TECHNISCHE UNIVERSITÄT DRESDEN

Dresden Database Systems Group

Make Larger Vector Register Sizes New Challenges?

Lessons Learned from the Area of Vectorized Lightweight Compression Algorithms

Dirk Habich, Patrick Damme, Annett Ungethüm, Wolfgang Lehner

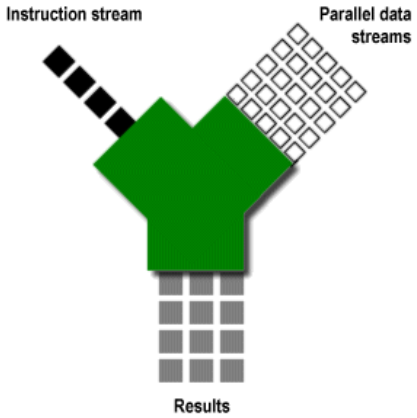
DBTest Workshop, 2018-06-15

Vectorization using SIMD

Dresden Database Systems Group

Single Instruction Multiple Data (SIMD)

- same instruction on multiple data elements simultaneously



Heavily used in Database Systems

- to increase (single-thread) performance

Rethinking SIMD Vectorization for In-Memory Databases

Orestis Polychroniou* Columbia University
Arun Raghavan Oracle Labs
Kenneth A. Ross† Columbia University

Boosting Data Filtering on Columnar Encoding with SIMD

Hao Jiang
The University of Chicago
hajiang@cs.uchi

Aaron J. Elmore
The University of Chicago

Make the Most out of Your SIMD Investments: Counter Control Flow Divergence in Compiled Query Pipelines

Harald Lang
Technical University of Munich
Munich, Germany
harald.lang@in.tum.de

Andreas Kipf
Technical University of Munich
Munich, Germany
andreas.kipf@in.tum.de

Linnea Passing
Technical University of Munich
Munich, Germany
linnea.passing@in.tum.de

Peter Boncz
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
boncz@cwi.nl

Thomas Neumann
Technical University of Munich
Munich, Germany
thomas.neumann@in.tum.de

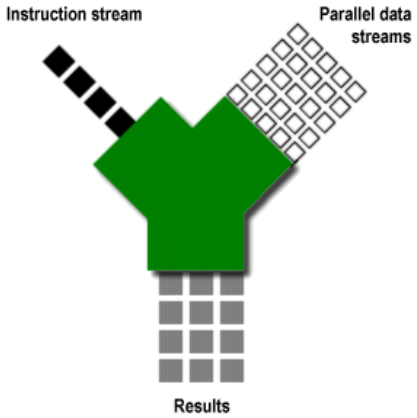
Alfons Kemper
Technical University of Munich
Munich, Germany
alfons.kemper@in.tum.de

Trend of Increasing Vector Register Sizes

Dresden Database Systems Group

Single Instruction Multiple Data (SIMD)

- same instruction on multiple data elements simultaneously



Heavily used in Database Systems

- to increase (single-thread) performance

Rethinking SIMD Vectorization for In-Memory Databases

Orestis Polychroniou* Columbia University
Arun Raghavan Oracle Labs
Kenneth A. Ross† Columbia University

Boosting Data Filtering on Columnar Encoding with SIMD

Hao Jiang
The University of Chicago
hajiang@cs.uchi

Aaron J. Elmore
The University of Chicago

Make the Most out of Your SIMD Investments: Counter Control Flow Divergence in Compiled Query Pipelines

Harald Lang
Technical University of Munich
Munich, Germany
harald.lang@in.tum.de

Andreas Kipf
Technical University of Munich
Munich, Germany
andreas.kipf@in.tum.de

Linnea Passing
Technical University of Munich
Munich, Germany
linnea.passing@in.tum.de

Peter Boncz
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
boncz@cwi.nl

Thomas Neumann
Technical University of Munich
Munich, Germany
thomas.neumann@in.tum.de

Alfons Kemper
Technical University of Munich
Munich, Germany
alfons.kemper@in.tum.de

INTEL SIMD Development

SSE (128 bit) 4 → AVX(2) (256 bit) 8 → AVX-512 (512 bit) 16

Data Compression

Dresden Database Systems Group

Technique = abstract idea of how compression works

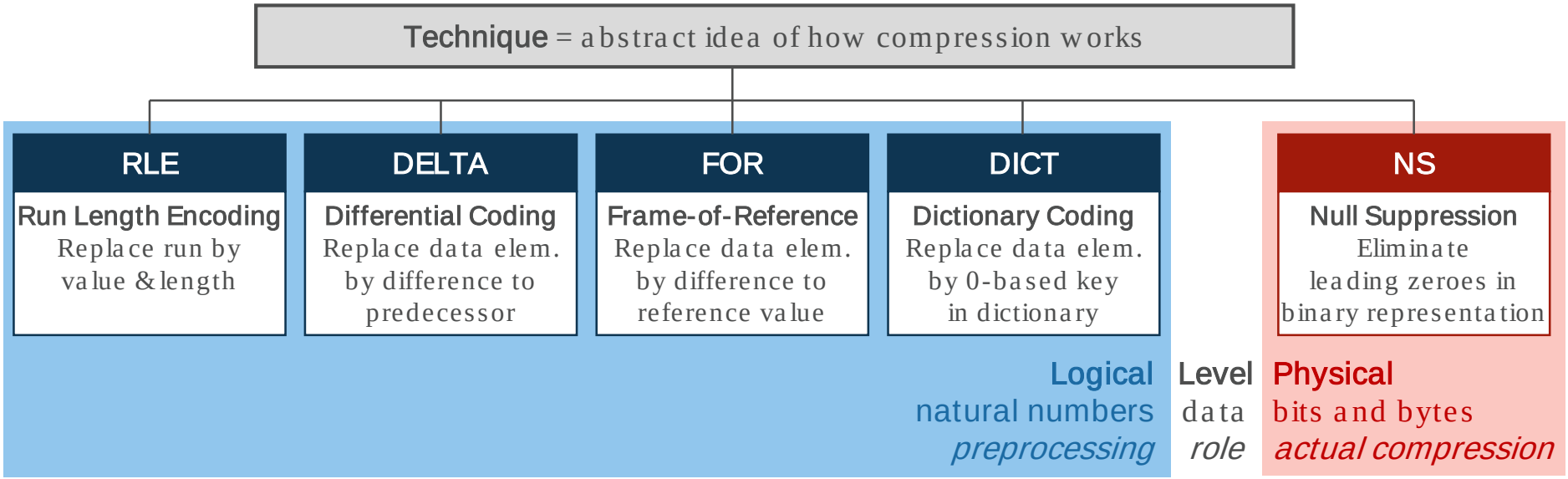
RLE	DELTA	FOR	DICT	NS
Run Length Encoding Replace run by value & length	**Differential Coding** Replace data elem. by difference to predecessor	**Frame-of-Reference** Replace data elem. by difference to reference value	**Dictionary Coding** Replace data elem. by 0-based key in dictionary	**Null Suppression** Eliminate leading zeroes in binary representation

Logical natural numbers *preprocessing*

Level data *role*

Physical bits and bytes *actual compression*

Data Compression

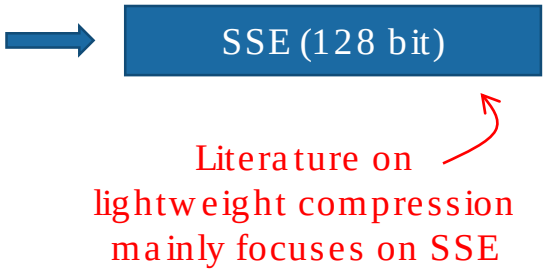


Algorithm = concrete combination of one or more of these techniques

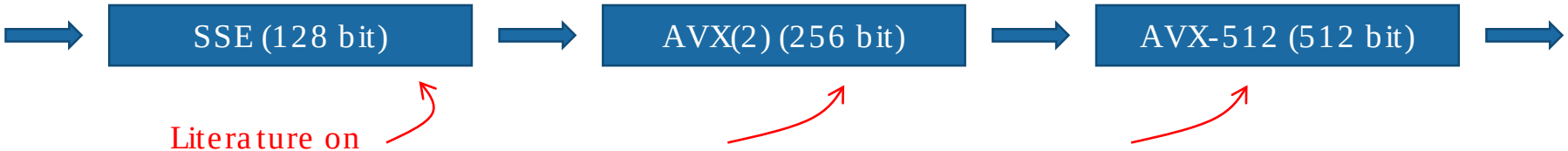
Two algorithms for the same technique might differ in, e.g.

- their **data layout**
- their use of **vectorization** using SIMD instruction set extensions

Vectorization and Compression

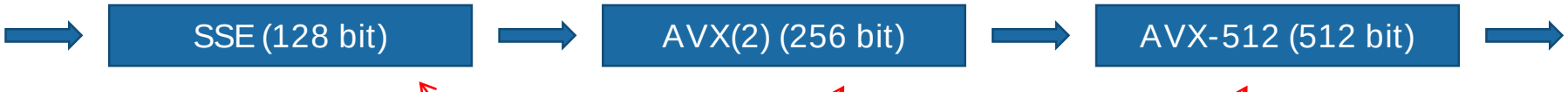


Vectorization and Compression



Vectorization and Compression

Dresden Database Systems Group



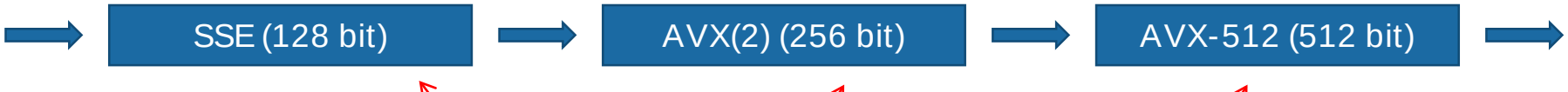
Literature on lightweight compression mainly focuses on SSE

Increasing vector sizes promise performance improvements (hopefully speedup of 2 respectively 4 compared to 128-bit)

How to employ recent SIMD extensions for lightweight data compression?

Vectorization and Compression

Dresden Database Systems Group



Literature on lightweight compression mainly focuses on SSE

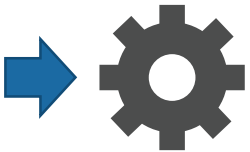
Increasing vector sizes promise performance improvements (hopefully speedup of 2 respectively 4 compared to 128-bit)

How to employ recent SIMD extensions for lightweight data compression?

SSE-implementation of some algorithm

straightforward port

- Substitute SSE instructions for their AVX(2) and AVX-512 counterparts
- Slightly adapt memory layout where necessary
- Easy to do (if possible)



AVX-implementation
AVX-512-implementation

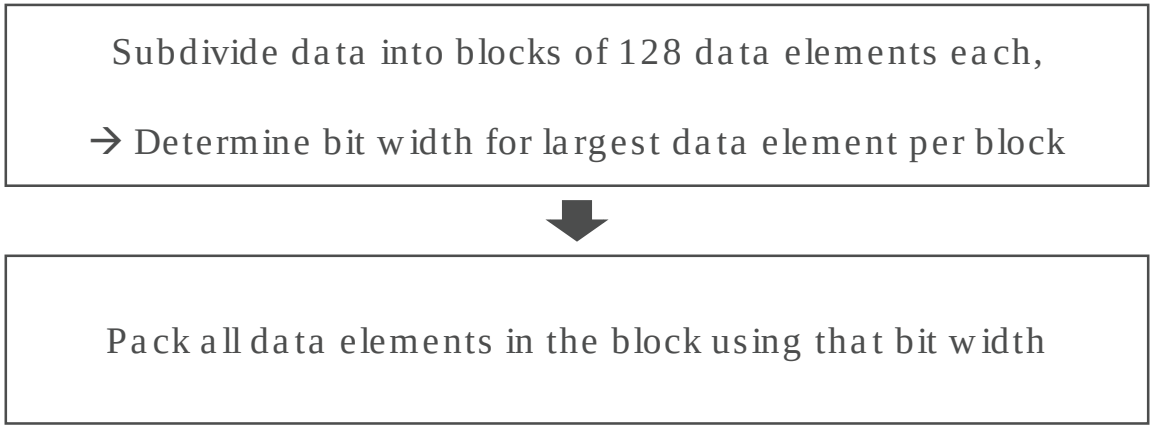
Evaluation - First Example

SIMD-BP128* – Basic Idea

Dresden Database Systems Group

Null Suppression Algorithm

- very efficient from a performance as well as compression ratio perspective



* D. Lemire and L. Boytsov. Decoding billions of integers per second through vectorization. Softw., Pract. Exper., 45(1), 2015.

SIMD-BP128 – Compression

Dresden Database Systems Group

Uncompressed input (32 bits/int)

Compressed output (10 bits/int)

SIMD-BP128 – Compression

Dresden Database Systems Group

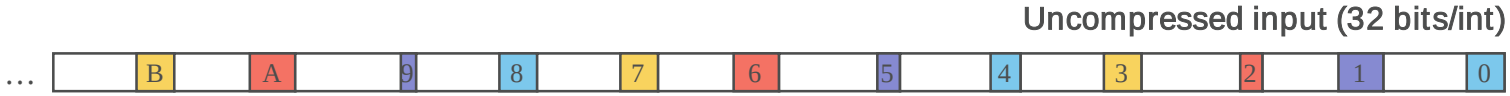
Uncompressed input (32 bits/int)

Processing

Compressed output (10 bits/int)

SIMD-BP128 – Compression

Dresden Database Systems Group



Processing

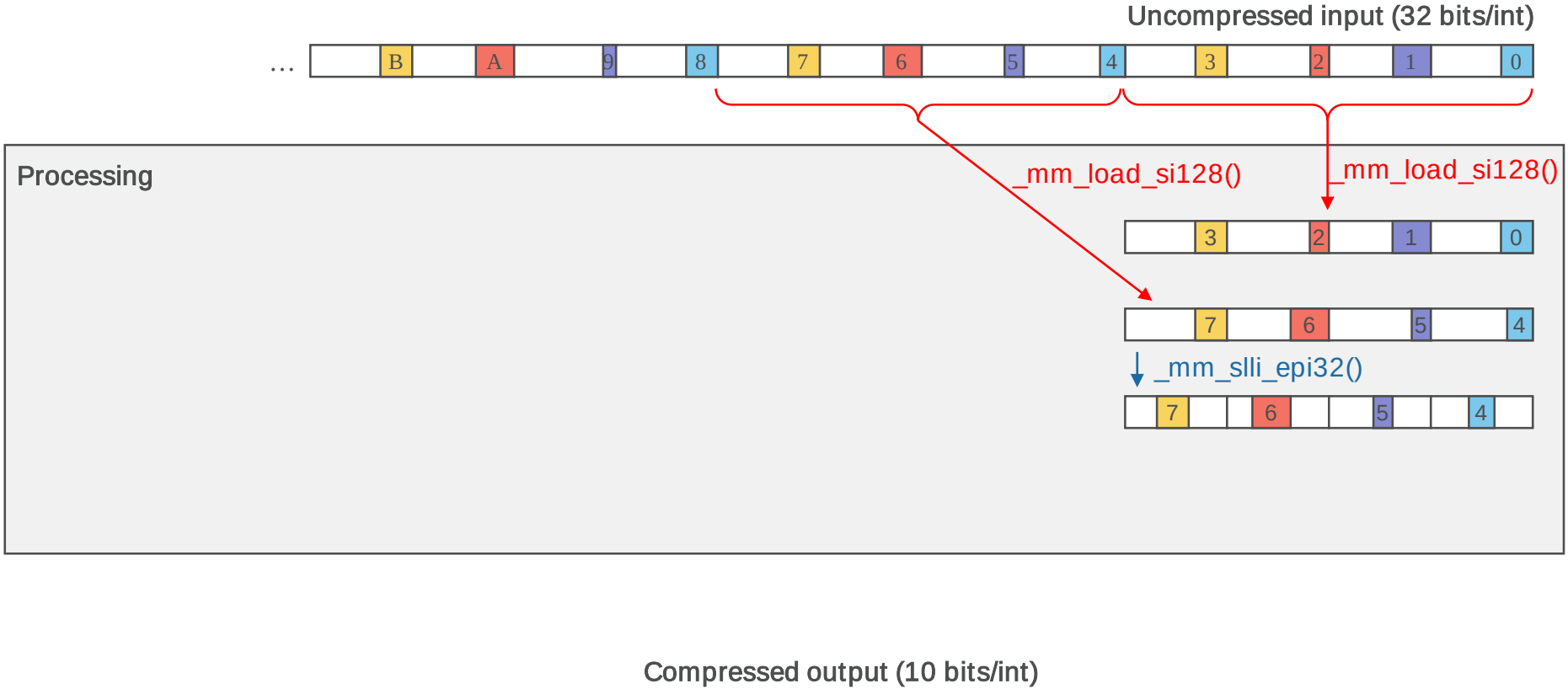
_mm_load_si128()

Compressed output (10 bits/int)

SIMD-BP128 – Compression

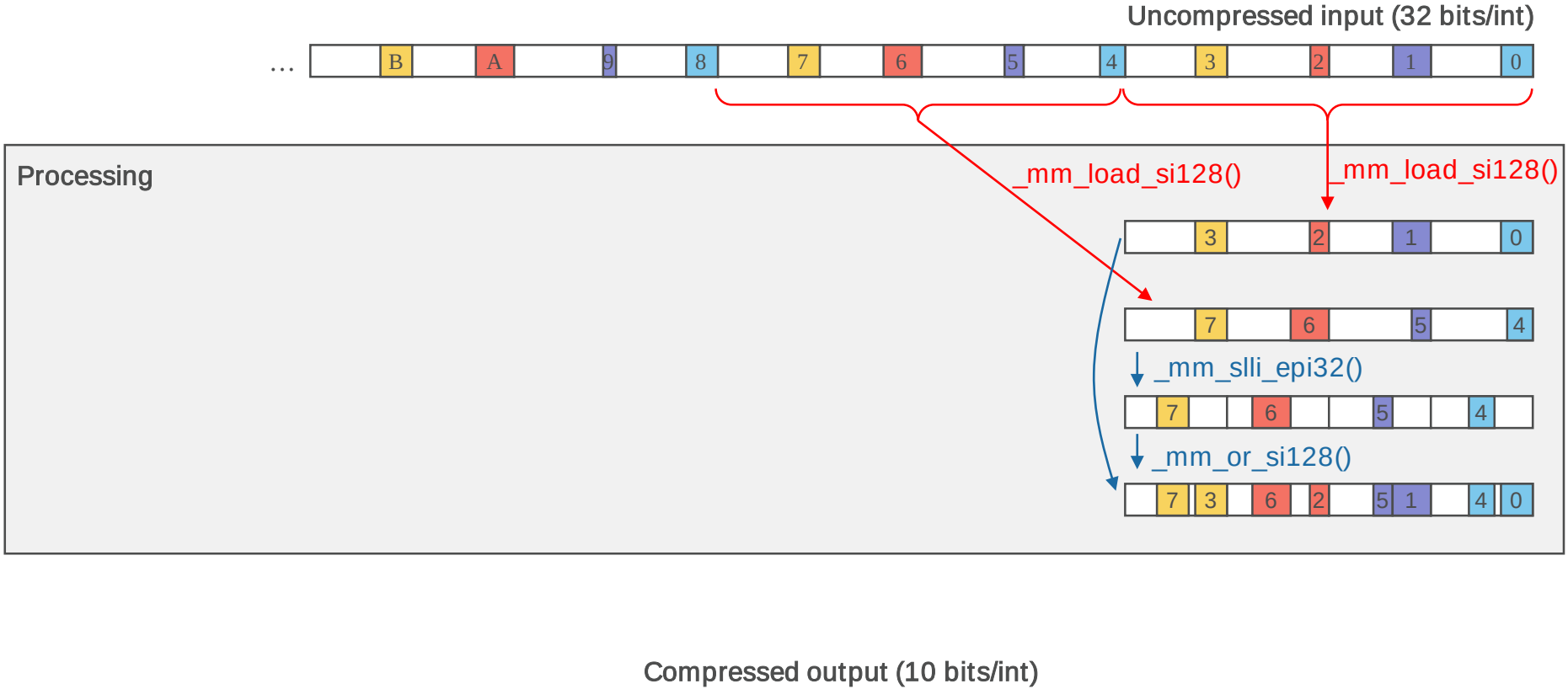
SIMD-BP128 – Compression

Dresden Database Systems Group



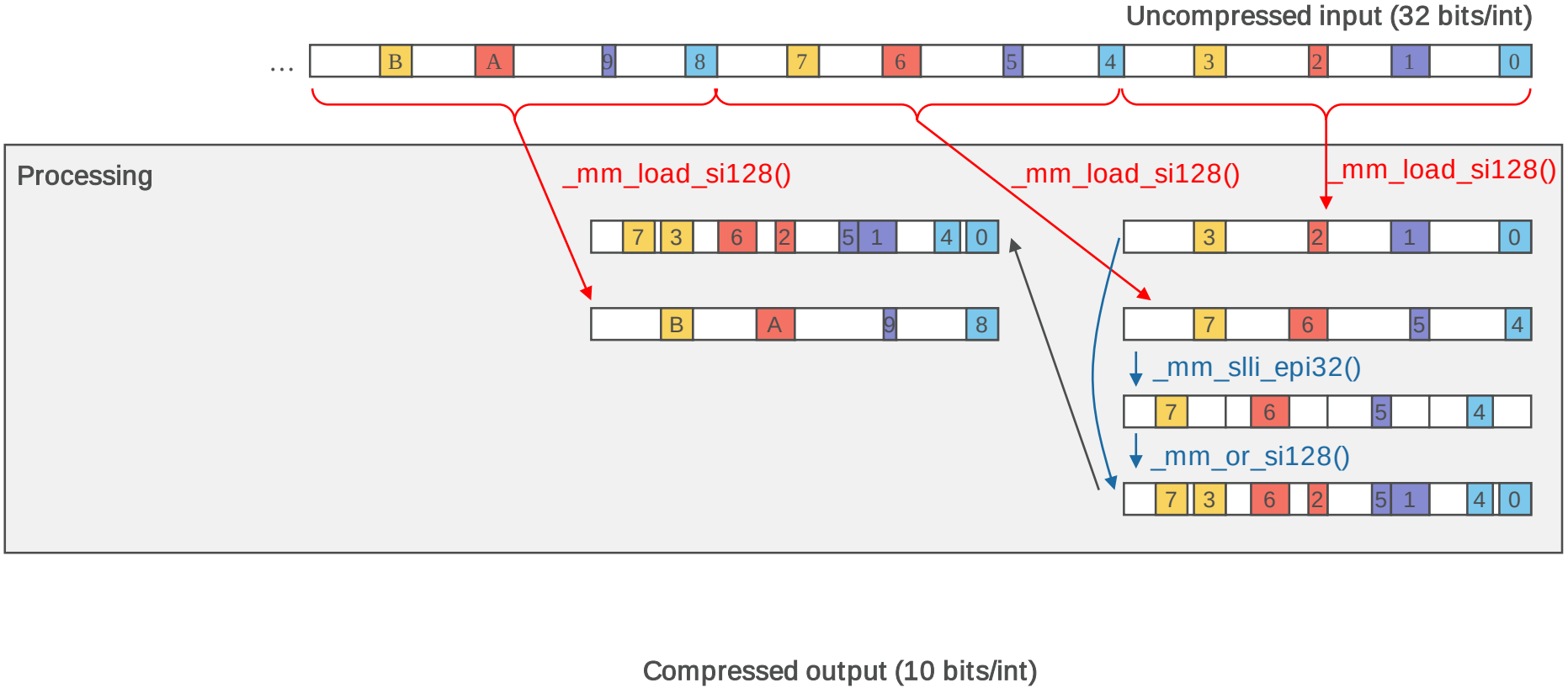
SIMD-BP128 – Compression

Dresden Database Systems Group



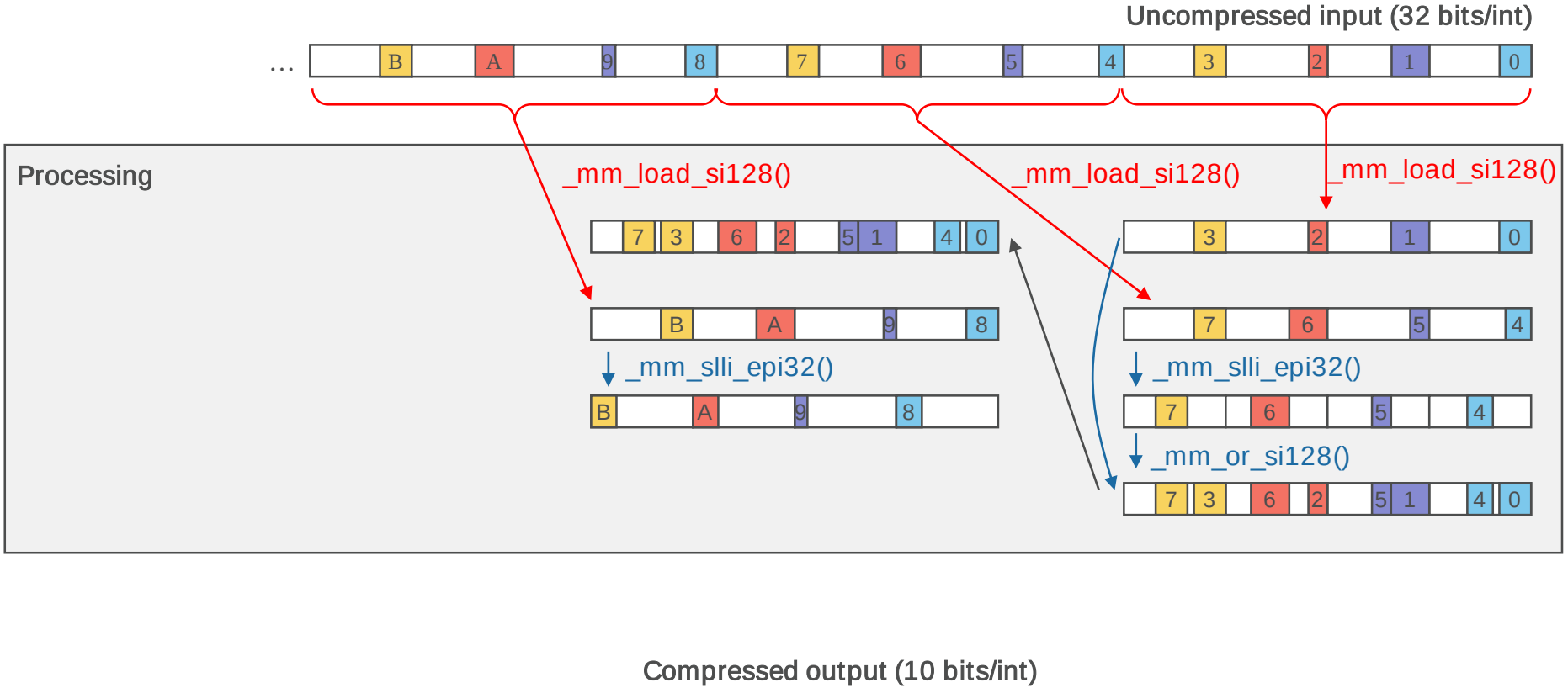
SIMD-BP128 – Compression

Dresden Database Systems Group



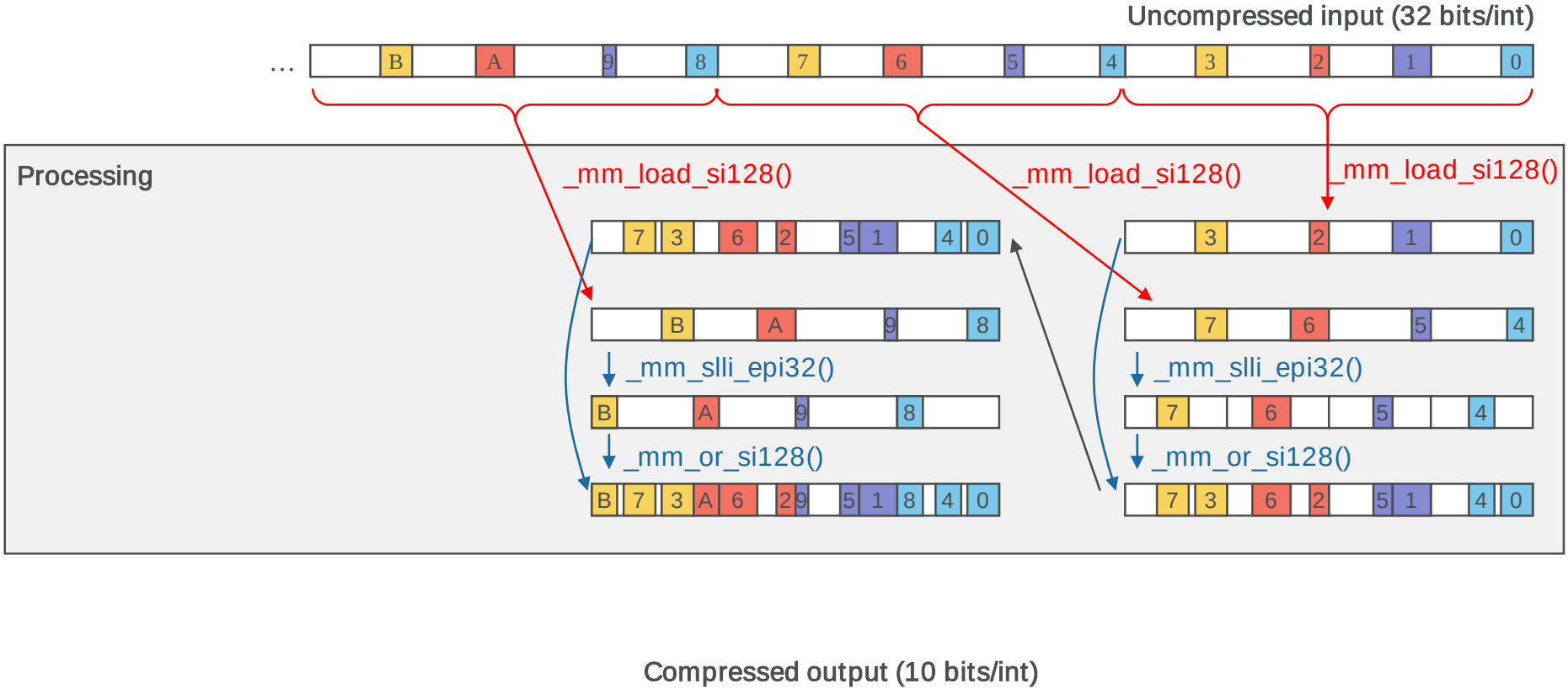
SIMD-BP128 – Compression

Dresden Database Systems Group



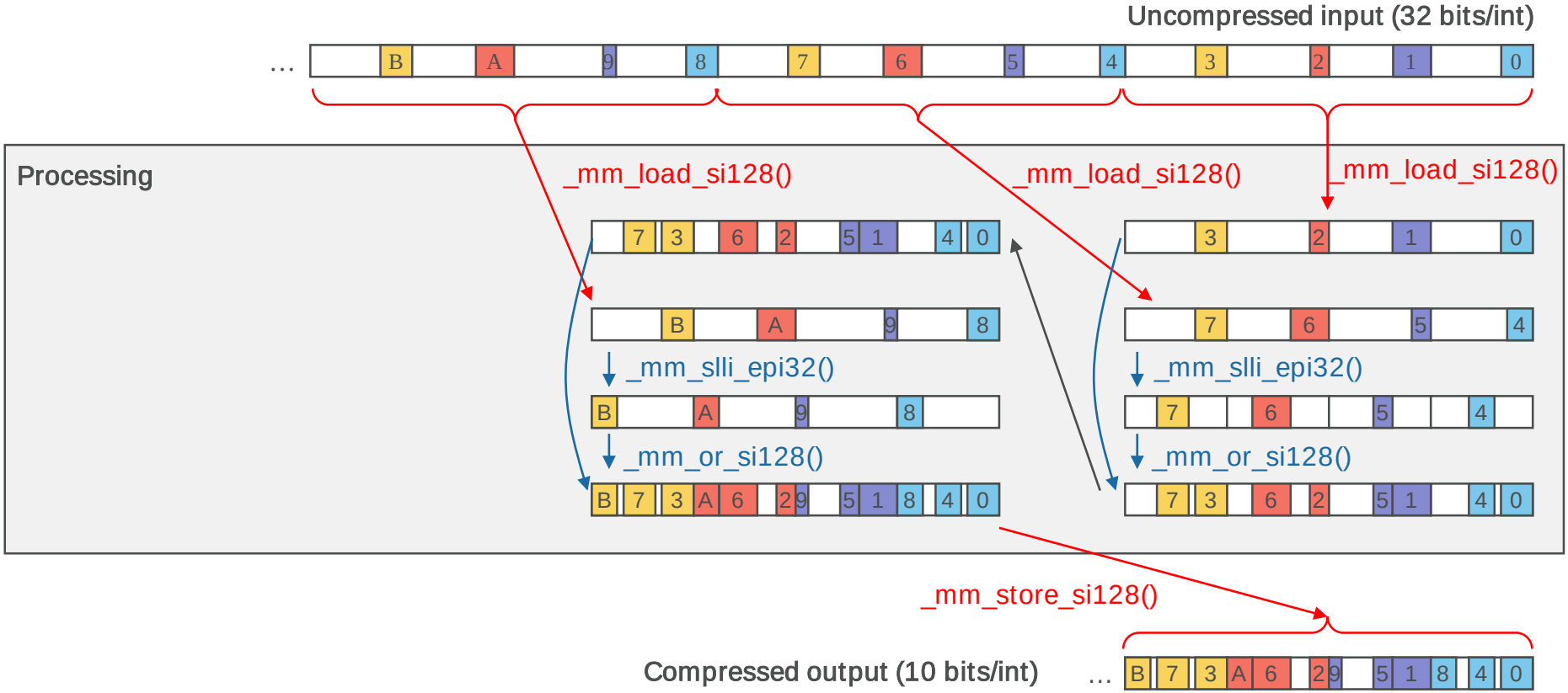
SIMD-BP128 – Compression

Dresden Database Systems Group



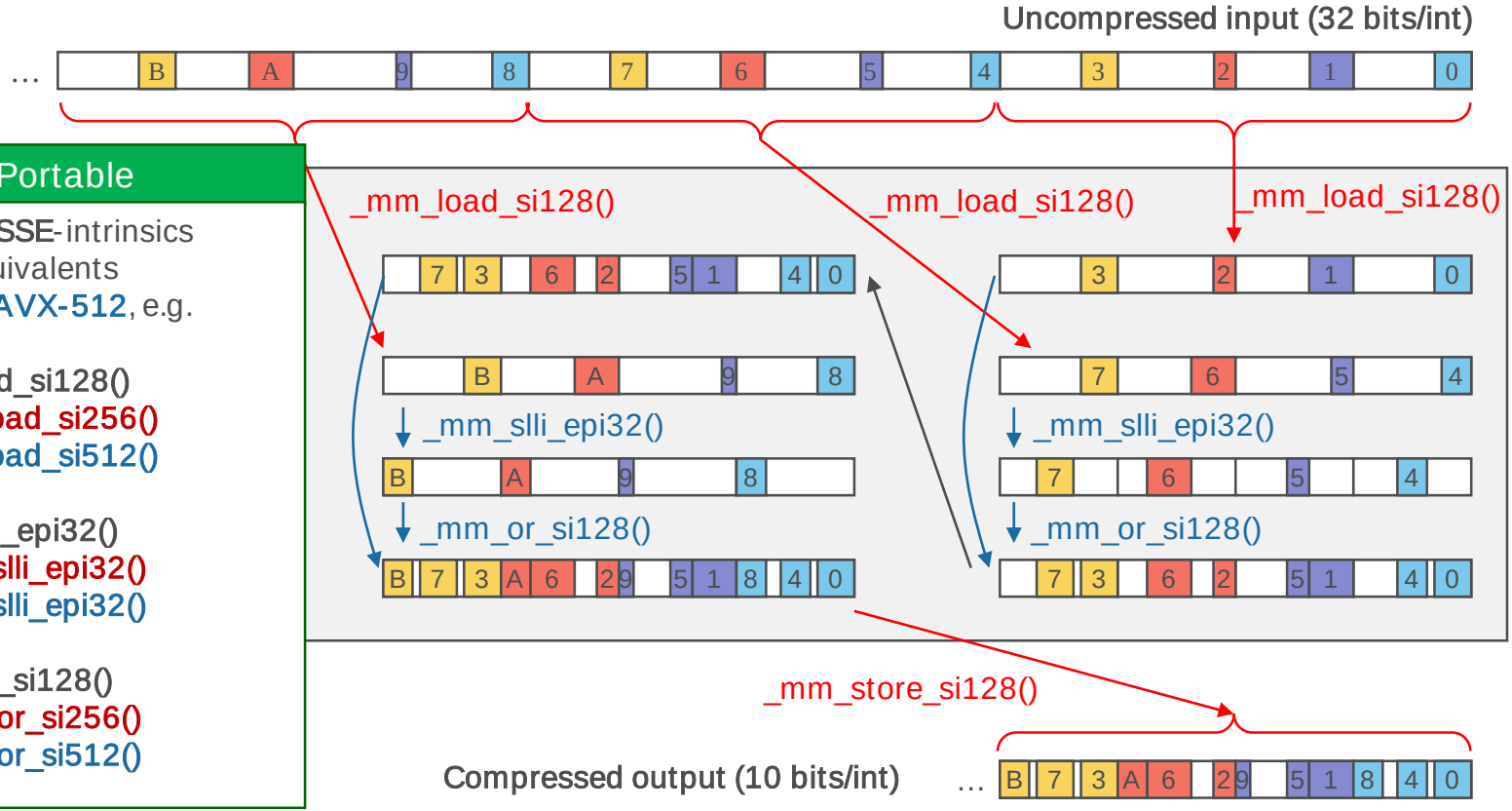
SIMD-BP128 – Compression

Dresden Database Systems Group



SIMD-BP128 – Compression

Dresden Database Systems Group



Trivially Portable

All employed **SSE**-intrinsics have equivalents in **AVX2** and **AVX-512**, e.g.

_mm_load_si128()
_mm256_load_si256()
_mm512_load_si512()

_mm_slli_epi32()
_mm256_slli_epi32()
_mm512_slli_epi32()

_mm_or_si128()
_mm256_or_si256()
_mm512_or_si512()

SIMD-BP128* – Idea

Dresden Database Systems Group

block size = size of vector register in bits
→ guarantees alignment of output

Subdivide data into blocks of 128 data elements each,

→ Determine bit width for largest data element per block

Pack all data elements in the block using that bit width

* D. Lemire and L. Boytsov. Decoding billions of integers per second through vectorization. Softw., Pract. Exper., 45(1), 2015.

SIMD-BP128* – Idea

Dresden Database Systems Group

block size = size of vector register in bits
→ guarantees alignment of output

Ported versions need blocks of
- 256 elements (AVX2)
- 512 elements (AVX-512)

Subdivide data into blocks of 128 data elements each,

→ Determine bit width for largest data element per block

↓

Pack all data elements in the block using that bit width

* D. Lemire and L. Boytsov. Decoding billions of integers per second through vectorization. Softw., Pract. Exper., 45(1), 2015.

Evaluation Setup

Dresden Database Systems Group

Algorithms

- Implemented in C/C++
  - Some by the original authors
  - Some implemented by us
- Compiled using g++-7.0.1 –O3

Synthetic Data

- Allows to vary the data properties carefully

Evaluation System

- Intel Xeon Phi 7250
  - 1.4 GHz
  - L1-cache: 32 KB (data)
  - L2-cache: 1 MB
- 6x32 GB DDR4 @ 2400 MHz

Measurements

- All experiments completely in-memory
- disk not touched during time measurements

- Compression ratio reported in bits/int
  - Lower is better
  - Uncompressed data has 32 bits/int

- Speeds reported in million integers per second (mis)
  - Higher is better
  - Only single-thread performance

SIMD-BP128 – Evaluation

Dresden Database Systems Group

Data

- 100 M 32-bit integers
- Unsorted
- 4-bit values vs. 28-bit outliers
- We vary the outlier ratio

SIMD-BP128 – Evaluation

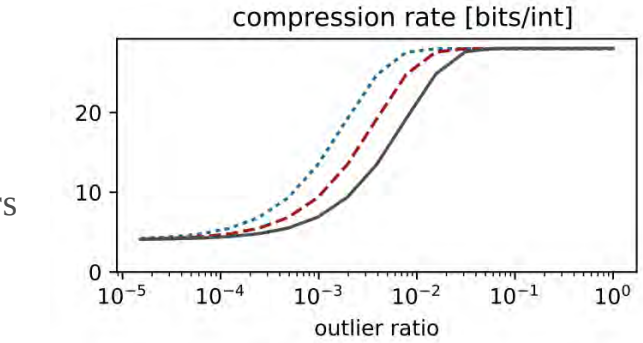
Dresden Database Systems Group

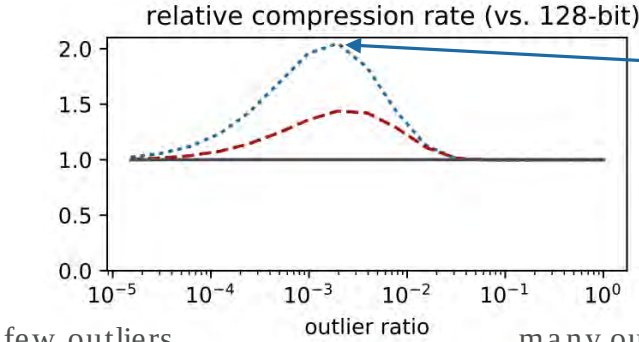
Data

- 100 M 32-bit integers
- Unsorted
- 4-bit values vs. 28-bit outliers
- We vary the outlier ratio

Insights

- Increasing vector size
  → increasing vulnerability to outliers





twice as much memory compared to 128-bit

few outliers — many outliers

SIMD-BP512 SIMD-BP256 SIMD-BP128

SIMD-BP128 – Evaluation

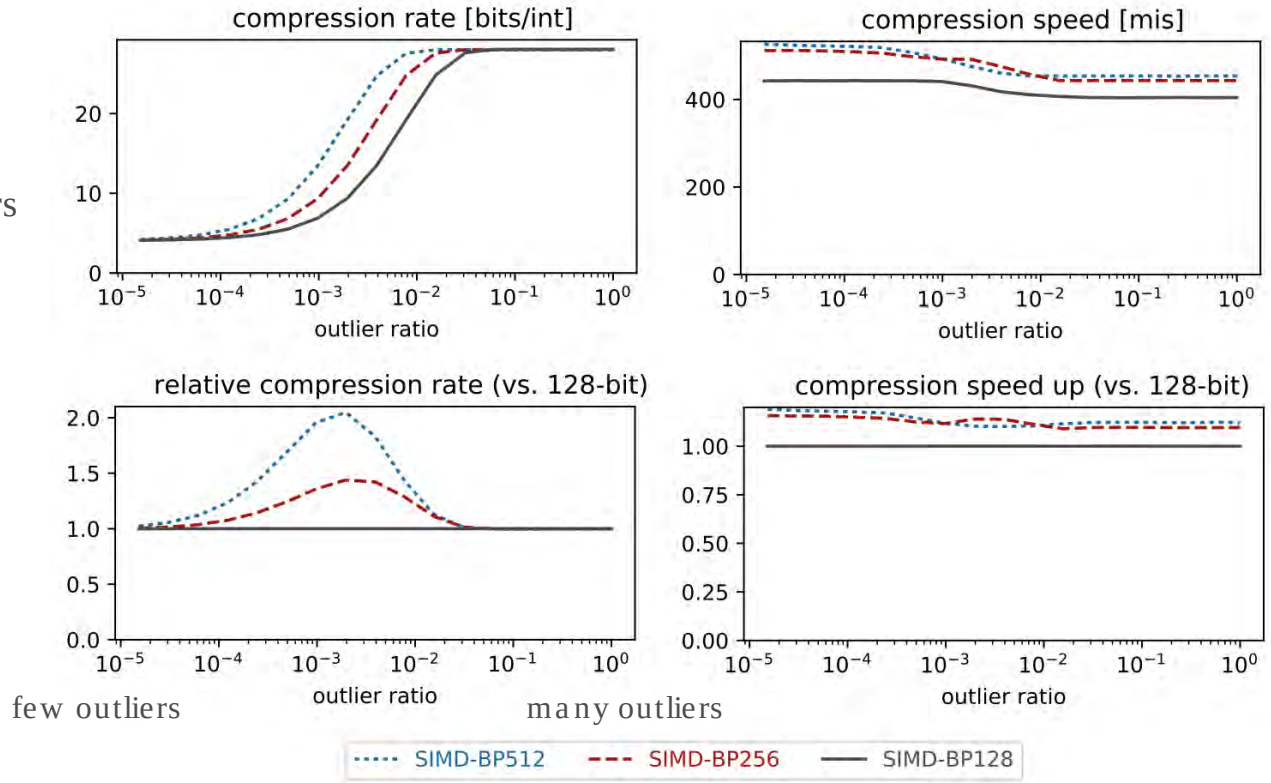
Dresden Database Systems Group

Data

- 100 M 32-bit integers
- Unsorted
- 4-bit values vs. 28-bit outliers
- We vary the outlier ratio

Insights

- Increasing vector size
  → increasing vulnerability to outliers
- Suboptimal speed ups
  - Far away from speedups of 2 or 4



Evaluation Second Example

Run-Length Encoding

Dresden Database Systems Group

Basic Idea

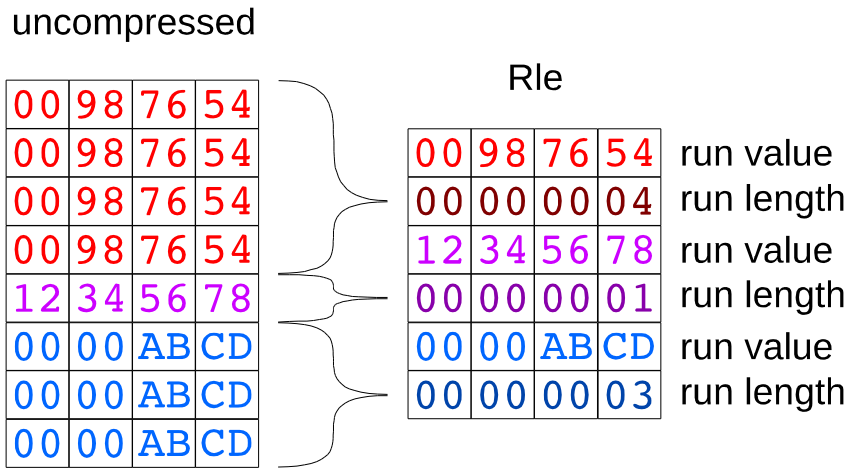
- View subsequent occurrences of the same value as a run
- Each run representable by its value and length → just two integers

RLE-SIMD

- Uses SIMD instructions to parallelize comparisons
- Proposed for 128-bit vectorization

Porting to Larger Vector Sizes

- easily portable using a straightforward approach



Evaluation using Different Vector Sizes

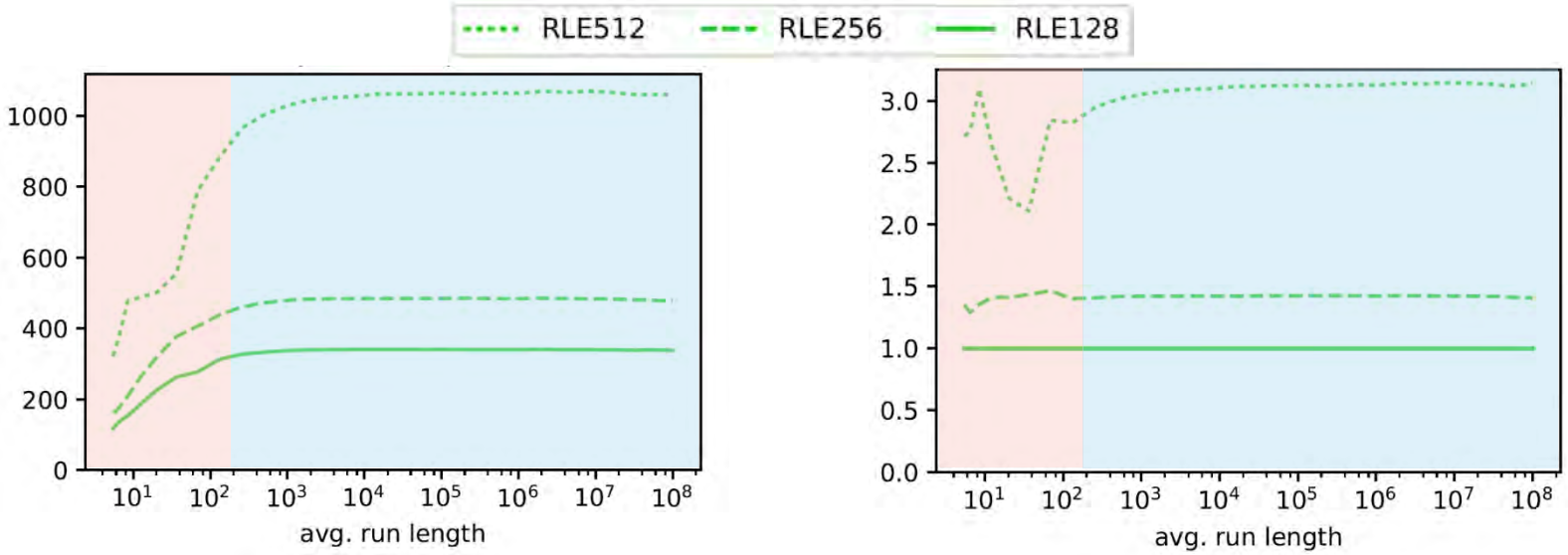
Dresden Database Systems Group

Compression Speed

- Measured in million integers per second (mis)

Speedup

- Compared to baseline of 128-bit



non-well performing area

well-performing area

Evaluation using Different Vector Sizes

Dresden Database Systems Group

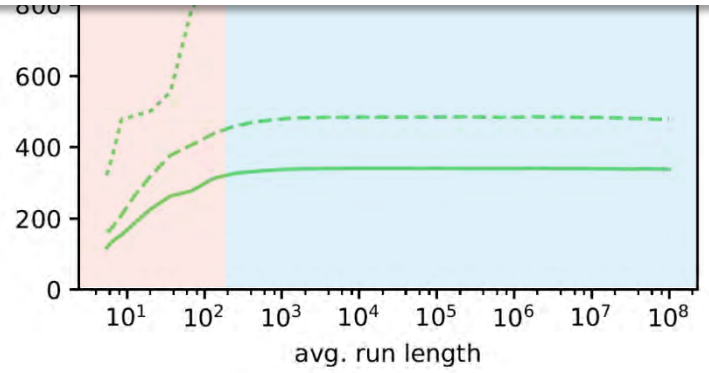
Conflict Detection-based Run-Length Encoding — AVX-512 CD Instruction Set in Action

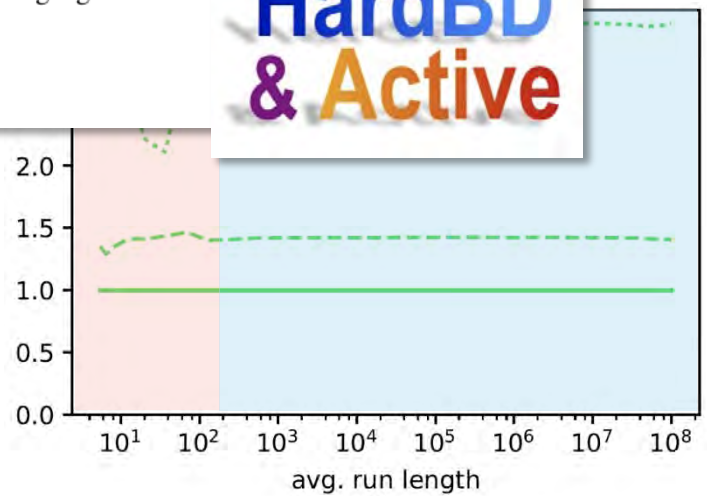
Annett Ungethüm, Johannes Pietrzyk, Patrick Damme, Dirk Habich, Wolfgang Lehner

Database Systems Group, Technische Universität Dresden
Dresden, Germany
{firstname.lastname}@tu-dresden.de

HardBD & Active

ne of 128-bit



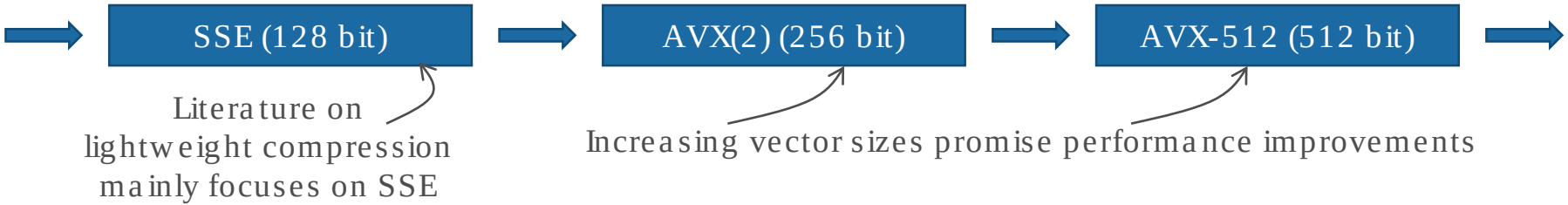


non-well performing area

well-performing area

Conclusion

Dresden Database Systems Group

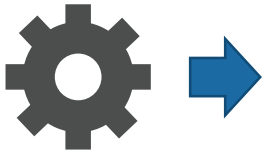


Advantage

- Straightforward porting usually feasible

Disadvantages

- Desired speedups usually not achieved
- Negative effect on compression ratio

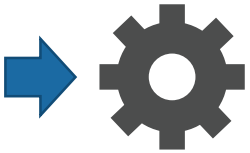


SSE-implementation of some algorithm

Straightforward Port

CONCLUSION

NOT THE RIGHT WAY



AVX-implementation
AVX-512-implementation