Batch processing Vs Stream processing

Features	Batch Processing	Stream Processing
Data Scope	Queries or process overall or most of the data in the dataset	Queries or process over data within a rolling time window or on just the most recent data record.
Data size	Large batches of data	Individual records or microbatches
Performance	Latencies in minutes to hours	Latencies in seconds or milliseconds
Libraries or Framework	SparkCore, MapReduce	Spark Streaming, Storm

Need for Spark Streaming

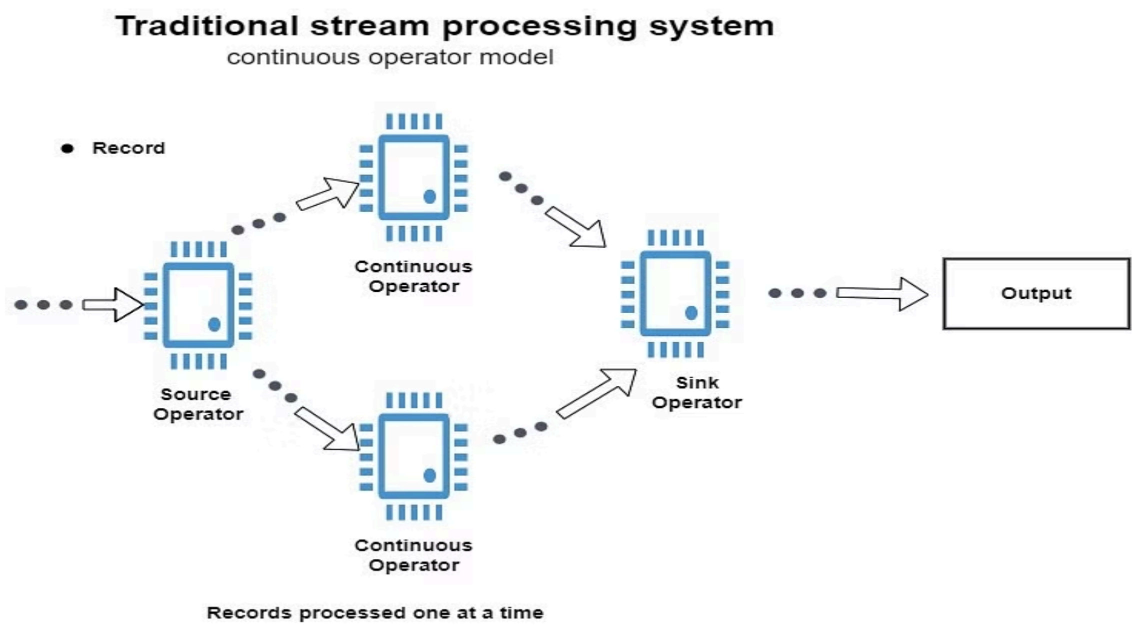
Companies are gathering increasing amounts of data, and they want to get value from the data in real-time. Online transactions, social networks, and sensors generate data that must be monitored and acted upon continuously. Consequently, real-time stream processing is now more significant than ever.
For instance, online purchases require all the associated data (Eg. date, time, items, price) to be stored and ready for organizations to analyze and make prompt decisions based on customer behavior. Another application requiring pre-trained fraud models is detecting fraudulent bank transactions (through data streams), and it can dramatically reduce the likelihood of fraud occurring.
Many traditional stream processing systems use a continuous operator model to process data. This model works as follows:

- Ingest streaming data from a range of data sources (Eg. live logs, IoT device data, system telemetry data, etc.) into some data ingestion systems, such as Apache Kafka, Amazon Kinesis, etc.
- Stream processing engines are designed to process data in parallel on a cluster.
- The results are output to the downstream systems like Cassandra, HBase, Kafka, etc.

Workers run one or more continuous operators on each node. Continuous operators process one record and pass the records along with other operators in the pipeline. The "source" operator receives data, while the "sink" operators perform the output to downstream systems. Continuous operators are natural and easy-to-use models.

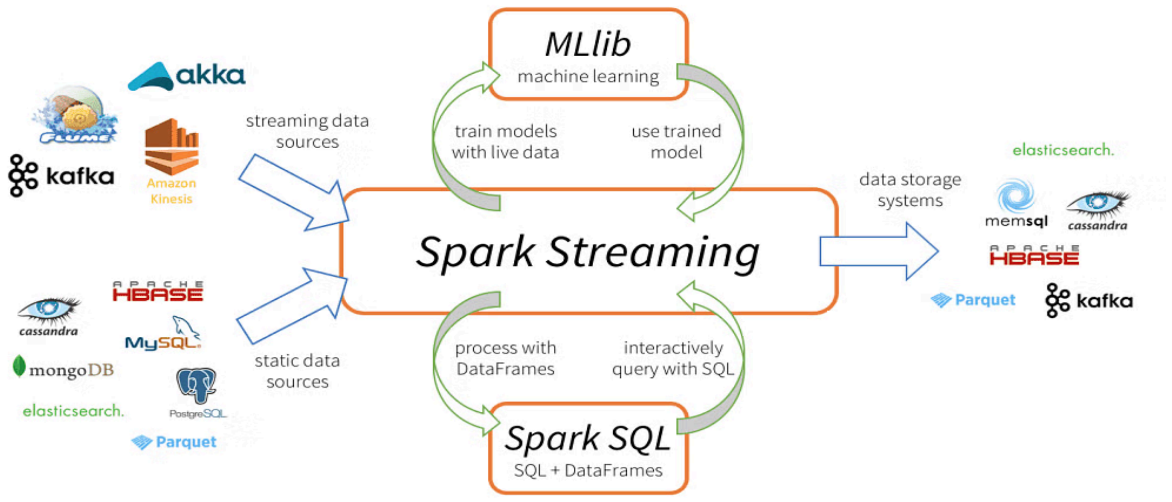
However, given the need for complex real-time analytics, these traditional architectures have some challenges:-

- Fast Failure and Straggler Recovery
- Load Balancing
- Unification of Streaming, Batch, and Interactive Workloads
- Advanced Analytics with SQL Queries and Machine learning


Traditional stream processing system
continuous operator model
● Record
Source Operator
Continuous Operator
Continuous Operator
Sink Operator
Output
Records processed one at a time

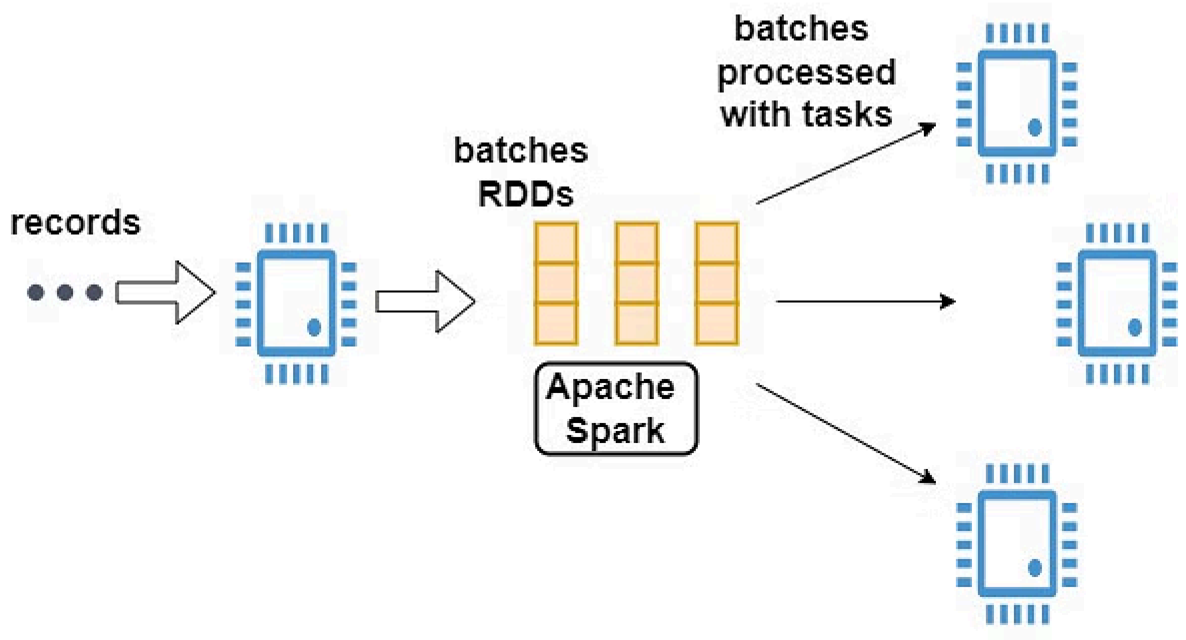
Spark Streaming

Spark Streaming is a scalable fault-tolerant streaming processing system that natively supports both batch and streaming workloads. Spark Streaming is an extension of the core Spark API that allows data engineers and data scientists to process real-time data from various sources including (but not limited to) Kafka, Flume, and Amazon Kinesis. This processed data can be pushed out to file systems, databases, and live dashboards. Its key abstraction is a Discretized Stream or, in short, a DStream, which represents a stream of data divided into small batches. DStreams are built on RDDs, Spark's core data abstraction. This allows Spark Streaming to seamlessly integrate with any other Spark components like MLlib and Spark SQL. Spark Streaming is different from other systems that either have a processing engine designed only for streaming, or have similar batch and streaming APIs but compile internally to different engines. Spark's single execution engine and unified programming model for batch and streaming lead to some unique benefits over other traditional streaming systems.



Spark Streaming Workflow

Spark Streaming discretizes streaming data into tiny, sub-second micro-batches instead of treating it as a single record at a time. The Receivers of Spark Streaming accept data concurrently and buffer it in the memory of Spark workers. In the next step, the latency-optimized Spark engine processes the batches in a matter of milliseconds and outputs the results to other systems. Unlike the traditional continuous operator model, the Spark tasks are dynamically assigned to the workers based on the data location and available resources. Ensure better load distribution and faster fault recovery using this technique.



Records processed in batches with short tasks
Each batch is an RDD (partitioned dataset)