DRAFT ARCHITECTURE DOCUMENTATION

Personal AI Platform: Key Architecture Design

An Implemented, Human-Governed Multi-Agent Architecture

Local inference, dedicated coding, event-driven coordination, and multimodal production

DRAFT ARCHITECTURE DOCUMENT · V0.5

A working personal AI platform spanning local inference, durable agent execution, and editable artifact production.

WEB REVISION

21 October 2026

SERVED CONTEXT

524,288 tokens

ARCHITECTURE AND INTEGRATION

John Pezzulli

DOCUMENT SCOPE

Implemented system and measured paths

DOCUMENT CONTENTS

Executive summary

Data sovereignty in the Penny execution path

Implemented agentic capabilities

1. System requirements and architecture boundary

2. Physical deployment and service placement

3. Penny Royal inference runtime

4. HiCache, NIXL, and long-context reuse

5. Context engineering and human-approved skill learning

6. Event-driven multi-agent execution and human review

7. Identity, authorization, and native execution

8. Agent Library and authenticated file delivery

9. Connected processing services

10. Representative workflows

11. Human supervision and review

12. Operational visibility

Appendix A. Interface and state reference

Penny Royal implementation

Executive summary

This is a draft and may have minor inaccuracies.

This is the personal AI platform I use for sustained engineering, research, and creative work. It serves several people, runs model inference on my own hardware, and turns that inference into agents that can inspect sources, use tools, produce editable files, and continue an assignment across pauses and review.

Penny Royal is the customized SGLang runtime I built for shared local inference. Open WebUI provides the multi-user browser work surface. Pi provides a dedicated coding environment. Customized Hermes supplies an independent harness for selected tasks. A standalone Penny executor coordinates durable assignments through OWUI or Pi, while Codex supplies the separate supervisor and review surface. Native Linux identities own tool execution; Versity Agent Library retains file originals; image, speech, and document tools extend the same working system.

Data sovereignty is a core property of work executed entirely through Penny. Model inference, conversations and coding sessions, tool execution, media processing, and durable files remain on operator-controlled hosts. Connected research supplies outside information through the requests sent to search providers and websites. The OpenAI-backed Codex supervisor is a separate model and conversation path.

My runtime work has brought generation speed to **nearly 2× the original stock setup**. The architecture combines a 524,288-token served context window, reusable prefix state across GPU memory, host RAM and SSD, identity-aware tools, event-driven task coordination, and authenticated file delivery. The following chapters show the components and the paths that connect them: from an interactive conversation or coding assignment to model execution, native tools, retained state, and a finished result.

Concern	Component	Responsibility
Shared inference	Penny Royal / customized SGLang	Model execution, long context, speculation, and reusable prefix state
Browser work surface	Personal Open WebUI Fork	Authentication, chats, context assembly, tool interaction, and file display
Coding work environment	Pi	Coding sessions, repository tools, isolated worktrees, and reviewable changes
Selected-task harness	Customized Hermes	Its own prompts, profiles, memory, sessions, tools, and artifact workflow, sharing Penny Royal inference
Durable coordination	Standalone Penny executor	Task identity, lifecycle, capacity, steering, results, return notifications, and post-review closeout
Supervision	Human owner and Codex	Objectives, review, feedback, and consequential authority
Native execution	Identity router, Open Terminal, and coding identity	Commands and working files under the authorized Linux account
Durable files	Versity Agent Library	User-scoped originals and selected outputs for OWUI workflows
Specialized capabilities	AI Tools, ComfyUI, Dubline, and shared document tools	Images, speech, research, and editable artifacts

Data sovereignty in the Penny execution path

The operator controls the hosts that assemble Penny's model requests, run inference, retain application and task state, execute tools, process media, and store files. Open WebUI, the standalone executor, Pi, Customized Hermes, and native terminals run on room101. Penny Royal and its cache tiers run on thegrid. Versity stores durable originals on thegrid; ComfyUI and Dubline use Neomatrix for image and speech processing.

Data or execution	Operator-controlled location
Model requests, token generation, and reusable model state	OWUI, Pi, or selected Hermes request assembly; Penny Royal inference and GPU, RAM, and SSD cache tiers
User conversations and display files	Open WebUI application state on room101
Assignment status, steering, and compact results	Standalone executor ledger on room101
Coding sessions, source changes, and task outputs	Pi sessions and dedicated worker workspaces on room101
Selected Hermes tasks	Native Hermes harness, memory, sessions, tools, and working state on room101
User tool processes, browser profiles, and working files	Mapped native Linux accounts on room101
Uploaded originals and selected library outputs	Versity Agent Library on thegrid
Image generation and editing	AI Tools on thegrid and ComfyUI on Neomatrix, returning results to the authenticated work surface
Speech recognition and synthesis	Dubline on Neomatrix, with OWUI handling the user session
Editable documents and presentations	Native workspace, shared tools, and authenticated file delivery

A search provider receives the query sent to it; a visited website receives ordinary browser requests. Penny evaluates the returned material using local inference. Codex has its own OpenAI-backed model interaction and saved conversation, separate from this Penny-local path.

Figure S1 — Penny data sovereignty boundary

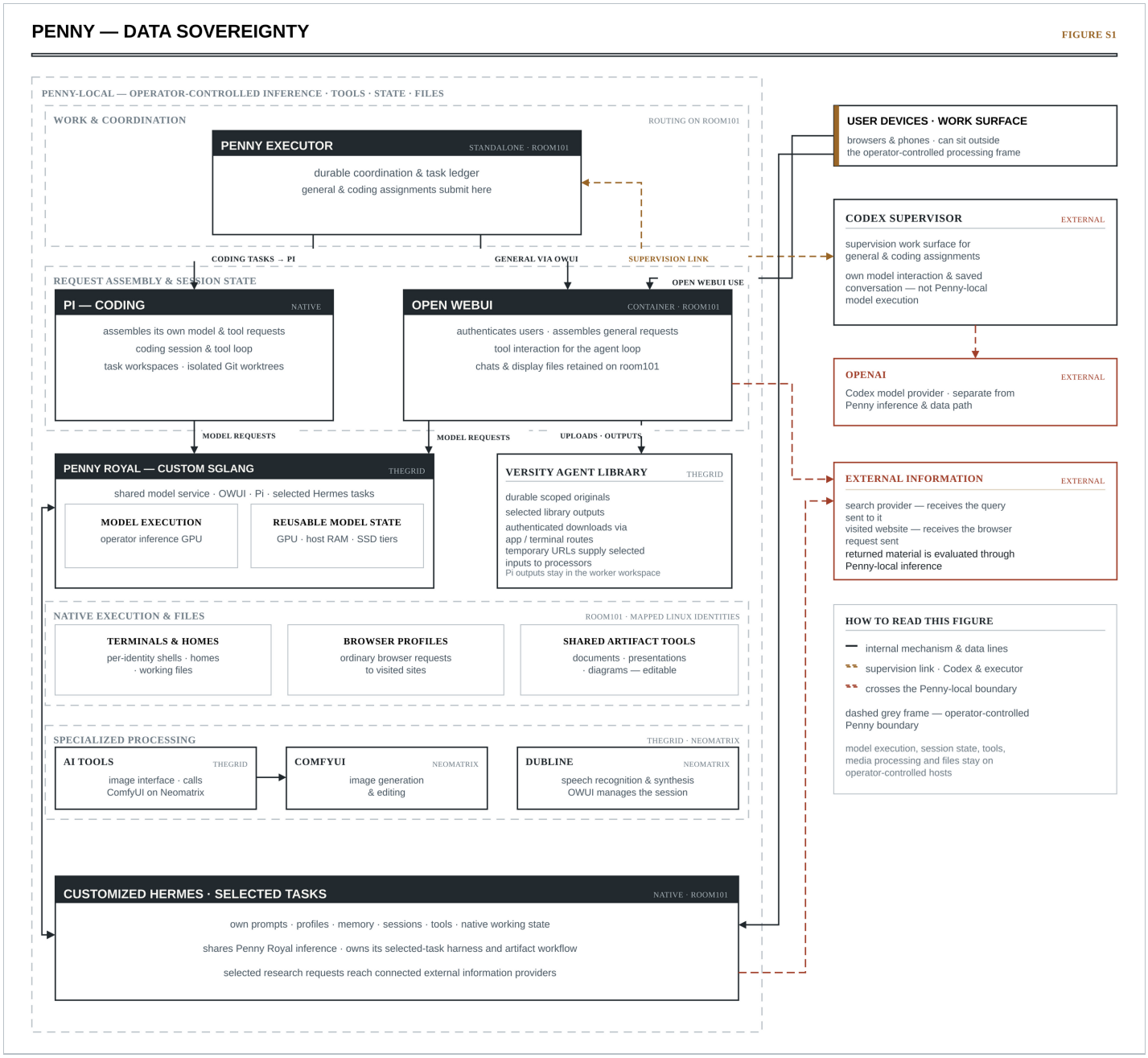


Figure S1 — Penny data sovereignty boundary

The local S3 interface is Versity storage. User grants and native identities determine which tools and objects an account can reach. OWUI and terminal file routes provide authenticated delivery to people, while temporary storage URLs supply selected inputs to processors.

Implemented agentic capabilities

Penny can pursue an assignment through multiple model and tool turns, retain the work when it pauses, use bounded helpers, and return a result for review. The browser and coding environments share inference while maintaining their own conversations or sessions, tool loops, and working state.

Capability	Mechanism
Interactive multi-user work	OWUI accounts, model and tool grants, saved chats, and native user workspaces
Durable delegated work	Executor task identity, ledger, retained execution state, and resume path
Dedicated coding	Pi’s native coding tools, isolated Git worktrees, and reviewable outputs
Event-driven return	An actionable task transition queues a notification to the originating supervisor
Bounded collaboration	OWUI subagents, Pi’s one native async helper, and independent supervisory review for separable work
Context management	Layered instructions, selected skills, request usage display, and compaction
Personal skill proposals	A procedure can be staged for its owner’s approval and later discovery
Multimodal production	Images, speech, documents, spreadsheets, presentations, and editable diagrams
Operational visibility	Aggregate metrics on Prometheus/Grafana and a view-only Pi execution dashboard

These capabilities meet in real workflows. A general assignment can inspect a set of files and create an editable presentation through OWUI. A coding assignment can use Pi to examine a repository, change source in an isolated worktree, and return a patch. Both can pause for a decision and resume under the same durable task identity.

1. System requirements and architecture boundary

The workload is substantial, repeated agent work: multi-turn engineering, long-context analysis, repository and browser interaction, image and document production, and reuse of files across conversations. Several people use the platform with different capabilities and native identities.

The system separates the model service from the environments that use it. Penny Royal owns inference and reusable computation. OWUI owns authenticated conversations and browser interaction. Pi owns coding sessions and its repository tool loop. Customized Hermes owns the harness and working state for its selected tasks. The executor owns assignments that outlive a single response. Native accounts own commands and working files; Agent Library owns durable originals for the OWUI file path.

Workload and component responsibilities

Workload	Component response
Long, repeated context	A large served window, request compaction, and tiered prefix-state reuse
Work that pauses or outlives a supervisor turn	A durable task record, saved execution state, stable workspace, and return notification
Different users and permissions	OWUI grants, explicit identity routing, and native account ownership
Repository engineering	Pi sessions, dedicated coding identity, isolated worktrees, and reviewable changes
Files reused across sessions	Library originals separated from chat display copies and working copies
Image, speech, and document production	Specialized processors connected to authenticated work and file routes

The **data plane** carries model requests, tool inputs and outputs, media, and files. The **control plane** carries authentication, grants, task lifecycle, steering, notifications, and human decisions. A notification can wake the supervisor without carrying the full work trace. A signed processing URL can supply an image to a renderer while a different, authenticated route serves the result to the user.

Shared capability installations supply the tools; native identities supply their writable state. A browser or document renderer can be installed once while user profiles, downloads, source changes, and working documents remain under their own accounts.

Figure 1 — Logical architecture

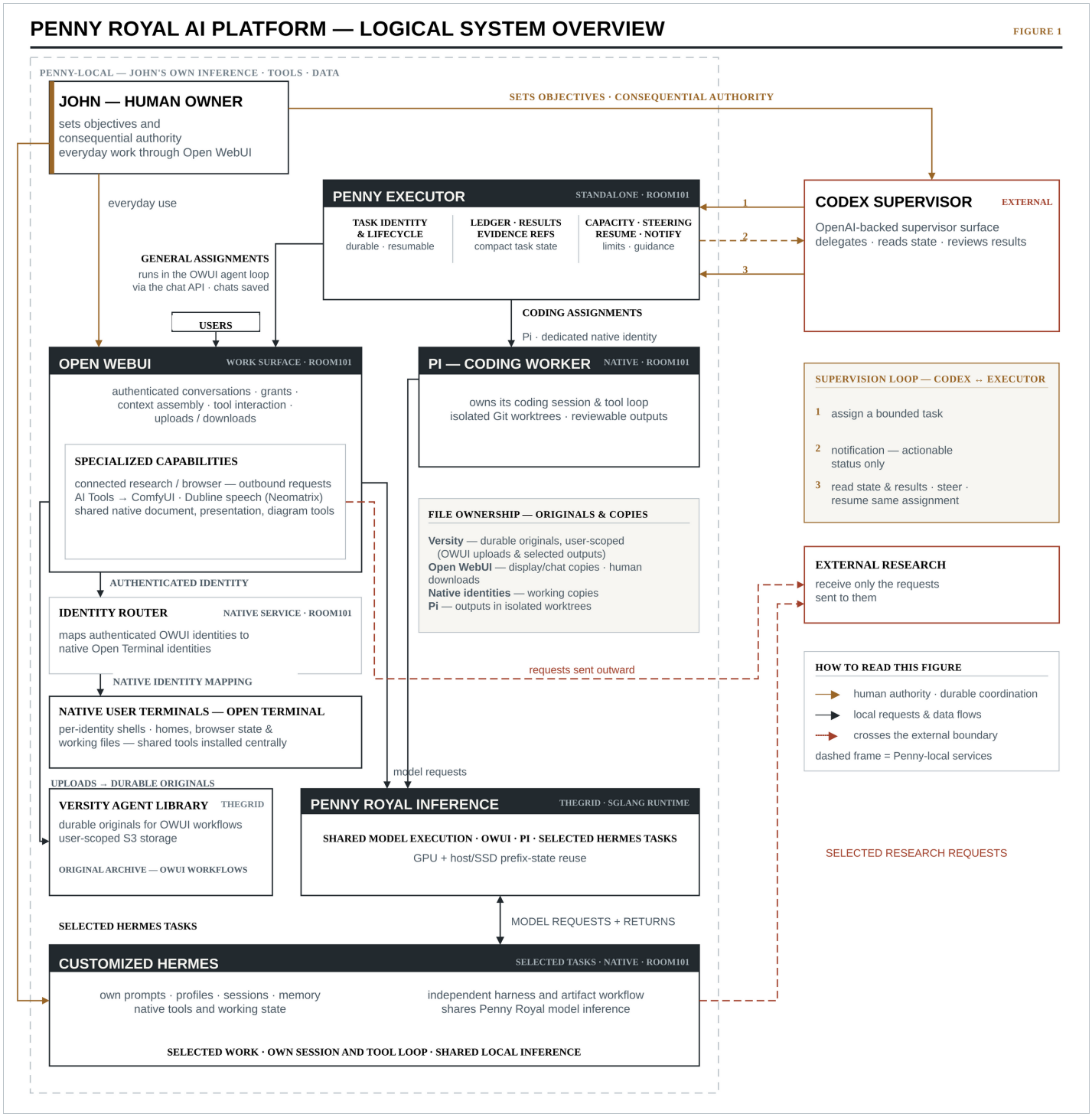


Figure 1 — Logical architecture

The overview shows the standalone executor feeding OWUI and Pi, with Customized Hermes handling selected tasks through its independent harness. All three share Penny Royal inference. Identity routing, native execution, specialized capabilities, and file storage complete the working paths.

2. Physical deployment and service placement

The platform uses two physical machines and one application VM. **Thegrid** hosts the main inference GPU, shared storage, ingress, and **room101**, a VM on that server. **Neomatrix** is a separate workstation supplying image and speech processing as well as the usual Codex Desktop work surface.

Host or boundary	Components
Thegrid physical server	Penny Royal inference, GPU/host/SSD cache tiers, model files, Caddy ingress, AI Tools, and Versity Agent Library
Room101 VM on thegrid	OWUI application container; standalone native executor, Pi, Customized Hermes, identity router, and native terminals
Neomatrix workstation	ComfyUI, Dubline, and Codex Desktop
External model and research services	Codex’s OpenAI-backed model path; connected search providers and websites

Host and execution boundaries

Thegrid’s RTX PRO 6000 Blackwell serves the language-model workload. Model files and the NIXL cache filesystem stay with that runtime. The room101 VM carries the browser application, task coordination, and native execution without requiring its own inference GPU.

OWUI runs as a rootless Podman application container. The executor, Pi, Customized Hermes, identity router, and Open Terminal instances are native processes outside that container. OWUI reaches the router through the host gateway, and the router selects the terminal mapped to the authenticated user. The coding route runs under its dedicated native identity.

Neomatrix’s RTX 5090 supplies a second GPU domain. AI Tools on thegrid calls ComfyUI there for image generation and editing; OWUI calls Dubline there for speech recognition and synthesis. Codex Desktop also runs on the workstation, with its own external model connection. Language-model, image, and speech workloads therefore have distinct execution and resource paths.

Browsers and phones enter through Caddy on thegrid and authenticate in OWUI. User downloads pass through OWUI or its terminal proxy. Durable originals remain in Versity storage, separate from the chat database, native working files, and inference cache.

Figure 2 — Deployment architecture

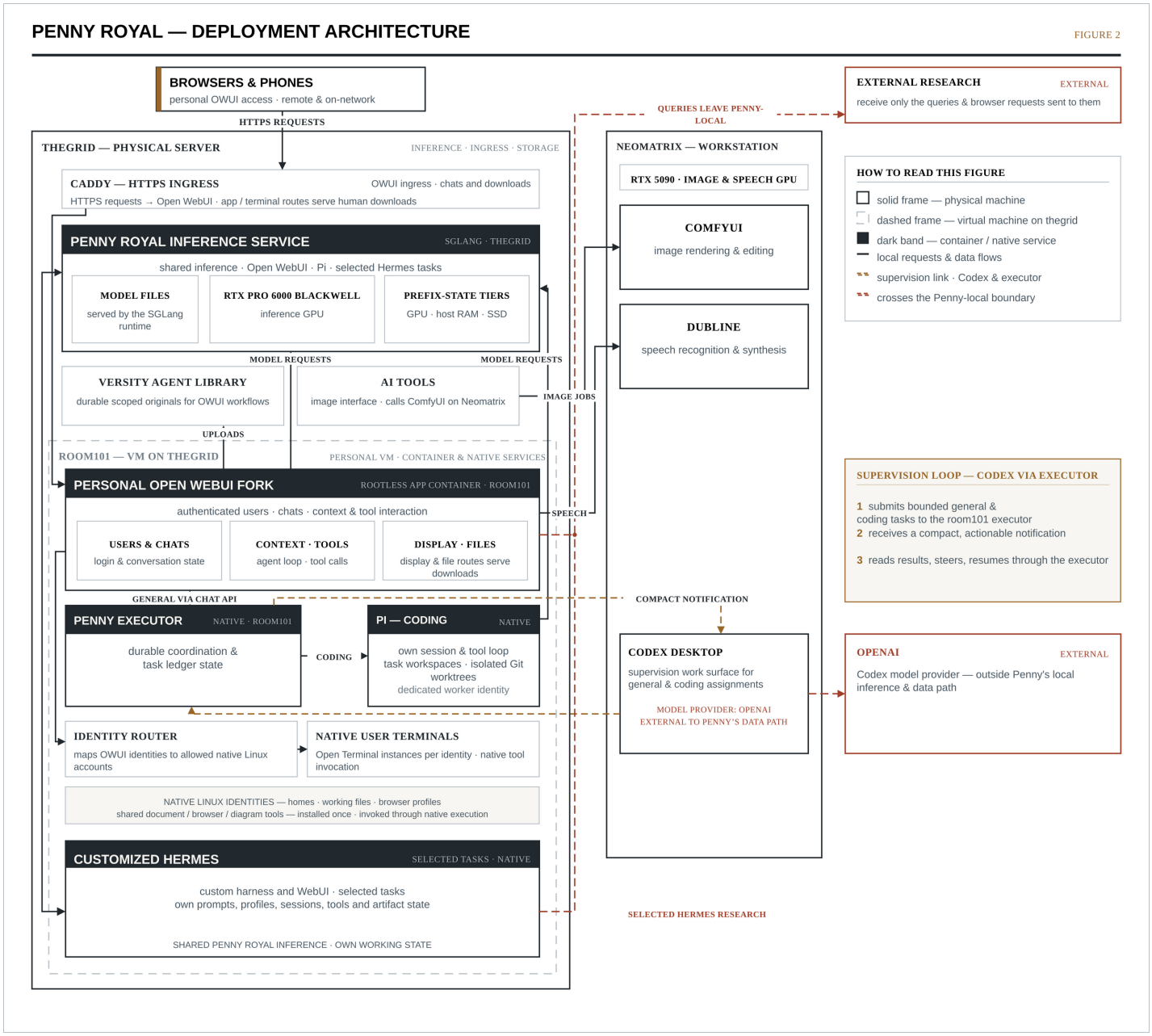


Figure 2 — Deployment architecture

The placement view nests room101 inside thegrid and distinguishes its application container from native coordination and execution. Cross-host calls connect inference, image processing, speech, storage, and the supervisor.

3. Penny Royal inference runtime

Penny Royal is my customized SGLang-derived inference runtime. I built it to make sustained, long-context agent work practical on one RTX PRO 6000 Blackwell GPU: fast generation, a 524,288-token served request context, and reusable model state that survives movement between GPU memory, host RAM, and local SSD.

Shared inference, independent work environments

Open WebUI, Pi, and Customized Hermes call the same OpenAI-compatible model endpoint. OWUI assembles authenticated conversations and their tools. Pi assembles coding sessions and repository work. Customized Hermes is used for selected tasks, with its own prompts, profiles, memory, sessions, tools, and artifact workflow. Each environment owns the context it sends and the actions it takes with the response.

That separation allows several agent environments to share one accelerated model service while keeping their own working state. The standalone executor coordinates durable OWUI and Pi assignments; Hermes has its own session and tool loop.

Fast generation on one GPU

Penny Royal 3.1 runs the Qwen3.8 Flash-Next NVFP4 profile by default, with the Flash-Next DFlash2 profile as the explicit optional alternative. Penny Royal delivers **nearly 2× the generation speed** of my original stock runtime on this system. The engineering combines model-specific execution paths, kernels suited to the SM120 GPU, efficient precision and memory layouts, and speculative decoding. Those improvements operate alongside the large context window and tiered cache, and my public Penny Royal repository carries the current profiles and full measurements.

On 21 October 2026, Penny Royal 3.1's native NEXTN MTP/FR-Spec profile, using my OrcaRouter Qwen3.8 Flash-Next ModelOpt NVFP4 checkpoint on one RTX PRO 6000 Blackwell, averaged **252.83 output tokens/s** through a completed coding workflow and **216.23 output tokens/s** through a completed ten-slide presentation workflow. Both ran at medium reasoning with one generating request and block-8 verification. These active generation means include reasoning; prefill, tool execution, rendering, and idle time are separate. The current measurements carry the full settings and results.

The maintained default is native NEXTN multi-token prediction with FR-Spec. A target runs the request and its own prediction path proposes candidate steps, so there is no separate draft checkpoint to load or keep in step. DFlash2 is an optional external drafter over the same target weights and the same cache hierarchy. A separate draft model proposes candidates, and FlashInfer on the SM120 GPU verifies them.

Profile	Speculation path	Architectural role
Default	Native NEXTN multi-token prediction with FR-Spec	The target's own prediction path proposes candidate steps; no separate draft model is involved.
Optional	DFlash2 over the same target weights and cache hierarchy	A separate draft model proposes candidates and FlashInfer verification on the SM120 target decides what is accepted.

In both configurations the target remains authoritative over accepted tokens, and speculative width follows active concurrency: a wider batch of active requests narrows the width spent per request. The GPU key-value pool is shared and sized automatically at startup inside the profile memory budget. The acceleration comes from doing useful work ahead of verification and making each execution phase cheaper. Correct continuation requires the model state to match the accepted tokens after every verification step.

Figure 3a — Inside the selected inference profile

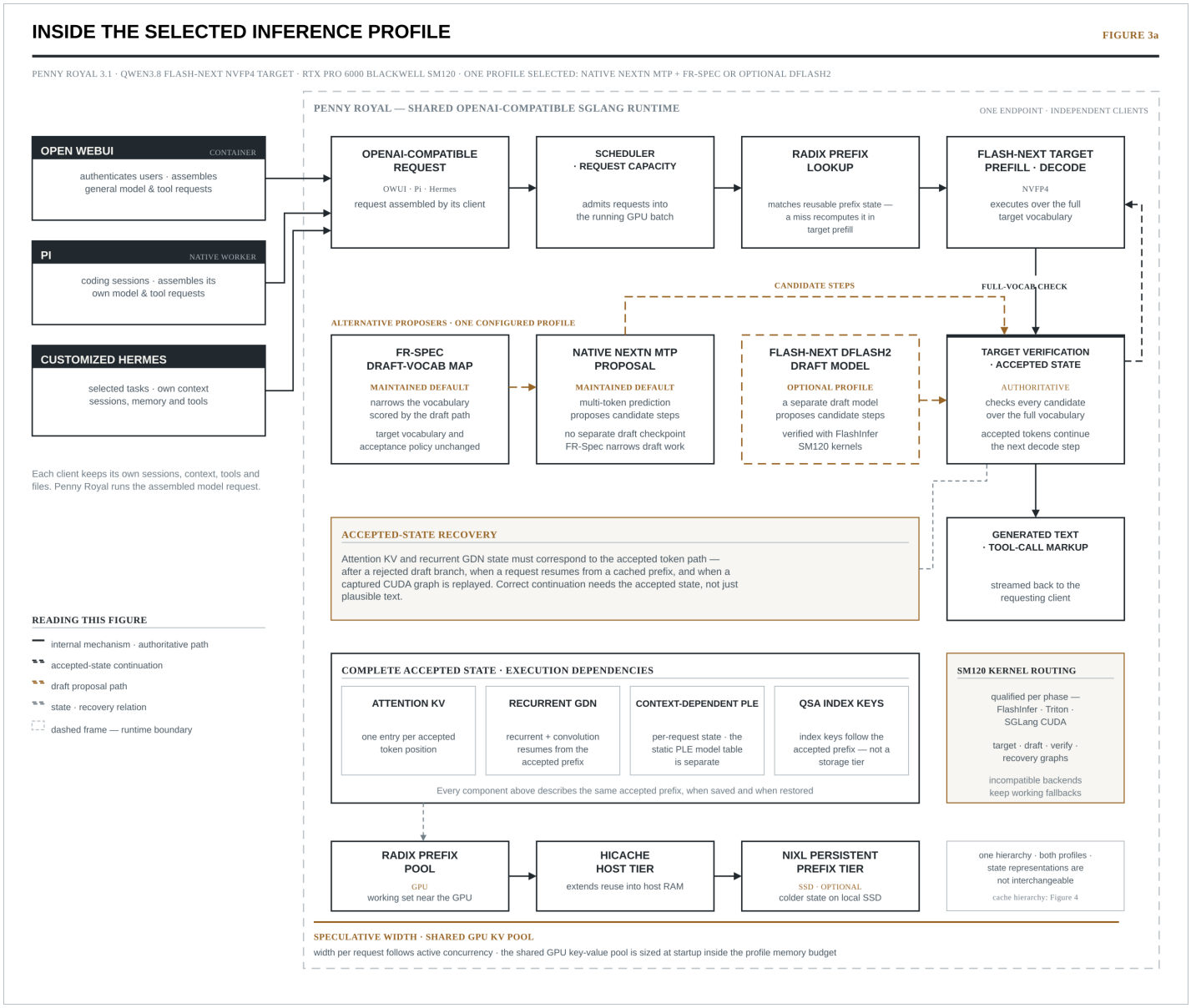


Figure 3a — Inside the selected inference profile

The Flash-Next path coordinates attention, recurrent state, sparse-attention indexing, and native draft prediction. FR-Spec reduces draft scoring work while target verification retains the full vocabulary. An optional DFlash2 drafter replaces the native proposal path and keeps the same verification authority. Recovery returns every dependent state component to the accepted token path before generation continues.

Figure 3b — An inference request

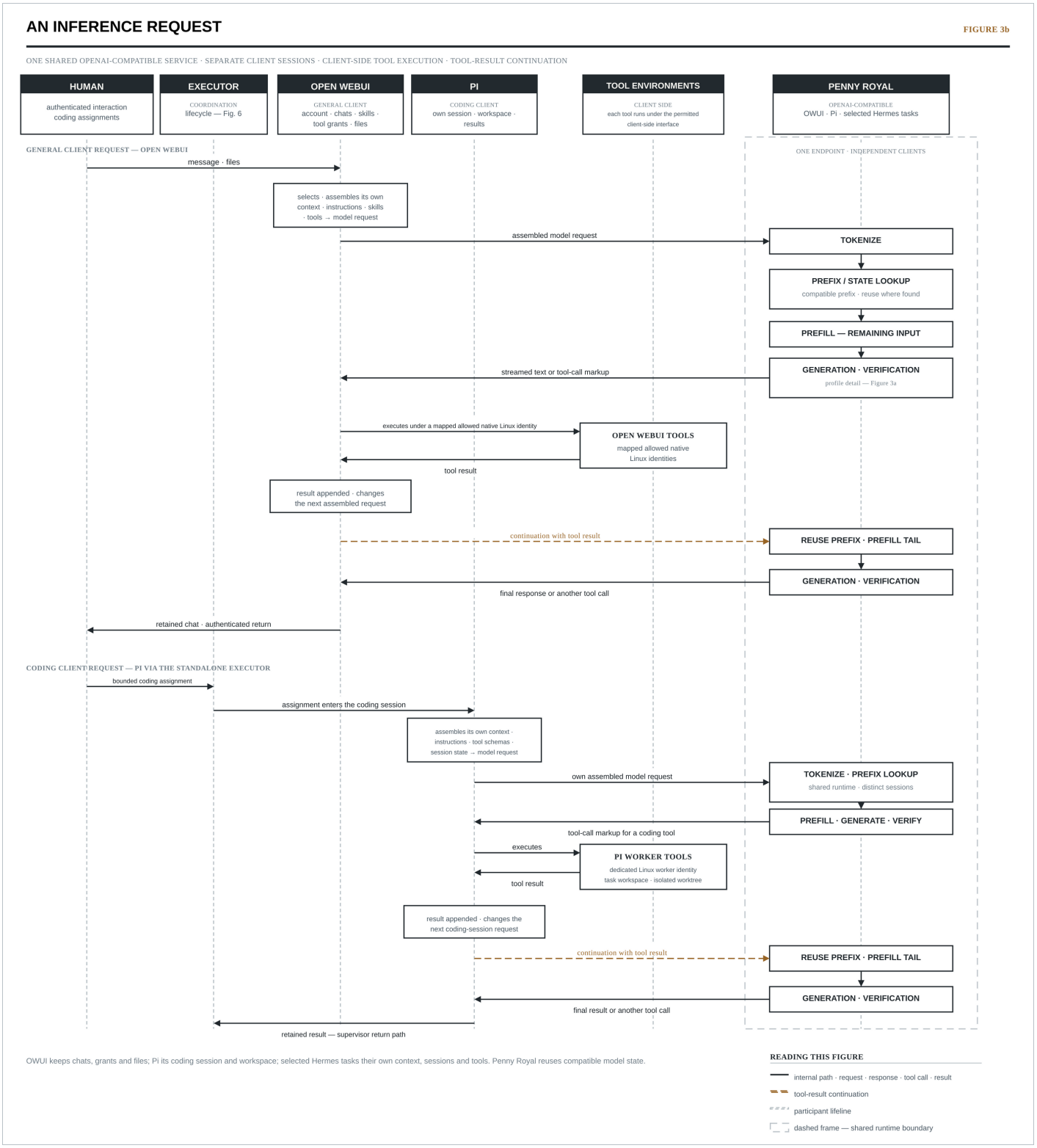


Figure 3b — An inference request

A client assembles the request; Penny Royal tokenizes it, looks for reusable prefix state, computes the remaining input, and generates a response. A tool call returns to the client for execution. The resulting tool output becomes part of the next assembled request, creating another opportunity to reuse the unchanged prefix.

4. HiCache, NIXL, and long-context reuse

An agent repeatedly sends instructions, conversation history, source material, and prior tool results back to the model. Prefill computes the model state for that input. Recomputing a long unchanged prefix on every turn would waste much of the advantage of fast generation.

Penny Royal keeps reusable prefix state in three local tiers:

Tier	Role	Request behavior
GPU radix cache	Hot computed state beside model execution	A compatible resident prefix is reused immediately.
HiCache host RAM	Extends reusable state beyond GPU residency	Compatible state is restored to the GPU when needed.
NIXL local SSD (optional)	Optional persistence of colder computed state through the selected POSIX FILE backend	Compatible prefixes can be restored after eviction or a service restart; GPU and RAM reuse stay available without it.

All three tiers run on thegrid. Requests, cache writes, restoration, and generation stay on my own hardware. The SSD tier is optional. The GPU pool and host RAM tier carry reuse on their own, so GPU and RAM reuse remain available whenever disk persistence is disabled. The 524,288-token served window is the capacity of one model request; the shared GPU state pool and lower cache tiers support active and reusable prefixes across requests.

Complete state is the cache contract

Attention keys and values are only part of the state needed to continue a hybrid model correctly. Flash-Next also carries recurrent memory, context-dependent PLE state, and sparse-attention index data. These components must describe the same accepted prefix when it is saved and when it is restored.

Flash-Next state	Why it must be preserved
Packed target and native-MTP attention KV	Supplies the attention state for the prefix in both target and native prediction paths.
Complete GDN recurrent and convolution state	Preserves the model’s accumulated recurrent memory at the accepted token boundary.
PLE accepted, pending, and ngram sibling state	Keeps the model’s context-dependent PLE state aligned with that boundary.
Compressed QSA index keys	Restores the indexing state used by sparse attention over the prefix.

Saving attention KV alone would leave other parts of the model out of step. Penny Royal’s persistence path saves and restores the complete compatible representation, with recovery and load ordering that bring its components back together before continuation. The 27B/DFlash2 profile has the corresponding requirement for its target, draft, and recurrent state.

The large PLE model table has a separate role: it is shared model data, distinct from the context-dependent PLE state in a cached prefix. Its RAM or optional SSD placement is handled separately from NIXL’s computed-prefix storage.

Reuse across turns and restarts

The first request computes a new prefix and can populate the hierarchy. A later request with the same compatible prefix can reuse GPU state or restore it from RAM or SSD, then prefill only the new tail. Appended tool output, new instructions, and page boundaries can leave a short amount of real prefill even when most of the request is restored.

Compatibility includes the model, tokenizer, precision, state layout, and relevant speculative configuration. Cache namespaces keep different representations separate, and two speculative configurations over the same target weights do not make their cache representations interchangeable. A compatible restart can reuse persisted state; a representation change starts with a fresh namespace and recomputes its prefixes.

Figure 4 — Cache hierarchy and request reuse

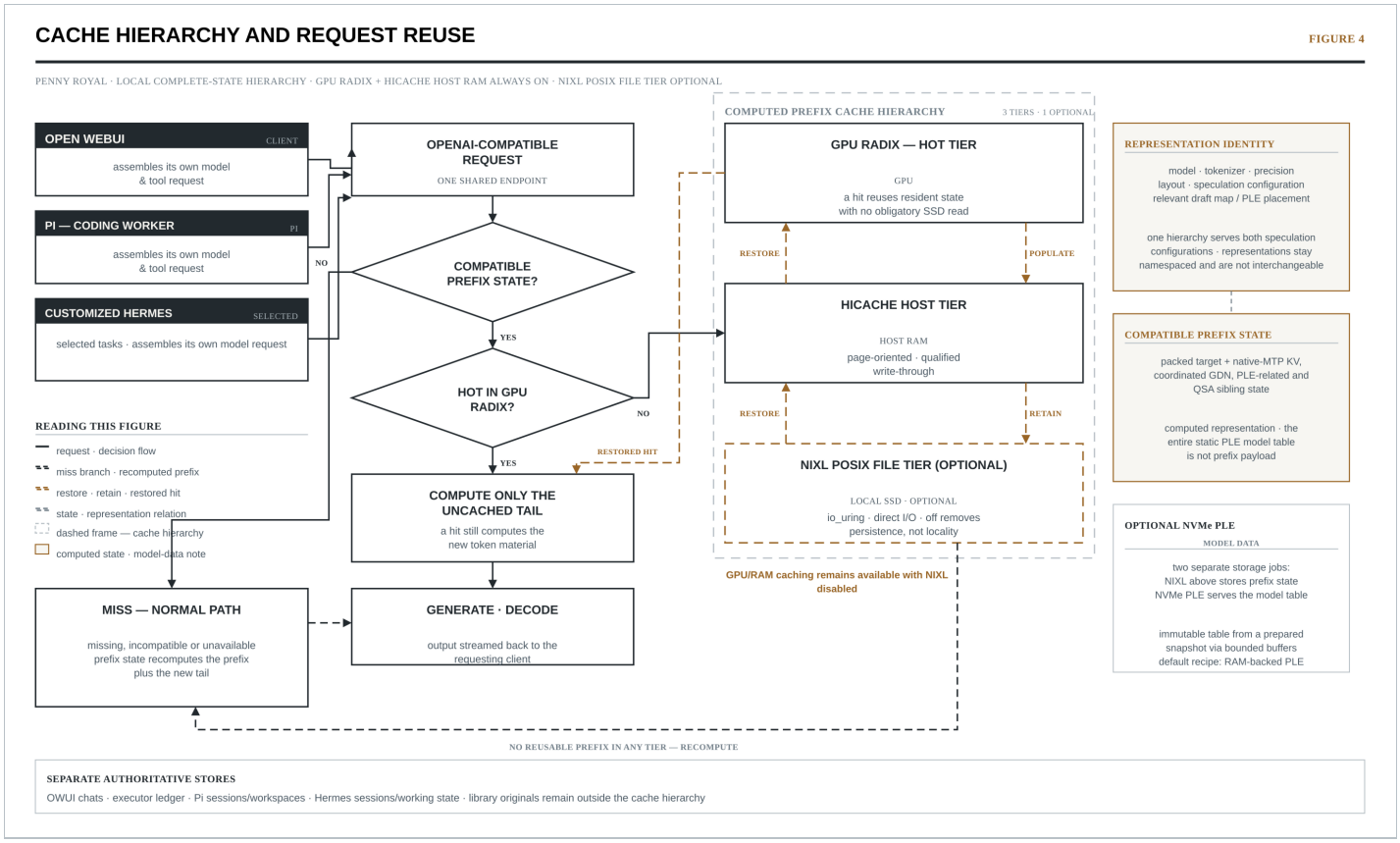


Figure 4 — Cache hierarchy and request reuse

A Flash-Next restart check on this system restored **489,856 of 489,879 input tokens**, recomputed the remaining 23, and returned all three exact long-context markers. It demonstrates the intended architecture: a nearly full context window can resume from complete persisted model state on local SSD, with only the unmatched tail requiring fresh computation. The SSD tier is the optional part of that picture; the complete-state hierarchy below the request path is not.

Reusable computation and durable work

Prefix caching preserves computation. OWUI retains conversations and files, Pi retains coding sessions and workspaces, and Hermes retains its own memory and session state. Those authoritative stores have separate lifetimes from the cache. Losing a cached prefix causes recomputation from the next assembled request while the application's retained work remains intact.

Compaction and caching also serve different purposes. Each client decides which material belongs in the next prompt. Penny Royal decides which computation for that prompt can be reused. A restored prefix still occupies the request's logical context budget; reuse reduces the work needed to process it.

5. Context engineering and human-approved skill learning

The work environment turns inference into an agent interaction. OWUI assembles authenticated conversations, instructions, tool descriptions, files, and tool results. Pi assembles the coding session and repository work for its own tool loop. Customized Hermes assembles its own prompts, memory, session context, and tool definitions for selected tasks. These clients share the model service while retaining their own working state.

OWUI context assembly

The model-facing request contains several layers: shared guidance, account-specific instructions, project context, selected skills, conversation messages, tool schemas, file references, and prior tool results. These layers serve different purposes. A shared contract supplies common behaviour; account guidance calibrates the interaction; a skill supplies a procedure for particular work; a browser page or document supplies material to examine.

OWUI retains the full saved chat while compaction can replace older model-facing turns with a checkpoint summary and recent working context. Compaction runs before model calls, including tool continuations. The context indicator displays the latest provider request's input usage against the configured model window, with estimated or unknown readings identified separately.

The deployment retains built-in file tools while disabling automatic File Context injection. Penny deliberately selects and reads an attachment through a visible tool path. That supports file-intensive work in which the selected source and the resulting tool output matter to the next step.

Figure 5 — Request context is assembled from layers

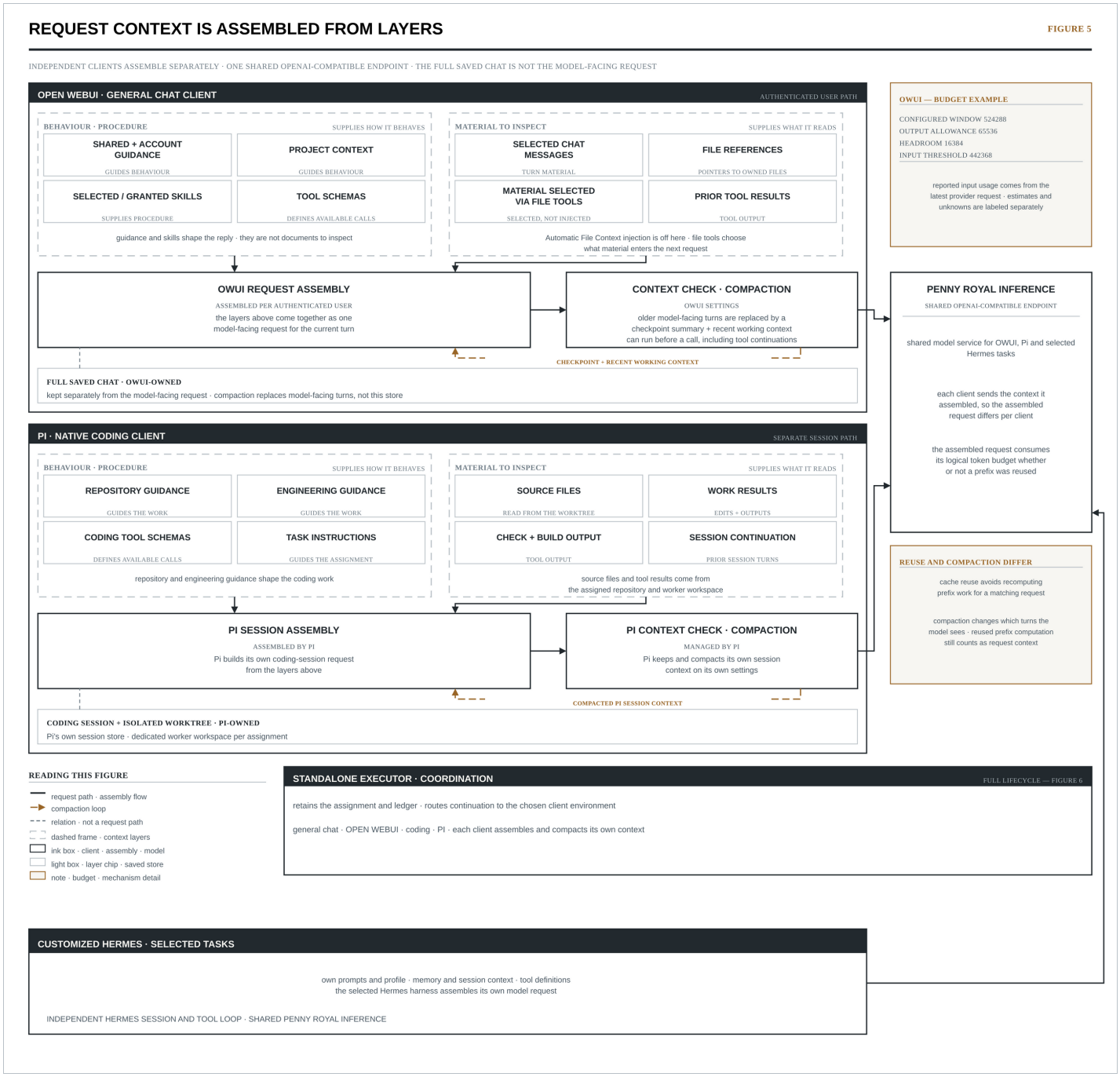


Figure 5 — Request context is assembled from layers

Context assembly determines what reaches the model. Prefix caching determines how much computation can be reused for that request. The request still occupies its logical token budget when part of its prefix is restored from cache.

OWUI context budget example

The documented OWUI budget uses a 524,288-token model window, a 65,536-token output allowance, and 16,384 tokens of headroom for later-assembled material and estimation differences. The corresponding input compaction threshold is 442,368 tokens: 442,368 + 65,536 + 16,384 = 524,288.

Quantity	OWUI budget example	Purpose
Model window	524,288 tokens	Combined model-request capacity
Output allowance	65,536 tokens	Room for the response
Additional headroom	16,384 tokens	Room for additions and estimation differences
Input threshold/cap	442,368 tokens	Trigger for request compaction
Context indicator numerator	Latest provider input-token count	Usage of the request that just ran

Pi manages the context of its own coding session. The executor retains the assignment and routes continuation to the selected environment; OWUI's context display and compaction settings belong to the OWUI path.

Skills and personal procedures

A skill supplies instructions for a class of work. OWUI exposes shared capabilities through registration and grants, with personal skills limited to their owner. The native coding environment carries its own engineering guidance and tools, including repository navigation through CodeGraph.

The personal OWUI fork also includes a skill-proposal path. Background review or an explicit learning request can draft a procedure from a user's tool-using conversation and stage it in the owner's Skill Creation folder. The owner can approve, reject, or request revision. Approval checks the staged change and intervening edits before the personal skill becomes discoverable in later work.

Figure 5a — Human-in-the-loop skill creation

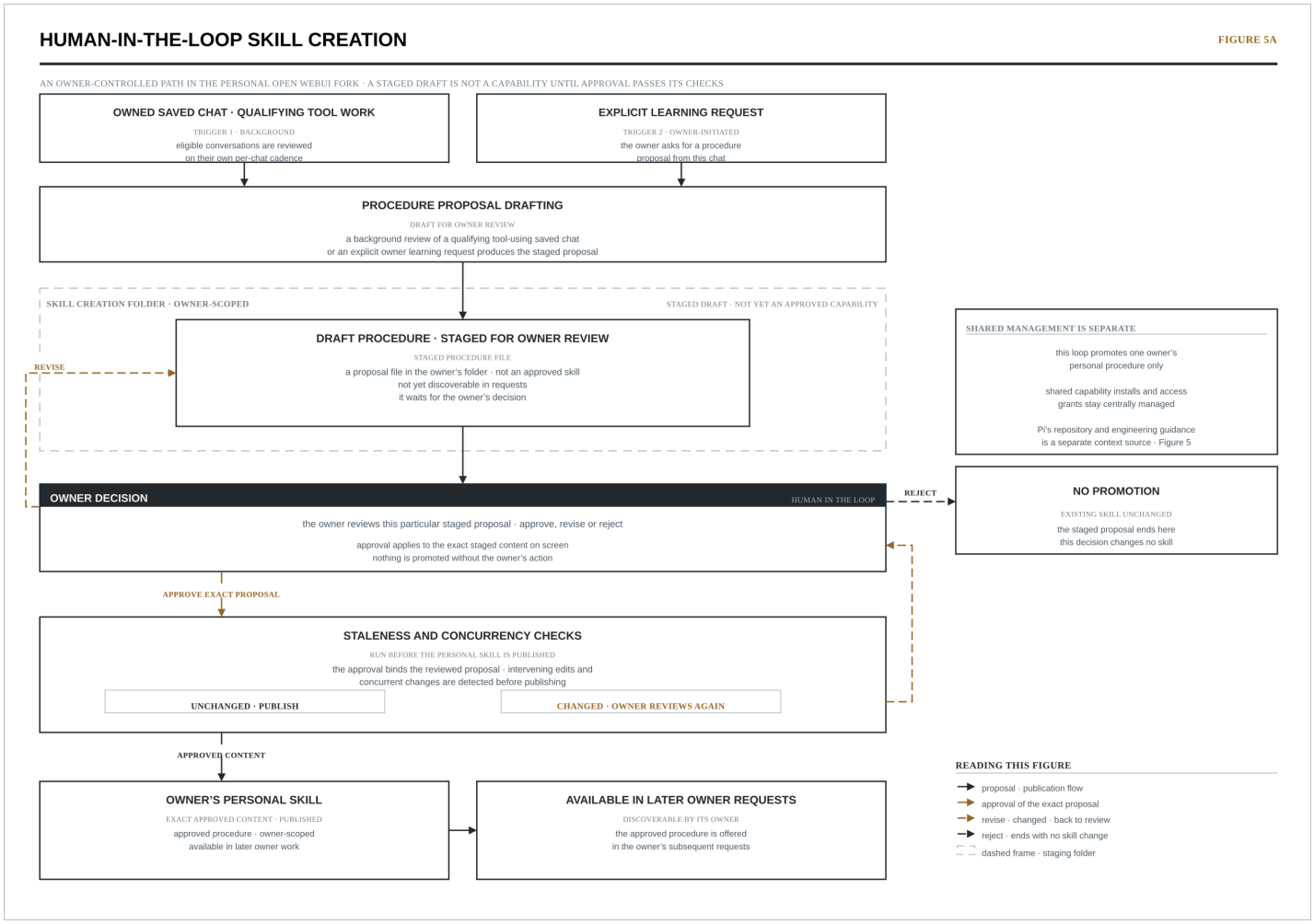


Figure 5a — Human-in-the-loop skill creation

The proposal, revision, and approval loop turns a useful procedure into an owner-controlled capability. Shared capability installations and access grants remain centrally managed.

6. Event-driven multi-agent execution and human review

The standalone Penny executor gives an assignment a lifecycle beyond a single model response. It runs as a native service on room101, independently of the OWUI application container, and coordinates two execution environments: OWUI for general work and Pi for coding.

Codex submits the objective, source context, boundaries, and originating supervisor destination. The executor creates a durable task ID and workspace, records compact state in its ledger, and starts the selected environment. The task can continue across multiple invocations, pause for input, or return a result for review.

Two execution routes

Route	Execution environment	Retained work
General assignment	OWUI's authenticated chat API and normal agent/tool loop	Saved worker chat, native workspace, and generated artifacts
Coding assignment	Pi under a dedicated Linux worker identity	Coding session, isolated Git worktree, and reviewable source changes
Shared coordination	Standalone executor	Task identity, status, steering, capacity, compact result, and references to work

General assignments can use the browser, research, file, media, and document capabilities exposed through OWUI. Coding assignments use Pi's native repository tools and engineering guidance. Pi has its own session and tool loop, with isolated source changes and reviewable patch output.

Both environments call Penny Royal for inference. The executor manages the assignment while the selected environment manages model turns, tool calls, and the detailed execution trace.

Task state and execution state

The ledger records the objective, current status, progress, steering, result, and references. Detailed work remains in the saved OWUI chat or Pi session and its workspace. A supervisor can read the current task state quickly and inspect the actual files, source diff, or execution trace when reviewing the result.

The executor's configured capacity is three active tasks shared across the two routes. A running assignment occupies a slot. Paused work and completed results retain their state while releasing execution capacity; a full executor returns busy. Repository work can use an exact base revision and an isolated worktree. Research or document work can begin in a task workspace without a checkout.

Task state	Working behaviour
Running / continue	The selected environment has more work to perform
Blocked	A specific answer is needed; work and task state are retained
Ready for review	The result is available for supervisor inspection
Interrupted	Retained session and workspace state are reconciled before continuation
Failed / cancelled	The assignment has stopped with a recorded outcome

Event-driven return and continuation

An actionable transition queues a compact notification for the originating Codex task. The event carries task identity and status. Codex reads the current ledger and result, inspects the relevant work, and supplies feedback through steering and resume.

For OWUI work, continuation uses the saved worker chat and native workspace. For Pi work, it uses the retained coding session and worktree. A pause preserves completed actions and the source changes already made. Review feedback returns to the same durable assignment.

The notification path uses a restricted receiver to queue attention on the supervisor’s originating host. The supervisor reads current state when it wakes, so task status, results, and steering stay anchored to the ledger rather than to the timing of a notification.

Collaboration and review

OWUI supports bounded subagents for separable investigations or analysis. The parent Penny task collects their results and incorporates them into its response. Across work surfaces, Codex supervises the durable Penny assignment and reviews the actual returned artifact or source change.

The human owner sets the objective and decides consequential scope and release steps. The executor and work environments provide the retained state, tools, and return path that make sustained delegated work practical.

Native async helpers in Pi

Pi’s native asynchronous helper lets a coding task send a separable investigation off to its own Pi run while the parent Penny keeps working. One helper runs per parent at a time, and a helper does not delegate further. The helper is not a second lane through the executor’s main capacity, so it costs no additional task slot.

A helper finishing on its own cannot mark the parent assignment complete. Completion needs the helper result to reach the parent model context, the parent to incorporate it and respond, and Pi to settle.

OWUI’s existing helper route stays a separate mechanism, and the supervision boundary does not change: Codex still reviews the actual returned work.

Figure 6 — Event-driven multi-agent assignment and return

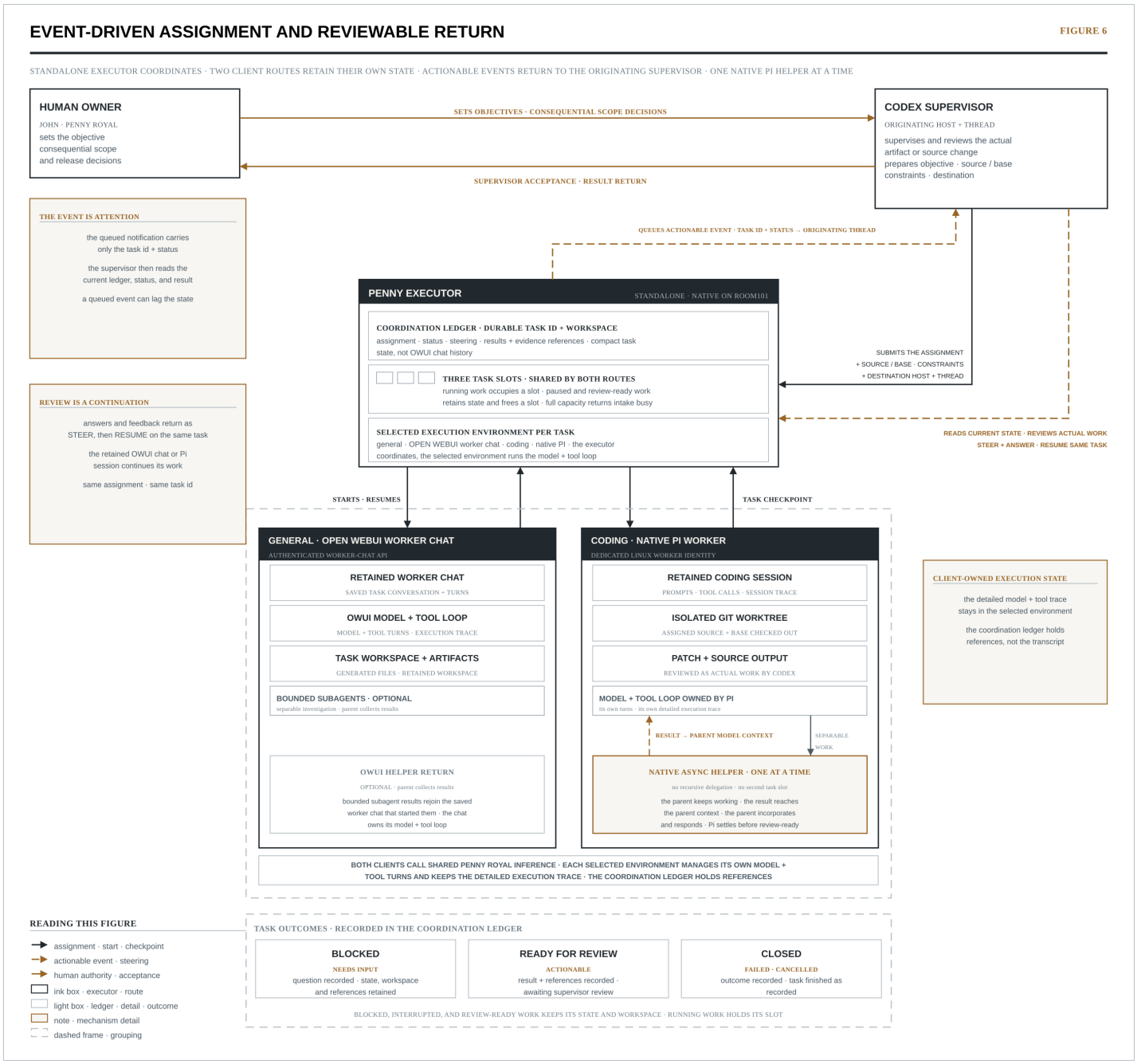


Figure 6 — Event-driven multi-agent assignment and return

The sequence follows an assignment through execution, retained state, notification, review, and continuation. General and coding work share the coordination lifecycle while retaining their own execution state. The Pi column also shows the single native async helper lane and the return it must reach.

7. Identity, authorization, and native execution

OWUI capability groups grant access to models, research, image work, document tools, and agent workspaces independently. Only people who need native workspace execution receive Linux accounts and native Open Terminal services. A chat- or image-only account need not have a shell. The identity router maps an authenticated, allowed OWUI identity to that person's terminal and rejects missing or unknown identities.

Shared binaries and skills live in a centrally managed capability tree. User-written files, browser profiles, caches, and experiments live under that user's account. The platform owner's private presentation assets are in a separate private capability tree. This makes it possible to share a document renderer or browser installation while preserving separate writable state and private material.

Pi uses a dedicated native coding identity for its sessions, task workspaces, and isolated worktrees. That route shares the model service and durable executor with general assignments while retaining its own tool loop and source changes.

Two authorization layers

OWUI grants answer *which capabilities may this account invoke?* Linux identities answer *whose processes and files are used when the capability runs?* The system needs both. An image-only user can invoke an allowed image tool without receiving a native terminal. A workspace-enabled user has an account-specific native terminal and a mapped home directory. The identity router checks the authenticated OWUI identity and an allowlist before proxying the Open Terminal interface; an absent or unknown identity fails closed.

Capabilities are additive. Global access stays narrow, and groups supply model, research, image, workspace, document, or private resources. This avoids a separate model copy for every person's combination of permissions. It also keeps personal interface preferences under each user's control: deployment reconciliation manages shared invariants and selected prompts/grants, rather than treating every user-owned setting as drift.

Boundary	Mechanism	Architectural purpose
OWUI account to capability	Authenticated session plus group/resource grants	Limits visible models, skills, tools, and administrative functions
OWUI account to terminal	Identity router with explicit user mapping	Prevents a missing identity from reaching a default or another person's terminal
Terminal to filesystem	Native Linux UID and per-user home	Keeps writable work and browser state separate
Terminal to shared tools	Root-managed capability tree with read/execute group access	Shares runtimes without granting users control of installations
Private material	OWUI grants plus filesystem and mount-namespace controls	Keeps private documentation and brand assets out of friend workspaces
Browser file delivery	OWUI session and terminal/file ownership checks	Avoids public artifact URLs for personal results

The shared browser illustrates the difference between installation and state. The agent-browser binary and matching skill are installed once. Browser processes, profiles, sockets, downloads, and screenshots live under the invoking user's home. The default persistent browser session lets one user's task continue across tool calls, while concurrent browser jobs under that same account require distinct sessions. A web page's contents and metadata are treated as input to evaluate, not instructions with authority over the agent.

Thegrid's shared NFS artifact mount maps client identities to one server-side identity. Native terminal services therefore use filesystem and mount-namespace boundaries to separate user-private paths. This is a specific correction to a real infrastructure constraint, not a generic claim that NFS permissions enforce per-user privacy in this deployment.

Figure 7 — From grant to operating-system identity

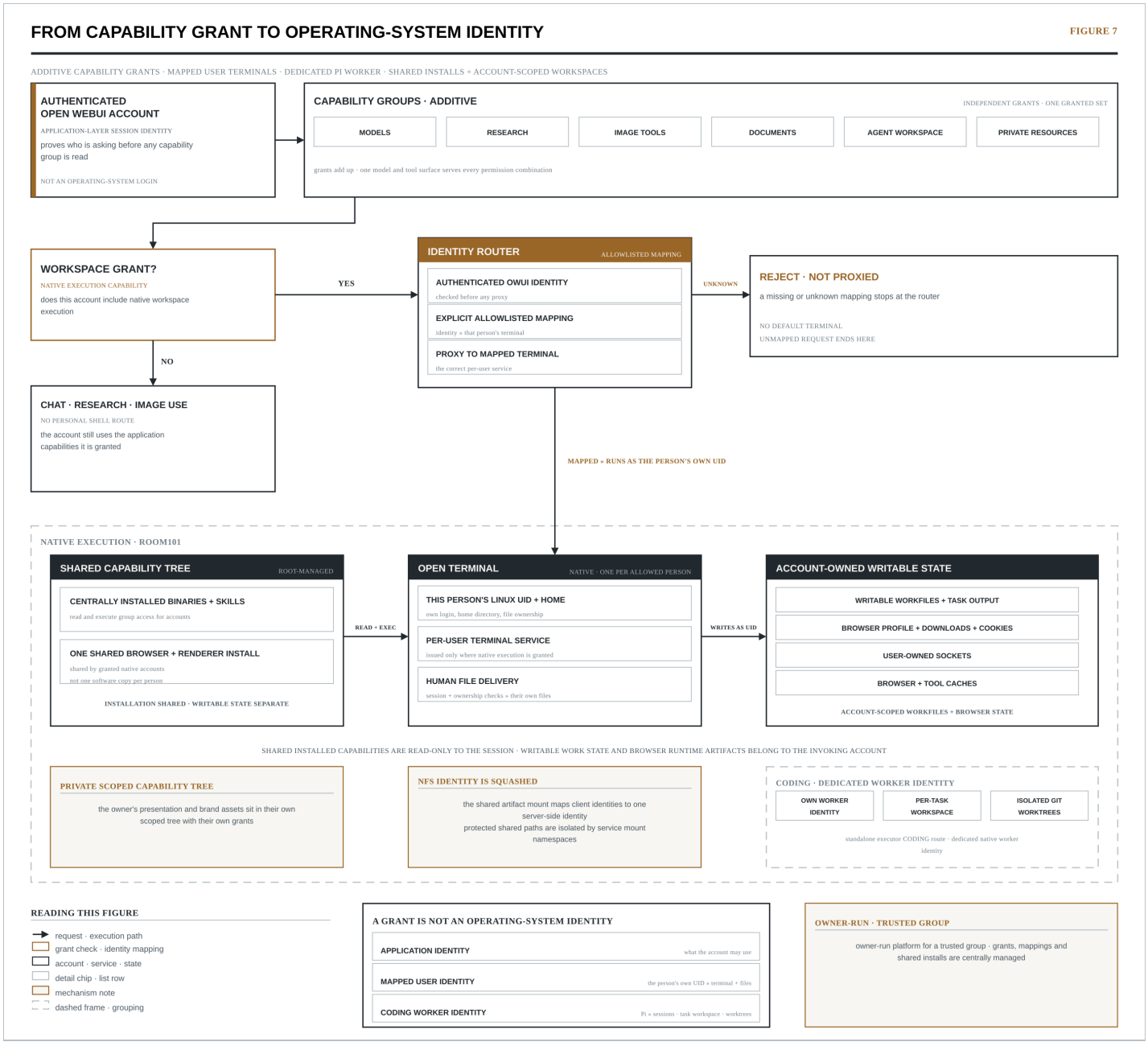


Figure 7 — From grant to operating-system identity

This small trusted-group design uses OWUI ownership checks and Unix ownership. Shared NFS artifact storage uses squashed identity, so native terminals hide protected paths through service mount namespaces; server-side mode bits alone cannot separate those paths.

8. Agent Library and authenticated file delivery

The OWUI file path gives a file three distinct forms: a display copy, a durable original, and a native working copy. Open WebUI keeps a display copy associated with a chat. Agent Library keeps a durable original in S3-compatible Versity storage under the user's private or shared identity. A native workspace holds a working copy for tools to inspect or modify. These copies differ in ownership and lifetime, so the document treats them separately.

Copy	Purpose	Lifetime and access
OWUI file	Display and chat attachment	Tied to OWUI's authenticated file handling and chat experience.
Agent Library object	Durable original or deliberate published result	Stored under the mapped user's private or shared S3 permissions; independent of a working chat.
Native workspace file	Tool input, intermediate, or finished working copy	Owned by the Linux user; can be removed or regenerated from the library when appropriate.

The identity router uses the mapped user's library identity for private and shared objects; a missing mapping does not fall back to another account. Renaming changes display metadata while retaining the same object identity and bytes. Neither behavior is a substitute for application authentication at download time.

An OWUI upload hook archives files handled by its upload path, including native MCP image responses. When a chat attachment goes through the terminal router, the router archives it before forwarding a workspace copy. Library tools let Penny list, fetch, publish, and rename objects within the mapped user's permissions. A new published result receives a new identity rather than overwriting the earlier object. The library is not a replacement frontend; the user normally remains in OWUI.

For coding assignments, Pi retains repository changes and outputs in its task workspace. The library path described here serves OWUI's uploaded files and selected artifacts.

Object identity, permissions, and copies

Versity provides the S3-compatible API and object storage. The integration maps OWUI users to private buckets and a shared bucket. The router uses credentials for the mapped regular users and does not use the storage administrator identity for ordinary agent work. Bucket ownership and policy implement the library access model; the application's authenticated identity and router mapping determine which storage identity a given request receives.

Library keys are UUID-prefixed so two uploads called image.png do not collide. The original filename and a display name can remain metadata. Renaming the display name updates metadata on the same key and preserves the object identity and bytes. A new output publish assigns a new UUID rather than replacing an older original. These mechanics are mundane but important: agent work often produces iterations, and overwriting an input or losing the distinction between a filename and object identity would make later retrieval unreliable.

Two ingress paths serve different user actions. OWUI's upload hook archives content handled by its file upload route and preserves OWUI's own file ID for display. A terminal attachment passes through the identity router, which archives it before placing the UUID-prefixed working copy in the user's workspace. Both paths avoid forcing a user to manage a separate S3 console for ordinary work. The standalone storage Explorer remains an administrative or direct-storage interface, not the normal OWUI file-delivery path.

A library fetch returns a working copy that tools can read or edit. A publish call records a deliberate output in the library. Fetch and publish can return an OWUI-relative download route, which Penny can present without inventing a URL. OWUI and its terminal proxy still authenticate the browser request. This is why the location of the working copy matters: removing a copy that backs a current terminal download link breaks that link, even though the library original remains available for another fetch.

Figure 8 — File lifecycle and delivery

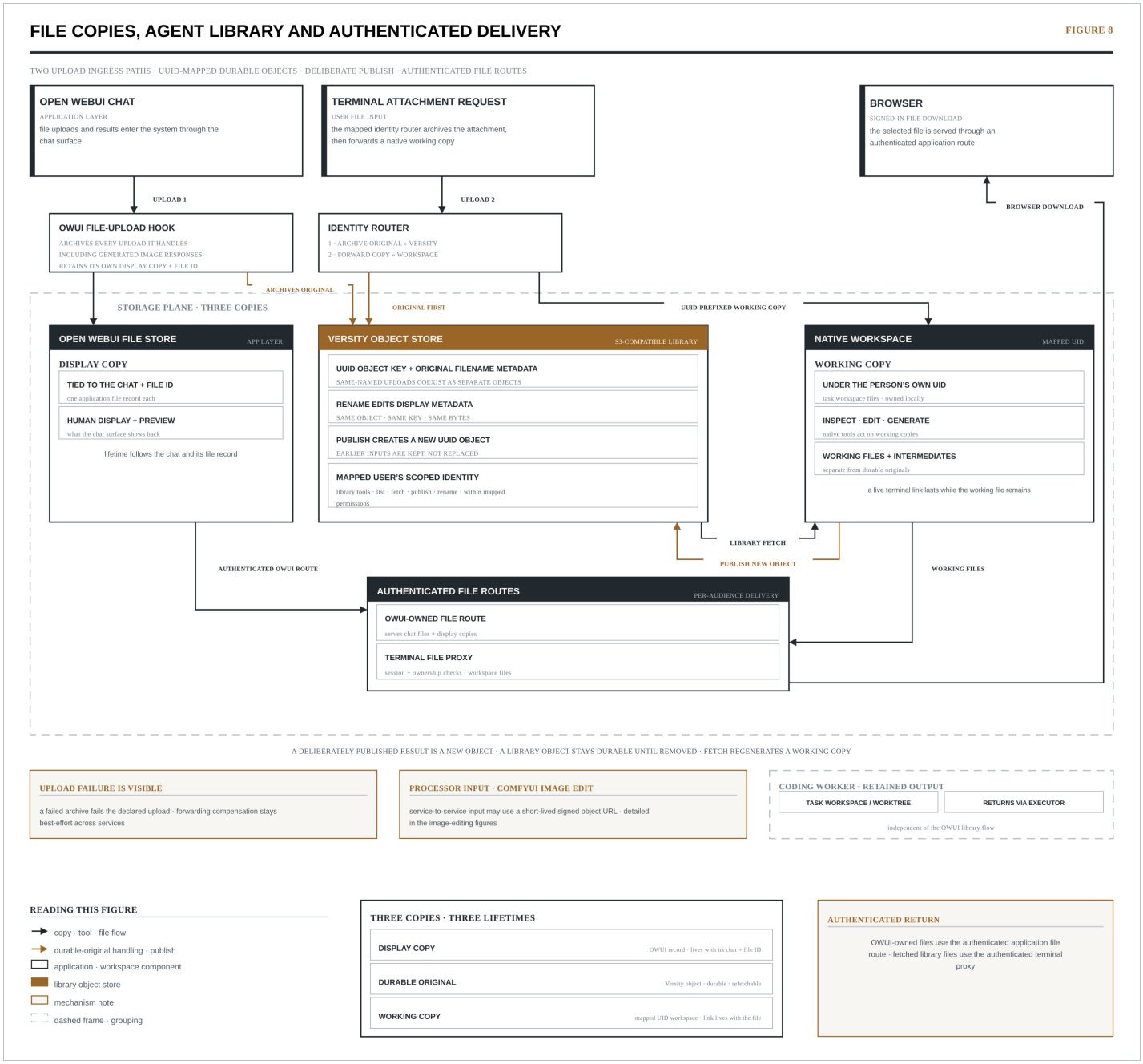


Figure 8 — File lifecycle and delivery

The browser does not need direct S3 access. Library-only results can be fetched into the user's workspace and delivered through the authenticated terminal file route. Files already owned by OWUI use its authenticated file route. A live download link depends on the working copy remaining present; the durable library object can be fetched again if needed.

Image editing has a different internal input path: a selected library object can receive a short-lived signed URL for ComfyUI to read into memory. That URL is for service-to-service processing and is not the user-facing download link. The integration issues these URLs for a limited processing window.

This architecture makes a useful separation: the original survives beyond an individual chat, tools get disposable working copies, and people receive files through their authenticated application. Backup and restore of all library state are a separate operational question; the existence of S3-compatible storage alone does not prove a tested restore.

File transfer and retention

Archival is part of the declared upload contract. A storage failure fails the upload visibly rather than returning success while silently losing its durable copy. The router writes transfer data through private temporary files and cleans them up after use. If archival succeeds but forwarding to a terminal fails, it attempts to remove the new object. That compensation is best effort. There is no distributed transaction across OWUI, the router, the native terminal, and S3; an interrupted request can leave an object that an operator must identify. This is a narrower, more honest guarantee than claiming every cross-service step is atomic.

The integration's configured upload limit is 512 MiB. The router spools its transfer to private temporary storage, but upstream OWUI proxy and Open Terminal paths still buffer some uploads in memory. This affects capacity planning for very large files and prevents the library chapter from implying a fully streaming, arbitrary-size pipeline. It does not change the normal object identity or authenticated download model.

The durable library comprises Versity objects, metadata, IAM state, and application mappings. OWUI retains its own chat database and display files. These storage domains preserve different parts of the user's work.

Authenticated download as an architecture boundary

A browser download has a different trust relationship from a service-to-service signed input. The former should be checked against the logged-in OWUI user and the selected file route. The latter is a temporary capability used by a processor such as ComfyUI to read a specific object. The generated artifact can be archived and then offered through OWUI. Keeping these routes separate means a workflow can use S3-compatible storage internally without making object storage public or teaching an agent to expose signed URLs as final results.

Dated multi-user acceptance exercised these boundaries directly: duplicate filenames remained independent, OWUI and fetched-library downloads matched original bytes, one user's private object was denied to the other, and unauthenticated download attempts failed. Those tests validate the selected paths at the time they ran; they do not replace review when the application fork or storage integration changes.

9. Connected processing services

The main agent path can select connected research tools, an interactive browser, image services, speech, and document tools. Their contracts are intentionally different.

Research services return information for Penny to evaluate and cite. The interactive browser is a stateful Chromium session running through a user's native terminal, with that user's profile and downloads. It can continue across tool calls; page content remains untrusted input. The distinction matters because finding a page and filling a form are different activities with different state and consequences.

For images, AI Tools owns the named MCP contract and invokes tracked ComfyUI workflows on Neomatrix. ComfyUI performs generation or editing; image bytes return to the harness for archival and authenticated display. Agent Library supplies selected edit inputs through temporary signed URLs rather than a permanent shared input directory.

For speech, OWUI owns microphone upload, transcription display, read-aloud interaction, and personal voice selection. Dubline on Neomatrix owns local decoding, speech inference, voice references, and GPU serialization. The current integration uses MOSS transcription and IndexTTS output. Speech and ComfyUI share workstation GPU resources; this build does not claim a general scheduler for them.

Document tools run in the native user's workspace using shared installations and, when granted, private assets. That places generated files within the same identity and delivery scheme as other agent work.

Connected research and interactive browsing

Research is available in the normal Penny Royal experience through centrally managed MCP connections. The Research grant controls access to those connections. Native OWUI web search is disabled in this deployment, avoiding two competing search paths with different tool selection and citation behavior. A research result is input for Penny to evaluate; neither the search provider nor a retrieved page owns the conversation's instructions.

Browser Research and Interactive Browser solve different tasks. Browser Research selects, retrieves, and extracts information from the web. Interactive Browser operates a persistent Chromium session under the user's native Linux identity. It can click, enter text, navigate tabs, wait for page state, and handle downloads across terminal calls. The agent reads the installed browser skill and uses the browser's accessibility snapshot and element references rather than assuming page coordinates or inventing selectors. Its profile and socket are isolated by the user's home, and page content remains untrusted even when it appears inside a tool result.

This distinction is relevant to enterprise workflow design. Research can inform a decision; browser interaction can change external state. The latter therefore needs an explicit user or supervisor checkpoint for credentials, CAPTCHA, or consequential external actions. The platform's architecture makes that difference visible through capability grants and tool contracts.

Image generation, image editing, and result presentation

AI Tools is the named image-service interface presented to the agent. It owns MCP tool contracts and the tracked workflows sent to ComfyUI on Neomatrix. ComfyUI owns model execution and rendering. Image bytes return through the service path and are handled by OWUI as native image content, which gives the application a chance to archive the output and serve an authenticated display URL.

An edit adds a source-selection problem. Penny can identify one to three Agent Library objects under the user's library access, request short-lived source URLs, and pass them to AI Tools in the requested order. ComfyUI's URL input node loads the images into memory for the edit graph. The signed URL is a transport credential with a short lifetime; it does not become a permanent artifact or a browser download link. The final PNG returns to OWUI, where the ordinary archive and display path applies.

The application fork's native MCP image-reference handling matters at this boundary. A tool can return image content without first publishing a public URL. OWUI persists the display copy and supplies an authenticated reference. Model-facing guidance lets Penny embed that reference in a response while the server enforces access control. Generated-image and reference checks exercised this path.

Figure 9 — Specialized processing rejoins the authenticated user path

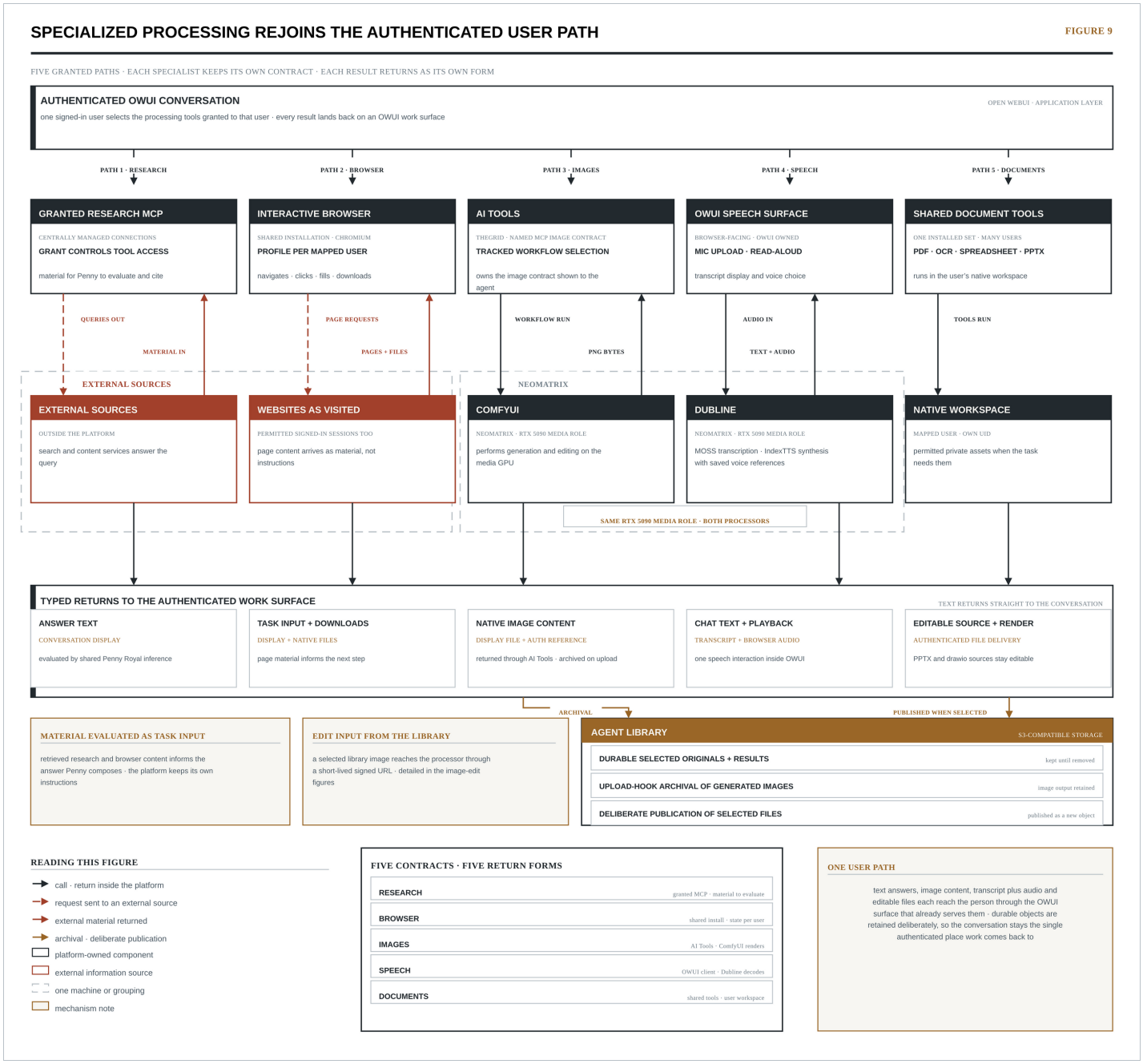


Figure 9 — Specialized processing rejoins the authenticated user path

Image editing across the storage and processing boundary

Image editing demonstrates why the three-copy file model is useful. The source image can be retained as an Agent Library object even after its original chat is closed. Penny selects the specific object under the user's library identity and requests a short-lived internal URL. AI Tools submits that URL to a tracked ComfyUI workflow. ComfyUI reads the source bytes into memory, renders the edit, and returns the resulting PNG through AI Tools as native MCP image content. OWUI creates an authenticated display file and the archival hook records the result. If the user decides the revision is worth retaining as a named library result, a publish operation records another object without overwriting the source.

The separation keeps each actor’s responsibility narrow. Agent Library controls retained objects and per-user S3 access. AI Tools owns the image-tool contract and workflow selection. ComfyUI performs GPU rendering. OWUI owns the logged-in user’s display and download. The signed input URL is not displayed as a permanent artifact link, and ComfyUI does not become a general-purpose object browser.

Figure 9a — Image-edit sequence

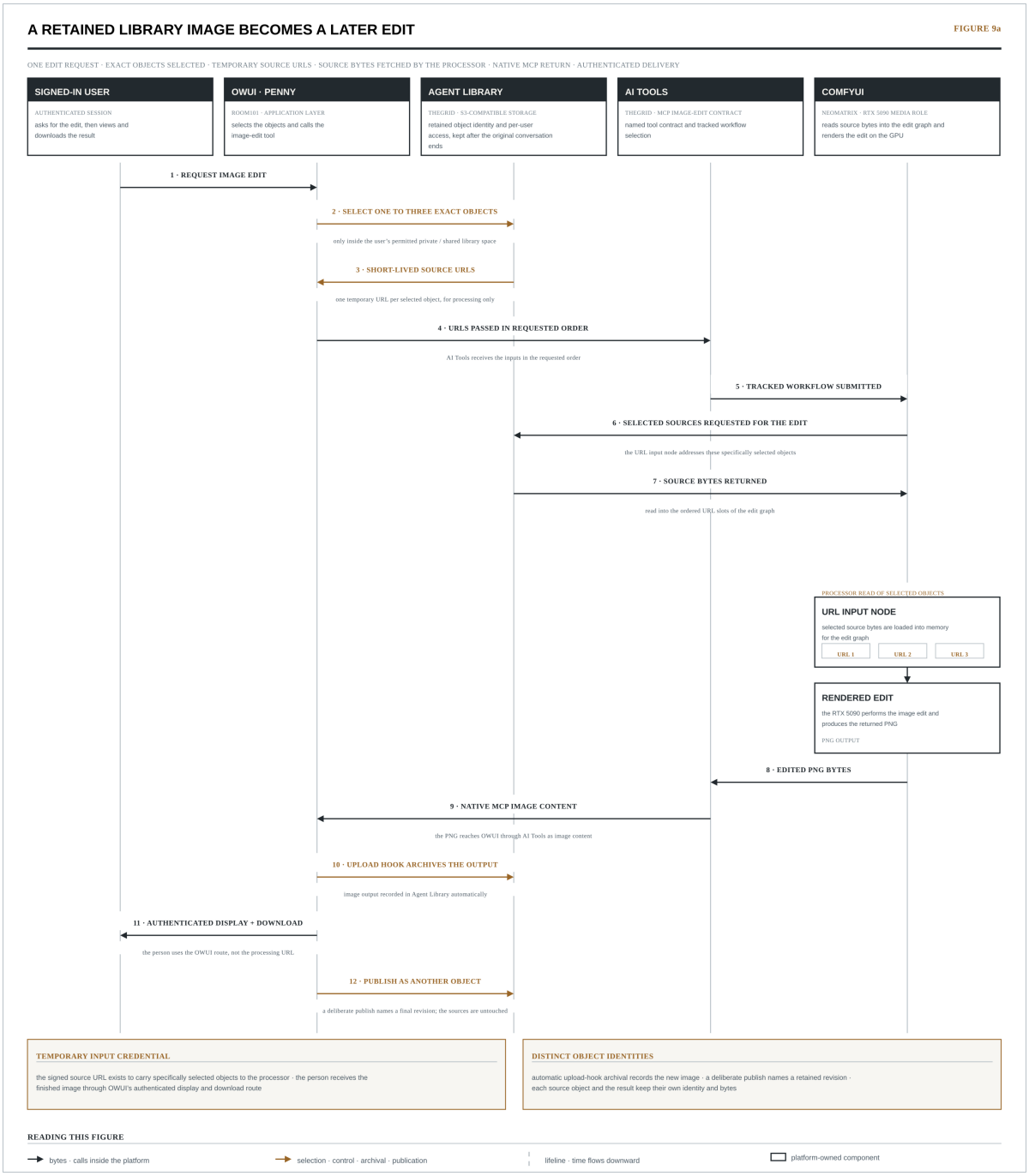


Figure 9a — Image-edit sequence

Speech input and playback

OWUI is the browser-facing speech client. It receives microphone recordings, shows transcript text, initiates read-aloud, and retains each person’s voice selection. Dubline on Neomatrix provides the local speech API. Its service owns decoding, MOSS transcription, IndexTTS synthesis, voice references, model lifecycle, GPU serialization, and request cleanup. The browser does not use a separate Dubline UI.

The input contract matters. OWUI’s audio bypass passes the original uploaded recording through once, allowing the backend to maintain its decoding and speaker context. Without that bypass, OWUI’s normal preprocessing can convert or split a large upload before it reaches the speech model. The same bypass affects output processing, so the TTS provider is configured to request MP3 and the backend returns actual MPEG audio. Input and output behavior were therefore qualified together.

Read-aloud splits text by paragraph to stay within Dubline’s per-request limit. A single paragraph longer than that limit can fail explicitly; the system does not claim unlimited synthesis by virtue of splitting. Voice choice is user-owned in OWUI. A model-level voice override would take precedence and would undermine that personalization. Browser Voice Mode was accepted as a practical user flow, while individual voice quality remains a subjective choice.

Figure 9b — Speech ownership and return path

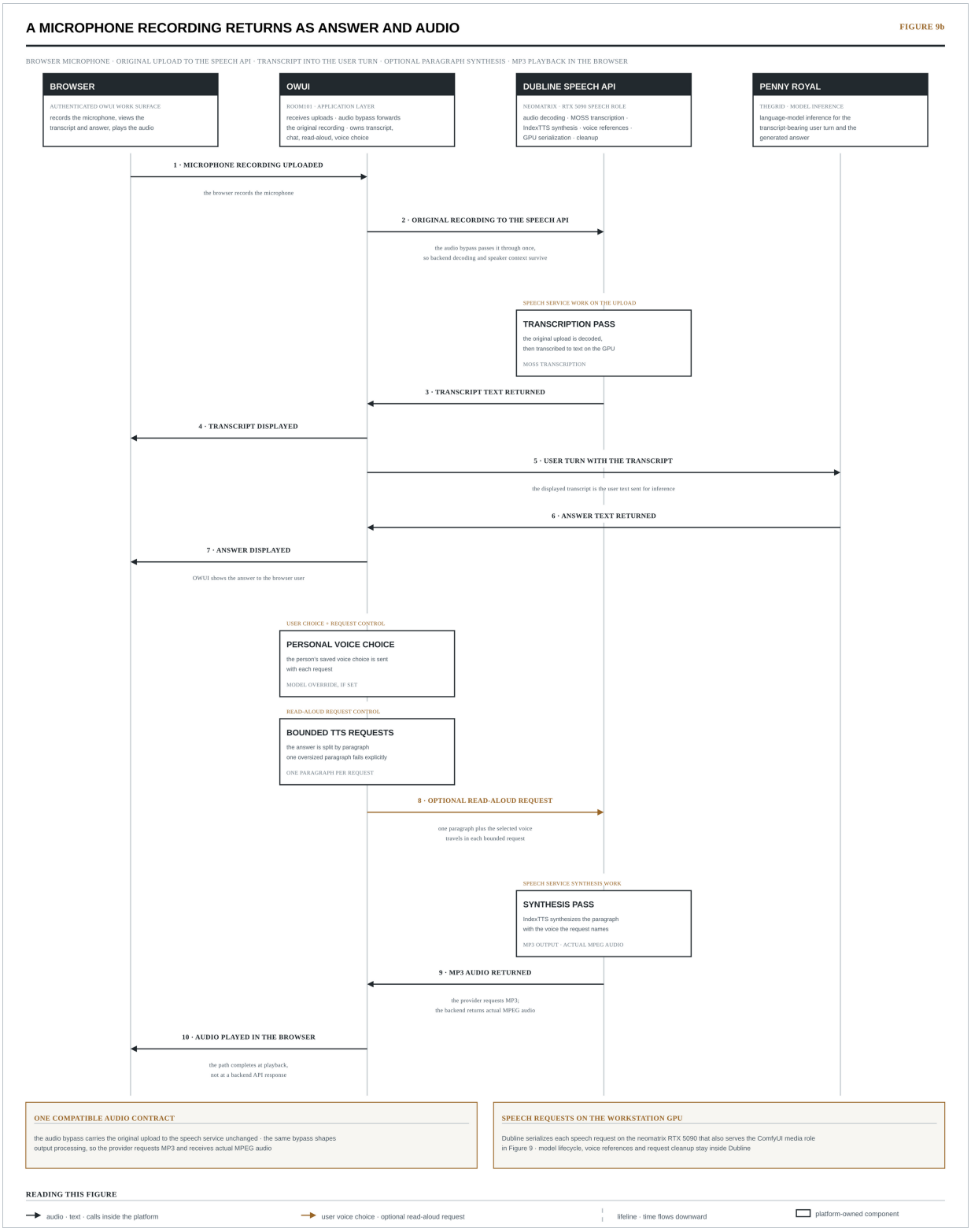


Figure 9b — Speech ownership and return path

A successful speech backend response verifies only part of the user path. Transcription must reach the chat, Penny must answer, and playback must reach the browser for the full interaction to work. The same principle applies to image output: a successful ComfyUI render does not by itself verify OWUI persistence or authenticated display.

Documents and editable diagrams

Document creation runs through the native terminal and shared capability installation. The shared tree contains document, spreadsheet, PDF, OCR, presentation, and diagram tools. A granted user can write and modify work in their own home while the shared installation remains read-only to them. This provides a consistent route for input inspection, generation, rendering, validation, and authenticated delivery.

The architecture values editable outputs. A presentation remains a PPTX with source material and rendered pages for review. A hardware or network diagram remains a draw.io file with reusable shapes and connections rather than only a flat image. Catalog-backed assets improve the physical specificity of a diagram, but they do not validate an invented topology. Technical source facts and visual layout still require review.

Private brand capabilities can overlay the shared document stack. A deck owner’s approved assets and licensed fonts can remain in a separate private capability tree with corresponding OWUI grants. Application authorization and filesystem placement both protect them.

10. Representative workflows

The following walkthroughs trace four different outcomes through the same architecture: supervised engineering, file-backed image editing, editable presentation production, and long-context continuation. Each identifies the initiating authority, the services and state involved, the returned artifact, and the acceptance boundary.

Workflow A — Human-in-the-loop engineering assignment

Assignment. The human owner and Codex establish the problem, source state, boundaries, and expected result. Codex submits a coding assignment to the standalone executor with an exact source revision and originating supervisor destination. The executor creates the durable task and starts Pi in an isolated task workspace.

Execution. Pi runs the coding session under the dedicated native worker identity. The session is a native JSONL RPC conversation between the executor and Pi, and it needs neither OWUI nor Open Terminal on this path. Pi can inspect the repository, navigate source, change files in an isolated Git worktree, and run focused checks using Penny Royal inference for its model turns. The session and workspace retain the detailed work while the executor records progress and compact task state.

Pause and return. A question raised in the Pi session pauses the assignment, releases its execution slot, and keeps the session, workspace, and existing changes. The executor queues an actionable notification to Codex. The supervisor reads the current task record and question, supplies steering, and resumes the same assignment; resume continues the same Pi session and worktree without replaying earlier turns. Waiting releases its execution slot.

Review and result. Pi returns reviewable source changes and their checks. Completion has a precise definition on this path: Pi settlement plus the executor's bounded shutdown of the running Pi process retires that invocation and frees its capacity slot, while the retained session and worktree stay available for review and resume. An executor restart reconciles interrupted Pi sessions before it admits new work. Codex examines the actual diff and behaviour, and returns any defects to the same task and worktree. The human owner decides acceptance and any publication or deployment. After review, Codex and I preserve the useful outputs and then explicitly request release; only then does the executor release the workspace, session, and patch material it owns. Otherwise inactive retention expiry applies: retention can be extended, active tasks are excluded, and an incomplete cleanup retries. Ordinary conversations, shared installations, and separately uploaded OWUI files keep their own lifetimes through all of it. A general assignment follows the same coordination path through a saved OWUI chat and its granted tools.

Figure 10 — Engineering assignment control and evidence

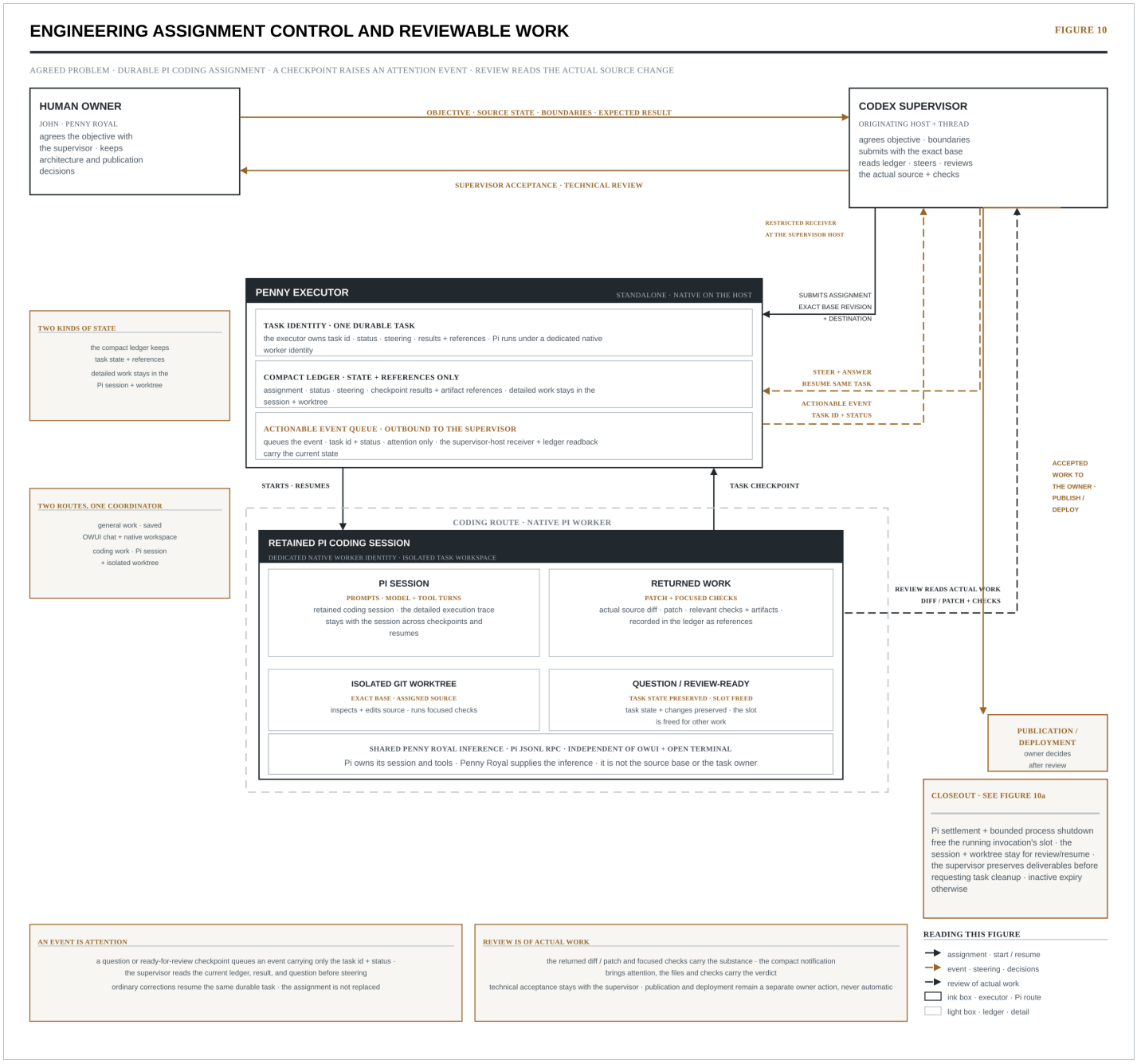


Figure 10 — Engineering assignment control and evidence

Figure 10 stops where the reviewed source and its evidence are recorded. Figure 10a, shown below, continues from that point through resume, restart recovery, and closeout. Codex and I preserve the useful review outputs first and explicitly request release; only then does the executor release the workspace, session, and patch material it owns. Otherwise inactive retention expiry applies: a retention extension keeps that material for a stated period, active tasks are excluded from expiry, and a cleanup that fails is retried rather than recorded as done.

Figure 10a — Coding task closeout and retention

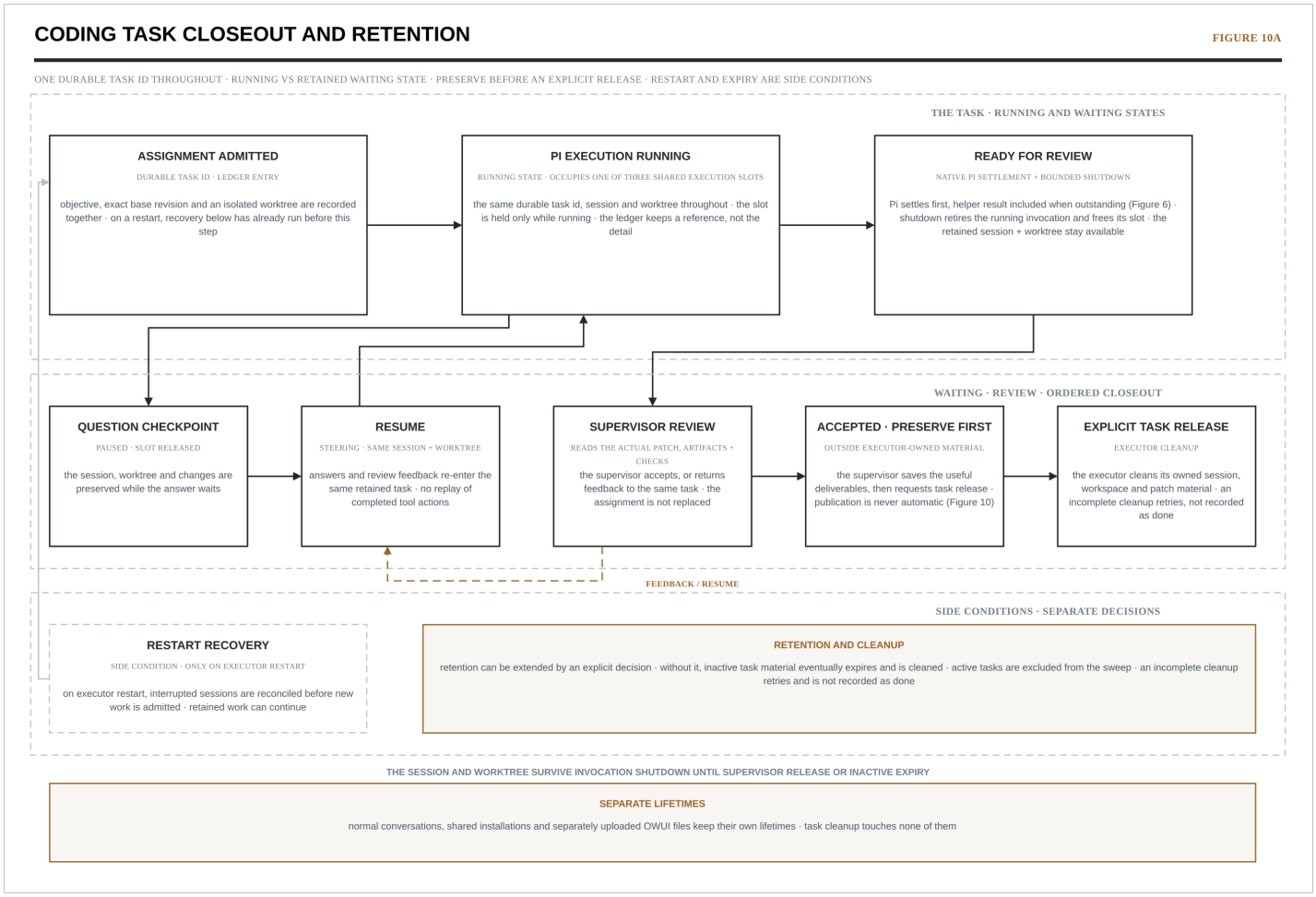


Figure 10a — Coding task closeout and retention

Workflow B — A retained image used in a later edit

A user first supplies an image to OWUI. The application maintains the authenticated chat display copy, while the integration archives the original bytes in Agent Library with a unique key and metadata. This prevents a later file called image.png from overwriting the earlier image. The durable object remains useful after the source conversation is no longer the active workspace.

In a later chat, Penny lists or searches the user's allowed library space and fetches the exact source. The terminal receives a working copy under the user's Linux identity. A display-name change can make the object easier to locate without changing its stable ID or bytes. If the task is an image edit, Penny requests a short-lived source URL and supplies it to AI Tools in an explicit input order. ComfyUI reads the source into the edit graph and returns PNG bytes through AI Tools. OWUI archives and displays the new image. A deliberate publish operation can retain the final result as another library object, leaving the input unchanged.

The download path is independent of the S3 processing URL. OWUI can serve its own file object through its authenticated file API. A library-only object can be fetched into the native workspace and presented through the authenticated terminal file route. If that workspace copy is removed, its existing terminal link may stop working; the library object can still be fetched again. That behavior is visible in the architecture and preferable to treating a temporary signed storage URL as a permanent deliverable.

The same pattern applies to non-image files: preserve an original, work on a copy, publish deliberate results with distinct identity, and let the application mediate user delivery.

Workflow C — An editable branded presentation

The branded-deck path demonstrates editable presentation production through a private, user-specific capability built on the shared toolchain.

1. **Brief and authority.** The deck owner supplies the audience, argument, approved facts, and source material. The private presentation capability is granted only to the authorized OWUI identity. Its brand overlay and licensed assets live in a private filesystem tree; generic presentation tools are shared.
2. **Creation.** The agent uses the private capability within the owner's native workspace. The toolchain creates editable slide content and a PPTX, rather than a slide-shaped raster image. Source material and intermediate files remain available for revision.
3. **Review.** The deck is rendered to pages and inspected for hierarchy, font use, legibility, overflow, visual consistency, and fidelity to the brief. The deck owner reviews technical and customer-specific claims. Large prose changes are made in the editable source and regenerated, preserving layout quality.
4. **Revision and delivery.** The owner requests changes through the normal conversation. The agent revises, rerenders, and returns the deck through authenticated file delivery. A deliberate library publish can preserve a result for later work.

The light and dark HPE examples below show the result of that workflow: private brand grants, shared presentation tools, native workspace ownership, editable PPTX output, render-and-review, and authenticated delivery. Customer facts remain subject to the deck owner's approval.

Figure 11 — Presentation artifact pipeline

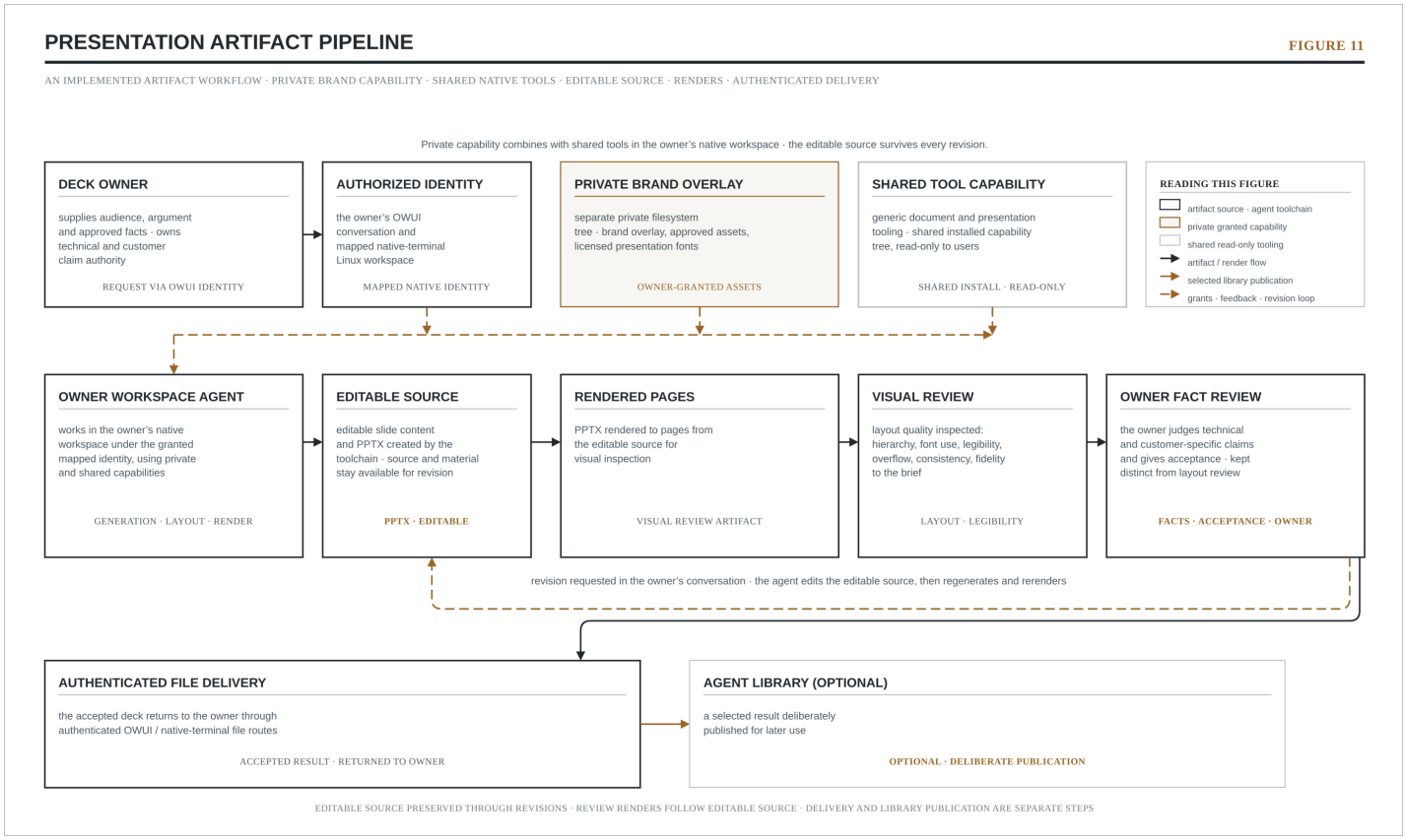
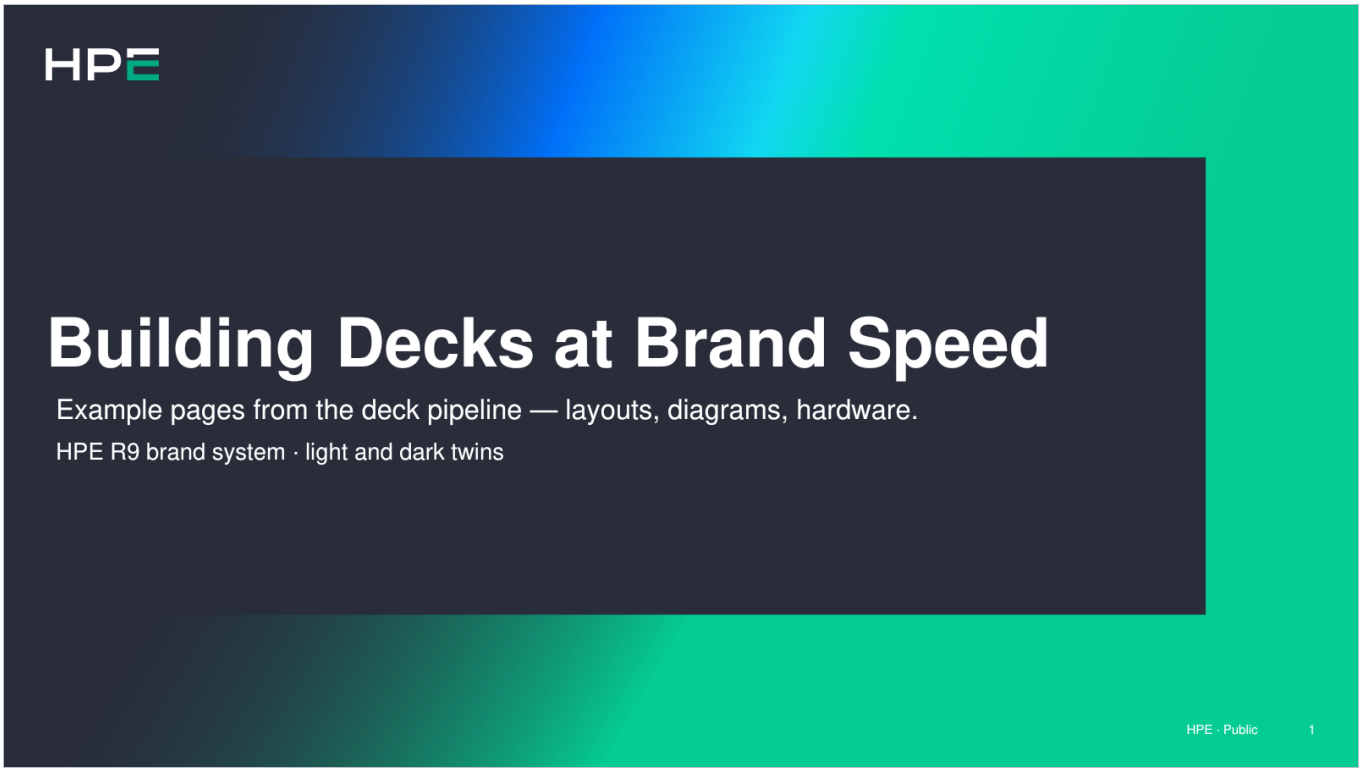


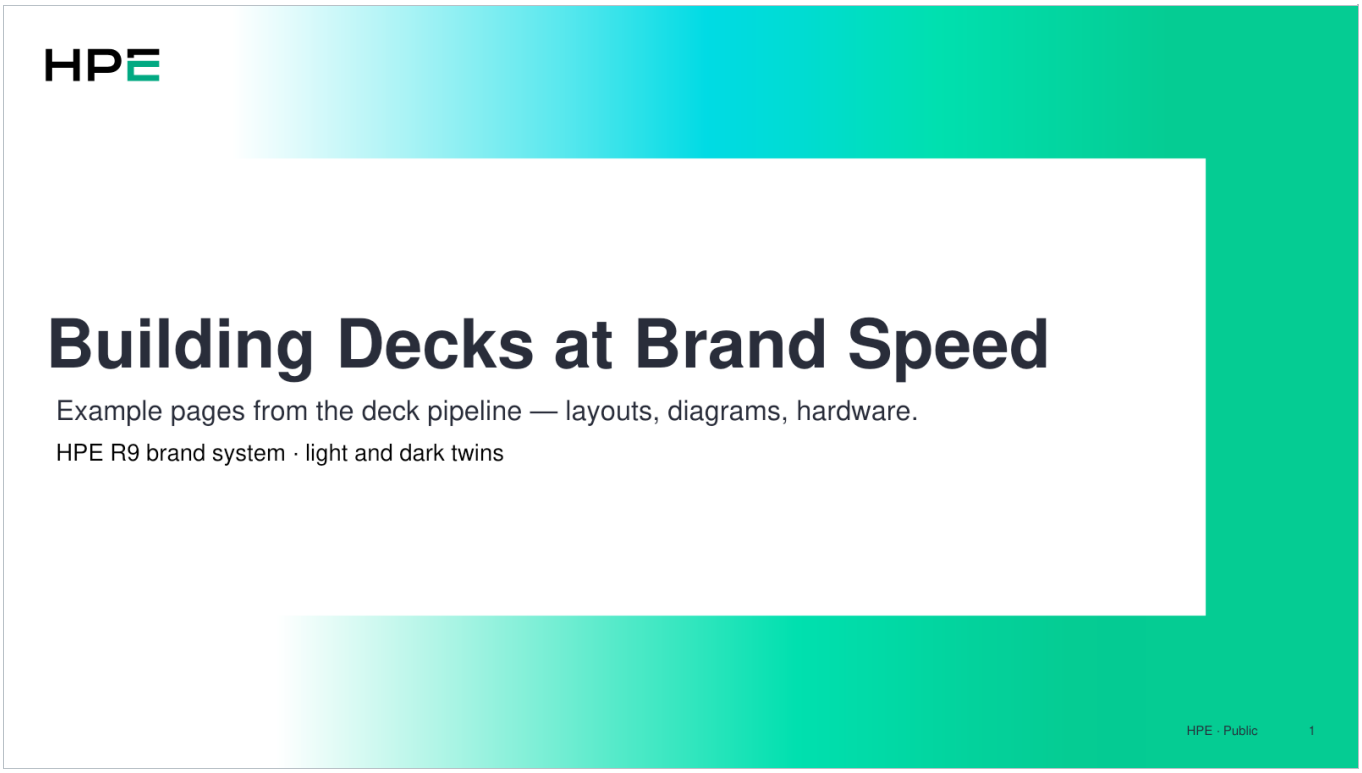
Figure 11 — Presentation artifact pipeline

One branded deck, two presentation themes

The same editable content model was rendered through HPE light and dark brand themes. These reviewed page images demonstrate the private presentation capability built on the shared document toolchain.

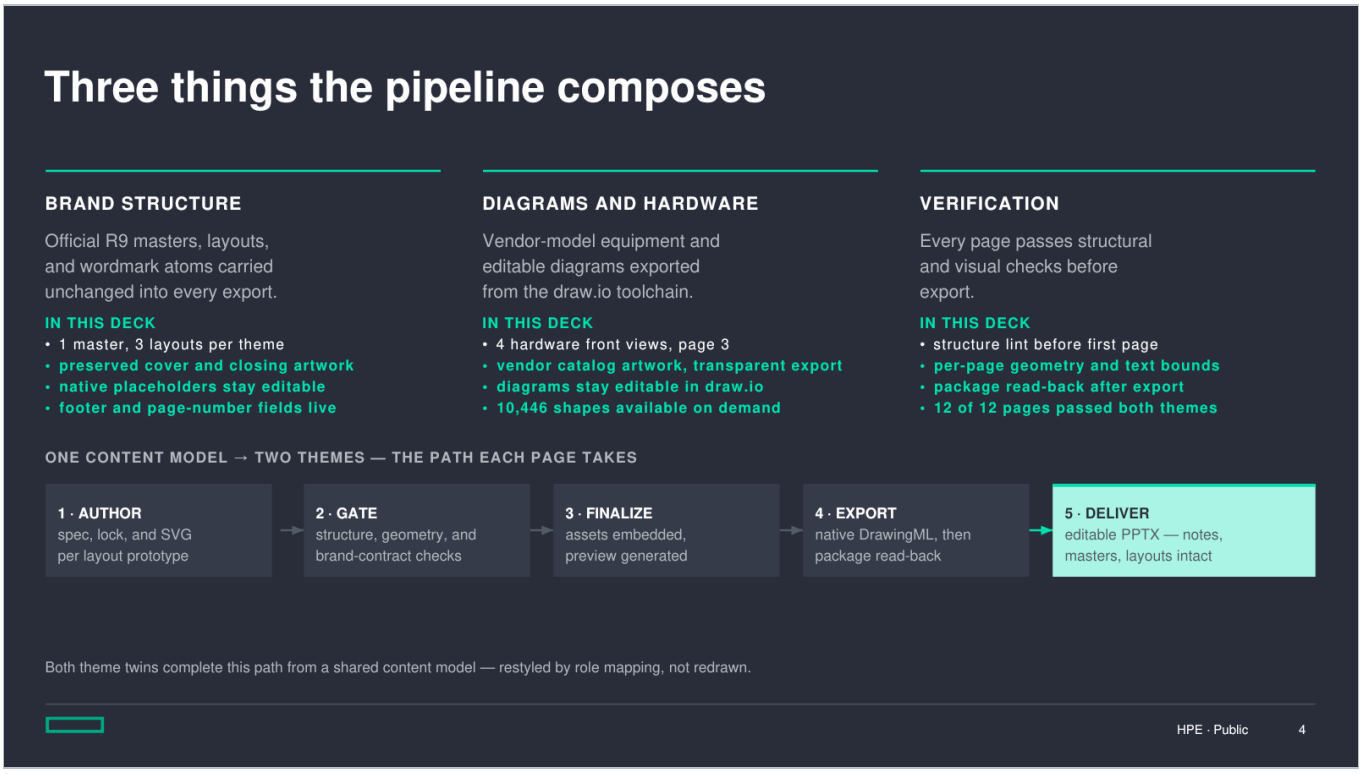


HPE dark theme — Cover page



HPE light theme

Cover page



HPE dark theme

Pipeline page

Three things the pipeline composes

BRAND STRUCTURE

Official R9 masters, layouts, and wordmark atoms carried unchanged into every export.

IN THIS DECK

- 1 master, 3 layouts per theme
- **preserved cover and closing artwork**
- **native placeholders stay editable**
- **footer and page-number fields live**

DIAGRAMS AND HARDWARE

Vendor-model equipment and editable diagrams exported from the draw.io toolchain.

IN THIS DECK

- 4 hardware front views, page 3
- **vendor catalog artwork, transparent export**
- **diagrams stay editable in draw.io**
- **10,446 shapes available on demand**

VERIFICATION

Every page passes structural and visual checks before export.

IN THIS DECK

- structure lint before first page
- **per-page geometry and text bounds**
- **package read-back after export**
- **12 of 12 pages passed both themes**

ONE CONTENT MODEL → TWO THEMES — THE PATH EACH PAGE TAKES

1 · AUTHOR spec, lock, and SVG per layout prototype → **2 · GATE** structure, geometry, and brand-contract checks → **3 · FINALIZE** assets embedded, preview generated → **4 · EXPORT** native DrawingML, then package read-back → **5 · DELIVER** editable PPTX — notes, masters, layouts intact

Both theme twins complete this path from a shared content model — restyled by role mapping, not redrawn.

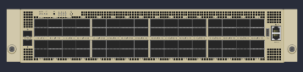
HPE light theme — Pipeline page

Real hardware, rendered — not clipart

HPE Alletra Storage MP X10000 Front

HPE Alletra Storage MP X10000

HPE Apollo 6500 Gen10

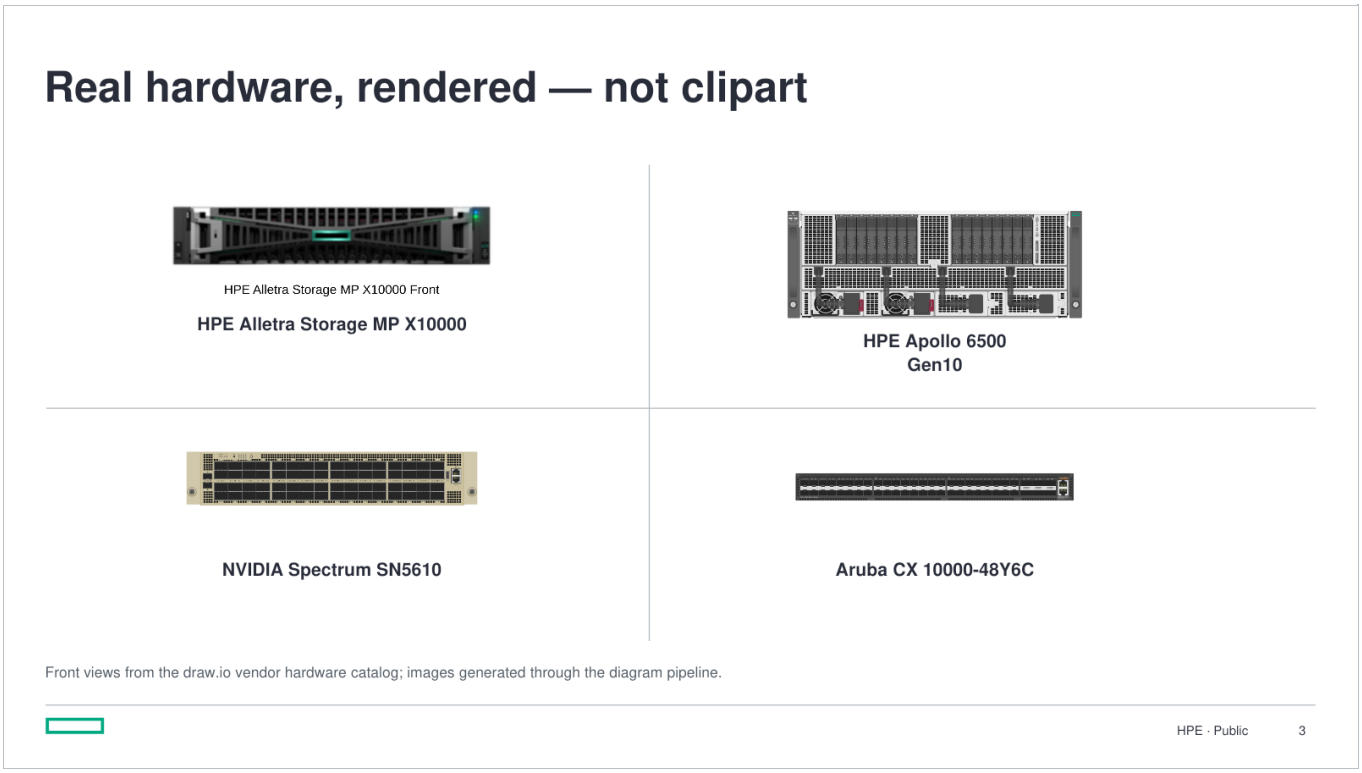


NVIDIA Spectrum SN5610

Aruba CX 10000-48Y6C

Front views from the draw.io vendor hardware catalog; images generated through the diagram pipeline.

HPE dark theme — Hardware page



HPE light theme

Hardware page

Workflow D — Long-context continuation

A long agent session repeatedly presents shared background, recent messages, and new tool results to the model. OWUI and Pi assemble and retain the context of their own work. OWUI measures the assembled request against the configured model window. The context circle displays the latest provider prompt count when available. Backend compaction can replace older model-facing turns with a checkpoint summary when a forthcoming request approaches its safety threshold, while the complete saved chat remains available for inspection.

Penny Royal addresses a different cost. If the assembled prefix is compatible with cached state, the runtime can reuse GPU pages or restore older state from HiCache/NIXL and prefill only the new tail. Application compaction decides what information enters the next prompt. Prefix caching decides how much computation required for that prompt can be reused. One mechanism does not replace the other.

The Flash-Next restart check restored 489,856 tokens from a 489,879-token input. For a long-running engineering agent, compatible prefix reuse avoids recomputing most shared context as new tool results and decisions accumulate.

11. Human supervision and review

The platform supports long stretches of independent agent work while keeping the human owner involved at the points that matter: objectives, architectural tradeoffs, persistent changes, and release. Codex supplies supervision and source review; Penny supplies execution through the selected work environment.

For an engineering assignment, the supervisor examines the actual diff, relevant source, and focused checks. A fresh reviewer can investigate a concrete concern with a narrow source brief. Findings return to the same Penny task and worktree so the change and its corrections remain one piece of work.

From result to accepted work

A returned result supplies files, source changes, and a concise account of what was done. Review establishes whether it meets the objective. A diagram or presentation is inspected in its rendered form; a code change is inspected in source and exercised through the relevant behaviour. The human owner can request revision or accept the outcome.

Review effort follows the change. A substantive implementation receives source and behaviour review; a prose or layout change receives an editorial and rendered check. Repeated findings that expose a design problem bring the work back to a human decision.

Figure 12 — Bounded multi-agent review and architecture stop

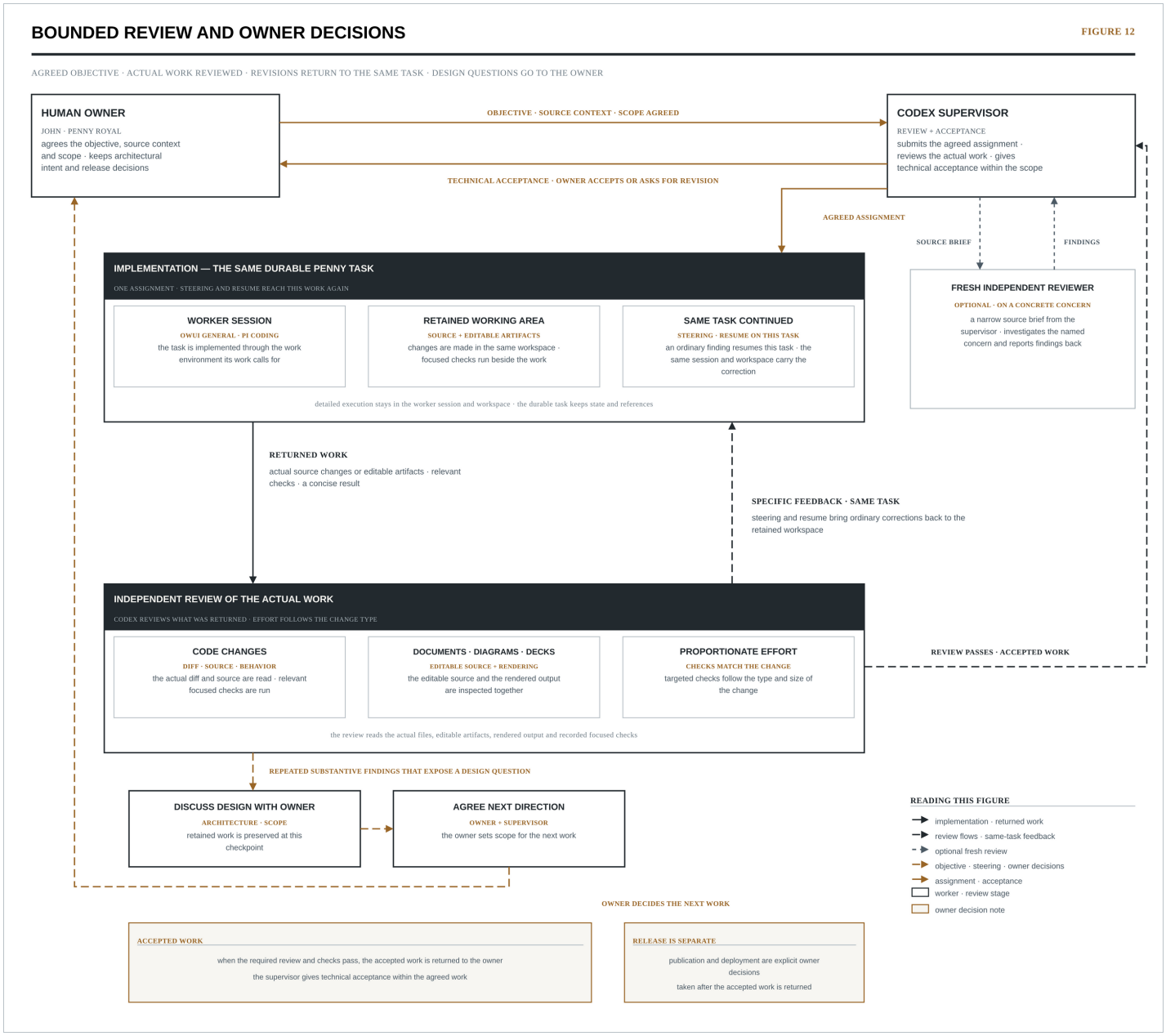


Figure 12 — Bounded multi-agent review and architecture stop

The workflow connects implementation, independent inspection, feedback, and the owner's decision. Durable tasks retain the work across those transitions.

The same pattern extends beyond code. Editable presentation source survives a visual revision. A retained library original survives a later image edit. An owner-approved personal skill carries a useful procedure into a future conversation. Each result has an identifiable owner and a place to persist.

12. Operational visibility

Two collections answer two different questions, and I keep them apart on purpose. One tells me how the platform is behaving. The other tells me what my coding tasks are doing.

The metrics path covers the platform. Penny Royal publishes aggregate runtime counters and node_exporter publishes host counters on thegrid; Prometheus collects both and Grafana displays them. What I read there is workload, cache tier behaviour, latency, resource use, and whether each service is up. Conversation content and user or request identities stay out of it, which keeps the telemetry useful without turning it into a transcript store.

The Pi execution dashboard sits on room101 and covers the coding assignments. My Pi sessions feed it directly: their activity, transcripts, and tool output behind each task, shown view-only to everyone who can reach it. It observes work. It does not accept prompts, answers, resumes, or cancellations. Those control actions stay with the executor and the Codex supervisor, exactly as in Chapter 6.

Both paths are read-only in this architecture. Prometheus holds aggregate numbers with no identities; the Pi dashboard holds task detail drawn from Pi session state.

Figure 13 — Operational visibility paths

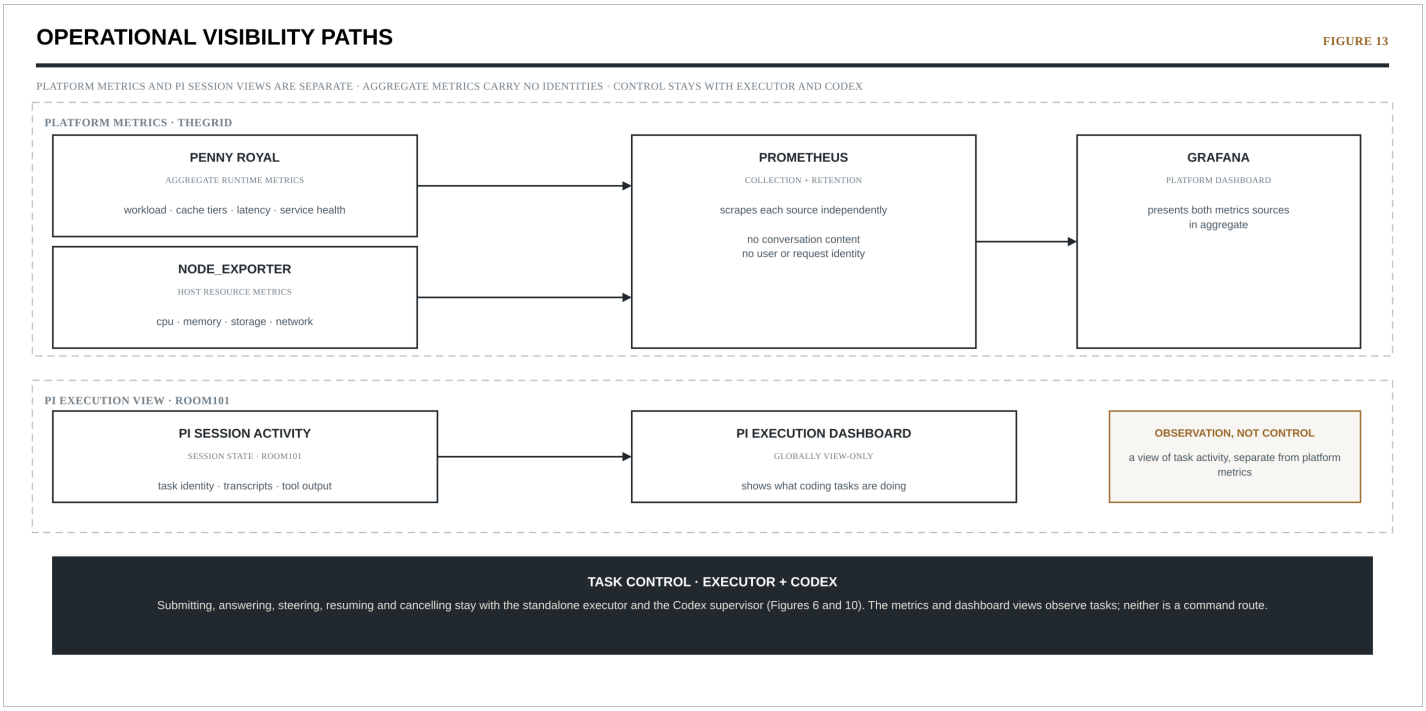


Figure 13 — Operational visibility paths

The figure keeps the two collection paths distinct: counters on one side, task transcripts on the other.

Appendix A. Interface and state reference

The preceding chapters use diagrams to show major paths. The following inventory records the interface contract at each crossing. It is intentionally limited to architectural responsibility rather than ports, credentials, or installation commands.

Service interfaces

Caller	Callee	Interface and payload	Identity and returned state
Browser or phone	Caddy and OWUI	HTTPS application session; messages, uploads, and downloads	OWUI authenticates the person and owns chat/file access decisions
OWUI, Pi, or selected Hermes tasks	Penny Royal	OpenAI-compatible model request with context and tool definitions	Inference returns model output and usage; the caller retains its conversation or session
OWUI	Identity router	Authenticated terminal-work requests, including HTTP, streaming files, and WebSocket traffic	Router maps allowed OWUI identity to one native terminal; missing identity is rejected
Identity router	Native Open Terminal	Proxied tool execution under the mapped service	Linux UID owns process, home, browser state, and working files
OWUI	Connected research MCP	Tool invocation and query payload	Grant-controlled result becomes untrusted source material in the chat
OWUI	AI Tools MCP	Named image generation/editing call; ordered inputs where needed	AI Tools returns image content; OWUI persists and displays it
AI Tools	ComfyUI	Tracked image workflow and temporary source inputs	ComfyUI renders and returns PNG bytes, without becoming the durable library
OWUI	Dubline	Speech transcription or synthesis request	OWUI owns browser interaction; Dubline owns media inference and returned text/audio
OWUI upload hook	Versity Agent Library	S3-compatible archive of an OWUI file	Library object gets stable identity; OWUI keeps its display copy
Router library tools	Versity Agent Library	User-scoped list, fetch, publish, rename, and source URL operations	Mapped private/shared S3 identity governs object access
Codex	Standalone Penny executor	General or coding submission; status, result, steer, and resume	Executor owns durable task identity, ledger, and execution references
Executor	OWUI	Authenticated general assignment and continuation	OWUI owns the saved worker chat and agent loop

Caller	Callee	Interface and payload	Identity and returned state
Executor	Pi	Native coding assignment and retained session continuation	Dedicated worker identity owns the coding session, isolated worktree, and outputs; Pi settlement plus bounded process shutdown retires the running invocation and frees capacity, and the retained session and worktree stay available
Executor notifier	Originating Codex task	Validated status event through a restricted receiver	Codex wakes and reads current executor state; event contains no full result
Penny Royal and node_exporter	Prometheus, then Grafana	Aggregate runtime and host metrics on thegrid	Workload, cache, latency, resource, and health readings with no conversation content or user or request identities
Pi sessions	Pi execution dashboard	Read-only view of session activity, transcripts, and tool output on room101	Globally view-only; prompting, answering, resuming, and cancelling stay with the executor and Codex

This inventory also identifies where a substitute implementation could fit. A different model server must preserve the model API and usable context/usage reporting. A different object store must preserve user-scoped object identity and signed processing inputs. A different browser harness would need equivalent authenticated file routes, saved execution state, and tool permissions before the agent workflow would retain the same properties.

State ownership and lifetime

State	Primary owner	Lifetime and recoverability
Model checkpoints and runtime implementation	My Penny Royal runtime on thegrid	Selected model weights, execution configuration, and shared inference endpoint
GPU, host, and NIXL prefix state	Penny Royal runtime and local cache tiers	Reusable computation; disposable when incompatible or lost
OWUI accounts, chats, settings, and uploaded display files	OWUI application state	Persistent application data with account ownership; separate from model cache
Penny task ledger	Standalone executor	Compact assignment status, results, and references; retained across supervisor attention changes
General worker chat	OWUI chat state	Full model and tool work trace for the general assignment
Coding session	Pi session state	Retained coding interaction, tool work, and continuation state; Pi settlement and bounded process shutdown retire the running invocation without discarding the session

State	Primary owner	Lifetime and recoverability
Task review and closeout outputs	Standalone executor	The supervisor preserves useful outputs and explicitly requests release, then executor-owned workspace, session, and patch material are released; otherwise inactive retention expiry applies, retention can be extended, active tasks are excluded, and an incomplete cleanup retries
Selected Hermes task state	Native Hermes harness	Its own prompts, profiles, memory, sessions, tools, and retained artifacts
Task workspaces and Git worktrees	Dedicated native worker identity	Retained through blocked/review states; unpublished source must be handled explicitly
Human user homes and browser profiles	Each user's Linux identity	User-writable, private working state; may be damaged by that user without changing shared tools
Shared capability binaries and skills	Deployment-managed capability tree	Centrally installed, readable/executable by granted users, not writable by ordinary users
Private brand assets and final documentation	Private owner and application/filesystem grants	Deliberately restricted material; excluded from generic shared capability tree
Agent Library originals and metadata	Versity S3-compatible storage	Durable file objects independent of chat and workspace copies; backup/restore needs objects and metadata
Short-lived signed image input URLs	Library signing operation	Temporary service input; not a durable file or user-facing download
OWUI/terminal download links	Authenticated application file routes	Browser access checked by the user session; terminal links depend on the working copy's presence

A common source of architecture errors is to conflate these lifetimes. A NIXL namespace can be cleared while the conversation remains intact. A working file can be deleted while its library original persists. An OWUI chat can contain full worker output while the executor ledger carries only a compact review result. A source checkout can be present on a machine without being the revision selected by the running service. The design assigns each fact to the layer that can prove it.

Control-plane checkpoints

Several transitions intentionally stop automation from acquiring more authority than its input grants.

1. **Account grant:** an administrator assigns OWUI capability groups. The model's instruction text does not create access.
2. **Terminal mapping:** the router resolves the authenticated OWUI user to an explicitly allowed native identity. Unknown users do not inherit a default terminal.
3. **Task submission:** the supervisor supplies objective, boundaries, and a return target. The executor's capacity limit can reject the submission.

4. **Blocked task:** a native question pauses execution while retaining state. The supervisor supplies an answer before the same task resumes.
5. **Personal skill proposal:** the owner accepts, rejects, or revises a staged candidate before it becomes persistent behavior.
6. **Artifact review:** a rendered deck, diagram, code change, or image result is inspected before it is accepted as finished.
7. **Publication or deployment:** reviewed source or customer-facing material moves outward only through a separately authorized decision.

These checkpoints are not a universal human-in-the-loop rule for every tool action. They are placed where this deployment crosses ownership, persistence, or consequence boundaries.

Penny Royal implementation

My public Penny Royal repository contains the runtime implementation, supported profiles, launch configuration, release notes, and full performance and persistence results. This page describes how that runtime fits into the wider platform. The v0.5 PDF captures this edition; the v0.4 PDF and the earlier v0.3 PDF remain available as previous editions.