PROJECT PHILIP

A building *for* the agent

Seven floors. The same architecture Apple shipped for itself. Built end-to-end on the firm's existing primitives — identity, container platform, image trust, API gateways, inference clouds, immutable audit. No new vendor required.

DELIVERABLE	Reference architecture · sandbox PoC
TARGET	Monday · April 27
ENVELOPE	6-axis · cryptographic · auditable

Project Philip — the building, end to end

A plain-language walkthrough of the safe-agent reference architecture · v0.2

April 2026.

The setup

Imagine you walk into a sixty-story office tower on Park Avenue and you need something done.

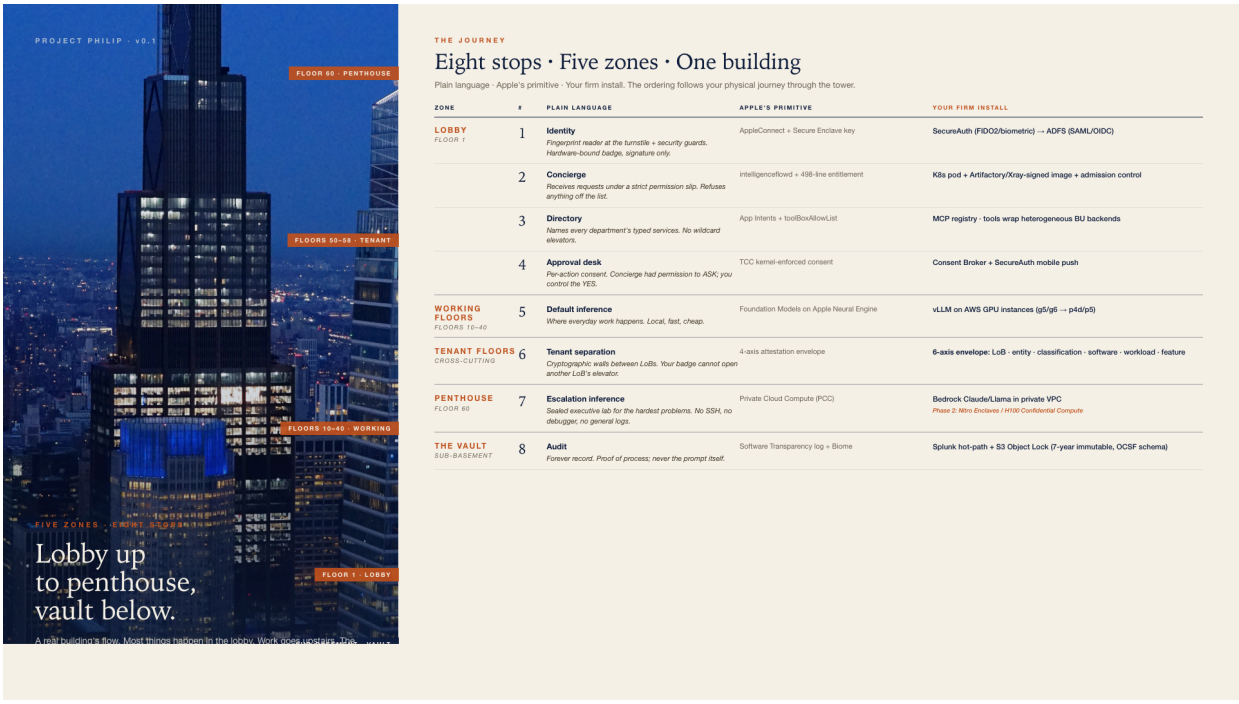

Figure 1. *The complete seven-floor reference, end to end. Plain language · Apple's primitive · the firm install. Each floor solves a problem the other six cannot.*

The building has a lobby. The building has security. The building has tenant floors with their own private elevators. There's a penthouse executive lab on the sixtieth floor with floor-to-ceiling glass and a panoramic view of Manhattan, where the hardest problems get worked on, sealed away from the rest of the building. And below the lobby, behind a heavy steel door, there's the vault — where every entry, every elevator ride, every signature is recorded forever.

Most of these things are invisible while they're working — you only notice them if they fail. That's the whole point.

Philip is your firm's version of that building. Apple already built one for themselves and called the concierge `intelligenceflowd`. We've built ours, calling the concierge Philip — and **a working sandbox PoC of it is currently running on this laptop**, with two model backends and the full safe-agent runtime, end to end. The five zones are the same as Apple's. The rules between them are the same. The mechanism is different because Apple owns their building outright and we're building ours from the systems we already have — but anyone walking through Philip will get the same experience as anyone walking through Apple's, with two notable differences: our envelope is *stricter* (six axes instead of four), and our audit retention is *longer* (seven years immutable instead of ninety days).

Here's the journey through the running PoC, lobby to penthouse to vault.

The lobby — Floor 1

You walk in through the front doors. You're standing in the lobby. Four things happen here, almost all at once.

Stop 1 — The fingerprint reader at the turnstile

The first thing you see is the turnstile. There's a fingerprint reader on it — actually it's reading your phone, which is reading your fingerprint, which is unlocking a key that lives inside the chip in your pocket. The chip will *use* the key, but it will not *show* the key to anyone. Not to the building. Not to the turnstile. Not even to you.

The turnstile sees a signed message — *"this is Kevin, today, on this device, at this moment"* — and lets you through. The signature is fresh every time. There's nothing for an attacker to steal because there's no copy of your key anywhere outside the chip.

The security guards stand a few feet behind the turnstile. If anything looks wrong, they intervene. They are the building's last word on whether the turnstile's signature is genuine.

Apple uses: AppleConnect SSO + the Secure Enclave key. **The PoC uses:** WebAuthn passkeys via `python-fido2`, with the credential held in the Mac's Keychain — which is itself backed by the Mac's Secure Enclave, so we're using Apple's own primitive directly. A passkey was registered for Kevin at first run and validates every session. **The firm install (production): SecureAuth** sits in front and does the hardware-bound part — FIDO2/WebAuthn, push MFA, biometric checks, adaptive-risk gating.

ADFS does the federation: SecureAuth hands ADFS a verified-user assertion; ADFS issues a SAML or OIDC token that Philip's daemon validates. ADFS itself doesn't speak FIDO2 natively — but that doesn't matter, because SecureAuth is doing the FIDO2 work in front of it. Phishing-resistant hardware-bound identity all the way through, using only systems the firm already owns.

Stop 2 — The concierge desk

Past the turnstile, there's a concierge desk in the middle of the lobby. The concierge greets you and takes your request.

The concierge has a **permission slip** taped to the inside of their desk. The permission slip lists every action they're allowed to handle and every floor in the building they're allowed to send a request to. Anything not on the list, they politely refuse. Doesn't matter how nicely you ask. The permission slip is the concierge's hard limit, set by building management — not by the concierge, not by you.

If a clever attacker gets the concierge to *agree* to do something off the list, the building security system itself refuses to let them. The permission slip is enforced one floor higher than the concierge — by the building itself.

Apple uses: the `intelligenceflowd` daemon plus its 498-line code-signed entitlement file. **The PoC uses:** a FastAPI service running inside an OrbStack container, with `manifest.yaml` validated at boot. The container runs as a non-root user with a read-only filesystem; the manifest is mounted as the only writable resource declaration. If the manifest is missing or malformed, the daemon refuses to start. **The firm install:** a **Kubernetes pod** on the firm's internal platform, with the container image hosted in **JFrog Artifactory** and signed by **Xray** at promotion gates. An **admission controller** (OPA Gatekeeper or Kyverno) refuses to start any pod whose image isn't an Xray-signed hash. The "permission slip" is a Kubernetes ConfigMap mounted at boot.

Stop 3 — The directory board

Behind the concierge desk, on the lobby wall, there's a directory. It lists every tenant of the building, every floor they occupy, and every service that floor offers. The concierge looks here to figure out where to route your request.

The directory has **typed verbs only**. You can ask the mailroom to "deliver mail" or "fetch unread mail." You cannot ask the mailroom to "do whatever you want with the entire mail system." Each department has a small list of named actions they perform, with specific paperwork required for each. There is no "wildcard" elevator that takes you anywhere.

This is what keeps a clever attacker from talking the concierge into doing something dangerous. There is no general-purpose action door. There are only specific, named verbs that the building's owners pre-approved.

Apple uses: the App Intents framework, plus a `toolBoxAllowList` that says which features can call which intents. **The PoC uses:** an MCP server (Anthropic's open Model Context Protocol) acting as the registry. Each tool is a Python function with a JSON-schema argument signature. The registry has a per-feature allowlist — for example, the `mail_triage` feature is allowed to call `mail.read` and `mail.draft`, but NOT `calendar.create_meeting`. The daemon checks the allowlist before any dispatch. **The firm install:** the **MCP registry stays the same** — it's a protocol, portable across any environment. What changes is the *implementations* underneath. Different business units expose their APIs differently — some through Apigee, some through service mesh, some through direct mTLS, some through legacy SOAP. *Each MCP tool wraps whatever the underlying API actually uses.* Philip's daemon never sees that mess. It sees one standard tool catalog. **MCP becomes the firm-wide standard for "how an agent reaches a service,"** even though the underlying services were never built to a common standard. That alone may be the single most strategic outcome of building Philip.

Stop 4 — The approval desk

You've made your request. The concierge has looked up the right floor. But before you head to the elevators, there's one more desk to clear.

A clerk at the approval desk asks YOU directly, on your phone: *"the concierge would like to send your request to the mailroom — is that okay?"* You answer yes or no. The concierge *cannot* answer for you. The clerk is enforced by the building, not by the concierge — and they only listen to your fingerprint, your face, or your physical signature.

The concierge had permission to ASK. You have control over the actual yes.

Apple uses: the TCC (Transparency, Consent, and Control) database — kernel-enforced, system-rendered consent prompts. **The PoC uses:** a FastAPI consent broker that issues opaque OAuth-style tokens. For the PoC, the consent prompt is a CLI dialog (the human approves on the same machine, fingerprint-authenticated through the Mac's Touch ID). Each token is scoped to a single tool invocation, expires within minutes, and is recorded in the audit ledger. **The firm install:** a **firm-built Consent Broker service** — a strategic capability Philip creates that every future agent the firm ships can reuse. The broker pushes the prompt through **the SecureAuth mobile app**, the same channel users already use for MFA. Every consent prompt is biometrically confirmed on the user's phone, in a separate channel from the browser. A compromised browser session cannot fake the consent. Stronger than a web modal, equivalent to TCC's kernel-enforced posture.

A pause — where do you configure what the agent can actually do?

Two places, working together. This is the question every Engineering reviewer asks first.

1 — The daemon's manifest (`manifest.yaml`, mounted at the daemon's boot). *The agent's blast radius.* This declares which categories of resources the daemon is *capable* of touching at all — mail, calendar, MCP tools, the local Qwen endpoint, the Anthropic API, the audit DB. Apple's equivalent is the 498-line code-signed entitlement file. Updates to this file require an Engineering pull request plus a Trust & Evals release-veto sign-off.

2 — The per-feature allowlist in the MCP registry (`tool_registry.py` policy section). *Which features can invoke which tools.* For each named feature (`mail_triage`, `calendar_summary`, …), you list the exact tool functions it's allowed to call. Apple's equivalent is the `toolBoxAllowList`. Same Engineering-PR + Trust-&-Evals chain.

Plus two runtime gates that layer on top of those configurations:

3 — The Consent Broker (Floor 1, Stop 4). Even if both manifest and allowlist allow, the *user* has the final yes per action.

4 — The 6-axis envelope policy daemon (cross-cutting). Even if all of the above allow, the tenancy gate can refuse based on line of business, regulated entity, data classification, software version, workload, or feature ID.

Adding a new agent capability typically looks like: write the Python function with a JSON-schema argument signature → register it in the MCP server → add it to the relevant feature's allowlist → if the new tool touches a new resource category, update `manifest.yaml` to grant the entitlement → Engineering reviews via PR → Trust & Evals signs off via the release veto. The whole loop is gitops-style. No agent capability ever lands without going through both reviewers.

The elevators — Floors 10 through 40 — the working floors

You take the elevator up. Your floor is somewhere in the middle of the building.

This is where most actual work gets done. Standard offices, standard teams, standard meeting rooms. The work happens here, fast, without leaving the building.

One important detail in this PoC: there are TWO model backends registered, both selectable, side by side — exactly like Apple's Enchanté. When you ask Philip to summarize your mail, you're given a choice (or a comparison view), the same way Enchanté shows Apple Foundation Models alongside Claude and Gemini for the same prompt. The PoC's UX deliberately mirrors that pattern.

The two models in this PoC:

- **Local — Qwen 3 8B INT4 on the 3090**, served over an OpenAI-compatible endpoint via vLLM. Fast (~50–100 tokens/sec for one user). Cheap (free, runs on your own hardware). No network round-trip outside the LAN. Great for routine summarization, drafting, classification.
- **Cloud — Anthropic Claude Sonnet 4.5, called directly via the Anthropic API.** Smarter (frontier capability for complex reasoning, longer-context, higher fidelity outputs). Slower per-token but lower per-task time on hard work. Costs a few cents per query. Used when the local model isn't good enough.

Both models are reached through the same MCP `chat` tool, with a `model` parameter (`local-qwen` or `anthropic-claude`). Both go through the same envelope (the request carries the model name as part of the workload axis), the same consent flow, and the same audit ledger. Switching is a one-line UI choice — there is no model-specific code path the user can see.

Apple uses: Foundation Models running on the Apple Neural Engine (local) plus white-label Claude / Gemini weights inside Apple's PCC (cloud). **The PoC uses:** as above — Qwen 3 8B via vLLM on the 3090 (local) + Anthropic Claude Sonnet 4.5 via the Anthropic API (cloud, *not* Bedrock — direct API for the PoC's simplicity). **The firm install: vLLM running on AWS GPU instances** (g5/g6 for early scale, p4d/p5 for larger open-weights models) inside the firm's VPC for the local leg. **Bedrock private VPC endpoint** for the cloud leg (Anthropic Claude, Llama, Mistral all available without prompts ever leaving the firm's VPC). Same multi-model selection UX; same envelope/consent/audit pipeline; just running inside the firm's infrastructure.

The penthouse — Floor 60

Some requests are too complex for the working floors, and even more importantly, some requests need a higher security guarantee than "the vendor still runs the runtime."

The penthouse is the executive lab. Floor-to-ceiling glass, panoramic view of Manhattan, a small clean room behind a heavier door. Inside the room, the hardest problems get worked on — the ones the working floors can't crack alone, and the ones whose data is too sensitive for any vendor-managed

runtime. The room is sealed: no SSH, no debugger, no general-purpose logs leave the room. The work happens. The answer comes back. The room forgets the request the moment the elevator descends.

Even the executives whose offices are next to the lab cannot read what was inside.

Apple uses: Private Cloud Compute (PCC) — Apple-silicon servers running a hardened iOS-derived OS in Apple's data centers. No SSH. No debugger. No general-purpose logging. Hardware-bound Secure Enclave keys on the server. Build images published to a public transparency log within ninety days. **The PoC uses:** *the penthouse property is not exercised in the sandbox PoC.* The PoC's Claude leg uses the standard Anthropic API; the vendor manages the runtime. This is fine for a sandbox demo but does not satisfy "no insider access" for production-sensitive data. **The firm install (Phase 1): AWS Bedrock with private VPC endpoint** — prompts don't leave the firm's VPC, but Anthropic still manages the runtime. Same posture as JPMC, Goldman, and most enterprise Anthropic deployments. **Phase 2 hardening:** upgrade to **AWS Nitro Enclaves** or **NVIDIA H100 Confidential Compute** — that's the firm-equivalent of PCC, the property regulators care most about: even the cloud operator cannot read what's inside.

The tenant floors — every floor above the lobby

Notice something: the working floors and the penthouse aren't the same floors for every tenant.

Different lines of business rent different floors. Your LoB has its own dedicated floors, with its own elevator stops, its own keycards. Your badge cannot open another LoB's elevator. The directory board in *your* lobby doesn't even *list* the other LoBs' departments. The penthouse executive lab is segregated too — your LoB's escalation requests go to your LoB's reserved penthouse session; you cannot see another LoB's results, and they cannot see yours.

This separation isn't a wall someone painted between offices. It's **cryptographic** — the request itself, from the moment it leaves the concierge desk, carries a sealed envelope that says *"this came from LoB X, regulated entity Y, data classification Z, software W, workload V, feature U."* Every elevator the request takes, every door it opens, every desk it passes — they all check the envelope. Block any one of those six labels and the whole request gets sent back to the lobby.

Apple uses: CloudAttestation environments (production / qa1Internal / qa2Internal / quarantine) plus a four-axis envelope `(tenant, software, workload, featureId)`. **The PoC uses:** a request envelope generated by FastAPI middleware on every inbound call, plus a policy daemon that can

short-circuit on any axis. For the sandbox demo all axes are populated with `dev` values; the architecture is real even though the data is single-tenant. **The firm install:** a **six-axis envelope** — broader than Apple's because regulated firms have more dimensions of segregation:

1. **Line of business** — CIB, AWM, CCB, Markets
2. **Regulated entity** — broker-dealer, commercial bank, insurance subsidiary
3. **Data classification** — Public / Internal / Confidential / Restricted
4. **Software** — which build hash of Philip is serving the request
5. **Workload** — which model adapter, which routing config, which model backend (Qwen vs Claude)
6. **Feature ID** — which named feature is calling

The policy daemon checks all six. Each axis can short-circuit independently. All six logged. **Bedrock inference profiles** are pinned per-LoB so the cloud itself enforces tenancy at the inference boundary. *This is stricter than Apple's design — call that out in the spec; regulators will care.*

The vault — sub-basement

Below the lobby, behind a heavy steel door, is the vault.

Every time you walked in. Every floor your concierge contacted. Every consent you gave on the approval desk. Every elevator ride to the working floors. Every escalation to the penthouse. Every answer that came back. **Every single one** gets a line in the vault's ledger with a timestamp, a hash of the request envelope, the version of the building software that was running, and the hash of the response.

What's NOT in the vault: the actual content of the request. Not your prompt. Not the answer. Just the cryptographic proof that *something* happened, by *whom*, in *which* version of the building, and that you can later verify nothing's been tampered with.

This is what lets the building owner prove to a regulator: *"we can show you what happened. We cannot show you what was said. By design."*

Apple uses: the Software Transparency log (every served PCC build is published to a tamper-evident log within 90 days) plus structured Biome safety streams. **The PoC uses:** an append-only SQLite `audit.db` for per-request lines, plus a `trust_ledger.db` that records the container image hash and the manifest hash at every boot. A simple CLI lets you query by user, feature, model, or date range. Years from now, the same SQLite file remains queryable; nothing about the schema is vendor-locked.

The firm install: Splunk for hot-path SIEM (search, alerting, dashboards). **S3 with Object Lock in compliance mode** for 7+ year immutable retention — the regulatory baseline for SR 11-7 model-risk evidence. Every Philip event flows in **OCSF** schema (the modern open standard) with the request envelope hash, the agent's service-account SAM, the human's UPN, the model version, and the response hash. *Phase 2 hardening: add a **Sigstore-style append-only public build-attestation log** on top of Splunk + S3 — the firm equivalent of Apple's transparency log, where the published artifact is the build hash, not the prompt.*

What the running PoC demonstrates

Pick any moment from a demo run on this laptop. You can walk down to the audit DB and prove four things from a single SQL query against `audit.db`:

1. **Who** asked for the action — the WebAuthn user-handle that signed the session, plus the Philip session ID that carried the request. *Not the prompt itself.*
2. **What** the concierge was allowed to do at that time — the manifest hash and the per-feature allowlist hash, both signed and immutable in `trust_ledger.db`.
3. **Where** the work actually happened — which model backend (Qwen-on-3090 or Anthropic-Claude), which container image hash, which envelope.
4. **Whether** anything went wrong — every safety classifier event, every refused request, every short-circuit at any of the six envelope axes.

You cannot prove the prompt itself, because that wasn't recorded. That's the *deliberate trade-off*: the firm gets cryptographic proof of process integrity without the firm becoming the custodian of every employee's questions. **Every regulator playbook works against this property; almost no current enterprise AI deployment provides it.**

This is the architecture Apple built for themselves. We have a working demo of it on this Mac.

Where the firm version is *stricter* than Apple

Two places worth surfacing in the leadership presentation:

- **Six-axis envelope** vs Apple's four. The firm's regulatory reality demands segregation by line of business, regulated entity, and data classification — three axes Apple doesn't have at consumer scale. Philip's policy daemon enforces all six.
- **Seven-year immutable retention** vs Apple's 90-day transparency log. SR 11-7 / SOX / SEC retention rules are non-negotiable. S3 Object Lock in compliance mode is the proof artifact.

Two places where Phase 2 work tightens the gap:

- **Workload identity** — gMSA today, SPIFFE/Vault for short-lived next year.
- **Inference attestation** — Bedrock private VPC for Phase 1, Nitro Enclaves or H100 Confidential Compute for Phase 2.

Both gaps are documented honestly in the v0.2 reference architecture spec; neither is a Day-1 blocker.

The single most important thing to internalize

Each zone of the building solves a problem that the other zones *cannot* solve. They are not redundant. They are not "defense in depth in case one fails." They are different problems with different solutions, and the building only works because all of them are present.

Skip the approval desk in the lobby (consent), and the concierge can do anything they have permission to do, even if you don't want them to. Skip the directory board (typed verbs), and the concierge can be tricked into doing dangerous things even within their permission slip. Skip the tenant separation, and one tenant's bad request poisons everyone's penthouse session. Skip the vault, and you can never prove what happened — and the regulator will not accept that.

This is why "just install Claude Cowork" is not the same thing. Claude Cowork is a fantastic concierge. But it's a concierge in a building someone else owns, with a vault you don't control, on tenant floors you don't enforce. That's fine for many things. It is not the same as Philip.

— Project Philip · v0.2 · April 2026