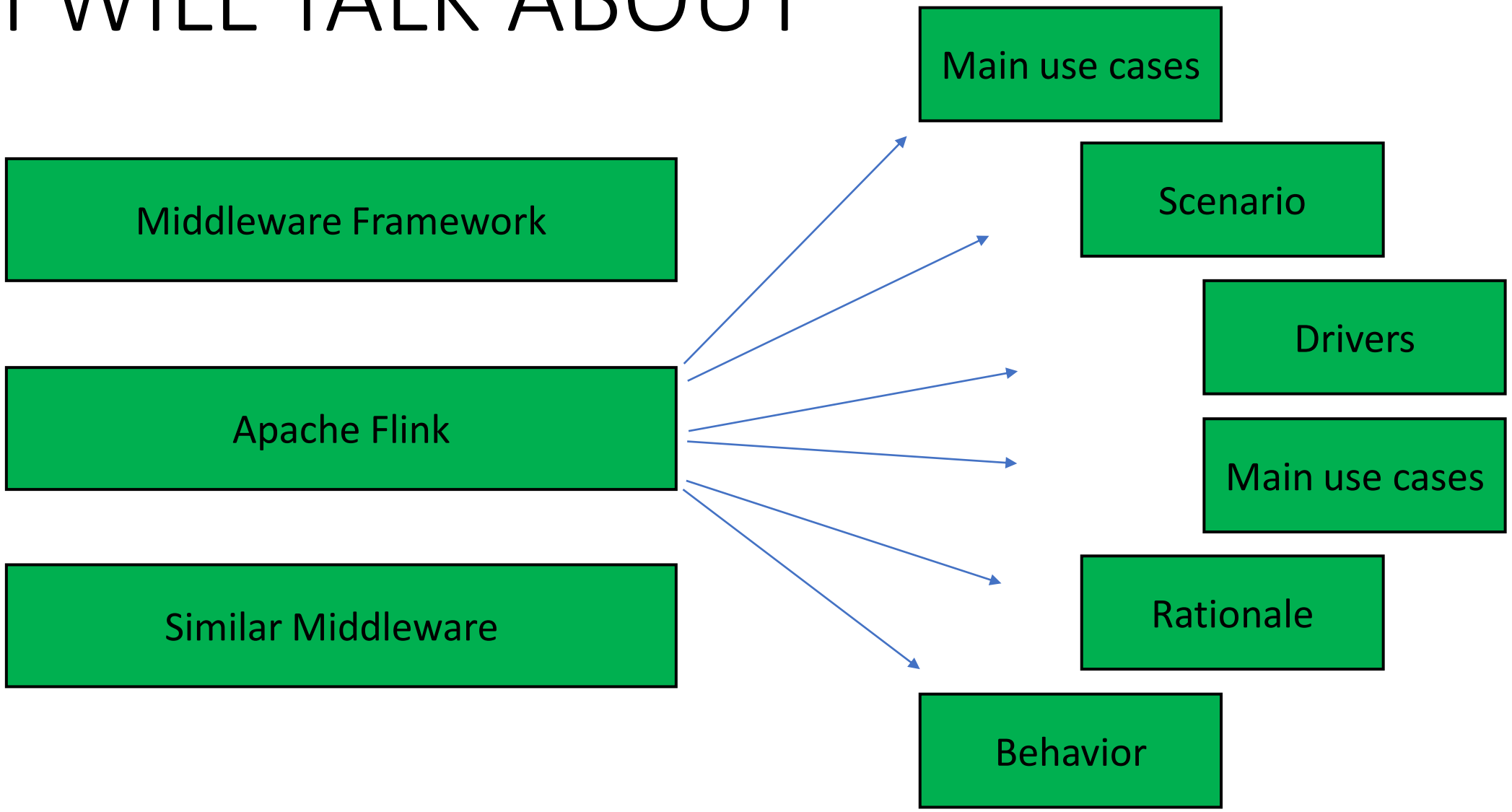
APACHE FLINK

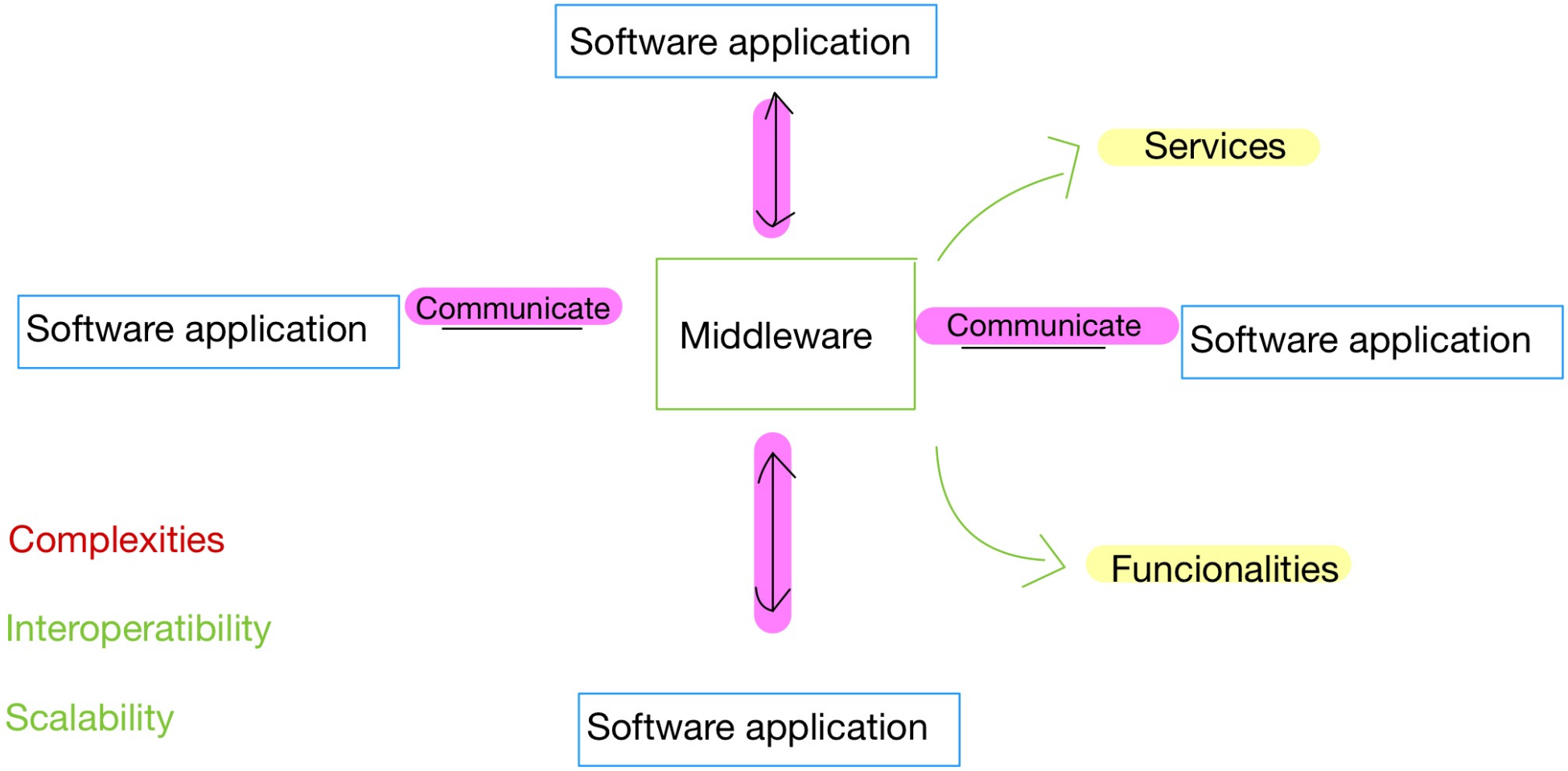
DISTRIBUTED SOFTWARE SYSTEM MIDTERM PRESENTATION

Carlos Lozano Alemañy

I WILL TALK ABOUT



MIDDLEWARE FRAMEWORK



Complexities

Interoperatibility

Scalability

Reliability and Fault tolerance

MAIN USE CASES(1/2)

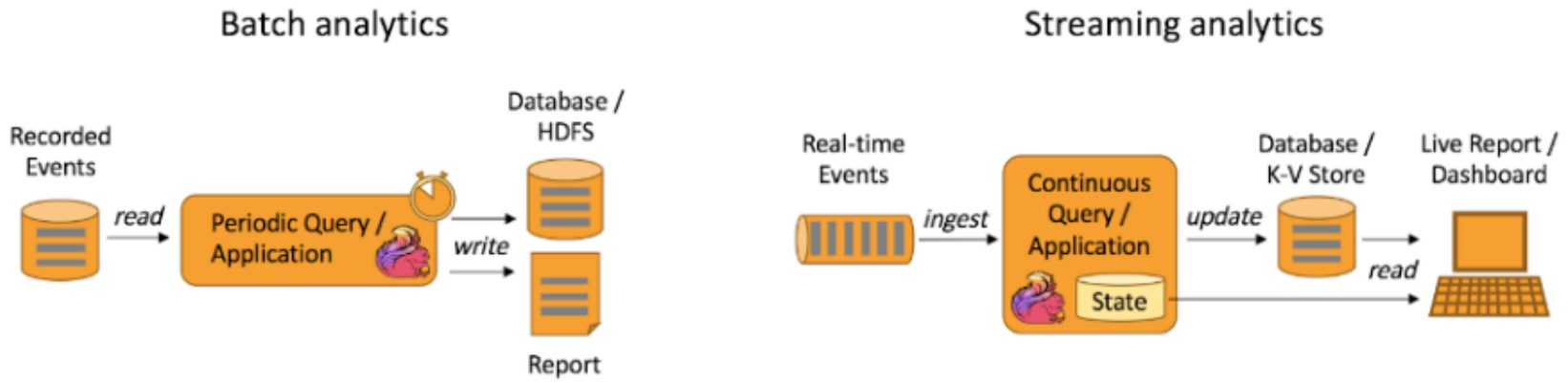
1. **Batch processing**

Apache Flink offers support for batch processing, a feature particularly advantageous for tasks involving the simultaneous handling of extensive datasets. This capability proves crucial in scenarios like data storage, where the efficiency of managing massive volumes of data is of utmost importance

Also offers analytical tasks which focus on extracting valuable information and insights from raw data.

2. **Real time data processsing (Stream Processing):**

Instead of working with fixed datasets, streaming queries or applications process real-time event streams continuously, generating and updating results as events are received.

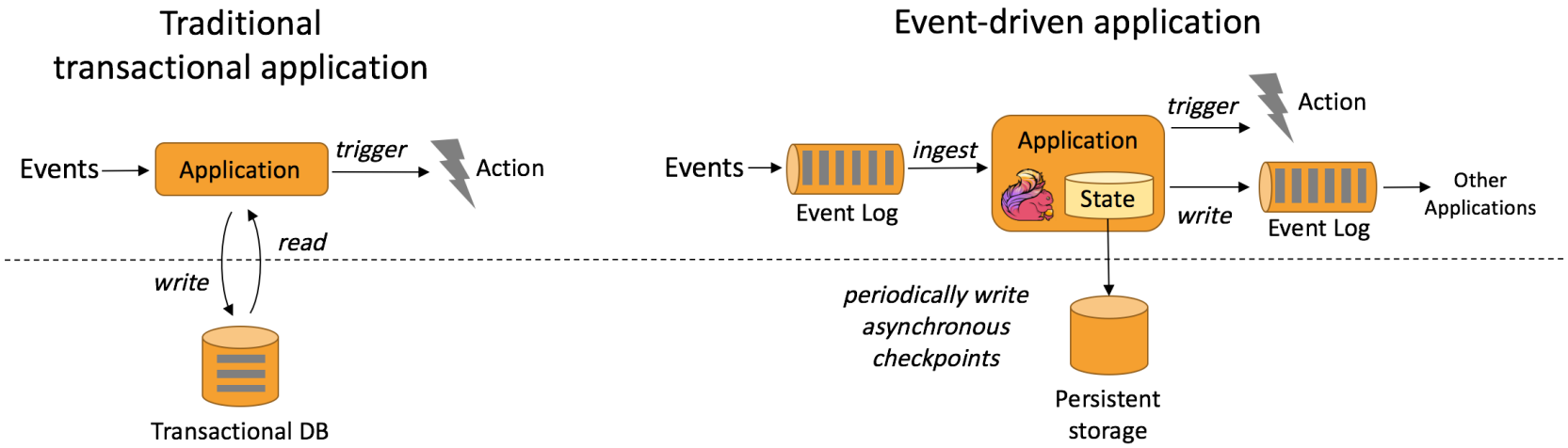


MAIN USE CASES(2/2)

3. Event-driven Applications

An event-driven application is a type of application that maintains a state and processes events from one or more event streams

Unlike traditional setups, event-driven applications rely on stateful stream processing. In this model, data and computation occur in close proximity, facilitating local data access through in-memory or disk storage. To ensure fault tolerance, the application periodically creates checkpoints that are stored in a remote persistent storage system.



SCENARIO(1/2)

A company that offers an online music streaming service, with millions of users worldwide. Users interact with the platform listening to songs, creating playlists, rating songs, skipping songs, etc.

WHAT USER EXPECTS

Time Personalization: Users expect a personalized experience. They want song recommendations based on their current tastes and activities.

Trend Detection: The platform must be able to detect real-time trends and adjust popular playlists or recommendations accordingly.

User Behavior Monitoring: Monitor user behavior to understand which features are most used, which genres are popular at certain times and how interacts with the platform.

Live Updates: Addition of new songs, changes in user ratings, should be quickly reflected in recommendations and playlists.

Integration with External Data Sources: Sharing the favorite songs, playlists, etc on the social media accounts.

SCENARIO(2/2)

WHAT APACHE PROVIDES

Using Flink's **real-time processing capability**, personalized recommendations can be generated for users based on their listening history and current activities.

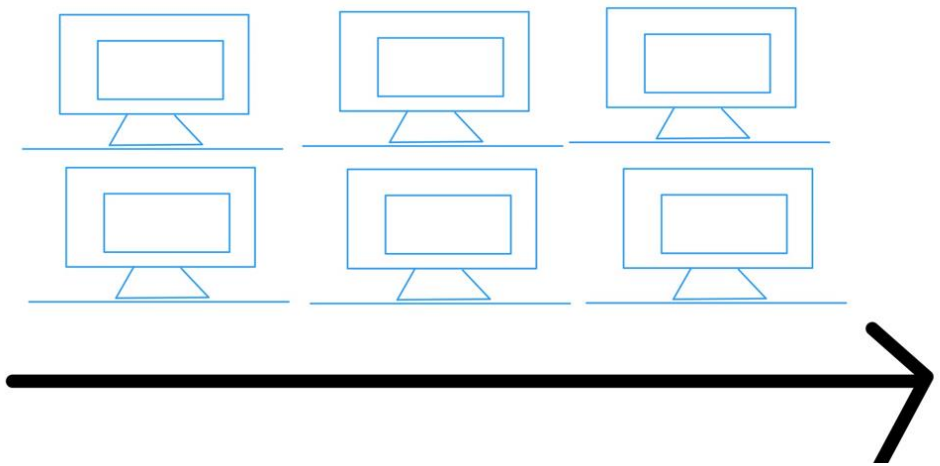
Flink analyzes emerging patterns in streaming data **to detect trends** and dynamically adjust playlists and popular recommendations.

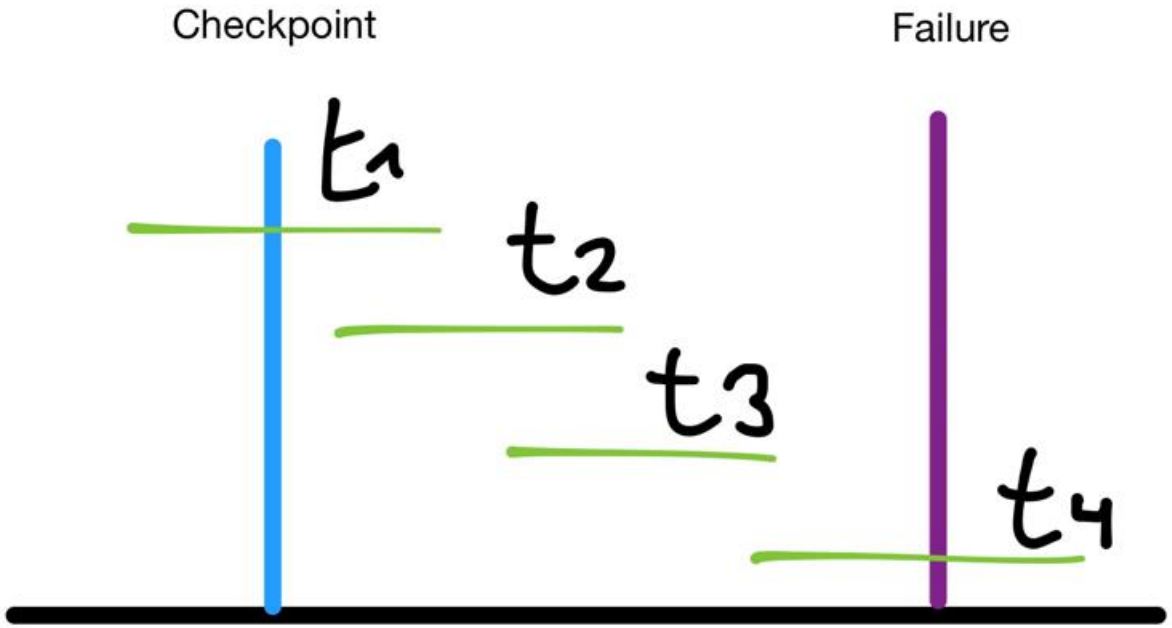
Flink enables **real-time monitoring** of user behavior, providing instant analytics on how users interact with the platform.

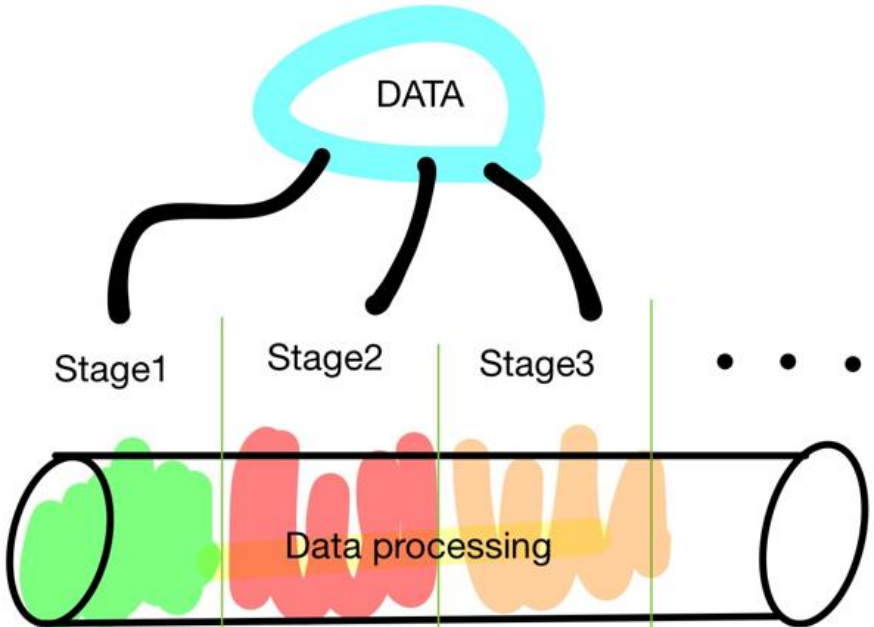
Flink can **integrate with external data sources**, such as social networks, to enrich analytics and recommendations with additional information on preferences and trends.

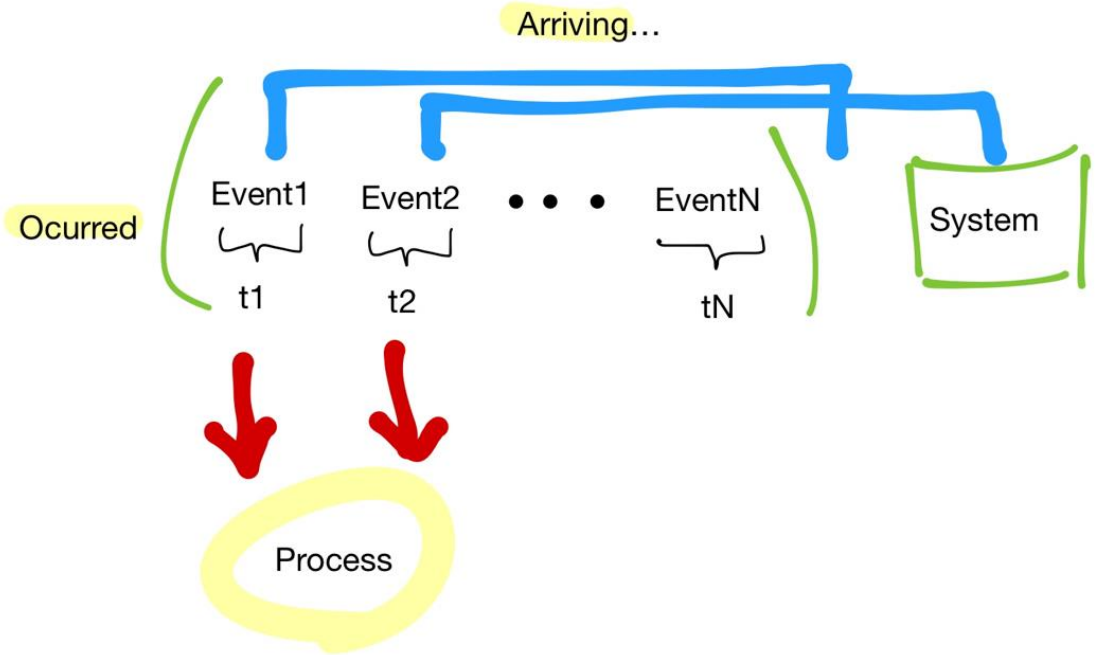
Expected Results: In this scenario, Apache Flink becomes an essential tool for processing and analyzing large volumes of data efficiently and in real-time, enabling a highly personalized and dynamic music streaming experience and also actionable analytics.

DRIVERS




Checkpoint
Failure
t1
t2
t3
t4


DATA
Stage1
Stage2
Stage3
Data processing


Arriving...
Ocurred
Event1
Event2
EventN
System
t1
t2
tN
Process

STRUCTURE

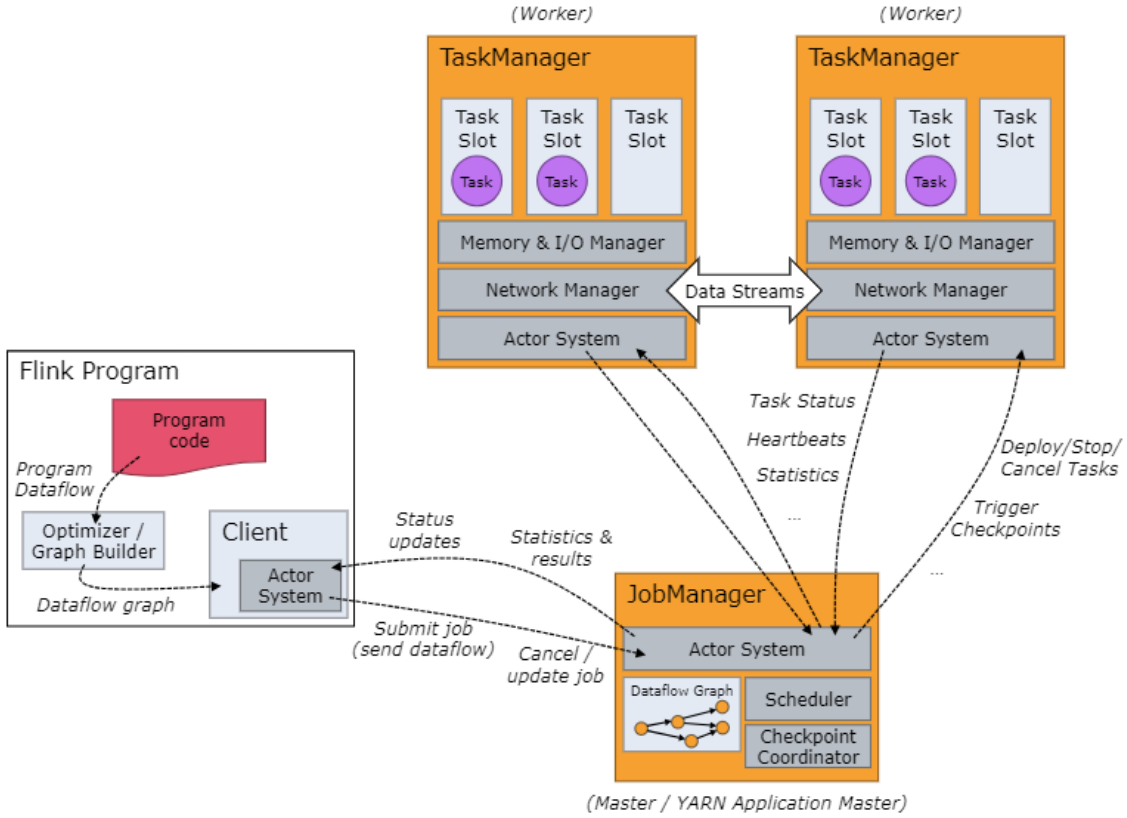
Apache Flink's architecture follows a master/worker pattern, where a central coordinator (master) manages and distributes tasks to worker nodes. There are two distinct processes in Flink, illustrated in the following diagram:

Job Managers (Master)

JobManagers are tasked with distributing tasks among TaskManagers, overseeing checkpoint management, and handling recovery in the event of failures

Task Managers (Workers)

TaskManagers play a crucial role in executing tasks and managing buffers for operators, among other responsibilities. Each TaskManager operates as a Java Virtual Machine (JVM) process, and tasks are executed within multiple threads known as task slots.



WHY FLINK HAS THIS FORM?

Apache Flink has the form it does due to strategic design choices aimed at addressing specific challenges and requirements within the realm of distributed data processing.

The framework's architecture and features are shaped by key considerations to ensure its effectiveness in various use cases. Here's why Apache Flink has the form it does:

Unified Real-Time and Batch Processing

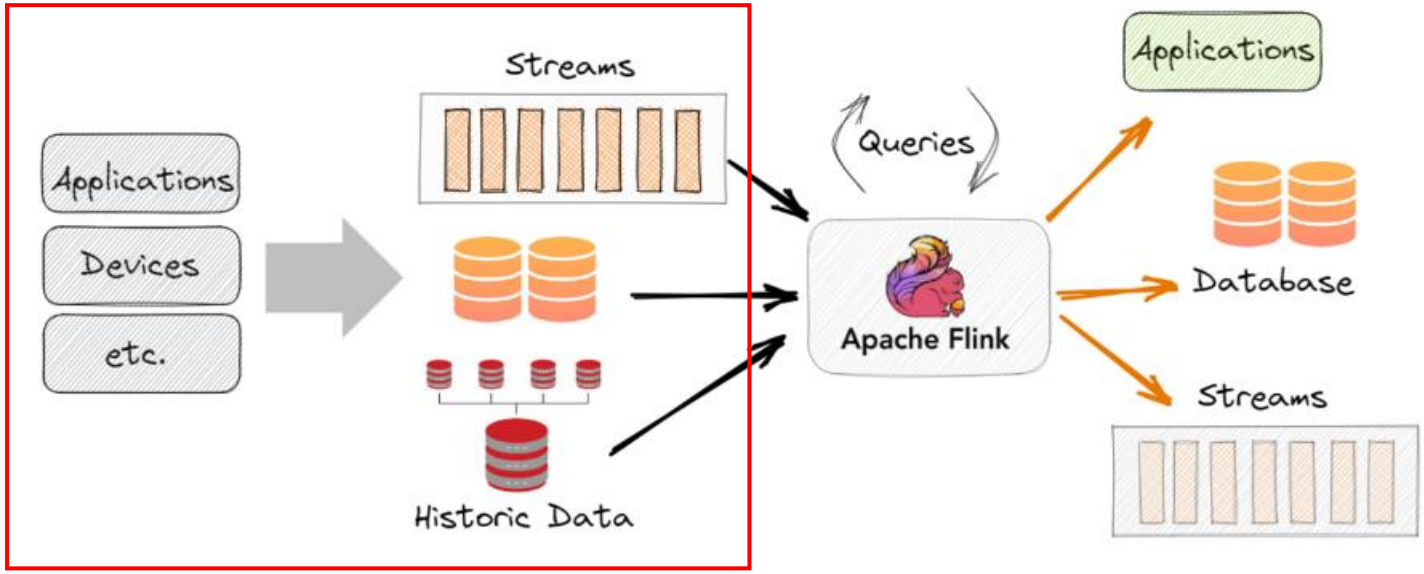
Pipelined Data Processing

- Dynamic Scaling and Horizontal Scalability

- Exactly-Once Semantics and Checkpoints

- Event Time Processing for Accuracy

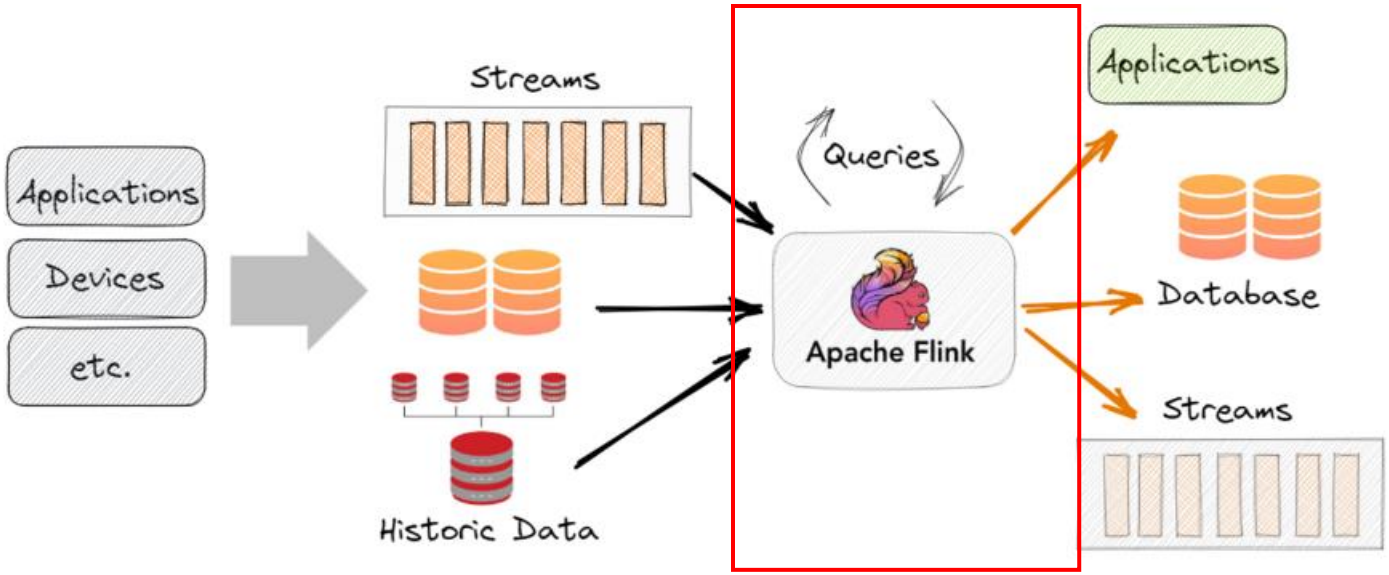
BEHAVIOR (1/3)



Data Ingest

In the first phase, Apache Flink receives data from various sources, be it streaming, batch or databases. This is accomplished through built-in connectors that allow interaction with various storage systems and data sources.

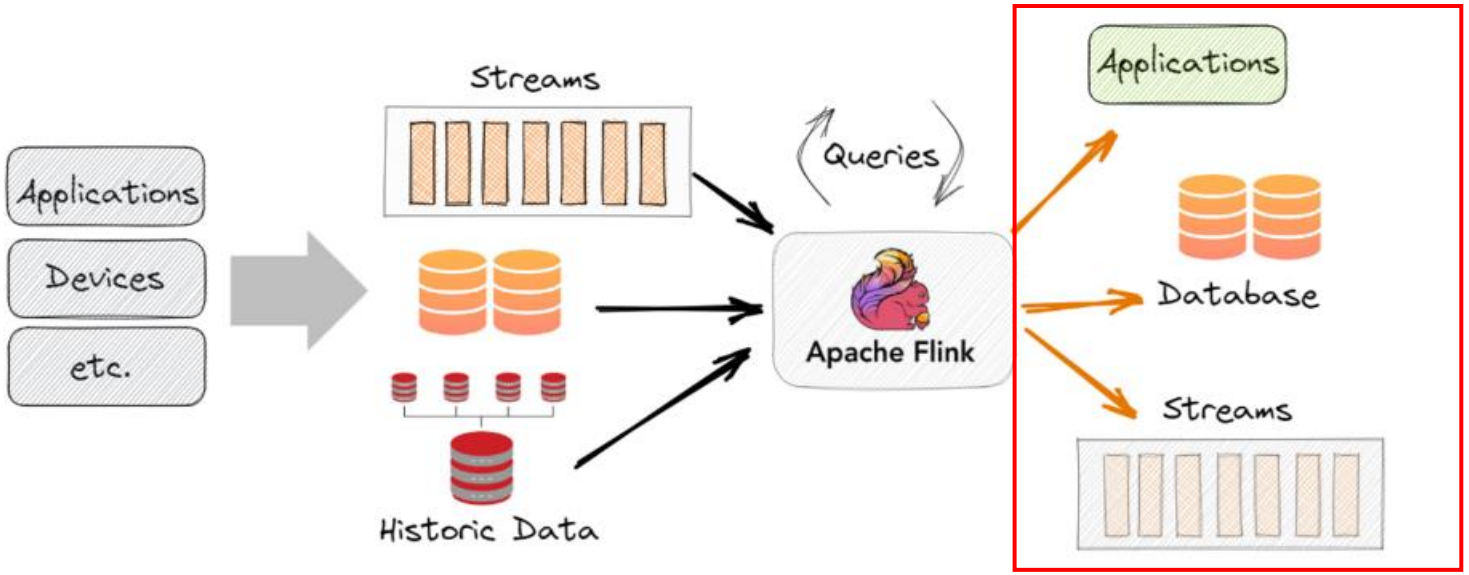
BEHAVIOR (2/3)



Data Processing

In the processing stage, Flink builds a pipeline that describes how to transform and analyze the data. Flink operators will be applied to the data streams. In addition, the system manages state, processes events based on their time of occurrence, and performs checkpoints to ensure fault tolerance.

BEHAVIOR (3/3)



Delivering Results

Once processed, the data is delivered to various destinations. Flink has connectors that allow integration with external systems, making it possible to write processed data to file systems, databases, messaging systems or external applications.

SIMILAR MIDDLEWARE

In the landscape of distributed systems, Apache Flink has several counterparts, two of them could be the following ones:

Apache Spark: Initially focused on batch processing, with micro-batch based streaming capabilities.

Latency: Moderate latency due to micro-batch streaming model.

Fault Tolerance: Uses data replication for fault tolerance.

Apache Storm: Designed specifically for real-time data processing, based on data streams.

Latency: Low latency, prioritizing real-time event processing.

Fault Tolerance: Guarantees fault tolerance through at-least-once processing.

Apache Flink: Designed for real-time data processing and for batch processing.

Latency: Low latency.

Fault Tolerance: Guarantees fault tolerance using checkpoints.

CONCLUSION

In conclusion, Apache Flink emerges as a comprehensive and powerful tool in the distributed systems and data processing landscape. From its ability to handle large volumes of data to its real-time processing model, Flink excels in several key areas.

Real-Time Processing, Efficient Batch Processing, Scalability and Fault Tolerance, horizontal scalability, its native checkpointing system ensures robust fault tolerance and the ability to Integrate with External Sources.

When considering these aspects, it is clear that Apache Flink not only addresses the complexities of large-scale data management, but also sets a standard for efficient real-time and batch processing. Its adoption translates into improved performance, increased responsiveness and a solid foundation for evolving distributed applications.