What is Apache spark?

Agenda

- **What is Apache Spark?**
- **Brief History**
- **Spark Stack Components**
- **Core Spark Concepts**
  - Spark context
  - Executors

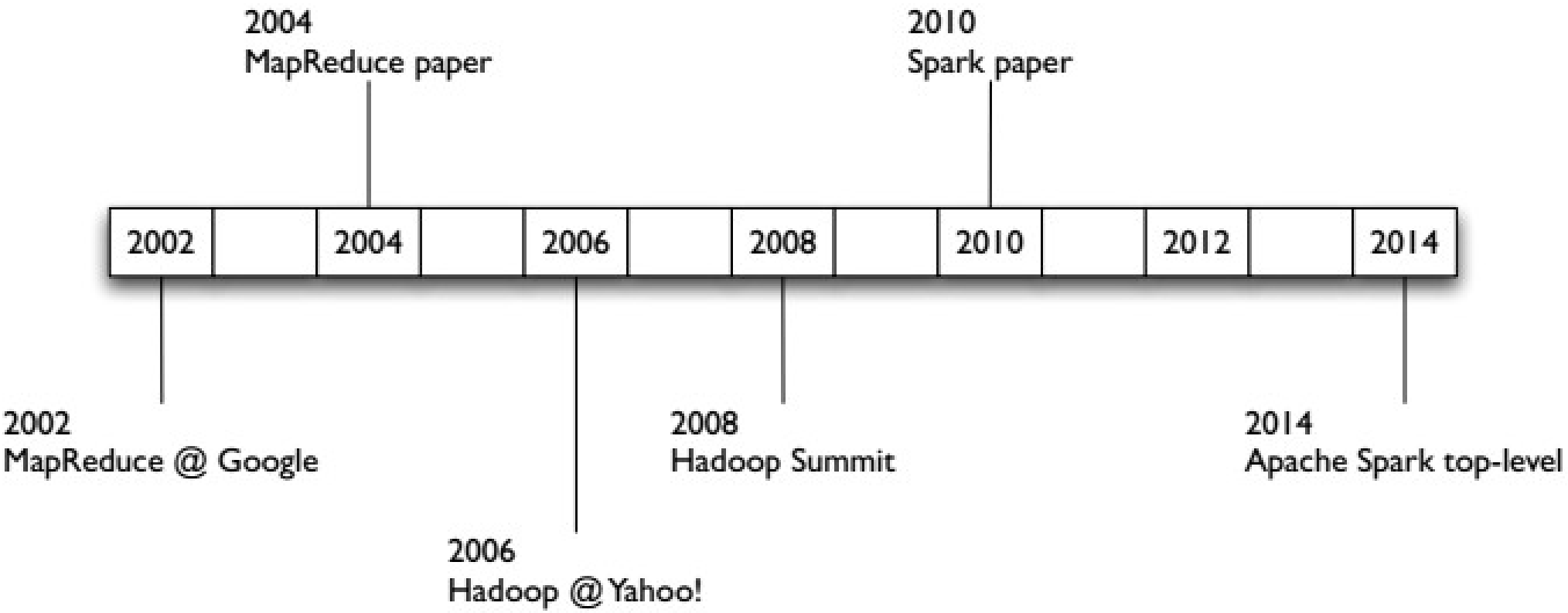
What is Spark?

- **a cluster computing platform designed to be fast and general purpose.**

What is Spark?

- A “computational engine” that is responsible for
  - scheduling,
  - distributing and
  - monitoring
- applications consisting of many computational tasks across a cluster.

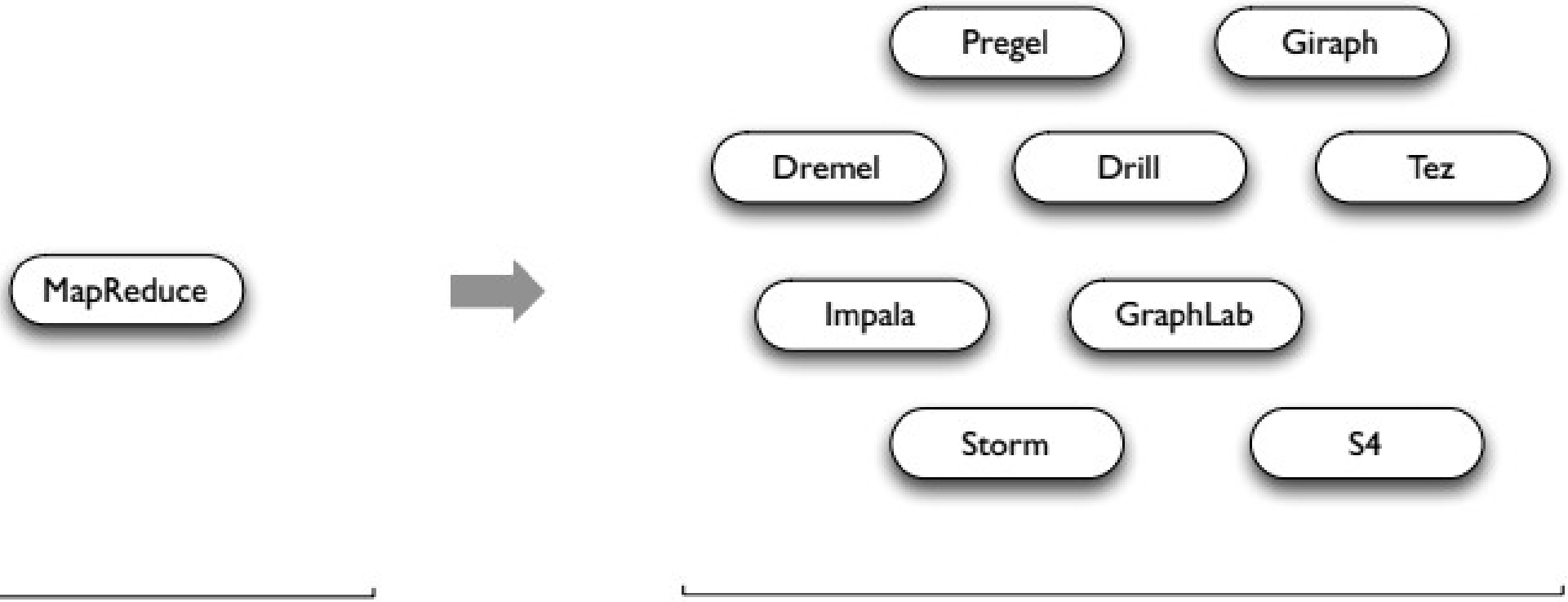
Brief History



Problems with Map Reduce

- MapReduce use cases showed two major limitations:
  - difficultly of programming directly in MR
  - performance bottlenecks, or
  - batch not fitting all the use cases
- MR doesn’t compose well for large applications which led to building workarounds.

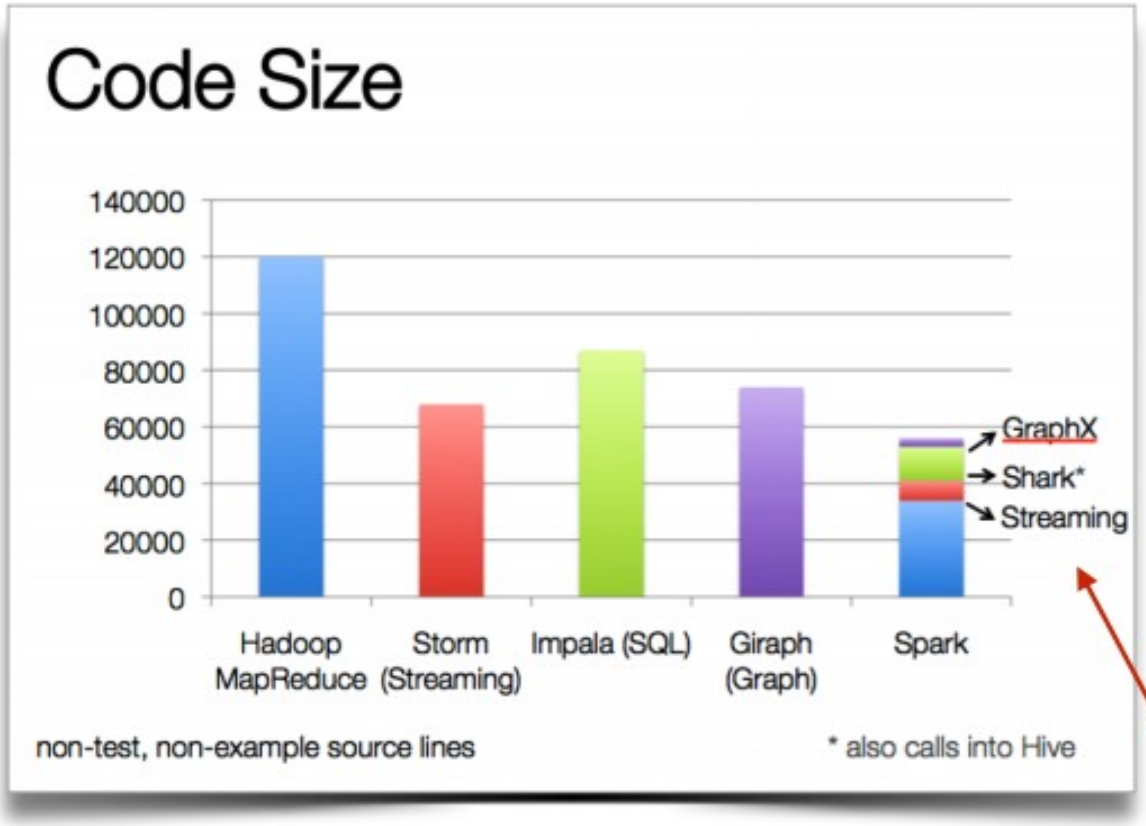
Problems with Map Reduce



General Batch Processing

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

New Apps within same Engine



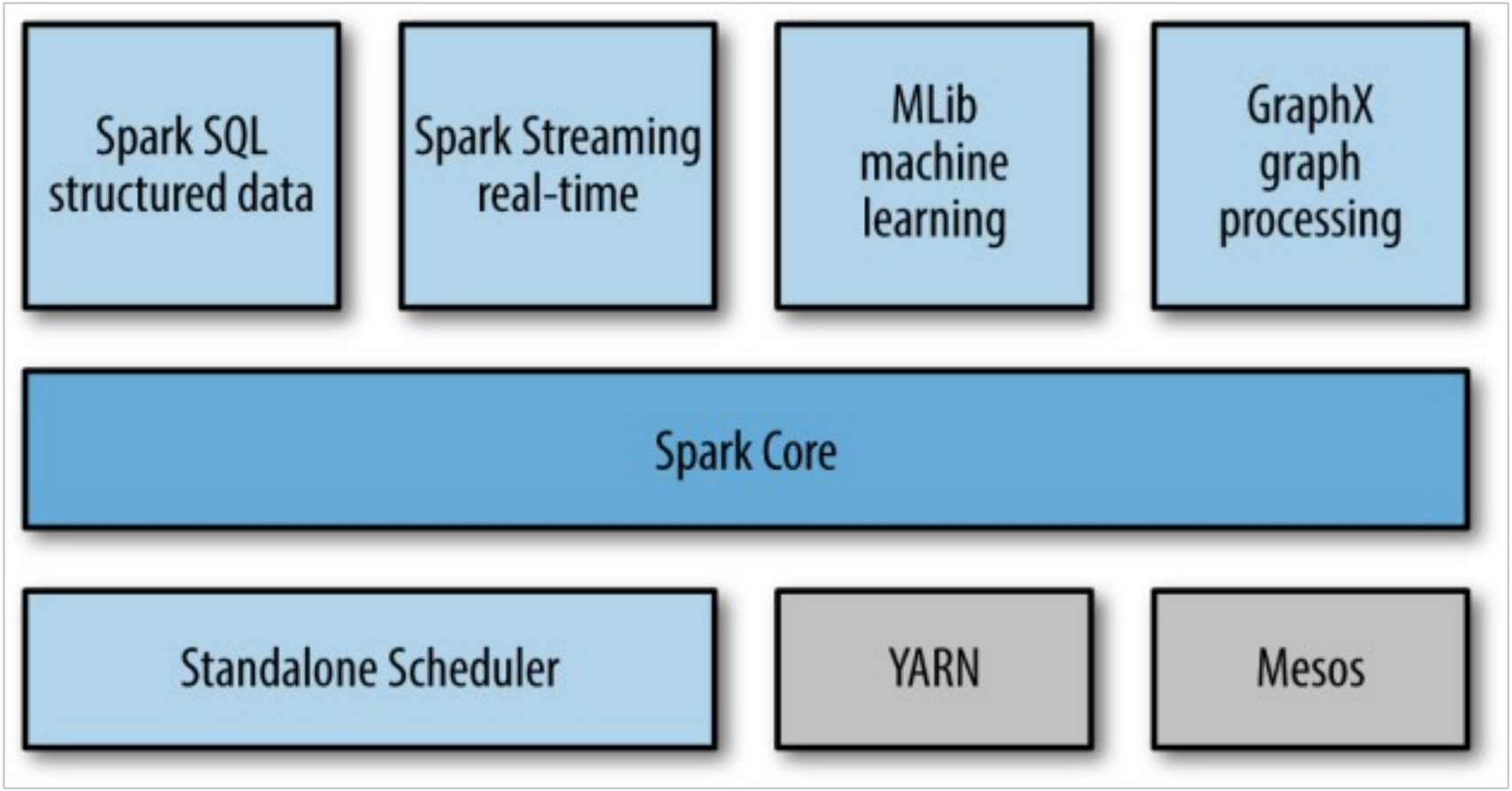
The State of Spark, and Where We're Going Next
Matei Zaharia
Spark Summit (2013)
youtu.be/nU6vO2EJAb4

used as libs, instead of specialized systems

Spark:

- 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

Spark Stack


Spark SQL
structured data
Spark Streaming
real-time
MLib
machine
learning
GraphX
graph
processing
Spark Core
Standalone Scheduler
YARN
Mesos

Spark Stack

- Spark Core
- Spark SQL
- Spark Streaming
- GraphX
- MLib

Spark Core

- **Basic functionality like**
  - **components for task scheduling,**
  - **memory management,**
  - **fault recovery,**
  - **interacting with storage systems.**
- **It includes API that defines resilient distributed data(RDDs), which are Spark's main programming abstraction.**
- **RDDs represent a collection of items distributed across many compute nodes that can be manipulated in parallel.**

Spark SQL

- For working with structured data.
- Querying data via SQL as well as Hive Query Language (HQL).
- Supports many sources of data, including Hive tables, Parquet, and JSON.
- Can be embedded into programs for RDDs in Python, Java, and Scala.

Spark Streaming

- For processing of live streams of data.
- Examples:
  - logfiles generated by production web servers
  - queues of messages containing status updates posted by users of a web service.

GraphX

- GraphX is a library for
  - manipulating graphs (e.g., a social network's friend graph) and
  - performing graph-parallel computations.
- provides various operators for manipulating graphs (e.g., subgraph and mapVertices) and a library of common graph algorithms (e.g., PageRank and triangle counting)

MLlib

- A library containing common machine learning (ML).
- provides multiple types of machine learning algorithms, including
  - classification,
  - regression,
  - clustering, and
  - collaborative filtering,
- as well as supporting functionality such as model evaluation and data import.

Does Spark need Hadoop?

- **Spark does not require Hadoop; it simply has support for storage systems implementing the Hadoop APIs.**

Core Spark Concepts

Driver Program

- Every Spark application consists of a driver program.
- It
  - launches various parallel operations on a cluster,
  - contains your application's main function and
  - defines distributed data sets on the cluster,
  - applies operations to them.
- In shell programs, Spark shell itself is driver program.

SparkContext

- Driver programs access Spark through a SparkContext object.
- Which represents a connection to a computing cluster.
- which tells Spark how to access a cluster
- In Spark shell, a special interpreter-aware SparkContext is already created “sc”.
- Making your own SparkContext will not work.
- Can be used to build RDD.

Master Parameter

- The master parameter for a SparkContext determines which cluster to use.
- $ MASTER=local[4] ./spark-shell

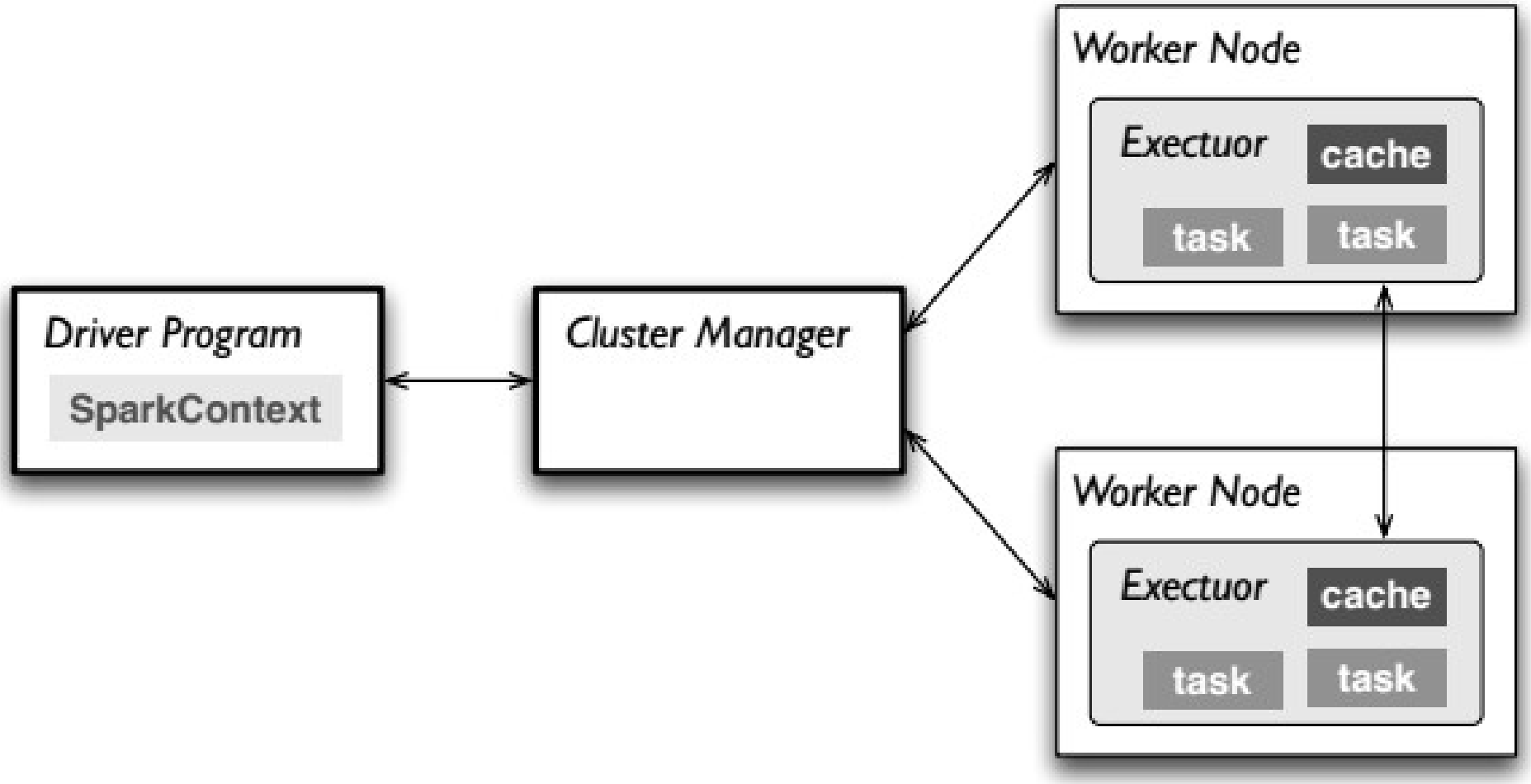
Master Parameter

master	*description*
`local`	run Spark locally with one worker thread (no parallelism)
`local[K]`	run Spark locally with K worker threads (ideally set to # cores)
`spark://HOST:PORT`	connect to a Spark standalone cluster; PORT depends on config (7077 by default)
`mesos://HOST:PORT`	connect to a Mesos cluster; PORT depends on config (5050 by default)

How does it work?

1. connects to a cluster manager which allocate resources across applications
2. acquires executors on cluster nodes worker processes to run computations and store data
3. sends app code to the executors
4. sends tasks for the executors to run

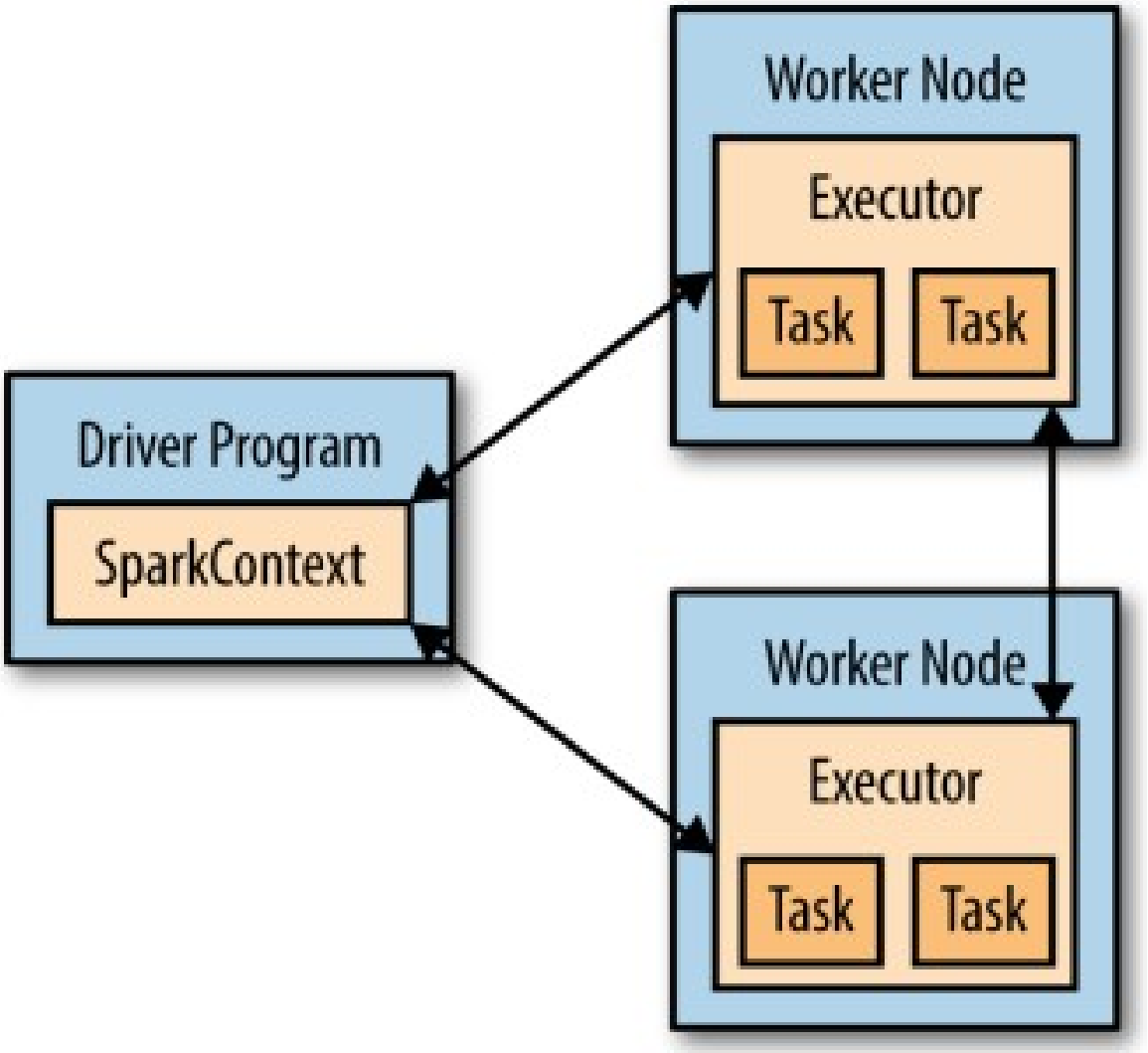
Cluster Execution



Executors

- Various operations can be performed on RDDs.
- To run these operations, driver programs typically manage a number of nodes called executors.
- Different executor nodes work on different part of file to be processed.

Components for distributed execution



RDDs in short

- **Resilient Distributed Datasets (RDD) are the primary abstraction in Spark – a fault-tolerant collection of elements that can be operated on in parallel**

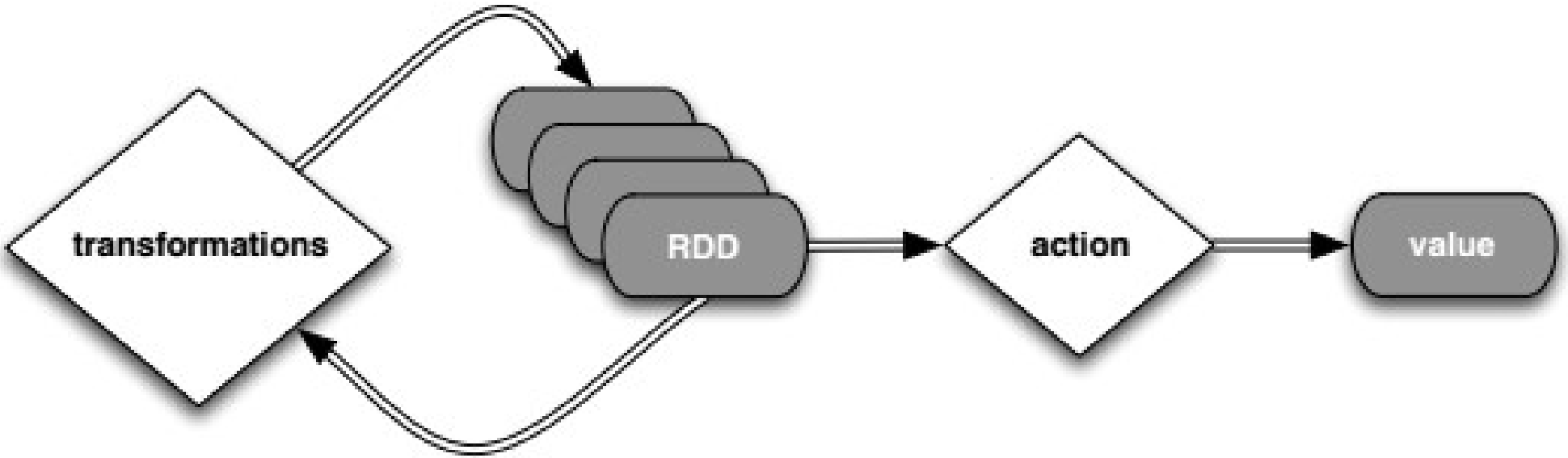
Types:

- There are currently two types:
  - parallelized collections –
    - take an existing Scala collection and run functions on it in parallel
  - Hadoop datasets –
    - run functions on each record of a file in Hadoop distributed file system or any other storage system supported by Hadoop

Operations on RDDs

- **Two types of operations on RDDs:**
  - 1) Transformations
  - 2) Actions
- **transformations are lazy (not computed immediately)**
- **the transformed RDD gets recomputedwhen an action is run on it (default)**

RDD operations



Persistence- caching data

- Spark can persist (or cache) a dataset in memory across operations.
- Each node stores in memory any slices of it that it computes and reuses them in other actions on that dataset – often making future actions more than 10x faster.
- The cache is fault-tolerant: if any partition of an RDD is lost, it will automatically be recomputed using the transformations that originally created it.

Persistance

transformation	*description*
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER	Store RDD as serialized Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
MEMORY_AND_DISK_SER	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc	Same as the levels above, but replicate each partition on two cluster nodes.

Resources

- **Learning Spark: Lightning fast Data Analytics**
  - Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia (Author)
  - O'Reilly Media; 1st Edition.