



Architecture of DB Systems 05 Compression Techniques

Matthias Boehm, Patrick Damme

Graz University of Technology, Austria Computer Science and Biomedical Engineering Institute of Interactive Systems and Data Science BMK endowed chair for Data Management











Announcements/Org

#1 Video Recording

- Link in TUbe & TeachCenter (lectures will be public)
- Optional attendance (independent of COVID)
- Hybrid, in-person but video-recorded lectures
 - HS i5 + Webex: https://tugraz.webex.com/meet/m.boehm





#2 COVID-19 Precautions (HS i5)

- Room capacity: 24/48 (green/yellow), 12/48 (orange/red)
- TC lecture registrations (limited capacity, contact tracing)

max **24/90**

#3 Exams

- Planned as oral exams (beginning of February)
- # submitted projects → change to written exam if necessary





Agenda

- Motivation and Terminology
- Compression Techniques
- Compressed Query Processing





Motivation and Terminology





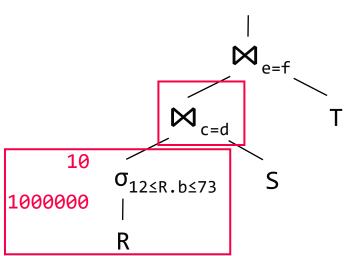
Recap: Access Methods and Physical Design

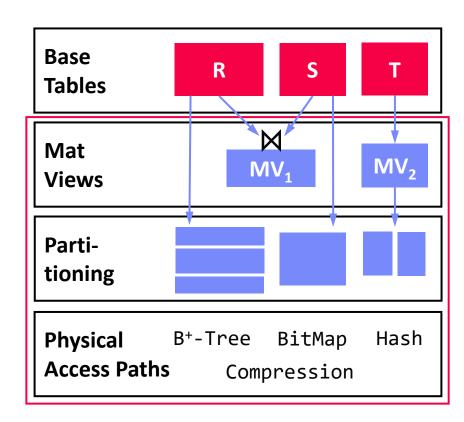
Performance Tuning via Physical Design

- Select physical data structures for relational schema and query workload
- #1: User-level, manual physical design by DBA (database administrator)
- #2: User/system-level automatic physical design via advisor tools

Example

```
SELECT * FROM R, S, T
 WHERE R.c = S.d AND S.e = T.f
    AND R.b BETWEEN 12 AND 73
```



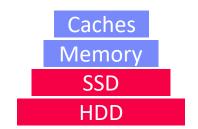




Motivation Storage Hierarchy

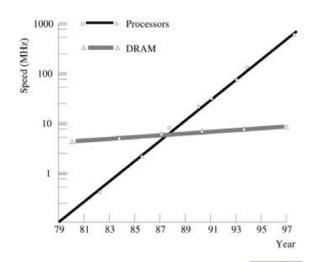
#1 Capacity

- Limited capacity of fast storage
- Keep larger datasets higher in storage hierarchy
- Avoid unnecessary I/O



#2 Bandwidth

- Memory Wall: increasing gap
 CPU vs Memory latency/bandwidth
- Reduce bandwidth requirements



[Stefan Manegold, Peter A. Boncz, Martin L. Kersten: Optimizing database architecture for the new bottleneck: memory access. **VLDB J. 9(3) 2000**]







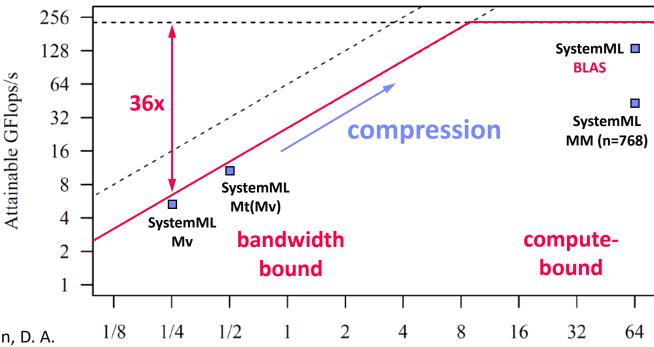
Excursus: Roofline Analysis

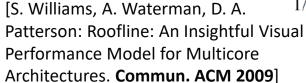
- Setup: 2x6 E5-2440 @2.4GHz-2.9GHz, DDR3 RAM @1.3GHz (ECC)
 - Max mem bandwidth (local): 2 sock x 3 chan x 8B x 1.3G trans/s → 2 x 32GB/s
 - Max mem bandwidth (QPI, full duplex) \rightarrow 2 x 12.8GB/s
 - Max floating point ops: 12 cores x 2*4dFP-units x $2.4GHz \rightarrow 2 \times 115.2GFlops/s$

Roofline Analysis

Off-chip memory traffic

Peak compute





Operational Intensity (Flops/Byte) (Experiments from 2017)

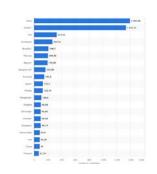


Motivation Data Characteristics

Skew

- Highly skewed value distributions (frequencies of distinct values)
- Small number of distinct items

China 1.4 India 1.3 USA 0.33 Germany 0.08 Austria 0.009



Correlation

- Correlation between tuple attributes
- Co-occurrences of attribute values

OrderDate < ReceiptDate (usually 2-3 days)

Lack of Tuple Order

- Relations are multi-sets of tuples (no ordering requirements)
- Flexibility for internal reorganization

[Vijayshankar Raman, Garret Swart: How to Wring a Table Dry: Entropy Compression of Relations and Querying of Compressed Relations. **VLDB 2006**]



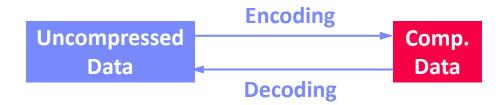




Compression Overview

Compression Codec

- Encoder
- Decoder

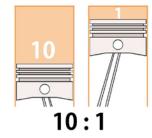


Lossless vs Lossy

- Lossless: guaranteed recovery of uncompressed data
- Lossy: moderate degradation / approximation
 - → Images, video, audio; ML training/scoring

Compression Ratio

- CR = Size-Uncompressed / Size-compressed
- Ineffective compression: CR < 1



Metrics

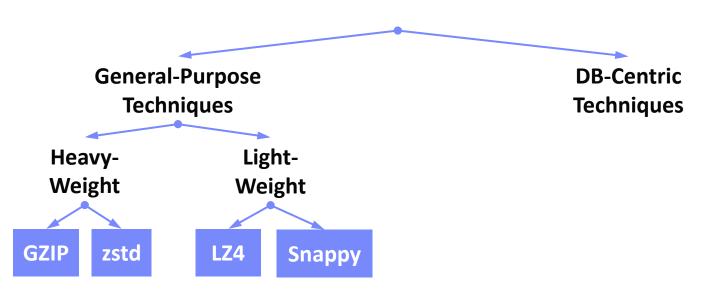
- Compression ratio vs encode/decode time vs encode/decode space
- Block-wise vs random access, operation performance, etc





Classification of Compression Techniques

Lossless Compression Schemes



Huffman + Lempel-Ziv





Excursus: General-purpose Compression

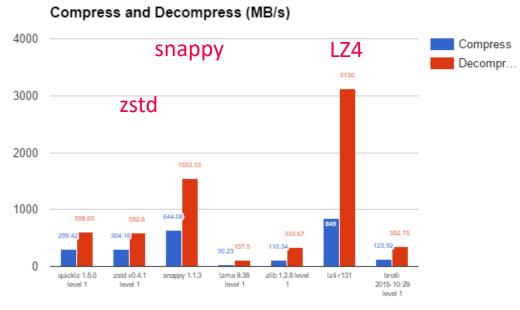
Compression/ Decompression

CR zstd: 5.24

CR snappy: 3.65

CR LZ4: 3.89

[https://web.archive.org/web/20200229 161007/https://www.percona.com/blog/ 2016/04/13/evaluating-databasecompression-methods-update/]



Compression Method

Example Apache Spark RDD Compression

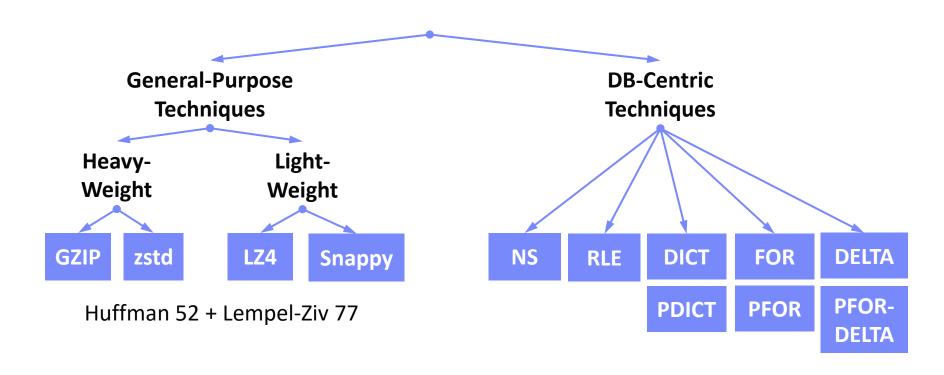
- org.apache.spark.io.LZ4CompressionCodec (default in 2.x, 3.x)
- org.apache.spark.io.SnappyCompressionCodec (default in 1.x)
- org.apache.spark.io.LZFCompressionCodec (default in 0.x)
- org.apache.spark.io.ZStdCompressionCodec





Classification of Compression Techniques, cont.

Lossless Compression Schemes



(all heavy-weight from a DB perspective)





Compression Techniques





42

Null Suppression (NS)

[Benjamin Schlegel, Rainer Gemulla, Wolfgang Lehner: Fast integer compression using SIMD instructions. **DaMoN 2010**]



Overview

 Compress integers by omitting leading zeros via variable-length codes

00000000 00000000 00000000 00101010

- Universal compression scheme w/o need for upper bound
- Byte-Aligned (Example)
 - Store mask of two bits to indicate leading zero bytes

42 **11** 00101010

• 2 bits + [1,4] bytes \rightarrow max CR (INT32) = 3.2

11 00000111

- Bit-Aligned (Example: Elias Gamma Encoding)
 - Store $N = \lfloor \log_2 x \rfloor$ zero bits followed by effective bits

42 00000 101010

■ $2 * [1,32] - 1 \text{ bits } \rightarrow \text{max CR (INT32)} = 32$

00 111

- Word-Aligned (Example: Simple-8b)
 - Pack a variable number of integers (max 2⁶⁰-1) into 64bit

60 data bits, 4 selector bits (16 classes: 60x1b, 30x2b, ..., 1x60b)







Null Suppression (NS), cont.

[Jeff Dean: Challenges in Building Large-Scale Information Retrieval Systems, Keynote **WSDM 2009**]



Varint (Variable-Length Integers)

[also Byte-Aligned]

- **Base 128 Varint** 0000001 |1111111||0||0000011 .|1111111||<mark>1</mark>|1111111||<mark>0</mark>|0000111 (continuation bits) 511 131071 **Prefix Varint** 1111111 | 00000111 00 000001 |<mark>10</mark>|11111 |11111111| 00000111 (2 bit #bytes) 511 131071 **Group Varint** |00000001||11111111||0000001 11111111 |11111111||00000001||00000011 131071
- Examples:
 - Google Protobuf messages, SQLite custom varint
- Zig-Zag Encoding
 - Map signed integers to unsigned integers to have small varint byte length





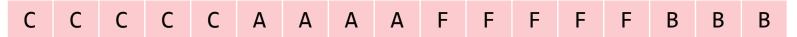
Run-Length Encoding (RLE)

Overview

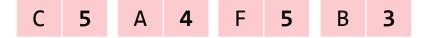
- Compress sequences of equal values via runs of (value[,start],run-length)
- Redundant 'start' allows parallelization / unordered storage
- Applicable to arbitrary data types (defined equals())

Example

Uncompressed



Compressed



- Different physical encodings for values and lengths:
- E.g., split runs w/ length $\geq 2^{16}$ to fit into fixed 2 byte



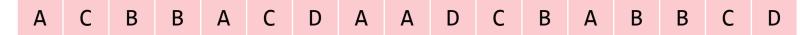


Dictionary Encoding (DICT)

- Overview
 - Build dictionary of distinct items and encode values as dictionary positions
 - Applicable to arbitrary data types → integer codes

Example

Uncompressed



Compressed

D

3



1 C • Explicit or implicit (position) codes

Fixed bit width: log₂ |Dict|

Different ordering of dictionary (alphanumeric, frequency)





Dictionary Encoding (DICT), cont.

Order-preserving Dictionaries

Create sorted dictionary where order(codes) = order(values) [Carsten Binnig, Stefan Hildenbrand, Franz Färber: Dictionary-based order-preserving string compression for main memory column stores. **SIGMOD 2009**]



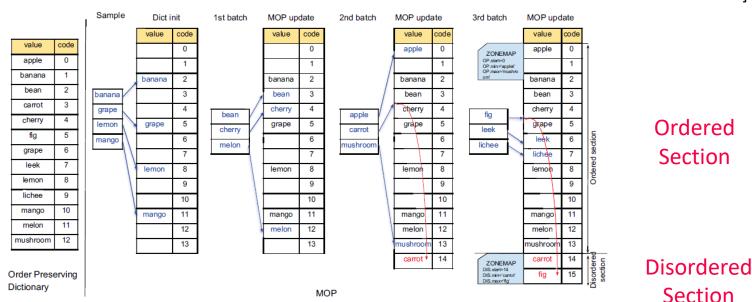
- Support for updates via sparse code assignment (e.g., 10, 20, 30)
- CS-Array-Trie / CS-Prefix-Tree as encode/decode index w/ shared leafs

Mostly Order-preserving Dictionaries

Ordered and disordered dictionary sections

[Chunwei Liu et al: Mostly Order Preserving Dictionaries. ICDE 2019]





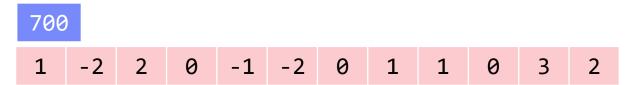


Frame of Reference Encoding (FOR)

- Overview
 - Compress values by storing delta (difference) to reference value
 - Mostly integer types → smaller integer domain
- Example
 - Uncompressed

701	698	702	700	699	698	700	701	701	700	703	702

Compressed



Cannot handle trends very well





Delta Encoding (DELTA)

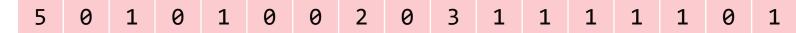
Overview

- Compress values by storing delta (difference) to previous value
- Mostly integer types (good when sorted) → smaller integer domain
- Dedicated techniques for differences of file contents (diff/git)

Example

Uncompressed

Compressed



- Delta
- Double Delta (differences of differences)

Can create RLE opportunities for linear trend





Patched Compression Methods (PFOR)

- Patched Frame of Reference (PFOR)
 - Store positive offsets to reference value

[Marcin Zukowski, Sándor Héman, Niels Nes, Peter A. Boncz: Super-Scalar RAM-CPU Cache Compression. **ICDE 2006**]

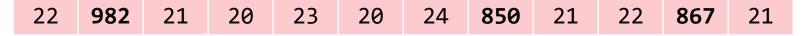


- Exceptions in uncompressed form (accessible via entry points and offsets to next exception)
- Branchless two-pass decoding

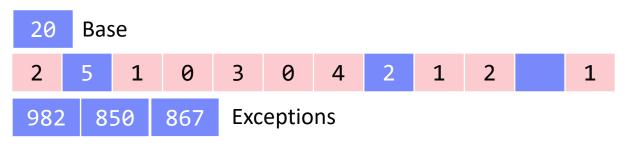
Example

Uncompressed

Outliers would destroy fixed-width codes



Compressed







Patched Compression Methods (Others)

PFOR-DELTA

 Apply cascade of DELTA – PFOR (PFOR on differences) [Marcin Zukowski, Sándor Héman, Niels Nes, Peter A. Boncz: Super-Scalar RAM-CPU Cache Compression. **ICDE 2006**]



Handling of exceptions to handle large differences of subsequent values

Patched Dictionary Compression (PDICT)

- Dictionary encoding, where only frequent values are encoded
- Exceptions for infrequent values, previous/new dictionary per block
- Reduces dictionary size

Removes long tail of infrequent distinct items from dictionary





Excursus: SIMD Implementation

- How to reduce overhead of (de)compression?
 - Vectorization of (de)compression by SIMD instructions
 - (De)compress several data elements at once
 - Main driver of research on database-centric lightweight compression for years
- SIMD-BP128

[Daniel Lemire, Leonid Boytsov: Decoding billions of integers per second through vectorization.

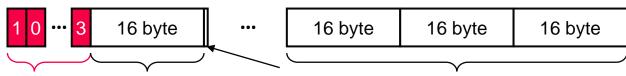
Softw. Pract. Exp. 45(1) (2015)]



 Targets compression of 32-bit ints using Intel's SSE with 128-bit vectors (16 byte)

Sub-divide integer sequence into blocks of 128 integers

- Determine maximum bit width in a block by bitwise OR
- Pack all integers in a block using that number of bits
- Dedicated vectorized (un)packing routine for each bit width
- Store bit width in [0, 32] as one byte, combine 16 of those in memory



bit widths: 16x 1 byte block 0 (1 bit/int) block 1 (0 bit/int)

block 15 (3 bit/int)





Excursus: SIMD Implementation, cont.

Vectorized (un)packing

[Daniel Lemire, Leonid Boytsov: Decoding billions of integers per second through vectorization. **Softw. Pract. Exp.** 45(1) (2015)]



Uncompressed (32 bit/int) Compressed (9 bit/int) Compressed (9 bit/int) scalar codec, SIMD codec, v0 v1 value-by-value vector-by-vector ...v12 **8**V v4 v5 v4 v3.... **⋖** v5 ...v13 v9 v3 1. Load 1. Load ...v10 v8 v7... ...v14 v10 v6 v4 v13 v12 v11 ...v14 v10... 2. Bit I-shift 2. Bit I-shift ...v15 v11 v5 v15 v14... v12... 3. Bit OR 3. Bit OR **v6** v13... ٧7 v14... v8 1. Bit r-shift 1. Bit r-shift v15... v9 2. Bit AND 2. Bit AND 3. Store 3. Store v11 **Horizontal layout** k-way vertical layout v13 Codes of consecutive values Codes of consecutive values v15 in consecutive bytes in consecutive words Natural for scalar codecs Natural for SIMD codecs 32 bit word Common #bits for k values



Adopted by many vectorized

compression algorithms



Comparative Evaluation

Experimental Survey

- Different data characteristics
- Compression methods:

DELTA, RLE, FOR, RLE, DICT, SIMD-BP128, SIMD-FastPFOR, 4-Wise NS, 4-Gamma, Masked VByte, Simple-8b, SIMD-GroupSimple

Cascades of compression methods

[Patrick Damme, Dirk Habich, Juliana Hildebrandt, Wolfgang Lehner: Lightweight Data Compression Algorithms: An Experimental Survey (Experiments and Analyses). **EDBT 2017**]





"[...] there is no single-best lightweight integer compression algorithm. The compression rates and performances of all algorithms differ significantly, depending on the data characteristics and the employed SIMD extension."

Towards a Cost-based Selection

- Logical and physical level
- Cost estimation functions

[Patrick Damme, Annett Ungethüm, Juliana Hildebrandt, Dirk Habich, Wolfgang Lehner: From a Comprehensive Experimental Survey to a Cost-based Selection Strategy for Lightweight Integer Compression Algorithms.

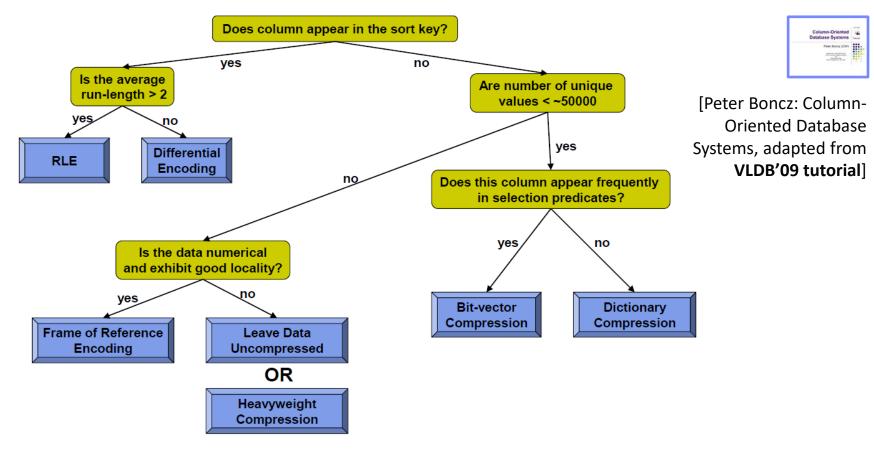


ACM Trans. Database Syst. 44(3) 2019]





Selecting Compression Methods



Inspired by C-Store Compression Paper [Daniel J. Abadi, Samuel Madden, Miguel Ferreira: Integrating compression and execution in column-oriented database systems. **SIGMOD 2006**]







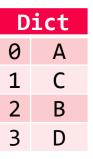
Compressed Query Processing



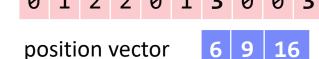


Selection Predicates

- Equivalence Predicates σ_{attr='D'}(R)
 - DICT: code lookup



 $D \rightarrow 3$



RLE: return RLE runs

C 5 A 4	D 5	F	3
---------	-----	---	---

- Range Predicates σ_{3<a<7} (R)
 - #1 sort the dictionary by value (insert tradeoff)
 - #2 expand small integer domains + dictionary lookup (e.g., $\sigma_{a=4 \text{ V a=5 V a=6}}$ (R))
 - #3 decompress otherwise





Selection Predicates, cont.

Order Preserving Dictionaries

- Direct support for range predicates on encoded data
- Support for LIKE predicates (suffix)

[Carsten Binnig, Stefan Hildenbrand, Franz Färber: Dictionary-based order-preserving string compression for main memory column stores. **SIGMOD 2009**]



String-dictionary (order preserving)

value	code
Whole Milk - Gallon	32000
Whole Milk - Quart	32100

Product column (encoded)

rid	p_name
1	32000
499	32100

Query (original):

Select SUM(o_total), p_name
 From Sales, Products
Where p_name='Whole Milk*'
 Group by p_name

value	code
Whole Milk - Gallon	32000
Whole Milk - Half Gallon	32050
Whole Milk - Quart	32100

rid	p_name
499	32100
500	32000
999	32050

Query (rewritten):

Select SUM(o_total), p_name
From Sales, Products
Where p_name ≥ 32000
And p_name ≤ 32100
Group by p_name





Selection on Bit-packed Data

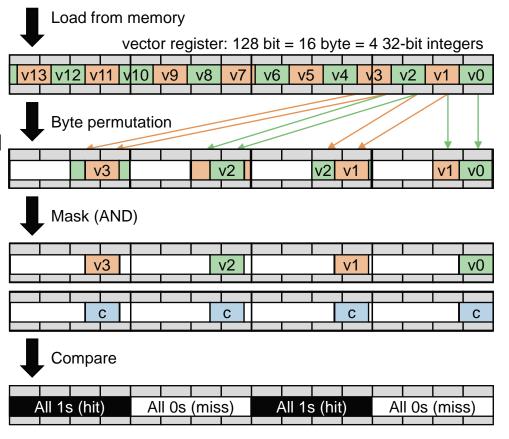
[Thomas Willhalm, Nicolae Popovici, Yazan Boshmaf, Hasso Plattner, Alexander Zeier, Jan Schaffner:



SIMD-Scan: Ultra Fast in-Memory Table **SIMD-Scan (2009)** Scan using on-Chip Vector Processing

Units. **PVLDB 2(1) (2009)**]

- - Horizontal bit-packing
 - Point and range predicates
 - SIMD processing (SSE)
 - 32-bit comparisons
 - No full decompression, predicate's constant is shifted accordingly in the beginning
 - Sub-optimal bit-level parallelism
- BitWeaving (2013)
 - Bit-parallel methods
 - Early pruning
- ByteSlice (2015)
 - More SIMD-friendly







Grouping and Aggregations

- Basic Hash Aggregates
 - Grouping directly with compressed codes
 - DICT, FOR, RLE, etc

Dict						
0	Α					
1	C					
2	В					
3	D					

Has	h Table
0	Agg A
3	Agg D
1	Agg C
2	Agg B

0	1	2	2	0	1	3	0	0	3	1	2	0	2	2	1	3



- RLE sum → agg += run-length*run-value
- RLE min \rightarrow agg = min(agg, run-value)
- FOR sum → for all codes: agg += code; agg += |codes| * base-value





Joins

Overview Compressed Joins

- (Equi-)Joins directly over compressed data
- Beware: binary operation
 → encodings need to match (global code)
- Recoding of one of the inputs if necessary (e.g., DB2 BLU recode inner)

		RID=S	SID	
R	RID		SID	S
	9		7	
	1		3	
	7		1	
i	nner	M	9	
 al -	innan		7	
code sma	inner ller)		oute	er

Encoding-Specific Aggregation

- One input RLE: decompress other and output RLE encoded data
- One input bitvector: decompress other and output RLE encoded data (obtained from bitvector)





Abstractions for Simpler Code

Motivation

- Code complexity for combinations of encoding schemes
- Affects all operators → maintenance operators/compression schemes
- Compressed Block Properties

[Daniel J. Abadi, Samuel Madden, Miguel Ferreira: Integrating compression and execution in column-



- isOneValue(): block contains just oriented database systems. **SIGMOD 2006**] one value and many positions for that value
- isValueSorted(): all values of the block are sorted
- isPosContig(): block contains consecutive subset of column
- Iterator Access:
 getNext(), asArray()
- Block Information: getSize(), getStartValue(), 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





Abstractions for Simpler Code, cont.

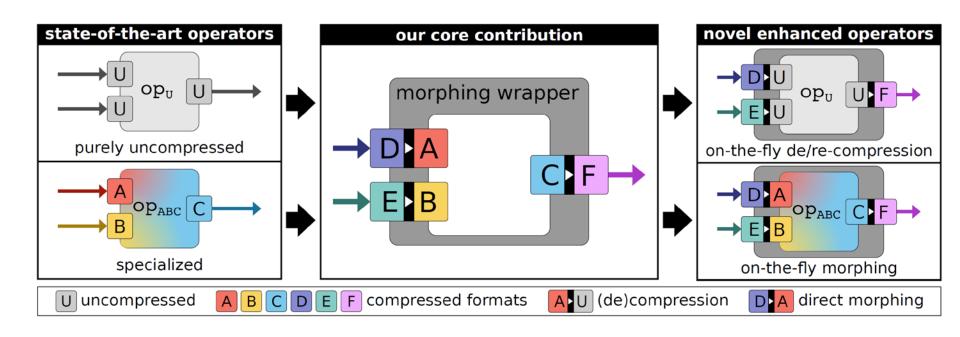
Motivation

 Improve query performance by (re)compressing intermediates [Patrick Damme, Annett Ungethüm, Johannes Pietrzyk, Alexander Krause, Dirk Habich, Wolfgang Lehner: MorphStore: Analytical Query Engine with a Holistic Compression-Enabled Processing Model.

PVLDB 13(11) 2020]



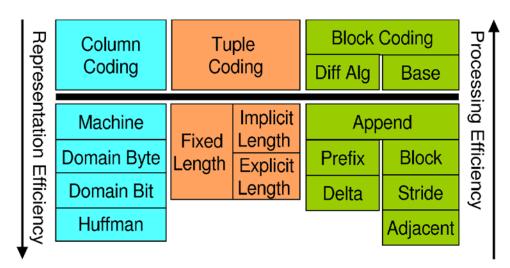
Change from one compressed format to another







Data Layout – Compression Granularity



[Allison L. Holloway, Vijayshankar Raman, Garret Swart, David J. DeWitt: How to barter bits for chronons: compression and bandwidth trade offs for database scans. **SIGMOD 2007**]



"All the results have shown that the Huffman coded and delta coded formats compress better but normally take more CPU time. [...] When I/O and memory subsystem times are also included in the decision, the format to choose becomes less clear-cut. If a physical format optimizer or system administrator had this information and a fast scan generator, they could make a more informed choice as to the best way to store the data."

Column Coding

Select encoding for individual attributes (column values) – tradeoffs

Tuple Coding

Combine column codes into tuple codes (fixed, variable)

03 Buffer Pool Management

Block Coding

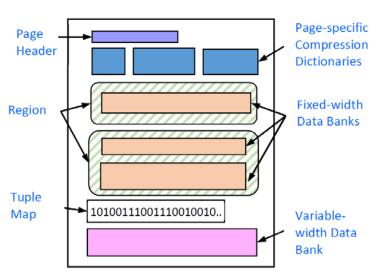
Compress a sequence of tuples into a compressed block (concat, diff)



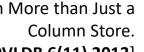


Data Layout – Example Block Layouts

DB2 BLU



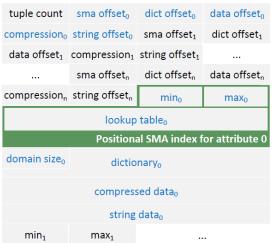
[Vijayshankar Raman et al: DB2 with BLU Acceleration: So Much More than Just a



PVLDB 6(11) 2013]



03 Buffer Pool **Management** 04 Index Structures and **Partitioning 07 Query Compilation** and Parallelization



[Harald Lang: Data Blocks: Hybrid OLTP and OLAP on Compressed Storage using both Vectorization and Compilation. SIGMOD 2016]







DB-Compression Beyond Relational Databases

Information Retrieval

- (web) search engines
- Inverted index, postings lists (sorted lists of document ids)

Time Series

- Internet of Things, sensor networks, server/application metrics, etc.
- Sequences of data points (measurement + time)

Machine Learning

- Various application fields
- Matrices/tensors of various characteristics

Graph Databases

- Social networks, road networks, proteins, etc.
- Graphs represented as adjacency lists, matrices

Also benefit from the compression techniques discussed today







Summary and Q&A

- Motivation and Terminology
- Compression Techniques
- Compressed Query Processing
- Next Lectures (Part B)
 - 06 Query Processing (operators, execution models) [Nov 10]
 - 07 Query Compilation and Parallelization [Nov 17]
 - 08 Query Optimization (rewrites, costs, join ordering) [Nov 24]
 - 10 Adaptive Query Processing [Dec 01]

