Strata NY 2018

September 12, 2018

marmaray

Apache Hadoop Ingestion & Dispersal Framework

Danny Chen dannyc@uber.com,
Omkar Joshi omkar@uber.com
Eric Sayle esayle@uber.com

Uber Hadoop Platform Team

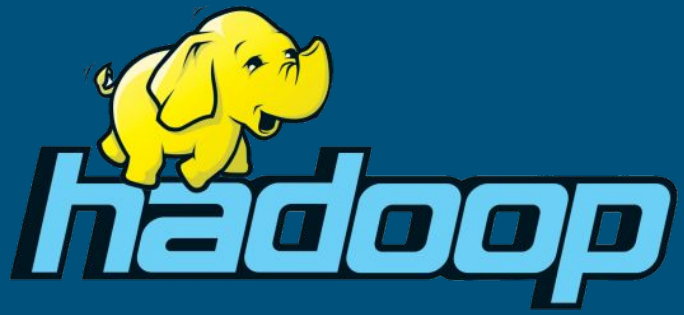
UBER

Agenda

- Mission
- Overview
- Need for Hadoop ingestion & dispersal framework
- Deep Dive
  - High Level Architecture
  - Abstractions and Building Blocks
- Configuration & Monitoring of Jobs
- Completeness & Data Deletion
- Learnings

Uber Apache Hadoop Platform Team Mission

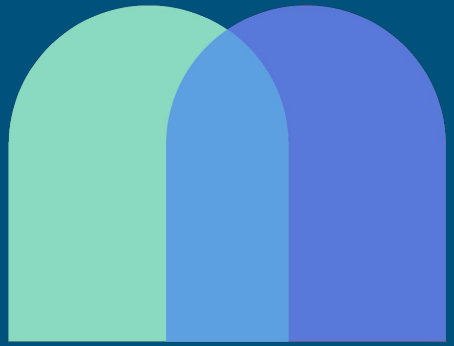
Build products to support reliable, scalable, easy-to-use, compliant, and efficient data transfer (both ingestion & dispersal) as well as data storage leveraging the Hadoop ecosystem.



Overview

- Any Source to Any Sink
- Ease of onboarding
- Business impact & importance of data & data store location
- Suite of Hadoop ecosystem tools

Introducing



marmaray

ANY SOURCE, ANY SINK.

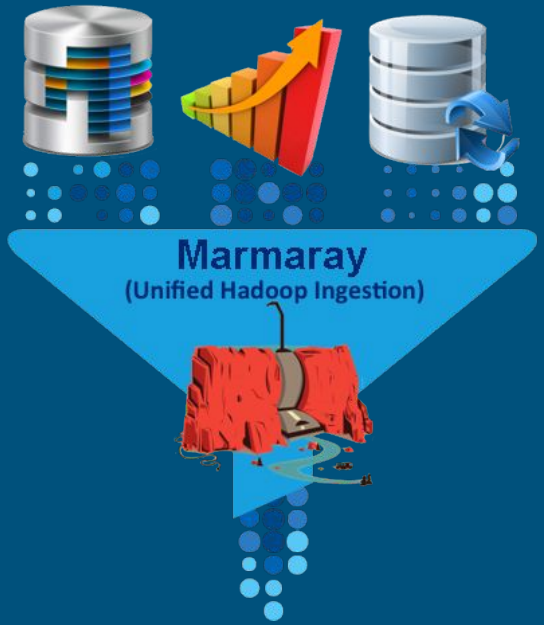
Open Sourced in September 2018

https://github.com/uber/marmaray

Blog Post: https://eng.uber.com/marmaray-hadoop-ingestion-open-source/

Marmaray (Ingestion): Why?

- Raw data needed in Hadoop data lake
- Ingested raw data -> Derived Datasets
- Reliable and correct schematized data
- Maintenance of multiple data pipelines



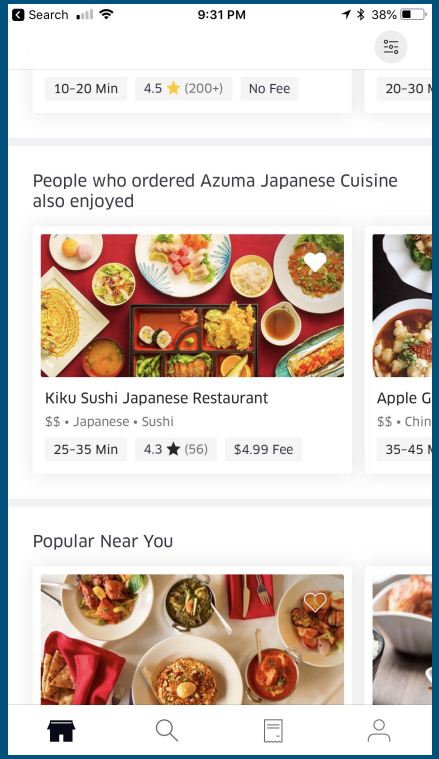
Marmaray (Dispersal): Why?

- Derived datasets in Hive
- Need arose to serve live traffic
- Duplicate and ad hoc dispersal pipelines
- Future dispersal needs

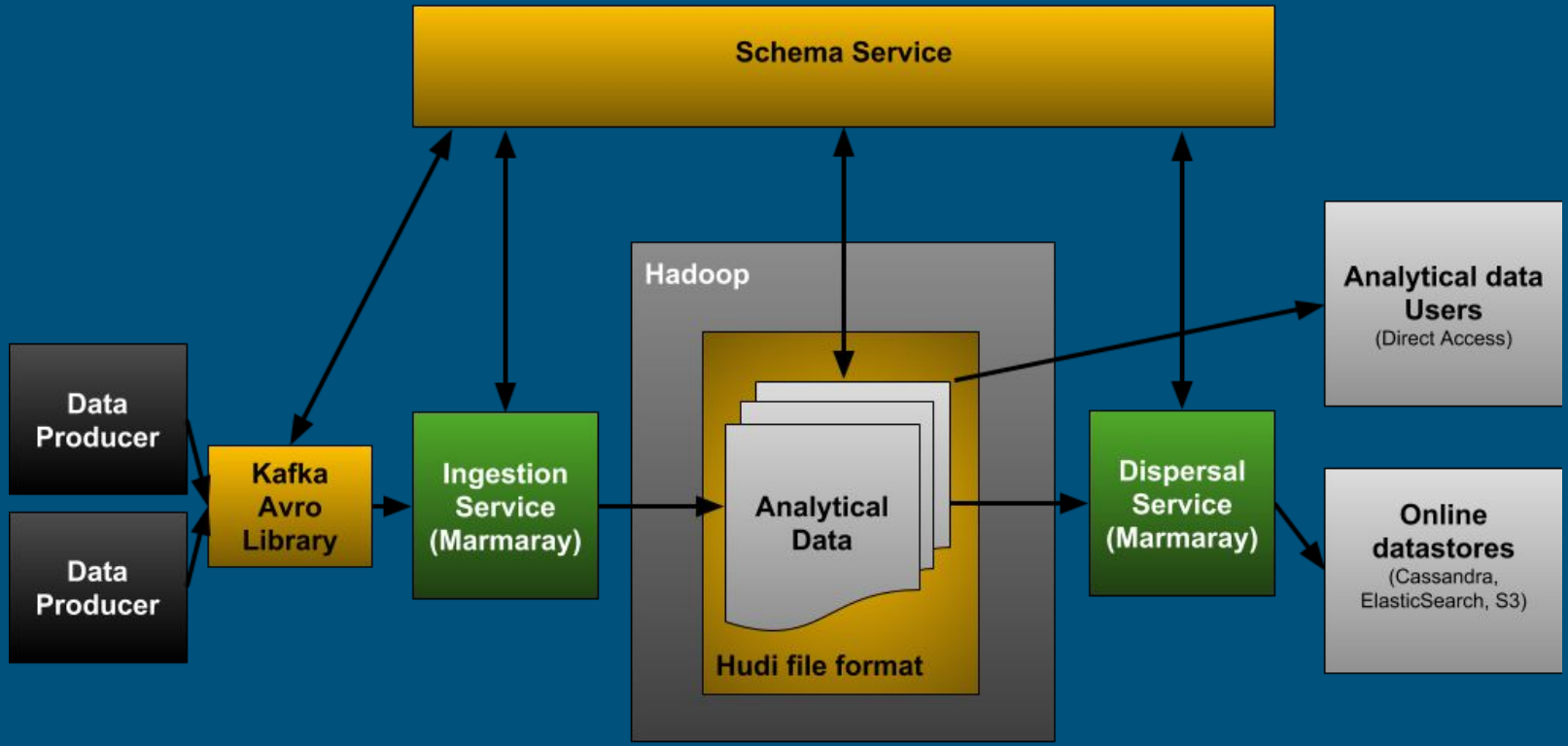
Marmaray: Main Features

- Release to production end of 2017
- Automated schema management
- Integration w/ monitoring & alerting systems
- Fully integrated with workflow orchestration tool
- Extensible architecture
- Open sourced

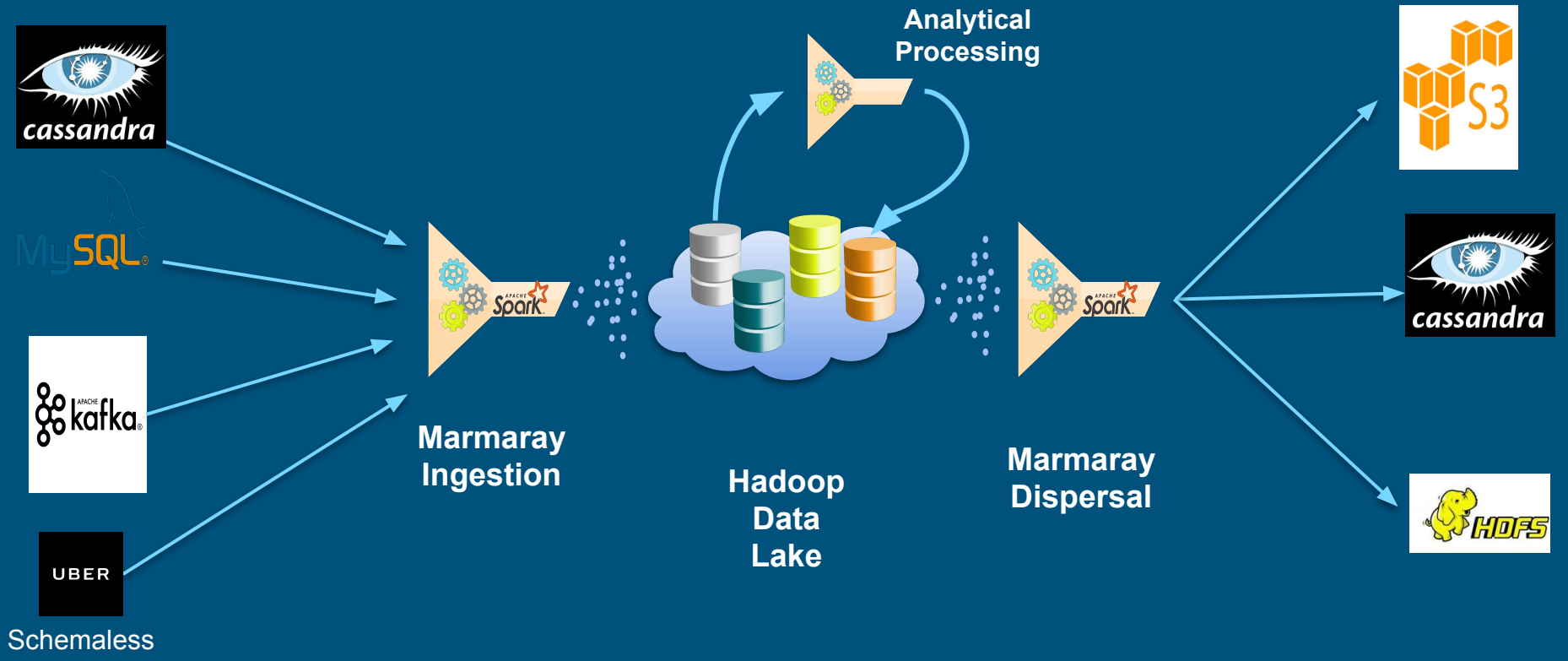
Marmary: Uber Eats Use Case



Hadoop Data Ecosystem at Uber

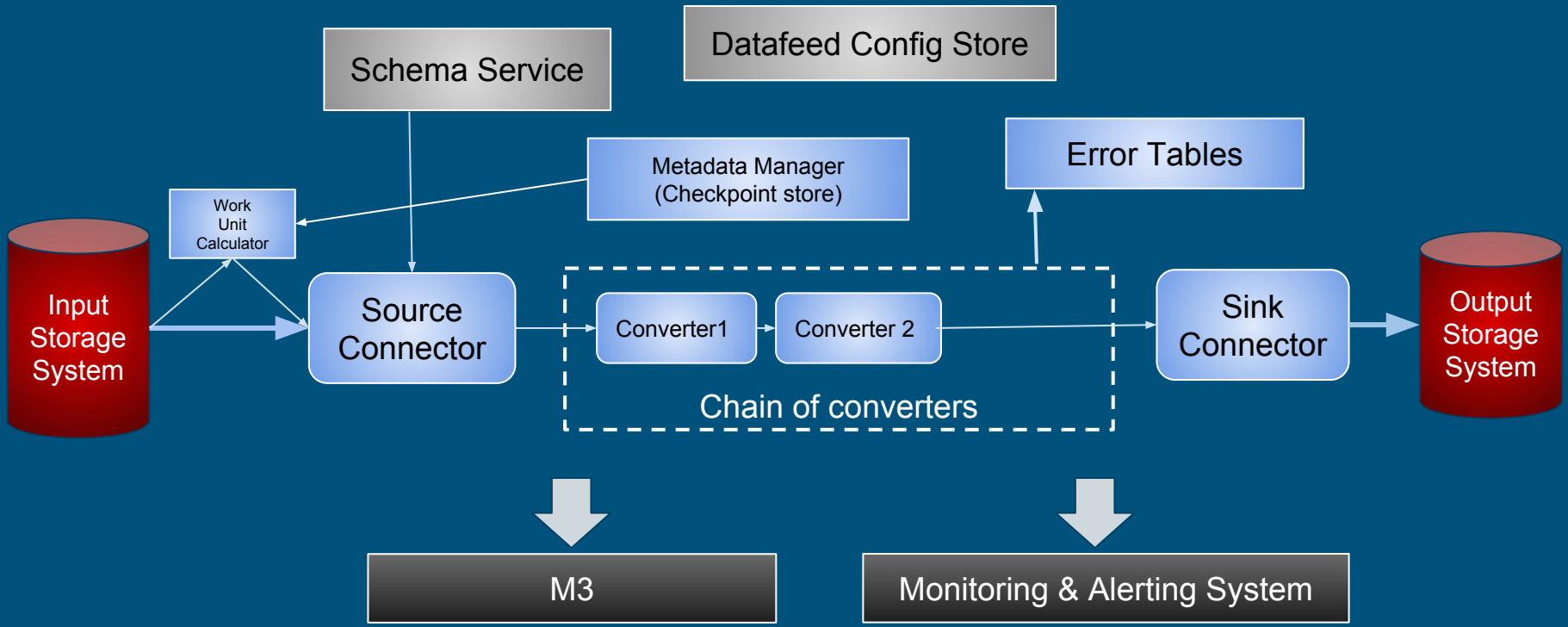


Hadoop Data Ecosystem at Uber

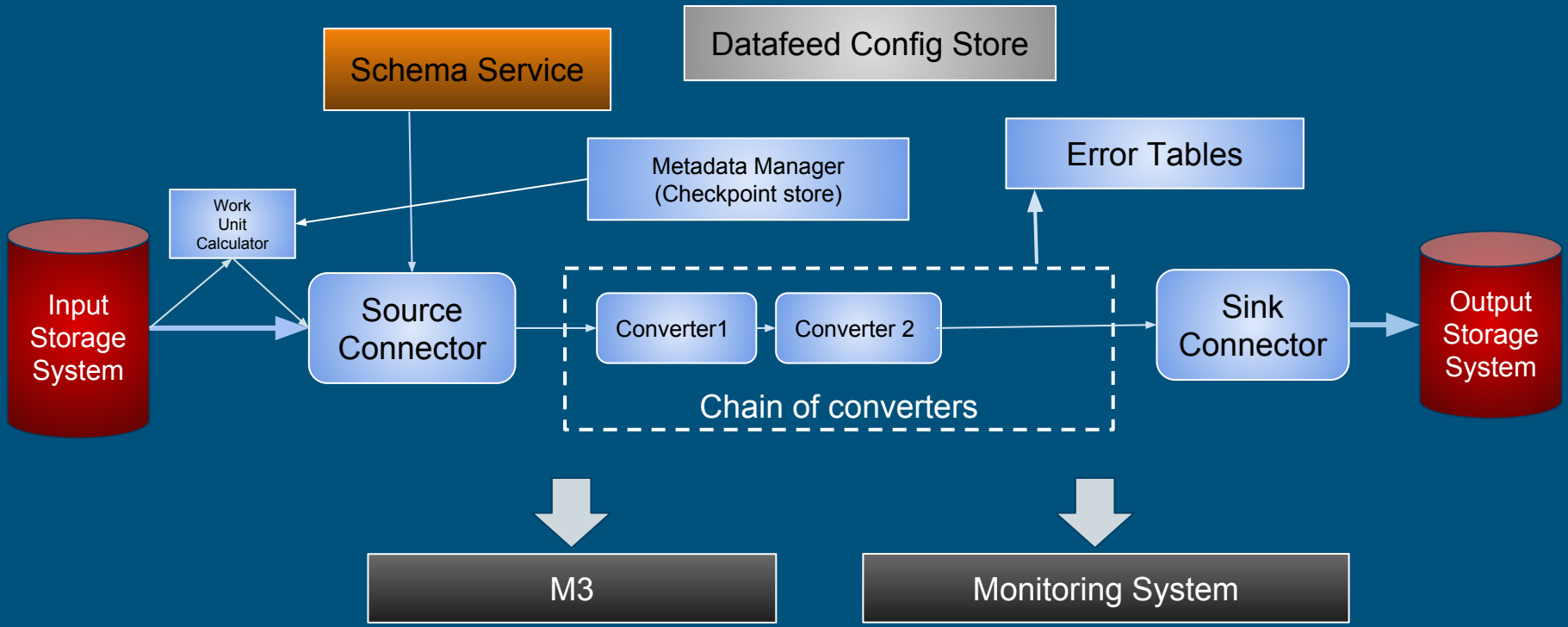


High-Level Architecture & Technical Deep Dive

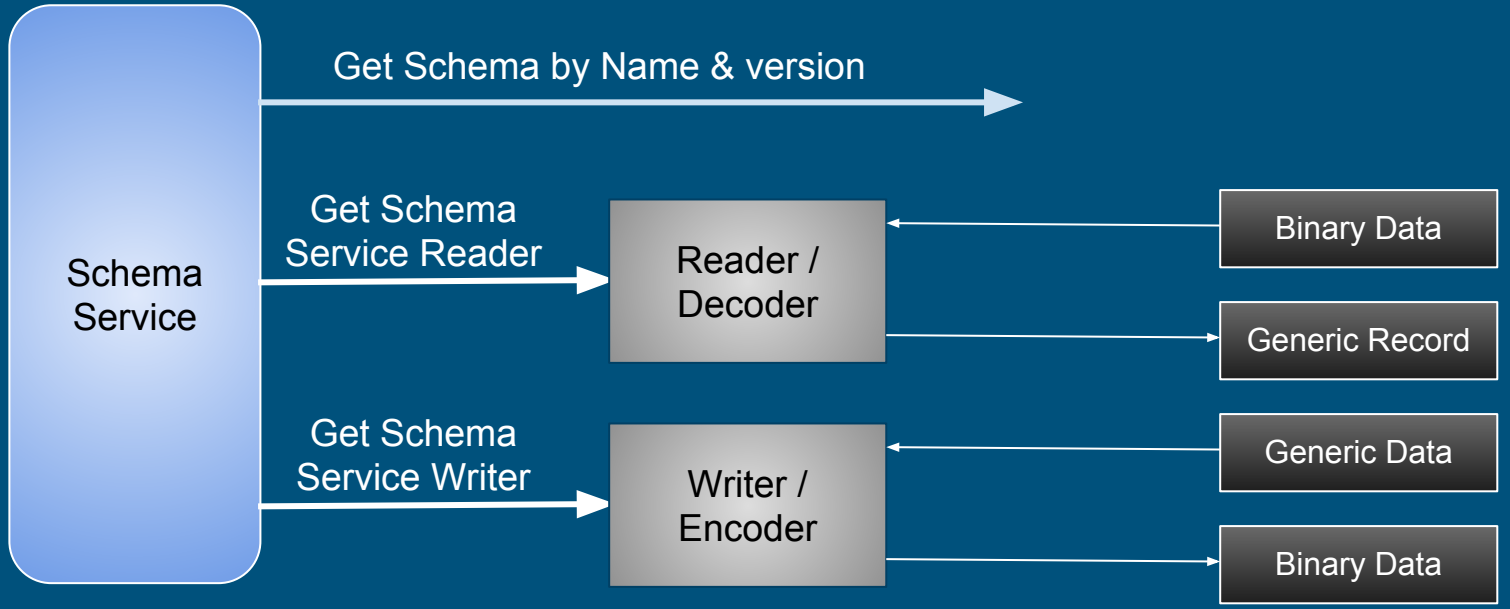
High-Level Architecture



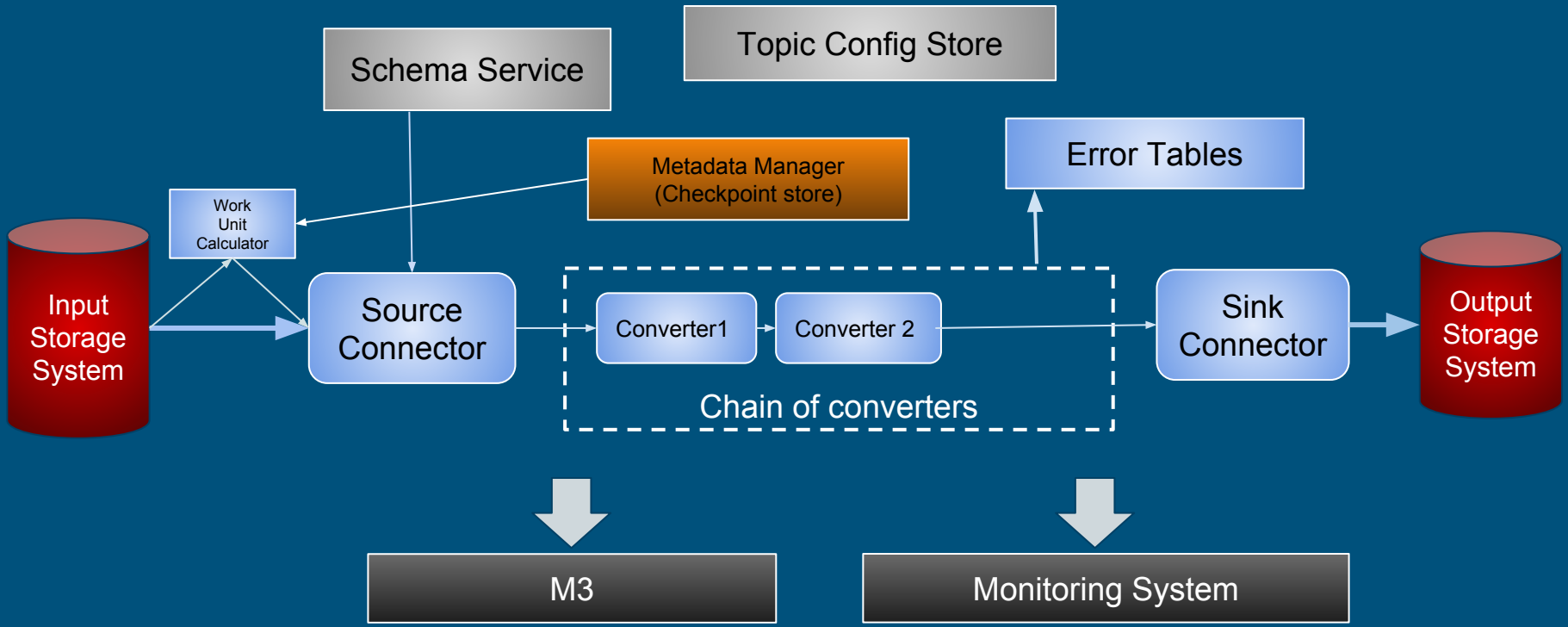
High-Level Architecture



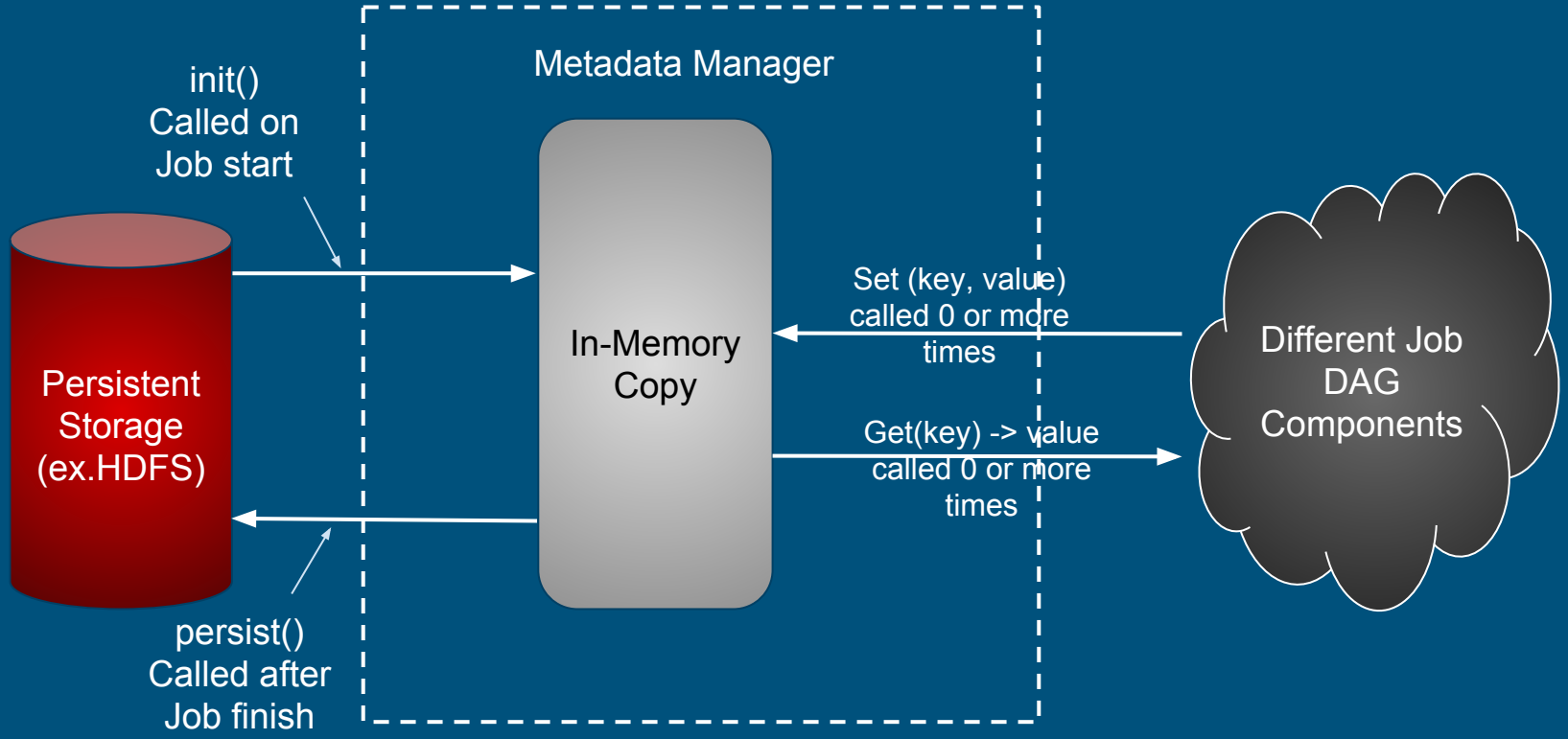
Schema Service



High-Level Architecture

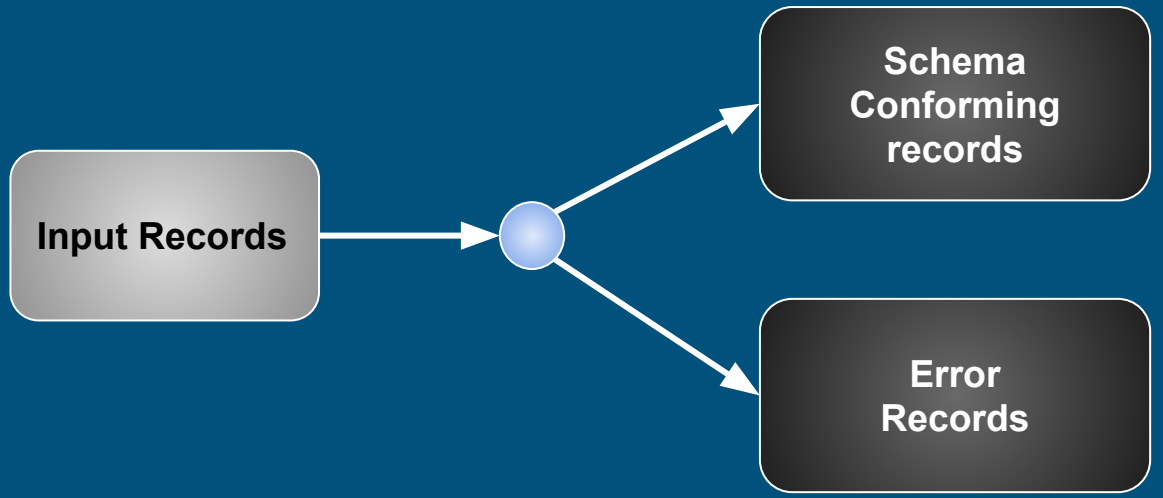


Metadata Manager

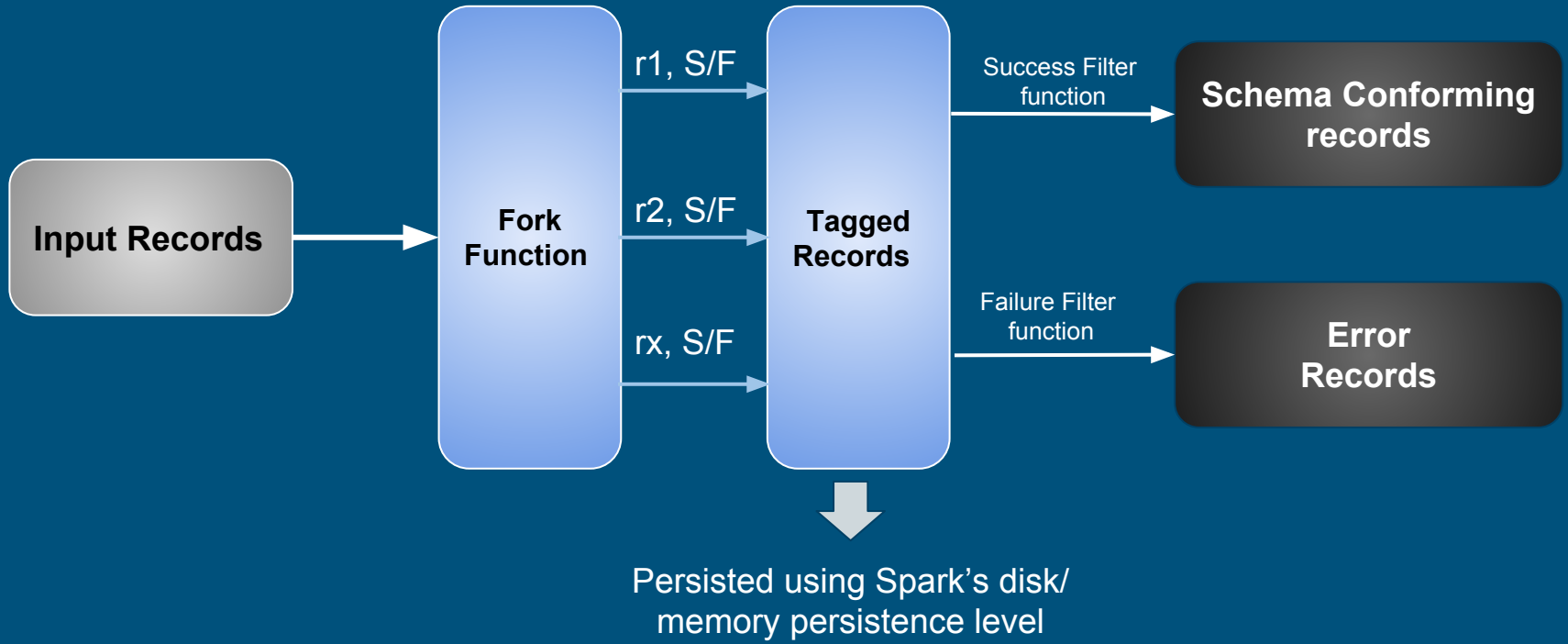


Fork Operator - Why is it needed?

- Avoid reprocessing input records
- Avoid re-reading input records (or in Spark, re-executing input transformations)

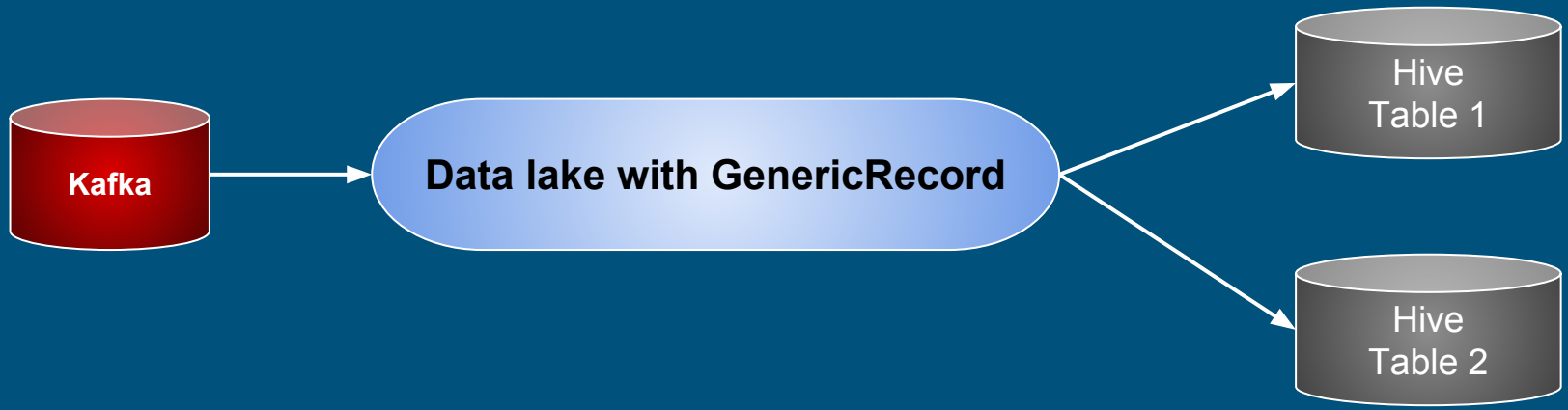


Fork Operator & Fork Function



Easy to Add Support for new Source & Sink

Support for Writing into Multiple Systems



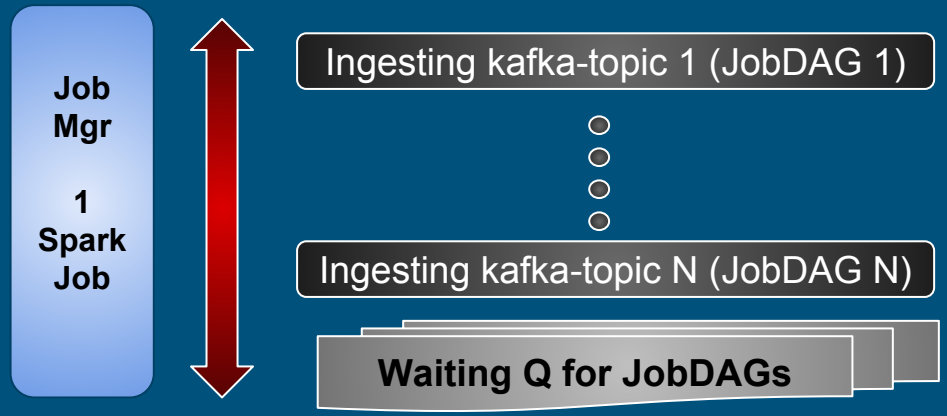
JobDag & JobDagActions

Need for running multiple JobDags together

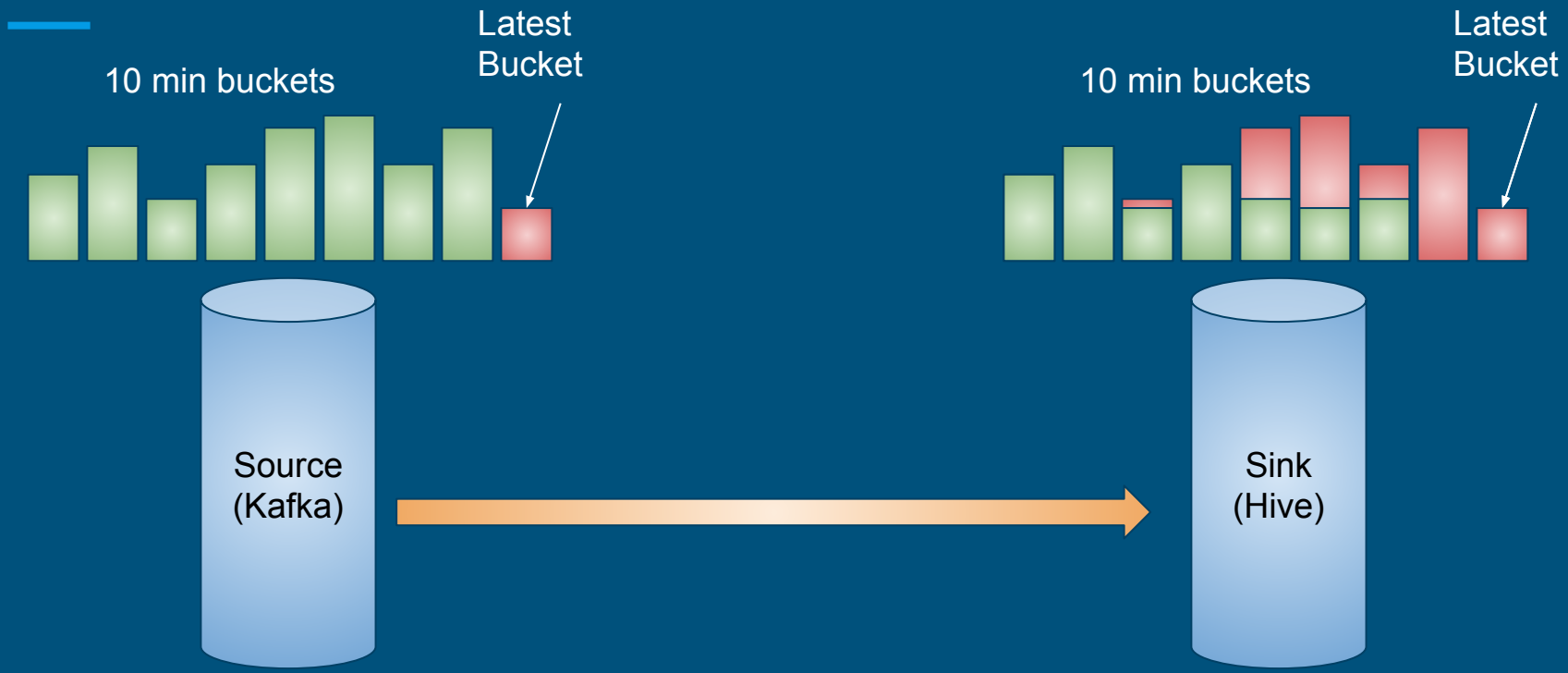
- Frequency of data arrival
- Number of messages
- Avg record size & complexity of schema
- Spark job has Driver + executors (1 or more)
- Not efficient model to handle spikes
- Too many topics to ingest. 2000+

JobManager

- Single Spark job for running ingestion for 300+ topics
- Executes multiple JobDAGs
- Manages execution ordering for multiple JobDAGs
- Manages shared Spark context
- Enables job and tier-level locking



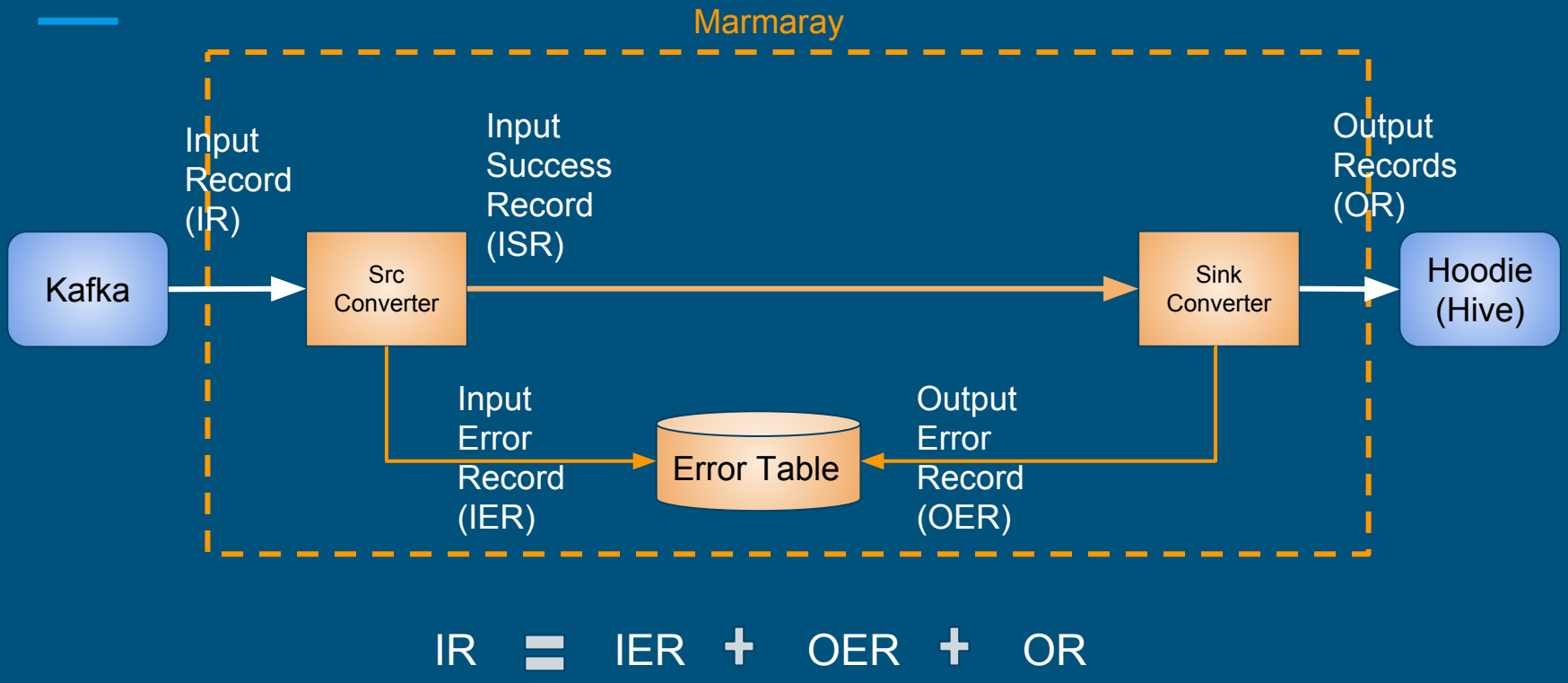
Completeness



Completeness contd..

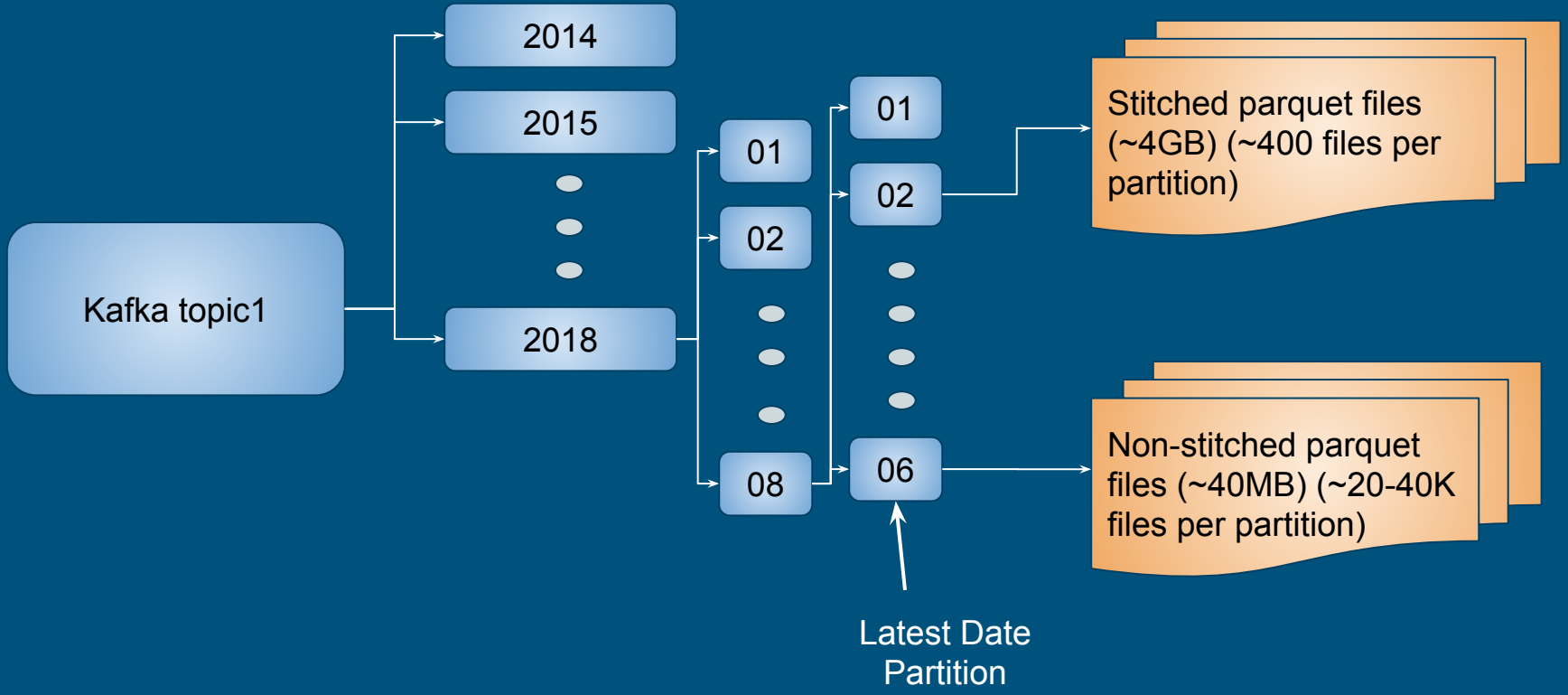
- Why not run queries on source and sink dataset periodically?
  - Possible for very small datasets
  - Won’t work for billions of records; very expensive!!
- Bucketizing records
  - How about creating time based buckets say for every 2min or 10min.
  - Count records at source and sink during every runs
    - Does it give 100% guarantee?? No but w.h.p. it is close to it.

Completeness - High level approach



IR = IER + OER + OR

Hadoop old way of storing kafka data

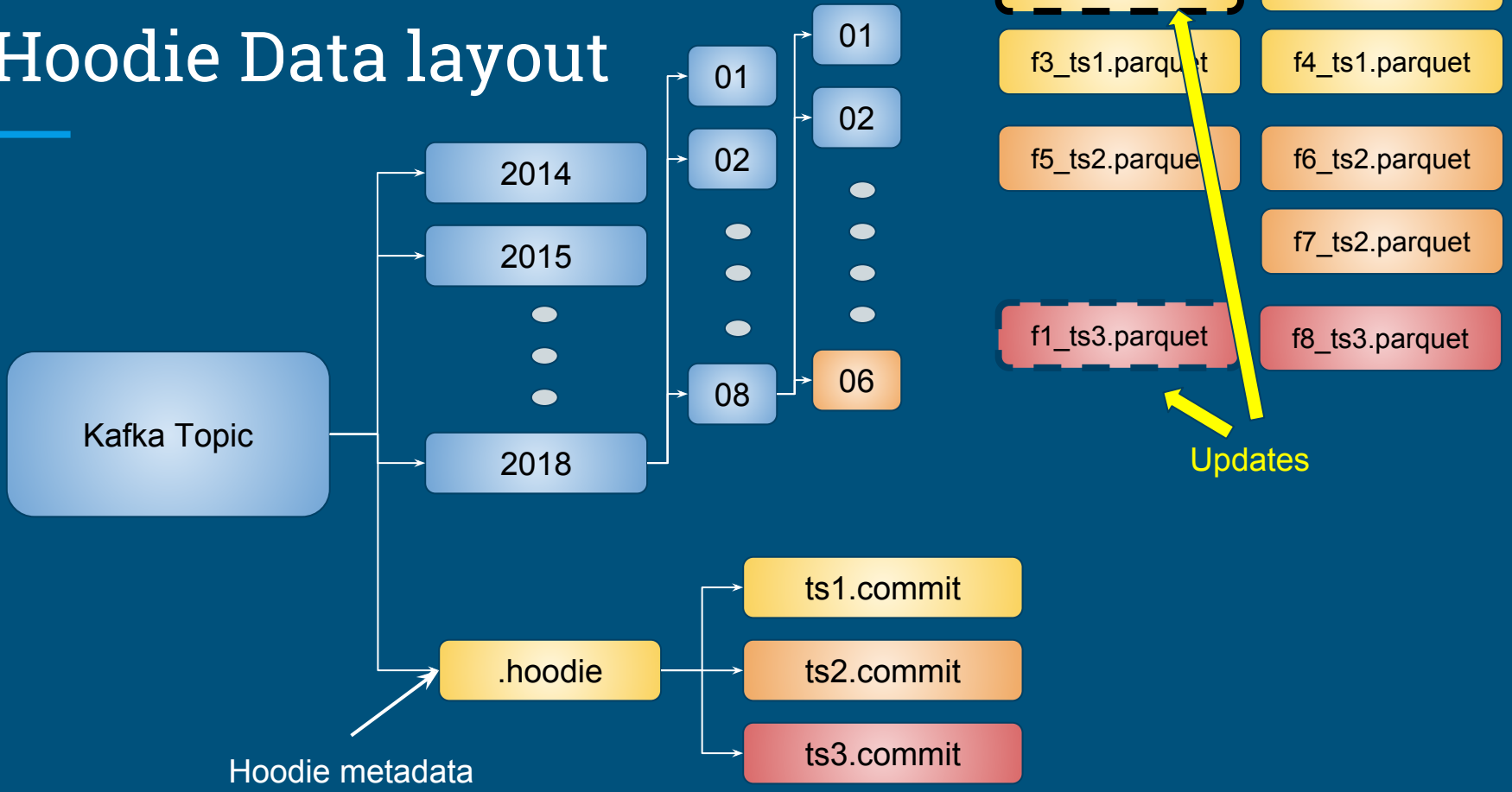


Data Deletion (Kafka)

- Old architecture is designed to be append/read only
- No indexes
  - Need to scan entire partition to find out if record is present or not
- Only way to update is to rewrite entire partition
  - Re-writing entire partition for
- GDPR requires all data to be cleaned up once user requests deletion
- This is a big architectural change and many companies are struggling to solve this

Marmaray + HUDI (hoodie) to rescue

Hoodie Data layout



Configuration

```
common:
  hadoop:
    fs.defaultFS: "hdfs://namenode/"
  hoodie:
    table_name: "mydb.table1"
    base_path: "/path/to/my.db/table1"
    metrics_prefix: "marmaray"
    enable_metrics: true
    parallelism: 64
  kafka:
    conn:
      bootstrap.servers: "kafkanode1:9092,kafkanode2:9092"
      fetch.wait.max.ms: 1000
      socket.receive.buffer.bytes: 5242880
      fetch.message.max.bytes: 20971520
      auto.commit.enable: false
      fetch.min.bytes: 5242880
  source:
    topic_name: "topic1"
    max_messages: 1024
    read_parallelism: 64
  error_table:
    enabled: true
    dest_path: "/path/to/my.db/table1/.error"
    date_partitioned: true
```

Monitoring & Alerting

Learnings

- Spark
  - Off heap memory usage of spark and YARN killing our containers
  - External shuffle server overloading
- Parquet
  - Better record compression with column alignments
- Kafka
  - Be gentle while reading from kafka brokers
- Cassandra
  - Cassandra SSTable streaming (no throttling) , no monitoring
  - No backfill for dispersal

External Acknowledgments

https://gobblin.readthedocs.io/en/latest/

Other Relevant Talks

*Your 5 billion rides are arriving now: Scaling Apache Spark for data pipelines and intelligent systems at Ube*r - Wed 11:20am

Hudi: Unifying storage and serving for batch and near-real-time analytics - Wed 5:25 pm

We are hiring!

marmaray

Positions available: **Seattle, Palo Alto** & **San Francisco**

email : hadoop-platform-jobs@uber.com

Useful links

- https://github.com/uber/marmaray
- https://eng.uber.com/marmaray-hadoop-ingestion-open-source/
- https://github.com/uber/hudi
- https://eng.uber.com/michelangelo/
- https://eng.uber.com/m3/

marmaray

Q & A?

Thank you

Questions: email ospo@uber.com

Follow our Facebook page:
www.facebook.com/uberopensource