

Agent2Agent A Visual Guide

Sample Book — Chapters 1 to 2

A Beginner-Friendly Introduction to Interoperable AI Agents for
Systems Engineers

Protocol baseline: A2A Protocol 1.0

Reiner Braun

Introduction

This book is designed for readers who are taking their first steps with Agent2Agent, including systems engineers who want to build the right foundation from the start.

Rather than asking you to memorize more than you need, the book focuses on the essential concepts that will support your future study and day-to-day work. More advanced topics are included only when they are likely to become useful to many readers, and even then they are introduced only far enough to explain their purpose, context, and overall role.

Processes, relationships, and structures that can be difficult to picture from text alone are supported by diagrams. By moving between explanations, visuals, and examples, you can build a clearer and more practical understanding of the subject.

Product names, service names, logos, icons, and other identifying marks mentioned in this book are trademarks or registered trademarks of their respective owners. This book is not an official publication, certified training resource, or endorsed learning material of any company or organization.

How to Use This Book

Read the chapters in order on your first pass. The sequence moves from the purpose of A2A and the minimum technical background to the protocol model, implementation patterns, security, operations, interoperability, and architecture. This order lets each chapter build on concepts already established.

Before reading a chapter, review its **What You'll Be Able to Do** section. It identifies the practical capability that the chapter is intended to build and helps you separate essential material from supporting detail.

Use the diagrams to understand relationships, processes, boundaries, state changes, and data movement that are difficult to picture from prose alone. The diagrams support the written explanation; they do not replace it.

Run the code and hands-on examples yourself whenever your environment permits. Observe the requests, identifiers, Task states, errors, and Artifacts rather than treating the examples as text to memorize.

At the end of each chapter, use the **Chapter Summary** and **Review Questions** to check whether you can explain the fundamentals without looking back. Revisit the relevant section when an answer depends on memorized wording rather than a clear mental model.

Advanced-topic overviews do not need to be memorized at this stage. Treat them as signposts that show where production concerns and later study connect to the foundation developed in this book.

Table of Contents

Introduction

How to Use This Book

Part I — Foundation and Positioning

Chapter 1 — A2A in One Picture

- The integration problem A2A addresses
- Independent and opaque agents
- The essential protocol objects
- A complete exchange from start to finish
- Recurring example: patch-readiness assessment
- Where A2A is useful
- What A2A is not
- The central mental model

Chapter 2 — The Minimum Technical Foundation

- Model, agent, tool, service, and orchestrator
- Client and server communication
- HTTP, HTTPS, and TLS
- JSON and machine-readable contracts
- Media types and content modes
- Synchronous and asynchronous work
- Authentication and authorization
- Stateful and stateless operations
- Why long-running work needs a lifecycle
- Foundation applied to the patch-readiness example

Part I — Foundation and Positioning

Chapter 1 — A2A in One Picture

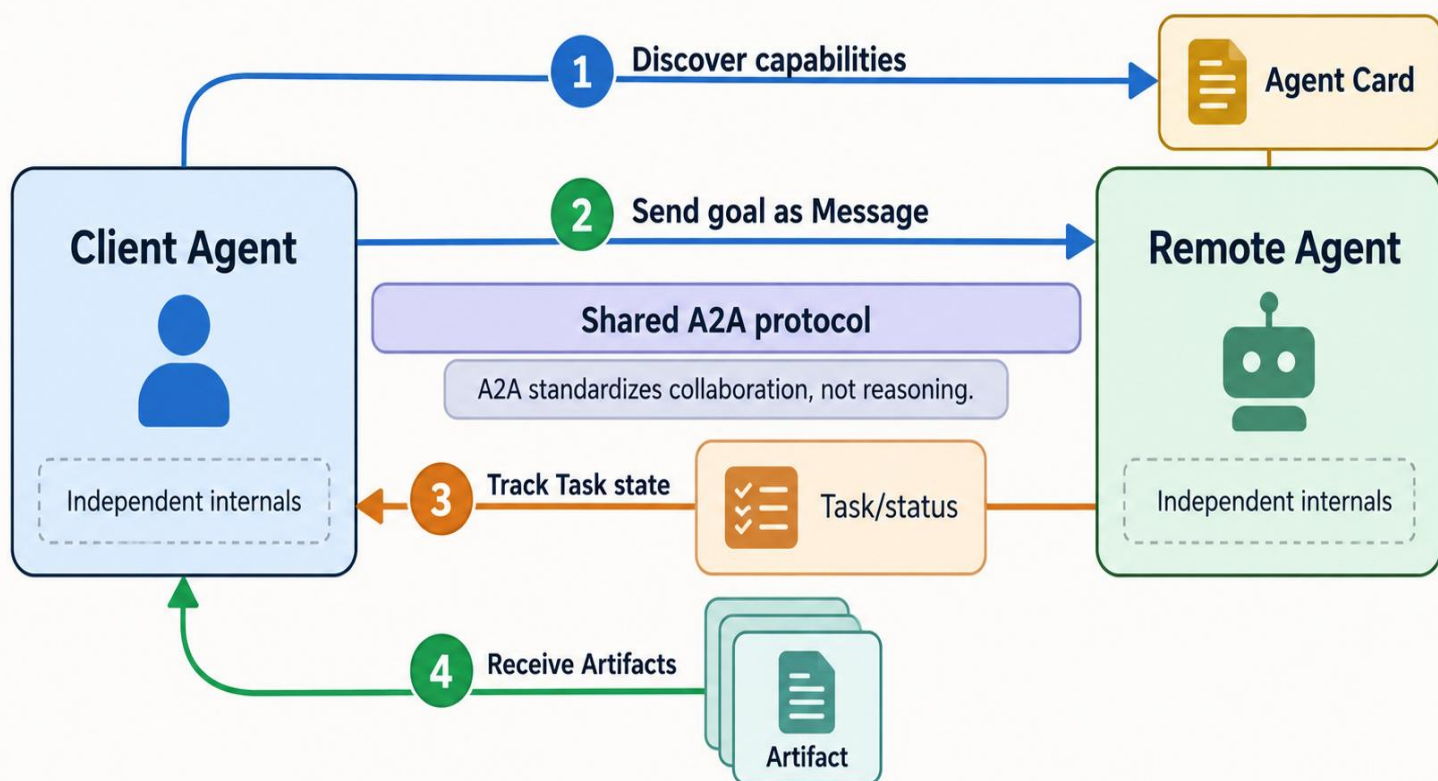
What You'll Be Able to Do

After completing this chapter, you will be able to state in plain language why A2A exists and trace a basic interaction from discovery to delivered results.

Agent2Agent, usually shortened to A2A, is a standard way for independent AI agents to discover one another, exchange work, and track the results without exposing how either agent is built internally. It is most useful when one system needs to delegate a meaningful task to another system that has its own logic, tools, state, and operational owner.

Chapter 1: A2A in One Picture

1-0 A2A big-picture collaboration loop



001

Figure 1-1: A2A big-picture collaboration loop

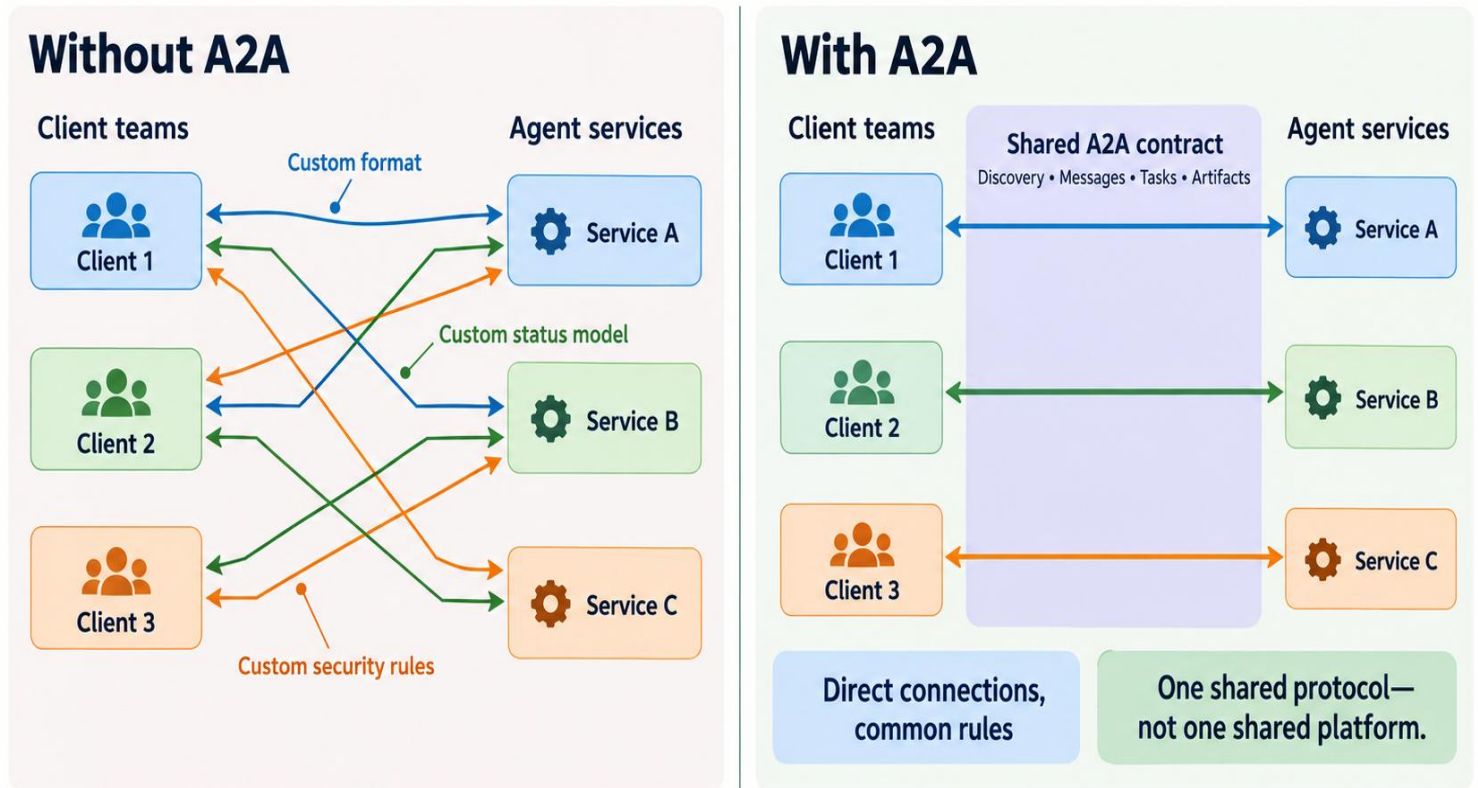
An **AI agent** is a software system that can interpret a goal, decide how to pursue it, use tools or services, and return a result. A2A does not define how an agent reasons or which model, framework, or tools it uses. Instead, it defines the boundary between agents: how one agent learns what another can do, sends it a request, follows the work, and receives the deliverables.

The integration problem A2A addresses

Organizations rarely build every agent with the same framework, programming language, deployment platform, or vendor product. One team may operate a Python-based operations agent, while another team exposes a Java service backed by a workflow engine. A partner company may use a completely different model and cloud platform.

Chapter 1: A2A in One Picture

1-1 From bespoke agent integrations to one shared protocol boundary



002

Figure 1-2: From bespoke agent integrations to one shared protocol boundary

Without a shared protocol, every connection tends to become a custom integration. The teams must agree on network addresses, request formats, status handling, file transfer, identity checks, long-running work, and error behavior. The first integration may be manageable, but the number of custom agreements grows quickly as more agents are added.

A2A provides a common interaction model for this boundary. It standardizes several questions that otherwise need a project-specific answer:

- What capabilities does the remote agent advertise?
- Which protocol interface and version can the client use?
- Which content types can each side send and receive?
- Does the request produce an immediate answer or a tracked unit of work?
- How does the client observe progress and retrieve final deliverables?
- How can related interactions be associated without exposing internal memory?

The protocol therefore acts like a **common service desk form and tracking system**. A requester does not need to know which internal teams, scripts, databases, or decision rules the service desk uses. The requester needs to know what service is offered, what information to provide, how to identify the ticket, and how to obtain the result.

Independent and opaque agents

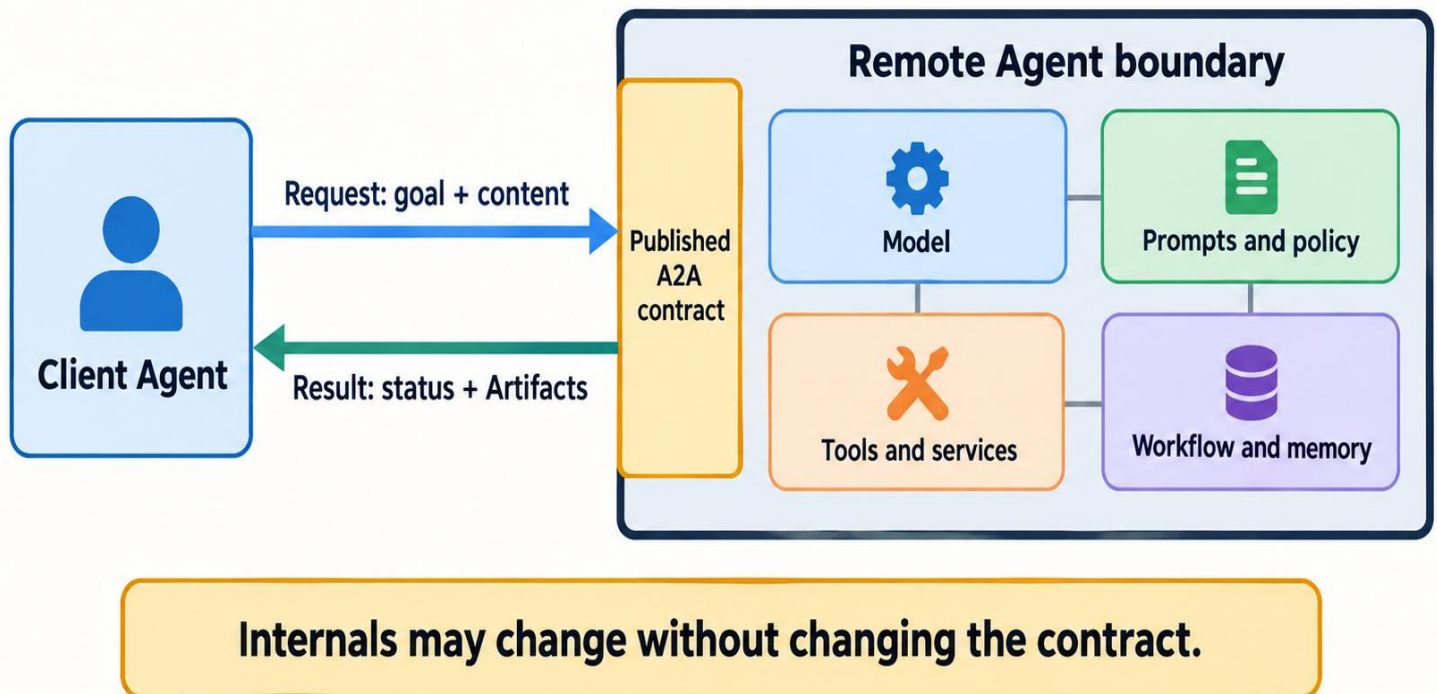
A remote A2A agent is often described as **opaque**, meaning that callers use its published contract but do not depend on its internal model, prompts, memory, tools, or workflow. Opaque does not mean mysterious or unaccountable. The agent still publishes what it can do, which interfaces it supports, which content formats it accepts, and which security requirements apply. Its implementation simply remains behind its service boundary.

This separation is important for systems engineering. A remote team can change its model, replace an internal tool, or reorganize its workflow without forcing every caller to change, provided that the A2A contract remains compatible. The caller can likewise change its own orchestration logic without requiring the remote agent to understand it.

A2A standardizes collaboration, not internal intelligence.

Chapter 1: A2A in One Picture

1-2 Opaque agent boundary



003

Figure 1-3: Opaque agent boundary

The essential protocol objects

A complete A2A interaction uses a small set of concepts. These definitions are intentionally brief here; later chapters explain each one in detail.

An **Agent Card** is a machine-readable description of an agent's identity, skills, supported interfaces, content modes, optional capabilities, and security requirements.

A **Message** is one communication turn sent by the client or the remote agent.

A **Part** is one item of content inside a Message or Artifact, such as text, structured JSON data, inline bytes, or a file URL. **JSON**, or JavaScript Object Notation, is a text format for structured data, while a **URL**, or Uniform Resource Locator, is an address for a network resource.

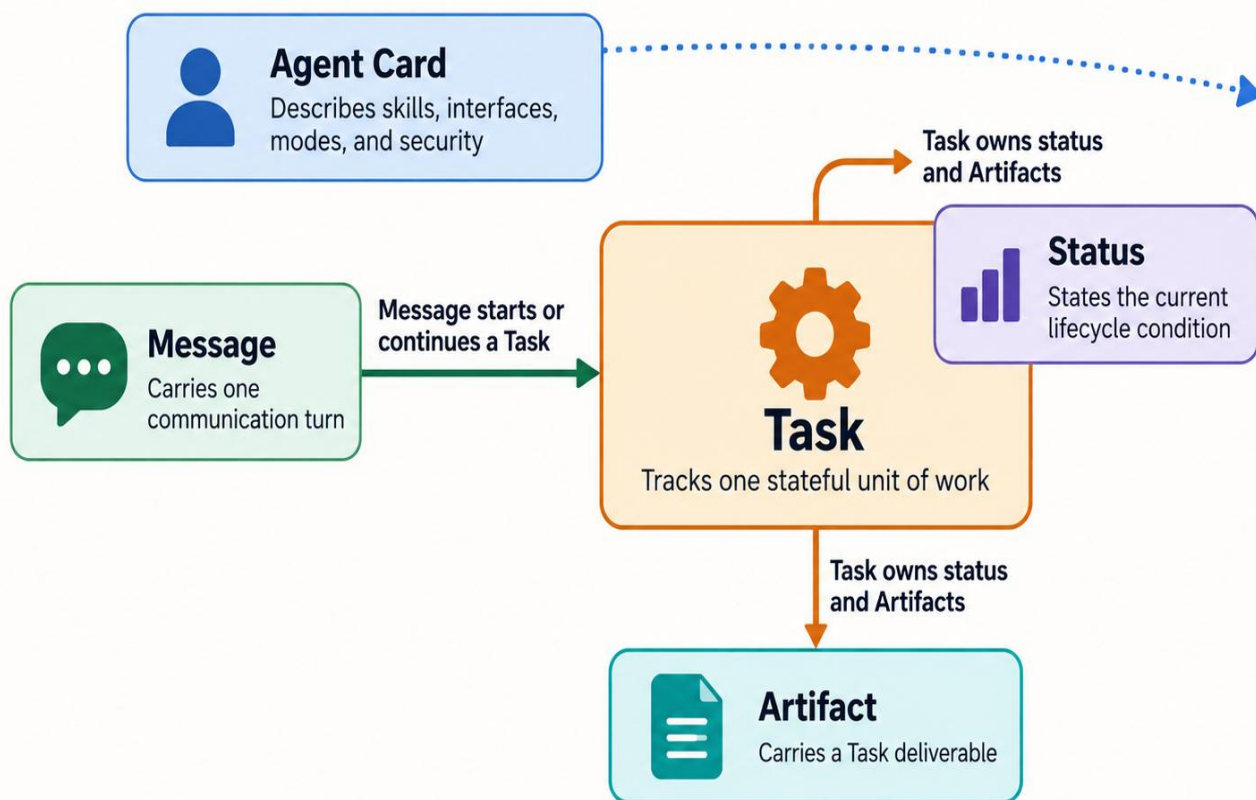
A **Task** is a server-created, stateful unit of work that can move through states such as submitted, working, input required, completed, failed, or canceled. **Stateful** means that information about the work persists across more than one interaction.

An **Artifact** is a deliverable produced by a Task, such as a report, configuration proposal, image, or structured result.

A **context** groups related interactions so that several Tasks and Messages can remain logically connected.

Chapter 1: A2A in One Picture

1-3 Relationships among Agent Card, Message, Task, status, and Artifact



004

Figure 1-4: Relationships among Agent Card, Message, Task, status, and Artifact

These objects separate conversation, work tracking, and deliverables. That separation is one of the most important ideas in A2A. A status message such as “Scanning 40 servers” is not the same thing as the final readiness report, and neither is the same thing as the Task that tracks the work.

A complete exchange from start to finish

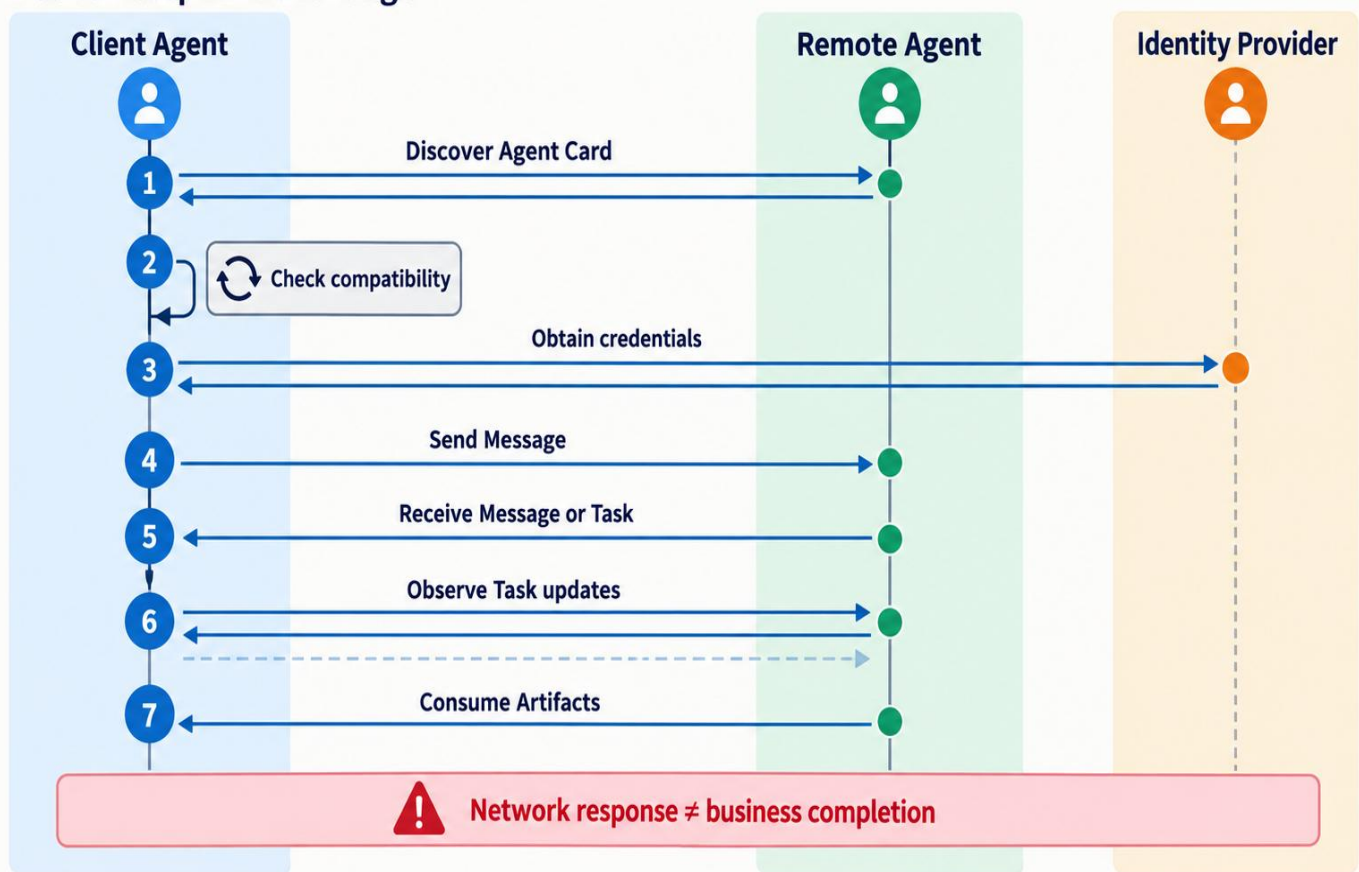
The big picture can be understood as a sequence of seven actions.

- 1. Discover the agent.** The client obtains the remote agent's Agent Card from a well-known location, a registry, or configuration.

2. **Check compatibility.** The client confirms that the card advertises an appropriate skill, protocol binding, protocol version, content mode, and optional capability. A **protocol binding** is the concrete mapping of A2A operations to a network protocol and its request format.
3. **Authenticate when required.** Authentication proves the client's identity. The client obtains credentials—evidence of that identity—through the advertised security mechanism and sends them through the selected network protocol's standard security channel.
4. **Send a Message.** The client describes the requested outcome and supplies the necessary content Parts.
5. **Receive a Message or Task.** A simple interaction may return a direct Message. More substantial work normally returns a Task with a server-generated identifier.
6. **Observe the work.** The client can retrieve the Task state and, when supported, receive incremental updates through streaming, where updates arrive over an open connection, or through notifications.
7. **Consume the Artifacts.** When the Task completes, the client reads and validates the deliverables attached to it.

Chapter 1: A2A in One Picture

1-4 Seven-step A2A exchange



005

Figure 1-5: Seven-step A2A exchange

This sequence is stable even when implementations differ. Clients using different supported protocol bindings may use different network formats, but they work with the same underlying concepts.

Recurring example: patch-readiness assessment

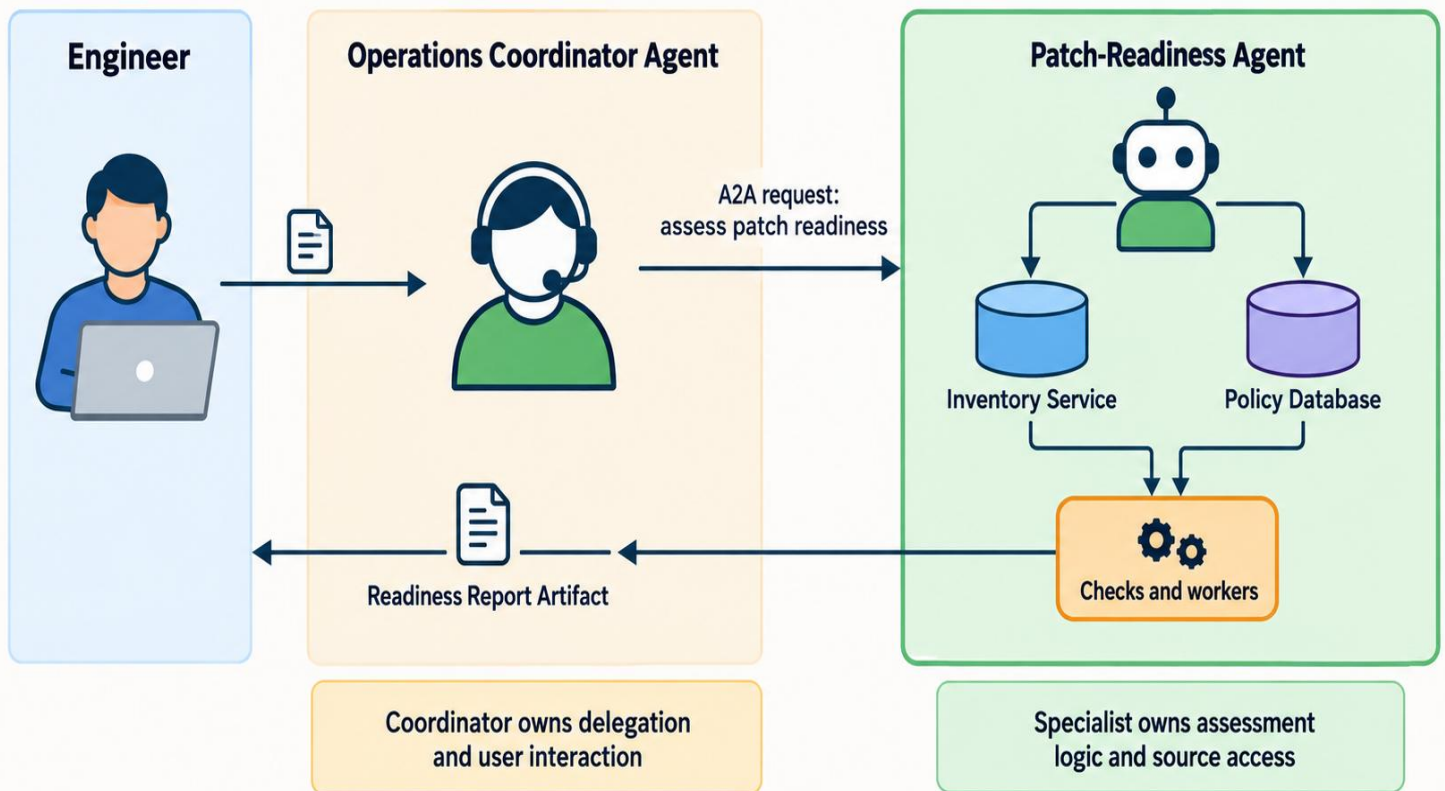
The examples in this guide use two agents operated as separate systems.

The **operations coordinator agent** receives a request from an engineer to assess whether a group of servers is ready for an operating-system patch. It knows which specialist should perform the assessment, but it does not contain the specialist's policy logic.

The **patch-readiness agent** is operated by another team. It can analyze server facts, apply patch policies, identify blockers, and return a structured report. Its implementation may use inventory services, policy databases, scripts, or human approval, but those details are not part of the A2A contract.

Chapter 1: A2A in One Picture

1-5 Patch-readiness example topology and responsibility split



006

Figure 1-6: Patch-readiness example topology and responsibility split

A typical request may contain structured records such as:

```
{
  "servers": [
    {
      "hostname": "web-01",
      "os": "Ubuntu 24.04",
      "freeDiskGb": 18,
      "pendingReboot": false
    },
    {
      "hostname": "db-01",
      "os": "Ubuntu 22.04",
      "freeDiskGb": 4,
      "pendingReboot": true
    }
  ]
}
```

The patch-readiness agent may return a Task because the assessment requires multiple checks. The completed Task may contain an Artifact with machine-readable findings:

```
{
  "ready": ["web-01"],
}
```

```
"blocked": [
  {
    "hostname": "db-01",
    "reasons": ["insufficient free disk", "pending reboot"]
  }
]
```

The coordinator can process this Artifact without knowing how the specialist reached the conclusion. It can display the findings, request clarification, or pass the result to another approved process.

Where A2A is useful

A2A is a strong fit when the remote system is more than a single function call. Common examples include:

- A coordinator delegating a complex task to a specialist owned by another team.
- An enterprise agent requesting analysis from a partner organization's agent.
- A long-running investigation that produces progress updates and several deliverables.
- A task that may pause to request human input or additional authorization.
- A cross-vendor integration where both sides need a portable agent contract.

The common feature is **delegated responsibility**. The remote agent is expected to manage part of the work, not merely execute one primitive operation.

Chapter 1: A2A in One Picture

1-6 Delegated-responsibility spectrum

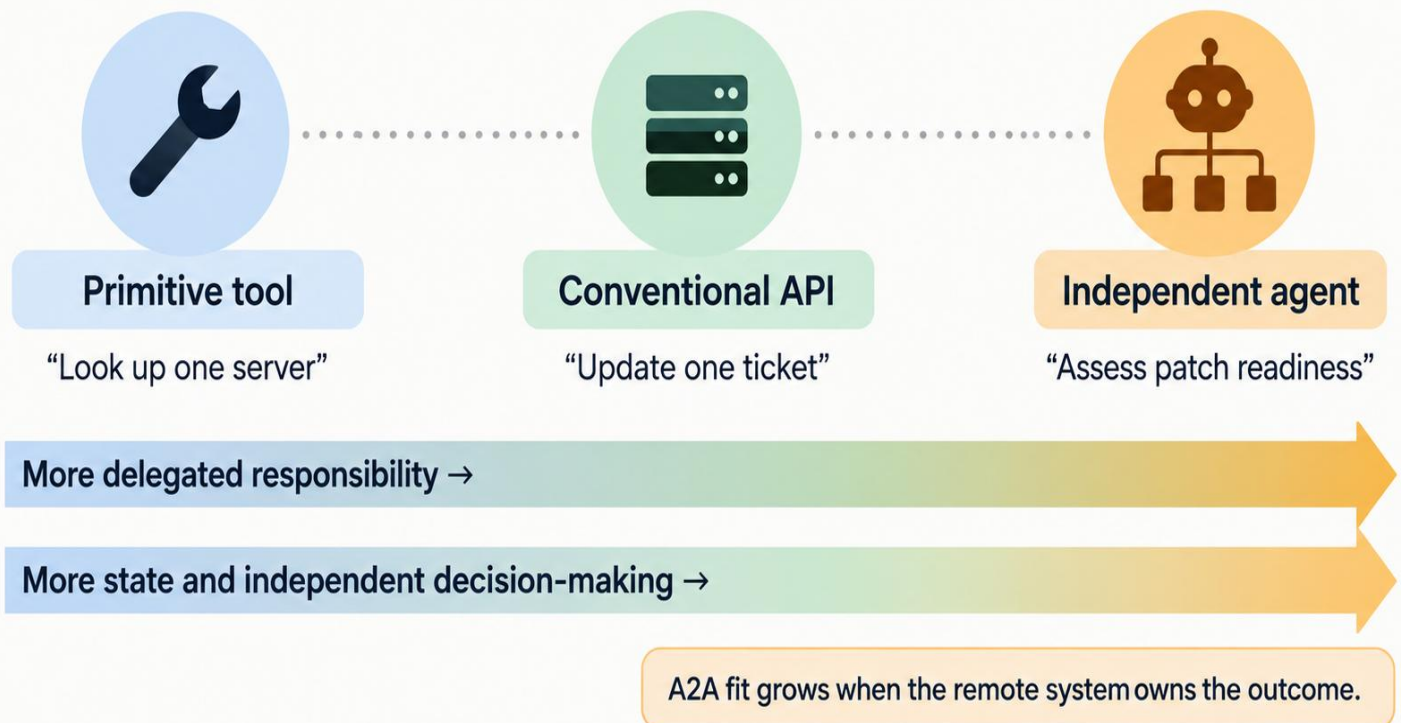


Figure 1-7: Delegated-responsibility spectrum

What A2A is not

A2A is not an agent framework. It does not provide planning algorithms, memory management, prompt templates, or model execution.

An **LLM**, or large language model, is a model trained to process and generate language. An **API**, or application programming interface, is a defined way for software to request capabilities from another system. A2A is not an LLM API and does not standardize model invocation settings or instructions.

A2A is not a general-purpose tool protocol. The **Model Context Protocol (MCP)** is a standard interface for connecting AI applications to tools and resources; an agent may use MCP inside its own boundary.

A2A is not a workflow engine. It represents collaboration across an agent boundary, but it does not prescribe the internal sequence of activities used to complete a Task.

A2A is not a trust decision. An Agent Card can describe an agent, but the card's existence does not prove that the publisher is authorized, reliable, or safe for a particular operation.

A2A is also not required for every integration. A stable calculation, database lookup, or CRUD operation may be clearer and cheaper to expose as a conventional API or tool. **CRUD** means create, read, update, and delete—the four basic operations used for many data resources.

The central mental model

Think of A2A as a contract between two independently operated workers:

- The remote worker publishes the services it can perform.
- The requester chooses a compatible communication route.
- The requester supplies a goal and relevant information.
- The remote worker either answers directly or creates tracked work.
- The requester follows the work through standard states.
- The remote worker returns named deliverables.

The protocol boundary remains stable while each agent's internal implementation can evolve independently. That is the primary value A2A brings to systems engineering.

Chapter Summary

- A2A standardizes collaboration between independently operated agents while allowing each side to keep its internal implementation private.
- Agent Cards support discovery; Messages carry communication; Tasks track work; states report progress; and Artifacts carry deliverables.
- A2A is an integration protocol, not an agent framework, model API, workflow engine, tool protocol, or automatic trust decision.

Review Questions

1. What problem does A2A solve between independently operated agents?
2. How do a Message, a Task, and an Artifact differ?
3. Name one integration for which A2A would add unnecessary complexity.

Chapter 2 — The Minimum Technical Foundation

What You'll Be Able to Do

After completing this chapter, you will be able to use the minimum web-service, security, and state concepts needed to reason about A2A correctly.

A2A is easiest to understand as a web-service protocol with an additional model for agent capabilities, long-running Tasks, conversations, and deliverables. A systems engineer does not need deep machine-learning knowledge to use it, but must be comfortable with several basic software and networking concepts.

Model, agent, tool, service, and orchestrator

These terms are often used loosely, which makes architecture discussions confusing. Their roles are different.

An **AI model** is a computational component that produces outputs from inputs, such as a language model that generates text. A model does not automatically have tools, durable task state, permissions, or responsibility for completing a business outcome.

An **agent** is an application that uses models, deterministic logic, tools, and state to pursue a goal. The agent is the operational system that receives a request and decides how to handle it.

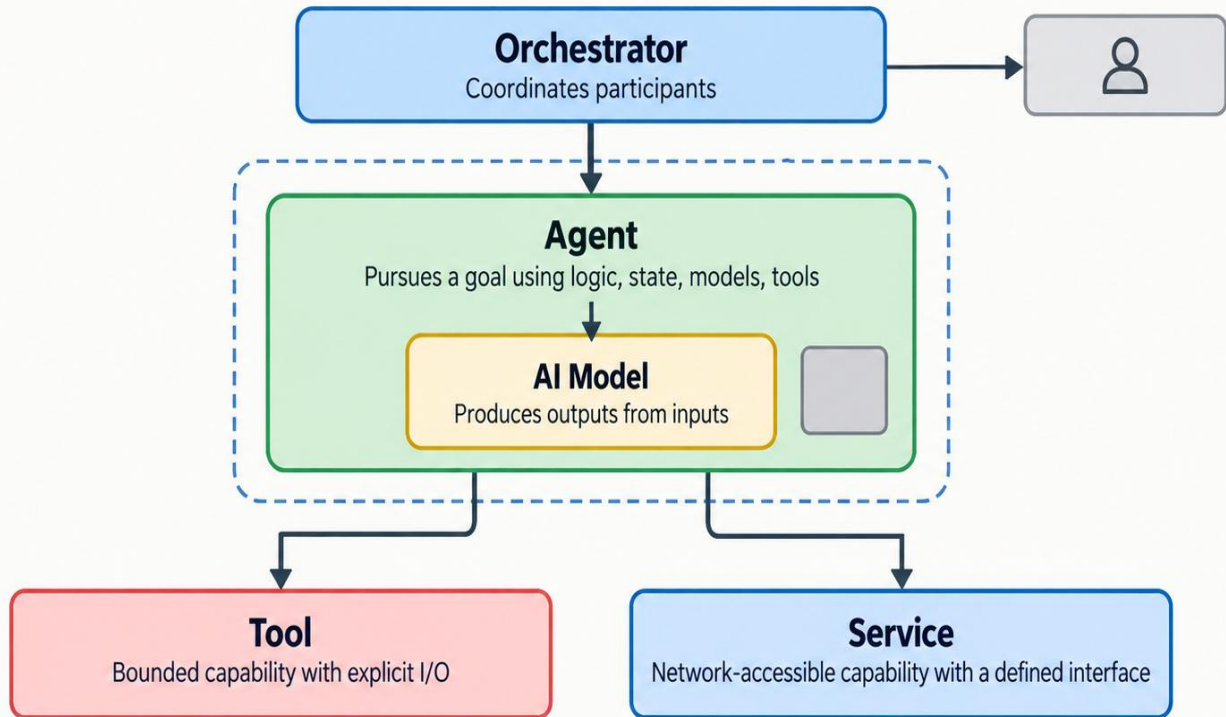
A **tool** is a bounded capability an agent can invoke, such as looking up a server record, calculating a value, or creating a ticket. Tools normally have explicit inputs and outputs and do not independently negotiate a broad goal.

A **service** is a network-accessible software capability with a defined interface. An A2A remote agent is a service, but not every service is an agent.

An **orchestrator** coordinates several steps or participants. It may be an agent that makes dynamic decisions, a workflow engine that follows an explicit process, or ordinary application code.

Chapter 2: The Minimum Technical Foundation

2-1 Model, agent, tool, service, and orchestrator role map



An A2A remote agent is a service; not every service is an agent.

008

Figure 2-1: Model, agent, tool, service, and orchestrator role map

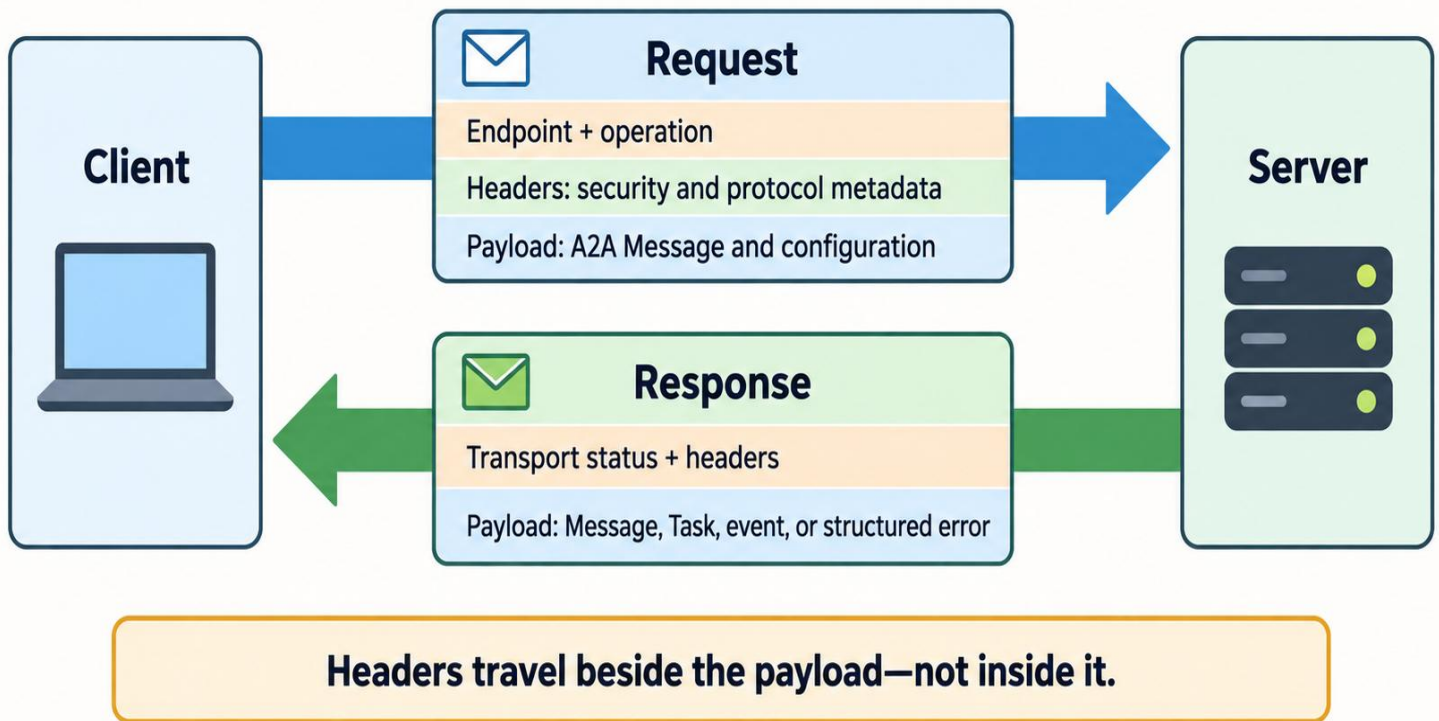
In the patch-readiness example, a language model might interpret a natural-language request. The operations coordinator is the agent. An inventory lookup is a tool or conventional service. The patch-readiness system is a remote agent because it accepts responsibility for a broader assessment. The coordinator acts as an orchestrator when it delegates work and combines results.

Client and server communication

A **client** initiates a network interaction, while a **server** listens for requests and returns responses. In A2A, the system initiating an operation is the A2A client, and the remote agent exposing the protocol is the A2A server.

Chapter 2: The Minimum Technical Foundation

2-2 Client-server request and response anatomy



009

Figure 2-2: Client-server request and response anatomy

An **endpoint** is a network address at which a particular operation or service is available. For an HTTP-based interface, an endpoint is normally represented by a URL.

A request commonly contains:

- A target endpoint.
- An operation or HTTP method.
- Headers carrying protocol and security information.
- A payload carrying the A2A data.

A response commonly contains:

- A success or error status.
- Headers describing the response.
- A payload containing a Message, Task, event, or structured error.

A **header** is metadata transported alongside a request or response. For example, an authorization credential and the requested A2A protocol version can be carried in headers without becoming part of the agent's conversational content.

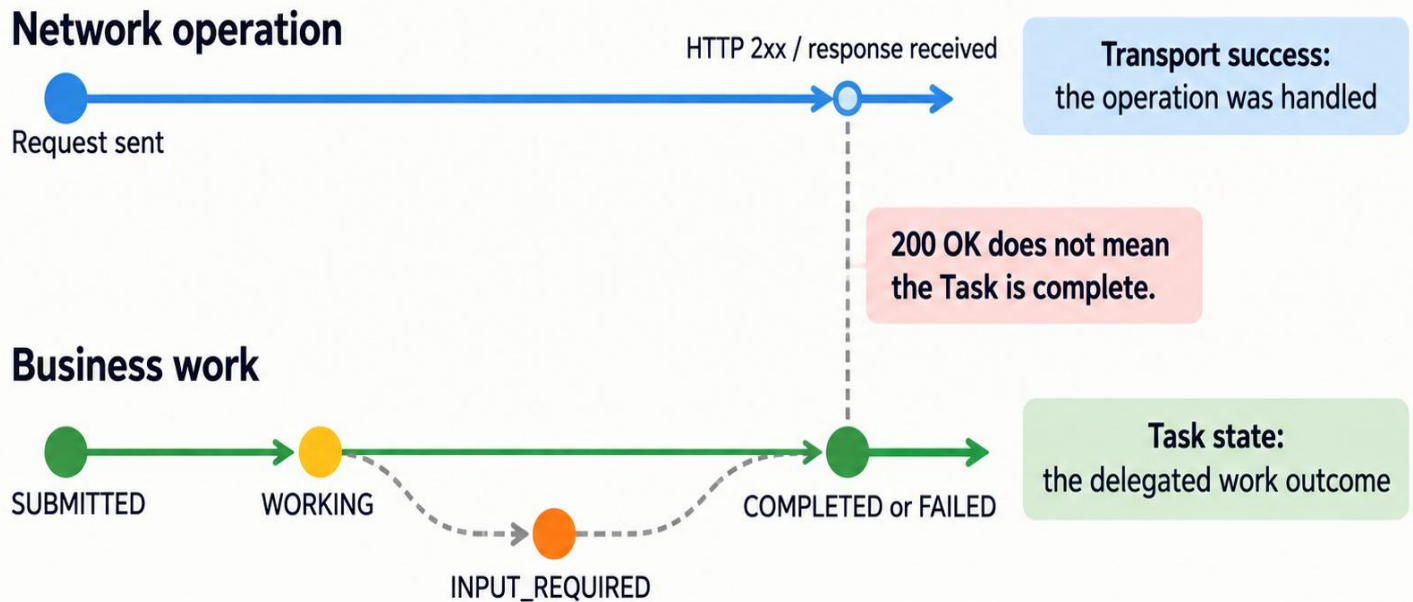
A **payload** is the main data carried by the request or response. In an A2A Send Message operation, the payload includes the Message and may include request configuration.

A **status code** summarizes the transport-level outcome. In HTTP, a [2xx](#) status means that the HTTP request was handled successfully. It does not necessarily mean that the agent's Task completed successfully. The response may contain a Task that is still working, waiting for input, or already failed.

This distinction is fundamental:

Chapter 2: The Minimum Technical Foundation

2-2 Transport success versus Task progress



010

Figure 2-3: Transport success versus Task progress

Transport success describes the network operation; Task state describes the delegated work.

HTTP, HTTPS, and TLS

HTTP is a request-response protocol widely used for web APIs. A client sends a request to a server, and the server returns a response.

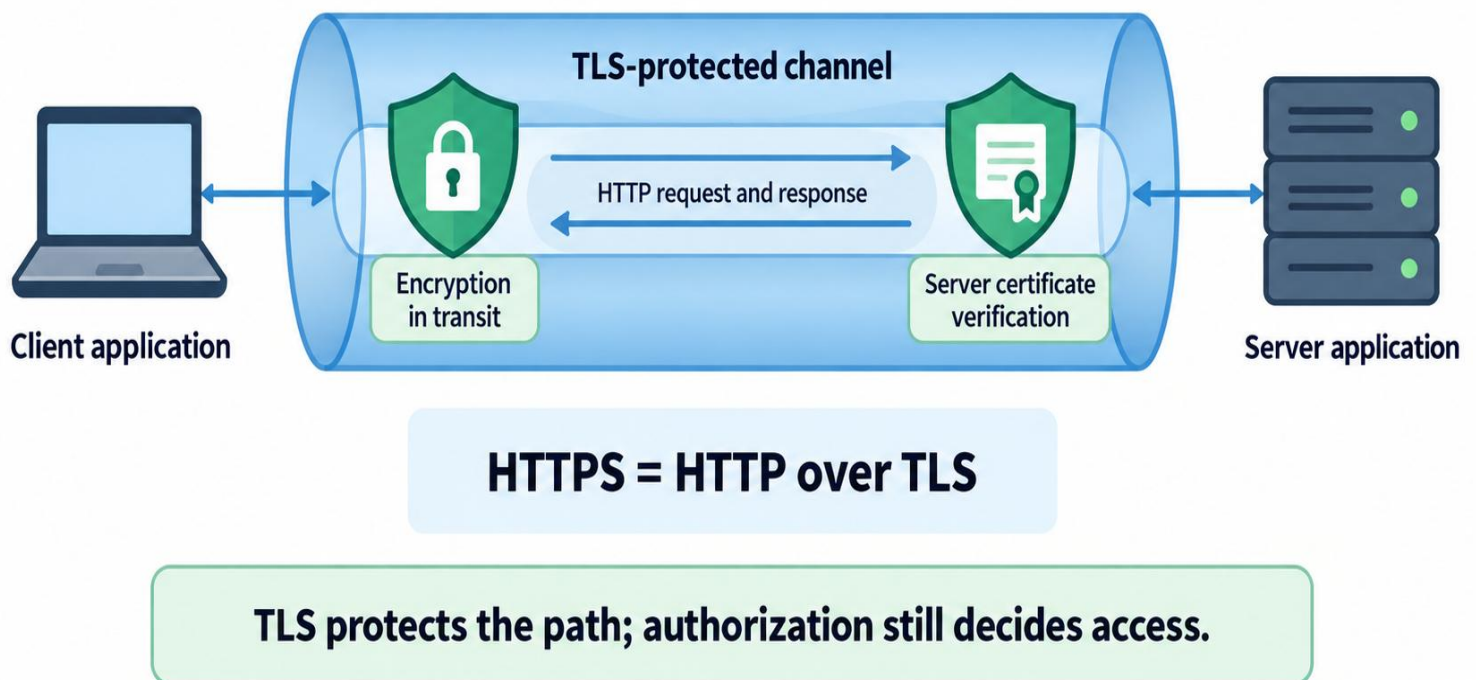
HTTPS is HTTP protected by TLS. **TLS**, or Transport Layer Security, encrypts data in transit and allows the client to verify the server's certificate and network identity.

For a practical analogy, ordinary HTTP is like sending a readable postcard, while HTTPS is like using a sealed, tamper-evident courier channel and checking that the courier reached the intended office. The analogy is imperfect, but it captures the two main purposes: confidentiality in transit and server identity verification.

A2A reuses standard web security mechanisms rather than inventing a separate encryption system. Security policy, credential handling, and authorization remain implementation responsibilities even when the transport is protected.

Chapter 2: The Minimum Technical Foundation

2-3 HTTP protected by TLS



011

Figure 2-4: HTTP protected by TLS

JSON and machine-readable contracts

JSON, or JavaScript Object Notation, is a text format for objects, arrays, strings, numbers, booleans, and null values. A2A uses JSON representations for HTTP-based exchange and defines a canonical data model from which those representations are derived.

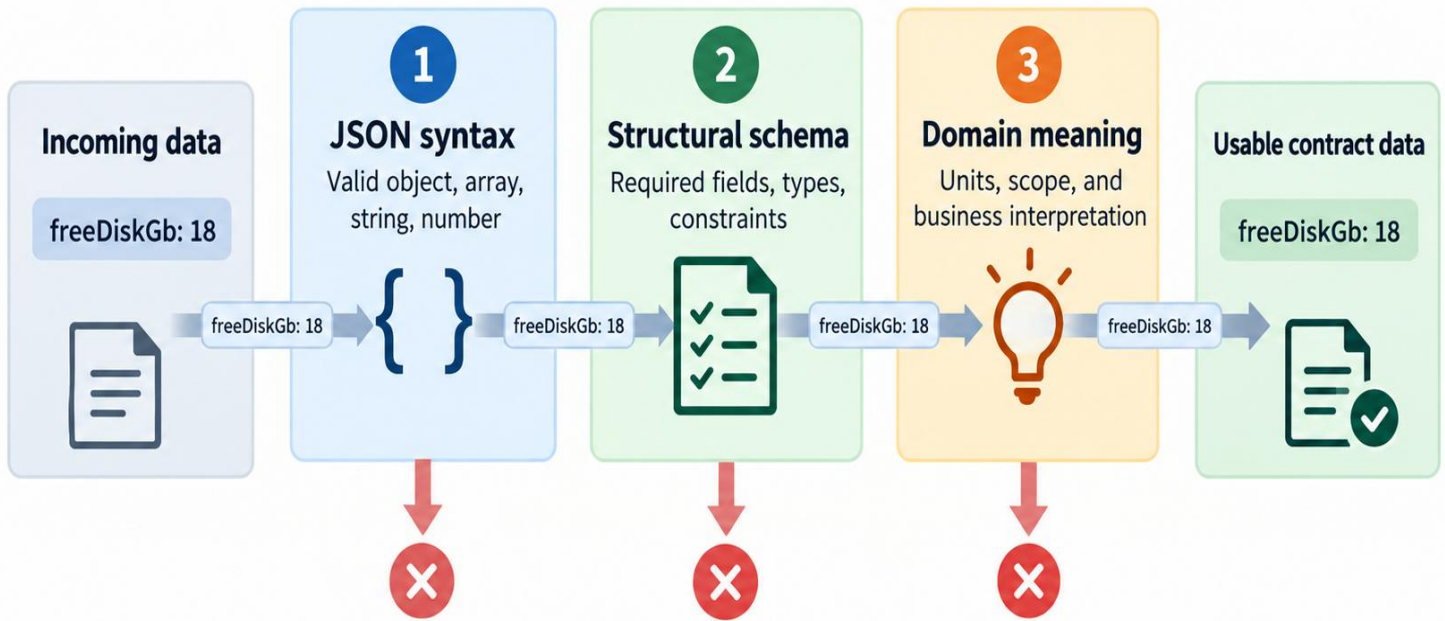
A simple JSON object looks like this:

```
{
  "hostname": "web-01",
  "pendingReboot": false,
  "freeDiskGb": 18
}
```

JSON provides syntax, but syntax alone does not tell a client which fields are required or what they mean. A **schema** is a machine-readable description of a data structure, including field names, types, and constraints. Schemas allow tools and implementations to validate payloads consistently.

Chapter 2: The Minimum Technical Foundation

2-4 Syntax, schema, and domain meaning as three contract layers



Valid JSON can still be structurally or semantically wrong.

012

Figure 2-5: Syntax, schema, and domain meaning as three contract layers

A contract is valuable because independent teams may interpret ordinary prose differently. For example, the field `freeDiskGb` is clearer than an unlabelled number, but the teams must still agree whether decimals are allowed, whether the value refers to a particular filesystem, and what happens when it is unknown. A2A standardizes its core objects; domain-specific structured data still needs an agreed schema between the participating systems.

Media types and content modes

A **media type**, often called a MIME type, identifies the format of content. MIME stands for Multipurpose Internet Mail Extensions, although media types are now used far beyond email. Common examples include:

- `text/plain` for plain text.
- `application/json` for JSON data.
- `application/pdf` for a PDF document.
- `image/png` for a PNG image.

A2A uses media types to describe what an agent accepts and produces. This allows a client to avoid sending a format the remote agent cannot process and to request a result it can consume.

The media type is not merely a filename extension. It is part of the contract. A file named `report.dat` may still contain JSON, but a client should not have to guess. The sender should identify the content correctly.

Synchronous and asynchronous work

A **synchronous** interaction makes the caller wait for a result before continuing. An **asynchronous** interaction allows the work to continue independently while the caller observes or retrieves the result later.

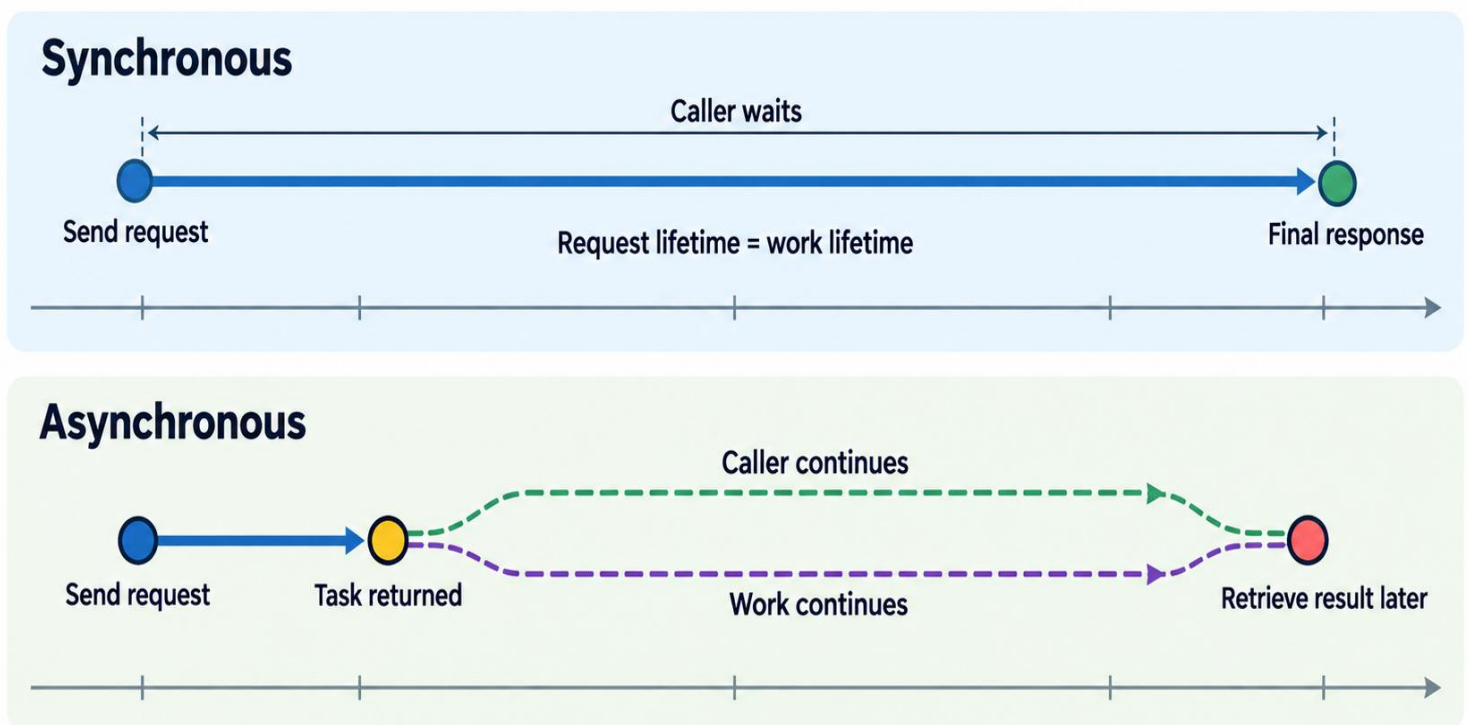
Consider two operations:

1. An inventory lookup retrieves one server record from a fast database and returns it immediately.
2. A patch-readiness assessment checks inventory, maintenance policy, disk capacity, reboot state, and application dependencies for 500 servers.

The first operation is naturally synchronous. The second may take minutes, depend on slow systems, or pause for human input. Holding one ordinary request open until every step is finished can be inefficient and fragile.

Chapter 2: The Minimum Technical Foundation

2-6 Synchronous and asynchronous work timelines



Business-work lifetime can outlive one network request.

013

Figure 2-6: Synchronous and asynchronous work timelines

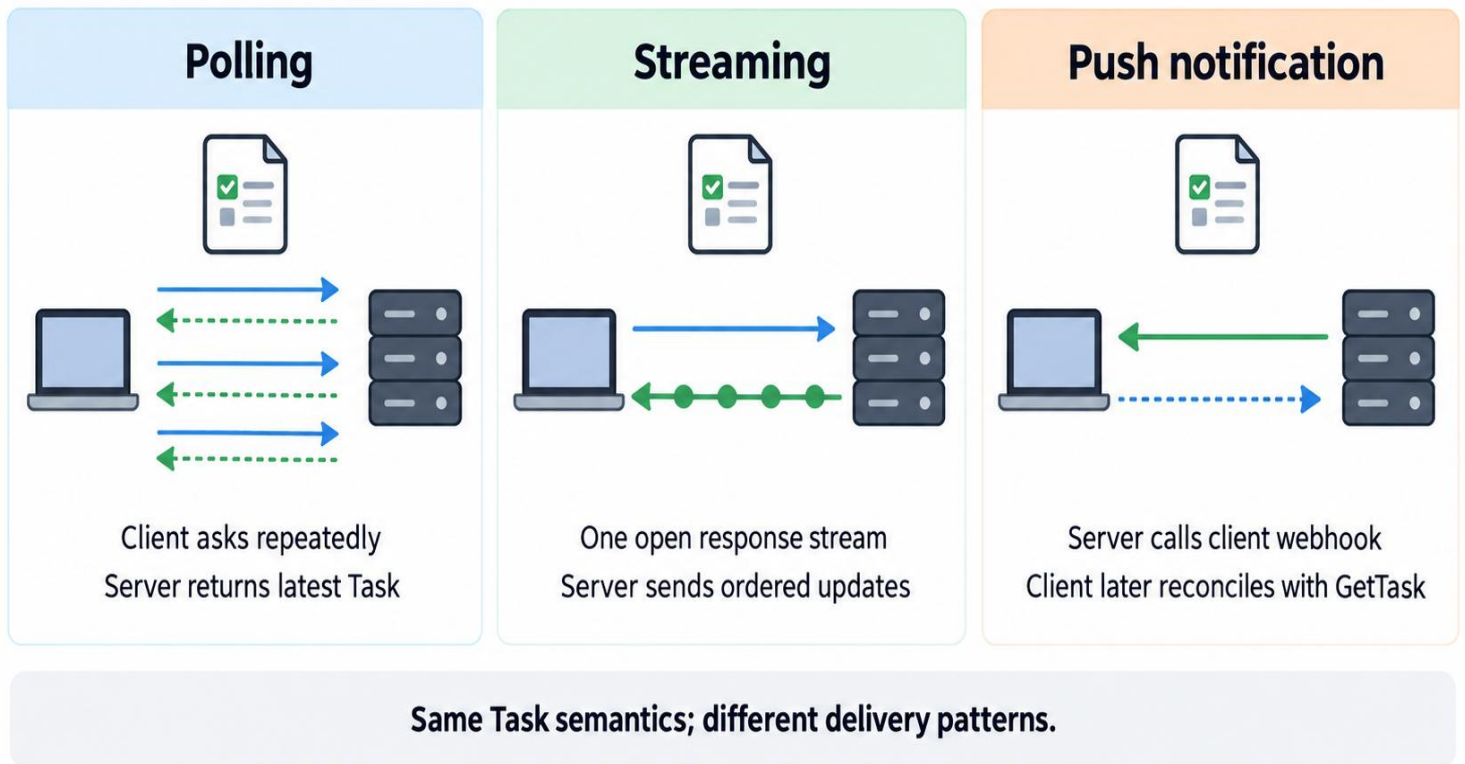
A2A supports several broad interaction patterns:

- **Direct response:** The remote agent returns a Message when no tracked Task is needed.
- **Polling:** The client periodically asks for the latest Task state.
- **Streaming:** The server sends ordered updates over an open connection as they occur.
- **Push notification:** The server sends an HTTP request to a client-provided webhook when an update is available.

A **webhook** is an HTTP endpoint supplied by one system so another system can call it when an event occurs. **Server-Sent Events**, abbreviated SSE, is an HTTP mechanism through which a server sends a sequence of events over a long-lived response connection.

Chapter 2: The Minimum Technical Foundation

2-6 Polling, streaming, and push at a glance



014

Figure 2-7: Polling, streaming, and push at a glance

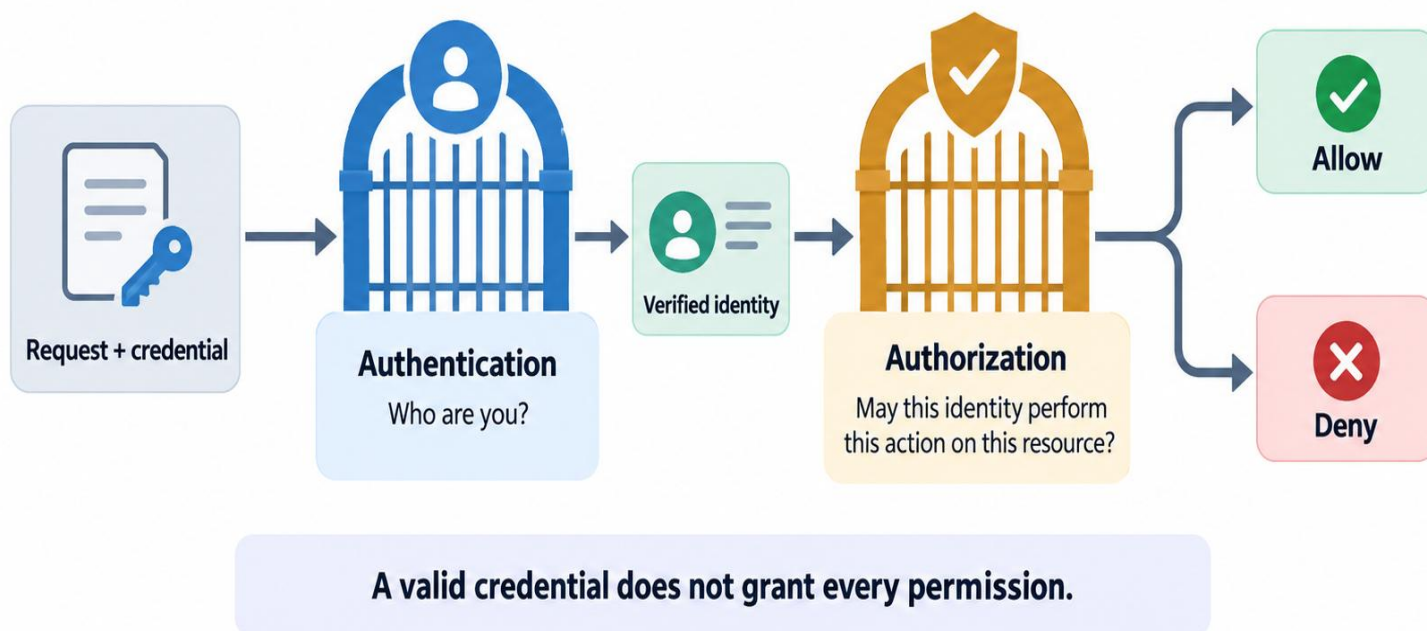
These mechanisms are introduced here only to establish the vocabulary. The important foundation is that **the lifetime of the business work does not have to equal the lifetime of one network request.**

Authentication and authorization

Authentication establishes who or what is making a request. **Authorization** decides what that authenticated identity is permitted to do.

Chapter 2: The Minimum Technical Foundation

2-7 Authentication versus authorization



015

Figure 2-8: Authentication versus authorization

A workplace analogy is useful. Showing a company badge at the entrance authenticates the badge holder. The access-control system then authorizes entry only to approved rooms. A valid badge does not grant access to every laboratory, office, or data center.

The same distinction applies to A2A. **OAuth** is a standard framework that issues scoped access tokens, an **API key** is a credential value assigned to a caller, and a **client certificate** proves a client identity through TLS. A client may present any valid credential and still lack permission to invoke a sensitive skill or read a Task created by another caller.

Credentials normally belong in standard transport mechanisms such as an HTTP [Authorization](#) header. They should not be placed in ordinary Message Parts, because Message content may be stored, logged, copied into Task history, or processed as untrusted business data.

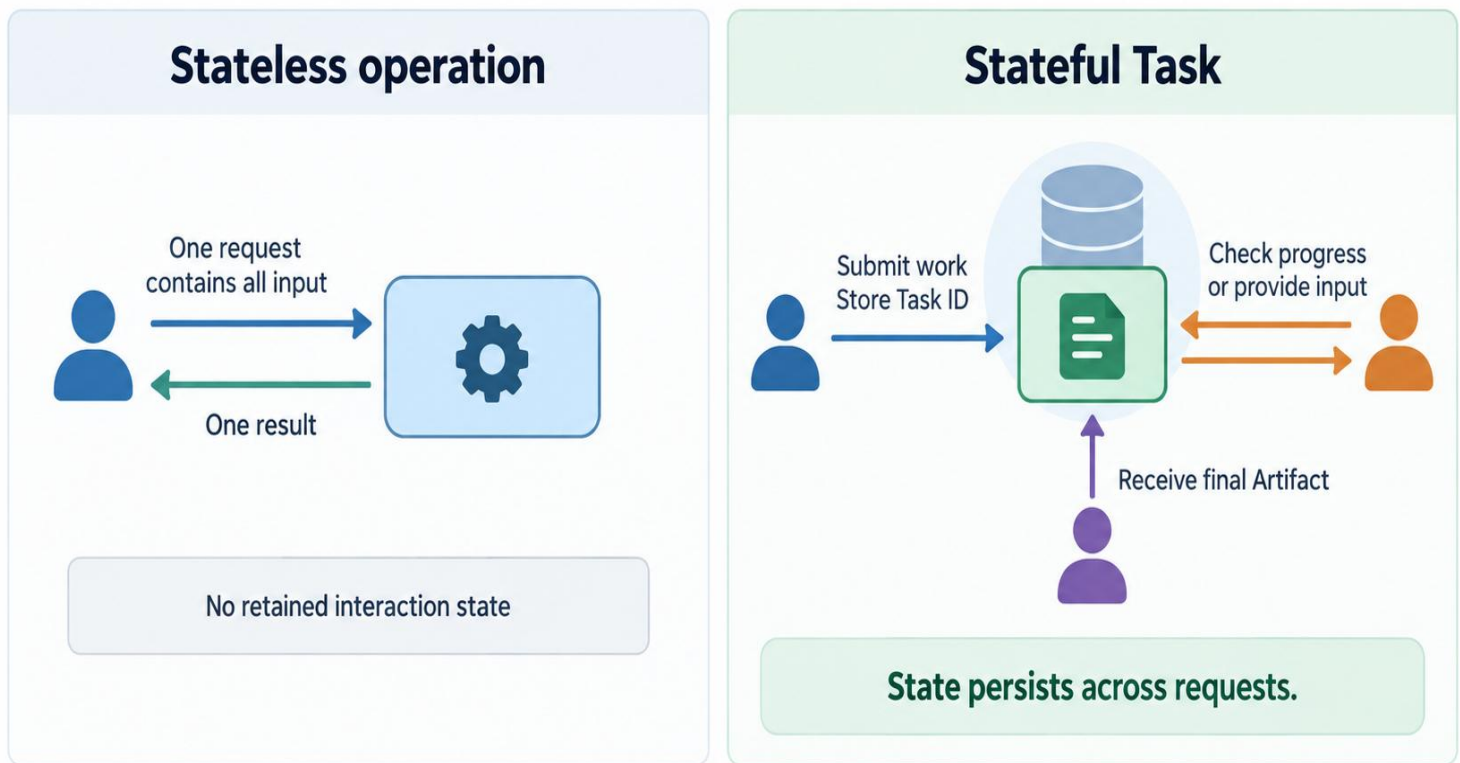
Stateful and stateless operations

A **stateless** operation can be handled using only the information in the current request. A calculator operation such as 12×7 is stateless because the server does not need an earlier interaction to produce 84.

A **stateful** operation depends on information that persists across requests. A patch assessment is stateful when the client submits it, checks its progress later, answers a question, and retrieves the completed report.

Chapter 2: The Minimum Technical Foundation

2-8 Stateless operation versus stateful Task



016

Figure 2-9: Stateless operation versus stateful Task

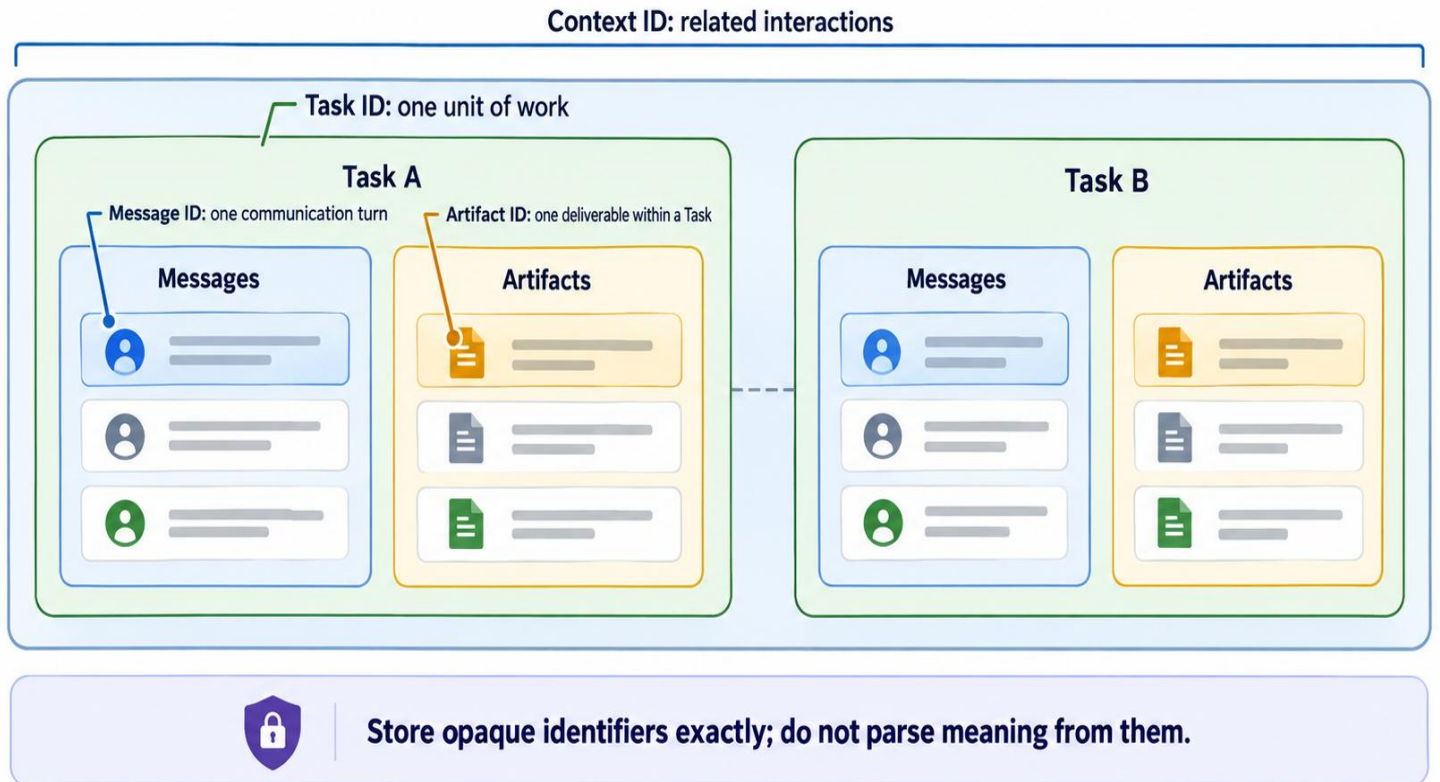
Stateful work requires durable identifiers. A2A uses several identifiers with distinct purposes:

- A Message identifier distinguishes one communication turn.
- A Task identifier identifies one unit of work.
- A context identifier associates related interactions.
- An Artifact identifier distinguishes one deliverable within a Task.

The identifiers should be treated as opaque values. An **opaque identifier** is a value whose internal format has no meaning that a client should rely on. A client should store and return the identifier exactly rather than parsing it to infer dates, users, or routing information.

Chapter 2: The Minimum Technical Foundation

2-8 Identifier scopes across a stateful interaction



017

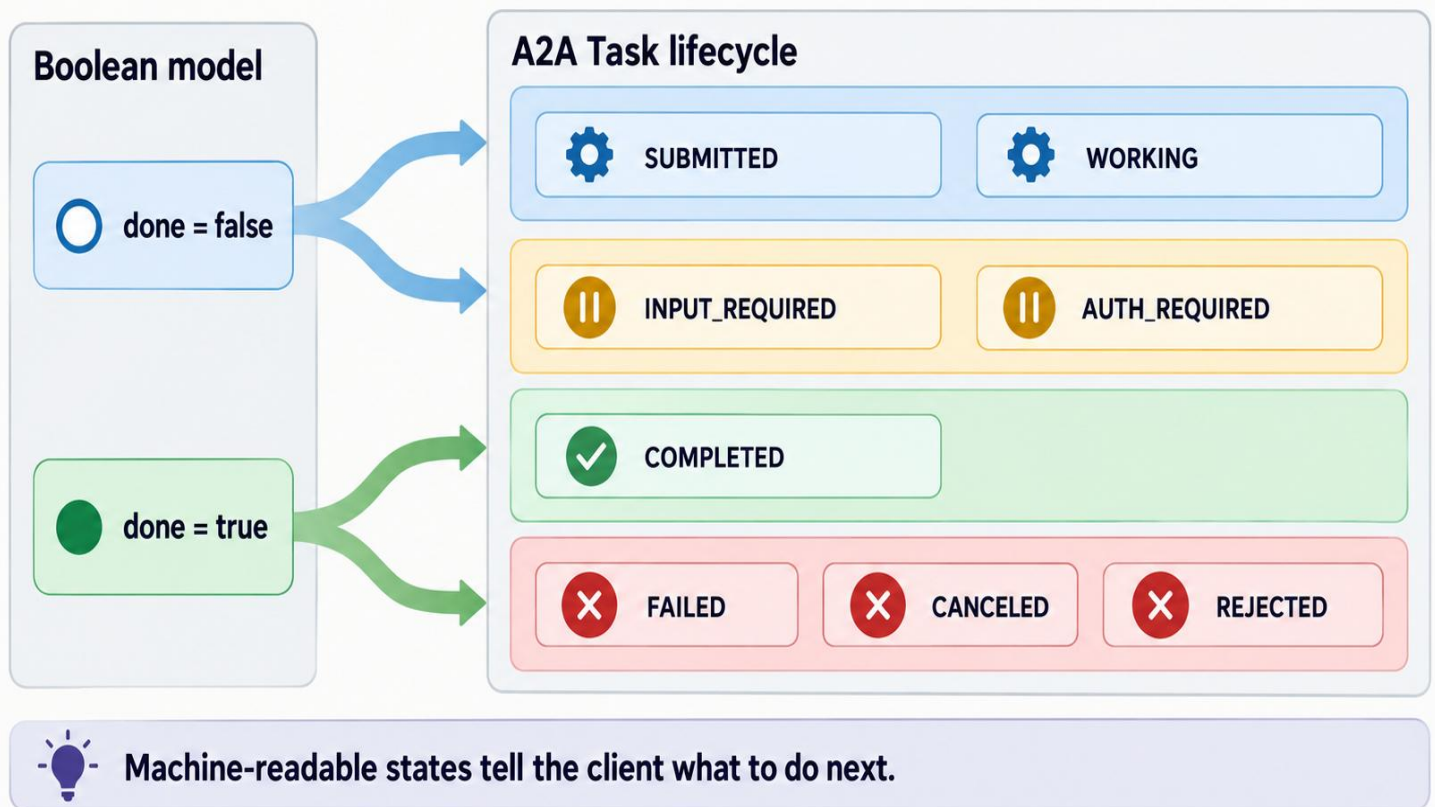
Figure 2-10: Identifier scopes across a stateful interaction

Why long-running work needs a lifecycle

A boolean value such as `done: true` or `done: false` is too weak for realistic delegated work. The client must be able to distinguish several conditions:

Chapter 2: The Minimum Technical Foundation

2-9 Why a boolean completion flag is insufficient



018

Figure 2-11: Why a boolean completion flag is insufficient

- The server accepted the request but has not started it.
- The work is actively progressing.
- The agent needs additional input.
- Additional authorization is required.
- The work completed successfully.
- The work failed.
- The work was canceled or rejected.

A **lifecycle** is the defined set of states and transitions through which an object can move. A2A gives Tasks such a lifecycle so clients do not need to infer progress from free-form text.

This makes automation safer. A coordinator can branch on a Task state rather than searching a status sentence for words such as “finished” or “waiting.” Human-readable messages remain useful, but machine-readable state is the authoritative control signal.

Foundation applied to the patch-readiness example

The inventory lookup and patch assessment can now be classified accurately.

The inventory lookup is a bounded tool or service operation. The caller sends a server identifier and receives a record. A direct API is likely sufficient.

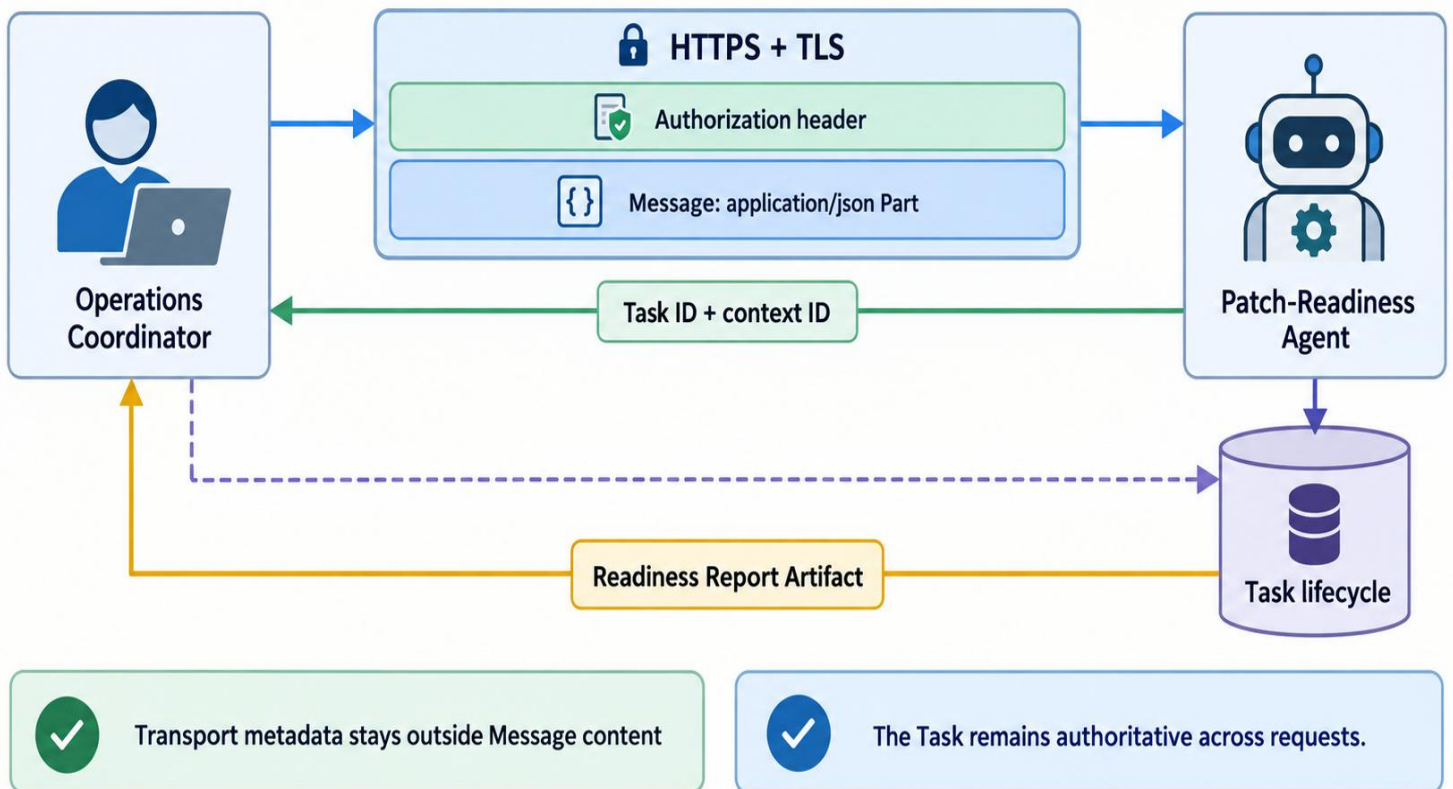
The patch-readiness assessment is delegated work. It may combine several data sources, apply policy, ask which maintenance window applies, and produce more than one deliverable. The remote system owns the assessment logic and may evolve independently. A2A is a reasonable boundary.

The request may use HTTPS, carry an authorization token in a header, and include an A2A Message with an `application/json` Part. The response may contain a Task identifier. The coordinator can later retrieve the Task, interpret its lifecycle state, and consume a structured Artifact.

Once these ordinary web-service concepts are clear, A2A's agent-specific model becomes a small, understandable layer rather than a completely new type of system.

Chapter 2: The Minimum Technical Foundation

2-10 Foundation concepts applied to the patch-readiness interaction



019

Figure 2-12: Foundation concepts applied to the patch-readiness interaction

Chapter Summary

- A2A builds on familiar client-server communication, HTTPS, JSON, media types, credentials, and machine-readable contracts.
- Long-running work is different from a synchronous request because the work needs a durable identifier and lifecycle after the initial response.
- Authentication establishes identity, authorization controls permitted actions, and Task state—not transport success—reports work progress.

Review Questions

1. Why can an HTTP success response coexist with a Task that is still working or later fails?
2. What is the difference between authentication and authorization?
3. Why does long-running work need a durable identifier and lifecycle?