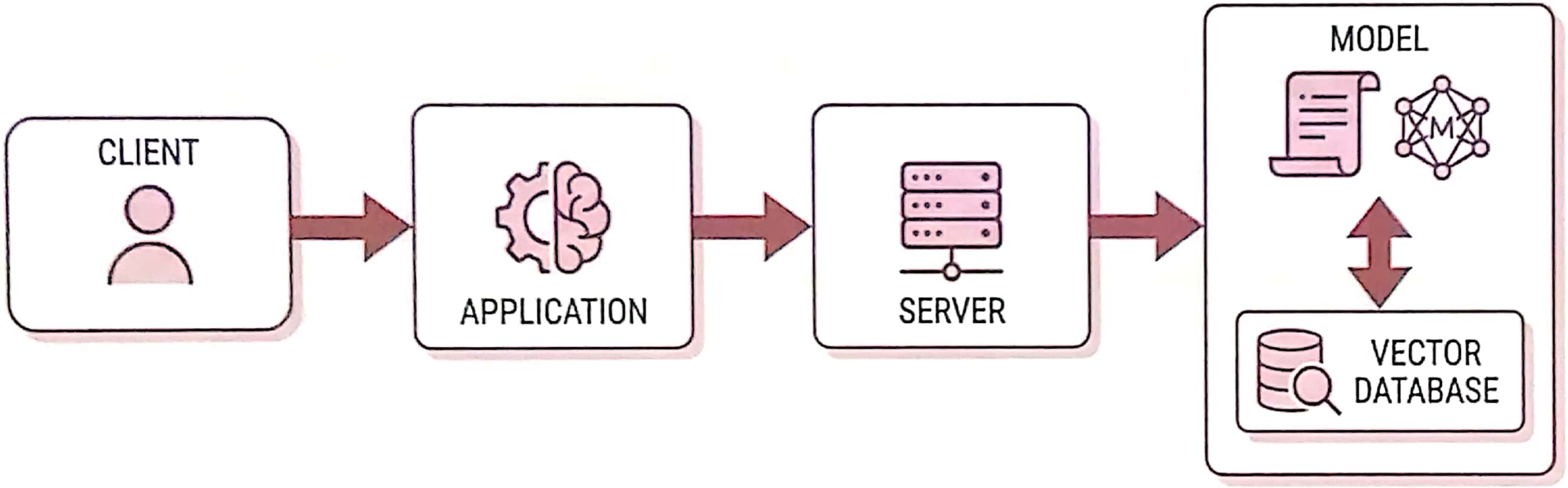
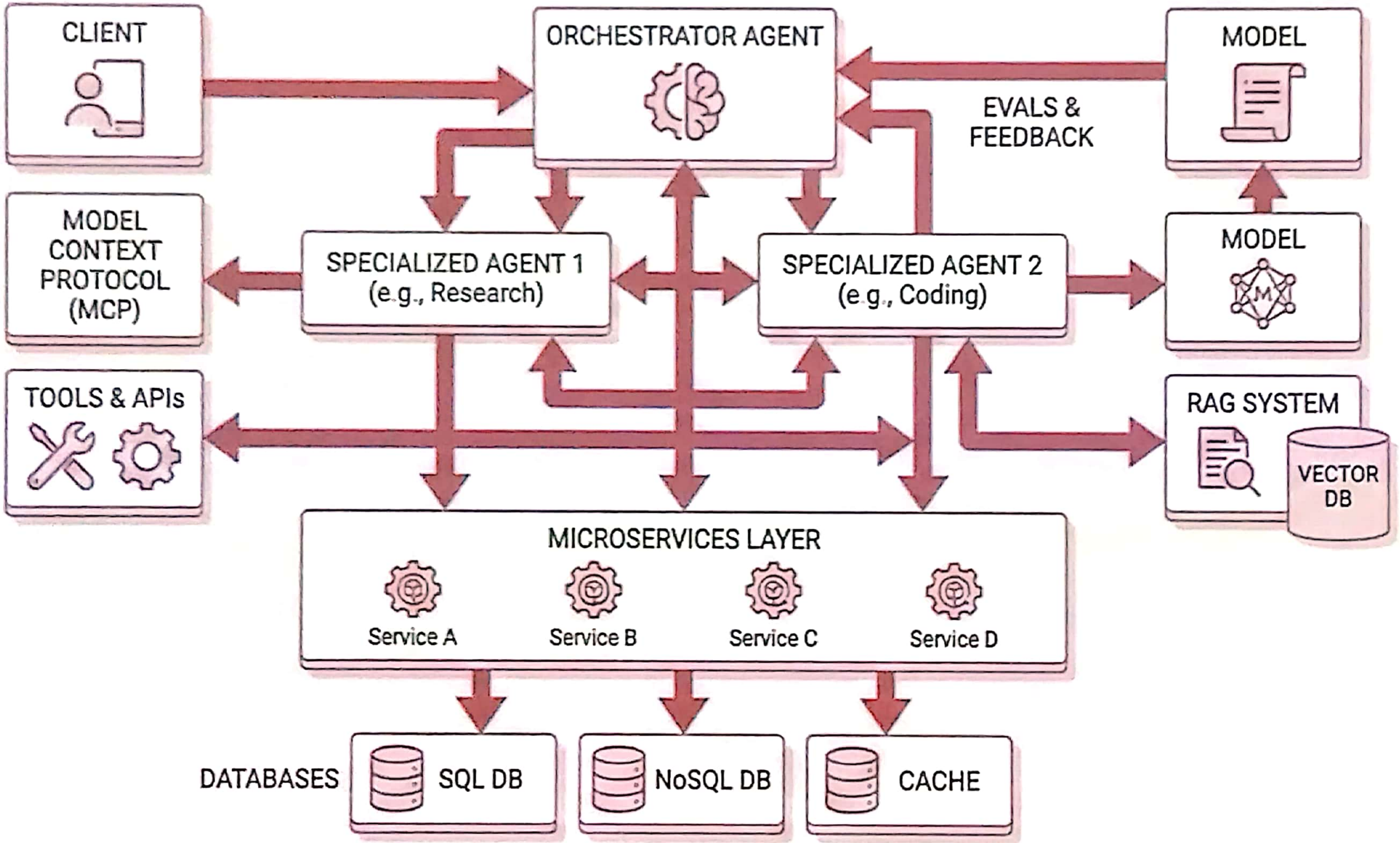
High-Level Architecture: Client-App-Server-Model Flow with Vector DB Retrieval



Complex Multi-Agent Architecture with Advanced Components

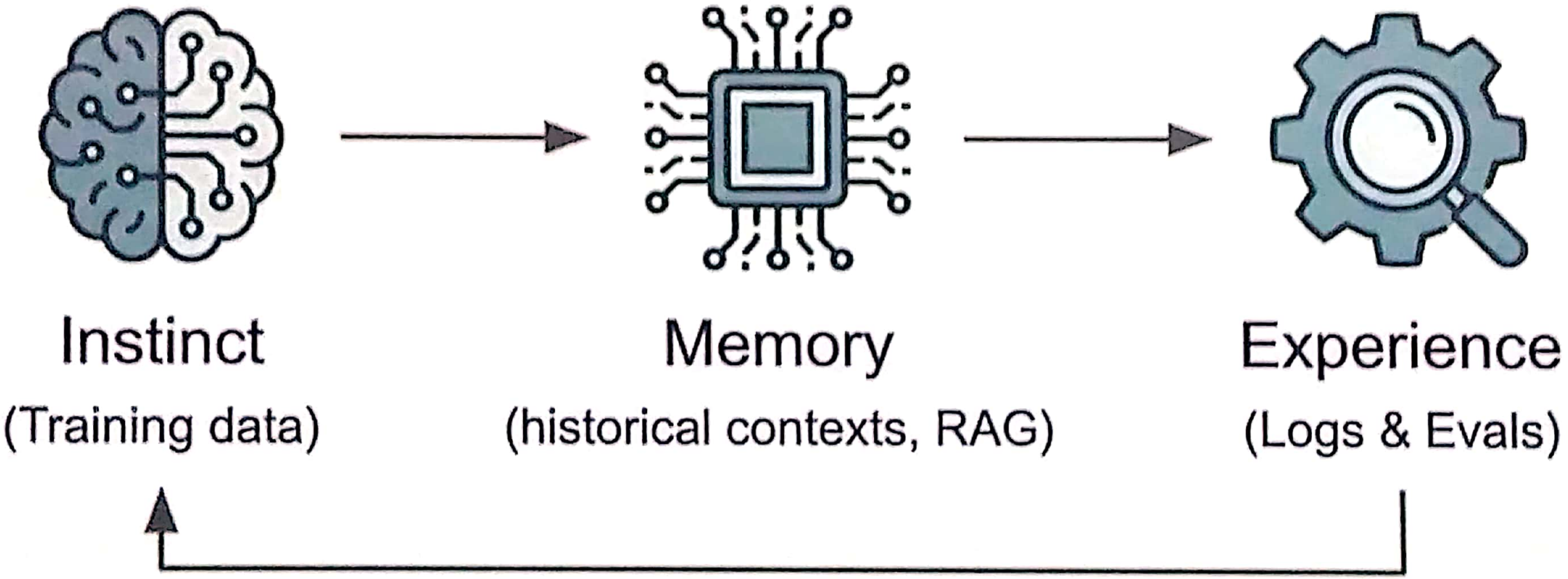


Challenges of scaling

- Data grows over time, ad-hoc approaches to retrieve data
- Irrelevant, high-confidence data confuses the agent
- Vector databases cannot consistently store structured data
- Impossible to validate what data the agent used for reasoning
- Consistent snapshots of all your data

Managing massive amounts of data is the primary problem

The Problem



Instinct

(Training data)

Memory

(historical contexts, RAG)

Experience

(Logs & Evals)

Data needs for an Agentic AI Application

The Problem

training_data_01

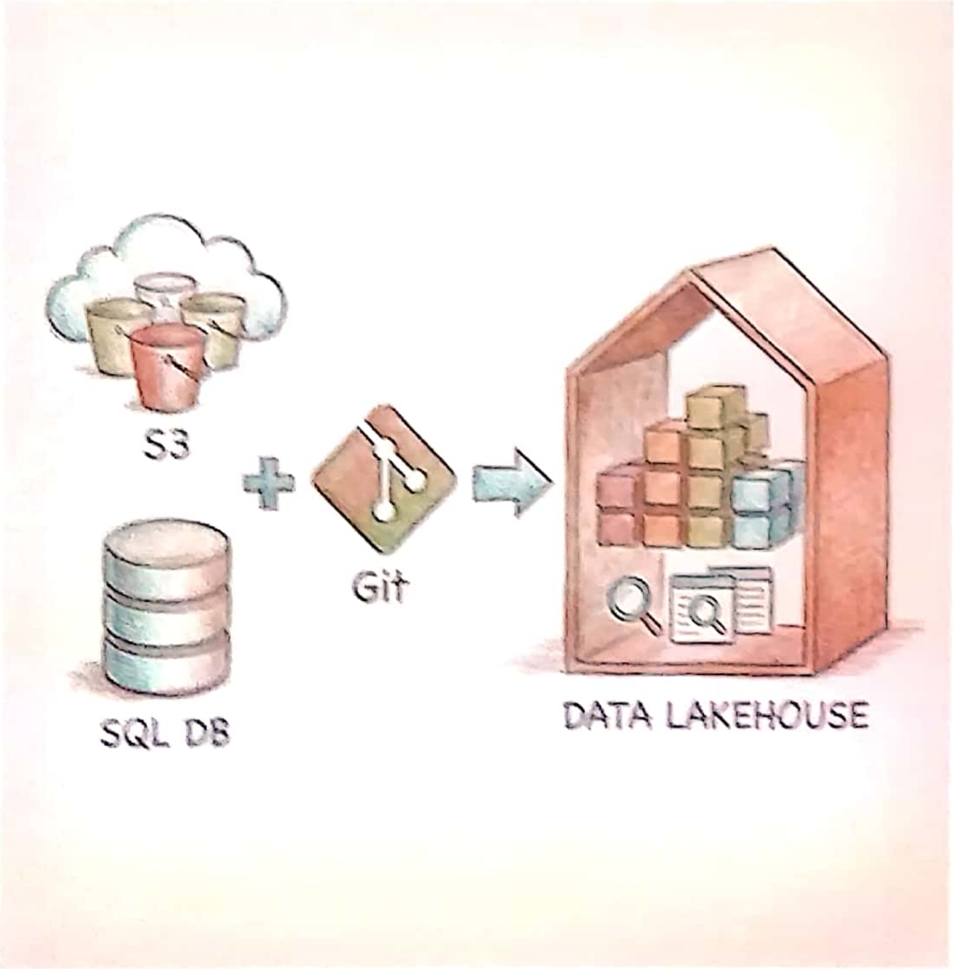
training_data_02

training_data_main _20250612

data_final_01_last

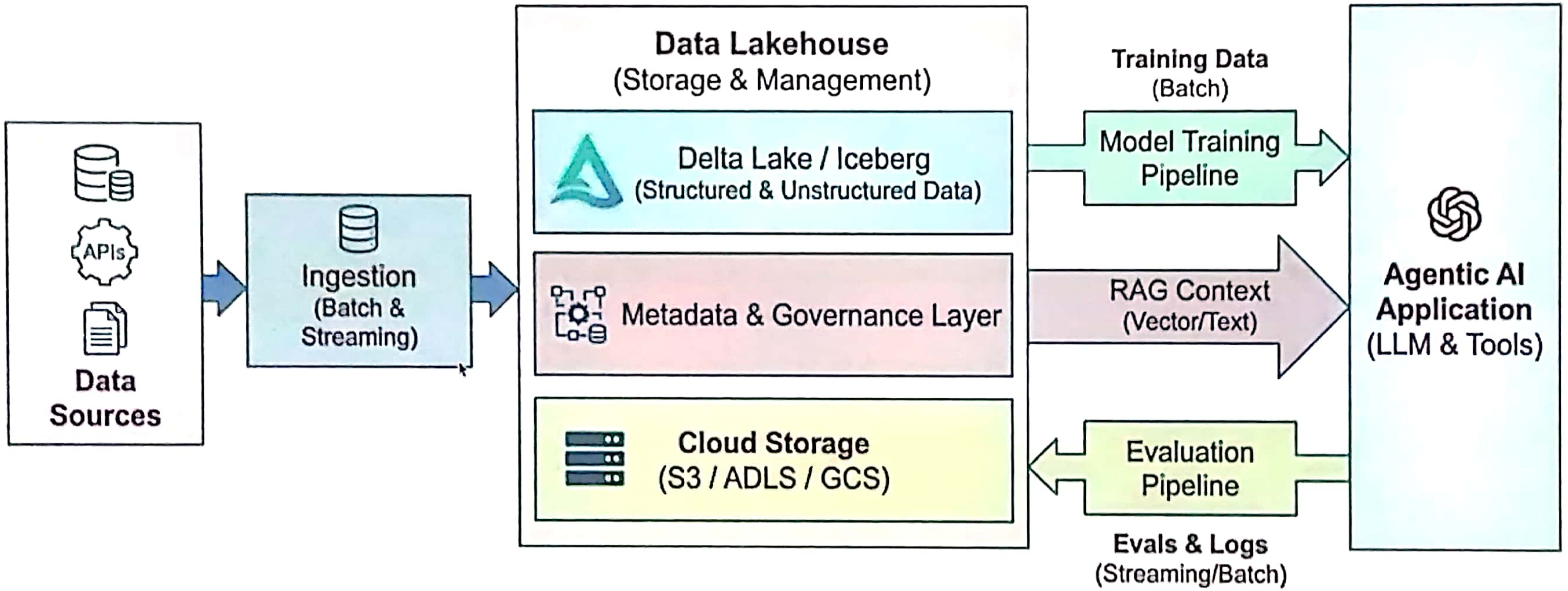
- How do you manage code?
  - versioned, auditable
  - branch, review, merge
- How do you manage data?
  - file dumps in S3
  - improper naming
  - no versioning
  - **TOTAL MESS!**

The Solution



- Git for Data
- Auditable, versioned changes to data using snapshots
- Time travel - use a specific historic snapshot of data
- Supports ACID Transactions

System Design: Agentic AI with Data Lakehouse



Data Sources

APIs

Ingestion (Batch & Streaming)

Data Lakehouse (Storage & Management)

Delta Lake / Iceberg (Structured & Unstructured Data)

Metadata & Governance Layer

Cloud Storage (S3 / ADLS / GCS)

Training Data (Batch)

Model Training Pipeline

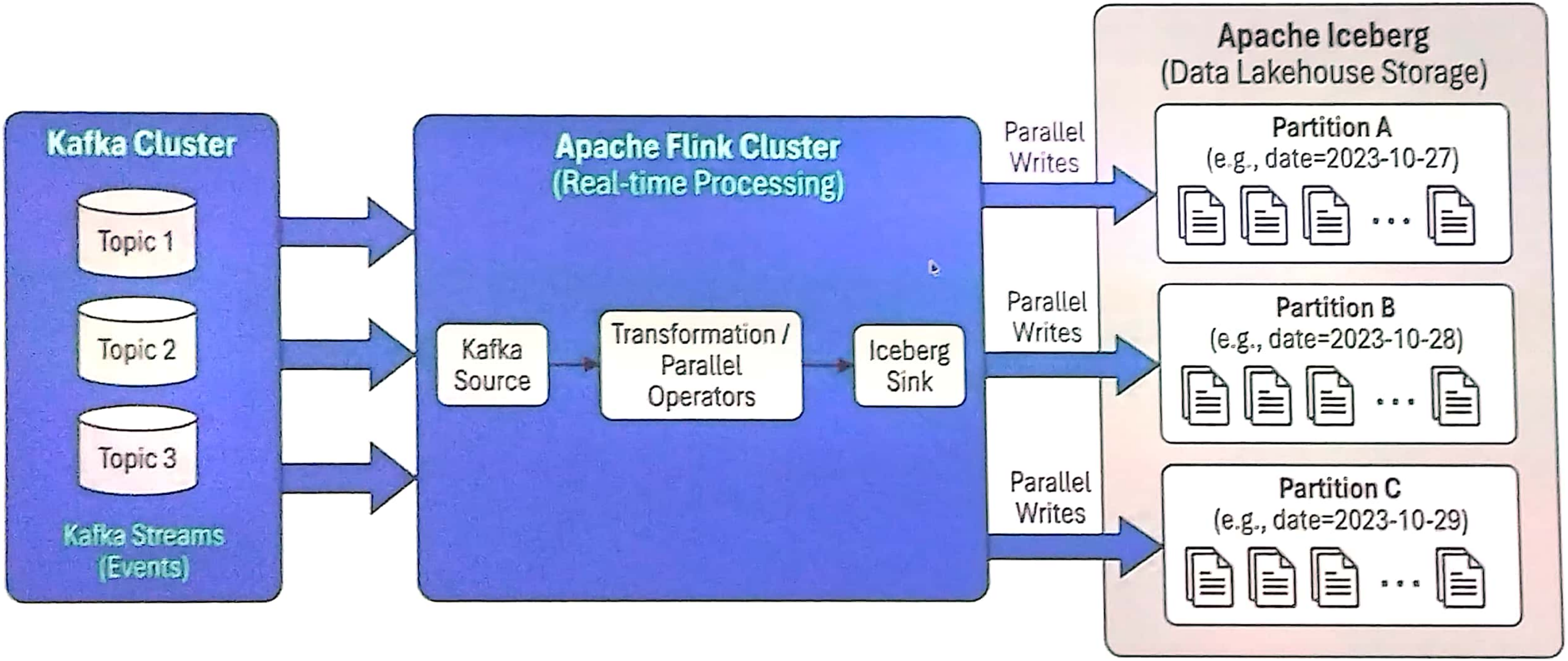
RAG Context (Vector/Text)

Evaluation Pipeline

Evals & Logs (Streaming/Batch)

Agentic AI Application (LLM & Tools)

High-Level Design: Real-time Flink Ingestion to Iceberg Partitions



Efficiently handles massive data streams with horizontal scalability and partitioned storage.

A word count program

```
ds = .... read data set

mapped_ds = ds.map(lambda x: (x[0].upper(), x[1], x[2]), output_type=type_info)

with_timestamps_ds =
mapped_ds.assign_timestamps_and_watermarks(watermark_strategy)

keyed_ds = with_timestamps_ds.key_by(lambda x: x[0])

windowed_ds =
keyed_ds.window(TumblingEventTimeWindows.of(Time.seconds(5))) \
.reduce(CountAggregator())

env.execute()
```