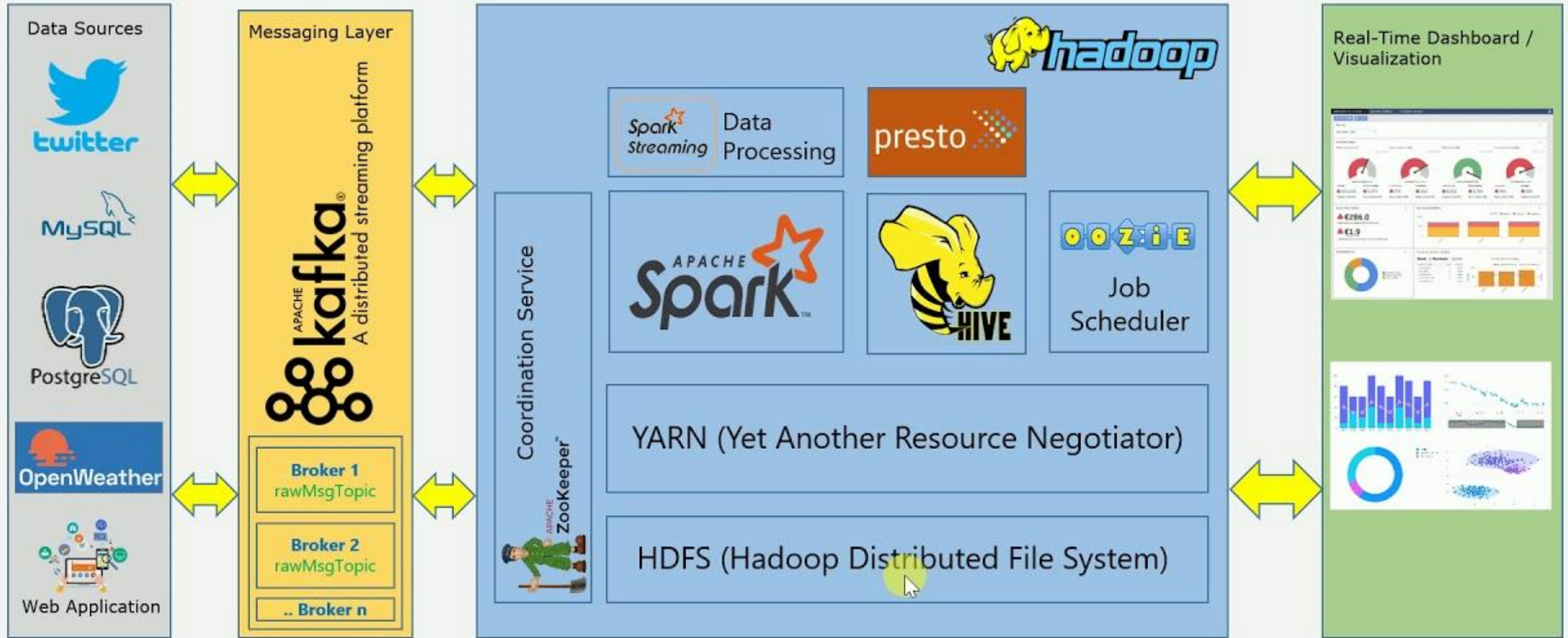


# Real-Time Dashboard Reference Architecture





# Spark Streaming

*Sunbeam Infotech*



# Spark Structured Streaming

- Spark Structured Streaming consider dataframe to be unbounded (infinitely growing).
- Transformations & Actions
  - Transformations are same as spark dataframe. Few transformations are not yet implemented.
  - Action is starting the stream & print results.
- Input sources
  - socket, rate, files, flume, kinesis, kafka
- Output sinks
  - console, memory, files, flume, kafka, foreach



# Spark Structured Streaming

- Output modes

- Every mode is not supported for every type of query.
- append: output result of current micro-batch is available. (not supported for aggregate operations).
- complete: complete result including prev result & current micro-batch result is available.
- update: only results modified in current micro-batch are available.

- Triggers

- By default, micro-batches are processed one after another.
- Trigger can specify time duration after which each batch is to be processed.

→ moment current batch is completed,  
next batch processing begins.  
(collect data + operations + output.)

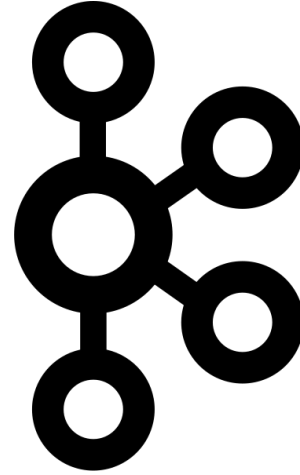
- Event time processing

- Time at which event is generated at source, is "event time".
- Can process out-of-order data.
- Watermark feature is used define for how much time data should be considered (how much time should be wait before processing data).

for continuous processing:

Trigger, Continuous()



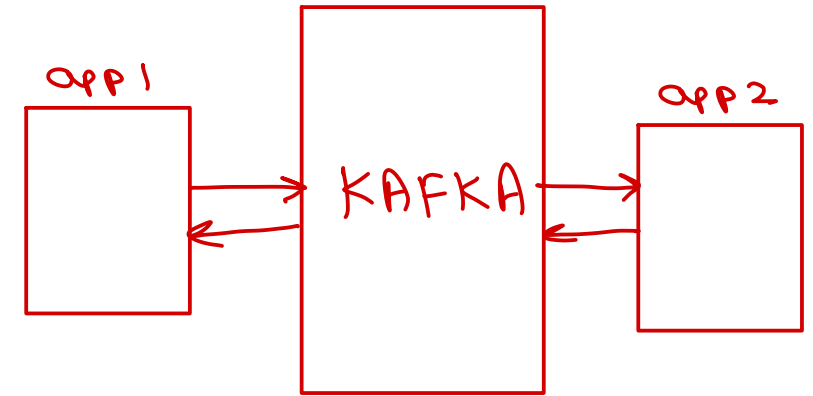


# Apache Kafka

*Sunbeam Infotech*

# Introduction

- Kafka is a distributed messaging system.
- Developed at LinkedIn and open sourced in 2011.
- Implemented in Scala.
- Used by LinkedIn, Twitter, Uber, Yahoo, airbnb, ...



- <https://www.slideshare.net/jhols1/kafka-atlmeetuppublicv2> ✓
- <https://www.slideshare.net/AkashVacher/kafka-meetup-ppt-1> ✓



# Apache Kafka

- Advantages

- High throughput
- Disk based
- Durable
- Scalable (*horizontal scaling*)
- Low latency
- Finite retention
- Strong ordering
- Exact once delivery

- Applications

- Stream processing
- Log processing
- Notifications
- Customer tracking



# Kafka Terminologies

- Broker

- Each node in kafka cluster is called as "kafka broker".
- Each broker have its own id (configured in \$KAFKA\_HOME/config/server.properties).
- Broker is JVM process -- Kafka -- running on port 9092.

- Topic

- The messages in kafka are logically divided under topics.
- The topics are configured as per need of application.

- Partitions

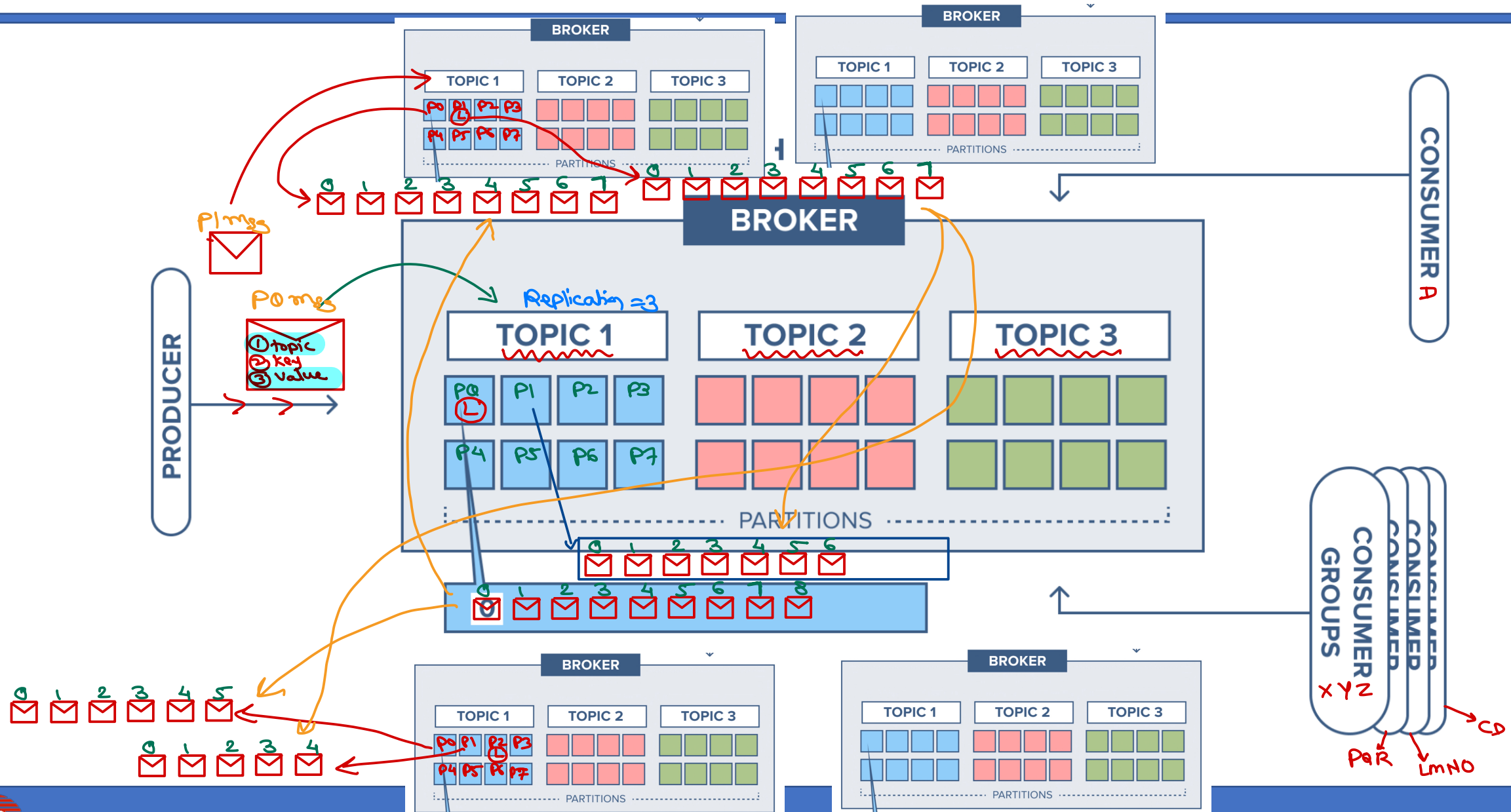
- Topics are physically divided under partitions.
- The messages data is actually stored into log files (.log) for each partition under /tmp/kafka-logs (default directory).
- Each partition each message is numbered (indexed) from 0,1,2,... called as "offsets".
- Partitions have replication factor. If replication factor is 3, partition data will be copied on two more brokers.
- In case of replication of partitions, one of the partition is leader while others will be followers/replicas. These replicas are always try to be in sync with leader.
- The replicas which are not too behind the leader and said to "in-sync replicas" (ISRs).

→ replication  
→ Partitions  
→ retention

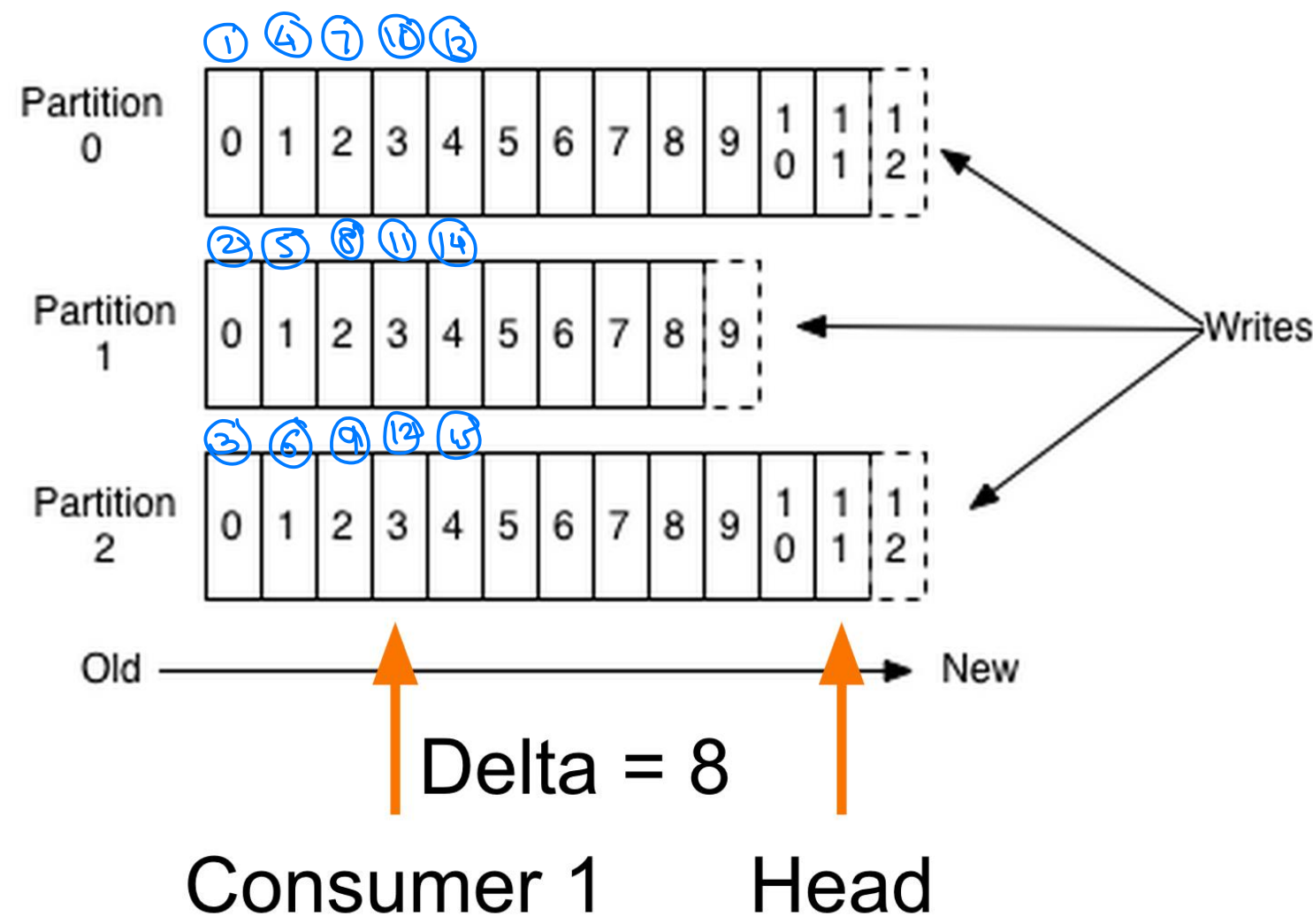




# Kafka Terminologies



# Kafka Topic



# Kafka Terminologies

- Producer

- Client process of kafka which send messages to the broker, is said to be producer.
- These client processes can be implemented in Java, Python, C/C++, ...
- Producer publish data to the topics, which is sent to partitions, so that partitions are load balanced. It may use round-robin algorithm or may do key-based division.
- Producer always write data to the leader of the partition. → *high code*

- Consumer

- Client process of kafka which receive messages from the broker, is said to be consumer.
- These client processes can be implemented in Java, Python, C/C++, ...
- Clients subscribe to the kafka topics.
- It will read the data from any of the replica, which have data required for that client.
- The data in a partition is guaranteed to be received in sequence (of writing). However data in different partitions is not guaranteed to read in sequence of writing.




# Kafka Terminologies

- Consumer groups

- Each topic can have one or more consumers -- divided into one or more consumer groups.
- Consumer group -- kafka ensure than only one consumer from ~~a~~ consumer group will get a message. each


- if there are three consumer groups as follows:

- G1 -- C1a, C1b, C1c
  - G2 -- C2a, C2b
  - G3 -- C3a, C3b, C3c, C3d
- 


- If message "m0" is received into kafka, it will be received by exactly one consumer from each group.

- Possible organization of consumers in a group:

- example1:

- G1 -- C1
  - G2 -- C2
  - G3 -- C3
  - each message is received by each consumer.
  - broadcast system.
- 

- example2:

- G1 -- C1a, C1b, C1c
  - each message is received by only one consumer.
  - parallel processing of records.
- 



# Kafka Installation

- Download and extract Kafka.
- In `~/.bashrc`
  - `export KAFKA_HOME=/path/to/kafka`
  - `export PATH=$KAFKA_HOME/bin:$PATH`
- In `$KAFKA_HOME/config/server.properties`
  - Uncomment line → `listeners=PLAINTEXT://:9092`
- Start Kafka
  - `terminal> zookeeper-server-start.sh $KAFKA_HOME/config/zookeeper.properties`
  - `terminal> kafka-server-start.sh $KAFKA_HOME/config/server.properties`



# Using Kafka

- Create topic
  - `kafka-topics.sh --zookeeper localhost:2181 --create --topic spark03 --replication-factor 1 --partitions 2`
  - Verify: `ls /tmp/kafka-logs`
- List topics
  - `kafka-topics.sh --zookeeper localhost:2181 --list`
- Write to topic (producer)
  - `echo "one" | kafka-console-producer.sh --broker-list localhost:9092 --topic spark03`
- Read from topic (consumer)
  - `kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic spark03`
  - `kafka-console-consumer.sh --bootstrap-server localhost:9092 --partition 0 --offset 2 --topic spark03`
  - `kafka-console-consumer.sh --bootstrap-server localhost:9092 --from-beginning --topic spark03`
- Consumer Groups
  - `kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic spark03 --group 1`
  - `kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic spark03 --group 1`
  - `kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic spark03 --group 2`

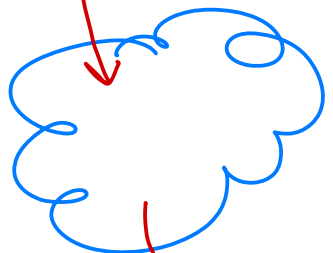


# Spark Kafka Integration

- Spark dataframes are divided into the partitions. When dataframe created from kafka, one dataframe partition is mapped to one kafka partition.
- Spark dataframe partitions are on workers i.e. each worker will read corresponding partition from kafka topic partition. Each worker will also maintain its own offset.
- When a streaming query is executed, a consumer is created.
  - one query(job) = one consumer group --> each group will have consumers equal to number of workers used to read from kafka.
- If two different jobs are processing two different kafka topics, number of dataframe partitions in each job is equal to number of kafka partitions in that topic. All tasks (threads) processing the partitions corresponding to one topic will be in one consumer group.
- If two different jobs are processing same kafka topic (for doing different processing), still there will be two different consumer groups. Each group will contain number of consumers equal to number of dataframe partitions (which in turn equal to number of kafka partitions). Note that data will be read twice from each kafka partition to create two different dataframe partitions (one for each job).

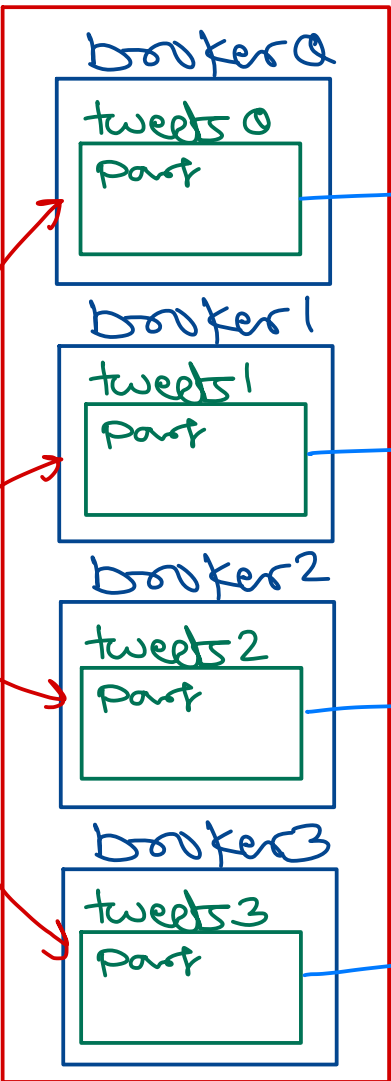


Twitter

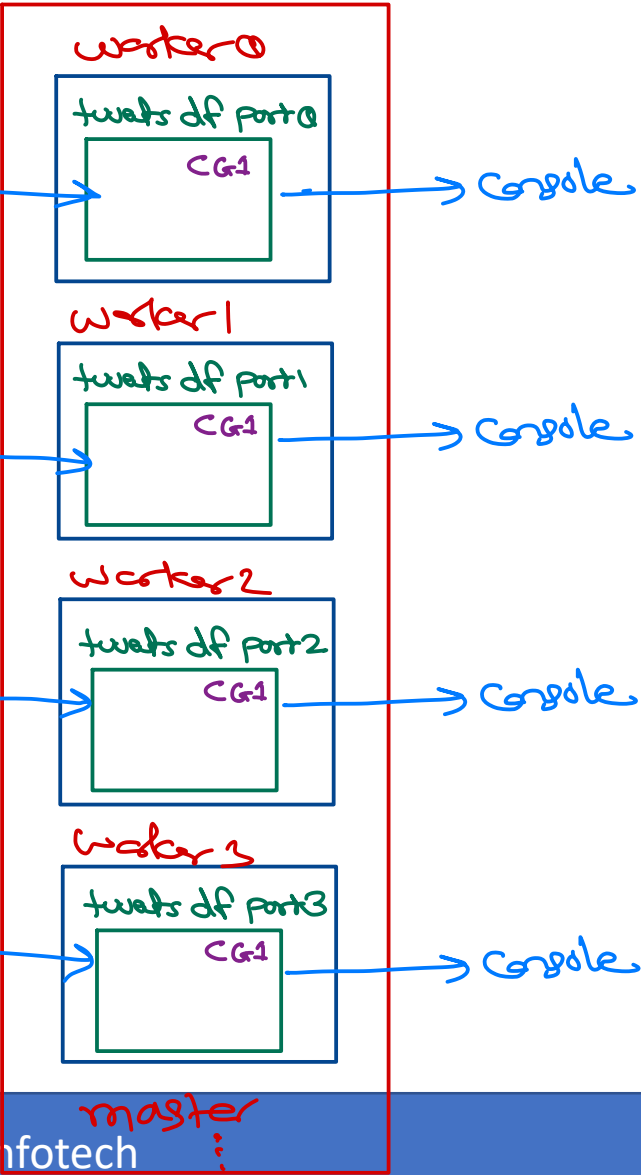


twitter-kafka-producer.py  
=  
=

kafka cluster

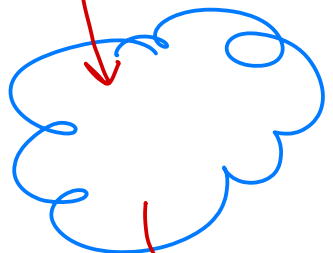


spark cluster



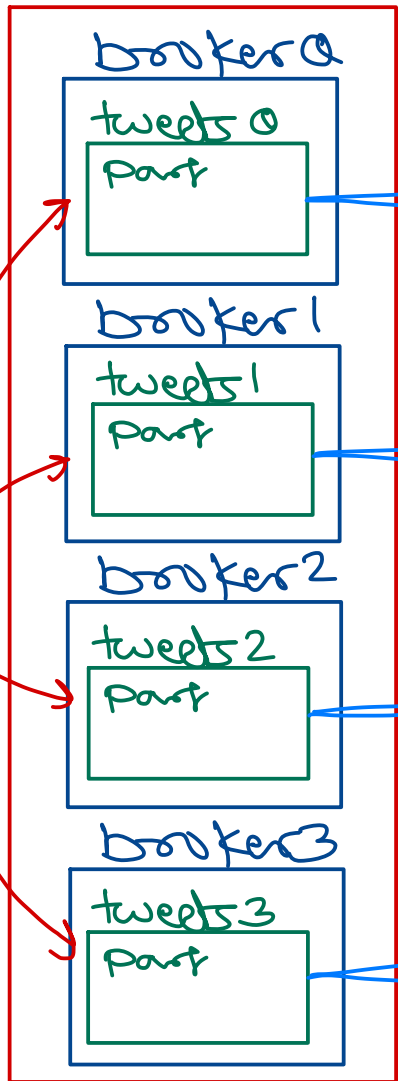


Twitter

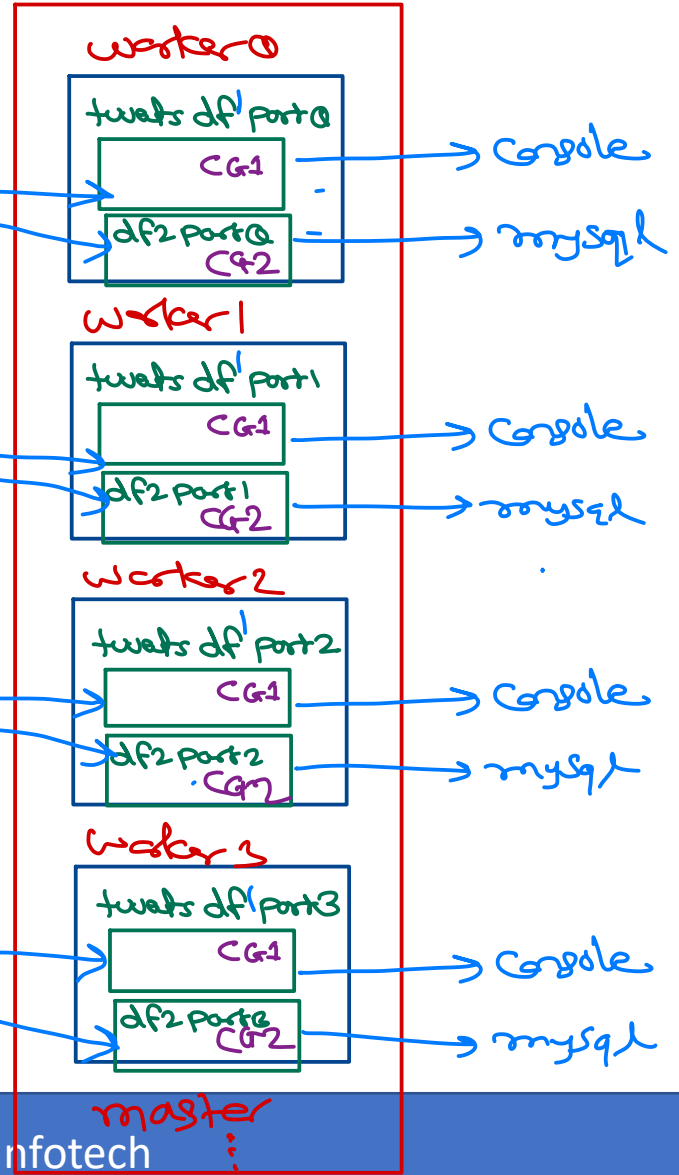


twitter-kafka-producer.py  
=  
=

### kafka cluster



### spark cluster





Thank you!

*Nilesh Ghule <nilesh@sunbeaminfo.com>*

