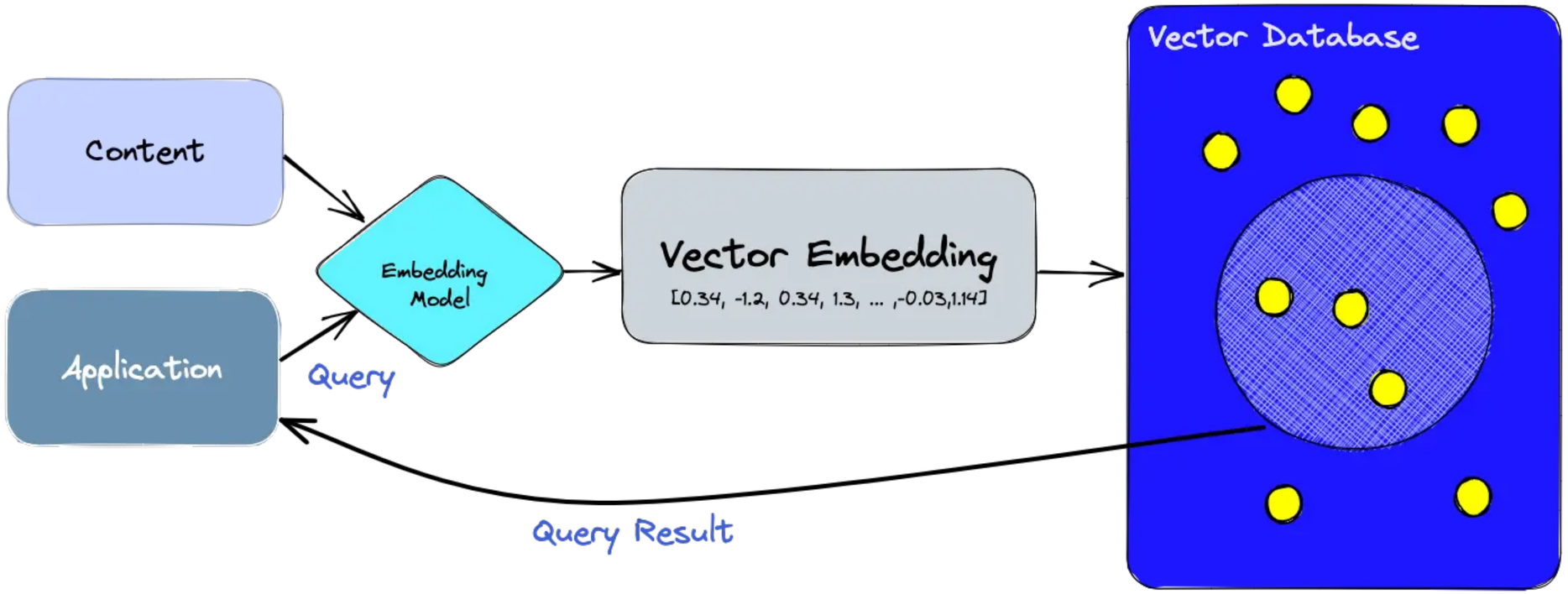
CS 744: Vector Databases

Jason Mohoney

Spring 2025

What is a Vector Database?

Database that can store high-dimensional vectors and process similarity search (nearest-neighbor) queries



The Venture Capital Zoo

Pinecone

$100M Series B (2023)

ROCKSET

Acquired by **OpenAI** for 100M+ (2024)

zilliz

$60M Series C (2022)

Weaviate

$50M Series B (2024)

drant

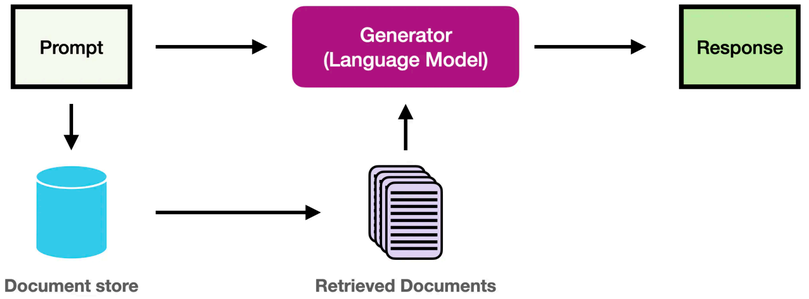
$28M Series A (2024)

Vespa

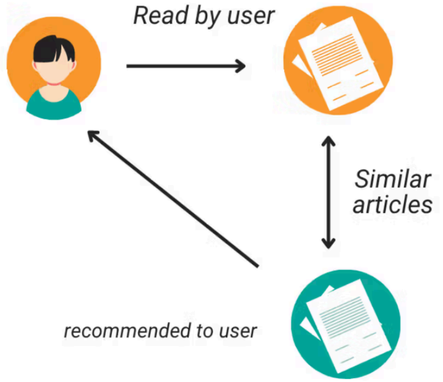
$31M Series A (2023)

Applications

Retrieval-Augmented Generation

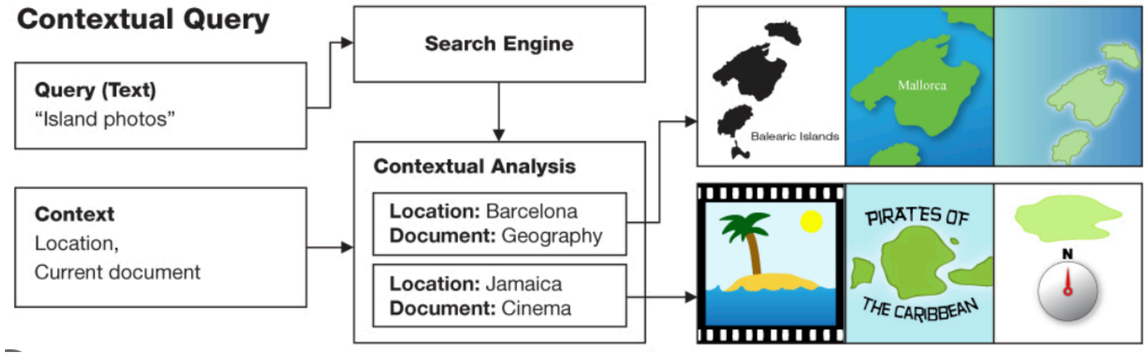


Recommendation



Multi-modal Search

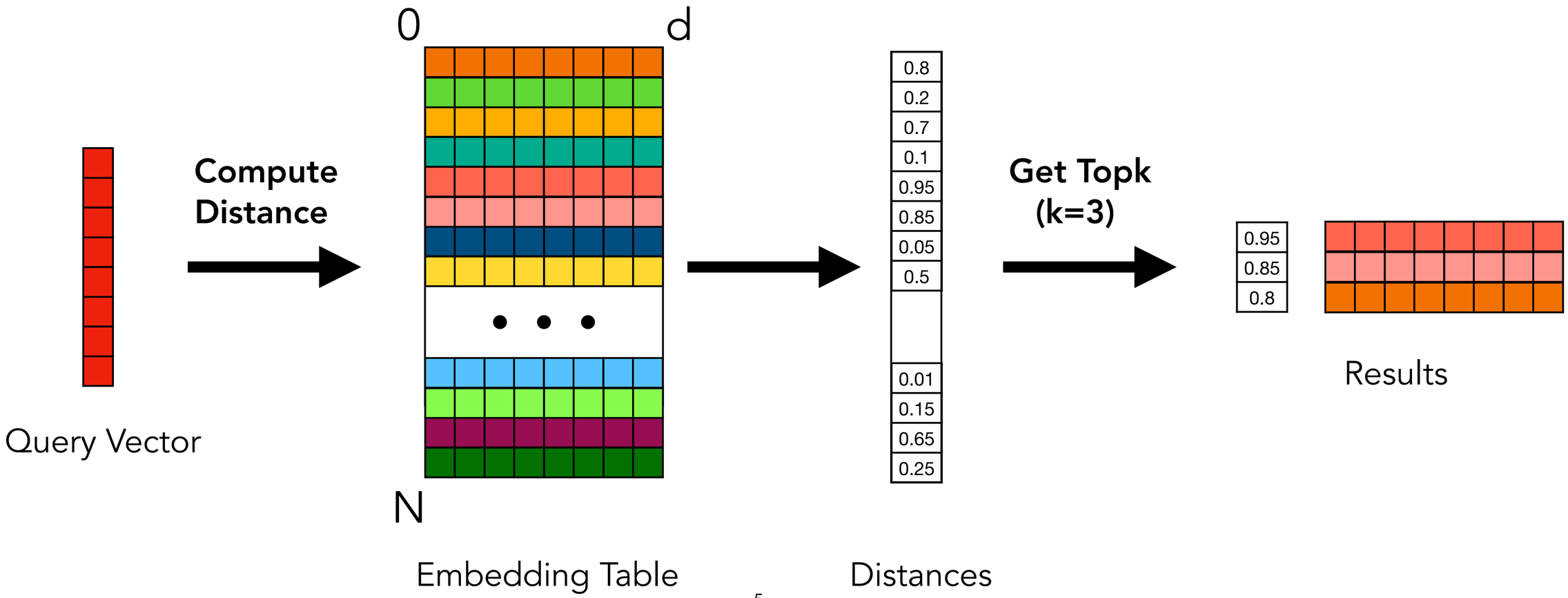
- Text
- Images
- Audio



Vector Search

(a.k.a Similarity Search, KNN/ANN Search)

Goal: Get top-k nearest vectors to query vector by distance



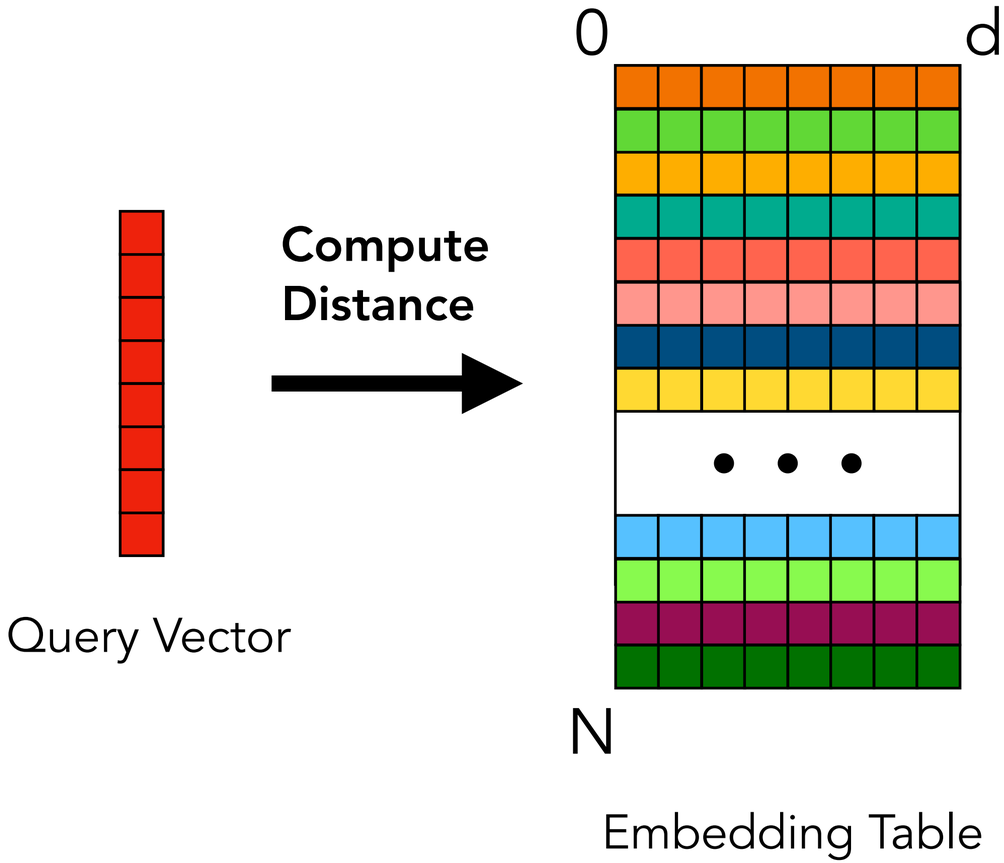
Systems Challenges

Scale

N can be in the billions, d can be in the thousands

Memory-Bound

Performance limited by how fast you can scan the embedding table



The Need for Vector Indexes

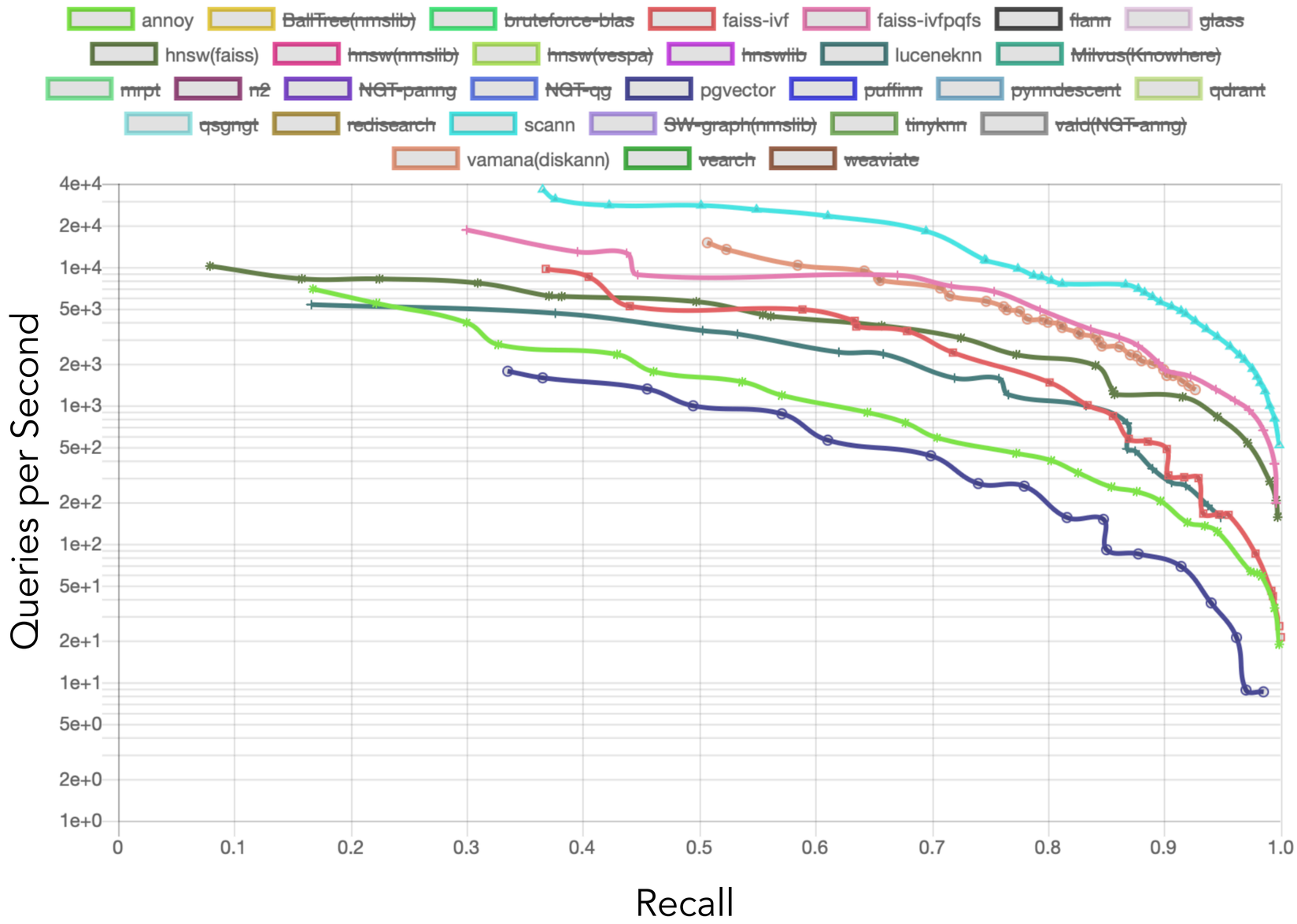
Brute-force scanning of all N vectors is not feasible for large N

Vector indexes allow you to scan a subset of N to find the topk

The catch: results are approximate, the true top-k might not be returned

Runtime vs. Recall Tradeoff

annbenchmarks.com: Glove-1M with k = 10

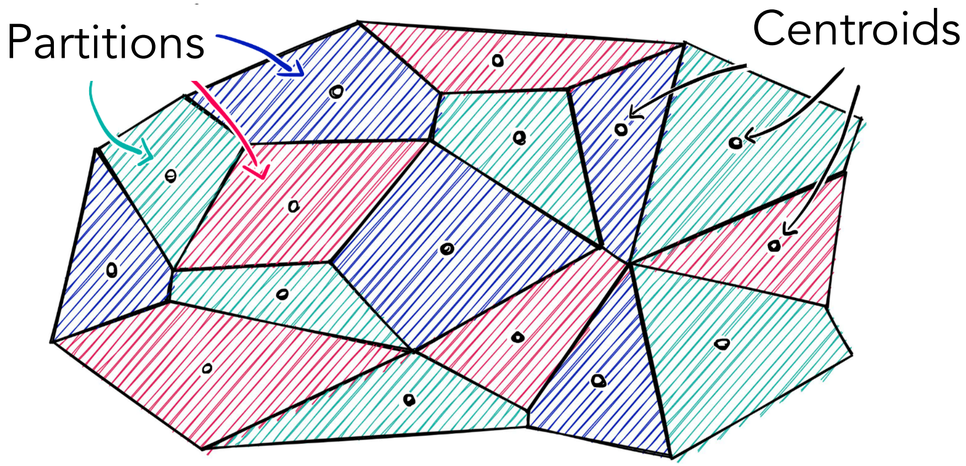


Partitioned Indexes

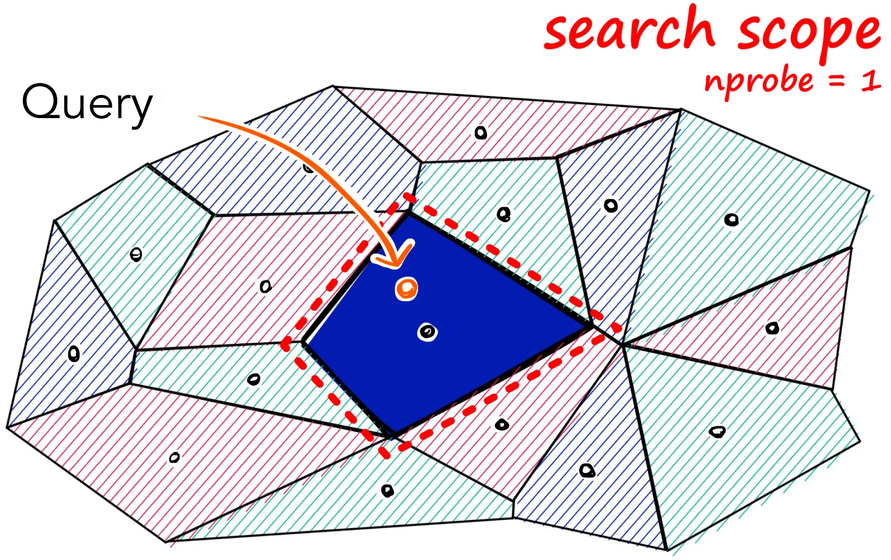
Partition vectors such that queries only need to scan a subset of partitions.

Build: Cluster (e.g. k-means) vectors into nlist partitons

Search: Scan nprobe partitions to return approximate topk



Partitioned Index



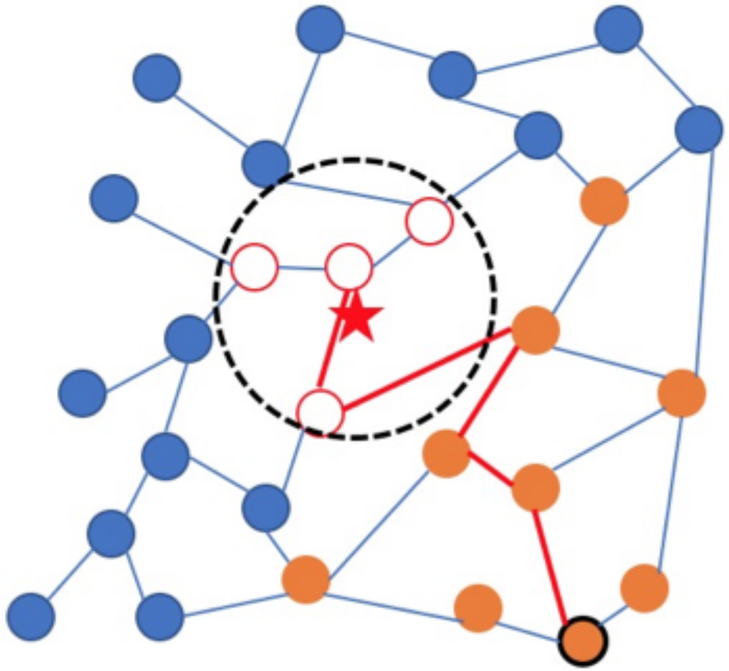
Searching a Partitioned Index

Graph Indexes

Build a proximity graph over the vectors.
Traverse graph to obtain top-k

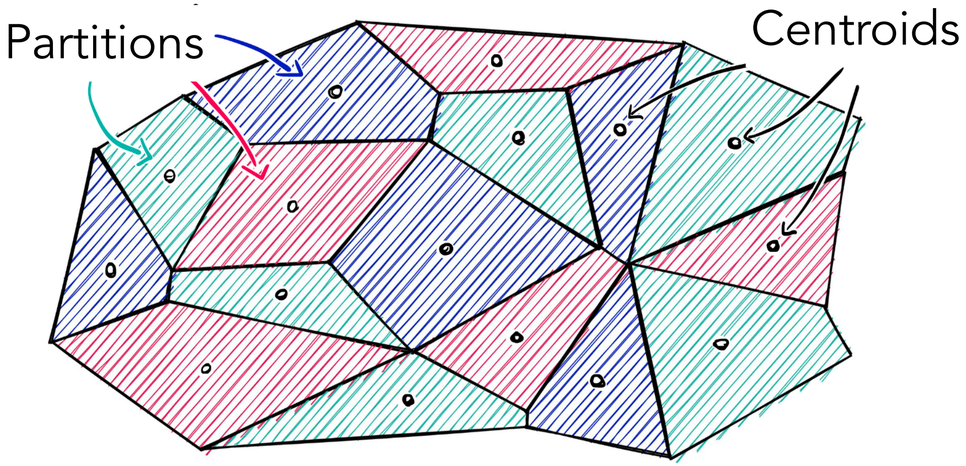
Build: Create a R-degree proximity graph by connecting close vectors (nodes)

Search: Perform Greedy Traversal of the graph to find the top-k



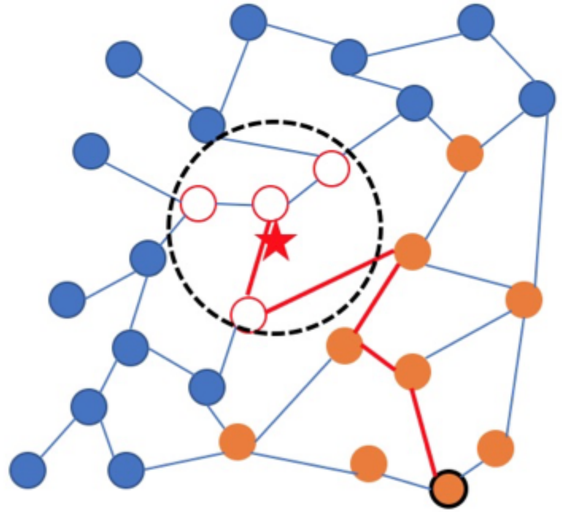
Query Processing with a Graph Index

Partitioned vs. Graph Indexes



Pros

Cons



Pros

Cons

Other Techniques

Vector Compression:

Convert d-dimensional vectors into d’-dimensional vectors: d’ < d

Data-type quantization:

Convert from float32s to float16s, float8s, binary, etc.

Open-Source Vector Indexing Systems

Faiss (Facebook)

High performance implementation of many vector search algorithms/indexes. Used broadly by vector databases

SPANN (Microsoft)

Partitioned vector index with a graph index over the centroids

SCANN (Google)

Hierarchical partitioned vector index

DiskANN (Microsoft)

Graph index that supports disk-based indexing and search

Hybrid Queries

Combine vector similarity search with filters

Query Embedding

Ex Machina

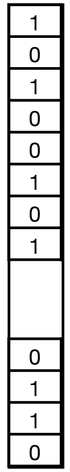
Example Query

Recommend me a **Sci-FI movie** similar to **Ex Machina**

Filter

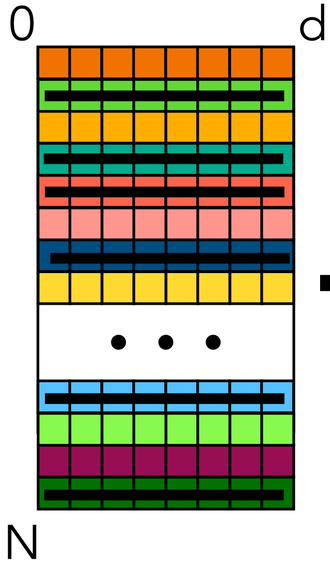
```
type == "movie"
        &
genre =="sci-fi"
```

Get entries that pass filter

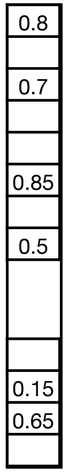


Filter Bitmap

Compute filtered distance

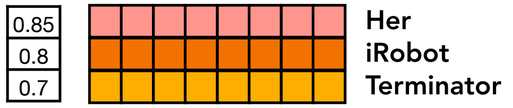


Embedding Table



Distances

Get Topk (k=3)



Results

Vector Search: Practical Challenges

A. How to handle hybrid queries?

B. How to handle updates?

C. How meet recall or latency targets?

D. How to deploy on cloud hardware?

a) GPUs

b) Block Storage: SSDs, S3

c) Multi-Node & NUMA

Analytic-DBV (Alibaba, VLDB ’20)

One of the first descriptions of a cloud-based vector database

Main contributions:

1. Ability to natively execute hybrid queries
2. Ability to handle updates
3. Shows how SQL can be extended to support similarity search
4. A new type of partitioned vector index (VGPQ)

Analytic-DBV: Architecture

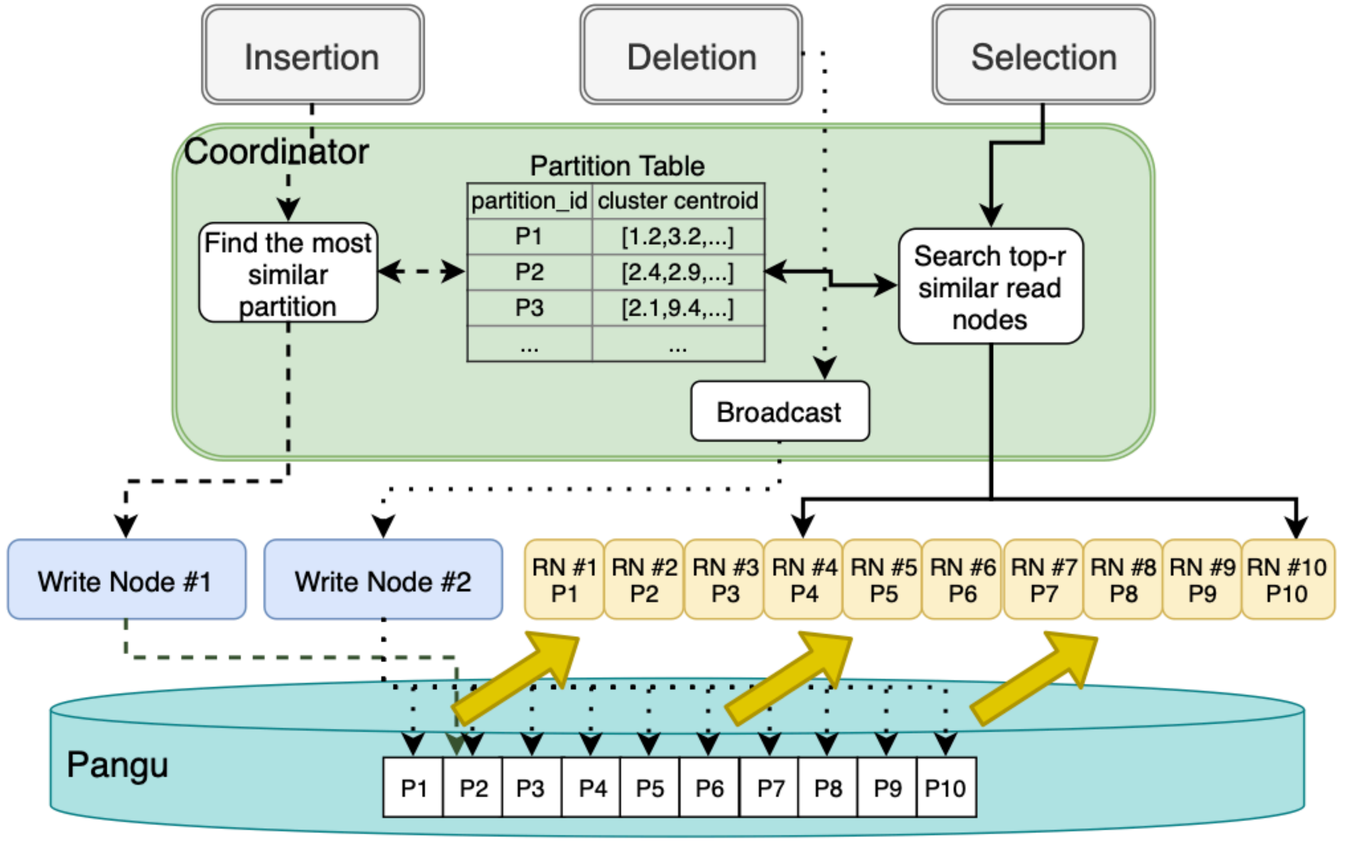


Figure 5: Clustering-based partition pruning

Analytic-DBV: Hybrid Query Processing

Uses both b-trees (for filters) and vector indexes to process hybrid queries.

Basic query optimizer: Selects between multiple query plans depending on the filter selectivity

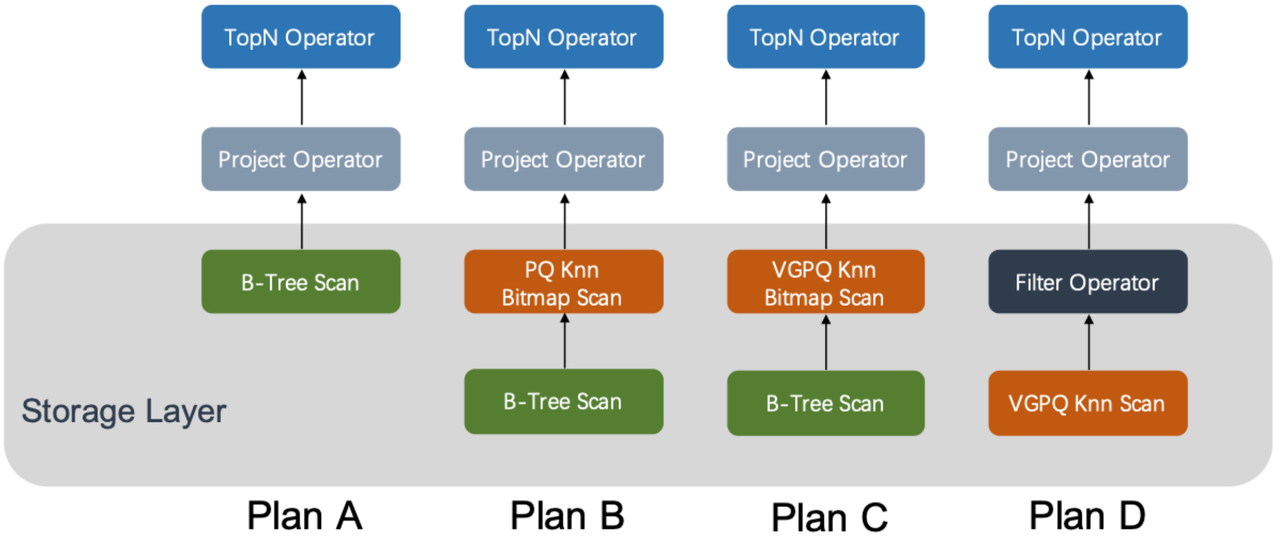


Figure 9: Physical plans of $Q2$

Analytic-DBV: Handling Updates

Maintains a secondary index for recent updates. Periodically flushing and rebuilding the main index

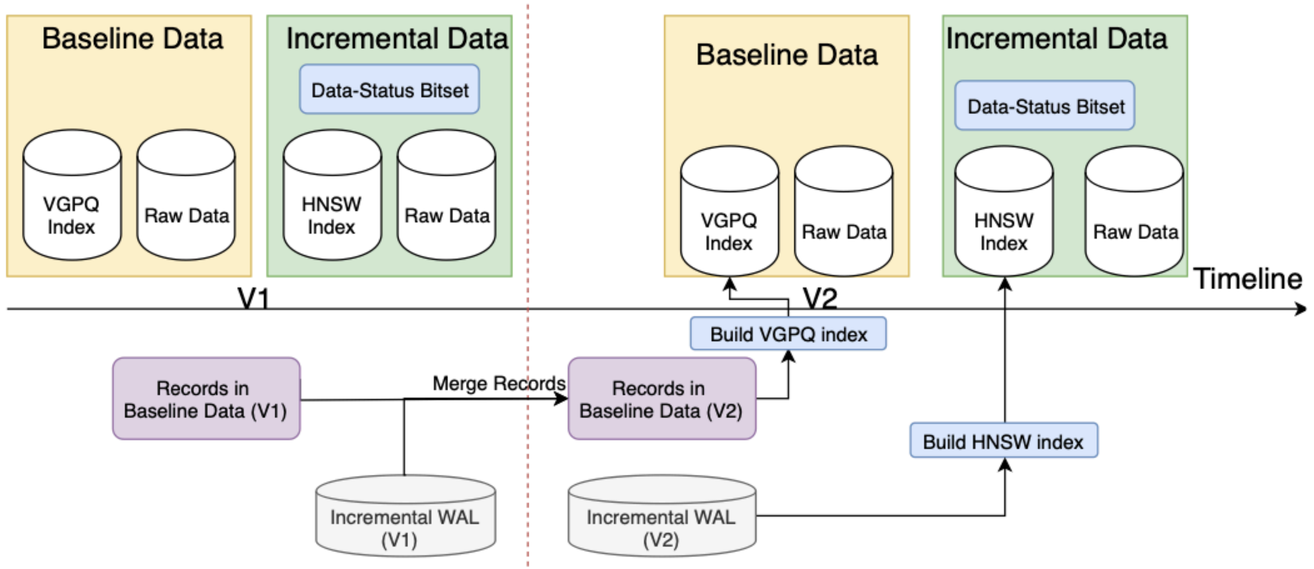


Figure 4: Baseline Data and Incremental Data in ADBV

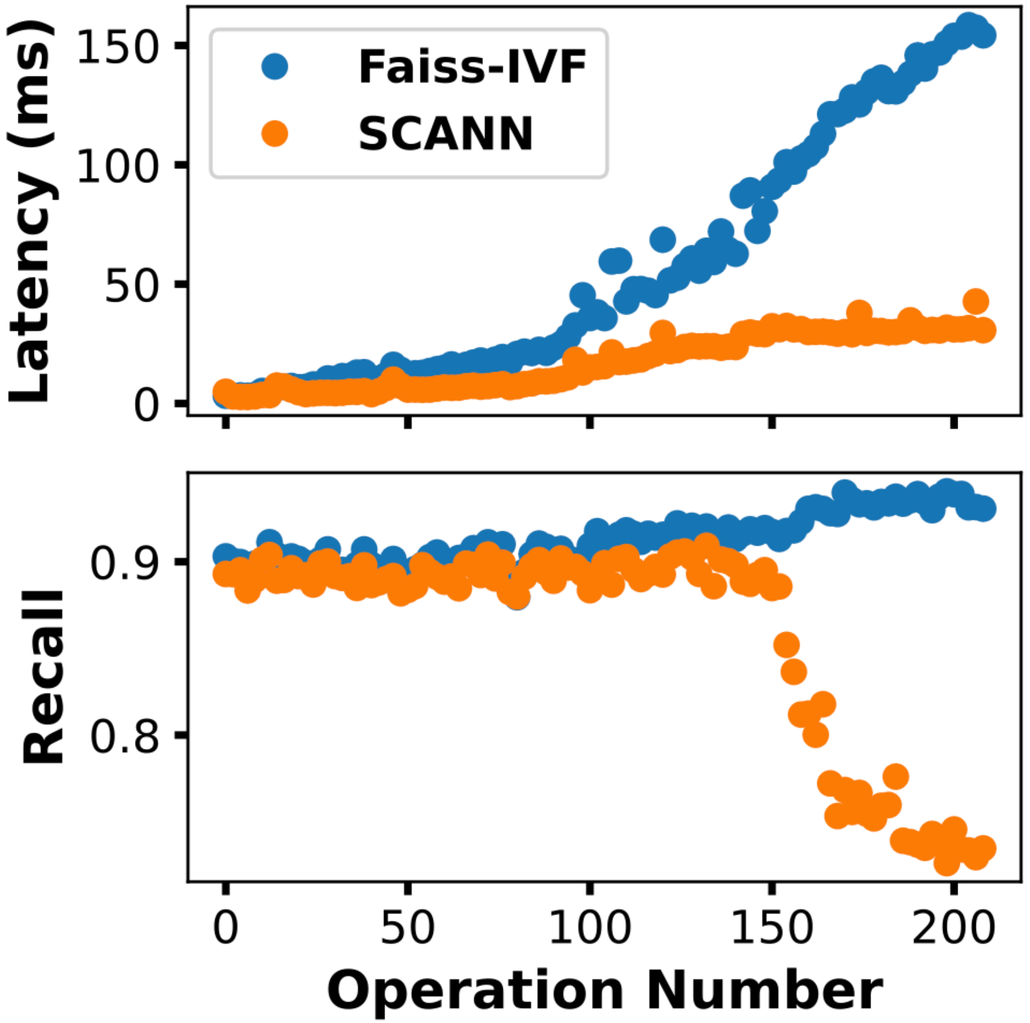
Quake: Motivation

Updates **degrade the latency and recall** of partitioned vector search indexes

Why?

1. Partitions grow and become imbalanced.
   E.g. Faiss-IVF

2. Existing maintenance methods require retuning the system to maintain recall after maintenance. E.g. SCANN



Practical Challenges Addressed by Quake

A. How to handle hybrid queries?

B. **How to handle updates?**

C. **How meet recall or latency targets?**

D. How to deploy on cloud hardware?

a) GPUs

b) Block Storage: SSDs, S3

c) Multi-Node & **NUMA**

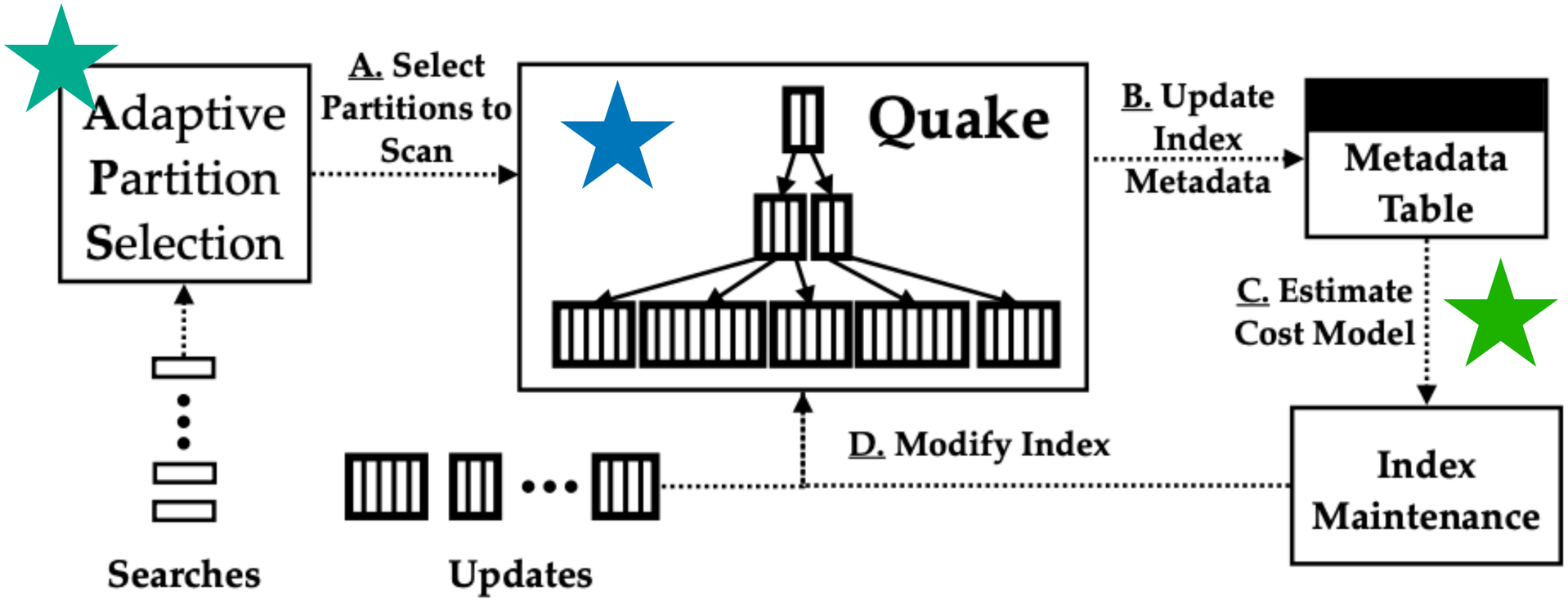
Quake (OSDI ’25): Main Ideas

Quake is a multi-level partitioned index with three main components:

Cost model for query latency based on *index access patterns* to drive incremental maintenance

Adaptive partition selection to automatically set the number of partitions to scan

NUMA-aware query processing to maximize memory bandwidth



Highlight: Quake meets recall targets without tuning

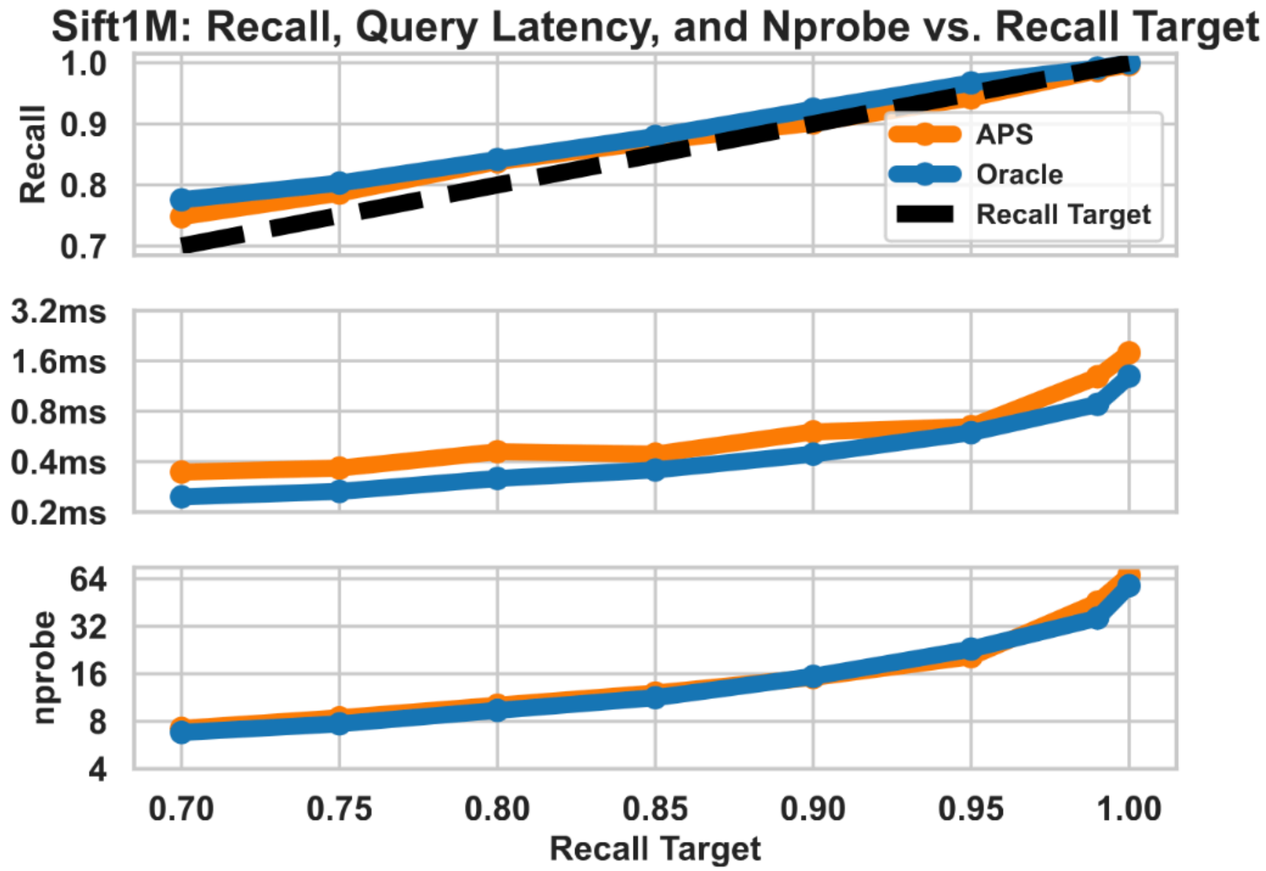


Figure 6: APS vs. Oracle on SIFT1M. APS achieves similar recall and latency w.r.t. an oracle tuned for each recall target.

CS 744 projects extending Quake

A. **How to handle hybrid queries?**

B. How to handle updates?

C. How meet recall or latency targets?

D. **How to deploy on cloud hardware?**

a) **GPUs**

b) **Block Storage: SSDs, S3**

c) **Multi-Node** & NUMA

Github

Discussion

Consider an application where **large batches** of ***hybrid queries*** are processed using a **partitioned index**.

1. Come up with an example application that uses batch processing.

   a) Are there notable properties of your queries? E.g. selectivity, duplication

2. What optimizations can be done to improve throughput?

https://forms.gle/uAGTqQFQLnADU2oYA

Discussion

1. Come up with an example application that uses batch processing.

   a) Are there notable properties of your queries? E.g. selectivity, duplication

2. What optimizations can be done to improve throughput?

Next Steps

- Next week: recommendation model training (Shivaram back!)
- Project check-ins **due April 7th**