Lecture #22

Carnegie Mellon University

ADVANCED DATABASE SYSTEMS

Vectorized Execution (Part II)

@Andy_Pavlo // 15-721 // Spring 2018

TODAY'S AGENDA

Bit-Slicing
Bit-Weaving
Relaxed Operator Fusion (Prashanth)

BITMAP ENCODING

Original Data

id	sex
1	M
2	M
3	M
4	F
6	M
7	F
8	M
9	M

BITMAP ENCODING

Original Data

id	sex
1	M
2	M
3	M
4	F
6	M
7	F
8	M
9	M

Compressed Data

id	sex: M	sex: F
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 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.

Approach #3: Hierarchical Encoding

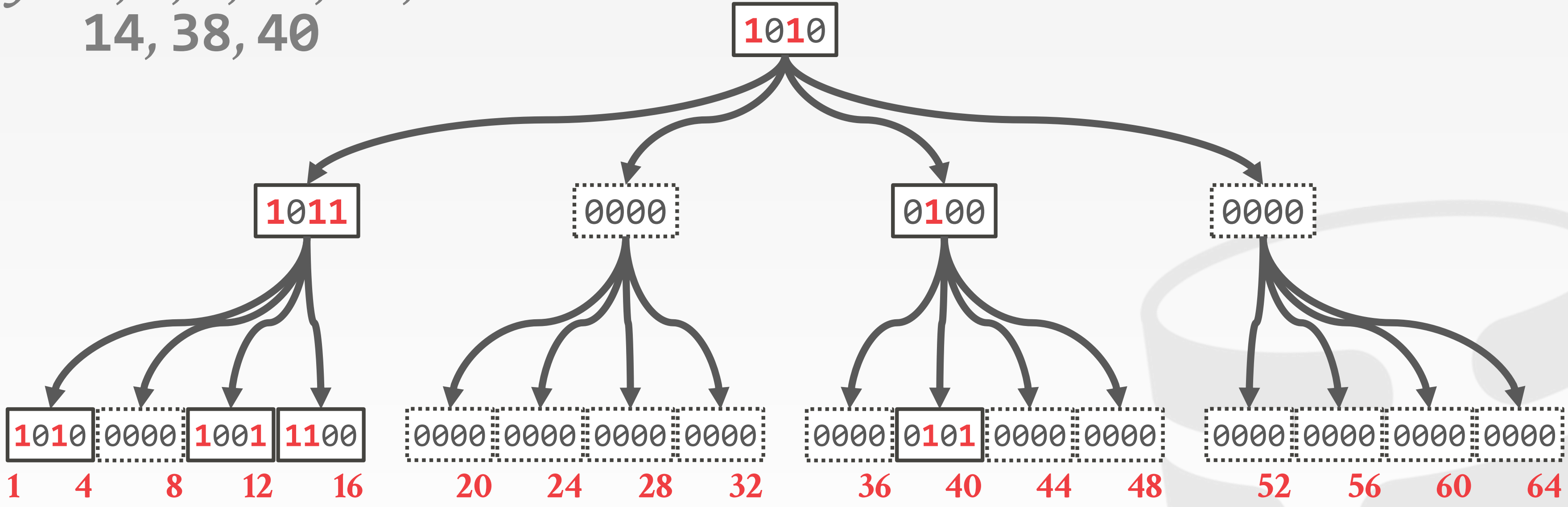
→ Use a tree to identify empty key ranges.

Approach #4: Bit-sliced Encoding

→ Use a Bitmap per bit location across all values.

HIERARCHICAL ENCODING

Keys: **1, 3, 9, 12, 13, 14, 38, 40**

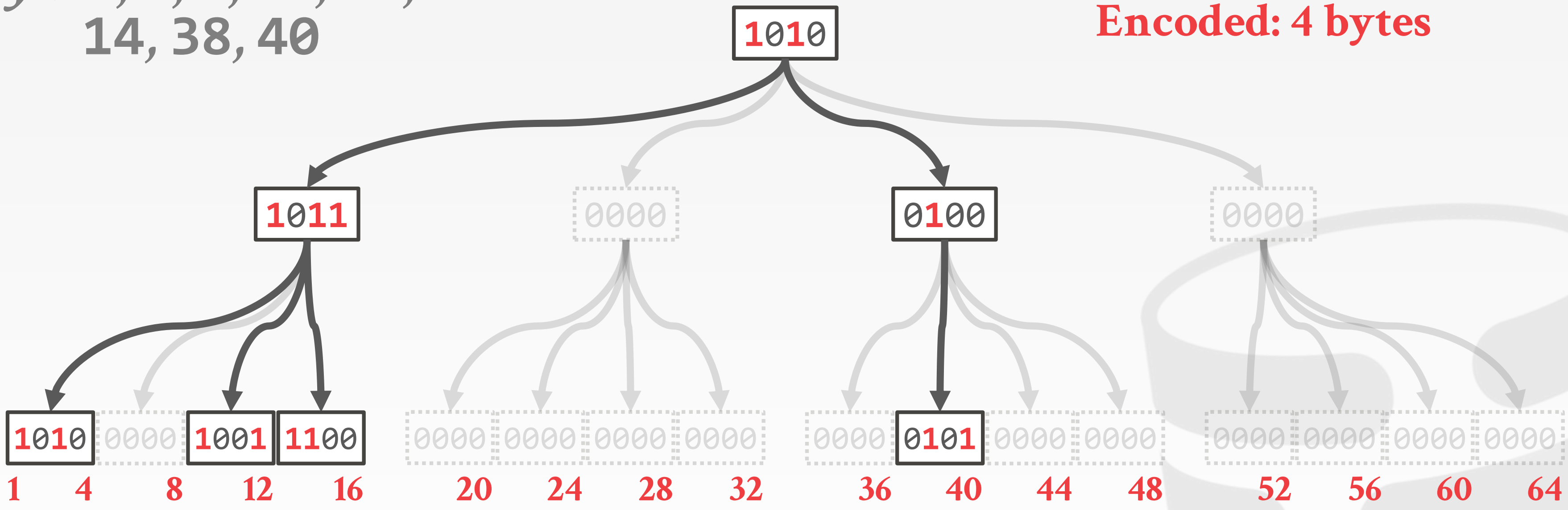


HIERARCHICAL BITMAP INDEX: AN EFFICIENT AND SCALABLE INDEXING TECHNIQUE FOR SET-VALUED ATTRIBUTES
Advances in Databases and Information Systems 2003

HIERARCHICAL ENCODING

Keys: **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

BIT-SLICED ENCODING

Original Data

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

Bit-Slices

N?	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	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

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

N?	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

N?	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
```

BIT-SLICED ENCODING

Original Data

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

Bit-Slices

N?	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

N?	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.

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)`

→ 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…

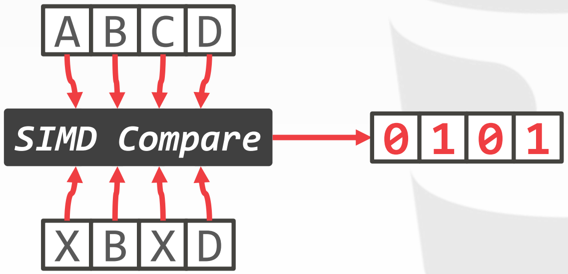
Intel added **POPCNT** SIMD instruction in 2008.

OBSERVATION

The bit width of compressed data does not always fit naturally into SIMD register lanes.

→ This means that the DBMS has to do extra work to transform data into the proper format.

Just because the **lanes** are fully utilized does not mean the **bits** are fully utilized…

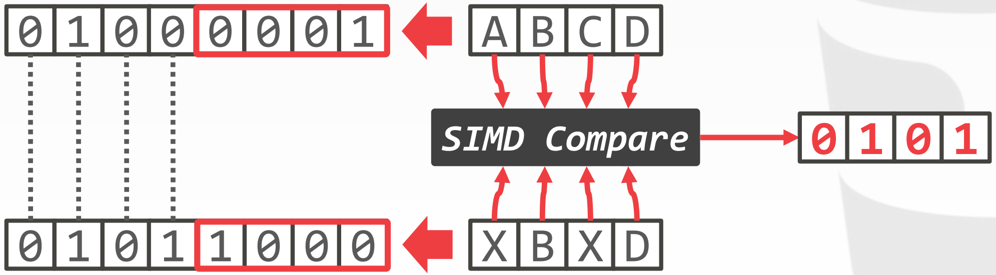


OBSERVATION

The bit width of compressed data does not always fit naturally into SIMD register lanes.

→ This means that the DBMS has to do extra work to transform data into the proper format.

Just because the **lanes** are fully utilized does not mean the **bits** are fully utilized…



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.

BITWEAVING: FAST SCANS FOR MAIN MEMORY DATA PROCESSING
SIGMOD 2013

BITWEAVIN

Alternative storage layout for co
that is designed for efficient pre
on compressed data using SIMD
→ Order-preserving dictionary encod
→ Bit-level parallelization.
→ Only require common instruction

Implemented in Wisconsin's Q
Became an Apache Incubator p

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

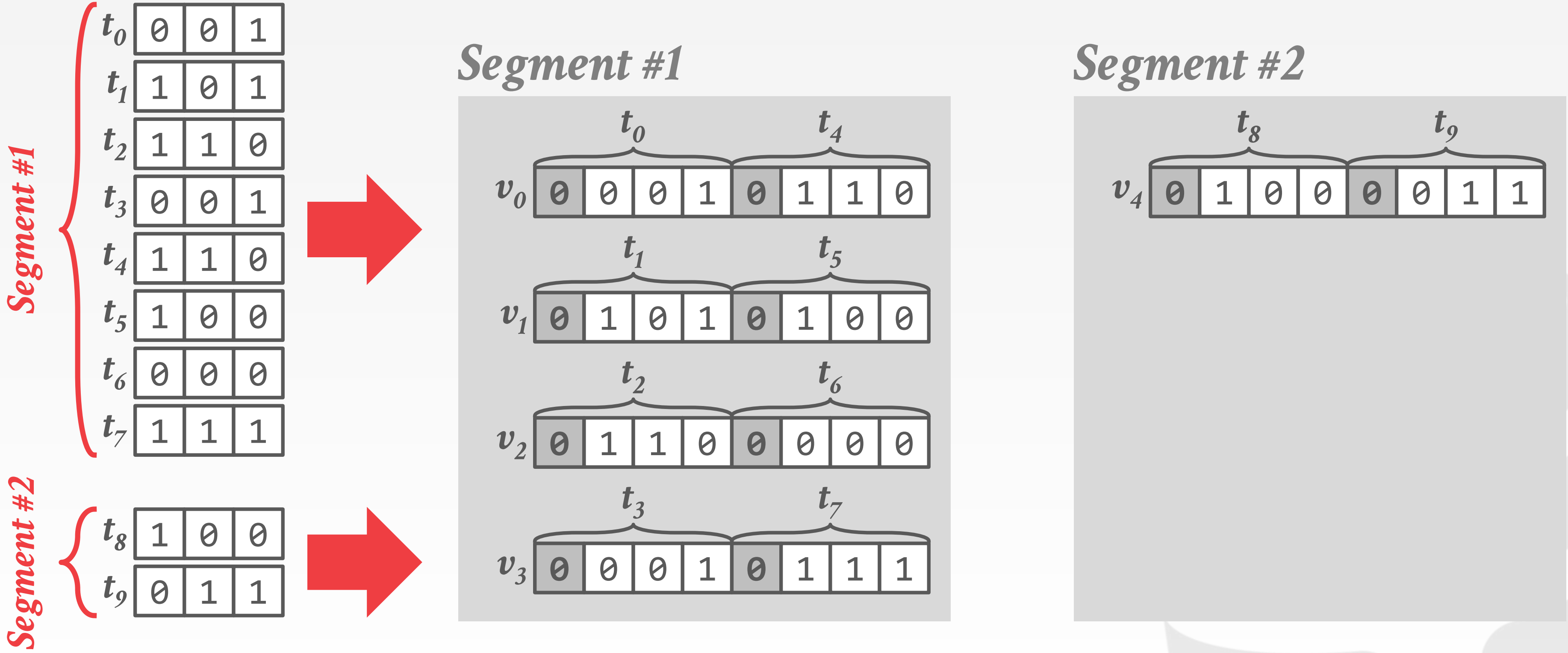
Segment #1

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*

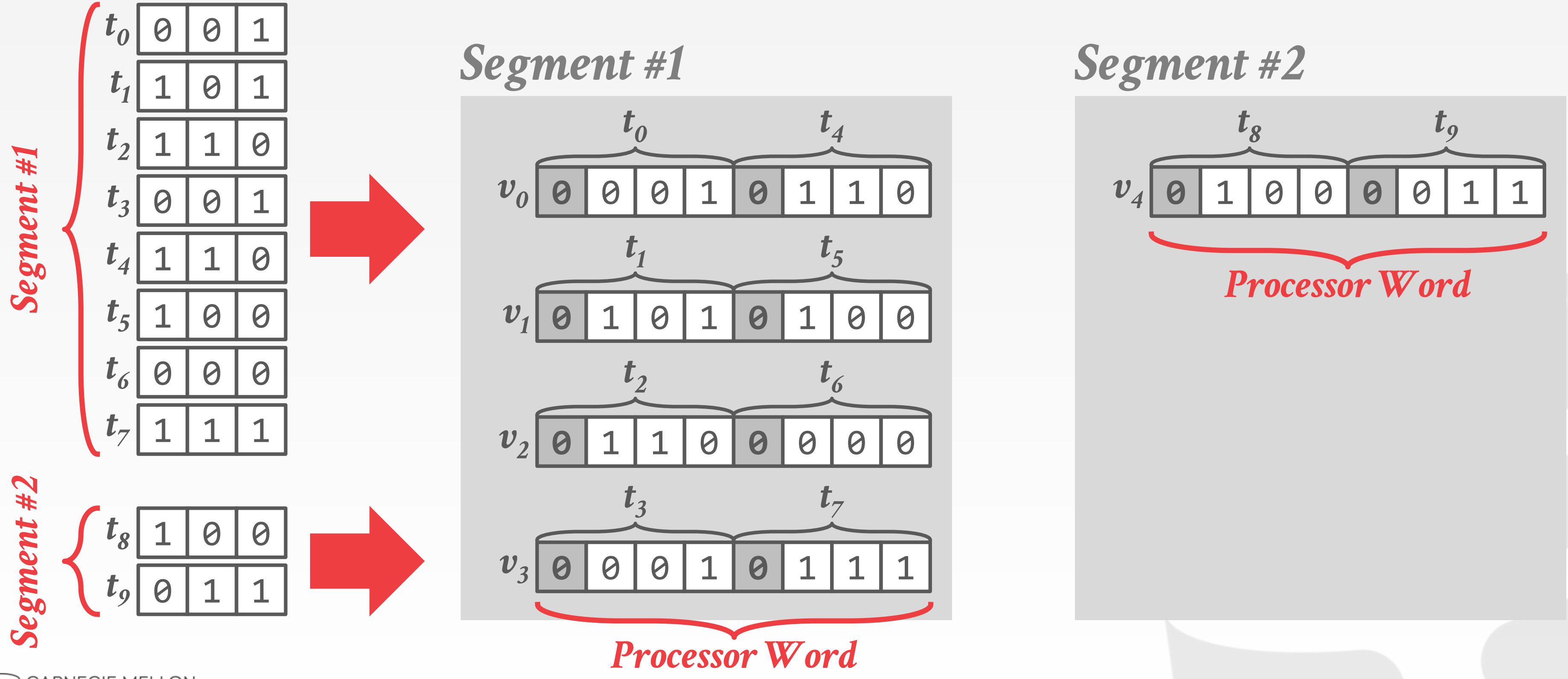
Segment #2

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

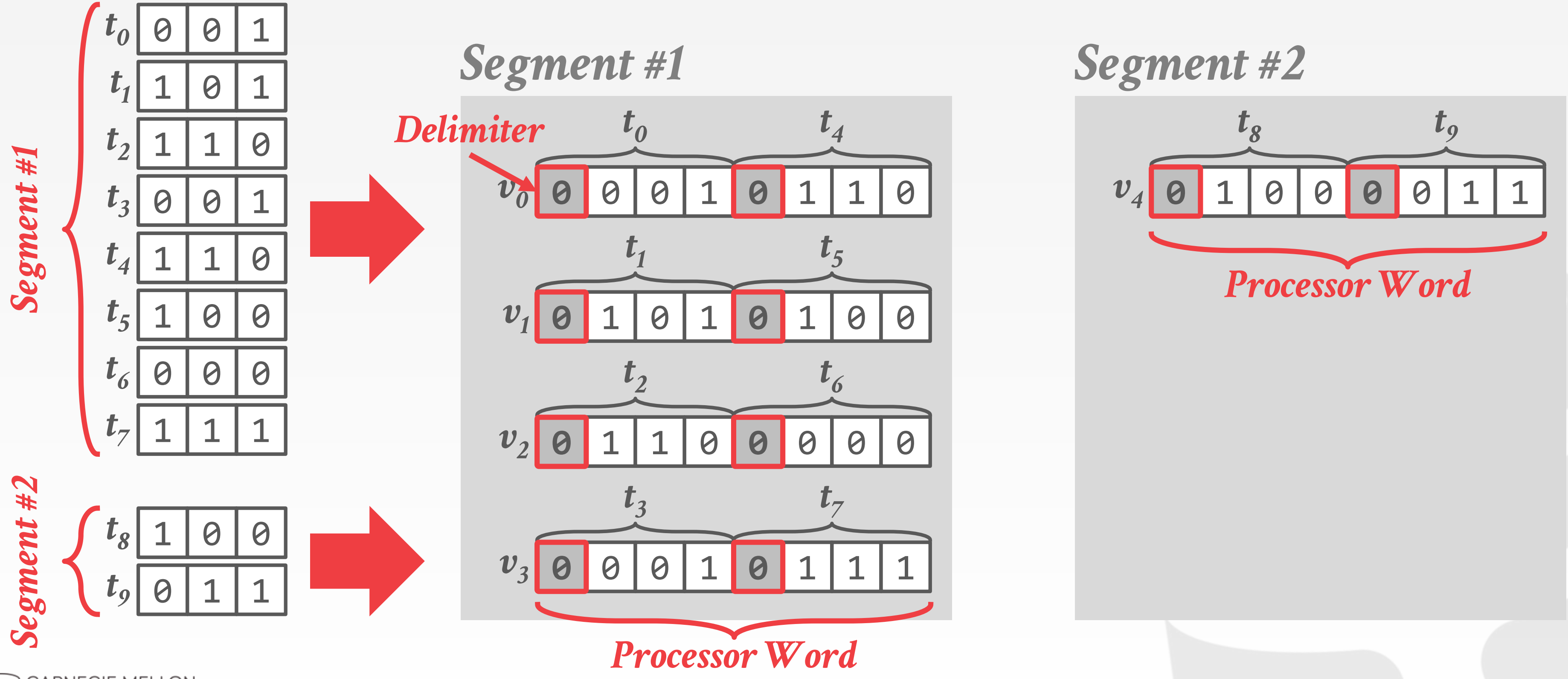
HORIZONTAL STORAGE



HORIZONTAL STORAGE

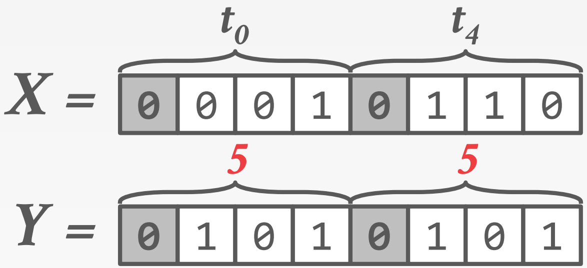


HORIZONTAL STORAGE



BITWEAVING/H – EXAMPLE

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



Source: Jignesh Patel

BITWEAVING/H – EXAMPLE

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

$$X = \underbrace{0\,0\,0\,1}_{t_0}\ \underbrace{0\,1\,1\,0}_{t_4}$$

$$Y = \underbrace{0\,1\,0\,1}_{5}\ \underbrace{0\,1\,0\,1}_{5}$$

$$mask = 0\,1\,1\,1\ 0\,1\,1\,1$$

$$(Y+(X \oplus mask)) \land \neg mask = 1\,0\,0\,0\ 0\,0\,0\,0$$

BITWEAVING/H – EXAMPLE

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

$$X = \overbrace{\boxed{0}\boxed{0}\boxed{0}\boxed{1}}^{t_0}\overbrace{\boxed{0}\boxed{1}\boxed{1}\boxed{0}}^{t_4}$$

$$Y = \overbrace{\boxed{0}\boxed{1}\boxed{0}\boxed{1}}^{5}\overbrace{\boxed{0}\boxed{1}\boxed{0}\boxed{1}}^{5}$$

$$mask = \boxed{0}\boxed{1}\boxed{1}\boxed{1}\boxed{0}\boxed{1}\boxed{1}\boxed{1}$$

$$(Y+(X \oplus mask)) \wedge \neg mask = \underbrace{\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}}_{\textit{Selection Vector}}$$

BITWEAVING/H – EXAMPLE

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

$$X = \underbrace{\overbrace{0\;0\;0\;1}^{t_0}}_{5}\;\underbrace{\overbrace{0\;1\;1\;0}^{t_4}}_{5}$$

$$Y = 0\;1\;0\;1\;\;0\;1\;0\;1$$

$$mask = 0\;1\;1\;1\;\;0\;1\;1\;1$$

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

BITWEAVING/H – EXAMPLE

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

$$X = \underbrace{0\,|\,0\,|\,0\,|\,1}_{t_0}\,|\,\underbrace{0\,|\,1\,|\,1\,|\,0}_{t_4}$$

$$Y = \overbrace{0\,|\,1\,|\,0\,|\,1}^{5}\,|\,\overbrace{0\,|\,1\,|\,0\,|\,1}^{5}$$

$$mask = 0\,|\,1\,|\,1\,|\,1\,|\,0\,|\,1\,|\,1\,|\,1$$

$$(Y+(X\oplus mask))\wedge\neg mask = \boxed{1}\,|\,0\,|\,0\,|\,0\,|\,\boxed{0}\,|\,0\,|\,0\,|\,0$$

$1 < 5$ $\quad$ $5 < 6$

Only requires three instructions to evaluate a single word.

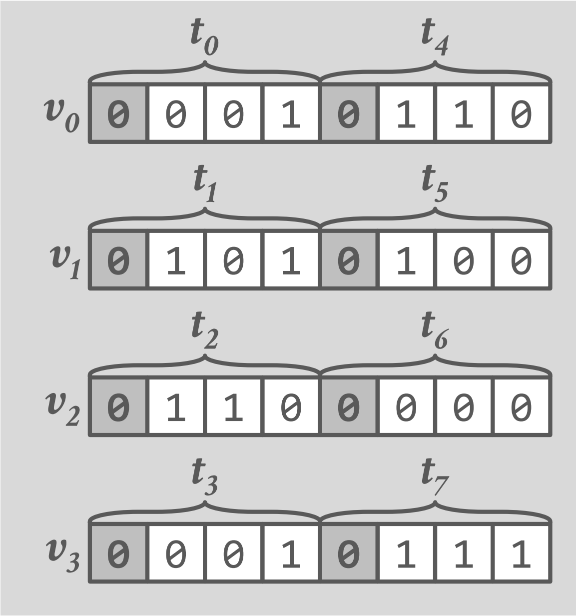
Works on any word size and encoding length.

Paper contains algorithms for other operators.

Source: Jignesh Patel

BITWEAVING/H – EXAMPLE

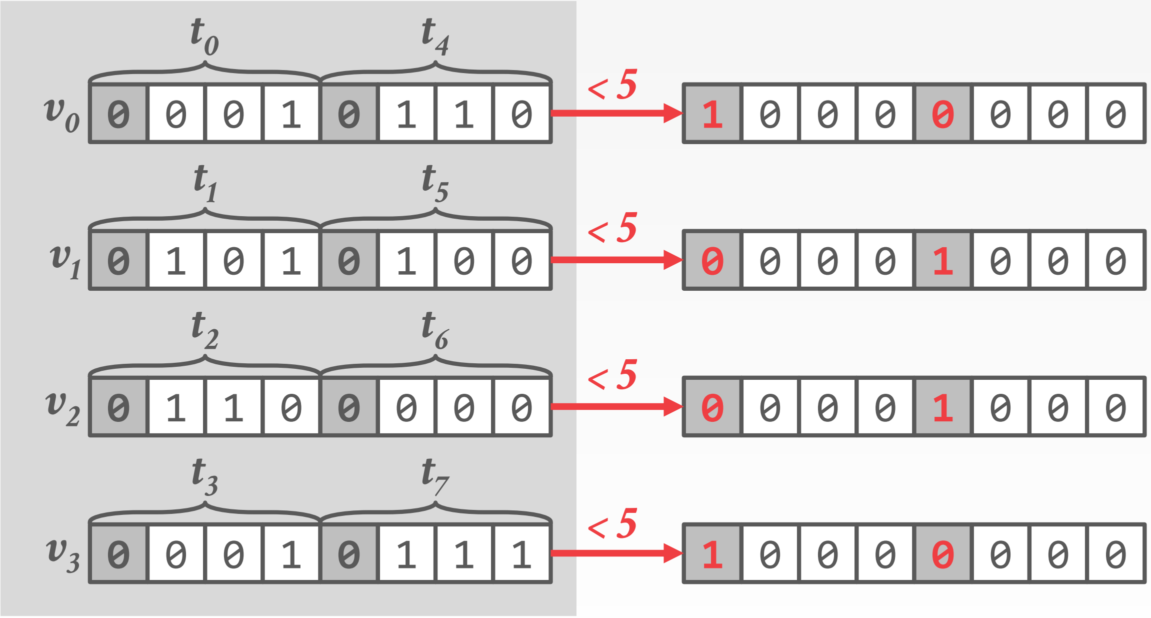
```
SELECT * FROM table
 WHERE val < 5
```



Source: Jignesh Patel

BITWEAVING/H – EXAMPLE

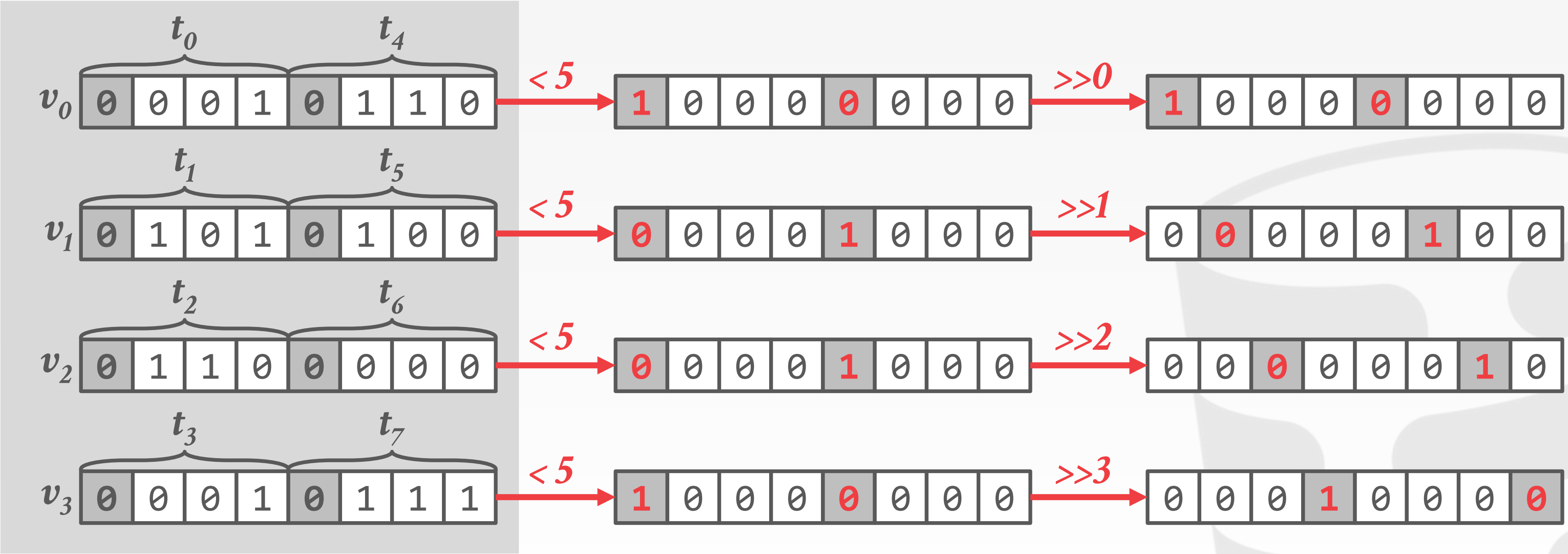
```
SELECT * FROM table
 WHERE val < 5
```



Source: Jignesh Patel

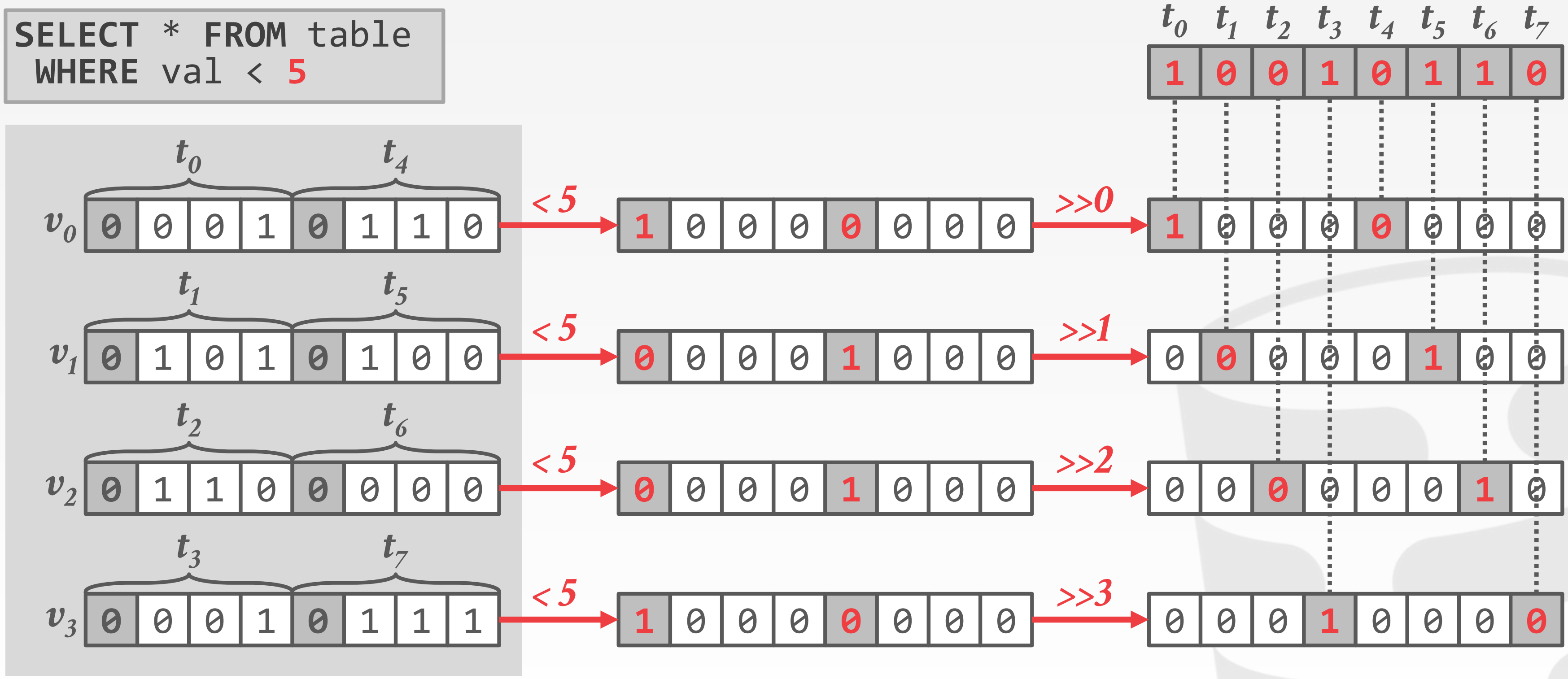
BITWEAVING/H – EXAMPLE

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



BITWEAVING/H – EXAMPLE

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



SELECTION VECTOR

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

Have to convert it into offsets / positions.

→ Approach #1: Iteration
→ Approach #2: Pre-compute Positions Table

SELECTION VECTOR

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

Have to convert it into offsets / positions.
→ Approach #1: Iteration
→ Approach #2: Pre-compute 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

SELECTION VECTOR

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

Have to convert it into offsets / positions.

→ Approach #1: Iteration

→ Approach #2: Pre-compute 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.

Have to convert it into offsets / positions.

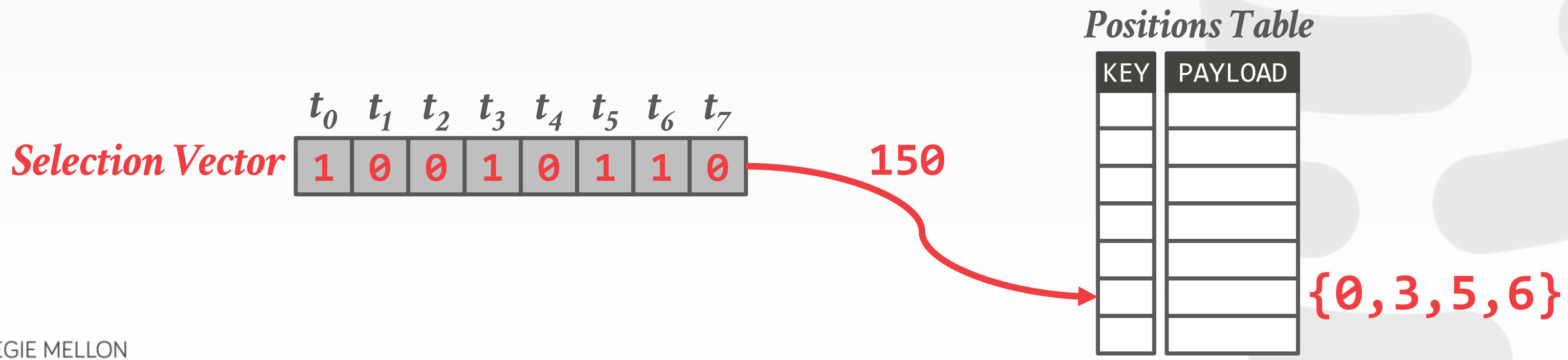
→ Approach #1: Iteration
→ Approach #2: Pre-compute Positions Table

	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

SELECTION VECTOR

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

Have to convert it into offsets / positions.
→ Approach #1: Iteration
→ Approach #2: Pre-compute Positions Table



VERTICAL STORAGE

Segment	Tuple			
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

VERTICAL STORAGE

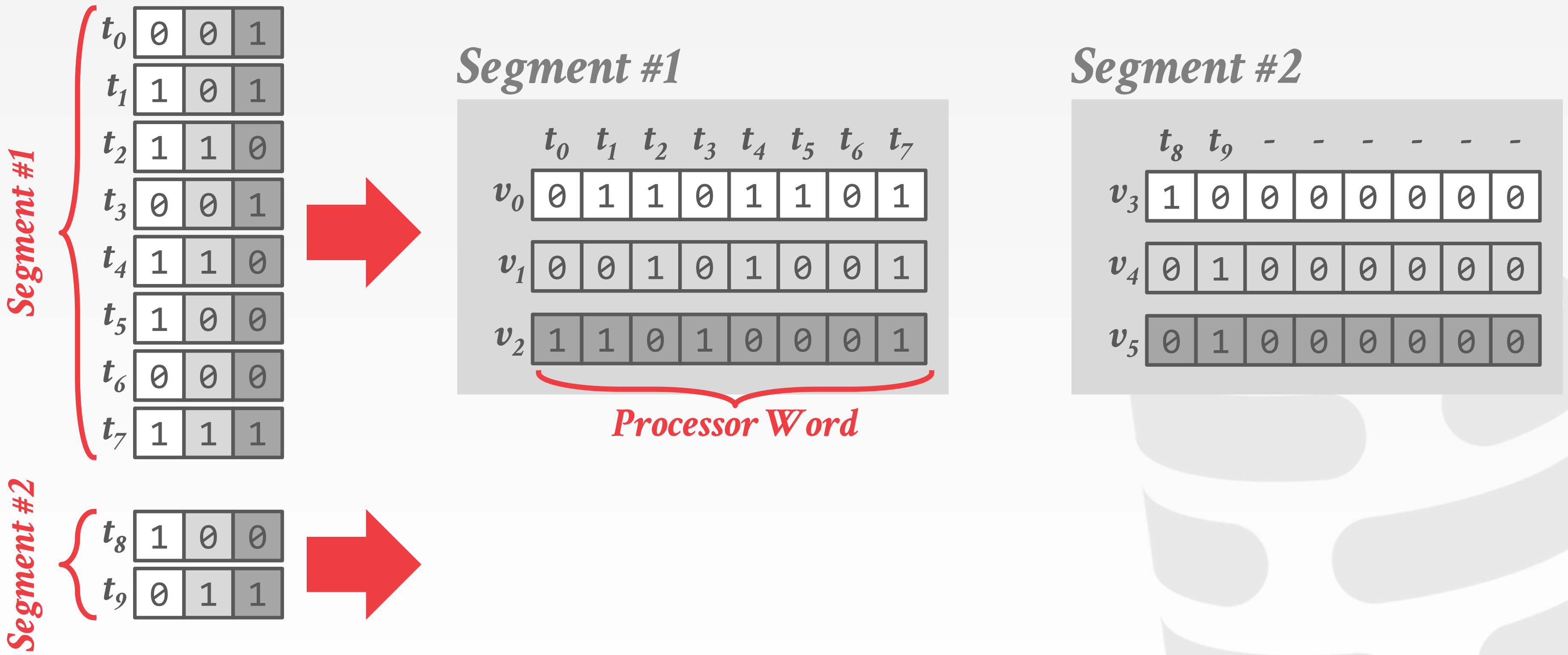
Segment #1

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

VERTICAL STORAGE



BITWEAVING/V – EXAMPLE

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

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

BITWEAVING/V – EXAMPLE

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

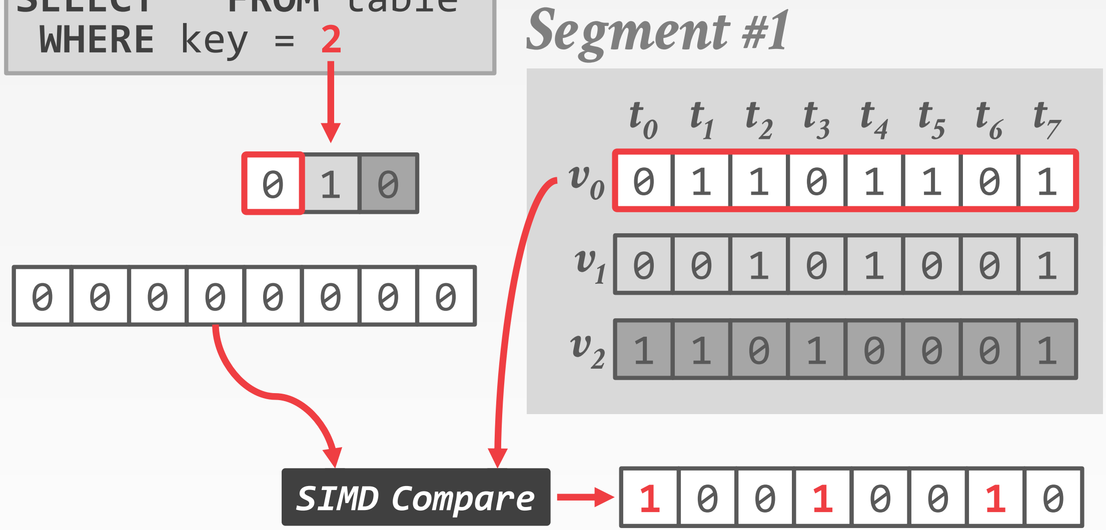
0 1 0

Segment #1

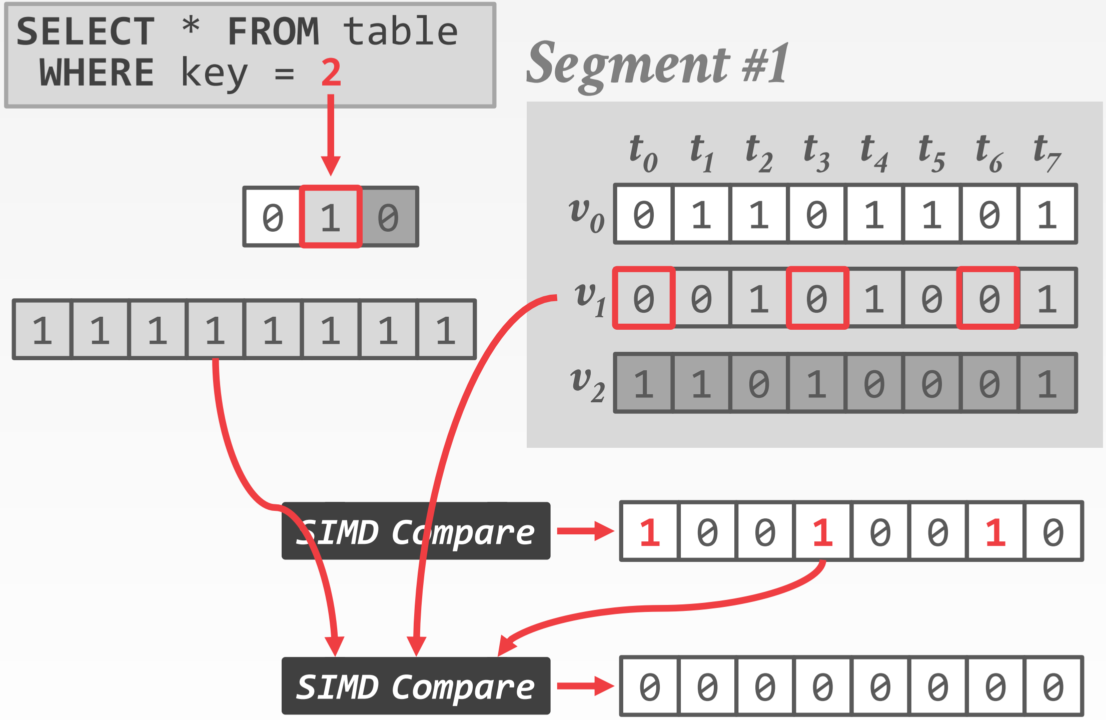
	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
v_0	0	1	1	0	1	1	0	1
v_1	0	0	1	0	1	0	0	1
v_2	1	1	0	1	0	0	0	1

BITWEAVING/V – EXAMPLE

```
SELECT * FROM table
 WHERE key = 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.

EVALUATION

Single-threaded execution of a single query derived from TPC-H benchmark.
→ Selectivity: 10%

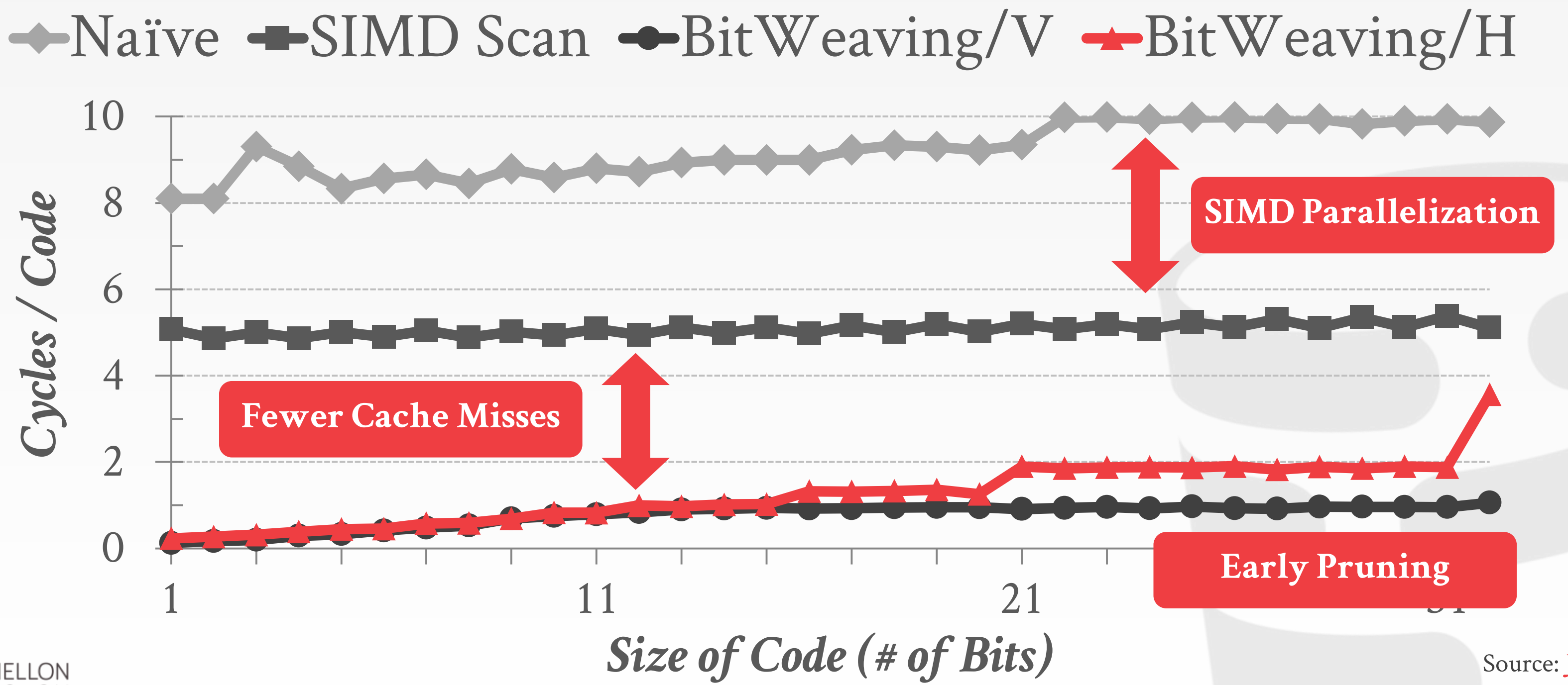
10GB TPC-H Database
→ 1 billion tuples
→ Uniform distribution

```
SELECT COUNT(*)
  FROM R
 WHERE R.a < C
```


Source: Jignesh Patel

EVALUATION

TPC-H Aggregation Query
Intel Xeon X5650 @ 2.66 GHz



Source: Jignesh Patel

PARTING THOUGHTS

Just like in query compilation, getting the best performance with vectorization requires the DBMS to store data in a way that is best for the CPU and not the best for humans' understanding.