Architecture of DB Systems
05 Compression Techniques

Matthias Boehm

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

SCIENCE
PASSION
TECHNOLOGY

Last update: Nov 04, 2020

Announcements/Org

- **#1 Video Recording**
  - Link in **TeachCenter** & **TUbe** (lectures will be public)
  - Optional attendance (independent of COVID)

- **#2 COVID-19 Restrictions** (HS i5)
  - Corona Traffic Light: **Orange + Lockdown**
  - Max 25% room capacity (TC registrations)
  - Temporarily webex lectures and recording

max 18/74

Agenda

- **Motivation and Terminology**
- **Compression Techniques**
- **Compressed Query Processing**
- **Time Series Compression**

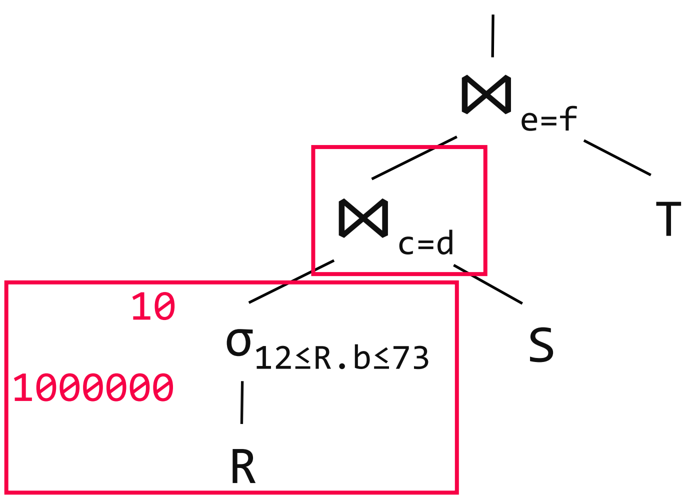
Motivation and Terminology

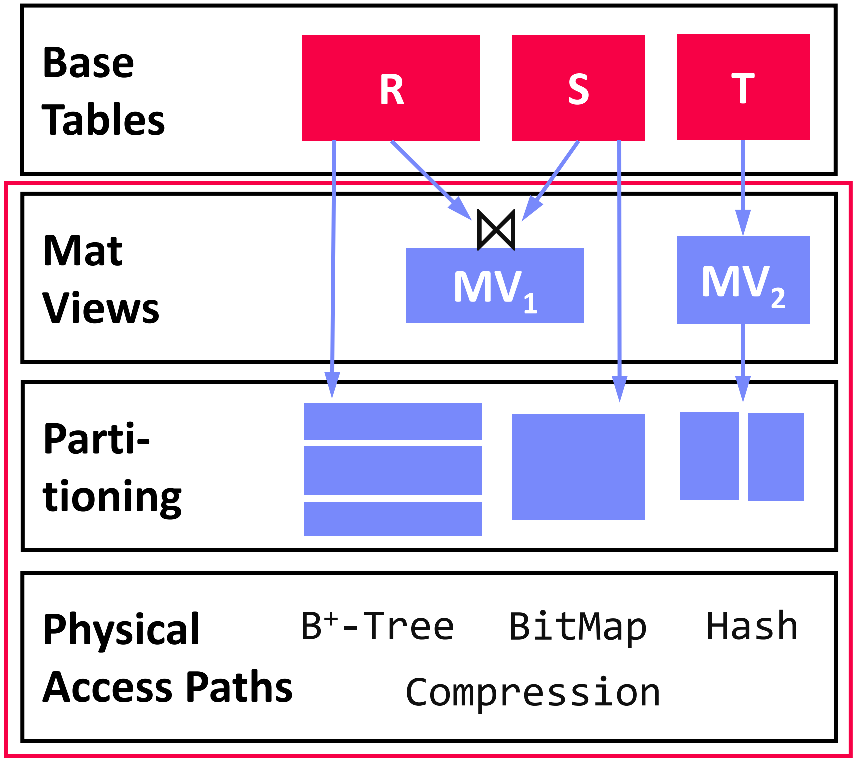
Recap: Access Methods and Physical Design

- **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**

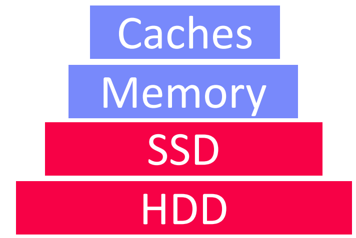
```
SELECT * FROM R, S, T
  WHERE R.c = S.d AND S.e = T.f
    AND R.b BETWEEN 12 AND 73
```



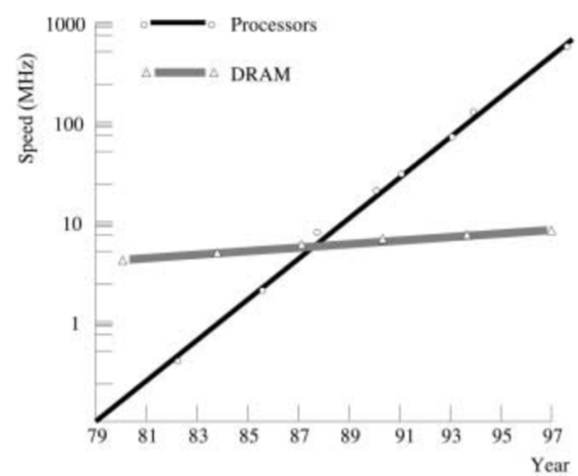


Motivation Storage Hierarchy

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



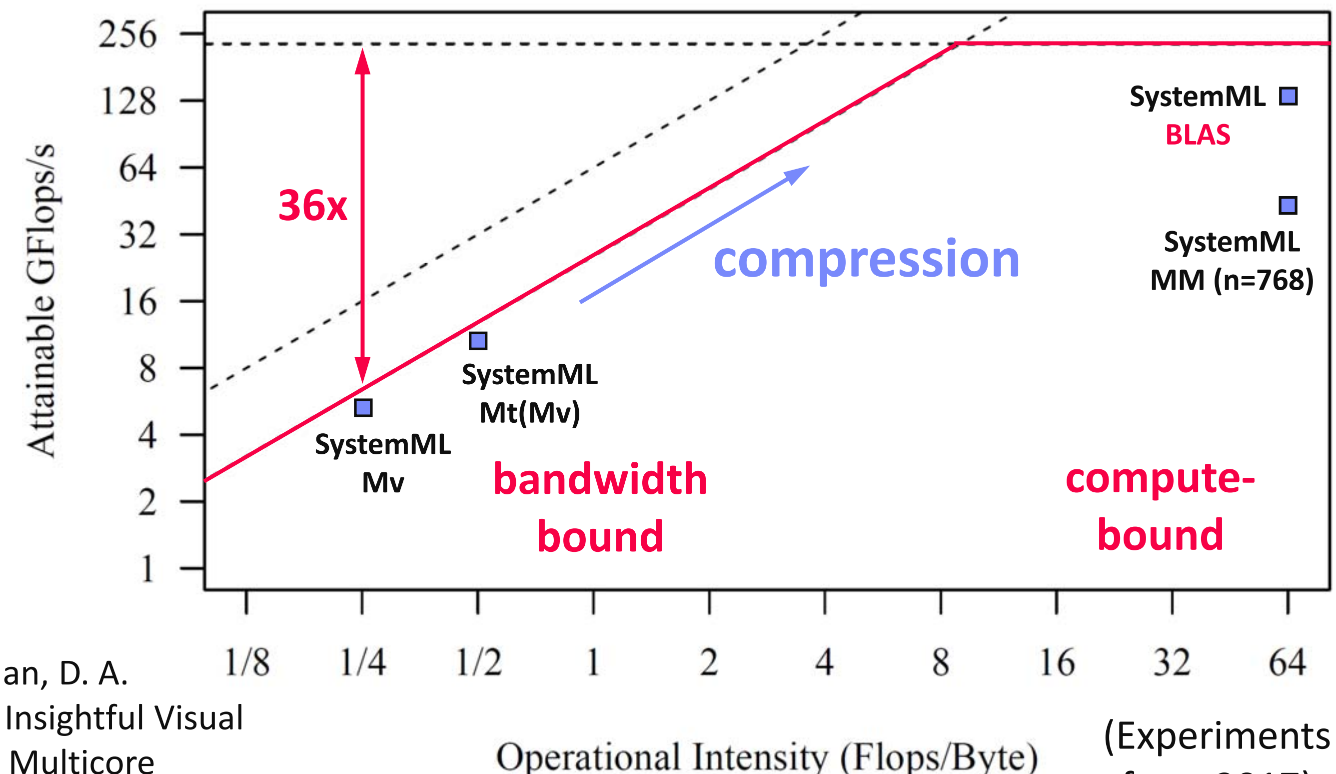
- **#2 Bandwidth**
  - **Memory Wall:** increasing gap
    CPU vs Memory latency/bandwidth
  - Reduce bandwidth requirements



[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) → **2 x 12.8GB/s**
  - Max floating point ops: 12 cores x 2*4dFP-units x 2.4GHz → **2 x 115.2GFlops/s**
- **Roofline Analysis**
  - Off-chip memory traffic
  - Peak compute



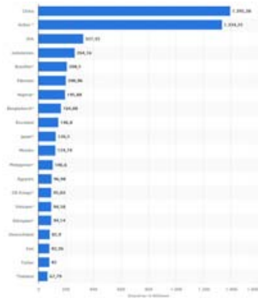
(Experiments from 2017)

[S. Williams, A. Waterman, D. A. Patterson: Roofline: An Insightful Visual Performance Model for Multicore Architectures. **Commun. ACM 2009**]

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

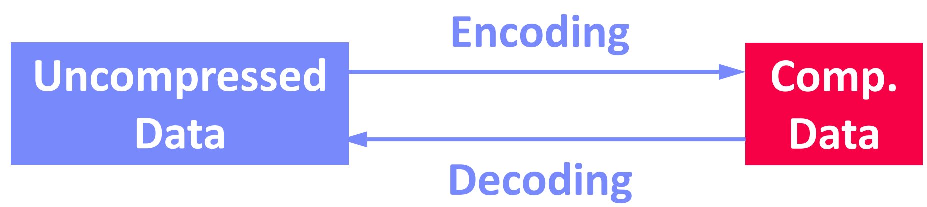
`OrderDate < ReceiptDate`
(usually 2-3 days)

- **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**]

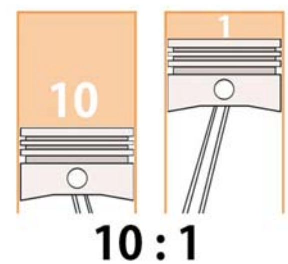
Compression Overview

- **Compression Codec**
  - Encoder
  - Decoder



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

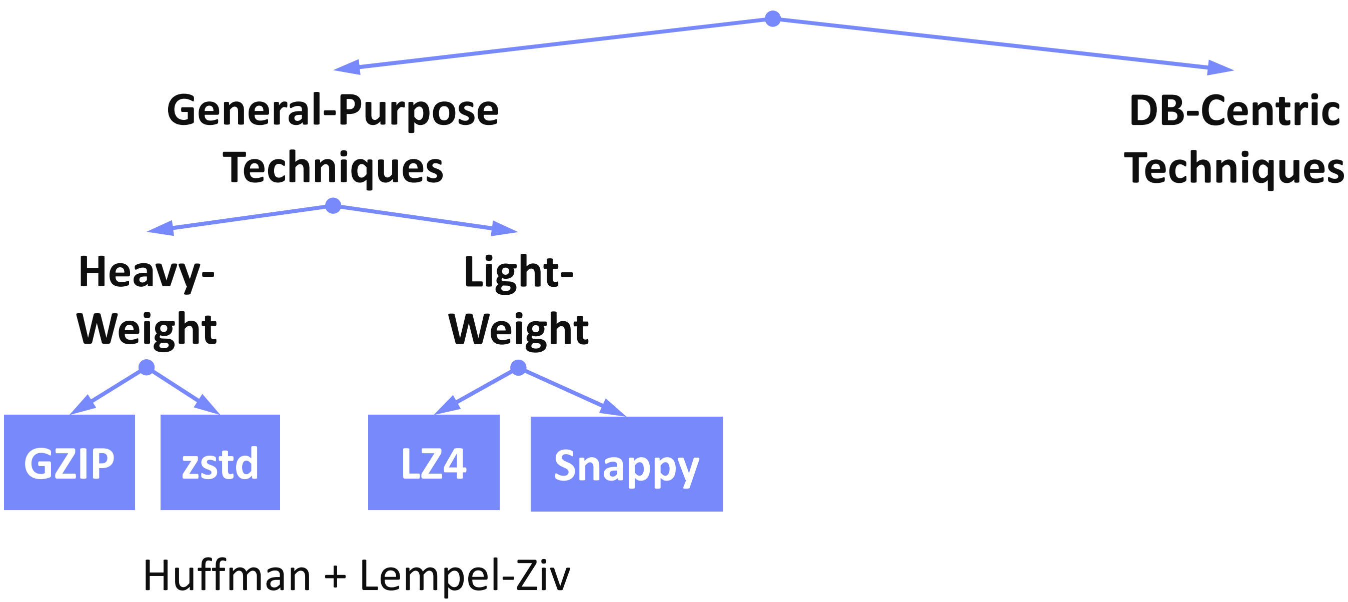
- **Compression Ratio**
  - CR = Size-Uncompressed / Size-compressed
  - Ineffective compression: CR < 1



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

Classification of Compression Techniques

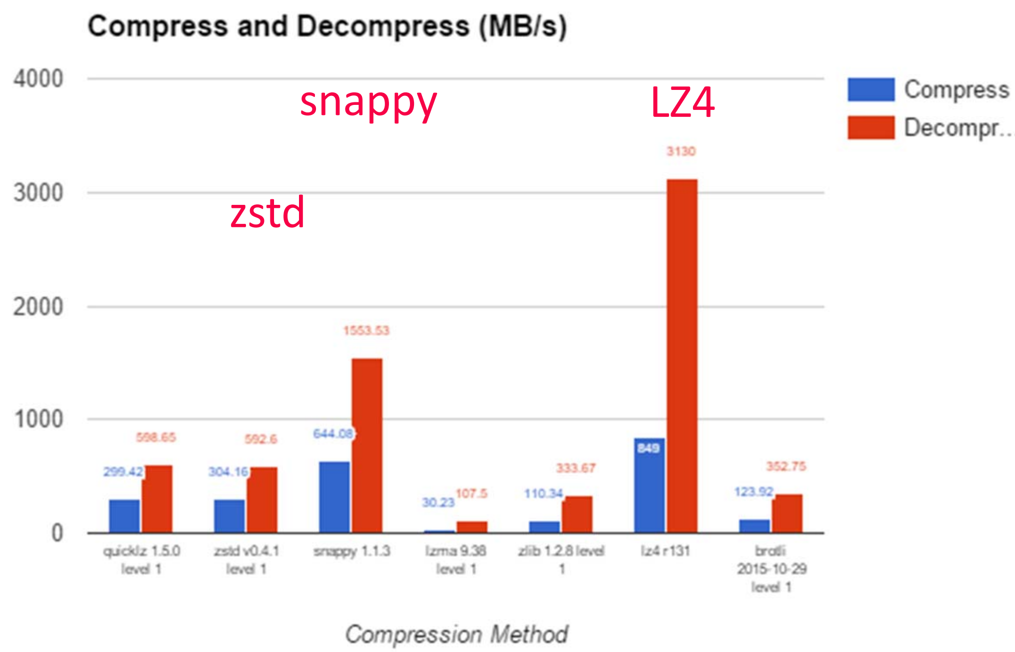
- **Lossless Compression Schemes**



Excursus: General-purpose Compression

- **Compression/ Decompression**
  - CR zstd: 5.24
  - CR snappy: 3.65
  - CR LZ4: 3.89

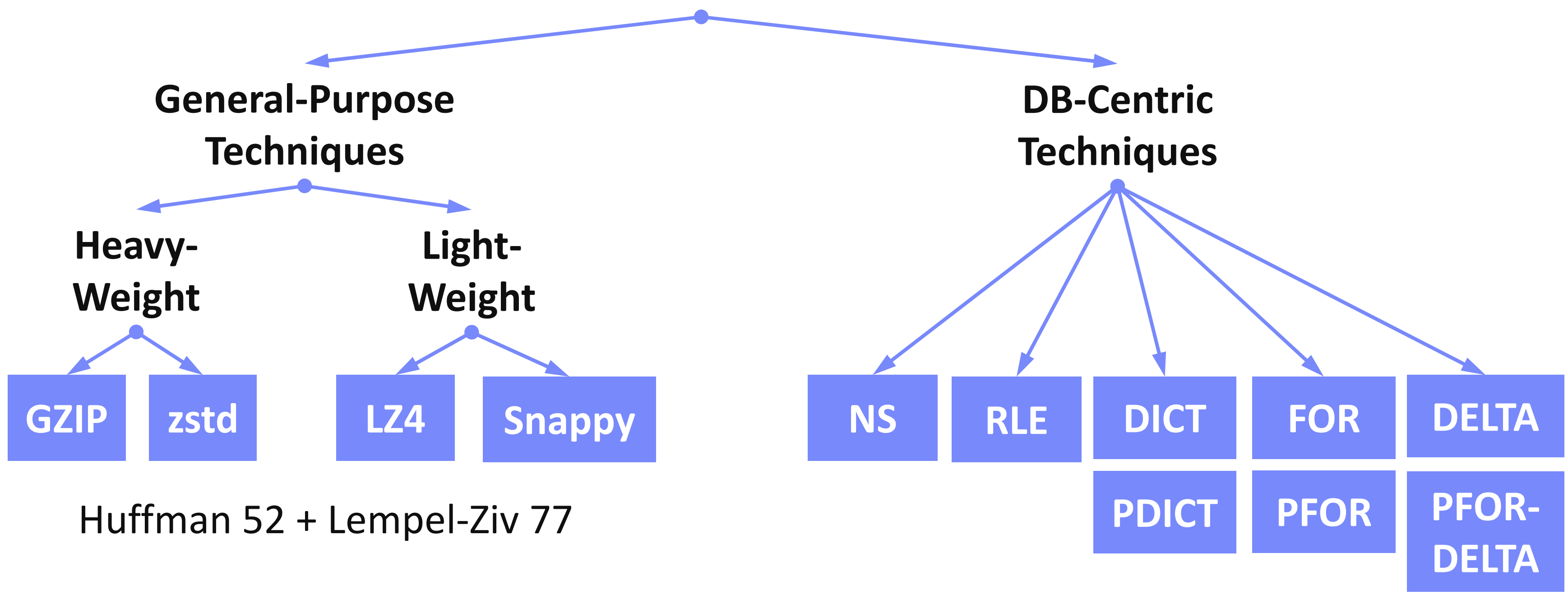
[https://web.archive.org/web/20200229161007/https://www.percona.com/blog/2016/04/13/evaluating-database-compression-methods-update/]



- **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

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
  - Universal compression scheme w/o need for upper bound

42
`00000000 | 00000000 | 00000000 | 00101010`

- **Byte-Aligned**
  - Store mask of two bits to indicate leading zero bytes
  - 2 bits + [1,4] bytes → max CR (INT32) = 3.2

42 `11 | 00101010`
7 `11 | 00000111`

- **Bit-Aligned** (Elias Gamma Encoding)
  - Store $\mathrm{N} = \lfloor \log_2 x \rfloor$ zero bits followed by effective bits
  - 2 * [1,32] -1 bits → max CR (INT32) = 32

42 `00000 | 101010`
7 `00 | 111`

- **Word-Aligned** (Simple-8b)
  - Pack a variable number of integers (max 2^{60}) into 64bit
  - 60 data bits, 4 selector bits (16 classes: 60x1b, 30x2b, 20x3b, 15x4b, 12x5b, …)

Null Suppression (NS), cont.

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

- **Varint (Variable-Length Integers)**
  - Base 128 Varint (continuation bits)

    `0|0000001` → 1
    `1|1111111` `0|0000011` → 511
    `1|1111111` `1|1111111` `0|0000111` → 131071

  - Prefix Varint (2 bit #bytes)

    `00|000001` → 1
    `01|111111` `00000111` → 511
    `10|111111` `11111111` `00000111` → 131071

  - Group Varint

    `00|01|10|00` `00000001` `11111111` `00000001`
    `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

C	C	C	C	C	A	A	A	A	F	F	F	F	F	B	B	B

  - **Compressed**

C	**5**	A	**4**	F	**5**	B	**3**

    - Different physical encodings for values and lengths:
    - E.g., split runs w/ length ≥ 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

A	C	B	B	A	C	D	A	A	D	C	B	A	B	B	C	D

  - **Compressed**

Dict	
0	A
1	C
2	B
3	D

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

- Explicit or implicit (position) codes
- Fixed bit width: $\log_2 |\text{Dict}|$
- Different ordering of dictionary (alphanumeric, frequency)

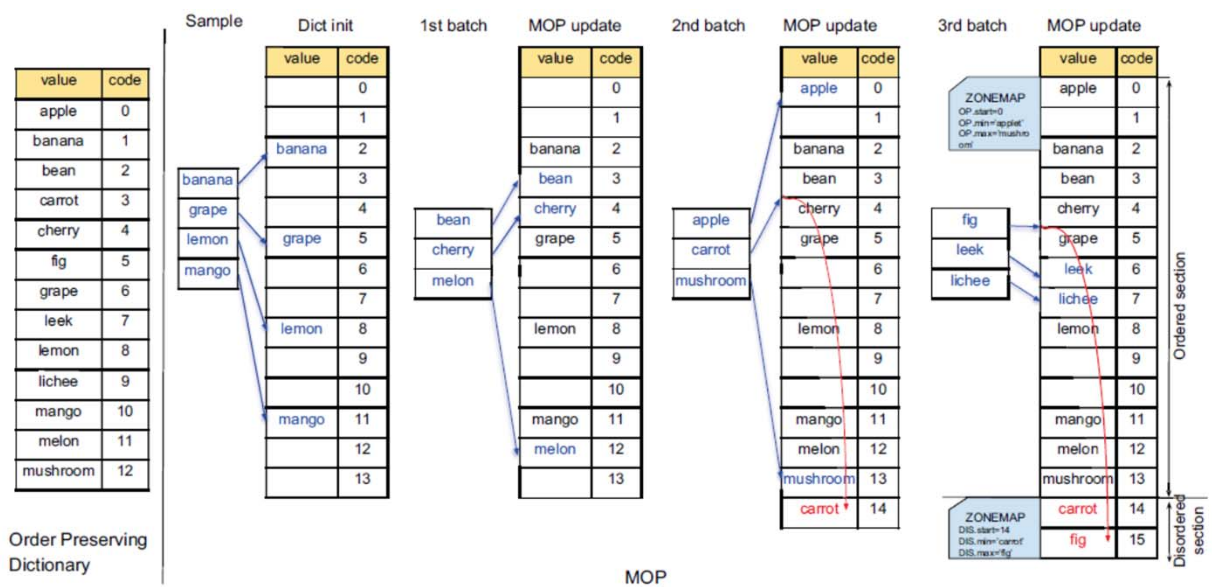
Dictionary Encoding (DICT), cont.

- **Order-preserving Dictionaries**
  - Create sorted dictionary where `order(codes) = order(values)`
  - 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

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

- **Mostly Order-preserving Dictionaries**
  - Ordered and disordered dictionary sections

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



Ordered Section

Disordered Section

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**

700

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

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

5	5	6	6	7	7	7	9	9	12	13	14	15	16	17	17	18

  - **Compressed**

5	0	1	0	1	0	0	2	0	3	1	1	1	1	1	0	1

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

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

- **Example**
  - Uncompressed

Outliers would destroy fixed-width codes

22	**982**	21	20	23	20	24	**850**	21	22	**867**	21

  - **Compressed**

20	Base

2	5	1	0	3	0	4	2	1	2		1

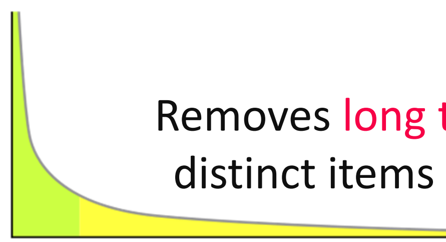
982	850	867	Exceptions

Patched Compression Methods (Others)

- **PFOR-DELTA**
  - Apply cascade of DELTA – PFOR (PFOR on differences)
  - Handling of exceptions to handle large differences of subsequent values

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

- **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 and Evaluation

- **Experimental Survey**
  - Different data characteristics
  - Compression methods:

    DELTA, RLE, FOR, RLE, DICT,
    SIMD-BP128, SIMD-FastPFOR,
    4-Wise NS, 4-Gamme, Masked VByte,
    Simple-8b, SIMD-GroupSimple
  - Cascades of compression methods

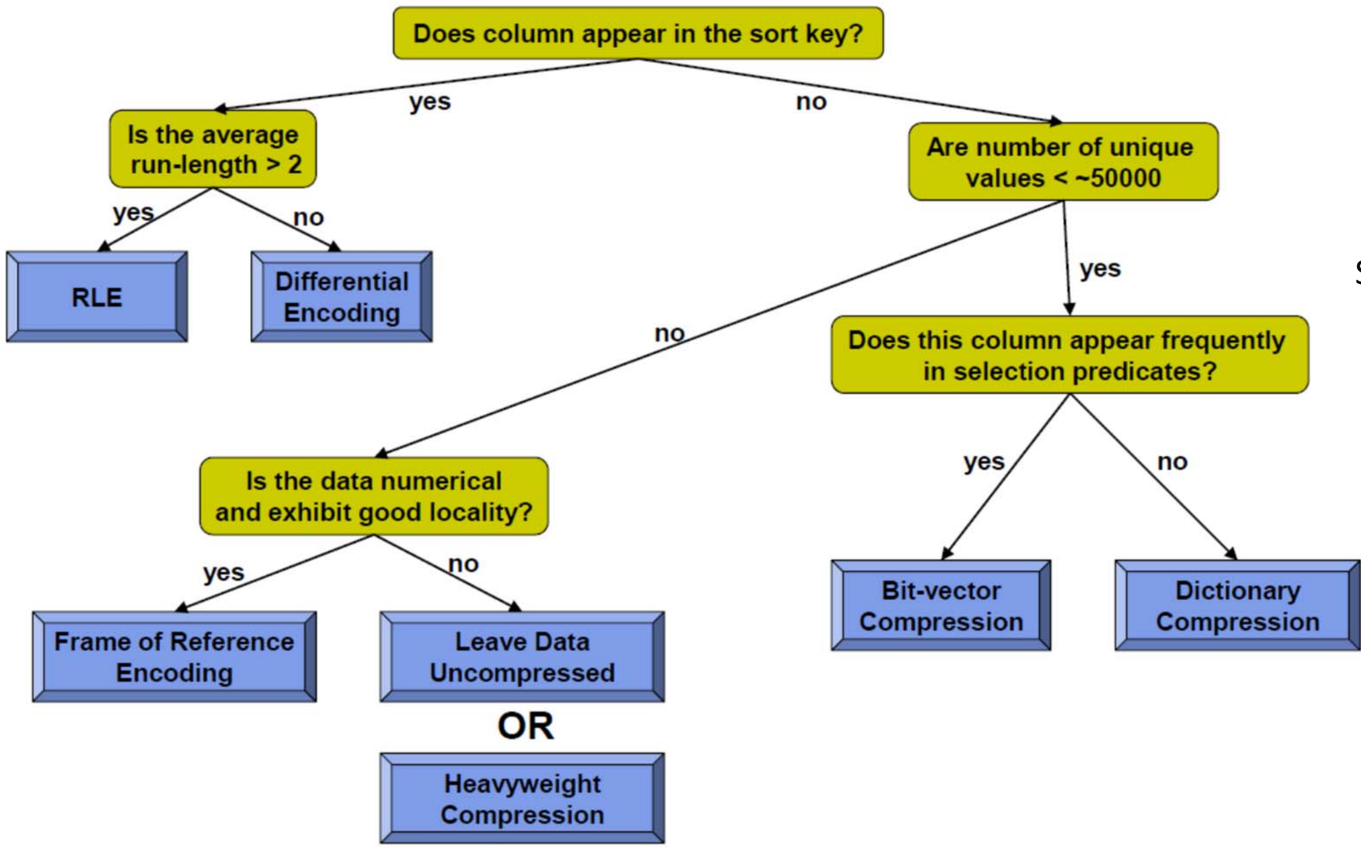
[Patrick Damme, Dirk Habich, Juliana Hildebrandt, Wolfgang Lehner: Lightweight Data Compression Algorithms: An Experimental Survey (Experiments and Analyses). **EDBT 2017**]

"[…] 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



[Peter Boncz: Column-Oriented Database Systems, adapted from **VLDB'09 tutorial**]

- **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 $\sigma_{attr='D'}(R)$**
  - DICT: code lookup

Dict	
0	A
1	C
2	B
3	D

D → 3

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

position vector

6	9	16

  - RLE: return RLE runs

C	5	A	4	D	5	F	3

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

Selection Predictions, 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
```

Grouping and Aggregations

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

Dict	
0	A
1	C
2	B
3	D

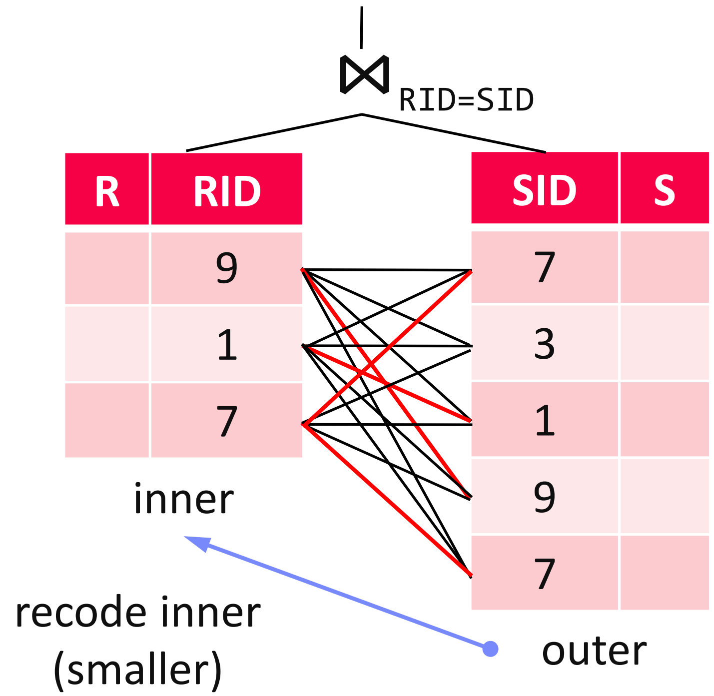
0	1	2	2	0	1	3	0	0	3	1	2	0	2	2	1	3

Hash Table	
0	Agg A
3	Agg D
1	Agg C
2	Agg B

- **Encoding-Specific Aggregation**
  - RLE sum → agg += run-length*run-value
  - RLE min → 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)



- **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-oriented database systems. **SIGMOD 2006**]

  - `isOneValue()`: block contains just one value and many positions for that value
  - `isValueSorted()`: all values of the block are sorted
  - `isPosContig()`: block contains consecutive subset of column

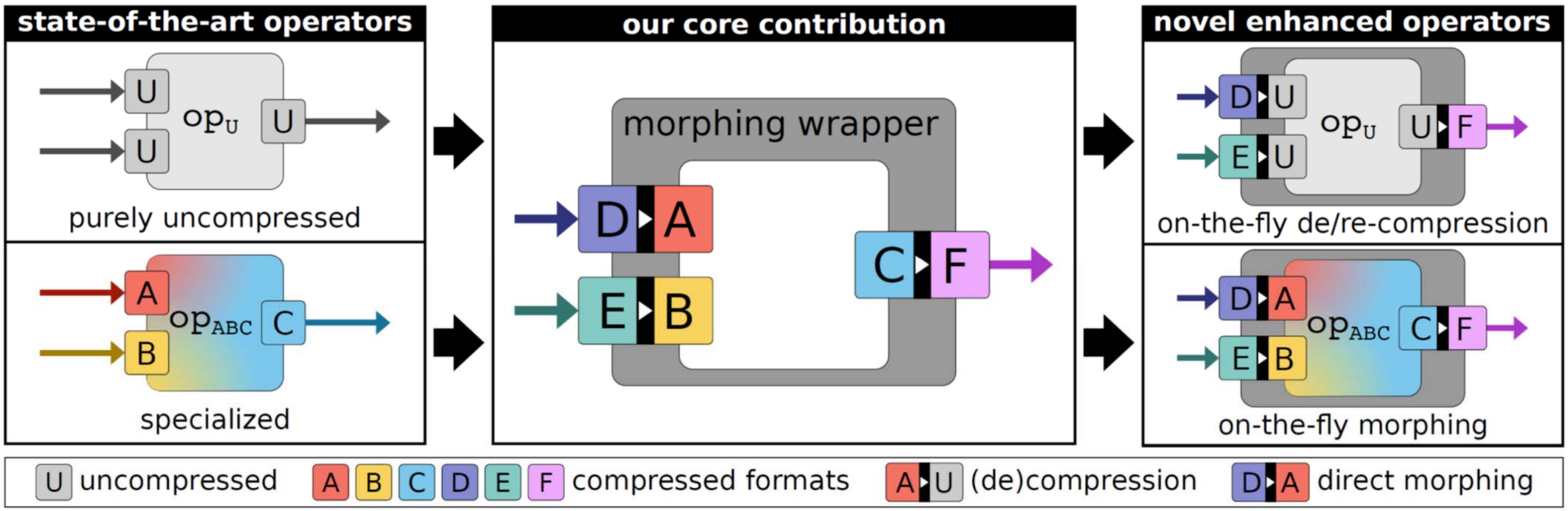
- **Iterator Access:** `getNext()`, `asArray()`
- **Block Information:** `getSize()`, `getStartValue()`, `getEndPosition()`

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

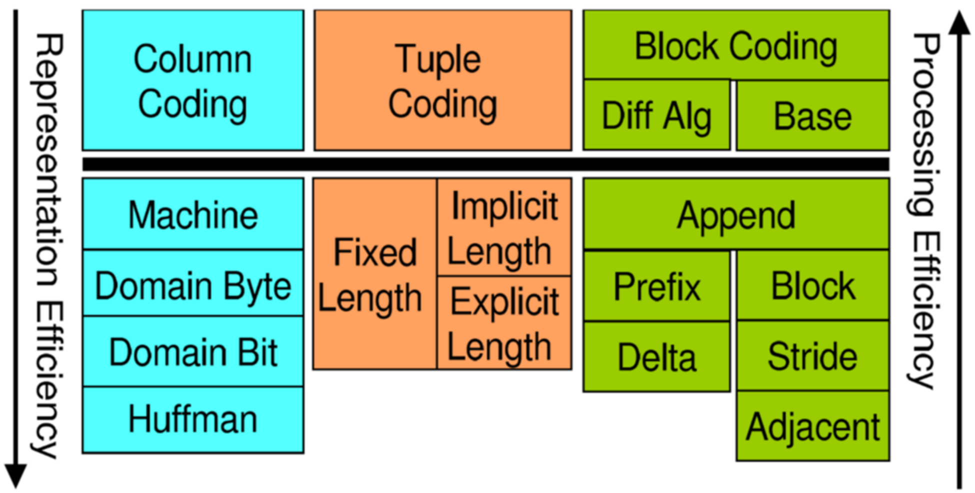
Abstractions for Simpler Code, cont.

- **Motivation**
  - Improve query performance by (re)compressing intermediates
  - Change from one compressed format to another

[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**]



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**]

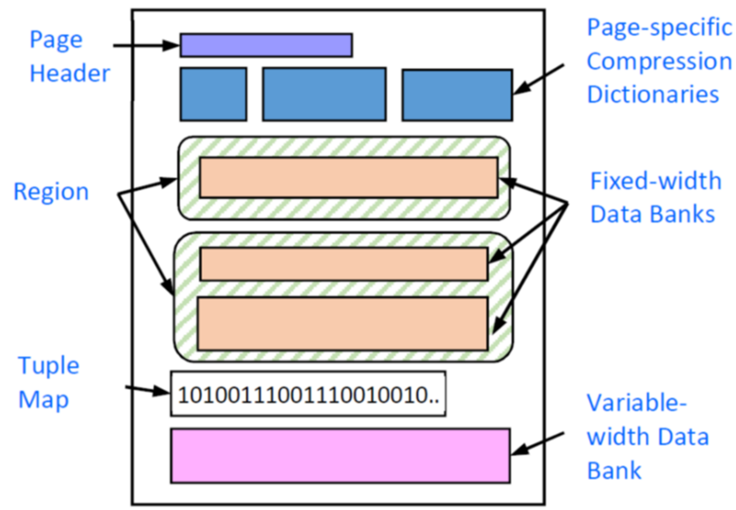
"All the results have shown that the Huffman coded and delta coded formats compress better but normally take more CPU time. […] When I/O and memory subsystem times are also included in the decision, the format to choose becomes less clear-cut. If a physical format optimizer or system administrator had 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)
- **Block Coding**
  - Compress a sequence of tuples into a compressed block (concat, diff)

03 Buffer Pool Management

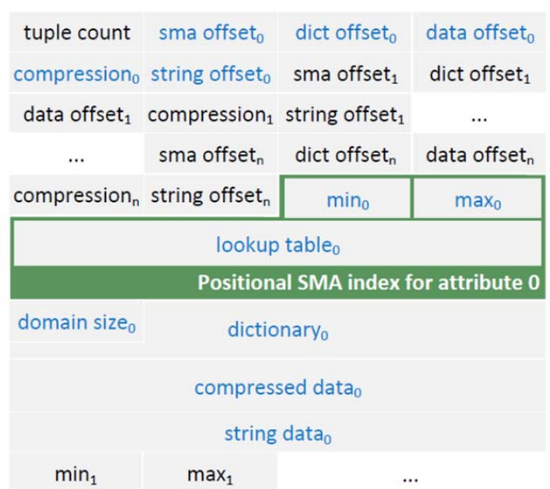
Data Layout – Example Block Layouts

- **DB2 BLU**



[Vijayshankar Raman et al: DB2 with BLU Acceleration: So Much More than Just a Column Store. **PVLDB 6(11) 2013**]

- **Data Blocks**



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

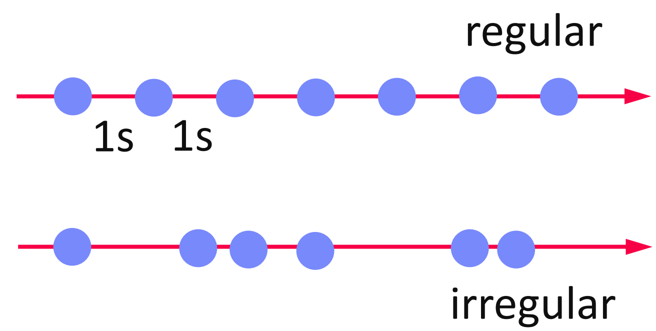
03 Buffer Pool Management
04 Index Structures and Partitioning
07 Query Compilation and Parallelization

Time Series Compression

Motivation and Terminology

- **Ubiquitous Time Series**
  - **Domains:** Internet-of-Things (IoT), sensor networks, smart production/planet, telemetry, stock trading, server/application metrics, event/log streams
  - **Applications:** monitoring, anomaly detection, time series forecasting
  - Dedicated storage and analysis techniques → Specialized systems

- **Terminology**
  - Time series X is a sequence of data points x_i for a specific measurement identity (e.g., sensor) and time granularity
  - **Regular** (equidistant) time series (x_i) vs **irregular** time series (t_i, x_i)

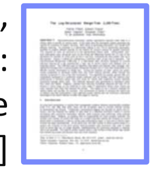






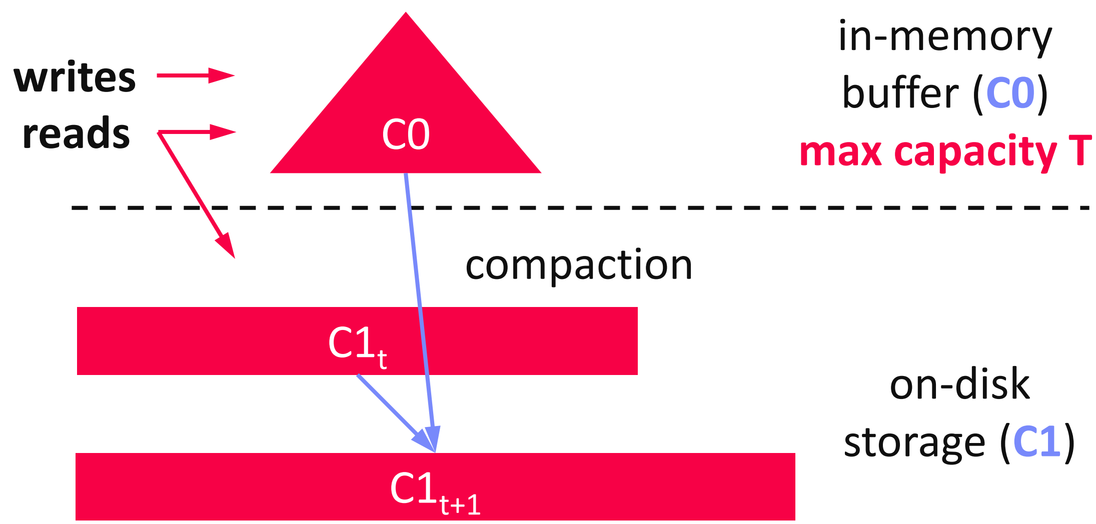

Log-structured Merge Trees

[Patrick E. O'Neil, Edward Cheng, Dieter Gawlick, Elizabeth J. O'Neil: The Log-Structured Merge-Tree (LSM-Tree). **Acta Inf. 1996**]



- **LSM Overview**
  - Many KV-stores rely on LSM-trees as their storage engine (e.g., **BigTable**, **DynamoDB**, **LevelDB**, **Riak**, **RocksDB**, **Cassandra**, **HBase**)
  - **Approach:** Buffers writes in memory, flushes data as sorted runs to storage, merges runs into larger runs of next level (compaction)

- **System Architecture**
  - Writes in C0
  - Reads against C0 and C1 (w/ buffer for C1)
  - Compaction (rolling merge): sort, merge, including **deduplication**



Example InfluxDB

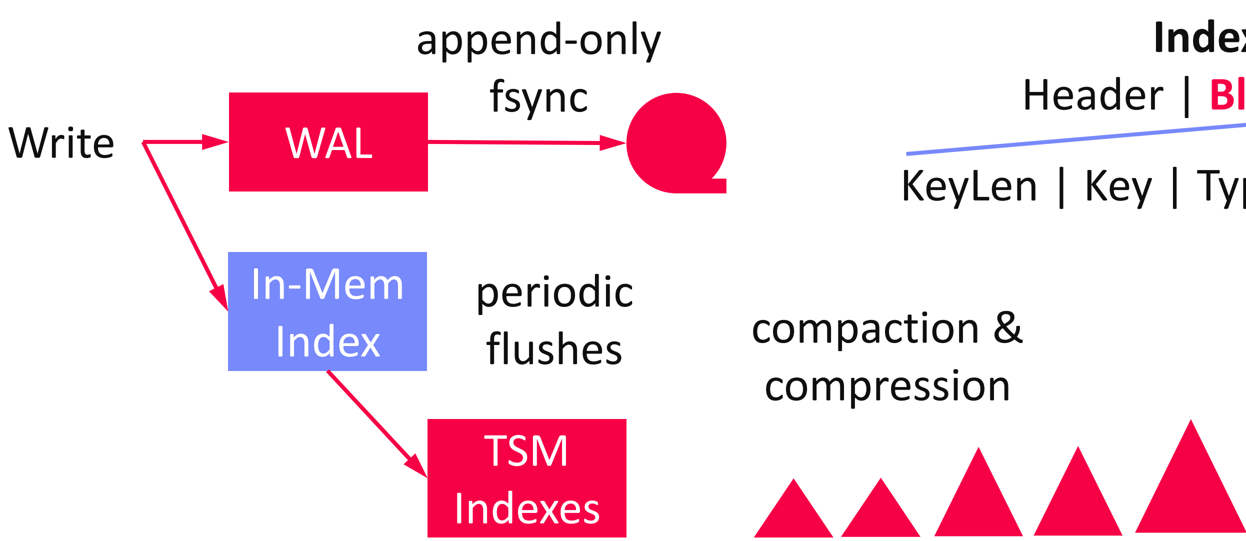
- **Input Data**

```
cpu,region=west,host=A
 user=85,sys=2,idle=10 1443782126
```

cpu = Measurement; region=west,host=A = Tags; user=85,sys=2,idle=10 = Fields (values); 1443782126 = Time

[Paul Dix: InfluxDB Storage Engine Internals, CMU Seminar, 09/2017]

- **System Architecture**
  - Written in Go, originally **key-value store**, now **dedicated storage engine**
  - **Time Structured Merge Tree (TSM)**, similar to LSM
  - Organized in shards, TSM indexes and inverted index for reads



Index per TSM file:

Header | **Blocks** | Index | Footer

KeyLen | Key | Type | Min T | Max T| Off | …

periodic drop of shards (files) according to retention policy

Example InfluxDB, cont.

- **Compression (of blocks)**
  - Compress up to 1000 values per block (Type | Len | Timestamps | Values)
  - **Timestamps:** Delta + Run-length encoding for regular time series; Simple8B or uncompressed for irregular
  - **Values:** double delta for FP64, bits for Bool, double delta + zig zag for INT64, Snappy for strings

- **Query Processing**
  - SQL-like and functional APIs for filtering (e.g., range) and aggregation
  - Inverted indexes

```
SELECT percentile(90, user)
  FROM cpu WHERE time>now()-12h
  AND "region"='west'
  GROUP BY time(10m), host
```

Measurement to fields:
cpu → [user,sys,idle]
host → [A, B]
Region → [west, east]

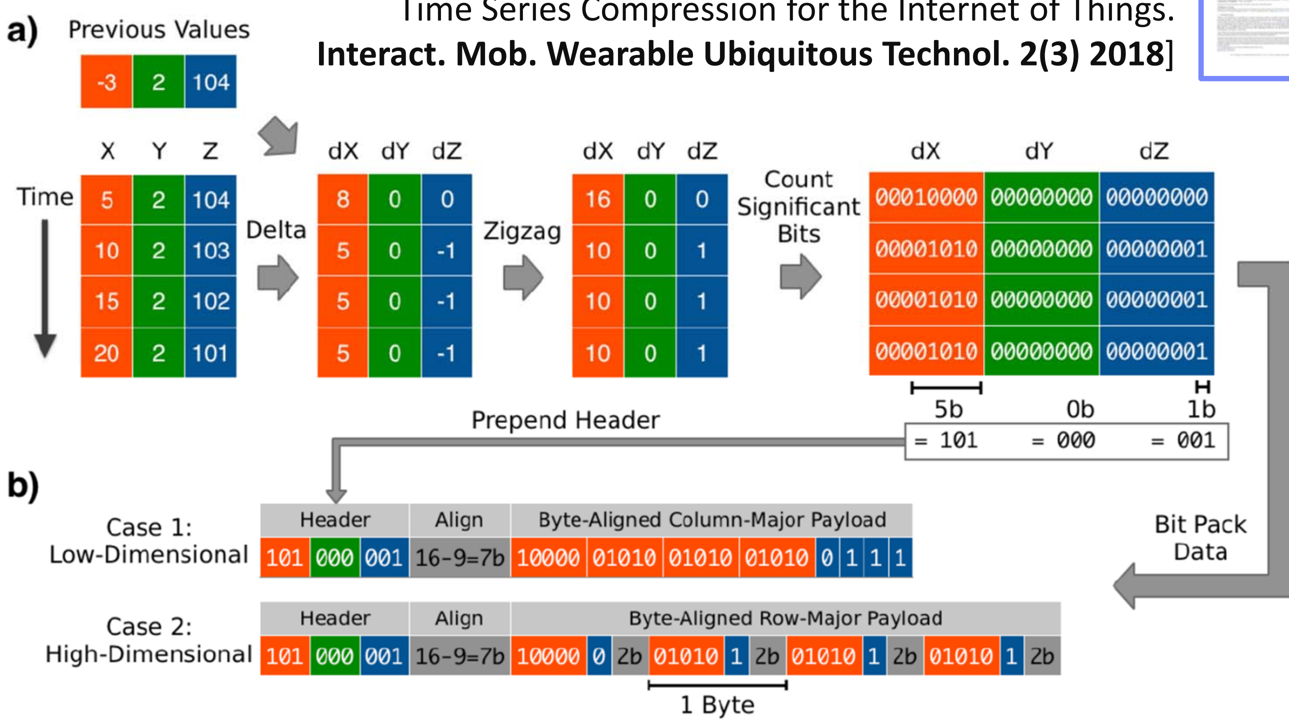
Posting lists:
cpu → [1,2,3,4,5,6]
host=A → [1,2,3]
host=B → [4,5,6]
region=west → [1,2,3]

Lossless, Predictive Time Series Compression

- **Motivation**
  - Sampled sensor data with lots of compression potential
  - Small blocksize (end devices), fast decompression, lossless

- **Sprintz**
  - Forecasting
  - Bit packing
  - RLE zeros
  - Entropy coding

[Davis W. Blalock, Samuel Madden, John V. Guttag: Sprintz: Time Series Compression for the Internet of Things. **Interact. Mob. Wearable Ubiquitous Technol. 2(3) 2018**]



Summary and Q&A

- **Motivation and Terminology**
- **Compression Techniques**
- **Compressed Query Processing**
- **Time Series Compression**

- **Next Lectures** (Part B)
  - **Nov 11:** no lecture, work on your programming projects
  - **06 Query Processing** (operators, execution models) [Nov 18]
  - **07 Query Compilation and Parallelization** [Nov 25]
  - **08 Query Optimization I** (normalization, rewrites, unnesting) [Dec 02]
  - **09 Query Optimization II** (cost models, join ordering) [Dec 09]
  - **10 Adaptive Query Processing** [Dec 16]