



Apache Kafka

Melanga Dissanayake

SEPTEMBER 17, 2015

About Me

Melanga Dissanayake

Senior Software Engineer - EPAM System (Shenzhen)

- Over 10 years of enterprise application development experience
- Focused on financial domain

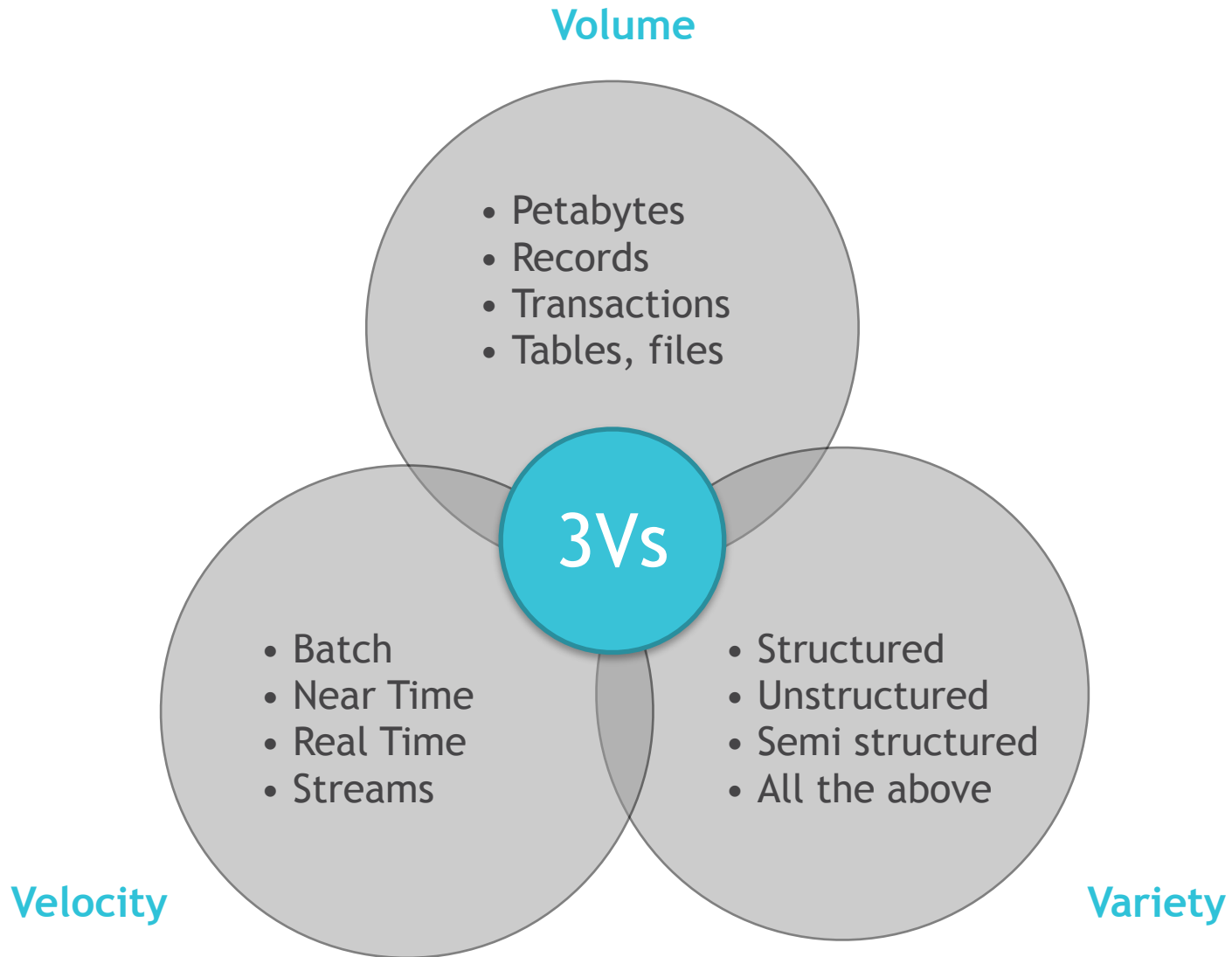
Big data is like teenage sex:
everyone talks about it,
nobody knows how to do it,
everyone thinks everyone else is
doing it, so everyone claims they
are doing it...

(Dan Ariely)

What is Big Data?



What is Big Data?



What is Big Data?

- Traditional Queue
- Website Activity Tracking
- Metrics - Operational data
- Log Aggregation
- Stream Processing
- Event Sourcing
- Commit Log

Where do we put Big Data?

- Traditional Database?
- Flat files?

HDFS

- HDFS - Hadoop Distributed File System
 - Highly fault-tolerant
 - Java based file system
 - Runs on commodity hardware
 - Concurrent data access (Coordinated by YARN)

How do we put these Big Data

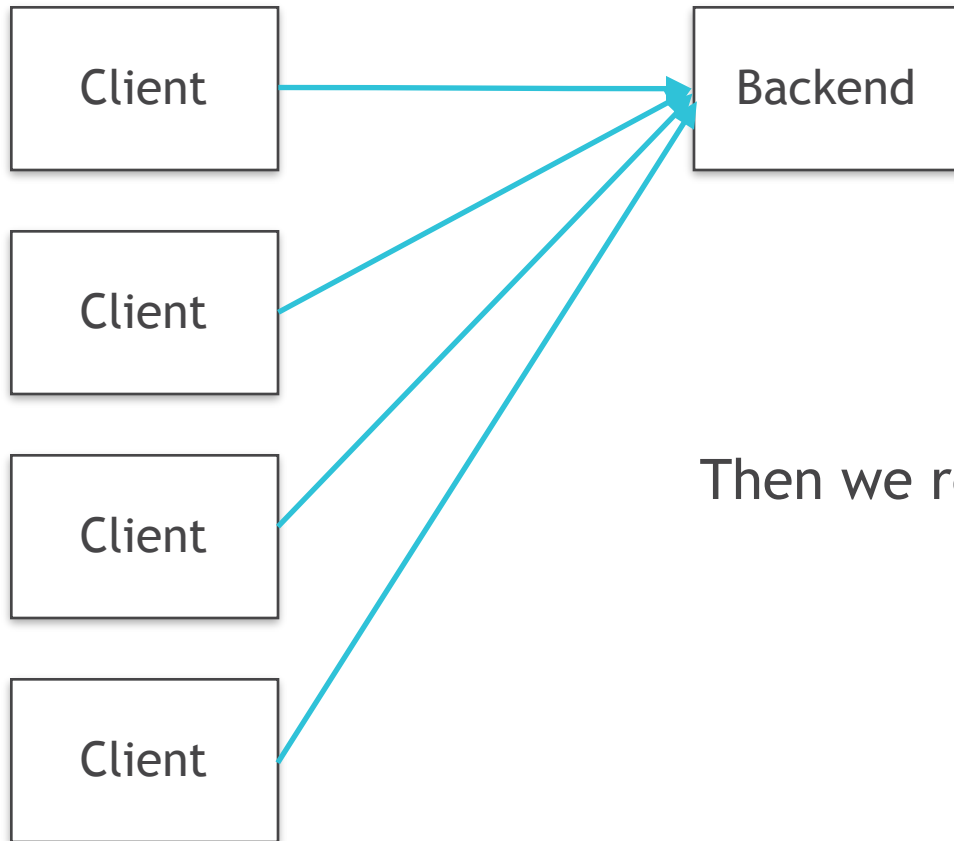
- CSV file dump
- ETL
- Other messaging systems (ActiveMQ, RabbitMQ, etc)

Data pipeliens



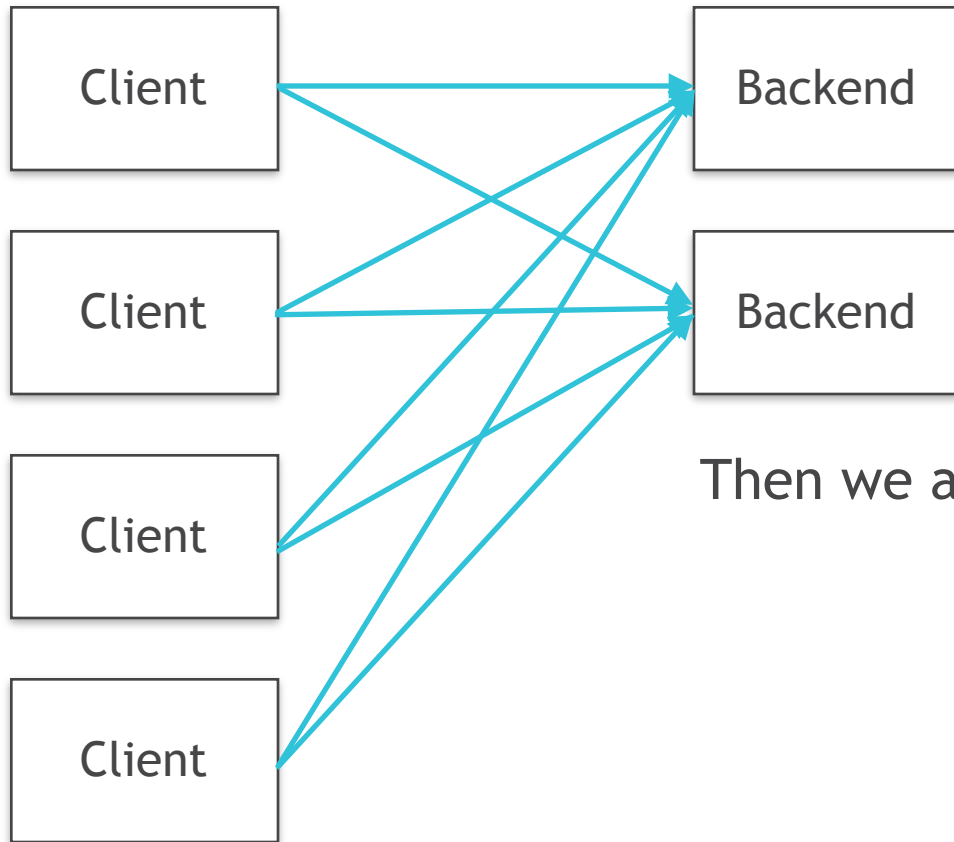
Data pipeline starts like this

Data pipeliens



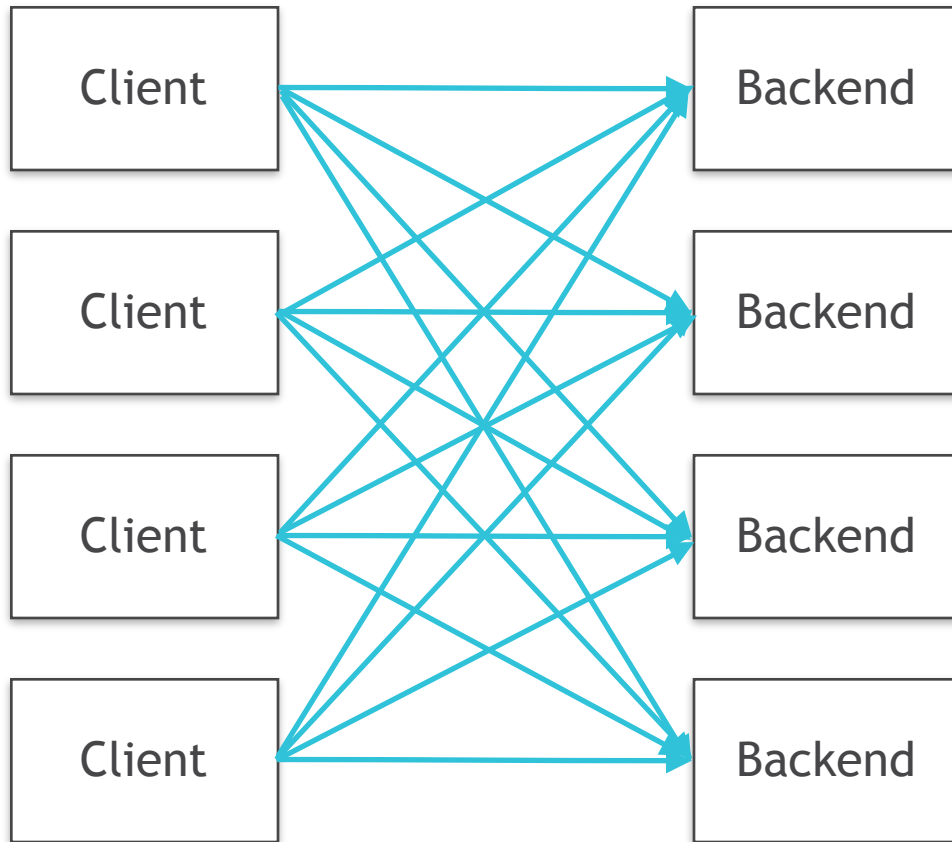
Then we reuse them

Data pipeliens



Then we add more backends

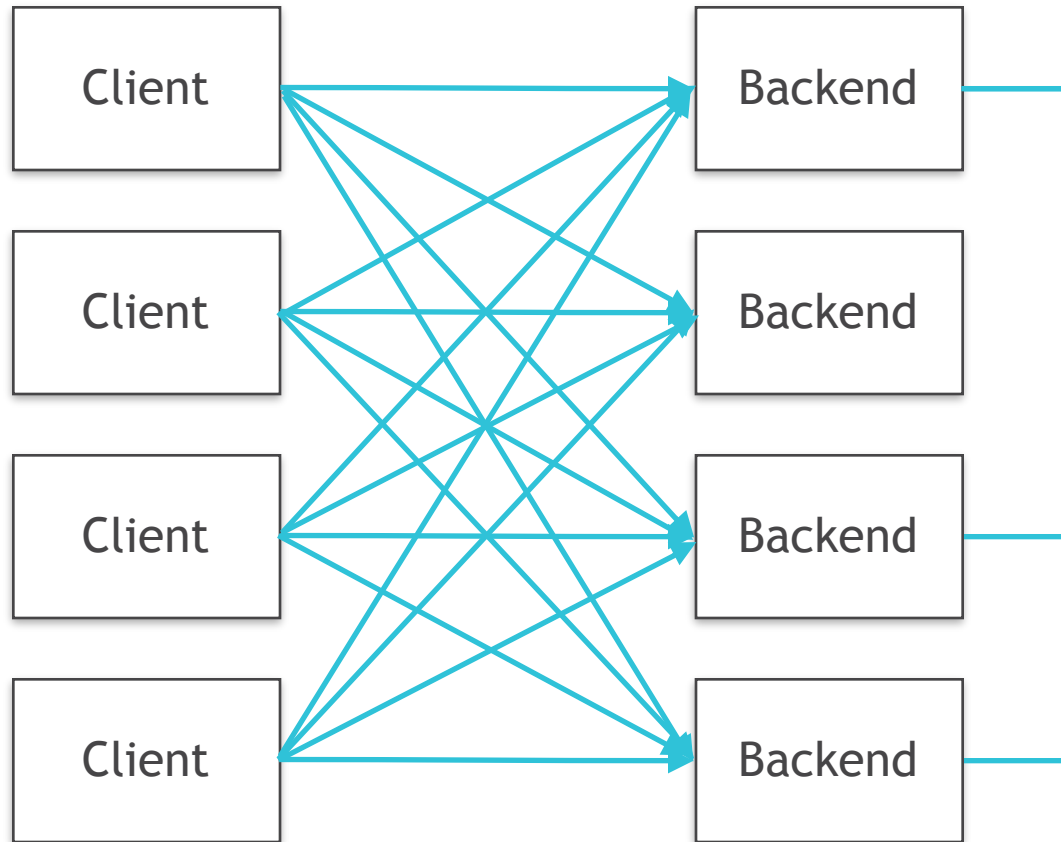
Data pipelines



Then it starts to look like this

Source: Cloudera

Data pipelines

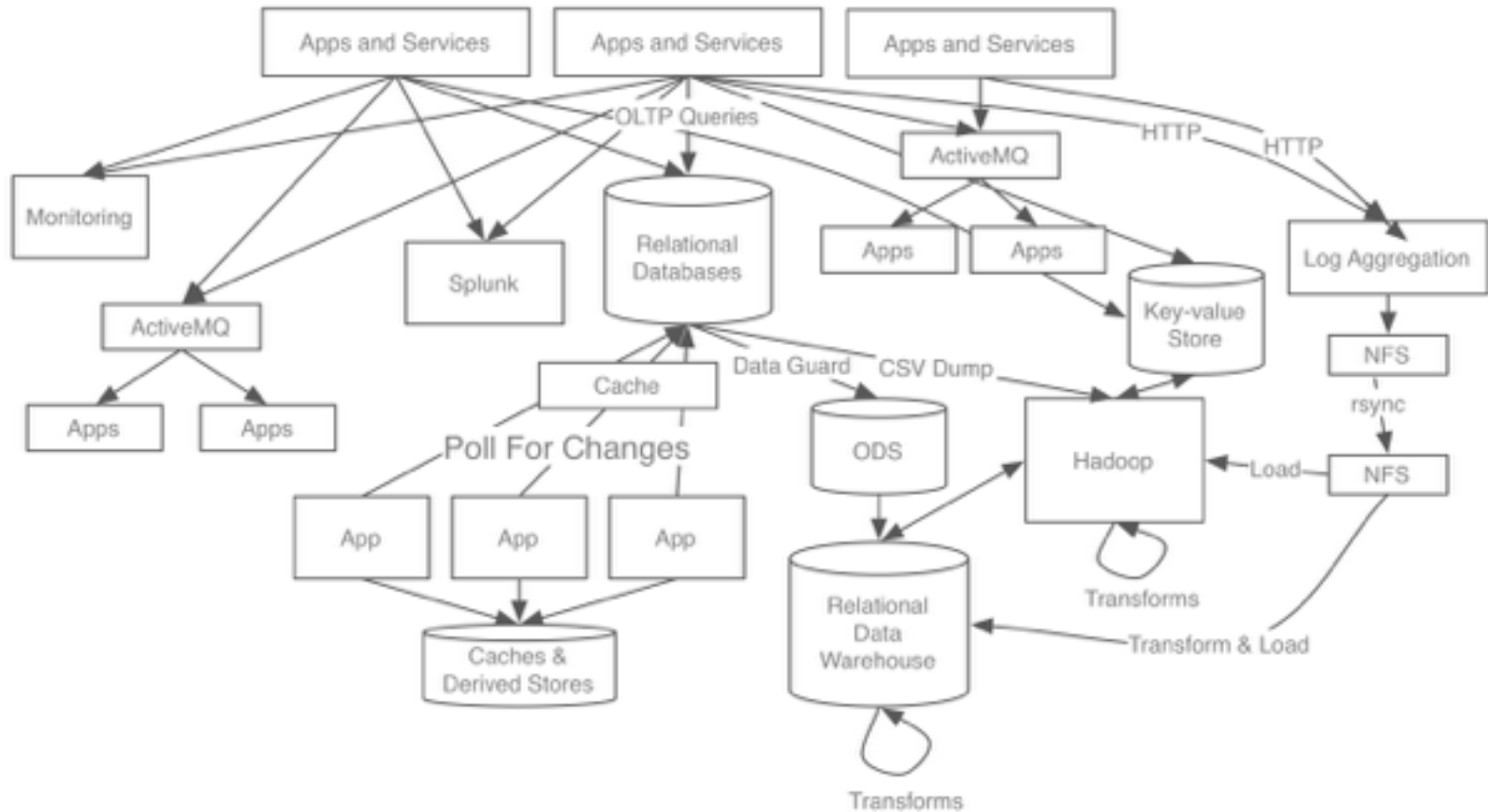


with may be some of this

Source: Cloudera

Data pipelines

ended up having



Source: LinkedIn


What is Apache Kafka



Apache Kafka is an open-source message broker rethought as a distributed commit log.

Developed using Scala and heavily influenced by transaction logs.

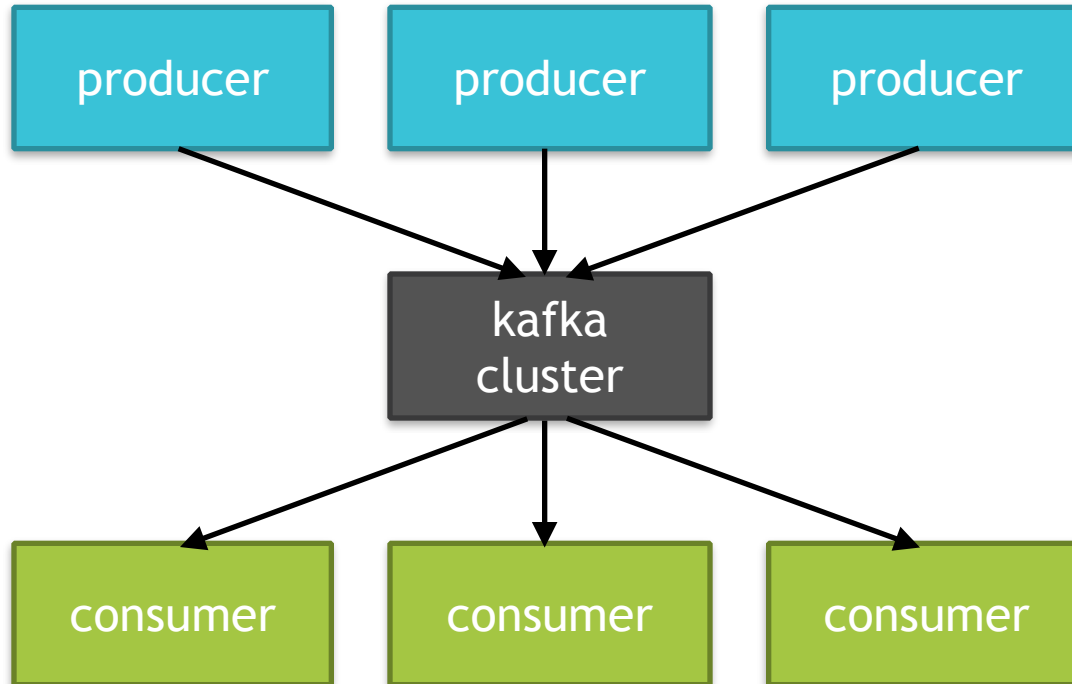
Little bit of history

Apache Kafka was initially developed by **LinkedIn**  to pipeline the data across various internal systems.

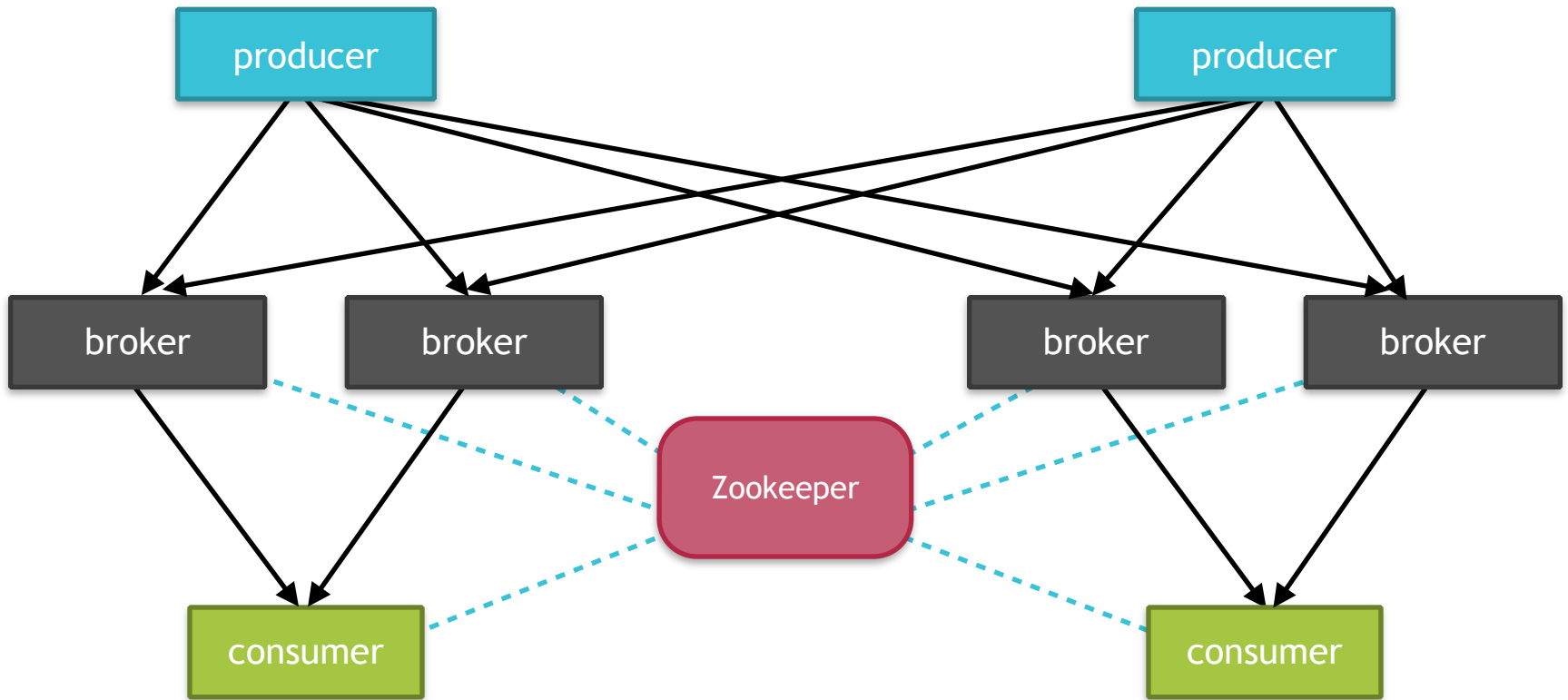
Developed as a an internal project in early 2011 project was released under open-source license.

Graduated from Apache Incubator on October 2012.

Kafka in nutshell



Kafka Architecture

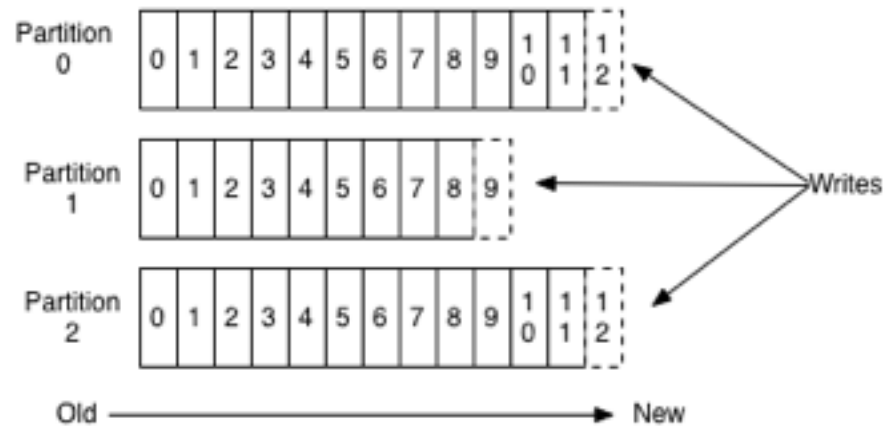


Kafka Architecture

- Communication between all nodes based on high performance simple binary API over TCP
- Runs on as a cluster of brokers which is a one or more servers in this case
- High performance low level APIs for producer/consumer
- REST API via Kafka REST Proxy

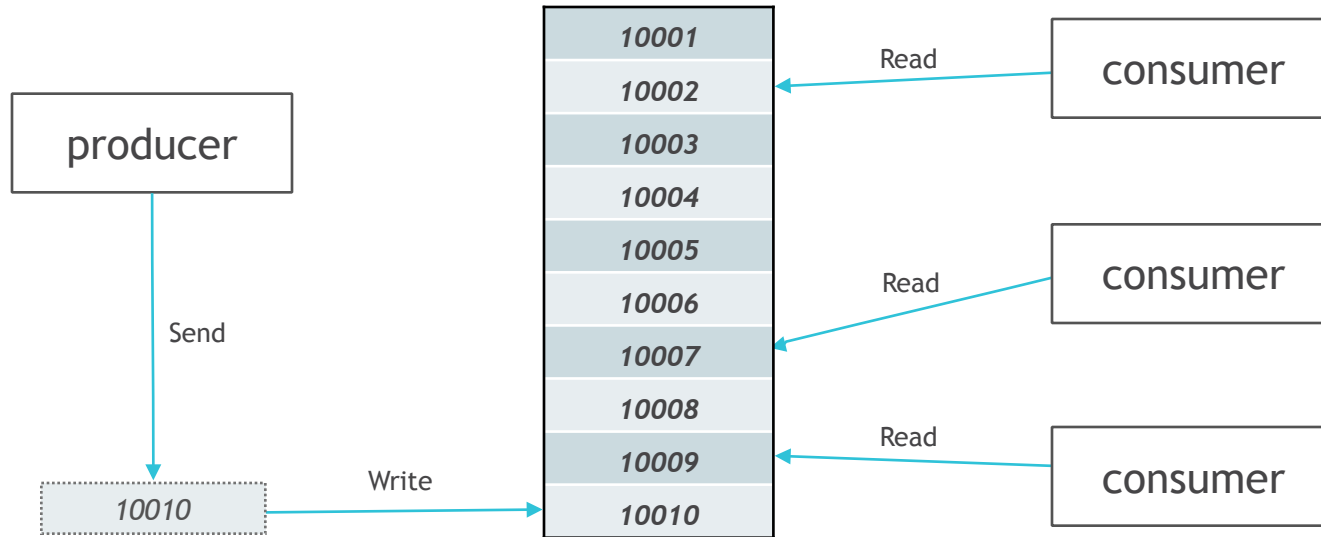
Topics & Partitions

Anatomy of a Topic



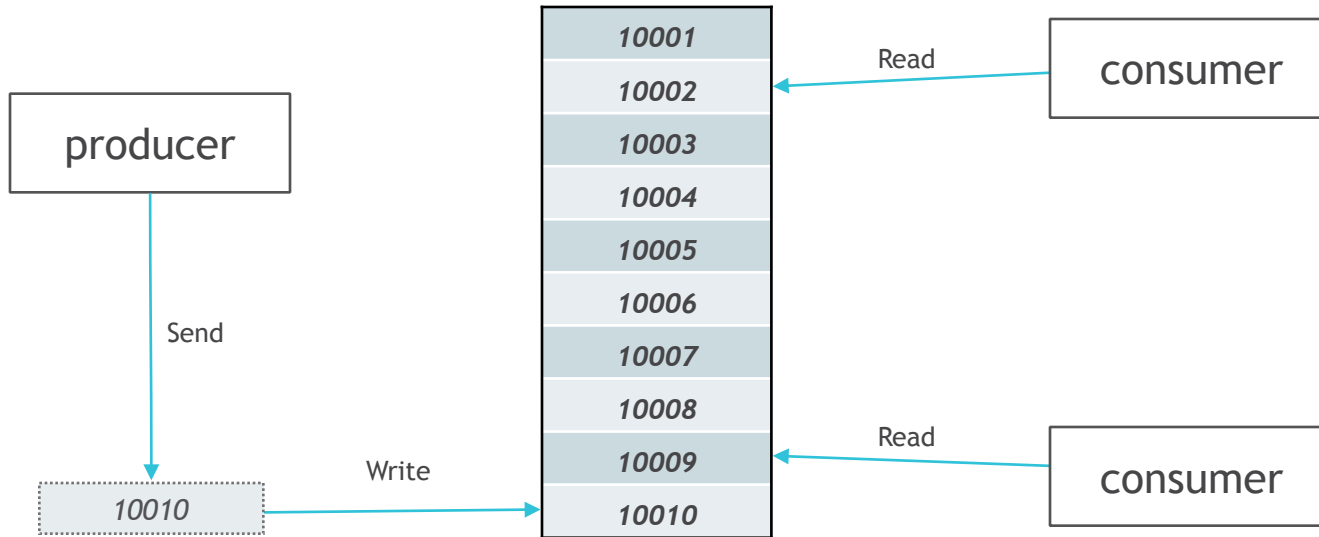
- A Topic is a category or feed name to which messages are published.
- Each topic separated to partitioned log where messages are kept.
- Partitions are replicated and distributed across the Kafka cluster for high availability and fault tolerance.

Partition



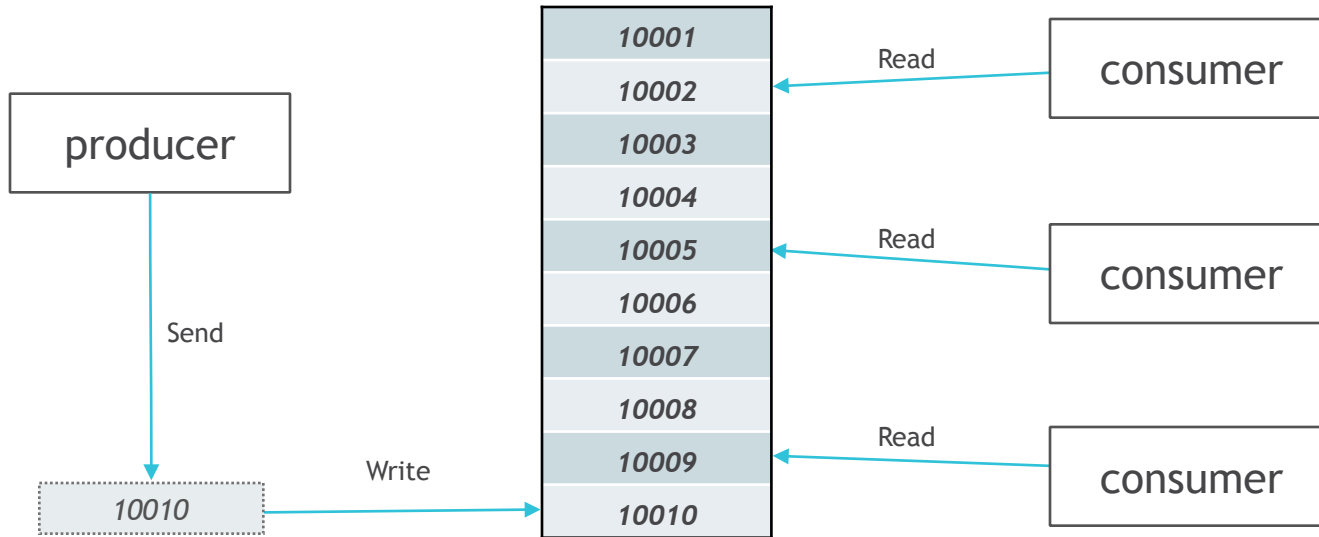
- Multiple consumers can read from same topic on their own pace
- Messages are kept on log for predefined period of time
- Consumer maintain the message offset

Partition



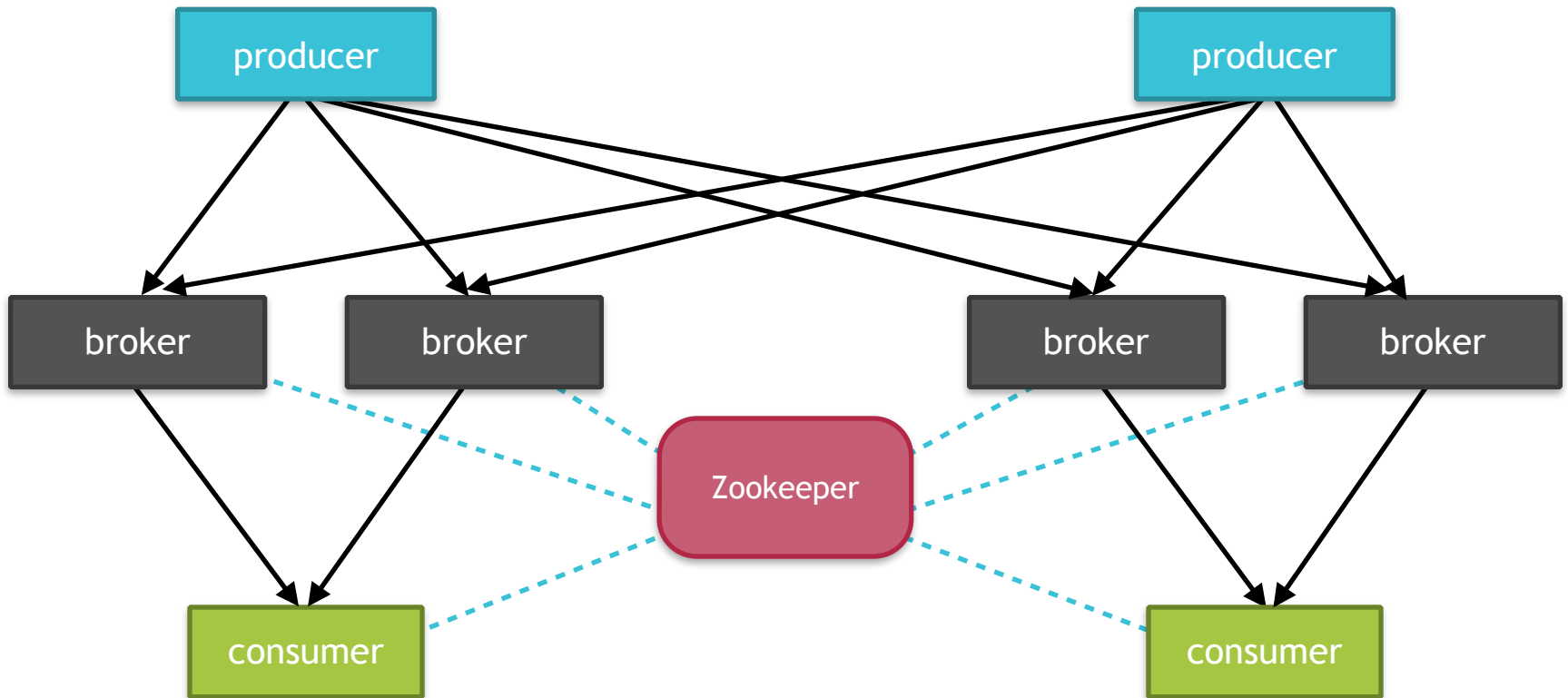
- Consumers can go away

Partition



- and come back

Kafka Architecture

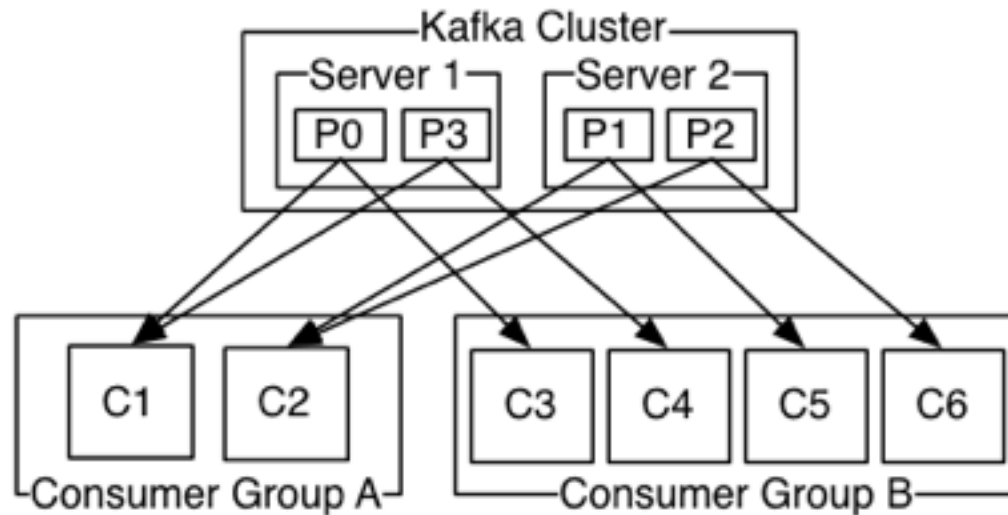


Consumers and Consumer Groups

Kafka addresses 2 traditional messaging models

- Queuing
- Publish-subscribe

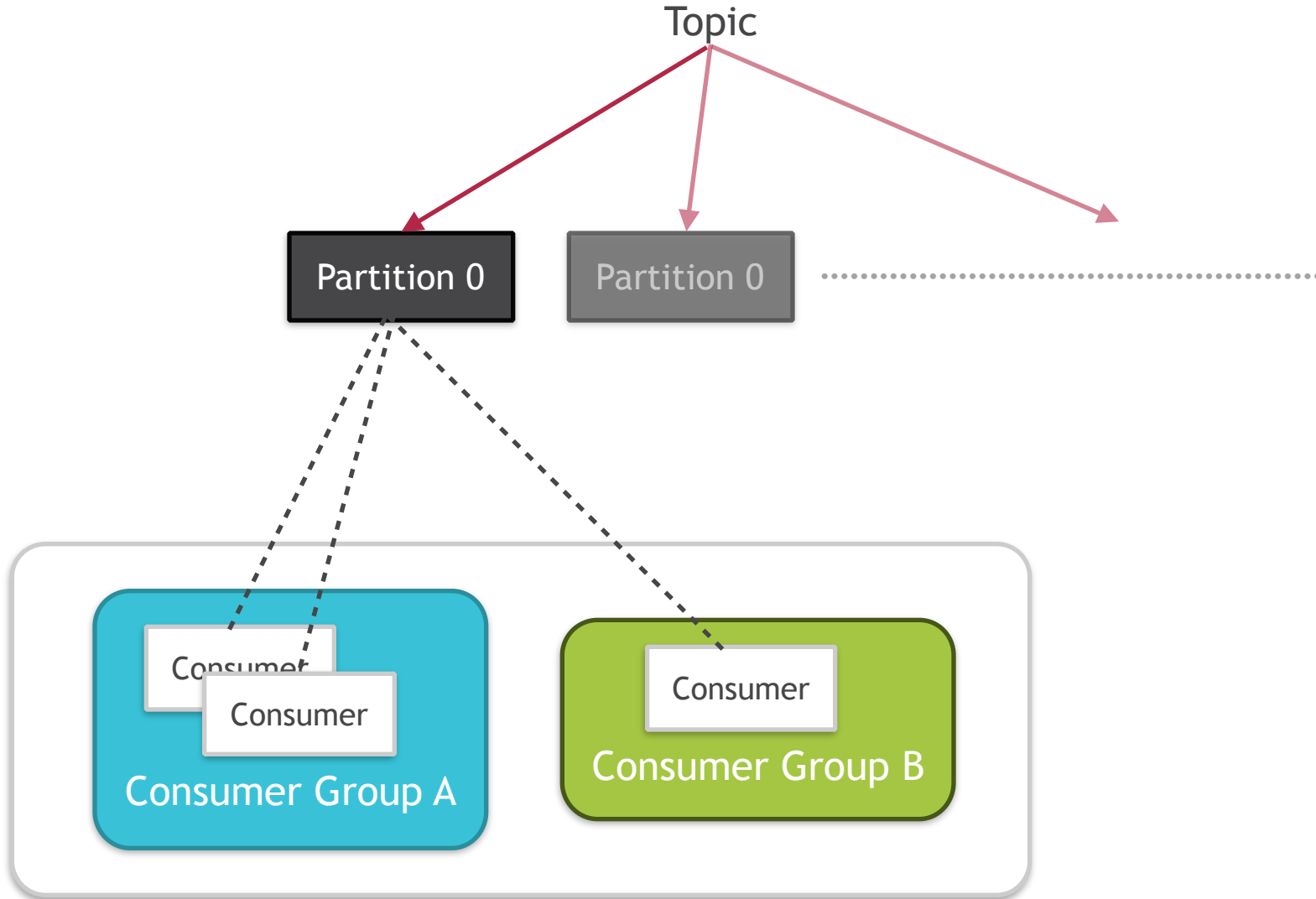
using “consumer group”, a single consumer abstraction



Consumers and Consumer Groups



Consumers and Consumer Groups



Message order and parallelism

TRADITIONAL QUEUE

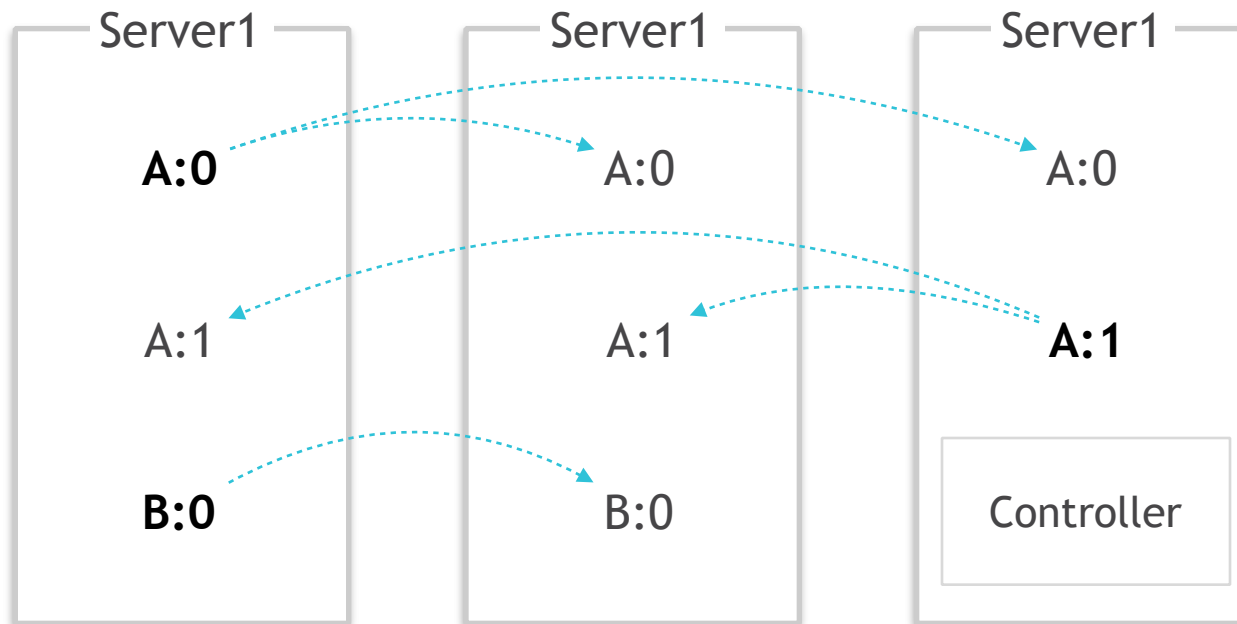
- Retain messages in-order and handover to consumer in-order.
- Message order is not guaranteed when it's come to parallel processing unless it's a exclusive consumer per queue.

KAFKA WAY - THE PARTITION

- Partition on topics
- Partition is assigned to a consumer group so that each portion consumed by a single consumer in consumer group.
- Message order guaranteed on partition basis
 - Message key can be used to order explicitly (consumer)
 - One consumer instance per partition within the consumer group

Replication

- Each portion of a topic has a 1 leader and 0 or more replicas
- Partitions are selected “round-robin” to balance the load unless it is required to maintain the order
- Leader handles all writes to the partition

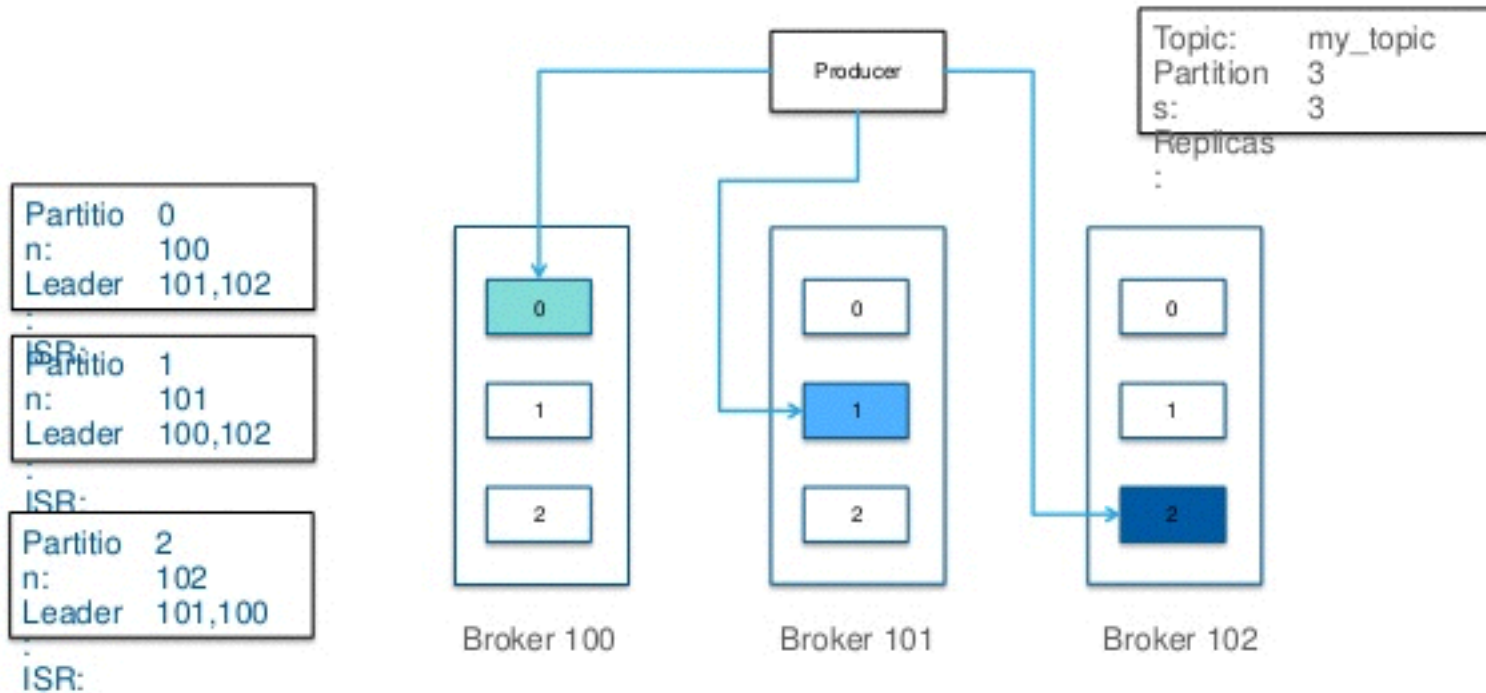


Durability and throughput

- Durability can be configured on producer level
- Durability ~ Throughput
- ISR - group of in-sync replicas for a partition

<i>Durability</i>	<i>Behaviour</i>	<i>Per Event Latency</i>
<i>Highest</i>	<i>ACK all ISRs have received</i>	<i>Highest</i>
<i>Medium</i>	<i>ACK once the leader has received</i>	<i>Medium</i>
<i>Lowest</i>	<i>No ACK required</i>	<i>Lowest</i>

Durability and throughput

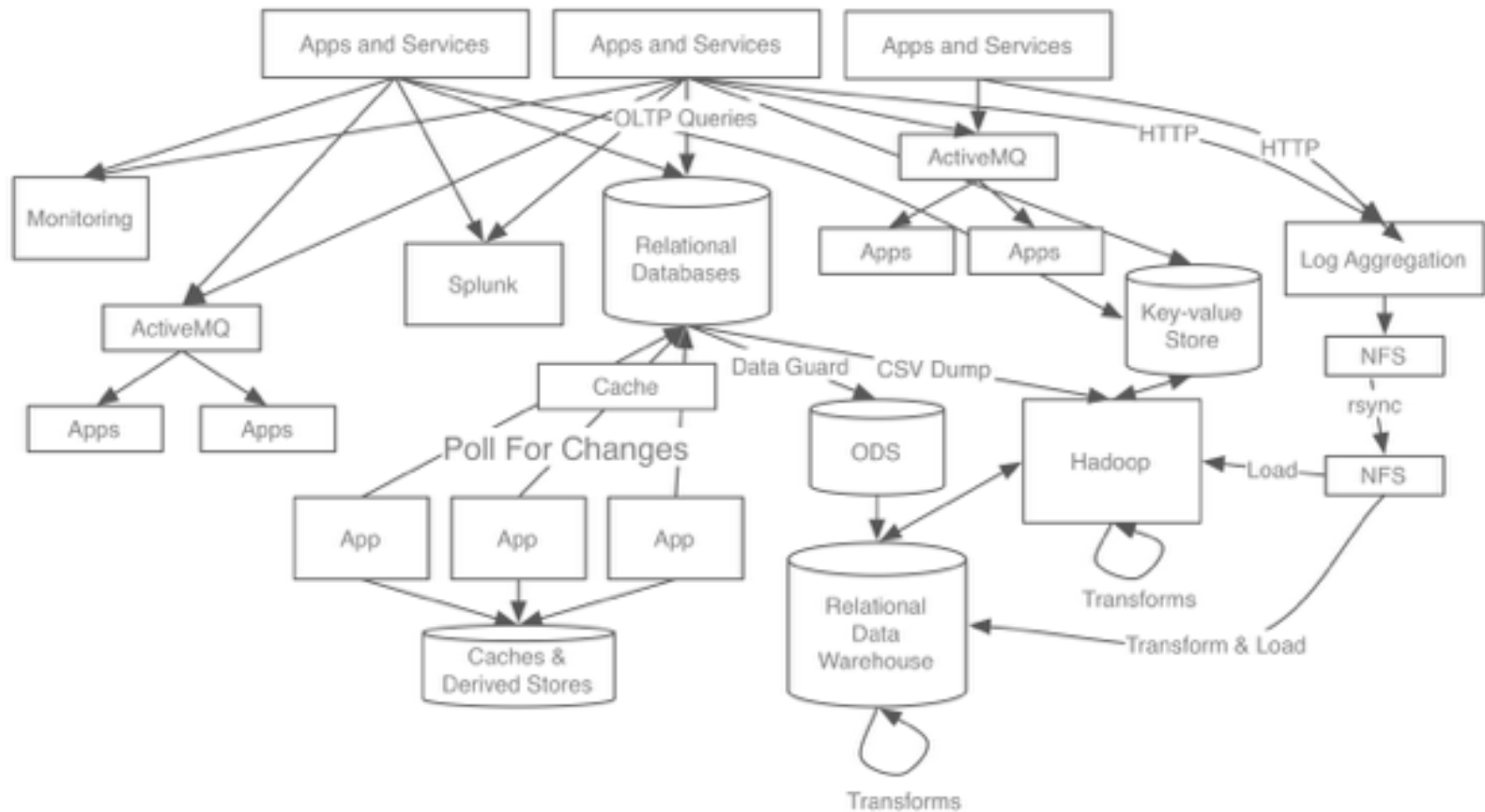


cloudera

© 2015 Cloudera, Inc. All rights reserved. 23

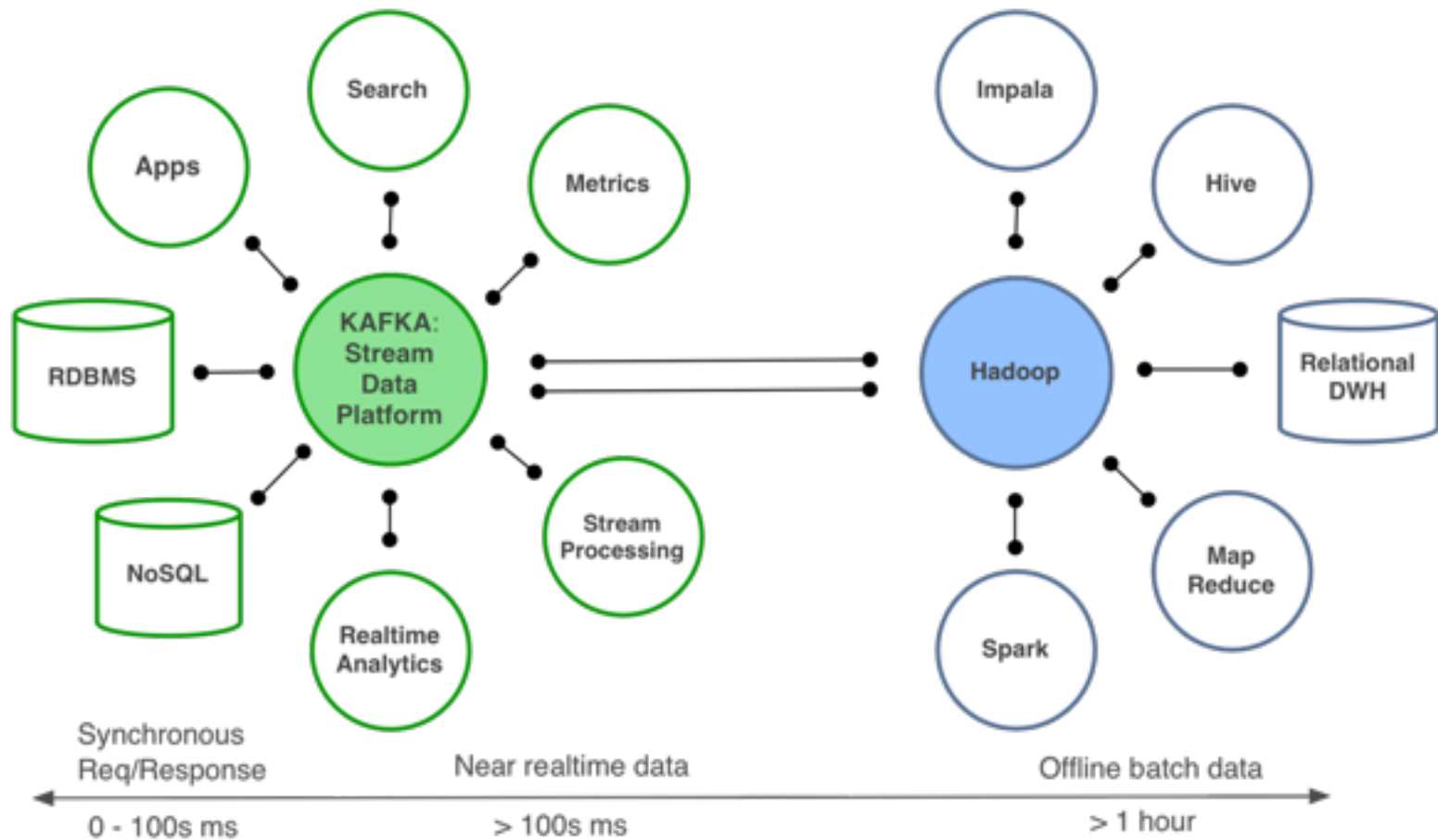
Use case - LinkedIn

Traditional jerry-rigged piped architecture



Source: LinkedIn

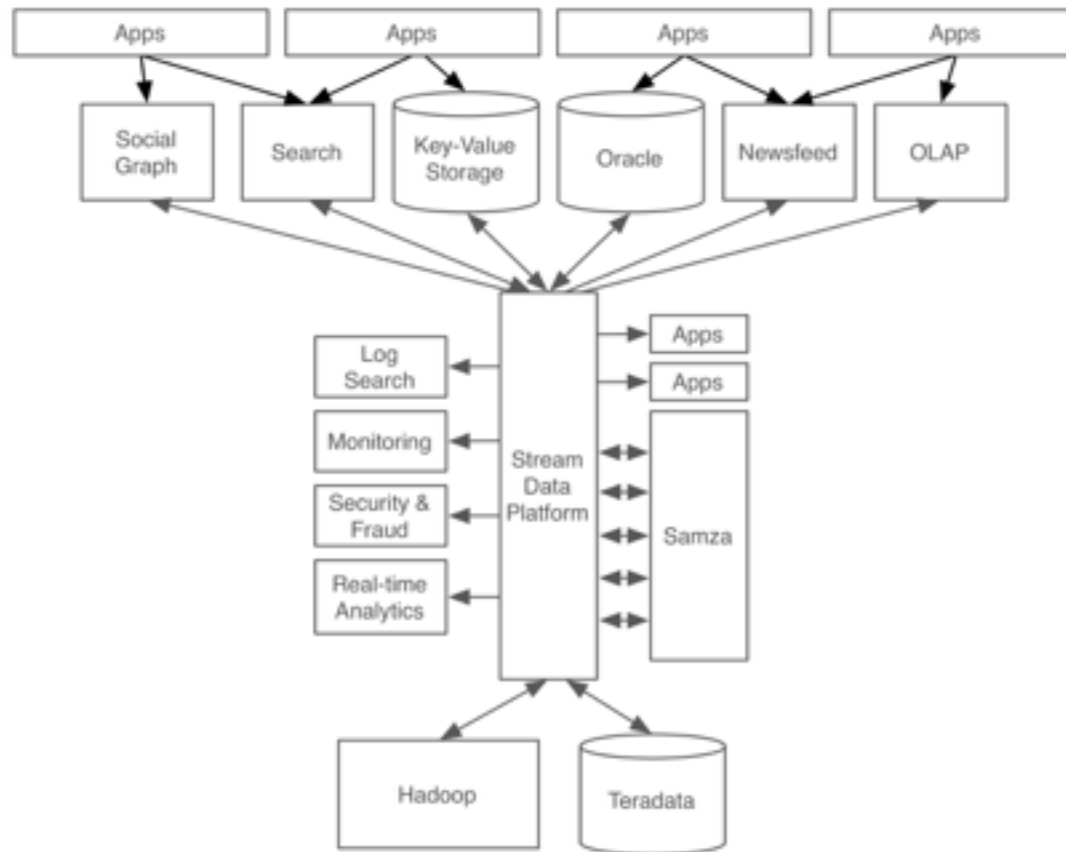
Use case - LinkedIn



Source: LinkedIn

Use case - LinkedIn

Stream-centric data architecture



Source: LinkedIn

What is Stream Processing?

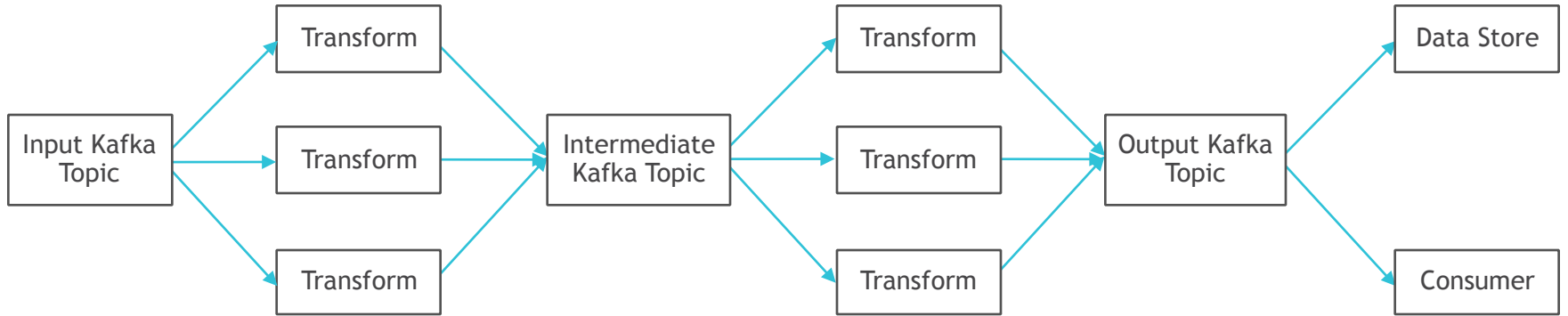


"I'm having trouble streaming."

Stream Processing

Stream processing is a
generalisation of Batch processing

Stream Processing



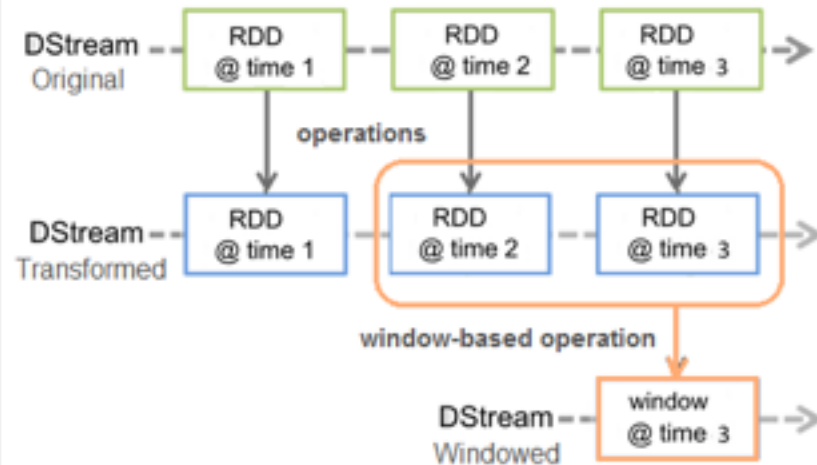
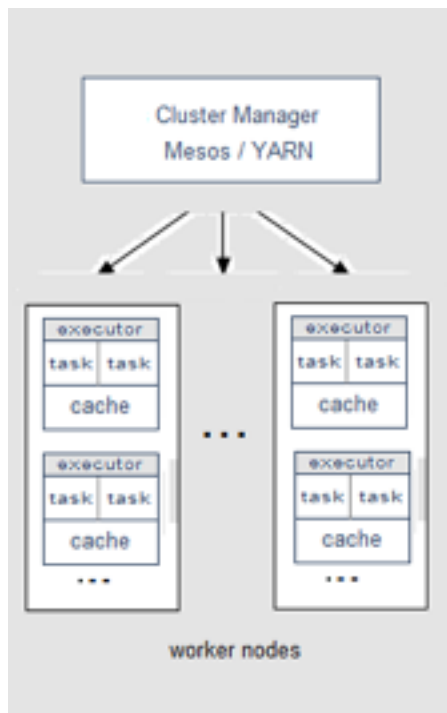
`cat input | grep "foo" | wc`

Stream Processing with Kafka



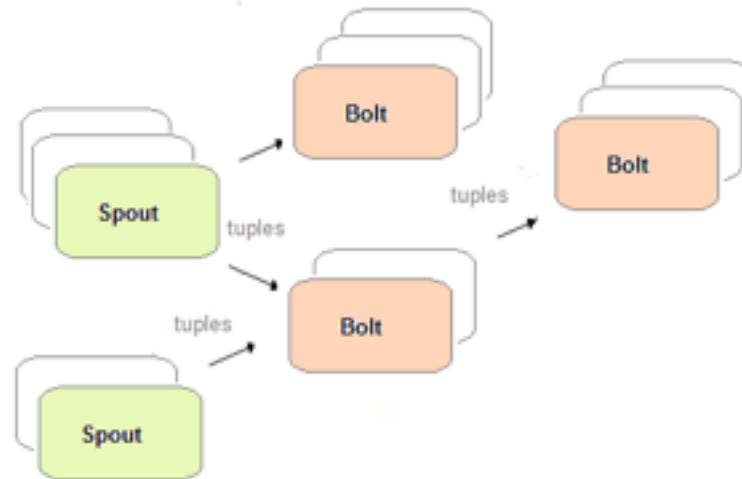
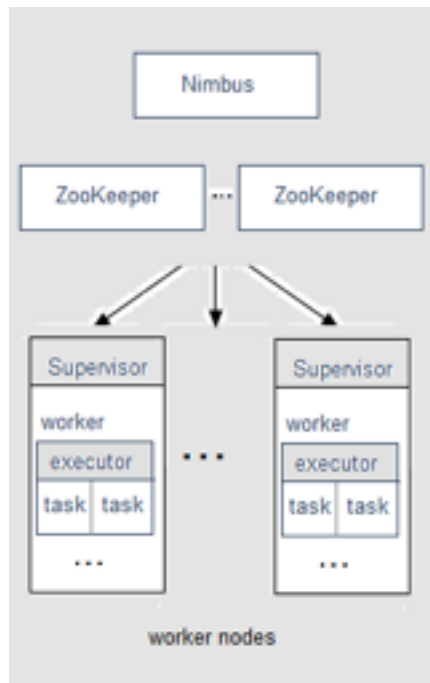
Spark

- Data-Parallel computation
- Micro batch processing
- APIs in Java, Scala, Python
- In-memory storage



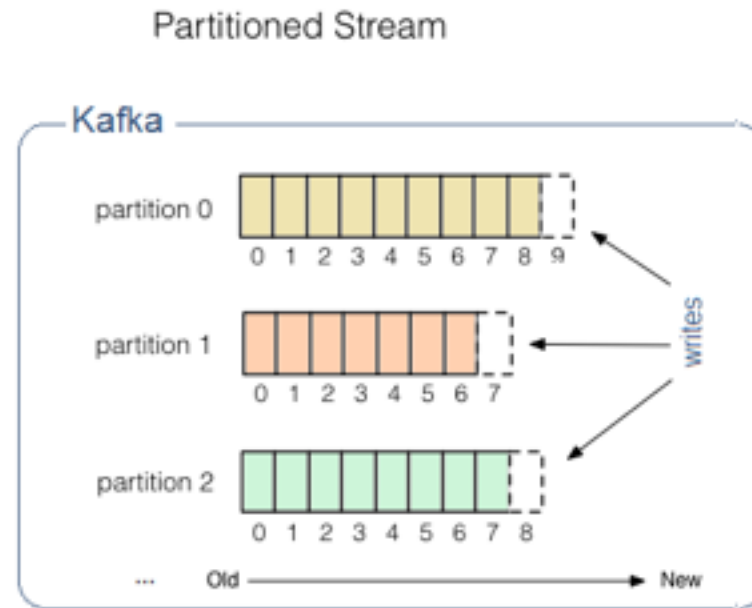
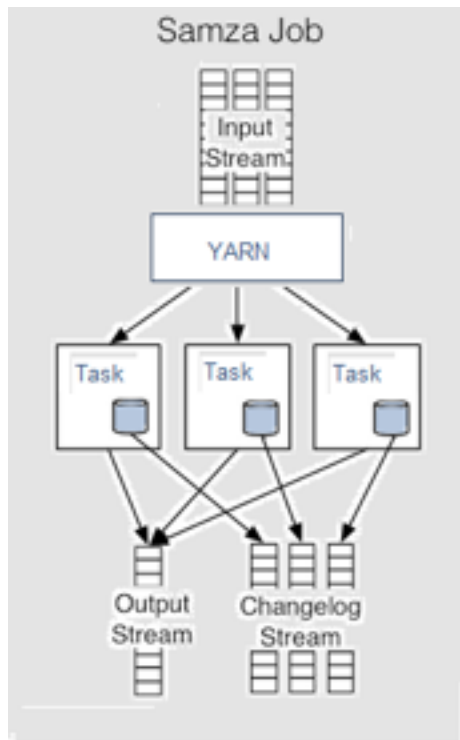
Storm

- Even-Parallel computation
- One-at-a-time processing
- Micro batch processing is possible with Trident
- APIs in Java, Scala, Python, Clojure, Ruby, etc
- Suitable for processing complex event data
- Transform unstructured data in to desired format

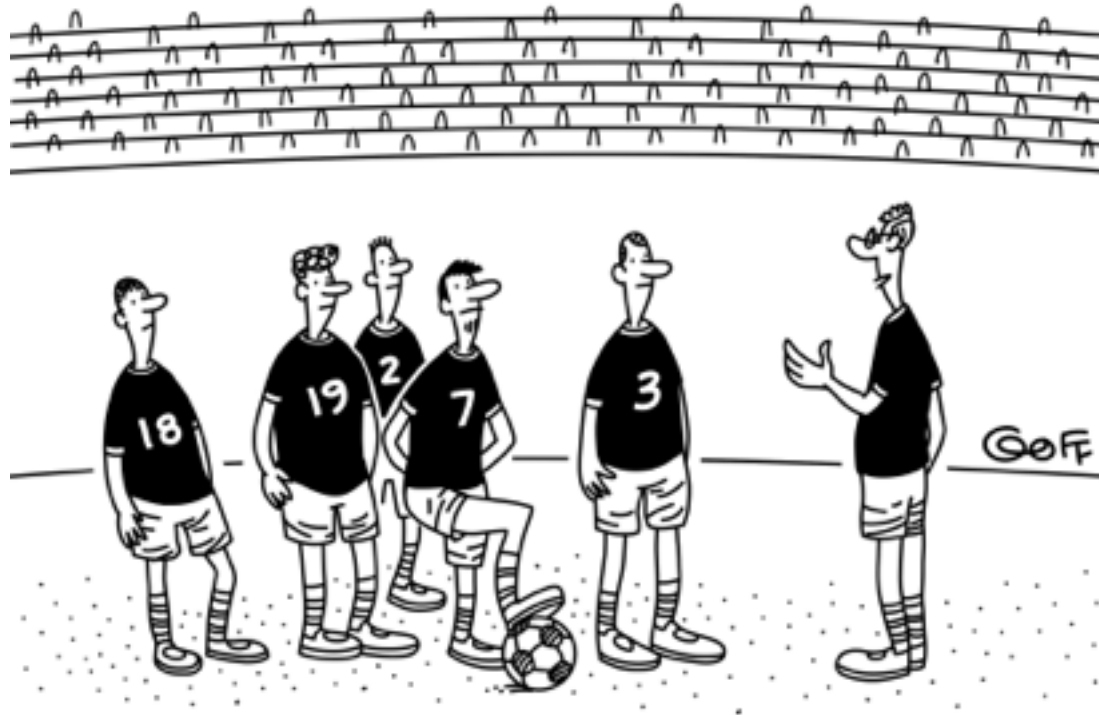


Samza

- One-at-a-time processing
- Based on messages and partitions
- APIs in Java, Scala
- Suitable for processing large amount of data



samza



“Remember, the other team is counting on Big Data insights based on previous games. So, kick the ball with your other foot.”