Big Data Foundations

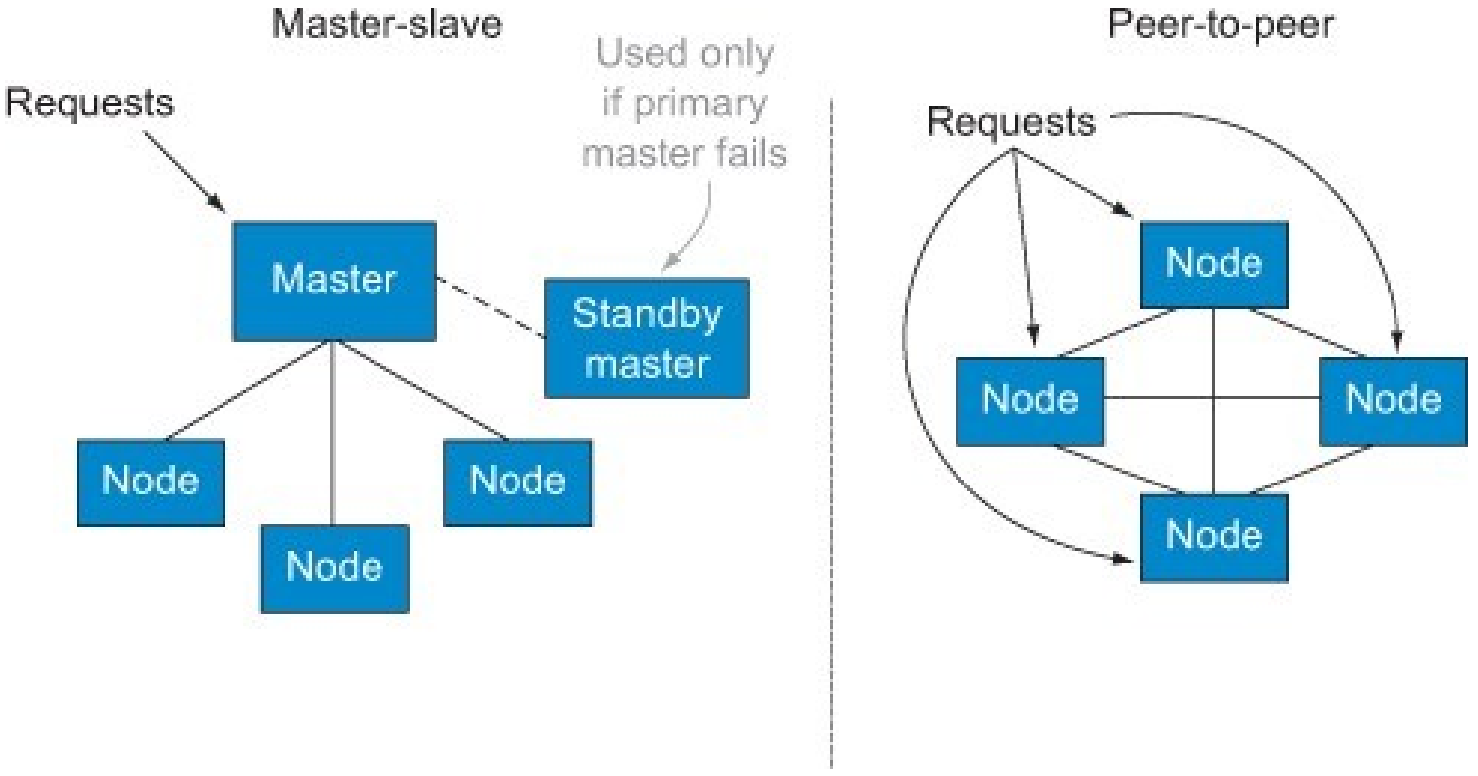
- Sai Kiran Krishna Murthy

Agenda

- Day 1: Introduction
- **Day 2: HDFS, Yarn, Spark, Kafka**
- Day 3: Practical Showcase
- Day 4: Free day

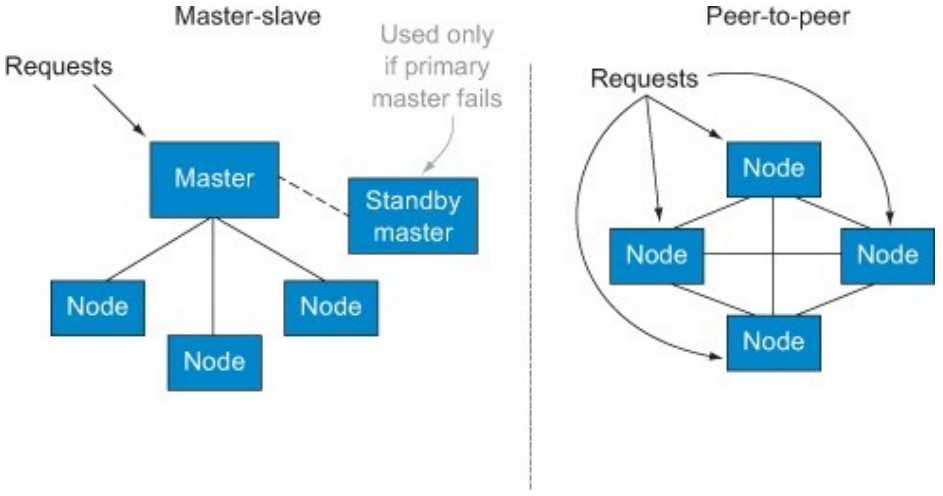
Recap

Scale out: In practice



Features

- Concurrency
- Scalability
- Availability
- Fault Tolerance



Map + Reduce

- Split a big problem in smaller pieces
- Map each task to a counter (worker)
- Reduce the results of each worker, until you have only one result

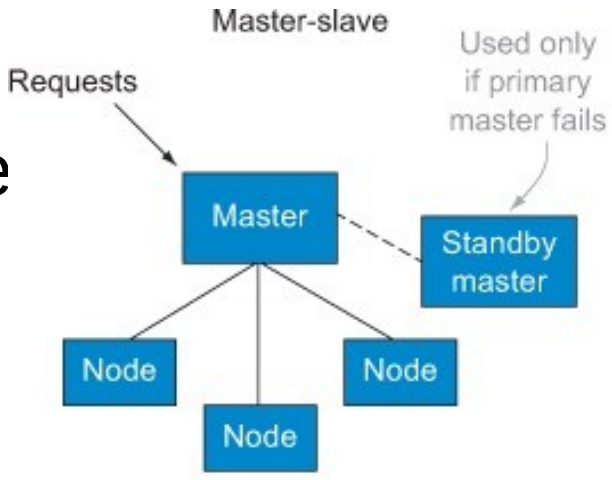
All the components together

- Storage = HDFS
- Processing Framework = Spark
- Scheduler = Yarn
- Synchronization = Zoo Keeper
- SQL interface for HDFS = Hive
- NoSQL Database = HBase
- Message Queue = Kafka
- Notebook = Zeppelin

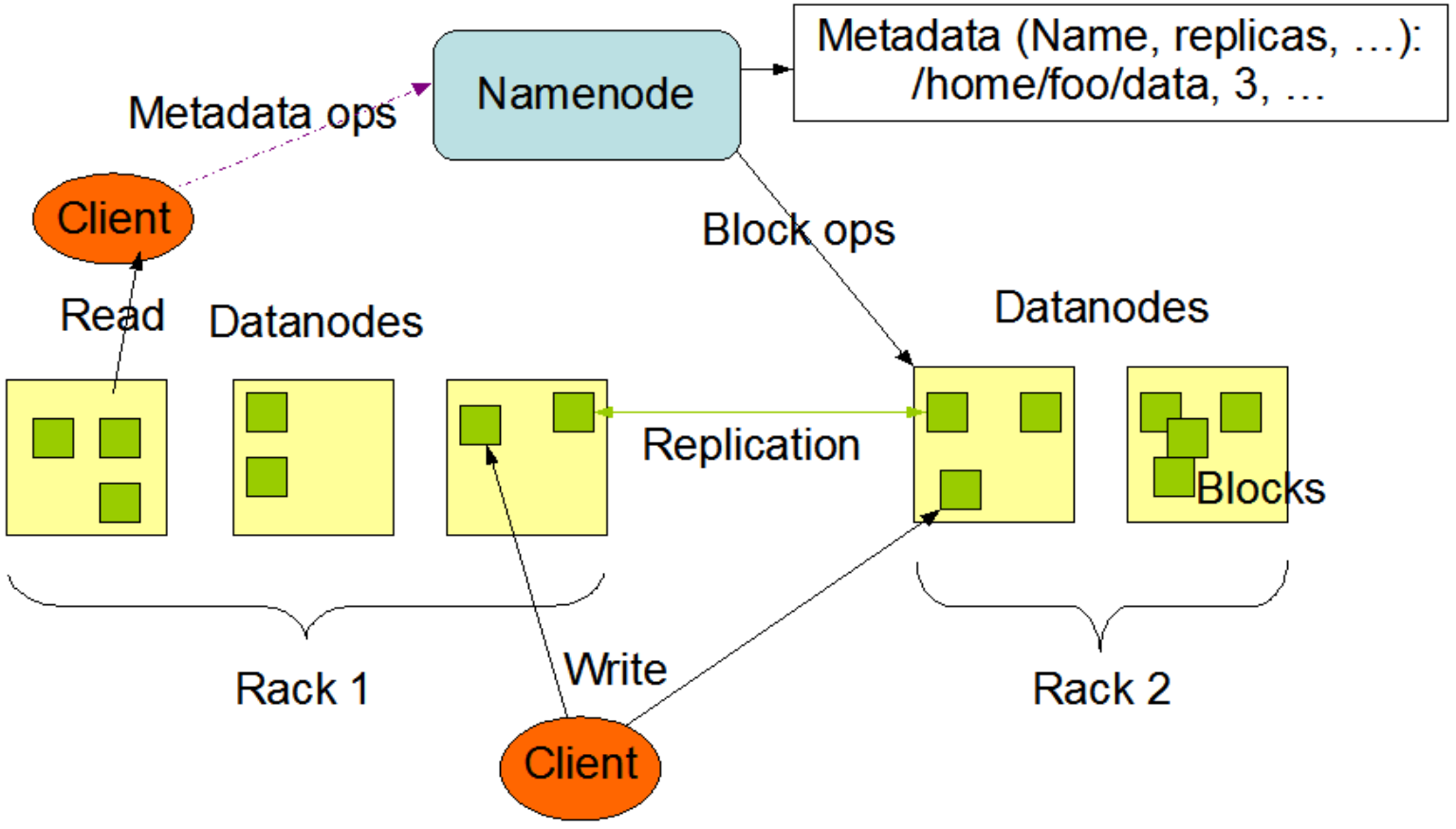
HDFS

Storage layer: HDFS

- Hadoop Distributed File System
- Uses the master slave architecture
- Master = Name Node
- Standby Master = Secondary name node
- Slave = Data Node



Architecture



What are blocks?

- Every file in hdfs is divided into blocks
- How many blocks / file => determined by BlockSize
- Traditional BlockSize = 64MB, 128MB, 256MB

Replication

- Every block in HDFS is replicated.
- The number of replicas is determined by the config ReplicationFactor
- Replication Factor: Defines the number of replicas of a block, that should be maintained in the cluster

Name Node vs Data Node

- Manages File System Namespace
  - Maps a file name to a set of **blocks**
  - Maps a block to the DataNodes where it resides
- Cluster Configuration Management
- **Replication** Engine for Blocks

- A Block Server
  - Stores data in the local file system (e.g. ext3)
  - Stores metadata of a block (e.g. CRC)
  - Serves data and metadata to Clients
- Block Report
  - Periodically sends a report of all existing blocks to the NameNode

How are Failures Detected: Heartbeats

- DataNodes send hearbeat to the NameNode
  - Once every 3 seconds
- NameNode uses heartbeats to detect DataNode failure

Rebalancer

- Goal: % disk full on DataNodes should be similar
  - Usually run when new DataNodes are added
  - Cluster is online when Rebalancer is active
  - Rebalancer is throttled to avoid network congestion

Exercise

- What is the blocksize in our acc cluster?
- What is the replication factor in our acc cluster?

Discussion

- What happens when we lose a data node?
  - Do we lose data?
- What is a good block size?
  - What happens when the block size is too low vs too high
  - Impact of too many blocks?

Apache Avro

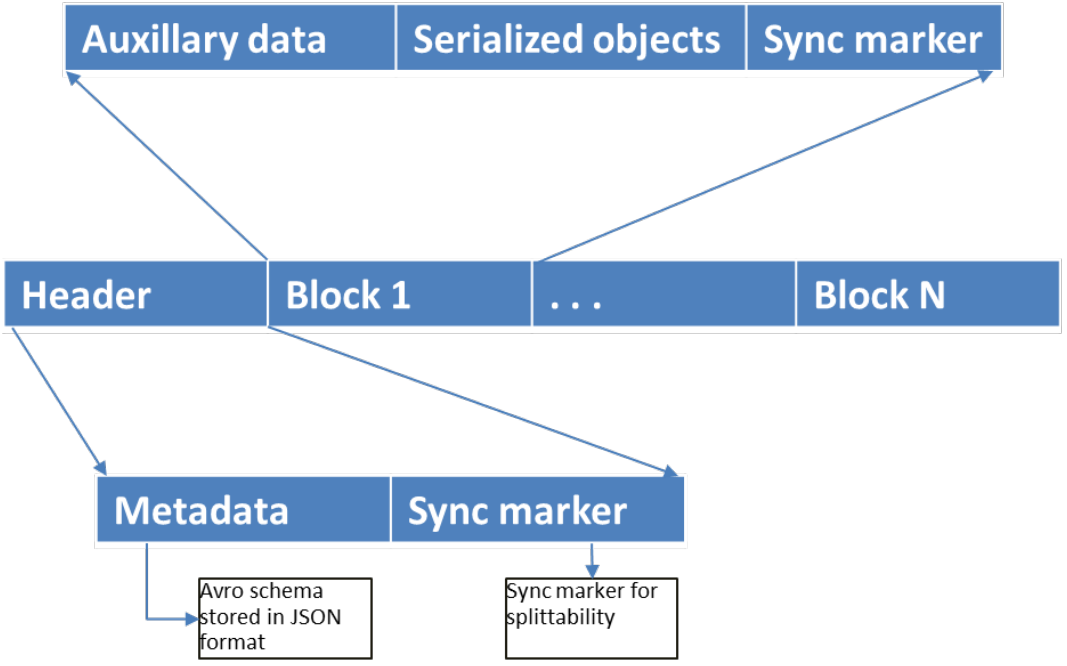
Apache Avro

- Language neutral data serialization system
  - Write a file in python and read it in C
- AVRO data is described using language independent schema
- AVRO schemas are usually written in JSON and data is encoded in binary format
- Supports **schema evolution**
  - producers and consumers at different versions of schema
- Supports **compression** and are **splittable**

Structure of an Avro File

Sample AVRO schema in JSON format Avro file structure

```
{
  "type" : "record",
  "name" : "tweets",
  "fields" : [ {
    "name" : "username",
    "type" : "string",
}, {
    "name" : "tweet",
    "type" : "string",
}, {
    "name" : "timestamp",
    "type" : "long",
} ],
  "doc:" : "schema for storing tweets"
}
```

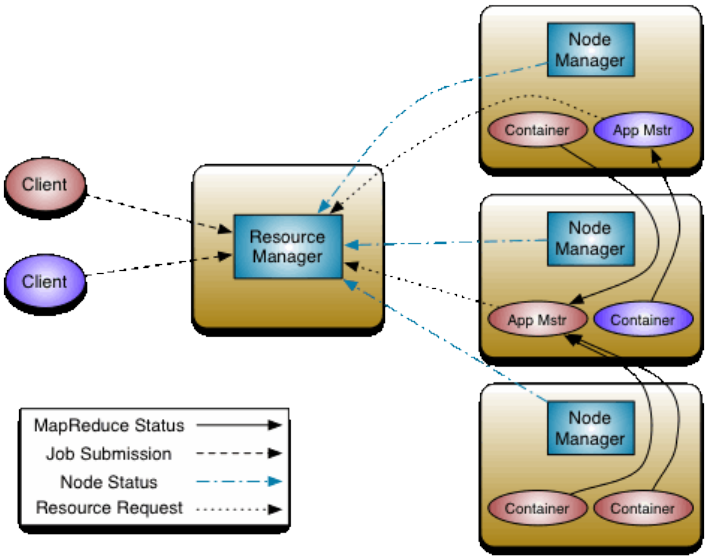


Apache Yarn

Apache Yarn

- YARN = Yet Another Resource Negotiator
- Cluster Manager
  - Resource Allocation
  - Resource Deallocation
- Language / Platform independent

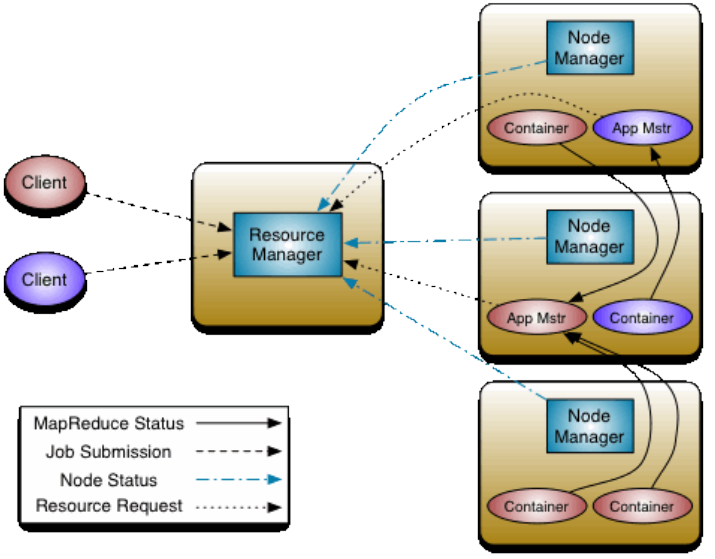
Yarn Achitecture



- Resource Manager = Master
- Node Manager = Slave
- Container = Unit of allocation incorporating resource elements such as memory, cpu, disk,network etc, to execute a specific application

Yarn Execution Sequence

1. A client program submits the application
2. ResourceManager allocates a specified container to start the ApplicationMaster
3. ApplicationMaster, on boot-up, registers with ResourceManager
4. ApplicationMaster negotiates with ResourceManager for appropriate resource containers



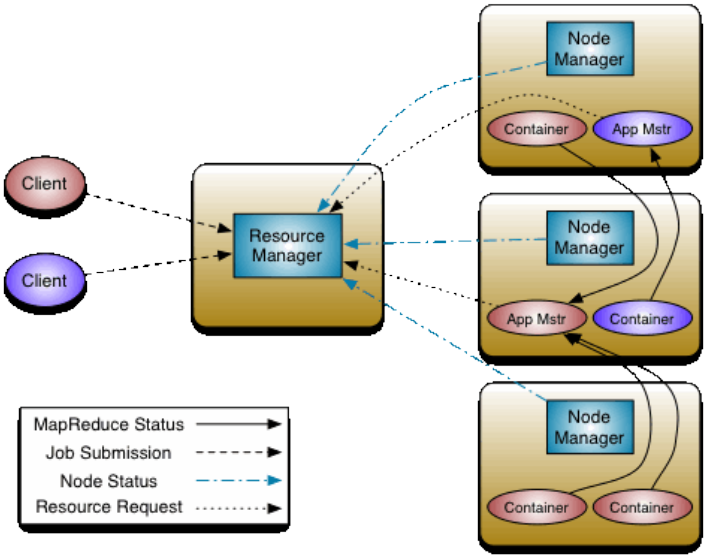
Yarn Execution Sequence 2

5. On successful container allocations, ApplicationMaster contacts NodeManager to launch the container

6. Application code is executed within the container, and then ApplicationMaster is responded with the execution status

7. During execution, the client communicates directly with ApplicationMaster or ResourceManager to get status, progress updates etc.

8. Once the application is complete, ApplicationMaster unregisters with ResourceManager and shuts down, allowing its own container process



Apache Spark

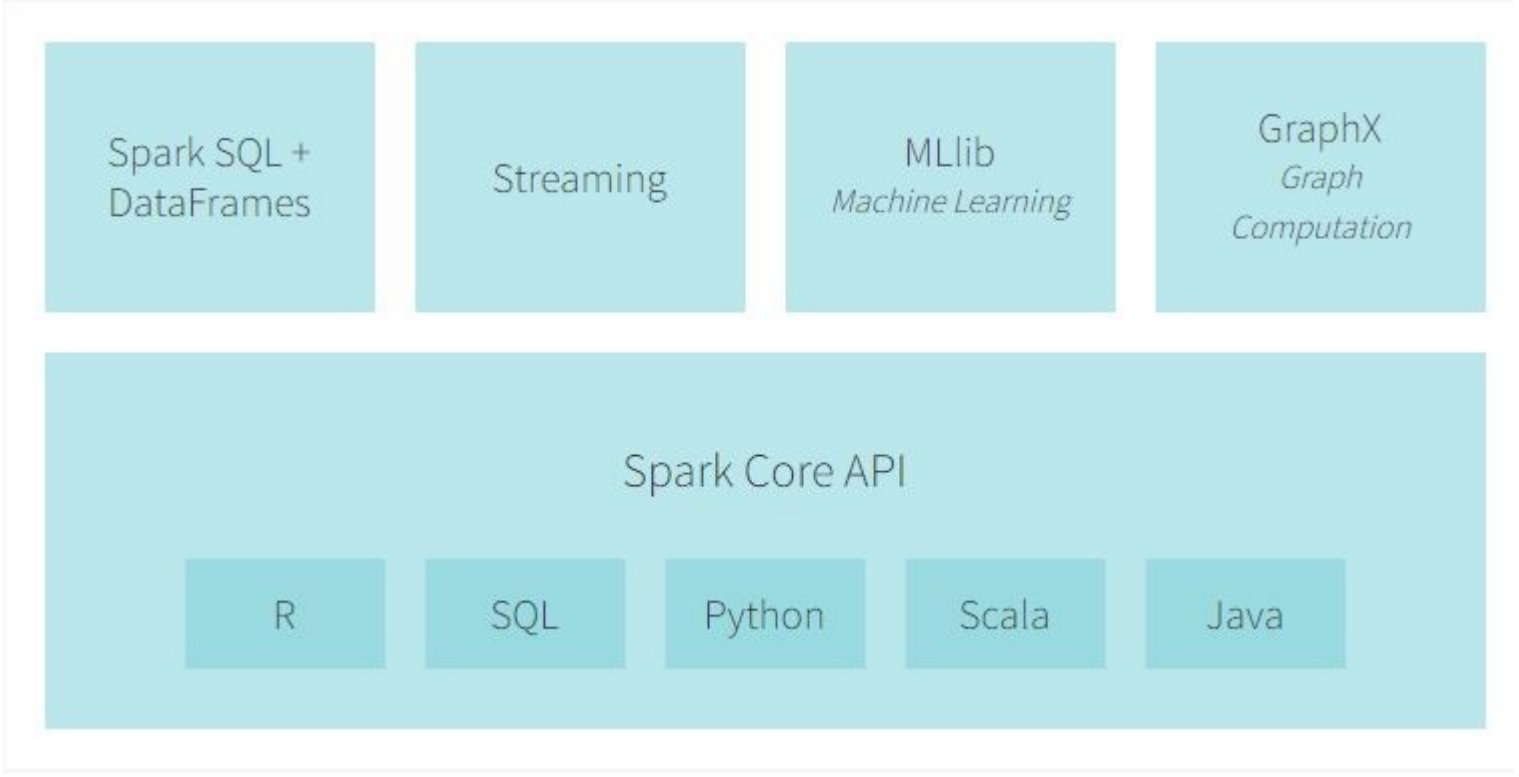
Apache Spark

Apache Spark is a lightning-fast **unified analytics engine** for big data and machine learning. It was originally developed at UC Berkeley in 2009

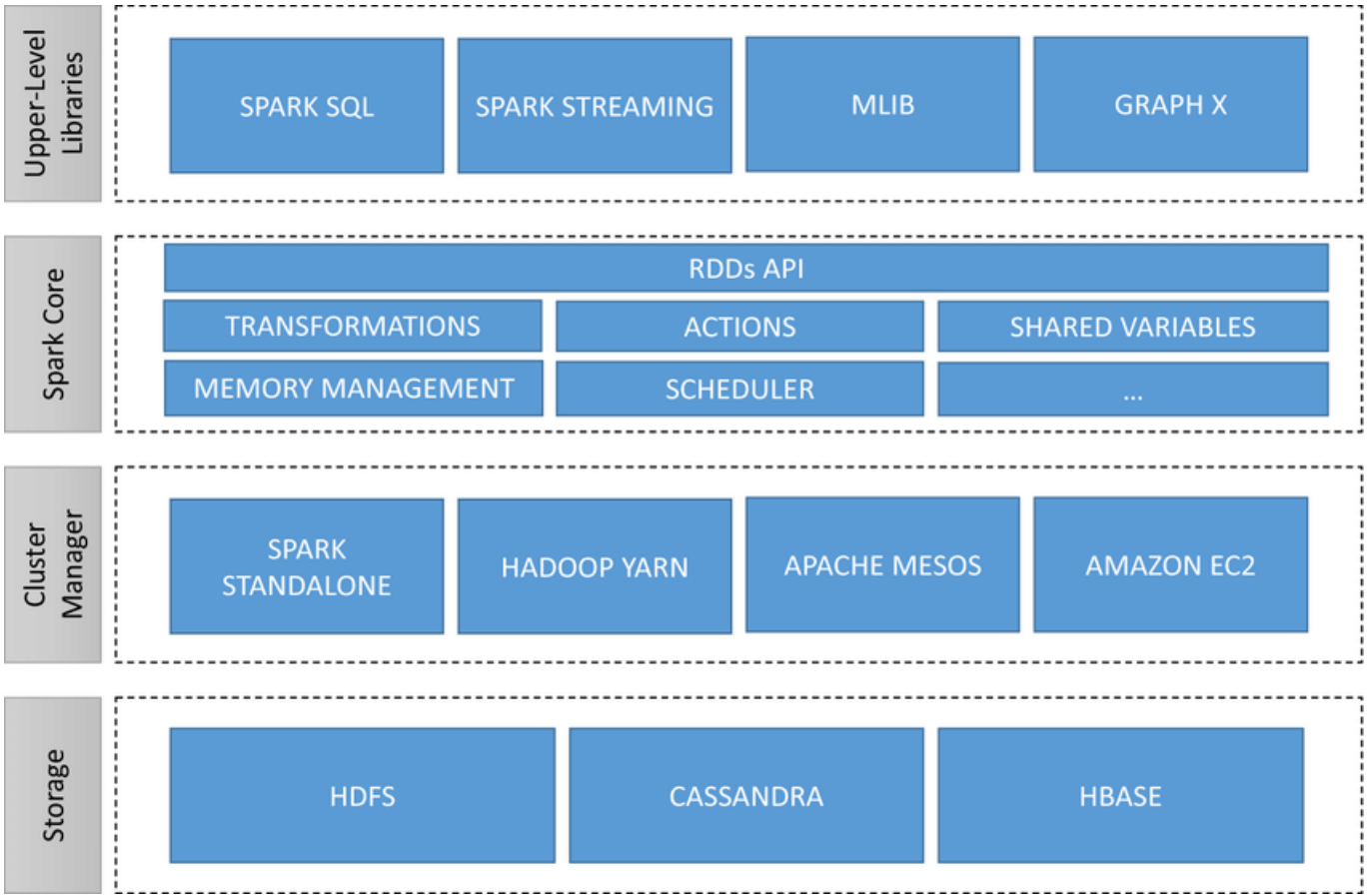
Apache Spark™ is a unified analytics engine for large-scale data processing

Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance.

Spark Ecosystem



How it fits in



RDDs

Resilient Distributed Datasets (RDD)

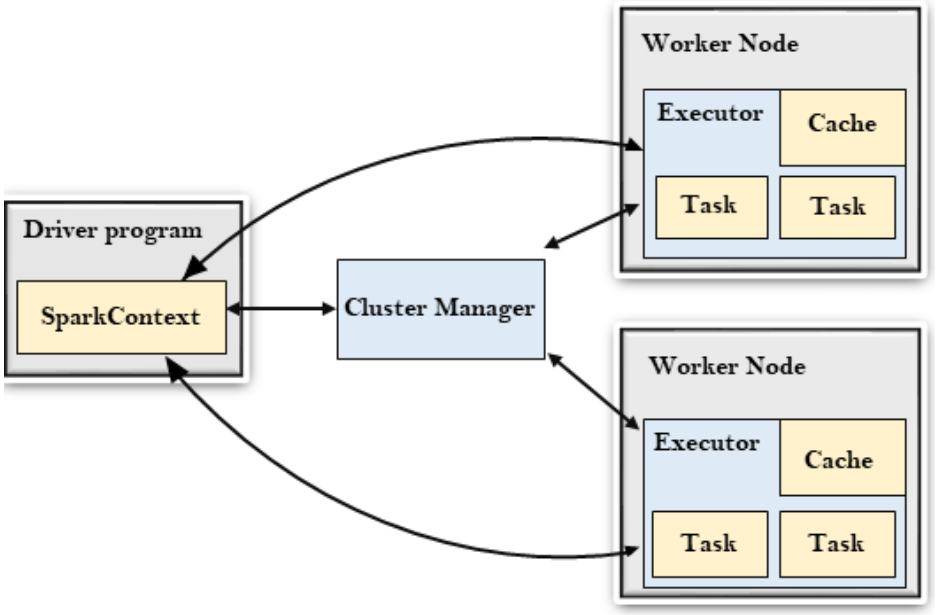
- Spark's primary abstraction
- Distributed collection of elements
- Parallelized across the cluster
- Two types of RDD operations
  - Transformations
    - Creates a DAG
    - Lazy evaluations
    - No return value
  - Actions
    - Performs the transformations and the action that follows
    - Returns a value
- Fault tolerance
- Caching

Transformations vs Actions

Transformations				
map	join	union	distinct	repartition
mapPartitions	flatMap	intersection	pipe	coalesce
cartesian	cogroup	filter	sample	
sortByKey	groupByKey	reduceByKey	aggregateByKey	
mapPartitionswithIndex		repartitionAndSortWithinPartitions		

Actions				
reduce	take	collect	takeSample	count
takeOrdered	countByKey	first	foreach	saveAsTextFile
saveAsSequenceFile		saveAsObjectFile		

Spark Execution Model



- Driver = Master
- Executor = Slave
- CM = Yarn
- Worker = Node
- Task = Unit of work

Apache Kafka

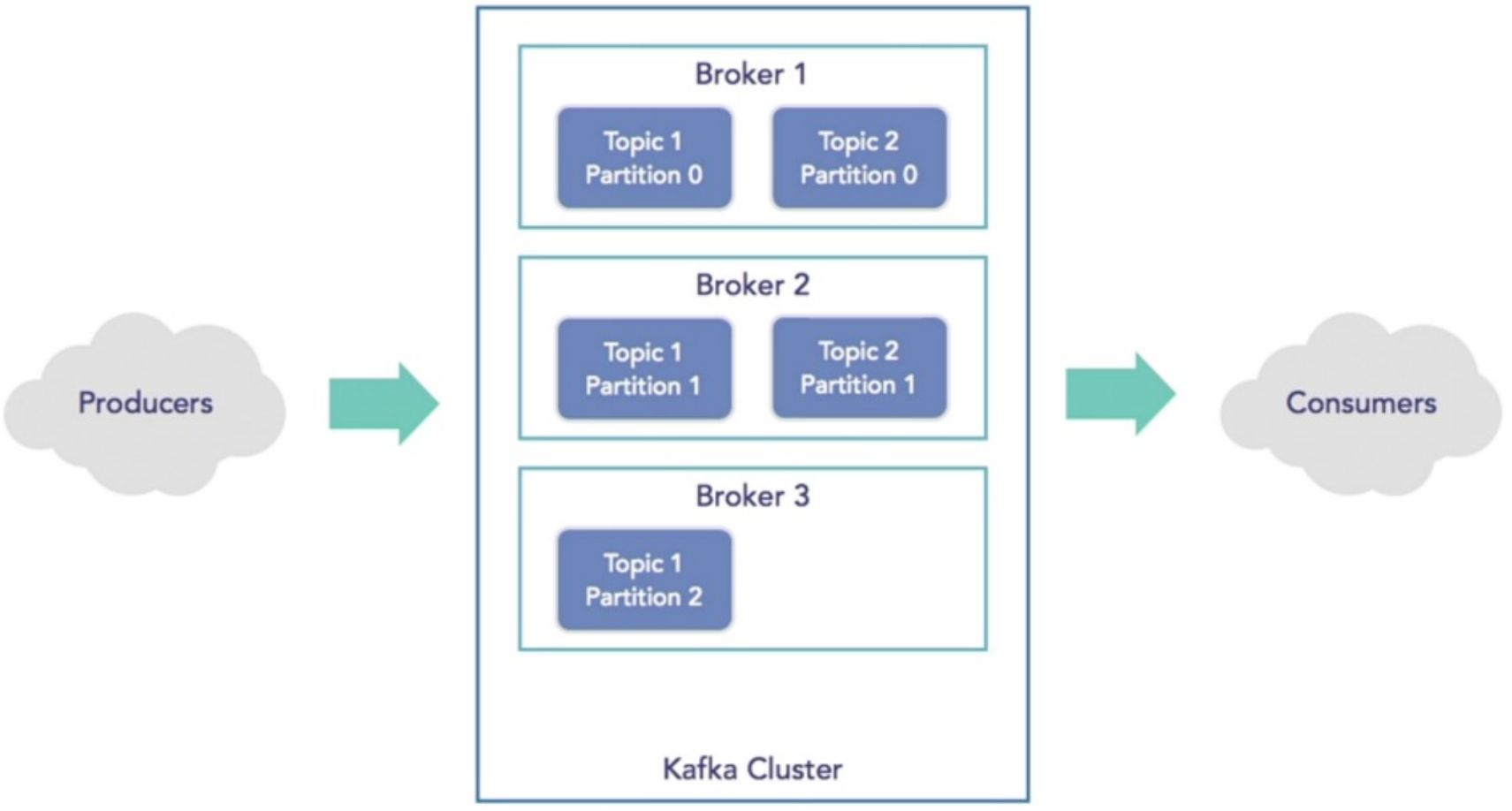
Apache Kafka

- Highly scalable, fault tolerant and distributed Message Queue (Distributed commit log)
- Designed for low latency and high throughput
- Ideal for stream processing (HDFS is for Batch processing)

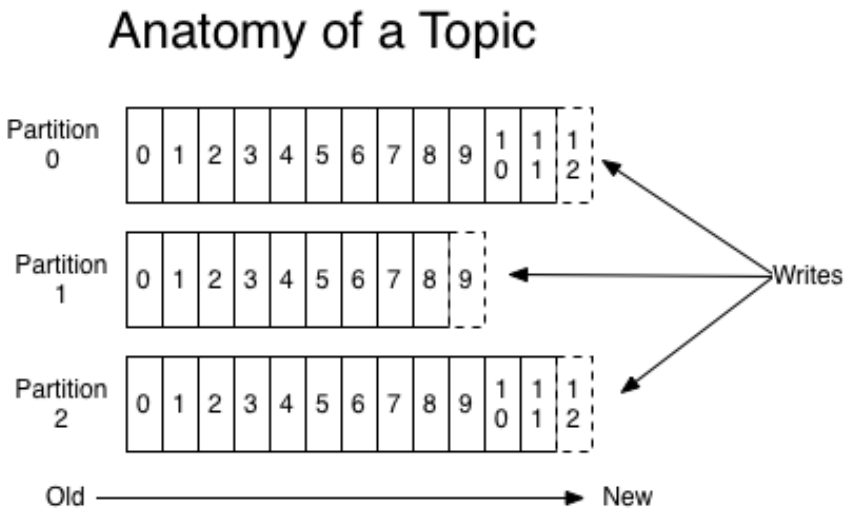
Kafka Components

- Producers
  - Writes data to topics
- Broker
  - Manages the storage of messages in the topic(s)
  - If Kafka has more than one broker, that is what we call a Kafka cluster.
- Consumers
  - Read data from topics
- Zookeeper
  - Maintains metadata
  - Synchronization

Kafka Architecture

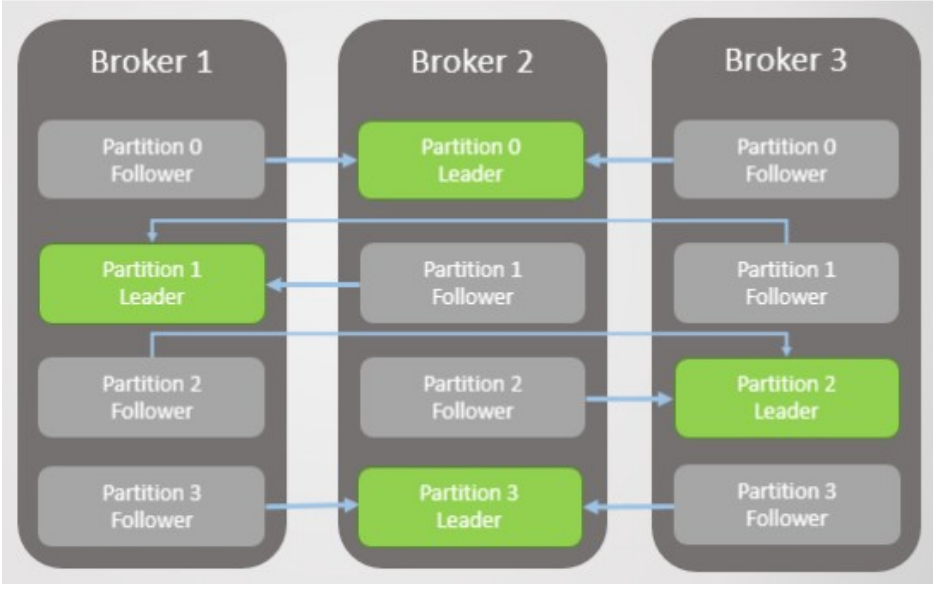


Diving deeper



- Topic similar to a table in Oracle
- Contains 1 or more Partitions
- Every Partitions has a sequence of messages
- Each message has a retention time, after which the message is deleted

Replication in Kafka



- Replication is on a topic level
- Is defined by the property “replication-factor”
- Writes happen at the leader and then propagated to the follower