Data-Intensive Distributed Computing
CS 451/651 (Fall 2025)

UNIVERSITY OF WATERLOO

Stream Processing
(v1.00)

Week 13: November 27, 2025

Jimmy Lin
David R. Cheriton School of Computer Science
University of Waterloo

These slides are available at http://lintool.github.io/bigdata-2025f/

Key Questions

What are the challenges associated with stream processing?

What are the common patterns in connecting
data producers and data consumers?

What are some common data structures and algorithms for
processing unbounded streams?

What's the distinction between event time and processing time?

How do stream processing platforms fit into the lakehouse?

What is a data stream?

Sequence of items:

Structured (e.g., tuples)

Ordered (implicitly or timestamped)

Arriving continuously at (possibly) high volumes

Potentially unbounded (i.e., never ending)

Sometimes not possible to store entirely

Sometimes not possible to even examine all items

Applications

Network traffic monitoring
Datacenter telemetry monitoring
Sensor networks monitoring
Credit card fraud detection
Stock market analysis
Online mining of click streams
Monitoring social media streams

Key: real-time insights, analytics, decision making, etc.

What exactly do you want to do?

```
word_counts = (
    text_file.flatMap(lambda line: line.lower().split(" "))
    .filter(lambda word: word.strip() != "")
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a + b)
)
```

Whatever you were doing before, except on streams!

What exactly do you want to do?

"Standard" relational operations:

Select

Project

Transform (i.e., apply custom UDF)

Group by

Join

Aggregations

What else do you need to make this "work"?

Issues of Semantics

Group by… aggregate

When do you stop grouping and start aggregating?

Joining a stream and a static source

Simple lookup

Joining two streams

How long do you wait for the join key in the other stream?

Joining two streams, group by and aggregation

When do you stop joining?

You need windows!

The Fundamental Challenge

How do you consume an unbounded stream with finite resources?

Bound space consumption

Introduce windows

Use clever data structures and algorithms

Spark Streaming

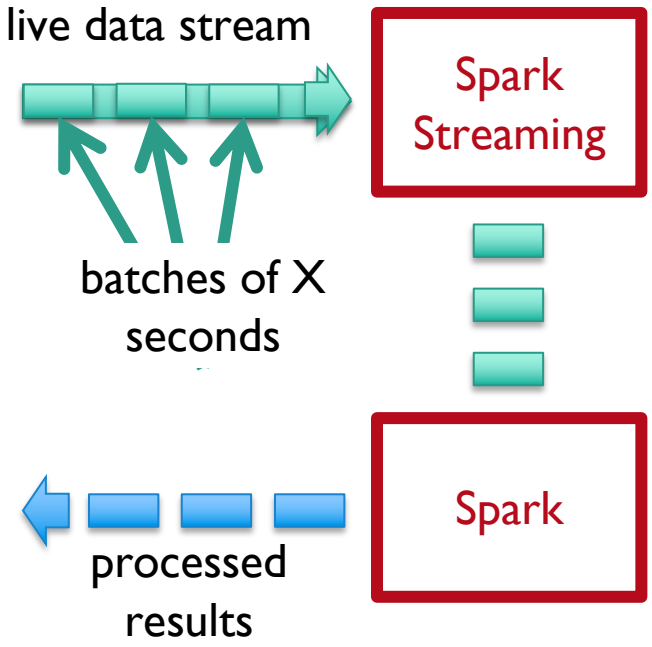
Spark Streaming: Discretized Streams

Run a streaming computation as a series of very small, deterministic batch jobs

Chop up the stream into batches of *X* seconds

Process as RDDs!

Return results in batches



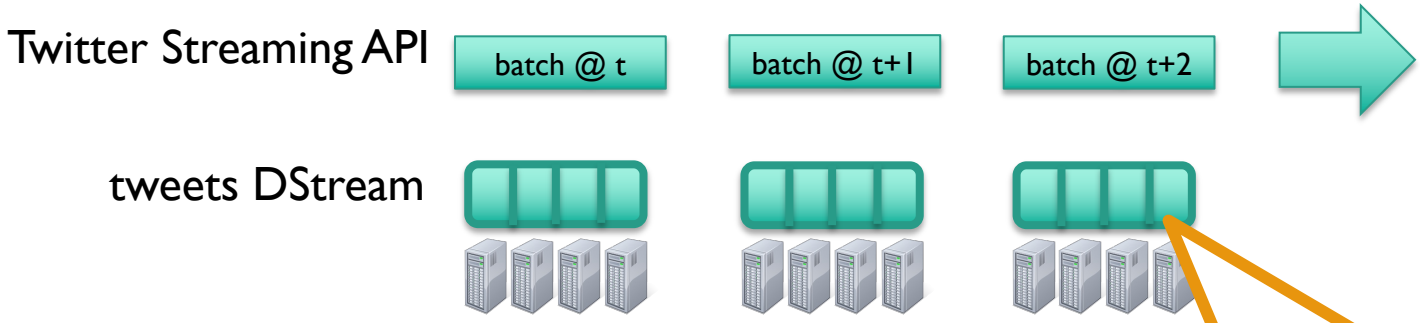
Typical batch window ~1s

Source: All following Spark Streaming slides by Tathagata Das

Example: Get hashtags from Twitter

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
```

DStream: a sequence of RDD representing a stream of data



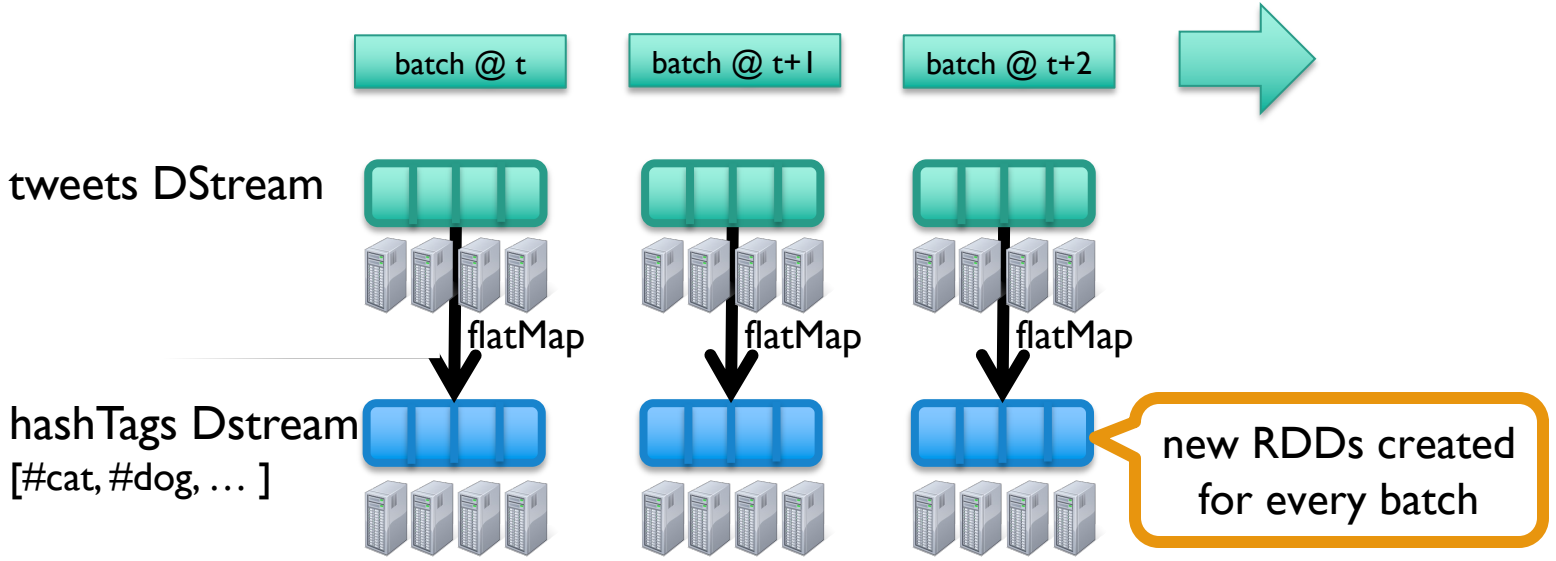
stored in memory as an RDD
(immutable, distributed)

Example: Get hashtags from Twitter

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
```

new DStream

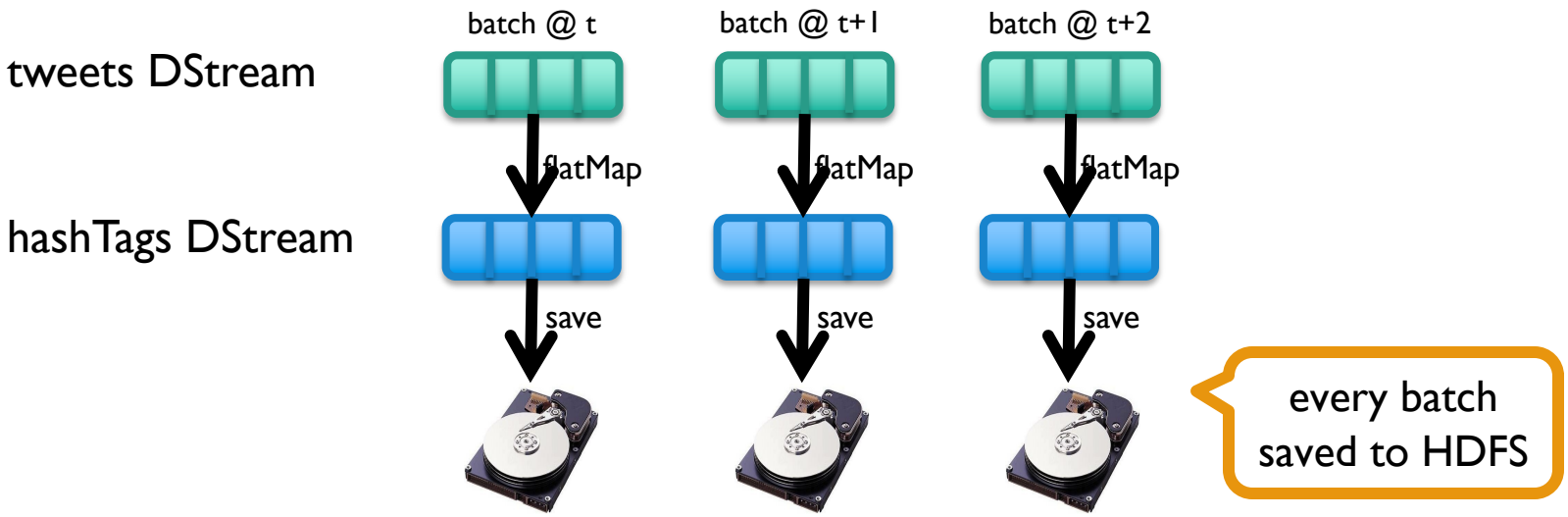
transformation: modify data in one Dstream to create another DStream



Example: Get hashtags from Twitter

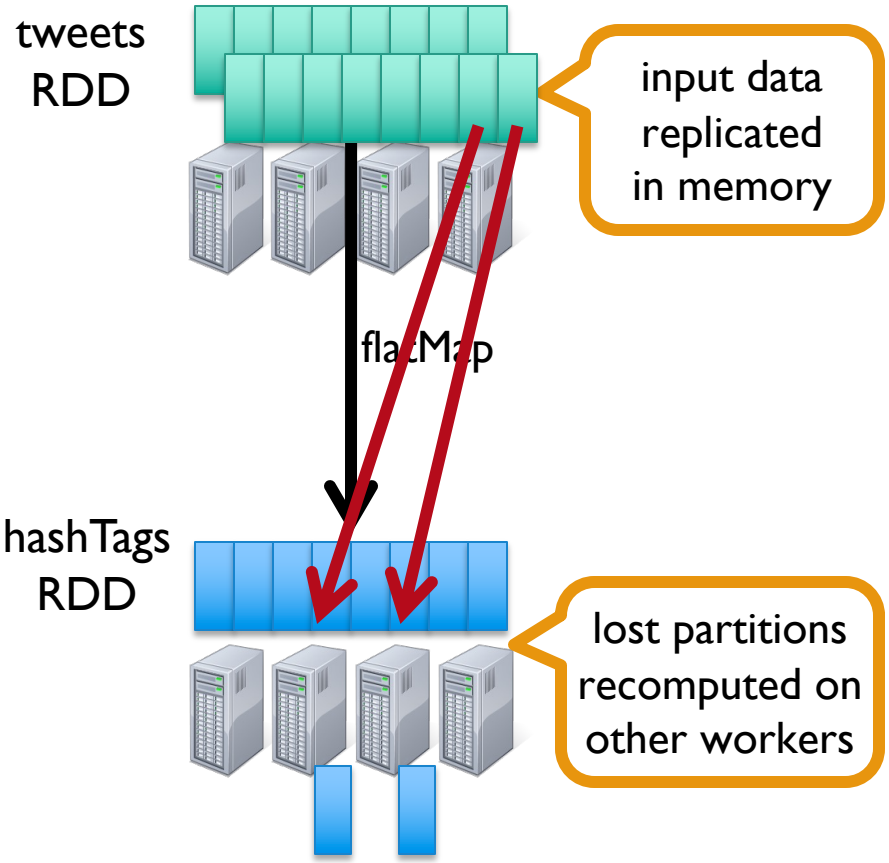
```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
```

output operation: to push data to external storage



Fault Tolerance

Bottom line: they're just RDDs!



Key Concepts

DStream – sequence of RDDs representing a stream of data

Twitter, HDFS, Kafka, ZeroMQ, TCP sockets, …

Transformations – modify data from on DStream to another

Standard RDD operations – map, countByValue, reduce, join, …

Stateful operations – window, countByValueAndWindow, …

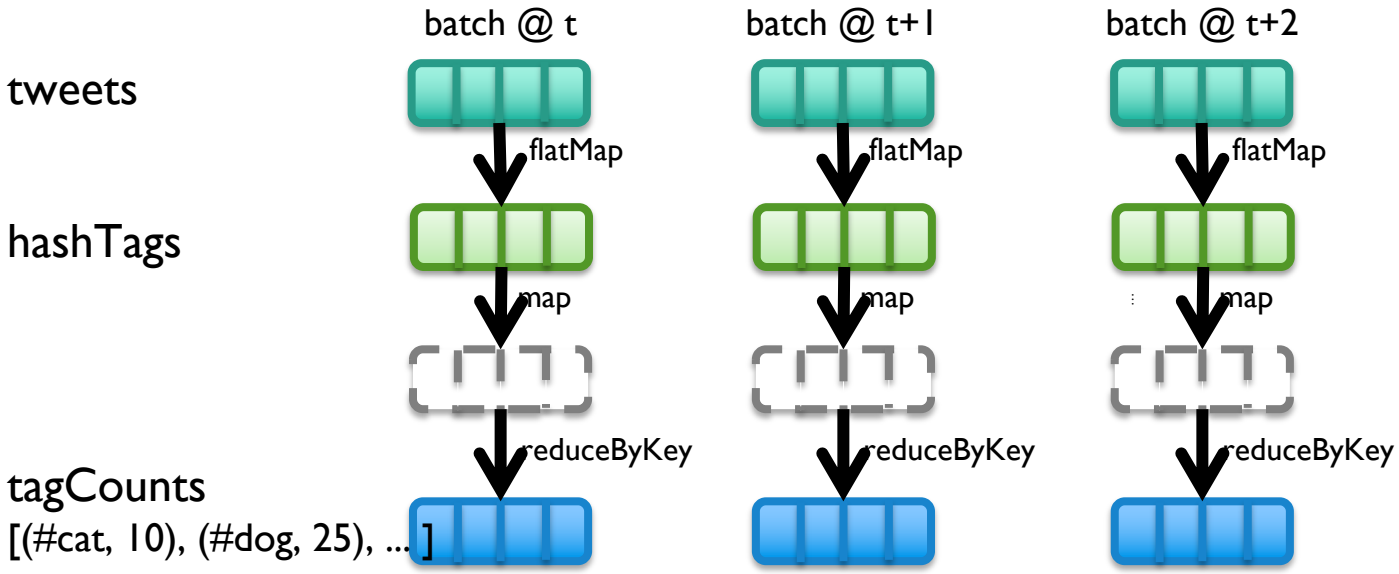
Output Operations – send data to external entity

saveAsHadoopFiles – saves to HDFS

foreach – do anything with each batch of results

Example: Count the hashtags

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
val tagCounts = hashTags.countByValue()
```

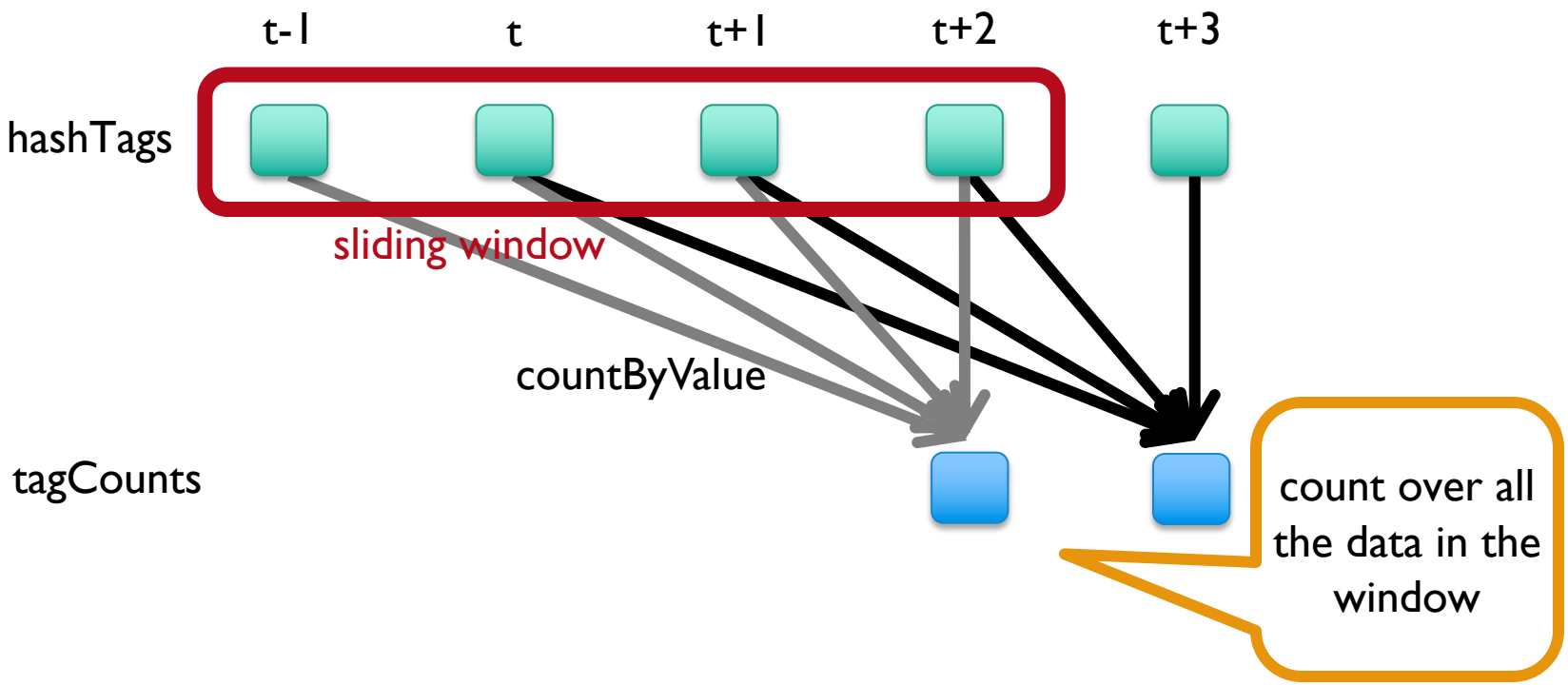


Example: Count the hashtags over last 10 mins

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
val tagCounts = hashTags.window(Minutes(10), Seconds(1)).countByValue()
```

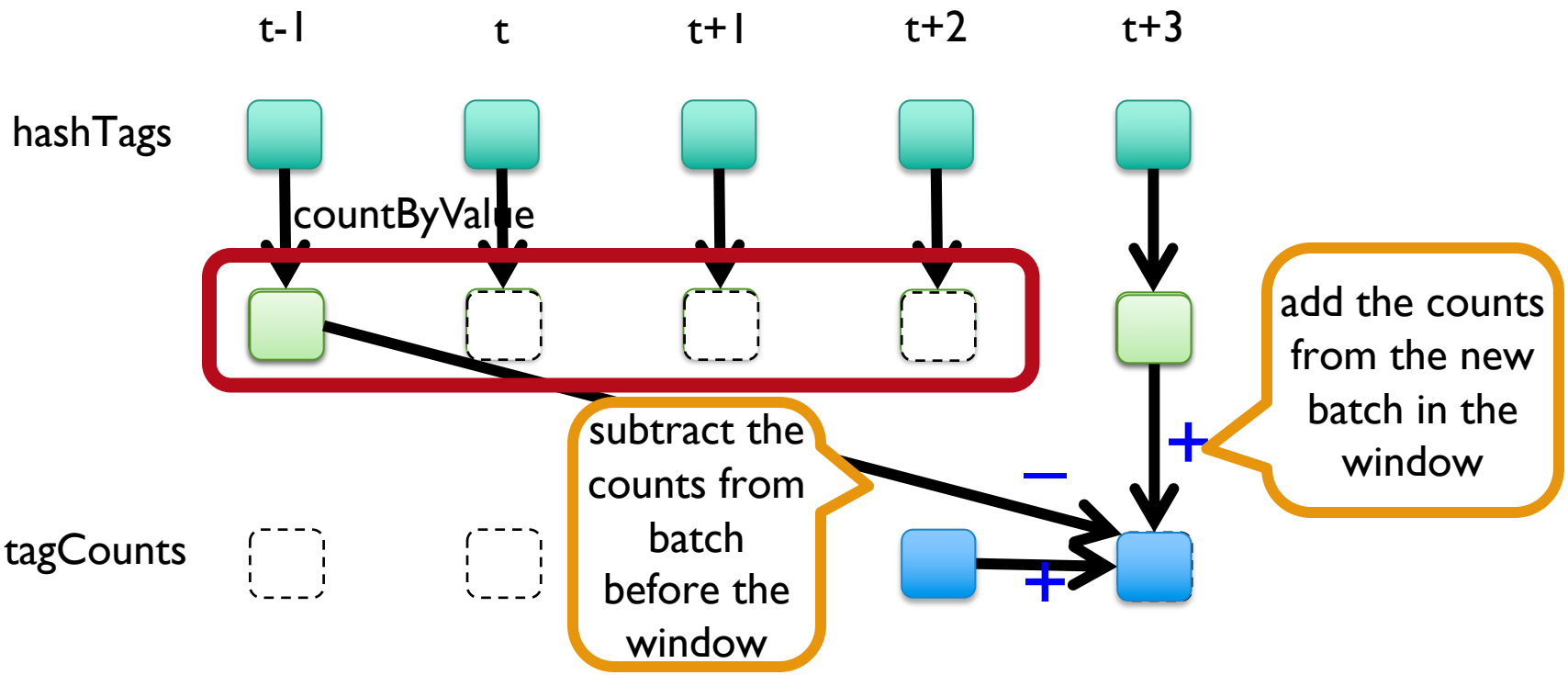
Example: Count the hashtags over last 10 mins

```
val tagCounts = hashTags.window(Minutes(10), Seconds(1)).countByValue()
```



Smart window-based countByValue

```
val tagCounts = hashtags.countByValueAndWindow(Minutes(10), Seconds(1))
```



Smart window-based reduce

Incremental counting generalizes to many reduce operations

Need a function to "inverse reduce" ("subtract" for counting)

```
val tagCounts = hashtags
    .countByValueAndWindow(Minutes(10), Seconds(1))
```

```
val tagCounts = hashtags
    .reduceByKeyAndWindow(_ + _, _ - _, Minutes(10), Seconds(1))
```

Demo

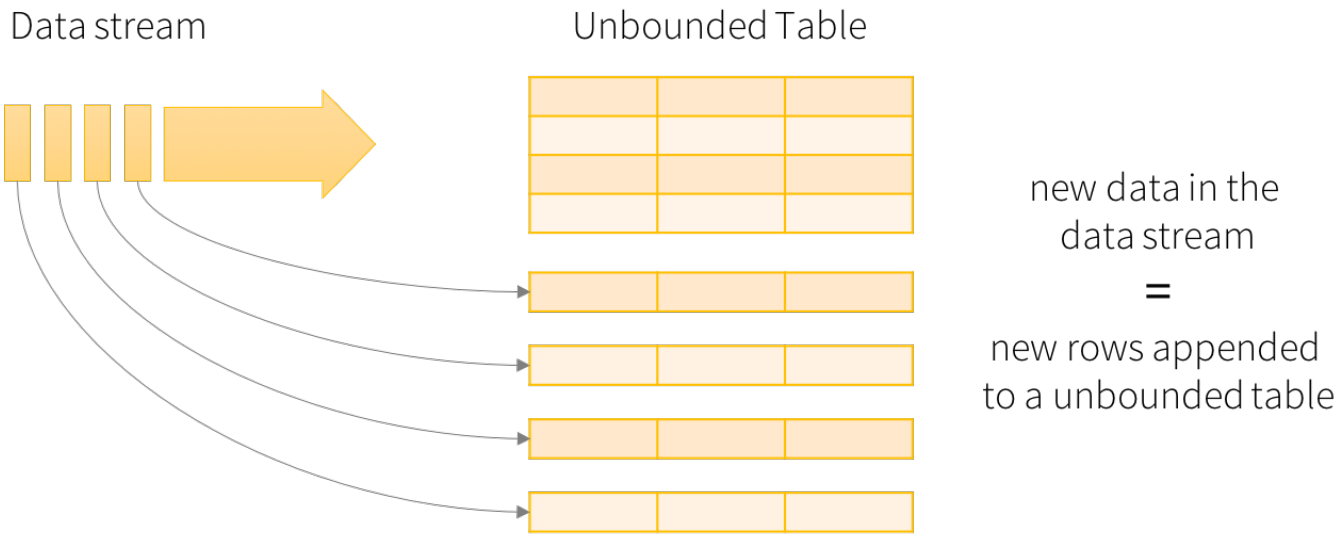
```
# Create a local StreamingContext with a batch interval of 5 second
ssc = StreamingContext(sc, 5)

# Create a DStream that will connect to hostname:port
lines = ssc.socketTextStream("localhost", 9999)

# Split each line into words
words = lines.flatMap(lambda line: line.split(" "))

# Count each word in each batch
pairs = words.map(lambda word: (word, 1))
wordCounts = pairs.reduceByKey(lambda x, y: x + y)

# Print the first ten elements of each RDD generated in this
# DStream to the console
wordCounts.pprint()

ssc.start()             # Start the computation
ssc.awaitTermination()  # Wait for the computation to terminate
```

What's the problem?
event time vs. processing time

Spark *Structured* Streaming

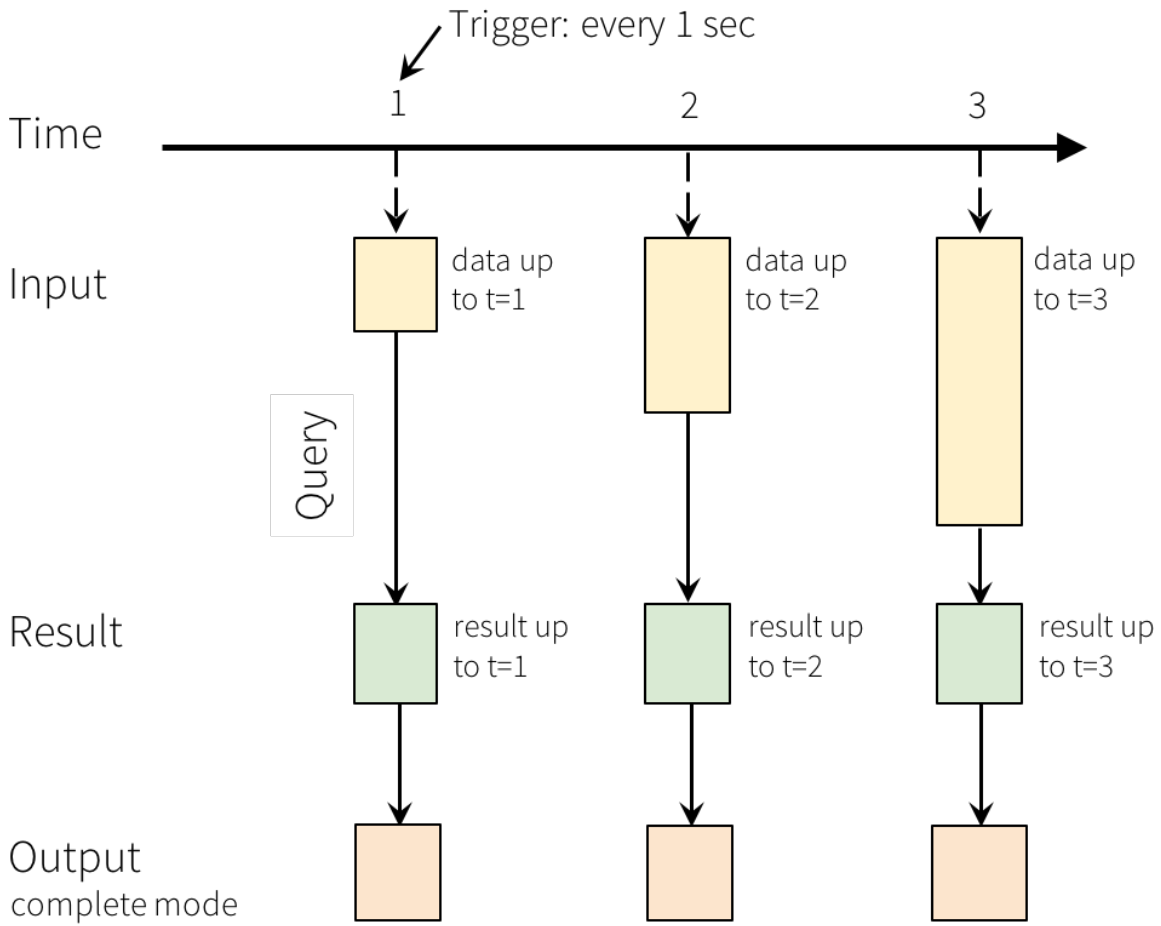
Programming Model

From RDDs to DataFrames
From bounded to unbounded tables



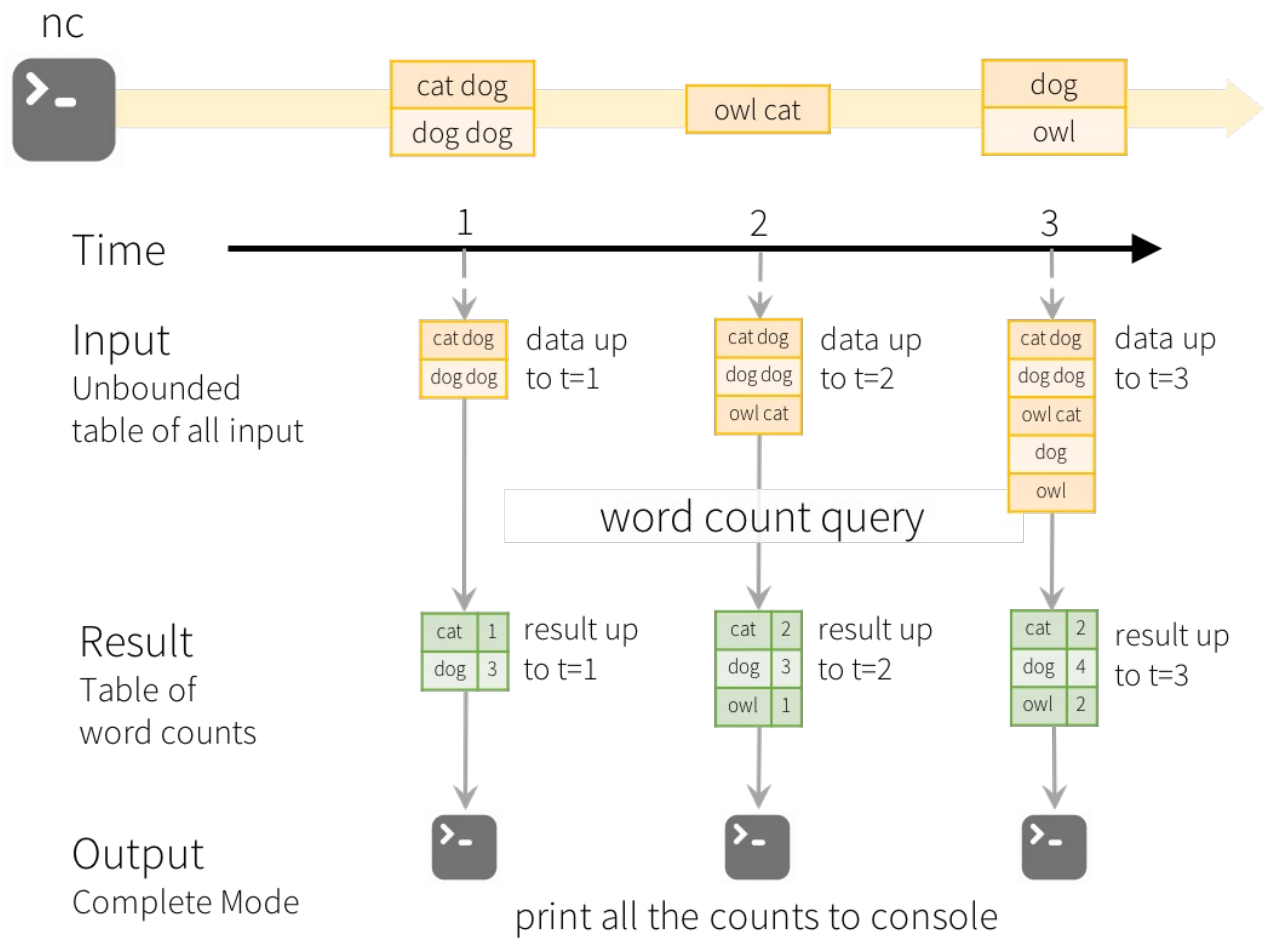
Data stream as an unbounded table

Source: Spark Structured Streaming Documentation



Programming Model for Structured Streaming

Source: Spark Structured Streaming Documentation



Model of the Quick Example

Source: Spark Structured Streaming Documentation

What to do with results?

"Complete" Mode

The entire updated Result Table will be produced

"Append" Mode

Only new rows appended to the Result Table will be produced

"Update" Mode

Only updated rows in the Result Table will be produced

Demo

```
# Create DataFrame streaming input lines from localhost:9999
lines = spark \
    .readStream \
    .format("socket") \
    .option("host", "localhost") \
    .option("port", 9999) \
    .load()

# Split the lines into words
words = lines.select(explode(split(lines.value, " ")).alias("word"))

# Generate running word count
wordCounts = words.groupBy("word").count()

# Start the query, printing counts to console
query = wordCounts \
    .writeStream \
    .outputMode("complete") \
    .format("console") \
    .start()
```

What's the problem?
event time vs. processing time

We need watermarks

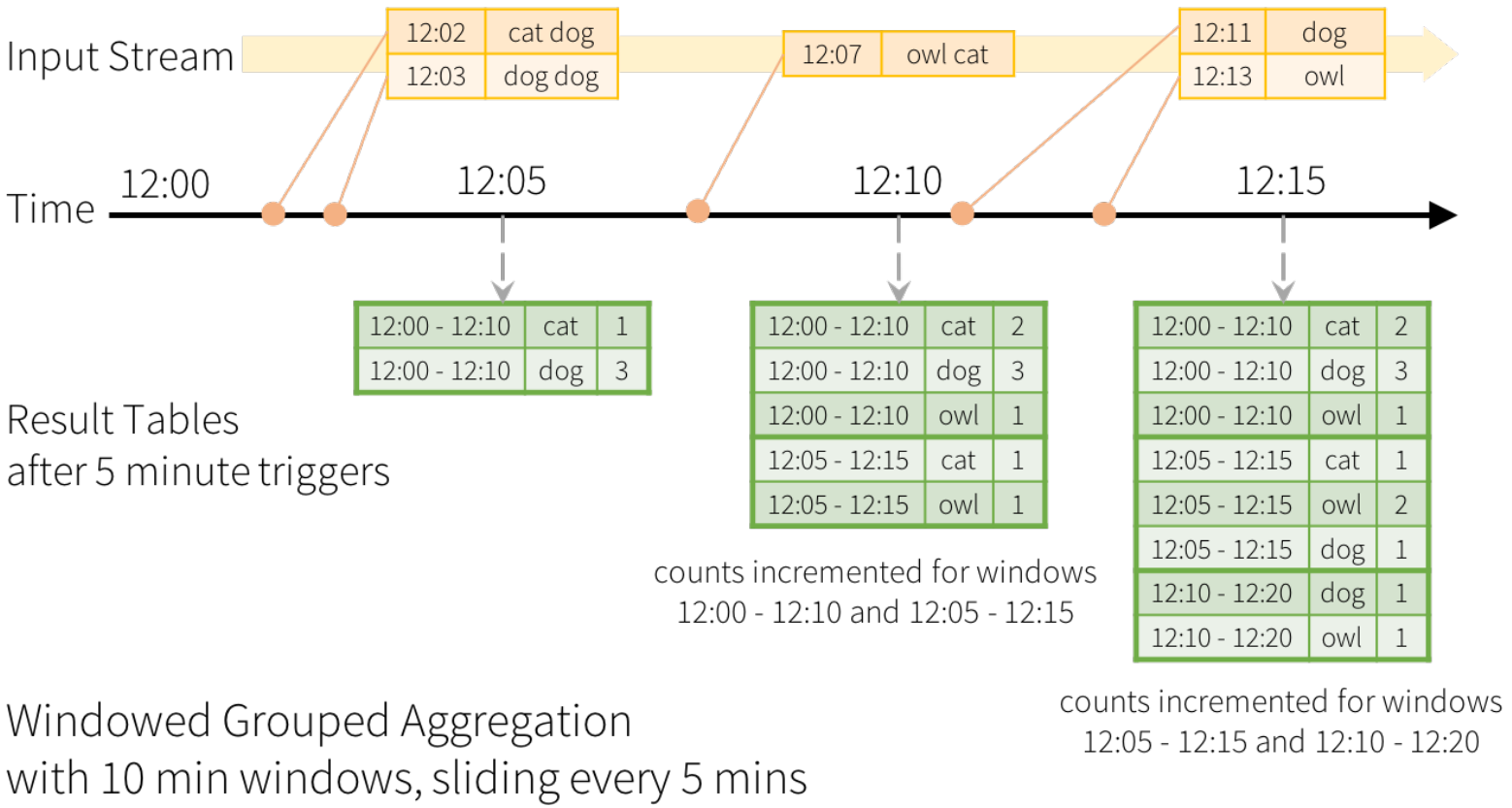
= a notion of input completeness with respect to event times

"all input data with event times less than *X* have been observed"

heuristic vs. perfect

We need triggers

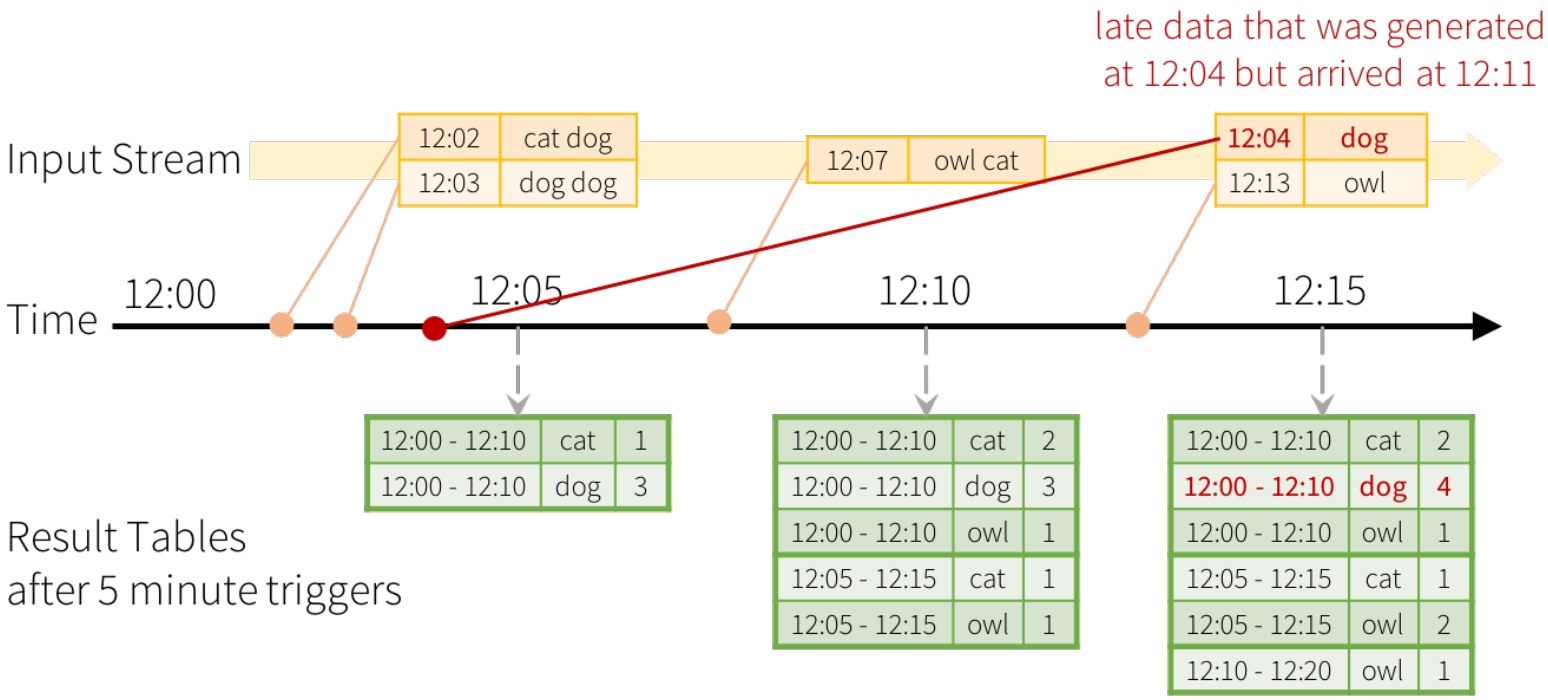
= when the query is actually executed



Source: Spark Structured Streaming Documentation

Windows on Event Time

```
words = ...
# streaming DataFrame of schema
# { timestamp: Timestamp, word: String }

# Group the data by window and word and compute the count of
# each group
windowedCounts = words.groupBy(
    window(words.timestamp, "10 minutes", "5 minutes"),
    words.word
).count()
```



Late data handling in
Windowed Grouped Aggregation

Source: Spark Structured Streaming Documentation

Windows on Event Time

```
words = ...
# streaming DataFrame of schema
# { timestamp: Timestamp, word: String }

# Group the data by window and word and compute the count of
# each group
windowedCounts = words \
    .withWatermark("timestamp", "10 minutes") \
    .groupBy(
        window(words.timestamp, "10 minutes", "5 minutes"),
        words.word) \
    .count()
```

= allow events to be up to 10 minutes late

Kafka Streams

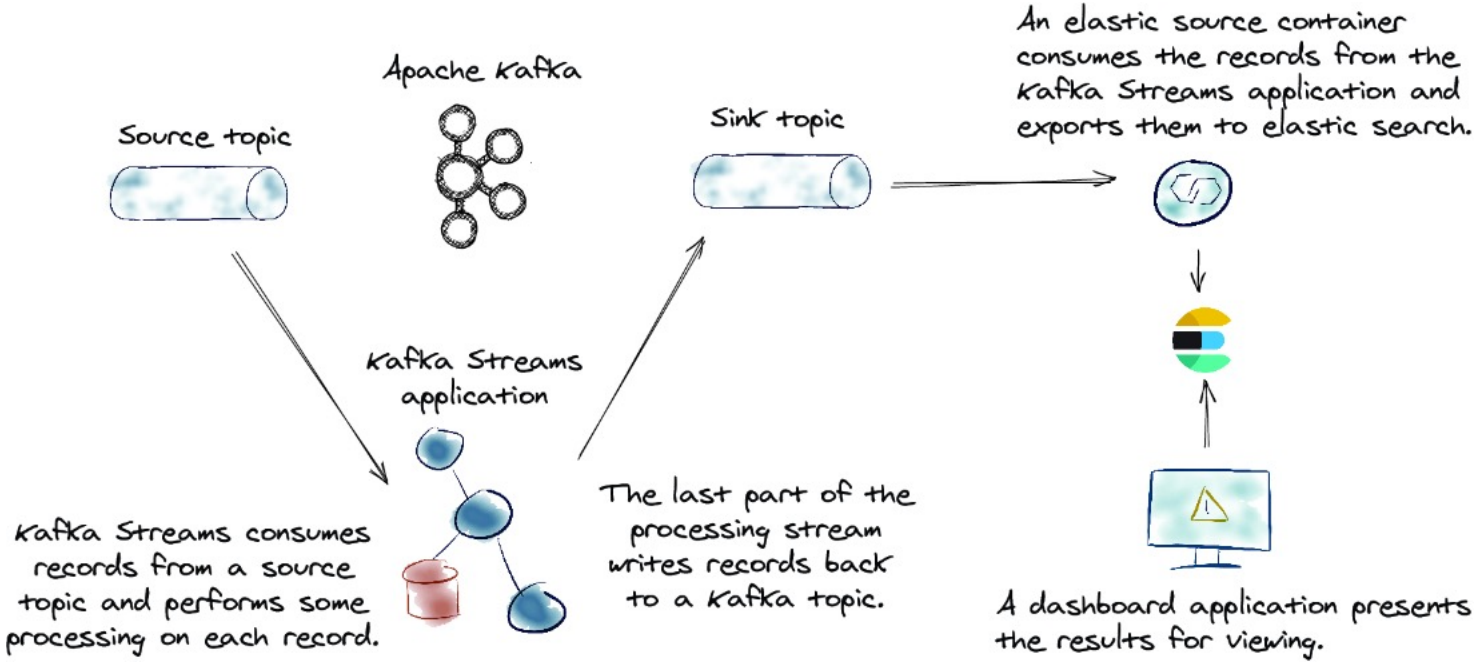
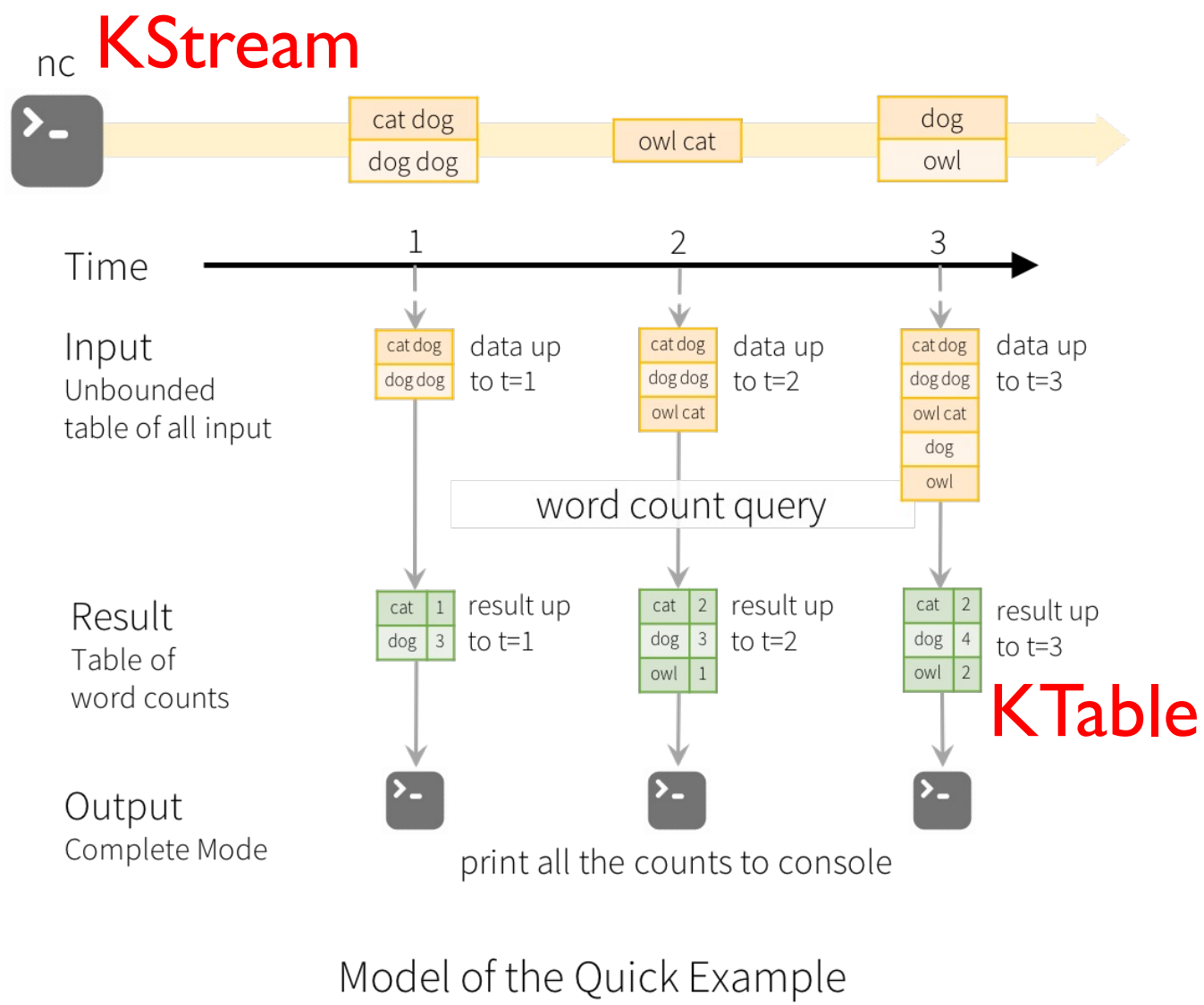


Image from Kafka Streams in Action

Similar Ideas…

```
KStream<String,Double> iotHeatSensorStream =
  builder.stream("heat-sensor-input",
    Consumed.with(stringSerde, doubleSerde));

iotHeatSensorStream.groupByKey()
      .windowedBy(TimeWindows.ofSizeWithNoGrace(
                      Duration.ofMinutes(1))
        .advanceBy(Duration.ofSeconds(10)))
        .aggregate(() -> new IotSensorAggregation(tempThreshold),
         aggregator,
         Materialized.with(stringSerde, aggregationSerde))
         .toStream().to("sensor-agg-output",
           Produced.with(serdeString,
                       sensorAggregationSerde))
```



Model of the Quick Example

Source: Spark Structured Streaming Documentation

Why?

Historical Developments

Kafka: open-sourced 2011

Spark Streaming: released 2013

Kafka Streams: released 2016

Spark Structured Streaming: released 2016

Wait a sec, sounds like you're just building a

Database!

So why not just use one?

TimescaleDB is an open-source
Postgres extension

```
CREATE TABLE sensors(
  id SERIAL PRIMARY KEY,
  type VARCHAR(50),
  location VARCHAR(50)
);

CREATE TABLE sensor_data (
  time TIMESTAMPTZ NOT NULL,
  sensor_id INTEGER,
  temperature DOUBLE PRECISION,
  cpu DOUBLE PRECISION,
  FOREIGN KEY (sensor_id) REFERENCES sensors (id)
) WITH (
  tsdb.hypertable
);
```

```
SELECT
  sensors.location,
  time_bucket('30 minutes', time) AS period,
  AVG(temperature) AS avg_temp,
  last(temperature, time) AS last_temp,
  AVG(cpu) AS avg_cpu
FROM sensor_data JOIN sensors on sensor_data.sensor_id = sensors.id
GROUP BY period, sensors.location;
```

```
Location  |         period         | avg_temp | last_temp | avg_cpu
----------+------------------------+----------+-----------+----------
 ceiling  | 2020-03-31 15:30:00+00 |   25.45  |    24.32  |   0.44
 floor    | 2020-03-31 15:30:00+00 |   43.43  |    80.00  |   0.57
 ceiling  | 2020-03-31 16:00:00+00 |   53.85  |    43.52  |   0.49
 floor    | 2020-03-31 16:00:00+00 |   47.00  |    23.02  |   0.53
 ceiling  | 2020-03-31 16:30:00+00 |   58.78  |    63.66  |   0.49
 floor    | 2020-03-31 16:30:00+00 |   44.61  |     2.22  |   0.43
 ceiling  | 2020-03-31 17:00:00+00 |   35.70  |    42.94  |   0.55
 floor    | 2020-03-31 17:00:00+00 |   62.28  |    52.66  |   0.45
...
```

What happened to event time vs. processing time?

Putting everything together…

Kafka

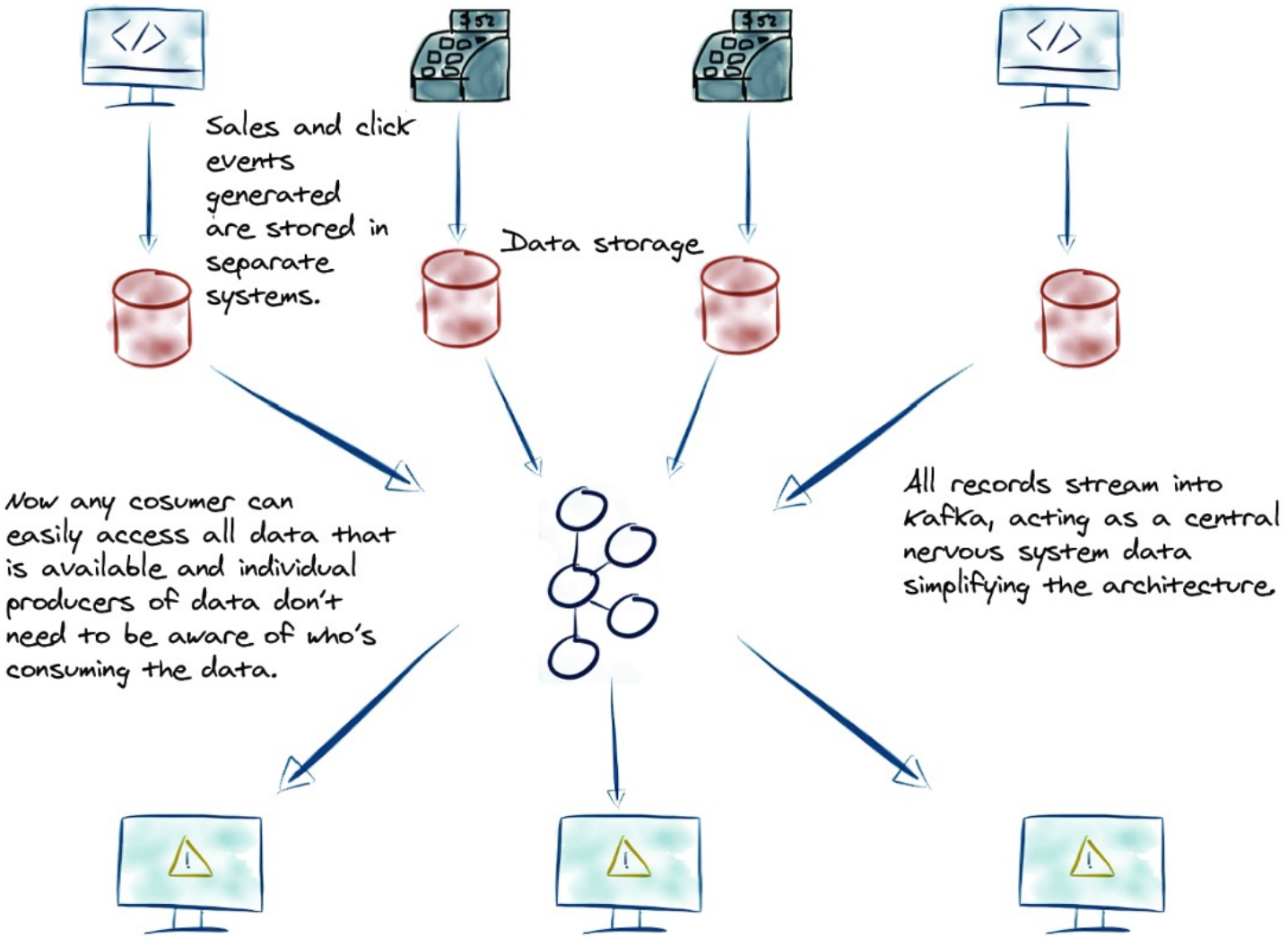


Image from Kafka Streams in Action

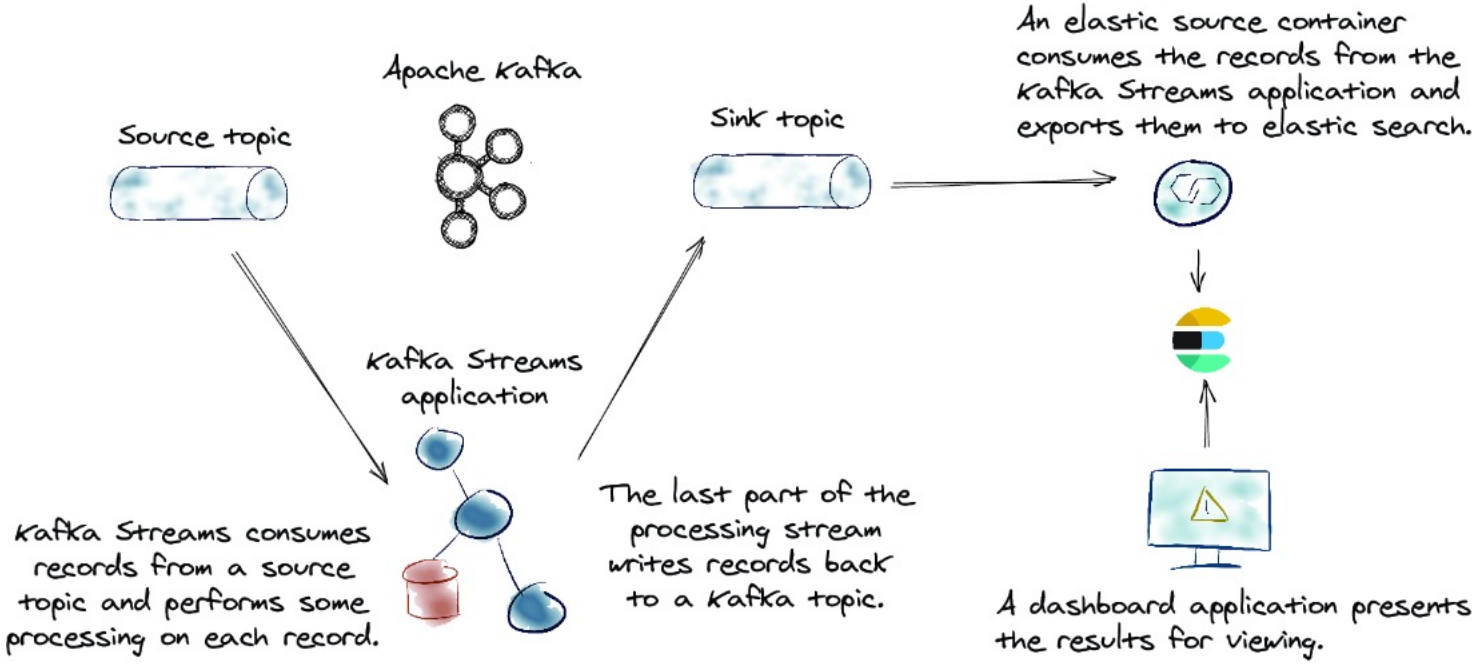
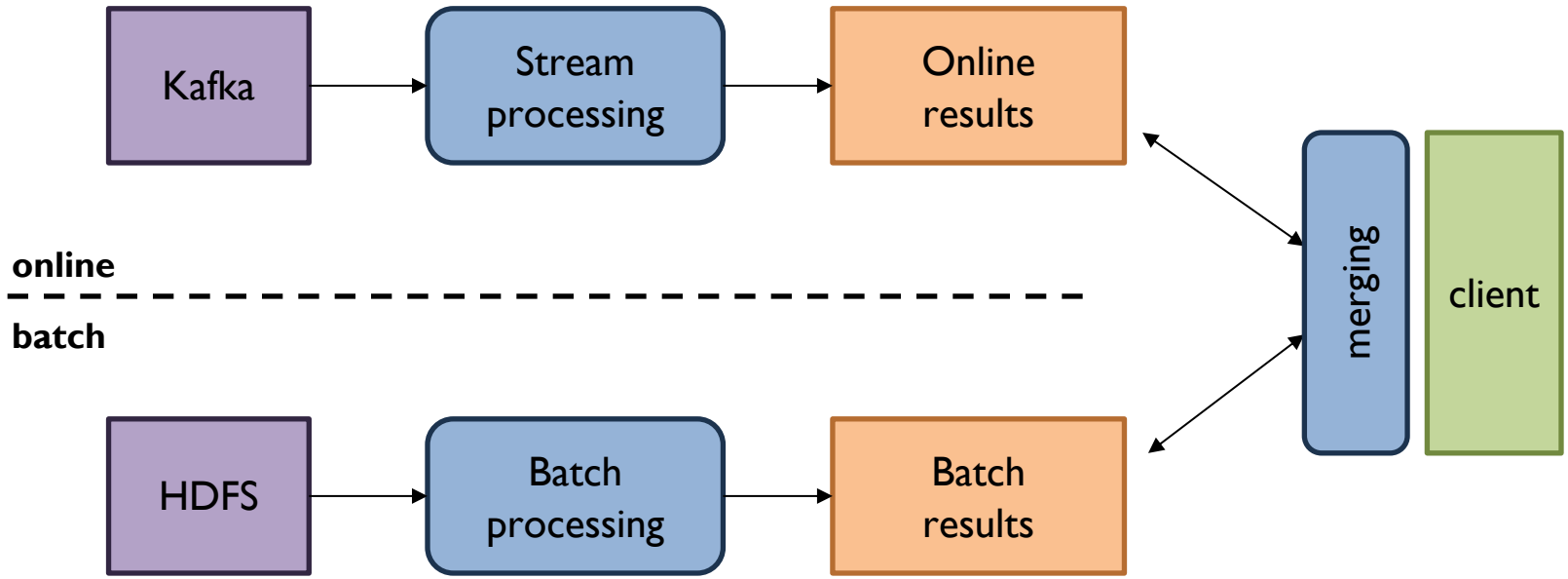


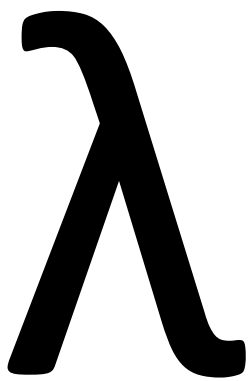
Image from Kafka Streams in Action

How does Kafka fit into your data lake?

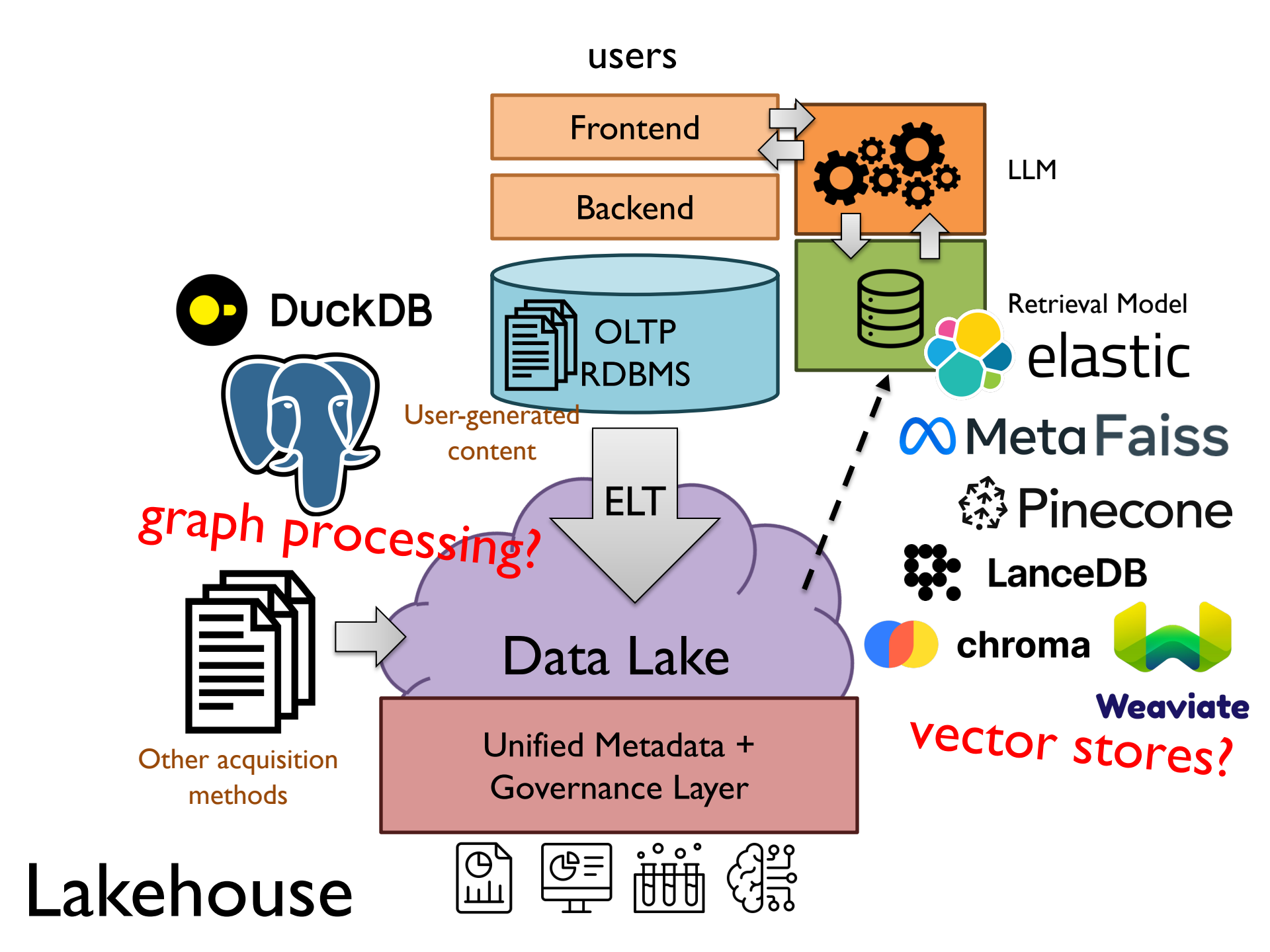
Hybrid Stream/Batch Processing

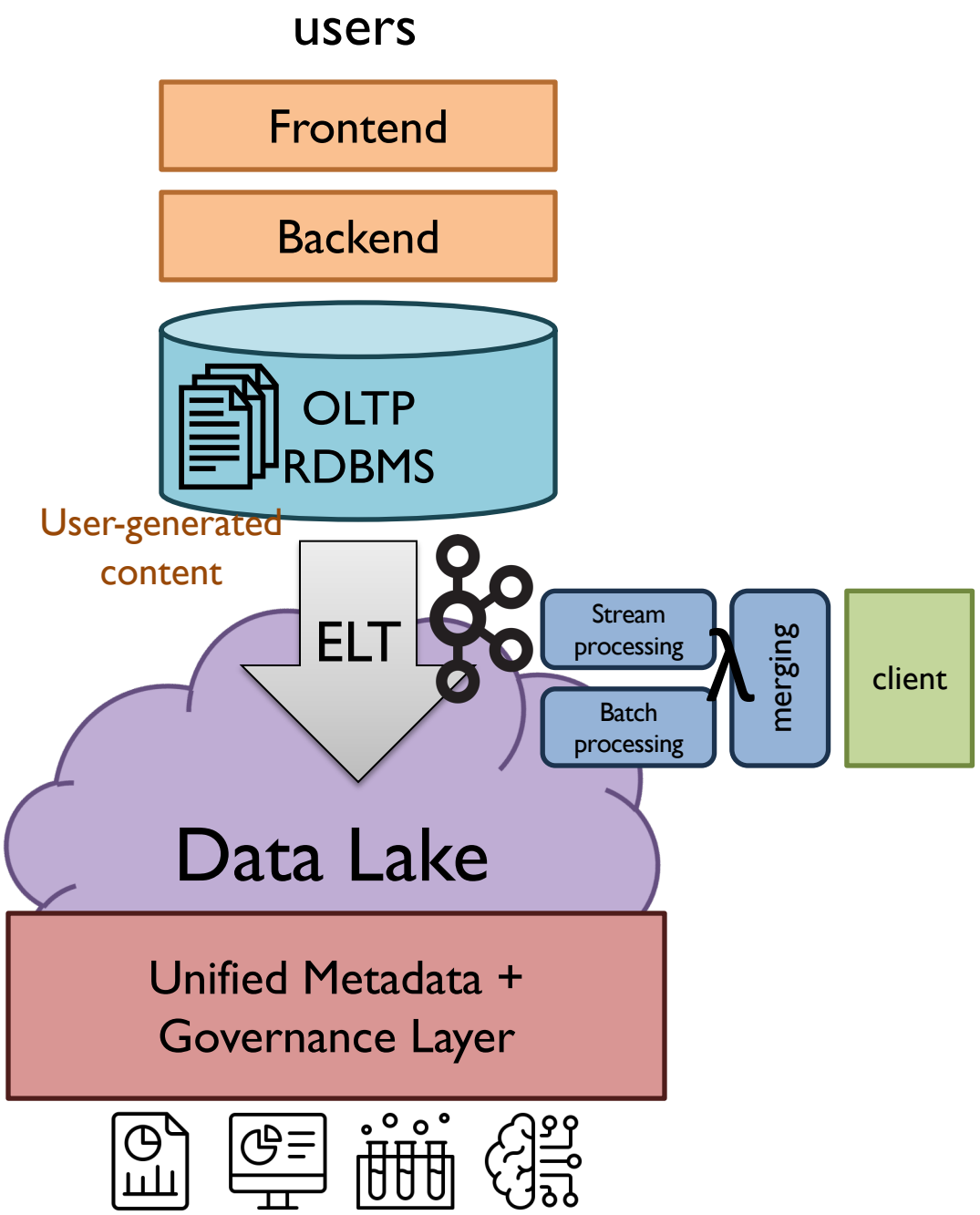
Example: count historical clicks and clicks in real time



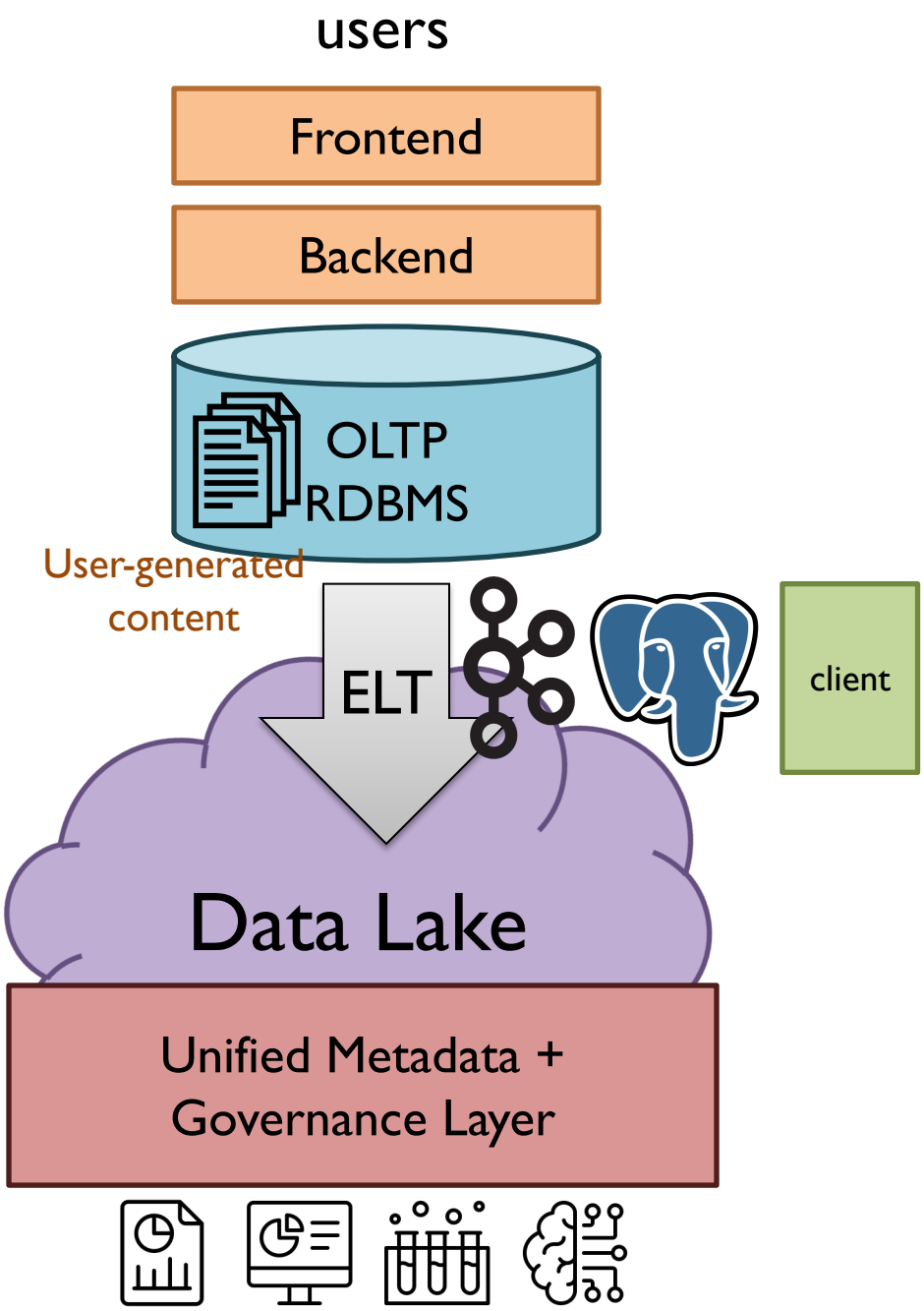


(I hate this.)

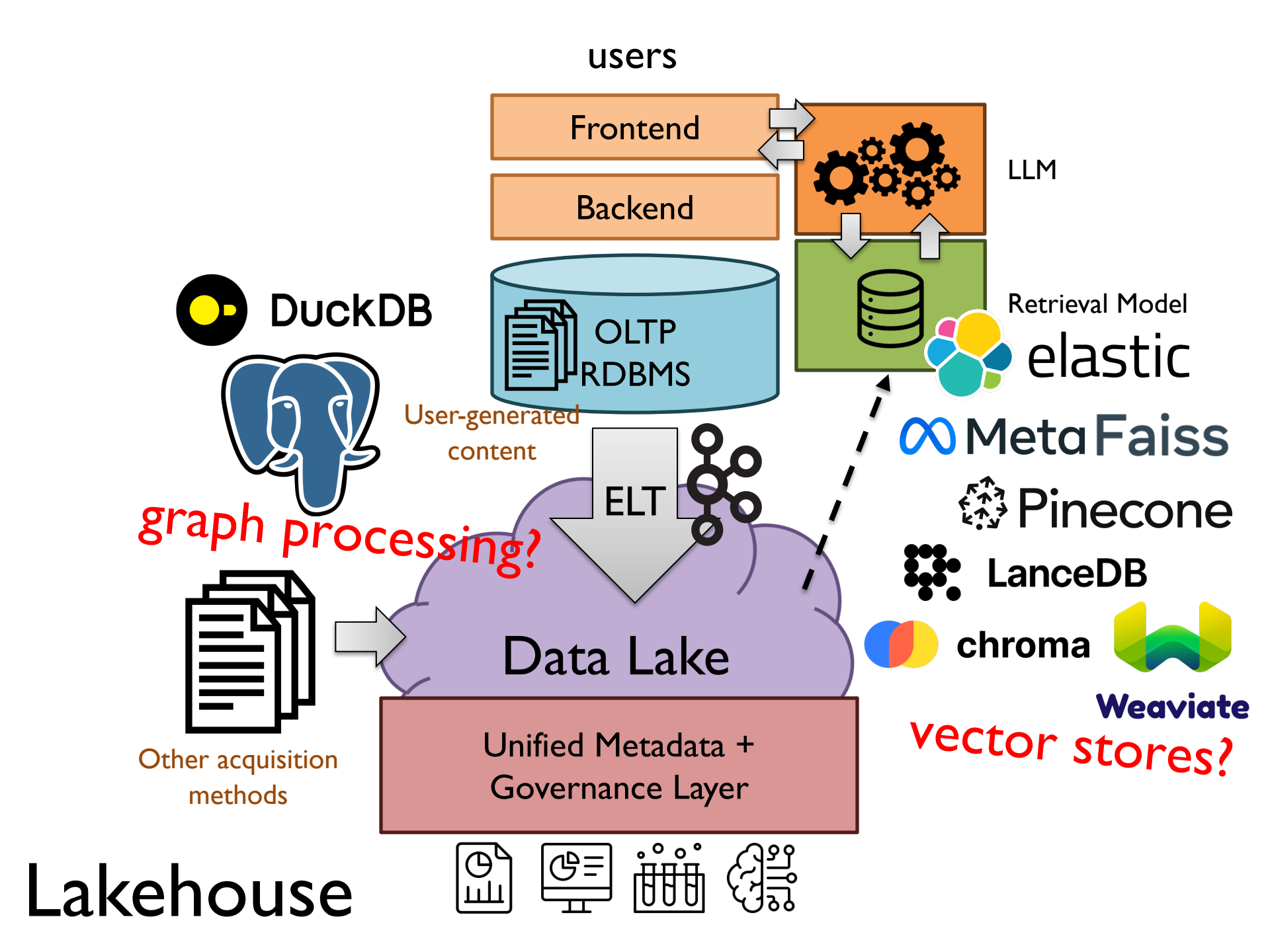

users
Frontend
Backend
LLM
DuckDB
OLTP
RDBMS
Retrieval Model
elastic
MetaFaiss
Pinecone
LanceDB
chroma
Weaviate
User-generated
content
ELT
graph processing?
Data Lake
Unified Metadata +
Governance Layer
Other acquisition
methods
vector stores?
Lakehouse

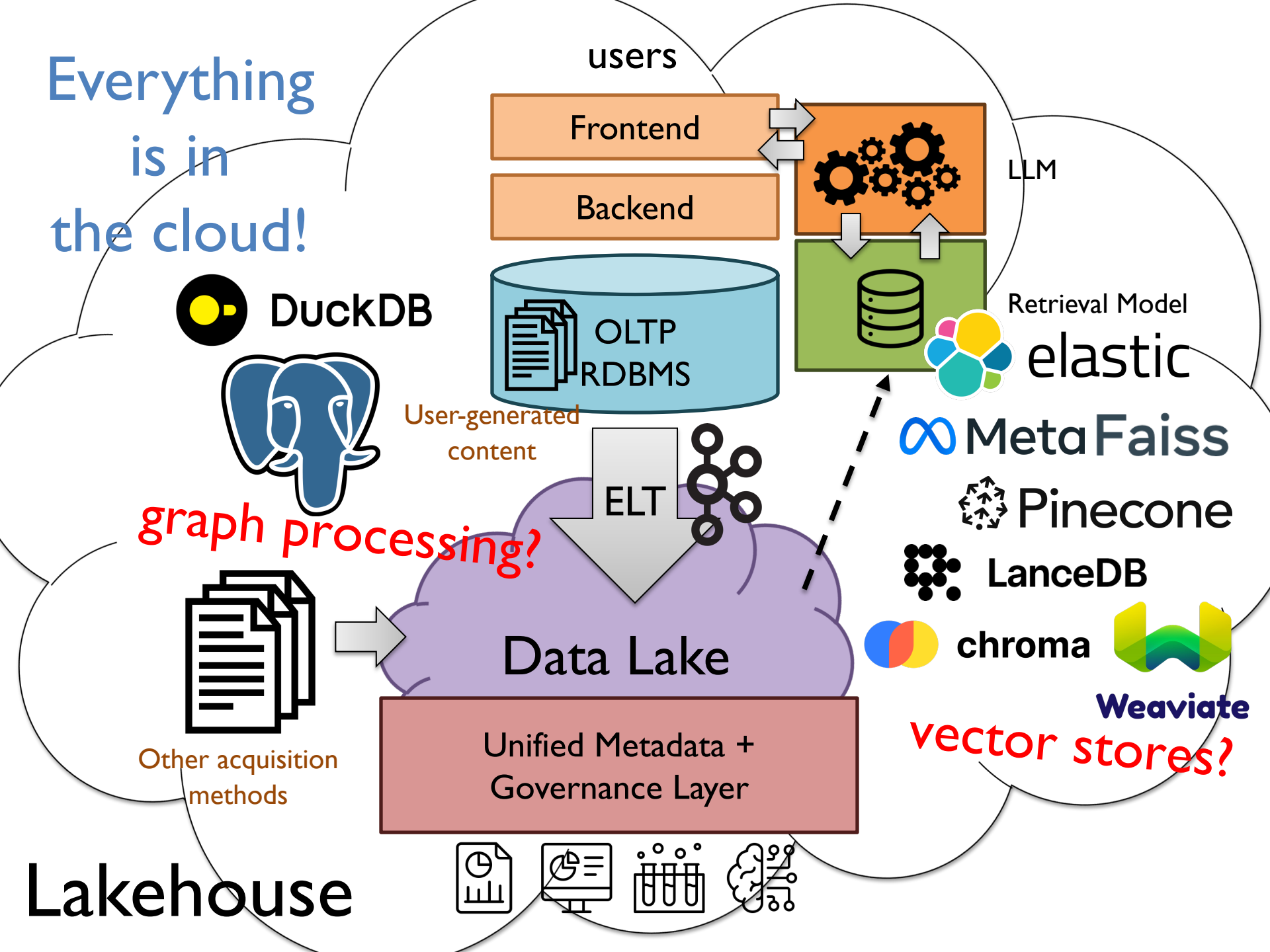

users
Frontend
Backend
OLTP
RDBMS
User-generated
content
ELT
Stream
processing
Batch
processing
merging
client
Data Lake
Unified Metadata +
Governance Layer

Lakehouse


users
Frontend
Backend
OLTP
RDBMS
User-generated
content
ELT
client
Data Lake
Unified Metadata +
Governance Layer

Lakehouse

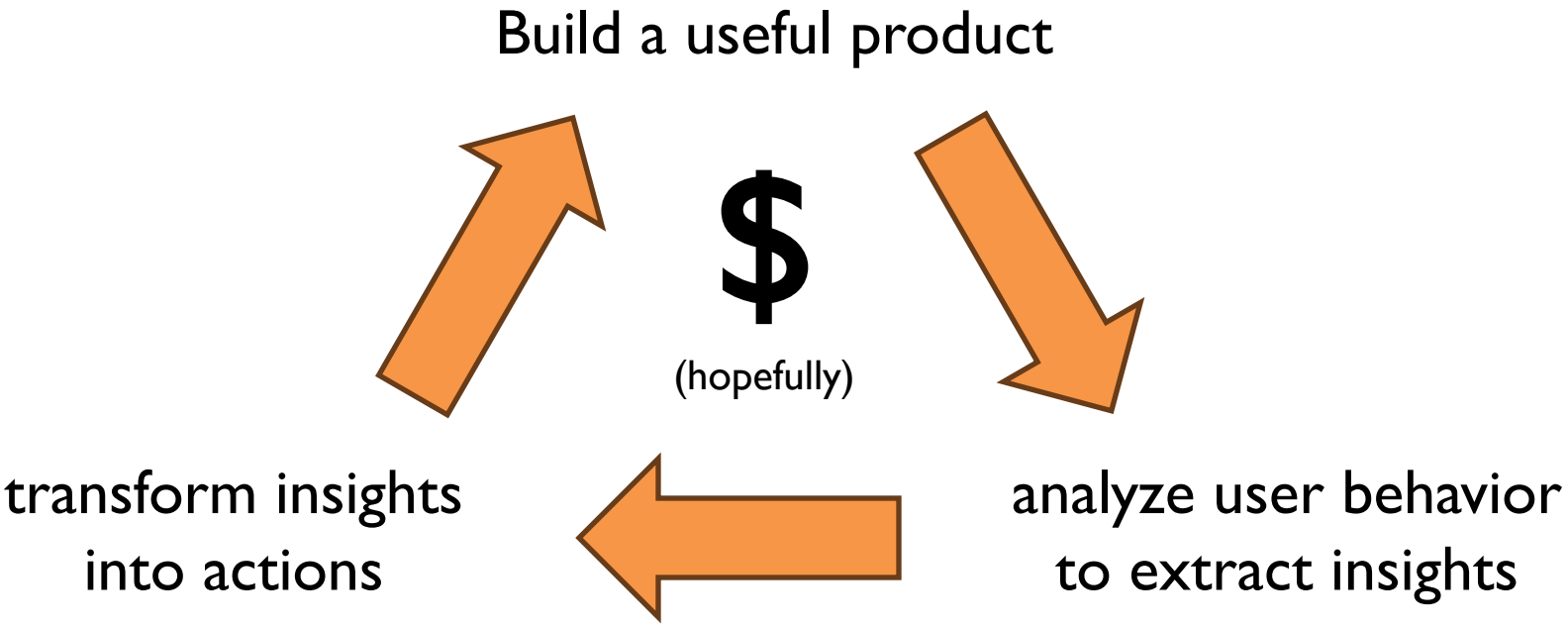

users
Frontend
Backend
LLM
DuckDB
OLTP
RDBMS
Retrieval Model
elastic
MetaFaiss
Pinecone
LanceDB
chroma
Weaviate
User-generated
content
ELT
graph processing?
Data Lake
Unified Metadata +
Governance Layer
Other acquisition
methods
vector stores?
Lakehouse


Everything is in the cloud!
users
Frontend
Backend
LLM
OLTP RDBMS
Retrieval Model
elastic
DuckDB
User-generated content
ELT
graph processing?
MetaFaiss
Pinecone
LanceDB
chroma
Weaviate
vector stores?
Data Lake
Unified Metadata + Governance Layer
Other acquisition methods
Lakehouse

Back to the beginning…

The Data Flywheel

(a virtuous cycle)



Google. Facebook. Twitter. Amazon. Uber.

What's this course about?

The *infrastructure* that supports the data flywheel.

data platforms

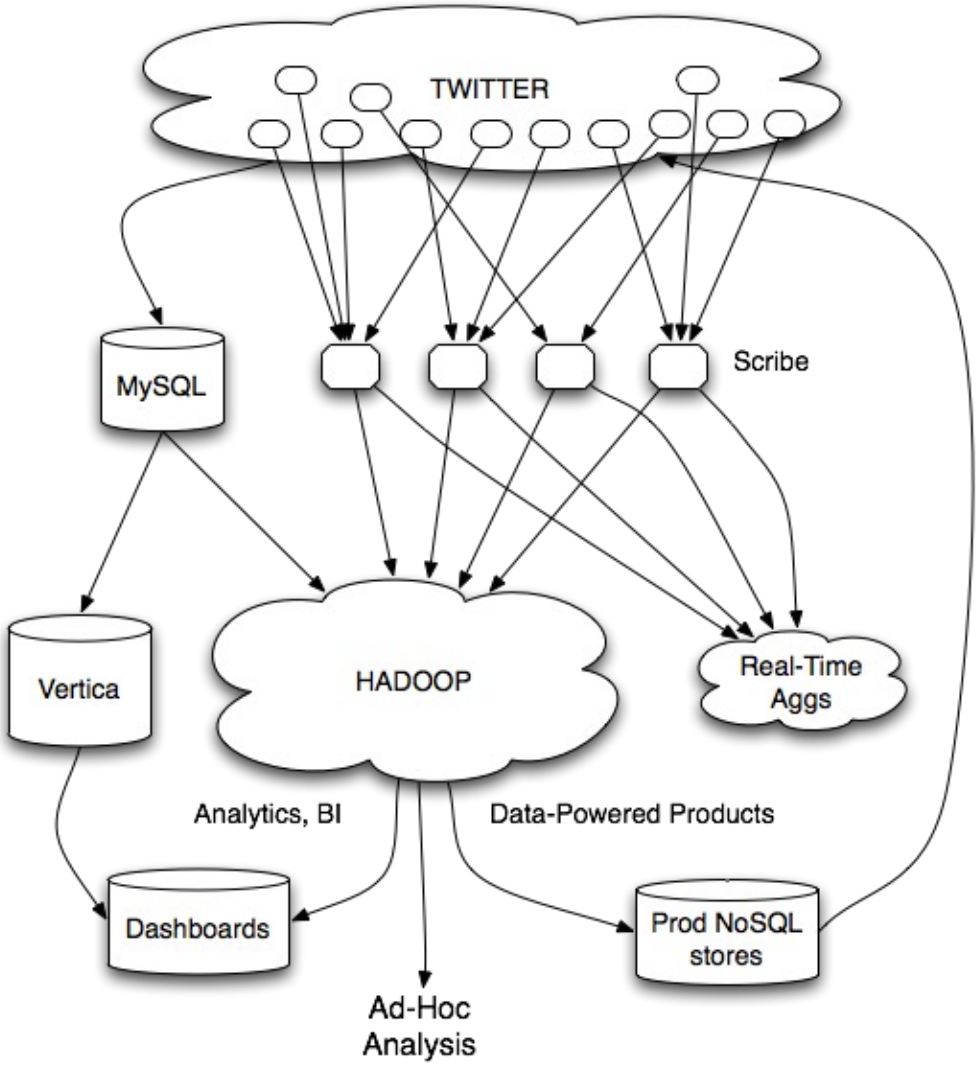
lakehouse = data warehouse + data lake

data engineering

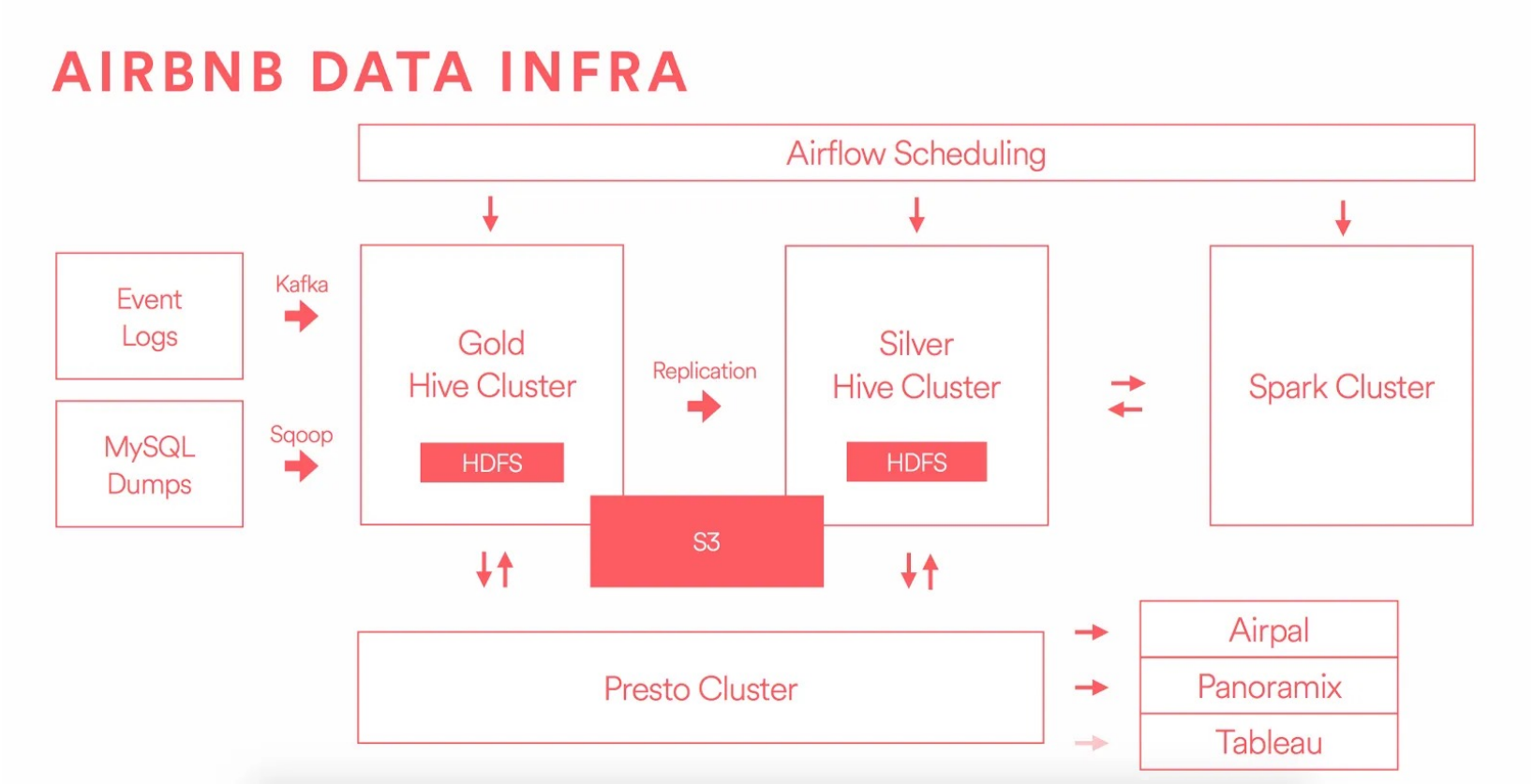
What problems do data platforms solve?

Ingesting, storing, manipulating, maintaining, serving…
the data that supports the data flywheel.

Some examples…

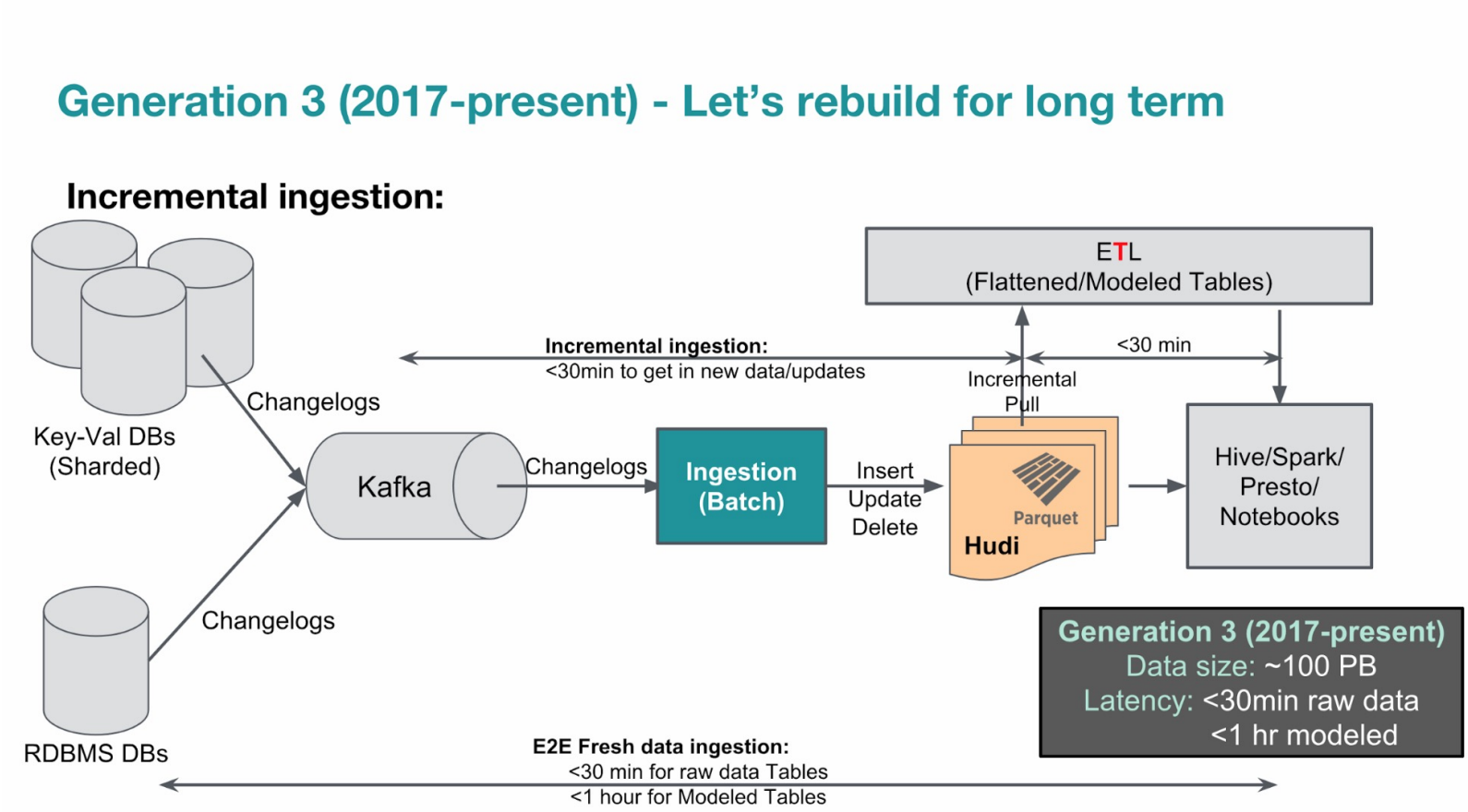


Twitter's data platform (circa 2012)



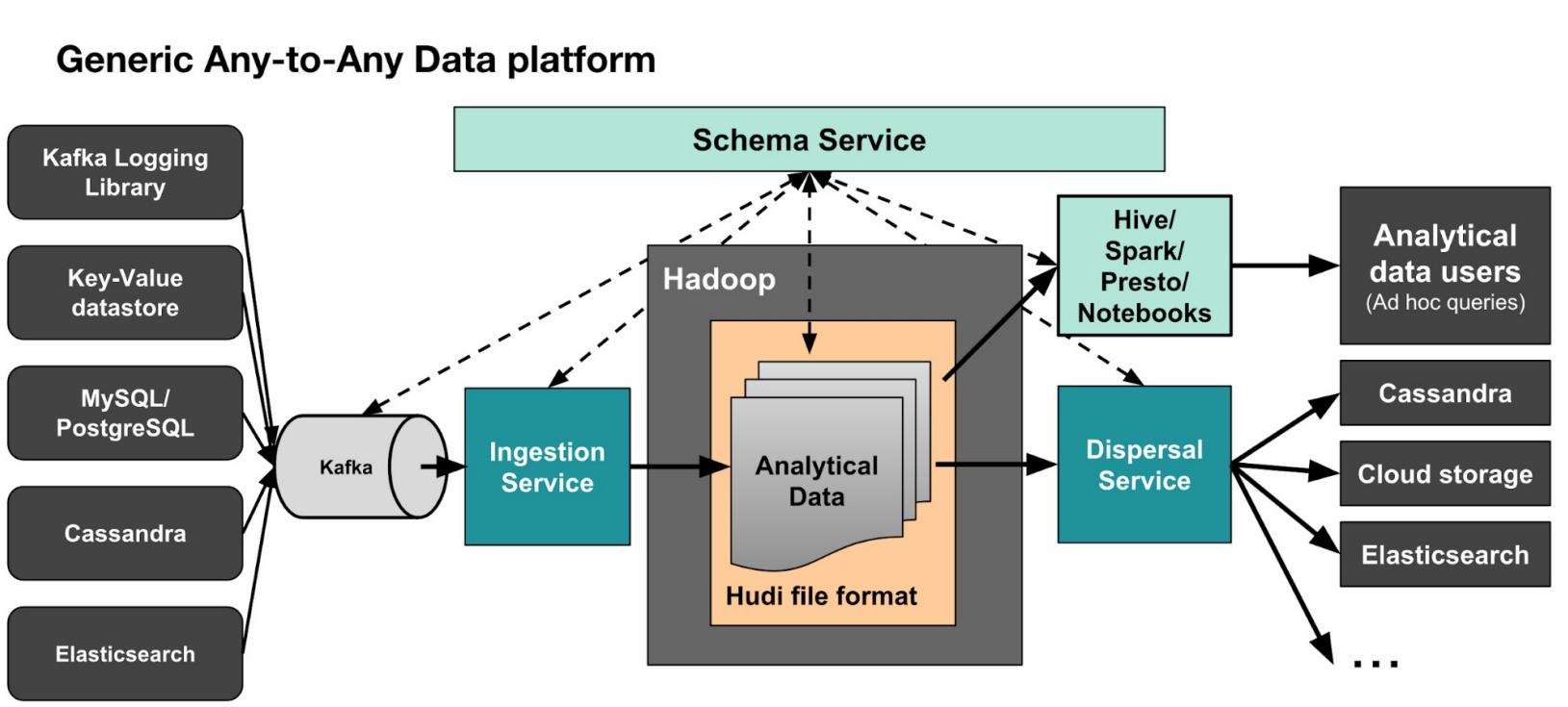
AirBnB's data platform (circa 2016)

https://medium.com/airbnb-engineering/data-infrastructure-at-airbnb-8adfb34f169c



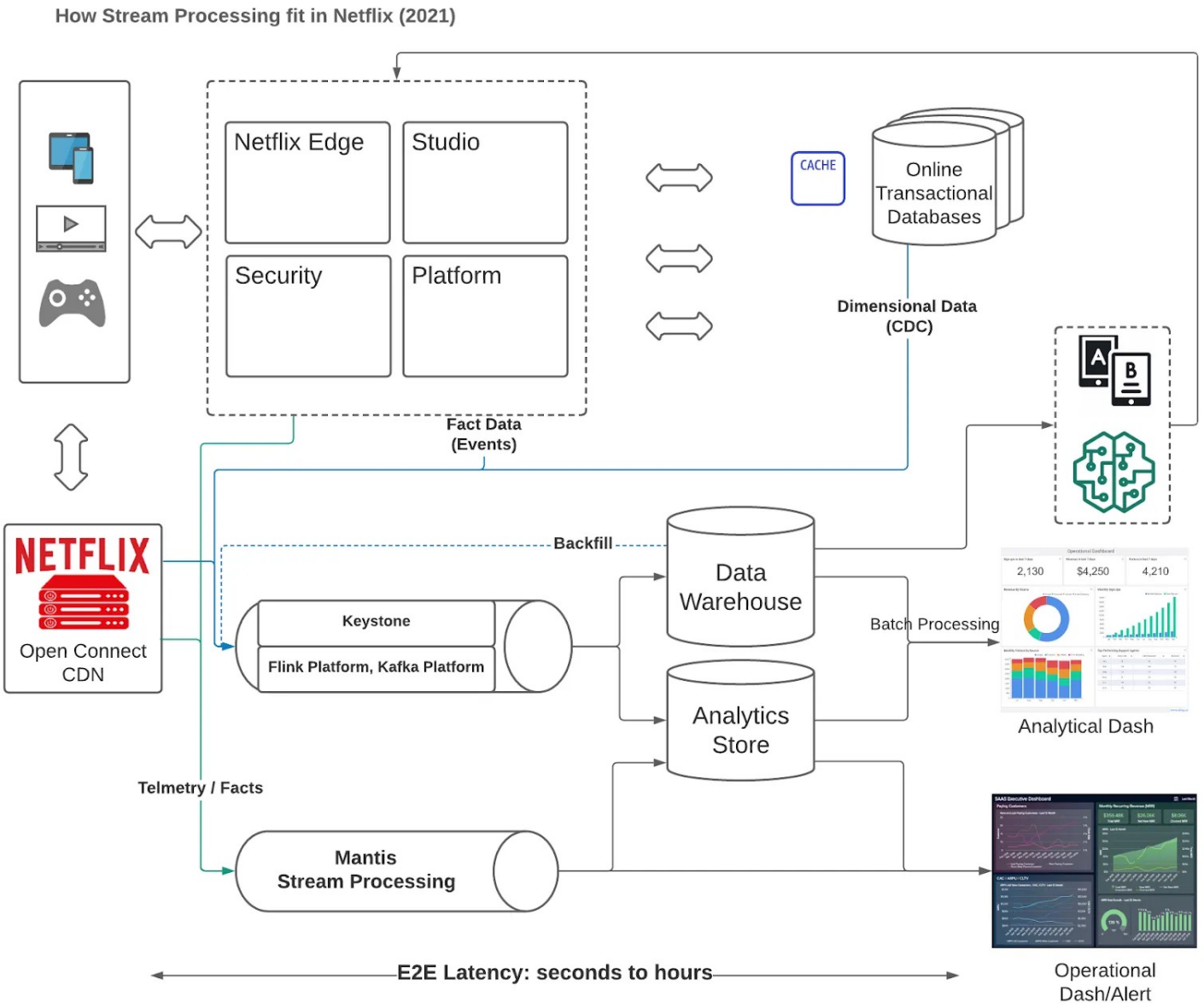
Uber's data platform (circa 2018)

https://www.uber.com/en-CA/blog/uber-big-data-platform/



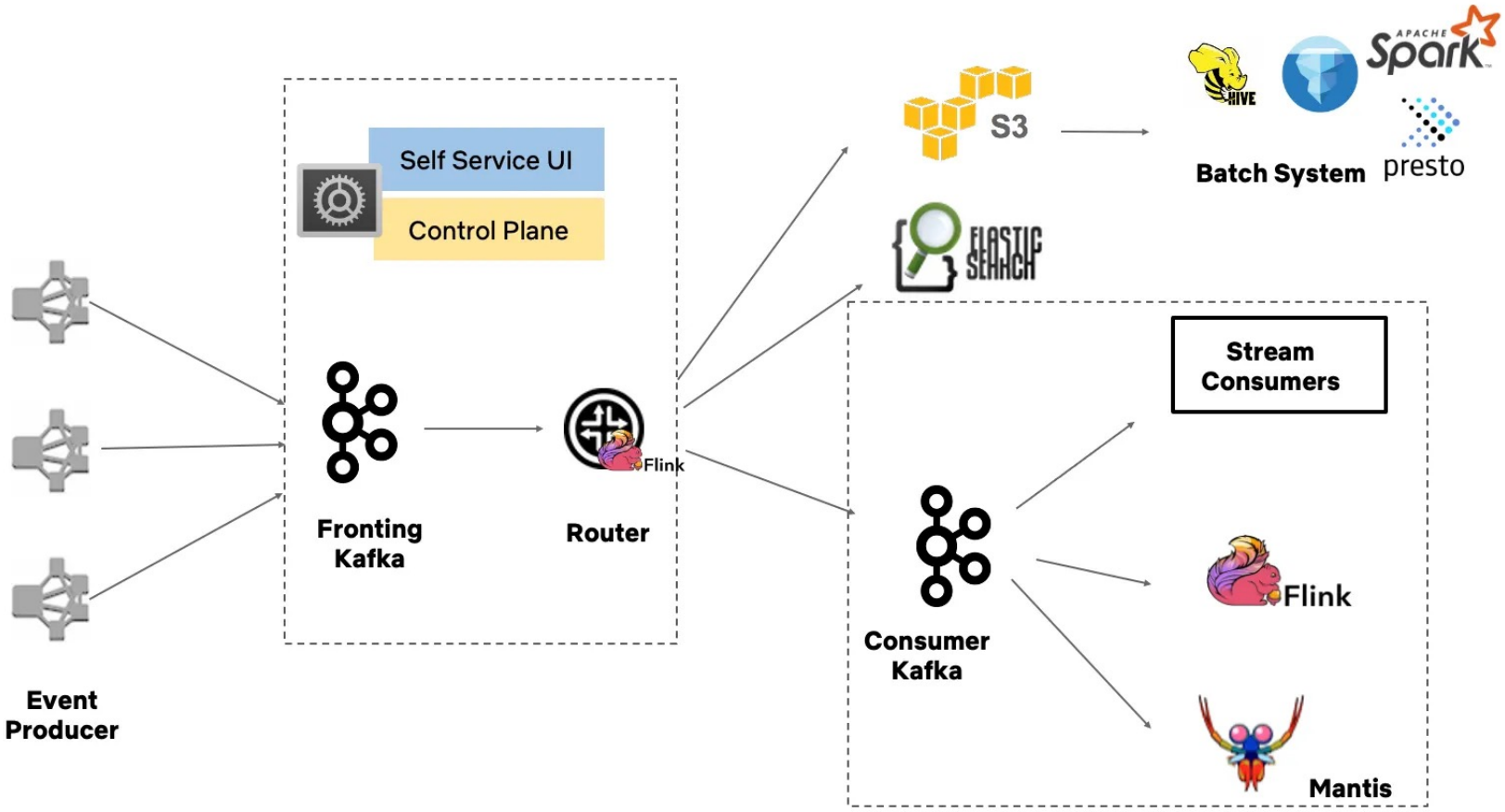
Uber's data platform (circa 2018)

https://www.uber.com/en-CA/blog/uber-big-data-platform/



Netflix's data platform (circa 2021)

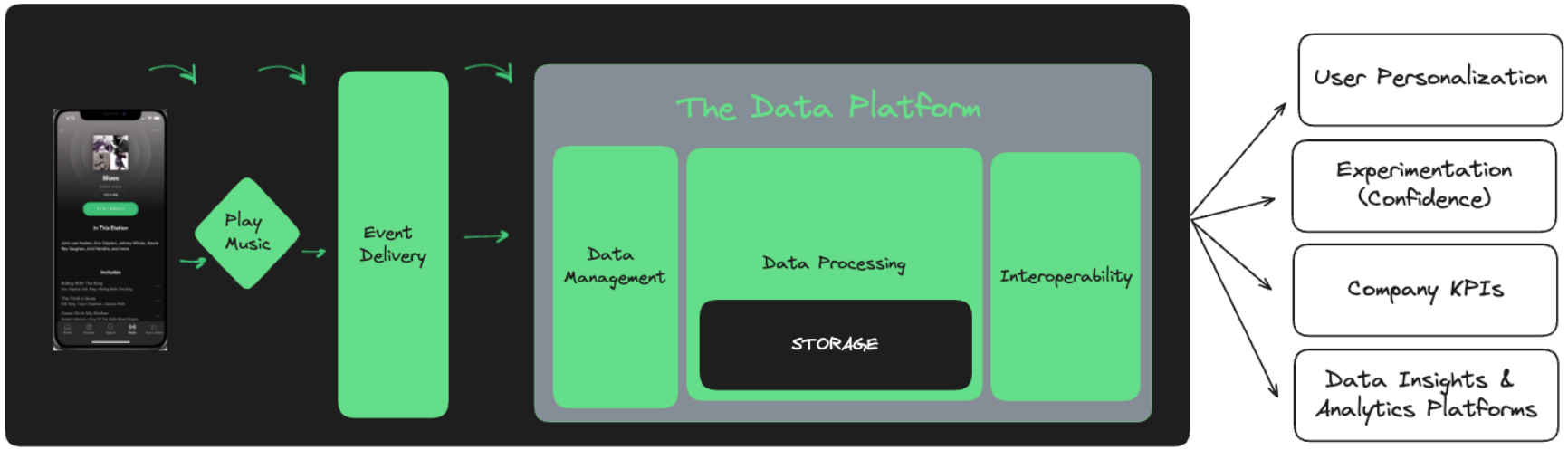
https://zhenzhongxu.com/the-four-innovation-phases-of-netflixs-trillions-scale-real-time-data-infrastructure-2370938d7f01



Publish, Collect, Move & Compute event data

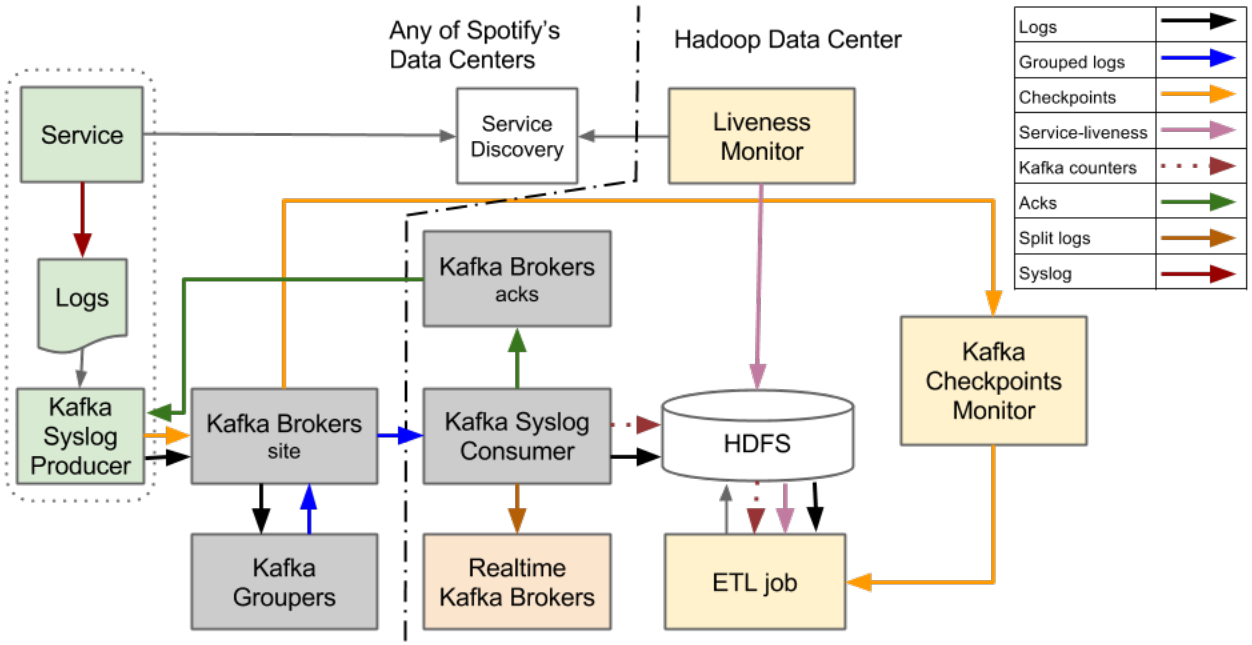
Netflix's data platform (circa 2021)

https://zhenzhongxu.com/the-four-innovation-phases-of-netflixs-trillions-scale-real-time-data-infrastructure-2370938d7f01



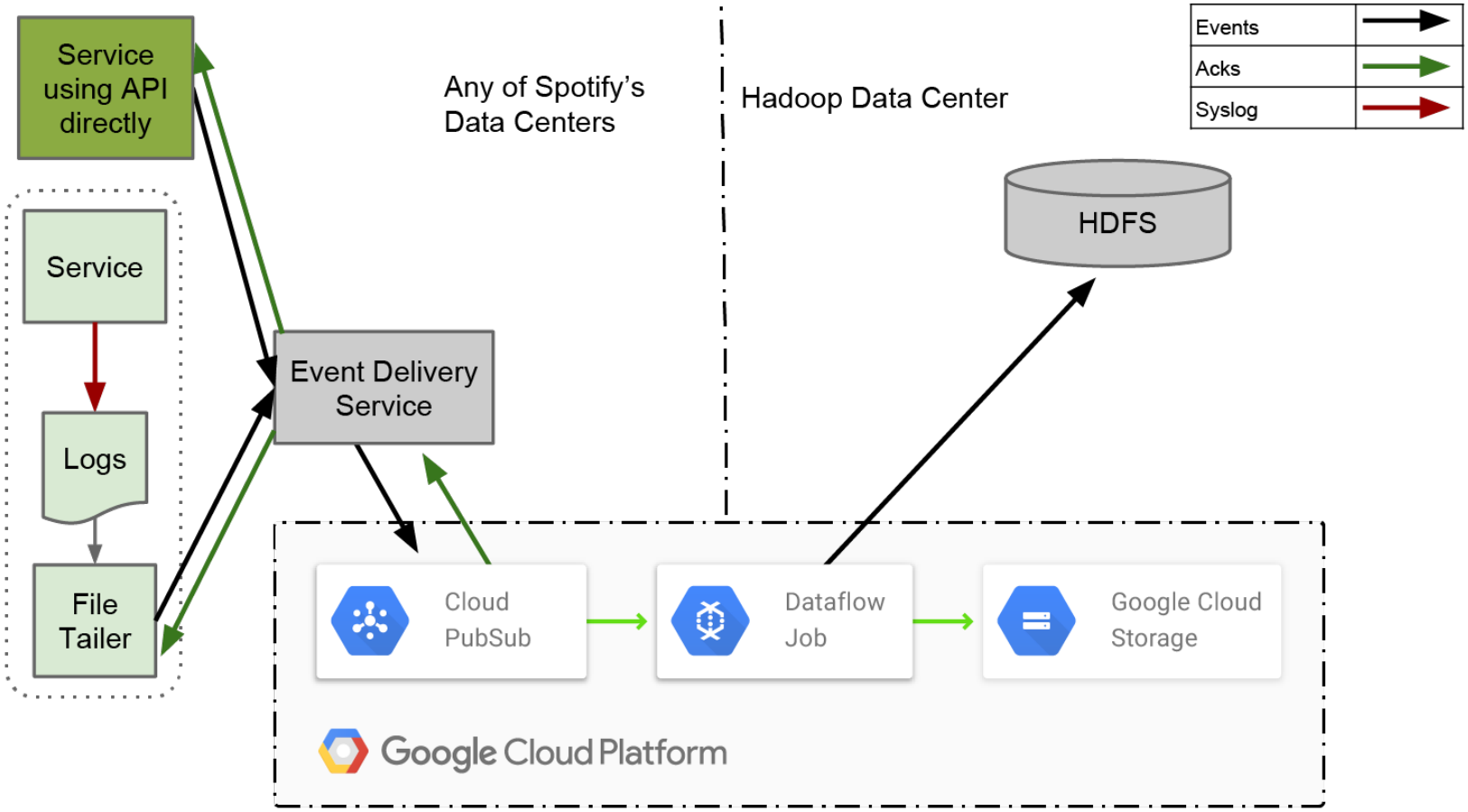
Spotify's data platform (circa 2024)

https://engineering.atspotify.com/2024/5/data-platform-explained-part-ii



Spotify's data delivery architecture (circa 2016, pre-cloud)

https://engineering.atspotify.com/2016/02/spotifys-event-delivery-the-road-to-the-cloud-part-i



Spotify's data delivery architecture (circa 2016, post-cloud)

https://engineering.atspotify.com/2016/3/spotifys-event-delivery-the-road-to-the-cloud-part-ii

What problems do data platforms solve?

Ingesting, storing, manipulating, maintaining, serving…
the data that supports the data flywheel.


富嶽三十六景
神奈川沖
浪裏