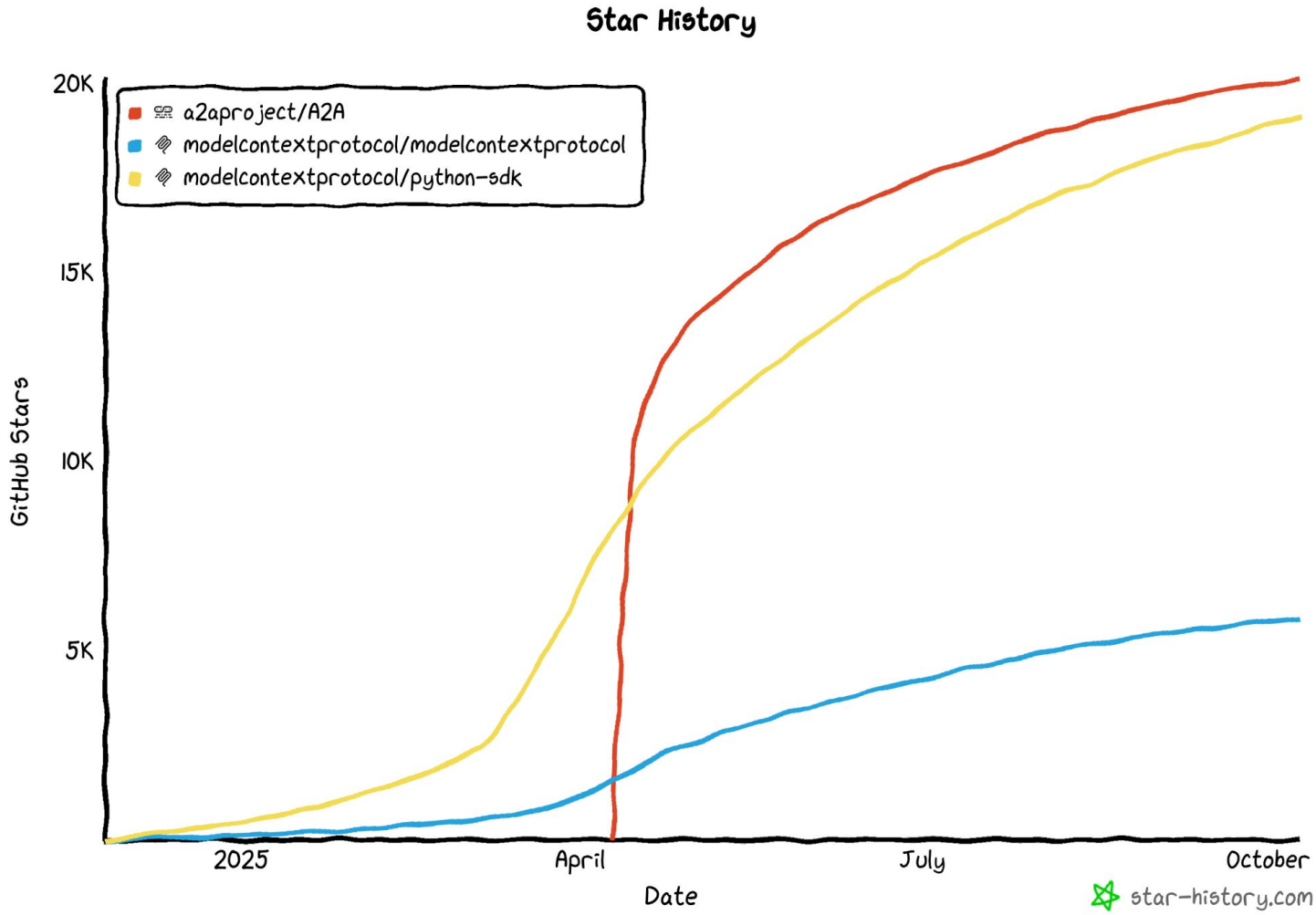
Agentic System: A2A

A2A: Agenda & Overview

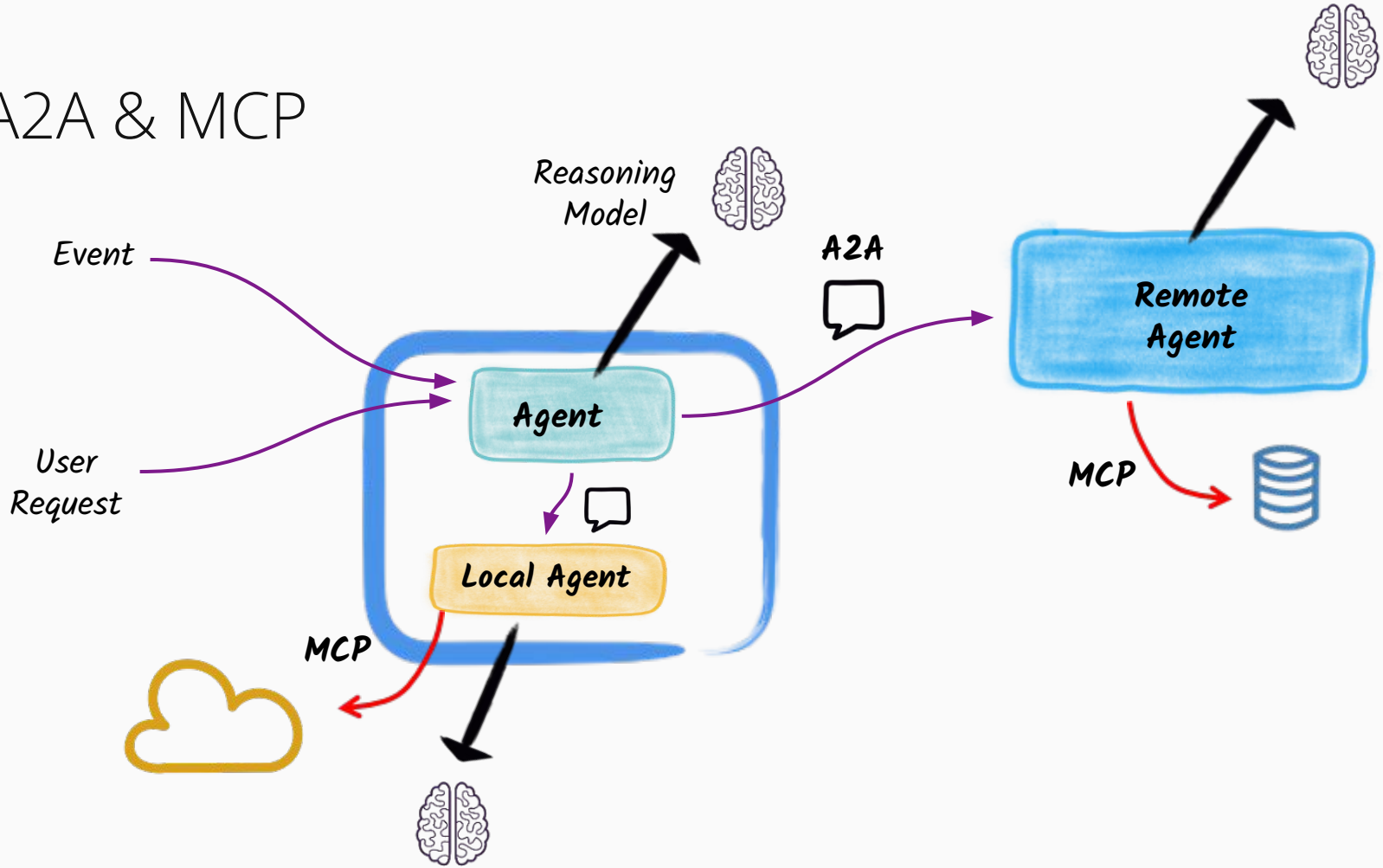
- Introduction to A2A and its significance
- Understand the shift from agents to tools
- Deep dive into A2A Java SDK transports
- Practical implementation with code examples
- Explore advanced A2A security and extensions

A2A: Internet for AI Agents

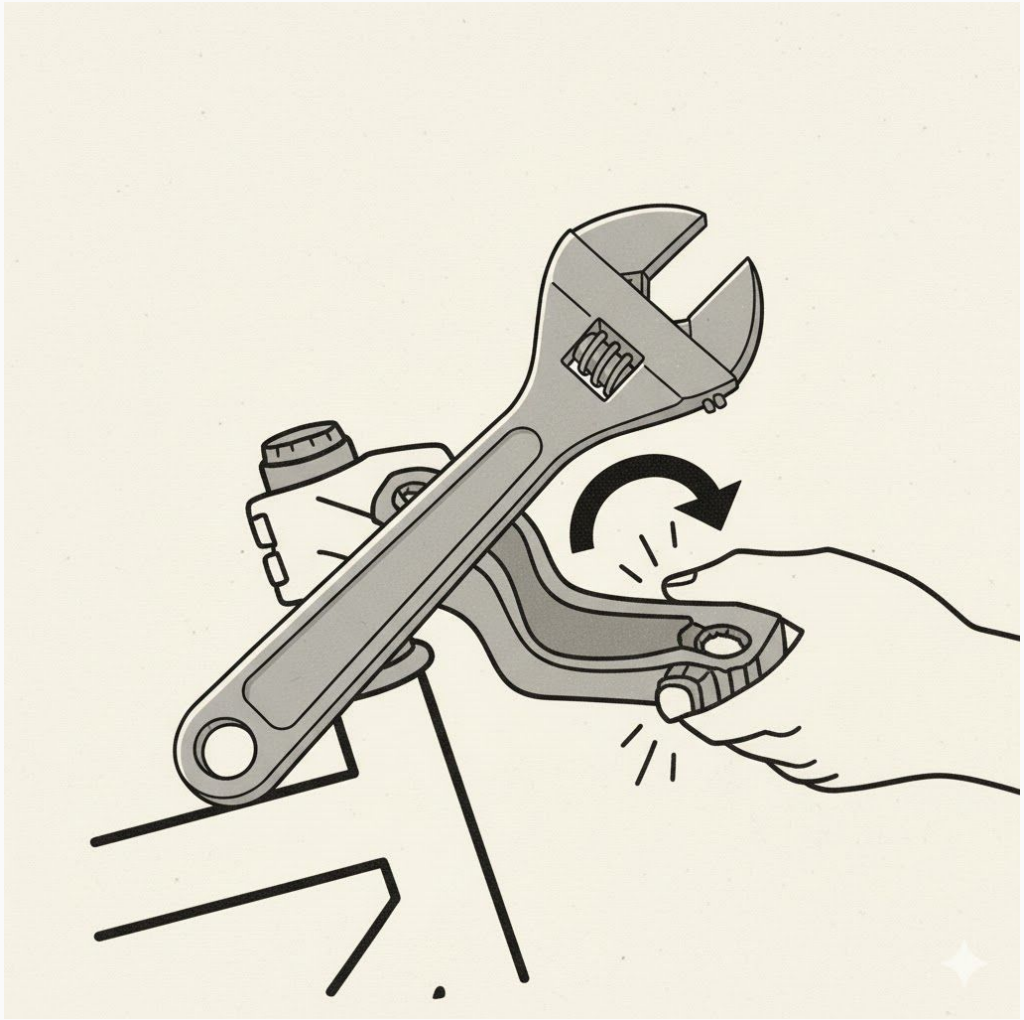
- Open standard for AI agent communication
- Developed by Google, now Linux Foundation
- Vendor-neutral and community-driven initiative
- Fosters an interoperable 'Internet of Agents'
- RedHat/IBM docated and maintain Java SDK and protocol TCK


Star History
20K
15K
10K
5K
GitHub Stars
a2aproject/A2A
modelcontextprotocol/modelcontextprotocol
modelcontextprotocol/python-sdk
2025
April
July
October
Date
star-history.com

A2A & MCP


Reasoning Model
Event
A2A
Remote Agent
Agent
User Request
MCP
Local Agent
MCP

A wrench is a tool



And you can use the wrench tool for easy task, especially with good information

A plumber is an agent, using the wrench tool

Agents ≠ Tools

Even with the right tool you may fail…
Because you don’t have the skills

Even if you give me a wrench…I need a plumber

Agents ≠ Tools

And you should not use it as a tool

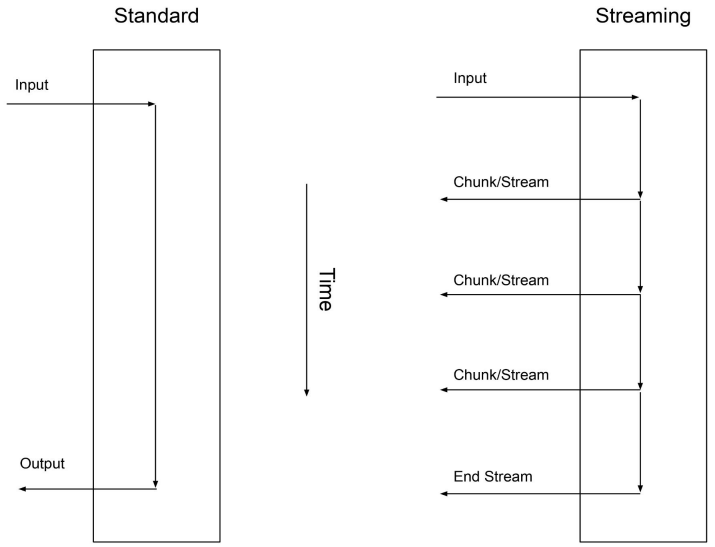
Agents ≠ Tools

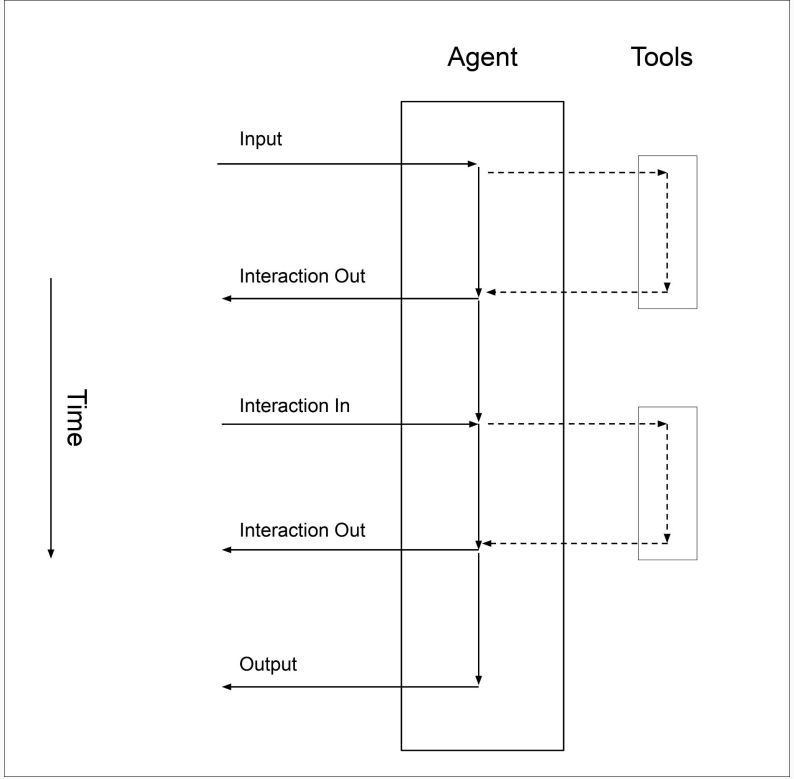
A "Tool" is...

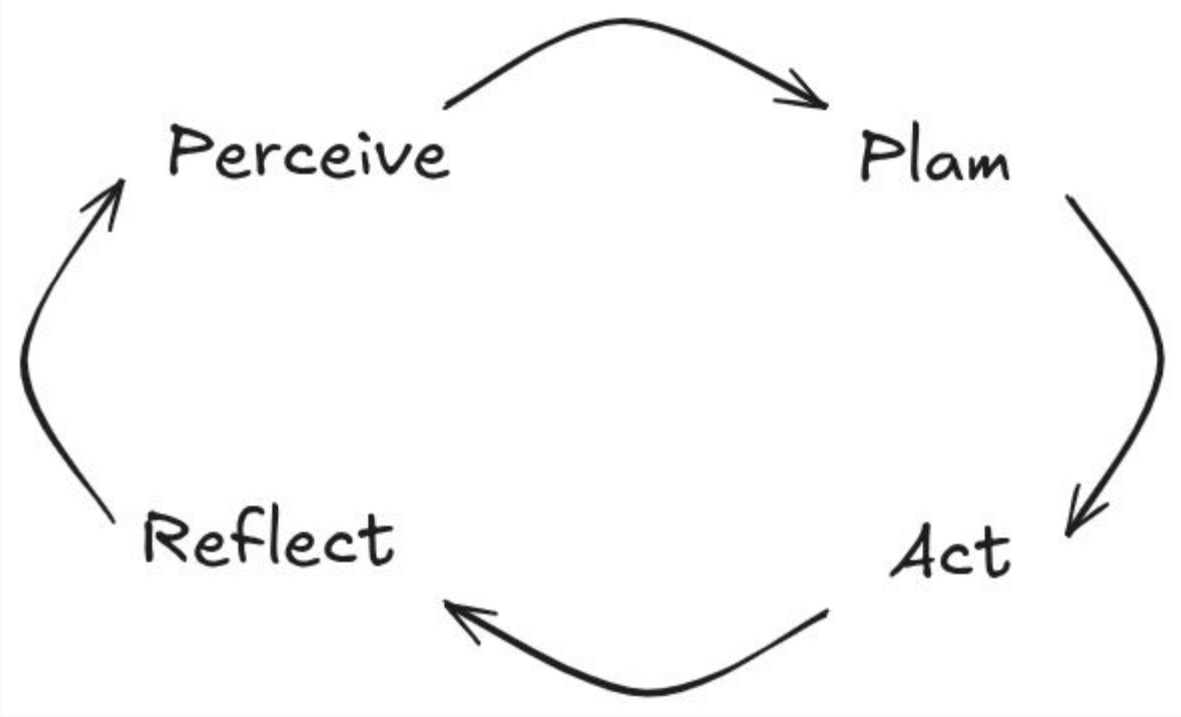
- Structured & Time-Boxed.
- Like a traditional API call: function(input) → output.
- Predictable and stateless.
- This is how we've built software for decades.

An "Agent" engages in...

- Multi-turn, unbounded interactions.
- Stateful conversations where history matters.
- Perceive-plan-act-reflect


Standard
Input
Output
Time
Streaming
Input
Chunk/Stream
Chunk/Stream
Chunk/Stream
End Stream


Agent
Tools
Input
Interaction Out
Interaction In
Interaction Out
Output
Time


Perceive
Plam
Act
Reflect

A2A in Agent Ecosystem

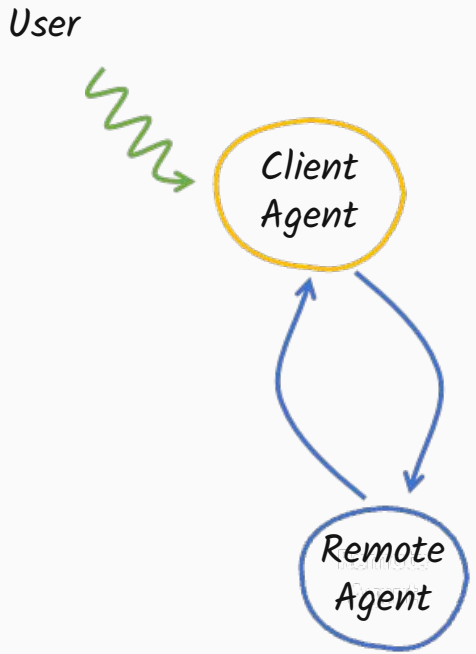
- A2A defines agent-to-agent communication for collaboration
- It complements the Model Context Protocol (MCP)
- MCP focuses on agent-to-tool communication standards

Core A2A Actors

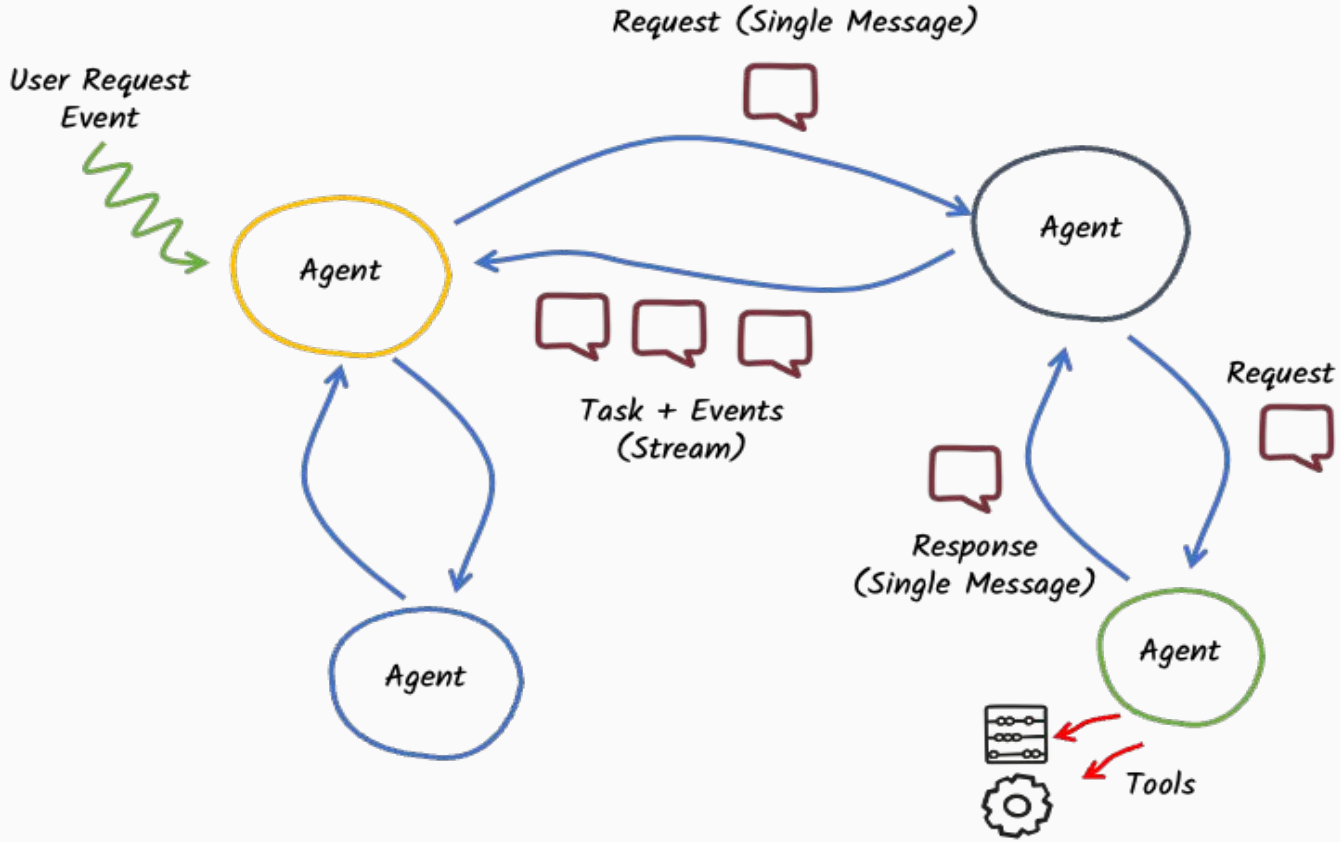
User: The initiator of a request or goal (human or automated service).

Client Agent (A2A Client): An application or agent that acts for the user and starts the communication.

Remote Agent (A2A Server): An agent that receives and processes tasks from the client, functioning as a "black box" system.



A2A (Agent To Agent)

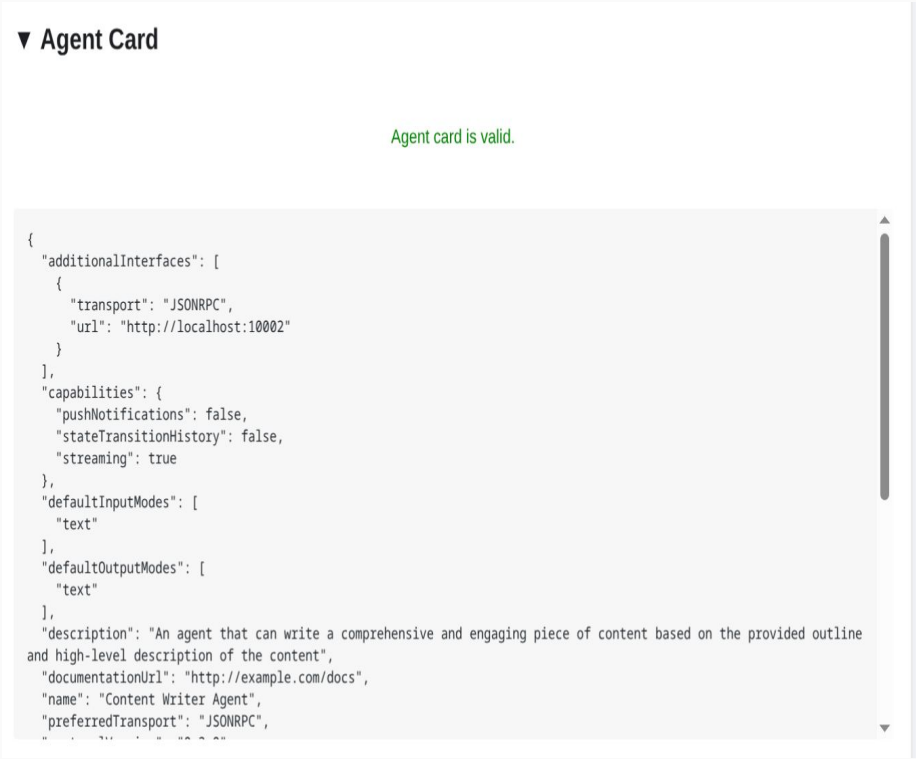


Key A2A Protocol Elements

Agent Card: A JSON profile detailing an agent's identity, capabilities, and endpoint. Its purpose is agent discovery and interaction.
Message: A single turn of communication (e.g., a question or answer) used to convey instructions or updates.
Task: A stateful unit of work for tracking long-running operations and enabling collaboration.
Part: The basic content container (text, file, etc.) used within messages and artifacts.
Artifact: A tangible output from a task, such as a document or image, representing the concrete result.

The Agent Card

```java
@Produces
@PublicAgentCard
public AgentCard agentCard() {
  return new AgentCard.Builder()
      .name(name:"Content Writer Agent")
      .description(
          "An agent that can write a "
              + "comprehensive and engaging piece of content "
              + "based on the provided outline and high-level "
              + "description of the content")
      .url("http://localhost:" + httpPort)
      .version(version:"1.0.0")
      .documentationUrl(documentationUrl:"http://example.com/docs")
      .capabilities(
          new AgentCapabilities.Builder()
              .streaming(streaming:true)
              .pushNotifications(pushNotifications:false)
              .stateTransitionHistory(stateTransitionHistory:false)
              .build())
      .defaultInputModes(Collections.singletonList("text"))
      .defaultOutputModes(Collections.singletonList("text"))
      .skills(
          Collections.singletonList(
              new AgentSkill.Builder()
                  .id(id:"writer")
                  .name(name:"Writes content using an outline")
                  .description(
                      "Writes content using a given "
                          + "outline and high-level description of "
                          + "the content")
                  .tags(List.of("writer"))
                  .examples(
                      List.of(
                          "Write a short, upbeat, and "
                              + "encouraging twitter post about learning "
                              + "Java. Base your writing on the given "
                              + "outline."))
                  .build()))
      .protocolVersion(protocolVersion:"0.3.0")
      .build();
}
```



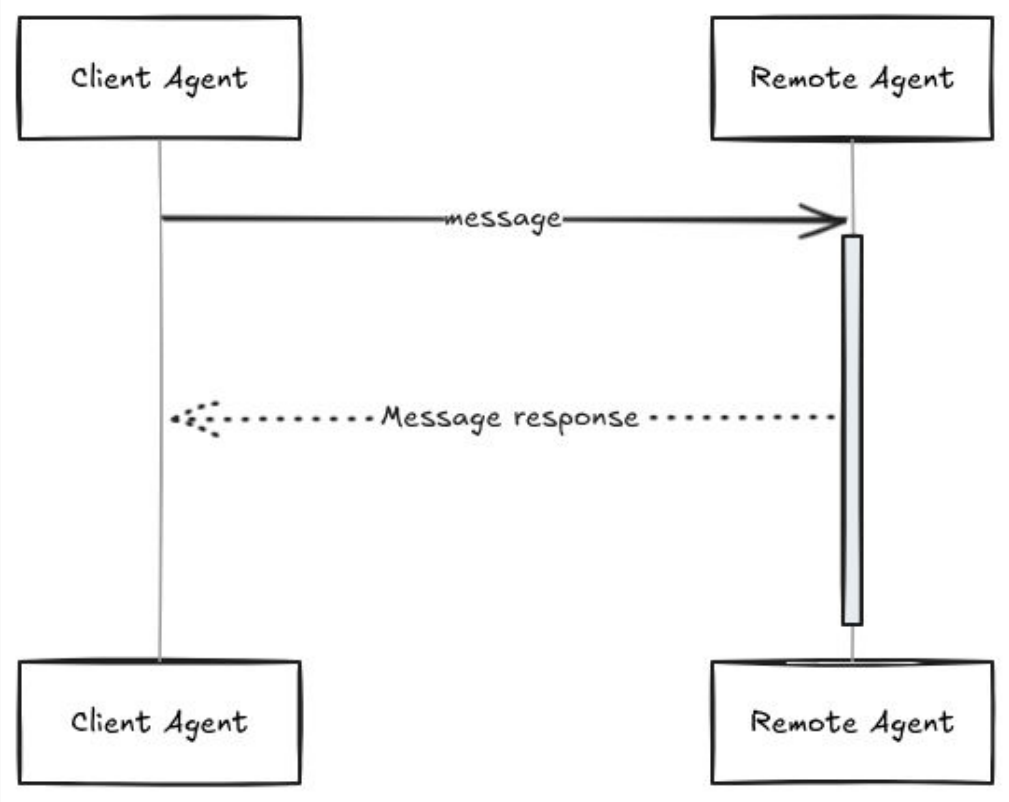
Agent card discover mechanism

host/.well-known/agent.card.json

Registry (no official one available)

Client configuration

Synchronous Message exchange



Task

The Primary Container

A **Task** represents the entire unit of work or a specific job assigned to an agent. Every interaction, from a simple request to a complex multi-step workflow, is encapsulated within a `Task`.

Creation & Lifecycle

A new `Task` is automatically created when a client sends the first message using the `message/send` method. It progresses through various states until it's completed, fails, or is cancelled.

Designed for Asynchronicity

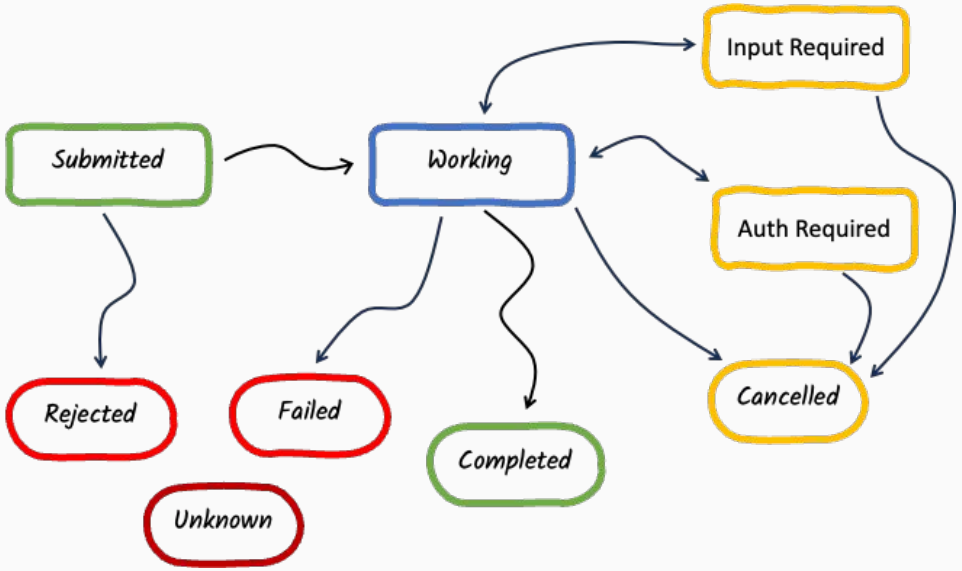
Tasks are perfect for long-running jobs. A client can start a `Task`, get a unique `task_id`, and then check its progress later using `tasks/get` or receive real-time updates with `message/stream`.

Task state

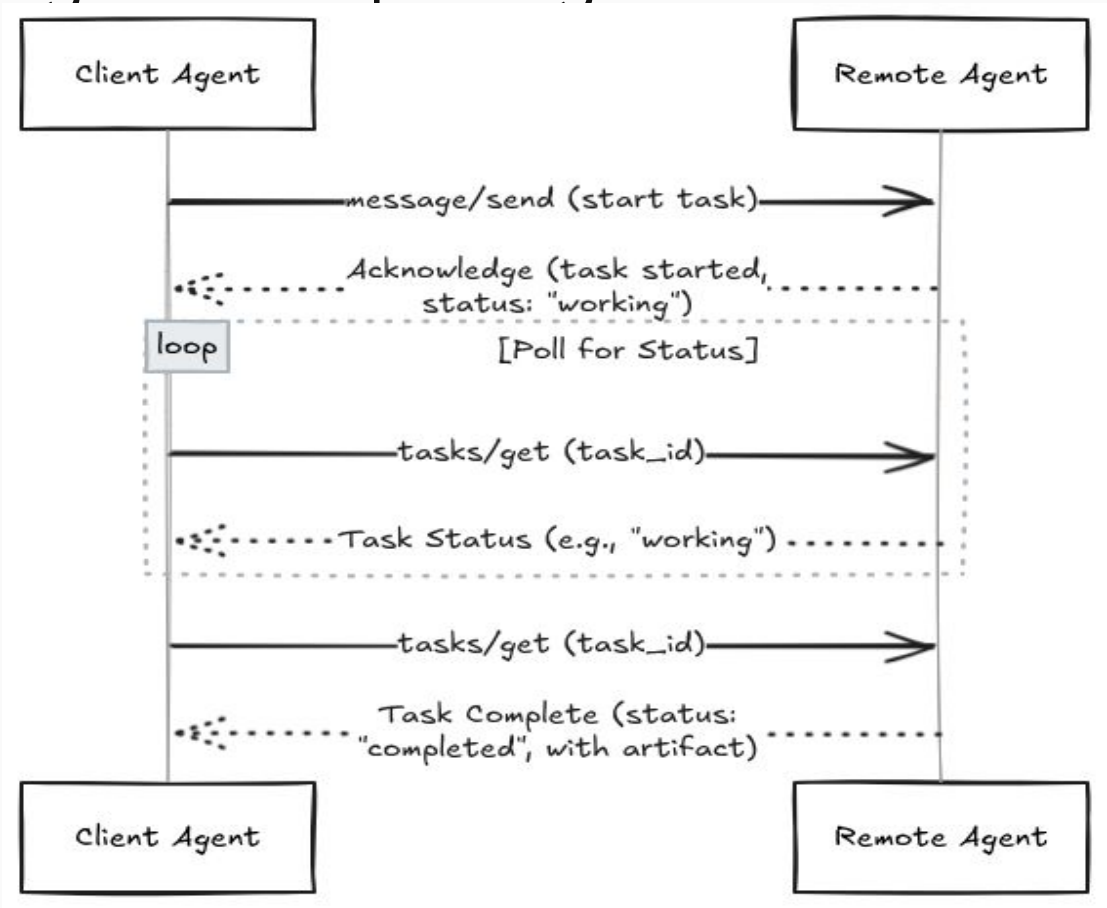
Task Lifecycle has

- Initial state (Submitted)
- Working State
- Blocking state (Input Required, Auth Required)
- Final state (Rejected, Failed, completed, Cancelled)
- What about “Unknown”

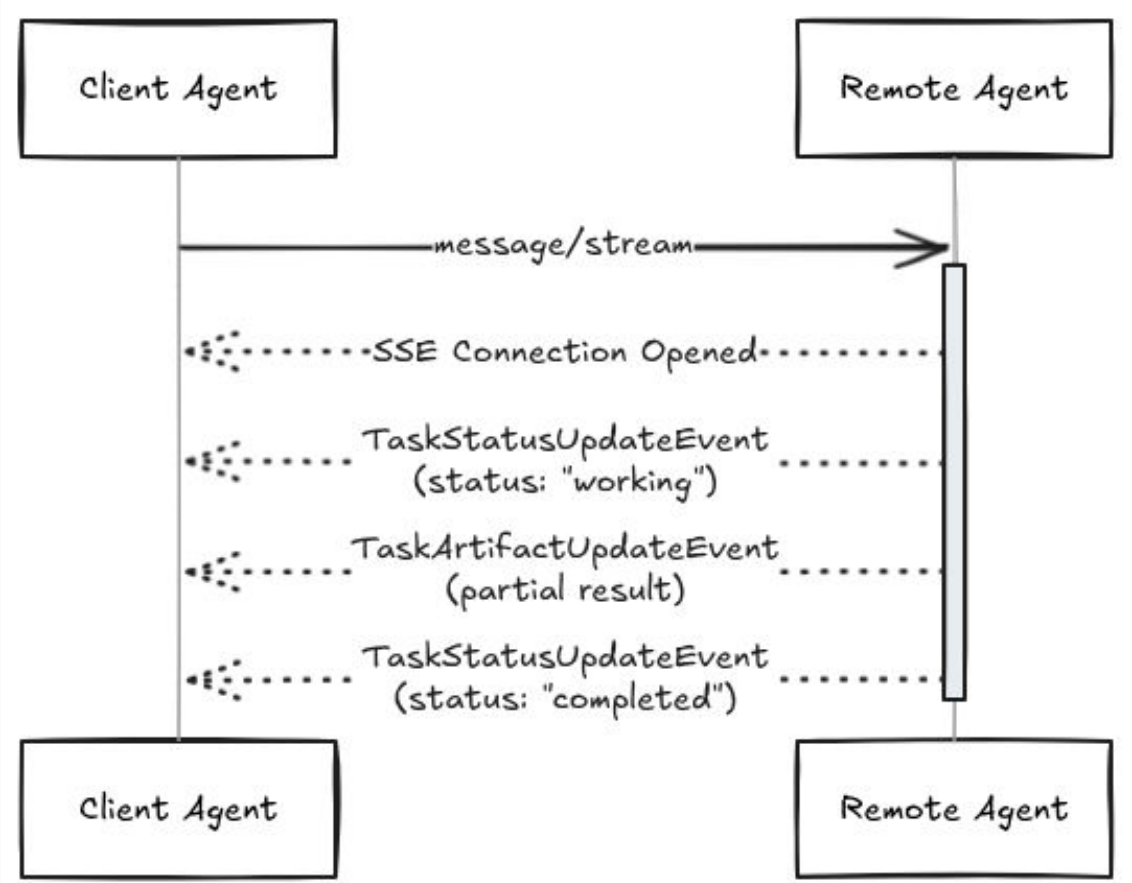
Task Lifecycle



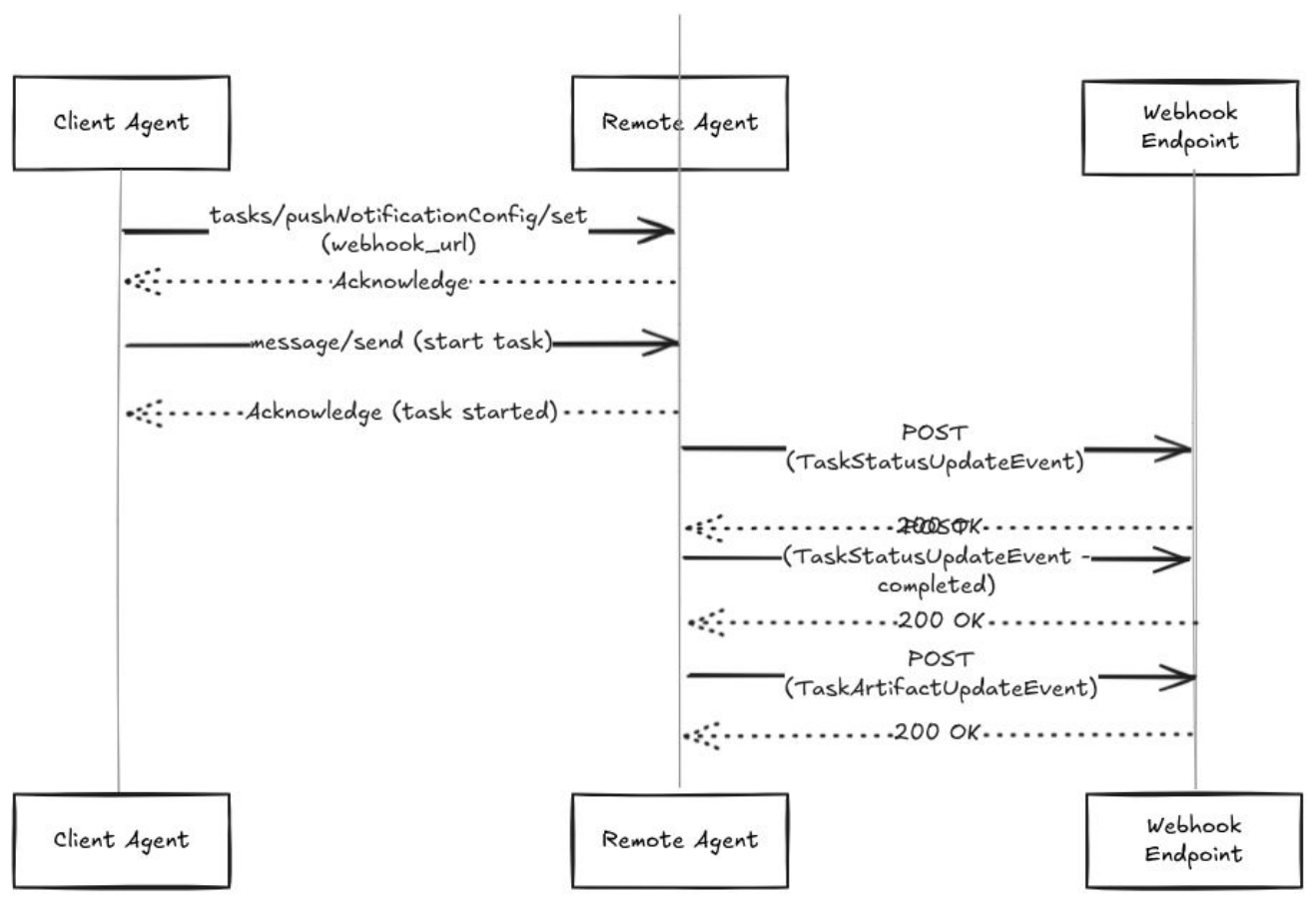
Task management - polling



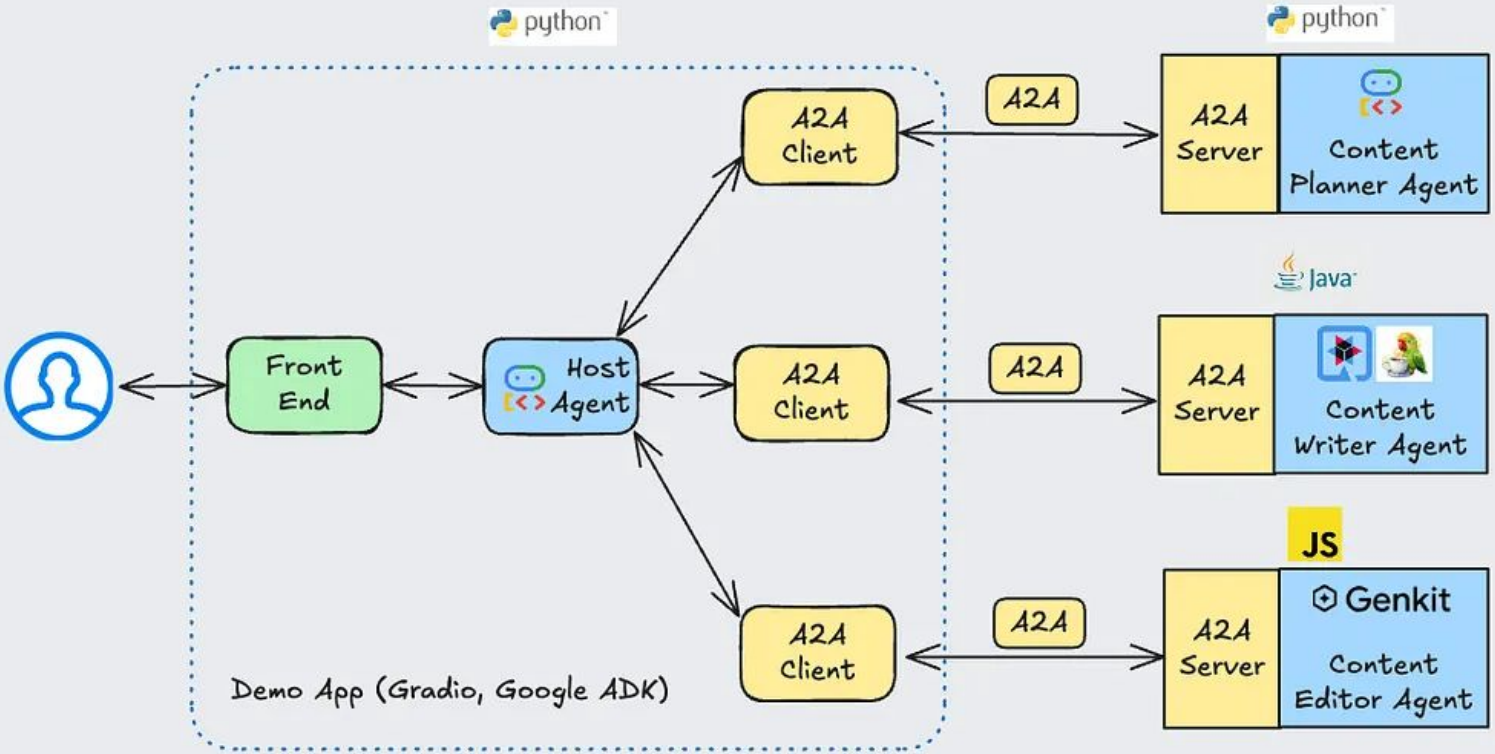
Task management - SSE



Task management - webhook



Multi-Agent System for Content Creation



Agent	Role	Technology Stack	Description
Host	A2A Client	Python, Google ADK, A2A Python SDK	Serves as the central orchestrator, routing requests to the appropriate agent based on the task at hand.
Content Planner	A2A Server	Python, Google ADK, A2A Python SDK	Receives a high-level content request and creates a detailed content outline.
Content Writer	A2A Server	Java, Quarkus LangChain4j, A2A Java SDK	Generates engaging content from a content outline.
Content Editor	A2A Server	TypeScript, Genkit, A2A JS SDK	Proofreads and polishes the given content.

A2A is transport agnostic

A2A is transport-agnostic, meaning it can work over various protocols.

Supported Transports:

- **JSON-RPC 2.0:** A popular and simple remote procedure call protocol.
- **gRPC:** A high-performance RPC framework from Google.
- **HTTP+JSON/REST:** Standard web protocols for broad compatibility.

Real-time updates:

- Uses **Server-Sent Events (SSE)**

Java SDK has an SPI for transports

Multi transport support

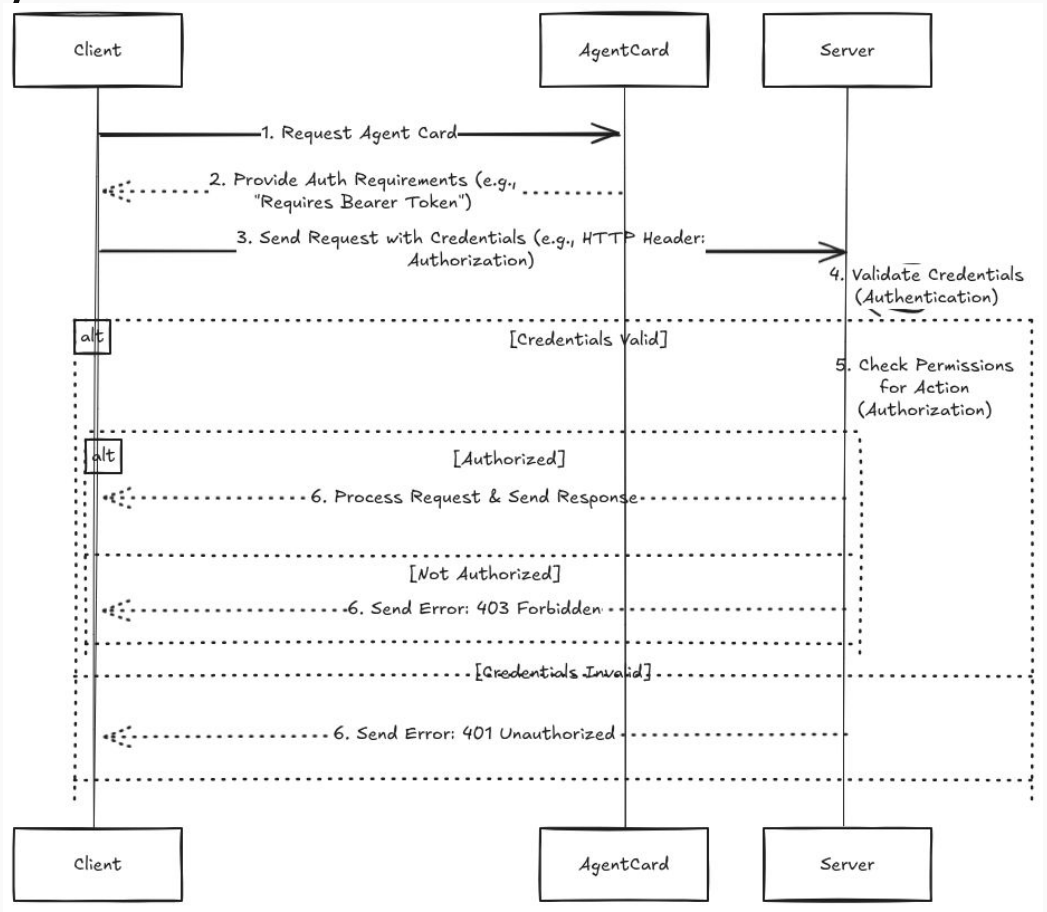
```
// the configured agent card

entCard
entCard agentCard() {
new AgentCard.Builder()
me(name:"Dice Agent")
scription(
 "Rolls an N-sided dice and answers questions about the "
     + "outcome of the dice rolls. Can also answer questio
     + "about prime numbers.")
eferredTransport(TransportProtocol.GRPC.asString())
l("localhost:" + httpPort)
rsion(version:"1.0.0")
cumentationUrl(documentationUrl:"http://example.com/docs")
```

```
.protocolVersion(protocolVersion: 0.3.0")
.additionalInterfaces(
    List.of(
        new AgentInterface(TransportProtoc
            "localhost:" + httpPort),
        new AgentInterface(
            TransportProtocol.JSONRPC.asSt
            "http://localhost:" + http
.build();
```

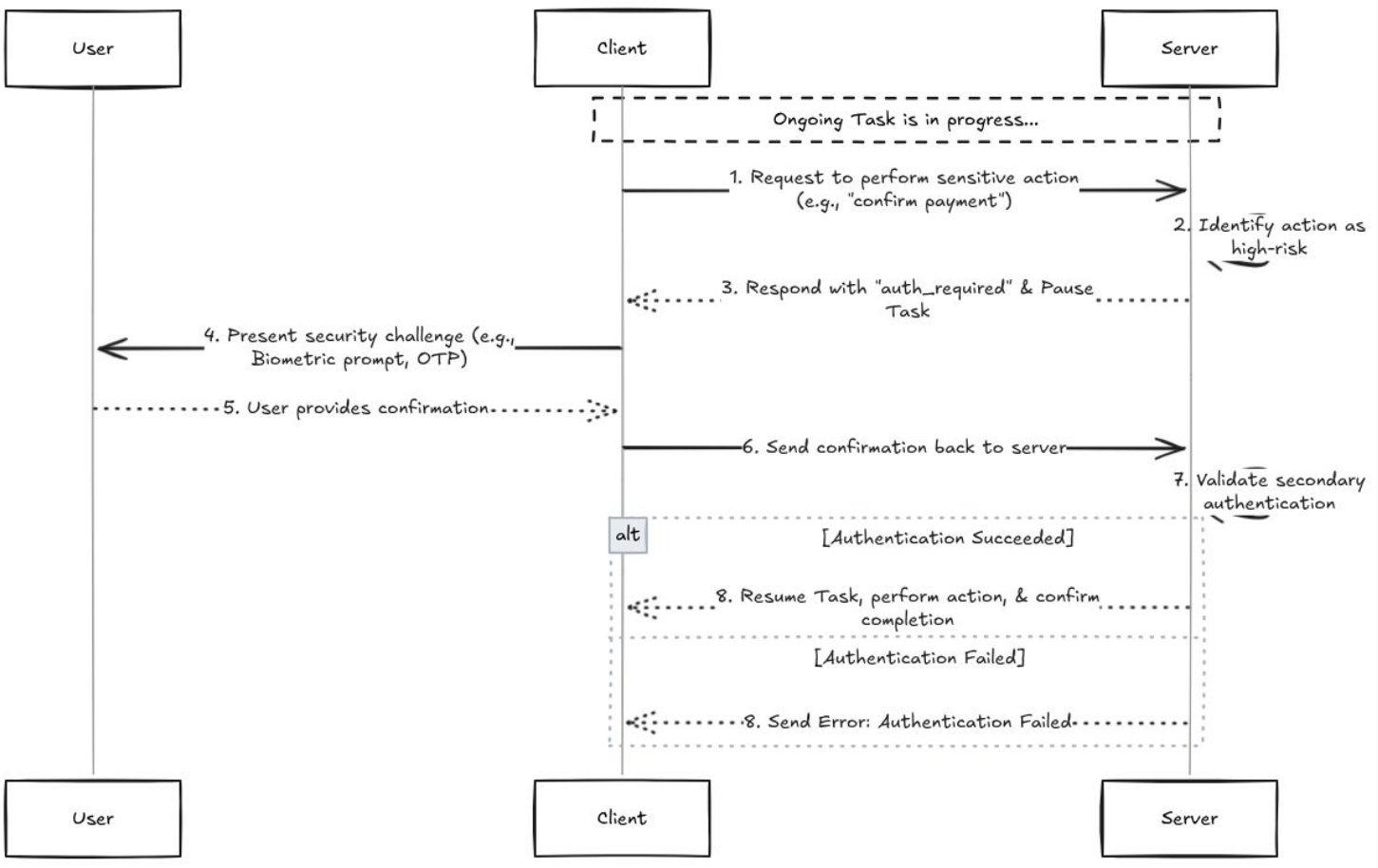
Security Foundations: Aligning with Web Standards

- **Built on OpenAPI & Web Standards:** A2A doesn't create a new security model. Instead, it aligns with the widely adopted **OpenAPI Specification**, leveraging proven, enterprise-grade security practices.
- **Authentication Mechanisms:** The specific method is declared in the `Agent Card`. Common types include:
  - **API Key:** A simple secret token sent in the HTTP headers. Best for server-to-server or internal service communication.
  - **HTTP Bearer (OAuth 2.0 / JWT):** The standard for delegated authorization. A client obtains a temporary access token (often a JWT) after a user grants permission, allowing the agent to act on the user's behalf.
  - **OpenID Connect (OIDC):** An identity layer built on top of OAuth 2.0. It provides user authentication information, allowing the agent to know *who* the user is, not just what they can do.
- **Transport Security (HTTPS):** All communication in production **MUST** be encrypted using HTTPS to protect data in transit.
- **Server Identity Verification:** The client **MUST** validate the server's TLS certificate to prevent man-in-the-middle attacks and ensure it's connecting to the authentic agent.

Security Basic Flow



Security Task auth-required



Extending the Protocol

What are Extensions?

Extensions are a formal way to add **new, optional capabilities** to the core A2A protocol. They allow agents and clients to agree on extra features beyond the base specification, ensuring interoperability.

How do they work?

- **Discovery:** declared in**Agent Card**, identified byURI.
- **Activation:** A client include the URI in the `X-A2A-Extensions` HTTP header.
- **Graceful Degradation:** If a client or server doesn't support an extension, communication falls back to the core protocol without breaking.

What can they add?

- **New Data:** structured information to the `AgentCard`.
- **New Methods:** Introduce custom RPC methods
- **New Rules ("Profiles"):** Impose extra requirements or state changes on core messages.
- **New Task States:** Enhance the task lifecycle with custom states.

Key Considerations

- **Security**
- **Versioning**
- **Reusability**

AP2: The Agent Payments Protocol

What is it?

An open protocol designed to create a secure, auditable, and interoperable framework for **payments initiated by AI agents**.

It is an A2A extension.

Core Concept: Trust through "Mandates"

AP2's security is built on **Mandates**

- **Intent Mandate:** Captures the user's initial high-level request and constraints (e.g., "Buy running shoes for under $100").
- **Cart Mandate:** Created when a user approves a specific cart of items. It's an unchangeable record of the exact items and price.
- **Payment Mandate:** Signals to the payment network that an AI agent is involved, providing visibility for risk management.

Key GoalsExtending the Protocol

- **Authorization:** Mandates provide clear, auditable proof that an agent had the authority to make a purchase.
- **Authenticity:** Ensures an agent's actions accurately reflect the user's true intent.
- **Accountability:** Creates a clear audit trail to resolve disputes or incorrect transactions.
- **Interoperability:** Payment-agnostic, supporting everything from credit cards to real-time bank transfers and stablecoins.

Java Implementation is coming

a2a-x402: Monetizing Agents with Web3

What is it?

An **A2A extension** that revives the spirit of the HTTP `402 Payment Required` status code, enabling AI agents to **charge for their services** and receive payments directly on-chain. I

How it works: A Simple Payment Flow

1. **Payment Required:** A "merchant" agent, when called, responds with a quote detailing the cost and payment method.
2. **Payment Submitted:** The "client" agent signs the payment details and sends them back as cryptographic proof.
3. **Payment Completed:** The merchant verifies the on-chain payment, settles it, and then delivers the requested service (e.g., API call, data processing, AI inference).

Key Concepts

- **Standardized Monetization:** Provides a common, interoperable language for payments between agents.
- **On-Chain Settlement:** Uses blockchains (like Ethereum) as a secure and decentralized payment rail.
- **Built for Web3:** Designed to work with cryptocurrencies and stablecoins, directly integrating with the Web3 ecosystem.

EIP-8004: A Trust Layer for AI Agents

What is it?

An Ethereum standard that creates a **trust and reputation layer** for AI agents, allowing them to discover, choose, and interact with each other across different organizations without pre-existing trust.

The Problem it Solves

Protocols like A2A work well *within* trusted boundaries, but they lack a mechanism to establish trust in an open, permissionless environment. EIP-8004 adds this missing decentralized trust layer.

Key Goal

To foster an **open, cross-organizational agent economy** on Ethereum by providing tools for agents to build a verifiable reputation, enabling secure interactions in a trustless setting.

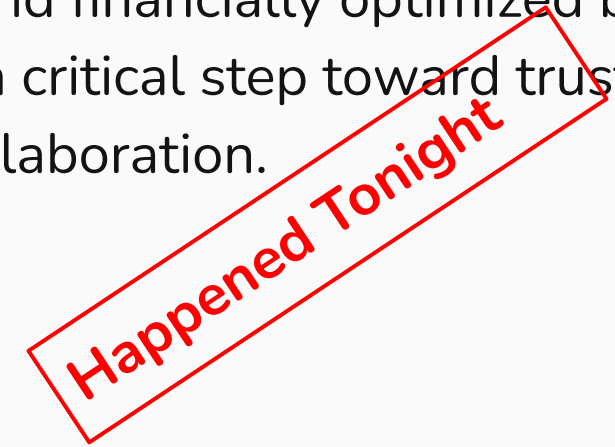
How it Works: On-Chain Registries

It introduces three lightweight, on-chain registries that act as a public record for agents:

- **Identity Registry:** Provides a portable, censorship-resistant ID for each agent.
- **Reputation Registry:** Allows client agents to leave on-chain attestations (feedback) about server agents after a task is complete.
- **Validation Registry:** A generic hook for verifying an agent's work, using pluggable trust models like crypto-economic security (staking) or cryptographic proofs (like TEE attestations).

Agent Gateway protocol

The Agent Gateway Protocol (AGP) offers a powerful and necessary enhancement layer over the foundational A2A structure. By implementing Policy-Based Routing, AGP ensures that distributed AI systems are not only efficient and financially optimized but also secure and policy-compliant—a critical step toward trustworthy, industrial-scale multi-agent collaboration.



Agent Gateway protocol

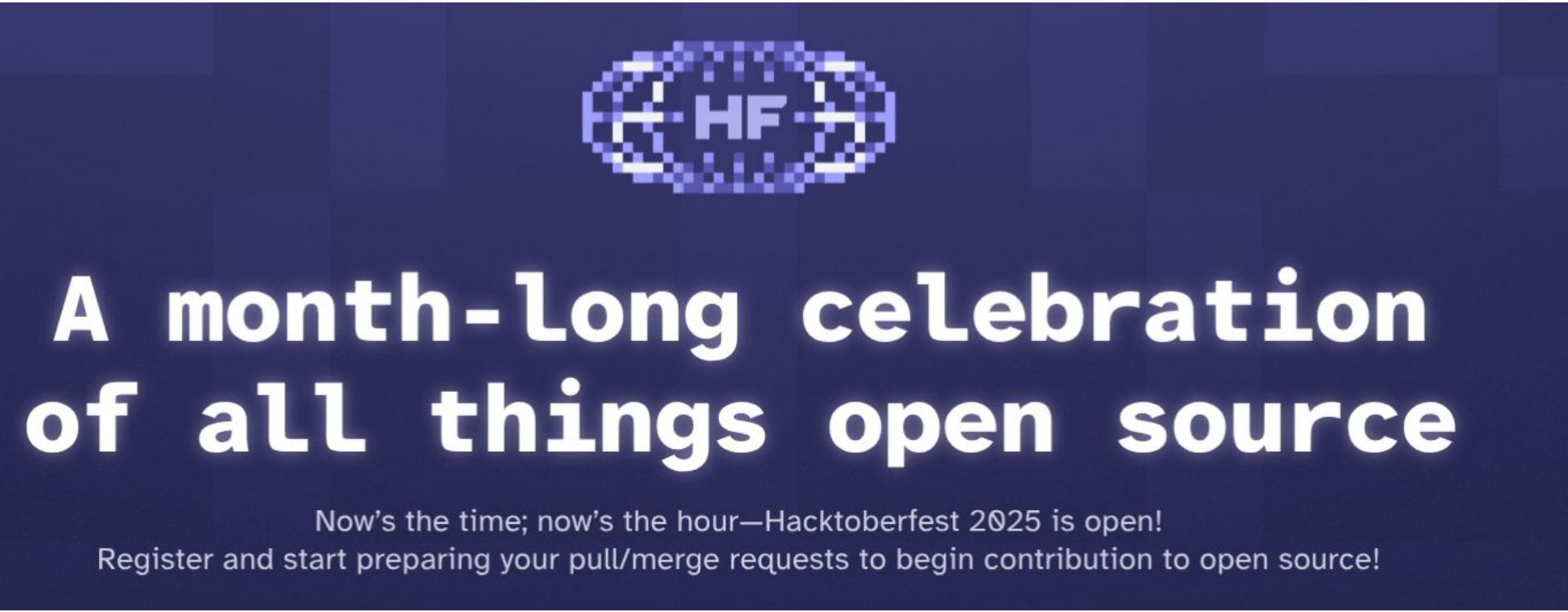
Example Announcement Payload

```
{
  "capability": "financial_analysis:quarterly",
  "version": "1.5",
  "cost": 0.05,
  "policy": {
    "requires_auth": "level_3"
  }
}
```

Example Intent Payload

```
{
  "target_capability": "billing:invoice:generate",
  "payload": {
    "customer_id": 123,
    "amount": 99.99
  },
  "policy_constraints": {
    "requires_pii": true
  }
}
```

Java SDK contributing



Code, slides and more

https://maeste.it/devoxx/

Bonus/Teaser :)

D&D Analogy

The Agent: Your Character

- An agent acts as a character on the board
- Characters are avatars with specific roles
- Characters have unique capabilities
- Characters have persistent states
- Characters have well defined sheets of skills and abilities
- Characters may create plans for long running tasks and involve the use of multiple tools
- Characters can collaborate with other characters to achieve a goal

MCP Tools: Your Armaments

- MCP Tools are equipment and armaments
- Character uses equipment and armaments
- Each armament provide structured actions for the player
- Each armament has defined rules and effects

LLM: Player's Intelligence

- LLM is the intelligence of the player
- Player uses the LLM to achieve goals
- LLM helps to make decisions
- LLM is core to player actions

Dice Roll

When the player (LLM) decides to act, the outcome isn't guaranteed. But more important people doesn't trust each other...

Here is where Security of A2A and more advanced technique for untrusted agents and reputations comes into play

A2A standard protocol

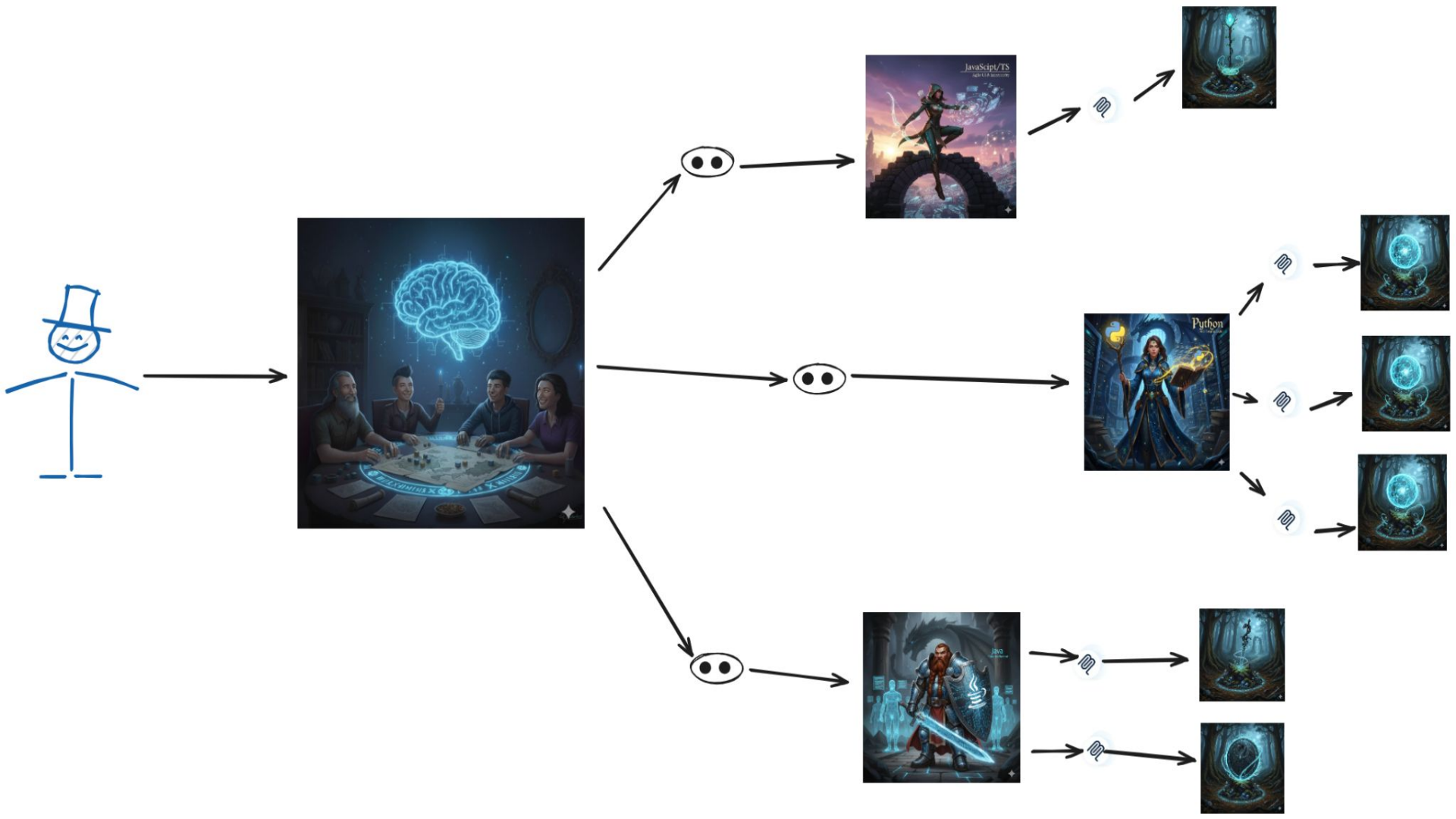
For a good game we need good rules

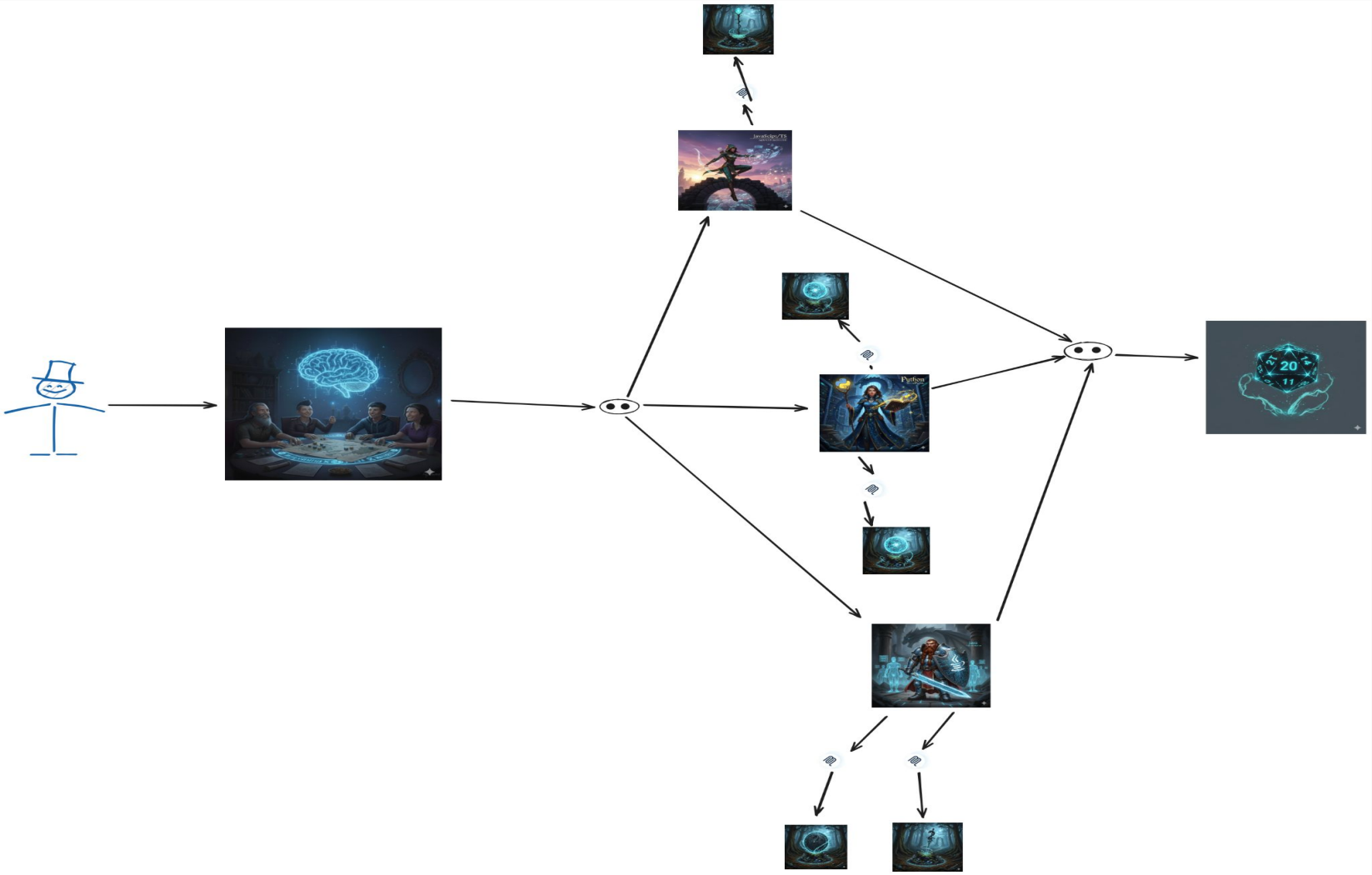
The same for a good agent communication

A2A: Strategic "Table Talk"

- A2A enables strategic collaboration between LLM players.
- It facilitates multi-turn conversations and shared intent.
- Players communicate plans and adjust actions accordingly.
- A2A governs how LLMs collaborate effectively in a shared environment.

Agent Interoperability: difference is a value


JavaScipt/TS
Python
Java


JavaScript/TS
Python
20

Code, slides and more

https://maeste.it/devoxx/