COLUMNAR COMPRESSION

Rob Story
@oceankidbilly
work @simple doing data things on JVM

C-STORE

Abadi, PhD: *Query Execution in Column Oriented Databases*

REDSHIFT

Byte-Dict/ Text255/Text32k

```
["foo", "bar", "baz"] -> Dictionary

[0 1 1 2 0 2 0 1] -> Index Array

["foo" "bar" "bar" "baz"
 "foo" "baz" "foo" "bar"] -> Result
```

Delta Encoding

Original data value	Original size (bytes)	Difference (delta)
1	4	
5	4	4
50	4	45
200	4	150
185	4	-15
220	4	35
221	4	1

Compressed value	Compressed Size
1	1+4 (flag + actual value)
4	1
45	1
150	1+4 (flag + actual value)
-15	1
35	1
1	1

LZO

Mostly Encoding

Original value	Original INT or BIGINT size (bytes)	MOSTLY8 compressed size (bytes)	MOSTLY16 compressed size (bytes)	MOSTLY32 compressed size (bytes)
1	4	1	2	4
10	4	1	2	4
100	4	1	2	4
1000	4	Same as raw data size	2	4
10000	4		2	4
20000	4		2	4
40000	8		Same as raw data size	4
100000	8			4
2000000000	8			4

Runlength

Original data value	Original size (bytes)
Blue	4
Blue	4
Green	5
Green	5
Green	5
Blue	4
Yellow	6
Yellow	6
Yellow	6

Compressed value (token)	Compressed size (bytes)
{2,Blue}	5
	0
{3,Green}	6
	0
	0
{1,Blue}	5
{4,Yellow}	7
	0
	0

C-Store

Encodings

BitVector

```
{"foo" [0 0 1 0 1 0 1 1 1 0]
 "bar" [1 0 0 1 0 0 0 0 0 1]
 "baz" [0 1 0 0 0 1 0 0 0 0]}
```

Compressed Dictionaries

Cardinality: 32 possible values

Only need 2 bytes: 5 bits (1-32)

"foo" = 00000 "bar" = 00001

"baz"= 000010

x0000000000100010 -> "bar" "baz" "foo"

Keep *all* permutations of bits: 32^3

Scan reads 3 values/time

Frequency Partitioning (minimize variability in each page)

```
0,1   00, 10, 01      0      000, 010, 001, 100
[A B]  [C D C E]   [F F F F]      [G H I J]
```

Properties	**Iterator Access**	**Block Information**
isOneValue()	getNext()	getSize()
isValueSorted()	asArray()	getStartValue()
isPosContig()		getEndPosition()

Encoding Type	**Sorted?**	**1 value?**	**Pos. contig.?**
RLE	yes	yes	yes
Bit-string	yes	yes	no
Null Supp.	no/yes	no	yes
Lempel-Ziv	no/yes	no	yes
Dictionary	no/yes	no	yes
Uncompressed	no/yes	no	no/yes

Compression Aware Aggregates

```
{2,Blue}
{3,Green}
{1,Blue}
{4,Yellow}

sum(encoded.keys())
sum(encoded.keys()) / encoded.length
```

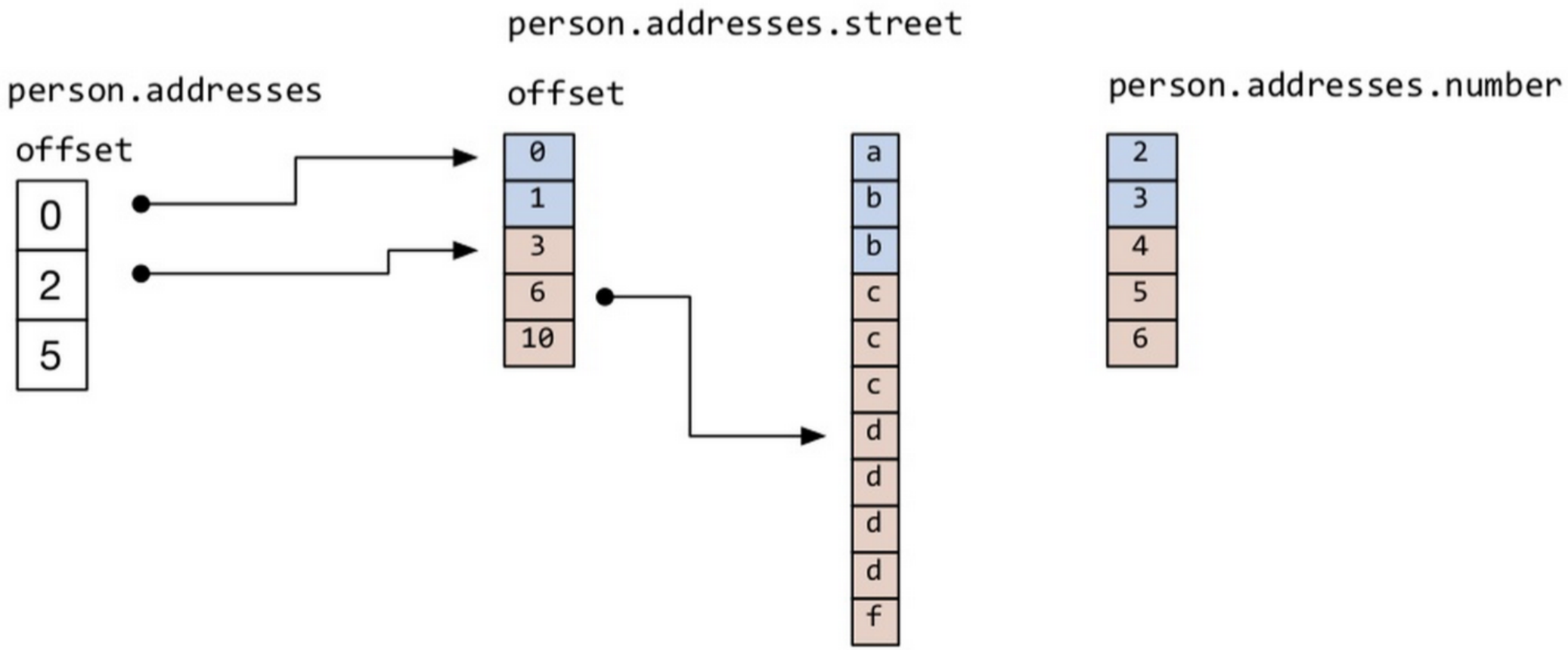
Parallel Predicate Evaluation

```
select *
from table
where col_1 = "foo"
and col 2 = "bar"
```

```
["foo", "bar", "baz"]
[0 1 1 2 0 2 0 1]

["foo" "bar"]
[0 1 0 0 1 0 1 0]

[1 0 0 0 1 0 1 0]
[0 1 0 0 1 0 1 0]
bitwise and...
[0 0 0 0 1 0 1 0]

"take" all the things
```

RLE! Frequency Partitioned!

Array<Struct> example



cloudera

