Project Brief

synTagma: a system where identity is coordinate and address is space.

Taeho Lee, Founder & Architect, lee@ssccs.org
SSCCS Foundation

synTagma[1] is a spatial coordinate space computing system based on core primitive Tagma, where the address is a coordinate in an N-dimensional geometric space. This is made possible by a 16-bit Unicode block allocated to a 3-axis writing system, 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 (Unicode code point), a 3-D coordinate (Axis 0, Axis 1, Axis 2), and a displayable Coord. This triple interpretation enables hash-less content addressing with zero collision probability and single-cycle combinational decoding at an estimated ~300 gates[2]. Every content-addressable system today generates identifiers through a hash function — SHA-256, UUID, hash tables. The industry's answer to this cost has been faster hash units and larger hash tables. Tagma replaces the hash function with a pronounceable Coord carrying its own coordinate system.

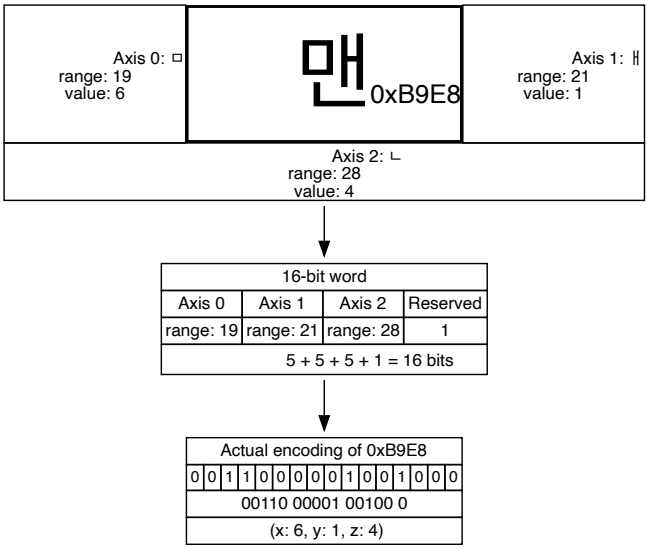

Figure 1: Coordinate 0xB9E8: three-axis coordinate (6, 1, 4) packed into a 16-bit word.

Unicode assigns each script a fixed address block. One such block (U+AC00–U+D7AF) occupies a contiguous 16-bit segment. Its encoding formula embeds three independent structural axes into every code point:

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

A hardware decoder extracts the three axis fields in one cycle. Of 65,536 possible 16-bit values, 11,172 are structurally valid and the remaining 54,364 are immediately detectable as invalid.

CoordPath chains multiple Coords into higher dimensions (6 = 18 axes, 19 = 57 axes), with each axis position representing a different application-defined dimension — region, device type, timestamp, shard. The arithmetic operations are invariant; only the axis interpretation changes per deployment.

No other Unicode script satisfies all four conditions required: a contiguous fixed-width 16-bit address range, a closed-form composition formula, a complete three-axis decomposition, and an open international standard. This composition block is unique.

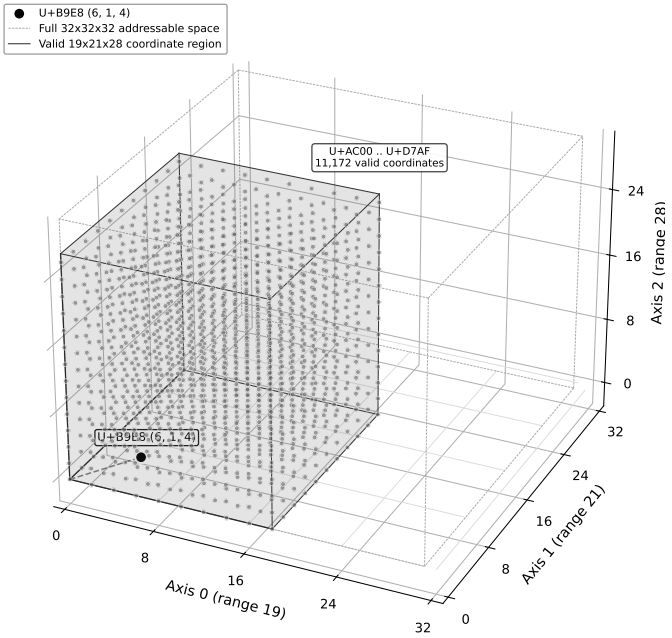

Figure 2: Three-dimensional bitcell organization. 19 x 21 x 28 lattice, 11,172 of 65,536 states valid.

The cost[3] of hashing: what Tagma removes

CoordSpace is the Rust type family that realizes the Tagma core primitive: a generic direct-address array indexed by coordinate, zero hash and collision. The benchmarks below compare its performance against standard hash maps.

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

Measured lookup latency: 0.39 ns for a single-Coord native CoordSpace (dense array, no allocator) vs 227 ns for SHA-256

[1] synTagma is an independent open-source project (Pre-release, Apache 2.0) under the SSCCS Foundation.

[2] Pre-silicon estimate based on the gate-level specification; actual gate count and cycle timing will be confirmed through standard-cell synthesis and static timing analysis with the coprocessor (e.g. RISC-V XIF).

[3] All benchmarks were run with cargo bench, criterion on ARMv8.4-A Firestorm. Code: Github

(↑**582x**). Native CoordSpace is flat across all N: 0.39 ns at every depth, because every Coord resolves to a single array load regardless of Coord count. The tree fallback (CoordSpaceN) scales linearly with N: 2.69 ns at N=3, 58.6 ns at N=19, because each level requires a heap dereference and enum match. Recursive depth is bounded by schema, not by 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 of 10M entries — 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).

This is **the elimination of hashing itself, not replacing `HashMap` as a storage** which is one of the fastest general-purpose hash based storage by C-grade machine code (not an interpreted-language benchmark). Tagma matches or exceeds this baseline is incidental: HashMap degrades linearly while Tagma does not.

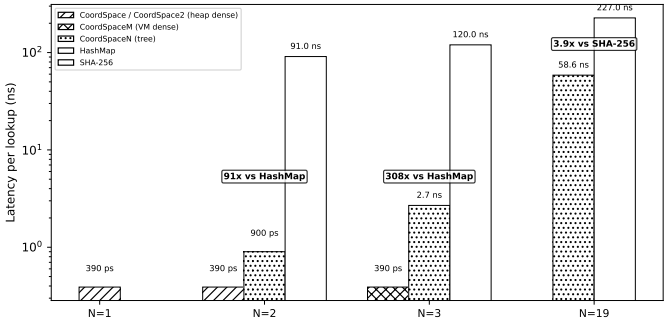

Figure 3: Identity generation latency

SHA-256 lookup costs 227 ns; the tree fallback (CoordSpaceN) reaches 2^{256} at 19 Coords for 58.6 ns (↑3.9x). The native dense path (CoordSpace / CoordSpace2 / CoordSpaceM3) holds at a flat 0.39 ns.

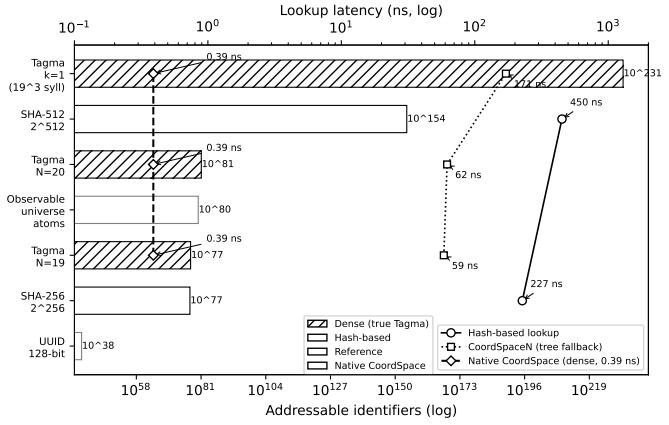

Figure 4: Addressable space (bars, log) and lookup latency (line, right axis).

Address space grows with N; the tree fallback lookup cost scales as O(N), while the native dense path is O(1) flat. Tagma recursion k=1 reaches 10^{231} identifiers (SHA-512 space × 10^{77}) at 171 ns, exceeds every hash system at a fraction of the cost.

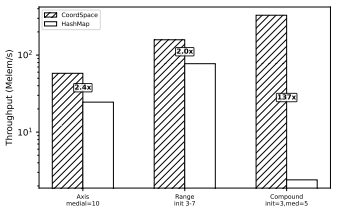

Figure 5: Spatial query throughput

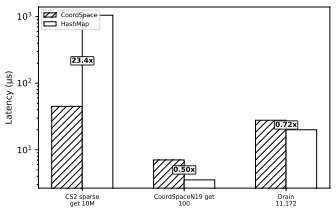

Figure 6: Edge cases: sparse get, deep get, drain

Spatial query: CoordSet bitwise AND resolves compound axis filters at 329 Melem/s — 137x faster than HashMap scan. Edge: CS2 sparse get sustains 23.4x at 10M entries; CoordSpaceN19 get shows 19-dereference cost (0.50x); drain is 0.72x on the full space.

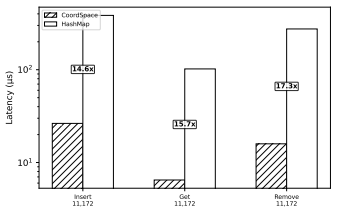

Figure 7: Bulk operations

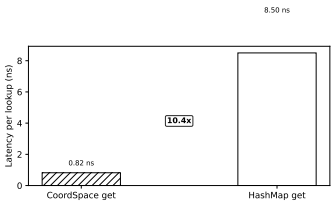

Figure 8: Single-get microbenchmark

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.

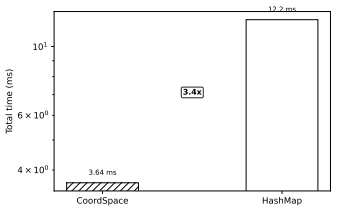

Figure 9: Mixed workload (500k ops)

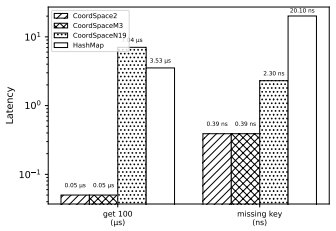

Figure 10: Deep tree: get and nonexistent key across the CoordSpace family

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: CoordSpace2 and CoordSpaceM3 (dense, N=2 and VM N=3) reach 0.05 µs for 100 gets (0.39 ns per access), CoordSpaceN19 incurs tree traversal cost (7.04 µs), and HashMap stays at 3.53 µs. Nonexistent key lookup favors dense encoding (0.39 ns) over tree depth (2.30 ns) and hash miss (20.1 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, not index scans. The figures above

measure the consequence: HashMap degrades with data volume; the coordinate space does not.

Resource efficiency at 10M scale

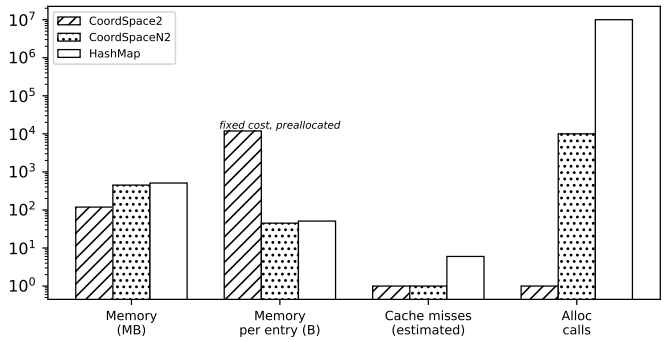

Figure 11: Resource usage at 10K entries: CoordSpace2 (dense), CoordSpaceN2 (tree), and HashMap. Dense coordinate space uses a fixed 119 MB regardless of occupancy, making per-entry cost (11900 B) high at low density but allocation-free. Tree scales per-prefix: 45 B/entry with minimal allocations. HashMap pays per-entry malloc overhead (51 B/entry) and accumulates 10M allocation calls.

Three allocation strategies produce distinct resource profiles. CoordSpace2 (dense) preallocates the full 11,172 x 11,172 grid as a flat array: 119 MB, one allocation call, one cache miss per lookup. Cost is fixed regardless of occupancy – at 10K entries the 11900 B/entry overhead is high, but at 10M entries it drops to 11.9 B/entry. CoordSpaceN2 (tree) allocates one 44 KB leaf node per written prefix (11,172-slot). Memory scales with **prefix count, not entry count**: 10001 allocations for 10K entries across a handful of prefixes, 45 B/entry at this density. HashMap allocates per-entry: 10 million insertions produce 10 million allocation calls, each with malloc overhead, bucket resizing, and rehashing.

The practical consequence: dense eliminates allocation entirely but pays a fixed memory floor. Tree memory is fixed at prefix-creation time and independent of per-prefix density. HashMap's footprint and allocation cost grow with every entry and exceed both coordinate-space strategies at scale.

Secondary benchmarks confirm the same scaling pattern across all depths. CoordSpaceN2 insert/get at 1,000 entries completes in 791 µs (insert) and 4.99 µs (get). At 100,000 entries with 1,000 unique prefixes, CoordSpaceN2 insert allocates nodes in 15.6 ms while HashMap inserts in 6.6 ms without preallocation; get is 48.2 µs vs 32.1 µs. These non-core paths follow the expected tradeoff: tree node allocation overhead at small scale inverts at large scale where HashMap's per-entry cost compounds.

Verification by exhaustive enumeration

The Tagma coordinate space is exhaustively enumerable: 65,536 possible 16-bit values, of which exactly 11,172 satisfy the composition formula. A verification harness generates all 65,536 inputs, applies the decoder specification, and records every result. The bijection is verified by enumeration: no collisions, no unassigned values within the block. Every implementation, hardware or software, can be verified against the same ground truth in milliseconds on any commodity system.

A SHA-256 engine cannot be exhaustively verified across its full input space. The Tagma coordinate space can, because it is bounded and its formula is closed-form.

Application domains

Embedded systems. The 11,172-identifier space fits in a single 22 KB no-allocator array. Every coordinate is a direct array index: one load, no hashing, no collisions, no resizing.

LLM inference cache. KV caches indexed by token prefixes use CoordPath-based direct access: lookup cost is O(N) direct array accesses with zero hash computation. Production cache sizes (10^4–10^7) are covered by 2–4 Coords.

Graph and multi-dimensional query. Each node maps to a coordinate, each edge type to a bit array. Adjacency reduces to a bitwise AND over 175 machine words — no index intersection.

General-purpose addressing. Wherever UUIDs, hash keys, or sequence numbers are used today, synTagma provides a shorter, faster, human-readable alternative with zero collision probability.

Boundaries

Tagma does not replace cryptographic primitives. SHA-256 remains for signatures, Merkle proofs, and preimage resistance. Encryption, authentication, and key derivation are outside the primitive's scope. Tagma replaces the use of hashes as structural identifiers and addresses. The two can also be combined: SHA-256 output encoded as 19 Coords is more readable than 64 hex characters while preserving the same 2^{-256} collision probability.

synTagma is built on open international standards and runs on open ISA (RISC-V) – unencumbered by proprietary licensing or regional dependencies. The Tagma primitive is a neutral, sovereign coordinate space for the global computing landscape.

Design from 1443: a writing system becomes an address space

This block was allocated in Unicode 2.0 (1996).[4] Its encoding formula provides three independent axes — Axis 0, Axis 1, and Axis 2 — the coordinate ranges used by Tagma. This structure is analogous to an apartment building numbering system: given "101", you know the floor and room without a central directory. Tagma leverages this pre-existing coordinate space as a universal address space, eliminating the need for hash-based directories.

[4] The constants 19, 21, and 28 correspond to the number of initial consonants, medial vowels, and final consonants in Hangul (invented 1443), the compositional writing system encoded in block U+AC00–U+D7AF.