ClickHouse

CONFLUENT

Meetup

Work Club @ 8 Chifley

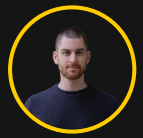
Sydney, Australia

June 19, 2025 at 5:30 PM AEST

Tech Talks

Maximising Analytics with ClickHouse and Kafka Integration

Johnny Mirza, ClickHouse Solution Architect, & Ravi Bhardwaj, Confluent Principal Technical Support Engineer



Nightwatch: How Laravel Monitors Billions of Events a Month with ClickHouse

James Carpenter, Senior Infrastructure Engineer @ Laravel

Real-Time CDC at Scale: Building Secure Data Pipelines with ClickHouse Cloud and AWS PrivateLink

Peter Hanssens, Founder of DataEngBytes & Founder and Principal Consultant @ Cloud Shuttle

Maximising Analytics with ClickHouse and Kafka Integration

Powering Enterprise AI at Scale

2025

ClickHouse

Ravi Bhardwaj

Technical Support Engineer, Confluent

ravi-bhardwaj

ravib777

- Full Stack Trace Engineer by trade
- 15 years in Middleware & Messaging space

Johnny Mirza

Solution Architect @ ClickHouse

johnnym

jmirza-debug

- Enthusiastic about Big Data and Analytics
- Excited about the future of AI
- Marketing, Community Support, DevRel etc..

Agenda

01 Introduction

02 Apache Kafka 101

03 Integrating Kafka with ClickHouse

04 Recent improvements to Kafka Support

05 Demo

06 Questions

Introduction ClickHouse

ClickHouse

The Most Popular Analytics Database on the Planet

#1
Analytics DB on DB-Engines

Over
39,000
GitHub Stars

Over
200,000
Community Members

Key Features

Some of the cool things ClickHouse can do

1 Speaks SQL

Most SQL-compatible UIs, editors, applications, frameworks will just work!

2 Lots of writes

Up to several million writes per second - per server.

3 Distributed

Replicated and sharded, largest known cluster is 4000 servers.

4 Highly efficient storage

Lots of encoding and compression options - e.g. 20x from uncompressed CSV.

5 Very fast queries

Scan and process even billions of rows per second and use vectorized query execution.

6 Joins and lookups

Allows separating fact and dimension tables in a star schema.

Use cases

Real-time Analytics

Power interactive applications and dashboards that analyze and aggregate large amounts of data on the fly. Run complex internal analytics in ms, not mins or hrs.

Disney+ · CLOUDFLARE · Microsoft · lyft · Contentsquare · highlight.io

Logs, events, traces

Monitor with confidence your logs, events, and traces. Detect anomalies, fraud, network or infrastructure issues, and more.

Darwinium · ebay · BENOCS · resmo · zomato · runreveal

Business intelligence

Interactively slice and dice your data for analysis, reporting, and building internal applications. Evaluate user behaviors, ad and media perf, market dynamics, and more.

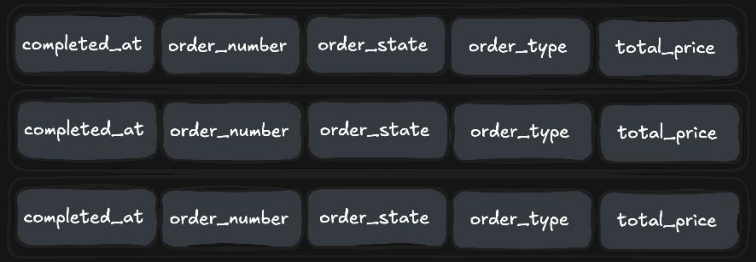
ROKT · Deutsche Bank · QuickCheck · NANO CORP. · TrillaBit · HIFI

ML and Gen AI

Execute fast and efficient vector search. Plug-and-play Generative AI models from any provider. Use lightning-fast aggregations to power model training at petabyte scale.

denic · DeepL · ADMIXER · ensemble

Row vs Column



Row-based

Excess disk Reads We need data in column, but it's stored in rows. So entire rows are read.

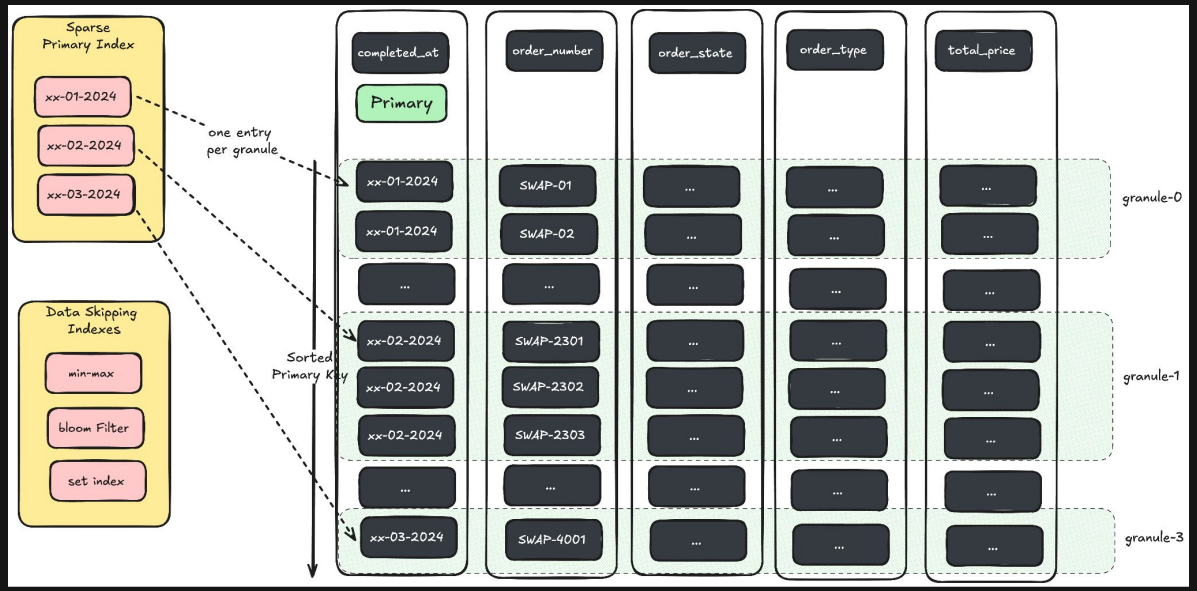
Column-Based

Column Pruning it only pick the subset of columns that are openly declared in the SQL query, thus eliminating 99% columns (990 of 1000 columns) from the disk read consideration.

Vectorized Execution
data processed in **large arrays**, which could be the entire length of the column loaded into RAM.

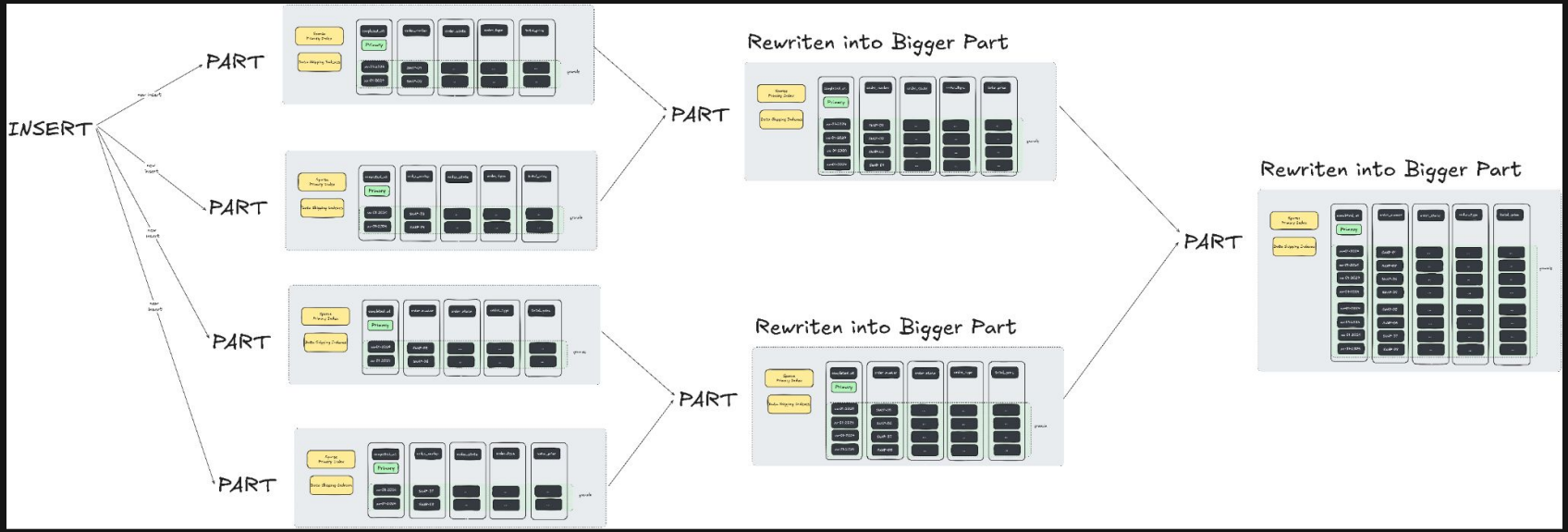
+ **reduces CPU cache miss rates**

Multiple Layers of Indexes



- **Granules** are chunks of rows, and **ClickHouse** groups rows into granules based on the index_granularity setting.
- **Sparse Primary Index**: **ClickHouse** uses the *sparse primary index* to quickly jump to the relevant granules, skipping over large portions of the data that don't need to be read.
- **Skip Indexes**: These allow **ClickHouse** to skip entire granules if it can determine from the index that no rows in the granule satisfy the query's filter.

Efficient Merging

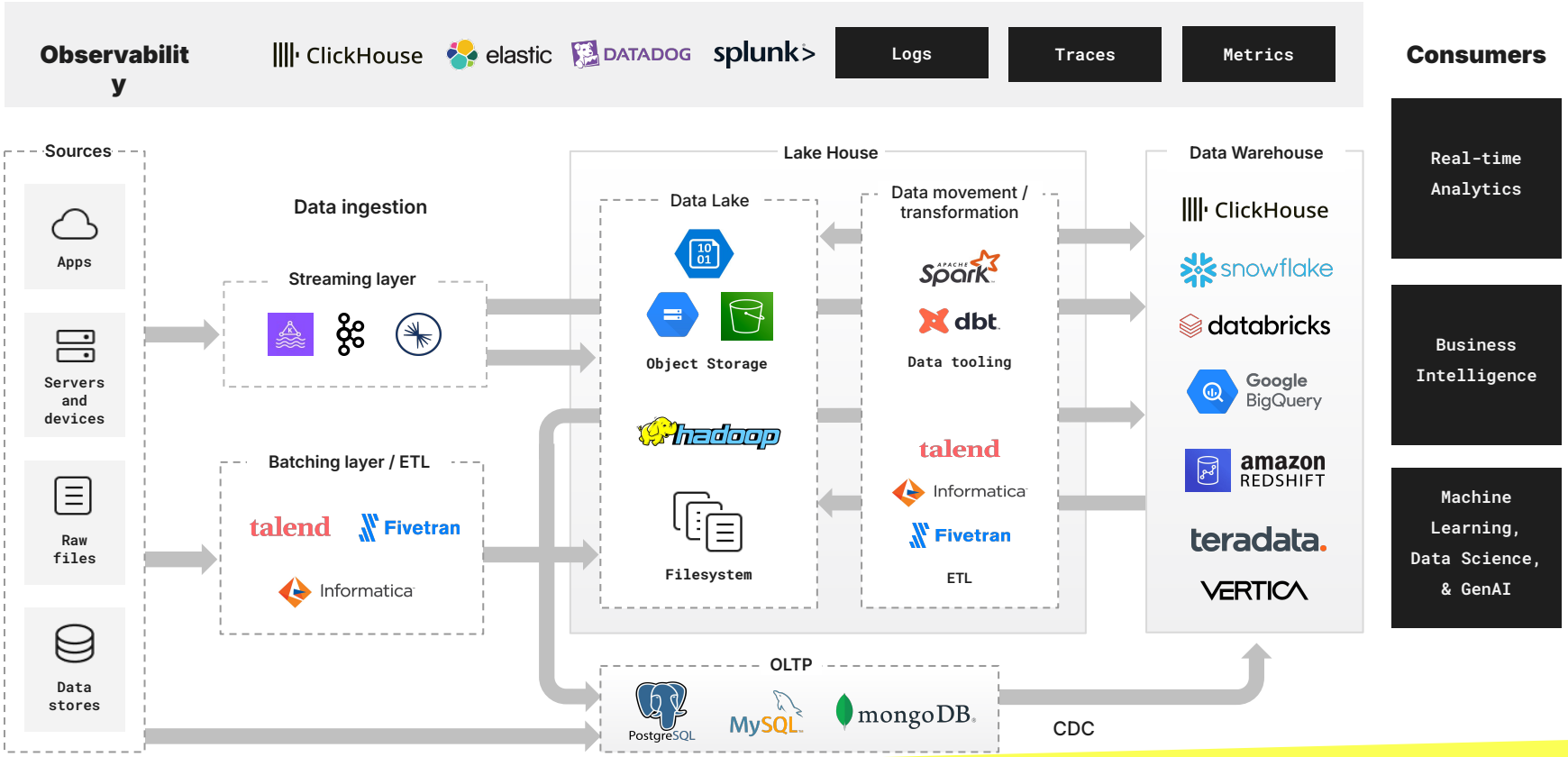


Parts: Each time data is inserted, a new part is created. Multiple parts can accumulate over time, which can slow down query performance.

To improve performance and reduce the number of parts, **ClickHouse** periodically runs merging operations.

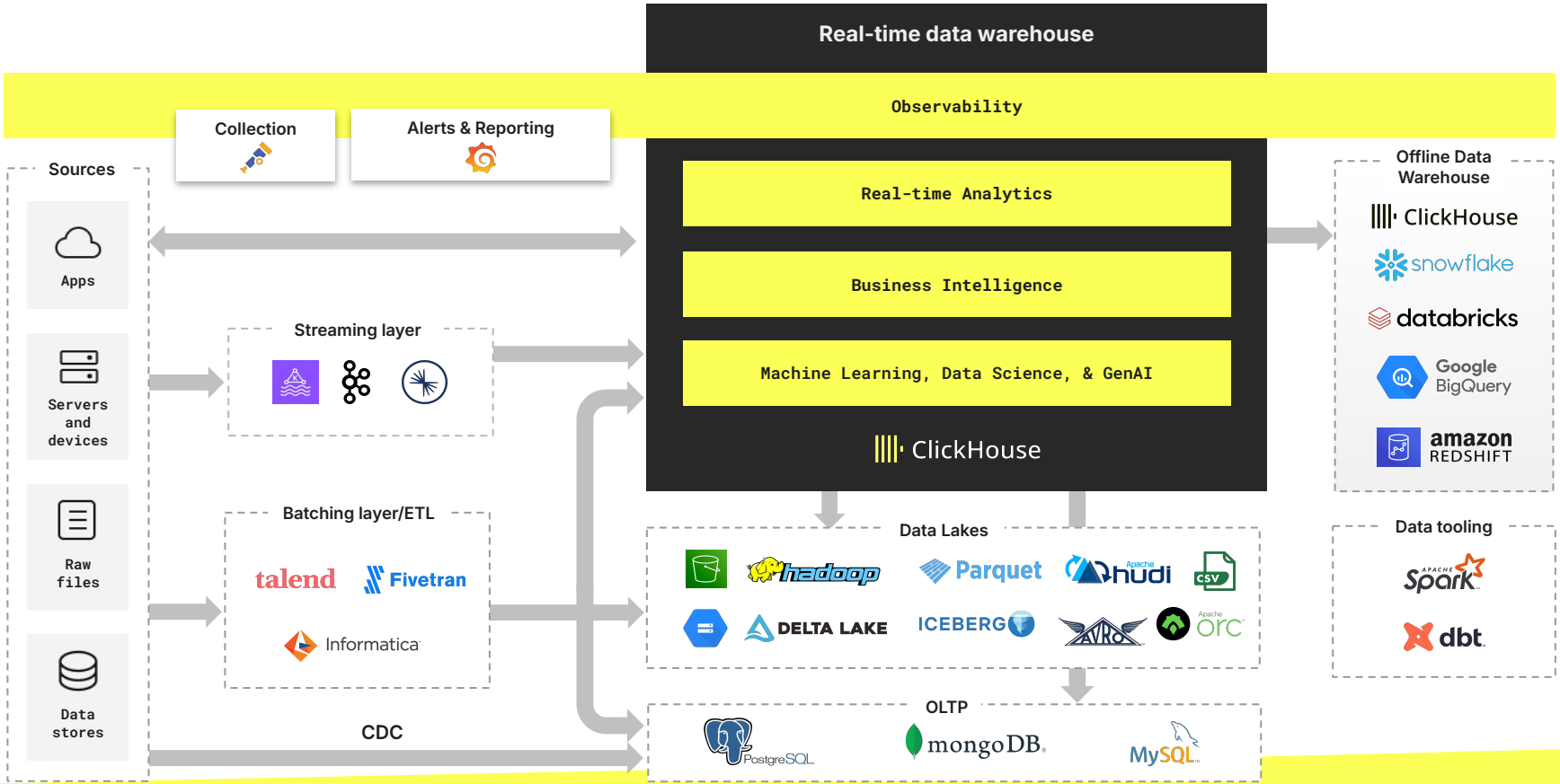
Traditional Data Architecture

Operationally complex with skyrocketing expense



Modern Data Architecture

Reduces system bloat and increases resource efficiency

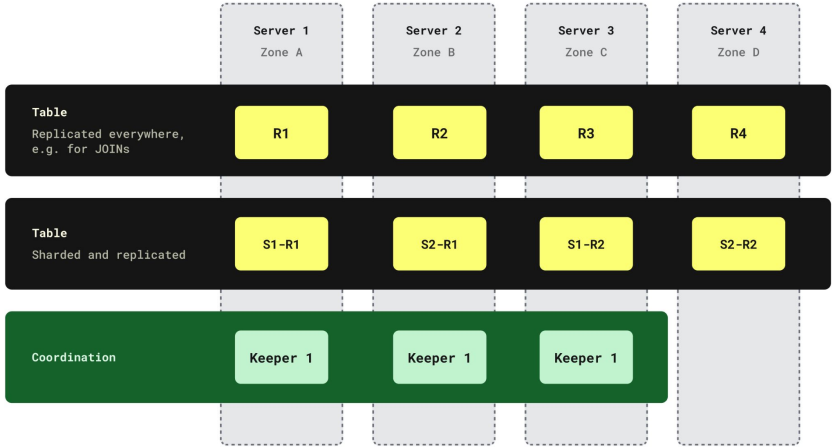


ClickHouse

Self-managed

- ✔ Open-source
- ✔ Flexible architecture
- ✔ Efficient and robust
- ✔ Support contracts available

Sample self-managed architecture

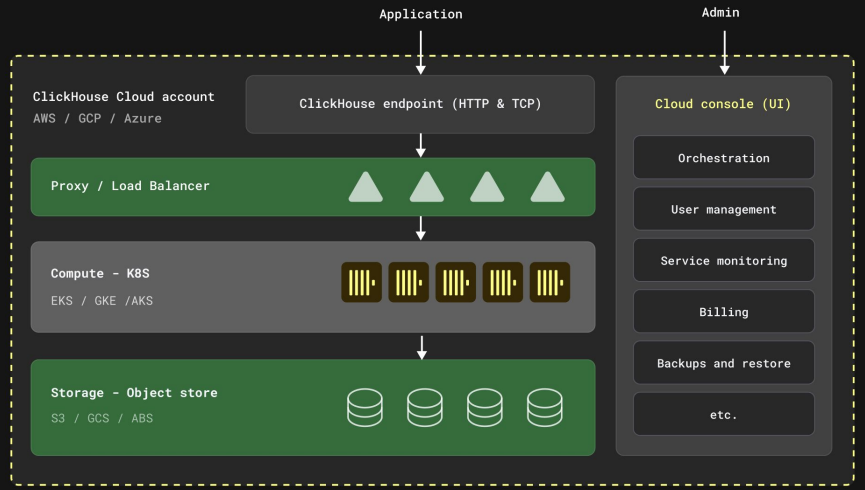


ClickHouse

Cloud

✔ Easy to use	Managed for you
✔ Feature-rich	Cloud-first features & tooling
✔ Fast	Automatically maximizes performance/efficiency
✔ Scalable	Scale seamlessly
✔ Reliable	Ensure reliability
✔ PAYG	SaaS usage and capacity based pricing

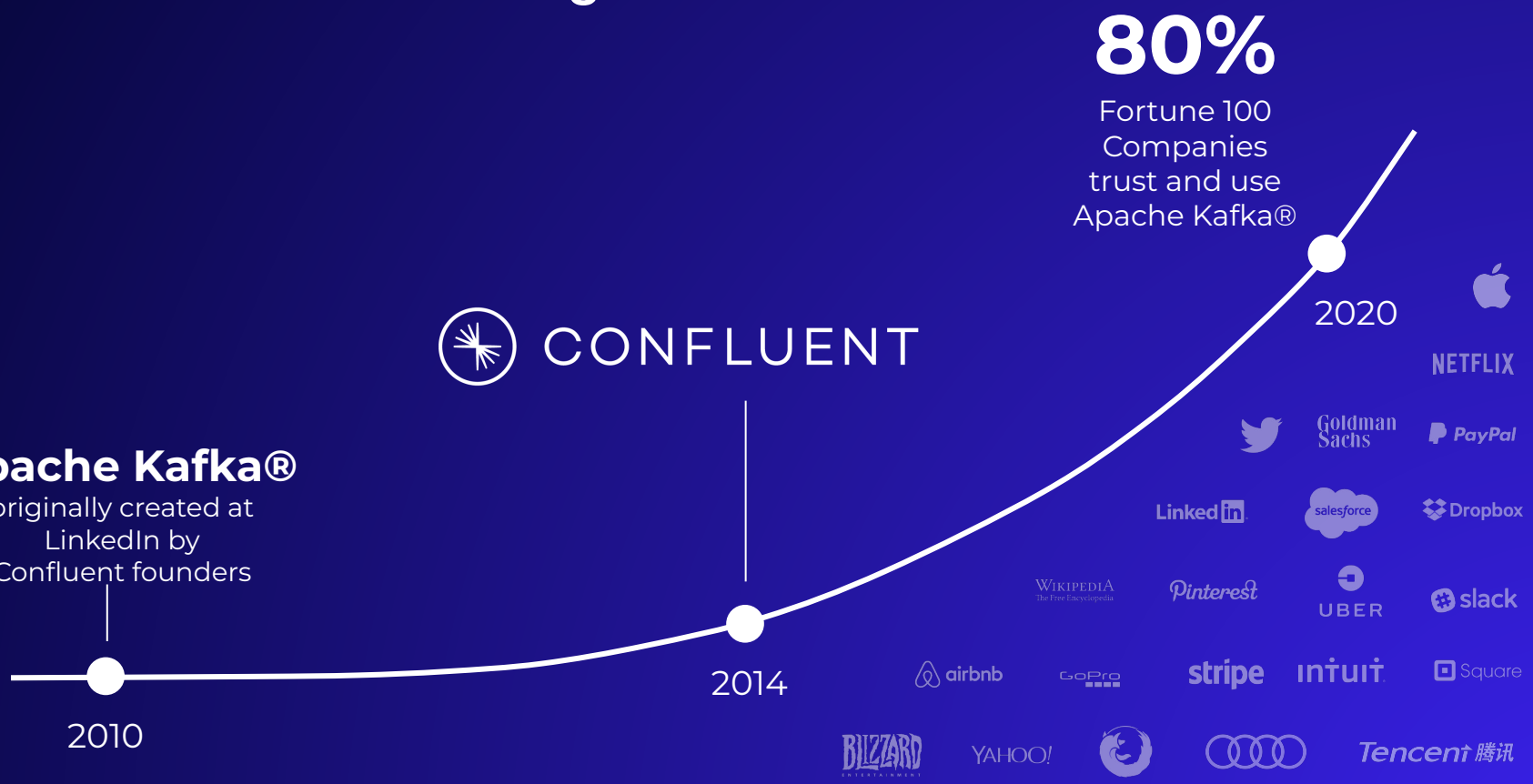
ClickHouse Cloud architecture



Introduction to Confluent

Introducing Apache Kafka®

The Rise of Event Streaming



Confluent Overview

Jay Kreps
CEO

Neha Narkhede

Jun Rao
VP Engineering

Founded by the original creators of **Apache Kafka**

Founded **September 2014**

Technology developed while **at LinkedIn**

BENCHMARK

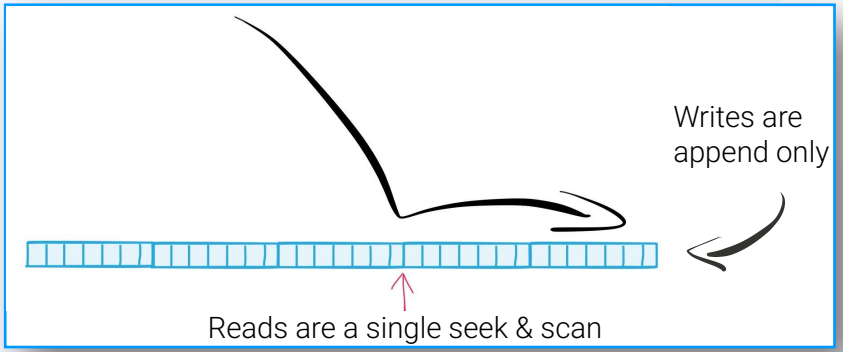
Index Ventures

SEQUOIA

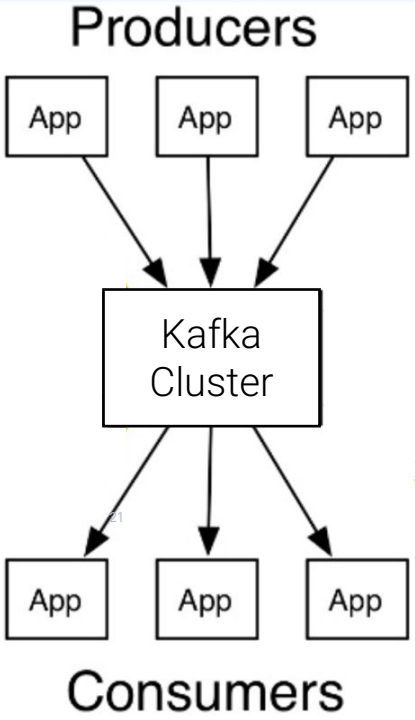
Apache Kafka - Publish & Subscribe Reimagined

Kafka

A **Distributed Commit Log**. Publish and subscribe to streams of records. Highly scalable, high throughput. Supports transactions. Persisted data.



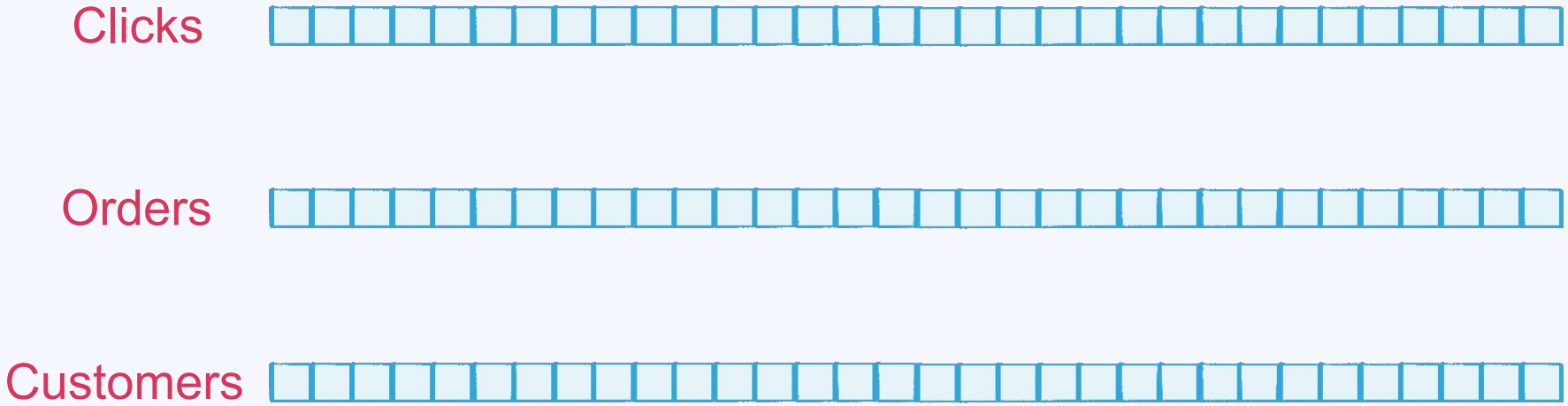
1. Publish Stream Events



2. Store

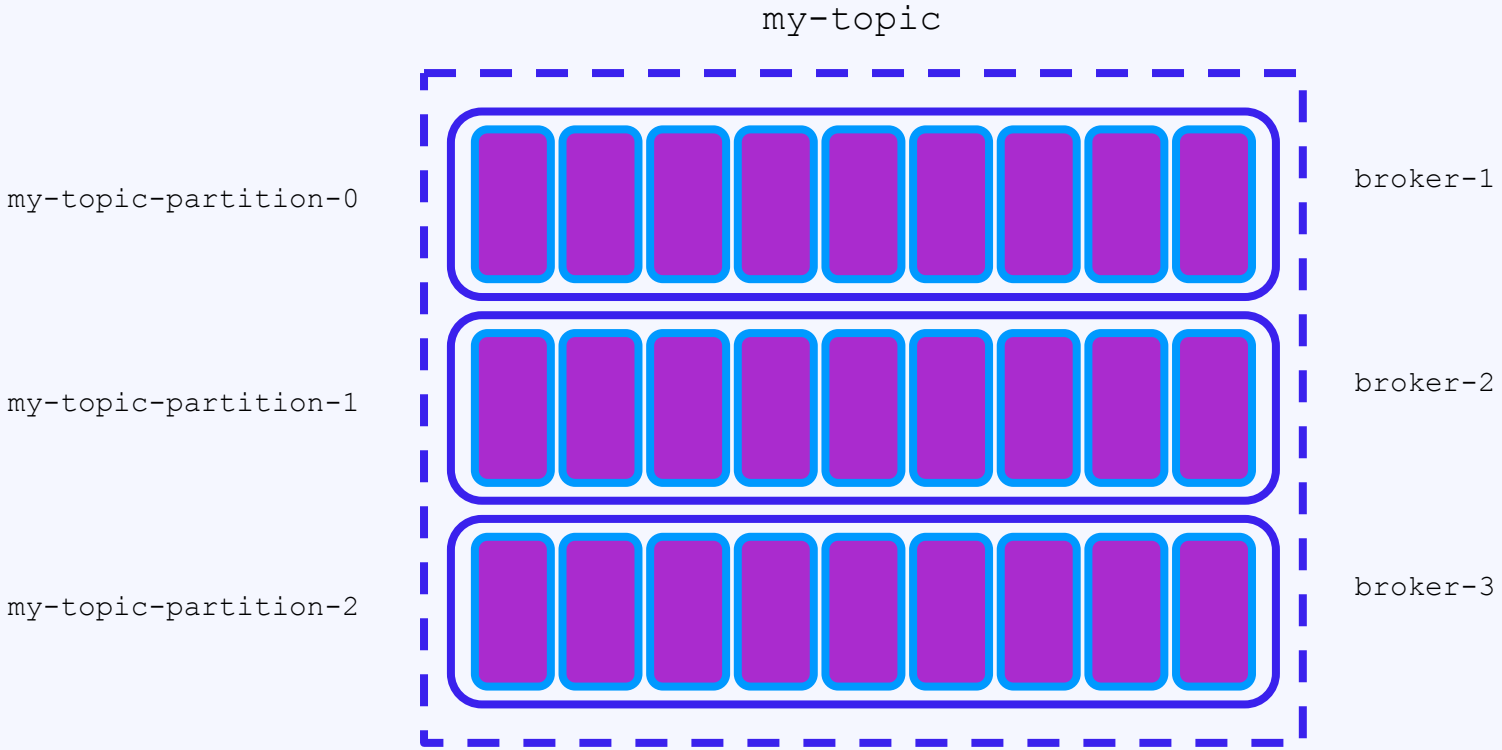
3. Process & Consume

Topics

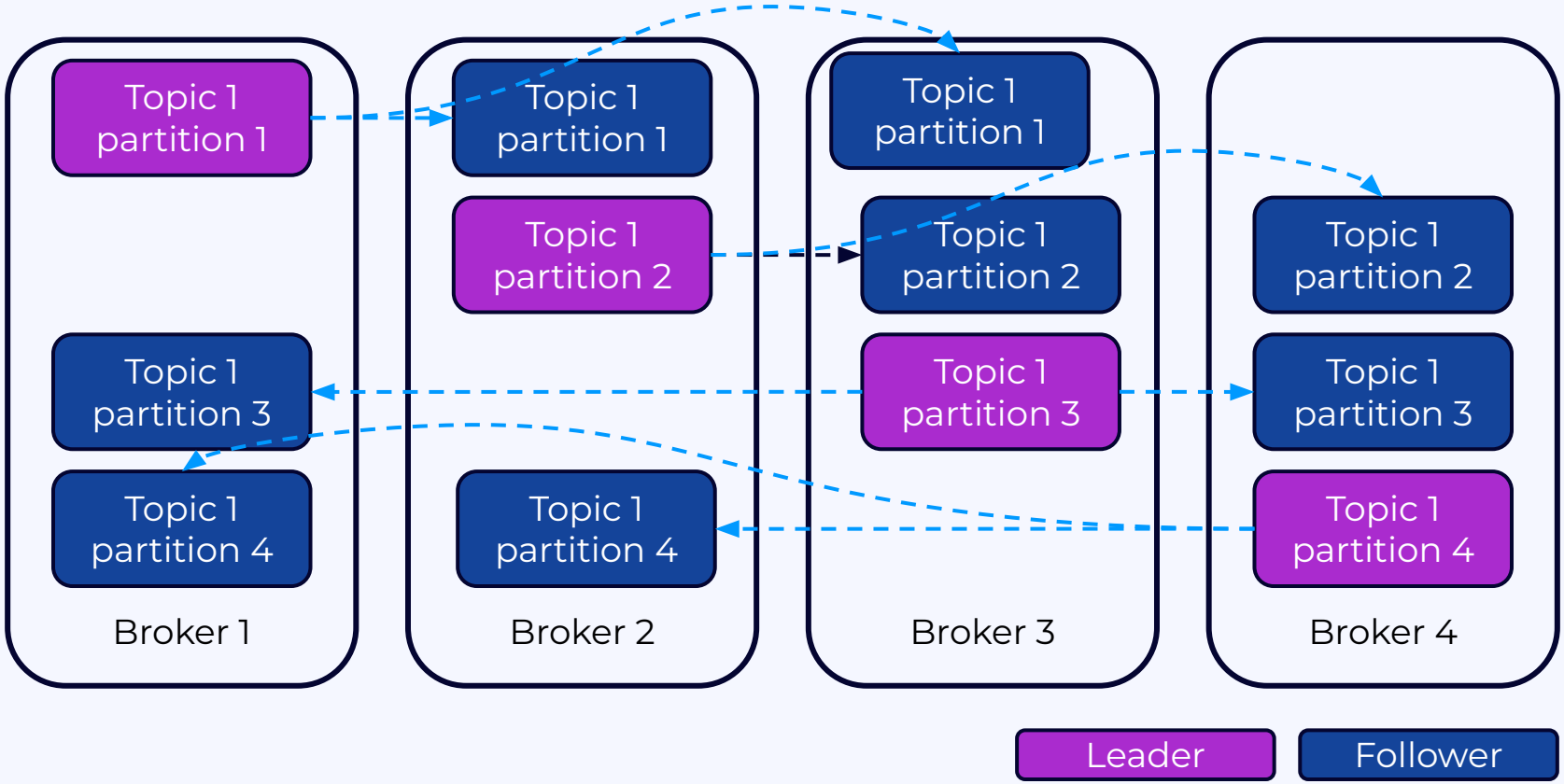


Topics are similar in concept
to tables in a database

Kafka Topics

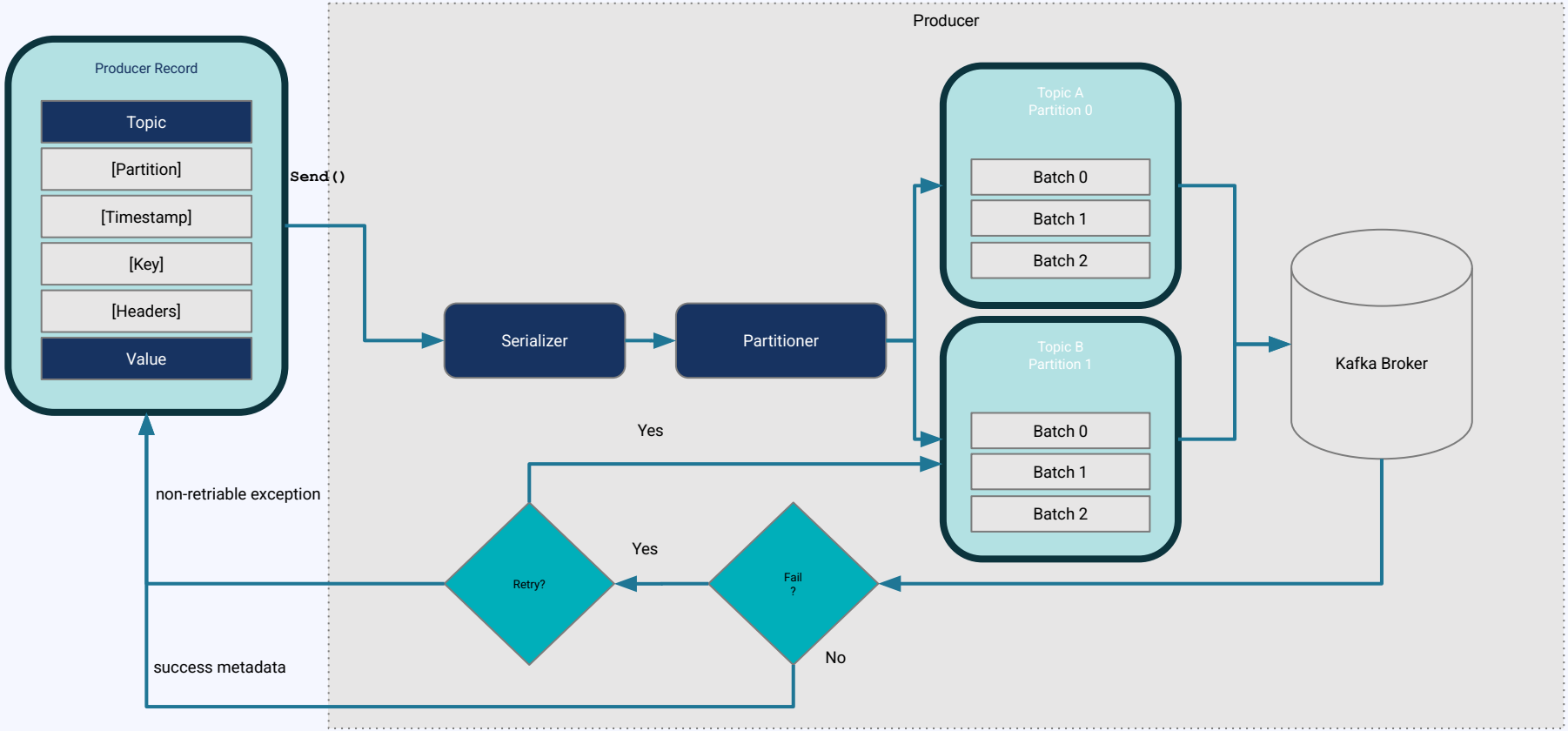


How Does Kafka Provide Resilience?



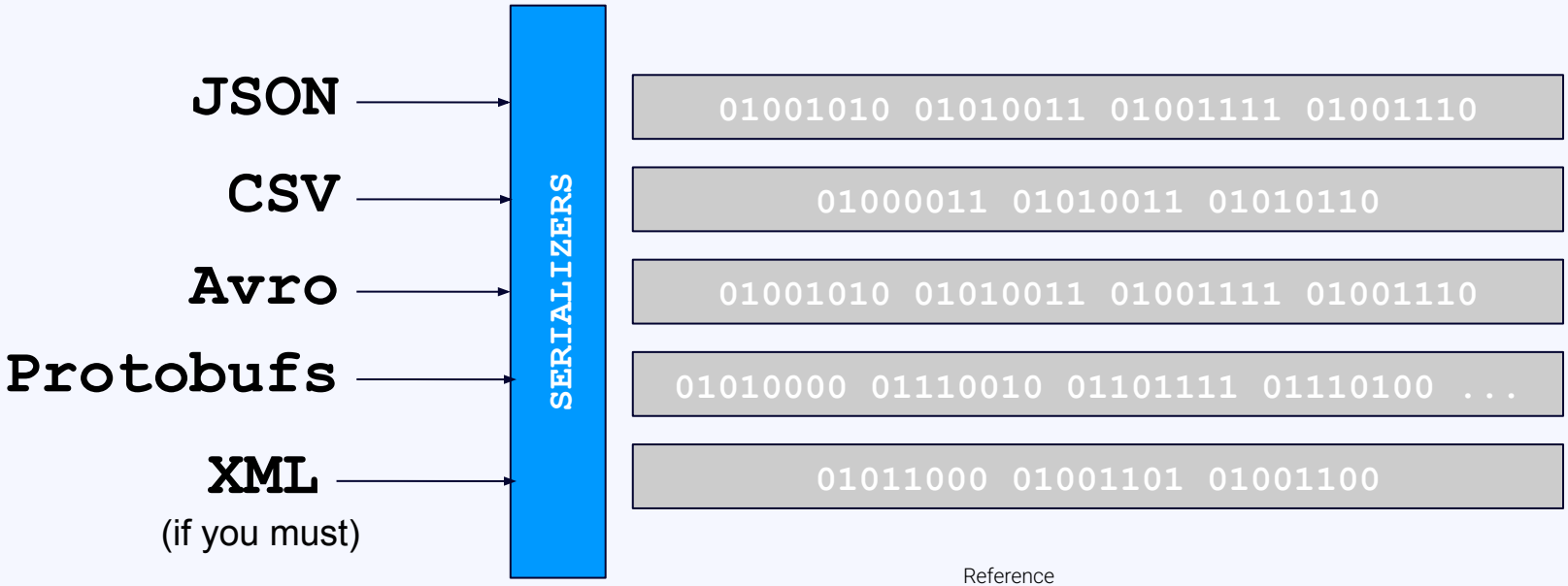
Producing to Kafka

What happens inside a producer?



The Serializer

Kafka doesn't care about what you send to it as long as it's been converted to a byte stream beforehand.



Reference
https://kafka.apache.org/10/documentation/streams/developer-guide/datatypes.html

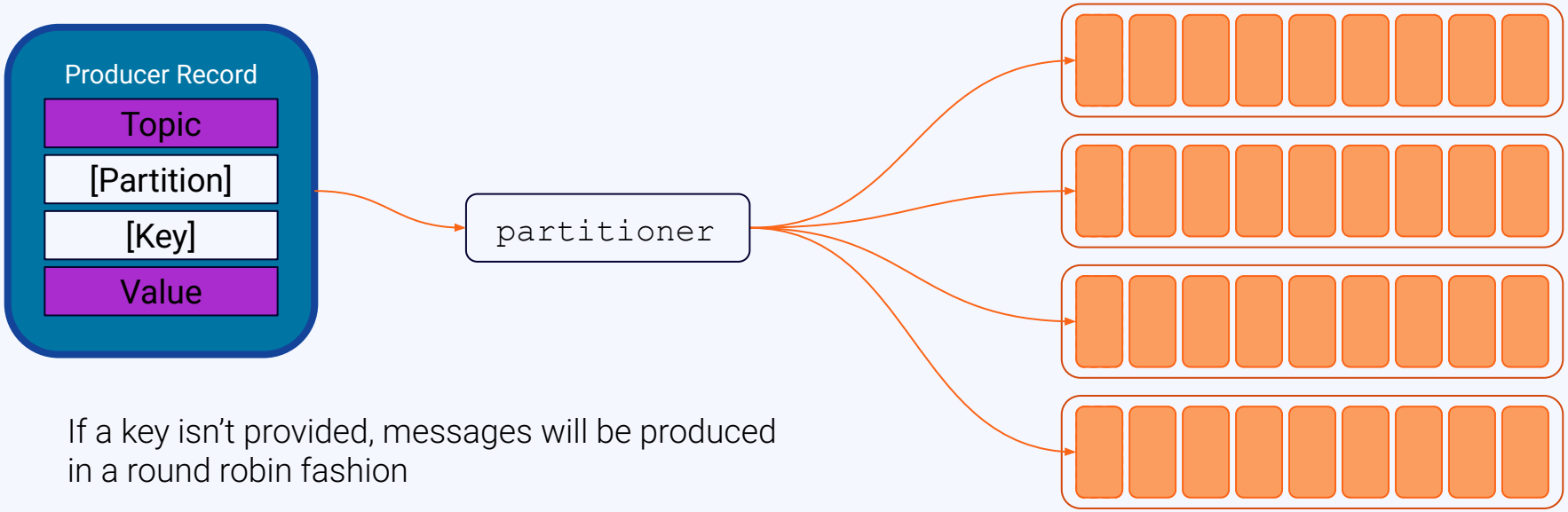
The Serializer

```
private Properties kafkaProps = new Properties();

kafkaProps.put("bootstrap.servers", "broker1:9092,broker2:9092");
kafkaProps.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
kafkaProps.put("value.serializer", "io.confluent.kafka.serializers.KafkaAvroSerializer");
kafkaProps.put("schema.registry.url", "https://schema-registry:8083");

producer = new KafkaProducer<String, SpecificRecord>(kafkaProps);
```

Reference
https://kafka.apache.org/10/documentation/streams/developer-guide/datatypes.html

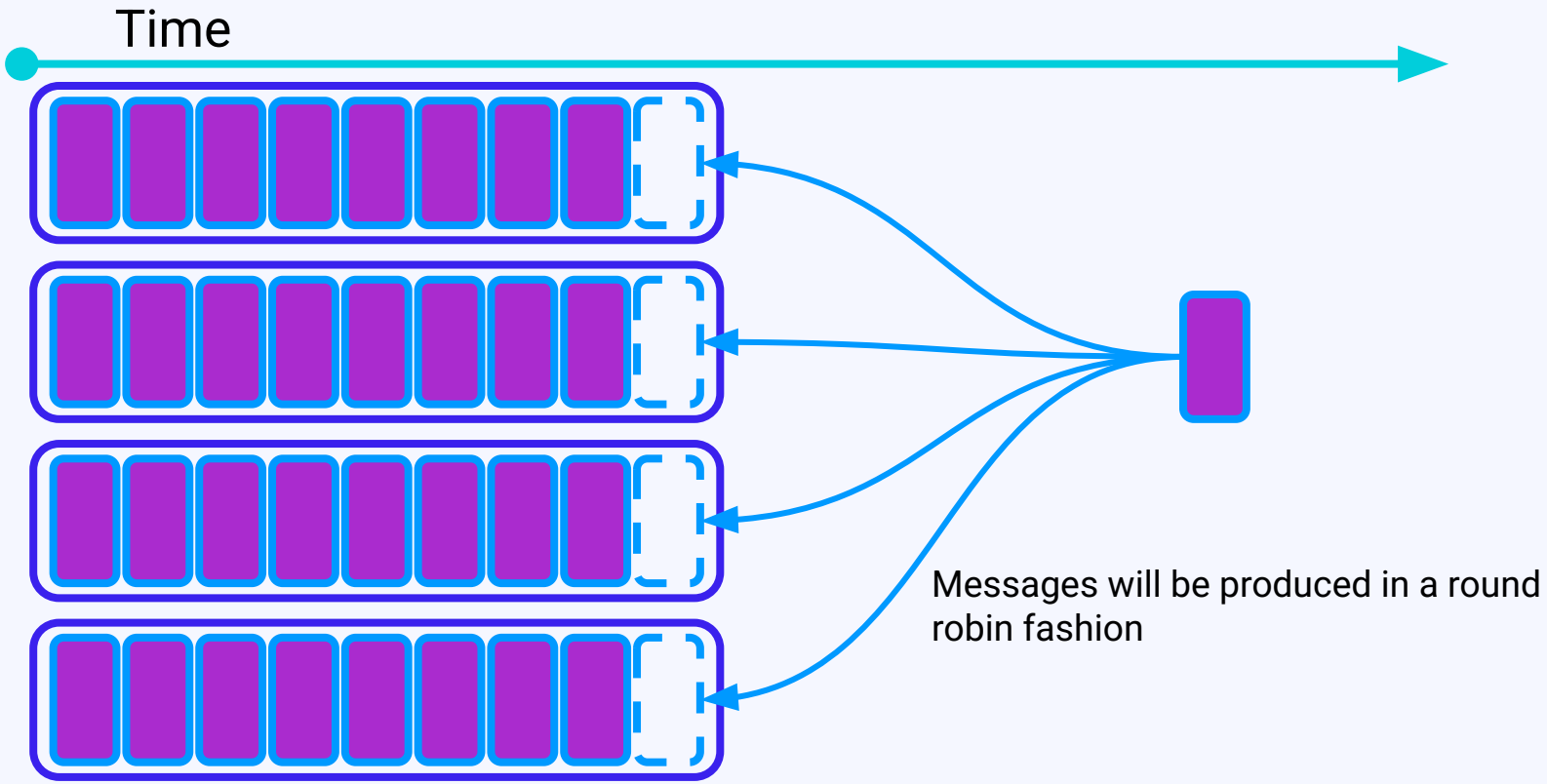
Record Keys and why they're important - Ordering

Record keys determine the partition with the default kafka partitioner

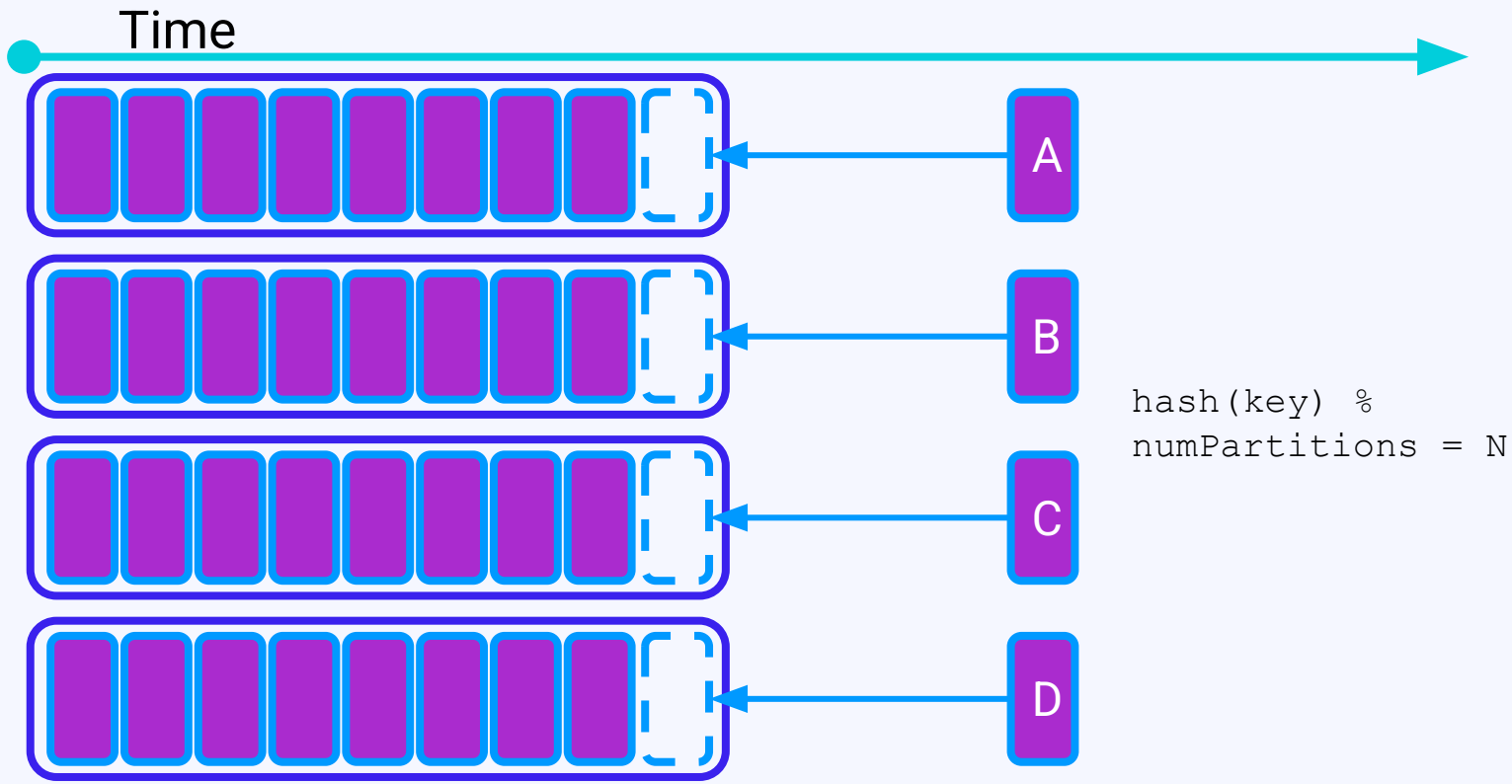


If a key isn't provided, messages will be produced in a round robin fashion

Producing to Kafka - No Key

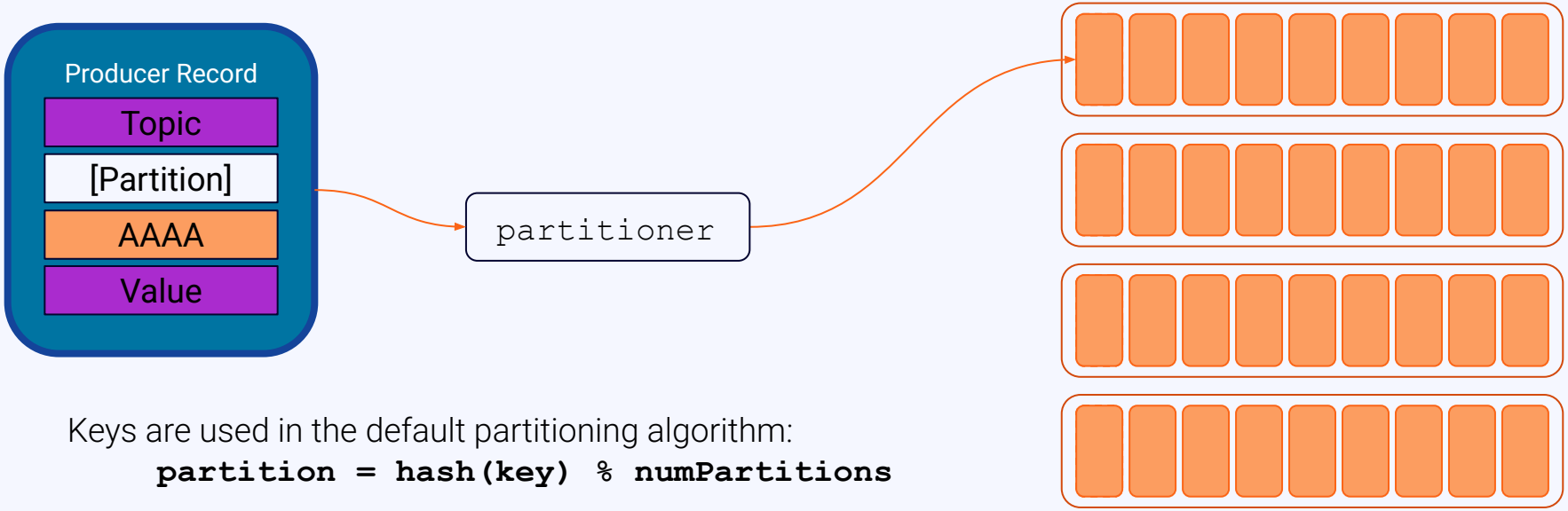


Producing to Kafka - With Key



Record Keys and why they're important - Ordering

Record keys determine the partition with the default kafka partitioner, and therefore guarantee order for a key

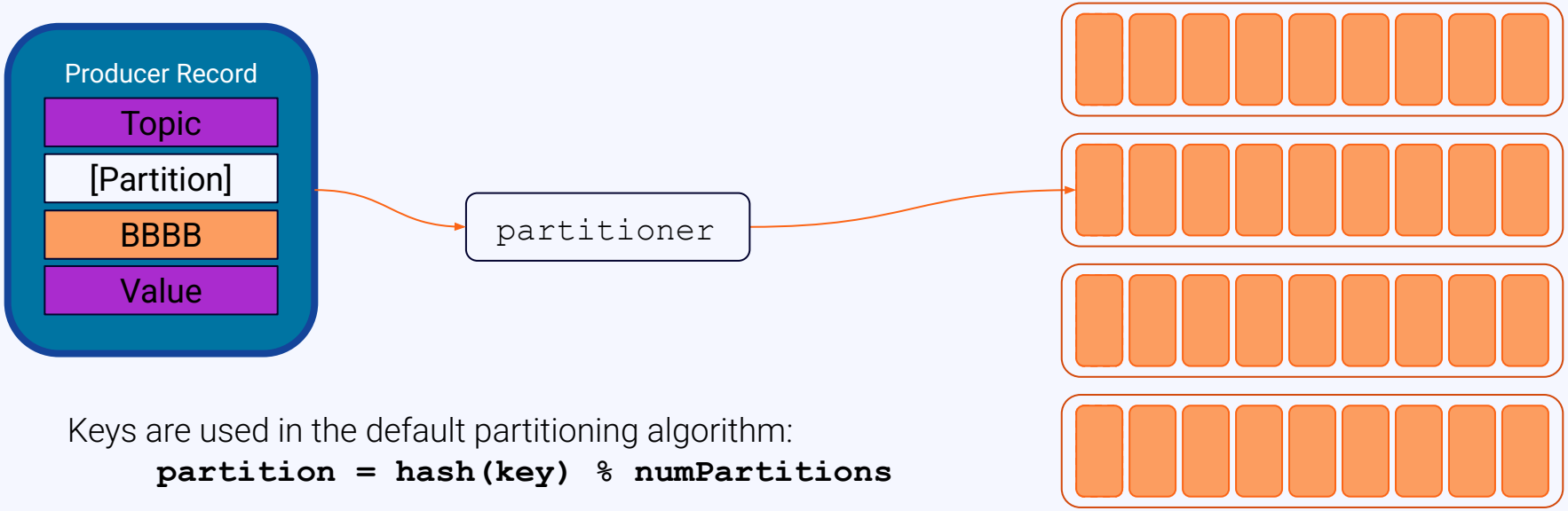


Keys are used in the default partitioning algorithm:

```
partition = hash(key) % numPartitions
```

Record Keys and why they're important - Ordering

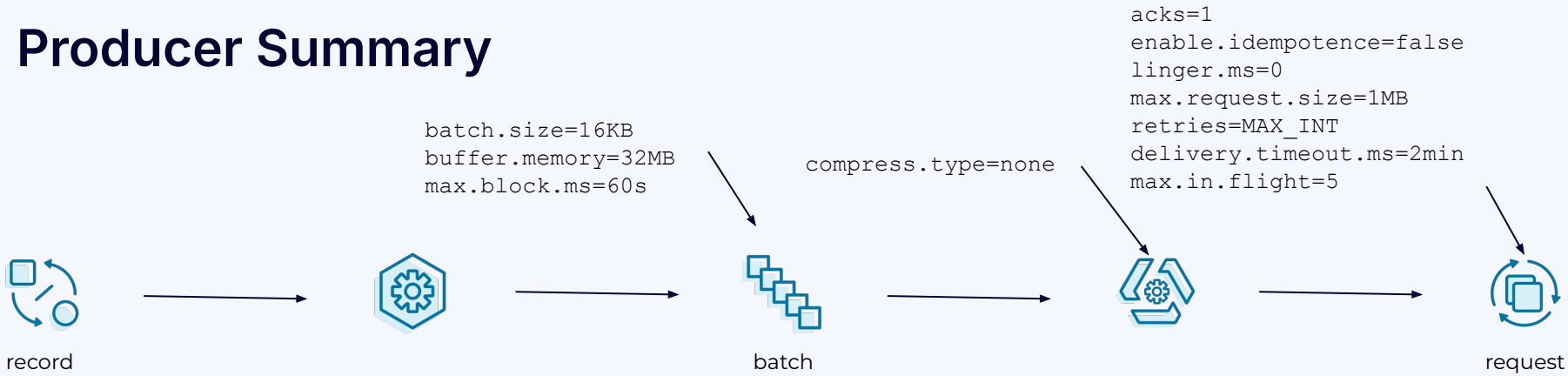
Record keys determine the partition with the default kafka partitioner, and therefore guarantee order for a key



Keys are used in the default partitioning algorithm:

```
partition = hash(key) % numPartitions
```

Producer Summary



Serializer

- Retrieves and caches schemas from Schema Registry

Partitioner

- Java client uses murmur2 for hashing
- If key not provided performs round robin
- If keys unbalanced it will overload one leader

Record accumulator

- Buffer per partition, seldom used partitions may not achieve high batching due to this
- If many producers are in the same JVM, memory and GC could become important
- Sticky partitioner could be used to increase batches in the case of round robin (KIP-408)

Compression

- Allows faster transfer to the producer
- Reduces the inter broker replication load
- Uses less page cache and disk space at broker
- Compress the batch not the record
- Gzip is more CPU intensive, Snappy is lighter, LZ4/ZStd are a good balance*

Sender thread

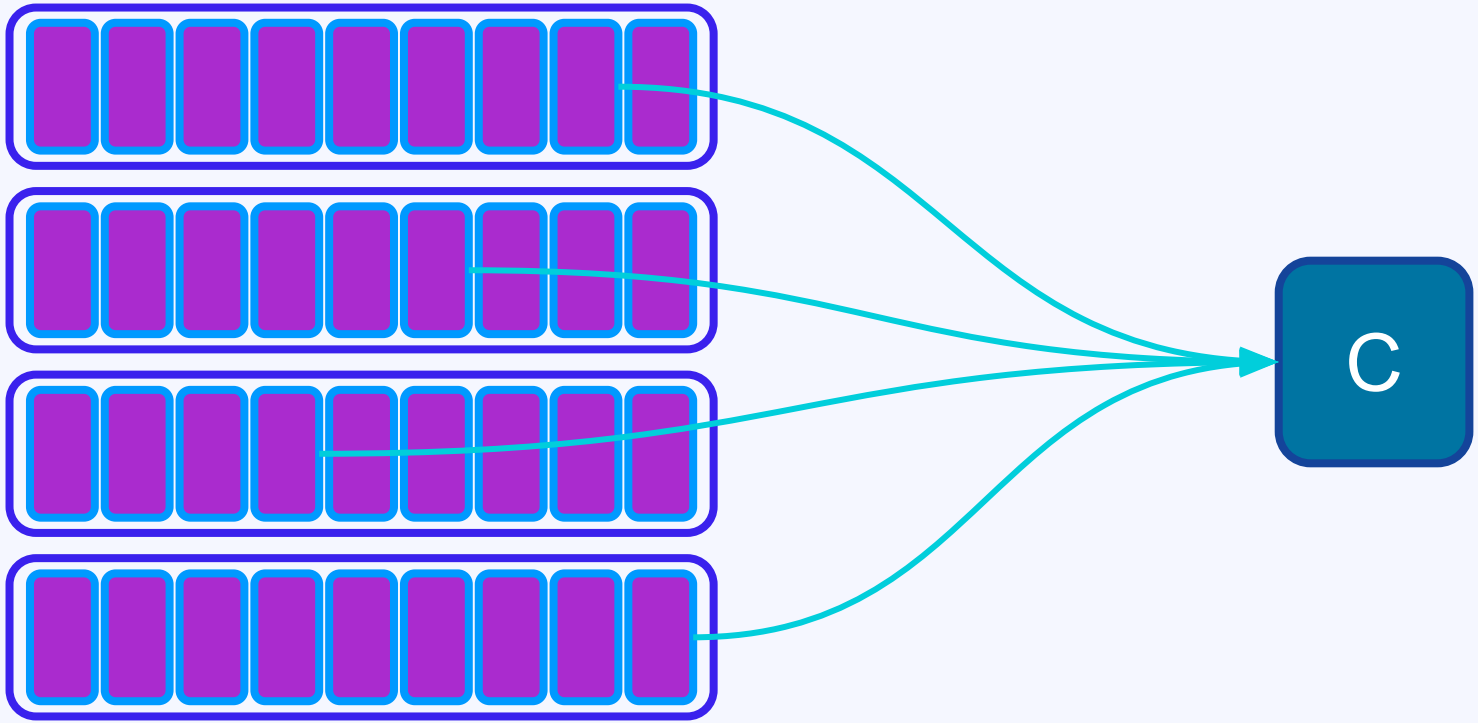
- Batches grouped in a producer request and sent by broker
- Multiple batches to different partitions could be in the same producer request

Consuming from Kafka

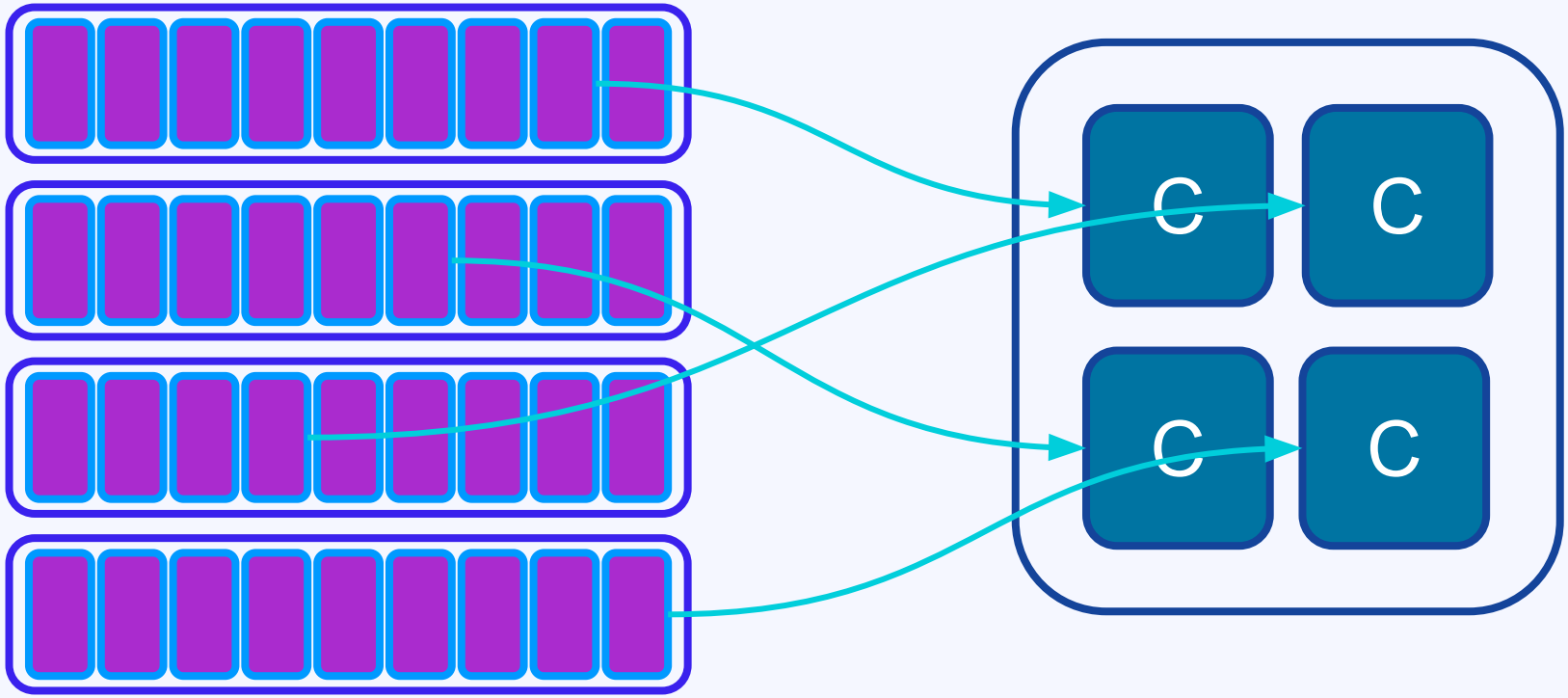
A basic Java Consumer

```
  final Consumer<String, String> consumer = new KafkaConsumer<String,
String>(props);
  consumer.subscribe(Arrays.asList(topic));
  try {
    while (true) {
      ConsumerRecords<String, String> records = consumer.poll(100);
      for (ConsumerRecord<String, String> record : records) {
        -- Do Some Work --
      }
    }
  } finally {
    consumer.close();
  }
}
```

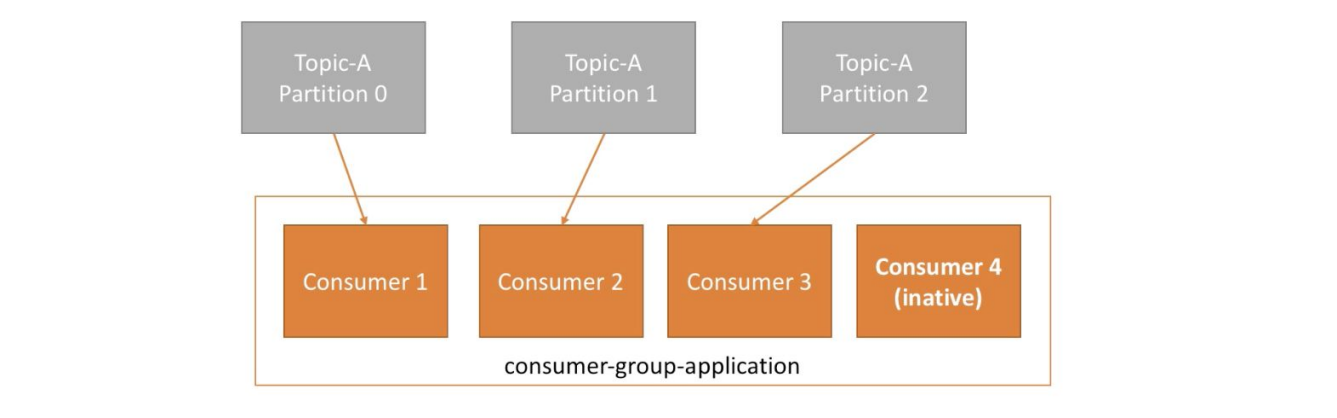
Consuming From Kafka - Single Consumer



Consuming From Kafka - Grouped Consumers

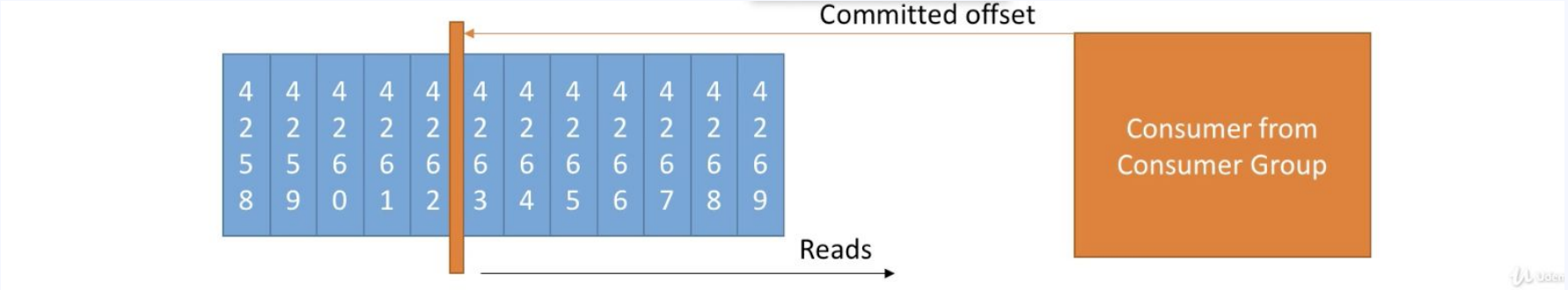


What if you have more Consumers than partitions?



Consumer Offsets

- Kafka stores the offsets at which a consumer group has been reading
- Offsets are committed in an internal **topic** called **_consumer_offsets**
- When a consumer in a group has processed data received from Kafka it should be committing the offsets
- If a consumer dies Kafka will be able to tell it where it was so it can start reading again



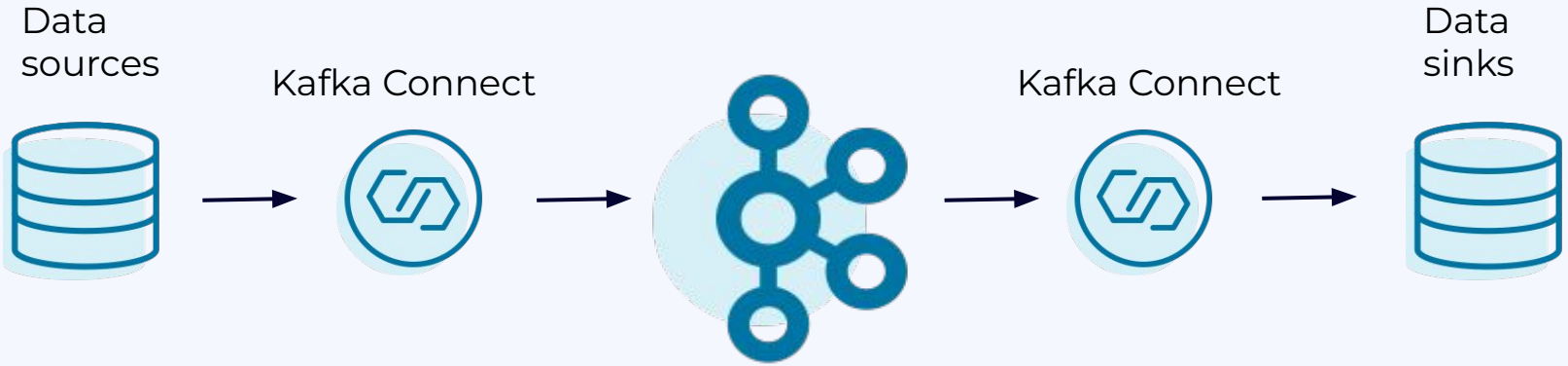
Kafka Connect

No/Low Code connectivity to many systems

Kafka Connect

No-Code way of connecting known systems (databases, object storage, queues, etc) to Apache Kafka

Some code can be written to do custom transforms and data conversions though maybe out of the box Single Message Transforms and Converters exist

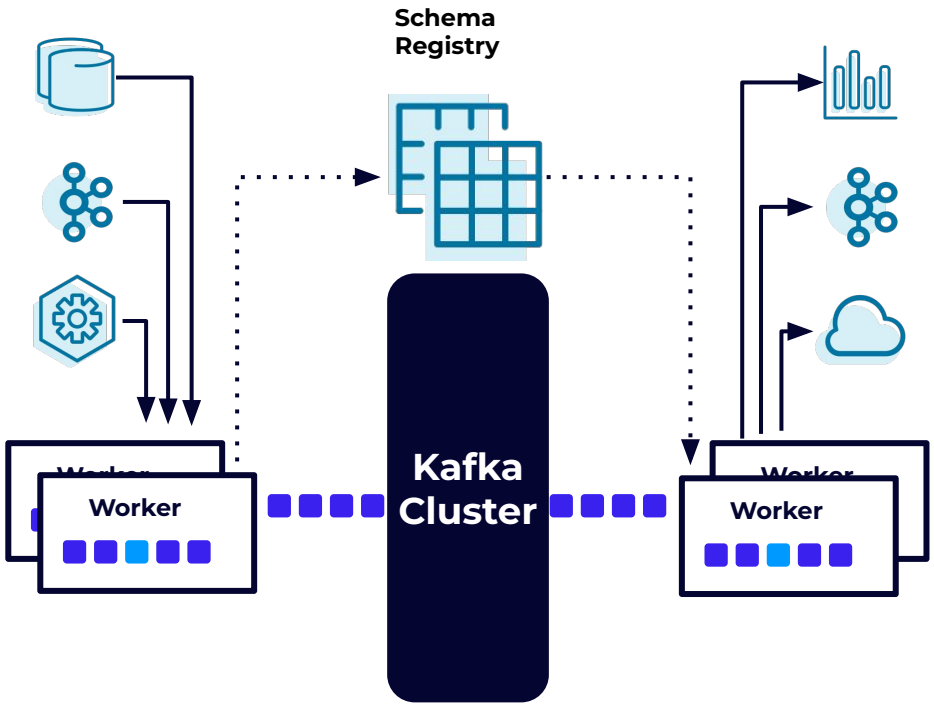


Kafka Connect Durable Data Pipelines

Integrate upstream and downstream systems with Apache Kafka®

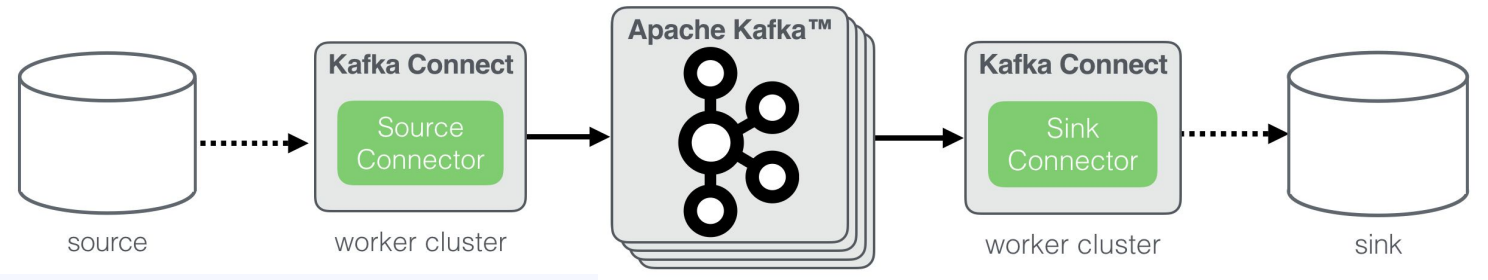
- **Capture** schema from sources, use schema to inform data sinks
- **Highly Available** workers ensure data pipelines aren't interrupted
- **Extensible** framework API for building custom connectors

Kafka Connect



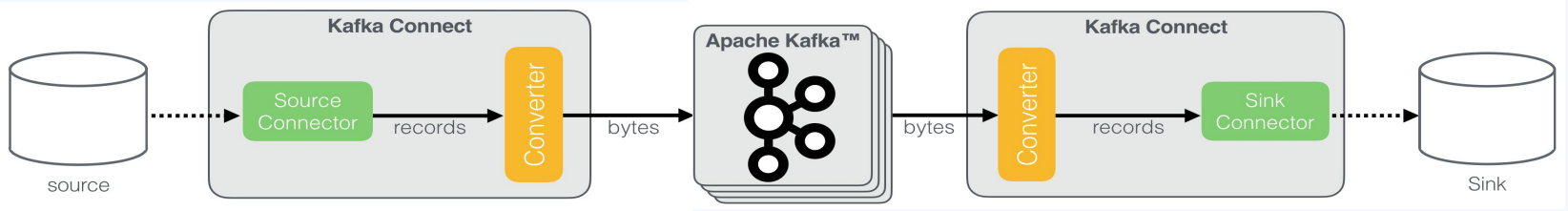
Kafka Connect

Connectors are reusable components that know how to talk to specific **sources and sinks**.



Kafka Connect

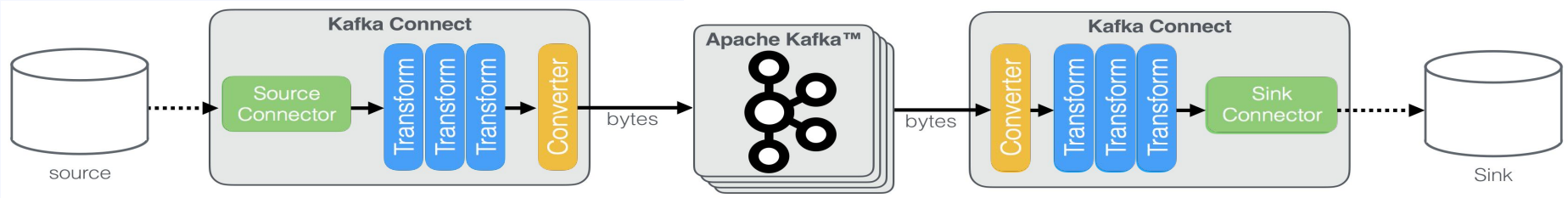
Convert between the **source and sink record** objects and the **binary format** used to persist them in Kafka.



JSON, Avro, and others

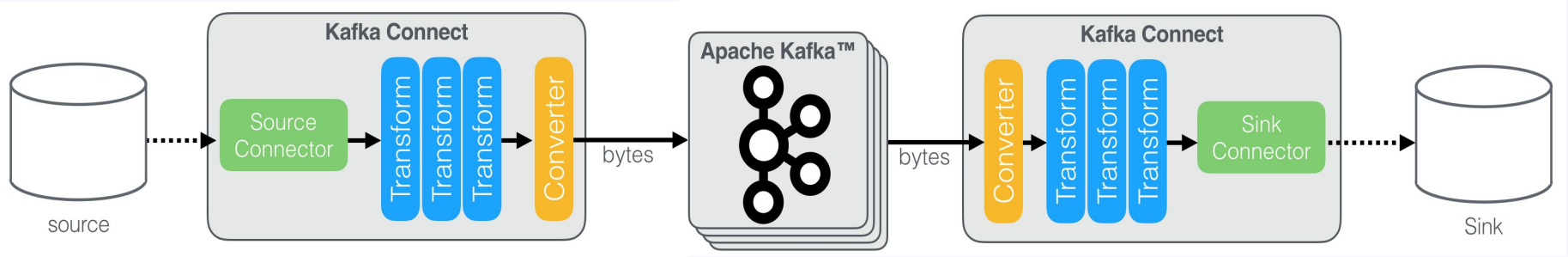
Kafka Connect

Connectors, transforms, and converters
are **pluggable components**.



Kafka Connect

Modify the structure of keys and values, topics, and partition of **source and sink record objects**.



CONFLUENT
Developer

Your Apache Kafka journey begins here

developer.confluent.io

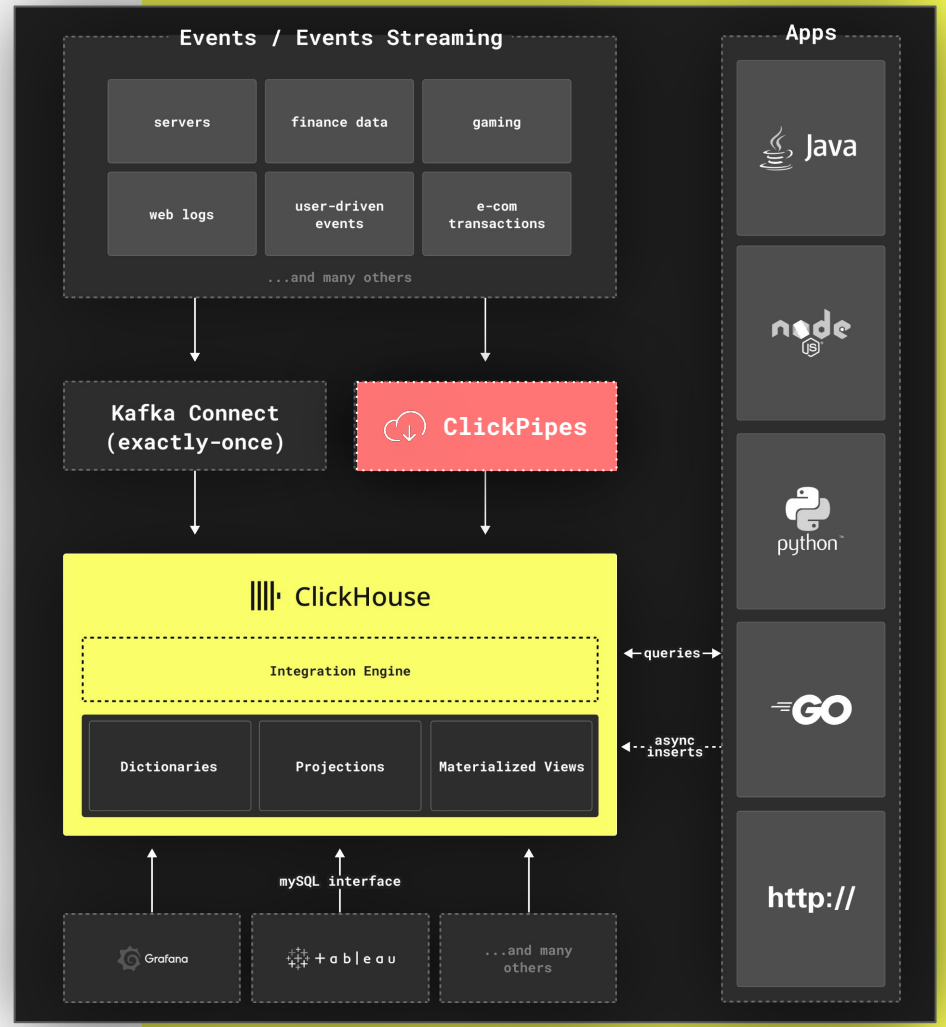
ClickHouse

Integrating Kafka with ClickHouse

$(\text{real-time})^2$

ClickPipes *noun*
/klɪk paɪps/

Cloud-Native experience to ingest data from remote data sources to ClickHouse Cloud

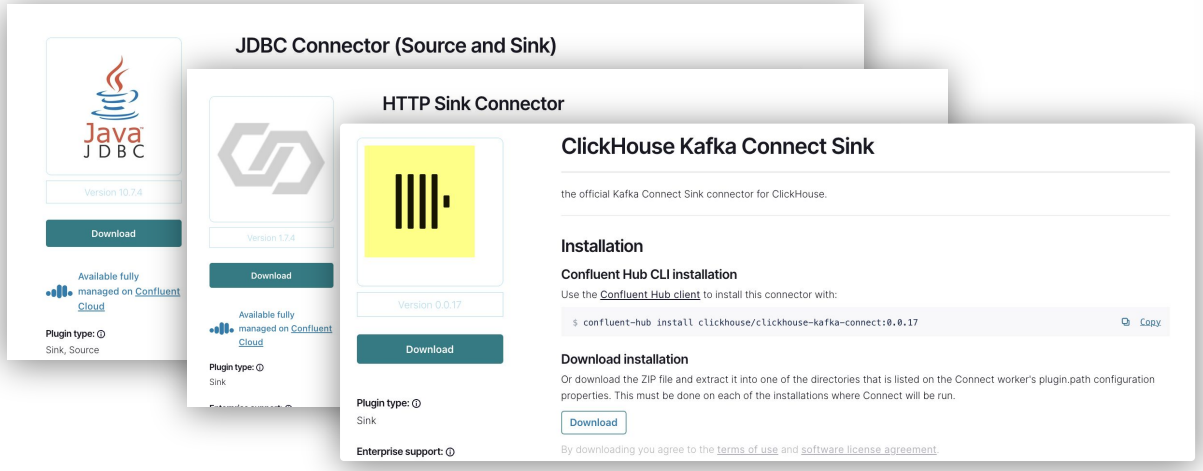


Existing Kafka Integration Options

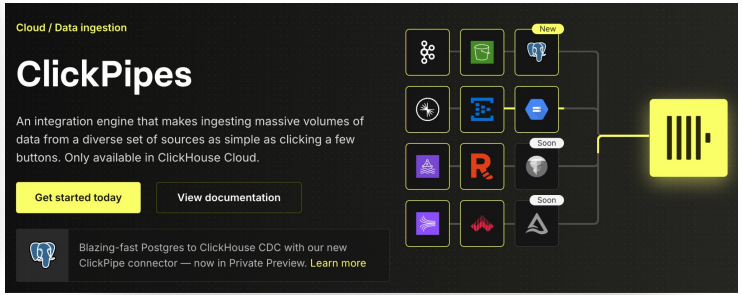
01 Kafka Table Engine

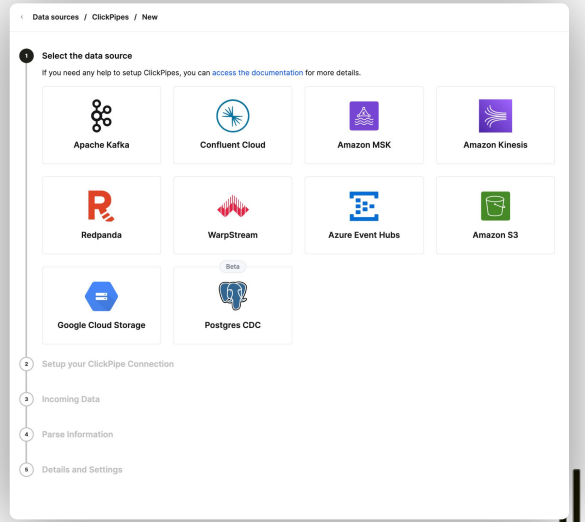
```
CREATE TABLE queue (
        timestamp UInt64,
        level String,
        message String
    )
ENGINE = Kafka('localhost:9092', 'topic', 'group1', 'JSONEachRow');
```

02 ClickHouse Kafka Connect Sink



03 ClickPipes

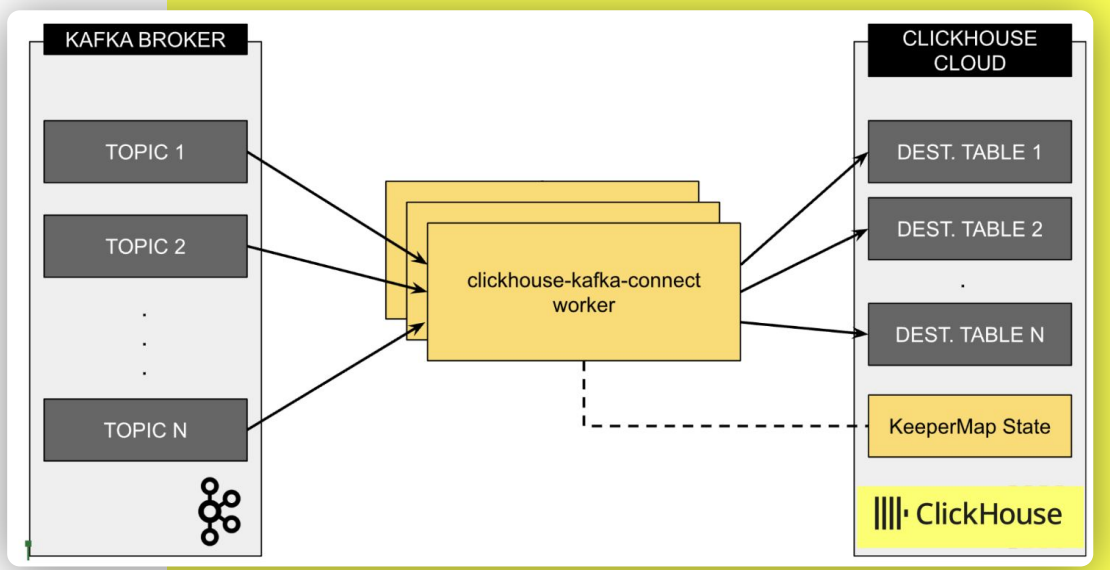




ClickHouse

ClickHouse Kafka Connect Sink

- Shipped with out-of-the-box **exactly-once semantics** powered by ClickHouse core feature named **KeeperMap** (table engine used as a state store by the connector) and allows for minimalistic architecture
- **Core integration**: Built, maintained, and **supported by ClickHouse**
- Tested **continuously** against ClickHouse Cloud.
- Data inserts with a **declared schema** (schema-based data, e.g. Avro, Protobuf, etc.) and **schemaless** (e.g. JSON)
- Support for all data types of ClickHouse



ClickHouse

Demo

KEEP IN TOUCH!

ClickHouse Sydney Meetup Group

clickhouse.com/slack

@clickhousedb

@clickhouseinc #clickhouseDB

clickhouse

ClickHouse

MIGRATION FROM BIGQUERY TO CLICKHOUSE VIRTUAL WORKSHOP

July 9, 2025 - 11:30 AM SGT

ClickHouse + CONFLUENT

Questions