Vector Database Systems

Shan-Hung Wu and DataLab
CS, NTHU

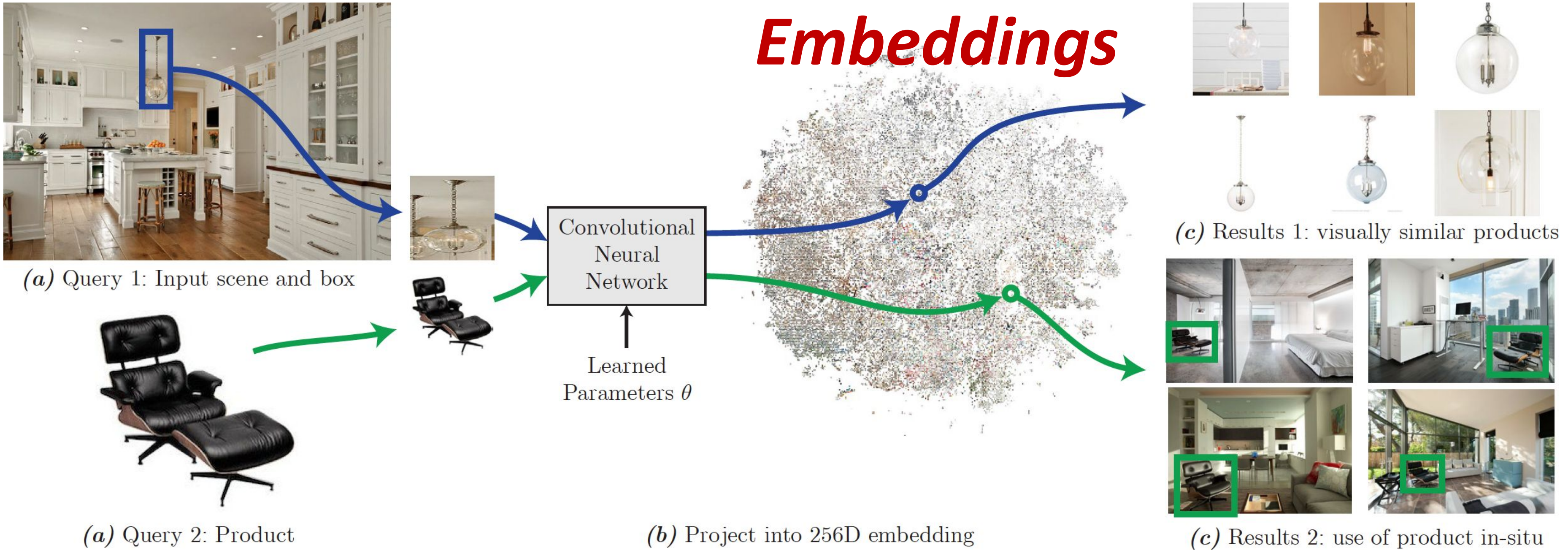
Outline

- Why Vector DBMS?
- AKNN Search Algorithms
- Challenges at System-level
- Case study: PASE (System R-like)
  – Data model & Query Format
  – Index Building & Update
  – Planning & Cost Estimation
- Case study: Milvus (purpose-built)
  – Storage & Consistency Model
  – Computing & Threads
  – Query Algorithms

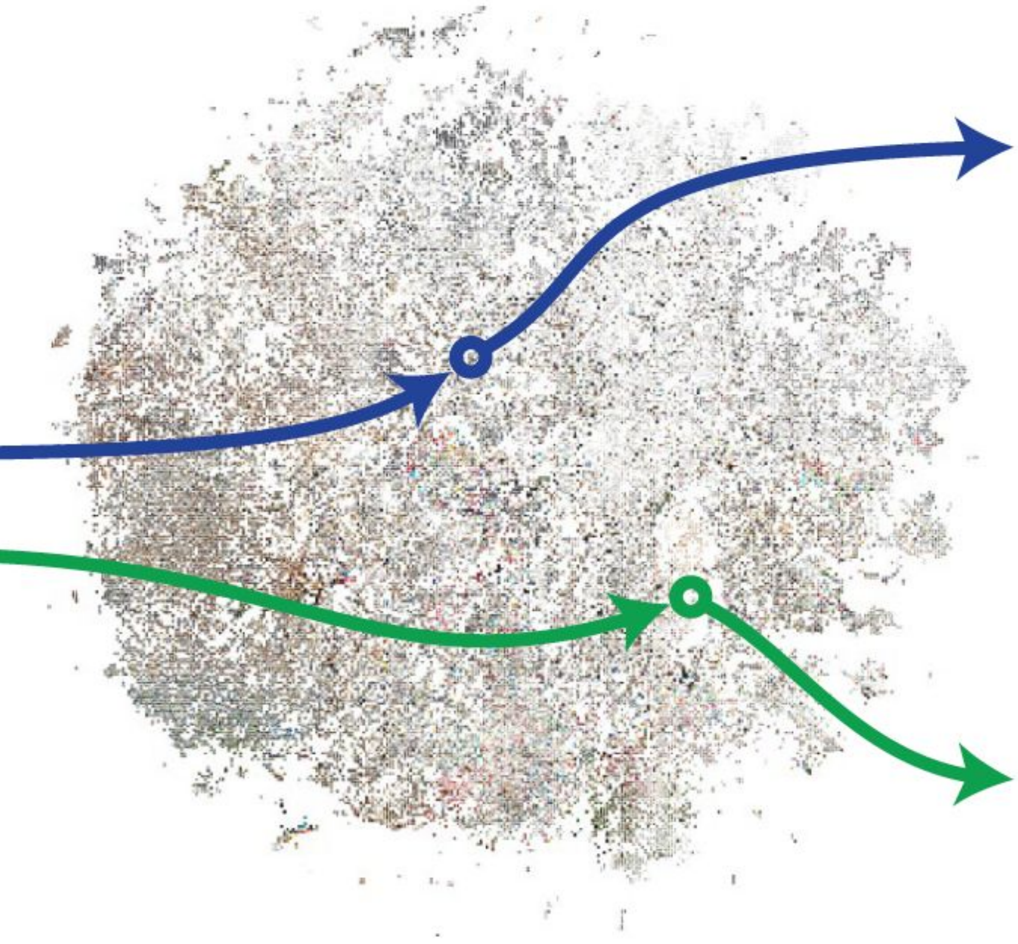
Outline

- Why Vector DBMS?
- AKNN Search Algorithms
- Challenges at System-level
- Case study: PASE (System R-like)
  - Data model & Query Format
  - Index Building & Update
  - Planning & Cost Estimation
- Case study: Milvus (purpose-built)
  - Storage & Consistency Model
  - Computing & Threads
  - Query Algorithms

The Emerge of AI & Embeddings



- Used by search engines, recommender systems, personalized ads, etc.



How to store & search billions of embeddings?

Outline

- Why Vector DBMS?
- AKNN Search Algorithms
- Challenges at System-level
- Case study: PASE (System R-like)
  - Data model & Query Format
  - Index Building & Update
  - Planning & Cost Estimation
- Case study: Milvus (purpose-built)
  - Storage & Consistency Model
  - Computing & Threads
  - Query Algorithms

Approximate K Nearest Neighbor (AKNN) Search

- Given a query vector q, find k vectors $V=\{v_1, v_2, \ldots, v_k\}$ in storage that are approximately nearest to q
- Distance measure?
  - Euclidian distance, cosine similarity, etc.
- The higher ***recall*** the better
  - Let ground truth: V^*
  - Recall = $| V \cap V^* | / | V^* |$

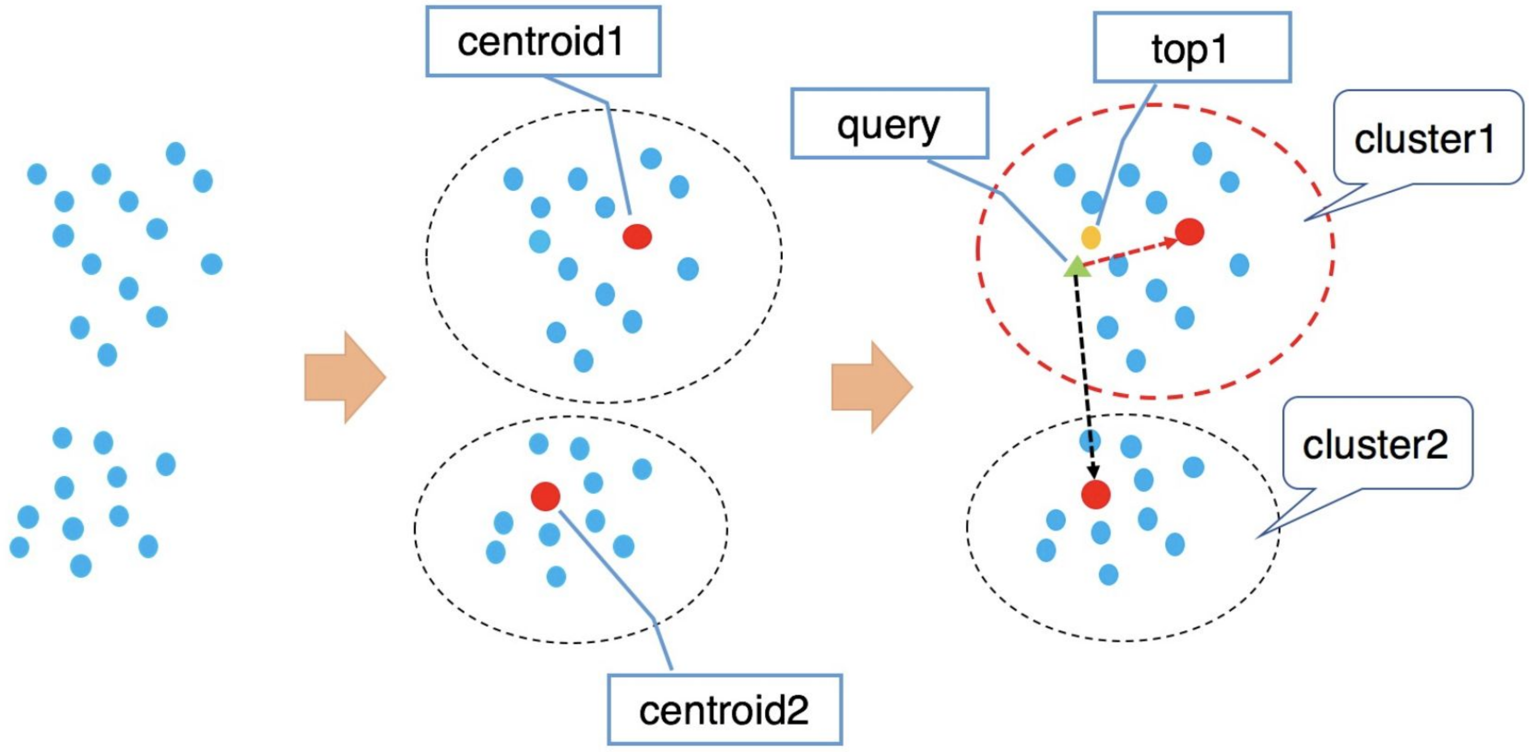
AKNN Algorithms

- Tree-based: KD-tree, R-tree
- Quantization-based: IVF_FLAT/SQ8/PQ
- Graph-based: HNSW, NSG, SSG
- Locality sensitive hashing (LSH)

AKNN Algorithms

- Tree-based: KD-tree, R-tree
  - Runs slowly on high-dimensional data
- ***Quantization-based***: IVF_FLAT/SQ8/PQ
  - ***High recall***, ***codebooks are update-insensitive***
- Graph-based: HNSW, NSG, SSG
  - High recall, graph take time/space to maintain
- Locality sensitive hashing (LSH)
  - Low recall

IVF_FLAT/SQ8/PQ



- Search in each cluster: brut force (FLAT) vs. compressed (SQ8) vs. quantization of subvectors (PQ)

Outline

- Why Vector DBMS?
- AKNN Search Algorithms
- Challenges at System-level
- Case study: PASE (System R-like)
  - Data model & Query Format
  - Index Building & Update
  - Planning & Cost Estimation
- Case study: Milvus (purpose-built)
  - Storage & Consistency Model
  - Computing & Threads
  - Query Algorithms

AKNN Libraries from AI Community

- Facebook **Faiss**, Microsoft **SPTAG**, Spotify **Annoy**, etc.
  - Implement various AKNN algorithms

- Pros: computation optimized
  - Support SIMD instructions (SSE, AVX, AVX2)
  - Faiss even supports GPU acceleration

AKNN Libraries from AI Community

- Facebook **Faiss**, Microsoft **SPTAG**, Spotify **Annoy**, etc.
  – Implement various AKNN algorithms

- ***Cons***:
  – Assume memory storage only
  – No support for dynamic data (updates/deletes)
  – No attribute filtering (e.g., “100 < price < 200”)

Outline

- Why Vector DBMS?
- AKNN Search Algorithms
- Challenges at System-level
- Case study: PASE (System R-like)
  - Data model & Query Format
  - Index Building & Update
  - Planning & Cost Estimation
- Case study: Milvus (purpose-built)
  - Storage & Consistency Model
  - Computing & Threads
  - Query Algorithms

PASE

- "PostgreSQL Ultra-High-Dimensional Approximate Nearest Neighbor Search Extension," in SIGMOD'20
  - A PostgreSQL extension
  - Can be implemented in any System R-like DBMS
- Pros:
  - Supports disk storage
  - Supports dynamic data
  - Supports attribute filtering

Data Model

- Treats vectors as a ***field*** in a table
  - Type: `float_vector(d)`
- Index creation:

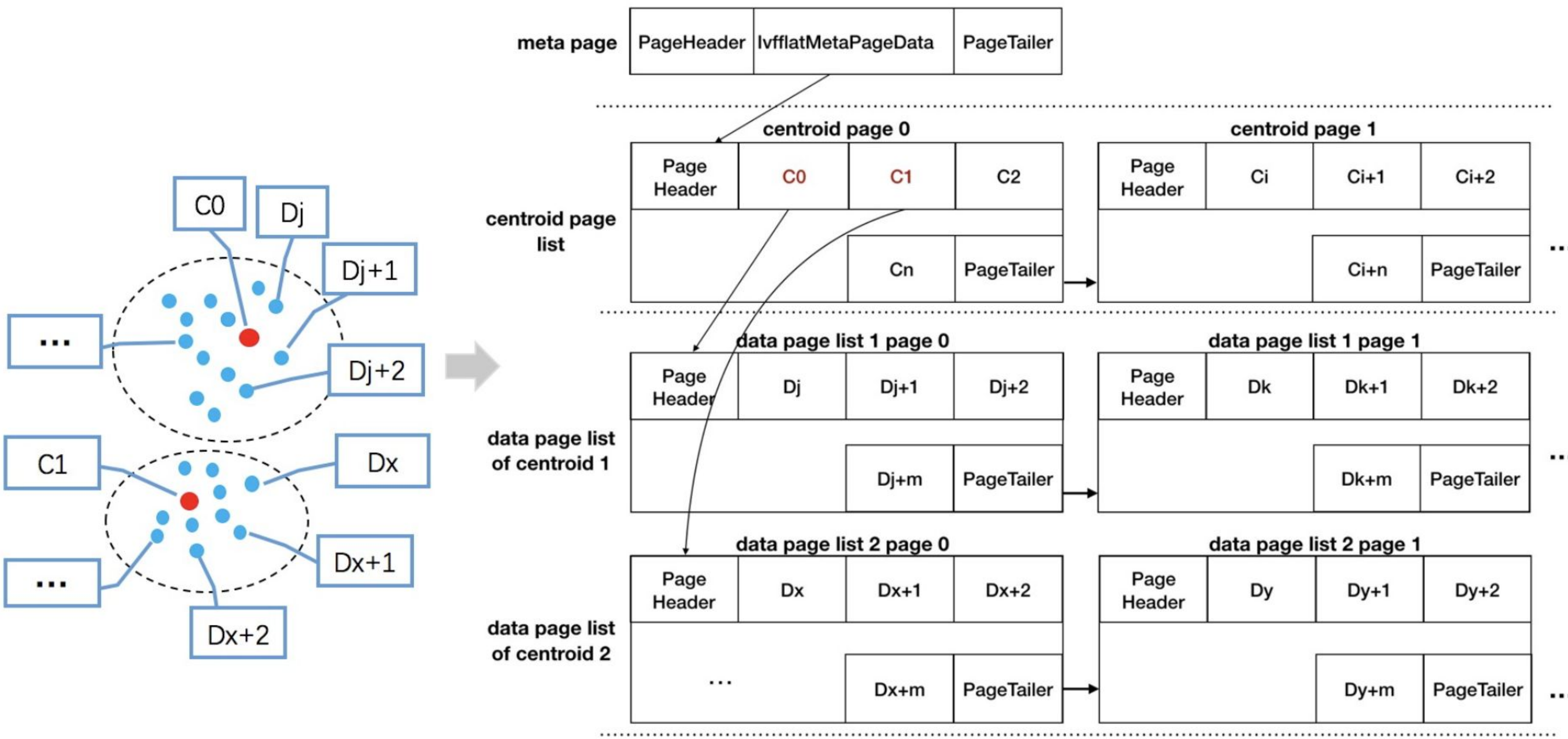
```
CREATE INDEX idx_text ON posts(text_vector)
USING ivf_flat;
```

Query Format

- AKNN query:

```
SELECT p.id,
       p.text_vector <-> '...' AS dist
FROM posts AS p
ORDER BY dist ASC LIMIT 10;
```

Index Building (IVF_FLAT)



- Each page is the unit of buffering and searching

Index Update (IVF_FLAT)

- Do nothing if the data distribution does not change
- Otherwise, continue clustering for few iterations

Planning

```
SELECT p.id,
       p.text_vector <-> '...' AS dist
FROM posts AS p
ORDER BY dist ASC LIMIT 10;
```

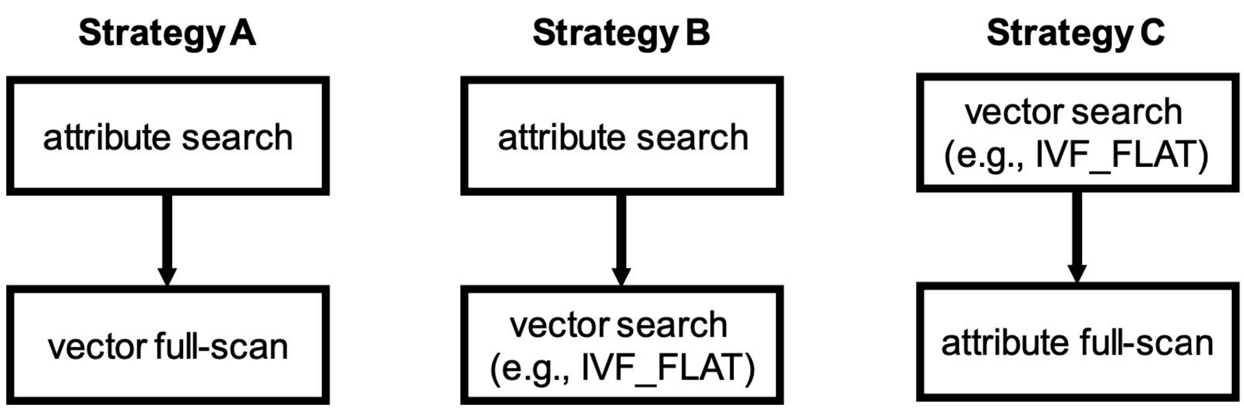
- New `SortPlan` in algebra tree
  - Needs to estimate its own cost

Cost Estimation (IVF_FLAT)

- To select top clusters: B(centroid file)
- Scan for each cluster: B(data file of a centroid)

Attribute Filtering

```
SELECT p.id,
       p.text_vector <-> ‘...’ AS dist
FROM posts AS p
WHERE p.date < ‘...'
ORDER BY dist ASC LIMIT 10;
```



- Best strategy determined by estimated costs

Outline

- Why Vector DBMS?
- AKNN Search Algorithms
- Challenges at System-level
- Case study: PASE (System R-like)
  - Data model & Query Format
  - Index Building & Update
  - Planning & Cost Estimation
- Case study: Milvus (purpose-built)
  - Storage & Consistency Model
  - Computing & Threads
  - Query Algorithms

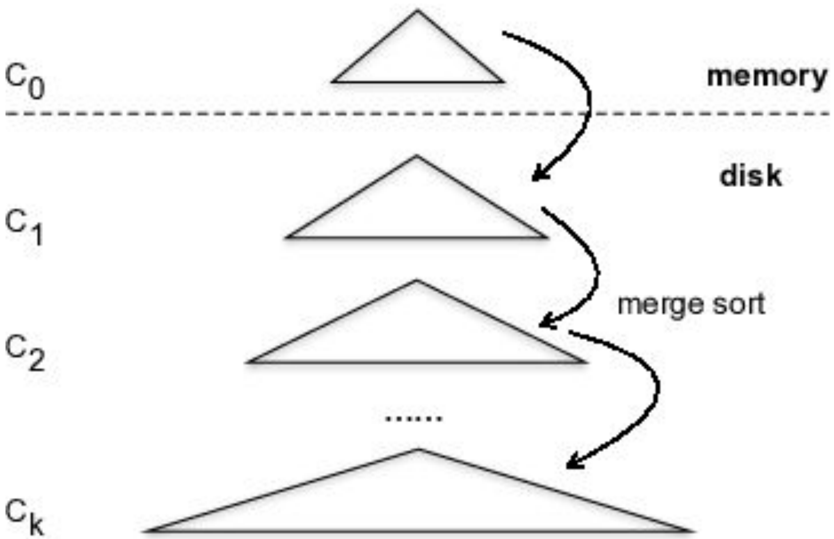
Milvus

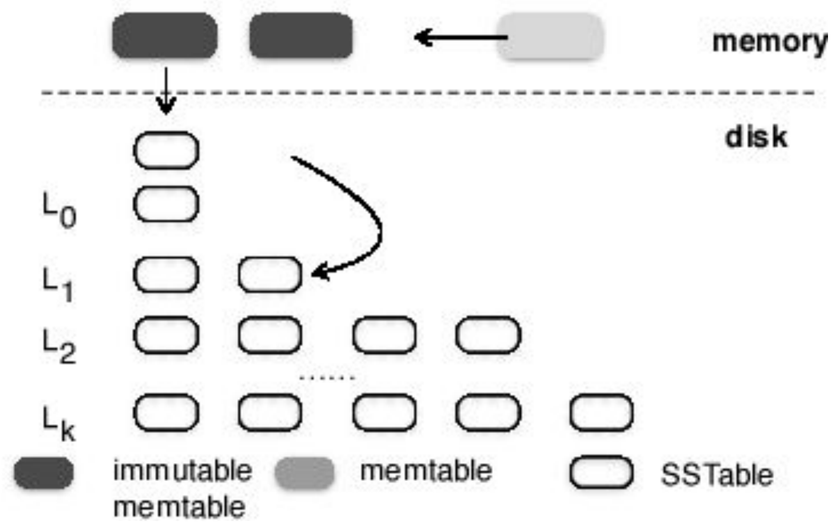
- "Milvus: A Purpose-Built Vector Data Management System," in SIGMOD'21
  - A dedicated system

- Pros:
  - Supports disk storage, dynamic data, attribute filtering
  - Much higher performance than PASE

Storage

- Column storage based on ***Log Structured Merge*** (***LSM***) tree





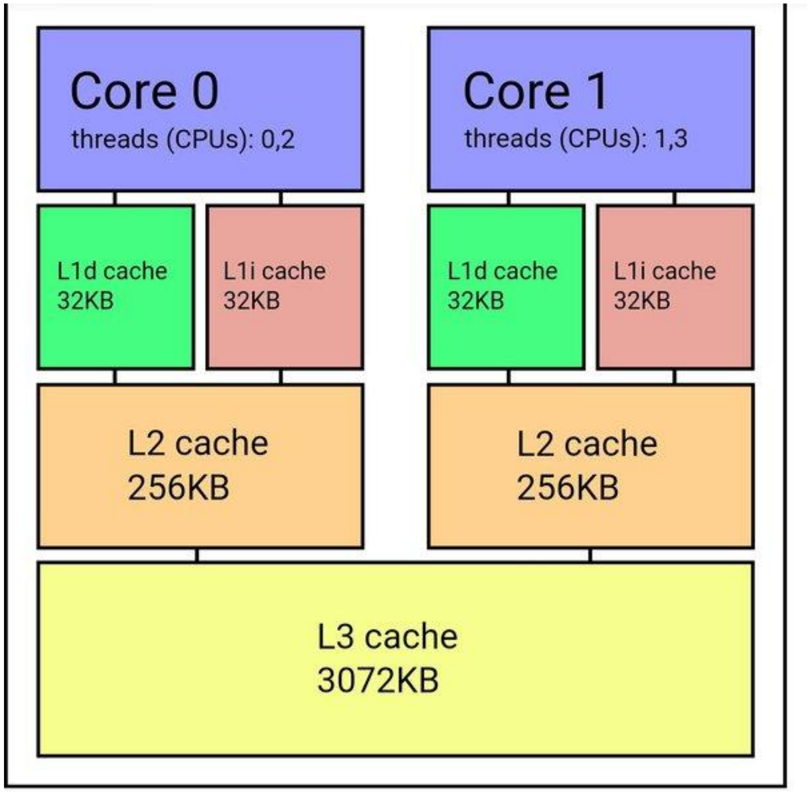
- Out-of-place updates
- SSTables (segments) are the unit of buffering/searching

Consistency Model: Snapshot Isolation

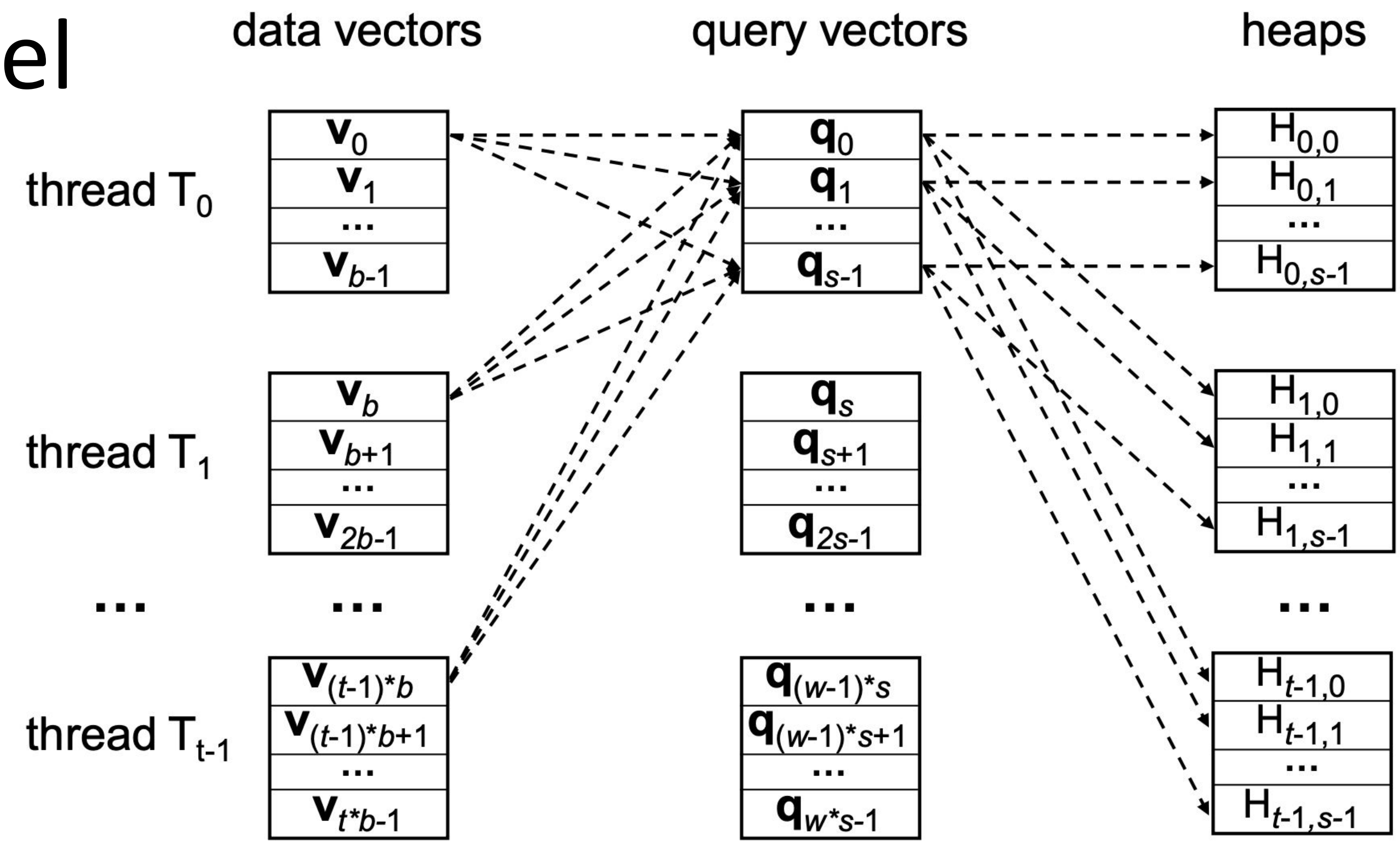
- Every update creates a new data version
- Readers read a consistent snapshot of data
  - Not always the latest one
  - ***Not blocked by writers***
- Milvus maintains snapshots of LSM tree:
  - Snapshot 1: {segment 1}
  - Snapshot 2: {segment 1, segment 2}
  - Snapshot 3: {segment 2, segment 3, segment 4}
  - …

Thread Model

- In PASE, one thread is assigned for each request
  – Hight L3 cache miss rate
- Milvus:
  – Process *m* requests at once
  – ***One thread per cached segment in L3***



Thread Model



- For each query *s:*
  1. Each of *t* threads outputs temp AKNN results in heap $H_{t,s}$ (in L3)
  2. Then, $\{H_{0,s}, H_{1,s}, \ldots\}$ are merged to get final AKNN results
- 1.5 ~ 2.7 speedup

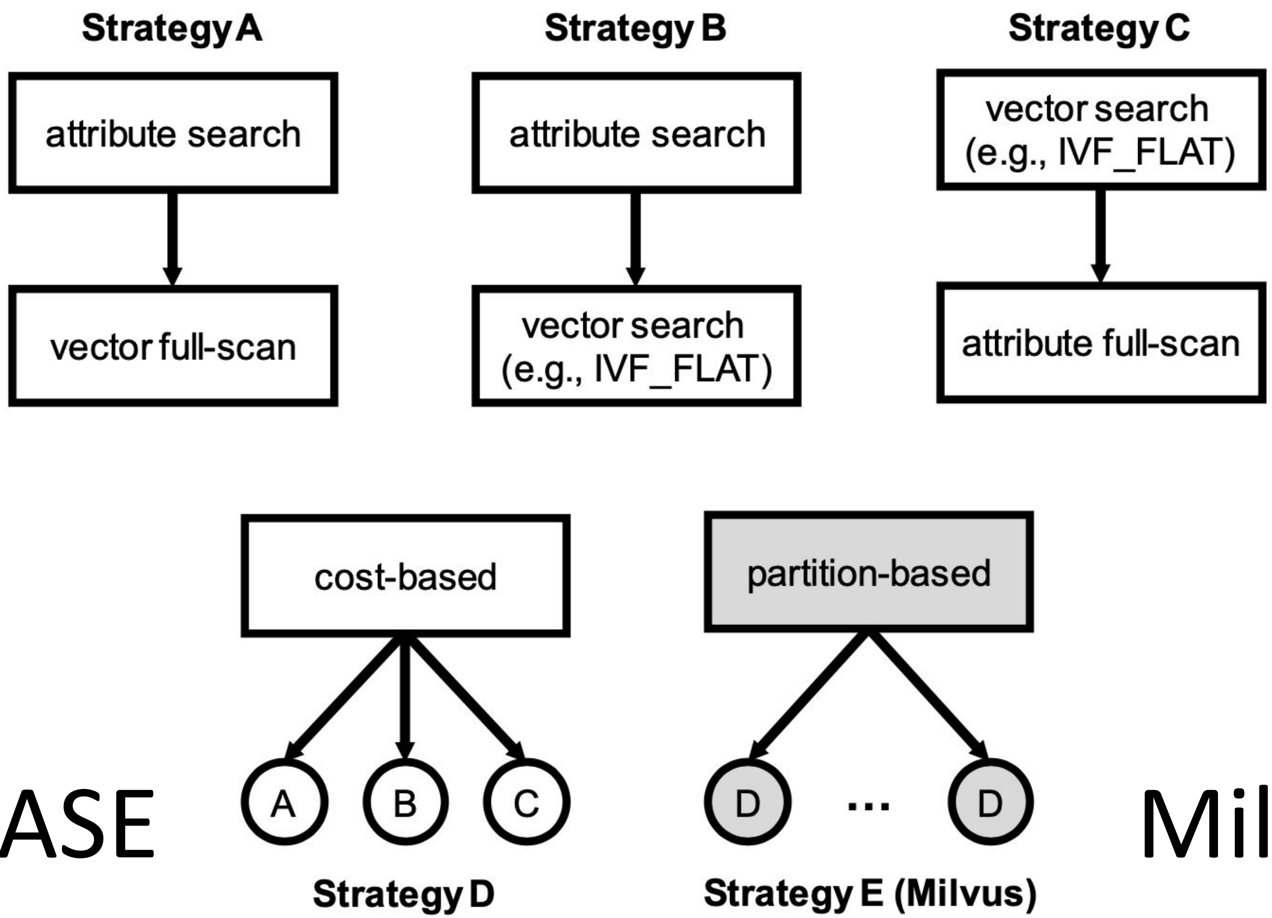
Computing

- Computing the distance between two vectors involves many, ***parallelable*** floating point operations
- Milvus supports hardware acceleration:
  - SIMD instructions on CPU: SSE, AVX, AVX2, AVX512
  - GPU, multi-GPU
  - Also balances GPU speedup vs. bus transfer delay:

Algorithm 1: SQ8H

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

Query Planning



PASE

Milvus

- Partition based on frequently queried attributes
- 13x speedup

Final Project

- Grading
  - Pioneer-run presentation ***30%***
  - Benchmark ***50%***
  - Report ***20%***

Benchmark Grading

- ***Sum(recall per query)*** within a fixed benchmark period

Dataset

- Based on Sift1M dataset
  - ***1M*** vectors
  - ***128 floats*** per vector
  - ***Non-uniform*** distribution
- Running machine can load only ***1/8*** dataset
  - Buffer pool size is only 64M

Benchmark Period

1. ***Data population period***
   – Where you can build your index
2. Query period
   – ***3% inserts***
   – ***3% updates***
   – With spatial/temporal locality

Hints

- Focus on ***I/O optimizations*** first
  - E.g., dimension reduction, special/temporal locality, etc.
- Make CPU busier
  - E.g., by increasing block size
  - So, SIMD can be more useful
- Insert/updates matters
  - K-means++, schedular, etc.
- Parameter tuning