Compressed Data Structures for Query Processing

Compressed Bitmap Representations

Anna Bolotina

University of Salzburg

Advanced Topics in Data Analytics
Winter semester 2026/27

When simple RLE becomes expensive

Raw baseline: $N = 1{,}024$ bits.

Pattern	Stored runs	RLE bits
512 zeros, then 512 ones	2	$2 \times 17 = 34$
Alternating 0 and 1	1,024	$1{,}024 \times 17 = 17{,}408$

Mixed regions

Many short runs can cost more than storing the bits directly.

Simplified RLE model: 1 value bit + 16 length bits per run (lengths 1–65,535). Metadata excluded.

Operations on a bitmap representation

For the same universe $\{0, \ldots, N-1\}$:

- **Store** membership compactly.
- **Combine** predicates with AND and OR.
- **Count** the matching positions.
- **Iterate** over matching positions.

Separate costs

Counting matches and producing all matching positions can require different work.

WAH: literals and fills

Word-Aligned Hybrid (WAH) groups the original bits.

Original groups	Encoded word	Kind	Meaning
1010101	0 1010101	Literal	Keep seven bits
0000000 0000000	1 0 000010	Fill	Two zero groups
1111111 1111111	1 1 000010	Fill	Two one groups

Leading flag: 0 = literal, 1 = fill.

A literal keeps the bits. A fill stores a value and a group count.

Simplified model: 8-bit encoded words, 7 original bits per group. Spaces separate fields.

[1] FastBit: WAH compression.

WAH literal words

Flag: 1 bit	Payload: 7 bits
0	1010101

Encoded word: 01010101

- Flag 0 identifies a literal.
- Its payload represents **seven original positions**.

32-bit WAH

One encoded literal word holds a flag and **31 original bits**.

[1] FastBit: WAH compression.

WAH fill words

Flag: 1 bit	Value: 1 bit	Group counter: 6 bits
1	0	000010

10000010 represents $\underbrace{0000000\ 0000000}_{\text{2 groups} = 14 \text{ positions}}$

- The value selects all-zero or all-one groups.
- The counter measures **groups**, each with seven bits here.

32-bit WAH: flag + value + 30-bit counter, counting 31-bit groups.

[1] FastBit: WAH compression.

WAH encoding: a complete example

Input: 56 positions in eight 7-bit groups

```
0000000 0000000 0000000 0000000 1010101 1111111 1111111 1111111
```

Simplified model. Encoded size excludes length metadata.

WAH encoding: a complete example

Input: 56 positions in eight 7-bit groups

`0000000 0000000 0000000 0000000 1010101 1111111 1111111 1111111`

Groups	Encoded word	Decoding
1–4	1 0 000100	4 all-zero groups

Simplified model. Encoded size excludes length metadata.

WAH encoding: a complete example

Input: 56 positions in eight 7-bit groups

```
0000000 0000000 0000000 0000000 1010101 1111111 1111111 1111111
```

Groups	Encoded word	Decoding
1–4	1 0 000100	4 all-zero groups
5	0 1010101	1 literal group

Simplified model. Encoded size excludes length metadata.

WAH encoding: a complete example

Input: 56 positions in eight 7-bit groups

```
0000000 0000000 0000000 0000000 1010101 1111111 1111111 1111111
```

Groups	Encoded word	Decoding
1–4	1 0 000100	4 all-zero groups
5	0 1010101	1 literal group
6–8	1 1 000011	3 all-one groups

Encoded size: 3 words $\times$ 8 bits $=$ 24 bits.

Original bitmap: 56 bits.

Simplified model. Encoded size excludes length metadata.

AND with WAH literals and fills

One aligned 7-bit group at a time

Left input	Right input	AND result
Zero-fill group	1010101	Zero-fill group
One-fill group	1010101	1010101
1010101	1100011	1000001

Different fill lengths

Consume the smaller remaining group count, then continue with the remainder.

Result encoding may merge adjacent uniform groups into fills.

Exercise: WAH encoding

Encode these 42 positions with the simplified 8-bit model:

```
0000000  0000001  1111111  1111111  0011001  0000000
```

1. Which groups can share a fill?
2. Can a fill include the last bit of the second group?
3. Write the encoded words. What does each counter count?

Exercise: WAH encoding

Encode these 42 positions with the simplified 8-bit model:

```
0000000  0000001  1111111  1111111  0011001  0000000
```

1. Which groups can share a fill?
2. Can a fill include the last bit of the second group?
3. Write the encoded words. What does each counter count?

One encoding

`1 0 000001  0 0000001  1 1 000010  0 0011001  1 0 000001`

0000001 is a literal. Fills cover whole 7-bit groups.

EWAH: a marker before the literals

Enhanced Word-Aligned Hybrid (EWAH) uses fills and literals.

- WAH: each encoded word has its own flag.
- EWAH: a separate **marker** describes a fill and the literals that follow.

Example of a marker instruction

"Write two all-zero words, then copy the next two words as literals."

The literal words need no flag: the marker tells us how many to read.

64-bit EWAH here: each literal holds 64 original bits.

[2] Git bitmap format, Appendix A.

Reading an EWAH sequence

Original bitmap: five 64-bit words

All zeros	All zeros	L_1	L_2	All ones

L_1 and L_2 contain mixed bits, so we keep them as literals.

[2] Git bitmap format, Appendix A.

Reading an EWAH sequence

Original bitmap: five 64-bit words

All zeros	All zeros	L_1	L_2	All ones

L_1 and L_2 contain mixed bits, so we keep them as literals.

$$\underbrace{\text{Marker 1} \quad L_1 \quad L_2}_{\text{first chunk}} \qquad \text{Marker 2}$$

Marker	Instruction
1	Write 2 zero words, then copy 2 literals (L_1, L_2).
2	Write 1 all-one word, then copy 0 literals.

After L_1 and L_2, the decoder expects another marker.

[2] Git bitmap format, Appendix A.

Inside a 64-bit EWAH marker

The example marker $H(B, K, M) = H(0, 2, 2)$ occupies one word:

M: 31 bits	K: 32 bits	B: 1 bit
Literal count	Uniform-word count	Repeated bit
2	2	0
Bits 63–33	Bits 32–1	Bit 0

Binary encoding (high bits first):

```
0000000000000000000000000000010 00000000000000000000000000000010 0
```

Spaces separate fields. Each counter includes leading zeros to fill its field.

Write **2 all-zero words**, then copy **2 literal words** from the stream.

[2] Git bitmap format, Appendix A.

Example of an EWAH operation: AND

Encoded inputs:

$A: \ H(0,2,1)\ L_1 \qquad B: \ H(1,1,2)\ L_2\ L_3$

	Word 1	Word 2	Word 3
A	0^{64}	0^{64}	L_1
B	1^{64}	L_2	L_3
$A \text{ AND } B$	0^{64}	0^{64}	$R = L_1 \text{ AND } L_3$

Columns show logical 64-bit words. 0^{64} means all zeros, 1^{64} all ones.

Encoded result: $H(0,2,1)\ R$

Zero fills produce zeros. Literal pairs use wordwise AND.

[2] Git bitmap format, Appendix A.

WAH and EWAH: size and query time

Example: 126 alternating bits, both formats using 64-bit words

Format	Encoded content	Size
WAH	2 literals, each holding 63 input bits	128 bits
EWAH	1 marker + 2 full literal words	192 bits

- EWAH stores 64 data bits per literal, but adds a marker. Longer literal sequences amortize that overhead.
- The smaller representation need not give the faster query. Data patterns and implementation also matter.

Calculated payload sizes. The final EWAH word has 2 padding bits. External metadata excluded.

Format layouts: [1] FastBit WAH, [2] Git EWAH.

Break

10 minutes

Next: Roaring and local container choices.

Different regions within one bitmap

Region	Illustrative pattern
Sparse	0001000000000010...
Dense, fragmented	1110111101111011...
Clustered	0000000011111111...

Roaring

Partition the universe into blocks and choose a representation locally.

Short excerpts illustrate patterns. Actual Roaring blocks have 2^{16} positions.

Bitmap positions as a set of integers

Position	0	1	2	3	4	5	6	7
Bit	0	0	1	0	0	1	0	1

$$S = \{i : B[i] = 1\} = \{2, 5, 7\}$$

- Positions use **zero-based indexing**.
- These integers need not equal original record IDs.
- Keep N when the finite universe matters, for example for NOT.

Block number and local position

Divide positions into blocks of **65,536**. Number both from zero.

Block	Global positions
0	0–65,535
1	65,536–131,071

Where is position 65,539?

Block 1 starts at 65,536.

Local position: $65{,}539 - 65{,}536 = 3$.

$$x = h \times 65{,}536 + \ell \qquad (h\text{: block, } \ell\text{: local position})$$

32-bit position: high 16 bits $= h$, low 16 bits $= \ell$.

Roaring stores no container for a block containing only zeros.

Array containers

$x = h \times 65{,}536 + \ell$: one container stores the low parts for key h.

Sparse block, key $h = 1$

Global positions: $\{65{,}539, 65{,}545, 65{,}636\}$

Sorted local values: $\{3, 9, 100\}$

- Store each local value as an unsigned 16-bit integer.
- For k values: $2k$ bytes of value payload.
- This example: $2 \times 3 = 6$ bytes before metadata.

[3] Roaring format specification.

Bitmap containers

$x = h \times 65{,}536 + \ell$: the local bit index is ℓ.

One block, key $h = 1$

A local bitmap has one bit for every $\ell \in \{0, \ldots, 65{,}535\}$.

Global position 65,539 sets local bit 3.

$$\text{Bitmap payload} = 65{,}536 \text{ bits} = 8{,}192 \text{ bytes}$$

For k present values (k set bits):

$$\text{Bits per present value} = \frac{65{,}536}{k}$$

More present values share the same fixed bitmap payload.

For a bitmap container, $k > 0$. Headers and other metadata excluded.

[3] Roaring format specification.

Run containers

Present local positions: $\{3, 4, 5, 6, 7, 20, 21\}$

Inclusive range	Stored start (2 B)	Stored length -1 (2 B)
$[3, 7]$	3	$5 - 1 = 4$
$[20, 21]$	20	$2 - 1 = 1$

Stored pairs: $(3, 4)$ and $(20, 1)$. Zero gaps stay implicit.

Run count: 2, stored in 2 bytes.

It tells the decoder how many pairs to read.

$$\text{Size} = 2 + 4r = 2 + 4 \times 2 = 10 \text{ bytes}$$

Lecture 2 RLE: zero- and one-runs, 16-bit counters. See Lemire et al., Section 3 for the 32-bit variant.

[3] Roaring format specification.

Array, bitmap and run container sizes

The same local values, stored in three different ways

Universe size	Present values (k)	Local positions	Array (B)	Bitmap (B)	Run (B)
65,536	100	$0, 2, \ldots, 198$	200	8,192	402
65,536	4,096	$0, 2, \ldots, 8{,}190$	8,192	8,192	16,386
65,536	10,000	$0, 2, \ldots, 19{,}998$	20,000	8,192	40,002
65,536	10,000	$0, 1, \ldots, 9{,}999$	20,000	8,192	6

Sizes in bytes: array $2k$, bitmap 8,192, run $2 + 4r$ (r: number of one-runs).

At **4,096 values**, array and bitmap both use **8,192 B**.

Consecutive values share **one stored range**.

Calculated alternatives. Run sizes include the 2 B run count. All exclude bitmap-level headers, keys and offsets.

[3] Roaring format specification.

AND between Roaring containers

First match block keys. Different keys share no positions.

Matching container types	Local intersection example
Array / array	$\{2,5,7\} \cap \{1,5,7,9\} = \{5,7\}$
Bitmap / bitmap	00100101 AND 01000101 = 00000101
Array / bitmap	Test $\{2,5,7\}$ in 01000101, retaining $\{5,7\}$

- Sorted keys, array values and runs enable **logarithmic lookup**.

Same logical result

Choose the operation for each pair of container types, then encode the result.

Bitmap excerpts show local positions 0–7 from left to right.

Global density hides local structure

Four blocks, each with 65,536 possible positions:

Key	Local pattern	Ones	Plausible container
0	Values $3, 9, 100$	3	Array
1	Every even local position	32,768	Bitmap
2	Range [10,000, 19,999]	10,000	Run
3	All zeros	0	None

$$N = 262{,}144, \qquad k = 42{,}771, \qquad k/N \approx 16.3\%$$

Predictions about storage and query time

Data pattern	Hypothesis to test
Long uniform regions	WAH/EWAH replace many groups with fills.
Few values in each block	Roaring arrays can use little payload.
Dense fragmented blocks	Bitmap containers and raw word operations may be competitive.
Mixed local patterns	Roaring may benefit from different container types.

Smallest representation: fastest AND/OR? Why might the rankings differ?

Hypotheses only. No timing measurements are shown.

Comparable experiments

- Hold N and k fixed when comparing different value patterns.
- Give every format the **same logical input sets**.

  For $A = \{2, 5, 7\}$, Raw, WAH, EWAH and Roaring must all represent exactly $\{2, 5, 7\}$.
- For AND/OR queries, vary how many positions A and B share.
- Define whether a query materializes its output or only counts it.

Equal input densities can yield different output sizes

If $|A| = |B| = k$ and $t = |A \cap B|$, then $|A \cup B| = 2k - t$.

Assignment 1: Compression Benchmark

Work in groups of 2–3 students.

Research question

How do data patterns affect representation size and the time to compute AND and OR?

- Form hypotheses using fills, literals and local containers.
- Compare formats under the same workload.
- Explain the observations using the representation structure.

Proposed representations and datasets

Representations: raw baseline, WAH, EWAH, Roaring, plus one additional format chosen by your group.

Use existing libraries supported by the starter framework.

Family	Reproducible generator definition
Sparse scattered	Sample k distinct positions uniformly.
Sparse clustered	Place the same k ones in disjoint intervals.
Dense scattered	Complement a sparse scattered zero set.
Dense clustered	Complement a matched clustered zero set.
Mixed regions	Assign a generator and k to each block.

Several N and densities, matched clustering cases and feasible A/B overlaps. Record seeds and interval settings.

Metrics and measurement procedure

Metric	Measurement boundary
Serialized size	Bytes in the saved bitmap representation, including format metadata.
Construction time	Build from the same prepared input. Specify whether optimization time is included.
AND / OR time	Produce a materialized result. Specify whether result allocation is timed.

- Generate inputs and check correctness outside query timings.
- Warm up where needed, repeat runs, report a timing summary.
- Record hardware, versions and representation settings.

Measure process memory separately. Count-only queries are an optional extension.

Correctness against a raw reference

For each format and each input pair:

$$\text{decode}(A \,\text{AND}\, B) = A_{\text{raw}} \,\text{AND}\, B_{\text{raw}}$$

Check OR in the same way, over **all valid positions**.

- Empty/full sets, identical inputs and disjoint inputs.
- Values around word boundaries and the 65,536-position boundary.
- The last partial group or block, and valid logical length N.
- Saving and loading preserves the bitmap.

Equal cardinality alone does not prove equal membership. Mask padding outside N.

Your group's experimental work

Use the starter framework to:

1. **Design inputs and a testable hypothesis**
   Choose value patterns and parameters; state what would contradict your prediction.
2. **Measure all five representations**
   Size, construction time, and both AND and OR; repeat runs.
3. **Explain results using representation statistics**
   Count relevant fills, literals, exceptions or container types.
4. **Prepare your own worked example**
   Show the additional format's encoding and one AND or OR operation.

Required comparison

Two datasets with the same N and k, but different value patterns.
Show how and why your additional format's size or query time changes.

What the report should explain

1. **Setup and hypotheses**
   Generators, seeds, libraries, settings and the tested workload.
2. **Results with units**
   Serialized bytes, construction time, AND/OR time and variability.
3. **Structural explanation**
   How fills, literals, container types and output size explain the results.
4. **Limits of the comparison**
   Overheads included, measurement boundaries and unsupported cases.
5. **Group members and contributions**
   Names, each member's work and percentage contribution (total: 100%).

Submit your report via Blackboard.

Group presentations

Presentations on 10 November 2026.

Optional consultation next week.

10 minutes per group + 5 minutes for questions.

1. **Explain your additional format**
   Its main idea and how it differs from WAH/EWAH.
2. **Show your own worked example**
   Encoded fields, size in bytes, and one worked AND or OR example.
3. **Present your investigation**
   Research question, setup and results for both AND and OR.
   Compare with Raw, WAH, EWAH and Roaring.
4. **Explain the results and limitations**
   Connect size and query time to the representation's structure.

What determines a useful representation?

- **WAH/EWAH:** uniform regions become fills, mixed regions use literals.
- **Roaring:** block structure permits local container choices.
- **Data arrangement:** equal density can hide different compression opportunities.
- **Workload:** storage and query time require separate evidence.

Discussion

Which pattern would you test first, and what result would challenge your prediction?

WAH boundaries and finite length

For w-bit encoded WAH words:

$$\text{literal payload} = w - 1, \qquad \text{fill counter width} = w - 2.$$

- Fills represent whole $(w-1)$-bit uniform groups.
- A long fill may need several words when its counter is full.
- A final partial group needs a defined padding/length policy.
- Finite-universe NOT must exclude padded positions beyond N.

[1] FastBit: WAH compression.

Additional container intersections

Pair	Example over local positions
Run / run	$[3,7] \cap [5,10] = [5,7]$
Array / run	$\{2,5,7,9\} \cap [3,7] = \{5,7\}$
Bitmap / run	Keep set bits within the run's interval.

- AND uses matching block keys.
- OR also includes containers whose keys occur in only one input.
- Result cardinality and structure can change its container type.

Overlap and a possible result table

For $a = |A|$, $b = |B|$, $t = |A \cap B|$ in a universe of size N:

$$\max(0, a + b - N) \leq t \leq \min(a, b), \qquad |A \cup B| = a + b - t.$$

Suggested result fields

Workload and seed, N, a, b, t, format and settings, serialized bytes, construction time, AND/OR time, repetitions and variability.

A proposed reporting aid. No benchmark results or final submission format are implied.

Technical references

1. FastBit: Word-Aligned Hybrid Compression for Bitmap Indices
   WAH literals, fills and operations on compressed bitmaps.
2. Git bitmap format, Appendix A
   Concrete 64-bit EWAH markers and serialization.
3. Roaring bitmap format specification
   32-bit partitioning, containers and serialized payload layouts.

Worked inputs and payload calculations in this lecture are constructed teaching examples.