Lecture #04

Carnegie Mellon University

ADVANCED DATABASE SYSTEMS

OLAP Indexes

Andy Pavlo // 15-721 // Spring 2023

LAST CLASS

We discussed the advantages of the columnar storage model via PAX for OLAP workloads.

All attributes in a columnar database must be fixed-length to enable offset addressing.

If the DBMS assumes that its data files are immutable, it enables several optimization opportunities.

OBSERVATION

OLTP DBMSs use indexes to find individual tuples without performing sequential scans.

→ Tree-based indexes (B+Trees) are meant for queries with low selectivity predicates.
→ Also need to accommodate incremental updates.

But OLAP queries don't necessarily need to find individual tuples and data files are read-only.

How can we speed up sequential scans?

SEQUENTIAL SCAN OPTIMIZATIONS

Data Prefetching

Task Parallelization / Multi-threading

Clustering / Sorting

Late Materialization

Materialized Views / Result Caching

Data Skipping

Data Parallelization / Vectorization

Code Specialization / Compilation

DATA SKIPPING

Approach #1: Approximate Queries (Lossy)

→ Execute queries on a sampled subset of the entire table to produce approximate results.
→ Examples: BlinkDB, Redshift, ComputeDB, XDB, Oracle, Snowflake, Google BigQuery, DataBricks

Approach #2: Data Pruning (Loseless)

→ Use auxiliary data structures for evaluating predicates to quickly identify portions of a table that the DBMS can skip instead of examining tuples individually.
→ DBMS must consider trade-offs between ***scope*** vs. ***filter efficacy***, ***manual*** vs. ***automatic***.

DATA CONSIDERATIONS

Predicate Selectivity

→ How many tuples will satisfy a query's predicates.

Skewness

→ Whether an attribute has all unique values or contain many repeated values.

Clustering / Sorting

→ Whether the table is pre-sorted on the attributes accessed in a query's predicates.

TODAY'S AGENDA

Zone Maps
Bitmap Indexes
Bit-Slicing
Bit-Weaving
Column Imprints
Column Sketches

ZONE MAPS

Pre-computed aggregates for the attribute values in a block of tuples. DBMS checks the zone map first to decide whether it wants to access the block.

→ Originally called ***Small Materialized Aggregates*** (SMA)
→ DBMS automatically creates/maintains this meta-data.

```
SELECT * FROM table
 WHERE val > 600
```

Original Data

val
100
200
300
400
400

Zone Map

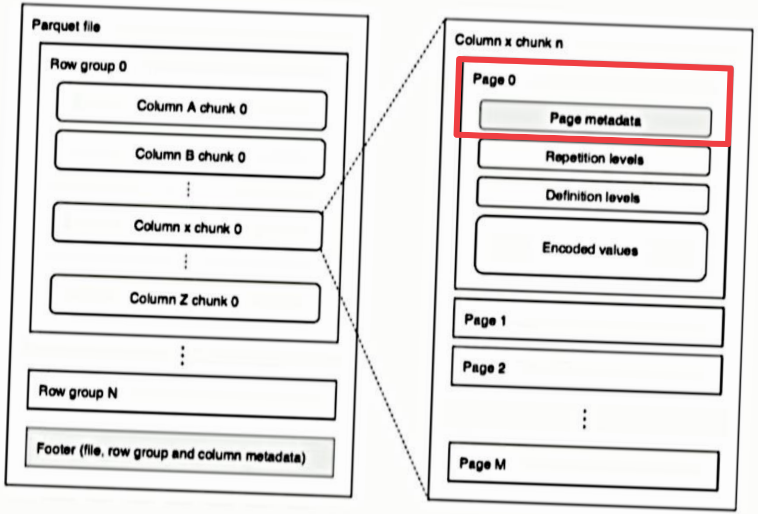
type	val
MIN	100
MAX	400
AVG	280
SUM	1400
COUNT	5

SMALL MATERIALIZED AGGREGATES: A LIGHT WEIGHT INDEX STRUCTURE FOR DATA WAREHOUSING
VLDB 1998

ZONE MAPS

Parquet: data organization

- Data organization
  - Row-groups (*default 128MB*)
  - Column chunks
  - Pages (*default 1MB*)
    - Metadata
      - Min
      - Max
      - Count
    - Rep/def levels
    - Encoded values



databricks

SELECT
WHERE

SMALL MAT
LIGHT WEIGHT INDEX STRUCTURE
DATA WAREHOUSING
VLDB 1998

OBSERVATION

Trade-off between scope vs. filter efficacy.

→ If the scope is too large, then the zone maps will be useless.

→ If the scope is too small, then the DBMS will spend too much checking zone maps.

Zone Maps are only useful when the target attribute's position and values are correlated.

BITMAP INDEXES

Store a separate Bitmap for each unique value for an attribute where an offset in the vector corresponds to a tuple.

→ The i^{th} position in the Bitmap corresponds to the i^{th} tuple in the table.

Typically segmented into chunks to avoid allocating large blocks of contiguous memory.

→ Example: One per row group in PAX.

MODEL 204 ARCHITECTURE AND PERFORMANCE
HIGH PERFORMANCE TRANSACTION SYSTEMS 1987

BITMAP INDEXES

Original Data

id	lit?
1	Y
2	Y
3	Y
4	N
6	Y
7	N
8	Y
9	Y

Compressed Data

id	lit?	
	Y	N
1	1	0
2	1	0
3	1	0
4	0	1
6	1	0
7	0	1
8	1	0
9	1	0

BITMAP INDEXES

Original Data

id	lit?
1	Y
2	Y
3	Y
4	N
6	Y
7	N
8	Y
9	Y

Compressed Data

id	lit?	
	Y	N
1	1	0
2	1	0
3	1	0
4	0	1
6	1	0
7	0	1
8	1	0
9	1	0

BITMAP INDEXES: EXAMPLE

```
CREATE TABLE customer_dim (
  id INT PRIMARY KEY,
  name VARCHAR(32),
  email VARCHAR(64),
  address VARCHAR(64),
  zip_code INT
);
```

```
SELECT id, email FROM customer_dim
 WHERE zip_code IN (15216,15217,15218);
```

Take the intersection of three bitmaps to find matching tuples.

Assume we have 10 million tuples.
43,000 zip codes in the US.
→ 10000000 × 43000 = **53.75GB**

This is **wasteful** because most entries in the bitmaps will be zeros.
→ Original: 10000000 × 32-bit = **40MB**

BITMAP INDEX: DESIGN CHOICES

Encoding Scheme
→ How to represent and organize data in a Bitmap.

Compression
→ How to reduce the size of sparse Bitmaps.

BITMAP INDEX: ENCODING

Approach #1: Equality Encoding

→ Basic scheme with one Bitmap per unique value.

Approach #2: Range Encoding

→ Use one Bitmap per interval instead of one per value.
→ Example: PostgreSQL BRIN

Approach #3: Hierarchical Encoding

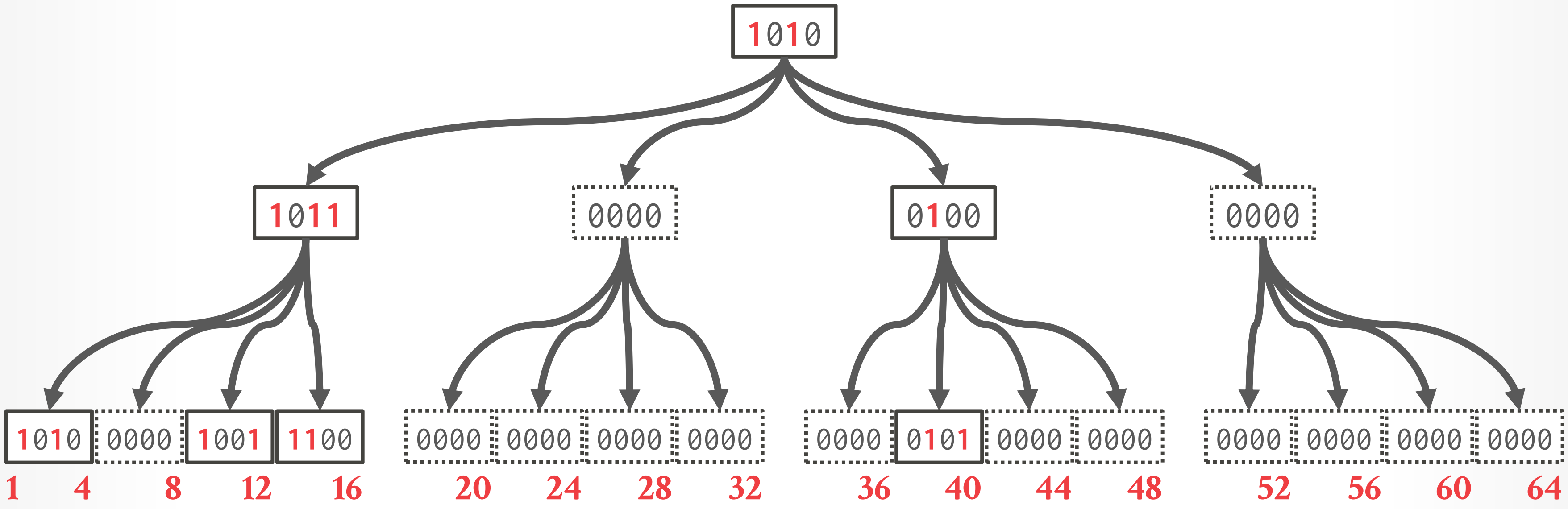
→ Use a tree to identify empty key ranges.

Approach #4: Bit-sliced Encoding

→ Use a Bitmap per bit location across all values.

HIERARCHICAL ENCODING

Offsets: 1, 3, 9, 12, 13, 14, 38, 40



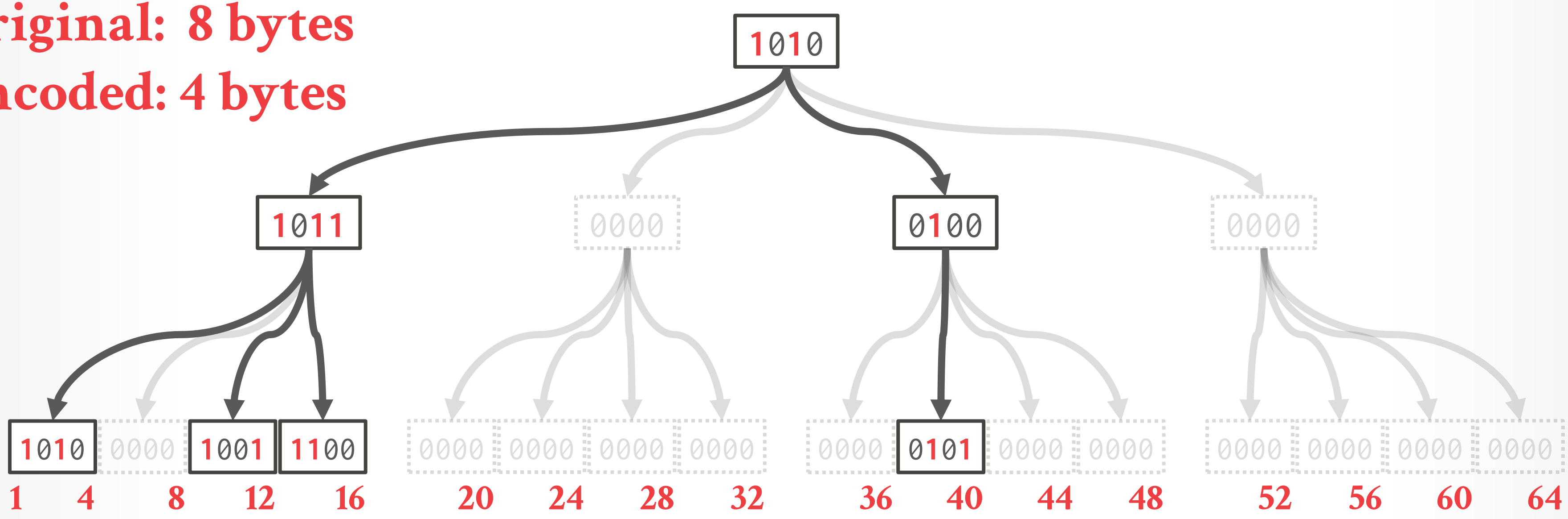
HIERARCHICAL BITMAP INDEX: AN EFFICIENT AND SCALABLE INDEXING TECHNIQUE FOR SET-VALUED ATTRIBUTES
ADVANCES IN DATABASES AND INFORMATION SYSTEMS 2003

HIERARCHICAL ENCODING

Offsets: 1, 3, 9, 12, 13, 14, 38, 40

Original: 8 bytes

Encoded: 4 bytes



HIERARCHICAL BITMAP INDEX: AN EFFICIENT AND SCALABLE INDEXING TECHNIQUE FOR SET-VALUED ATTRIBUTES
ADVANCES IN DATABASES AND INFORMATION SYSTEMS 2003

BIT-SLICED ENCODING

Original Data

id	zipcode
1	21042
2	15217
3	02903
4	90220
6	14623
7	53703

Bit-Slices

null	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	1	0	0	0	1	1	0	0	1	0

bin(21042)→ 00101001000110010

Source: Jignesh Patel

BIT-SLICED ENCODING

Original Data

id	zipcode
1	21042
2	15217
3	02903
4	90220
6	14623
7	53703

Bit-Slices

null	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	1	0	0	0	1	1	0	0	1	0
0	0	0	0	1	1	1	0	1	1	0	1	1	1	0	0	0	1
0	0	0	0	0	0	1	0	1	1	0	1	0	1	0	1	1	1
0	1	0	1	1	0	0	0	0	0	0	1	1	0	1	1	0	0
0	0	0	0	1	1	1	0	0	1	0	0	0	1	1	1	1	1
0	0	1	1	0	1	0	0	0	1	1	1	0	0	0	1	1	1

```
SELECT * FROM customer_dim
 WHERE zipcode < 15217
```

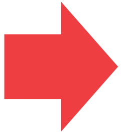
Walk each slice and construct a result bitmap.

Source: Jignesh Patel

BIT-SLICED ENCODING

Original Data

id	zipcode
1	21042
2	15217
3	02903
4	90220
6	14623
7	53703



Bit-Slices

null	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	1	0	0	0	1	1	0	0	1	0
0	0	0	0	1	1	1	0	1	1	0	1	1	1	0	0	0	1
0	0	0	0	0	0	1	0	1	1	0	1	0	1	0	1	1	1
0	1	0	1	1	0	0	0	0	0	0	1	1	0	1	1	0	0
0	0	0	0	1	1	1	0	0	1	0	0	0	1	1	1	1	1
0	0	1	1	0	1	0	0	0	1	1	1	0	0	0	1	1	1

```sql
SELECT * FROM customer_dim
 WHERE zipcode < 15217
```

Walk each slice and construct a result bitmap.
Skip entries that have **1** in first 3 slices (16, 15, 14)

Source: Jignesh Patel

BIT-SLICED ENCODING

Bit-slices can also be used for efficient aggregate computations.

Example: **SUM(attr)** using Hamming Weight

→ First, count the number of **1**s in **slice**$_{17}$ and multiply the count by 2^{17}

→ Then, count the number of **1**s in **slice**$_{16}$ and multiply the count by 2^{16}

→ Repeat for the rest of slices…

Intel added **POPCNT** SIMD instruction in 2008.

BITWEAVING

Alternative storage layout for columnar databases that is designed for efficient predicate evaluation on compressed data using SIMD.

→ Order-preserving dictionary encoding.
→ Bit-level parallelization.
→ Only require common instructions (no scatter/gather)

Implemented in Wisconsin's QuickStep engine.

→ Became an Apache Incubator project in 2016 but then died in 2018.

BITWEAVING: FAST SCANS FOR MAIN MEMORY DATA PROCESSING
SIGMOD 2013

BITWEAVING – STORAGE LAYOUTS

Approach #1: Horizontal

→ Row-oriented storage at the bit-level

Approach #2: Vertical

→ Column-oriented storage at the bit-level

HORIZONTAL STORAGE

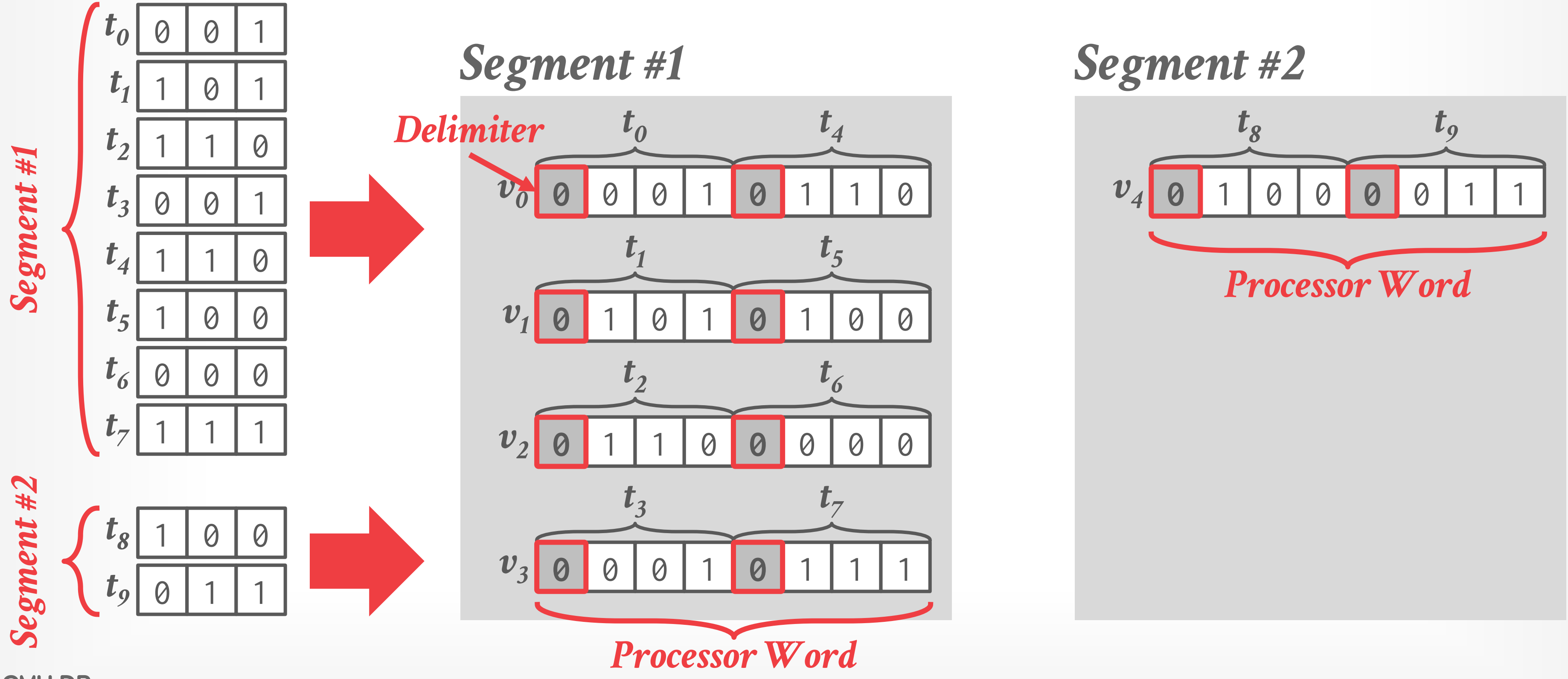
Segment #1

t_0	0	0	1	=1
t_1	1	0	1	=5
t_2	1	1	0	=6
t_3	0	0	1	=1
t_4	1	1	0	=6
t_5	1	0	0	=4
t_6	0	0	0	=0
t_7	1	1	1	=7

Segment #2

t_8	1	0	0	=4
t_9	0	1	1	=3

HORIZONTAL STORAGE



BITWEAVING/H – EXAMPLE

```sql
SELECT * FROM table
 WHERE val < 5
```

	t_0				t_4			
X =	0	0	0	1	0	1	1	0
	5				5			
Y =	0	1	0	1	0	1	0	1
$mask$ =	0	1	1	1	0	1	1	1
$(Y+(X\oplus mask))\wedge\neg mask$=	$\boxed{1}$	0	0	0	$\boxed{0}$	0	0	0
	$1 < 5$				$5 < 6$			

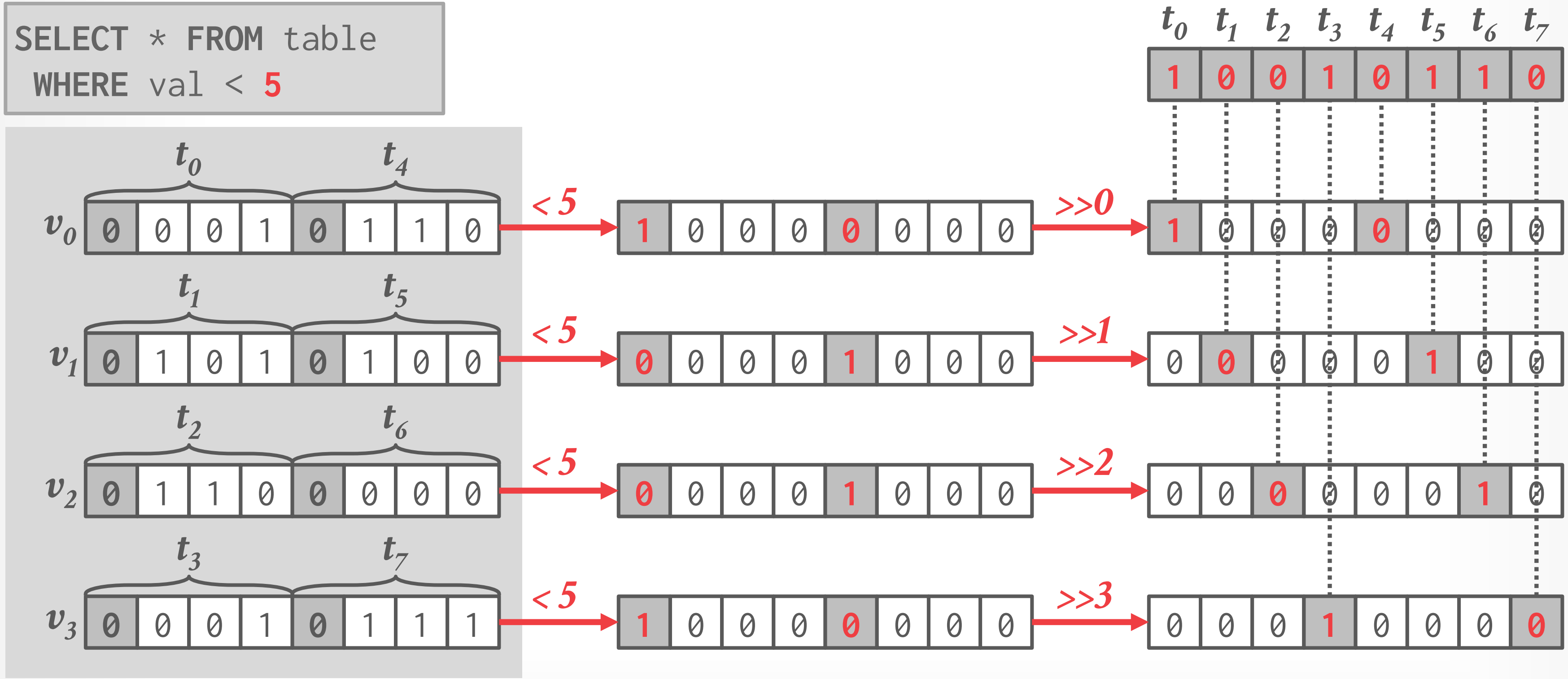
Only requires three instructions to evaluate a single word.

Works on any word size and encoding length.

Paper contains algorithms for other operators.

BITWEAVING/H – EXAMPLE

```
SELECT * FROM table
 WHERE val < 5
```



Source: Jignesh Patel

SELECTION VECTOR

SIMD comparison operators produce a bit mask that specifies which tuples satisfy a predicate.
→ DBMS must convert it into column offsets.

Approach #1: Iteration

Approach #2: Pre-computed Positions Table

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
Selection Vector	1	0	0	1	0	1	1	0

```
tuples = [ ]
for (i=0; i<n; i++) {
  if sv[i] == 1
    tuples.add(i);
}
```

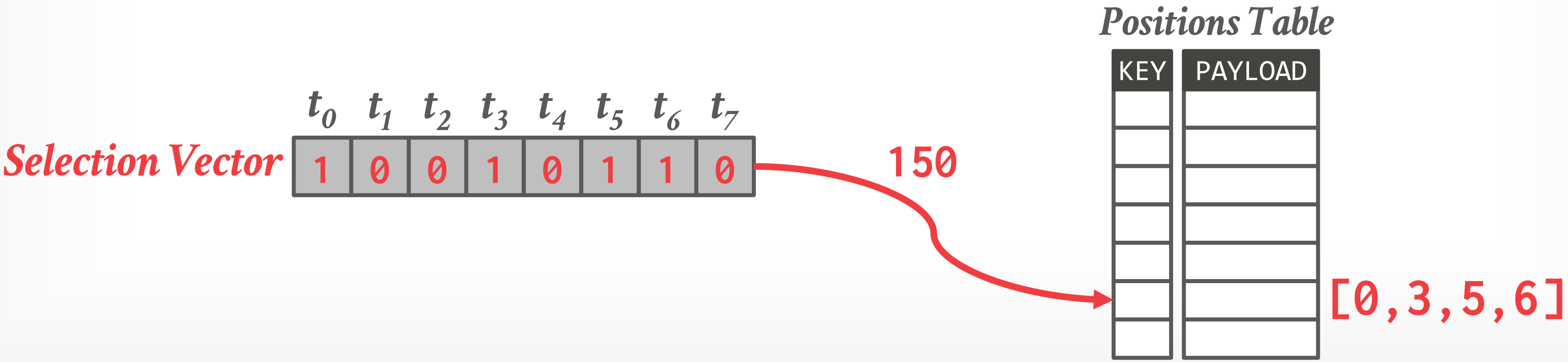
SELECTION VECTOR

SIMD comparison operators produce a bit mask that specifies which tuples satisfy a predicate.

→ DBMS must convert it into column offsets.

Approach #1: Iteration

Approach #2: Pre-computed Positions Table



VERTICAL STORAGE

Segment #1

t_0	0	0	1
t_1	1	0	1
t_2	1	1	0
t_3	0	0	1
t_4	1	1	0
t_5	1	0	0
t_6	0	0	0
t_7	1	1	1

Segment #2

t_8	1	0	0
t_9	0	1	1

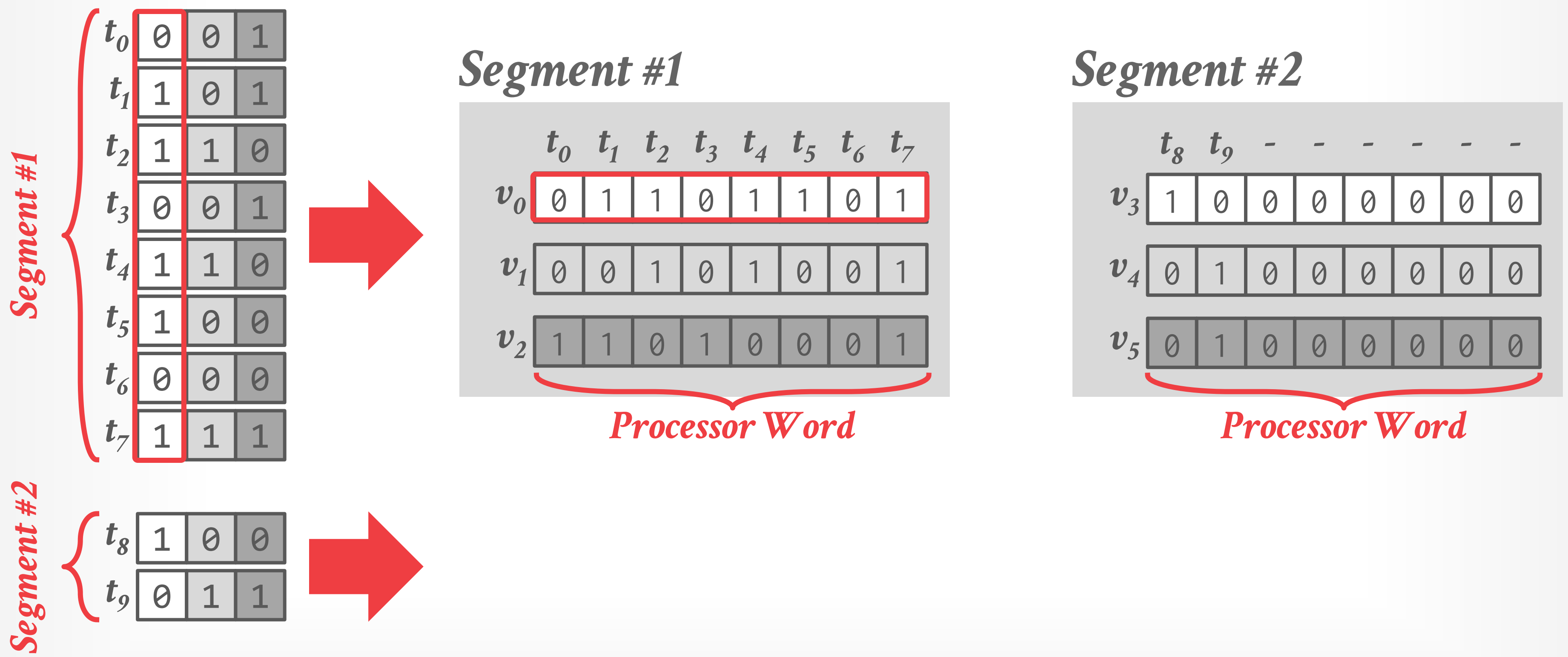
Segment #1

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
v_0	0	1	1	0	1	1	0	1
v_1	0	0	1	0	1	0	0	1
v_2	1	1	0	1	0	0	0	1

Segment #2

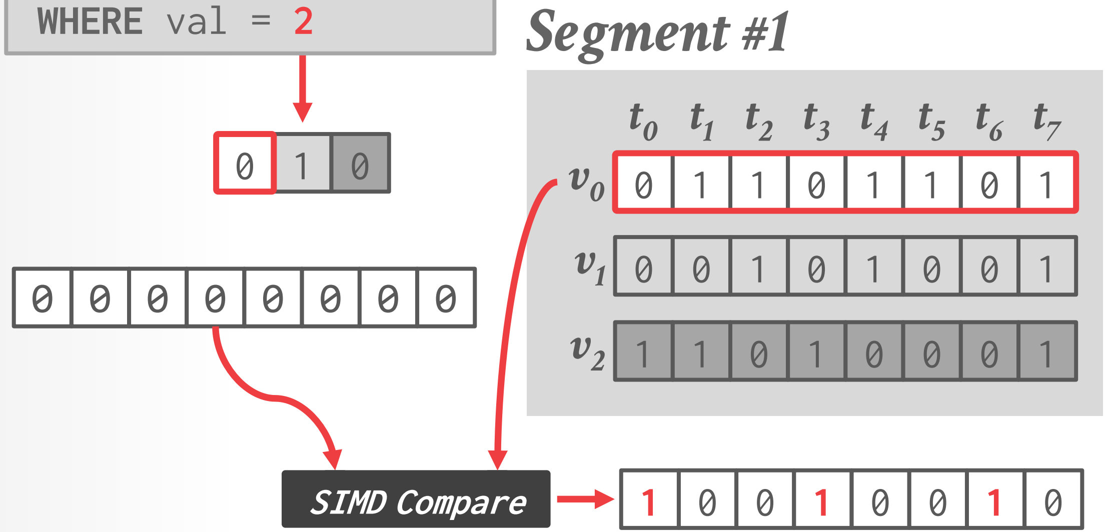
	t_8	t_9	-	-	-	-	-	-
v_3	1	0	0	0	0	0	0	0
v_4	0	1	0	0	0	0	0	0
v_5	0	1	0	0	0	0	0	0

VERTICAL STORAGE

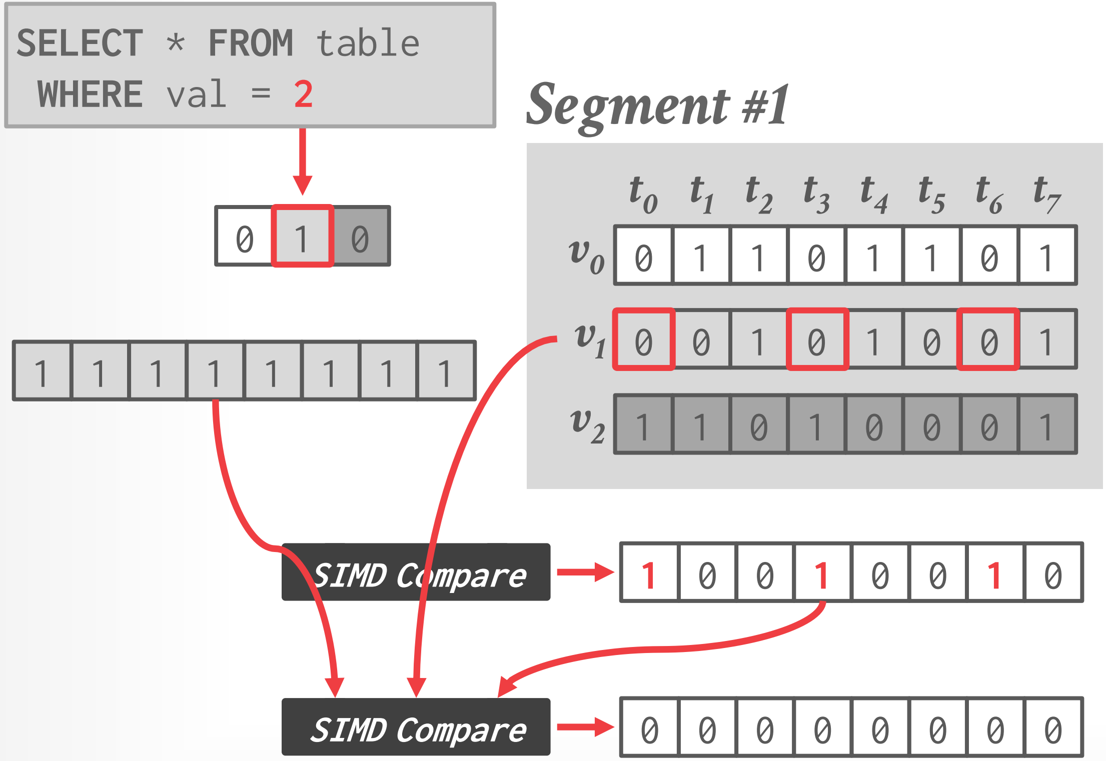


BITWEAVING/V – EXAMPLE

```
SELECT * FROM table
 WHERE val = 2
```



BITWEAVING/V – EXAMPLE



Can perform early pruning just like in BitMap indexes.

The last vector is skipped because all bits in previous comparison are zero.

TODAY'S AGENDA

~~Zone Maps~~
~~Bitmap Indexes~~
~~Bit-Slicing~~
~~Bit-Weaving~~
Column Imprints
Column Sketches

OBSERVATION

All the previous Bitmap schemes were about storing exact/loseless representations of columnar data.

The DBMS could give up some accuracy in exchange for faster evaluation in the common case.

→ It still must always check the original data to avoid false positives.

COLUMN IMPRINTS

Store a bitmap that indicates whether there is a bit set at a bit-slice of cache-line values.

COLUMN IMPRINTS: A SECONDARY INDEX STRUCTURE
SIGMOD 2013

COLUMN SKETCHES

A variation of range-encoded Bitmaps that uses a smaller sketch codes to indicate that a tuple's value exists in a range.

DBMS must automatically figure out the best mapping of codes.
→ Trade-off between distribution of values and compactness.
→ Assign unique codes to frequent values to avoid false positives.

COLUMN SKETCHES: A SCAN ACCELERATOR FOR RAPID AND ROBUST PREDICATE EVALUATION
SIGMOD 2013

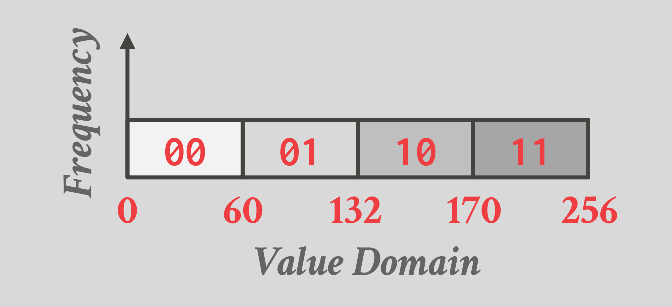
COLUMN SKETCHES

Original Data

val
13
191
56
92
81
140
231
172

8-bits

Histogram



Compression Map

60	00
132	01
170	10
256	11

Sketched Column

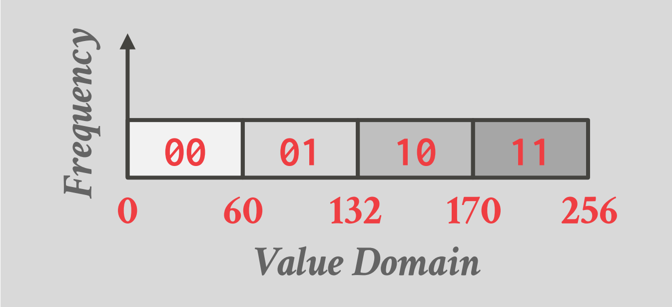
code
00
10
11
00
01
01
10
11

2-bits

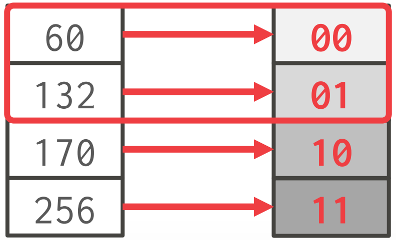
Source: Brian Hentschel

COLUMN SKETCHES

Histogram

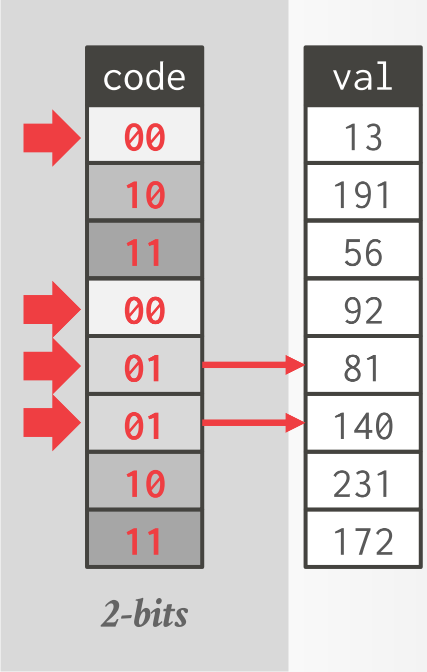


Compression Map



map(90) = 01

Sketched Column




```
SELECT * FROM table
 WHERE val < 90
```

Source: Brian Hentschel

PARTING THOUGHTS

Zone Maps are the most widely used method to accelerate sequential scans.

Bitmap indexes are more common in NSM DBMSs than columnar OLAP systems.

We're ignoring multi-dimensional and inverted indexes…

NEXT CLASS

Data Compression (Tuples, Indexes)

PROJECT #1 – FOREIGN DATA WRAPPER

You will create a foreign data wrapper for PostgreSQL for performing simple scans with predicate pushdown on a custom columnar data format.

The goal is to get you familiar with extending Postgres and writing a simple scan operator.

Due Date: Sunday Feb 26th

Prompt*: A dramatic renaissance painting of a scottish terrier riding on top of an African elephant running through Carnegie Mellon University.*

https://15721.courses.cs.cmu.edu/spring2023/project1.html

PROJECT #1 – TASKS

We provide a simple columnar data file format.

Write a parser for this file that can scan individual columns and stitching tuples back together.

Integrate it with Postgres as a foreign table.

Add support for predicate evaluation.

DEVELOPMENT HINTS

Write the basic parser code separately first, then connect it to Postgres.

We recommend completing this project in C++ using PGXS.

→ You can try asking Chi about doing it in Rust but he has better things to do in his life…

Gradescope is for meant for grading, **not** debugging. Write your own local tests.

THINGS TO NOTE

Do **not** change any file other than the ones that you submit to Gradescope.

Make sure you fork our version of Postgres.

Post your questions on Piazza or come to TA office hours.

PLAGIARISM WARNING

Your project implementation must be your own work.

→ You may **not** copy source code from other groups or the web.

Plagiarism will **not** be tolerated. See CMU's Policy on Academic Integrity for additional information.

NEXT CLASS

Two-Phase Locking

Isolation Levels