AI AGENTS IN PRODUCTION

One-stop guide: from the fundamentals of agents to building and maintaining them in production

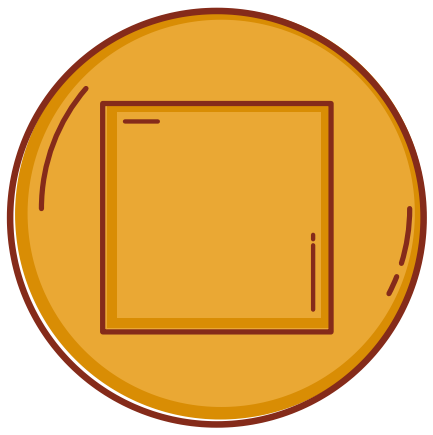
Bhavishya Pandit

THE CHALLENGE

Most AI agents today are demos that break in production. Across all types of AI initiatives, up to **85–88% of pilots never make it to production** often because architecture was ignored in pilot design. Below are some reasons for failure:

Frequent Failures

No Memory



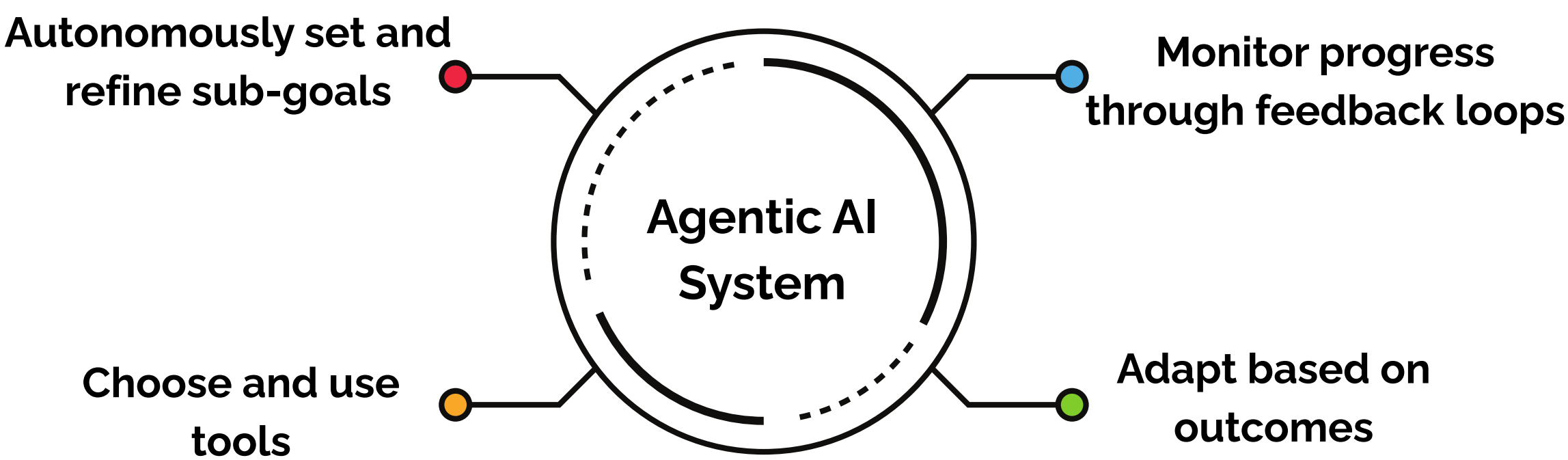
Tool Use is Brittle

Hard to Debug

Don’t worry this post will show you how to build production-grade agentic AI systems from the ground up. First, let’s define what an agentic system is.

What Is an Agentic AI System?

An agentic AI system isn't just a glorified prompt-chain.
It's an architecture built to:



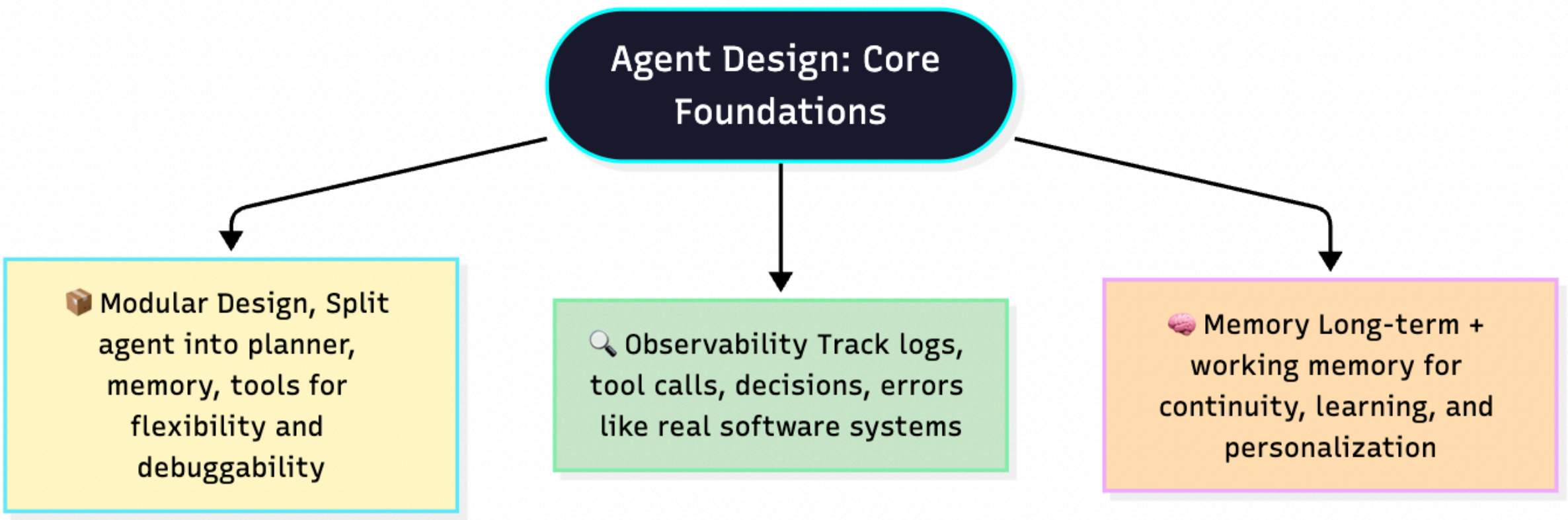
- Autonomously set and refine sub-goals
- Choose and use tools
- Monitor progress through feedback loops
- Adapt based on outcomes

Think: less chatbot, more co-pilot. It doesn't just talk it executes.
👉 And to make that happen, design isn't optional. It's the starting point.

Making Agents Production-Ready

Here's how to build for production, not just proof-of-concept:

1. **Design in Modules, not Monoliths**: Split your agent into roles: planning, execution, memory, tools. This makes it easier to debug, scale, and improve without breaking the whole system.

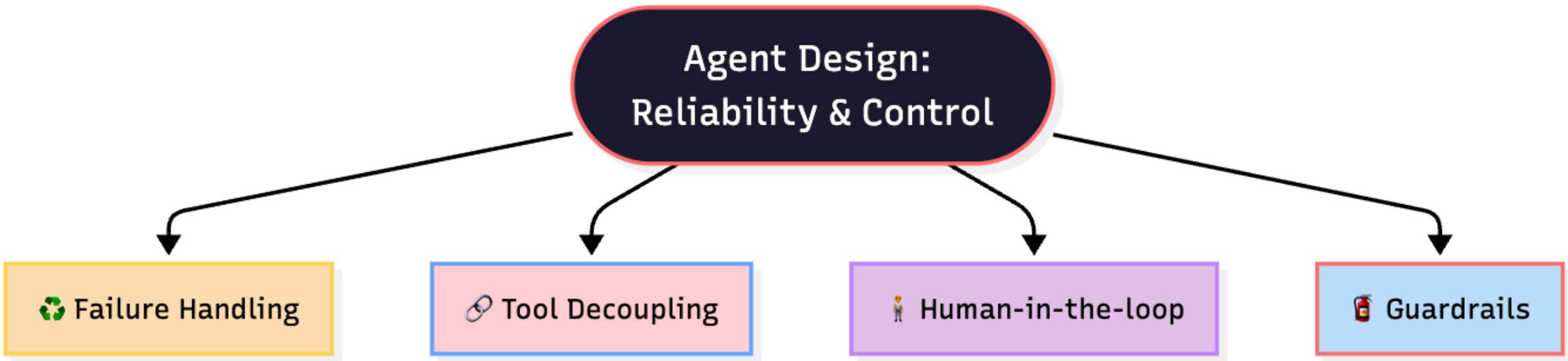


2. **Add observability everywhere**: You can't fix what you can't see. Track every decision, tool call, and failure using structured logs and traces.

3. **Use persistent, evolving memory:** Let agents remember beyond a single chat. Store useful context, past successes, and failures that help the agent grow over time.

4. **Build for Failure not just Success:** Add retries, fallbacks, and checkpoint recovery to make sure one error doesn't end the workflow.

5. **Decouple the toolchain:** Don't hardcode tools like you're writing a script. Let the agent flexibly pick, switch, or skip tools based on what's available.



6. **Keep a human in the loop (when it matters)**: In high-risk tasks, autonomy needs supervision. Let users approve actions, correct outputs, or even interrupt the flow.

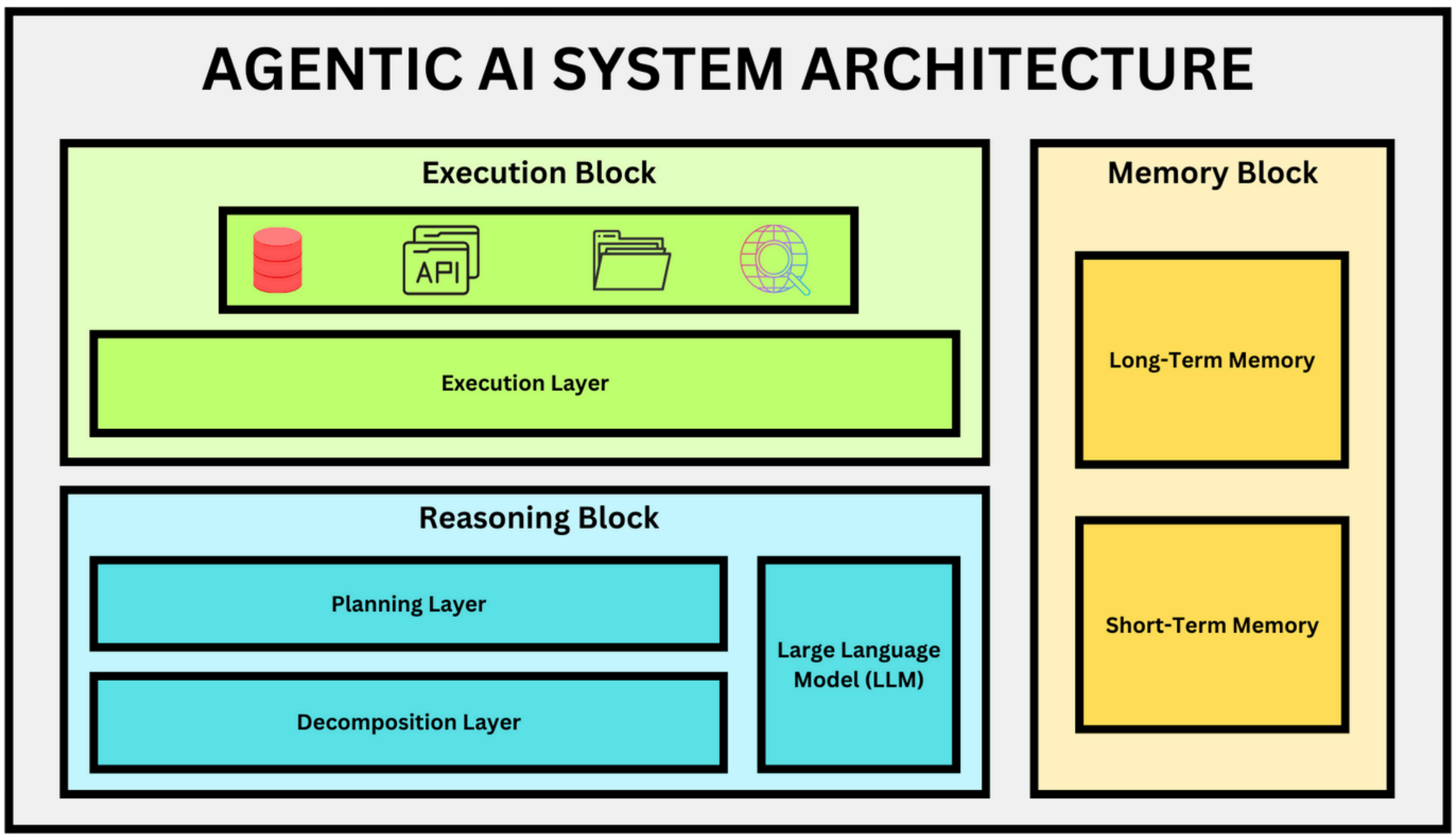
7. **Guard autonomy with guardrails**: Freedom without boundaries leads to chaos. Enforce timeouts, cost limits, and validation rules to keep your agent safe and compliant.

We'll break down each of these ideas memory, planning, tool use, guardrails, and more in the upcoming slides.
Let's get practical. 👇

Start with the Outcome

Bad agent design begins with:
“Let’s use Claude 3.5 or GPT-4…”

Good agent design begins with:
“What’s the outcome this agent should achieve?” Ask:

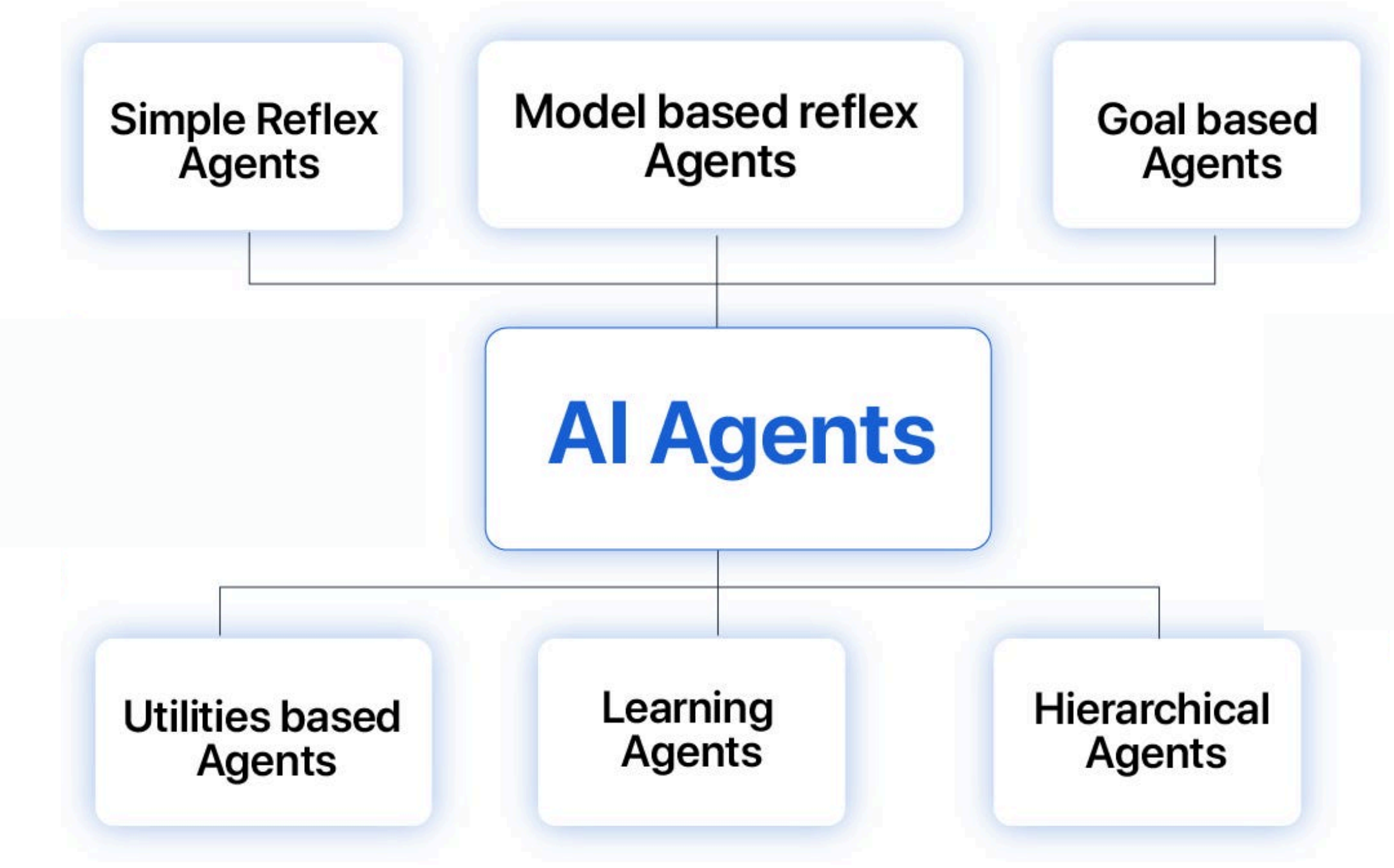


Components of Agentic Design

- Is the goal to retrieve, classify, plan, generate, or summarize?
- What tools, APIs, or user context does it need?
- What does “success” look like and how will the agent know it succeeded?

Pick an Agent Architecture

No two agents are alike. The architecture you pick defines its capabilities.



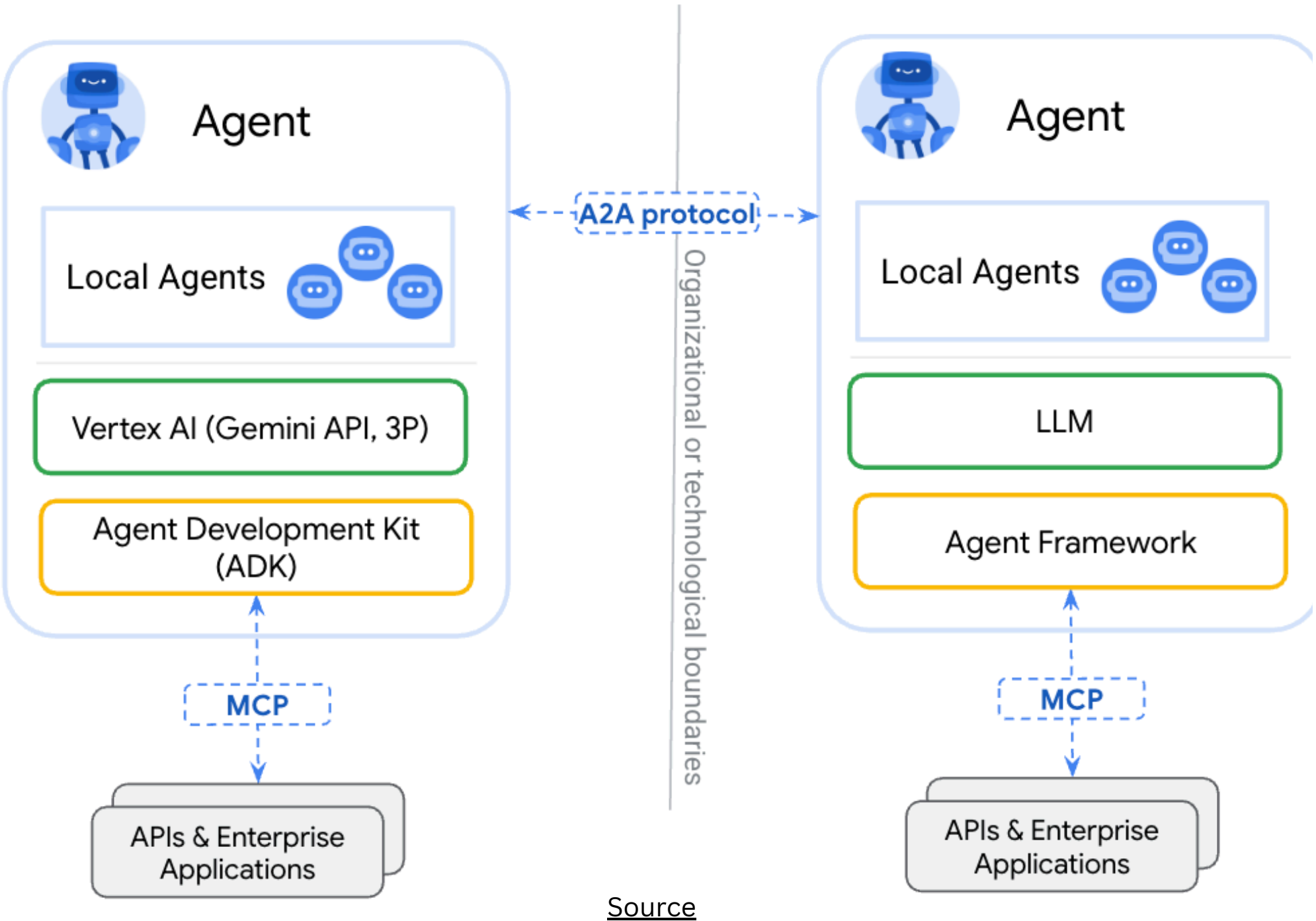
📌 Choose based on your needs:

- Planning vs task execution, Workflow complexity
- Toolchain compatibility, Safety requirements

- **ReAct**: Ideal for reasoning + retrieval in a step-by-step format
- **LangGraph**: Build branching workflows and state-aware agents
- **CrewAI**: Manage teams of collaborative agents, each with roles
- **DSPy**: Declarative agent framework, auto-optimizing the reasoning process

Integrating MCP + A2A

- **MCP**: A structured interface for passing memory, task states, and tools into an LLM ensuring consistency, traceability, and reusability across calls.
- **A2A**: Allows agents to collaborate, delegate tasks, verify work, or share memory in real-time.



Source

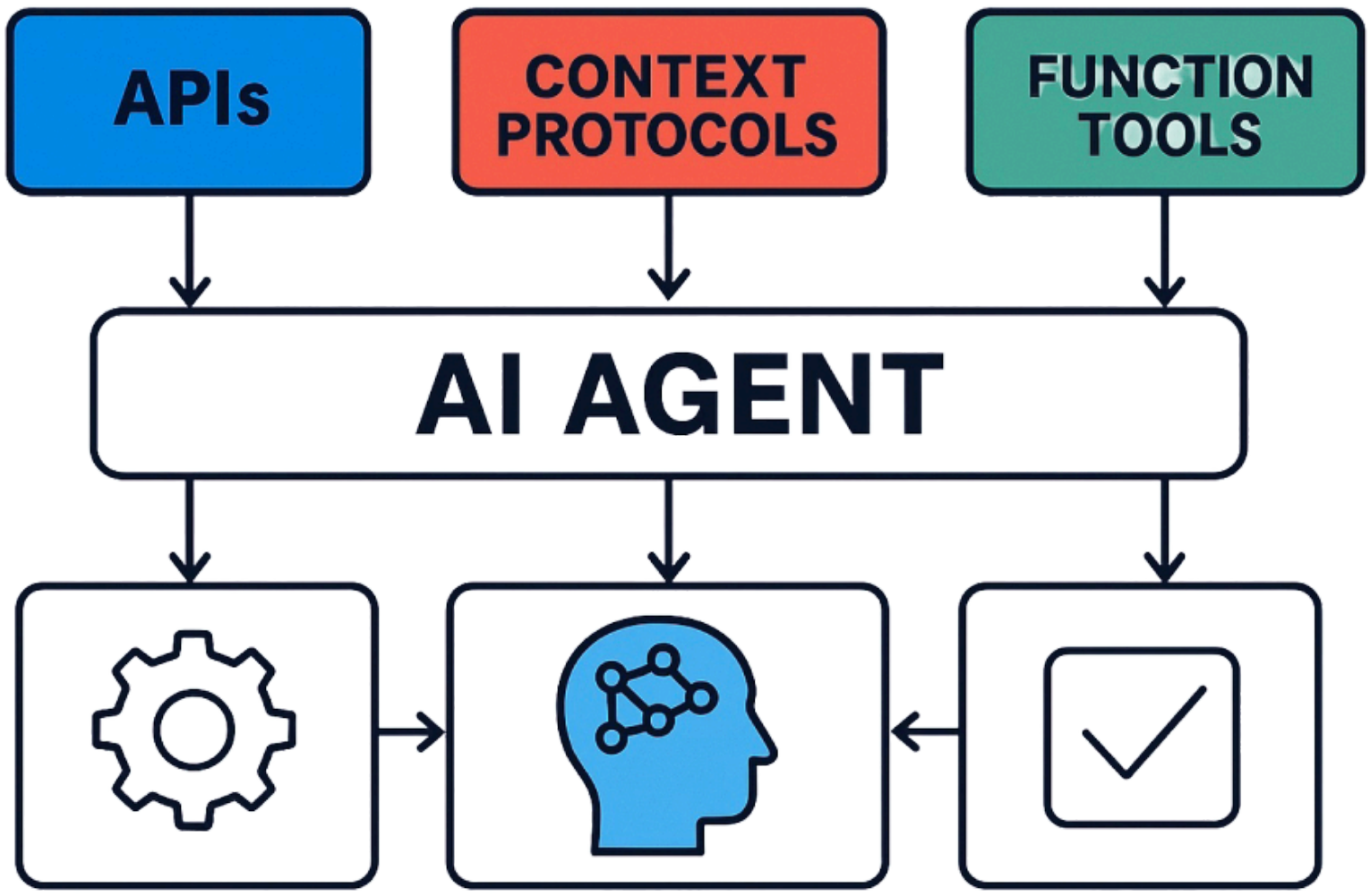
Why Integrate MCP + A2A?

- Maintain Shared Context Across Agents.
- Enable Modular Collaboration.
- Stabilize Multi-Step Reasoning.
- Build Swappable, Interoperable Agents.

How to Actually Integrate MCP + A2A

Tools that help:

- **LangGraph**: Lets you design multi-agent workflows with memory, retries, and message passing
- **CrewAI**: Easily assign roles, goals, tools, and communication paths among agents.
- **DSPy**: Allows routing reasoning substeps across modular agent components



Source

Example Use Case: "Planner Agent"

→ uses MCP to define context

→ calls Retriever Agent via A2A

→ sends results to Analyzer Agent

→ Critique Agent validates

→ Synthesis Agent produces output

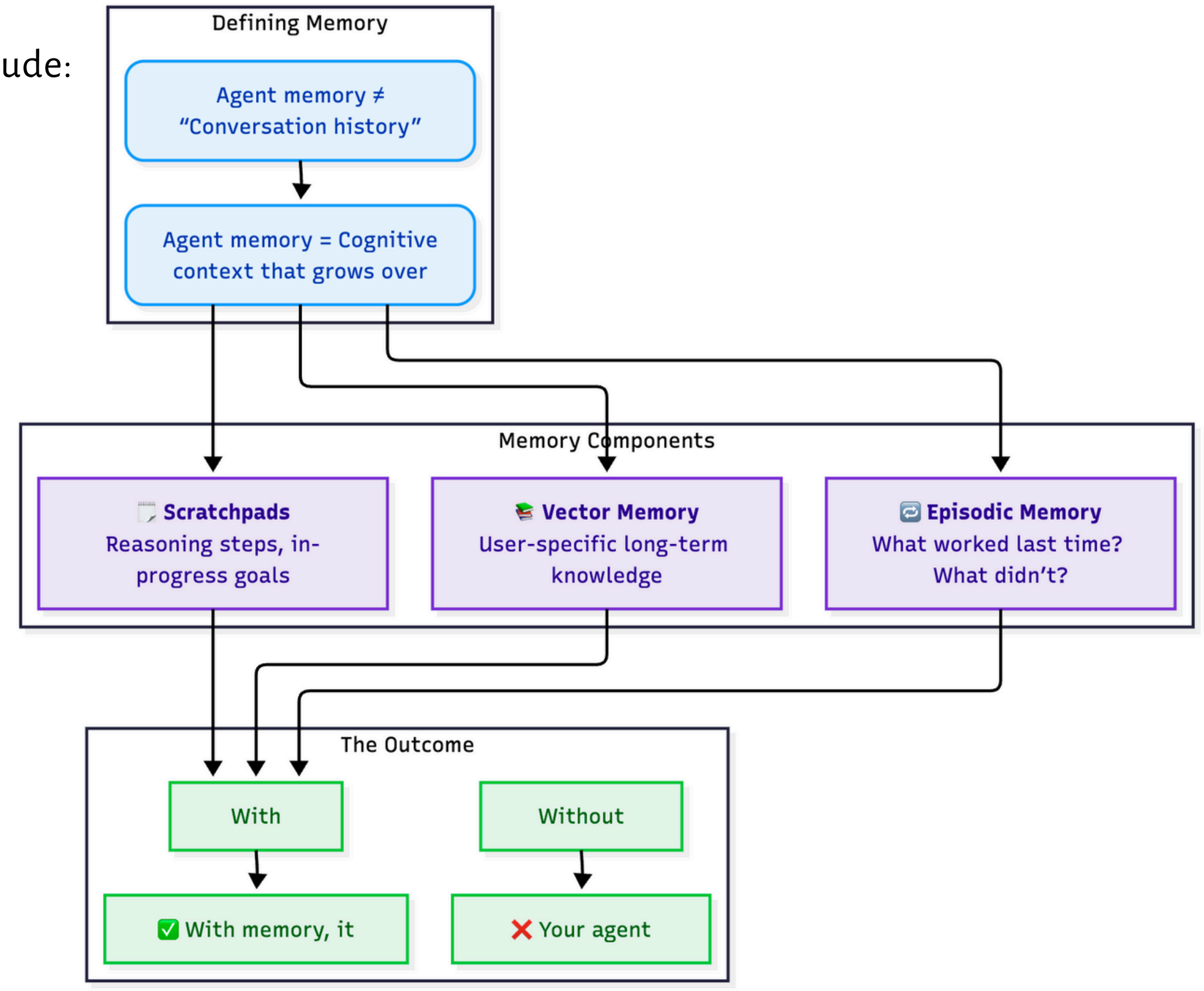
→ all context logged & passed via MCP

Give It a Working Memory

Agent memory ≠ “Conversation history”

Agent memory = Cognitive context that grows over time

Include:



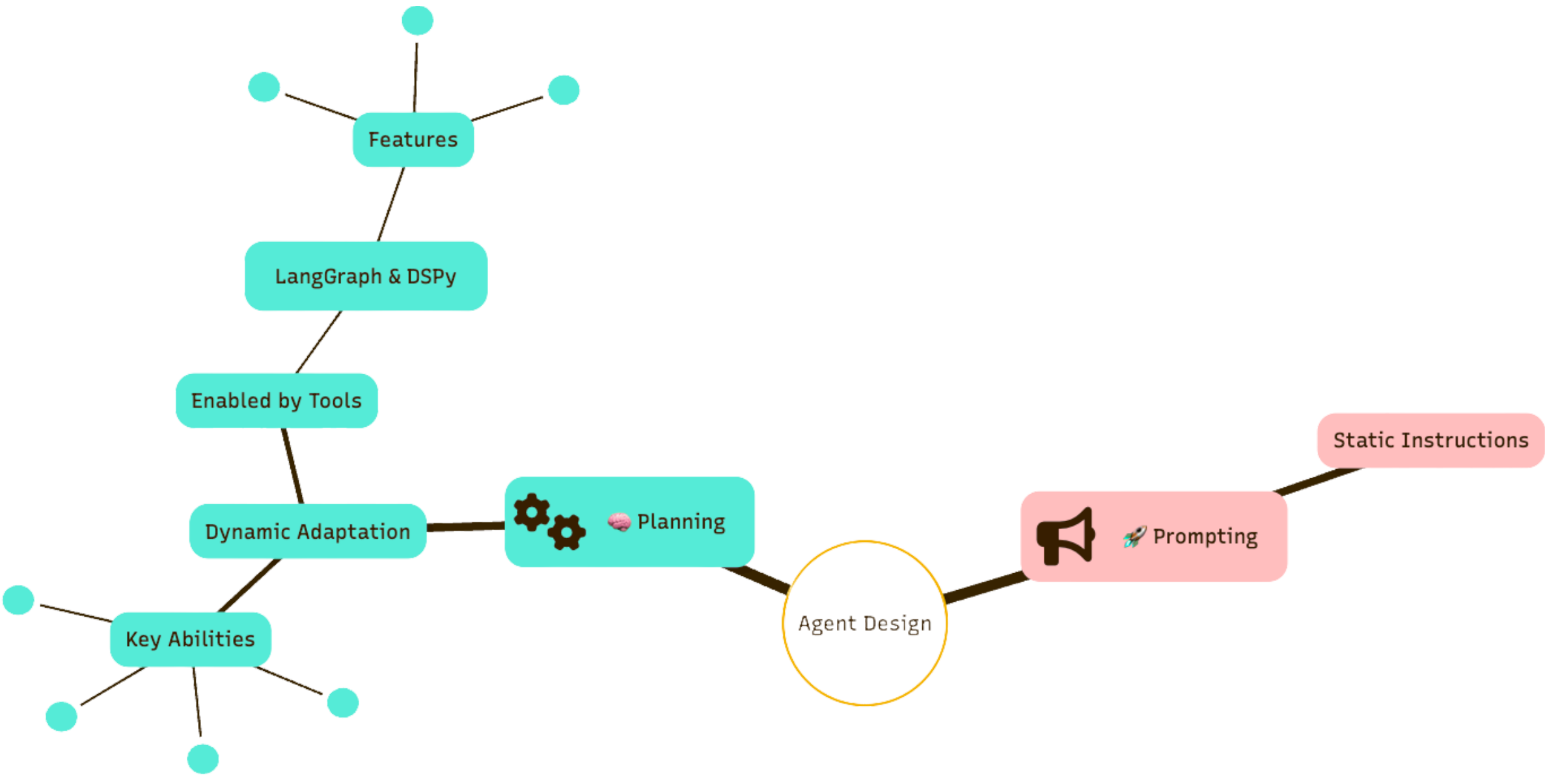
- **Scratchpads** → Reasoning steps, in-progress goals
- **Vector memory** → User-specific long-term knowledge
- **Episodic memory** → What worked last time? What didn’t?

Without memory, your agent repeats. With memory, it evolves.

Planning > Prompting

Prompting is about instructions. Planning is about adapting.

An agent should be able to:

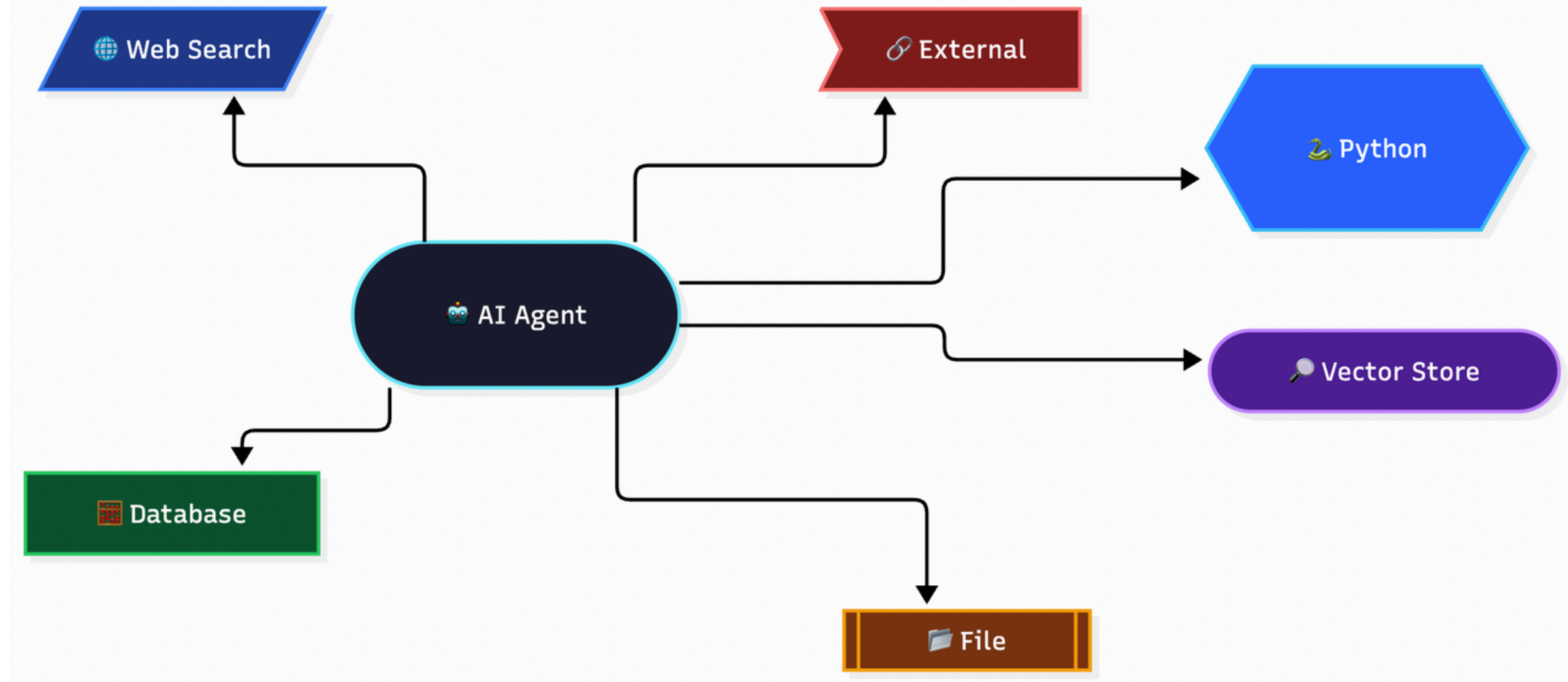


- Break a large task into subtasks
- Decide order of execution
- Adjust plan based on tool results
- Detect when it's stuck and recover

Tools like LangGraph and DSPy are built exactly for this planning with checkpoints, branches, and loops.

Equip the Agent with the Right Tools

Common tools include:



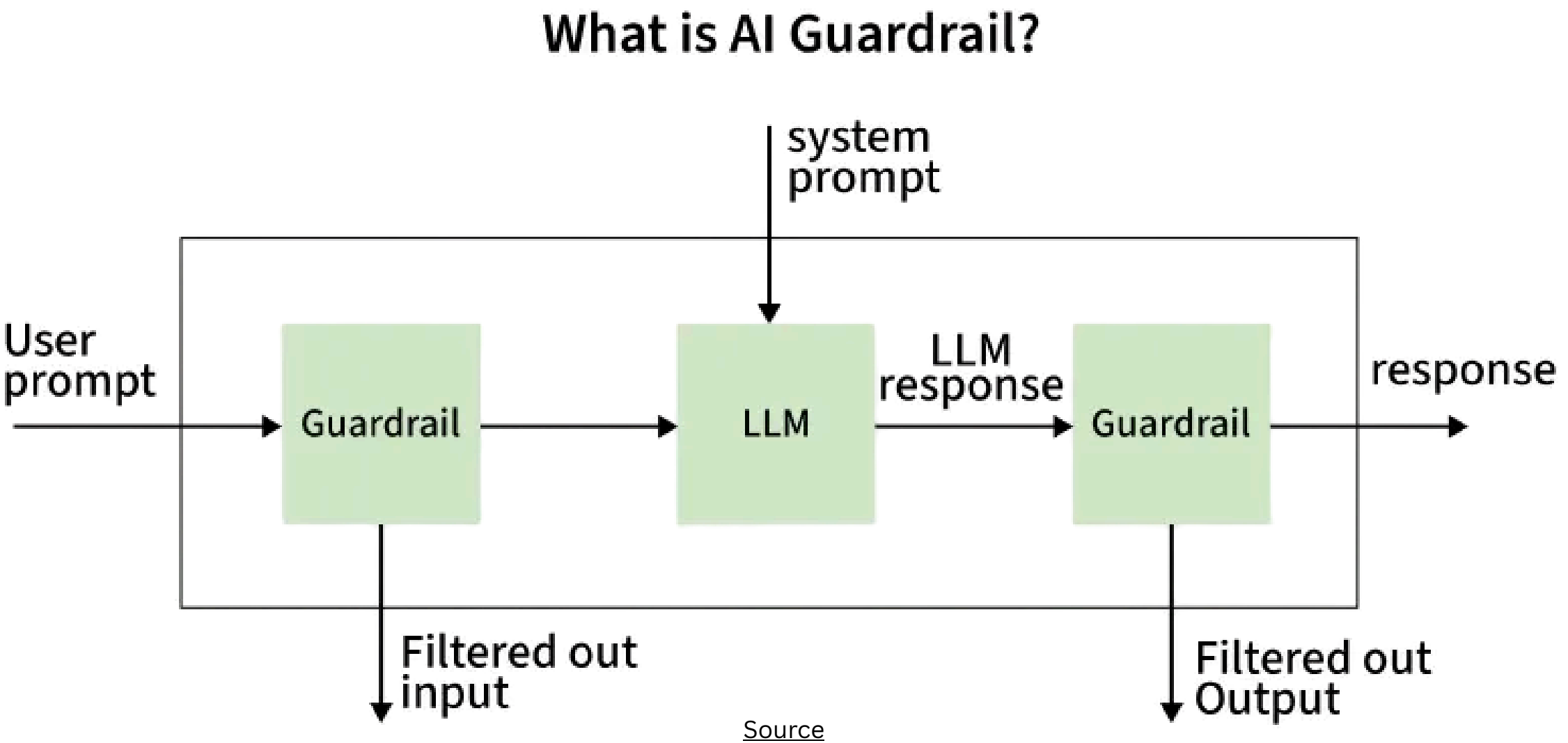
- Web search or browsing
- Database queries
- File parsers (PDF, Excel, DOCX)
- Retrieval from vector stores
- Python for calculations or data cleaning
- External APIs (Zapier, Twilio, Notion, etc.)

💡 Each tool makes the agent more capable but also increases risk.

Add Guardrails

Autonomy doesn't mean chaos. Design guardrails that allow flexibility without risking failure.

Key constraints:



Source

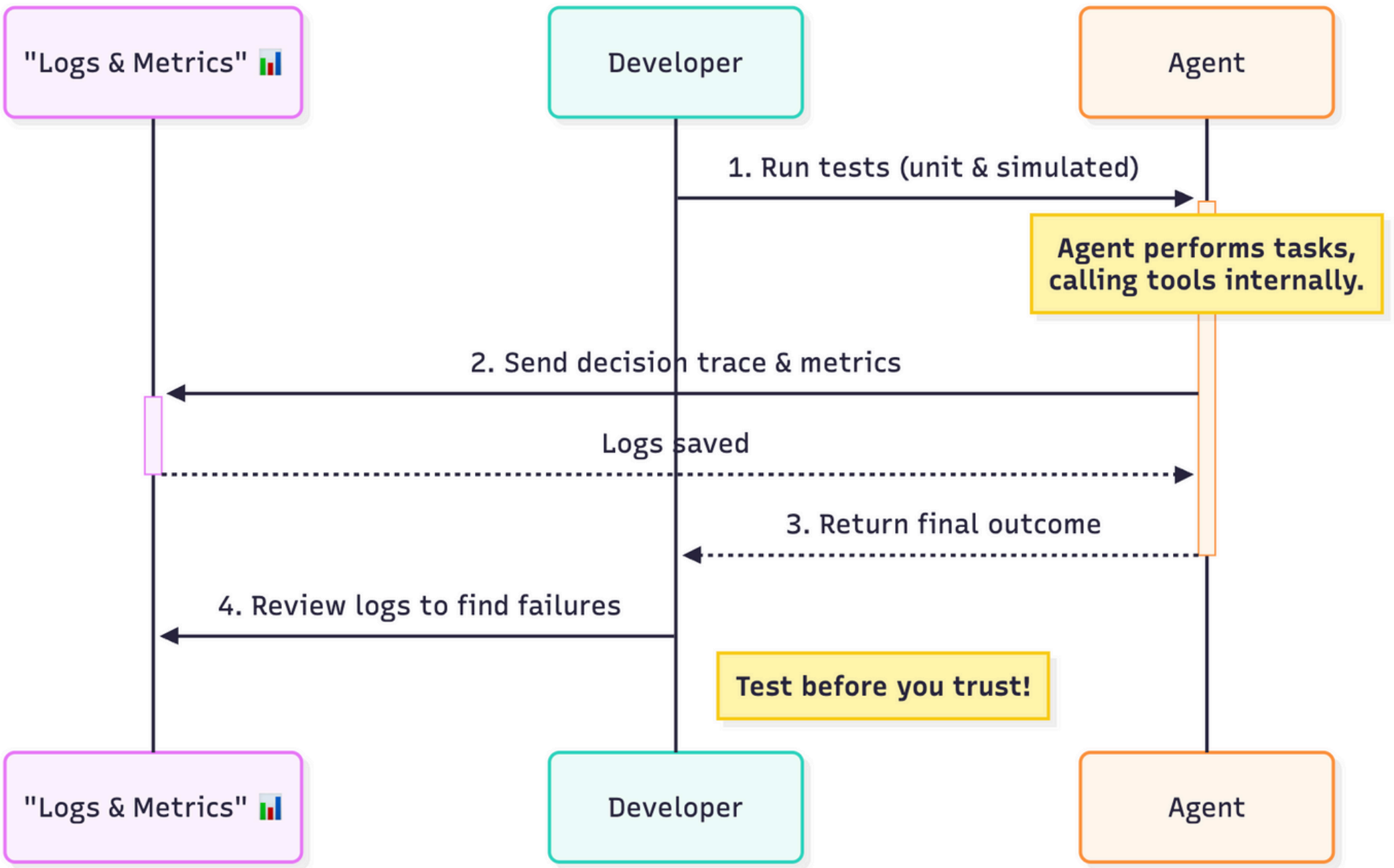
- **Timeouts** → prevent infinite loops
- **Cost limits** → control API usage
- **Human-in-the-loop** → for sensitive decisions
- **Output validators** → ensure correct formats and content
- **Policy filters** → for compliance, ethics, and bias

Autonomy thrives within boundaries. Design for that balance.

Test Your Agent

Most failed agents weren't dumb. They were just untested.

To design reliable systems, build:



- **Unit tests** → individual tool calls
- **Simulated runs** → end-to-end workflows
- **Metrics** → task success, retries, hallucination rate
- **Logs** → trace agent decisions for debugging

Test before you trust. Because your agent will only be as smart as your failure modes allow.

Bhavishya Pandit

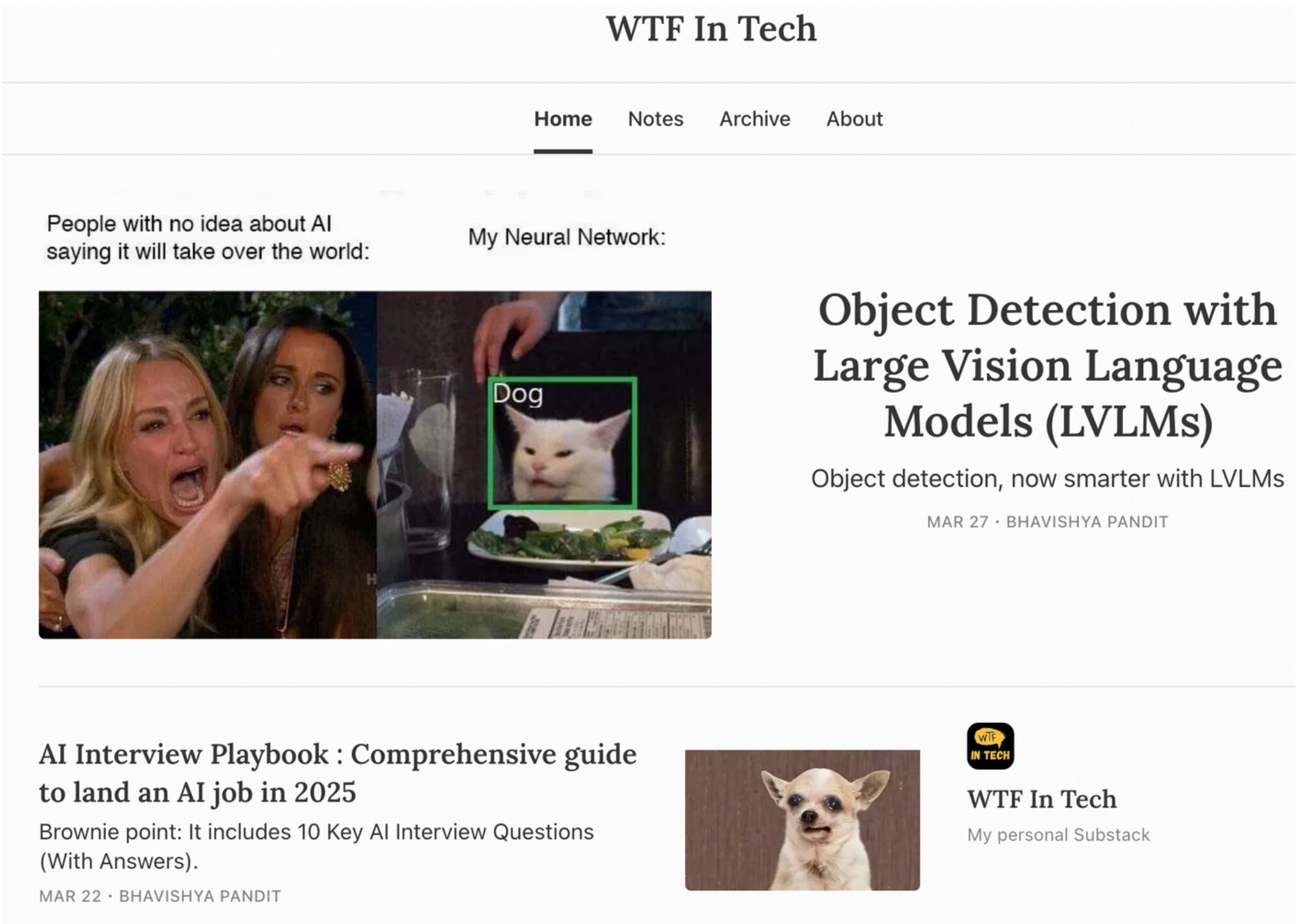
Liked it? Stay Ahead with Our Tech Newsletter! 🚀

👉 Join 1k+ leaders and professionals to stay ahead in GenAI!

🔗 https://bhavishyapandit9.substack.com/

Join our newsletter for:

- Step-by-step guides to mastering complex topics
- Industry trends & innovations delivered straight to your inbox
- Actionable tips to enhance your skills and stay competitive
- Insights on cutting-edge AI & software development



💡 Whether you're a developer, researcher, or tech enthusiast, this newsletter is your shortcut to staying informed and ahead of the curve.

Bhavishya Pandit

Follow to stay updated on Generative AI

LIKE

COMMENT

REPOST