# Design Document for Real-Time Fraud Detection System

## 1. Introduction

### Overview

Financial institutions handle a large volume of transactions every second, and detecting fraudulent transactions in real-time is essential to safeguard customer assets. This system is designed to detect fraud in real-time, processing incoming transactions as data streams, identifying suspicious activities, and triggering alerts. The architecture leverages both real-time and batch processing patterns for flexibility and scalability.

### Objective

To build a scalable fraud detection system that continuously processes and analyzes financial transactions using Apache Kafka, Apache Spark, and Python.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 2. Architecture Overview

The solution consists of two primary data processing approaches:

## Batch Data Processing through Scheduler

The batch data processing approach leverages Apache Spark for processing large volumes of transaction data stored in CSV or OLTP systems. The system utilizes Apache Airflow as a job scheduler, set to run every 10 minutes. This ensures that the transaction data is ingested at regular intervals, cleaned, transformed, and stored in Parquet format for further analysis.

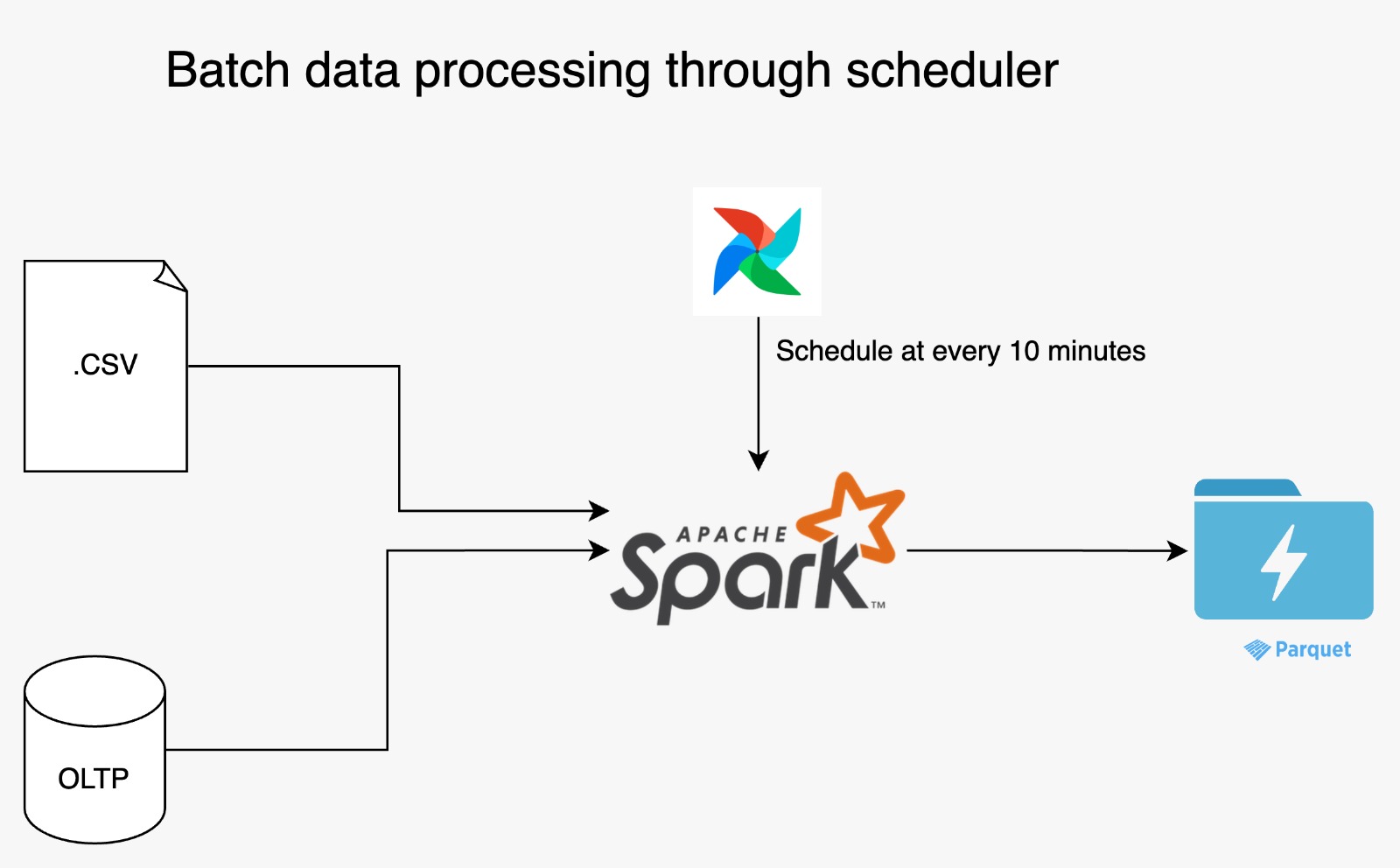


Figure 1: Batch Data Processing through Scheduler

## Real-Time Data Streaming through Broker

In the real-time processing architecture, transactions are ingested through a Kafka Broker. The broker architecture pattern is implemented using a Python-based producer and consumer, with Apache Spark handling real-time data processing. This setup allows the system to respond instantly to suspicious transactions and store the processed results in a Parquet file for immediate alerts and historical review.

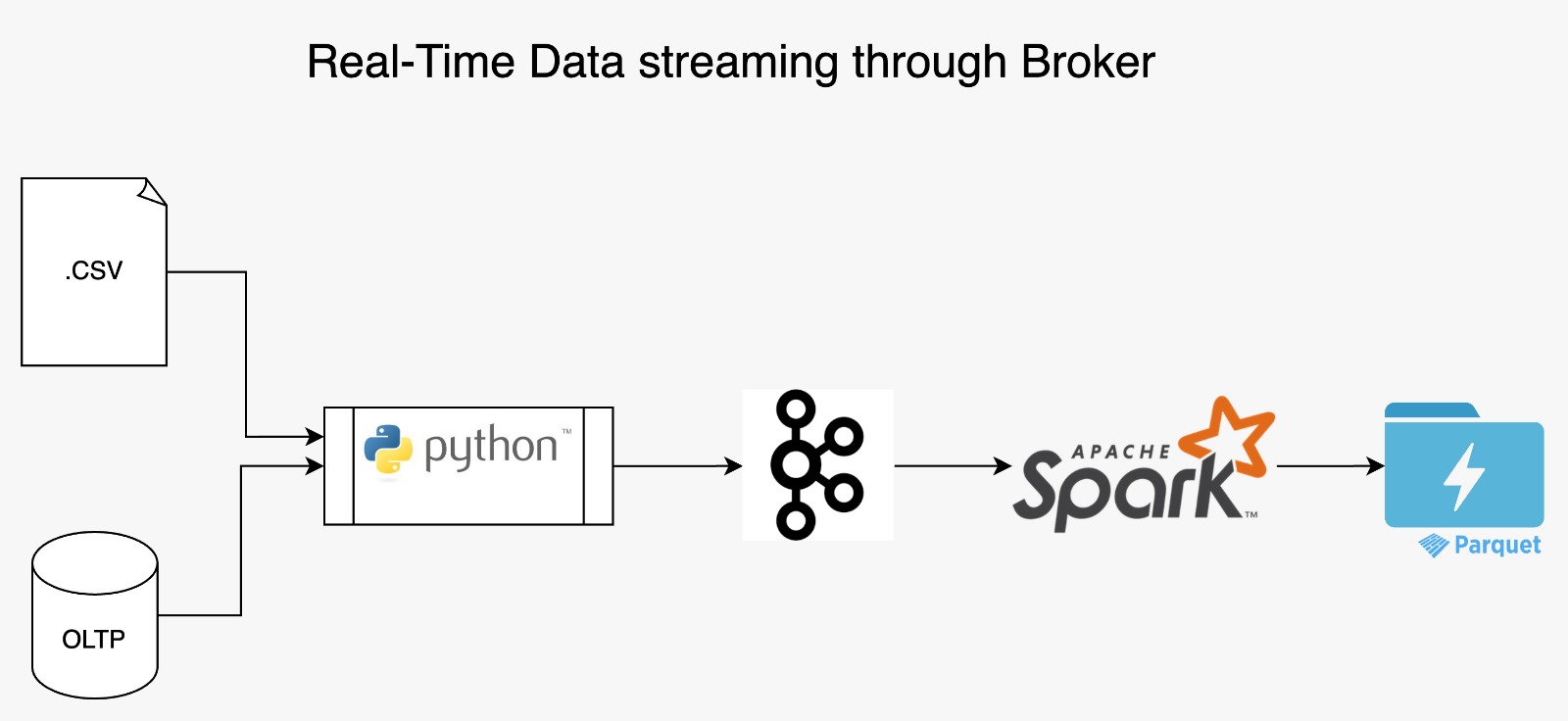


Figure 2: Real-Time Data Streaming through Broker

### 2.1 Real-Time Data Streaming (Broker Architecture Pattern)

• Data Source: Financial transactions from various sources (CSV files and OLTP systems).  
• Data Ingestion: Python scripts ingest data and feed it into Apache Kafka.  
• Data Processing: Apache Spark Streaming processes the real-time data.  
• Data Storage: Results are stored in Parquet format for efficient querying.

### 2.2 Batch Processing (Batch Processing Architecture Pattern)

• Data Source: The same financial transaction data from CSV and OLTP systems.  
• Data Processing: Apache Spark processes the data in batches.  
• Scheduling: The system uses Apache Airflow to schedule the batch processing every 10 minutes.  
• Data Storage: Results are also stored in Parquet format for further analysis.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 3. Technology Stack

• Data Streaming: Apache Kafka  
• Programming Language: Python (for data ingestion and processing)  
• Distributed Data Processing: Apache Spark  
• Job Scheduling: Apache Airflow  
• Data Storage: Parquet format for efficient data querying and storage

## 4. Architecture Patterns

### 4.1 Broker Architecture Pattern (Real-Time)

• Use Case: Ideal for real-time fraud detection where the system must react instantly to events.  
• Key Benefits:  
 o Decouples producers (transaction sources) from consumers (fraud detection engines).  
 o Ensures scalability by distributing data over a cluster of machines.  
 o Provides resilience through Kafka’s fault-tolerant design.

### 4.2 Batch Processing Architecture Pattern

• Use Case: Suitable for non-time-sensitive analysis tasks like periodic reporting or aggregate statistics on historical transaction data.  
• Key Benefits:  
 o Efficient processing of large volumes of data in a single batch.  
 o Cost-effective resource utilization by scheduling jobs during off-peak hours.  
 o Ensures consistency and accuracy in data processing.

## 5. System Components

### Kafka Producer

• Module: kafka\_producer.py  
• Description:  
 Ingests transactions from CSV files and OLTP systems.  
Sends batches of transaction data to the Kafka broker.

### Kafka Consumer & Stream Processing

• Module: mainapp\_stream.py  
• Description:  
 Automatically reads from the Kafka topic.  
Applies fraud detection logic using machine learning models.  
Flags transactions as either legitimate (0) or fraudulent (1).

### Batch Processing

• Module: mainapp\_batch.py  
• Description:  
 Processes all input data from CSV files using PySpark.  
Analyzes transaction data based on attributes like time of transaction, location, and transaction amount.  
Flags fraudulent transactions and outputs the results to a Parquet file.

### Masked Data Preparation

• Module: fake\_data\_preparation.py  
• Description:  
 Prepares synthetic or masked data to test the batch processing and streaming systems.

### Job Scheduling

• Module: airflow\_spark\_submit.py  
• Description:  
 Schedules the batch job (mainapp\_batch.py) to run every 10 minutes using Apache Airflow.

## 6. Why Kafka?

• Speed: Capable of handling high throughput, Kafka can manage large volumes of data efficiently.  
• Scalability: Kafka distributes data across clusters, enabling it to handle more data than a single machine could process.  
• Durability: Messages are persisted and replicated across brokers, ensuring no data loss.  
• Resilience: Kafka’s distributed design ensures fault tolerance, allowing the system to operate even if some brokers fail.

## 7. Future Scope

• Machine Learning for Fraud Detection: Currently, the system flags fraudulent transactions based on business logic. In the future, machine learning models can be integrated for more accurate fraud detection.  
• Data Analytics: Implementing advanced analytics for trend analysis and reporting based on the historical transaction data processed in batches.

## 8. Conclusion

This architecture ensures that the system can process both real-time and batch transactions effectively. By leveraging Apache Kafka for real-time streaming and Apache Spark for batch processing, the system is scalable, resilient, and designed to meet the high throughput requirements of financial institutions. The system's extensibility allows for future enhancements like machine learning integration and more sophisticated fraud detection mechanisms.