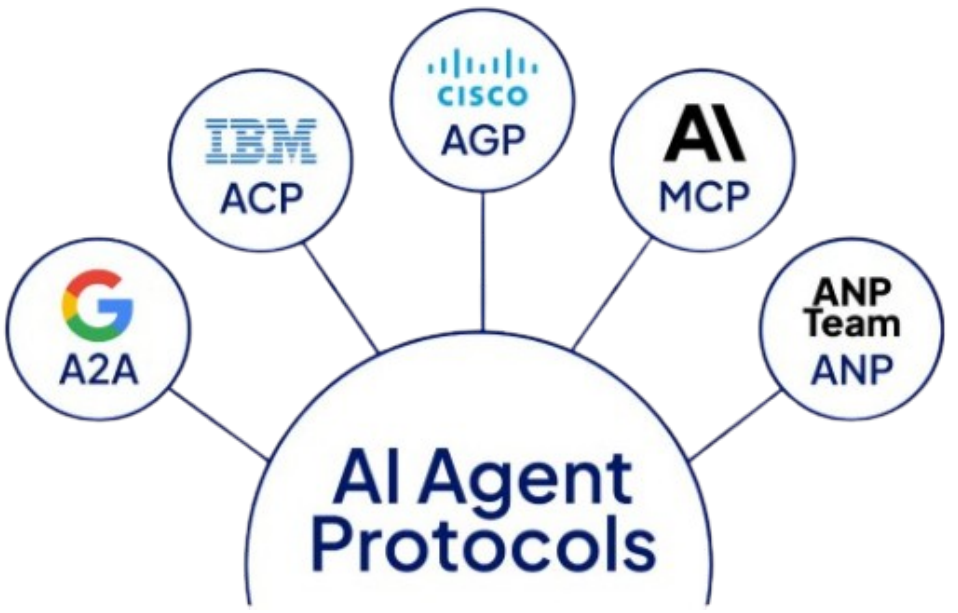
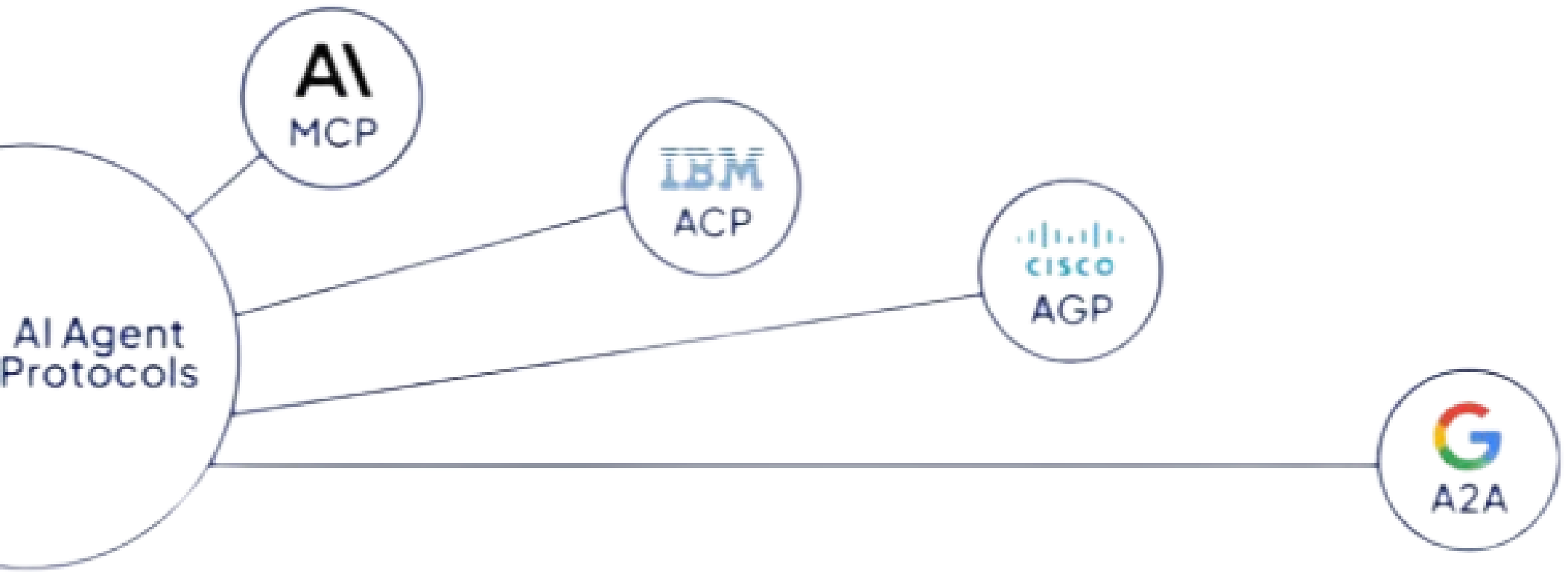
DroneX AI



Overview of AI Agent Protocols

Karn Singh



Intro

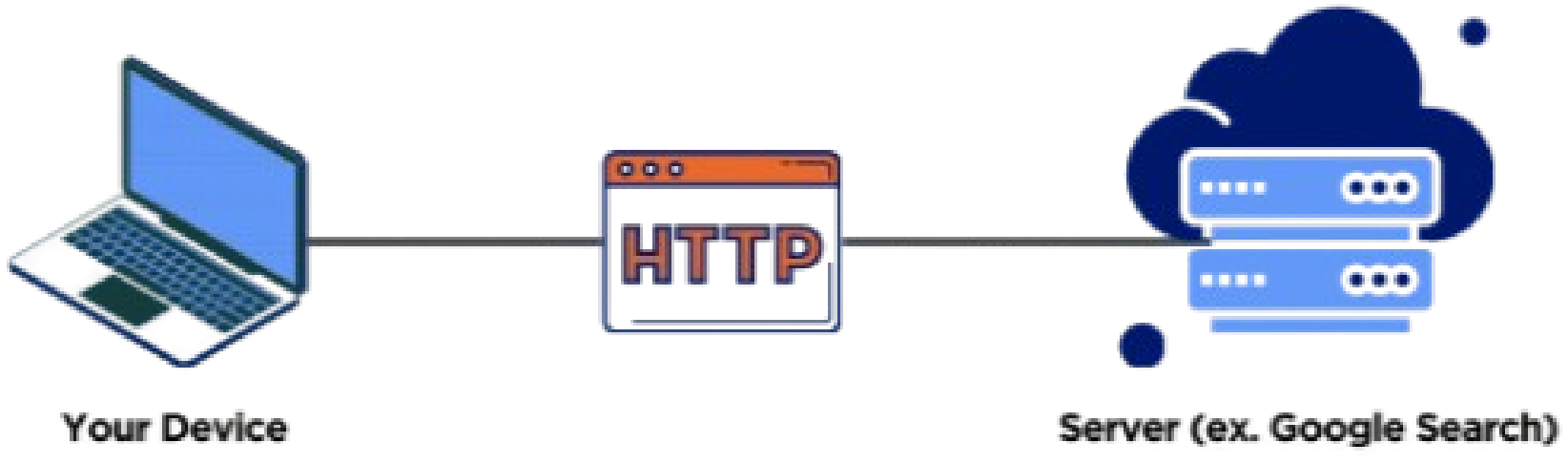
AI Agent Protocols Are Shaping the Future of Enterprise Workflows

In 2025, leading tech companies are investing heavily in AI Agent Protocols to streamline and automate operations. In this overview, we'll explore some of the most widely adopted protocols and examine their unique architectures and capabilities—helping you choose the right fit for your specific workflow needs.

What Are AI Agent Protocols?

AI Agent Protocols act as a **universal communication standard** that enables **different AI agents** to interact and collaborate within a multi-agent system — even if they are built using different technologies or architectures.

"Think of AI Agent Protocols as the HTTP of AI agents — a shared language for coordination."



Why Are AI Agent Protocols Important?

The Problem Before:

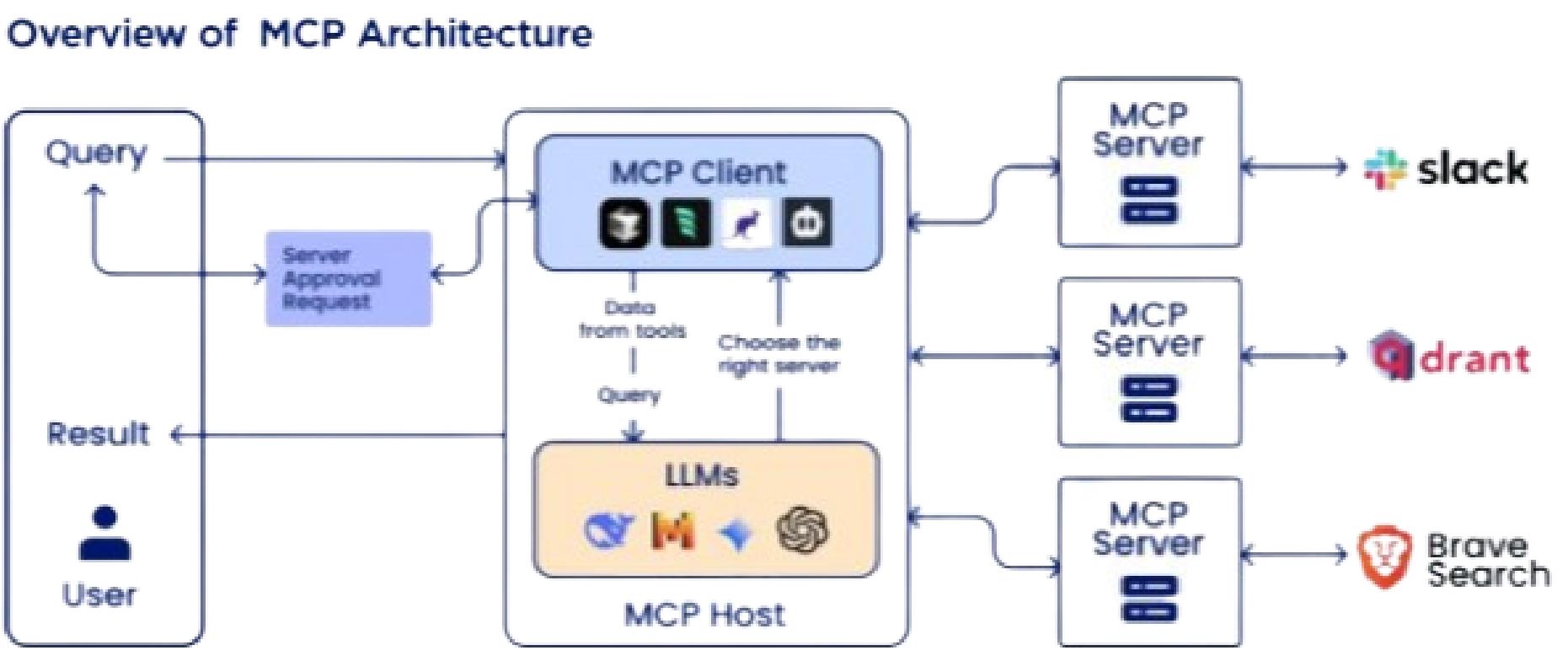
- Traditional agent communication was **fragmented** and **incompatible** — each system used its **own custom format** or framework.
- It was like having separate chatrooms where everyone spoke a different language.

The Solution Now:

- Modern **interoperability protocols** have unified how AI agents communicate.
- They enable **seamless coordination**, easier **tool integration**, and **standardized interactions** — making multi-agent systems more scalable and efficient.

MCP: Model Context Protocol

- **Developed by Anthropic**
- MCP is an **open, standardized communication protocol** that enables **two-way interaction** between **LLMs (clients)** and **external tools** or services (servers).
- Think of it like a universal USB-C port — connecting AI models to any external data or tool seamlessly.



MCP Workflow: Step-by-Step Breakdown

1. User Prompt:
A user initiates a request (query) through an MCP-compatible interface, asking to perform a specific task — such as fetching data, generating a summary, or automating a workflow.

2. MCP Client Activation:
The MCP client receives this prompt and forwards it to the underlying LLM (Large Language Model).

3. Query Dispatch:
The query is officially passed on from the client to the LLM for understanding and planning.

4. LLM Processing:
The LLM interprets the task and identifies which tool (external system) is best suited to complete it. It may also generate initial instructions or sub-queries for the tool.

5. Server Selection:
Based on the LLM's recommendation, the client selects the appropriate MCP server (e.g., Slack, Search API, database tool).

6. Optional User Approval:
If security is a concern or sensitive actions are involved, a human-in-the-loop (HITL) check is triggered. The client prompts the user to approve the tool or action.

7. Task Execution by MCP Server:
The selected MCP server processes the task using available data from both the user's query and its own database/toolset.

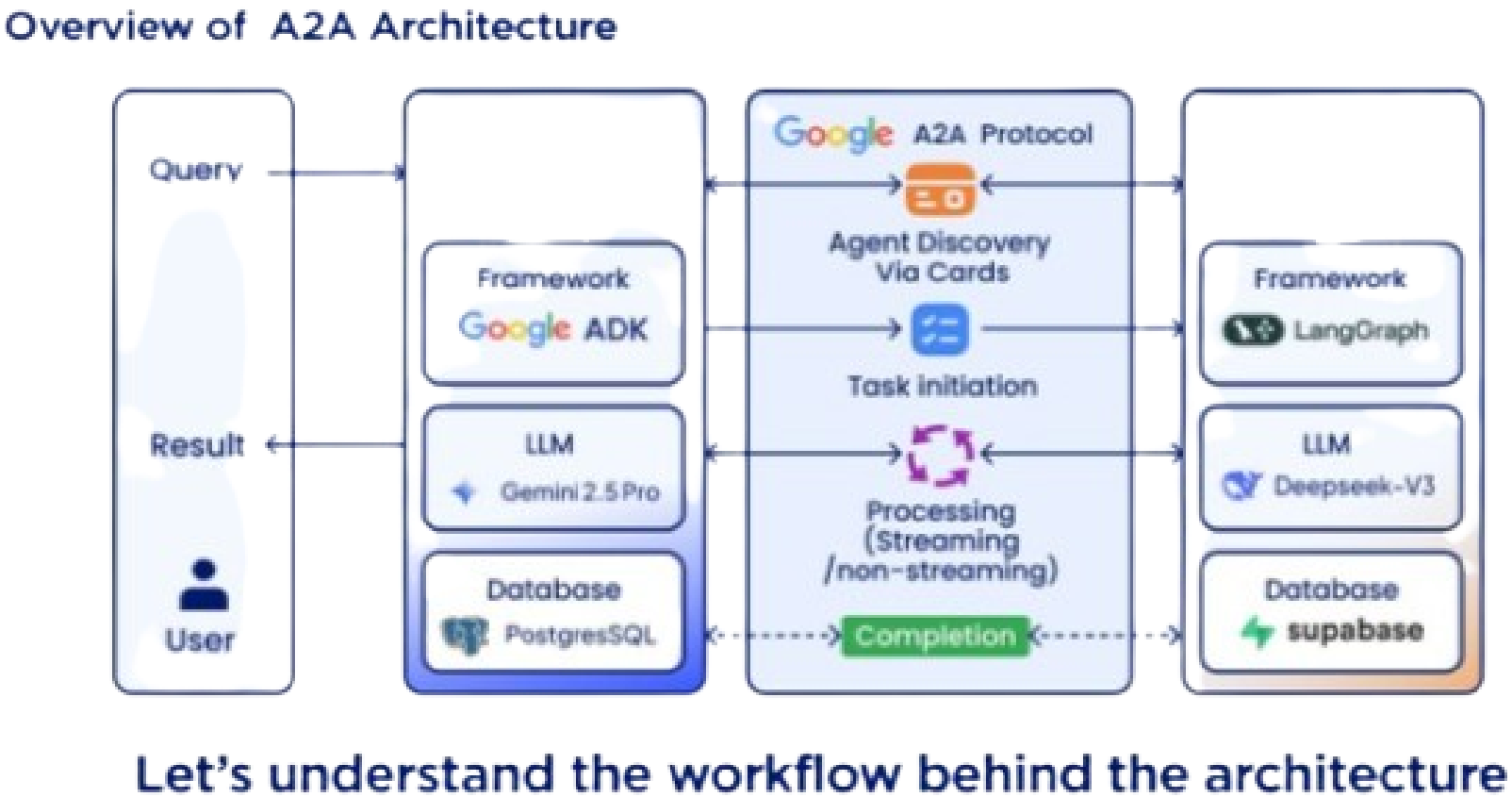
8. Result Delivery:
The outcome of the task is returned to the MCP client, which then formats and sends the final result back to the user.

A2A (Agent-to-Agent Protocol)

Developed by Google

- **A2A is an open, standardized protocol that enables seamless communication, collaboration, and task execution between AI agents—regardless of vendor, framework, or architecture.**

- It ensures interoperability between agents, facilitating discovery and interaction across diverse ecosystems. While A2A governs agent-to-agent interactions, the Model Context Protocol (MCP) by Anthropic focuses on allowing agents to access external tools and data sources.



A2A Protocol – Workflow Breakdown

1. User Query:
The workflow begins when a user submits a query to the Client Agent (AI Agent 1), asking for information or requesting a task to be executed.

2. Agent Discovery (Agent Card):
Each agent publishes a JSON-based Agent Card, listing its capabilities, skills, endpoint, and required credentials.
The Client Agent uses this metadata to discover and select the most suitable partner agent for the task.

3. Task Creation:
The Client Agent sends a task request. Every task is uniquely identified and follows a state-driven lifecycle (e.g., created → in-progress → completed).

4. Task Execution & Processing:
The Server Agent receives the task and begins processing. This may involve:
Streaming updates (via SSE) on task progress and intermediate artifacts
Synchronous processing for simpler tasks with immediate output

- Optional Interaction: If additional input is needed mid-task, the client can continue the dialogue using the same Task ID (via /tasks/send or /tasks/send/subscribe endpoints).

5. Task Completion:
Once the task finishes its processing pipeline, it reaches a final (terminal) state.

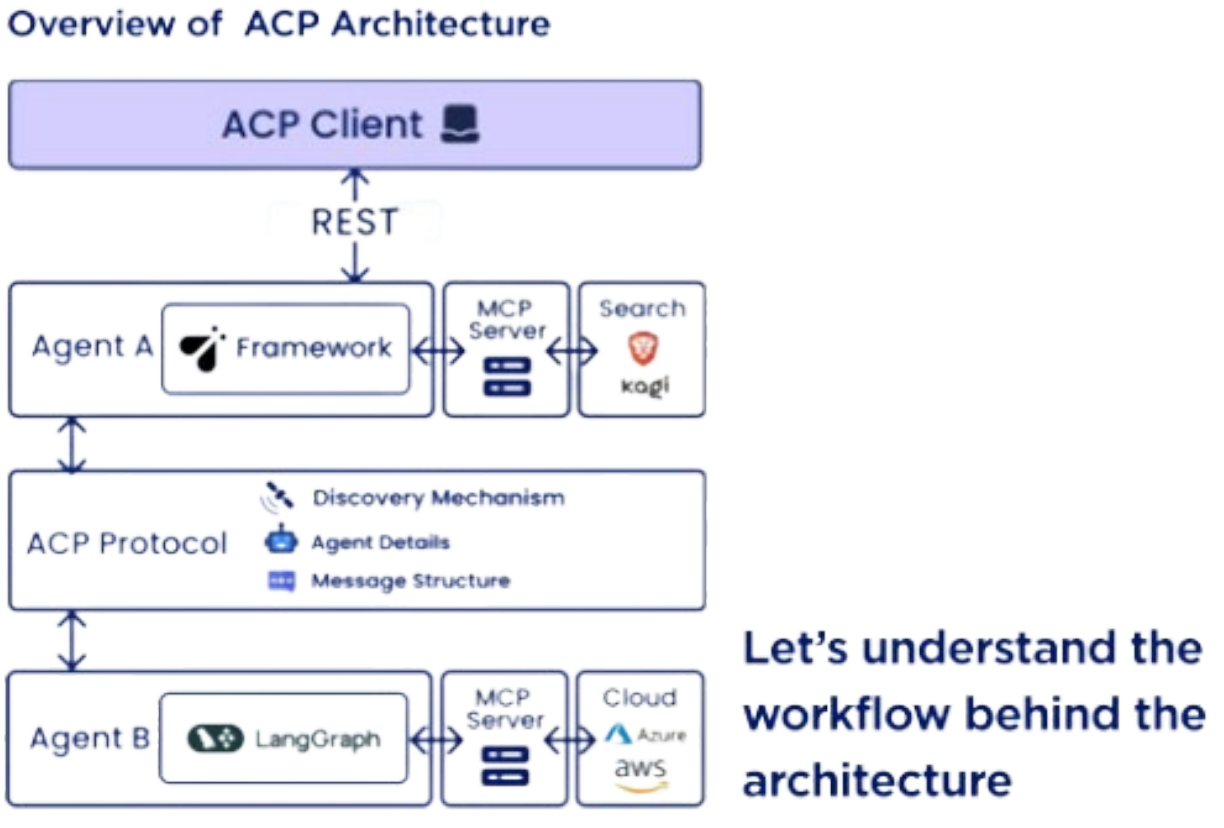
6. Result Delivery:
The completed result is returned to the user via the original client agent interface.

ACP (Agent Communication Protocol)

Developed by IBM

- ACP is an open, vendor-neutral communication protocol created by IBM for seamless interactions between AI agents. It was officially released in May 2025 under the Linux Foundation through the BeeAI project.
- ACP provides a RESTful, HTTP-based "wire format", allowing AI agents to:

1. Exchange messages
2. Assign and coordinate tasks
3. Operate both synchronously and asynchronously



ACP Workflow:

1. **Discover Agents:**
The ACP Client locates available agents (like Agent A & B) using REST calls to a shared registry or manifest.
It collects metadata such as endpoints, capabilities, and security configurations via REST APIs.

2. Identify Capabilities:
Agent A examines Agent B's published metadata to understand its capabilities and supported operations.

3. Acquire Authorization:
Agent A receives a signed token that specifies what actions it's allowed to perform on Agent B.

4. Invoke Task:
Using the token, Agent A sends a POST /run request to Agent B containing task instructions and permissions.

5. Execute Task:
Agent B carries out the request using the LangGraph framework and may access cloud services via MCP.

6. Stream Response:
Agent B delivers real-time updates or final outputs back to Agent A (or the ACP Client) through Server-Sent Events (SSE).

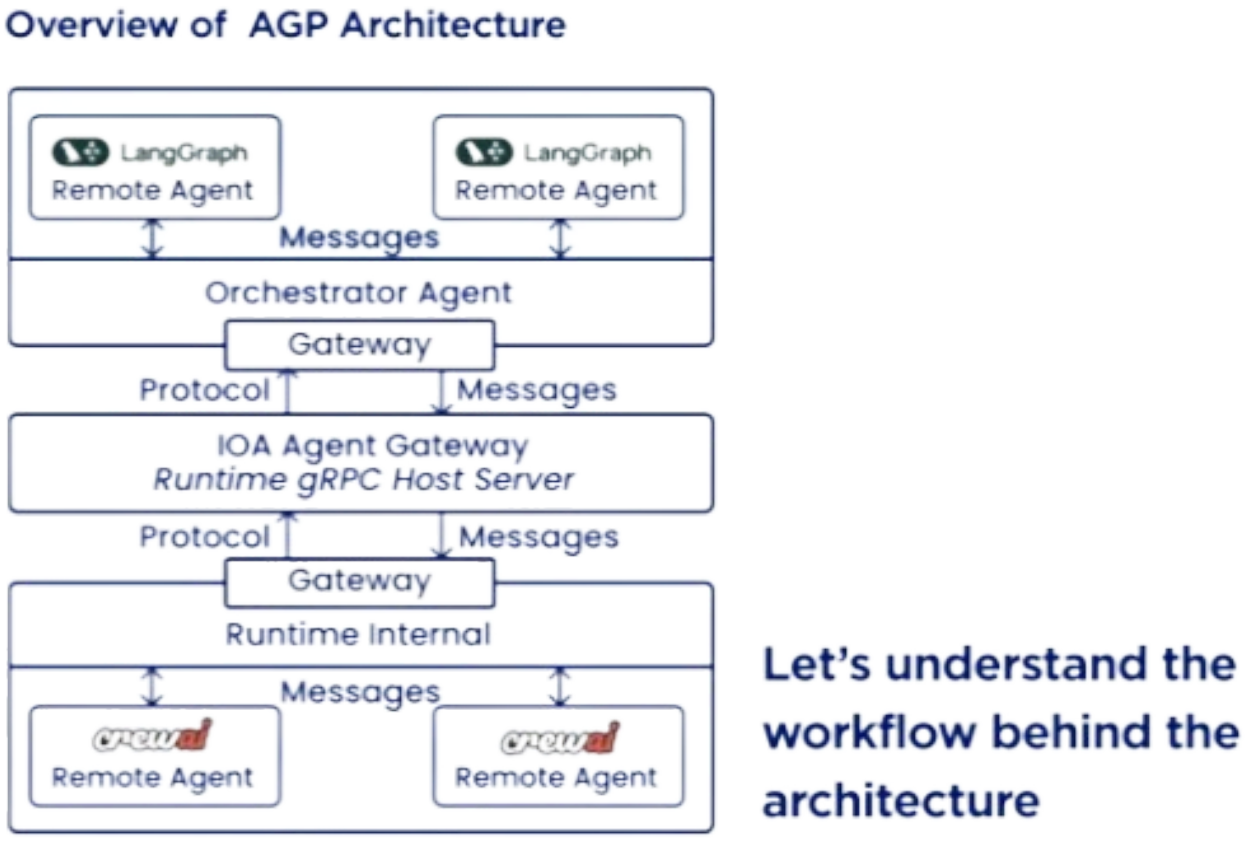
AGP (Agent Gateway Protocol)

Developed by Cisco

As part of the Outshift initiative, AGP aims to power a **secure Internet of Agents**.
AGP leverages **gRPC over HTTP/2** as a robust and standardized communication layer that supports:

- Real-time streaming
- Publish/Subscribe (pub/sub) messaging
- Request-Response patterns

This architecture is optimized for high-performance, scalable agent communication across distributed environments.



AGP Workflow:

1. Register Agents

Agents (built on LangGraph, CrewAI, etc.) connect securely to the IOA Gateway using gRPC, verifying identity and sharing their capabilities.

2. Configure Gateway

The gateway sets up control and data layers to handle authentication, routing rules, namespaces, and messaging policies.

3. Enable Messaging Patterns

Agents communicate via the gateway using supported patterns like:

- Request/Response
- Publish/Subscribe
- Fire-and-Forget
- Real-time Streaming (gRPC)

4. Route Messages

The gateway intelligently routes messages between agents and orchestrators, enforcing real-time policies and tenant isolation.

5. Enforce Security

All communication is encrypted (TLS over HTTP/2), authenticated, and authorized. Additional payload encryption (e.g., MLS) can be applied.

6. Monitor and Manage

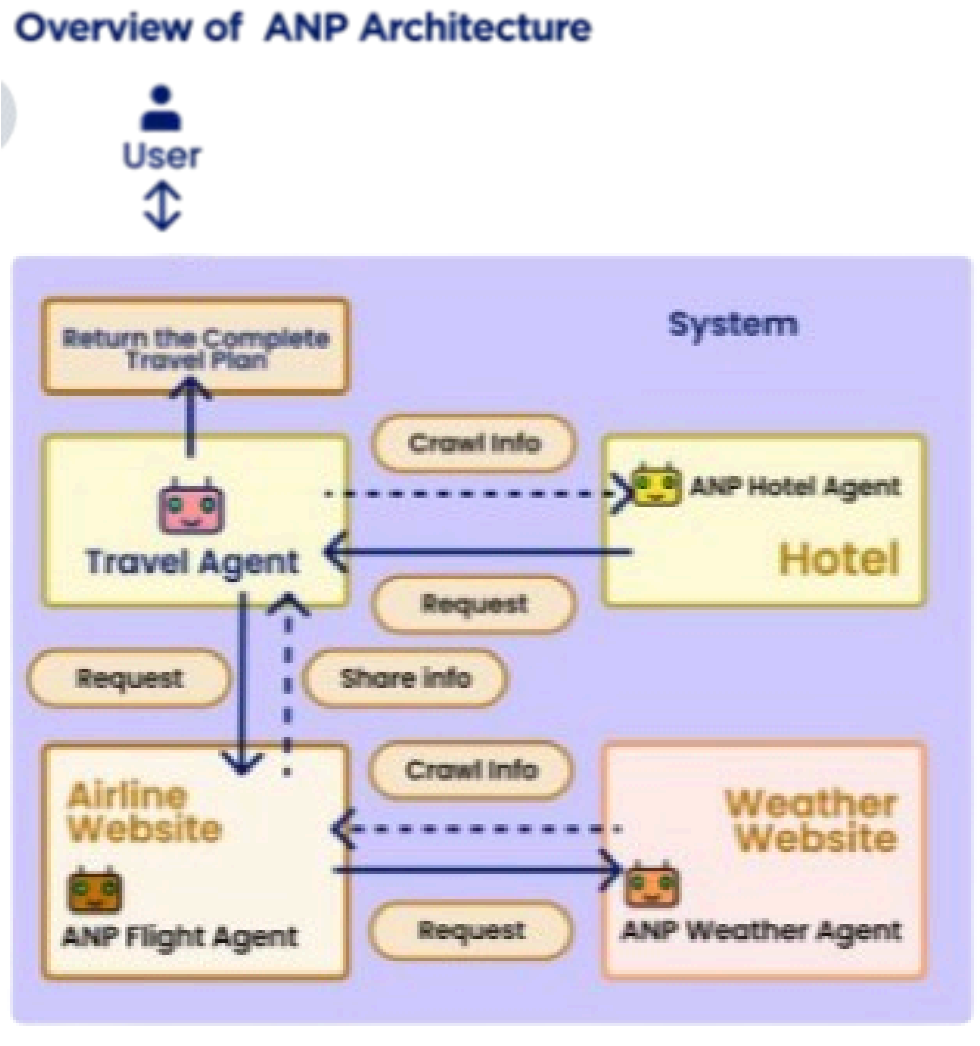
The gateway offers visibility into agent operations through telemetry, token management, and discovery tools—ensuring secure, observable agent lifecycles.

ANP (Agent Network Protocol)

Developed by the ANP Team,

ANP is an open-source protocol designed to support seamless interoperability across diverse AI agents.
It enables agents to work together using:

- JSON-LD for linked data representation
- W3C Decentralized Identifiers (DID) for secure agent identity
- A layered system for agent discovery, negotiation, and collaboration



Let's understand the workflow behind the architecture

ANP Workflow:

1. Discover Available Agents
The local agent contacts a discovery service (predefined endpoint) to retrieve a list of other agents. It uses JSON-LD metadata to identify who's available.

2. Fetch Agent Capabilities
It downloads each agent's description file—also in JSON-LD format—to understand their available features and services.

3. Start Interaction
The agent builds a request using standardized formats defined by the Application Protocol Layer.

4. Authenticate Securely
Using W3C Decentralized Identifiers (DID), the agent attaches cryptographic credentials to secure communications, ensuring end-to-end encryption.

5. Send the Request
The agent sends the request using the Meta-Protocol Layer, allowing for flexible protocol negotiation (even via natural language).

6. Process the Response
When the response arrives, the agent reads the structured data (e.g., JSON-LD) and extracts useful information to complete the task

AGORA Protocol

Developed at the University of Oxford

Agora is a scalable communication protocol designed for LLM-based agents.
It enables agents to negotiate and align on communication protocols using **natural language and JSON**, allowing seamless interaction across various services.

Agora excels at:

- Converting natural language into machine-readable protocol instructions
- Generating standardized, agent-compatible protocols
- Supporting personalized, user-driven workflows

Agora Workflow:

1. Understand User Intent
The agent uses LLM-based capabilities to interpret natural language input and extract structured intent from the user's query.

2. Discover Agent Capabilities
It retrieves JSON-formatted metadata about available agents, including supported protocols and features.

3. Generate Protocols
Based on the user's intent, the agent creates tailored, standardized communication protocols for each required service.

4. Secure the Interaction
Authentication tokens are embedded into the request to ensure secure and trusted communication with each target agent.

5. Distribute Protocols
Agora routes the generated protocols to appropriate service agents for execution via its distribution mechanism.

6. Combine Responses
The agent gathers and integrates the responses from all service agents to deliver a complete and meaningful result to the user.

Best Use Case per Protocol

MCP – Model Context Protocol (AI):

Ideal for scenarios where AI assistants must access live, external context (e.g., weather, market data, sensor input) in a secure, standardized, and real-time manner to guide their decisions.

A2A – Agent-to-Agent Protocol (Google):

Perfect for coordinating multiple AI agents within a modular enterprise system to solve multi-step, real-world workflows via structured, asynchronous communication.

ACP – Agent Communication Protocol (IBM):

Designed for complex task management, ACP excels in inter-agent collaboration, legacy system integration, real-time data exchange, request cancellation, and robust task persistence.

Best Use-Case for Each Protocol

- **AGP – Agent Gateway Protocol (Cisco):**

Ideal for building event-driven, cross-network agent workflows. AGP is best suited for enabling agents to seamlessly coordinate across different network environments and infrastructures.

- **ANP – Agent Network Protocol (ANP Team):**

Optimized for cross-domain agent interoperability, ANP allows agents from various vendors and architectures to collaborate effectively in complex, distributed multi-agent systems.

- **AGORA – University of Oxford:**

Designed for transforming natural language into agent protocols, AGORA empowers users to define workflows and agent behavior using simple, conversational commands—ideal for user-centric interaction models.

Conclusion

With the growing relevance of Multi-Agent architectures across various AI products and services, establishing reliable communication bridges between agents remains a critical part of an effective workflow.

To explore further,
refer to the linked documentation and repositories provided with this post.

DroneX AI

WAS THIS POST USEFUL?

FOLLOW FOR MORE!

REPOST