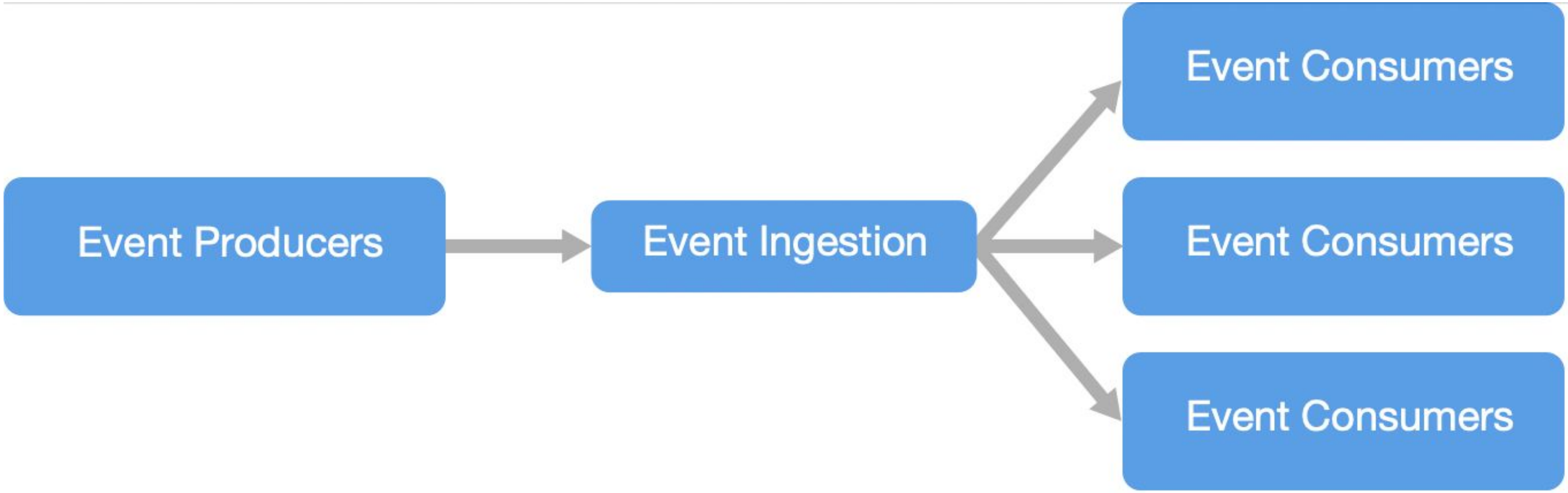
SE 464
Week 10

Architectures: Event-Driven, Streaming vs Batch, Leader-Follower

Event-Driven Architecture

Event Driven Architecture

- Definition: Architecture style that orchestrates behavior around events
- Core Concept
- Event producers and consumers
- Events as significant state changes

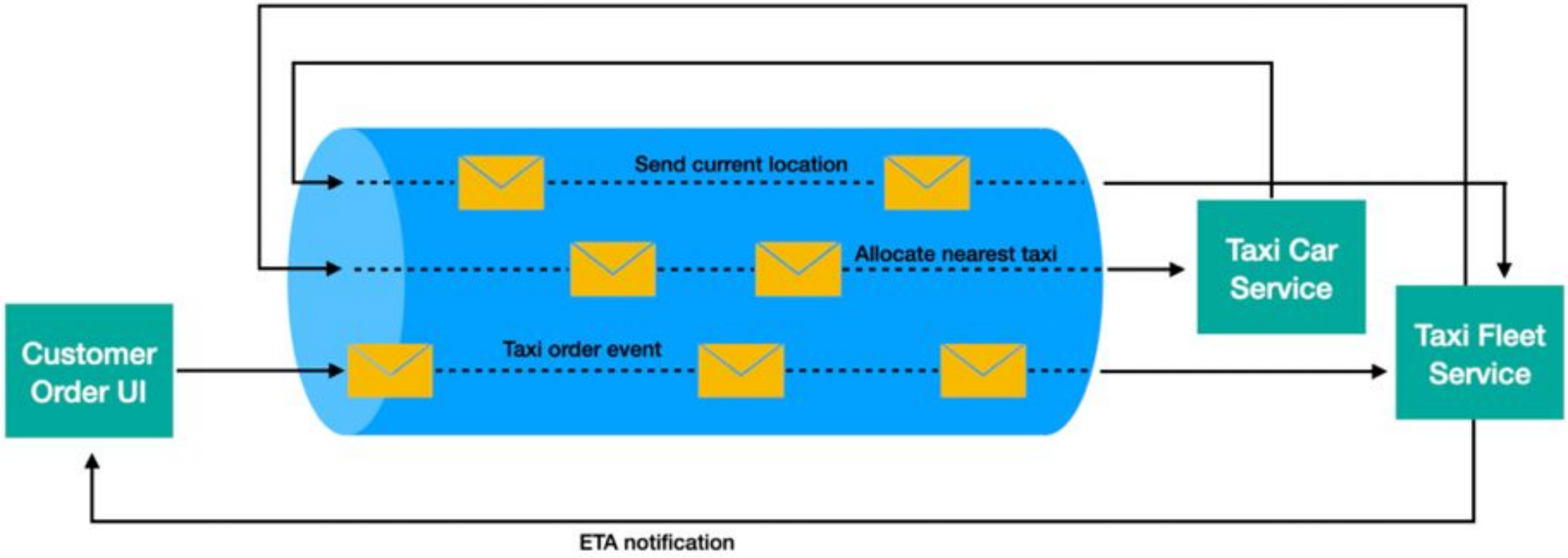

Event Producers
Event Ingestion
Event Consumers
Event Consumers
Event Consumers

Characteristics

- Event-Centric: Focuses on event production, detection, consumption
- Asynchronous Communication
- Decoupling: Event producers and consumers

Use Cases

- Real-time Data Processing
  - Financial transactions, monitoring systems
- Distributed Systems
  - Microservices, cloud-based applications
- Complex Event Processing
  - Fraud detection, recommendation systems



Send current location
Allocate nearest taxi
Taxi order event
Customer Order UI
Taxi Car Service
Taxi Fleet Service
ETA notification

Benefits

- Scalability: Handles fluctuating loads effectively
- Flexibility: Easy to add/remove components
- Responsiveness: Real-time reaction to events

Benefits

- Resilience: Reduces impact of a single component failure
- Maintainability: Simplifies maintenance and upgrades
- Improved User Experience
  - Faster, more interactive applications

Challenges

- Event Consistency
  - Ensuring event reliability and ordering
- Complexity in Event Handling
  - Designing effective event consumers
- Integration with Legacy Systems

Challenges

- Event Granularity
  - Balancing event size and informativeness
- Event Processing Logic
  - Distributing logic between producers and consumers

Challenges

- Error Handling and Recovery
- Monitoring and Debugging
  - Tracing and logging event flows
- Security Concerns
  - Protecting event data in transit and at rest

Streaming vs Batch Processing

Big Data

- Exponential Growth of Data
  - Critical for organizations
  - Driven by technological advancements
- Big Data Processing Cycle
  - Includes: Collection, Preparation, Input, Processing, Output/Interpretation, Storage
- Importance of Data Analytics
  - Essential for extracting valuable insights
  - Supports decision-making and strategy

Big Data

- Data Volume Increase
  - Estimated 50 trillion gigabytes (50 zettabytes) currently
  - Expected growth to 175 ZB by 2025
- Cloud Computing Influence
  - Facilitates data storage, processing, and analysis
  - Diverse Data Sources
  - Social, machine, transactional data
- https://youtu.be/bqqCTC9nQDY?si=HICUQa-5H2jA6-mu

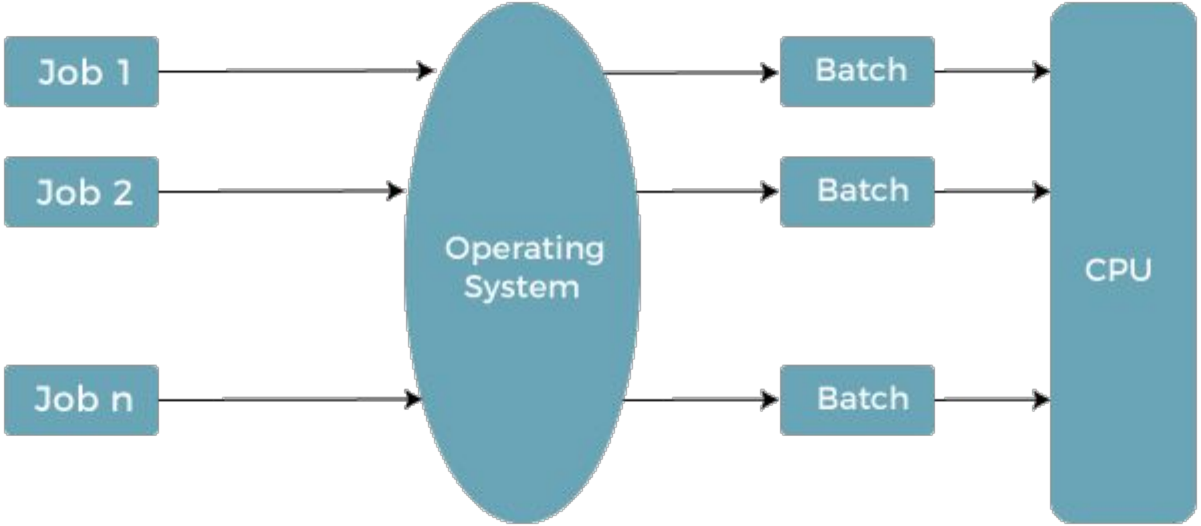
Challenges

- Security and privacy concerns
  - Managing Volume, Velocity, Variety, and Veracity
- Big Data Project Lifecycle
  - Stages: Problem Formulation, Data Collection, Data Preprocessing, Data Storage, Data Analysis, Data Visualization and Reporting, Data Quality
- Significance for Organizations
  - Enhancing growth and competitive edge

Batch Processing

- Processing large data chunks
- Data already stored over time
- Jobs start and finish
- Processed in sequential order
- Advantages
  - Efficient for large jobs
  - Works offline, conserving resources

Batch Processing



Batch Processing Use Cases

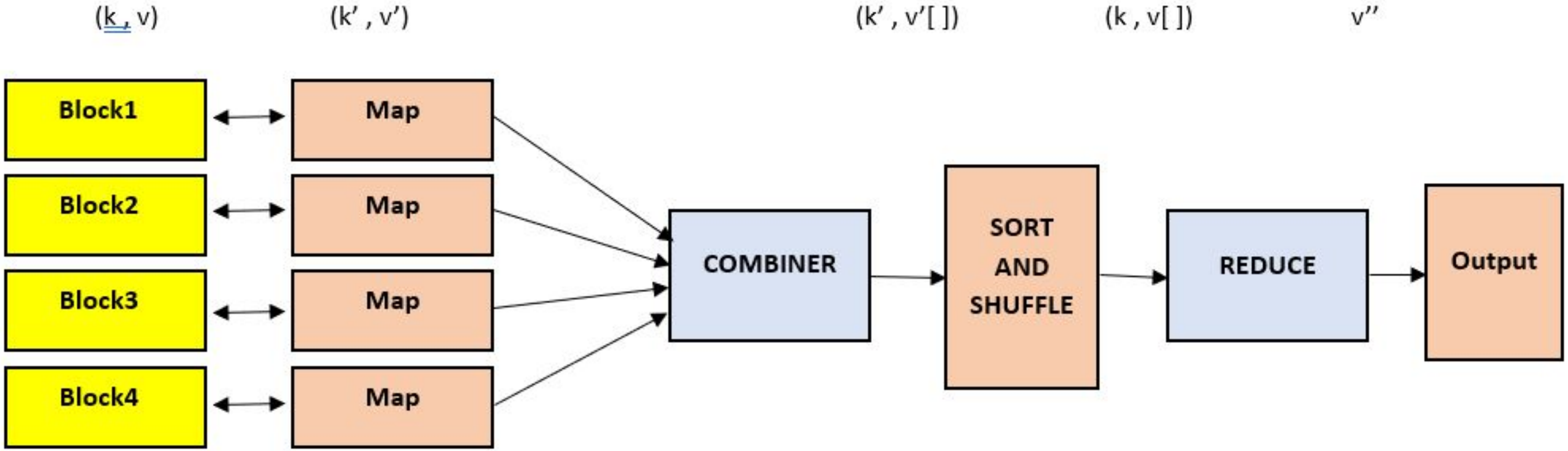
- Typical Applications
  - Financial transactions analysis
  - Payroll, billing, customer orders
- Processing Characteristics
  - High latency (minutes to hours)
  - Suitable for non-time-sensitive jobs
- Technology Examples
  - Hadoop
  - Cloudera (Hadoop distribution)

Batch Processing in Big Data

- Crucial for structured, large-volume data
- Lifecycle Stages
  - Data collection and preparation
  - Data storage and analysis
  - Result generation and interpretation
- Challenges
  - Managing large data volumes
  - Ensuring data quality and security

MapReduce

- Programming model for large data volumes
- Parallel processing of independent tasks
- Process
  - Input splitting
  - Map phase
  - Shuffle and sort phase
  - Reduce phase
- Key Features
  - Divide and conquer method
  - Key-value pair transformation


(k , v)
(k' , v')
(k' , v'[ ])
(k , v[ ])
v''
Block1
Block2
Block3
Block4
Map
Map
Map
Map
COMBINER
SORT AND SHUFFLE
REDUCE
Output

Key Steps in MapReduce

- Input Splitting: Divides data into manageable pieces
- Map Phase: Applies map function to each piece
- Shuffle and Sort: Organizes output of map phase
- Reduce Phase: Aggregates and processes shuffled data
- Output
  - Final result compiled from reduce phase
- Advantages
  - Scalability
  - Fault tolerance

Applications of MapReduce

- Large-scale data analysis
- Distributed computing tasks
- Technological Implementation
  - Hadoop: Popular framework implementing MapReduce
- Challenges and Considerations
  - Handling large data volumes
  - Optimizing performance

Stream Processing

- Real-time data processing
- Characteristics
  - Low latency
  - Continuous data flow
- Importance
  - Immediate insights
  - Rapid decision-making

Stream Processing

- Data Handling
  - High-velocity data streams
  - Continuous data acquisition
- Applications
  - Fraud detection
  - Online recommendations
  - Monitoring systems

Stream Processing Challenges

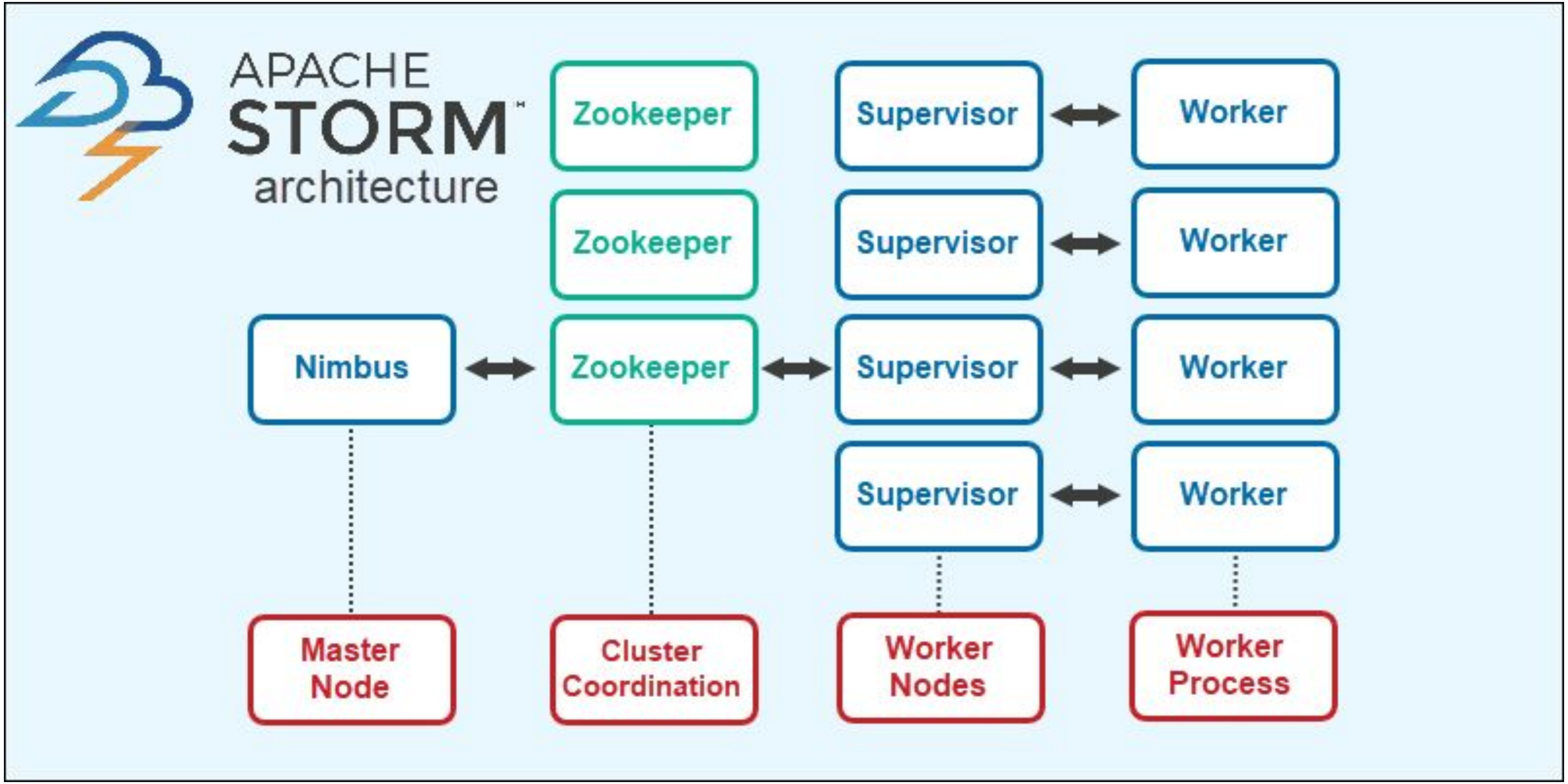
- Data Volume and Velocity
  - Handling large, fast-moving data streams
- Real-time Analysis Requirements
  - Immediate data processing and analysis
- Resource Management
  - Efficient use of memory and processing power

Apache Storm

- Real-time computation system
- Processes data streams
- Key Components
  - Spouts: Data stream sources
  - Bolts: Data processing units
- Features
  - Scalability
  - Fault tolerance

Apache Storm in Practice

- Use Cases
  - Real-time analytics
  - Distributed RPC (Remote Procedure Calls)
- Strengths
  - Easy to set up and operate
  - Integrates with various data sources
- Limitations
  - Complex state management
  - Limited data windowing capabilities


APACHE
STORM
architecture
Zookeeper
Zookeeper
Zookeeper
Nimbus
Supervisor
Supervisor
Supervisor
Supervisor
Worker
Worker
Worker
Worker
Master Node
Cluster Coordination
Worker Nodes
Worker Process

Comparing Batch vs Stream

- Key Differentiators
  - Purpose
  - Data handling
  - Processing requirements
- Batch for large data volumes
- Stream for real-time analytics

Comparing Batch vs Stream

- Batch Processing Paradigm
  - MapReduce
  - Offline analysis
- Stream Processing Paradigm
  - Stream processors
  - Online, continuous analysis

Comparing Batch vs Stream

- Batch Processing
  - Scheduler-based job execution
  - Ideal for historical data analysis
- Stream Processing
  - Continuous data processing
  - Suited for real-time decision making

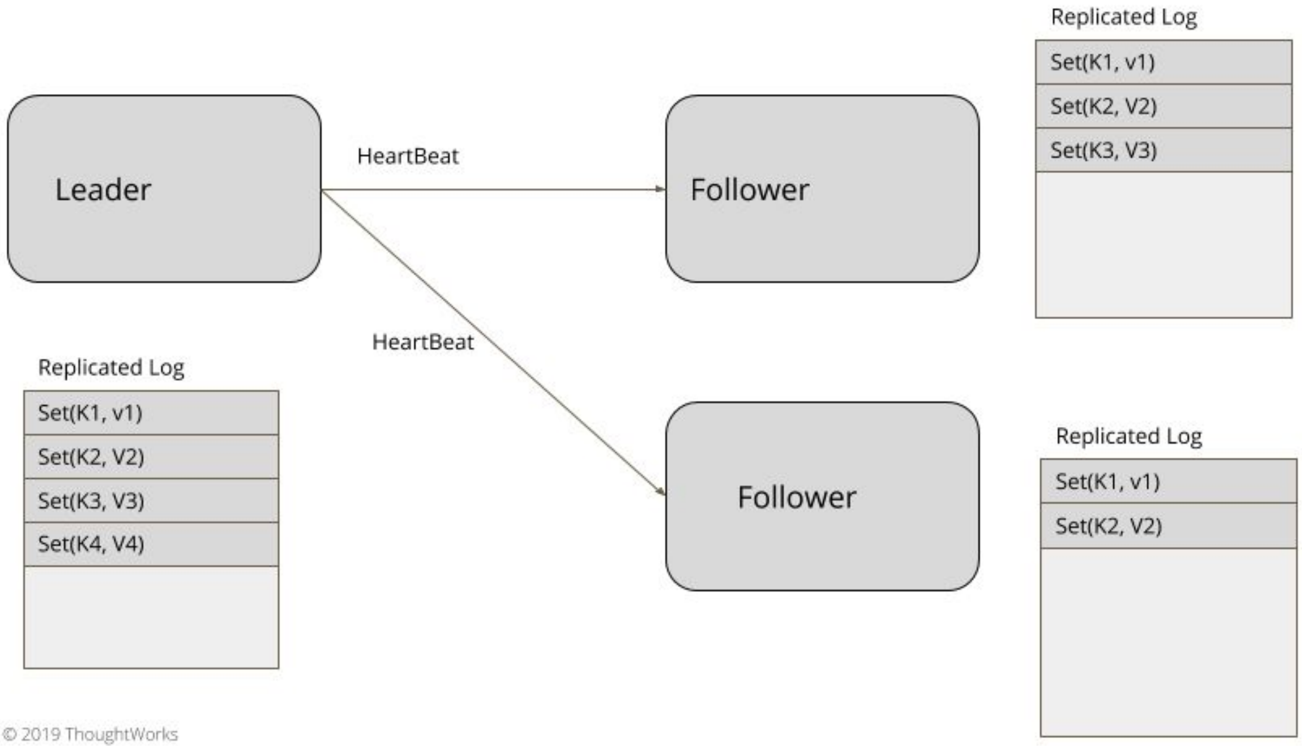
Leader-Follower Architecture

Leader Follower Architecture

- Definition: A concurrency pattern for distributed systems
- Core Concept
  - Dynamic role assignment among threads
  - Leader handles events, followers await their turn

Components

- Thread Management
  - Single active leader thread
  - Pool of passive follower threads
- Event Handling
  - Leader demultiplexes and dispatches events


Leader
HeartBeat
Follower
HeartBeat
Follower
Replicated Log
Set(K1, v1)
Set(K2, V2)
Set(K3, V3)
Set(K4, V4)
Replicated Log
Set(K1, v1)
Set(K2, V2)
Set(K3, V3)
Replicated Log
Set(K1, v1)
Set(K2, V2)

Use Cases

- High-Volume Transaction Systems
  - OLTP, server-based applications
- Resource-Constrained Environments
  - Where thread overhead is a concern
- Complex Event Processing
  - Systems requiring real-time responsiveness

Benefits

- Resource Efficiency
  - Reduced thread count
  - Lowered system overhead
- Scalability
  - Handles fluctuating workloads effectively
- Flexibility
  - Dynamic role switching enhances adaptability

Benefits

- Simplified Synchronization
  - Reduced complexity in thread coordination
- Improved Throughput
  - Efficient event handling and processing
- Fault Tolerance
  - Leader failure leads to a new leader election

Challenges

- Complexity in Design and Maintenance
  - Requires careful coordination of thread roles
- Bottlenecks
  - Leader thread can become a performance bottleneck
- Debugging and Monitoring
  - Tracing issues in dynamic role assignments

Challenges

- Event Consistency and Ordering
  - Ensuring reliable event handling
- Failure Recovery Mechanisms
  - Handling leader thread failures
- Integration with Legacy Systems
  - Adapting the pattern to existing infrastructures

Leader Election and Synchronization

- Leader Election Mechanisms
  - Techniques for assigning the leader role
- Thread Synchronization
  - Ensuring smooth role transitions
- Coordination with External Systems
  - Use of tools like Zookeeper for leader management

Case Study: Apache Kafka

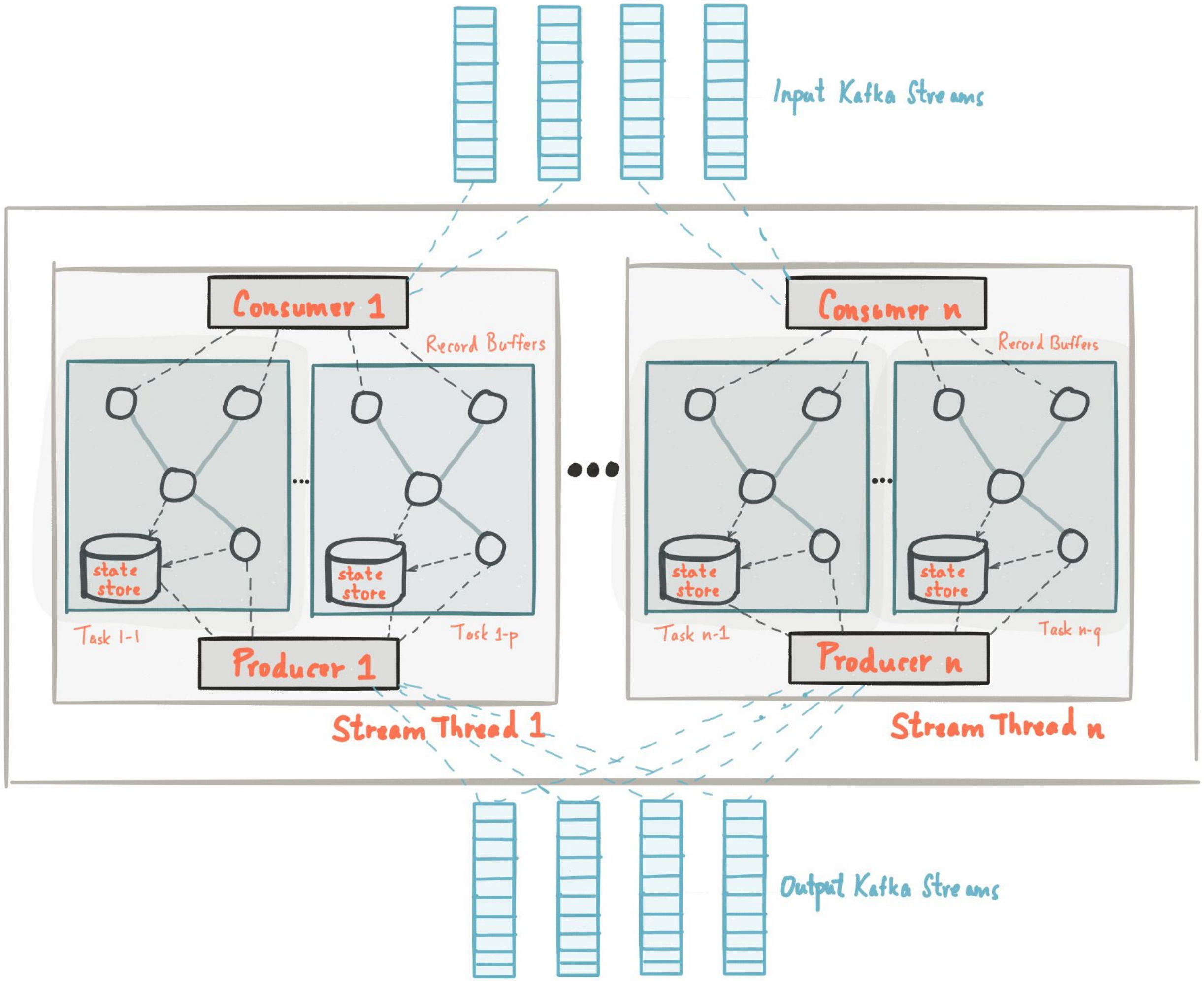
- Distributed streaming platform
- Use in handling real-time data feeds

Leader-Follower in Kafka

- Basics of Kafka Architecture
  - Producers, Consumers, Brokers
- Leader-Follower in Brokers
  - Partition leadership among brokers

Kafka Architecture

- Multiple brokers
- Leader and follower partitions
- Role of Zookeeper
  - Coordinates broker leadership


Input Kafka Streams
Consumer 1
Record Buffers
state store
Task 1-1
Task 1-p
Producer 1
Stream Thread 1
Consumer n
Record Buffers
state store
Task n-1
Task n-q
Producer n
Stream Thread n
Output Kafka Streams

Data Flow in Kafka

- Data Write Flow Diagram
  - Producers to leader partitions
  - Replication to followers
- Data Read Flow
  - Consumers reading from leaders or followers

Leader Election in Kafka

- Mechanism of Leader Election
  - Broker failure scenarios
  - Zookeeper's role in electing new leaders
- Consistency Guarantees
  - How Kafka ensures data consistency during leadership changes

Replication and Fault Tolerance

- Replication Strategy
  - In-sync replica sets
- Fault Tolerance
  - Handling broker downtimes
  - Ensuring data availability and integrity

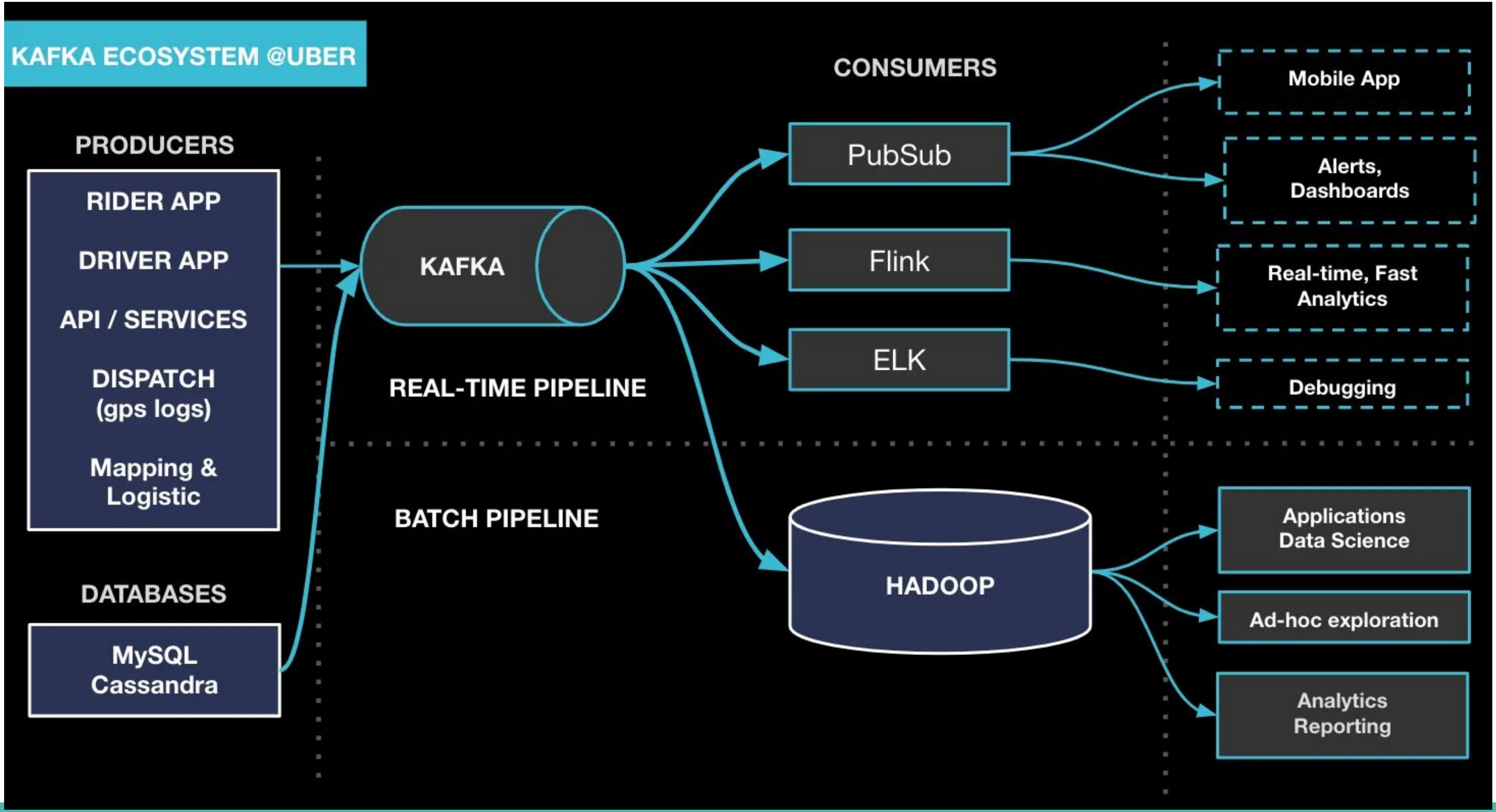
Challenges

- Leader-Follower Impact on Throughput
  - Balancing load between leaders and followers
- Latency in Leader-Follower Replication
  - Factors affecting latency
- Bottlenecks in Leader Performance
  - High traffic and large data volumes
- Handling Follower Lag
  - Strategies to keep followers in sync

Example Use Case for Kafka

- Scenario: E-commerce Platform Analytics
  - Real-time tracking of user behavior and transactions
  - Aggregation of data for instant analytics
- Kafka’s Role
  - High-throughput data ingestion from various sources
  - Reliable data streaming to analytics engines
  - Ensures data consistency and availability
  - Facilitates rapid data processing and decision making

Another Real-Life Example



Sources

- https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/event-driven
- https://www.dre.vanderbilt.edu/~schmidt/PDF/lf.pdf
- https://kafka.apache.org/21/documentation/streams/architecture.html