Tagma

Hashless spatial primitive on a fixed 16-bit Unicode, 3-axis composition space

Taeho Lee*
SSCCS Foundation
ssccs.org

July, 2026

Abstract

Tagma[1] is a computing primitive where the address *is* the coordinate — not a flat pointer, but a point in an N-dimensional geometric space. This is made possible by a fixed 16-bit contiguous Unicode closed-form composition block *(U+AC00-U+D7AF)*, which provides a collision-free, hash-less, structurally addressable coordinate space. Every valid 16-bit value in this space is simultaneously a 1-D address, a 3-D coordinate (Axis 0, Axis 1, Axis 2). This triple interpretation enables hash-less content addressing [*4.3*] with zero collision probability and single-cycle combinational decoding (~300 gates). The coordinate arithmetic admits unbounded N-Coord composition — the address space grows as $11,172^N$ while lookup cost remains O(N) per Coord. We present a gate-level decoder specification, a processor pipeline attachment reference (e.g., RISC-V XIF)[2], a software reference implementation [*A*], and a full benchmark suite [*B*]. Measured lookup latency: 0.38 ns vs 227 ns for SHA-256 (597x) at single-Coord; 54.9 ns vs 227 ns (4.1x) at SHA-256 scale (19 Coords). Structural prefix queries show the decisive advantage: a direct-address table answers nonexistent prefix in 1.65 ns while a general-purpose hash table scans 10M entries in 23.05 ms (14.0Mx). Bit-set axis filters resolve compound queries at 329 Melem/s vs hash table scan at 2.4 Melem/s (137x, bitwise AND vs full scan). The same table completes 10M sparse get operations in 44.9 ms vs hash table 1.05 s (23.4x). The coordinate space is an open international standard (Unicode) and a public good — unencumbered by proprietary licensing.

tagma-10.5281/zenodo.21302508-df1c1b-260803

1 Introduction

Unicode is an international standard that assigns each script a fixed address in a shared encoding space. Within this space, this block [1] occupies a contiguous 16-bit segment whose closed-form composition formula embeds three independent structural axes into every code point.

Every content-addressable system today generates identifiers through a hash function. This indirection layer imposes a predictable cost: gate area, cycle latency, storage overhead, and probabilistic collision resolution. Tagma shows that this fixed 16-bit block can replace the hash function entirely where cryptographic integrity is not required, using a combinational decoder.

The composition formula [2] [3] is:

$$C(i, m, f) = \text{U+AC00} + 588i + 28m + f, \quad 0 \le i < 19,\ 0 \le m < 21,\ 0 \le f < 28$$

*Founder & Architect; Corresponding author: lee@ssccs.org

[1] Greek *τάγμα* from *σύνταγμα* (syn-tagma, *co-ordinate* in English), "a well-ordered arrangement of constituent elements". *Tagma* is the system name; *Coord* is the concrete implementation type.

[2] Repository: Github (Pre-release, Apache 2.0), Benchmark suite code

Every valid 16-bit value carries **three simultaneous interpretations**: a 1-D Unicode address, a 3-D coordinate. Internally, the 16 bits are organized as a 3-axis coordinate packed into a 16-bit word:

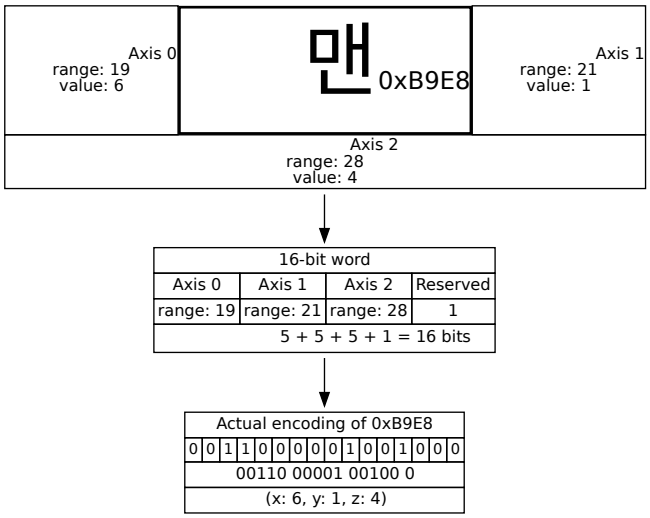

Figure 1: Coordinate 0xB9E8: three-axis coordinate (6, 1, 4) packed into a 16-bit word.

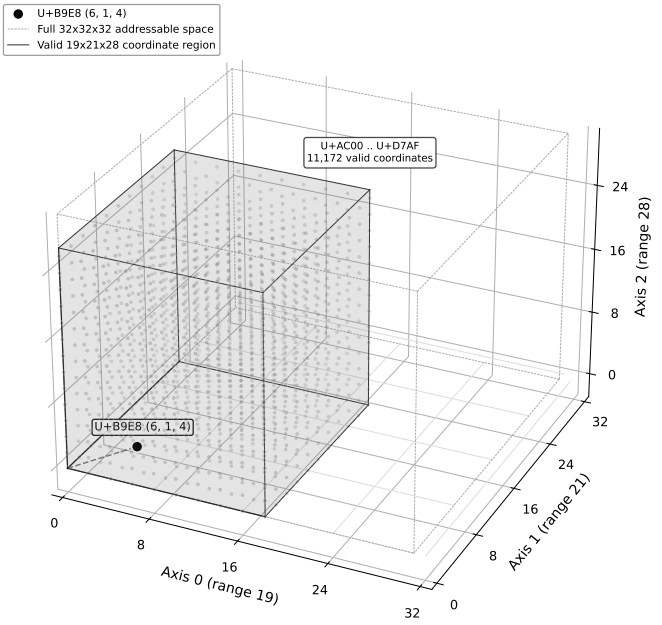

Figure 2: Three-dimensional bitcell organization. 19 x 21 x 28 lattice, 11,172 of 65,536 states valid.

2 The Structural Conjunction

Tagma requires the simultaneous satisfaction of four properties: **a contiguous fixed-width 16-bit address range, a closed-form composition formula, a complete three-axis decomposition, and an open international standard.** The absence of any single property makes a Tagma-compatible coordinate space impossible.

This block satisfies all four. It resides on Unicode's Basic Multilingual Plane (Plane 0) — the foundational layer that every implementation must support — making Tagma universally compatible without requiring supplementary planes or special handling. Every other encoding on the BMP uses atomic assignments inherited from ASCII. None embeds three structural axes into each code point.

Script	Contiguous Address	Combinatorial Structure	Three-Axis Decomposition	Tagma-compatible?
Hangul	Y U+AC00-U+D7AF	Y 19x21x28, no exceptions	Y Onset-Nucleus-Coda	Yes
ASCII	Y U+0000-U+007F	N	N	No
Japanese	Y U+3040-U+30FF	N	N	No
CJK Unified Ideographs	Y U+4E00-U+9FFF	N (infinite, irregular)	N	No
Arabic	Y U+0600-U+06FF	N	N	No

3 The Encoding Problem

Variable-length encoding has architectural consequences that are independent of any particular implementation. Memory alignment is not guaranteed. Instruction decoding must detect boundary positions. Prefetch and pipeline

efficiency are affected. These are structural properties of variable-length design, not engineering limitations. The structural constraint in Tagma arises from the fixed composition formula of this block. As established in the Structural Conjunction section, only 11,172 of 65,536 states satisfy the composition formula. A hardware decoder can distinguish valid from invalid states using combinational logic derived from that formula.

Encoding	Storage unit	Addressable symbols	Wasted bits/unit	Fixed alignment	Structural HW check
ASCII	1 byte (8 bits)	128 of 256	1 bit (12.5%)	yes	no
Latin-1	1 byte	191 of 256	variable	yes	no
UTF-8 (English)	1-4 bytes	unlimited	none for ASCII	no	no
UTF-16 (BMP)	2 bytes	63,488 of 65,536	2,048 reserved	no	no
UTF-32	4 bytes	unlimited	75-88% for common use	yes	no
Tagma	**2 bytes**	**11,172 of 65,536**	**54,364 structurally invalid (error detection)**	**yes**	**yes (54,364 invalid)**

Because the coordinate is a fixed 16-bit BMP value, the address role needs no decoder, no parser, and no operating system: `*(base + coord)` is the entire lookup, from a kernel bootloader to an MMU-less microcontroller. Text input still requires one conversion from the carrier encoding to the code point.

4 The Identity Problem

Every system that stores data by content rather than by location must generate identifiers. Current approaches and their hardware cost:

Method	Identifier size	Generation cost	Collision	Lookup structure	Tagma advantage
Pointer	32-64 bits	zero	none	direct (location-based)	—
Hash (SHA-256)	256 bits	~10K gates, 64-75 cycles	probabilistic	hash table + resolution	~30x fewer gates, zero collisions
UUID [4]	128 bits	entropy-dependent	probabilistic	hash table	deterministic, no entropy needed
CAM	per-bit comparison	9-16 transistors/bit	none	associative [5]	~2.5x less area for sparse sets
Tagma	**16 bits**	**1 cycle (combinational)**	**none (formulaic)**	**direct (coordinate = address)**	baseline

A single Coord covers 11,172 identifiers — sufficient for sensor arrays, embedded device registries, and moderate-scale lookups [6]. N-Coord composition (below) extends this to UUID-scale and SHA-256-scale spaces without changing the decoder or the arithmetic.

The practical implication for everyday identifiers is direct. UUID generation requires entropy collection, version/variant bit insertion, and hyphen formatting at approximately 100 ns or more; its 128-bit output expressed in hex or Base64 consumes 36 or 22 characters respectively. Tagma produces a 6-Coord identifier (18 axes, 1.9×10^{24} space) in approximately 150 ns, and a 10-Coord identifier exceeding UUID space in approximately 260 ns, each requiring only multiplications and additions, as a human-readable string with no special characters, no padding, and zero collision probability. Similarly, a SHA-256 hash output as 64 hex characters is matched by 19 Tagma

Coords covering the same 2^{256} space in approximately 456 ns. In all cases Tagma output is shorter, faster, and structurally self-validating. A software reference implementation confirming these measurements is described in Appendix [A].

4.1 N-Coord Composition: From 16 Bits to SHA-256 Scale

A single Tagma Coord provides 11,172 unique identifiers over 16 bits. For larger address spaces, Coords compose linearly:

$$S(N) = 11,172^N \approx 10^{4.05N}$$

Linearization. An N-Coord coordinate (c_1, ..., c_N) maps to a single linear index via row-major order:

$$\text{index}(c_1, \dots, c_N) = \sum_{k=1}^{N} c_k \times 11,172^{\,N-k}$$

This requires exactly $(N-1)$ multiply-add pairs, a fixed sequence of $2(N-1)$ arithmetic operations independent of the addressable space. Since N is a compile-time constant, the lookup remains O(1) for any N-Coord composition.

Coords	Axes	Identifier space	Equivalent to
1	3	1.12×10^4	Sensor tags
2	6	1.25×10^8	Database records
3	9	1.39×10^{12}	Distributed nodes
6	18	1.94×10^{24}	Below UUID (3.4×10^{38})
8	24	2.41×10^{32}	Approaches UUID
9	27	2.99×10^{36}	Below UUID (3.4×10^{38})
10	30	2.69×10^{40}	Exceeds UUID (3.4×10^{38})
19	57	1.94×10^{77}	SHA-256 (256-bit) equivalent

Each Coord retains independent hardware-verifiable validity ($19 \times 21 \times 28 = 11,172$ gates per Coord). The N-tuple identity space is the Cartesian product of N independent 3D spaces, yielding a $3N$-dimensional structural coordinate system. We call this N-Coord sequence a CoordPath, distinct from the atomic Coord.

This is not merely a larger address space, but a coordinate algebra of $3N$ independent axes whose semantics are entirely application-defined: a sensor network may assign axis 0 to device type, axis 1 to geographic zone, and axis 2 to timestamp, while a database system assigns the same axis positions to table, partition, and row. The coordinate arithmetic – composition, decomposition, linearisation, Hamming distance, axis projection – is invariant under semantic reinterpretation because the axes are fungible; the algebra depends only on the invariant structural relations among them, not on what any particular axis represents. This distinguishes Tagma from Euclidean space, where axes are bound to physical dimensions, and from hash space, where all structure is destroyed.

4.2 Recursive State Space Expansion

The composition formula $C(i, m, f)$ accepts three axes whose ranges are fixed: 19, 21, and 28. Nothing in the formula requires these ranges to be atomic. A SynTagma is the structure that results when these axes are themselves composed of CoordPaths. An N-Coord CoordPath can occupy any axis position, replacing the original range with its own $11,172^{\,N}$ values.

Let $\mathbb{T}_0$ be the set of all valid 19-Coord CoordPaths:

$$S_0 = |\mathbb{T}_0| = 11,172^{19} \approx 1.94 \times 10^{77}$$

Define $\mathbb{T}_1$ as the Cartesian product of three $\mathbb{T}_0$ coordinates:

$$\mathbb{T}_1 = \mathbb{T}_0 \times \mathbb{T}_0 \times \mathbb{T}_0$$

$$S_1 = |\mathbb{T}_1| = S_0^3 = (11,172^{19})^3 = 11,172^{57}$$

A CoordPath of length $3k \cdot N$ Coords can be interpreted as either a flat sequence of $3kN$ atomic coordinates or a k-deep nested structure of SynTagma triplets. Both interpretations produce the same linear index through the same arithmetic.

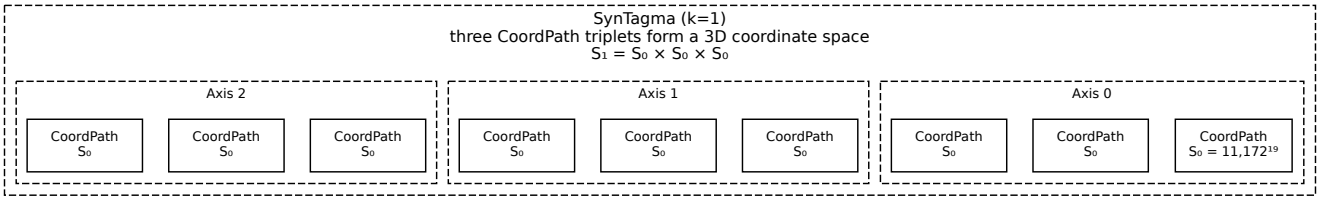

Figure 3: SynTagma contains three CoordPath triplets as its axes.

Access follows from the recursive linearisation formula:

$$\text{index}(a, b, c) = a \cdot S_{k-1}^2 + b \cdot S_{k-1} + c$$

Each of a, b, c is itself linearised recursively until atomic Coords are reached. The number of integer operations per lookup is proportional to the total Coord count, independent of the number of stored entries.

4.2.1 Sparse Allocation

The identifier space S_k is not a storage allocation. A CoordSpace (Appendix [A]) allocates a root array of 11,172 pointers (approximately 89 KB on 64-bit systems) and creates child nodes only along CoordPaths that are actually written to. Memory consumption is proportional to the number of stored entries, not the size of the address space.

4.2.2 Magnitude

For $k = 1$ with a 19-Coord base:

$$S_1 \approx 7.30 \times 10^{231}$$

The estimated number of atoms in the observable universe is approximately 10^{80}. The ratio:

$$\frac{S_1}{10^{80}} \approx 7.30 \times 10^{151}$$

A single recursive step on a 19-Coord base exceeds the atomic count of the observable universe by 151 orders of magnitude. For $k = 2$:

$$S_2 = S_1^3 \approx 3.89 \times 10^{695}$$

No physically meaningful comparison remains. There is no mathematical terminal; the formula admits unbounded k. The practical limit is in the silicon budget of the implementation.

4.2.3 From Observable Horizon to the Whole Universe

The comparison value 10^{80} (atoms in the observable universe) is a familiar reference point, but it is defined by the cosmic light horizon at approximately 46.5 billion light-years. This is a causal limit, not a physical boundary. The actual universe may be many orders of magnitude larger or infinite. The coordinate space exceeds not only this local figure but any physically conceivable finite universe. Even the most generous estimates for the total baryon count of a finite universe under standard ΛCDM curvature bounds fall short of S_1 by more than 145 orders of magnitude. The coordinate space is not bounded by cosmological horizons; it is a mathematical object indexed by arithmetic composition.

4.3 Hash-less computation: Direct structural addressing

In conventional computing, identity is established through indirection:

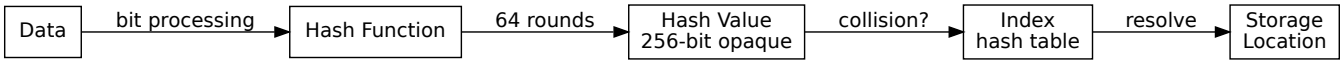

Figure 4: Traditional hash-based identity model.

Each step adds cost: hash computation, collision handling, dynamic resizing, cache-unfriendly access patterns. Tagma replaces this with direct structural addressing:

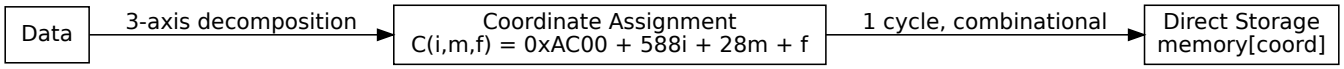

Figure 5: Tagma direct structural addressing.

This one-cycle path from data to address eliminates the scaling relationship between entry count and retrieval cost, a property that standard complexity analysis cannot capture.

4.4 Structural Consequences

The transition from hash-based indirection to coordinate-based addressing produces several interrelated consequences that define Tagma's operational regime. They are not independent properties of the implementation; they are necessary corollaries of the coordinate identity model.

Query reduces to arithmetic. A coordinate's three axes are independent fields in a 16-bit word. A query that filters by axis value is a range projection over that field — a bitmask and comparison, not an index scan. Proximity search is field-wise Hamming distance, computable as a single bitwise operation. The coordinate arithmetic is the query engine; no hash tables, B-trees, or CAM cells are needed.

Index structures are eliminated. In current systems, identity generation and index maintenance are separate concerns: SHA-256 produces an opaque identifier, then a separate hash table or B-tree maps that identifier to a storage location. The coordinate is identity and address simultaneously. A Tagma identifier can be used directly as a memory address, array index, or routing path without an intervening translation layer. The elimination is not an optimization of the index layer; the index layer ceases to exist as a separate structure.

The coordinate space is a physical structure, not a software abstraction. The decoder that maps a 16-bit coordinate to its three axis fields is a combinational circuit operating on a fixed encoding formula. A software library implementing the same formula on a general-purpose CPU is a faithful simulation of that circuit, not a design intent of its own. The space exists at the gate level before any software runs. This reverses the conventional relationship between software and hardware: software does not define the addressing scheme and then implement it in gates; the addressing scheme is already a gate-level structure, and software either uses it (on a Tagma-equipped processor) or simulates it (on a general-purpose CPU).

For example, the software reference implementation reveals this inversion clearly. The `Coord::to_axes()` method decomposes a Coord into its three axis fields using arithmetic:

```
pub fn to_axes(self) -> (u8, u8, u8) {
    let v = self.0 as usize;
    let initial = (v / 588) as u8;
    let medial = (v % 588 / 28) as u8;
    let final_ = (v % 28) as u8;
    (initial, medial, final_)
}
```

The code computes division and modulo because it is simulating a gate-level structure on a sequential processor. In hardware, the same fields are present as three contiguous 5-bit groups in the 16-bit word. The hardware does not compute; it reads. The division and modulo in the software are artifacts of simulating a parallel bit extraction on a sequential ALU. A Tagma-equipped processor executes `to_axes()` in zero cycles — the axes are already present on the output wires of the decoder. The software complexity inversely indicates the hardware simplicity: the more arithmetic the reference implementation requires, the more directly the hardware structure implements it. This is the opposite of hash functions, where software and hardware complexity are proportional: SHA-256 is expensive in both domains.

The coordinate space is a search engine, not a store. Tagma replaces the key-value model with a searchable coordinate space. A hash table answers only "value at this exact key"; it cannot answer "entries satisfying a condition" without scanning. Tagma inverts this: the coordinate space is itself the execution plan, with projections, proximity searches, and set operations all reducing to coordinate arithmetic. The departure is fundamental: hash tables answer "get(key)", coordinate spaces answer "navigate(condition)" – and the latter is what most queries need.

Property	Key-value store (HashMap)	Coordinate space (Tagma)
Input	Key (string, hash)	Coordinate (Coord, CoordPath)
Output	Value	Value + position in coordinate space
Query model	Exact key match only	Axis projection, range, proximity, set operations
String dependency	String keys require encoding, parsing, hashing	String is a display layer; the address is structural
Index requirement	Separate index structures for conditional queries	Identity is the coordinate; no separate index exists
Scaling variable	Entry count (collision chains, rehashing)	Coord Depth d (fixed per schema, data-independent)

The shift from string-based to coordinate-based addressing eliminates an entire class of costs that conventional systems treat as unavoidable:

Vanishing cost	String-based	Tagma (Coord-based)	Effect
Memory allocation	heap allocation	a stack-allocated `u16`	Zero heap fragmentation; allocator and GC load eliminated
Encoding validation	Every input string must be verified as valid UTF-8	A `Coord` is a pre-validated 16-bit integer; any valid `u16` in range is structurally valid	Validation circuit and cycles eliminated

Vanishing cost	String-based	Tagma (Coord-based)	Effect
Comparison	`str1 == str2` iterates bytes up to the shorter string length	`coord1 == coord2` is a single CPU compare instruction	$O(n) \to O(1)$; one cycle
Hashing (storage)	A string key must hash its entire byte sequence before the map can be indexed	A `Coord` is itself a complete array index	Hash function call eliminated; zero collision resolution
Serialization / parsing	Keys in JSON, MessagePack, or protocol buffers must be parsed, validated, and copied	A `Coord` is a fixed 2-byte binary value; the serialized form is the in-memory form	Parsing overhead eliminated; zero-copy direct mapping

These are not optimizations of the string path but its elimination. A system that never uses strings as addresses never pays string costs. The question of whether Tagma could be faster than a hash table on a particular benchmark is therefore misdirected: Tagma does not compete with hash tables on hash-table workloads. It competes by making the addressing substrate invisible — an operation that hash tables cannot attempt because they depend on the very indirection Tagma removes.

The question of string keys dissolves under this framing. An application that stores "user:123:item:456" must parse, hash, and index that string before it can retrieve anything. Tagma assigns a CoordPath directly — Coord for user type, Coord for user ID, Coord for item — and the path is both identifier and address. The string remains a human-readable label generated from the CoordPath for display, not parsed into one for storage. This separation eliminates hashing from the critical path.

The deeper distinction is between a map and a space. A hash map is lossy compression: it preserves only the mapping from each key to its value, discarding all structural relationships between keys. Two keys that differ by a single bit land in completely unrelated buckets; proximity is lost, axis structure is lost, the geometry of the identifier space is destroyed by the hash function. No amount of magnification applied to a map reveals the terrain it abstracts. Tagma is the uncompressed original: every coordinate retains its full structural position in the space, and every operation — lookup, proximity search, axis projection, set membership — operates on the coordinates directly. The space does not compress away structure because the structure is the address.

4.5 The Complexity of Structure: Coord Depth

Standard complexity analysis classifies algorithms by how their cost scales with input size N — the number of data items. Tagma's retrieval cost does not scale with data volume at all. It scales with **Coord Depth** d, the number of Coords required to uniquely identify an entity under a given scheme. These are fundamentally different kinds of quantities: one measures data inventory, the other measures structural dimensionality.

d is a system constant determined at schema design time. A sensor identifier requires $d = 1$ (single Coord, 11,172 identifiers). UUID-scale identity uses $d = 6$. SHA-256-scale identity uses $d = 19$. Adding a billion entries does not increase d.

$$\text{Retrieval cost} = O(d), \quad d \ll \log N_{\text{entries}}$$

Metric	Hash table	Tagma
Scaling variable	Entry count N	Coord Depth d (fixed per scheme)
Growth with data	Increases	None
Worst case	$O(N)$ (collision chain)	$O(d)$ (same as average)
Operation	Chop and mix bits	Decompose along structural axes

Metric	Hash table	Tagma
Lookup	Hash + resolve + dereference	Direct array access
Output	Opaque 256-bit value	Self-describing 16-bit value

A hash function receives a key and produces an address by destroying the key's internal structure. The lookup is indirect: hash, resolve collision, dereference. Coord inverts this: the Coord sequence carries its own address — the data is already the address. This inversion makes Coord Depth a distinct scaling regime, expressible neither as $O(1)$ (which implies independence from any parameter) nor as $O(N)$ (which implies dependence on data volume). Coord Depth $O(d)$ parameterizes retrieval by the identifier's inherent dimensionality, a quantity bounded by the application schema, not by the data set.

The engineering frontier. The mathematical bound holds for any fixed d: $2(d-1)$ arithmetic operations per lookup. On a general-purpose CPU, however, each operation incurs mechanical overhead: instruction dispatch, register pressure, pipeline latency, and cache hierarchy effects. At $d = 1$ these are negligible (one array access). At $d = 19$ the 18 multiply-add pairs accumulate approximately 456 ns on a GitHub Actions CI runner (x86_64) — still 10x faster than a single SHA-256 computation, but measurably more than the O(1) ideal of a single cycle. This gap follows from running a structurally parallel operation on a sequential von Neumann machine rather than from a flaw in the theory. A dedicated combinational decoder will execute all d Coord decodes concurrently at gate level, collapsing the gap entirely.

The elimination of separate index structures is a direct consequence of coordinate identity:

Layer	Current (SHA-256)	Tagma
Identity	$H = \text{SHA256}(\text{data})$ (256-bit opaque)	$\text{Tagma}(i, m, f) = \text{U+AC00}+588i+28m+f$ (16-bit transparent)
Index	separate structures: HashMap, OrderedIndex	none: identity = coordinate
Query	$\text{Vec}[o] \cap \text{Vec}[c] \cap \text{Vec}[t]$ – $O(K \times N)$ intersection	coord.decompose() – $O(1)$ direct field extraction

This property, verified in the reference implementation, is the software-level manifestation of the same principle the hardware decoder implements: the coordinate IS the identity, and the identity IS the coordinate. No indirection, no intersection, no hash tables.

5 The Decoder

The decoder extracts three 5-bit fields from a 16-bit input in three combinational stages:

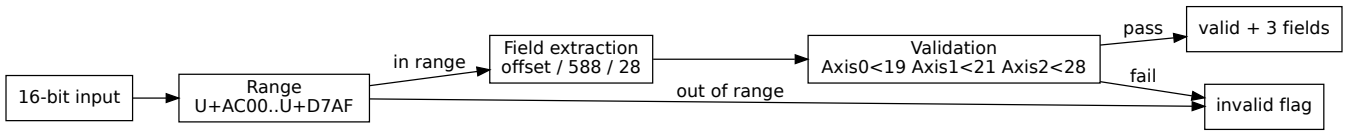

Figure 6: Three-stage decoder flow: range check, field extraction, validation.

Stage 1: Range check. Two comparators verify the input is within [U+AC00, U+D7AF]. An out-of-range value produces an immediate invalid flag.

Stage 2: Field extraction. For in-range values:

Step	Operation	Output	Range
1	`input - 0xAC00`	offset	0-11,171

Step	Operation	Output	Range
2	`offset \div 588`	Axis 0	0-18
3	`offset \bmod 588`	remainder	0-587
4	`remainder \div 28`	Axis 1	0-20
5	`remainder \bmod 28`	Axis 2	0-27

Stage 3: Validation. Three comparators check: Axis 0 < 19, Axis 1 < 21, Axis 2 < 28. Any failure sets the invalid flag. Tagma coordinates support proximity-based search without dedicated search hardware. Given two coordinates, the field-wise Hamming distance is computed by three banks of parallel XOR gates. This enables associative lookup without hash tables, B-trees, or CAM cells.

Component	Gates
Range check	~50
Offset subtraction	~80
Division by 588 (multiply-shift)	~80
Division by 28 (multiply-shift)	~60
Validation comparators x3	~30
Total	**~300**

Approximately 300 gate equivalents in 28nm. The decoder is smaller than a single 32-bit multiplier. The full compliance criteria for Tagma-compatible implementations are defined in Section [*11*].

6 Reference Implementation: Coprocessor Attachment

Tagma can be attached to any processor pipeline through a standard coprocessor interface for custom instructions that does not require modifying the core pipeline. An example implementation uses the RISC-V XIF interface [7], [8]. The Rust reference implementation (`Coord`) serves as the functional golden model: every hardware instruction must produce results bit-exact with the corresponding Rust function.

Figure 7: Tagma position in the system stack.

Three custom instructions implement the core `Coord` API:

tagma_check rd, rs1. Decodes the lower 16 bits of rs1. Returns a 16-bit value packed as: valid flag (bit 15, 1 = valid), reserved (bit 14, reads as zero), Axis 0 (bits 13–9, 5 bits, range 0–18), Axis 1 (bits 8–4, 5 bits, range 0–20), Axis 2 (bits 3–0, 5 bits, range 0–27). Invalid inputs produce a zero valid flag with undefined field values. Corresponds to `Coord::new(rs1).map(|c| c.to_axes())`. One cycle, combinational.

tagma_compose rd, rs1, rs2, imm5. Composes a coordinate from three 5-bit axis fields. rs1 supplies the Axis 0 field (selected by imm5 = 0), rs2 supplies the remaining two fields, or the immediate fields are extracted from rs1[14:0] directly (Axis 0 at bits 14–10, Axis 1 at bits 9–5, Axis 2 at bits 4–0). Returns the 16-bit coordinate index (0–11171) in rd, or zero if any axis is out of range. Corresponds to `Coord::from_axes(initial, medial, final).map(|c| c.index())`. One cycle, combinational.

tagma_dist rd, rs1, rs2. Computes field-wise absolute difference between the two 16-bit coordinates. Returns three 5-bit distances packed as: Axis 0 (bits 14–10), Axis 1 (bits 9–5), Axis 2 (bits 4–0); bit 15 is reserved. Corresponds to `Coord::hamming_distance(rs1, rs2)`. One cycle, combinational.

6.1 Verification

The coordinate space is exhaustively enumerable: 65,536 values, of which 11,172 are valid. A verification harness generates all 65,536 inputs, applies the decoder specification, and records the results. The bijection is verified by enumeration: no collisions, no unassigned values within the block.

6.2 Architectural Mapping

A coordinate in this space is a 16-bit value whose identity is its structure. The composition rules of Axis 0, Axis 1, and Axis 2 correspond to the Scheme that defines which coordinates are valid. Validity predicates (range checks) implement the Field by defining admissible coordinate ranges. Enumerating coordinates under constraints implements Observation: a hardware-level selection of valid states.

7 Comparison with Existing Paradigms

Content-Addressable Memory provides content-based lookup in one cycle but at 9-16 transistors per bit versus 6 for SRAM [5]. At the Tagma maximum of 11,172 entries, Tagma with direct mapping uses standard SRAM cells at approximately 2.5x less transistor count than CAM, at the cost of 54,364 unused states. Tagma is more area-efficient when the stored set is sparse; CAM is more efficient when dense.

The power difference is larger than the transistor count difference. Every CAM search cycle precharges all matchlines (one per stored word) and conditionally discharges them through the comparison cells, consuming approximately 0.3–1.0 pJ per bit per search. By contrast, an SRAM read of the addressed word dissipates approximately 0.05–0.2 pJ per bit [5]. For an 11,172-entry, 16-bit array, a full CAM search activates all 11,172 matchlines and 16 searchlines simultaneously, whereas a Tagma direct SRAM read activates a single wordline and 16 bitlines, yielding an energy difference of approximately four orders of magnitude per operation.

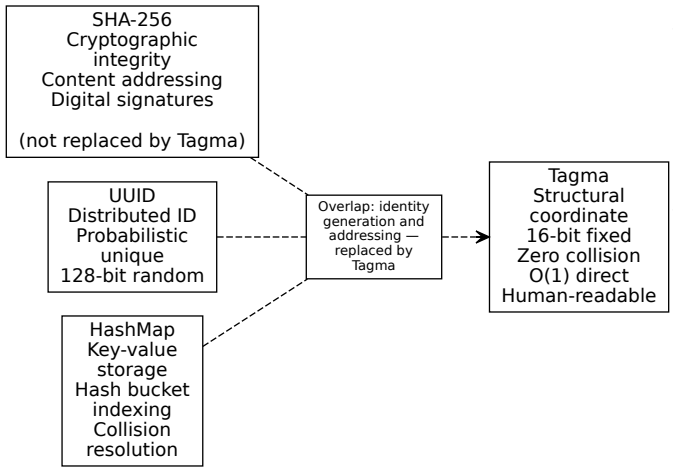

Figure 8: Tagma occupies the structural coordinate space, replacing the hash-based addressing role in all three while leaving cryptographic functions to SHA-256.

Existing addressing paradigms each optimize a different axis: pointers minimize generation cost (zero) but provide no content-addressability; hash functions maximize identifier space at the cost of indirection; CAM cells offer full associativity at a transistor premium. Tagma occupies a fourth category, structural addressing, where identifier, address, and structure converge into a single 16-bit value decodable at the gate level. Every coordinate maps to exactly one slot, in one cycle, with zero collision resolution. Hash-based systems accept probabilistic guarantees and variable latency; Tagma guarantees both latency and uniqueness by construction. Combined with N-Coord composition (yielding up to 1.94×10^{77} identifiers at 19 Coords), the bound dissolves for practical purposes while per-Coord determinism is preserved.

A comprehensive comparison across dimensions not covered by individual paradigms above:

Attribute	Traditional (Hash-based)	Tagma
Identifier generation	Hash (thousands of cycles)	Arithmetic (1-2 cycles)
Collision resistance	Probabilistic (finite hash space)	Guaranteed (coordinate is unique)
Memory layout	Dynamic hash table (variable size)	Fixed array (predictable size)

Attribute	Traditional (Hash-based)	Tagma
Concurrency	Requires synchronization	Lock-free (slot-level atomic)
Self-validation	Separate checksum/hash	Built-in (16-bit range check)
Output form	Opaque hex or Base64	Displayable as Unicode text
Structural operations	None (random access only)	Proximity search, slicing, distance
Hardware mapping	Complex (hash function gates)	Simple (decoder + address lines)

In addition to the differentiators listed above, structural coordinates enable two capabilities unavailable to hash-based addressing:

- **Proximity search**: Coordinates (x+1, y+1, z+1) are neighboring positions, enabling locality queries without index intersection.
- **Axis slicing**: All coordinates with a fixed medial value are extracted by projecting a single axis.

What Tagma replaces, partially replaces, and does not replace:

Domain	Hash Role	Tagma Replaces?	Tagma Alternative
ID generation	hash(data) to produce unique identifier	Yes	Coordinate assignment (1-2 cycles, zero collision)
Hash map key	hash(key) to compute bucket index	Yes	CoordSpace (direct array index)
Content addressing	data hash as storage address	Partial	Coordinate-to-data mapping; requires coordinator
Cache key	file metadata hash as cache key	Partial	Composite Tagma coordinate from content attributes
Integrity verification	data hash to detect tampering	No	Retain SHA-256 or BLAKE3
Digital signatures	hash-then-sign for non-repudiation	No	Retain Ed25519 or ECDSA
Key derivation	HKDF for key expansion	No	Retain HKDF
Password hashing	bcrypt/Argon2 for work-factor	No	Retain bcrypt or Argon2

8 Memory Hierarchy Penetration

(Future direction, no current implementation timeline.)

The coprocessor path (above) would attach Tagma to a linear memory system. The coordinate is decoded and the result is used as an address in conventional memory.

8.1 Memory Controller Mapping

The Tagma coordinate space can be mapped to a physical address region configured to bypass virtual-to-physical translation through memory protection or non-standard memory region attributes. The controller then receives each coordinate directly as a physical address and maps `Axis0-Axis1-Axis2` to `row-column` within the existing memory array. General-purpose platforms support custom address mapping logic at the memory controller or system level.

8.2 Three-Dimensional Decode

The bitcell array can be organized as a 19 x 21 x 28 grid (11,172 cells) rather than a linear array. Alternatively, a memory controller can map the 11,172 valid coordinates to a dense linear array through a simple packed-index transformation. The three coordinate fields each have their own decoder:

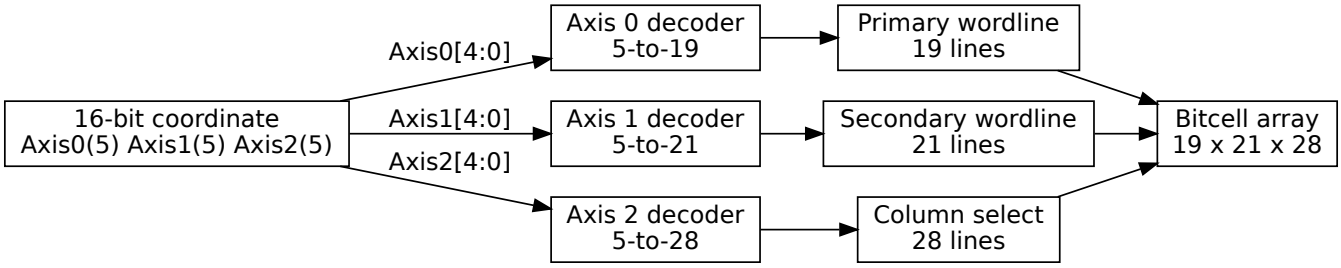

Figure 9: Three parallel decode paths from the 16-bit coordinate fields to the bitcell array wordlines.

Parameter	Linear (SRAM)	Tagma 3D (SRAM)
Decoder type	1 decoder (N-to-2^N)	3 decoders (5-to-19, 5-to-21, 5-to-28)
Worst-case decoder delay	O(2^N)	O(32) max
Address mapping	row + column	Axis 0 + Axis 1 + Axis 2
Address translation	required	none
Structural validity	external (ECC)	built-in

The bitcell array remains standard 6T SRAM. The change is in the peripheral circuitry: three independent decoders replace the single row-column decoder pair, and the sense amplifier layout must accommodate three-axis access. This increases peripheral complexity but leaves the bitcell array unchanged [9].

In a 28 nm CMOS process, a 6T SRAM high-density bitcell occupies approximately 0.127 μm^2. The 11,172-cell core array therefore covers approximately 1,419 μm^2 (0.0014 mm^2). With peripheral circuitry – three decoders, sense amplifiers, and wordline drivers – the total area approximately doubles to 0.003–0.004 mm^2, roughly 30x smaller than a single 32-bit multiplier in the same process.

9 Application Domains

The deterministic single-cycle decode makes Tagma suitable for real-time and safety-critical systems. The three-decoder topology of the future path reduces worst-case decode latency compared to a single linear decoder.

Radiation-tolerant computing. The structural validity check embedded in every decode provides inherent error detection. A single-bit upset that maps a valid coordinate outside the U+AC00-U+D7AF range produces an immediate invalid flag. Errors that map one valid coordinate to another are not detected by the structural check alone and require ECC supplementation, as noted in Boundaries. The decoder's small gate count (approximately 300 gates) makes triplication feasible at lower cost than protecting a full hash unit.

An exact enumeration of all 11,172 valid coordinates confirms the detection rate[3]. Each valid coordinate has 16 possible single-bit-flip destinations. Averaged across all valid states, 12.14 of those 16 destinations remain within [U+AC00, U+D7AF] (standard deviation 0.61; range 10–13). The resulting average SEU detection rate is 24.1% from the structural check alone, before any ECC supplementation. This detection rate comes at zero additional hardware cost, as a free byproduct of the structural encoding. When combined with ECC, the structural check handles the subset of errors that map valid coordinates outside the valid range, while ECC handles the remaining cases.

[3]Computed by exhaustive enumeration: for each of the 11,172 valid Coord values, all 16 single-bit-flip neighbors are tested against the valid range [U+AC00, U+D7AF].

Real-time object identification. In sensor fusion for autonomous systems, Tagma coordinates serve as deterministic object identifiers that do not require hash computation or lookup tables. Each new object is assigned a coordinate at encoding time; subsequent frames reference the same coordinate without recomputation.

Proximity search. The three-axis structure enables field-wise Hamming distance computation through parallel XOR gates, supporting nearest-neighbor lookup without CAM cells or hash-based index intersection.

KV cache addressing. LLM inference engines maintain key-value caches indexed by token sequence prefixes. Current implementations use hash tables or radix trees with $O(L \times H)$ cost per lookup (sequence length L, hash cost H). Tagma replaces this with CoordPath-based direct access: each prefix maps to a unique CoordPath of length L, and lookup cost is $O(L)$ array accesses with zero hash computation. Production KV cache sizes (typically 10^4–10^7 entries) are covered by 2–4 Coords (1.25×10^8 to 1.55×10^{16} identifiers), with deterministic O(1) access and no collision resolution.

Graph adjacency and multi-dimensional query. Graph engines check adjacency via hash lookups or index intersections ($O(\deg(v))$) and query multi-dimensional attributes via composite indexes or join operations. Tagma represents each node as a Coord and each edge type as a CoordSet; adjacency reduces to a single bitwise AND over 175 machine words. Multi-dimensional queries (e.g., "nodes with Axis 0=a, Axis 1=b") project directly to axis ranges without index intersection, with cost independent of graph size.

10 Boundaries

The 11,172-identifier bound per Coord is a consequence of the 19 x 21 x 28 composition formula, not a configurable parameter. It defines the single-Coord direct-address range. Applications requiring larger identifier spaces compose multiple Coords via CoordPath (Section N-Coord Composition). The decoder's structural validity check does not eliminate the need for full error-correcting codes: single-bit errors that map one valid coordinate to another are not detected.

Tagma does not replace cryptographic primitives. SHA-256 remains for signatures, Merkle proofs, and preimage resistance. Encryption, authentication, and key derivation are outside its scope. Tagma replaces the use of hashes as structural identifiers and addresses.

For applications requiring content determinism, where the same data must always produce the same identifier, SHA-256 provides content fingerprinting while Tagma provides human-readable encoding of that fingerprint:

Property	SHA-256 hex	SHA-256 with Tagma
Output	64 hex characters	19 Coords
Determinism	Yes	Yes (SHA preserved)
Human readable	No	Yes
Self-validating	No	Yes (each Coord checked)
Collision resistance	2^-256	2^-256 (SHA preserved)

The combination serves use cases such as file identification, content-addressed storage, and commit hashing where determinism is required but hex output is not. An implementation example combining SHA-256 with Tagma encoding is provided in Appendix [E].

11 Compliance

An implementation is Tagma-compatible iff it satisfies all of the following conditions:

1. **Composition correctness.** The composition formula $C(i, m, f) = \text{U+AC00} + 588i + 28m + f$ must produce the correct Unicode code point for every valid combination of axes ($19 \times 21 \times 28 = 11,172$ triplets).

2. **Structural validity.** Every 16-bit value in the range [U+AC00, U+AC00 + 11,172) must decode to a valid (i, m, f) triplet. Every value outside this range must be rejected, including the 12 filler positions U+D7A4..U+D7AF within the Unicode block but outside the composition formula. Total: 11,172 valid values and 54,364 invalid values in the 16-bit space.

3. **Decomposition correctness.** Decomposition must be the functional inverse of composition: decompose(compose(i, m, f)) = (i, m, f) for all 11,172 valid triplets.

4. **Linearization uniqueness.** The linearization function must be injective over the N-Coord product space. Distinct N-Coord tuples must produce distinct linear indices.

5. **Bit-exactness.** All implementations must produce identical results for the same input across all languages, platforms, and hardware configurations: composition, decomposition, and linearization.

6. **Coord atomicity.** Coord is a single-Coord atomic value. An implementation must not impose application-level semantics on Coord's three axis fields or assume any particular storage strategy for CoordPaths. Coord's only invariant is structural validity.

12 Vision

Tagma defines a new type of silicon primitive: a combinational decoder that derives identity from structure at approximately 300 gates and one cycle. At this gate cost, content-addressable operation becomes viable where hash-based approaches are too expensive in power, area, or latency. The structural validity check embedded in every decode provides inherent error detection, relevant for radiation-tolerant computing in space environments. The coordinate space that enables this is this Unicode block, an open international standard and a public good. Tagma will be released as open-source silicon, inviting the next conversation: what else becomes possible when identity costs less than a single multiply. The broader hardware design implications of this shift are discussed in Appendix [*D*].

The SynTagma specification [10] complements this document. It defines how the recursive state space expansion described in the previous section is realised across physical topologies: routing, transport framing, device-boundary resolution, and distributed coordination. Where this document defines the invariant, SynTagma defines the protocol.

References

[1] Unicode Consortium, "Hangul syllables (u+AC00–u+D7AF)." Unicode Standard, Chapter 3: Conformance, 2020. Available: https://www.unicode.org/charts/PDF/UAC00.pdf

[2] Unicode Consortium, "The arithmetically specified decompositions of precomposed hangul syllables." Unicode Standard, Chapter 3: Conformance, Section 3.12, 2024. Available: https://www.unicode.org/L2/L2003/03203-arithmetic-decomp.pdf

[3] K. Karlsson, "The arithmetically specified decompositions of precomposed hangul syllables." Unicode Technical Report L2/03-203, Jun. 12, 2003. Available: https://www.unicode.org/L2/L2003/03203-arithmetic-decomp.pdf

[4] P. Leach, M. Mealling, and R. Salz, "RFC 4122: A universally unique IDentifier (UUID) URN namespace." Internet Engineering Task Force, Jul. 2005. Available: https://datatracker.ietf.org/doc/html/rfc4122

[5] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, 2006, doi: 10.1109/JSSC.2005.864128.

[6] A. Banerjee and S. W. Hussain, "An 8T single bit-line content addressable memory cell for high-performance searching applications," in *2024 international conference on microelectronics (ICM)*, 2024, pp. 1–6. doi: 10.1109/ICM63406.2024.10815912.

[7] OpenHW Group, *Core-v eXtension interface (CV-x-IF) specification.* 2023. Available: https://docs.openhwgroup.org/projects/openhw-group-core-v-xif/

[8] OpenHW Group, *CV-x-IF interface and coprocessor: CVA6 user documentation.* 2023. Available: https://cva6.readthedocs.io/en/latest/01_cva6_user/CVX_Interface_Coprocessor.html

[9] Intelligent Computing Research Group, *OpenRAM: An open-source static random access memory compiler.* 2022. Available: https://github.com/VLSIDA/OpenRAM

[10] SSCCS Foundation, "synTagma: Spatial coordinate space computing system based on tagma." SSCCS Document Suite, 2026. Available: https://docs.ssccs.org/projects/syntagma/tagma/

[11] SSCCS Foundation, "Tagma: Content-addressable structural primitive." GitHub repository (Apache 2.0), 2026. Available: https://github.com/ssccsorg/syntagma

[12] SSCCS Foundation, "synTagma benchmark suite." GitHub repository, 2026. Available: https://github.com/ssccsorg/syntagma/blob/main/sw/rust/benches/bench.rs

Appendices

A Reference Implementation

The coordinate space is implemented as a multi-crate Rust reference implementation[4]. The `Coord` type is defined in the core library and is bit-exact with the hardware decoder. The workspace is published on GitHub (Apache 2.0) [11].

A.1 Core Types

Type	Description	Key property
Coord	16-bit newtype valid in $[0, 11171]$. Three-axis decomposition, composition, Hamming distance, Unicode display.	Bit-exact with hardware decoder
CoordPath<N>	Compile-time N-element Coord array. Index path through multi-level address table. Each element is a direct array index at the corresponding tree depth.	No hashing, no equality comparison
CoordSet	Fixed-size bit array over 11,172-coordinate space. $[\texttt{u64}; 175]$ (1.4 KB, zero heap, `Copy`). Bitwise union, intersection, difference.	Single-bit ops; 175-word AND for compound axis filter

These three types are always available (no allocator required). With the `alloc` feature, the Space family below is added.

The **CoordPath** types above treat coordinate space as a tree: index paths through multi-level arrays. **CoordCube** (from `tagma-geo`) reinterprets the same CoordPath keys as **D-dimensional coordinates**, enabling proximity, bounding box, and distance queries that CoordPath alone cannot express [C]. The two access patterns share the same underlying storage; CoordCube is a zero-cost view (construction 0.96 ns, axis extraction at raw path speed 319 ps).

A.1.1 Space Family

The `CoordSpace` series provides hash-free, collision-free coordinate-indexed spaces backed by direct array addressing.

Type	Depth	Address space	Allocation	Latency	Exploration pattern
CoordSpace	1	11,172	None (inline array)	0.39 ns	General single-Coord, no_std, MCU
CoordSpace2	2	1.25×10^8	Dense heap (119 MB)	0.39 ns	Two-axis space
CoordSpace3	3	1.39×10^{12}	mmap (1.27 TB)	0.39 ns	Large-scale dense space
CoordSpace2	2	1.25×10^8	Heap (lazy)	0.90 ns	Two-axis space
CoordSpace6	6	1.94×10^{24}	Heap (lazy)	5.96 ns	UUID-scale identity
CoordSpace19	19	$\approx 2^{256}$	Heap (lazy)	47.8 ns	SHA-256-scale
DynCoordSpace	Runtime	Unlimited (`&[Coord]`)	Heap (lazy)	N/A	Variable-depth paths

[4] Tagma Software Reference Implementation in Rust: https://docs.ssccs.org/projects/syntagma/tagma/wp/impl.html

A.2 Serialization: `base11172`

A `no_std` + `alloc` crate providing Tagma's native serialization format. Every coordinate index 0..11171 maps to exactly one Unicode character (U+AC00 + index). A pair of Coords encodes a 16-bit value. The encoding is self-validating: characters outside U+AC00..U+D7AF are immediately detectable as invalid. No special characters, padding, escaping.

Code-level analysis of all types – including `Coord` bit layout, `CoordSpace` inline array with niche optimization, `CoordSet` bit iteration with `trailing_zeros`, and `CoordSpaceN` sparse tree with lazy node allocation – is in the reference implementation report Reference Implementation.

B Benchmarks

A SHA-256 engine requires approximately 10,000 gates and 64–75 cycles per operation, then needs collision resolution and dynamic resizing. UUID generation requires entropy collection and delivers probabilistic uniqueness. The Tagma decoder replaces this with a combinational decoder and a 16-bit register: one Coord covers 11,172 identifiers; six Coords (18 axes) exceed typical distributed system needs; nineteen match the SHA-256 2^{256} space.

- Lookup latency: native CoordSpace (dense array) is flat at **0.39 ns** across all depths — every Coord resolves to a single array load. The tree fallback (CoordSpaceN) scales linearly with depth: 2.69 ns at N=3, 47.8 ns at N=19 (SHA-256 scale, ↑**4.7x** vs SHA-256's 227 ns). Native CoordSpace reaches **582x** vs SHA-256. Recursive depth is bounded by schema, not data volume: 10^4 and 10^{77} entries both cost N dereferences in the fallback path, while the native dense path costs a constant 0.39 ns.
- Nonexistent prefix lookup: CoordSpace **1.65 ns** (structural, navigates to the branch and returns None) vs HashMap **23.05 ms** (↑**14.0Mx**, full scan — HashMap has no structural prefix index). Sparse get at 10M entries: CoordSpaceN2 completes all 10M operations in 44.9 ms vs HashMap 1.05 s (↑23.4x).

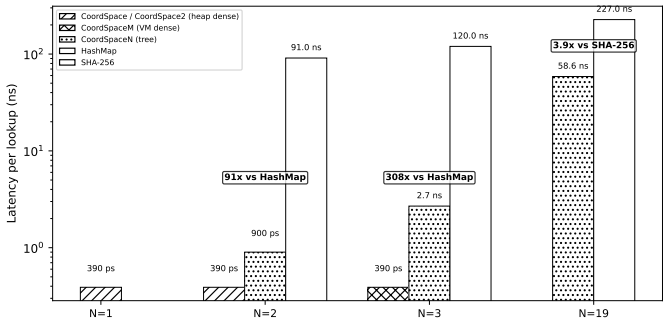

Figure 10: Identity generation latency

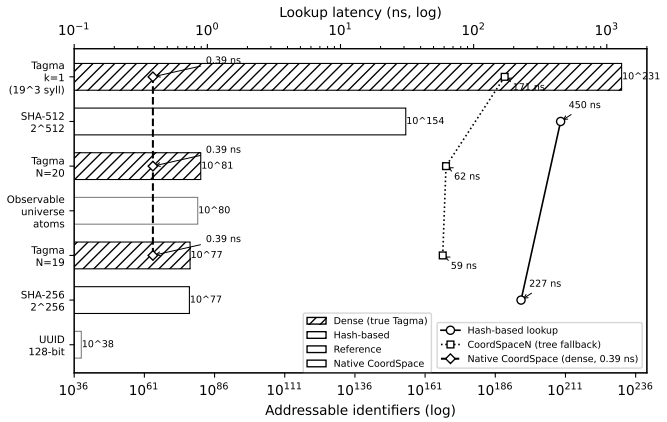

Figure 11: Addressable space (bars, log) and lookup latency (line, right axis).

- Identity generation: SHA-256 lookup costs 227 ns; the tree fallback (CoordSpaceN) reaches 2^{256} at 19 Coords for 47.8 ns (↑3.9x). The native dense path (CoordSpace, CoordSpace2, CoordSpaceM3) holds at a flat 0.39 ns.
- Address space: tree fallback lookup cost scales as O(N); native dense path is O(1) flat. Tagma recursion k=1 reaches 10^{231} identifiers (SHA-512 space × 10^{77}) at 171 ns.

Tagma assigns every point in a geometric space a structural address that is simultaneously a coordinate, an identifier, and a computation target. HashMap stores values by hashing keys by comparison. Querying this space is spatial computation: axis projection, set membership, proximity, and coordinate slicing are arithmetic operations. The figures above measure the consequence: HashMap degrades with data volume; the coordinate space does not.

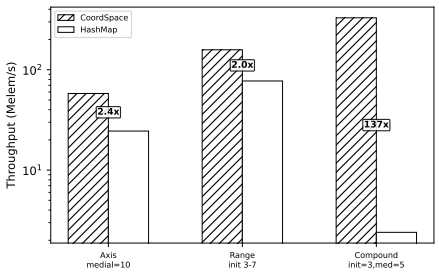

Figure 12: Spatial query throughput

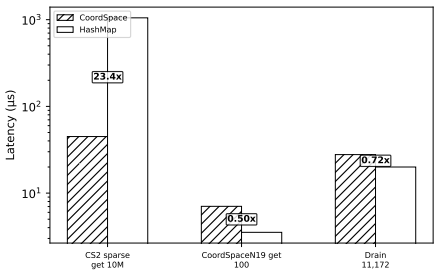

Figure 13: Edge cases: sparse get, deep get, drain

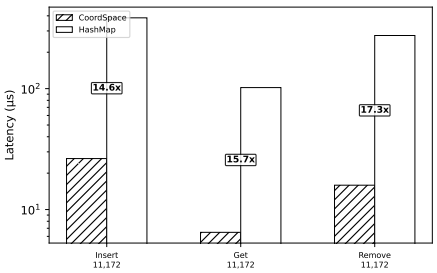

Figure 14: Bulk operations

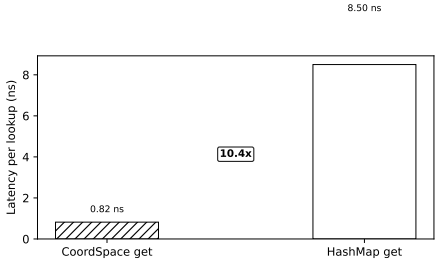

Figure 15: Single-get microbenchmark

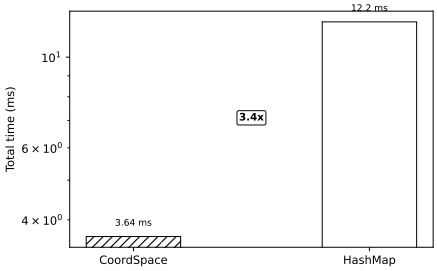

Figure 16: Mixed workload (500k ops)

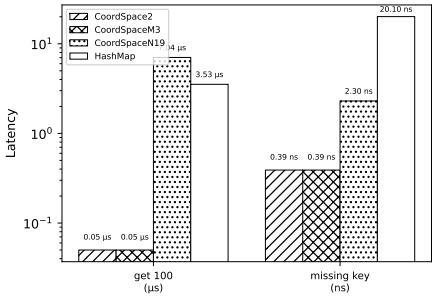

Figure 17: Deep tree: get and nonexistent key across the CoordSpace family

Rust's `std::collections::HashMap` compiles to C-grade machine code within 5-10% of theoretical CPU throughput. Whether Tagma matches or exceeds this baseline is incidental: HashMap degrades linearly with collision rate and entry count while Tagma does not. A coordinate-slice query costs the same at 10^4 entries as at 10^{77} entries: one array dereference per Coord.

Metric	SHA-256	CoordSpace (N=1)	CoordSpace2 (N=2)	CoordSpaceM3 (N=3)	CoordSpaceN6 (tree)	CoordSpaceN19 (tree)
Latency (ARMv8.4-A Firestorm)	227 ns	0.39 ns	0.39 ns	0.40 ns	5.44 ns	53.2 ns
Backing	hash	inline array	heap alloc_zeroed	mmap MAP_NORESERVE	sparse tree	sparse tree
Allocation	per-entry heap	22 KB	119 MB	1.27 TB (virtual)	per-node	per-node
Identity size	32 bytes	2 bytes	4 bytes	6 bytes	12 bytes	38 bytes
Addressable space	2^{256}	1.12×10^4	1.25×10^8	1.39×10^{12}	1.94×10^{24}	1.94×10^{77}
Collision probability	negligible (2^{-128})	zero	zero	zero	zero	zero
Native	—	Yes (dense)	Yes (dense)	Yes (dense)	No (fallback)	No (fallback)

All figures are software measurements on ARMv8.4-A Firestorm (2020), compiled with `rustc` stable in release mode. Full benchmark source is included in the repository [12].

C CoordCube: Spatial Interpretation Layer

The CoordCube layer reinterprets existing CoordPath storage keys as D-dimensional coordinates without modifying the underlying key, enabling proximity queries, bounding box enumeration, and distance metrics that fall outside the core CoordPath scope. The full design, benchmarks, and comparison with existing systems are described in the Tagma-Geo whitepaper[5].

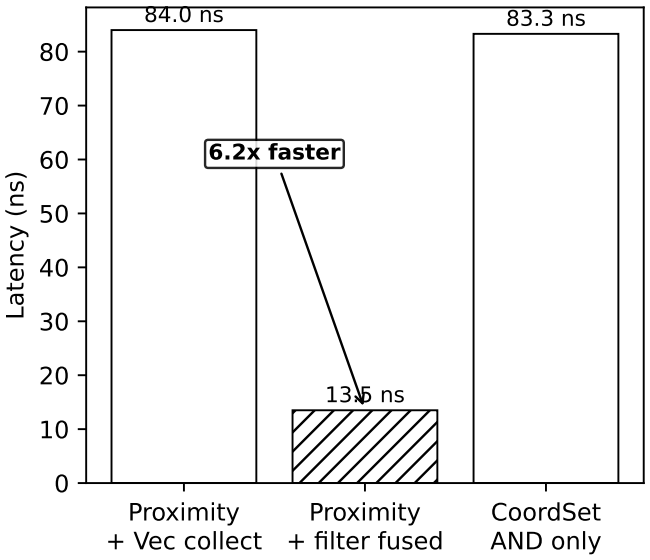

Figure 18: Compound query efficiency: iterator fusion vs collect-then-filter

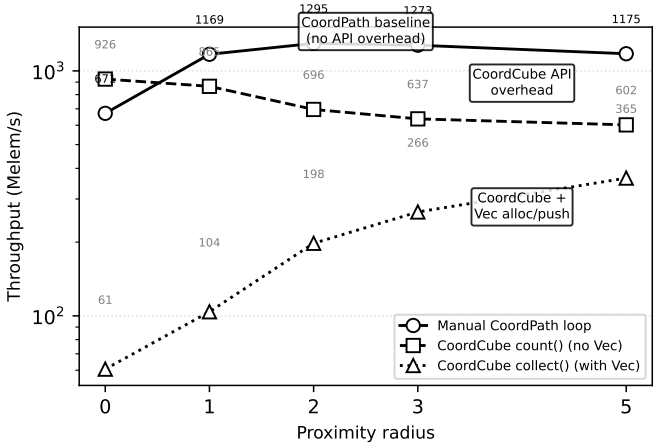

Figure 19: Proximity path generation throughput: manual CoordPath vs CoordCube

Query	CoordPath	CoordCube
Point lookup	O(k) direct	O(k) direct
Neighborhood of P	external index required	proximity(r)
Bounding box	manual loop	bounding_box()
Distance metric	manual compute	1.75 ns hamming
Scale cost (10M entries)	O(k log N)	bounded O(paths)
Empty region check	O(k log N) None	15.7 ns immediate
Compound axis filter	O(N) scan	85.7 ns AND

D Hardware Design Implications

Tagma changes the problem that hardware must solve alongside the speed at which it solves it. Conventional hardware spends area and energy on finding data through hash computation, cache tag matching, and address translation. Tagma replaces finding with knowing: the coordinate is known at encoding time, so the hardware need only decode.

Layer	Conventional approach	Tagma-based approach
ISA	Instructions compute or look up addresses	Instructions carry Tagma coordinates as direct operands
Pipeline	Branch prediction, cache miss handling	Predictable access patterns from coordinate regularity
Cache	Tag comparison, associative lookup	Direct-indexed cache lines, no tag match

[5]Tagma-Geo whitepaper (draft): https://docs.ssccs.org/projects/syntagma/tagma/geo

Layer	Conventional approach	Tagma-based approach
Accelerator	Dedicated hash unit for DHT or content addressing	Coordinate arithmetic only; hash unit eliminated
Energy	Dynamic voltage scaling to cover worst-case hash latency	Fixed, minimal decode path; predictable power

Conventional hardware searches. Tagma hardware interprets. This shifts the hardware design problem from faster computation to simpler decoding.

E SHA-256 with Tagma Encoding

SHA-256 output is uniformly distributed, so the modulo-11172 mapping to each Coord value is statistically unbiased. The overall collision probability of the 19-Coord output remains 2^{-256}, preserved from the underlying hash.

F Tagma-KV[6]: Key-Value Store on Coordinate Primitives

Tagma-KV builds a practical key-value storage engine on top of the coordinate primitives described in this document. Where the core Tagma library provides collision-free O(1) addressing within a single address space, Tagma-KV extends the model to handle legacy infrastructure requirements that fall outside the core library scope: multi-node coordinated sharding, persistence to backing stores, protocol adaptation (Redis RESP, S3 REST), and operational tooling.

The structural addressing model handles what legacy systems delegate to hash functions and index structures: key placement, collision resolution, and range partitioning. Tagma-KV supplies what the coordinate model intentionally abstracts away: durable storage, wire protocols, and cluster management.

The full design and benchmark results are described in the Tagma-KV whitepaper. Selected CoordCube benchmarks that measure KV-relevant metrics are reproduced below.

F.1 Throughput: Store Density and Proximity

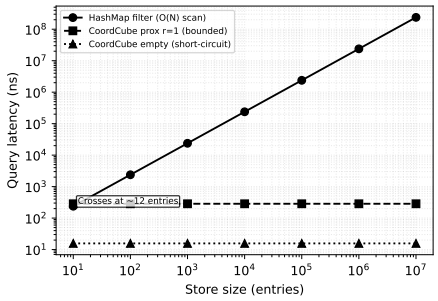

Figure 20: Query cost vs store size: CoordCube proximity vs HashMap filter

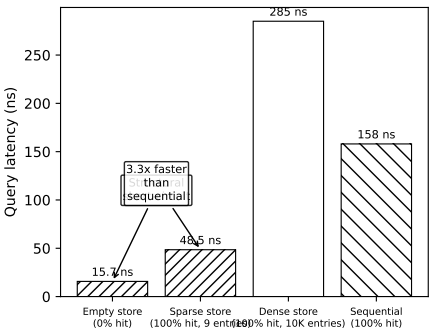

Figure 21: CoordCube vs sequential lookup: hit-rate crossover

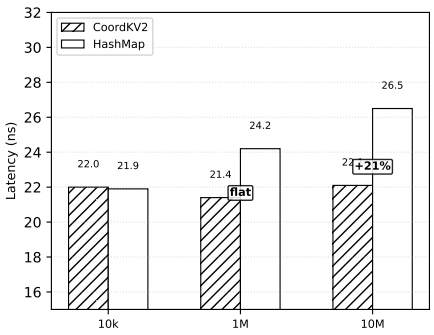

Figure 22: Scale-invariant KV get latency: CoordKV2 vs HashMap

- CoordCube proximity on dense stores adds 127 ns overhead over sequential lookup, but this overhead is dominated by Vec allocation/push (87%), not coordinate arithmetic (13%).

[6]Tagma-KV whitepaper (DOI: 10.5281/zenodo.21550431): https://docs.ssccs.org/projects/syntagma/tagma/kv

- On sparse stores, CoordCube is up to 3.3x faster than sequential lookup (48.5 ns vs 158 ns) because it avoids tree lookups for nonexistent paths.
- On empty stores, CoordCube returns immediately at 15.7 ns (pure path generation cost) — sequential lookup still pays 158 ns for 9 tree misses.
- The crossover point where CoordCube becomes faster than sequential is at roughly 55% hit rate. Below this, generating paths and checking is cheaper than looking up known paths.
- CoordCube query cost is bounded by region size (path count), not store size. HashMap spatial queries cost O(N) full scan. At 10M entries, CoordCube proximity is 285 ns vs HashMap filter at 238 ms — a million-fold advantage.
- Hierarchical queries (CoordCube proximity + manual post-filter) are faster than direct KV proximity on multi-character dimensions (547 ns vs 639 ns).

F.2 Throughput: General KV Operations

- Edge: CS2 sparse get sustains 23.4x at 10M entries; CS19 get shows 19-dereference cost (0.50x); drain is 0.72x on the full space.
- Bulk operations: CoordSpace outperforms HashMap by 14.6–17.3x across all operations on the full 11,172-entry space.
- Single-get microbenchmark isolates the per-operation cost: 0.82 ns vs 8.50 ns.
- Stress test: under 500,000 interleaved insert, get, remove, and update operations, CoordSpace completes in 3.64 ms vs HashMap 12.2 ms.
- Deep tree: CoordSpaceN19 at 100 entries shows the 19-dereference tree traversal cost (7.04 us vs 3.53 us for HashMap). Nonexistent key lookup remains depth-independent: CoordSpace2 0.39 ns, CoordSpaceN19 2.30 ns, HashMap 20.1 ns.

G Structural Enumeration in Practice: RISC-V Verification

ExaVerif[7] exhaustively verifies RISC-V custom instruction encodings. Its standard pipeline generates the full Cartesian product of field domains and filters each combination through constraint checks. At the CVA6 CV-X-IF space (5 fields, 33,554,432 raw combinations), this takes 29.7 seconds and yields 229,376 valid encodings.

The Tagma-based structural pipeline replaces post-hoc filtering with structure-preserving generation. Cross-field constraints (funct3→funct7 mapping, oneof, enable_mask) are encoded into a `DynCoordSpace<CoordSet>` during enumeration setup. The iterator visits only combinations that satisfy those constraints a priori. Constraint evaluation on visited combinations is identical to the standard pipeline.

The two charts below capture the result. The first shows the speedup across all fixtures from the standard pipeline to structural enumeration: 79x at Ibex scale, 766x at CVA6 R4 scale. The second zooms into the CVA6 encoding space, showing the full 33M space in 29.7 seconds versus 31.3 milliseconds — a 950x reduction in verification time.

Fixture	Raw space	Valid space	Density	Standard evaluate	Structural verify	Speedup
Ibex R-type	524,288	92,160	17.6%	3.66 s	46.1 ms	79x
CVA6 R4	2,097,152	12,288	0.6%	1.23 s	1.60 ms	766x
CVA6 full	33,554,432	229,376	0.7%	29.7 s	31.3 ms	950x

The speedup is not algorithmic optimization. It is a change in enumeration strategy: the standard pipeline generates all 33,554,432 combinations then filters, while the structural pipeline generates only the 229,376 valid combinations by construction. Every combination that the structural pipeline visits, it evaluates with the same constraint checks as the standard pipeline. The 99.3% of the space that is invalid is never allocated, never iterated, and never tested — this is not faster filtering; it is the absence of filtering.

[7] Project ExaVerif, Exhaustive Verification for RISC-V Custom Instructions https://docs.ssccs.org/projects/ev

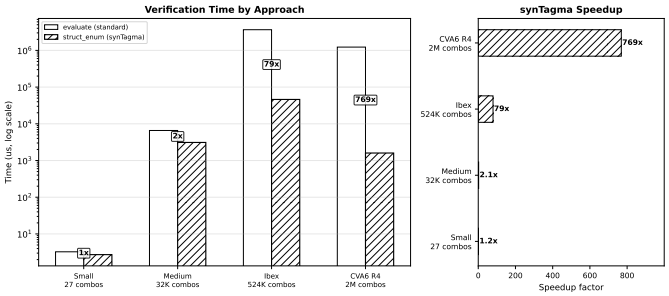

Figure 23: Verification time: standard evaluate vs structural enumeration across four fixture sizes. At CVA6 R4 2M scale, structural enumeration achieves 766x speedup.

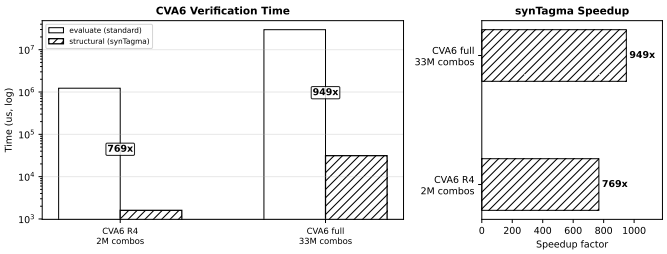

Figure 24: CVA6 CV-X-IF encoding space: standard evaluate (29.7s for 33M) vs structural_verify with all constraint checks (31.3ms), a 950x speedup.

H CoordSpace20

This is a single atom's address in a coordinate space larger than the observable universe:

> 맨가억빈힣쐭롮직랯풓쳱겨뇨됴듸뤼뮈븨싀쨔

Twenty Korean syllables (U+AC00–U+D7AF) form a coordinate path. Each syllable encodes a value drawn from 11,172 possibilities through its initial, medial, and final decompositions, yielding $11{,}172^{20} \approx 2.17 \times 10^{81}$ possible addresses. This is enough to assign a unique address to each atom in a volume 21.7 times larger than the observable universe. Yet the entire address is a 20-syllable Korean string. It is not a hash of a larger datum, but the address itself rendered in human-readable Hangul, decodable by anyone who reads Korean without a hex dump.

H.1 Syllable to Coordinate Mapping

#	Syllable	Initial	Medial	Final	Linear Index
1	맨	6 (ㅁ)	1 (ㅏ)	4 (ㄴ)	3,560
2	가	0 (ㄱ)	0 (ㅏ)	0	0
3	억	11 (ㅇ)	3 (ㅓ)	13 (ㄱ)	6,565
4	빈	7 (ㅂ)	20 (ㅣ)	8 (ㄴ)	4,684
5	힣	18 (ㅎ)	20 (ㅣ)	27 (ㅎ)	11,171
6	쐭	10 (ㅆ)	11 (ㅚ)	1 (ㄱ)	6,189
7	롮	5 (ㄹ)	8 (ㅚ)	18 (ㅍ)	3,182
8	직	12 (ㅈ)	20 (ㅣ)	1 (ㄱ)	7,617
9	랯	5 (ㄹ)	1 (ㅏ)	23 (ㅌ)	2,991
10	풓	17 (ㅍ)	18 (ㅡ)	27 (ㅎ)	10,527
11	쳱	14 (ㅊ)	5 (ㅕ)	5 (ㄴ)	8,377
12	겨	0 (ㄱ)	5 (ㅕ)	20	160
13	뇨	2 (ㄴ)	6 (ㅛ)	24 (ㅎ)	1,368
14	됴	3 (ㄷ)	9 (ㅛ)	20	2,036
15	듸	3 (ㄷ)	19 (ㅟ)	0	2,296
16	뤼	5 (ㄹ)	16 (ㅟ)	0	3,388
17	뮈	6 (ㅁ)	6 (ㅛ)	24 (ㅎ)	3,720
18	븨	7 (ㅂ)	0 (ㅡ)	20	4,136
19	싀	9 (ㅅ)	19 (ㅟ)	0	5,824
20	쨔	11 (ㅉ)	1 (ㅏ)	0	6,868

H.2 Hexadecimal View

```
0x0DE8 0x0000 0x19A5 0x124C 0x2BA3
0x182D 0x0C6E 0x1DC1 0x0BAF 0x291F
0x20B9 0x00A0 0x0558 0x07F4 0x08F8
0x0D3C 0x0E88 0x1028 0x16C0 0x1AD4
```