The Primitives of Agent Architecture

The Primitives of Agent Architecture

A Reference Guide

By Hadi Nayebi

Hadosh Academy — Agents Series, Essay 4 (Companion Reference)

March 2026

What Is a Primitive?

In agent architecture, a **primitive** is any building block you can name, define, and combine with other building blocks.

Primitives are not always atomic. A plugin is a primitive — even though it is made of hooks, skills, and files. A seed agent is a primitive — even though it contains everything else on this list.

What makes something a primitive is not that it cannot be decomposed further. What makes it a primitive is that **it has a name, a boundary, and a job**. It compartmentalizes one idea so you can think about it separately, combine it with others, and build something larger.

This is how complex systems work. Biology does not build organisms from one giant instruction. It builds them from cells, which are built from organelles, which are built from molecules. Each level has its own primitives. Each primitive has a name and a boundary. The power comes from composition.

Agent architecture works the same way.

Information Primitives

These primitives compartmentalize **what the agent knows**.

File

The most basic unit of persistent information. A file holds text, data, instructions, or configuration. Everything the agent remembers between sessions lives in a file. Everything the agent reads to understand its job comes from a file.

A file is to an agent what a neuron is to a brain — the smallest unit that can carry a signal.

Directory

A container that groups related files. Directories create **compartments** — boundaries that separate one domain of knowledge from another.

A legal case folder is a directory. A project workspace is a directory. The agent's brain (`.claude/`) is a directory. Each one creates a context boundary: what is inside is relevant, what is outside is not.

System Message

The first thing the agent reads. The system message — usually stored in a file like `CLAUDE.md` — tells the agent who it is, what rules to follow, and how to behave. It is the agent's briefing before every session.

Memory File

A file that persists what the agent has learned across sessions. User preferences, project patterns, past decisions. Without memory files, every session starts from zero. With them, the agent accumulates experience.

Knowledge File

A condensed summary of what the agent has learned about a specific topic. Unlike memory files (which track patterns and preferences), knowledge files capture **distilled understanding** — the result of compressing many observations into reusable insight.

Context Window

The only primitive on this list that is temporary. The context window is the LLM's working desk — everything it can see right now. Files, instructions, conversation, tool results — all of it competes for space on the desk. When the desk fills up, old information falls off.

The context window is where all the other primitives converge. It is the meeting point between persistent architecture and live reasoning.

Behavior Primitives

These primitives compartmentalize **what the agent can do**.

LLM (Large Language Model)

The engine. The LLM produces tokens — text that can be conversation, reasoning, or instructions to external systems. It does not think. It predicts what should come next. Every other primitive on this list exists to give the LLM the right context so its predictions become useful action.

Tool

A single capability the agent can invoke. Read a file. Search the web. Run a calculation. Send an email. Each tool does one thing. The agent decides when to use it based on the task at hand.

Tools are the agent's hands. The LLM decides what to reach for. The tool does the reaching.

Hook

An automatic reflex. A hook fires at a specific moment in the agent's workflow — before a file is edited, after a task completes, when the agent tries to stop working. You do not trigger hooks. They trigger themselves.

Hooks are what make an agent disciplined instead of improvised. They enforce rules the way reflexes enforce posture — without conscious effort, every time.

Skill

A set of instructions the agent follows when it recognizes a specific situation. Skills live in files. When the situation matches, the agent loads the skill into its context window and executes it. Skills can be simple (a checklist) or complex (a multi-step workflow with decision points).

Skills are learned behavior. A new agent has few. An experienced one has many.

Command

An explicit action triggered by the user. You type a short name — `/review`, `/commit`, `/brainstorm` — and the agent runs a specific workflow. Unlike skills (which activate automatically), commands activate on demand. They are the buttons on the control panel.

Sub-agent

A specialist. When the main agent faces a task that needs focused attention, it can spin up a sub-agent — a separate instance with its own context window, its own instructions, and a specific job. The sub-agent works independently and reports back.

Sub-agents let the main agent delegate without losing focus. They are the difference between doing everything yourself and having a team.

Connection Primitives

These primitives compartmentalize **how the agent connects to the outside world**.

MCP (Model Context Protocol)

A standard adapter between the agent and an external service. Your email, your calendar, your project management tool, your database — each one speaks its own language. An MCP translates between the agent and the service, so data flows in both directions without manual intervention.

MCPs are what turn a local agent into a connected one.

API (Application Programming Interface)

A programmatic doorway into an external system. APIs predate agents — they are how all software communicates. MCPs are built on top of APIs, adding a standard protocol that agents understand natively. When no MCP exists for a service, the agent can still reach it through its raw API.

Composition Primitives

These primitives compartmentalize **how smaller primitives combine into larger ones**.

Plugin

A bundled package of primitives that adds a complete capability. A plugin might contain hooks, skills, tools, memory files, and instructions — all wired together to serve one purpose.

Think of a plugin as an organ transplant. You do not add one cell at a time. You add a functioning organ — heart, kidney, lung — and it integrates with the body. A plugin does the same for an agent: new reflexes, new knowledge, new skills, all compartmentalized within the plugin’s boundaries.

Persona

The agent's identity. A persona is not a single file — it is the **emergent result** of all the instructions, memory, rules, and style that define how the agent behaves. The system message seeds it. Memory files deepen it. Hooks enforce it. Over time, the persona becomes as distinctive as a person's character.

Seed Agent

A starting template designed to grow. A seed agent ships with a basic set of primitives — some skills, some hooks, some memory structures, a persona outline — and you customize it through conversation. The seed is not the final agent. It is the foundation the final agent grows from.

Agentic Workforce

Multiple seed agents, each specialized for a different domain, working under one human's direction. One handles legal research. One manages billing. One runs marketing. Each agent is a separate composition of primitives, sharing some infrastructure (the LLM, the machine) but compartmentalized in their knowledge, skills, and personas.

How Primitives Compose

Primitives do not exist in isolation. They stack.

Level 1 — Files. Raw information. A single instruction, a single memory, a single data point.

Level 2 — Behaviors. Hooks, skills, and tools turn files into action. A hook reads a rule file and enforces it. A skill reads an instruction file and follows it.

Level 3 — Bundles. Plugins package behaviors and files into self-contained capabilities. Install one, and the agent gains a new set of reflexes and knowledge.

Level 4 — Agents. A seed agent composes plugins, persona, memory, and connections into a coherent identity. Everything below serves the agent's purpose.

Level 5 — Workforces. Multiple agents compose into a distributed system, each one handling a different slice of complexity.

At every level, the same principle holds: **name it, draw a boundary around it, define its job.** That is what makes it a primitive. That is what makes composition possible.

The beauty of this architecture is that you do not need to understand every level to use it. You can work with seed agents without knowing how hooks are implemented. You can install plugins without understanding the files inside them. Each boundary protects you from complexity you do not need to see — until you want to.

Companion to "The Language of Agents" — Essay 4 of 8 in the Hadosh Academy series on agent architecture.