APACHE SPARK

A HANDS-ON INTRODUCTION TO BIG DATA ANALYTICS

AGENDA

Brief History

Introduction to Spark

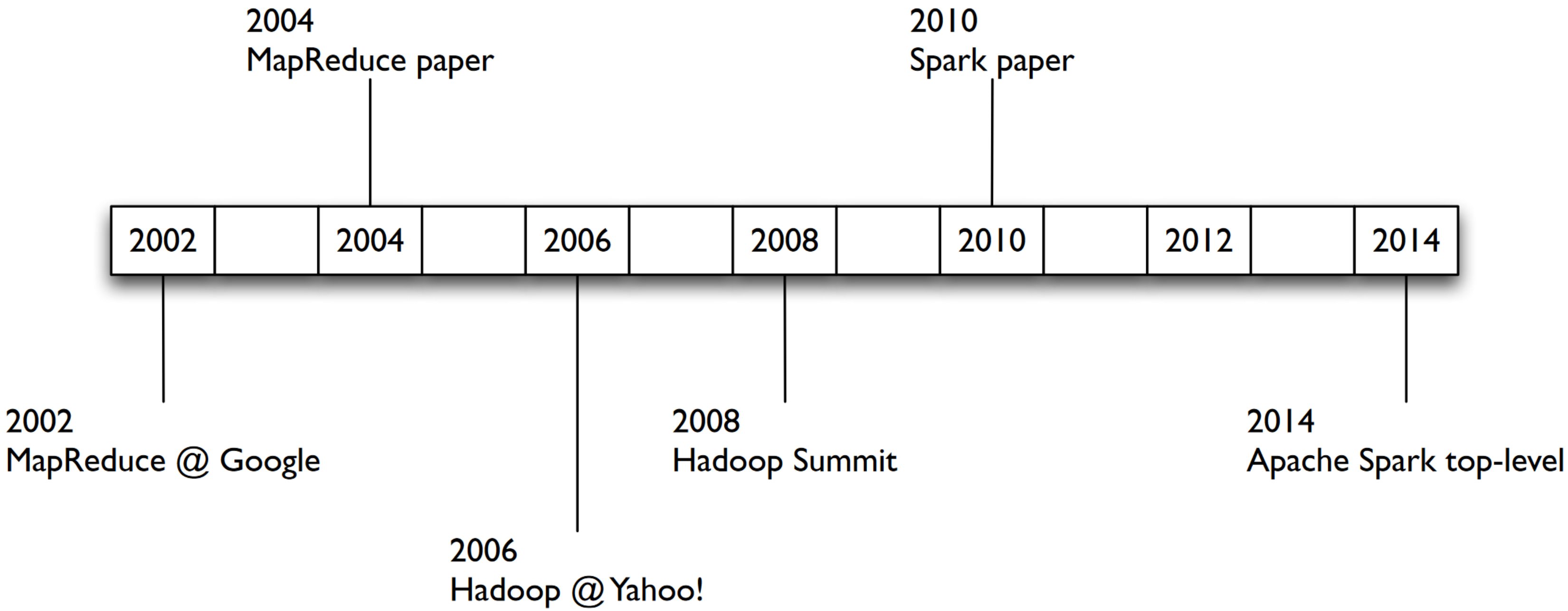
Spark Components

Architecture

Hadoop Ecosystem

BRIEF HISTORY

FROM MAP REDUCE TO APACHE SPARK



2004
MapReduce paper

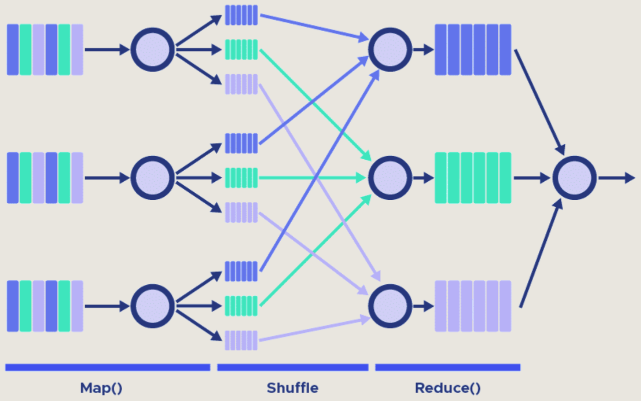
2010
Spark paper

2002
MapReduce @ Google

2006
Hadoop @ Yahoo!

2008
Hadoop Summit

2014
Apache Spark top-level



INTRODUCING MAP/REDUCE

Map

- The input data is split into smaller blocks.
- The number of mappers to be used is decided based on the size of the data to be processed and the memory block available on each mapper.
- Each block is then assigned to a mapper for processing.
- Each 'worker' node applies the map function to the local data, and writes the output to temporary storage.

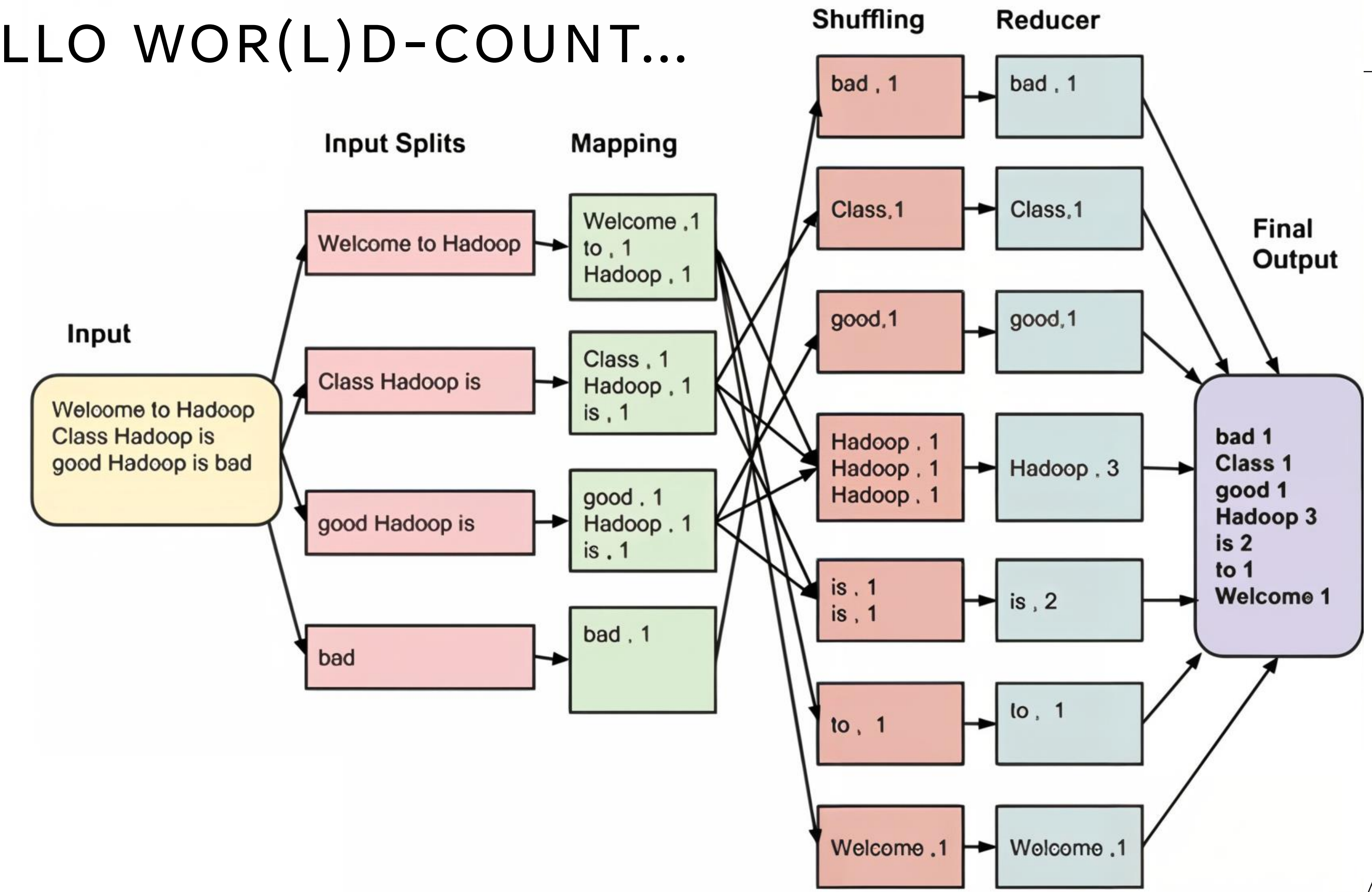
Shuffle, combine and partition

- Worker nodes redistribute data based on the output keys (produced by the map function). All data belonging to one key end up on the same worker node,
- As an optional process the combiner (a reducer) can run individually on each mapper to reduce the data footprint and make shuffling and sorting easier.
- Partition (not optional) is the process that decides how the data has to be presented to the reducer and also assigns it to a particular reducer.

Reduce

- A reducer cannot start while a mapper is still in progress.
- Worker nodes process each group of `<key,value>` pairs output data, in parallel to produce `<key,value>` pairs as output.
- All the map output values that have the same key are assigned to a single reducer, which then aggregates the values for that key.
- Unlike the map function, the reduce function is optional.

HELLO WOR(L)D-COUNT…

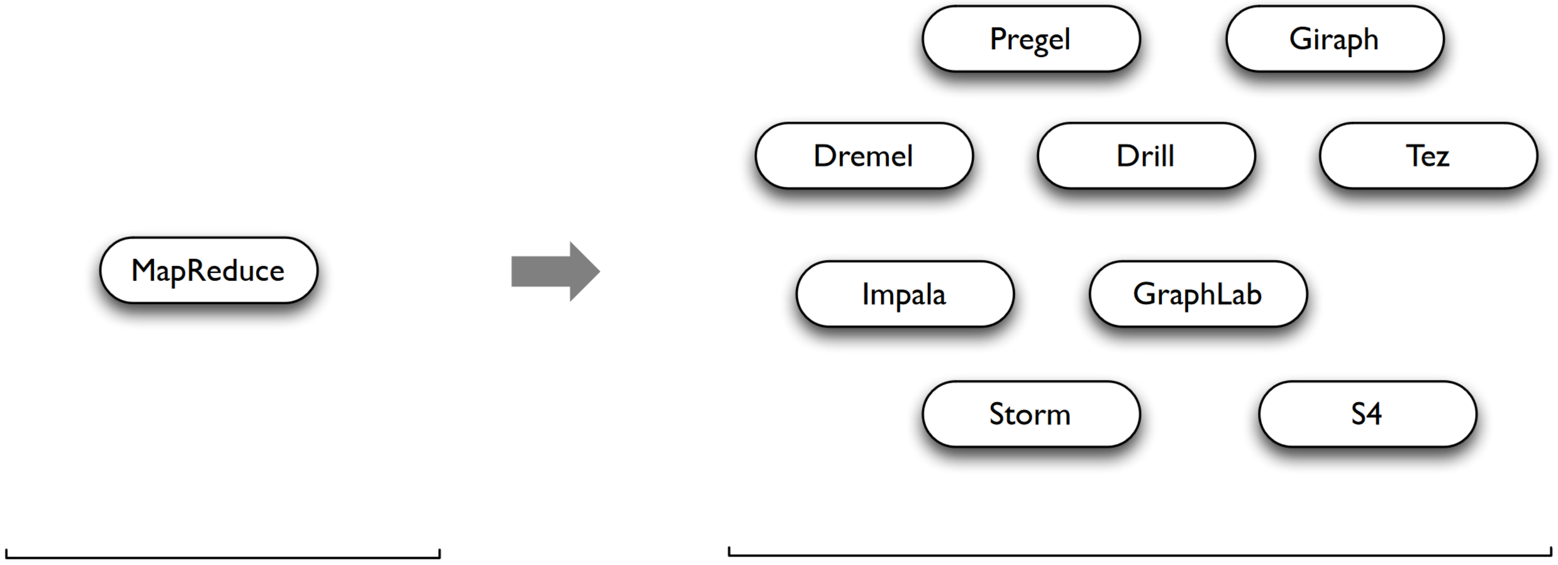


DESPITE THE SUCCESS…

…MapReduce was plagued by:

- steep programming learning curve
- performance bottlenecks
- lack of support for use cases where **batch** wasn’t good enough

…SO, SPECIALIZED ALTERNATIVES APPEARED…



MapReduce

Pregel

Giraph

Dremel

Drill

Tez

Impala

GraphLab

Storm

S4

General Batch Processing

Specialized Systems:
iterative, interactive, streaming, graph, etc.

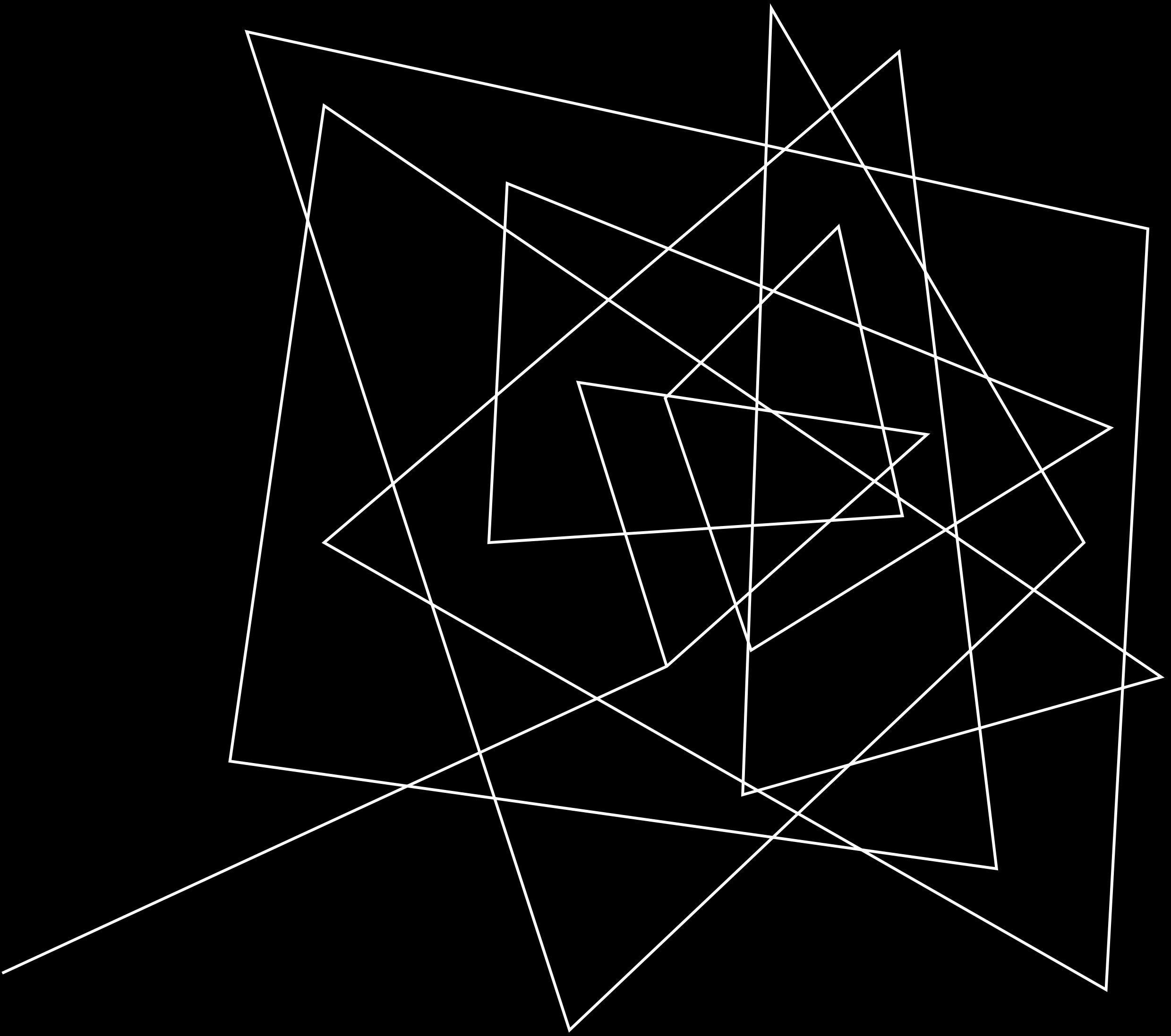
…INSTEAD, SPARK BUILT UPON MAP/REDUCE…

Spark's goal is to generalize MapReduce by introducing:

- fast data sharing
- general DAGs

And manages to achieve its design goals:

- handles batch, interactive, and real-time within a single framework
- native integration with Java, Python, Scala
- programming at a higher level of abstraction
- more general: map/reduce is just one set of supported constructs



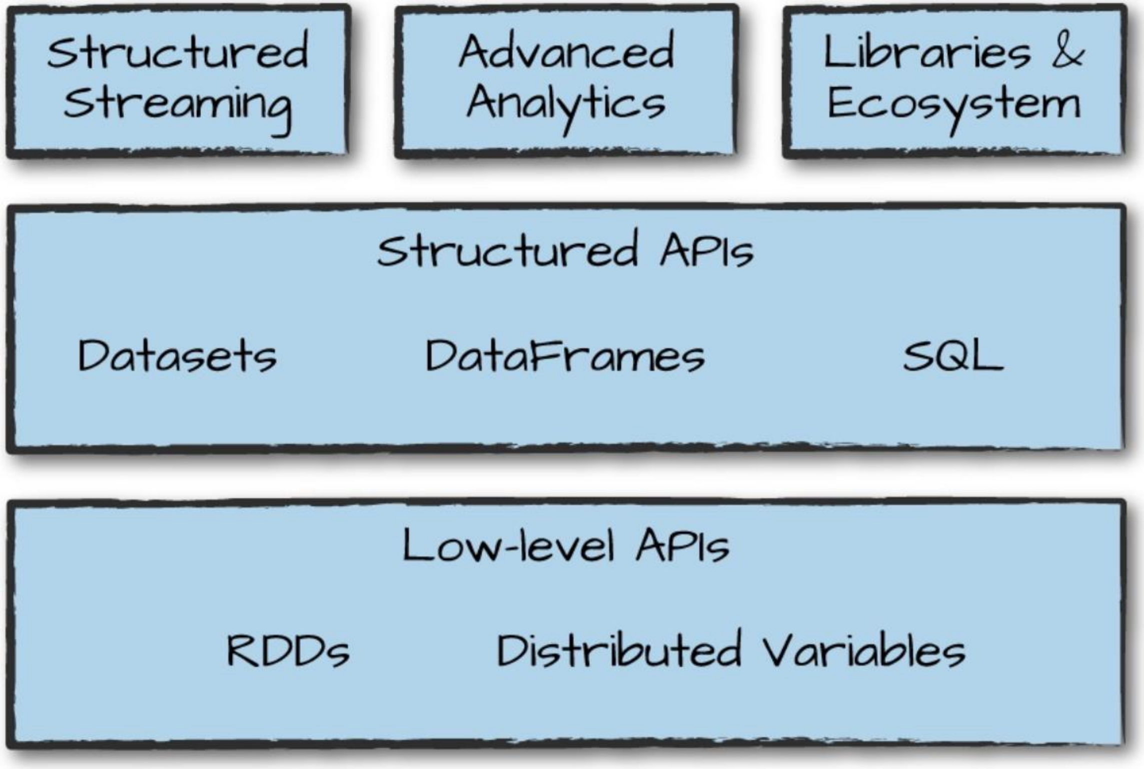
INTRODUCTION TO SPARK

WHAT IS APACHE SPARK?

Apache Spark is a unified computing engine and a set of libraries for parallel data processing on computer clusters. Spark supports multiple widely used programming languages (Python, Java, Scala, and R), includes libraries for diverse tasks ranging from SQL to streaming and machine learning, and runs anywhere from a laptop to a cluster of thousands of servers. This makes it an easy system to start with and scale-up to big data processing or incredibly large scale.

FEATURES OF APACHE SPARK

- offers a unified platform for writing big data applications
- highly abstracted programming
- supports multiple different APIs and libraries



HADOOP VS SPARK

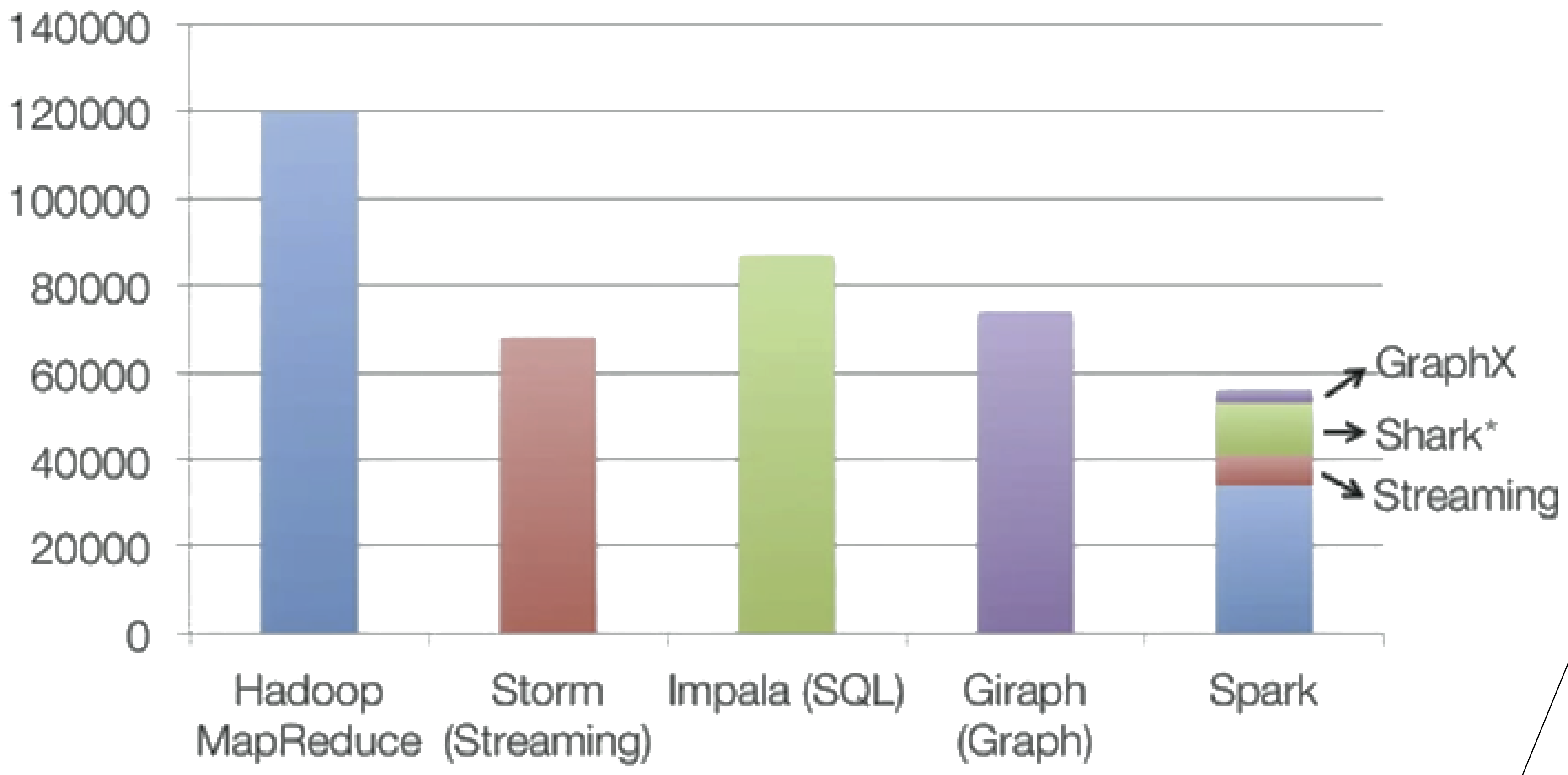
Hadoop

- Processing data using MapReduce in Hadoop is slow (as it uses a lot of **disk I/O**)
- Strictly restricted to batch processing
- More lines of (java) code

Spark

- Spark processes data 100x faster than Hadoop MapReduce (**in-memory** computing)
- Supports both batch and real-time proccessing
- Fewer lines of (scala) code

CODE SIZE COMPARISON (2013)

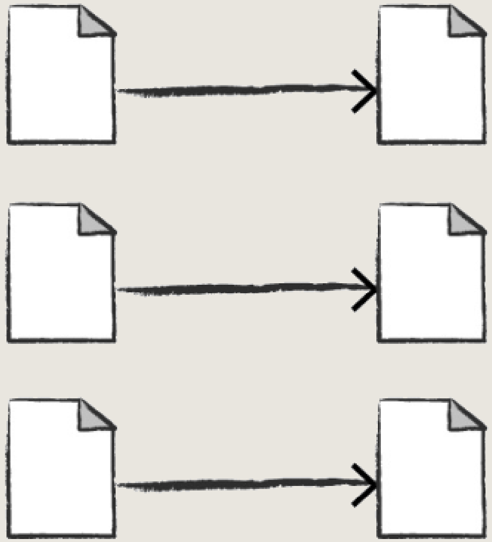


non-test, non-example source lines

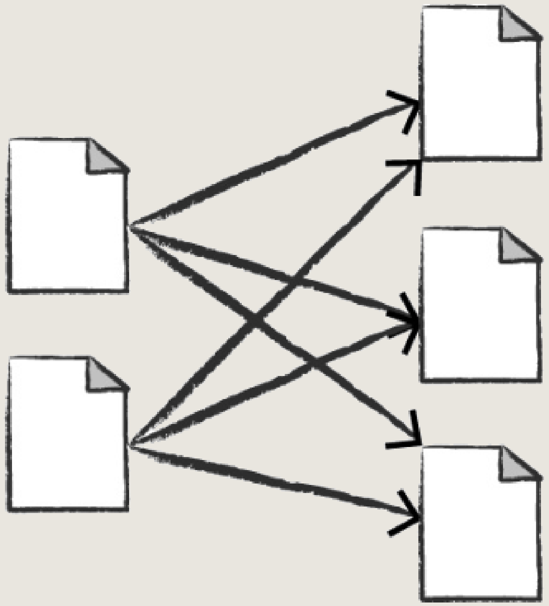
* also calls into Hive

CORE CONCEPT I: DATA TRANSFORMATION TYPES

Narrow (Map)



Wide (Reduce)



CORE CONCEPT II: RESILIENT DISTRIBUTED DATASETS

- RDDs were the primary API in Spark 1.X
- an RDD represents an immutable, partitioned collection of records that can be operated on in parallel
- RDDs track the series of **data transformations** used to build them to recompute lost data
- everything is RDD

CORE CONCEPT III: LAZY EVALUATION

Lazy evaluation means that Spark will wait until the very last moment to execute the graph of computation instructions.

SPARK COMPONENTS

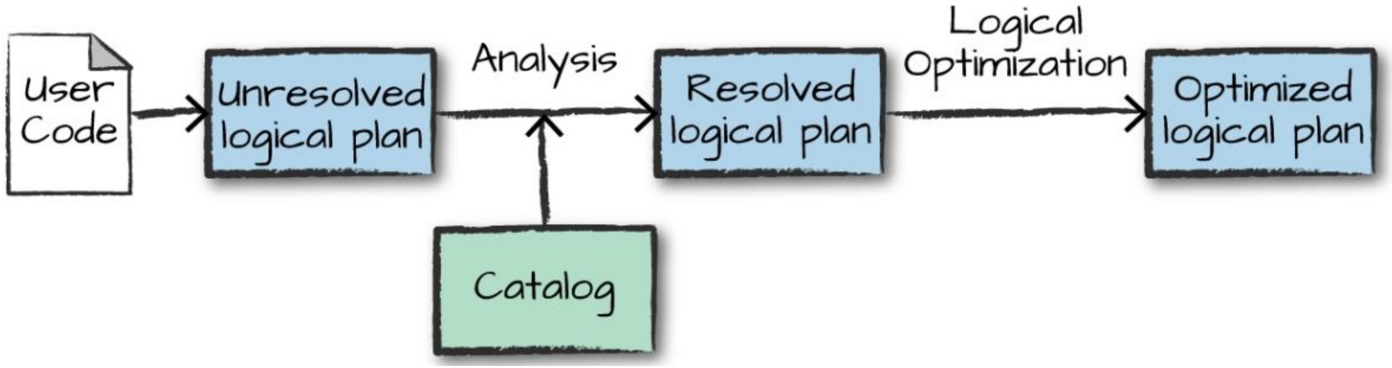
SPARK COMPONENTS (1/3)

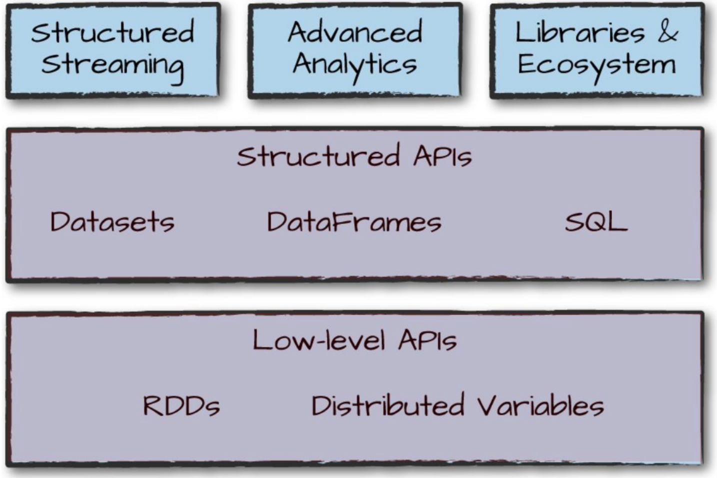
Spark Core:

- Memory management (**RDDs**)
- Fault recovery
- Scheduling, distributing & monitoring jobs
- Interacting with storage systems

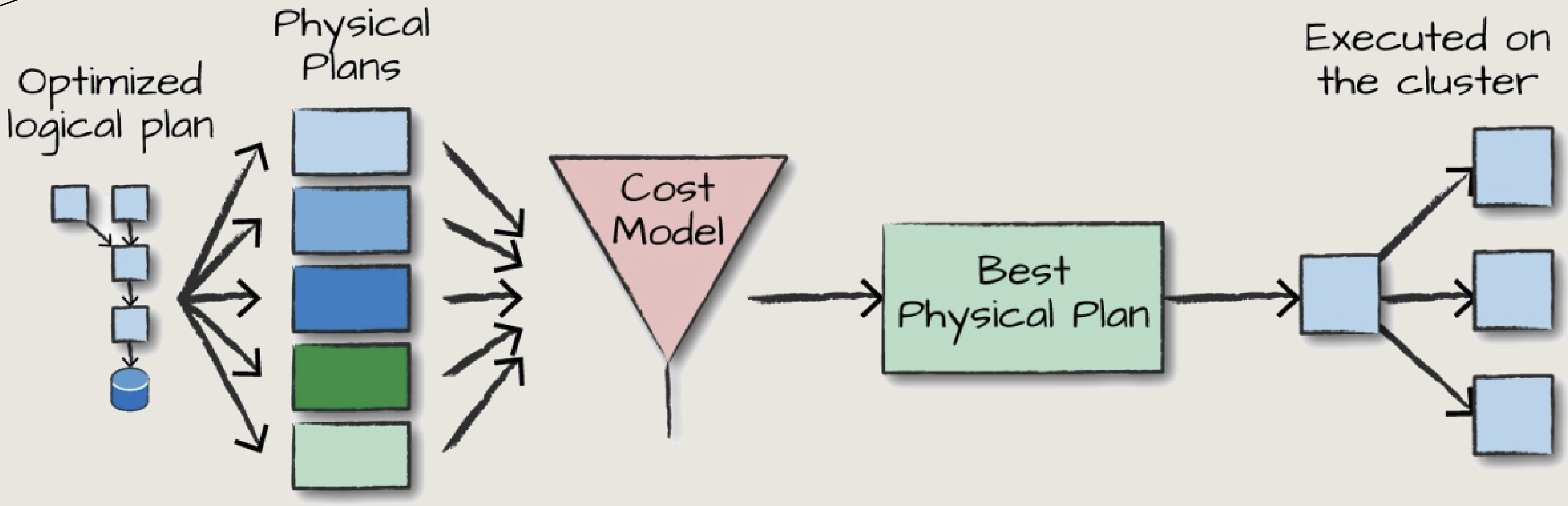
Spark SQL:

- Used for structured and semi-structured data processing
- ***Catalyst Optimizer*** 🧠 💪
- **DataFrames + SQL**





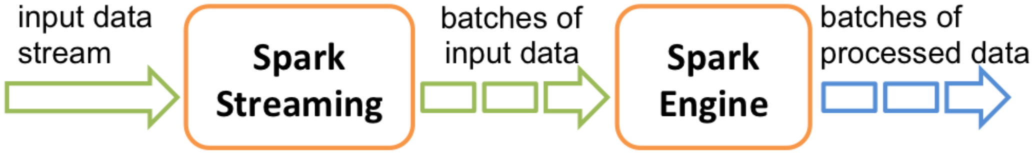
SPARK SQL: THE CATALYST OPTIMIZER



SPARK COMPONENTS (2/3)

Spark Streaming:

- Secure, reliable and fast processing of live data streams



input data stream

Spark Streaming

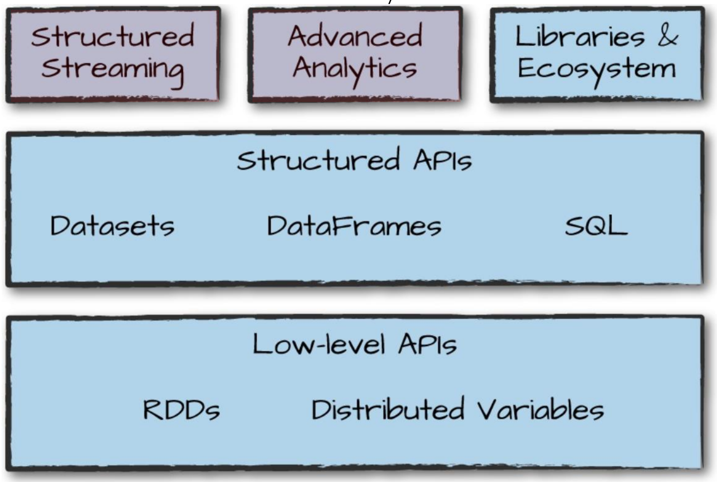
batches of input data

Spark Engine

batches of processed data

Spark MLlib:

- Contains libraries that implementat various machine learning algorithms (clustering, classification, collaborative filtering, etc.)



Structured Streaming

Advanced Analytics

Libraries & Ecosystem

Structured APIs

Datasets DataFrames SQL

Low-level APIs

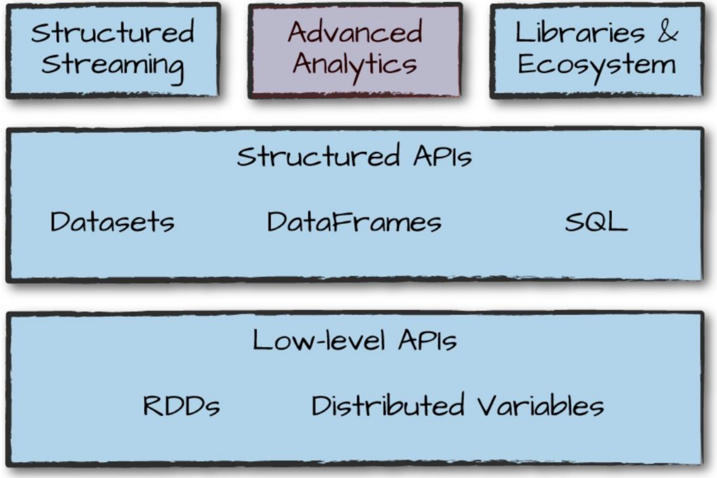
RDDs Distributed Variables

GraphX Spark MLlib Spark Streaming Spark SQL

SPARK COMPONENTS (3/3)

Spark GraphX:

- Graph computation engine and data store
- Uniform tool for ETL
- Exploratory data analysis
- Interactive graph computations



Structured Streaming

Advanced Analytics

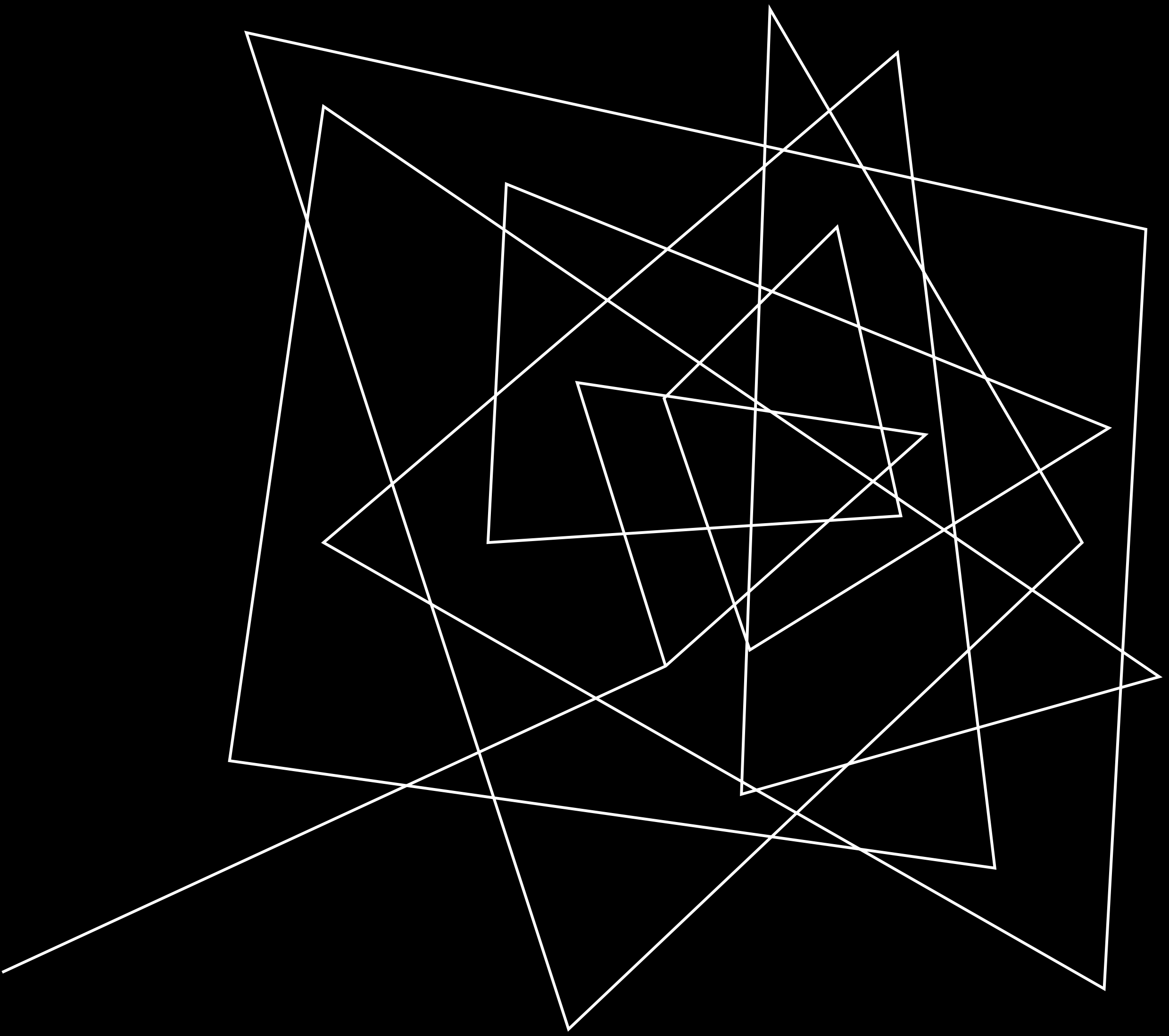
Libraries & Ecosystem

Structured APIs

Datasets DataFrames SQL

Low-level APIs

RDDs Distributed Variables



SPARK ARCHITECTURE

EVERY SPARK APPLICATION'S HIGH-LEVEL COMPONENTS…

The Spark driver:

- the controller process of the execution of a Spark Application
- maintains all of the state of the Spark cluster (the state and tasks of the executors)
- interfaces with the cluster manager to get physical resources and launch executors

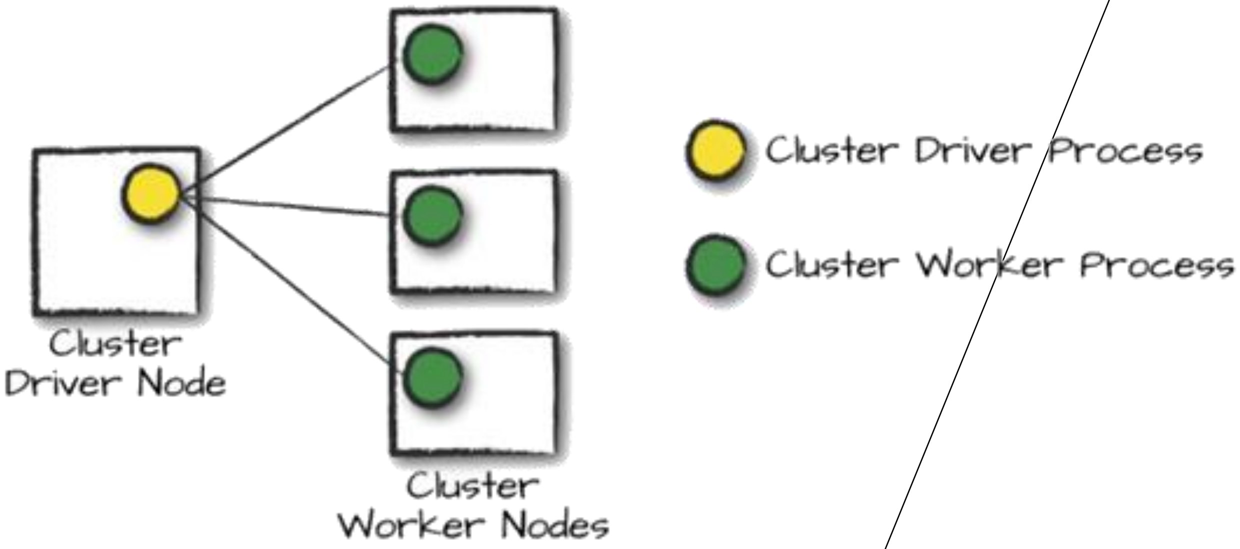
The Spark executors:

- the processes that perform the tasks assigned by the Spark driver
- they run the tasks assigned by the driver and report back their state (success or failure) and results

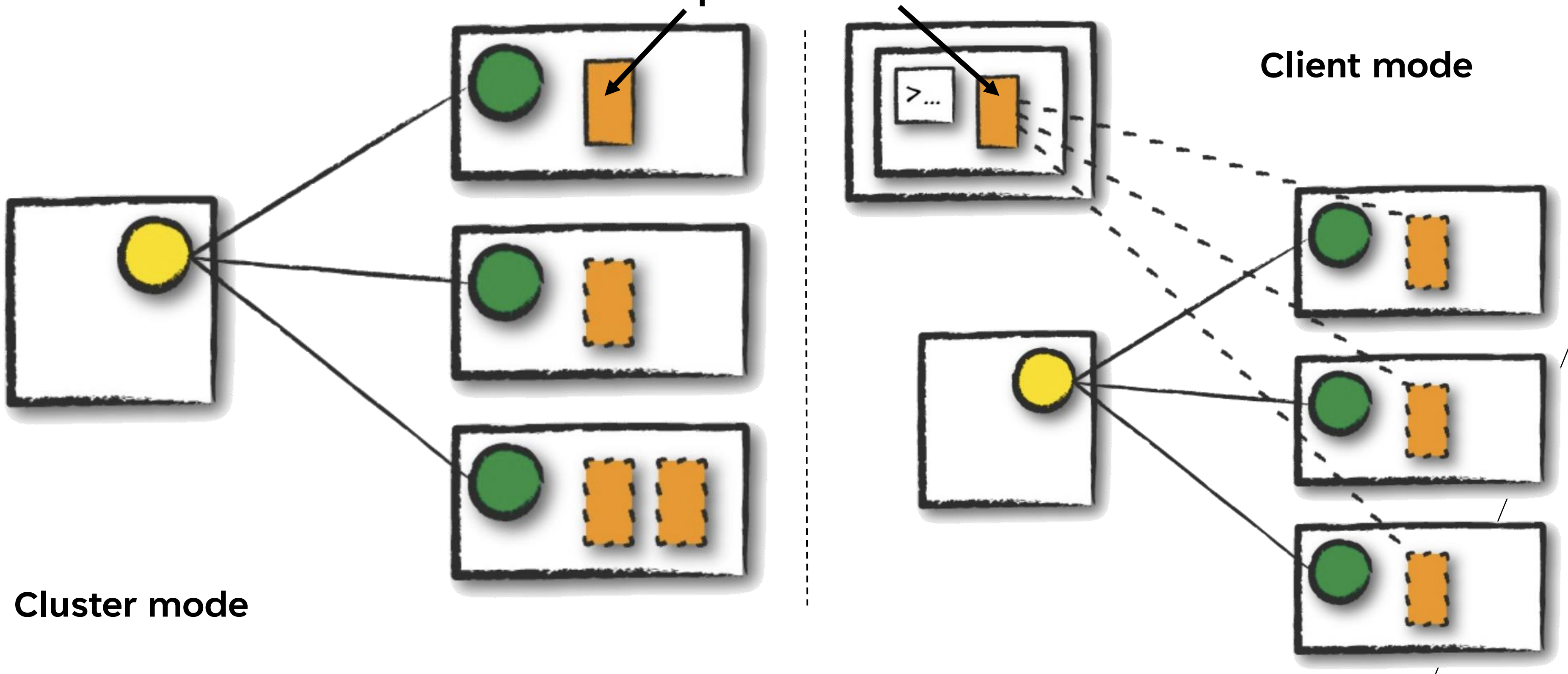
EVERY SPARK APPLICATION'S HIGH-LEVEL COMPONENTS…

The cluster manager:

- a framework responsible for maintaining a cluster of machines that will run Spark Application(s)
- a cluster manager will have its own "*driver*" and "worker" abstractions – *<u>tied to physical machines</u>* rather than processes (as they are in Spark)



CLUSTER VS CLIENT EXECUTION MODES

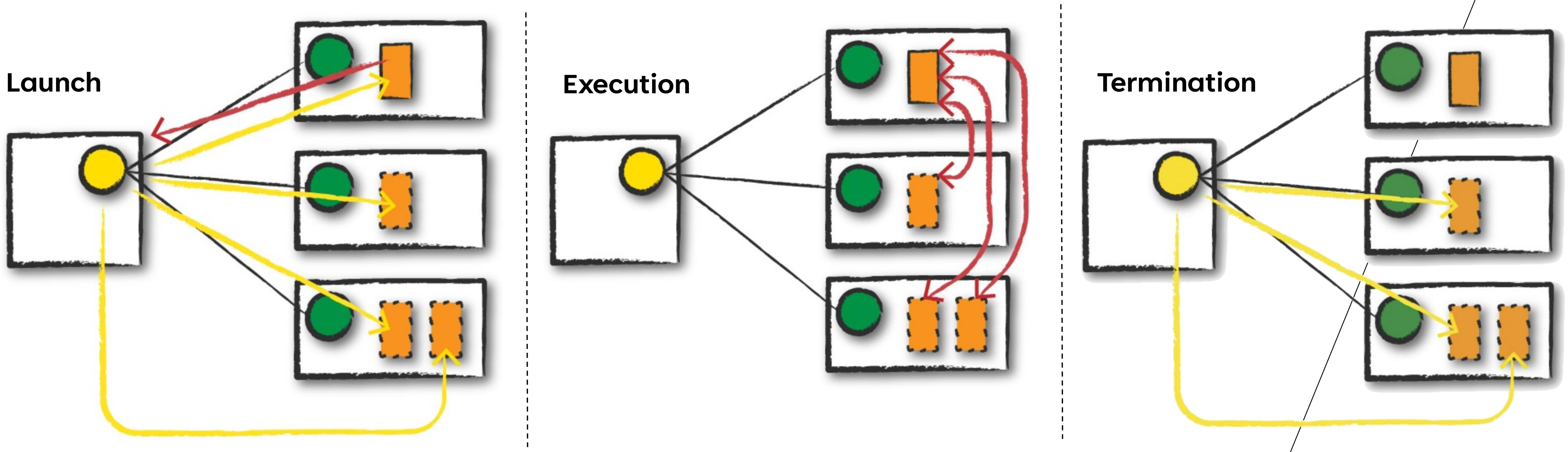


Spark Driver

Client mode

Cluster mode

LAUNCH, EXECUTION, TERMINATION



Launch

Execution

Termination

SPARK CLUSTER MANAGERS

By default, applications submitted to the standalone mode cluster will run in FIFO order and each application will try to use all available nodes – 1 executor per node.

Apache Mesos is an open-source project used to manage computing clusters supported by the Hadoop ecosystem.

Apache YARN is the cluster resource manager of Hadoop since version 2.0. Spark can use YARN as its cluster manager.

Kubernetes is an open-source system for automatic deployment, scaling and management of containerized applications.

HADOOP ECOSYSTEM

HADOOP DISTRIBUTED FILE SYSTEM

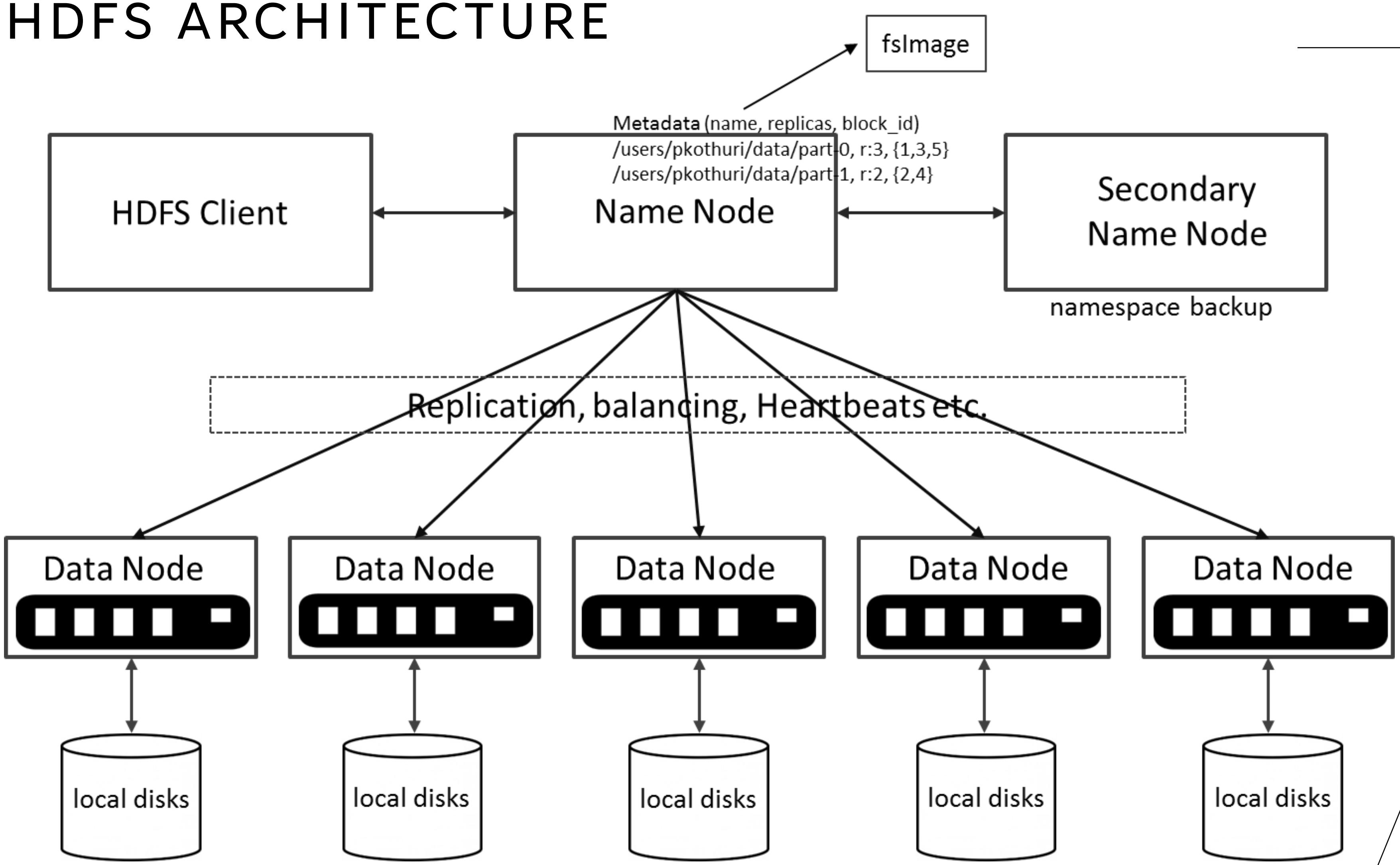
What is HDFS

- HDFS is a distributed file system that is fault tolerant, scalable and extremely easy to expand.
- HDFS is the primary distributed storage for Hadoop applications.
- HDFS provides interfaces for applications to move themselves closer to data.

HDFS Components

- **NameNode**: the heart of an HDFS filesystem - maintains and manages file system metadata. e.g; what blocks make up a file, and on which datanodes those blocks are stored.
- **DataNode**: where HDFS stores the actual data, there are usually quite a few of these.

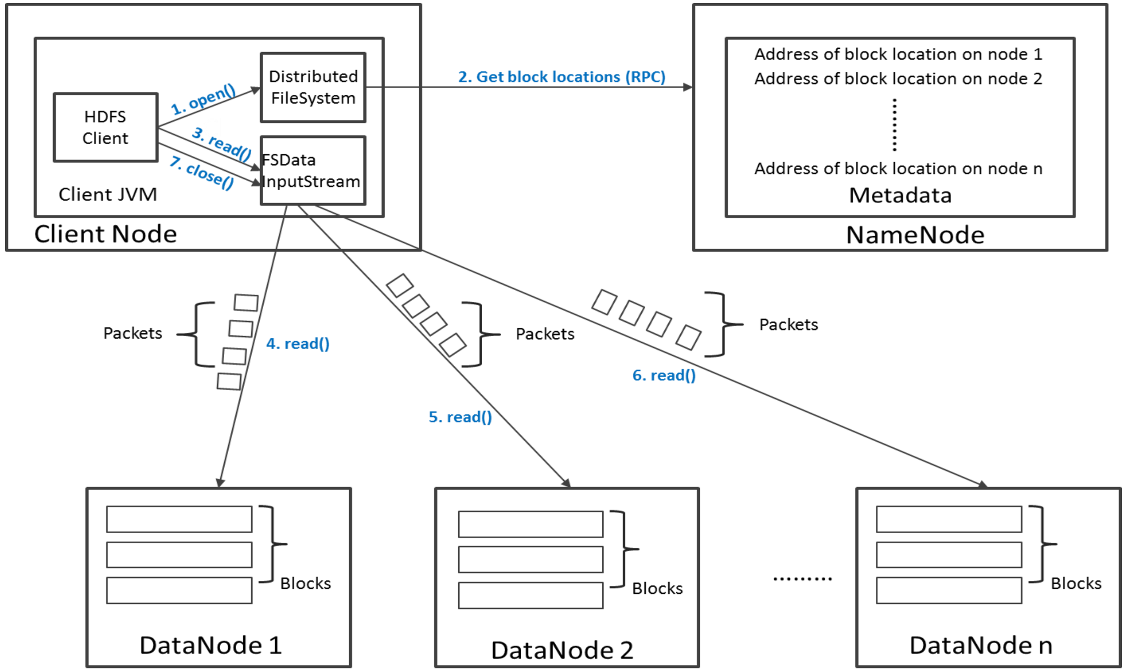
HDFS ARCHITECTURE



HDFS – DATA ORGANIZATION

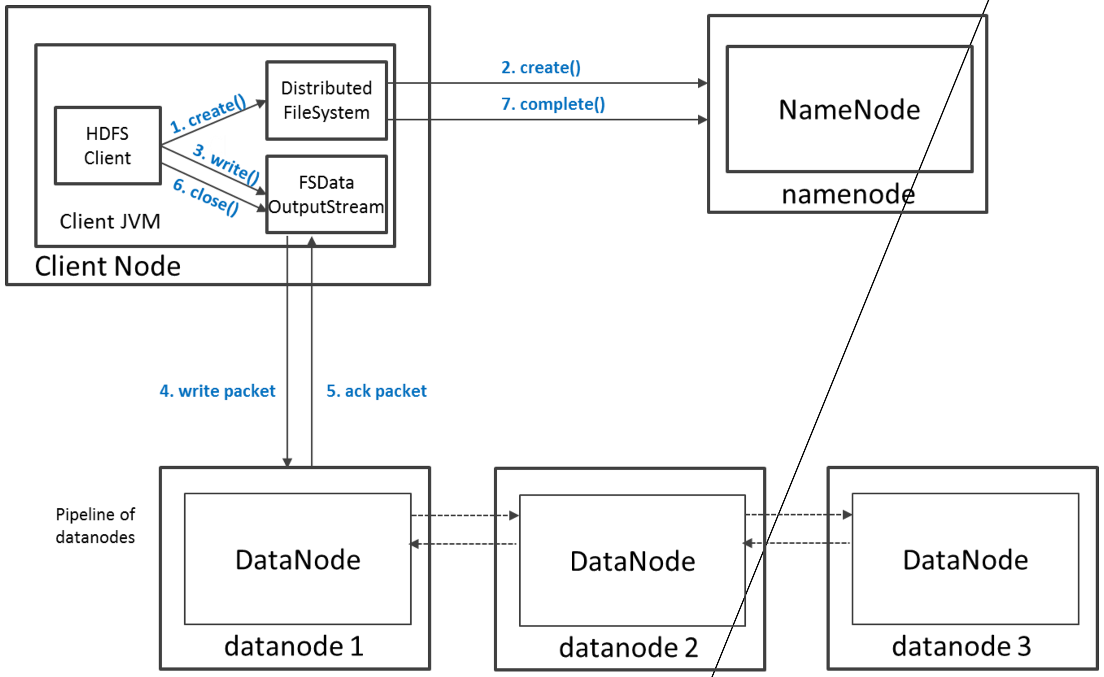
- Each file written into HDFS is split into data blocks.
- Each block is stored on one or more nodes (replication factor)
- Each copy of the block is called replica.
- Block placement policy:
    - First replica is placed on the local node.
    - Second replica is placed in a different rack.
    - Third replica is placed in the same rack as the second replica.

READ/WRITE OPERATIONS IN HDFS



Read Operation

Write Operation



YET ANOTHER RESOURCE NEGOTIATOR

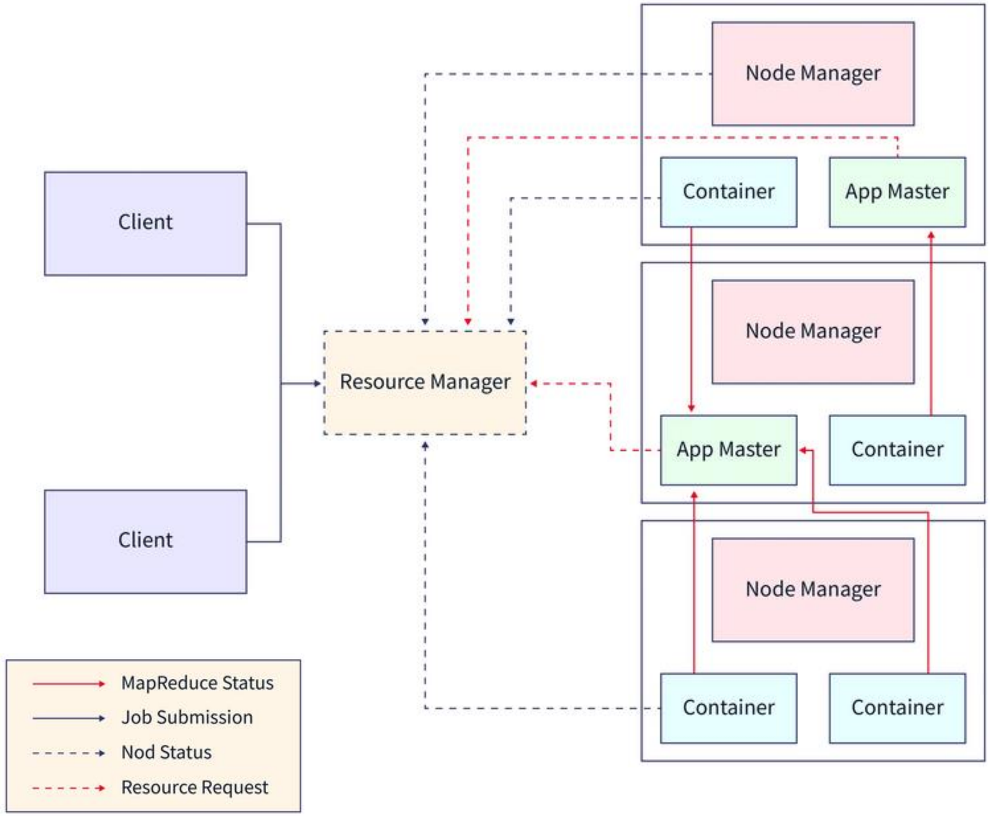
What is YARN

- YARN is a core component of the Hadoop ecosystem responsible for managing resources and scheduling jobs across the cluster.
- Yarn acts as the **operating system** for Hadoop, managing and allocating computational resources (CPU, memory) to various applications.

HDFS Components

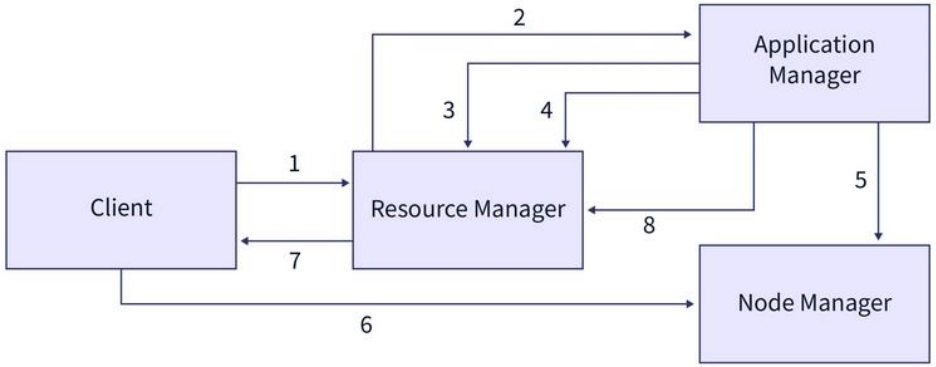
- **ResourceManager (RM):** The global master that manages and allocates resources across all applications in the cluster.
- **NodeManager (NM):** Runs on each node in the cluster, managing individual resources like CPU, memory, and disk for that node.
- **ApplicationMaster (AM):** Each application has its own AM that negotiates resources with the RM and works with NMs to execute and monitor tasks.

HADOOP YARN ARCHITECTURE & APPLICATION WORKFLOW



YARN Component Architecture

YARN Application Workflow



WORKSHOP GOALS & TAKEAWAYS

Session 1

- Introduction to Spark and Hadoop ecosystem
  - Spark, YARN, HDFS etc.
- Configuration of a Spark-over-YARN cluster
- Execution of examples using different interfaces
  - RDDs, DataFrame, SQL
- Experimentation with command-line and interactive shell execution

Session 2

- Introduction of a real-world use case.
- Implementation of solutions to complex queries.

THANK YOU

Nikolaos Chalvantzis

Computer Systems Lab, ECE School, NTUA, Greece

nchalv@cslab.ece.ntua.gr

PRESENTATION SOURCES

- https://www.oreilly.com/library/view/spark-the-definitive/9781491912201/
- https://www.scaler.com/topics/hadoop/yarn-in-hadoop/
- https://stanford.edu/~rezab/sparkclass/slides/itas_workshop.pdf