Column Sketches: A Scan Accelerator for Rapid and Robust Predicate Evaluation

Alexander Beischl

December 3, 2018

Motivation

Evaluating a Query

Flight Delays

Flight-Nr.	Delay
1	2
2	73
3	2
4	5
5	40
6	56
7	17
8	2
9	12
10	72
⋮	

1024 Tuples

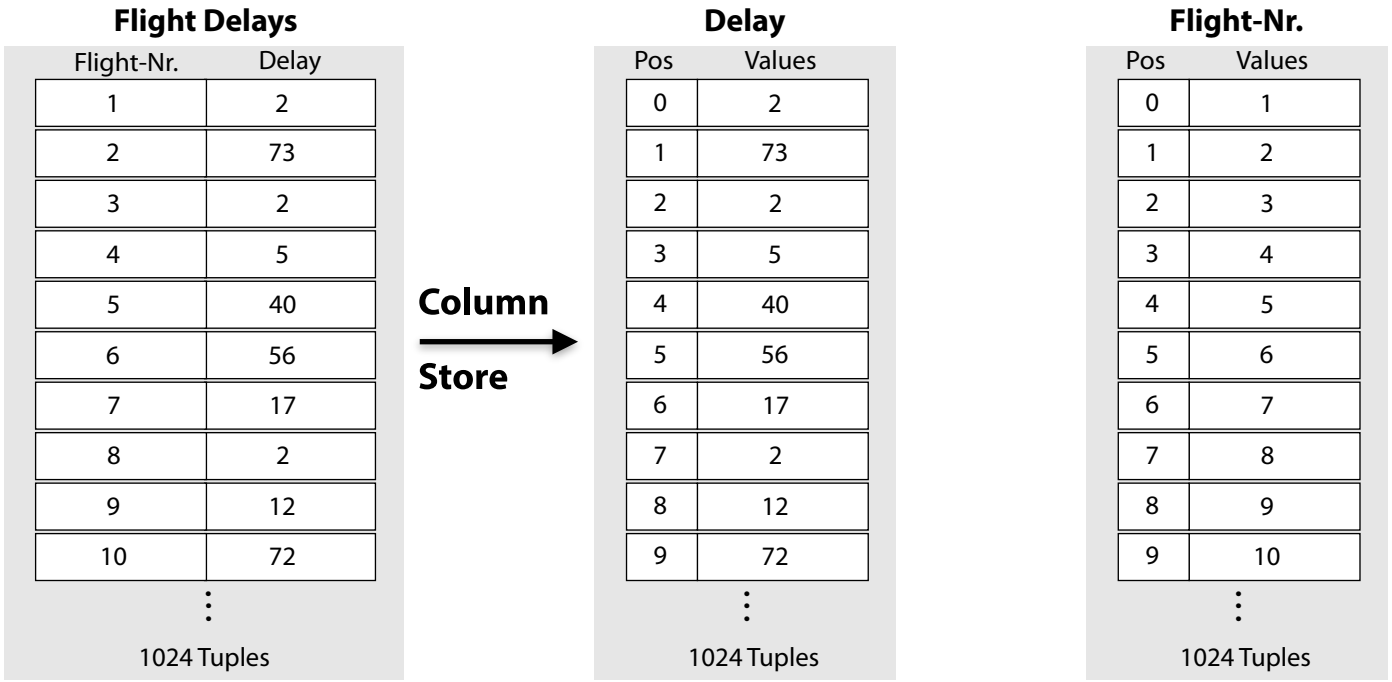
```
select  Flight-Nr.
from    Flight_Delays
where   Delay < 15
```

Result

Flight-Nr.
1
3
4
8
9
⋮

Naive Approach

Table Scan



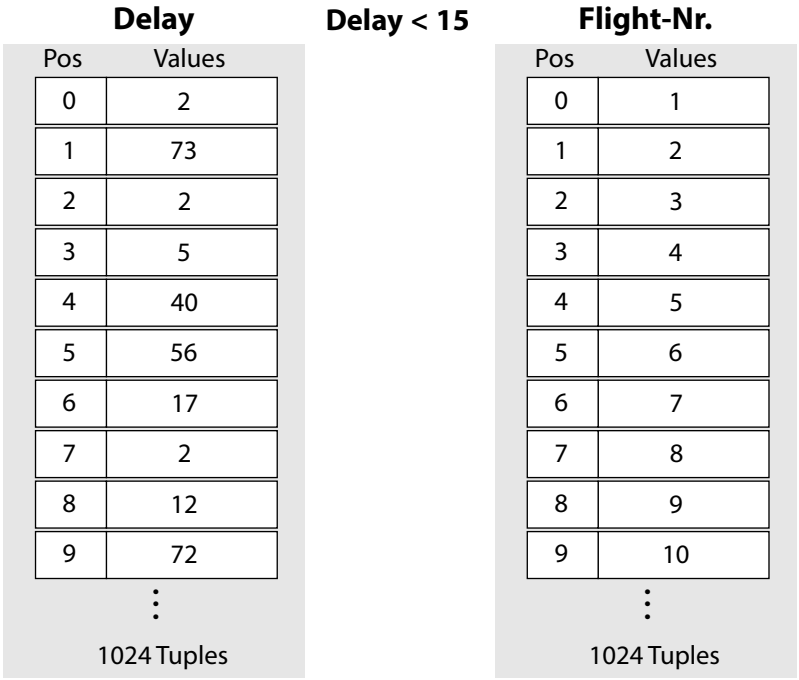
Naive Approach

Table Scan

select Flight-Nr.

from Flight_Delays

where Delay < 15



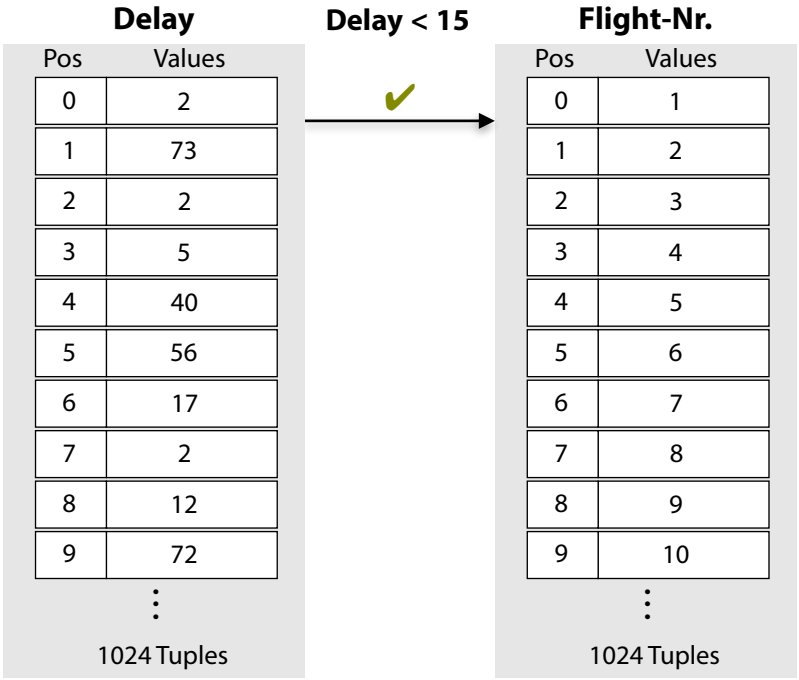
Naive Approach

Table Scan

select Flight-Nr.

from Flight_Delays

where Delay < 15



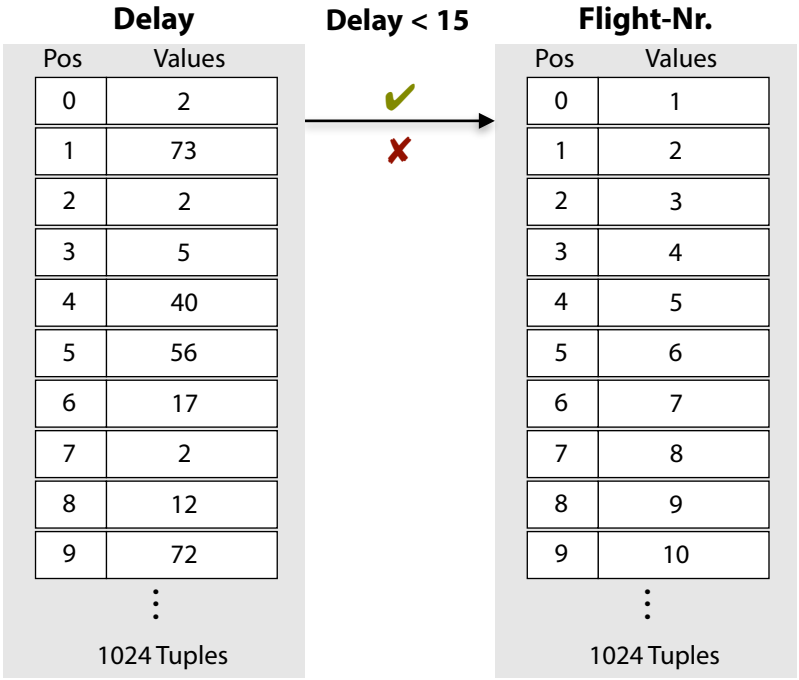
Naive Approach

Table Scan

select Flight-Nr.

from Flight_Delays

where Delay < 15



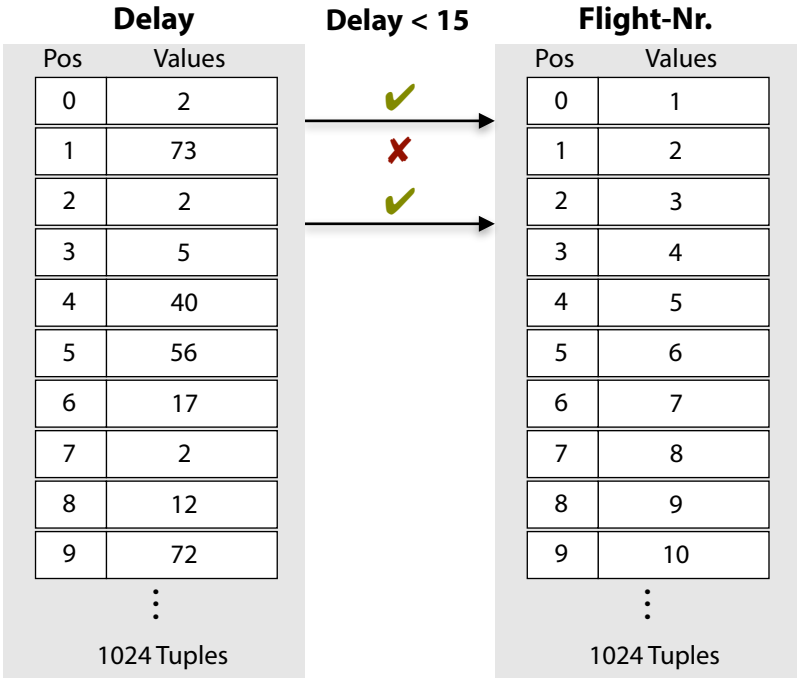
Naive Approach

Table Scan

select Flight-Nr.

from Flight_Delays

where Delay < 15



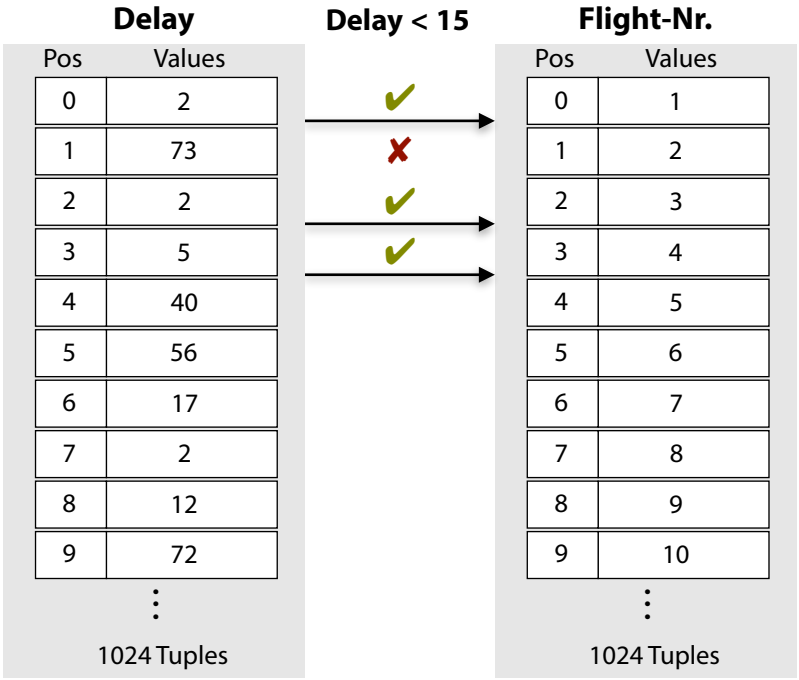
Naive Approach

Table Scan

select Flight-Nr.

from Flight_Delays

where Delay < 15



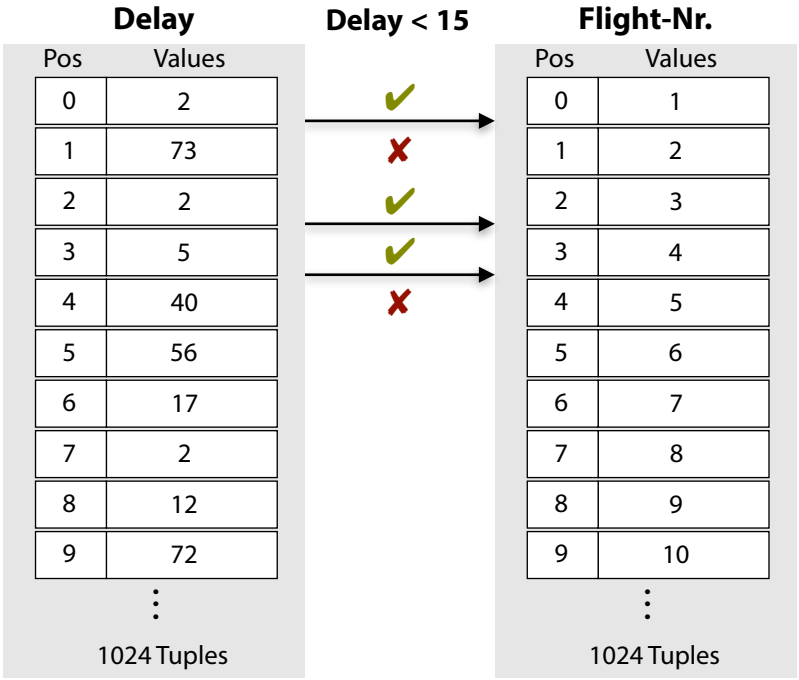
Naive Approach

Table Scan

select Flight-Nr.

from Flight_Delays

where Delay < 15



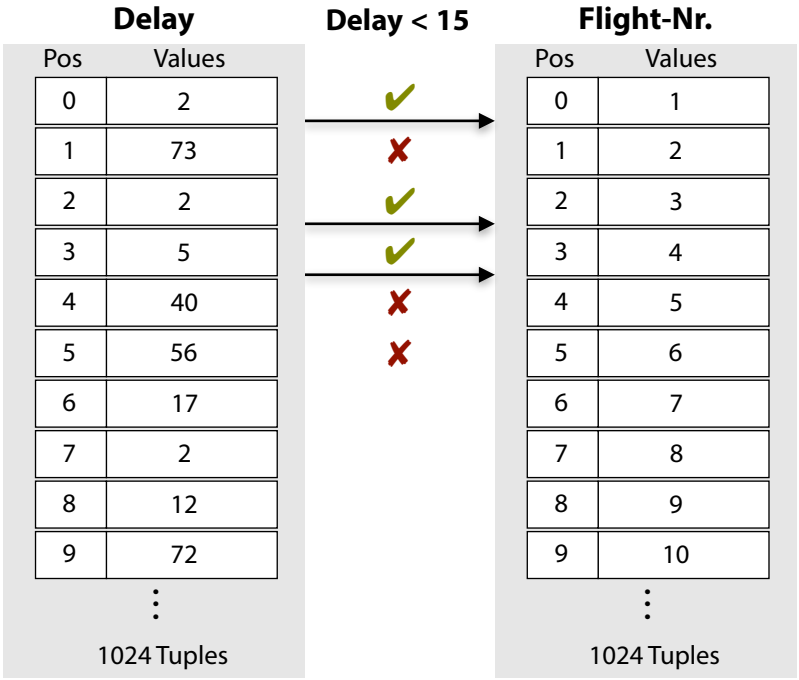
Naive Approach

Table Scan

select Flight-Nr.

from Flight_Delays

where Delay < 15



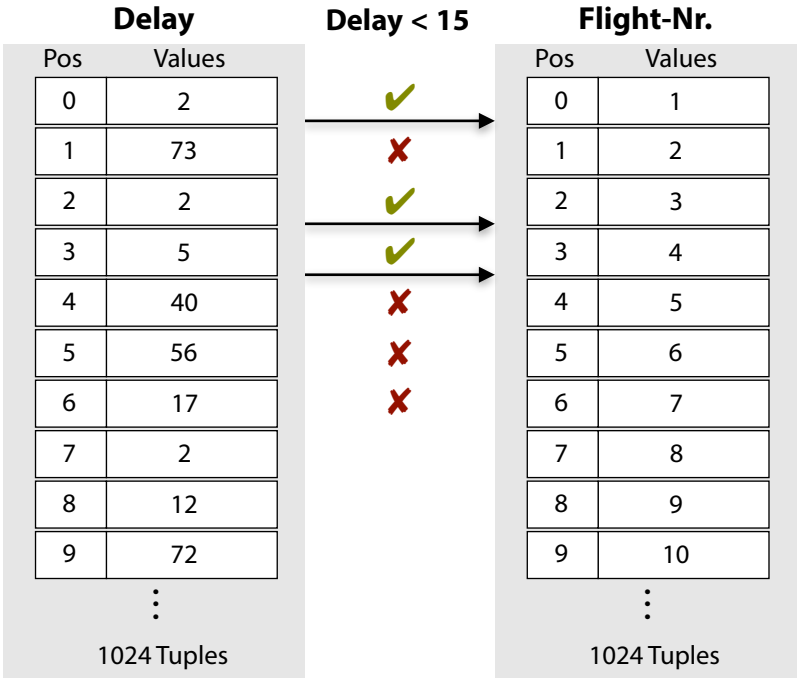
Naive Approach

Table Scan

select Flight-Nr.

from Flight_Delays

where Delay < 15



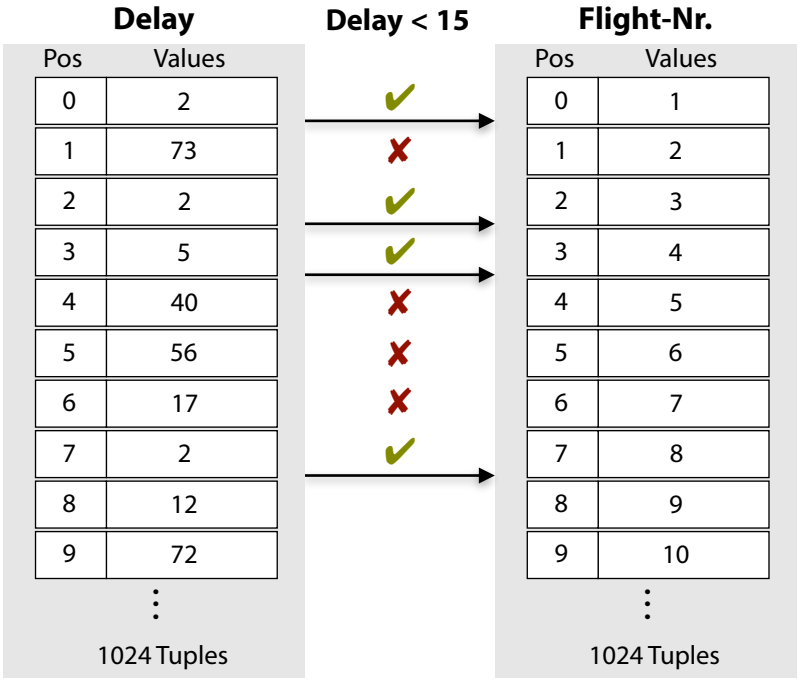
Naive Approach

Table Scan

select Flight-Nr.

from Flight_Delays

where Delay < 15



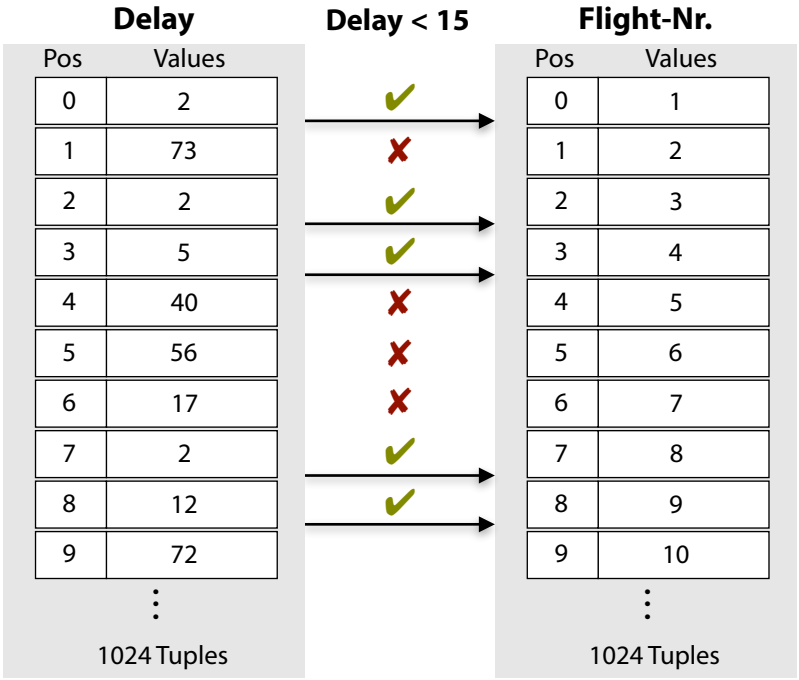
Naive Approach

Table Scan

select Flight-Nr.

from Flight_Delays

where Delay < 15



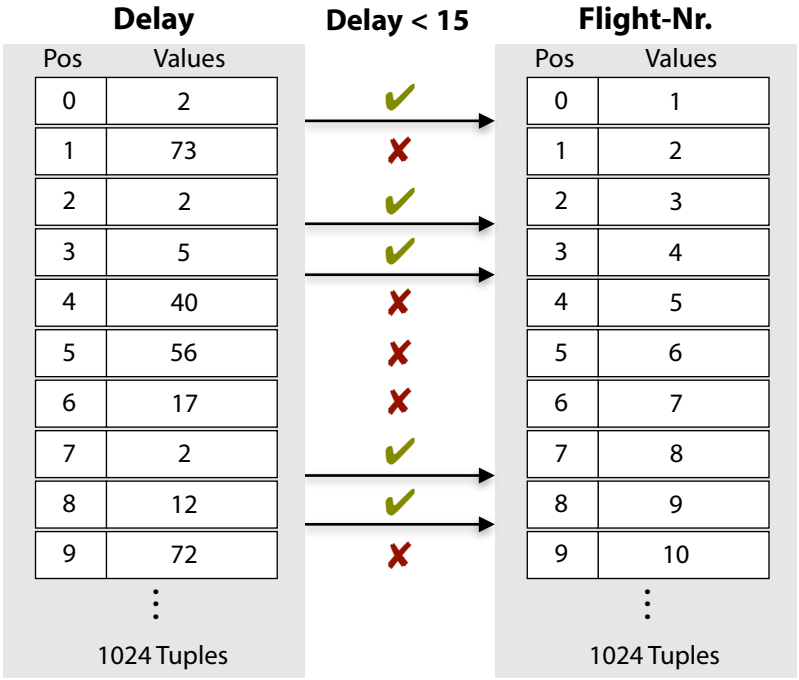
Naive Approach

Table Scan

select Flight-Nr.

from Flight_Delays

where Delay < 15



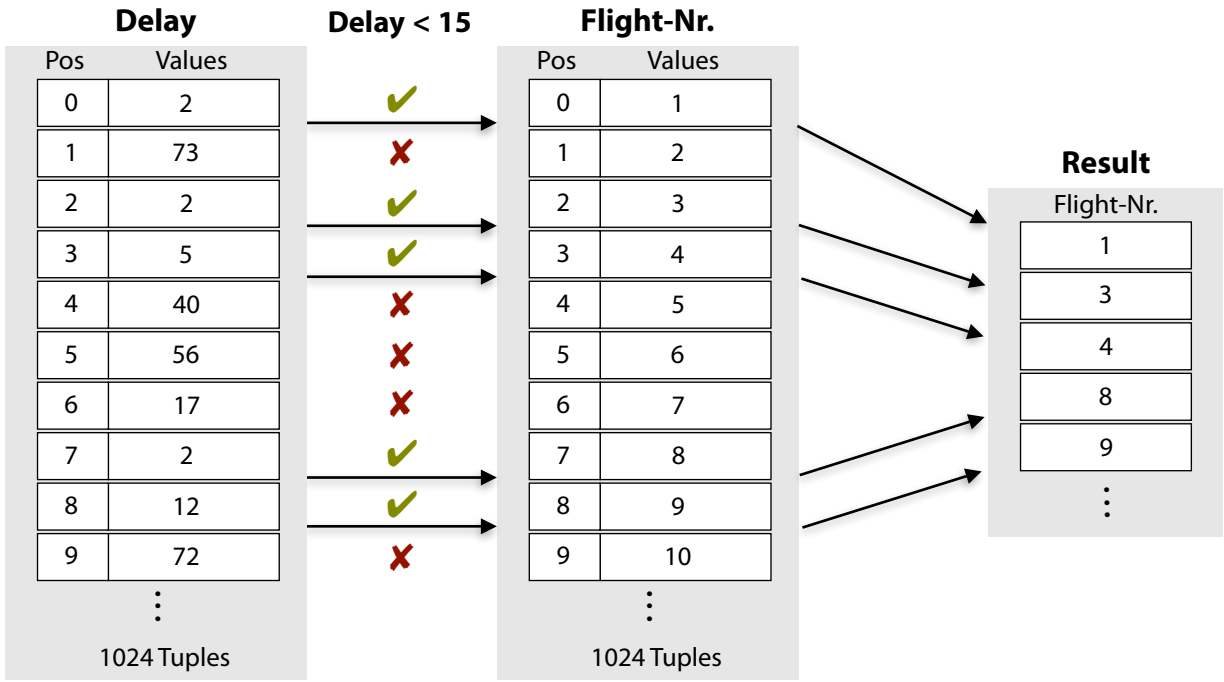
Naive Approach

Table Scan

select Flight-Nr.

from Flight_Delays

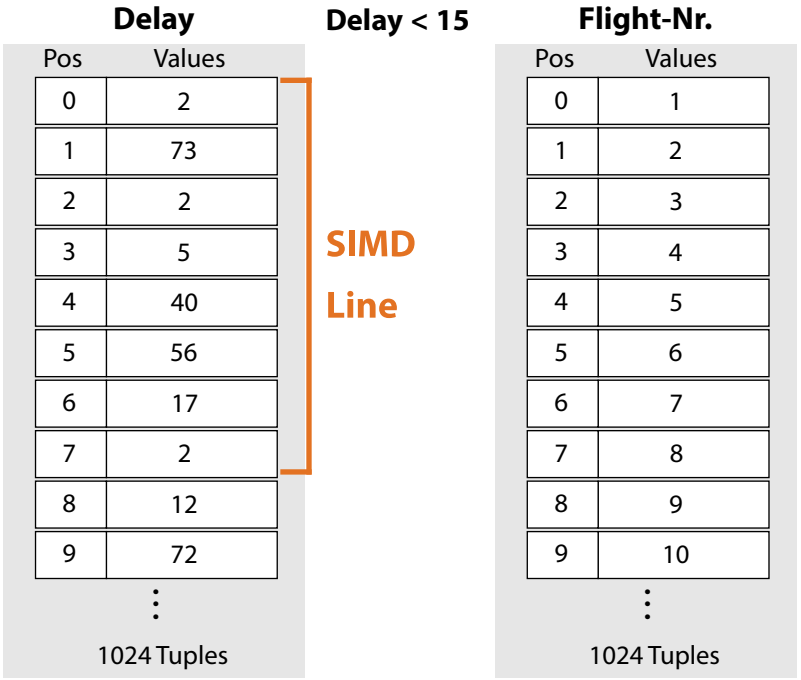
where Delay < 15



Naive Approach

Table Scan with SIMD

select Flight-Nr.
from Flight_Delays
where Delay < 15



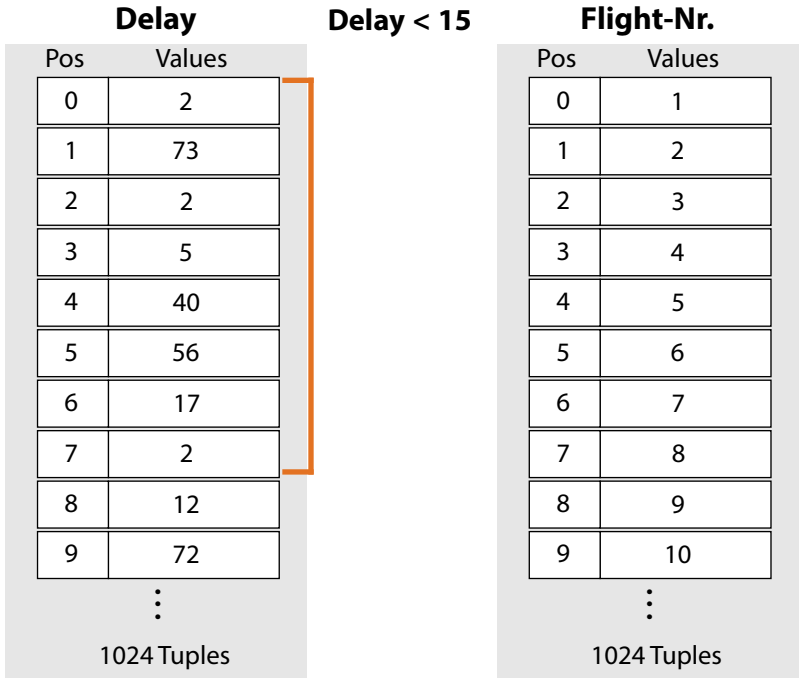
Naive Approach

Table Scan with SIMD

select Flight-Nr.

from Flight_Delays

where Delay < 15



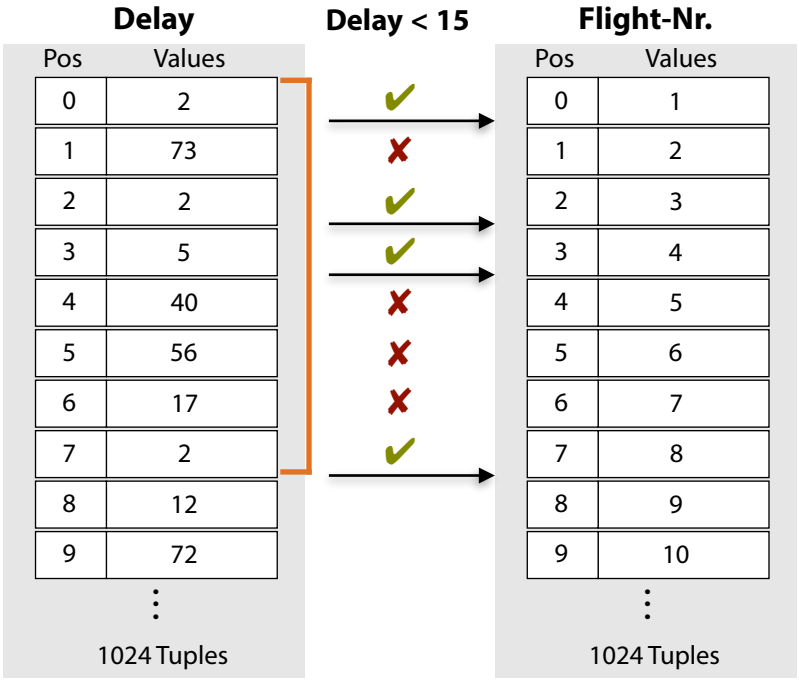
Naive Approach

Table Scan with SIMD

select Flight-Nr.

from Flight_Delays

where Delay < 15



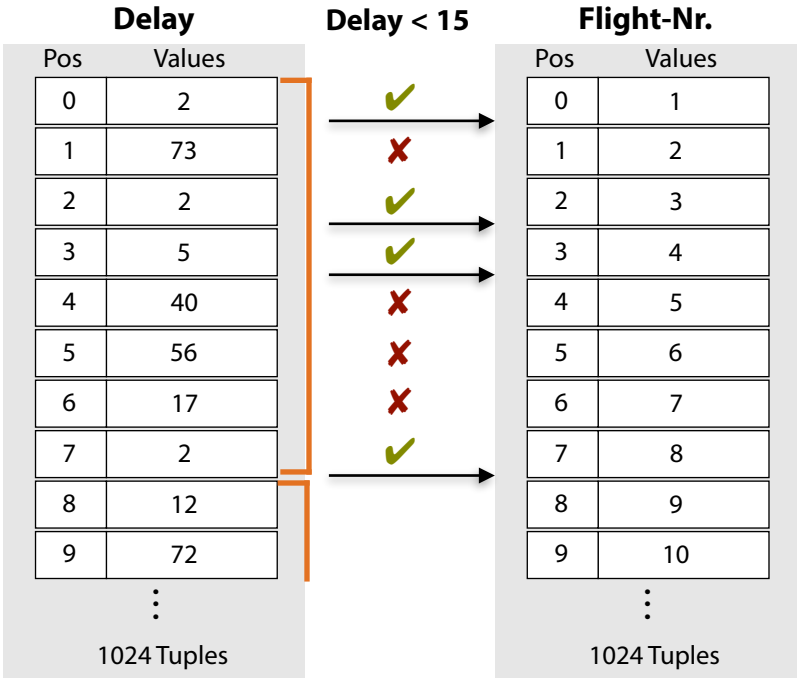
Naive Approach

Table Scan with SIMD

select Flight-Nr.

from Flight_Delays

where Delay < 15



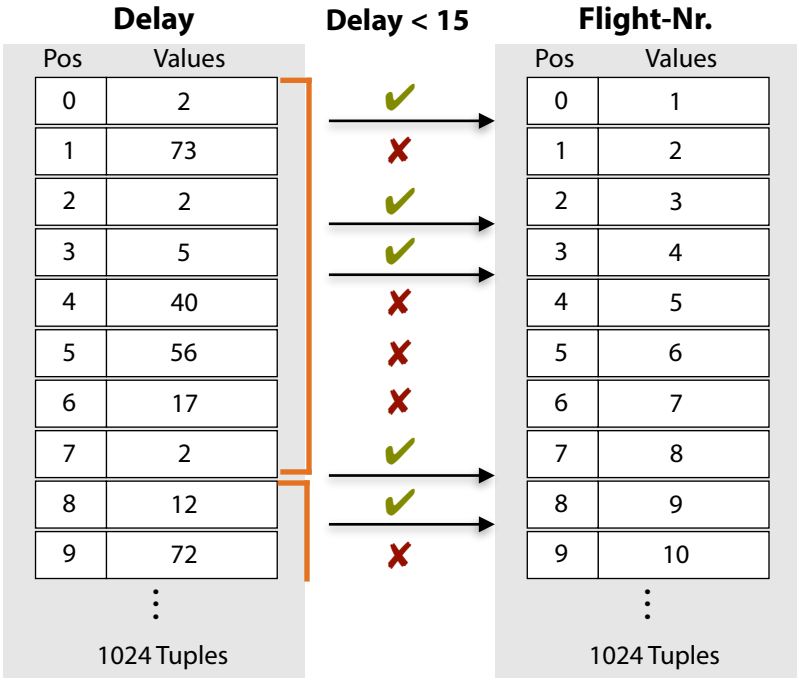
Naive Approach

Table Scan with SIMD

select Flight-Nr.

from Flight_Delays

where Delay < 15



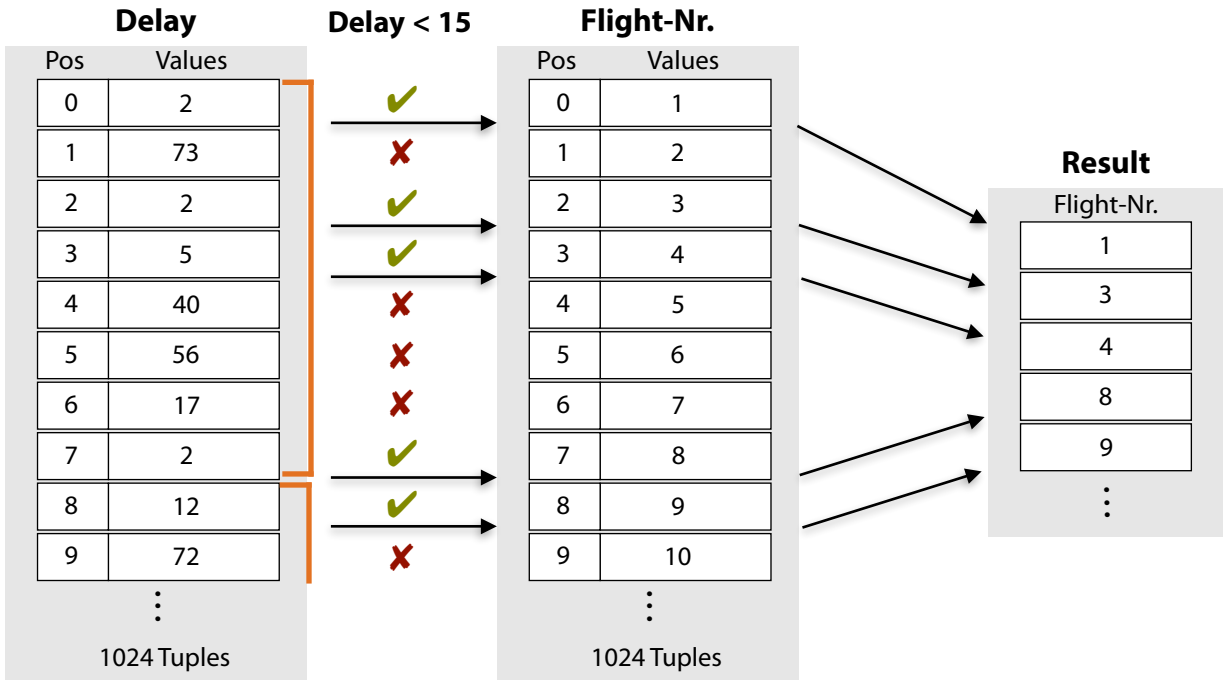
Naive Approach

Table Scan with SIMD

select Flight-Nr.

from Flight_Delays

where Delay < 15



More Complex Approach

Index Structures

- Auxiliary data structures to speed up queries

- Traditional index
  - i.e. B-Trees
  - Only efficient for very selective predicates (< 1%)

- Lightweight index structures
  - i.e. Column Imprints [3], BitWeaving/V [4], PSMAs[5]
  - Only efficient for data with clustering

➡ Only efficient for specific workloads

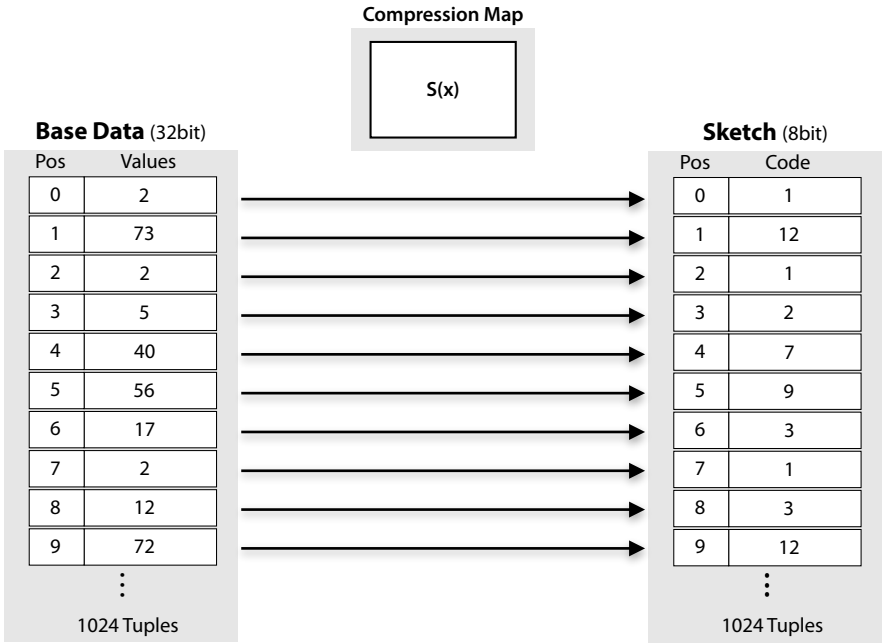
Column Sketches

Concept

Compression Map
- Maps values to codes in sketch
- Lossy compression

Sketch
- Small fixed-width codes
- Lookup
  I. Scan the sketch
  II. Only lookup base data if necessary

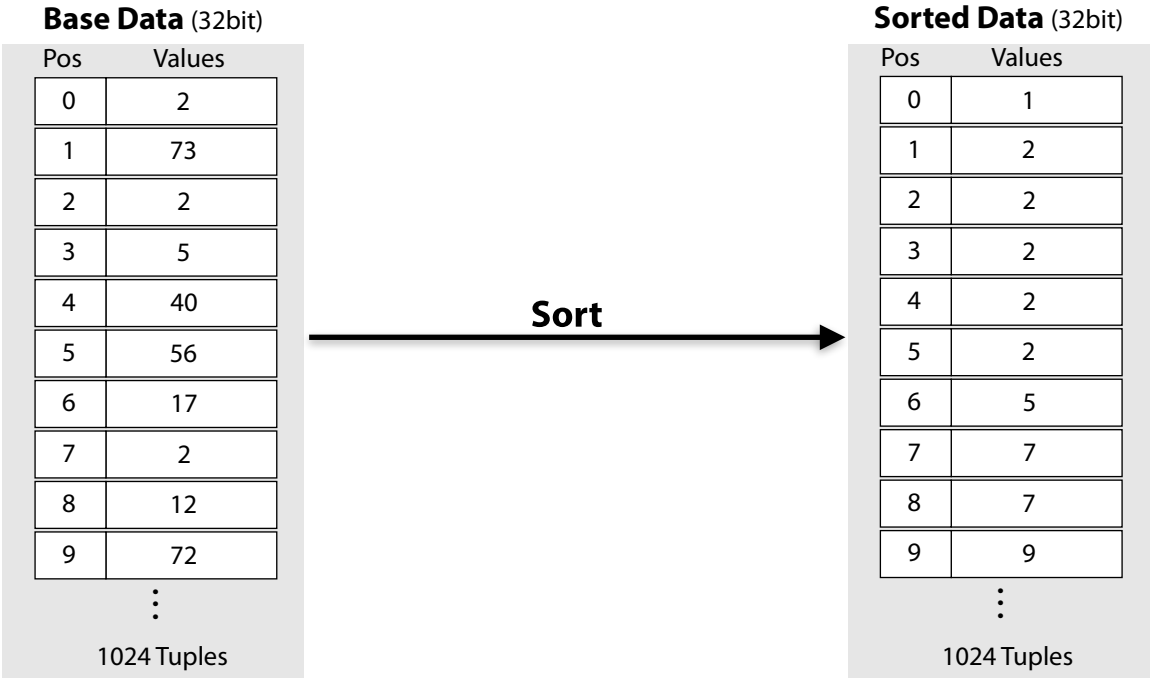
➡ Reduced data movement costs



Column Sketches

Order Preserving - Build Compression Map

I. Sort the data temporarily



Column Sketches

Order Preserving - Build Compression Map

II. Find frequent values

1. Check values at pos:

$$\frac{n}{c}, \frac{2n}{c}, \ldots, \frac{(c-1)n}{c}$$

c = #codes, n = #tuples

2. Search first & last occurrence

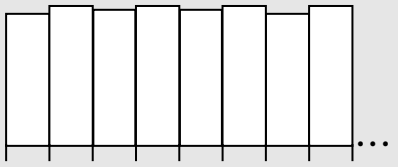
3. Frequency $> \frac{n}{c}$:
   - Bin: $c * \frac{midpoint}{n}$
   - Set bin unique

Sorted Data (32bit)

Pos	Values
0	1
1	2
2	2
3	2
4	2
5	2
6	5
7	7
8	7
9	9
⋮	

1024 Tuples

Compression Map



Histogram (256 bins)

Column Sketches

Order Preserving - Build Compression Map

II. Find frequent values

1. Check values at pos:

   $\frac{n}{c}, \frac{2n}{c}, \ldots, \frac{(c-1)n}{c}$

   c = #codes, n = #tuples

2. Search first & last occurrence

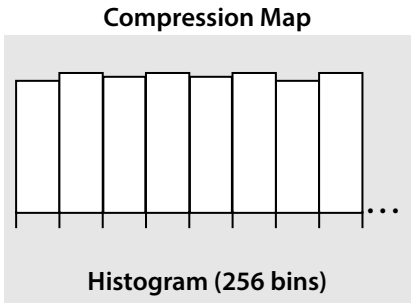
3. Frequency $> \frac{n}{c}$:
   - Bin: $c * \frac{midpoint}{n}$
   - Set bin unique

Sorted Data (32bit)

Pos	Values
0	1
1	2
2	2
3	2
4	2
5	2
6	5
7	7
8	7
9	9
⋮	

1024 Tuples

← check $\frac{n}{c} = 4$



Column Sketches

Order Preserving - Build Compression Map

II. Find frequent values

1. Check values at pos:

$$\frac{n}{c}, \frac{2n}{c}, \ldots, \frac{(c-1)n}{c}$$

c = #codes, n = #tuples

2. Search first & last occurrence

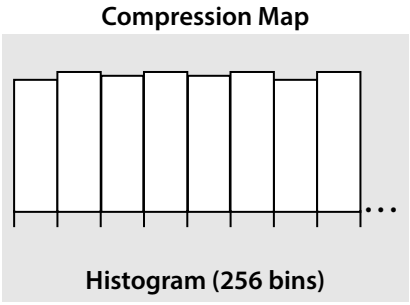
3. Frequency $> \frac{n}{c}$:
   - Bin: $c * \frac{midpoint}{n}$
   - Set bin unique

Sorted Data (32bit)

Pos	Values
0	1
1	2
2	2
3	2
4	2
5	2
6	5
7	7
8	7
9	9
⋮	

1024 Tuples

← frequency = 5 > 4



Column Sketches

Order Preserving - Build Compression Map

II. Find frequent values

1. Check values at pos:

$$\frac{n}{c}, \frac{2n}{c}, \ldots, \frac{(c-1)n}{c}$$

c = #codes, n = #tuples

2. Search first & last occurrence

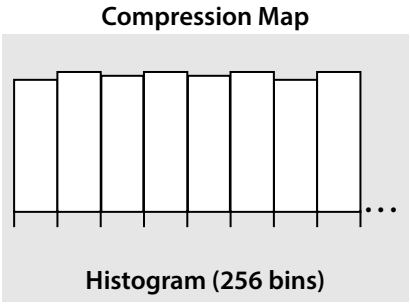
3. Frequency $> \frac{n}{c}$:
   - Bin: $c * \frac{midpoint}{n}$
   - Set bin unique

Sorted Data (32bit)

Pos	Values
0	1
1	2
2	2
3	2
4	2
5	2
6	5
7	7
8	7
9	9
⋮	

1024 Tuples

← midpoint = 3
bin = 0.75 = 1



Column Sketches

Order Preserving - Build Compression Map

II. Find frequent values

1. Check values at pos:

$$\frac{n}{c}, \frac{2n}{c}, \ldots, \frac{(c-1)n}{c}$$

c = #codes, n = #tuples

2. Search first & last occurrence

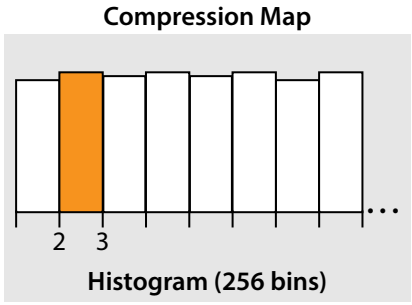
3. Frequency $> \frac{n}{c}$:
   - Bin: $c * \frac{midpoint}{n}$
   - Set bin unique

Sorted Data (32bit)

Pos	Values
0	1
1	2
2	2
3	2
4	2
5	2
6	5
7	7
8	7
9	9
⋮	

1024 Tuples

← midpoint = 3
bin = 0.75 = 1



Column Sketches

Order Preserving - Build Compression Map

II. Find frequent values

1. Check values at pos:

   $\frac{n}{c}, \frac{2n}{c}, \ldots, \frac{(c-1)n}{c}$

   c = #codes, n = #tuples

2. Search first & last occurrence

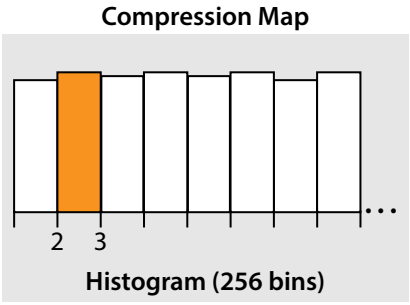
3. Frequency $> \frac{n}{c}$:
   - Bin: $c * \frac{midpoint}{n}$
   - Set bin unique

Sorted Data (32bit)

Pos	Values
0	1
1	2
2	2
3	2
4	2
5	2
6	5
7	7
8	7
9	9
⋮	

1024 Tuples

← check $\frac{2n}{c} = 8$



Column Sketches

Order Preserving - Build Compression Map

II. Find frequent values

1. Check values at pos:

$$\frac{n}{c}, \frac{2n}{c}, \ldots, \frac{(c-1)n}{c}$$

c = #codes, n = #tuples

2. Search first & last occurrence

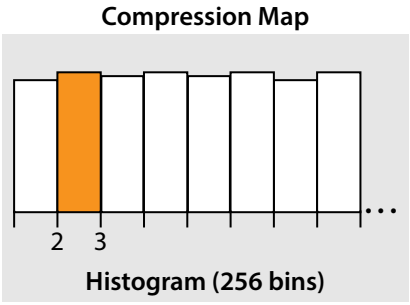
3. Frequency $> \frac{n}{c}$:
   - Bin: $c * \frac{midpoint}{n}$
   - Set bin unique

Sorted Data (32bit)

Pos	Values
0	1
1	2
2	2
3	2
4	2
5	2
6	5
7	7
8	7
9	9

1024 Tuples

frequency = 2



Column Sketches

Order Preserving - Build Compression Map

II. Find frequent values

1. Check values at pos:

$$\frac{n}{c}, \frac{2n}{c}, \ldots, \frac{(c-1)n}{c}$$

c = #codes, n = #tuples

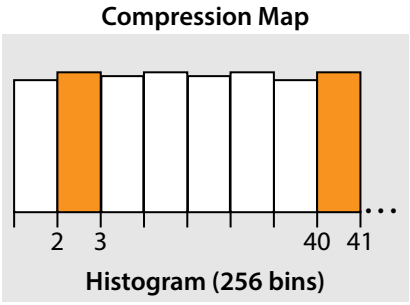
2. Search first & last occurrence

3. Frequency $> \frac{n}{c}$:
   - Bin: $c * \frac{midpoint}{n}$
   - Set bin unique

Sorted Data (32bit)

Pos	Values
0	1
1	2
2	2
3	2
4	2
5	2
6	5
7	7
8	7
9	9
⋮	

1024 Tuples



Column Sketches

Order Preserving - Build Compression Map

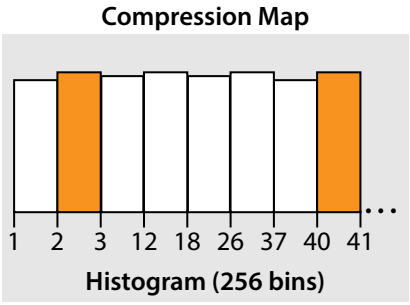
III. Fill empty bins

- per non-unique bin: at most $\frac{2n}{c}$ tuples

Sorted Data (32bit)

Pos	Values
0	1
1	2
2	2
3	2
4	2
5	2
6	5
7	7
8	7
9	9
⋮	

1024 Tuples



Column Sketches

Order Preserving - Build Compression Map

IV. Equi-depth histogram

Each bin either:

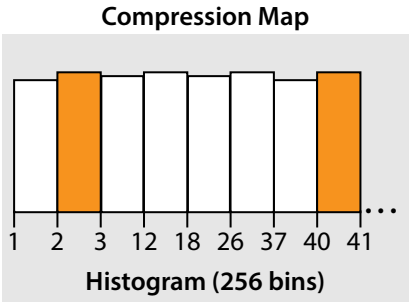
- unique:
  one value
  frequency > $\frac{n}{c}$

- non-unique:
  multiple values
  total frequency ≤ $\frac{2n}{c}$

Sorted Data (32bit)

Pos	Values
0	1
1	2
2	2
3	2
4	2
5	2
6	5
7	7
8	7
9	9
⋮	

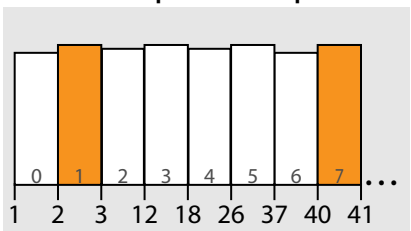
1024 Tuples



Column Sketches

Order Preserving - Build Sketch

Compression Map



Histogram (256 bins)

Base Data (32bit)

Pos	Values
0	2
1	73
2	2
3	5
4	40
5	56
6	17
7	2
8	12
9	72
⋮	

1024 Tuples

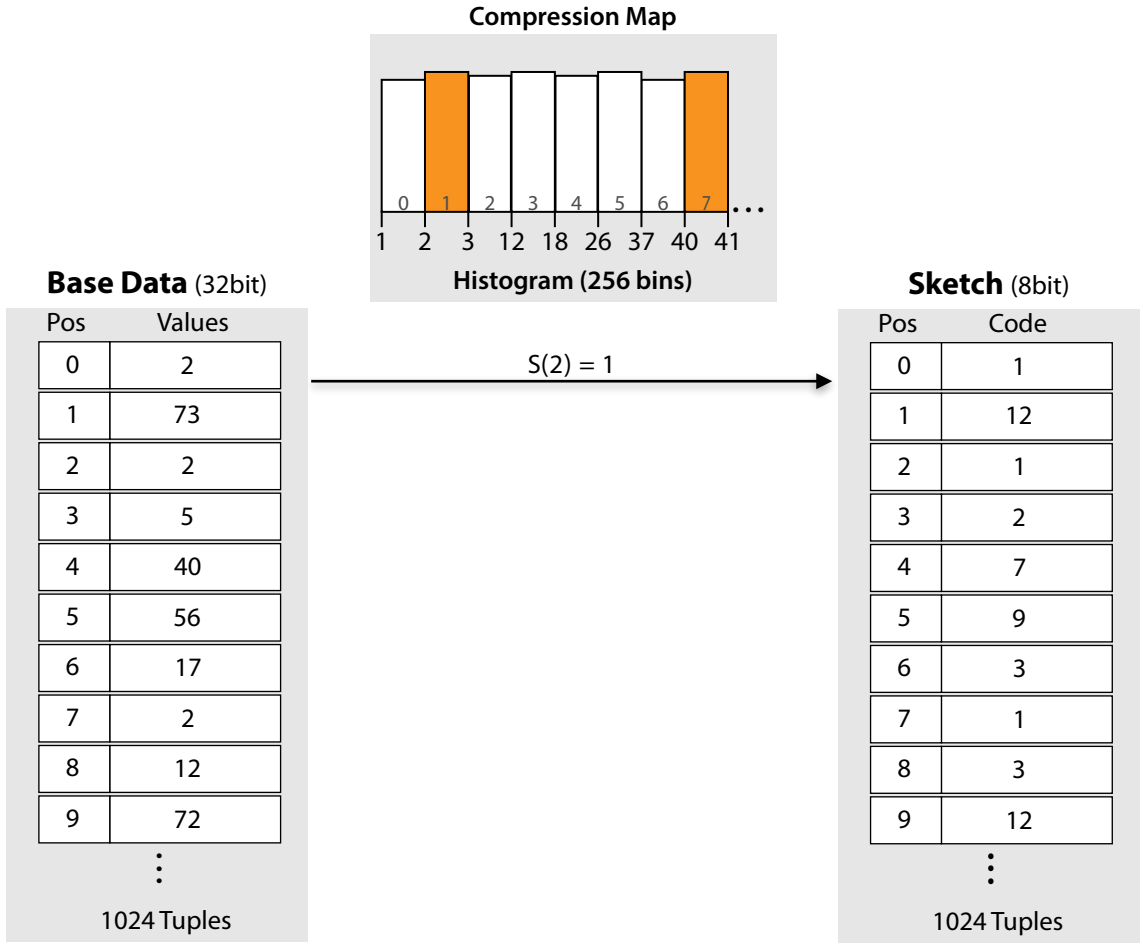
Sketch (8bit)

Pos	Code
0	1
1	12
2	1
3	2
4	7
5	9
6	3
7	1
8	3
9	12
⋮	

1024 Tuples

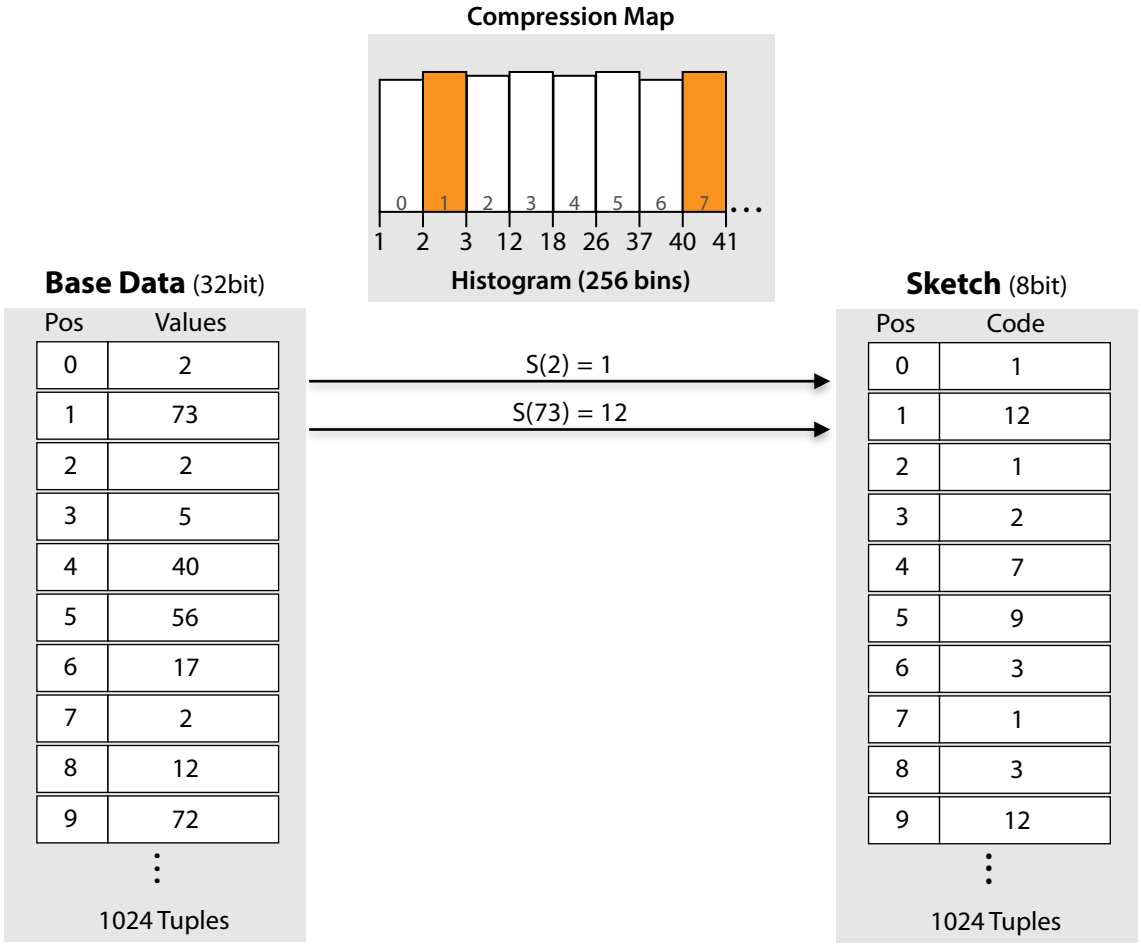
Column Sketches

Order Preserving - Build Sketch



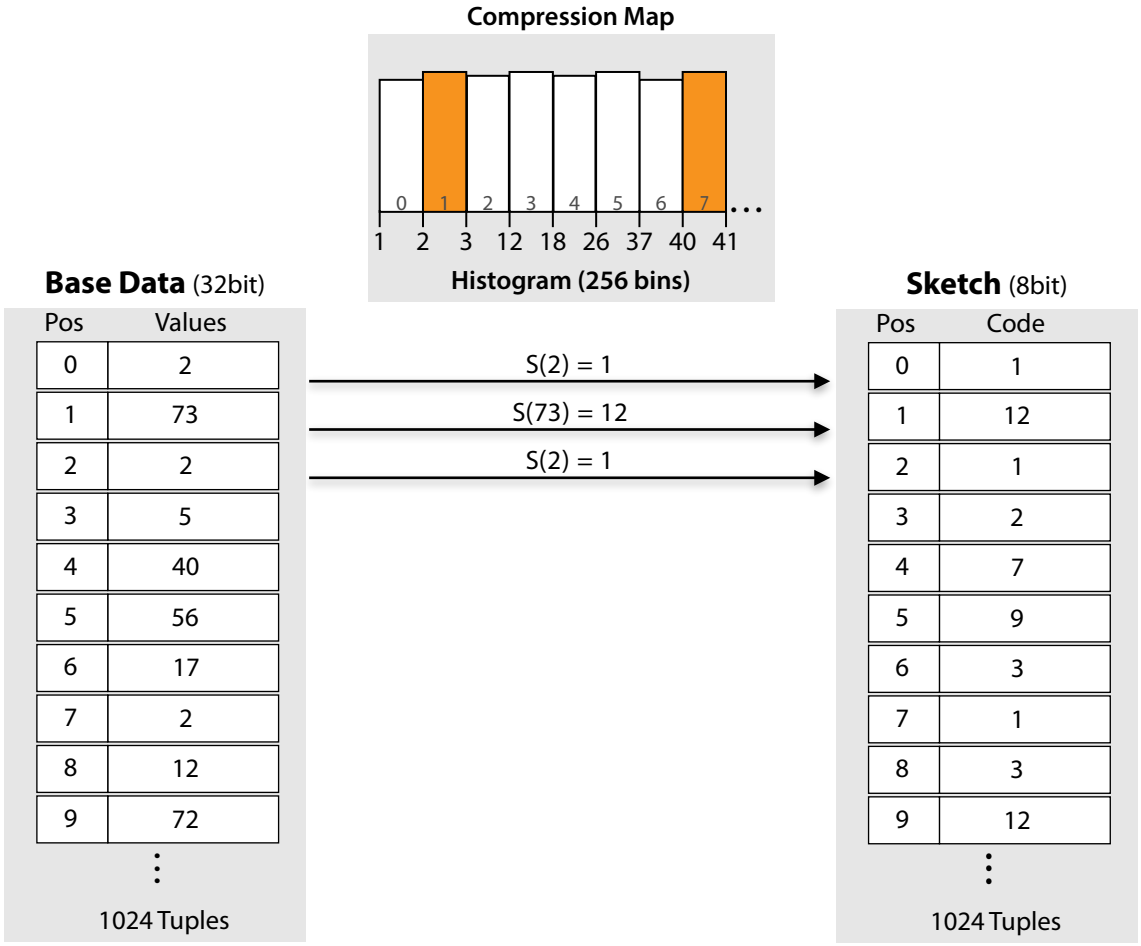
Column Sketches

Order Preserving - Build Sketch



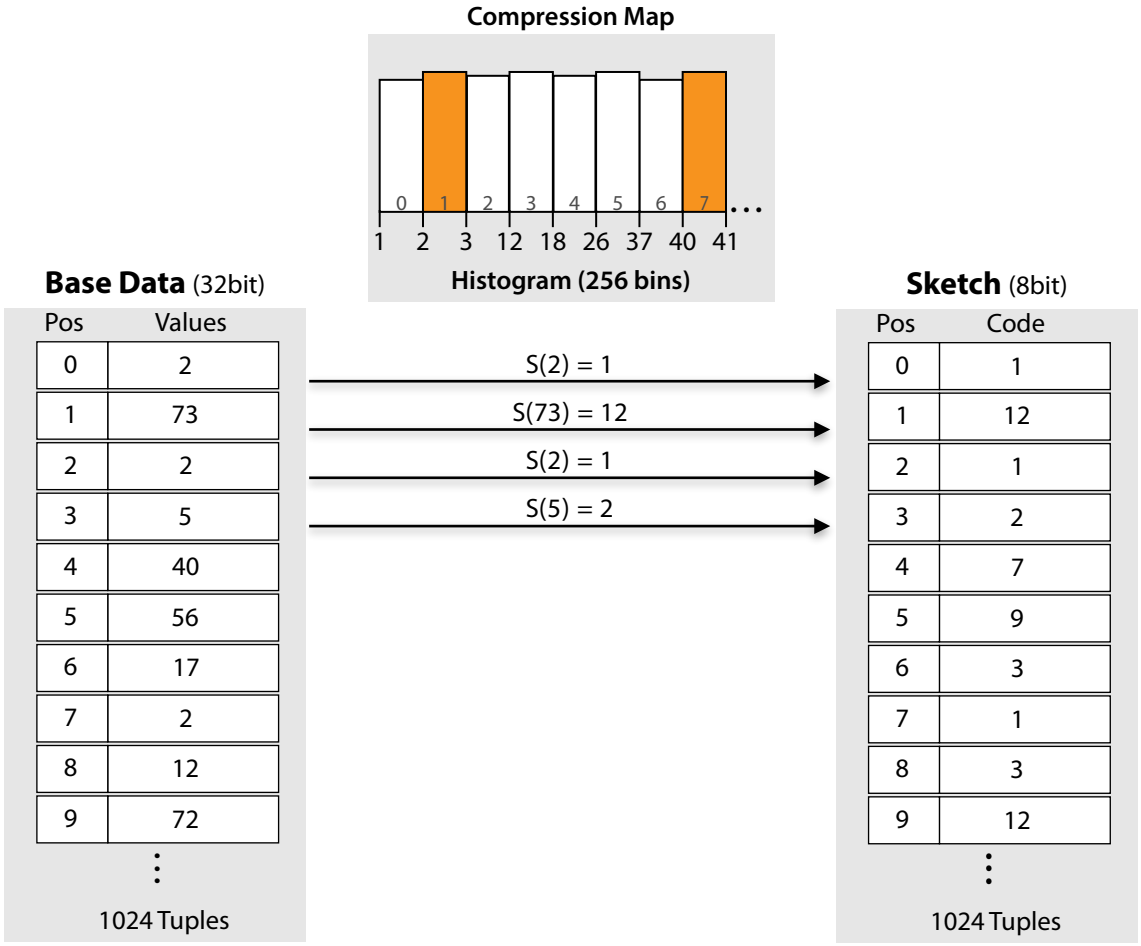
Column Sketches

Order Preserving - Build Sketch



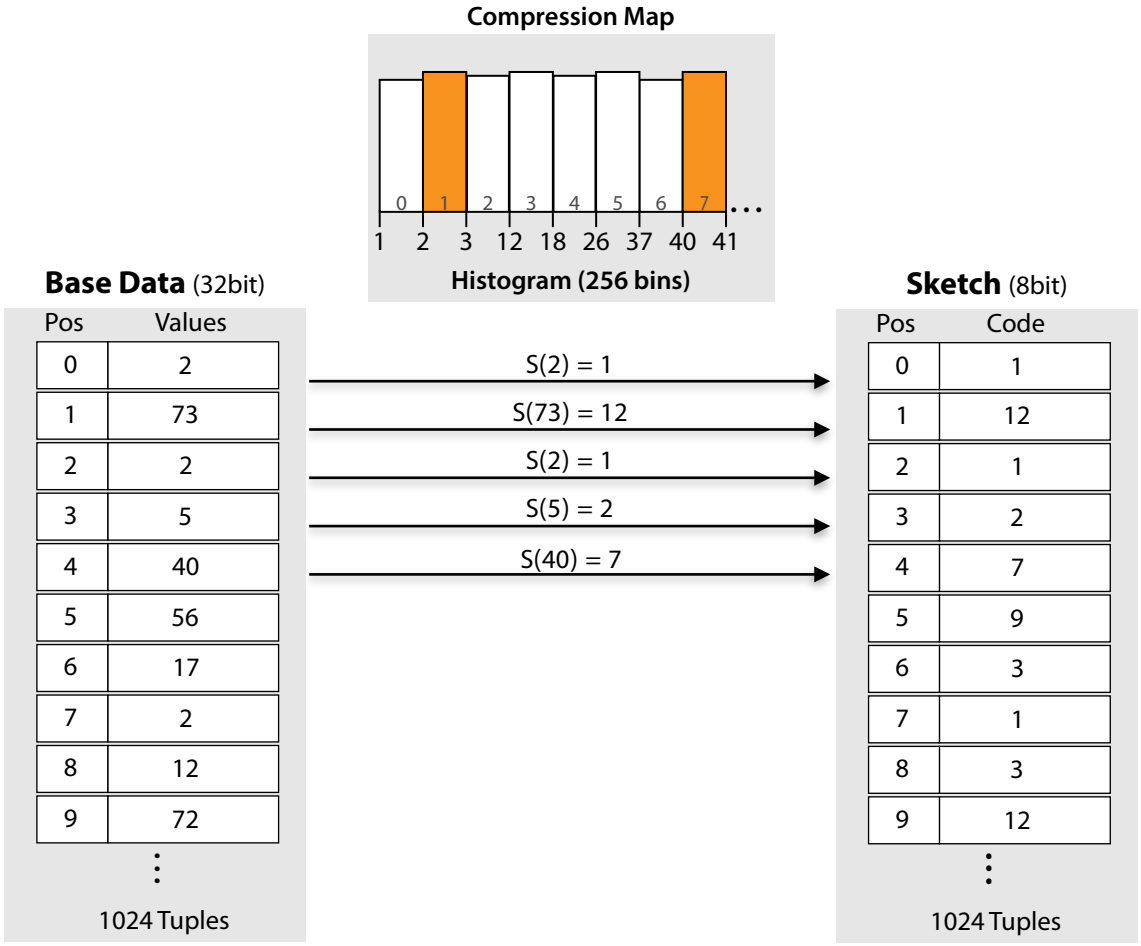
Column Sketches

Order Preserving - Build Sketch



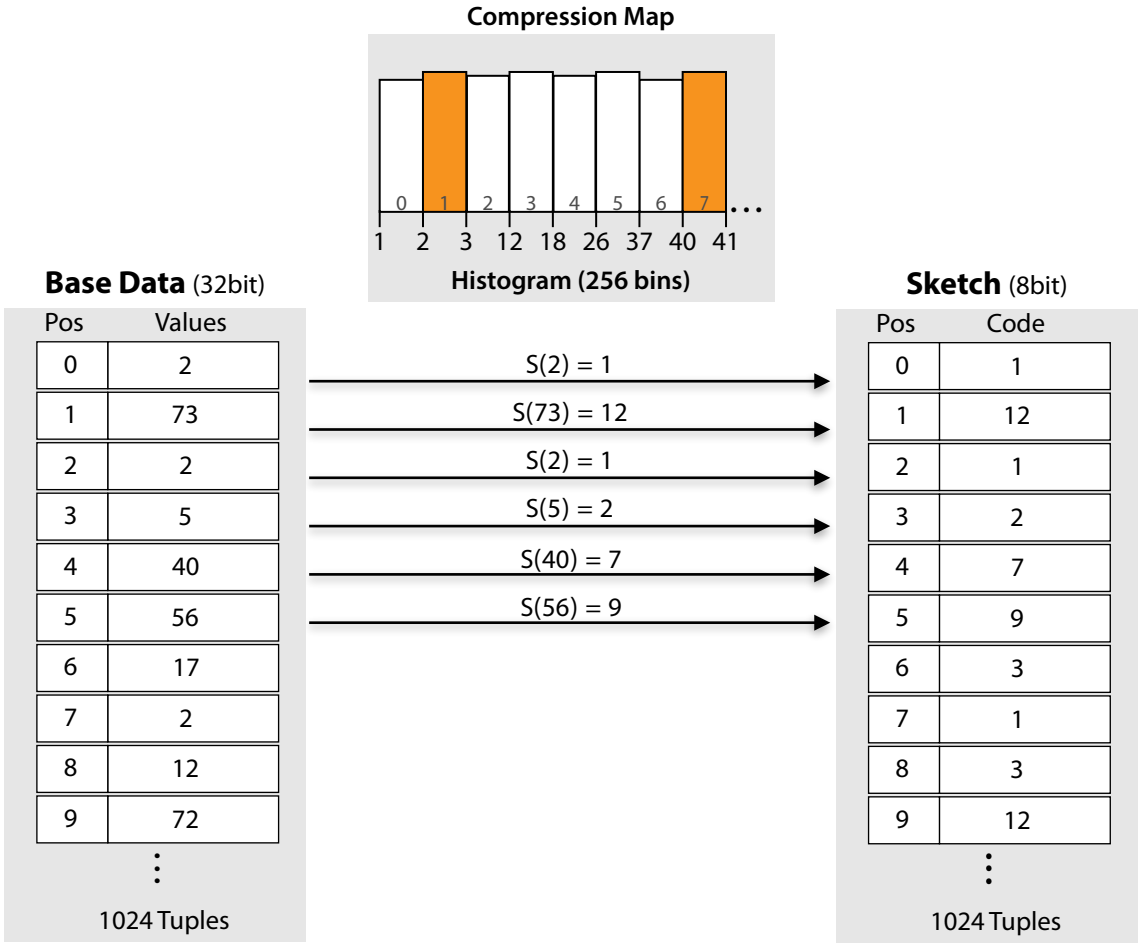
Column Sketches

Order Preserving - Build Sketch



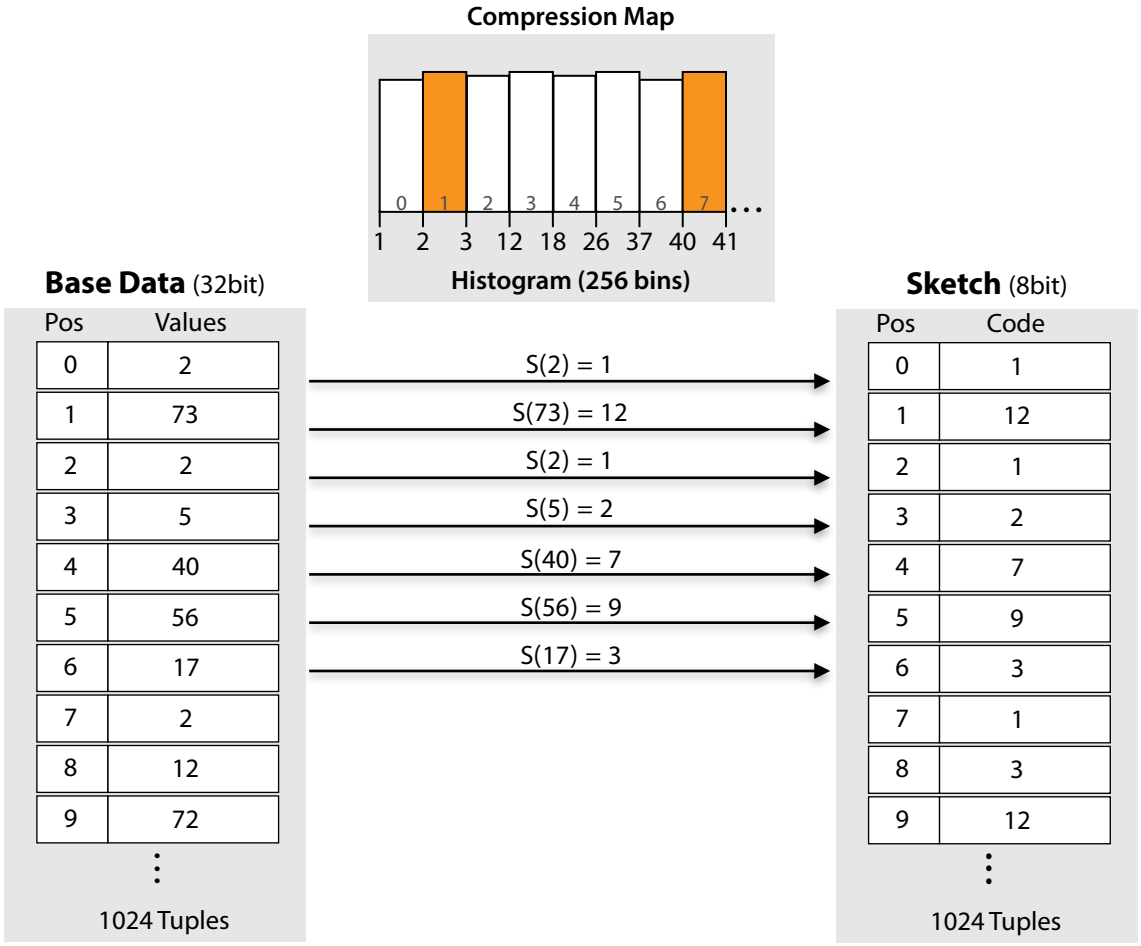
Column Sketches

Order Preserving - Build Sketch



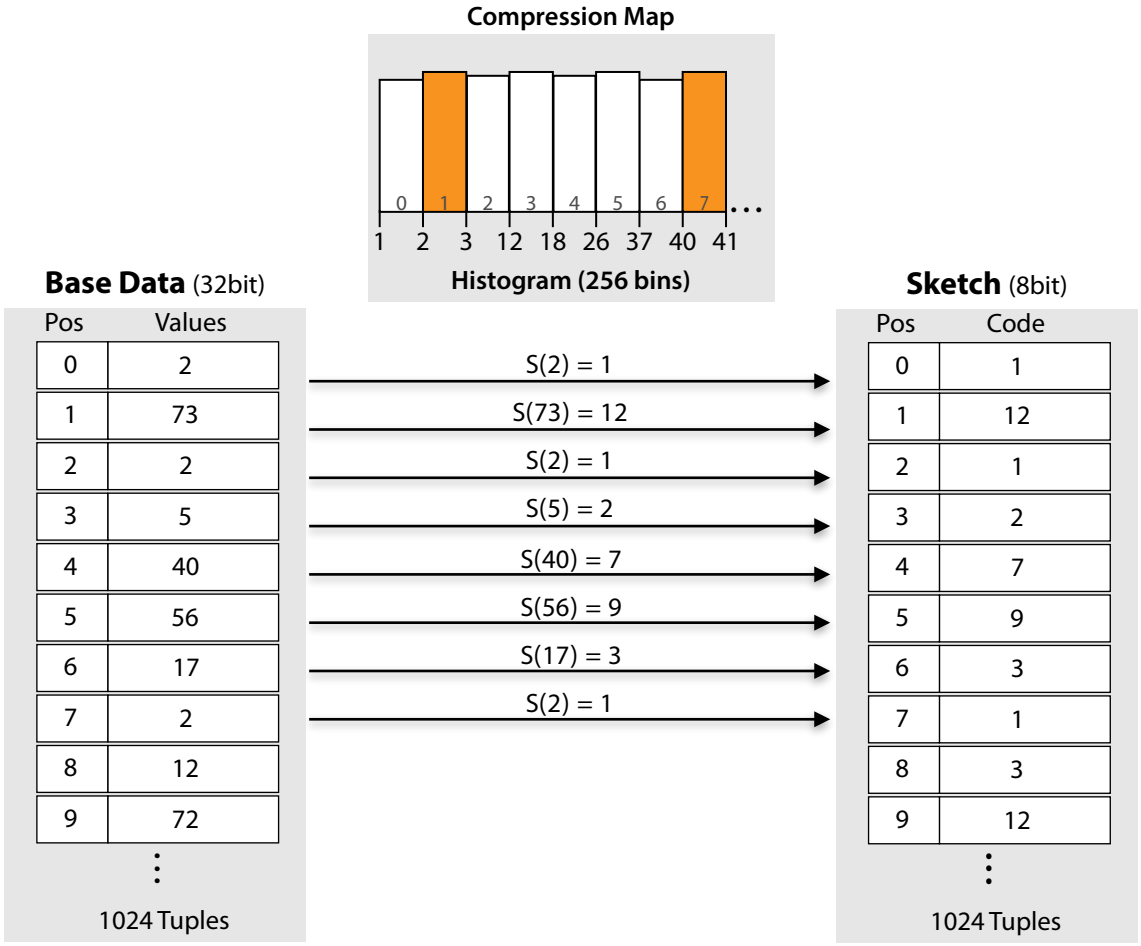
Column Sketches

Order Preserving - Build Sketch



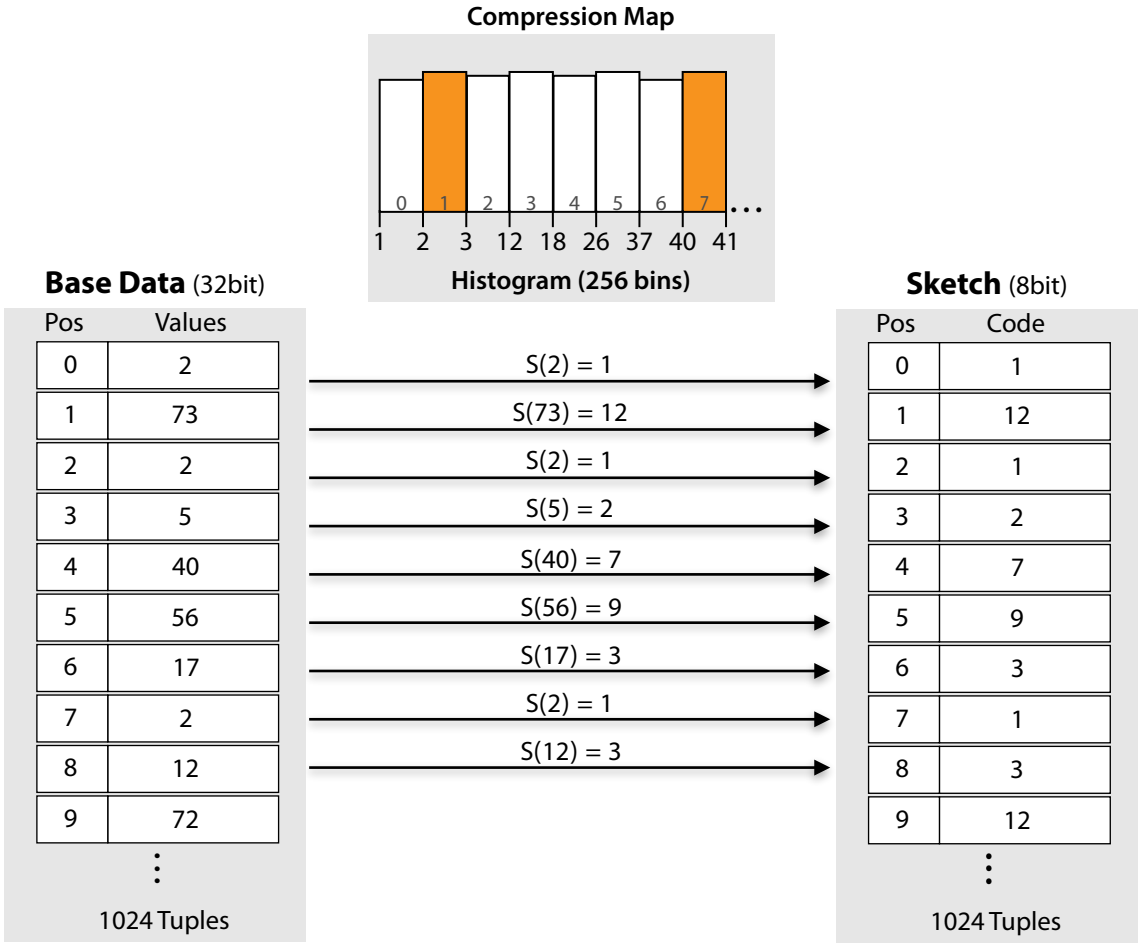
Column Sketches

Order Preserving - Build Sketch



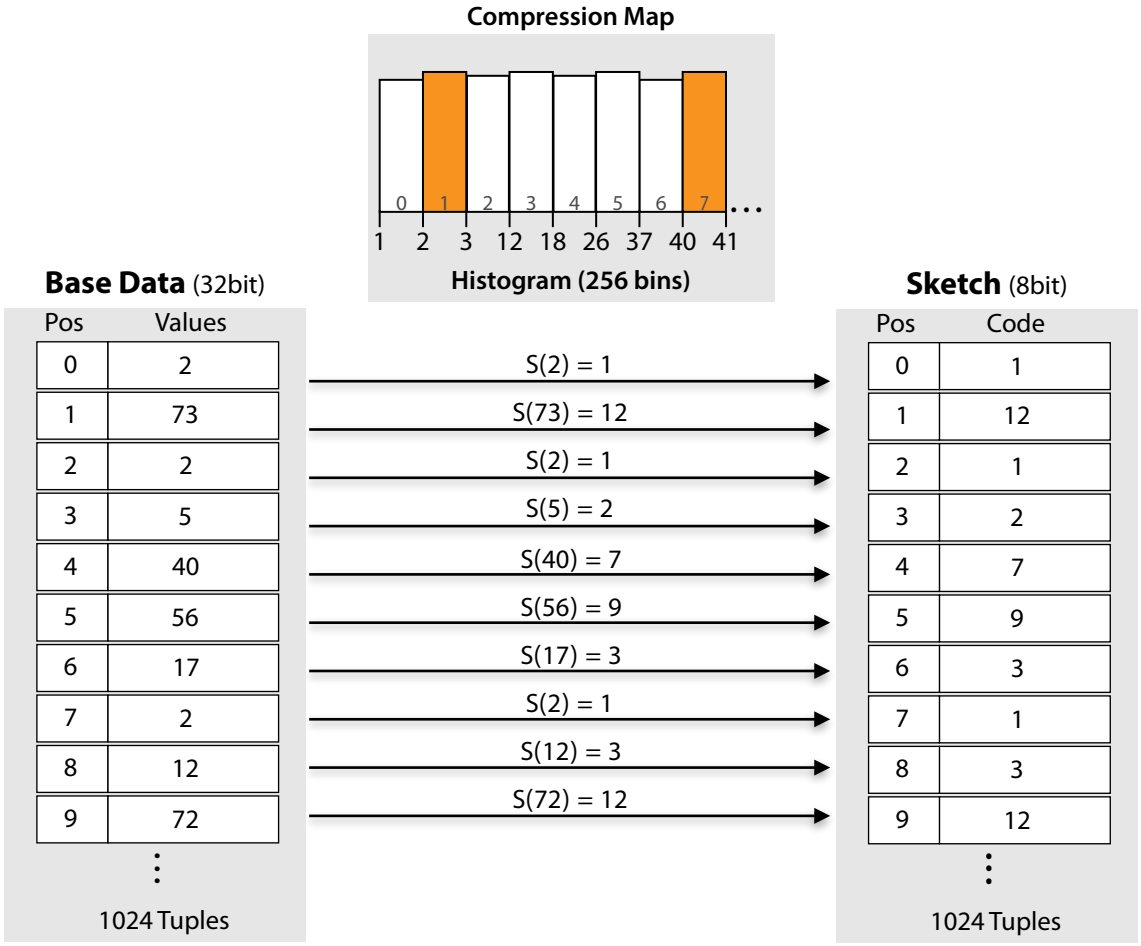
Column Sketches

Order Preserving - Build Sketch



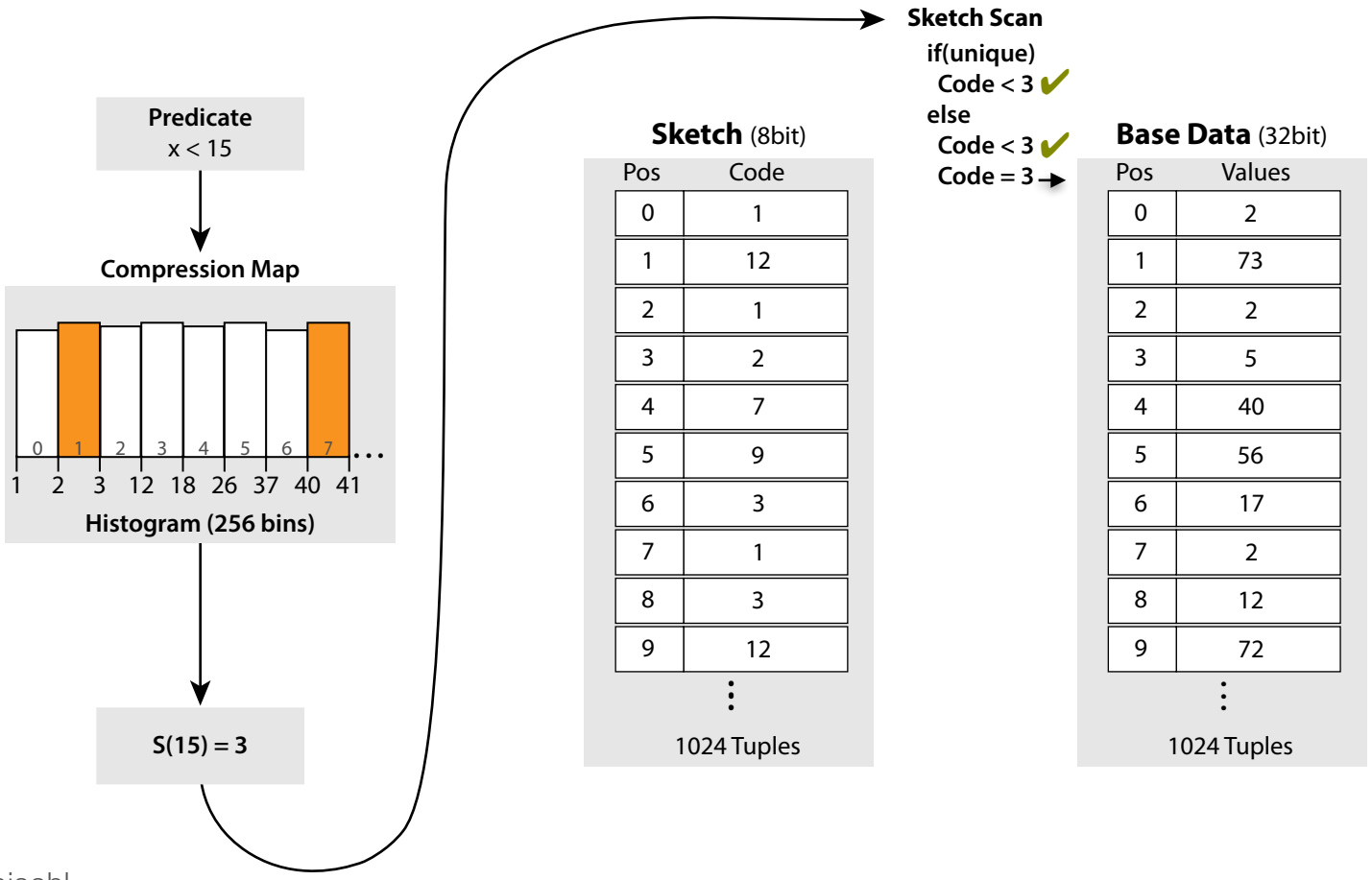
Column Sketches

Order Preserving - Build Sketch



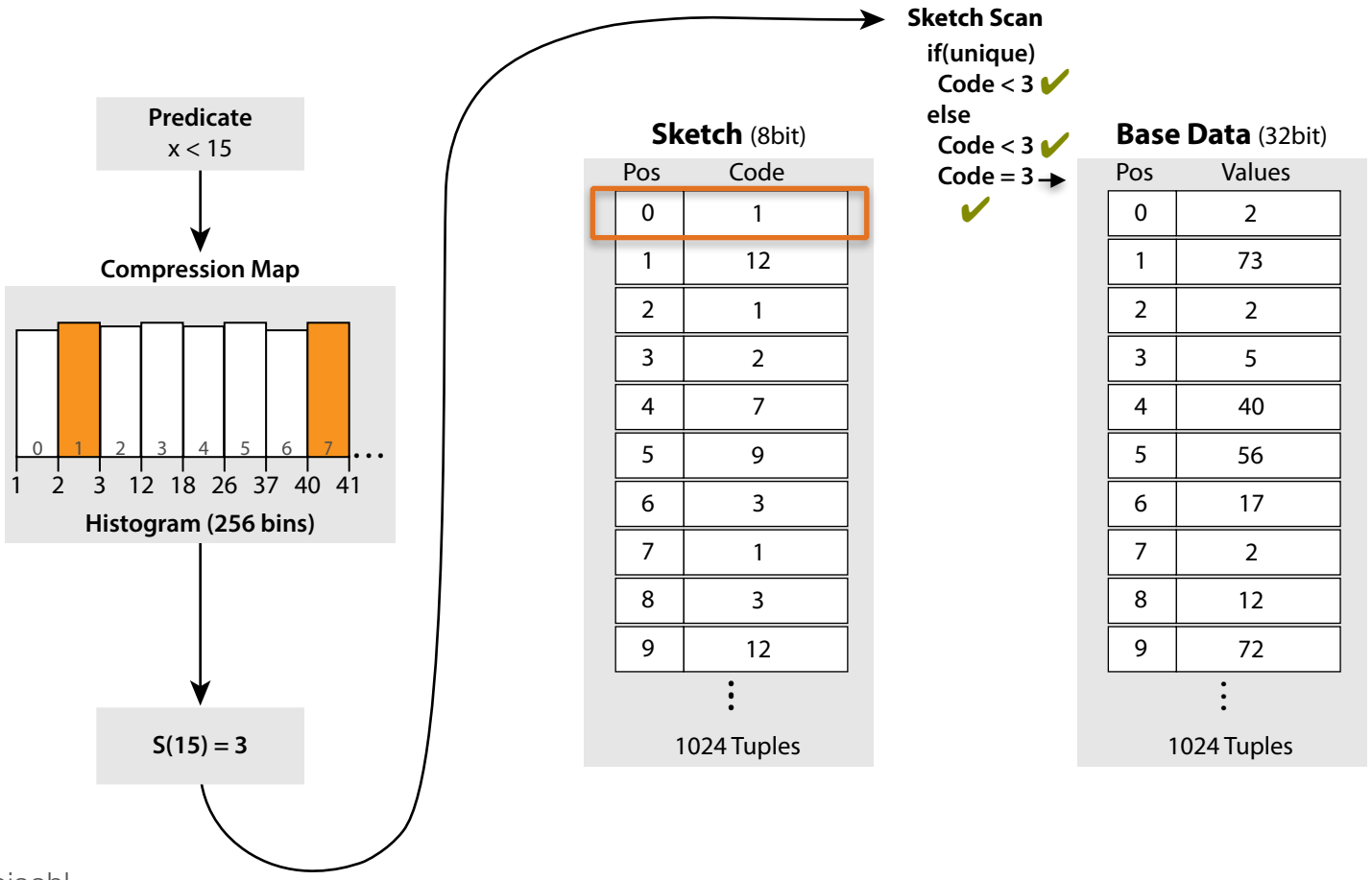
Column Sketches

Order Preserving - Lookup



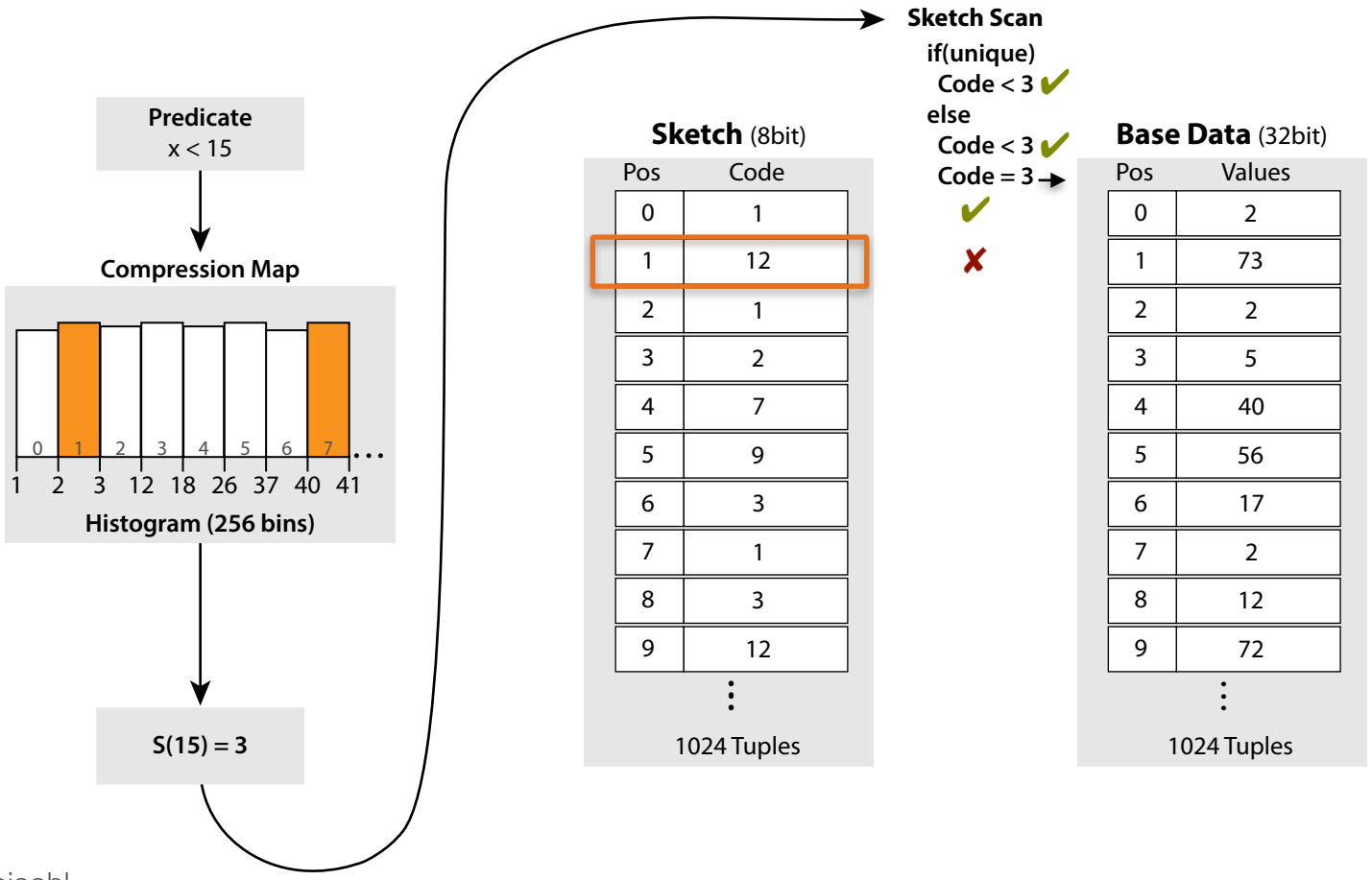
Column Sketches

Order Preserving - Lookup



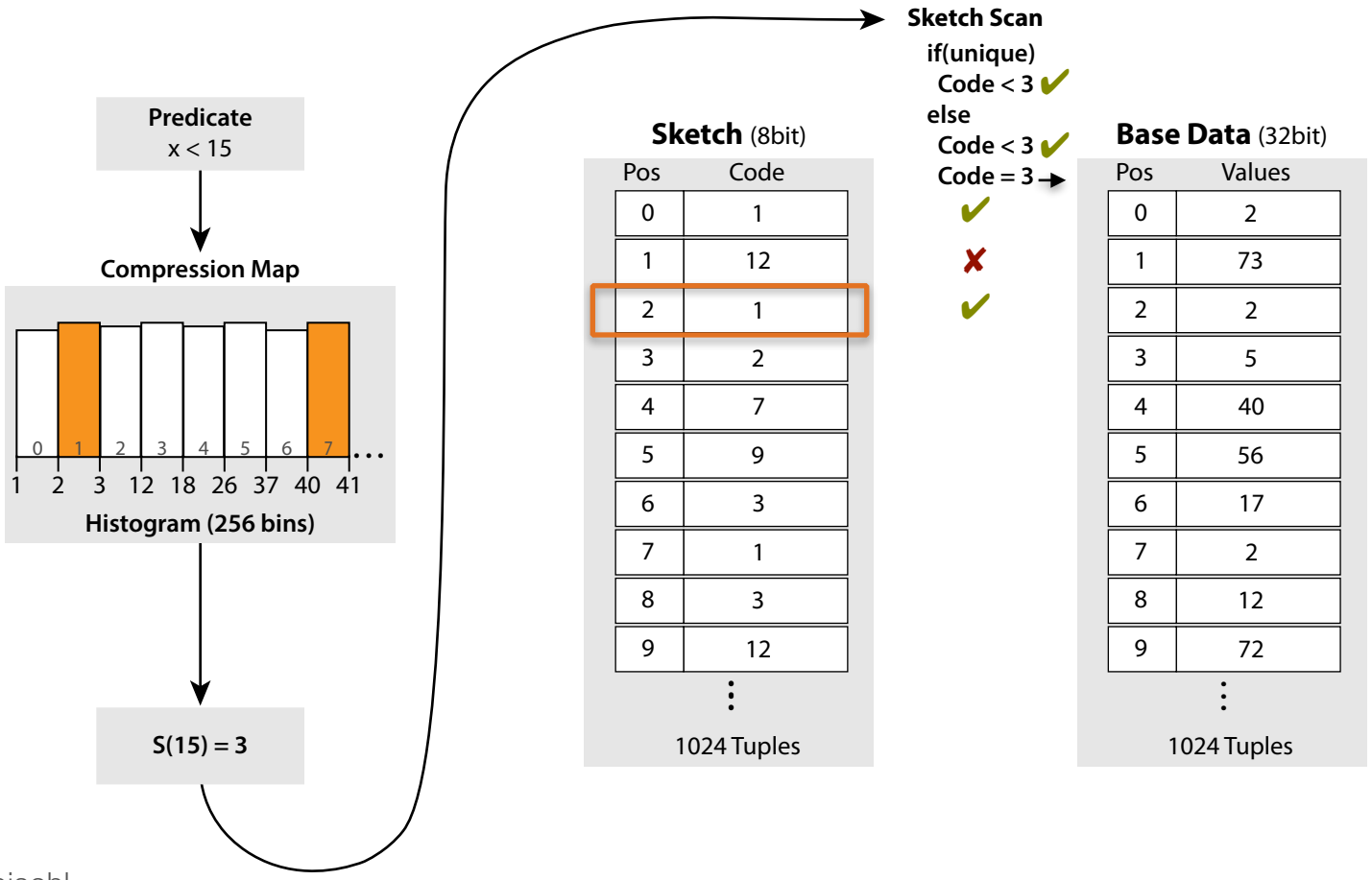
Column Sketches

Order Preserving - Lookup



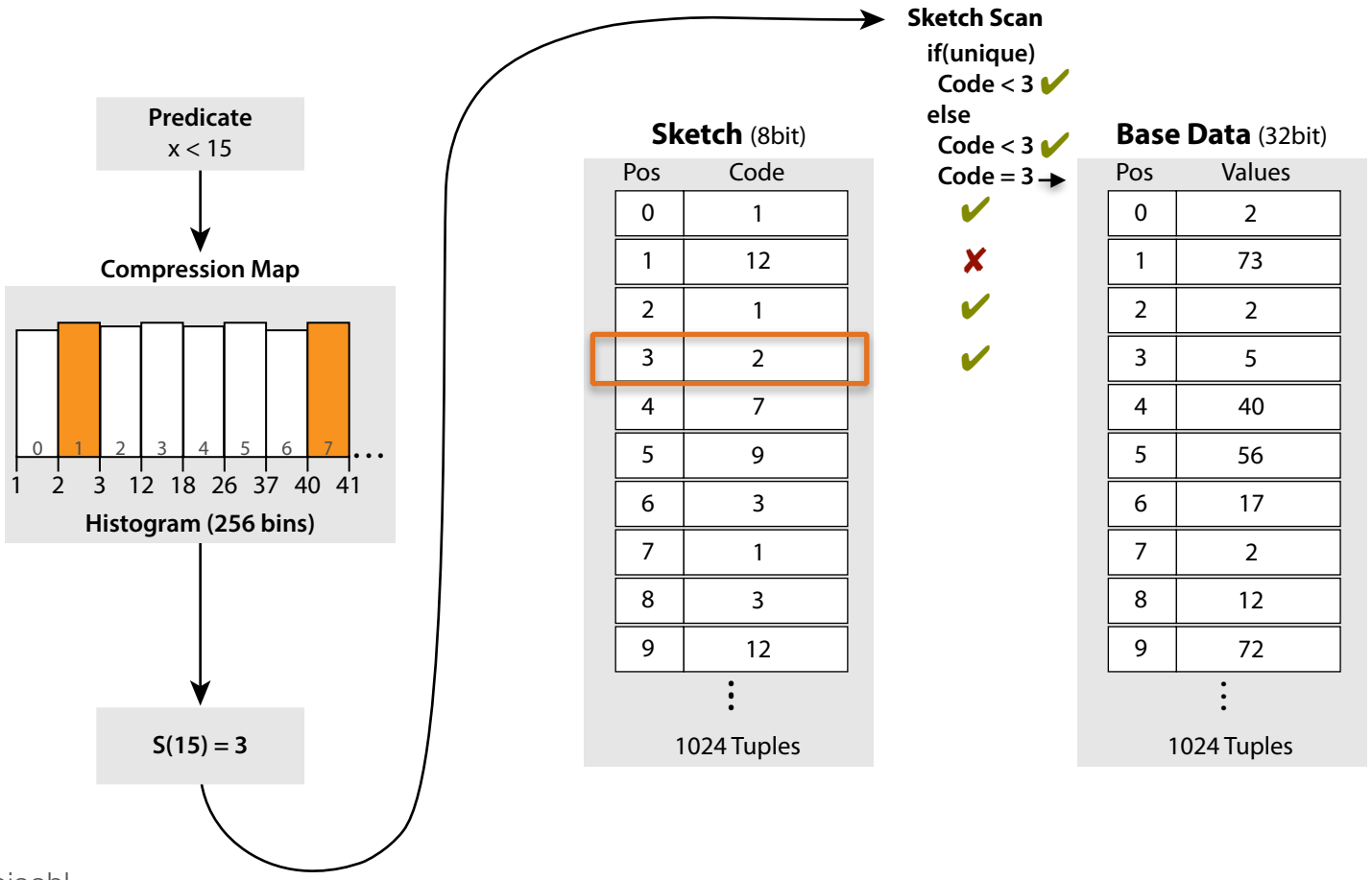
Column Sketches

Order Preserving - Lookup



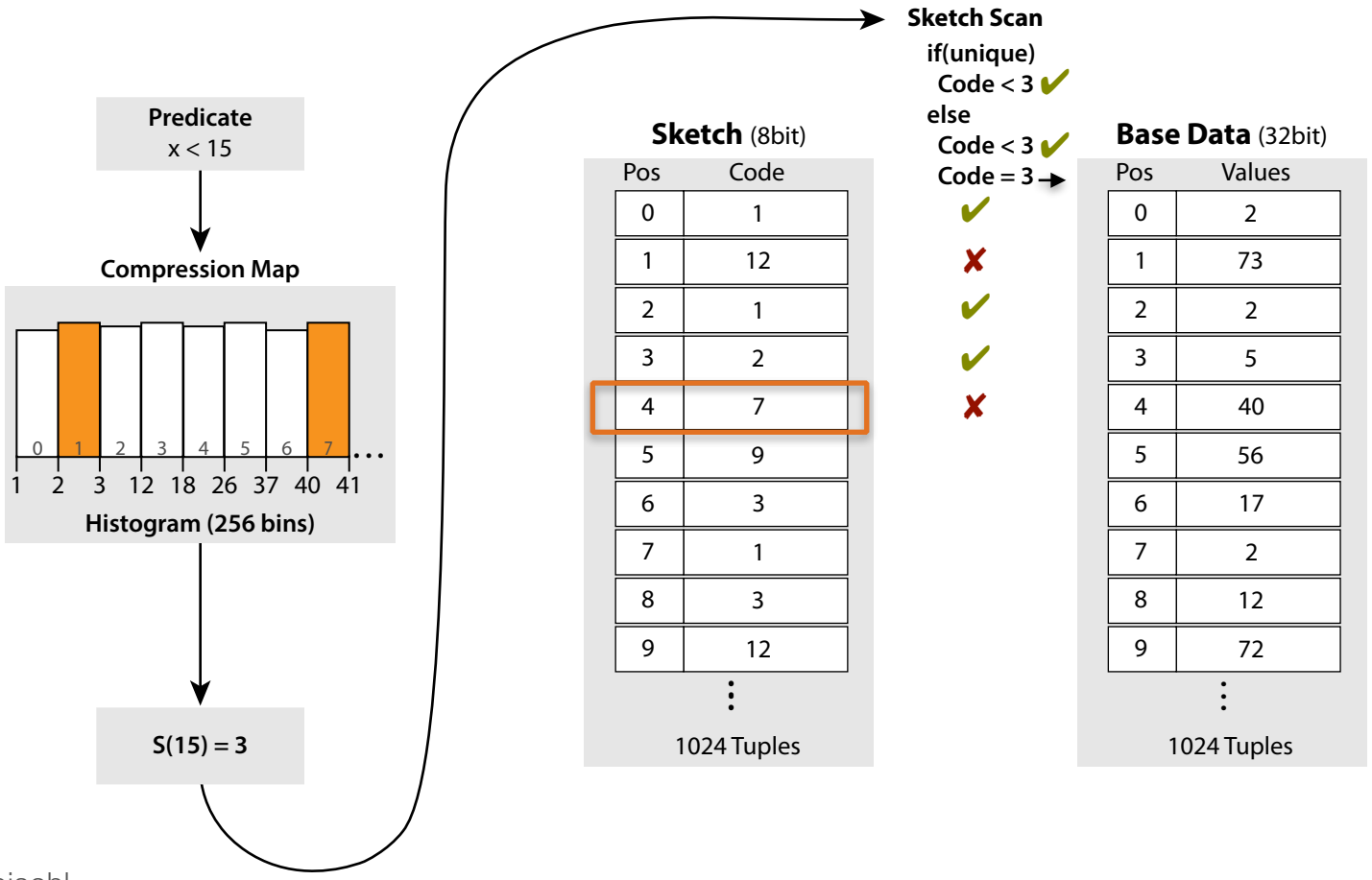
Column Sketches

Order Preserving - Lookup



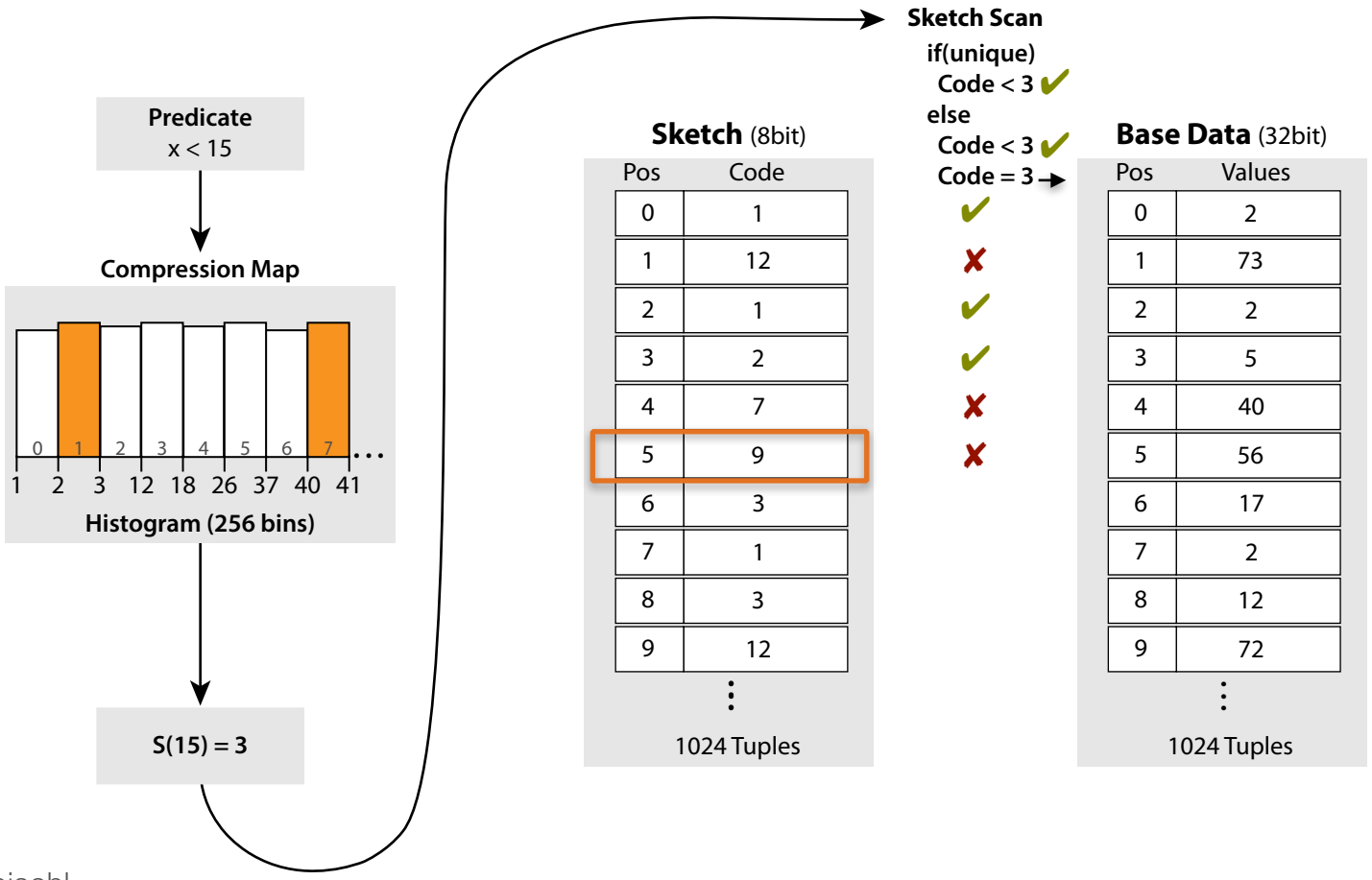
Column Sketches

Order Preserving - Lookup



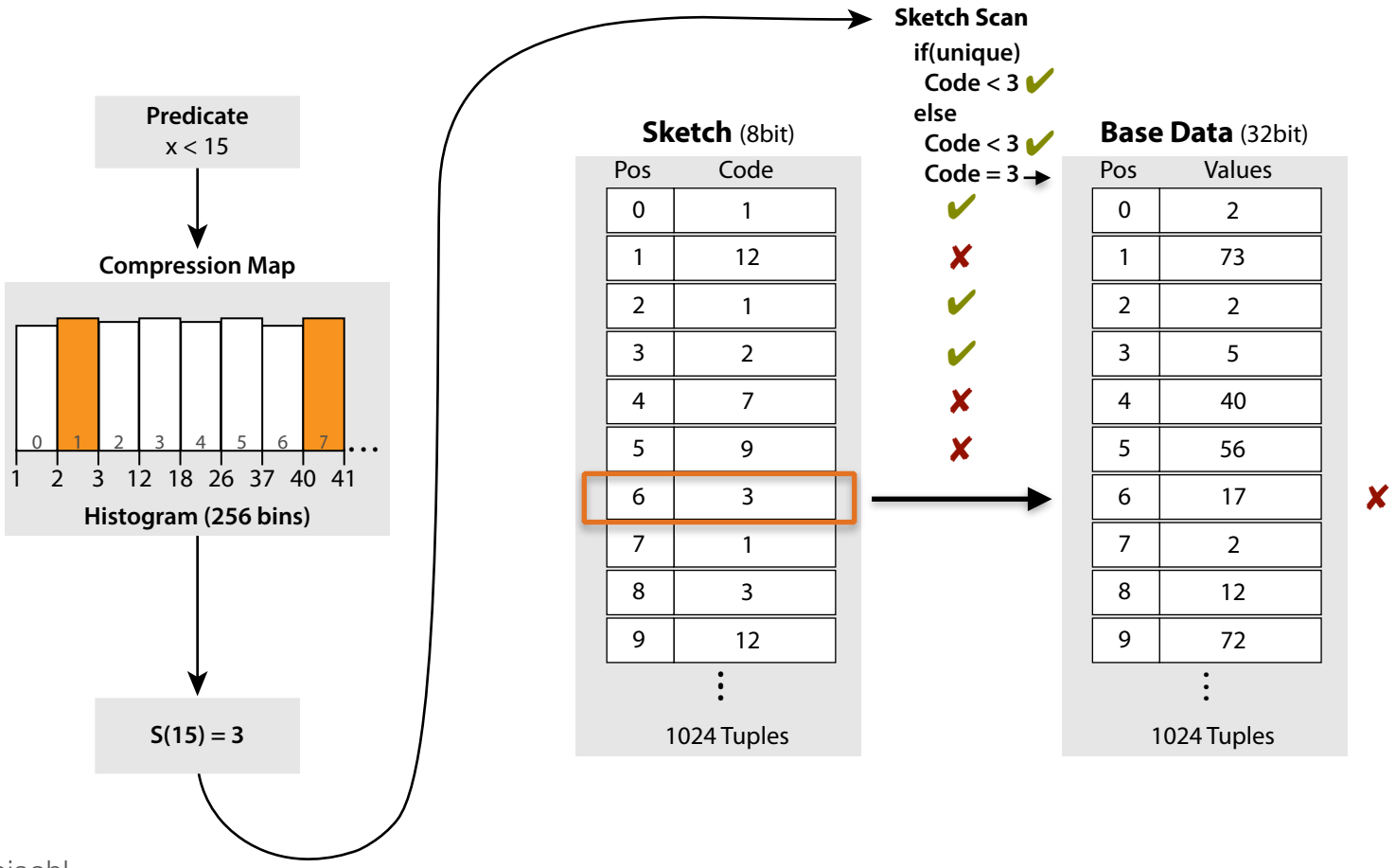
Column Sketches

Order Preserving - Lookup



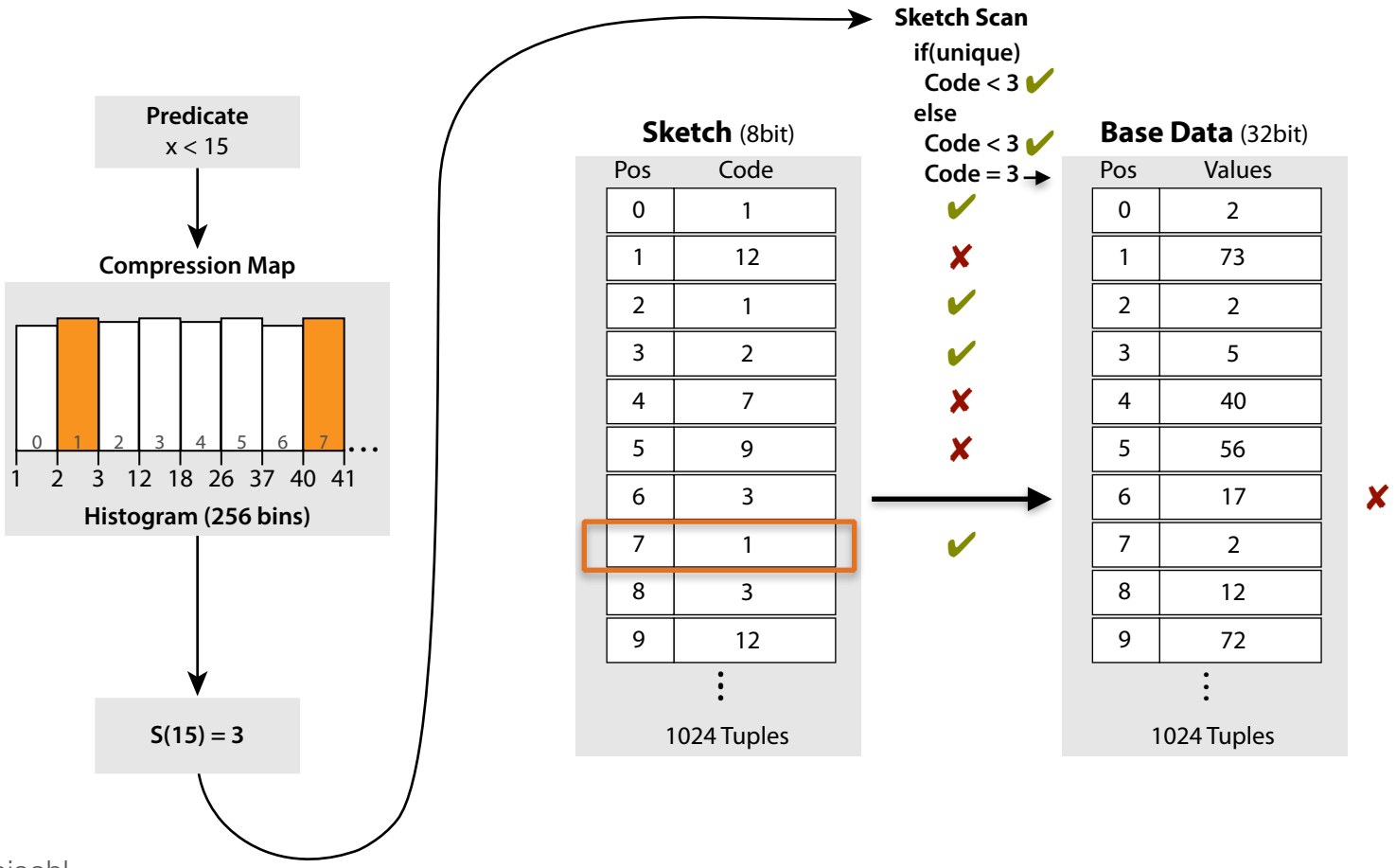
Column Sketches

Order Preserving - Lookup



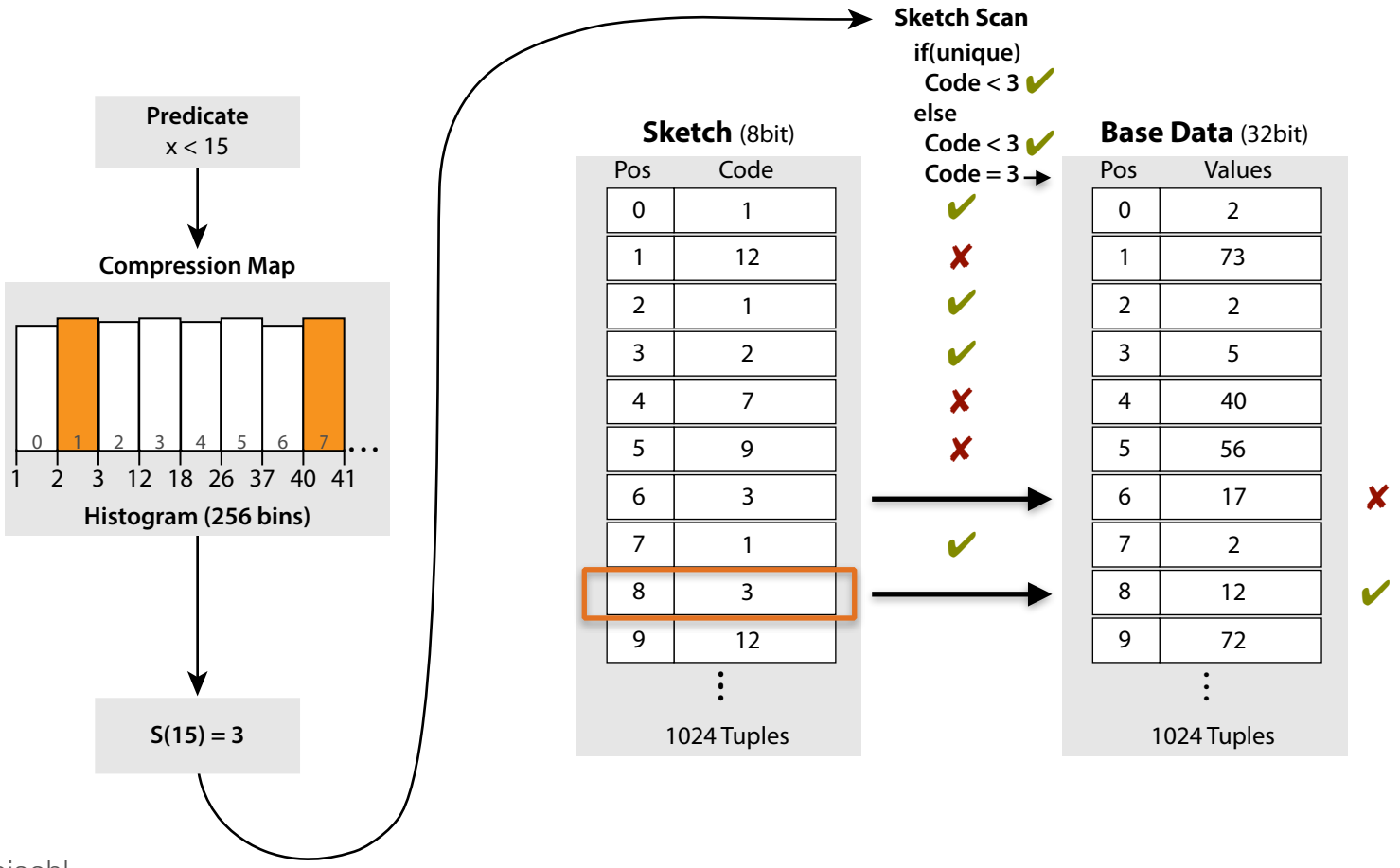
Column Sketches

Order Preserving - Lookup



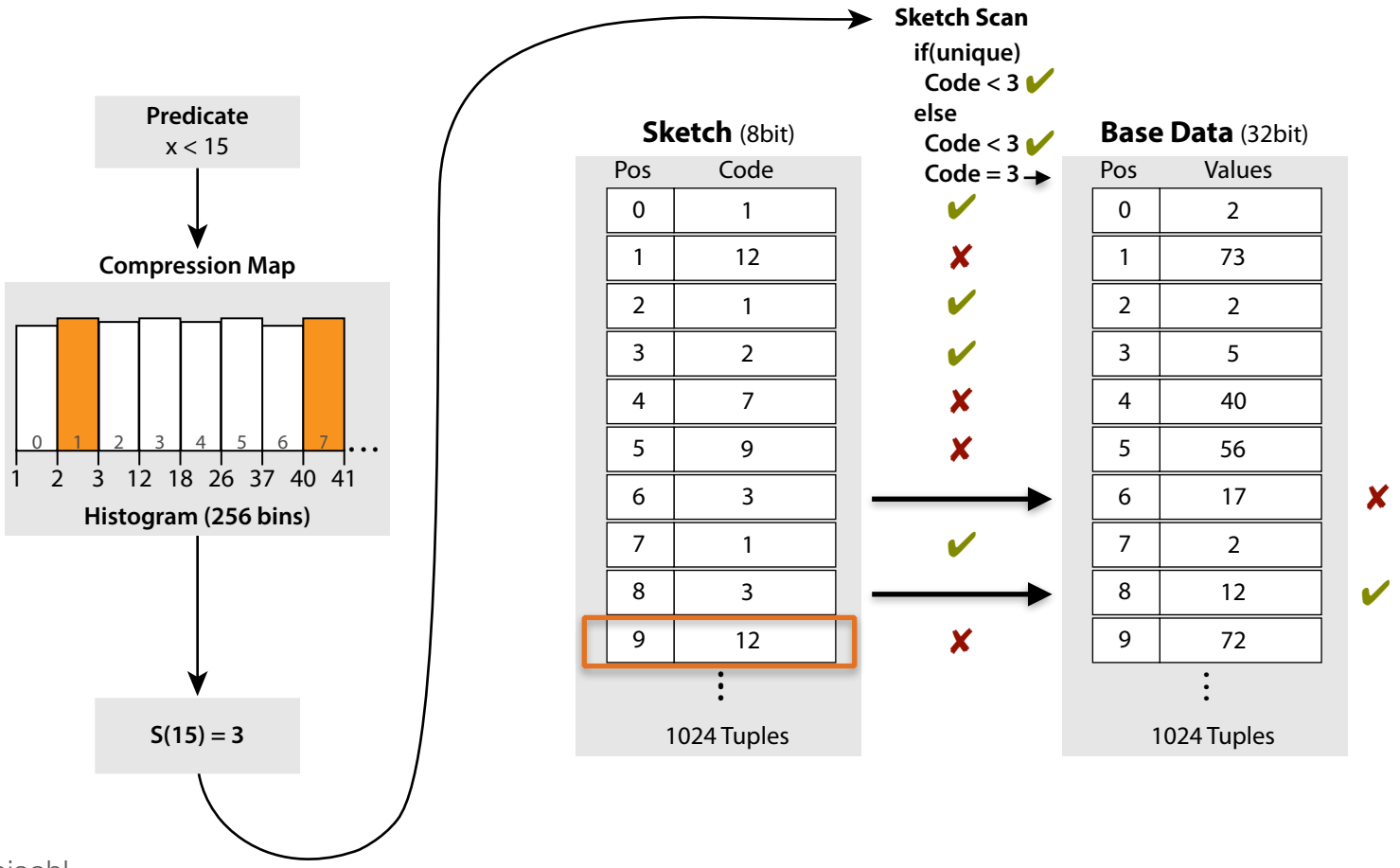
Column Sketches

Order Preserving - Lookup



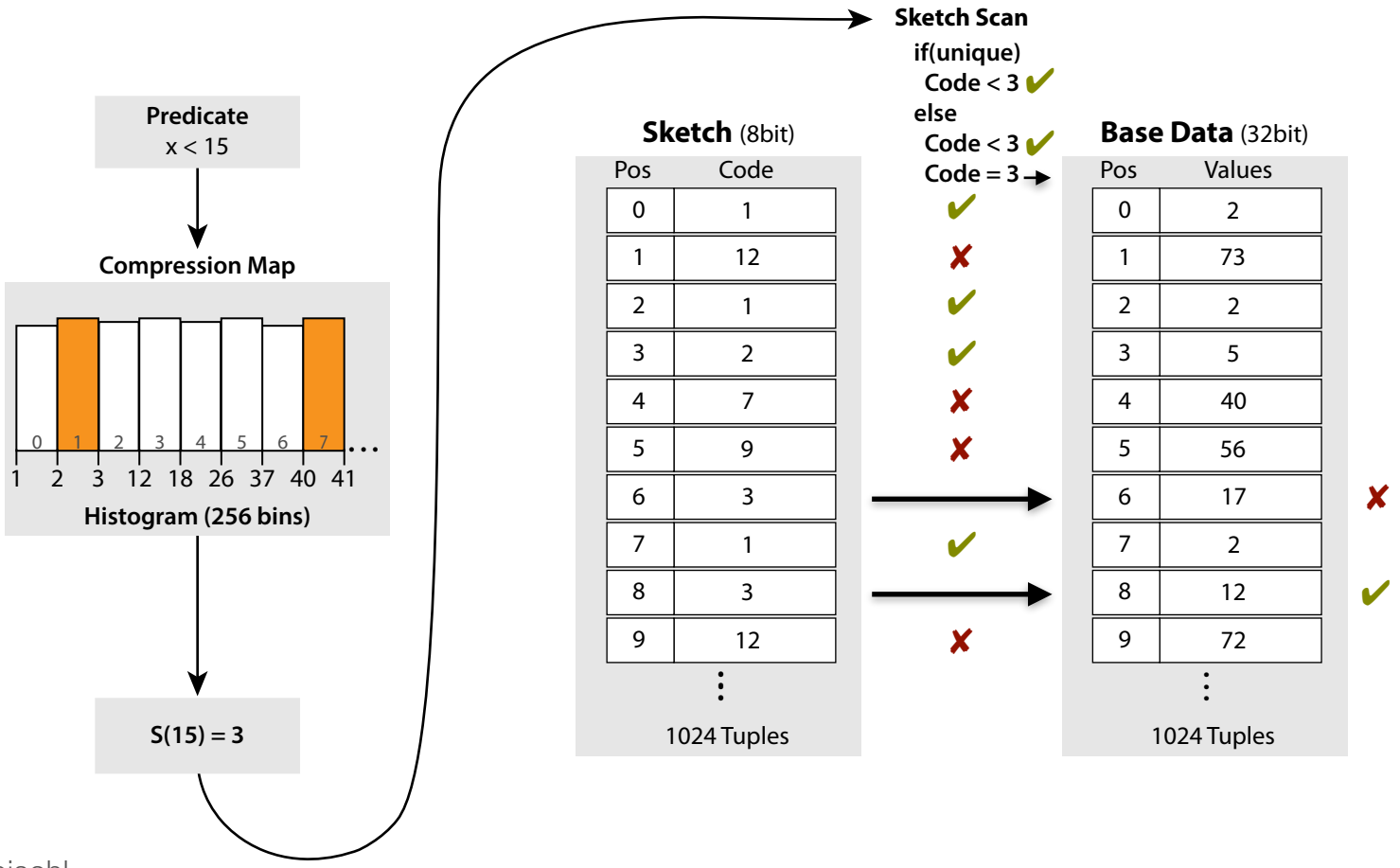
Column Sketches

Order Preserving - Lookup



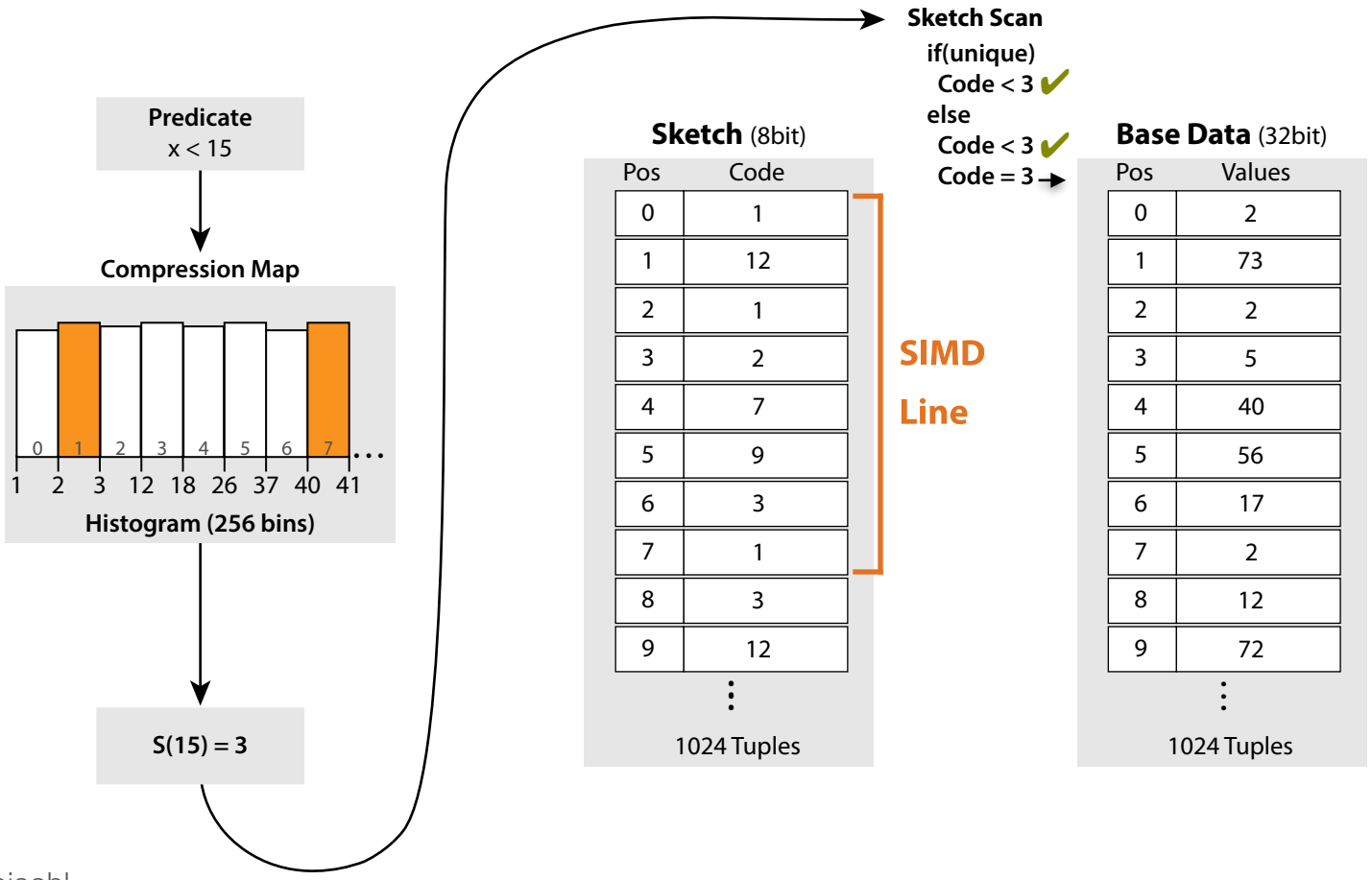
Column Sketches

Order Preserving - Lookup



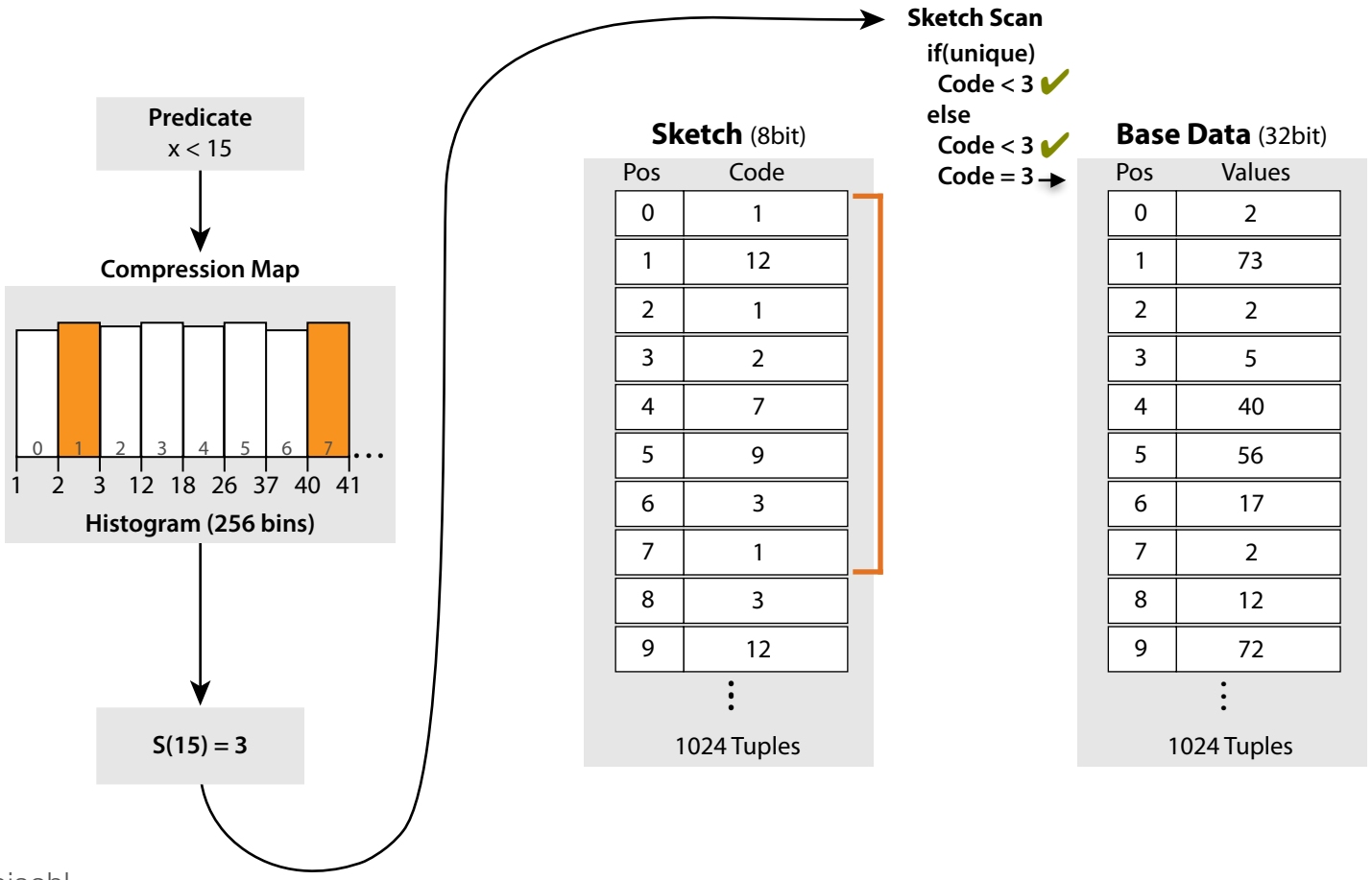
Column Sketches

Order Preserving - Lookup SIMD



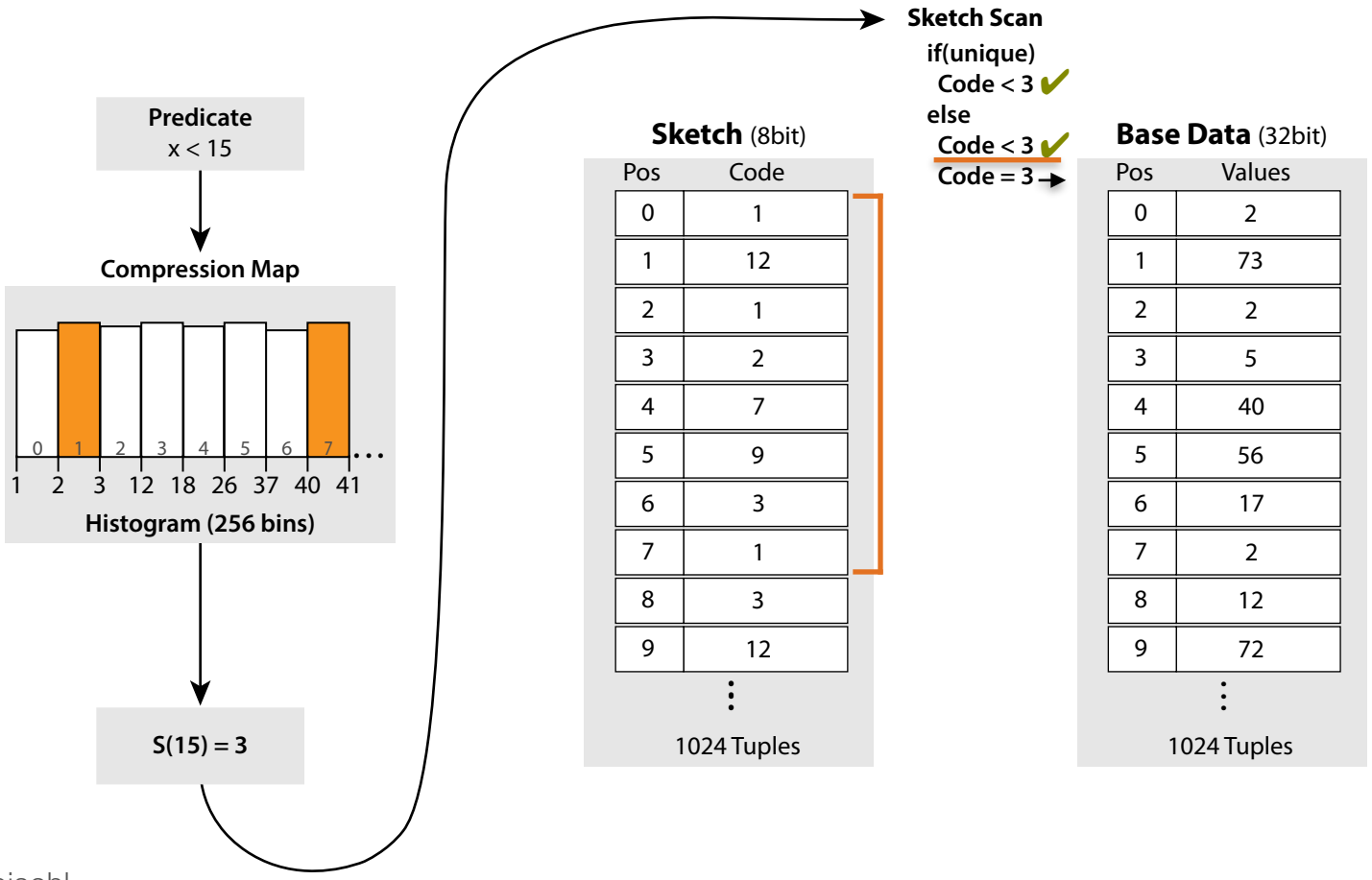
Column Sketches

Order Preserving - Lookup SIMD



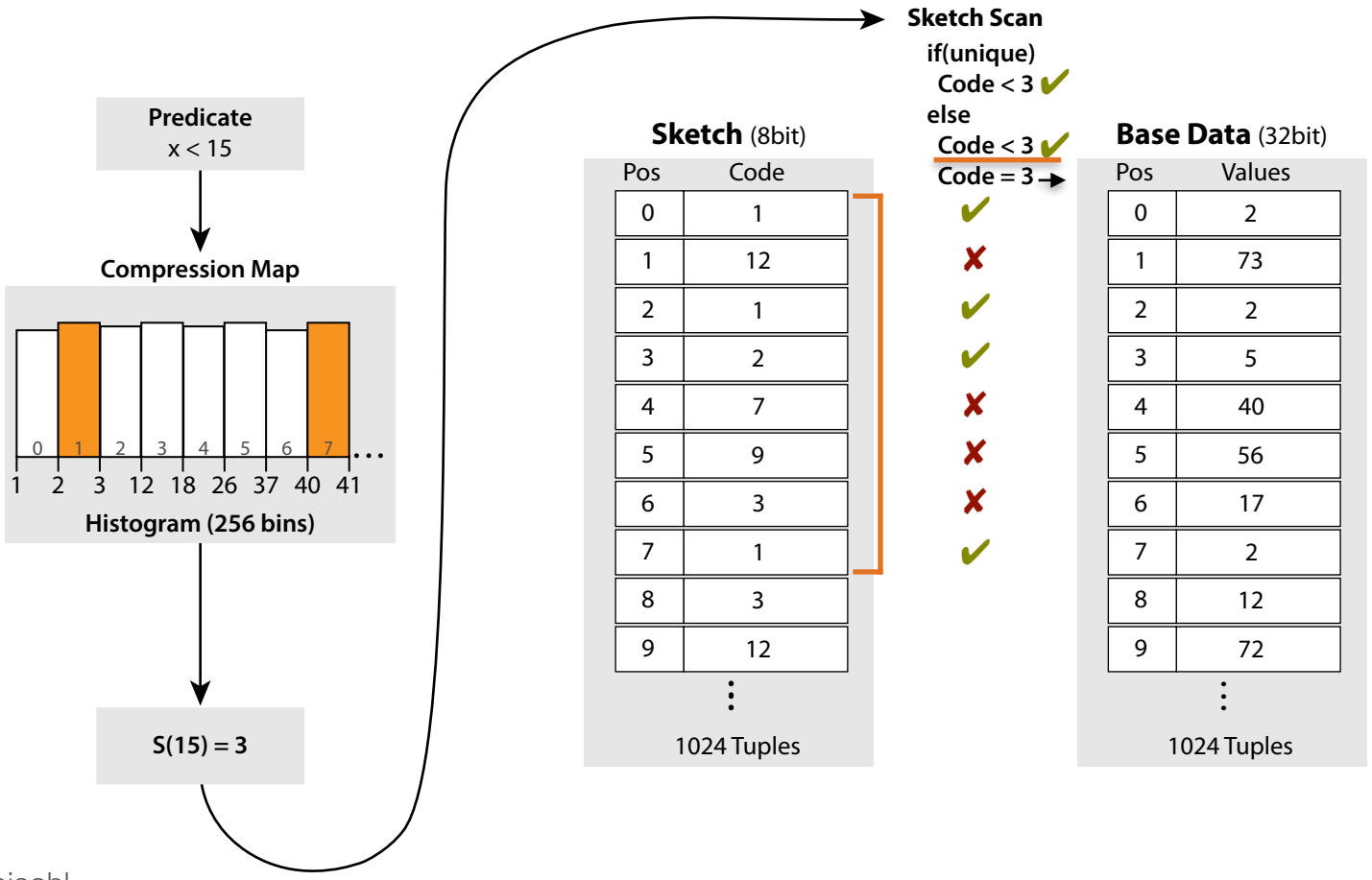
Column Sketches

Order Preserving - Lookup SIMD



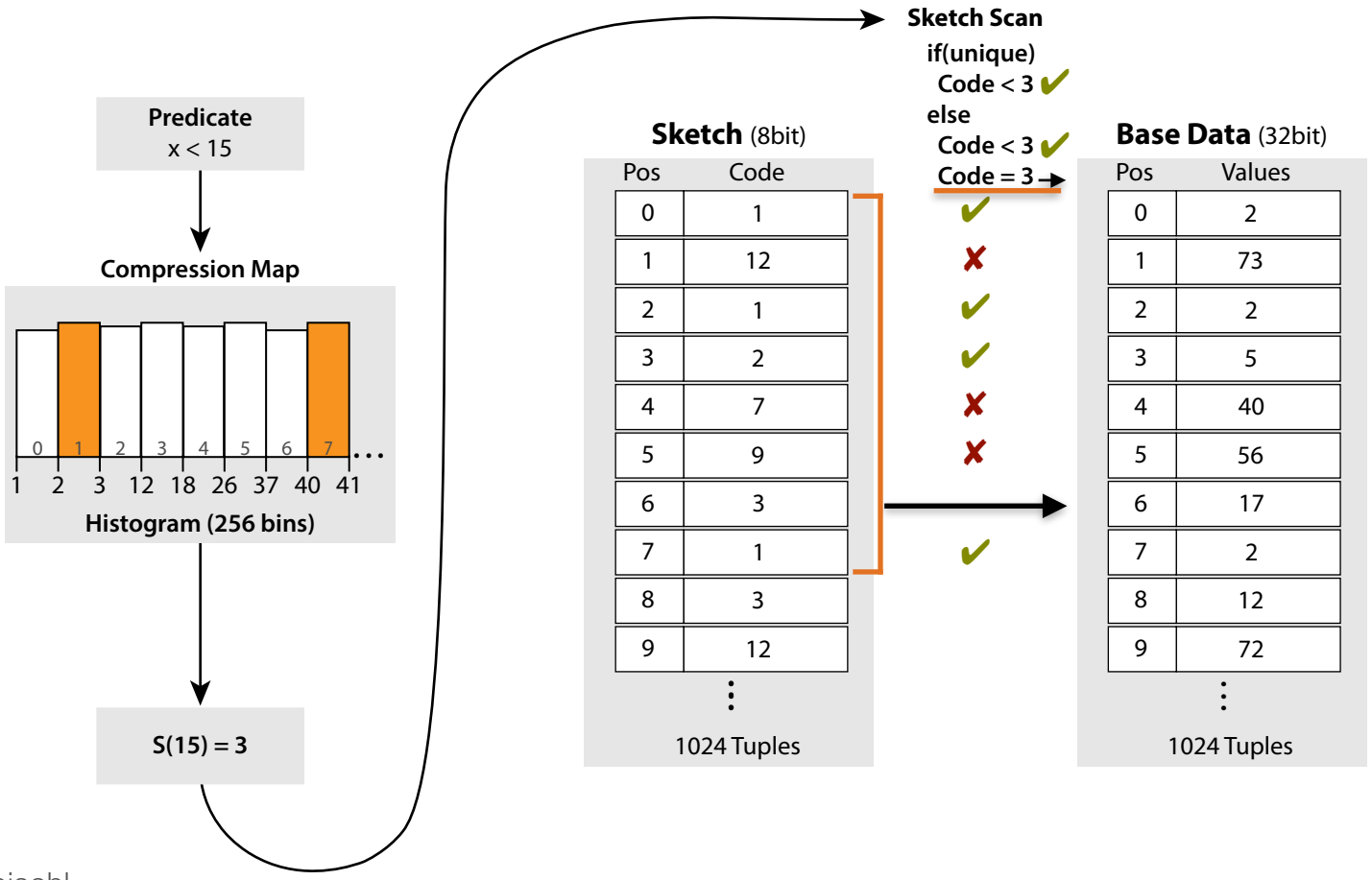
Column Sketches

Order Preserving - Lookup SIMD



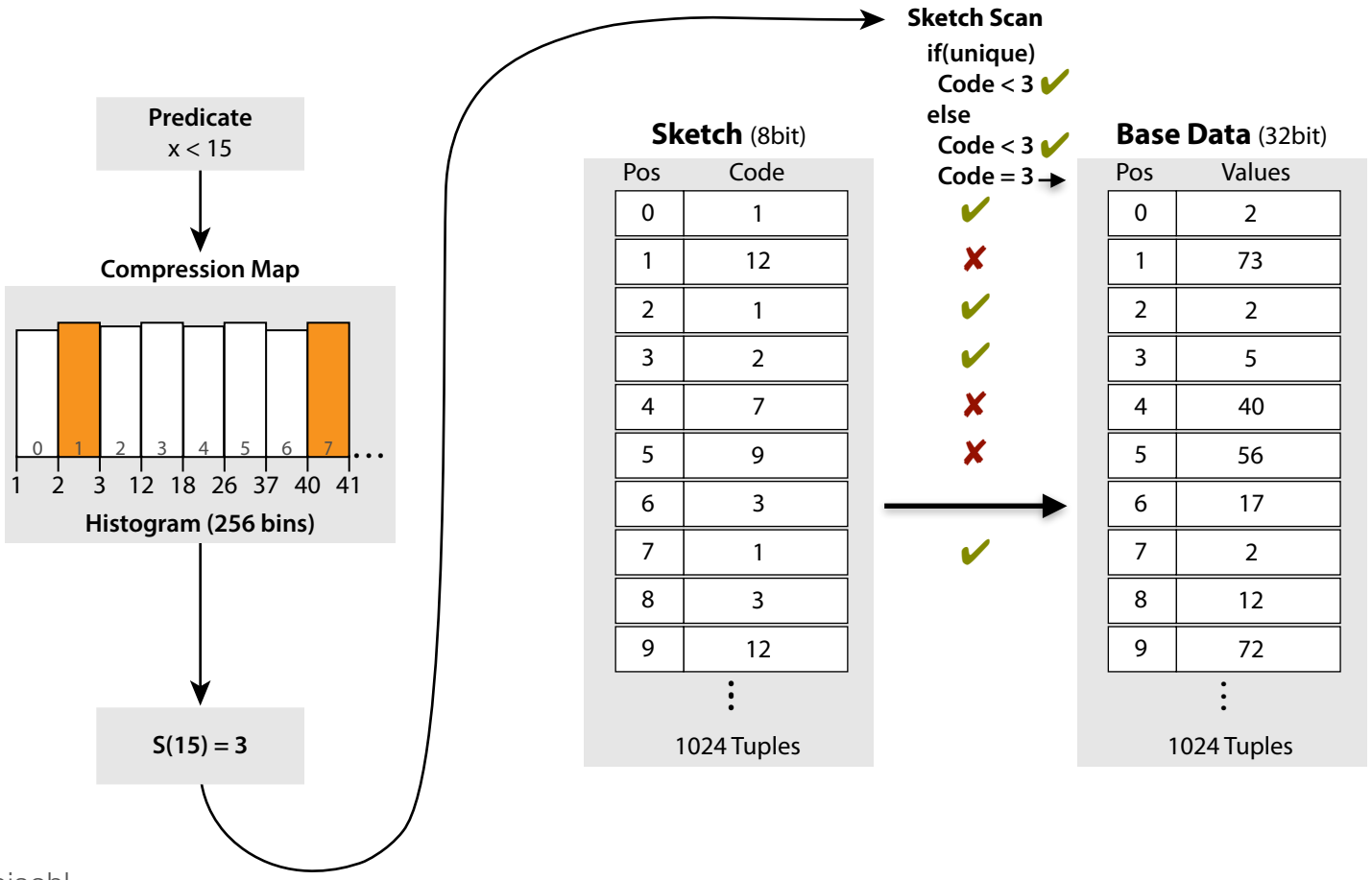
Column Sketches

Order Preserving - Lookup SIMD



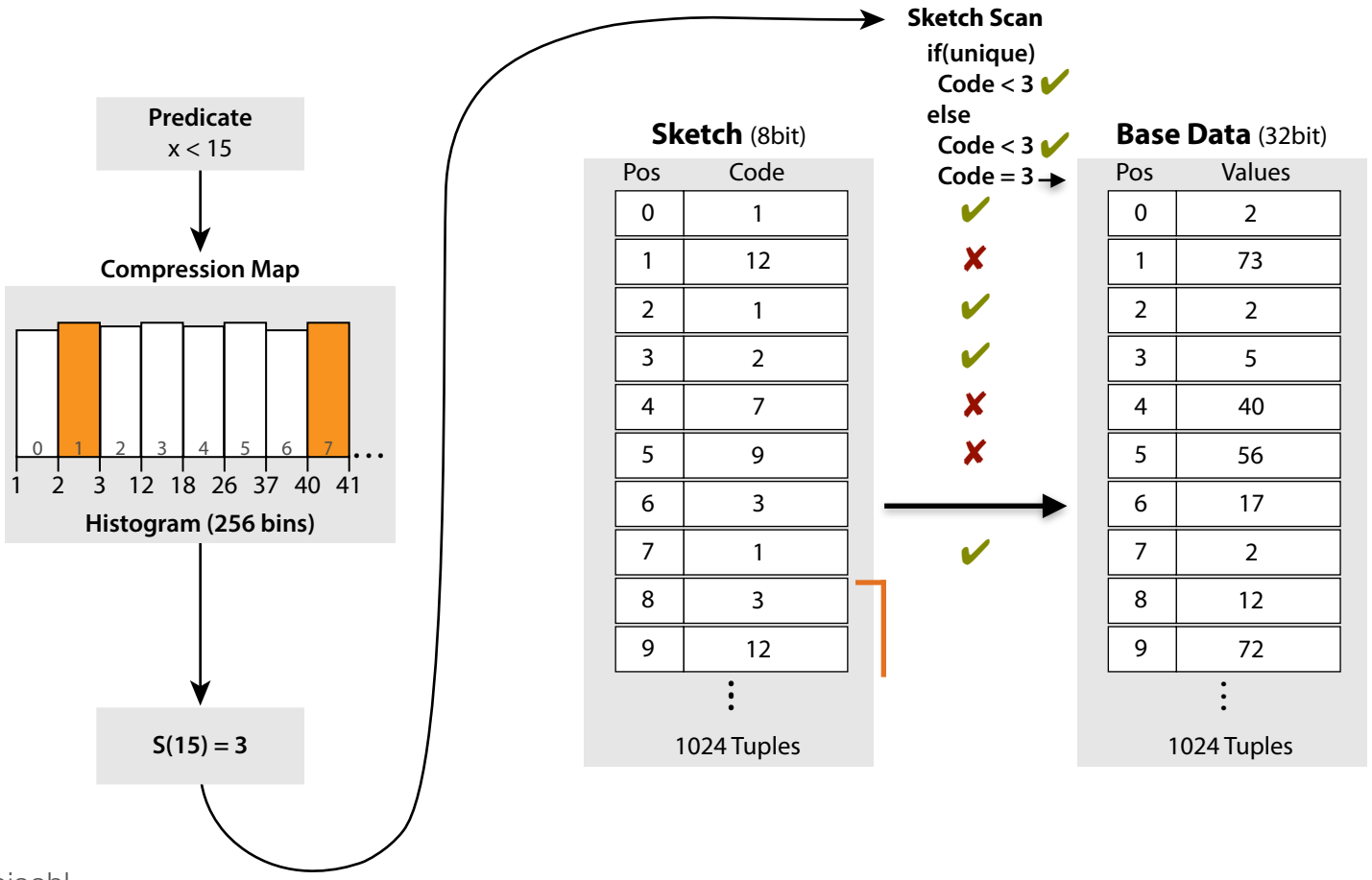
Column Sketches

Order Preserving - Lookup SIMD



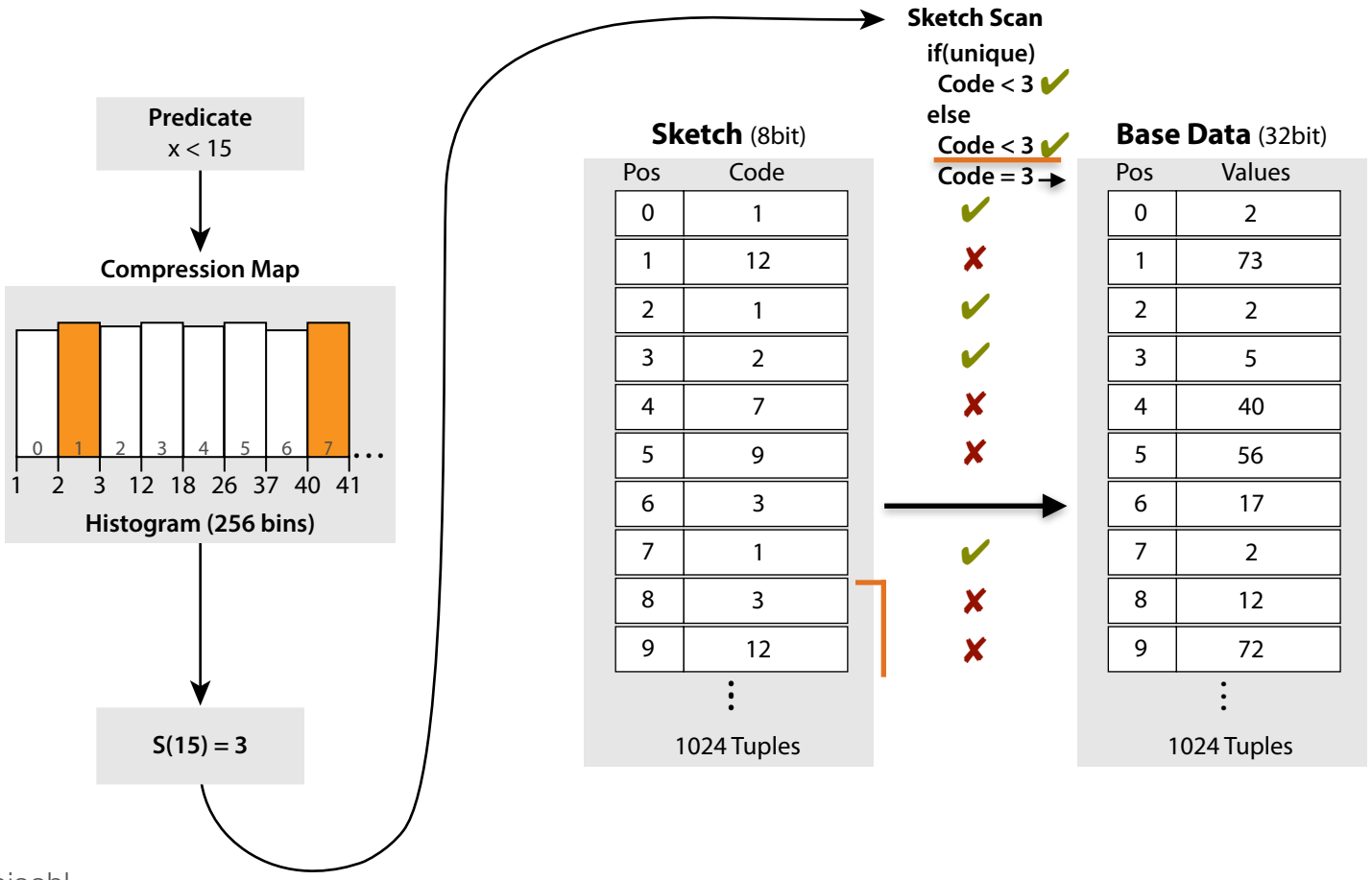
Column Sketches

Order Preserving - Lookup SIMD



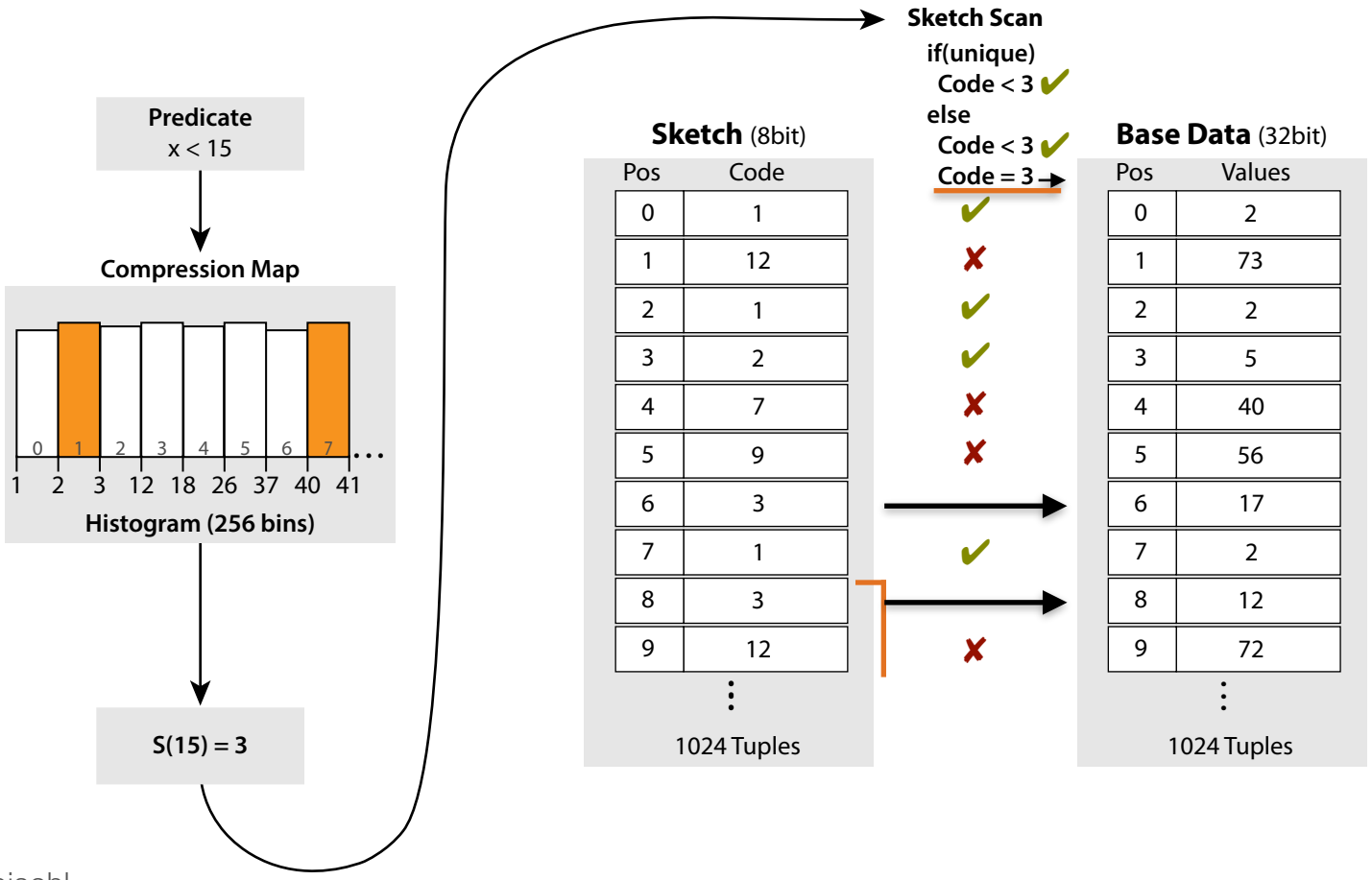
Column Sketches

Order Preserving - Lookup SIMD



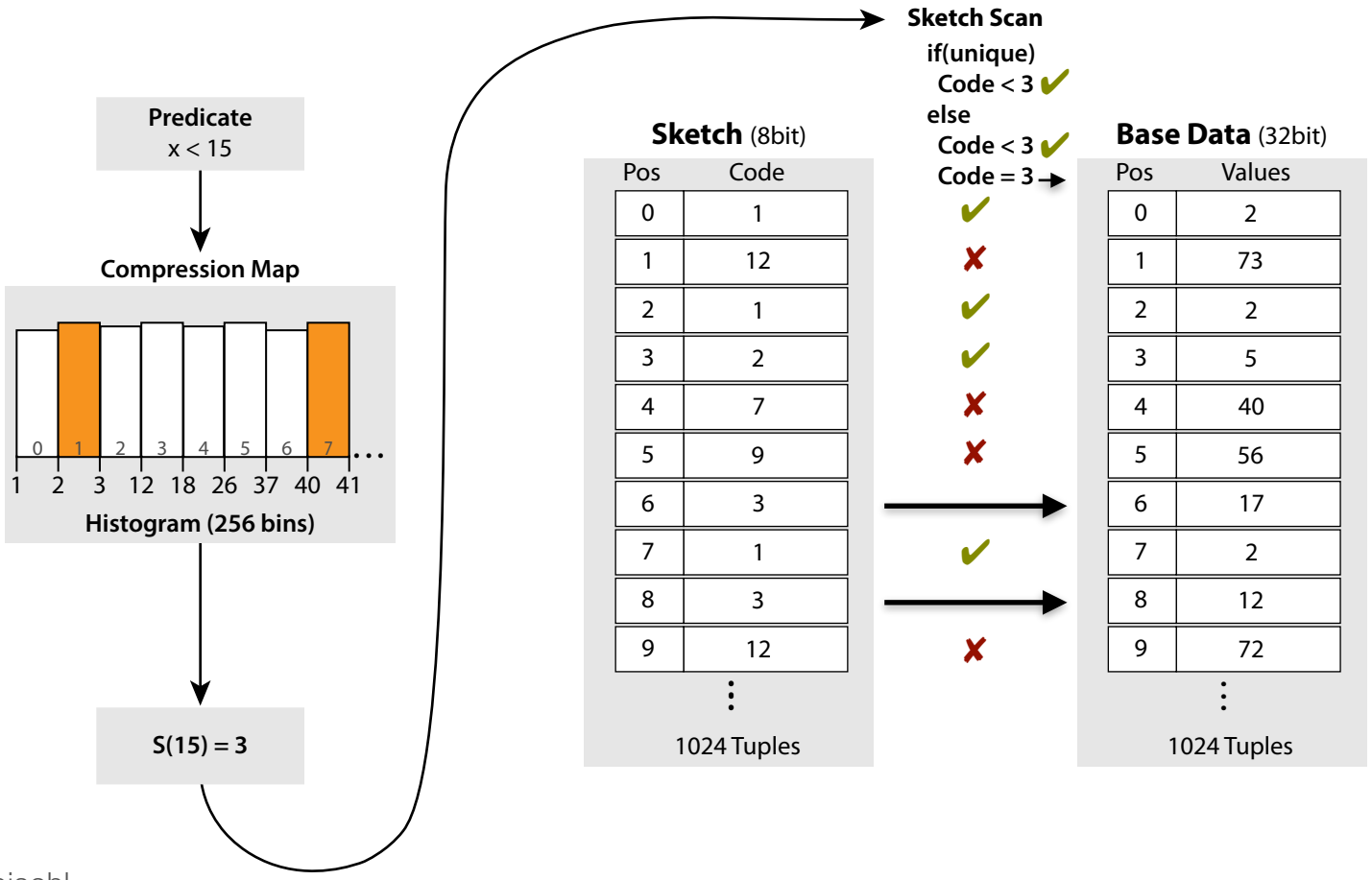
Column Sketches

Order Preserving - Lookup SIMD



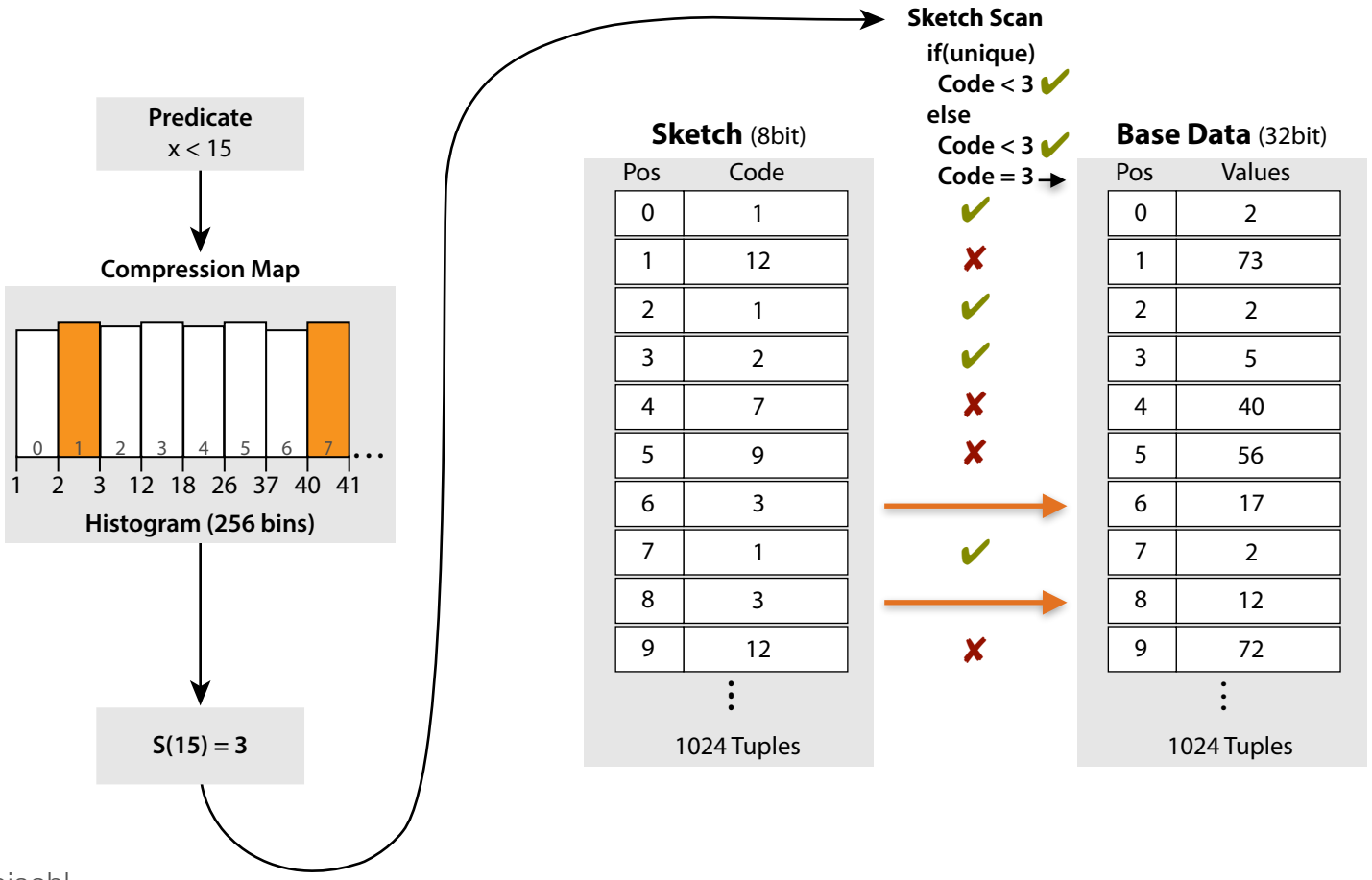
Column Sketches

Order Preserving - Lookup SIMD



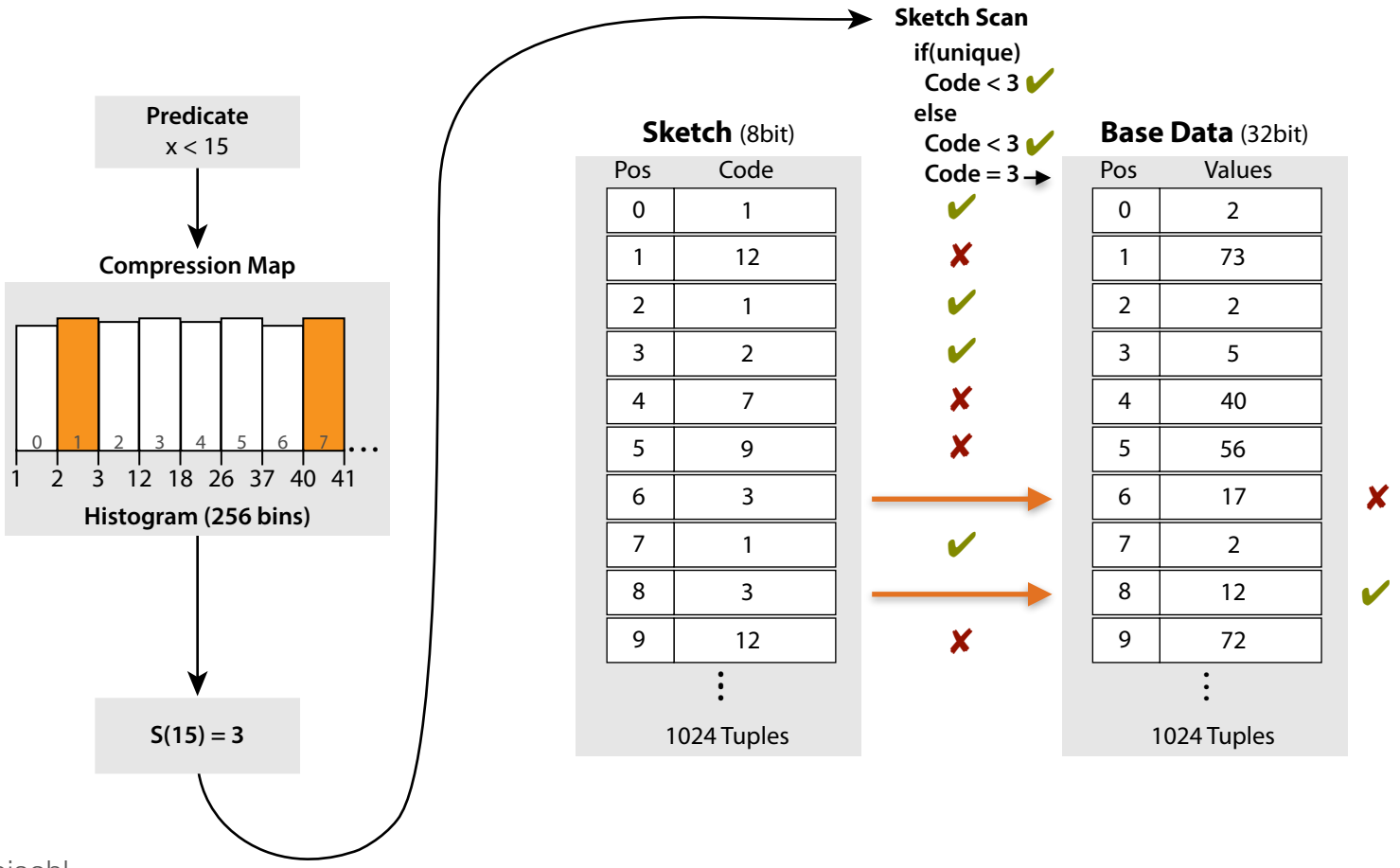
Column Sketches

Order Preserving - Lookup SIMD



Column Sketches

Order Preserving - Lookup SIMD



Column Sketches

Non-Order Preserving - Build

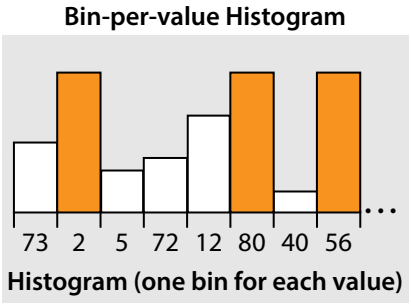
I. Cluster the data temporarily

Base Data (32bit)

Pos	Values
0	2
1	73
2	2
3	5
4	40
5	56
6	17
7	2
8	12
9	72
⋮	

1024 Tuples

Create a histogram (1 value per bin)



Column Sketches

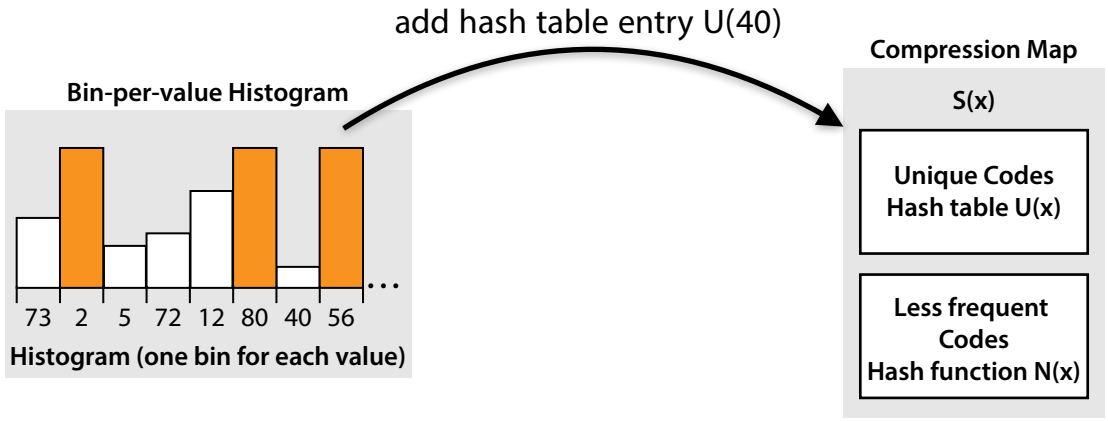
Non-Order Preserving - Build

II. Find frequent values

1. Check each bin in the histogram

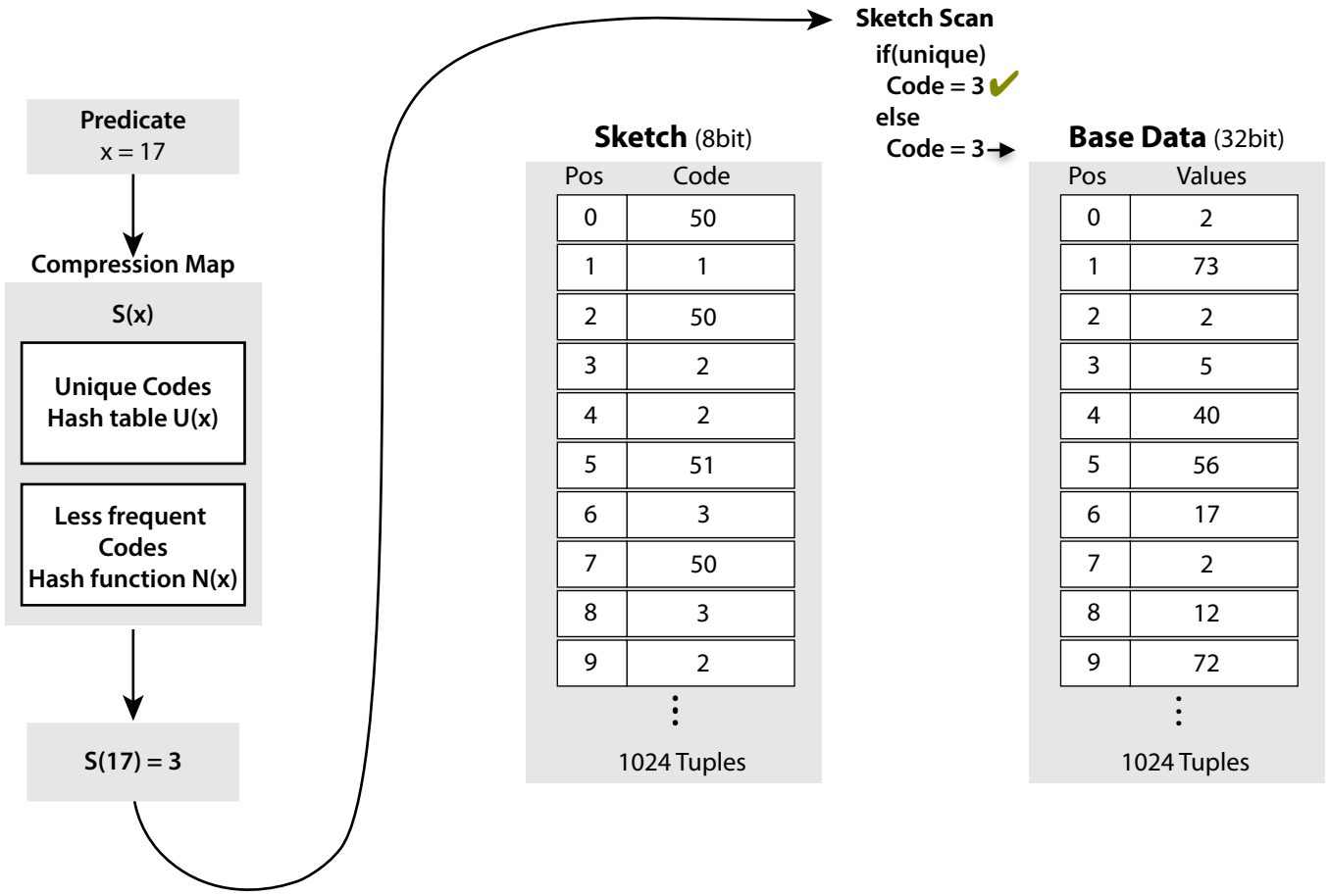
2. Frequency > $\frac{n}{c}$:
   - unique
   - assign value in the hash table

else:
   - code = output of N(x)



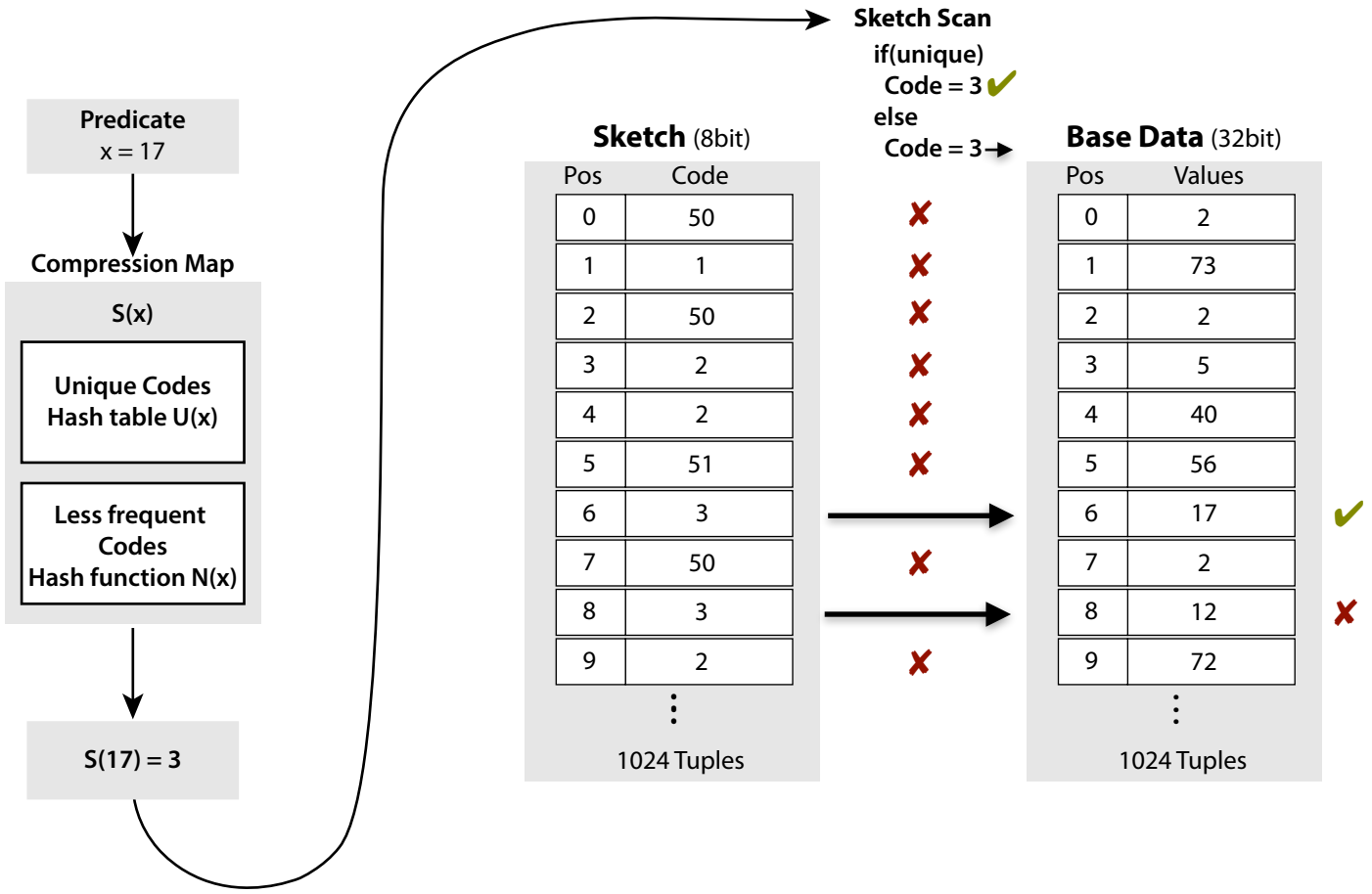
Column Sketches

Non-Order Preserving - Lookup



Column Sketches

Non-Order Preserving - Lookup



Evaluation

Parameters

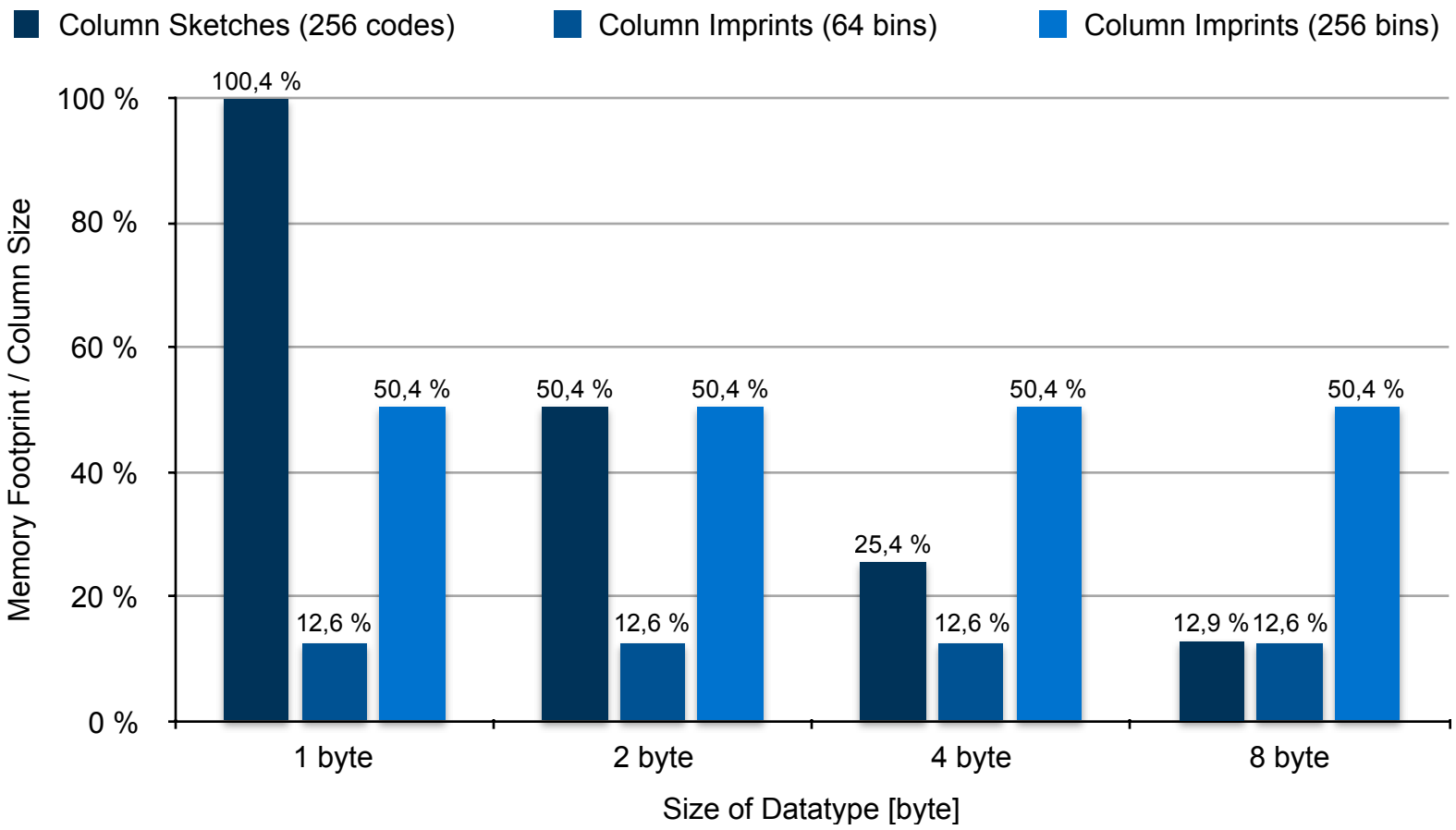
- Uniform dataset: artificially created
- Skewed dataset: Airline on-time performance [6]
- Data blocks of 2^{16} tuples
- Multi-threaded, but each data block executed on a single thread

Machine

CPU	Intel i7-4578
# of Cores	2
# of Threads	4
Base Frequency	3.0 GHz
L1 Data Cache	32 KiB
L2 Data Cache	256 KiB
RAM	16GiB

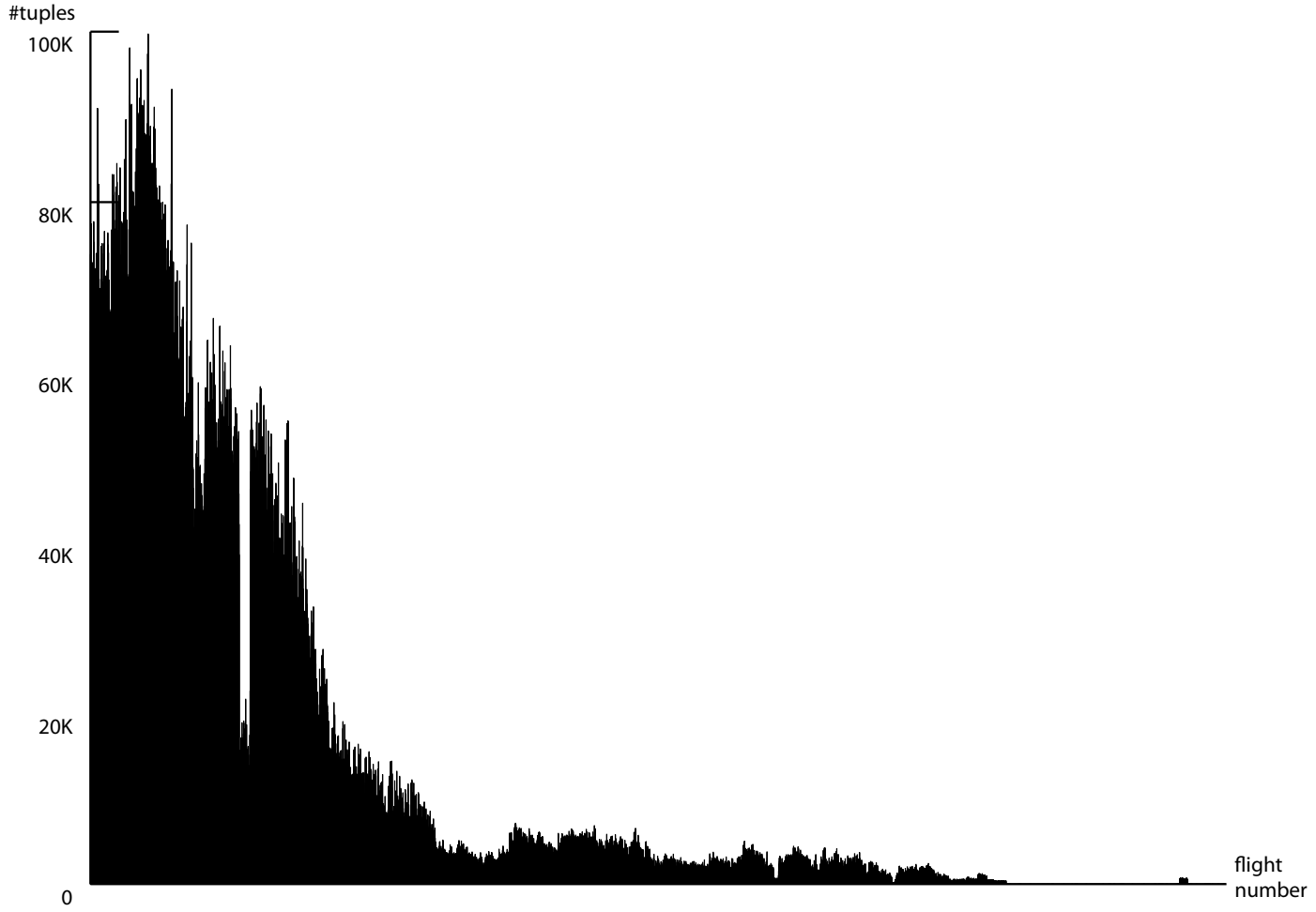
Evaluation

Memory Footprint



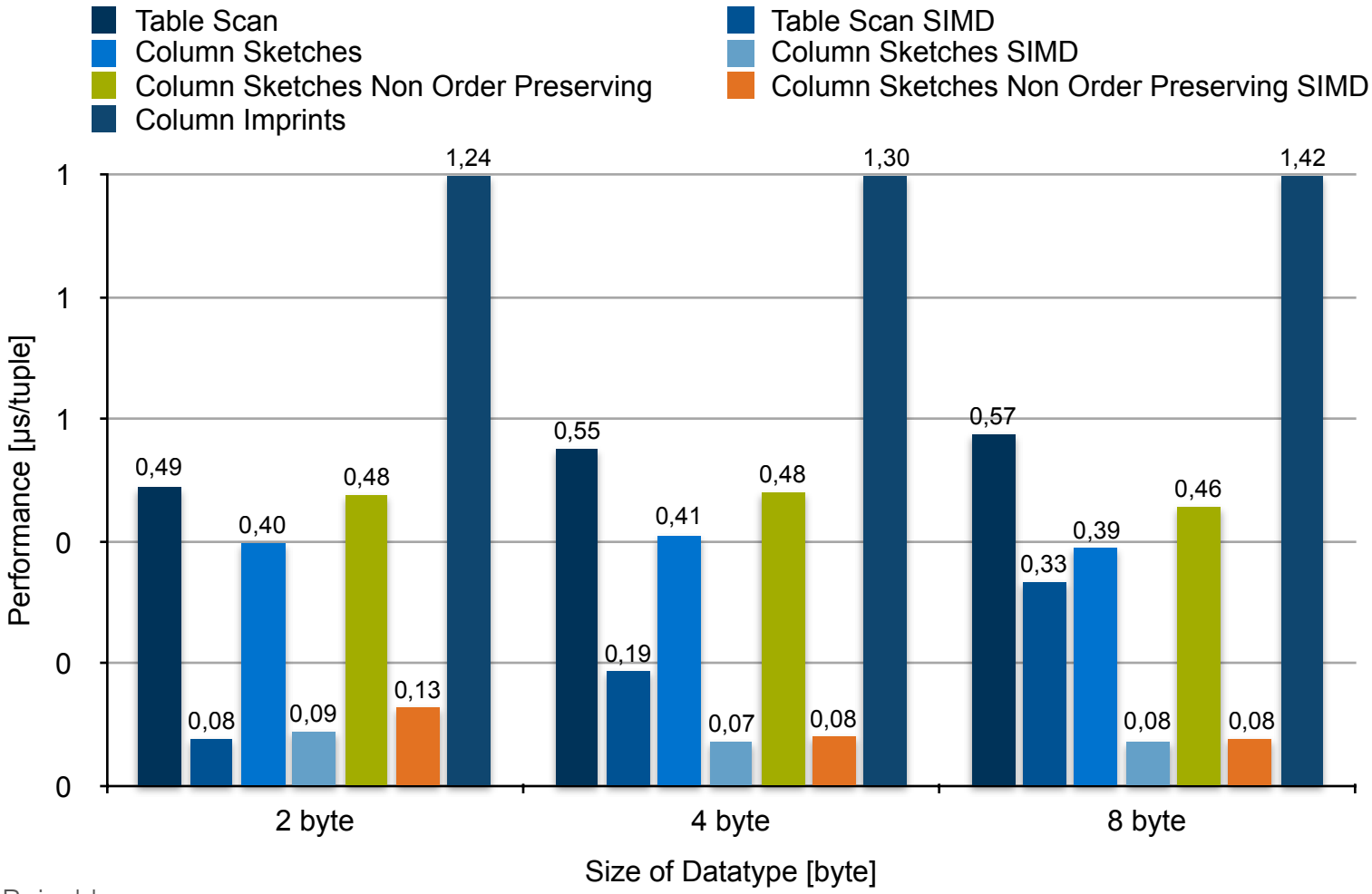
Evaluation

Performance - Skewed Data



Evaluation

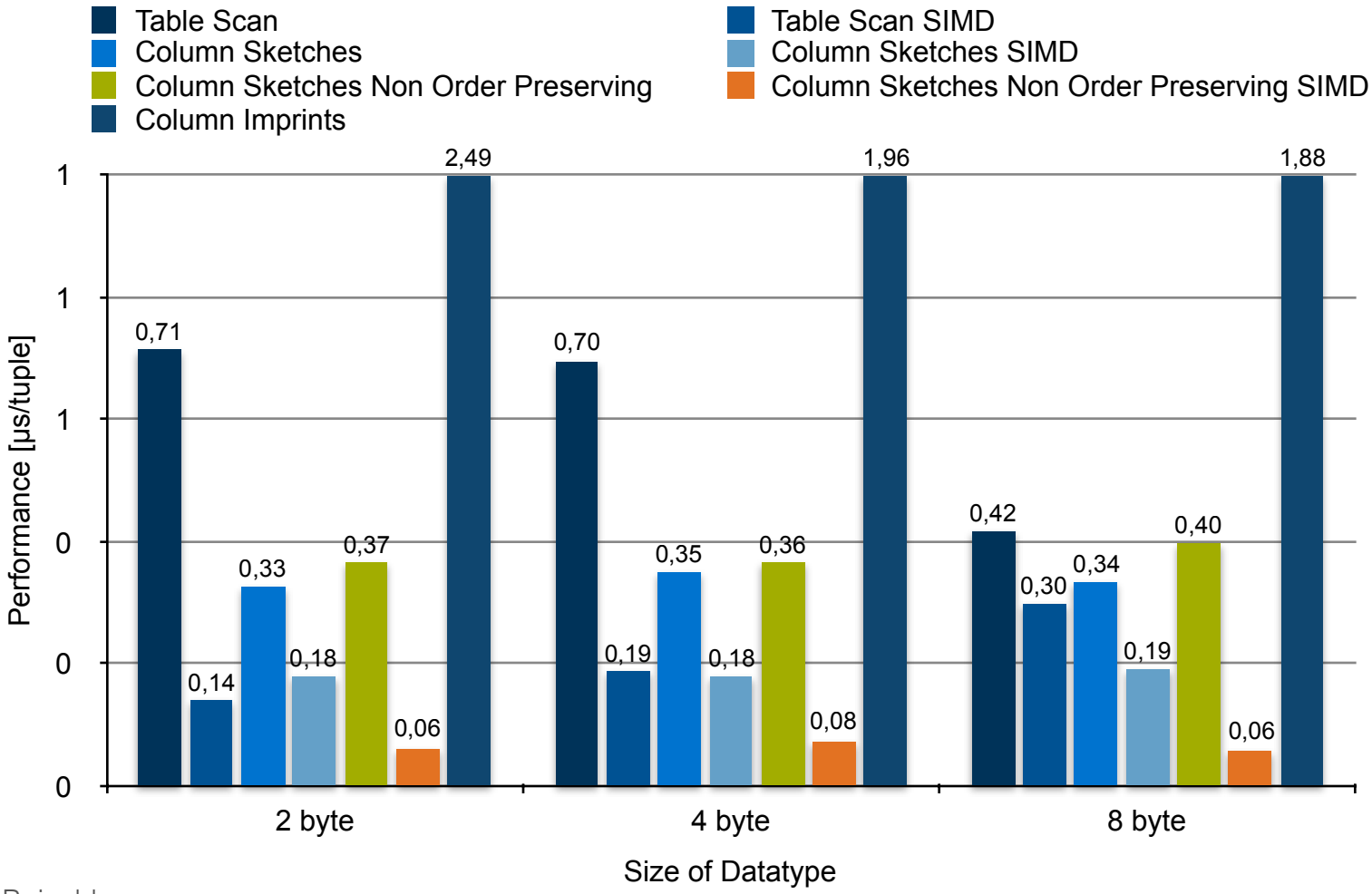
Performance - Skewed Data



Skewed distributed data, source [6], domain [0;9912], 5.963.776 tuples, all data types signed, lookup: EQ 8 (0.26% selectivity), non unique code

Evaluation

Performance - Uniformly Distributed Data



Uniformly distributed data, artificially generated, domain [0;255], 2^{16} tuples, all data types signed, lookup: EQ 9 (0.38% selectivity), non unique code

Summary

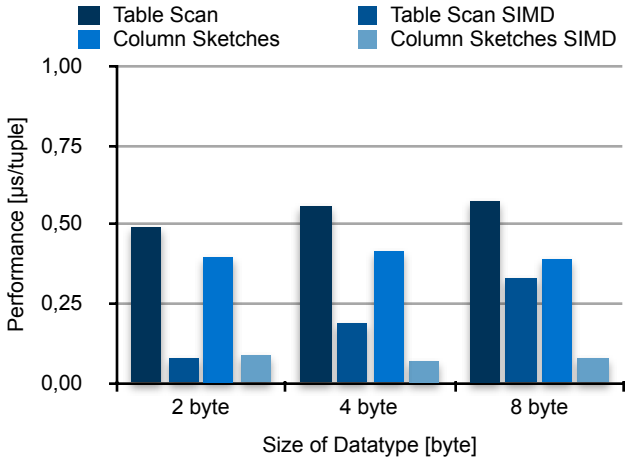
✓ Column Sketches are robust independent of
- data ordering
- data distribution
- query selectivity

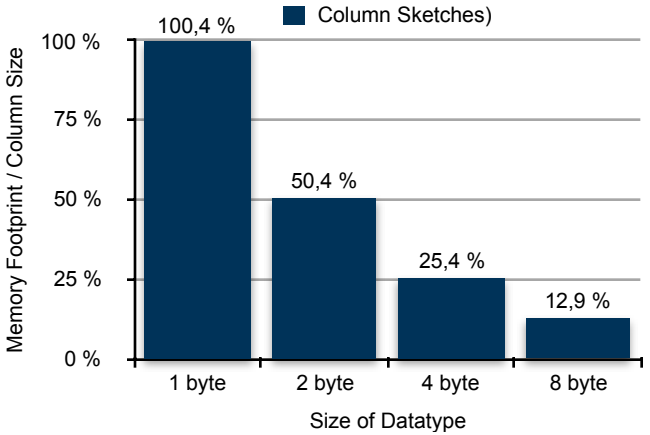
✓ Higher performance due to less memory movement

✘ Only lightweight for 8-byte data types

✘ SIMD-variant only suited for signed data types

✘ Histogram algorithm requires sorting base data





References

[1] B. Hentschel, M. S. Kester, S. Idreos. *Column Sketches: A Scan Accelerator for Rapid and Robust Predicate Evaluation*

[2] G. Moerkotte. *Small materialized aggregates : a light weight index structure for data warehousing*

[3] L. Sidirourgos, M. Kersten. *Column Imprints: A Secondary Index Structure*

[4] Y. Li, J. M. Patel. BitWeaving: *Fast Scans for Main Memory Data Processing*

[5] H. Lang, T. Mühlbauer, F. Funke, P. A. Boncz, T. Neumann, A. Kemper: *Data blocks: Hybrid oltp and olap on compressed storage using both vectorization and compilation*

[6] Airline on-time performance*, http://stat-computing.org/dataexpo/2009*

Thank you very much for your attention

Questions?