The Future of Real-Time in Spark

Cheng Lian <lian@databricks.com>

Spark Summit China, Beijing, May 15, 2016

databricks

Why Real-time?

Making decisions faster is valuable.

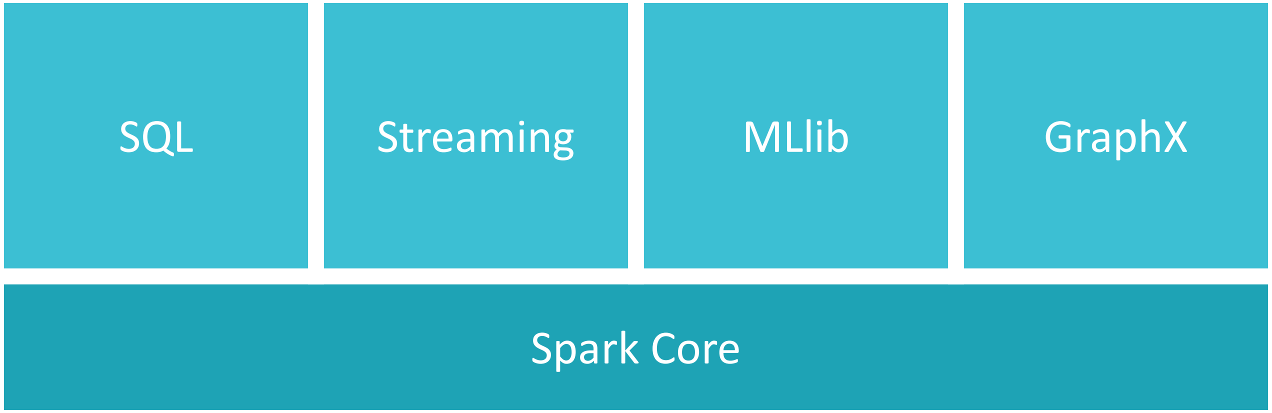
- Preventing credit card fraud
- Monitoring industrial machinery
- Human-facing dashboards
- …

Streaming Engine

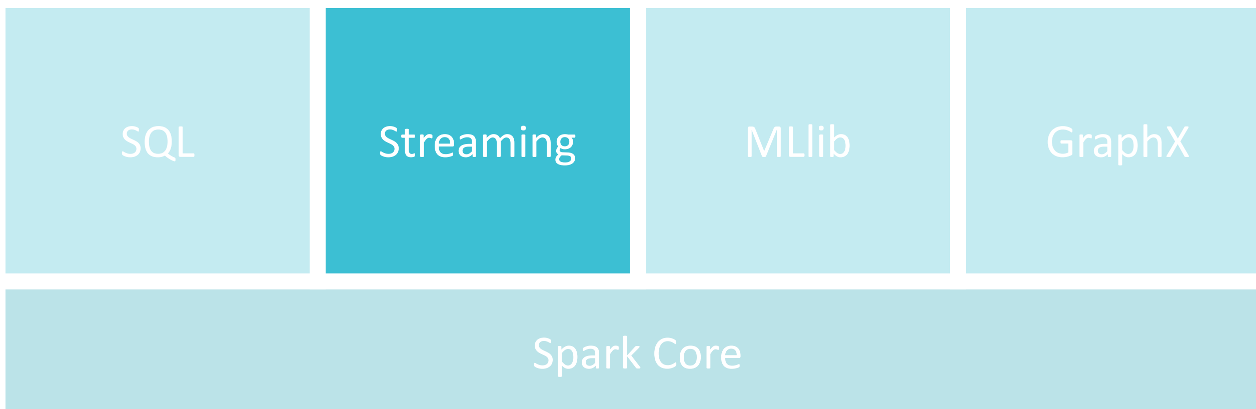
Noun.

Takes an input stream and produces an output stream.

Spark Unified Stack



Spark Unified Stack



Introduced 3 years ago in Spark 0.7

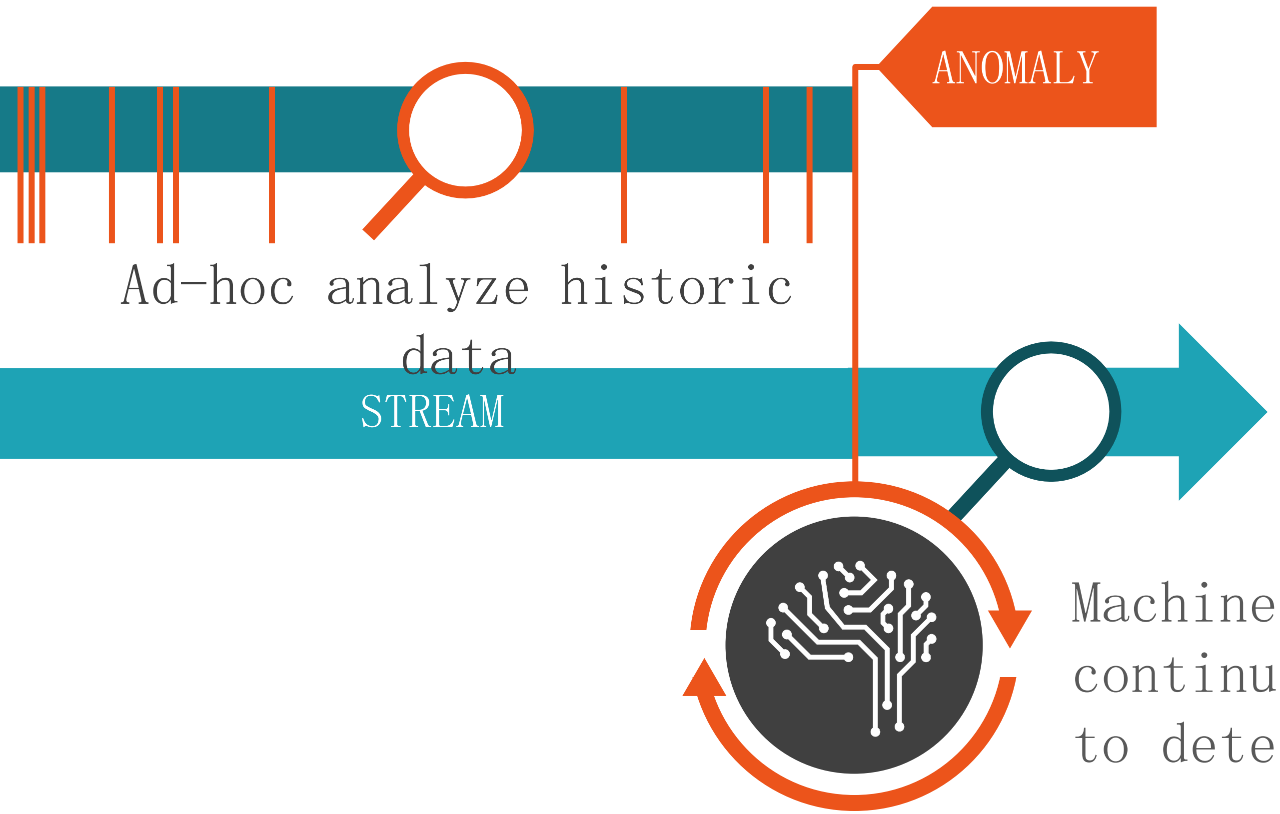
50% users consider most important part of Spark

Spark Streaming

- First attempt at unifying streaming and batch
- State management built in
- Exactly-once semantics
- Features required for large clusters
  - Straggler mitigation
  - Dynamic load balancing
  - Fast fault-recovery

Streaming computations don’t run in isolation.

Use Case: Fraud Detection



Machine learning model continuously updates to detect new anomalies

Continuous Application

Noun.

An end-to-end application that acts on real-time data.

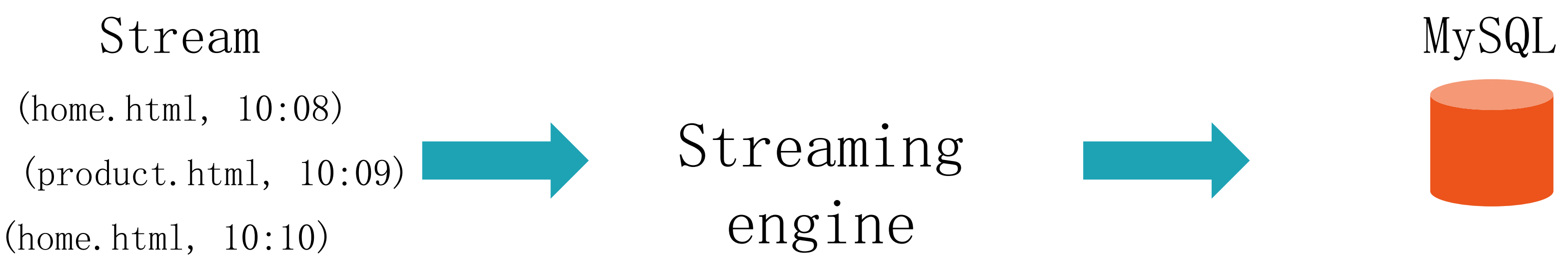
Challenges Building Continous Application

Integration with non-streaming systems often an after-thought

- Interactive, batch, relational databases, machine learning…

Streaming programming models are complex

Integration Example



Page	Minute	Visits
home	10:09	21
pricing	10:10	30
...	...	...

What can go wrong?

- Late events
- Partial outputs to MySQL
- State recovery on failure
- Distributed reads/writes
- ...

Complex Programming Models

Data
Late arrival, varying distribution over time, …

Processing
Business logic change & new ops
(windows, sessions)

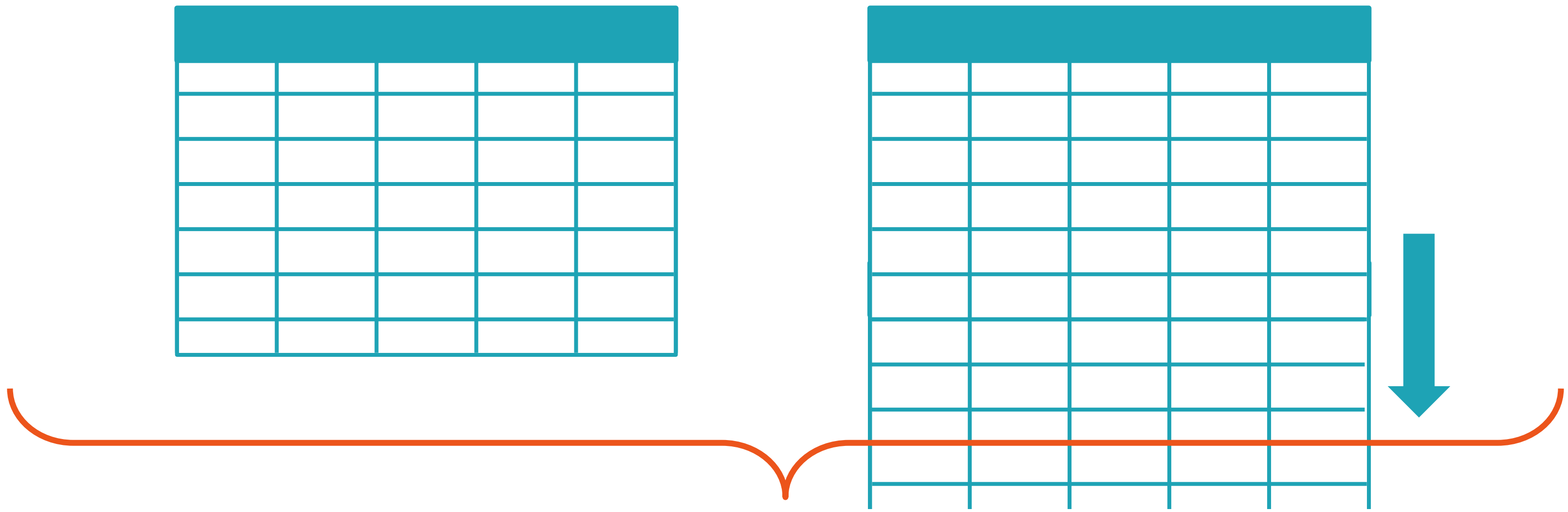
Output
How do we define
output over time & correctness?

Structured Streaming

The simplest way to perform streaming analysis is not having to **reason** about streaming.

Spark 1.3
Static DataFrames

Spark 2.0
Infinite DataFrames



Single API !

Structured Streaming

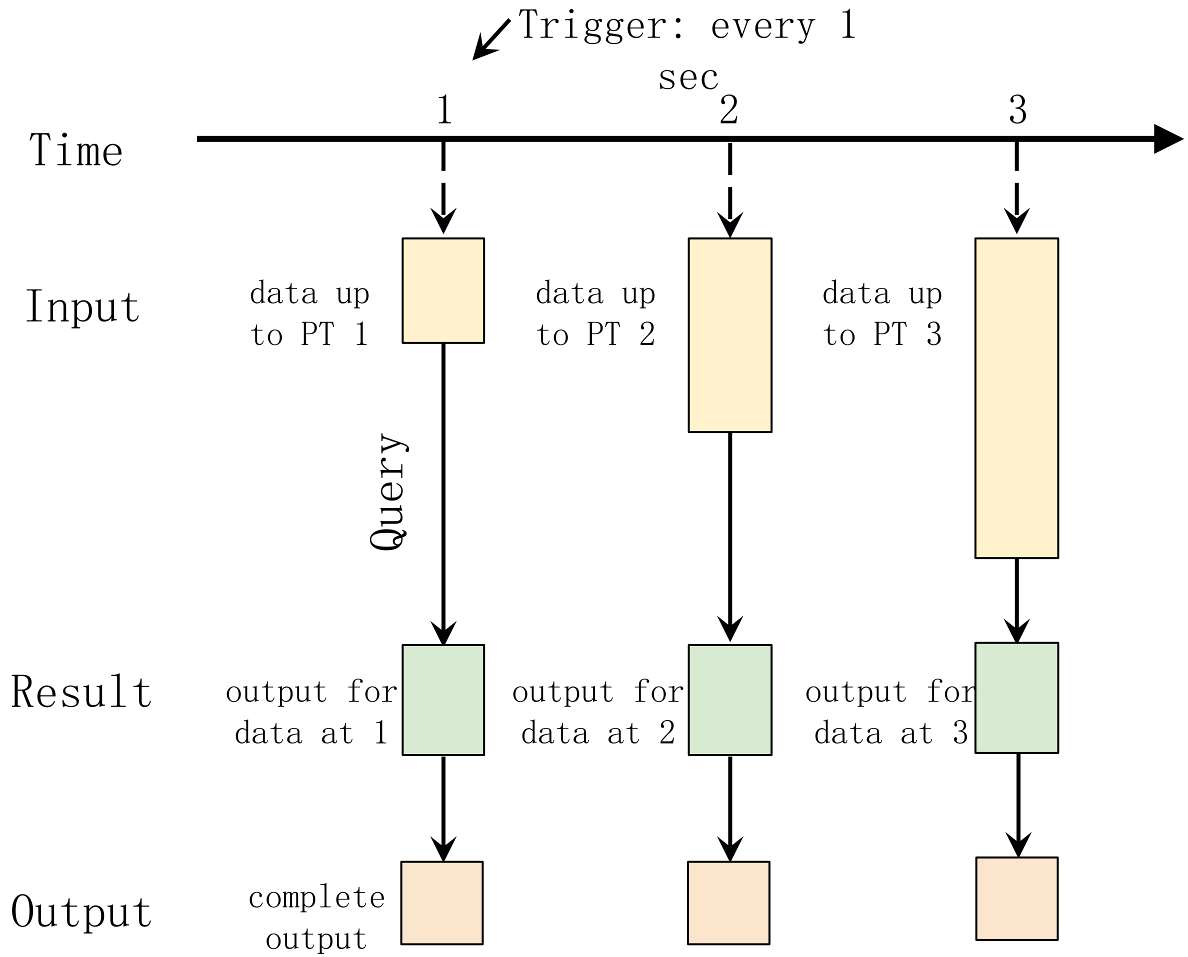
High-level streaming API built on Spark SQL engine

- Runs the same queries on DataFrames
- Event time, windowing, sessions, sources, & sinks

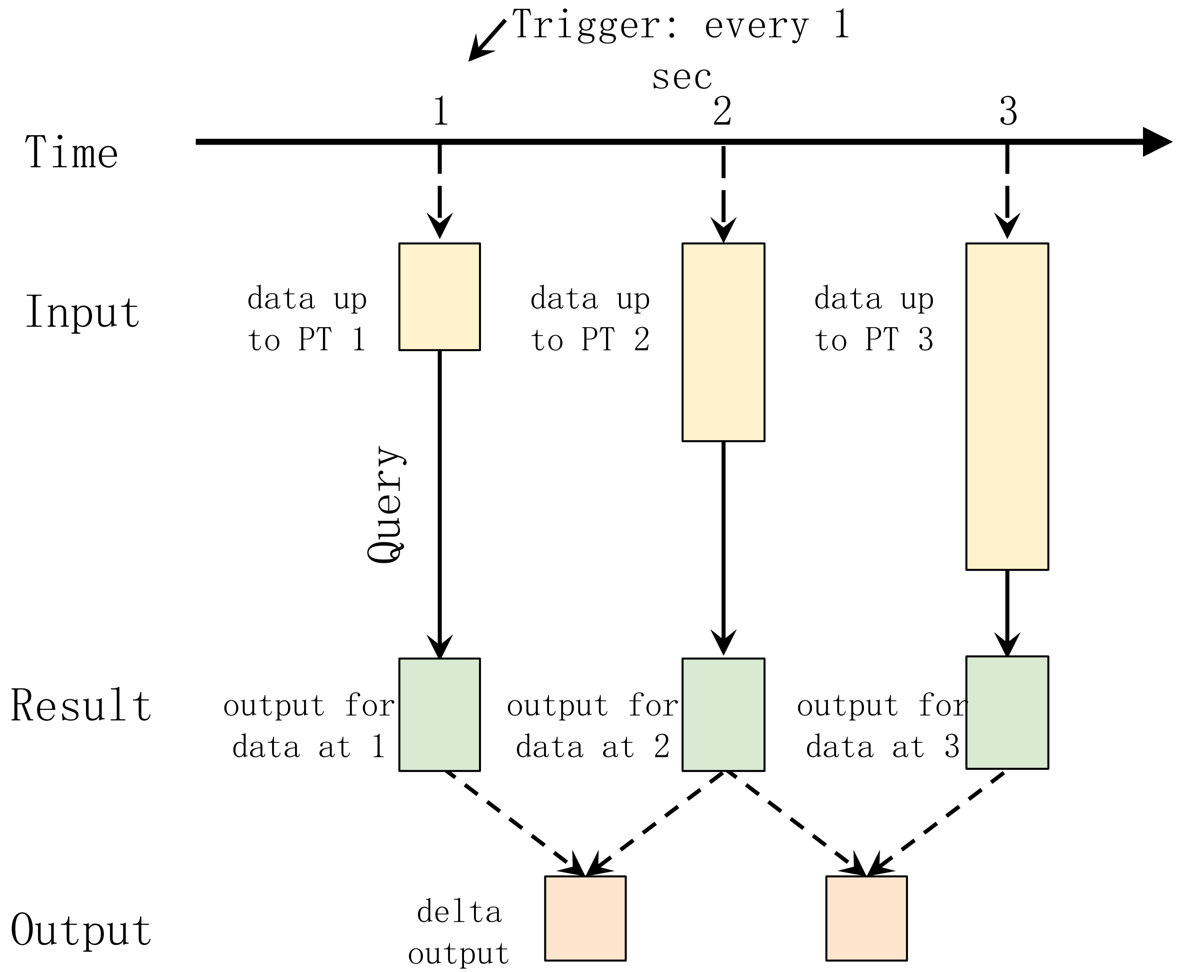
Unifies streaming, interactive and batch queries

- Aggregate data in a stream, then serve using JDBC
- Change queries at runtime
- Build and apply ML models

Model


Trigger: every 1 sec
Time
1
2
3
Input
data up to PT 1
data up to PT 2
data up to PT 3
Query
Result
output for data at 1
output for data at 2
output for data at 3
Output
complete output

Model


Trigger: every 1 sec
1
2
3
Time
Input
data up to PT 1
data up to PT 2
data up to PT 3
Query
Result
output for data at 1
output for data at 2
output for data at 3
Output
delta output

Model Details

Input sources

Append-only tables

Queries

New operators for windowing, sessions, etc.

Triggers

Based on time (e.g. every 1 sec)

Output modes

Complete, deltas, update-in-place

Example: ETL

Input sources

Files in S3

Queries

map (transform each record)

Triggers

"Every 5 sec"

Output modes

"New records" (deltas), into S3 sink

Example: Page View Count

Input sources

Records in Kafka

Queries

```
SELECT COUNT(*)
GROUP BY page, minute(evtime)
```

Triggers

"Every 5 sec"

Output modes

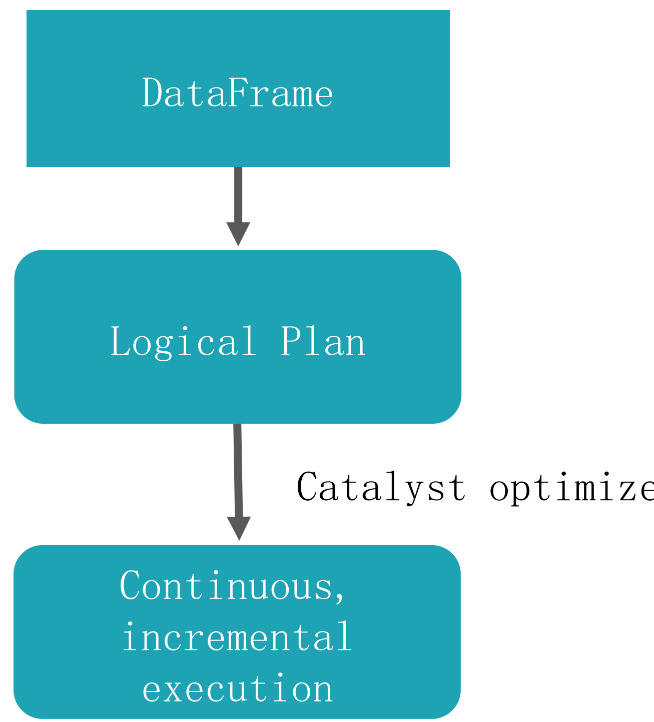
Update-in-place, into MySQL

Note:
This will automatically update "old" records on late data!

Execution

Logically:
DataFrame operations on static data
(i.e. as easy to understand as batch)

Physically:
Spark automatically runs the query in streaming fashion
(i.e. incrementally and continuously)



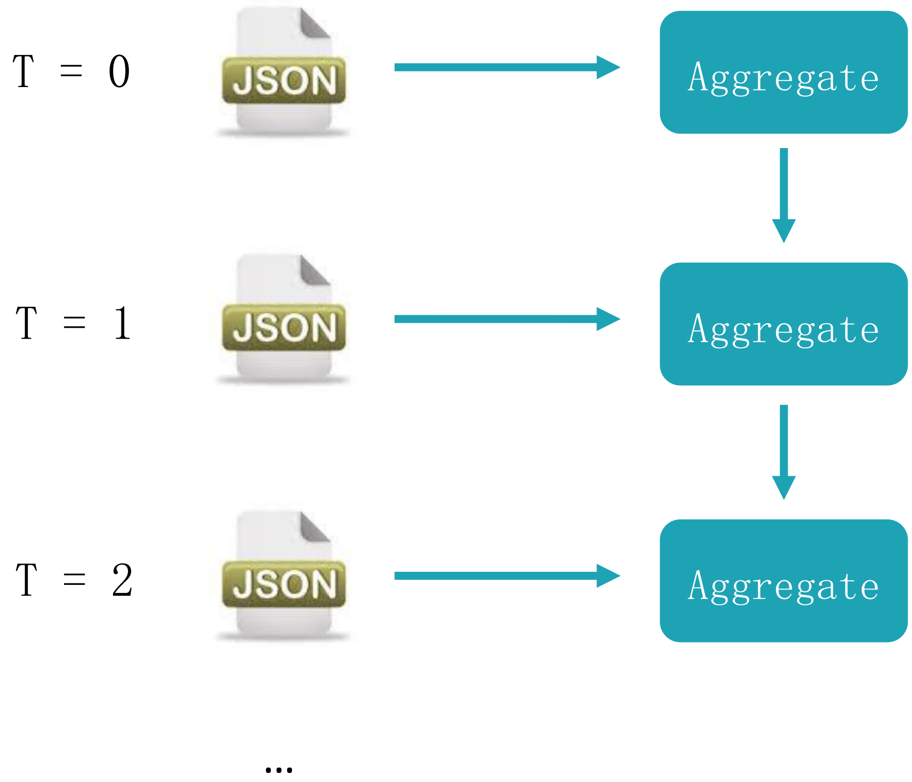
Example: Batch Aggregation

```
logs = ctx.read.format("json").open("s3://logs")

logs.groupBy(logs.user_id).agg(sum(logs.time))
    .write.format("jdbc")
    .save("jdbc:mysql//...")
```

Example: Continuous Aggregation

```
logs = ctx.read.format("json").stream("s3://logs")

logs.groupBy(logs.user_id).agg(sum(logs.time))
    .write.format("jdbc")
    .stream("jdbc:mysql//...")
```

Automatic Incremental Execution



Rest of Spark Will Follow

- Interactive queries should just work
- Spark's data source API will be updated to support seamless streaming integration
  - Exactly once semantics end-to-end
  - Different output modes (complete, delta, update-in-place)
- ML algorithms will be updated too

databricks

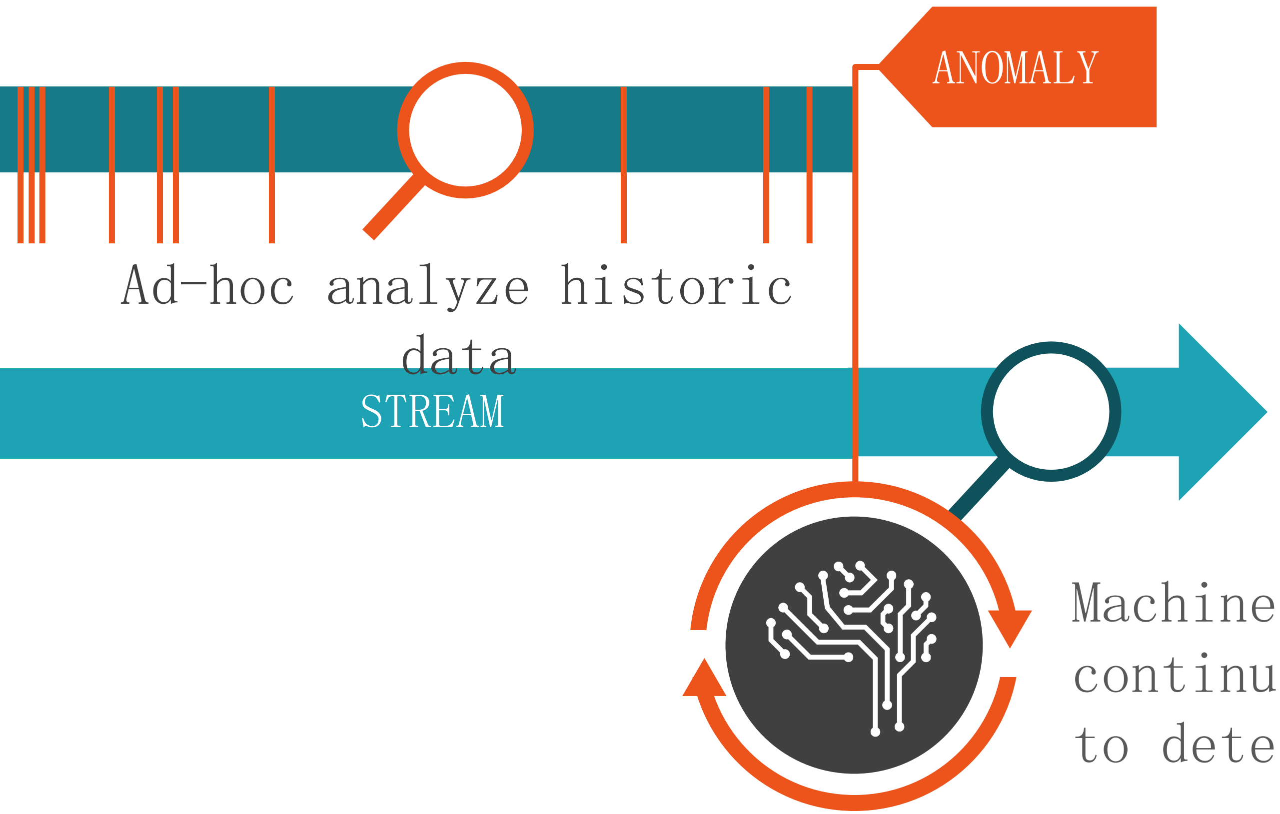
What can we do with this that’s hard with other engines?

Ad-hoc, interactive queries

Dynamic changing queries

Benefits of Spark: elastic scaling, straggler mitigation, etc

Use Case: Fraud Detection



Machine learning model continuously updates to detect new anomalies

Timeline

Spark 2.0

- API foundation
- Kafka, file systems, and databases
- Event-time aggregations

Spark 2.1+

- Continous SQL
- BI application integration
- Other streaming sources/sinks
- Machine learning

Thank you.

databricks™