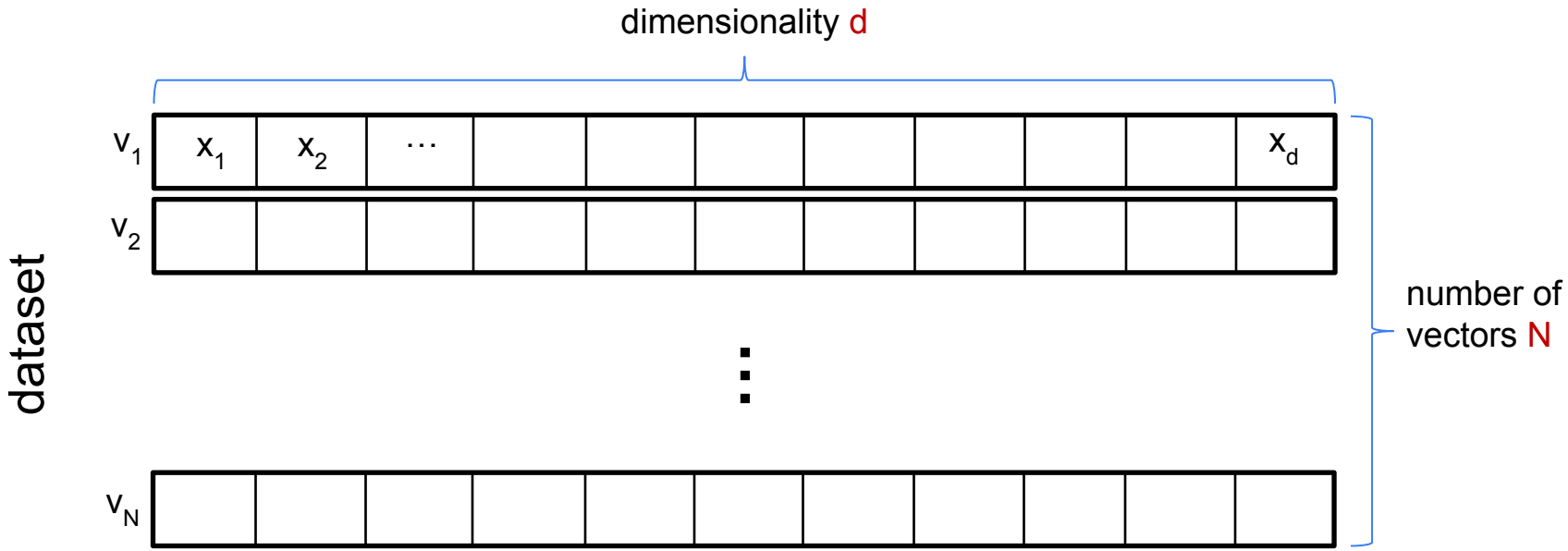
- represented as *d*-dimensional array



x_i is a real number

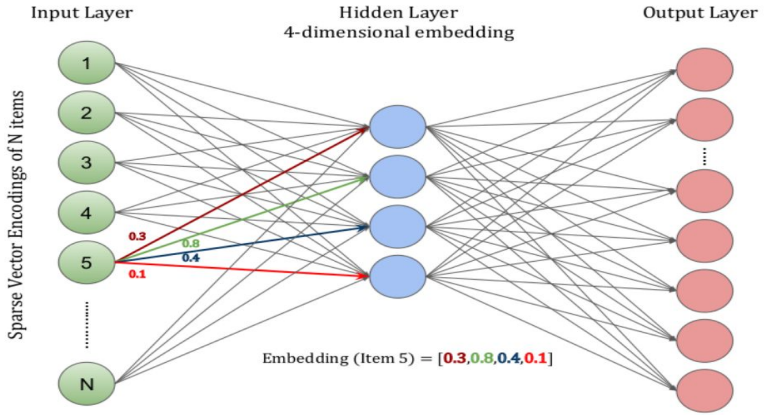
Vector Collections

- represented as *N d*-dimensional arrays

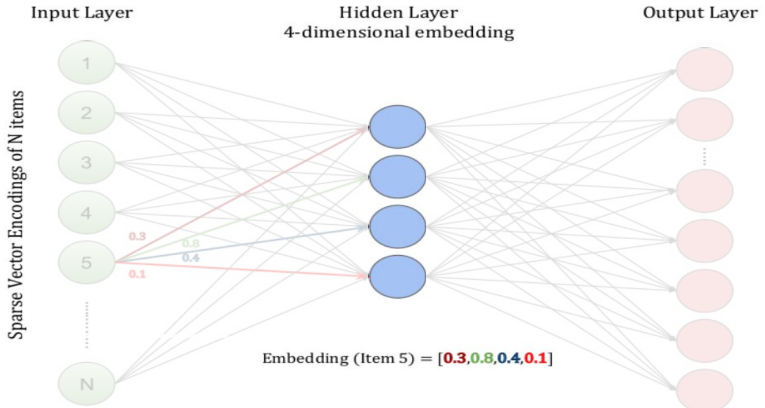


sequences
text
images
video
graphs
molecules
...

sequences
text
images
video
graphs
molecules
...

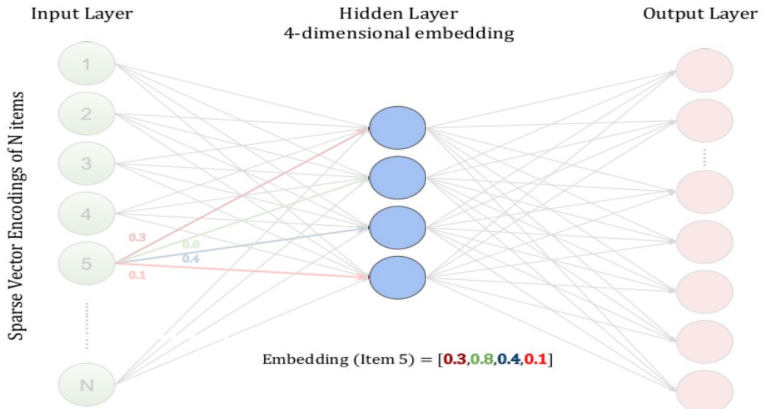


sequences
text
images
video
graphs
molecules
...



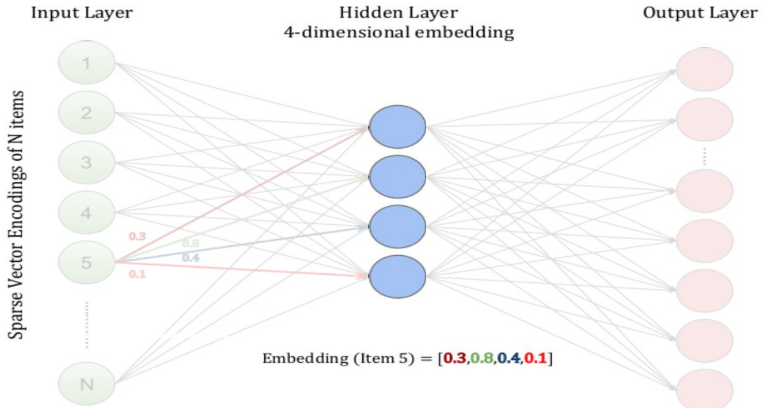
x_1	x_2	⋯	x_d

sequences
text
images
video
graphs
molecules
...



deep embeddings
high-d vectors learned using a DNN

sequences
text
images
video
graphs
molecules
...



deep embeddings
high-d vectors learned using a DNN

- image retrieval
- recommendations
- entity matching
- fraud detection
- drug discovery
- retrieval augmented generation (RAG)
- …

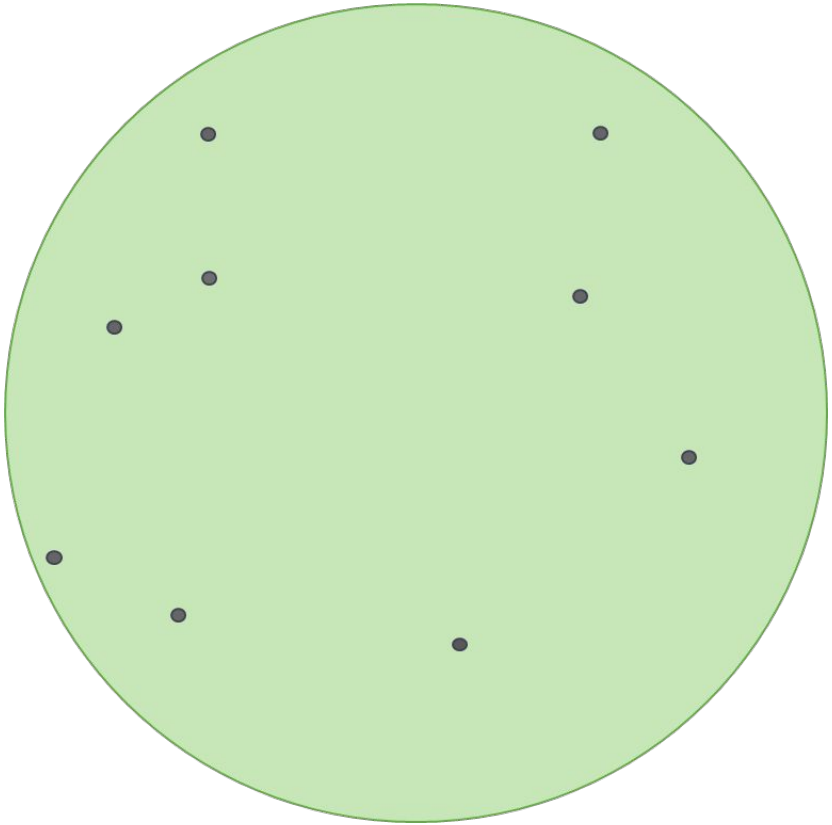
Vector Search

- find similar vectors, based on some distance function

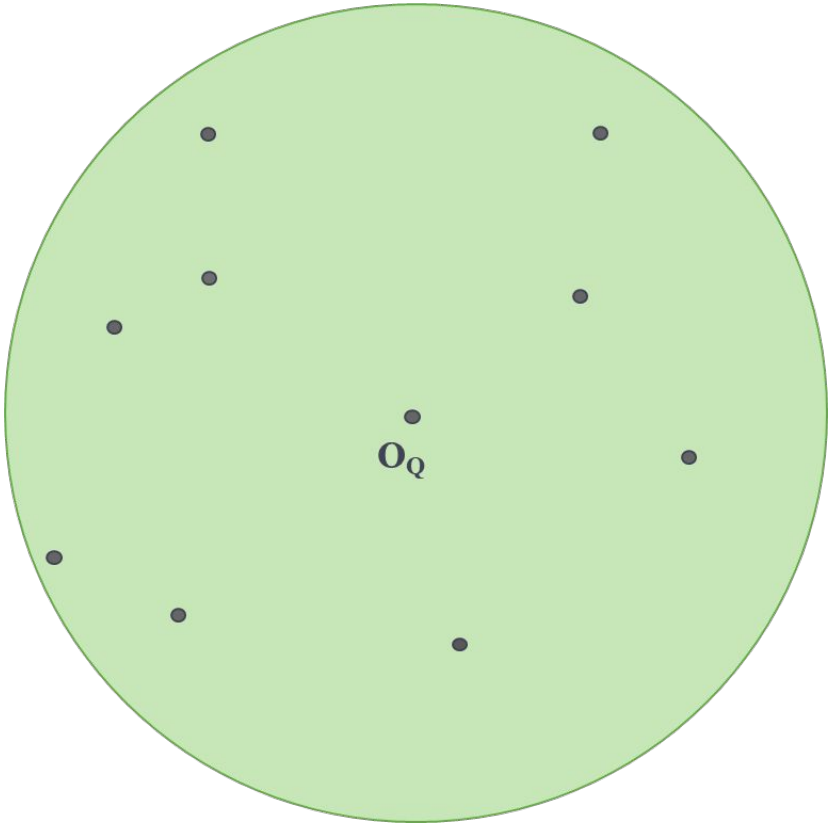
- popular distance functions (similarity/dissimilarity)
  - Euclidean (L2)
  - cosine similarity
  - inner-product
- Euclidean is equivalent to cosine similarity and maximum inner-product (under some non-restrictive vector transformations)

- types of similarity search
  - ε-range
  - Nearest Neighbor (NN)

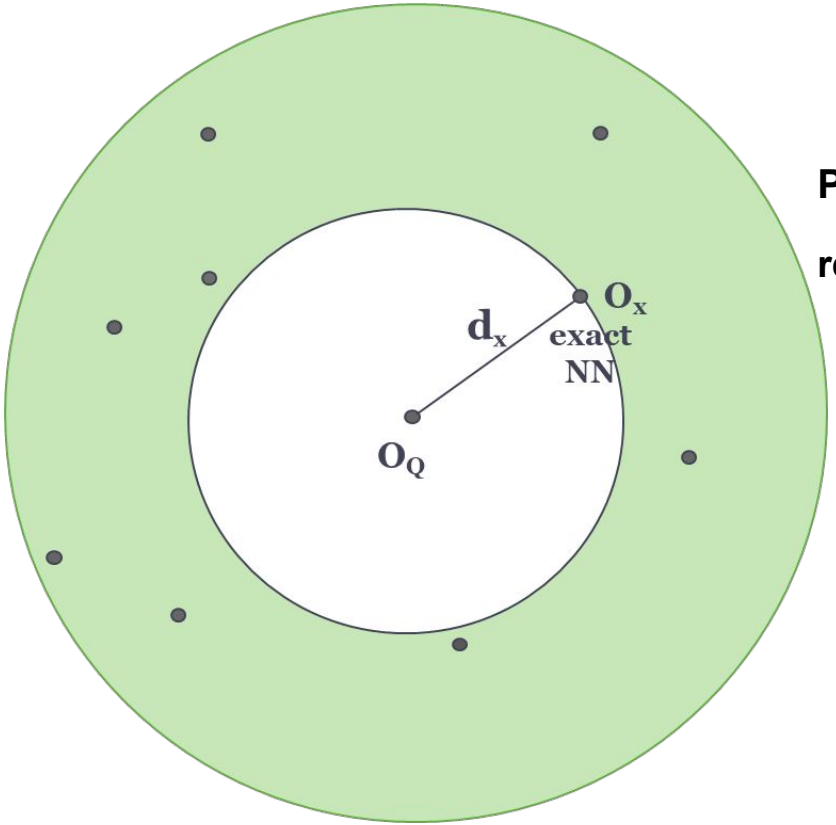
Vector Nearest Neighbor Search



Vector Nearest Neighbor Search



Vector Nearest Neighbor Search



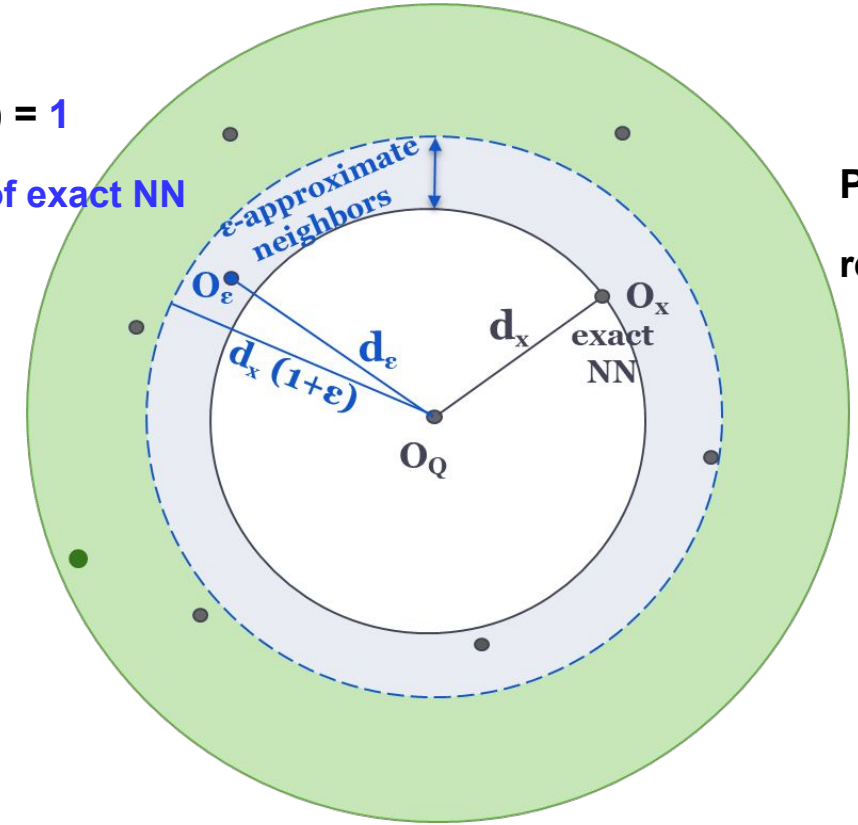
Prob($d_x = \min\{d_i\}$) = 1

result is exact NN

Vector Nearest Neighbor Search

Prob(d_ε <= d_x (1+ε)) = 1

result within (1+ ε) of exact NN with probability 1



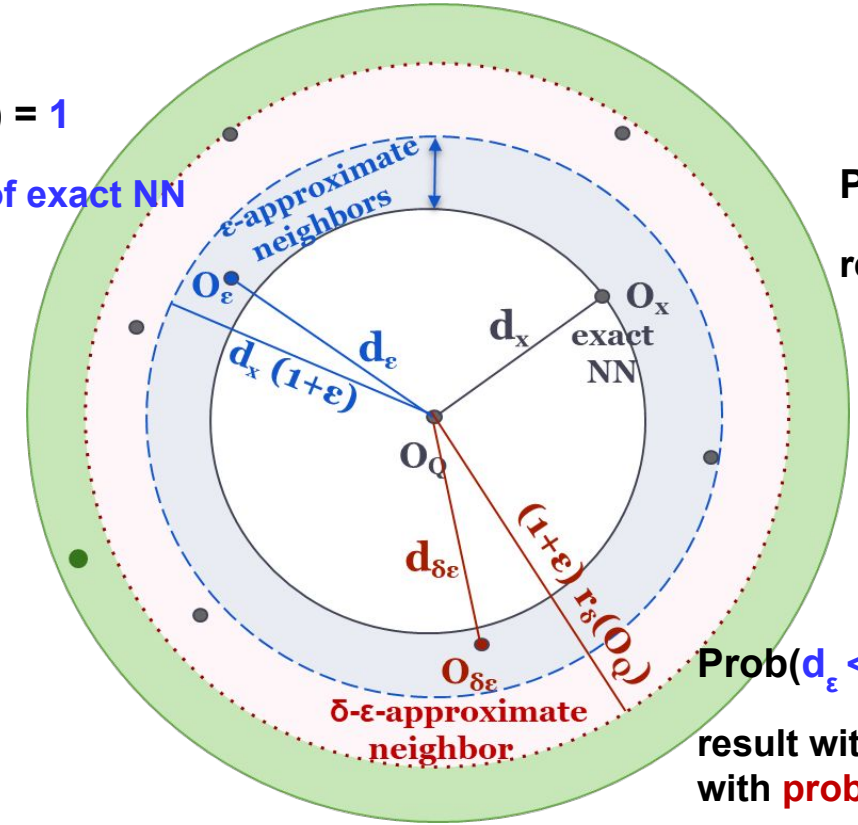
Prob(d_x = min{d_i}) = 1

result is exact NN

Vector Nearest Neighbor Search

Prob(d_ε <= d_x (1+ε)) = 1

result within (1+ ε) of exact NN with probability 1



Prob(d_x = min{d_i}) = 1

result is exact NN

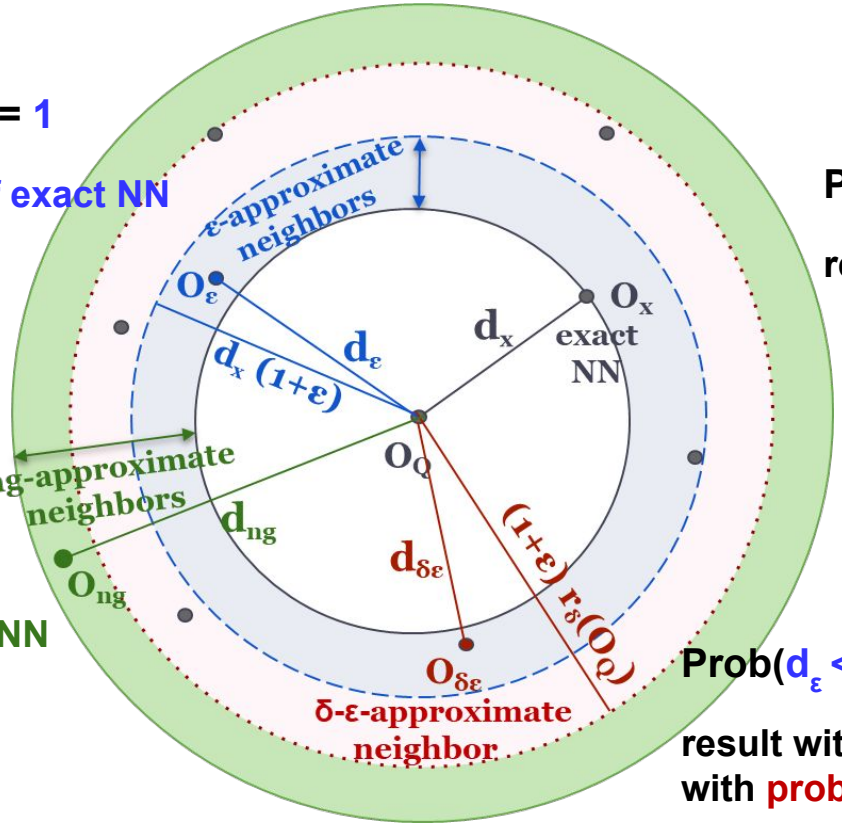
Prob(d_ε <= d_x (1+ε)) >= δ

result within (1+ ε) of exact NN with probability at least δ

Vector Nearest Neighbor Search

Prob(d_ε <= d_x (1+ε)) = 1

result within (1+ ε) of exact NN with probability 1



Prob(d_x = min{d_i}) = 1

result is exact NN

Prob(d_{ng} <>= ?) = ?

result within ? of exact NN

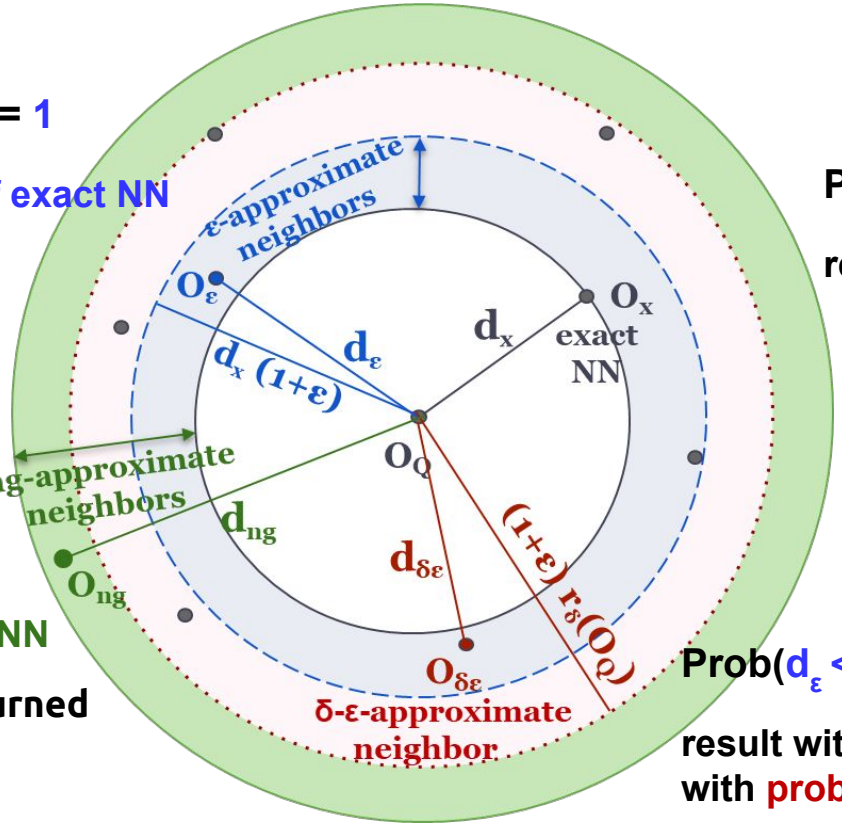
Prob(d_ε <= d_x (1+ε)) >= δ

result within (1+ ε) of exact NN with probability at least δ

Vector Nearest Neighbor Search

Prob(d_ε <= d_x (1+ε)) = 1

result within (1+ ε) of exact NN with probability 1



Prob(d_x = min{d_i}) = 1

result is exact NN

Prob(d_{ng} <>= ?) = ?

result within ? of exact NN

***recall* is the number of returned answers that are correct (part of the groundtruth)**

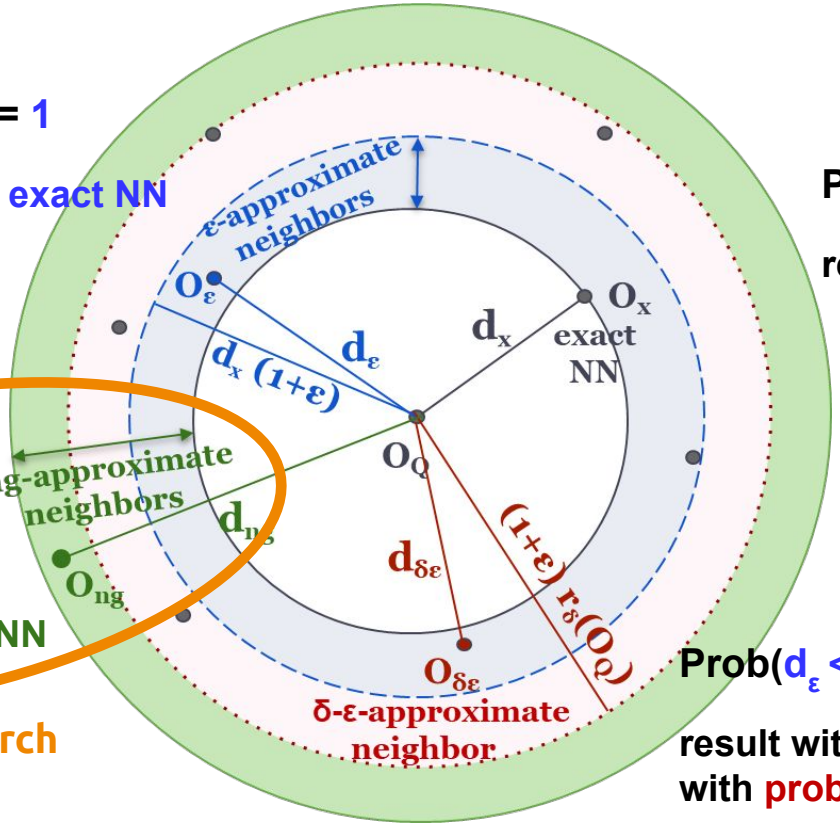
Prob(d_ε <= d_x (1+ε)) >= δ

result within (1+ ε) of exact NN with probability at least δ

Vector Nearest Neighbor Search

Prob(d_ε <= d_x (1+ε)) = 1

result within (1+ ε) of exact NN with probability 1



Prob(d_{ng} <>= ?) = ?

result within ? of exact NN

ng-approximate NN search

Prob(d_x = min{d_i}) = 1

result is exact NN

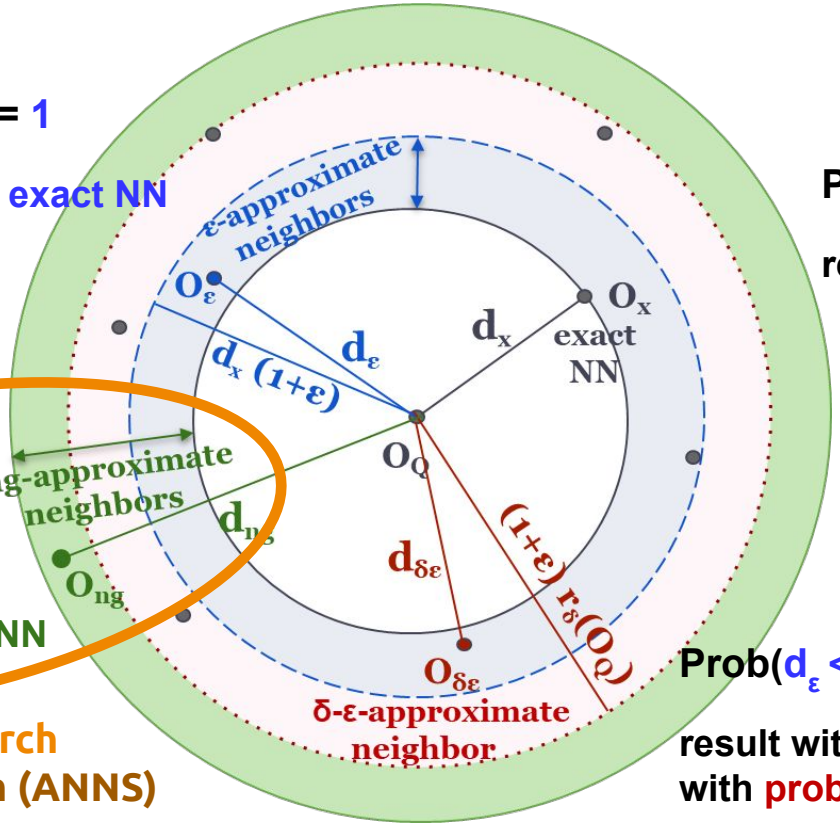
Prob(d_ε <= d_x (1+ε)) >= δ

result within (1+ ε) of exact NN with probability at least δ

Vector Nearest Neighbor Search

Prob(d_ε <= d_x (1+ε)) = 1

result within (1+ ε) of exact NN with probability 1



Prob(d_{ng} <>= ?) = ?

result within ? of exact NN

ng-approximate NN search
Approximate NN Search (ANNS)

Prob(d_x = min{d_i}) = 1

result is exact NN

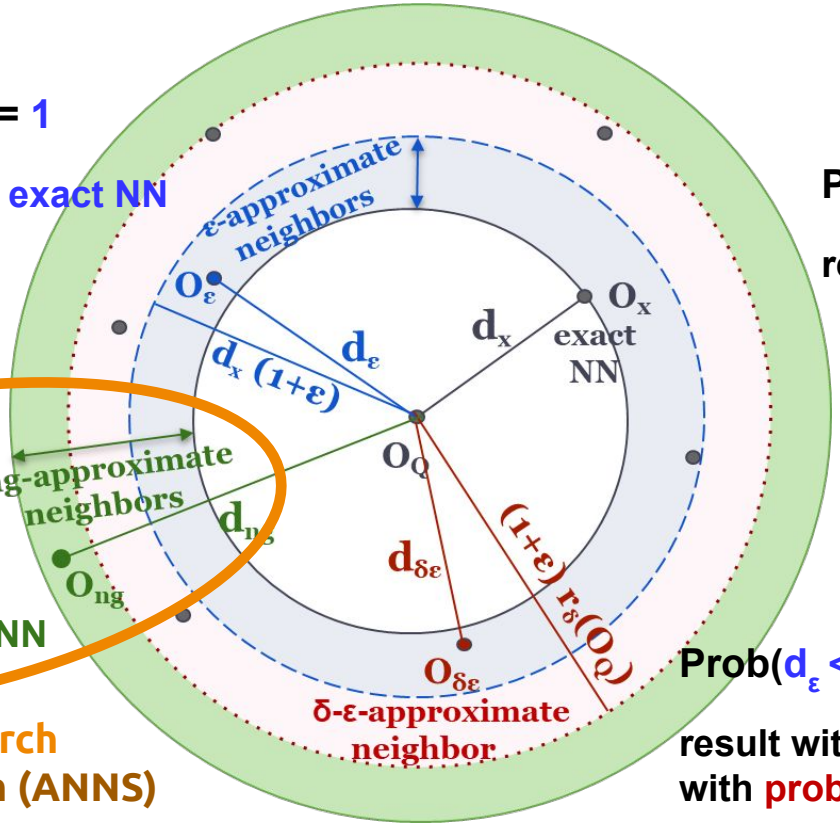
Prob(d_ε <= d_x (1+ε)) >= δ

result within (1+ ε) of exact NN with probability at least δ

Vector Nearest Neighbor Search

Prob(d_ε <= d_x (1+ε)) = 1

result within (1+ ε) of exact NN with probability 1



Prob(d_x = min{d_i}) = 1

result is exact NN

Prob(d_{ng} <>= ?) = ?

result within ? of exact NN

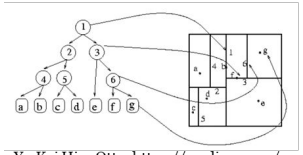
ng-approximate NN search
Approximate NN Search (ANNS)
vector search

Prob(d_ε <= d_x (1+ε)) >= δ

result within (1+ ε) of exact NN with probability at least δ

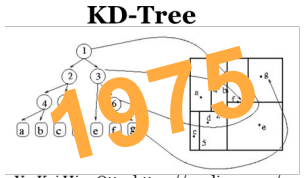
A Brief History of Vector Similarity Search

KD-Tree



Yu Kai Him Otto, https://medium.com/

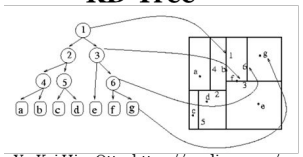
A Brief History of Vector Similarity Search



Yu Kai Him Otto, https://medium.com/

A Brief History of Vector Similarity Search

KD-Tree

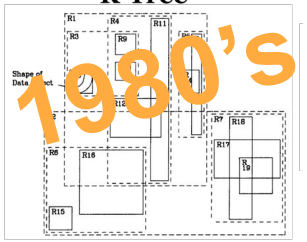


Yu Kai Him Otto, https://medium.com/

Publications

Bentley-CACM'75

R-Tree

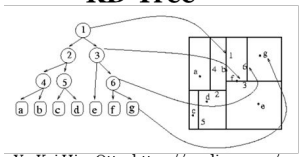


Publications

Guttman-SIGMOD'84

A Brief History of Vector Similarity Search

KD-Tree

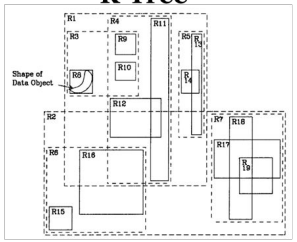


Yu Kai Him Otto, https://medium.com/

Publications

Bentley-CACM'75

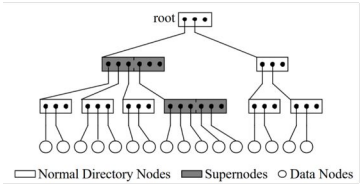
R-Tree



Publications

Guttman-SIGMOD'84

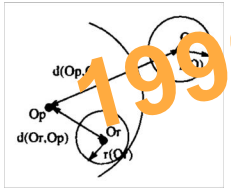
X-Tree



Publications

Berchtold et al - VLDB'96

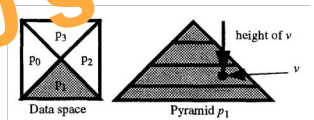
M-Tree



Publications

Ciaccia et al-VLDB'97

Pyramid technique



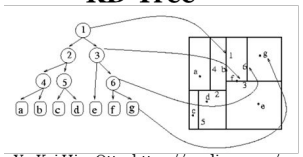
Publications

Berchtold et al - SIGMOD'98

1990's

A Brief History of Vector Similarity Search

KD-Tree

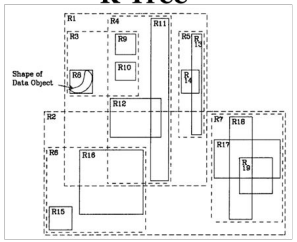


Yu Kai Him Otto, https://medium.com/

Publications

Bentley-CACM'75

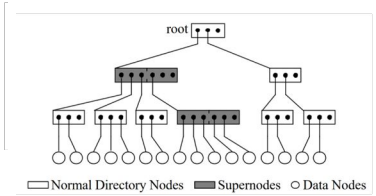
R-Tree



Publications

Guttman-SIGMOD'84

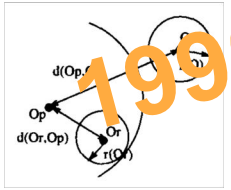
X-Tree



Publications

Berchtold et al - VLDB'96

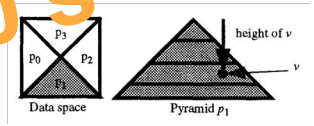
M-Tree



Publications

Ciaccia et al-VLDB'97

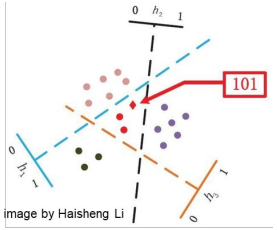
Pyramid technique



Publications

Berchtold et al - SIGMOD'98

LSH



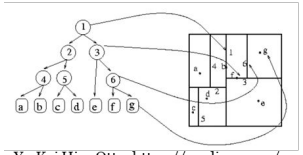
Publications

Indyk et al. STOC' 98

1990's

A Brief History of Vector Similarity Search

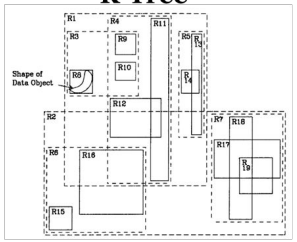
KD-Tree



Yu Kai Him Otto, https://medium.com/

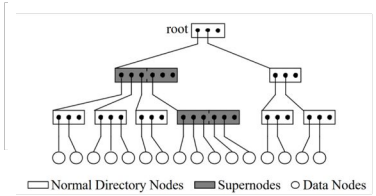
Publications: Bentley-CACM'75

R-Tree



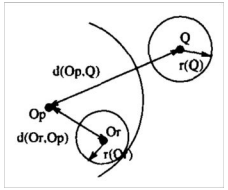
Publications: Guttman-SIGMOD'84

X-Tree



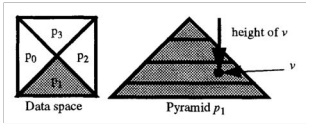
Publications: Berchtold et al - VLDB'96

M-Tree



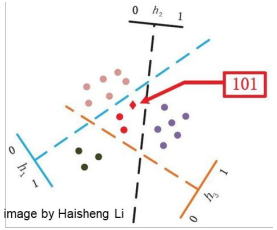
Publications: Ciaccia et al- VLDB'97

Pyramid technique



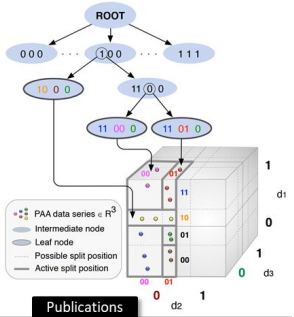
Publications: Berchtold et al - SIGMOD'98

LSH



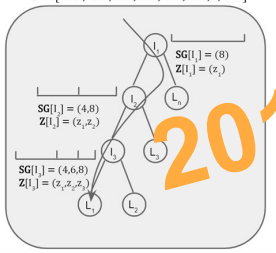
Publications: Indyk et al. STOC' 98

iSAX



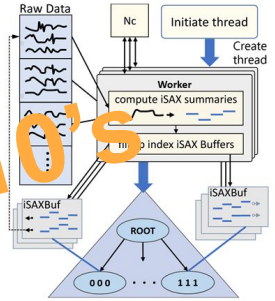
Publications: Shieh-KDD'08

DSTREE



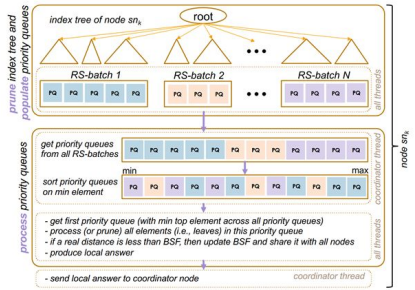
Publications: Wang-PVLDB'13

MESSI



Publications: Peng et al. VLDBJ' 21

Odyssey

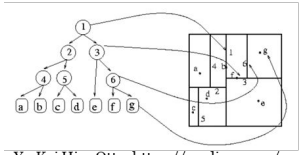


Publications: Chatzakis et al. PVLDB' 23

2010's

A Brief History of Vector Similarity Search

KD-Tree

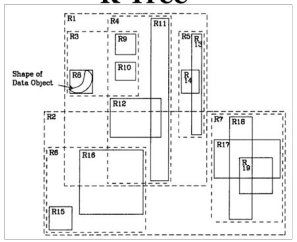


Yu Kai Him Otto, https://medium.com/

Publications

Bentley-CACM'75

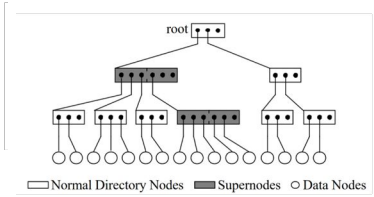
R-Tree



Publications

Guttman-SIGMOD'84

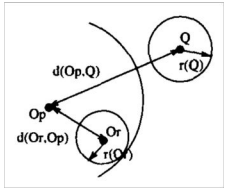
X-Tree



Publications

Berchtold et al - VLDB'96

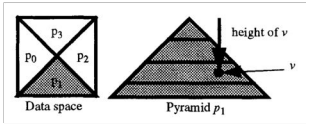
M-Tree



Publications

Ciaccia et al- VLDB'97

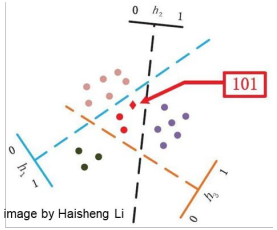
Pyramid technique



Publications

Berchtold et al - SIGMOD'98

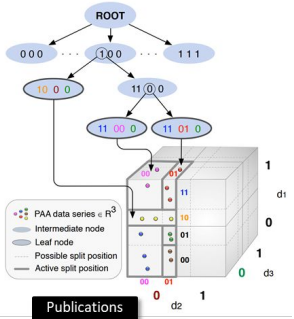
LSH



Publications

Indyk et al. STOC' 98

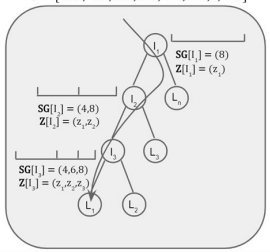
iSAX



Publications

Shieh-KDD'08

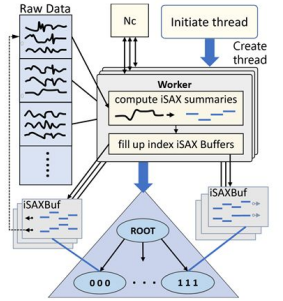
DSTREE



Publications

Wang-PVLDB'13

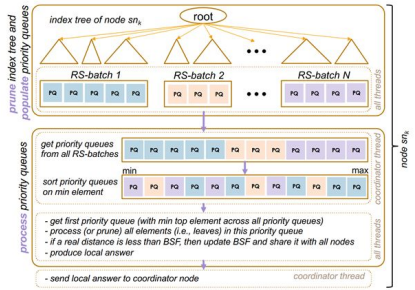
MESSI



Publications

Peng et al. VLDBJ' 21

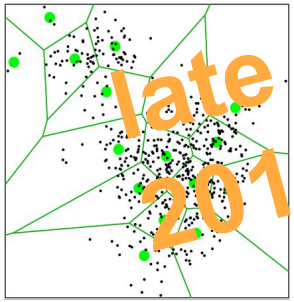
Odyssey



Publications

Chatzakis et al. PVLDB' 23

IVF

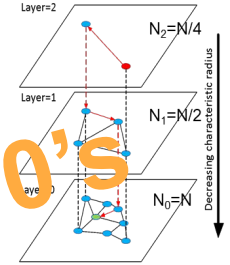


late 2010's

Publications

Sivic et al. ICCV' 03

HNSW

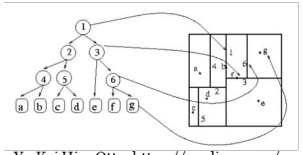


Publications

Malkov et al. TPAMI' 20 Arxiv'16

A Brief History of Vector Similarity Search

KD-Tree

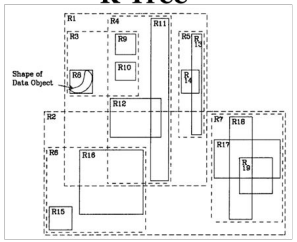


Yu Kai Him Otto, https://medium.com/

Publications

Bentley-CACM'75

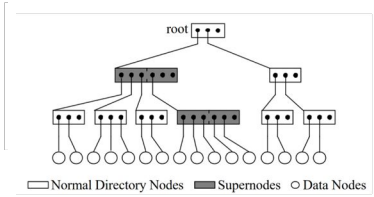
R-Tree



Publications

Guttman-SIGMOD'84

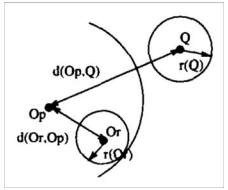
X-Tree



Publications

Berchtold et al - VLDB'96

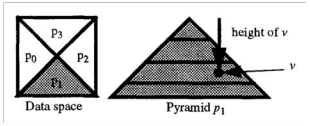
M-Tree



Publications

Ciaccia et al- VLDB'97

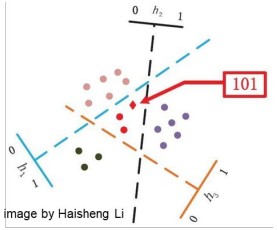
Pyramid technique



Publications

Berchtold et al - SIGMOD'98

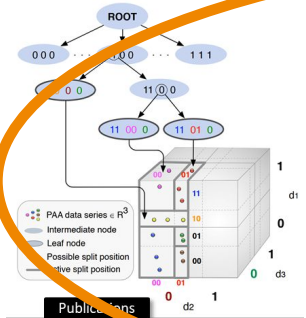
LSH



Publications

Indyk et al. STOC' 98

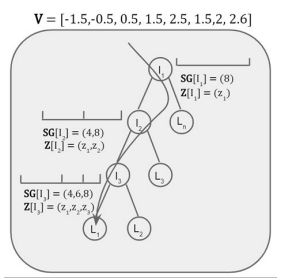
iSAX



Publications

Shieh-KDD'08

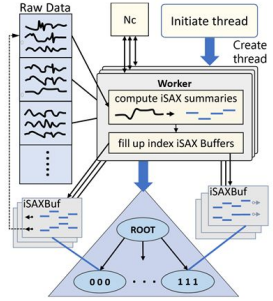
DSTREE



Publications

Wang-PVLDB'13

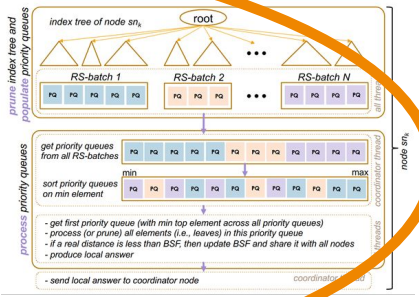
MESSI



Publications

Peng et al. VLDBJ' 21

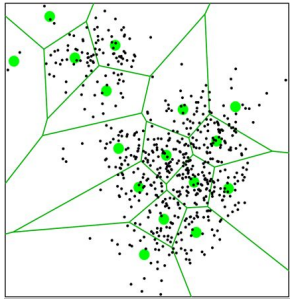
Odyssey



Publications

Chatzakis et al. PVLDB' 23

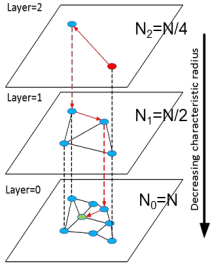
IVF



Publications

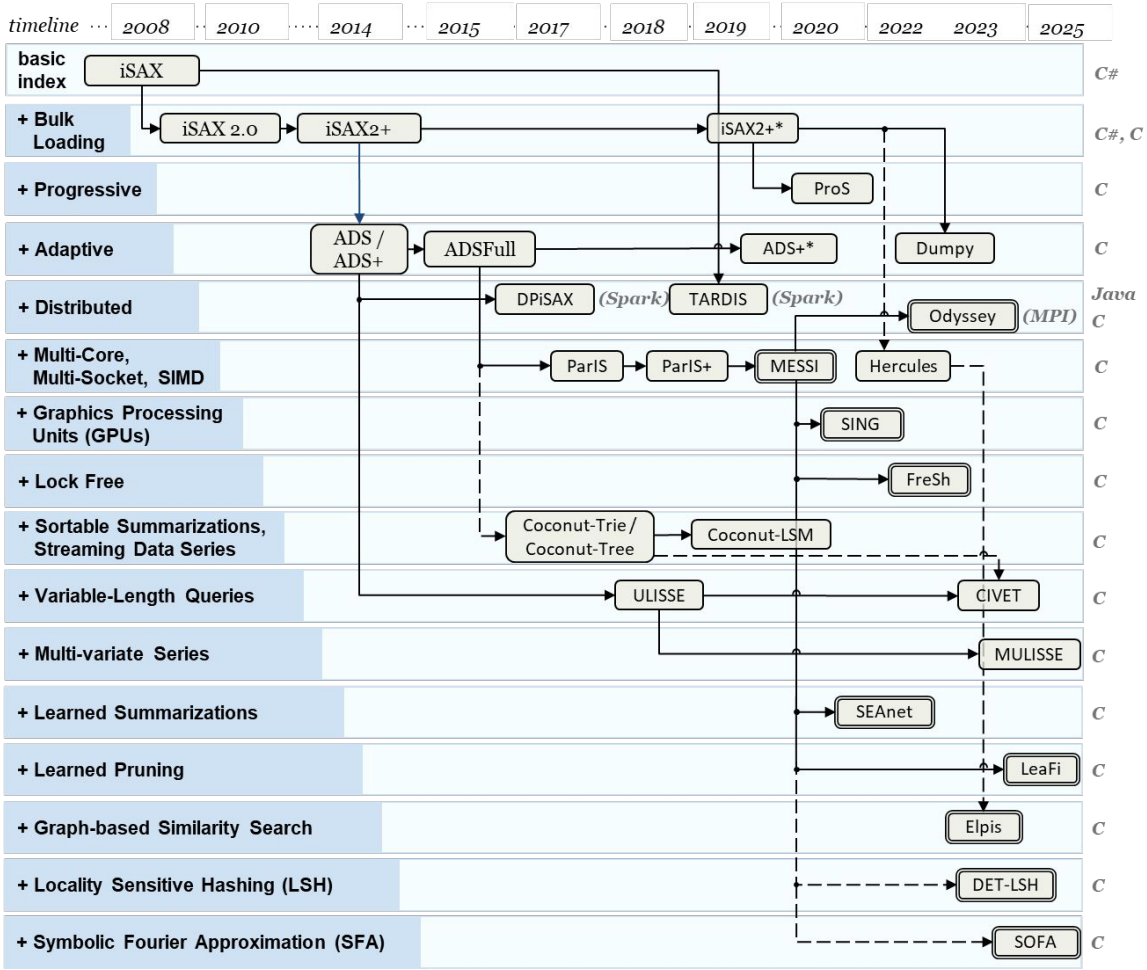
Sivic et al. ICCV' 03

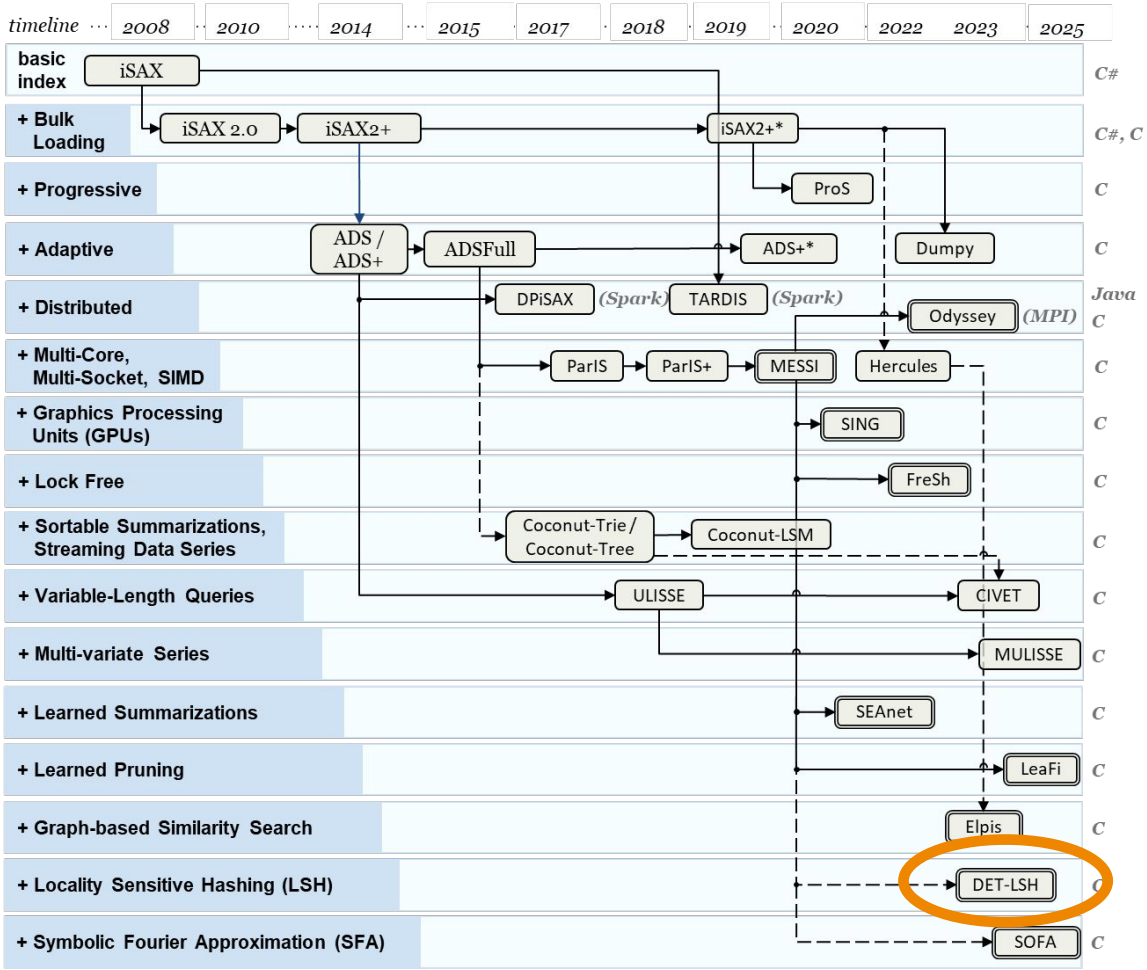
HNSW



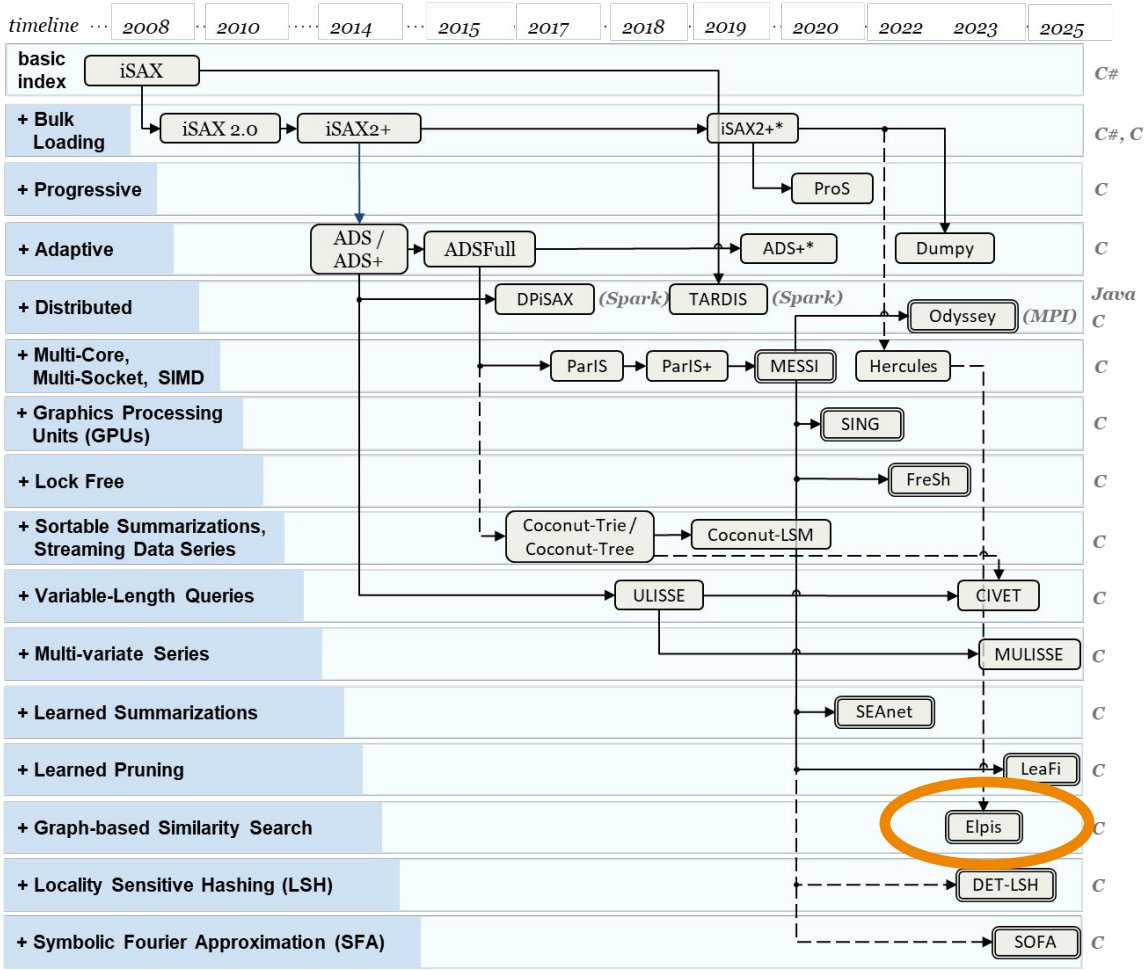
Publications

Malkov et al. TPAMI' 20 Arxiv'16

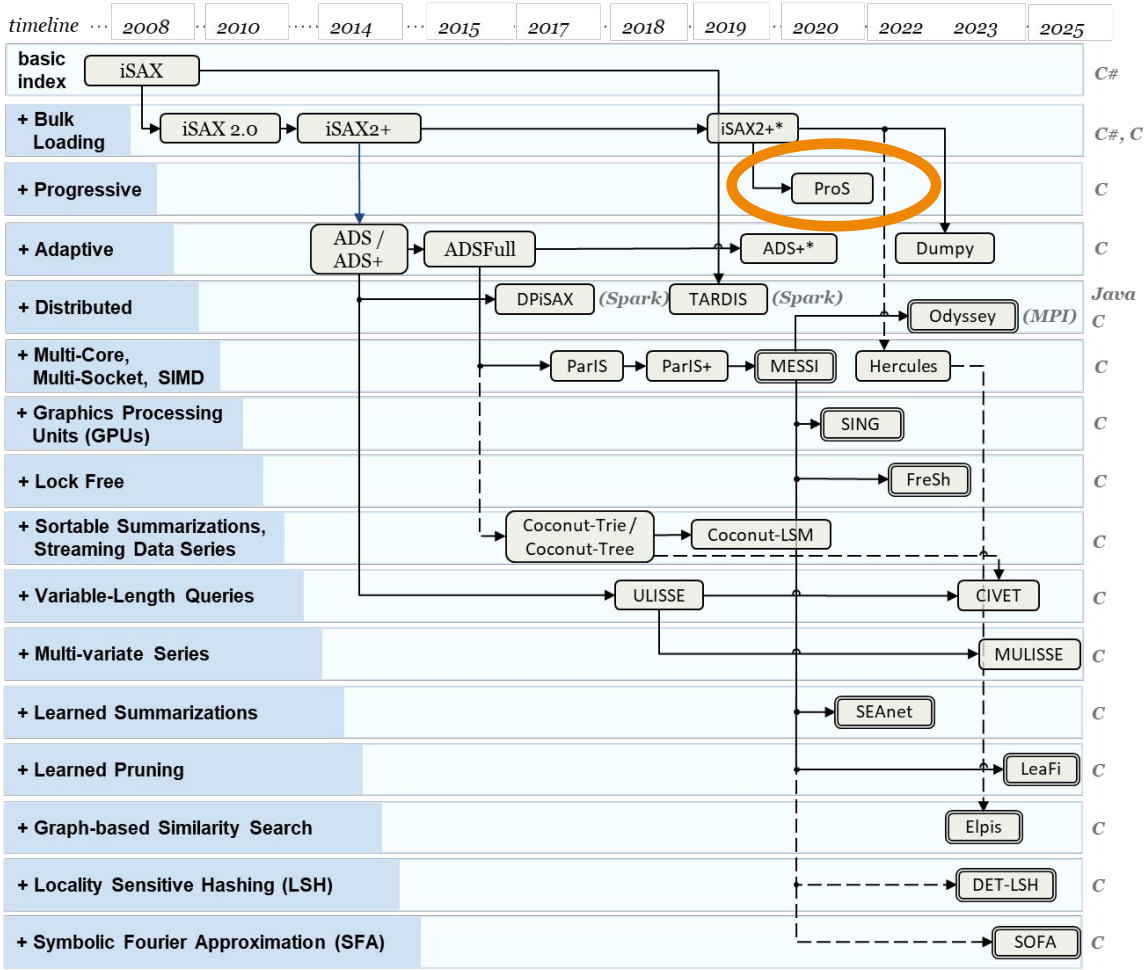

timeline
2008
2010
2014
2015
2017
2018
2019
2020
2022
2023
2025
basic index
iSAX
C#
+ Bulk Loading
iSAX 2.0
iSAX2+
iSAX2+*
C#, C
+ Progressive
ProS
C
+ Adaptive
ADS / ADS+
ADSFull
ADS+*
Dumpy
C
+ Distributed
DPiSAX
(Spark)
TARDIS
(Spark)
Odyssey
(MPI)
Java
C
+ Multi-Core, Multi-Socket, SIMD
ParIS
ParIS+
MESSI
Hercules
C
+ Graphics Processing Units (GPUs)
SING
C
+ Lock Free
FreSh
C
+ Sortable Summarizations, Streaming Data Series
Coconut-Trie / Coconut-Tree
Coconut-LSM
C
+ Variable-Length Queries
ULISSE
CIVET
C
+ Multi-variate Series
MULISSE
C
+ Learned Summarizations
SEAnet
C
+ Learned Pruning
LeaFi
C
+ Graph-based Similarity Search
Elpis
C
+ Locality Sensitive Hashing (LSH)
DET-LSH
C
+ Symbolic Fourier Approximation (SFA)
SOFA
C



DET-LSH hybrid solution
δ-ε-approximate
iSAX + LSH



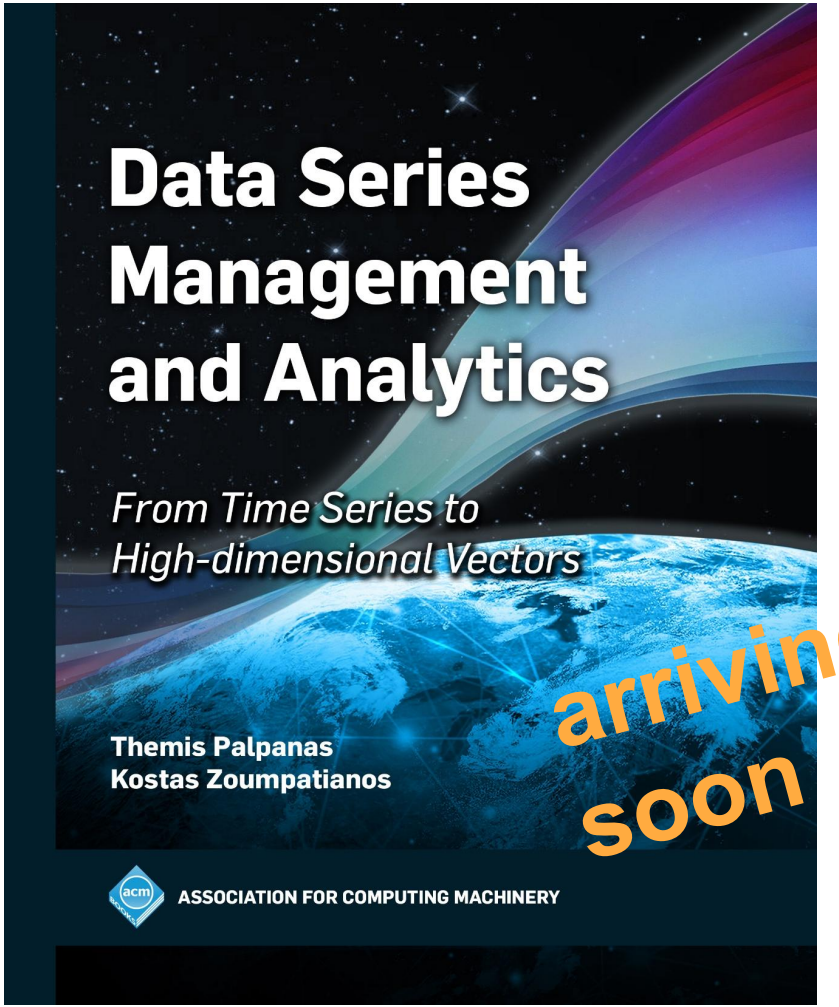
ELPIS
ng-approximate
DSTREE + HNSW



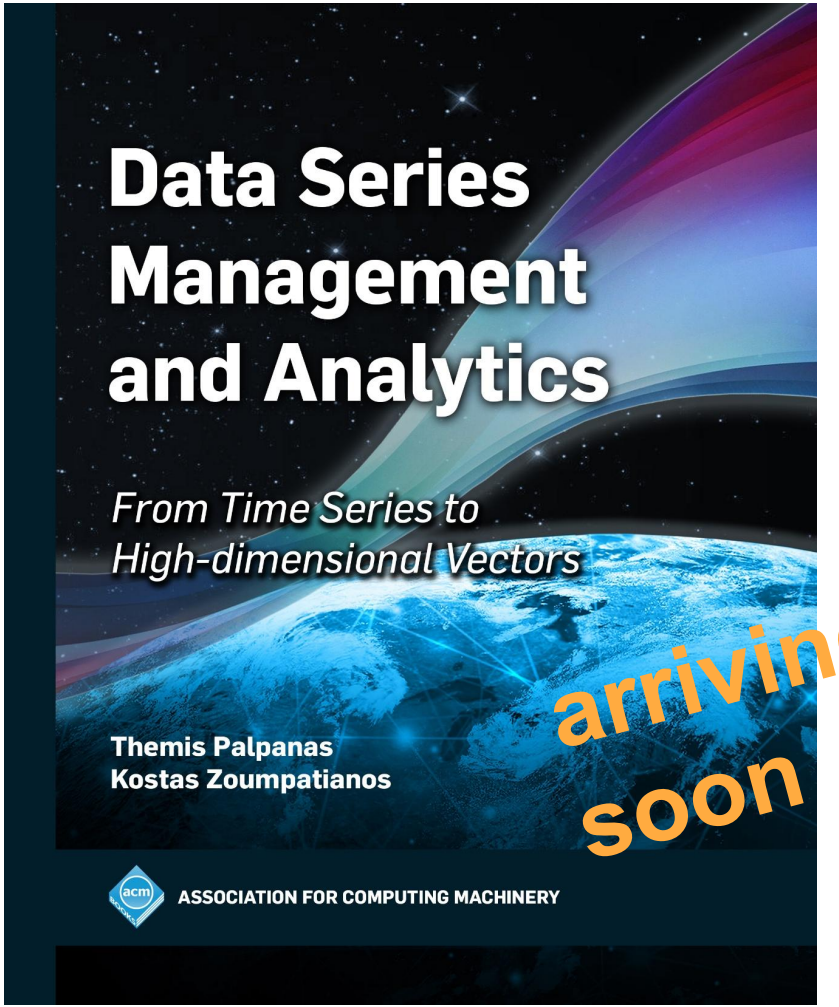
ProS

progressive query answering

useful for ANNS early termination



arriving soon



arriving soon

DaiSy library for Fast Exact Similarity Search

Vector Similarity Search Communities

- high-d vector similarity search relevant to many communities
  - data management
  - information retrieval / text search
  - parallel and distributed computing
  - time series
  - machine learning / deep learning
  - computer architecture

Vector Similarity Search Communities

- high-d vector similarity search relevant to many communities
  - data management
  - information retrieval / text search
  - parallel and distributed computing
  - time series
  - machine learning / deep learning
  - computer architecture

- cross-fertilization of ideas is useful
- bringing together these communities
  - **IEEE Bulletin on Data Engineering** Special Issues: **High-Dimensional Vector Similarity Search**

September 2023

From Time Series to Vector Databases

September 2024

Role of Machine Learning and Future Perspectives

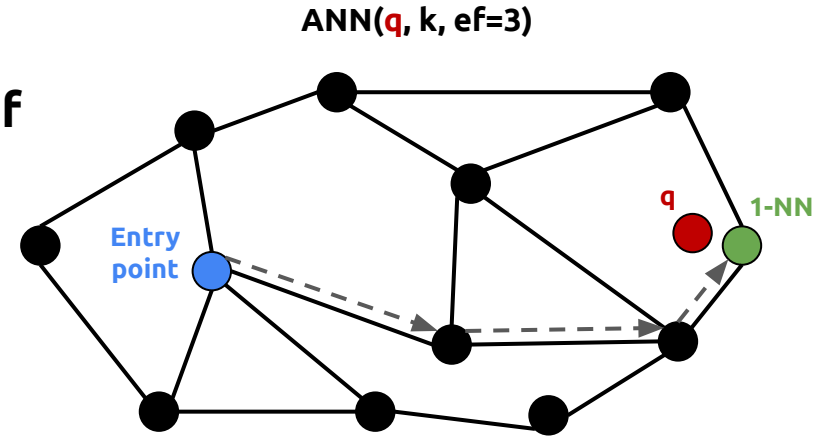
Vector Approximate NN Search (ANNS)
[ng-approximate search]
[vector search]

- we focus on approximate NN search with no guarantees
- study solutions based on:
  - k-NN-Graphs
  - Inverted Files

Malkov et al. "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs." PAMI 2018.
Fu et al. "Fast approximate nearest neighbor search with the navigating spreading-out graph." PVLDB 2017.
Subramanya et al. "Diskann: Fast accurate billion-point nearest neighbor search on a single node." NeurIPS 2019.

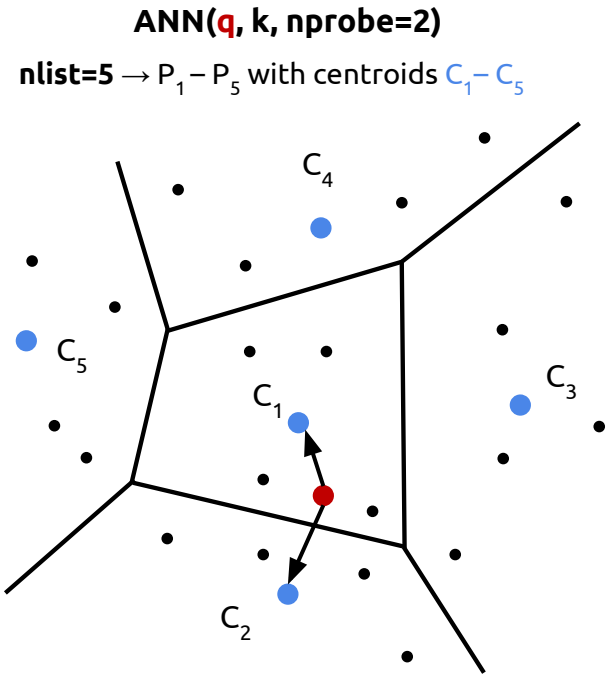
Vector Search with kNN-Graph Indexes

- Create a proximity graph index over the vector collections
- Greedy search from **entry point** with effort **ef**
- Variations:
  - HNSW: Hierarchical
  - Vamana: Relaxed edge pruning
  - NSG: Optimized search pruning



Vector Search with Inverted File (IVF) Indexes

- **Partition** the dataset using clustering with **nlist centroids**
- Visit the closest **nprobe** partitions of a query
- Variations:
  - **FAISS-IVF**: Efficient IVF implementation
  - **ScaNN**: Hierarchical IVF, optimizations in vector assignment



Vector Search Resources

- comparative evaluations
  - Echihabi et al. “Return of the Lernaean Hydra: Experimental Evaluation of Data Series Approximate Similarity Search” PVLDB 2019
  - Azizi et al. “Graph-Based Vector Search: An Experimental Evaluation of the State-of-the-Art” SIGMOD 2025
  - Kang et al. “BigVectorBench: Heterogeneous Data Embedding and Compound Queries are Essential in Evaluating Vector Databases” PVLDB 2025
  - Big-ANN Benchmark <https://big-ann-benchmarks.com/>
- surveys
  - Wang et al. “Graph- and Tree-based Indexes for High-dimensional Vector Similarity Search: Analyses, Comparisons, and Future Directions” IEEE Data Eng. Bull. 2023
  - Pan et al. “Survey of Vector Database Management Systems” VLDBJ 2024
- tutorials
  - Qin et al. “High-Dimensional Similarity Qyery Processing for Data Science” KDD 2021
  - Chronis et al. “Filtered Vector Search: State-of-the-art and Research Opportunities” VLDB 2025

Outline of this Tutorial

- Parallel and Distributed ANNS
  - exploit available hardware

Outline of this Tutorial

- Parallel and Distributed ANNS
  - exploit available hardware
- Streaming ANNS
  - employ incremental solutions

Outline of this Tutorial

- Parallel and Distributed ANNS
  - exploit available hardware
- Streaming ANNS
  - employ incremental solutions
- Early Termination ANNS
  - avoid wasted effort

Outline of this Tutorial

- Parallel and Distributed ANNS
  - exploit available hardware
- Streaming ANNS
  - employ incremental solutions
- Early Termination ANNS
  - avoid wasted effort
- Quality Evaluation of ANNS
  - select right measures

Parallel and Distributed Vector Search

Many thanks to Peng et al. (iQAN), Azizi et al. (ELPIS), Deng et al. (Pyramid), Dang et al. (BatANN), Adams et al. (SPIRE), Zhi et al. (CoTra), Li et al. (Tagore), Xu et al. (Harmony), Mohoney et al. (Quake), Zhang et al. (Auncel), Gottesbüren et al. (Graph Partitioning), who provided supporting material for this part of the tutorial.

Outline

Parallel Vector Search

- Goals and Challenges
- Parallelism for Fast Index Construction
- Parallelism for Low Latency Query Answering

Outline

Parallel Vector Search

- Goals and Challenges
- Parallelism for Fast Index Construction
- Parallelism for Low Latency Query Answering

Distributed Vector Search

- Goals and Challenges
- k-NN Graph-based Distributed Systems Overview
- IVF-Based Distributed Systems Overview

Parallel Vector Search

- Sequential vector search fails to saturate modern hardware, underutilizes cores and vector units, and quickly becomes memory-bound as dataset size grows.

Parallel Vector Search

- Sequential vector search fails to saturate modern hardware, underutilizes cores and vector units, and quickly becomes memory-bound as dataset size grows.

- Exploiting multi-core CPUs and GPUs through parallel execution increase throughput, reduce tail latency, and fully utilize available hardware resources.

Parallel Vector Search

- Sequential vector search fails to saturate modern hardware, underutilizes cores and vector units, and quickly becomes memory-bound as dataset size grows.

- Exploiting multi-core CPUs and GPUs through parallel execution increase throughput, reduce tail latency, and fully utilize available hardware resources.

- Parallel execution introduces synchronization overhead, contention on shared index structures, load imbalance across threads, and cache/NUMA locality issues.

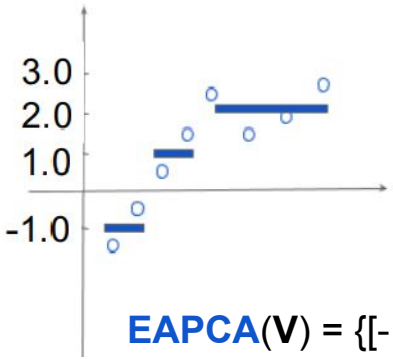
Parallel Vector Search - ELPIS [1/3]

- **Goal:** Decrease build indexing time and reach good throughput using parallelism in both build indexing in-memory ng-approximate vector search.

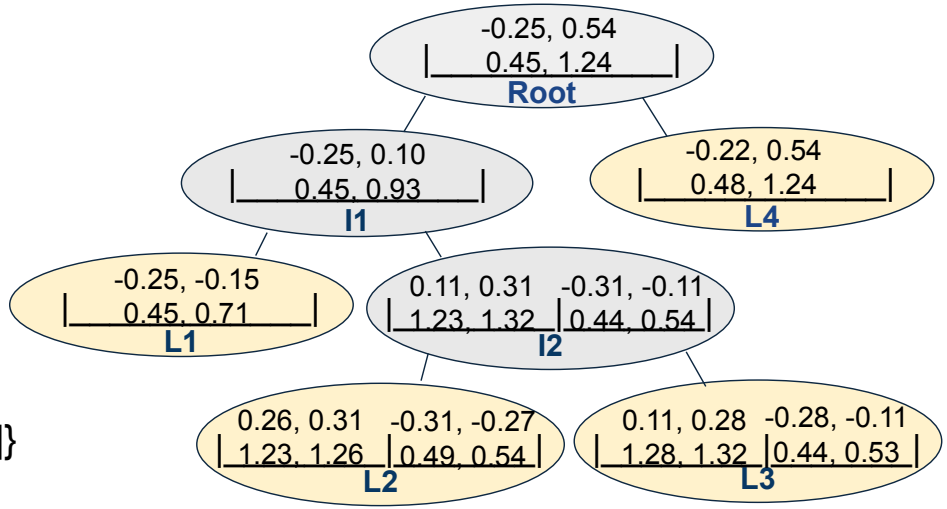
Parallel Vector Search - ELPIS [1/3]

- **Goal:** Decrease build indexing time and reach good throughput using parallelism in both build indexing in-memory ng-approximate vector search.
- **Contributions:** Hybrid solution that combines a shallow tree at the top with a knn-graph, inside each leaf and uses EAPCA data series summarization.

V = {-1.5,-0.5,0.5,1.5,2.5,1.5,2,2.6}

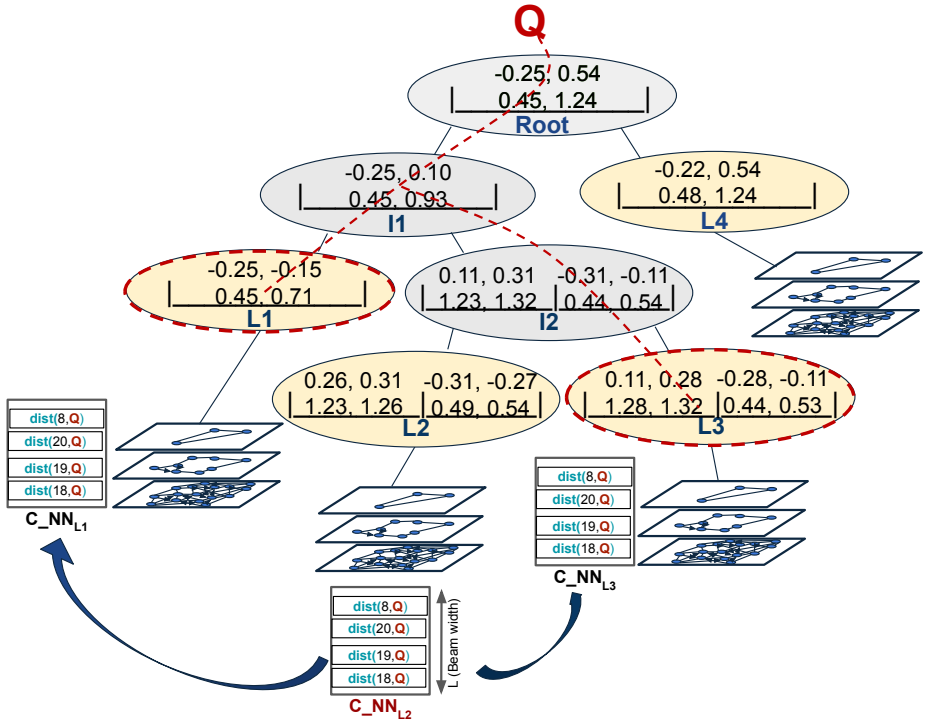


EAPCA(**V**) = {[-1,0.5],[1,0.5],[2.15,0.44]}



Parallel Vector Search - ELPIS [2/3]

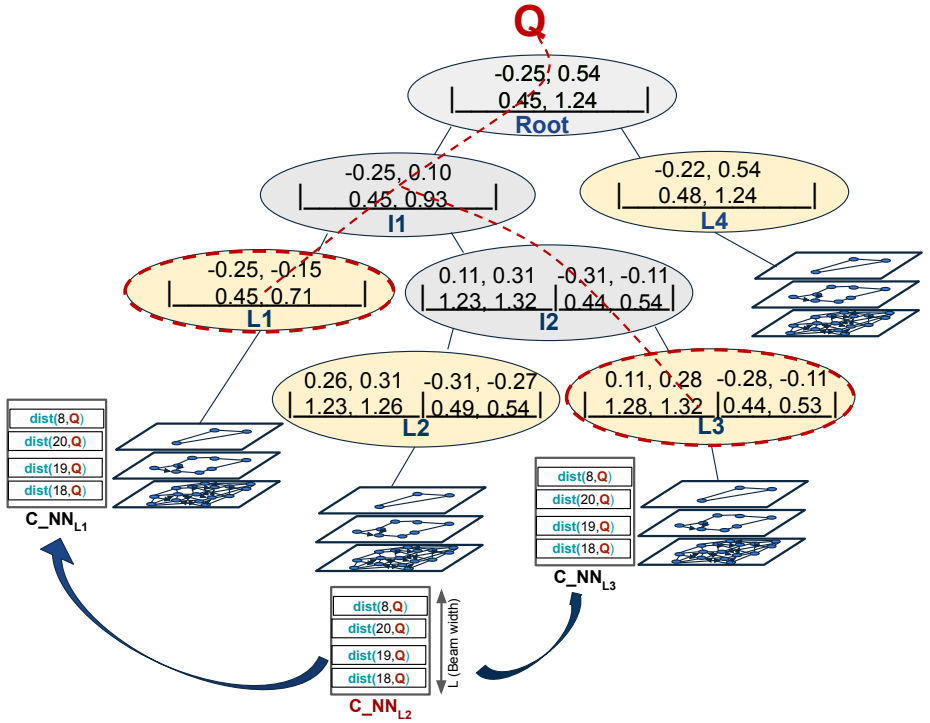
- **Build Index:** concurrently constructs graphs using *nw1* workers, with each worker building HNSW graph within each leaf in parallel using *nw2* workers.



Parallel Vector Search - ELPIS [2/3]

- **Build Index:** concurrently constructs graphs using *nw1* workers, with each worker building HNSW graph within each leaf in parallel using *nw2* workers.

- **Query Answering:** selects up to (nprobes - 1) additional leaves, and *nw* worker explore them in parallel.

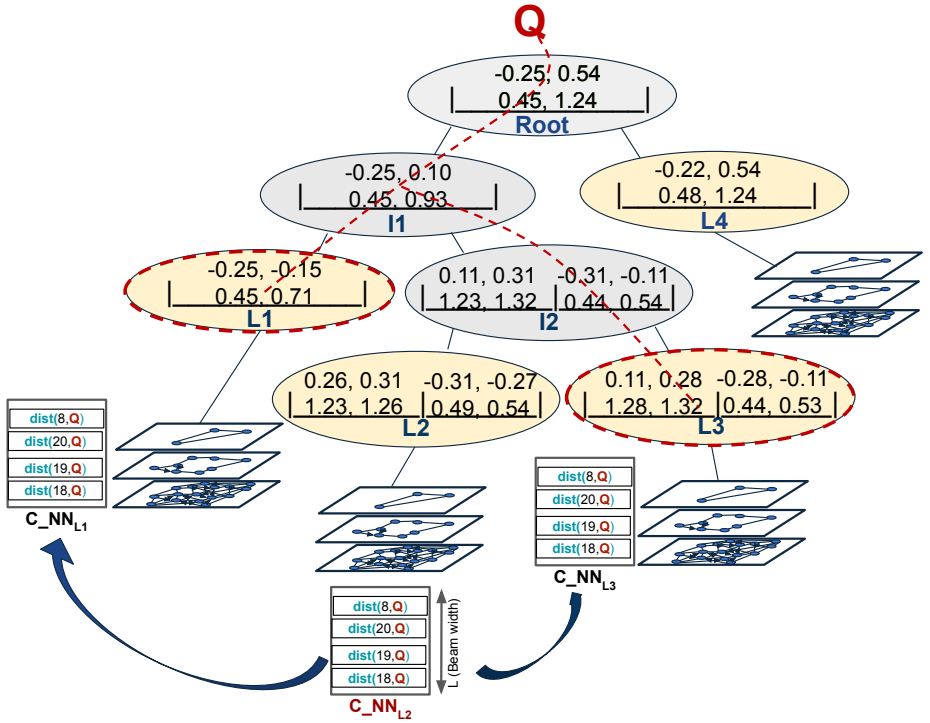


Parallel Vector Search - ELPIS [2/3]

- **Build Index:** concurrently constructs graphs using *nw1* workers, with each worker building HNSW graph within each leaf in parallel using *nw2* workers.

- **Query Answering:** selects up to (nprobes - 1) additional leaves, and *nw* worker explore them in parallel.

up to 4x faster in index building
up to 10x faster in query answering



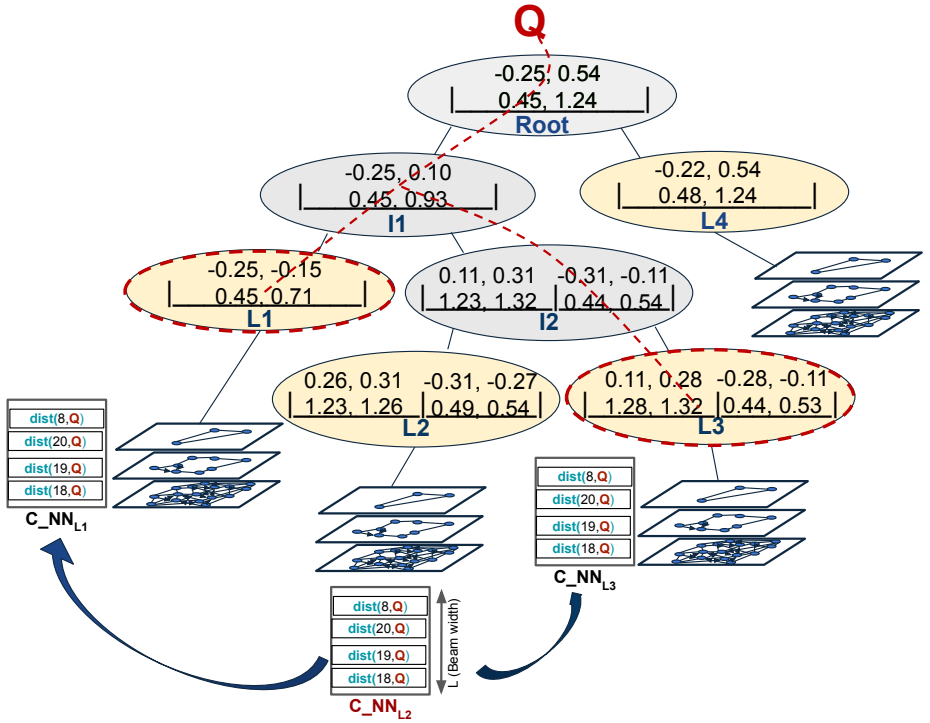
Parallel Vector Search - ELPIS [2/3]

- **Build Index:** concurrently constructs graphs using *nw1* workers, with each worker building HNSW graph within each leaf in parallel using *nw2* workers.

- **Query Answering:** selects up to (nprobes - 1) additional leaves, and *nw* worker explore them in parallel.

up to 4x faster in index building
up to 10x faster in query answering

sensitive to number of clusters.



Parallel Vector Search - ELPIS [3/3]

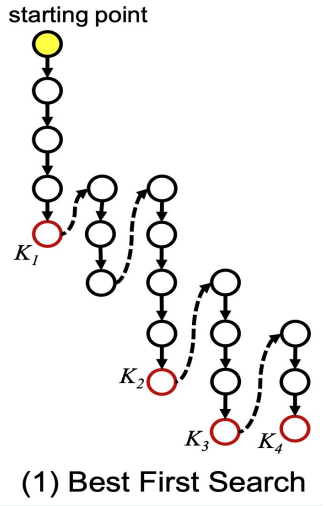
Related Systems

- **ParlayANN**: It’s a parallel and deterministic framework for graph-based ANNS. It introduces techniques such as prefix-doubling batch insertion that builds the graph incrementally in exponentially growing batches.

- **Tagore**: It’s a fast library accelerated by GPUs for graph indexing. It introduces GNN-Descent, a GPU-specific algorithm for efficient k-Nearest Neighbor (k-NN) graph initialization made by a two-phase descent procedure.

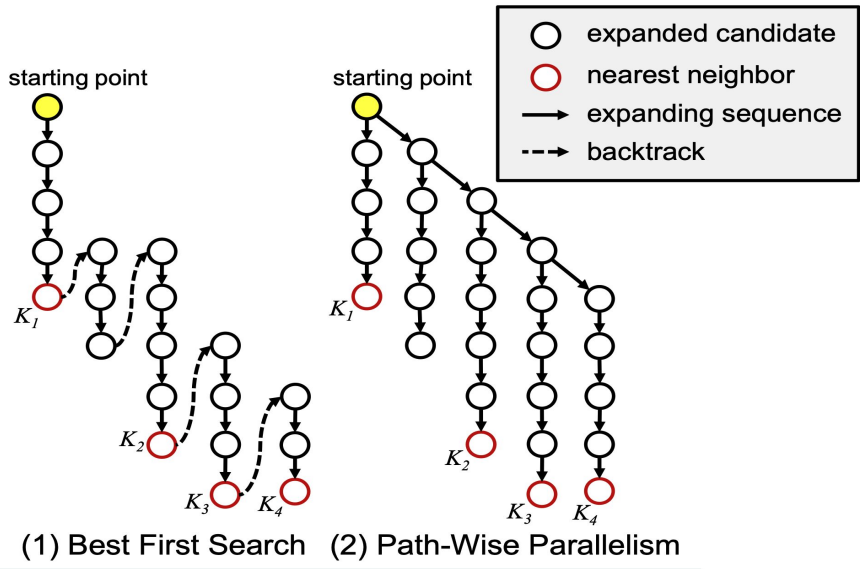
Parallel Vector Search - iQAN

- **Goal:** Given a graph index, how to reduce search latency of a single query by parallelizing Best First Search?



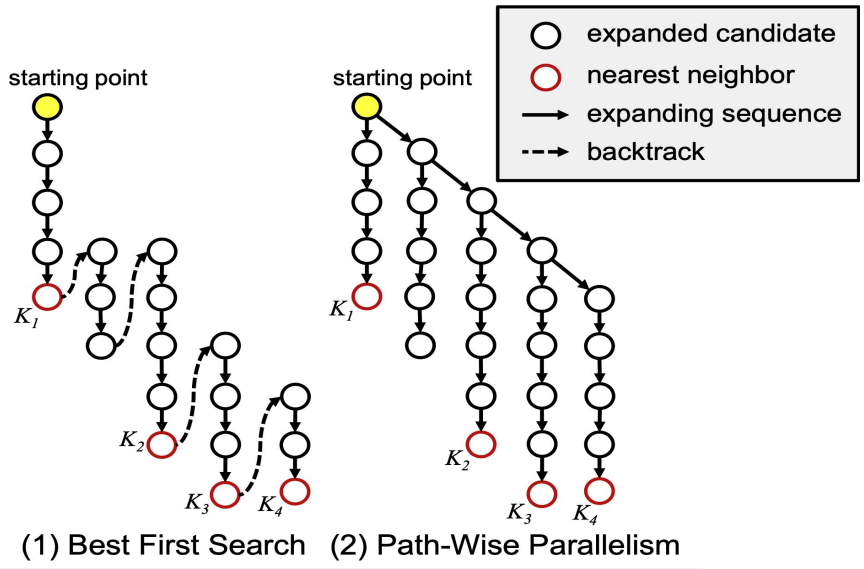
Parallel Vector Search - iQAN

- **Goal:** Given a graph index, how to reduce search latency of a single query by parallelizing Best First Search?
- **Contribution:** **Path-Wise Parallelism** in contrast to Edge-Wise Parallelism.



Parallel Vector Search - iQAN

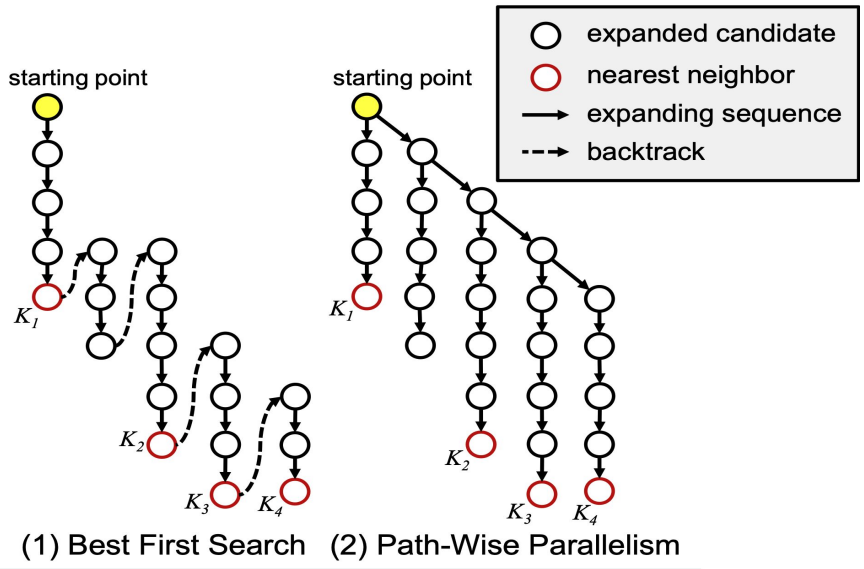
- **Goal:** Given a graph index, how to reduce search latency of a single query by parallelizing Best First Search?
- **Contribution:** **Path-Wise Parallelism** in contrast to Edge-Wise Parallelism.



Reduces iterations depths for convergence.

Parallel Vector Search - iQAN

- **Goal:** Given a graph index, how to reduce search latency of a single query by parallelizing Best First Search?
- **Contribution:** **Path-Wise Parallelism** in contrast to Edge-Wise Parallelism.

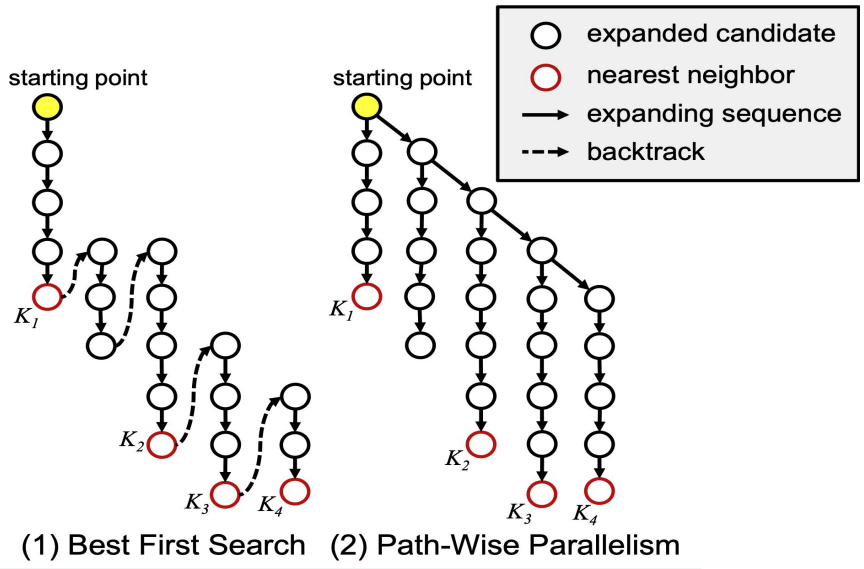


Reduces iterations depths for convergence.

Increases distance computations: unnecessary expansions.

Parallel Vector Search - iQAN

- **Goal:** Given a graph index, how to reduce search latency of a single query by parallelizing Best First Search?
- **Contribution:** **Path-Wise Parallelism** in contrast to Edge-Wise Parallelism.



Reduces iterations depths for convergence.

Increases distance computations: unnecessary expansions.

Related Systems

FAISS GPU: The core contribution is a highly optimized in-register k-selection algorithm (WarpSelect).

Mohoney et al. “Quake: adaptive indexing for vector search.” OSDI 2025.

Parallel Vector Search - Quake

Goal: An adaptive approach that exploits NUMA processors for low latency.

- **NUMA-aware data placement.**
  Partitions are distributed across NUMA nodes and bound to specific cores.

Mohoney et al. “Quake: adaptive indexing for vector search.” OSDI 2025.

Parallel Vector Search - Quake

Goal: An adaptive approach that exploits NUMA processors for low latency.

- **NUMA-aware data placement.**
  Partitions are distributed across NUMA nodes and bound to specific cores.
- **NUMA-aware Work Scheduling.**
  Queries are scheduled to threads based on partition locality.

Mohoney et al. “Quake: adaptive indexing for vector search.” OSDI 2025.

Parallel Vector Search - Quake

Goal: An adaptive approach that exploits NUMA processors for low latency.

- **NUMA-aware data placement.**
  Partitions are distributed across NUMA nodes and bound to specific cores.
- **NUMA-aware Work Scheduling.**
  Queries are scheduled to threads based on partition locality.
- **NUMA-aware query execution with APS.**
  Worker threads scan local partitions while a coordinator merges results and estimates recall.

Mohoney et al. “Quake: adaptive indexing for vector search.” OSDI 2025.

Parallel Vector Search - Quake

Goal: An adaptive approach that exploits NUMA processors for low latency.

- **NUMA-aware data placement.**
  Partitions are distributed across NUMA nodes and bound to specific cores.
- **NUMA-aware Work Scheduling.**
  Queries are scheduled to threads based on partition locality.
- **NUMA-aware query execution with APS.**
  Worker threads scan local partitions while a coordinator merges results and estimates recall.

Minimizes remote memory accesses via data-local execution.

Mohoney et al. “Quake: adaptive indexing for vector search.” OSDI 2025.

Parallel Vector Search - Quake

Goal: An adaptive approach that exploits NUMA processors for low latency.

- **NUMA-aware data placement.**
  Partitions are distributed across NUMA nodes and bound to specific cores.
- **NUMA-aware Work Scheduling.**
  Queries are scheduled to threads based on partition locality.
- **NUMA-aware query execution with APS.**
  Worker threads scan local partitions while a coordinator merges results and estimates recall.

Minimizes remote memory accesses via data-local execution.

NUMA cores imbalance in case of skewed workloads.

Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.

Distributed Vector Search

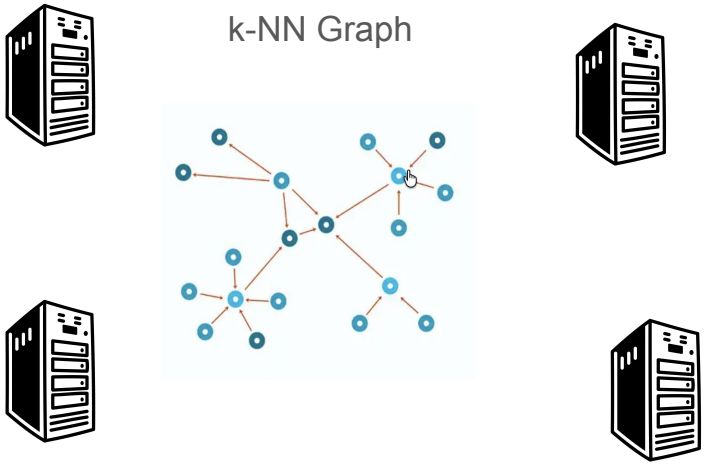
- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.

Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.

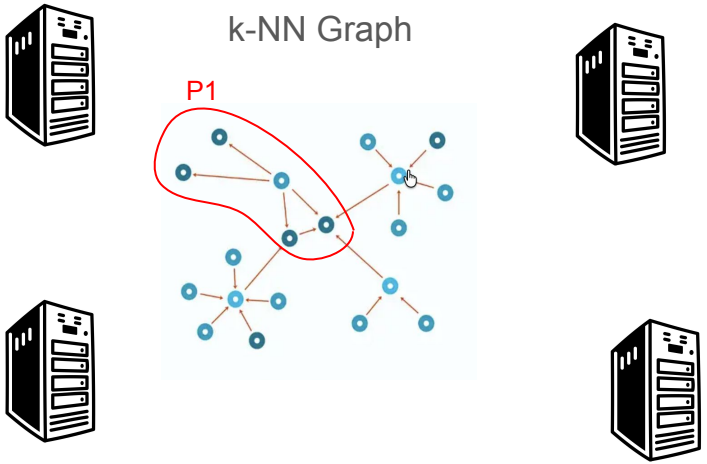
Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.



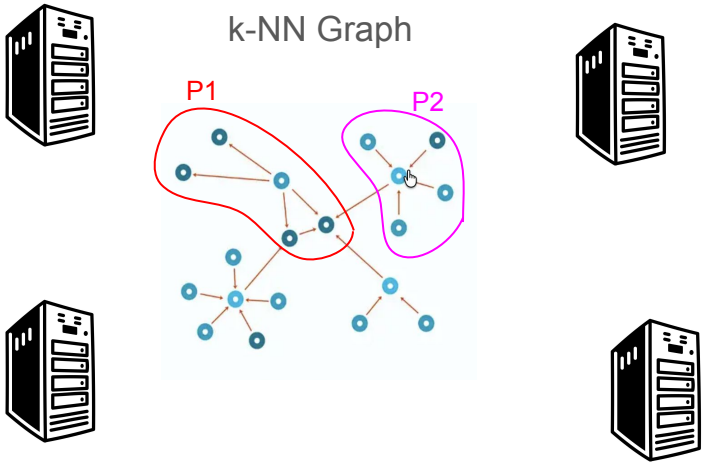
Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.



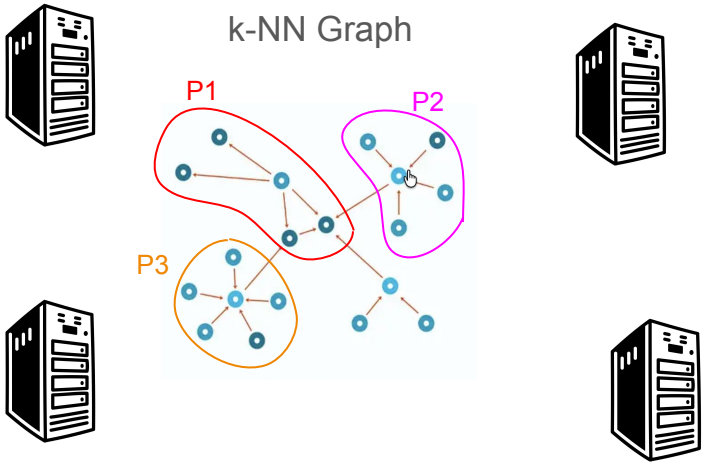
Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.



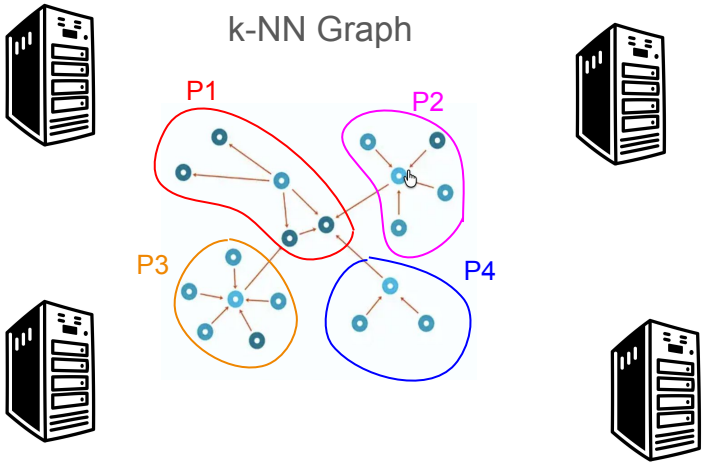
Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.



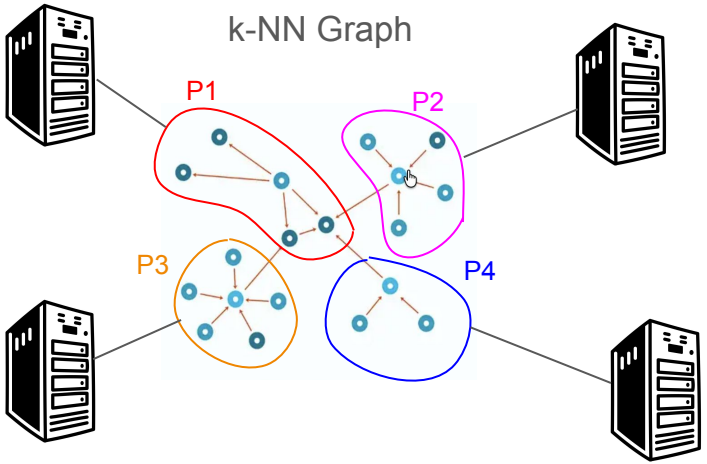
Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.



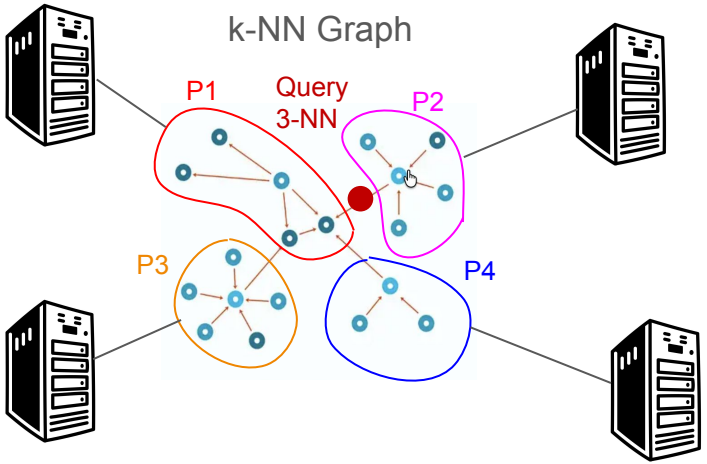
Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.



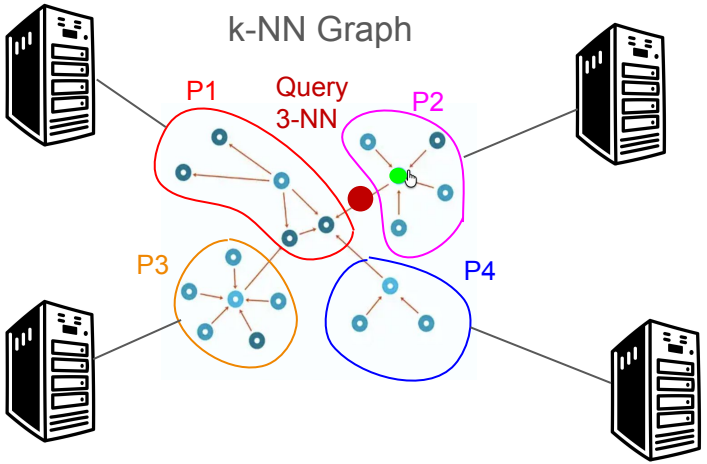
Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.



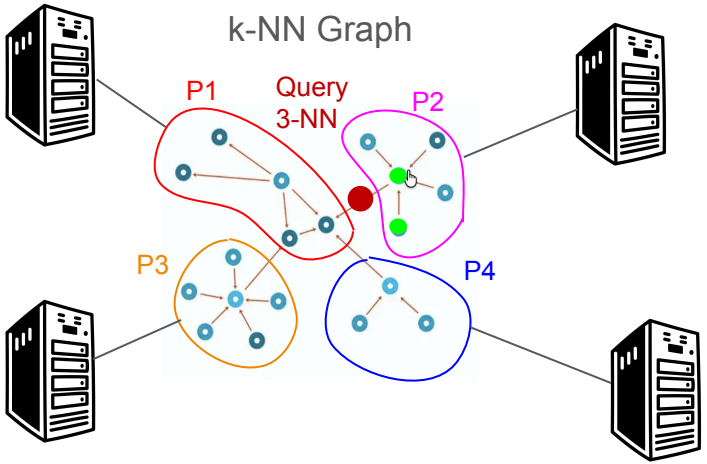
Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.



Distributed Vector Search

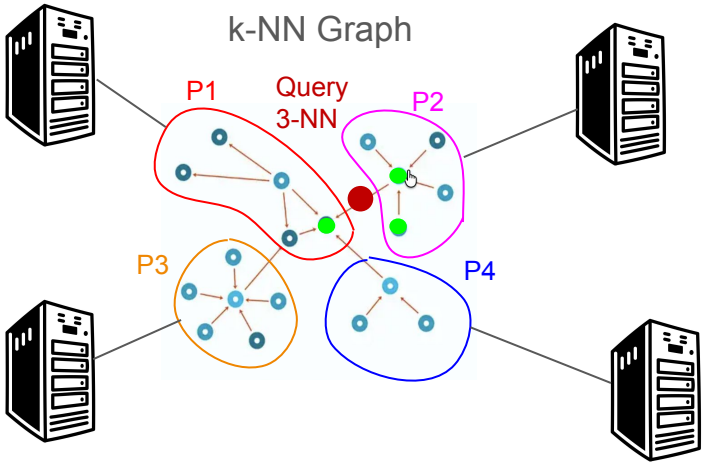
- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.



Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.

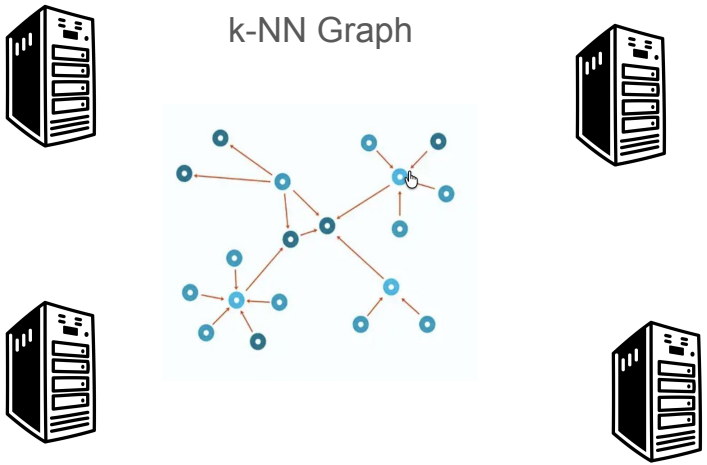
To reach high recall, we may need to search in multiple nodes.



Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.

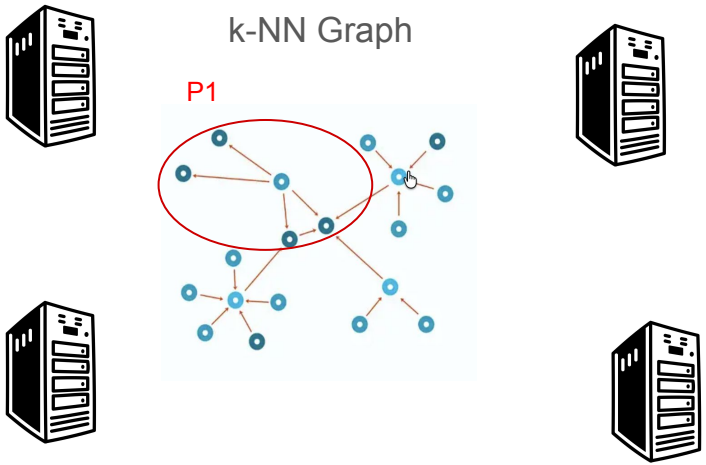
We need replication.



Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.

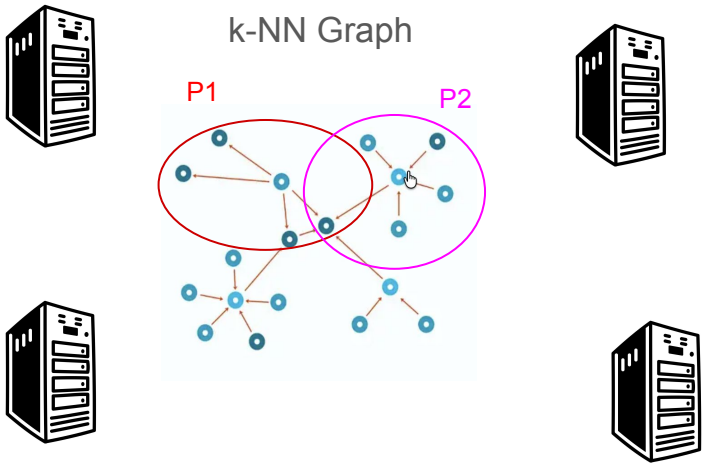
We need replication.



Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.

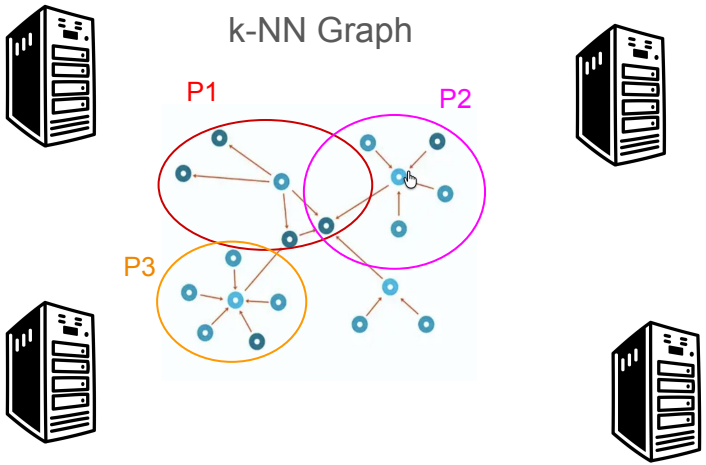
We need replication.



Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.

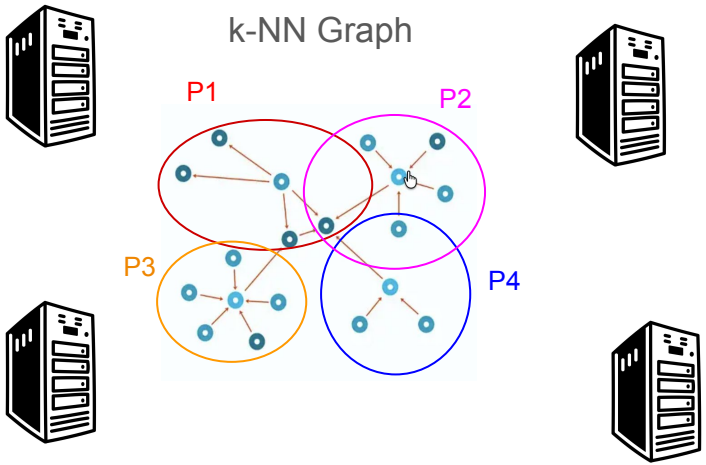
We need replication.



Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.

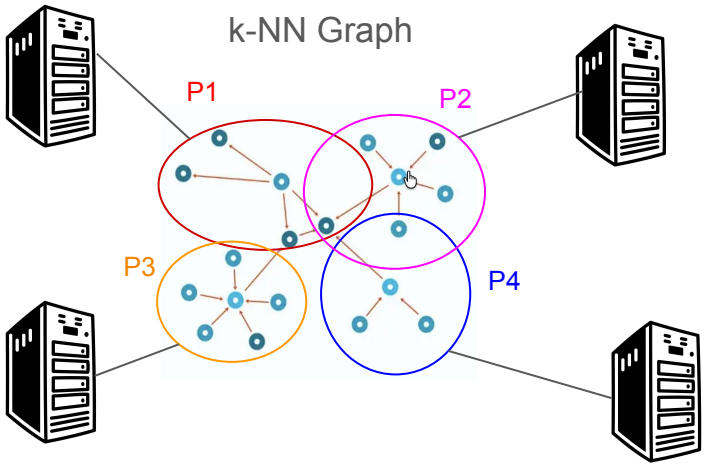
We need replication.



Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.

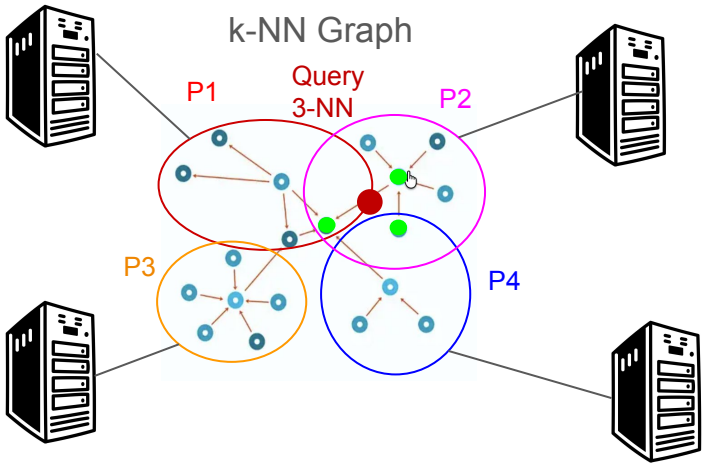
We need replication.



Distributed Vector Search

- The rapid growth of high-dimensional vector datasets make single-machine processing insufficient.
- Single-node systems are constrained by memory capacity, memory bandwidth, and core-level parallelism.
- Partitioning vectors across nodes introduces trade-offs between load balance, recall, and communication overhead.

Query answered by searching only in P2!



Distributed Vector Search - CoTra [1/3]

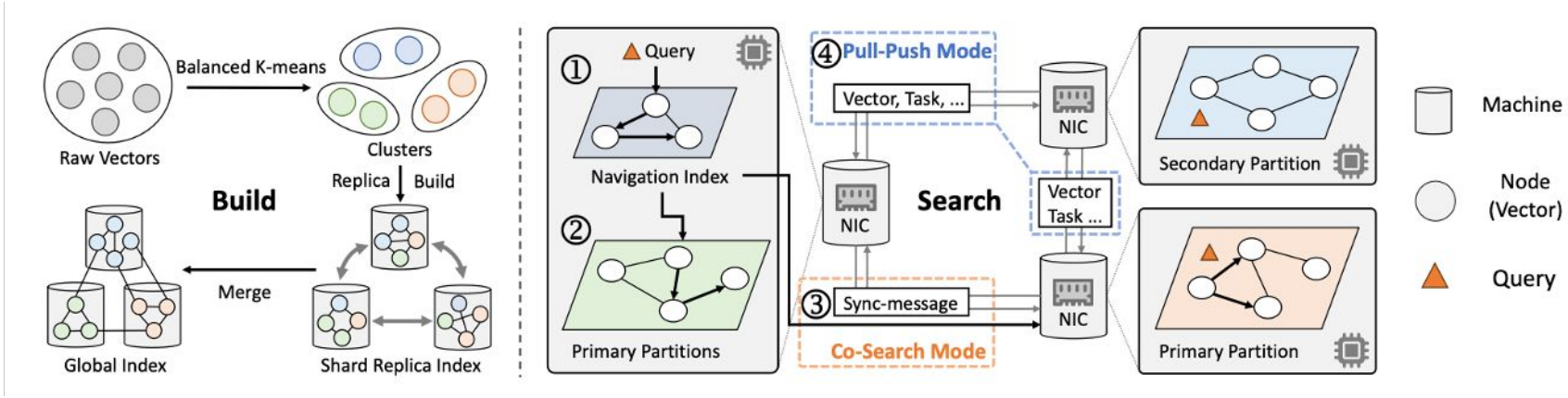
Goal: Resolve the tension between computation and communication to achieve near-linear scalability.

Distributed Vector Search - CoTra [1/3]

Goal: Resolve the tension between computation and communication to achieve near-linear scalability.

Contributions:

- **Collaborative Search:** Differentiated execution modes (Co-Search & Push-Pull).

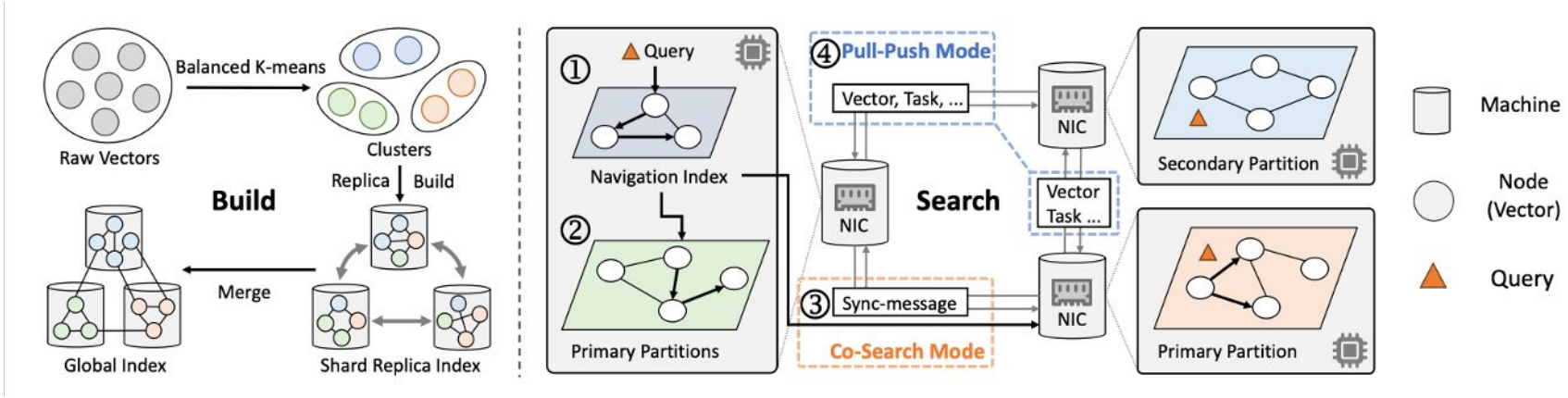


Distributed Vector Search - CoTra [1/3]

Goal: Resolve the tension between computation and communication to achieve near-linear scalability.

Contributions:

- **Collaborative Search:** Differentiated execution modes (Co-Search & Push-Pull).
- **RDMA Integration:** Low-latency, high-throughput communication.



Distributed Vector Search - CoTra [2/3]

Partitioning & Navigation

- **Balanced K-means Partitioning:** Each machine keeps a similar number of vectors and the adjacency lists of these vectors in the graph index.

Distributed Vector Search - CoTra [2/3]

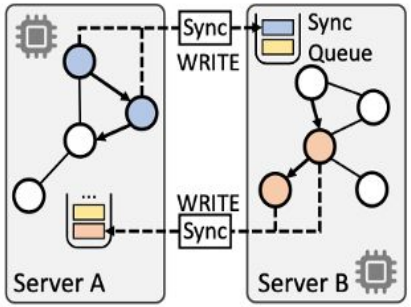
Partitioning & Navigation

- **Balanced K-means Partitioning:** Each machine keeps a similar number of vectors and the adjacency lists of these vectors in the graph index.
- **Navigation Index:** A small sample (e.g., 1%) of the dataset is used to build a replicated navigation index (proximity graph). Quickly identifies "Primary" vs. "Secondary" partitions for each query.

Distributed Vector Search - CoTra [3/3]

Collaborative Traversal

- **Co-Search Mode (Primary Partitions):** Machines hosting many relevant vectors run local search.

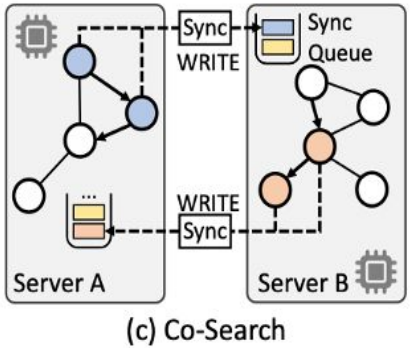


(c) Co-Search

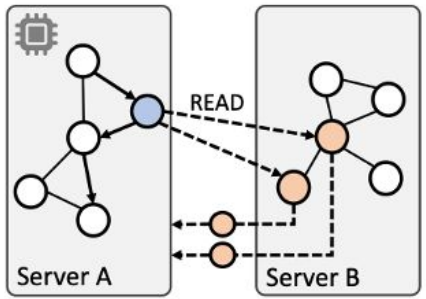
Distributed Vector Search - CoTra [3/3]

Collaborative Traversal

- **Co-Search Mode (Primary Partitions):** Machines hosting many relevant vectors run local search.
- **Pull-Push Mode (Secondary Partitions):** Machines hosting few relevant vectors act as "workers."



(c) Co-Search



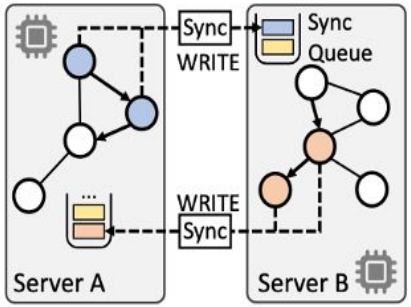
(a) Pull Data

Distributed Vector Search - CoTra [3/3]

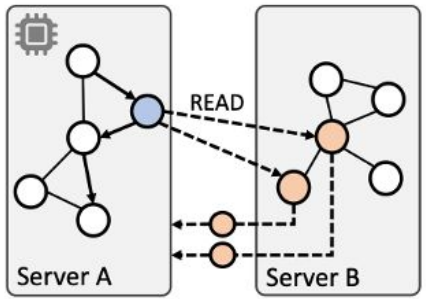
Collaborative Traversal

- **Co-Search Mode (Primary Partitions):** Machines hosting many relevant vectors run local search.
- **Pull-Push Mode (Secondary Partitions):** Machines hosting few relevant vectors act as "workers."

Near-Linear Scalability



(c) Co-Search

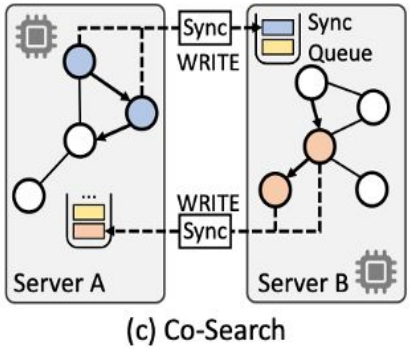


(a) Pull Data

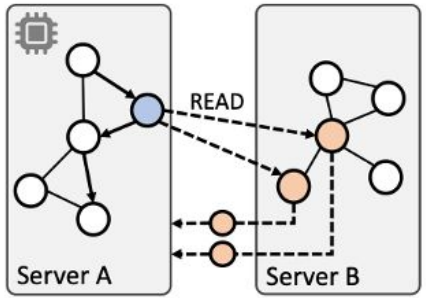
Distributed Vector Search - CoTra [3/3]

Collaborative Traversal

- **Co-Search Mode (Primary Partitions):** Machines hosting many relevant vectors run local search.
- **Pull-Push Mode (Secondary Partitions):** Machines hosting few relevant vectors act as "workers."



(c) Co-Search



(a) Pull Data

Near-Linear Scalability

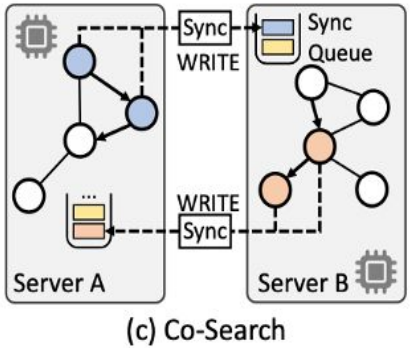
RDMA is expensive!

Zhi et al. “Towards Efficient and Scalable Distributed Vector Search with RDMA”. arXiv 2025
Dang et al. “Pyramid: A General Framework for Distributed Similarity Search on Large-Scale Dataset”. IEEE Big Data 2019.

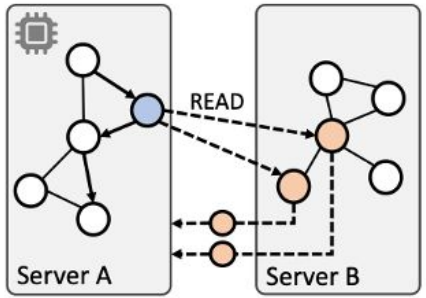
Distributed Vector Search - CoTra [3/3]

Collaborative Traversal

- **Co-Search Mode (Primary Partitions):** Machines hosting many relevant vectors run local search.
- **Pull-Push Mode (Secondary Partitions):** Machines hosting few relevant vectors act as "workers."



(c) Co-Search



(a) Pull Data

Near-Linear Scalability

RDMA is expensive!

Related Systems

Pyramid: It constructs a small meta-HNSW over centroids.

Distributed Vector Search - Graph Partitioning [1/3]

- **Goal:** Fast approximate k-NN graph construction based on recursive clustering. Identify a straightforward way to partition global graph.

Distributed Vector Search - Graph Partitioning [1/3]

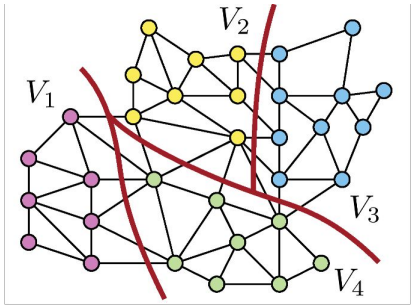
- **Goal:** Fast approximate k-NN graph construction based on recursive clustering.

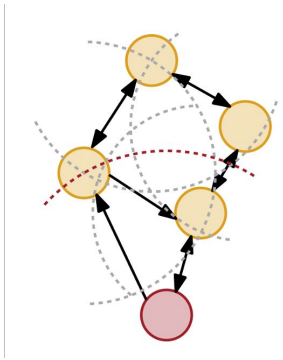
  Identify a straightforward way to partition global graph.
- **Contributions:** **Partitioning** via Dense ball clusters algorithm. **Routing** via Hierarchical K-Means-based Routing (KRT) or Locality Sensitive Hashing-based Routing (HRT).

Distributed Vector Search - Graph Partitioning [1/3]

- **Goal:** Fast approximate k-NN graph construction based on recursive clustering.

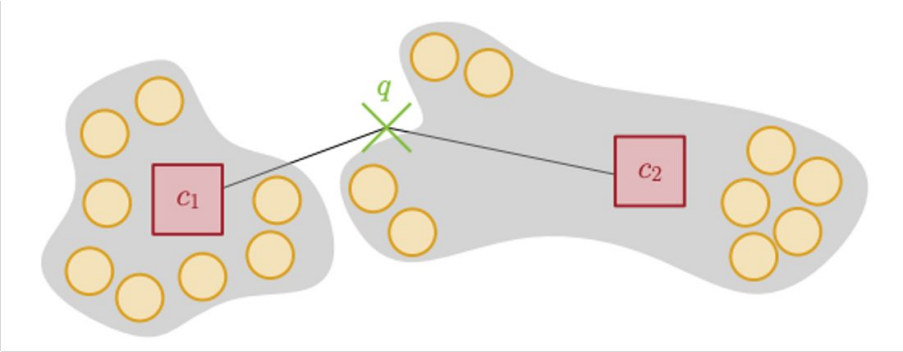
  Identify a straightforward way to partition global graph.
- **Contributions:** **Partitioning** via Dense ball clusters algorithm. **Routing** via Hierarchical K-Means-based Routing (KRT) or Locality Sensitive Hashing-based Routing (HRT).
- Minimize cut edges corresponds to concentrating neighbors in the same shard





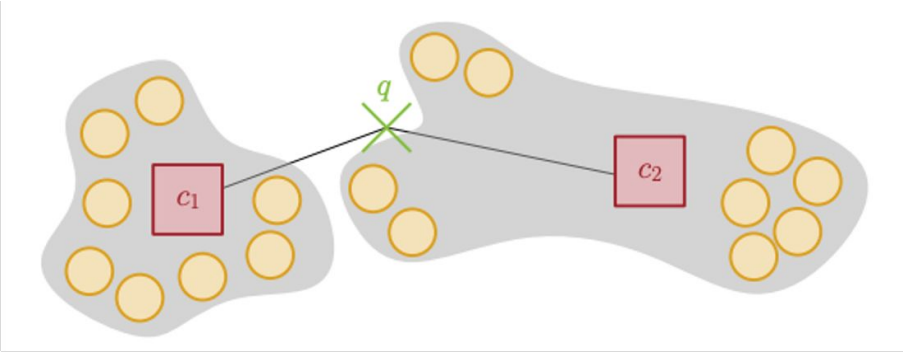
Distributed Vector Search - Graph Partitioning [2/3]

- Each shard is summarized by multiple coarse representatives. A compact routing index is built from all representatives.



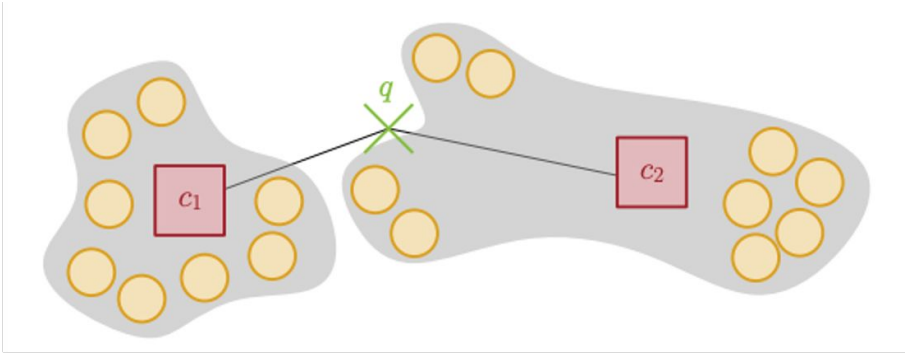
Distributed Vector Search - Graph Partitioning [2/3]

- Each shard is summarized by multiple coarse representatives. A compact routing index is built from all representatives.
- **KRT Routing.** Hierarchical k-means applied within each shard to build the routing index.



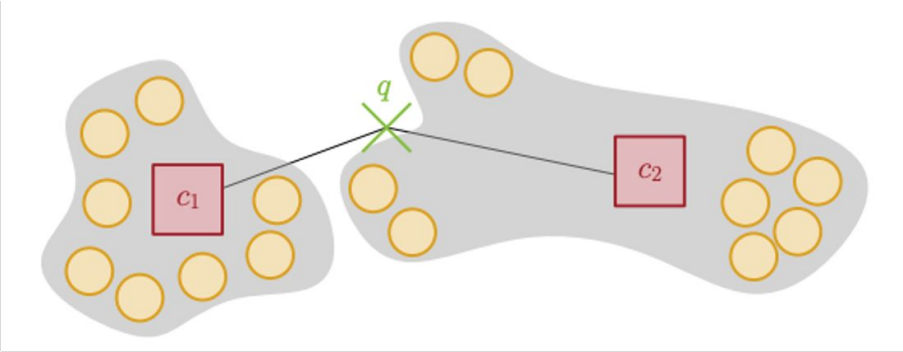
Distributed Vector Search - Graph Partitioning [2/3]

- Each shard is summarized by multiple coarse representatives. A compact routing index is built from all representatives.
- **KRT Routing.** Hierarchical k-means applied within each shard to build the routing index.
- **HRT Routing.** Based on LSH, the routing index is built by uniformly sampling *m* data points.



Distributed Vector Search - Graph Partitioning [2/3]

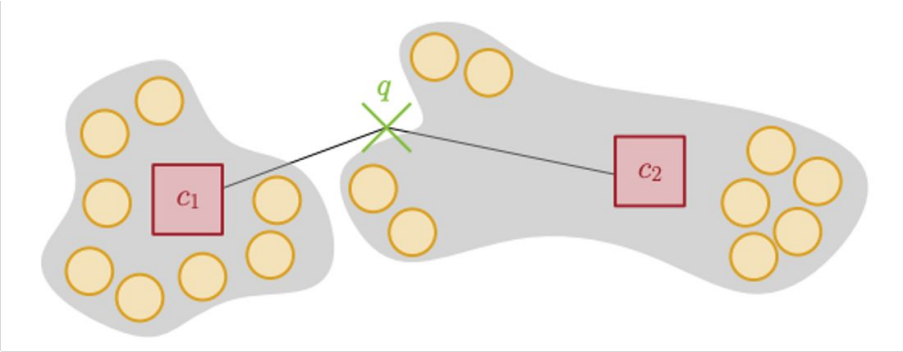
- Each shard is summarized by multiple coarse representatives. A compact routing index is built from all representatives.
- **KRT Routing.** Hierarchical k-means applied within each shard to build the routing index.
- **HRT Routing.** Based on LSH, the routing index is built by uniformly sampling *m* data points.



Global Graph building and fast routing.

Distributed Vector Search - Graph Partitioning [2/3]

- Each shard is summarized by multiple coarse representatives. A compact routing index is built from all representatives.
- **KRT Routing.** Hierarchical k-means applied within each shard to build the routing index.
- **HRT Routing.** Based on LSH, the routing index is built by uniformly sampling *m* data points.



Global Graph building and fast routing.

Partitioning time is 3x slower.

Dang et al.”Passing the Baton: High Throughput Distributed Disk-Based Vector Search with BatANN”. arXiv 2025.
Adams et al. “DistributedANN: Efficient Scaling of a Single DiskANN Graph Across Thousands of Computers”. arXiv 2025.
Xu et al. “Scalable Distributed Vector Search via Accuracy Preserving Index Construction”. arXiv 2025.

Distributed Vector Search - Graph Partitioning [3/3]

Related Systems

- **BatANN**: Uses Graph Partitioning. Its core innovation is an asynchronous state-passing mechanism: it transfers the entire beam-search state to the destination server.

- **DISTRIBUTEDANN**: It scales a single global DISKANN graph across thousands of machines by treating a distributed key-value store as a large shared disk.

- **SPIRE:** Builds a multi-level hierarchical index by recursively clustering centroids until the top level fits within a single machine’s memory. During query answering, the system performs a top-down traversal.

Distributed Vector Search - IVF Systems

Auncel

- Provides **bounded error or bounded latency guarantees**.
- Its core innovation is a **query-aware, whitebox Error–Latency Profile (ELP)**.

Harmony

- Builds a standard running k-means.
- It then distributes the index using a **multi-granularity layout**: clusters are sharded across machines (vector-based partition), and each vector is further split by dimension (dimension-based partition).

Summary of the main Parallel and Distributed Systems

Parallel Method	Index	Hardware	Optimization Target
ELPIS [1]	Tree / k-NN Graph	CPU	Throughput
iQAN [2]	k-NN Graph	CPU	Latency
Tagore [3]	k-NN Graph	GPU	Indexing
Quake [4]	IVF	CPU	Latency
ParlayANN [5]	k-NN Graph	CPU	Throughput
FAISS [6]	IVF	GPU	Throughput

Distributed Method	Index	Storage	Optimization Target
BatANN [7]	k-NN Graph	On-disk	Throughput
Graph Partitioning [14]	k-NN Graph	In-memory	Throughput
SPIRE [8]	k-NN Graph	Both	Throughput
Auncel [9]	IVF	In-memory	Latency
Pyramid [10]	k-NN Graph	In-memory	Throughput
DistributedANN [11]	k-NN Graph	Both	Latency
Harmony [12]	IVF	In-memory	Throughput
CoTra [13]	K-NN Graph	In-memory	Latency

Open problems and future directions

- Indexing building: partitioning which preserves locality
- Query processing: avoid making all machines to be involved
- Hybrid Search: Combined vector similarity and scalar/attribute filtering in distributed environments.
- Still no support for streaming settings in distributed vector search.

Streaming Vector Search

Many thanks to Lu et al. (VectraFlow), Singh et al. (FreshDiskANN), Xu et al. (IP-DiskANN), Zhang et al. (CleANN), Gong et al. (VStream), who provided supporting material for this part of the tutorial

Motivation

- Vectors are no longer static
- Frequent and massive amounts of:
  - Insertions
  - Deletions
  - Updates
- Static ANN indexes assume offline construction

Motivation

- Vectors are no longer static
- Frequent and massive amounts of:
  - Insertions
  - Deletions
  - Updates
- Static ANN indexes assume offline construction

Static ANN ≠ Dynamic Workloads

Outline

- Operations & Update models
- Approaches:
  - **Batched** Approaches
  - **K-NN Graph** Approaches
  - **IVF** Approaches
  - Other Systems
- Open Problems

Operations

Operation	Challenges
Insert	**Graph connectivity/centroid staleness**
Delete	**“Broken” edges / imbalance of distribution**
Update	**Data drift**

Operations

Operation	Challenges
Insert	**Graph connectivity/centroid staleness**
Delete	**“Broken” edges / imbalance of distribution**
Update	**Data drift**

Maintaining index quality under updates is hard

Four update models [1/3]

Granularity: Batched **vs** Incremental

Materialization: In-place **vs** Out of place

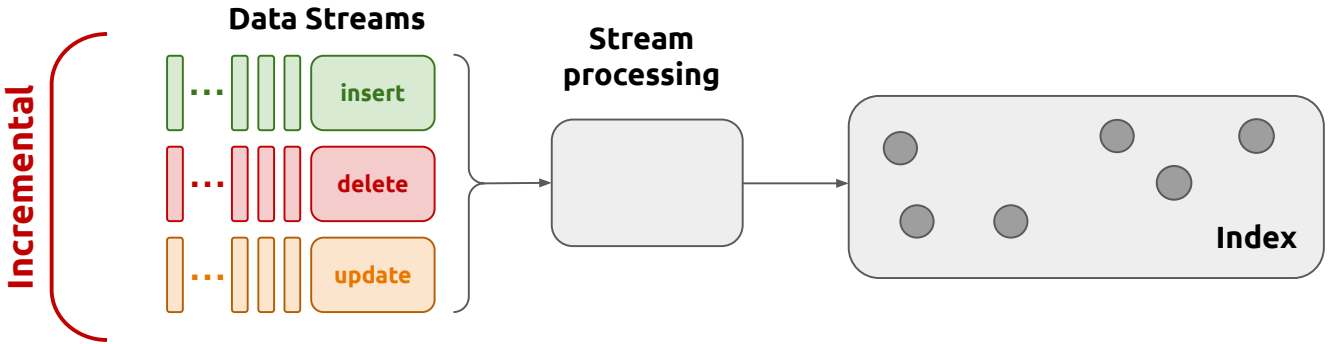
Four update models [1/3]

Granularity: Batched **vs** Incremental

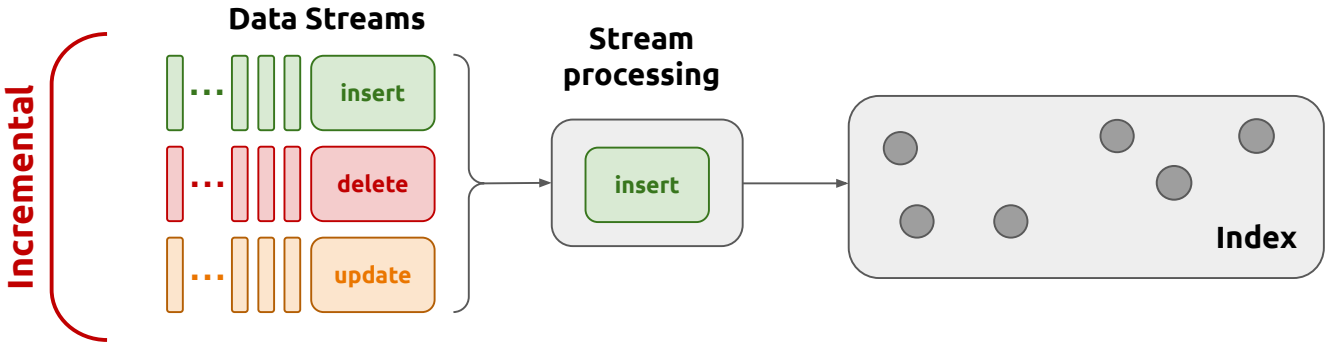
Materialization: In-place **vs** Out of place

Whether updates happen **one-by-one or in batches**, and whether index changes happen in the **index itself** or in a **secondary data structure**

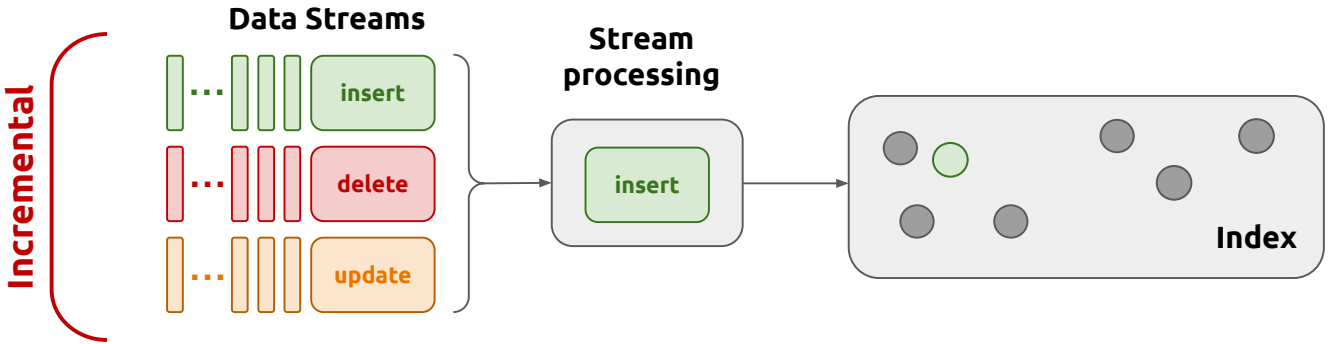
Four update models [1/3] - **Granularity**



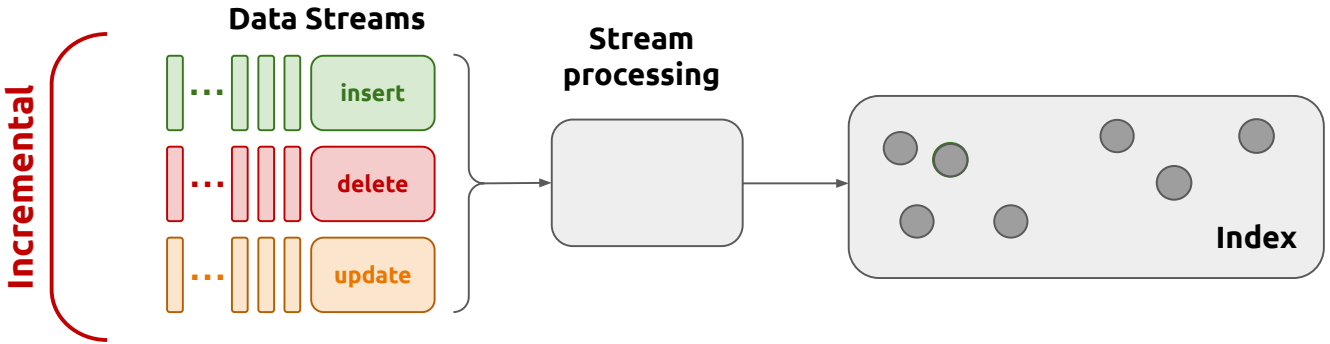
Four update models [1/3] - **Granularity**



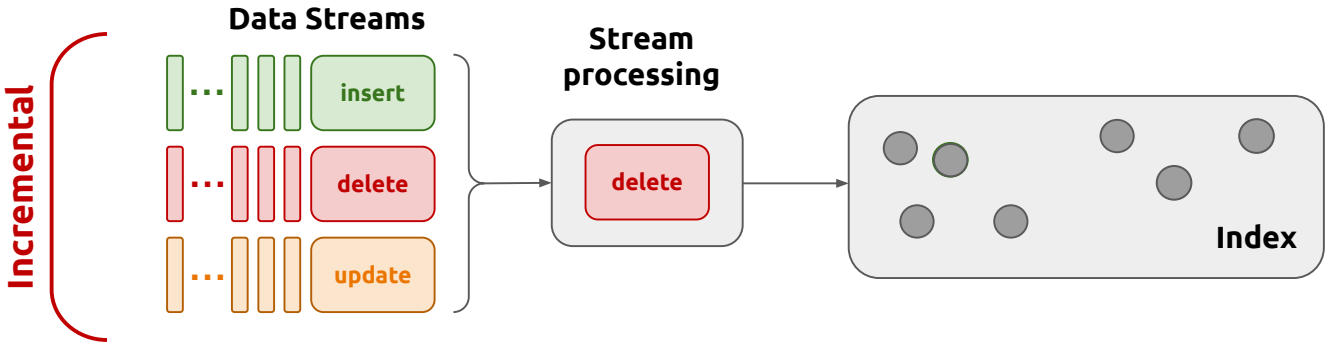
Four update models [1/3] - **Granularity**



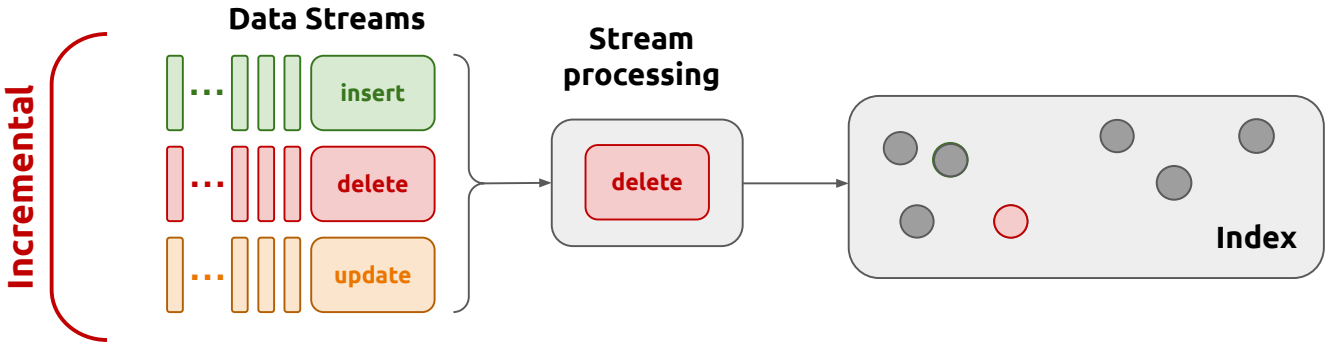
Four update models [1/3] - **Granularity**



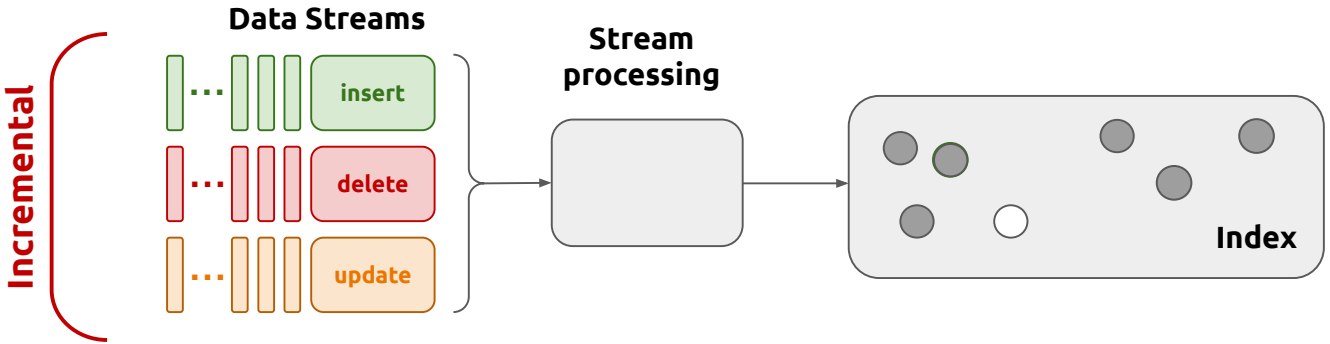
Four update models [1/3] - **Granularity**



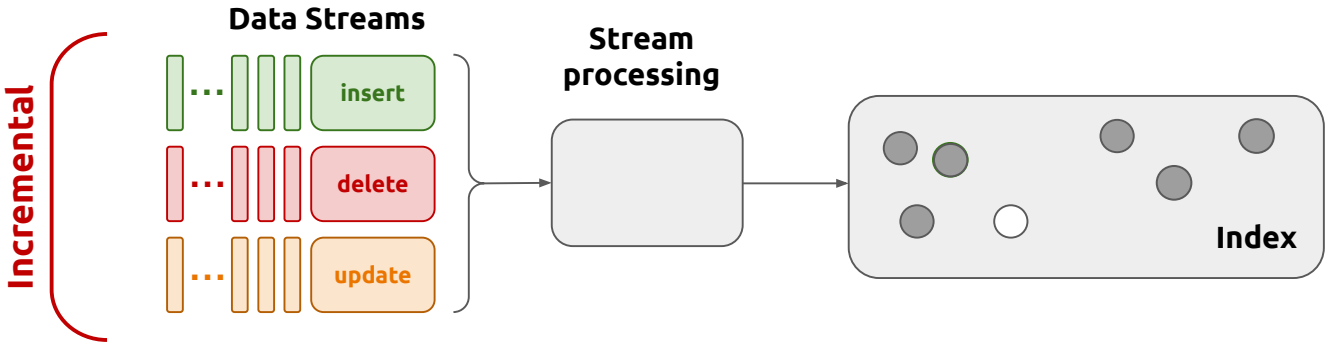
Four update models [1/3] - **Granularity**



Four update models [1/3] - **Granularity**

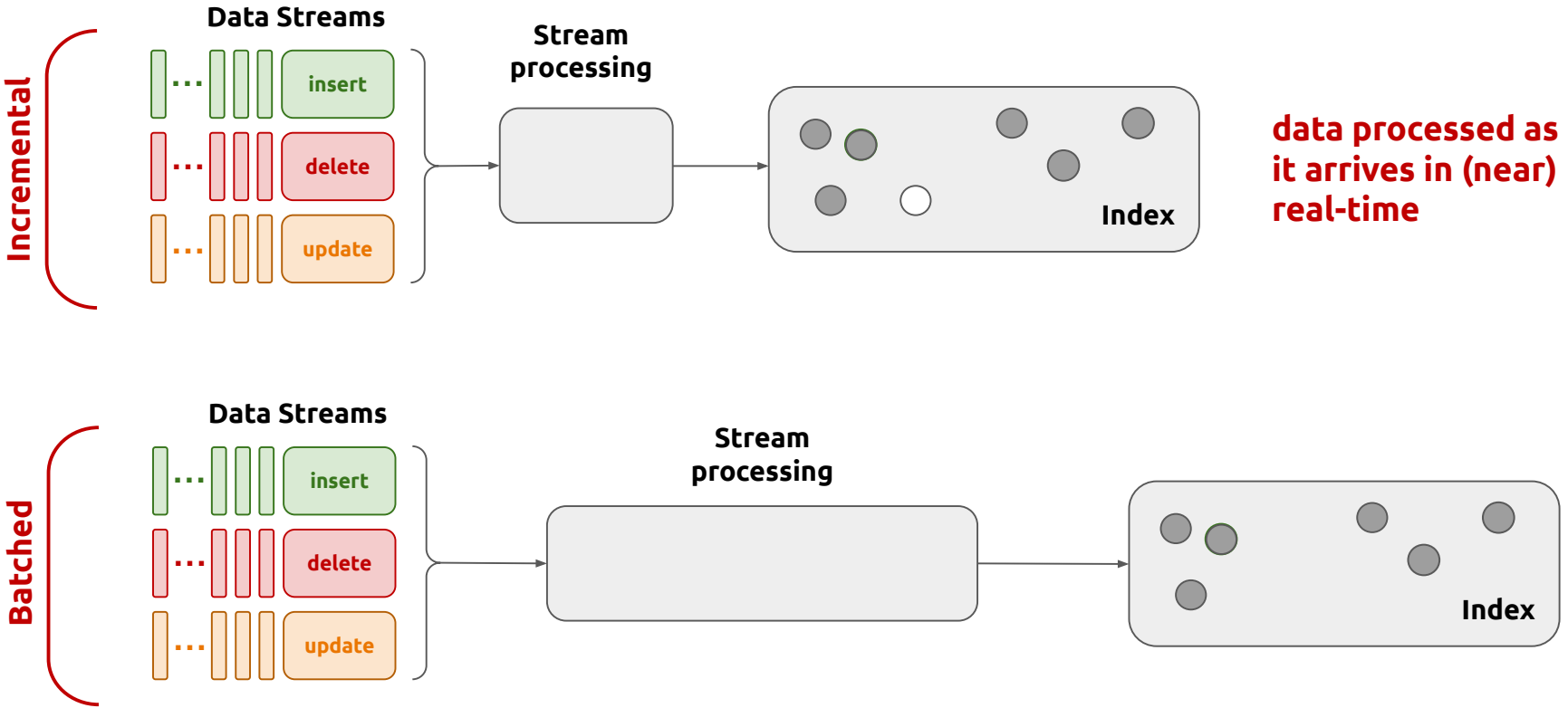


Four update models [1/3] - **Granularity**

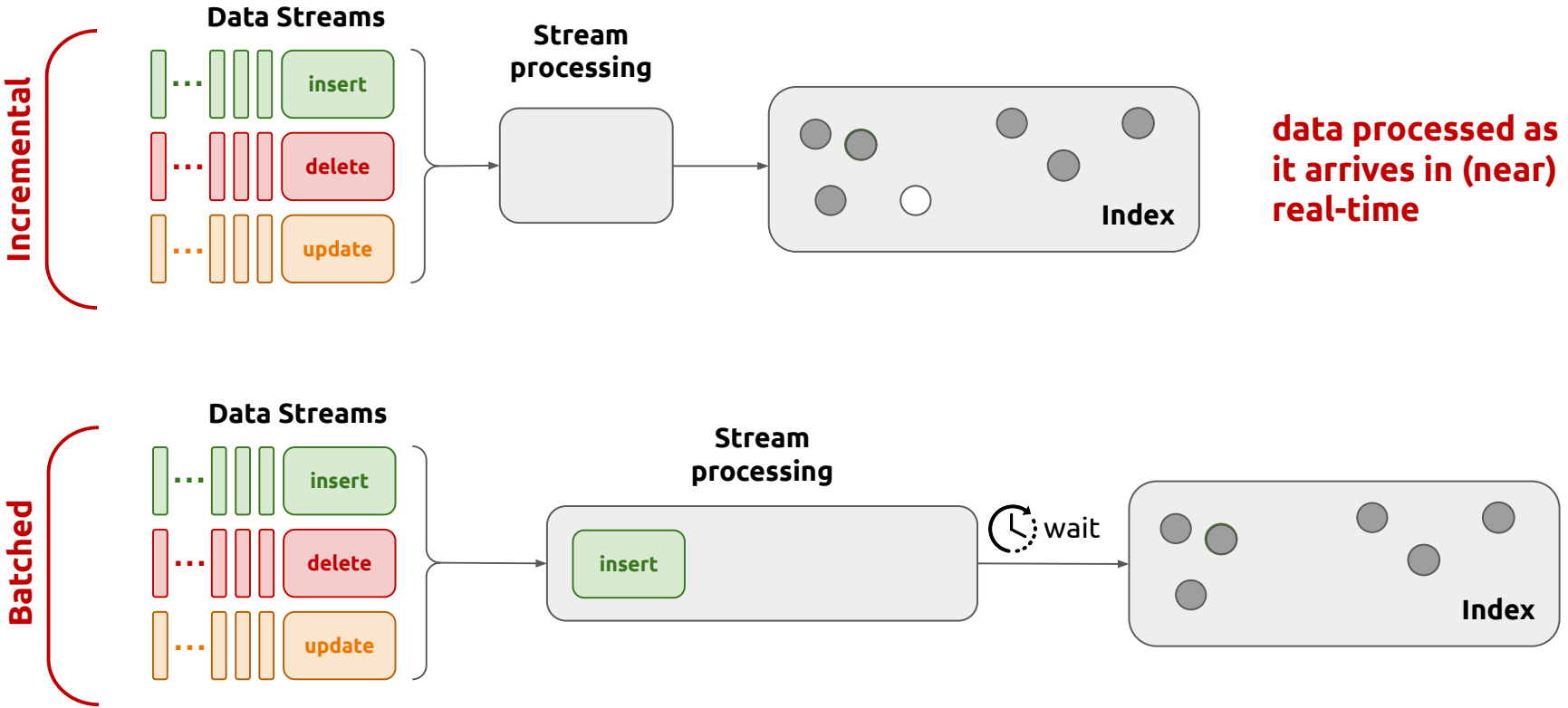


data processed as it arrives in (near) real-time

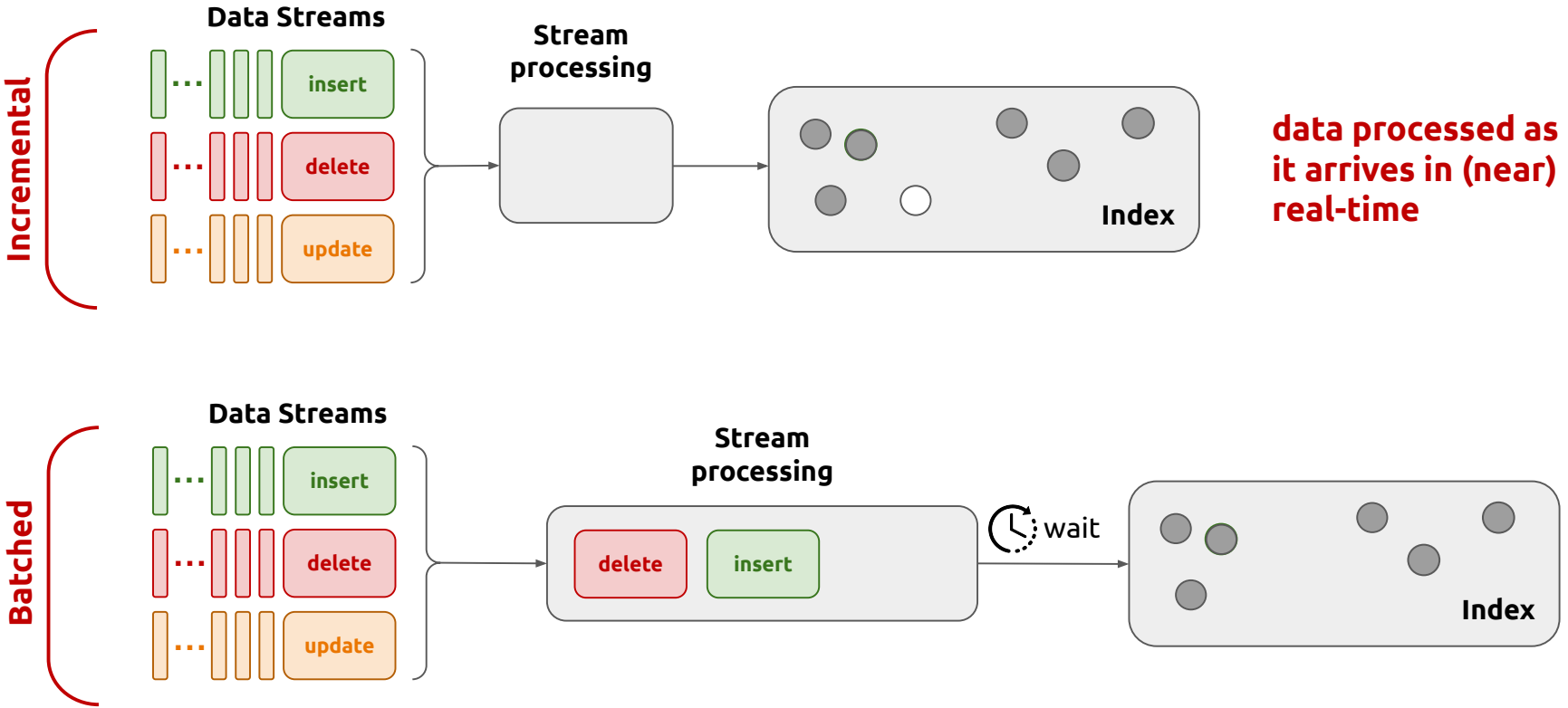
Four update models [1/3] - **Granularity**



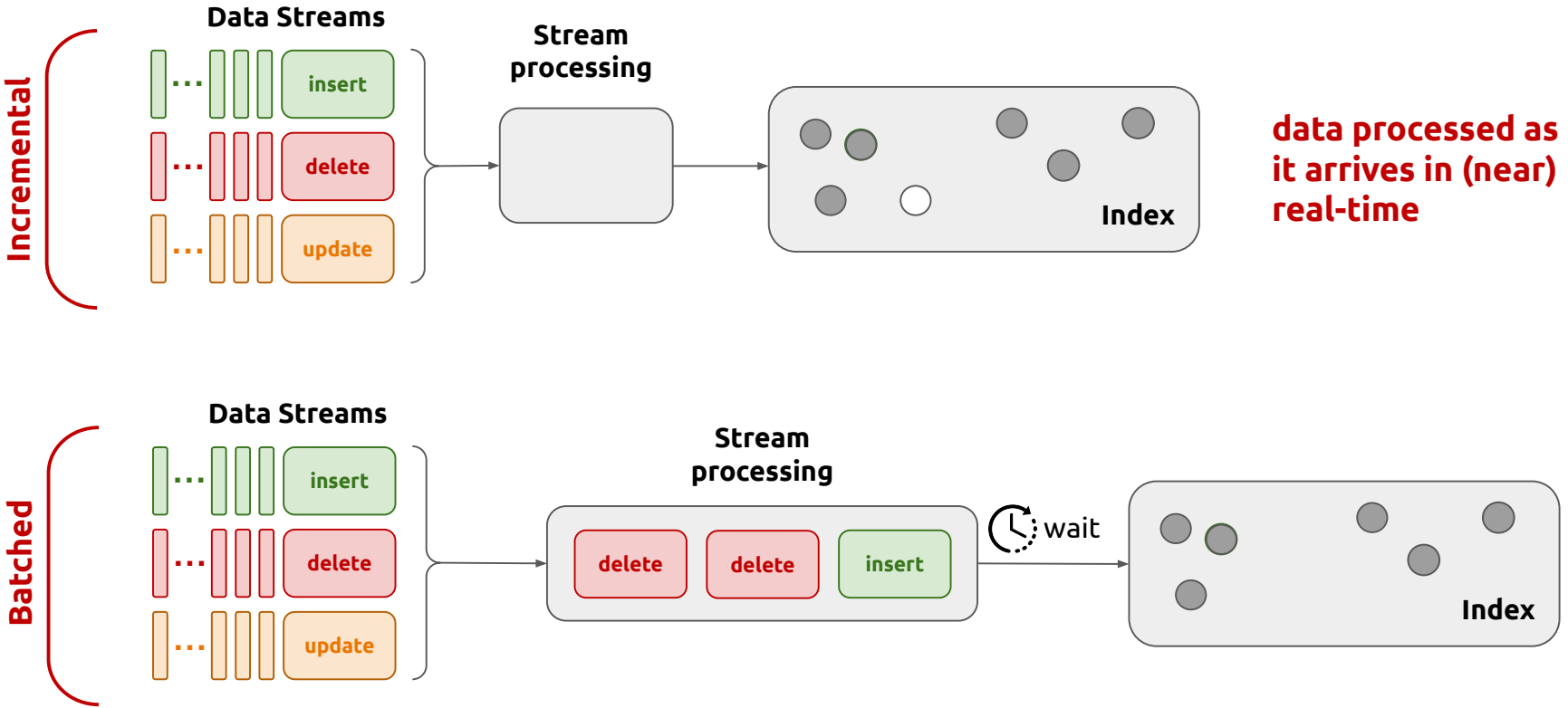
Four update models [1/3] - **Granularity**



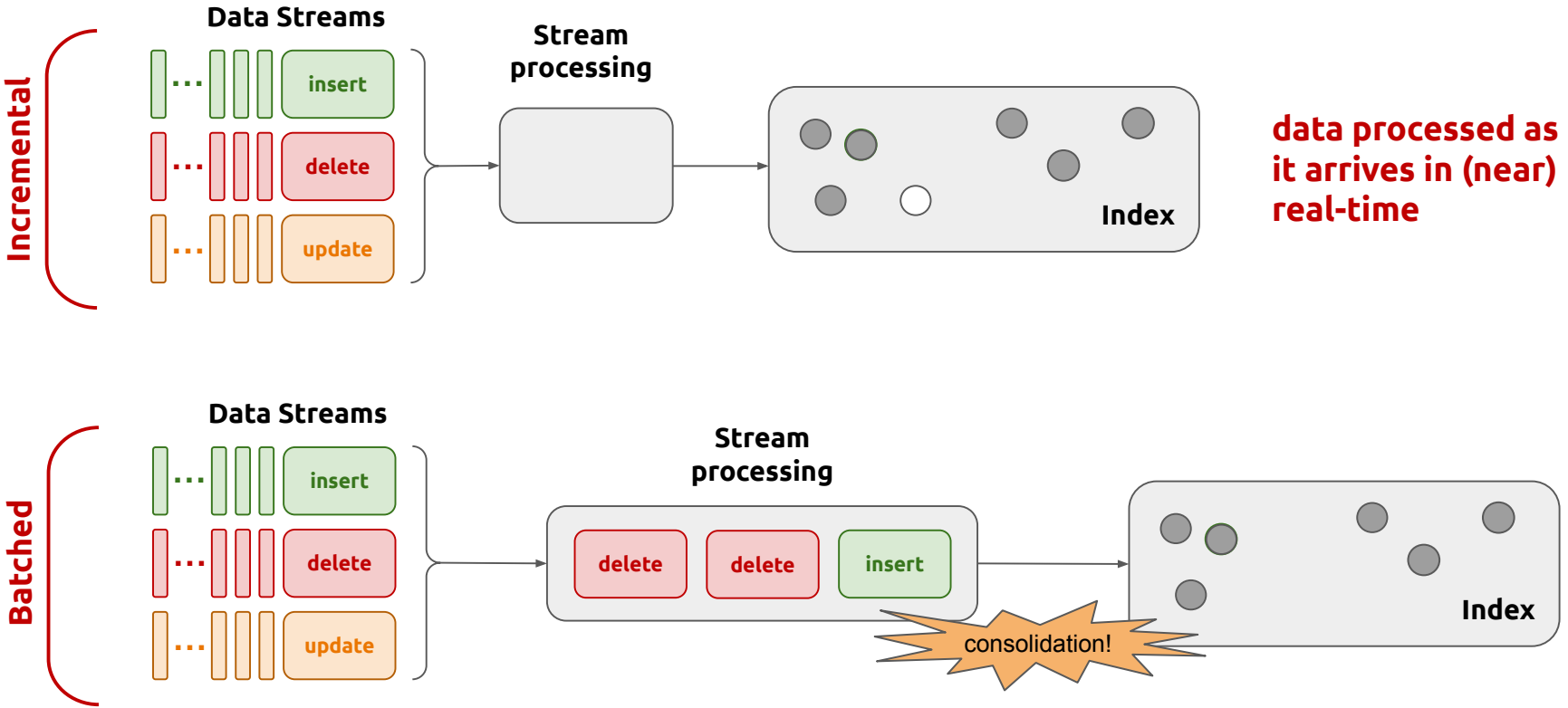
Four update models [1/3] - **Granularity**



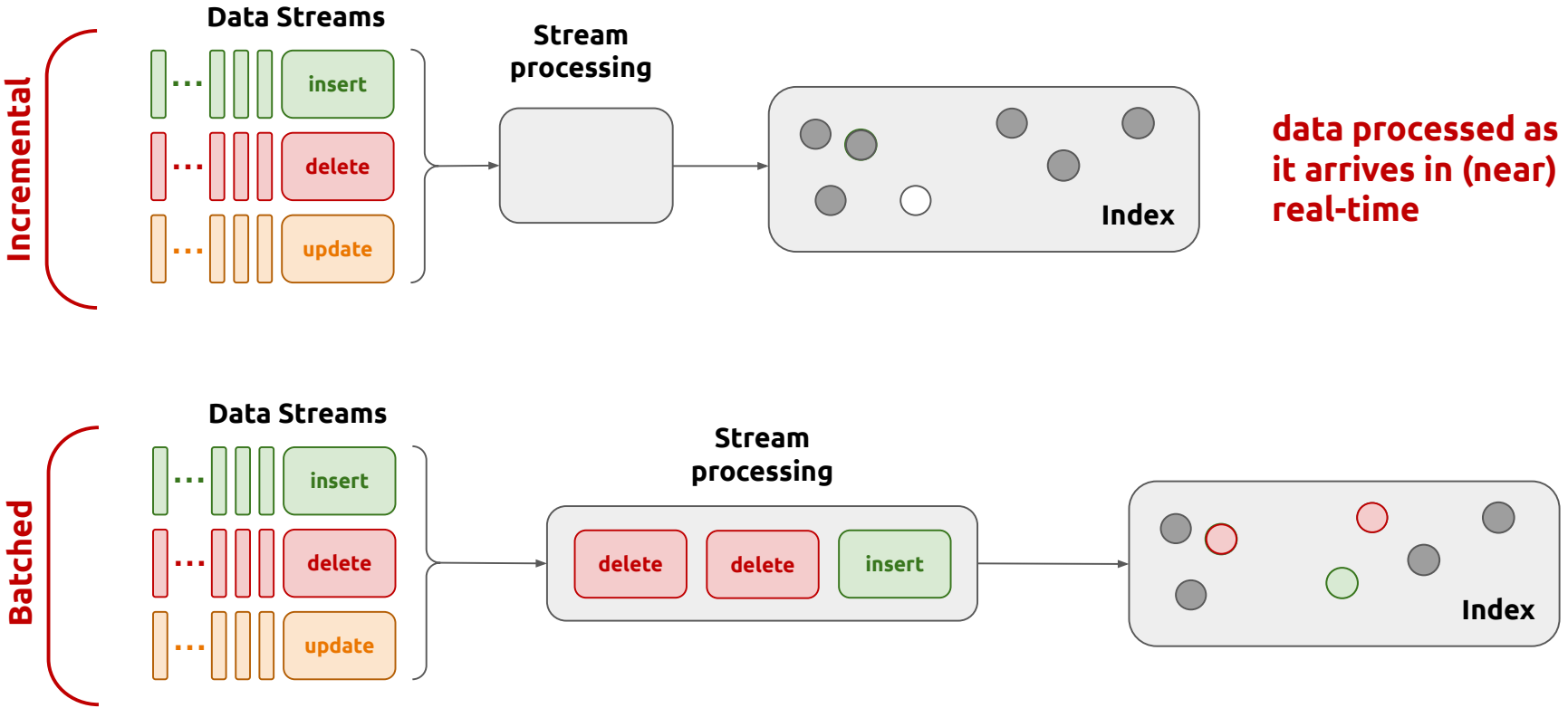
Four update models [1/3] - **Granularity**



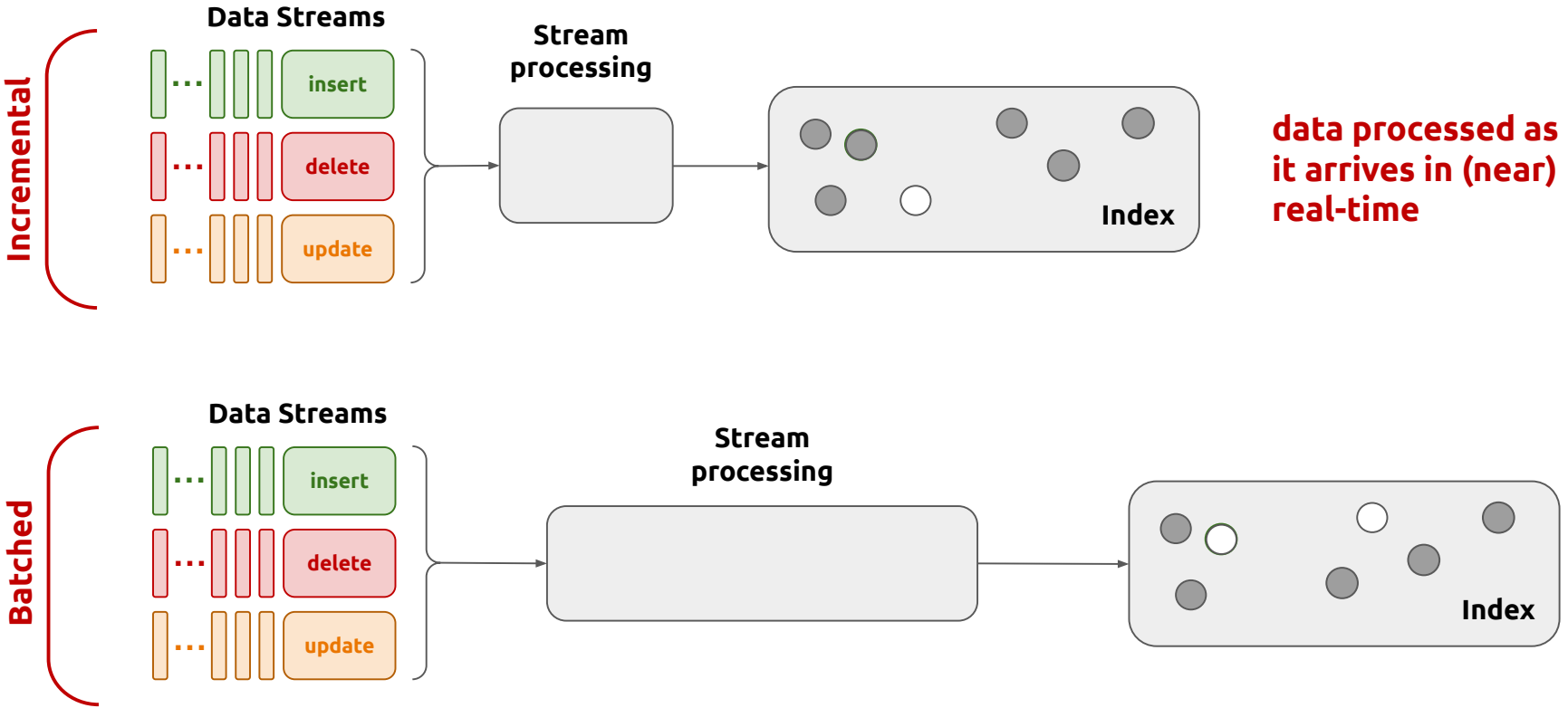
Four update models [1/3] - **Granularity**



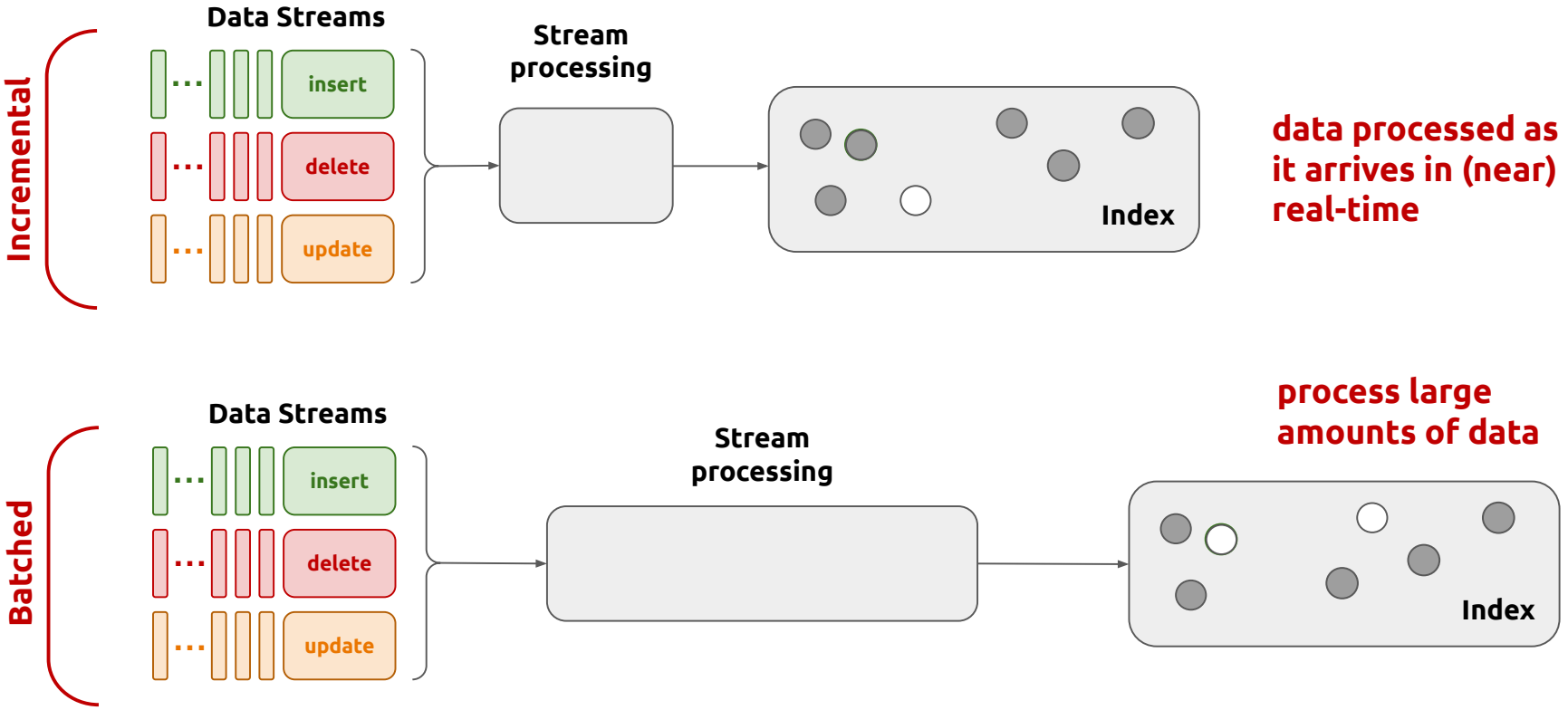
Four update models [1/3] - **Granularity**



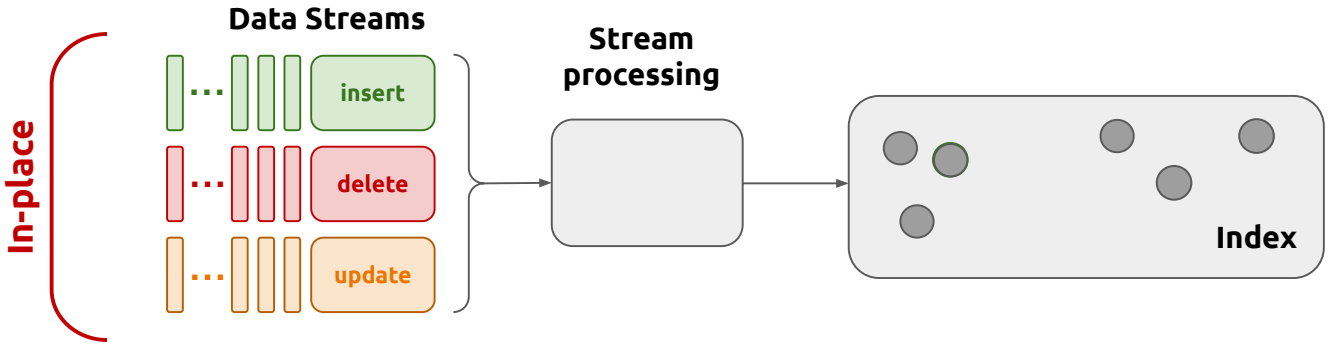
Four update models [1/3] - **Granularity**



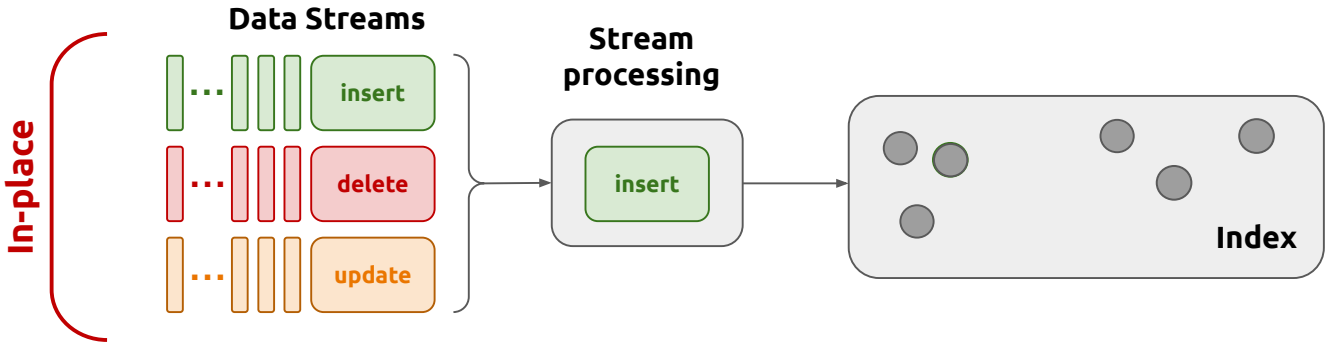
Four update models [1/3] - **Granularity**



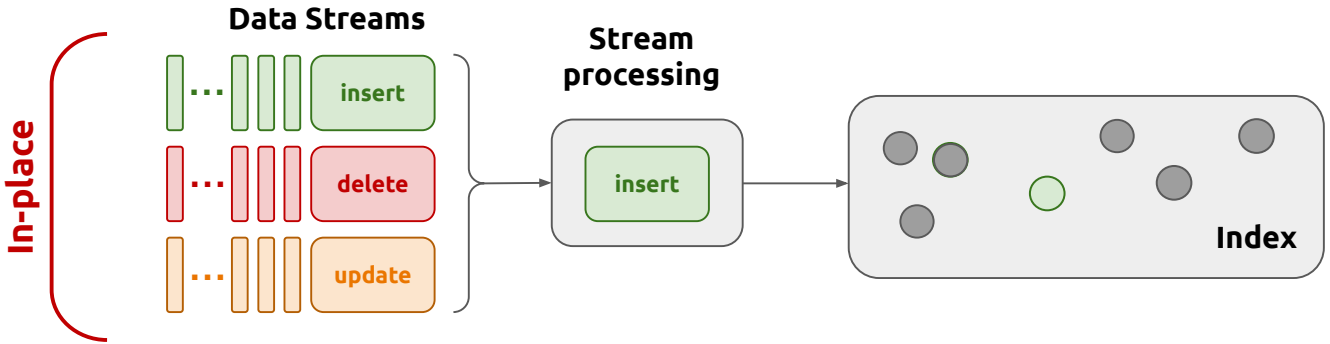
Four update models [2/3] - **Materialization**



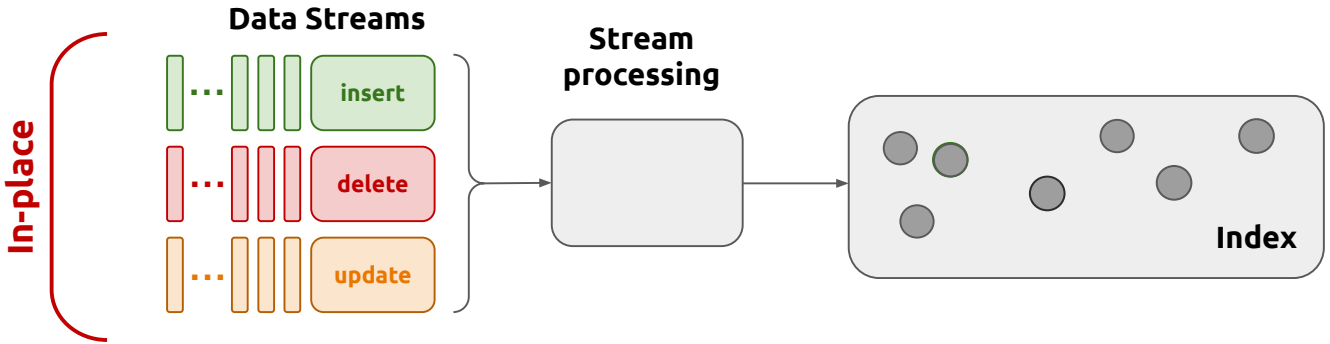
Four update models [2/3] - **Materialization**



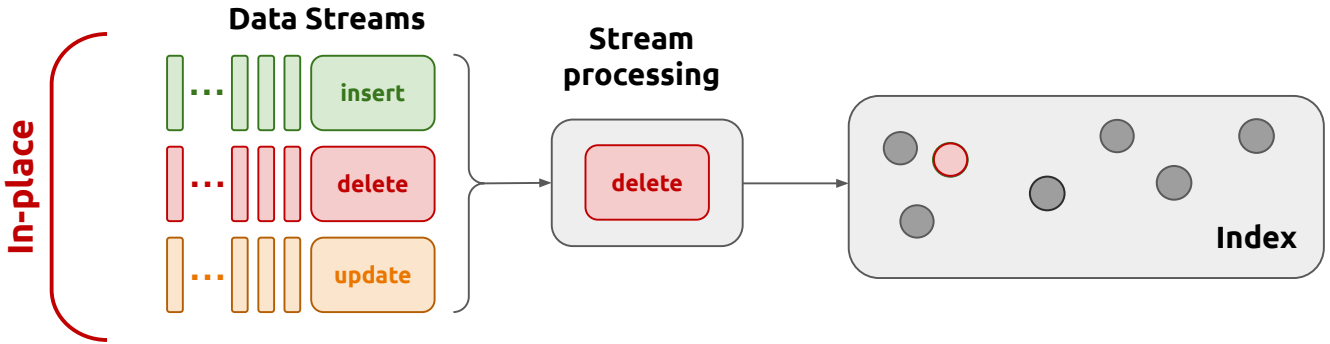
Four update models [2/3] - **Materialization**



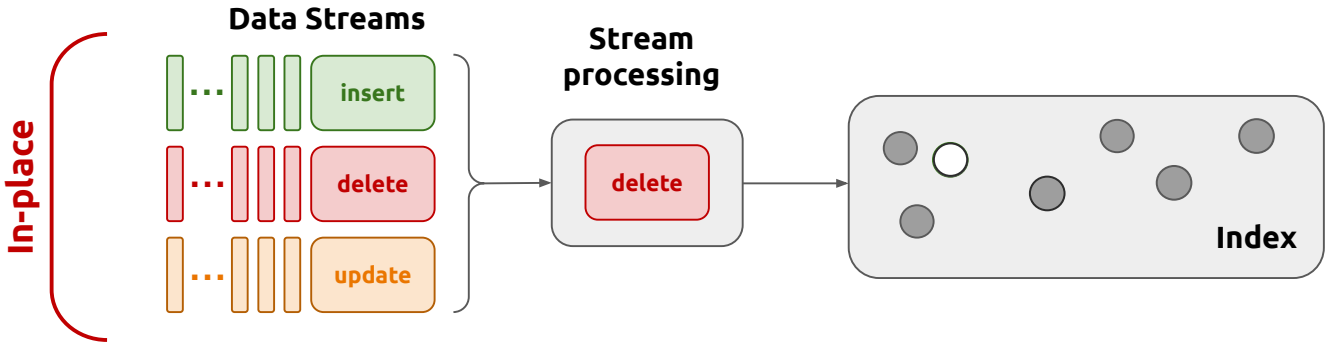
Four update models [2/3] - **Materialization**



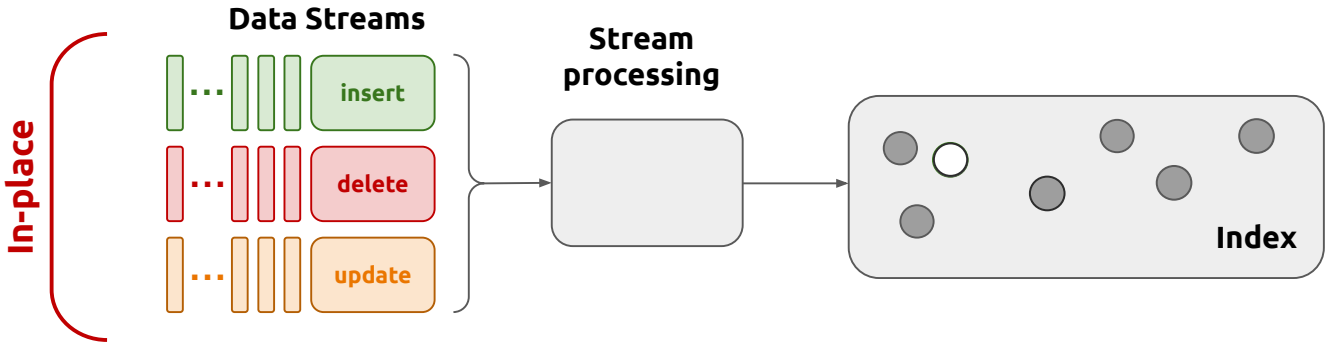
Four update models [2/3] - **Materialization**



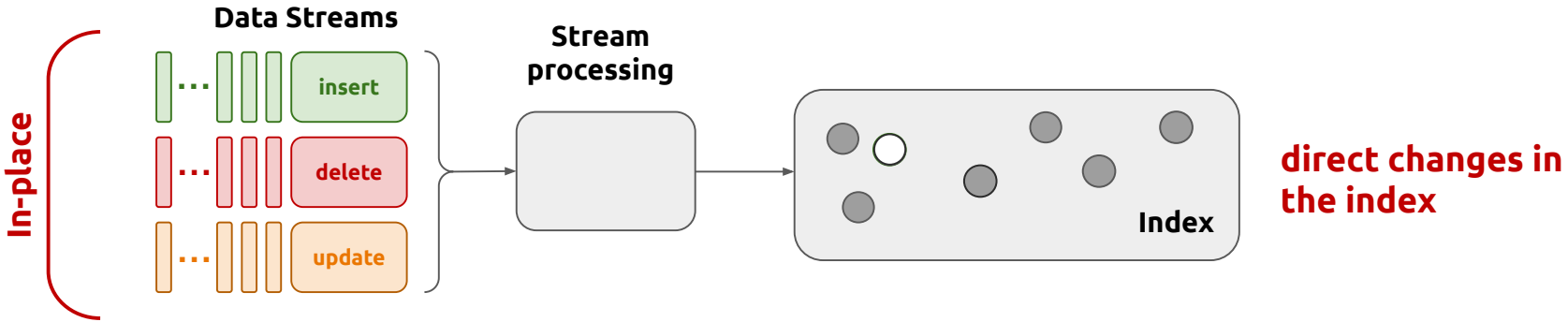
Four update models [2/3] - **Materialization**



Four update models [2/3] - **Materialization**

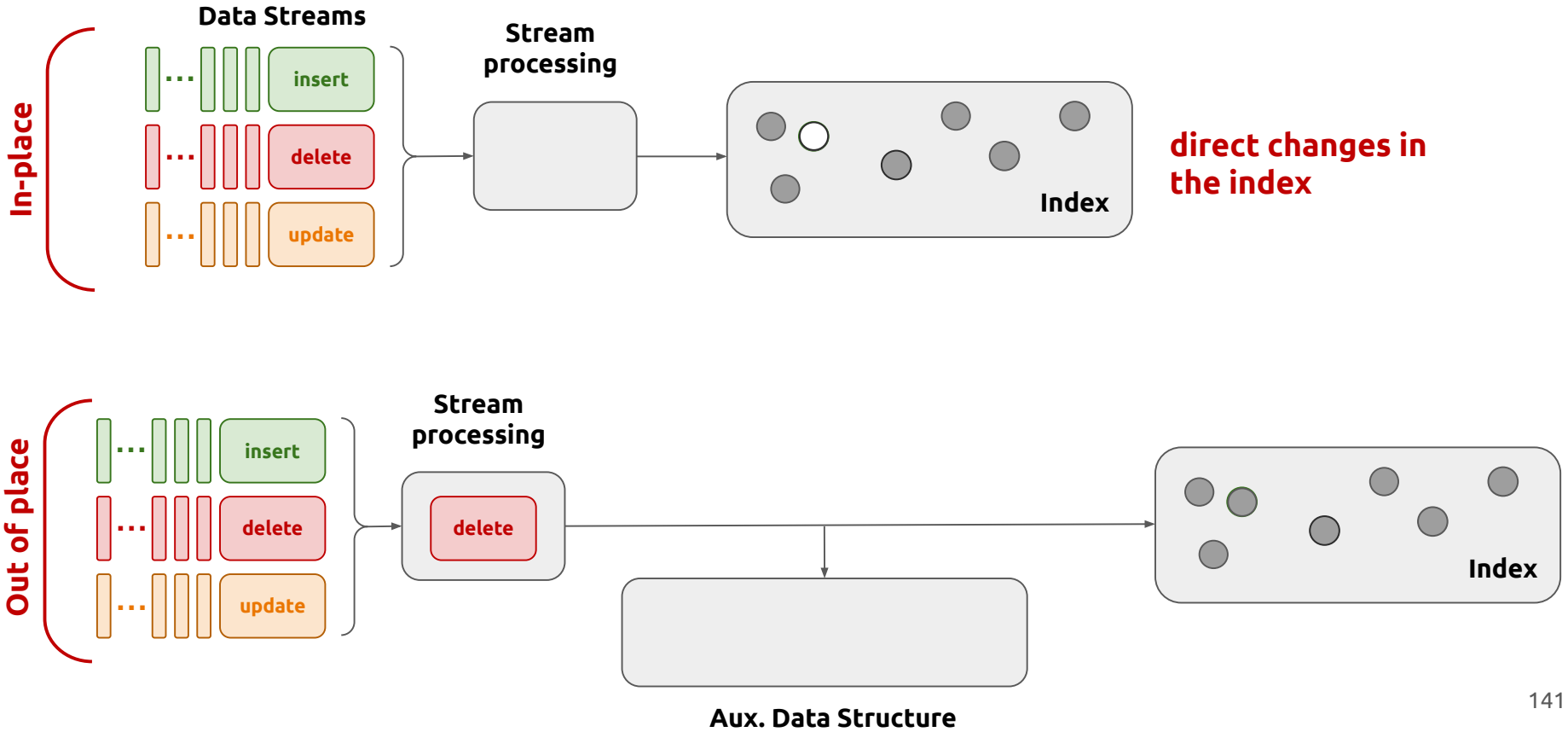


Four update models [2/3] - **Materialization**

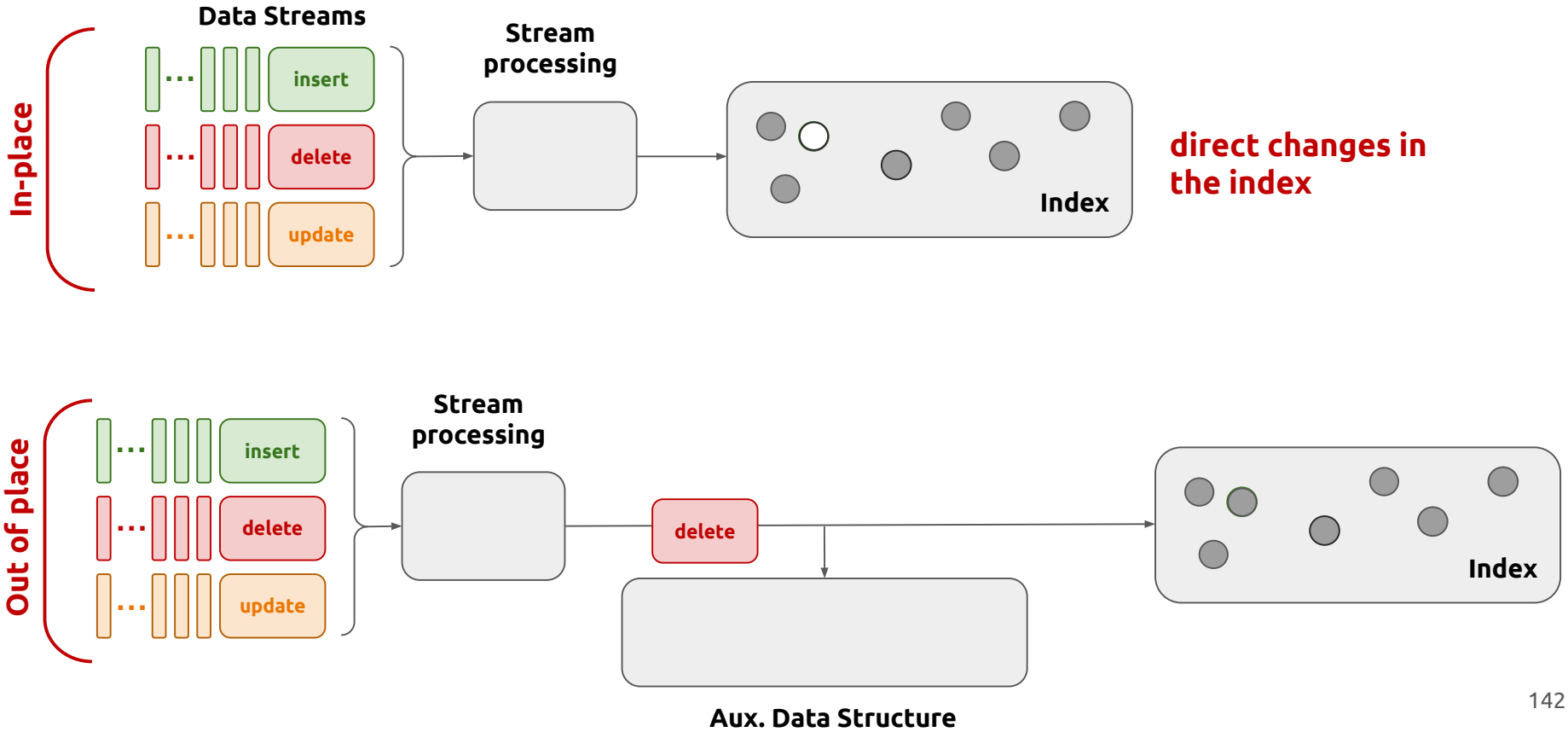


direct changes in the index

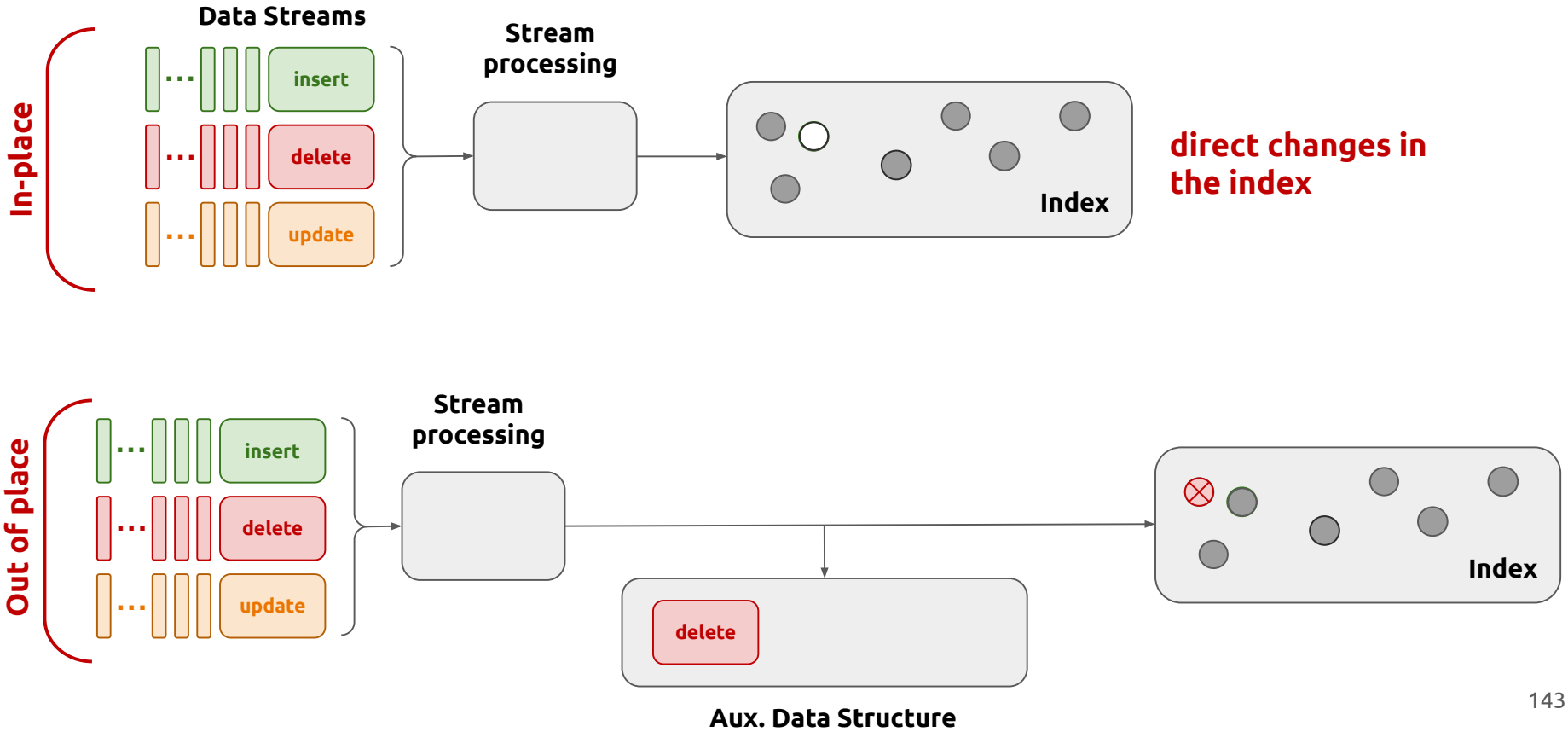
Four update models [2/3] - **Materialization**



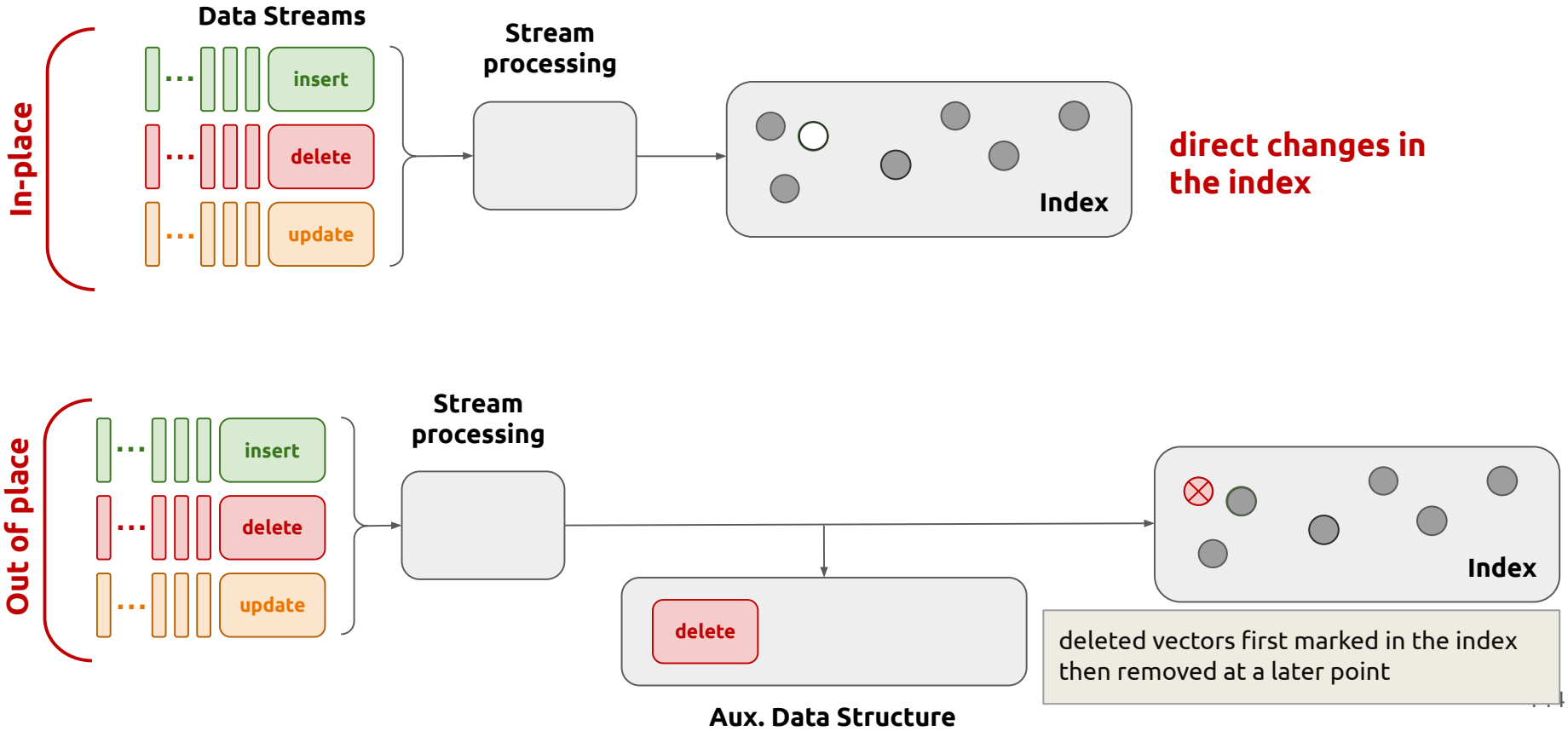
Four update models [2/3] - **Materialization**



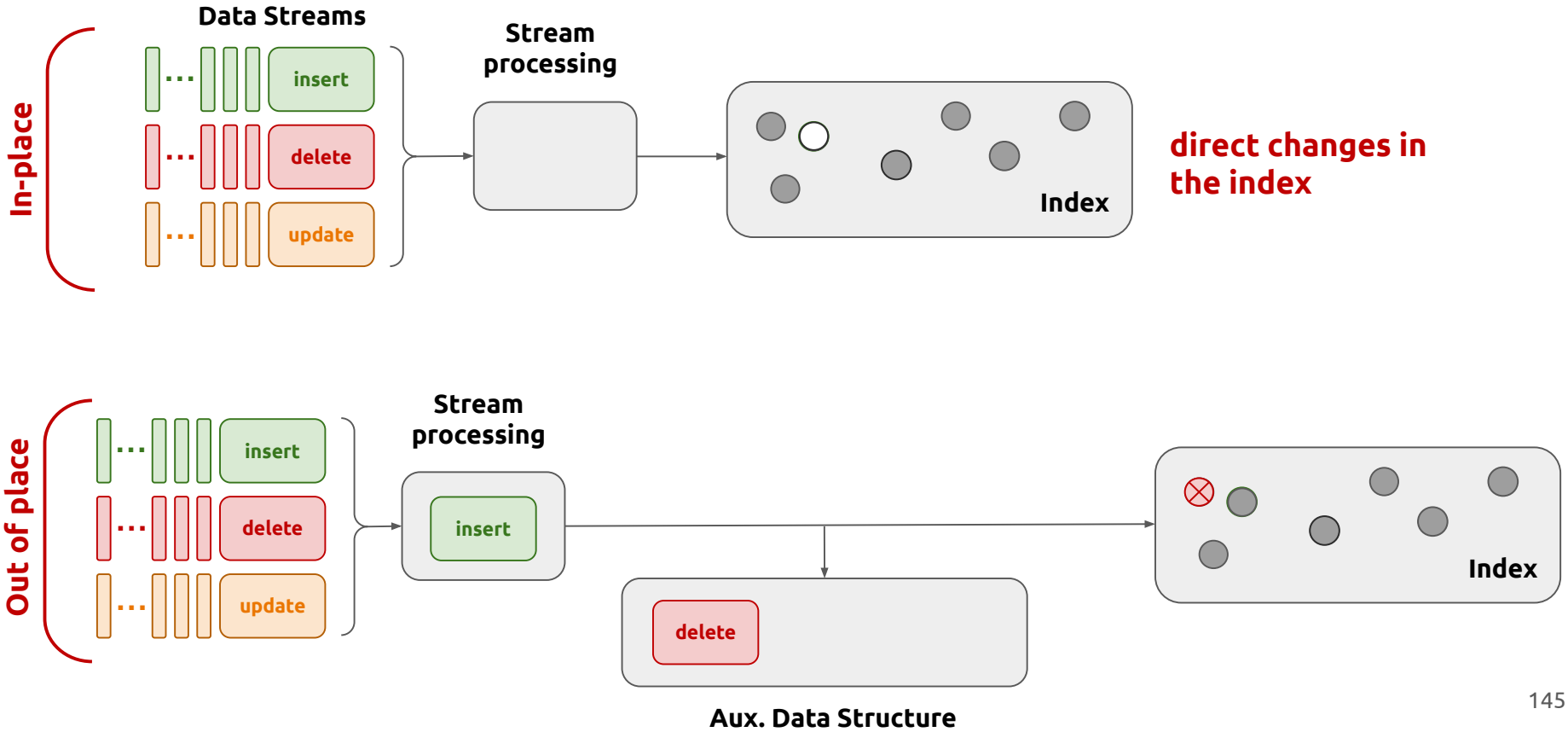
Four update models [2/3] - **Materialization**



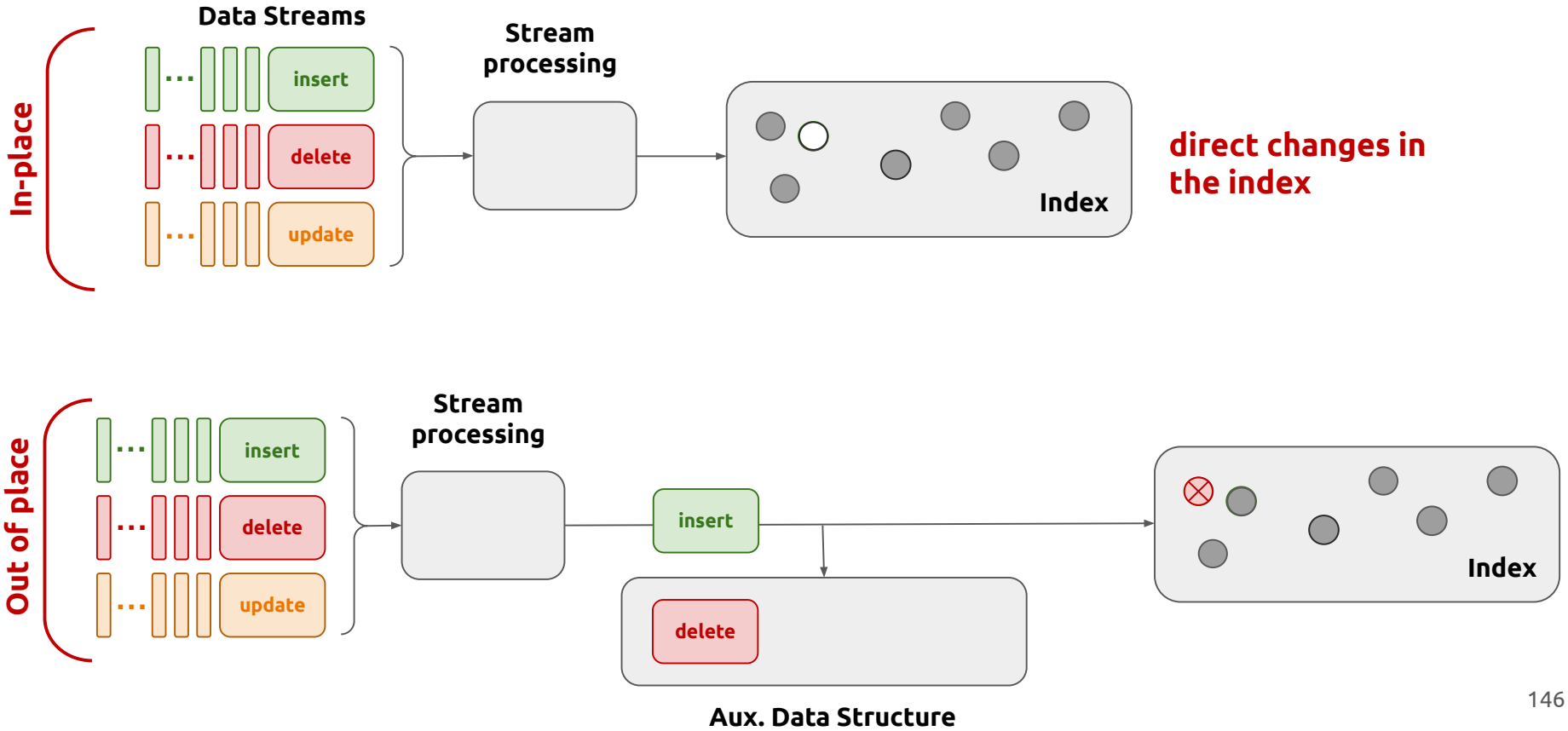
Four update models [2/3] - **Materialization**



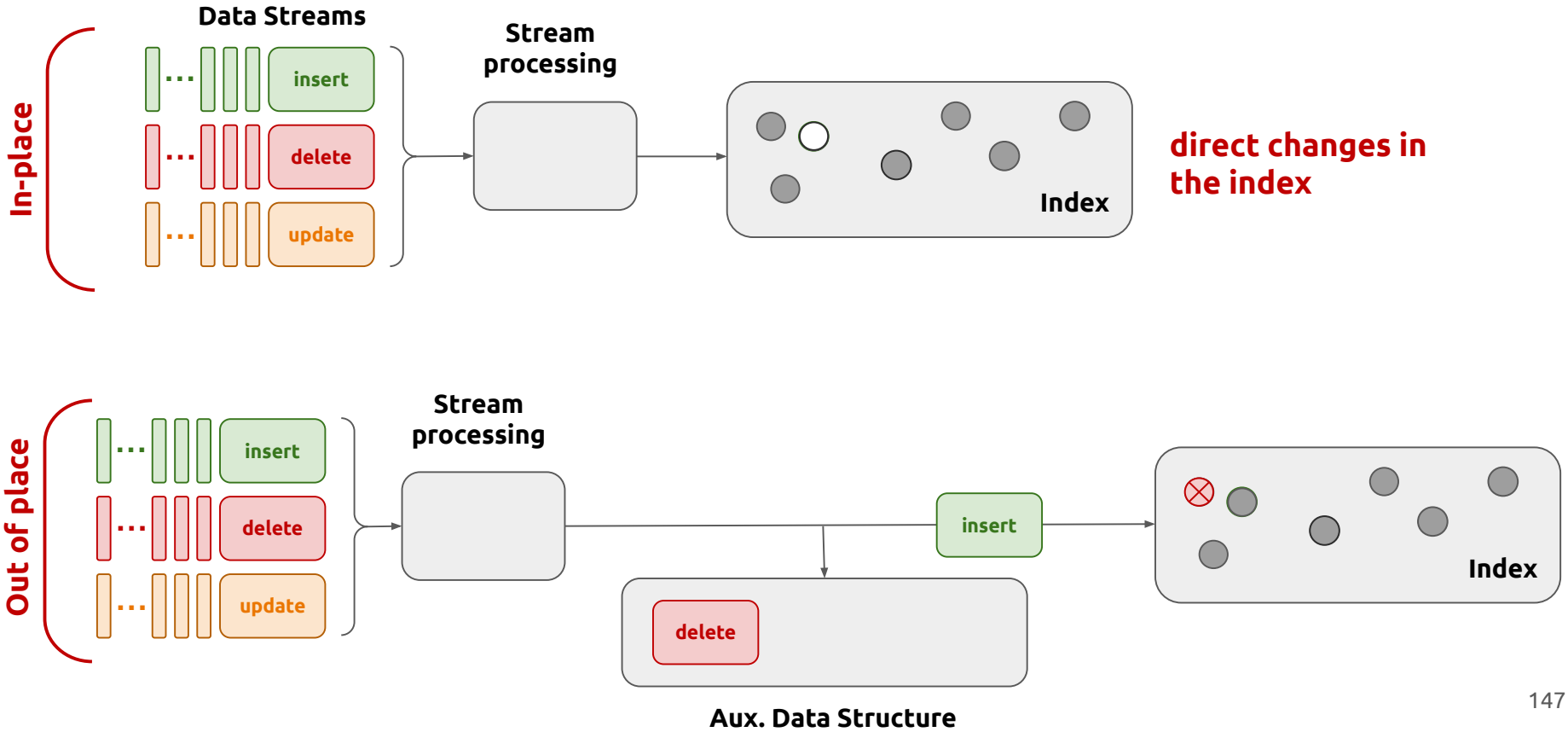
Four update models [2/3] - **Materialization**



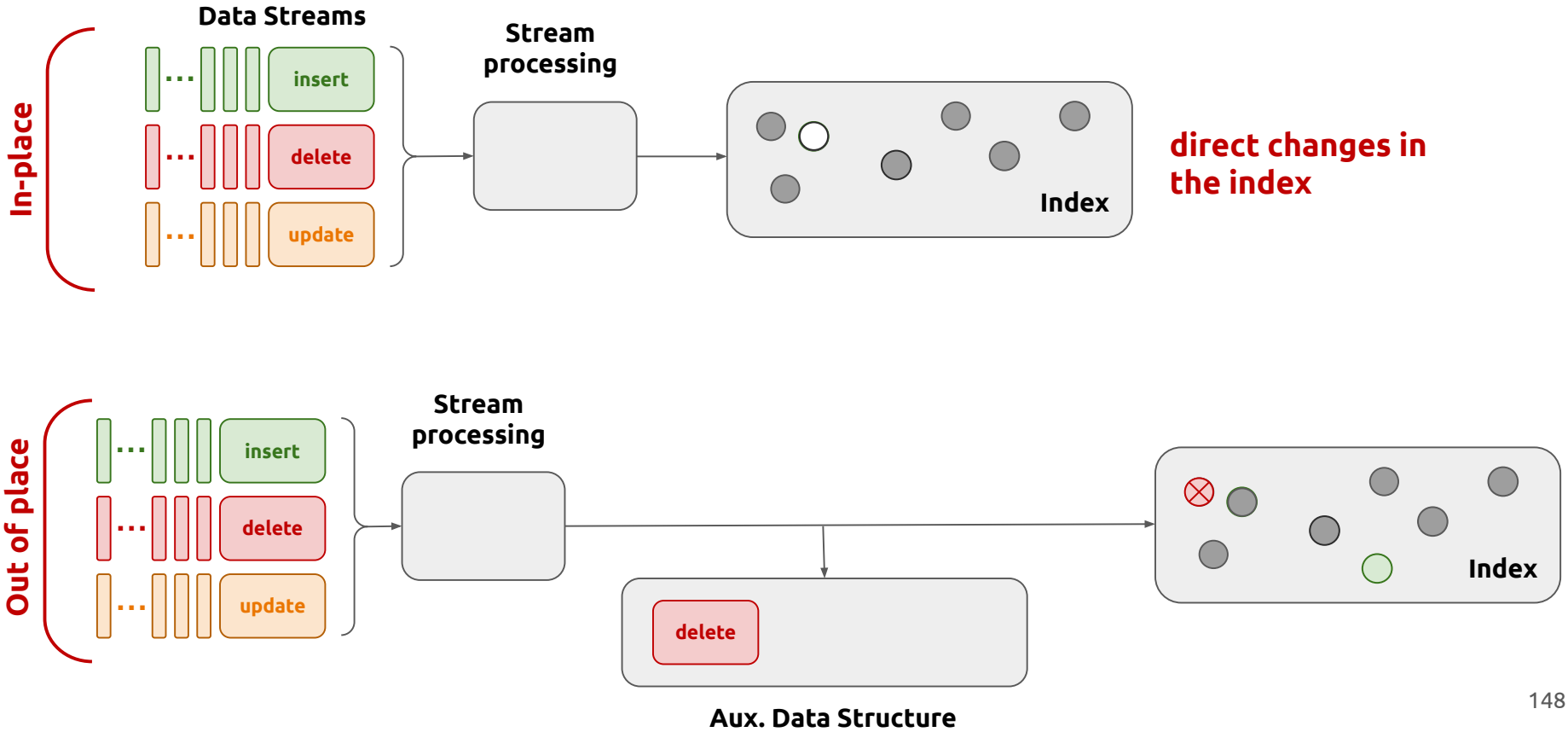
Four update models [2/3] - **Materialization**



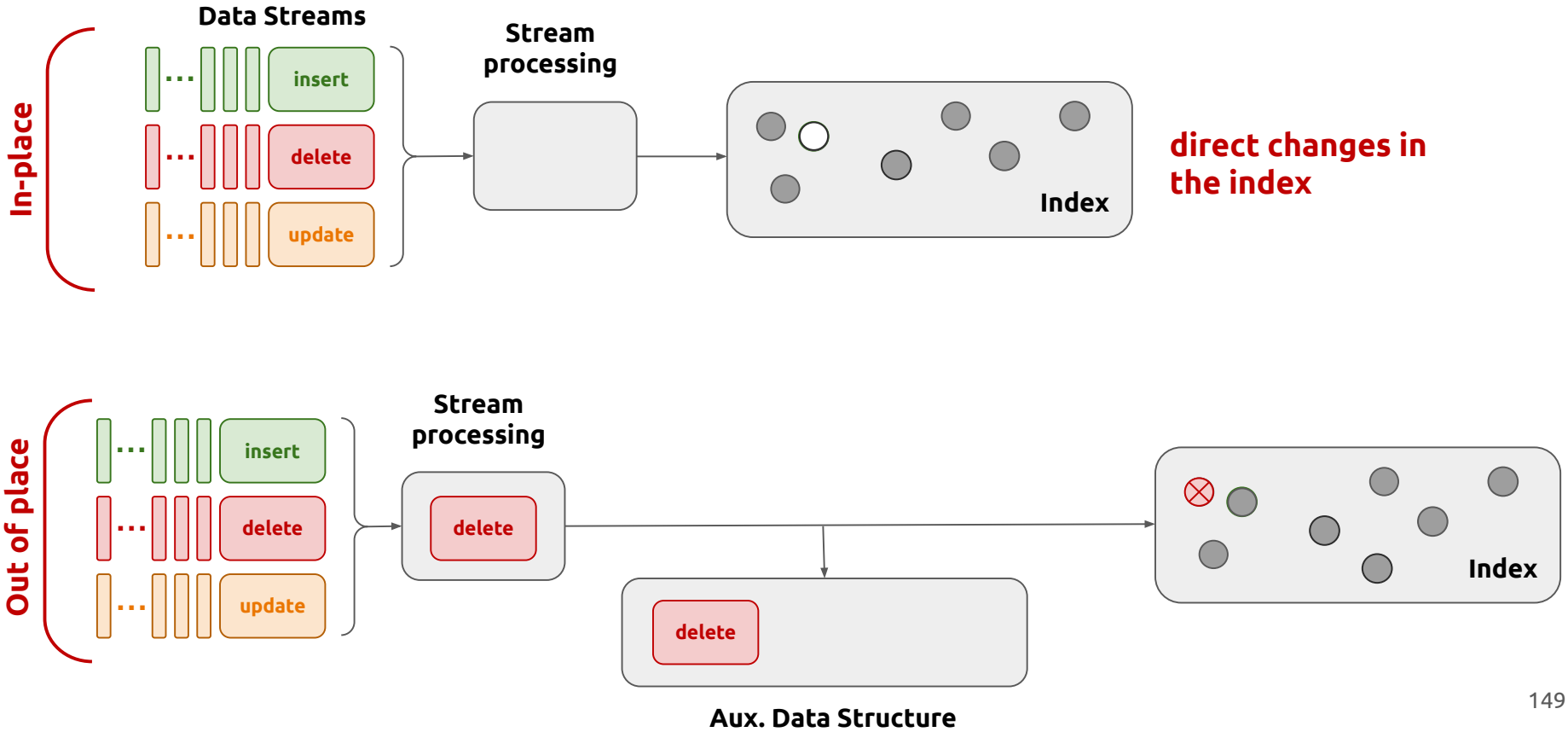
Four update models [2/3] - **Materialization**



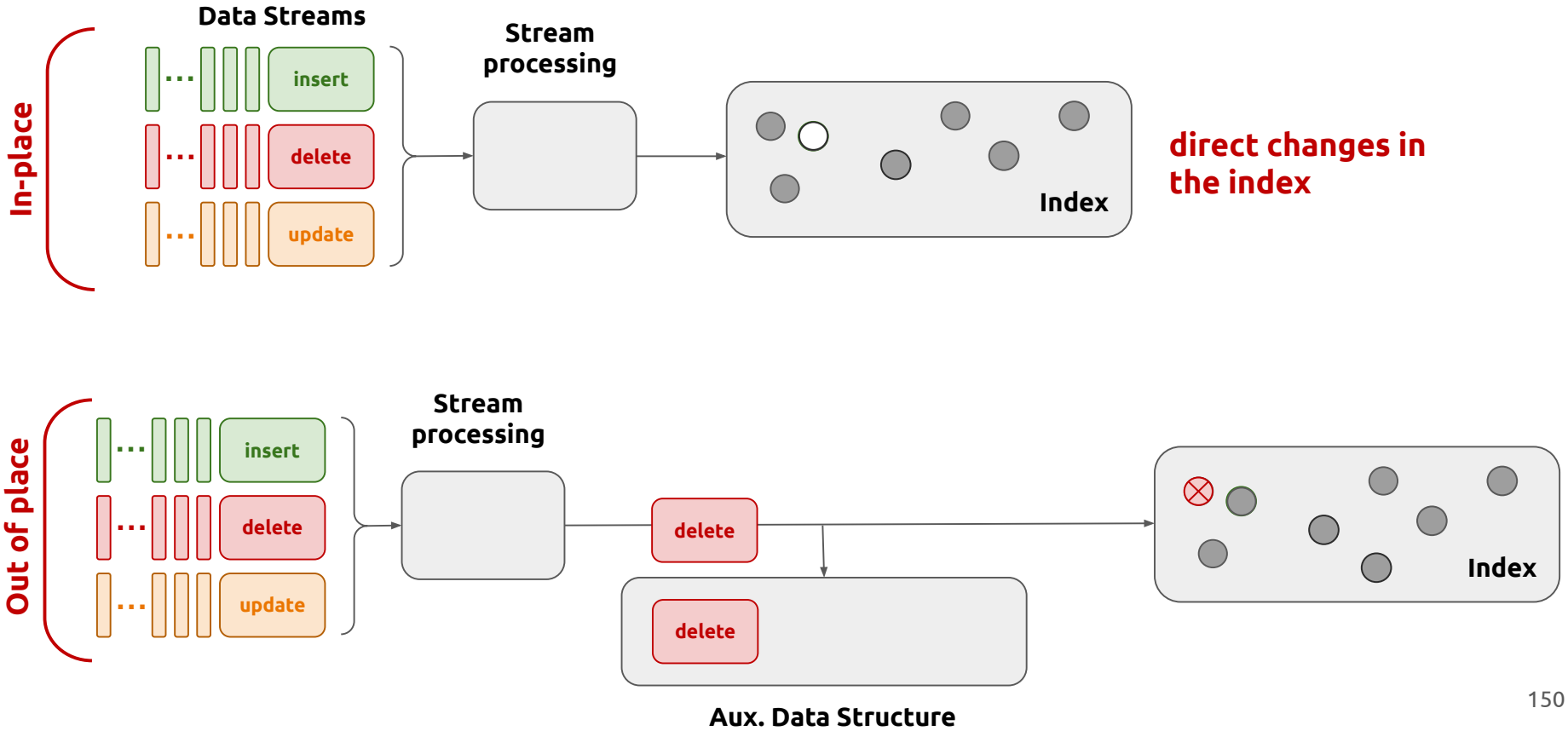
Four update models [2/3] - **Materialization**



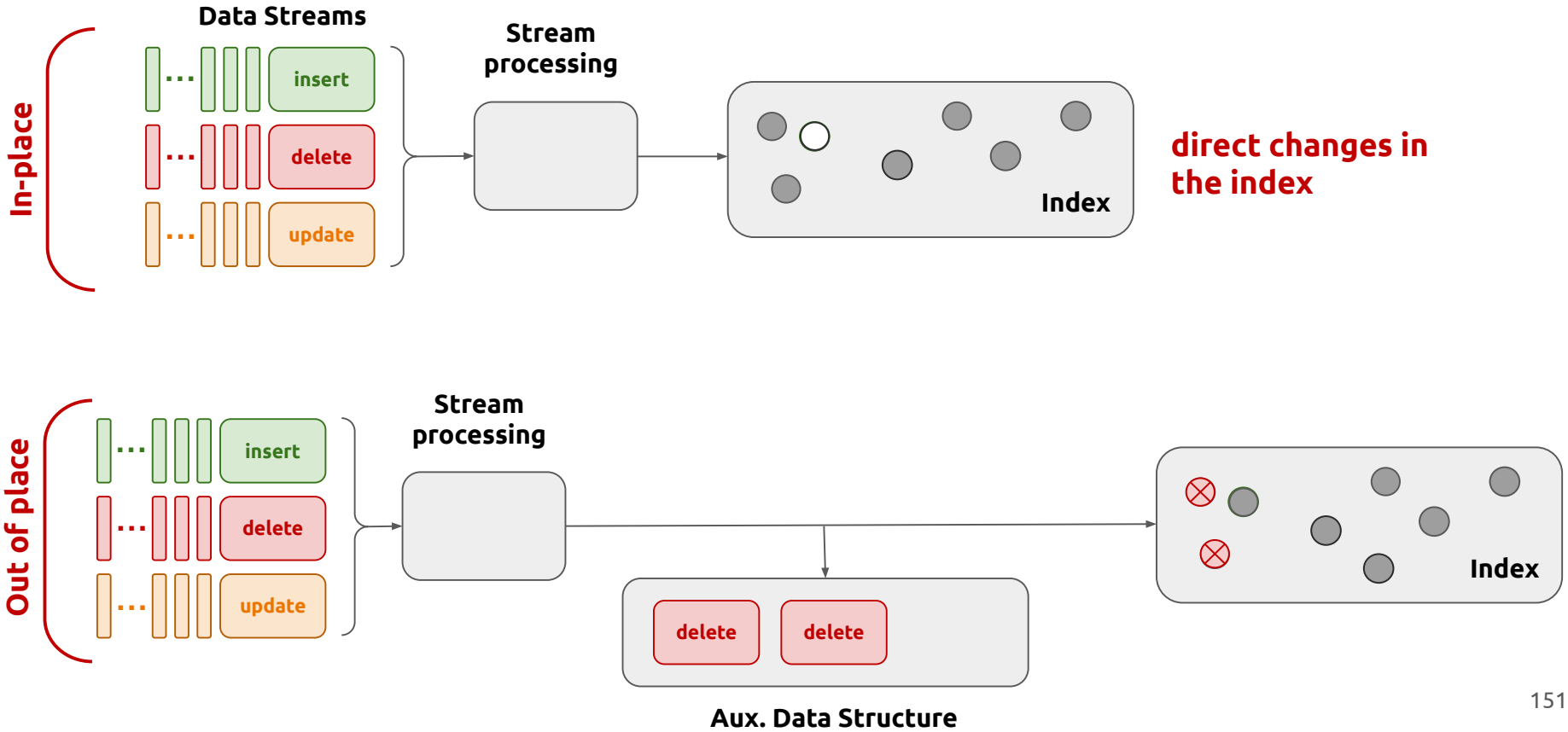
Four update models [2/3] - **Materialization**



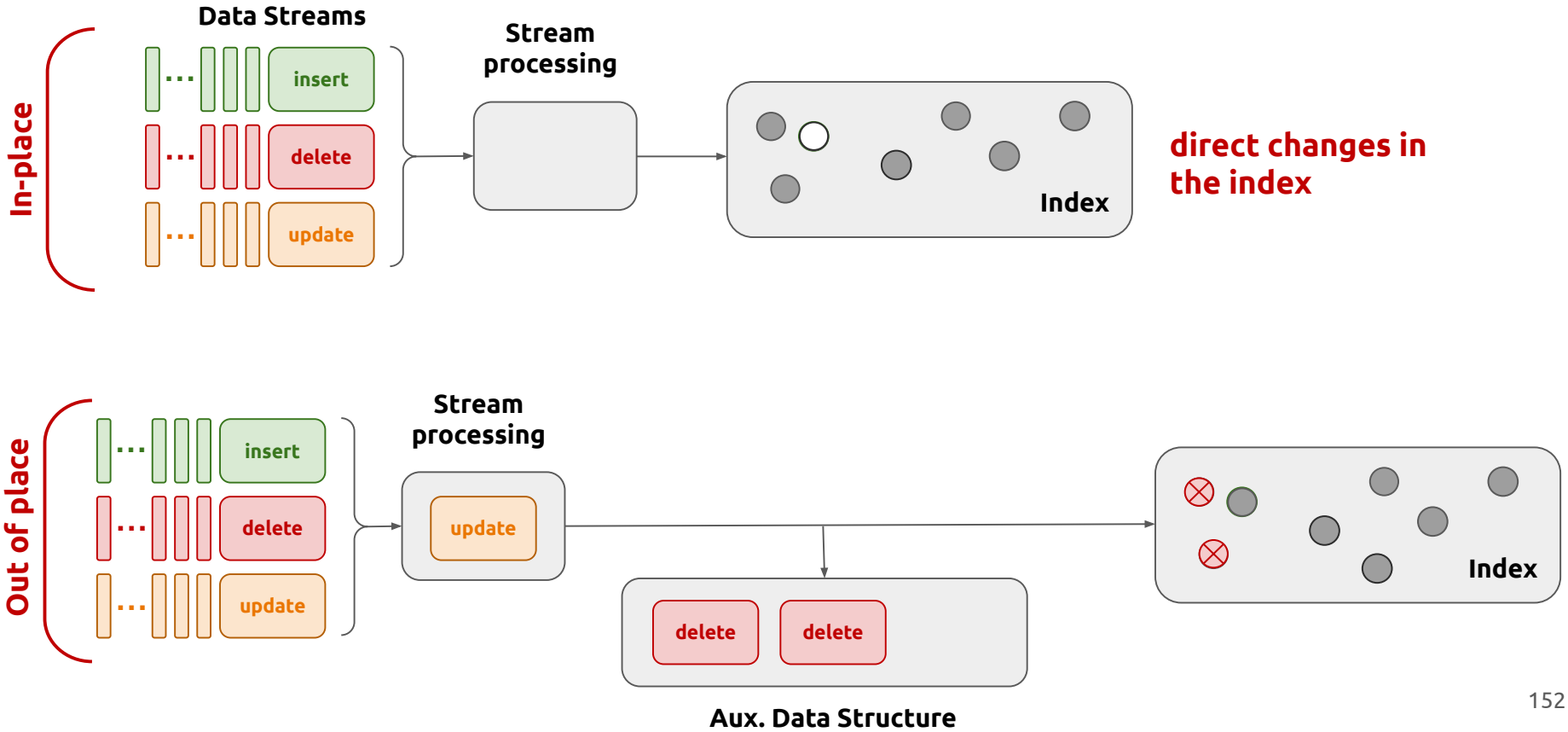
Four update models [2/3] - **Materialization**



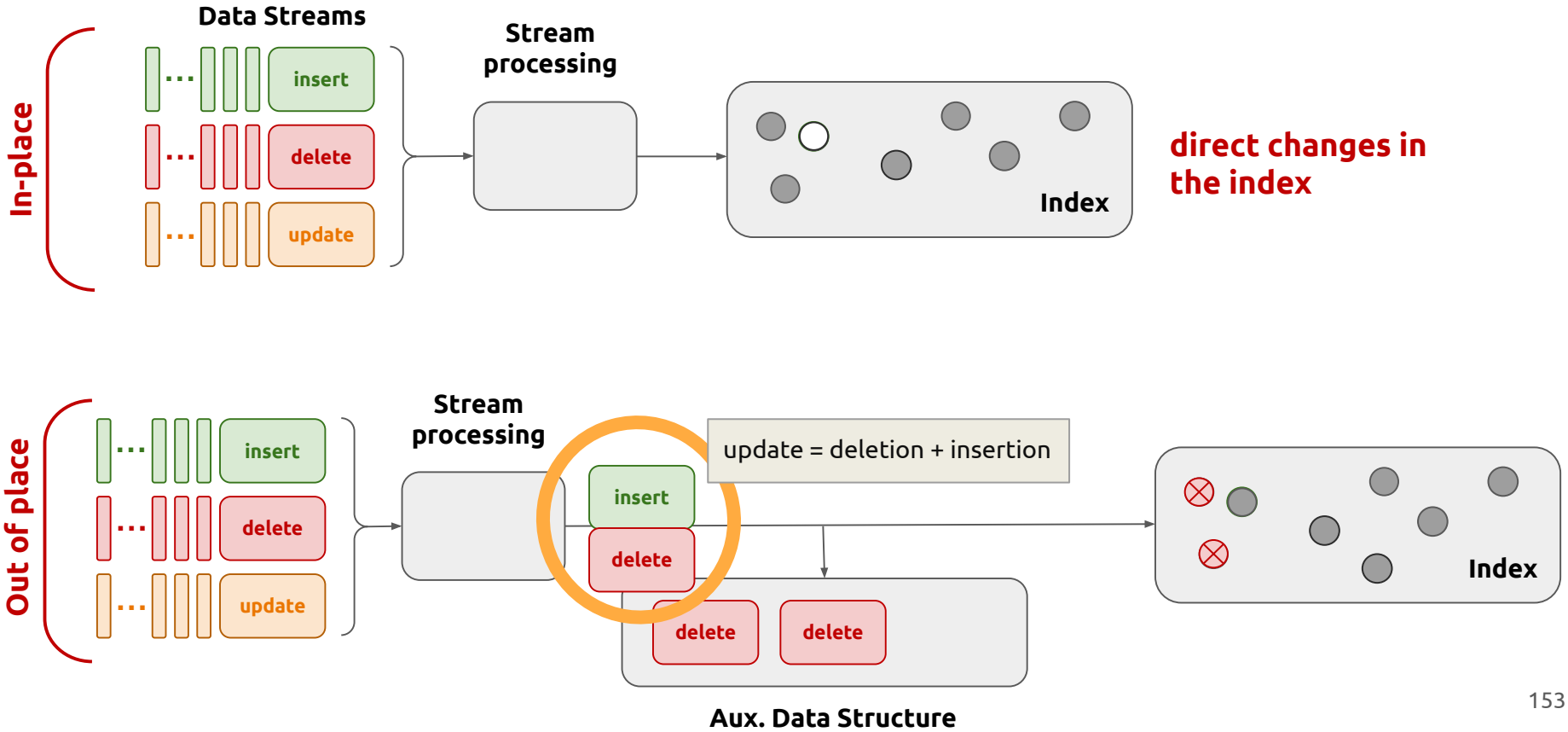
Four update models [2/3] - **Materialization**



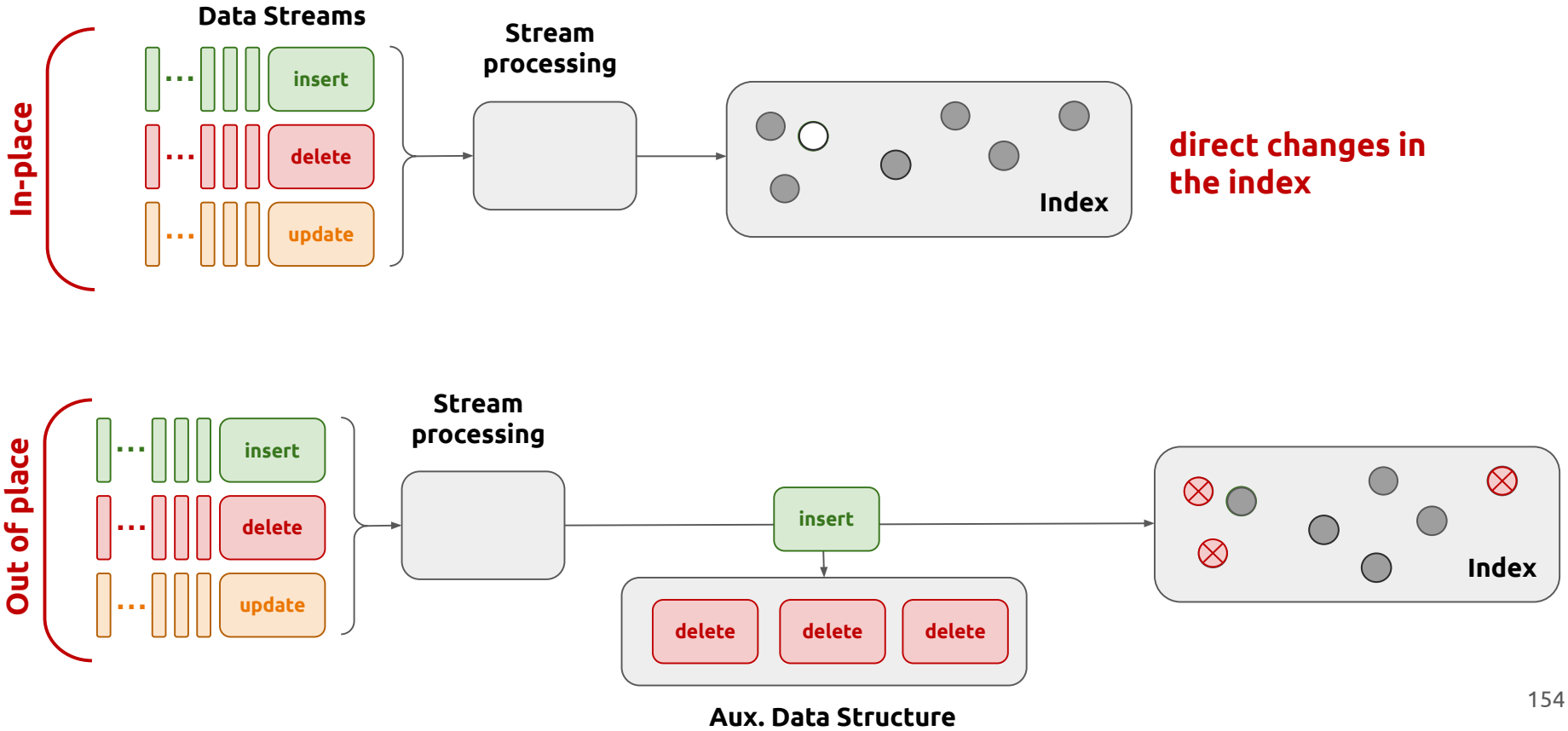
Four update models [2/3] - **Materialization**



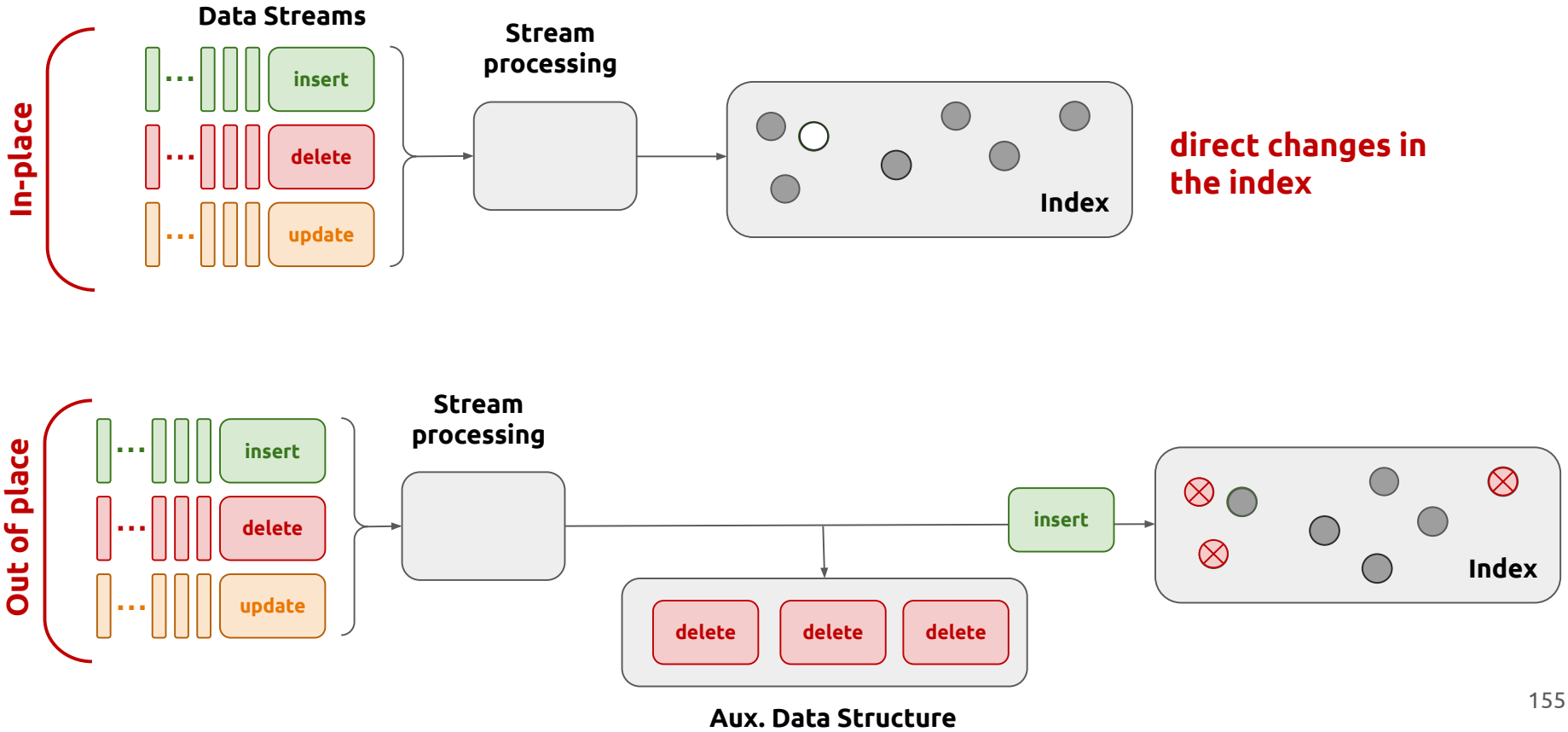
Four update models [2/3] - **Materialization**



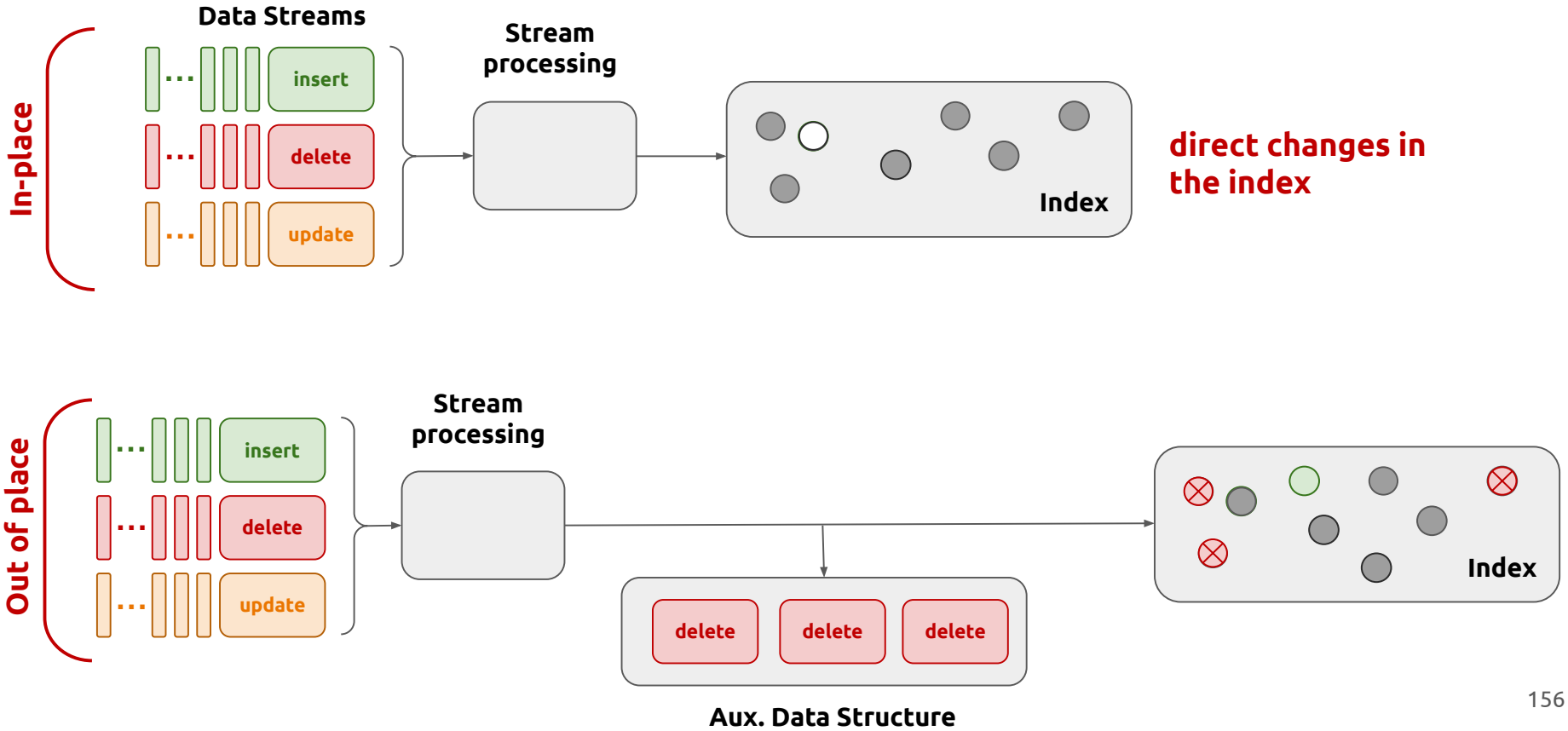
Four update models [2/3] - **Materialization**



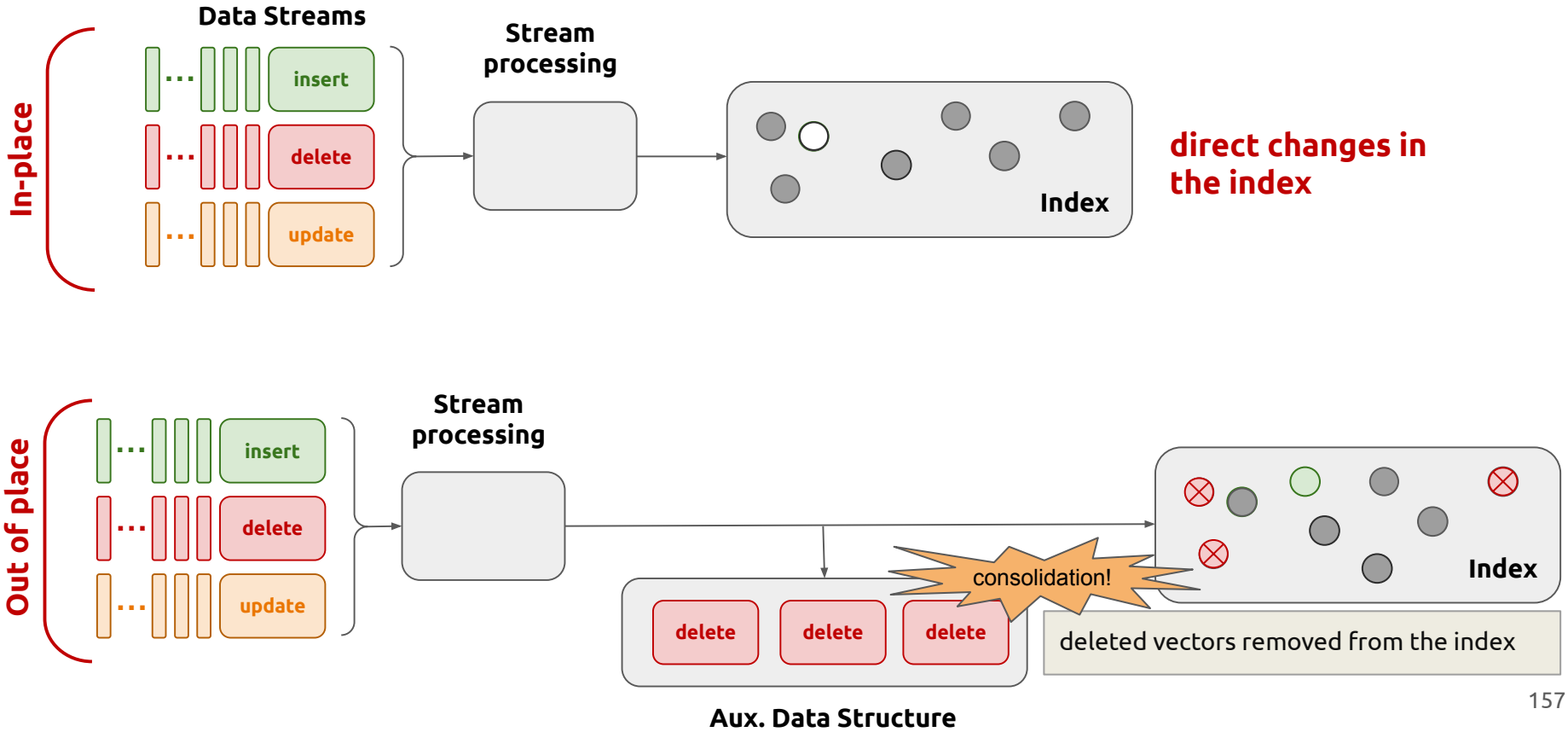
Four update models [2/3] - **Materialization**



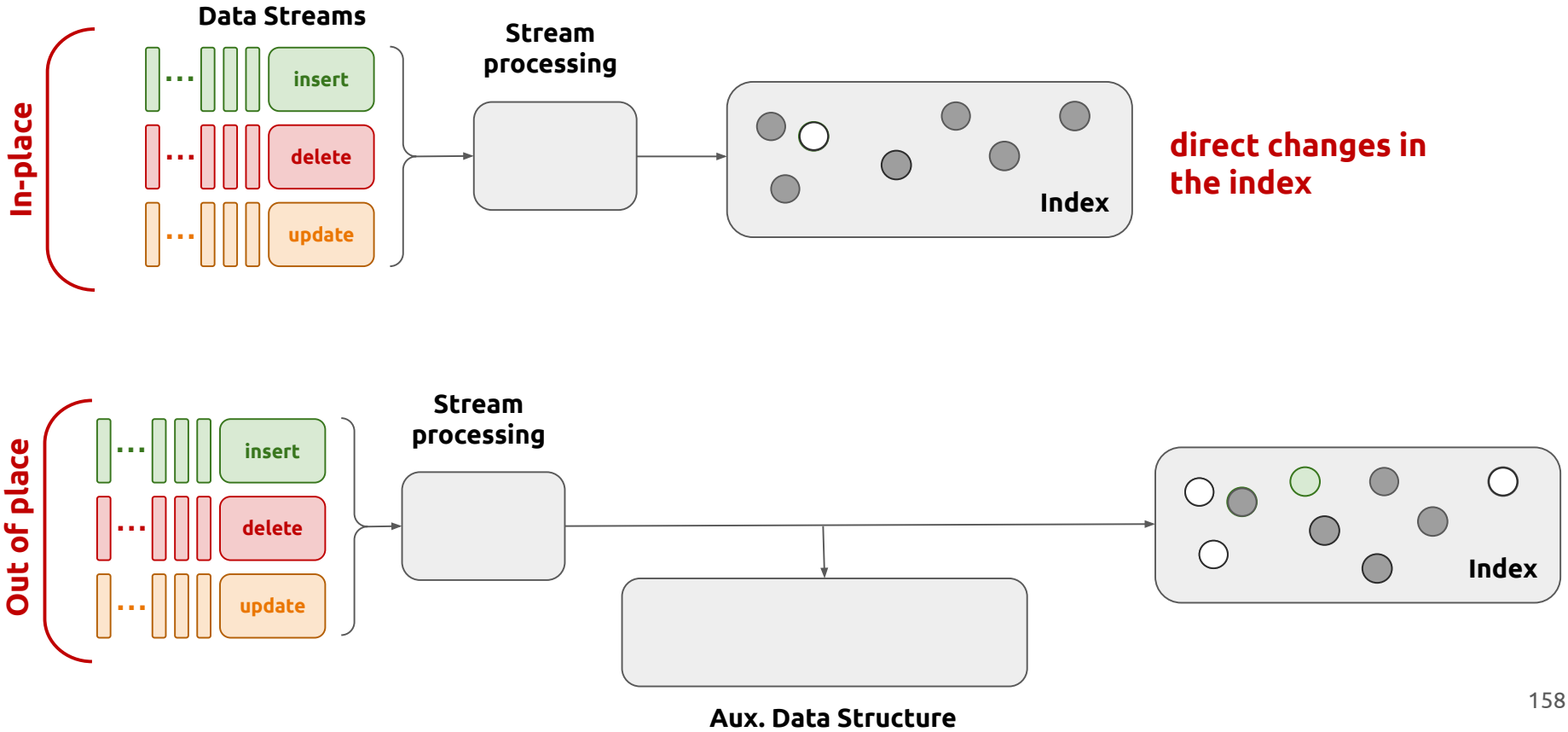
Four update models [2/3] - **Materialization**



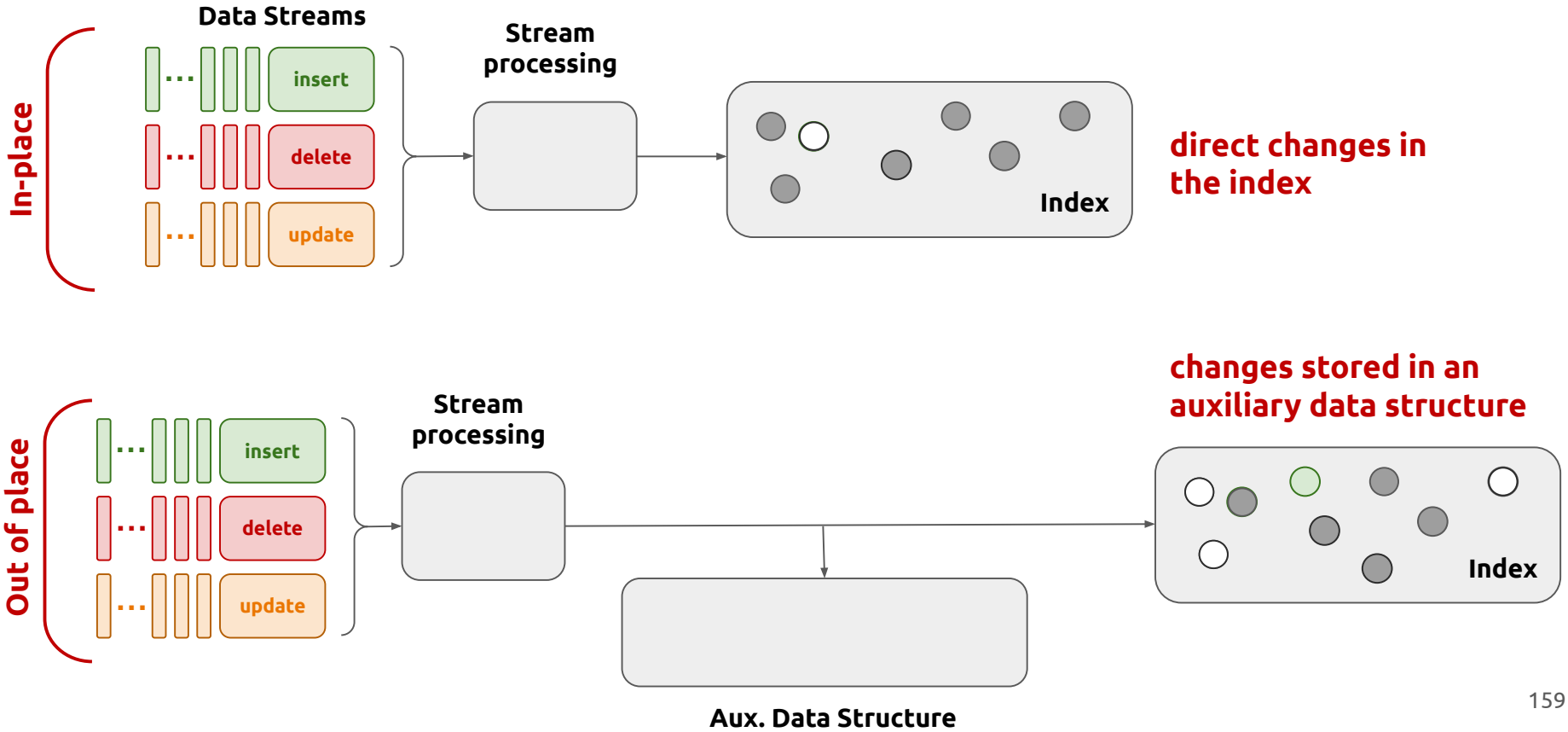
Four update models [2/3] - **Materialization**



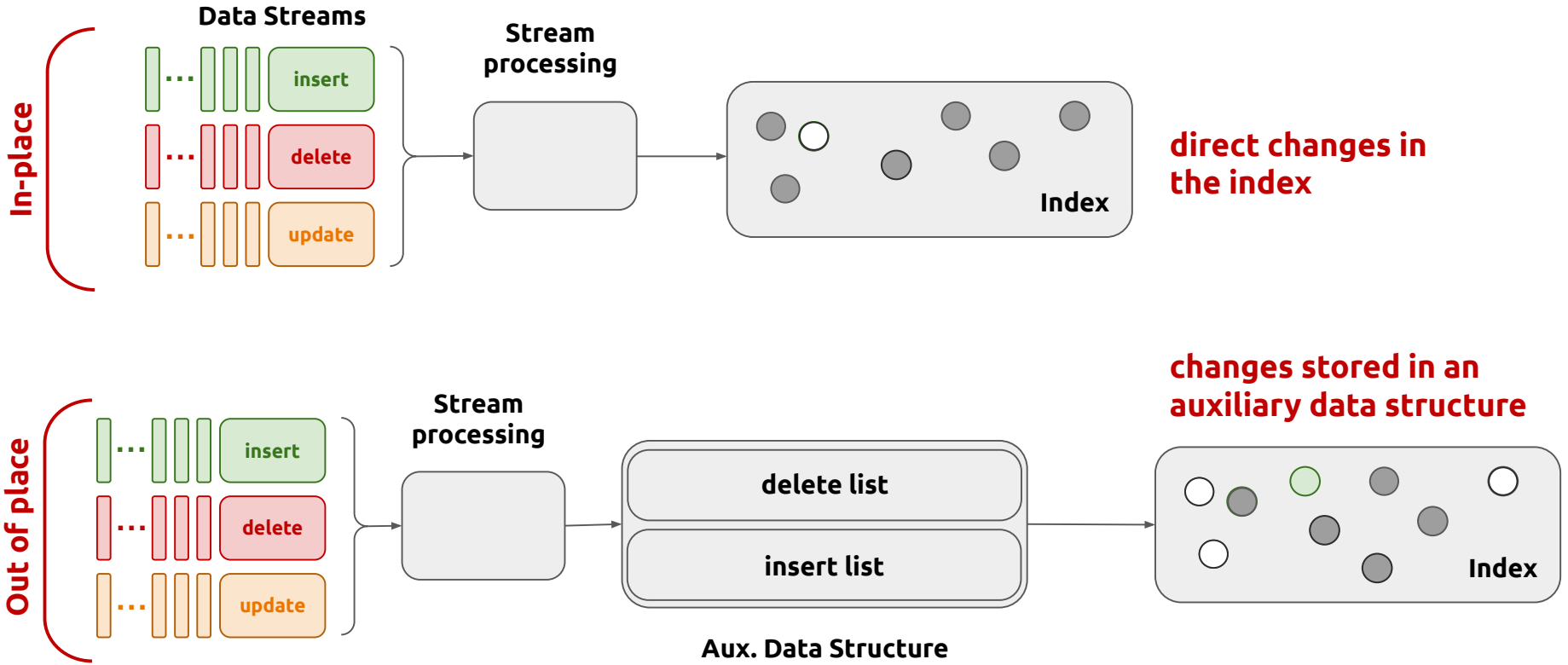
Four update models [2/3] - **Materialization**



Four update models [2/3] - **Materialization**



Four update models [2/3] - **Materialization**



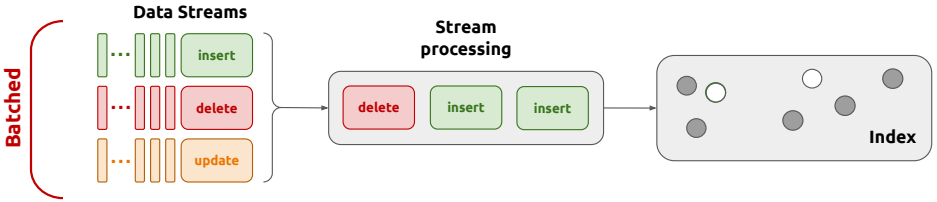
Four update models [2/3]

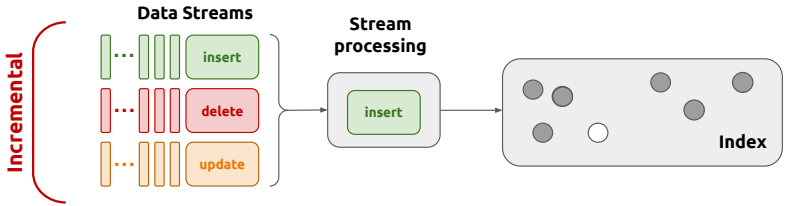
Batched

- Accumulate updates
- Periodic rebuild
- High quality
- High latency

Incremental

- Immediate visibility
- Local rebuilds
- Low latency
- Risk of index degradation





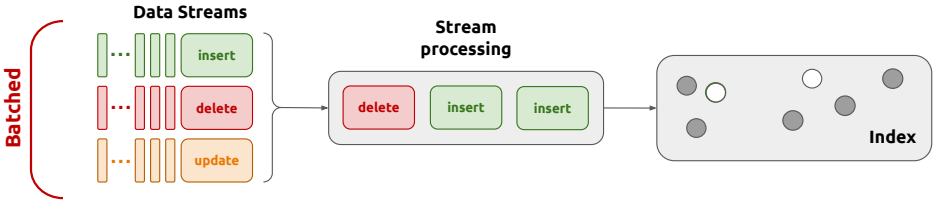
Four update models [2/3]

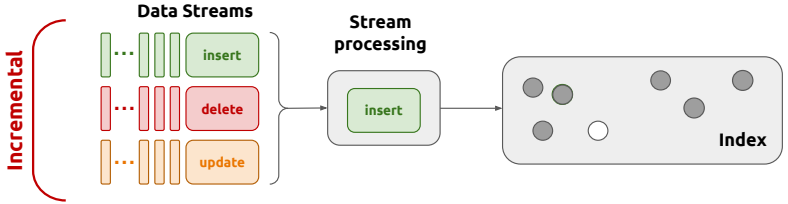
Batched

- Accumulate updates
- Periodic rebuild
- High quality
- High latency

Incremental

- Immediate visibility
- Local rebuilds
- Low latency
- Risk of index degradation





Latency vs Recall (Freshness) Tradeoff

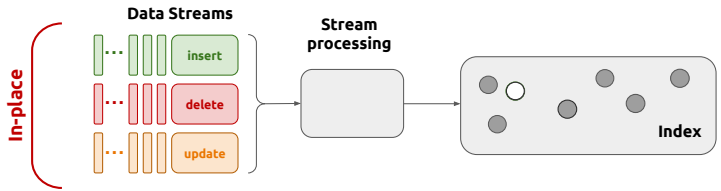
Four update models [3/3]

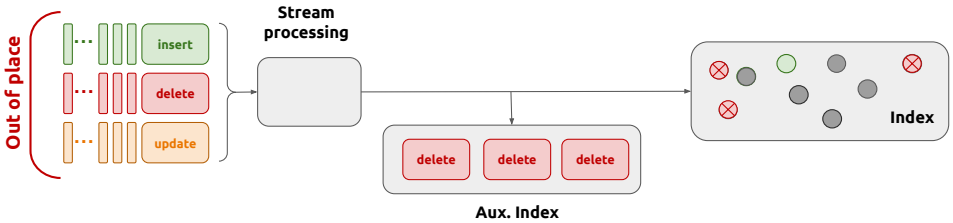
In-place

- Direct modifications on the index
- Immediate visibility of updates
- Memory efficient
- Complex concurrency

Out of place

- Updates stored separately
- Periodic merge into main index
- Simpler consistency
- Higher memory usage





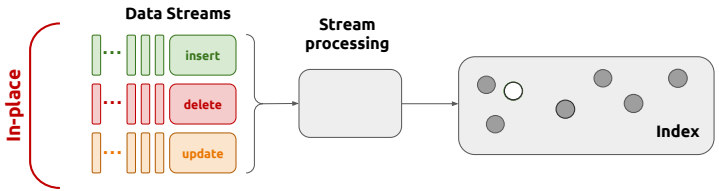
Four update models [3/3]

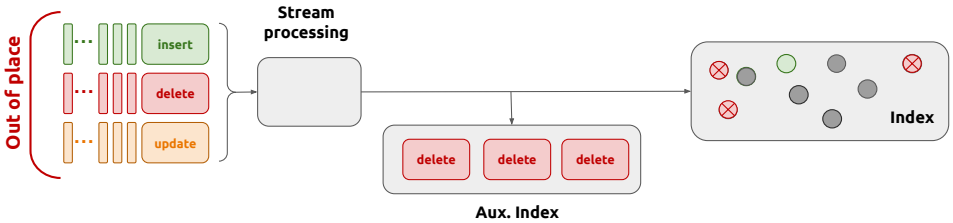
In-place

- Direct modifications on the index
- Immediate visibility of updates
- Memory efficient
- Complex concurrency

Out of place

- Updates stored separately
- Periodic merge into main index
- Simpler consistency
- Higher memory usage





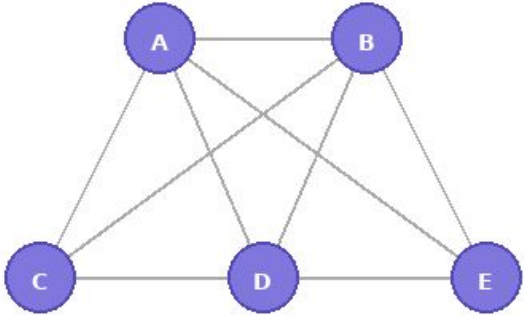
Memory vs Latency Tradeoff

Batched approaches

DEG [1/1]

Existing graph indexes are **not dynamic**, or lack connectivity guarantees

- Guaranteed connectivity via Eulerian graph property
- Incremental construction by replacing existing edges (no rebuild needed)
- Dynamic edge optimization: continuously swaps edges to minimize average neighbor distance

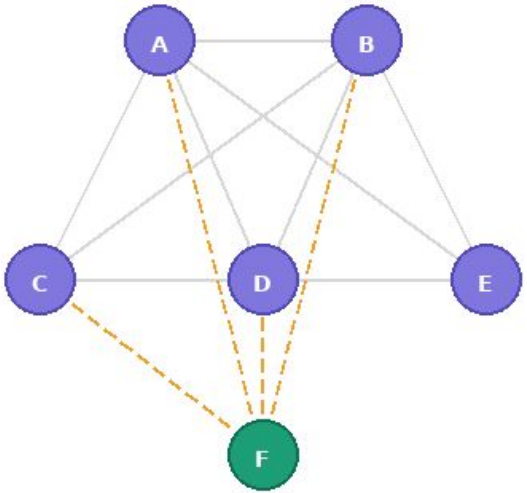


for degre=4

DEG [1/1]

Existing graph indexes are **not dynamic**, or lack connectivity guarantees

- Guaranteed connectivity via Eulerian graph property
- Incremental construction by replacing existing edges (no rebuild needed)
- Dynamic edge optimization: continuously swaps edges to minimize average neighbor distance

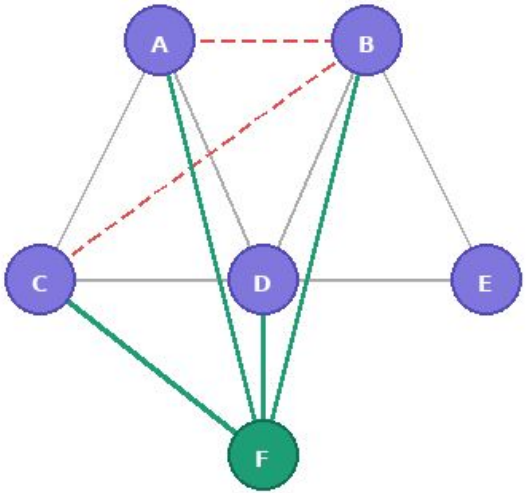


for degre=4

DEG [1/1]

Existing graph indexes are **not dynamic**, or lack connectivity guarantees

- Guaranteed connectivity via Eulerian graph property
- Incremental construction by replacing existing edges (no rebuild needed)
- Dynamic edge optimization: continuously swaps edges to minimize average neighbor distance



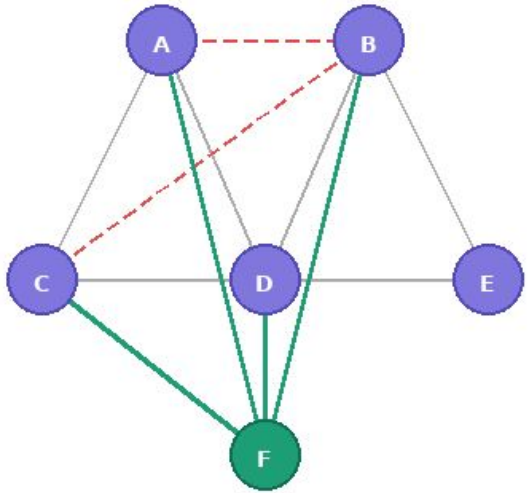
for degre=4

DEG [1/1]

Existing graph indexes are **not dynamic**, or lack connectivity guarantees

- Guaranteed connectivity via Eulerian graph property
- Incremental construction by replacing existing edges (no rebuild needed)
- Dynamic edge optimization: continuously swaps edges to minimize average neighbor distance

Results: 15–50% faster than HNSW at high recall



for degre=4

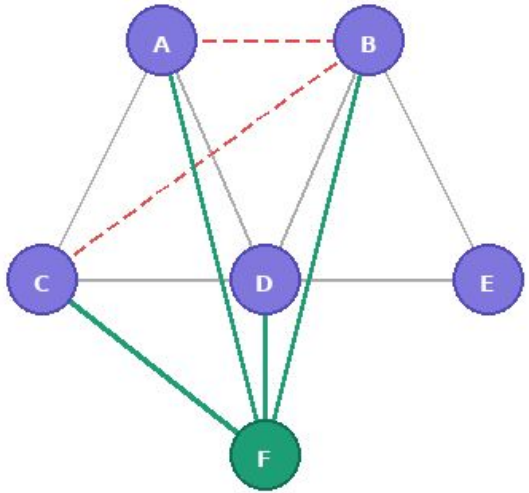
DEG [1/1]

Existing graph indexes are **not dynamic**, or lack connectivity guarantees

- Guaranteed connectivity via Eulerian graph property
- Incremental construction by replacing existing edges (no rebuild needed)
- Dynamic edge optimization: continuously swaps edges to minimize average neighbor distance

Results: 15–50% faster than HNSW at high recall

No deletion support yet



for degre=4

SVS [1/1]

Graph-based search is memory-bandwidth limited; existing compression (PQ) requires keeping full-precision vectors for re-ranking, defeating the purpose of compression.

- Locally-adaptive Vector Quantization (LVQ): per-vector scaling + scalar quantization
- Two-level quantization (LVQ-B1×B2) for residuals to recover accuracy

SVS [1/1]

Graph-based search is memory-bandwidth limited; existing compression (PQ) requires keeping full-precision vectors for re-ranking, defeating the purpose of compression.

- Locally-adaptive Vector Quantization (LVQ): per-vector scaling + scalar quantization
- Two-level quantization (LVQ-B1×B2) for residuals to recover accuracy

Up to 20.7× throughput over FAISS-IVFPQfs; up to 3× memory reduction; 33× multi-thread scaling

Static/batch workloads

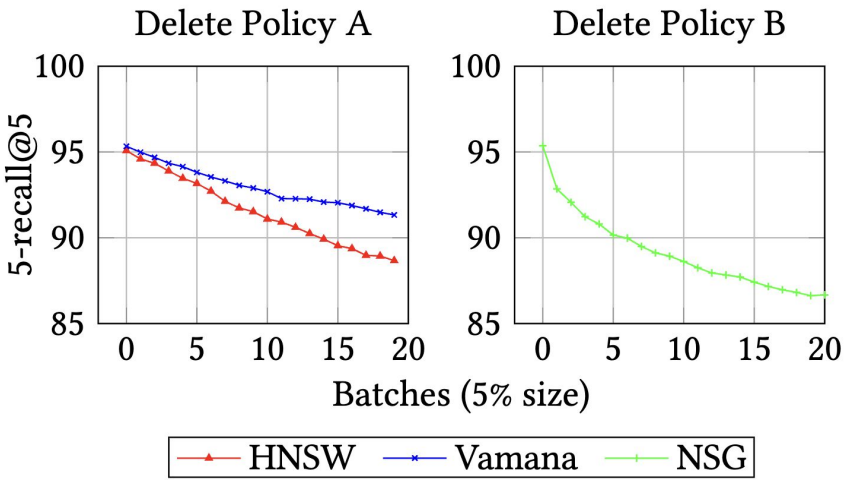
K-NN Graph approaches

FreshDiskANN [1/4]

Motivation: ***Deletes are hard***

HNSW, NSG, and Vamana

- Delete Policy A → Remove all edges
- Delete Policy B → Remove all edges
  Add all in/out pairs and prune



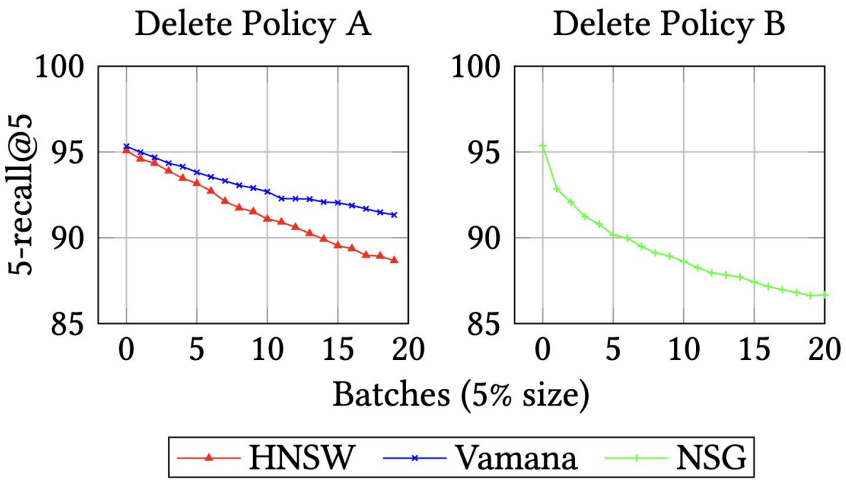
FreshDiskANN [1/4]

Motivation: ***Deletes are hard***

HNSW, NSG, and Vamana

- Delete Policy A → Remove all edges
- Delete Policy B → Remove all edges
  Add all in/out pairs and prune

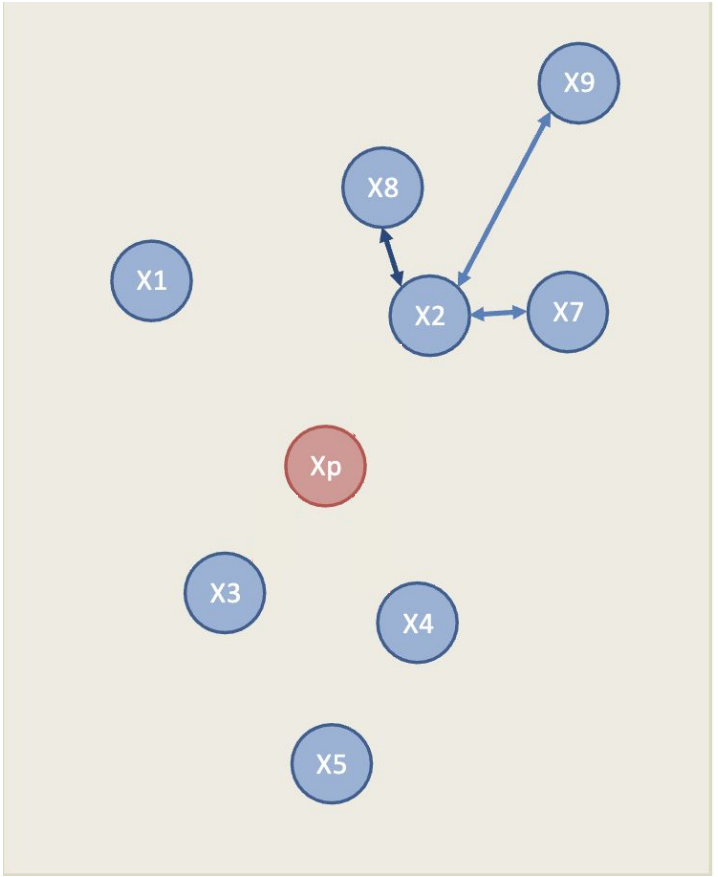
This graph becomes sparse!



FreshDiskANN [2/4]

Insert 

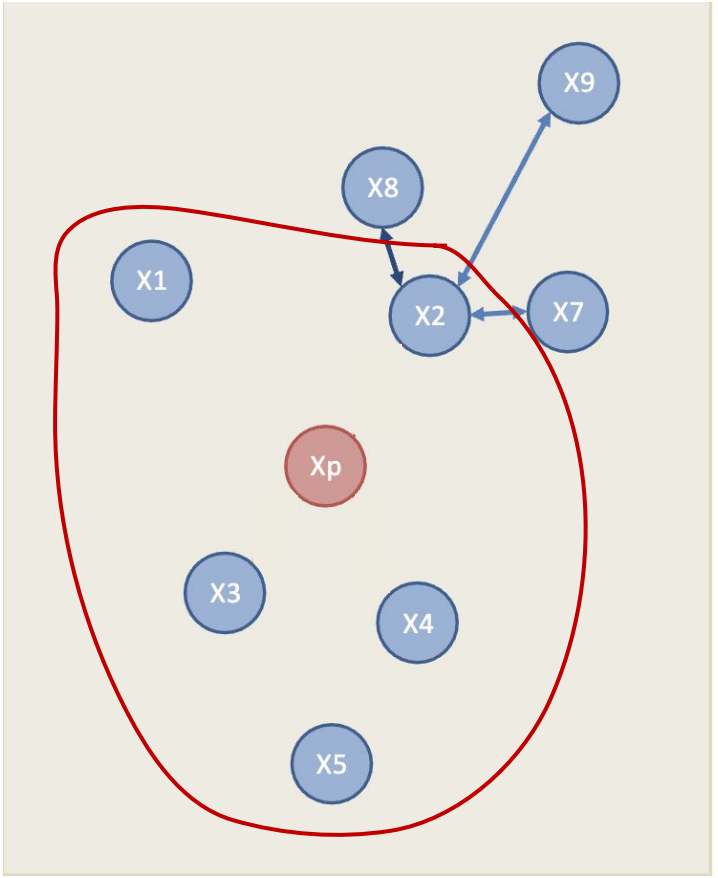
1. **Query Nearest Neighbors**: {X1, X2, X3, X4, X5}



FreshDiskANN [2/4]

Insert 

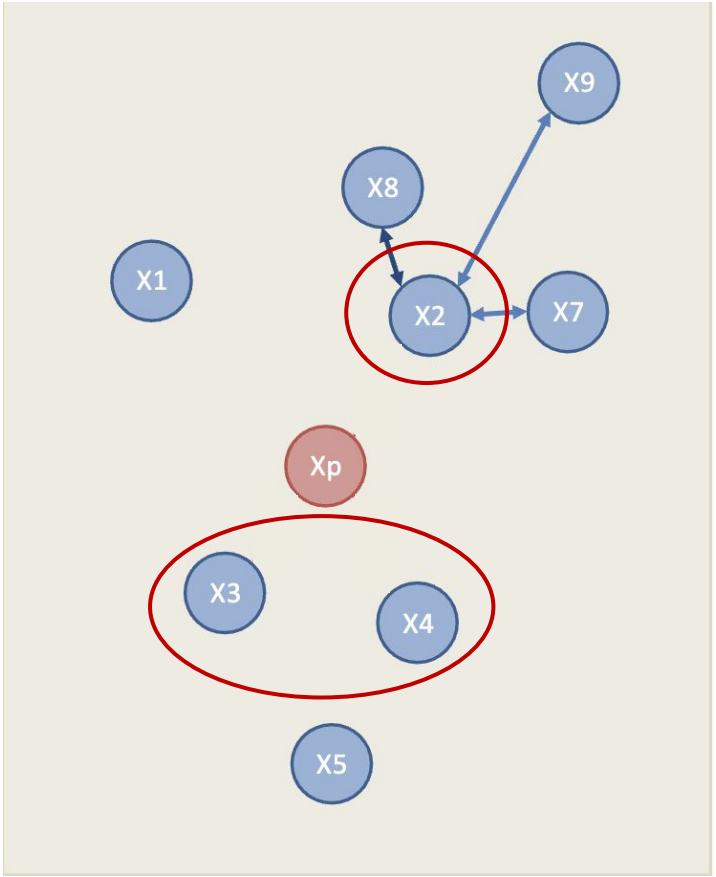
1. **Query Nearest Neighbors**: {X1, X2, X3, X4, X5}



FreshDiskANN [2/4]

Insert 

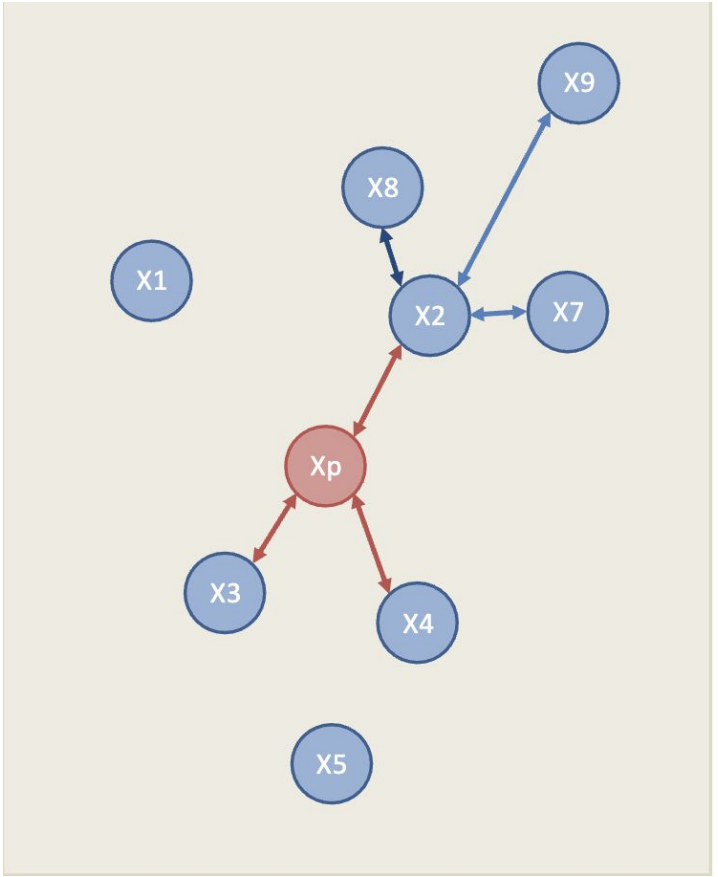
1. **Query Nearest Neighbors**: {X1, X2, X3, X4, X5}
2. **Find Candidate Out-Neighbors:** {X2, X3, X4}



FreshDiskANN [2/4]

Insert 

1. **Query Nearest Neighbors**: {X1, X2, X3, X4, X5}
2. **Find Candidate Out-Neighbors:** {X2, X3, X4}
3. **Add Bi-Directional Edges:** (Xp <-> X2), (Xp <-> X3), (Xp <-> X4)

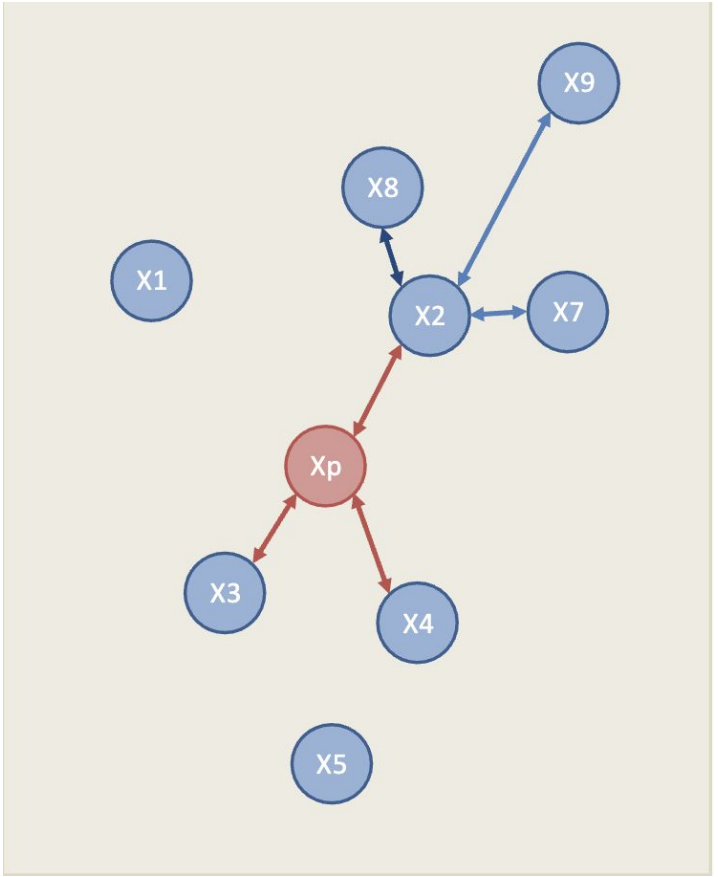


FreshDiskANN [2/4]

Insert 

1. **Query Nearest Neighbors**: {X1, X2, X3, X4, X5}
2. **Find Candidate Out-Neighbors:** {X2, X3, X4}
3. **Add Bi-Directional Edges:** (Xp <-> X2), (Xp <-> X3), (Xp <-> X4)

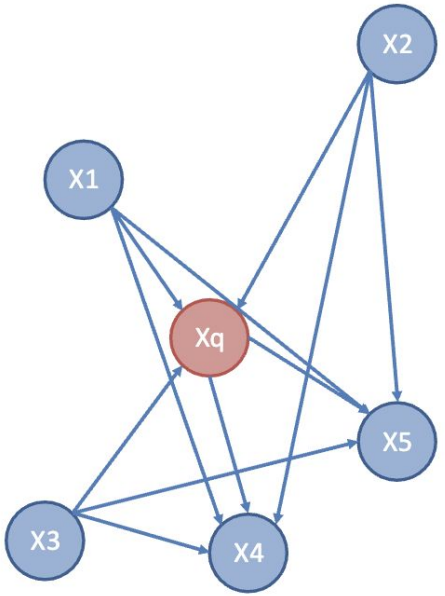
Concurrency Control with **lock**



FreshDiskANN [3/4]

Update

- **Lazy Deletion:**
  - Stay there with In-neighbours (X1, X2) and out-neighbours (X3, X4, X5)
  - Add to Delete List
- **Search Adjustments:**
  - Filter result with Delete List
- **Delete Consolidation**
  - Batch update
  - Delete node / add edges
  - Prune



Delete List:

IP-DiskANN [1/2]

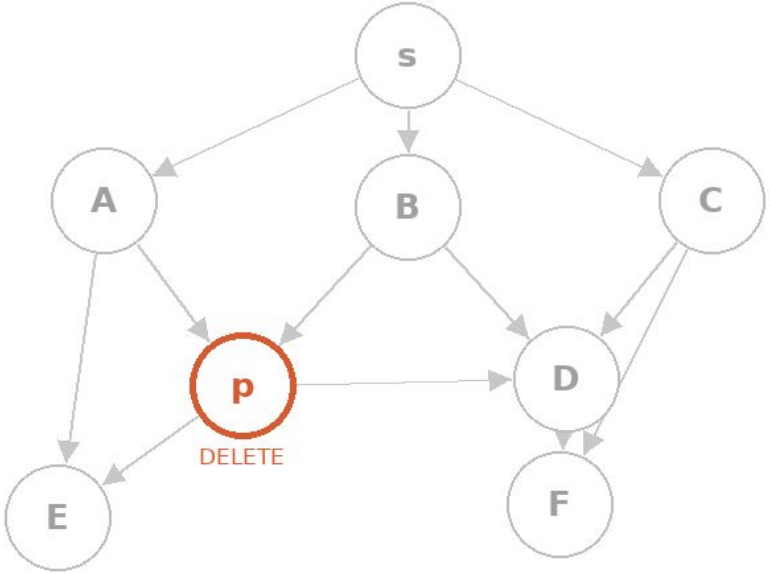
Handles deletions without batch consolidation

- FreshDiskANN accumulates deletions and consolidates **periodically**, causing **latency spikes** and **resource overheads**
- IP-DiskANN processes every insertion and deletion **in-place**, maintaining a single live graph at all times

IP-DiskANN vs FreshDiskANN
35% faster deletions and recall improvement

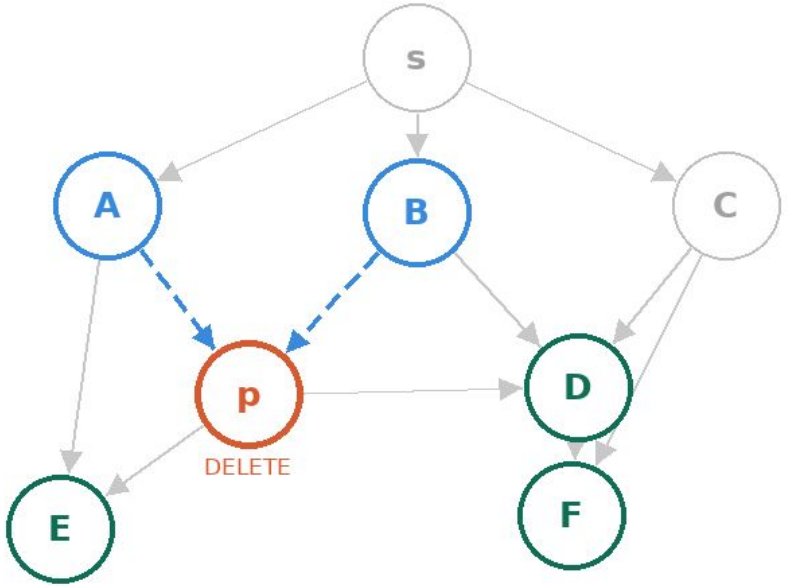
IP-DiskANN [2/2]

- **GreedySearch** with P as query



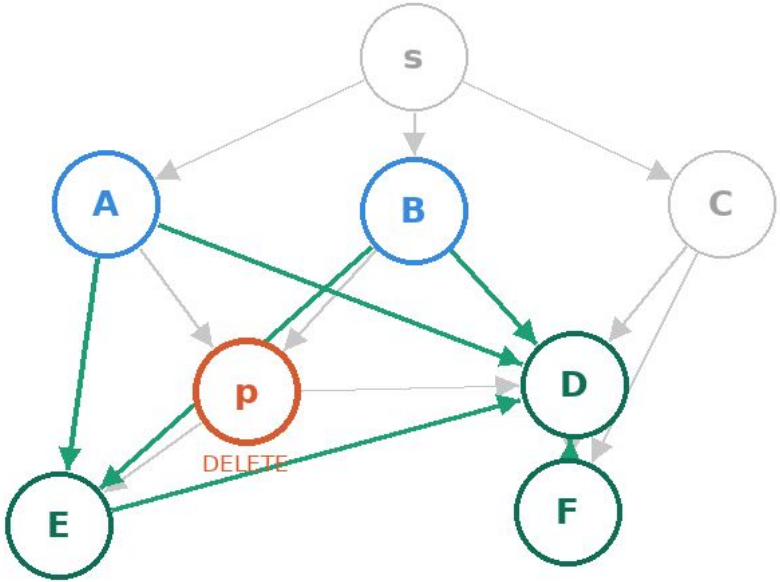
IP-DiskANN [2/2]

- **GreedySearch** with P as query
- Find approximate in-neighbors: {A, B}



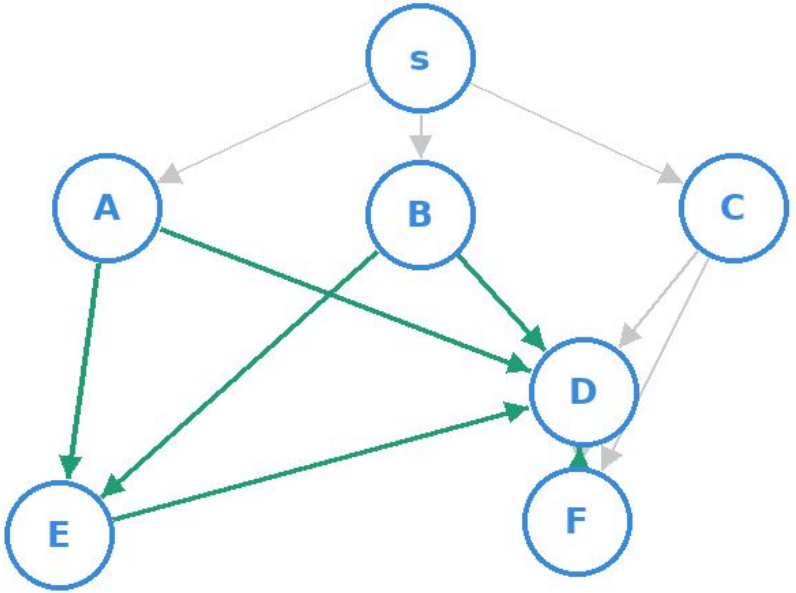
IP-DiskANN [2/2]

- **GreedySearch** with P as query
- Find approximate in-neighbors: {A, B}
- Rewire in-neighbors: add closest Candidates {D, E, F} to each {A, B}
- Rewire out-neighbors: for each w in Nout(p), find nodes near w and add edge → w



IP-DiskANN [2/2]

- **GreedySearch** with P as query
- Find approximate in-neighbors: {A, B}
- Rewire in-neighbors: add closest Candidates {D, E, F} to each {A, B}
- Rewire out-neighbors: for each w in Nout(p), find nodes near w and add edge → w
- Remove p immediately



CleANN [1/3]

Problem with existing systems:

- **FreshDiskANN:** periodic global consolidation → throughput spikes
- **IP-DiskANN:** in-place deletions but no concurrency
- Both suffer from insertion robustness

CleANN's solution:

- **Bridge Building:** adds "shortcut" edges during search tree traversal
- **On-the-fly Consolidation:** cleanup triggered by query traffic, not batch scans
- **Semi-Lazy Cleaning:** deleted nodes recycled after C consolidations, no full cleanup needed

CleANN [1/3]

Problem with existing systems:

- **FreshDiskANN:** periodic global consolidation → throughput spikes
- **IP-DiskANN:** in-place deletions but no concurrency
- Both suffer from insertion robustness

CleANN's solution:

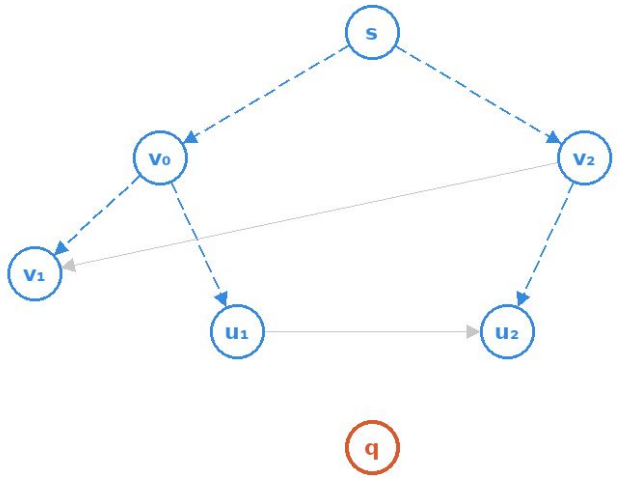
- **Bridge Building:** adds "shortcut" edges during search tree traversal
- **On-the-fly Consolidation:** cleanup triggered by query traffic, not batch scans
- **Semi-Lazy Cleaning:** deleted nodes recycled after C consolidations, no full cleanup needed

CleANN vs FreshDiskANN: 7–1200x throughput, similar recall
CleANN vs IP-DiskANN: 3x update throughput, 1.5x search throughput (single thread)

explore how the parameter selection affects performance

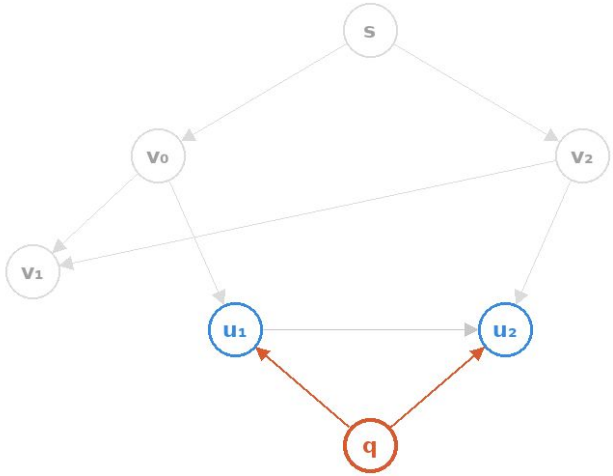
CleANN [2/3] - Insert

- **GreedySearch** with q as query



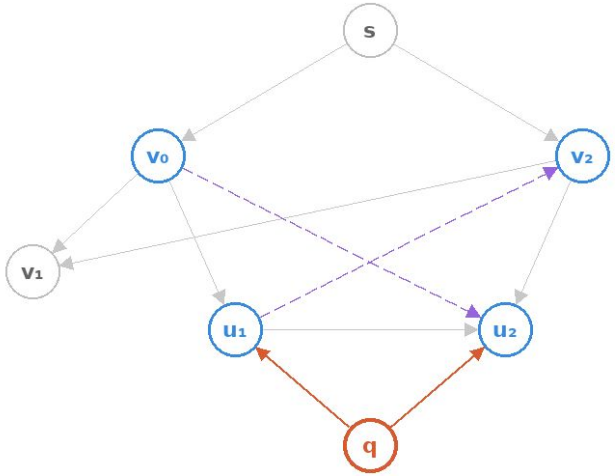
CleANN [2/3] - Insert

- **GreedySearch** with q as query
- **Find ANN**: $\{u_1, u_2\}$; q is connected.



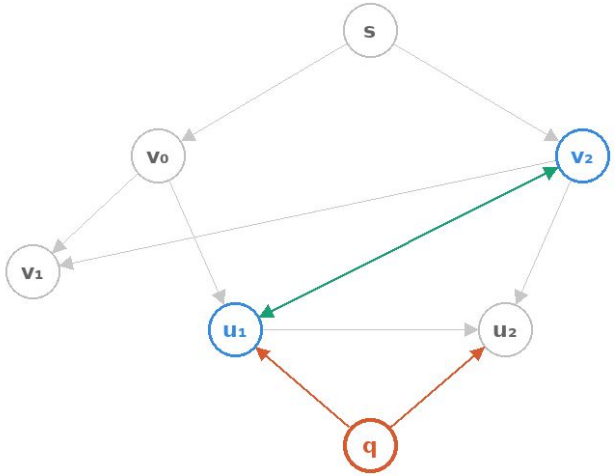
CleANN [2/3] - Insert

- **GreedySearch** with q as query
- **Find ANN:** $\{u_1, u_2\}$; q is connected.
- Find **bridge** candidates $\{v_0 \leftrightarrow u_2, u_1 \leftrightarrow v_2\}$



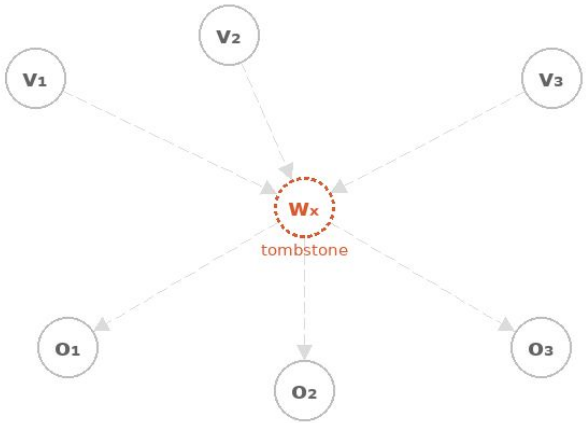
CleANN [2/3] - Insert

- **GreedySearch** with q as query
- **Find ANN**: $\{u_1, u_2\}$; q is connected.
- Find **bridge** candidates $\{v_0 \leftrightarrow u_2, u_1 \leftrightarrow v_2\}$
- $v_0 \rightarrow u_2$ is pruned; bridge edge $u_1 \leftrightarrow v_2$ is added, **improving navigability**.



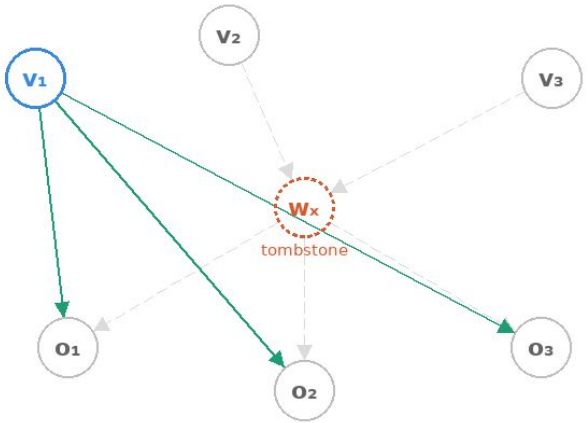
CleANN [3/3] - Delete

- x is deleted — w_x **becomes a tombstone**, queries still traverse it but filter it out.



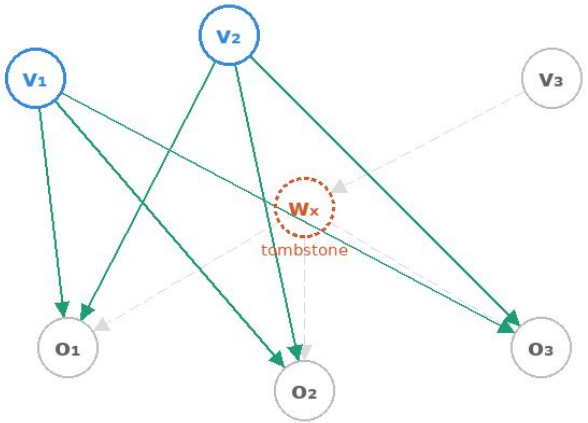
CleANN [3/3] - Delete

- x is deleted — w_x **becomes a tombstone**, queries still traverse it but filter it out.
- A search traverses $v_1 \rightarrow w_x$; v_1 **immediately consolidates**, absorbing $\{o_1, o_2, o_3\}$ into $N(v_1)$. Edge $v_1 \rightarrow w_x$ removed.



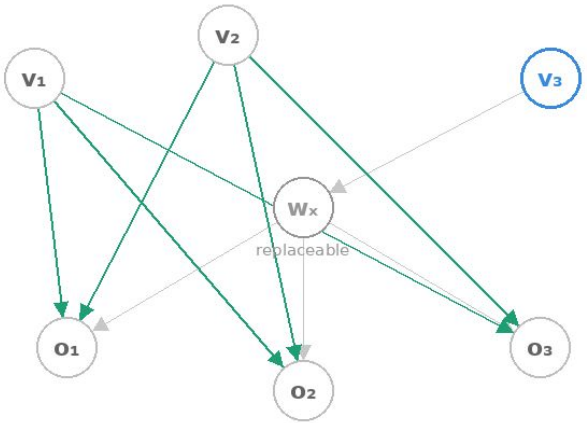
CleANN [3/3] - Delete

- x is deleted — w_x **becomes a tombstone**, queries still traverse it but filter it out.
- A search traverses $v_1 \rightarrow w_x$; v_1 **immediately consolidates**, absorbing $\{o_1, o_2, o_3\}$ into $N(v_1)$. Edge $v_1 \rightarrow w_x$ removed.
- v_2 also consolidates with w_x. $H(w_x)$ **hits threshold** C=2 → w_x marked **replaceable**.



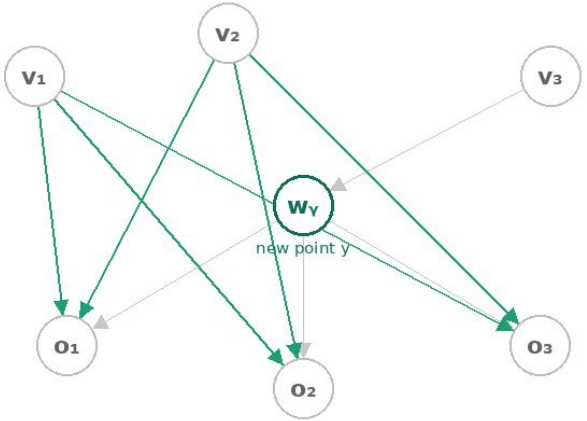
CleANN [3/3] - Delete

- x is deleted — w_x **becomes a tombstone**, queries still traverse it but filter it out.
- A search traverses $v_1 \rightarrow w_x$; v_1 **immediately consolidates**, absorbing $\{o_1, o_2, o_3\}$ into $N(v_1)$. Edge $v_1 \rightarrow w_x$ removed.
- v_2 also consolidates with w_x. $H(w_x)$ **hits threshold** C=2 → w_x marked **replaceable**.



CleANN [3/3] - Delete

- x is deleted — w_x **becomes a tombstone**, queries still traverse it but filter it out.
- A search traverses $v_1 \rightarrow w_x$; v_1 **immediately consolidates**, absorbing $\{o_1, o_2, o_3\}$ into $N(v_1)$. Edge $v_1 \rightarrow w_x$ removed.
- v_2 also consolidates with w_x. $H(w_x)$ **hits threshold** C=2 → w_x marked **replaceable**.
- New point y **reuses** w_x's slot (now w_y).



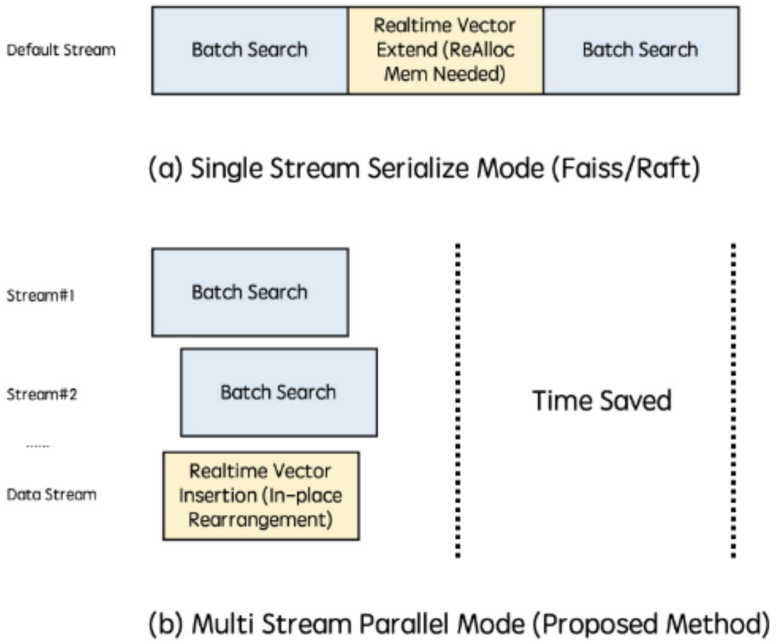
IVF approaches

RTAMS-GANNS [1/1]

Fixes real-time insertion on GPUs with two ideas:

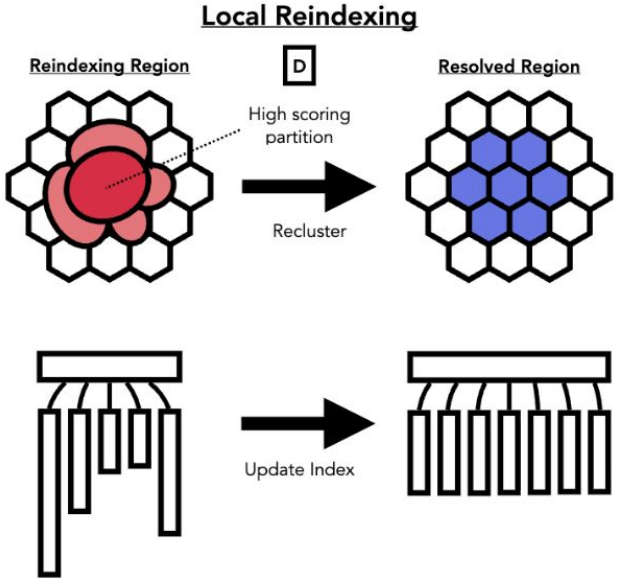
- **Memory blocks:** instead of one big array, memory is pre-split into small linked blocks — **new vectors** claim the next free block
- **Multi-stream execution:** search queries each get their own stream, all run in parallel
- Result: **40–80% lower latency**, deployed live at Xiaohongshu serving 100M+ users daily

Only for IVF



Streaming for IVF [1/2] - Ada-IVF

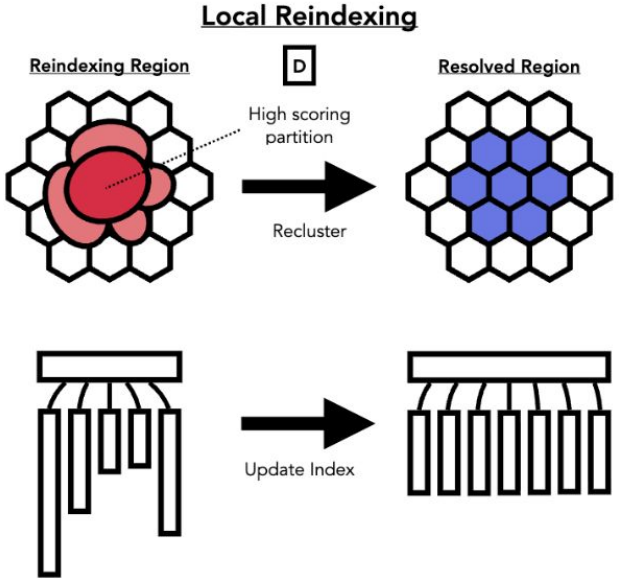
Problem: Centroids become outdated as distribution shifts



Streaming for IVF [1/2] - Ada-IVF

Problem: Centroids become outdated as distribution shifts

Can be sensitive to distribution shift

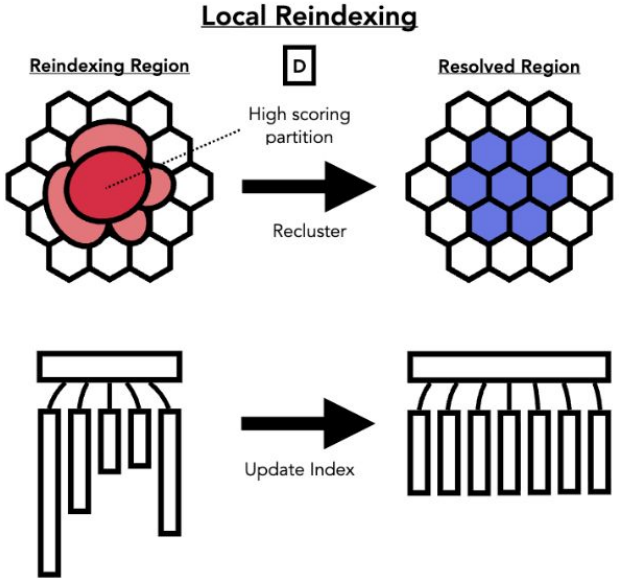


Streaming for IVF [1/2] - Ada-IVF

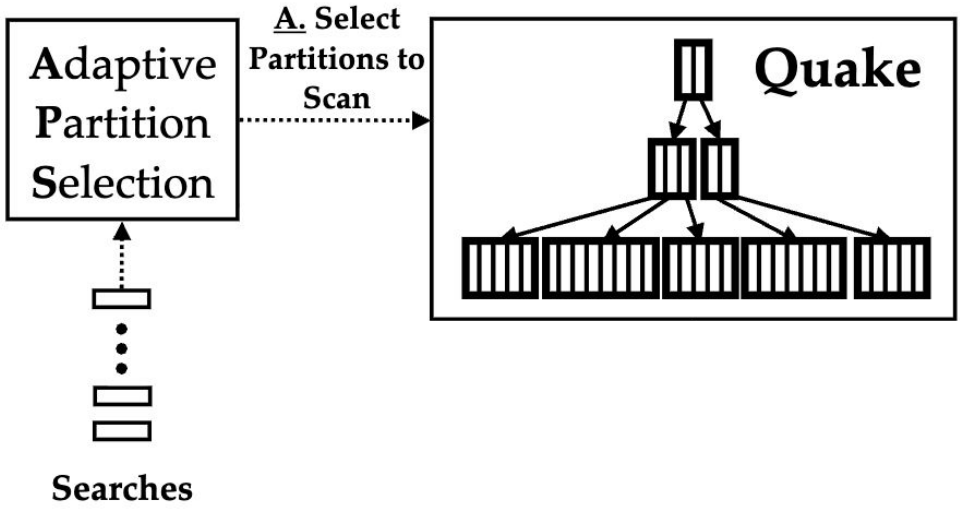
Problem: Centroids become outdated as distribution shifts

Can be sensitive to distribution shift

Local updates for IVF

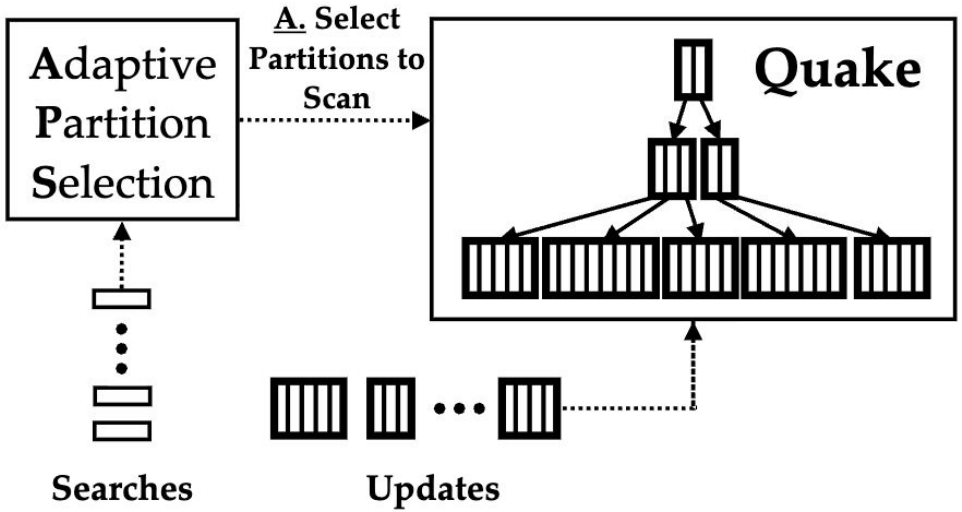


Streaming for IVF [2/2] - Quake



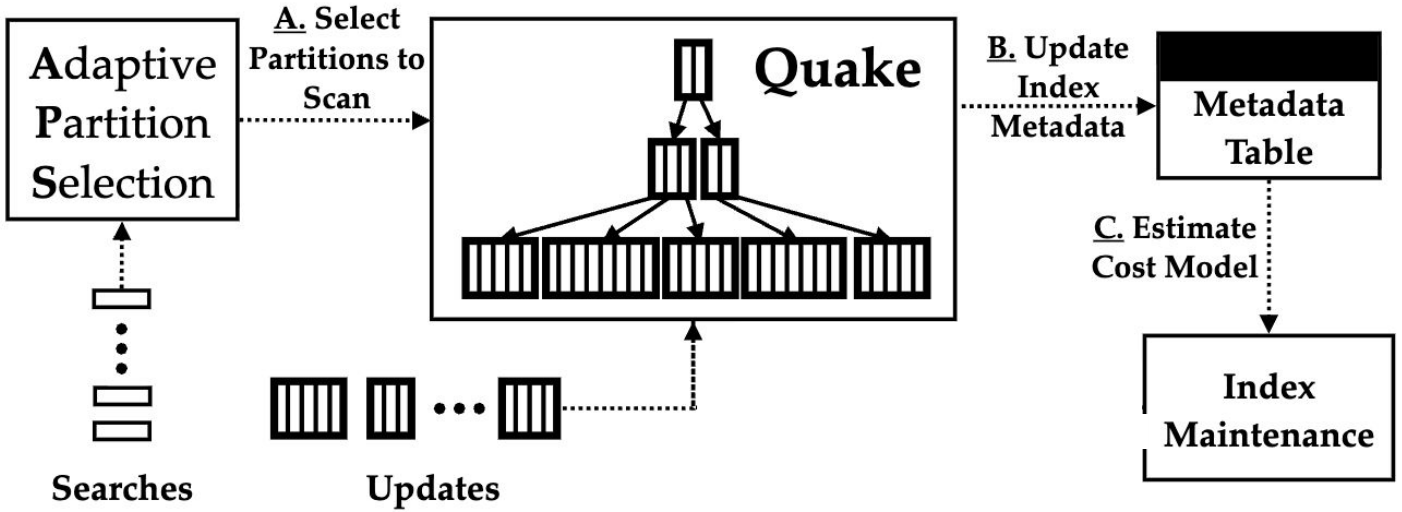
Problem: Data is usually dynamic & skewed

Streaming for IVF [2/2] - Quake



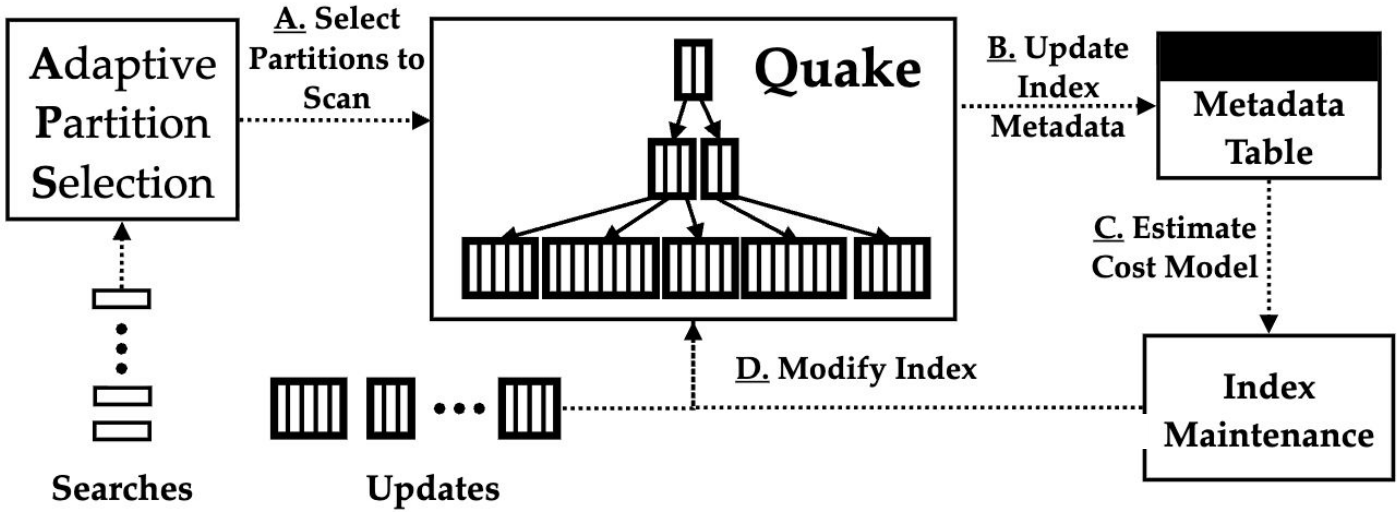
Problem: Data is usually dynamic & skewed

Streaming for IVF [2/2] - Quake



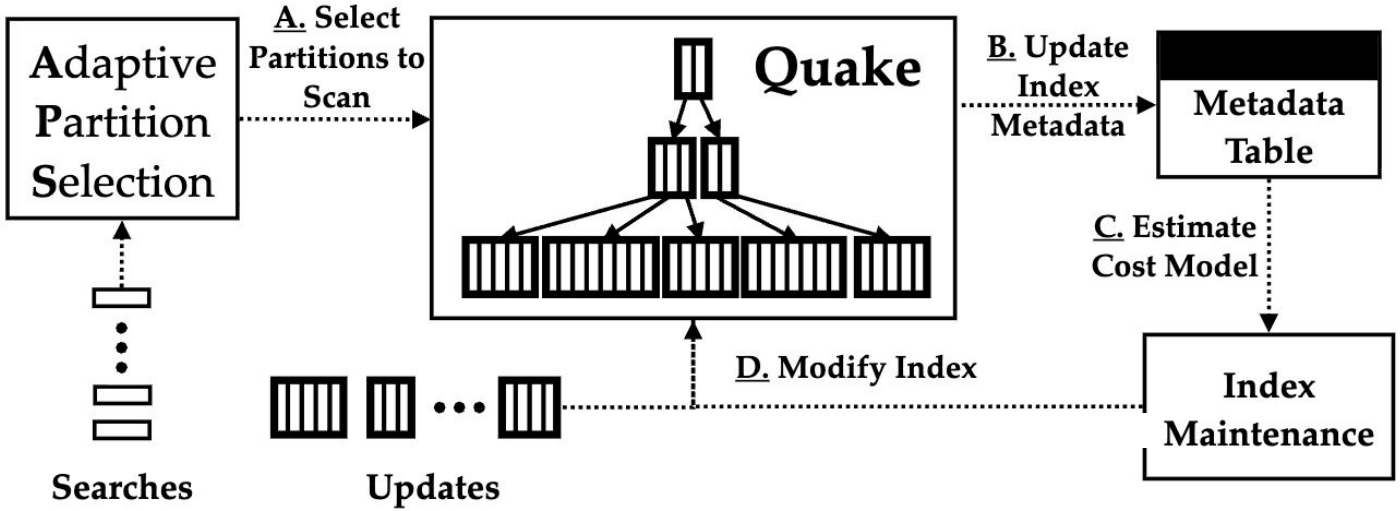
Problem: Data is usually dynamic & skewed

Streaming for IVF [2/2] - Quake



Problem: Data is usually dynamic & skewed

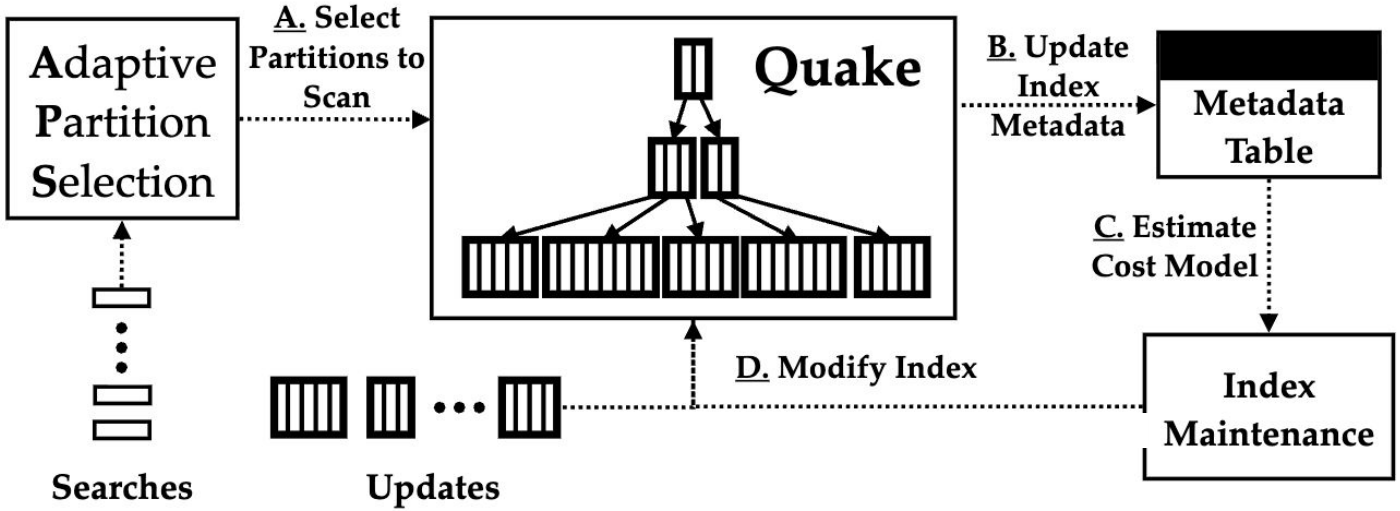
Streaming for IVF [2/2] - Quake



Adaptive maintenance guided by cost model

Problem: Data is usually dynamic & skewed

Streaming for IVF [2/2] - Quake



Adaptive maintenance guided by cost model

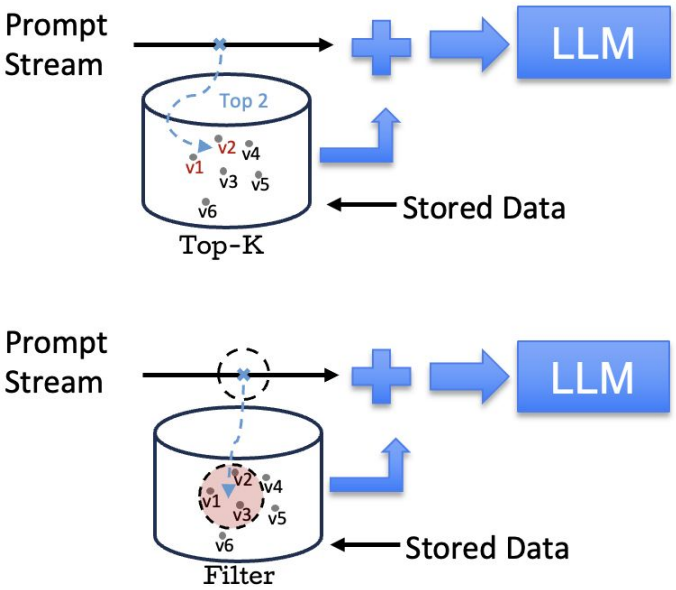
explore how searches, updates & maintenance can run in parallel

Problem: Data is usually dynamic & skewed

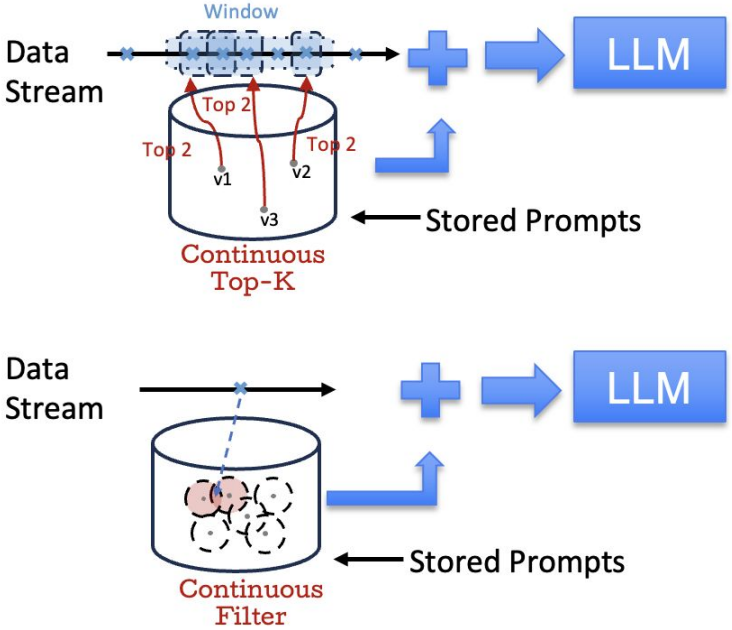
Other Systems

VectraFlow [1/2]

Traditional RAG

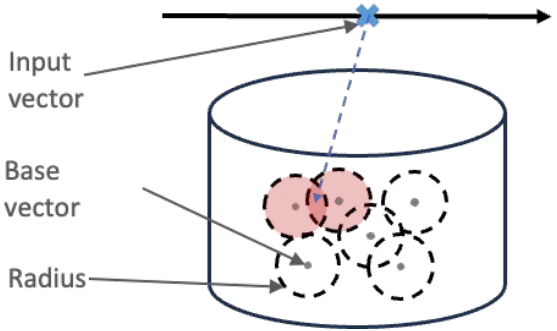


Continuous RAG



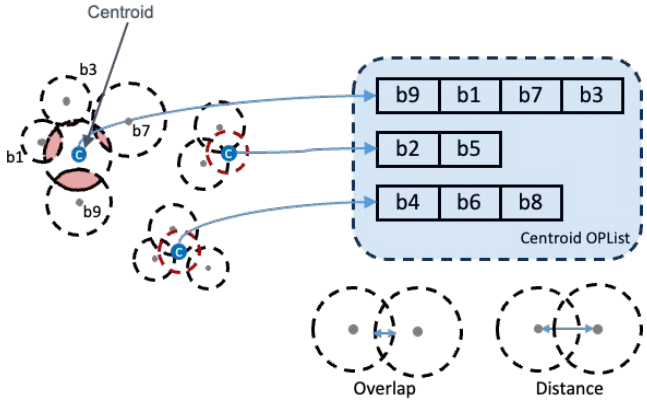
VectraFlow [2/2]- iV-Filter

- Continuous prompts as base vectors
- Each base vector has a radius
  - Similarity threshold
- Selects input vectors that fall within the radii of base vectors



OPList (Overlapped Partition List):

- OPList: list of base vectors that overlap with the given base vector
- Centroid OPList: cluster base vectors and assign a radius + OPList to each centroid

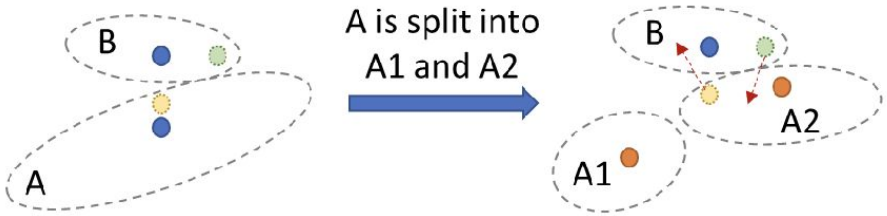


SPFresh [1/1]

Problem: Existing systems **avoid in-place updates**, accumulate changes in secondary index and periodically rebuild entire index globally.
This rebuild for DiskANN is expensive as it requires more than a day for a 1-billion vector index.

SPFresh:

- Uses LIRE protocol
- Split/merge partitions locally and reassign only the vectors at the boundaries



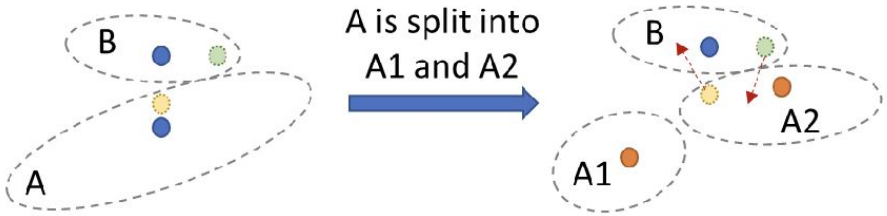
SPFresh [1/1]

Problem: Existing systems **avoid in-place updates**, accumulate changes in secondary index and periodically rebuild entire index globally.
This rebuild for DiskANN is expensive as it requires more than a day for a 1-billion vector index.

SPFresh:

- Uses LIRE protocol
- Split/merge partitions locally and reassign only the vectors at the boundaries

Works with cluster-based indexes



SPFresh [1/1]

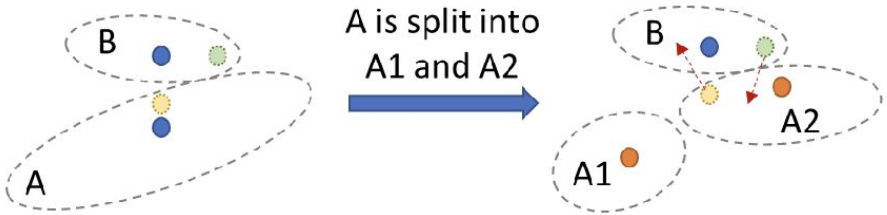
Problem: Existing systems **avoid in-place updates**, accumulate changes in secondary index and periodically rebuild entire index globally.
This rebuild for DiskANN is expensive as it requires more than a day for a 1-billion vector index.

SPFresh:

- Uses LIRE protocol
- Split/merge partitions locally and reassign only the vectors at the boundaries

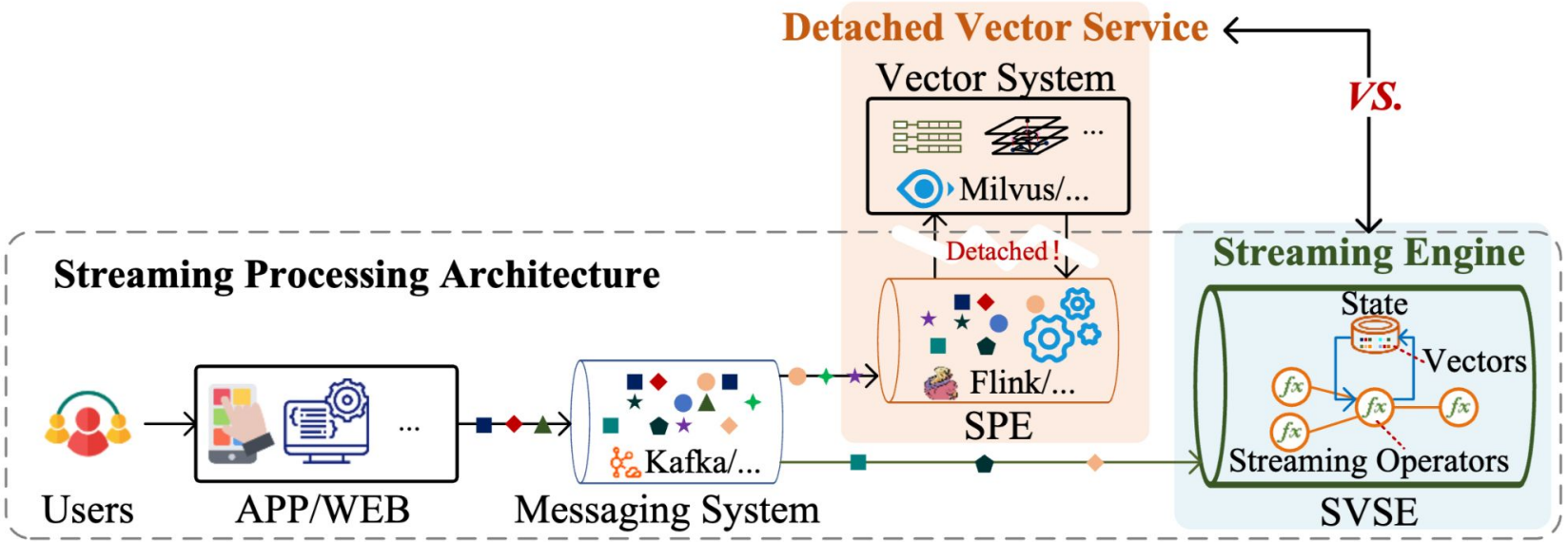
Works with cluster-based indexes

No global rebuilds



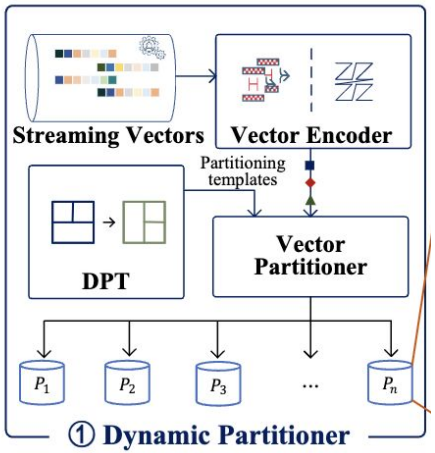
VStream [1/4]

Existing vector DBs operate as external services detached from streaming engines.



VStream [2/4]

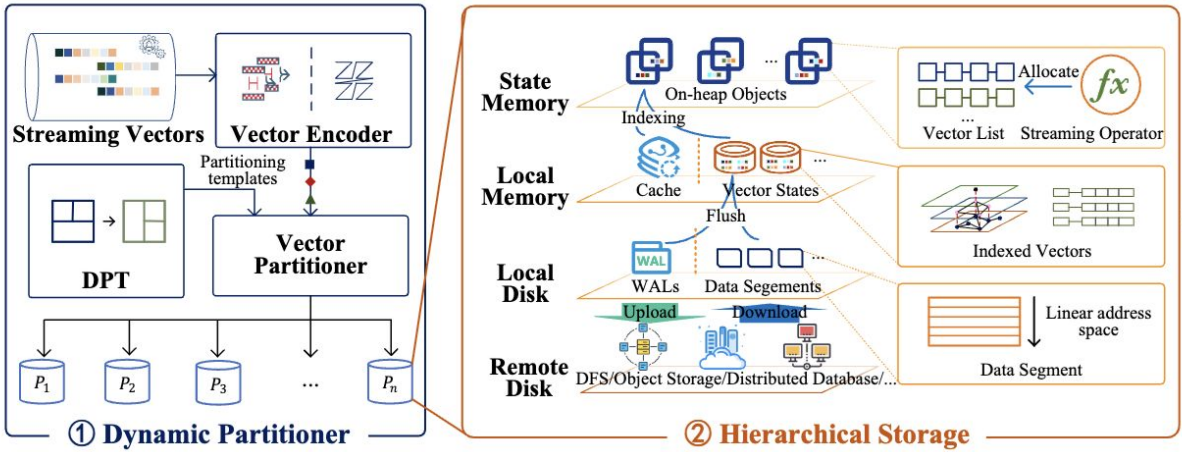
Dynamic partitioner: LSH + space-filling curves groups similar vectors in the same partition → queries skip irrelevant partitions



VStream [3/4]

Dynamic partitioner: LSH + space-filling curves groups similar vectors in the same partition → queries skip irrelevant partitions

Hierarchical storage: 4 levels, recent data stays at top

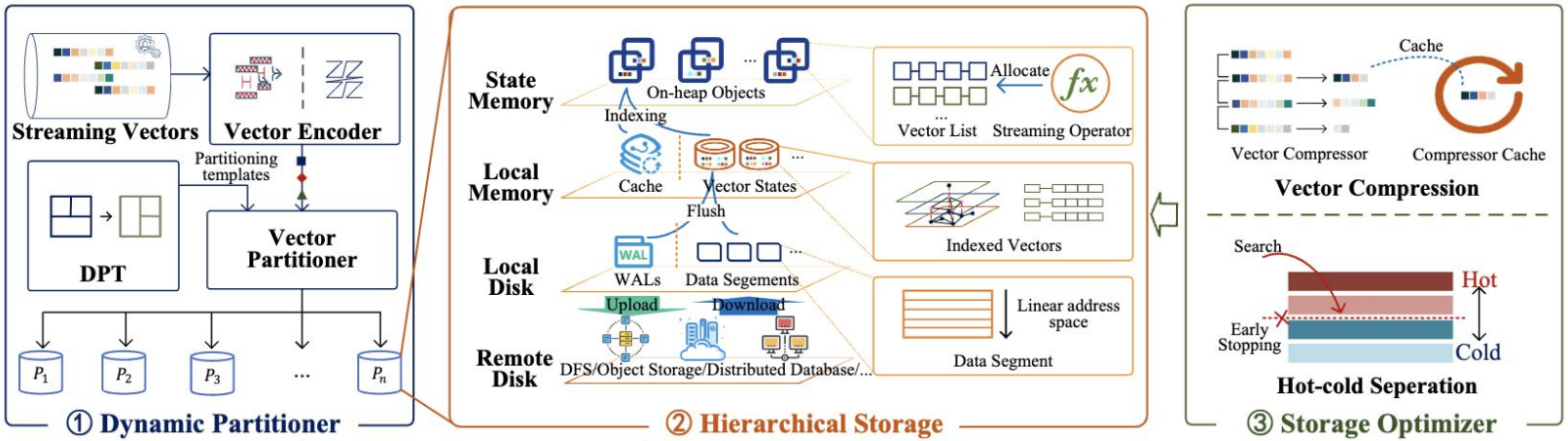


VStream [4/4]

Dynamic partitioner: LSH + space-filling curves groups similar vectors in the same partition → queries skip irrelevant partitions

Hierarchical storage: 4 levels, recent data stays at top

Hot-cold separation: scores data segments by access frequency, search hits, and freshness

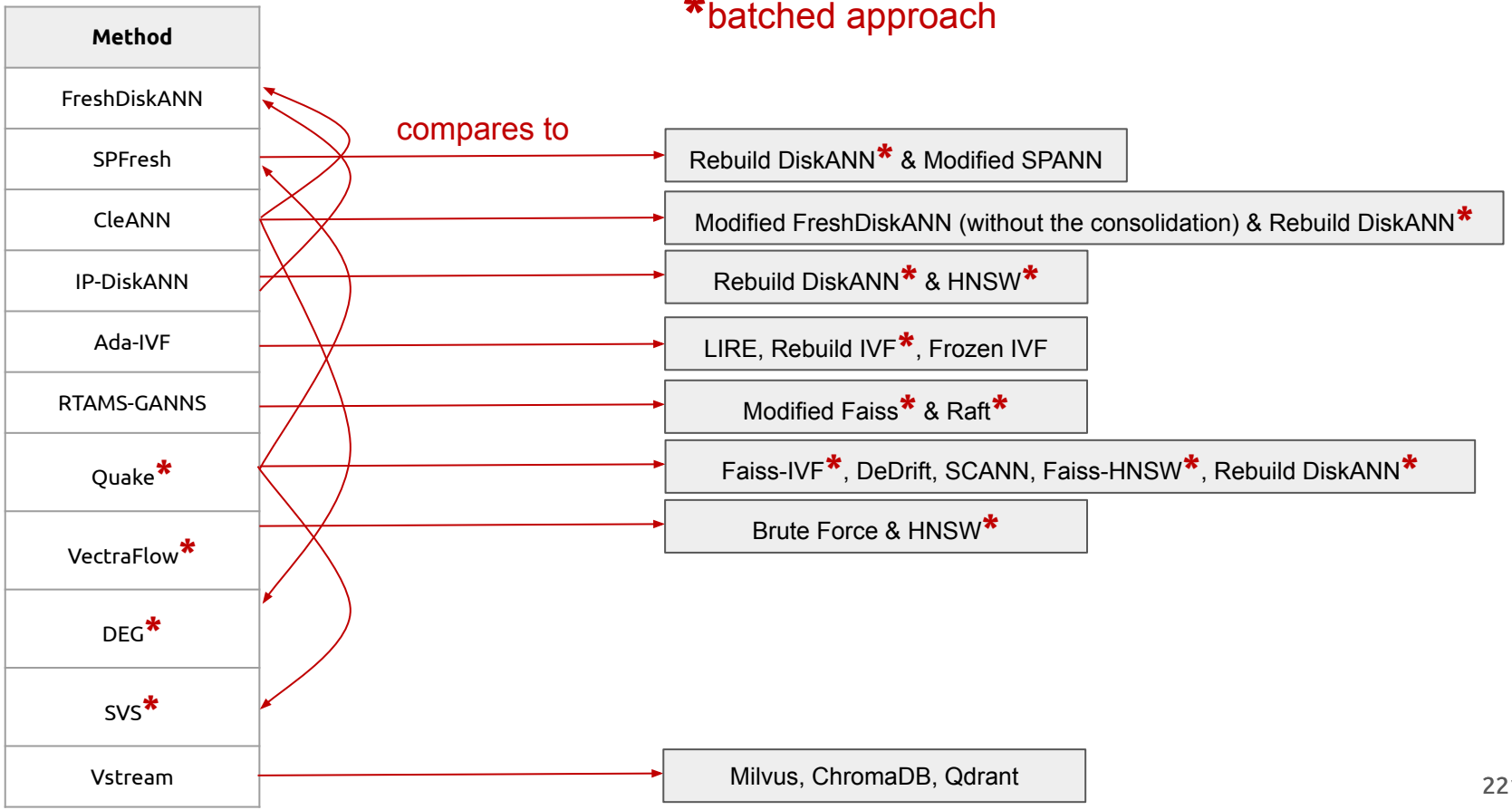


Summary of the approaches

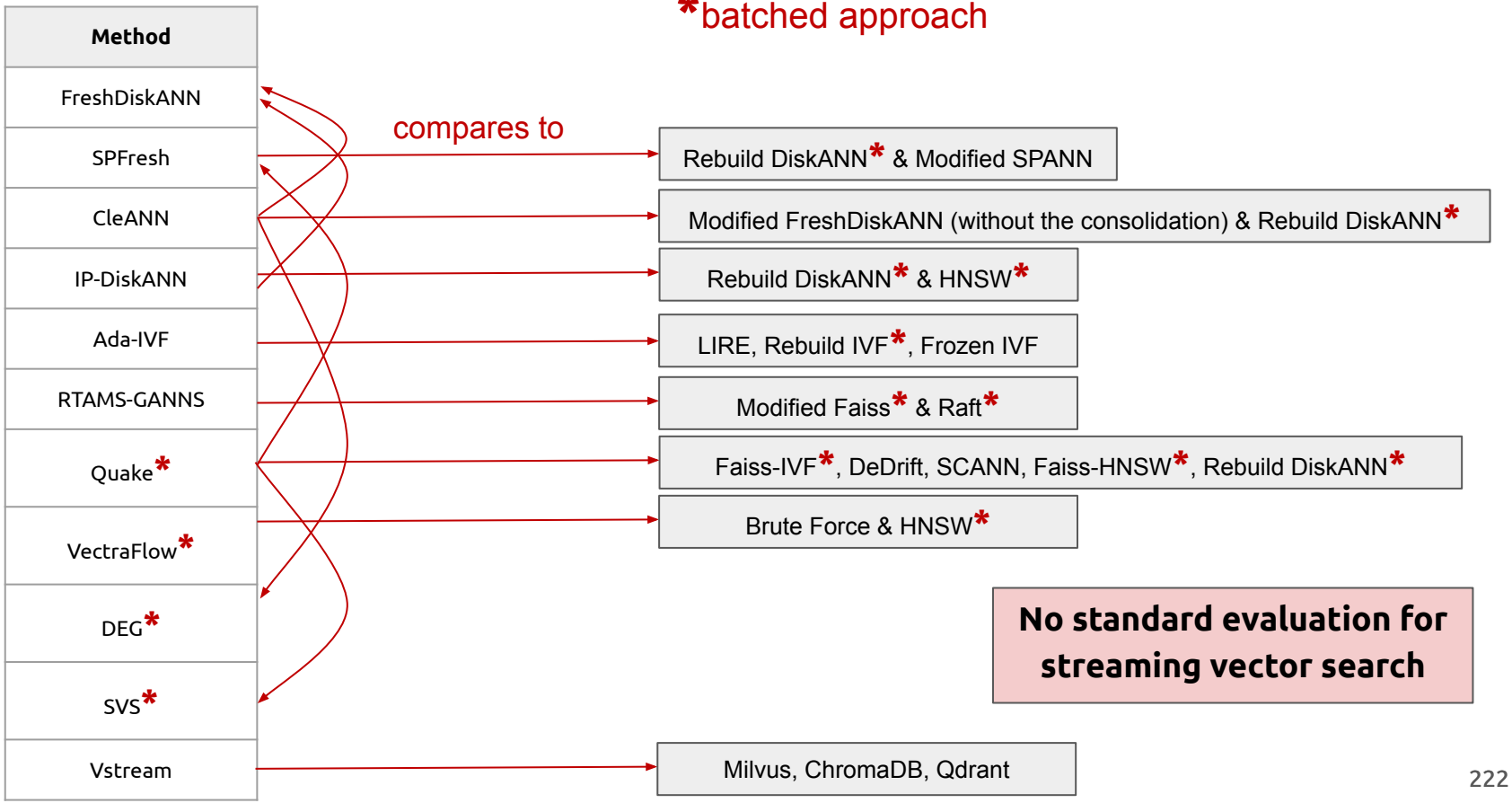
Method	Index	updates	hardware	execution
FreshDiskANN	k-NN Graph	incremental	CPU	parallel
SPFresh	k-NN Graph	incremental	CPU	parallel
CleANN	k-NN Graph	incremental	CPU	parallel
IP-DiskANN	k-NN Graph	incremental	CPU	parallel
Ada-IVF	IVF	incremental	CPU	parallel
RTAMS-GANNS	IVF	incremental	GPU	parallel
Quake	IVF	batched	CPU	parallel
VectraFlow	OPList	batched	CPU	parallel
DEG	k-NN Graph	batched	CPU	parallel
SVS	k-NN Graph	batched	CPU	parallel
VStream	k-NN Graph / IVF	incremental	CPU	parallel & distributed

Open Problems

Evaluation Gap



Evaluation Gap



No standard evaluation for streaming vector search

What Would a Streaming ANNS Benchmark Should include?

Workload Mix:

- Configurable **insert/delete ratio**
- **Skewed vs uniform** access patterns

Freshness Metric:

- **Time-to-visibility**: how quickly a newly inserted vector becomes searchable
- **Staleness**: what fraction of results are from deleted vectors

Scale:

- **Million to billion** vectors
- **real-world vs synthetic** datasets

What's missing today:

- **No standard benchmark** all aforementioned
- Each paper defines its own workload, making comparisons **very hard**

Simhadri, Harsha Vardhan, et al. Results of the Big ANN: NeurIPS'23 competition.

Future directions

- Comparative evaluation between streaming methods
- Standardize datasets, streaming workloads, metrics
- Maintaining index quality under updates
- Handling distribution drift
- Hybrid update strategies

Early Termination

Many thanks to Conglong Li et al. (LAET), Sonia Horchidan et al. (ConANN), Chao Zhang et al. (Ada-ef), who provided supporting material

Problems with current vector search applications

- All queries are processed with **the same search hyperparameters**

Problems with current vector search applications

- All queries are processed with **the same search hyperparameters**

- Hyperparameter setting is challenging
  - **Recall vs speed tradeoff → extensive tuning**
  - **Mapping** hyperparameters to average recall

Problems with current vector search applications

- All queries are processed with **the same search hyperparameters**

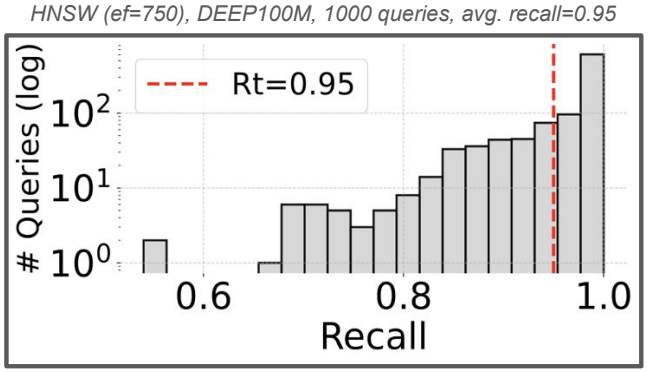
- Hyperparameter setting is challenging
  - **Recall vs speed tradeoff → extensive tuning**
  - **Mapping** hyperparameters to average recall

- Suboptimal results for **individual** queries

Problems with current vector search applications

- All queries are processed with **the same search hyperparameters**

- Hyperparameter setting is challenging
  - **Recall vs speed tradeoff → extensive tuning**
  - **Mapping** hyperparameters to average recall

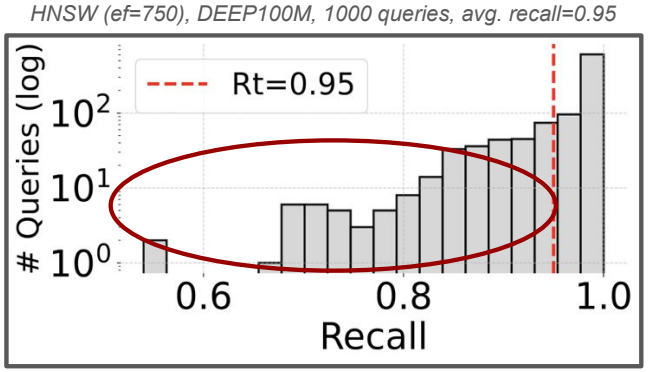


- Suboptimal results for **individual** queries

Problems with current vector search applications

- All queries are processed with **the same search hyperparameters**

- Hyperparameter setting is challenging
  - **Recall vs speed tradeoff → extensive tuning**
  - **Mapping** hyperparameters to average recall

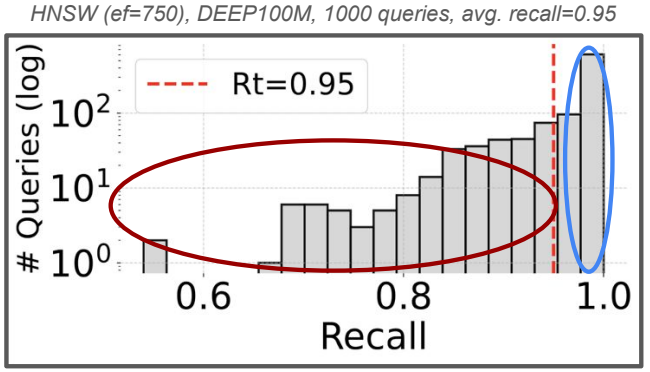


- Suboptimal results for **individual** queries
  - hard queries might **undershoot** (low recall) → inaccurate results!

Problems with current vector search applications

- All queries are processed with **the same search hyperparameters**

- Hyperparameter setting is challenging
  - **Recall vs speed tradeoff → extensive tuning**
  - **Mapping** hyperparameters to average recall



- Suboptimal results for **individual** queries
  - hard queries might **undershoot** (low recall) → inaccurate results!
  - easy queries might **overshoot** (slow processing) → wasted effort, monetary costs!

Vector search with declarative recall

- **Adaptive search**
  - **Adaptively** deliver the defined target recall for queries individually
  - **Higher** effort for **hard** queries, **lower** effort for **easy** queries

Transition to declarative recall interface

ANN(q, k, parameters)

Vector search with declarative recall

- **Adaptive search**
  - **Adaptively** deliver the defined target recall for queries individually
  - **Higher** effort for **hard** queries, **lower** effort for **easy** queries

Transition to declarative recall interface

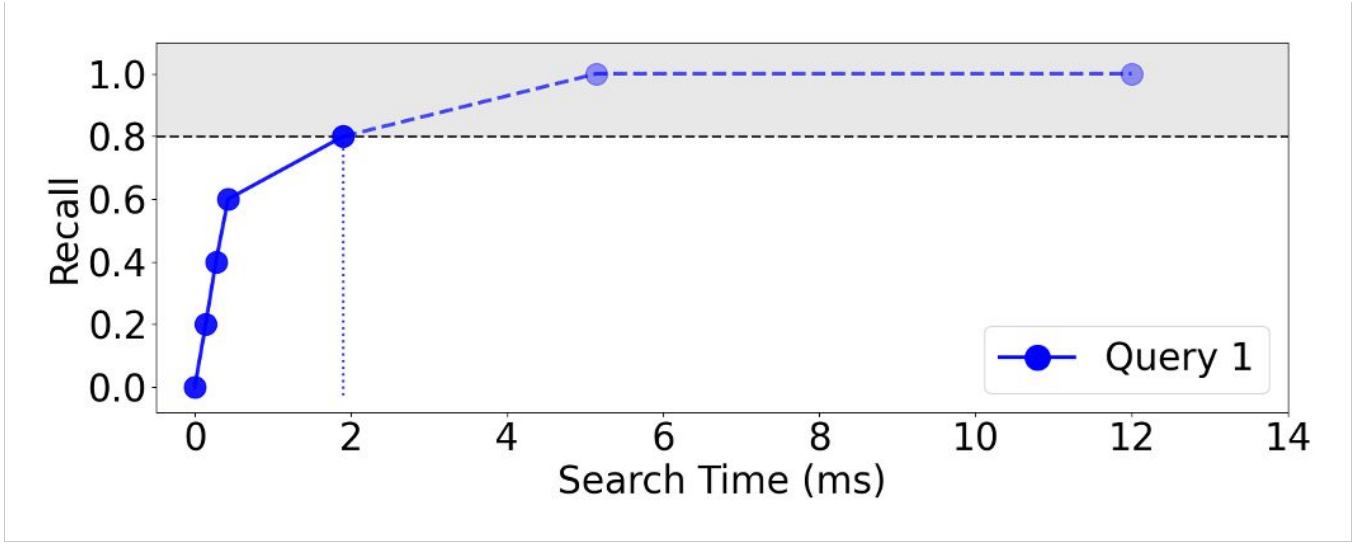
ANN(q, k, parameters) → **ANN(q, k, R_{target})**

Declarative recall through early termination

Search behaviors are very different between different queries!

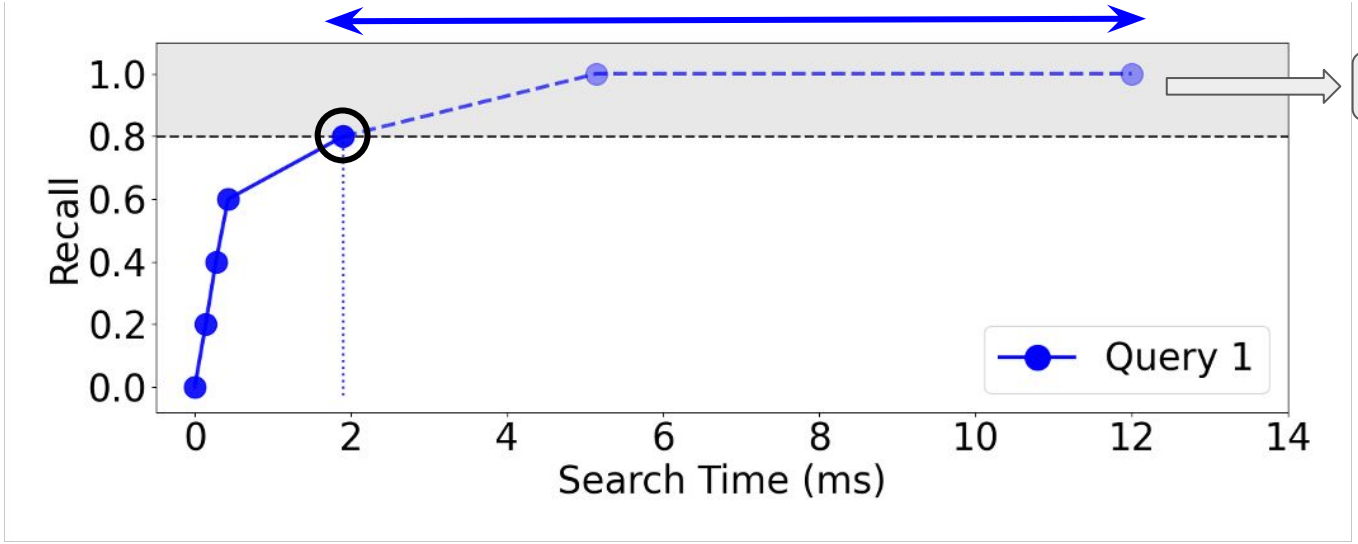
Declarative recall through early termination

Search behaviors are very different between different queries!



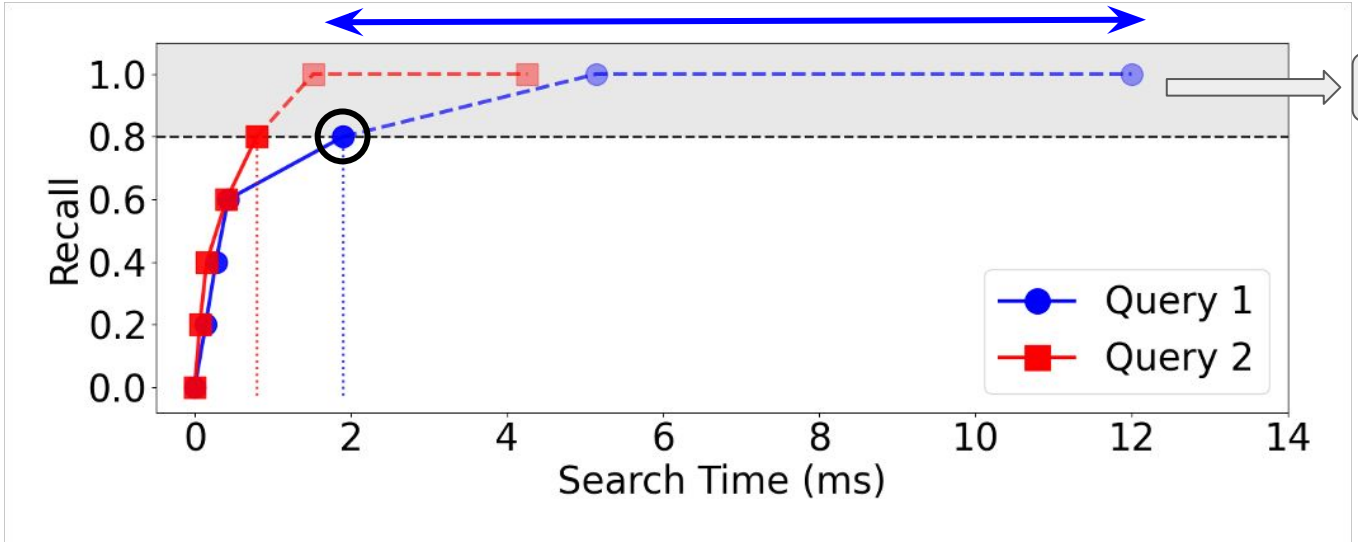
Declarative recall through early termination

Search behaviors are very different between different queries!



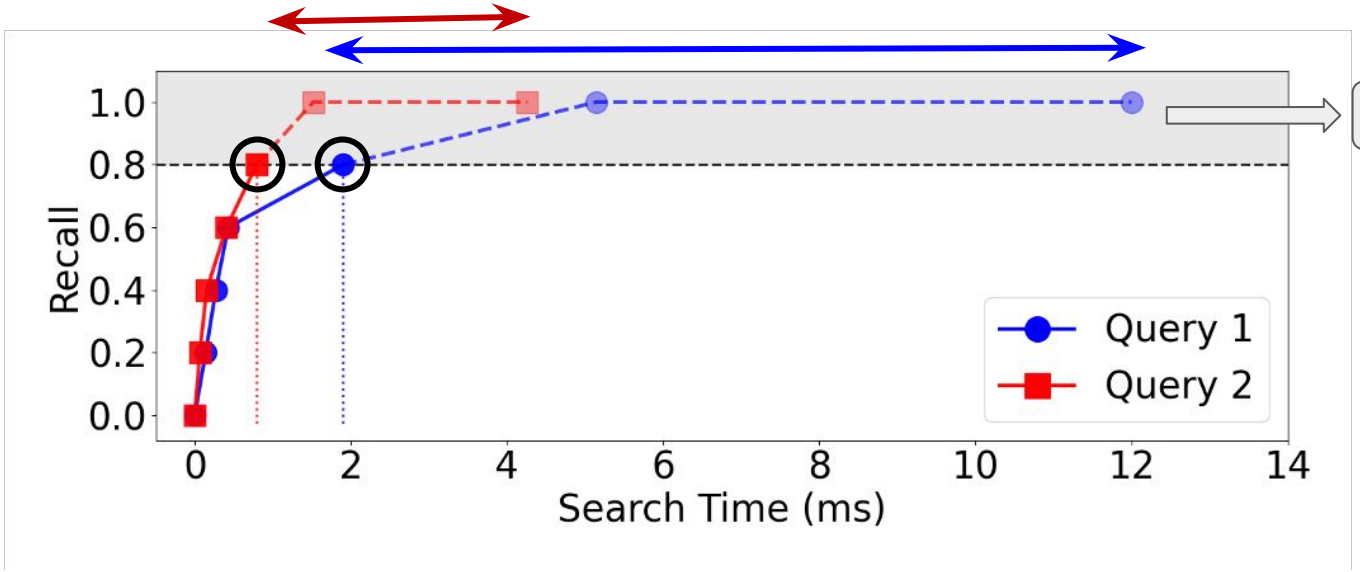
Declarative recall through early termination

Search behaviors are very different between different queries!



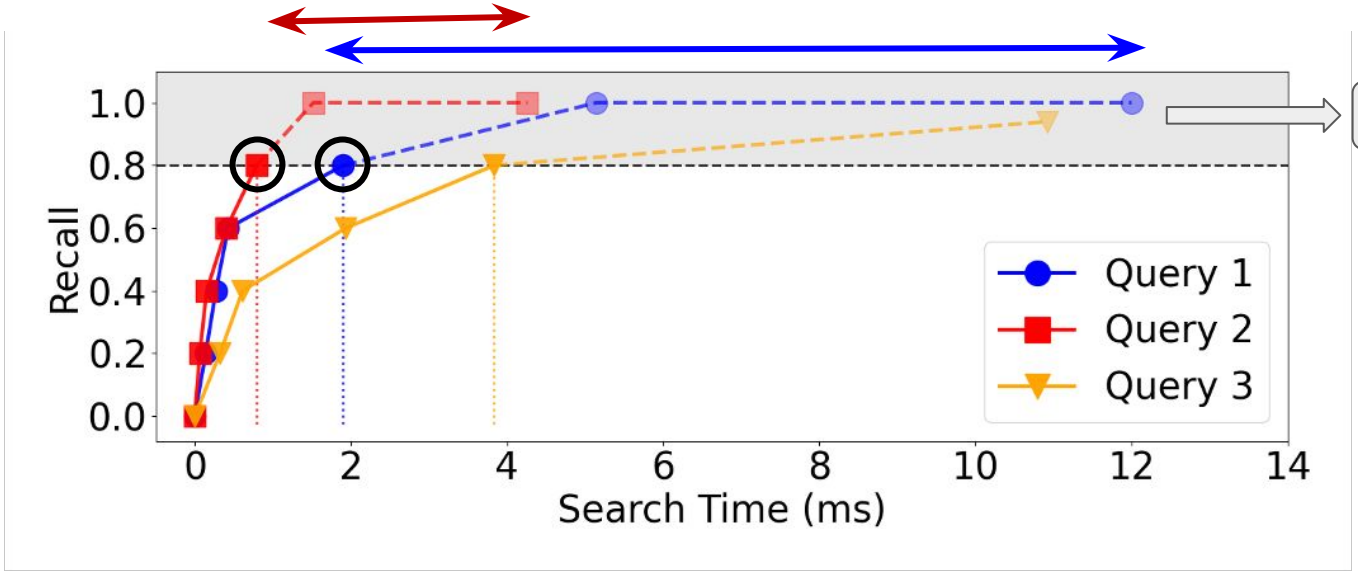
Declarative recall through early termination

Search behaviors are very different between different queries!



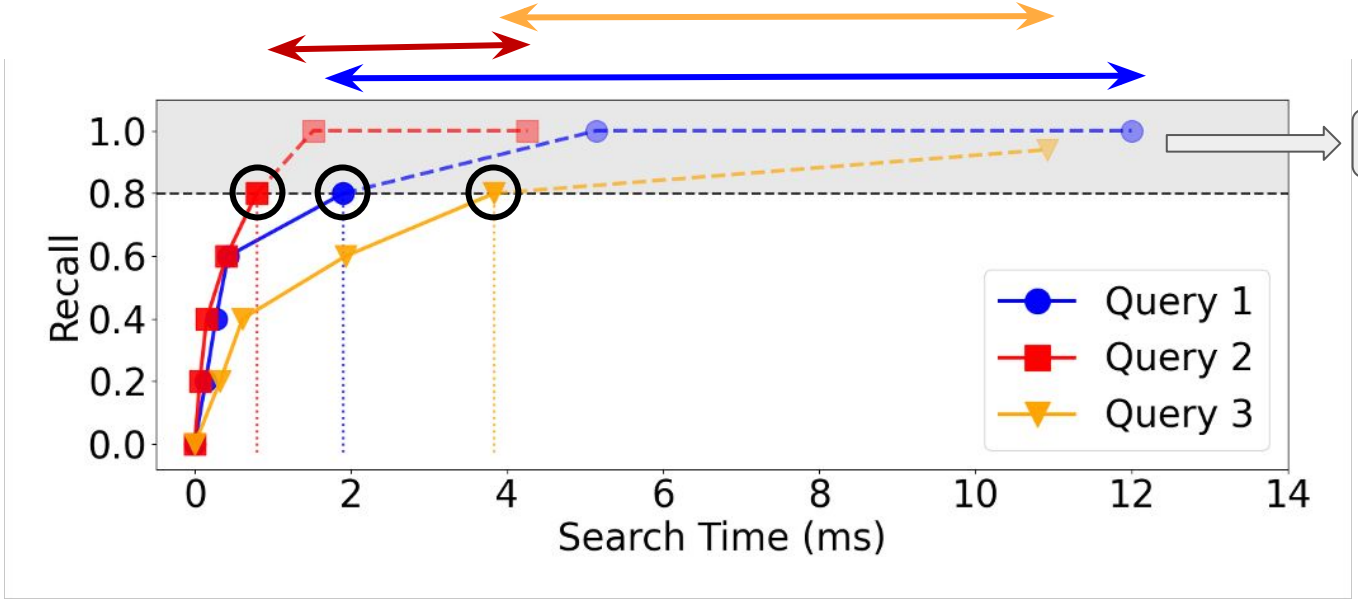
Declarative recall through early termination

Search behaviors are very different between different queries!



Declarative recall through early termination

Search behaviors are very different between different queries!



Different queries, different recalls, different points in time → adaptive search!

SPANN (IVF)

- Adaptively terminate when the current partition is too far away from the query
  - Use distance of query to first centroid for reference

SPANN (IVF)

- Adaptively terminate when the current partition is too far away from the query
  - Use distance of query to first centroid for reference

- Early terminate when

$$\text{dist}_{q \to Ci} > (1 + \text{error})\ \text{dist}_{q \to C1}$$

SPANN (IVF)

- Adaptively terminate when the current partition is too far away from the query
  - Use distance of query to first centroid for reference

- Early terminate when

$$dist_{q \to Ci} > (1 + error) \, dist_{q \to C1}$$

Tune for each recall target!

SPANN (IVF)

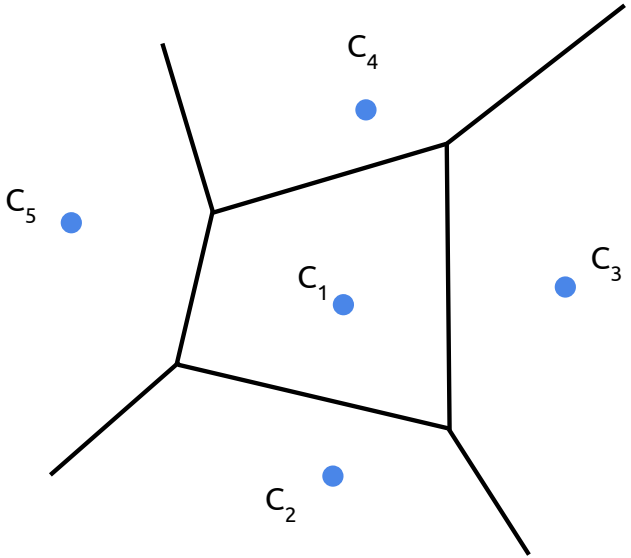
- Adaptively terminate when the current partition is too far away from the query
  - Use distance of query to first centroid for reference

- Early terminate when

$$dist_{q \to Ci} > (1 + error) \, dist_{q \to C1}$$

Tune for each recall target!

IVF partitions $P_1 - P_5$ with centroids $C_1 - C_5$



SPANN (IVF)

- Adaptively terminate when the current partition is too far away from the query
  - Use distance of query to first centroid for reference

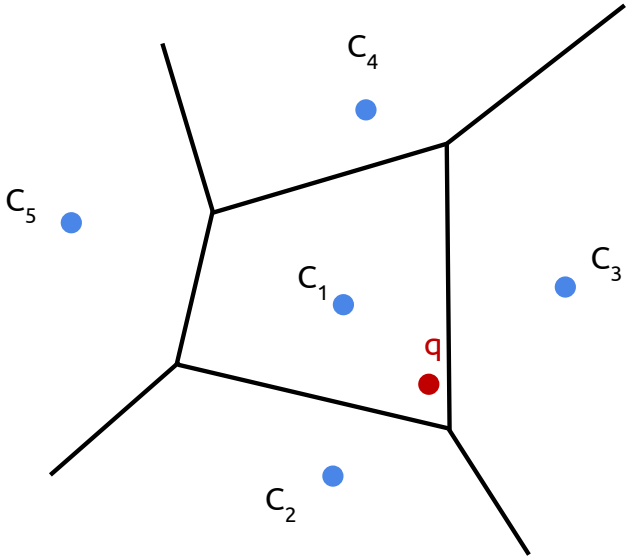
- Early terminate when

$$dist_{q \to Ci} > (1 + error) \, dist_{q \to C1}$$

Tune for each recall target!

ANN(q, k, 0.80) → error=0.9

IVF partitions $P_1 - P_5$ with centroids $C_1 - C_5$



SPANN (IVF)

- Adaptively terminate when the current partition is too far away from the query
  - Use distance of query to first centroid for reference

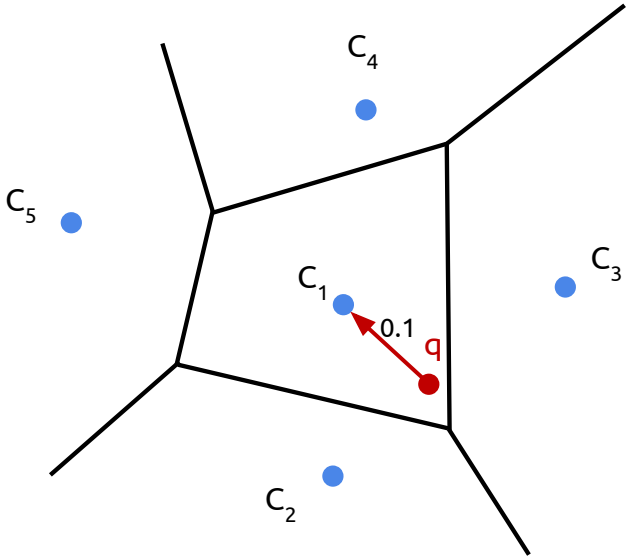
- Early terminate when

$$dist_{q \to Ci} > (1 + error) \, dist_{q \to C1}$$

Tune for each recall target!

ANN(q, k, 0.80) → error=0.9

IVF partitions $P_1 - P_5$ with centroids $C_1 - C_5$



SPANN (IVF)

- Adaptively terminate when the current partition is too far away from the query
  - Use distance of query to first centroid for reference

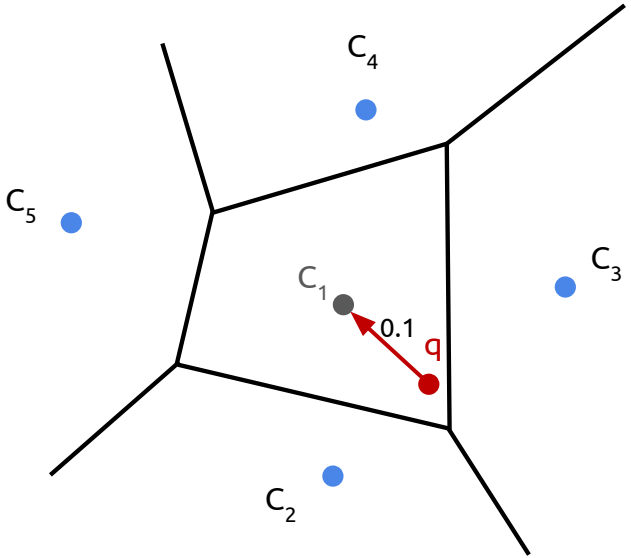
- Early terminate when

$$dist_{q \to Ci} > (1 + error) \, dist_{q \to C1}$$

Tune for each recall target!

ANN(q, k, 0.80) → error=0.9

IVF partitions $P_1 - P_5$ with centroids $C_1 - C_5$



SPANN (IVF)

- Adaptively terminate when the current partition is too far away from the query
  - Use distance of query to first centroid for reference

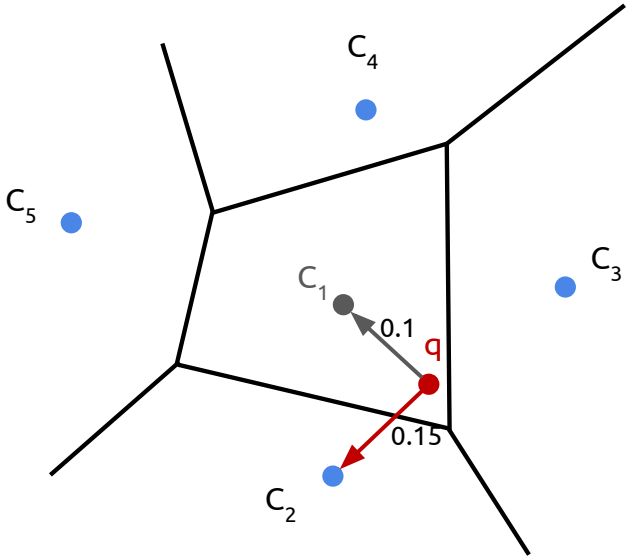
- Early terminate when

$$dist_{q \to Ci} > (1 + error)\, dist_{q \to C1}$$

Tune for each recall target!

ANN(q, k, 0.80) → error=0.9

IVF partitions $P_1 - P_5$ with centroids $C_1 - C_5$



SPANN (IVF)

- Adaptively terminate when the current partition is too far away from the query
  - Use distance of query to first centroid for reference

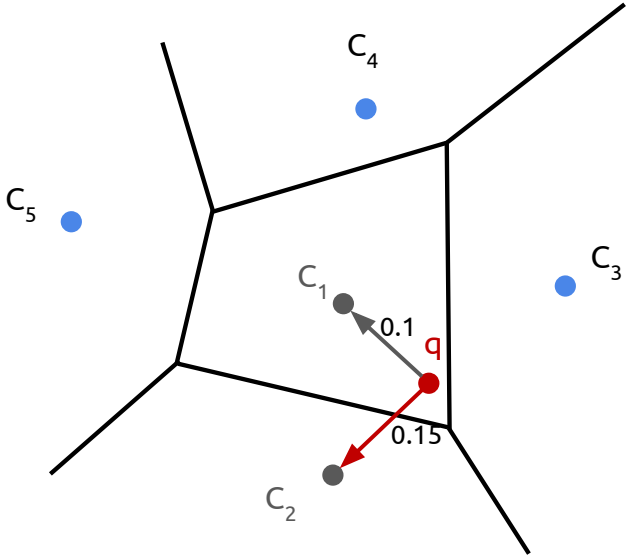
- Early terminate when

$$\text{dist}_{q \to Ci} > (1 + \text{error})\ \text{dist}_{q \to C1}$$

Tune for each recall target!

ANN(q, k, 0.80) → error=0.9

IVF partitions $P_1 - P_5$ with centroids $C_1 - C_5$



SPANN (IVF)

- Adaptively terminate when the current partition is too far away from the query
  - Use distance of query to first centroid for reference

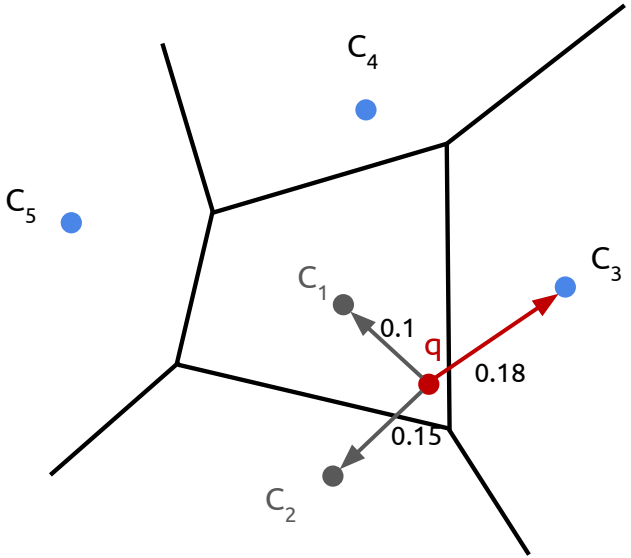
- Early terminate when

$$\text{dist}_{q \to Ci} > (1 + \text{error})\ \text{dist}_{q \to C1}$$

Tune for each recall target!

ANN(q, k, 0.80) → error=0.9

IVF partitions $P_1 - P_5$ with centroids $C_1 - C_5$



SPANN (IVF)

- Adaptively terminate when the current partition is too far away from the query
  - Use distance of query to first centroid for reference

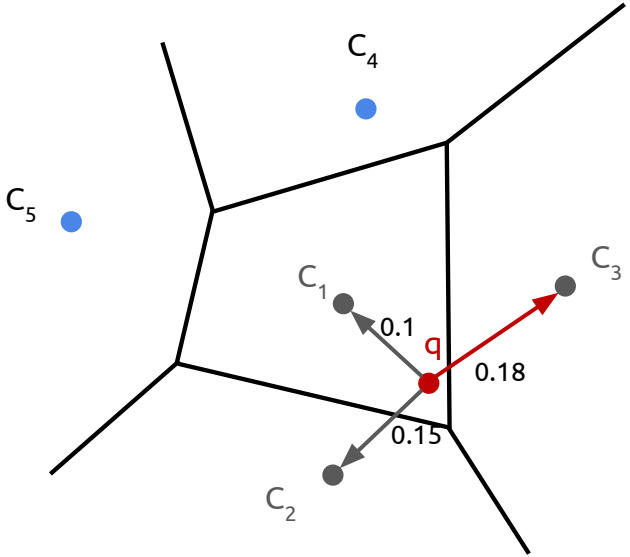
- Early terminate when

$$\text{dist}_{q \to Ci} > (1 + \text{error})\ \text{dist}_{q \to C1}$$

Tune for each recall target!

ANN(q, k, 0.80) → error=0.9

IVF partitions $P_1 - P_5$ with centroids $C_1 - C_5$



SPANN (IVF)

- Adaptively terminate when the current partition is too far away from the query
  - Use distance of query to first centroid for reference

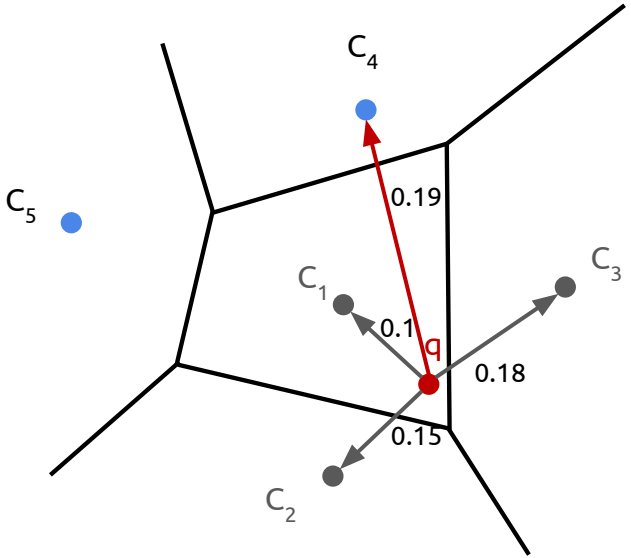
- Early terminate when

$$dist_{q \to Ci} > (1 + error)\, dist_{q \to C1}$$

Tune for each recall target!

ANN(q, k, 0.80) → error=0.9

IVF partitions $P_1 - P_5$ with centroids $C_1 - C_5$



SPANN (IVF)

- Adaptively terminate when the current partition is too far away from the query
  - Use distance of query to first centroid for reference

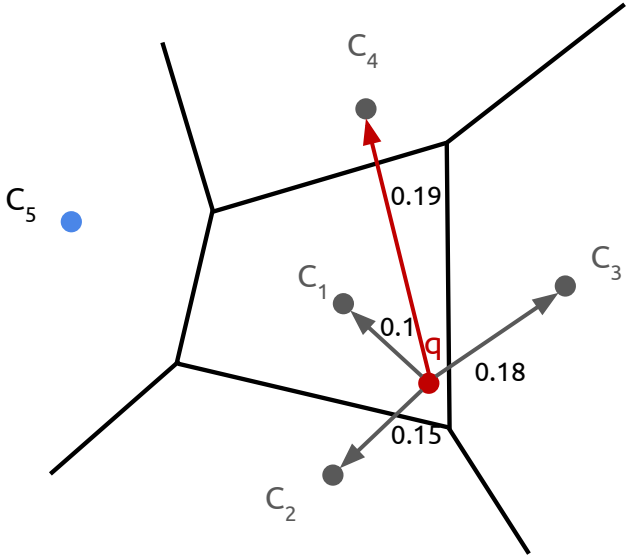
- Early terminate when

$$dist_{q \to Ci} > (1 + error)\, dist_{q \to C1}$$

Tune for each recall target!

ANN(q, k, 0.80) → error=0.9

IVF partitions $P_1 - P_5$ with centroids $C_1 - C_5$



SPANN (IVF)

- Adaptively terminate when the current partition is too far away from the query
  - Use distance of query to first centroid for reference

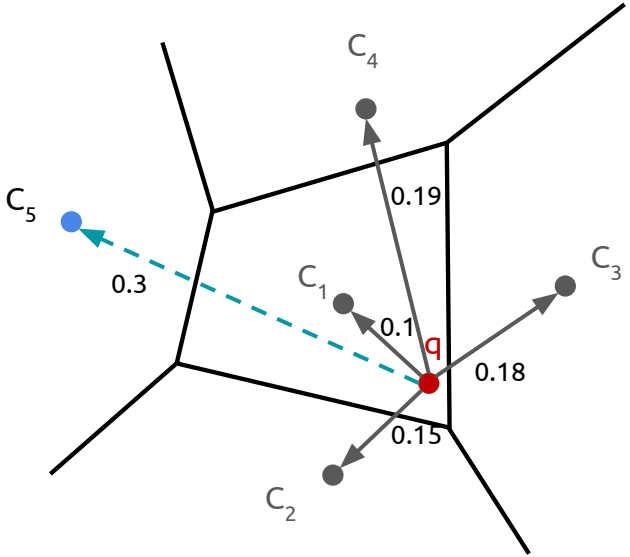
- Early terminate when

$$dist_{q \to Ci} > (1 + error)\, dist_{q \to C1}$$

Tune for each recall target!

ANN(q, k, 0.80) → error=0.9

IVF partitions $P_1 - P_5$ with centroids $C_1 - C_5$



SPANN (IVF)

- Adaptively terminate when the current partition is too far away from the query
  - Use distance of query to first centroid for reference

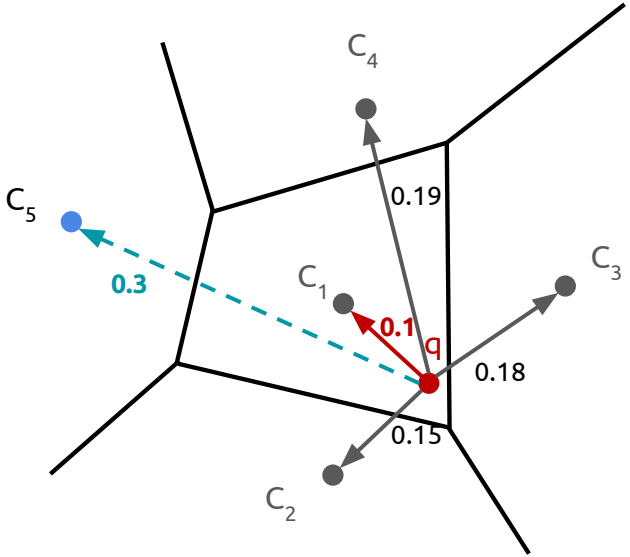
- Early terminate when

$$\text{dist}_{q \to Ci} > (1 + \text{error})\ \text{dist}_{q \to C1}$$

Tune for each recall target!

ANN(q, k, 0.80) → error=0.9

IVF partitions $P_1 - P_5$ with centroids $C_1 - C_5$



SPANN (IVF)

- Adaptively terminate when the current partition is too far away from the query
  - Use distance of query to first centroid for reference

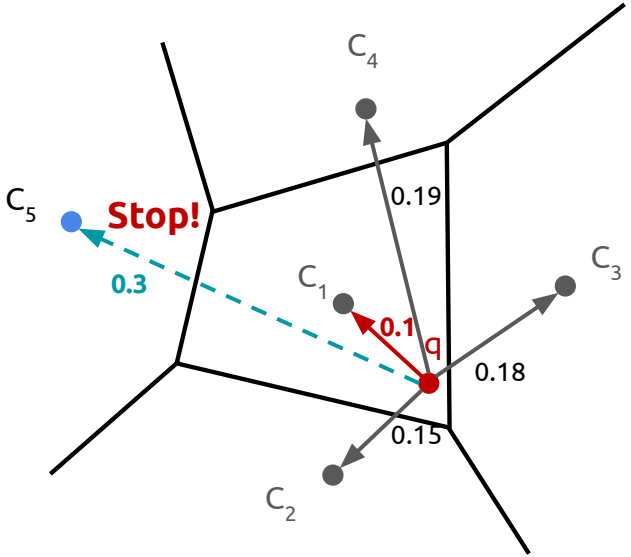
- Early terminate when

$$\text{dist}_{q \to Ci} > (1 + \text{error})\ \text{dist}_{q \to C1}$$

Tune for each recall target!

ANN(q, k, 0.80) → error=0.9

IVF partitions $P_1 - P_5$ with centroids $C_1 - C_5$



SPANN (IVF)

- Adaptively terminate when the current partition is too far away from the query
  - Use distance of query to first centroid for reference

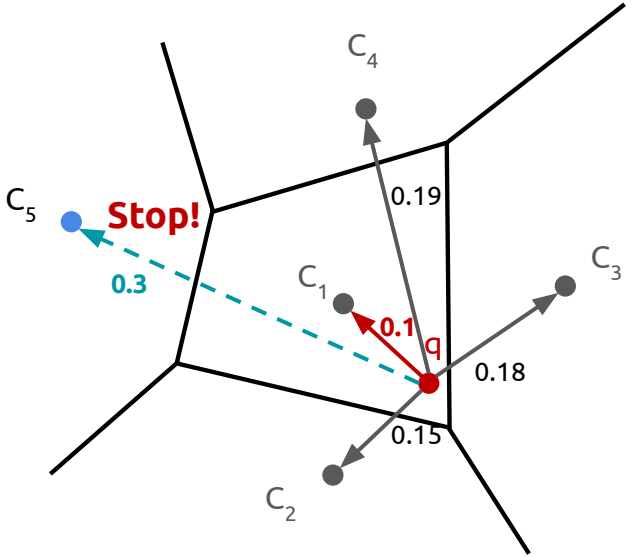
- Early terminate when

$$\text{dist}_{q \to Ci} > (1 + \text{error})\ \text{dist}_{q \to C1}$$

Tune for each recall target!

ANN(q, k, 0.80) → error=0.9

IVF partitions $P_1 - P_5$ with centroids $C_1 - C_5$



Rule based

Extension to kNN-graphs
Optimized error tuning

Quake (IVF)

- Estimate the recall each IVF partition contributes
  - **Volume overlap** between the **hypersphere of query and k-th nn (ρ_k)** and each partition **P_i**

$$\textbf{Recall}_{P_i} = \textbf{Vol}(B(q, \rho_k) \cap P_i) \,) \,/\, \textbf{Vol}(B(q, \rho_k))$$

Quake (IVF)

- Estimate the recall each IVF partition contributes
  - **Volume overlap** between the **hypersphere of query and k-th nn (ρ_k)** and each partition **P_i**

$$\textbf{Recall}_{Pi} = \textbf{Vol}(B(q, \rho_k) \cap P_i) \,) \,/\, \textbf{Vol}(B(q, \rho_k))$$

- Visit each partition under target recall is reached
  - After each partition, each Recall_{Pi} is re-evaluated based on the updated k-th nn

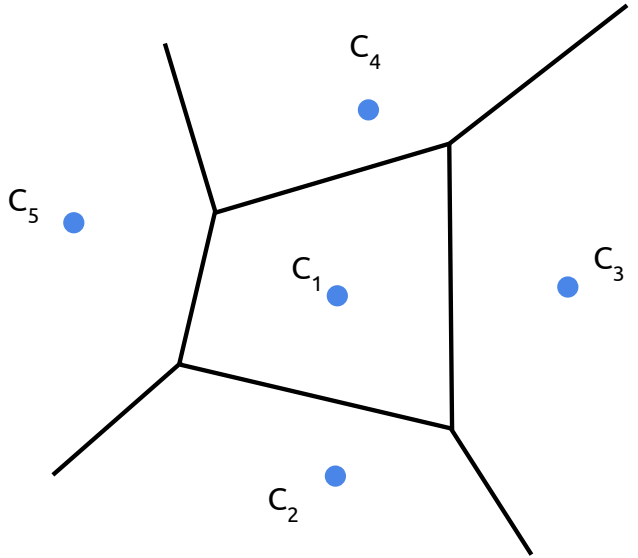
Quake (IVF)

- Estimate the recall each IVF partition contributes
  - **Volume overlap** between the **hypersphere of query and k-th nn (ρ_k)** and each partition **P_i**

$$\textbf{Recall}_{Pi} = \textbf{Vol}(B(q, \rho_k) \cap P_i)) \,/\, \textbf{Vol}(B(q, \rho_k))$$

- Visit each partition under target recall is reached
  - After each partition, each Recall_{Pi} is re-evaluated based on the updated k-th nn

IVF partitions $P_1 - P_5$ with centroids $C_1 - C_5$

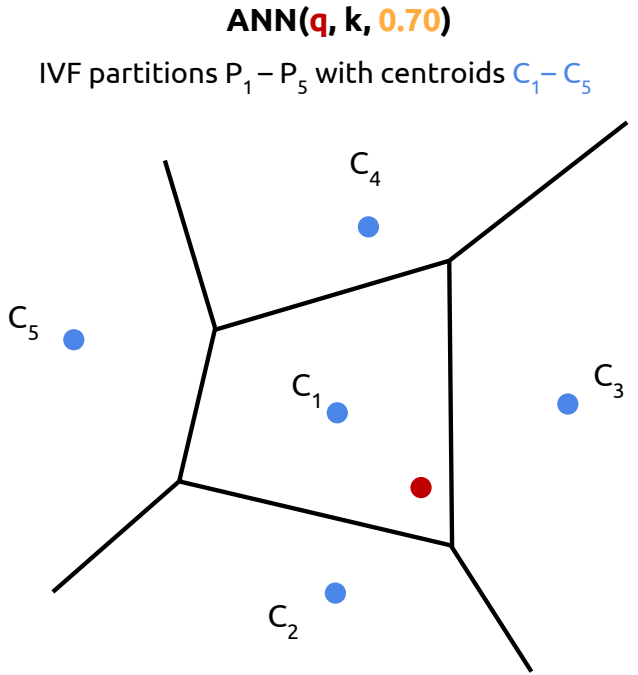


Quake (IVF)

- Estimate the recall each IVF partition contributes
  - **Volume overlap** between the **hypersphere of query and k-th nn (ρ_k)** and each partition **P_i**

$$\textbf{Recall}_{Pi} = \textbf{Vol}(B(q, \rho_k) \cap P_i)) \,/\, \textbf{Vol}(B(q, \rho_k))$$

- Visit each partition under target recall is reached
  - After each partition, each Recall_{Pi} is re-evaluated based on the updated k-th nn



Quake (IVF)

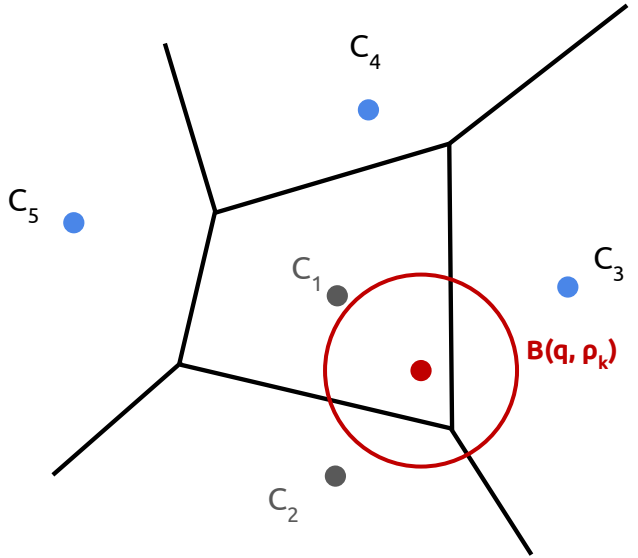
- Estimate the recall each IVF partition contributes
  - **Volume overlap** between the **hypersphere of query and k-th nn (ρ_k)** and each partition **P_i**

$$\textbf{Recall}_{Pi} = \textbf{Vol}(B(q, \rho_k) \cap P_i)) \,/\, \textbf{Vol}(B(q, \rho_k))$$

- Visit each partition under target recall is reached
  - After each partition, each Recall_{Pi} is re-evaluated based on the updated k-th nn

ANN(q, k, 0.70)

IVF partitions $P_1 - P_5$ with centroids $C_1 - C_5$



After searching 2 partitions:

Quake (IVF)

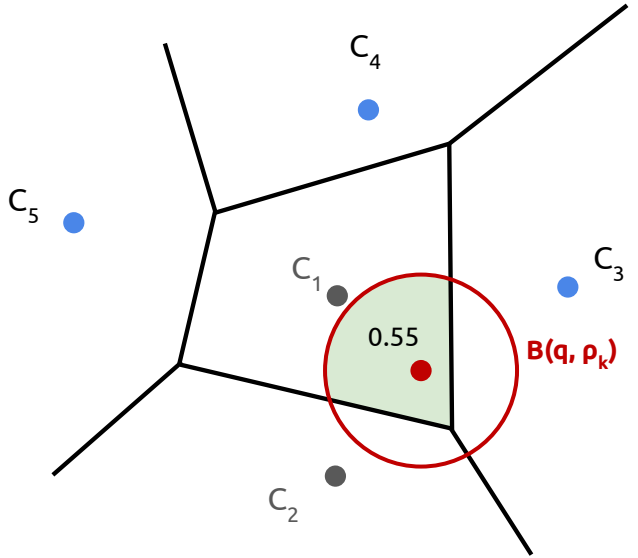
- Estimate the recall each IVF partition contributes
  - **Volume overlap** between the **hypersphere of query and k-th nn (ρ_k)** and each partition **P_i**

$$\text{Recall}_{P_i} = \text{Vol}(B(q, \rho_k) \cap P_i)) / \text{Vol}(B(q, \rho_k))$$

- Visit each partition under target recall is reached
  - After each partition, each Recall_{P_i} is re-evaluated based on the updated k-th nn

ANN(q, k, 0.70)

IVF partitions P_1 – P_5 with centroids C_1 – C_5



After searching 2 partitions:

Quake (IVF)

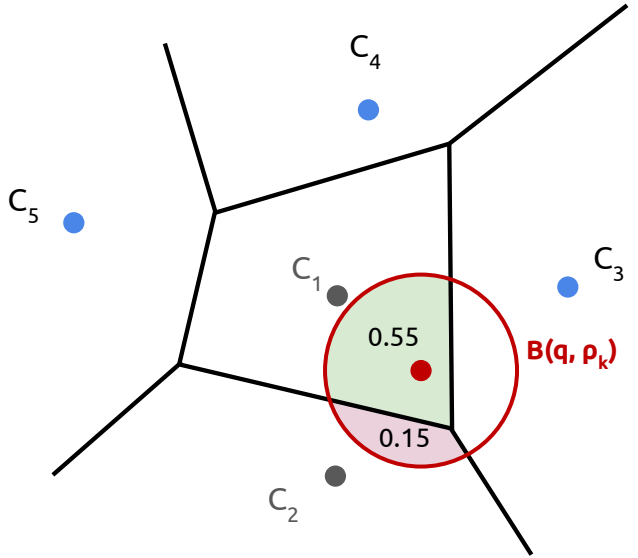
- Estimate the recall each IVF partition contributes
  - **Volume overlap** between the **hypersphere of query and k-th nn (ρ_k)** and each partition **P_i**

$$\text{Recall}_{Pi} = \text{Vol}(B(q, \rho_k) \cap P_i)) / \text{Vol}(B(q, \rho_k))$$

- Visit each partition under target recall is reached
  - After each partition, each Recall_{Pi} is re-evaluated based on the updated k-th nn

ANN(q, k, 0.70)

IVF partitions P_1 – P_5 with centroids C_1 – C_5



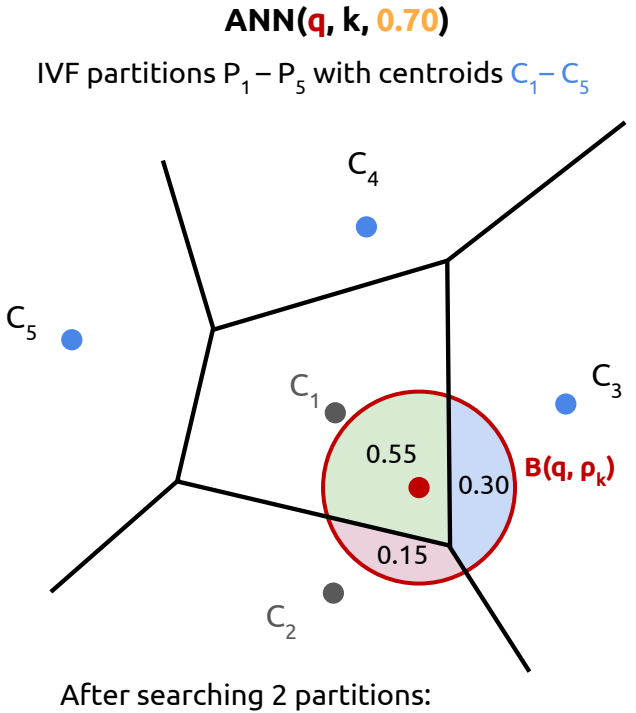
After searching 2 partitions:

Quake (IVF)

- Estimate the recall each IVF partition contributes
  - **Volume overlap** between the **hypersphere of query and k-th nn (ρ_k)** and each partition **P_i**

$$\text{Recall}_{Pi} = \text{Vol}(B(q, \rho_k) \cap P_i)) / \text{Vol}(B(q, \rho_k))$$

- Visit each partition under target recall is reached
  - After each partition, each Recall_{Pi} is re-evaluated based on the updated k-th nn



Quake (IVF)

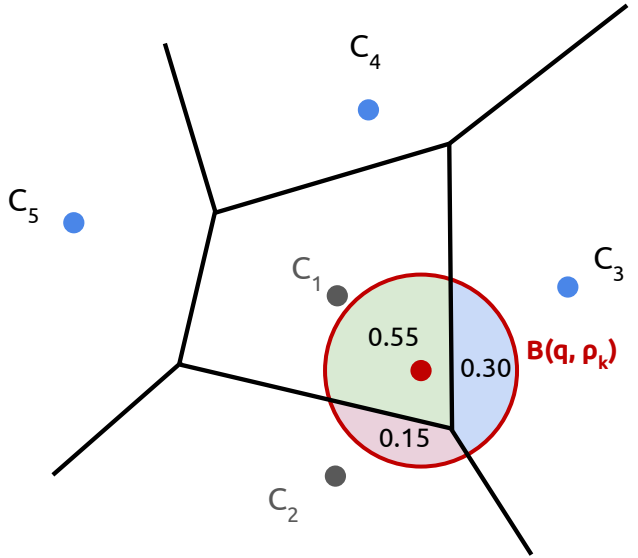
- Estimate the recall each IVF partition contributes
  - **Volume overlap** between the **hypersphere of query and k-th nn (ρ_k)** and each partition **P_i**

$$\textbf{Recall}_{Pi} = \textbf{Vol}(B(q, \rho_k) \cap P_i)) \,/\, \textbf{Vol}(B(q, \rho_k))$$

- Visit each partition under target recall is reached
  - After each partition, each Recall_{Pi} is re-evaluated based on the updated k-th nn

ANN(q, k, 0.70)

IVF partitions $P_1 - P_5$ with centroids $C_1 - C_5$



After searching 2 partitions:
Estimated recall = 0.55 + 0.15 = 0.70 → Terminate!

Quake (IVF)

- Estimate the recall each IVF partition contributes
  - **Volume overlap** between the **hypersphere of query and k-th nn (ρ_k)** and each partition **P_i**

$$\textbf{Recall}_{Pi} = \textbf{Vol}(B(q, \rho_k) \cap P_i)) \,/\, \textbf{Vol}(B(q, \rho_k))$$

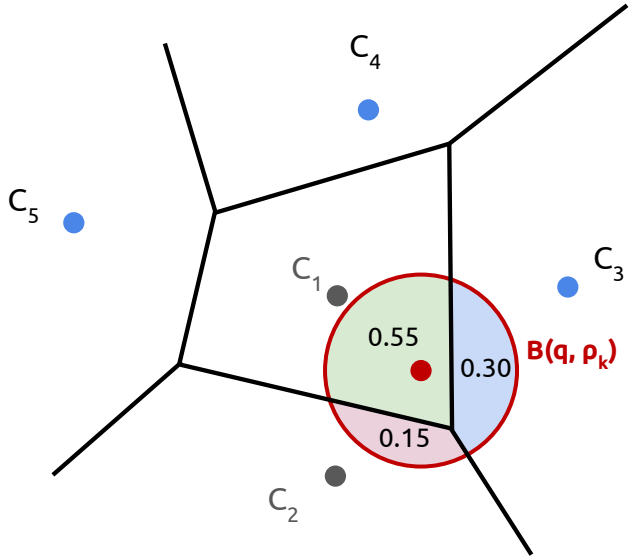
- Visit each partition under target recall is reached
  - After each partition, each Recall_{Pi} is re-evaluated based on the updated k-th nn

Independent of dataset
Lightweight setup

Extension to kNN-graphs

ANN(q, k, 0.70)

IVF partitions $P_1 – P_5$ with centroids $C_1 – C_5$



After searching 2 partitions:
Estimated recall = 0.55 + 0.15 = 0.70 → Terminate!

ConANN (IVF)

- Terminates the search using a global **conformal-calibrated threshold λ** that **guarantees** reaching target recall in **expectation**.

ConANN (IVF)

- Terminates the search using a global **conformal-calibrated threshold λ** that **guarantees** reaching target recall in **expectation**.
  - Terminate once k_{th} distance after processing a partition is $\mathbf{k_{th} \leq \lambda}$

ConANN (IVF)

- Terminates the search using a global **conformal-calibrated threshold λ** that **guarantees** reaching target recall in **expectation**.
  - Terminate once k_{th} distance after processing a partition is $\mathbf{k_{th} \leq \lambda}$

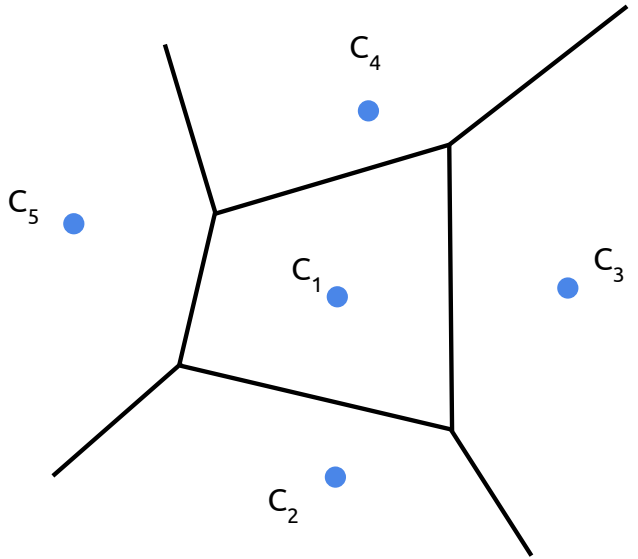
- Conformal Risk Control (CRC)
  - Find the minimum λ so that the expectation of recall across the queries of workload is $\mathbb{E}[R] \geq R_{target}$

In the original paper False Negative Rate is used as ***FNR = 1 - R_{target}***

ConANN (IVF)

- Terminates the search using a global **conformal-calibrated threshold λ** that **guarantees** reaching target recall in **expectation**.
  - Terminate once k_{th} distance after processing a partition is $\mathbf{k_{th} \leq \lambda}$
- Conformal Risk Control (CRC)
  - Find the minimum λ so that the expectation of recall across the queries of workload is $\mathbb{E}[R] \geq R_{target}$

IVF partitions $P_1 – P_5$ with centroids $C_1 – C_5$



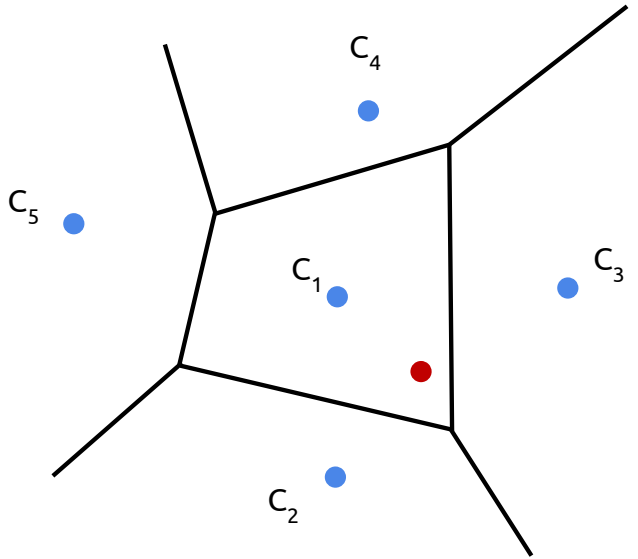
In the original paper False Negative Rate is used as ***FNR = 1 - R_{target}***

ConANN (IVF)

- Terminates the search using a global **conformal-calibrated threshold λ** that **guarantees** reaching target recall in **expectation**.
  - Terminate once k_{th} distance after processing a partition is $\mathbf{k_{th} \leq \lambda}$
- Conformal Risk Control (CRC)
  - Find the minimum λ so that the expectation of recall across the queries of workload is $\mathbb{E}[R] \geq R_{target}$

ANN(q, k, 0.80) → λ=0.1

IVF partitions P_1 – P_5 with centroids C_1 – C_5



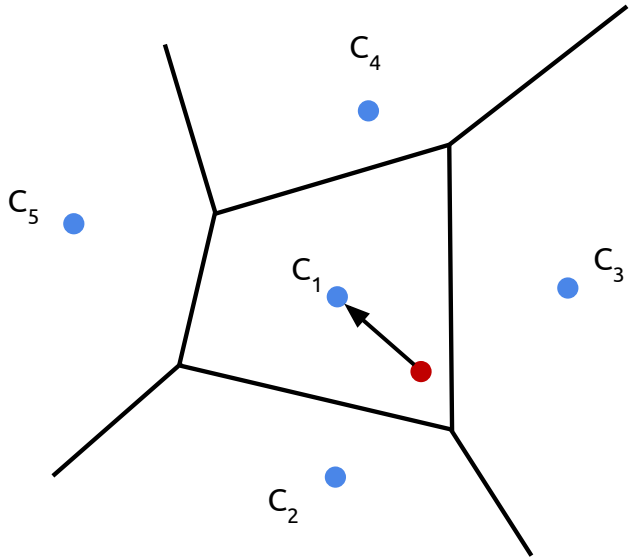
In the original paper False Negative Rate is used as ***FNR = 1 - R_{target}***

ConANN (IVF)

- Terminates the search using a global **conformal-calibrated threshold λ** that **guarantees** reaching target recall in **expectation**.
  - Terminate once k_{th} distance after processing a partition is $\mathbf{k_{th} \leq \lambda}$
- Conformal Risk Control (CRC)
  - Find the minimum λ so that the expectation of recall across the queries of workload is $\mathbb{E}[R] \geq R_{target}$

ANN(q, k, 0.80) → λ=0.1

IVF partitions P_1 – P_5 with centroids C_1 – C_5



In the original paper False Negative Rate is used as ***FNR = 1 - R_{target}***

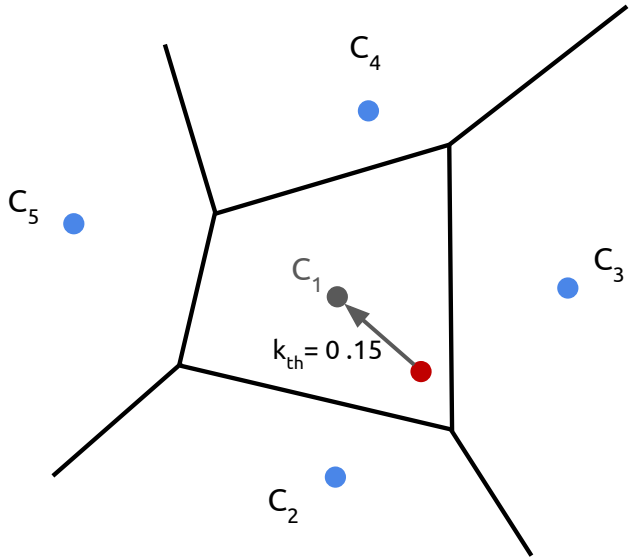
ConANN (IVF)

- Terminates the search using a global **conformal-calibrated threshold λ** that **guarantees** reaching target recall in **expectation**.
  - Terminate once k_{th} distance after processing a partition is $\mathbf{k_{th} \leq \lambda}$

- Conformal Risk Control (CRC)
  - Find the minimum λ so that the expectation of recall across the queries of workload is $\mathbb{E}[R] \geq R_{target}$

ANN(q, k, 0.80) → λ=0.1

IVF partitions $P_1 – P_5$ with centroids $C_1 – C_5$



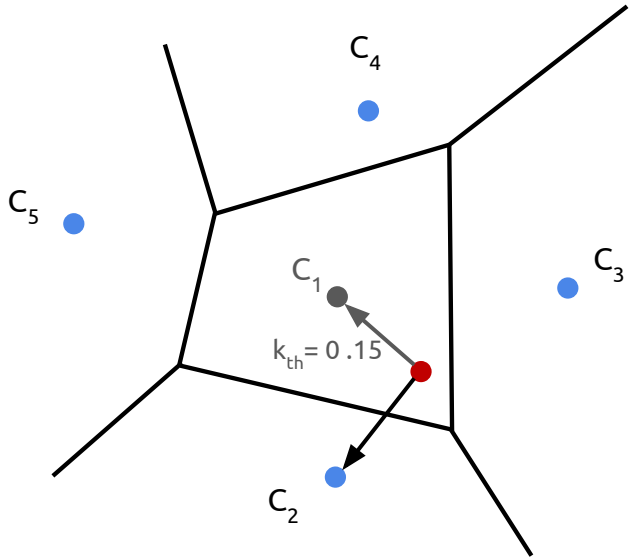
In the original paper False Negative Rate is used as ***FNR = 1 - R_{target}***

ConANN (IVF)

- Terminates the search using a global **conformal-calibrated threshold λ** that **guarantees** reaching target recall in **expectation**.
  - Terminate once k_{th} distance after processing a partition is $\mathbf{k_{th} \leq \lambda}$
- Conformal Risk Control (CRC)
  - Find the minimum λ so that the expectation of recall across the queries of workload is $\mathbb{E}[R] \geq R_{target}$

ANN(q, k, 0.80) → λ=0.1

IVF partitions $P_1 – P_5$ with centroids $C_1 – C_5$



In the original paper False Negative Rate is used as ***FNR = 1 - R_{target}***

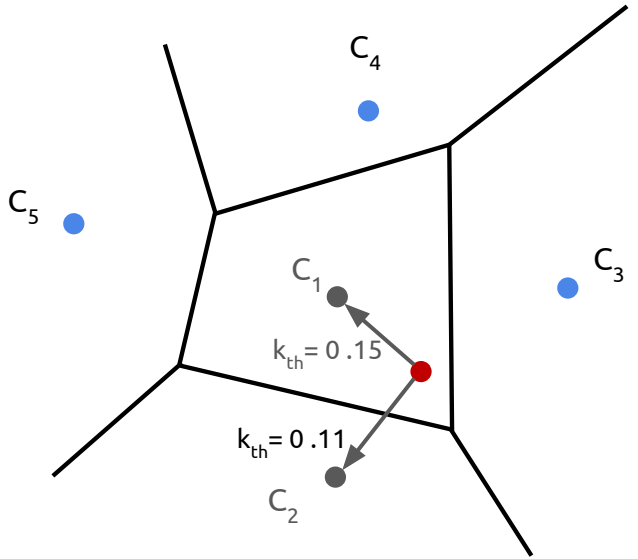
ConANN (IVF)

- Terminates the search using a global **conformal-calibrated threshold λ** that **guarantees** reaching target recall in **expectation**.
  - Terminate once k_{th} distance after processing a partition is $\mathbf{k_{th} \leq \lambda}$

- Conformal Risk Control (CRC)
  - Find the minimum λ so that the expectation of recall across the queries of workload is $\mathbb{E}[R] \geq R_{target}$

ANN(q, k, 0.80) → λ=0.1

IVF partitions $P_1 – P_5$ with centroids $C_1 – C_5$



In the original paper False Negative Rate is used as ***FNR = 1 - R_{target}***

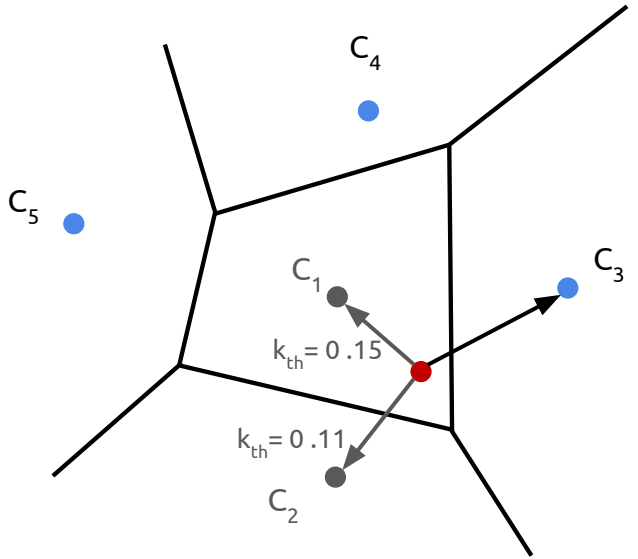
ConANN (IVF)

- Terminates the search using a global **conformal-calibrated threshold λ** that **guarantees** reaching target recall in **expectation**.
  - Terminate once k_{th} distance after processing a partition is $\mathbf{k_{th} \leq \lambda}$

- Conformal Risk Control (CRC)
  - Find the minimum λ so that the expectation of recall across the queries of workload is $\mathbb{E}[R] \geq R_{target}$

ANN(q, k, 0.80) → λ=0.1

IVF partitions $P_1 – P_5$ with centroids $C_1 – C_5$



In the original paper False Negative Rate is used as ***FNR = 1 - R_{target}***

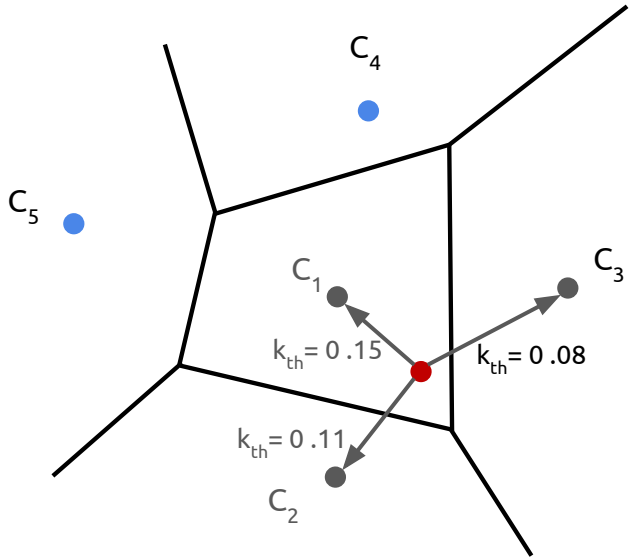
ConANN (IVF)

- Terminates the search using a global **conformal-calibrated threshold λ** that **guarantees** reaching target recall in **expectation**.
  - Terminate once k_{th} distance after processing a partition is $\mathbf{k_{th} \leq \lambda}$

- Conformal Risk Control (CRC)
  - Find the minimum λ so that the expectation of recall across the queries of workload is $\mathbb{E}[R] \geq R_{target}$

ANN(q, k, 0.80) → λ=0.1

IVF partitions $P_1 – P_5$ with centroids $C_1 – C_5$

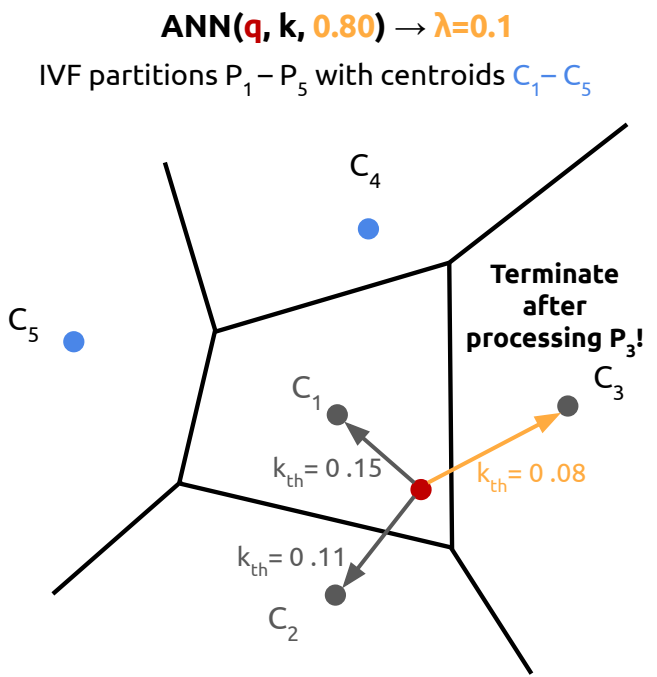


In the original paper False Negative Rate is used as ***FNR = 1 - R_{target}***

ConANN (IVF)

- Terminates the search using a global **conformal-calibrated threshold λ** that **guarantees** reaching target recall in **expectation**.
  - Terminate once k_{th} distance after processing a partition is $\mathbf{k_{th} \leq \lambda}$

- Conformal Risk Control (CRC)
  - Find the minimum λ so that the expectation of recall across the queries of workload is $\mathbb{E}[R] \geq R_{target}$



In the original paper False Negative Rate is used as ***FNR = 1 - R_{target}***

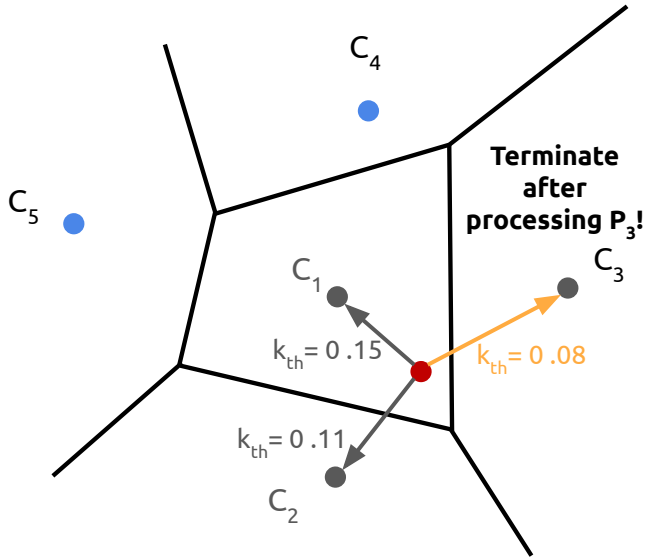
ConANN (IVF)

- Terminates the search using a global **conformal-calibrated threshold λ** that **guarantees** reaching target recall in **expectation**.
  - Terminate once k_{th} distance after processing a partition is $\mathbf{k_{th} \leq \lambda}$

- Conformal Risk Control (CRC)
  - Find the minimum λ so that the expectation of recall across the queries of workload is $\mathbb{E}[R] \geq R_{target}$

ANN(q, k, 0.80) → λ=0.1

IVF partitions $P_1 – P_5$ with centroids $C_1 – C_5$



Conformal Guarantees

Extension to kNN-graphs
Per-query guarantees
Comparison with other approaches

In the original paper False Negative Rate is used as ***FNR = 1 - R_{target}***

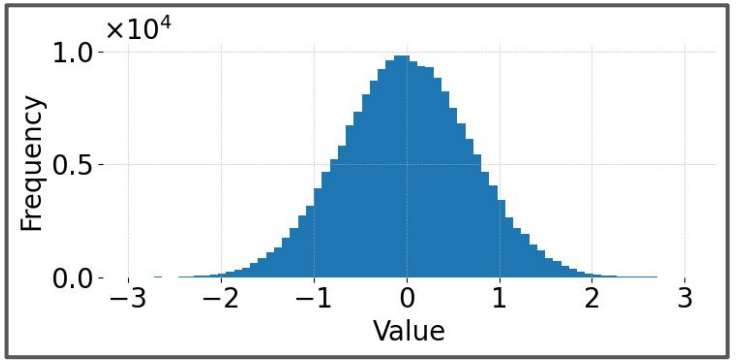
Ada-ef (HNSW, Inner Product)

- Preliminary search → **Predict ef (HNSW)**

Ada-ef (HNSW, Inner Product)

- Preliminary search → **Predict ef (HNSW)**
- Inner product: distances of query to the dataset are **normally distributed**

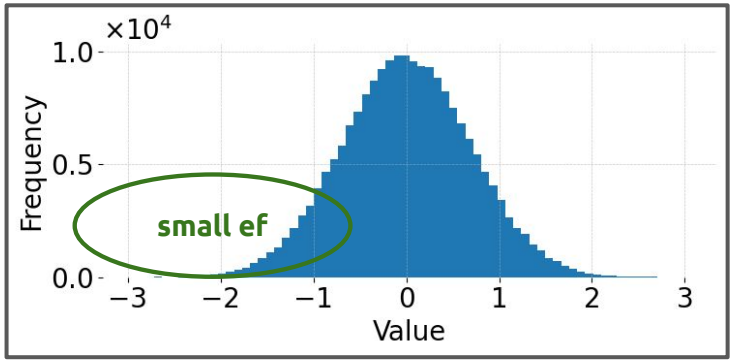
GLOVE query distances to all dataset vectors



Ada-ef (HNSW, Inner Product)

- Preliminary search → **Predict ef (HNSW)**
- Inner product: distances of query to the dataset are **normally distributed**
- Preliminary search: find neighbor distances in the
  - leftmost tail → **easier query, small ef, higher score**

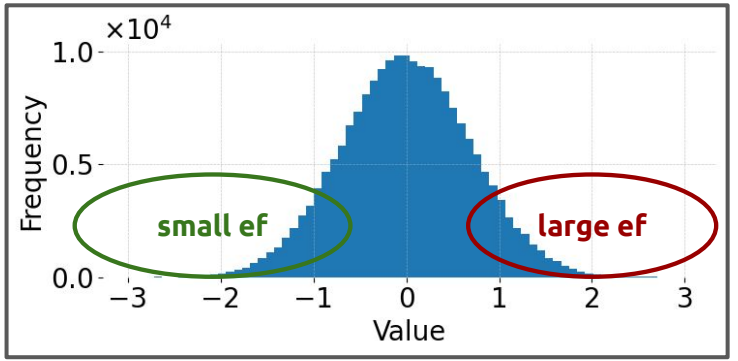
GLOVE query distances to all dataset vectors



Ada-ef (HNSW, Inner Product)

- Preliminary search → **Predict ef (HNSW)**
- Inner product: distances of query to the dataset are **normally distributed**
- Preliminary search: find neighbor distances in the
  - leftmost tail → **easier query, small ef, higher score**
  - rightmost tail → **harder query, large ef, lower score**

GLOVE query distances to all dataset vectors



Ada-ef (HNSW, Inner Product)

Offline

Ada-ef (HNSW, Inner Product)

Offline

Precompute dataset's covariance & mean

Ada-ef (HNSW, Inner Product)

Offline

Precompute dataset's covariance & mean

Precompute score table (200 queries)
Score → [ef, recall]

Score	EF_Recall
…	…
54	((100, 0.81), (150, 0.85), …, (575, 0.96))
…	…
90	((100, 0.85), (150, 0.91), (250, 0.965))
…	…
100	((100, 1.0), (150, 1.0))

Ada-ef (HNSW, Inner Product)

Offline

Precompute dataset's covariance & mean

Precompute score table (200 queries)
Score → [ef, recall]

Score	EF_Recall
…	…
54	((100, 0.81), (150, 0.85), …, (575, 0.96))
…	…
90	((100, 0.85), (150, 0.91), (250, 0.965))
…	…
100	((100, 1.0), (150, 1.0))

Online

ANN(q, k, 0.80)

Ada-ef (HNSW, Inner Product)

Offline

Precompute dataset's covariance & mean

Precompute score table (200 queries)
Score → [ef, recall]

Score	EF_Recall
…	…
54	((100, 0.81), (150, 0.85), …, (575, 0.96))
…	…
90	((100, 0.85), (150, 0.91), (250, 0.965))
…	…
100	((100, 1.0), (150, 1.0))

Online

ANN(q, k, 0.80) →

Preliminary Search
Gather distances after 2-hops of HNSW search
D = [...]

Ada-ef (HNSW, Inner Product)

Offline

Precompute dataset's covariance & mean

Precompute score table (200 queries)
Score → [ef, recall]

Score	EF_Recall
…	…
54	((100, 0.81), (150, 0.85), …, (575, 0.96))
…	…
90	((100, 0.85), (150, 0.91), (250, 0.965))
…	…
100	((100, 1.0), (150, 1.0))

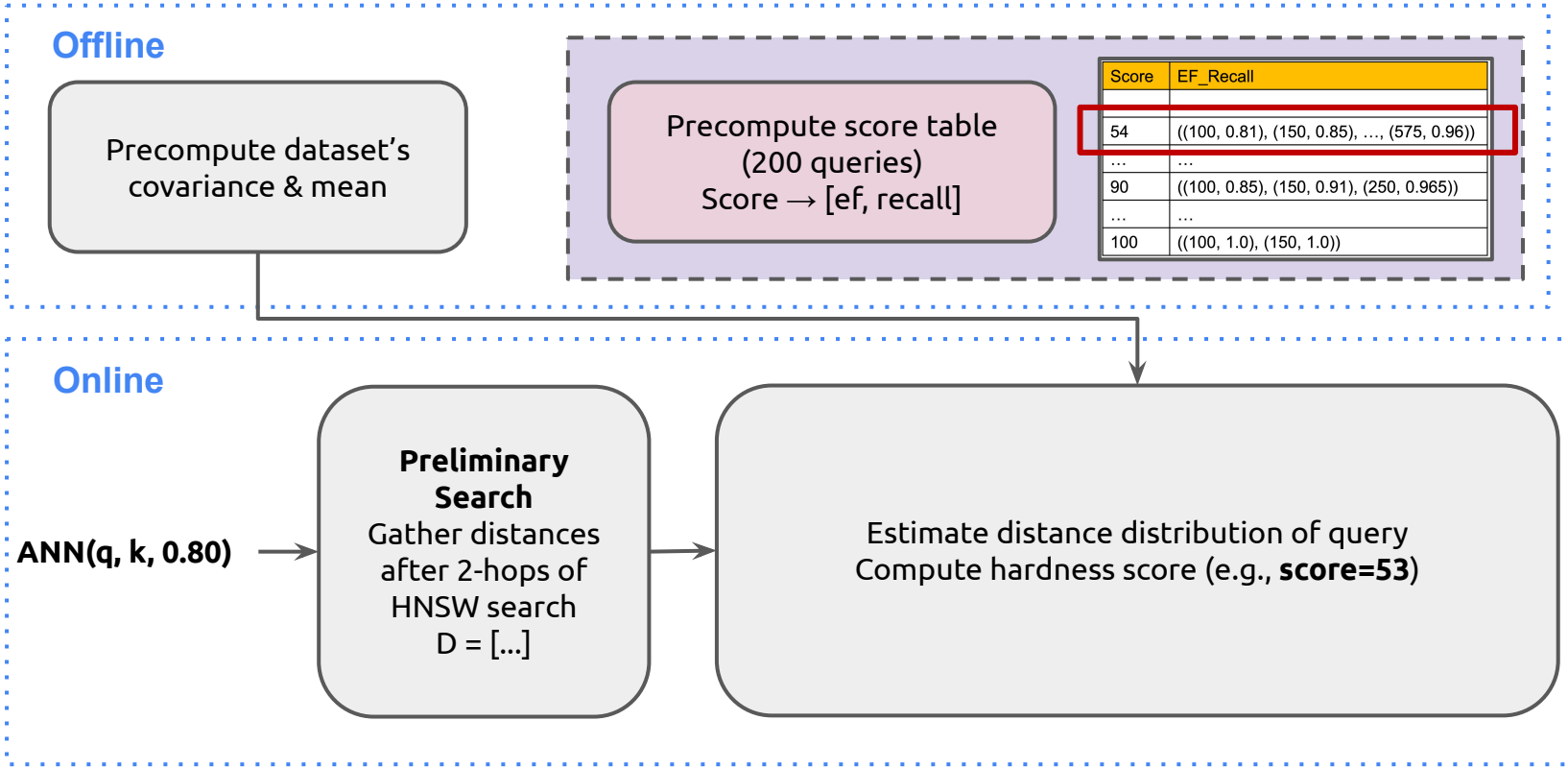
Online

ANN(q, k, 0.80) →

Preliminary Search
Gather distances after 2-hops of HNSW search
D = [...]

→ Estimate distance distribution of query
Compute hardness score (e.g., **score=53**)

Ada-ef (HNSW, Inner Product)



Ada-ef (HNSW, Inner Product)

Offline

Precompute dataset's covariance & mean

Precompute score table (200 queries)
Score → [ef, recall]

Score	EF_Recall
54	((100, 0.81), (150, 0.85), …, (575, 0.96))
…	…
90	((100, 0.85), (150, 0.91), (250, 0.965))
…	…
100	((100, 1.0), (150, 1.0))

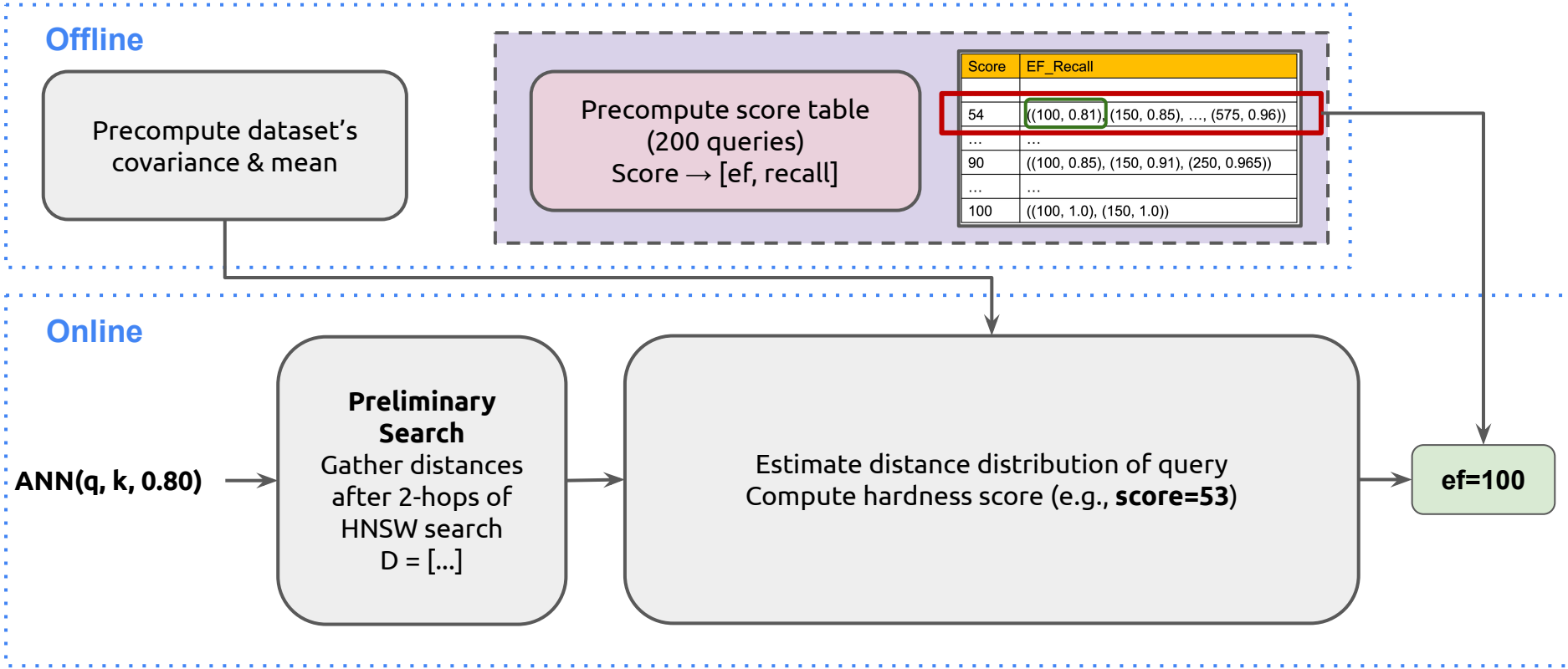
Online

ANN(q, k, 0.80) →

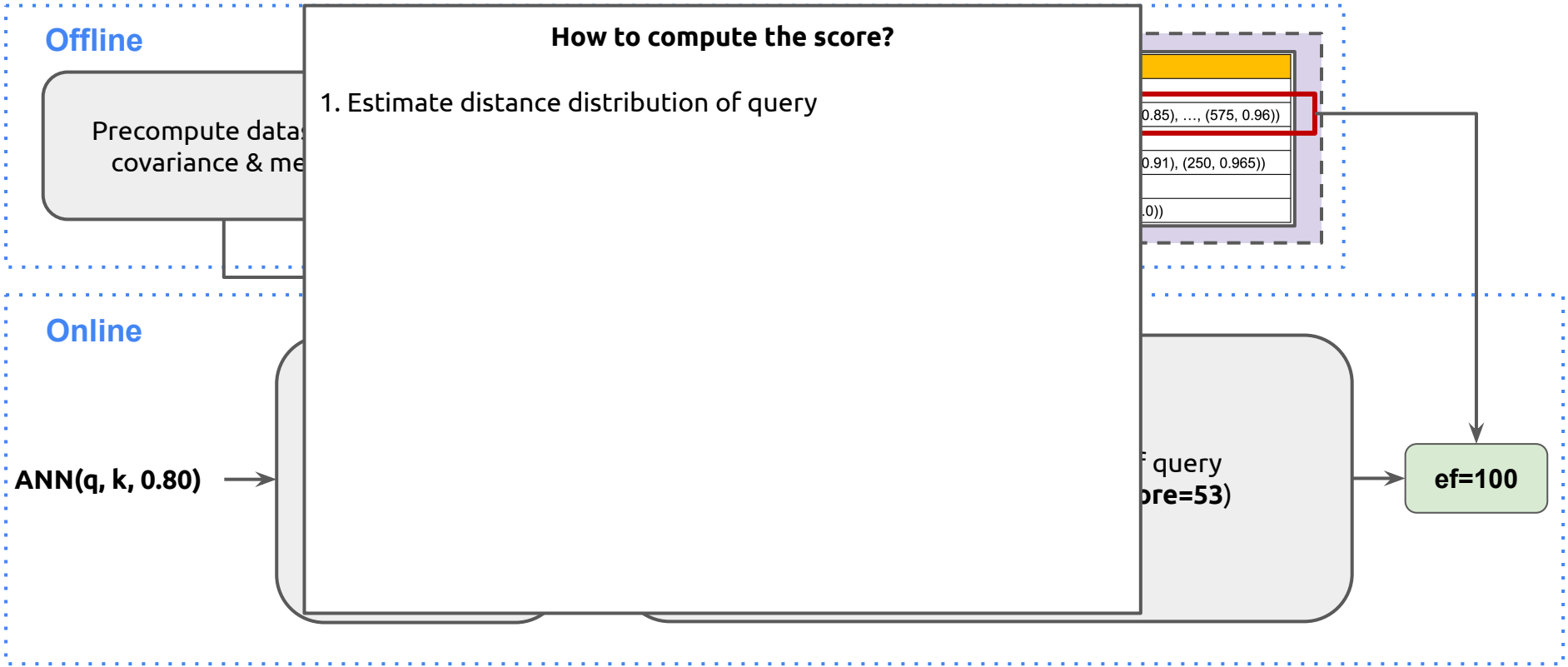
Preliminary Search
Gather distances after 2-hops of HNSW search
D = [...]

Estimate distance distribution of query
Compute hardness score (e.g., **score=53**)

Ada-ef (HNSW, Inner Product)



Ada-ef (HNSW, Inner Product)



Ada-ef (HNSW, Inner Product)

Offline

Precompute data
covariance & me

0.85), …, (575, 0.96))
0.91), (250, 0.965))
.0))

Online

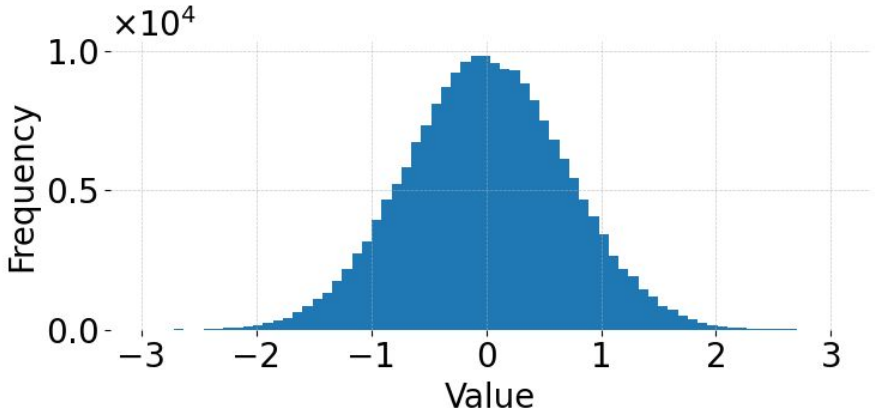
ANN(q, k, 0.80)

f query
ore=53)

ef=100

How to compute the score?

1. Estimate distance distribution of query



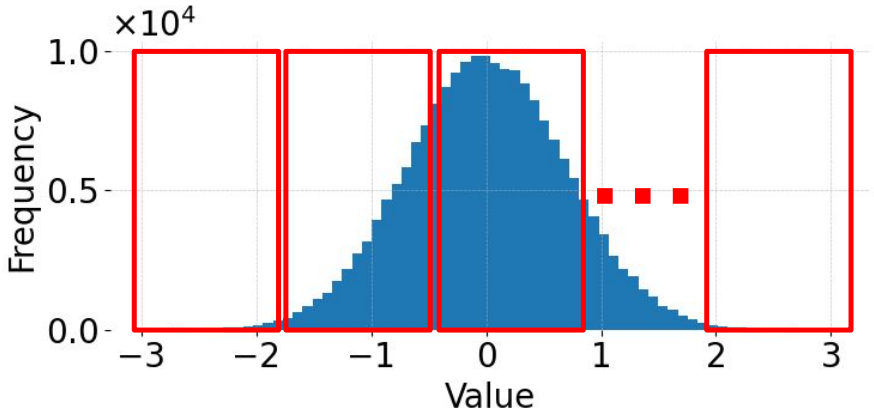
Ada-ef (HNSW, Inner Product)

Offline

Precompute data
covariance & me

How to compute the score?

1. Estimate distance distribution of query
2. Discretize in 1000 buckets (0.1 percentile each)



0.85), …, (575, 0.96))
0.91), (250, 0.965))
.0))

Online

ANN(q, k, 0.80)

query
ore=53)

ef=100

Ada-ef (HNSW, Inner Product)

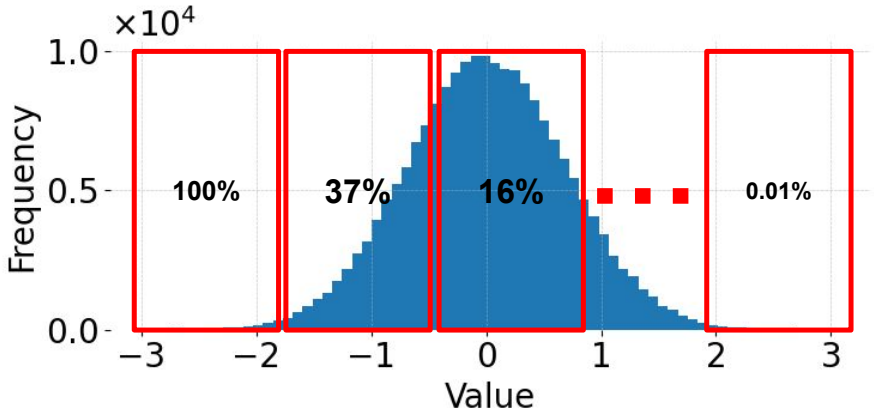
Offline

Precompute data
covariance & me

0.85), …, (575, 0.96))
0.91), (250, 0.965))
.0))

How to compute the score?

1. Estimate distance distribution of query
2. Discretize in 1000 buckets (0.1 percentile each)
3. Assign score of exponential decay in each bucket $w_i = 100e^{-i+1}$



Online

ANN(q, k, 0.80)

query
ore=53)

ef=100

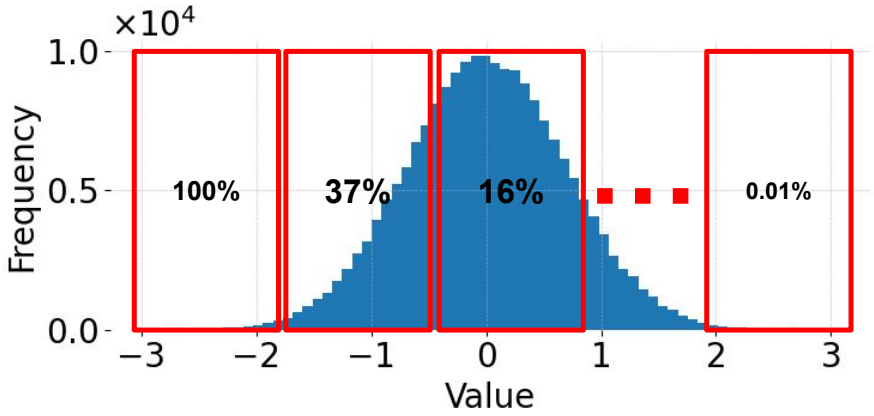
Ada-ef (HNSW, Inner Product)

Offline

Precompute data
covariance & me

How to compute the score?

1. Estimate distance distribution of query
2. Discretize in 1000 buckets (0.1 percentile each)
3. Assign score of exponential decay in each bucket $w_i = 100e^{-i+1}$
4. Allocate the |D| distances found in the buckets



0.85), …, (575, 0.96))
0.91), (250, 0.965))
.0))

Online

ANN(q, k, 0.80)

query
ore=53)

ef=100

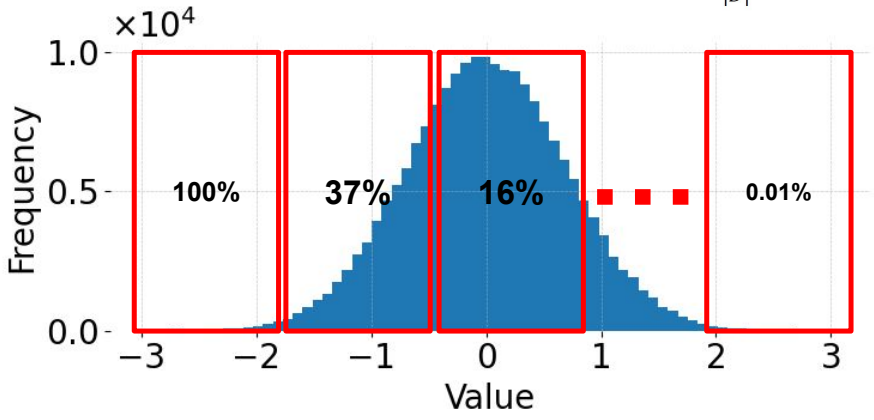
Ada-ef (HNSW, Inner Product)

Offline

Precompute data
covariance & me

How to compute the score?

1. Estimate distance distribution of query
2. Discretize in 1000 buckets (0.1 percentile each)
3. Assign score of exponential decay in each bucket $w_i = 100e^{-i+1}$
4. Allocate the |D| distances found in the buckets
5. Compute score as weighted average $s(\mathbf{q}) = \sum_{i=1}^{m}(w_i \frac{c_i}{|D|})$



0.85), …, (575, 0.96))
0.91), (250, 0.965))
.0))

Online

ANN(q, k, 0.80)

query
ore=53)

ef=100

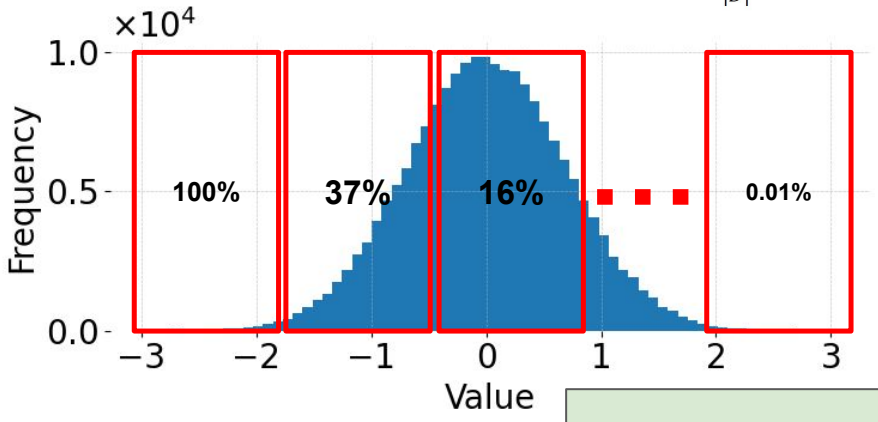
Ada-ef (HNSW, Inner Product)

Offline

Precompute data
covariance & me

How to compute the score?

1. Estimate distance distribution of query
2. Discretize in 1000 buckets (0.1 percentile each)
3. Assign score of exponential decay in each bucket $w_i = 100e^{-i+1}$
4. Allocate the |D| distances found in the buckets
5. Compute score as weighted average $s(\mathbf{q}) = \sum_{i=1}^{m}(w_i \frac{c_i}{|D|})$



0.85), …, (575, 0.96))
0.91), (250, 0.965))
.0))

Online

ANN(q, k, 0.80)

query
ore=53)

ef=100

Rule based
Lightweight setup

Extension for L2
Extension to IVF

LAET (HNSW & IVF)

Preliminary search → predict **number of distance calcs. (ndc)** to be performed

LAET (HNSW & IVF)

Preliminary search → predict **number of distance calcs. (ndc)** to be performed

Index	Feature
Both	Query dimensions Distance to 1st nearest neighbor (D_1) Distance to 10th nearest neighbor (D_{10})
HNSW	Distance to first node in the base layer (D_{start}) D_1 / D_{start} D_{10} / D_{start}
IVF	$dist_{q \to Ci} / dist_{q \to C1}$ with i in {10, 20, …, 100} D_1 / D_{10} $D_1 / dist_{q \to C1}$

LAET (HNSW & IVF)

Preliminary search → predict **number of distance calcs. (ndc)** to be performed

1 per query, after ?? NDC in preliminary search

Index	Feature
Both	Query dimensions Distance to 1st nearest neighbor (D_1) Distance to 10th nearest neighbor (D_{10})
HNSW	Distance to first node in the base layer (D_{start}) D_1 / D_{start} D_{10} / D_{start}
IVF	$dist_{q \to Ci} / dist_{q \to C1}$ with i in {10, 20, …, 100} D_1 / D_{10} $D_1 / dist_{q \to C1}$

LAET (HNSW & IVF)

Preliminary search → predict **number of distance calcs. (ndc)** to be performed

1 per query, after ?? NDC in preliminary search

Index	Feature
Both	Query dimensions Distance to 1st nearest neighbor (D_1) Distance to 10th nearest neighbor (D_{10})
HNSW	Distance to first node in the base layer (D_{start}) D_1 / D_{start} D_{10} / D_{start}
IVF	$dist_{q \to Ci} / dist_{q \to C1}$ with i in {10, 20, …, 100} D_1 / D_{10} $D_1 / dist_{q \to C1}$

$NDC_{maxRecall}$

LAET (HNSW & IVF)

Preliminary search → predict **number of distance calcs. (ndc)** to be performed

1 per query, after ?? NDC in preliminary search

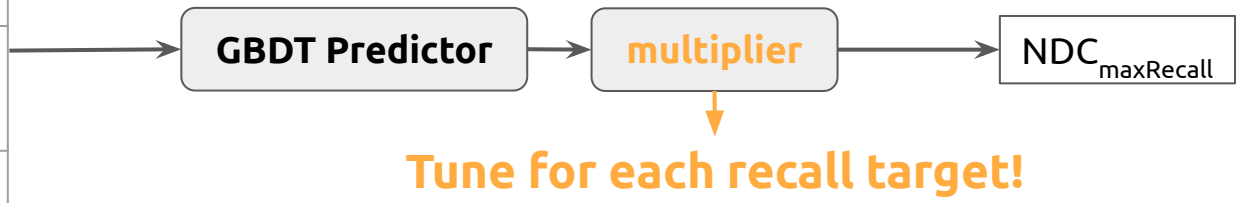
Index	Feature
Both	Query dimensions Distance to 1st nearest neighbor (D_1) Distance to 10th nearest neighbor (D_{10})
HNSW	Distance to first node in the base layer (D_{start}) D_1 / D_{start} D_{10} / D_{start}
IVF	$dist_{q \to Ci} / dist_{q \to C1}$ with i in {10, 20, …, 100} D_1 / D_{10} $D_1 / dist_{q \to C1}$

LAET (HNSW & IVF)

Preliminary search → predict **number of distance calcs. (ndc)** to be performed

1 per query, after ?? NDC in preliminary search

Index	Feature
Both	Query dimensions Distance to 1st nearest neighbor (D_1) Distance to 10th nearest neighbor (D_{10})
HNSW	Distance to first node in the base layer (D_{start}) D_1 / D_{start} D_{10} / D_{start}
IVF	$dist_{q \to Ci} / dist_{q \to C1}$ with i in {10, 20, …, 100} D_1 / D_{10} $D_1 / dist_{q \to C1}$

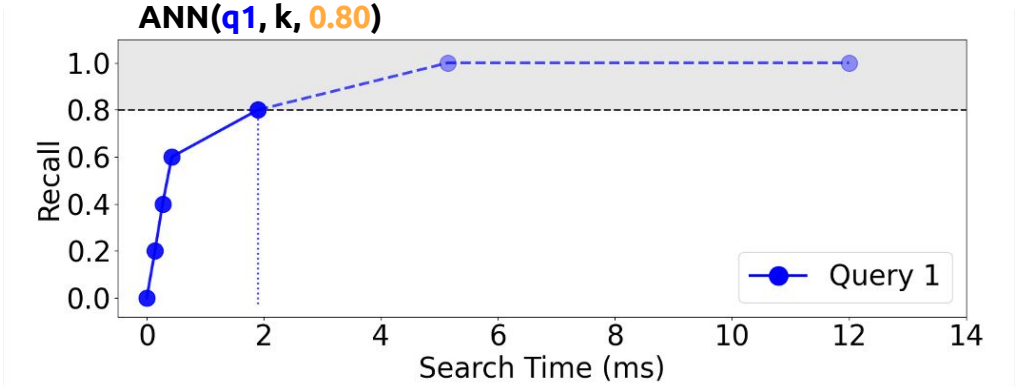


LAET (HNSW & IVF)

Preliminary search → predict **number of distance calcs. (ndc)** to be performed

1 per query, after ?? NDC in preliminary search

Index	Feature
Both	Query dimensions Distance to 1st nearest neighbor (D_1) Distance to 10th nearest neighbor (D_{10})
HNSW	Distance to first node in the base layer (D_{start}) D_1 / D_{start} D_{10} / D_{start}
IVF	$dist_{q \to Ci} / dist_{q \to C1}$ with i in {10, 20, …, 100} D_1 / D_{10} $D_1 / dist_{q \to C1}$

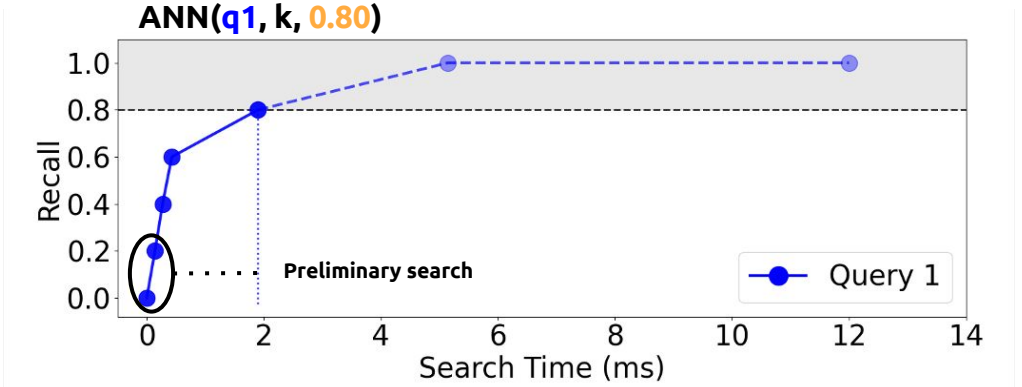


LAET (HNSW & IVF)

Preliminary search → predict **number of distance calcs. (ndc)** to be performed

1 per query, after ?? NDC in preliminary search

Index	Feature
Both	Query dimensions Distance to 1st nearest neighbor (D_1) Distance to 10th nearest neighbor (D_{10})
HNSW	Distance to first node in the base layer (D_{start}) D_1 / D_{start} D_{10} / D_{start}
IVF	$dist_{q \to Ci} / dist_{q \to C1}$ with i in {10, 20, …, 100} D_1 / D_{10} $D_1 / dist_{q \to C1}$

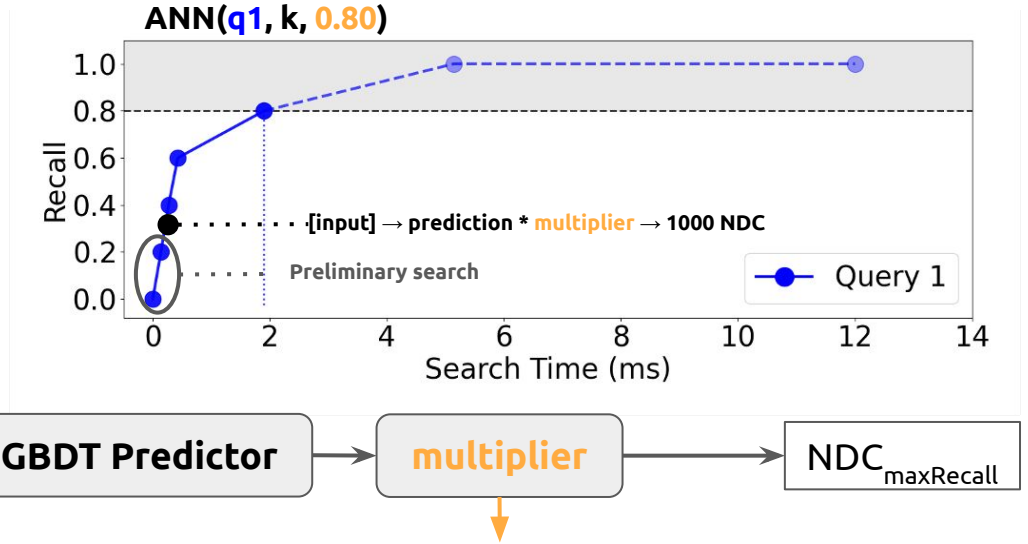


LAET (HNSW & IVF)

Preliminary search → predict **number of distance calcs. (ndc)** to be performed

1 per query, after ?? NDC in preliminary search

Index	Feature
Both	Query dimensions Distance to 1st nearest neighbor (D_1) Distance to 10th nearest neighbor (D_{10})
HNSW	Distance to first node in the base layer (D_{start}) D_1 / D_{start} D_{10} / D_{start}
IVF	$dist_{q \to Ci} / dist_{q \to C1}$ with i in {10, 20, …, 100} D_1 / D_{10} $D_1 / dist_{q \to C1}$

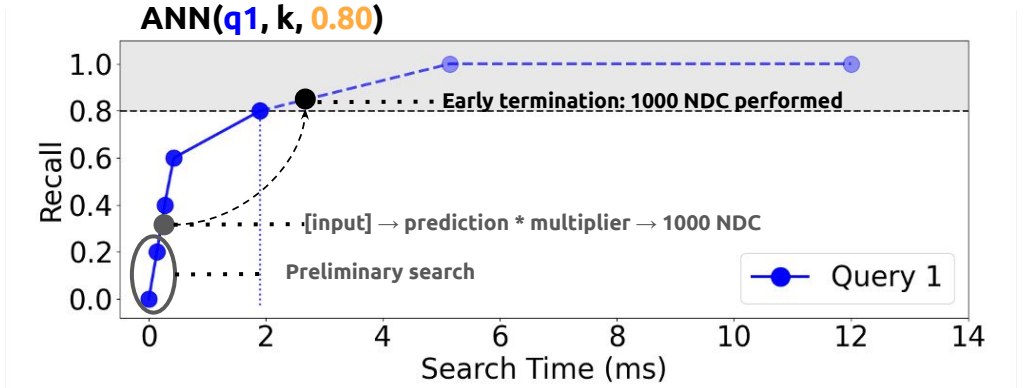


LAET (HNSW & IVF)

Preliminary search → predict **number of distance calcs. (ndc)** to be performed

1 per query, after ?? NDC in preliminary search

Index	Feature
Both	Query dimensions Distance to 1st nearest neighbor (D_1) Distance to 10th nearest neighbor (D_{10})
HNSW	Distance to first node in the base layer (D_{start}) D_1 / D_{start} D_{10} / D_{start}
IVF	$dist_{q \to Ci} / dist_{q \to C1}$ with i in {10, 20, …, 100} D_1 / D_{10} $D_1 / dist_{q \to C1}$

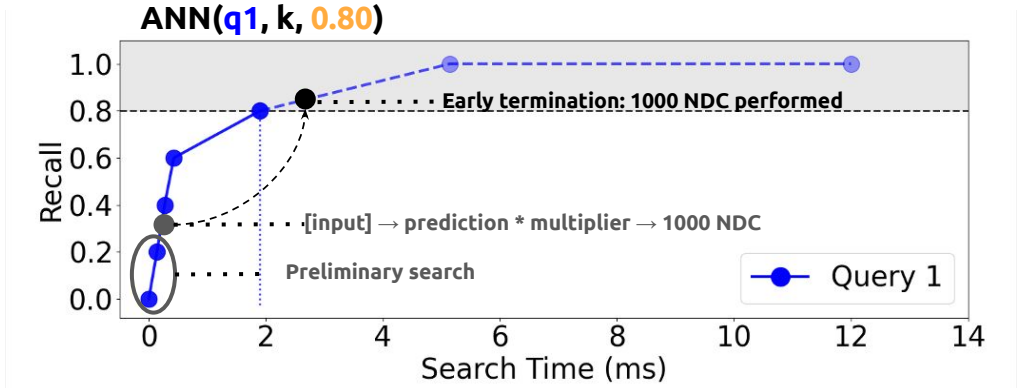


LAET (HNSW & IVF)

Preliminary search → predict **number of distance calcs. (ndc)** to be performed

1 per query, after ?? NDC in preliminary search

Index	Feature
Both	Query dimensions Distance to 1st nearest neighbor (D_1) Distance to 10th nearest neighbor (D_{10})
HNSW	Distance to first node in the base layer (D_{start}) D_1 / D_{start} D_{10} / D_{start}
IVF	$dist_{q \to Ci}$ / $dist_{q \to C1}$ with i in {10, 20, …, 100} D_1 / D_{10} D_1 / $dist_{q \to C1}$



Generalizable to different indexes!

Optimization of tuning for preliminary amount of search and multiplier

DARTH (HNSW & IVF)

Predict the recall of a query any time during the search

DARTH (HNSW & IVF)

Predict the recall of a query any time during the search

Type	Feature (**IVF**)
Index	nstep/**npartition** ndis ninserts
NN Dist	firstNN closestNN furthestNN **currPartitionDist**
NN Stats	Avg, var Med, perc25 perc75

DARTH (HNSW & IVF)

Predict the recall of a query any time during the search

many samples per query

Type	Feature (**IVF**)
Index	nstep/**npartition** ndis ninserts
NN Dist	firstNN closestNN furthestNN **currPartitionDist**
NN Stats	Avg, var Med, perc25 perc75

DARTH (HNSW & IVF)

Predict the recall of a query any time during the search

many samples per query

Type	Feature (IVF)
Index	nstep/**npartition** ndis ninserts
NN Dist	firstNN closestNN furthestNN **currPartitionDist**
NN Stats	Avg, var Med, perc25 perc75

Current Recall

DARTH (HNSW & IVF)

Predict the recall of a query any time during the search

many samples per query

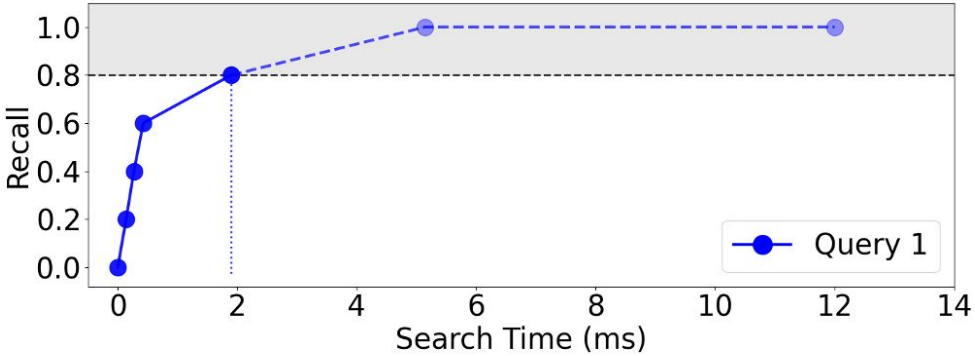
Type	Feature (IVF)
Index	nstep/**npartition** ndis ninserts
NN Dist	firstNN closestNN furthestNN **currPartitionDist**
NN Stats	Avg, var Med, perc25 perc75

DARTH (HNSW & IVF)

Predict the recall of a query any time during the search

many samples per query

Type	Feature (IVF)
Index	nstep/**npartition** ndis ninserts
NN Dist	firstNN closestNN furthestNN **currPartitionDist**
NN Stats	Avg, var Med, perc25 perc75



GBDT Recall Predictor
Trained in seconds for 100Ms samples
Inference in 0.03ms

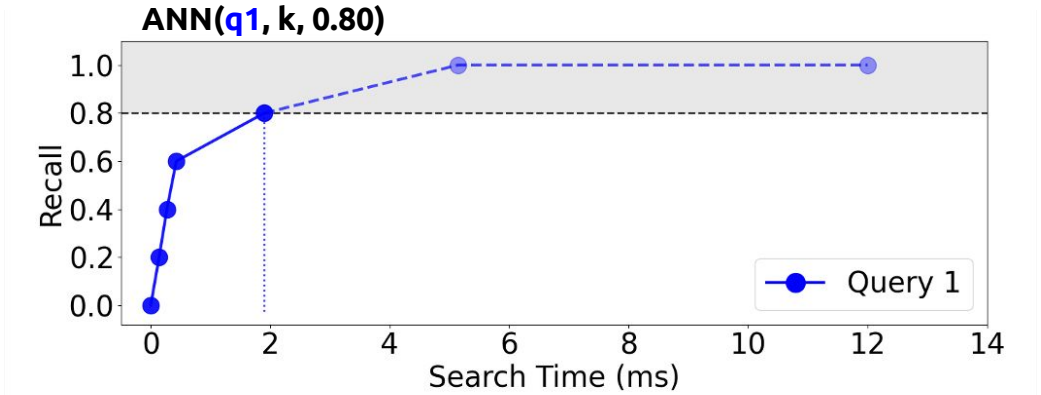
Current Recall

DARTH (HNSW & IVF)

Predict the recall of a query any time during the search

many samples per query

Type	Feature (IVF)
Index	nstep/**npartition** ndis ninserts
NN Dist	firstNN closestNN furthestNN **currPartitionDist**
NN Stats	Avg, var Med, perc25 perc75



GBDT Recall Predictor
Trained in seconds for 100Ms samples
Inference in 0.03ms

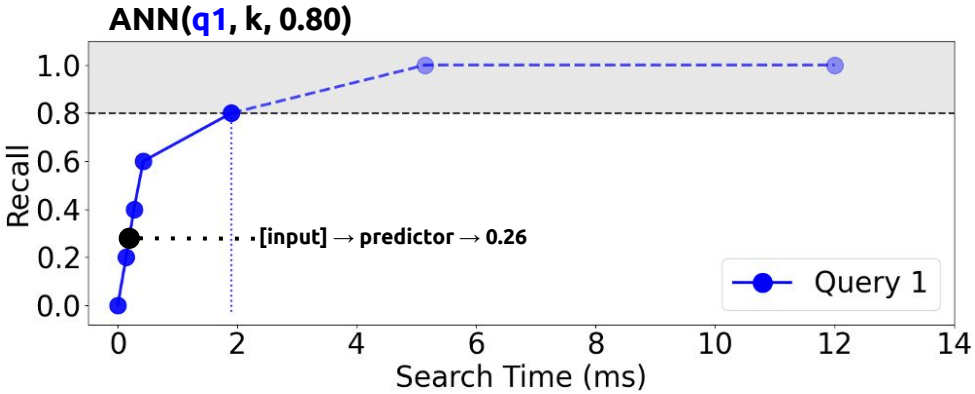
Current Recall

DARTH (HNSW & IVF)

Predict the recall of a query any time during the search

many samples per query

Type	Feature (IVF)
Index	nstep/**npartition** ndis ninserts
NN Dist	firstNN closestNN furthestNN **currPartitionDist**
NN Stats	Avg, var Med, perc25 perc75



GBDT Recall Predictor
Trained in seconds for 100Ms samples
Inference in 0.03ms

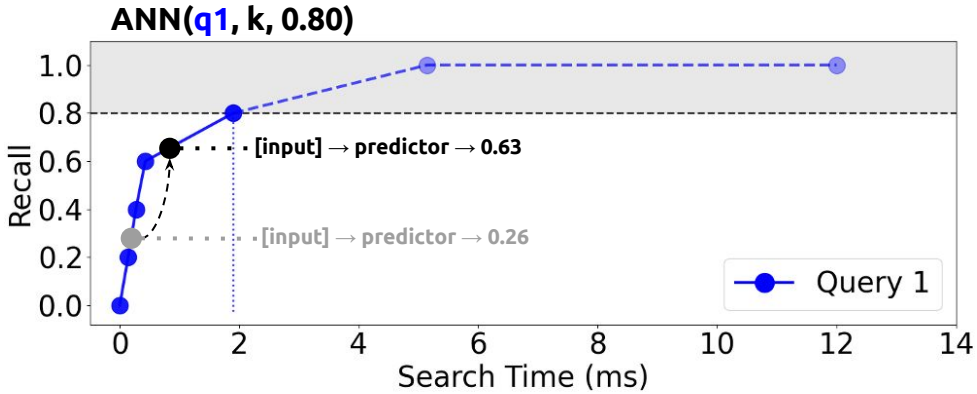
Current Recall

DARTH (HNSW & IVF)

Predict the recall of a query any time during the search

many samples per query

Type	Feature (IVF)
Index	nstep/**npartition** ndis ninserts
NN Dist	firstNN closestNN furthestNN **currPartitionDist**
NN Stats	Avg, var Med, perc25 perc75



GBDT Recall Predictor
Trained in seconds for 100Ms samples
Inference in 0.03ms

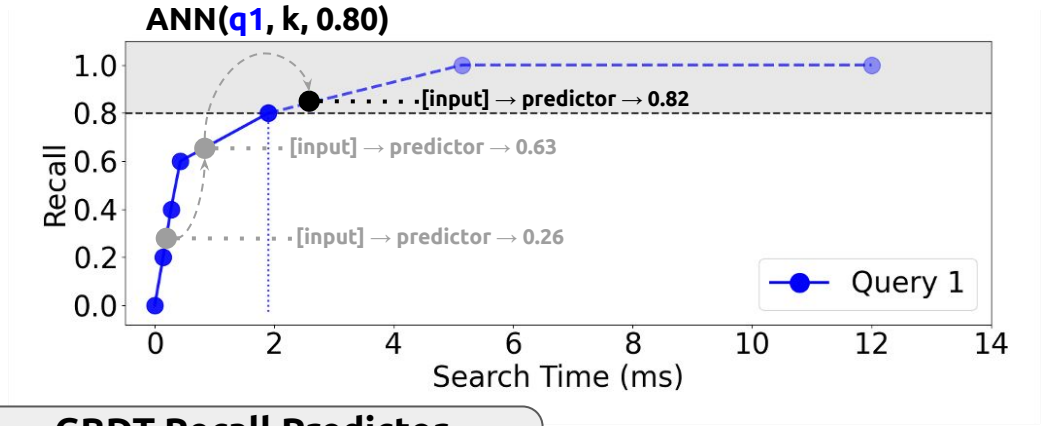
Current Recall

DARTH (HNSW & IVF)

Predict the recall of a query any time during the search

many samples per query

Type	Feature (IVF)
Index	nstep/**npartition** ndis ninserts
NN Dist	firstNN closestNN furthestNN **currPartitionDist**
NN Stats	Avg, var Med, perc25 perc75



GBDT Recall Predictor
Trained in seconds for 100Ms samples
Inference in 0.03ms

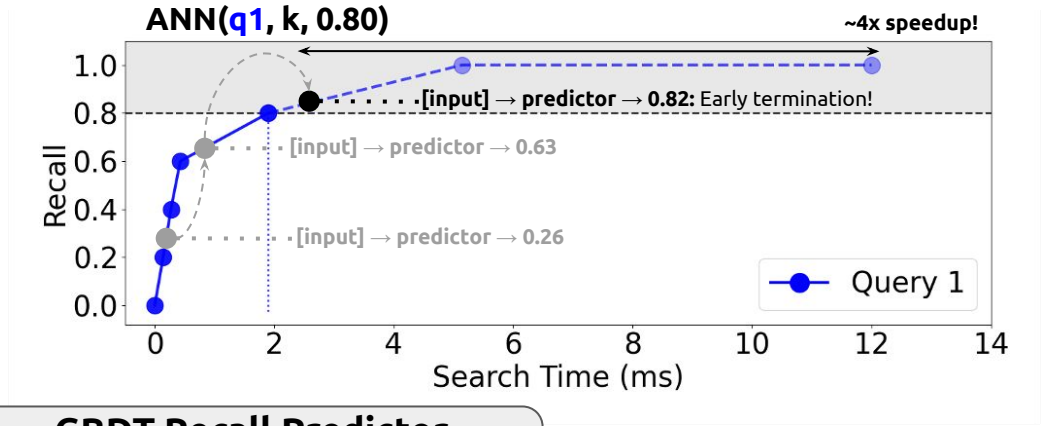
Current Recall

DARTH (HNSW & IVF)

Predict the recall of a query any time during the search

many samples per query

Type	Feature (IVF)
Index	nstep/**npartition** ndis ninserts
NN Dist	firstNN closestNN furthestNN **currPartitionDist**
NN Stats	Avg, var Med, perc25 perc75



GBDT Recall Predictor
Trained in seconds for 100Ms samples
Inference in 0.03ms

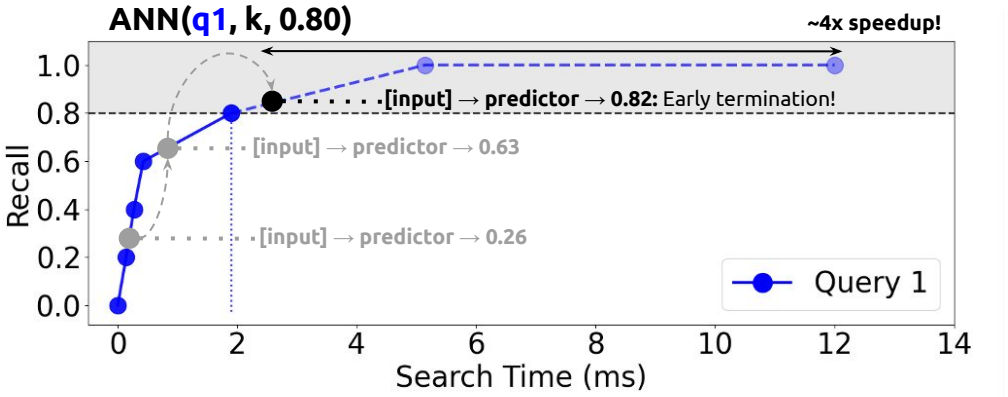
Current Recall

DARTH (HNSW & IVF)

Predict the recall of a query any time during the search

many samples per query

Type	Feature (IVF)
Index	nstep/**npartition** ndis ninserts
NN Dist	firstNN closestNN furthestNN **currPartitionDist**
NN Stats	Avg, var Med, perc25 perc75



GBDT Recall Predictor
Trained in seconds for 100Ms samples
Inference in 0.03ms

Current Recall

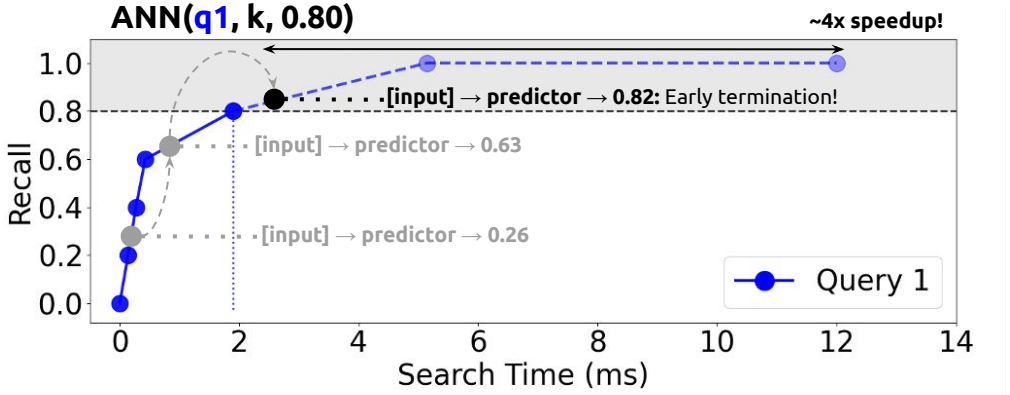
Adopted by Google

DARTH (HNSW & IVF)

Predict the recall of a query any time during the search

many samples per query

Type	Feature (IVF)
Index	nstep/**npartition** ndis ninserts
NN Dist	firstNN closestNN furthestNN **currPartitionDist**
NN Stats	Avg, var Med, perc25 perc75



GBDT Recall Predictor
Trained in seconds for 100Ms samples
Inference in 0.03ms

Current Recall

Generalizable to different indexes!
Heuristics instead of tuning!

Optimization for the amount of training samples

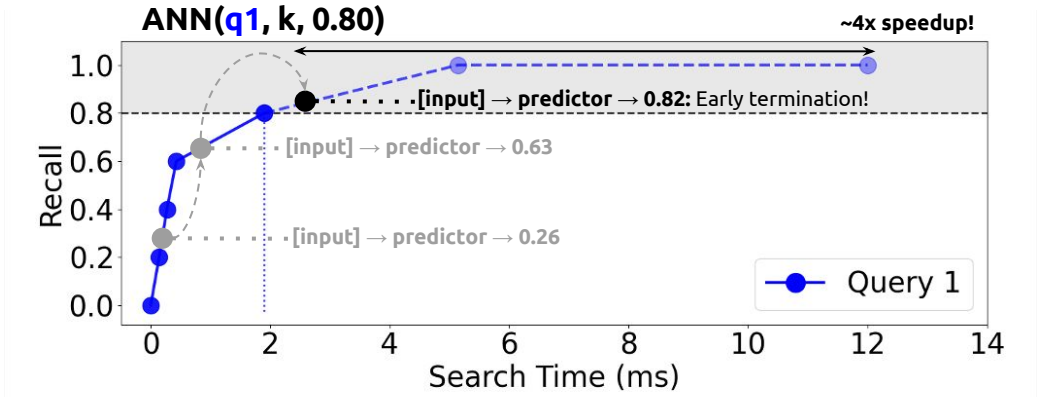
Adopted by Google

DARTH (HNSW & IVF)

Predict the recall of a query any time during the search

many samples per query

Type	Feature (IVF)
Index	nstep/**npartition** ndis ninserts
NN Dist	firstNN closestNN furthestNN **currPartitionDist**
NN Stats	Avg, var Med, perc25 perc75



GBDT Recall Predictor
Trained in seconds for 100Ms samples
Inference in 0.03ms

Current Recall

Most methods require GroundTruths (GT) for training

GT calculation can be super fast with **DaiSy**!

Generalizable to different indexes!
Heuristics instead of tuning!

Optimization for the amount of training samples

Adopted by Google

Zhang et al. “Fast, Approximate Vector Queries on Very Large Unstructured Datasets” NSDI 2023
Zheng et al. "Learned probing cardinality estimation for high-dimensional approximate NN search." ICDE 2023.
Zeng et al. "Lira: A learning-based query-aware partition framework for large-scale ann search." WebConf 2025.
Teofili et al. "Patience in proximity: a simple early termination strategy for HNSW graph traversal in approximate K-nearest neighbor search." ECIR 2025.
Busolin et al. "Early exit strategies for approximate k-nn search in dense retrieval." ICIKM 2024.

The list goes on!

- **Auncel (IVF)**: Estimate the recall of each IVF partition using geometric features
- **PCE-IVFPQ (IVF):** Estimate a minimum nprobe for a target recall
- **LIRA (IVF)**: Predict the probability a partition contains a groundtruth
  - Tunable probability threshold → target recall
- **PiP (HNSW)** & **PatEE (IVF)**: Terminate based on the change rate of the identified nearest neighbors (Patience)

Open problems and future directions

- **Comparative evaluation between early termination methods**
  - All of the methods compare to plain search with tuning (BSL)
  - Combinations?

Method	Index	Compares to
LAET (2021)	HNSW & IVF	BSL
SPANN (2021)	IVF	BSL
PatEE (2024)	IVF	LAET, BSL
PCE-IVFPQ (2023)	IVF	BSL
Auncel (2023)	IVF	LAET, BSL
LIRA (2025)	IVF	BSL
Quake (2025)	IVF	Auncel, SPANN, LAET
PiP (2025)	HSNW	BSL
DARTH (2025)	HNSW & IVF	LAET, BSL
Ada-ef (2025)	HNSW	DARTH, LAET, BSL
ConANN (2025)	IVF	BSL

Open problems and future directions

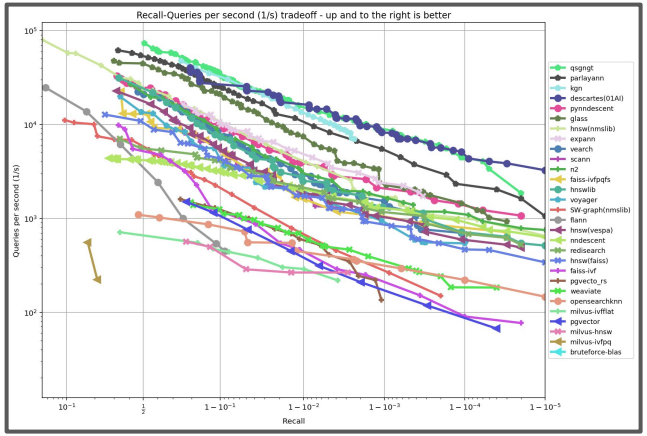
- **Comparative evaluation between early termination methods**
  - All of the methods compare to plain search with tuning (BSL)
  - Combinations?
- Extensions to **filtered vector search**
  - Difficult to capture the search behavior
- Extensions to **streaming**
  - Quake, Ada-ef already support that!

Method	Index	Compares to
LAET (2021)	HNSW & IVF	BSL
SPANN (2021)	IVF	BSL
PatEE (2024)	IVF	LAET, BSL
PCE-IVFPQ (2023)	IVF	BSL
Auncel (2023)	IVF	LAET, BSL
LIRA (2025)	IVF	BSL
Quake (2025)	IVF	Auncel, SPANN, LAET
PiP (2025)	HSNW	BSL
DARTH (2025)	HNSW & IVF	LAET, BSL
Ada-ef (2025)	HNSW	DARTH, LAET, BSL
ConANN (2025)	IVF	BSL

Vector Search Quality Evaluation

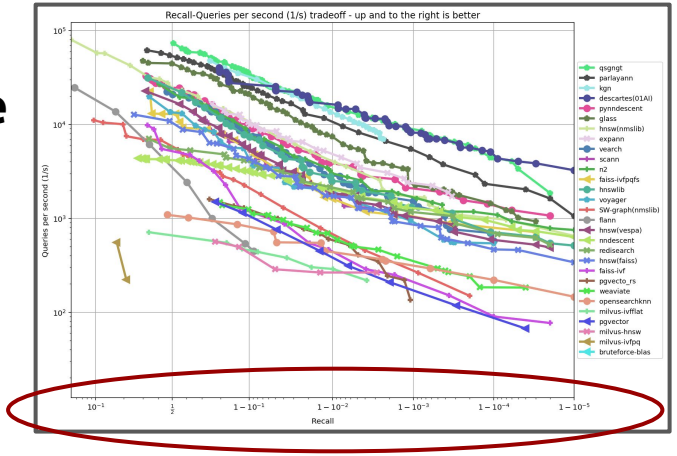
Pitfalls of vector search evaluation

- Vector search is mainly evaluated using **average recall** over a query workload
  - ANN benchmarks → **y axis: QPS, x axis: Avg. recall**



Pitfalls of vector search evaluation

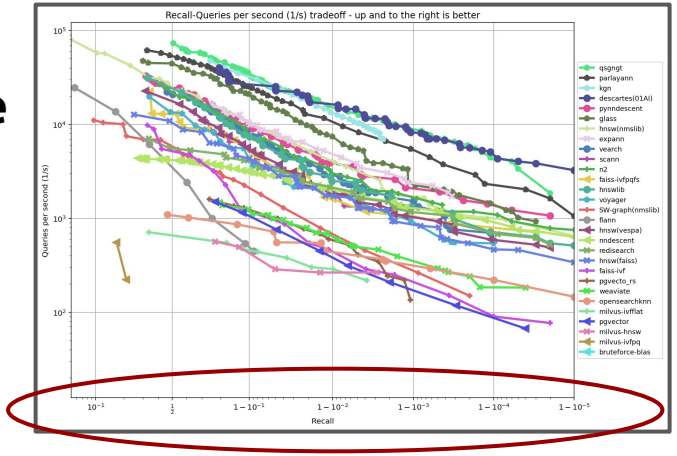
- Vector search is mainly evaluated using **average recall** over a query workload
  - ANN benchmarks → **y axis: QPS, x axis: Avg. recall**



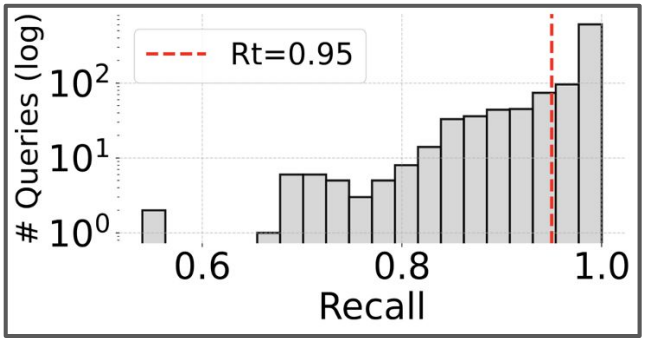
Pitfalls of vector search evaluation

- Vector search is mainly evaluated using **average recall** over a query workload
  - ANN benchmarks → **y axis: QPS, x axis: Avg. recall**

- Average recall can be **misleading**

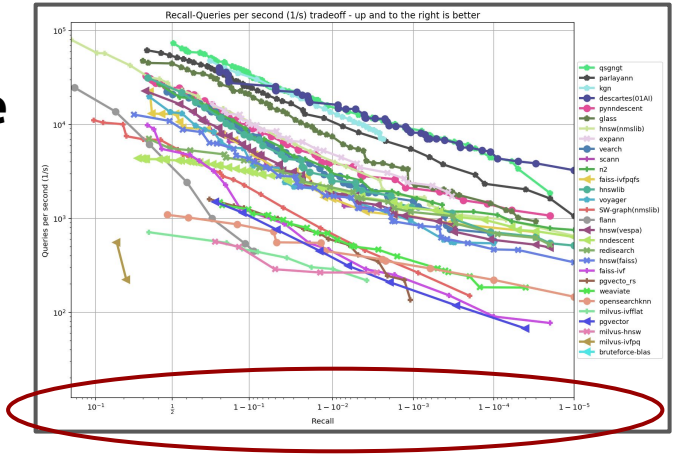


HNSW (ef=750), DEEP100M, 1000 queries, avg. recall=0.95



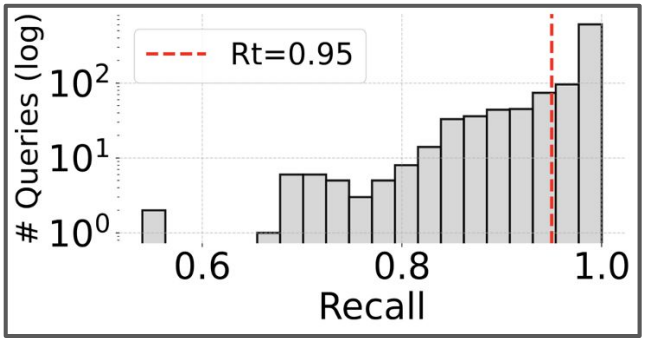
Pitfalls of vector search evaluation

- Vector search is mainly evaluated using **average recall** over a query workload
  - ANN benchmarks → **y axis: QPS, x axis: Avg. recall**



- Average recall can be **misleading**
  - Recall **distributions**
  - P95,99 percentiles
  - Worst errors
  - Queries under target

HNSW (ef=750), DEEP100M, 1000 queries, avg. recall=0.95

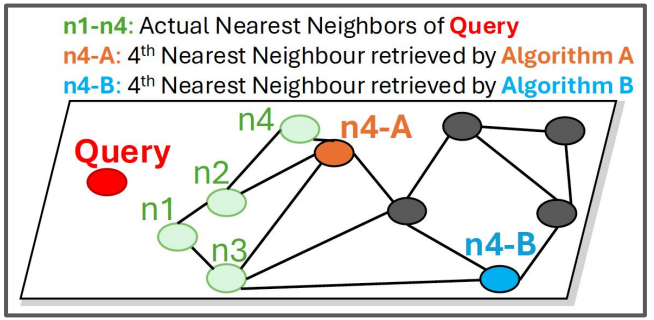


Beyond Recall

- Recall measure has drawbacks
  - It evaluates the result quality as binary decisions
  - **A retrieved vector is either a hit or not**
  - **Semantic search:** Non-gt results might still be relevant!

Beyond Recall

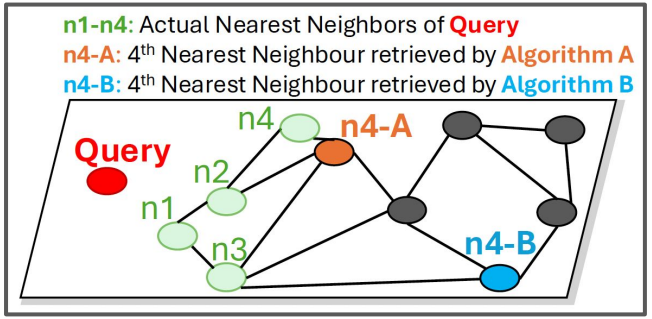
- Recall measure has drawbacks
  - It evaluates the result quality as binary decisions
  - **A retrieved vector is either a hit or not**
  - **Semantic search:** Non-gt results might still be relevant!



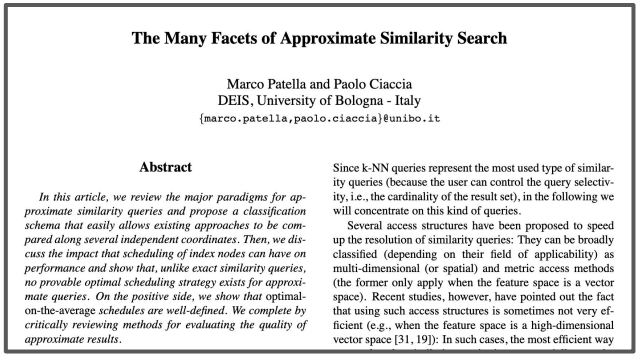
- **Alg. A** and **Alg. B** have same recall, but **Alg. A** has better semantic results!
  - **Relative Distance Error (RDE)** captures this!

Beyond Recall

- Recall measure has drawbacks
  - It evaluates the result quality as binary decisions
  - **A retrieved vector is either a hit or not**
  - **Semantic search:** Non-gt results might still be relevant!



- **Alg. A** and **Alg. B** have same recall, but **Alg. A** has better semantic results!
  - **Relative Distance Error (RDE)** captures this!
  - The IR community has proposed such measures many years ago!
    *SISAP **2008**, Patella et al.: The Many Facets of Approximate Similarity Search*



The Many Facets of Approximate Similarity Search

Marco Patella and Paolo Ciaccia
DEIS, University of Bologna - Italy
{marco.patella,paolo.ciaccia}@unibo.it

Abstract

In this article, we review the major paradigms for approximate similarity queries and propose a classification schema that easily allows existing approaches to be compared along several independent coordinates. Then, we discuss the impact that scheduling of index nodes can have on performance and show that, unlike exact similarity queries, no provable optimal scheduling strategy exists for approximate queries. On the positive side, we show that optimal-on-the-average *schedules are well-defined. We complete by critically reviewing methods for evaluating the quality of approximate results.*

Since k-NN queries represent the most used type of similarity queries (because the user can control the query selectivity, i.e., the cardinality of the result set), in the following we will concentrate on this kind of queries.

Several access structures have been proposed to speed up the resolution of similarity queries: They can be broadly classified (depending on their field of applicability) as multi-dimensional (or spatial) and metric access methods (the former only apply when the feature space is a vector space). Recent studies, however, have pointed out the fact that using such access structures is sometimes not very efficient (e.g., when the feature space is a high-dimensional vector space [31, 19]): In such cases, the most efficient way

We should consider additional measures beyond recall!

Remarks on query hardness

Query **hardness** corresponds to the effort needed to reach a recall target

Remarks on query hardness

Query **hardness** corresponds to the effort needed to reach a recall target

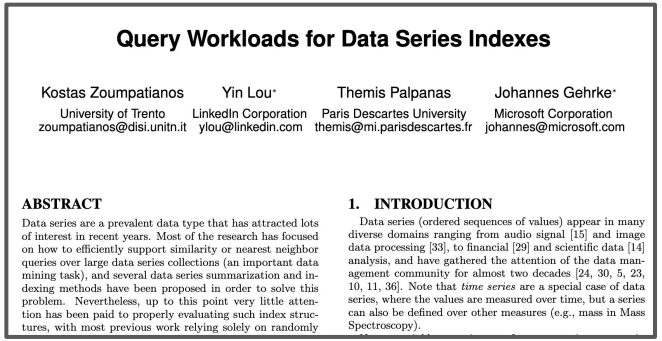
Measure	Description
LID	Intrinsic query hardness based on the distribution of GT nn distances for a given query. (higher is harder)
RC	Closeness of the k-th distance of the query to the average distance of the k-nn (lower is harder)
…	
Steiner-hardness	Difficulty of reaching the true nn of query in a graph index (higher is harder)

Wang et al. "Steiner-Hardness: A Query Hardness Measure for Graph-Based ANN Indexes." PVLDB 2025.
Ceccarello et al. "Evaluating and generating query workloads for high dimensional vector similarity search." SIGKDD 2025.
Zoumpatianos et al. "Query workloads for data series indexes." SIGKDD 2015.

Remarks on query hardness

Query **hardness** corresponds to the effort needed to reach a recall target

Measure	Description
LID	Intrinsic query hardness based on the distribution of GT nn distances for a given query. (higher is harder)
RC	Closeness of the k-th distance of the query to the average distance of the k-nn (lower is harder)
…	
Steiner-hardness	Difficulty of reaching the true nn of query in a graph index (higher is harder)



Query Workloads for Data Series Indexes

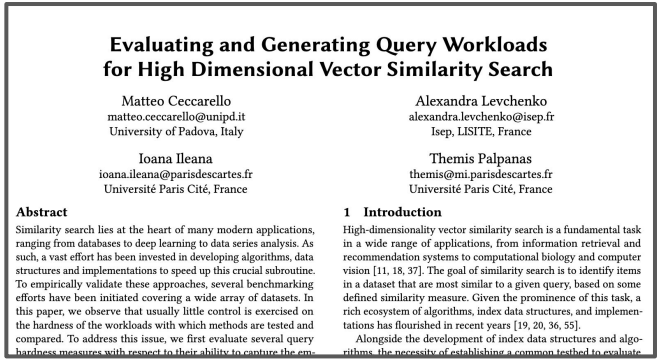
Kostas Zoumpatianos — University of Trento — zoumpatianos@disi.unitn.it
Yin Lou* — LinkedIn Corporation — ylou@linkedin.com
Themis Palpanas — Paris Descartes University — themis@mi.parisdescartes.fr
Johannes Gehrke* — Microsoft Corporation — johannes@microsoft.com

ABSTRACT

Data series are a prevalent data type that has attracted lots of interest in recent years. Most of the research has focused on how to efficiently support similarity or nearest neighbor queries over large data series collections (an important data mining task), and several data series summarization and indexing methods have been proposed in order to solve this problem. Nevertheless, up to this point very little attention has been paid to properly evaluating such index structures, with most previous work relying solely on randomly

1. INTRODUCTION

Data series (ordered sequences of values) appear in many diverse domains ranging from audio signal [15] and image data processing [33], to financial [29] and scientific data [14] analysis, and have gathered the attention of the data management community for almost two decades [24, 30, 5, 23, 10, 11, 36]. Note that *time series* are a special case of data series, where the values are measured over time, but a series can also be defined over other measures (e.g., mass in Mass Spectroscopy).



Evaluating and Generating Query Workloads for High Dimensional Vector Similarity Search

Matteo Ceccarello — matteo.ceccarello@unipd.it — University of Padova, Italy
Alexandra Levchenko — alexandra.levchenko@isep.fr — Isep, LISITE, France
Ioana Ileana — ioana.ileana@parisdescartes.fr — Université Paris Cité, France
Themis Palpanas — themis@mi.parisdescartes.fr — Université Paris Cité, France

Abstract

Similarity search lies at the heart of many modern applications, ranging from databases to deep learning to data series analysis. As such, a vast effort has been invested in developing algorithms, data structures and implementations to speed up this crucial subroutine. To empirically validate these approaches, several benchmarking efforts have been initiated covering a wide array of datasets. In this paper, we observe that usually little control is exercised on the hardness of the workloads with which methods are tested and compared. To address this issue, we first evaluate several query hardness measures with respect to their ability to capture the em-

1 Introduction

High-dimensionality vector similarity search is a fundamental task in a wide range of applications, from information retrieval and recommendation systems to computational biology and computer vision [11, 18, 37]. The goal of similarity search is to identify items in a dataset that are most similar to a given query, based on some defined similarity measure. Given the prominence of this task, a rich ecosystem of algorithms, index data structures, and implementations has flourished in recent years [19, 20, 36, 55].

Alongside the development of index data structures and algorithms, the necessity of establishing a common testbed to evaluate

Towards better evaluation suites

Measure	Description	Range	Improvement
Recall	proportion of GTs retrieved over k	0 → 1	Higher is better

Towards better evaluation suites

Measure	Description	Range	Improvement
Recall	proportion of GTs retrieved over k	$0 \rightarrow 1$	Higher is better
RDE	deviation of retrieved distances to GT distances	$\infty \rightarrow 0$	Lower is better

Towards better evaluation suites

Measure	Description	Range	Improvement
Recall	proportion of GTs retrieved over k	0 → 1	Higher is better
RDE	deviation of retrieved distances to GT distances	∞ → 0	Lower is better
RQUT	Ratio of the Queries that fall Under the recall Target	1 → 0	Lower is better

Towards better evaluation suites

Measure	Description	Range	Improvement
Recall	proportion of GTs retrieved over k	0 → 1	Higher is better
RDE	deviation of retrieved distances to GT distances	∞ → 0	Lower is better
RQUT	Ratio of the Queries that fall Under the recall Target	1 → 0	Lower is better
Worst Errors	recall deviation of the queries that fell under the recall target	1 → 0	Lower is better

Towards better evaluation suites

Measure	Description	Range	Improvement
Recall	proportion of GTs retrieved over k	0 → 1	Higher is better
RDE	deviation of retrieved distances to GT distances	∞ → 0	Lower is better
RQUT	Ratio of the Queries that fall Under the recall Target	1 → 0	Lower is better
Worst Errors	recall deviation of the queries that fell under the recall target	1 → 0	Lower is better
Query Hardness	Hardness measures of the queries of the workload (LID, RC, Steiner, …)	-	-

Concluding Remarks

- Parallel and Distributed Vector Search
  - Still no solutions that probe only one shard
  - Still communication required. Advanced components that leverage this overhead are expensive
  - Hybrid search as future direction

- Streaming
  - No single winner, each approach tackles a different problem
  - Index quality under updates is an open problem
  - Lack of a standard benchmarks

- Early termination enables adaptive vector search
  - Comparative evaluations, filtered vector search, streaming

- Vector search quality comparison should go beyond average recall

Acknowledgements

We thank the authors for providing us the material and slides of their work.

References [1/4]

[1] Azizi et al. ELPIS: Graph-Based Similarity Search for Scalable Data Science. VLDB 2023

[2] Peng et al. iQAN: Fast and Accurate Vector Search with Efficient Intra-Query Parallelism on Multi-Core Architectures. PPoPP 2023.

[3] Li et al. Scalable Graph Indexing using GPUs for Approximate Nearest Neighbor Search. ACM Manag. Data 2025.

[4] Mohoney et al. Quake: Adaptive Indexing for Vector Search. OSDI 2025.

[5] Dobson et al. ParlayANN: Scalable and Deterministic Parallel Graph-Based Approximate Nearest Neighbor Search. PPoPP 2023.

[6] Douze et al. The Faiss Library. arXive 2025.

[7] Dang et al. Passing the Baton: High Throughput Distributed Disk-Based

Vector Search with BatANN. arXiv 2025.

[8] Xu et al. Scalable Distributed Vector Search via Accuracy Preserving Index Construction. arXiv 2025.

[9] Zhang et al. Fast, Approximate Vector Queries on Very Large Unstructured Datasets. NSDI 2023.

[10] Deng et al. Pyramid: A General Framework for Distributed Similarity Search on Large-Scale Datasets. IEEE Big Data 2019.

[11] Adams et al. DistributedANN: Efficient Scaling of a Single DiskANN Graph Across Thousands of Computers. arXiv 2025.

[12] Xu et al. Harmony: A Scalable Distributed Vector Database for High-Throughput Approximate Nearest Neighbor Search. ACM Manag. Data 2025.

[13] Zhi et al. Towards Efficient and Scalable Distributed Vector Search with RDMA. arXiv 2025.

[14] Gottesbüren et al. Unleashing Graph Partitioning for Large-Scale Nearest Neighbor Search. VLDB 2025.

References [2/4]

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

[16] Xu et al. "In-place Updates of a Graph Index for Streaming Approximate Nearest Neighbor Search" arXiv 2025

[17] Zhang et al. "CleANN: Efficient Full Dynamism in Graph-based Approximate Nearest Neighbor Search" arXiv 2025

[18] Xu et al. "SPFresh: Incremental In-Place Update for Billion-Scale Vector Search" SOSP 2023

[19] Mohoney et al. "Incremental IVF Index Maintenance for Streaming Vector Search" arXiv 2024

[20] Mohoney et al. "Quake: Adaptive Indexing for Vector Search" OSDI 2025

[21] Lu et al. "VectraFlow: Integrating Vectors into Stream Processing" CIDR 2025

[22] Sun et al. "A Real-Time Adaptive Multi-Stream GPU System for Online Approximate Nearest Neighborhood Search" CIKM 2024

[23] Gong et al. "VStream: A Distributed Streaming Vector Search System" VLDB 2025

[24] Hezel et al. "Fast Approximate Nearest Neighbor Search with a Dynamic Exploration Graph using Continuous Refinement" arXiv 2023

[25] Aguerrebere et al. "Similarity Search in the Blink of an Eye with Compressed Indices" VLDB 2023

References [3/4]

[26] Chen et al. "Spann: Highly-efficient billion-scale approximate nearest neighborhood search." NeurIPS 2021

[27] Horchidan et al. “ConANN: Conformal Approximate Nearest Neighbor Search” PVLDB 2026.

[28] Zhang et al. "Distribution-Aware Exploration for Adaptive HNSW Search." SIGMOD 2026.

[29] Li et al. "Learned adaptive early termination for approximate nearest neighbor search." SIGMOD 2021.

[30] Chatzakis et al. "Darth: Declarative recall through early termination for approximate nearest neighbor search." SIGMOD 2026.

[31] Zhang et al. “Fast, Approximate Vector Queries on Very Large Unstructured Datasets” NSDI 2023.

[32] Zheng et al. "Learned probing cardinality estimation for high-dimensional approximate NN search." ICDE 2023.

[33] Zeng et al. "Lira: A learning-based query-aware partition framework for large-scale ann search." WebConf 2025.

[34] Teofili et al. "Patience in proximity: a simple early termination strategy for HNSW graph traversal in approximate K-nearest neighbor search" ECIR 2025.

[35] Busolin et al. "Early exit strategies for approximate k-nn search in dense retrieval." ICIKM 2024.

[36] Patella et al. “The many facets of approximate similarity search” SISAP 2008.

References [4/4]

[37] Wang et al. "Steiner-Hardness: A Query Hardness Measure for Graph-Based ANN Indexes." PVLDB 2025.

[38] Ceccarello et al. "Evaluating and generating query workloads for high dimensional vector similarity search." SIGKDD 2025.

[39] Zoumpatianos et al. "Query workloads for data series indexes." SIGKDD 2015.

[40] Echihabi et al. “New Trends in High-D Vector Similarity Search: AI-driven, Progressive, and Distributed” PVLDB 2021.

[41] Wei et al. “DET-LSH: A Locality-Sensitive Hashing Scheme with Dynamic Encoding Tree for Approximate Nearest Neighbor Search” PVLDB 2024

[42] Azizi et al. “ELPIS: Graph-Based Similarity Search for Scalable Data Science” PVLDB 2023

[43] Echihabi et al. “ProS: data series progressive k-NN similarity search and classification with probabilistic quality guarantees” VLDBJ 2023

[44] Malkov et al. "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs." PAMI 2018.

[45] Fu et al. "Fast approximate nearest neighbor search with the navigating spreading-out graph." PVLDB 2017.

[46] Subramanya et al. "Diskann: Fast accurate billion-point nearest neighbor search on a single node." NeurIPS 2019.

[47] Douze, Matthijs et al. "The faiss library." IEEE Big Data 2025.

[48] Google Research. “Announcing ScaNN: Efficient Vector Similarity Search.” Google Research Blog 2020.

Thank you!

Questions?