Presented by:
Archit Kiran Kumar
Marcus Izumi
Liang Yu Lin

# What is Access Path Selection?

**High-Level Idea**

When a query is executed, the database must decide:

How should the data be accessed?

Should it:

- scan the full table?
- skip irrelevant blocks using Zonemaps?
- use Column Sketches for pruning?
- use Bitmap-based methods like RABIT?

# What is Access Path Selection?

This decision is called:

**Access Path Selection**

Why does it matter?

**A poor decision causes:**

- unnecessary full scans
- higher query latency
- wasted CPU and memory

**A good decision improves:**

- query speed
- analytical performance

**Goal:**

- **Choose the best access path based on workload characteristics.**

# WHY DUCKDB?

**DuckDB is ideal because:**

- Columnar analytical DBMS
- Vectorized execution engine
- Lightweight and easy to modify
- Built-in Zonemap support
- Existing Column Sketch implementation

Perfect for controlled systems experiments.

# ACCESS PATHS

**1. Zonemaps**

Store:

- minimum value
- maximum value

For a range of values

**2. Column Sketches**

Stores value ranges

→ preserve ordering

→supportSIMD+vectorized pruning

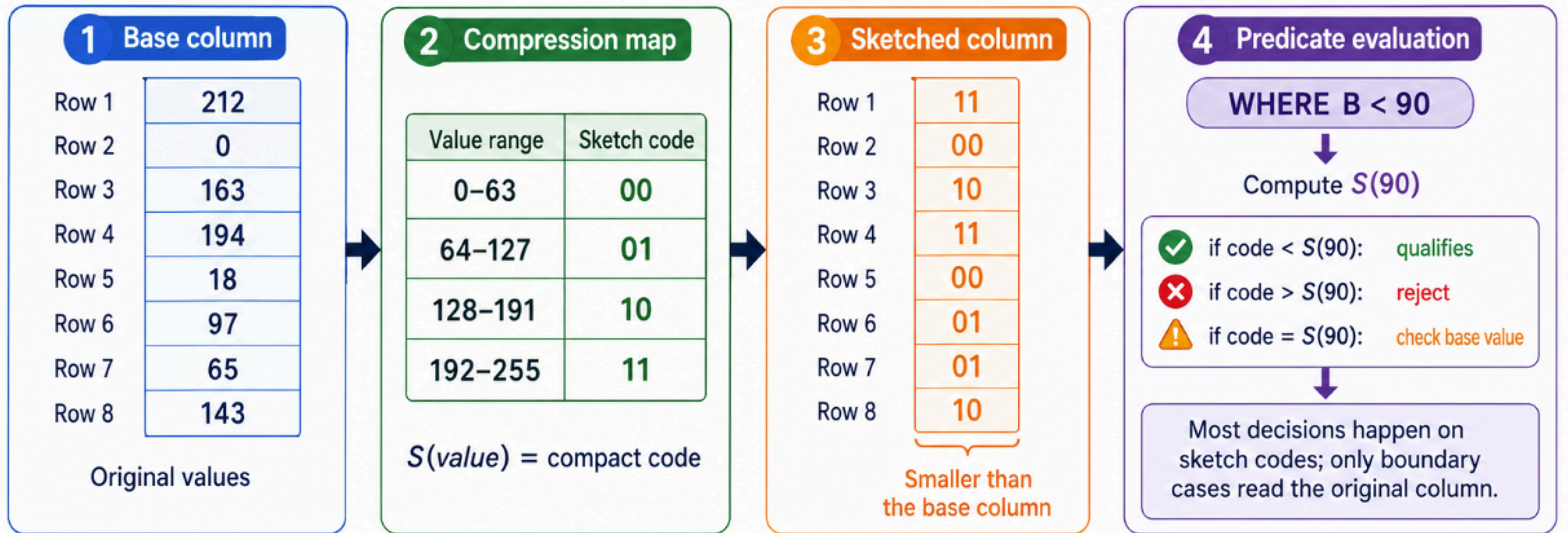
**3. RABIT**

Represent values using bit vectors

→ efficient range evaluation

→ bitwise predicate execution

# Column Sketch

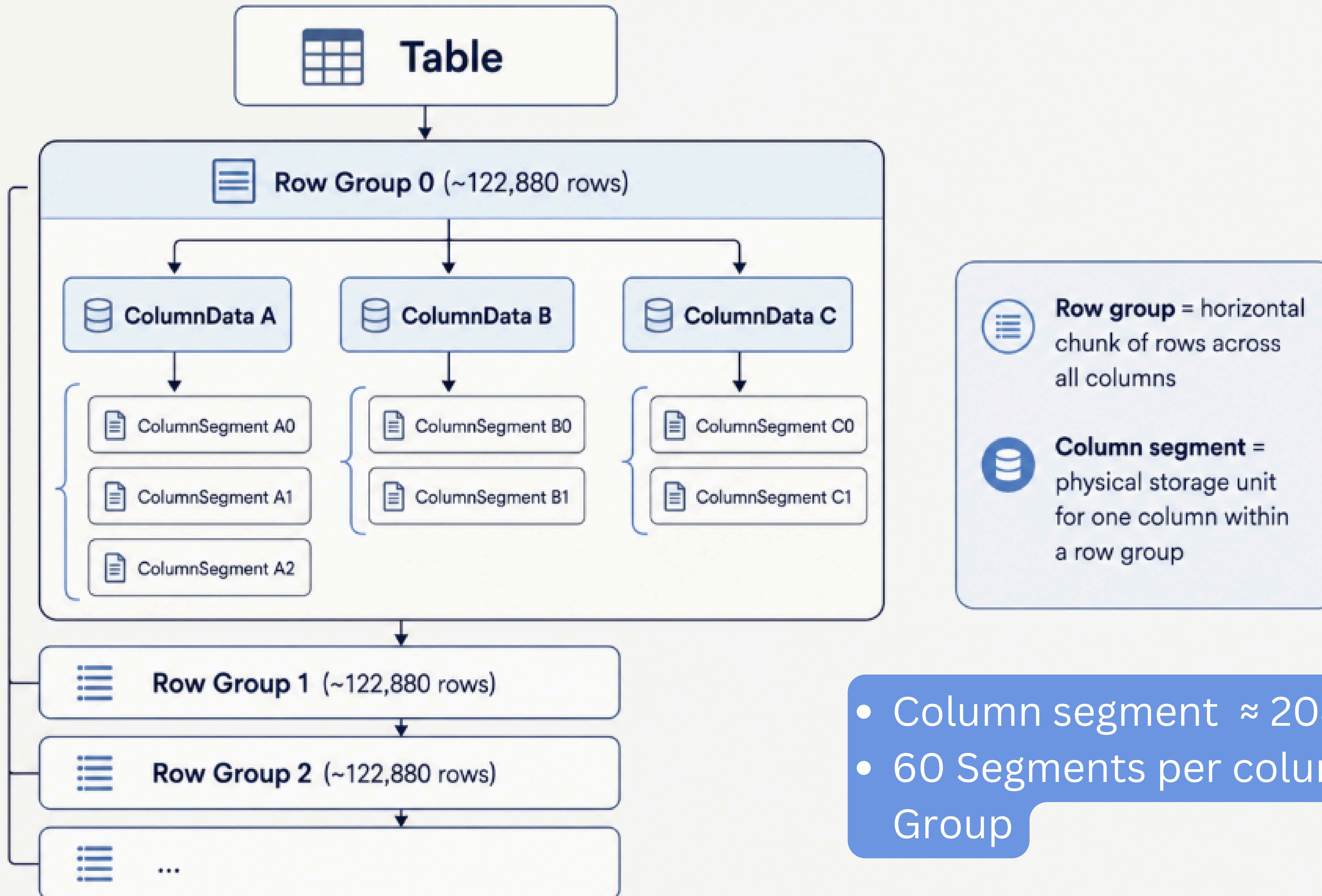



- Compact code-level information per-value
- Scans small codes (1-2 bytes) first and only reads full base values for uncertain cases

Overhead:

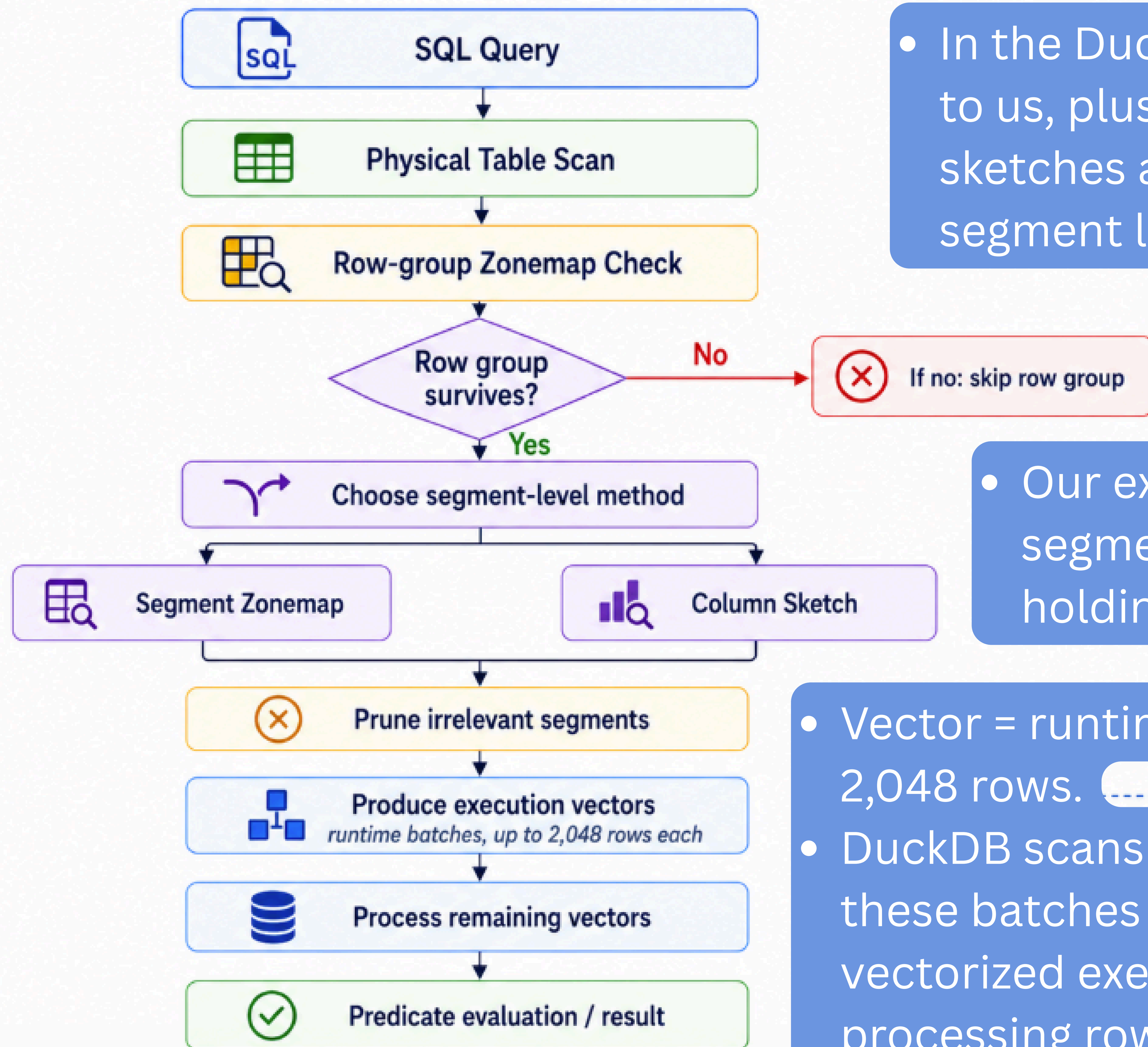
- Sketch adds extra evaluation work
- Can reduce scan work but still increase latency if the pruning benefit is too small

# DuckDB Storage Hierarchy




- Column segment ≈ 2048 rows
- 60 Segments per column for a Row Group

# Access-Path Decision During a DuckDB Table Scan




- In the DuckDB implementation provided to us, plus our modifications, column sketches are only used at the column-segment level.

- Our experiments compare the two segment-level methods while holding row-group pruning fixed

- Vector = runtime batch of up to 2,048 rows.
- DuckDB scans columnar data in these batches for efficient vectorized execution, rather than processing row-by-row.

Experiment Set-Up
& Methodology

# HARDWARE SETUP

- **Platform:** Boston University Shared Computing Cluster (SCC)
- **Environment:** Linux-based remote compute nodes with AVX-512 (Advanced Vector Extensions 512)
- **Database System:** DuckDB (modified source build)
- **Language & Tools:** C++, SQL, Python, VS Code, GitHub
- **Workload Execution:** Synthetic analytical workloads with repeated runs for stable averages

# EXPERIMENTAL FRAMEWORK

**Goal**

Compare access paths under identical conditions

Mode A: Row-group zonemap + segment level zonemap pruning
Mode B: Row-group zonemap + segment level column sketch pruning

RABIT is not included in this comparison because it was evaluated using a separate Python prototype

- RABIT implementation were not supported for DuckDB

# DATA GENERATION

**Synthetic Workload Generation**

**Base Schem**

- Each synthetic table has 10,000,000 rows
- 3 columns

**We generate variants of the base table, changing:**

- Cardinality, Sortedness, etc.

**Why Synthetic Data:**

- Gives precise control over one variable at a time

```
CREATE TABLE t_base AS
WITH params AS (
    SELECT 1000000::BIGINT AS ndv_a
)
SELECT
    i::BIGINT AS id,
    (i % p.ndv_a)::INTEGER AS a,
    (((i * 48271) % 10000) / 100.0)::DOUBLE AS b
FROM range(10000000) AS r(i)
CROSS JOIN params AS p;
```

# Important Metrics:

**vectors_processed:**

- Number of execution vectors scanned after pruning
- One vector = up to 2,048 rows
- High value means more data reached the normal scan/filter path
- Low value means pruning avoided scan work

**latency:**

- End-to-end query runtime from EXPLAIN ANALYZE
- Includes scan work, predicate evaluation, aggregation, and pruning overhead
- For sketch mode, this includes the cost of checking column sketches

# Experiment 1: Sortedness

Same logical table contents, different physical layouts:

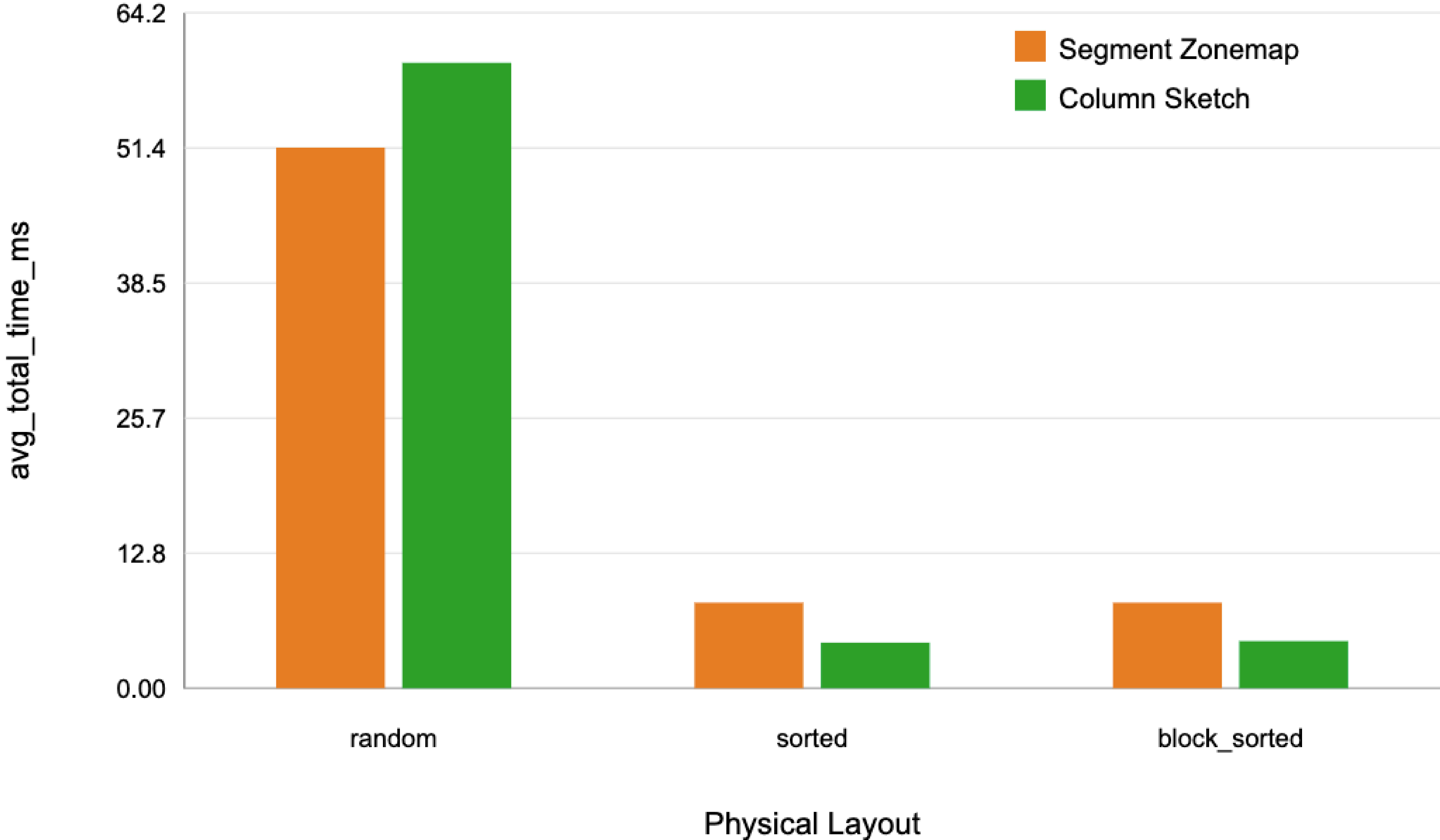
- random, sorted, block-sorted

For each layout, we run the same set of range queries:

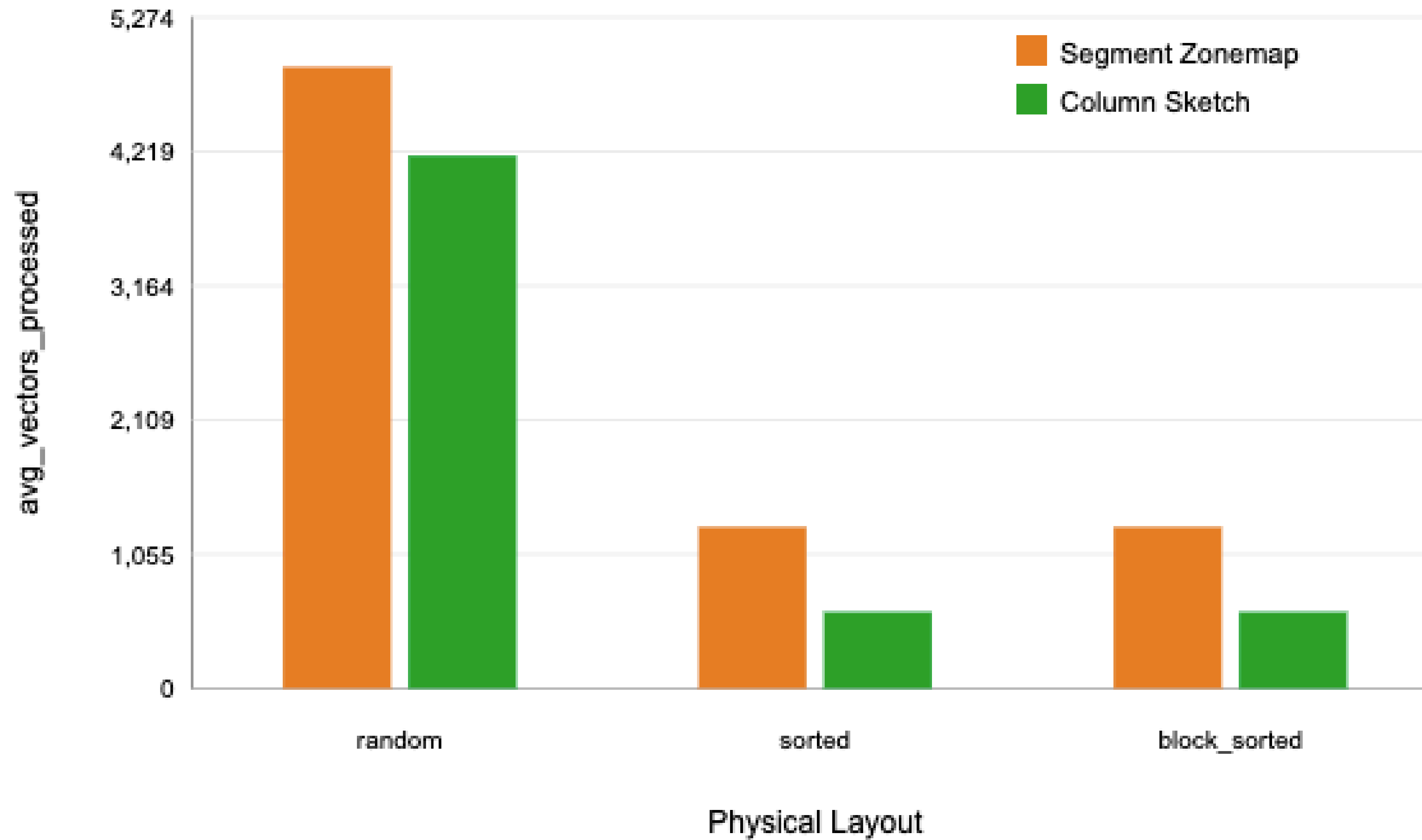
- 0.1%, 1%, 5%, 10%, 20%, 50%, and 90% selectivity
- Graph averages those query results into one value per layout

**Goal: Understand how physical clustering changes pruning effectiveness**

# Layout Vs. Latency



# Layout Vs. Vectors Processed



# Key Observation:

Random layout:

- Column Sketch processes fewer vectors but is slower

Sorted / block-sorted layout:

- Column Sketch processes fewer vectors and is faster

Random layout mixes values across segments, so sketch cannot eliminate enough work to pay for its overhead.

Implication:

- Use Segment Zonemap for random layout.
- Use Column Sketch when layout creates segment-level pruning opportunity.

# Experiment 2: NDV/Cardinality

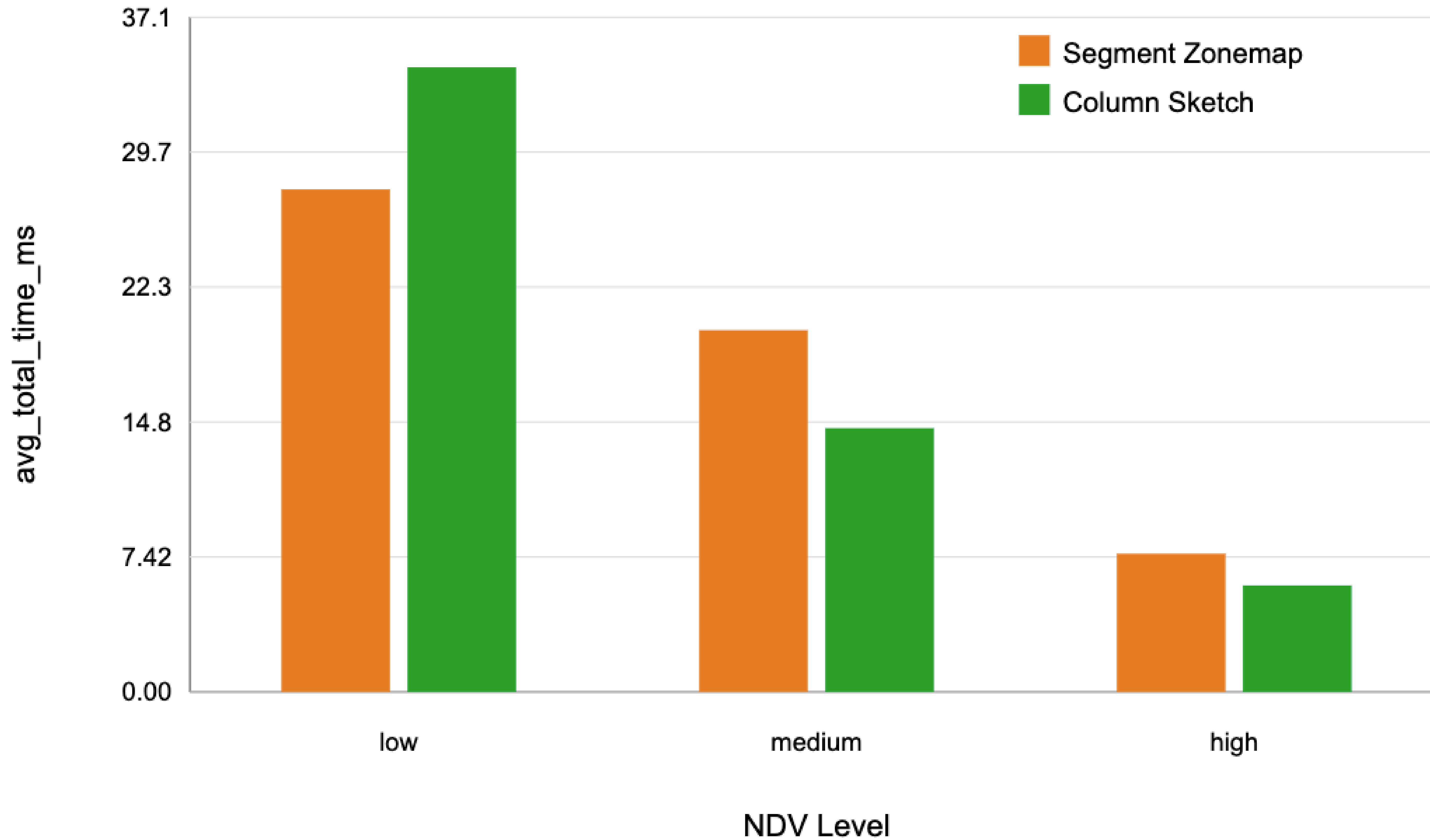
Same logical table contents, different cardinality levels for column a:

- low NDV = 100 distinct values
- medium NDV = 10,000 distinct values
- high NDV = 1,000,000 distinct values
  - Each value repeats ~10 times

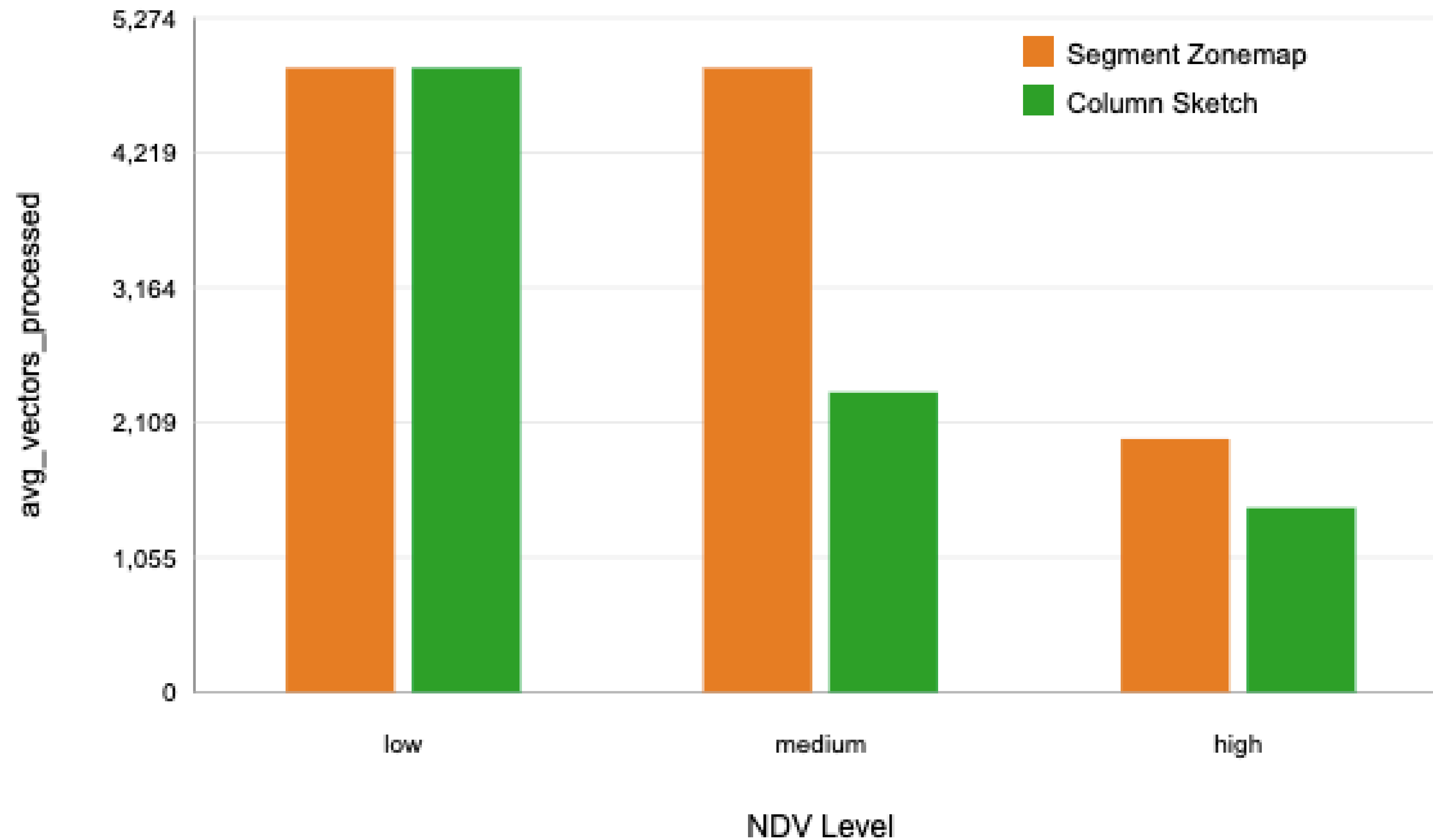
Again, average query results into one value per cardinality level

**Goal: Understand how value cardinality changes pruning effectiveness**

# Cardinality Vs. Latency



# Cardinality Vs. Vectors Processed



# Key Observation:

Low NDV:

- Sketch and zonemap process almost the same number of vectors, but sketch is slower.

Medium / high NDV:

- Sketch processes fewer vectors and becomes faster.

At low NDV, values repeat broadly across segments, so sketch cannot prove absence.
At higher NDV, values are sparser at the segment level, so sketch can eliminate more vectors.

Low NDV → use  Zonemap

Medium / high NDV → Column Sketch can win, especially for selective predicates.

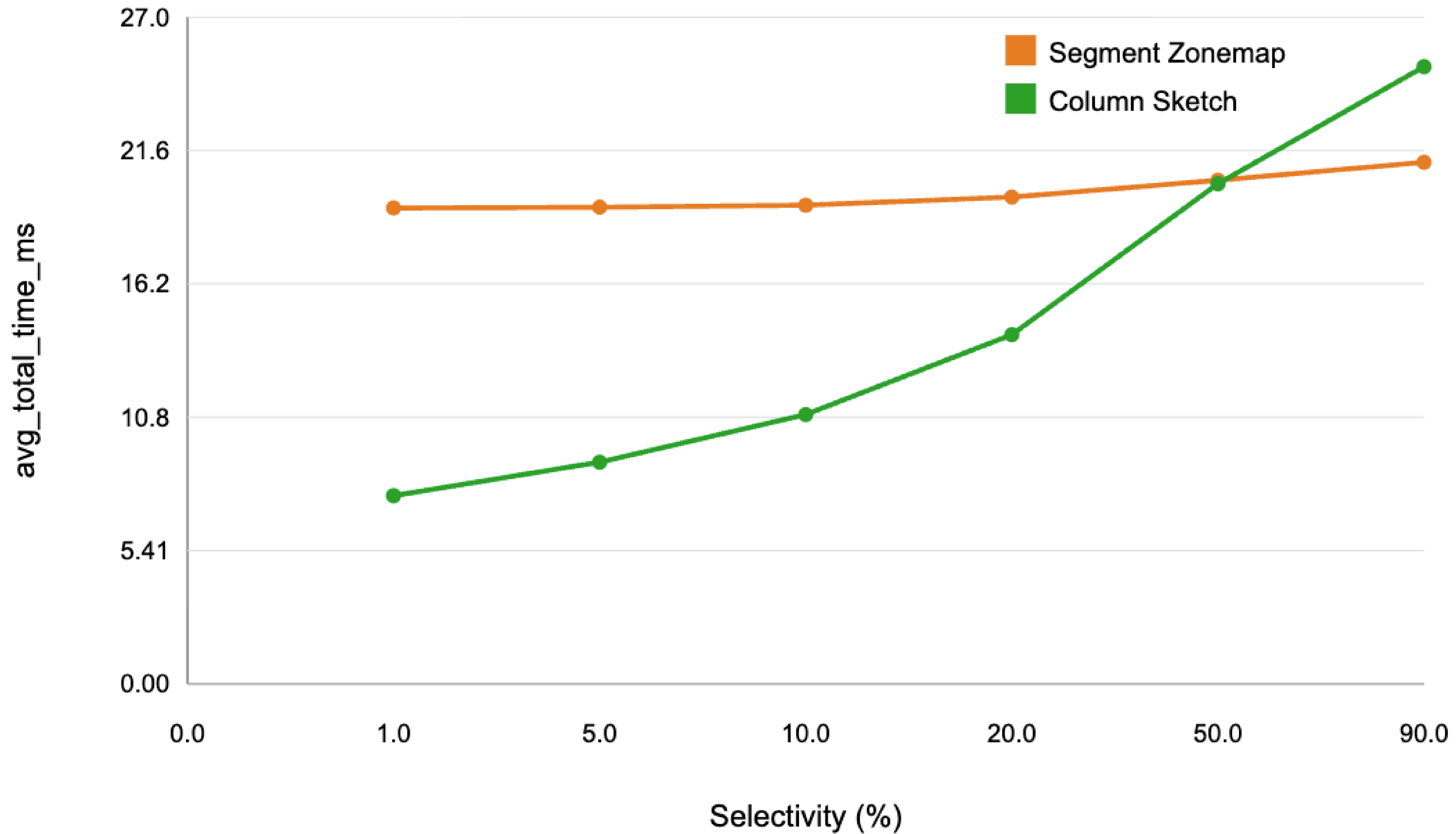
# Experiment 3: Crossover Analysis

**Now, we set selectivity as a variable**

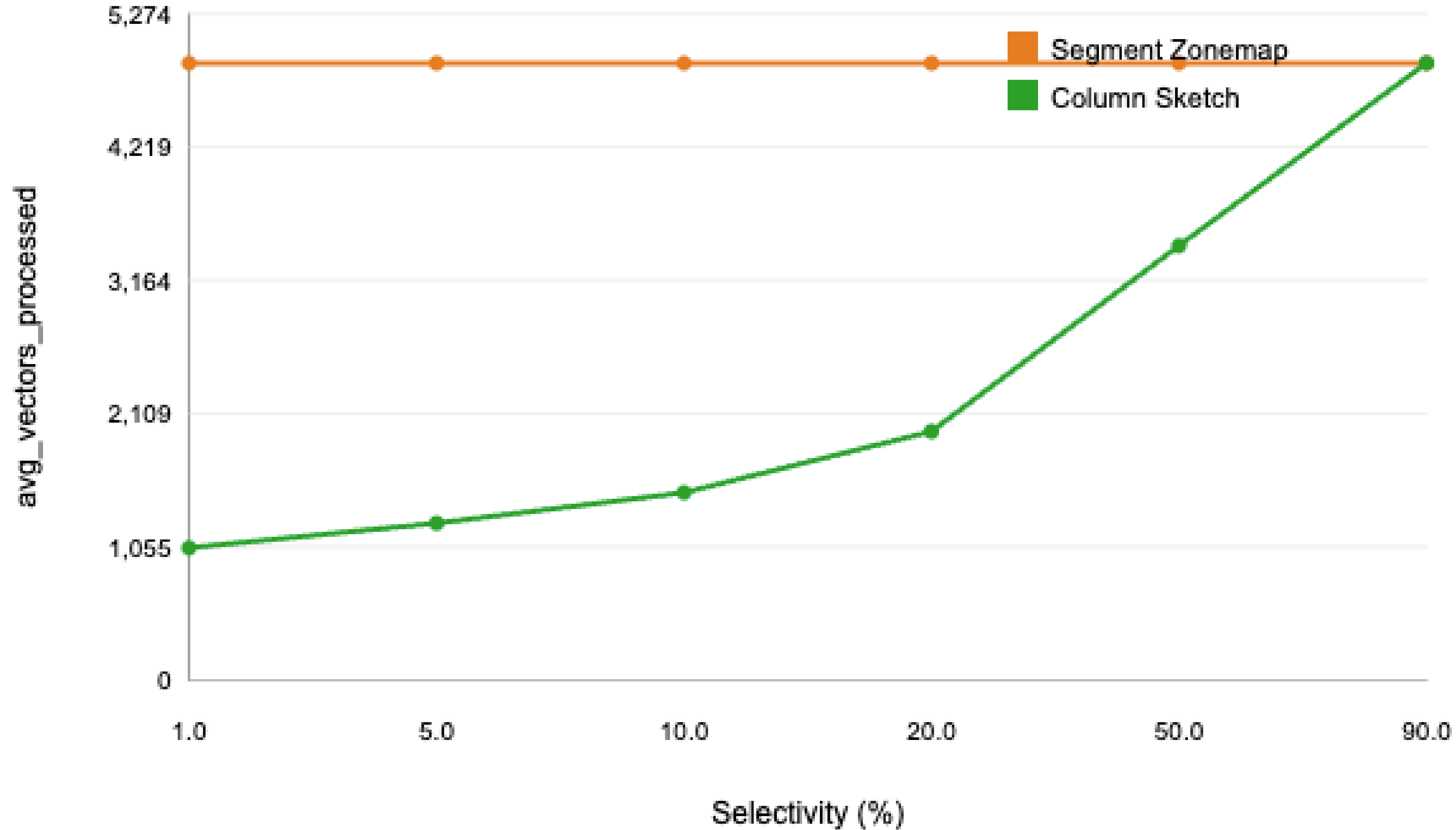
- **Vary selectivity under dataset of 10,000 distinct values**

**Goal: find the selectivity threshold where sketch changes from win → tie → loss**

# Selectivity vs Latency



# Selectivity vs Vectors Processed



# Key Observation:

**At 1–20% selectivity:**

- **Sketch processes far fewer vectors and is faster.**

**Around 50%:**

- **Sketch and zonemap are close.**

**At 90%:**

- **Sketch processes about the same number of vectors and becomes slower.**

As selectivity increases, fewer segments can be rejected.
Sketch overhead stays, but pruning benefit shrinks.

**Low / moderate selectivity → Column Sketch**

**High selectivity → Segment Zonemap**

**Crossover region → needs cost-based decision**

# RABIT ANALYSIS

**Goal:**
compare which access path performs better under different layouts and selectivity levels

**Metric:**
query latency (average runtime over multiple runs)

# RABIT ANALYSIS

## Three Contenders

- **Traditional Bitmap:** basic boolean array scanning (a >= low & a <= high)

- **DuckDB Native Scan**: underlying C++ engine to directly execute SQL SELECT count(*)

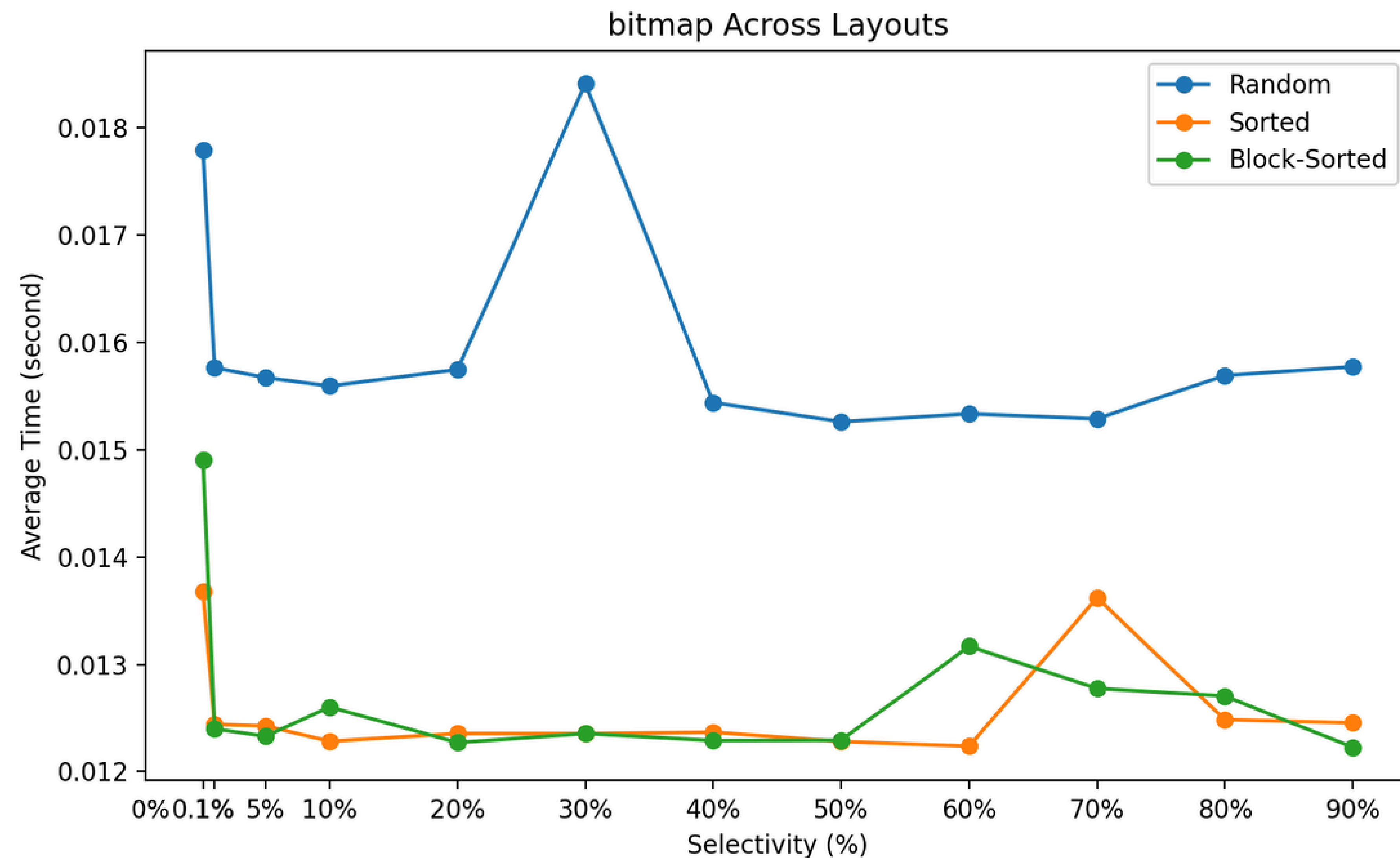
- **RABIT Algorithm:** pre-computes prefix bitmaps (~low_bitmap & high_bitmap)

# RABIT ANALYSIS

## Testing environment data credibility

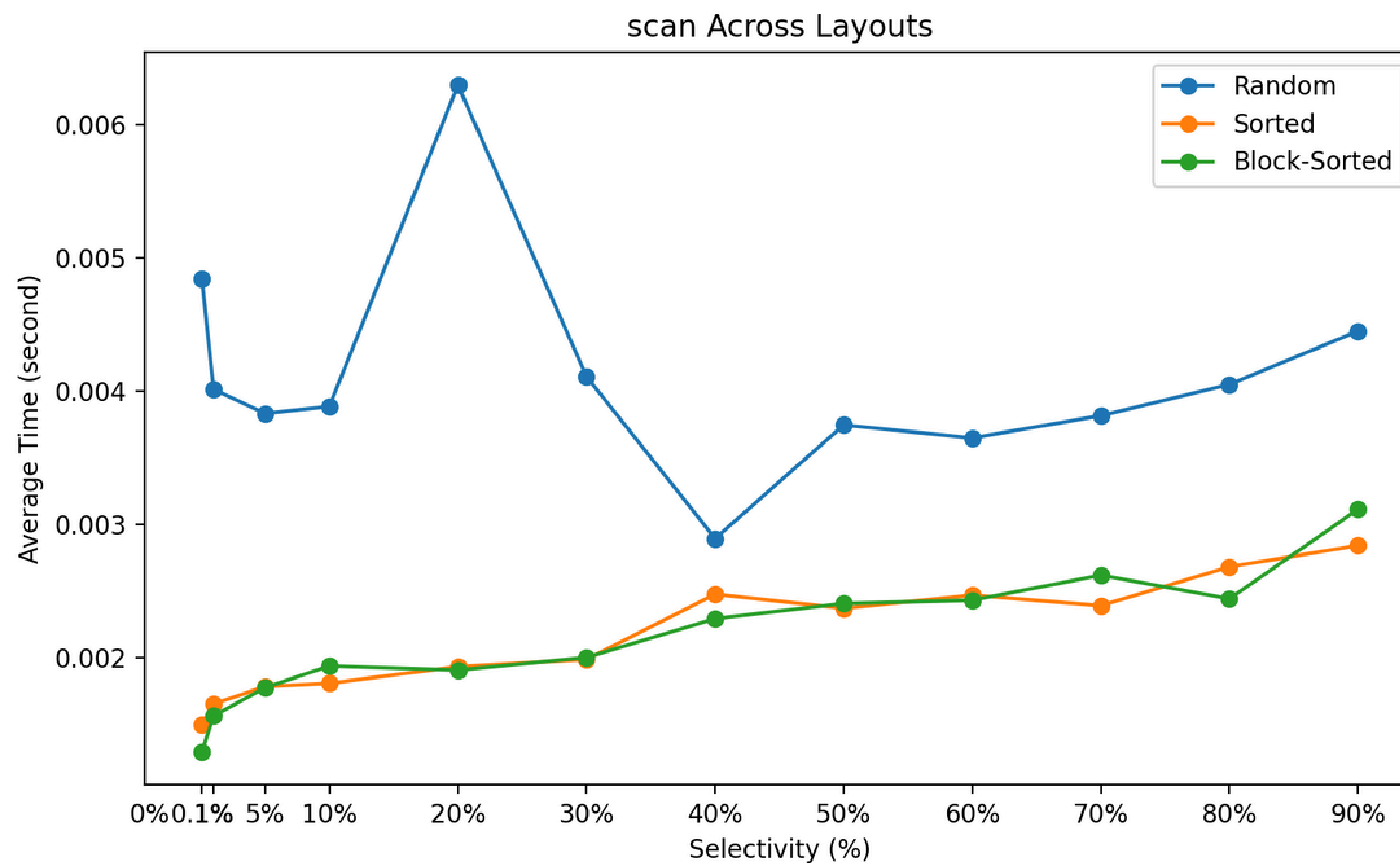
- **Data Layout:** We tested across three layouts: Random, Sorted, and Block-Sorted.
- **Selectivity:** The range of data we are querying, from 0.1% (finding a needle in a haystack) to 90%

# Bitmap



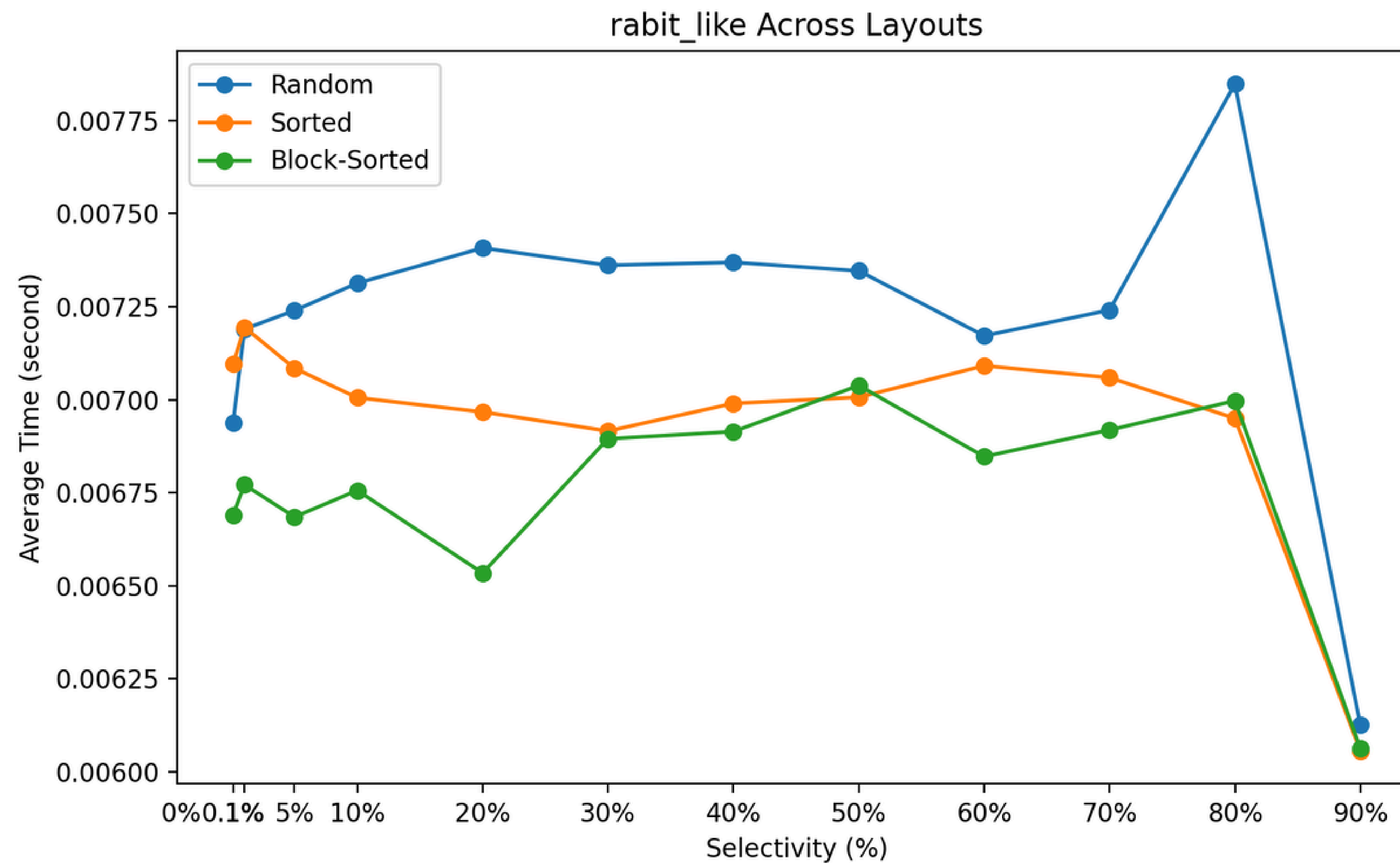

- The differences across layouts are smaller than in the scan case, although the random layout remains slower overall
- The average time ranges from 0.012 to 0.018

# SCAN



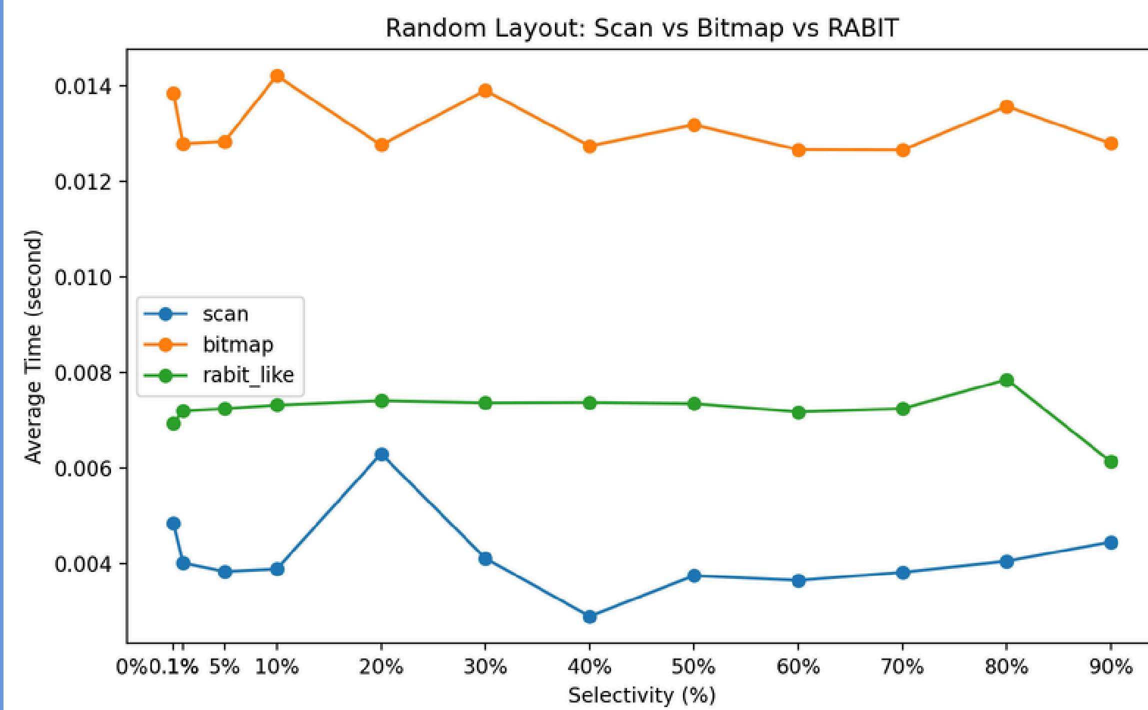

- Overall, the random layout has the highest scan latency
- The sorted and block-sorted layouts achieve lower average times
- scan latency generally increases as selectivity becomes larger

# RABIT

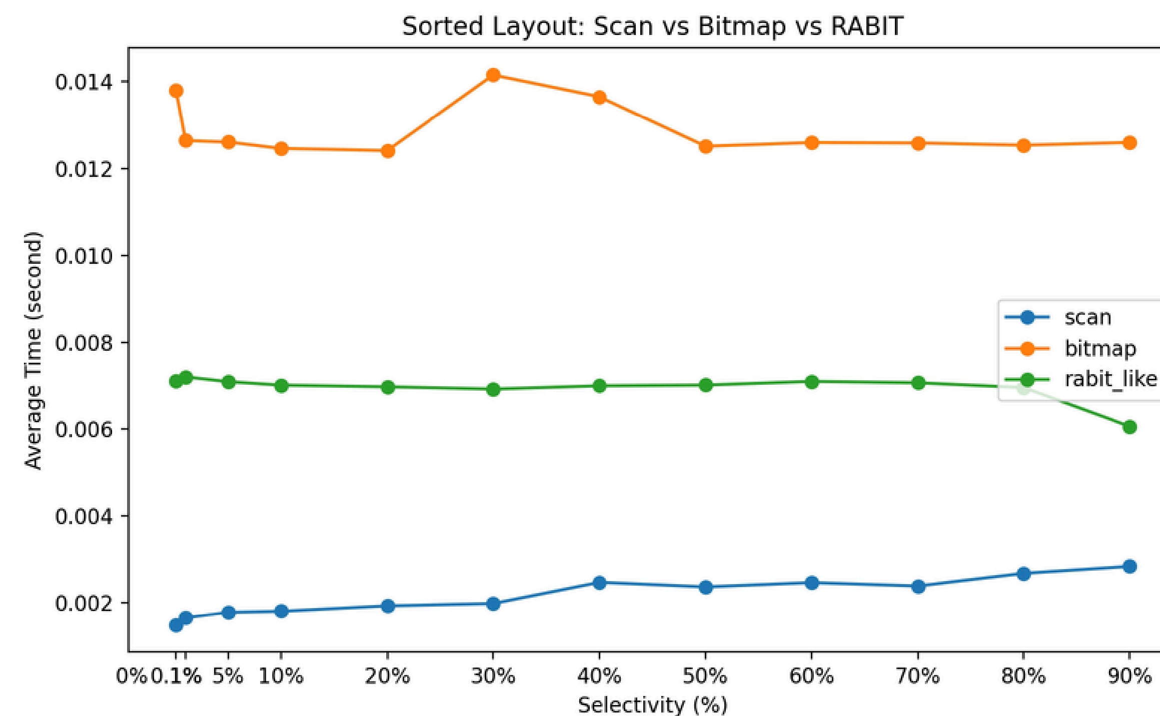



- At high selectivity, the runtime slightly decreases, likely because the bitmap pattern becomes more uniform.
- The random layout tends to be slower than sorted and block-sorted layouts

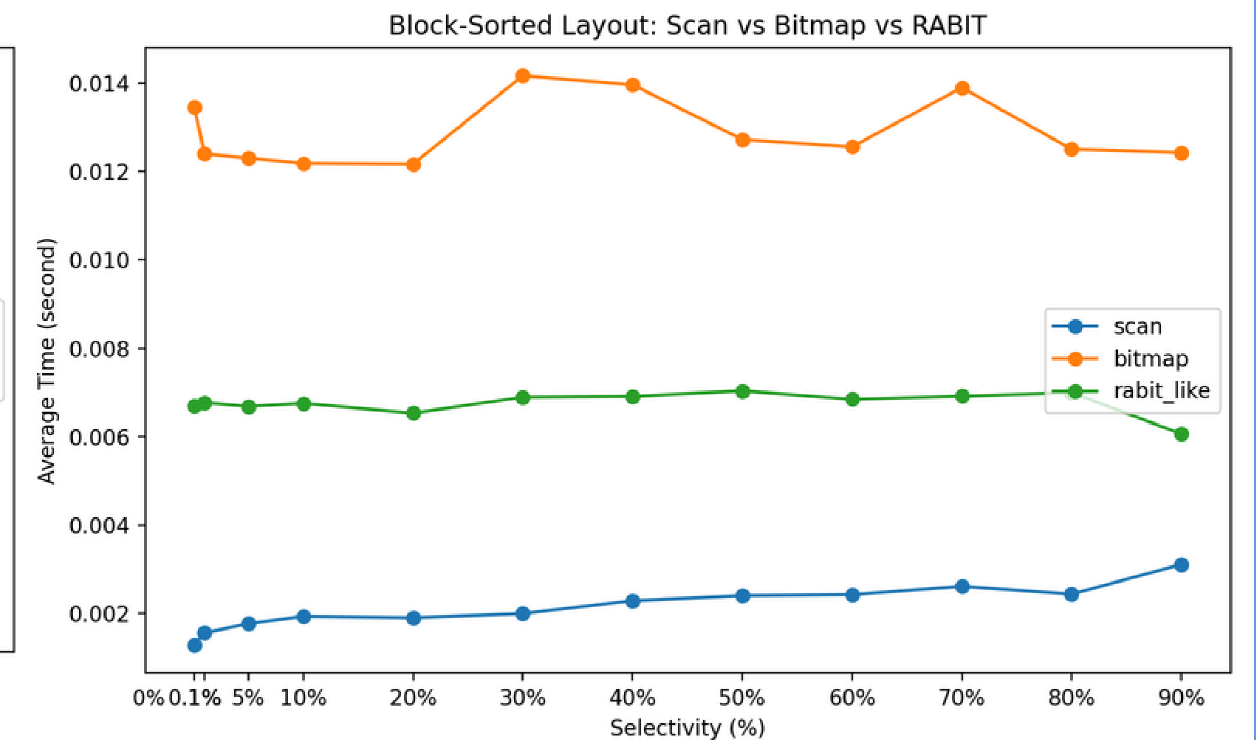
# SCAN vs. Bitmap vs. RABIT




random layout




sorted layout




block-sorted layout

# RABIT Analysis Conclusion

- Bitmap remains relatively flat because it scans the full array for each query
- RABIT gains speed by reusing prefix bitmaps instead of rebuilding range filters
- Sorted and block-sorted layouts improve performance by providing better data locality

# CONCLUSION

**Zonemap is cheap and stable**

- Best default when pruning opportunity is weak
- Works well when sketch overhead is not justified

**Column Sketch provides finer pruning**

- Can reduce vectors_processed significantly
- Wins when it eliminates enough scan work to offset sketch evaluation overhead

**RABIT shows algorithmic promise**

- Outperforms naive bitmap filtering
- But since our RABIT evaluation is Python-based, native DuckDB integration is needed for a fair engine-level comparison

# FUTURE WORK

- Implement native RABIT directly inside DuckDB
- Build automatic cost-based access path selection
- Test on larger real-world datasets
- Measure segment skip ratio and scan operator time
- Extend evaluation across more layouts and workloads

Questions?

# Access Paths Inside DuckDB

**Problem?**

- DuckDB normally makes scan-pruning decisions internally, so different pruning mechanisms can be mixed together
- This makes it hard to isolate which access  path caused a performance change.

**Our Goal**

Force controlled execution mode for scanning:

- **Row-group pruning via zonemaps + segment pruning via zonemap**
- **Row-group pruning via zonemaps+ segment pruning via column sketch**

# MODIFICATION: Control Flags

**DuckDB Internal Modifications**

- DuckDB automatically selects access paths.
- We needed controlled comparisons.

**Solution:**

Added configurable CLI flags:

- disable_sketch
- disable_zonemap

```
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
D SET disable_zonemap = true;
```

# MODIFICATION: More Metrics

**We needed additional metrics:**

- row groups pruned by zonemap
- column segments pruned by segment zonemap
- column segments pruned by column sketch
- vectors processed

**Solution:**

- Added scan-level counters inside the pruning paths:

```
SEQ_SCAN
- - - - - - - - - - -
t_layout_block_sorted
row_groups_pruned_by_zonema
p: 81
segments_pruned_by_zonemap:
1
segments_pruned_by_sketch:
0
vectors_processed: 32
- - - - - - - - - - -
- - - - - - - - - - -
Filters: a>=100000 AND a<
=100999 AND a IS NOT NULL
- - - - - - - - - - -
EC: 2000000
- - - - - - - - - - -
10000
(0.00s)
```

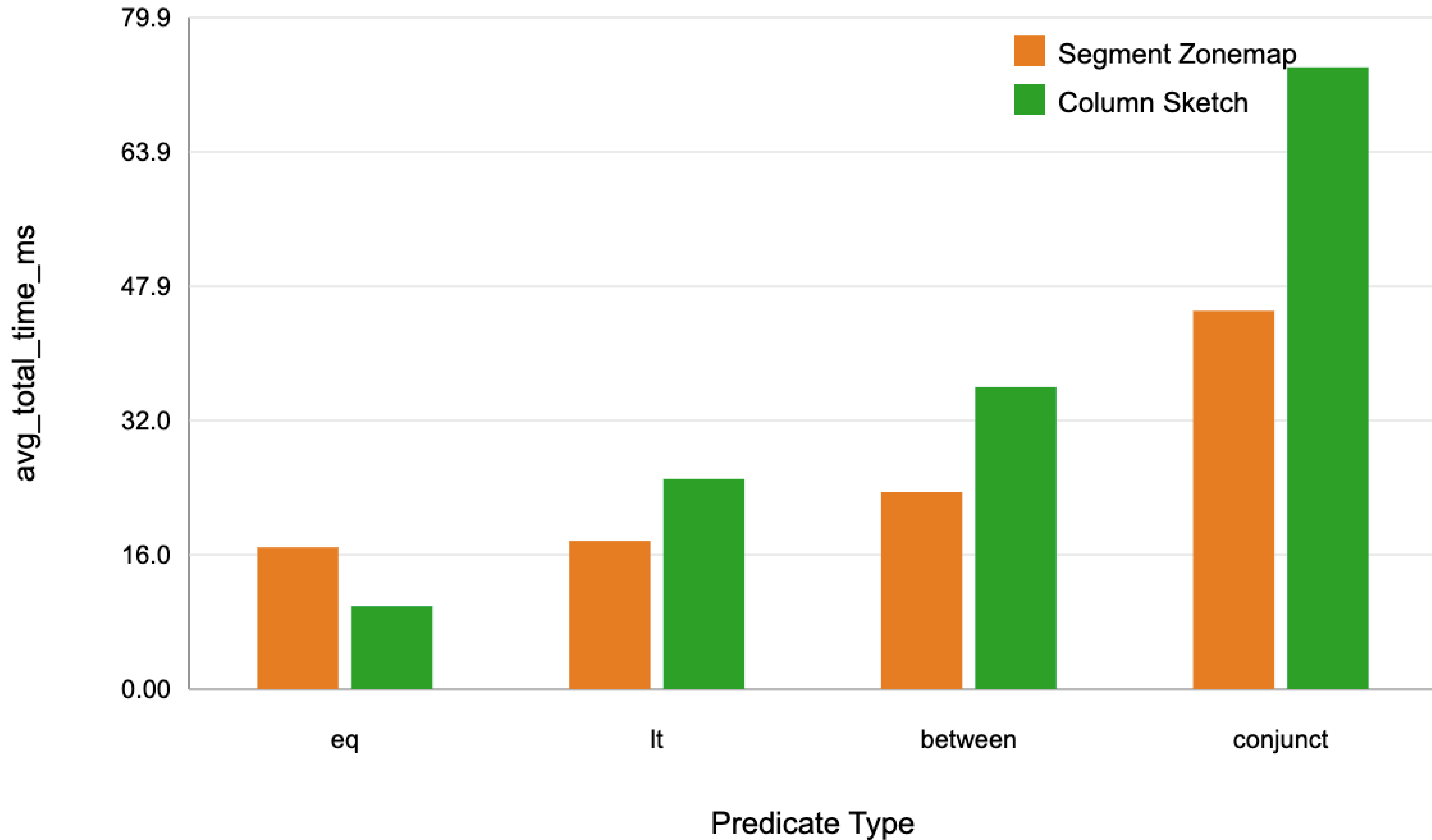
# Experiment 3: Predicate Type

Same workloads, different predicate forms:

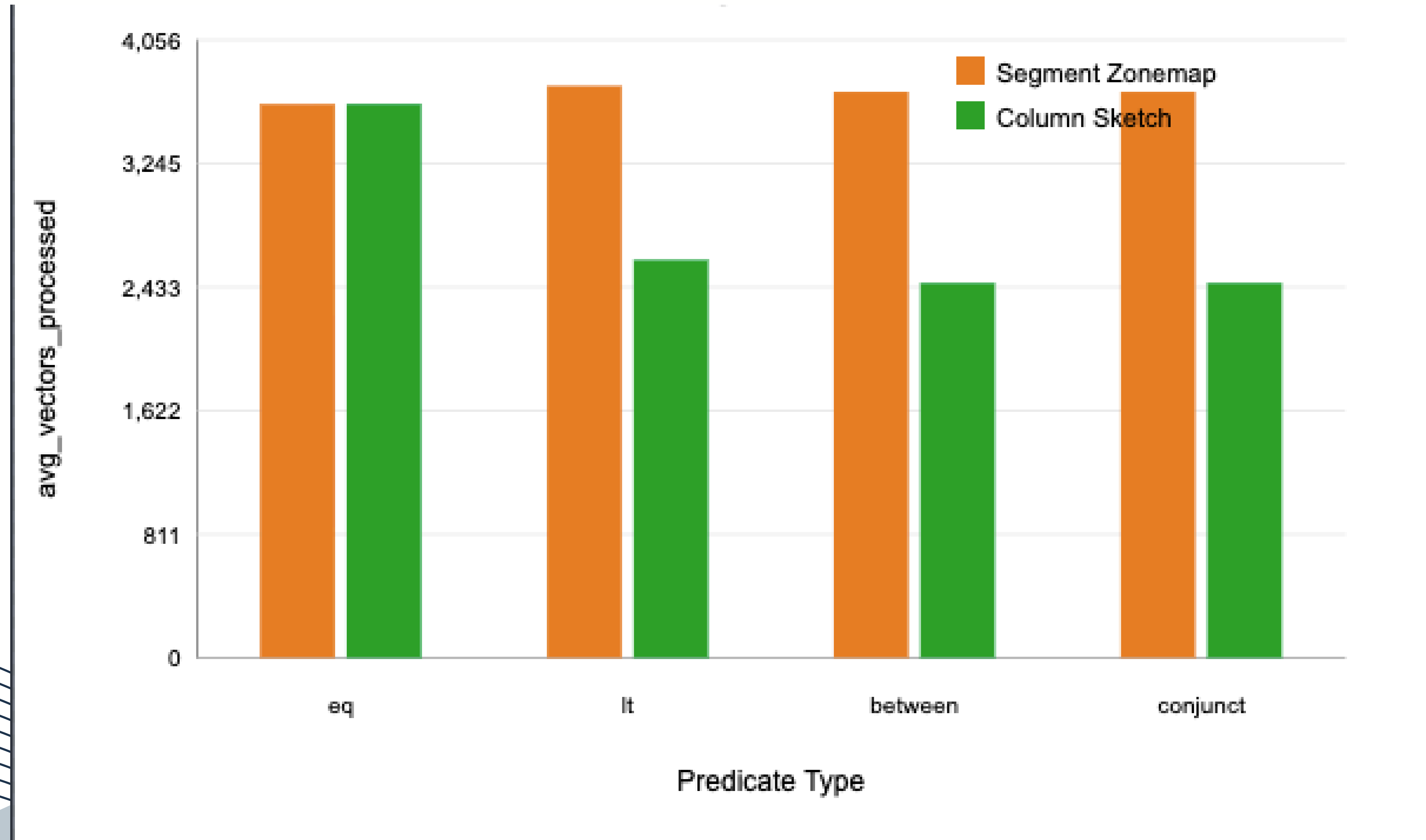
- equality
  - WHERE a = 42;
- less-than
  - WHERE a < 20;
- between
  - WHERE a BETWEEN 20 AND 29;
- conjunctive
  - WHERE a BETWEEN 20 AND 29 AND b >= 25.0;

**Goal: Understand which predicate changes pruning effectiveness**

# Predicate Vs. Latency



# Predicate vs Vectors Processed



# Key Observation:

**Equality:**

- **Vectors processed are similar; sketch is faster.**

**Range / conjunctive predicates:**

- **Sketch processes fewer vectors but is slower.**

**For range/conjunctive predicates, sketch reduces scan work, but extra sketch/predicate evaluation dominates.**

**Do not choose sketch only because it reduces vectors processed**

**For complex predicates, Segment Zonemap may be safer unless sketch gives a large latency win.**