Architecture

Overview of the core architecture behind the Agent Communication Protocol

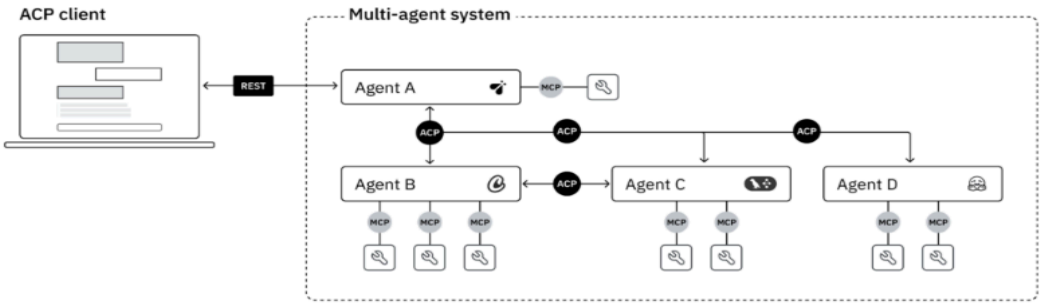
The Agent Communication Protocol (ACP) provides a standardized interface for agent communication, enabling seamless interaction between clients and servers, as well as between multiple agents in complex systems. This page explains the architectural patterns that ACP supports, from simple deployments to sophisticated multi-agent ecosystems.

Core Components

An ACP client can be used by an ACP agent, application, or other service that makes requests to an ACP server using the ACP protocol.

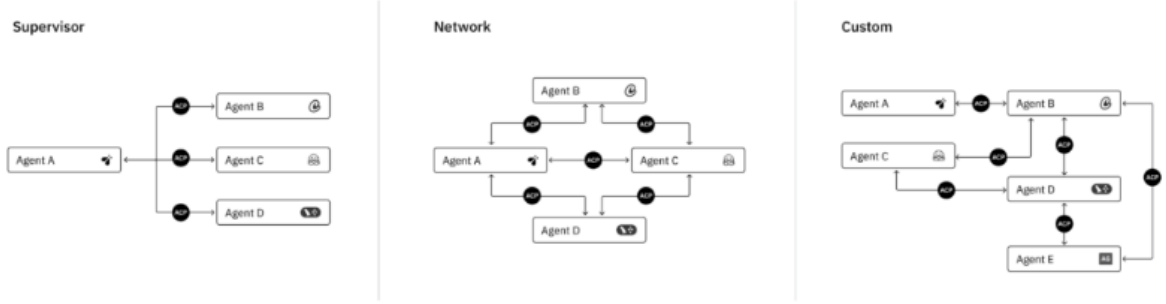
An ACP server can host one or more ACP agents that executes requests and returns results to the client using the ACP protocol. The purpose of the ACP server is to expose agents through a REST interface.

ACP servers and clients can be deployed in any combination—a single process can act as both a server (handling incoming requests) and a client (making outbound requests to other agents).



Example of an ACP client and ACP Agents of different frameworks communicating. Image used with permission.

Why ACP?



As the amount of AI Agents “in the wild” increases, so does the amount of complexity in navigating how to get the best outcome from each independent technology for your use case (without having to be constrained to a particular vendor). Each framework, platform, and toolkit out there offers unique advantages, but integrating them all together into one agent system is challenging.

Today, most agent systems operate in silos. They’re built on incompatible frameworks, expose custom APIs, and lack a shared protocol for communication. Connecting them requires fragile and non repeatable integrations that are expensive to build.

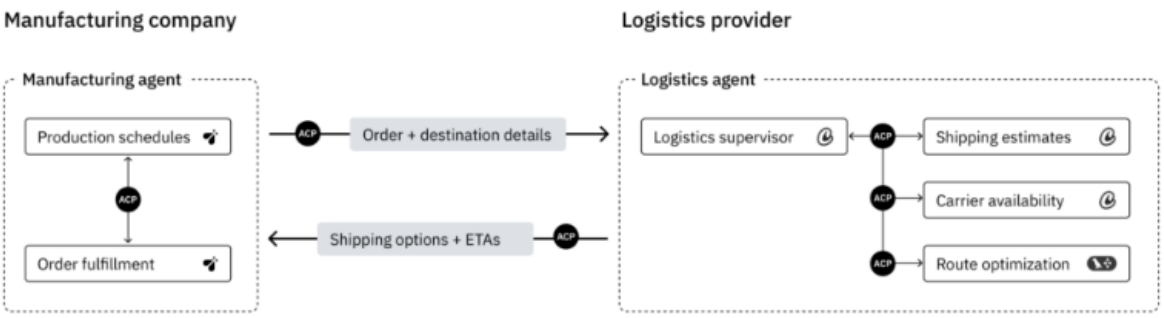
ACP represents a fundamental shift: from a fragmented, *ad hoc* ecosystem to an interconnected network of agents—each able to discover, understand, and collaborate with others, regardless of who built them or what stack they run on. With ACP, developers can harness the collective intelligence of diverse agents to build more powerful workflows than a single system can achieve alone.

Current Challenges:
Despite rapid growth in agent capabilities, real-world integration

remains a major bottleneck. Without a shared communication protocol, organizations face several recurring challenges:

- Framework Diversity: Organizations typically run hundreds or thousands of agents built using different frameworks like LangChain, CrewAI, AutoGen, or custom stacks.
- Custom Integration: Without a standard protocol, developers must write custom connectors for every agent interaction.
- Exponential Development: With n agents, you potentially need n(n-1)/2 different integration points (which makes large-scale agent ecosystems difficult to maintain).
- Cross-Organization Considerations: Different security models, authentication systems, and data formats complicate integration across companies.

Example:



A manufacturing company that uses an AI agent to manage production schedules and order fulfillment based on internal inventory and customer demand.

A logistics provider that runs an agent to offer real-time shipping estimates, carrier availability, and route optimization.

Now imagine the manufacturer's system needs to estimate delivery timelines for a large, custom equipment order to inform a customer quote.

Without ACP: This would require building a bespoke integration between the manufacturer's planning software and the logistics provider's APIs. This means handling authentication, data format mismatches, and service availability manually. These integrations are expensive, brittle, and hard to scale as more partners join.

With ACP: Each organization wraps its agent with an ACP interface. The manufacturing agent sends order and destination details to the logistics agent, which responds with real-time shipping options and ETAs. Both systems collaborate without exposing internals or writing custom integrations. New logistics partners can plug in simply by implementing ACP.