Vectorizing Database Column Scans with Complex Predicates

Thomas Willhalm, Ismail Oukid, Ingo Müller, Franz Faerber

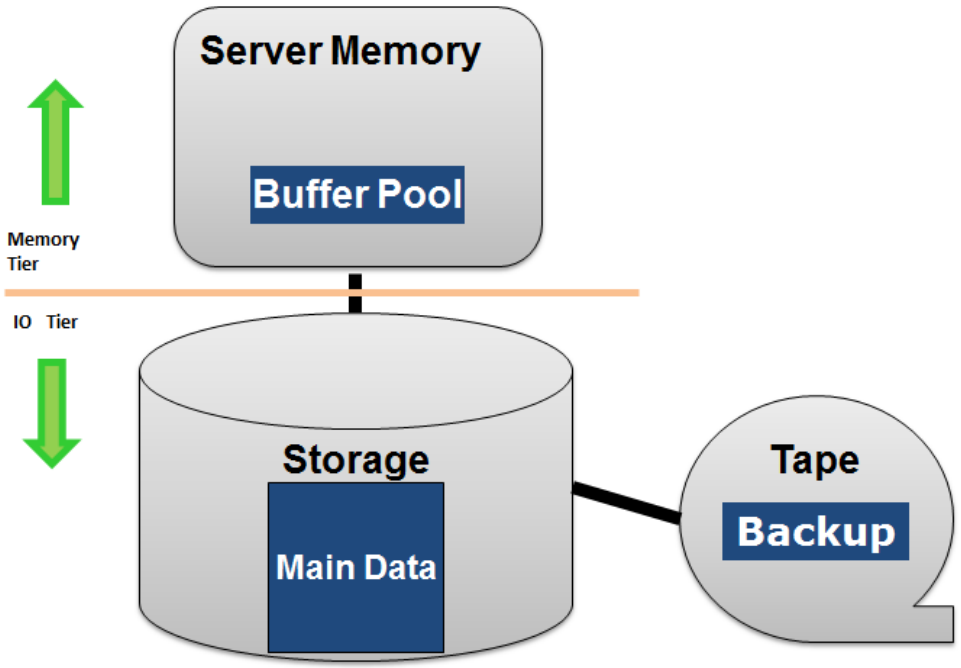
thomas.willhalm@intel.com, i.oukid@sap.com, ingo.mueller@kit.edu, franz.faerber@sap.com

August 26, 2013 / ADMS 2013



Introduction, Context, and Motivation

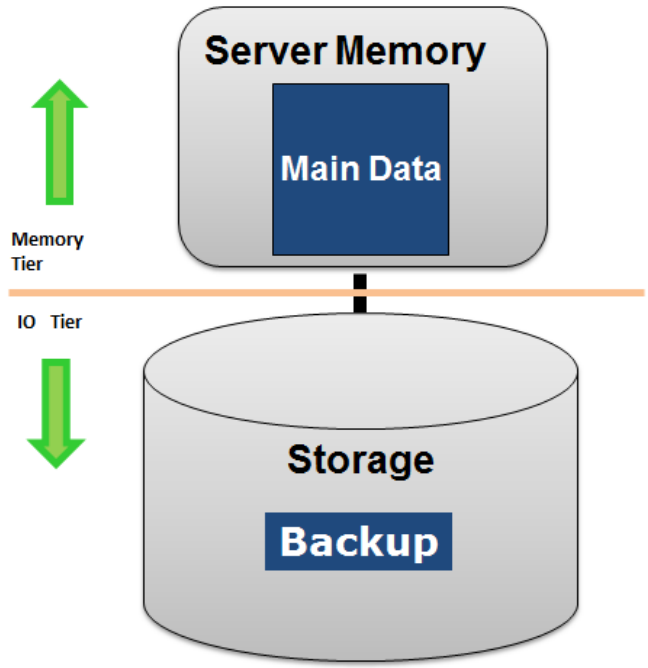
Traditional Architecture



- Memory has become large enough to contain all data.

Introduction, Context, and Motivation

In-memory Architecture



- Keeping data in memory allows faster access to data, overcomes disk latency.
- Backup is kept on storage.

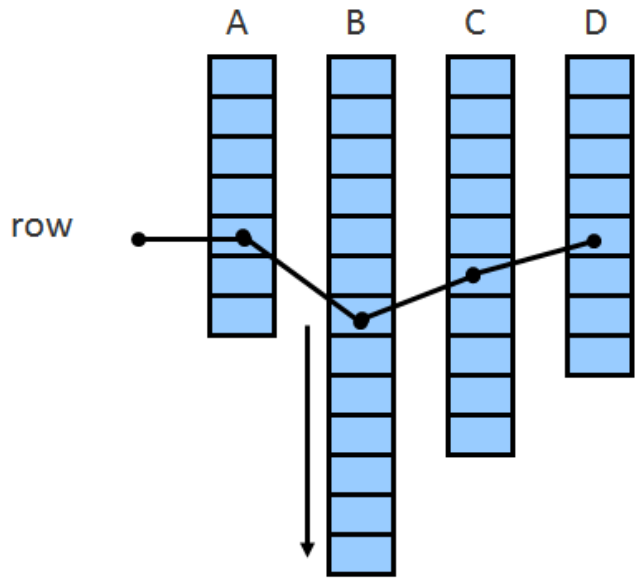
Introduction, Context, and Motivation

Disruptive change in RDBMS industry

- Rise of In-Memory Databases (IMDBs).
- Anticipated by research 10+ years ago.
- All major DB vendors are working on it.
- SAP leading with HANA as platform for all new apps.

Introduction, Context, and Motivation

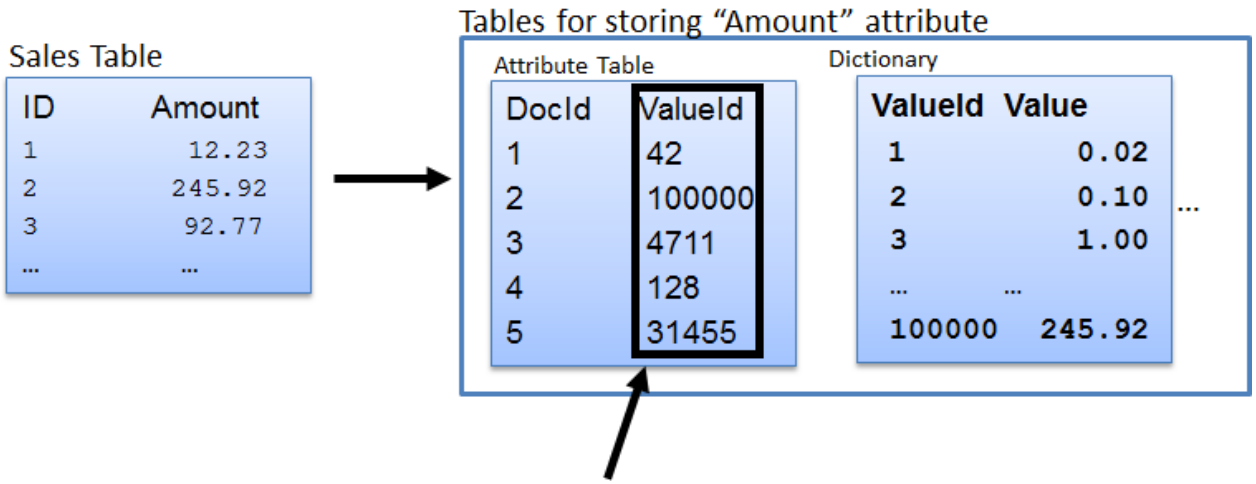
IMDBs often use Column-Orientation



Advantages of column-orientation:

- Columns compressed independently
- Leverages better compression
- Cache-friendly:
  - cache-line granularity
  - prefetching
- SIMD-friendly

Introduction, Context, and Motivation

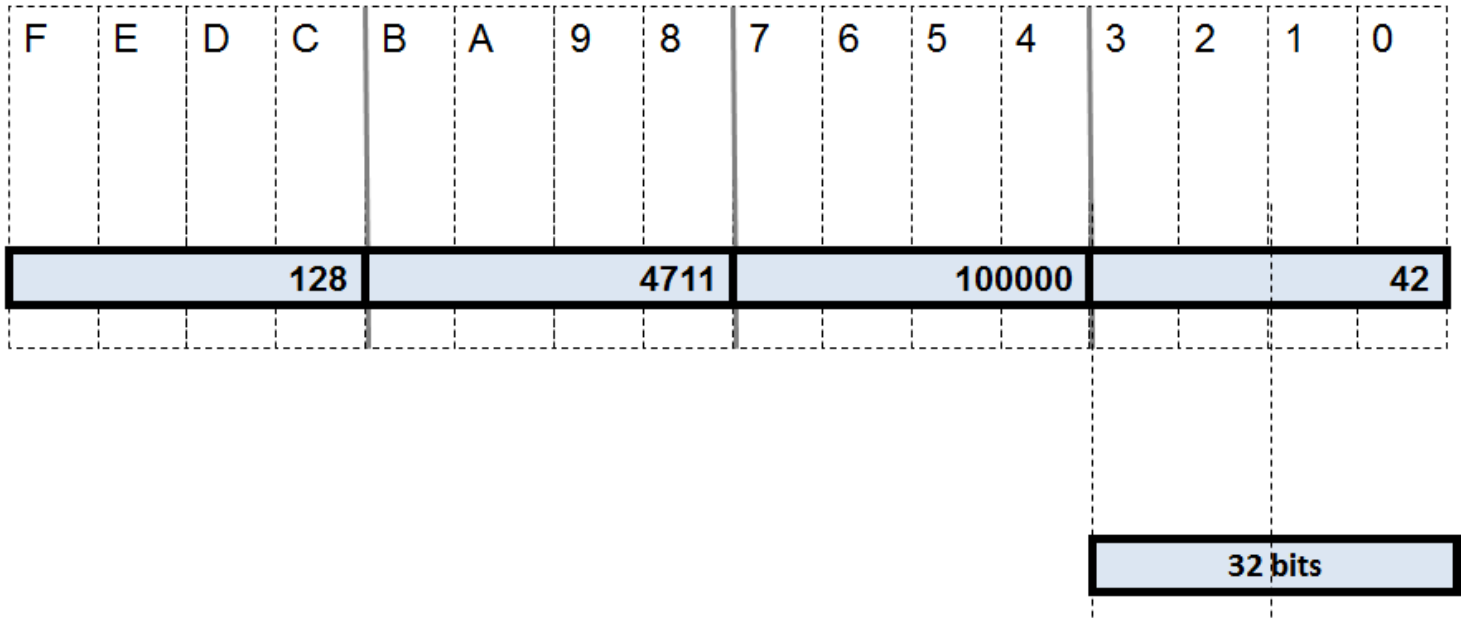


- "Dictionary" contains all distinct values (e.g. 100,000 entries)
- ValueId are integers from 0, 1, 2 ,3, . . . ,100000
- Max is N=100000 (number of distinct values), which needs 17 bits to represent ($[log_2 N] + 1$)
- Idea: instead of 32-bits, use 17-bits fields to store each ValueID. We then call "17" the "Bit-case"
- Accessing "Value" needs decompression into 32-bits

Compression building blocks: Bit-Fields

Packed bit-fields

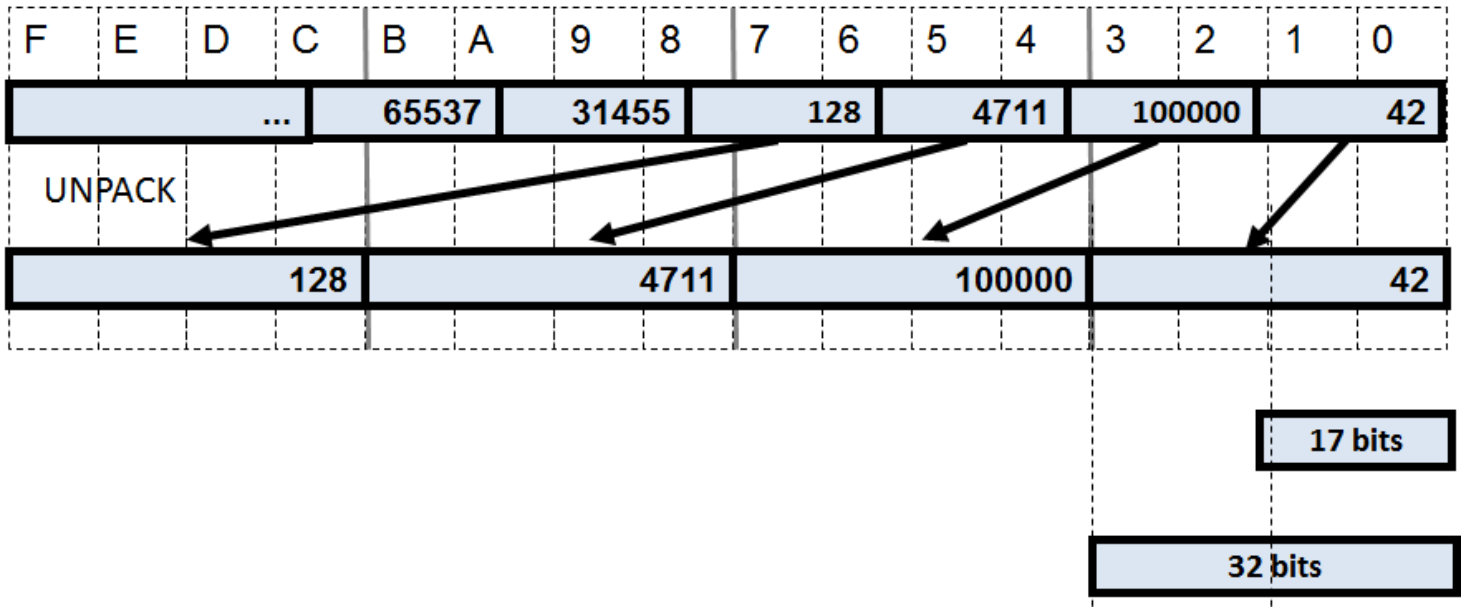
- Large number of integers, each with n number of bits



Compression building blocks: Bit-Fields

Packed bit-fields

- Large number of integers, each with n number of bits
- Example: 17-bit per entry:



32 different implementations for each n from 1 to 32.

Introduction, Context, and Motivation

For each query do a full-table scan, i.e.,

- Decompress required columns,
- Aggregate data according to predicate,
- Further processing.

> Performance of unpacking is key for full-table scans!

Vectorized Scan Framework: Overview

- We propose a framework for vectorized column scans with complex predicates.

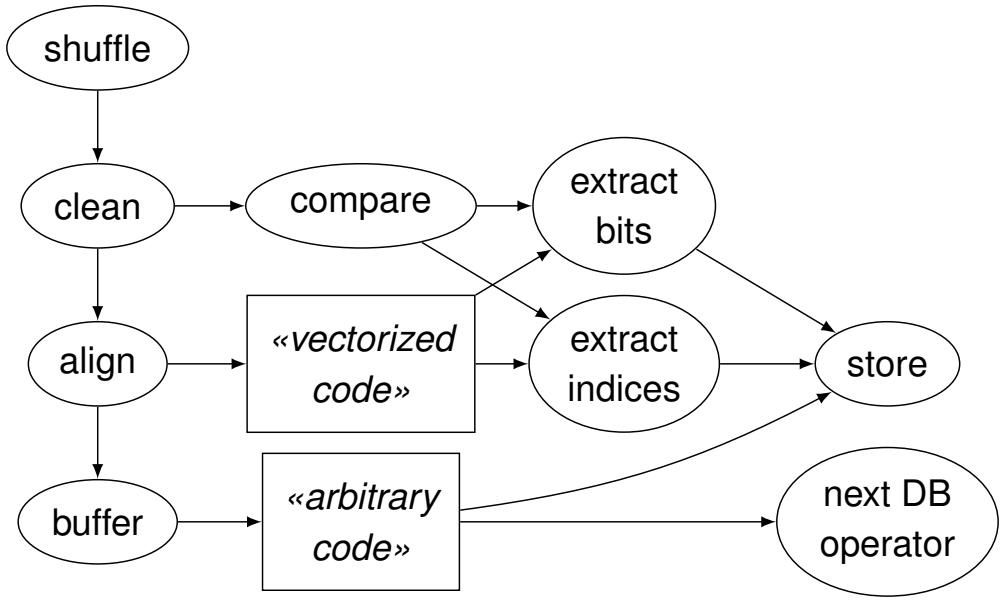
Vectorized Scan Framework: Overview

- We propose a framework for vectorized column scans with complex predicates.
- Different classes of predicates:
  - Range predicates
  - Vectorizable predicates
  - In-list predicates
  - Arbitrary predicates

Vectorized Scan Framework: Overview

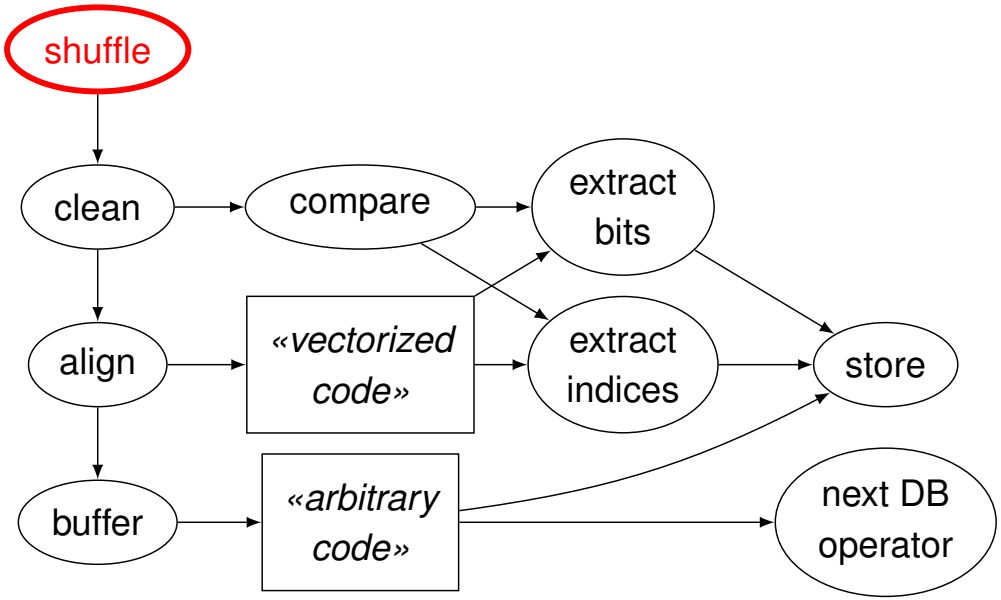
- We propose a framework for vectorized column scans with complex predicates.
- Different classes of predicates:
  - Range predicates
  - Vectorizable predicates
  - In-list predicates
  - Arbitrary predicates
- Different output formats:
  - Bit vectors
  - Index vectors
  - Unpacked data

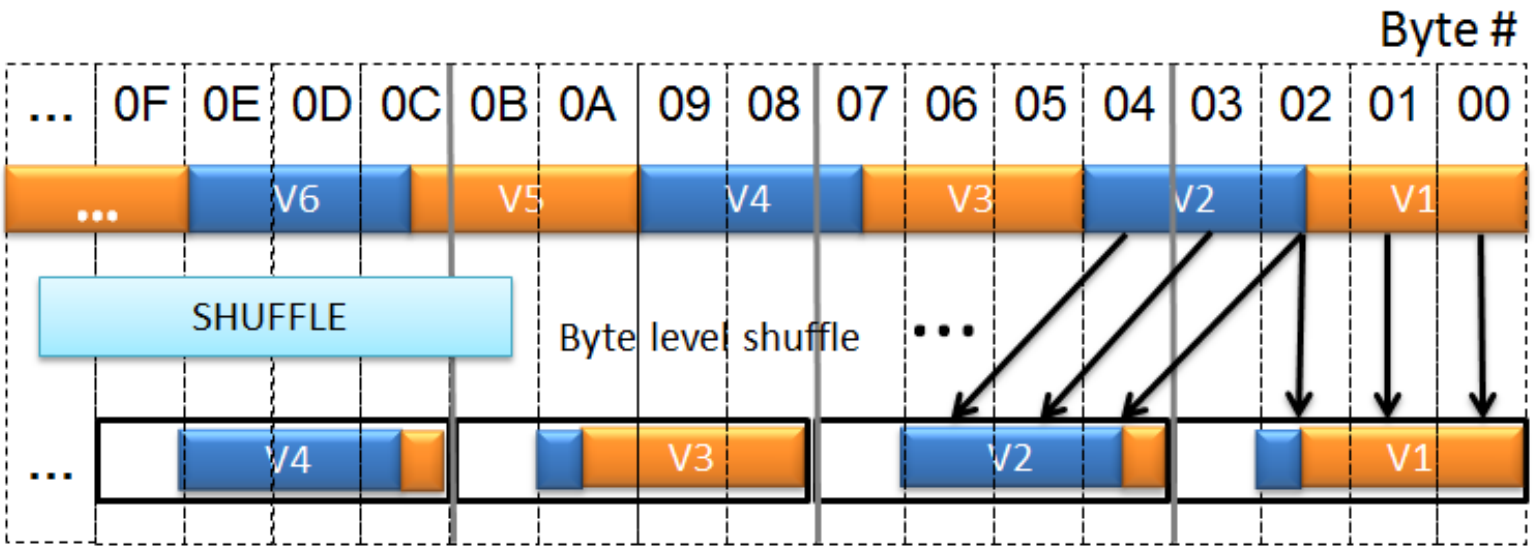
Vectorized Scan Framework: General Scan Algorithm



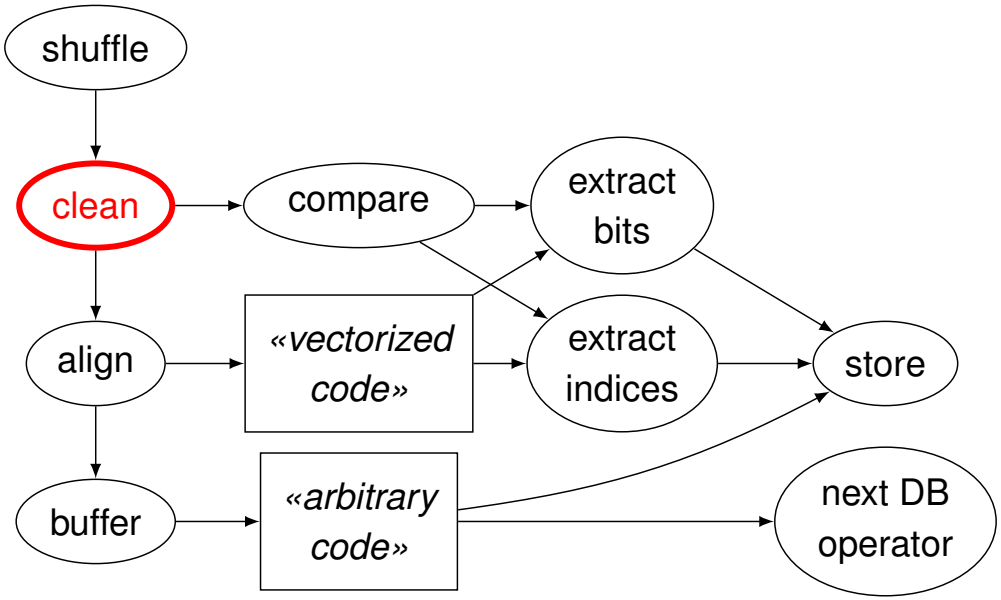
- Common scheme for all predicates and output formats
- Extensible with templates
- Vectorized processing of 8 values at a time (or more)
- Manual optimizations of all paths and all bit cases

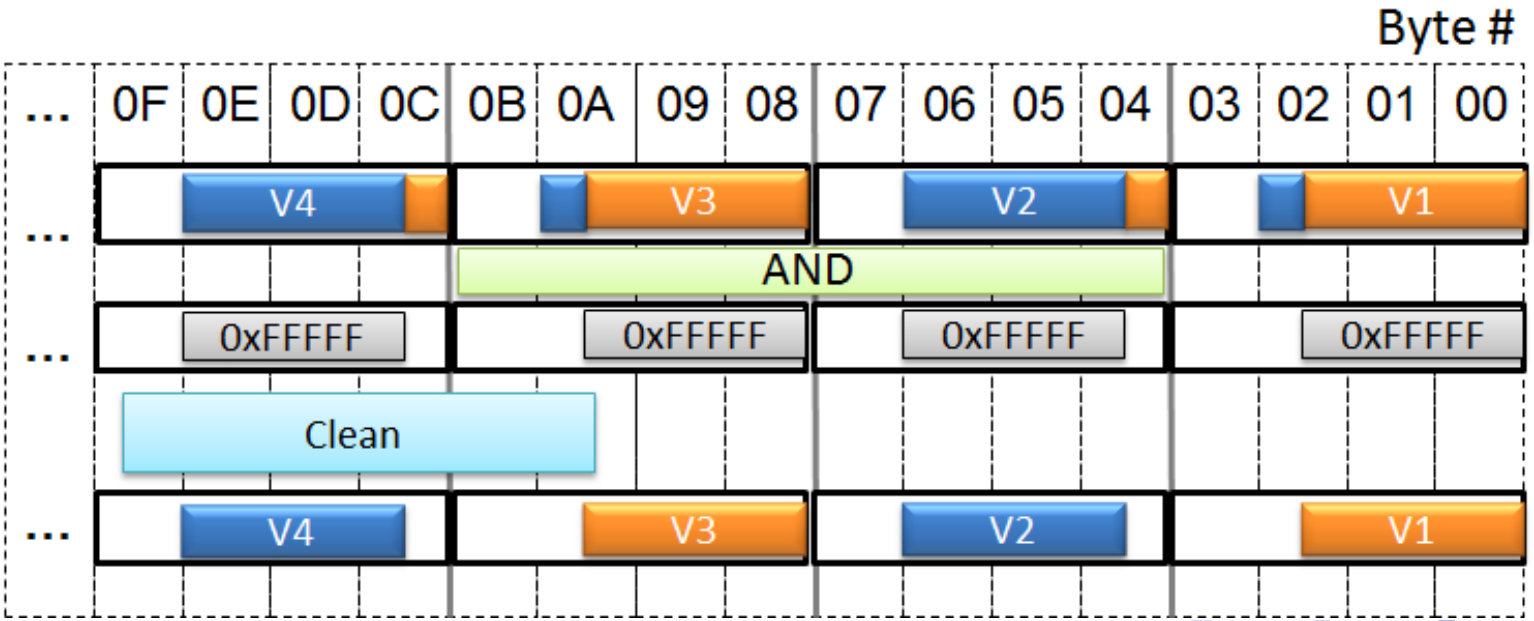
Vectorized Scan Framework: General Scan Algorithm



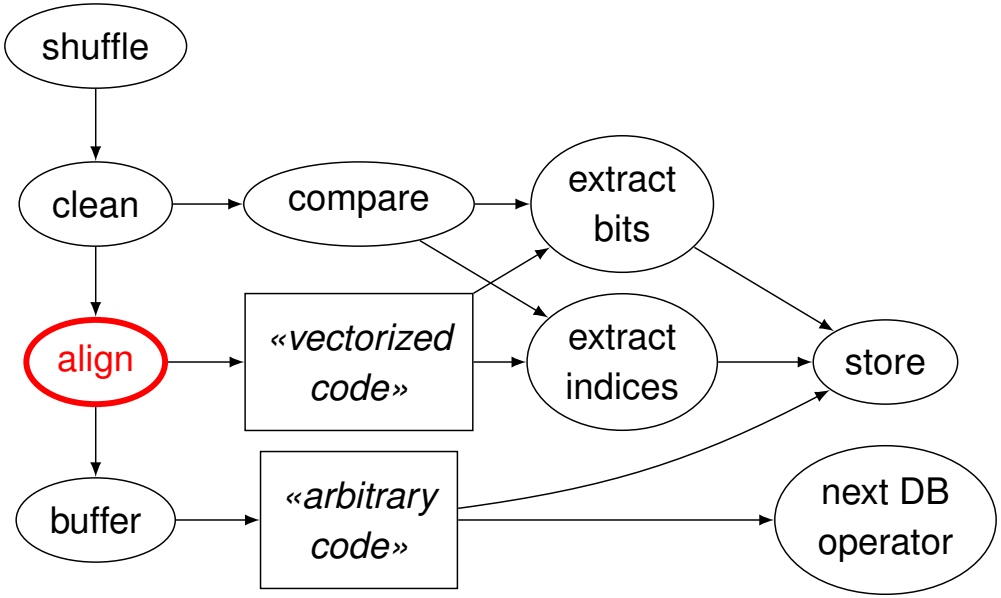


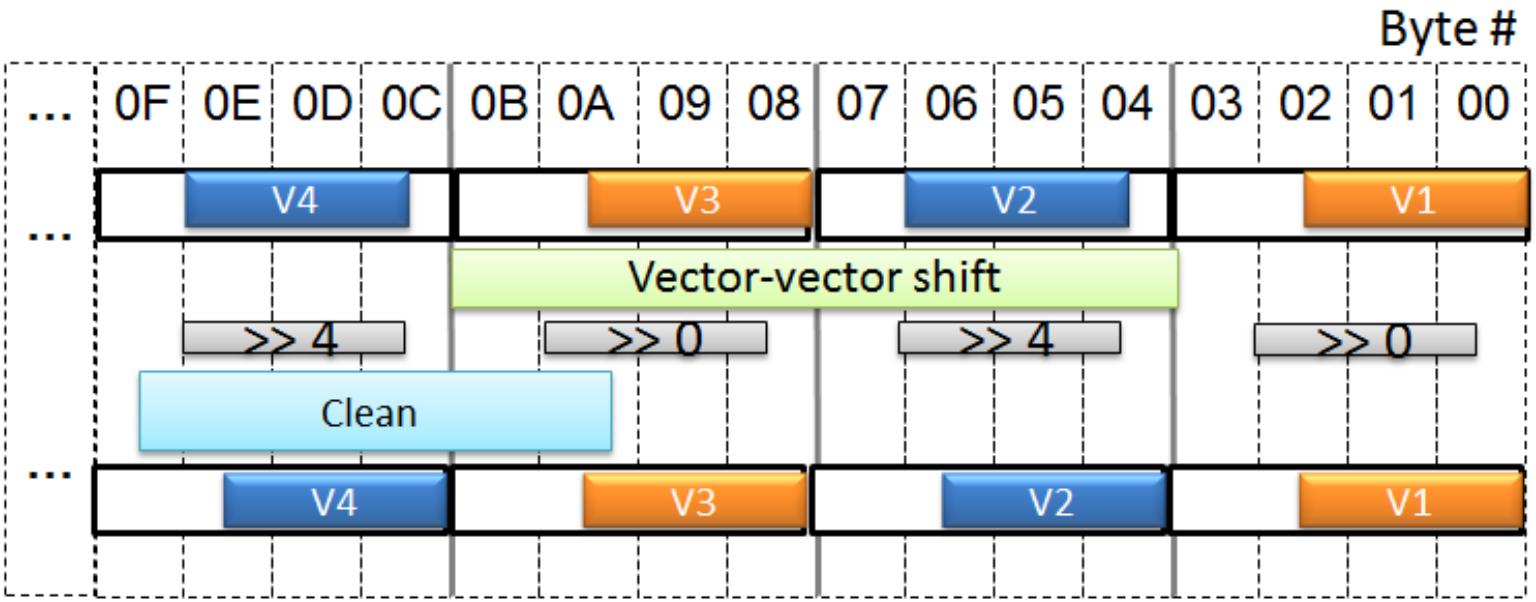
Vectorized Scan Framework: General Scan Algorithm



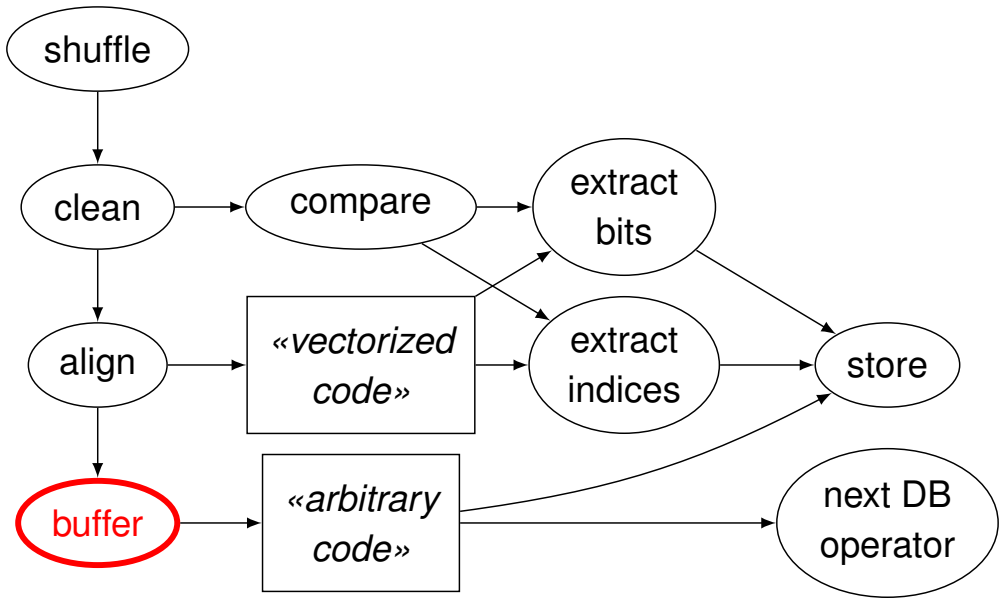


Vectorized Scan Framework: General Scan Algorithm



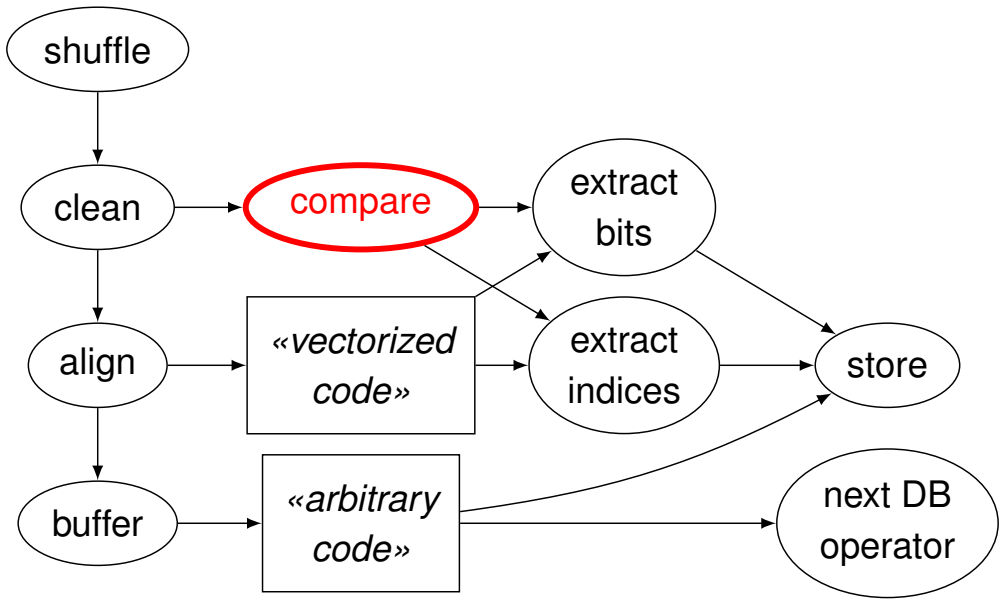


Vectorized Scan Framework: General Scan Algorithm



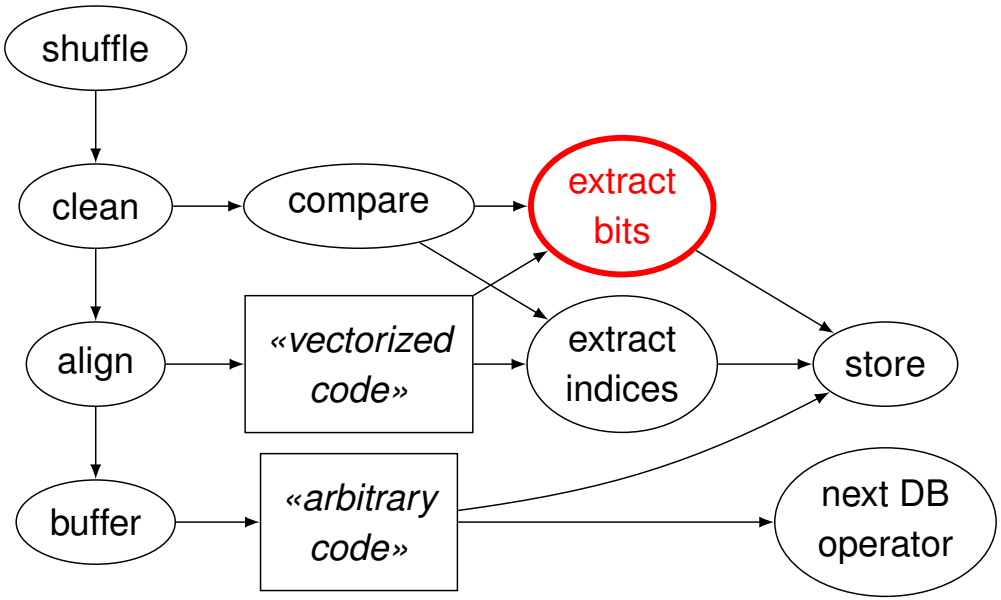
Write to Buffer

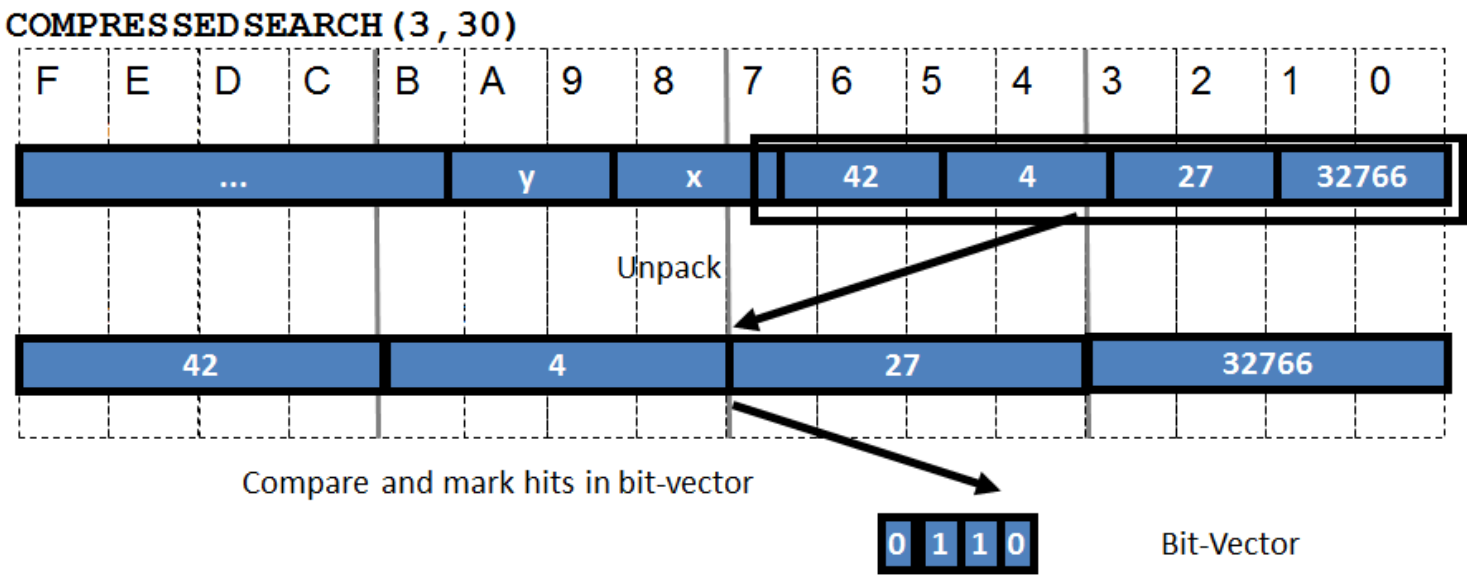
Vectorized Scan Framework: General Scan Algorithm



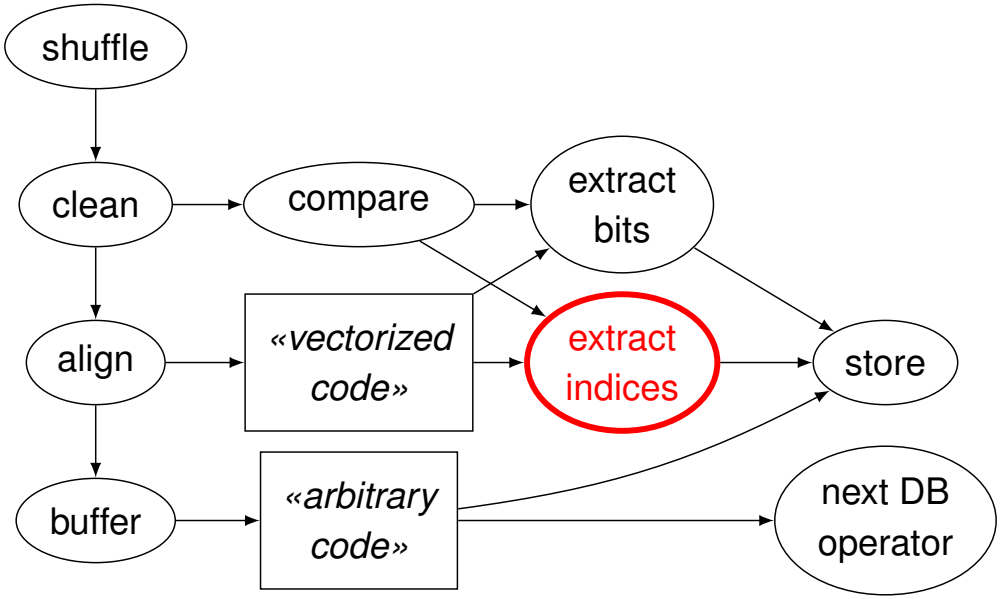
- Range predicates (can express all the equality predicates, i.e., $=$, $\neq$, $\geq$, $>$, $\leq$, $<$)

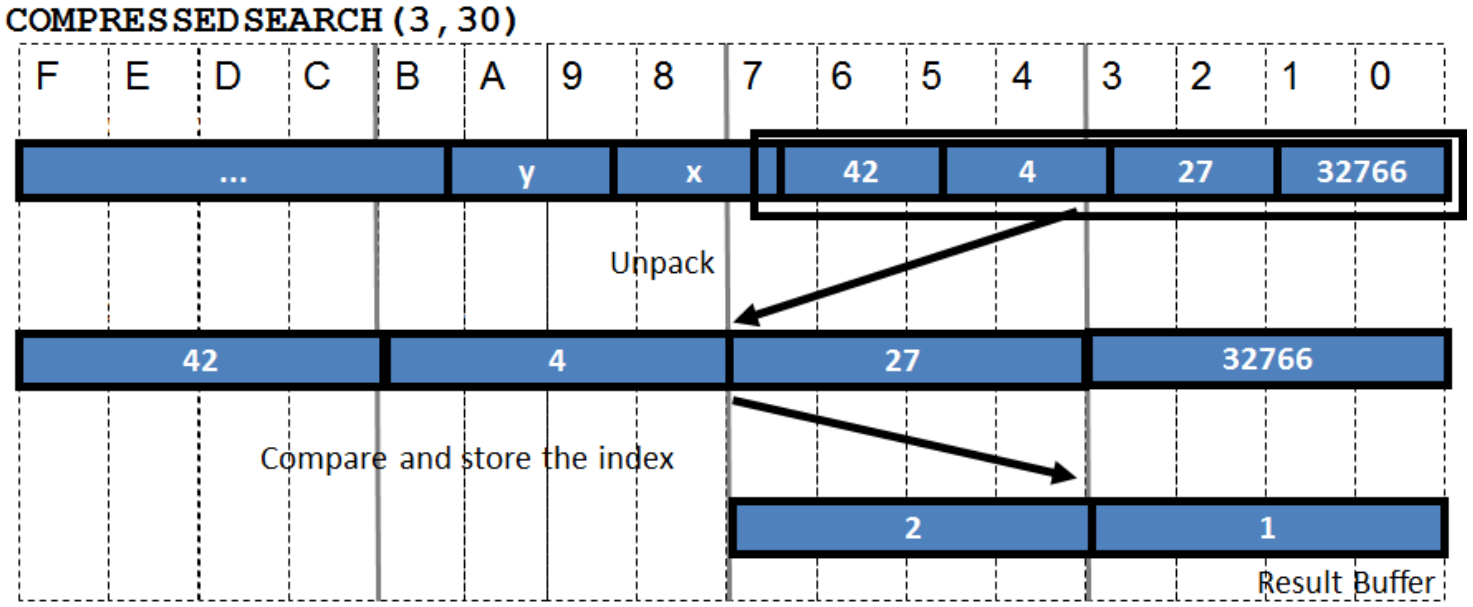
Vectorized Scan Framework: General Scan Algorithm



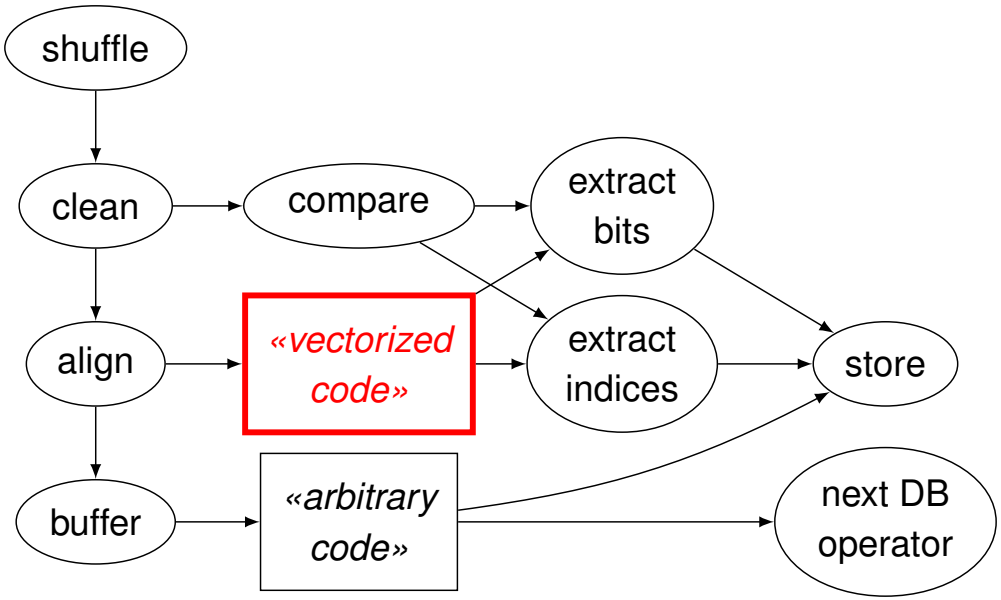


Vectorized Scan Framework: General Scan Algorithm



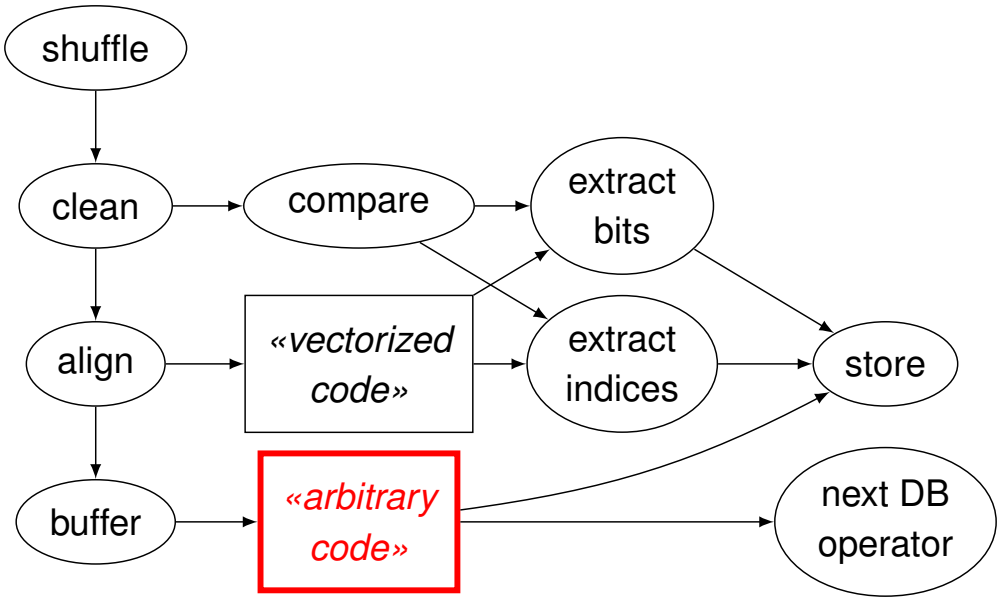


Vectorized Scan Framework: General Scan Algorithm



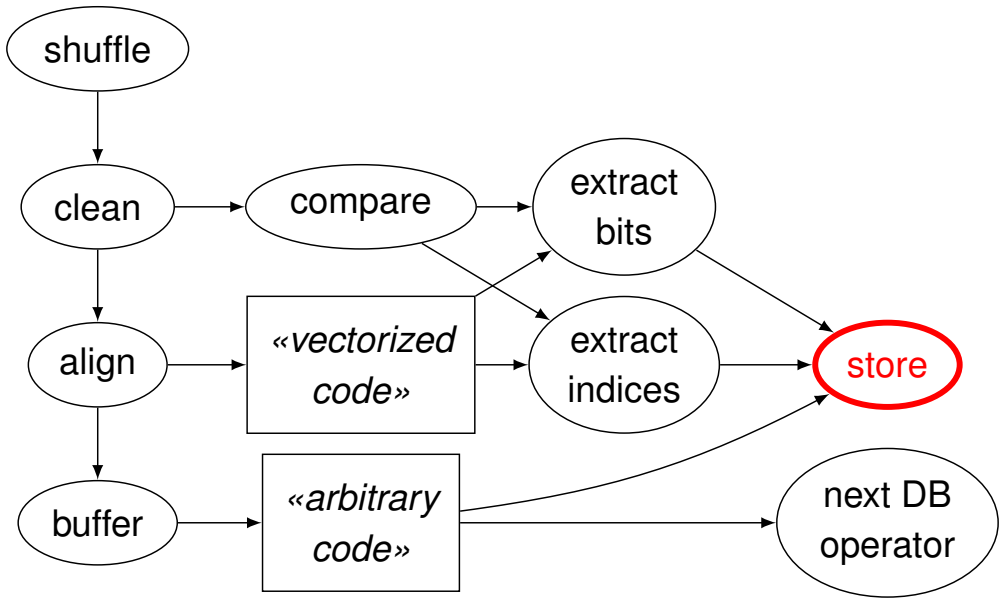
Many other predicates, including many arithmetic expressions on a single column, can easily be expressed using vector instructions.
Example: In-list predicate.

Vectorized Scan Framework: General Scan Algorithm



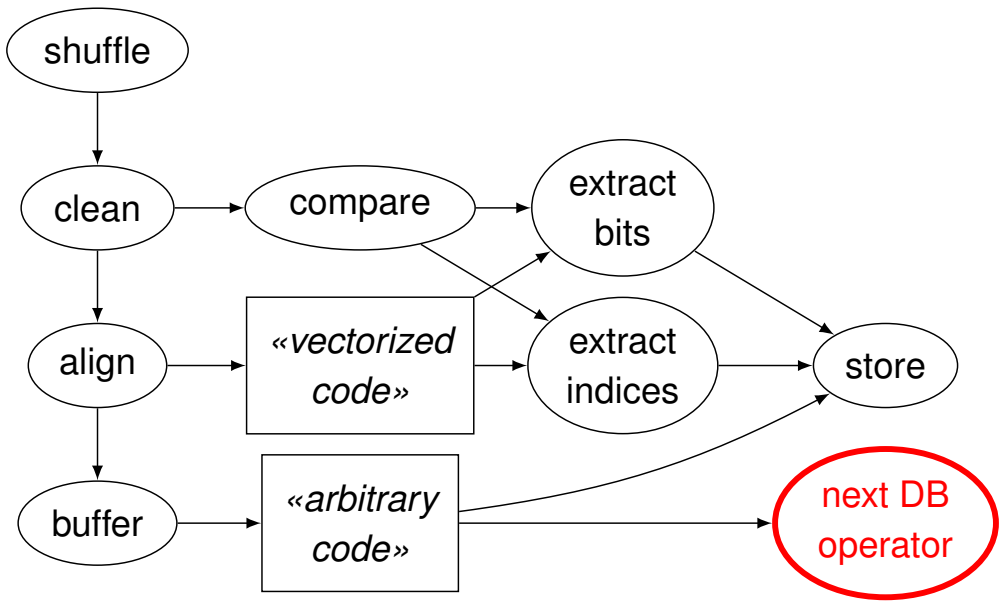
Fall-back mechanism, where a block of codewords is unpacked as machine words into a buffer in cache, on which arbitrary predicates can be applied.

Vectorized Scan Framework: General Scan Algorithm



Store the different classes of results

Vectorized Scan Framework: General Scan Algorithm



Unpack a colunm for the subsequent database operator.

Vectorized Scan Framework: Optimizations

- Optimizations can apply to a group of bit-cases. They can also be specific to one bit-case

Vectorized Scan Framework: Optimizations

- Optimizations can apply to a group of bit-cases. They can also be specific to one bit-case
- Bit-case level optimizations are performance critical

Vectorized Scan Framework: Optimizations

- Optimizations can apply to a group of bit-cases. They can also be specific to one bit-case
- Bit-case level optimizations are performance critical
- Reported 12 optimizations (refer to the paper)

Vectorized Scan Framework: Optimizations

- Optimizations can apply to a group of bit-cases. They can also be specific to one bit-case
- Bit-case level optimizations are performance critical
- Reported 12 optimizations (refer to the paper)
- Optimizations allow to compare up to 32 values in parallel

Vectorized Scan Framework: Optimizations

- Optimizations can apply to a group of bit-cases. They can also be specific to one bit-case
- Bit-case level optimizations are performance critical
- Reported 12 optimizations (refer to the paper)
- Optimizations allow to compare up to 32 values in parallel
- For In-list predicactes: "Permute" and "AvoidGather" are necessary to achieve optimal performance.

Vectorized Scan Framework: Optimizations

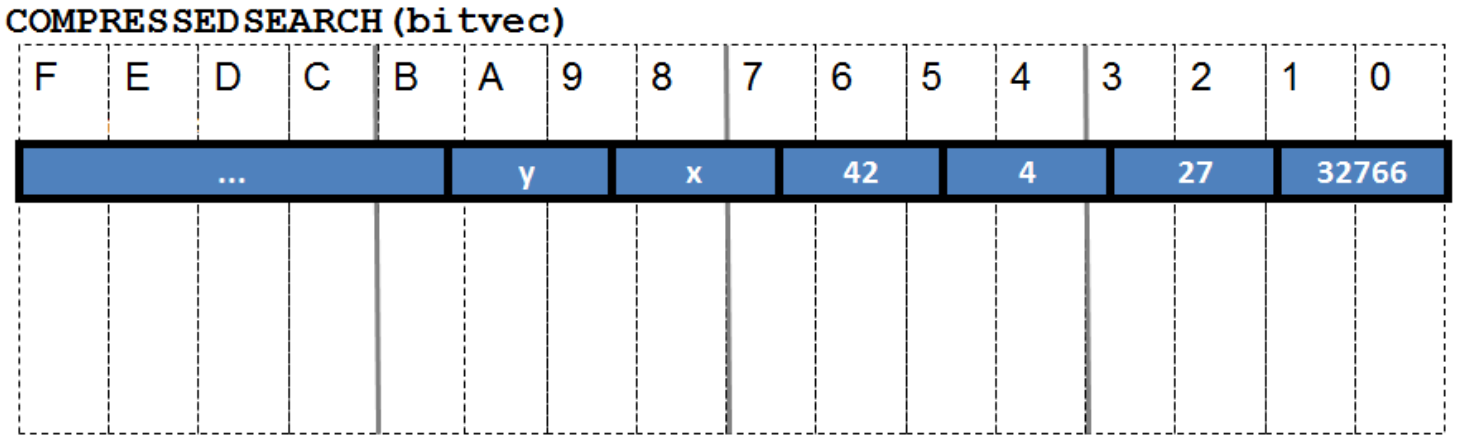
- Optimizations can apply to a group of bit-cases. They can also be specific to one bit-case
- Bit-case level optimizations are performance critical
- Reported 12 optimizations (refer to the paper)
- Optimizations allow to compare up to 32 values in parallel
- For In-list predicactes: "Permute" and "AvoidGather" are necessary to achieve optimal performance.

Example: Scan algorithm for In-list predicate

Vectorized Scan Framework: Optimizations

Search the values that are marked in a bitvector:

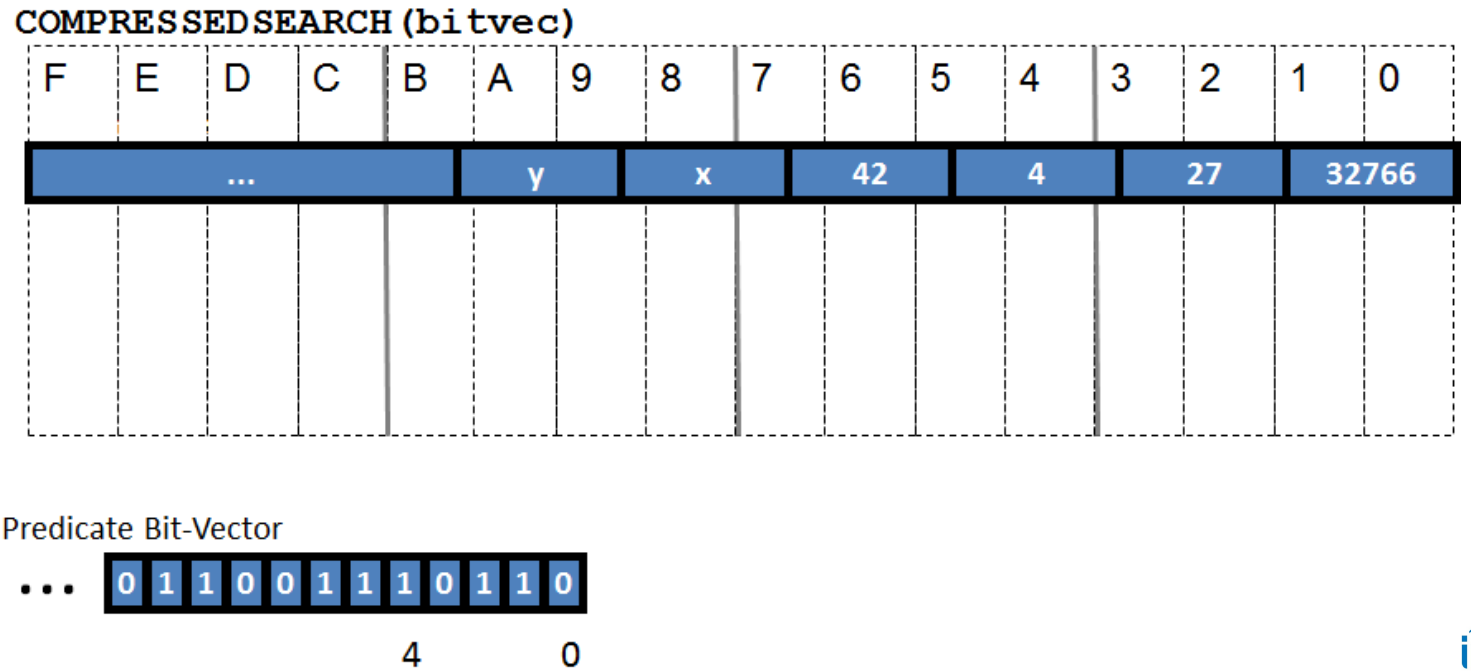
1. UNPACK
2. If corresponding bit is set
3. Mark bit in result bitvector



Vectorized Scan Framework: Optimizations

Search the values that are marked in a bitvector:

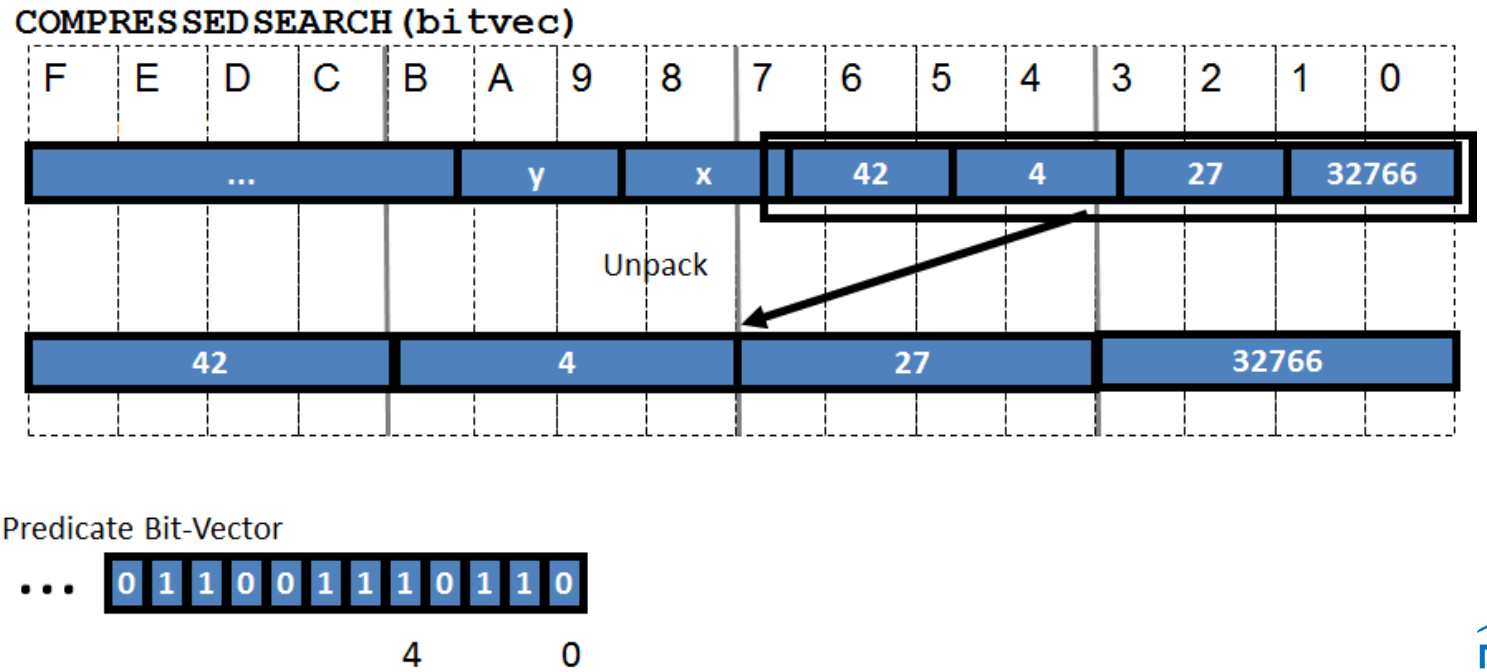
1. UNPACK
2. If corresponding bit is set
3. Mark bit in result bitvector



Vectorized Scan Framework: Optimizations

Search the values that are marked in a bitvector:

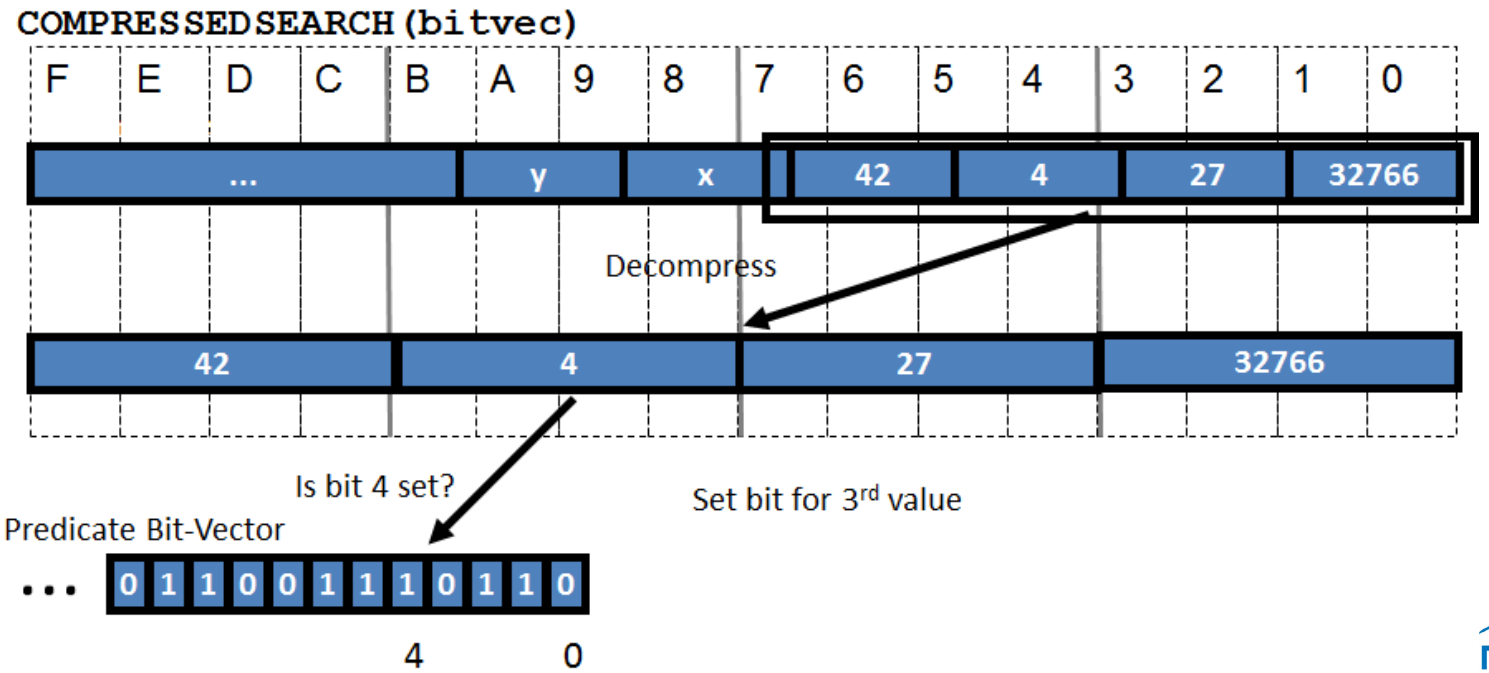
1. UNPACK
2. If corresponding bit is set
3. Mark bit in result bitvector



Vectorized Scan Framework: Optimizations

Search the values that are marked in a bitvector:

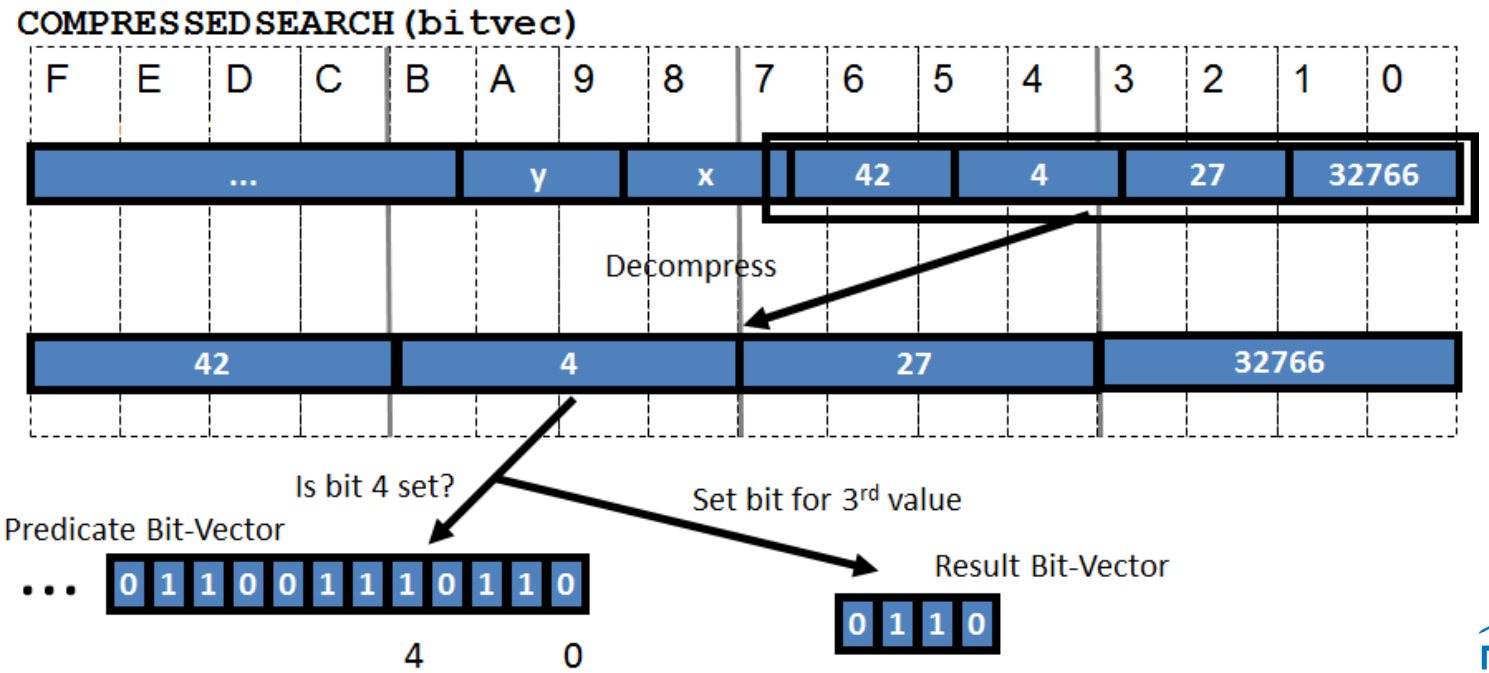
1. UNPACK
2. If corresponding bit is set
3. Mark bit in result bitvector



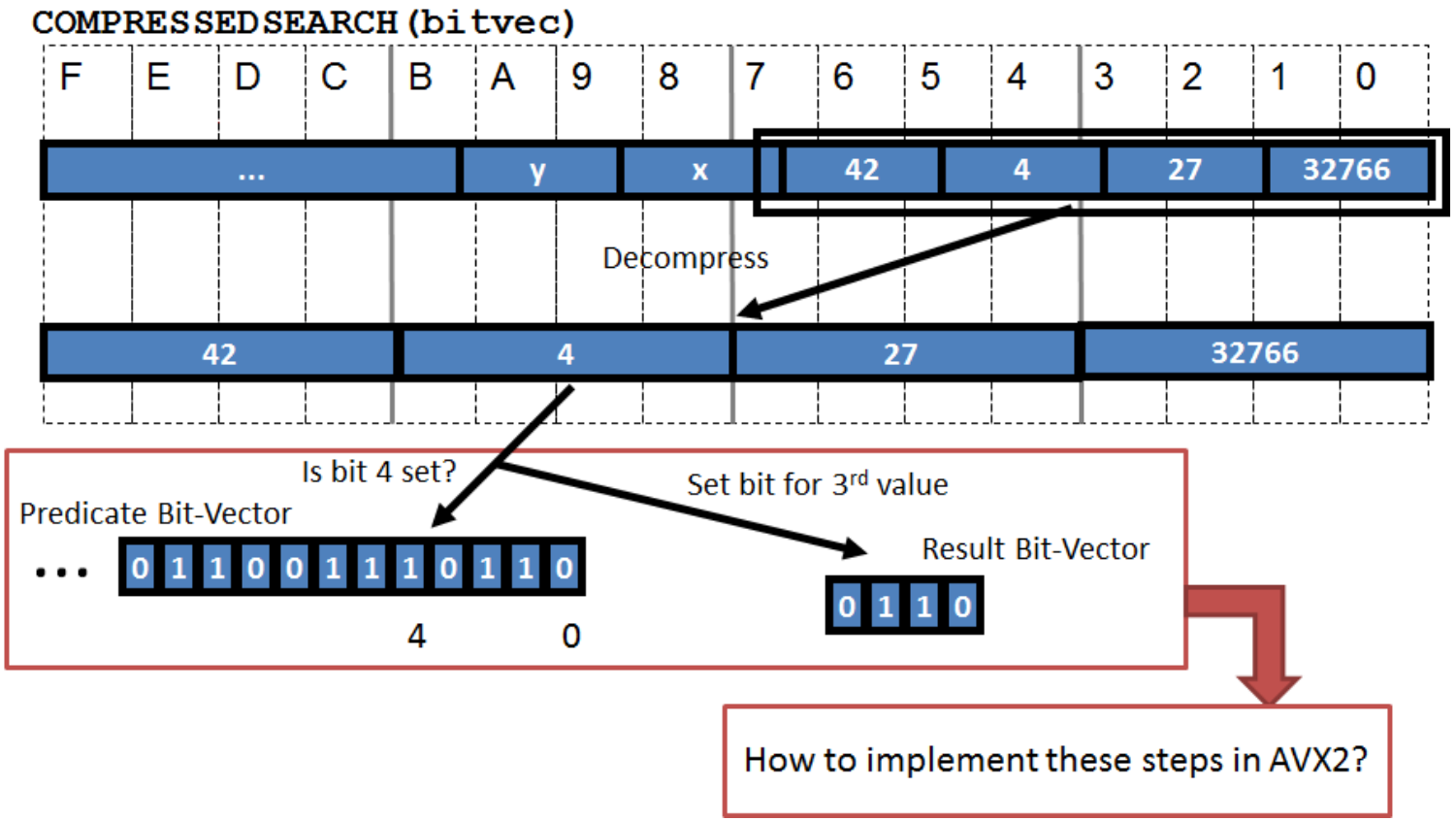
Vectorized Scan Framework: Optimizations

Search the values that are marked in a bitvector:

1. UNPACK
2. If corresponding bit is set
3. Mark bit in result bitvector



Vectorized Scan Framework: Optimizations

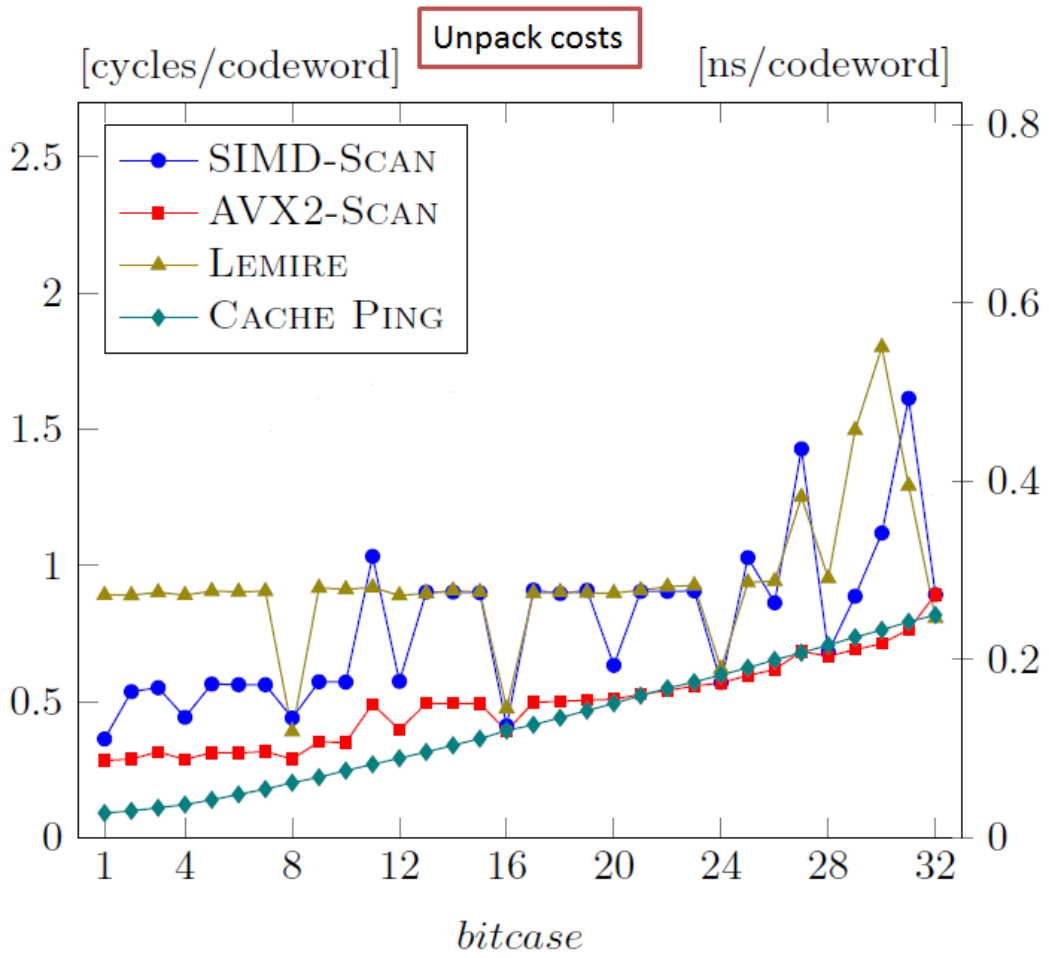


Vectorized Scan Framework: Optimizations

Two AVX2 instructions are key to achieve the parallelization of the Scan with in-list predicate algorithm:

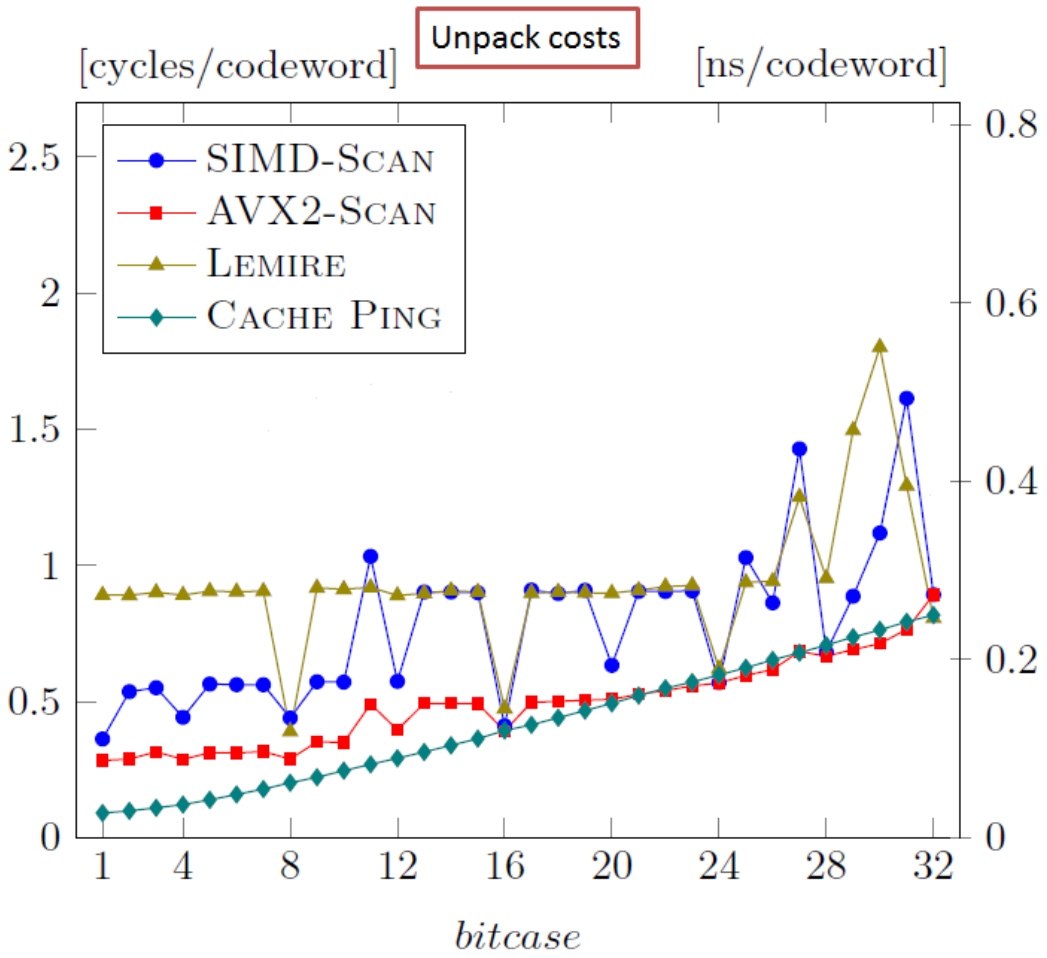
- Vector-vector shift instruction: The new vector-vector shift instructions allows shifting each word of the AVX register with an independant value. We use it to convert the values into words where only the bit at the index equal to the value is set to 1.
- Gather instruction: The new gather instructions loads elements from memory based on a base address and offsets for each data element. We use it to gather the different chunks of the predicate relevant to vectorized comparison.

Evaluation: Unpack Performance



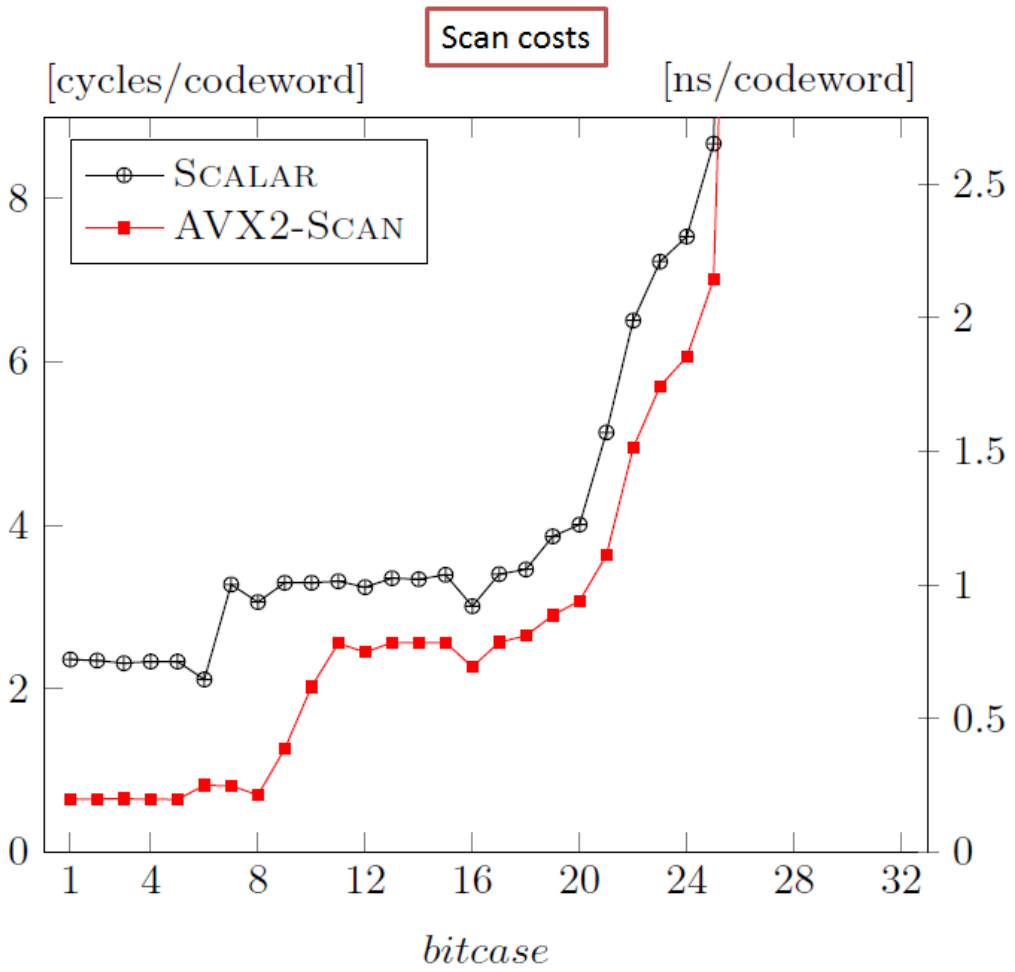
- Bit cases have different performances due to different optimizations.
- Example: bit case 16 is trivially easy.
- AVX2-Scan is 30% faster than SIMD-Scan
- Bit cases >16 are memory bound.

Evaluation: Unpack Performance



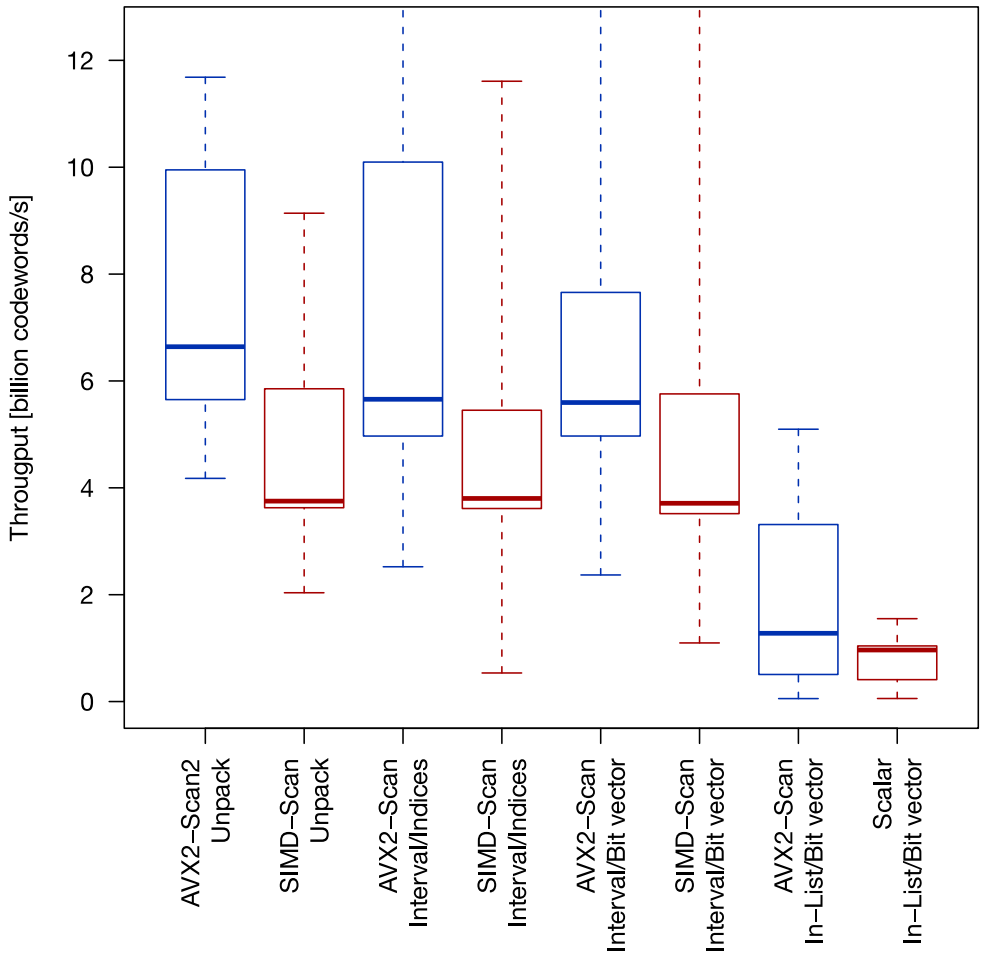
- Lemire's reimplemation of SIMD-Scan often matches our performance, but we found additional optimizations.
- Li's reimplemantation needs 5 cycles/codeword.

Evaluation: Scan Performance



- Optimizations "Permute" and "AvoidGather" work very well for bitcases <= 8.
- In these cases 4 times faster than Scalar.
- Bit cases >21 and >26 get penalties from L2 and L3 cache misses for the bit-vector predicate.

Evaluation: Throughtput Overview



- AVX2-Scan consistently 30% faster than SIMD-Scan.
- Throughputs between 4 and 10 billion codewords per second with peaks of 17 billion.

Conclusion and Outlook

- Our scan framework, in particular the AVX2-Scan implementation, is an effective means to improve scan performance.

Conclusion and Outlook

- Our scan framework, in particular the AVX2-Scan implementation, is an effective means to improve scan performance.
- Intel has recently released the description of Intel AVX-512. Most notable additions:

Conclusion and Outlook

- Our scan framework, in particular the AVX2-Scan implementation, is an effective means to improve scan performance.
- Intel has recently released the description of Intel AVX-512. Most notable additions:
  - 512 bit registers.
  - Mask registers.
  - Cross-lane shuffles.
  - Compress instruction.
  - Unsigned comparison.

Conclusion and Outlook

- Our scan framework, in particular the AVX2-Scan implementation, is an effective means to improve scan performance.
- Intel has recently released the description of Intel AVX-512. Most notable additions:
  - 512 bit registers.
  - Mask registers.
  - Cross-lane shuffles.
  - Compress instruction.
  - Unsigned comparison.
- It would be interesting to compare our approach with GPUs.