

The Shibuna Discussion Process and Engineering Standards

STATE

published

CREATED

2026-09-07

LAST UPDATED

2026-09-07

INTENDED STATUS

Published

AUTHORS

Sibuna Contributors <team@sibuna.local>

DISCUSSION

Process document establishing RFC/RFD design lifecycle, TigerStyle engineering standards, and Elm-style error reporting

LABELS

documentation, process, standards

Status of This Memo

This document is an internal Sibuna discussion record authored in Typst and tracked in git. It intentionally follows RFC-style structure so the design scope, rationale, trade-offs, and operational constraints remain explicit. Documents that still use the placeholder number XXXXX are provisional drafts. Numbered SID documents are part of the permanent project record.

Contents

Abstract	1
Introduction	1
The SID Lifecycle	1
Tooling and Numbering Workflow	1
Creating a Draft	1
Promoting a Draft	1
Listing Discussions	1
Engineering Principles and Code Standards	1
Hard Structural Limits	1
Core TigerStyle Priorities	1
Elm-Style Diagnostic and Error Reporting	1
The Three Elements of an Elm-Style Error	1

Concrete Diagnostic Example	1
Implementation Architecture	1
Recoverable Errors vs Invariant Assertions	1
Compilation Targets and Verification	1
Conclusion	1
Decision	1
Request contract	1
Authentication and work	1
Policy and memory	1
Measurement gates	1
Release toolchain	1
Native platforms and release boundaries	1
Remaining limits	1
1. Abstract	9
2. Introduction	9
3. The SID Lifecycle	9
4. Tooling and Numbering Workflow	10
4.1. Creating a Draft	10
4.2. Promoting a Draft	10
4.3. Listing Discussions	10
5. Engineering Principles and Code Standards	10
5.1. Hard Structural Limits	10
5.2. Core TigerStyle Priorities	11
6. Elm-Style Diagnostic and Error Reporting	11
6.1. The Three Elements of an Elm-Style Error	11
6.2. Concrete Diagnostic Example	11
6.3. Implementation Architecture	12
6.4. Recoverable Errors vs Invariant Assertions	12
7. Compilation Targets and Verification	12
8. Conclusion	12
9. Decision	9
10. Request contract	10
11. Authentication and work	11
12. Policy and memory	11
13. Measurement gates	11
14. Release toolchain	12
15. Native platforms and release boundaries	12
16. Remaining limits	12
Decision summary	1
Data model	1
Evaluation order	1
Built-in policy	1
Policy file	1
Dynamic policies	1
Verification	1
Milestones	1
Review corrections (2026-09-07)	1
17. Decision summary	9
18. Data model	9

19. Evaluation order	10
20. Built-in policy	10
21. Policy file	11
22. Dynamic policies	12
23. Verification	12
24. Milestones	12
25. Review corrections (2026-09-07)	12
Context and problem statement	1
Engine choice: automata and tokenizers, not regular expressions	1
Design	1
Byte-class pre-scan	1
26. Context and problem statement	9
27. Engine choice: automata and tokenizers, not regular expressions	9
28. Design	10
28.1. Byte-class pre-scan	10
Context and motivation	1
Zaxonlite foundations	1
Build and configuration	1
Schema	1
Hot-path isolation	1
Read-copy-update engine slot	1
Incident ring	1
Storage thread tick	1
Campaign clustering	1
Forensics	1
Product boundary	1
Delivery record	1
Verification	1
Open items	1
Review corrections (2026-09-07)	1
29. Context and motivation	9
30. Zaxonlite foundations	10
31. Build and configuration	10
32. Schema	10
33. Hot-path isolation	11
33.1. Read-copy-update engine slot	11
33.2. Incident ring	11
33.3. Storage thread tick	11
34. Campaign clustering	12
35. Forensics	12
36. Product boundary	12
37. Delivery record	12
38. Verification	12
39. Open items	13
40. Review corrections (2026-09-07)	13
Abstract	1
Introduction and Adversary Model	1
Setting	1
Axioms	1

Cost calculus	1
Part A: The Hashcash Tier	1
Part B: Proof of Sequential Work	1
Why sequential work	1
The construction	1
Theorems	1
Why not the alternatives	1
Implementation and calibration	1
Part C: Stateless Challenges and Symmetric Tokens	1
Pseudorandom functions and message authentication	1
Stateless challenge identifiers	1
Session tokens	1
Part D: Rate Limiting by the Generic Cell Rate Algorithm	1
Part E: Robin Hood Hashing for the Spent Set	1
Part F: Linear-Time Inspection	1
Multi-pattern matching	1
The semantic layer	1
Vectorised matchers	1
Part G: Read-Copy-Update Engine Slots	1
Part H: Load-Adaptive Difficulty	1
Part I: Campaign Clustering by Feature Hashing	1
Verification and Benchmark Gates	1
Security Considerations	1
Open Problems	1
References	1
41. Abstract	9
42. Introduction and Adversary Model	10
42.1. Setting	10
42.2. Axioms	10
42.3. Cost calculus	11
43. Part A: The Hashcash Tier	11
44. Part B: Proof of Sequential Work	12
44.1. Why sequential work	12
44.2. The construction	12
44.3. Theorems	13
44.4. Why not the alternatives	14
44.5. Implementation and calibration	15
45. Part C: Stateless Challenges and Symmetric Tokens	15
45.1. Pseudorandom functions and message authentication	15
45.2. Stateless challenge identifiers	16
45.3. Session tokens	17
46. Part D: Rate Limiting by the Generic Cell Rate Algorithm	17
47. Part E: Robin Hood Hashing for the Spent Set	18
48. Part F: Linear-Time Inspection	19
48.1. Multi-pattern matching	19
48.2. The semantic layer	20
48.3. Vectorised matchers	20
49. Part G: Read-Copy-Update Engine Slots	21
50. Part H: Load-Adaptive Difficulty	21

51. Part I: Campaign Clustering by Feature Hashing	22
52. Verification and Benchmark Gates	22
53. Security Considerations	23
54. Open Problems	23
55. References	24
Abstract	1
Introduction and Motivation	1
Operator Workflows	1
Design Principles	1
Don't make me think	1
Fast judgement: System 1	1
Careful judgement: System 2	1
Elegance	1
Principle audit by page	1
Terminology and Scope	1
Goals and Non-Goals	1
Goals	1
Non-Goals	1
Architecture Overview	1
Modules and directories	1
Process model and the isolation contract	1
Startup from the command line	1
The Serve Kernel	1
Authentication and Authorization	1
The Telemetry Pipeline	1
The sampler	1
The incident tap	1
The challenge funnel	1
Traffic sampling and top-k rankings	1
Data-plane changes this record requests	1
Cluster aggregation	1
The Real-Time Protocol	1
Data Model	1
GeoIP	1
Live earth globe on the landing page	1
Cluster Management	1
The Interface Module	1
Pages	1
Wireframes	1
The Build Pipeline	1
Security Considerations	1
Performance Budget	1
Delivery Plan	1
Verification	1
Resolved Design Decisions	1
Operational Contracts and Trade-offs	1
Retained observations	1
Rule observations	1
Management consistency	1

Browser and peer state	1
Acceptance and Open Questions	1
Query and authentication observation bounds	1
Launch scope (2026-09-11)	1
Post-launch completion (2026-09-11)	1
Interface budget review (2026-09-11)	1
References	1
56. Abstract	9
57. Introduction and Motivation	10
58. Operator Workflows	10
59. Design Principles	11
59.1. Don't make me think	11
59.2. Fast judgement: System 1	12
59.3. Careful judgement: System 2	13
59.4. Elegance	13
59.5. Principle audit by page	13
60. Terminology and Scope	14
61. Goals and Non-Goals	15
61.1. Goals	15
61.2. Non-Goals	15
62. Architecture Overview	16
62.1. Modules and directories	16
62.2. Process model and the isolation contract	17
62.3. Startup from the command line	18
63. The Serve Kernel	19
64. Authentication and Authorization	20
65. The Telemetry Pipeline	21
65.1. The sampler	21
65.2. The incident tap	22
65.3. The challenge funnel	23
65.4. Traffic sampling and top-k rankings	23
65.5. Data-plane changes this record requests	24
65.6. Cluster aggregation	24
66. The Real-Time Protocol	24
67. Data Model	26
68. GeoIP	28
68.1. Live earth globe on the landing page	29
69. Cluster Management	30
70. The Interface Module	30
70.1. Pages	31
71. Wireframes	33
72. The Build Pipeline	43
73. Security Considerations	44
74. Performance Budget	45
75. Delivery Plan	46
76. Verification	47
77. Resolved Design Decisions	48
78. Operational Contracts and Trade-offs	49
78.1. Retained observations	49

78.2. Rule observations	49
78.3. Management consistency	50
78.4. Browser and peer state	50
79. Acceptance and Open Questions	51
79.1. Query and authentication observation bounds	51
79.2. Launch scope (2026-09-11)	51
79.3. Post-launch completion (2026-09-11)	52
79.4. Interface budget review (2026-09-11)	53
80. References	53
Abstract	1
Introduction and Motivation	1
Terminology and Scope	1
Problem Statement and Architectural Challenges	1
The Gap in the Current Codebase	1
The Asymmetric Challenges of AI Bot Identification	1
System Invariants and Design Principles	1
Detailed Design	1
1. The Multi-Tier Bot Classification Engine	1
1.1 Provider and Bot Taxonomy	1
1.2 Comprehensive AI Bot Signatures Database	1
1.3 Sub-Microsecond Radix CIDR Verification	1
1.4 Behavioral and Structural Heuristics	1
2. High-Efficiency Streaming Analytics: Algorithms, Mathematics, and Storage Decoupling	1
2.1 The Analytics Dilemma: Why Traditional OLAP Fails Edge WAFs	1
2.2 Axiomatic Foundations of Edge Analytics	1
2.3 Contention-Free Cache-Line Striped Accumulation	1
2.4 Bounded Stochastic Sampling and Poisson Ring Dynamics	1
2.5 Space-Saving Heavy-Hitters Algorithm for Path Scraping	1
2.6 Micro-RRD Pre-Aggregated Rollups and Zaxonlite Storage Decoupling	1
2.7 Long-Term Telemetry Egress Strategy	1
3. Operator Console Dashboard Interface	1
3.1 Operator Screen Wireframes	1
3.2 Visual Components	1
3.3 Granular Tabular Components	1
3.4 Actionable Policy Integration	1
4. Elm-Style Operator Error and Diagnostic Reporting	1
Verification and Testing Matrix	1
Security Considerations	1
Rollout and Operational Plan	1
References	1
81. Abstract	9
82. Introduction and Motivation	9
83. Terminology and Scope	10
84. Problem Statement and Architectural Challenges	11
84.1. The Gap in the Current Codebase	11
84.2. The Asymmetric Challenges of AI Bot Identification	12
85. System Invariants and Design Principles	12
86. Detailed Design	13

86.1. 1. The Multi-Tier Bot Classification Engine	13
86.1.1. 1.1 Provider and Bot Taxonomy	13
86.1.2. 1.2 Comprehensive AI Bot Signatures Database	14
86.1.3. 1.3 Sub-Microsecond Radix CIDR Verification	15
86.1.4. 1.4 Behavioral and Structural Heuristics	16
86.2. 2. High-Efficiency Streaming Analytics: Algorithms, Mathematics, and Storage	
Decoupling	16
86.2.1. 2.1 The Analytics Dilemma: Why Traditional OLAP Fails Edge WAFs	16
86.2.2. 2.2 Axiomatic Foundations of Edge Analytics	17
86.2.3. 2.3 Contention-Free Cache-Line Striped Accumulation	17
86.2.4. 2.4 Bounded Stochastic Sampling and Poisson Ring Dynamics	18
86.2.5. 2.5 Space-Saving Heavy-Hitters Algorithm for Path Scraping	19
86.2.6. 2.6 Micro-RRD Pre-Aggregated Rollups and Zaxonlite Storage Decoupling . . .	20
86.2.7. 2.7 Long-Term Telemetry Egress Strategy	21
86.3. 3. Operator Console Dashboard Interface	22
86.3.1. 3.1 Operator Screen Wireframes	22
86.3.2. 3.2 Visual Components	24
86.3.3. 3.3 Granular Tabular Components	25
86.3.4. 3.4 Actionable Policy Integration	25
86.4. 4. Elm-Style Operator Error and Diagnostic Reporting	26
87. Verification and Testing Matrix	26
88. Security Considerations	26
89. Rollout and Operational Plan	27
90. References	27
Abstract	1
Introduction	1
Terminology and Scope	1
Problem Statement	1
Goals and Non-Goals	1
Goals	1
Non-Goals	1
State of the Art	1
Protocol Rules	1
Detailed Design	1
The decoder	1
Admission: decoding into the head buffer	1
Forwarding	1
Cost	1
Verification and Testing	1
Security Considerations	1
Rollout and Operational Plan	1
Implementation Record (2026-10-02)	1
References	1
91. Abstract	9
92. Introduction	9
93. Terminology and Scope	10
94. Problem Statement	10
95. Goals and Non-Goals	10
95.1. Goals	10

95.2. Non-Goals	11
96. State of the Art	11
97. Protocol Rules	11
98. Detailed Design	12
98.1. The decoder	12
98.2. Admission: decoding into the head buffer	13
98.3. Forwarding	14
98.4. Cost	15
99. Verification and Testing	15
100. Security Considerations	15
101. Rollout and Operational Plan	16
102. Implementation Record (2026-10-02)	16
103. References	17

1. Abstract

The *sibuna* monorepo requires a durable decision-record process for architectural, cryptographic, protocol, security, and operational changes across the high-performance Web AI Firewall and anti-crawler daemon. The project adopts Shibuna Discussions, or SID, as RFC/RFD-style Typst documents that support structured technical reasoning, long-lived references, high-quality PDF archival output, and an HTML discussion website.

This memo defines the lifecycle of an SID, the placeholder numbering workflow, the enforced coding guidelines and structural limits, the TigerStyle engineering priorities, and the project's Elm-style diagnostic error reporting standard.

2. Introduction

A Shibuna Discussion is a Typst document stored in git under `docs/sid/records`. Each discussion is part design memo, part review artifact, and part historical record. The structure intentionally follows IETF RFCs and Oxide RFDs because those formats force explicit scope, status, rationale, trade-offs, and operational constraints instead of relying on ephemeral chat messages or implicit assumptions.

SID is used for topics such as:

- Reverse proxy architecture and zero-allocation socket I/O loops
- Cryptographic proof-of-work algorithms (HashX, Argon2id, SHA-256 SIMD)
- Multi-pattern bot detection, SIMD Aho-Corasick, and Radix CIDR routing
- Session token formats, Ed25519 signing, and cookie binding
- State caching, lockless decay maps, and distributed store integrations
- Performance benchmarking methodology and reproducible comparison gates

3. The SID Lifecycle

Every SID progresses through explicit lifecycle states:

1. **Prediscussion (prediscussion)**: A draft with the placeholder number XXXXX. The problem statement, design overview, and open questions are being formulated.
2. **Discussion (discussion)**: The draft has been promoted to a four-digit number (NNNN), registered in `registry.typ`, and opened for review by the working group.
3. **Accepted (accepted)**: The working group agrees on the architectural direction and technical commitments.

4. **Committed (committed)**: Implementation has completed and merged to main, satisfying the defined verification and benchmark gates.
5. **Published (published)**: Normative design or process record that reflects active system guarantees.
6. **Abandoned (abandoned)**: The proposal was superseded or rejected.

4. Tooling and Numbering Workflow

To prevent merge conflicts and premature number squatting across concurrent branches, drafts use the placeholder prefix XXXXX-`<slug>.typ`.

4.1. Creating a Draft

To create a new draft:

```
zig build sid-new -- <slug>
```

This copies `docs/sid/template/rfc-template.typ` to `docs/sid/records/XXXXX-<slug>.typ`, stamps the creation date, and provides a target for local compilation.

4.2. Promoting a Draft

When a draft is ready for working group discussion:

```
zig build sid-promote -- <slug>
```

The tool:

1. Determines the next available four-digit sequence number NNNN.
2. Renames XXXXX-`<slug>.typ` to NNNN-`<slug>.typ`.
3. Sets state to discussion and status to Open for Discussion.
4. Updates `docs/sid/registry.typ` with the new entry metadata.
5. Updates `docs/sid/bundle.typ` for HTML and PDF generation.

4.3. Listing Discussions

```
zig build sid-list
```

Displays registered discussions, active placeholder drafts, and detects consistency warnings between the filesystem and `registry.typ`.

5. Engineering Principles and Code Standards

Sibuna inherits and strictly enforces the engineering priorities established across `insan.ai` and `paxos-zig`: **Safety** → **Performance** → **Developer Experience**, in that exact order.

5.1. Hard Structural Limits

All source code files (`*.zig`) across `build.zig`, `apps/`, `libs/`, and `tools/` must adhere to the following non-negotiable structural constraints, automated via `tools/check-style.awk` and verified on every `zig build fmt`:

1. **Function Length (*le*70 Lines of Code)**: Functions must not exceed 70 lines of code. Blank lines, comment-only lines, and docstrings are explicitly excluded from the line count calculation. Long functions must be decomposed into focused, cohesive helper functions placed immediately adjacent to their caller.
2. **Line Length (*le*99 Characters for Code)**: Lines in code files must not exceed 99 characters. Wrapping must prioritize readability and clear syntactic boundaries.

3. **Documentation Line-Length Exclusion:** The 99-character line length limit is **explicitly not applied** to docs/ (Typst documents, diagrams, and markdown documentation). Prose, formatted tables, and mathematical formulas must not be artificially broken by code line limits.
4. **File Length (*≤*1408 Lines of Code):** Each source code file must not exceed 1408 lines of code (excluding comments and blank lines). When a module approaches this boundary, it must be split across logical sub-packages.
5. **Forbidden Tab Characters:** Tab characters (`\t`) are forbidden in all source code files; standard 4-space indentation is required.

5.2. Core TigerStyle Priorities

- **Zero-Allocation Hot Path:** The firewall’s primary classification and challenge verification path must never allocate dynamic memory. Memory pools, arena buffers, and socket connection rings are allocated statically at startup or scoped to thread-local stack frames.
- **Explicit Control Flow:** Control flow must remain flat and shallow. Return early when preconditions fail to keep the happy path unnested. Avoid hidden side effects, callbacks, or complex meta-programming in performance-critical paths.
- **State Invariants Positively:** Use assertions (`std.debug.assert(...)`) to verify preconditions, postconditions, and loop invariants. State invariants positively in code comments before implementing their state transitions.
- **Handle Every Error Explicitly:** Never swallow, ignore, or discard errors. Errors must be handled at the boundary where they occur or propagated up with domain-specific diagnostic context.
- **Explain the “Why” in Comments:** Comments must not narrate syntax or restate what code does; they must explain why design invariants, memory bounds, or protocol ordering rules exist.

6. Elm-Style Diagnostic and Error Reporting

Sibuna adopts Elm-style error reporting (Compiler and Operator Errors for Humans) across all daemon subsystems, CLI commands, and HTTP client interactions. When an error occurs, the system must not emit an opaque numeric code or bare error name; it must provide a structured, operator-oriented explanation with actionable recovery directions.

6.1. The Three Elements of an Elm-Style Error

Every diagnostic consists of three mandatory components:

1. **A Distinct Visual Boundary:** A prominent uppercase title framed with ASCII hyphens, establishing an unambiguous header (e.g., `-- INVALID NONCE -----`).
2. **A Plain-English Explanation:** A concise description explaining what failed within the state machine and why the current operation was rejected.
3. **An Actionable Hint / Alternative:** A line prefixed with `Hint:` providing concrete direction on what the operator or client can do instead to resolve the error.

6.2. Concrete Diagnostic Example

```
-- CHALLENGE EXPIRED -----
```

The issued proof-of-work challenge has passed its 30-minute validity window.

Hint: Request a new challenge from `/challenge/make` and re-run the solver worker.

6.3. Implementation Architecture

Elm-style error handling is implemented as a core architectural service in:

- `libs/core/src/diagnostic.zig`: Exposes `Diagnostic` struct and `diagnostic.write(writer, title, message, hint)` to render formatted error blocks to any output stream.
- `libs/core/src/errors.zig`: Exposes `explainError(err: anyerror) []const u8`, mapping domain errors across challenge, token, network, policy, and store subsystems into structured explanations.

6.4. Recoverable Errors vs Invariant Assertions

The system strictly distinguishes between two failure classes:

- **Recoverable Operating Errors**: Expected events in untrusted network environments (expired tokens, invalid nonces, unrecognized headers, full rate-limit buckets). These are returned via Zig error sets and accompanied by human-friendly Elm-style explanations and hints.
- **Invariant Violations**: Internal programming bugs or impossible states (corrupted ring buffer pointers, unreachable switch arms). These halt execution immediately via `std.debug.assert(...)` in safe builds.

7. Compilation Targets and Verification

`build.zig` drives Typst compilation and verification:

- `zig build sid`: Builds PDFs for all registered records into `docs/build/`.
- `zig build sid -Dshd=<number_or_slug>`: Compiles a single record.
- `zig build sid-index`: Compiles the registry index PDF.
- `zig build sid-site`: Generates the HTML bundle into `docs/build/sid-site/`.
- `zig build fmt`: Runs `zig fmt --check` and `tools/check-style.sh` to enforce all structural limits.
- `zig build test`: Runs all library unit tests, including diagnostic formatting and error explanation checks.

8. Conclusion

By combining version-controlled Typst design records (SID), rigorous structural limits, TigerStyle engineering discipline, and Elm-style human-friendly diagnostics, Sibuna guarantees long-term codebase maintainability, extreme operational reliability, and sub-microsecond anti-crawler defense.