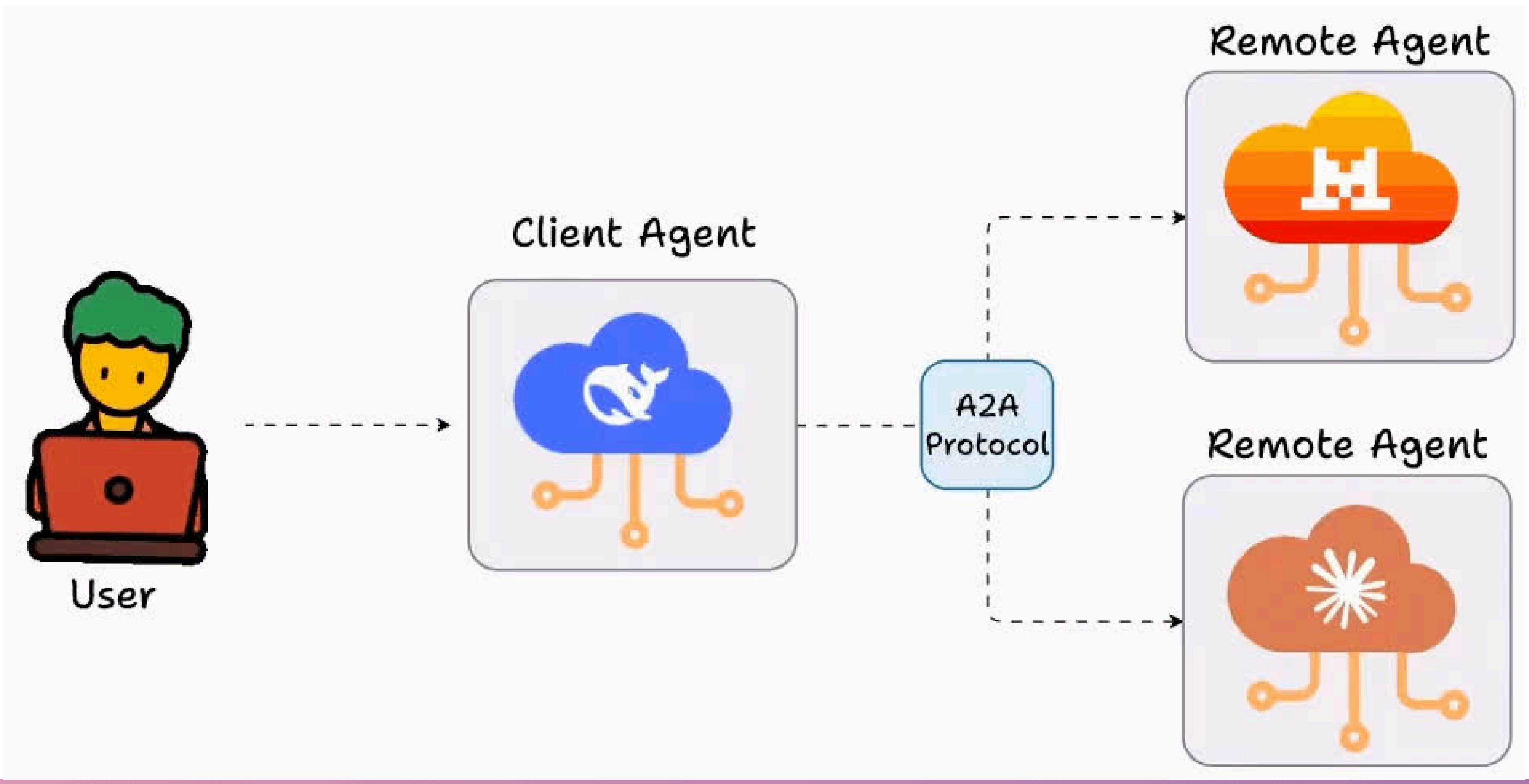
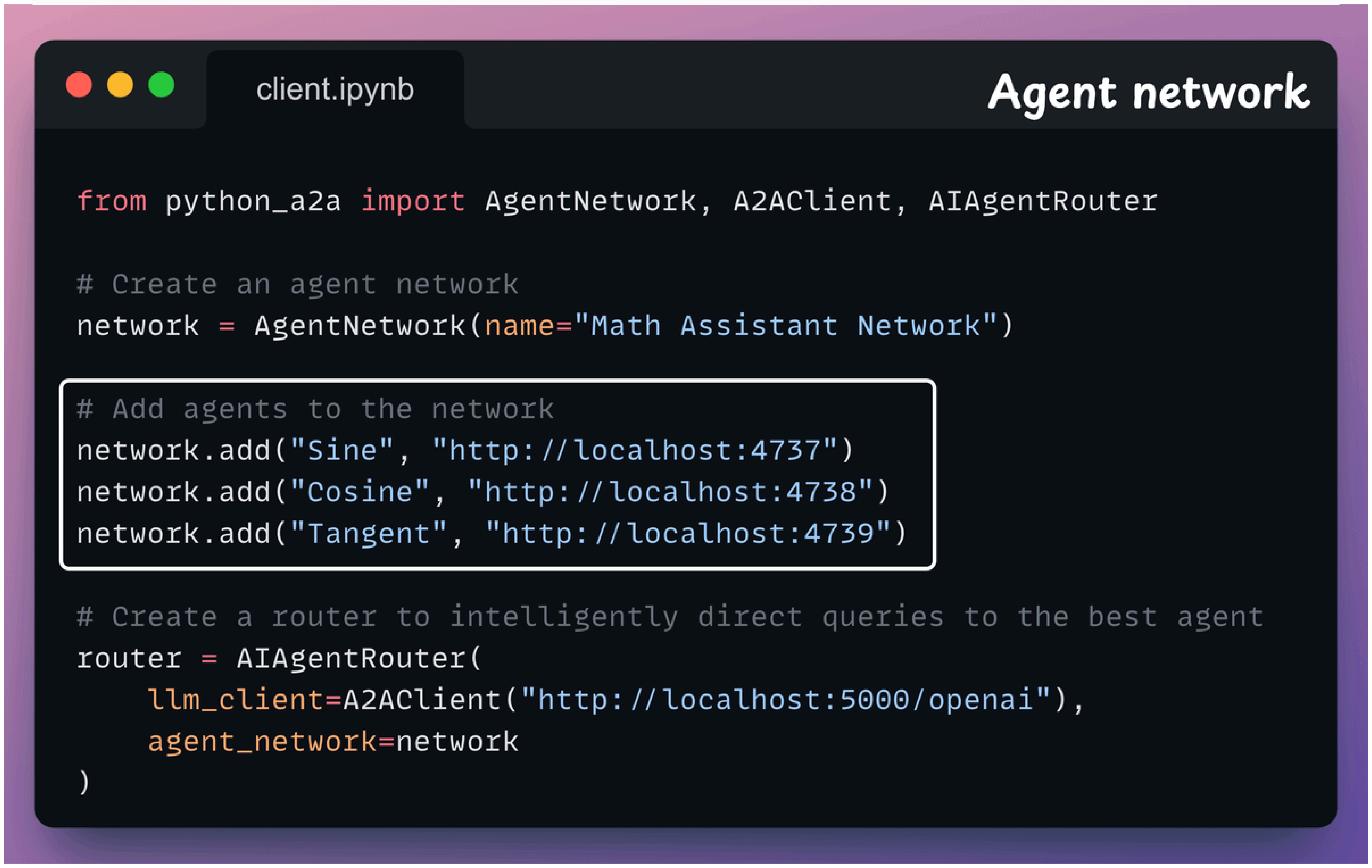
A2A Protocol, clearly explained!
(with code example)



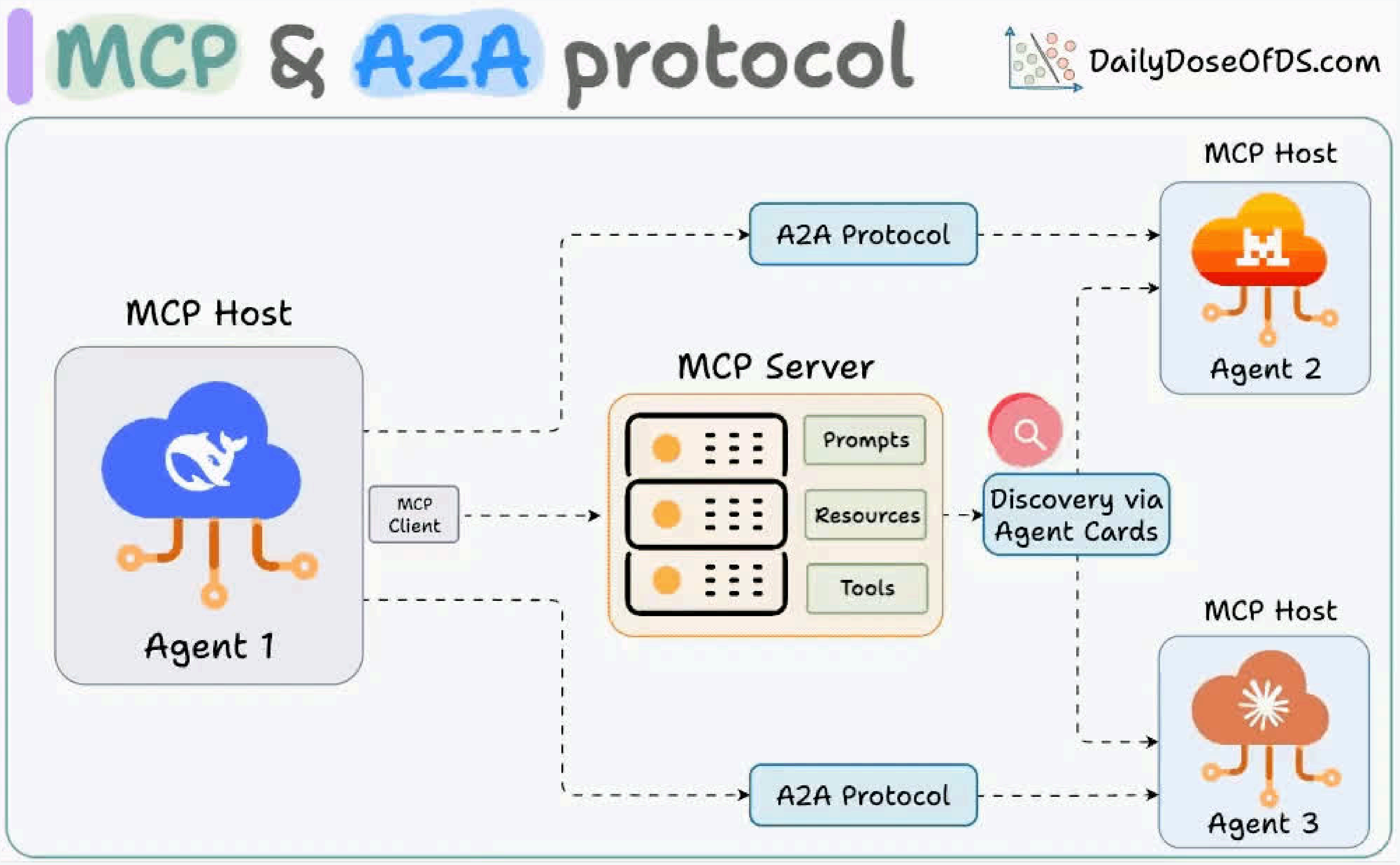


```python
from python_a2a import AgentNetwork, A2AClient, AIAgentRouter

# Create an agent network
network = AgentNetwork(name="Math Assistant Network")

# Add agents to the network
network.add("Sine", "http://localhost:4737")
network.add("Cosine", "http://localhost:4738")
network.add("Tangent", "http://localhost:4739")

# Create a router to intelligently direct queries to the best agent
router = AIAgentRouter(
    llm_client=A2AClient("http://localhost:5000/openai"),
    agent_network=network
)
```

Before diving in, here's what we'll do:

- Understand A2A in simple terms.
- Briefly compare A2A and MCP
- Next, we'll build three Agents and serve them locally.
- Finally, we'll talk to them via the Agent2Agent protocol by Google.

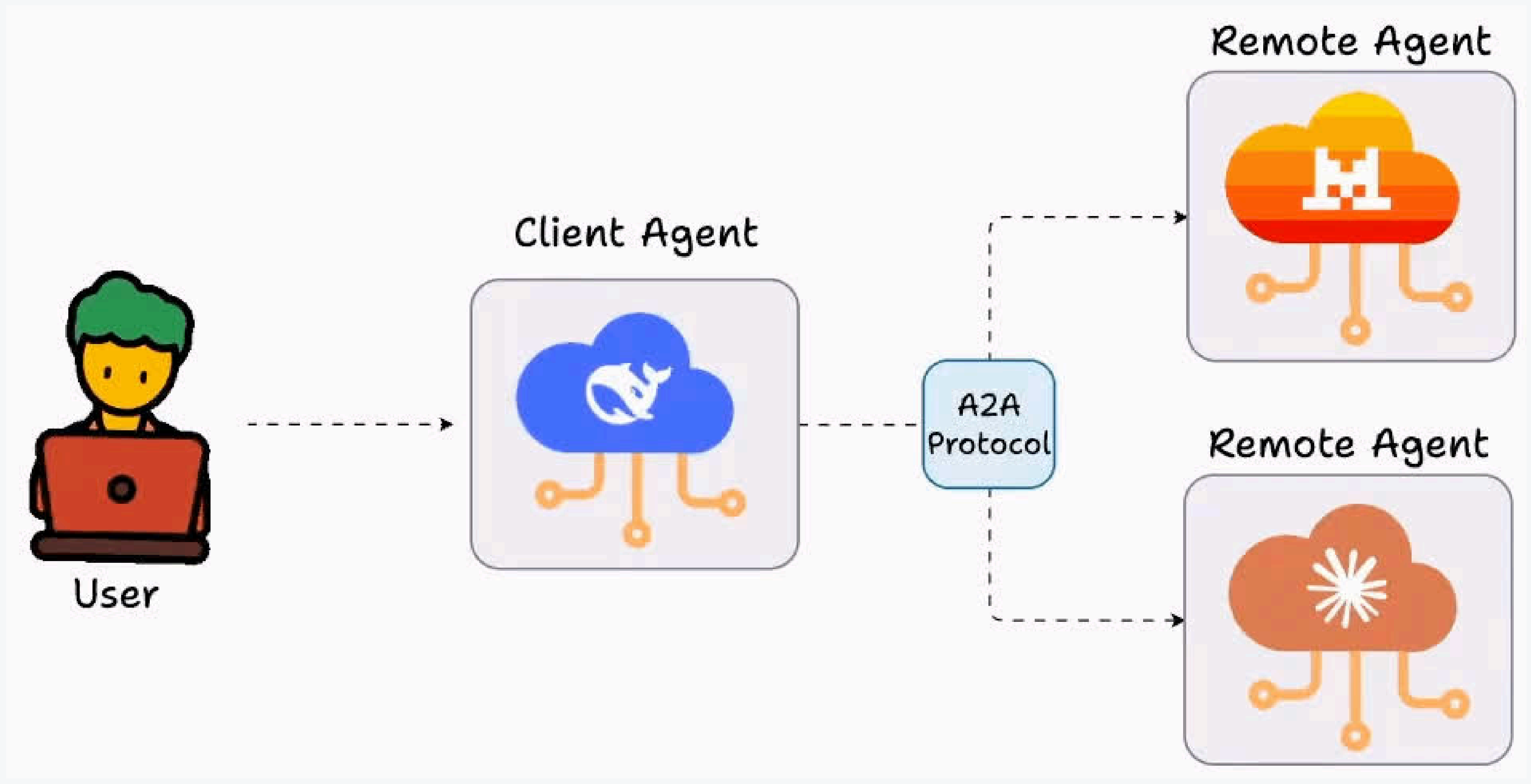
Let's dive in! 🚀



What is A2A?

A2A (Agent2Agent) enables multiple AI agents to work together on tasks without directly sharing their internal memory, thoughts, or tools.

Instead, they communicate by exchanging context, task updates, instructions, and data.

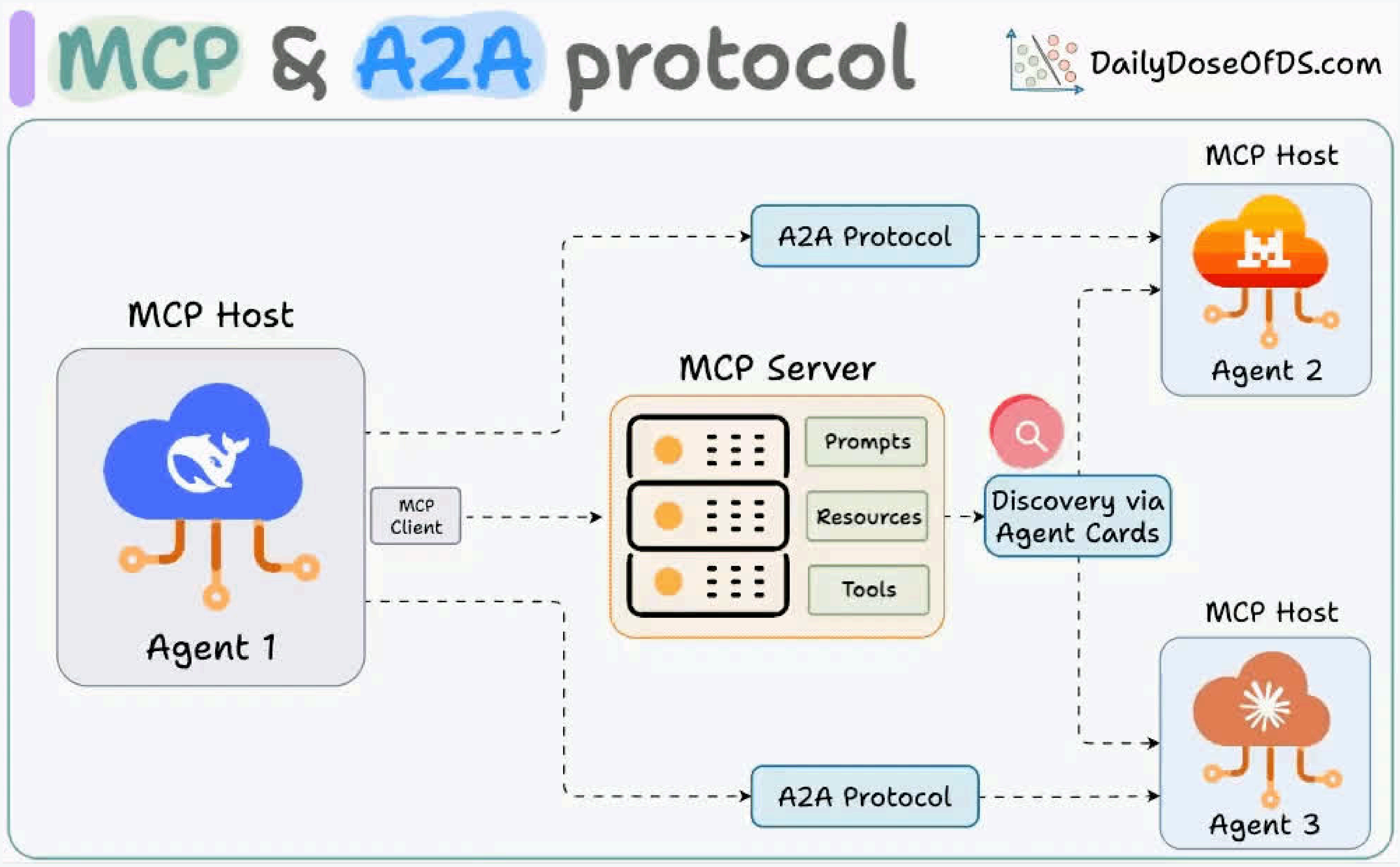


How does A2A fit with MCP?

Agentic apps need both:
- MCP connects agents to external tools
- A2A enables agent to agent collaboration

Visual below shows A2A and MCP collaboration.

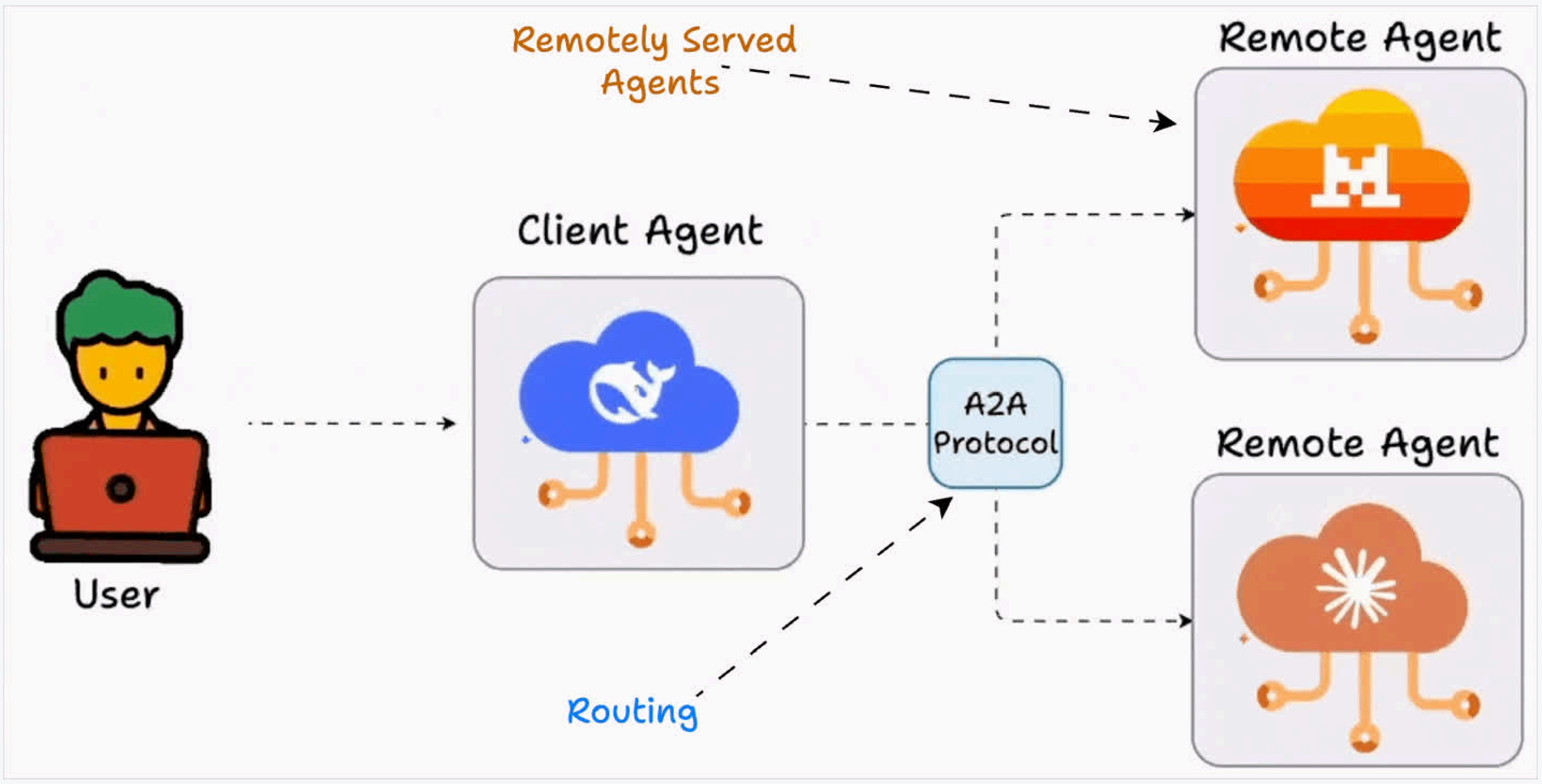
We've covered MCP before; today we'll focus on a hands-on A2A example.



Before we code, some key concepts to understand:

1. Agent Server: Hosts individual agents
2. Client Agent: Connects to multiple servers
3. Router: Routes queries to right agent

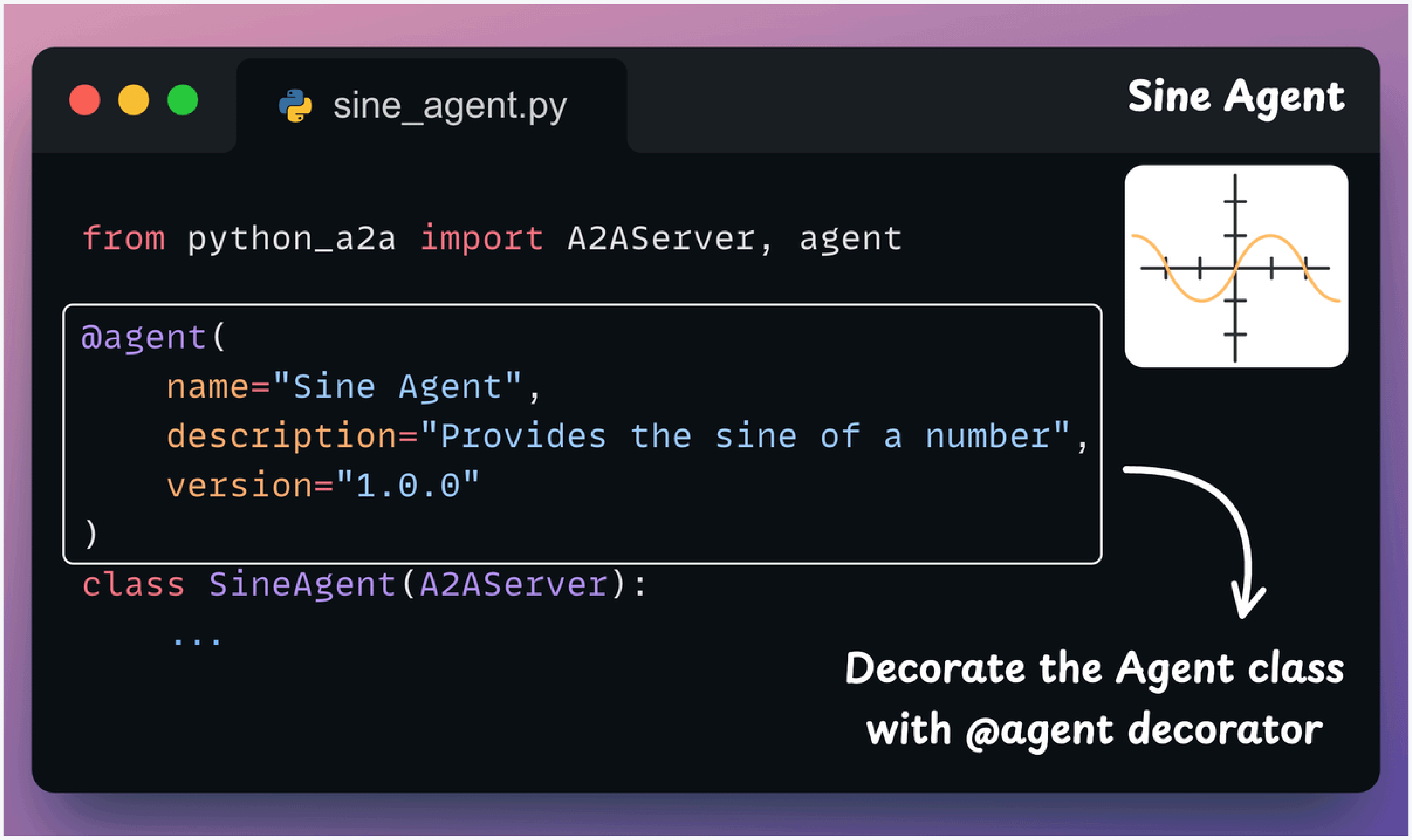
And here's what we'll do:
- Server 3 agents
- Build a client
- and a router



Let's start by learning how to serve an Agent.

- Create an Agent class that inherits from A2AServer.
- Decorate the class with the agent decorator, specifying the agent's name, description, and version.

Check this out👇



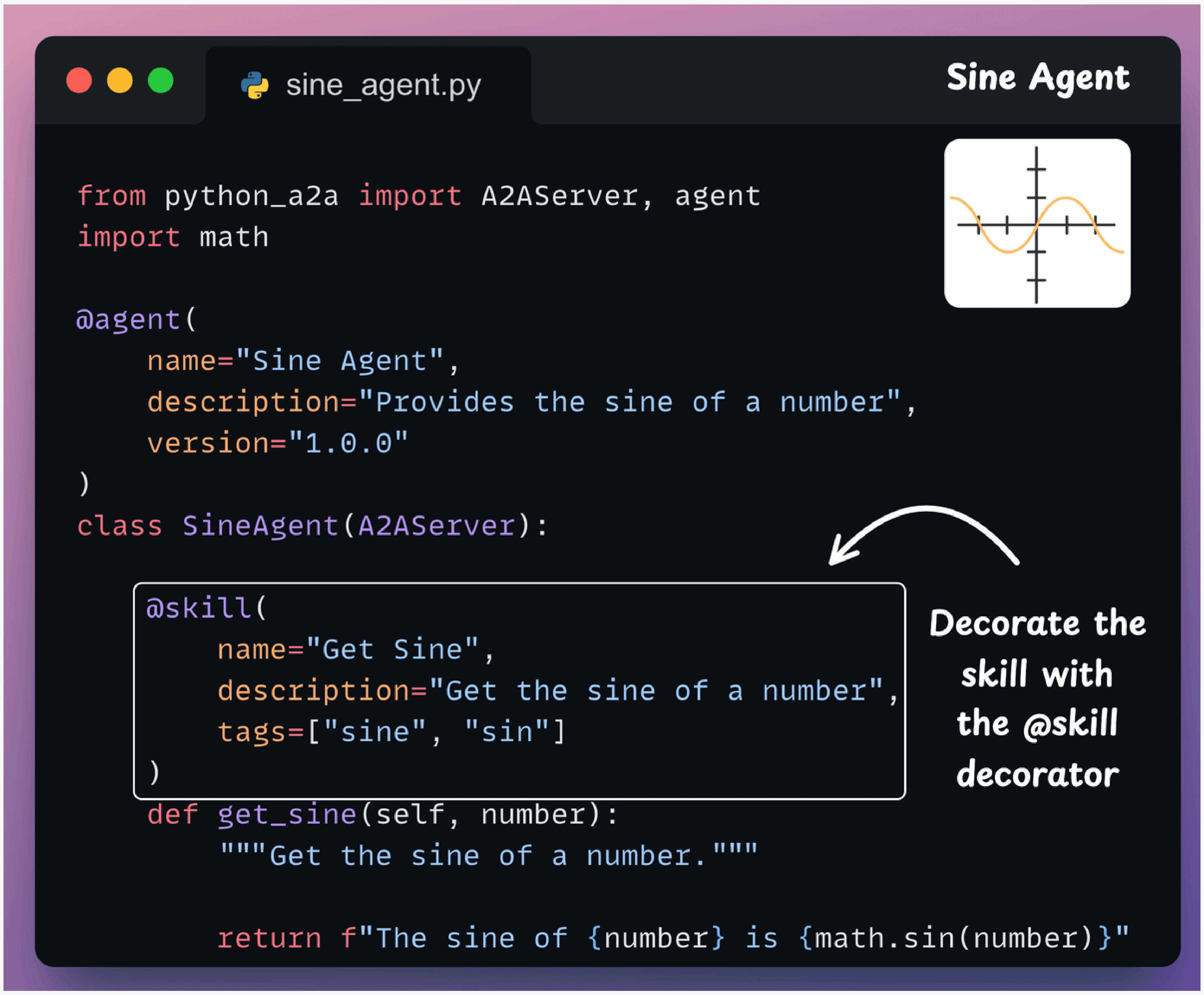
```python
from python_a2a import A2AServer, agent

@agent(
    name="Sine Agent",
    description="Provides the sine of a number",
    version="1.0.0"
)
class SineAgent(A2AServer):
    ...
```

Adding skills to the Agents

Each Agent has some skills that it can expose to other Agents. So the class must declare a method with the skill decorator.

For instance, a skill of our SineAgent is that it can compute the Sine of a number.

Here's how we do it 👇



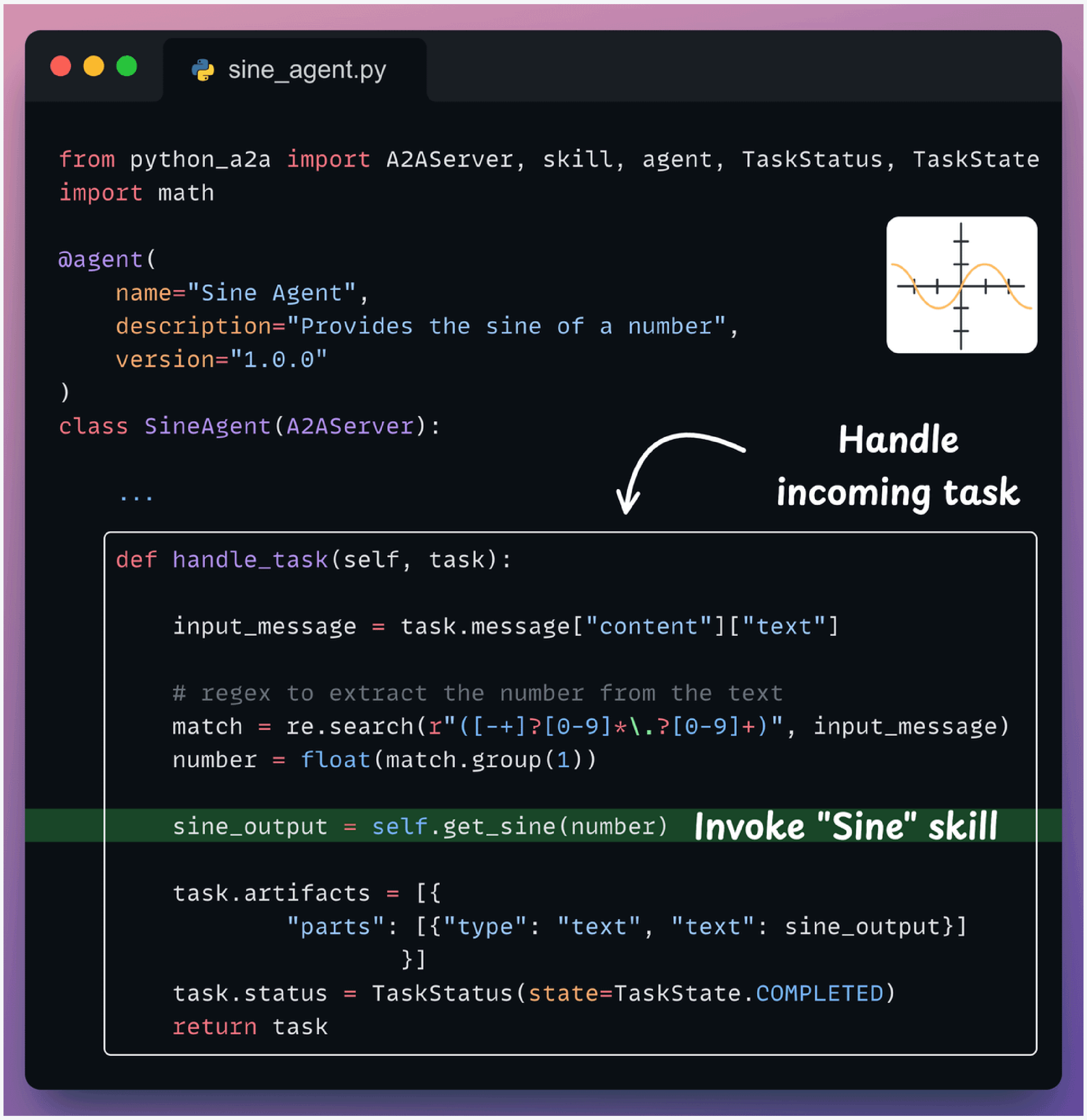
```python
from python_a2a import A2AServer, agent
import math

@agent(
    name="Sine Agent",
    description="Provides the sine of a number",
    version="1.0.0"
)
class SineAgent(A2AServer):

    @skill(
        name="Get Sine",
        description="Get the sine of a number",
        tags=["sine", "sin"]
    )
    def get_sine(self, number):
        """Get the sine of a number."""

        return f"The sine of {number} is {math.sin(number)}"
```

Handling tasks

An Agent can have multiple skills. Thus, a handle_task method accepts the input task, uses the available skills, and returns the output.

In our case, we simply invoke the Sine skill method after parsing the input.

Check this code👇

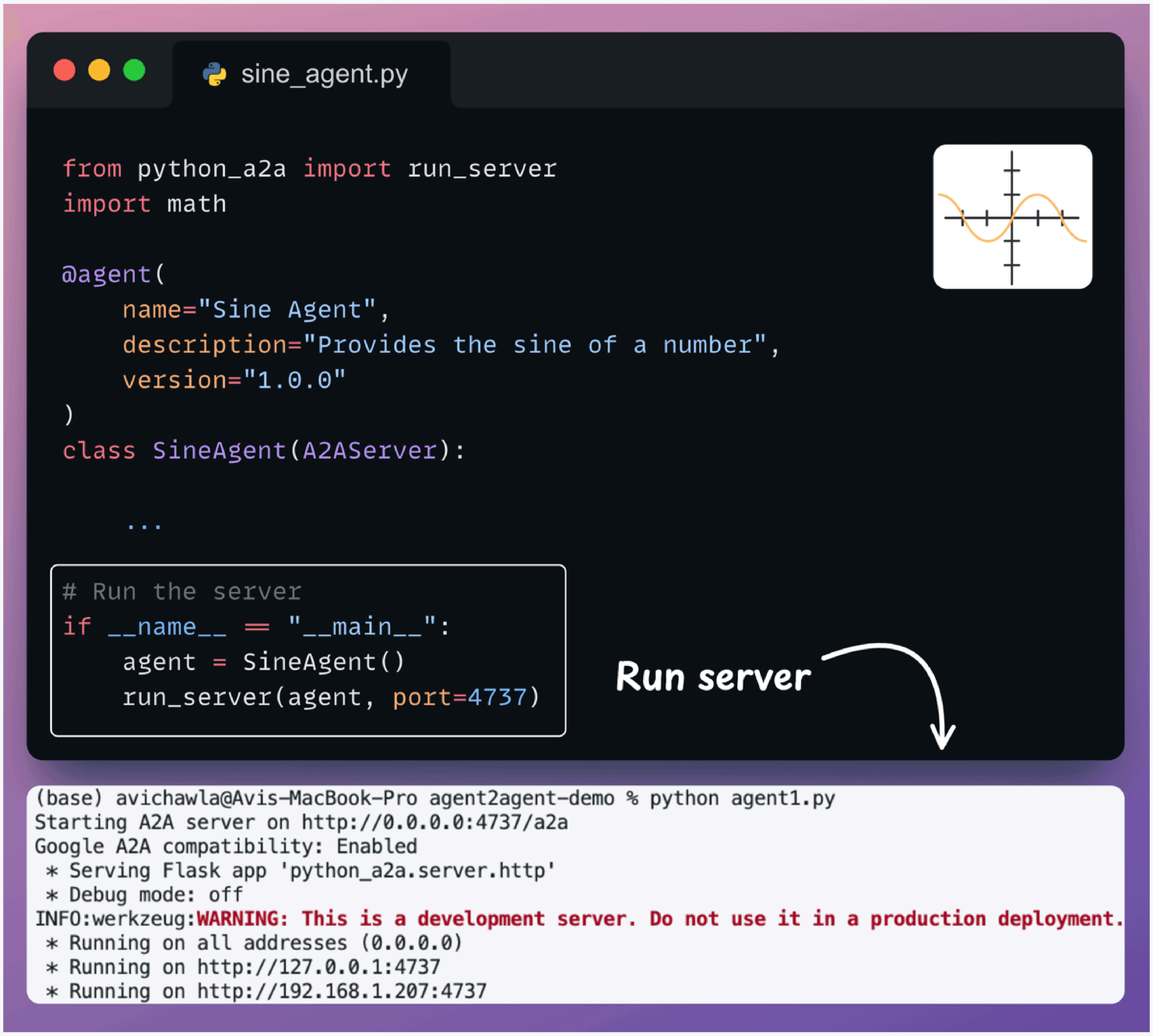

sine_agent.py

```python
from python_a2a import A2AServer, skill, agent, TaskStatus, TaskState
import math

@agent(
    name="Sine Agent",
    description="Provides the sine of a number",
    version="1.0.0"
)
class SineAgent(A2AServer):

    ...

    def handle_task(self, task):

        input_message = task.message["content"]["text"]

        # regex to extract the number from the text
        match = re.search(r"([-+]?[0-9]*\.?[0-9]+)", input_message)
        number = float(match.group(1))

        sine_output = self.get_sine(number)

        task.artifacts = [{
                "parts": [{"type": "text", "text": sine_output}]
                    }]
        task.status = TaskStatus(state=TaskState.COMPLETED)
        return task
```

Deploying the agent

Finally, in the main body of the agent, we deploy the Agent:

Check this code👇

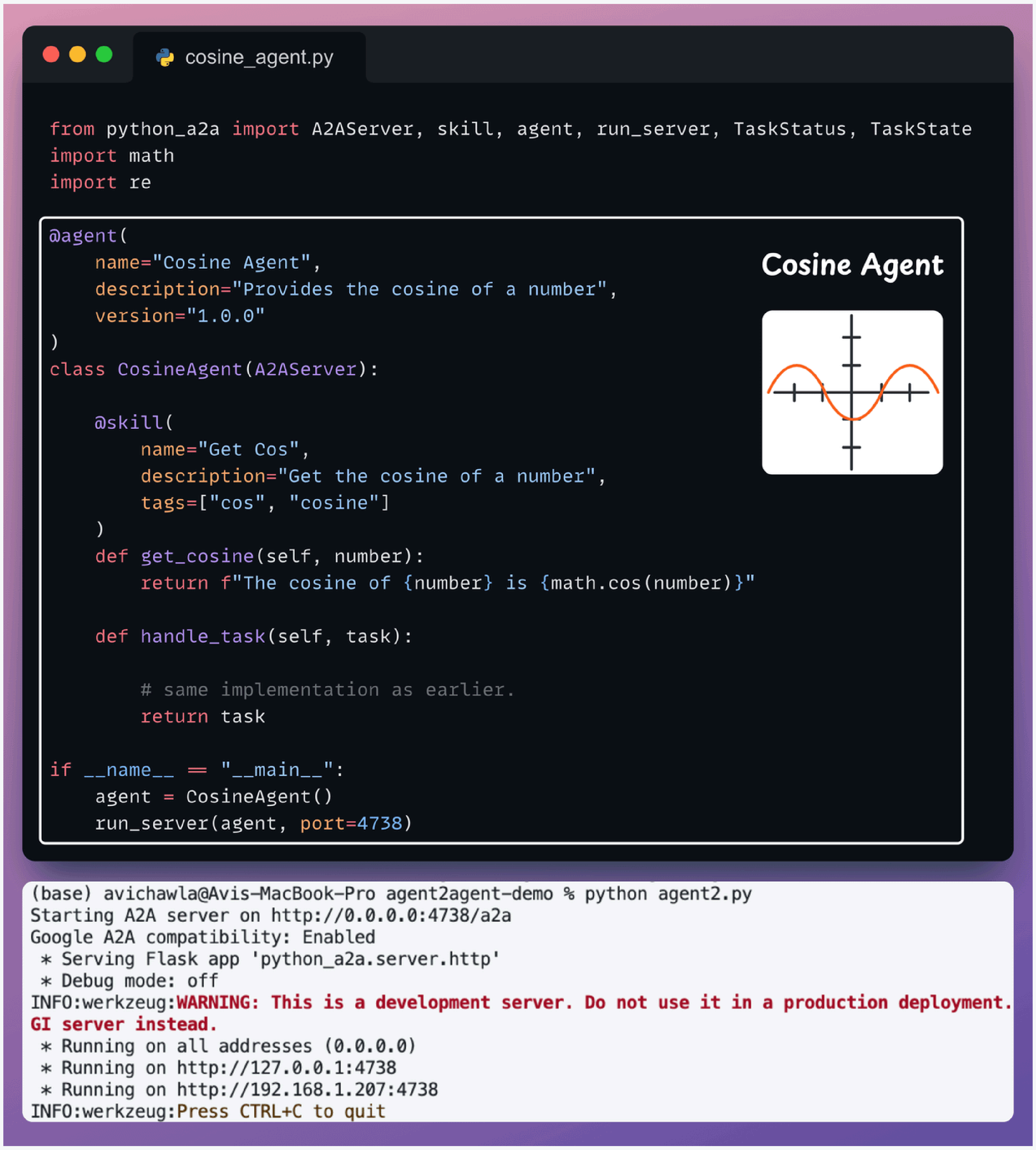


```python
from python_a2a import run_server
import math

@agent(
    name="Sine Agent",
    description="Provides the sine of a number",
    version="1.0.0"
)
class SineAgent(A2AServer):

    ...

# Run the server
if __name__ == "__main__":
    agent = SineAgent()
    run_server(agent, port=4737)
```

```
(base) avichawla@Avis-MacBook-Pro agent2agent-demo % python agent1.py
Starting A2A server on http://0.0.0.0:4737/a2a
Google A2A compatibility: Enabled
 * Serving Flask app 'python_a2a.server.http'
 * Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:4737
 * Running on http://192.168.1.207:4737
```

Similarly we'll build and deploy Cosine Agent

Here's the full code👇

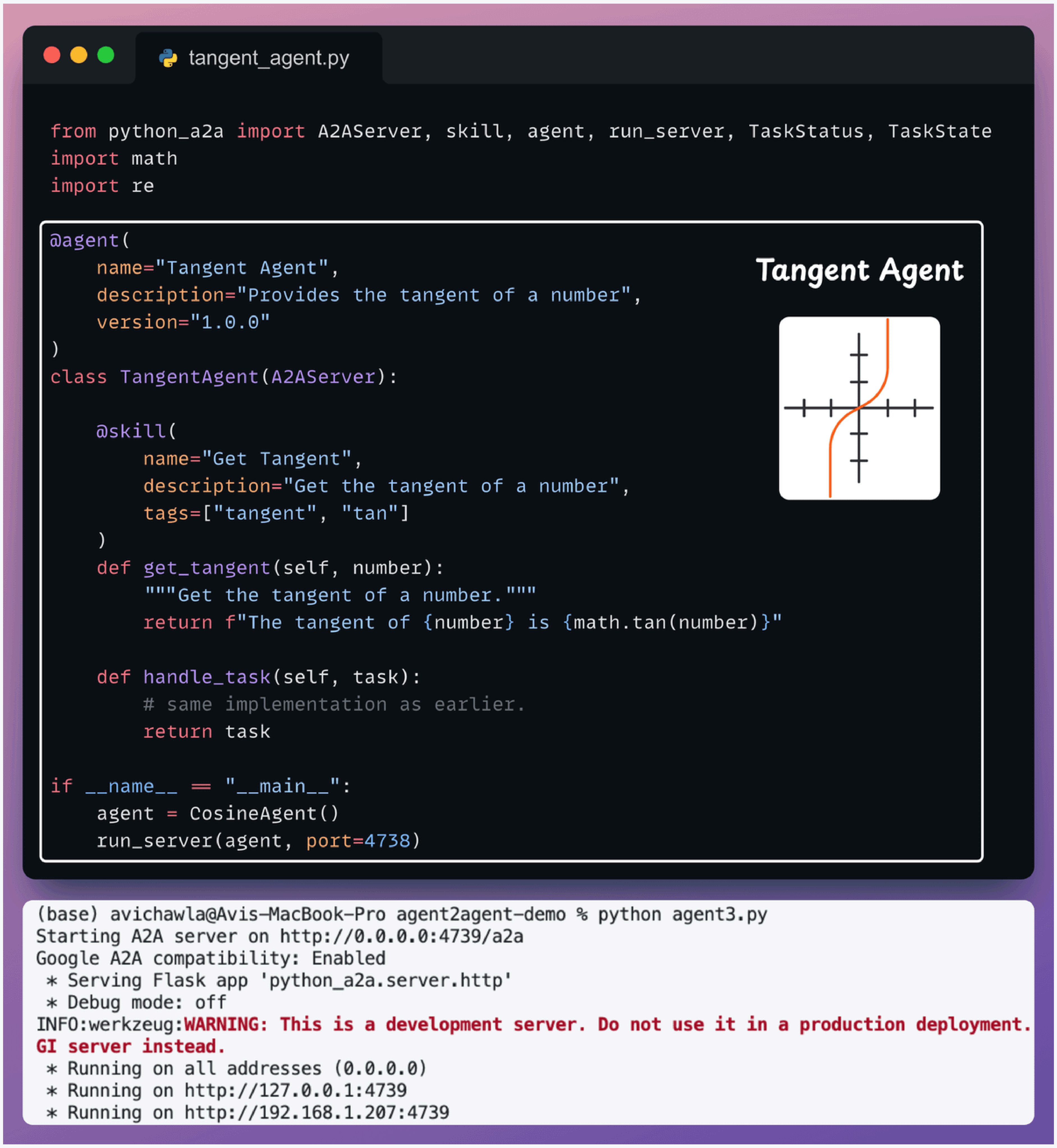


```python
from python_a2a import A2AServer, skill, agent, run_server, TaskStatus, TaskState
import math
import re

@agent(
    name="Cosine Agent",
    description="Provides the cosine of a number",
    version="1.0.0"
)
class CosineAgent(A2AServer):

    @skill(
        name="Get Cos",
        description="Get the cosine of a number",
        tags=["cos", "cosine"]
    )
    def get_cosine(self, number):
        return f"The cosine of {number} is {math.cos(number)}"

    def handle_task(self, task):

        # same implementation as earlier.
        return task

if __name__ == "__main__":
    agent = CosineAgent()
    run_server(agent, port=4738)
```

```
(base) avichawla@Avis-MacBook-Pro agent2agent-demo % python agent2.py
Starting A2A server on http://0.0.0.0:4738/a2a
Google A2A compatibility: Enabled
 * Serving Flask app 'python_a2a.server.http'
 * Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment.
GI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:4738
 * Running on http://192.168.1.207:4738
INFO:werkzeug:Press CTRL+C to quit
```

And finally, build and deploy the third agent, a Tangent Agent.

Here's the full code👇



tangent_agent.py

```python
from python_a2a import A2AServer, skill, agent, run_server, TaskStatus, TaskState
import math
import re

@agent(
    name="Tangent Agent",
    description="Provides the tangent of a number",
    version="1.0.0"
)
class TangentAgent(A2AServer):

    @skill(
        name="Get Tangent",
        description="Get the tangent of a number",
        tags=["tangent", "tan"]
    )
    def get_tangent(self, number):
        """Get the tangent of a number."""
        return f"The tangent of {number} is {math.tan(number)}"

    def handle_task(self, task):
        # same implementation as earlier.
        return task

if __name__ == "__main__":
    agent = CosineAgent()
    run_server(agent, port=4738)
```

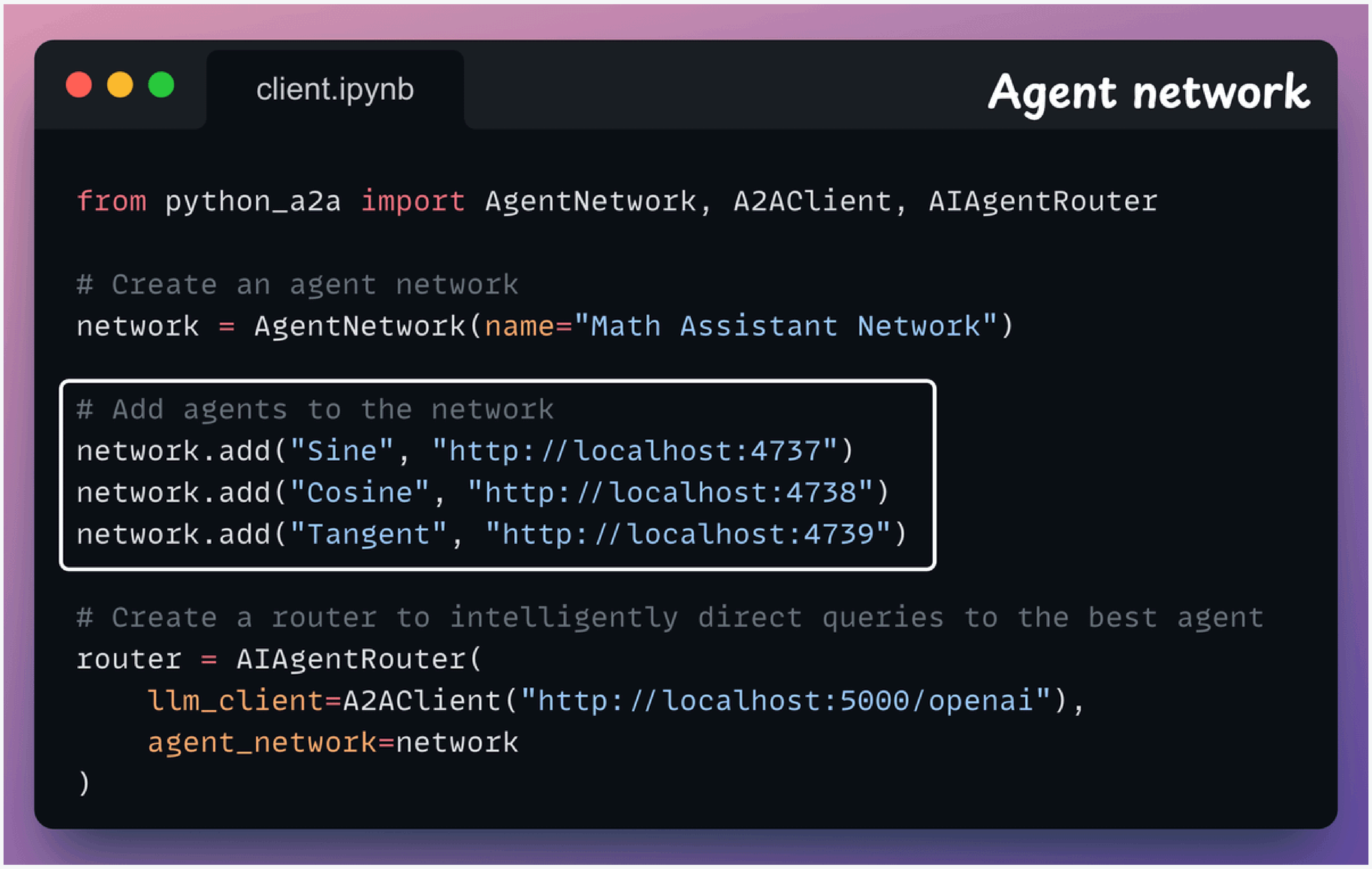
```
(base) avichawla@Avis-MacBook-Pro agent2agent-demo % python agent3.py
Starting A2A server on http://0.0.0.0:4739/a2a
Google A2A compatibility: Enabled
 * Serving Flask app 'python_a2a.server.http'
 * Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment.
GI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:4739
 * Running on http://192.168.1.207:4739
```

Next comes, client-side Agent integration

Now that Agents have been hosted and deployed, a client can integrate them into their existing applications.

We also add a router to intelligently direct queries.

Here's how we do it 👇



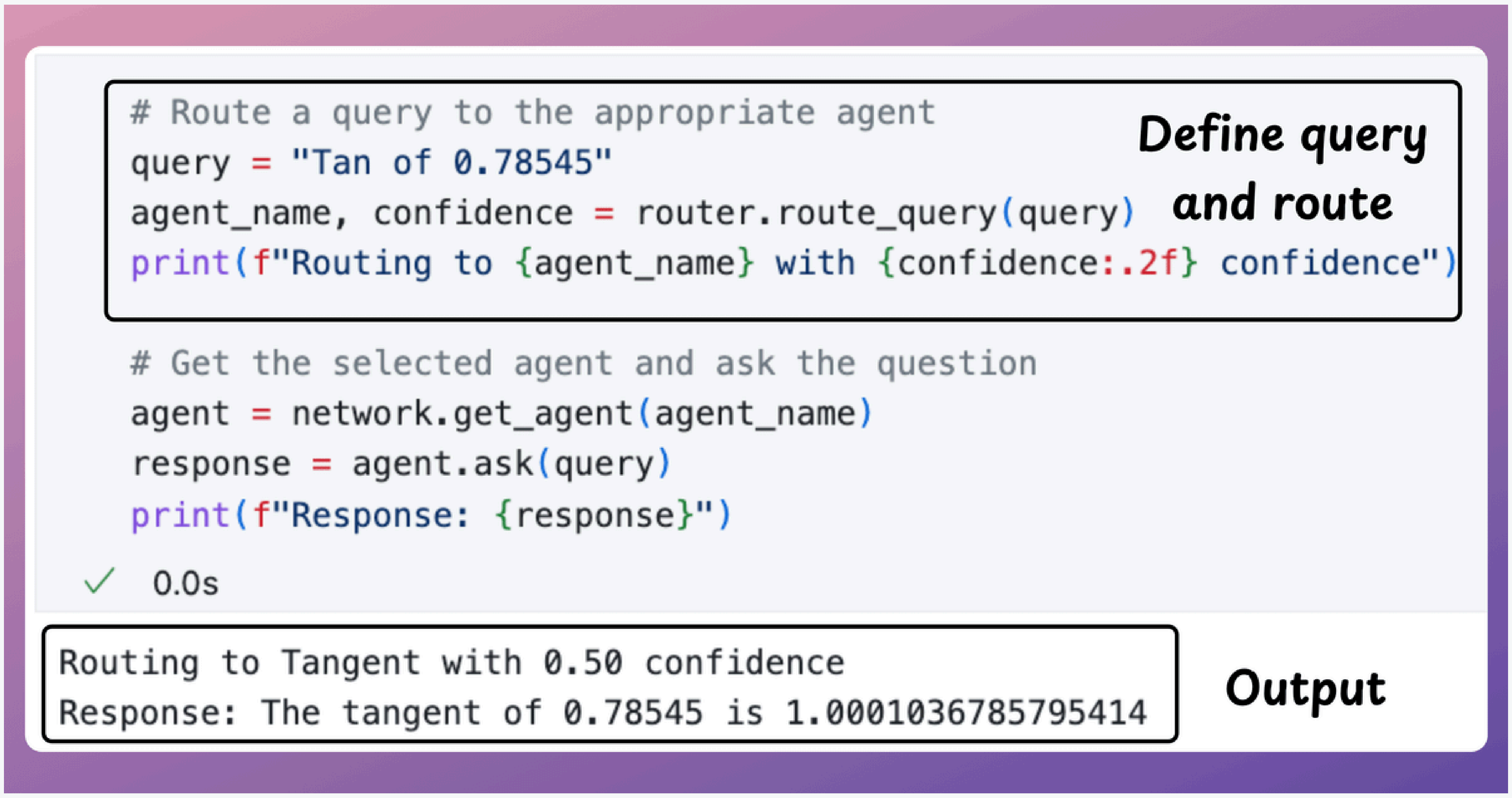
```python
from python_a2a import AgentNetwork, A2AClient, AIAgentRouter

# Create an agent network
network = AgentNetwork(name="Math Assistant Network")

# Add agents to the network
network.add("Sine", "http://localhost:4737")
network.add("Cosine", "http://localhost:4738")
network.add("Tangent", "http://localhost:4739")

# Create a router to intelligently direct queries to the best agent
router = AIAgentRouter(
    llm_client=A2AClient("http://localhost:5000/openai"),
    agent_network=network
)
```

Querying the agent network

With that, we can send a query to our network and see if it is being routed to the appropriate Agent.

Below, we have a query to compute the Tangent, and our router directed it correctly.

Check this code👇



```python
# Route a query to the appropriate agent
query = "Tan of 0.78545"
agent_name, confidence = router.route_query(query)
print(f"Routing to {agent_name} with {confidence:.2f} confidence")

# Get the selected agent and ask the question
agent = network.get_agent(agent_name)
response = agent.ask(query)
print(f"Response: {response}")
```

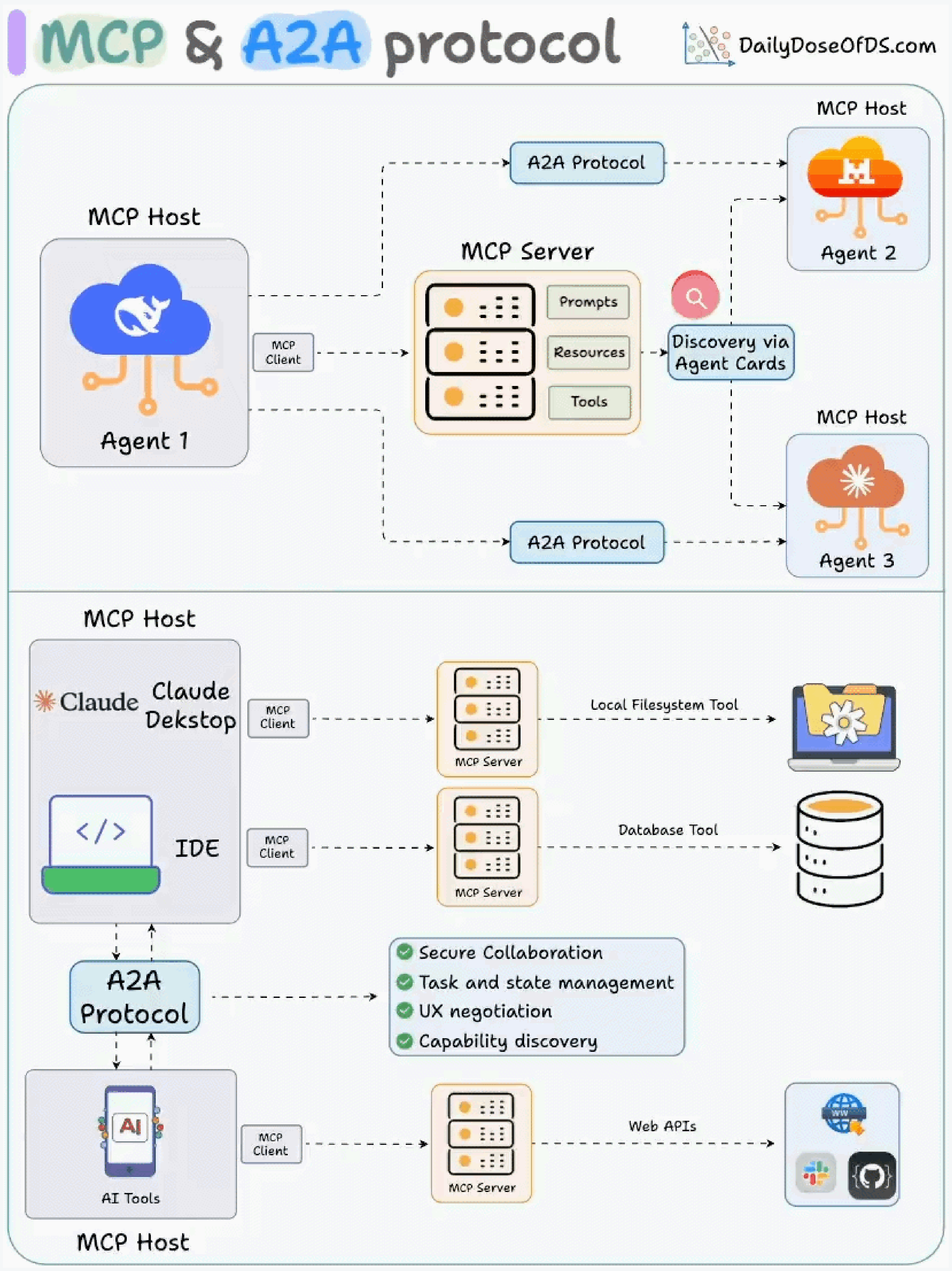
✓ 0.0s

```
Routing to Tangent with 0.50 confidence
Response: The tangent of 0.78545 is 1.0001036785795414
```

So, that's how you can build and deploy your own Agent network using A2A protocol.

To recap:
- MCP connects agents to external tools
- A2A enables agent to agent collaboration

I hope you enjoyed this tutorial! 🥂



Thank you

If you found it insightful, reshare with your network.

Find me → @akshay_pachaar ✓

For more insights and tutorials on LLMs, AI Agents, and Machine Learning!