MCP Servers in Digital Analytics

About me

Gunnar Griese
Digital Analytics Consultant

8-bit sheep
Digital Analytics Consultant & Analytics Engineer

- Specialized in Google Analytics, Google Tag Manager, and the Google Cloud Platform

IHM Business School
Teacher “Technical Digital Analytics”

- Responsible for the courses “Technical Digital Analytics” and “Tag Management”

Blog
gunnargriese.com

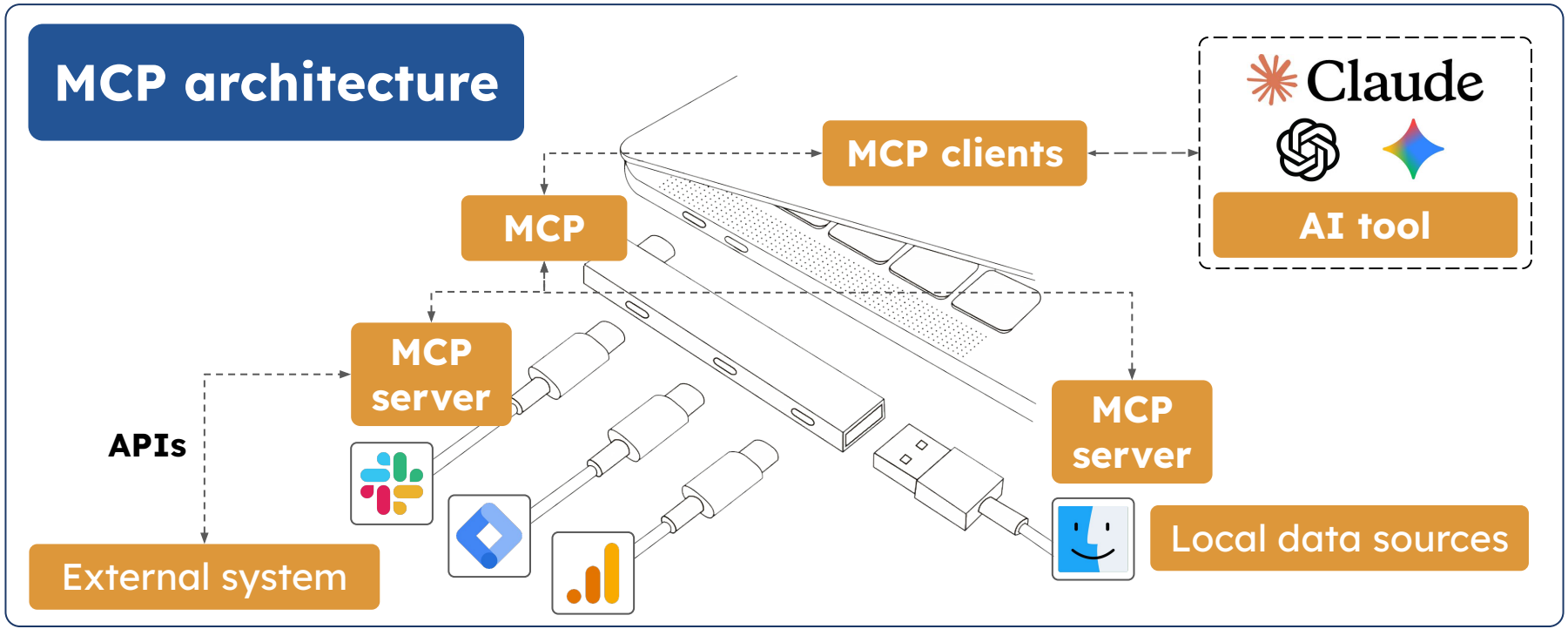
TABLE OF CONTENTS

For most ambitious use cases, AI agents need to do more than just chat — they need to interact with external services to access real-time information, manipulate data, and take actions on behalf of users. These capabilities are what transform a conversational AI into a truly useful agent.

April, 2025: https://www.madrona.com/what-mcps-rise-really-shows-a-tale-of-two-ecosystems/

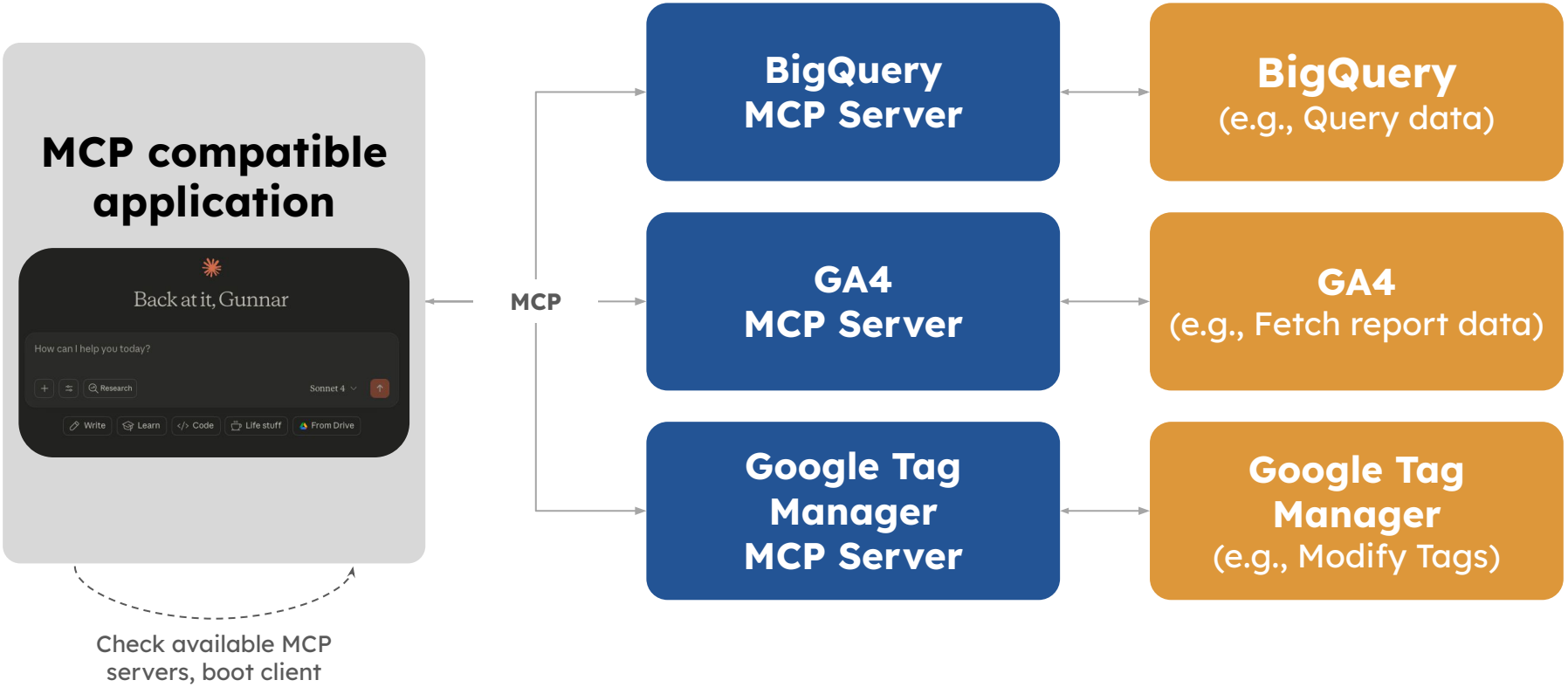
What are MCP Servers?

Model Context Protocol: The Universal Translator



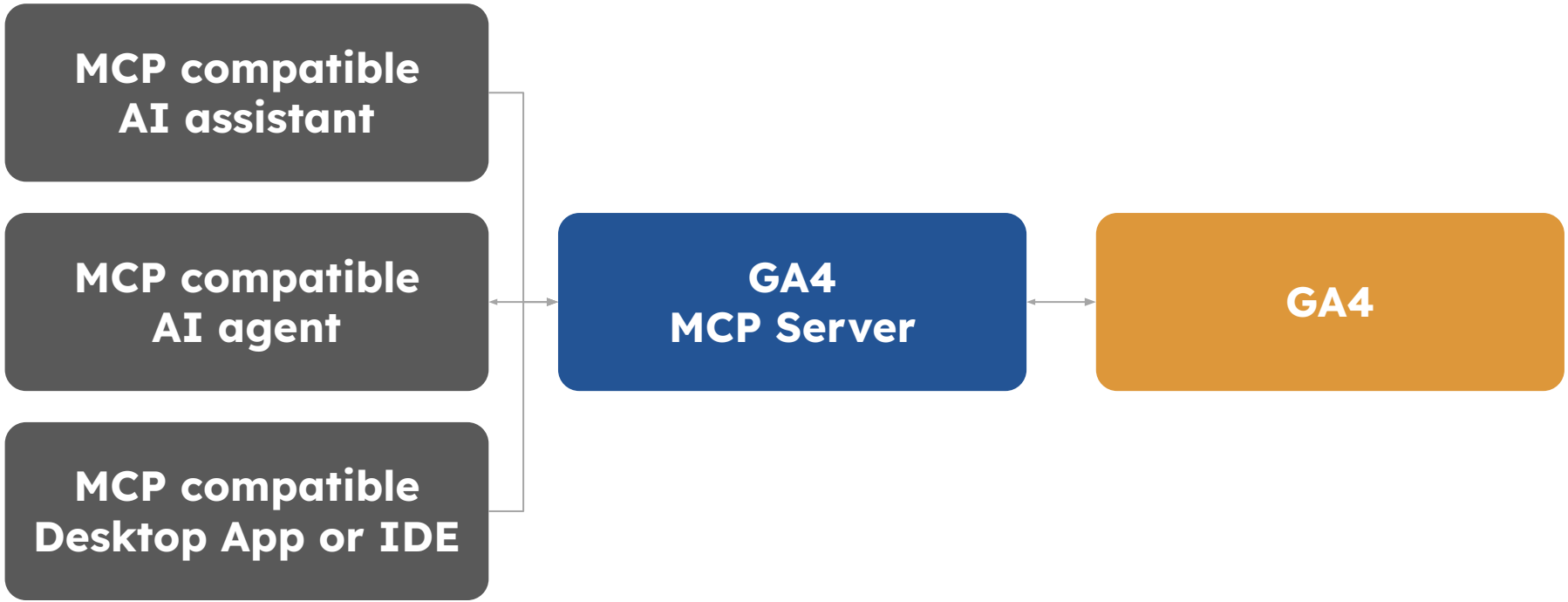
Source: https://apidog.com/blog/mcp-vs-api/

Use multiple MCP Servers at once



Source: https://learn.deeplearning.ai/courses/mcp-build-rich-context-ai-apps-with-anthropic/

MCP Servers are reusable by AI applications



Source: https://learn.deeplearning.ai/courses/mcp-build-rich-context-ai-apps-with-anthropic/

Win-win for the entire ecosystem

For AI application developers

- Connect your app to any MCP server with zero additional work

For tool or API developers

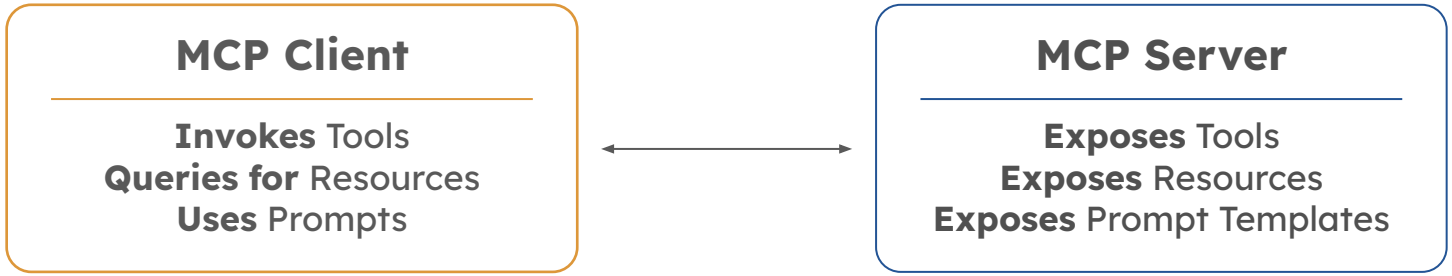
- Build an MCP server once which can be adopted everywhere

For AI application users

- AI applications gain extensive capabilities

How do MCP Servers work?

How does it work together?



Tools

Functions and tools that can be called by the client

- Retrieve data
- Update data
- Search data

Resources

Read-only data or exposed by the server

- Files
- Database records
- API responses

Prompt Templates

Pre-defined templates for AI interactions

- Document Q&A
- Transcript
- Output as JSON

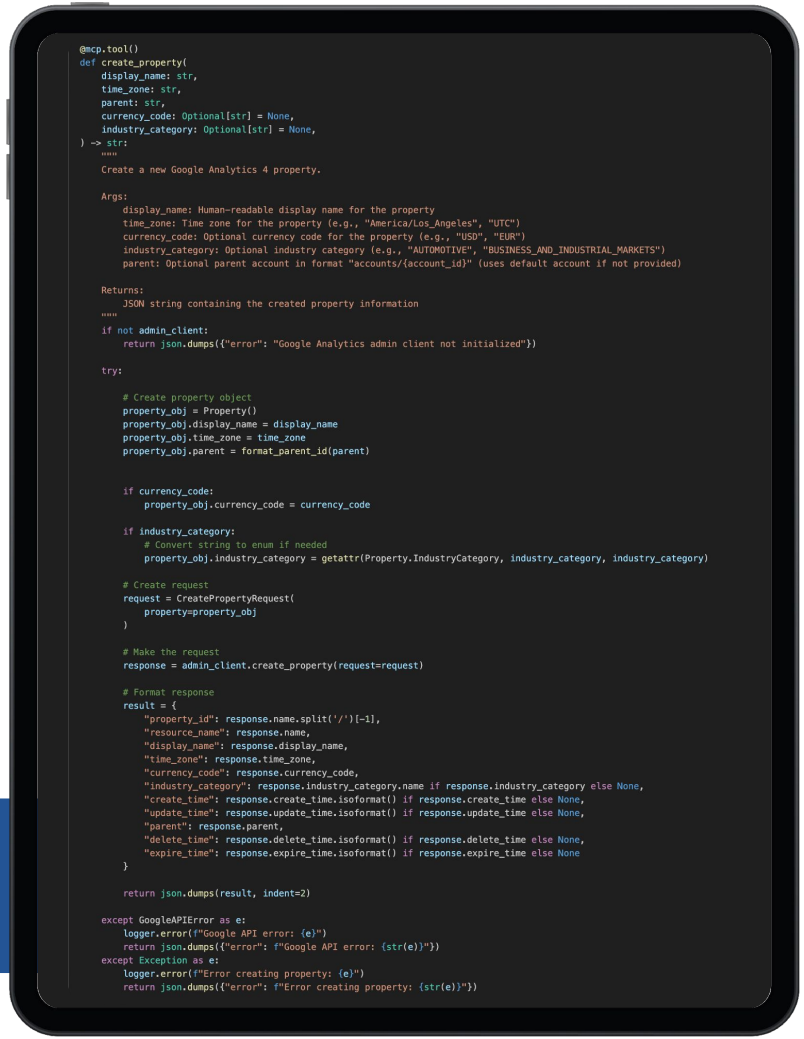
Source: https://learn.deeplearning.ai/courses/mcp-build-rich-context-ai-apps-with-anthropic/

Tools

- Functions that LLMs can call to perform specific actions, like **querying an API**.

- Similar to **POST endpoints** in a REST API, but specifically designed for LLM interactions.

Creating a GA4 property via the GA4 Admin API



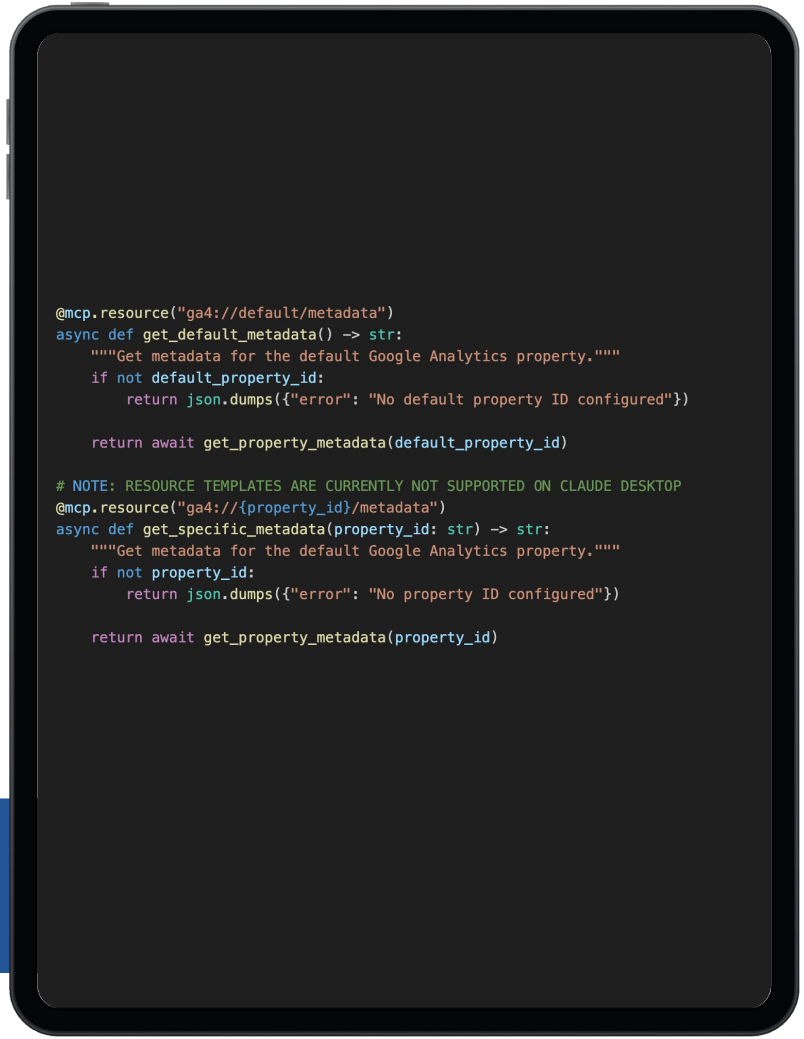
```python
@mcp.tool()
def create_property(
    display_name: str,
    time_zone: str,
    parent: str,
    currency_code: Optional[str] = None,
    industry_category: Optional[str] = None,
) -> str:
    """
    Create a new Google Analytics 4 property.

    Args:
        display_name: Human-readable display name for the property
        time_zone: Time zone for the property (e.g., "America/Los_Angeles", "UTC")
        currency_code: Optional currency code for the property (e.g., "USD", "EUR")
        industry_category: Optional industry category (e.g., "AUTOMOTIVE", "BUSINESS_AND_INDUSTRIAL_MARKETS")
        parent: Optional parent account in format "accounts/{account_id}" (uses default account if not provided)

    Returns:
        JSON string containing the created property information
    """
    if not admin_client:
        return json.dumps({"error": "Google Analytics admin client not initialized"})

    try:

        # Create property object
        property_obj = Property()
        property_obj.display_name = display_name
        property_obj.time_zone = time_zone
        property_obj.parent = format_parent_id(parent)


        if currency_code:
            property_obj.currency_code = currency_code

        if industry_category:
            # Convert string to enum if needed
            property_obj.industry_category = getattr(Property.IndustryCategory, industry_category, industry_category)

        # Create request
        request = CreatePropertyRequest(
            property=property_obj
        )

        # Make the request
        response = admin_client.create_property(request=request)

        # Format response
        result = {
            "property_id": response.name.split('/')[-1],
            "resource_name": response.name,
            "display_name": response.display_name,
            "time_zone": response.time_zone,
            "currency_code": response.currency_code,
            "industry_category": response.industry_category.name if response.industry_category else None,
            "create_time": response.create_time.isoformat() if response.create_time else None,
            "update_time": response.update_time.isoformat() if response.update_time else None,
            "parent": response.parent,
            "delete_time": response.delete_time.isoformat() if response.delete_time else None,
            "expire_time": response.expire_time.isoformat() if response.expire_time else None
        }

        return json.dumps(result, indent=2)

    except GoogleAPIError as e:
        logger.error(f"Google API error: {e}")
        return json.dumps({"error": f"Google API error: {str(e)}"})
    except Exception as e:
        logger.error(f"Error creating property: {e}")
        return json.dumps({"error": f"Error creating property: {str(e)}"})
```

Resources

- Data sources that LLMs can access for additional context.

- Similar to **GET endpoints** in a REST API (no computations).

Fetch metadata from the GA4 Data API



```python
@mcp.resource("ga4://default/metadata")
async def get_default_metadata() -> str:
    """Get metadata for the default Google Analytics property."""
    if not default_property_id:
        return json.dumps({"error": "No default property ID configured"})

    return await get_property_metadata(default_property_id)

# NOTE: RESOURCE TEMPLATES ARE CURRENTLY NOT SUPPORTED ON CLAUDE DESKTOP
@mcp.resource("ga4://{property_id}/metadata")
async def get_specific_metadata(property_id: str) -> str:
    """Get metadata for the default Google Analytics property."""
    if not property_id:
        return json.dumps({"error": "No property ID configured"})

    return await get_property_metadata(property_id)
```

Prompts

- **Reusable** (battle-proved) **templates** surfaced to end users and the LLM

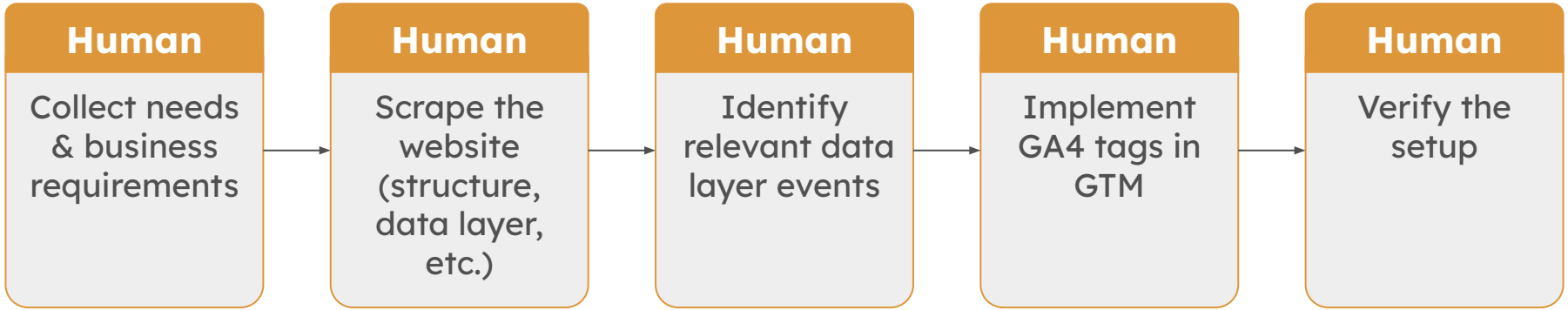
- Powerful way to **standardize** and share common **LLM interactions**

Standardize prompt to fetch a GA4 report

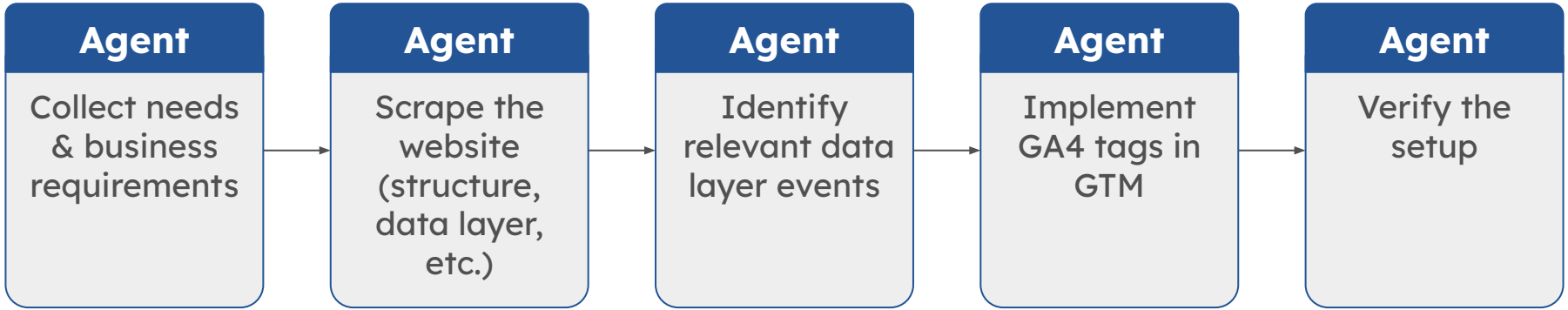
```python
@mcp.prompt()
async def analyze_ga4_metadata(property_id: str) -> str:
    """
    Generates a comprehensive analysis prompt for Claude to explore the metadata
    of a specific Google Analytics 4 property. This prompt guides Claude to
    utilize the `get_metadata_tool(property_id='{property_id}')` and then synthesize insights about the
    available dimensions and metrics.

    Args:
        property_id: The Google Analytics 4 property ID (e.g., "123456789") for which
                     to analyze metadata.

    Returns:
        A detailed prompt string for Claude, instructing it to fetch, parse, and
        analyze the GA4 property metadata.
    """
    return f"""Analyze the Google Analytics 4 property metadata for property ID '{property_id}'.

1.  First, use the `get_metadata_tool(property_id='{property_id}')` to retrieve the metadata.

2.  For the metadata found, extract and organize the following information for both dimensions and metrics:
    -   API Name
    -   UI Name (if available)
    -   Description
    -   Category (for both dimensions and metrics)
    -   Type (for metrics, e.g., INTEGER, FLOAT)

3.  Provide a comprehensive summary that includes:
    -   An overview of the total number of dimensions and metrics available.
    -   A breakdown of dimensions by category, highlighting the most common categories.
    -   A breakdown of metrics by category, highlighting the most common categories.
    -   Identify any potentially useful or commonly used dimensions (e.g., 'date', 'country', 'deviceCategory').
    -   Identify any potentially useful or commonly used metrics (e.g., 'activeUsers', 'screenPageViews', 'sessions').
    -   Suggest potential combinations of dimensions and metrics that could be used for common GA4 reports.
    -   Discuss any interesting or unusual dimensions/metrics.

4.  Organize your findings in a clear, structured format with headings and bullet points for easy readability.

Please present both detailed information about each dimension and metric and a high-level synthesis of the data capabil
ities of the '
{property_id}' property.
```

Why you should use MCP Servers in Digital Analytics

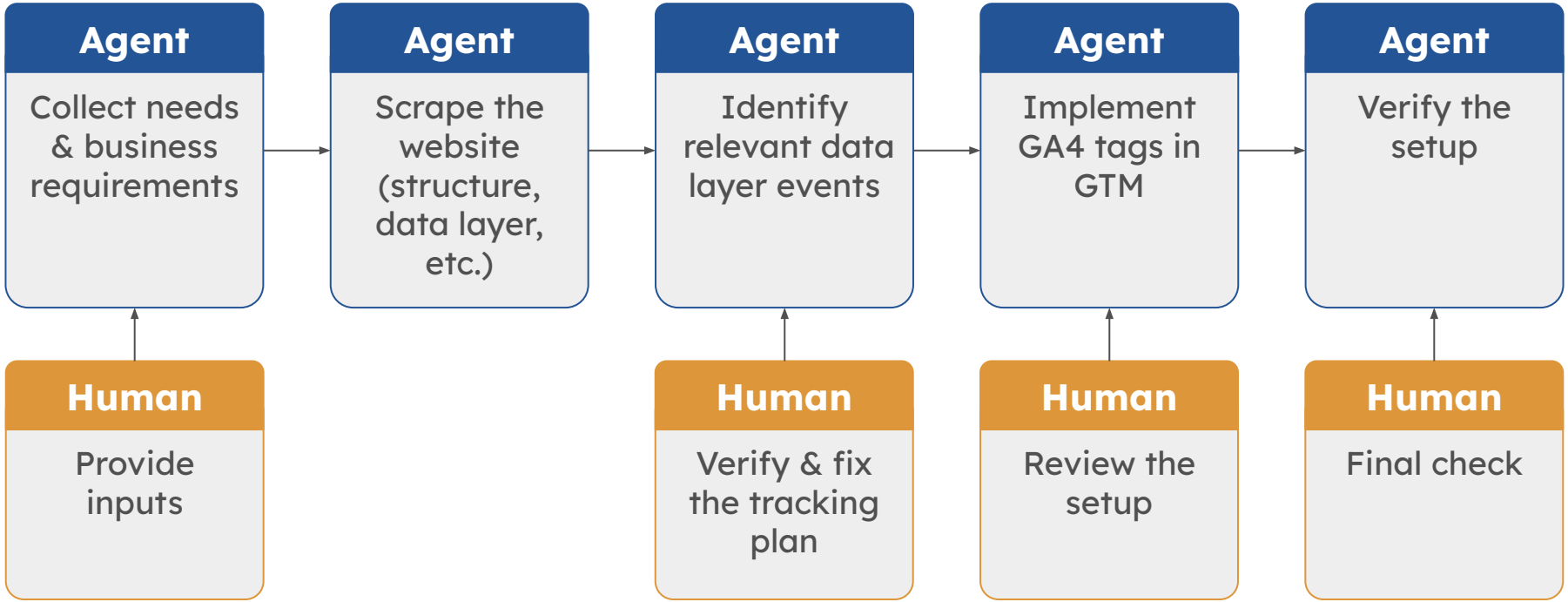
Use Case: End-to-end AI implementation



Use Case: End-to-end AI implementation



Use Case: End-to-end AI implementation



Use Case: An Agent's Digital Analytics Toolkit

Playwright MCP Server

MCP Server that provides **browser** automation **capabilities** using Playwright

GTM MCP Server

Stape

MCP Server that provides an **interface** to the **Google Tag Manager API**

Google Analytics MCP Server

MCP Server that provides an **interface** to the **GA4 Data & Admin API**

Demo

Demo: DataLayer Discovery

Demo: GA4 Property & Data Stream Setup

Demo: GTM Setup

Demo: GA4 Validation

What we've covered

MCP Servers as universal translators between AI & external services - enabling standardized connections that work across any MCP-compatible application

Win-win ecosystem benefits - AI application developers get zero-effort integrations, tool developers build once and deploy everywhere, and users gain extensive new capabilities

MCP architecture providing AI access to tools, resources, and prompt templates - giving LLMs the ability to perform actions, access data, and use pre-built interaction patterns

Real-world digital analytics applications - speed up your day-to-day work, unlock use cases that were not possible in the past. We'll see this being used more and more in the near future!

Thank you –
and let’s connect!

Gunnar Griese
Digital Analytics Consultant & Analytics Engineer
gunnargriese.com

8-bit-sheep
Copenhagen, Denmark