Provenance

What’s Happening in your Production Data and ML Systems?

Code Freeze 2020: Observability

Date: January 16, 2020

Speaker: Donald Sawyer

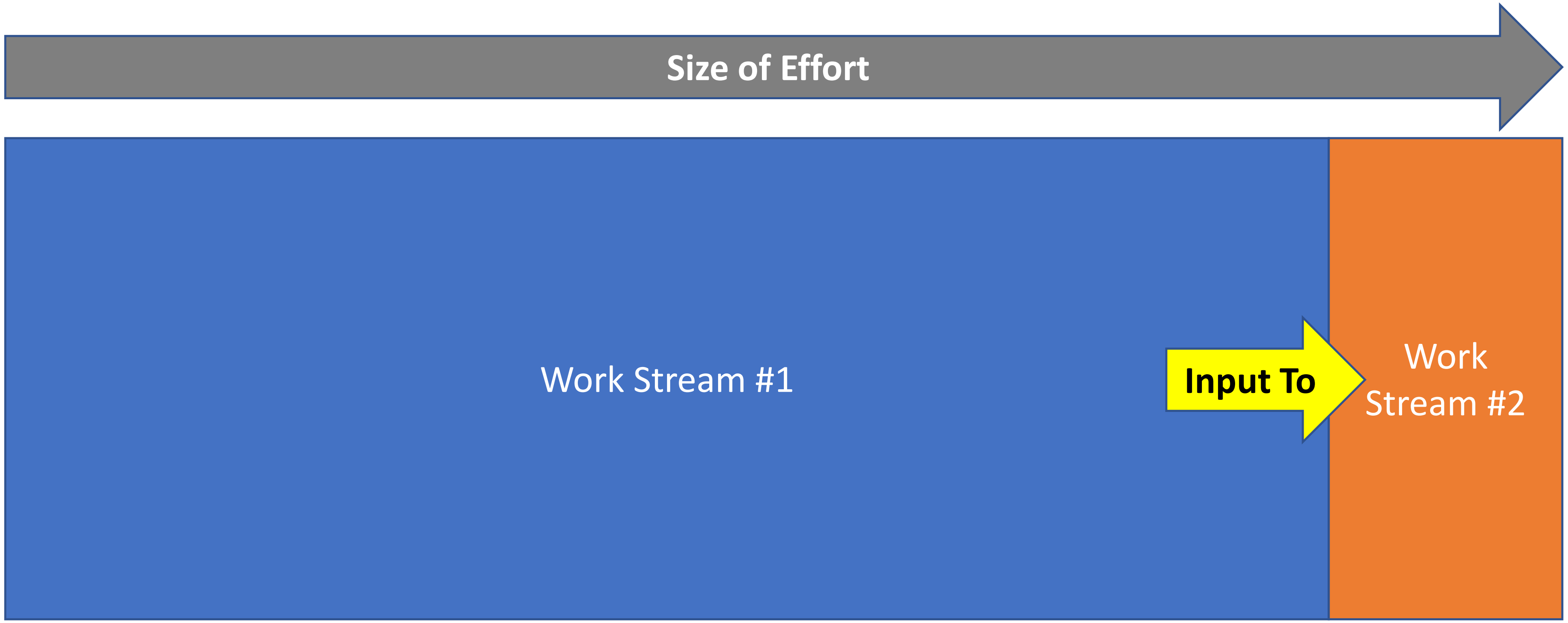
About Donald Sawyer

- Sr. Solutions Architect, Data Engineering
- Focuses on Hadoop, Spark, and Data Engineering
- Continuous learning and education
- We're hiring!
    - Data Engineers and Solution Architects
    - Machine Learning Architects & Engineers
    - Data Scientists

- MS Software Engineering (2016)
- Adjunct since 2017
    - Big Data Engineering and Architecture

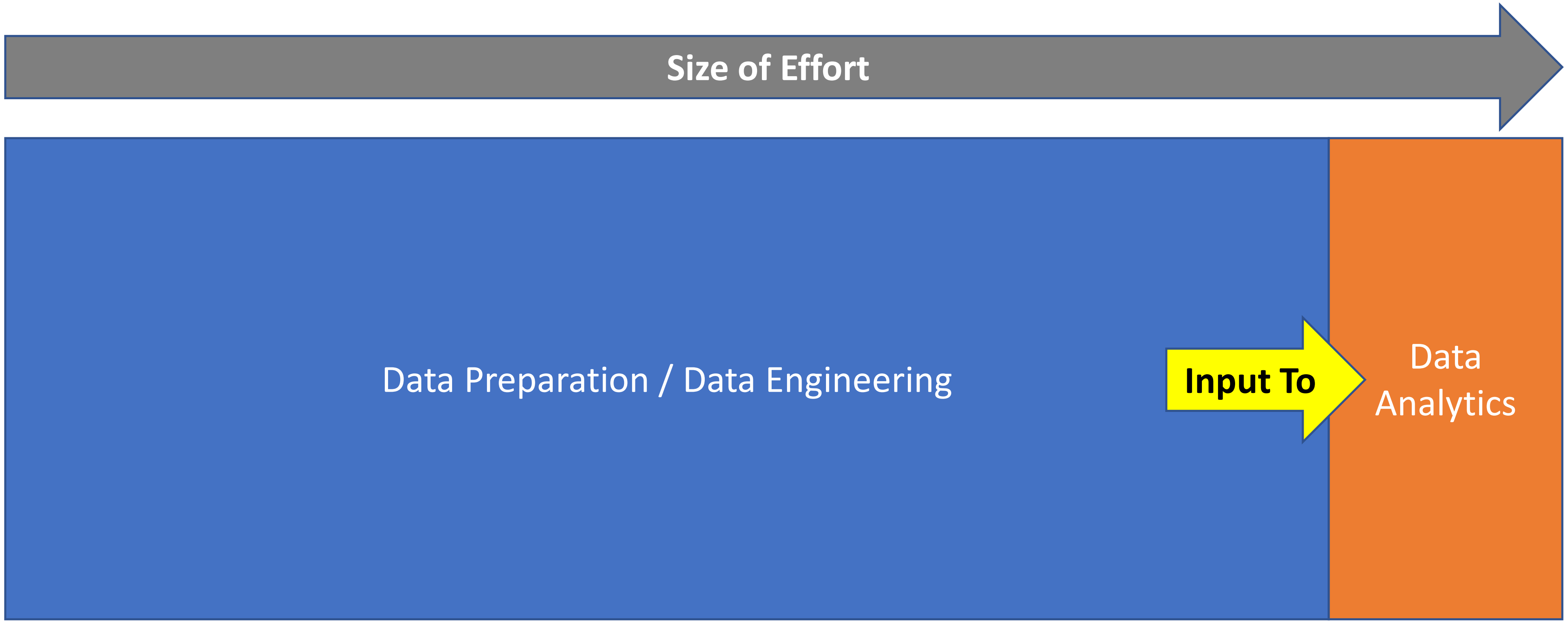
Why This Topic?

Where Would You Want To Spend Your Time?



Why This Topic?

Where Would You Want To Spend Your Time?



Abstract

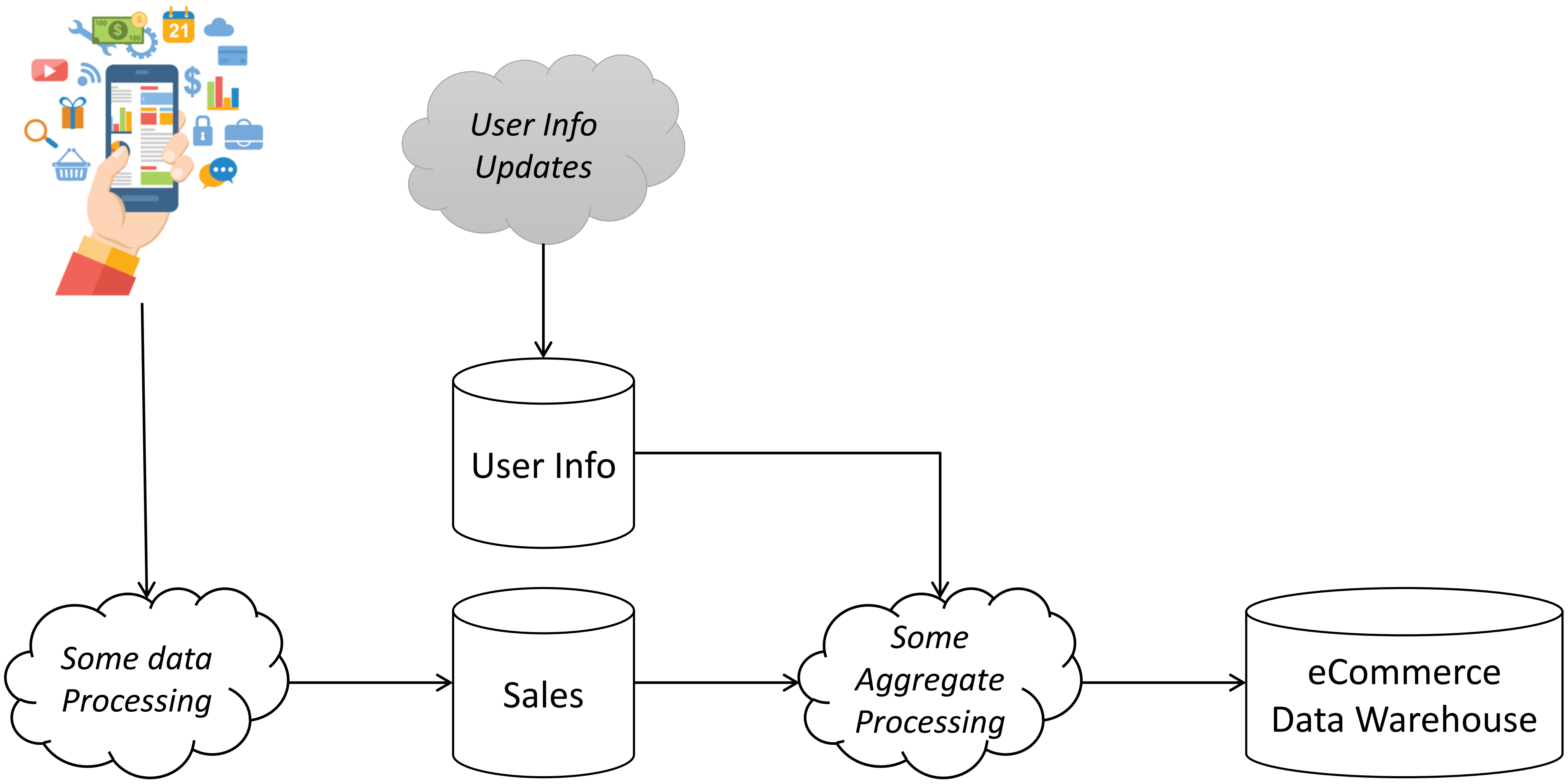
Data is often your company's most valuable asset, yet very few implementations of ETL and machine learning capabilities provide the ability to measure their effectiveness (quality), or their performance. Data and machine learning pipelines are built as multi-step software integrations, but when an issue arises, how will you determine what happened? Machine learning models degrade over time, but without the ability to observe them, your models could be ineffective long before someone notices.

In this talk, you will learn strategies for building visibility into data systems using data and ML provenance. Provenance is the concept of tracking the evolution of your data and models as data are moving through your system. As a side effect, you will also gain the measurements that typical software systems require to measure latency, throughput, load, and error rates...all without having to sift through dozens of logs from different systems in your technology stack.

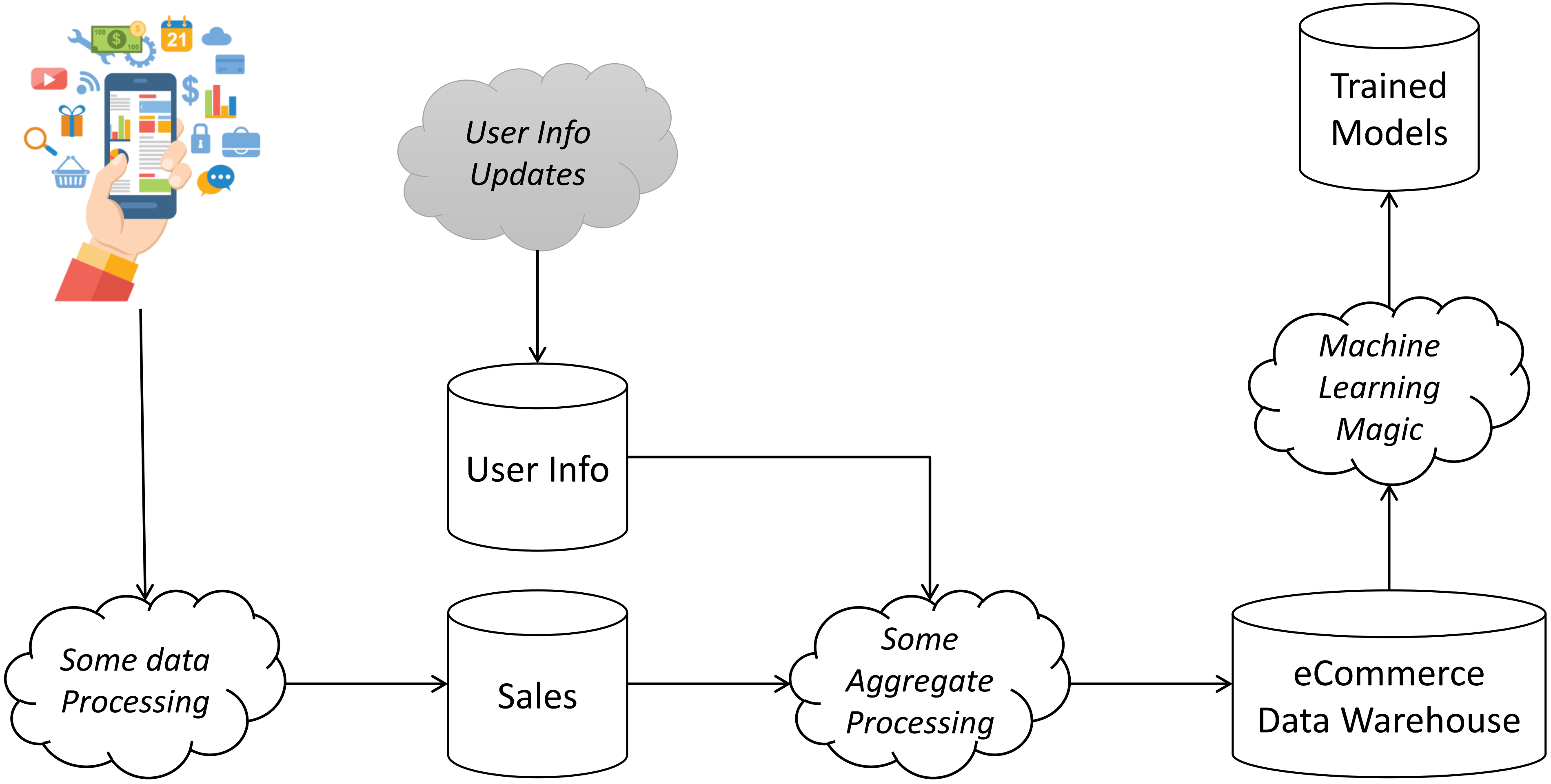
Introductory Concepts

A quick intro to provenance and pipelines

Data Pipeline



Data Pipeline + ML Pipeline



Defining Provenance

Describing the **origin** for data, but also its **changes over time**.

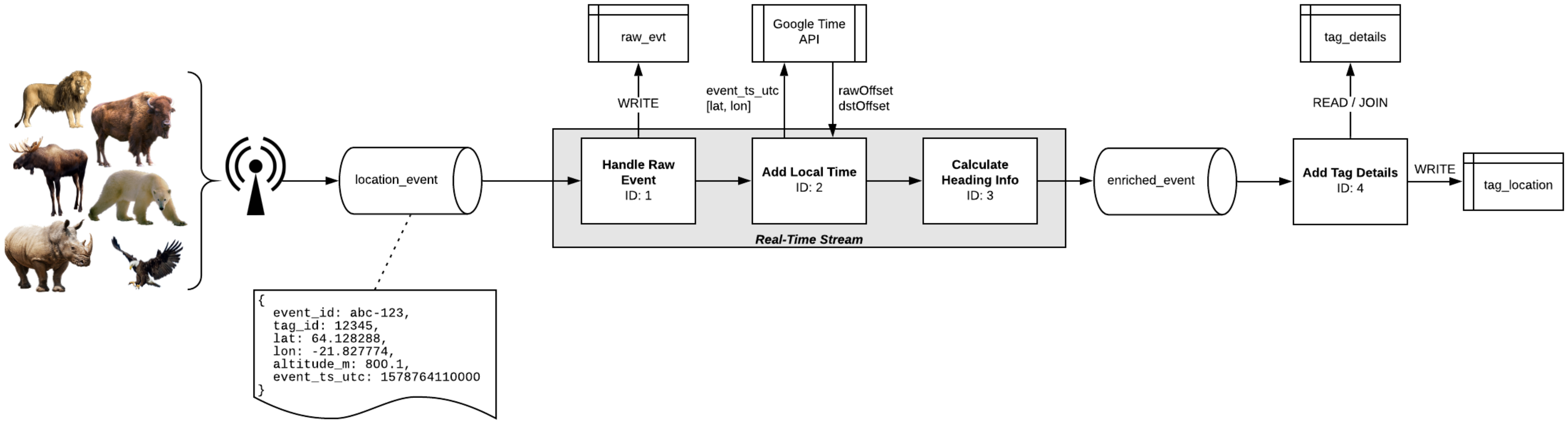
Common questions to answer with provenance:

- Why
- How
- Where

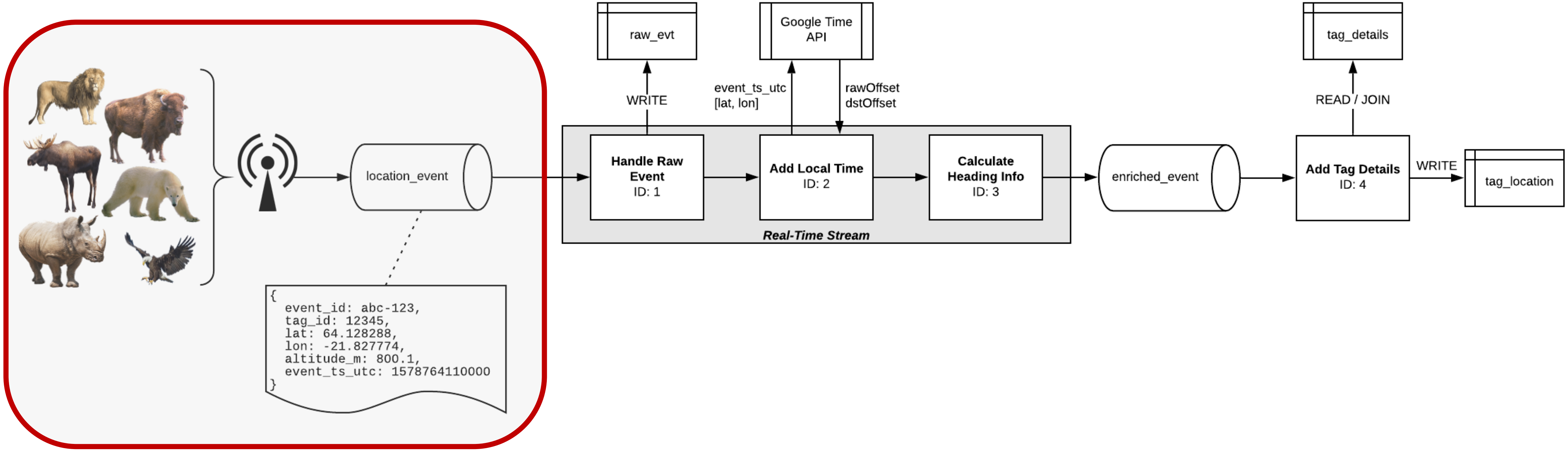
Decoupled Data System

Walkthrough of a decoupled data system example.

A Decoupled Pipeline, In Detail

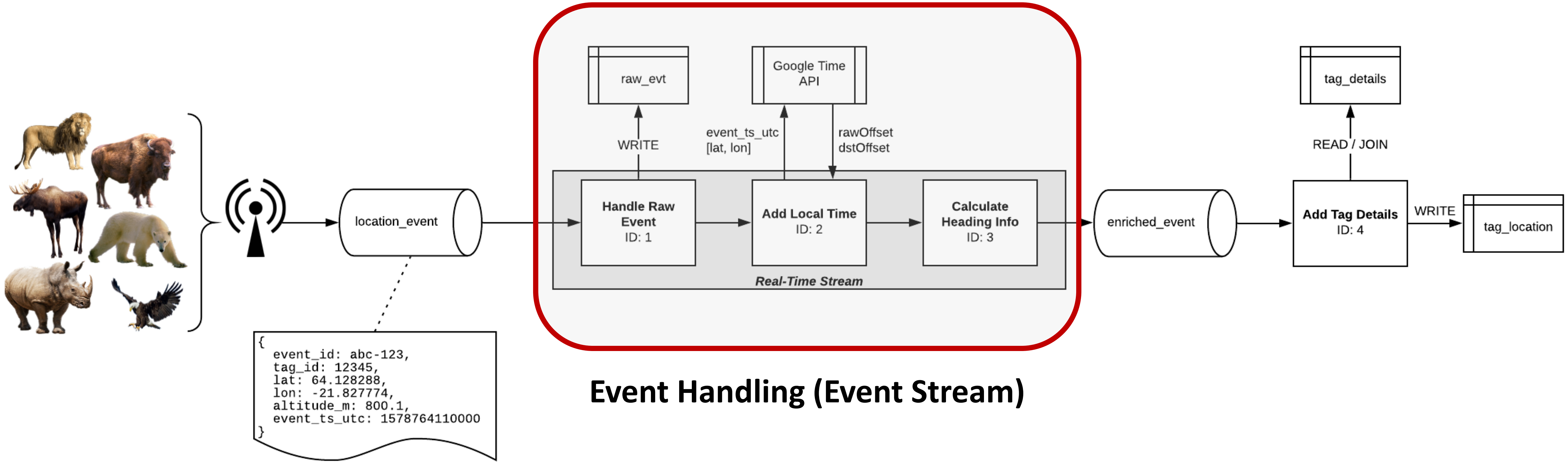


A Decoupled Pipeline, In Detail



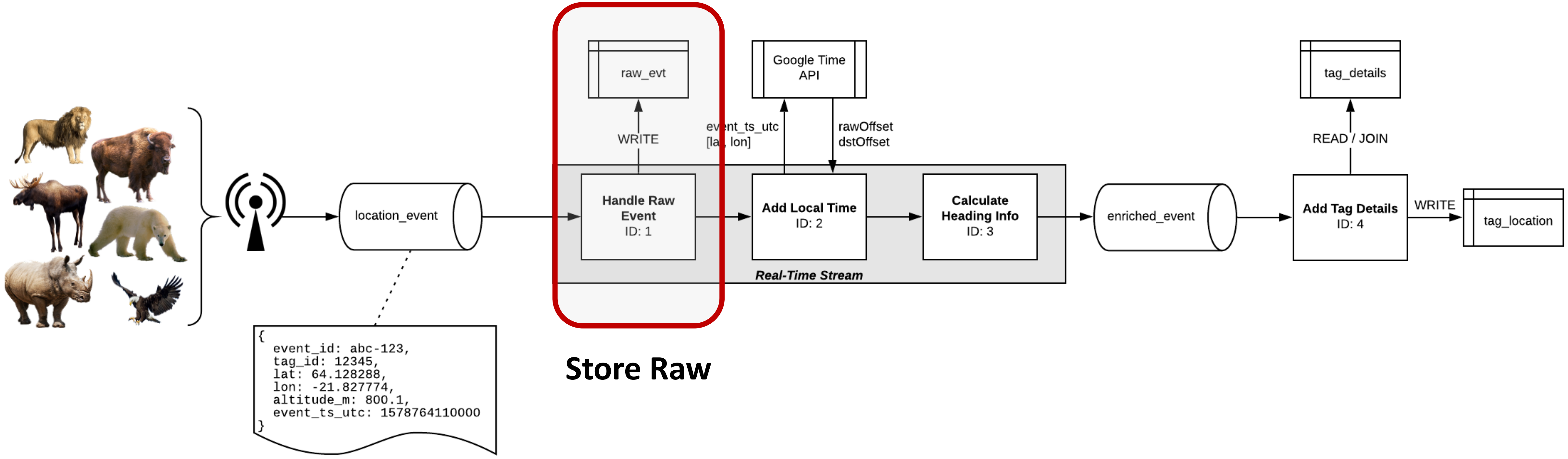
Event Generation

A Decoupled Pipeline, In Detail

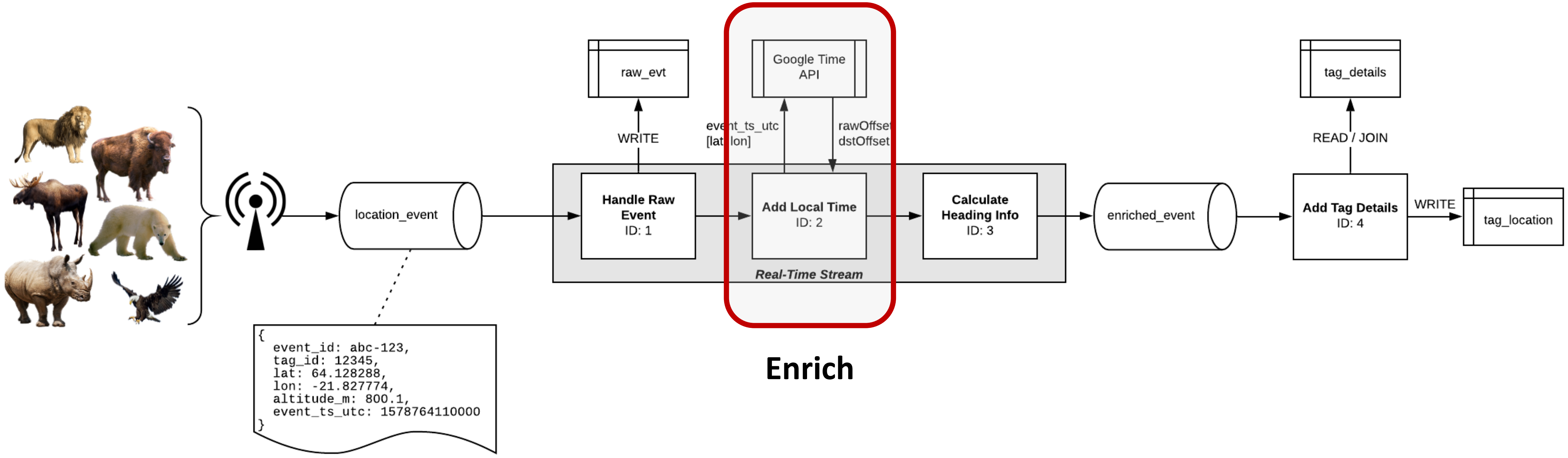


Event Handling (Event Stream)

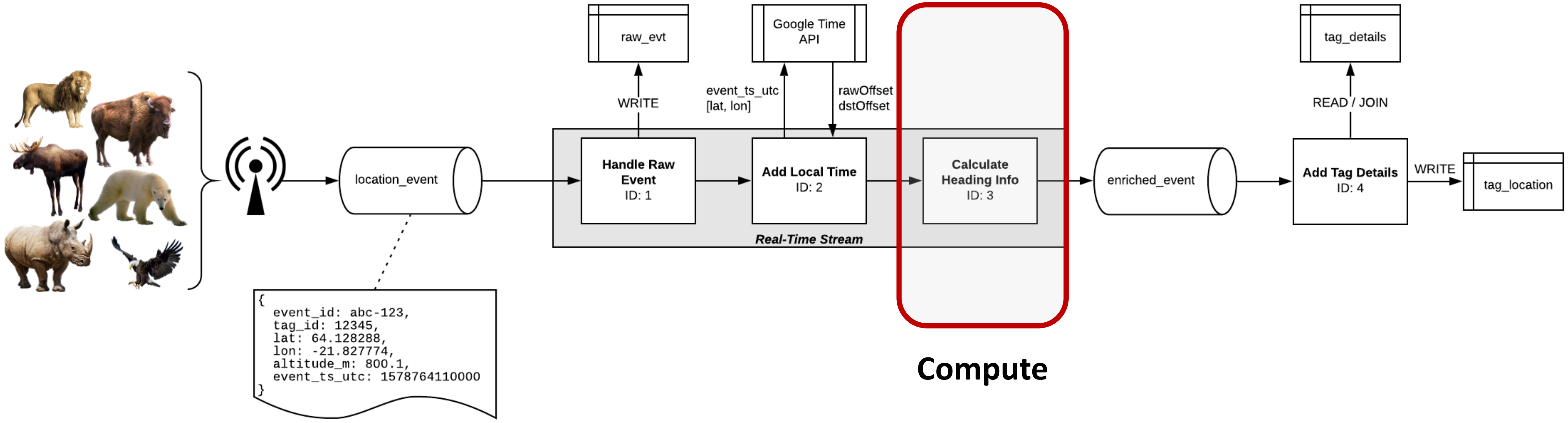
A Decoupled Pipeline, In Detail



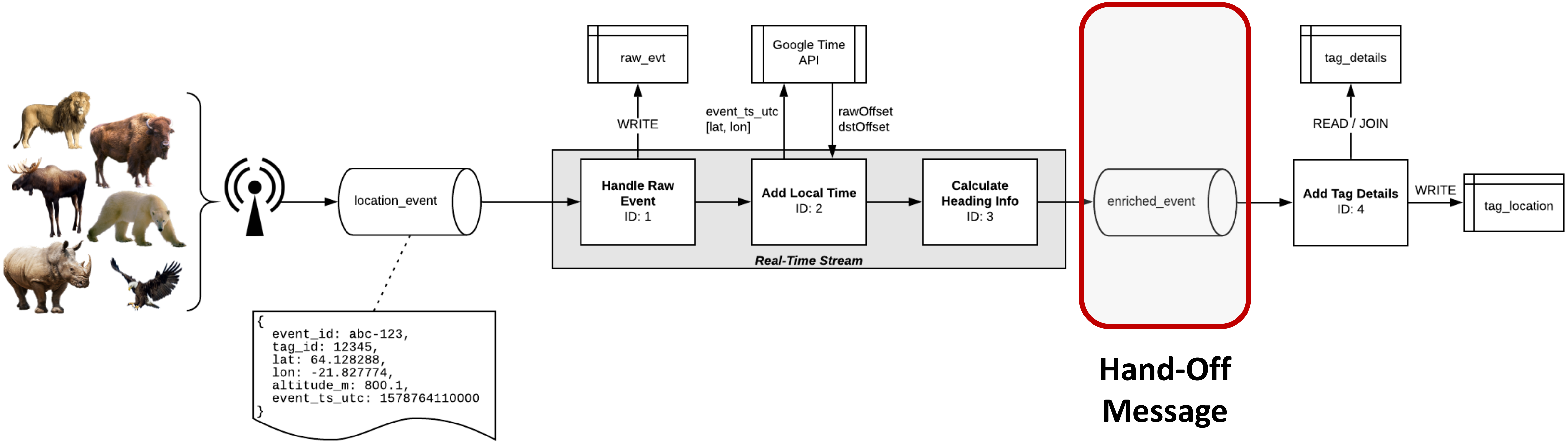
A Decoupled Pipeline, In Detail



A Decoupled Pipeline, In Detail

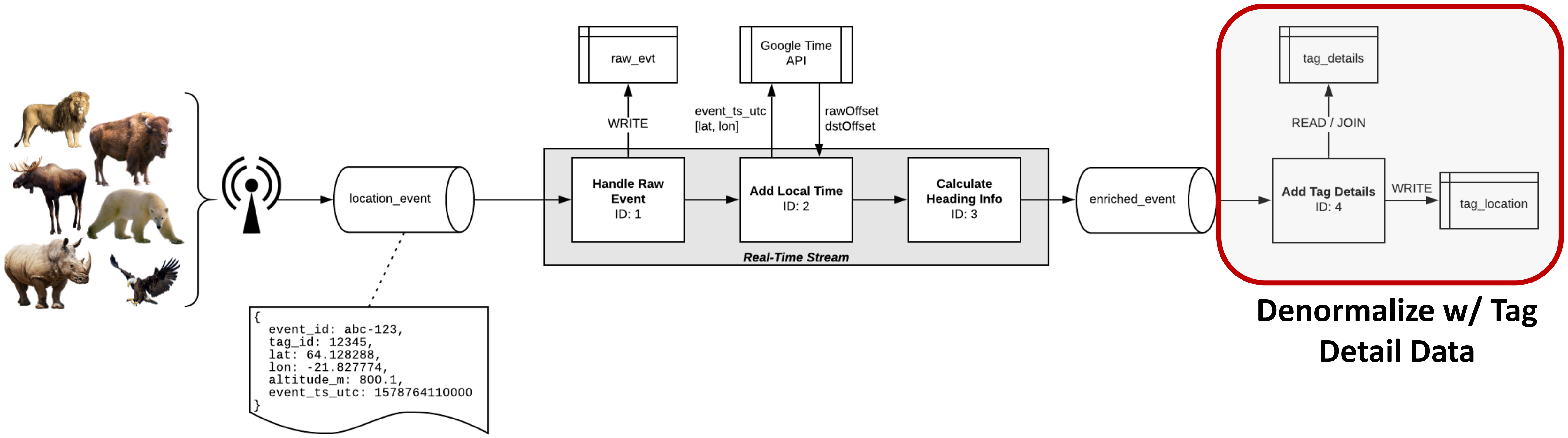


A Decoupled Pipeline, In Detail



Hand-Off Message

A Decoupled Pipeline, In Detail



Denormalize w/ Tag Detail Data

Provenance by Use Case

Exploring how provenance improves observability within a data pipeline.

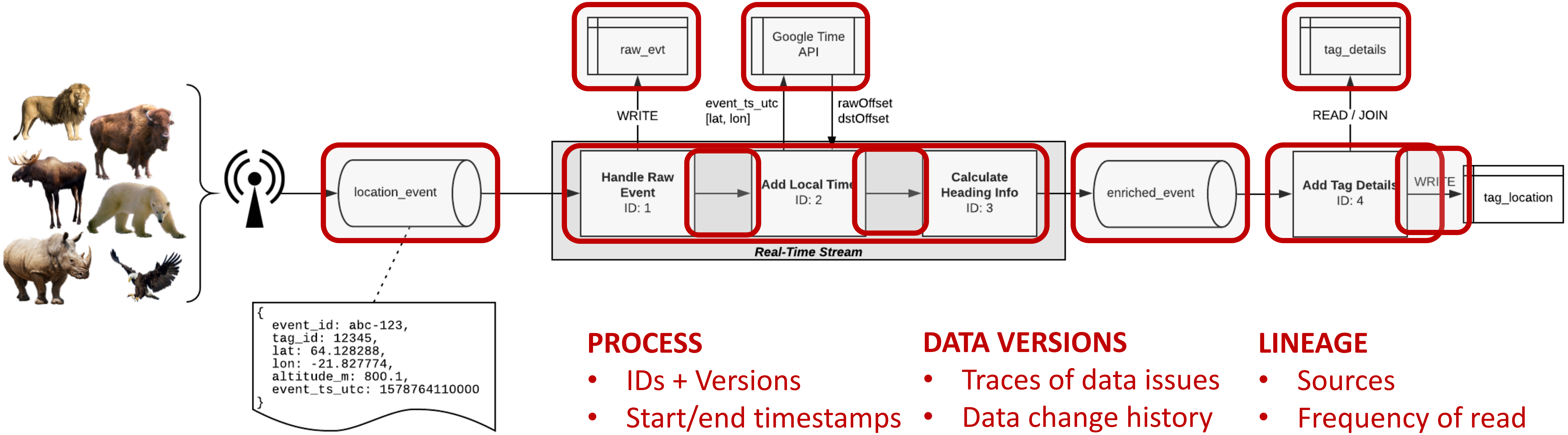
Uses for Provenance

- Audit trails and debugging
- Data quality
- Repeatability in data processing
- Informational

Use: Audit Trail & Debugging

- Audit the **process** by which data were produced
- Optimization of data derivation process (metrics)
- Runtime traces of execution to identify exceptions
- Backtracking DAGs to identify bad sources
- Identify the **nodes** that perform processing in a distributed system
- Use provenance as inputs to automated test cases for defect fixes

Example: Audit Trail & Debugging



PROCESS

- IDs + Versions
- Start/end timestamps (EPOCH)
- Transformations
- Inputs
- Configurations

DATA VERSIONS

- Traces of data issues
- Data change history
- Defect data

LINEAGE

- Sources
- Frequency of read

Use: Replication of Processing

Idea: Provide data for reproducibility of data results, or to re-run data through a portion of the pipeline.

Needs:

- Versioned data as it flows through the system**
- Operations performed (steps in DAG + versions)
- Parameters used for input or configuration

*** If you don't have data versions, you must have the raw sources*

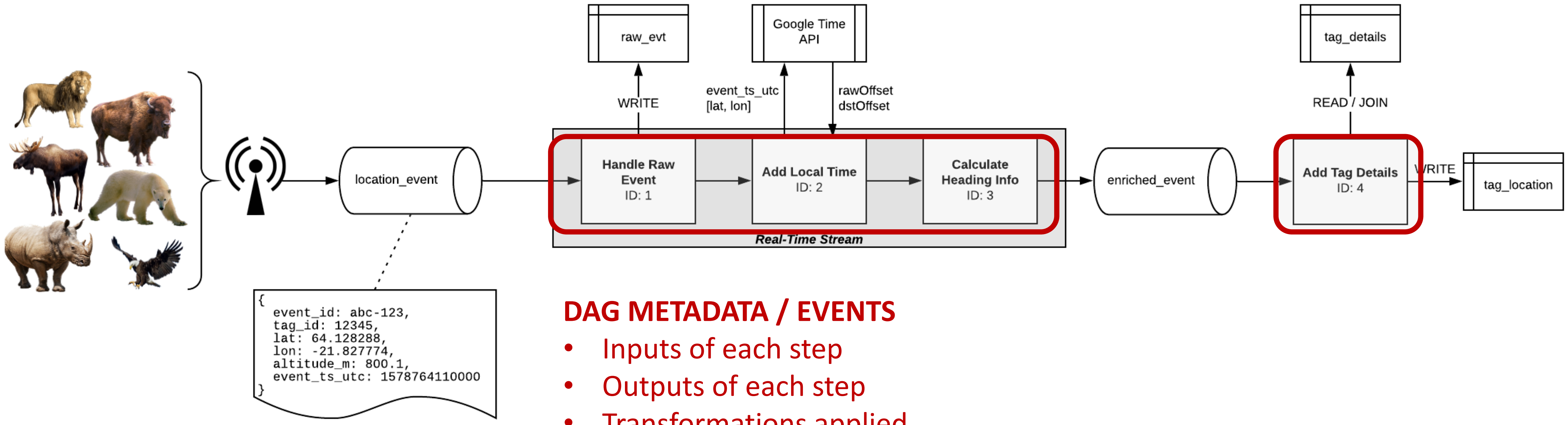
Use: Data and System Quality

Idea: Identify existence of errors and faults in a pipeline that produce invalid, inconsistent, or incomplete results.

Provenance needed to gain visibility to issues:

- Steps that handled the data
- Inputs to each step
- Outputs of each step
- Transformations applied
- Platform information

Example: Data and System Quality



DAG METADATA / EVENTS

- Inputs of each step
- Outputs of each step
- Transformations applied
- Errors/faults encountered

Use: Descriptive Analysis

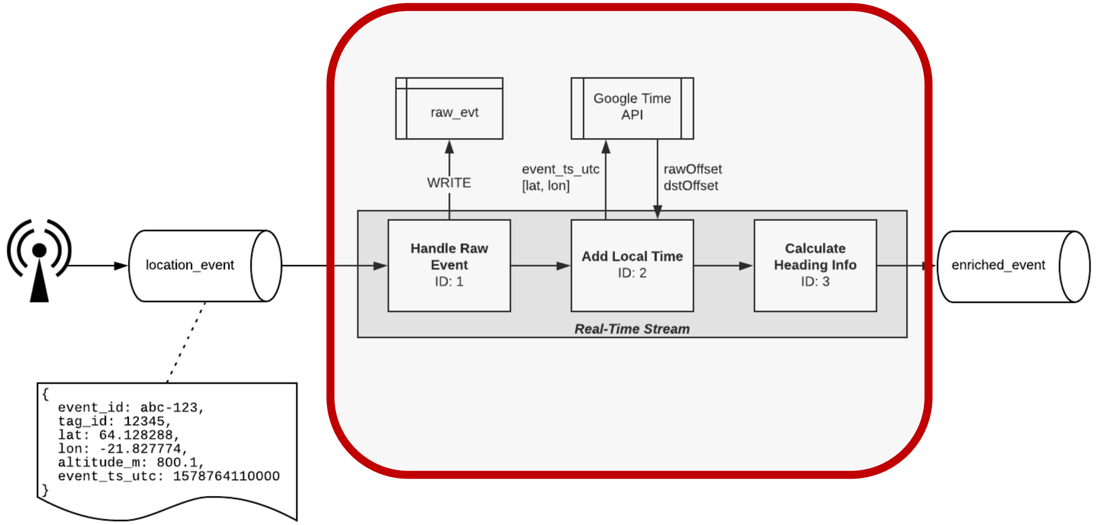
Idea: Use information about pipelines and sources for discoverability, analysis, or governance.

- Measure DAG performance against SLAs
- Identify pipelines that have similar sources
- Use quality provenance to determine commonly problematic sources
- Identify critical components in the data system

Architectural Considerations

- Not all data products require granular provenance
- Provenance can be exponentially larger than the data
- Use a flexible or generic schema
- Fast response times for logging provenance
- Storage considerations

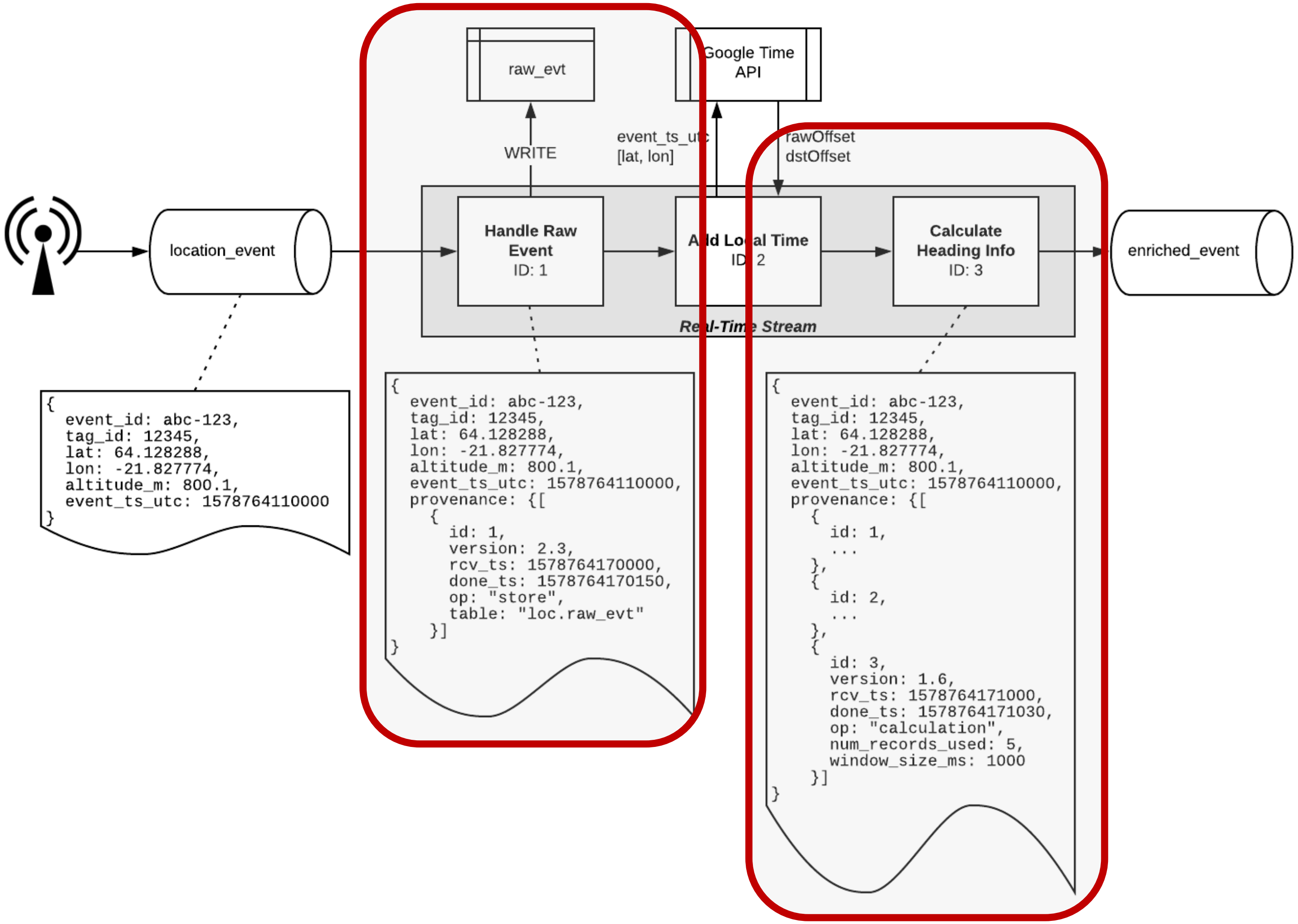
Storage Considerations



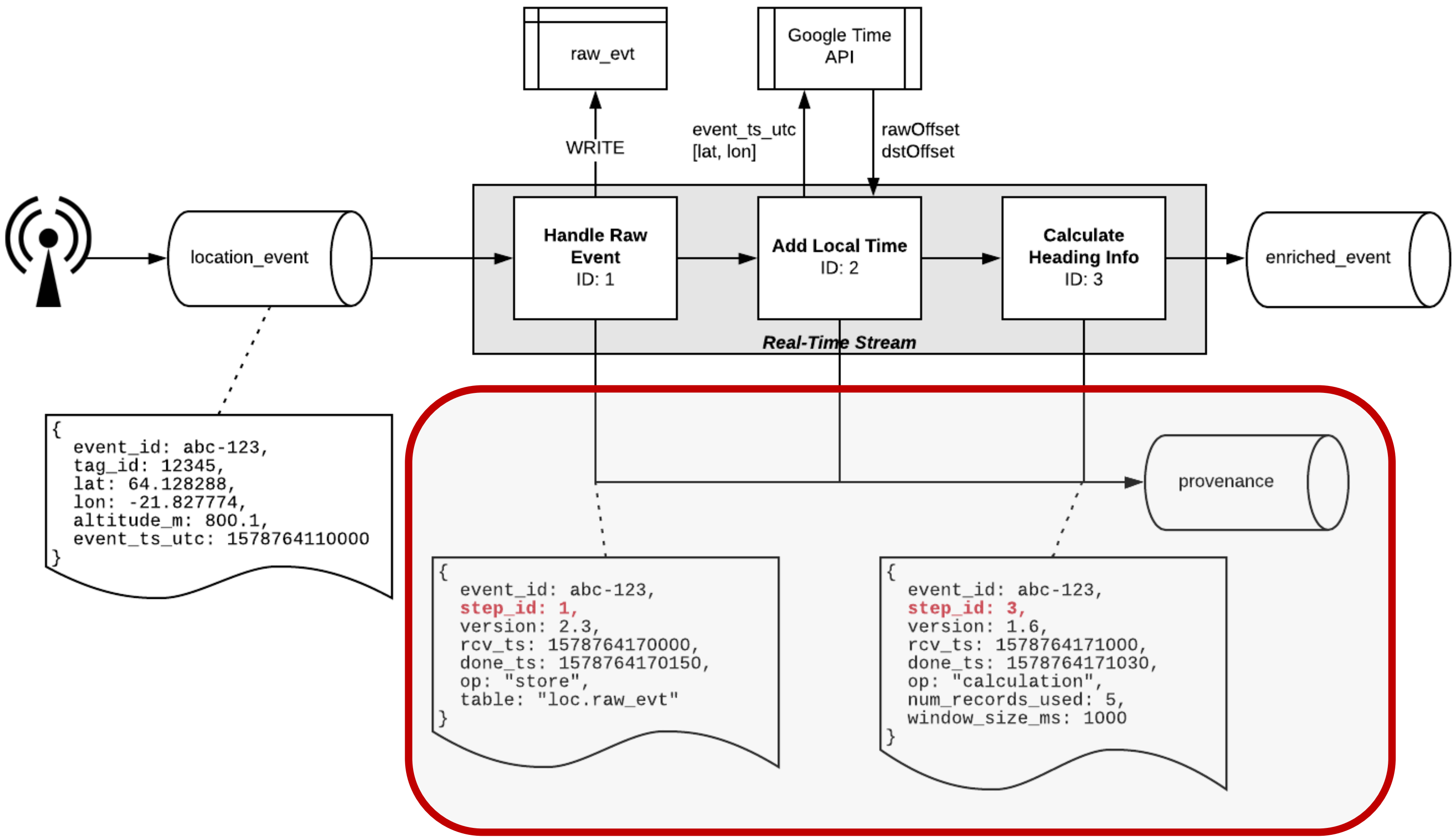
Some Storage Options

1. Attach to record
2. Send as event message(s)
3. Implement provenance API

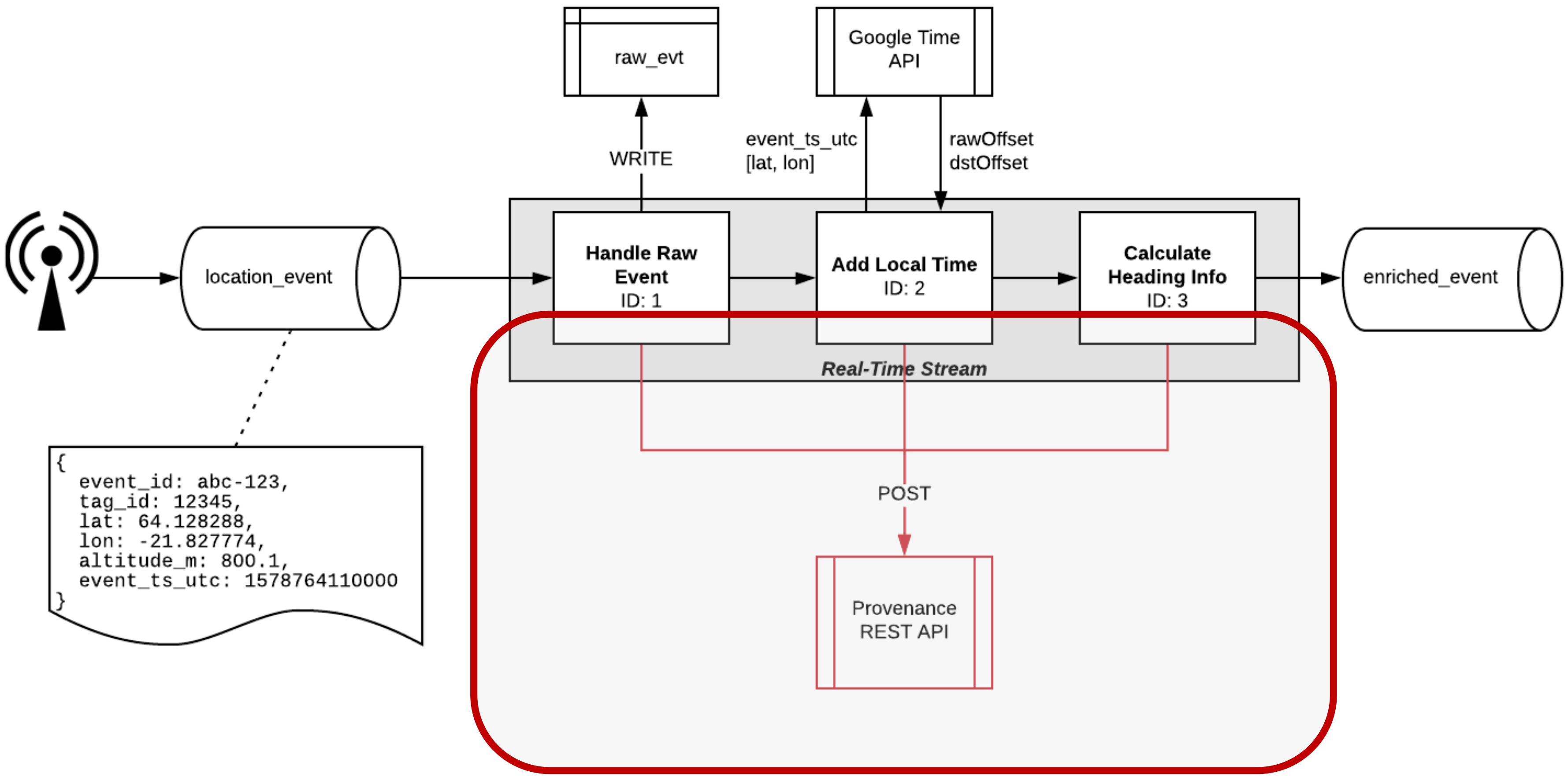
Storage Option: Attach to record



Storage Option: Send as Event Message(s)



Storage Option: Provenance API



Other Considerations

- Some tools already capture some provenance
- Some integration tools enable provenance tracking
  - Amundsen (Lyft)
  - Marquez (WeWork)
  - Databook (Uber)
  - DataHub (LinkedIn)
- Culture
  - Provenance isn’t automatic – build it in
  - Without culture change, provenance is sporadic
  - Must make it trust-worthy

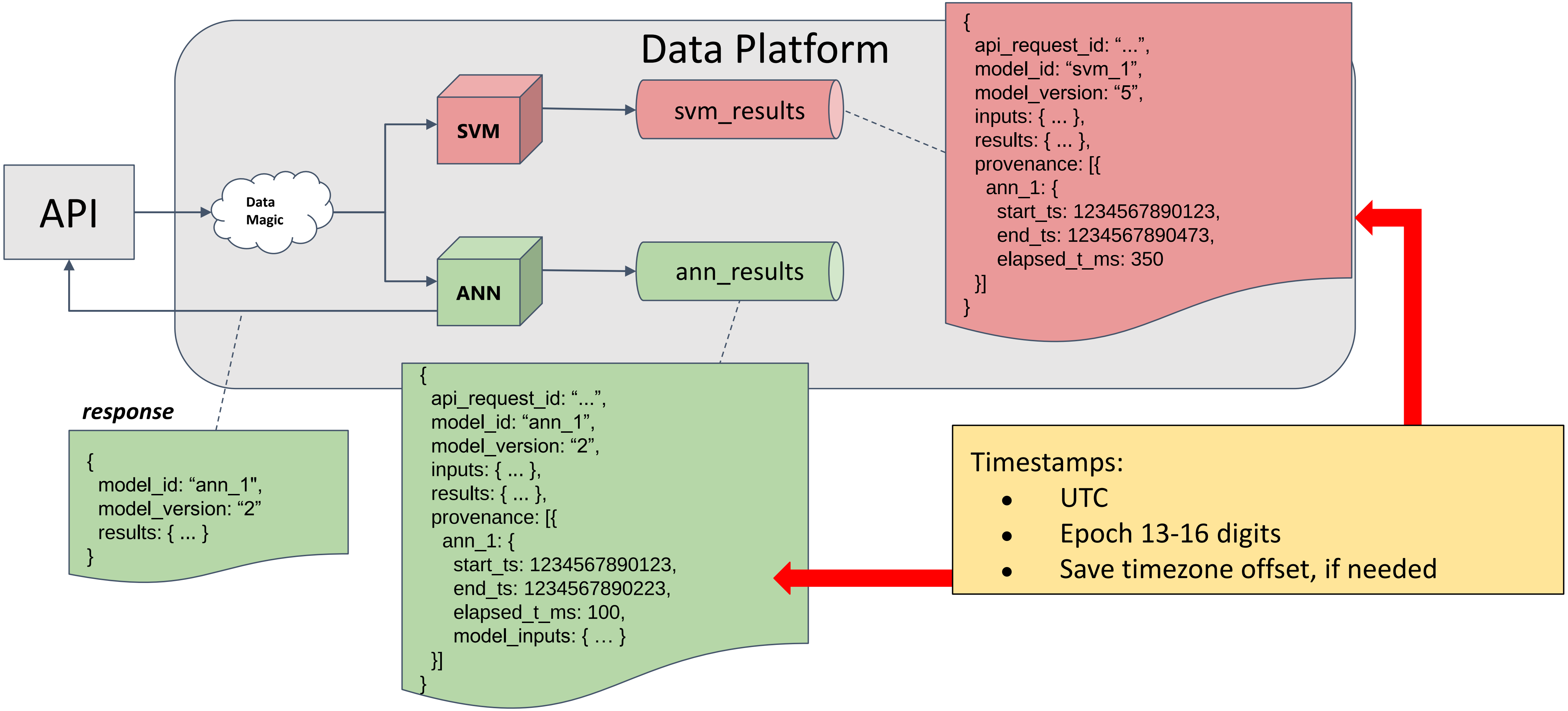
Machine Learning Provenance

Speedy additions for ML Provenance in deployment

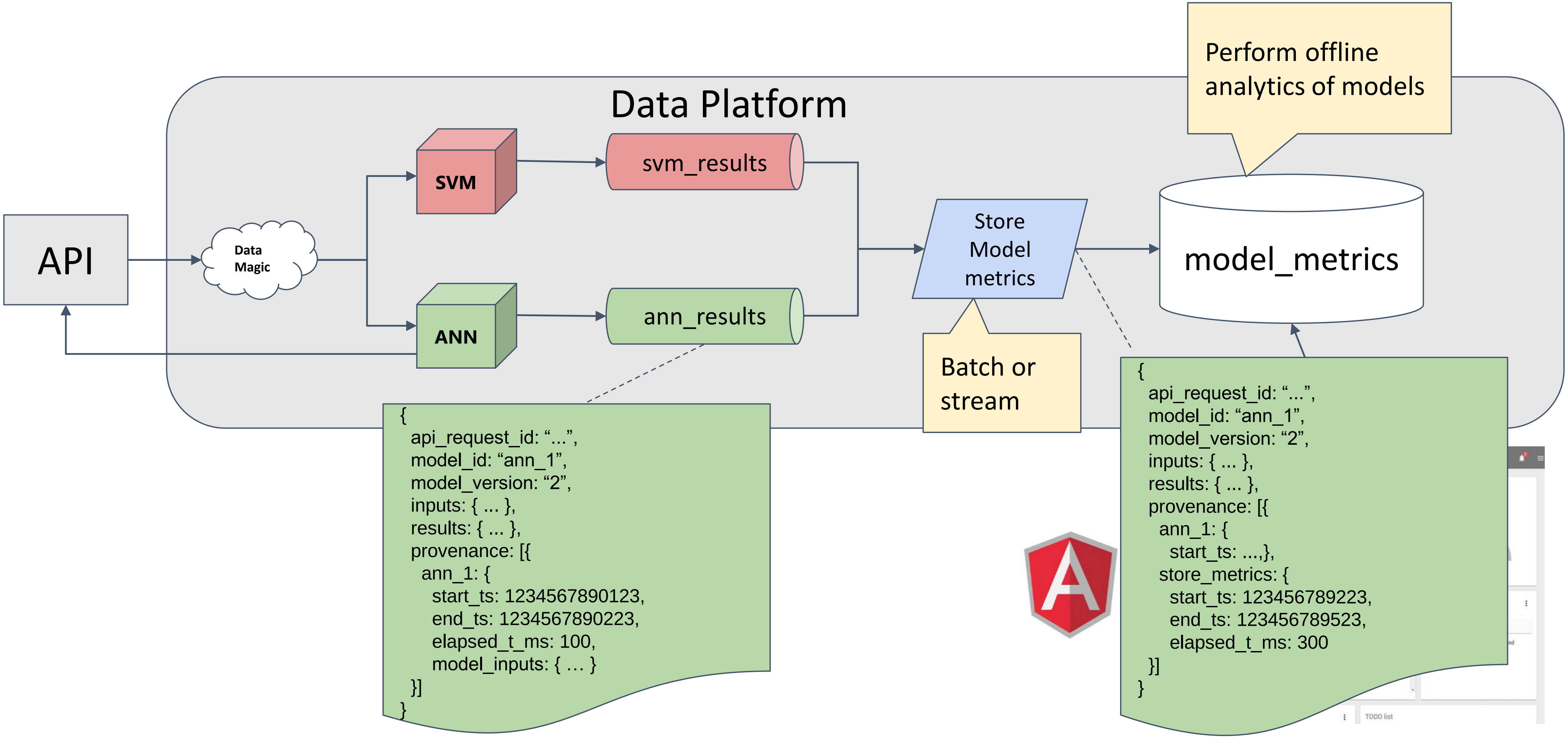
Data Provenance in Machine Learning

- Data provenance in engineering, **plus**
  - Model id, version, name
  - Model algorithm(s) used
  - Model inputs + results
  - Other useful metadata
- Why?
  - Measuring multiple deployed models
  - Reproducability
  - Traceability
  - Analyzing model results (bias, etc.)

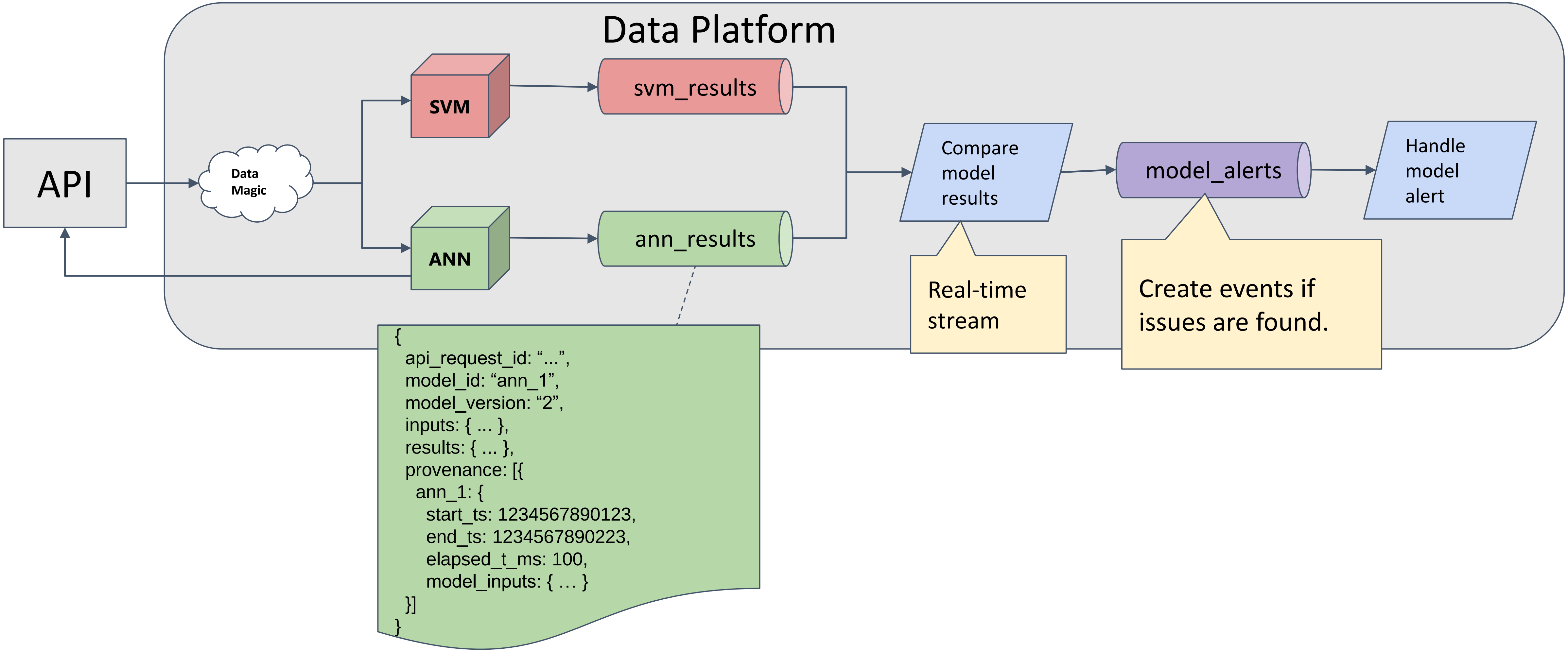
ML Provenance: The Basics

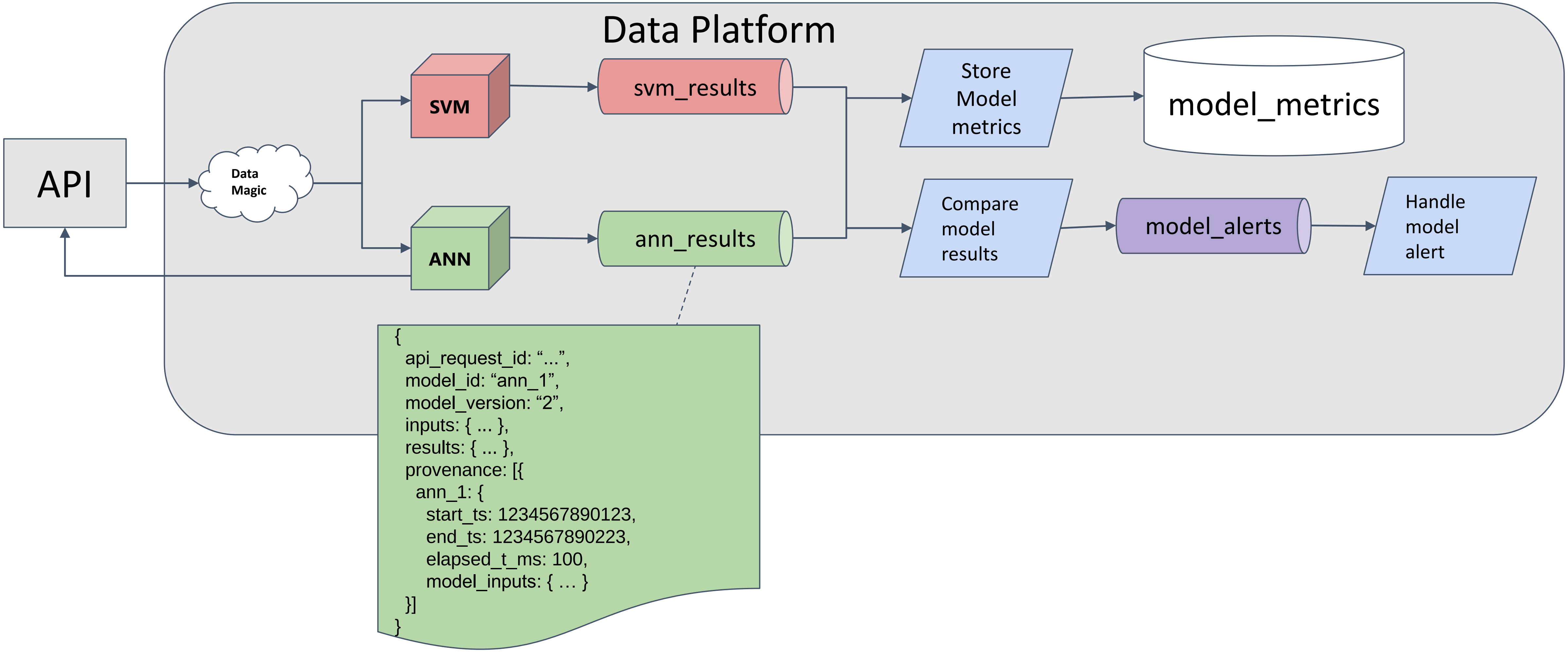


ML Provenance: Store Metrics



ML Provenance: Handling Issues in Models




Data Platform
API
Data
Magic
SVM
ANN
svm_results
ann_results
Store
Model
metrics
model_metrics
Compare
model
results
model_alerts
Handle
model
alert
{
api_request_id: "...",
model_id: "ann_1",
model_version: "2",
inputs: { ... },
results: { ... },
provenance: [{
ann_1: {
start_ts: 1234567890123,
end_ts: 1234567890223,
elapsed_t_ms: 100,
model_inputs: { … }
}]
}


NOW, ARE THERE
ANY QUESTIONS?