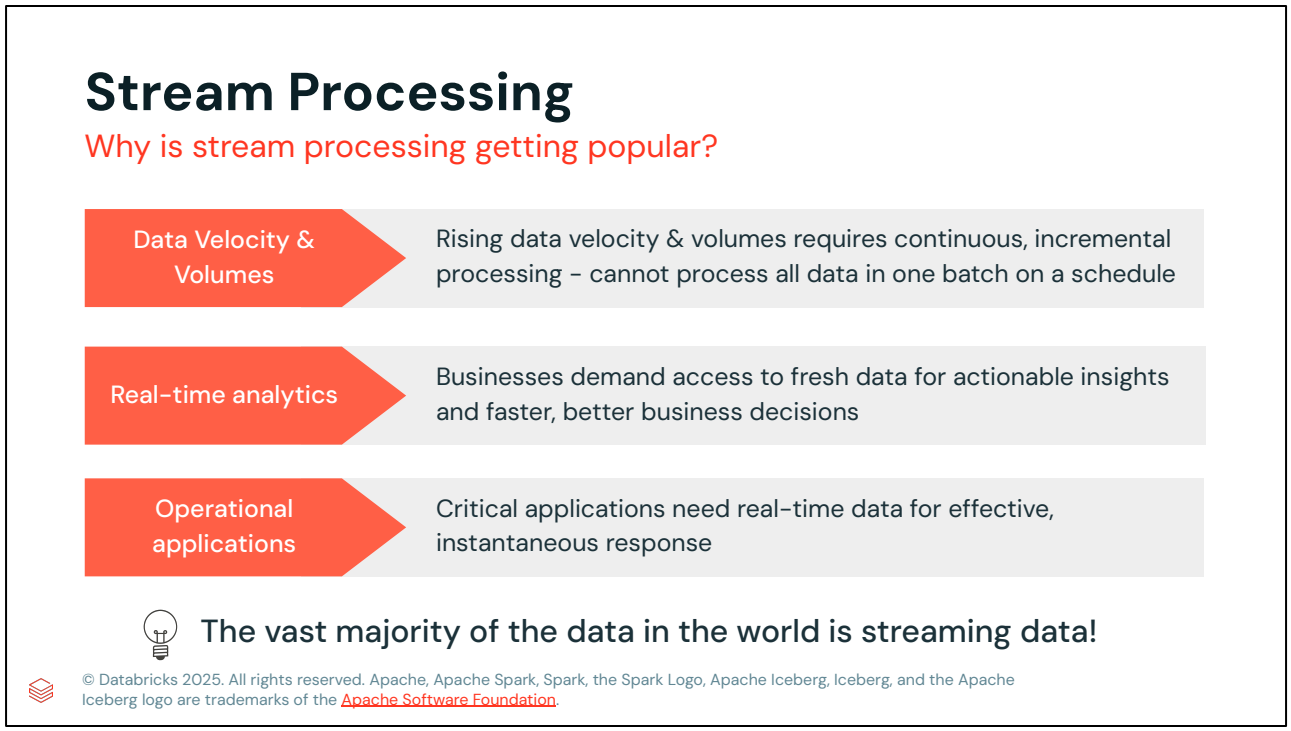
databricks

Introduction to Streaming

LECTURE

Streaming Data Concepts

Streaming Data Concepts explores real-time processing of continuous data streams, highlighting use cases, architectures, benefits, and the unique challenges of stream processing.



So why is streaming data becoming popular?

One reason is because of data velocity. Data velocity and volume require continuous incremental processing, and since it's impossible to process all data in one batch on a schedule, there is a need for streaming to process continuous, real-time data.

Another reason why streaming data is becoming more popular is the increased popularity of real-time analytics.
Businesses are demanding up-to-date data for faster, better business decisions, and fresh data is more actionable.

Finally, streaming data is becoming popular for operational applications. Critical operational applications need real-time data for effective and instantaneous response. An example of this is recommendation software, where we have something like information about a user's behavior on a website, and we want to recommend a product to them in real-time.

Stream Processing Use Cases

Stream processing is a key component of big data applications across all industries

Notifications

Real-time reporting

Incremental ETL

Update data to serve in real-time

Real-time decision making

Online ML

Let's go through some stream processing use cases.

Notifications
Notification alerting is arguably the most obvious streaming use case. Real-time notifications can be useful for things like reporting extreme weather conditions or database system failures.

Real-time reporting
Many organizations use streaming systems to run real-time dashboards that any employee can look at to monitor platform usage, system load volume uptime, and usage of new features as they're rolled out.

Incremental ETL
One of the most common applications to reduce latency is incremental ETL. Spark batch jobs are often used for ETL workloads that turn raw data into a structured format like Parquet to enable efficient queries update data in real-time.

Update data to serve in real-time
Streaming systems are frequently used to compute data that gets served interactively with another application.
An example of this would be a web analytics product that continuously tracks number of visits to each page. A streaming system could then be used to keep those counts up-to-date.

Stream Processing Use Cases

Stream processing is a key component of big data applications across all industries

Notifications

Real-time reporting

Incremental ETL

Update data to serve in real-time

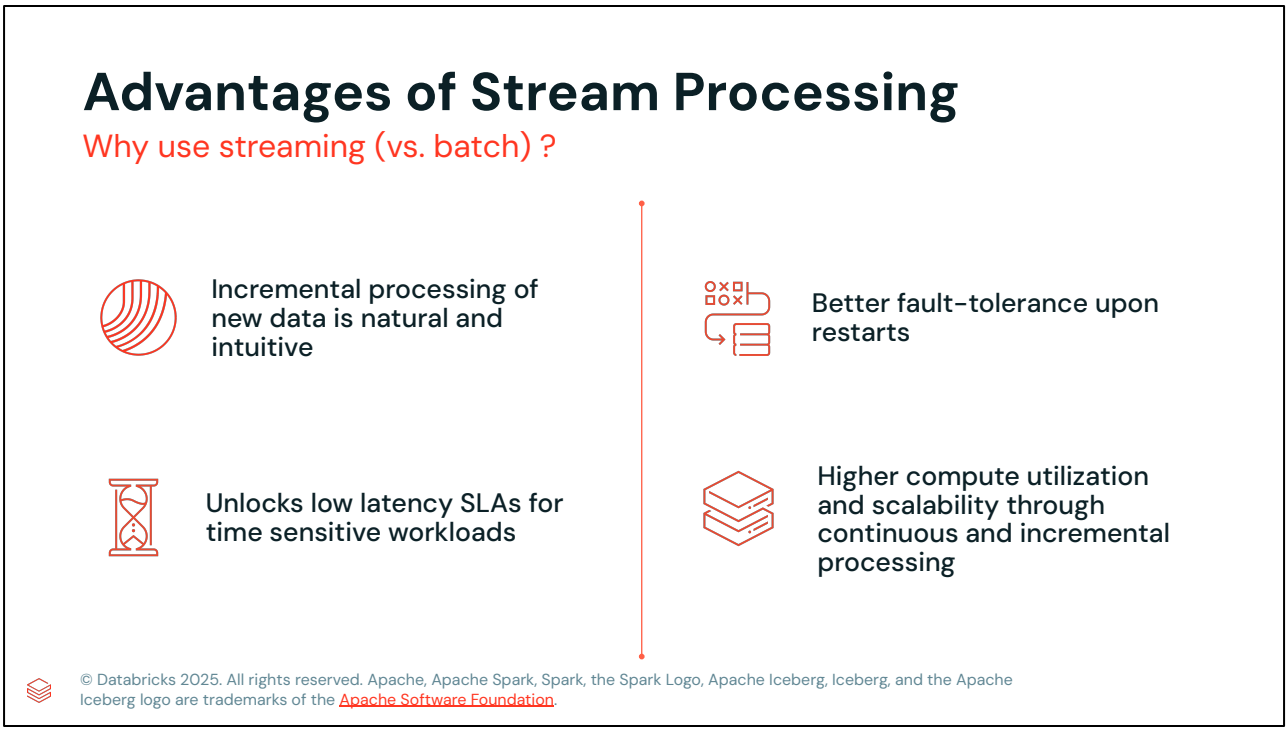
Real-time decision making

Online ML

Real-time decision making
Another stream processing use case is real-time decision making, where a system analyzes new inputs and responds to them automatically using business logic.
An example use case here would be a bank that wants to automatically detect credit card fraud.
In this scenario, it is crucial to notify the customer in real-time if a fraudulent transaction is detected on their card so that the card can be frozen immediately to prevent future fraudulent transactions.

Online ML
Online machine learning is a close derivative of real-time decision making.
In a situation where we are predicting clicks on a website, we can train a model on a combination of streaming and historical data from multiple users and then use that data to predict how users will interact with the website in the future.
Another example of online machine learning is an algorithm for a social media platform like TikTok.
As users scroll through TikTok, the algorithm captures data about which videos the user likes and shares, and uses this information to choose the next video the user sees.

Question: What other use cases for streaming data can you think of?



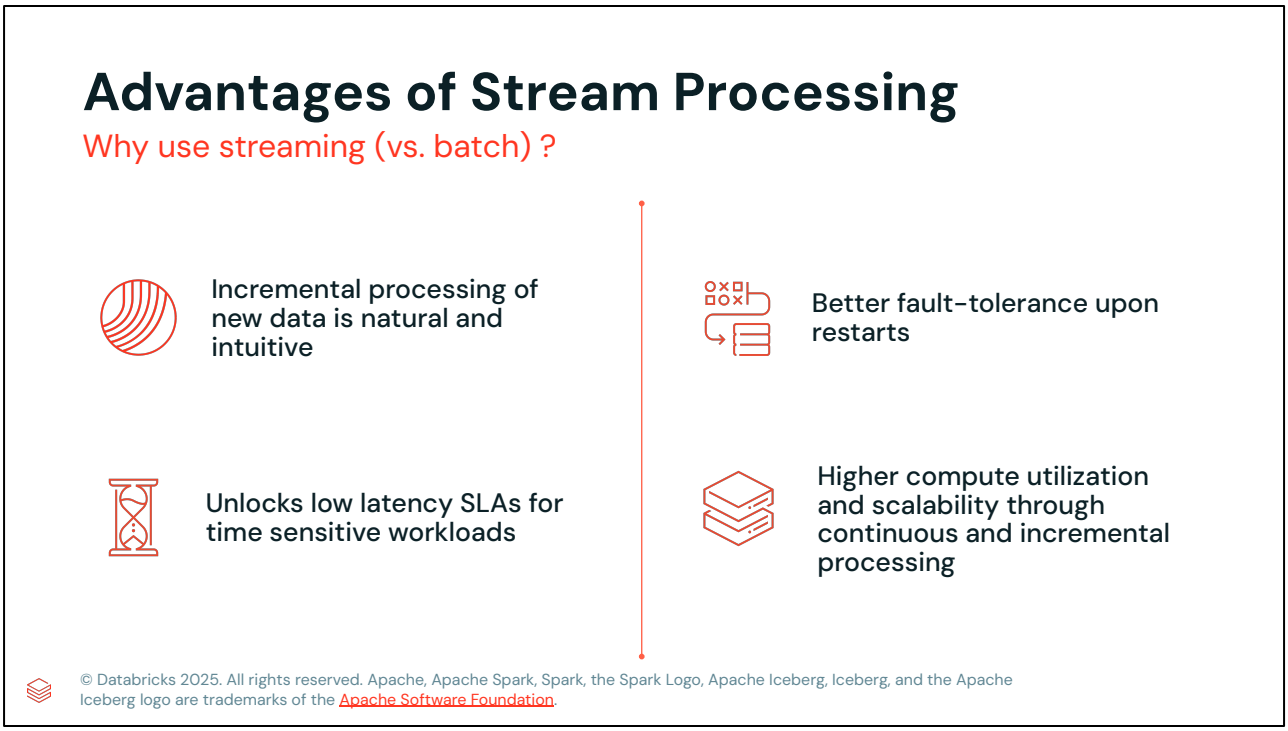
Let's discuss the advantages of stream processing and when to use it over batch processing.

A more intuitive way of capturing and processing continuous and unbounded data

- As humans, we are naturally good at incremental processing. Why? Incremental processing means smaller data sizes and faster run times. It is a more intuitive way of capturing and processing continuous and unbounded data because the data is being captured in real-time.

Unlocks lower latency for time sensitive applications and use cases

- Streaming processing is essential in two cases.
    - The first case is when a lower latency is required. When an application needs to respond quickly on a time scale of minutes, seconds, or even milliseconds, you would need a streaming system that you can keep state in the memory to get acceptable performance. Some examples of this are decision-making and alerting use cases.



Unlocks lower latency for time sensitive applications and use cases (Cont..)

- And then second stream processing can also be more efficient in updating a result than repeated batch jobs because it automatically incrementalizes this computation. So for example, if we want to compute web traffic statistics over the last 24 hours, a naively implemented batch job might scan all the data each time it runs, always processing 24 hours worth of data.
- On the other hand, streaming systems will only remember state from the previous computation and only count new data if you tell the streaming system to update your report every hour. For example, it would only need to process an hour's worth of data each time. So that might be, for example, the new data since the last report in a batch system, though, you would have to implement this kind of incremental computation by hand to get the same performance that results in extra work that the streaming system would have automatically given you out of the box.

Better fault-tolerance through checkpointing

- Another advantage of streaming processing is that it uses checkpointing to keep track of processed and unprocessed data, which enables fault tolerance upon failures.

Advantages of Stream Processing

Why use streaming (vs. batch) ?

Incremental processing of new data is natural and intuitive

Better fault-tolerance upon restarts

Unlocks low latency SLAs for time sensitive workloads

Higher compute utilization and scalability through continuous and incremental processing

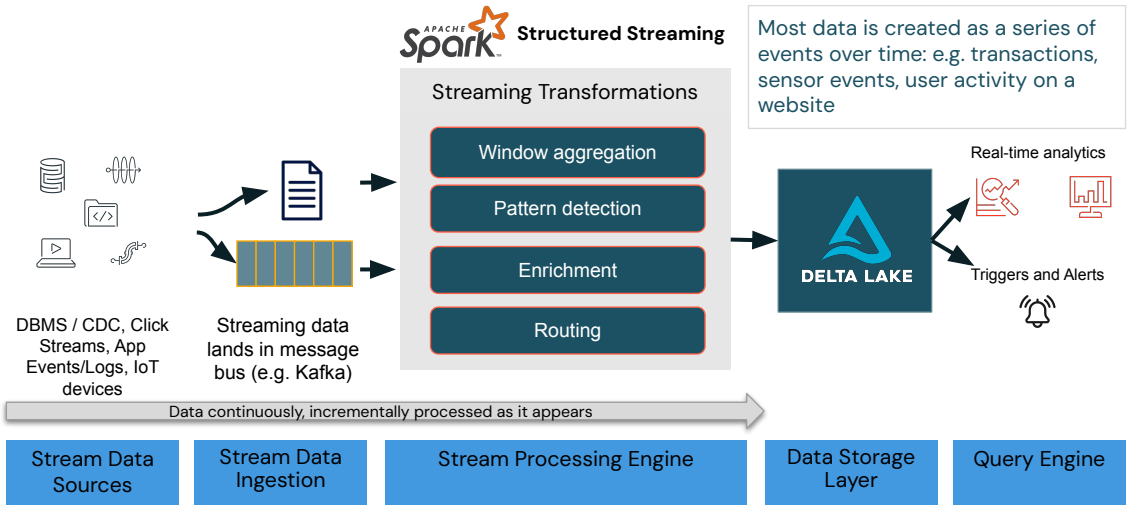
Automatic bookkeeping on new data

Higher compute utilization and scalability through continuous and incremental processing

- Streaming processing is continuous, which leads to constant and higher compute utilization. It also spreads the large chunk of data throughout the day, which leads to better scalability. So instead of running a job at a time when no one's online, we can run it all throughout the day, resulting in lower compute costs.

Stream Processing Architecture

Modern data processing pipeline



Modern data processing pipelines typically consist of a source such as a change data capture (CDC) feed, click streams, app events, logs, or IoT devices, and some sort of way to take that data and ingest it into our system.
Typically, this is a database like Kafka or Kinesis.
Then we want to take our data as it's being ingested and process it with some sort of stream processing engine.

We can also do things like window aggregation, pattern detection, and enrichment before sinking the data.
Delta Lake is a great storage layer for this. It offers a lot of benefits and comes native out of the box with Databricks.
Finally, we want to have some sort of query engine on top of that for building real-time analytics like dashboards or trigger alerts.

Challenges of Stream Processing

Stream processing is not easy

- Processing each event exactly once despite machine failures
- Processing out-of-order data based on application timestamps (also called event time)
- Maintaining large amounts of state
- Handling load imbalance and stragglers
- Determining how to update output sinks as new events arrive
- Writing data transactionally to output systems

However, there are some challenges that come with stream processing.

Processing out-of-order data based on application timestamps, also called event time, is difficult because we have to factor in the state of where data is arriving.

Also, maintaining large amounts of state is also not a trivial task.

Another challenge of stream processing is that data is processed exactly once, even if there are several machine failures, potentially leaving the system in an unusable state for extended amounts of time.

Handling load imbalance and stragglers is difficult because we have to consider what engine to use to process this data.

Similarly, determining how to update output sinks as new events arrive is challenging because we need to consider where all of the data should land after it has been processed.

Finally, writing data transactionally to output systems is also non-trivial.

Thank you for completing this lesson and continuing your journey to develop your skills with us.