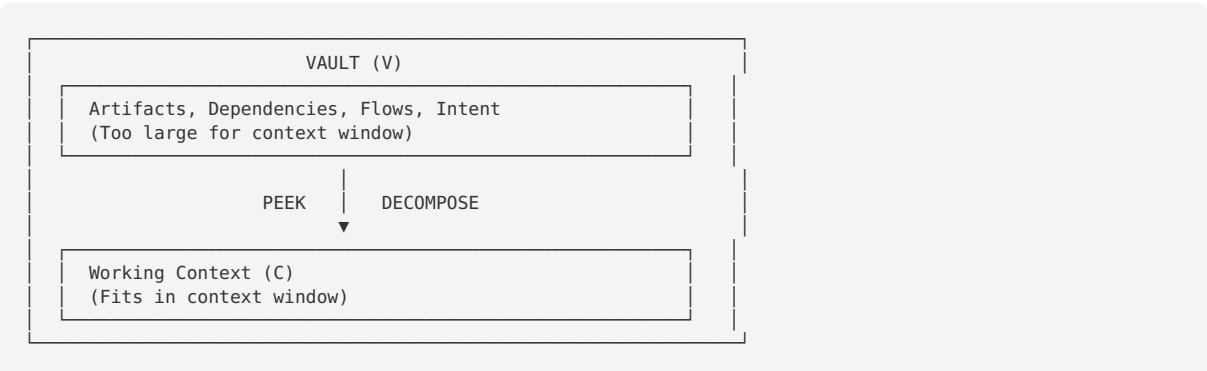
h-DSL: Context Vault for Code Navigation (v2)

Purpose

An h-DSL derived from h-M3 for modeling project structure and semantics, incorporating RLM principles for handling codebases that exceed context windows. The vault serves as an **external environment** that the AI interacts with programmatically—peeking, decomposing, and recursing rather than loading wholesale.

Core Principle: Codebase as External Environment

Following RLM's key insight: **treat the codebase as a variable in an external environment**, not as content to feed directly into the model.



The AI never loads the full vault. It: 1. Gets metadata about V (size, structure, summaries) 2. PEEKs into specific parts 3. DECOMPOSEs complex parts into sub-problems 4. RECURSEs over sub-parts with fresh context 5. JOINs results back

---

h-M3 Mapping

```
┌──────────────┬─────────────────┬──────────────────┬──────────────────┐
│  TELOS       │  UNDERSTANDING  │  RELEVANCE       │  why this code   │
├──────────────┼─────────────────┼──────────────────┼──────────────────┤
│  DYNAMICS    │  CONTEXT        │  NAVIGATION      │  how to explore  │
├──────────────┼─────────────────┼──────────────────┼──────────────────┤
│  STRUCTURE   │  ARTIFACT       │  DEPENDENCY      │  what connects   │
├──────────────┼─────────────────┼──────────────────┼──────────────────┤
│  SUBSTANCE   │  SYMBOL         │  INTENT          │  what means what │
└──────────────┴─────────────────┴──────────────────┴──────────────────┘
```

Prose Mode Correspondence

```
┌──────────────┬──────────────┬──────────────────────────────────┐
│ Prose Mode   │ Vault Concept│ Operation                        │
├──────────────┼──────────────┼──────────────────────────────────┤
│ DESCRIPTIVE  │ artifact.intent │ GROUND  (what is this code?)  │
│ DIRECTIVE    │ navigation      │ FOLLOW  (how to explore?)     │
│ EVALUATIVE   │ understanding   │ JUDGE   (enough context?)     │
│ RATIONALE    │ relevance       │ ACCEPT  (why this path?)      │
│ INFERENTIAL  │ dependency      │ DERIVE  (what follows?)       │
└──────────────┴──────────────┴──────────────────────────────────┘
```

Primitives

SUBSTANCE

SYMBOL (TERM)

```
symbol : Path | Name | Signature | Type | Tag
ops    : resolve, match, compare
```

INTENT (PROSE)

```
intent : Description | Rationale | Constraint | Pattern
modes  : descriptive | inferential
ops    : GROUND, DERIVE
```

STRUCTURE

ARTIFACT

```
artifact:
  kind     : module | file | class | function | type | config | flow
  path     : Symbol
  symbols  : Symbol[]
  intent   : Intent
  summary  : Intent           # One-line for PEEK
  size     : lines | complexity
  tags     : Tag[]
```

DEPENDENCY

```
dependency:
  kind   : imports | calls | extends | implements | configures | tests
  from   : Artifact
  to     : Artifact
  weight : critical | normal | weak
  intent : Intent             # Why this dependency exists
```

DYNAMICS

CONTEXT

```
context:
  task     : Task             # What we're trying to understand
  focus    : Artifact[]       # Currently examining (in window)
  loaded   : Artifact[]       # Fully loaded with details
  peeked   : Artifact[]       # Summary only (metadata)
  frontier : Artifact[]       # Candidates to explore
  excluded : Artifact[]       # Ruled out
  stack    : Context[]        # Parent contexts (for recursion)
  memo     : Map<Task, Understanding>  # Cached sub-results
```

NAVIGATION

```
navigation:
  kind   : peek | focus | expand | trace | filter | decompose | recurse | join | backtrack
  from   : Context
  to     : Context
  reason : Intent
```

TELOS

UNDERSTANDING

```
understanding:
  task       : Task
  required   : Artifact[]     # Must understand
  confidence : 0.0..1.0
  gaps       : Intent[]       # What's still unclear
  verified   : boolean        # Has been checked
```

RELEVANCE

```
relevance:
  strategy : depth_first | breadth_first | guided | parallel
  weights  : { task_match, centrality, recency, change_freq, complexity }
  filter   : Intent           # What to exclude
```

Navigation Operations

PEEK

Quick scan of artifact summary without full load. Returns metadata + one-line intent.

```
peek:
  target  : Symbol | Artifact
  returns : { path, kind, summary, size, tags, dep_count }
  cost    : O(1)
```

FILTER

Narrow search space before exploration. Eliminates obviously irrelevant artifacts.

```
filter:
  scope   : Artifact[]
  by      : Tag | Pattern | Intent
  exclude : true | false
  returns : Artifact[] (reduced)
```

FOCUS

Load full artifact details into working context.

```
focus:
  target : Artifact
  load   : [symbols, intent, behavior, dependencies]
  prune  : oldest from focus if over limit
```

EXPAND

Add related artifacts to frontier based on dependencies.

```
expand:
  from   : Artifact
  by     : dependency.kind[]
  filter : relevance.filter
  limit  : N
  add_to : frontier
```

TRACE

Follow dependency chain in specific direction.

```
trace:
  from      : Artifact
  direction : forward | backward | both
  kind      : calls | imports | extends | all
  depth     : N
  returns   : Artifact[] (ordered by distance)
```

DECOMPOSE

Break complex artifact into sub-parts for separate processing.

```
decompose:
  target   : Artifact
  by       : functions | classes | sections | concerns
  returns  : SubTask[]

subtask:
  part     : Artifact (sub-part)
  question : Intent (what to understand about this part)
```

RECURSE

Spawn sub-context for sub-task. Fresh context window, inherits memo.

```
recurse:
  subtask   : SubTask
  inherit   : memo, excluded
  push      : current context to stack
  returns   : Understanding (of sub-part)
```

JOIN

Merge sub-context results back into parent context.

```
join:
  results   : Understanding[]
  strategy  : merge | intersect | union
  pop       : restore parent from stack
  update    : parent.understanding, parent.memo
```

VERIFY

Check understanding is correct with focused re-examination.

```
verify:
  claim      : Understanding
  by         : re-trace | cross-check | counter-example
  confidence : 0.0..1.0
  update     : understanding.verified
```

BACKTRACK

Remove artifact from focus, optionally exclude from future consideration.

```
backtrack:
  remove  : Artifact
  exclude : boolean
  reason  : Intent
  restore : previous context state
```

Composition Operators

Navigation operations compose using h-M3 operators:

Sequence (;)

```
focus(A) ; expand(A) ; trace(A→calls)
```

Execute in order. Each sees result of previous.

Choice (|)

```
search(query) | trace(known→deps)
```

Select based on relevance. Use search if query clear, trace if starting point known.

Iterate (*)

```
(peek ; filter ; focus)*
```

Repeat until JUDGE returns sufficient understanding.

Parallel (‖)

```
trace(A→calls) ‖ trace(A→imports)
```

Execute concurrently, merge results. Useful for independent explorations.

Composed Patterns

```
# Depth-first exploration
pattern: (focus ; expand ; filter)*

# Recursive decomposition
pattern: decompose ; (recurse)* ; join

# Verified understanding
pattern: (focus ; expand)* ; verify ; JUDGE

# Parallel trace with merge
pattern: (trace→calls ‖ trace→imports) ; join
```

Context Building Protocol

Phase 1: ORIENT

```
orient:
  input  : task (user story, question, bug)
  GROUND : Extract keywords, entities, actions from task
  DERIVE : Infer likely artifact types and locations
  output : hypothesis[] (ranked by confidence)
```

Phase 2: FILTER

```
filter_phase:
  input  : vault, hypothesis[]
  peek   : top N artifacts matching hypothesis
  filter : exclude obviously irrelevant (wrong domain, test-only, deprecated)
  output : candidates[] (narrowed search space)
```

Phase 3: EXPLORE

```
explore:
  loop:
    - select  : highest relevance from candidates
    - focus   : load artifact details
    - GROUND  : understand what this artifact does
    - DERIVE  : infer connections to task
    - expand  : add dependencies to candidates
    - JUDGE   : sufficient understanding?

  branch:
    - if complex(artifact):
        decompose ; recurse* ; join
    - if gaps remain:
        DERIVE next artifacts ; continue loop
    - if confident:
        exit to VERIFY
```

Phase 4: VERIFY

```
verify_phase:
  input  : understanding
  check  : re-examine critical artifacts with fresh eyes
  cross  : verify dependencies are correctly traced
  output : understanding.verified = true | gaps[]

  if gaps:
    return to EXPLORE with gaps as new hypotheses
```

Phase 5: ACCEPT

```
accept:
  input    : verified understanding
  ACCEPT   : justify why this context is sufficient
  output   : final context { focus, loaded, confidence }
  cache    : memo[task] = understanding
```

Protocol Summary

```
ORIENT → FILTER → (EXPLORE ; VERIFY)* → ACCEPT

Where EXPLORE = (focus ; GROUND ; DERIVE ; expand ; JUDGE)*
      with decompose ; recurse* ; join for complex artifacts
```

Artifact Catalog

Module Entry

```
module:
  path    : src/main/plugins
  summary : "Plugin system for extending Puffin"
  size    : 4 files, 1200 lines

  intent: |
    Provides isolated extension points for Puffin functionality.
    Each plugin runs in its own context with controlled API surface.

  exports:
    PluginLoader  : "Discovers and validates plugins from directory"
    PluginManager : "Lifecycle management, activation, shutdown"
    PluginContext : "Sandboxed API for plugin code"

  patterns:
    registry   : "Central registration of handlers/components"
    lifecycle  : "activate() / deactivate() hooks"
    isolation  : "Each plugin gets own PluginContext instance"

  deps:
    → ipc-handlers : configures (registers plugin IPC)
    → database     : weak (some plugins use DB)
    ← plugins/*    : implements (plugins use this API)
    ← main.js      : calls (startup initialization)

  hotspots:
    plugin-manager.js:initialize : "Complex async startup"
    plugin-context.js:_cleanup   : "Resource cleanup on deactivate"

  tags: [core, extensibility, ipc, lifecycle]
```

Function Entry

```
function:
  path      : src/main/puffin-state.js:loadUserStories
  summary   : "Load stories from SQLite, error if not initialized"
  signature : async () → UserStory[]
  complexity: low

  intent: |
    Single source of truth for user story retrieval.
    Enforces database initialization—no silent fallback.

  behavior:
    pre  : database.isInitialized() == true
    post : returns UserStory[] from SQLite
    err  : throws if database not initialized

  deps:
    → database.userStories.findAll : calls
    → _loadUserStoriesFromSqlite   : calls (internal)
    ← open()                       : called_by (init)
    ← IPC:puffin:loadState         : called_by (renderer)

  tags: [persistence, sqlite, user-stories, strict]
```

Flow Entry

```
flow:
  name    : plugin-activation
  summary : "Plugin discovery to running state"

  intent: |
    Orchestrates the full plugin lifecycle from directory scan
    through validation, context creation, and activation.

  steps:
    1: PluginLoader.loadPlugins    → "Scan plugins directory"
    2: PluginLoader.validatePlugin → "Check manifest, entry point"
    3: PluginManager.initialize    → "Create PluginContext per plugin"
    4: Plugin.activate(context)    → "Run plugin's activate hook"
    5: ViewRegistry.register       → "Register UI views if any"

  branches:
    validation_fail : skip → emit(plugin:validation-failed)
    disabled        : skip → emit(plugin:skipped)
    activate_error  : skip → emit(plugin:activation-failed), continue

  invariants:
    - "Failed plugins don't block others"
    - "All plugins get context even if activation fails"

  tags: [lifecycle, startup, plugins, async]
```

Compact Notation

For inline vault references:

```
@m(plugins)                    → Module: src/main/plugins
@f(puffin-state:loadStories)   → Function entry
@flow(plugin-activation)       → Flow entry
@peek(database)                → Quick summary
@trace(PluginLoader→calls:2)   → 2-level call trace
@deps(PluginContext)           → All dependencies
@tags(sqlite,persistence)      → Artifacts with tags
@filter(!test,!deprecated)     → Exclude patterns
```

Memoization

Avoid re-exploring previously understood areas:

```
memo:
  key    : hash(task, artifacts)
  value  : Understanding
  ttl    : until artifact.modified changes
  scope  : session | persistent

lookup:
  before : any navigation operation
  check  : memo[similar_task]?
  reuse  : if confidence > threshold

store:
  after  : ACCEPT
  save   : memo[task] = understanding
```

Index Card Summary

```
╔═══════════════════════════════════════════════════════════════╗
║              h-DSL: Context Vault v2 (RLM-Informed)           ║
╠═══════════════════════════════════════════════════════════════╣
║ PRINCIPLE: Vault as external environment, not context content ║
╠═══════════════════════════════════════════════════════════════╣
║ SUBSTANCE   SYMBOL (path,sig,tag)   INTENT (purpose,pattern)  ║
║ STRUCTURE   ARTIFACT (unit)         DEPENDENCY (connection)   ║
║ DYNAMICS    CONTEXT (working+stack) NAVIGATION (transition)   ║
║ TELOS       UNDERSTANDING (goal)    RELEVANCE (selection)     ║
╠═══════════════════════════════════════════════════════════════╣
║ RLM OPERATIONS                                                ║
║   peek      : O(1) metadata scan                              ║
║   filter    : narrow search space                             ║
║   decompose : break into sub-tasks                            ║
║   recurse   : spawn sub-context                               ║
║   join      : merge sub-results                               ║
║   verify    : check understanding                             ║
╠═══════════════════════════════════════════════════════════════╣
║ NAVIGATION                                                    ║
║   focus     : load full details                               ║
║   expand    : add deps to frontier                            ║
║   trace     : follow dep chain                                ║
║   backtrack : remove, restore                                 ║
╠═══════════════════════════════════════════════════════════════╣
║ COMPOSITION                                                   ║
║   N₁ ; N₂   : sequence                                        ║
║   N₁ | N₂   : choice (relevance selects)                      ║
║   N*        : iterate (until JUDGE satisfied)                 ║
║   N₁ ‖ N₂   : parallel (merge results)                        ║
╠═══════════════════════════════════════════════════════════════╣
║ PROTOCOL                                                      ║
║   ORIENT → FILTER → (EXPLORE ; VERIFY)* → ACCEPT              ║
║   with decompose ; recurse* ; join for complex artifacts      ║
╠═══════════════════════════════════════════════════════════════╣
║ MEMO: cache understanding, reuse across similar tasks         ║
╚═══════════════════════════════════════════════════════════════╝
```

Design Rationale

Why External Environment?

The codebase is too large for context windows. RLM shows that treating it as an external variable with programmatic access outperforms both truncation and retrieval. The vault is the variable; navigation operations are the programmatic access.

Why Recursive Decomposition?

Complex artifacts (large modules, intricate flows) can't be understood atomically. DECOMPOSE + RECURSE + JOIN mirrors RLM's chunking pattern—break into manageable parts, understand each, synthesize.

Why Composition Operators?

Navigation is not a sequence of independent steps. Operations combine: search THEN trace, filter OR expand, (peek ; focus)* until done. h-M3's operators give this compositionality.

Why Memoization?

The same sub-problems recur. Understanding "how plugins work" shouldn't require re-exploration every time. Memo caches understanding keyed by task+artifacts.

Why Verify?

Understanding can be wrong. VERIFY forces re-examination before committing. RLM's "answer verification" pattern—use sub-calls to check main-call conclusions.

DERIVE Connects Everything

- Task → likely artifacts (abductive)
- Artifact → relevant dependencies (deductive)
- Gaps → next exploration (abductive)
- Evidence → sufficient understanding (inductive)

DERIVE is the reasoning that turns navigation into understanding.