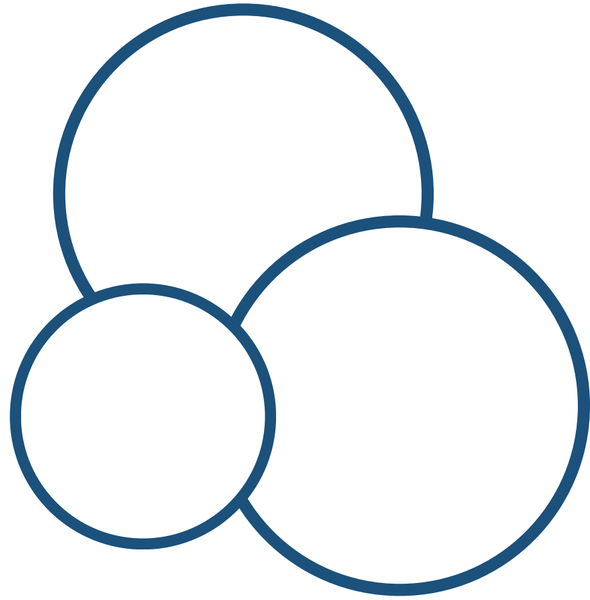
DATATECH LABS.

Data Engineering Course.

Week 4: Advanced Data Processing with Spark and Streaming Pipelines

Outline: Week 4

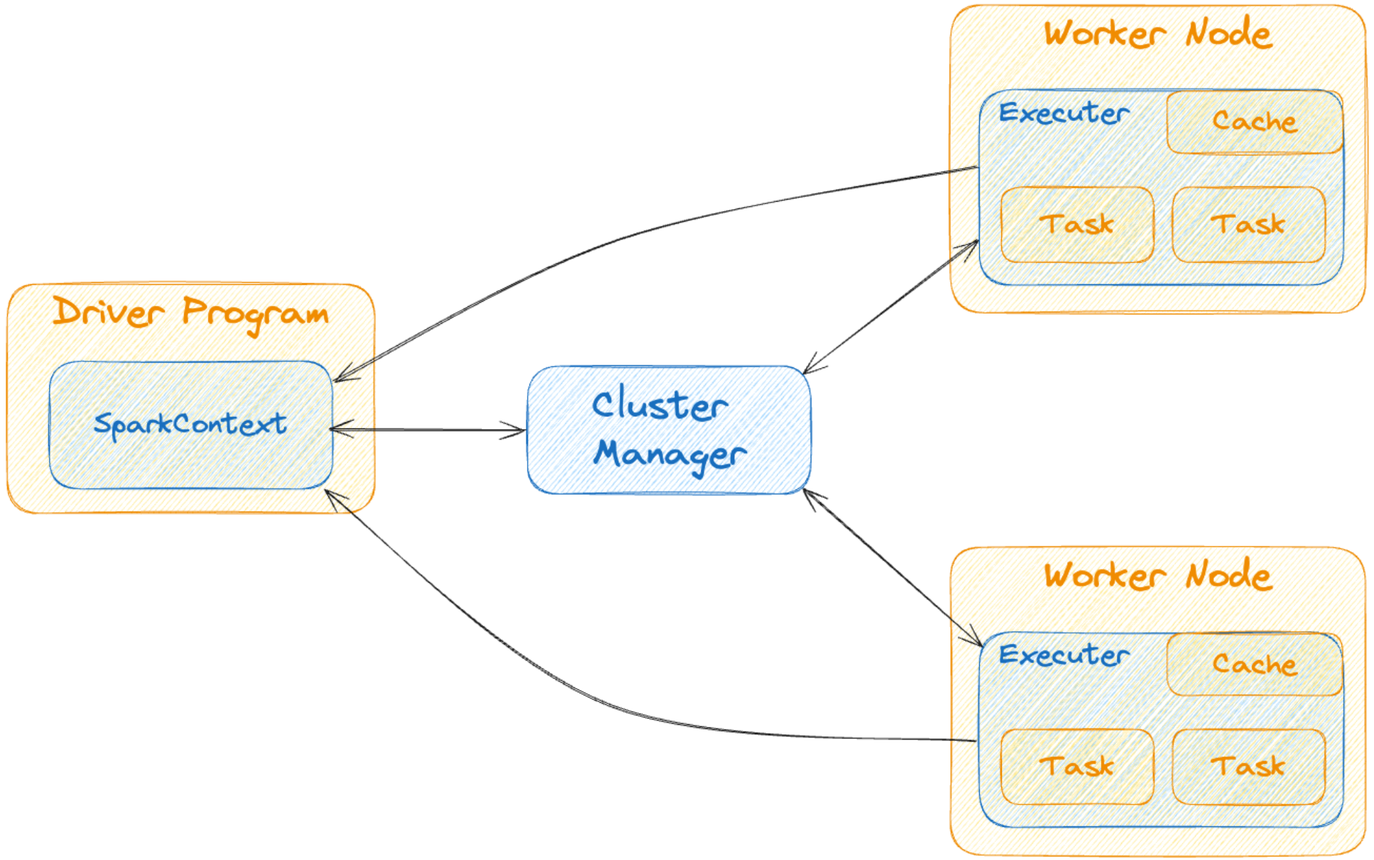
- Advanced Spark Programming
  - Spark RDDs, DataFrames, and Datasets
  - Spark SQL and analytics
  - Machine learning with Spark MLlib
- Streaming Data Processing
  - Introduction to stream processing concepts
  - Apache Kafka for real-time data ingestion
  - Stream processing with Apache Flink



Advanced Spark Programming

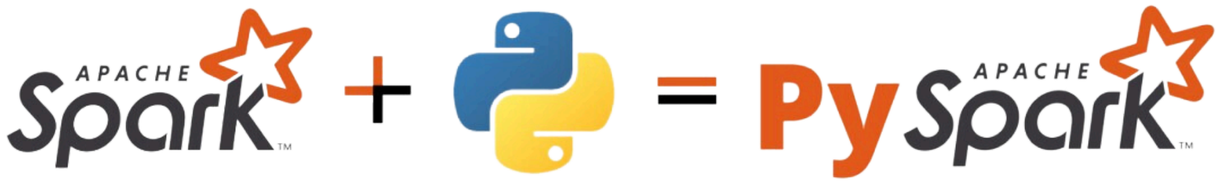
Spark Architecture (Recap)

- Driver Program
- Cluster Manager
- Worker Nodes
- Executors



Spark Architecture (Recap)

- Creating Spark Session.



Apache Spark Python PySpark

```python
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("Advanced Spark Operations") \
    .getOrCreate()

df = spark.createDataFrame(data, ["name", "age", "city"])
```

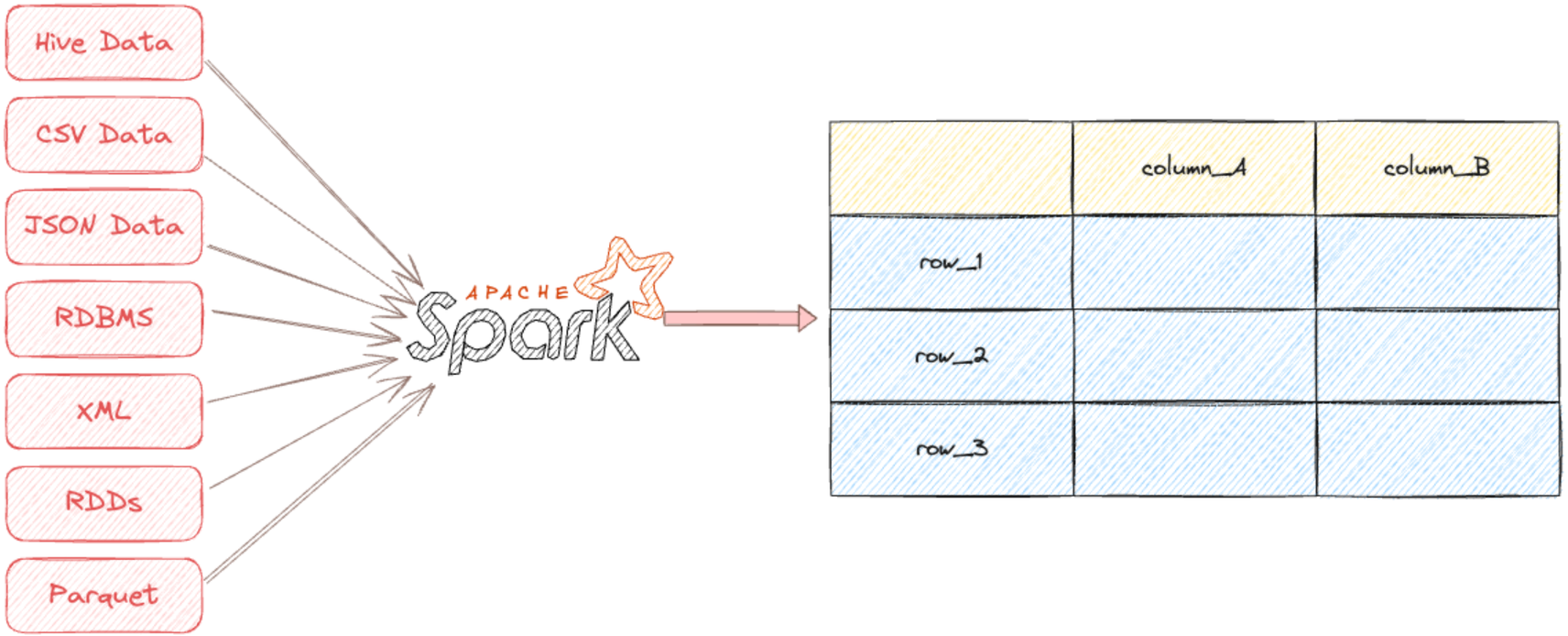
Spark DataFrames

- **Definition**: Distributed collection of data organized into named columns
- **Advantages** of DataFrames over RDDs:
  - Named columns
  - Optimized execution
  - SQL-like operations

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

# Create a SparkSession
spark = SparkSession.builder \
    .appName("DataFrame Operations Example") \
    .getOrCreate()

# Sample data for the DataFrame
data = [
    (1, "Alice", 28),
    (2, "Bob", 22),
    (3, "Charlie", 35)
]

# Define the DataFrame schema
columns = ["ID", "Name", "Age"]

# Create the DataFrame
df = spark.createDataFrame(data, columns)
```

Creating DataFrames

- From RDDs
- From a list of data
- Reading from files (CSV, JSON, Parquet)
- From Hive tables



Spark DataFrame Operations

- select(), filter(), groupBy()
- join(), union(), orderBy()
- withColumn(), drop()
- agg() for aggregations

```python
# Create a DataFrame
data = [("Alice", 25, "New York"),
        ("Bob", 30, "San Francisco"),
        ("Charlie", 35, "London")]

df = spark.createDataFrame(data, ["name", "age", "city"])

# DataFrame operations
df.show()
df.select("name", "age").show()
df.filter(df.age > 28).show()
df.groupBy("city").agg(avg("age") \
  .alias("avg_age")).show()
```

Spark SQL

- Running SQL queries on DataFrames
- Creating temporary views
- Catalog API for managing tables and databases

```
df.createOrReplaceTempView("people")

result = spark.sql(
    "SELECT * FROM people WHERE age > 28 ORDER BY age"
)

result.show()
```

Spark Datasets

- **Definition**: Strongly-typed version of DataFrames
- Benefits:
  - Type-safety
  - Lambda functions in queries

```python
from pyspark.sql.types import StructType, \
    StructField, StringType, IntegerType

schema = StructType([
    StructField("name", StringType(), True),
    StructField("age", IntegerType(), True),
    StructField("city", StringType(), True)
])

dataset = spark.createDataFrame(data, schema)
dataset.printSchema()
```

Working with Datasets

- Creating Datasets
- Dataset operations
- Converting between DataFrames and Datasets

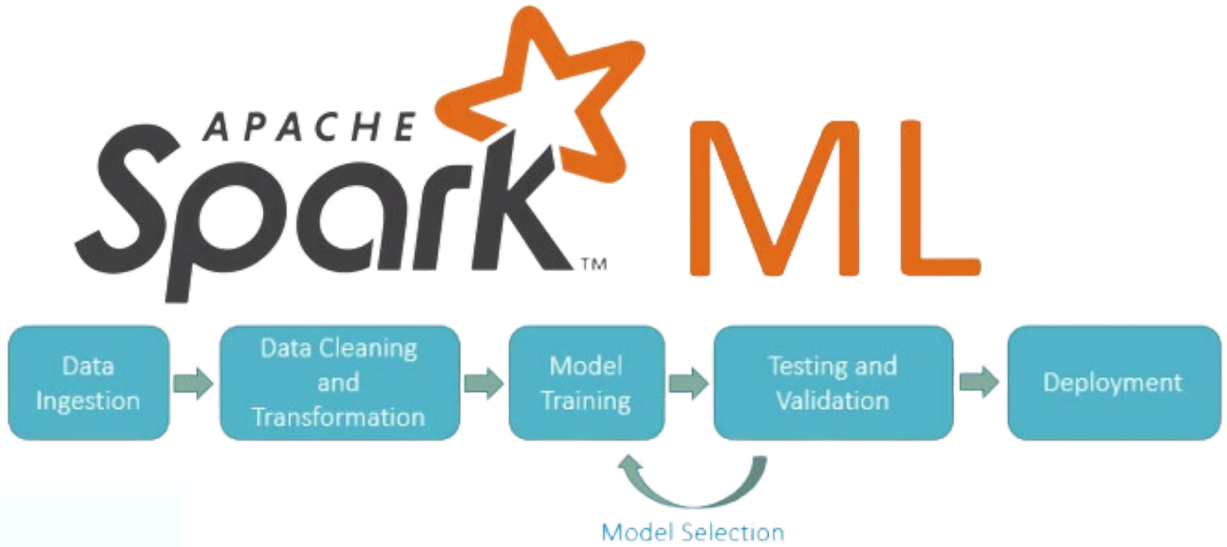
```
ds.take(10).foreach(println(_))
```

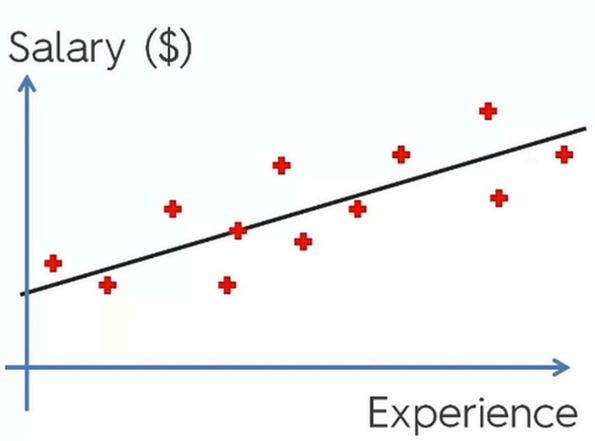
▸ (1) Spark Jobs

```
DeviceIoTData(8,868,US,USA,United States,1,meter-gauge-1xbYRYcj,51,68.161.225.1,38.0,green,-97.0,Celsius,34,1458444054093)
DeviceIoTData(7,1473,NO,NOR,Norway,2,sensor-pad-2n2Pea,70,213.161.254.1,62.47,red,6.15,Celsius,11,1458444054119)
DeviceIoTData(2,1556,IT,ITA,Italy,3,device-mac-36TWSKiT,44,88.36.5.1,42.83,red,12.83,Celsius,19,1458444054120)
DeviceIoTData(6,1080,US,USA,United States,4,sensor-pad-4mzWkz,32,66.39.173.154,44.06,yellow,-121.32,Celsius,28,1458444054121)
DeviceIoTData(4,931,PH,PHL,Philippines,5,therm-stick-5gimpUrBB,62,203.82.41.9,14.58,green,120.97,Celsius,25,1458444054122)
DeviceIoTData(3,1210,US,USA,United States,6,sensor-pad-6al7RTAobR,51,204.116.105.67,35.93,yellow,-85.46,Celsius,27,1458444054122)
DeviceIoTData(3,1129,CN,CHN,China,7,meter-gauge-7GeDoanM,26,220.173.179.1,22.82,yellow,108.32,Celsius,18,1458444054123)
DeviceIoTData(0,1536,JP,JPN,Japan,8,sensor-pad-8xUD6pzsQI,35,210.173.177.1,35.69,red,139.69,Celsius,27,1458444054123)
DeviceIoTData(3,807,JP,JPN,Japan,9,device-mac-9GcjZ2pw,85,118.23.68.227,35.69,green,139.69,Celsius,13,1458444054124)
DeviceIoTData(7,1470,US,USA,United States,10,sensor-pad-10BsywSYUF,56,208.109.163.218,33.61,red,-111.89,Celsius,26,1458444054125)
```

Machine Learning with Spark MLlib

- MLlib: is Spark's machine learning (ML) library. Its goal is to make practical machine learning scalable and easy
- Supported algorithms:
  - Classification
  - Regression
  - Clustering
  - Collaborative filtering



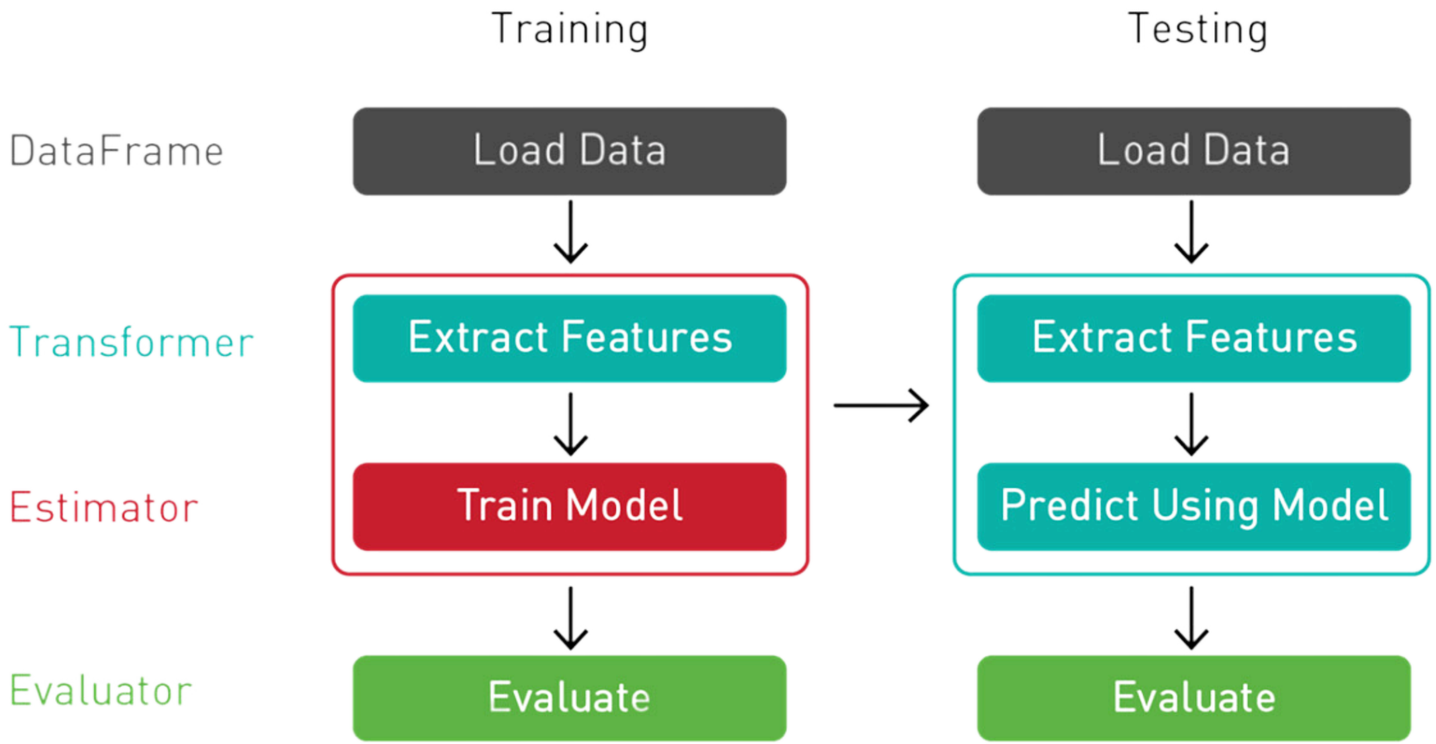


$$y = b_0 + b_1 * x$$

$$\text{Salary} = b_0 + b_1 * \text{Experience}$$

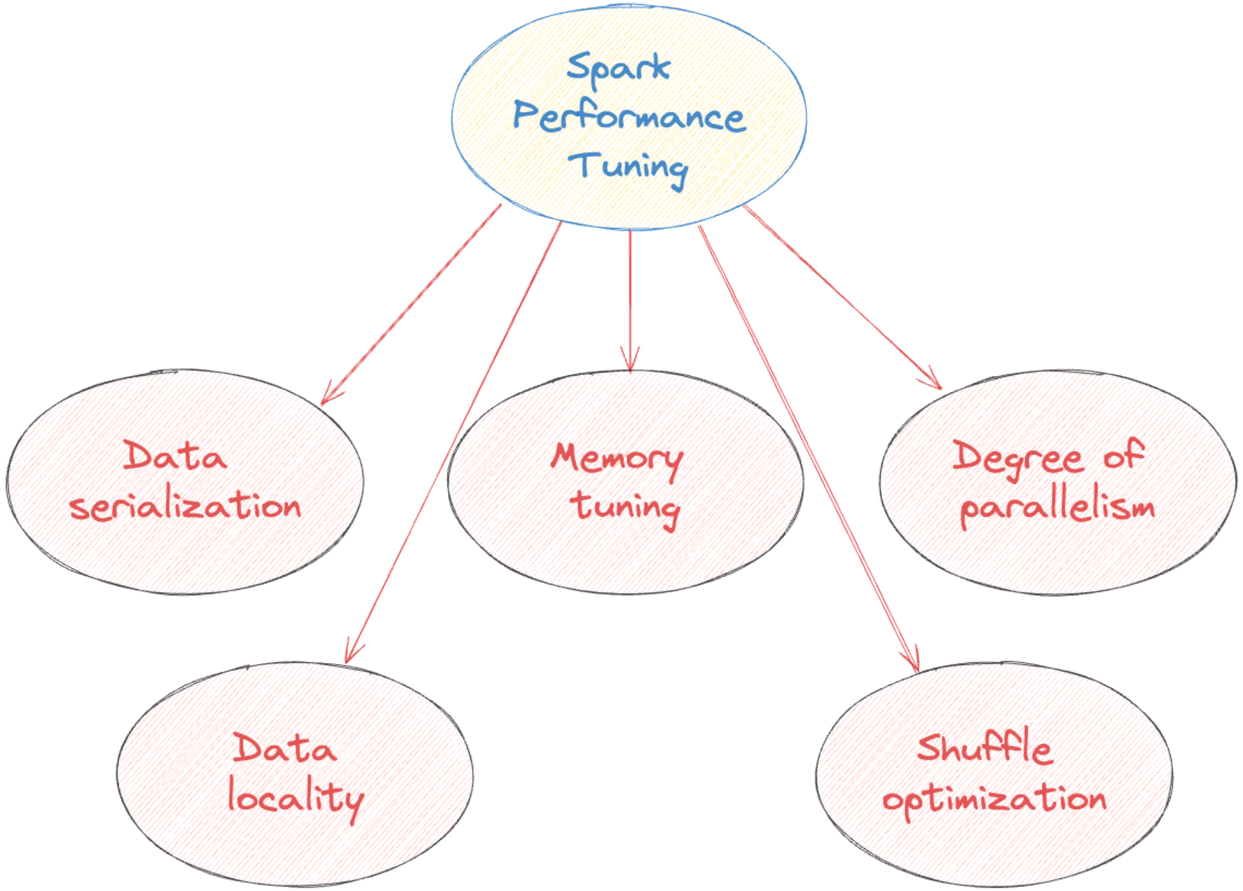
MLlib Pipeline APIs

- Transformers
- Estimators
- Parameters
- Pipeline composition



Spark Performance Tuning

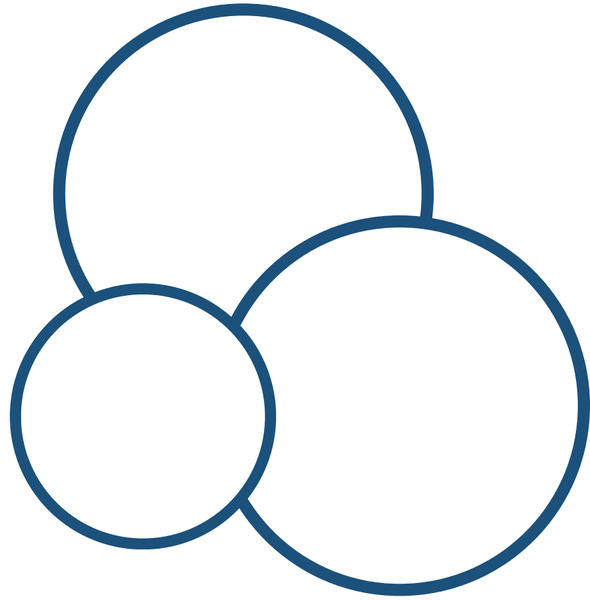
- Data serialization
- Memory tuning
- Degree of parallelism
- Data locality
- Shuffle optimization



Hands-on:

Deep Dive in Apache Spark/PySpark

Q&A and Discussion

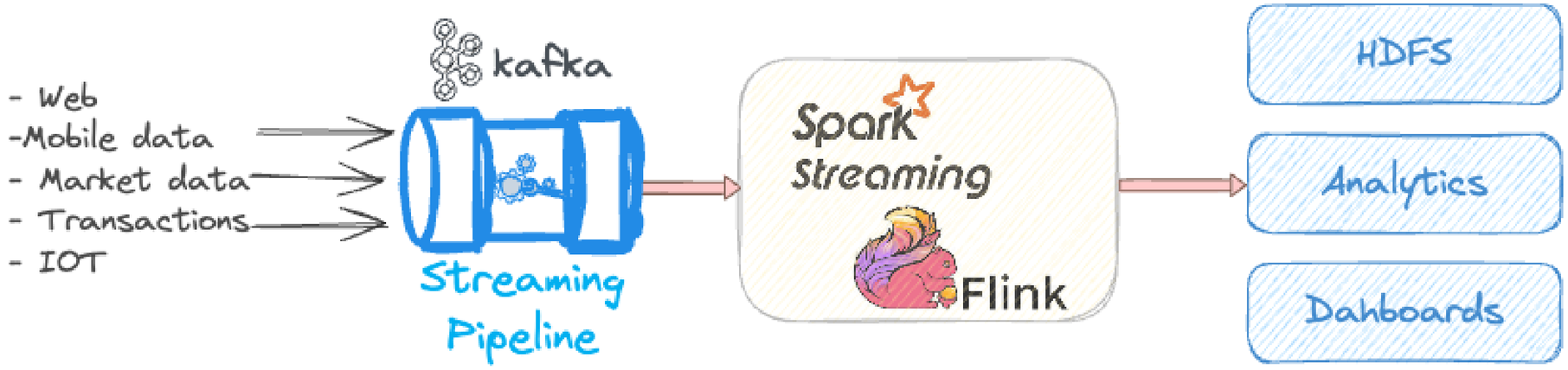


Streaming Data Processing

Streaming Data Processing

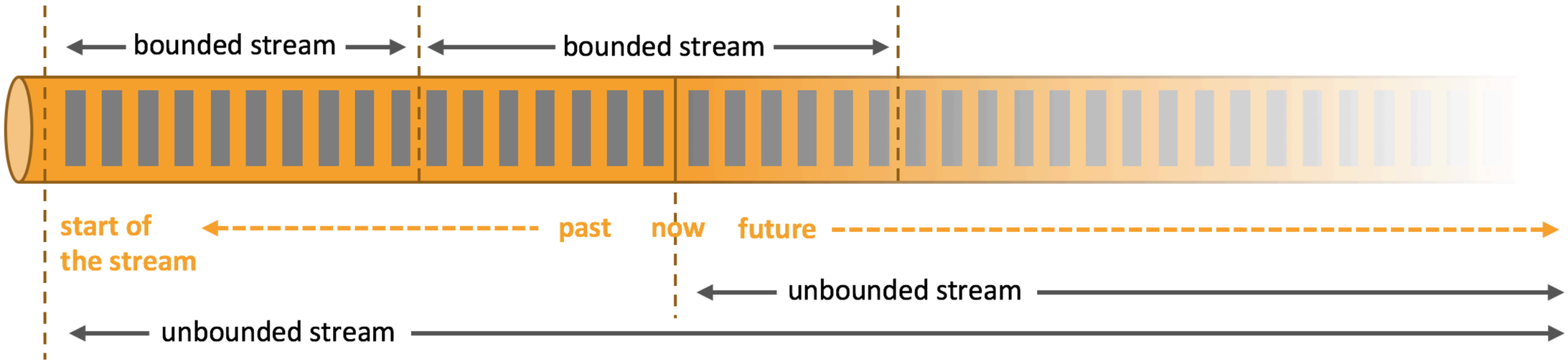
Introduction to Streaming Data Processing

- **Definition**: Processing data in real-time as it arrives. It is the continuous flow of data as it's generated, enabling real-time processing and analysis for immediate insights.
- **Importance** in modern data architectures:
  - Location data
  - Fraud detection
  - Real-time stock trades
  - Marketing, sales, and business analytics...

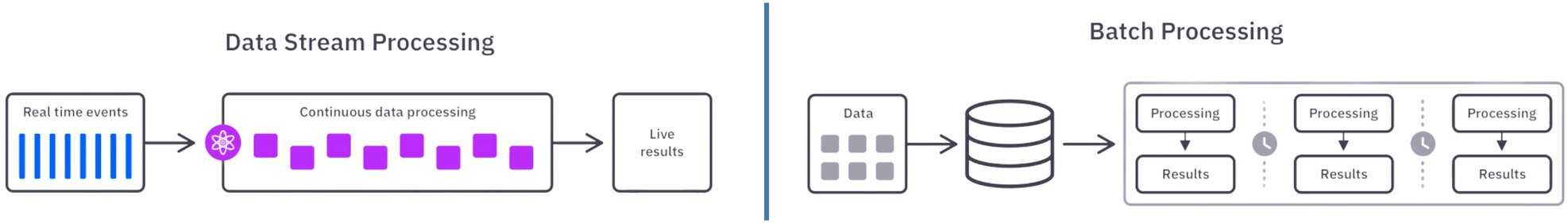


Characteristics of Streaming Data

- Continuous
- Unbounded
- Time-sensitive
- High volume and velocity



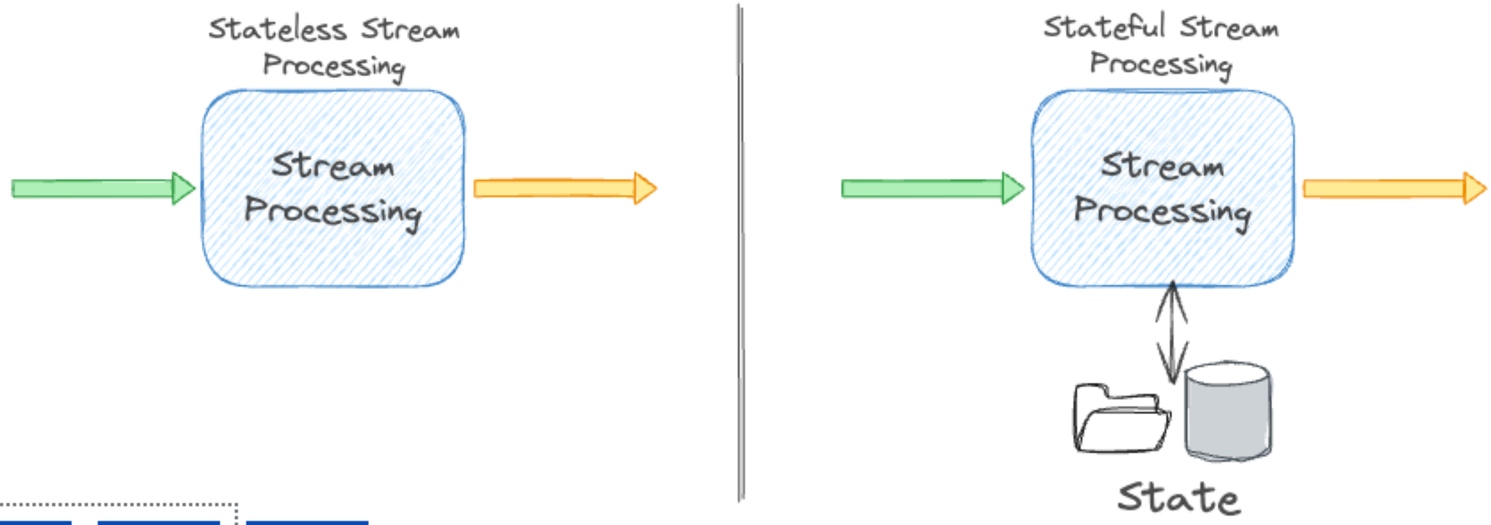
Streaming vs. Batch Processing

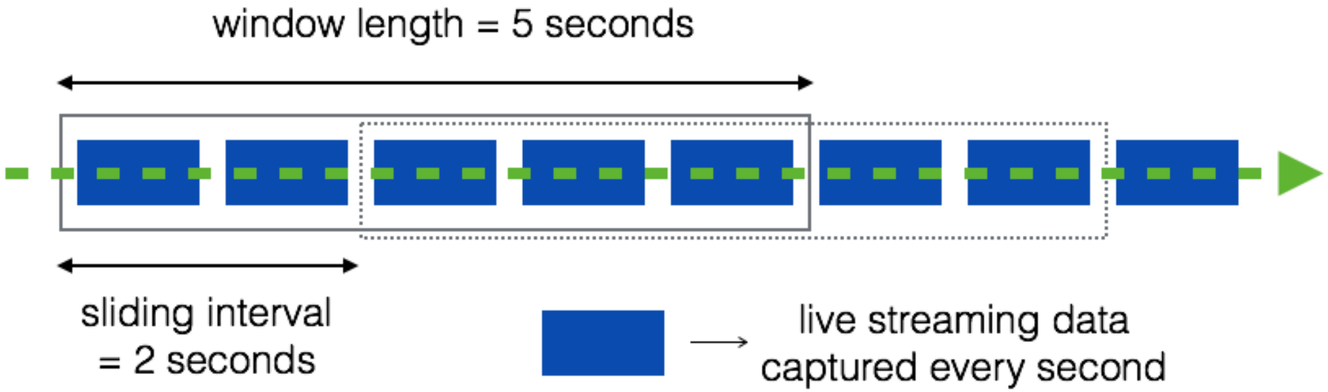


Aspect	Streaming	Batch
Data processing	Real-time	Periodic
Latency	Low	High
Throughput	Lower	Higher
Complexity	Higher	Lower
Use cases	Real-time analytics, monitoring	Historical analysis, reporting

Streaming Data Processing Concepts

- Event time vs. Processing time
- Windowing: it allows data to be processed in small, manageable chunks over a specified period
- State management

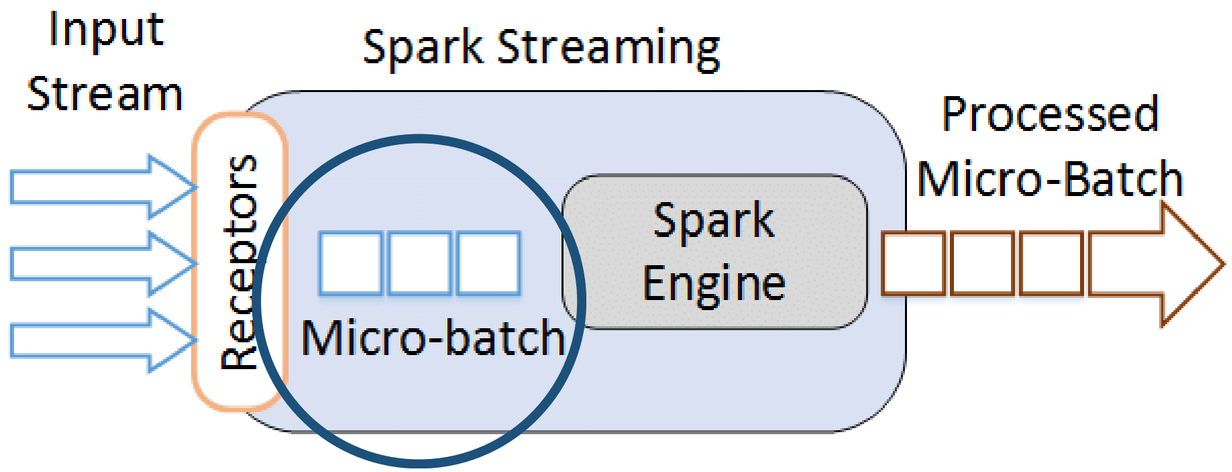




Streaming Processing Models & Data Sources

- Processing Models
  a. Record-at-a-time
  b. Micro-batch
  c. Hybrid

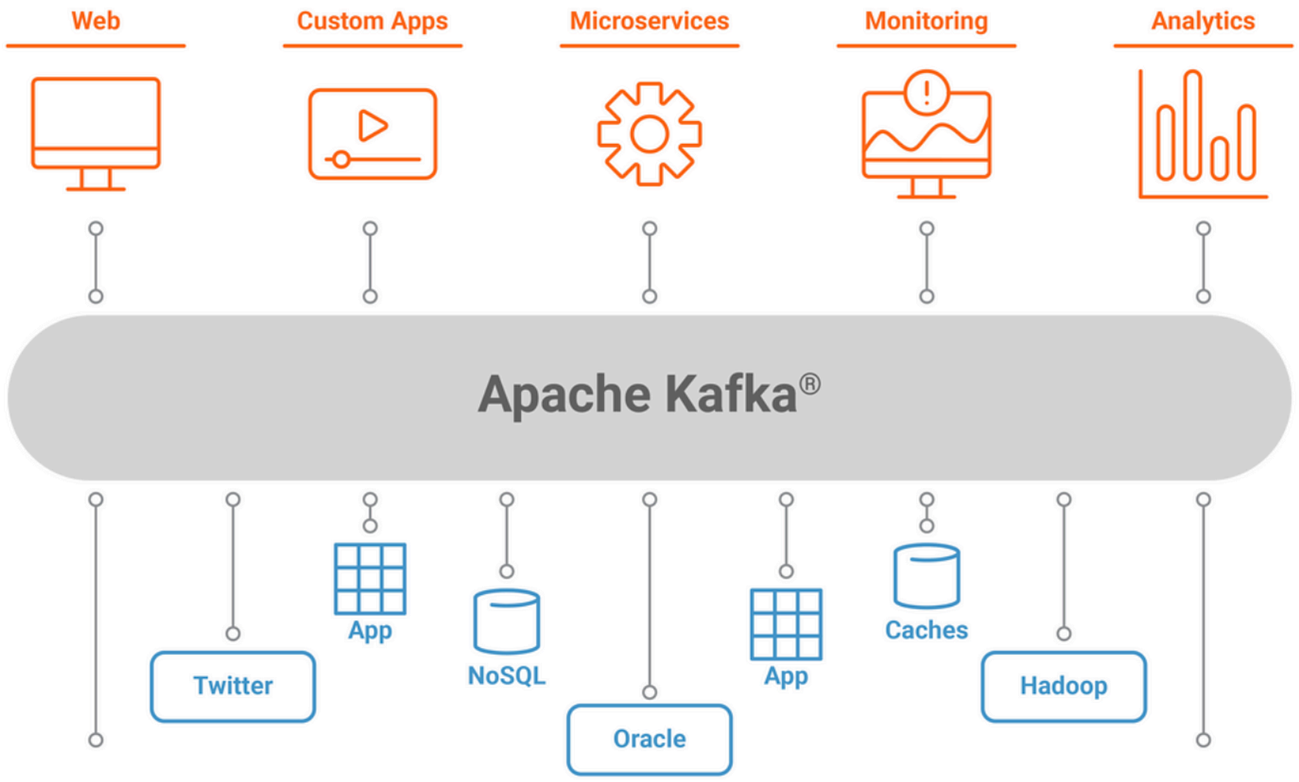
- Data Sources
  - IoT devices
  - Social media feeds
  - Financial market data
  - Web clicks and user activity
  - Logs and metrics



Data Streaming Tools

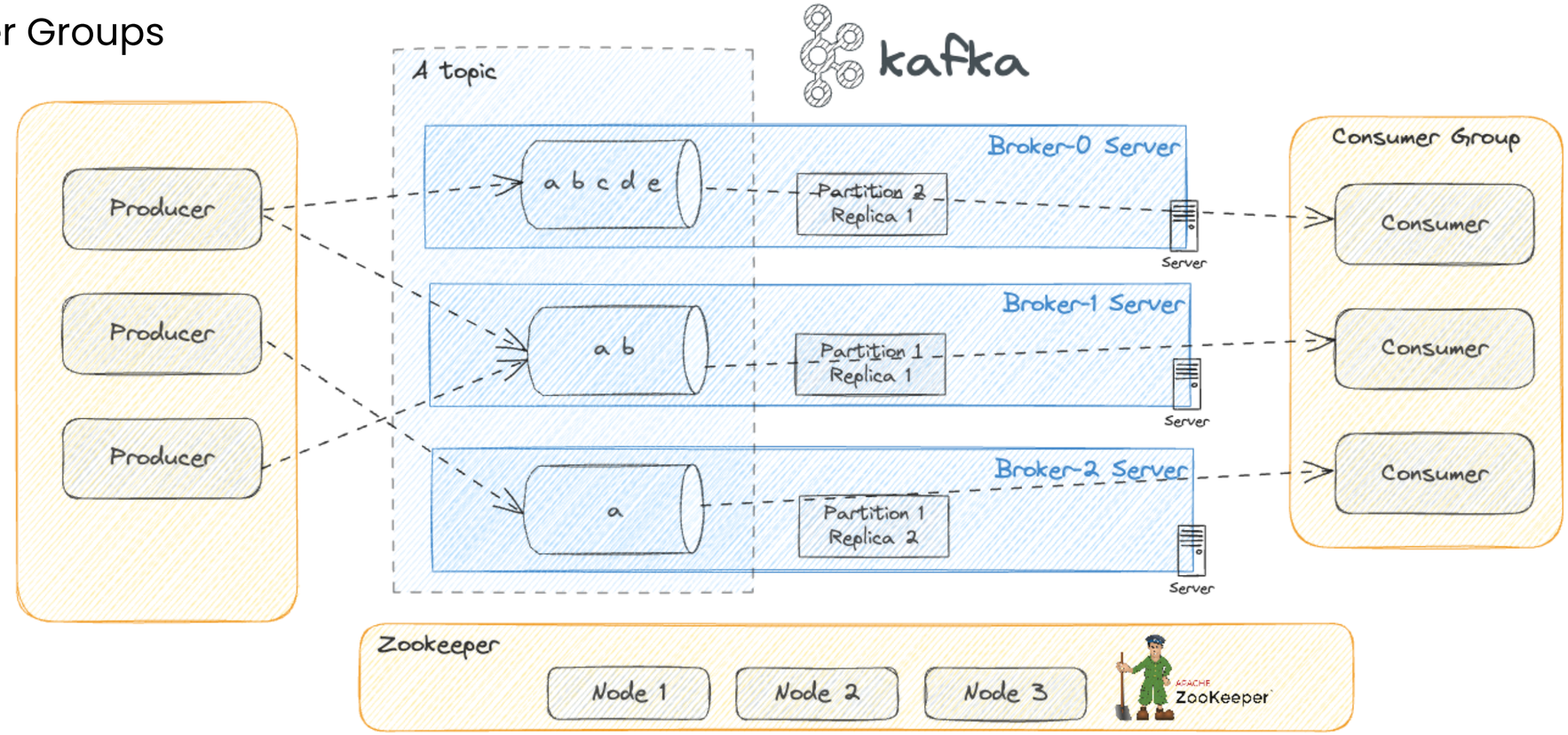
Introduction to Apache Kafka

- **Definition**: Distributed messaging queue platform
- **Key features**:
  - Publish-subscribe messaging
  - Fault-tolerant storage
  - Stream processing



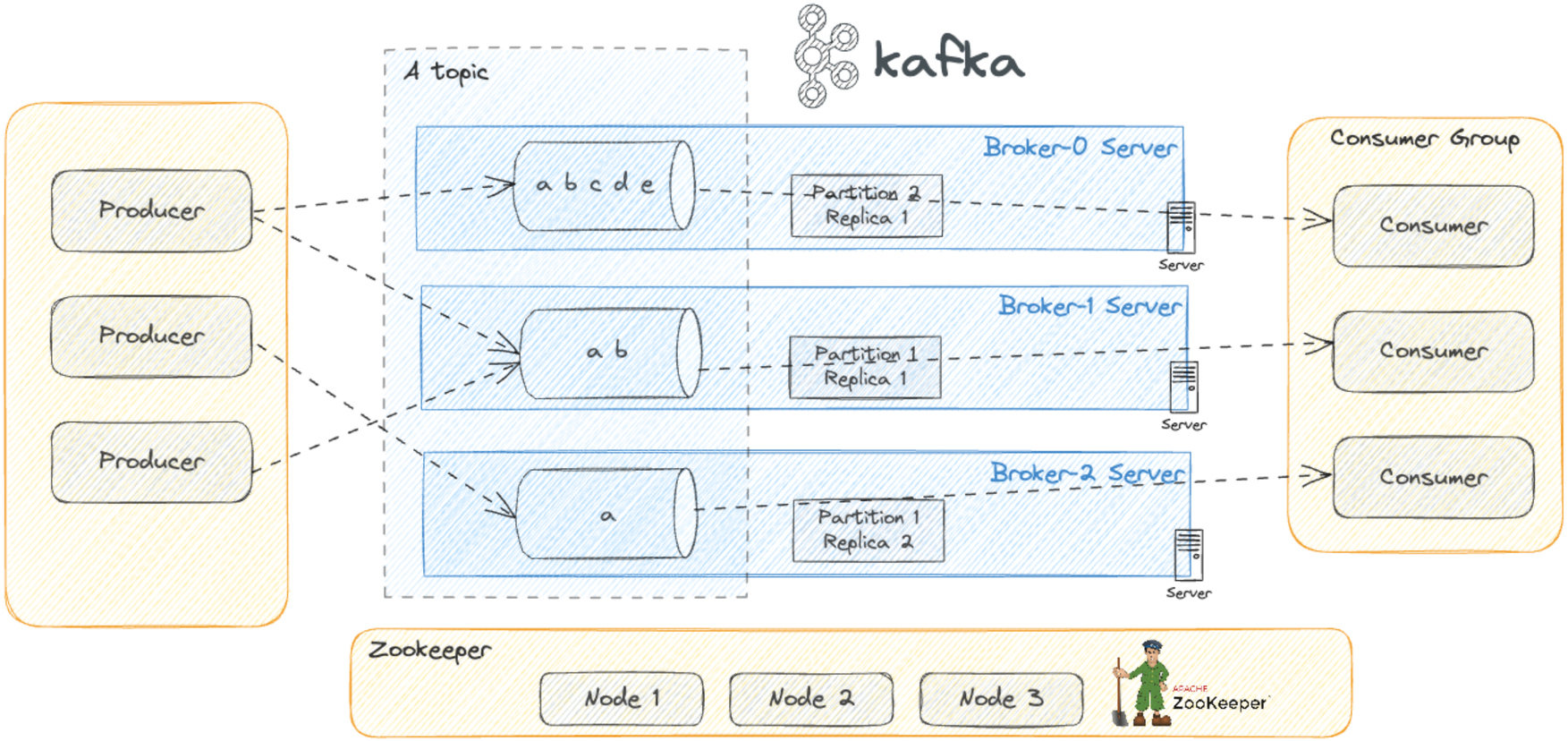
Kafka Architecture

- Brokers
- Topics and Partitions
- Producers
- Consumers and Consumer Groups
- ZooKeeper



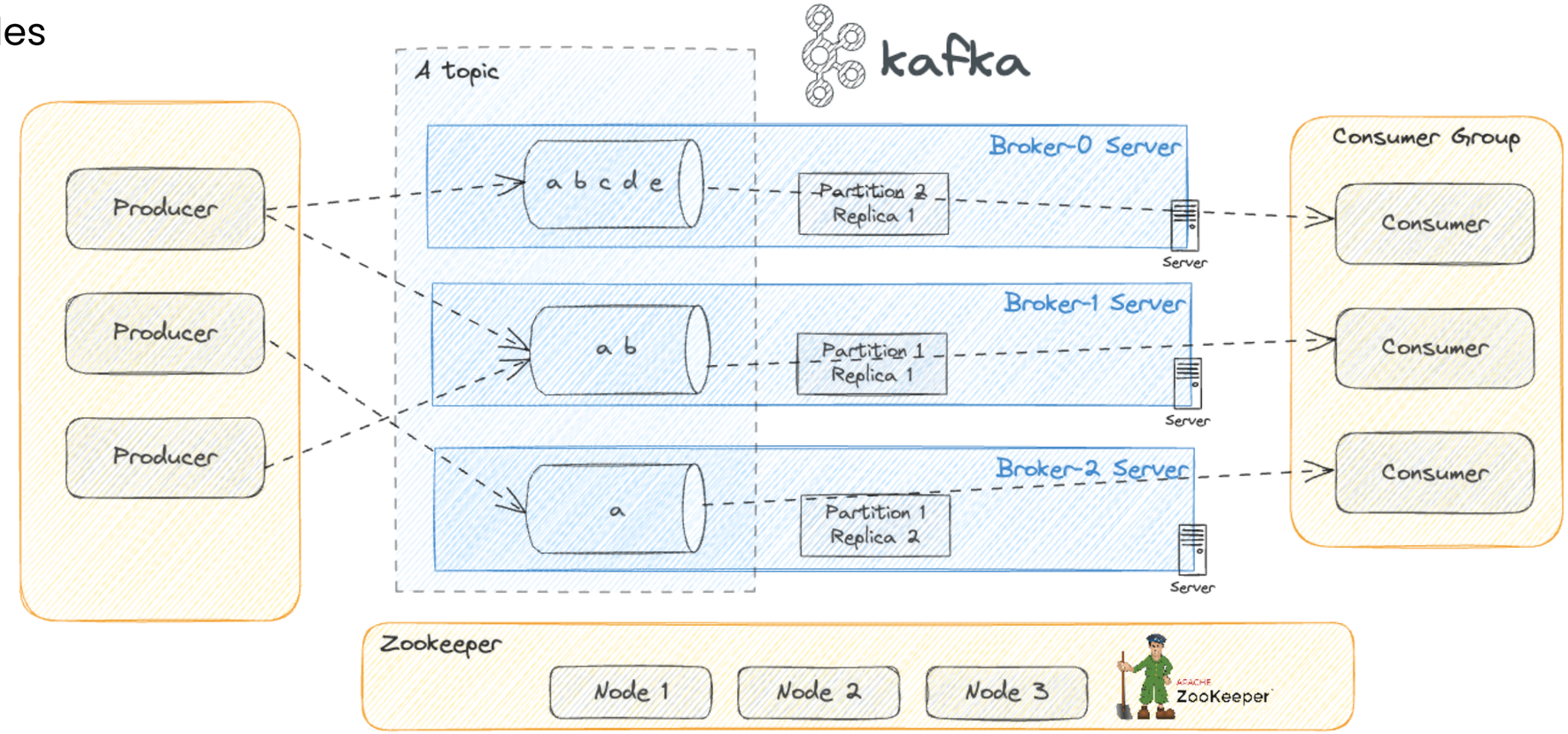
Kafka Topics and Partitions

- Topics: Categories for organizing messages
- Partitions: Units of parallelism within topics



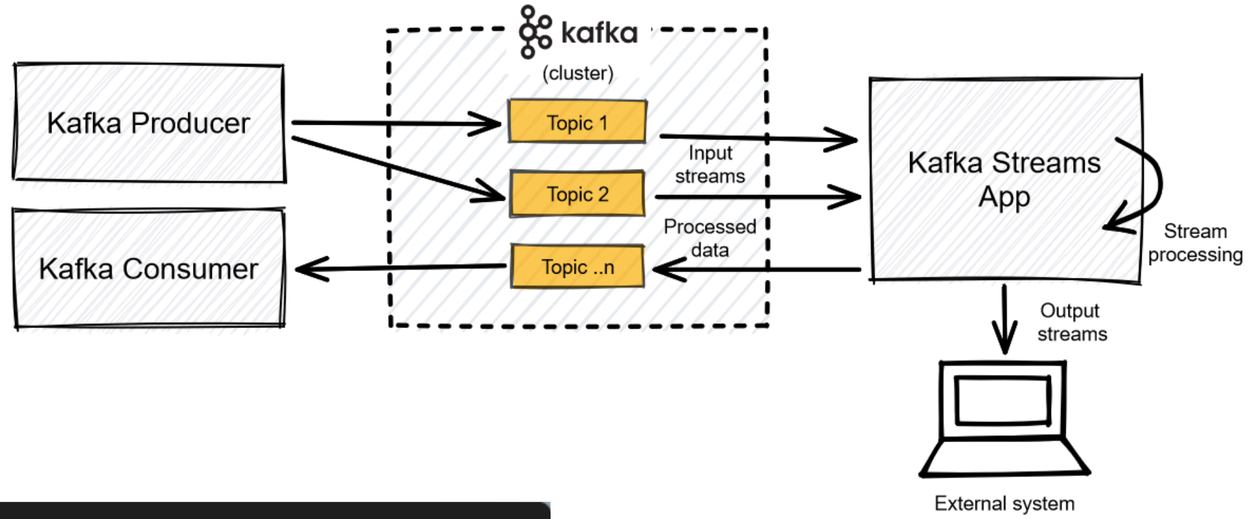
Kafka Producers & Consumers

- Kafka Producers:
  - Sending messages to topics
  - Partitioning strategies
  - Acknowledgment modes
- Kafka Consumers
  - Subscribing to topics
  - Consumer groups for scalability
  - Offset management



Kafka Streams

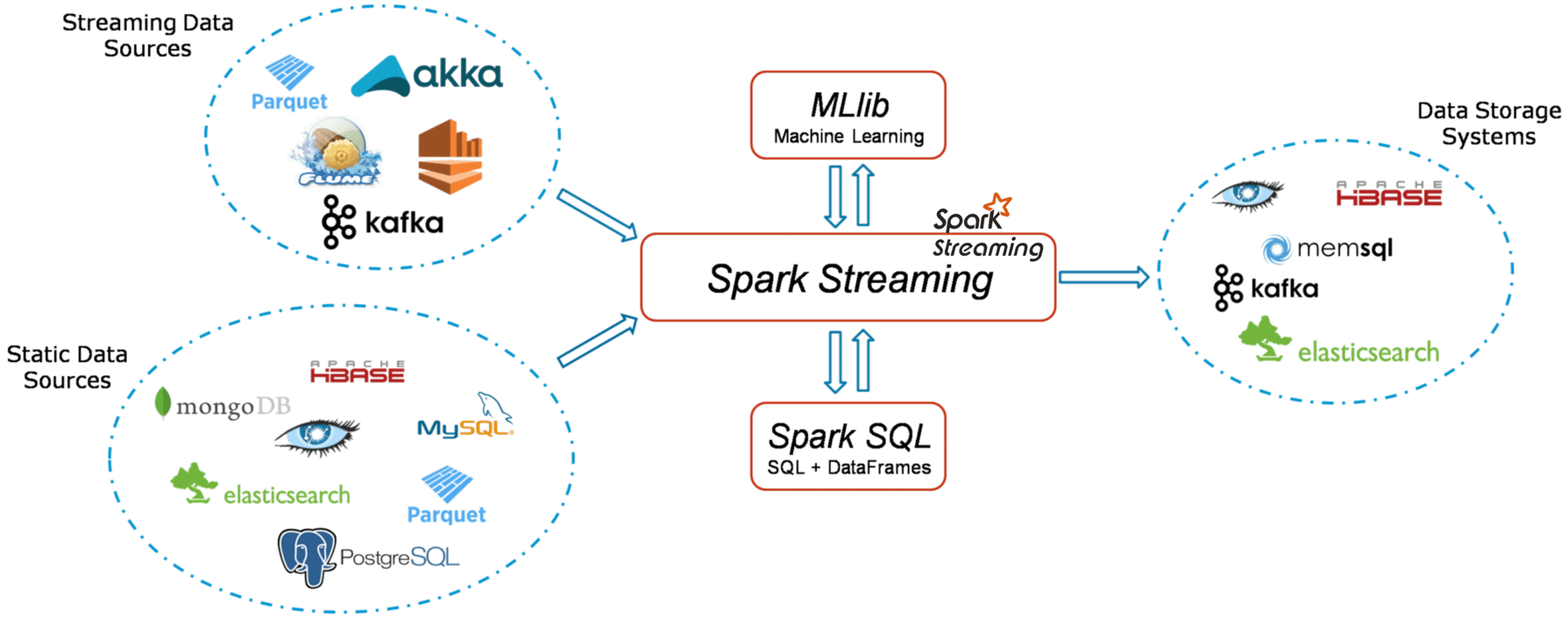
- Lightweight library for stream processing
- Stateful and stateless operations
- Exactly-once semantics




```
1  final StreamsBuilder builder = new StreamsBuilder();
2  builder.stream("widgets", Consumed.with(stringSerde, widgetsSerde))
3         .filter((key, widget) -> widget.getColour().equals("red"))
4         .to("widgets-red", Produced.with(stringSerde, widgetsSerde));
```

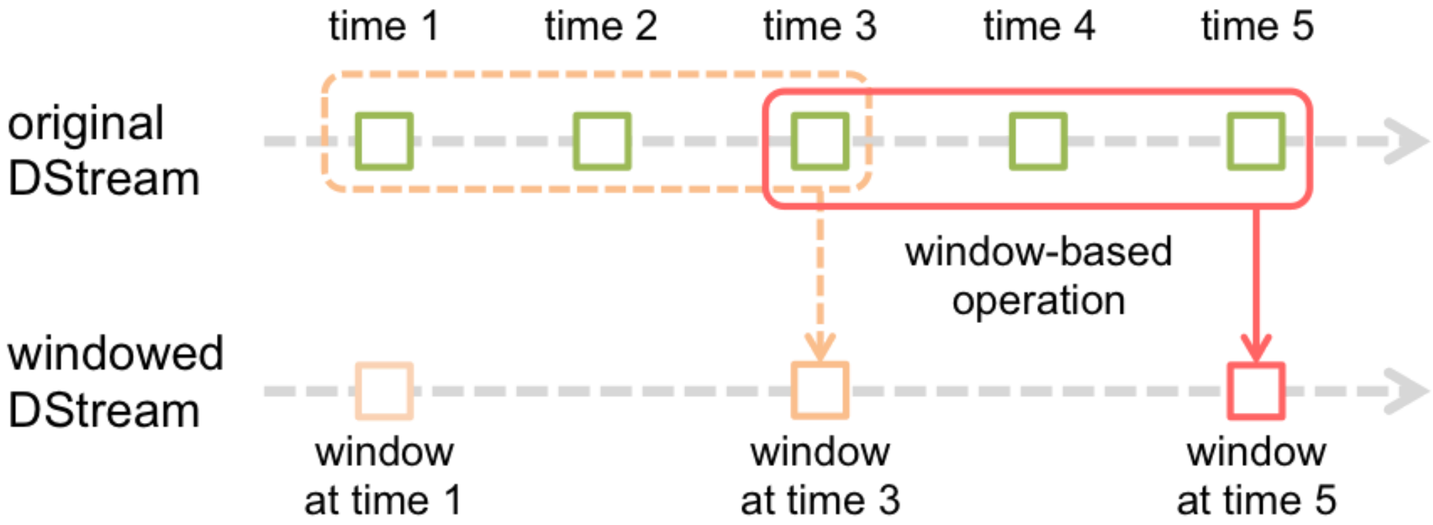
Spark Streaming

- Definition: Extension of the core Spark API for processing live data streams
- DStreams (Discretized Streams): Continuous stream of RDDs
- Input sources: Kafka, Flume, Kinesis, TCP sockets



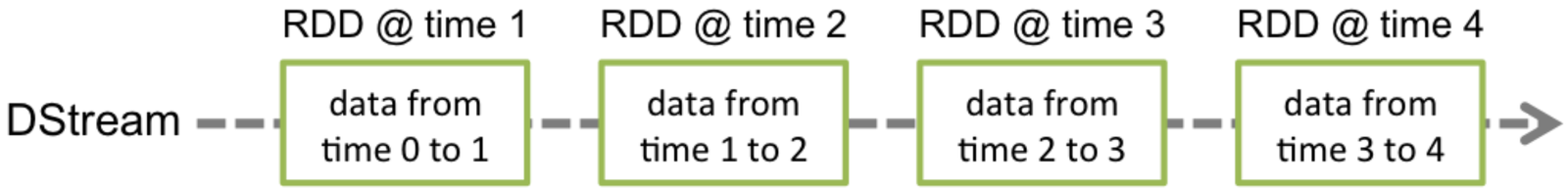
Spark Streaming Operations

- Transformations on DStreams
- Window operations
- Join operations
- Output operations



Structured Streaming

- Definition: Scalable and fault-tolerant stream processing built on the Spark SQL engine
- DStream is represented by a continuous series of RDDs
- Advantages over DStreams:
  - Simpler programming model
  - Stronger consistency guarantees



Structured Streaming Programming Model

- Input sources
- Operations and queries
- Output sinks
- Triggers and watermarking

Source

Transformations

Sink

```python
# Read streams from Kafka
votes_df = spark.readStream \
.format("kafka") \
.option("kafka.bootstrap.servers", KAFKA_BOOTSTRAP_SERVER) \
.option("subscribe", votes_topic) \
.option("startingOffsets", "earliest") \
.load() \
.selectExpr("CAST(value AS STRING)") \
.select(from_json(col("value"), vote_schema).alias("data")) \
.select("data.*")

# Data preprocessing: type casting and watermarking
votes_df = votes_df \
  .withColumn("voting_time", col("voting_time").cast(TimestampType())) \
  .withColumn('vote', col('vote').cast(IntegerType()))

enriched_votes_df = votes_df.withWatermark("voting_time", "1 minute")

# Write streams to Kafka
 votes_per_candidate_to_kafka = votes_per_candidate \
 .selectExpr("to_json(struct(*)) AS value") \
 .writeStream \
 .format("kafka") \
 .option("kafka.bootstrap.servers", KAFKA_BOOTSTRAP_SERVER) \
 .option("topic", votes_per_candidate_topic) \
 .option("checkpointLocation", "/opt/.../checkpoints/checkpoint1") \
 .outputMode("update") \
 .start()
```

Best Practices for Streaming Data Processing

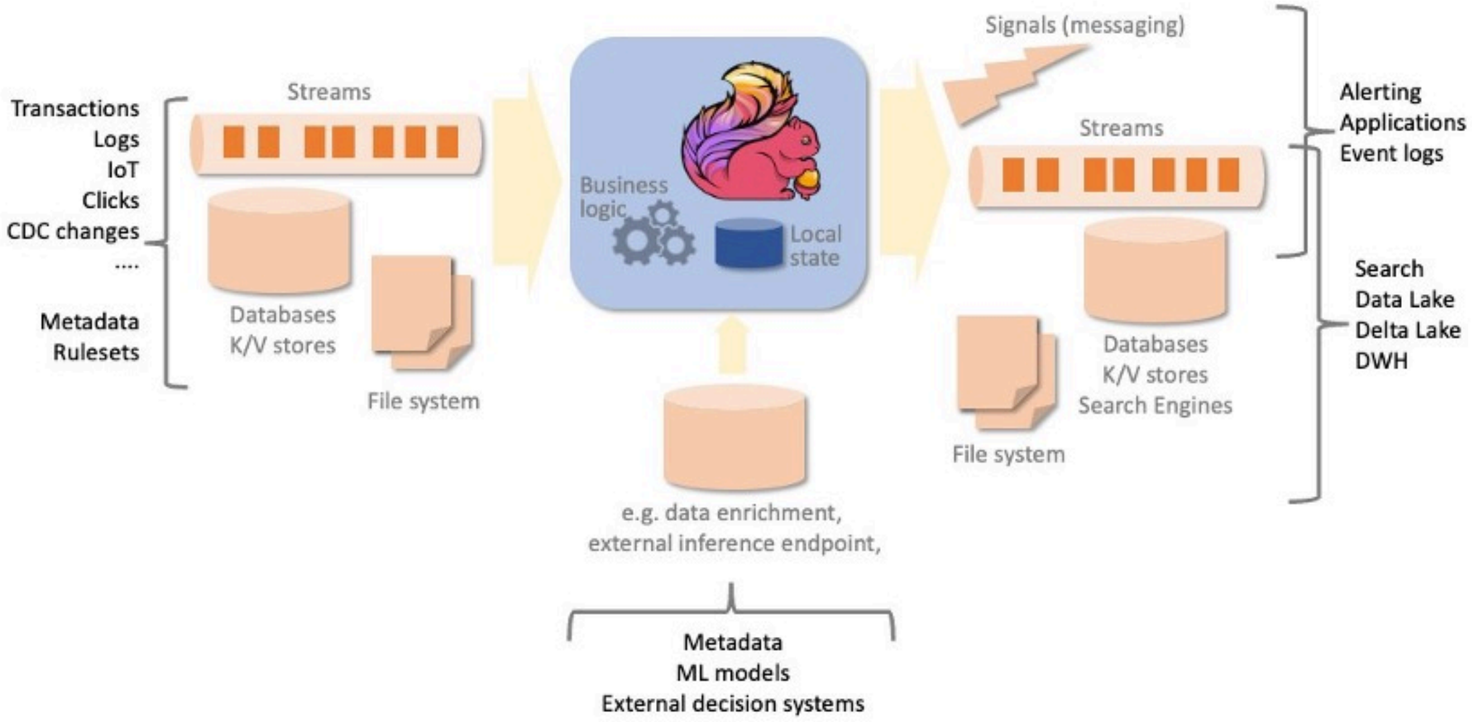
- Handling late and out-of-order data
- Scaling and performance tuning
- Monitoring and alerting
- Data quality and validation in streams

Hands-on:

Data Streaming Pipelines: Kafka & Spark-Streaming

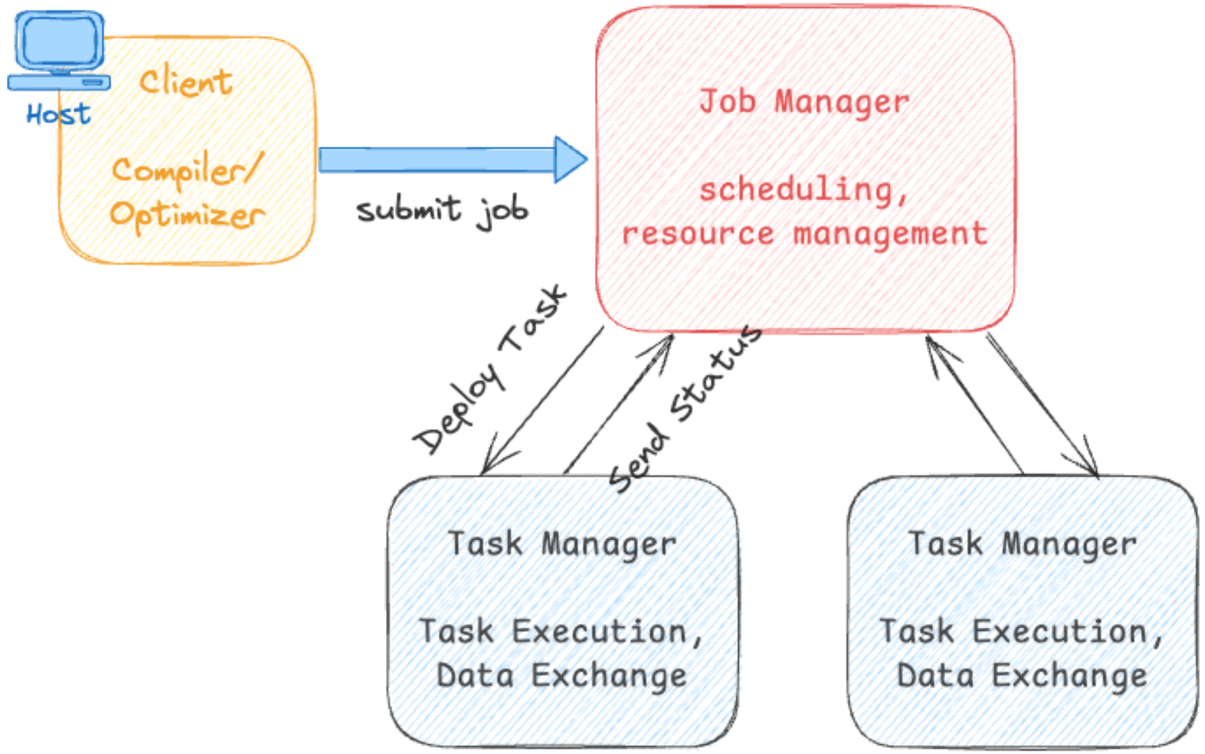
Introduction to Apache Flink

- **Definition**: Distributed processing engine for stateful computations over data streams
- **Key features**:
  - Event time processing
  - Stateful computations
  - Exactly-once semantics:
    each incoming event affects
    the final results exactly once



Flink Architecture

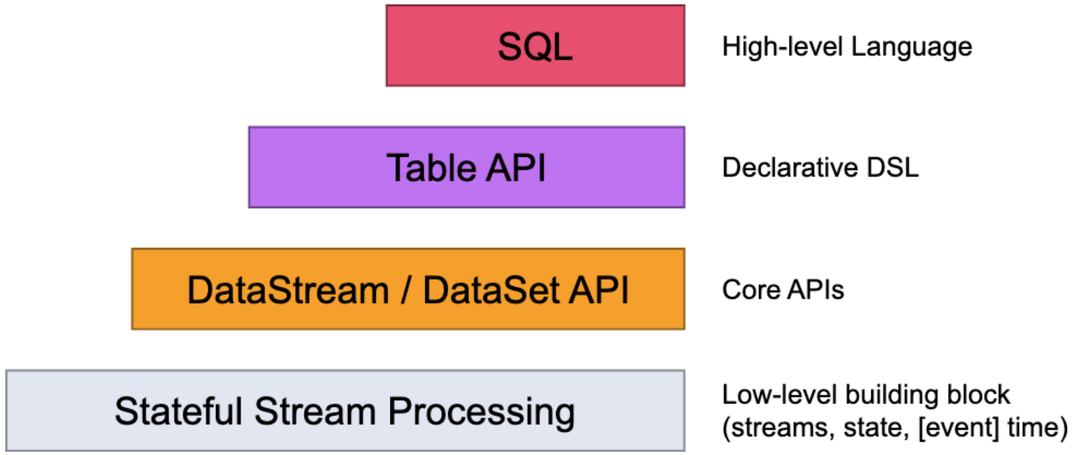
- JobManager
- TaskManagers
- Client

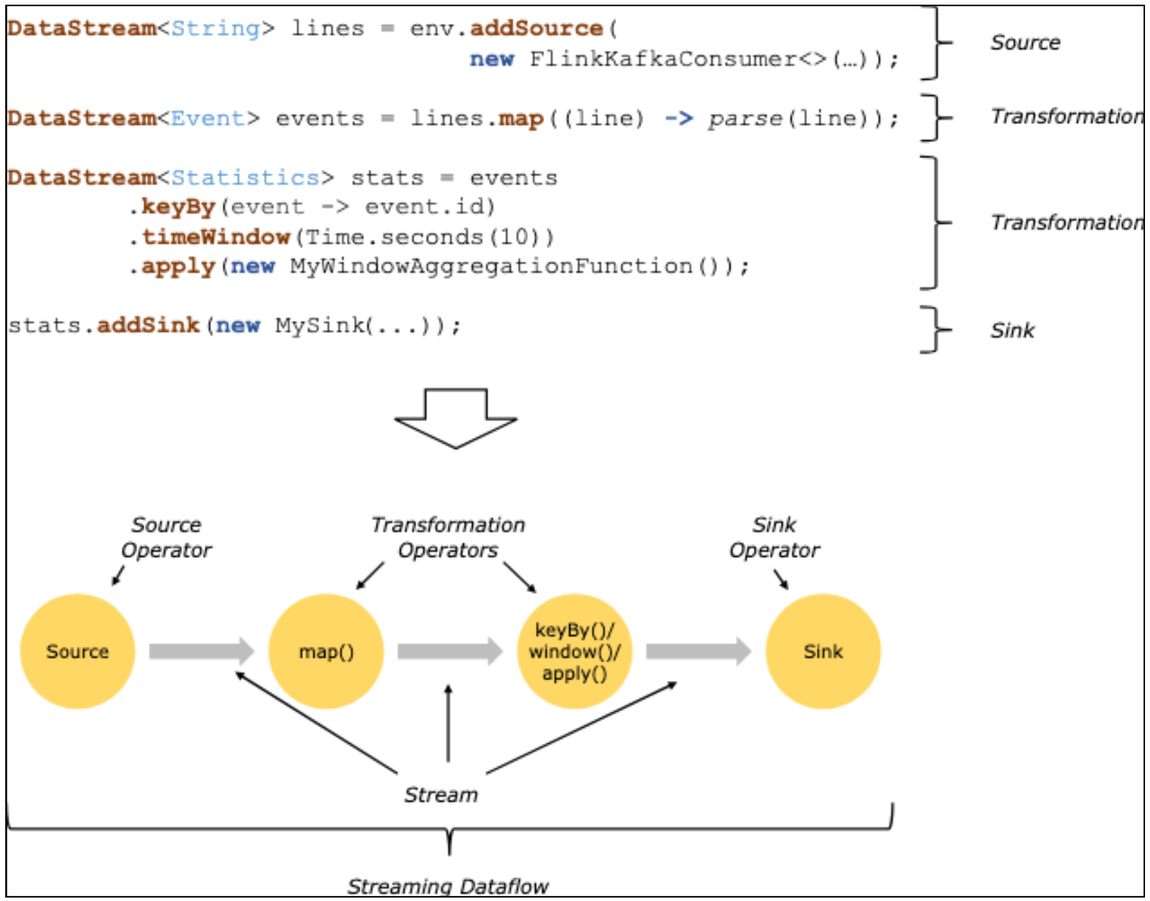


Flink Programming Model

Flink offers different levels of abstraction to develop streaming/batch applications:

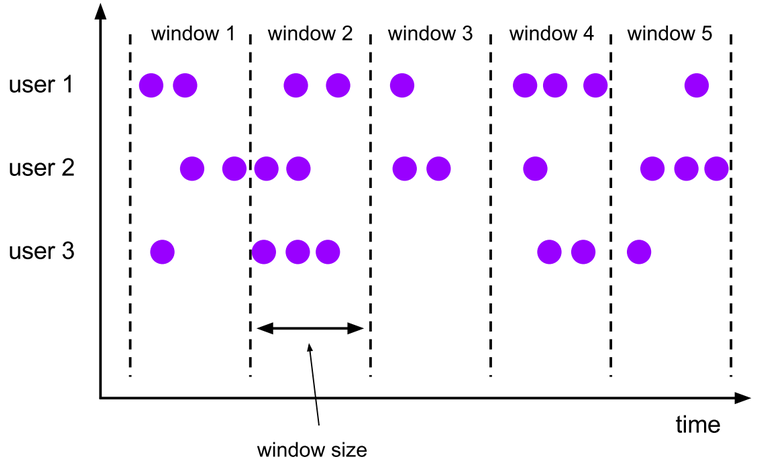
- DataStream API
- Table API and SQL
- ProcessFunction for low-level operations



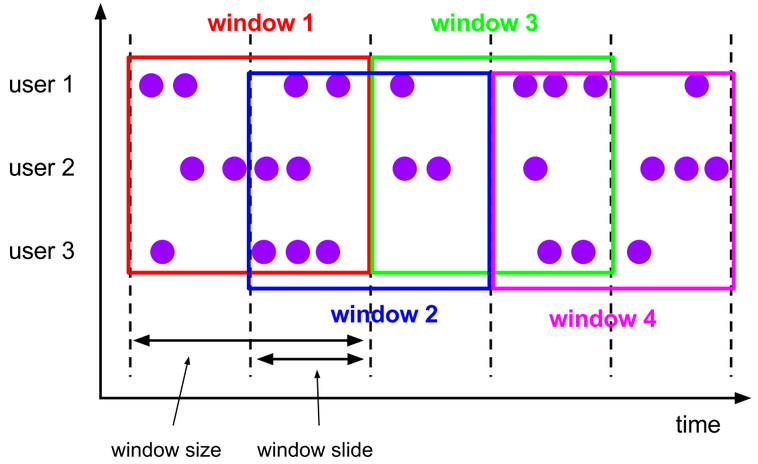


Flink Windows and Time

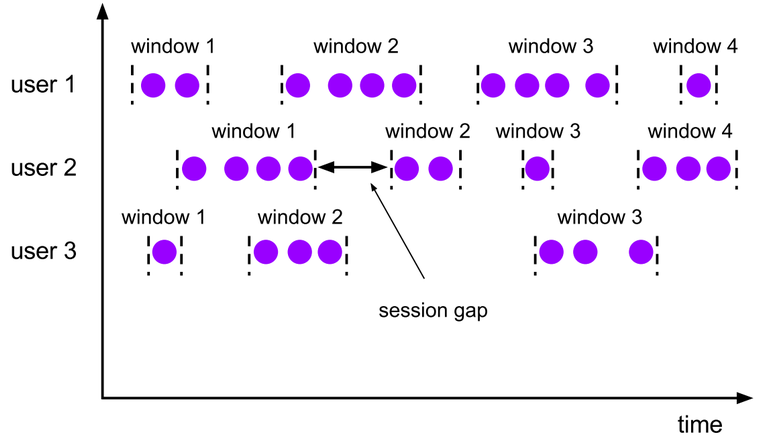
- Types of windows:
  - Tumbling windows
  - Sliding windows
  - Session windows
- Event time vs. Processing time



Tumbling windows



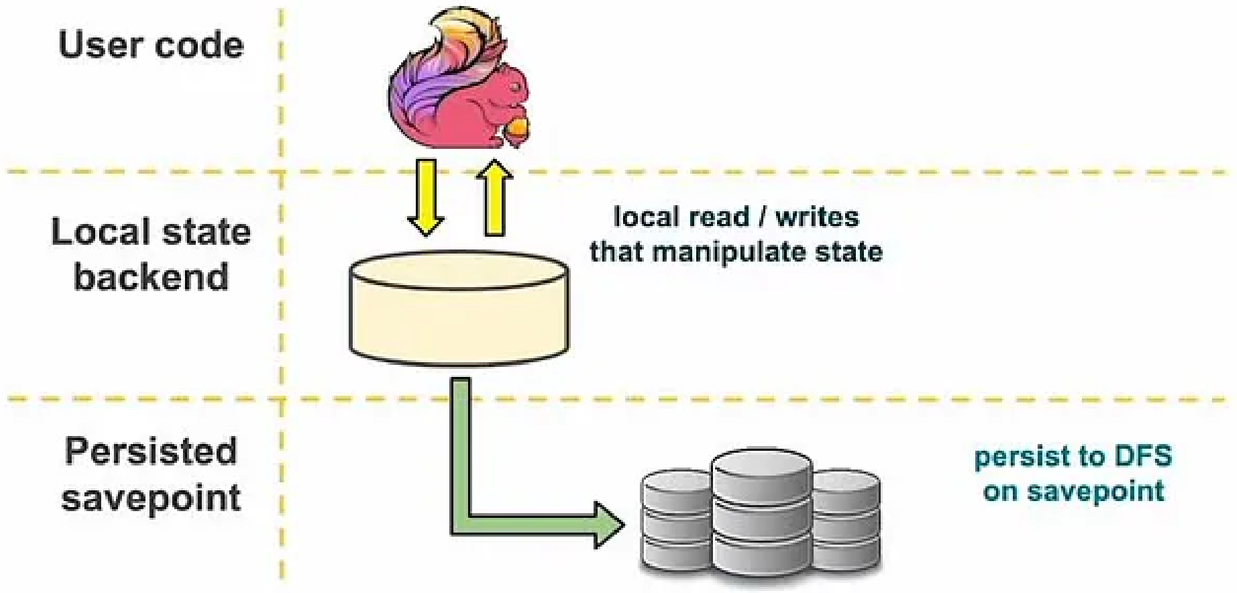
Sliding windows



Session windows

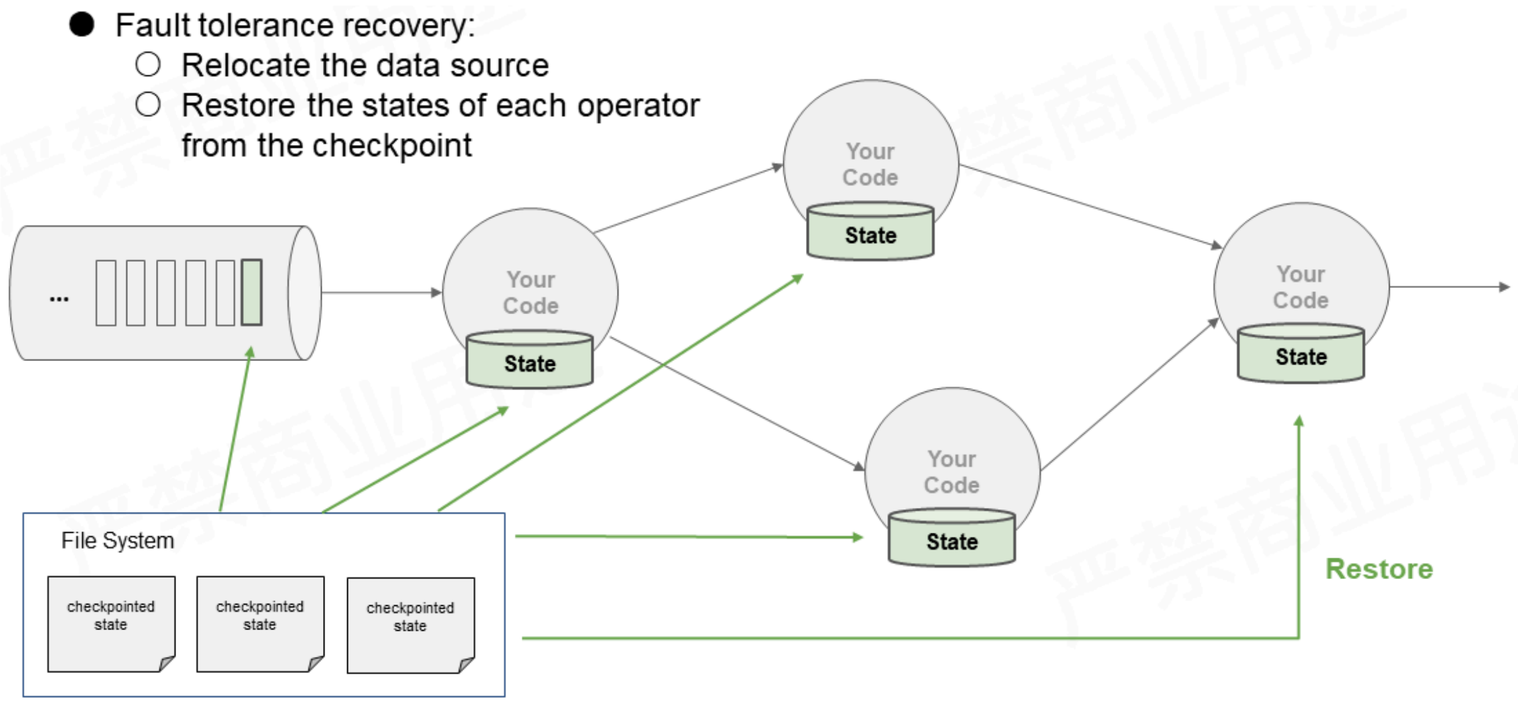
Flink State Management

- Keyed state
- Operator state
- State backends
  - RocksDB: is a high performance embedded database for key-value data.
  - heap memory



Flink Fault Tolerance

- Fault tolerance means restoration to the states before the faults.
  - Checkpointing
  - Savepoints
  - Exactly-once guarantees



Comparison of Streaming Frameworks

Feature	Kafka Streams	Apache Flink	Spark Streaming
Type	Stream Processing Library	Stream Processing Framework	Stream Processing Framework
Latency	Low	Very Low	Low (Micro-batch)
Processing Model	Record-by-Record	Record-by-Record or Batch-like	Micro-batch
State Management	RocksDB, in-memory	RocksDB, in-memory, custom	In-memory, HDFS
Fault Tolerance	Log-based storage	Checkpointing, Savepoints	Checkpointing
Scaling	Limited by Kafka Cluster	Dynamic scaling	Cluster-based scaling
Use Cases	Real-time analytics, ETL	Complex event processing, ETL	Real-time analytics, ETL, batch
Ease of Use	Simple, requires Kafka knowledge	Flexible but more complex	Moderate, requires Spark knowledge

Hands-on:

Data Streaming Pipelines: Flink

Q&A and Discussion

MEET OUR TEAM

Omar AlSaghier

Sr. Data Engineer

DATATECH LABS.

THANK YOU

OUR CONTACT

DataTechLabs

datatechlabs.ai@gmail.com

datatechlabs.ai

Amman, Jordan