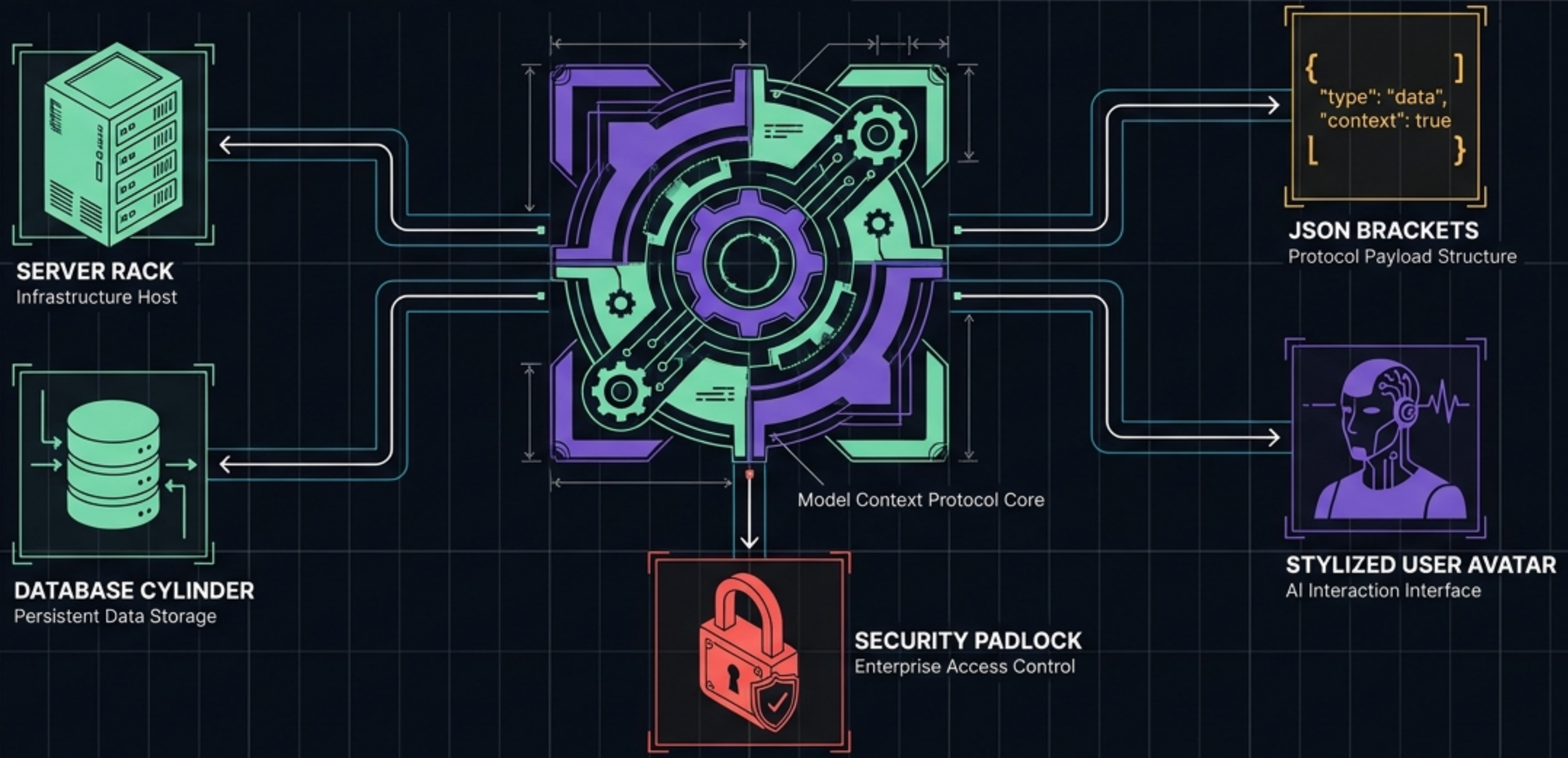
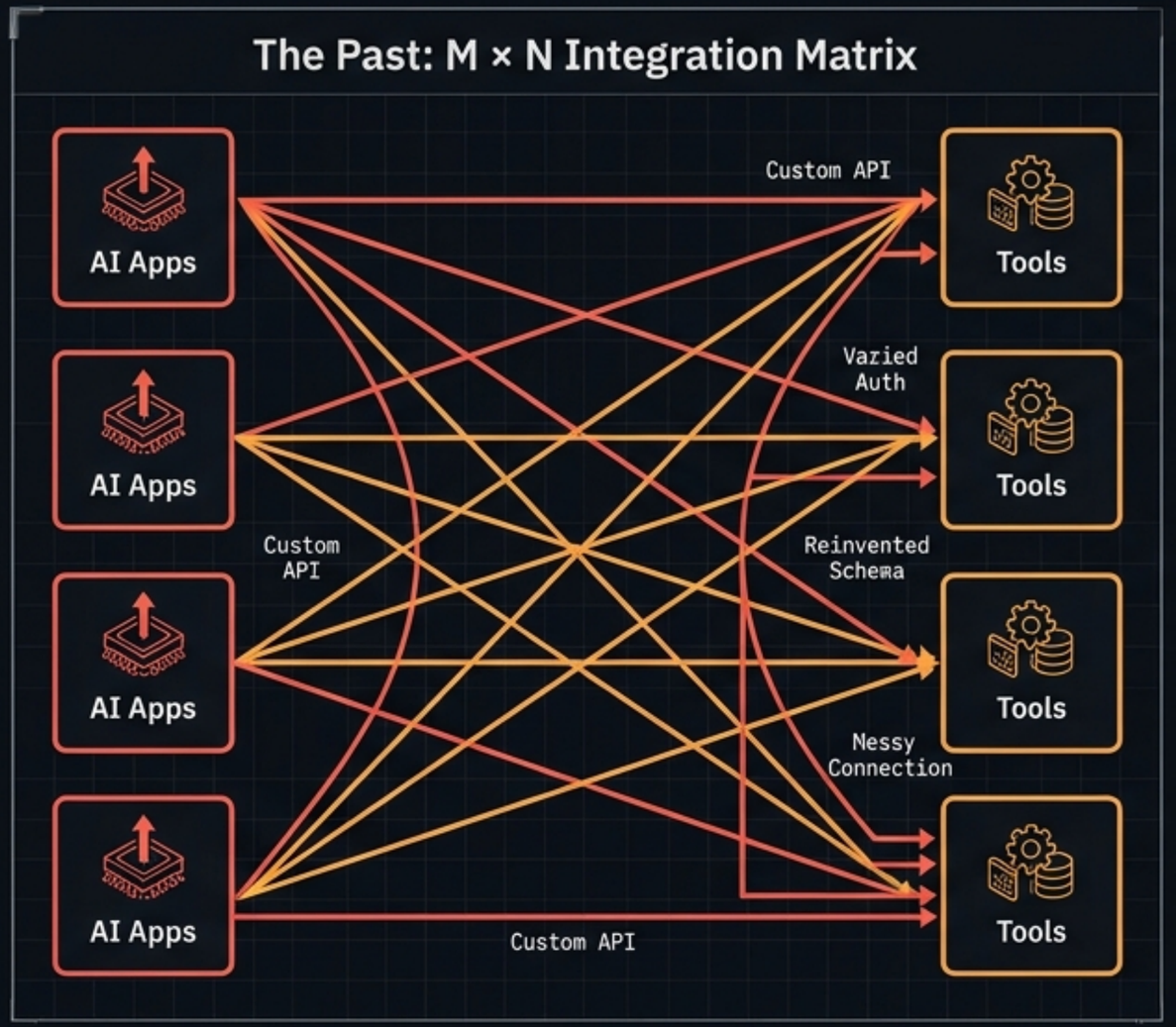
Model Context Protocol (MCP): The System Architecture Blueprint

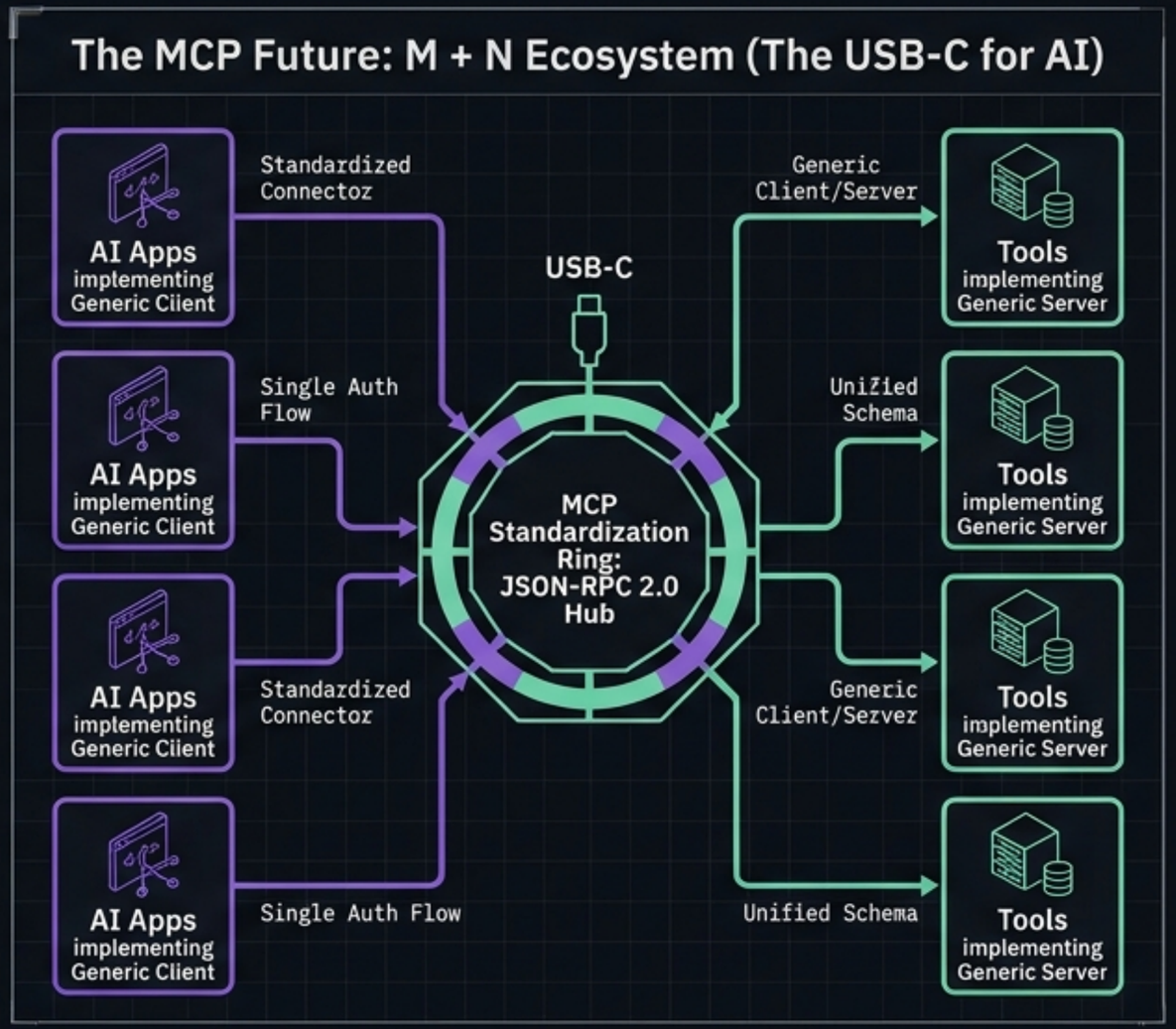
A comprehensive engineering guide to vendor-neutral AI integration, from core primitives to enterprise security (Spec 2026-07-28)



Escaping the M × N Integration Matrix

MCP standardizes how AI applications discover and use external tools and data over JSON-RPC 2.0.
It defines the connector, not the orchestration. Write a server once; every compliant host can use it.





The Architectural Triad: Separation of Concerns



Host

The AI application managing the agent loop.

Owns consent, policy, and what the model sees.

Creates and manages clients.

Aggregates context across isolated servers.

Client

The protocol connector inside the host.

1 Client ↔ 1 Server.

Handles transport, routing, and per-request metadata.



Server

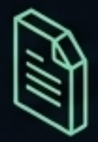
Exposes capabilities (tools/resources).

LLM-agnostic.

Enforces access control.

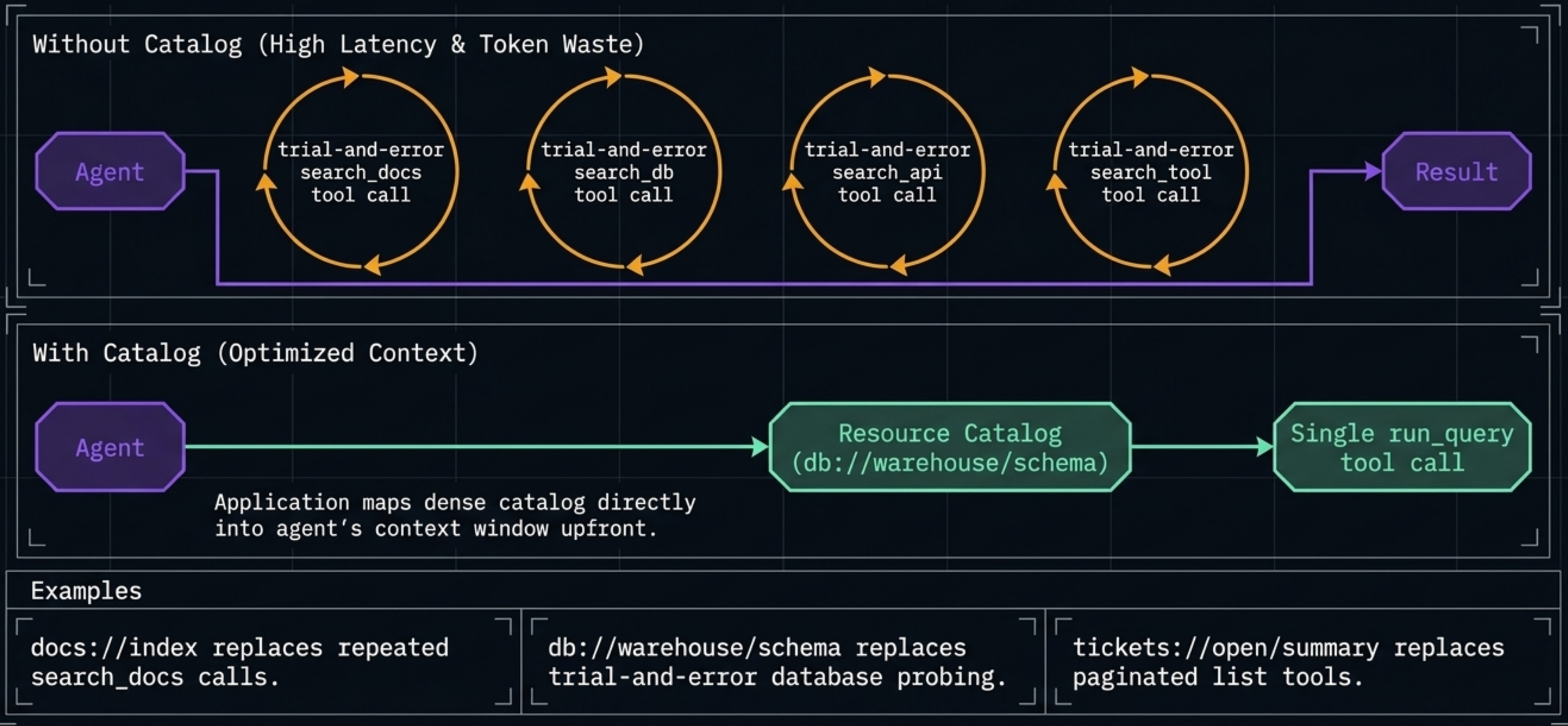
Core Rule: The Host decides which tools are exposed and whether an execution proceeds.

Protocol Primitives and Control Models

Primitive	Control	Purpose	Usage & UI
 Tools (Actions)	Model-controlled (Host-gated)	Executable actions with side effects/computation.	Usage: `tools/list` → `tools/call`. UI: Model decides; host requests human approval.
 Resources (Context)	Application-controlled	Read-only data identified by URI (files, schemas).	Usage: `resources/list` → `resources/read` (and `subscribe`). UI: App attaches via file picker or auto-include.
 Prompts (Workflows)	User-controlled	Reusable, parameterized message templates.	Usage: `prompts/list` → `prompts/get`. UI: Slash commands, menu items.

Strategic Context: Injecting Resource Catalogs

Concept: Without catalogs, agents burn turns exploring. Exposing a compact index as a subscribable resource gives the agent a map before it acts.

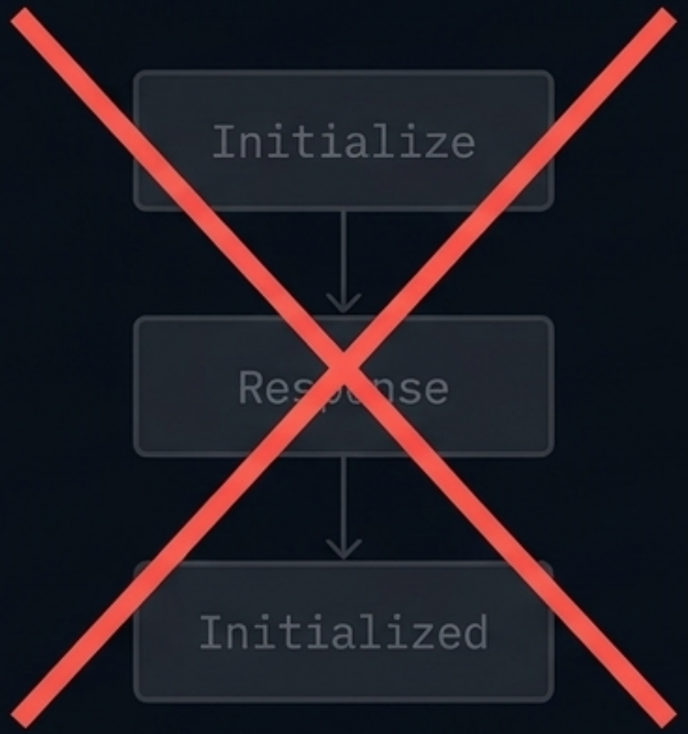


Examples

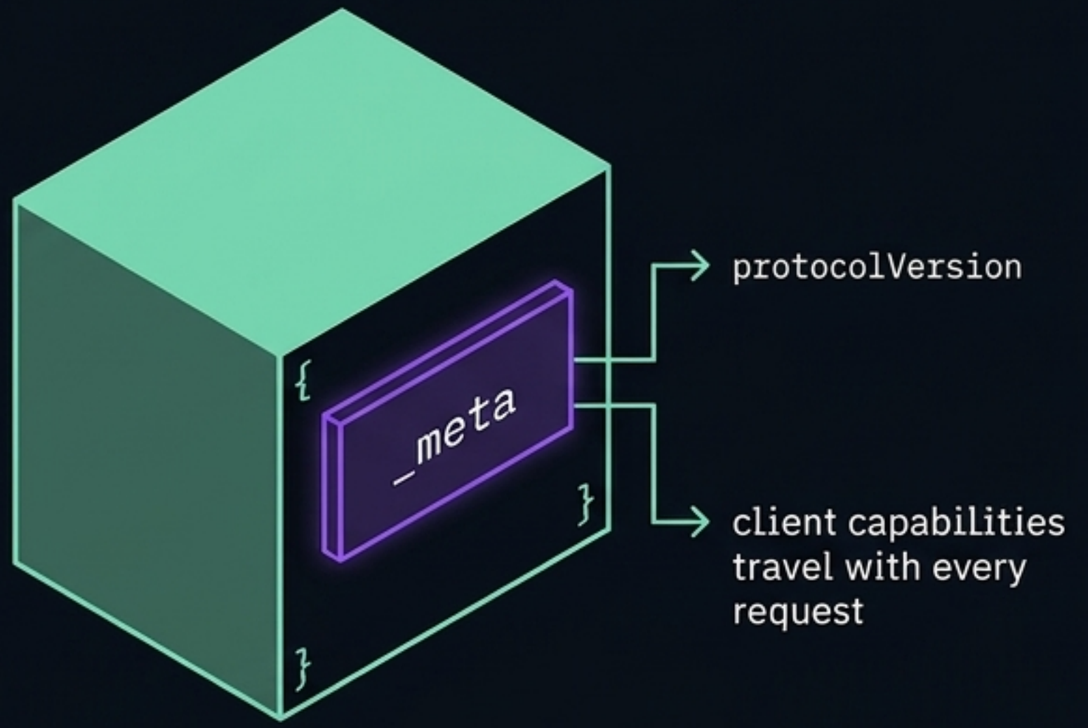
docs://index replaces repeated search_docs calls.	db://warehouse/schema replaces trial-and-error database probing.	tickets://open/summary replaces paginated list tools.

The Stateless Discovery Flow (Spec 2026-07-28)

Pre-2026 Stateful Handshake

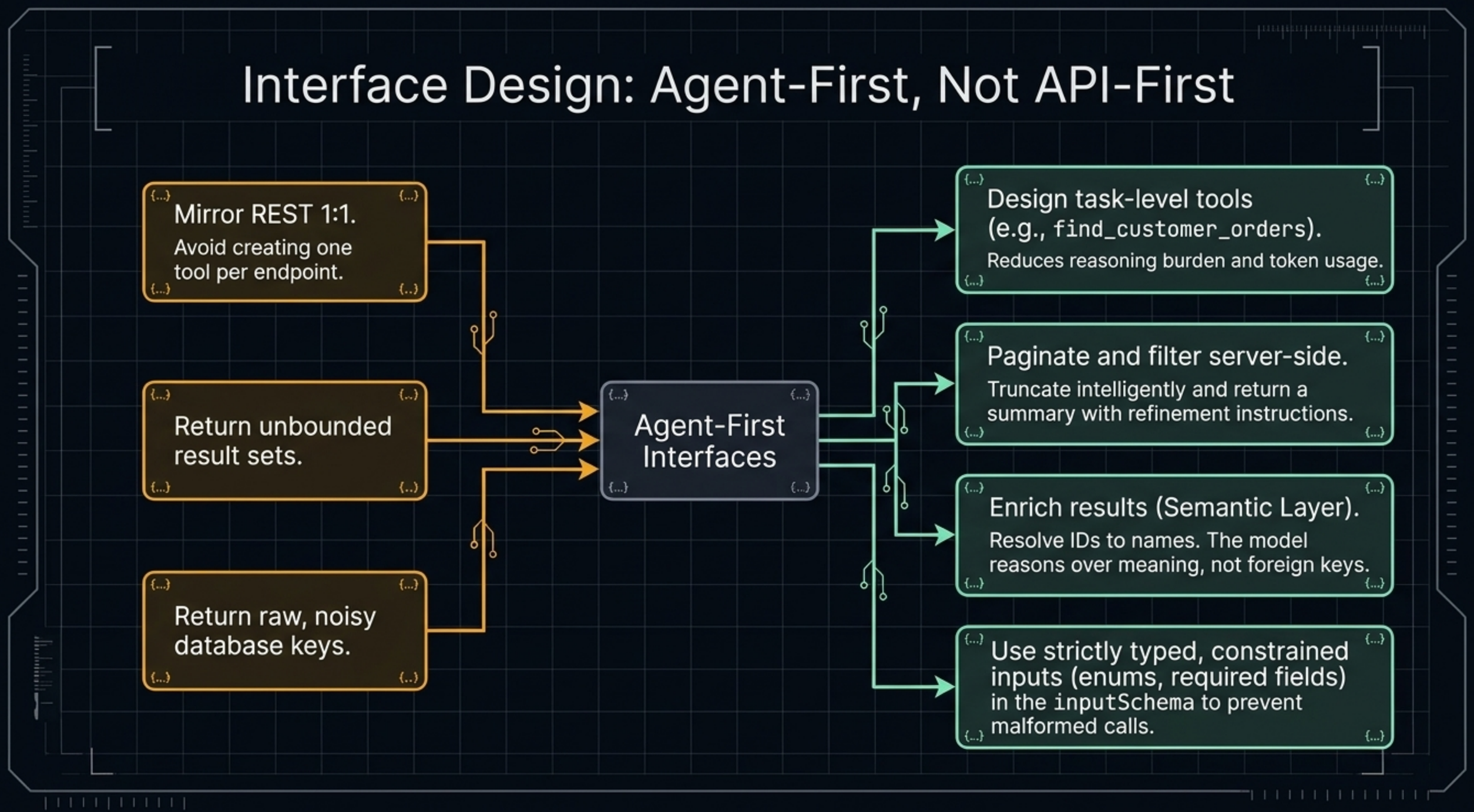


Stateless Request Node



1. **Discovery:** Servers MUST implement `server/discover` for protocol versions, capabilities, and identity.
2. **Per-Request Metadata:** The `initialize` handshake is gone. Every request now carries a `_meta` object.
3. **Pagination:** `tools/list` is paginated via cursor. Servers with `tools.listChanged` capability notify clients dynamically.

Interface Design: Agent-First, Not API-First



Descriptions are the Primary Selection Signal

The Reality: The model never sees your implementation. It picks a tool almost entirely based on name, description, and **inputSchema**.




```
{
  "name": "inventory_get_stock",
  "description": "Retrieves current stock level, reserved quantity, and
warehouse location for a specific product SKU. Does NOT provide historical
sales data.",
  "inputSchema": {
    "type": "object",
    "properties": {
      "sku": {
        "type": "string",
        "description": "The 8-character product code (e.g., AB-12345).",
        "pattern": "^[A-Z]{2}-\d{5}$"
      },
      "warehouse_id": {
        "type": "string",
        "description": "Optional warehouse identifier."
      }
    },
    "required": ["sku"]
  }
}
```

Purpose & Return
`Returns current stock level, reserved quantity, and warehouse...`

Boundary (When NOT to use)
For historical sales, use `get_sales_history` instead.

Input Formatting
`sku`: 8-character code like AB-12345. Not the product name.

Disambiguation Rule
If tools overlap, consolidate them with a constrained parameter (e.g., **report_type**), or strictly namespace domains (e.g., **inventory_get_stock**).

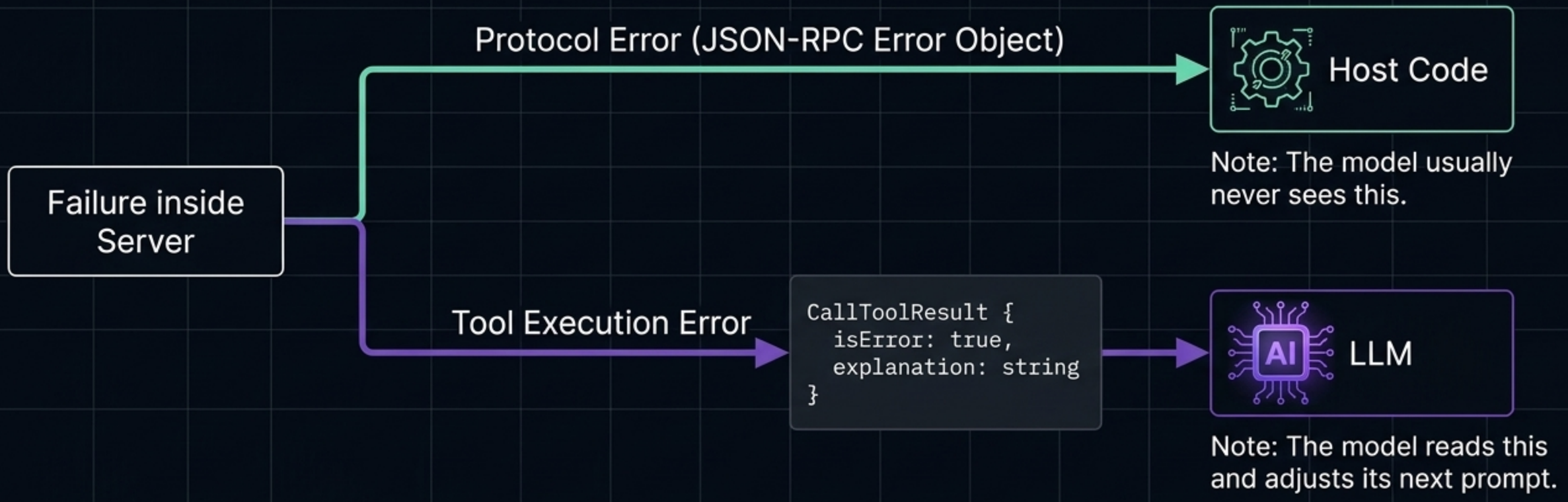
Standardized JSON-RPC 2.0 Error Codes

Principle: MCP strictly reuses standard JSON-RPC codes for protocol-level failures. Do not invent fields to carry meaning—use the code and message.

Code	Description	Meaning
-32700	Parse error	Body is not valid JSON.
-32600	Invalid request	Valid JSON, but missing JSON-RPC structures.
-32601	Method not found	Server does not implement the capability.
-32602	Invalid params	Parameter structure is wrong, or host names a tool that does not exist.
-32603	Internal error	Unexpected server-side fault.
-32000…-32099	Server error	Reserved for implementation-defined faults.

The Two-Channel Error Topology

Protocol Error vs. Execution Error: If a tool runs but fails due to validation or upstream rules, it MUST return as a normal result with isError: true.



The Trap

Catching an exception and returning an empty list []. The model will confidently hallucinate that “no records exist” rather than realizing the backend was unreachable.

Actionable Failures & Host-Driven Retries

Writing Errors for LLMs

A useful ‘content’ string must answer:

1. What failed?
2. Why?
3. What format would succeed?
4. Is repeating the call pointless?

Retries are Host Policy

MCP does not define retry semantics. The host decides, guided by server annotations:

`"idempotentHint: true"` or `"readOnlyHint: true"` → [SAFE FOR AUTOMATIC RETRY WITH BOUNDED BACKOFF]

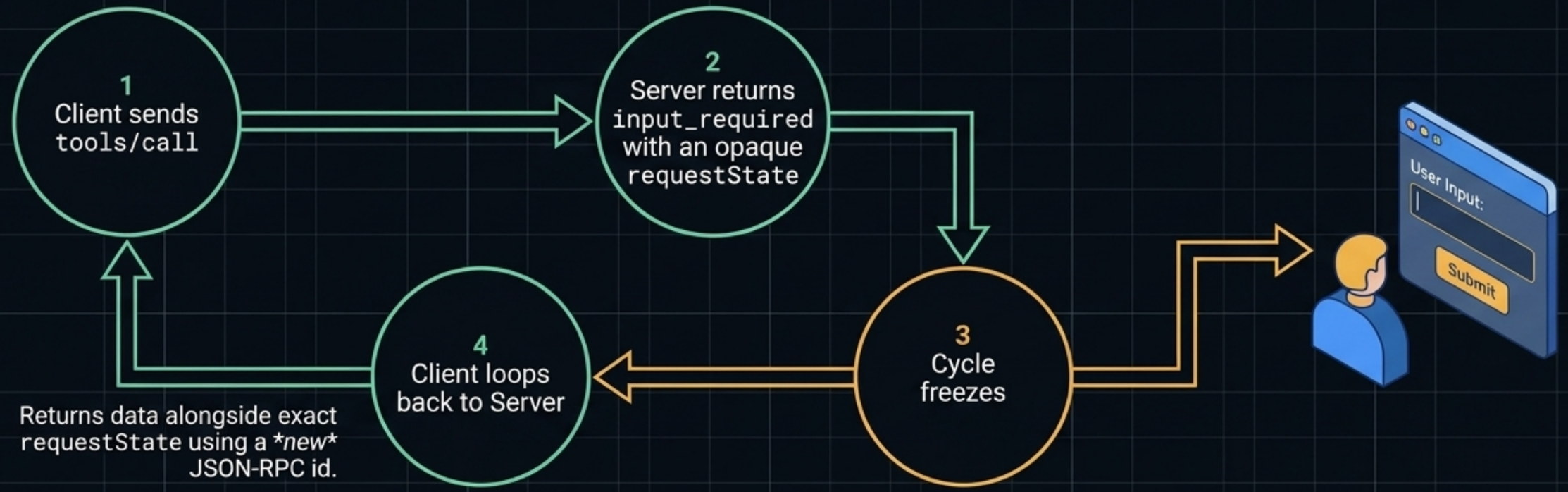
`"destructiveHint: true"` → [HOST MUST NOT AUTO-RETRY BLINDLY. Requires human confirmation or server idempotency keys.]

Security Rule

Annotations are untrusted hints. Bound every retry loop to protect session budgets.

The Elicitation Loop (Multi-Round-Trip Input)

Mechanism: The server pauses execution statelessly. The host renders a request to the user, collects the answer, and retries the call.



Form Mode

Server provides a JSON Schema. Host renders a UI.
Data is exposed to the MCP Client.
Use for confirmations and missing arguments.

URL Mode

User completes flow out-of-band in a browser.
Only the URL is exposed to the client.
Mandatory for sensitive credentials, OAuth, and payments.

Protocol Transports: stdio vs. Streamable HTTP

~~HTTP+SSE~~

Transport Matrix

stdio (Local)

Host spawns server as a subprocess.

Newline-delimited JSON-RPC on `stdin`/`stdout` (`stderr` for logs).

Runs with OS privileges. Uses env credentials.

Use for IDE plugins and local dev.

Streamable HTTP (Remote)

Single endpoint (e.g., `/mcp`).

HTTP POST with JSON or SSE stream replies.

Multi-tenant network service.

Uses OAuth 2.1.

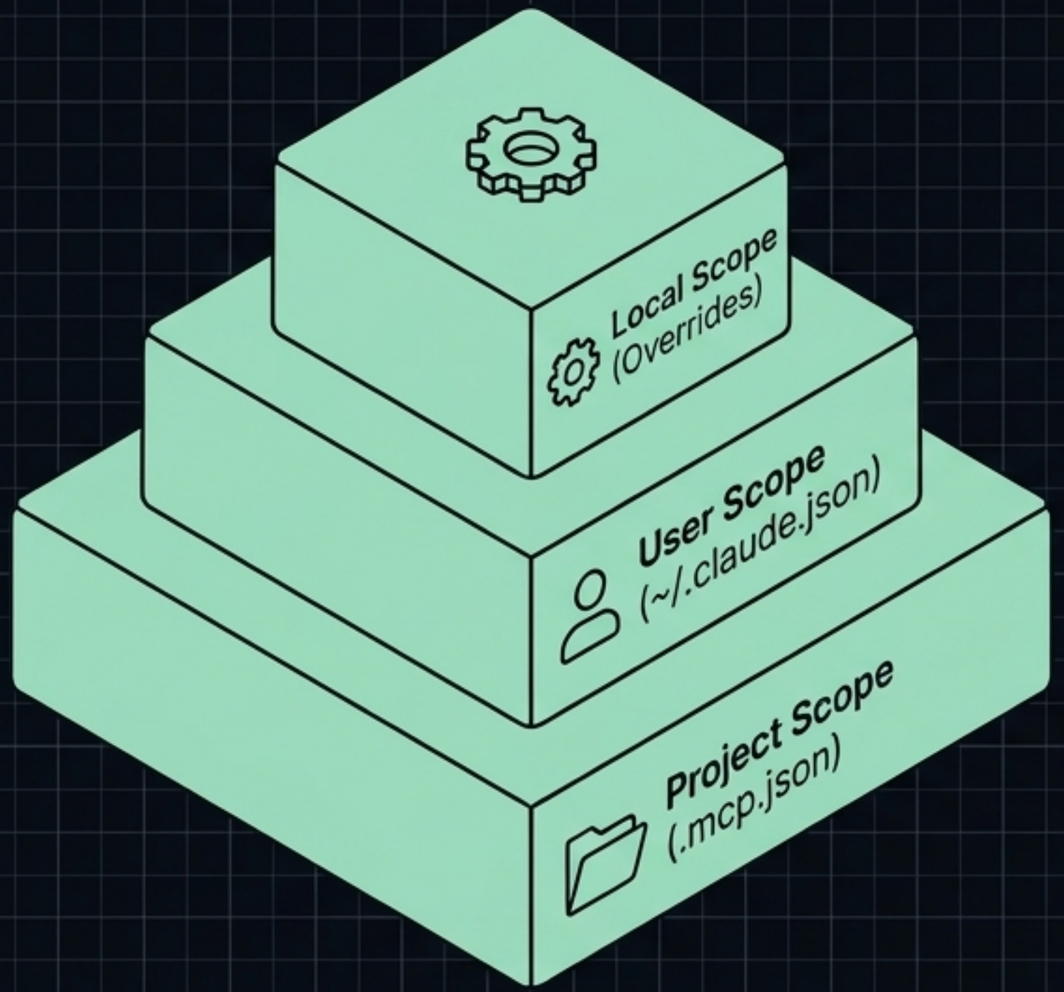
The 2026-07-28 Shift:

Streamable HTTP is now sessionless.

Cloud gateways route and authorize requests without parsing the JSON body using `Mcp-Method`, `Mcp-Name`, and `Mcp-Param-Region` headers.

Long-lived streams use `subscriptions/listen`.

Server Deployment & Configuration Scopes



Project Scope: Config file committed at repo root. Shared by team. Triggers approval prompts on clone to prevent auto-execution.

User Scope: Home directory config. Personal servers used across all projects.

Local Scope: Personal overrides for a specific project.

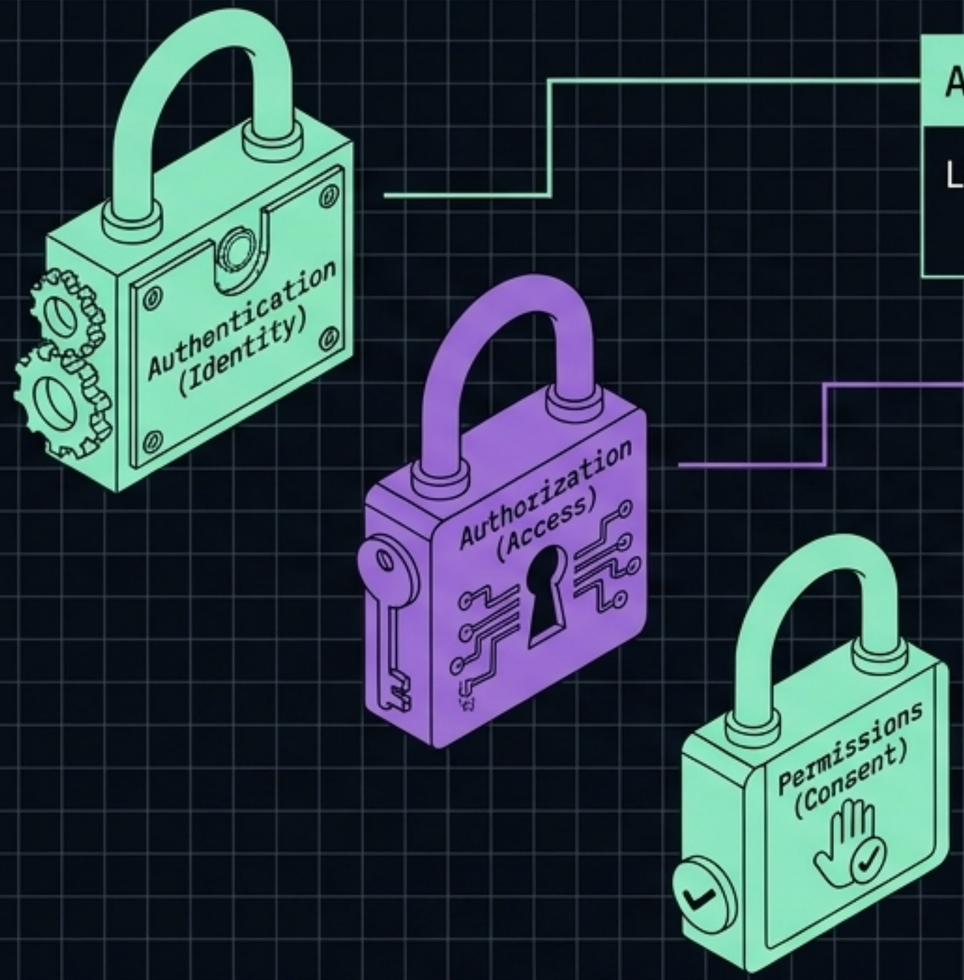
Secrets Management

Committed configs must **NEVER** contain secrets.

```
"args": {
  "apiKey": "${VAR:-default}",
  "url": "https://api.example.com/${ENV_URL}"
}
```

Use `${VAR:-default}` expansion to inject credentials from the developer's environment into args, headers, or URLs.

Authentication, Authorization, and Consent



Authentication (Identity)

Local uses OS/Env credentials. Remote uses OAuth 2.1 access tokens. The MCP server acts as an OAuth 2.1 Resource Server protected by an IdP.

Authorization (Access)

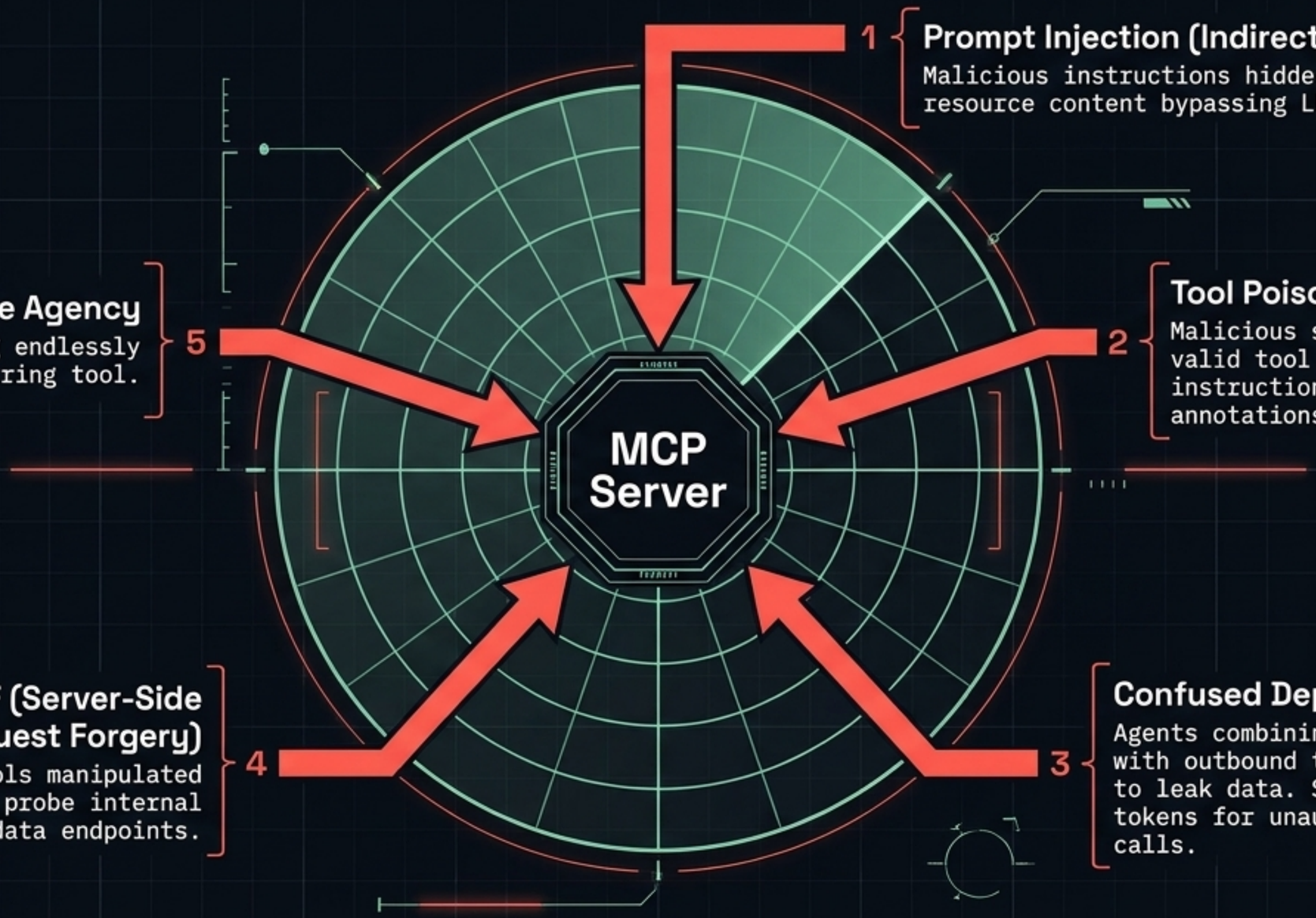
Validated via Token Scopes and Audience. Clients send RFC 8707 Resource Indicators so tokens are minted strictly for one specific server. Server publishes Protected Resource Metadata (RFC 9728).

Permissions (Consent)

Managed by the Host via Human-in-the-Loop (HITL) confirmation and tool allow-lists.

⚠ **CRITICAL RULE: No Token Passthrough.** Servers MUST NOT forward the client's MCP access token to upstream APIs. Prevents confused deputy attacks.

The Threat Landscape



1 **Prompt Injection (Indirect)**
Malicious instructions hidden in fetched resource content bypassing LLM guardrails.

2 **Tool Poisoning & Shadowing**
Malicious servers mimicking valid tool names, or hiding instructions in schema annotations.

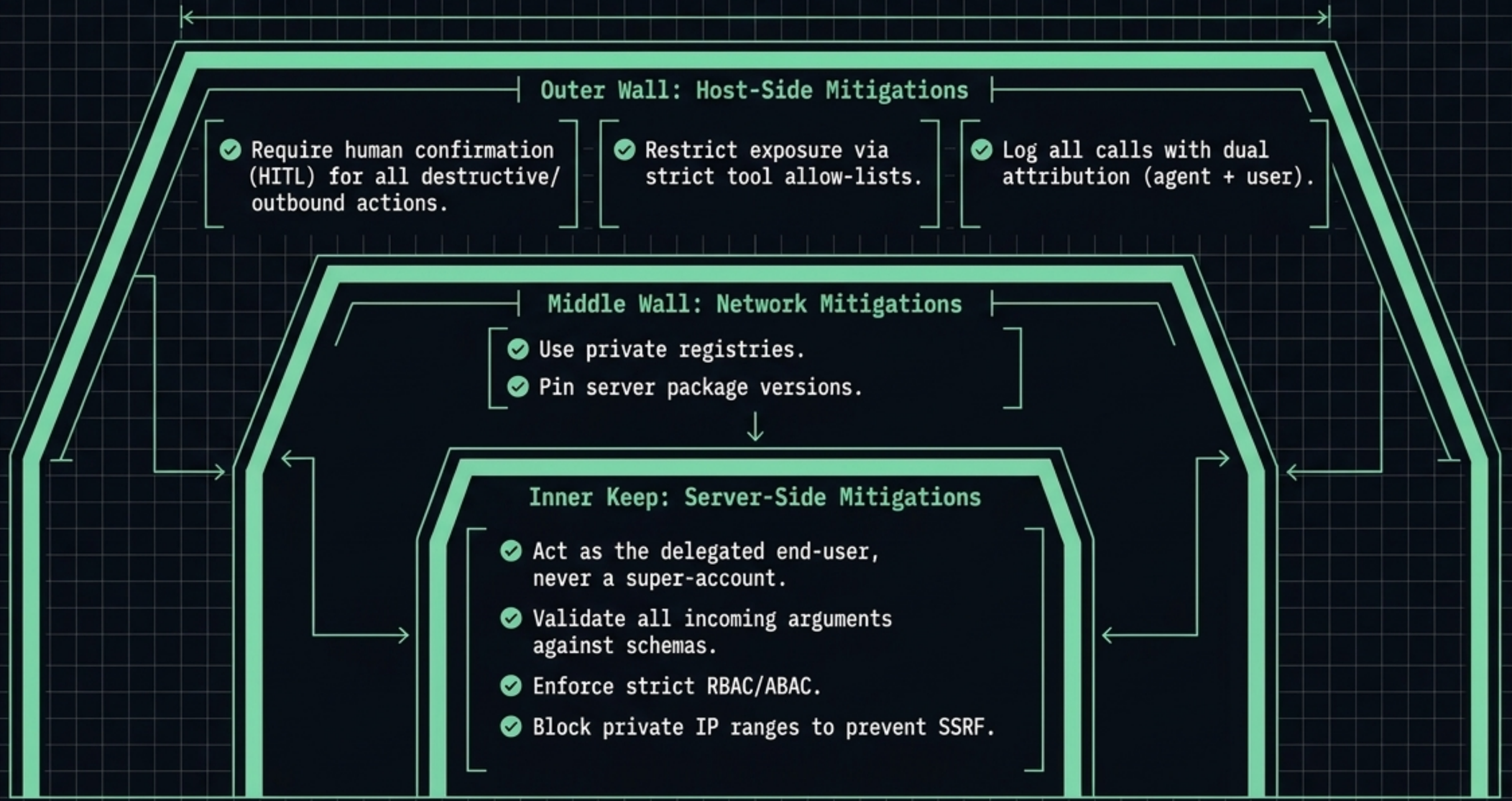
3 **Confused Deputy & Exfiltration**
Agents combining private-data tools with outbound tools (e.g., email) to leak data. Server misusing user tokens for unauthorized upstream calls.

4 **SSRF (Server-Side Request Forgery)**
Fetch-style tools manipulated by the agent to probe internal metadata endpoints.

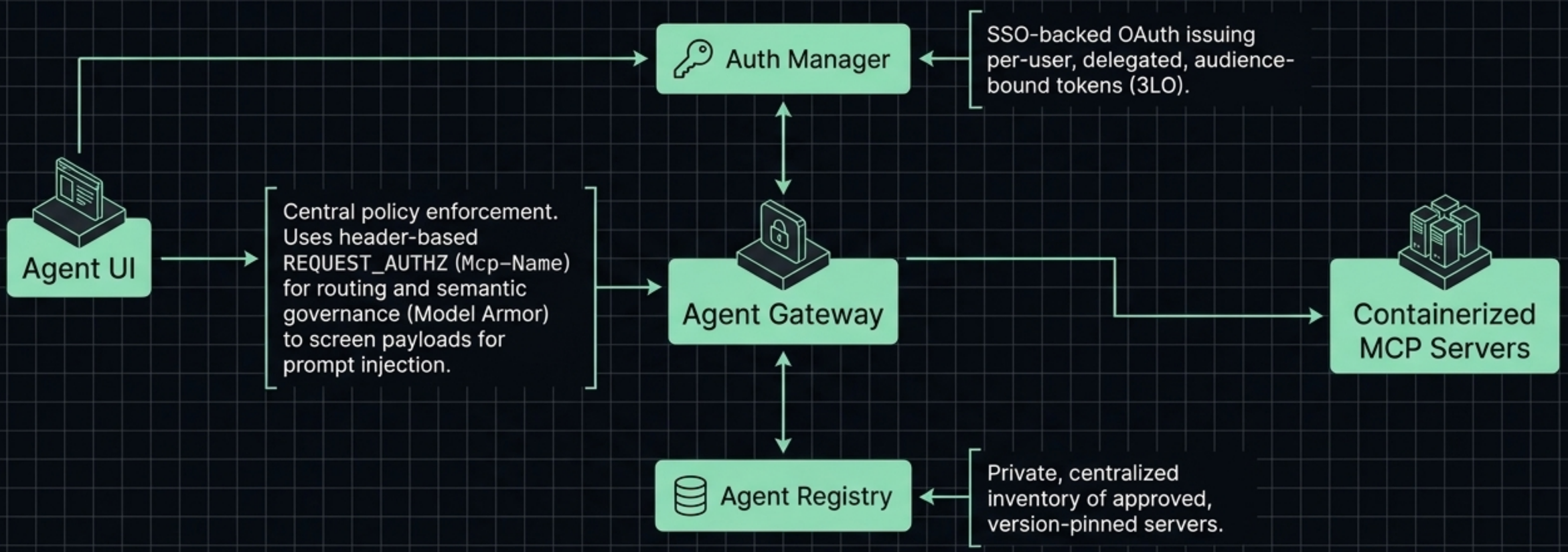
5 **Excessive Agency**
An agent looping endlessly on a cost-incurring tool.

Least Privilege & Defense in Depth

Core Maxim: The model is NOT a security boundary. Assume every description, argument, and result is adversarial.



The Enterprise Integration Blueprint



Observability

OpenTelemetry tracing of every tools/call, capturing latency, outcomes, and redacted arguments (replacing deprecated protocol logging).

Implementation Study: Google Cloud MCP Stack



IAM / Auth Badge

Google-hosted Endpoints

Stateless `*.googleapis.com/mcp` endpoints routed by region headers.

Gated by `roles/mcp.toolUser` and target resource IAM.

Custom Cloud Run Servers

Deployed with `--functional-type=mcp-server`
Sitting behind Cloud IAP and Agent Gateway, passing OIDC auth in `X-Serverless-Authorization`.

Gemini Enterprise Safeguards

Enforces a strict 100-action cap per custom MCP registration.

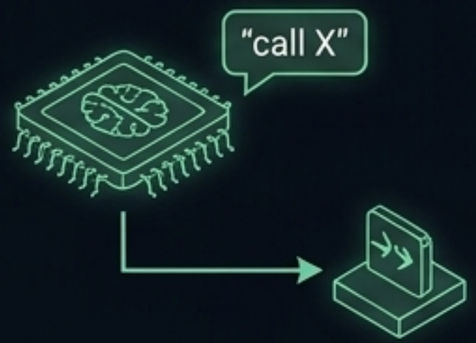
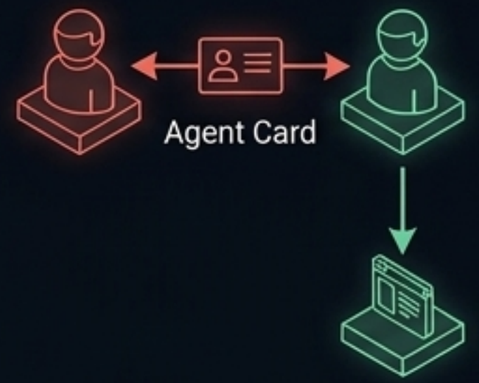
Auto-approves `readOnlyHint: true` but mandates explicit prompts for mutative actions.

MCP Toolbox for Databases pools sub-50ms parameterized queries.

Apigee MCP proxy exposes OpenAPI REST endpoints as governed MCP tools.

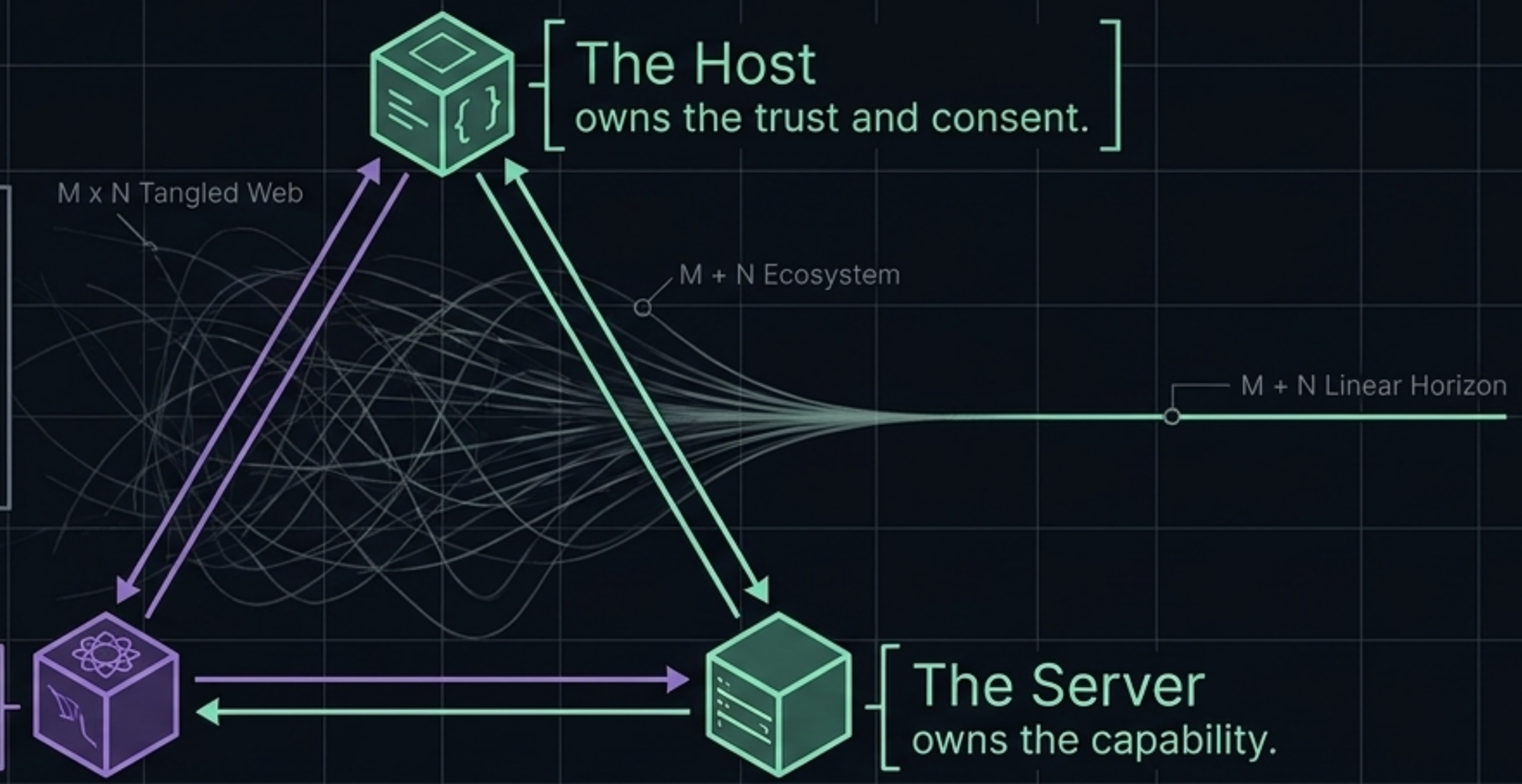
Data & API Bridges

Interoperability: MCP vs. The Ecosystem

Plain APIs (REST/gRPC)	Function Calling	Agent Frameworks (LangGraph, ADK)	A2A (Agent-to-Agent)
Built for deterministic code.	A model API feature where the LLM says “call X”.	Handle reasoning, orchestration, and memory.	Horizontal connectivity to autonomous, goal-driven agents via Agent Cards.
			
Built for deterministic code. MCP wraps these into self-describing, task-oriented tools for AI models.	A model API feature where the LLM says “call X”. MCP is the protocol that provides the definition of ”X” and executes it. They are complementary.	Handle reasoning, orchestration, and memory. act as the MCP Host/Client. MCP provides the pluggable connectivity layer underneath.	Horizontal connectivity to autonomous, goal-driven agents via Agent Cards. MCP is vertical (Agent-to-dumb-Tool).

The Power of a Narrow Contract

Synthesis: MCP is strictly an **interoperability protocol**, not an **orchestration framework**. It deliberately ignores planning, memory, multi-agent routing, and model selection.



Takeaway: By refusing to standardize the agent's brain, MCP successfully standardized the agent's nervous system—making the M + N ecosystem a reality.