CS 561 FINAL PRESENTATION

Access Path Selection in Modern Columnar DBMSs

Daniel Silla · Mofolaoluwarera Oladipo · Zach Verdieu

Focus: sketches, bitmaps, zone maps, and write-optimized structures

What is Access Path Selection

A modern engine rarely gets to optimize for just one query pattern.

Modern analytical DBMSs have several ways to evaluate predicates

Scans w/ Zone Maps

Uses min/max per block to skip data

Column Sketches

Evaluate predicates on smaller "sketched" data

Bitmap Indexing

(CUBIT/RABIT) Represent matches as bitmaps and combine them.

Each method reduces work differently, but each depends on different factors

Selectivity

How many rows match the query. Determines amount of work required.

Workload mix

Insert-only, mixed interleaved, and mixed sequential workloads can flip the winner.

Data Characteristics

Cardinality, ordering, and data distribution affect whether metadata or indexes pay off.

Research Gap

Modern Storage Changes Access Path Tradeoffs

Modern analytical DBMSs (like DuckDB) use PAX-like layouts internally

- No longer pure column stores OR row stores
- PAX provides better cache locality, efficient vectorized scans, and reduced scan cost & need for heavy indexing
- Existing work assumes traditional layouts → may not reflect modern performance behavior

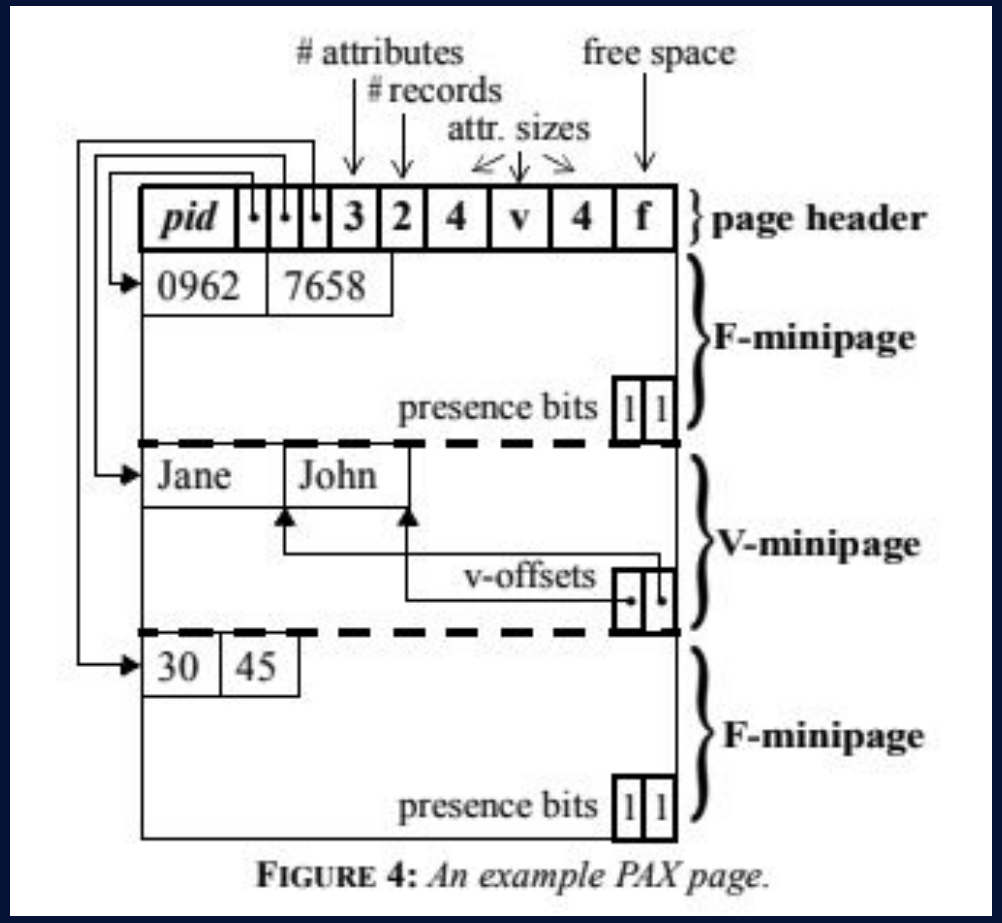


FIGURE 4: *An example PAX page.*

The core project question

How should a DBMS choose among access paths as workload mix and data shape change?

Access Path 1: Scans + Zone Maps

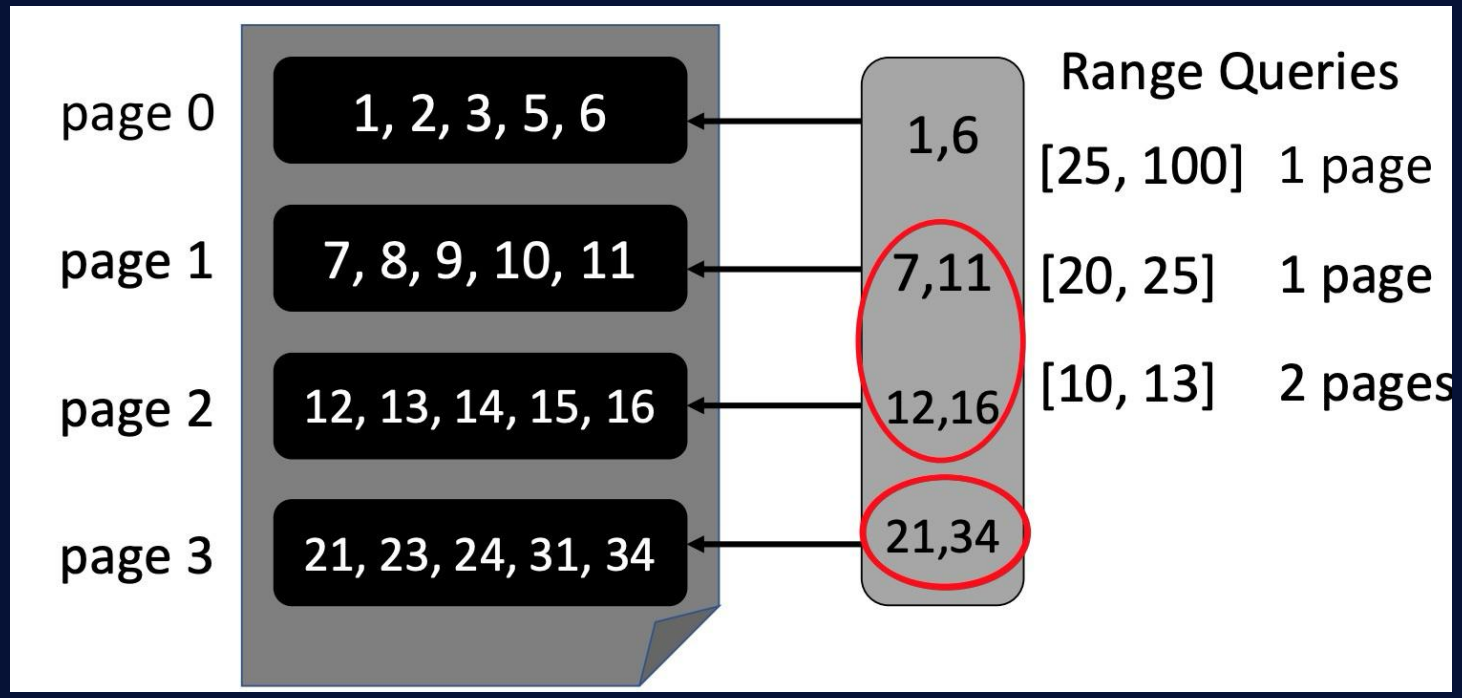
Background

Sequential scans augmented by tracking per bock min/max values

How it works

- Zone maps store min/max values for each block of data
- The system scans the data sequentially, using zone maps to skip blocks that don't match the predicate

Main benefit: Avoids full block scans, reduces I/O & CPU work



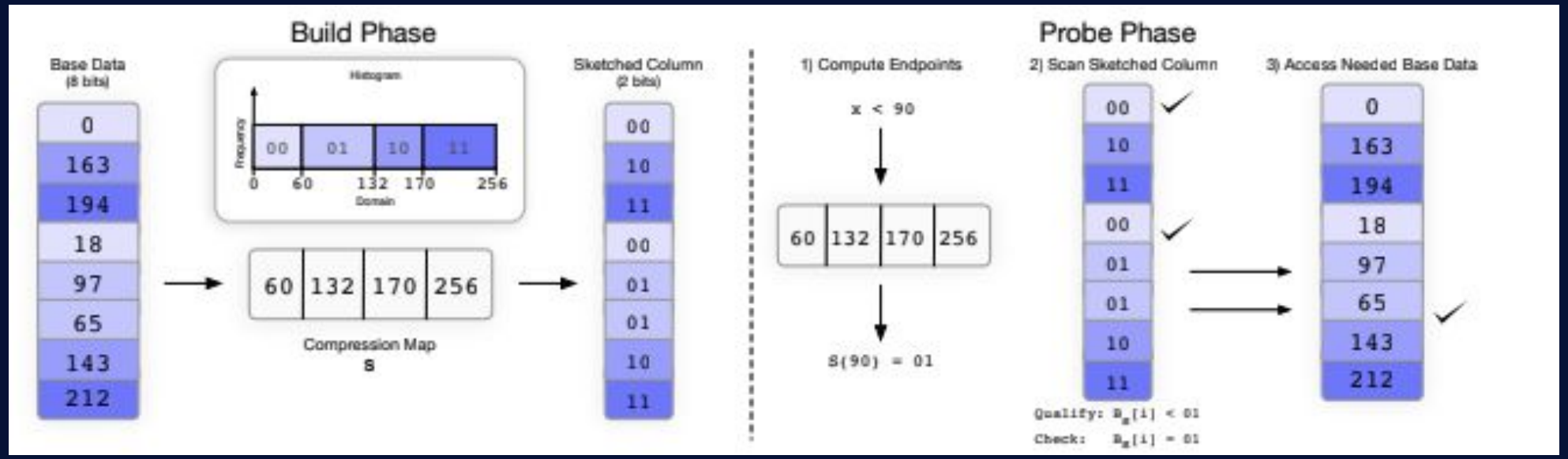
Access Path 2: Column Sketches

Build sketch column to limit base data access

How it works

- Build a compressed sketch column mapping each value to compressed code
- During a query: the system evaluates the predicate on the sketch column first
- Only accesses base data if it determines a potential match

Main benefit: Reduces data movement by avoiding accessing base data for many rows



Access Path 3: Bitmap Indexing

Background

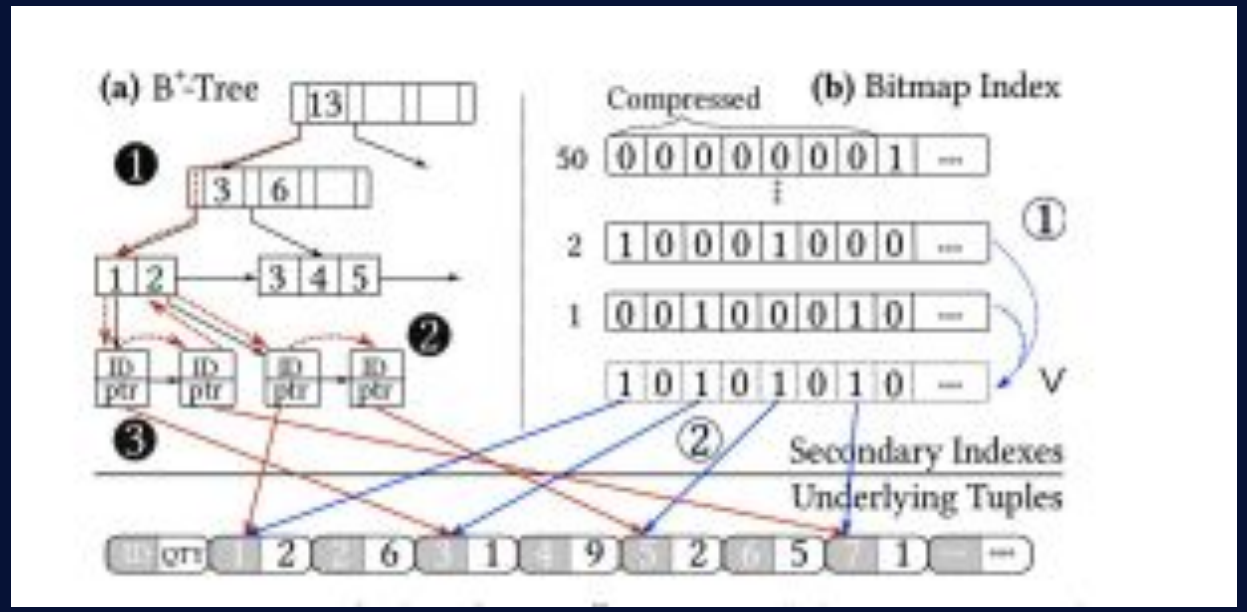
Precompute matching rows and combine results when querying

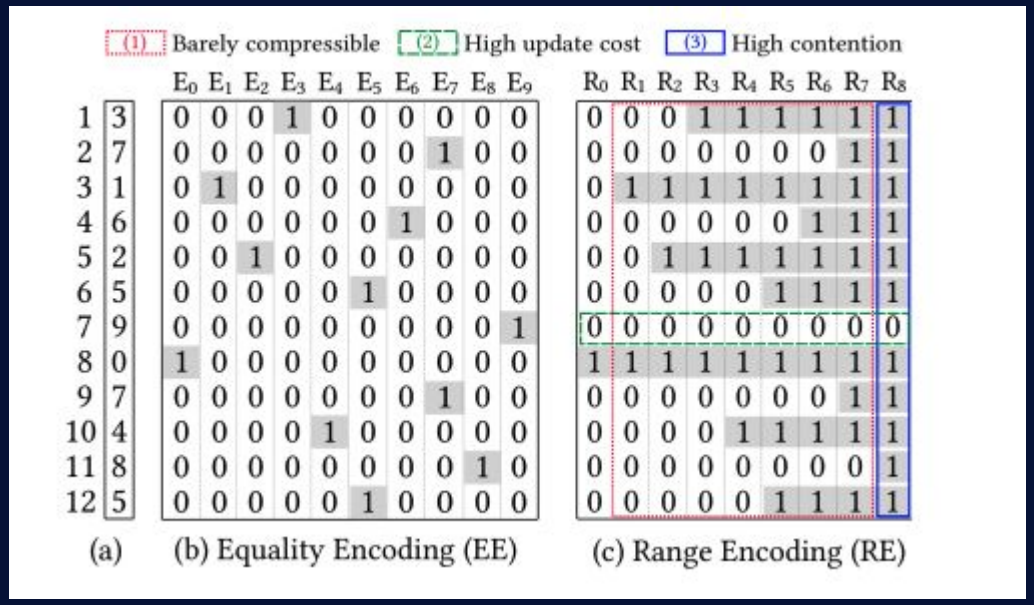
How it works

- For each value (or range), a bitvector is created with 1s denoting that the row that corresponds to the position matches the value, and 0's for rows that don't match
- During a query: combine bitvectors using bitwise AND/OR operations ⟶ the result is the matching rows

Main benefit: Avoids scanning entirely & uses fast bitwise operations

Limitations: Poor update support, inefficient range queries





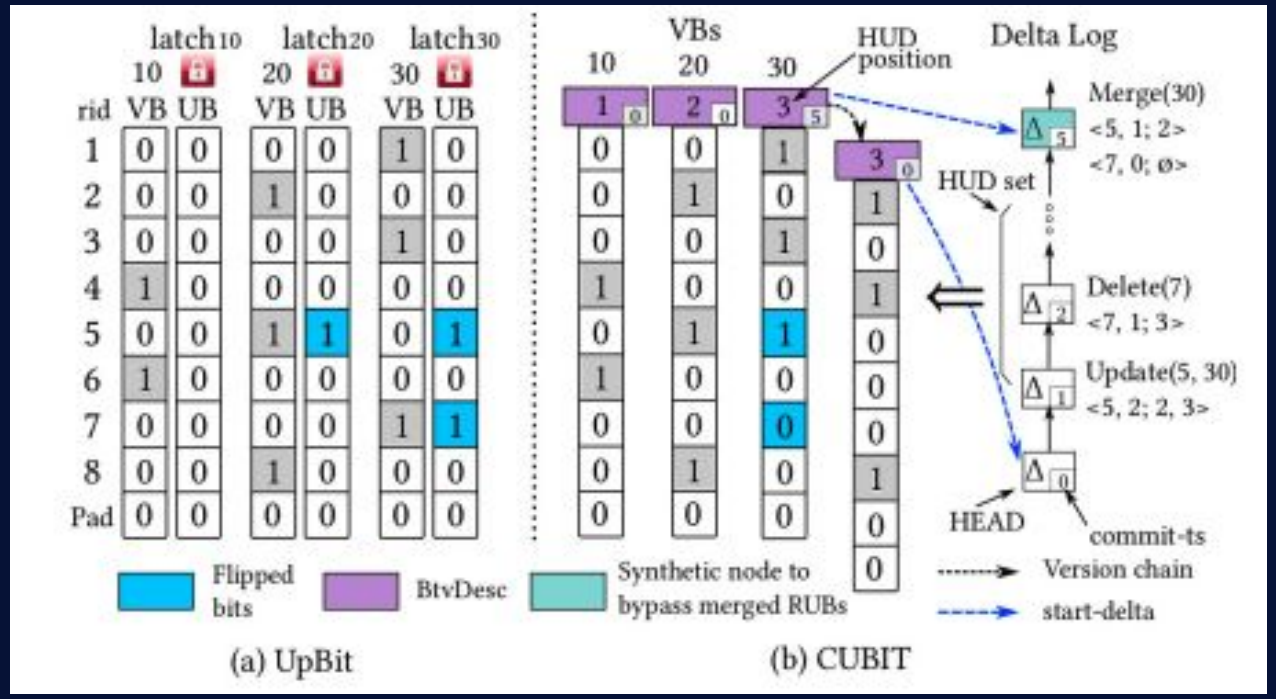
Access Path 3 cont.: CUBIT & RABIT

Background

Modifications to standard bitmap indexing for specific gains

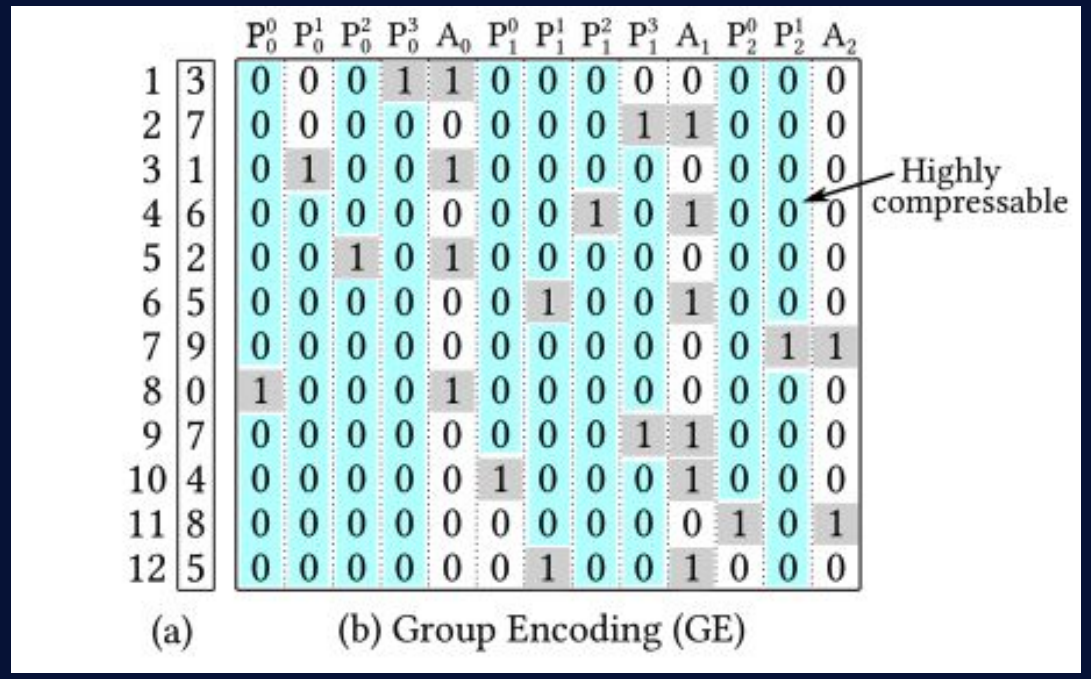
CUBIT

- uses multiversioned delta structures to facilitate fast/efficient updates



RABIT

- Uses Group encoding for efficient range queries



Problem Statement & Motivation

Motivation

The midway report became a turning point in how the scan study was interpreted.

The Challenge

Modern analytical DBMSs must choose between multiple data retrieval mechanisms. The wrong choice leads to wasted I/O, redundant computation, and degraded query performance.

Our Approach

Analyze access path efficiency across workloads and data properties using DuckDB — studying query selectivity, column cardinality, predicate structure, and data distribution.

Goal: Develop validated access path decision rules for analytical query workloads.

Experimental Setup

Schema: TPC-H lineitem (16 columns, realistic value distributions)

Three dimensions tested:

- Selectivity: 5%, 10%, 20%, 40%, 60%, 80% (fixed cardinality=1000, uniform)
- Cardinality: 10, 50, 100, 256, 500, 1K, 10K, 100K NDV (fixed 10% selectivity, uniform)
- Distribution: Uniform, Zipf, Clustered (fixed cardinality=1000, 10% selectivity)

Hardware & Methodology:

- Platform: BU SCC
- CPU: Intel Xeon Gold 6242 @ 2.80GHz
- RAM: 250GB
- SIMD: AVX-512
- Threads: 4
- Runs: 3 averaged after 1 warmup
- Engine: Custom DuckDB
- Metric: Average latency

Selectivity experiment

What was tested

lineitem: sketch-enabled dataset

lineitem_plain: identical baseline table using standard scan with zone maps

Five sketch-associated columns with thresholds tuned to hit different selectivity levels

Example query

```
SELECT *
FROM lineitem
WHERE l_shipdate < DATE '1993-01-01';
```

Why this mattered

The expected story was that more selective queries would favor Column Sketches.

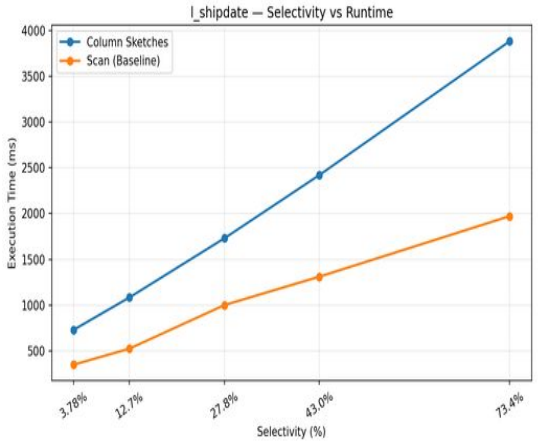
But the midway results showed the opposite: the baseline scan consistently outperformed the sketch-enabled table across all columns and selectivity levels.

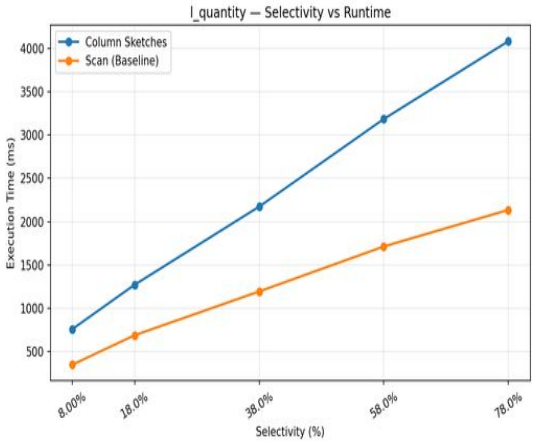
That forced a refinement of the hypothesis: selectivity alone was not enough to explain access-path quality.

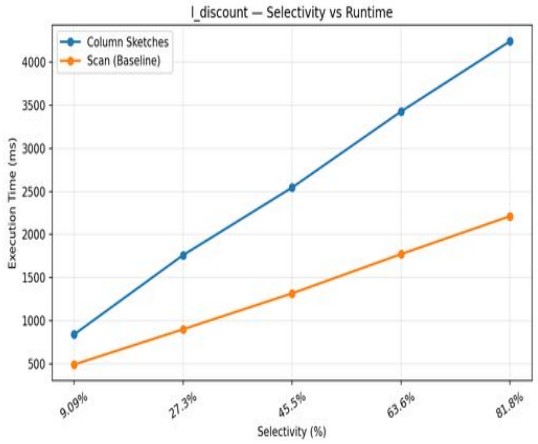
Selectivity changed runtime, not the ranking

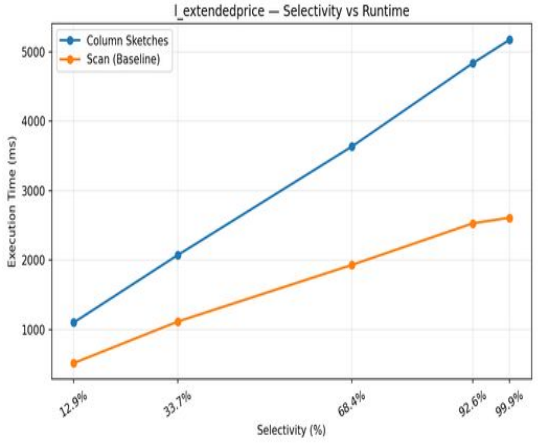
Midway plots

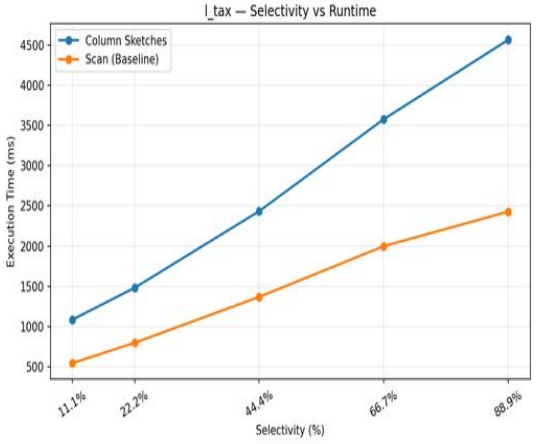
Across all five columns, Column Sketches stayed slower than the baseline scan.











Midway takeaway

Across all five sketch-enabled columns,
the baseline scan (zone maps) stayed faster.

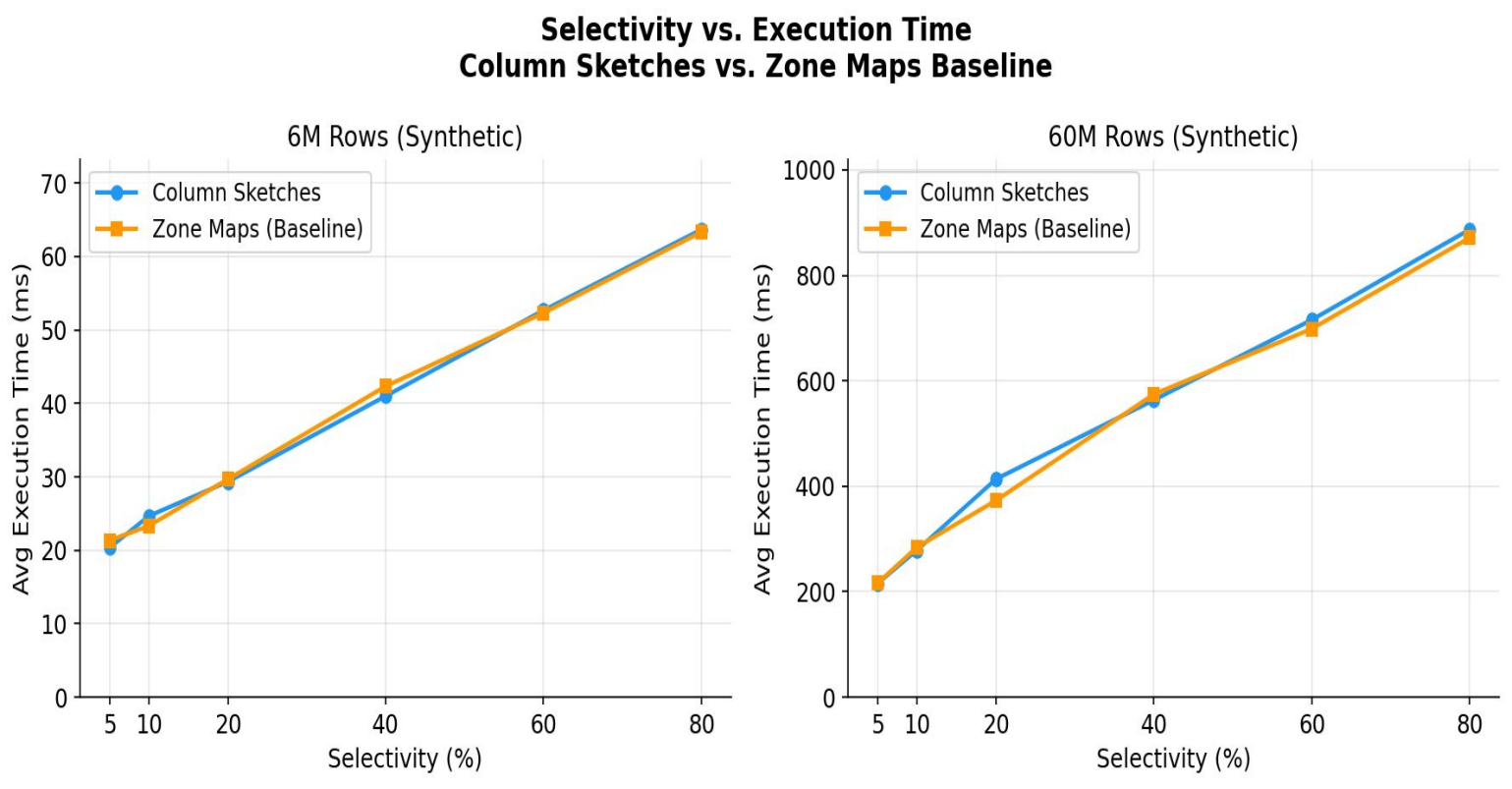
Selectivity changed runtime,
but did not change the ranking.

In practice, the **sketch evaluation** itself has a cost building the compressed codes, running AVX-512 SIMD lookups across the sketch column. So **the filtering benefit cancels out with the evaluation overhead**

Sketch vs. Baseline: Selectivity (Synthetic)

Final results

Results hold across scales



Both methods **scale linearly** and sketch shows no advantage at any selectivity level at either scale.

Cardinality experiment

What was tested

lineitem: sketch-enabled dataset

lineitem_base: identical baseline table using standard scan with zone maps

l_quantity column with NDV ranging from 10 to 100,000

Fixed selectivity at ~10%, uniform distribution

Example query

```
SELECT *
FROM lineitem
WHERE quantity < 100;
```

Why this mattered

Cardinality refers to the number of distinct values in a column. A column with 10 distinct values has low cardinality; one with 100,000 has high cardinality.

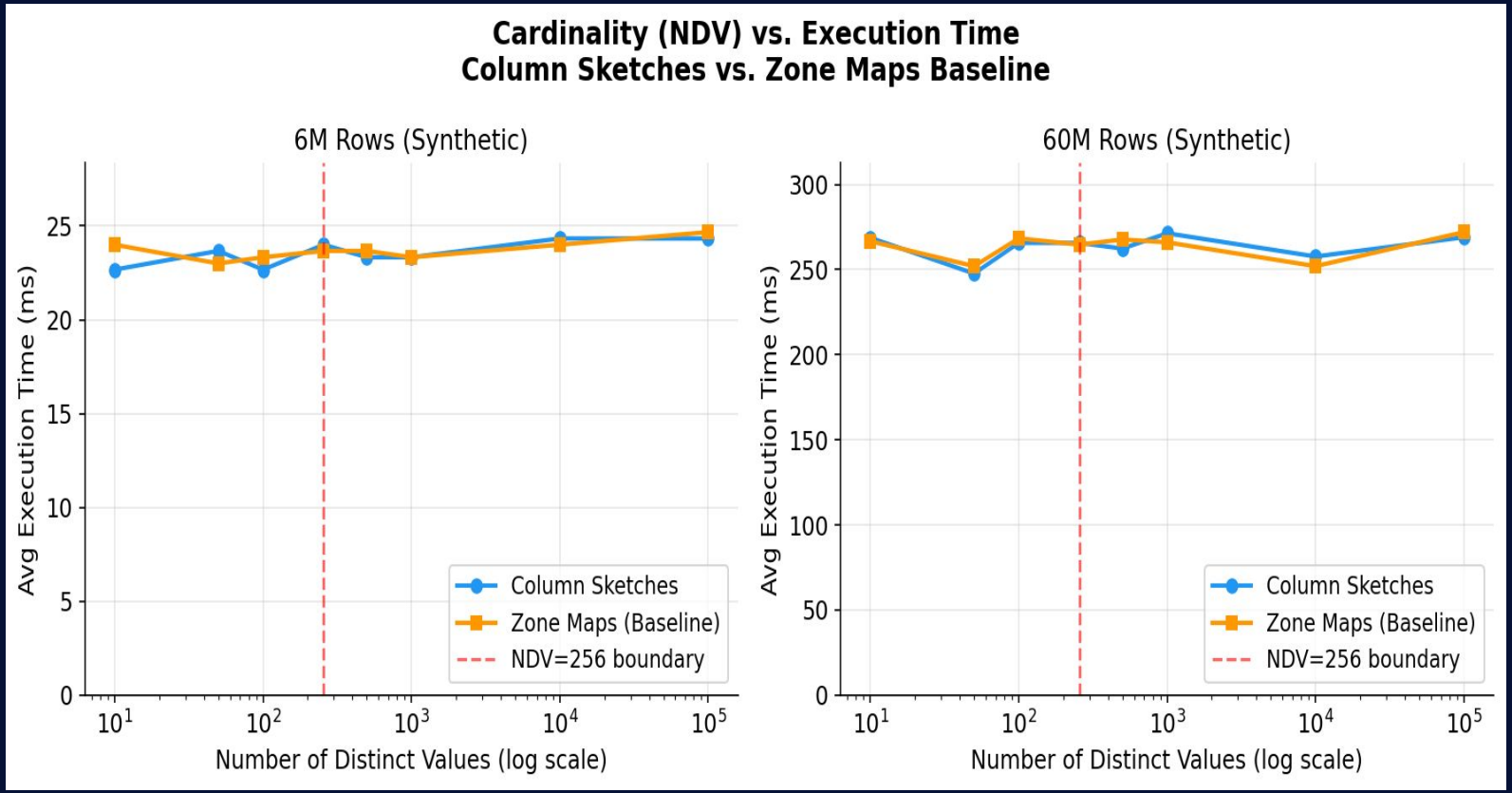
This matters for sketches because the sketch uses 1-byte codes and they're only 256 unique values

Theory predicts sketch **performance should degrade above NDV=256**, where internal code collisions increase and base data verification is required.

Sketch vs. Baseline: Cardinality (Synthetic)

Final results

NDV has no effect on sketch performance



Even though collisions do occur above NDV=256, the **false positive rate turns out to be low** enough that the extra base data verification adds negligible cost.

Distribution Experiment

What was tested

lineitem: sketch-enabled dataset

lineitem_base: identical baseline table using standard scan with zone maps

l_quantity column with fixed cardinality of 1,000 distinct values

Fixed selectivity at ~10%

Three distributions tested: Uniform, Zipf, Clustered

Example query

```
SELECT *
FROM lineitem
WHERE l_quantity < 100;
```

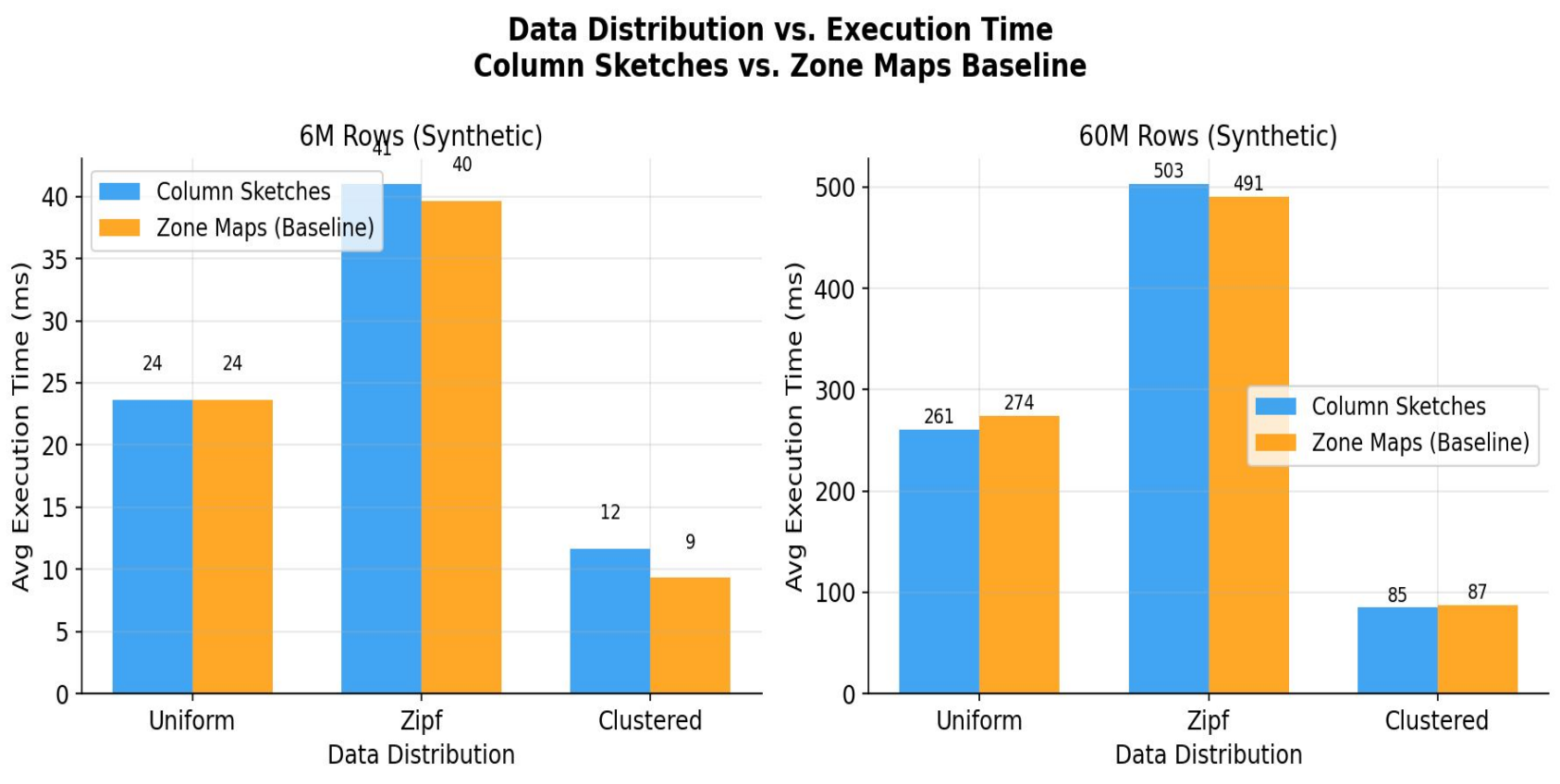
Why this mattered

Data distribution describes how values are arranged within a column. Uniform means all values appear equally. Zipf means a few values dominate. Clustered means similar values are stored together in sorted order.

Sketch vs. Baseline: Distribution (Synthetic)

Final results

Data layout matters more than access path choice



Clustered data dramatically reduces runtime for both methods. Zone maps skip entire row groups when data is sorted, making the access path choice irrelevant. On uniform and Zipf data both methods perform identically.

Key Takeaway: Data layout matters more than access path choice

Final scan studies: Experiment 1

RABIT/CUBIT experiment

Adding CUBIT and RABIT

Experiment: Run query 5 times per combination, take median, using: Plain (zone maps), Sketching, CUBIT, and RABIT
Query: SELECT count(*) FROM lineitem WHERE <col> < <selectivity threshold>

l_discount: Low cardinality (11 NDV: 0.00–0.10 step 0.01), uniformly distributed, no clustering

l_extendedprice: High cardinality (~1M+ NDV), near-continuous range, no meaningful clustering

l_quantity: Low cardinality (50 NDV: 1–50), uniformly distributed, no clustering

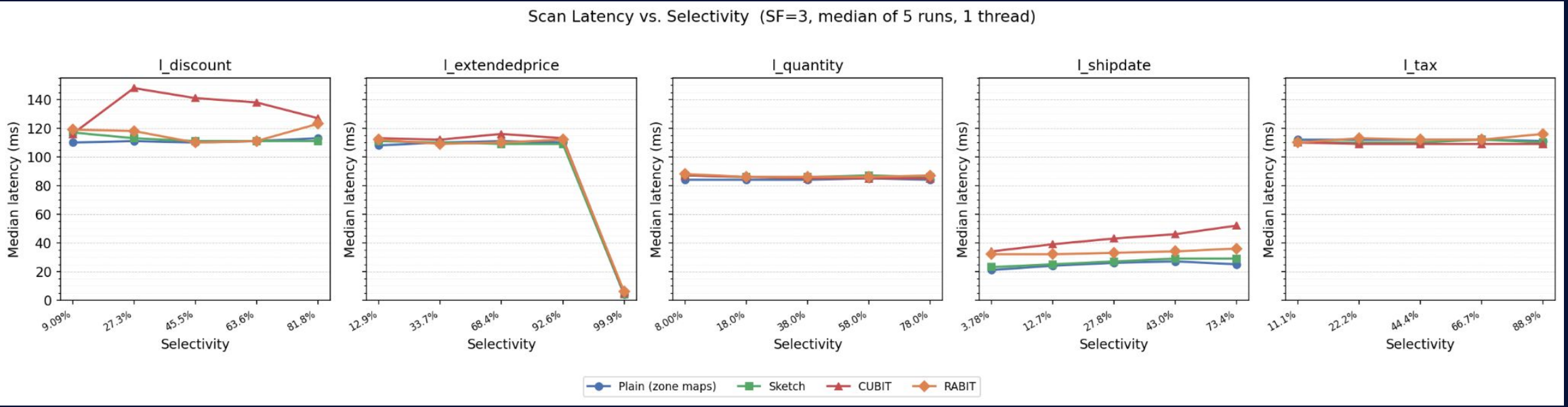
l_shipdate: High cardinality (dates ~1992–1998), physically sorted/clustered by date in storage

l_tax: Very low cardinality (9 NDV: 0.00–0.08 step 0.01), uniformly distributed, no clustering

Final scan study: the right answer depends on the column

Adding CUBIT and RABIT

RABIT/CUBIT results



Key interpretations

l_discount: Uniform, and no clustering mean all methods have to perform almost a full scan, CUBIT index is extra overhead

l_extendedprice: Uniform distribution, and no clustering again almost full scans

l_quantity: No clear winner, data is uniform between 1-50, zone maps can't skip blocks, and bitmaps can't prune

l_shipdate: Plain scans with zone maps benefit from clustering and skip whole groups efficiently.

l_tax: only 9 distinct tax values, 0.00 to 0.08, expected CUBIT to win, but we believe the uniform data leads to scans, giving less advantage to CUBIT

Final scan studies: Experiments 2 & 3

RABIT/CUBIT experiment

Adding CUBIT and RABIT

Experiment 2 Cardinality: Values created at random, tables are created with different numbers of distinct values (NDV) at 10, 100, 1,000, 10,000 and 100,000.

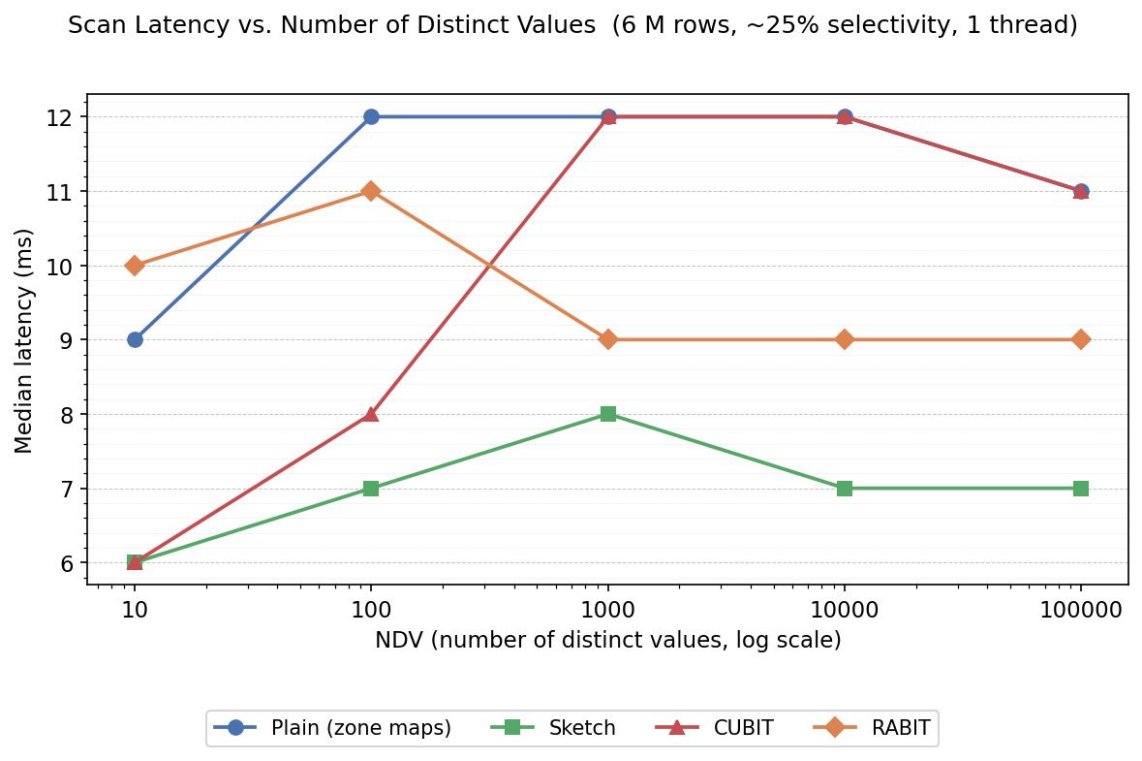
SELECT count(*) FROM synth WHERE val < NDV // 4 — targets ~25% selectivity

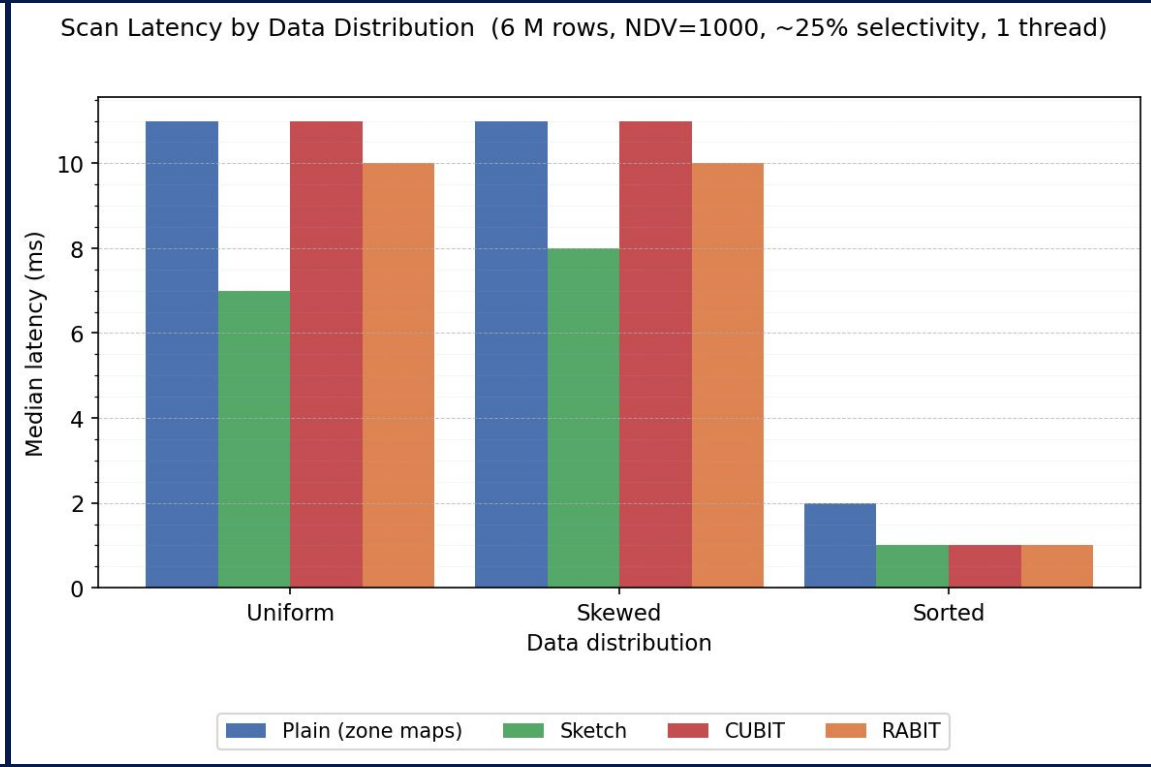
Experiment 3 Data Distribution: 3 data sets are created Uniform (random values [0,1000)), Skewed (values [0, 1000) concentrated near 0), Sorted determined values [0,1,...,999] each appearing 6,000 times.

SELECT count(*) FROM synth WHERE val < 250 25% selectivity

Cardinality and data layout explain the scan winners

The final experiments show why selectivity by itself was too weak a predictor.





Takeaway: low-NDV columns can justify CUBIT indexes, sorted data can make simple zone maps highly competitive for every method.

Final decision rules

The coherent story across all files is that adaptive selection beats fixed defaults.

High-selectivity scans:

When most rows qualify, sequential scan with zone maps is the most predictable and lowest-overhead choice.

Mixed writes + point lookups

Move to HashLinkList or HashSkipList once reads are introduced; lookup awareness matters immediately.

Low-cardinality range scans

CUBIT bitmap methods become attractive when the value domain is very small.

Clustered or ordered columns

Plain scans with zone maps remain strong when ordering lets the engine skip whole groups.

Bottom line: there is no universal best access path. The winner is a function of workload mix, cardinality, and layout.